```
**********************************************************
   88985 Wed Mar 30 07:00:02 2011
new/src/cpu/sparc/vm/cppInterpreter_sparc.cpp
**********************************************************
______unchanged_portion_omitted_

1014 void CppInterpreterGenerator::generate_compute_interpreter_state(const Register
1015                                                                  const Register pre
1016                                                                  bool native) {

1018   // On entry
1019   // G5_method - caller's method
1020   // Gargs - points to initial parameters (i.e. locals[0])
1021   // G2_thread - valid? (C1 only??)
1022   // "prev_state" - contains any previous frame manager state which we must save
1023   //
1024   // On return
1025   // "state" is a pointer to the newly allocated  state object. We must allocate
1026   // a new interpretState object and the method expression stack.

1028   assert_different_registers(state, prev_state);
1029   assert_different_registers(prev_state, G3_scratch);
1030   const Register Gtmp = G3_scratch;
1031   const Address constants         (G5_method, 0, in_bytes(methodOopDesc::constan
1032   const Address access_flags      (G5_method, 0, in_bytes(methodOopDesc::access_
1033   const Address size_of_parameters(G5_method, 0, in_bytes(methodOopDesc::size_of
1034   const Address max_stack         (G5_method, 0, in_bytes(methodOopDesc::max_sta
1035   const Address size_of_locals    (G5_method, 0, in_bytes(methodOopDesc::size_of

1037   // slop factor is two extra slots on the expression stack so that
1038   // we always have room to store a result when returning from a call without pa
1039   // that returns a result.

1041   const int slop_factor = 2*wordSize;

1043   const int fixed_size = ((sizeof(BytecodeInterpreter) + slop_factor) >> LogByte
1044                           //6815692//methodOopDesc::extra_stack_words() +  // ext
1045                           frame::memory_parameter_word_sp_offset +  // register s
1046                           (native ?  frame::interpreter_frame_extra_outgoing_argu

1048   // XXX G5_method valid

1050   // Now compute new frame size

1052   if (native) {
1053     __ lduh( size_of_parameters, Gtmp );
1054     __ calc_mem_param_words(Gtmp, Gtmp);     // space for native call parameters
1055   } else {
1056     __ lduh(max_stack, Gtmp);                // Full size expression stack
1057   }
1058   __ add(Gtmp, fixed_size, Gtmp);           // plus the fixed portion

1060   __ neg(Gtmp);                              // negative space for stack/parame
1061   __ and3(Gtmp, -WordsPerLong, Gtmp);        // make multiple of 2 (SP must be 2
1062   __ sll(Gtmp, LogBytesPerWord, Gtmp);       // negative space for frame in byte

1064   // Need to do stack size check here before we fault on large frames

1066   Label stack_ok;

1068   const int max_pages = StackShadowPages > (StackRedPages+StackYellowPages) ? St
1069                                                                               (S


1072   __ ld_ptr(G2_thread, in_bytes(Thread::stack_base_offset()), O0);
1073   __ ld_ptr(G2_thread, in_bytes(Thread::stack_size_offset()), O1);
```



```
1074   // compute stack bottom
1075   __ sub(O0, O1, O0);

1077   // Avoid touching the guard pages
1078   // Also a fudge for frame size of BytecodeInterpreter::run
1079   // It varies from 1k->4k depending on build type
1080   const int fudge = 6 * K;

1082   __ set(fudge + (max_pages * os::vm_page_size()), O1);

1084   __ add(O0, O1, O0);
1085   __ sub(O0, Gtmp, O0);
1086   __ cmp(SP, O0);
1087   __ brx(Assembler::greaterUnsigned, false, Assembler::pt, stack_ok);
1088   __ delayed()->nop();

1090     // throw exception return address becomes throwing pc

1092   __ call_VM(Oexception, CAST_FROM_FN_PTR(address, InterpreterRuntime::throw_Sta
1093   __ stop("never reached");

1095   __ bind(stack_ok);

1097   __ save(SP, Gtmp, SP);                      // setup new frame and register wi

1099   // New window I7 call_stub or previous activation
1100   // O6 - register save area, BytecodeInterpreter just below it, args/locals jus
1101   //
1102   __ sub(FP, sizeof(BytecodeInterpreter), state);        // Point to new Interpr
1103   __ add(state, STACK_BIAS, state );         // Account for 64bit bias

1105 #define XXX_STATE(field_name) state, in_bytes(byte_offset_of(BytecodeInterpreter

1107   // Initialize a new Interpreter state
1108   // orig_sp - caller's original sp
1109   // G2_thread - thread
1110   // Gargs - &locals[0] (unbiased?)
1111   // G5_method - method
1112   // SP (biased) - accounts for full size java stack, BytecodeInterpreter object


1115   __ set(0xdead0004, O1);


1118   __ st_ptr(Gargs, XXX_STATE(_locals));
1119   __ st_ptr(G0, XXX_STATE(_oop_temp));

1121   __ st_ptr(state, XXX_STATE(_self_link));                // point to self
1122   __ st_ptr(prev_state->after_save(), XXX_STATE(_prev_link)); // Chain interpret
1123   __ st_ptr(G2_thread, XXX_STATE(_thread));               // Store javathread

1125   if (native) {
1126     __ st_ptr(G0, XXX_STATE(_bcp));
1127   } else {
1128     __ ld_ptr(G5_method, in_bytes(methodOopDesc::const_offset()), O2); // get co
1129     __ add(O2, in_bytes(constMethodOopDesc::codes_offset()), O2);        // get
1130     __ st_ptr(O2, XXX_STATE(_bcp));
1131   }

1133   __ st_ptr(G0, XXX_STATE(_mdx));
1134   __ st_ptr(G5_method, XXX_STATE(_method));

1136   __ set((int) BytecodeInterpreter::method_entry, O1);
1137   __ st(O1, XXX_STATE(_msg));

1139   __ ld_ptr(constants, O3);
```

```
1140   __ ld_ptr(O3, constantPoolOopDesc::cache_offset_in_bytes(), O2);
1141   __ st_ptr(O2, XXX_STATE(_constants));

1143   __ st_ptr(G0, XXX_STATE(_result._to_call._callee));

1145   // Monitor base is just start of BytecodeInterpreter object;
1146   __ mov(state, O2);
1147   __ st_ptr(O2, XXX_STATE(_monitor_base));

1149   // Do we need a monitor for synchonized method?
1150   {
1151     __ ld(access_flags, O1);
1152     Label done;
1153     Label got_obj;
1154     __ btst(JVM_ACC_SYNCHRONIZED, O1);
1155     __ br( Assembler::zero, false, Assembler::pt, done);

1157     const int mirror_offset = klassOopDesc::klass_part_offset_in_bytes() + Klass
1158     __ delayed()->btst(JVM_ACC_STATIC, O1);
1159     __ ld_ptr(XXX_STATE(_locals), O1);
1160     __ br( Assembler::zero, true, Assembler::pt, got_obj);
1161     __ delayed()->ld_ptr(O1, 0, O1);                  // get receiver for not-st
1162     __ ld_ptr(constants, O1);
1163     __ ld_ptr( O1, constantPoolOopDesc::pool_holder_offset_in_bytes(), O1);
1164     // lock the mirror, not the klassOop
1165     __ ld_ptr( O1, mirror_offset, O1);

1167     __ bind(got_obj);

1169   #ifdef ASSERT
1170     __ tst(O1);
1171     __ breakpoint_trap(Assembler::zero);
1172   #endif // ASSERT

1174     const int entry_size           = frame::interpreter_frame_monitor_size() *
1175     __ sub(SP, entry_size, SP);                        // account for initial m
1176     __ sub(O2, entry_size, O2);                       // initial monitor
1177     __ st_ptr(O1, O2, BasicObjectLock::obj_offset_in_bytes()); // and allocate i
1178     __ bind(done);
1179   }

1181   // Remember initial frame bottom

1183   __ st_ptr(SP, XXX_STATE(_frame_bottom));

1185   __ st_ptr(O2, XXX_STATE(_stack_base));

1187   __ sub(O2, wordSize, O2);                     // prepush
1188   __ st_ptr(O2, XXX_STATE(_stack));                 // PREPUSH

1190   __ lduh(max_stack, O3);                       // Full size expression stack
1191   guarantee(!EnableInvokeDynamic, "no support yet for java.lang.invoke.MethodHan
1192   //6815692//if (EnableInvokeDynamic)
1191   guarantee(!EnableMethodHandles, "no support yet for java.lang.invoke.MethodHan
1192   //6815692//if (EnableMethodHandles)
1193   //6815692//  __ inc(O3, methodOopDesc::extra_stack_entries());
1194   __ sll(O3, LogBytesPerWord, O3);
1195   __ sub(O2, O3, O3);
1196 //  __ sub(O3, wordSize, O3);                     // so prepush doesn't look out
1197   __ st_ptr(O3, XXX_STATE(_stack_limit));

1199   if (!native) {
1200     //
1201     // Code to initialize locals
1202     //
1203     Register init_value = noreg;    // will be G0 if we must clear locals
```



```
1204     // Now zero locals
1205     if (true /* zerolocals */ || ClearInterpreterLocals) {
1206       // explicitly initialize locals
1207       init_value = G0;
1208     } else {
1209     #ifdef ASSERT
1210       // initialize locals to a garbage pattern for better debugging
1211       init_value = O3;
1212       __ set( 0x0F0F0F0F, init_value );
1213     #endif // ASSERT
1214     }
1215     if (init_value != noreg) {
1216       Label clear_loop;

1218       // NOTE: If you change the frame layout, this code will need to
1219       // be updated!
1220       __ lduh( size_of_locals, O2 );
1221       __ lduh( size_of_parameters, O1 );
1222       __ sll( O2, LogBytesPerWord, O2);
1223       __ sll( O1, LogBytesPerWord, O1 );
1224       __ ld_ptr(XXX_STATE(_locals), L2_scratch);
1225       __ sub( L2_scratch, O2, O2 );
1226       __ sub( L2_scratch, O1, O1 );

1228       __ bind( clear_loop );
1229       __ inc( O2, wordSize );

1231       __ cmp( O2, O1 );
1232       __ br( Assembler::lessEqualUnsigned, true, Assembler::pt, clear_loop );
1233       __ delayed()->st_ptr( init_value, O2, 0 );
1234     }
1235   }
1236 }
```

_____***unchanged_portion_omitted_***

****
88438 Wed Mar 30 07:00:03 2011
new/src/cpu/sparc/vm/interp_masm_sparc.cpp
****
_____***unchanged_portion_omitted_***

```
 740 void InterpreterMacroAssembler::get_cache_index_at_bcp(Register cache, Register
 741                                                       int bcp_offset, size_t in
 742   assert(bcp_offset > 0, "bcp is still pointing to start of bytecode");
 743   if (index_size == sizeof(u2)) {
 744     get_2_byte_integer_at_bcp(bcp_offset, cache, tmp, Unsigned);
 745   } else if (index_size == sizeof(u4)) {
 746     assert(EnableInvokeDynamic, "giant index used only for JSR 292");
 746     assert(EnableInvokeDynamic, "giant index used only for EnableInvokeDynamic")
 747     get_4_byte_integer_at_bcp(bcp_offset, cache, tmp);
 748     assert(constantPoolCacheOopDesc::decode_secondary_index(~123) == 123, "else
 749     xor3(tmp, -1, tmp);  // convert to plain index
 750   } else if (index_size == sizeof(u1)) {
 751     assert(EnableInvokeDynamic, "tiny index used only for JSR 292");
 751     assert(EnableMethodHandles, "tiny index used only for EnableMethodHandles");
 752     ldub(Lbcp, bcp_offset, tmp);
 753   } else {
 754     ShouldNotReachHere();
 755   }
 756 }
```

_____***unchanged_portion_omitted_***

****
17250 Wed Mar 30 07:00:04 2011
new/src/cpu/sparc/vm/interpreter_sparc.cpp
****
_____***unchanged_portion_omitted_***

```
 262 // Method handle invoker
 263 // Dispatch a method of the form java.lang.invoke.MethodHandles::invoke(...)
 264 address InterpreterGenerator::generate_method_handle_entry(void) {
 265   if (!EnableInvokeDynamic) {
 265   if (!EnableMethodHandles) {
 266     return generate_abstract_entry();
 267   }

 269   return MethodHandles::generate_method_handle_interpreter_entry(_masm);
 270 }
```

_____***unchanged_portion_omitted_***

****
124174 Wed Mar 30 07:00:05 2011
new/src/cpu/sparc/vm/templateTable_sparc.cpp
****

_____unchanged_portion_omitted_

```
 331 // Fast path for caching oop constants.
 332 // %%% We should use this to handle Class and String constants also.
 333 // %%% It will simplify the ldc/primitive path considerably.
 334 void TemplateTable::fast_aldc(bool wide) {
 335   transition(vtos, atos);

 337   if (!EnableInvokeDynamic) {
 338     // We should not encounter this bytecode if !EnableInvokeDynamic.
 337   if (!EnableMethodHandles) {
 338     // We should not encounter this bytecode if !EnableMethodHandles.
 339     // The verifier will stop it.  However, if we get past the verifier,
 340     // this will stop the thread in a reasonable way, without crashing the JVM.
 341     __ call_VM(noreg, CAST_FROM_FN_PTR(address,
 342                      InterpreterRuntime::throw_IncompatibleClassChangeError));
 343     // the call_VM checks for exception, so we should never return here.
 344     __ should_not_reach_here();
 345     return;
 346   }

 348   Register Rcache = G3_scratch;
 349   Register Rscratch = G4_scratch;

 351   resolve_cache_and_index(f1_oop, Otos_i, Rcache, Rscratch, wide ? sizeof(u2) :

 353   __ verify_oop(Otos_i);

 355   Label L_done;
 356   const Register Rcon_klass = G3_scratch;  // same as Rcache
 357   const Register Rarray_klass = G4_scratch;  // same as Rscratch
 358   __ load_klass(Otos_i, Rcon_klass);
 359   AddressLiteral array_klass_addr((address)Universe::systemObjArrayKlassObj_addr
 360   __ load_contents(array_klass_addr, Rarray_klass);
 361   __ cmp(Rarray_klass, Rcon_klass);
 362   __ brx(Assembler::notEqual, false, Assembler::pt, L_done);
 363   __ delayed()->nop();
 364   __ ld(Address(Otos_i, arrayOopDesc::length_offset_in_bytes()), Rcon_klass);
 365   __ tst(Rcon_klass);
 366   __ brx(Assembler::zero, true, Assembler::pt, L_done);
 367   __ delayed()->clr(Otos_i);    // executed only if branch is taken

 369   // Load the exception from the system-array which wraps it:
 370   __ load_heap_oop(Otos_i, arrayOopDesc::base_offset_in_bytes(T_OBJECT), Otos_i)
 371   __ throw_if_not_x(Assembler::never, Interpreter::throw_exception_entry(), G3_s

 373   __ bind(L_done);
 374 }
```

_____unchanged_portion_omitted_

****
51311 Wed Mar 30 07:00:06 2011
new/src/cpu/x86/vm/interp_masm_x86_32.cpp
****

_____***unchanged_portion_omitted_***

```
213 void InterpreterMacroAssembler::get_cache_index_at_bcp(Register reg, int bcp_off
214   assert(bcp_offset > 0, "bcp is still pointing to start of bytecode");
215   if (index_size == sizeof(u2)) {
216     load_unsigned_short(reg, Address(rsi, bcp_offset));
217   } else if (index_size == sizeof(u4)) {
218     assert(EnableInvokeDynamic, "giant index used only for JSR 292");
218     assert(EnableInvokeDynamic, "giant index used only for EnableInvokeDynamic")
219     movl(reg, Address(rsi, bcp_offset));
220     // Check if the secondary index definition is still ~x, otherwise
221     // we have to change the following assembler code to calculate the
222     // plain index.
223     assert(constantPoolCacheOopDesc::decode_secondary_index(~123) == 123, "else
224     notl(reg);  // convert to plain index
225   } else if (index_size == sizeof(u1)) {
226     assert(EnableInvokeDynamic, "tiny index used only for JSR 292");
226     assert(EnableMethodHandles, "tiny index used only for EnableMethodHandles");
227     load_unsigned_byte(reg, Address(rsi, bcp_offset));
228   } else {
229     ShouldNotReachHere();
230   }
231 }
```

_____***unchanged_portion_omitted_***

****
54043 Wed Mar 30 07:00:07 2011
new/src/cpu/x86/vm/interp_masm_x86_64.cpp
****
_____***unchanged_portion_omitted_***

```
 209 void InterpreterMacroAssembler::get_cache_index_at_bcp(Register index,
 210                                                      int bcp_offset,
 211                                                      size_t index_size) {
 212   assert(bcp_offset > 0, "bcp is still pointing to start of bytecode");
 213   if (index_size == sizeof(u2)) {
 214     load_unsigned_short(index, Address(r13, bcp_offset));
 215   } else if (index_size == sizeof(u4)) {
 216     assert(EnableInvokeDynamic, "giant index used only for JSR 292");
 216     assert(EnableInvokeDynamic, "giant index used only for EnableInvokeDynamic")
 217     movl(index, Address(r13, bcp_offset));
 218     // Check if the secondary index definition is still ~x, otherwise
 219     // we have to change the following assembler code to calculate the
 220     // plain index.
 221     assert(constantPoolCacheOopDesc::decode_secondary_index(~123) == 123, "else
 222     notl(index);  // convert to plain index
 223   } else if (index_size == sizeof(u1)) {
 224     assert(EnableInvokeDynamic, "tiny index used only for JSR 292");
 224     assert(EnableMethodHandles, "tiny index used only for EnableMethodHandles");
 225     load_unsigned_byte(index, Address(r13, bcp_offset));
 226   } else {
 227     ShouldNotReachHere();
 228   }
 229 }
```

_____***unchanged_portion_omitted_***

****
9803 Wed Mar 30 07:00:08 2011
new/src/cpu/x86/vm/interpreter_x86_32.cpp
****

*_____unchanged_portion_omitted_*

```
233 // Method handle invoker
234 // Dispatch a method of the form java.lang.invoke.MethodHandles::invoke(...)
235 address InterpreterGenerator::generate_method_handle_entry(void) {
236   if (!EnableInvokeDynamic) {
236   if (!EnableMethodHandles) {
237     return generate_abstract_entry();
238   }

240   address entry_point = MethodHandles::generate_method_handle_interpreter_entry(

242   return entry_point;
243 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   12279 Wed Mar 30 07:00:09 2011
new/src/cpu/x86/vm/interpreter_x86_64.cpp
**********************************************************
______unchanged_portion_omitted_


 320 // Method handle invoker
 321 // Dispatch a method of the form java.lang.invoke.MethodHandles::invoke(...)
 322 address InterpreterGenerator::generate_method_handle_entry(void) {
 323   if (!EnableInvokeDynamic) {
 323   if (!EnableMethodHandles) {
 324     return generate_abstract_entry();
 325   }

 327   address entry_point = MethodHandles::generate_method_handle_interpreter_entry(

 329   return entry_point;
 330 }
______unchanged_portion_omitted_
```

```
**********************************************************
   71864 Wed Mar 30 07:00:10 2011
new/src/cpu/x86/vm/templateInterpreter_x86_32.cpp
**********************************************************
______unchanged_portion_omitted_

1492 // asm based interpreter deoptimization helpers

1494 int AbstractInterpreter::layout_activation(methodOop method,
1495                                             int tempcount,
1496                                             int popframe_extra_args,
1497                                             int moncount,
1498                                             int callee_param_count,
1499                                             int callee_locals,
1500                                             frame* caller,
1501                                             frame* interpreter_frame,
1502                                             bool is_top_frame) {
1503   // Note: This calculation must exactly parallel the frame setup
1504   // in AbstractInterpreterGenerator::generate_method_entry.
1505   // If interpreter_frame!=NULL, set up the method, locals, and monitors.
1506   // The frame interpreter_frame, if not NULL, is guaranteed to be the right siz
1507   // as determined by a previous call to this method.
1508   // It is also guaranteed to be walkable even though it is in a skeletal state
1509   // NOTE: return size is in words not bytes

1511   // fixed size of an interpreter frame:
1512   int max_locals = method->max_locals() * Interpreter::stackElementWords;
1513   int extra_locals = (method->max_locals() - method->size_of_parameters()) *
1514                      Interpreter::stackElementWords;

1516   int overhead = frame::sender_sp_offset - frame::interpreter_frame_initial_sp_o

1518   // Our locals were accounted for by the caller (or last_frame_adjust on the tr
1519   // Since the callee parameters already account for the callee's params we only
1520   // the extra locals.


1523   int size = overhead +
1524          ((callee_locals - callee_param_count)*Interpreter::stackElementWords) +
1525          (moncount*frame::interpreter_frame_monitor_size()) +
1526          tempcount*Interpreter::stackElementWords + popframe_extra_args;

1528   if (interpreter_frame != NULL) {
1529 #ifdef ASSERT
1530     if (!EnableInvokeDynamic)
1530     if (!EnableMethodHandles)
1531       // @@@ FIXME: Should we correct interpreter_frame_sender_sp in the calling
1532       // Probably, since deoptimization doesn't work yet.
1533       assert(caller->unextended_sp() == interpreter_frame->interpreter_frame_sen
1534     assert(caller->sp() == interpreter_frame->sender_sp(), "Frame not properly w
1535 #endif

1537     interpreter_frame->interpreter_frame_set_method(method);
1538     // NOTE the difference in using sender_sp and interpreter_frame_sender_sp
1539     // interpreter_frame_sender_sp is the original sp of the caller (the unexten
1540     // and sender_sp is fp+8
1541     intptr_t* locals = interpreter_frame->sender_sp() + max_locals - 1;

1543     interpreter_frame->interpreter_frame_set_locals(locals);
1544     BasicObjectLock* montop = interpreter_frame->interpreter_frame_monitor_begin
1545     BasicObjectLock* monbot = montop - moncount;
1546     interpreter_frame->interpreter_frame_set_monitor_end(monbot);

1548     // Set last_sp
1549     intptr_t*  rsp = (intptr_t*) monbot  -
1550                      tempcount*Interpreter::stackElementWords -
```



```
1551                      popframe_extra_args;
1552     interpreter_frame->interpreter_frame_set_last_sp(rsp);

1554     // All frames but the initial (oldest) interpreter frame we fill in have a
1555     // value for sender_sp that allows walking the stack but isn't
1556     // truly correct. Correct the value here.

1558     if (extra_locals != 0 &&
1559         interpreter_frame->sender_sp() == interpreter_frame->interpreter_frame_s
1560       interpreter_frame->set_interpreter_frame_sender_sp(caller->sp() + extra_lo
1561     }
1562     *interpreter_frame->interpreter_frame_cache_addr() =
1563       method->constants()->cache();
1564   }
1565   return size;
1566 }
______unchanged_portion_omitted_
```

```
**********************************************************
   69765 Wed Mar 30 07:00:11 2011
new/src/cpu/x86/vm/templateInterpreter_x86_64.cpp
**********************************************************
______unchanged_portion_omitted_

1511 int AbstractInterpreter::layout_activation(methodOop method,
1512                                            int tempcount,
1513                                            int popframe_extra_args,
1514                                            int moncount,
1515                                            int callee_param_count,
1516                                            int callee_locals,
1517                                            frame* caller,
1518                                            frame* interpreter_frame,
1519                                            bool is_top_frame) {
1520   // Note: This calculation must exactly parallel the frame setup
1521   // in AbstractInterpreterGenerator::generate_method_entry.
1522   // If interpreter_frame!=NULL, set up the method, locals, and monitors.
1523   // The frame interpreter_frame, if not NULL, is guaranteed to be the
1524   // right size, as determined by a previous call to this method.
1525   // It is also guaranteed to be walkable even though it is in a skeletal state

1527   // fixed size of an interpreter frame:
1528   int max_locals = method->max_locals() * Interpreter::stackElementWords;
1529   int extra_locals = (method->max_locals() - method->size_of_parameters()) *
1530                      Interpreter::stackElementWords;

1532   int overhead = frame::sender_sp_offset -
1533                  frame::interpreter_frame_initial_sp_offset;
1534   // Our locals were accounted for by the caller (or last_frame_adjust
1535   // on the transistion) Since the callee parameters already account
1536   // for the callee's params we only need to account for the extra
1537   // locals.
1538   int size = overhead +
1539          (callee_locals - callee_param_count)*Interpreter::stackElementWords +
1540          moncount * frame::interpreter_frame_monitor_size() +
1541          tempcount* Interpreter::stackElementWords + popframe_extra_args;
1542   if (interpreter_frame != NULL) {
1543 #ifdef ASSERT
1544     if (!EnableInvokeDynamic)
1544     if (!EnableMethodHandles)
1545       // @@@ FIXME: Should we correct interpreter_frame_sender_sp in the calling
1546       // Probably, since deoptimization doesn't work yet.
1547       assert(caller->unextended_sp() == interpreter_frame->interpreter_frame_sen
1548     assert(caller->sp() == interpreter_frame->sender_sp(), "Frame not properly w
1549 #endif

1551     interpreter_frame->interpreter_frame_set_method(method);
1552     // NOTE the difference in using sender_sp and
1553     // interpreter_frame_sender_sp interpreter_frame_sender_sp is
1554     // the original sp of the caller (the unextended_sp) and
1555     // sender_sp is fp+16 XXX
1556     intptr_t* locals = interpreter_frame->sender_sp() + max_locals - 1;

1558     interpreter_frame->interpreter_frame_set_locals(locals);
1559     BasicObjectLock* montop = interpreter_frame->interpreter_frame_monitor_begin
1560     BasicObjectLock* monbot = montop - moncount;
1561     interpreter_frame->interpreter_frame_set_monitor_end(monbot);

1563     // Set last_sp
1564     intptr_t*  esp = (intptr_t*) monbot -
1565                      tempcount*Interpreter::stackElementWords -
1566                      popframe_extra_args;
1567     interpreter_frame->interpreter_frame_set_last_sp(esp);

1569     // All frames but the initial (oldest) interpreter frame we fill in have
```



```
1570     // a value for sender_sp that allows walking the stack but isn't
1571     // truly correct. Correct the value here.
1572     if (extra_locals != 0 &&
1573         interpreter_frame->sender_sp() ==
1574         interpreter_frame->interpreter_frame_sender_sp()) {
1575       interpreter_frame->set_interpreter_frame_sender_sp(caller->sp() +
1576                                                          extra_locals);
1577     }
1578     *interpreter_frame->interpreter_frame_cache_addr() =
1579       method->constants()->cache();
1580   }
1581   return size;
1582 }
______unchanged_portion_omitted_
```

****
117807 Wed Mar 30 07:00:12 2011
new/src/cpu/x86/vm/templateTable_x86_32.cpp
****
_____***unchanged_portion_omitted_***

```
 388 // Fast path for caching oop constants.
 389 // %%% We should use this to handle Class and String constants also.
 390 // %%% It will simplify the ldc/primitive path considerably.
 391 void TemplateTable::fast_aldc(bool wide) {
 392   transition(vtos, atos);

 394   if (!EnableInvokeDynamic) {
 395     // We should not encounter this bytecode if !EnableInvokeDynamic.
 394   if (!EnableMethodHandles) {
 395     // We should not encounter this bytecode if !EnableMethodHandles.
 396     // The verifier will stop it.  However, if we get past the verifier,
 397     // this will stop the thread in a reasonable way, without crashing the JVM.
 398     __ call_VM(noreg, CAST_FROM_FN_PTR(address,
 399                      InterpreterRuntime::throw_IncompatibleClassChangeError));
 400     // the call_VM checks for exception, so we should never return here.
 401     __ should_not_reach_here();
 402     return;
 403   }

 405   const Register cache = rcx;
 406   const Register index = rdx;

 408   resolve_cache_and_index(f1_oop, rax, cache, index, wide ? sizeof(u2) : sizeof(
 409   if (VerifyOops) {
 410     __ verify_oop(rax);
 411   }

 413   Label L_done, L_throw_exception;
 414   const Register con_klass_temp = rcx;  // same as Rcache
 415   __ movptr(con_klass_temp, Address(rax, oopDesc::klass_offset_in_bytes()));
 416   __ cmpptr(con_klass_temp, ExternalAddress((address)Universe::systemObjArrayKla
 417   __ jcc(Assembler::notEqual, L_done);
 418   __ cmpl(Address(rax, arrayOopDesc::length_offset_in_bytes()), 0);
 419   __ jcc(Assembler::notEqual, L_throw_exception);
 420   __ xorptr(rax, rax);
 421   __ jmp(L_done);

 423   // Load the exception from the system-array which wraps it:
 424   __ bind(L_throw_exception);
 425   __ movptr(rax, Address(rax, arrayOopDesc::base_offset_in_bytes(T_OBJECT)));
 426   __ jump(ExternalAddress(Interpreter::throw_exception_entry()));

 428   __ bind(L_done);
 429 }
```

_____***unchanged_portion_omitted_***

****
114546 Wed Mar 30 07:00:13 2011
new/src/cpu/x86/vm/templateTable_x86_64.cpp
****

_____***unchanged_portion_omitted_***

```
 402 // Fast path for caching oop constants.
 403 // %%% We should use this to handle Class and String constants also.
 404 // %%% It will simplify the ldc/primitive path considerably.
 405 void TemplateTable::fast_aldc(bool wide) {
 406   transition(vtos, atos);

 408   if (!EnableInvokeDynamic) {
 409     // We should not encounter this bytecode if !EnableInvokeDynamic.
 408   if (!EnableMethodHandles) {
 409     // We should not encounter this bytecode if !EnableMethodHandles.
 410     // The verifier will stop it.  However, if we get past the verifier,
 411     // this will stop the thread in a reasonable way, without crashing the JVM.
 412     __ call_VM(noreg, CAST_FROM_FN_PTR(address,
 413                        InterpreterRuntime::throw_IncompatibleClassChangeError));
 414     // the call_VM checks for exception, so we should never return here.
 415     __ should_not_reach_here();
 416     return;
 417   }

 419   const Register cache = rcx;
 420   const Register index = rdx;

 422   resolve_cache_and_index(f1_oop, rax, cache, index, wide ? sizeof(u2) : sizeof(
 423   if (VerifyOops) {
 424     __ verify_oop(rax);
 425   }

 427   Label L_done, L_throw_exception;
 428   const Register con_klass_temp = rcx;  // same as cache
 429   const Register array_klass_temp = rdx;  // same as index
 430   __ movptr(con_klass_temp, Address(rax, oopDesc::klass_offset_in_bytes()));
 431   __ lea(array_klass_temp, ExternalAddress((address)Universe::systemObjArrayKlas
 432   __ cmpptr(con_klass_temp, Address(array_klass_temp, 0));
 433   __ jcc(Assembler::notEqual, L_done);
 434   __ cmpl(Address(rax, arrayOopDesc::length_offset_in_bytes()), 0);
 435   __ jcc(Assembler::notEqual, L_throw_exception);
 436   __ xorptr(rax, rax);
 437   __ jmp(L_done);

 439   // Load the exception from the system-array which wraps it:
 440   __ bind(L_throw_exception);
 441   __ movptr(rax, Address(rax, arrayOopDesc::base_offset_in_bytes(T_OBJECT)));
 442   __ jump(ExternalAddress(Interpreter::throw_exception_entry()));

 444   __ bind(L_done);
 445 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  192471 Wed Mar 30 07:00:14 2011
new/src/share/vm/classfile/classFileParser.cpp
**********************************************************
```

```
/*
 * Copyright (c) 1997, 2011, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 *
 */

#include "precompiled.hpp"
#include "classfile/classFileParser.hpp"
#include "classfile/classLoader.hpp"
#include "classfile/javaClasses.hpp"
#include "classfile/symbolTable.hpp"
#include "classfile/systemDictionary.hpp"
#include "classfile/verificationType.hpp"
#include "classfile/verifier.hpp"
#include "classfile/vmSymbols.hpp"
#include "memory/allocation.hpp"
#include "memory/gcLocker.hpp"
#include "memory/oopFactory.hpp"
#include "memory/universe.inline.hpp"
#include "oops/constantPoolOop.hpp"
#include "oops/instanceKlass.hpp"
#include "oops/instanceMirrorKlass.hpp"
#include "oops/klass.inline.hpp"
#include "oops/klassOop.hpp"
#include "oops/klassVtable.hpp"
#include "oops/methodOop.hpp"
#include "oops/symbol.hpp"
#include "prims/jvmtiExport.hpp"
#include "runtime/javaCalls.hpp"
#include "runtime/perfData.hpp"
#include "runtime/reflection.hpp"
#include "runtime/signature.hpp"
#include "runtime/timer.hpp"
#include "services/classLoadingService.hpp"
#include "services/threadService.hpp"

// We generally try to create the oops directly when parsing, rather than
// allocating temporary data structures and copying the bytes twice. A
// temporary area is only needed when parsing utf8 entries in the constant
// pool and when parsing line number tables.

// We add assert in debug mode when class format is not checked.

#define JAVA_CLASSFILE_MAGIC              0xCAFEBABE
```



```
#define JAVA_MIN_SUPPORTED_VERSION        45
#define JAVA_MAX_SUPPORTED_VERSION        51
#define JAVA_MAX_SUPPORTED_MINOR_VERSION  0

// Used for two backward compatibility reasons:
// - to check for new additions to the class file format in JDK1.5
// - to check for bug fixes in the format checker in JDK1.5
#define JAVA_1_5_VERSION                  49

// Used for backward compatibility reasons:
// - to check for javac bug fixes that happened after 1.5
// - also used as the max version when running in jdk6
#define JAVA_6_VERSION                    50

// Used for backward compatibility reasons:
// - to check NameAndType_info signatures more aggressively
#define JAVA_7_VERSION                    51


void ClassFileParser::parse_constant_pool_entries(constantPoolHandle cp, int len
  // Use a local copy of ClassFileStream. It helps the C++ compiler to optimize
  // this function (_current can be allocated in a register, with scalar
  // replacement of aggregates). The _current pointer is copied back to
  // stream() when this function returns. DON'T call another method within
  // this method that uses stream().
  ClassFileStream* cfs0 = stream();
  ClassFileStream cfs1 = *cfs0;
  ClassFileStream* cfs = &cfs1;
#ifdef ASSERT
  assert(cfs->allocated_on_stack(),"should be local");
  u1* old_current = cfs0->current();
#endif

  // Used for batching symbol allocations.
  const char* names[SymbolTable::symbol_alloc_batch_size];
  int lengths[SymbolTable::symbol_alloc_batch_size];
  int indices[SymbolTable::symbol_alloc_batch_size];
  unsigned int hashValues[SymbolTable::symbol_alloc_batch_size];
  int names_count = 0;

  // parsing  Index 0 is unused
  for (int index = 1; index < length; index++) {
    // Each of the following case guarantees one more byte in the stream
    // for the following tag or the access_flags following constant pool,
    // so we don't need bounds-check for reading tag.
    u1 tag = cfs->get_u1_fast();
    switch (tag) {
      case JVM_CONSTANT_Class :
        {
          cfs->guarantee_more(3, CHECK);  // name_index, tag/access_flags
          u2 name_index = cfs->get_u2_fast();
          cp->klass_index_at_put(index, name_index);
        }
        break;
      case JVM_CONSTANT_Fieldref :
        {
          cfs->guarantee_more(5, CHECK);  // class_index, name_and_type_index, t
          u2 class_index = cfs->get_u2_fast();
          u2 name_and_type_index = cfs->get_u2_fast();
          cp->field_at_put(index, class_index, name_and_type_index);
        }
        break;
      case JVM_CONSTANT_Methodref :
        {
          cfs->guarantee_more(5, CHECK);  // class_index, name_and_type_index, t
          u2 class_index = cfs->get_u2_fast();
```

```
129           u2 name_and_type_index = cfs->get_u2_fast();
130           cp->method_at_put(index, class_index, name_and_type_index);
131         }
132         break;
133       case JVM_CONSTANT_InterfaceMethodref :
134         {
135           cfs->guarantee_more(5, CHECK);  // class_index, name_and_type_index, t
136           u2 class_index = cfs->get_u2_fast();
137           u2 name_and_type_index = cfs->get_u2_fast();
138           cp->interface_method_at_put(index, class_index, name_and_type_index);
139         }
140         break;
141       case JVM_CONSTANT_String :
142         {
143           cfs->guarantee_more(3, CHECK);  // string_index, tag/access_flags
144           u2 string_index = cfs->get_u2_fast();
145           cp->string_index_at_put(index, string_index);
146         }
147         break;
148       case JVM_CONSTANT_MethodHandle :
149       case JVM_CONSTANT_MethodType :
150         if (_major_version < Verifier::INVOKEDYNAMIC_MAJOR_VERSION) {
151           classfile_parse_error(
152             "Class file version does not support constant tag %u in class file %
153             tag, CHECK);
154         }
155         if (!EnableInvokeDynamic) {
155         if (!EnableMethodHandles) {
156           classfile_parse_error(
157             "This JVM does not support constant tag %u in class file %s",
158             tag, CHECK);
159         }
160         if (tag == JVM_CONSTANT_MethodHandle) {
161           cfs->guarantee_more(4, CHECK);  // ref_kind, method_index, tag/access_
162           u1 ref_kind = cfs->get_u1_fast();
163           u2 method_index = cfs->get_u2_fast();
164           cp->method_handle_index_at_put(index, ref_kind, method_index);
165         } else if (tag == JVM_CONSTANT_MethodType) {
166           cfs->guarantee_more(3, CHECK);  // signature_index, tag/access_flags
167           u2 signature_index = cfs->get_u2_fast();
168           cp->method_type_index_at_put(index, signature_index);
169         } else {
170           ShouldNotReachHere();
171         }
172         break;
173       case JVM_CONSTANT_InvokeDynamicTrans :  // this tag appears only in old cl
174       case JVM_CONSTANT_InvokeDynamic :
175         {
176           if (_major_version < Verifier::INVOKEDYNAMIC_MAJOR_VERSION) {
177             classfile_parse_error(
178               "Class file version does not support constant tag %u in class file
179               tag, CHECK);
180           }
181           if (!EnableInvokeDynamic) {
182             classfile_parse_error(
183               "This JVM does not support constant tag %u in class file %s",
184               tag, CHECK);
185           }
186           cfs->guarantee_more(5, CHECK);  // bsm_index, nt, tag/access_flags
187           u2 bootstrap_specifier_index = cfs->get_u2_fast();
188           u2 name_and_type_index = cfs->get_u2_fast();
189           if (tag == JVM_CONSTANT_InvokeDynamicTrans) {
190             if (!AllowTransitionalJSR292)
191               classfile_parse_error(
192                 "This JVM does not support transitional InvokeDynamic tag %u in
193                 tag, CHECK);
```



```
194             cp->invoke_dynamic_trans_at_put(index, bootstrap_specifier_index, na
195             break;
196           }
197           if (_max_bootstrap_specifier_index < (int) bootstrap_specifier_index)
198             _max_bootstrap_specifier_index = (int) bootstrap_specifier_index;  /
199           cp->invoke_dynamic_at_put(index, bootstrap_specifier_index, name_and_t
200         }
201         break;
202       case JVM_CONSTANT_Integer :
203         {
204           cfs->guarantee_more(5, CHECK);  // bytes, tag/access_flags
205           u4 bytes = cfs->get_u4_fast();
206           cp->int_at_put(index, (jint) bytes);
207         }
208         break;
209       case JVM_CONSTANT_Float :
210         {
211           cfs->guarantee_more(5, CHECK);  // bytes, tag/access_flags
212           u4 bytes = cfs->get_u4_fast();
213           cp->float_at_put(index, *(jfloat*)&bytes);
214         }
215         break;
216       case JVM_CONSTANT_Long :
217         // A mangled type might cause you to overrun allocated memory
218         guarantee_property(index+1 < length,
219                            "Invalid constant pool entry %u in class file %s",
220                            index, CHECK);
221         {
222           cfs->guarantee_more(9, CHECK);  // bytes, tag/access_flags
223           u8 bytes = cfs->get_u8_fast();
224           cp->long_at_put(index, bytes);
225         }
226         index++;   // Skip entry following eigth-byte constant, see JVM book p.
227         break;
228       case JVM_CONSTANT_Double :
229         // A mangled type might cause you to overrun allocated memory
230         guarantee_property(index+1 < length,
231                            "Invalid constant pool entry %u in class file %s",
232                            index, CHECK);
233         {
234           cfs->guarantee_more(9, CHECK);  // bytes, tag/access_flags
235           u8 bytes = cfs->get_u8_fast();
236           cp->double_at_put(index, *(jdouble*)&bytes);
237         }
238         index++;   // Skip entry following eigth-byte constant, see JVM book p.
239         break;
240       case JVM_CONSTANT_NameAndType :
241         {
242           cfs->guarantee_more(5, CHECK);  // name_index, signature_index, tag/ac
243           u2 name_index = cfs->get_u2_fast();
244           u2 signature_index = cfs->get_u2_fast();
245           cp->name_and_type_at_put(index, name_index, signature_index);
246         }
247         break;
248       case JVM_CONSTANT_Utf8 :
249         {
250           cfs->guarantee_more(2, CHECK);  // utf8_length
251           u2  utf8_length = cfs->get_u2_fast();
252           u1* utf8_buffer = cfs->get_u1_buffer();
253           assert(utf8_buffer != NULL, "null utf8 buffer");
254           // Got utf8 string, guarantee utf8_length+1 bytes, set stream position
255           cfs->guarantee_more(utf8_length+1, CHECK);  // utf8 string, tag/access
256           cfs->skip_u1_fast(utf8_length);

258           // Before storing the symbol, make sure it's legal
259           if (_need_verify) {
```

```
260             verify_legal_utf8((unsigned char*)utf8_buffer, utf8_length, CHECK);
261           }

263           if (EnableInvokeDynamic && has_cp_patch_at(index)) {
263           if (AnonymousClasses && has_cp_patch_at(index)) {
264             Handle patch = clear_cp_patch_at(index);
265             guarantee_property(java_lang_String::is_instance(patch()),
266                                "Illegal utf8 patch at %d in class file %s",
267                                index, CHECK);
268             char* str = java_lang_String::as_utf8_string(patch());
269             // (could use java_lang_String::as_symbol instead, but might as well
270             utf8_buffer = (u1*) str;
271             utf8_length = (int) strlen(str);
272           }

274           unsigned int hash;
275           Symbol* result = SymbolTable::lookup_only((char*)utf8_buffer, utf8_len
276           if (result == NULL) {
277             names[names_count] = (char*)utf8_buffer;
278             lengths[names_count] = utf8_length;
279             indices[names_count] = index;
280             hashValues[names_count++] = hash;
281             if (names_count == SymbolTable::symbol_alloc_batch_size) {
282               SymbolTable::new_symbols(cp, names_count, names, lengths, indices,
283               names_count = 0;
284             }
285           } else {
286             cp->symbol_at_put(index, result);
287           }
288         }
289         break;
290       default:
291         classfile_parse_error(
292           "Unknown constant tag %u in class file %s", tag, CHECK);
293         break;
294     }
295   }

297   // Allocate the remaining symbols
298   if (names_count > 0) {
299     SymbolTable::new_symbols(cp, names_count, names, lengths, indices, hashValue
300   }

302   // Copy _current pointer of local copy back to stream().
303 #ifdef ASSERT
304   assert(cfs0->current() == old_current, "non-exclusive use of stream()");
305 #endif
306   cfs0->set_current(cfs1.current());
307 }
```

unchanged_portion_omitted_

```
326 bool inline valid_cp_range(int index, int length) { return (index > 0 && index <

328 constantPoolHandle ClassFileParser::parse_constant_pool(TRAPS) {
329   ClassFileStream* cfs = stream();
330   constantPoolHandle nullHandle;

332   cfs->guarantee_more(3, CHECK_(nullHandle)); // length, first cp tag
333   u2 length = cfs->get_u2_fast();
334   guarantee_property(
335     length >= 1, "Illegal constant pool size %u in class file %s",
336     length, CHECK_(nullHandle));
337   constantPoolOop constant_pool =
338                       oopFactory::new_constantPool(length,
339                                                    oopDesc::IsSafeConc,
340                                                    CHECK_(nullHandle));
```



```
341   constantPoolHandle cp (THREAD, constant_pool);

343   cp->set_partially_loaded();    // Enables heap verify to work on partial const
344   ConstantPoolCleaner cp_in_error(cp); // set constant pool to be cleaned up.

346   // parsing constant pool entries
347   parse_constant_pool_entries(cp, length, CHECK_(nullHandle));

349   int index = 1;  // declared outside of loops for portability

351   // first verification pass - validate cross references and fixup class and str
352   for (index = 1; index < length; index++) {          // Index 0 is unused
353     jbyte tag = cp->tag_at(index).value();
354     switch (tag) {
355       case JVM_CONSTANT_Class :
356         ShouldNotReachHere();     // Only JVM_CONSTANT_ClassIndex should be pres
357         break;
358       case JVM_CONSTANT_Fieldref :
359         // fall through
360       case JVM_CONSTANT_Methodref :
361         // fall through
362       case JVM_CONSTANT_InterfaceMethodref : {
363         if (!_need_verify) break;
364         int klass_ref_index = cp->klass_ref_index_at(index);
365         int name_and_type_ref_index = cp->name_and_type_ref_index_at(index);
366         check_property(valid_cp_range(klass_ref_index, length) &&
367                        is_klass_reference(cp, klass_ref_index),
368                        "Invalid constant pool index %u in class file %s",
369                        klass_ref_index,
370                        CHECK_(nullHandle));
371         check_property(valid_cp_range(name_and_type_ref_index, length) &&
372                        cp->tag_at(name_and_type_ref_index).is_name_and_type(),
373                        "Invalid constant pool index %u in class file %s",
374                        name_and_type_ref_index,
375                        CHECK_(nullHandle));
376         break;
377       }
378       case JVM_CONSTANT_String :
379         ShouldNotReachHere();     // Only JVM_CONSTANT_StringIndex should be pre
380         break;
381       case JVM_CONSTANT_Integer :
382         break;
383       case JVM_CONSTANT_Float :
384         break;
385       case JVM_CONSTANT_Long :
386       case JVM_CONSTANT_Double :
387         index++;
388         check_property(
389           (index < length && cp->tag_at(index).is_invalid()),
390           "Improper constant pool long/double index %u in class file %s",
391           index, CHECK_(nullHandle));
392         break;
393       case JVM_CONSTANT_NameAndType : {
394         if (!_need_verify) break;
395         int name_ref_index = cp->name_ref_index_at(index);
396         int signature_ref_index = cp->signature_ref_index_at(index);
397         check_property(
398           valid_cp_range(name_ref_index, length) &&
399             cp->tag_at(name_ref_index).is_utf8(),
400           "Invalid constant pool index %u in class file %s",
401           name_ref_index, CHECK_(nullHandle));
402         check_property(
403           valid_cp_range(signature_ref_index, length) &&
404             cp->tag_at(signature_ref_index).is_utf8(),
405           "Invalid constant pool index %u in class file %s",
406           signature_ref_index, CHECK_(nullHandle));
```

```
407         break;
408       }
409       case JVM_CONSTANT_Utf8 :
410         break;
411       case JVM_CONSTANT_UnresolvedClass :         // fall-through
412       case JVM_CONSTANT_UnresolvedClassInError:
413         ShouldNotReachHere();     // Only JVM_CONSTANT_ClassIndex should be pres
414         break;
415       case JVM_CONSTANT_ClassIndex :
416         {
417           int class_index = cp->klass_index_at(index);
418           check_property(
419             valid_cp_range(class_index, length) &&
420               cp->tag_at(class_index).is_utf8(),
421             "Invalid constant pool index %u in class file %s",
422             class_index, CHECK_(nullHandle));
423           cp->unresolved_klass_at_put(index, cp->symbol_at(class_index));
424         }
425         break;
426       case JVM_CONSTANT_UnresolvedString :
427         ShouldNotReachHere();     // Only JVM_CONSTANT_StringIndex should be pre
428         break;
429       case JVM_CONSTANT_StringIndex :
430         {
431           int string_index = cp->string_index_at(index);
432           check_property(
433             valid_cp_range(string_index, length) &&
434               cp->tag_at(string_index).is_utf8(),
435             "Invalid constant pool index %u in class file %s",
436             string_index, CHECK_(nullHandle));
437           Symbol* sym = cp->symbol_at(string_index);
438           cp->unresolved_string_at_put(index, sym);
439         }
440         break;
441       case JVM_CONSTANT_MethodHandle :
442         {
443           int ref_index = cp->method_handle_index_at(index);
444           check_property(
445             valid_cp_range(ref_index, length) &&
446                 EnableInvokeDynamic,
446                 EnableMethodHandles,
447               "Invalid constant pool index %u in class file %s",
448               ref_index, CHECK_(nullHandle));
449           constantTag tag = cp->tag_at(ref_index);
450           int ref_kind  = cp->method_handle_ref_kind_at(index);
451           switch (ref_kind) {
452           case JVM_REF_getField:
453           case JVM_REF_getStatic:
454           case JVM_REF_putField:
455           case JVM_REF_putStatic:
456             check_property(
457               tag.is_field(),
458               "Invalid constant pool index %u in class file %s (not a field)",
459               ref_index, CHECK_(nullHandle));
460             break;
461           case JVM_REF_invokeVirtual:
462           case JVM_REF_invokeStatic:
463           case JVM_REF_invokeSpecial:
464           case JVM_REF_newInvokeSpecial:
465             check_property(
466               tag.is_method(),
467               "Invalid constant pool index %u in class file %s (not a method)",
468               ref_index, CHECK_(nullHandle));
469             break;
470           case JVM_REF_invokeInterface:
471             check_property(
```



```
472               tag.is_interface_method(),
473               "Invalid constant pool index %u in class file %s (not an interface
474               ref_index, CHECK_(nullHandle));
475             break;
476           default:
477             classfile_parse_error(
478               "Bad method handle kind at constant pool index %u in class file %s
479               index, CHECK_(nullHandle));
480           }
481           // Keep the ref_index unchanged.  It will be indirected at link-time.
482         }
483         break;
484       case JVM_CONSTANT_MethodType :
485         {
486           int ref_index = cp->method_type_index_at(index);
487           check_property(
488             valid_cp_range(ref_index, length) &&
489                cp->tag_at(ref_index).is_utf8() &&
490                EnableInvokeDynamic,
490                EnableMethodHandles,
491               "Invalid constant pool index %u in class file %s",
492               ref_index, CHECK_(nullHandle));
493         }
494         break;
495       case JVM_CONSTANT_InvokeDynamicTrans :
496       case JVM_CONSTANT_InvokeDynamic :
497         {
498           int name_and_type_ref_index = cp->invoke_dynamic_name_and_type_ref_ind
499           check_property(valid_cp_range(name_and_type_ref_index, length) &&
500                          cp->tag_at(name_and_type_ref_index).is_name_and_type(),
501                          "Invalid constant pool index %u in class file %s",
502                          name_and_type_ref_index,
503                          CHECK_(nullHandle));
504           if (tag == JVM_CONSTANT_InvokeDynamicTrans) {
505             int bootstrap_method_ref_index = cp->invoke_dynamic_bootstrap_method
506             check_property(valid_cp_range(bootstrap_method_ref_index, length) &&
507                            cp->tag_at(bootstrap_method_ref_index).is_method_hand
508                            "Invalid constant pool index %u in class file %s",
509                            bootstrap_method_ref_index,
510                            CHECK_(nullHandle));
511           }
512           // bootstrap specifier index must be checked later, when BootstrapMeth
513           break;
514         }
515       default:
516         fatal(err_msg("bad constant pool tag value %u",
517                       cp->tag_at(index).value()));
518         ShouldNotReachHere();
519         break;
520     } // end of switch
521   } // end of for

523   if (_cp_patches != NULL) {
524     // need to treat this_class specially...
525     assert(EnableInvokeDynamic, "");
525     assert(AnonymousClasses, "");
526     int this_class_index;
527     {
528       cfs->guarantee_more(8, CHECK_(nullHandle));  // flags, this_class, super_c
529       u1* mark = cfs->current();
530       u2 flags         = cfs->get_u2_fast();
531       this_class_index = cfs->get_u2_fast();
532       cfs->set_current(mark);  // revert to mark
533     }

535     for (index = 1; index < length; index++) {          // Index 0 is unused
```

```
536       if (has_cp_patch_at(index)) {
537         guarantee_property(index != this_class_index,
538                            "Illegal constant pool patch to self at %d in class f
539                             index, CHECK_(nullHandle));
540         patch_constant_pool(cp, index, cp_patch_at(index), CHECK_(nullHandle));
541       }
542     }
543     // Ensure that all the patches have been used.
544     for (index = 0; index < _cp_patches->length(); index++) {
545       guarantee_property(!has_cp_patch_at(index),
546                          "Unused constant pool patch at %d in class file %s",
547                          index, CHECK_(nullHandle));
548     }
549   }

551   if (!_need_verify) {
552     cp_in_error.set_in_error(false);
553     return cp;
554   }

556   // second verification pass - checks the strings are of the right format.
557   // but not yet to the other entries
558   for (index = 1; index < length; index++) {
559     jbyte tag = cp->tag_at(index).value();
560     switch (tag) {
561       case JVM_CONSTANT_UnresolvedClass: {
562         Symbol*  class_name = cp->unresolved_klass_at(index);
563         // check the name, even if _cp_patches will overwrite it
564         verify_legal_class_name(class_name, CHECK_(nullHandle));
565         break;
566       }
567       case JVM_CONSTANT_NameAndType: {
568         if (_need_verify && _major_version >= JAVA_7_VERSION) {
569           int sig_index = cp->signature_ref_index_at(index);
570           int name_index = cp->name_ref_index_at(index);
571           Symbol*  name = cp->symbol_at(name_index);
572           Symbol*  sig = cp->symbol_at(sig_index);
573           if (sig->byte_at(0) == JVM_SIGNATURE_FUNC) {
574             verify_legal_method_signature(name, sig, CHECK_(nullHandle));
575           } else {
576             verify_legal_field_signature(name, sig, CHECK_(nullHandle));
577           }
578         }
579         break;
580       }
581       case JVM_CONSTANT_Fieldref:
582       case JVM_CONSTANT_Methodref:
583       case JVM_CONSTANT_InterfaceMethodref: {
584         int name_and_type_ref_index = cp->name_and_type_ref_index_at(index);
585         // already verified to be utf8
586         int name_ref_index = cp->name_ref_index_at(name_and_type_ref_index);
587         // already verified to be utf8
588         int signature_ref_index = cp->signature_ref_index_at(name_and_type_ref_i
589         Symbol*  name = cp->symbol_at(name_ref_index);
590         Symbol*  signature = cp->symbol_at(signature_ref_index);
591         if (tag == JVM_CONSTANT_Fieldref) {
592           verify_legal_field_name(name, CHECK_(nullHandle));
593           if (_need_verify && _major_version >= JAVA_7_VERSION) {
594             // Signature is verified above, when iterating NameAndType_info.
595             // Need only to be sure it's the right type.
596             if (signature->byte_at(0) == JVM_SIGNATURE_FUNC) {
597               throwIllegalSignature(
598                   "Field", name, signature, CHECK_(nullHandle));
599             }
600           } else {
601             verify_legal_field_signature(name, signature, CHECK_(nullHandle));
```



```
602           }
603         } else {
604           verify_legal_method_name(name, CHECK_(nullHandle));
605           if (_need_verify && _major_version >= JAVA_7_VERSION) {
606             // Signature is verified above, when iterating NameAndType_info.
607             // Need only to be sure it's the right type.
608             if (signature->byte_at(0) != JVM_SIGNATURE_FUNC) {
609               throwIllegalSignature(
610                   "Method", name, signature, CHECK_(nullHandle));
611             }
612           } else {
613             verify_legal_method_signature(name, signature, CHECK_(nullHandle));
614           }
615           if (tag == JVM_CONSTANT_Methodref) {
616             // 4509014: If a class method name begins with '<', it must be "<ini
617             assert(name != NULL, "method name in constant pool is null");
618             unsigned int name_len = name->utf8_length();
619             assert(name_len > 0, "bad method name");  // already verified as leg
620             if (name->byte_at(0) == '<') {
621               if (name != vmSymbols::object_initializer_name()) {
622                 classfile_parse_error(
623                   "Bad method name at constant pool index %u in class file %s",
624                   name_ref_index, CHECK_(nullHandle));
625               }
626             }
627           }
628         }
629         break;
630       }
631       case JVM_CONSTANT_MethodHandle: {
632         int ref_index = cp->method_handle_index_at(index);
633         int ref_kind  = cp->method_handle_ref_kind_at(index);
634         switch (ref_kind) {
635         case JVM_REF_invokeVirtual:
636         case JVM_REF_invokeStatic:
637         case JVM_REF_invokeSpecial:
638         case JVM_REF_newInvokeSpecial:
639           {
640             int name_and_type_ref_index = cp->name_and_type_ref_index_at(ref_ind
641             int name_ref_index = cp->name_ref_index_at(name_and_type_ref_index);
642             Symbol*  name = cp->symbol_at(name_ref_index);
643             if (ref_kind == JVM_REF_newInvokeSpecial) {
644               if (name != vmSymbols::object_initializer_name()) {
645                 classfile_parse_error(
646                   "Bad constructor name at constant pool index %u in class file
647                   name_ref_index, CHECK_(nullHandle));
648               }
649             } else {
650               if (name == vmSymbols::object_initializer_name()) {
651                 classfile_parse_error(
652                   "Bad method name at constant pool index %u in class file %s",
653                   name_ref_index, CHECK_(nullHandle));
654               }
655             }
656           }
657           break;
658           // Other ref_kinds are already fully checked in previous pass.
659         }
660         break;
661       }
662       case JVM_CONSTANT_MethodType: {
663         Symbol* no_name = vmSymbols::type_name(); // place holder
664         Symbol*  signature = cp->method_type_signature_at(index);
665         verify_legal_method_signature(no_name, signature, CHECK_(nullHandle));
666         break;
667       }
```

```
668       case JVM_CONSTANT_Utf8: {
669         assert(cp->symbol_at(index)->refcount() != 0, "count corrupted");
670       }
671     }  // end of switch
672   }  // end of for

674   cp_in_error.set_in_error(false);
675   return cp;
676 }


679 void ClassFileParser::patch_constant_pool(constantPoolHandle cp, int index, Hand
680   assert(EnableInvokeDynamic, "");
680   assert(AnonymousClasses, "");
681   BasicType patch_type = T_VOID;
682   switch (cp->tag_at(index).value()) {

684   case JVM_CONSTANT_UnresolvedClass :
685     // Patching a class means pre-resolving it.
686     // The name in the constant pool is ignored.
687     if (java_lang_Class::is_instance(patch())) {
688       guarantee_property(!java_lang_Class::is_primitive(patch()),
689                          "Illegal class patch at %d in class file %s",
690                          index, CHECK);
691       cp->klass_at_put(index, java_lang_Class::as_klassOop(patch()));
692     } else {
693       guarantee_property(java_lang_String::is_instance(patch()),
694                          "Illegal class patch at %d in class file %s",
695                          index, CHECK);
696       Symbol* name = java_lang_String::as_symbol(patch(), CHECK);
697       cp->unresolved_klass_at_put(index, name);
698     }
699     break;

701   case JVM_CONSTANT_UnresolvedString :
702     // Patching a string means pre-resolving it.
703     // The spelling in the constant pool is ignored.
704     // The constant reference may be any object whatever.
705     // If it is not a real interned string, the constant is referred
706     // to as a "pseudo-string", and must be presented to the CP
707     // explicitly, because it may require scavenging.
708     cp->pseudo_string_at_put(index, patch());
709     break;

711   case JVM_CONSTANT_Integer : patch_type = T_INT;    goto patch_prim;
712   case JVM_CONSTANT_Float :   patch_type = T_FLOAT;  goto patch_prim;
713   case JVM_CONSTANT_Long :    patch_type = T_LONG;   goto patch_prim;
714   case JVM_CONSTANT_Double :  patch_type = T_DOUBLE; goto patch_prim;
715   patch_prim:
716     {
717       jvalue value;
718       BasicType value_type = java_lang_boxing_object::get_value(patch(), &value)
719       guarantee_property(value_type == patch_type,
720                          "Illegal primitive patch at %d in class file %s",
721                          index, CHECK);
722       switch (value_type) {
723       case T_INT:    cp->int_at_put(index,   value.i); break;
724       case T_FLOAT:  cp->float_at_put(index, value.f); break;
725       case T_LONG:   cp->long_at_put(index,  value.j); break;
726       case T_DOUBLE: cp->double_at_put(index, value.d); break;
727       default:       assert(false, "");
728       }
729     }
730     break;

732   default:
```



```
733     // %%% TODO: put method handles into CONSTANT_InterfaceMethodref, etc.
734     guarantee_property(!has_cp_patch_at(index),
735                        "Illegal unexpected patch at %d in class file %s",
736                        index, CHECK);
737     return;
738   }

740   // On fall-through, mark the patch as used.
741   clear_cp_patch_at(index);
742 }
_____unchanged_portion_omitted_

1578 #define MAX_ARGS_SIZE 255
1579 #define MAX_CODE_SIZE 65535
1580 #define INITIAL_MAX_LVT_NUMBER 256

1582 // Note: the parse_method below is big and clunky because all parsing of the cod
1583 // attribute is inlined. This is curbersome to avoid since we inline most of the
1584 // methodOop to save footprint, so we only know the size of the resulting method
1585 // entire method attribute is parsed.
1586 //
1587 // The promoted_flags parameter is used to pass relevant access_flags
1588 // from the method back up to the containing klass. These flag values
1589 // are added to klass's access_flags.

1591 methodHandle ClassFileParser::parse_method(constantPoolHandle cp, bool is_interf
1592                                            AccessFlags *promoted_flags,
1593                                            typeArrayHandle* method_annotations,
1594                                            typeArrayHandle* method_parameter_ann
1595                                            typeArrayHandle* method_default_annot
1596                                            TRAPS) {
1597   ClassFileStream* cfs = stream();
1598   methodHandle nullHandle;
1599   ResourceMark rm(THREAD);
1600   // Parse fixed parts
1601   cfs->guarantee_more(8, CHECK_(nullHandle)); // access_flags, name_index, descr

1603   int flags = cfs->get_u2_fast();
1604   u2 name_index = cfs->get_u2_fast();
1605   int cp_size = cp->length();
1606   check_property(
1607     valid_cp_range(name_index, cp_size) &&
1608       cp->tag_at(name_index).is_utf8(),
1609     "Illegal constant pool index %u for method name in class file %s",
1610     name_index, CHECK_(nullHandle));
1611   Symbol*  name = cp->symbol_at(name_index);
1612   verify_legal_method_name(name, CHECK_(nullHandle));

1614   u2 signature_index = cfs->get_u2_fast();
1615   guarantee_property(
1616     valid_cp_range(signature_index, cp_size) &&
1617       cp->tag_at(signature_index).is_utf8(),
1618     "Illegal constant pool index %u for method signature in class file %s",
1619     signature_index, CHECK_(nullHandle));
1620   Symbol*  signature = cp->symbol_at(signature_index);

1622   AccessFlags access_flags;
1623   if (name == vmSymbols::class_initializer_name()) {
1624     // We ignore the other access flags for a valid class initializer.
1625     // (JVM Spec 2nd ed., chapter 4.6)
1626     if (_major_version < 51) { // backward compatibility
1627       flags = JVM_ACC_STATIC;
1628     } else if ((flags & JVM_ACC_STATIC) == JVM_ACC_STATIC) {
1629       flags &= JVM_ACC_STATIC | JVM_ACC_STRICT;
1630     }
1631   } else {
```

```
1632      verify_legal_method_modifiers(flags, is_interface, name, CHECK_(nullHandle))
1633    }

1635    int args_size = -1;  // only used when _need_verify is true
1636    if (_need_verify) {
1637      args_size = ((flags & JVM_ACC_STATIC) ? 0 : 1) +
1638                  verify_legal_method_signature(name, signature, CHECK_(nullHandl
1639      if (args_size > MAX_ARGS_SIZE) {
1640        classfile_parse_error("Too many arguments in method signature in class fil
1641      }
1642    }

1644    access_flags.set_flags(flags & JVM_RECOGNIZED_METHOD_MODIFIERS);

1646    // Default values for code and exceptions attribute elements
1647    u2 max_stack = 0;
1648    u2 max_locals = 0;
1649    u4 code_length = 0;
1650    u1* code_start = 0;
1651    u2 exception_table_length = 0;
1652    typeArrayHandle exception_handlers(THREAD, Universe::the_empty_int_array());
1653    u2 checked_exceptions_length = 0;
1654    u2* checked_exceptions_start = NULL;
1655    CompressedLineNumberWriteStream* linenumber_table = NULL;
1656    int linenumber_table_length = 0;
1657    int total_lvt_length = 0;
1658    u2 lvt_cnt = 0;
1659    u2 lvtt_cnt = 0;
1660    bool lvt_allocated = false;
1661    u2 max_lvt_cnt = INITIAL_MAX_LVT_NUMBER;
1662    u2 max_lvtt_cnt = INITIAL_MAX_LVT_NUMBER;
1663    u2* localvariable_table_length;
1664    u2** localvariable_table_start;
1665    u2* localvariable_type_table_length;
1666    u2** localvariable_type_table_start;
1667    bool parsed_code_attribute = false;
1668    bool parsed_checked_exceptions_attribute = false;
1669    bool parsed_stackmap_attribute = false;
1670    // stackmap attribute - JDK1.5
1671    typeArrayHandle stackmap_data;
1672    u2 generic_signature_index = 0;
1673    u1* runtime_visible_annotations = NULL;
1674    int runtime_visible_annotations_length = 0;
1675    u1* runtime_invisible_annotations = NULL;
1676    int runtime_invisible_annotations_length = 0;
1677    u1* runtime_visible_parameter_annotations = NULL;
1678    int runtime_visible_parameter_annotations_length = 0;
1679    u1* runtime_invisible_parameter_annotations = NULL;
1680    int runtime_invisible_parameter_annotations_length = 0;
1681    u1* annotation_default = NULL;
1682    int annotation_default_length = 0;

1684    // Parse code and exceptions attribute
1685    u2 method_attributes_count = cfs->get_u2_fast();
1686    while (method_attributes_count--) {
1687      cfs->guarantee_more(6, CHECK_(nullHandle));  // method_attribute_name_index,
1688      u2 method_attribute_name_index = cfs->get_u2_fast();
1689      u4 method_attribute_length = cfs->get_u4_fast();
1690      check_property(
1691        valid_cp_range(method_attribute_name_index, cp_size) &&
1692          cp->tag_at(method_attribute_name_index).is_utf8(),
1693        "Invalid method attribute name index %u in class file %s",
1694        method_attribute_name_index, CHECK_(nullHandle));

1696      Symbol* method_attribute_name = cp->symbol_at(method_attribute_name_index);
1697      if (method_attribute_name == vmSymbols::tag_code()) {
```



```
1698        // Parse Code attribute
1699        if (_need_verify) {
1700          guarantee_property(!access_flags.is_native() && !access_flags.is_abstrac
1701                             "Code attribute in native or abstract methods in class f
1702                             CHECK_(nullHandle));
1703        }
1704        if (parsed_code_attribute) {
1705          classfile_parse_error("Multiple Code attributes in class file %s", CHECK
1706        }
1707        parsed_code_attribute = true;

1709        // Stack size, locals size, and code size
1710        if (_major_version == 45 && _minor_version <= 2) {
1711          cfs->guarantee_more(4, CHECK_(nullHandle));
1712          max_stack = cfs->get_u1_fast();
1713          max_locals = cfs->get_u1_fast();
1714          code_length = cfs->get_u2_fast();
1715        } else {
1716          cfs->guarantee_more(8, CHECK_(nullHandle));
1717          max_stack = cfs->get_u2_fast();
1718          max_locals = cfs->get_u2_fast();
1719          code_length = cfs->get_u4_fast();
1720        }
1721        if (_need_verify) {
1722          guarantee_property(args_size <= max_locals,
1723                             "Arguments can't fit into locals in class file %s", C
1724          guarantee_property(code_length > 0 && code_length <= MAX_CODE_SIZE,
1725                             "Invalid method Code length %u in class file %s",
1726                             code_length, CHECK_(nullHandle));
1727        }
1728        // Code pointer
1729        code_start = cfs->get_u1_buffer();
1730        assert(code_start != NULL, "null code start");
1731        cfs->guarantee_more(code_length, CHECK_(nullHandle));
1732        cfs->skip_u1_fast(code_length);

1734        // Exception handler table
1735        cfs->guarantee_more(2, CHECK_(nullHandle));  // exception_table_length
1736        exception_table_length = cfs->get_u2_fast();
1737        if (exception_table_length > 0) {
1738          exception_handlers =
1739              parse_exception_table(code_length, exception_table_length, cp, CHE
1740        }

1742        // Parse additional attributes in code attribute
1743        cfs->guarantee_more(2, CHECK_(nullHandle));  // code_attributes_count
1744        u2 code_attributes_count = cfs->get_u2_fast();

1746        unsigned int calculated_attribute_length = 0;

1748        if (_major_version > 45 || (_major_version == 45 && _minor_version > 2)) {
1749          calculated_attribute_length =
1750              sizeof(max_stack) + sizeof(max_locals) + sizeof(code_length);
1751        } else {
1752          // max_stack, locals and length are smaller in pre-version 45.2 classes
1753          calculated_attribute_length = sizeof(u1) + sizeof(u1) + sizeof(u2);
1754        }
1755        calculated_attribute_length +=
1756          code_length +
1757          sizeof(exception_table_length) +
1758          sizeof(code_attributes_count) +
1759          exception_table_length *
1760              ( sizeof(u2) +   // start_pc
1761                sizeof(u2) +   // end_pc
1762                sizeof(u2) +   // handler_pc
1763                sizeof(u2) );  // catch_type_index
```

```
1765       while (code_attributes_count--) {
1766         cfs->guarantee_more(6, CHECK_(nullHandle));  // code_attribute_name_inde
1767         u2 code_attribute_name_index = cfs->get_u2_fast();
1768         u4 code_attribute_length = cfs->get_u4_fast();
1769         calculated_attribute_length += code_attribute_length +
1770                                        sizeof(code_attribute_name_index) +
1771                                        sizeof(code_attribute_length);
1772         check_property(valid_cp_range(code_attribute_name_index, cp_size) &&
1773                        cp->tag_at(code_attribute_name_index).is_utf8(),
1774                        "Invalid code attribute name index %u in class file %s",
1775                        code_attribute_name_index,
1776                        CHECK_(nullHandle));
1777         if (LoadLineNumberTables &&
1778             cp->symbol_at(code_attribute_name_index) == vmSymbols::tag_line_numb
1779           // Parse and compress line number table
1780           parse_linenumber_table(code_attribute_length, code_length,
1781             &linenumber_table, CHECK_(nullHandle));

1783         } else if (LoadLocalVariableTables &&
1784                    cp->symbol_at(code_attribute_name_index) == vmSymbols::tag_lo
1785           // Parse local variable table
1786           if (!lvt_allocated) {
1787             localvariable_table_length = NEW_RESOURCE_ARRAY_IN_THREAD(
1788               THREAD, u2,  INITIAL_MAX_LVT_NUMBER);
1789             localvariable_table_start = NEW_RESOURCE_ARRAY_IN_THREAD(
1790               THREAD, u2*, INITIAL_MAX_LVT_NUMBER);
1791             localvariable_type_table_length = NEW_RESOURCE_ARRAY_IN_THREAD(
1792               THREAD, u2,  INITIAL_MAX_LVT_NUMBER);
1793             localvariable_type_table_start = NEW_RESOURCE_ARRAY_IN_THREAD(
1794               THREAD, u2*, INITIAL_MAX_LVT_NUMBER);
1795             lvt_allocated = true;
1796           }
1797           if (lvt_cnt == max_lvt_cnt) {
1798             max_lvt_cnt <<= 1;
1799             REALLOC_RESOURCE_ARRAY(u2, localvariable_table_length, lvt_cnt, max_
1800             REALLOC_RESOURCE_ARRAY(u2*, localvariable_table_start, lvt_cnt, max_
1801           }
1802           localvariable_table_start[lvt_cnt] =
1803             parse_localvariable_table(code_length,
1804                                       max_locals,
1805                                       code_attribute_length,
1806                                       cp,
1807                                       &localvariable_table_length[lvt_cnt],
1808                                       false,    // is not LVTT
1809                                       CHECK_(nullHandle));
1810           total_lvt_length += localvariable_table_length[lvt_cnt];
1811           lvt_cnt++;
1812         } else if (LoadLocalVariableTypeTables &&
1813                    _major_version >= JAVA_1_5_VERSION &&
1814                    cp->symbol_at(code_attribute_name_index) == vmSymbols::tag_lo
1815           if (!lvt_allocated) {
1816             localvariable_table_length = NEW_RESOURCE_ARRAY_IN_THREAD(
1817               THREAD, u2,  INITIAL_MAX_LVT_NUMBER);
1818             localvariable_table_start = NEW_RESOURCE_ARRAY_IN_THREAD(
1819               THREAD, u2*, INITIAL_MAX_LVT_NUMBER);
1820             localvariable_type_table_length = NEW_RESOURCE_ARRAY_IN_THREAD(
1821               THREAD, u2,  INITIAL_MAX_LVT_NUMBER);
1822             localvariable_type_table_start = NEW_RESOURCE_ARRAY_IN_THREAD(
1823               THREAD, u2*, INITIAL_MAX_LVT_NUMBER);
1824             lvt_allocated = true;
1825           }
1826           // Parse local variable type table
1827           if (lvtt_cnt == max_lvtt_cnt) {
1828             max_lvtt_cnt <<= 1;
1829             REALLOC_RESOURCE_ARRAY(u2, localvariable_type_table_length, lvtt_cnt
```



```
1830             REALLOC_RESOURCE_ARRAY(u2*, localvariable_type_table_start, lvtt_cnt
1831           }
1832           localvariable_type_table_start[lvtt_cnt] =
1833             parse_localvariable_table(code_length,
1834                                       max_locals,
1835                                       code_attribute_length,
1836                                       cp,
1837                                       &localvariable_type_table_length[lvtt_cnt]
1838                                       true,     // is LVTT
1839                                       CHECK_(nullHandle));
1840           lvtt_cnt++;
1841         } else if (UseSplitVerifier &&
1842                    _major_version >= Verifier::STACKMAP_ATTRIBUTE_MAJOR_VERSION
1843                    cp->symbol_at(code_attribute_name_index) == vmSymbols::tag_st
1844           // Stack map is only needed by the new verifier in JDK1.5.
1845           if (parsed_stackmap_attribute) {
1846             classfile_parse_error("Multiple StackMapTable attributes in class fi
1847           }
1848           typeArrayOop sm =
1849             parse_stackmap_table(code_attribute_length, CHECK_(nullHandle));
1850           stackmap_data = typeArrayHandle(THREAD, sm);
1851           parsed_stackmap_attribute = true;
1852         } else {
1853           // Skip unknown attributes
1854           cfs->skip_u1(code_attribute_length, CHECK_(nullHandle));
1855         }
1856       }
1857       // check method attribute length
1858       if (_need_verify) {
1859         guarantee_property(method_attribute_length == calculated_attribute_lengt
1860                            "Code segment has wrong length in class file %s", CHE
1861       }
1862     } else if (method_attribute_name == vmSymbols::tag_exceptions()) {
1863       // Parse Exceptions attribute
1864       if (parsed_checked_exceptions_attribute) {
1865         classfile_parse_error("Multiple Exceptions attributes in class file %s",
1866       }
1867       parsed_checked_exceptions_attribute = true;
1868       checked_exceptions_start =
1869             parse_checked_exceptions(&checked_exceptions_length,
1870                                      method_attribute_length,
1871                                      cp, CHECK_(nullHandle));
1872     } else if (method_attribute_name == vmSymbols::tag_synthetic()) {
1873       if (method_attribute_length != 0) {
1874         classfile_parse_error(
1875           "Invalid Synthetic method attribute length %u in class file %s",
1876           method_attribute_length, CHECK_(nullHandle));
1877       }
1878       // Should we check that there hasn't already been a synthetic attribute?
1879       access_flags.set_is_synthetic();
1880     } else if (method_attribute_name == vmSymbols::tag_deprecated()) { // 427612
1881       if (method_attribute_length != 0) {
1882         classfile_parse_error(
1883           "Invalid Deprecated method attribute length %u in class file %s",
1884           method_attribute_length, CHECK_(nullHandle));
1885       }
1886     } else if (_major_version >= JAVA_1_5_VERSION) {
1887       if (method_attribute_name == vmSymbols::tag_signature()) {
1888         if (method_attribute_length != 2) {
1889           classfile_parse_error(
1890             "Invalid Signature attribute length %u in class file %s",
1891             method_attribute_length, CHECK_(nullHandle));
1892         }
1893         cfs->guarantee_more(2, CHECK_(nullHandle));  // generic_signature_index
1894         generic_signature_index = cfs->get_u2_fast();
1895       } else if (method_attribute_name == vmSymbols::tag_runtime_visible_annotat
```

```
1896         runtime_visible_annotations_length = method_attribute_length;
1897         runtime_visible_annotations = cfs->get_u1_buffer();
1898         assert(runtime_visible_annotations != NULL, "null visible annotations");
1899         cfs->skip_u1(runtime_visible_annotations_length, CHECK_(nullHandle));
1900       } else if (PreserveAllAnnotations && method_attribute_name == vmSymbols::t
1901         runtime_invisible_annotations_length = method_attribute_length;
1902         runtime_invisible_annotations = cfs->get_u1_buffer();
1903         assert(runtime_invisible_annotations != NULL, "null invisible annotation
1904         cfs->skip_u1(runtime_invisible_annotations_length, CHECK_(nullHandle));
1905       } else if (method_attribute_name == vmSymbols::tag_runtime_visible_paramet
1906         runtime_visible_parameter_annotations_length = method_attribute_length;
1907         runtime_visible_parameter_annotations = cfs->get_u1_buffer();
1908         assert(runtime_visible_parameter_annotations != NULL, "null visible para
1909         cfs->skip_u1(runtime_visible_parameter_annotations_length, CHECK_(nullHa
1910       } else if (PreserveAllAnnotations && method_attribute_name == vmSymbols::t
1911         runtime_invisible_parameter_annotations_length = method_attribute_length
1912         runtime_invisible_parameter_annotations = cfs->get_u1_buffer();
1913         assert(runtime_invisible_parameter_annotations != NULL, "null invisible
1914         cfs->skip_u1(runtime_invisible_parameter_annotations_length, CHECK_(null
1915       } else if (method_attribute_name == vmSymbols::tag_annotation_default()) {
1916         annotation_default_length = method_attribute_length;
1917         annotation_default = cfs->get_u1_buffer();
1918         assert(annotation_default != NULL, "null annotation default");
1919         cfs->skip_u1(annotation_default_length, CHECK_(nullHandle));
1920       } else {
1921         // Skip unknown attributes
1922         cfs->skip_u1(method_attribute_length, CHECK_(nullHandle));
1923       }
1924     } else {
1925       // Skip unknown attributes
1926       cfs->skip_u1(method_attribute_length, CHECK_(nullHandle));
1927     }
1928   }

1930   if (linenumber_table != NULL) {
1931     linenumber_table->write_terminator();
1932     linenumber_table_length = linenumber_table->position();
1933   }

1935   // Make sure there's at least one Code attribute in non-native/non-abstract me
1936   if (_need_verify) {
1937     guarantee_property(access_flags.is_native() || access_flags.is_abstract() ||
1938                        "Absent Code attribute in method that is not native or abs
1939   }

1941   // All sizing information for a methodOop is finally available, now create it
1942   methodOop m_oop  = oopFactory::new_method(code_length, access_flags, linenumbe
1943                                             total_lvt_length, checked_exceptions
1944                                             oopDesc::IsSafeConc, CHECK_(nullHand
1945   methodHandle m (THREAD, m_oop);

1947   ClassLoadingService::add_class_method_size(m_oop->size()*HeapWordSize);

1949   // Fill in information from fixed part (access_flags already set)
1950   m->set_constants(cp());
1951   m->set_name_index(name_index);
1952   m->set_signature_index(signature_index);
1953   m->set_generic_signature_index(generic_signature_index);
1954 #ifdef CC_INTERP
1955   // hmm is there a gc issue here??
1956   ResultTypeFinder rtf(cp->symbol_at(signature_index));
1957   m->set_result_index(rtf.type());
1958 #endif

1960   if (args_size >= 0) {
1961     m->set_size_of_parameters(args_size);
```



```
1962   } else {
1963     m->compute_size_of_parameters(THREAD);
1964   }
1965 #ifdef ASSERT
1966   if (args_size >= 0) {
1967     m->compute_size_of_parameters(THREAD);
1968     assert(args_size == m->size_of_parameters(), "");
1969   }
1970 #endif

1972   // Fill in code attribute information
1973   m->set_max_stack(max_stack);
1974   m->set_max_locals(max_locals);
1975   m->constMethod()->set_stackmap_data(stackmap_data());

1977   /**
1978    * The exception_table field is the flag used to indicate
1979    * that the methodOop and it's associated constMethodOop are partially
1980    * initialized and thus are exempt from pre/post GC verification.  Once
1981    * the field is set, the oops are considered fully initialized so make
1982    * sure that the oops can pass verification when this field is set.
1983    */
1984   m->set_exception_table(exception_handlers());

1986   // Copy byte codes
1987   m->set_code(code_start);

1989   // Copy line number table
1990   if (linenumber_table != NULL) {
1991     memcpy(m->compressed_linenumber_table(),
1992            linenumber_table->buffer(), linenumber_table_length);
1993   }

1995   // Copy checked exceptions
1996   if (checked_exceptions_length > 0) {
1997     int size = checked_exceptions_length * sizeof(CheckedExceptionElement) / siz
1998     copy_u2_with_conversion((u2*) m->checked_exceptions_start(), checked_excepti
1999   }

2001   /* Copy class file LVT's/LVTT's into the HotSpot internal LVT.
2002    *
2003    * Rules for LVT's and LVTT's are:
2004    *   - There can be any number of LVT's and LVTT's.
2005    *   - If there are n LVT's, it is the same as if there was just
2006    *     one LVT containing all the entries from the n LVT's.
2007    *   - There may be no more than one LVT entry per local variable.
2008    *     Two LVT entries are 'equal' if these fields are the same:
2009    *       start_pc, length, name, slot
2010    *   - There may be no more than one LVTT entry per each LVT entry.
2011    *     Each LVTT entry has to match some LVT entry.
2012    *   - HotSpot internal LVT keeps natural ordering of class file LVT entries.
2013    */
2014   if (total_lvt_length > 0) {
2015     int tbl_no, idx;

2017     promoted_flags->set_has_localvariable_table();

2019     LVT_Hash** lvt_Hash = NEW_RESOURCE_ARRAY(LVT_Hash*, HASH_ROW_SIZE);
2020     initialize_hashtable(lvt_Hash);

2022     // To fill LocalVariableTable in
2023     Classfile_LVT_Element*  cf_lvt;
2024     LocalVariableTableElement* lvt = m->localvariable_table_start();

2026     for (tbl_no = 0; tbl_no < lvt_cnt; tbl_no++) {
2027       cf_lvt = (Classfile_LVT_Element *) localvariable_table_start[tbl_no];
```

```
2028       for (idx = 0; idx < localvariable_table_length[tbl_no]; idx++, lvt++) {
2029         copy_lvt_element(&cf_lvt[idx], lvt);
2030         // If no duplicates, add LVT elem in hashtable lvt_Hash.
2031         if (LVT_put_after_lookup(lvt, lvt_Hash) == false
2032           && _need_verify
2033           && _major_version >= JAVA_1_5_VERSION ) {
2034           clear_hashtable(lvt_Hash);
2035           classfile_parse_error("Duplicated LocalVariableTable attribute "
2036                                 "entry for '%s' in class file %s",
2037                                  cp->symbol_at(lvt->name_cp_index)->as_utf8(),
2038                                  CHECK_(nullHandle));
2039         }
2040       }
2041     }

2043     // To merge LocalVariableTable and LocalVariableTypeTable
2044     Classfile_LVT_Element* cf_lvtt;
2045     LocalVariableTableElement lvtt_elem;

2047     for (tbl_no = 0; tbl_no < lvtt_cnt; tbl_no++) {
2048       cf_lvtt = (Classfile_LVT_Element *) localvariable_type_table_start[tbl_no]
2049       for (idx = 0; idx < localvariable_type_table_length[tbl_no]; idx++) {
2050         copy_lvt_element(&cf_lvtt[idx], &lvtt_elem);
2051         int index = hash(&lvtt_elem);
2052         LVT_Hash* entry = LVT_lookup(&lvtt_elem, index, lvt_Hash);
2053         if (entry == NULL) {
2054           if (_need_verify) {
2055             clear_hashtable(lvt_Hash);
2056             classfile_parse_error("LVTT entry for '%s' in class file %s "
2057                                   "does not match any LVT entry",
2058                                    cp->symbol_at(lvtt_elem.name_cp_index)->as_ut
2059                                    CHECK_(nullHandle));
2060           }
2061         } else if (entry->_elem->signature_cp_index != 0 && _need_verify) {
2062           clear_hashtable(lvt_Hash);
2063           classfile_parse_error("Duplicated LocalVariableTypeTable attribute "
2064                                 "entry for '%s' in class file %s",
2065                                  cp->symbol_at(lvtt_elem.name_cp_index)->as_utf8
2066                                  CHECK_(nullHandle));
2067         } else {
2068           // to add generic signatures into LocalVariableTable
2069           entry->_elem->signature_cp_index = lvtt_elem.descriptor_cp_index;
2070         }
2071       }
2072     }
2073     clear_hashtable(lvt_Hash);
2074   }

2076   *method_annotations = assemble_annotations(runtime_visible_annotations,
2077                                              runtime_visible_annotations_length,
2078                                              runtime_invisible_annotations,
2079                                              runtime_invisible_annotations_lengt
2080                                              CHECK_(nullHandle));
2081   *method_parameter_annotations = assemble_annotations(runtime_visible_parameter
2082                                                        runtime_visible_parameter
2083                                                        runtime_invisible_paramet
2084                                                        runtime_invisible_paramet
2085                                                        CHECK_(nullHandle));
2086   *method_default_annotations = assemble_annotations(annotation_default,
2087                                                      annotation_default_length,
2088                                                      NULL,
2089                                                      0,
2090                                                      CHECK_(nullHandle));

2092   if (name == vmSymbols::finalize_method_name() &&
2093       signature == vmSymbols::void_method_signature()) {
```



```
2094     if (m->is_empty_method()) {
2095       _has_empty_finalizer = true;
2096     } else {
2097       _has_finalizer = true;
2098     }
2099   }
2100   if (name == vmSymbols::object_initializer_name() &&
2101       signature == vmSymbols::void_method_signature() &&
2102       m->is_vanilla_constructor()) {
2103     _has_vanilla_constructor = true;
2104   }

2106   if (EnableInvokeDynamic && (m->is_method_handle_invoke() ||
2106   if (EnableMethodHandles && (m->is_method_handle_invoke() ||
2107                               m->is_method_handle_adapter())) {
2108     THROW_MSG_(vmSymbols::java_lang_VirtualMachineError(),
2109                "Method handle invokers must be defined internally to the VM", nu
2110   }

2112   return m;
2113 }
______unchanged_portion_omitted_


2763 // Force MethodHandle.vmentry to be an unmanaged pointer.
2764 // There is no way for a classfile to express this, so we must help it.
2765 void ClassFileParser::java_lang_invoke_MethodHandle_fix_pre(constantPoolHandle c
2766                                                             typeArrayHandle fields,
2767                                                             FieldAllocationCount *fac_pt
2768                                                             TRAPS) {
2769   // Add fake fields for java.lang.invoke.MethodHandle instances
2770   //
2771   // This is not particularly nice, but since there is no way to express
2772   // a native wordSize field in Java, we must do it at this level.

2774   if (!EnableInvokeDynamic)  return;
2774   if (!EnableMethodHandles)  return;

2776   int word_sig_index = 0;
2777   const int cp_size = cp->length();
2778   for (int index = 1; index < cp_size; index++) {
2779     if (cp->tag_at(index).is_utf8() &&
2780         cp->symbol_at(index) == vmSymbols::machine_word_signature()) {
2781       word_sig_index = index;
2782       break;
2783     }
2784   }

2786   if (AllowTransitionalJSR292 && word_sig_index == 0)  return;
2787   if (word_sig_index == 0)
2788     THROW_MSG(vmSymbols::java_lang_VirtualMachineError(),
2789               "missing I or J signature (for vmentry) in java.lang.invoke.Method

2791   // Find vmentry field and change the signature.
2792   bool found_vmentry = false;
2793   for (int i = 0; i < fields->length(); i += instanceKlass::next_offset) {
2794     int name_index = fields->ushort_at(i + instanceKlass::name_index_offset);
2795     int sig_index  = fields->ushort_at(i + instanceKlass::signature_index_offset
2796     int acc_flags  = fields->ushort_at(i + instanceKlass::access_flags_offset);
2797     Symbol* f_name = cp->symbol_at(name_index);
2798     Symbol* f_sig  = cp->symbol_at(sig_index);
2799     if (f_name == vmSymbols::vmentry_name() && (acc_flags & JVM_ACC_STATIC) == 0
2800       if (f_sig == vmSymbols::machine_word_signature()) {
2801         // If the signature of vmentry is already changed, we're done.
2802         found_vmentry = true;
2803         break;
```

```
2804       }
2805       else if (f_sig == vmSymbols::byte_signature()) {
2806         // Adjust the field type from byte to an unmanaged pointer.
2807         assert(fac_ptr->nonstatic_byte_count > 0, "");
2808         fac_ptr->nonstatic_byte_count -= 1;

2810         fields->ushort_at_put(i + instanceKlass::signature_index_offset, word_si
2811         assert(wordSize == longSize || wordSize == jintSize, "ILP32 or LP64");
2812         if (wordSize == longSize)  fac_ptr->nonstatic_double_count += 1;
2813         else                       fac_ptr->nonstatic_word_count   += 1;

2815         FieldAllocationType atype = (FieldAllocationType) fields->ushort_at(i +
2816         assert(atype == NONSTATIC_BYTE, "");
2817         FieldAllocationType new_atype = (wordSize == longSize) ? NONSTATIC_DOUBL
2818         fields->ushort_at_put(i + instanceKlass::low_offset, new_atype);

2820         found_vmentry = true;
2821         break;
2822       }
2823     }
2824   }

2826   if (AllowTransitionalJSR292 && !found_vmentry)  return;
2827   if (!found_vmentry)
2828     THROW_MSG(vmSymbols::java_lang_VirtualMachineError(),
2829               "missing vmentry byte field in java.lang.invoke.MethodHandle");
2830 }


2833 instanceKlassHandle ClassFileParser::parseClassFile(Symbol* name,
2834                                                     Handle class_loader,
2835                                                     Handle protection_domain,
2836                                                     KlassHandle host_klass,
2837                                                     GrowableArray<Handle>* cp_pa
2838                                                     TempNewSymbol& parsed_name,
2839                                                     bool verify,
2840                                                     TRAPS) {
2841   // So that JVMTI can cache class file in the state before retransformable agen
2842   // have modified it
2843   unsigned char *cached_class_file_bytes = NULL;
2844   jint cached_class_file_length;

2846   ClassFileStream* cfs = stream();
2847   // Timing
2848   assert(THREAD->is_Java_thread(), "must be a JavaThread");
2849   JavaThread* jt = (JavaThread*) THREAD;

2851   PerfClassTraceTime ctimer(ClassLoader::perf_class_parse_time(),
2852                             ClassLoader::perf_class_parse_selftime(),
2853                             NULL,
2854                             jt->get_thread_stat()->perf_recursion_counts_addr(),
2855                             jt->get_thread_stat()->perf_timers_addr(),
2856                             PerfClassTraceTime::PARSE_CLASS);

2858   _has_finalizer = _has_empty_finalizer = _has_vanilla_constructor = false;
2859   _max_bootstrap_specifier_index = -1;

2861   if (JvmtiExport::should_post_class_file_load_hook()) {
2862     unsigned char* ptr = cfs->buffer();
2863     unsigned char* end_ptr = cfs->buffer() + cfs->length();

2865     JvmtiExport::post_class_file_load_hook(name, class_loader, protection_domain
2866                                            &ptr, &end_ptr,
2867                                            &cached_class_file_bytes,
2868                                            &cached_class_file_length);
```



```
2870     if (ptr != cfs->buffer()) {
2871       // JVMTI agent has modified class file data.
2872       // Set new class file stream using JVMTI agent modified
2873       // class file data.
2874       cfs = new ClassFileStream(ptr, end_ptr - ptr, cfs->source());
2875       set_stream(cfs);
2876     }
2877   }

2879   _host_klass = host_klass;
2880   _cp_patches = cp_patches;

2882   instanceKlassHandle nullHandle;

2884   // Figure out whether we can skip format checking (matching classic VM behavio
2885   _need_verify = Verifier::should_verify_for(class_loader(), verify);

2887   // Set the verify flag in stream
2888   cfs->set_verify(_need_verify);

2890   // Save the class file name for easier error message printing.
2891   _class_name = (name != NULL) ? name : vmSymbols::unknown_class_name();

2893   cfs->guarantee_more(8, CHECK_(nullHandle));  // magic, major, minor
2894   // Magic value
2895   u4 magic = cfs->get_u4_fast();
2896   guarantee_property(magic == JAVA_CLASSFILE_MAGIC,
2897                      "Incompatible magic value %u in class file %s",
2898                      magic, CHECK_(nullHandle));

2900   // Version numbers
2901   u2 minor_version = cfs->get_u2_fast();
2902   u2 major_version = cfs->get_u2_fast();

2904   // Check version numbers - we check this even with verifier off
2905   if (!is_supported_version(major_version, minor_version)) {
2906     if (name == NULL) {
2907       Exceptions::fthrow(
2908         THREAD_AND_LOCATION,
2909         vmSymbols::java_lang_UnsupportedClassVersionError(),
2910         "Unsupported major.minor version %u.%u",
2911         major_version,
2912         minor_version);
2913     } else {
2914       ResourceMark rm(THREAD);
2915       Exceptions::fthrow(
2916         THREAD_AND_LOCATION,
2917         vmSymbols::java_lang_UnsupportedClassVersionError(),
2918         "%s : Unsupported major.minor version %u.%u",
2919         name->as_C_string(),
2920         major_version,
2921         minor_version);
2922     }
2923     return nullHandle;
2924   }

2926   _major_version = major_version;
2927   _minor_version = minor_version;


2930   // Check if verification needs to be relaxed for this class file
2931   // Do not restrict it to jdk1.0 or jdk1.1 to maintain backward compatibility (
2932   _relax_verify = Verifier::relax_verify_for(class_loader());

2934   // Constant pool
2935   constantPoolHandle cp = parse_constant_pool(CHECK_(nullHandle));
```

```
2936   ConstantPoolCleaner error_handler(cp); // set constant pool to be cleaned up.

2938   int cp_size = cp->length();

2940   cfs->guarantee_more(8, CHECK_(nullHandle));  // flags, this_class, super_class

2942   // Access flags
2943   AccessFlags access_flags;
2944   jint flags = cfs->get_u2_fast() & JVM_RECOGNIZED_CLASS_MODIFIERS;

2946   if ((flags & JVM_ACC_INTERFACE) && _major_version < JAVA_6_VERSION) {
2947     // Set abstract bit for old class files for backward compatibility
2948     flags |= JVM_ACC_ABSTRACT;
2949   }
2950   verify_legal_class_modifiers(flags, CHECK_(nullHandle));
2951   access_flags.set_flags(flags);

2953   // This class and superclass
2954   instanceKlassHandle super_klass;
2955   u2 this_class_index = cfs->get_u2_fast();
2956   check_property(
2957     valid_cp_range(this_class_index, cp_size) &&
2958       cp->tag_at(this_class_index).is_unresolved_klass(),
2959     "Invalid this class index %u in constant pool in class file %s",
2960     this_class_index, CHECK_(nullHandle));

2962   Symbol*  class_name  = cp->unresolved_klass_at(this_class_index);
2963   assert(class_name != NULL, "class_name can't be null");

2965   // It's important to set parsed_name *before* resolving the super class.
2966   // (it's used for cleanup by the caller if parsing fails)
2967   parsed_name = class_name;
2968   // parsed_name is returned and can be used if there's an error, so add to
2969   // its reference count.  Caller will decrement the refcount.
2970   parsed_name->increment_refcount();

2972   // Update _class_name which could be null previously to be class_name
2973   _class_name = class_name;

2975   // Don't need to check whether this class name is legal or not.
2976   // It has been checked when constant pool is parsed.
2977   // However, make sure it is not an array type.
2978   if (_need_verify) {
2979     guarantee_property(class_name->byte_at(0) != JVM_SIGNATURE_ARRAY,
2980                        "Bad class name in class file %s",
2981                        CHECK_(nullHandle));
2982   }

2984   klassOop preserve_this_klass;   // for storing result across HandleMark

2986   // release all handles when parsing is done
2987   { HandleMark hm(THREAD);

2989     // Checks if name in class file matches requested name
2990     if (name != NULL && class_name != name) {
2991       ResourceMark rm(THREAD);
2992       Exceptions::fthrow(
2993         THREAD_AND_LOCATION,
2994         vmSymbols::java_lang_NoClassDefFoundError(),
2995         "%s (wrong name: %s)",
2996         name->as_C_string(),
2997         class_name->as_C_string()
2998       );
2999       return nullHandle;
3000     }
```



```
3002     if (TraceClassLoadingPreorder) {
3003       tty->print("[Loading %s", name->as_klass_external_name());
3004       if (cfs->source() != NULL) tty->print(" from %s", cfs->source());
3005       tty->print_cr("]");
3006     }

3008     u2 super_class_index = cfs->get_u2_fast();
3009     if (super_class_index == 0) {
3010       check_property(class_name == vmSymbols::java_lang_Object(),
3011                      "Invalid superclass index %u in class file %s",
3012                      super_class_index,
3013                      CHECK_(nullHandle));
3014     } else {
3015       check_property(valid_cp_range(super_class_index, cp_size) &&
3016                      is_klass_reference(cp, super_class_index),
3017                      "Invalid superclass index %u in class file %s",
3018                      super_class_index,
3019                      CHECK_(nullHandle));
3020       // The class name should be legal because it is checked when parsing const
3021       // However, make sure it is not an array type.
3022       bool is_array = false;
3023       if (cp->tag_at(super_class_index).is_klass()) {
3024         super_klass = instanceKlassHandle(THREAD, cp->resolved_klass_at(super_cl
3025         if (_need_verify)
3026           is_array = super_klass->oop_is_array();
3027       } else if (_need_verify) {
3028         is_array = (cp->unresolved_klass_at(super_class_index)->byte_at(0) == JV
3029       }
3030       if (_need_verify) {
3031         guarantee_property(!is_array,
3032                            "Bad superclass name in class file %s", CHECK_(nullHan
3033       }
3034     }

3036     // Interfaces
3037     u2 itfs_len = cfs->get_u2_fast();
3038     objArrayHandle local_interfaces;
3039     if (itfs_len == 0) {
3040       local_interfaces = objArrayHandle(THREAD, Universe::the_empty_system_obj_a
3041     } else {
3042       local_interfaces = parse_interfaces(cp, itfs_len, class_loader, protection
3043     }

3045     // Fields (offsets are filled in later)
3046     struct FieldAllocationCount fac = {0,0,0,0,0,0,0,0,0,0};
3047     objArrayHandle fields_annotations;
3048     typeArrayHandle fields = parse_fields(cp, access_flags.is_interface(), &fac,
3049     // Methods
3050     bool has_final_method = false;
3051     AccessFlags promoted_flags;
3052     promoted_flags.set_flags(0);
3053     // These need to be oop pointers because they are allocated lazily
3054     // inside parse_methods inside a nested HandleMark
3055     objArrayOop methods_annotations_oop = NULL;
3056     objArrayOop methods_parameter_annotations_oop = NULL;
3057     objArrayOop methods_default_annotations_oop = NULL;
3058     objArrayHandle methods = parse_methods(cp, access_flags.is_interface(),
3059                                            &promoted_flags,
3060                                            &has_final_method,
3061                                            &methods_annotations_oop,
3062                                            &methods_parameter_annotations_oop,
3063                                            &methods_default_annotations_oop,
3064                                            CHECK_(nullHandle));

3066     objArrayHandle methods_annotations(THREAD, methods_annotations_oop);
3067     objArrayHandle methods_parameter_annotations(THREAD, methods_parameter_annot
```

```
3068     objArrayHandle methods_default_annotations(THREAD, methods_default_annotatio

3070     // We check super class after class file is parsed and format is checked
3071     if (super_class_index > 0 && super_klass.is_null()) {
3072       Symbol*  sk  = cp->klass_name_at(super_class_index);
3073       if (access_flags.is_interface()) {
3074         // Before attempting to resolve the superclass, check for class format
3075         // errors not checked yet.
3076         guarantee_property(sk == vmSymbols::java_lang_Object(),
3077                            "Interfaces must have java.lang.Object as superclass
3078                            CHECK_(nullHandle));
3079       }
3080       klassOop k = SystemDictionary::resolve_super_or_fail(class_name,
3081                                                            sk,
3082                                                            class_loader,
3083                                                            protection_domain,
3084                                                            true,
3085                                                            CHECK_(nullHandle));

3087       KlassHandle kh (THREAD, k);
3088       super_klass = instanceKlassHandle(THREAD, kh());
3089       if (LinkWellKnownClasses)  // my super class is well known to me
3090         cp->klass_at_put(super_class_index, super_klass()); // eagerly resolve
3091     }
3092     if (super_klass.not_null()) {
3093       if (super_klass->is_interface()) {
3094         ResourceMark rm(THREAD);
3095         Exceptions::fthrow(
3096           THREAD_AND_LOCATION,
3097           vmSymbols::java_lang_IncompatibleClassChangeError(),
3098           "class %s has interface %s as super class",
3099           class_name->as_klass_external_name(),
3100           super_klass->external_name()
3101         );
3102         return nullHandle;
3103       }
3104       // Make sure super class is not final
3105       if (super_klass->is_final()) {
3106         THROW_MSG_(vmSymbols::java_lang_VerifyError(), "Cannot inherit from fina
3107       }
3108     }

3110     // Compute the transitive list of all unique interfaces implemented by this
3111     objArrayHandle transitive_interfaces = compute_transitive_interfaces(super_k

3113     // sort methods
3114     typeArrayHandle method_ordering = sort_methods(methods,
3115                                                    methods_annotations,
3116                                                    methods_parameter_annotations
3117                                                    methods_default_annotations,
3118                                                    CHECK_(nullHandle));

3120     // promote flags from parse_methods() to the klass' flags
3121     access_flags.add_promoted_flags(promoted_flags.as_int());

3123     // Size of Java vtable (in words)
3124     int vtable_size = 0;
3125     int itable_size = 0;
3126     int num_miranda_methods = 0;

3128     klassVtable::compute_vtable_size_and_num_mirandas(vtable_size,
3129                                                       num_miranda_methods,
3130                                                       super_klass(),
3131                                                       methods(),
3132                                                       access_flags,
3133                                                       class_loader,
```



```
3134                                                       class_name,
3135                                                       local_interfaces(),
3136                                                       CHECK_(nullHandle));

3138     // Size of Java itable (in words)
3139     itable_size = access_flags.is_interface() ? 0 : klassItable::compute_itable_

3141     // Field size and offset computation
3142     int nonstatic_field_size = super_klass() == NULL ? 0 : super_klass->nonstati
3143 #ifndef PRODUCT
3144     int orig_nonstatic_field_size = 0;
3145 #endif
3146     int static_field_size = 0;
3147     int next_static_oop_offset;
3148     int next_static_double_offset;
3149     int next_static_word_offset;
3150     int next_static_short_offset;
3151     int next_static_byte_offset;
3152     int next_static_type_offset;
3153     int next_nonstatic_oop_offset;
3154     int next_nonstatic_double_offset;
3155     int next_nonstatic_word_offset;
3156     int next_nonstatic_short_offset;
3157     int next_nonstatic_byte_offset;
3158     int next_nonstatic_type_offset;
3159     int first_nonstatic_oop_offset;
3160     int first_nonstatic_field_offset;
3161     int next_nonstatic_field_offset;

3163     // Calculate the starting byte offsets
3164     next_static_oop_offset      = instanceMirrorKlass::offset_of_static_fields()
3165     next_static_double_offset   = next_static_oop_offset +
3166                                   (fac.static_oop_count * heapOopSize);
3167     if ( fac.static_double_count &&
3168          (Universe::field_type_should_be_aligned(T_DOUBLE) ||
3169           Universe::field_type_should_be_aligned(T_LONG)) ) {
3170       next_static_double_offset = align_size_up(next_static_double_offset, Bytes
3171     }

3173     next_static_word_offset     = next_static_double_offset +
3174                                   (fac.static_double_count * BytesPerLong);
3175     next_static_short_offset    = next_static_word_offset +
3176                                   (fac.static_word_count * BytesPerInt);
3177     next_static_byte_offset     = next_static_short_offset +
3178                                   (fac.static_short_count * BytesPerShort);
3179     next_static_type_offset     = align_size_up((next_static_byte_offset +
3180                                   fac.static_byte_count ), wordSize );
3181     static_field_size           = (next_static_type_offset -
3182                                   next_static_oop_offset) / wordSize;

3184     // Add fake fields for java.lang.Class instances (also see below)
3185     if (class_name == vmSymbols::java_lang_Class() && class_loader.is_null()) {
3186       java_lang_Class_fix_pre(&nonstatic_field_size, &fac);
3187     }

3189     first_nonstatic_field_offset = instanceOopDesc::base_offset_in_bytes() +
3190                                    nonstatic_field_size * heapOopSize;
3191     next_nonstatic_field_offset = first_nonstatic_field_offset;

3193     // adjust the vmentry field declaration in java.lang.invoke.MethodHandle
3194     if (EnableInvokeDynamic && class_name == vmSymbols::java_lang_invoke_MethodH
3194     if (EnableMethodHandles && class_name == vmSymbols::java_lang_invoke_MethodH
3195       java_lang_invoke_MethodHandle_fix_pre(cp, fields, &fac, CHECK_(nullHandle)
3196     }
3197     if (AllowTransitionalJSR292 &&
3198         EnableInvokeDynamic && class_name == vmSymbols::java_dyn_MethodHandle()
```

```
3198          EnableMethodHandles && class_name == vmSymbols::java_dyn_MethodHandle()
3199        java_lang_invoke_MethodHandle_fix_pre(cp, fields, &fac, CHECK_(nullHandle)
3200      }
3201      if (AllowTransitionalJSR292 &&
3202          EnableInvokeDynamic && class_name == vmSymbols::sun_dyn_MethodHandleImpl
3202          EnableMethodHandles && class_name == vmSymbols::sun_dyn_MethodHandleImpl
3203        // allow vmentry field in MethodHandleImpl also
3204        java_lang_invoke_MethodHandle_fix_pre(cp, fields, &fac, CHECK_(nullHandle)
3205      }

3207      // Add a fake "discovered" field if it is not present
3208      // for compatibility with earlier jdk's.
3209      if (class_name == vmSymbols::java_lang_ref_Reference()
3210        && class_loader.is_null()) {
3211        java_lang_ref_Reference_fix_pre(&fields, cp, &fac, CHECK_(nullHandle));
3212      }
3213      // end of "discovered" field compactibility fix

3215      unsigned int nonstatic_double_count = fac.nonstatic_double_count;
3216      unsigned int nonstatic_word_count   = fac.nonstatic_word_count;
3217      unsigned int nonstatic_short_count  = fac.nonstatic_short_count;
3218      unsigned int nonstatic_byte_count   = fac.nonstatic_byte_count;
3219      unsigned int nonstatic_oop_count    = fac.nonstatic_oop_count;

3221      bool super_has_nonstatic_fields =
3222              (super_klass() != NULL && super_klass->has_nonstatic_fields());
3223      bool has_nonstatic_fields  =  super_has_nonstatic_fields ||
3224              ((nonstatic_double_count + nonstatic_word_count +
3225                nonstatic_short_count + nonstatic_byte_count +
3226                nonstatic_oop_count) != 0);


3229      // Prepare list of oops for oop map generation.
3230      int* nonstatic_oop_offsets;
3231      unsigned int* nonstatic_oop_counts;
3232      unsigned int nonstatic_oop_map_count = 0;

3234      nonstatic_oop_offsets = NEW_RESOURCE_ARRAY_IN_THREAD(
3235                THREAD, int, nonstatic_oop_count + 1);
3236      nonstatic_oop_counts  = NEW_RESOURCE_ARRAY_IN_THREAD(
3237                THREAD, unsigned int, nonstatic_oop_count + 1);

3239      // Add fake fields for java.lang.Class instances (also see above).
3240      // FieldsAllocationStyle and CompactFields values will be reset to default.
3241      if(class_name == vmSymbols::java_lang_Class() && class_loader.is_null()) {
3242        java_lang_Class_fix_post(&next_nonstatic_field_offset);
3243        nonstatic_oop_offsets[0] = first_nonstatic_field_offset;
3244        const uint fake_oop_count = (next_nonstatic_field_offset -
3245                                     first_nonstatic_field_offset) / heapOopSize;
3246        nonstatic_oop_counts[0] = fake_oop_count;
3247        nonstatic_oop_map_count = 1;
3248        nonstatic_oop_count -= fake_oop_count;
3249        first_nonstatic_oop_offset = first_nonstatic_field_offset;
3250      } else {
3251        first_nonstatic_oop_offset = 0; // will be set for first oop field
3252      }

3254  #ifndef PRODUCT
3255      if( PrintCompactFieldsSavings ) {
3256        next_nonstatic_double_offset = next_nonstatic_field_offset +
3257                                       (nonstatic_oop_count * heapOopSize);
3258        if ( nonstatic_double_count > 0 ) {
3259          next_nonstatic_double_offset = align_size_up(next_nonstatic_double_offse
3260        }
3261        next_nonstatic_word_offset  = next_nonstatic_double_offset +
3262                                      (nonstatic_double_count * BytesPerLong);
```



```
3263        next_nonstatic_short_offset = next_nonstatic_word_offset +
3264                                      (nonstatic_word_count * BytesPerInt);
3265        next_nonstatic_byte_offset  = next_nonstatic_short_offset +
3266                                      (nonstatic_short_count * BytesPerShort);
3267        next_nonstatic_type_offset  = align_size_up((next_nonstatic_byte_offset +
3268                                      nonstatic_byte_count ), heapOopSize );
3269        orig_nonstatic_field_size   = nonstatic_field_size +
3270        ((next_nonstatic_type_offset - first_nonstatic_field_offset)/heapOopSize);
3271      }
3272  #endif
3273      bool compact_fields   = CompactFields;
3274      int  allocation_style = FieldsAllocationStyle;
3275      if( allocation_style < 0 || allocation_style > 2 ) { // Out of range?
3276        assert(false, "0 <= FieldsAllocationStyle <= 2");
3277        allocation_style = 1; // Optimistic
3278      }

3280      // The next classes have predefined hard-coded fields offsets
3281      // (see in JavaClasses::compute_hard_coded_offsets()).
3282      // Use default fields allocation order for them.
3283      if( (allocation_style != 0 || compact_fields ) && class_loader.is_null() &&
3284          (class_name == vmSymbols::java_lang_AssertionStatusDirectives() ||
3285           class_name == vmSymbols::java_lang_Class() ||
3286           class_name == vmSymbols::java_lang_ClassLoader() ||
3287           class_name == vmSymbols::java_lang_ref_Reference() ||
3288           class_name == vmSymbols::java_lang_ref_SoftReference() ||
3289           class_name == vmSymbols::java_lang_StackTraceElement() ||
3290           class_name == vmSymbols::java_lang_String() ||
3291           class_name == vmSymbols::java_lang_Throwable() ||
3292           class_name == vmSymbols::java_lang_Boolean() ||
3293           class_name == vmSymbols::java_lang_Character() ||
3294           class_name == vmSymbols::java_lang_Float() ||
3295           class_name == vmSymbols::java_lang_Double() ||
3296           class_name == vmSymbols::java_lang_Byte() ||
3297           class_name == vmSymbols::java_lang_Short() ||
3298           class_name == vmSymbols::java_lang_Integer() ||
3299           class_name == vmSymbols::java_lang_Long())) {
3300        allocation_style = 0;     // Allocate oops first
3301        compact_fields   = false; // Don't compact fields
3302      }

3304      if( allocation_style == 0 ) {
3305        // Fields order: oops, longs/doubles, ints, shorts/chars, bytes
3306        next_nonstatic_oop_offset    = next_nonstatic_field_offset;
3307        next_nonstatic_double_offset = next_nonstatic_oop_offset +
3308                                       (nonstatic_oop_count * heapOopSize);
3309      } else if( allocation_style == 1 ) {
3310        // Fields order: longs/doubles, ints, shorts/chars, bytes, oops
3311        next_nonstatic_double_offset = next_nonstatic_field_offset;
3312      } else if( allocation_style == 2 ) {
3313        // Fields allocation: oops fields in super and sub classes are together.
3314        if( nonstatic_field_size > 0 && super_klass() != NULL &&
3315            super_klass->nonstatic_oop_map_size() > 0 ) {
3316          int map_size = super_klass->nonstatic_oop_map_size();
3317          OopMapBlock* first_map = super_klass->start_of_nonstatic_oop_maps();
3318          OopMapBlock* last_map = first_map + map_size - 1;
3319          int next_offset = last_map->offset() + (last_map->count() * heapOopSize)
3320          if (next_offset == next_nonstatic_field_offset) {
3321            allocation_style = 0;   // allocate oops first
3322            next_nonstatic_oop_offset    = next_nonstatic_field_offset;
3323            next_nonstatic_double_offset = next_nonstatic_oop_offset +
3324                                           (nonstatic_oop_count * heapOopSize);
3325          }
3326        }
3327        if( allocation_style == 2 ) {
3328          allocation_style = 1;     // allocate oops last
```

```
3329         next_nonstatic_double_offset = next_nonstatic_field_offset;
3330       }
3331     } else {
3332       ShouldNotReachHere();
3333     }

3335     int nonstatic_oop_space_count   = 0;
3336     int nonstatic_word_space_count  = 0;
3337     int nonstatic_short_space_count = 0;
3338     int nonstatic_byte_space_count  = 0;
3339     int nonstatic_oop_space_offset;
3340     int nonstatic_word_space_offset;
3341     int nonstatic_short_space_offset;
3342     int nonstatic_byte_space_offset;

3344     if( nonstatic_double_count > 0 ) {
3345       int offset = next_nonstatic_double_offset;
3346       next_nonstatic_double_offset = align_size_up(offset, BytesPerLong);
3347       if( compact_fields && offset != next_nonstatic_double_offset ) {
3348         // Allocate available fields into the gap before double field.
3349         int length = next_nonstatic_double_offset - offset;
3350         assert(length == BytesPerInt, "");
3351         nonstatic_word_space_offset = offset;
3352         if( nonstatic_word_count > 0 ) {
3353           nonstatic_word_count      -= 1;
3354           nonstatic_word_space_count = 1; // Only one will fit
3355           length -= BytesPerInt;
3356           offset += BytesPerInt;
3357         }
3358         nonstatic_short_space_offset = offset;
3359         while( length >= BytesPerShort && nonstatic_short_count > 0 ) {
3360           nonstatic_short_count       -= 1;
3361           nonstatic_short_space_count += 1;
3362           length -= BytesPerShort;
3363           offset += BytesPerShort;
3364         }
3365         nonstatic_byte_space_offset = offset;
3366         while( length > 0 && nonstatic_byte_count > 0 ) {
3367           nonstatic_byte_count       -= 1;
3368           nonstatic_byte_space_count += 1;
3369           length -= 1;
3370         }
3371         // Allocate oop field in the gap if there are no other fields for that.
3372         nonstatic_oop_space_offset = offset;
3373         if( length >= heapOopSize && nonstatic_oop_count > 0 &&
3374             allocation_style != 0 ) { // when oop fields not first
3375           nonstatic_oop_count      -= 1;
3376           nonstatic_oop_space_count = 1; // Only one will fit
3377           length -= heapOopSize;
3378           offset += heapOopSize;
3379         }
3380       }
3381     }

3383     next_nonstatic_word_offset  = next_nonstatic_double_offset +
3384                                   (nonstatic_double_count * BytesPerLong);
3385     next_nonstatic_short_offset = next_nonstatic_word_offset +
3386                                   (nonstatic_word_count * BytesPerInt);
3387     next_nonstatic_byte_offset  = next_nonstatic_short_offset +
3388                                   (nonstatic_short_count * BytesPerShort);

3390     int notaligned_offset;
3391     if( allocation_style == 0 ) {
3392       notaligned_offset = next_nonstatic_byte_offset + nonstatic_byte_count;
3393     } else { // allocation_style == 1
3394       next_nonstatic_oop_offset = next_nonstatic_byte_offset + nonstatic_byte_co
```



```
3395       if( nonstatic_oop_count > 0 ) {
3396         next_nonstatic_oop_offset = align_size_up(next_nonstatic_oop_offset, hea
3397       }
3398       notaligned_offset = next_nonstatic_oop_offset + (nonstatic_oop_count * hea
3399     }
3400     next_nonstatic_type_offset = align_size_up(notaligned_offset, heapOopSize );
3401     nonstatic_field_size = nonstatic_field_size + ((next_nonstatic_type_offset
3402                                    - first_nonstatic_field_offset)/heapOopSize);

3404     // Iterate over fields again and compute correct offsets.
3405     // The field allocation type was temporarily stored in the offset slot.
3406     // oop fields are located before non-oop fields (static and non-static).
3407     int len = fields->length();
3408     for (int i = 0; i < len; i += instanceKlass::next_offset) {
3409       int real_offset;
3410       FieldAllocationType atype = (FieldAllocationType) fields->ushort_at(i + in
3411       switch (atype) {
3412         case STATIC_OOP:
3413           real_offset = next_static_oop_offset;
3414           next_static_oop_offset += heapOopSize;
3415           break;
3416         case STATIC_BYTE:
3417           real_offset = next_static_byte_offset;
3418           next_static_byte_offset += 1;
3419           break;
3420         case STATIC_SHORT:
3421           real_offset = next_static_short_offset;
3422           next_static_short_offset += BytesPerShort;
3423           break;
3424         case STATIC_WORD:
3425           real_offset = next_static_word_offset;
3426           next_static_word_offset += BytesPerInt;
3427           break;
3428         case STATIC_ALIGNED_DOUBLE:
3429         case STATIC_DOUBLE:
3430           real_offset = next_static_double_offset;
3431           next_static_double_offset += BytesPerLong;
3432           break;
3433         case NONSTATIC_OOP:
3434           if( nonstatic_oop_space_count > 0 ) {
3435             real_offset = nonstatic_oop_space_offset;
3436             nonstatic_oop_space_offset += heapOopSize;
3437             nonstatic_oop_space_count  -= 1;
3438           } else {
3439             real_offset = next_nonstatic_oop_offset;
3440             next_nonstatic_oop_offset += heapOopSize;
3441           }
3442           // Update oop maps
3443           if( nonstatic_oop_map_count > 0 &&
3444               nonstatic_oop_offsets[nonstatic_oop_map_count - 1] ==
3445               real_offset -
3446               int(nonstatic_oop_counts[nonstatic_oop_map_count - 1]) *
3447               heapOopSize ) {
3448             // Extend current oop map
3449             nonstatic_oop_counts[nonstatic_oop_map_count - 1] += 1;
3450           } else {
3451             // Create new oop map
3452             nonstatic_oop_offsets[nonstatic_oop_map_count] = real_offset;
3453             nonstatic_oop_counts [nonstatic_oop_map_count] = 1;
3454             nonstatic_oop_map_count += 1;
3455             if( first_nonstatic_oop_offset == 0 ) { // Undefined
3456               first_nonstatic_oop_offset = real_offset;
3457             }
3458           }
3459           break;
3460         case NONSTATIC_BYTE:
```

```
3461           if( nonstatic_byte_space_count > 0 ) {
3462             real_offset = nonstatic_byte_space_offset;
3463             nonstatic_byte_space_offset += 1;
3464             nonstatic_byte_space_count  -= 1;
3465           } else {
3466             real_offset = next_nonstatic_byte_offset;
3467             next_nonstatic_byte_offset += 1;
3468           }
3469           break;
3470         case NONSTATIC_SHORT:
3471           if( nonstatic_short_space_count > 0 ) {
3472             real_offset = nonstatic_short_space_offset;
3473             nonstatic_short_space_offset += BytesPerShort;
3474             nonstatic_short_space_count  -= 1;
3475           } else {
3476             real_offset = next_nonstatic_short_offset;
3477             next_nonstatic_short_offset += BytesPerShort;
3478           }
3479           break;
3480         case NONSTATIC_WORD:
3481           if( nonstatic_word_space_count > 0 ) {
3482             real_offset = nonstatic_word_space_offset;
3483             nonstatic_word_space_offset += BytesPerInt;
3484             nonstatic_word_space_count  -= 1;
3485           } else {
3486             real_offset = next_nonstatic_word_offset;
3487             next_nonstatic_word_offset += BytesPerInt;
3488           }
3489           break;
3490         case NONSTATIC_ALIGNED_DOUBLE:
3491         case NONSTATIC_DOUBLE:
3492           real_offset = next_nonstatic_double_offset;
3493           next_nonstatic_double_offset += BytesPerLong;
3494           break;
3495         default:
3496           ShouldNotReachHere();
3497       }
3498       fields->short_at_put(i + instanceKlass::low_offset,  extract_low_short_fro
3499       fields->short_at_put(i + instanceKlass::high_offset, extract_high_short_fr
3500     }

3502     // Size of instances
3503     int instance_size;

3505     next_nonstatic_type_offset = align_size_up(notaligned_offset, wordSize );
3506     instance_size = align_object_size(next_nonstatic_type_offset / wordSize);

3508     assert(instance_size == align_object_size(align_size_up((instanceOopDesc::ba

3510     // Number of non-static oop map blocks allocated at end of klass.
3511     const unsigned int total_oop_map_count =
3512       compute_oop_map_count(super_klass, nonstatic_oop_map_count,
3513                             first_nonstatic_oop_offset);

3515     // Compute reference type
3516     ReferenceType rt;
3517     if (super_klass() == NULL) {
3518       rt = REF_NONE;
3519     } else {
3520       rt = super_klass->reference_type();
3521     }

3523     // We can now create the basic klassOop for this klass
3524     klassOop ik = oopFactory::new_instanceKlass(name, vtable_size, itable_size,
3525                                                 static_field_size,
3526                                                 total_oop_map_count,
```



```
3527                                                 rt, CHECK_(nullHandle));
3528     instanceKlassHandle this_klass (THREAD, ik);

3530     assert(this_klass->static_field_size() == static_field_size, "sanity");
3531     assert(this_klass->nonstatic_oop_map_count() == total_oop_map_count,
3532            "sanity");

3534     // Fill in information already parsed
3535     this_klass->set_access_flags(access_flags);
3536     this_klass->set_should_verify_class(verify);
3537     jint lh = Klass::instance_layout_helper(instance_size, false);
3538     this_klass->set_layout_helper(lh);
3539     assert(this_klass->oop_is_instance(), "layout is correct");
3540     assert(this_klass->size_helper() == instance_size, "correct size_helper");
3541     // Not yet: supers are done below to support the new subtype-checking fields
3542     //this_klass->set_super(super_klass());
3543     this_klass->set_class_loader(class_loader());
3544     this_klass->set_nonstatic_field_size(nonstatic_field_size);
3545     this_klass->set_has_nonstatic_fields(has_nonstatic_fields);
3546     this_klass->set_static_oop_field_count(fac.static_oop_count);
3547     cp->set_pool_holder(this_klass());
3548     error_handler.set_in_error(false);   // turn off error handler for cp
3549     this_klass->set_constants(cp());
3550     this_klass->set_local_interfaces(local_interfaces());
3551     this_klass->set_fields(fields());
3552     this_klass->set_methods(methods());
3553     if (has_final_method) {
3554       this_klass->set_has_final_method();
3555     }
3556     this_klass->set_method_ordering(method_ordering());
3557     // The instanceKlass::_methods_jmethod_ids cache and the
3558     // instanceKlass::_methods_cached_itable_indices cache are
3559     // both managed on the assumption that the initial cache
3560     // size is equal to the number of methods in the class. If
3561     // that changes, then instanceKlass::idnum_can_increment()
3562     // has to be changed accordingly.
3563     this_klass->set_initial_method_idnum(methods->length());
3564     this_klass->set_name(cp->klass_name_at(this_class_index));
3565     if (LinkWellKnownClasses || is_anonymous())  // I am well known to myself
3566       cp->klass_at_put(this_class_index, this_klass()); // eagerly resolve
3567     this_klass->set_protection_domain(protection_domain());
3568     this_klass->set_fields_annotations(fields_annotations());
3569     this_klass->set_methods_annotations(methods_annotations());
3570     this_klass->set_methods_parameter_annotations(methods_parameter_annotations(
3571     this_klass->set_methods_default_annotations(methods_default_annotations());

3573     this_klass->set_minor_version(minor_version);
3574     this_klass->set_major_version(major_version);

3576     // Set up methodOop::intrinsic_id as soon as we know the names of methods.
3577     // (We used to do this lazily, but now we query it in Rewriter,
3578     // which is eagerly done for every method, so we might as well do it now,
3579     // when everything is fresh in memory.)
3580     if (methodOopDesc::klass_id_for_intrinsics(this_klass->as_klassOop()) != vmS
3581       for (int j = 0; j < methods->length(); j++) {
3582         ((methodOop)methods->obj_at(j))->init_intrinsic_id();
3583       }
3584     }

3586     if (cached_class_file_bytes != NULL) {
3587       // JVMTI: we have an instanceKlass now, tell it about the cached bytes
3588       this_klass->set_cached_class_file(cached_class_file_bytes,
3589                                         cached_class_file_length);
3590     }

3592     // Miranda methods
```

```
3593     if ((num_miranda_methods > 0) ||
3594         // if this class introduced new miranda methods or
3595         (super_klass.not_null() && (super_klass->has_miranda_methods()))
3596         // super class exists and this class inherited miranda methods
3597         ) {
3598       this_klass->set_has_miranda_methods(); // then set a flag
3599     }

3601     // Additional attributes
3602     parse_classfile_attributes(cp, this_klass, CHECK_(nullHandle));

3604     // Make sure this is the end of class file stream
3605     guarantee_property(cfs->at_eos(), "Extra bytes at the end of class file %s",

3607     // VerifyOops believes that once this has been set, the object is completely
3608     // Compute transitive closure of interfaces this class implements
3609     this_klass->set_transitive_interfaces(transitive_interfaces());

3611     // Fill in information needed to compute superclasses.
3612     this_klass->initialize_supers(super_klass(), CHECK_(nullHandle));

3614     // Initialize itable offset tables
3615     klassItable::setup_itable_offset_table(this_klass);

3617     // Do final class setup
3618     fill_oop_maps(this_klass, nonstatic_oop_map_count, nonstatic_oop_offsets, no

3620     set_precomputed_flags(this_klass);

3622     // reinitialize modifiers, using the InnerClasses attribute
3623     int computed_modifiers = this_klass->compute_modifier_flags(CHECK_(nullHandl
3624     this_klass->set_modifier_flags(computed_modifiers);

3626     // check if this class can access its super class
3627     check_super_class_access(this_klass, CHECK_(nullHandle));

3629     // check if this class can access its superinterfaces
3630     check_super_interface_access(this_klass, CHECK_(nullHandle));

3632     // check if this class overrides any final method
3633     check_final_method_override(this_klass, CHECK_(nullHandle));

3635     // check that if this class is an interface then it doesn't have static meth
3636     if (this_klass->is_interface()) {
3637       check_illegal_static_method(this_klass, CHECK_(nullHandle));
3638     }

3640     // Allocate mirror and initialize static fields
3641     java_lang_Class::create_mirror(this_klass, CHECK_(nullHandle));

3643     ClassLoadingService::notify_class_loaded(instanceKlass::cast(this_klass()),
3644                                              false /* not shared class */);

3646     if (TraceClassLoading) {
3647       // print in a single call to reduce interleaving of output
3648       if (cfs->source() != NULL) {
3649         tty->print("[Loaded %s from %s]\n", this_klass->external_name(),
3650                    cfs->source());
3651       } else if (class_loader.is_null()) {
3652         if (THREAD->is_Java_thread()) {
3653           klassOop caller = ((JavaThread*)THREAD)->security_get_caller_class(1);
3654           tty->print("[Loaded %s by instance of %s]\n",
3655                      this_klass->external_name(),
3656                      instanceKlass::cast(caller)->external_name());
3657         } else {
3658           tty->print("[Loaded %s]\n", this_klass->external_name());
```



```
3659         }
3660       } else {
3661         ResourceMark rm;
3662         tty->print("[Loaded %s from %s]\n", this_klass->external_name(),
3663                    instanceKlass::cast(class_loader->klass())->external_name());
3664       }
3665     }

3667     if (TraceClassResolution) {
3668       // print out the superclass.
3669       const char * from = Klass::cast(this_klass())->external_name();
3670       if (this_klass->java_super() != NULL) {
3671         tty->print("RESOLVE %s %s (super)\n", from, instanceKlass::cast(this_kla
3672       }
3673       // print out each of the interface classes referred to by this class.
3674       objArrayHandle local_interfaces(THREAD, this_klass->local_interfaces());
3675       if (!local_interfaces.is_null()) {
3676         int length = local_interfaces->length();
3677         for (int i = 0; i < length; i++) {
3678           klassOop k = klassOop(local_interfaces->obj_at(i));
3679           instanceKlass* to_class = instanceKlass::cast(k);
3680           const char * to = to_class->external_name();
3681           tty->print("RESOLVE %s %s (interface)\n", from, to);
3682         }
3683       }
3684     }

3686 #ifndef PRODUCT
3687     if( PrintCompactFieldsSavings ) {
3688       if( nonstatic_field_size < orig_nonstatic_field_size ) {
3689         tty->print("[Saved %d of %d bytes in %s]\n",
3690                    (orig_nonstatic_field_size - nonstatic_field_size)*heapOopSize,
3691                    orig_nonstatic_field_size*heapOopSize,
3692                    this_klass->external_name());
3693       } else if( nonstatic_field_size > orig_nonstatic_field_size ) {
3694         tty->print("[Wasted %d over %d bytes in %s]\n",
3695                    (nonstatic_field_size - orig_nonstatic_field_size)*heapOopSize,
3696                    orig_nonstatic_field_size*heapOopSize,
3697                    this_klass->external_name());
3698       }
3699     }
3700 #endif

3702     // preserve result across HandleMark
3703     preserve_this_klass = this_klass();
3704   }

3706   // Create new handle outside HandleMark
3707   instanceKlassHandle this_klass (THREAD, preserve_this_klass);
3708   debug_only(this_klass->as_klassOop()->verify();)

3710   return this_klass;
3711 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   13980 Wed Mar 30 07:00:15 2011
new/src/share/vm/classfile/classFileParser.hpp
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 1997, 2011, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.
   8  *
   9  * This code is distributed in the hope that it will be useful, but WITHOUT
  10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  12  * version 2 for more details (a copy is included in the LICENSE file that
  13  * accompanied this code).
  14  *
  15  * You should have received a copy of the GNU General Public License version
  16  * 2 along with this work; if not, write to the Free Software Foundation,
  17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  18  *
  19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  20  * or visit www.oracle.com if you need additional information or have any
  21  * questions.
  22  *
  23  */

  25 #ifndef SHARE_VM_CLASSFILE_CLASSFILEPARSER_HPP
  26 #define SHARE_VM_CLASSFILE_CLASSFILEPARSER_HPP

  28 #include "classfile/classFileStream.hpp"
  29 #include "memory/resourceArea.hpp"
  30 #include "oops/oop.inline.hpp"
  31 #include "oops/typeArrayOop.hpp"
  32 #include "runtime/handles.inline.hpp"
  33 #include "utilities/accessFlags.hpp"

  35 class TempNewSymbol;
  36 // Parser for for .class files
  37 //
  38 // The bytes describing the class file structure is read from a Stream object

  40 class ClassFileParser VALUE_OBJ_CLASS_SPEC {
  41  private:
  42   bool _need_verify;
  43   bool _relax_verify;
  44   u2   _major_version;
  45   u2   _minor_version;
  46   Symbol* _class_name;
  47   KlassHandle _host_klass;
  48   GrowableArray<Handle>* _cp_patches; // overrides for CP entries

  50   bool _has_finalizer;
  51   bool _has_empty_finalizer;
  52   bool _has_vanilla_constructor;

  54   int _max_bootstrap_specifier_index;

  56   enum { fixed_buffer_size = 128 };
  57   u_char linenumbertable_buffer[fixed_buffer_size];

  59   ClassFileStream* _stream;             // Actual input stream

  61   enum { LegalClass, LegalField, LegalMethod }; // used to verify unqualified na
```



```
  63   // Accessors
  64   ClassFileStream* stream()                       { return _stream; }
  65   void set_stream(ClassFileStream* st)            { _stream = st; }

  67   // Constant pool parsing
  68   void parse_constant_pool_entries(constantPoolHandle cp, int length, TRAPS);

  70   constantPoolHandle parse_constant_pool(TRAPS);

  72   // Interface parsing
  73   objArrayHandle parse_interfaces(constantPoolHandle cp,
  74                                   int length,
  75                                   Handle class_loader,
  76                                   Handle protection_domain,
  77                                   Symbol* class_name,
  78                                   TRAPS);

  80   // Field parsing
  81   void parse_field_attributes(constantPoolHandle cp, u2 attributes_count,
  82                               bool is_static, u2 signature_index,
  83                               u2* constantvalue_index_addr,
  84                               bool* is_synthetic_addr,
  85                               u2* generic_signature_index_addr,
  86                               typeArrayHandle* field_annotations, TRAPS);
  87   typeArrayHandle parse_fields(constantPoolHandle cp, bool is_interface,
  88                                struct FieldAllocationCount *fac,
  89                                objArrayHandle* fields_annotations, TRAPS);

  91   // Method parsing
  92   methodHandle parse_method(constantPoolHandle cp, bool is_interface,
  93                             AccessFlags* promoted_flags,
  94                             typeArrayHandle* method_annotations,
  95                             typeArrayHandle* method_parameter_annotations,
  96                             typeArrayHandle* method_default_annotations,
  97                             TRAPS);
  98   objArrayHandle parse_methods (constantPoolHandle cp, bool is_interface,
  99                                 AccessFlags* promoted_flags,
 100                                 bool* has_final_method,
 101                                 objArrayOop* methods_annotations_oop,
 102                                 objArrayOop* methods_parameter_annotations_oop,
 103                                 objArrayOop* methods_default_annotations_oop,
 104                                 TRAPS);
 105   typeArrayHandle sort_methods (objArrayHandle methods,
 106                                 objArrayHandle methods_annotations,
 107                                 objArrayHandle methods_parameter_annotations,
 108                                 objArrayHandle methods_default_annotations,
 109                                 TRAPS);
 110   typeArrayHandle parse_exception_table(u4 code_length, u4 exception_table_lengt
 111                                         constantPoolHandle cp, TRAPS);
 112   void parse_linenumber_table(
 113       u4 code_attribute_length, u4 code_length,
 114       CompressedLineNumberWriteStream** write_stream, TRAPS);
 115   u2* parse_localvariable_table(u4 code_length, u2 max_locals, u4 code_attribute
 116                                 constantPoolHandle cp, u2* localvariable_table_l
 117                                 bool isLVTT, TRAPS);
 118   u2* parse_checked_exceptions(u2* checked_exceptions_length, u4 method_attribut
 119                                constantPoolHandle cp, TRAPS);
 120   void parse_type_array(u2 array_length, u4 code_length, u4* u1_index, u4* u2_in
 121                         u1* u1_array, u2* u2_array, constantPoolHandle cp, TRAPS
 122   typeArrayOop parse_stackmap_table(u4 code_attribute_length, TRAPS);

 124   // Classfile attribute parsing
 125   void parse_classfile_sourcefile_attribute(constantPoolHandle cp, instanceKlass
 126   void parse_classfile_source_debug_extension_attribute(constantPoolHandle cp,
 127                                                          instanceKlassHandle k, int lengt
 128   u2   parse_classfile_inner_classes_attribute(constantPoolHandle cp,
```

```
129                                             instanceKlassHandle k, TRAPS);
130   void parse_classfile_attributes(constantPoolHandle cp, instanceKlassHandle k,
131   void parse_classfile_synthetic_attribute(constantPoolHandle cp, instanceKlassH
132   void parse_classfile_signature_attribute(constantPoolHandle cp, instanceKlassH
133   void parse_classfile_bootstrap_methods_attribute(constantPoolHandle cp, instan

135   // Annotations handling
136   typeArrayHandle assemble_annotations(u1* runtime_visible_annotations,
137                                        int runtime_visible_annotations_length,
138                                        u1* runtime_invisible_annotations,
139                                        int runtime_invisible_annotations_length,

141   // Final setup
142   unsigned int compute_oop_map_count(instanceKlassHandle super,
143                                      unsigned int nonstatic_oop_count,
144                                      int first_nonstatic_oop_offset);
145   void fill_oop_maps(instanceKlassHandle k,
146                      unsigned int nonstatic_oop_map_count,
147                      int* nonstatic_oop_offsets,
148                      unsigned int* nonstatic_oop_counts);
149   void set_precomputed_flags(instanceKlassHandle k);
150   objArrayHandle compute_transitive_interfaces(instanceKlassHandle super,
151                                                objArrayHandle local_ifs, TRAPS);

153   // Special handling for certain classes.
154   // Add the "discovered" field to java.lang.ref.Reference if
155   // it does not exist.
156   void java_lang_ref_Reference_fix_pre(typeArrayHandle* fields_ptr,
157                                        constantPoolHandle cp,
158                                        FieldAllocationCount *fac_ptr, TRAPS);
159   // Adjust the field allocation counts for java.lang.Class to add
160   // fake fields.
161   void java_lang_Class_fix_pre(int* nonstatic_field_size,
162                                FieldAllocationCount *fac_ptr);
163   // Adjust the next_nonstatic_oop_offset to place the fake fields
164   // before any Java fields.
165   void java_lang_Class_fix_post(int* next_nonstatic_oop_offset);
166   // Adjust the field allocation counts for java.lang.invoke.MethodHandle to add
167   // a fake address (void*) field.
168   void java_lang_invoke_MethodHandle_fix_pre(constantPoolHandle cp,
169                                        typeArrayHandle fields,
170                                        FieldAllocationCount *fac_ptr, TRAPS);

172   // Format checker methods
173   void classfile_parse_error(const char* msg, TRAPS);
174   void classfile_parse_error(const char* msg, int index, TRAPS);
175   void classfile_parse_error(const char* msg, const char *name, TRAPS);
176   void classfile_parse_error(const char* msg, int index, const char *name, TRAPS
177   inline void guarantee_property(bool b, const char* msg, TRAPS) {
178     if (!b) { classfile_parse_error(msg, CHECK); }
179   }

181   inline void assert_property(bool b, const char* msg, TRAPS) {
182 #ifdef ASSERT
183     if (!b) { fatal(msg); }
184 #endif
185   }

187   inline void check_property(bool property, const char* msg, int index, TRAPS) {
188     if (_need_verify) {
189       guarantee_property(property, msg, index, CHECK);
190     } else {
191       assert_property(property, msg, CHECK);
192     }
193   }
```



```
195   inline void check_property(bool property, const char* msg, TRAPS) {
196     if (_need_verify) {
197       guarantee_property(property, msg, CHECK);
198     } else {
199       assert_property(property, msg, CHECK);
200     }
201   }

203   inline void guarantee_property(bool b, const char* msg, int index, TRAPS) {
204     if (!b) { classfile_parse_error(msg, index, CHECK); }
205   }
206   inline void guarantee_property(bool b, const char* msg, const char *name, TRAP
207     if (!b) { classfile_parse_error(msg, name, CHECK); }
208   }
209   inline void guarantee_property(bool b, const char* msg, int index, const char
210     if (!b) { classfile_parse_error(msg, index, name, CHECK); }
211   }

213   void throwIllegalSignature(
214       const char* type, Symbol* name, Symbol* sig, TRAPS);

216   bool is_supported_version(u2 major, u2 minor);
217   bool has_illegal_visibility(jint flags);

219   void verify_constantvalue(int constantvalue_index, int signature_index, consta
220   void verify_legal_utf8(const unsigned char* buffer, int length, TRAPS);
221   void verify_legal_class_name(Symbol* name, TRAPS);
222   void verify_legal_field_name(Symbol* name, TRAPS);
223   void verify_legal_method_name(Symbol* name, TRAPS);
224   void verify_legal_field_signature(Symbol* fieldname, Symbol* signature, TRAPS)
225   int  verify_legal_method_signature(Symbol* methodname, Symbol* signature, TRAP
226   void verify_legal_class_modifiers(jint flags, TRAPS);
227   void verify_legal_field_modifiers(jint flags, bool is_interface, TRAPS);
228   void verify_legal_method_modifiers(jint flags, bool is_interface, Symbol* name
229   bool verify_unqualified_name(char* name, unsigned int length, int type);
230   char* skip_over_field_name(char* name, bool slash_ok, unsigned int length);
231   char* skip_over_field_signature(char* signature, bool void_ok, unsigned int le

233   bool is_anonymous() {
234     assert(EnableInvokeDynamic || _host_klass.is_null(), "");
234     assert(AnonymousClasses || _host_klass.is_null(), "");
235     return _host_klass.not_null();
236   }
237   bool has_cp_patch_at(int index) {
238     assert(EnableInvokeDynamic, "");
238     assert(AnonymousClasses, "");
239     assert(index >= 0, "oob");
240     return (_cp_patches != NULL
241             && index < _cp_patches->length()
242             && _cp_patches->adr_at(index)->not_null());
243   }
244   Handle cp_patch_at(int index) {
245     assert(has_cp_patch_at(index), "oob");
246     return _cp_patches->at(index);
247   }
248   Handle clear_cp_patch_at(int index) {
249     Handle patch = cp_patch_at(index);
250     _cp_patches->at_put(index, Handle());
251     assert(!has_cp_patch_at(index), "");
252     return patch;
253   }
254   void patch_constant_pool(constantPoolHandle cp, int index, Handle patch, TRAPS

256   // Wrapper for constantTag.is_klass_[or_]reference.
257   // In older versions of the VM, klassOops cannot sneak into early phases of
258   // constant pool construction, but in later versions they can.
```

```
 259   // %%% Let's phase out the old is_klass_reference.
 260   bool is_klass_reference(constantPoolHandle cp, int index) {
 261     return ((LinkWellKnownClasses || EnableInvokeDynamic)
 261     return ((LinkWellKnownClasses || AnonymousClasses)
 262             ? cp->tag_at(index).is_klass_or_reference()
 263             : cp->tag_at(index).is_klass_reference());
 264   }

 266  public:
 267   // Constructor
 268   ClassFileParser(ClassFileStream* st) { set_stream(st); }

 270   // Parse .class file and return new klassOop. The klassOop is not hooked up
 271   // to the system dictionary or any other structures, so a .class file can
 272   // be loaded several times if desired.
 273   // The system dictionary hookup is done by the caller.
 274   //
 275   // "parsed_name" is updated by this method, and is the name found
 276   // while parsing the stream.
 277   instanceKlassHandle parseClassFile(Symbol* name,
 278                                      Handle class_loader,
 279                                      Handle protection_domain,
 280                                      TempNewSymbol& parsed_name,
 281                                      bool verify,
 282                                      TRAPS) {
 283     KlassHandle no_host_klass;
 284     return parseClassFile(name, class_loader, protection_domain, no_host_klass,
 285   }
 286   instanceKlassHandle parseClassFile(Symbol* name,
 287                                      Handle class_loader,
 288                                      Handle protection_domain,
 289                                      KlassHandle host_klass,
 290                                      GrowableArray<Handle>* cp_patches,
 291                                      TempNewSymbol& parsed_name,
 292                                      bool verify,
 293                                      TRAPS);

 295   // Verifier checks
 296   static void check_super_class_access(instanceKlassHandle this_klass, TRAPS);
 297   static void check_super_interface_access(instanceKlassHandle this_klass, TRAPS
 298   static void check_final_method_override(instanceKlassHandle this_klass, TRAPS)
 299   static void check_illegal_static_method(instanceKlassHandle this_klass, TRAPS)
 300 };
```

*unchanged_portion_omitted_*

```
**********************************************************
  126403 Wed Mar 30 07:00:16 2011
new/src/share/vm/classfile/javaClasses.cpp
**********************************************************
______unchanged_portion_omitted_


2302 // Support for java_lang_invoke_MethodHandle

2304 int java_lang_invoke_MethodHandle::_type_offset;
2305 int java_lang_invoke_MethodHandle::_vmtarget_offset;
2306 int java_lang_invoke_MethodHandle::_vmentry_offset;
2307 int java_lang_invoke_MethodHandle::_vmslots_offset;

2309 int java_lang_invoke_MemberName::_clazz_offset;
2310 int java_lang_invoke_MemberName::_name_offset;
2311 int java_lang_invoke_MemberName::_type_offset;
2312 int java_lang_invoke_MemberName::_flags_offset;
2313 int java_lang_invoke_MemberName::_vmtarget_offset;
2314 int java_lang_invoke_MemberName::_vmindex_offset;

2316 int java_lang_invoke_DirectMethodHandle::_vmindex_offset;

2318 int java_lang_invoke_BoundMethodHandle::_argument_offset;
2319 int java_lang_invoke_BoundMethodHandle::_vmargslot_offset;

2321 int java_lang_invoke_AdapterMethodHandle::_conversion_offset;

2323 void java_lang_invoke_MethodHandle::compute_offsets() {
2324   klassOop k = SystemDictionary::MethodHandle_klass();
2325   if (k != NULL && EnableInvokeDynamic) {
2325   if (k != NULL && EnableMethodHandles) {
2326     bool allow_super = false;
2327     if (AllowTransitionalJSR292)  allow_super = true;  // temporary, to access j
2328     compute_offset(_type_offset,      k, vmSymbols::type_name(),      vmSymbols:
2329     compute_offset(_vmtarget_offset,  k, vmSymbols::vmtarget_name(),  vmSymbols:
2330     compute_offset(_vmentry_offset,   k, vmSymbols::vmentry_name(),   vmSymbols:

2332     // Note:  MH.vmslots (if it is present) is a hoisted copy of MH.type.form.vm
2333     // It is optional pending experiments to keep or toss.
2334     compute_optional_offset(_vmslots_offset, k, vmSymbols::vmslots_name(), vmSym
2335   }
2336 }

2338 void java_lang_invoke_MemberName::compute_offsets() {
2339   klassOop k = SystemDictionary::MemberName_klass();
2340   if (k != NULL && EnableInvokeDynamic) {
2340   if (k != NULL && EnableMethodHandles) {
2341     compute_offset(_clazz_offset,     k, vmSymbols::clazz_name(),     vmSymbols:
2342     compute_offset(_name_offset,      k, vmSymbols::name_name(),      vmSymbols:
2343     compute_offset(_type_offset,      k, vmSymbols::type_name(),      vmSymbols:
2344     compute_offset(_flags_offset,     k, vmSymbols::flags_name(),     vmSymbols:
2345     compute_offset(_vmtarget_offset,  k, vmSymbols::vmtarget_name(),  vmSymbols:
2346     compute_offset(_vmindex_offset,   k, vmSymbols::vmindex_name(),   vmSymbols:
2347   }
2348 }

2350 void java_lang_invoke_DirectMethodHandle::compute_offsets() {
2351   klassOop k = SystemDictionary::DirectMethodHandle_klass();
2352   if (k != NULL && EnableInvokeDynamic) {
2352   if (k != NULL && EnableMethodHandles) {
2353     compute_offset(_vmindex_offset,   k, vmSymbols::vmindex_name(),   vmSymbols:
2354   }
2355 }

2357 void java_lang_invoke_BoundMethodHandle::compute_offsets() {
```



```
2358   klassOop k = SystemDictionary::BoundMethodHandle_klass();
2359   if (k != NULL && EnableInvokeDynamic) {
2359   if (k != NULL && EnableMethodHandles) {
2360     compute_offset(_vmargslot_offset, k, vmSymbols::vmargslot_name(), vmSymbols:
2361     compute_offset(_argument_offset,  k, vmSymbols::argument_name(),  vmSymbols:
2362   }
2363 }

2365 void java_lang_invoke_AdapterMethodHandle::compute_offsets() {
2366   klassOop k = SystemDictionary::AdapterMethodHandle_klass();
2367   if (k != NULL && EnableInvokeDynamic) {
2367   if (k != NULL && EnableMethodHandles) {
2368     compute_offset(_conversion_offset, k, vmSymbols::conversion_name(), vmSymbol
2369   }
2370 }
______unchanged_portion_omitted_


2979 // Compute non-hard-coded field offsets of all the classes in this file
2980 void JavaClasses::compute_offsets() {

2982   java_lang_Class::compute_offsets();
2983   java_lang_Thread::compute_offsets();
2984   java_lang_ThreadGroup::compute_offsets();
2985   if (EnableInvokeDynamic) {
2985   if (EnableMethodHandles) {
2986     java_lang_invoke_MethodHandle::compute_offsets();
2987     java_lang_invoke_MemberName::compute_offsets();
2988     java_lang_invoke_DirectMethodHandle::compute_offsets();
2989     java_lang_invoke_BoundMethodHandle::compute_offsets();
2990     java_lang_invoke_AdapterMethodHandle::compute_offsets();
2991     java_lang_invoke_MethodType::compute_offsets();
2992     java_lang_invoke_MethodTypeForm::compute_offsets();
2993   }
2994   if (EnableInvokeDynamic) {
2993     java_lang_invoke_CallSite::compute_offsets();
2994   }
2995   java_security_AccessControlContext::compute_offsets();
2996   // Initialize reflection classes. The layouts of these classes
2997   // changed with the new reflection implementation in JDK 1.4, and
2998   // since the Universe doesn't know what JDK version it is until this
2999   // point we defer computation of these offsets until now.
3000   java_lang_reflect_AccessibleObject::compute_offsets();
3001   java_lang_reflect_Method::compute_offsets();
3002   java_lang_reflect_Constructor::compute_offsets();
3003   java_lang_reflect_Field::compute_offsets();
3004   if (JDK_Version::is_gte_jdk14x_version()) {
3005     java_nio_Buffer::compute_offsets();
3006   }
3007   if (JDK_Version::is_gte_jdk15x_version()) {
3008     sun_reflect_ConstantPool::compute_offsets();
3009     sun_reflect_UnsafeStaticFieldAccessorImpl::compute_offsets();
3010   }
3011   sun_misc_AtomicLongCSImpl::compute_offsets();

3013   // generated interpreter code wants to know about the offsets we just computed
3014   AbstractAssembler::update_delayed_values();
3015 }
______unchanged_portion_omitted_
```

```
**********************************************************
  126231 Wed Mar 30 07:00:17 2011
new/src/share/vm/classfile/systemDictionary.cpp
**********************************************************
______unchanged_portion_omitted_

 972 // Note: this method is much like resolve_from_stream, but
 973 // updates no supplemental data structures.
 974 // TODO consolidate the two methods with a helper routine?
 975 klassOop SystemDictionary::parse_stream(Symbol* class_name,
 976                                         Handle class_loader,
 977                                         Handle protection_domain,
 978                                         ClassFileStream* st,
 979                                         KlassHandle host_klass,
 980                                         GrowableArray<Handle>* cp_patches,
 981                                         TRAPS) {
 982   TempNewSymbol parsed_name = NULL;

 984   // Parse the stream. Note that we do this even though this klass might
 985   // already be present in the SystemDictionary, otherwise we would not
 986   // throw potential ClassFormatErrors.
 987   //
 988   // Note: "name" is updated.
 989   // Further note:  a placeholder will be added for this class when
 990   //   super classes are loaded (resolve_super_or_fail). We expect this
 991   //   to be called for all classes but java.lang.Object; and we preload
 992   //   java.lang.Object through resolve_or_fail, not this path.

 994   instanceKlassHandle k = ClassFileParser(st).parseClassFile(class_name,
 995                                                              class_loader,
 996                                                              protection_domain,
 997                                                              host_klass,
 998                                                              cp_patches,
 999                                                              parsed_name,
1000                                                              true,
1001                                                              THREAD);

1003   // We don't redefine the class, so we just need to clean up whether there
1004   // was an error or not (don't want to modify any system dictionary
1005   // data structures).
1006   // Parsed name could be null if we threw an error before we got far
1007   // enough along to parse it -- in that case, there is nothing to clean up.
1008   if (parsed_name != NULL) {
1009     unsigned int p_hash = placeholders()->compute_hash(parsed_name,
1010                                                        class_loader);
1011     int p_index = placeholders()->hash_to_index(p_hash);
1012     {
1013     MutexLocker mu(SystemDictionary_lock, THREAD);
1014     placeholders()->find_and_remove(p_index, p_hash, parsed_name, class_loader,
1015     SystemDictionary_lock->notify_all();
1016     }
1017   }

1019   if (host_klass.not_null() && k.not_null()) {
1020     assert(EnableInvokeDynamic, "");
1020     assert(AnonymousClasses, "");
1021     // If it's anonymous, initialize it now, since nobody else will.
1022     k->set_host_klass(host_klass());

1024     {
1025       MutexLocker mu_r(Compile_lock, THREAD);

1027       // Add to class hierarchy, initialize vtables, and do possible
1028       // deoptimizations.
1029       add_to_hierarchy(k, CHECK_NULL); // No exception, but can block
```



```
1031       // But, do not add to system dictionary.
1032     }

1034     k->eager_initialize(THREAD);

1036     // notify jvmti
1037     if (JvmtiExport::should_post_class_load()) {
1038         assert(THREAD->is_Java_thread(), "thread->is_Java_thread()");
1039         JvmtiExport::post_class_load((JavaThread *) THREAD, k());
1040     }
1041   }

1043   return k();
1044 }
______unchanged_portion_omitted_

1928 bool SystemDictionary::initialize_wk_klass(WKID id, int init_opt, TRAPS) {
1929   assert(id >= (int)FIRST_WKID && id < (int)WKID_LIMIT, "oob");
1930   int  info = wk_init_info[id - FIRST_WKID];
1931   int  sid  = (info >> CEIL_LG_OPTION_LIMIT);
1932   Symbol* symbol = vmSymbols::symbol_at((vmSymbols::SID)sid);
1933   klassOop*    klassp = &_well_known_klasses[id];
1934   bool pre_load = (init_opt < SystemDictionary::Opt);
1935   bool try_load = true;
1936   if (init_opt == SystemDictionary::Opt_Kernel) {
1937 #ifndef KERNEL
1938     try_load = false;
1939 #endif //KERNEL
1940   }
1941   Symbol* backup_symbol = NULL;  // symbol to try if the current symbol fails
1942   if (init_opt == SystemDictionary::Pre_JSR292) {
1943     if (!EnableInvokeDynamic)  try_load = false;  // do not bother to load such
1943     if (!EnableMethodHandles)  try_load = false;  // do not bother to load such
1944     if (AllowTransitionalJSR292) {
1945       backup_symbol = find_backup_class_name(symbol);
1946       if (try_load && PreferTransitionalJSR292) {
1947         while (backup_symbol != NULL) {
1948           (*klassp) = resolve_or_null(backup_symbol, CHECK_0); // try backup ear
1949           if (TraceMethodHandles) {
1950             ResourceMark rm;
1951             tty->print_cr("MethodHandles: try backup first for %s => %s (%s)",
1952                           symbol->as_C_string(), backup_symbol->as_C_string(),
1953                           ((*klassp) == NULL) ? "no such class" : "backup load s
1954           }
1955           if ((*klassp) != NULL)  return true;
1956           backup_symbol = find_backup_class_name(backup_symbol);  // find next b
1957         }
1958       }
1959     }
1960   }
1961   if ((*klassp) != NULL)  return true;
1962   if (!try_load)          return false;
1963   while (symbol != NULL) {
1964     bool must_load = (pre_load && (backup_symbol == NULL));
1965     if (must_load) {
1966       (*klassp) = resolve_or_fail(symbol, true, CHECK_0); // load required class
1967     } else {
1968       (*klassp) = resolve_or_null(symbol,       CHECK_0); // load optional klass
1969     }
1970     if ((*klassp) != NULL)  return true;
1971     // Go around again.  Example of long backup sequence:
1972     // java.lang.invoke.MemberName, java.dyn.MemberName, sun.dyn.MemberName, ONL
1973     if (TraceMethodHandles && (backup_symbol != NULL)) {
1974       ResourceMark rm;
1975       tty->print_cr("MethodHandles: backup for %s => %s",
1976                     symbol->as_C_string(), backup_symbol->as_C_string());
```

```
1977     }
1978     symbol = backup_symbol;
1979     if (AllowTransitionalJSR292)
1980       backup_symbol = find_backup_class_name(symbol);
1981   }
1982   return false;
1983 }
______unchanged_portion_omitted_


2011 void SystemDictionary::initialize_preloaded_classes(TRAPS) {
2012   assert(WK_KLASS(Object_klass) == NULL, "preloaded classes should only be initi
2013   // Preload commonly used klasses
2014   WKID scan = FIRST_WKID;
2015   // first do Object, String, Class
2016   initialize_wk_klasses_through(WK_KLASS_ENUM_NAME(Class_klass), scan, CHECK);

2018   debug_only(instanceKlass::verify_class_klass_nonstatic_oop_maps(WK_KLASS(Class

2020   // Fixup mirrors for classes loaded before java.lang.Class.
2021   // These calls iterate over the objects currently in the perm gen
2022   // so calling them at this point is matters (not before when there
2023   // are fewer objects and not later after there are more objects
2024   // in the perm gen.
2025   Universe::initialize_basic_type_mirrors(CHECK);
2026   Universe::fixup_mirrors(CHECK);

2028   // do a bunch more:
2029   initialize_wk_klasses_through(WK_KLASS_ENUM_NAME(Reference_klass), scan, CHECK

2031   // Preload ref klasses and set reference types
2032   instanceKlass::cast(WK_KLASS(Reference_klass))->set_reference_type(REF_OTHER);
2033   instanceRefKlass::update_nonstatic_oop_maps(WK_KLASS(Reference_klass));

2035   initialize_wk_klasses_through(WK_KLASS_ENUM_NAME(PhantomReference_klass), scan
2036   instanceKlass::cast(WK_KLASS(SoftReference_klass))->set_reference_type(REF_SOF
2037   instanceKlass::cast(WK_KLASS(WeakReference_klass))->set_reference_type(REF_WEA
2038   instanceKlass::cast(WK_KLASS(FinalReference_klass))->set_reference_type(REF_FI
2039   instanceKlass::cast(WK_KLASS(PhantomReference_klass))->set_reference_type(REF_

2041   // JSR 292 classes
2042   WKID jsr292_group_start = WK_KLASS_ENUM_NAME(MethodHandle_klass);
2043   WKID jsr292_group_end   = WK_KLASS_ENUM_NAME(CallSite_klass);
2044   initialize_wk_klasses_until(jsr292_group_start, scan, CHECK);
2041   WKID meth_group_start = WK_KLASS_ENUM_NAME(MethodHandle_klass);
2042   WKID meth_group_end   = WK_KLASS_ENUM_NAME(WrongMethodTypeException_klass);
2043   initialize_wk_klasses_until(meth_group_start, scan, CHECK);
2044   if (EnableMethodHandles) {
2045     initialize_wk_klasses_through(meth_group_end, scan, CHECK);
2046   }
2047   if (_well_known_klasses[meth_group_start] == NULL) {
2048     // Skip the rest of the method handle classes, if MethodHandle is not loaded
2049     scan = WKID(meth_group_end+1);
2050   }
2051   WKID indy_group_start = WK_KLASS_ENUM_NAME(Linkage_klass);
2052   WKID indy_group_end   = WK_KLASS_ENUM_NAME(CallSite_klass);
2053   initialize_wk_klasses_until(indy_group_start, scan, CHECK);
2045   if (EnableInvokeDynamic) {
2046     initialize_wk_klasses_through(jsr292_group_end, scan, CHECK);
2047   } else {
2048     // Skip the JSR 292 classes, if not enabled.
2049     scan = WKID(jsr292_group_end + 1);
2055     initialize_wk_klasses_through(indy_group_end, scan, CHECK);
2056   }
2057   if (_well_known_klasses[indy_group_start] == NULL) {
2058     // Skip the rest of the dynamic typing classes, if Linkage is not loaded.
```



```
2059     scan = WKID(indy_group_end+1);
2050   }

2052   initialize_wk_klasses_until(WKID_LIMIT, scan, CHECK);

2054   _box_klasses[T_BOOLEAN] = WK_KLASS(Boolean_klass);
2055   _box_klasses[T_CHAR]    = WK_KLASS(Character_klass);
2056   _box_klasses[T_FLOAT]   = WK_KLASS(Float_klass);
2057   _box_klasses[T_DOUBLE]  = WK_KLASS(Double_klass);
2058   _box_klasses[T_BYTE]    = WK_KLASS(Byte_klass);
2059   _box_klasses[T_SHORT]   = WK_KLASS(Short_klass);
2060   _box_klasses[T_INT]     = WK_KLASS(Integer_klass);
2061   _box_klasses[T_LONG]    = WK_KLASS(Long_klass);
2062   //_box_klasses[T_OBJECT]  = WK_KLASS(object_klass);
2063   //_box_klasses[T_ARRAY]   = WK_KLASS(object_klass);

2065 #ifdef KERNEL
2066   if (sun_jkernel_DownloadManager_klass() == NULL) {
2067     warning("Cannot find sun/jkernel/DownloadManager");
2068   }
2069 #endif // KERNEL

2071   { // Compute whether we should use loadClass or loadClassInternal when loading
2072     methodOop method = instanceKlass::cast(ClassLoader_klass())->find_method(vmS
2073     _has_loadClassInternal = (method != NULL);
2074   }
2075   { // Compute whether we should use checkPackageAccess or NOT
2076     methodOop method = instanceKlass::cast(ClassLoader_klass())->find_method(vmS
2077     _has_checkPackageAccess = (method != NULL);
2078   }
2079 }
______unchanged_portion_omitted_


2396 methodOop SystemDictionary::find_method_handle_invoke(Symbol* name,
2397                                                        Symbol* signature,
2398                                                        KlassHandle accessing_klas
2399                                                        TRAPS) {
2400   if (!EnableInvokeDynamic)  return NULL;
2410   if (!EnableMethodHandles)  return NULL;
2401   vmSymbols::SID name_id = vmSymbols::find_sid(name);
2402   assert(name_id != vmSymbols::NO_SID, "must be a known name");
2403   unsigned int hash  = invoke_method_table()->compute_hash(signature, name_id);
2404   int          index = invoke_method_table()->hash_to_index(hash);
2405   SymbolPropertyEntry* spe = invoke_method_table()->find_entry(index, hash, sign
2406   methodHandle non_cached_result;
2407   if (spe == NULL || spe->property_oop() == NULL) {
2408     spe = NULL;
2409     // Must create lots of stuff here, but outside of the SystemDictionary lock.
2410     if (THREAD->is_Compiler_thread())
2411       return NULL;              // do not attempt from within compiler
2412     bool for_invokeGeneric = (name_id == vmSymbols::VM_SYMBOL_ENUM_NAME(invokeGe
2413     if (AllowInvokeForInvokeGeneric && name_id == vmSymbols::VM_SYMBOL_ENUM_NAME
2414       for_invokeGeneric = true;
2415     bool found_on_bcp = false;
2416     Handle mt = find_method_handle_type(signature, accessing_klass,
2417                                         for_invokeGeneric,
2418                                         found_on_bcp, CHECK_NULL);
2419     KlassHandle  mh_klass = SystemDictionaryHandles::MethodHandle_klass();
2420     methodHandle m = methodOopDesc::make_invoke_method(mh_klass, name, signature
2421                                                        mt, CHECK_NULL);
2422     // Now grab the lock.  We might have to throw away the new method,
2423     // if a racing thread has managed to install one at the same time.
2424     if (found_on_bcp) {
2425       MutexLocker ml(SystemDictionary_lock, Thread::current());
2426       spe = invoke_method_table()->find_entry(index, hash, signature, name_id);
```

```
2427       if (spe == NULL)
2428         spe = invoke_method_table()->add_entry(index, hash, signature, name_id);
2429       if (spe->property_oop() == NULL)
2430         spe->set_property_oop(m());
2431     } else {
2432       non_cached_result = m;
2433     }
2434   }
2435   if (spe != NULL && spe->property_oop() != NULL) {
2436     assert(spe->property_oop()->is_method(), "");
2437     return (methodOop) spe->property_oop();
2438   } else {
2439     return non_cached_result();
2440   }
2441 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   34472 Wed Mar 30 07:00:18 2011
new/src/share/vm/classfile/systemDictionary.hpp
**********************************************************
```

```
/*
 * Copyright (c) 1997, 2011, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 *
 */

#ifndef SHARE_VM_CLASSFILE_SYSTEMDICTIONARY_HPP
#define SHARE_VM_CLASSFILE_SYSTEMDICTIONARY_HPP

#include "classfile/classFileStream.hpp"
#include "classfile/classLoader.hpp"
#include "oops/objArrayOop.hpp"
#include "oops/symbol.hpp"
#include "runtime/java.hpp"
#include "runtime/reflectionUtils.hpp"
#include "utilities/hashtable.hpp"

// The system dictionary stores all loaded classes and maps:
//
//   [class name,class loader] -> class   i.e.  [Symbol*,oop] -> klassOop
//
// Classes are loaded lazily. The default VM class loader is
// represented as NULL.

// The underlying data structure is an open hash table with a fixed number
// of buckets. During loading the loader object is locked, (for the VM loader
// a private lock object is used). Class loading can thus be done concurrently,
// but only by different loaders.
//
// During loading a placeholder (name, loader) is temporarily placed in
// a side data structure, and is used to detect ClassCircularityErrors
// and to perform verification during GC.  A GC can occur in the midst
// of class loading, as we call out to Java, have to take locks, etc.
//
// When class loading is finished, a new entry is added to the system
// dictionary and the place holder is removed. Note that the protection
// domain field of the system dictionary has not yet been filled in when
// the "real" system dictionary entry is created.
//
// Clients of this class who are interested in finding if a class has
// been completely loaded -- not classes in the process of being loaded --
// can read the SystemDictionary unlocked. This is safe because
//    - entries are only deleted at safepoints
//    - readers cannot come to a safepoint while actively examining
```



```
//         an entry  (an entry cannot be deleted from under a reader)
//    - entries must be fully formed before they are available to concurrent
//         readers (we must ensure write ordering)
//
// Note that placeholders are deleted at any time, as they are removed
// when a class is completely loaded. Therefore, readers as well as writers
// of placeholders must hold the SystemDictionary_lock.
//

class Dictionary;
class PlaceholderTable;
class LoaderConstraintTable;
class HashtableBucket;
class ResolutionErrorTable;
class SymbolPropertyTable;

// Certain classes are preloaded, such as java.lang.Object and java.lang.String.
// They are all "well-known", in the sense that no class loader is allowed
// to provide a different definition.
//
// These klasses must all have names defined in vmSymbols.

#define WK_KLASS_ENUM_NAME(kname)    kname##_knum

// Each well-known class has a short klass name (like object_klass),
// a vmSymbol name (like java_lang_Object), and a flag word
// that makes some minor distinctions, like whether the klass
// is preloaded, optional, release-specific, etc.
// The order of these definitions is significant; it is the order in which
// preloading is actually performed by initialize_preloaded_classes.

#define WK_KLASSES_DO(template)                                                \
  /* well-known classes */                                                     \
  template(Object_klass,                java_lang_Object,               Pre) \
  template(String_klass,                java_lang_String,               Pre) \
  template(Class_klass,                 java_lang_Class,                Pre) \
  template(Cloneable_klass,             java_lang_Cloneable,            Pre) \
  template(ClassLoader_klass,           java_lang_ClassLoader,          Pre) \
  template(Serializable_klass,          java_io_Serializable,           Pre) \
  template(System_klass,                java_lang_System,               Pre) \
  template(Throwable_klass,             java_lang_Throwable,            Pre) \
  template(Error_klass,                 java_lang_Error,                Pre) \
  template(ThreadDeath_klass,           java_lang_ThreadDeath,          Pre) \
  template(Exception_klass,             java_lang_Exception,            Pre) \
  template(RuntimeException_klass,      java_lang_RuntimeException,     Pre) \
  template(ProtectionDomain_klass,      java_security_ProtectionDomain, Pre) \
  template(AccessControlContext_klass,  java_security_AccessControlContext, Pre
  template(ClassNotFoundException_klass, java_lang_ClassNotFoundException, Pre)
  template(NoClassDefFoundError_klass,  java_lang_NoClassDefFoundError, Pre) \
  template(LinkageError_klass,          java_lang_LinkageError,         Pre) \
  template(ClassCastException_klass,    java_lang_ClassCastException,   Pre) \
  template(ArrayStoreException_klass,   java_lang_ArrayStoreException,  Pre) \
  template(VirtualMachineError_klass,   java_lang_VirtualMachineError,  Pre) \
  template(OutOfMemoryError_klass,      java_lang_OutOfMemoryError,     Pre) \
  template(StackOverflowError_klass,    java_lang_StackOverflowError,   Pre) \
  template(IllegalMonitorStateException_klass, java_lang_IllegalMonitorStateExce
  template(Reference_klass,             java_lang_ref_Reference,        Pre) \
                                                                               \
  /* Preload ref klasses and set reference types */                            \
  template(SoftReference_klass,         java_lang_ref_SoftReference,    Pre) \
  template(WeakReference_klass,         java_lang_ref_WeakReference,    Pre) \
  template(FinalReference_klass,        java_lang_ref_FinalReference,   Pre) \
  template(PhantomReference_klass,      java_lang_ref_PhantomReference, Pre) \
  template(Finalizer_klass,             java_lang_ref_Finalizer,        Pre) \
                                                                               \
  template(Thread_klass,                java_lang_Thread,               Pre) \
```

```
 129   template(ThreadGroup_klass,            java_lang_ThreadGroup,          Pre) \
 130   template(Properties_klass,             java_util_Properties,           Pre) \
 131   template(reflect_AccessibleObject_klass, java_lang_reflect_AccessibleObject, P
 132   template(reflect_Field_klass,          java_lang_reflect_Field,        Pre) \
 133   template(reflect_Method_klass,         java_lang_reflect_Method,       Pre) \
 134   template(reflect_Constructor_klass,    java_lang_reflect_Constructor,  Pre) \
 135                                                                               \
 136   /* NOTE: needed too early in bootstrapping process to have checks based on JDK
 137   /* Universe::is_gte_jdk14x_version() is not set up by this point. */        \
 138   /* It's okay if this turns out to be NULL in non-1.4 JDKs. */               \
 139   template(reflect_MagicAccessorImpl_klass,          sun_reflect_MagicAccessorIm
 140   template(reflect_MethodAccessorImpl_klass, sun_reflect_MethodAccessorImpl, Opt
 141   template(reflect_ConstructorAccessorImpl_klass, sun_reflect_ConstructorAccesso
 142   template(reflect_DelegatingClassLoader_klass, sun_reflect_DelegatingClassLoade
 143   template(reflect_ConstantPool_klass,  sun_reflect_ConstantPool,       Opt_Only
 144   template(reflect_UnsafeStaticFieldAccessorImpl_klass, sun_reflect_UnsafeStatic
 145                                                                               \
 146   /* support for dynamic typing; it's OK if these are NULL in earlier JDKs */ \
 147   template(MethodHandle_klass,           java_lang_invoke_MethodHandle,     Pre_
 148   template(MemberName_klass,             java_lang_invoke_MemberName,       Pre_
 149   template(MethodHandleImpl_klass,       sun_dyn_MethodHandleImpl,          Opt)
 150   template(MethodHandleNatives_klass,    java_lang_invoke_MethodHandleNatives, P
 151   template(AdapterMethodHandle_klass,    java_lang_invoke_AdapterMethodHandle, P
 152   template(BoundMethodHandle_klass,      java_lang_invoke_BoundMethodHandle, Pre
 153   template(DirectMethodHandle_klass,     java_lang_invoke_DirectMethodHandle, Pr
 154   template(MethodType_klass,             java_lang_invoke_MethodType,       Pre_
 155   template(MethodTypeForm_klass,         java_lang_invoke_MethodTypeForm,   Pre_
 156   template(WrongMethodTypeException_klass, java_lang_invoke_WrongMethodTypeExcep
 157   template(Linkage_klass,                java_lang_invoke_Linkage,          Opt)
 158   template(CallSite_klass,               java_lang_invoke_CallSite,         Pre_
 159   /* Note: MethodHandle must be first, and CallSite last in group */          \
 160                                                                               \
 161   template(StringBuffer_klass,           java_lang_StringBuffer,         Pre) \
 162   template(StringBuilder_klass,          java_lang_StringBuilder,        Pre) \
 163                                                                               \
 164   /* It's NULL in non-1.4 JDKs. */                                            \
 165   template(StackTraceElement_klass,      java_lang_StackTraceElement,    Opt) \
 166   /* Universe::is_gte_jdk14x_version() is not set up by this point. */        \
 167   /* It's okay if this turns out to be NULL in non-1.4 JDKs. */               \
 168   template(java_nio_Buffer_klass,        java_nio_Buffer,                Opt) \
 169                                                                               \
 170   /* If this class isn't present, it won't be referenced. */                  \
 171   template(sun_misc_AtomicLongCSImpl_klass, sun_misc_AtomicLongCSImpl,   Opt) \
 172                                                                               \
 173   template(sun_jkernel_DownloadManager_klass, sun_jkernel_DownloadManager, Opt_K
 174                                                                               \
 175   template(sun_misc_PostVMInitHook_klass, sun_misc_PostVMInitHook, Opt)       \
 176                                                                               \
 177   /* Preload boxing klasses */                                                \
 178   template(Boolean_klass,                java_lang_Boolean,              Pre) \
 179   template(Character_klass,              java_lang_Character,            Pre) \
 180   template(Float_klass,                  java_lang_Float,                Pre) \
 181   template(Double_klass,                 java_lang_Double,               Pre) \
 182   template(Byte_klass,                   java_lang_Byte,                 Pre) \
 183   template(Short_klass,                  java_lang_Short,                Pre) \
 184   template(Integer_klass,                java_lang_Integer,              Pre) \
 185   template(Long_klass,                   java_lang_Long,                 Pre) \
 186   /*end*/


 189 class SystemDictionary : AllStatic {
 190   friend class VMStructs;
 191   friend class CompactingPermGenGen;
 192   friend class SystemDictionaryHandles;
 193   NOT_PRODUCT(friend class instanceKlassKlass;)
```



```
 195  public:
 196   enum WKID {
 197     NO_WKID = 0,

 199     #define WK_KLASS_ENUM(name, ignore_s, ignore_o) WK_KLASS_ENUM_NAME(name),
 200     WK_KLASSES_DO(WK_KLASS_ENUM)
 201     #undef WK_KLASS_ENUM

 203     WKID_LIMIT,

 205     FIRST_WKID = NO_WKID + 1
 206   };

 208   enum InitOption {
 209     Pre,                        // preloaded; error if not present
 210     Pre_JSR292,                 // preloaded if EnableInvokeDynamic
 210     Pre_JSR292,                 // preloaded if EnableMethodHandles

 212     // Order is significant.  Options before this point require resolve_or_fail.
 213     // Options after this point will use resolve_or_null instead.

 215     Opt,                        // preload tried; NULL if not present
 216     Opt_Only_JDK14NewRef,       // preload tried; use only with NewReflection
 217     Opt_Only_JDK15,             // preload tried; use only with JDK1.5+
 218     Opt_Kernel,                 // preload tried only #ifdef KERNEL
 219     OPTION_LIMIT,
 220     CEIL_LG_OPTION_LIMIT = 4    // OPTION_LIMIT <= (1<<CEIL_LG_OPTION_LIMIT)
 221   };


 224   // Returns a class with a given class name and class loader.  Loads the
 225   // class if needed. If not found a NoClassDefFoundError or a
 226   // ClassNotFoundException is thrown, depending on the value on the
 227   // throw_error flag.  For most uses the throw_error argument should be set
 228   // to true.

 230   static klassOop resolve_or_fail(Symbol* class_name, Handle class_loader, Handl
 231   // Convenient call for null loader and protection domain.
 232   static klassOop resolve_or_fail(Symbol* class_name, bool throw_error, TRAPS);
 233 private:
 234   // handle error translation for resolve_or_null results
 235   static klassOop handle_resolution_exception(Symbol* class_name, Handle class_l

 237 public:

 239   // Returns a class with a given class name and class loader.
 240   // Loads the class if needed. If not found NULL is returned.
 241   static klassOop resolve_or_null(Symbol* class_name, Handle class_loader, Handl
 242   // Version with null loader and protection domain
 243   static klassOop resolve_or_null(Symbol* class_name, TRAPS);

 245   // Resolve a superclass or superinterface. Called from ClassFileParser,
 246   // parse_interfaces, resolve_instance_class_or_null, load_shared_class
 247   // "child_name" is the class whose super class or interface is being resolved.
 248   static klassOop resolve_super_or_fail(Symbol* child_name,
 249                                         Symbol* class_name,
 250                                         Handle class_loader,
 251                                         Handle protection_domain,
 252                                         bool is_superclass,
 253                                         TRAPS);

 255   // Parse new stream. This won't update the system dictionary or
 256   // class hierarchy, simply parse the stream. Used by JVMTI RedefineClasses.
 257   static klassOop parse_stream(Symbol* class_name,
 258                                Handle class_loader,
 259                                Handle protection_domain,
```

```
260                               ClassFileStream* st,
261                               TRAPS) {
262     KlassHandle nullHandle;
263     return parse_stream(class_name, class_loader, protection_domain, st, nullHan
264   }
265   static klassOop parse_stream(Symbol* class_name,
266                               Handle class_loader,
267                               Handle protection_domain,
268                               ClassFileStream* st,
269                               KlassHandle host_klass,
270                               GrowableArray<Handle>* cp_patches,
271                               TRAPS);

273   // Resolve from stream (called by jni_DefineClass and JVM_DefineClass)
274   static klassOop resolve_from_stream(Symbol* class_name, Handle class_loader,
275                                       Handle protection_domain,
276                                       ClassFileStream* st, bool verify, TRAPS);

278   // Lookup an already loaded class. If not found NULL is returned.
279   static klassOop find(Symbol* class_name, Handle class_loader, Handle protectio

281   // Lookup an already loaded instance or array class.
282   // Do not make any queries to class loaders; consult only the cache.
283   // If not found NULL is returned.
284   static klassOop find_instance_or_array_klass(Symbol* class_name,
285                                                Handle class_loader,
286                                                Handle protection_domain,
287                                                TRAPS);

289   // If the given name is known to vmSymbols, return the well-know klass:
290   static klassOop find_well_known_klass(Symbol* class_name);

292   // Lookup an instance or array class that has already been loaded
293   // either into the given class loader, or else into another class
294   // loader that is constrained (via loader constraints) to produce
295   // a consistent class.  Do not take protection domains into account.
296   // Do not make any queries to class loaders; consult only the cache.
297   // Return NULL if the class is not found.
298   //
299   // This function is a strict superset of find_instance_or_array_klass.
300   // This function (the unchecked version) makes a conservative prediction
301   // of the result of the checked version, assuming successful lookup.
302   // If both functions return non-null, they must return the same value.
303   // Also, the unchecked version may sometimes be non-null where the
304   // checked version is null.  This can occur in several ways:
305   //   1. No query has yet been made to the class loader.
306   //   2. The class loader was queried, but chose not to delegate.
307   //   3. ClassLoader.checkPackageAccess rejected a proposed protection domain.
308   //   4. Loading was attempted, but there was a linkage error of some sort.
309   // In all of these cases, the loader constraints on this type are
310   // satisfied, and it is safe for classes in the given class loader
311   // to manipulate strongly-typed values of the found class, subject
312   // to local linkage and access checks.
313   static klassOop find_constrained_instance_or_array_klass(Symbol* class_name,
314                                                            Handle class_loader,
315                                                            TRAPS);

317   // Iterate over all klasses in dictionary
318   //   Just the classes from defining class loaders
319   static void classes_do(void f(klassOop));
320   // Added for initialize_itable_for_klass to handle exceptions
321   static void classes_do(void f(klassOop, TRAPS), TRAPS);
322   //   All classes, and their class loaders
323   static void classes_do(void f(klassOop, oop));
324   //   All classes, and their class loaders
325   //   (added for helpers that use HandleMarks and ResourceMarks)
```



```
326   static void classes_do(void f(klassOop, oop, TRAPS), TRAPS);
327   // All entries in the placeholder table and their class loaders
328   static void placeholders_do(void f(Symbol*, oop));

330   // Iterate over all methods in all klasses in dictionary
331   static void methods_do(void f(methodOop));

333   // Garbage collection support

335   // This method applies "blk->do_oop" to all the pointers to "system"
336   // classes and loaders.
337   static void always_strong_oops_do(OopClosure* blk);
338   static void always_strong_classes_do(OopClosure* blk);
339   // This method applies "blk->do_oop" to all the placeholders.
340   static void placeholders_do(OopClosure* blk);

342   // Unload (that is, break root links to) all unmarked classes and
343   // loaders.  Returns "true" iff something was unloaded.
344   static bool do_unloading(BoolObjectClosure* is_alive);

346   // Applies "f->do_oop" to all root oops in the system dictionary.
347   static void oops_do(OopClosure* f);

349   // System loader lock
350   static oop system_loader_lock()           { return _system_loader_lock_obj; }

352 private:
353   //    Traverses preloaded oops: various system classes.  These are
354   //    guaranteed to be in the perm gen.
355   static void preloaded_oops_do(OopClosure* f);
356   static void lazily_loaded_oops_do(OopClosure* f);

358 public:
359   // Sharing support.
360   static void reorder_dictionary();
361   static void copy_buckets(char** top, char* end);
362   static void copy_table(char** top, char* end);
363   static void reverse();
364   static void set_shared_dictionary(HashtableBucket* t, int length,
365                                     int number_of_entries);
366   // Printing
367   static void print()                    PRODUCT_RETURN;
368   static void print_class_statistics()  PRODUCT_RETURN;
369   static void print_method_statistics() PRODUCT_RETURN;

371   // Number of contained klasses
372   // This is both fully loaded classes and classes in the process
373   // of being loaded
374   static int number_of_classes();

376   // Monotonically increasing counter which grows as classes are
377   // loaded or modifications such as hot-swapping or setting/removing
378   // of breakpoints are performed
379   static inline int number_of_modifications()     { assert_locked_or_safepoint(C
380   // Needed by evolution and breakpoint code
381   static inline void notice_modification()        { assert_locked_or_safepoint(C

383   // Verification
384   static void verify();

386 #ifdef ASSERT
387   static bool is_internal_format(Symbol* class_name);
388 #endif

390   // Verify class is in dictionary
391   static void verify_obj_klass_present(Handle obj,
```

```
392                                     Symbol* class_name,
393                                     Handle class_loader);

395   // Initialization
396   static void initialize(TRAPS);

398   // Fast access to commonly used classes (preloaded)
399   static klassOop check_klass(klassOop k) {
400     assert(k != NULL, "preloaded klass not initialized");
401     return k;
402   }

404   static klassOop check_klass_Pre(klassOop k) { return check_klass(k); }
405   static klassOop check_klass_Pre_JSR292(klassOop k) { return EnableInvokeDynami
406   static klassOop check_klass_Opt(klassOop k) { return k; }
407   static klassOop check_klass_Opt_Kernel(klassOop k) { return k; } //== Opt
408   static klassOop check_klass_Opt_Only_JDK15(klassOop k) {
409     assert(JDK_Version::is_gte_jdk15x_version(), "JDK 1.5 only");
410     return k;
411   }
412   static klassOop check_klass_Opt_Only_JDK14NewRef(klassOop k) {
413     assert(JDK_Version::is_gte_jdk14x_version() && UseNewReflection, "JDK 1.4 on
414     // despite the optional loading, if you use this it must be present:
415     return check_klass(k);
416   }

418   static bool initialize_wk_klass(WKID id, int init_opt, TRAPS);
419   static void initialize_wk_klasses_until(WKID limit_id, WKID &start_id, TRAPS);
420   static void initialize_wk_klasses_through(WKID end_id, WKID &start_id, TRAPS)
421     int limit = (int)end_id + 1;
422     initialize_wk_klasses_until((WKID) limit, start_id, THREAD);
423   }

425   static Symbol* find_backup_symbol(Symbol* symbol, const char* from_prefix, con

427 public:
428   #define WK_KLASS_DECLARE(name, ignore_symbol, option) \
429     static klassOop name() { return check_klass_##option(_well_known_klasses[WK_
430   WK_KLASSES_DO(WK_KLASS_DECLARE);
431   #undef WK_KLASS_DECLARE

433   // Local definition for direct access to the private array:
434   #define WK_KLASS(name) _well_known_klasses[SystemDictionary::WK_KLASS_ENUM_NAM

436   static klassOop box_klass(BasicType t) {
437     assert((uint)t < T_VOID+1, "range check");
438     return check_klass(_box_klasses[t]);
439   }
440   static BasicType box_klass_type(klassOop k);  // inverse of box_klass

442   // methods returning lazily loaded klasses
443   // The corresponding method to load the class must be called before calling th
444   static klassOop abstract_ownable_synchronizer_klass() { return check_klass(_ab

446   static void load_abstract_ownable_synchronizer_klass(TRAPS);

448   static Symbol* find_backup_class_name(Symbol* class_name_symbol);
449   static Symbol* find_backup_signature(Symbol* signature_symbol);

451 private:
452   // Tells whether ClassLoader.loadClassInternal is present
453   static bool has_loadClassInternal()       { return _has_loadClassInternal; }

455 public:
456   // Tells whether ClassLoader.checkPackageAccess is present
457   static bool has_checkPackageAccess()      { return _has_checkPackageAccess; }
```



```
459   static bool Class_klass_loaded()          { return WK_KLASS(Class_klass) != NU
460   static bool Cloneable_klass_loaded()      { return WK_KLASS(Cloneable_klass) !

462   // Returns default system loader
463   static oop java_system_loader();

465   // Compute the default system loader
466   static void compute_java_system_loader(TRAPS);

468 private:
469   // Mirrors for primitive classes (created eagerly)
470   static oop check_mirror(oop m) {
471     assert(m != NULL, "mirror not initialized");
472     return m;
473   }

475 public:
476   // Note:  java_lang_Class::primitive_type is the inverse of java_mirror

478   // Check class loader constraints
479   static bool add_loader_constraint(Symbol* name, Handle loader1,
480                                     Handle loader2, TRAPS);
481   static char* check_signature_loaders(Symbol* signature, Handle loader1,
482                                        Handle loader2, bool is_method, TRAPS);

484   // JSR 292
485   // find the java.lang.invoke.MethodHandles::invoke method for a given signatur
486   static methodOop find_method_handle_invoke(Symbol* name,
487                                              Symbol* signature,
488                                              KlassHandle accessing_klass,
489                                              TRAPS);
490   // ask Java to compute a java.lang.invoke.MethodType object for a given signat
491   static Handle    find_method_handle_type(Symbol* signature,
492                                            KlassHandle accessing_klass,
493                                            bool for_invokeGeneric,
494                                            bool& return_bcp_flag,
495                                            TRAPS);
496   // ask Java to compute a java.lang.invoke.MethodHandle object for a given CP e
497   static Handle    link_method_handle_constant(KlassHandle caller,
498                                                int ref_kind, //e.g., JVM_REF_inv
499                                                KlassHandle callee,
500                                                Symbol* name,
501                                                Symbol* signature,
502                                                TRAPS);
503   // ask Java to create a dynamic call site, while linking an invokedynamic op
504   static Handle    make_dynamic_call_site(Handle bootstrap_method,
505                                           // Callee information:
506                                           Symbol* name,
507                                           methodHandle signature_invoker,
508                                           Handle info,
509                                           // Caller information:
510                                           methodHandle caller_method,
511                                           int caller_bci,
512                                           TRAPS);

514   // coordinate with Java about bootstrap methods
515   static Handle    find_bootstrap_method(methodHandle caller_method,
516                                          int caller_bci,  // N.B. must be an inv
517                                          int cache_index, // must be correspondi
518                                          Handle &argument_info_result, // static
519                                          TRAPS);

521   // Utility for printing loader "name" as part of tracing constraints
522   static const char* loader_name(oop loader) {
523     return ((loader) == NULL ? "<bootloader>" :
```

```
524             instanceKlass::cast((loader)->klass())->name()->as_C_string() );
525   }

527   // Record the error when the first attempt to resolve a reference from a const
528   // pool entry to a class fails.
529   static void add_resolution_error(constantPoolHandle pool, int which, Symbol* e
530   static Symbol* find_resolution_error(constantPoolHandle pool, int which);

532  private:

534   enum Constants {
535     _loader_constraint_size = 107,                    // number of entries in c
536     _resolution_error_size  = 107,                    // number of entries in r
537     _invoke_method_size     = 139,                    // number of entries in i
538     _nof_buckets            = 1009                    // number of buckets in h
539   };


542   // Static variables

544   // Hashtable holding loaded classes.
545   static Dictionary*            _dictionary;

547   // Hashtable holding placeholders for classes being loaded.
548   static PlaceholderTable*      _placeholders;

550   // Hashtable holding classes from the shared archive.
551   static Dictionary*            _shared_dictionary;

553   // Monotonically increasing counter which grows with
554   // _number_of_classes as well as hot-swapping and breakpoint setting
555   // and removal.
556   static int                    _number_of_modifications;

558   // Lock object for system class loader
559   static oop                    _system_loader_lock_obj;

561   // Constraints on class loaders
562   static LoaderConstraintTable*  _loader_constraints;

564   // Resolution errors
565   static ResolutionErrorTable*   _resolution_errors;

567   // Invoke methods (JSR 292)
568   static SymbolPropertyTable*    _invoke_method_table;

570 public:
571   // for VM_CounterDecay iteration support
572   friend class CounterDecay;
573   static klassOop try_get_next_class();

575 private:
576   static void validate_protection_domain(instanceKlassHandle klass,
577                                          Handle class_loader,
578                                          Handle protection_domain, TRAPS);

580   friend class VM_PopulateDumpSharedSpace;
581   friend class TraversePlaceholdersClosure;
582   static Dictionary*         dictionary() { return _dictionary; }
583   static Dictionary*         shared_dictionary() { return _shared_dictionary; }
584   static PlaceholderTable*   placeholders() { return _placeholders; }
585   static LoaderConstraintTable* constraints() { return _loader_constraints; }
586   static ResolutionErrorTable* resolution_errors() { return _resolution_errors;
587   static SymbolPropertyTable* invoke_method_table() { return _invoke_method_tabl

589   // Basic loading operations
```



```
590   static klassOop resolve_instance_class_or_null(Symbol* class_name, Handle clas
591   static klassOop resolve_array_class_or_null(Symbol* class_name, Handle class_l
592   static instanceKlassHandle handle_parallel_super_load(Symbol* class_name, Symb
593   // Wait on SystemDictionary_lock; unlocks lockObject before
594   // waiting; relocks lockObject with correct recursion count
595   // after waiting, but before reentering SystemDictionary_lock
596   // to preserve lock order semantics.
597   static void double_lock_wait(Handle lockObject, TRAPS);
598   static void define_instance_class(instanceKlassHandle k, TRAPS);
599   static instanceKlassHandle find_or_define_instance_class(Symbol* class_name,
600                                                            Handle class_loader,
601                                                            instanceKlassHandle k, TRAPS);
602   static instanceKlassHandle load_shared_class(Symbol* class_name,
603                                                Handle class_loader, TRAPS);
604   static instanceKlassHandle load_shared_class(instanceKlassHandle ik,
605                                                Handle class_loader, TRAPS);
606   static instanceKlassHandle load_instance_class(Symbol* class_name, Handle clas
607   static Handle compute_loader_lock_object(Handle class_loader, TRAPS);
608   static void check_loader_lock_contention(Handle loader_lock, TRAPS);
609   static bool is_parallelCapable(Handle class_loader);
610   static bool is_parallelDefine(Handle class_loader);

612   static klassOop find_shared_class(Symbol* class_name);

614   // Setup link to hierarchy
615   static void add_to_hierarchy(instanceKlassHandle k, TRAPS);

617 private:
618   // We pass in the hashtable index so we can calculate it outside of
619   // the SystemDictionary_lock.

621   // Basic find on loaded classes
622   static klassOop find_class(int index, unsigned int hash,
623                              Symbol* name, Handle loader);
624   static klassOop find_class(Symbol* class_name, Handle class_loader);

626   // Basic find on classes in the midst of being loaded
627   static Symbol* find_placeholder(Symbol* name, Handle loader);

629   // Updating entry in dictionary
630   // Add a completely loaded class
631   static void add_klass(int index, Symbol* class_name,
632                         Handle class_loader, KlassHandle obj);

634   // Add a placeholder for a class being loaded
635   static void add_placeholder(int index,
636                               Symbol* class_name,
637                               Handle class_loader);
638   static void remove_placeholder(int index,
639                                  Symbol* class_name,
640                                  Handle class_loader);

642   // Performs cleanups after resolve_super_or_fail. This typically needs
643   // to be called on failure.
644   // Won't throw, but can block.
645   static void resolution_cleanups(Symbol* class_name,
646                                   Handle class_loader,
647                                   TRAPS);

649   // Initialization
650   static void initialize_preloaded_classes(TRAPS);

652   // Class loader constraints
653   static void check_constraints(int index, unsigned int hash,
654                                 instanceKlassHandle k, Handle loader,
655                                 bool defining, TRAPS);
```

```
 656   static void update_dictionary(int d_index, unsigned int d_hash,
 657                                 int p_index, unsigned int p_hash,
 658                                 instanceKlassHandle k, Handle loader, TRAPS);

 660   // Variables holding commonly used klasses (preloaded)
 661   static klassOop _well_known_klasses[];

 663   // Lazily loaded klasses
 664   static volatile klassOop _abstract_ownable_synchronizer_klass;

 666   // table of box klasses (int_klass, etc.)
 667   static klassOop _box_klasses[T_VOID+1];

 669   static oop  _java_system_loader;

 671   static bool _has_loadClassInternal;
 672   static bool _has_checkPackageAccess;
 673 };
```

*unchanged_portion_omitted_*

****
54256 Wed Mar 30 07:00:19 2011
new/src/share/vm/interpreter/linkResolver.cpp
****
_____***unchanged_portion_omitted_***

```
 172 //------------------------------------------------------------------------------
 173 // Method resolution
 174 //
 175 // According to JVM spec. $5.4.3c & $5.4.3d

 177 void LinkResolver::lookup_method_in_klasses(methodHandle& result, KlassHandle kl
 178   methodOop result_oop = klass->uncached_lookup_method(name, signature);
 179   if (EnableInvokeDynamic && result_oop != NULL) {
 179   if (EnableMethodHandles && result_oop != NULL) {
 180     switch (result_oop->intrinsic_id()) {
 181     case vmIntrinsics::_invokeExact:
 182     case vmIntrinsics::_invokeGeneric:
 183     case vmIntrinsics::_invokeDynamic:
 184       // Do not link directly to these.  The VM must produce a synthetic one usi
 185       return;
 186     }
 187   }
 188   result = methodHandle(THREAD, result_oop);
 189 }
```
_____***unchanged_portion_omitted_***

```
 213 void LinkResolver::lookup_implicit_method(methodHandle& result,
 214                                           KlassHandle klass, Symbol* name, Symbo
 215                                           KlassHandle current_klass,
 216                                           TRAPS) {
 217   if (EnableInvokeDynamic &&
 217   if (EnableMethodHandles &&
 218       klass() == SystemDictionary::MethodHandle_klass() &&
 219       methodOopDesc::is_method_handle_invoke_name(name)) {
 220     if (!THREAD->is_Compiler_thread() && !MethodHandles::enabled()) {
 221       // Make sure the Java part of the runtime has been booted up.
 222       klassOop natives = SystemDictionary::MethodHandleNatives_klass();
 223       if (natives == NULL || instanceKlass::cast(natives)->is_not_initialized())
 224         Symbol* natives_name = vmSymbols::java_lang_invoke_MethodHandleNatives()
 225         if (natives != NULL && AllowTransitionalJSR292)  natives_name = Klass::c
 226         SystemDictionary::resolve_or_fail(natives_name,
 227                                           Handle(),
 228                                           Handle(),
 229                                           true,
 230                                           CHECK);
 231       }
 232     }
 233     methodOop result_oop = SystemDictionary::find_method_handle_invoke(name,
 234                                                                         signature
 235                                                                         current_k
 236                                                                         CHECK);
 237     if (result_oop != NULL) {
 238       assert(result_oop->is_method_handle_invoke() && result_oop->signature() ==
 239       result = methodHandle(THREAD, result_oop);
 240     }
 241   }
 242 }
```
_____***unchanged_portion_omitted_***

****
19542 Wed Mar 30 07:00:20 2011
new/src/share/vm/oops/constantPoolKlass.cpp
****

_____***unchanged_portion_omitted_***

```
 285 void constantPoolKlass::oop_push_contents(PSPromotionManager* pm, oop obj) {
 286   assert(obj->is_constantPool(), "should be constant pool");
 287   constantPoolOop cp = (constantPoolOop) obj;
 288   if (cp->tags() != NULL &&
 289       (!JavaObjectsInPerm || (EnableInvokeDynamic && cp->has_pseudo_string()))) {
 289       (!JavaObjectsInPerm || (AnonymousClasses && cp->has_pseudo_string()))) {
 290     for (int i = 1; i < cp->length(); ++i) {
 291       if (cp->tag_at(i).is_string()) {
 292         oop* base = cp->obj_at_addr_raw(i);
 293         if (PSScavenge::should_scavenge(base)) {
 294           pm->claim_or_forward_depth(base);
 295         }
 296       }
 297     }
 298   }
 299 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   34424 Wed Mar 30 07:00:21 2011
new/src/share/vm/oops/constantPoolOop.hpp
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 1997, 2011, Oracle and/or its affiliates. All rights reserved.
   2  * Copyright (c) 1997, 2010, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.
   8  *
   9  * This code is distributed in the hope that it will be useful, but WITHOUT
  10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  12  * version 2 for more details (a copy is included in the LICENSE file that
  13  * accompanied this code).
  14  *
  15  * You should have received a copy of the GNU General Public License version
  16  * 2 along with this work; if not, write to the Free Software Foundation,
  17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  18  *
  19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  20  * or visit www.oracle.com if you need additional information or have any
  21  * questions.
  22  *
  23  */

  25 #ifndef SHARE_VM_OOPS_CONSTANTPOOLOOP_HPP
  26 #define SHARE_VM_OOPS_CONSTANTPOOLOOP_HPP

  28 #include "oops/arrayOop.hpp"
  29 #include "oops/cpCacheOop.hpp"
  30 #include "oops/symbol.hpp"
  31 #include "oops/typeArrayOop.hpp"
  32 #include "utilities/constantTag.hpp"
  33 #ifdef TARGET_ARCH_x86
  34 # include "bytes_x86.hpp"
  35 #endif
  36 #ifdef TARGET_ARCH_sparc
  37 # include "bytes_sparc.hpp"
  38 #endif
  39 #ifdef TARGET_ARCH_zero
  40 # include "bytes_zero.hpp"
  41 #endif
  42 #ifdef TARGET_ARCH_arm
  43 # include "bytes_arm.hpp"
  44 #endif
  45 #ifdef TARGET_ARCH_ppc
  46 # include "bytes_ppc.hpp"
  47 #endif

  49 // A constantPool is an array containing class constants as described in the
  50 // class file.
  51 //
  52 // Most of the constant pool entries are written during class parsing, which
  53 // is safe.  For klass and string types, the constant pool entry is
  54 // modified when the entry is resolved.  If a klass or string constant pool
  55 // entry is read without a lock, only the resolved state guarantees that
  56 // the entry in the constant pool is a klass or String object and
  57 // not a Symbol*.

  59 class SymbolHashMap;

  61 class CPSlot VALUE_OBJ_CLASS_SPEC {
```



```
  62   intptr_t _ptr;
  63  public:
  64   CPSlot(intptr_t ptr): _ptr(ptr) {}
  65   CPSlot(void* ptr): _ptr((intptr_t)ptr) {}
  66   CPSlot(oop ptr): _ptr((intptr_t)ptr) {}
  67   CPSlot(Symbol* ptr): _ptr((intptr_t)ptr | 1) {}

  69   intptr_t value()   { return _ptr; }
  70   bool is_oop()      { return (_ptr & 1) == 0; }
  71   bool is_metadata() { return (_ptr & 1) == 1; }

  73   oop get_oop() {
  74     assert(is_oop(), "bad call");
  75     return oop(_ptr);
  76   }
  77   Symbol* get_symbol() {
  78     assert(is_metadata(), "bad call");
  79     return (Symbol*)(_ptr & ~1);
  80   }
  81 };

  83 class constantPoolOopDesc : public oopDesc {
  84   friend class VMStructs;
  85   friend class BytecodeInterpreter;  // Directly extracts an oop in the pool for
  86  private:
  87   typeArrayOop         _tags; // the tag array describing the constant pool's co
  88   constantPoolCacheOop _cache;         // the cache holding interpreter runtime
  89   klassOop             _pool_holder;   // the corresponding class
  90   typeArrayOop         _operands;      // for variable-sized (InvokeDynamic) nod
  91   int                  _flags;         // a few header bits to describe contents
  92   int                  _length; // number of elements in the array
  93   volatile bool        _is_conc_safe; // if true, safe for concurrent
  94                                       // GC processing
  95   // only set to non-zero if constant pool is merged by RedefineClasses
  96   int                  _orig_length;

  98   void set_tags(typeArrayOop tags)             { oop_store_without_check((oop*)&
  99   void tag_at_put(int which, jbyte t)          { tags()->byte_at_put(which, t);
 100   void release_tag_at_put(int which, jbyte t)  { tags()->release_byte_at_put(whi

 102   void set_operands(typeArrayOop operands)     { oop_store_without_check((oop*)&

 104   enum FlagBit {
 105     FB_has_invokedynamic = 1,
 106     FB_has_pseudo_string = 2
 107   };

 109   int flags() const                       { return _flags; }
 110   void set_flags(int f)                   { _flags = f; }
 111   bool flag_at(FlagBit fb) const          { return (_flags & (1 << (int)fb)) !
 112   void set_flag_at(FlagBit fb);
 113   // no clear_flag_at function; they only increase

 115  private:
 116   intptr_t* base() const { return (intptr_t*) (((char*) this) + sizeof(constantP
 117   oop* tags_addr()       { return (oop*)&_tags; }
 118   oop* cache_addr()      { return (oop*)&_cache; }
 119   oop* operands_addr()   { return (oop*)&_operands; }

 121   CPSlot slot_at(int which) {
 122     assert(is_within_bounds(which), "index out of bounds");
 123     // There's a transitional value of zero when converting from
 124     // Symbol->0->Klass for G1 when resolving classes and strings.
 125     // wait for the value to be non-zero (this is temporary)
 126     volatile intptr_t adr = (intptr_t)OrderAccess::load_ptr_acquire(obj_at_addr_
 127     if (adr == 0 && which != 0) {
```

```
128       constantTag t = tag_at(which);
129       if (t.is_unresolved_klass() || t.is_klass() ||
130           t.is_unresolved_string() || t.is_string()) {
131         while ((adr = (intptr_t)OrderAccess::load_ptr_acquire(obj_at_addr_raw(wh
132       }
133     }
134     return CPSlot(adr);
135   }

137   void slot_at_put(int which, CPSlot s) const {
138     assert(is_within_bounds(which), "index out of bounds");
139     *(intptr_t*)&base()[which] = s.value();
140   }
141   oop* obj_at_addr_raw(int which) const {
142     assert(is_within_bounds(which), "index out of bounds");
143     return (oop*) &base()[which];
144   }

146   void obj_at_put_without_check(int which, oop o) {
147     assert(is_within_bounds(which), "index out of bounds");
148     oop_store_without_check((volatile oop *)obj_at_addr_raw(which), o);
149   }

151   void obj_at_put(int which, oop o) const {
152     assert(is_within_bounds(which), "index out of bounds");
153     oop_store((volatile oop*)obj_at_addr_raw(which), o);
154   }

156   jint* int_at_addr(int which) const {
157     assert(is_within_bounds(which), "index out of bounds");
158     return (jint*) &base()[which];
159   }

161   jlong* long_at_addr(int which) const {
162     assert(is_within_bounds(which), "index out of bounds");
163     return (jlong*) &base()[which];
164   }

166   jfloat* float_at_addr(int which) const {
167     assert(is_within_bounds(which), "index out of bounds");
168     return (jfloat*) &base()[which];
169   }

171   jdouble* double_at_addr(int which) const {
172     assert(is_within_bounds(which), "index out of bounds");
173     return (jdouble*) &base()[which];
174   }

176  public:
177   typeArrayOop tags() const                 { return _tags; }
178   typeArrayOop operands() const             { return _operands; }

180   bool has_pseudo_string() const            { return flag_at(FB_has_pseudo_strin
181   bool has_invokedynamic() const            { return flag_at(FB_has_invokedynami
182   void set_pseudo_string()                  {    set_flag_at(FB_has_pseudo_strin
183   void set_invokedynamic()                  {    set_flag_at(FB_has_invokedynami

185   // Klass holding pool
186   klassOop pool_holder() const              { return _pool_holder; }
187   void set_pool_holder(klassOop k)          { oop_store_without_check((oop*)&_po
188   oop* pool_holder_addr()                   { return (oop*)&_pool_holder; }

190   // Interpreter runtime support
191   constantPoolCacheOop cache() const        { return _cache; }
192   void set_cache(constantPoolCacheOop cache){ oop_store((oop*)&_cache, cache); }
```



```
194   // Assembly code support
195   static int tags_offset_in_bytes()         { return offset_of(constantPoolOopDe
196   static int cache_offset_in_bytes()        { return offset_of(constantPoolOopDe
197   static int operands_offset_in_bytes()     { return offset_of(constantPoolOopDe
198   static int pool_holder_offset_in_bytes()  { return offset_of(constantPoolOopDe

200   // Storing constants

202   void klass_at_put(int which, klassOop k) {
203     // Overwrite the old index with a GC friendly value so
204     // that if G1 looks during the transition during oop_store it won't
205     // assert the symbol is not an oop.
206     *obj_at_addr_raw(which) = NULL;
207     assert(k != NULL, "resolved class shouldn't be null");
208     obj_at_put_without_check(which, k);
209     // The interpreter assumes when the tag is stored, the klass is resolved
210     // and the klassOop is a klass rather than a Symbol*, so we need
211     // hardware store ordering here.
212     release_tag_at_put(which, JVM_CONSTANT_Class);
213     if (UseConcMarkSweepGC) {
214       // In case the earlier card-mark was consumed by a concurrent
215       // marking thread before the tag was updated, redirty the card.
216       obj_at_put_without_check(which, k);
217     }
218   }

220   // For temporary use while constructing constant pool
221   void klass_index_at_put(int which, int name_index) {
222     tag_at_put(which, JVM_CONSTANT_ClassIndex);
223     *int_at_addr(which) = name_index;
224   }

226   // Temporary until actual use
227   void unresolved_klass_at_put(int which, Symbol* s) {
228     release_tag_at_put(which, JVM_CONSTANT_UnresolvedClass);
229     slot_at_put(which, s);
230   }

232   void method_handle_index_at_put(int which, int ref_kind, int ref_index) {
233     tag_at_put(which, JVM_CONSTANT_MethodHandle);
234     *int_at_addr(which) = ((jint) ref_index<<16) | ref_kind;
235   }

237   void method_type_index_at_put(int which, int ref_index) {
238     tag_at_put(which, JVM_CONSTANT_MethodType);
239     *int_at_addr(which) = ref_index;
240   }

242   void invoke_dynamic_at_put(int which, int bootstrap_specifier_index, int name_
243     tag_at_put(which, JVM_CONSTANT_InvokeDynamic);
244     *int_at_addr(which) = ((jint) name_and_type_index<<16) | bootstrap_specifier
245   }

247   void invoke_dynamic_trans_at_put(int which, int bootstrap_method_index, int na
248     tag_at_put(which, JVM_CONSTANT_InvokeDynamicTrans);
249     *int_at_addr(which) = ((jint) name_and_type_index<<16) | bootstrap_method_in
250     assert(AllowTransitionalJSR292, "");
251   }

253   // Temporary until actual use
254   void unresolved_string_at_put(int which, Symbol* s) {
255     release_tag_at_put(which, JVM_CONSTANT_UnresolvedString);
256     slot_at_put(which, s);
257   }

259   void int_at_put(int which, jint i) {
```

```
260     tag_at_put(which, JVM_CONSTANT_Integer);
261     *int_at_addr(which) = i;
262   }
263
264   void long_at_put(int which, jlong l) {
265     tag_at_put(which, JVM_CONSTANT_Long);
266     // *long_at_addr(which) = l;
267     Bytes::put_native_u8((address)long_at_addr(which), *((u8*) &l));
268   }
269
270   void float_at_put(int which, jfloat f) {
271     tag_at_put(which, JVM_CONSTANT_Float);
272     *float_at_addr(which) = f;
273   }
274
275   void double_at_put(int which, jdouble d) {
276     tag_at_put(which, JVM_CONSTANT_Double);
277     // *double_at_addr(which) = d;
278     // u8 temp = *(u8*) &d;
279     Bytes::put_native_u8((address) double_at_addr(which), *((u8*) &d));
280   }
281
282   Symbol** symbol_at_addr(int which) const {
283     assert(is_within_bounds(which), "index out of bounds");
284     return (Symbol**) &base()[which];
285   }
286
287   void symbol_at_put(int which, Symbol* s) {
288     assert(s->refcount() != 0, "should have nonzero refcount");
289     tag_at_put(which, JVM_CONSTANT_Utf8);
290     slot_at_put(which, s);
291   }
292
293   void string_at_put(int which, oop str) {
294     // Overwrite the old index with a GC friendly value so
295     // that if G1 looks during the transition during oop_store it won't
296     // assert the symbol is not an oop.
297     *obj_at_addr_raw(which) = NULL;
298     assert(str != NULL, "resolved string shouldn't be null");
299     obj_at_put(which, str);
300     release_tag_at_put(which, JVM_CONSTANT_String);
301     if (UseConcMarkSweepGC) {
302       // In case the earlier card-mark was consumed by a concurrent
303       // marking thread before the tag was updated, redirty the card.
304       obj_at_put_without_check(which, str);
305     }
306   }
307
308   void object_at_put(int which, oop str) {
309     obj_at_put(which, str);
310     release_tag_at_put(which, JVM_CONSTANT_Object);
311     if (UseConcMarkSweepGC) {
312       // In case the earlier card-mark was consumed by a concurrent
313       // marking thread before the tag was updated, redirty the card.
314       obj_at_put_without_check(which, str);
315     }
316   }
317
318   // For temporary use while constructing constant pool
319   void string_index_at_put(int which, int string_index) {
320     tag_at_put(which, JVM_CONSTANT_StringIndex);
321     *int_at_addr(which) = string_index;
322   }
323
324   void field_at_put(int which, int class_index, int name_and_type_index) {
325     tag_at_put(which, JVM_CONSTANT_Fieldref);
```



```
326     *int_at_addr(which) = ((jint) name_and_type_index<<16) | class_index;
327   }
328
329   void method_at_put(int which, int class_index, int name_and_type_index) {
330     tag_at_put(which, JVM_CONSTANT_Methodref);
331     *int_at_addr(which) = ((jint) name_and_type_index<<16) | class_index;
332   }
333
334   void interface_method_at_put(int which, int class_index, int name_and_type_ind
335     tag_at_put(which, JVM_CONSTANT_InterfaceMethodref);
336     *int_at_addr(which) = ((jint) name_and_type_index<<16) | class_index;  // No
337   }
338
339   void name_and_type_at_put(int which, int name_index, int signature_index) {
340     tag_at_put(which, JVM_CONSTANT_NameAndType);
341     *int_at_addr(which) = ((jint) signature_index<<16) | name_index;  // Not so
342   }
343
344   // Tag query
345
346   constantTag tag_at(int which) const { return (constantTag)tags()->byte_at_acqu
347
348   // Whether the entry is a pointer that must be GC'd.
349   bool is_pointer_entry(int which) {
350     constantTag tag = tag_at(which);
351     return tag.is_klass() ||
352       tag.is_string() ||
353       tag.is_object();
354   }
355
356   // Whether the entry points to an object for ldc (resolved or not)
357   bool is_object_entry(int which) {
358     constantTag tag = tag_at(which);
359     return is_pointer_entry(which) ||
360       tag.is_unresolved_klass() ||
361       tag.is_unresolved_string() ||
362       tag.is_symbol();
363   }
364
365   // Fetching constants
366
367   klassOop klass_at(int which, TRAPS) {
368     constantPoolHandle h_this(THREAD, this);
369     return klass_at_impl(h_this, which, CHECK_NULL);
370   }
371
372   Symbol* klass_name_at(int which);  // Returns the name, w/o resolving.
373
374   klassOop resolved_klass_at(int which) {  // Used by Compiler
375     guarantee(tag_at(which).is_klass(), "Corrupted constant pool");
376     // Must do an acquire here in case another thread resolved the klass
377     // behind our back, lest we later load stale values thru the oop.
378     return klassOop(CPSlot(OrderAccess::load_ptr_acquire(obj_at_addr_raw(which))
379   }
380
381   // This method should only be used with a cpool lock or during parsing or gc
382   Symbol* unresolved_klass_at(int which) {     // Temporary until actual use
383     Symbol* s = CPSlot(OrderAccess::load_ptr_acquire(obj_at_addr_raw(which))).ge
384     // check that the klass is still unresolved.
385     assert(tag_at(which).is_unresolved_klass(), "Corrupted constant pool");
386     return s;
387   }
388
389   // RedefineClasses() API support:
390   Symbol* klass_at_noresolve(int which) { return klass_name_at(which); }
```

```
392   jint int_at(int which) {
393     assert(tag_at(which).is_int(), "Corrupted constant pool");
394     return *int_at_addr(which);
395   }
396
397   jlong long_at(int which) {
398     assert(tag_at(which).is_long(), "Corrupted constant pool");
399     // return *long_at_addr(which);
400     u8 tmp = Bytes::get_native_u8((address)&base()[which]);
401     return *((jlong*)&tmp);
402   }
403
404   jfloat float_at(int which) {
405     assert(tag_at(which).is_float(), "Corrupted constant pool");
406     return *float_at_addr(which);
407   }
408
409   jdouble double_at(int which) {
410     assert(tag_at(which).is_double(), "Corrupted constant pool");
411     u8 tmp = Bytes::get_native_u8((address)&base()[which]);
412     return *((jdouble*)&tmp);
413   }
414
415   Symbol* symbol_at(int which) {
416     assert(tag_at(which).is_utf8(), "Corrupted constant pool");
417     return slot_at(which).get_symbol();
418   }
419
420   oop string_at(int which, TRAPS) {
421     constantPoolHandle h_this(THREAD, this);
422     return string_at_impl(h_this, which, CHECK_NULL);
423   }
424
425   oop object_at(int which) {
426     assert(tag_at(which).is_object(), "Corrupted constant pool");
427     return slot_at(which).get_oop();
428   }
429
430   // A "pseudo-string" is an non-string oop that has found is way into
431   // a String entry.
432   // Under EnableInvokeDynamic this can happen if the user patches a live
432   // Under AnonymousClasses this can happen if the user patches a live
433   // object into a CONSTANT_String entry of an anonymous class.
434   // Method oops internally created for method handles may also
435   // use pseudo-strings to link themselves to related metaobjects.
436
437   bool is_pseudo_string_at(int which);
438
439   oop pseudo_string_at(int which) {
440     assert(tag_at(which).is_string(), "Corrupted constant pool");
441     return slot_at(which).get_oop();
442   }
443
444   void pseudo_string_at_put(int which, oop x) {
445     assert(EnableInvokeDynamic, "");
445     assert(AnonymousClasses, "");
446     set_pseudo_string();       // mark header
447     assert(tag_at(which).is_string() || tag_at(which).is_unresolved_string(), "C
448     string_at_put(which, x);    // this works just fine
449   }
450
451   // only called when we are sure a string entry is already resolved (via an
452   // earlier string_at call.
453   oop resolved_string_at(int which) {
454     assert(tag_at(which).is_string(), "Corrupted constant pool");
455     // Must do an acquire here in case another thread resolved the klass
```



```
456     // behind our back, lest we later load stale values thru the oop.
457     return CPSlot(OrderAccess::load_ptr_acquire(obj_at_addr_raw(which))).get_oop
458   }
459
460   // This method should only be used with a cpool lock or during parsing or gc
461   Symbol* unresolved_string_at(int which) {    // Temporary until actual use
462     Symbol* s = CPSlot(OrderAccess::load_ptr_acquire(obj_at_addr_raw(which))).ge
463     // check that the string is still unresolved.
464     assert(tag_at(which).is_unresolved_string(), "Corrupted constant pool");
465     return s;
466   }
467
468   // Returns an UTF8 for a CONSTANT_String entry at a given index.
469   // UTF8 char* representation was chosen to avoid conversion of
470   // java_lang_Strings at resolved entries into Symbol*s
471   // or vice versa.
472   // Caller is responsible for checking for pseudo-strings.
473   char* string_at_noresolve(int which);
474
475   jint name_and_type_at(int which) {
476     assert(tag_at(which).is_name_and_type(), "Corrupted constant pool");
477     return *int_at_addr(which);
478   }
479
480   int method_handle_ref_kind_at(int which) {
481     assert(tag_at(which).is_method_handle(), "Corrupted constant pool");
482     return extract_low_short_from_int(*int_at_addr(which));  // mask out unwante
483   }
484   int method_handle_index_at(int which) {
485     assert(tag_at(which).is_method_handle(), "Corrupted constant pool");
486     return extract_high_short_from_int(*int_at_addr(which));  // shift out unwan
487   }
488   int method_type_index_at(int which) {
489     assert(tag_at(which).is_method_type(), "Corrupted constant pool");
490     return *int_at_addr(which);
491   }
492   // Derived queries:
493   Symbol* method_handle_name_ref_at(int which) {
494     int member = method_handle_index_at(which);
495     return impl_name_ref_at(member, true);
496   }
497   Symbol* method_handle_signature_ref_at(int which) {
498     int member = method_handle_index_at(which);
499     return impl_signature_ref_at(member, true);
500   }
501   int method_handle_klass_index_at(int which) {
502     int member = method_handle_index_at(which);
503     return impl_klass_ref_index_at(member, true);
504   }
505   Symbol* method_type_signature_at(int which) {
506     int sym = method_type_index_at(which);
507     return symbol_at(sym);
508   }
509
510   int invoke_dynamic_name_and_type_ref_index_at(int which) {
511     assert(tag_at(which).is_invoke_dynamic(), "Corrupted constant pool");
512     return extract_high_short_from_int(*int_at_addr(which));
513   }
514   int invoke_dynamic_bootstrap_specifier_index(int which) {
515     assert(tag_at(which).value() == JVM_CONSTANT_InvokeDynamic, "Corrupted const
516     return extract_low_short_from_int(*int_at_addr(which));
517   }
518   int invoke_dynamic_operand_base(int which) {
519     int bootstrap_specifier_index = invoke_dynamic_bootstrap_specifier_index(whi
520     return operand_offset_at(operands(), bootstrap_specifier_index);
521   }
```

```
522   // The first part of the operands array consists of an index into the second p
523   // Extract a 32-bit index value from the first part.
524   static int operand_offset_at(typeArrayOop operands, int bootstrap_specifier_in
525     int n = (bootstrap_specifier_index * 2);
526     assert(n >= 0 && n+2 <= operands->length(), "oob");
527     // The first 32-bit index points to the beginning of the second part
528     // of the operands array.  Make sure this index is in the first part.
529     DEBUG_ONLY(int second_part = build_int_from_shorts(operands->short_at(0),
530                                                        operands->short_at(1)));
531     assert(second_part == 0 || n+2 <= second_part, "oob (2)");
532     int offset = build_int_from_shorts(operands->short_at(n+0),
533                                        operands->short_at(n+1));
534     // The offset itself must point into the second part of the array.
535     assert(offset == 0 || offset >= second_part && offset <= operands->length(),
536     return offset;
537   }
538   static void operand_offset_at_put(typeArrayOop operands, int bootstrap_specifi
539     int n = bootstrap_specifier_index * 2;
540     assert(n >= 0 && n+2 <= operands->length(), "oob");
541     operands->short_at_put(n+0, extract_low_short_from_int(offset));
542     operands->short_at_put(n+1, extract_high_short_from_int(offset));
543   }
544   static int operand_array_length(typeArrayOop operands) {
545     if (operands == NULL || operands->length() == 0)  return 0;
546     int second_part = operand_offset_at(operands, 0);
547     return (second_part / 2);
548   }

550 #ifdef ASSERT
551   // operand tuples fit together exactly, end to end
552   static int operand_limit_at(typeArrayOop operands, int bootstrap_specifier_ind
553     int nextidx = bootstrap_specifier_index + 1;
554     if (nextidx == operand_array_length(operands))
555       return operands->length();
556     else
557       return operand_offset_at(operands, nextidx);
558   }
559   int invoke_dynamic_operand_limit(int which) {
560     int bootstrap_specifier_index = invoke_dynamic_bootstrap_specifier_index(whi
561     return operand_limit_at(operands(), bootstrap_specifier_index);
562   }
563 #endif //ASSERT

565   // layout of InvokeDynamic bootstrap method specifier (in second part of opera
566   enum {
567         _indy_bsm_offset  = 0,  // CONSTANT_MethodHandle bsm
568         _indy_argc_offset = 1,  // u2 argc
569         _indy_argv_offset = 2   // u2 argv[argc]
570   };
571   int invoke_dynamic_bootstrap_method_ref_index_at(int which) {
572     assert(tag_at(which).is_invoke_dynamic(), "Corrupted constant pool");
573     if (tag_at(which).value() == JVM_CONSTANT_InvokeDynamicTrans)
574       return extract_low_short_from_int(*int_at_addr(which));
575     int op_base = invoke_dynamic_operand_base(which);
576     return operands()->short_at(op_base + _indy_bsm_offset);
577   }
578   int invoke_dynamic_argument_count_at(int which) {
579     assert(tag_at(which).is_invoke_dynamic(), "Corrupted constant pool");
580     if (tag_at(which).value() == JVM_CONSTANT_InvokeDynamicTrans)
581       return 0;
582     int op_base = invoke_dynamic_operand_base(which);
583     int argc = operands()->short_at(op_base + _indy_argc_offset);
584     DEBUG_ONLY(int end_offset = op_base + _indy_argv_offset + argc;
585                int next_offset = invoke_dynamic_operand_limit(which));
586     assert(end_offset == next_offset, "matched ending");
587     return argc;
```



```
588   }
589   int invoke_dynamic_argument_index_at(int which, int j) {
590     int op_base = invoke_dynamic_operand_base(which);
591     DEBUG_ONLY(int argc = operands()->short_at(op_base + _indy_argc_offset));
592     assert((uint)j < (uint)argc, "oob");
593     return operands()->short_at(op_base + _indy_argv_offset + j);
594   }

596   // The following methods (name/signature/klass_ref_at, klass_ref_at_noresolve,
597   // name_and_type_ref_index_at) all expect to be passed indices obtained
598   // directly from the bytecode.
599   // If the indices are meant to refer to fields or methods, they are
600   // actually rewritten constant pool cache indices.
601   // The routine remap_instruction_operand_from_cache manages the adjustment
602   // of these values back to constant pool indices.

604   // There are also "uncached" versions which do not adjust the operand index; s

606   // FIXME: Consider renaming these with a prefix "cached_" to make the distinct
607   // In a few cases (the verifier) there are uses before a cpcache has been buil
608   // which are handled by a dynamic check in remap_instruction_operand_from_cach
609   // FIXME: Remove the dynamic check, and adjust all callers to specify the corr

611   // Lookup for entries consisting of (klass_index, name_and_type index)
612   klassOop klass_ref_at(int which, TRAPS);
613   Symbol* klass_ref_at_noresolve(int which);
614   Symbol* name_ref_at(int which)                { return impl_name_ref_at(which,
615   Symbol* signature_ref_at(int which)           { return impl_signature_ref_at(w

617   int klass_ref_index_at(int which)               { return impl_klass_ref_index_
618   int name_and_type_ref_index_at(int which)       { return impl_name_and_type_re

620   // Lookup for entries consisting of (name_index, signature_index)
621   int name_ref_index_at(int which_nt);            // ==  low-order jshort of nam
622   int signature_ref_index_at(int which_nt);       // == high-order jshort of nam

624   BasicType basic_type_for_signature_at(int which);

626   // Resolve string constants (to prevent allocation during compilation)
627   void resolve_string_constants(TRAPS) {
628     constantPoolHandle h_this(THREAD, this);
629     resolve_string_constants_impl(h_this, CHECK);
630   }

632 private:
633   enum { _no_index_sentinel = -1, _possible_index_sentinel = -2 };
634 public:

636   // Resolve late bound constants.
637   oop resolve_constant_at(int index, TRAPS) {
638     constantPoolHandle h_this(THREAD, this);
639     return resolve_constant_at_impl(h_this, index, _no_index_sentinel, THREAD);
640   }

642   oop resolve_cached_constant_at(int cache_index, TRAPS) {
643     constantPoolHandle h_this(THREAD, this);
644     return resolve_constant_at_impl(h_this, _no_index_sentinel, cache_index, THR
645   }

647   oop resolve_possibly_cached_constant_at(int pool_index, TRAPS) {
648     constantPoolHandle h_this(THREAD, this);
649     return resolve_constant_at_impl(h_this, pool_index, _possible_index_sentinel
650   }

652   // Klass name matches name at offset
653   bool klass_name_at_matches(instanceKlassHandle k, int which);
```

```
655   // Sizing
656   int length() const                  { return _length; }
657   void set_length(int length)         { _length = length; }

659   // Tells whether index is within bounds.
660   bool is_within_bounds(int index) const {
661     return 0 <= index && index < length();
662   }

664   static int header_size()            { return sizeof(constantPoolOopDesc)/Heap
665   static int object_size(int length)  { return align_object_size(header_size()
666   int object_size()                   { return object_size(length()); }

668   bool is_conc_safe()                 { return _is_conc_safe; }
669   void set_is_conc_safe(bool v)       { _is_conc_safe = v; }

671   friend class constantPoolKlass;
672   friend class ClassFileParser;
673   friend class SystemDictionary;

675   // Used by compiler to prevent classloading.
676   static klassOop klass_at_if_loaded          (constantPoolHandle this_oop, int
677   static klassOop klass_ref_at_if_loaded      (constantPoolHandle this_oop, int
678   // Same as above - but does LinkResolving.
679   static klassOop klass_ref_at_if_loaded_check(constantPoolHandle this_oop, int

681   // Routines currently used for annotations (only called by jvm.cpp) but which
682   // future by other Java code. These take constant pool indices rather than
683   // constant pool cache indices as do the peer methods above.
684   Symbol* uncached_klass_ref_at_noresolve(int which);
685   Symbol* uncached_name_ref_at(int which)               { return impl_name_ref
686   Symbol* uncached_signature_ref_at(int which)          { return impl_signatur
687   int       uncached_klass_ref_index_at(int which)        { return impl_klass_
688   int       uncached_name_and_type_ref_index_at(int which)  { return impl_name_a

690   // Sharing
691   int pre_resolve_shared_klasses(TRAPS);
692   void shared_symbols_iterate(SymbolClosure* closure0);
693   void shared_tags_iterate(OopClosure* closure0);
694   void shared_strings_iterate(OopClosure* closure0);

696   // Debugging
697   const char* printable_name_at(int which) PRODUCT_RETURN0;

699 #ifdef ASSERT
700   enum { CPCACHE_INDEX_TAG = 0x10000 };  // helps keep CP cache indices distinct
701 #else
702   enum { CPCACHE_INDEX_TAG = 0 };        // in product mode, this zero value is
703 #endif //ASSERT

705  private:

707   Symbol* impl_name_ref_at(int which, bool uncached);
708   Symbol* impl_signature_ref_at(int which, bool uncached);
709   int       impl_klass_ref_index_at(int which, bool uncached);
710   int       impl_name_and_type_ref_index_at(int which, bool uncached);

712   int remap_instruction_operand_from_cache(int operand);  // operand must be bia

714   // Used while constructing constant pool (only by ClassFileParser)
715   jint klass_index_at(int which) {
716     assert(tag_at(which).is_klass_index(), "Corrupted constant pool");
717     return *int_at_addr(which);
718   }
```



```
720   jint string_index_at(int which) {
721     assert(tag_at(which).is_string_index(), "Corrupted constant pool");
722     return *int_at_addr(which);
723   }

725   // Performs the LinkResolver checks
726   static void verify_constant_pool_resolve(constantPoolHandle this_oop, KlassHan

728   // Implementation of methods that needs an exposed 'this' pointer, in order to
729   // handle GC while executing the method
730   static klassOop klass_at_impl(constantPoolHandle this_oop, int which, TRAPS);
731   static oop string_at_impl(constantPoolHandle this_oop, int which, TRAPS);

733   // Resolve string constants (to prevent allocation during compilation)
734   static void resolve_string_constants_impl(constantPoolHandle this_oop, TRAPS);

736   static oop resolve_constant_at_impl(constantPoolHandle this_oop, int index, in

738  public:
739   // Merging constantPoolOop support:
740   bool compare_entry_to(int index1, constantPoolHandle cp2, int index2, TRAPS);
741   void copy_cp_to(int start_i, int end_i, constantPoolHandle to_cp, int to_i, TR
742     constantPoolHandle h_this(THREAD, this);
743     copy_cp_to_impl(h_this, start_i, end_i, to_cp, to_i, THREAD);
744   }
745   static void copy_cp_to_impl(constantPoolHandle from_cp, int start_i, int end_i
746   static void copy_entry_to(constantPoolHandle from_cp, int from_i, constantPool
747   int  find_matching_entry(int pattern_i, constantPoolHandle search_cp, TRAPS);
748   int  orig_length() const                { return _orig_length; }
749   void set_orig_length(int orig_length)   { _orig_length = orig_length; }

751   // Decrease ref counts of symbols that are in the constant pool
752   // when the holder class is unloaded
753   void unreference_symbols();

755   // JVMTI accesss - GetConstantPool, RetransformClasses, ...
756   friend class JvmtiConstantPoolReconstituter;

758  private:
759   jint cpool_entry_size(jint idx);
760   jint hash_entries_to(SymbolHashMap *symmap, SymbolHashMap *classmap);

762   // Copy cpool bytes into byte array.
763   // Returns:
764   //  int > 0, count of the raw cpool bytes that have been copied
765   //        0, OutOfMemory error
766   //       -1, Internal error
767   int  copy_cpool_bytes(int cpool_size,
768                         SymbolHashMap* tbl,
769                         unsigned char *bytes);
770 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   46839 Wed Mar 30 07:00:22 2011
new/src/share/vm/oops/instanceKlass.hpp
**********************************************************
______unchanged_portion_omitted_

 136 class instanceKlass: public Klass {
 137   friend class VMStructs;
 138  public:
 139   // See "The Java Virtual Machine Specification" section 2.16.2-5 for a detaile
 140   // of the class loading & initialization procedure, and the use of the states.
 141   enum ClassState {
 142     unparsable_by_gc = 0,               // object is not yet parsable by gc. Val
 143     allocated,                          // allocated (but not yet linked)
 144     loaded,                             // loaded and inserted in class hierarch
 145     linked,                             // successfully linked/verified (but not
 146     being_initialized,                  // currently running class initializer
 147     fully_initialized,                  // initialized (successfull final state)
 148     initialization_error                // error happened during initialization
 149   };

 151  public:
 152   oop* oop_block_beg() const { return adr_array_klasses(); }
 153   oop* oop_block_end() const { return adr_methods_default_annotations() + 1; }

 155   enum {
 156     implementors_limit = 2              // how many implems can we track?
 157   };

 159  protected:
 160   //
 161   // The oop block.  See comment in klass.hpp before making changes.
 162   //

 164   // Array classes holding elements of this class.
 165   klassOop        _array_klasses;
 166   // Method array.
 167   objArrayOop     _methods;
 168   // Int array containing the original order of method in the class file (for
 169   // JVMTI).
 170   typeArrayOop    _method_ordering;
 171   // Interface (klassOops) this class declares locally to implement.
 172   objArrayOop     _local_interfaces;
 173   // Interface (klassOops) this class implements transitively.
 174   objArrayOop     _transitive_interfaces;
 175   // Instance and static variable information, 5-tuples of shorts [access, name
 176   // index, sig index, initval index, offset].
 177   typeArrayOop    _fields;
 178   // Constant pool for this class.
 179   constantPoolOop _constants;
 180   // Class loader used to load this class, NULL if VM loader used.
 181   oop             _class_loader;
 182   // Protection domain.
 183   oop             _protection_domain;
 184   // Host class, which grants its access privileges to this class also.
 185   // This is only non-null for an anonymous class (JSR 292 enabled).
 185   // This is only non-null for an anonymous class (AnonymousClasses enabled).
 186   // The host class is either named, or a previously loaded anonymous class.
 187   klassOop        _host_klass;
 188   // Class signers.
 189   objArrayOop     _signers;
 190   // inner_classes attribute.
 191   typeArrayOop    _inner_classes;
 192   // Implementors of this interface (not valid if it overflows)
 193   klassOop        _implementors[implementors_limit];
 194   // invokedynamic bootstrap method (a java.lang.invoke.MethodHandle)
```



```
 195   oop             _bootstrap_method;  // AllowTransitionalJSR292 ONLY
 196   // Annotations for this class, or null if none.
 197   typeArrayOop    _class_annotations;
 198   // Annotation objects (byte arrays) for fields, or null if no annotations.
 199   // Indices correspond to entries (not indices) in fields array.
 200   objArrayOop     _fields_annotations;
 201   // Annotation objects (byte arrays) for methods, or null if no annotations.
 202   // Index is the idnum, which is initially the same as the methods array index.
 203   objArrayOop     _methods_annotations;
 204   // Annotation objects (byte arrays) for methods' parameters, or null if no
 205   // such annotations.
 206   // Index is the idnum, which is initially the same as the methods array index.
 207   objArrayOop     _methods_parameter_annotations;
 208   // Annotation objects (byte arrays) for methods' default values, or null if no
 209   // such annotations.
 210   // Index is the idnum, which is initially the same as the methods array index.
 211   objArrayOop     _methods_default_annotations;

 213   //
 214   // End of the oop block.
 215   //

 217   // Name of source file containing this klass, NULL if not specified.
 218   Symbol*         _source_file_name;
 219   // the source debug extension for this klass, NULL if not specified.
 220   Symbol*         _source_debug_extension;
 221   // Generic signature, or null if none.
 222   Symbol*         _generic_signature;
 223   // Array name derived from this class which needs unreferencing
 224   // if this class is unloaded.
 225   Symbol*         _array_name;

 227   // Number of heapOopSize words used by non-static fields in this klass
 228   // (including inherited fields but after header_size()).
 229   int             _nonstatic_field_size;
 230   int             _static_field_size;    // number words used by static fields (
 231   int             _static_oop_field_count;// number of static oop fields in this
 232   int             _nonstatic_oop_map_size;// size in words of nonstatic oop map
 233   bool            _is_marked_dependent;  // used for marking during flushing and
 234   bool            _rewritten;            // methods rewritten.
 235   bool            _has_nonstatic_fields; // for sizing with UseCompressedOops
 236   bool            _should_verify_class;  // allow caching of preverification
 237   u2              _minor_version;        // minor version number of class file
 238   u2              _major_version;        // major version number of class file
 239   ClassState      _init_state;           // state of class
 240   Thread*         _init_thread;          // Pointer to current thread doing init
 241   int             _vtable_len;           // length of Java vtable (in words)
 242   int             _itable_len;           // length of Java itable (in words)
 243   ReferenceType   _reference_type;       // reference type
 244   OopMapCache*    volatile _oop_map_cache;   // OopMapCache for all methods in t
 245   JNIid*          _jni_ids;              // First JNI identifier for static fiel
 246   jmethodID*      _methods_jmethod_ids;  // jmethodIDs corresponding to method_i
 247   int*            _methods_cached_itable_indices;  // itable_index cache for JNI
 248   nmethodBucket*  _dependencies;         // list of dependent nmethods
 249   nmethod*        _osr_nmethods_head;    // Head of list of on-stack replacement
 250   BreakpointInfo* _breakpoints;          // bpt lists, managed by methodOop
 251   int             _nof_implementors;     // No of implementors of this interface
 252   // Array of interesting part(s) of the previous version(s) of this
 253   // instanceKlass. See PreviousVersionWalker below.
 254   GrowableArray<PreviousVersionNode *>* _previous_versions;
 255   u2              _enclosing_method_class_index;  // Constant pool index for cla
 256   u2              _enclosing_method_method_index; // Constant pool index for nam
 257   // JVMTI fields can be moved to their own structure - see 6315920
 258   unsigned char * _cached_class_file_bytes;       // JVMTI: cached class file, b
 259   jint            _cached_class_file_len;         // JVMTI: length of above
 260   JvmtiCachedClassFieldMap* _jvmti_cached_class_field_map;  // JVMTI: used durin
```

```
261   volatile u2     _idnum_allocated_count;         // JNI/JVMTI: increments with

263   // embedded Java vtable follows here
264   // embedded Java itables follows here
265   // embedded static fields follows here
266   // embedded nonstatic oop-map blocks follows here

268   friend class instanceKlassKlass;
269   friend class SystemDictionary;

271  public:
272   bool has_nonstatic_fields() const        { return _has_nonstatic_fields; }
273   void set_has_nonstatic_fields(bool b)    { _has_nonstatic_fields = b; }

275   // field sizes
276   int nonstatic_field_size() const         { return _nonstatic_field_size; }
277   void set_nonstatic_field_size(int size)  { _nonstatic_field_size = size; }

279   int static_field_size() const            { return _static_field_size; }
280   void set_static_field_size(int size)     { _static_field_size = size; }

282   int static_oop_field_count() const        { return _static_oop_field_count; }
283   void set_static_oop_field_count(int size) { _static_oop_field_count = size; }

285   // Java vtable
286   int  vtable_length() const               { return _vtable_len; }
287   void set_vtable_length(int len)          { _vtable_len = len; }

289   // Java itable
290   int  itable_length() const               { return _itable_len; }
291   void set_itable_length(int len)          { _itable_len = len; }

293   // array klasses
294   klassOop array_klasses() const           { return _array_klasses; }
295   void set_array_klasses(klassOop k)       { oop_store_without_check((oop*) &_ar

297   // methods
298   objArrayOop methods() const              { return _methods; }
299   void set_methods(objArrayOop a)          { oop_store_without_check((oop*) &_me
300   methodOop method_with_idnum(int idnum);

302   // method ordering
303   typeArrayOop method_ordering() const     { return _method_ordering; }
304   void set_method_ordering(typeArrayOop m) { oop_store_without_check((oop*) &_me

306   // interfaces
307   objArrayOop local_interfaces() const          { return _local_interfaces; }
308   void set_local_interfaces(objArrayOop a)      { oop_store_without_check((oop*)
309   objArrayOop transitive_interfaces() const     { return _transitive_interfaces;
310   void set_transitive_interfaces(objArrayOop a) { oop_store_without_check((oop*)

312   // fields
313   // Field info extracted from the class file and stored
314   // as an array of 7 shorts
315   enum FieldOffset {
316     access_flags_offset    = 0,
317     name_index_offset      = 1,
318     signature_index_offset = 2,
319     initval_index_offset   = 3,
320     low_offset             = 4,
321     high_offset            = 5,
322     generic_signature_offset = 6,
323     next_offset            = 7
324   };

326   typeArrayOop fields() const              { return _fields; }
```



```
327   int offset_from_fields( int index ) const {
328     return build_int_from_shorts( fields()->ushort_at(index + low_offset),
329                                   fields()->ushort_at(index + high_offset) );
330   }

332   void set_fields(typeArrayOop f)          { oop_store_without_check((oop*) &_fi

334   // inner classes
335   typeArrayOop inner_classes() const       { return _inner_classes; }
336   void set_inner_classes(typeArrayOop f)   { oop_store_without_check((oop*) &_in

338   enum InnerClassAttributeOffset {
339     // From http://mirror.eng/products/jdk/1.1/docs/guide/innerclasses/spec/inne
340     inner_class_inner_class_info_offset = 0,
341     inner_class_outer_class_info_offset = 1,
342     inner_class_inner_name_offset = 2,
343     inner_class_access_flags_offset = 3,
344     inner_class_next_offset = 4
345   };

347   // method override check
348   bool is_override(methodHandle super_method, Handle targetclassloader, Symbol*

350   // package
351   bool is_same_class_package(klassOop class2);
352   bool is_same_class_package(oop classloader2, Symbol* classname2);
353   static bool is_same_class_package(oop class_loader1, Symbol* class_name1, oop

355   // find an enclosing class (defined where original code was, in jvm.cpp!)
356   klassOop compute_enclosing_class(bool* inner_is_member, TRAPS) {
357     instanceKlassHandle self(THREAD, this->as_klassOop());
358     return compute_enclosing_class_impl(self, inner_is_member, THREAD);
359   }
360   static klassOop compute_enclosing_class_impl(instanceKlassHandle self,
361                                                bool* inner_is_member, TRAPS);

363   // tell if two classes have the same enclosing class (at package level)
364   bool is_same_package_member(klassOop class2, TRAPS) {
365     instanceKlassHandle self(THREAD, this->as_klassOop());
366     return is_same_package_member_impl(self, class2, THREAD);
367   }
368   static bool is_same_package_member_impl(instanceKlassHandle self,
369                                           klassOop class2, TRAPS);

371   // initialization state
372   bool is_loaded() const                   { return _init_state >= loaded; }
373   bool is_linked() const                   { return _init_state >= linked; }
374   bool is_initialized() const              { return _init_state == fully_initial
375   bool is_not_initialized() const          { return _init_state <  being_initial
376   bool is_being_initialized() const        { return _init_state == being_initial
377   bool is_in_error_state() const           { return _init_state == initializatio
378   bool is_reentrant_initialization(Thread *thread)  { return thread == _init_thr
379   int  get_init_state()                    { return _init_state; } // Useful for
380   bool is_rewritten() const                { return _rewritten; }

382   // defineClass specified verification
383   bool should_verify_class() const         { return _should_verify_class; }
384   void set_should_verify_class(bool value) { _should_verify_class = value; }

386   // marking
387   bool is_marked_dependent() const         { return _is_marked_dependent; }
388   void set_is_marked_dependent(bool value) { _is_marked_dependent = value; }

390   // initialization (virtuals from Klass)
391   bool should_be_initialized() const;  // means that initialize should be called
392   void initialize(TRAPS);
```

```
393   void link_class(TRAPS);
394   bool link_class_or_fail(TRAPS); // returns false on failure
395   void unlink_class();
396   void rewrite_class(TRAPS);
397   methodOop class_initializer();

399   // set the class to initialized if no static initializer is present
400   void eager_initialize(Thread *thread);

402   // reference type
403   ReferenceType reference_type() const     { return _reference_type; }
404   void set_reference_type(ReferenceType t) { _reference_type = t; }

406   // find local field, returns true if found
407   bool find_local_field(Symbol* name, Symbol* sig, fieldDescriptor* fd) const;
408   // find field in direct superinterfaces, returns the interface in which the fi
409   klassOop find_interface_field(Symbol* name, Symbol* sig, fieldDescriptor* fd)
410   // find field according to JVM spec 5.4.3.2, returns the klass in which the fi
411   klassOop find_field(Symbol* name, Symbol* sig, fieldDescriptor* fd) const;
412   // find instance or static fields according to JVM spec 5.4.3.2, returns the k
413   klassOop find_field(Symbol* name, Symbol* sig, bool is_static, fieldDescriptor

415   // find a non-static or static field given its offset within the class.
416   bool contains_field_offset(int offset) {
417     return instanceOopDesc::contains_field_offset(offset, nonstatic_field_size()
418   }

420   bool find_local_field_from_offset(int offset, bool is_static, fieldDescriptor*
421   bool find_field_from_offset(int offset, bool is_static, fieldDescriptor* fd) c

423   // find a local method (returns NULL if not found)
424   methodOop find_method(Symbol* name, Symbol* signature) const;
425   static methodOop find_method(objArrayOop methods, Symbol* name, Symbol* signat

427   // lookup operation (returns NULL if not found)
428   methodOop uncached_lookup_method(Symbol* name, Symbol* signature) const;

430   // lookup a method in all the interfaces that this class implements
431   // (returns NULL if not found)
432   methodOop lookup_method_in_all_interfaces(Symbol* name, Symbol* signature) con

434   // constant pool
435   constantPoolOop constants() const        { return _constants; }
436   void set_constants(constantPoolOop c)    { oop_store_without_check((oop*) &_co

438   // class loader
439   oop class_loader() const                 { return _class_loader; }
440   void set_class_loader(oop l)             { oop_store((oop*) &_class_loader, l)

442   // protection domain
443   oop protection_domain()                  { return _protection_domain; }
444   void set_protection_domain(oop pd)       { oop_store((oop*) &_protection_domai

446   // host class
447   oop host_klass() const                   { return _host_klass; }
448   void set_host_klass(oop host)            { oop_store((oop*) &_host_klass, host
449   bool is_anonymous() const                { return _host_klass != NULL; }

451   // signers
452   objArrayOop signers() const              { return _signers; }
453   void set_signers(objArrayOop s)          { oop_store((oop*) &_signers, oop(s))

455   // source file name
456   Symbol* source_file_name() const         { return _source_file_name; }
457   void set_source_file_name(Symbol* n);
```



```
459   // minor and major version numbers of class file
460   u2 minor_version() const                 { return _minor_version; }
461   void set_minor_version(u2 minor_version) { _minor_version = minor_version; }
462   u2 major_version() const                 { return _major_version; }
463   void set_major_version(u2 major_version) { _major_version = major_version; }

465   // source debug extension
466   Symbol* source_debug_extension() const   { return _source_debug_extension; }
467   void set_source_debug_extension(Symbol* n);

469   // symbol unloading support (refcount already added)
470   Symbol* array_name()                     { return _array_name; }
471   void set_array_name(Symbol* name)        { assert(_array_name == NULL, "name a

473   // nonstatic oop-map blocks
474   static int nonstatic_oop_map_size(unsigned int oop_map_count) {
475     return oop_map_count * OopMapBlock::size_in_words();
476   }
477   unsigned int nonstatic_oop_map_count() const {
478     return _nonstatic_oop_map_size / OopMapBlock::size_in_words();
479   }
480   int nonstatic_oop_map_size() const { return _nonstatic_oop_map_size; }
481   void set_nonstatic_oop_map_size(int words) {
482     _nonstatic_oop_map_size = words;
483   }

485   // RedefineClasses() support for previous versions:
486   void add_previous_version(instanceKlassHandle ikh, BitMap *emcp_methods,
487         int emcp_method_count);
488   // If the _previous_versions array is non-NULL, then this klass
489   // has been redefined at least once even if we aren't currently
490   // tracking a previous version.
491   bool has_been_redefined() const { return _previous_versions != NULL; }
492   bool has_previous_version() const;
493   void init_previous_versions() {
494     _previous_versions = NULL;
495   }
496   GrowableArray<PreviousVersionNode *>* previous_versions() const {
497     return _previous_versions;
498   }

500   // JVMTI: Support for caching a class file before it is modified by an agent t
501   void set_cached_class_file(unsigned char *class_file_bytes,
502                              jint class_file_len)     { _cached_class_file_len =
503                                                         _cached_class_file_bytes
504   jint get_cached_class_file_len()                    { return _cached_class_fil
505   unsigned char * get_cached_class_file_bytes()       { return _cached_class_fil

507   // JVMTI: Support for caching of field indices, types, and offsets
508   void set_jvmti_cached_class_field_map(JvmtiCachedClassFieldMap* descriptor) {
509     _jvmti_cached_class_field_map = descriptor;
510   }
511   JvmtiCachedClassFieldMap* jvmti_cached_class_field_map() const {
512     return _jvmti_cached_class_field_map;
513   }

515   // for adding methods, constMethodOopDesc::UNSET_IDNUM means no more ids avail
516   inline u2 next_method_idnum();
517   void set_initial_method_idnum(u2 value)             { _idnum_allocated_count =

519   // generics support
520   Symbol* generic_signature() const                   { return _generic_signatur
521   void set_generic_signature(Symbol* sig)             { _generic_signature = sig

523   u2 enclosing_method_class_index() const             { return _enclosing_method
524   u2 enclosing_method_method_index() const            { return _enclosing_method
```

```
525   void set_enclosing_method_indices(u2 class_index,
526                                     u2 method_index)  { _enclosing_method_class_
527                                                         _enclosing_method_method

529   // JSR 292 support
530   oop bootstrap_method() const                        { return _bootstrap_method
531   void set_bootstrap_method(oop mh)                   { oop_store(&_bootstrap_me

533   // jmethodID support
534   static jmethodID get_jmethod_id(instanceKlassHandle ik_h,
535                                   methodHandle method_h);
536   static jmethodID get_jmethod_id_fetch_or_update(instanceKlassHandle ik_h,
537                                   size_t idnum, jmethodID new_id, jmethodID* new_jmeths,
538                                   jmethodID* to_dealloc_id_p,
539                                   jmethodID** to_dealloc_jmeths_p);
540   static void get_jmethod_id_length_value(jmethodID* cache, size_t idnum,
541                                size_t *length_p, jmethodID* id_p);
542   jmethodID jmethod_id_or_null(methodOop method);

544   // cached itable index support
545   void set_cached_itable_index(size_t idnum, int index);
546   int cached_itable_index(size_t idnum);

548   // annotations support
549   typeArrayOop class_annotations() const              { return _class_annotation
550   objArrayOop fields_annotations() const              { return _fields_annotatio
551   objArrayOop methods_annotations() const             { return _methods_annotati
552   objArrayOop methods_parameter_annotations() const   { return _methods_paramete
553   objArrayOop methods_default_annotations() const     { return _methods_default_
554   void set_class_annotations(typeArrayOop md)           { oop_store_without_che
555   void set_fields_annotations(objArrayOop md)           { set_annotations(md, &
556   void set_methods_annotations(objArrayOop md)          { set_annotations(md, &
557   void set_methods_parameter_annotations(objArrayOop md) { set_annotations(md, &
558   void set_methods_default_annotations(objArrayOop md)   { set_annotations(md, &
559   typeArrayOop get_method_annotations_of(int idnum)
560                                                       { return get_method_annotations_
561   typeArrayOop get_method_parameter_annotations_of(int idnum)
562                                                       { return get_method_annotations_
563   typeArrayOop get_method_default_annotations_of(int idnum)
564                                                       { return get_method_annotations_
565   void set_method_annotations_of(int idnum, typeArrayOop anno)
566                                                       { set_methods_annotations_of(idn
567   void set_method_parameter_annotations_of(int idnum, typeArrayOop anno)
568                                                       { set_methods_annotations_of(idn
569   void set_method_default_annotations_of(int idnum, typeArrayOop anno)
570                                                       { set_methods_annotations_of(idn

572   // allocation
573   DEFINE_ALLOCATE_PERMANENT(instanceKlass);
574   instanceOop allocate_instance(TRAPS);
575   instanceOop allocate_permanent_instance(TRAPS);

577   // additional member function to return a handle
578   instanceHandle allocate_instance_handle(TRAPS)      { return instanceHandle(TH

580   objArrayOop allocate_objArray(int n, int length, TRAPS);
581   // Helper function
582   static instanceOop register_finalizer(instanceOop i, TRAPS);

584   // Check whether reflection/jni/jvm code is allowed to instantiate this class;
585   // if not, throw either an Error or an Exception.
586   virtual void check_valid_for_instantiation(bool throwError, TRAPS);

588   // initialization
589   void call_class_initializer(TRAPS);
590   void set_initialization_state_and_notify(ClassState state, TRAPS);
```



```
592   // OopMapCache support
593   OopMapCache* oop_map_cache()               { return _oop_map_cache; }
594   void set_oop_map_cache(OopMapCache *cache) { _oop_map_cache = cache; }
595   void mask_for(methodHandle method, int bci, InterpreterOopMap* entry);

597   // JNI identifier support (for static fields - for jni performance)
598   JNIid* jni_ids()                                { return _jni_ids; }
599   void set_jni_ids(JNIid* ids)                    { _jni_ids = ids; }
600   JNIid* jni_id_for(int offset);

602   // maintenance of deoptimization dependencies
603   int mark_dependent_nmethods(DepChange& changes);
604   void add_dependent_nmethod(nmethod* nm);
605   void remove_dependent_nmethod(nmethod* nm);

607   // On-stack replacement support
608   nmethod* osr_nmethods_head() const         { return _osr_nmethods_head; };
609   void set_osr_nmethods_head(nmethod* h)     { _osr_nmethods_head = h; };
610   void add_osr_nmethod(nmethod* n);
611   void remove_osr_nmethod(nmethod* n);
612   nmethod* lookup_osr_nmethod(const methodOop m, int bci, int level, bool match_

614   // Breakpoint support (see methods on methodOop for details)
615   BreakpointInfo* breakpoints() const       { return _breakpoints; };
616   void set_breakpoints(BreakpointInfo* bps) { _breakpoints = bps; };

618   // support for stub routines
619   static int init_state_offset_in_bytes()    { return offset_of(instanceKlass, _
620   static int init_thread_offset_in_bytes()   { return offset_of(instanceKlass, _

622   // subclass/subinterface checks
623   bool implements_interface(klassOop k) const;

625   // Access to implementors of an interface. We only store the count
626   // of implementors, and in case, there are only a few
627   // implementors, we store them in a short list.
628   // This accessor returns NULL if we walk off the end of the list.
629   klassOop implementor(int i) const {
630     return (i < implementors_limit)? _implementors[i]: (klassOop) NULL;
631   }
632   int  nof_implementors() const       { return _nof_implementors; }
633   void add_implementor(klassOop k);  // k is a new class that implements this in
634   void init_implementor();           // initialize

636   // link this class into the implementors list of every interface it implements
637   void process_interfaces(Thread *thread);

639   // virtual operations from Klass
640   bool is_leaf_class() const              { return _subklass == NULL; }
641   objArrayOop compute_secondary_supers(int num_extra_slots, TRAPS);
642   bool compute_is_subtype_of(klassOop k);
643   bool can_be_primary_super_slow() const;
644   klassOop java_super() const             { return super(); }
645   int oop_size(oop obj)  const            { return size_helper(); }
646   int klass_oop_size() const              { return object_size(); }
647   bool oop_is_instance_slow() const       { return true; }

649   // Iterators
650   void do_local_static_fields(FieldClosure* cl);
651   void do_nonstatic_fields(FieldClosure* cl); // including inherited fields
652   void do_local_static_fields(void f(fieldDescriptor*, TRAPS), TRAPS);

654   void methods_do(void f(methodOop method));
655   void array_klasses_do(void f(klassOop k));
656   void with_array_klasses_do(void f(klassOop k));
```

```
657   bool super_types_do(SuperTypeClosure* blk);

659   // Casting from klassOop
660   static instanceKlass* cast(klassOop k) {
661     assert(k->is_klass(), "must be");
662     Klass* kp = k->klass_part();
663     assert(kp->null_vtbl() || kp->oop_is_instance_slow(), "cast to instanceKlass
664     return (instanceKlass*) kp;
665   }

667   // Sizing (in words)
668   static int header_size()            { return align_object_offset(oopDesc::head
669   int object_size() const             { return object_size(align_object_offset(v
670   static int vtable_start_offset()    { return header_size(); }
671   static int vtable_length_offset()   { return oopDesc::header_size() + offset_o
672   static int object_size(int extra)   { return align_object_size(header_size() +

674   intptr_t* start_of_vtable() const        { return ((intptr_t*)as_klassOop()) +
675   intptr_t* start_of_itable() const        { return start_of_vtable() + align_ob
676   int  itable_offset_in_words() const { return start_of_itable() - (intptr_t*)as

678   intptr_t* end_of_itable() const          { return start_of_itable() + itable_l

680   address static_field_addr(int offset);

682   OopMapBlock* start_of_nonstatic_oop_maps() const {
683     return (OopMapBlock*)(start_of_itable() + align_object_offset(itable_length(
684   }

686   // Allocation profiling support
687   juint alloc_size() const            { return _alloc_count * size_helper(); }
688   void set_alloc_size(juint n)        {}

690   // Use this to return the size of an instance in heap words:
691   int size_helper() const {
692     return layout_helper_to_size_helper(layout_helper());
693   }

695   // This bit is initialized in classFileParser.cpp.
696   // It is false under any of the following conditions:
697   //  - the class is abstract (including any interface)
698   //  - the class has a finalizer (if !RegisterFinalizersAtInit)
699   //  - the class size is larger than FastAllocateSizeLimit
700   //  - the class is java/lang/Class, which cannot be allocated directly
701   bool can_be_fastpath_allocated() const {
702     return !layout_helper_needs_slow_path(layout_helper());
703   }

705   // Java vtable/itable
706   klassVtable* vtable() const;        // return new klassVtable wrapper
707   inline methodOop method_at_vtable(int index);
708   klassItable* itable() const;        // return new klassItable wrapper
709   methodOop method_at_itable(klassOop holder, int index, TRAPS);

711   // Garbage collection
712   void oop_follow_contents(oop obj);
713   int  oop_adjust_pointers(oop obj);
714   bool object_is_parsable() const { return _init_state != unparsable_by_gc; }
715        // Value of _init_state must be zero (unparsable_by_gc) when klass field

717   void follow_weak_klass_links(
718     BoolObjectClosure* is_alive, OopClosure* keep_alive);
719   void release_C_heap_structures();

721   // Parallel Scavenge and Parallel Old
722   PARALLEL_GC_DECLS
```



```
724   // Naming
725   const char* signature_name() const;

727   // Iterators
728   int oop_oop_iterate(oop obj, OopClosure* blk) {
729     return oop_oop_iterate_v(obj, blk);
730   }

732   int oop_oop_iterate_m(oop obj, OopClosure* blk, MemRegion mr) {
733     return oop_oop_iterate_v_m(obj, blk, mr);
734   }

736 #define InstanceKlass_OOP_OOP_ITERATE_DECL(OopClosureType, nv_suffix)      \
737   int  oop_oop_iterate##nv_suffix(oop obj, OopClosureType* blk);           \
738   int  oop_oop_iterate##nv_suffix##_m(oop obj, OopClosureType* blk,        \
739                                       MemRegion mr);

741   ALL_OOP_OOP_ITERATE_CLOSURES_1(InstanceKlass_OOP_OOP_ITERATE_DECL)
742   ALL_OOP_OOP_ITERATE_CLOSURES_2(InstanceKlass_OOP_OOP_ITERATE_DECL)

744 #ifndef SERIALGC
745 #define InstanceKlass_OOP_OOP_ITERATE_BACKWARDS_DECL(OopClosureType, nv_suffix)
746   int  oop_oop_iterate_backwards##nv_suffix(oop obj, OopClosureType* blk);

748   ALL_OOP_OOP_ITERATE_CLOSURES_1(InstanceKlass_OOP_OOP_ITERATE_BACKWARDS_DECL)
749   ALL_OOP_OOP_ITERATE_CLOSURES_2(InstanceKlass_OOP_OOP_ITERATE_BACKWARDS_DECL)
750 #endif // !SERIALGC

752 private:
753   // initialization state
754 #ifdef ASSERT
755   void set_init_state(ClassState state);
756 #else
757   void set_init_state(ClassState state) { _init_state = state; }
758 #endif
759   void set_rewritten()                  { _rewritten = true; }
760   void set_init_thread(Thread *thread)  { _init_thread = thread; }

762   u2 idnum_allocated_count() const      { return _idnum_allocated_count; }
763   // The RedefineClasses() API can cause new method idnums to be needed
764   // which will cause the caches to grow. Safety requires different
765   // cache management logic if the caches can grow instead of just
766   // going from NULL to non-NULL.
767   bool idnum_can_increment() const      { return has_been_redefined(); }
768   jmethodID* methods_jmethod_ids_acquire() const
769          { return (jmethodID*)OrderAccess::load_ptr_acquire(&_methods_jmethod_id
770   void release_set_methods_jmethod_ids(jmethodID* jmeths)
771          { OrderAccess::release_store_ptr(&_methods_jmethod_ids, jmeths); }

773   int* methods_cached_itable_indices_acquire() const
774          { return (int*)OrderAccess::load_ptr_acquire(&_methods_cached_itable_in
775   void release_set_methods_cached_itable_indices(int* indices)
776          { OrderAccess::release_store_ptr(&_methods_cached_itable_indices, indic

778   inline typeArrayOop get_method_annotations_from(int idnum, objArrayOop annos);
779   void set_annotations(objArrayOop md, objArrayOop* md_p)  { oop_store_without_c
780   void set_methods_annotations_of(int idnum, typeArrayOop anno, objArrayOop* md_

782   // Offsets for memory management
783   oop* adr_array_klasses() const     { return (oop*)&this->_array_klasses;}
784   oop* adr_methods() const           { return (oop*)&this->_methods;}
785   oop* adr_method_ordering() const   { return (oop*)&this->_method_ordering;}
786   oop* adr_local_interfaces() const  { return (oop*)&this->_local_interfaces;}
787   oop* adr_transitive_interfaces() const  { return (oop*)&this->_transitive_inte
788   oop* adr_fields() const            { return (oop*)&this->_fields;}
```

```
 789   oop* adr_constants() const         { return (oop*)&this->_constants;}
 790   oop* adr_class_loader() const      { return (oop*)&this->_class_loader;}
 791   oop* adr_protection_domain() const { return (oop*)&this->_protection_domain;}
 792   oop* adr_host_klass() const        { return (oop*)&this->_host_klass;}
 793   oop* adr_signers() const           { return (oop*)&this->_signers;}
 794   oop* adr_inner_classes() const     { return (oop*)&this->_inner_classes;}
 795   oop* adr_implementors() const      { return (oop*)&this->_implementors[0];}
 796   oop* adr_bootstrap_method() const  { return (oop*)&this->_bootstrap_method;}
 797   oop* adr_methods_jmethod_ids() const            { return (oop*)&this->_method
 798   oop* adr_methods_cached_itable_indices() const   { return (oop*)&this->_method
 799   oop* adr_class_annotations() const   { return (oop*)&this->_class_annotations;
 800   oop* adr_fields_annotations() const  { return (oop*)&this->_fields_annotations
 801   oop* adr_methods_annotations() const { return (oop*)&this->_methods_annotation
 802   oop* adr_methods_parameter_annotations() const { return (oop*)&this->_methods_
 803   oop* adr_methods_default_annotations() const { return (oop*)&this->_methods_de

 805   // Static methods that are used to implement member methods where an exposed t
 806   // is needed due to possible GCs
 807   static bool link_class_impl                           (instanceKlassHandle thi
 808   static bool verify_code                               (instanceKlassHandle thi
 809   static void initialize_impl                           (instanceKlassHandle thi
 810   static void eager_initialize_impl                     (instanceKlassHandle thi
 811   static void set_initialization_state_and_notify_impl  (instanceKlassHandle thi
 812   static void call_class_initializer_impl               (instanceKlassHandle thi
 813   static klassOop array_klass_impl                      (instanceKlassHandle thi
 814   static void do_local_static_fields_impl               (instanceKlassHandle thi
 815   /* jni_id_for_impl for jfieldID only */
 816   static JNIid* jni_id_for_impl                         (instanceKlassHandle thi

 818   // Returns the array class for the n'th dimension
 819   klassOop array_klass_impl(bool or_null, int n, TRAPS);

 821   // Returns the array class with this class as element type
 822   klassOop array_klass_impl(bool or_null, TRAPS);

 824 public:
 825   // sharing support
 826   virtual void remove_unshareable_info();
 827   virtual void shared_symbols_iterate(SymbolClosure* closure);

 829   // jvm support
 830   jint compute_modifier_flags(TRAPS) const;

 832 public:
 833   // JVMTI support
 834   jint jvmti_class_status() const;

 836  public:
 837   // Printing
 838   void oop_print_value_on(oop obj, outputStream* st);
 839 #ifndef PRODUCT
 840   void oop_print_on      (oop obj, outputStream* st);

 842   void print_dependent_nmethods(bool verbose = false);
 843   bool is_dependent_nmethod(nmethod* nm);
 844 #endif

 846   // Verification
 847   const char* internal_name() const;
 848   void oop_verify_on(oop obj, outputStream* st);

 850 #ifndef PRODUCT
 851   static void verify_class_klass_nonstatic_oop_maps(klassOop k) PRODUCT_RETURN;
 852 #endif
 853 };
```

_____***unchanged_portion_omitted_***

****
20523 Wed Mar 30 07:00:23 2011
new/src/share/vm/oops/klass.cpp
****
_____***unchanged_portion_omitted_***

```
 499 const char* Klass::external_name() const {
 500   if (oop_is_instance()) {
 501     instanceKlass* ik = (instanceKlass*) this;
 502     if (ik->is_anonymous()) {
 503       assert(EnableInvokeDynamic, "");
 503       assert(AnonymousClasses, "");
 504       intptr_t hash = ik->java_mirror()->identity_hash();
 505       char     hash_buf[40];
 506       sprintf(hash_buf, "/" UINTX_FORMAT, (uintx)hash);
 507       size_t   hash_len = strlen(hash_buf);

 509       size_t result_len = name()->utf8_length();
 510       char*  result     = NEW_RESOURCE_ARRAY(char, result_len + hash_len + 1);
 511       name()->as_klass_external_name(result, (int) result_len + 1);
 512       assert(strlen(result) == result_len, "");
 513       strcpy(result + result_len, hash_buf);
 514       assert(strlen(result) == result_len + hash_len, "");
 515       return result;
 516     }
 517   }
 518   if (name() == NULL)  return "<unknown>";
 519   return name()->as_klass_external_name();
 520 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   41043 Wed Mar 30 07:00:24 2011
new/src/share/vm/oops/methodOop.hpp
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 1997, 2011, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.
   8  *
   9  * This code is distributed in the hope that it will be useful, but WITHOUT
  10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  12  * version 2 for more details (a copy is included in the LICENSE file that
  13  * accompanied this code).
  14  *
  15  * You should have received a copy of the GNU General Public License version
  16  * 2 along with this work; if not, write to the Free Software Foundation,
  17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  18  *
  19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  20  * or visit www.oracle.com if you need additional information or have any
  21  * questions.
  22  *
  23  */

  25 #ifndef SHARE_VM_OOPS_METHODOOP_HPP
  26 #define SHARE_VM_OOPS_METHODOOP_HPP

  28 #include "classfile/vmSymbols.hpp"
  29 #include "code/compressedStream.hpp"
  30 #include "compiler/oopMap.hpp"
  31 #include "interpreter/invocationCounter.hpp"
  32 #include "oops/constMethodOop.hpp"
  33 #include "oops/constantPoolOop.hpp"
  34 #include "oops/instanceKlass.hpp"
  35 #include "oops/oop.hpp"
  36 #include "oops/typeArrayOop.hpp"
  37 #include "utilities/accessFlags.hpp"
  38 #include "utilities/growableArray.hpp"

  40 // A methodOop represents a Java method.
  41 //
  42 // Memory layout (each line represents a word). Note that most applications load
  43 // so keeping the size of this structure small has a big impact on footprint.
  44 //
  45 // We put all oops and method_size first for better gc cache locality.
  46 //
  47 // The actual bytecodes are inlined after the end of the methodOopDesc struct.
  48 //
  49 // There are bits in the access_flags telling whether inlined tables are present
  50 // Note that accessing the line number and local variable tables is not performa
  51 // Accessing the checked exceptions table is used by reflection, so we put that
  52 // to it fast.
  53 //
  54 // The line number table is compressed and inlined following the byte codes. It
  55 // byte following the byte codes. The checked exceptions table and the local var
  56 // after the line number table, and indexed from the end of the method. We do no
  57 // exceptions table since the average length is less than 2, and do not bother t
  58 // variable table either since it is mostly absent.
  59 //
  60 // Note that native_function and signature_handler has to be at fixed offsets (r
  61 //
  62 // |------------------------------------------------------|
```



```
  63 // | header                                               |
  64 // | klass                                                |
  65 // |------------------------------------------------------|
  66 // | constMethodOop                 (oop)                 |
  67 // | constants                      (oop)                 |
  68 // |------------------------------------------------------|
  69 // | methodData                     (oop)                 |
  70 // | interp_invocation_count                              |
  71 // |------------------------------------------------------|
  72 // | access_flags                                         |
  73 // | vtable_index                                         |
  74 // |------------------------------------------------------|
  75 // | result_index (C++ interpreter only)                  |
  76 // |------------------------------------------------------|
  77 // | method_size             | max_stack                  |
  78 // | max_locals              | size_of_parameters         |
  79 // |------------------------------------------------------|
  80 // | intrinsic_id, (unused)  |  throwout_count            |
  81 // |------------------------------------------------------|
  82 // | num_breakpoints         |  (unused)                  |
  83 // |------------------------------------------------------|
  84 // | invocation_counter                                   |
  85 // | backedge_counter                                     |
  86 // |------------------------------------------------------|
  87 // |           prev_time (tiered only, 64 bit wide)       |
  88 // |                                                      |
  89 // |------------------------------------------------------|
  90 // |                  rate (tiered)                       |
  91 // |------------------------------------------------------|
  92 // | code                           (pointer)             |
  93 // | i2i                            (pointer)             |
  94 // | adapter                        (pointer)             |
  95 // | from_compiled_entry            (pointer)             |
  96 // | from_interpreted_entry         (pointer)             |
  97 // |------------------------------------------------------|
  98 // | native_function       (present only if native)       |
  99 // | signature_handler     (present only if native)       |
 100 // |------------------------------------------------------|
```

```
 103 class CheckedExceptionElement;
 104 class LocalVariableTableElement;
 105 class AdapterHandlerEntry;
 106 class methodDataOopDesc;

 108 class methodOopDesc : public oopDesc {
 109  friend class methodKlass;
 110  friend class VMStructs;
 111  private:
 112   constMethodOop    _constMethod;               // Method read-only data.
 113   constantPoolOop   _constants;                 // Constant pool
 114   methodDataOop     _method_data;
 115   int               _interpreter_invocation_count; // Count of times invoked (re
 116   AccessFlags       _access_flags;              // Access flags
 117   int               _vtable_index;              // vtable index of this method
 118                                                 // note: can have vtables with
 119 #ifdef CC_INTERP
 120   int               _result_index;              // C++ interpreter needs for co
 121 #endif
 122   u2                _method_size;               // size of this object
 123   u2                _max_stack;                 // Maximum number of entries on
 124   u2                _max_locals;                // Number of local variables us
 125   u2                _size_of_parameters;        // size of the parameter block
 126   u1                _intrinsic_id;              // vmSymbols::intrinsic_id (0 =
 127   u2                _interpreter_throwout_count; // Count of times method was ex
 128   u2                _number_of_breakpoints;     // fullspeed debugging support
```

```
129   InvocationCounter _invocation_counter;         // Incremented before each acti
130   InvocationCounter _backedge_counter;           // Incremented before each back

132 #ifdef TIERED
133   jlong             _prev_time;                   // Previous time the rate was
134   float             _rate;                        // Events (invocation and back
135 #endif

137 #ifndef PRODUCT
138   int               _compiled_invocation_count;  // Number of nmethod invocation
139 #endif
140   // Entry point for calling both from and to the interpreter.
141   address _i2i_entry;           // All-args-on-stack calling convention
142   // Adapter blob (i2c/c2i) for this methodOop. Set once when method is linked.
143   AdapterHandlerEntry* _adapter;
144   // Entry point for calling from compiled code, to compiled code if it exists
145   // or else the interpreter.
146   volatile address _from_compiled_entry;        // Cache of: _code ? _code->entr
147   // The entry point for calling both from and to compiled code is
148   // "_code->entry_point()".  Because of tiered compilation and de-opt, this
149   // field can come and go.  It can transition from NULL to not-null at any
150   // time (whenever a compile completes).  It can transition from not-null to
151   // NULL only at safepoints (because of a de-opt).
152   nmethod* volatile _code;                      // Points to the corresponding
153   volatile address           _from_interpreted_entry; // Cache of _code ? _adapt

155  public:

157   // accessors for instance variables
158   constMethodOop constMethod() const            { return _constMethod; }
159   void set_constMethod(constMethodOop xconst)    { oop_store_without_check((oop*


162   static address make_adapters(methodHandle mh, TRAPS);
163   volatile address from_compiled_entry() const   { return (address)OrderAccess::
164   volatile address from_interpreted_entry() const{ return (address)OrderAccess::

166   // access flag
167   AccessFlags access_flags() const              { return _access_flags;  }
168   void set_access_flags(AccessFlags flags)       { _access_flags = flags; }

170   // name
171   Symbol* name() const                          { return _constants->symbol_at(
172   int name_index() const                        { return constMethod()->name_in
173   void set_name_index(int index)                { constMethod()->set_name_index

175   // signature
176   Symbol* signature() const                     { return _constants->symbol_at(
177   int signature_index() const                   { return constMethod()->signatu
178   void set_signature_index(int index)           { constMethod()->set_signature_

180   // generics support
181   Symbol* generic_signature() const             { int idx = generic_signature_i
182   int generic_signature_index() const           { return constMethod()->generic
183   void set_generic_signature_index(int index)    { constMethod()->set_generic_si

185   // annotations support
186   typeArrayOop annotations() const              { return instanceKlass::cast(me
187   typeArrayOop parameter_annotations() const     { return instanceKlass::cast(me
188   typeArrayOop annotation_default() const        { return instanceKlass::cast(me

190 #ifdef CC_INTERP
191   void set_result_index(BasicType type);
192   int  result_index()                           { return _result_index; }
193 #endif
```



```
195   // Helper routine: get klass name + "." + method name + signature as
196   // C string, for the purpose of providing more useful NoSuchMethodErrors
197   // and fatal error handling. The string is allocated in resource
198   // area if a buffer is not provided by the caller.
199   char* name_and_sig_as_C_string();
200   char* name_and_sig_as_C_string(char* buf, int size);

202   // Static routine in the situations we don't have a methodOop
203   static char* name_and_sig_as_C_string(Klass* klass, Symbol* method_name, Symbo
204   static char* name_and_sig_as_C_string(Klass* klass, Symbol* method_name, Symbo

206   Bytecodes::Code java_code_at(int bci) const {
207     return Bytecodes::java_code_at(this, bcp_from(bci));
208   }
209   Bytecodes::Code code_at(int bci) const {
210     return Bytecodes::code_at(this, bcp_from(bci));
211   }

213   // JVMTI breakpoints
214   Bytecodes::Code orig_bytecode_at(int bci) const;
215   void        set_orig_bytecode_at(int bci, Bytecodes::Code code);
216   void set_breakpoint(int bci);
217   void clear_breakpoint(int bci);
218   void clear_all_breakpoints();
219   // Tracking number of breakpoints, for fullspeed debugging.
220   // Only mutated by VM thread.
221   u2   number_of_breakpoints() const             { return _number_of_breakpoints
222   void incr_number_of_breakpoints()              { ++_number_of_breakpoints; }
223   void decr_number_of_breakpoints()              { --_number_of_breakpoints; }
224   // Initialization only
225   void clear_number_of_breakpoints()             { _number_of_breakpoints = 0; }

227   // index into instanceKlass methods() array
228   u2 method_idnum() const           { return constMethod()->method_idnum(); }
229   void set_method_idnum(u2 idnum)   { constMethod()->set_method_idnum(idnum); }

231   // code size
232   int code_size() const                 { return constMethod()->code_size(); }

234   // method size
235   int method_size() const                       { return _method_size; }
236   void set_method_size(int size) {
237     assert(0 <= size && size < (1 << 16), "invalid method size");
238     _method_size = size;
239   }

241   // constant pool for klassOop holding this method
242   constantPoolOop constants() const             { return _constants; }
243   void set_constants(constantPoolOop c)         { oop_store_without_check((oop*

245   // max stack
246   int  max_stack() const                        { return _max_stack; }
247   void set_max_stack(int size)                  { _max_stack = size; }

249   // max locals
250   int  max_locals() const                       { return _max_locals; }
251   void set_max_locals(int size)                 { _max_locals = size; }

253   int highest_comp_level() const;
254   void set_highest_comp_level(int level);
255   int highest_osr_comp_level() const;
256   void set_highest_osr_comp_level(int level);

258   // Count of times method was exited via exception while interpreting
259   void interpreter_throwout_increment() {
260     if (_interpreter_throwout_count < 65534) {
```

```
261       _interpreter_throwout_count++;
262     }
263   }

265   int  interpreter_throwout_count() const        { return _interpreter_throwout_
266   void set_interpreter_throwout_count(int count) { _interpreter_throwout_count =

268   // size of parameters
269   int  size_of_parameters() const                { return _size_of_parameters; }

271   bool has_stackmap_table() const {
272     return constMethod()->has_stackmap_table();
273   }

275   typeArrayOop stackmap_data() const {
276     return constMethod()->stackmap_data();
277   }

279   void set_stackmap_data(typeArrayOop sd) {
280     constMethod()->set_stackmap_data(sd);
281   }

283   // exception handler table
284   typeArrayOop exception_table() const
285                                  { return constMethod()->exception_table(); }
286   void set_exception_table(typeArrayOop e)
287                                { constMethod()->set_exception_table(e); }
288   bool has_exception_handler() const
289                              { return constMethod()->has_exception_handler(); }

291   // Finds the first entry point bci of an exception handler for an
292   // exception of klass ex_klass thrown at throw_bci. A value of NULL
293   // for ex_klass indicates that the exception klass is not known; in
294   // this case it matches any constraint class. Returns -1 if the
295   // exception cannot be handled in this method. The handler
296   // constraint classes are loaded if necessary. Note that this may
297   // throw an exception if loading of the constraint classes causes
298   // an IllegalAccessError (bugid 4307310) or an OutOfMemoryError.
299   // If an exception is thrown, returns the bci of the
300   // exception handler which caused the exception to be thrown, which
301   // is needed for proper retries. See, for example,
302   // InterpreterRuntime::exception_handler_for_exception.
303   int fast_exception_handler_bci_for(KlassHandle ex_klass, int throw_bci, TRAPS)

305   // method data access
306   methodDataOop method_data() const              {
307     return _method_data;
308   }
309   void set_method_data(methodDataOop data)       {
310     oop_store_without_check((oop*)&_method_data, (oop)data);
311   }

313   // invocation counter
314   InvocationCounter* invocation_counter() { return &_invocation_counter; }
315   InvocationCounter* backedge_counter()   { return &_backedge_counter; }

317 #ifdef TIERED
318   // We are reusing interpreter_invocation_count as a holder for the previous ev
319   // We can do that since interpreter_invocation_count is not used in tiered.
320   int prev_event_count() const                   { return _interpreter_invocatio
321   void set_prev_event_count(int count)           { _interpreter_invocation_count
322   jlong prev_time() const                        { return _prev_time; }
323   void set_prev_time(jlong time)                 { _prev_time = time; }
324   float rate() const                             { return _rate; }
325   void set_rate(float rate)                      { _rate = rate; }
326 #endif
```



```
328   int invocation_count();
329   int backedge_count();

331   bool was_executed_more_than(int n);
332   bool was_never_executed()                      { return !was_executed_more_tha

334   static void build_interpreter_method_data(methodHandle method, TRAPS);

336   int interpreter_invocation_count() {
337     if (TieredCompilation) return invocation_count();
338     else return _interpreter_invocation_count;
339   }
340   void set_interpreter_invocation_count(int count) { _interpreter_invocation_cou
341   int increment_interpreter_invocation_count() {
342     if (TieredCompilation) ShouldNotReachHere();
343     return ++_interpreter_invocation_count;
344   }

346 #ifndef PRODUCT
347   int  compiled_invocation_count() const         { return _compiled_invocation_c
348   void set_compiled_invocation_count(int count)  { _compiled_invocation_count =
349 #endif // not PRODUCT

351   // Clear (non-shared space) pointers which could not be relevant
352   // if this (shared) method were mapped into another JVM.
353   void remove_unshareable_info();

355   // nmethod/verified compiler entry
356   address verified_code_entry();
357   bool check_code() const;      // Not inline to avoid circular ref
358   nmethod* volatile code() const                 { assert( check_code(), "" ); r
359   void clear_code();            // Clear out any compiled code
360   static void set_code(methodHandle mh, nmethod* code);
361   void set_adapter_entry(AdapterHandlerEntry* adapter) {  _adapter = adapter; }
362   address get_i2c_entry();
363   address get_c2i_entry();
364   address get_c2i_unverified_entry();
365   AdapterHandlerEntry* adapter() {  return _adapter; }
366   // setup entry points
367   void link_method(methodHandle method, TRAPS);
368   // clear entry points. Used by sharing code
369   void unlink_method();

371   // vtable index
372   enum VtableIndexFlag {
373     // Valid vtable indexes are non-negative (>= 0).
374     // These few negative values are used as sentinels.
375     highest_unused_vtable_index_value = -5,
376     invalid_vtable_index    = -4,  // distinct from any valid vtable index
377     garbage_vtable_index    = -3,  // not yet linked; no vtable layout yet
378     nonvirtual_vtable_index = -2   // there is no need for vtable dispatch
379     // 6330203 Note:  Do not use -1, which was overloaded with many meanings.
380   };
381   DEBUG_ONLY(bool valid_vtable_index() const     { return _vtable_index >= nonvi
382   int  vtable_index() const                      { assert(valid_vtable_index(),
383                                                    return _vtable_index; }
384   void set_vtable_index(int index)               { _vtable_index = index; }

386   // interpreter entry
387   address interpreter_entry() const              { return _i2i_entry; }
388   // Only used when first initialize so we can set _i2i_entry and _from_interpre
389   void set_interpreter_entry(address entry)      { _i2i_entry = entry;  _from_in
390   int  interpreter_kind(void) {
391     return constMethod()->interpreter_kind();
392   }
```

```
393   void set_interpreter_kind();
394   void set_interpreter_kind(int kind) {
395     constMethod()->set_interpreter_kind(kind);
396   }

398   // native function (used for native methods only)
399   enum {
400     native_bind_event_is_interesting = true
401   };
402   address native_function() const                { return *(native_function_addr
403   // Must specify a real function (not NULL).
404   // Use clear_native_function() to unregister.
405   void set_native_function(address function, bool post_event_flag);
406   bool has_native_function() const;
407   void clear_native_function();

409   // signature handler (used for native methods only)
410   address signature_handler() const              { return *(signature_handler_ad
411   void set_signature_handler(address handler);

413   // Interpreter oopmap support
414   void mask_for(int bci, InterpreterOopMap* mask);

416 #ifndef PRODUCT
417   // operations on invocation counter
418   void print_invocation_count();
419 #endif

421   // byte codes
422   void    set_code(address code)      { return constMethod()->set_code(code); }
423   address code_base() const           { return constMethod()->code_base(); }
424   bool    contains(address bcp) const { return constMethod()->contains(bcp); }

426   // prints byte codes
427   void print_codes() const            { print_codes_on(tty); }
428   void print_codes_on(outputStream* st) const                    PRODUCT_RETUR
429   void print_codes_on(int from, int to, outputStream* st) const  PRODUCT_RETUR

431   // checked exceptions
432   int checked_exceptions_length() const
433                           { return constMethod()->checked_exceptions_length(); }
434   CheckedExceptionElement* checked_exceptions_start() const
435                            { return constMethod()->checked_exceptions_start(); }

437   // localvariable table
438   bool has_localvariable_table() const
439                            { return constMethod()->has_localvariable_table(); }
440   int localvariable_table_length() const
441                          { return constMethod()->localvariable_table_length(); }
442   LocalVariableTableElement* localvariable_table_start() const
443                           { return constMethod()->localvariable_table_start(); }

445   bool has_linenumber_table() const
446                                { return constMethod()->has_linenumber_table(); }
447   u_char* compressed_linenumber_table() const
448                         { return constMethod()->compressed_linenumber_table(); }

450   // method holder (the klassOop holding this method)
451   klassOop method_holder() const                 { return _constants->pool_holde

453   void compute_size_of_parameters(Thread *thread); // word size of parameters (r
454   Symbol* klass_name() const;                    // returns the name of the meth
455   BasicType result_type() const;                 // type of the method result
456   int result_type_index() const;                 // type index of the method res
457   bool is_returning_oop() const                  { BasicType r = result_type();
458   bool is_returning_fp() const                   { BasicType r = result_type();
```



```
460   // Checked exceptions thrown by this method (resolved to mirrors)
461   objArrayHandle resolved_checked_exceptions(TRAPS) { return resolved_checked_ex

463   // Access flags
464   bool is_public() const                         { return access_flags().is_publ
465   bool is_private() const                        { return access_flags().is_priv
466   bool is_protected() const                      { return access_flags().is_prot
467   bool is_package_private() const                { return !is_public() && !is_pr
468   bool is_static() const                         { return access_flags().is_stat
469   bool is_final() const                          { return access_flags().is_fina
470   bool is_synchronized() const                   { return access_flags().is_sync
471   bool is_native() const                         { return access_flags().is_nati
472   bool is_abstract() const                       { return access_flags().is_abst
473   bool is_strict() const                         { return access_flags().is_stri
474   bool is_synthetic() const                      { return access_flags().is_synt

476   // returns true if contains only return operation
477   bool is_empty_method() const;

479   // returns true if this is a vanilla constructor
480   bool is_vanilla_constructor() const;

482   // checks method and its method holder
483   bool is_final_method() const;
484   bool is_strict_method() const;

486   // true if method needs no dynamic dispatch (final and/or no vtable entry)
487   bool can_be_statically_bound() const;

489   // returns true if the method has any backward branches.
490   bool has_loops() {
491     return access_flags().loops_flag_init() ? access_flags().has_loops() : compu
492   };

494   bool compute_has_loops_flag();

496   bool has_jsrs() {
497     return access_flags().has_jsrs();
498   };
499   void set_has_jsrs() {
500     _access_flags.set_has_jsrs();
501   }

503   // returns true if the method has any monitors.
504   bool has_monitors() const                      { return is_synchronized() || a
505   bool has_monitor_bytecodes() const             { return access_flags().has_mon

507   void set_has_monitor_bytecodes()               { _access_flags.set_has_monitor

509   // monitor matching. This returns a conservative estimate of whether the monit
510   // properly nest in the method. It might return false, even though they actu
511   // has not been computed yet.
512   bool guaranteed_monitor_matching() const       { return access_flags().is_moni
513   void set_guaranteed_monitor_matching()         { _access_flags.set_monitor_mat

515   // returns true if the method is an accessor function (setter/getter).
516   bool is_accessor() const;

518   // returns true if the method is an initializer (<init> or <clinit>).
519   bool is_initializer() const;

521   // returns true if the method is static OR if the classfile version < 51
522   bool has_valid_initializer_flags() const;

524   // returns true if the method name is <clinit> and the method has
```

```
525   // valid static initializer flags.
526   bool is_static_initializer() const;
527
528   // compiled code support
529   // NOTE: code() is inherently racy as deopt can be clearing code
530   // simultaneously. Use with caution.
531   bool has_compiled_code() const                 { return code() != NULL; }
532
533   // sizing
534   static int object_size(bool is_native);
535   static int header_size()                       { return sizeof(methodOopDesc)/
536   int object_size() const                        { return method_size(); }
537
538   bool object_is_parsable() const                { return method_size() > 0; }
539
540   // interpreter support
541   static ByteSize const_offset()                 { return byte_offset_of(methodO
542   static ByteSize constants_offset()             { return byte_offset_of(methodO
543   static ByteSize access_flags_offset()          { return byte_offset_of(methodO
544 #ifdef CC_INTERP
545   static ByteSize result_index_offset()          { return byte_offset_of(methodO
546 #endif /* CC_INTERP */
547   static ByteSize size_of_locals_offset()        { return byte_offset_of(methodO
548   static ByteSize size_of_parameters_offset()    { return byte_offset_of(methodO
549   static ByteSize from_compiled_offset()         { return byte_offset_of(methodO
550   static ByteSize code_offset()                  { return byte_offset_of(methodO
551   static ByteSize invocation_counter_offset()    { return byte_offset_of(methodO
552   static ByteSize backedge_counter_offset()      { return byte_offset_of(methodO
553   static ByteSize method_data_offset()           {
554     return byte_offset_of(methodOopDesc, _method_data);
555   }
556   static ByteSize interpreter_invocation_counter_offset() { return byte_offset_o
557 #ifndef PRODUCT
558   static ByteSize compiled_invocation_counter_offset() { return byte_offset_of(m
559 #endif // not PRODUCT
560   static ByteSize native_function_offset()       { return in_ByteSize(sizeof(met
561   static ByteSize from_interpreted_offset()      { return byte_offset_of(methodO
562   static ByteSize interpreter_entry_offset()     { return byte_offset_of(methodO
563   static ByteSize signature_handler_offset()     { return in_ByteSize(sizeof(met
564   static ByteSize max_stack_offset()             { return byte_offset_of(methodO
565
566   // for code generation
567   static int method_data_offset_in_bytes()       { return offset_of(methodOopDes
568   static int interpreter_invocation_counter_offset_in_bytes()
569                                                  { return offset_of(methodOopDes
570   static int intrinsic_id_offset_in_bytes()      { return offset_of(methodOopDes
571   static int intrinsic_id_size_in_bytes()        { return sizeof(u1); }
572
573   // Static methods that are used to implement member methods where an exposed t
574   // is needed due to possible GCs
575   static objArrayHandle resolved_checked_exceptions_impl(methodOop this_oop, TRA
576
577   // Returns the byte code index from the byte code pointer
578   int     bci_from(address bcp) const;
579   address bcp_from(int     bci) const;
580   int validate_bci_from_bcx(intptr_t bcx) const;
581
582   // Returns the line number for a bci if debugging information for the method i
583   // -1 is returned otherwise.
584   int line_number_from_bci(int bci) const;
585
586   // Reflection support
587   bool is_overridden_in(klassOop k) const;
588
589   // JSR 292 support
590   bool is_method_handle_invoke() const             { return access_flags().is_m
```



```
591   static bool is_method_handle_invoke_name(vmSymbols::SID name_sid);
592   static bool is_method_handle_invoke_name(Symbol* name) {
593     return is_method_handle_invoke_name(vmSymbols::find_sid(name));
594   }
595   // Tests if this method is an internal adapter frame from the
596   // MethodHandleCompiler.
597   bool is_method_handle_adapter() const;
598   static methodHandle make_invoke_method(KlassHandle holder,
599                                          Symbol* name, //invokeExact or invokeGe
600                                          Symbol* signature, //anything at all
601                                          Handle method_type,
602                                          TRAPS);
603   // these operate only on invoke methods:
604   oop method_handle_type() const;
605   static jint* method_type_offsets_chain();  // series of pointer-offsets, termi
606   // presize interpreter frames for extra interpreter stack entries, if needed
607   // method handles want to be able to push a few extra values (e.g., a bound re
608   // invokedynamic sometimes needs to push a bootstrap method, call site, and ar
609   // all without checking for a stack overflow
610   static int extra_stack_entries() { return EnableInvokeDynamic ? (int) MethodHa
610   static int extra_stack_entries() { return (EnableMethodHandles ? (int)MethodHa
611   static int extra_stack_words();  // = extra_stack_entries() * Interpreter::sta
612
613   // RedefineClasses() support:
614   bool is_old() const                                { return access_flags().is_o
615   void set_is_old()                                  { _access_flags.set_is_old()
616   bool is_obsolete() const                           { return access_flags().is_o
617   void set_is_obsolete()                             { _access_flags.set_is_obsol
618   // see the definition in methodOop.cpp for the gory details
619   bool should_not_be_cached() const;
620
621   // JVMTI Native method prefixing support:
622   bool is_prefixed_native() const                    { return access_flags().is_p
623   void set_is_prefixed_native()                      { _access_flags.set_is_prefi
624
625   // Rewriting support
626   static methodHandle clone_with_new_data(methodHandle m, u_char* new_code, int
627                                           u_char* new_compressed_linenumber_tabl
628
629   // Get this method's jmethodID -- allocate if it doesn't exist
630   jmethodID jmethod_id()                             { methodHandle this_h(this);
631                                                        return instanceKlass::get_
632
633   // Lookup the jmethodID for this method.  Return NULL if not found.
634   // NOTE that this function can be called from a signal handler
635   // (see AsyncGetCallTrace support for Forte Analyzer) and this
636   // needs to be async-safe. No allocation should be done and
637   // so handles are not used to avoid deadlock.
638   jmethodID find_jmethod_id_or_null()                { return instanceKlass::cast
639
640   // JNI static invoke cached itable index accessors
641   int cached_itable_index()                          { return instanceKlass::cast
642   void set_cached_itable_index(int index)            { instanceKlass::cast(method
643
644   // Support for inlining of intrinsic methods
645   vmIntrinsics::ID intrinsic_id() const          { return (vmIntrinsics::ID) _in
646   void     set_intrinsic_id(vmIntrinsics::ID id) {                          _in
647
648   // Helper routines for intrinsic_id() and vmIntrinsics::method().
649   void init_intrinsic_id();     // updates from _none if a match
650   static vmSymbols::SID klass_id_for_intrinsics(klassOop holder);
651
652   // On-stack replacement support
653   bool has_osr_nmethod(int level, bool match_level) {
654    return instanceKlass::cast(method_holder())->lookup_osr_nmethod(this, Invocat
655   }
```

```
 657   nmethod* lookup_osr_nmethod_for(int bci, int level, bool match_level) {
 658     return instanceKlass::cast(method_holder())->lookup_osr_nmethod(this, bci, l
 659   }

 661   // Inline cache support
 662   void cleanup_inline_caches();

 664   // Find if klass for method is loaded
 665   bool is_klass_loaded_by_klass_index(int klass_index) const;
 666   bool is_klass_loaded(int refinfo_index, bool must_be_resolved = false) const;

 668   // Indicates whether compilation failed earlier for this method, or
 669   // whether it is not compilable for another reason like having a
 670   // breakpoint set in it.
 671   bool is_not_compilable(int comp_level = CompLevel_any) const;
 672   void set_not_compilable(int comp_level = CompLevel_all, bool report = true);
 673   void set_not_compilable_quietly(int comp_level = CompLevel_all) {
 674     set_not_compilable(comp_level, false);
 675   }
 676   bool is_not_osr_compilable(int comp_level = CompLevel_any) const {
 677     return is_not_compilable(comp_level) || access_flags().is_not_osr_compilable
 678   }
 679   void set_not_osr_compilable()               { _access_flags.set_not_osr_compil
 680   bool is_not_c1_compilable() const           { return access_flags().is_not_c1_
 681   void set_not_c1_compilable()                { _access_flags.set_not_c1_compila
 682   bool is_not_c2_compilable() const           { return access_flags().is_not_c2_
 683   void set_not_c2_compilable()                { _access_flags.set_not_c2_compila

 685   // Background compilation support
 686   bool queued_for_compilation() const  { return access_flags().queued_for_compil
 687   void set_queued_for_compilation()    { _access_flags.set_queued_for_compilatio
 688   void clear_queued_for_compilation()  { _access_flags.clear_queued_for_compilat

 690   // Resolve all classes in signature, return 'true' if successful
 691   static bool load_signature_classes(methodHandle m, TRAPS);

 693   // Return if true if not all classes references in signature, including return
 694   static bool has_unloaded_classes_in_signature(methodHandle m, TRAPS);

 696   // Printing
 697   void print_short_name(outputStream* st)        /*PRODUCT_RETURN*/; // prints a
 698   void print_name(outputStream* st)              PRODUCT_RETURN; // prints as "v

 700   // Helper routine used for method sorting
 701   static void sort_methods(objArrayOop methods,
 702                            objArrayOop methods_annotations,
 703                            objArrayOop methods_parameter_annotations,
 704                            objArrayOop methods_default_annotations,
 705                            bool idempotent = false);

 707   // size of parameters
 708   void set_size_of_parameters(int size)         { _size_of_parameters = size; }
 709  private:

 711   // Inlined elements
 712   address* native_function_addr() const         { assert(is_native(), "must be
 713   address* signature_handler_addr() const       { return native_function_addr()

 715   // Garbage collection support
 716   oop*  adr_constMethod() const                 { return (oop*)&_constMethod;
 717   oop*  adr_constants() const                   { return (oop*)&_constants;
 718   oop*  adr_method_data() const                 { return (oop*)&_method_data;
 719 };
```

*unchanged_portion_omitted_*

```
**********************************************************
  113920 Wed Mar 30 07:00:28 2011
new/src/share/vm/prims/methodHandles.cpp
**********************************************************
______unchanged_portion_omitted_
 108 #endif


 111 //------------------------------------------------------------------------------
 112 // MethodHandles::generate_adapters
 113 //
 114 void MethodHandles::generate_adapters() {
 115   if (!EnableInvokeDynamic || SystemDictionary::MethodHandle_klass() == NULL)  r
 115   if (!EnableMethodHandles || SystemDictionary::MethodHandle_klass() == NULL)  r

 117   assert(_adapter_code == NULL, "generate only once");

 119   ResourceMark rm;
 120   TraceTime timer("MethodHandles adapters generation", TraceStartupTime);
 121   _adapter_code = MethodHandlesAdapterBlob::create(_adapter_code_size);
 122   if (_adapter_code == NULL)
 123     vm_exit_out_of_memory(_adapter_code_size, "CodeCache: no room for MethodHand
 124   CodeBuffer code(_adapter_code);
 125   MethodHandlesAdapterGenerator g(&code);
 126   g.generate();
 127 }
______unchanged_portion_omitted_


 144 void MethodHandles::set_enabled(bool z) {
 145   if (_enabled != z) {
 146     guarantee(z && EnableInvokeDynamic, "can only enable once, and only if -XX:+
 146     guarantee(z && EnableMethodHandles, "can only enable once, and only if -XX:+
 147     _enabled = z;
 148   }
 149 }
______unchanged_portion_omitted_

2582 // More entry points specifically for EnableInvokeDynamic.
2582 // FIXME: Remove methods2 after AllowTransitionalJSR292 is removed.
2583 static JNINativeMethod methods2[] = {
2584   {CC"registerBootstrap",       CC"("CLS MH")V",                FN_PTR(MHN_regis
2585   {CC"getBootstrap",            CC"("CLS")"MH,                  FN_PTR(MHN_getBo
2586   {CC"setCallSiteTarget",       CC"("CST MH")V",                FN_PTR(MHN_setCa
2587 };
______unchanged_portion_omitted_

2615 // This one function is exported, used by NativeLookup.

2617 JVM_ENTRY(void, JVM_RegisterMethodHandleMethods(JNIEnv *env, jclass MHN_class))
2618   assert(MethodHandles::spot_check_entry_names(), "entry enum is OK");

2620   if (!EnableInvokeDynamic) {
2621     warning("JSR 292 is disabled in this JVM.  Use -XX:+UnlockDiagnosticVMOption
2621   // note: this explicit warning-producing stuff will be replaced by auto-detect

2623   if (!EnableMethodHandles) {
2624     warning("JSR 292 method handles are disabled in this JVM.  Use -XX:+UnlockEx
2622     return;  // bind nothing
2623   }

2625   if (SystemDictionary::MethodHandleNatives_klass() != NULL &&
2626       SystemDictionary::MethodHandleNatives_klass() != java_lang_Class::as_klass
2627     warning("multiple versions of MethodHandleNatives in boot classpath; conside
2628     THROW_MSG(vmSymbols::java_lang_InternalError(), "multiple versions of Method
2629   }
```



```
2631   bool enable_MH = true;

2633   // Loop control.  FIXME: Replace by dead reckoning after AllowTransitionalJSR2
2634   bool registered_natives = false;
2635   bool try_plain = true, try_JDYN = true, try_IDYN = true;
2636   for (;;) {
2637     ThreadToNativeFromVM ttnfv(thread);

2639     if      (try_plain) { try_plain = false; }
2640     else if (try_JDYN)  { try_JDYN  = false; hack_signatures(methods, sizeof(met
2641     else if (try_IDYN)  { try_IDYN  = false; hack_signatures(methods, sizeof(met
2642     else                { break; }
2643     int status = env->RegisterNatives(MHN_class, methods, sizeof(methods)/sizeof
2644     if (env->ExceptionOccurred()) {
2645       env->ExceptionClear();
2646       // and try again...
2647     } else {
2648       registered_natives = true;
2649       break;
2650     }
2651   }
2652   if (!registered_natives) {
2653     MethodHandles::set_enabled(false);
2654     warning("JSR 292 method handle code is mismatched to this JVM.  Disabling su
2655     enable_MH = false;
2656   }

2658   if (enable_MH) {
2659     bool found_raise_exception = false;
2660     KlassHandle MHN_klass = SystemDictionaryHandles::MethodHandleNatives_klass()
2661     KlassHandle MHI_klass = SystemDictionaryHandles::MethodHandleImpl_klass();
2662     // Loop control.  FIXME: Replace by dead reckoning after AllowTransitionalJS
2663     bool try_MHN = true, try_MHI = AllowTransitionalJSR292;
2664     for (;;) {
2665       KlassHandle try_klass;
2666       if      (try_MHN) { try_MHN = false; try_klass = MHN_klass; }
2667       else if (try_MHI) { try_MHI = false; try_klass = MHI_klass; }
2668       else              { break; }
2669       if (try_klass.is_null())  continue;
2670       TempNewSymbol raiseException_name = SymbolTable::new_symbol("raiseExceptio
2671       TempNewSymbol raiseException_sig = SymbolTable::new_symbol("(ILjava/lang/O
2672       methodOop raiseException_method  = instanceKlass::cast(try_klass->as_klass
2673                     ->find_method(raiseException_name, raiseException_sig);
2674       if (raiseException_method != NULL && raiseException_method->is_static()) {
2675         MethodHandles::set_raise_exception_method(raiseException_method);
2676         found_raise_exception = true;
2677         break;
2678       }
2679     }
2680     if (!found_raise_exception) {
2681       warning("JSR 292 method handle code is mismatched to this JVM.  Disabling
2682       enable_MH = false;
2683     }
2684   }

2686   if (enable_MH) {
2687     if (AllowTransitionalJSR292) {
2688       // We need to link the MethodHandleImpl klass before we generate
2689       // the method handle adapters as the _raise_exception adapter uses
2690       // one of its methods (and its c2i-adapter).
2691       klassOop k = SystemDictionary::MethodHandleImpl_klass();
2692       if (k != NULL) {
2693         instanceKlass* ik = instanceKlass::cast(k);
2694         ik->link_class(CHECK);
2695       }
```

```
2696     }

2698     MethodHandles::generate_adapters();
2699     MethodHandles::set_enabled(true);
2700   }

2705   if (!EnableInvokeDynamic) {
2706     warning("JSR 292 invokedynamic is disabled in this JVM.  Use -XX:+UnlockExpe
2707     return;  // bind nothing
2708   }

2702   if (AllowTransitionalJSR292) {
2703     ThreadToNativeFromVM ttnfv(thread);

2705     int status = env->RegisterNatives(MHN_class, methods2, sizeof(methods2)/size
2706     if (env->ExceptionOccurred()) {
2707       // Don't do this, since it's too late:
2708       //   MethodHandles::set_enabled(false)
2709       env->ExceptionClear();
2710     }
2711   }
2712 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   62884 Wed Mar 30 07:00:29 2011
new/src/share/vm/prims/unsafe.cpp
**********************************************************
______unchanged_portion_omitted_

1502 #undef CC
1503 #undef FN_PTR

1505 #undef ADR
1506 #undef LANG
1507 #undef OBJ
1508 #undef CLS
1509 #undef CTR
1510 #undef FLD
1511 #undef MTH
1512 #undef THR
1513 #undef DC0_Args
1514 #undef DC1_Args

1516 #undef DECLARE_GETSETOOP
1517 #undef DECLARE_GETSETNATIVE


1520 // This one function is exported, used by NativeLookup.
1521 // The Unsafe_xxx functions above are called only from the interpreter.
1522 // The optimizer looks at names and signatures to recognize
1523 // individual functions.

1525 JVM_ENTRY(void, JVM_RegisterUnsafeMethods(JNIEnv *env, jclass unsafecls))
1526   UnsafeWrapper("JVM_RegisterUnsafeMethods");
1527   {
1528     ThreadToNativeFromVM ttnfv(thread);
1529     {
1530       env->RegisterNatives(unsafecls, loadavg_method, sizeof(loadavg_method)/siz
1531       if (env->ExceptionOccurred()) {
1532         if (PrintMiscellaneous && (Verbose || WizardMode)) {
1533           tty->print_cr("Warning:  SDK 1.6 Unsafe.loadavg not found.");
1534         }
1535         env->ExceptionClear();
1536       }
1537     }
1538     {
1539       env->RegisterNatives(unsafecls, prefetch_methods, sizeof(prefetch_methods)
1540       if (env->ExceptionOccurred()) {
1541         if (PrintMiscellaneous && (Verbose || WizardMode)) {
1542           tty->print_cr("Warning:  SDK 1.6 Unsafe.prefetchRead/Write not found."
1543         }
1544         env->ExceptionClear();
1545       }
1546     }
1547     {
1548       env->RegisterNatives(unsafecls, memcopy_methods, sizeof(memcopy_methods)/s
1549       if (env->ExceptionOccurred()) {
1550         if (PrintMiscellaneous && (Verbose || WizardMode)) {
1551           tty->print_cr("Warning:  SDK 1.7 Unsafe.copyMemory not found.");
1552         }
1553         env->ExceptionClear();
1554         env->RegisterNatives(unsafecls, memcopy_methods_15, sizeof(memcopy_metho
1555         if (env->ExceptionOccurred()) {
1556           if (PrintMiscellaneous && (Verbose || WizardMode)) {
1557             tty->print_cr("Warning:  SDK 1.5 Unsafe.copyMemory not found.");
1558           }
1559           env->ExceptionClear();
1560         }
1561       }
```



```
1562     }
1563     if (EnableInvokeDynamic) {
1563     if (AnonymousClasses) {
1564       env->RegisterNatives(unsafecls, anonk_methods, sizeof(anonk_methods)/sizeo
1565       if (env->ExceptionOccurred()) {
1566         if (PrintMiscellaneous && (Verbose || WizardMode)) {
1567           tty->print_cr("Warning:  SDK 1.7 Unsafe.defineClass (anonymous version
1568         }
1569         env->ExceptionClear();
1570       }
1571     }
1572     int status = env->RegisterNatives(unsafecls, methods, sizeof(methods)/sizeof
1573     if (env->ExceptionOccurred()) {
1574       if (PrintMiscellaneous && (Verbose || WizardMode)) {
1575         tty->print_cr("Warning:  SDK 1.6 version of Unsafe not found.");
1576       }
1577       env->ExceptionClear();
1578       // %%% For now, be backward compatible with an older class:
1579       status = env->RegisterNatives(unsafecls, methods_15, sizeof(methods_15)/si
1580     }
1581     if (env->ExceptionOccurred()) {
1582       if (PrintMiscellaneous && (Verbose || WizardMode)) {
1583         tty->print_cr("Warning:  SDK 1.5 version of Unsafe not found.");
1584       }
1585       env->ExceptionClear();
1586       // %%% For now, be backward compatible with an older class:
1587       status = env->RegisterNatives(unsafecls, methods_141, sizeof(methods_141)/
1588     }
1589     if (env->ExceptionOccurred()) {
1590       if (PrintMiscellaneous && (Verbose || WizardMode)) {
1591         tty->print_cr("Warning:  SDK 1.4.1 version of Unsafe not found.");
1592       }
1593       env->ExceptionClear();
1594       // %%% For now, be backward compatible with an older class:
1595       status = env->RegisterNatives(unsafecls, methods_140, sizeof(methods_140)/
1596     }
1597     guarantee(status == 0, "register unsafe natives");
1598   }
1599 JVM_END
```

```
**********************************************************
  124648 Wed Mar 30 07:00:30 2011
new/src/share/vm/runtime/arguments.cpp
**********************************************************
______unchanged_portion_omitted_

2868 // Parse entry point called from JNI_CreateJavaVM

2870 jint Arguments::parse(const JavaVMInitArgs* args) {

2872   // Sharing support
2873   // Construct the path to the archive
2874   char jvm_path[JVM_MAXPATHLEN];
2875   os::jvm_path(jvm_path, sizeof(jvm_path));
2876 #ifdef TIERED
2877   if (strstr(jvm_path, "client") != NULL) {
2878     force_client_mode = true;
2879   }
2880 #endif // TIERED
2881   char *end = strrchr(jvm_path, *os::file_separator());
2882   if (end != NULL) *end = '\0';
2883   char *shared_archive_path = NEW_C_HEAP_ARRAY(char, strlen(jvm_path) +
2884                                                strlen(os::file_separator()) + 20);
2885   if (shared_archive_path == NULL) return JNI_ENOMEM;
2886   strcpy(shared_archive_path, jvm_path);
2887   strcat(shared_archive_path, os::file_separator());
2888   strcat(shared_archive_path, "classes");
2889   DEBUG_ONLY(strcat(shared_archive_path, "_g");)
2890   strcat(shared_archive_path, ".jsa");
2891   SharedArchivePath = shared_archive_path;

2893   // Remaining part of option string
2894   const char* tail;

2896   // If flag "-XX:Flags=flags-file" is used it will be the first option to be pr
2897   bool settings_file_specified = false;
2898   const char* flags_file;
2899   int index;
2900   for (index = 0; index < args->nOptions; index++) {
2901     const JavaVMOption *option = args->options + index;
2902     if (match_option(option, "-XX:Flags=", &tail)) {
2903       flags_file = tail;
2904       settings_file_specified = true;
2905     }
2906     if (match_option(option, "-XX:+PrintVMOptions", &tail)) {
2907       PrintVMOptions = true;
2908     }
2909     if (match_option(option, "-XX:-PrintVMOptions", &tail)) {
2910       PrintVMOptions = false;
2911     }
2912     if (match_option(option, "-XX:+IgnoreUnrecognizedVMOptions", &tail)) {
2913       IgnoreUnrecognizedVMOptions = true;
2914     }
2915     if (match_option(option, "-XX:-IgnoreUnrecognizedVMOptions", &tail)) {
2916       IgnoreUnrecognizedVMOptions = false;
2917     }
2918     if (match_option(option, "-XX:+PrintFlagsInitial", &tail)) {
2919       CommandLineFlags::printFlags();
2920       vm_exit(0);
2921     }

2923 #ifndef PRODUCT
2924     if (match_option(option, "-XX:+PrintFlagsWithComments", &tail)) {
2925       CommandLineFlags::printFlags(true);
2926       vm_exit(0);
2927     }
```



```
2928 #endif
2929   }

2931   if (IgnoreUnrecognizedVMOptions) {
2932     // uncast const to modify the flag args->ignoreUnrecognized
2933     *(jboolean*)(&args->ignoreUnrecognized) = true;
2934   }

2936   // Parse specified settings file
2937   if (settings_file_specified) {
2938     if (!process_settings_file(flags_file, true, args->ignoreUnrecognized)) {
2939       return JNI_EINVAL;
2940     }
2941   }

2943   // Parse default .hotspotrc settings file
2944   if (!settings_file_specified) {
2945     if (!process_settings_file(".hotspotrc", false, args->ignoreUnrecognized)) {
2946       return JNI_EINVAL;
2947     }
2948   }

2950   if (PrintVMOptions) {
2951     for (index = 0; index < args->nOptions; index++) {
2952       const JavaVMOption *option = args->options + index;
2953       if (match_option(option, "-XX:", &tail)) {
2954         logOption(tail);
2955       }
2956     }
2957   }

2959   // Parse JavaVMInitArgs structure passed in, as well as JAVA_TOOL_OPTIONS and
2960   jint result = parse_vm_init_args(args);
2961   if (result != JNI_OK) {
2962     return result;
2963   }

2965 #ifndef PRODUCT
2966   if (TraceBytecodesAt != 0) {
2967     TraceBytecodes = true;
2968   }
2969   if (CountCompiledCalls) {
2970     if (UseCounterDecay) {
2971       warning("UseCounterDecay disabled because CountCalls is set");
2972       UseCounterDecay = false;
2973     }
2974   }
2975 #endif // PRODUCT

2977   // Transitional
2978   if (EnableMethodHandles || AnonymousClasses) {
2979     if (!EnableInvokeDynamic && !FLAG_IS_DEFAULT(EnableInvokeDynamic)) {
2980       warning("EnableMethodHandles and AnonymousClasses are obsolete.  Keeping E
2981     } else {
2982       EnableInvokeDynamic = true;
2983     }
2977   if (EnableInvokeDynamic && !EnableMethodHandles) {
2978     if (!FLAG_IS_DEFAULT(EnableMethodHandles)) {
2979       warning("forcing EnableMethodHandles true because EnableInvokeDynamic is t
2984   }

2986   // JSR 292 is not supported before 1.7
2987   if (!JDK_Version::is_gte_jdk17x_version()) {
2988     if (EnableInvokeDynamic) {
2989       if (!FLAG_IS_DEFAULT(EnableInvokeDynamic)) {
2990         warning("JSR 292 is not supported before 1.7.  Disabling support.");
```

```
2981      EnableMethodHandles = true;
2991        }
2992        EnableInvokeDynamic = false;
2983    if (EnableMethodHandles && !AnonymousClasses) {
2984      if (!FLAG_IS_DEFAULT(AnonymousClasses)) {
2985        warning("forcing AnonymousClasses true because EnableMethodHandles is true
2993      }
2987      AnonymousClasses = true;
2994    }

2996    if (EnableInvokeDynamic && ScavengeRootsInCode == 0) {
2989    if ((EnableMethodHandles || AnonymousClasses) && ScavengeRootsInCode == 0) {
2997      if (!FLAG_IS_DEFAULT(ScavengeRootsInCode)) {
2998        warning("forcing ScavengeRootsInCode non-zero because EnableInvokeDynamic
2991        warning("forcing ScavengeRootsInCode non-zero because EnableMethodHandles
2999      }
3000      ScavengeRootsInCode = 1;
3001    }
3002    if (!JavaObjectsInPerm && ScavengeRootsInCode == 0) {
3003      if (!FLAG_IS_DEFAULT(ScavengeRootsInCode)) {
3004        warning("forcing ScavengeRootsInCode non-zero because JavaObjectsInPerm is
3005      }
3006      ScavengeRootsInCode = 1;
3007    }

3009    if (PrintGCDetails) {
3010      // Turn on -verbose:gc options as well
3011      PrintGC = true;
3012    }

3014    // Set object alignment values.
3015    set_object_alignment();

3017  #ifdef SERIALGC
3018    force_serial_gc();
3019  #endif // SERIALGC
3020  #ifdef KERNEL
3021    no_shared_spaces();
3022  #endif // KERNEL

3024    // Set flags based on ergonomics.
3025    set_ergonomics_flags();

3027    set_shared_spaces_flags();

3029    // Check the GC selections again.
3030    if (!check_gc_consistency()) {
3031      return JNI_EINVAL;
3032    }

3034    if (TieredCompilation) {
3035      set_tiered_flags();
3036    } else {
3037      // Check if the policy is valid. Policies 0 and 1 are valid for non-tiered s
3038      if (CompilationPolicyChoice >= 2) {
3039        vm_exit_during_initialization(
3040          "Incompatible compilation policy selected", NULL);
3041      }
3042    }

3044  #ifndef KERNEL
3045    // Set heap size based on available physical memory
3046    set_heap_size();
3047    // Set per-collector flags
3048    if (UseParallelGC || UseParallelOldGC) {
3049      set_parallel_gc_flags();
```



```
3050    } else if (UseConcMarkSweepGC) { // should be done before ParNew check below
3051      set_cms_and_parnew_gc_flags();
3052    } else if (UseParNewGC) {  // skipped if CMS is set above
3053      set_parnew_gc_flags();
3054    } else if (UseG1GC) {
3055      set_g1_gc_flags();
3056    }
3057  #endif // KERNEL

3059  #ifdef SERIALGC
3060    assert(verify_serial_gc_flags(), "SerialGC unset");
3061  #endif // SERIALGC

3063    // Set bytecode rewriting flags
3064    set_bytecode_flags();

3066    // Set flags if Aggressive optimization flags (-XX:+AggressiveOpts) enabled.
3067    set_aggressive_opts_flags();

3069    // Turn off biased locking for locking debug mode flags,
3070    // which are subtlely different from each other but neither works with
3071    // biased locking.
3072    if (UseHeavyMonitors
3073  #ifdef COMPILER1
3074        || !UseFastLocking
3075  #endif // COMPILER1
3076      ) {
3077      if (!FLAG_IS_DEFAULT(UseBiasedLocking) && UseBiasedLocking) {
3078        // flag set to true on command line; warn the user that they
3079        // can't enable biased locking here
3080        warning("Biased Locking is not supported with locking debug flags"
3081                "; ignoring UseBiasedLocking flag." );
3082      }
3083      UseBiasedLocking = false;
3084    }

3086  #ifdef CC_INTERP
3087    // Clear flags not supported by the C++ interpreter
3088    FLAG_SET_DEFAULT(ProfileInterpreter, false);
3089    FLAG_SET_DEFAULT(UseBiasedLocking, false);
3090    LP64_ONLY(FLAG_SET_DEFAULT(UseCompressedOops, false));
3091  #endif // CC_INTERP

3093  #ifdef COMPILER2
3094    if (!UseBiasedLocking || EmitSync != 0) {
3095      UseOptoBiasInlining = false;
3096    }
3097  #endif

3099    if (PrintAssembly && FLAG_IS_DEFAULT(DebugNonSafepoints)) {
3100      warning("PrintAssembly is enabled; turning on DebugNonSafepoints to gain add
3101      DebugNonSafepoints = true;
3102    }

3104  #ifndef PRODUCT
3105    if (CompileTheWorld) {
3106      // Force NmethodSweeper to sweep whole CodeCache each time.
3107      if (FLAG_IS_DEFAULT(NmethodSweepFraction)) {
3108        NmethodSweepFraction = 1;
3109      }
3110    }
3111  #endif

3113    if (PrintCommandLineFlags) {
3114      CommandLineFlags::printSetFlags();
3115    }
```

```
3117   // Apply CPU specific policy for the BiasedLocking
3118   if (UseBiasedLocking) {
3119     if (!VM_Version::use_biased_locking() &&
3120         !(FLAG_IS_CMDLINE(UseBiasedLocking))) {
3121       UseBiasedLocking = false;
3122     }
3123   }

3125   // set PauseAtExit if the gamma launcher was used and a debugger is attached
3126   // but only if not already set on the commandline
3127   if (Arguments::created_by_gamma_launcher() && os::is_debugger_attached()) {
3128     bool set = false;
3129     CommandLineFlags::wasSetOnCmdline("PauseAtExit", &set);
3130     if (!set) {
3131       FLAG_SET_DEFAULT(PauseAtExit, true);
3132     }
3133   }

3135   return JNI_OK;
3136 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  280674 Wed Mar 30 07:00:31 2011
new/src/share/vm/runtime/globals.hpp
**********************************************************
______unchanged_portion_omitted_

 316 // use this for flags that are true by default in the debug version but
 317 // false in the optimized version, and vice versa
 318 #ifdef ASSERT
 319 #define trueInDebug  true
 320 #define falseInDebug false
 321 #else
 322 #define trueInDebug  false
 323 #define falseInDebug true
 324 #endif

 326 // use this for flags that are true per default in the product build
 327 // but false in development builds, and vice versa
 328 #ifdef PRODUCT
 329 #define trueInProduct  true
 330 #define falseInProduct false
 331 #else
 332 #define trueInProduct  false
 333 #define falseInProduct true
 334 #endif

 336 // use this for flags that are true per default in the tiered build
 337 // but false in non-tiered builds, and vice versa
 338 #ifdef TIERED
 339 #define  trueInTiered true
 340 #define falseInTiered false
 341 #else
 342 #define  trueInTiered false
 343 #define falseInTiered true
 344 #endif

 346 // develop flags are settable / visible only during development and are constant
 347 // product flags are always settable / visible
 348 // notproduct flags are settable / visible only during development and are not d

 350 // A flag must be declared with one of the following types:
 351 // bool, intx, uintx, ccstr.
 352 // The type "ccstr" is an alias for "const char*" and is used
 353 // only in this file, because the macrology requires single-token type names.

 355 // Note: Diagnostic options not meant for VM tuning or for product modes.
 356 // They are to be used for VM quality assurance or field diagnosis
 357 // of VM bugs.  They are hidden so that users will not be encouraged to
 358 // try them as if they were VM ordinary execution options.  However, they
 359 // are available in the product version of the VM.  Under instruction
 360 // from support engineers, VM customers can turn them on to collect
 361 // diagnostic information about VM problems.  To use a VM diagnostic
 362 // option, you must first specify +UnlockDiagnosticVMOptions.
 363 // (This master switch also affects the behavior of -Xprintflags.)
 364 //
 365 // experimental flags are in support of features that are not
 366 //    part of the officially supported product, but are available
 367 //    for experimenting with. They could, for example, be performance
 368 //    features that may not have undergone full or rigorous QA, but which may
 369 //    help performance in some cases and released for experimentation
 370 //    by the community of users and developers. This flag also allows one to
 371 //    be able to build a fully supported product that nonetheless also
 372 //    ships with some unsupported, lightly tested, experimental features.
 373 //    Like the UnlockDiagnosticVMOptions flag above, there is a corresponding
 374 //    UnlockExperimentalVMOptions flag, which allows the control and
 375 //    modification of the experimental flags.
```



```
 376 //
 377 // Nota bene: neither diagnostic nor experimental options should be used casuall
 378 //    and they are not supported on production loads, except under explicit
 379 //    direction from support engineers.
 380 //
 381 // manageable flags are writeable external product flags.
 382 //    They are dynamically writeable through the JDK management interface
 383 //    (com.sun.management.HotSpotDiagnosticMXBean API) and also through JConsole
 384 //    These flags are external exported interface (see CCC).  The list of
 385 //    manageable flags can be queried programmatically through the management
 386 //    interface.
 387 //
 388 //    A flag can be made as "manageable" only if
 389 //    - the flag is defined in a CCC as an external exported interface.
 390 //    - the VM implementation supports dynamic setting of the flag.
 391 //      This implies that the VM must *always* query the flag variable
 392 //      and not reuse state related to the flag state at any given time.
 393 //    - you want the flag to be queried programmatically by the customers.
 394 //
 395 // product_rw flags are writeable internal product flags.
 396 //    They are like "manageable" flags but for internal/private use.
 397 //    The list of product_rw flags are internal/private flags which
 398 //    may be changed/removed in a future release.  It can be set
 399 //    through the management interface to get/set value
 400 //    when the name of flag is supplied.
 401 //
 402 //    A flag can be made as "product_rw" only if
 403 //    - the VM implementation supports dynamic setting of the flag.
 404 //      This implies that the VM must *always* query the flag variable
 405 //      and not reuse state related to the flag state at any given time.
 406 //
 407 // Note that when there is a need to support develop flags to be writeable,
 408 // it can be done in the same way as product_rw.

 410 #define RUNTIME_FLAGS(develop, develop_pd, product, product_pd, diagnostic, expe
 411                                                                             \
 412   lp64_product(bool, UseCompressedOops, false,                              \
 413             "Use 32-bit object references in 64-bit VM. "                   \
 414             "lp64_product means flag is always constant in 32 bit VM")      \
 415                                                                             \
 416   notproduct(bool, CheckCompressedOops, true,                               \
 417             "generate checks in encoding/decoding code in debug VM")        \
 418                                                                             \
 419   product_pd(uintx, HeapBaseMinAddress,                                     \
 420             "OS specific low limit for heap base address")                  \
 421                                                                             \
 422   diagnostic(bool, PrintCompressedOopsMode, false,                          \
 423             "Print compressed oops base address and encoding mode")         \
 424                                                                             \
 425   lp64_product(intx, ObjectAlignmentInBytes, 8,                             \
 426           "Default object alignment in bytes, 8 is minimum")                \
 427                                                                             \
 428   /* UseMembar is theoretically a temp flag used for memory barrier         \
 429    * removal testing.  It was supposed to be removed before FCS but has     \
 430    * been re-added (see 6401008) */                                         \
 431   product_pd(bool, UseMembar,                                               \
 432           "(Unstable) Issues membars on thread state transitions")          \
 433                                                                             \
 434   /* Temp PPC Flag to allow disabling the use of lwsync on ppc platforms    \
 435    * that don't support it.  This will be replaced by processor detection   \
 436    * logic.                                                                 \
 437    */                                                                       \
 438   product(bool, UsePPCLWSYNC, true,                                         \
 439           "Use lwsync instruction if true, else use slower sync")           \
 440                                                                             \
 441   /* Temporary: See 6948537 */                                             \
```

```
  experimental(bool, UseMemSetInBOT, true,                                  \
          "(Unstable) uses memset in BOT updates in GC code")               \
                                                                            \
  diagnostic(bool, UnlockDiagnosticVMOptions, trueInDebug,                  \
          "Enable normal processing of flags relating to field diagnostics")\
                                                                            \
  experimental(bool, UnlockExperimentalVMOptions, false,                    \
          "Enable normal processing of flags relating to experimental features")
                                                                            \
  product(bool, JavaMonitorsInStackTrace, true,                             \
          "Print info. about Java monitor locks when the stacks are dumped")\
                                                                            \
  product_pd(bool, UseLargePages,                                           \
          "Use large page memory")                                          \
                                                                            \
  product_pd(bool, UseLargePagesIndividualAllocation,                       \
          "Allocate large pages individually for better affinity")          \
                                                                            \
  develop(bool, LargePagesIndividualAllocationInjectError, false,           \
          "Fail large pages individual allocation")                         \
                                                                            \
  develop(bool, TracePageSizes, false,                                      \
          "Trace page size selection and usage.")                           \
                                                                            \
  product(bool, UseNUMA, false,                                             \
          "Use NUMA if available")                                          \
                                                                            \
  product(bool, ForceNUMA, false,                                           \
          "Force NUMA optimizations on single-node/UMA systems")            \
                                                                            \
  product(intx, NUMAChunkResizeWeight, 20,                                  \
          "Percentage (0-100) used to weight the current sample when "      \
          "computing exponentially decaying average for "                   \
          "AdaptiveNUMAChunkSizing")                                        \
                                                                            \
  product(intx, NUMASpaceResizeRate, 1*G,                                   \
          "Do not reallocate more that this amount per collection")         \
                                                                            \
  product(bool, UseAdaptiveNUMAChunkSizing, true,                           \
          "Enable adaptive chunk sizing for NUMA")                          \
                                                                            \
  product(bool, NUMAStats, false,                                           \
          "Print NUMA stats in detailed heap information")                  \
                                                                            \
  product(intx, NUMAPageScanRate, 256,                                      \
          "Maximum number of pages to include in the page scan procedure")  \
                                                                            \
  product_pd(bool, NeedsDeoptSuspend,                                       \
          "True for register window machines (sparc/ia64)")                 \
                                                                            \
  product(intx, UseSSE, 99,                                                 \
          "Highest supported SSE instructions set on x86/x64")              \
                                                                            \
  product(uintx, LargePageSizeInBytes, 0,                                   \
          "Large page size (0 to let VM choose the page size")              \
                                                                            \
  product(uintx, LargePageHeapSizeThreshold, 128*M,                         \
          "Use large pages if max heap is at least this big")               \
                                                                            \
  product(bool, ForceTimeHighResolution, false,                             \
          "Using high time resolution(For Win32 only)")                     \
                                                                            \
  develop(bool, TraceItables, false,                                        \
          "Trace initialization and use of itables")                        \
                                                                            \
  develop(bool, TracePcPatching, false,                                     \
```



```
          "Trace usage of frame::patch_pc")                                 \
                                                                            \
  develop(bool, TraceJumps, false,                                          \
          "Trace assembly jumps in thread ring buffer")                     \
                                                                            \
  develop(bool, TraceRelocator, false,                                      \
          "Trace the bytecode relocator")                                   \
                                                                            \
  develop(bool, TraceLongCompiles, false,                                   \
          "Print out every time compilation is longer than "                \
          "a given threashold")                                             \
                                                                            \
  develop(bool, SafepointALot, false,                                       \
          "Generates a lot of safepoints. Works with "                      \
          "GuaranteedSafepointInterval")                                    \
                                                                            \
  product_pd(bool, BackgroundCompilation,                                   \
          "A thread requesting compilation is not blocked during "          \
          "compilation")                                                    \
                                                                            \
  product(bool, PrintVMQWaitTime, false,                                    \
          "Prints out the waiting time in VM operation queue")              \
                                                                            \
  develop(bool, BailoutToInterpreterForThrows, false,                       \
          "Compiled methods which throws/catches exceptions will be "       \
          "deopt and intp.")                                                \
                                                                            \
  develop(bool, NoYieldsInMicrolock, false,                                 \
          "Disable yields in microlock")                                    \
                                                                            \
  develop(bool, TraceOopMapGeneration, false,                               \
          "Shows oopmap generation")                                        \
                                                                            \
  product(bool, MethodFlushing, true,                                       \
          "Reclamation of zombie and not-entrant methods")                  \
                                                                            \
  develop(bool, VerifyStack, false,                                         \
          "Verify stack of each thread when it is entering a runtime call") \
                                                                            \
  develop(bool, ForceUnreachable, false,                                    \
          "(amd64) Make all non code cache addresses to be unreachable with rip-
                                                                            \
  notproduct(bool, StressDerivedPointers, false,                            \
          "Force scavenge when a derived pointers is detected on stack "    \
          "after rtm call")                                                 \
                                                                            \
  develop(bool, TraceDerivedPointers, false,                                \
          "Trace traversal of derived pointers on stack")                   \
                                                                            \
  notproduct(bool, TraceCodeBlobStacks, false,                              \
          "Trace stack-walk of codeblobs")                                  \
                                                                            \
  product(bool, PrintJNIResolving, false,                                   \
          "Used to implement -v:jni")                                       \
                                                                            \
  notproduct(bool, PrintRewrites, false,                                    \
          "Print methods that are being rewritten")                         \
                                                                            \
  product(bool, UseInlineCaches, true,                                      \
          "Use Inline Caches for virtual calls ")                           \
                                                                            \
  develop(bool, InlineArrayCopy, true,                                      \
          "inline arraycopy native that is known to be part of "            \
          "base library DLL")                                               \
                                                                            \
  develop(bool, InlineObjectHash, true,                                     \
```

```
574           "inline Object::hashCode() native that is known to be part "      \
575           "of base library DLL")                                            \
576                                                                             \
577   develop(bool, InlineObjectCopy, true,                                     \
578           "inline Object.clone and Arrays.copyOf[Range] intrinsics")        \
579                                                                             \
580   develop(bool, InlineNatives, true,                                        \
581           "inline natives that are known to be part of base library DLL")   \
582                                                                             \
583   develop(bool, InlineMathNatives, true,                                    \
584           "inline SinD, CosD, etc.")                                        \
585                                                                             \
586   develop(bool, InlineClassNatives, true,                                   \
587           "inline Class.isInstance, etc")                                   \
588                                                                             \
589   develop(bool, InlineAtomicLong, true,                                     \
590           "inline sun.misc.AtomicLong")                                     \
591                                                                             \
592   develop(bool, InlineThreadNatives, true,                                  \
593           "inline Thread.currentThread, etc")                               \
594                                                                             \
595   develop(bool, InlineReflectionGetCallerClass, true,                       \
596           "inline sun.reflect.Reflection.getCallerClass(), known to be part "\
597           "of base library DLL")                                            \
598                                                                             \
599   develop(bool, InlineUnsafeOps, true,                                      \
600           "inline memory ops (native methods) from sun.misc.Unsafe")        \
601                                                                             \
602   develop(bool, ConvertCmpD2CmpF, true,                                     \
603           "Convert cmpD to cmpF when one input is constant in float range") \
604                                                                             \
605   develop(bool, ConvertFloat2IntClipping, true,                             \
606           "Convert float2int clipping idiom to integer clipping")           \
607                                                                             \
608   develop(bool, SpecialStringCompareTo, true,                               \
609           "special version of string compareTo")                            \
610                                                                             \
611   develop(bool, SpecialStringIndexOf, true,                                 \
612           "special version of string indexOf")                              \
613                                                                             \
614   develop(bool, SpecialStringEquals, true,                                  \
615           "special version of string equals")                               \
616                                                                             \
617   develop(bool, SpecialArraysEquals, true,                                  \
618           "special version of Arrays.equals(char[],char[])")                \
619                                                                             \
620   product(bool, UseSSE42Intrinsics, false,                                  \
621           "SSE4.2 versions of intrinsics")                                  \
622                                                                             \
623   develop(bool, TraceCallFixup, false,                                      \
624           "traces all call fixups")                                         \
625                                                                             \
626   develop(bool, DeoptimizeALot, false,                                      \
627           "deoptimize at every exit from the runtime system")               \
628                                                                             \
629   notproduct(ccstrlist, DeoptimizeOnlyAt, "",                               \
630           "a comma separated list of bcis to deoptimize at")                \
631                                                                             \
632   product(bool, DeoptimizeRandom, false,                                    \
633           "deoptimize random frames on random exit from the runtime system")\
634                                                                             \
635   notproduct(bool, ZombieALot, false,                                       \
636           "creates zombies (non-entrant) at exit from the runt. system")    \
637                                                                             \
638   product(bool, UnlinkSymbolsALot, false,                                   \
639           "unlink unreferenced symbols from the symbol table at safepoints")\
```



```
640                                                                             \
641   notproduct(bool, WalkStackALot, false,                                    \
642           "trace stack (no print) at every exit from the runtime system")   \
643                                                                             \
644   develop(bool, Debugging, false,                                           \
645           "set when executing debug methods in debug.ccp "                  \
646           "(to prevent triggering assertions)")                             \
647                                                                             \
648   notproduct(bool, StrictSafepointChecks, trueInDebug,                      \
649           "Enable strict checks that safepoints cannot happen for threads " \
650           "that used No_Safepoint_Verifier")                                \
651                                                                             \
652   notproduct(bool, VerifyLastFrame, false,                                  \
653           "Verify oops on last frame on entry to VM")                       \
654                                                                             \
655   develop(bool, TraceHandleAllocation, false,                               \
656           "Prints out warnings when suspicious many handles are allocated") \
657                                                                             \
658   product(bool, UseCompilerSafepoints, true,                                \
659           "Stop at safepoints in compiled code")                            \
660                                                                             \
661   product(bool, UseSplitVerifier, true,                                     \
662           "use split verifier with StackMapTable attributes")               \
663                                                                             \
664   product(bool, FailOverToOldVerifier, true,                                \
665           "fail over to old verifier when split verifier fails")            \
666                                                                             \
667   develop(bool, ShowSafepointMsgs, false,                                   \
668           "Show msg. about safepoint synch.")                               \
669                                                                             \
670   product(bool, SafepointTimeout, false,                                    \
671           "Time out and warn or fail after SafepointTimeoutDelay "          \
672           "milliseconds if failed to reach safepoint")                      \
673                                                                             \
674   develop(bool, DieOnSafepointTimeout, false,                               \
675           "Die upon failure to reach safepoint (see SafepointTimeout)")     \
676                                                                             \
677   /* 50 retries * (5 * current_retry_count) millis = ~6.375 seconds */      \
678   /* typically, at most a few retries are needed */                         \
679   product(intx, SuspendRetryCount, 50,                                      \
680           "Maximum retry count for an external suspend request")            \
681                                                                             \
682   product(intx, SuspendRetryDelay, 5,                                       \
683           "Milliseconds to delay per retry (* current_retry_count)")        \
684                                                                             \
685   product(bool, AssertOnSuspendWaitFailure, false,                          \
686           "Assert/Guarantee on external suspend wait failure")              \
687                                                                             \
688   product(bool, TraceSuspendWaitFailures, false,                            \
689           "Trace external suspend wait failures")                           \
690                                                                             \
691   product(bool, MaxFDLimit, true,                                           \
692           "Bump the number of file descriptors to max in solaris.")         \
693                                                                             \
694   notproduct(bool, LogEvents, trueInDebug,                                  \
695           "Enable Event log")                                               \
696                                                                             \
697   product(bool, BytecodeVerificationRemote, true,                           \
698           "Enables the Java bytecode verifier for remote classes")          \
699                                                                             \
700   product(bool, BytecodeVerificationLocal, false,                           \
701           "Enables the Java bytecode verifier for local classes")           \
702                                                                             \
703   develop(bool, ForceFloatExceptions, trueInDebug,                          \
704           "Force exceptions on FP stack under/overflow")                    \
705                                                                             \
```

```
706   develop(bool, SoftMatchFailure, trueInProduct,                           \
707           "If the DFA fails to match a node, print a message and bail out") \
708                                                                             \
709   develop(bool, VerifyStackAtCalls, false,                                  \
710           "Verify that the stack pointer is unchanged after calls")         \
711                                                                             \
712   develop(bool, TraceJavaAssertions, false,                                 \
713           "Trace java language assertions")                                 \
714                                                                             \
715   notproduct(bool, CheckAssertionStatusDirectives, false,                   \
716           "temporary - see javaClasses.cpp")                                \
717                                                                             \
718   notproduct(bool, PrintMallocFree, false,                                  \
719           "Trace calls to C heap malloc/free allocation")                   \
720                                                                             \
721   product(bool, PrintOopAddress, false,                                     \
722           "Always print the location of the oop")                           \
723                                                                             \
724   notproduct(bool, VerifyCodeCacheOften, false,                             \
725           "Verify compiled-code cache often")                               \
726                                                                             \
727   develop(bool, ZapDeadCompiledLocals, false,                               \
728           "Zap dead locals in compiler frames")                             \
729                                                                             \
730   notproduct(bool, ZapDeadLocalsOld, false,                                 \
731           "Zap dead locals (old version, zaps all frames when "             \
732           "entering the VM")                                                \
733                                                                             \
734   notproduct(bool, CheckOopishValues, false,                                \
735           "Warn if value contains oop ( requires ZapDeadLocals)")           \
736                                                                             \
737   develop(bool, UseMallocOnly, false,                                       \
738           "use only malloc/free for allocation (no resource area/arena)")   \
739                                                                             \
740   develop(bool, PrintMalloc, false,                                         \
741           "print all malloc/free calls")                                    \
742                                                                             \
743   develop(bool, PrintMallocStatistics, false,                               \
744           "print malloc/free statistics")                                   \
745                                                                             \
746   develop(bool, ZapResourceArea, trueInDebug,                               \
747           "Zap freed resource/arena space with 0xABABABAB")                 \
748                                                                             \
749   notproduct(bool, ZapVMHandleArea, trueInDebug,                            \
750           "Zap freed VM handle space with 0xBCBCBCBC")                      \
751                                                                             \
752   develop(bool, ZapJNIHandleArea, trueInDebug,                              \
753           "Zap freed JNI handle space with 0xFEFEFEFE")                     \
754                                                                             \
755   notproduct(bool, ZapStackSegments, trueInDebug,                           \
756              "Zap allocated/freed Stack segments with 0xFADFADED")          \
757                                                                             \
758   develop(bool, ZapUnusedHeapArea, trueInDebug,                             \
759           "Zap unused heap space with 0xBAADBABE")                          \
760                                                                             \
761   develop(bool, TraceZapUnusedHeapArea, false,                              \
762           "Trace zapping of unused heap space")                             \
763                                                                             \
764   develop(bool, CheckZapUnusedHeapArea, false,                              \
765           "Check zapping of unused heap space")                             \
766                                                                             \
767   develop(bool, ZapFillerObjects, trueInDebug,                              \
768           "Zap filler objects with 0xDEAFBABE")                             \
769                                                                             \
770   develop(bool, PrintVMMessages, true,                                      \
771           "Print vm messages on console")                                   \
```



```
772                                                                             \
773   product(bool, PrintGCApplicationConcurrentTime, false,                    \
774           "Print the time the application has been running")                \
775                                                                             \
776   product(bool, PrintGCApplicationStoppedTime, false,                       \
777           "Print the time the application has been stopped")                \
778                                                                             \
779   notproduct(uintx, ErrorHandlerTest, 0,                                    \
780           "If > 0, provokes an error after VM initialization; the value"    \
781           "determines which error to provoke.  See test_error_handler()"    \
782           "in debug.cpp.")                                                  \
783                                                                             \
784   develop(bool, Verbose, false,                                             \
785           "Prints additional debugging information from other modes")       \
786                                                                             \
787   develop(bool, PrintMiscellaneous, false,                                  \
788           "Prints uncategorized debugging information (requires +Verbose)") \
789                                                                             \
790   develop(bool, WizardMode, false,                                          \
791           "Prints much more debugging information")                         \
792                                                                             \
793   product(bool, ShowMessageBoxOnError, false,                               \
794           "Keep process alive on VM fatal error")                           \
795                                                                             \
796   product(bool, CreateMinidumpOnCrash, false,                               \
797           "Create minidump on VM fatal error")                              \
798                                                                             \
799   product_pd(bool, UseOSErrorReporting,                                     \
800           "Let VM fatal error propagate to the OS (ie. WER on Windows)")    \
801                                                                             \
802   product(bool, SuppressFatalErrorMessage, false,                           \
803           "Do NO Fatal Error report [Avoid deadlock]")                      \
804                                                                             \
805   product(ccstrlist, OnError, "",                                           \
806           "Run user-defined commands on fatal error; see VMError.cpp "      \
807           "for examples")                                                   \
808                                                                             \
809   product(ccstrlist, OnOutOfMemoryError, "",                                \
810           "Run user-defined commands on first java.lang.OutOfMemoryError")  \
811                                                                             \
812   manageable(bool, HeapDumpBeforeFullGC, false,                             \
813           "Dump heap to file before any major stop-world GC")               \
814                                                                             \
815   manageable(bool, HeapDumpAfterFullGC, false,                              \
816           "Dump heap to file after any major stop-world GC")                \
817                                                                             \
818   manageable(bool, HeapDumpOnOutOfMemoryError, false,                       \
819           "Dump heap to file when java.lang.OutOfMemoryError is thrown")    \
820                                                                             \
821   manageable(ccstr, HeapDumpPath, NULL,                                     \
822           "When HeapDumpOnOutOfMemoryError is on, the path (filename or"    \
823           "directory) of the dump file (defaults to java_pid<pid>.hprof"    \
824           "in the working directory)")                                      \
825                                                                             \
826   develop(uintx, SegmentedHeapDumpThreshold, 2*G,                           \
827           "Generate a segmented heap dump (JAVA PROFILE 1.0.2 format) "     \
828           "when the heap usage is larger than this")                        \
829                                                                             \
830   develop(uintx, HeapDumpSegmentSize, 1*G,                                  \
831           "Approximate segment size when generating a segmented heap dump") \
832                                                                             \
833   develop(bool, BreakAtWarning, false,                                      \
834           "Execute breakpoint upon encountering VM warning")                \
835                                                                             \
836   product_pd(bool, UseVectoredExceptions,                                   \
837           "Temp Flag - Use Vectored Exceptions rather than SEH (Windows Only)")
```

```
838                                                                             \
839   develop(bool, TraceVMOperation, false,                                    \
840           "Trace vm operations")                                            \
841                                                                             \
842   develop(bool, UseFakeTimers, false,                                       \
843           "Tells whether the VM should use system time or a fake timer")    \
844                                                                             \
845   diagnostic(bool, LogCompilation, false,                                   \
846           "Log compilation activity in detail to hotspot.log or LogFile")   \
847                                                                             \
848   product(bool, PrintCompilation, false,                                    \
849           "Print compilations")                                             \
850                                                                             \
851   diagnostic(bool, TraceNMethodInstalls, false,                             \
852              "Trace nmethod intallation")                                   \
853                                                                             \
854   diagnostic(intx, ScavengeRootsInCode, 1,                                  \
855              "0: do not allow scavengable oops in the code cache; "         \
856              "1: allow scavenging from the code cache; "                    \
857              "2: emit as many constants as the compiler can see")           \
858                                                                             \
859   diagnostic(bool, TraceOSRBreakpoint, false,                               \
860              "Trace OSR Breakpoint ")                                       \
861                                                                             \
862   diagnostic(bool, TraceCompileTriggered, false,                            \
863              "Trace compile triggered")                                     \
864                                                                             \
865   diagnostic(bool, TraceTriggers, false,                                    \
866              "Trace triggers")                                              \
867                                                                             \
868   product(bool, AlwaysRestoreFPU, false,                                    \
869           "Restore the FPU control word after every JNI call (expensive)")  \
870                                                                             \
871   notproduct(bool, PrintCompilation2, false,                                \
872           "Print additional statistics per compilation")                    \
873                                                                             \
874   diagnostic(bool, PrintAdapterHandlers, false,                             \
875           "Print code generated for i2c/c2i adapters")                      \
876                                                                             \
877   develop(bool, VerifyAdapterSharing, false,                                \
878           "Verify that the code for shared adapters is the equivalent")     \
879                                                                             \
880   diagnostic(bool, PrintAssembly, false,                                    \
881           "Print assembly code (using external disassembler.so)")           \
882                                                                             \
883   diagnostic(ccstr, PrintAssemblyOptions, NULL,                             \
884           "Options string passed to disassembler.so")                       \
885                                                                             \
886   diagnostic(bool, PrintNMethods, false,                                    \
887           "Print assembly code for nmethods when generated")                \
888                                                                             \
889   diagnostic(bool, PrintNativeNMethods, false,                              \
890           "Print assembly code for native nmethods when generated")         \
891                                                                             \
892   develop(bool, PrintDebugInfo, false,                                      \
893           "Print debug information for all nmethods when generated")        \
894                                                                             \
895   develop(bool, PrintRelocations, false,                                    \
896           "Print relocation information for all nmethods when generated")   \
897                                                                             \
898   develop(bool, PrintDependencies, false,                                   \
899           "Print dependency information for all nmethods when generated")   \
900                                                                             \
901   develop(bool, PrintExceptionHandlers, false,                              \
902           "Print exception handler tables for all nmethods when generated") \
903                                                                             \
```



```
904   develop(bool, InterceptOSException, false,                                \
905           "Starts debugger when an implicit OS (e.g., NULL) "               \
906           "exception happens")                                              \
907                                                                             \
908   notproduct(bool, PrintCodeCache, false,                                   \
909           "Print the compiled_code cache when exiting")                     \
910                                                                             \
911   develop(bool, PrintCodeCache2, false,                                     \
912           "Print detailed info on the compiled_code cache when exiting")    \
913                                                                             \
914   diagnostic(bool, PrintStubCode, false,                                    \
915           "Print generated stub code")                                      \
916                                                                             \
917   product(bool, StackTraceInThrowable, true,                                \
918           "Collect backtrace in throwable when exception happens")          \
919                                                                             \
920   product(bool, OmitStackTraceInFastThrow, true,                            \
921           "Omit backtraces for some 'hot' exceptions in optimized code")    \
922                                                                             \
923   product(bool, ProfilerPrintByteCodeStatistics, false,                     \
924           "Prints byte code statictics when dumping profiler output")       \
925                                                                             \
926   product(bool, ProfilerRecordPC, false,                                    \
927           "Collects tick for each 16 byte interval of compiled code")       \
928                                                                             \
929   product(bool, ProfileVM, false,                                           \
930           "Profiles ticks that fall within VM (either in the VM Thread "    \
931           "or VM code called through stubs)")                               \
932                                                                             \
933   product(bool, ProfileIntervals, false,                                    \
934           "Prints profiles for each interval (see ProfileIntervalsTicks)")  \
935                                                                             \
936   notproduct(bool, ProfilerCheckIntervals, false,                           \
937           "Collect and print info on spacing of profiler ticks")            \
938                                                                             \
939   develop(bool, PrintJVMWarnings, false,                                    \
940           "Prints warnings for unimplemented JVM functions")                \
941                                                                             \
942   product(bool, PrintWarnings, true,                                        \
943           "Prints JVM warnings to output stream")                           \
944                                                                             \
945   notproduct(uintx, WarnOnStalledSpinLock, 0,                               \
946           "Prints warnings for stalled SpinLocks")                          \
947                                                                             \
948   develop(bool, InitializeJavaLangSystem, true,                             \
949           "Initialize java.lang.System - turn off for individual "          \
950           "method debugging")                                               \
951                                                                             \
952   develop(bool, InitializeJavaLangString, true,                             \
953           "Initialize java.lang.String - turn off for individual "          \
954           "method debugging")                                               \
955                                                                             \
956   develop(bool, InitializeJavaLangExceptionsErrors, true,                   \
957           "Initialize various error and exception classes - turn off for "  \
958           "individual method debugging")                                    \
959                                                                             \
960   product(bool, RegisterFinalizersAtInit, true,                             \
961           "Register finalizable objects at end of Object.<init> or "        \
962           "after allocation")                                               \
963                                                                             \
964   develop(bool, RegisterReferences, true,                                   \
965           "Tells whether the VM should register soft/weak/final/phantom "   \
966           "references")                                                     \
967                                                                             \
968   develop(bool, IgnoreRewrites, false,                                      \
969           "Supress rewrites of bytecodes in the oopmap generator. "         \
```

```
 970           "This is unsafe!")                                          \
 971                                                                       \
 972   develop(bool, PrintCodeCacheExtension, false,                       \
 973           "Print extension of code cache")                            \
 974                                                                       \
 975   develop(bool, UsePrivilegedStack, true,                             \
 976           "Enable the security JVM functions")                        \
 977                                                                       \
 978   develop(bool, IEEEPrecision, true,                                  \
 979           "Enables IEEE precision (for INTEL only)")                  \
 980                                                                       \
 981   develop(bool, ProtectionDomainVerification, true,                   \
 982           "Verifies protection domain before resolution in system "   \
 983           "dictionary")                                               \
 984                                                                       \
 985   product(bool, ClassUnloading, true,                                 \
 986           "Do unloading of classes")                                  \
 987                                                                       \
 988   diagnostic(bool, LinkWellKnownClasses, false,                       \
 989           "Resolve a well known class as soon as its name is seen")   \
 990                                                                       \
 991   develop(bool, DisableStartThread, false,                            \
 992           "Disable starting of additional Java threads "              \
 993           "(for debugging only)")                                     \
 994                                                                       \
 995   develop(bool, MemProfiling, false,                                  \
 996           "Write memory usage profiling to log file")                 \
 997                                                                       \
 998   notproduct(bool, PrintSystemDictionaryAtExit, false,                \
 999           "Prints the system dictionary at exit")                     \
1000                                                                       \
1001   diagnostic(bool, UnsyncloadClass, false,                            \
1002           "Unstable: VM calls loadClass unsynchronized. Custom "      \
1003           "class loader  must call VM synchronized for findClass "    \
1004           "and defineClass.")                                         \
1005                                                                       \
1006   product(bool, AlwaysLockClassLoader, false,                         \
1007           "Require the VM to acquire the class loader lock before calling " \
1008           "loadClass() even for class loaders registering "           \
1009           "as parallel capable")                                      \
1010                                                                       \
1011   product(bool, AllowParallelDefineClass, false,                      \
1012           "Allow parallel defineClass requests for class loaders "    \
1013           "registering as parallel capable")                          \
1014                                                                       \
1015   product(bool, MustCallLoadClassInternal, false,                     \
1016           "Call loadClassInternal() rather than loadClass()")         \
1017                                                                       \
1018   product_pd(bool, DontYieldALot,                                     \
1019           "Throw away obvious excess yield calls (for SOLARIS only)") \
1020                                                                       \
1021   product_pd(bool, ConvertSleepToYield,                               \
1022           "Converts sleep(0) to thread yield "                        \
1023           "(may be off for SOLARIS to improve GUI)")                  \
1024                                                                       \
1025   product(bool, ConvertYieldToSleep, false,                           \
1026           "Converts yield to a sleep of MinSleepInterval to simulate Win32 " \
1027           "behavior (SOLARIS only)")                                  \
1028                                                                       \
1029   product(bool, UseBoundThreads, true,                                \
1030           "Bind user level threads to kernel threads (for SOLARIS only)") \
1031                                                                       \
1032   develop(bool, UseDetachedThreads, true,                             \
1033           "Use detached threads that are recycled upon termination "  \
1034           "(for SOLARIS only)")                                       \
1035                                                                       \
```



```
1036   product(bool, UseLWPSynchronization, true,                          \
1037           "Use LWP-based instead of libthread-based synchronization "  \
1038           "(SPARC only)")                                             \
1039                                                                       \
1040   product(ccstr, SyncKnobs, NULL,                                     \
1041           "(Unstable) Various monitor synchronization tunables")      \
1042                                                                       \
1043   product(intx, EmitSync, 0,                                          \
1044           "(Unsafe,Unstable) "                                        \
1045           " Controls emission of inline sync fast-path code")         \
1046                                                                       \
1047   product(intx, AlwaysInflate, 0, "(Unstable) Force inflation")       \
1048                                                                       \
1049   product(intx, MonitorBound, 0, "Bound Monitor population")          \
1050                                                                       \
1051   product(bool, MonitorInUseLists, false, "Track Monitors for Deflation") \
1052                                                                       \
1053   product(intx, Atomics, 0,                                           \
1054           "(Unsafe,Unstable) Diagnostic - Controls emission of atomics") \
1055                                                                       \
1056   product(intx, FenceInstruction, 0,                                  \
1057           "(Unsafe,Unstable) Experimental")                           \
1058                                                                       \
1059   product(intx, SyncFlags, 0, "(Unsafe,Unstable) Experimental Sync flags" ) \
1060                                                                       \
1061   product(intx, SyncVerbose, 0, "(Unstable)" )                        \
1062                                                                       \
1063   product(intx, ClearFPUAtPark, 0, "(Unsafe,Unstable)" )              \
1064                                                                       \
1065   product(intx, hashCode, 0,                                          \
1066          "(Unstable) select hashCode generation algorithm" )          \
1067                                                                       \
1068   product(intx, WorkAroundNPTLTimedWaitHang, 1,                       \
1069          "(Unstable, Linux-specific)"                                 \
1070          " avoid NPTL-FUTEX hang pthread_cond_timedwait" )            \
1071                                                                       \
1072   product(bool, FilterSpuriousWakeups, true,                          \
1073           "Prevent spurious or premature wakeups from object.wait "   \
1074           "(Solaris only)")                                           \
1075                                                                       \
1076   product(intx, NativeMonitorTimeout, -1, "(Unstable)" )              \
1077   product(intx, NativeMonitorFlags, 0, "(Unstable)" )                 \
1078   product(intx, NativeMonitorSpinLimit, 20, "(Unstable)" )            \
1079                                                                       \
1080   develop(bool, UsePthreads, false,                                   \
1081           "Use pthread-based instead of libthread-based synchronization " \
1082           "(SPARC only)")                                             \
1083                                                                       \
1084   product(bool, AdjustConcurrency, false,                             \
1085           "call thr_setconcurrency at thread create time to avoid "   \
1086           "LWP starvation on MP systems (For Solaris Only)")          \
1087                                                                       \
1088   develop(bool, UpdateHotSpotCompilerFileOnError, true,               \
1089           "Should the system attempt to update the compiler file when " \
1090           "an error occurs?")                                         \
1091                                                                       \
1092   product(bool, ReduceSignalUsage, false,                             \
1093           "Reduce the use of OS signals in Java and/or the VM")       \
1094                                                                       \
1095   notproduct(bool, ValidateMarkSweep, false,                          \
1096           "Do extra validation during MarkSweep collection")          \
1097                                                                       \
1098   notproduct(bool, RecordMarkSweepCompaction, false,                  \
1099           "Enable GC-to-GC recording and querying of compaction during " \
1100           "MarkSweep")                                                \
1101                                                                       \
```

```
1102   develop_pd(bool, ShareVtableStubs,                                       \
1103           "Share vtable stubs (smaller code but worse branch prediction")   \
1104                                                                             \
1105   develop(bool, LoadLineNumberTables, true,                                 \
1106           "Tells whether the class file parser loads line number tables")   \
1107                                                                             \
1108   develop(bool, LoadLocalVariableTables, true,                              \
1109           "Tells whether the class file parser loads local variable tables")\
1110                                                                             \
1111   develop(bool, LoadLocalVariableTypeTables, true,                          \
1112           "Tells whether the class file parser loads local variable type tables"
1113                                                                             \
1114   product(bool, AllowUserSignalHandlers, false,                             \
1115           "Do not complain if the application installs signal handlers "    \
1116           "(Solaris & Linux only)")                                         \
1117                                                                             \
1118   product(bool, UseSignalChaining, true,                                    \
1119           "Use signal-chaining to invoke signal handlers installed "        \
1120           "by the application (Solaris & Linux only)")                      \
1121                                                                             \
1122   product(bool, UseAltSigs, false,                                          \
1123           "Use alternate signals instead of SIGUSR1 & SIGUSR2 for VM "      \
1124           "internal signals (Solaris only)")                                \
1125                                                                             \
1126   product(bool, UseSpinning, false,                                         \
1127           "Use spinning in monitor inflation and before entry")             \
1128                                                                             \
1129   product(bool, PreSpinYield, false,                                        \
1130           "Yield before inner spinning loop")                               \
1131                                                                             \
1132   product(bool, PostSpinYield, true,                                        \
1133           "Yield after inner spinning loop")                                \
1134                                                                             \
1135   product(bool, AllowJNIEnvProxy, false,                                    \
1136           "Allow JNIEnv proxies for jdbx")                                  \
1137                                                                             \
1138   product(bool, JNIDetachReleasesMonitors, true,                            \
1139           "JNI DetachCurrentThread releases monitors owned by thread")      \
1140                                                                             \
1141   product(bool, RestoreMXCSROnJNICalls, false,                              \
1142           "Restore MXCSR when returning from JNI calls")                    \
1143                                                                             \
1144   product(bool, CheckJNICalls, false,                                       \
1145           "Verify all arguments to JNI calls")                              \
1146                                                                             \
1147   product(bool, UseFastJNIAccessors, true,                                  \
1148           "Use optimized versions of Get<Primitive>Field")                  \
1149                                                                             \
1150   product(bool, EagerXrunInit, false,                                       \
1151           "Eagerly initialize -Xrun libraries; allows startup profiling, "  \
1152           " but not all -Xrun libraries may support the state of the VM at this
1153                                                                             \
1154   product(bool, PreserveAllAnnotations, false,                              \
1155           "Preserve RuntimeInvisibleAnnotations as well as RuntimeVisibleAnnotat
1156                                                                             \
1157   develop(uintx, PreallocatedOutOfMemoryErrorCount, 4,                      \
1158           "Number of OutOfMemoryErrors preallocated with backtrace")        \
1159                                                                             \
1160   product(bool, LazyBootClassLoader, true,                                  \
1161           "Enable/disable lazy opening of boot class path entries")         \
1162                                                                             \
1163   diagnostic(bool, UseIncDec, true,                                         \
1164           "Use INC, DEC instructions on x86")                               \
1165                                                                             \
1166   product(bool, UseNewLongLShift, false,                                    \
1167           "Use optimized bitwise shift left")                               \
```



```
1168                                                                             \
1169   product(bool, UseStoreImmI16, true,                                       \
1170           "Use store immediate 16-bits value instruction on x86")           \
1171                                                                             \
1172   product(bool, UseAddressNop, false,                                       \
1173           "Use '0F 1F [addr]' NOP instructions on x86 cpus")                \
1174                                                                             \
1175   product(bool, UseXmmLoadAndClearUpper, true,                              \
1176           "Load low part of XMM register and clear upper part")             \
1177                                                                             \
1178   product(bool, UseXmmRegToRegMoveAll, false,                               \
1179           "Copy all XMM register bits when moving value between registers") \
1180                                                                             \
1181   product(bool, UseXmmI2D, false,                                           \
1182           "Use SSE2 CVTDQ2PD instruction to convert Integer to Double")     \
1183                                                                             \
1184   product(bool, UseXmmI2F, false,                                           \
1185           "Use SSE2 CVTDQ2PS instruction to convert Integer to Float")      \
1186                                                                             \
1187   product(bool, UseXMMForArrayCopy, false,                                  \
1188           "Use SSE2 MOVQ instruction for Arraycopy")                        \
1189                                                                             \
1190   product(bool, UseUnalignedLoadStores, false,                              \
1191           "Use SSE2 MOVDQU instruction for Arraycopy")                      \
1192                                                                             \
1193   product(intx, FieldsAllocationStyle, 1,                                   \
1194           "0 - type based with oops first, 1 - with oops last, "            \
1195           "2 - oops in super and sub classes are together")                 \
1196                                                                             \
1197   product(bool, CompactFields, true,                                        \
1198           "Allocate nonstatic fields in gaps between previous fields")      \
1199                                                                             \
1200   notproduct(bool, PrintCompactFieldsSavings, false,                        \
1201           "Print how many words were saved with CompactFields")             \
1202                                                                             \
1203   product(bool, UseBiasedLocking, true,                                     \
1204           "Enable biased locking in JVM")                                   \
1205                                                                             \
1206   product(intx, BiasedLockingStartupDelay, 4000,                            \
1207           "Number of milliseconds to wait before enabling biased locking")  \
1208                                                                             \
1209   diagnostic(bool, PrintBiasedLockingStatistics, false,                     \
1210           "Print statistics of biased locking in JVM")                      \
1211                                                                             \
1212   product(intx, BiasedLockingBulkRebiasThreshold, 20,                       \
1213           "Threshold of number of revocations per type to try to "          \
1214           "rebias all objects in the heap of that type")                    \
1215                                                                             \
1216   product(intx, BiasedLockingBulkRevokeThreshold, 40,                       \
1217           "Threshold of number of revocations per type to permanently "     \
1218           "revoke biases of all objects in the heap of that type")          \
1219                                                                             \
1220   product(intx, BiasedLockingDecayTime, 25000,                              \
1221           "Decay time (in milliseconds) to re-enable bulk rebiasing of a "  \
1222           "type after previous bulk rebias")                                \
1223                                                                             \
1224   develop(bool, JavaObjectsInPerm, false,                                   \
1225           "controls whether Classes and interned Strings are allocated"     \
1226           "in perm.  This purely intended to allow debugging issues"        \
1227           "in production.")                                                 \
1228                                                                             \
1229   /* tracing */                                                             \
1230                                                                             \
1231   notproduct(bool, TraceRuntimeCalls, false,                                \
1232           "Trace run-time calls")                                           \
1233                                                                             \
```

```
1234   develop(bool, TraceJNICalls, false,                                       \
1235           "Trace JNI calls")                                                \
1236                                                                             \
1237   notproduct(bool, TraceJVMCalls, false,                                    \
1238           "Trace JVM calls")                                                \
1239                                                                             \
1240   product(ccstr, TraceJVMTI, NULL,                                          \
1241           "Trace flags for JVMTI functions and events")                     \
1242                                                                             \
1243   /* This option can change an EMCP method into an obsolete method. */     \
1244   /* This can affect tests that except specific methods to be EMCP. */     \
1245   /* This option should be used with caution. */                           \
1246   product(bool, StressLdcRewrite, false,                                    \
1247           "Force ldc -> ldc_w rewrite during RedefineClasses")              \
1248                                                                             \
1249   product(intx, TraceRedefineClasses, 0,                                    \
1250           "Trace level for JVMTI RedefineClasses")                          \
1251                                                                             \
1252   develop(bool, StressMethodComparator, false,                              \
1253           "run the MethodComparator on all loaded methods")                 \
1254                                                                             \
1255   /* change to false by default sometime after Mustang */                  \
1256   product(bool, VerifyMergedCPBytecodes, true,                              \
1257           "Verify bytecodes after RedefineClasses constant pool merging")   \
1258                                                                             \
1259   develop(bool, TraceJNIHandleAllocation, false,                            \
1260           "Trace allocation/deallocation of JNI handle blocks")             \
1261                                                                             \
1262   develop(bool, TraceThreadEvents, false,                                   \
1263           "Trace all thread events")                                        \
1264                                                                             \
1265   develop(bool, TraceBytecodes, false,                                      \
1266           "Trace bytecode execution")                                       \
1267                                                                             \
1268   develop(bool, TraceClassInitialization, false,                            \
1269           "Trace class initialization")                                     \
1270                                                                             \
1271   develop(bool, TraceExceptions, false,                                     \
1272           "Trace exceptions")                                               \
1273                                                                             \
1274   develop(bool, TraceICs, false,                                            \
1275           "Trace inline cache changes")                                     \
1276                                                                             \
1277   notproduct(bool, TraceInvocationCounterOverflow, false,                   \
1278           "Trace method invocation counter overflow")                       \
1279                                                                             \
1280   develop(bool, TraceInlineCacheClearing, false,                            \
1281           "Trace clearing of inline caches in nmethods")                    \
1282                                                                             \
1283   develop(bool, TraceDependencies, false,                                   \
1284           "Trace dependencies")                                             \
1285                                                                             \
1286   develop(bool, VerifyDependencies, trueInDebug,                            \
1287          "Exercise and verify the compilation dependency mechanism")        \
1288                                                                             \
1289   develop(bool, TraceNewOopMapGeneration, false,                            \
1290           "Trace OopMapGeneration")                                         \
1291                                                                             \
1292   develop(bool, TraceNewOopMapGenerationDetailed, false,                    \
1293           "Trace OopMapGeneration: print detailed cell states")             \
1294                                                                             \
1295   develop(bool, TimeOopMap, false,                                          \
1296           "Time calls to GenerateOopMap::compute_map() in sum")             \
1297                                                                             \
1298   develop(bool, TimeOopMap2, false,                                         \
1299           "Time calls to GenerateOopMap::compute_map() individually")       \
```



```
1300                                                                             \
1301   develop(bool, TraceMonitorMismatch, false,                                \
1302           "Trace monitor matching failures during OopMapGeneration")        \
1303                                                                             \
1304   develop(bool, TraceOopMapRewrites, false,                                 \
1305           "Trace rewritting of method oops during oop map generation")      \
1306                                                                             \
1307   develop(bool, TraceSafepoint, false,                                      \
1308           "Trace safepoint operations")                                     \
1309                                                                             \
1310   develop(bool, TraceICBuffer, false,                                       \
1311           "Trace usage of IC buffer")                                       \
1312                                                                             \
1313   develop(bool, TraceCompiledIC, false,                                     \
1314           "Trace changes of compiled IC")                                   \
1315                                                                             \
1316   notproduct(bool, TraceZapDeadLocals, false,                               \
1317           "Trace zapping dead locals")                                      \
1318                                                                             \
1319   develop(bool, TraceStartupTime, false,                                    \
1320           "Trace setup time")                                               \
1321                                                                             \
1322   product(ccstr, HPILibPath, NULL,                                          \
1323           "Specify alternate path to HPI library")                          \
1324                                                                             \
1325   develop(bool, TraceProtectionDomainVerification, false,                   \
1326           "Trace protection domain verifcation")                            \
1327                                                                             \
1328   develop(bool, TraceClearedExceptions, false,                              \
1329           "Prints when an exception is forcibly cleared")                   \
1330                                                                             \
1331   product(bool, TraceClassResolution, false,                                \
1332           "Trace all constant pool resolutions (for debugging)")            \
1333                                                                             \
1334   product(bool, TraceBiasedLocking, false,                                  \
1335           "Trace biased locking in JVM")                                    \
1336                                                                             \
1337   product(bool, TraceMonitorInflation, false,                               \
1338           "Trace monitor inflation in JVM")                                 \
1339                                                                             \
1340   /* assembler */                                                           \
1341   product(bool, Use486InstrsOnly, false,                                    \
1342           "Use 80486 Compliant instruction subset")                         \
1343                                                                             \
1344   /* gc */                                                                  \
1345                                                                             \
1346   product(bool, UseSerialGC, false,                                         \
1347           "Use the serial garbage collector")                               \
1348                                                                             \
1349   product(bool, UseG1GC, false,                                             \
1350           "Use the Garbage-First garbage collector")                        \
1351                                                                             \
1352   product(bool, UseParallelGC, false,                                       \
1353           "Use the Parallel Scavenge garbage collector")                    \
1354                                                                             \
1355   product(bool, UseParallelOldGC, false,                                    \
1356           "Use the Parallel Old garbage collector")                         \
1357                                                                             \
1358   product(bool, UseParallelOldGCCompacting, true,                           \
1359           "In the Parallel Old garbage collector use parallel compaction")  \
1360                                                                             \
1361   product(bool, UseParallelDensePrefixUpdate, true,                         \
1362           "In the Parallel Old garbage collector use parallel dense"        \
1363           " prefix update")                                                 \
1364                                                                             \
1365   product(uintx, HeapMaximumCompactionInterval, 20,                         \
```

```
1366           "How often should we maximally compact the heap (not allowing "   \
1367           "any dead space)")                                                \
1368                                                                             \
1369   product(uintx, HeapFirstMaximumCompactionCount, 3,                        \
1370           "The collection count for the first maximum compaction")          \
1371                                                                             \
1372   product(bool, UseMaximumCompactionOnSystemGC, true,                       \
1373           "In the Parallel Old garbage collector maximum compaction for "   \
1374           "a system GC")                                                    \
1375                                                                             \
1376   product(uintx, ParallelOldDeadWoodLimiterMean, 50,                        \
1377           "The mean used by the par compact dead wood"                      \
1378           "limiter (a number between 0-100).")                              \
1379                                                                             \
1380   product(uintx, ParallelOldDeadWoodLimiterStdDev, 80,                      \
1381           "The standard deviation used by the par compact dead wood"        \
1382           "limiter (a number between 0-100).")                              \
1383                                                                             \
1384   product(bool, UseParallelOldGCDensePrefix, true,                          \
1385           "Use a dense prefix with the Parallel Old garbage collector")     \
1386                                                                             \
1387   product(uintx, ParallelGCThreads, 0,                                      \
1388           "Number of parallel threads parallel gc will use")                \
1389                                                                             \
1390   develop(bool, ParallelOldGCSplitALot, false,                              \
1391           "Provoke splitting (copying data from a young gen space to"       \
1392           "multiple destination spaces)")                                   \
1393                                                                             \
1394   develop(uintx, ParallelOldGCSplitInterval, 3,                             \
1395           "How often to provoke splitting a young gen space")               \
1396                                                                             \
1397   develop(bool, TraceRegionTasksQueuing, false,                             \
1398           "Trace the queuing of the region tasks")                          \
1399                                                                             \
1400   product(uintx, ConcGCThreads, 0,                                          \
1401           "Number of threads concurrent gc will use")                       \
1402                                                                             \
1403   product(uintx, YoungPLABSize, 4096,                                       \
1404           "Size of young gen promotion labs (in HeapWords)")                \
1405                                                                             \
1406   product(uintx, OldPLABSize, 1024,                                         \
1407           "Size of old gen promotion labs (in HeapWords)")                  \
1408                                                                             \
1409   product(uintx, GCTaskTimeStampEntries, 200,                               \
1410           "Number of time stamp entries per gc worker thread")              \
1411                                                                             \
1412   product(bool, AlwaysTenure, false,                                        \
1413           "Always tenure objects in eden. (ParallelGC only)")               \
1414                                                                             \
1415   product(bool, NeverTenure, false,                                         \
1416           "Never tenure objects in eden, May tenure on overflow "           \
1417           "(ParallelGC only)")                                              \
1418                                                                             \
1419   product(bool, ScavengeBeforeFullGC, true,                                 \
1420           "Scavenge youngest generation before each full GC, "              \
1421           "used with UseParallelGC")                                        \
1422                                                                             \
1423   develop(bool, ScavengeWithObjectsInToSpace, false,                        \
1424           "Allow scavenges to occur when to_space contains objects.")       \
1425                                                                             \
1426   product(bool, UseConcMarkSweepGC, false,                                  \
1427           "Use Concurrent Mark-Sweep GC in the old generation")             \
1428                                                                             \
1429   product(bool, ExplicitGCInvokesConcurrent, false,                         \
1430           "A System.gc() request invokes a concurrent collection;"          \
1431           " (effective only when UseConcMarkSweepGC)")                      \
```



```
1432                                                                             \
1433   product(bool, ExplicitGCInvokesConcurrentAndUnloadsClasses, false,        \
1434           "A System.gc() request invokes a concurrent collection and "      \
1435           "also unloads classes during such a concurrent gc cycle "         \
1436           "(effective only when UseConcMarkSweepGC)")                       \
1437                                                                             \
1438   product(bool, GCLockerInvokesConcurrent, false,                           \
1439           "The exit of a JNI CS necessitating a scavenge also"              \
1440           " kicks off a bkgrd concurrent collection")                       \
1441                                                                             \
1442   product(uintx, GCLockerEdenExpansionPercent, 5,                           \
1443           "How much the GC can expand the eden by while the GC locker  "    \
1444           "is active (as a percentage)")                                    \
1445                                                                             \
1446   develop(bool, UseCMSAdaptiveFreeLists, true,                              \
1447           "Use Adaptive Free Lists in the CMS generation")                  \
1448                                                                             \
1449   develop(bool, UseAsyncConcMarkSweepGC, true,                              \
1450           "Use Asynchronous Concurrent Mark-Sweep GC in the old generation")\
1451                                                                             \
1452   develop(bool, RotateCMSCollectionTypes, false,                            \
1453           "Rotate the CMS collections among concurrent and STW")            \
1454                                                                             \
1455   product(bool, UseCMSBestFit, true,                                        \
1456           "Use CMS best fit allocation strategy")                           \
1457                                                                             \
1458   product(bool, UseCMSCollectionPassing, true,                              \
1459           "Use passing of collection from background to foreground")        \
1460                                                                             \
1461   product(bool, UseParNewGC, false,                                         \
1462           "Use parallel threads in the new generation.")                    \
1463                                                                             \
1464   product(bool, ParallelGCVerbose, false,                                   \
1465           "Verbose output for parallel GC.")                                \
1466                                                                             \
1467   product(intx, ParallelGCBufferWastePct, 10,                               \
1468           "wasted fraction of parallel allocation buffer.")                 \
1469                                                                             \
1470   product(bool, ParallelGCRetainPLAB, true,                                 \
1471           "Retain parallel allocation buffers across scavenges.")           \
1472                                                                             \
1473   product(intx, TargetPLABWastePct, 10,                                     \
1474           "target wasted space in last buffer as pct of overall allocation")\
1475                                                                             \
1476   product(uintx, PLABWeight, 75,                                            \
1477           "Percentage (0-100) used to weight the current sample when"       \
1478           "computing exponentially decaying average for ResizePLAB.")       \
1479                                                                             \
1480   product(bool, ResizePLAB, true,                                           \
1481           "Dynamically resize (survivor space) promotion labs")             \
1482                                                                             \
1483   product(bool, PrintPLAB, false,                                           \
1484           "Print (survivor space) promotion labs sizing decisions")         \
1485                                                                             \
1486   product(intx, ParGCArrayScanChunk, 50,                                    \
1487           "Scan a subset and push remainder, if array is bigger than this") \
1488                                                                             \
1489   product(bool, ParGCUseLocalOverflow, false,                               \
1490           "Instead of a global overflow list, use local overflow stacks")   \
1491                                                                             \
1492   product(bool, ParGCTrimOverflow, true,                                    \
1493           "Eagerly trim the local overflow lists (when ParGCUseLocalOverflow") \
1494                                                                             \
1495   notproduct(bool, ParGCWorkQueueOverflowALot, false,                       \
1496           "Whether we should simulate work queue overflow in ParNew")       \
1497                                                                             \
```

```
1498   notproduct(uintx, ParGCWorkQueueOverflowInterval, 1000,                   \
1499           "An 'interval' counter that determines how frequently "           \
1500           "we simulate overflow; a smaller number increases frequency")     \
1501                                                                             \
1502   product(uintx, ParGCDesiredObjsFromOverflowList, 20,                      \
1503           "The desired number of objects to claim from the overflow list")  \
1504                                                                             \
1505   product(uintx, CMSParPromoteBlocksToClaim, 16,                             \
1506           "Number of blocks to attempt to claim when refilling CMS LAB for "\
1507           "parallel GC.")                                                   \
1508                                                                             \
1509   product(uintx, OldPLABWeight, 50,                                         \
1510           "Percentage (0-100) used to weight the current sample when"       \
1511           "computing exponentially decaying average for resizing CMSParPromoteBl
1512                                                                             \
1513   product(bool, ResizeOldPLAB, true,                                        \
1514           "Dynamically resize (old gen) promotion labs")                    \
1515                                                                             \
1516   product(bool, PrintOldPLAB, false,                                        \
1517           "Print (old gen) promotion labs sizing decisions")                \
1518                                                                             \
1519   product(uintx, CMSOldPLABMin, 16,                                         \
1520           "Min size of CMS gen promotion lab caches per worker per blksize")\
1521                                                                             \
1522   product(uintx, CMSOldPLABMax, 1024,                                       \
1523           "Max size of CMS gen promotion lab caches per worker per blksize")\
1524                                                                             \
1525   product(uintx, CMSOldPLABNumRefills, 4,                                   \
1526           "Nominal number of refills of CMS gen promotion lab cache"        \
1527           " per worker per block size")                                     \
1528                                                                             \
1529   product(bool, CMSOldPLABResizeQuicker, false,                             \
1530           "Whether to react on-the-fly during a scavenge to a sudden"       \
1531           " change in block demand rate")                                   \
1532                                                                             \
1533   product(uintx, CMSOldPLABToleranceFactor, 4,                              \
1534           "The tolerance of the phase-change detector for on-the-fly"       \
1535           " PLAB resizing during a scavenge")                               \
1536                                                                             \
1537   product(uintx, CMSOldPLABReactivityFactor, 2,                             \
1538           "The gain in the feedback loop for on-the-fly PLAB resizing"      \
1539           " during a scavenge")                                             \
1540                                                                             \
1541   product(uintx, CMSOldPLABReactivityCeiling, 10,                           \
1542           "The clamping of the gain in the feedback loop for on-the-fly"    \
1543           " PLAB resizing during a scavenge")                               \
1544                                                                             \
1545   product(bool, AlwaysPreTouch, false,                                      \
1546           "It forces all freshly committed pages to be pre-touched.")       \
1547                                                                             \
1548   product_pd(intx, CMSYoungGenPerWorker,                                    \
1549           "The maximum size of young gen chosen by default per GC worker "  \
1550           "thread available")                                               \
1551                                                                             \
1552   product(bool, GCOverheadReporting, false,                                 \
1553          "Enables the GC overhead reporting facility")                      \
1554                                                                             \
1555   product(intx, GCOverheadReportingPeriodMS, 100,                           \
1556           "Reporting period for conc GC overhead reporting, in ms ")        \
1557                                                                             \
1558   product(bool, CMSIncrementalMode, false,                                  \
1559           "Whether CMS GC should operate in \"incremental\" mode")          \
1560                                                                             \
1561   product(uintx, CMSIncrementalDutyCycle, 10,                               \
1562           "CMS incremental mode duty cycle (a percentage, 0-100).  If"      \
1563           "CMSIncrementalPacing is enabled, then this is just the initial"  \
```



```
1564           "value")                                                         \
1565                                                                            \
1566   product(bool, CMSIncrementalPacing, true,                                \
1567           "Whether the CMS incremental mode duty cycle should be "         \
1568           "automatically adjusted")                                        \
1569                                                                            \
1570   product(uintx, CMSIncrementalDutyCycleMin, 0,                            \
1571           "Lower bound on the duty cycle when CMSIncrementalPacing is "    \
1572           "enabled (a percentage, 0-100)")                                 \
1573                                                                            \
1574   product(uintx, CMSIncrementalSafetyFactor, 10,                           \
1575           "Percentage (0-100) used to add conservatism when computing the " \
1576           "duty cycle")                                                    \
1577                                                                            \
1578   product(uintx, CMSIncrementalOffset, 0,                                  \
1579           "Percentage (0-100) by which the CMS incremental mode duty cycle" \
1580           " is shifted to the right within the period between young GCs")  \
1581                                                                            \
1582   product(uintx, CMSExpAvgFactor, 50,                                      \
1583           "Percentage (0-100) used to weight the current sample when"      \
1584           "computing exponential averages for CMS statistics.")            \
1585                                                                            \
1586   product(uintx, CMS_FLSWeight, 75,                                        \
1587           "Percentage (0-100) used to weight the current sample when"      \
1588           "computing exponentially decating averages for CMS FLS statistics.") \
1589                                                                            \
1590   product(uintx, CMS_FLSPadding, 1,                                        \
1591           "The multiple of deviation from mean to use for buffering"       \
1592           "against volatility in free list demand.")                       \
1593                                                                            \
1594   product(uintx, FLSCoalescePolicy, 2,                                     \
1595           "CMS: Aggression level for coalescing, increasing from 0 to 4")  \
1596                                                                            \
1597   product(bool, FLSAlwaysCoalesceLarge, false,                             \
1598           "CMS: Larger free blocks are always available for coalescing")   \
1599                                                                            \
1600   product(double, FLSLargestBlockCoalesceProximity, 0.99,                  \
1601           "CMS: the smaller the percentage the greater the coalition force")\
1602                                                                            \
1603   product(double, CMSSmallCoalSurplusPercent, 1.05,                        \
1604           "CMS: the factor by which to inflate estimated demand of small"  \
1605           " block sizes to prevent coalescing with an adjoining block")    \
1606                                                                            \
1607   product(double, CMSLargeCoalSurplusPercent, 0.95,                        \
1608           "CMS: the factor by which to inflate estimated demand of large"  \
1609           " block sizes to prevent coalescing with an adjoining block")    \
1610                                                                            \
1611   product(double, CMSSmallSplitSurplusPercent, 1.10,                       \
1612           "CMS: the factor by which to inflate estimated demand of small"  \
1613           " block sizes to prevent splitting to supply demand for smaller" \
1614           " blocks")                                                       \
1615                                                                            \
1616   product(double, CMSLargeSplitSurplusPercent, 1.00,                       \
1617           "CMS: the factor by which to inflate estimated demand of large"  \
1618           " block sizes to prevent splitting to supply demand for smaller" \
1619           " blocks")                                                       \
1620                                                                            \
1621   product(bool, CMSExtrapolateSweep, false,                                \
1622           "CMS: cushion for block demand during sweep")                    \
1623                                                                            \
1624   product(uintx, CMS_SweepWeight, 75,                                      \
1625           "Percentage (0-100) used to weight the current sample when "     \
1626           "computing exponentially decaying average for inter-sweep "      \
1627           "duration")                                                      \
1628                                                                            \
1629   product(uintx, CMS_SweepPadding, 1,                                      \
```

```
1630           "The multiple of deviation from mean to use for buffering "       \
1631           "against volatility in inter-sweep duration.")                    \
1632                                                                             \
1633   product(uintx, CMS_SweepTimerThresholdMillis, 10,                         \
1634           "Skip block flux-rate sampling for an epoch unless inter-sweep "  \
1635           "duration exceeds this threhold in milliseconds")                 \
1636                                                                             \
1637   develop(bool, CMSTraceIncrementalMode, false,                             \
1638           "Trace CMS incremental mode")                                     \
1639                                                                             \
1640   develop(bool, CMSTraceIncrementalPacing, false,                           \
1641           "Trace CMS incremental mode pacing computation")                  \
1642                                                                             \
1643   develop(bool, CMSTraceThreadState, false,                                 \
1644           "Trace the CMS thread state (enable the trace_state() method)")   \
1645                                                                             \
1646   product(bool, CMSClassUnloadingEnabled, false,                            \
1647           "Whether class unloading enabled when using CMS GC")              \
1648                                                                             \
1649   product(uintx, CMSClassUnloadingMaxInterval, 0,                           \
1650           "When CMS class unloading is enabled, the maximum CMS cycle count"\
1651           " for which classes may not be unloaded")                         \
1652                                                                             \
1653   product(bool, CMSCompactWhenClearAllSoftRefs, true,                       \
1654           "Compact when asked to collect CMS gen with clear_all_soft_refs") \
1655                                                                             \
1656   product(bool, UseCMSCompactAtFullCollection, true,                        \
1657           "Use mark sweep compact at full collections")                     \
1658                                                                             \
1659   product(uintx, CMSFullGCsBeforeCompaction, 0,                             \
1660           "Number of CMS full collection done before compaction if > 0")    \
1661                                                                             \
1662   develop(intx, CMSDictionaryChoice, 0,                                     \
1663           "Use BinaryTreeDictionary as default in the CMS generation")      \
1664                                                                             \
1665   product(uintx, CMSIndexedFreeListReplenish, 4,                            \
1666           "Replenish an indexed free list with this number of chunks")     \
1667                                                                             \
1668   product(bool, CMSReplenishIntermediate, true,                             \
1669           "Replenish all intermediate free-list caches")                    \
1670                                                                             \
1671   product(bool, CMSSplitIndexedFreeListBlocks, true,                        \
1672           "When satisfying batched demand, split blocks from the "          \
1673           "IndexedFreeList whose size is a multiple of requested size")     \
1674                                                                             \
1675   product(bool, CMSLoopWarn, false,                                         \
1676           "Warn in case of excessive CMS looping")                          \
1677                                                                             \
1678   develop(bool, CMSOverflowEarlyRestoration, false,                         \
1679           "Whether preserved marks should be restored early")               \
1680                                                                             \
1681   product(uintx, MarkStackSize, NOT_LP64(32*K) LP64_ONLY(4*M),              \
1682           "Size of marking stack")                                          \
1683                                                                             \
1684   product(uintx, MarkStackSizeMax, NOT_LP64(4*M) LP64_ONLY(512*M),          \
1685           "Max size of marking stack")                                      \
1686                                                                             \
1687   notproduct(bool, CMSMarkStackOverflowALot, false,                         \
1688           "Whether we should simulate frequent marking stack / work queue"  \
1689           " overflow")                                                      \
1690                                                                             \
1691   notproduct(uintx, CMSMarkStackOverflowInterval, 1000,                     \
1692           "An ‘interval’ counter that determines how frequently"            \
1693           " we simulate overflow; a smaller number increases frequency")    \
1694                                                                             \
1695   product(uintx, CMSMaxAbortablePrecleanLoops, 0,                           \
```



```
1696           "(Temporary, subject to experimentation)"                         \
1697           "Maximum number of abortable preclean iterations, if > 0")        \
1698                                                                             \
1699   product(intx, CMSMaxAbortablePrecleanTime, 5000,                          \
1700           "(Temporary, subject to experimentation)"                         \
1701           "Maximum time in abortable preclean in ms")                       \
1702                                                                             \
1703   product(uintx, CMSAbortablePrecleanMinWorkPerIteration, 100,              \
1704           "(Temporary, subject to experimentation)"                         \
1705           "Nominal minimum work per abortable preclean iteration")          \
1706                                                                             \
1707   manageable(intx, CMSAbortablePrecleanWaitMillis, 100,                     \
1708           "(Temporary, subject to experimentation)"                         \
1709           " Time that we sleep between iterations when not given"           \
1710           " enough work per iteration")                                     \
1711                                                                             \
1712   product(uintx, CMSRescanMultiple, 32,                                     \
1713           "Size (in cards) of CMS parallel rescan task")                    \
1714                                                                             \
1715   product(uintx, CMSConcMarkMultiple, 32,                                   \
1716           "Size (in cards) of CMS concurrent MT marking task")              \
1717                                                                             \
1718   product(uintx, CMSRevisitStackSize, 1*M,                                  \
1719           "Size of CMS KlassKlass revisit stack")                           \
1720                                                                             \
1721   product(bool, CMSAbortSemantics, false,                                   \
1722           "Whether abort-on-overflow semantics is implemented")             \
1723                                                                             \
1724   product(bool, CMSParallelRemarkEnabled, true,                             \
1725           "Whether parallel remark enabled (only if ParNewGC)")             \
1726                                                                             \
1727   product(bool, CMSParallelSurvivorRemarkEnabled, true,                     \
1728           "Whether parallel remark of survivor space"                       \
1729           " enabled (effective only if CMSParallelRemarkEnabled)")          \
1730                                                                             \
1731   product(bool, CMSPLABRecordAlways, true,                                  \
1732           "Whether to always record survivor space PLAB bdries"             \
1733           " (effective only if CMSParallelSurvivorRemarkEnabled)")          \
1734                                                                             \
1735   product(bool, CMSConcurrentMTEnabled, true,                               \
1736           "Whether multi-threaded concurrent work enabled (if ParNewGC)")   \
1737                                                                             \
1738   product(bool, CMSPermGenPrecleaningEnabled, true,                         \
1739           "Whether concurrent precleaning enabled in perm gen"              \
1740           " (effective only when CMSPrecleaningEnabled is true)")           \
1741                                                                             \
1742   product(bool, CMSPrecleaningEnabled, true,                                \
1743           "Whether concurrent precleaning enabled")                         \
1744                                                                             \
1745   product(uintx, CMSPrecleanIter, 3,                                        \
1746           "Maximum number of precleaning iteration passes")                 \
1747                                                                             \
1748   product(uintx, CMSPrecleanNumerator, 2,                                   \
1749           "CMSPrecleanNumerator:CMSPrecleanDenominator yields convergence"  \
1750           " ratio")                                                         \
1751                                                                             \
1752   product(uintx, CMSPrecleanDenominator, 3,                                 \
1753           "CMSPrecleanNumerator:CMSPrecleanDenominator yields convergence"  \
1754           " ratio")                                                         \
1755                                                                             \
1756   product(bool, CMSPrecleanRefLists1, true,                                 \
1757           "Preclean ref lists during (initial) preclean phase")             \
1758                                                                             \
1759   product(bool, CMSPrecleanRefLists2, false,                                \
1760           "Preclean ref lists during abortable preclean phase")             \
1761                                                                             \
```

```
1762   product(bool, CMSPrecleanSurvivors1, false,                                \
1763           "Preclean survivors during (initial) preclean phase")              \
1764                                                                              \
1765   product(bool, CMSPrecleanSurvivors2, true,                                 \
1766           "Preclean survivors during abortable preclean phase")              \
1767                                                                              \
1768   product(uintx, CMSPrecleanThreshold, 1000,                                 \
1769           "Don't re-iterate if #dirty cards less than this")                 \
1770                                                                              \
1771   product(bool, CMSCleanOnEnter, true,                                       \
1772           "Clean-on-enter optimization for reducing number of dirty cards") \
1773                                                                              \
1774   product(uintx, CMSRemarkVerifyVariant, 1,                                  \
1775           "Choose variant (1,2) of verification following remark")           \
1776                                                                              \
1777   product(uintx, CMSScheduleRemarkEdenSizeThreshold, 2*M,                    \
1778           "If Eden used is below this value, don't try to schedule remark") \
1779                                                                              \
1780   product(uintx, CMSScheduleRemarkEdenPenetration, 50,                       \
1781           "The Eden occupancy % at which to try and schedule remark pause") \
1782                                                                              \
1783   product(uintx, CMSScheduleRemarkSamplingRatio, 5,                          \
1784           "Start sampling Eden top at least before yg occupancy reaches"     \
1785           " 1/<ratio> of the size at which we plan to schedule remark")      \
1786                                                                              \
1787   product(uintx, CMSSamplingGrain, 16*K,                                     \
1788           "The minimum distance between eden samples for CMS (see above)")   \
1789                                                                              \
1790   product(bool, CMSScavengeBeforeRemark, false,                              \
1791           "Attempt scavenge before the CMS remark step")                     \
1792                                                                              \
1793   develop(bool, CMSTraceSweeper, false,                                      \
1794           "Trace some actions of the CMS sweeper")                           \
1795                                                                              \
1796   product(uintx, CMSWorkQueueDrainThreshold, 10,                             \
1797           "Don't drain below this size per parallel worker/thief")           \
1798                                                                              \
1799   manageable(intx, CMSWaitDuration, 2000,                                    \
1800           "Time in milliseconds that CMS thread waits for young GC")         \
1801                                                                              \
1802   product(bool, CMSYield, true,                                              \
1803           "Yield between steps of concurrent mark & sweep")                  \
1804                                                                              \
1805   product(uintx, CMSBitMapYieldQuantum, 10*M,                                \
1806           "Bitmap operations should process at most this many bits"          \
1807           "between yields")                                                  \
1808                                                                              \
1809   product(bool, CMSDumpAtPromotionFailure, false,                            \
1810           "Dump useful information about the state of the CMS old "          \
1811           " generation upon a promotion failure.")                           \
1812                                                                              \
1813   product(bool, CMSPrintChunksInDump, false,                                 \
1814           "In a dump enabled by CMSDumpAtPromotionFailure, include "         \
1815           " more detailed information about the free chunks.")               \
1816                                                                              \
1817   product(bool, CMSPrintObjectsInDump, false,                                \
1818           "In a dump enabled by CMSDumpAtPromotionFailure, include "         \
1819           " more detailed information about the allocated objects.")         \
1820                                                                              \
1821   diagnostic(bool, FLSVerifyAllHeapReferences, false,                        \
1822           "Verify that all refs across the FLS boundary "                    \
1823           " are to valid objects")                                           \
1824                                                                              \
1825   diagnostic(bool, FLSVerifyLists, false,                                    \
1826           "Do lots of (expensive) FreeListSpace verification")               \
1827                                                                              \
```



```
1828   diagnostic(bool, FLSVerifyIndexTable, false,                               \
1829           "Do lots of (expensive) FLS index table verification")             \
1830                                                                              \
1831   develop(bool, FLSVerifyDictionary, false,                                  \
1832           "Do lots of (expensive) FLS dictionary verification")              \
1833                                                                              \
1834   develop(bool, VerifyBlockOffsetArray, false,                               \
1835           "Do (expensive!) block offset array verification")                 \
1836                                                                              \
1837   product(bool, BlockOffsetArrayUseUnallocatedBlock, false,                  \
1838           "Maintain _unallocated_block in BlockOffsetArray"                  \
1839           " (currently applicable only to CMS collector)")                   \
1840                                                                              \
1841   develop(bool, TraceCMSState, false,                                        \
1842           "Trace the state of the CMS collection")                           \
1843                                                                              \
1844   product(intx, RefDiscoveryPolicy, 0,                                       \
1845           "Whether reference-based(0) or referent-based(1)")                 \
1846                                                                              \
1847   product(bool, ParallelRefProcEnabled, false,                               \
1848           "Enable parallel reference processing whenever possible")          \
1849                                                                              \
1850   product(bool, ParallelRefProcBalancingEnabled, true,                       \
1851           "Enable balancing of reference processing queues")                 \
1852                                                                              \
1853   product(intx, CMSTriggerRatio, 80,                                         \
1854           "Percentage of MinHeapFreeRatio in CMS generation that is "        \
1855           "allocated before a CMS collection cycle commences")               \
1856                                                                              \
1857   product(intx, CMSTriggerPermRatio, 80,                                     \
1858           "Percentage of MinHeapFreeRatio in the CMS perm generation that " \
1859           "is allocated before a CMS collection cycle commences, that "      \
1860           "also collects the perm generation")                               \
1861                                                                              \
1862   product(uintx, CMSBootstrapOccupancy, 50,                                  \
1863           "Percentage CMS generation occupancy at which to "                 \
1864           "initiate CMS collection for bootstrapping collection stats")      \
1865                                                                              \
1866   product(intx, CMSInitiatingOccupancyFraction, -1,                          \
1867           "Percentage CMS generation occupancy to start a CMS collection "   \
1868           "cycle. A negative value means that CMSTriggerRatio is used")      \
1869                                                                              \
1870   product(uintx, InitiatingHeapOccupancyPercent, 45,                         \
1871           "Percentage of the (entire) heap occupancy to start a "            \
1872           "concurrent GC cycle. It us used by GCs that trigger a "           \
1873           "concurrent GC cycle based on the occupancy of the entire heap, " \
1874           "not just one of the generations (e.g., G1). A value of 0 "        \
1875           "denotes 'do constant GC cycles'.")                                \
1876                                                                              \
1877   product(intx, CMSInitiatingPermOccupancyFraction, -1,                      \
1878           "Percentage CMS perm generation occupancy to start a "             \
1879           "CMScollection cycle. A negative value means that "                \
1880           "CMSTriggerPermRatio is used")                                     \
1881                                                                              \
1882   product(bool, UseCMSInitiatingOccupancyOnly, false,                        \
1883           "Only use occupancy as a crierion for starting a CMS collection") \
1884                                                                              \
1885   product(intx, CMSIsTooFullPercentage, 98,                                  \
1886           "An absolute ceiling above which CMS will always consider the "    \
1887           "perm gen ripe for collection")                                    \
1888                                                                              \
1889   develop(bool, CMSTestInFreeList, false,                                    \
1890           "Check if the coalesced range is already in the "                  \
1891           "free lists as claimed")                                           \
1892                                                                              \
1893   notproduct(bool, CMSVerifyReturnedBytes, false,                            \
```

```
1894           "Check that all the garbage collected was returned to the "       \
1895           "free lists.")                                                    \
1896                                                                             \
1897   notproduct(bool, ScavengeALot, false,                                     \
1898           "Force scavenge at every Nth exit from the runtime system "       \
1899           "(N=ScavengeALotInterval)")                                       \
1900                                                                             \
1901   develop(bool, FullGCALot, false,                                          \
1902           "Force full gc at every Nth exit from the runtime system "        \
1903           "(N=FullGCALotInterval)")                                         \
1904                                                                             \
1905   notproduct(bool, GCALotAtAllSafepoints, false,                            \
1906           "Enforce ScavengeALot/GCALot at all potential safepoints")        \
1907                                                                             \
1908   product(bool, PrintPromotionFailure, false,                               \
1909           "Print additional diagnostic information following "              \
1910           " promotion failure")                                             \
1911                                                                             \
1912   notproduct(bool, PromotionFailureALot, false,                             \
1913           "Use promotion failure handling on every youngest generation "    \
1914           "collection")                                                     \
1915                                                                             \
1916   develop(uintx, PromotionFailureALotCount, 1000,                           \
1917           "Number of promotion failures occurring at ParGCAllocBuffer"      \
1918           "refill attempts (ParNew) or promotion attempts "                 \
1919           "(other young collectors) ")                                      \
1920                                                                             \
1921   develop(uintx, PromotionFailureALotInterval, 5,                           \
1922           "Total collections between promotion failures alot")              \
1923                                                                             \
1924   experimental(intx, WorkStealingSleepMillis, 1,                            \
1925           "Sleep time when sleep is used for yields")                       \
1926                                                                             \
1927   experimental(uintx, WorkStealingYieldsBeforeSleep, 1000,                  \
1928           "Number of yields before a sleep is done during workstealing")    \
1929                                                                             \
1930   experimental(uintx, WorkStealingHardSpins, 4096,                          \
1931           "Number of iterations in a spin loop between checks on "          \
1932           "time out of hard spin")                                          \
1933                                                                             \
1934   experimental(uintx, WorkStealingSpinToYieldRatio, 10,                     \
1935           "Ratio of hard spins to calls to yield")                          \
1936                                                                             \
1937   product(uintx, PreserveMarkStackSize, 1024,                               \
1938           "Size for stack used in promotion failure handling")              \
1939                                                                             \
1940   develop(uintx, ObjArrayMarkingStride, 512,                                \
1941           "Number of ObjArray elements to push onto the marking stack"      \
1942           "before pushing a continuation entry")                            \
1943                                                                             \
1944   product_pd(bool, UseTLAB, "Use thread-local object allocation")           \
1945                                                                             \
1946   product_pd(bool, ResizeTLAB,                                              \
1947           "Dynamically resize tlab size for threads")                       \
1948                                                                             \
1949   product(bool, ZeroTLAB, false,                                            \
1950           "Zero out the newly created TLAB")                                \
1951                                                                             \
1952   product(bool, FastTLABRefill, true,                                       \
1953           "Use fast TLAB refill code")                                      \
1954                                                                             \
1955   product(bool, PrintTLAB, false,                                           \
1956           "Print various TLAB related information")                         \
1957                                                                             \
1958   product(bool, TLABStats, true,                                            \
1959           "Print various TLAB related information")                         \
```



```
1960                                                                             \
1961   product(bool, PrintRevisitStats, false,                                   \
1962           "Print revisit (klass and MDO) stack related information")        \
1963                                                                             \
1964   EMBEDDED_ONLY(product(bool, LowMemoryProtection, true,                    \
1965           "Enable LowMemoryProtection"))                                    \
1966                                                                             \
1967   product_pd(bool, NeverActAsServerClassMachine,                            \
1968           "Never act like a server-class machine")                          \
1969                                                                             \
1970   product(bool, AlwaysActAsServerClassMachine, false,                       \
1971           "Always act like a server-class machine")                         \
1972                                                                             \
1973   product_pd(uint64_t, MaxRAM,                                              \
1974           "Real memory size (in bytes) used to set maximum heap size")      \
1975                                                                             \
1976   product(uintx, ErgoHeapSizeLimit, 0,                                      \
1977           "Maximum ergonomically set heap size (in bytes); zero means use " \
1978           "MaxRAM / MaxRAMFraction")                                        \
1979                                                                             \
1980   product(uintx, MaxRAMFraction, 4,                                         \
1981           "Maximum fraction (1/n) of real memory used for maximum heap "    \
1982           "size")                                                           \
1983                                                                             \
1984   product(uintx, DefaultMaxRAMFraction, 4,                                  \
1985           "Maximum fraction (1/n) of real memory used for maximum heap "    \
1986           "size; deprecated: to be renamed to MaxRAMFraction")              \
1987                                                                             \
1988   product(uintx, MinRAMFraction, 2,                                         \
1989           "Minimum fraction (1/n) of real memory used for maxmimum heap "   \
1990           "size on systems with small physical memory size")                \
1991                                                                             \
1992   product(uintx, InitialRAMFraction, 64,                                    \
1993           "Fraction (1/n) of real memory used for initial heap size")       \
1994                                                                             \
1995   product(bool, UseAutoGCSelectPolicy, false,                               \
1996           "Use automatic collection selection policy")                      \
1997                                                                             \
1998   product(uintx, AutoGCSelectPauseMillis, 5000,                             \
1999           "Automatic GC selection pause threshhold in ms")                  \
2000                                                                             \
2001   product(bool, UseAdaptiveSizePolicy, true,                                \
2002           "Use adaptive generation sizing policies")                        \
2003                                                                             \
2004   product(bool, UsePSAdaptiveSurvivorSizePolicy, true,                      \
2005           "Use adaptive survivor sizing policies")                          \
2006                                                                             \
2007   product(bool, UseAdaptiveGenerationSizePolicyAtMinorCollection, true,     \
2008           "Use adaptive young-old sizing policies at minor collections")    \
2009                                                                             \
2010   product(bool, UseAdaptiveGenerationSizePolicyAtMajorCollection, true,     \
2011           "Use adaptive young-old sizing policies at major collections")    \
2012                                                                             \
2013   product(bool, UseAdaptiveSizePolicyWithSystemGC, false,                   \
2014           "Use statistics from System.GC for adaptive size policy")         \
2015                                                                             \
2016   product(bool, UseAdaptiveGCBoundary, false,                               \
2017           "Allow young-old boundary to move")                               \
2018                                                                             \
2019   develop(bool, TraceAdaptiveGCBoundary, false,                             \
2020           "Trace young-old boundary moves")                                 \
2021                                                                             \
2022   develop(intx, PSAdaptiveSizePolicyResizeVirtualSpaceAlot, -1,             \
2023           "Resize the virtual spaces of the young or old generations")      \
2024                                                                             \
2025   product(uintx, AdaptiveSizeThroughPutPolicy, 0,                           \
```

```
2026           "Policy for changeing generation size for throughput goals")      \
2027                                                                             \
2028   product(uintx, AdaptiveSizePausePolicy, 0,                                \
2029           "Policy for changing generation size for pause goals")            \
2030                                                                             \
2031   develop(bool, PSAdjustTenuredGenForMinorPause, false,                     \
2032           "Adjust tenured generation to achive a minor pause goal")         \
2033                                                                             \
2034   develop(bool, PSAdjustYoungGenForMajorPause, false,                       \
2035           "Adjust young generation to achive a major pause goal")           \
2036                                                                             \
2037   product(uintx, AdaptiveSizePolicyInitializingSteps, 20,                   \
2038           "Number of steps where heuristics is used before data is used")   \
2039                                                                             \
2040   develop(uintx, AdaptiveSizePolicyReadyThreshold, 5,                       \
2041           "Number of collections before the adaptive sizing is started")    \
2042                                                                             \
2043   product(uintx, AdaptiveSizePolicyOutputInterval, 0,                       \
2044           "Collecton interval for printing information; zero => never")     \
2045                                                                             \
2046   product(bool, UseAdaptiveSizePolicyFootprintGoal, true,                   \
2047           "Use adaptive minimum footprint as a goal")                       \
2048                                                                             \
2049   product(uintx, AdaptiveSizePolicyWeight, 10,                              \
2050           "Weight given to exponential resizing, between 0 and 100")        \
2051                                                                             \
2052   product(uintx, AdaptiveTimeWeight,       25,                              \
2053           "Weight given to time in adaptive policy, between 0 and 100")     \
2054                                                                             \
2055   product(uintx, PausePadding, 1,                                           \
2056           "How much buffer to keep for pause time")                         \
2057                                                                             \
2058   product(uintx, PromotedPadding, 3,                                        \
2059           "How much buffer to keep for promotion failure")                  \
2060                                                                             \
2061   product(uintx, SurvivorPadding, 3,                                        \
2062           "How much buffer to keep for survivor overflow")                  \
2063                                                                             \
2064   product(uintx, AdaptivePermSizeWeight, 20,                                \
2065           "Weight for perm gen exponential resizing, between 0 and 100")    \
2066                                                                             \
2067   product(uintx, PermGenPadding, 3,                                         \
2068           "How much buffer to keep for perm gen sizing")                    \
2069                                                                             \
2070   product(uintx, ThresholdTolerance, 10,                                    \
2071           "Allowed collection cost difference between generations")         \
2072                                                                             \
2073   product(uintx, AdaptiveSizePolicyCollectionCostMargin, 50,                \
2074           "If collection costs are within margin, reduce both by full "     \
2075           "delta")                                                          \
2076                                                                             \
2077   product(uintx, YoungGenerationSizeIncrement, 20,                          \
2078           "Adaptive size percentage change in young generation")            \
2079                                                                             \
2080   product(uintx, YoungGenerationSizeSupplement, 80,                         \
2081           "Supplement to YoungedGenerationSizeIncrement used at startup")   \
2082                                                                             \
2083   product(uintx, YoungGenerationSizeSupplementDecay, 8,                     \
2084           "Decay factor to YoungedGenerationSizeSupplement")                \
2085                                                                             \
2086   product(uintx, TenuredGenerationSizeIncrement, 20,                        \
2087           "Adaptive size percentage change in tenured generation")          \
2088                                                                             \
2089   product(uintx, TenuredGenerationSizeSupplement, 80,                       \
2090           "Supplement to TenuredGenerationSizeIncrement used at startup")   \
2091                                                                             \
```



```
2092   product(uintx, TenuredGenerationSizeSupplementDecay, 2,                   \
2093           "Decay factor to TenuredGenerationSizeIncrement")                 \
2094                                                                             \
2095   product(uintx, MaxGCPauseMillis, max_uintx,                                \
2096           "Adaptive size policy maximum GC pause time goal in msec, "       \
2097           "or (G1 Only) the max. GC time per MMU time slice")               \
2098                                                                             \
2099   product(uintx, GCPauseIntervalMillis, 0,                                  \
2100           "Time slice for MMU specification")                               \
2101                                                                             \
2102   product(uintx, MaxGCMinorPauseMillis, max_uintx,                          \
2103           "Adaptive size policy maximum GC minor pause time goal in msec")  \
2104                                                                             \
2105   product(uintx, GCTimeRatio, 99,                                           \
2106           "Adaptive size policy application time to GC time ratio")         \
2107                                                                             \
2108   product(uintx, AdaptiveSizeDecrementScaleFactor, 4,                       \
2109           "Adaptive size scale down factor for shrinking")                  \
2110                                                                             \
2111   product(bool, UseAdaptiveSizeDecayMajorGCCost, true,                      \
2112           "Adaptive size decays the major cost for long major intervals")   \
2113                                                                             \
2114   product(uintx, AdaptiveSizeMajorGCDecayTimeScale, 10,                     \
2115           "Time scale over which major costs decay")                        \
2116                                                                             \
2117   product(uintx, MinSurvivorRatio, 3,                                       \
2118           "Minimum ratio of young generation/survivor space size")          \
2119                                                                             \
2120   product(uintx, InitialSurvivorRatio, 8,                                   \
2121           "Initial ratio of eden/survivor space size")                      \
2122                                                                             \
2123   product(uintx, BaseFootPrintEstimate, 256*M,                              \
2124           "Estimate of footprint other than Java Heap")                     \
2125                                                                             \
2126   product(bool, UseGCOverheadLimit, true,                                   \
2127           "Use policy to limit of proportion of time spent in GC "          \
2128           "before an OutOfMemory error is thrown")                          \
2129                                                                             \
2130   product(uintx, GCTimeLimit, 98,                                           \
2131           "Limit of proportion of time spent in GC before an OutOfMemory"   \
2132           "error is thrown (used with GCHeapFreeLimit)")                    \
2133                                                                             \
2134   product(uintx, GCHeapFreeLimit, 2,                                        \
2135           "Minimum percentage of free space after a full GC before an "     \
2136           "OutOfMemoryError is thrown (used with GCTimeLimit)")             \
2137                                                                             \
2138   develop(uintx, AdaptiveSizePolicyGCTimeLimitThreshold, 5,                 \
2139           "Number of consecutive collections before gc time limit fires")   \
2140                                                                             \
2141   product(bool, PrintAdaptiveSizePolicy, false,                             \
2142           "Print information about AdaptiveSizePolicy")                     \
2143                                                                             \
2144   product(intx, PrefetchCopyIntervalInBytes, -1,                            \
2145           "How far ahead to prefetch destination area (<= 0 means off)")    \
2146                                                                             \
2147   product(intx, PrefetchScanIntervalInBytes, -1,                            \
2148           "How far ahead to prefetch scan area (<= 0 means off)")           \
2149                                                                             \
2150   product(intx, PrefetchFieldsAhead, -1,                                    \
2151           "How many fields ahead to prefetch in oop scan (<= 0 means off)") \
2152                                                                             \
2153   develop(bool, UsePrefetchQueue, true,                                     \
2154           "Use the prefetch queue during PS promotion")                     \
2155                                                                             \
2156   diagnostic(bool, VerifyBeforeExit, trueInDebug,                           \
2157           "Verify system before exiting")                                   \
```

```
2158                                                                              \
2159   diagnostic(bool, VerifyBeforeGC, false,                                    \
2160           "Verify memory system before GC")                                  \
2161                                                                              \
2162   diagnostic(bool, VerifyAfterGC, false,                                     \
2163           "Verify memory system after GC")                                   \
2164                                                                              \
2165   diagnostic(bool, VerifyDuringGC, false,                                    \
2166           "Verify memory system during GC (between phases)")                 \
2167                                                                              \
2168   diagnostic(bool, GCParallelVerificationEnabled, true,                      \
2169           "Enable parallel memory system verification")                      \
2170                                                                              \
2171   diagnostic(bool, DeferInitialCardMark, false,                              \
2172           "When +ReduceInitialCardMarks, explicitly defer any that "         \
2173            "may arise from new_pre_store_barrier")                           \
2174                                                                              \
2175   diagnostic(bool, VerifyRememberedSets, false,                              \
2176           "Verify GC remembered sets")                                       \
2177                                                                              \
2178   diagnostic(bool, VerifyObjectStartArray, true,                             \
2179           "Verify GC object start array if verify before/after")             \
2180                                                                              \
2181   product(bool, DisableExplicitGC, false,                                    \
2182           "Tells whether calling System.gc() does a full GC")                \
2183                                                                              \
2184   notproduct(bool, CheckMemoryInitialization, false,                         \
2185           "Checks memory initialization")                                    \
2186                                                                              \
2187   product(bool, CollectGen0First, false,                                     \
2188           "Collect youngest generation before each full GC")                 \
2189                                                                              \
2190   diagnostic(bool, BindCMSThreadToCPU, false,                                \
2191           "Bind CMS Thread to CPU if possible")                              \
2192                                                                              \
2193   diagnostic(uintx, CPUForCMSThread, 0,                                      \
2194           "When BindCMSThreadToCPU is true, the CPU to bind CMS thread to") \
2195                                                                              \
2196   product(bool, BindGCTaskThreadsToCPUs, false,                              \
2197           "Bind GCTaskThreads to CPUs if possible")                          \
2198                                                                              \
2199   product(bool, UseGCTaskAffinity, false,                                    \
2200           "Use worker affinity when asking for GCTasks")                     \
2201                                                                              \
2202   product(uintx, ProcessDistributionStride, 4,                               \
2203           "Stride through processors when distributing processes")           \
2204                                                                              \
2205   product(uintx, CMSCoordinatorYieldSleepCount, 10,                          \
2206           "number of times the coordinator GC thread will sleep while "      \
2207           "yielding before giving up and resuming GC")                       \
2208                                                                              \
2209   product(uintx, CMSYieldSleepCount, 0,                                      \
2210           "number of times a GC thread (minus the coordinator) "             \
2211           "will sleep while yielding before giving up and resuming GC")      \
2212                                                                              \
2213   /* gc tracing */                                                           \
2214   manageable(bool, PrintGC, false,                                           \
2215           "Print message at garbage collect")                                \
2216                                                                              \
2217   manageable(bool, PrintGCDetails, false,                                    \
2218           "Print more details at garbage collect")                           \
2219                                                                              \
2220   manageable(bool, PrintGCDateStamps, false,                                 \
2221           "Print date stamps at garbage collect")                            \
2222                                                                              \
2223   manageable(bool, PrintGCTimeStamps, false,                                 \
```



```
2224           "Print timestamps at garbage collect")                             \
2225                                                                              \
2226   product(bool, PrintGCTaskTimeStamps, false,                                \
2227           "Print timestamps for individual gc worker thread tasks")          \
2228                                                                              \
2229   develop(intx, ConcGCYieldTimeout, 0,                                       \
2230           "If non-zero, assert that GC threads yield within this # of ms.") \
2231                                                                              \
2232   notproduct(bool, TraceMarkSweep, false,                                    \
2233           "Trace mark sweep")                                                \
2234                                                                              \
2235   product(bool, PrintReferenceGC, false,                                     \
2236           "Print times spent handling reference objects during GC "          \
2237           " (enabled only when PrintGCDetails)")                             \
2238                                                                              \
2239   develop(bool, TraceReferenceGC, false,                                     \
2240           "Trace handling of soft/weak/final/phantom references")            \
2241                                                                              \
2242   develop(bool, TraceFinalizerRegistration, false,                           \
2243          "Trace registration of final references")                           \
2244                                                                              \
2245   notproduct(bool, TraceScavenge, false,                                     \
2246           "Trace scavenge")                                                  \
2247                                                                              \
2248   product_rw(bool, TraceClassLoading, false,                                 \
2249           "Trace all classes loaded")                                        \
2250                                                                              \
2251   product(bool, TraceClassLoadingPreorder, false,                            \
2252           "Trace all classes loaded in order referenced (not loaded)")       \
2253                                                                              \
2254   product_rw(bool, TraceClassUnloading, false,                               \
2255           "Trace unloading of classes")                                      \
2256                                                                              \
2257   product_rw(bool, TraceLoaderConstraints, false,                            \
2258           "Trace loader constraints")                                        \
2259                                                                              \
2260   product(bool, TraceGen0Time, false,                                        \
2261           "Trace accumulated time for Gen 0 collection")                     \
2262                                                                              \
2263   product(bool, TraceGen1Time, false,                                        \
2264           "Trace accumulated time for Gen 1 collection")                     \
2265                                                                              \
2266   product(bool, PrintTenuringDistribution, false,                            \
2267           "Print tenuring age information")                                  \
2268                                                                              \
2269   product_rw(bool, PrintHeapAtGC, false,                                     \
2270           "Print heap layout before and after each GC")                      \
2271                                                                              \
2272   product_rw(bool, PrintHeapAtGCExtended, false,                             \
2273           "Prints extended information about the layout of the heap "        \
2274           "when -XX:+PrintHeapAtGC is set")                                  \
2275                                                                              \
2276   product(bool, PrintHeapAtSIGBREAK, true,                                   \
2277           "Print heap layout in response to SIGBREAK")                       \
2278                                                                              \
2279   manageable(bool, PrintClassHistogramBeforeFullGC, false,                   \
2280           "Print a class histogram before any major stop-world GC")          \
2281                                                                              \
2282   manageable(bool, PrintClassHistogramAfterFullGC, false,                    \
2283           "Print a class histogram after any major stop-world GC")           \
2284                                                                              \
2285   manageable(bool, PrintClassHistogram, false,                               \
2286           "Print a histogram of class instances")                            \
2287                                                                              \
2288   develop(bool, TraceWorkGang, false,                                        \
2289           "Trace activities of work gangs")                                  \
```

```
2290                                                                              \
2291   product(bool, TraceParallelOldGCTasks, false,                              \
2292           "Trace multithreaded GC activity")                                 \
2293                                                                              \
2294   develop(bool, TraceBlockOffsetTable, false,                                \
2295           "Print BlockOffsetTable maps")                                     \
2296                                                                              \
2297   develop(bool, TraceCardTableModRefBS, false,                               \
2298           "Print CardTableModRefBS maps")                                    \
2299                                                                              \
2300   develop(bool, TraceGCTaskManager, false,                                   \
2301           "Trace actions of the GC task manager")                            \
2302                                                                              \
2303   develop(bool, TraceGCTaskQueue, false,                                     \
2304           "Trace actions of the GC task queues")                             \
2305                                                                              \
2306   develop(bool, TraceGCTaskThread, false,                                    \
2307           "Trace actions of the GC task threads")                            \
2308                                                                              \
2309   product(bool, PrintParallelOldGCPhaseTimes, false,                         \
2310           "Print the time taken by each parallel old gc phase."              \
2311           "PrintGCDetails must also be enabled.")                            \
2312                                                                              \
2313   develop(bool, TraceParallelOldGCMarkingPhase, false,                       \
2314           "Trace parallel old gc marking phase")                             \
2315                                                                              \
2316   develop(bool, TraceParallelOldGCSummaryPhase, false,                       \
2317           "Trace parallel old gc summary phase")                             \
2318                                                                              \
2319   develop(bool, TraceParallelOldGCCompactionPhase, false,                    \
2320           "Trace parallel old gc compaction phase")                          \
2321                                                                              \
2322   develop(bool, TraceParallelOldGCDensePrefix, false,                        \
2323           "Trace parallel old gc dense prefix computation")                  \
2324                                                                              \
2325   develop(bool, IgnoreLibthreadGPFault, false,                               \
2326           "Suppress workaround for libthread GP fault")                      \
2327                                                                              \
2328   product(bool, PrintJNIGCStalls, false,                                     \
2329           "Print diagnostic message when GC is stalled"                      \
2330           "by JNI critical section")                                         \
2331                                                                              \
2332   /* JVMTI heap profiling */                                                 \
2333                                                                              \
2334   diagnostic(bool, TraceJVMTIObjectTagging, false,                           \
2335           "Trace JVMTI object tagging calls")                                \
2336                                                                              \
2337   diagnostic(bool, VerifyBeforeIteration, false,                             \
2338           "Verify memory system before JVMTI iteration")                     \
2339                                                                              \
2340   /* compiler interface */                                                   \
2341                                                                              \
2342   develop(bool, CIPrintCompilerName, false,                                  \
2343           "when CIPrint is active, print the name of the active compiler")   \
2344                                                                              \
2345   develop(bool, CIPrintCompileQueue, false,                                  \
2346           "display the contents of the compile queue whenever a "            \
2347           "compilation is enqueued")                                         \
2348                                                                              \
2349   develop(bool, CIPrintRequests, false,                                      \
2350           "display every request for compilation")                           \
2351                                                                              \
2352   product(bool, CITime, false,                                               \
2353           "collect timing information for compilation")                      \
2354                                                                              \
2355   develop(bool, CITimeEach, false,                                           \
```



```
2356           "display timing information after each successful compilation")   \
2357                                                                              \
2358   develop(bool, CICountOSR, true,                                            \
2359           "use a separate counter when assigning ids to osr compilations")   \
2360                                                                              \
2361   develop(bool, CICompileNatives, true,                                      \
2362           "compile native methods if supported by the compiler")             \
2363                                                                              \
2364   develop_pd(bool, CICompileOSR,                                             \
2365           "compile on stack replacement methods if supported by the "        \
2366           "compiler")                                                        \
2367                                                                              \
2368   develop(bool, CIPrintMethodCodes, false,                                   \
2369           "print method bytecodes of the compiled code")                     \
2370                                                                              \
2371   develop(bool, CIPrintTypeFlow, false,                                      \
2372           "print the results of ciTypeFlow analysis")                        \
2373                                                                              \
2374   develop(bool, CITraceTypeFlow, false,                                      \
2375           "detailed per-bytecode tracing of ciTypeFlow analysis")            \
2376                                                                              \
2377   develop(intx, CICloneLoopTestLimit, 100,                                   \
2378           "size limit for blocks heuristically cloned in ciTypeFlow")        \
2379                                                                              \
2380   develop(intx, OSROnlyBCI, -1,                                              \
2381           "OSR only at this bci.  Negative values mean exclude that bci")    \
2382                                                                              \
2383   /* temp diagnostics */                                                     \
2384                                                                              \
2385   diagnostic(bool, TraceRedundantCompiles, false,                            \
2386           "Have compile broker print when a request already in the queue is"\
2387           " requested again")                                                \
2388                                                                              \
2389   diagnostic(bool, InitialCompileFast, false,                                \
2390           "Initial compile at CompLevel_fast_compile")                       \
2391                                                                              \
2392   diagnostic(bool, InitialCompileReallyFast, false,                          \
2393           "Initial compile at CompLevel_really_fast_compile (no profile)")   \
2394                                                                              \
2395   diagnostic(bool, FullProfileOnReInterpret, true,                           \
2396           "On re-interpret unc-trap compile next at CompLevel_fast_compile")\
2397                                                                              \
2398   /* compiler */                                                             \
2399                                                                              \
2400   product(intx, CICompilerCount, CI_COMPILER_COUNT,                          \
2401           "Number of compiler threads to run")                               \
2402                                                                              \
2403   product(intx, CompilationPolicyChoice, 0,                                  \
2404           "which compilation policy (0/1)")                                  \
2405                                                                              \
2406   develop(bool, UseStackBanging, true,                                       \
2407           "use stack banging for stack overflow checks (required for "       \
2408           "proper StackOverflow handling; disable only to measure cost "     \
2409           "of stackbanging)")                                                \
2410                                                                              \
2411   develop(bool, Use24BitFPMode, true,                                        \
2412           "Set 24-bit FPU mode on a per-compile basis ")                     \
2413                                                                              \
2414   develop(bool, Use24BitFP, true,                                            \
2415           "use FP instructions that produce 24-bit precise results")         \
2416                                                                              \
2417   develop(bool, UseStrictFP, true,                                           \
2418           "use strict fp if modifier strictfp is set")                       \
2419                                                                              \
2420   develop(bool, GenerateSynchronizationCode, true,                           \
2421           "generate locking/unlocking code for synchronized methods and "    \
```

```
2422          "monitors")                                                   \
2423                                                                        \
2424   develop(bool, GenerateCompilerNullChecks, true,                      \
2425          "Generate explicit null checks for loads/stores/calls")       \
2426                                                                        \
2427   develop(bool, GenerateRangeChecks, true,                             \
2428          "Generate range checks for array accesses")                   \
2429                                                                        \
2430   develop_pd(bool, ImplicitNullChecks,                                 \
2431          "generate code for implicit null checks")                     \
2432                                                                        \
2433   product(bool, PrintSafepointStatistics, false,                       \
2434          "print statistics about safepoint synchronization")           \
2435                                                                        \
2436   product(intx, PrintSafepointStatisticsCount, 300,                    \
2437          "total number of safepoint statistics collected "             \
2438          "before printing them out")                                   \
2439                                                                        \
2440   product(intx, PrintSafepointStatisticsTimeout,  -1,                  \
2441          "print safepoint statistics only when safepoint takes"        \
2442          " more than PrintSafepointSatisticsTimeout in millis")        \
2443                                                                        \
2444   product(bool, TraceSafepointCleanupTime, false,                      \
2445          "print the break down of clean up tasks performed during"     \
2446          " safepoint")                                                 \
2447                                                                        \
2448   develop(bool, InlineAccessors, true,                                 \
2449          "inline accessor methods (get/set)")                          \
2450                                                                        \
2451   product(bool, Inline, true,                                          \
2452          "enable inlining")                                            \
2453                                                                        \
2454   product(bool, ClipInlining, true,                                    \
2455          "clip inlining if aggregate method exceeds DesiredMethodLimit") \
2456                                                                        \
2457   develop(bool, UseCHA, true,                                          \
2458          "enable CHA")                                                 \
2459                                                                        \
2460   product(bool, UseTypeProfile, true,                                  \
2461          "Check interpreter profile for historically monomorphic calls") \
2462                                                                        \
2463   product(intx, TypeProfileMajorReceiverPercent, 90,                   \
2464          "% of major receiver type to all profiled receivers")         \
2465                                                                        \
2466   notproduct(bool, TimeCompiler, false,                                \
2467          "time the compiler")                                          \
2468                                                                        \
2469   notproduct(bool, TimeCompiler2, false,                               \
2470          "detailed time the compiler (requires +TimeCompiler)")        \
2471                                                                        \
2472   diagnostic(bool, PrintInlining, false,                               \
2473          "prints inlining optimizations")                              \
2474                                                                        \
2475   diagnostic(bool, PrintIntrinsics, false,                             \
2476          "prints attempted and successful inlining of intrinsics")     \
2477                                                                        \
2478   product(bool, UseCountLeadingZerosInstruction, false,                \
2479          "Use count leading zeros instruction")                        \
2480                                                                        \
2481   product(bool, UsePopCountInstruction, false,                         \
2482          "Use population count instruction")                           \
2483                                                                        \
2484   diagnostic(ccstrlist, DisableIntrinsic, "",                          \
2485          "do not expand intrinsics whose (internal) names appear here") \
2486                                                                        \
2487   develop(bool, StressReflectiveCode, false,                           \
```



```
2488          "Use inexact types at allocations, etc., to test reflection")  \
2489                                                                        \
2490   develop(bool, EagerInitialization, false,                            \
2491          "Eagerly initialize classes if possible")                     \
2492                                                                        \
2493   develop(bool, TraceMethodReplacement, false,                         \
2494          "Print when methods are replaced do to recompilation")        \
2495                                                                        \
2496   develop(bool, PrintMethodFlushing, false,                            \
2497          "print the nmethods being flushed")                           \
2498                                                                        \
2499   notproduct(bool, LogMultipleMutexLocking, false,                     \
2500          "log locking and unlocking of mutexes (only if multiple locks " \
2501          "are held)")                                                  \
2502                                                                        \
2503   develop(bool, UseRelocIndex, false,                                  \
2504          "use an index to speed random access to relocations")         \
2505                                                                        \
2506   develop(bool, StressCodeBuffers, false,                              \
2507          "Exercise code buffer expansion and other rare state changes") \
2508                                                                        \
2509   diagnostic(bool, DebugNonSafepoints, trueInDebug,                    \
2510          "Generate extra debugging info for non-safepoints in nmethods") \
2511                                                                        \
2512   diagnostic(bool, DebugInlinedCalls, true,                            \
2513          "If false, restricts profiled locations to the root method only") \
2514                                                                        \
2515   product(bool, PrintVMOptions, trueInDebug,                           \
2516          "Print flags that appeared on the command line")              \
2517                                                                        \
2518   product(bool, IgnoreUnrecognizedVMOptions, false,                    \
2519          "Ignore unrecognized VM options")                             \
2520                                                                        \
2521   product(bool, PrintCommandLineFlags, false,                          \
2522          "Print flags specified on command line or set by ergonomics") \
2523                                                                        \
2524   product(bool, PrintFlagsInitial, false,                              \
2525          "Print all VM flags before argument processing and exit VM")  \
2526                                                                        \
2527   product(bool, PrintFlagsFinal, false,                                \
2528          "Print all VM flags after argument and ergonomic processing") \
2529                                                                        \
2530   notproduct(bool, PrintFlagsWithComments, false,                      \
2531          "Print all VM flags with default values and descriptions and exit")\
2532                                                                        \
2533   diagnostic(bool, SerializeVMOutput, true,                            \
2534          "Use a mutex to serialize output to tty and hotspot.log")     \
2535                                                                        \
2536   diagnostic(bool, DisplayVMOutput, true,                              \
2537          "Display all VM output on the tty, independently of LogVMOutput") \
2538                                                                        \
2539   diagnostic(bool, LogVMOutput, trueInDebug,                           \
2540          "Save VM output to hotspot.log, or to LogFile")               \
2541                                                                        \
2542   diagnostic(ccstr, LogFile, NULL,                                     \
2543          "If LogVMOutput is on, save VM output to this file [hotspot.log]") \
2544                                                                        \
2545   product(ccstr, ErrorFile, NULL,                                      \
2546          "If an error occurs, save the error data to this file "       \
2547          "[default: ./hs_err_pid%p.log] (%p replaced with pid)")       \
2548                                                                        \
2549   product(bool, DisplayVMOutputToStderr, false,                        \
2550          "If DisplayVMOutput is true, display all VM output to stderr") \
2551                                                                        \
2552   product(bool, DisplayVMOutputToStdout, false,                        \
2553          "If DisplayVMOutput is true, display all VM output to stdout") \
```

```
2554                                                                              \
2555   product(bool, UseHeavyMonitors, false,                                     \
2556           "use heavyweight instead of lightweight Java monitors")            \
2557                                                                              \
2558   notproduct(bool, PrintSymbolTableSizeHistogram, false,                     \
2559           "print histogram of the symbol table")                             \
2560                                                                              \
2561   notproduct(bool, ExitVMOnVerifyError, false,                               \
2562           "standard exit from VM if bytecode verify error "                  \
2563           "(only in debug mode)")                                            \
2564                                                                              \
2565   notproduct(ccstr, AbortVMOnException, NULL,                                \
2566           "Call fatal if this exception is thrown.  Example: "               \
2567           "java -XX:AbortVMOnException=java.lang.NullPointerException Foo") \
2568                                                                              \
2569   notproduct(ccstr, AbortVMOnExceptionMessage, NULL,                         \
2570           "Call fatal if the exception pointed by AbortVMOnException "      \
2571           "has this message.")                                               \
2572                                                                              \
2573   develop(bool, DebugVtables, false,                                         \
2574           "add debugging code to vtable dispatch")                           \
2575                                                                              \
2576   develop(bool, PrintVtables, false,                                         \
2577           "print vtables when printing klass")                               \
2578                                                                              \
2579   notproduct(bool, PrintVtableStats, false,                                  \
2580           "print vtables stats at end of run")                               \
2581                                                                              \
2582   develop(bool, TraceCreateZombies, false,                                   \
2583           "trace creation of zombie nmethods")                               \
2584                                                                              \
2585   notproduct(bool, IgnoreLockingAssertions, false,                           \
2586           "disable locking assertions (for speed)")                          \
2587                                                                              \
2588   notproduct(bool, VerifyLoopOptimizations, false,                           \
2589           "verify major loop optimizations")                                 \
2590                                                                              \
2591   product(bool, RangeCheckElimination, true,                                 \
2592           "Split loop iterations to eliminate range checks")                 \
2593                                                                              \
2594   develop_pd(bool, UncommonNullCast,                                         \
2595           "track occurrences of null in casts; adjust compiler tactics")     \
2596                                                                              \
2597   develop(bool, TypeProfileCasts,  true,                                     \
2598           "treat casts like calls for purposes of type profiling")           \
2599                                                                              \
2600   develop(bool, MonomorphicArrayCheck, true,                                 \
2601           "Uncommon-trap array store checks that require full type check")   \
2602                                                                              \
2603   diagnostic(bool, ProfileDynamicTypes, true,                                \
2604           "do extra type profiling and use it more aggressively")            \
2605                                                                              \
2606   develop(bool, DelayCompilationDuringStartup, true,                         \
2607           "Delay invoking the compiler until main application class is "     \
2608           "loaded")                                                          \
2609                                                                              \
2610   develop(bool, CompileTheWorld, false,                                      \
2611           "Compile all methods in all classes in bootstrap class path "      \
2612           "(stress test)")                                                   \
2613                                                                              \
2614   develop(bool, CompileTheWorldPreloadClasses, true,                         \
2615           "Preload all classes used by a class before start loading")        \
2616                                                                              \
2617   notproduct(intx, CompileTheWorldSafepointInterval, 100,                    \
2618           "Force a safepoint every n compiles so sweeper can keep up")       \
2619                                                                              \
```



```
2620   develop(bool, TraceIterativeGVN, false,                                    \
2621           "Print progress during Iterative Global Value Numbering")          \
2622                                                                              \
2623   develop(bool, FillDelaySlots, true,                                        \
2624           "Fill delay slots (on SPARC only)")                                \
2625                                                                              \
2626   develop(bool, VerifyIterativeGVN, false,                                   \
2627           "Verify Def-Use modifications during sparse Iterative Global "     \
2628           "Value Numbering")                                                 \
2629                                                                              \
2630   notproduct(bool, TracePhaseCCP, false,                                     \
2631           "Print progress during Conditional Constant Propagation")          \
2632                                                                              \
2633   develop(bool, TimeLivenessAnalysis, false,                                 \
2634           "Time computation of bytecode liveness analysis")                  \
2635                                                                              \
2636   develop(bool, TraceLivenessGen, false,                                     \
2637           "Trace the generation of liveness analysis information")           \
2638                                                                              \
2639   notproduct(bool, TraceLivenessQuery, false,                                \
2640           "Trace queries of liveness analysis information")                  \
2641                                                                              \
2642   notproduct(bool, CollectIndexSetStatistics, false,                         \
2643           "Collect information about IndexSets")                             \
2644                                                                              \
2645   develop(bool, PrintDominators, false,                                      \
2646           "Print out dominator trees for GVN")                               \
2647                                                                              \
2648   develop(bool, UseLoopSafepoints, true,                                     \
2649           "Generate Safepoint nodes in every loop")                          \
2650                                                                              \
2651   notproduct(bool, TraceCISCSpill, false,                                    \
2652           "Trace allocators use of cisc spillable instructions")             \
2653                                                                              \
2654   notproduct(bool, TraceSpilling, false,                                     \
2655           "Trace spilling")                                                  \
2656                                                                              \
2657   product(bool, SplitIfBlocks, true,                                         \
2658           "Clone compares and control flow through merge points to fold "    \
2659           "some branches")                                                   \
2660                                                                              \
2661   develop(intx, FastAllocateSizeLimit, 128*K,                                \
2662           /* Note:  This value is zero mod 1<<13 for a cheap sparc set. */   \
2663           "Inline allocations larger than this in doublewords must go slow")\
2664                                                                              \
2665   product(bool, AggressiveOpts, false,                                       \
2666           "Enable aggressive optimizations - see arguments.cpp")             \
2667                                                                              \
2668   product(bool, UseStringCache, false,                                       \
2669           "Enable String cache capabilities on String.java")                 \
2670                                                                              \
2671   /* statistics */                                                           \
2672   develop(bool, CountCompiledCalls, false,                                   \
2673           "counts method invocations")                                       \
2674                                                                              \
2675   notproduct(bool, CountRuntimeCalls, false,                                 \
2676           "counts VM runtime calls")                                         \
2677                                                                              \
2678   develop(bool, CountJNICalls, false,                                        \
2679           "counts jni method invocations")                                   \
2680                                                                              \
2681   notproduct(bool, CountJVMCalls, false,                                     \
2682           "counts jvm method invocations")                                   \
2683                                                                              \
2684   notproduct(bool, CountRemovableExceptions, false,                          \
2685           "count exceptions that could be replaced by branches due to "      \
```

```
2686           "inlining")                                                     \
2687                                                                           \
2688   notproduct(bool, ICMissHistogram, false,                                \
2689           "produce histogram of IC misses")                               \
2690                                                                           \
2691   notproduct(bool, PrintClassStatistics, false,                           \
2692           "prints class statistics at end of run")                        \
2693                                                                           \
2694   notproduct(bool, PrintMethodStatistics, false,                          \
2695           "prints method statistics at end of run")                       \
2696                                                                           \
2697   /* interpreter */                                                       \
2698   develop(bool, ClearInterpreterLocals, false,                            \
2699           "Always clear local variables of interpreter activations upon "  \
2700           "entry")                                                        \
2701                                                                           \
2702   product_pd(bool, RewriteBytecodes,                                      \
2703           "Allow rewriting of bytecodes (bytecodes are not immutable)")    \
2704                                                                           \
2705   product_pd(bool, RewriteFrequentPairs,                                  \
2706           "Rewrite frequently used bytecode pairs into a single bytecode") \
2707                                                                           \
2708   diagnostic(bool, PrintInterpreter, false,                               \
2709           "Prints the generated interpreter code")                        \
2710                                                                           \
2711   product(bool, UseInterpreter, true,                                     \
2712           "Use interpreter for non-compiled methods")                     \
2713                                                                           \
2714   develop(bool, UseFastSignatureHandlers, true,                           \
2715           "Use fast signature handlers for native calls")                 \
2716                                                                           \
2717   develop(bool, UseV8InstrsOnly, false,                                   \
2718           "Use SPARC-V8 Compliant instruction subset")                    \
2719                                                                           \
2720   product(bool, UseNiagaraInstrs, false,                                  \
2721           "Use Niagara-efficient instruction subset")                     \
2722                                                                           \
2723   develop(bool, UseCASForSwap, false,                                     \
2724           "Do not use swap instructions, but only CAS (in a loop) on SPARC")\
2725                                                                           \
2726   product(bool, UseLoopCounter, true,                                     \
2727           "Increment invocation counter on backward branch")              \
2728                                                                           \
2729   product(bool, UseFastEmptyMethods, true,                                \
2730           "Use fast method entry code for empty methods")                 \
2731                                                                           \
2732   product(bool, UseFastAccessorMethods, true,                             \
2733           "Use fast method entry code for accessor methods")              \
2734                                                                           \
2735   product_pd(bool, UseOnStackReplacement,                                 \
2736            "Use on stack replacement, calls runtime if invoc. counter "   \
2737            "overflows in loop")                                           \
2738                                                                           \
2739   notproduct(bool, TraceOnStackReplacement, false,                        \
2740           "Trace on stack replacement")                                   \
2741                                                                           \
2742   develop(bool, PoisonOSREntry, true,                                     \
2743            "Detect abnormal calls to OSR code")                           \
2744                                                                           \
2745   product_pd(bool, PreferInterpreterNativeStubs,                          \
2746           "Use always interpreter stubs for native methods invoked via "  \
2747           "interpreter")                                                  \
2748                                                                           \
2749   develop(bool, CountBytecodes, false,                                    \
2750           "Count number of bytecodes executed")                           \
2751                                                                           \
```



```
2752   develop(bool, PrintBytecodeHistogram, false,                            \
2753           "Print histogram of the executed bytecodes")                    \
2754                                                                           \
2755   develop(bool, PrintBytecodePairHistogram, false,                        \
2756           "Print histogram of the executed bytecode pairs")               \
2757                                                                           \
2758   diagnostic(bool, PrintSignatureHandlers, false,                         \
2759           "Print code generated for native method signature handlers")    \
2760                                                                           \
2761   develop(bool, VerifyOops, false,                                        \
2762           "Do plausibility checks for oops")                              \
2763                                                                           \
2764   develop(bool, CheckUnhandledOops, false,                                \
2765           "Check for unhandled oops in VM code")                          \
2766                                                                           \
2767   develop(bool, VerifyJNIFields, trueInDebug,                             \
2768           "Verify jfieldIDs for instance fields")                         \
2769                                                                           \
2770   notproduct(bool, VerifyJNIEnvThread, false,                             \
2771           "Verify JNIEnv.thread == Thread::current() when entering VM "   \
2772           "from JNI")                                                     \
2773                                                                           \
2774   develop(bool, VerifyFPU, false,                                         \
2775           "Verify FPU state (check for NaN's, etc.)")                     \
2776                                                                           \
2777   develop(bool, VerifyThread, false,                                      \
2778           "Watch the thread register for corruption (SPARC only)")        \
2779                                                                           \
2780   develop(bool, VerifyActivationFrameSize, false,                         \
2781           "Verify that activation frame didn't become smaller than its "  \
2782           "minimal size")                                                 \
2783                                                                           \
2784   develop(bool, TraceFrequencyInlining, false,                            \
2785           "Trace frequency based inlining")                               \
2786                                                                           \
2787   notproduct(bool, TraceTypeProfile, false,                               \
2788           "Trace type profile")                                           \
2789                                                                           \
2790   develop_pd(bool, InlineIntrinsics,                                      \
2791            "Inline intrinsics that can be statically resolved")           \
2792                                                                           \
2793   product_pd(bool, ProfileInterpreter,                                    \
2794            "Profile at the bytecode level during interpretation")         \
2795                                                                           \
2796   develop_pd(bool, ProfileTraps,                                          \
2797           "Profile deoptimization traps at the bytecode level")           \
2798                                                                           \
2799   product(intx, ProfileMaturityPercentage, 20,                            \
2800           "number of method invocations/branches (expressed as % of "     \
2801           "CompileThreshold) before using the method's profile")          \
2802                                                                           \
2803   develop(bool, PrintMethodData, false,                                   \
2804            "Print the results of +ProfileInterpreter at end of run")      \
2805                                                                           \
2806   develop(bool, VerifyDataPointer, trueInDebug,                           \
2807           "Verify the method data pointer during interpreter profiling")  \
2808                                                                           \
2809   develop(bool, VerifyCompiledCode, false,                                \
2810           "Include miscellaneous runtime verifications in nmethod code; " \
2811           "default off because it disturbs nmethod size heuristics")      \
2812                                                                           \
2813   notproduct(bool, CrashGCForDumpingJavaThread, false,                    \
2814           "Manually make GC thread crash then dump java stack trace;  "   \
2815           "Test only")                                                    \
2816                                                                           \
2817   /* compilation */                                                       \
```

```
2818   product(bool, UseCompiler, true,                                          \
2819           "use compilation")                                                \
2820                                                                             \
2821   develop(bool, TraceCompilationPolicy, false,                              \
2822           "Trace compilation policy")                                       \
2823                                                                             \
2824   develop(bool, TimeCompilationPolicy, false,                               \
2825           "Time the compilation policy")                                    \
2826                                                                             \
2827   product(bool, UseCounterDecay, true,                                      \
2828            "adjust recompilation counters")                                 \
2829                                                                             \
2830   develop(intx, CounterHalfLifeTime,    30,                                 \
2831           "half-life time of invocation counters (in secs)")                \
2832                                                                             \
2833   develop(intx, CounterDecayMinIntervalLength,   500,                       \
2834           "Min. ms. between invocation of CounterDecay")                    \
2835                                                                             \
2836   product(bool, AlwaysCompileLoopMethods, false,                            \
2837           "when using recompilation, never interpret methods "             \
2838           "containing loops")                                               \
2839                                                                             \
2840   product(bool, DontCompileHugeMethods, true,                               \
2841           "don't compile methods > HugeMethodLimit")                        \
2842                                                                             \
2843   /* Bytecode escape analysis estimation. */                                \
2844   product(bool, EstimateArgEscape, true,                                    \
2845           "Analyze bytecodes to estimate escape state of arguments")        \
2846                                                                             \
2847   product(intx, BCEATraceLevel, 0,                                          \
2848           "How much tracing to do of bytecode escape analysis estimates")   \
2849                                                                             \
2850   product(intx, MaxBCEAEstimateLevel, 5,                                    \
2851           "Maximum number of nested calls that are analyzed by BC EA.")     \
2852                                                                             \
2853   product(intx, MaxBCEAEstimateSize, 150,                                   \
2854           "Maximum bytecode size of a method to be analyzed by BC EA.")     \
2855                                                                             \
2856   product(intx,  AllocatePrefetchStyle, 1,                                  \
2857           "0 = no prefetch, "                                               \
2858           "1 = prefetch instructions for each allocation, "                 \
2859           "2 = use TLAB watermark to gate allocation prefetch, "            \
2860           "3 = use BIS instruction on Sparc for allocation prefetch")       \
2861                                                                             \
2862   product(intx,  AllocatePrefetchDistance, -1,                              \
2863           "Distance to prefetch ahead of allocation pointer")               \
2864                                                                             \
2865   product(intx,  AllocatePrefetchLines, 1,                                  \
2866           "Number of lines to prefetch ahead of allocation pointer")        \
2867                                                                             \
2868   product(intx,  AllocatePrefetchStepSize, 16,                              \
2869           "Step size in bytes of sequential prefetch instructions")         \
2870                                                                             \
2871   product(intx,  AllocatePrefetchInstr, 0,                                  \
2872           "Prefetch instruction to prefetch ahead of allocation pointer")   \
2873                                                                             \
2874   product(intx,  ReadPrefetchInstr, 0,                                      \
2875           "Prefetch instruction to prefetch ahead")                         \
2876                                                                             \
2877   /* deoptimization */                                                      \
2878   develop(bool, TraceDeoptimization, false,                                 \
2879           "Trace deoptimization")                                           \
2880                                                                             \
2881   develop(bool, DebugDeoptimization, false,                                 \
2882           "Tracing various information while debugging deoptimization")     \
2883                                                                             \
```



```
2884   product(intx, SelfDestructTimer, 0,                                       \
2885           "Will cause VM to terminate after a given time (in minutes) "     \
2886           "(0 means off)")                                                  \
2887                                                                             \
2888   product(intx, MaxJavaStackTraceDepth, 1024,                               \
2889           "Max. no. of lines in the stack trace for Java exceptions "       \
2890           "(0 means all)")                                                  \
2891                                                                             \
2892   NOT_EMBEDDED(develop(intx, GuaranteedSafepointInterval, 1000,             \
2893           "Guarantee a safepoint (at least) every so many milliseconds "    \
2894           "(0 means none)"))                                                \
2895                                                                             \
2896   EMBEDDED_ONLY(product(intx, GuaranteedSafepointInterval, 0,               \
2897           "Guarantee a safepoint (at least) every so many milliseconds "    \
2898           "(0 means none)"))                                                \
2899                                                                             \
2900   product(intx, SafepointTimeoutDelay, 10000,                               \
2901           "Delay in milliseconds for option SafepointTimeout")              \
2902                                                                             \
2903   product(intx, NmethodSweepFraction, 4,                                    \
2904           "Number of invocations of sweeper to cover all nmethods")         \
2905                                                                             \
2906   product(intx, NmethodSweepCheckInterval, 5,                               \
2907           "Compilers wake up every n seconds to possibly sweep nmethods")   \
2908                                                                             \
2909   notproduct(intx, MemProfilingInterval, 500,                               \
2910           "Time between each invocation of the MemProfiler")                \
2911                                                                             \
2912   develop(intx, MallocCatchPtr, -1,                                         \
2913           "Hit breakpoint when mallocing/freeing this pointer")             \
2914                                                                             \
2915   notproduct(intx, AssertRepeat, 1,                                         \
2916           "number of times to evaluate expression in assert "               \
2917           "(to estimate overhead); only works with -DUSE_REPEATED_ASSERTS") \
2918                                                                             \
2919   notproduct(ccstrlist, SuppressErrorAt, "",                                \
2920           "List of assertions (file:line) to muzzle")                       \
2921                                                                             \
2922   notproduct(uintx, HandleAllocationLimit, 1024,                            \
2923           "Threshold for HandleMark allocation when +TraceHandleAllocation "\
2924           "is used")                                                        \
2925                                                                             \
2926   develop(uintx, TotalHandleAllocationLimit, 1024,                          \
2927           "Threshold for total handle allocation when "                     \
2928           "+TraceHandleAllocation is used")                                 \
2929                                                                             \
2930   develop(intx, StackPrintLimit, 100,                                       \
2931           "number of stack frames to print in VM-level stack dump")         \
2932                                                                             \
2933   notproduct(intx, MaxElementPrintSize, 256,                                \
2934           "maximum number of elements to print")                            \
2935                                                                             \
2936   notproduct(intx, MaxSubklassPrintSize, 4,                                 \
2937           "maximum number of subklasses to print when printing klass")      \
2938                                                                             \
2939   product(intx, MaxInlineLevel, 9,                                          \
2940           "maximum number of nested calls that are inlined")                \
2941                                                                             \
2942   product(intx, MaxRecursiveInlineLevel, 1,                                 \
2943           "maximum number of nested recursive calls that are inlined")      \
2944                                                                             \
2945   product_pd(intx, InlineSmallCode,                                         \
2946           "Only inline already compiled methods if their code size is "     \
2947           "less than this")                                                 \
2948                                                                             \
2949   product(intx, MaxInlineSize, 35,                                          \
```

```
2950          "maximum bytecode size of a method to be inlined")                \
2951                                                                            \
2952   product_pd(intx, FreqInlineSize,                                         \
2953          "maximum bytecode size of a frequent method to be inlined")       \
2954                                                                            \
2955   product(intx, MaxTrivialSize, 6,                                         \
2956          "maximum bytecode size of a trivial method to be inlined")        \
2957                                                                            \
2958   product(intx, MinInliningThreshold, 250,                                 \
2959          "min. invocation count a method needs to have to be inlined")     \
2960                                                                            \
2961   develop(intx, AlignEntryCode, 4,                                         \
2962          "aligns entry code to specified value (in bytes)")                \
2963                                                                            \
2964   develop(intx, MethodHistogramCutoff, 100,                                \
2965          "cutoff value for method invoc. histogram (+CountCalls)")         \
2966                                                                            \
2967   develop(intx, ProfilerNumberOfInterpretedMethods, 25,                    \
2968          "# of interpreted methods to show in profile")                    \
2969                                                                            \
2970   develop(intx, ProfilerNumberOfCompiledMethods, 25,                       \
2971          "# of compiled methods to show in profile")                       \
2972                                                                            \
2973   develop(intx, ProfilerNumberOfStubMethods, 25,                           \
2974          "# of stub methods to show in profile")                           \
2975                                                                            \
2976   develop(intx, ProfilerNumberOfRuntimeStubNodes, 25,                      \
2977          "# of runtime stub nodes to show in profile")                     \
2978                                                                            \
2979   product(intx, ProfileIntervalsTicks, 100,                                \
2980          "# of ticks between printing of interval profile "                \
2981          "(+ProfileIntervals)")                                            \
2982                                                                            \
2983   notproduct(intx, ScavengeALotInterval,     1,                            \
2984          "Interval between which scavenge will occur with +ScavengeALot")  \
2985                                                                            \
2986   notproduct(intx, FullGCALotInterval,     1,                              \
2987          "Interval between which full gc will occur with +FullGCALot")     \
2988                                                                            \
2989   notproduct(intx, FullGCALotStart,     0,                                 \
2990          "For which invocation to start FullGCAlot")                       \
2991                                                                            \
2992   notproduct(intx, FullGCALotDummies,  32*K,                               \
2993          "Dummy object allocated with +FullGCALot, forcing all objects "   \
2994          "to move")                                                        \
2995                                                                            \
2996   develop(intx, DontYieldALotInterval,    10,                              \
2997          "Interval between which yields will be dropped (milliseconds)")   \
2998                                                                            \
2999   develop(intx, MinSleepInterval,     1,                                   \
3000          "Minimum sleep() interval (milliseconds) when "                   \
3001          "ConvertSleepToYield is off (used for SOLARIS)")                  \
3002                                                                            \
3003   product(intx, EventLogLength,  2000,                                     \
3004          "maximum nof events in event log")                                \
3005                                                                            \
3006   develop(intx, ProfilerPCTickThreshold,    15,                            \
3007          "Number of ticks in a PC buckets to be a hotspot")                \
3008                                                                            \
3009   notproduct(intx, DeoptimizeALotInterval,     5,                          \
3010          "Number of exits until DeoptimizeALot kicks in")                  \
3011                                                                            \
3012   notproduct(intx, ZombieALotInterval,     5,                              \
3013          "Number of exits until ZombieALot kicks in")                      \
3014                                                                            \
3015   develop(bool, StressNonEntrant, false,                                   \
```



```
3016          "Mark nmethods non-entrant at registration")                      \
3017                                                                           \
3018   diagnostic(intx, MallocVerifyInterval,     0,                          \
3019          "if non-zero, verify C heap after every N calls to "             \
3020          "malloc/realloc/free")                                           \
3021                                                                           \
3022   diagnostic(intx, MallocVerifyStart,     0,                             \
3023          "if non-zero, start verifying C heap after Nth call to "         \
3024          "malloc/realloc/free")                                           \
3025                                                                           \
3026   product(intx, TypeProfileWidth,     2,                                \
3027          "number of receiver types to record in call/cast profile")       \
3028                                                                           \
3029   develop(intx, BciProfileWidth,      2,                                 \
3030          "number of return bci's to record in ret profile")               \
3031                                                                           \
3032   product(intx, PerMethodRecompilationCutoff, 400,                        \
3033          "After recompiling N times, stay in the interpreter (-1=>'Inf')") \
3034                                                                           \
3035   product(intx, PerBytecodeRecompilationCutoff, 200,                      \
3036          "Per-BCI limit on repeated recompilation (-1=>'Inf')")           \
3037                                                                           \
3038   product(intx, PerMethodTrapLimit,  100,                                 \
3039          "Limit on traps (of one kind) in a method (includes inlines)")   \
3040                                                                           \
3041   product(intx, PerBytecodeTrapLimit,  4,                                 \
3042          "Limit on traps (of one kind) at a particular BCI")              \
3043                                                                           \
3044   develop(intx, FreqCountInvocations,  1,                                 \
3045          "Scaling factor for branch frequencies (deprecated)")            \
3046                                                                           \
3047   develop(intx, InlineFrequencyRatio,    20,                              \
3048          "Ratio of call site execution to caller method invocation")      \
3049                                                                           \
3050   develop_pd(intx, InlineFrequencyCount,                                  \
3051          "Count of call site execution necessary to trigger frequent "    \
3052          "inlining")                                                      \
3053                                                                           \
3054   develop(intx, InlineThrowCount,    50,                                  \
3055          "Force inlining of interpreted methods that throw this often")   \
3056                                                                           \
3057   develop(intx, InlineThrowMaxSize,   200,                                \
3058          "Force inlining of throwing methods smaller than this")          \
3059                                                                           \
3060   product(intx, AliasLevel,     3,                                        \
3061          "0 for no aliasing, 1 for oop/field/static/array split, "        \
3062          "2 for class split, 3 for unique instances")                     \
3063                                                                           \
3064   develop(bool, VerifyAliases, false,                                     \
3065          "perform extra checks on the results of alias analysis")         \
3066                                                                           \
3067   develop(intx, ProfilerNodeSize,  1024,                                  \
3068          "Size in K to allocate for the Profile Nodes of each thread")    \
3069                                                                           \
3070   develop(intx, V8AtomicOperationUnderLockSpinCount,    50,               \
3071          "Number of times to spin wait on a v8 atomic operation lock")    \
3072                                                                           \
3073   product(intx, ReadSpinIterations,   100,                                \
3074          "Number of read attempts before a yield (spin inner loop)")      \
3075                                                                           \
3076   product_pd(intx, PreInflateSpin,                                        \
3077          "Number of times to spin wait before inflation")                 \
3078                                                                           \
3079   product(intx, PreBlockSpin,    10,                                      \
3080          "Number of times to spin in an inflated lock before going to "   \
3081          "an OS lock")                                                    \
```

```
3082                                                                            \
3083   /* gc parameters */                                                      \
3084   product(uintx, InitialHeapSize, 0,                                       \
3085           "Initial heap size (in bytes); zero means OldSize + NewSize")    \
3086                                                                            \
3087   product(uintx, MaxHeapSize, ScaleForWordSize(96*M),                      \
3088           "Maximum heap size (in bytes)")                                  \
3089                                                                            \
3090   product(uintx, OldSize, ScaleForWordSize(4*M),                           \
3091           "Initial tenured generation size (in bytes)")                    \
3092                                                                            \
3093   product(uintx, NewSize, ScaleForWordSize(1*M),                           \
3094           "Initial new generation size (in bytes)")                        \
3095                                                                            \
3096   product(uintx, MaxNewSize, max_uintx,                                    \
3097           "Maximum new generation size (in bytes), max_uintx means set "   \
3098           "ergonomically")                                                 \
3099                                                                            \
3100   product(uintx, PretenureSizeThreshold, 0,                                \
3101           "Maximum size in bytes of objects allocated in DefNew "          \
3102           "generation; zero means no maximum")                             \
3103                                                                            \
3104   product(uintx, TLABSize, 0,                                              \
3105           "Starting TLAB size (in bytes); zero means set ergonomically")   \
3106                                                                            \
3107   product(uintx, MinTLABSize, 2*K,                                         \
3108           "Minimum allowed TLAB size (in bytes)")                          \
3109                                                                            \
3110   product(uintx, TLABAllocationWeight, 35,                                 \
3111           "Allocation averaging weight")                                   \
3112                                                                            \
3113   product(uintx, TLABWasteTargetPercent, 1,                                \
3114           "Percentage of Eden that can be wasted")                         \
3115                                                                            \
3116   product(uintx, TLABRefillWasteFraction,    64,                           \
3117           "Max TLAB waste at a refill (internal fragmentation)")           \
3118                                                                            \
3119   product(uintx, TLABWasteIncrement,    4,                                 \
3120           "Increment allowed waste at slow allocation")                    \
3121                                                                            \
3122   product(intx, SurvivorRatio, 8,                                          \
3123           "Ratio of eden/survivor space size")                             \
3124                                                                            \
3125   product(intx, NewRatio, 2,                                               \
3126           "Ratio of new/old generation sizes")                             \
3127                                                                            \
3128   product_pd(uintx, NewSizeThreadIncrease,                                 \
3129           "Additional size added to desired new generation size per "      \
3130           "non-daemon thread (in bytes)")                                  \
3131                                                                            \
3132   product_pd(uintx, PermSize,                                              \
3133           "Initial size of permanent generation (in bytes)")               \
3134                                                                            \
3135   product_pd(uintx, MaxPermSize,                                           \
3136           "Maximum size of permanent generation (in bytes)")               \
3137                                                                            \
3138   product(uintx, MinHeapFreeRatio,    40,                                  \
3139           "Min percentage of heap free after GC to avoid expansion")       \
3140                                                                            \
3141   product(uintx, MaxHeapFreeRatio,    70,                                  \
3142           "Max percentage of heap free after GC to avoid shrinking")       \
3143                                                                            \
3144   product(intx, SoftRefLRUPolicyMSPerMB, 1000,                             \
3145           "Number of milliseconds per MB of free space in the heap")       \
3146                                                                            \
3147   product(uintx, MinHeapDeltaBytes, ScaleForWordSize(128*K),               \
```



```
3148           "Min change in heap space due to GC (in bytes)")                 \
3149                                                                            \
3150   product(uintx, MinPermHeapExpansion, ScaleForWordSize(256*K),            \
3151           "Min expansion of permanent heap (in bytes)")                    \
3152                                                                            \
3153   product(uintx, MaxPermHeapExpansion, ScaleForWordSize(4*M),              \
3154           "Max expansion of permanent heap without full GC (in bytes)")    \
3155                                                                            \
3156   product(intx, QueuedAllocationWarningCount, 0,                           \
3157           "Number of times an allocation that queues behind a GC "         \
3158           "will retry before printing a warning")                          \
3159                                                                            \
3160   diagnostic(uintx, VerifyGCStartAt,   0,                                  \
3161           "GC invoke count where +VerifyBefore/AfterGC kicks in")          \
3162                                                                            \
3163   diagnostic(intx, VerifyGCLevel,     0,                                   \
3164           "Generation level at which to start +VerifyBefore/AfterGC")      \
3165                                                                            \
3166   develop(uintx, ExitAfterGCNum,   0,                                      \
3167           "If non-zero, exit after this GC.")                              \
3168                                                                            \
3169   product(intx, MaxTenuringThreshold,    15,                               \
3170           "Maximum value for tenuring threshold")                          \
3171                                                                            \
3172   product(intx, InitialTenuringThreshold,     7,                           \
3173           "Initial value for tenuring threshold")                          \
3174                                                                            \
3175   product(intx, TargetSurvivorRatio,    50,                                \
3176           "Desired percentage of survivor space used after scavenge")      \
3177                                                                            \
3178   product(uintx, MarkSweepDeadRatio,     5,                                \
3179           "Percentage (0-100) of the old gen allowed as dead wood."        \
3180           "Serial mark sweep treats this as both the min and max value."   \
3181           "CMS uses this value only if it falls back to mark sweep."       \
3182           "Par compact uses a variable scale based on the density of the"  \
3183           "generation and treats this as the max value when the heap is"   \
3184           "either completely full or completely empty.  Par compact also"  \
3185           "has a smaller default value; see arguments.cpp.")               \
3186                                                                            \
3187   product(uintx, PermMarkSweepDeadRatio,    20,                            \
3188           "Percentage (0-100) of the perm gen allowed as dead wood."       \
3189           "See MarkSweepDeadRatio for collector-specific comments.")       \
3190                                                                            \
3191   product(intx, MarkSweepAlwaysCompactCount,     4,                        \
3192           "How often should we fully compact the heap (ignoring the dead " \
3193           "space parameters)")                                             \
3194                                                                            \
3195   product(intx, PrintCMSStatistics, 0,                                     \
3196           "Statistics for CMS")                                            \
3197                                                                            \
3198   product(bool, PrintCMSInitiationStatistics, false,                       \
3199           "Statistics for initiating a CMS collection")                    \
3200                                                                            \
3201   product(intx, PrintFLSStatistics, 0,                                     \
3202           "Statistics for CMS' FreeListSpace")                             \
3203                                                                            \
3204   product(intx, PrintFLSCensus, 0,                                         \
3205           "Census for CMS' FreeListSpace")                                 \
3206                                                                            \
3207   develop(uintx, GCExpandToAllocateDelayMillis, 0,                         \
3208           "Delay in ms between expansion and allocation")                  \
3209                                                                            \
3210   product(intx, DeferThrSuspendLoopCount,     4000,                        \
3211           "(Unstable) Number of times to iterate in safepoint loop "       \
3212           " before blocking VM threads ")                                  \
3213                                                                            \
```

```
3214   product(intx, DeferPollingPageLoopCount,     -1,                          \
3215           "(Unsafe,Unstable) Number of iterations in safepoint loop "       \
3216           "before changing safepoint polling page to RO ")                  \
3217                                                                             \
3218   product(intx, SafepointSpinBeforeYield, 2000,  "(Unstable)")              \
3219                                                                             \
3220   product(bool, PSChunkLargeArrays, true,                                   \
3221           "true: process large arrays in chunks")                           \
3222                                                                             \
3223   product(uintx, GCDrainStackTargetSize, 64,                                \
3224           "how many entries we'll try to leave on the stack during "        \
3225           "parallel GC")                                                    \
3226                                                                             \
3227   /* stack parameters */                                                    \
3228   product_pd(intx, StackYellowPages,                                        \
3229           "Number of yellow zone (recoverable overflows) pages")            \
3230                                                                             \
3231   product_pd(intx, StackRedPages,                                           \
3232           "Number of red zone (unrecoverable overflows) pages")             \
3233                                                                             \
3234   product_pd(intx, StackShadowPages,                                        \
3235           "Number of shadow zone (for overflow checking) pages"             \
3236           " this should exceed the depth of the VM and native call stack")  \
3237                                                                             \
3238   product_pd(intx, ThreadStackSize,                                         \
3239           "Thread Stack Size (in Kbytes)")                                  \
3240                                                                             \
3241   product_pd(intx, VMThreadStackSize,                                       \
3242           "Non-Java Thread Stack Size (in Kbytes)")                         \
3243                                                                             \
3244   product_pd(intx, CompilerThreadStackSize,                                 \
3245           "Compiler Thread Stack Size (in Kbytes)")                         \
3246                                                                             \
3247   develop_pd(uintx, JVMInvokeMethodSlack,                                   \
3248           "Stack space (bytes) required for JVM_InvokeMethod to complete")  \
3249                                                                             \
3250   product(uintx, ThreadSafetyMargin, 50*M,                                  \
3251           "Thread safety margin is used on fixed-stack LinuxThreads (on "   \
3252           "Linux/x86 only) to prevent heap-stack collision. Set to 0 to "   \
3253           "disable this feature")                                           \
3254                                                                             \
3255   /* code cache parameters */                                               \
3256   develop(uintx, CodeCacheSegmentSize, 64,                                  \
3257           "Code cache segment size (in bytes) - smallest unit of "          \
3258           "allocation")                                                     \
3259                                                                             \
3260   develop_pd(intx, CodeEntryAlignment,                                      \
3261           "Code entry alignment for generated code (in bytes)")             \
3262                                                                             \
3263   product_pd(intx, OptoLoopAlignment,                                       \
3264           "Align inner loops to zero relative to this modulus")             \
3265                                                                             \
3266   product_pd(uintx, InitialCodeCacheSize,                                   \
3267           "Initial code cache size (in bytes)")                             \
3268                                                                             \
3269   product_pd(uintx, ReservedCodeCacheSize,                                  \
3270           "Reserved code cache size (in bytes) - maximum code cache size")  \
3271                                                                             \
3272   product(uintx, CodeCacheMinimumFreeSpace, 500*K,                          \
3273           "When less than X space left, we stop compiling.")                \
3274                                                                             \
3275   product_pd(uintx, CodeCacheExpansionSize,                                 \
3276           "Code cache expansion size (in bytes)")                           \
3277                                                                             \
3278   develop_pd(uintx, CodeCacheMinBlockLength,                                \
3279           "Minimum number of segments in a code cache block.")              \
```



```
3280                                                                             \
3281   notproduct(bool, ExitOnFullCodeCache, false,                              \
3282           "Exit the VM if we fill the code cache.")                         \
3283                                                                             \
3284   product(bool, UseCodeCacheFlushing, false,                                \
3285           "Attempt to clean the code cache before shutting off compiler")   \
3286                                                                             \
3287   product(intx,  MinCodeCacheFlushingInterval, 30,                          \
3288           "Min number of seconds between code cache cleaning sessions")     \
3289                                                                             \
3290   product(uintx,  CodeCacheFlushingMinimumFreeSpace, 1500*K,                \
3291           "When less than X space left, start code cache cleaning")         \
3292                                                                             \
3293   /* interpreter debugging */                                               \
3294   develop(intx, BinarySwitchThreshold, 5,                                   \
3295           "Minimal number of lookupswitch entries for rewriting to binary " \
3296           "switch")                                                         \
3297                                                                             \
3298   develop(intx, StopInterpreterAt, 0,                                       \
3299           "Stops interpreter execution at specified bytecode number")       \
3300                                                                             \
3301   develop(intx, TraceBytecodesAt, 0,                                        \
3302           "Traces bytecodes starting with specified bytecode number")       \
3303                                                                             \
3304   /* compiler interface */                                                  \
3305   develop(intx, CIStart, 0,                                                 \
3306           "the id of the first compilation to permit")                      \
3307                                                                             \
3308   develop(intx, CIStop,    -1,                                              \
3309           "the id of the last compilation to permit")                       \
3310                                                                             \
3311   develop(intx, CIStartOSR,     0,                                          \
3312           "the id of the first osr compilation to permit "                  \
3313           "(CICountOSR must be on)")                                        \
3314                                                                             \
3315   develop(intx, CIStopOSR,    -1,                                           \
3316           "the id of the last osr compilation to permit "                   \
3317           "(CICountOSR must be on)")                                        \
3318                                                                             \
3319   develop(intx, CIBreakAtOSR,    -1,                                        \
3320           "id of osr compilation to break at")                              \
3321                                                                             \
3322   develop(intx, CIBreakAt,    -1,                                           \
3323           "id of compilation to break at")                                  \
3324                                                                             \
3325   product(ccstrlist, CompileOnly, "",                                       \
3326           "List of methods (pkg/class.name) to restrict compilation to")    \
3327                                                                             \
3328   product(ccstr, CompileCommandFile, NULL,                                  \
3329           "Read compiler commands from this file [.hotspot_compiler]")      \
3330                                                                             \
3331   product(ccstrlist, CompileCommand, "",                                    \
3332           "Prepend to .hotspot_compiler; e.g. log,java/lang/String.<init>") \
3333                                                                             \
3334   product(bool, CICompilerCountPerCPU, false,                               \
3335           "1 compiler thread for log(N CPUs)")                              \
3336                                                                             \
3337   develop(intx, CIFireOOMAt,    -1,                                        \
3338           "Fire OutOfMemoryErrors throughout CI for testing the compiler "  \
3339           "(non-negative value throws OOM after this many CI accesses "     \
3340           "in each compile)")                                               \
3341                                                                             \
3342   develop(intx, CIFireOOMAtDelay, -1,                                       \
3343           "Wait for this many CI accesses to occur in all compiles before " \
3344           "beginning to throw OutOfMemoryErrors in each compile")           \
3345                                                                             \
```

```
3346   notproduct(bool, CIObjectFactoryVerify, false,                            \
3347           "enable potentially expensive verification in ciObjectFactory")   \
3348                                                                             \
3349   /* Priorities */                                                          \
3350   product_pd(bool, UseThreadPriorities,  "Use native thread priorities")    \
3351                                                                             \
3352   product(intx, ThreadPriorityPolicy, 0,                                    \
3353           "0 : Normal.                                                    "\
3354           "    VM chooses priorities that are appropriate for normal      "\
3355           "    applications. On Solaris NORM_PRIORITY and above are mapped "\
3356           "    to normal native priority. Java priorities below NORM_PRIORITY"\
3357           "    map to lower native priority values. On Windows applications"\
3358           "    are allowed to use higher native priorities. However, with  "\
3359           "    ThreadPriorityPolicy=0, VM will not use the highest possible"\
3360           "    native priority, THREAD_PRIORITY_TIME_CRITICAL, as it may   "\
3361           "    interfere with system threads. On Linux thread priorities   "\
3362           "    are ignored because the OS does not support static priority "\
3363           "    in SCHED_OTHER scheduling class which is the only choice for"\
3364           "    non-root, non-realtime applications.                        "\
3365           "1 : Aggressive.                                                 "\
3366           "    Java thread priorities map over to the entire range of      "\
3367           "    native thread priorities. Higher Java thread priorities map "\
3368           "    to higher native thread priorities. This policy should be   "\
3369           "    used with care, as sometimes it can cause performance       "\
3370           "    degradation in the application and/or the entire system. On "\
3371           "    Linux this policy requires root privilege.")                \
3372                                                                             \
3373   product(bool, ThreadPriorityVerbose, false,                               \
3374           "print priority changes")                                         \
3375                                                                             \
3376   product(intx, DefaultThreadPriority, -1,                                  \
3377           "what native priority threads run at if not specified elsewhere (-1 me
3378                                                                             \
3379   product(intx, CompilerThreadPriority, -1,                                 \
3380           "what priority should compiler threads run at (-1 means no change)") \
3381                                                                             \
3382   product(intx, VMThreadPriority, -1,                                       \
3383           "what priority should VM threads run at (-1 means no change)")    \
3384                                                                             \
3385   product(bool, CompilerThreadHintNoPreempt, true,                          \
3386           "(Solaris only) Give compiler threads an extra quanta")           \
3387                                                                             \
3388   product(bool, VMThreadHintNoPreempt, false,                               \
3389           "(Solaris only) Give VM thread an extra quanta")                  \
3390                                                                             \
3391   product(intx, JavaPriority1_To_OSPriority, -1, "Map Java priorities to OS prio
3392   product(intx, JavaPriority2_To_OSPriority, -1, "Map Java priorities to OS prio
3393   product(intx, JavaPriority3_To_OSPriority, -1, "Map Java priorities to OS prio
3394   product(intx, JavaPriority4_To_OSPriority, -1, "Map Java priorities to OS prio
3395   product(intx, JavaPriority5_To_OSPriority, -1, "Map Java priorities to OS prio
3396   product(intx, JavaPriority6_To_OSPriority, -1, "Map Java priorities to OS prio
3397   product(intx, JavaPriority7_To_OSPriority, -1, "Map Java priorities to OS prio
3398   product(intx, JavaPriority8_To_OSPriority, -1, "Map Java priorities to OS prio
3399   product(intx, JavaPriority9_To_OSPriority, -1, "Map Java priorities to OS prio
3400   product(intx, JavaPriority10_To_OSPriority,-1, "Map Java priorities to OS prio
3401                                                                             \
3402   /* compiler debugging */                                                  \
3403   notproduct(intx, CompileTheWorldStartAt,     1,                           \
3404           "First class to consider when using +CompileTheWorld")            \
3405                                                                             \
3406   notproduct(intx, CompileTheWorldStopAt, max_jint,                         \
3407           "Last class to consider when using +CompileTheWorld")             \
3408                                                                             \
3409   develop(intx, NewCodeParameter,      0,                                   \
3410           "Testing Only: Create a dedicated integer parameter before "      \
3411           "putback")                                                        \
```



```
3412                                                                             \
3413   /* new oopmap storage allocation */                                       \
3414   develop(intx, MinOopMapAllocation,     8,                                 \
3415           "Minimum number of OopMap entries in an OopMapSet")               \
3416                                                                             \
3417   /* Background Compilation */                                              \
3418   develop(intx, LongCompileThreshold,     50,                               \
3419           "Used with +TraceLongCompiles")                                   \
3420                                                                             \
3421   product(intx, StarvationMonitorInterval,    200,                          \
3422           "Pause between each check in ms")                                 \
3423                                                                             \
3424   /* recompilation */                                                       \
3425   product_pd(intx, CompileThreshold,                                        \
3426           "number of interpreted method invocations before (re-)compiling") \
3427                                                                             \
3428   product_pd(intx, BackEdgeThreshold,                                       \
3429           "Interpreter Back edge threshold at which an OSR compilation is invoke
3430                                                                             \
3431   product(intx, Tier0InvokeNotifyFreqLog, 7,                                \
3432           "Interpreter (tier 0) invocation notification frequency.")        \
3433                                                                             \
3434   product(intx, Tier2InvokeNotifyFreqLog, 11,                               \
3435           "C1 without MDO (tier 2) invocation notification frequency.")     \
3436                                                                             \
3437   product(intx, Tier3InvokeNotifyFreqLog, 10,                               \
3438           "C1 with MDO profiling (tier 3) invocation notification "         \
3439           "frequency.")                                                     \
3440                                                                             \
3441   product(intx, Tier0BackedgeNotifyFreqLog, 10,                             \
3442           "Interpreter (tier 0) invocation notification frequency.")        \
3443                                                                             \
3444   product(intx, Tier2BackedgeNotifyFreqLog, 14,                             \
3445           "C1 without MDO (tier 2) invocation notification frequency.")     \
3446                                                                             \
3447   product(intx, Tier3BackedgeNotifyFreqLog, 13,                             \
3448           "C1 with MDO profiling (tier 3) invocation notification "         \
3449           "frequency.")                                                     \
3450                                                                             \
3451   product(intx, Tier2CompileThreshold, 0,                                   \
3452           "threshold at which tier 2 compilation is invoked")               \
3453                                                                             \
3454   product(intx, Tier2BackEdgeThreshold, 0,                                  \
3455           "Back edge threshold at which tier 2 compilation is invoked")     \
3456                                                                             \
3457   product(intx, Tier3InvocationThreshold, 200,                              \
3458           "Compile if number of method invocations crosses this "           \
3459           "threshold")                                                      \
3460                                                                             \
3461   product(intx, Tier3MinInvocationThreshold, 100,                           \
3462           "Minimum invocation to compile at tier 3")                        \
3463                                                                             \
3464   product(intx, Tier3CompileThreshold, 2000,                                \
3465           "Threshold at which tier 3 compilation is invoked (invocation "   \
3466           "minimum must be satisfied.")                                     \
3467                                                                             \
3468   product(intx, Tier3BackEdgeThreshold,  7000,                              \
3469           "Back edge threshold at which tier 3 OSR compilation is invoked") \
3470                                                                             \
3471   product(intx, Tier4InvocationThreshold, 5000,                             \
3472           "Compile if number of method invocations crosses this "           \
3473           "threshold")                                                      \
3474                                                                             \
3475   product(intx, Tier4MinInvocationThreshold, 600,                           \
3476           "Minimum invocation to compile at tier 4")                        \
3477                                                                             \
```

```
3478   product(intx, Tier4CompileThreshold, 15000,                              \
3479           "Threshold at which tier 4 compilation is invoked (invocation "   \
3480           "minimum must be satisfied.")                                     \
3481                                                                             \
3482   product(intx, Tier4BackEdgeThreshold, 40000,                              \
3483           "Back edge threshold at which tier 4 OSR compilation is invoked") \
3484                                                                             \
3485   product(intx, Tier3DelayOn, 5,                                            \
3486           "If C2 queue size grows over this amount per compiler thread "    \
3487           "stop compiling at tier 3 and start compiling at tier 2")         \
3488                                                                             \
3489   product(intx, Tier3DelayOff, 2,                                           \
3490           "If C2 queue size is less than this amount per compiler thread "  \
3491           "allow methods compiled at tier 2 transition to tier 3")          \
3492                                                                             \
3493   product(intx, Tier3LoadFeedback, 5,                                       \
3494           "Tier 3 thresholds will increase twofold when C1 queue size "     \
3495           "reaches this amount per compiler thread")                        \
3496                                                                             \
3497   product(intx, Tier4LoadFeedback, 3,                                       \
3498           "Tier 4 thresholds will increase twofold when C2 queue size "     \
3499           "reaches this amount per compiler thread")                        \
3500                                                                             \
3501   product(intx, TieredCompileTaskTimeout, 50,                               \
3502           "Kill compile task if method was not used within "                \
3503           "given timeout in milliseconds")                                  \
3504                                                                             \
3505   product(intx, TieredStopAtLevel, 4,                                       \
3506           "Stop at given compilation level")                                \
3507                                                                             \
3508   product(intx, Tier0ProfilingStartPercentage, 200,                         \
3509           "Start profiling in interpreter if the counters exceed tier 3"    \
3510           "thresholds by the specified percentage")                         \
3511                                                                             \
3512   product(intx, TieredRateUpdateMinTime, 1,                                 \
3513           "Minimum rate sampling interval (in milliseconds)")               \
3514                                                                             \
3515   product(intx, TieredRateUpdateMaxTime, 25,                                \
3516           "Maximum rate sampling interval (in milliseconds)")               \
3517                                                                             \
3518   product_pd(bool, TieredCompilation,                                       \
3519           "Enable tiered compilation")                                      \
3520                                                                             \
3521   product(bool, PrintTieredEvents, false,                                   \
3522           "Print tiered events notifications")                              \
3523                                                                             \
3524   product(bool, StressTieredRuntime, false,                                 \
3525           "Alternate client and server compiler on compile requests")       \
3526                                                                             \
3527   product_pd(intx, OnStackReplacePercentage,                                \
3528           "NON_TIERED number of method invocations/branches (expressed as %"\
3529           "of CompileThreshold) before (re-)compiling OSR code")            \
3530                                                                             \
3531   product(intx, InterpreterProfilePercentage, 33,                           \
3532           "NON_TIERED number of method invocations/branches (expressed as %"\
3533           "of CompileThreshold) before profiling in the interpreter")       \
3534                                                                             \
3535   develop(intx, MaxRecompilationSearchLength,    10,                        \
3536           "max. # frames to inspect searching for recompilee")              \
3537                                                                             \
3538   develop(intx, MaxInterpretedSearchLength,     3,                          \
3539           "max. # interp. frames to skip when searching for recompilee")    \
3540                                                                             \
3541   develop(intx, DesiredMethodLimit,  8000,                                  \
3542           "desired max. method size (in bytecodes) after inlining")         \
3543                                                                             \
```



```
3544   develop(intx, HugeMethodLimit,  8000,                                     \
3545           "don't compile methods larger than this if "                      \
3546           "+DontCompileHugeMethods")                                        \
3547                                                                             \
3548   /* New JDK 1.4 reflection implementation */                               \
3549                                                                             \
3550   develop(bool, UseNewReflection, true,                                     \
3551           "Temporary flag for transition to reflection based on dynamic "   \
3552           "bytecode generation in 1.4; can no longer be turned off in 1.4 " \
3553           "JDK, and is unneeded in 1.3 JDK, but marks most places VM "      \
3554           "changes were needed")                                            \
3555                                                                             \
3556   develop(bool, VerifyReflectionBytecodes, false,                           \
3557           "Force verification of 1.4 reflection bytecodes. Does not work "  \
3558           "in situations like that described in 4486457 or for "            \
3559           "constructors generated for serialization, so can not be enabled "\
3560           "in product.")                                                    \
3561                                                                             \
3562   product(bool, ReflectionWrapResolutionErrors, true,                       \
3563           "Temporary flag for transition to AbstractMethodError wrapped "   \
3564           "in InvocationTargetException. See 6531596")                      \
3565                                                                             \
3566                                                                             \
3567   develop(intx, FastSuperclassLimit, 8,                                     \
3568           "Depth of hardwired instanceof accelerator array")                \
3569                                                                             \
3570   /* Properties for Java libraries  */                                      \
3571                                                                             \
3572   product(intx, MaxDirectMemorySize, -1,                                    \
3573           "Maximum total size of NIO direct-buffer allocations")            \
3574                                                                             \
3575   /* temporary developer defined flags  */                                  \
3576                                                                             \
3577   diagnostic(bool, UseNewCode, false,                                       \
3578           "Testing Only: Use the new version while testing")                \
3579                                                                             \
3580   diagnostic(bool, UseNewCode2, false,                                      \
3581           "Testing Only: Use the new version while testing")                \
3582                                                                             \
3583   diagnostic(bool, UseNewCode3, false,                                      \
3584           "Testing Only: Use the new version while testing")                \
3585                                                                             \
3586   /* flags for performance data collection */                               \
3587                                                                             \
3588   NOT_EMBEDDED(product(bool, UsePerfData, true,                             \
3589           "Flag to disable jvmstat instrumentation for performance testing" \
3590           "and problem isolation purposes."))                               \
3591                                                                             \
3592   EMBEDDED_ONLY(product(bool, UsePerfData, false,                           \
3593           "Flag to disable jvmstat instrumentation for performance testing" \
3594           "and problem isolation purposes."))                               \
3595                                                                             \
3596   product(bool, PerfDataSaveToFile, false,                                  \
3597           "Save PerfData memory to hsperfdata_<pid> file on exit")          \
3598                                                                             \
3599   product(ccstr, PerfDataSaveFile, NULL,                                    \
3600           "Save PerfData memory to the specified absolute pathname,"        \
3601            "%p in the file name if present will be replaced by pid")        \
3602                                                                             \
3603   product(intx, PerfDataSamplingInterval, 50 /*ms*/,                        \
3604           "Data sampling interval in milliseconds")                         \
3605                                                                             \
3606   develop(bool, PerfTraceDataCreation, false,                               \
3607           "Trace creation of Performance Data Entries")                     \
3608                                                                             \
3609   develop(bool, PerfTraceMemOps, false,                                     \
```

```
3610           "Trace PerfMemory create/attach/detach calls")                  \
3611                                                                           \
3612   product(bool, PerfDisableSharedMem, false,                              \
3613           "Store performance data in standard memory")                    \
3614                                                                           \
3615   product(intx, PerfDataMemorySize, 32*K,                                 \
3616           "Size of performance data memory region. Will be rounded "      \
3617           "up to a multiple of the native os page size.")                 \
3618                                                                           \
3619   product(intx, PerfMaxStringConstLength, 1024,                           \
3620           "Maximum PerfStringConstant string length before truncation")   \
3621                                                                           \
3622   product(bool, PerfAllowAtExitRegistration, false,                       \
3623           "Allow registration of atexit() methods")                       \
3624                                                                           \
3625   product(bool, PerfBypassFileSystemCheck, false,                         \
3626           "Bypass Win32 file system criteria checks (Windows Only)")      \
3627                                                                           \
3628   product(intx, UnguardOnExecutionViolation, 0,                           \
3629           "Unguard page and retry on no-execute fault (Win32 only)"       \
3630           "0=off, 1=conservative, 2=aggressive")                          \
3631                                                                           \
3632   /* Serviceability Support */                                            \
3633                                                                           \
3634   product(bool, ManagementServer, false,                                  \
3635           "Create JMX Management Server")                                 \
3636                                                                           \
3637   product(bool, DisableAttachMechanism, false,                            \
3638          "Disable mechanism that allows tools to attach to this VM")      \
3639                                                                           \
3640   product(bool, StartAttachListener, false,                               \
3641           "Always start Attach Listener at VM startup")                   \
3642                                                                           \
3643   manageable(bool, PrintConcurrentLocks, false,                           \
3644           "Print java.util.concurrent locks in thread dump")              \
3645                                                                           \
3646   product(bool, TransmitErrorReport, false,                               \
3647           "Enable error report transmission on erroneous termination")   \
3648                                                                           \
3649   product(ccstr, ErrorReportServer, NULL,                                 \
3650           "Override built-in error report server address")                \
3651                                                                           \
3652   /* Shared spaces */                                                     \
3653                                                                           \
3654   product(bool, UseSharedSpaces, true,                                    \
3655           "Use shared spaces in the permanent generation")                \
3656                                                                           \
3657   product(bool, RequireSharedSpaces, false,                               \
3658           "Require shared spaces in the permanent generation")            \
3659                                                                           \
3660   product(bool, DumpSharedSpaces, false,                                  \
3661            "Special mode: JVM reads a class list, loads classes, builds " \
3662            "shared spaces, and dumps the shared spaces to a file to be "  \
3663            "used in future JVM runs.")                                    \
3664                                                                           \
3665   product(bool, PrintSharedSpaces, false,                                 \
3666           "Print usage of shared spaces")                                 \
3667                                                                           \
3668   product(uintx, SharedDummyBlockSize, 512*M,                             \
3669           "Size of dummy block used to shift heap addresses (in bytes)")  \
3670                                                                           \
3671   product(uintx, SharedReadWriteSize,  NOT_LP64(12*M) LP64_ONLY(13*M),    \
3672           "Size of read-write space in permanent generation (in bytes)")  \
3673                                                                           \
3674   product(uintx, SharedReadOnlySize,   10*M,                              \
3675           "Size of read-only space in permanent generation (in bytes)")   \
```



```
3676                                                                           \
3677   product(uintx, SharedMiscDataSize,    NOT_LP64(4*M) LP64_ONLY(5*M),     \
3678           "Size of the shared data area adjacent to the heap (in bytes)")  \
3679                                                                           \
3680   product(uintx, SharedMiscCodeSize,    4*M,                              \
3681           "Size of the shared code area adjacent to the heap (in bytes)")  \
3682                                                                           \
3683   diagnostic(bool, SharedOptimizeColdStart, true,                         \
3684           "At dump time, order shared objects to achieve better "         \
3685           "cold startup time.")                                           \
3686                                                                           \
3687   develop(intx, SharedOptimizeColdStartPolicy, 2,                         \
3688           "Reordering policy for SharedOptimizeColdStart "                \
3689           "0=favor classload-time locality, 1=balanced, "                 \
3690           "2=favor runtime locality")                                     \
3691                                                                           \
3692   diagnostic(bool, SharedSkipVerify, false,                               \
3693           "Skip assert() and verify() which page-in unwanted shared "     \
3694           "objects. ")                                                    \
3695                                                                           \
3696   diagnostic(bool, EnableInvokeDynamic, true,                             \
3697           "support JSR 292 (method handles, invokedynamic, "              \
3698           "anonymous classes")                                            \
3699                                                                           \
3700 #endif /* ! codereview */
3701   product(bool, AnonymousClasses, false,                                  \
3702           "support sun.misc.Unsafe.defineAnonymousClass (deprecated)")    \
3696           "support sun.misc.Unsafe.defineAnonymousClass")                 \
3703                                                                           \
3704   experimental(bool, EnableMethodHandles, false,                          \
3705           "support method handles (deprecated)")                          \
3699           "support method handles (true by default under JSR 292)")       \
3706                                                                           \
3707   diagnostic(intx, MethodHandlePushLimit, 3,                              \
3708           "number of additional stack slots a method handle may push")    \
3709                                                                           \
3710   develop(bool, TraceMethodHandles, false,                                \
3711           "trace internal method handle operations")                      \
3712                                                                           \
3713   diagnostic(bool, VerifyMethodHandles, trueInDebug,                      \
3714           "perform extra checks when constructing method handles")        \
3715                                                                           \
3716   diagnostic(bool, OptimizeMethodHandles, true,                           \
3717           "when constructing method handles, try to improve them")        \
3718                                                                           \
3719   experimental(bool, TrustFinalNonStaticFields, false,                    \
3720           "trust final non-static declarations for constant folding")     \
3721                                                                           \
3716   experimental(bool, EnableInvokeDynamic, false,                          \
3717           "recognize the invokedynamic instruction")                      \
3718                                                                           \
3722   experimental(bool, AllowTransitionalJSR292, true,                       \
3723           "recognize pre-PFD formats of invokedynamic")                   \
3724                                                                           \
3725   experimental(bool, PreferTransitionalJSR292, false,                     \
3726           "prefer pre-PFD APIs on boot class path, if they exist")        \
3727                                                                           \
3728   experimental(bool, AllowInvokeForInvokeGeneric, false,                  \
3729           "accept MethodHandle.invoke and MethodHandle.invokeGeneric "    \
3730           "as equivalent methods")                                        \
3731                                                                           \
3732   develop(bool, TraceInvokeDynamic, false,                                \
3733           "trace internal invoke dynamic operations")                     \
3734                                                                           \
3735   diagnostic(bool, PauseAtStartup,      false,                            \
3736           "Causes the VM to pause at startup time and wait for the pause " \
```

```
3737           "file to be removed (default: ./vm.paused.<pid>)")               \
3738                                                                            \
3739   diagnostic(ccstr, PauseAtStartupFile, NULL,                              \
3740           "The file to create and for whose removal to await when pausing " \
3741           "at startup. (default: ./vm.paused.<pid>)")                      \
3742                                                                            \
3743   diagnostic(bool, PauseAtExit, false,                                     \
3744           "Pause and wait for keypress on exit if a debugger is attached")  \
3745                                                                            \
3746   product(bool, ExtendedDTraceProbes,    false,                            \
3747           "Enable performance-impacting dtrace probes")                    \
3748                                                                            \
3749   product(bool, DTraceMethodProbes, false,                                 \
3750           "Enable dtrace probes for method-entry and method-exit")         \
3751                                                                            \
3752   product(bool, DTraceAllocProbes, false,                                  \
3753           "Enable dtrace probes for object allocation")                    \
3754                                                                            \
3755   product(bool, DTraceMonitorProbes, false,                                \
3756           "Enable dtrace probes for monitor events")                       \
3757                                                                            \
3758   product(bool, RelaxAccessControlCheck, false,                            \
3759           "Relax the access control checks in the verifier")               \
3760                                                                            \
3761   diagnostic(bool, PrintDTraceDOF, false,                                  \
3762              "Print the DTrace DOF passed to the system for JSDT probes")  \
3763                                                                            \
3764   product(uintx, StringTableSize, 1009,                                    \
3765           "Number of buckets in the interned String table")                \
3766                                                                            \
3767   product(bool, UseVMInterruptibleIO, false,                               \
3768           "(Unstable, Solaris-specific) Thread interrupt before or with "  \
3769           "EINTR for I/O operations results in OS_INTRPT. The default value"\
3770           " of this flag is true for JDK 6 and earliers")

3772 /*
3773  *  Macros for factoring of globals
3774  */

3776 // Interface macros
3777 #define DECLARE_PRODUCT_FLAG(type, name, value, doc)    extern "C" type name;
3778 #define DECLARE_PD_PRODUCT_FLAG(type, name, doc)        extern "C" type name;
3779 #define DECLARE_DIAGNOSTIC_FLAG(type, name, value, doc) extern "C" type name;
3780 #define DECLARE_EXPERIMENTAL_FLAG(type, name, value, doc) extern "C" type name;
3781 #define DECLARE_MANAGEABLE_FLAG(type, name, value, doc) extern "C" type name;
3782 #define DECLARE_PRODUCT_RW_FLAG(type, name, value, doc) extern "C" type name;
3783 #ifdef PRODUCT
3784 #define DECLARE_DEVELOPER_FLAG(type, name, value, doc)  const type name = value;
3785 #define DECLARE_PD_DEVELOPER_FLAG(type, name, doc)      const type name = pd_##n
3786 #define DECLARE_NOTPRODUCT_FLAG(type, name, value, doc)
3787 #else
3788 #define DECLARE_DEVELOPER_FLAG(type, name, value, doc)  extern "C" type name;
3789 #define DECLARE_PD_DEVELOPER_FLAG(type, name, doc)      extern "C" type name;
3790 #define DECLARE_NOTPRODUCT_FLAG(type, name, value, doc)  extern "C" type name;
3791 #endif
3792 // Special LP64 flags, product only needed for now.
3793 #ifdef _LP64
3794 #define DECLARE_LP64_PRODUCT_FLAG(type, name, value, doc) extern "C" type name;
3795 #else
3796 #define DECLARE_LP64_PRODUCT_FLAG(type, name, value, doc) const type name = valu
3797 #endif // _LP64

3799 // Implementation macros
3800 #define MATERIALIZE_PRODUCT_FLAG(type, name, value, doc)   type name = value;
3801 #define MATERIALIZE_PD_PRODUCT_FLAG(type, name, doc)       type name = pd_##name
3802 #define MATERIALIZE_DIAGNOSTIC_FLAG(type, name, value, doc) type name = value;
```



```
3803 #define MATERIALIZE_EXPERIMENTAL_FLAG(type, name, value, doc) type name = value;
3804 #define MATERIALIZE_MANAGEABLE_FLAG(type, name, value, doc) type name = value;
3805 #define MATERIALIZE_PRODUCT_RW_FLAG(type, name, value, doc) type name = value;
3806 #ifdef PRODUCT
3807 #define MATERIALIZE_DEVELOPER_FLAG(type, name, value, doc) /* flag name is const
3808 #define MATERIALIZE_PD_DEVELOPER_FLAG(type, name, doc)     /* flag name is const
3809 #define MATERIALIZE_NOTPRODUCT_FLAG(type, name, value, doc)
3810 #else
3811 #define MATERIALIZE_DEVELOPER_FLAG(type, name, value, doc) type name = value;
3812 #define MATERIALIZE_PD_DEVELOPER_FLAG(type, name, doc)     type name = pd_##name
3813 #define MATERIALIZE_NOTPRODUCT_FLAG(type, name, value, doc) type name = value;
3814 #endif
3815 #ifdef _LP64
3816 #define MATERIALIZE_LP64_PRODUCT_FLAG(type, name, value, doc)   type name = valu
3817 #else
3818 #define MATERIALIZE_LP64_PRODUCT_FLAG(type, name, value, doc) /* flag is constan
3819 #endif // _LP64

3821 RUNTIME_FLAGS(DECLARE_DEVELOPER_FLAG, DECLARE_PD_DEVELOPER_FLAG, DECLARE_PRODUCT

3823 RUNTIME_OS_FLAGS(DECLARE_DEVELOPER_FLAG, DECLARE_PD_DEVELOPER_FLAG, DECLARE_PROD

3825 #endif // SHARE_VM_RUNTIME_GLOBALS_HPP
```

****
106266 Wed Mar 30 07:00:32 2011
new/src/share/vm/runtime/sharedRuntime.cpp
****
******unchanged_portion_omitted_

```
1678 char* SharedRuntime::generate_wrong_method_type_message(JavaThread* thread,
1679                                                         oopDesc* required,
1680                                                         oopDesc* actual) {
1681   if (TraceMethodHandles) {
1682     tty->print_cr("WrongMethodType thread="PTR_FORMAT" req="PTR_FORMAT" act="PTR
1683                   thread, required, actual);
1684   }
1685   assert(EnableInvokeDynamic, "");
1685   assert(EnableMethodHandles, "");
1686   oop singleKlass = wrong_method_type_is_for_single_argument(thread, required);
1687   char* message = NULL;
1688   if (singleKlass != NULL) {
1689     const char* objName = "argument or return value";
1690     if (actual != NULL) {
1691       // be flexible about the junk passed in:
1692       klassOop ak = (actual->is_klass()
1693                      ? (klassOop)actual
1694                      : actual->klass());
1695       objName = Klass::cast(ak)->external_name();
1696     }
1697     Klass* targetKlass = Klass::cast(required->is_klass()
1698                                      ? (klassOop)required
1699                                      : java_lang_Class::as_klassOop(required));
1700     message = generate_class_cast_message(objName, targetKlass->external_name())
1701   } else {
1702     // %%% need to get the MethodType string, without messing around too much
1703     // Get a signature from the invoke instruction
1704     const char* mhName = "method handle";
1705     const char* targetType = "the required signature";
1706     vframeStream vfst(thread, true);
1707     if (!vfst.at_end()) {
1708       Bytecode_invoke call(vfst.method(), vfst.bci());
1709       methodHandle target;
1710       {
1711         EXCEPTION_MARK;
1712         target = call.static_target(THREAD);
1713         if (HAS_PENDING_EXCEPTION) { CLEAR_PENDING_EXCEPTION; }
1714       }
1715       if (target.not_null()
1716           && target->is_method_handle_invoke()
1717           && required == target->method_handle_type()) {
1718         targetType = target->signature()->as_C_string();
1719       }
1720     }
1721     klassOop kignore; int fignore;
1722     methodOop actual_method = MethodHandles::decode_method(actual,
1723                                                            kignore, fignore);
1724     if (actual_method != NULL) {
1725       if (methodOopDesc::is_method_handle_invoke_name(actual_method->name()))
1726         mhName = "$";
1727       else
1728         mhName = actual_method->signature()->as_C_string();
1729       if (mhName[0] == '$')
1730         mhName = actual_method->signature()->as_C_string();
1731     }
1732     message = generate_class_cast_message(mhName, targetType,
1733                                           " cannot be called as ");
1734   }
1735   if (TraceMethodHandles) {
1736     tty->print_cr("WrongMethodType => message=%s", message);
```



```
1737   }
1738   return message;
1739 }
```

******unchanged_portion_omitted_

```
*********************************************************
  156168 Wed Mar 30 07:00:33 2011
new/src/share/vm/runtime/thread.cpp
*********************************************************
______unchanged_portion_omitted_
```