```
  //堆栈方向与书上常见的相反
  //这里要从下往上看，下面的内存地址高，上面的小，每push一次rsp要减去4（左边的标号*4 + rbp就得到它的内存地址）
  //------------------------------------------------------------------------------------------------------------------------
  // Call stubs are used to call Java from C
  //    return_from_Java 是紧跟在call *%eax后面的那条指令的地址
  //    [ return_from_Java     ] <--- rsp
  //    [ argument word n      ]
  //      ...
  // -N [ argument word 1      ]
  // -7 [ Possible padding for stack alignment ]
  // -6 [ Possible padding for stack alignment ]
  // -5 [ Possible padding for stack alignment ]
  // -4 [ mxcsr save           ] <--- rsp_after_call
  // -3 [ saved rbx,            ]
  // -2 [ saved rsi            ]
  // -1 [ saved rdi            ]
  //  0 [ saved rbp,            ] <--- rbp,
  //  1 [ return address       ]
  //  2 [ ptr. to call wrapper ]
  //  3 [ result               ]
  //  4 [ result_type          ]
  //  5 [ method               ]
  //  6 [ entry_point          ]
  //  7 [ parameters           ]
  //  8 [ parameter_size       ]
  //  9 [ thread               ]

StubRoutines::call_stub [0x07f003b4, 0x07f00485[ (209 bytes)
  0x07f003b4: push   %ebp
  0x07f003b5: mov    %esp,%ebp

  //0x20(%ebp) = 从内存地址(%ebp的值(是一个内存地址) + 32(4*8, 正好是第8项))中取值
  //所以最后%ecx的值是parameter_size
  0x07f003b7: mov    0x20(%ebp),%ecx
  //每个参数都占4个字节，所以把%ecx左移两位就算出所有的参数共占用多少字节
  0x07f003ba: shl    $0x2,%ecx
  //$0x10是16，因为要保存rdi、rsi、rbx、mxcsr这4个寄存器的值，每个占4个字节，所以再加16
  0x07f003bd: add    $0x10,%ecx
  //把%esp移到最后一个parameter位置
  0x07f003c0: sub    %ecx,%esp
  //堆栈按16位对齐，这里去掉后4位，相当于减去后4位的值，如果前面两条指令得到的字节数不够16的整数倍，这里就会减小%esp的值
  0x07f003c2: and    $0xfffffff0,%esp

  //按惯例，被调用者(被call指令调用)要保存rdi、rsi、rbx这3个寄存器的值
  0x07f003c5: mov    %edi,-0x4(%ebp)
  0x07f003c8: mov    %esi,-0x8(%ebp)
  0x07f003cb: mov    %ebx,-0xc(%ebp)

  //保存mxcsr寄存器的值，属于SSE，在VS中的寄存器窗口右击，然后选择SSE就可以看到了
  0x07f003ce: stmxcsr -0x10(%ebp)

  0x07f003d2: mov    -0x10(%ebp),%eax
  0x07f003d5: and    $0xffc0,%eax
  0x07f003db: cmp    0x56005778,%eax
  0x07f003e1: je     0x07f003ee
  0x07f003e7: ldmxcsr 0x56005778 //如果0x56005778(在数据段)中的值与%eax中的值不同，则把0x56005778中的值保存到mxcsr寄存器

  //对应CTRL寄存器，在VS中的寄存器窗口右击，然后选择Floating Point就可以看到了
  0x07f003ee: fldcw  0x56005768 //Loads the 16-bit source operand into the FPU control word.

  // make sure we have no pending exceptions
  0x07f003f4: mov    0x24(%ebp),%ecx //对应thread
  0x07f003f7: cmpl   $0x0,0x4(%ecx) //看看thread对象的_pending_exception字段是否为0，不为0就表示有pending exceptions
  0x07f003fe: je     0x07f00415
  //stop("StubRoutines::call_stub: entered with pending exception");
  0x07f00404: push   $0x55ce7d38
  0x07f00409: call   0x07f0040e
  0x07f0040e: pusha
  0x07f0040f: call   0x557bdbf0
  0x07f00414: hlt

  ;; pass parameters if any
  0x07f00415: mov    0x20(%ebp),%ecx
  0x07f00418: test   %ecx,%ecx //parameter_size是0就直接跳过参数处理
  0x07f0041a: je     0x07f00430
  0x07f00420: mov    0x1c(%ebp),%edx //对应parameters
  0x07f00423: xor    %ebx,%ebx //把%ebx设为0

  //从后往前遍历参数，然后放到堆栈中
  //parameters是个数组，所以parameters的内存地址就是第一个数组元素的地址，
  //第i(i>=0)个元素的地址 = parameters的内存地址 + i*4
```

```
//因为%ecx是parameter_size, 按%ecx递减时算出的地址是多了 4 个字节的，所以要减去4，
//比如parameters的内存地址是0x11223300，有3个数组元素，
//那么每个数组元素的内存地址分别是
//parameters[0] = 0x11223300 = parameters的内存地址 = %edx的值
//parameters[1] = 0x11223304
//parameters[2] = 0x11223308
//此时%ecx = 3
//从后往前遍历参数时，先计算第三个元素的内存地址
//parameters[2] = %edx + %ecx*4 - 0x4 =  0x11223300 + 12 - 0x4 = 0x11223308
//最后再把0x11223308中存放的值放到%eax
;; loop:
0x07f00425: mov    -0x4(%edx,%ecx,4),%eax
0x07f00429: mov    %eax,(%esp,%ebx,4)
0x07f0042c: inc    %ebx
0x07f0042d: dec    %ecx
0x07f0042e: jne    0x07f00425

;; parameters_done:
//这两条很关键，%ebx中存放着method，
//接下来就要重点关注method entry point (kind = zerolocals)如何使用%ebx
0x07f00430: mov    0x14(%ebp),%ebx //对应method
0x07f00433: mov    0x18(%ebp),%eax //对应entry_point
0x07f00436: mov    %esp,%esi
;; call Java function
0x07f00438: call   *%eax

;; call_stub_return_address:
0x07f0043a: mov    0xc(%ebp),%edi  //result
0x07f0043d: mov    0x10(%ebp),%esi //result_type

0x07f00440: cmp    $0xb,%esi  //T_LONG = 0xb
0x07f00443: je     0x07f00472
0x07f00449: cmp    $0x6,%esi  //T_FLOAT
0x07f0044c: je     0x07f00479
0x07f00452: cmp    $0x7,%esi  //T_DOUBLE
0x07f00455: je     0x07f0047f
0x07f0045b: mov    %eax,(%edi) //把结果放到result
;; exit:
0x07f0045d: lea    -0x10(%ebp),%esp
0x07f00460: ldmxcsr -0x10(%ebp)
0x07f00464: mov    -0xc(%ebp),%ebx
0x07f00467: mov    -0x8(%ebp),%esi
0x07f0046a: mov    -0x4(%ebp),%edi
0x07f0046d: add    $0x10,%esp
0x07f00470: pop    %ebp
0x07f00471: ret
;; is_long:
0x07f00472: mov    %eax,(%edi)
0x07f00474: mov    %edx,0x4(%edi)
0x07f00477: jmp    0x07f0045d
;; is_float:
0x07f00479: movss  %xmm0,(%edi)
0x07f0047d: jmp    0x07f0045d
;; is_double:
0x07f0047f: movsd  %xmm0,(%edi)
0x07f00483: jmp    0x07f0045d
```