```
**********************************************************
   15038 Mon Oct 10 06:19:23 2011
new/src/cpu/x86/vm/methodHandles_x86.hpp
**********************************************************
______unchanged_portion_omitted_

public:

// The stack just after the recursive call from a ricochet frame
// looks something like this.  Offsets are marked in words, not bytes.
// rsi (r13 on LP64) is part of the interpreter calling sequence
// which tells the callee where is my real rsp (for frame walking).
// (...lower memory addresses)
// rsp:     [ return pc                 ]   always the global RicochetBlob::boun
// rsp+1:   [ recursive arg N           ]
// rsp+2:   [ recursive arg N-1         ]
// ...
// rsp+N:   [ recursive arg 1           ]
// rsp+N+1: [ recursive method handle   ]
// ...
// rbp-6:   [ cleanup continuation pc   ]   <-- (struct RicochetFrame)
// rbp-5:   [ saved target MH           ]   the MH we will call on the saved arg
// rbp-4:   [ saved args layout oop     ]   an int[] array which describes argum
// rbp-3:   [ saved args pointer        ]   address of transformed adapter arg M
// rbp-2:   [ conversion                ]   information about how the return val
// rbp-1:   [ exact sender sp           ]   exact TOS (rsi/r13) of original send
// rbp+0:   [ saved sender fp           ]   (for original sender of AMH)
// rbp+1:   [ saved sender pc           ]   (back to original sender of AMH)
// rbp+2:   [ transformed adapter arg M ]   <-- (extended TOS of original sender
// rbp+3:   [ transformed adapter arg M-1]
// ...
// rbp+M+1: [ transformed adapter arg 1 ]
// rbp+M+2: [ padding                   ] <-- (rbp + saved args base offset)
// ...      [ optional padding]
// (higher memory addresses...)
//
// The arguments originally passed by the original sender
// are lost, and arbitrary amounts of stack motion might have
// happened due to argument transformation.
// (This is done by C2I/I2C adapters and non-direct method handles.)
// This is why there is an unpredictable amount of memory between
// the extended and exact TOS of the sender.
// The ricochet adapter itself will also (in general) perform
// transformations before the recursive call.
//
// The transformed and saved arguments, immediately above the saved
// return PC, are a well-formed method handle invocation ready to execute.
// When the GC needs to walk the stack, these arguments are described
// via the saved arg types oop, an int[] array with a private format.
// This array is derived from the type of the transformed adapter
// method handle, which also sits at the base of the saved argument
// bundle.  Since the GC may not be able to fish out the int[]
// array, so it is pushed explicitly on the stack.  This may be
// an unnecessary expense.
//
// The following register conventions are significant at this point:
// rsp       the thread stack, as always; preserved by caller
// rsi/r13   exact TOS of recursive frame (contents of [rbp-2])
// rcx       recursive method handle (contents of [rsp+N+1])
// rbp       preserved by caller (not used by caller)
// Unless otherwise specified, all registers can be blown by the call.
//
// If this frame must be walked, the transformed adapter arguments
// will be found with the help of the saved arguments descriptor.
//
// Therefore, the descriptor must match the referenced arguments.
```

```
// The arguments must be followed by at least one word of padding,
// which will be necessary to complete the final method handle call.
// That word is not treated as holding an oop.  Neither is the word
//
// The word pointed to by the return argument pointer is not
// treated as an oop, even if points to a saved argument.
// This allows the saved argument list to have a "hole" in it
// to receive an oop from the recursive call.
// (The hole might temporarily contain RETURN_VALUE_PLACEHOLDER.)
//
// When the recursive callee returns, RicochetBlob::bounce_addr will
// immediately jump to the continuation stored in the RF.
// This continuation will merge the recursive return value
// into the saved argument list.  At that point, the original
// rsi, rbp, and rsp will be reloaded, the ricochet frame will
// disappear, and the final target of the adapter method handle
// will be invoked on the transformed argument list.

class RicochetFrame {
  friend class MethodHandles;
  friend class VMStructs;

 private:
  intptr_t* _continuation;          // what to do when control gets back here
  oopDesc*  _saved_target;          // target method handle to invoke on saved_a
  oopDesc*  _saved_args_layout;     // caching point for MethodTypeForm.vmlayout
  intptr_t* _saved_args_base;       // base of pushed arguments (slot 0, arg N)
  intptr_t  _conversion;            // misc. information from original AdapterMe
  intptr_t* _exact_sender_sp;       // parallel to interpreter_frame_sender_sp (
  intptr_t* _sender_link;           // *must* coincide with frame::link_offset (
  address   _sender_pc;             // *must* coincide with frame::return_addr_o

 public:
  intptr_t* continuation() const        { return _continuation; }
  oop       saved_target() const        { return _saved_target; }
  oop       saved_args_layout() const   { return _saved_args_layout; }
  intptr_t* saved_args_base() const     { return _saved_args_base; }
  intptr_t  conversion() const          { return _conversion; }
  intptr_t* exact_sender_sp() const     { return _exact_sender_sp; }
  intptr_t* sender_link() const         { return _sender_link; }
  address   sender_pc() const           { return _sender_pc; }

  intptr_t* extended_sender_sp() const {
    // The extended sender SP is above the current RicochetFrame.
    return (intptr_t*) (((address) this) + sizeof(RicochetFrame));
  }
  intptr_t* extended_sender_sp() const  { return saved_args_base(); }

  intptr_t  return_value_slot_number() const {
    return adapter_conversion_vminfo(conversion());
  }
  BasicType return_value_type() const {
    return adapter_conversion_dest_type(conversion());
  }
  bool has_return_value_slot() const {
    return return_value_type() != T_VOID;
  }
  intptr_t* return_value_slot_addr() const {
    assert(has_return_value_slot(), "");
    return saved_arg_slot_addr(return_value_slot_number());
  }
  intptr_t* saved_target_slot_addr() const {
    return saved_arg_slot_addr(saved_args_length());
  }
  intptr_t* saved_arg_slot_addr(int slot) const {
    assert(slot >= 0, "");
    return (intptr_t*)( (address)saved_args_base() + (slot * Interpreter::stackE
  }

  jint      saved_args_length() const;
  jint      saved_arg_offset(int arg) const;

  // GC interface
  oop*  saved_target_addr()                     { return (oop*)&_saved_target; }
  oop*  saved_args_layout_addr()                { return (oop*)&_saved_args_layo

  oop  compute_saved_args_layout(bool read_cache, bool write_cache);

  // Compiler/assembler interface.
  static int continuation_offset_in_bytes()     { return offset_of(RicochetFrame
  static int saved_target_offset_in_bytes()     { return offset_of(RicochetFrame
  static int saved_args_layout_offset_in_bytes(){ return offset_of(RicochetFrame
  static int saved_args_base_offset_in_bytes()  { return offset_of(RicochetFrame
  static int conversion_offset_in_bytes()       { return offset_of(RicochetFrame
  static int exact_sender_sp_offset_in_bytes()  { return offset_of(RicochetFrame
  static int sender_link_offset_in_bytes()      { return offset_of(RicochetFrame
  static int sender_pc_offset_in_bytes()        { return offset_of(RicochetFrame

  // This value is not used for much, but it apparently must be nonzero.
  static int frame_size_in_bytes()              { return sender_link_offset_in_b

#ifdef ASSERT
  // The magic number is supposed to help find ricochet frames within the bytes
  enum { MAGIC_NUMBER_1 = 0xFEED03E, MAGIC_NUMBER_2 = 0xBEEF03E };
  static int magic_number_1_offset_in_bytes()   { return -wordSize; }
  static int magic_number_2_offset_in_bytes()   { return sizeof(RicochetFrame);
  intptr_t magic_number_1() const               { return *(intptr_t*)((address)t
  intptr_t magic_number_2() const               { return *(intptr_t*)((address)t
#endif //ASSERT

  enum { RETURN_VALUE_PLACEHOLDER = (NOT_DEBUG(0) DEBUG_ONLY(42)) };

  static void verify_offsets() NOT_DEBUG_RETURN;
  void verify() const NOT_DEBUG_RETURN; // check for MAGIC_NUMBER, etc.
  void zap_arguments() NOT_DEBUG_RETURN;

  static void generate_ricochet_blob(MacroAssembler* _masm,
                                     // output params:
                                     int* bounce_offset,
                                     int* exception_offset,
                                     int* frame_size_in_words);

  static void enter_ricochet_frame(MacroAssembler* _masm,
                                   Register rcx_recv,
                                   Register rax_argv,
                                   address return_handler,
                                   Register rbx_temp);
  static void leave_ricochet_frame(MacroAssembler* _masm,
                                   Register rcx_recv,
                                   Register new_sp_reg,
                                   Register sender_pc_reg);

  static Address frame_address(int offset = 0) {
    // The RicochetFrame is found by subtracting a constant offset from rbp.
    return Address(rbp, - sender_link_offset_in_bytes() + offset);
  }

  static RicochetFrame* from_frame(const frame& fr) {
    address bp = (address) fr.fp();
    RicochetFrame* rf = (RicochetFrame*)(bp - sender_link_offset_in_bytes());
    rf->verify();
    return rf;
```

```
  }

  static void verify_clean(MacroAssembler* _masm) NOT_DEBUG_RETURN;
};
```

______***unchanged_portion_omitted_***

****
  102580 Mon Oct 10 06:19:24 2011
new/src/cpu/x86/vm/methodHandles_x86.cpp
****

______***unchanged_portion_omitted_***

```
void MethodHandles::RicochetFrame::verify() const {
  verify_offsets();
  assert(magic_number_1() == MAGIC_NUMBER_1, err_msg(PTR_FORMAT " == " PTR_FORMA
  assert(magic_number_2() == MAGIC_NUMBER_2, err_msg(PTR_FORMAT " == " PTR_FORMA
  assert(magic_number_1() == MAGIC_NUMBER_1, "");
  assert(magic_number_2() == MAGIC_NUMBER_2, "");
  if (!Universe::heap()->is_gc_active()) {
    if (saved_args_layout() != NULL) {
      assert(saved_args_layout()->is_method(), "must be valid oop");
    }
    if (saved_target() != NULL) {
      assert(java_lang_invoke_MethodHandle::is_instance(saved_target()), "checki
    }
  }
  int conv_op = adapter_conversion_op(conversion());
  assert(conv_op == java_lang_invoke_AdapterMethodHandle::OP_COLLECT_ARGS ||
         conv_op == java_lang_invoke_AdapterMethodHandle::OP_FOLD_ARGS ||
         conv_op == java_lang_invoke_AdapterMethodHandle::OP_PRIM_TO_REF,
         "must be a sane conversion");
  if (has_return_value_slot()) {
    assert(*return_value_slot_addr() == RETURN_VALUE_PLACEHOLDER, "");
  }
}
```

______***unchanged_portion_omitted_***

****
   23074 Mon Oct 10 06:19:25 2011
new/src/cpu/x86/vm/frame_x86.cpp
****

______*unchanged_portion_omitted_*

```
void frame::patch_pc(Thread* thread, address pc) {
  address* pc_addr = &(((address*) sp())[-1]);
#endif /* ! codereview */
  if (TracePcPatching) {
    tty->print_cr("patch_pc at address " INTPTR_FORMAT " [" INTPTR_FORMAT " -> "
                  pc_addr, *pc_addr, pc);
    tty->print_cr("patch_pc at address" INTPTR_FORMAT " [" INTPTR_FORMAT " -> "
                  &((address *)sp())[-1], ((address *)sp())[-1], pc);
  }
  assert(_pc == *pc_addr, err_msg("must be: " INTPTR_FORMAT " == " INTPTR_FORMAT
  *pc_addr = pc;
  ((address *)sp())[-1] = pc;
  _cb = CodeCache::find_blob(pc);
  address original_pc = nmethod::get_deopt_original_pc(this);
  if (original_pc != NULL) {
    assert(original_pc == _pc, "expected original PC to be stored before patchin
    _deopt_state = is_deoptimized;
    // leave _pc as is
  } else {
    _deopt_state = not_deoptimized;
    _pc = pc;
  }
}
```

______*unchanged_portion_omitted_*