```
2.4   Function Calls
Instruction call <address> is used to perform calls. It does exactly the following:
  push rip
  jmp <address>
The address now stored in the stack (former rip contents) is called return address.
Any function can accept an unlimited number of arguments.
The first six arguments are passed in rdi, rsi, rdx, rcx, r8, and r9, respectively.
The rest is passed on to the stack in reverse order.

  Apparently, the fragile mechanism of call and ret only works when the state of the stack is carefully
managed. One should not invoke ret unless the stack is exactly in the same state as when the function
started. Otherwise, the processor will take whatever is on top of the stack as a return address and use it as the
new rip content, which will certainly lead to executing garbage.
  Now let’s talk about how functions use registers. Obviously, executing a function can change registers.
There are two types of registers.
  - Callee-saved registers must be restored by the procedure being called. So, if it needs
    to change them, it has to change them back.
    These registers are callee-saved: rbx, rbp, rsp, r12-r15, a total of seven registers.
  - Caller-saved registers should be saved before invoking a function and restored after. One
    does not have to save and restore them if their value will not be of importance after.
    All other registers are caller-saved.
These two categories are a convention. That is, a programmer must follow this agreement by
  - Saving and restoring callee-saved registers.
  - Being always aware that caller-saved registers can be changed during function execution.

StubRoutines::call_stub [0x00007f8ab9000564, 0x00007f8ab900079b[ (567 bytes)
  ;; enter
  0x00007f8ab9000564: push   %rbp
  0x00007f8ab9000565: mov    %rsp,%rbp
  ;; subptr
  0x00007f8ab9000568: sub    $0x60,%rsp         # 16x6=96=8x12 也就是留出12个8字节的寄存器
  0x00007f8ab900056c: mov    %r9,-0x8(%rbp)     # -0x8%rbp 指向 parameters
  0x00007f8ab9000570: mov    %r8,-0x10(%rbp)    # -0x10%rbp 指向 entry point
  ;; c_rarg3
  0x00007f8ab9000574: mov    %rcx,-0x18(%rbp)   # 16+8=24, 8x3, 从rbp往下数第3个, it is "method"
  ;; c_rarg2
  0x00007f8ab9000578: mov    %edx,-0x20(%rbp)   # 16x2=32, 8x4, 从rbp往下数第4个, it is "result type"
  ;; c_rarg1
  0x00007f8ab900057b: mov    %rsi,-0x28(%rbp)   # 40, 8x5, 从rbp往下数第5个,  it is "result address"
  ;; c_rarg0
  0x00007f8ab900057f: mov    %rdi,-0x30(%rbp)   # 48, 8x6, 从rbp往下数第6个,  it is "call wrapper address"
  ;; rbx
  0x00007f8ab9000583: mov    %rbx,-0x38(%rbp)   # 56, 从rbp往下数第7个,  it is "rbx"
  ;; r12
  0x00007f8ab9000587: mov    %r12,-0x40(%rbp)   # 从rbp往下数第8个,  it is "r12"
  0x00007f8ab900058b: mov    %r13,-0x48(%rbp)   # 从rbp往下数第9个,  it is "r13"
  0x00007f8ab900058f: mov    %r14,-0x50(%rbp)   # 从rbp往下数第10个,  it is "r14"
  0x00007f8ab9000593: mov    %r15,-0x58(%rbp)   # 从rbp往下数第11个,  it is "r15"
  ;; stmxcsr
  0x00007f8ab9000597: stmxcsr -0x60(%rbp)
  ;; movl
  0x00007f8ab900059b: mov    -0x60(%rbp),%eax   # 16x6=96, 8x12, 指向第一个入参
  ;; andl
  0x00007f8ab900059e: and    $0xffc0,%eax       #
  ;; cmp32
  0x00007f8ab90005a4: cmp    0x1698e19e(%rip),%eax        # 0x00007f8acf98e748
  ;; jcc
  0x00007f8ab90005aa: je     0x00007f8ab90005b7
  ;; ldmxcsr
  0x00007f8ab90005b0: ldmxcsr 0x1698e191(%rip)        # 0x00007f8acf98e748
  ;; bind
  ;; Load up thread register
  0x00007f8ab90005b7: mov    0x18(%rbp),%r15    # 16+8=24, 8x3, 从rbp往上数第3个,  it is "thread"
  ;; reinit_heapbase
  0x00007f8ab90005bb: mov    0x169adc1e(%rip),%r12        # 0x00007f8acf9ae1e0
  ;; cmpptr
  0x00007f8ab90005c2: cmpq   $0x0,0x8(%r15)
  ;; jcc
  0x00007f8ab90005ca: je     0x00007f8ab9000647
  ;; stop
  0x00007f8ab90005d0: mov    %rsp,-0x28(%rsp)   # -0x28, 16x2+8=40=8x5, 把rsp往下移5个位置
  0x00007f8ab90005d5: sub    $0x80,%rsp         # 16x8, 也就是把rsp往下移16个位置
  0x00007f8ab90005dc: mov    %rax,0x78(%rsp)    # 14
  0x00007f8ab90005e1: mov    %rcx,0x70(%rsp)    # 13
  0x00007f8ab90005e6: mov    %rdx,0x68(%rsp)    # 12
  0x00007f8ab90005eb: mov    %rbx,0x60(%rsp)    # 11
  0x00007f8ab90005f0: mov    %rbp,0x50(%rsp)    # 10
  0x00007f8ab90005f5: mov    %rsi,0x48(%rsp)    # 9
  0x00007f8ab90005fa: mov    %rdi,0x40(%rsp)    # 16x4/8=8, rdi
  0x00007f8ab90005ff: mov    %r8,0x38(%rsp)     # 7, r8
  0x00007f8ab9000604: mov    %r9,0x30(%rsp)     # 6, r9
  0x00007f8ab9000609: mov    %r10,0x28(%rsp)    # 5, r10
  0x00007f8ab900060e: mov    %r11,0x20(%rsp)    # 4, r11
  0x00007f8ab9000613: mov    %r12,0x18(%rsp)    # 3, r12
  0x00007f8ab9000618: mov    %r13,0x10(%rsp)    # 2, r13
  0x00007f8ab900061d: mov    %r14,0x8(%rsp)     # 1, r14
```

```
0x00007f8ab9000622: mov    %r15,(%rsp)        # %rsp, r15
0x00007f8ab9000626: movabs $0x7f8acf3a6170,%rdi
0x00007f8ab9000630: movabs $0x7f8ab90005d0,%rsi
0x00007f8ab900063a: mov    %rsp,%rdx
0x00007f8ab900063d: and    $0xfffffffffffffff0,%rsp
0x00007f8ab9000641: callq  0x00007f8aceddb9fa
0x00007f8ab9000646: hlt
;; bind
;; pass parameters if any
;; movl
0x00007f8ab9000647: mov    0x10(%rbp),%ecx    # 16, 8x2, 从rbp往上数第2个, it is "parameterSize" moved to %ecx
;; testl
0x00007f8ab900064a: test   %ecx,%ecx          # 检验parameter_size是否为0,若是0则直接跳过参数处理
;; jcc
0x00007f8ab900064c: je     0x00007f8ab9000664
;; movptr
0x00007f8ab9000652: mov    -0x8(%rbp),%rdx    # 8, 从rbp往下数第1个, it is "parameters"
;; movl
0x00007f8ab9000656: mov    %ecx,%esi  # move data in ecx, parameterSize", to esi
;; BIND
;; loop:
;; movptr
0x00007f8ab9000658: mov    (%rdx),%rax    # pass value of "parameters" to rax
;; addptr
0x00007f8ab900065b: add    $0x8,%rdx      # "parameters" 的下一个指令是 rbp
;; decrementl
0x00007f8ab900065f: dec    %esi   # parameterSize-1
;; push
0x00007f8ab9000661: push   %rax
;; jcc
0x00007f8ab9000662: jne    0x00007f8ab9000658
;; BIND
;; parameters_done:
;; movptr
0x00007f8ab9000664: mov    -0x18(%rbp),%rbx   # -0x18, 16+8=24=8x3, 从rbp往下数第3个, it is "method"
;; movptr
0x00007f8ab9000668: mov    -0x10(%rbp),%rsi   # -0x10, 16=8x2, 从rbp往下数第2个, it is "entry point"
;; mov
0x00007f8ab900066c: mov    %rsp,%r13  # 把当前的栈顶, rsp, 放到 r13
;; call Java function
0x00007f8ab900066f: callq  *%rsi  # call "entry point"
;; call_stub_return_address:
;; movptr
0x00007f8ab9000671: mov    -0x28(%rbp),%rdi   # -0x28=32+8=40=8x5, 从rbp往下数第5个, it is "result address"
;; movl
0x00007f8ab9000675: mov    -0x20(%rbp),%esi   # -0x20=16x2=8x4, 从rbp往下数第4个, it is "result type"
;; cmpl
0x00007f8ab9000678: cmp    $0xc,%esi
0x00007f8ab900067b: je     0x00007f8ab9000781
0x00007f8ab9000681: cmp    $0xb,%esi
0x00007f8ab9000684: je     0x00007f8ab9000781
0x00007f8ab900068a: cmp    $0x6,%esi
0x00007f8ab900068d: je     0x00007f8ab9000789
0x00007f8ab9000693: cmp    $0x7,%esi
0x00007f8ab9000696: je     0x00007f8ab9000792
;; movl
0x00007f8ab900069c: mov    %eax,(%rdi)
;; BIND
;; exit:
;; lea
0x00007f8ab900069e: lea    -0x60(%rbp),%rsp
;; cmpptr
0x00007f8ab90006a2: cmp    0x18(%rbp),%r15
;; jcc
0x00007f8ab90006a6: jne    0x00007f8ab90006e6
;; get_thread
0x00007f8ab90006ac: push   %rax
0x00007f8ab90006ad: push   %rdi
0x00007f8ab90006ae: push   %rsi
0x00007f8ab90006af: push   %rdx
0x00007f8ab90006b0: push   %rcx
0x00007f8ab90006b1: push   %r8
0x00007f8ab90006b3: push   %r9
0x00007f8ab90006b5: push   %r10
0x00007f8ab90006b7: mov    %rsp,%r10
0x00007f8ab90006ba: and    $0xfffffffffffffff0,%rsp
0x00007f8ab90006be: push   %r10
0x00007f8ab90006c0: push   %r11
0x00007f8ab90006c2: mov    $0x1,%edi
0x00007f8ab90006c7: callq  0x00007f8acf9c32d0
0x00007f8ab90006cc: pop    %r11
0x00007f8ab90006ce: pop    %rsp
0x00007f8ab90006cf: pop    %r10
0x00007f8ab90006d1: pop    %r9
0x00007f8ab90006d3: pop    %r8
0x00007f8ab90006d5: pop    %rcx
```

```
    0x00007f8ab90006d6: pop    %rdx
    0x00007f8ab90006d7: pop    %rsi
    0x00007f8ab90006d8: pop    %rdi
    0x00007f8ab90006d9: mov    %rax,%rbx
    0x00007f8ab90006dc: pop    %rax
    ;; cmpptr
    0x00007f8ab90006dd: cmp    %rbx,%r15
    ;; jcc
    0x00007f8ab90006e0: je     0x00007f8ab9000763
    ;; bind
    ;; jcc
    0x00007f8ab90006e6: je     0x00007f8ab9000763
    ;; stop
    0x00007f8ab90006ec: mov    %rsp,-0x28(%rsp)
    0x00007f8ab90006f1: sub    $0x80,%rsp
    0x00007f8ab90006f8: mov    %rax,0x78(%rsp)
    0x00007f8ab90006fd: mov    %rcx,0x70(%rsp)
    0x00007f8ab9000702: mov    %rdx,0x68(%rsp)
    0x00007f8ab9000707: mov    %rbx,0x60(%rsp)
    0x00007f8ab900070c: mov    %rbp,0x50(%rsp)
    0x00007f8ab9000711: mov    %rsi,0x48(%rsp)
    0x00007f8ab9000716: mov    %rdi,0x40(%rsp)
    0x00007f8ab900071b: mov    %r8,0x38(%rsp)
    0x00007f8ab9000720: mov    %r9,0x30(%rsp)
    0x00007f8ab9000725: mov    %r10,0x28(%rsp)
    0x00007f8ab900072a: mov    %r11,0x20(%rsp)
    0x00007f8ab900072f: mov    %r12,0x18(%rsp)
    0x00007f8ab9000734: mov    %r13,0x10(%rsp)
    0x00007f8ab9000739: mov    %r14,0x8(%rsp)
    0x00007f8ab900073e: mov    %r15,(%rsp)
    0x00007f8ab9000742: movabs $0x7f8acf3a6250,%rdi
    0x00007f8ab900074c: movabs $0x7f8ab90006ec,%rsi
    0x00007f8ab9000756: mov    %rsp,%rdx
    0x00007f8ab9000759: and    $0xfffffffffffffff0,%rsp
    0x00007f8ab900075d: callq  0x00007f8aceddb9fa
    0x00007f8ab9000762: hlt
    ;; bind
    ;; r15
    0x00007f8ab9000763: mov    -0x58(%rbp),%r15   # -0x58=16x5+8=88=8x11, 从rbp往下数第11个, it is "r15"
    0x00007f8ab9000767: mov    -0x50(%rbp),%r14
    0x00007f8ab900076b: mov    -0x48(%rbp),%r13
    0x00007f8ab900076f: mov    -0x40(%rbp),%r12
    0x00007f8ab9000773: mov    -0x38(%rbp),%rbx
    ;; ldmxcsr
    0x00007f8ab9000777: ldmxcsr -0x60(%rbp)
    ;; addptr
    0x00007f8ab900077b: add    $0x60,%rsp
    ;; pop
    0x00007f8ab900077f: pop    %rbp
    ;; ret
    0x00007f8ab9000780: retq
    ;; is_long:
    0x00007f8ab9000781: mov    %rax,(%rdi)
    0x00007f8ab9000784: jmpq   0x00007f8ab900069e
    ;; is_float:
    0x00007f8ab9000789: vmovss %xmm0,(%rdi)
    0x00007f8ab900078d: jmpq   0x00007f8ab900069e
    ;; is_double:
    0x00007f8ab9000792: vmovsd %xmm0,(%rdi)
    0x00007f8ab9000796: jmpq   0x00007f8ab900069e
```