第4篇-JVM终于开始调用Java主类的main()方法啦

Original 鸠摩 深入剖析Java虚拟机HotSpot 2021-12-06 15:57

收录于合集

#java 9 #运行时 9 #hotspot 10 #main() 3 #虚拟机 10

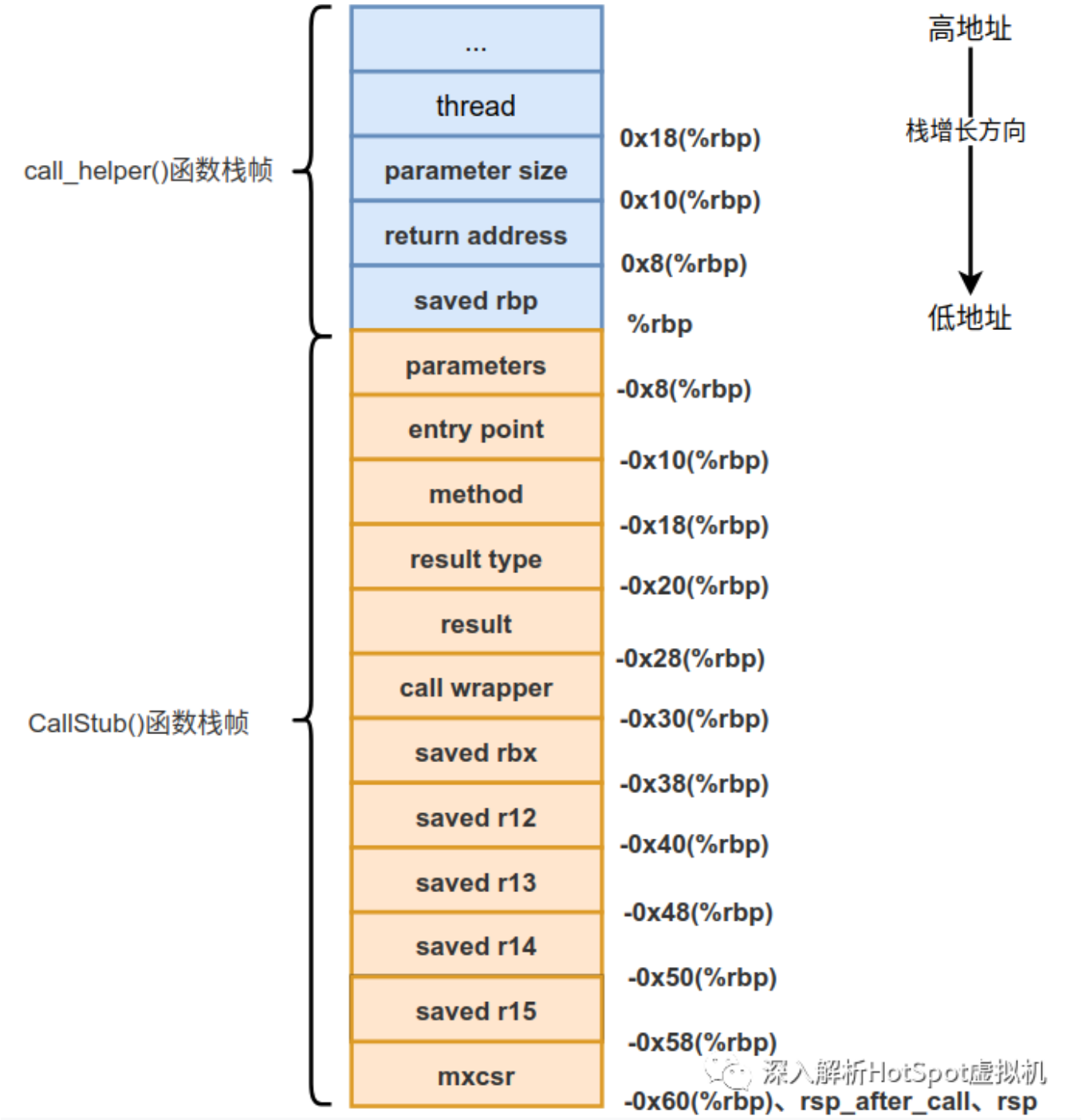
深入剖析Java虚拟机HotSpot

对HotSpot VM进行深度源码剖析，如果要系统的学习相关内容，推荐作者的《深入剖析Ja...

84篇原创内容

公众号

在 第3篇-CallStub新栈帧的创建 中我们介绍了generate_call_stub()函数的部分实现，完成了向CallStub栈帧中压入参数的操作，此时的栈的状态如下图所示。



继续看generate_call_stub()函数的实现，接来下会加载线程寄存器，代码如下：

```
__ movptr(r15_thread, thread);

__ reinit_heapbase();
```

生成的汇编代码如下：

```
mov    0x18(%rbp),%r15

mov    0x1764212b(%rip),%r12
```

对照着上面的栈图看一下0x18(%rbp)这个位置存储的是thread，将这个参数存储到%r15寄存器中。

如果在调用Java方法时有参数需要传递时，还需要传递参数，代码如下：

```
// pass parameters if any
BLOCK_COMMENT("pass parameters if any");
Label parameters_done;
// parameter_size拷贝到c_rarg3即rcx寄存器中
__ movl(c_rarg3, parameter_size);
// 校验c_rarg3的数值是否合法。两操作数作与运
// 算,仅修改标志位,不回送结果
__ testl(c_rarg3, c_rarg3);
// 如果不合法则跳转到parameters_done分支上
__ jcc(Assembler::zero, parameters_done);


// 如果执行下面的逻辑，那么就表示
// parameter_size的值不为0,
// 也就是需要为调用的java方法提供参数
Label loop;
// 将地址parameters包含的数据即参数对象
// 的指针拷贝到c_rarg2寄存器中
__ movptr(c_rarg2, parameters);
// 将c_rarg3中值拷贝到c_rarg1中,
// 即将参数个数复制到c_rarg1中
__ movl(c_rarg1, c_rarg3);
__ BIND(loop);
// 将c_rarg2指向的内存中包含的地址复制到rax中
__ movptr(rax, Address(c_rarg2, 0));
// c_rarg2中的参数对象的指针加上指针
// 宽度8字节，即指向下一个参数
__ addptr(c_rarg2, wordSize);
// 将c_rarg1中的值减一
__ decrementl(c_rarg1);
// 传递方法调用参数
__ push(rax);
// 如果参数个数大于0则跳转到loop继续
__ jcc(Assembler::notZero, loop);
```

这里是个循环，用于传递参数，相当于如下代码：

```
while(%esi){
 rax = *arg
 push_arg(rax)
```

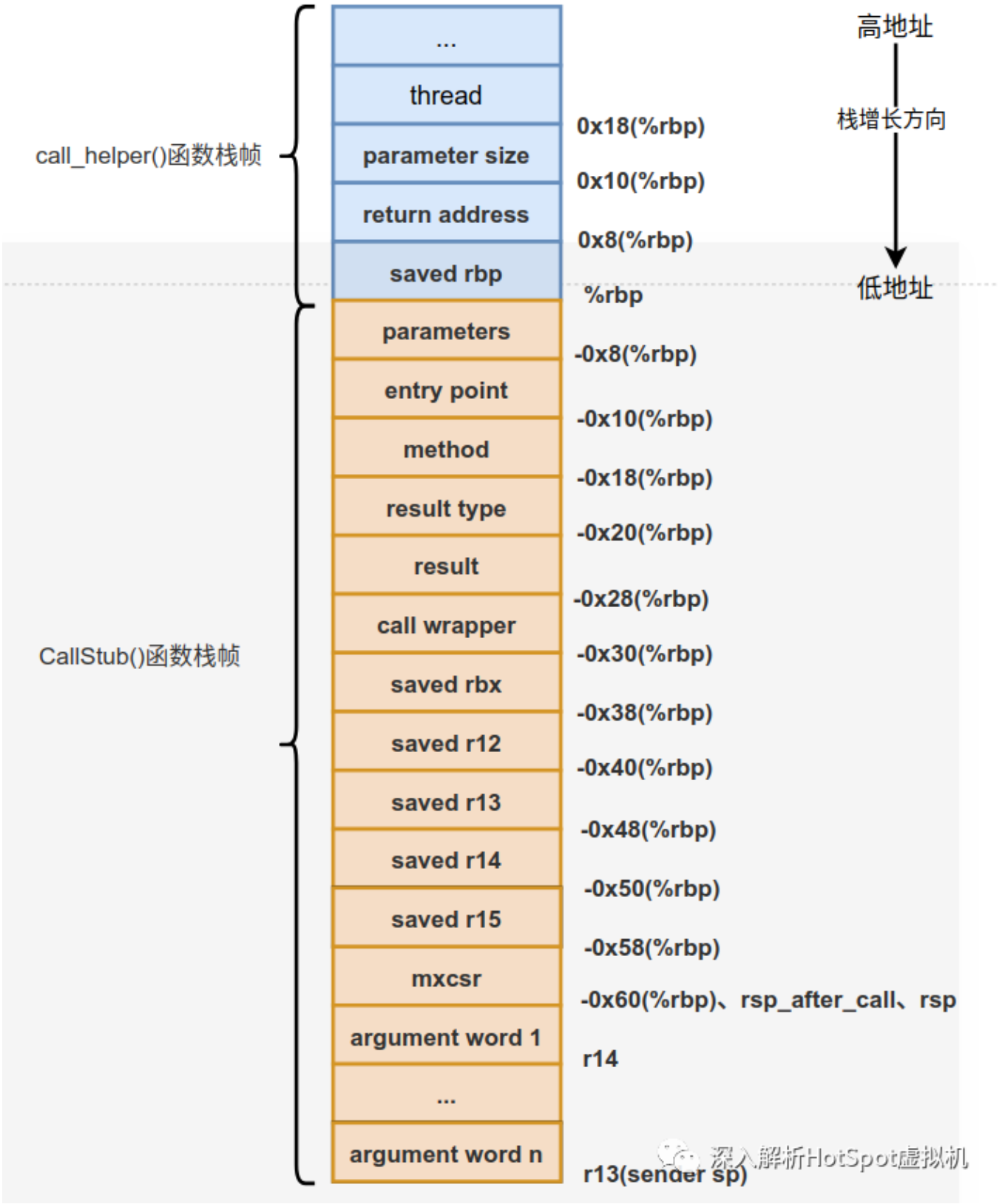
```
 arg++; // ptr++
 %esi--; // counter--
}
```

生成的汇编代码如下：

```
// 将栈中parameter size送到%ecx中
mov 0x10(%rbp),%ecx
// 做与运算，只有当%ecx中的值为
// 0时才等于0
test %ecx,%ecx
// 没有参数需要传递，直接跳转
// 到parameters_done即可
je 0x00007fdf4500079a


// -- loop --
// 汇编执行到这里，说明
// paramter size不为0,需要传递参数
mov -0x8(%rbp),%rdx
mov %ecx,%esi
mov (%rdx),%rax
add $0x8,%rdx
dec %esipush %rax
// 跳转到loop
jne 0x00007fdf4500078e
```

因为要调用Java方法，所以会为Java方法压入实际的参数，也就是压入parameter size个从parameters开始取的参数。压入参数后的栈帧状态如下图所示。



当把需要调用Java方法的参数准备就绪后，接下来就会调用Java方法。这里需要重点提示一下Java解释执行时的方法调用约定，不像C/C++在x86下的调用约定一样，不需要通过寄存器来传递参数，而是通过栈来传递参数的，说的更直白一些，是通过局部变量表来传递参数的，所以上图CallStub()函数栈帧中的argument word1 ... argument word n其实是被调用的Java方法局部变量表的一部分。

下面接着看调用Java方法的代码，如下：

```
// 调用Java方法
// -- parameters_done --
__ BIND(parameters_done);
// 将method地址包含的数据接Method*拷贝到rbx中
__ movptr(rbx, method);
// 将解释器的入口地址拷贝到c_rarg1寄存器中
__ movptr(c_rarg1, entry_point);
// 将rsp寄存器的数据拷贝到r13寄存器中
__ mov(r13, rsp); // set sender sp
BLOCK_COMMENT("call Java function");
// 调用解释器的解释函数，从而调用Java方法
// 调用的时候传递c_rarg1，也就是解释器的入口地址
__ call(c_rarg1);
```

生成的汇编代码如下：

```
// 将Method*送到%rbx中
mov     -0x18(%rbp),%rbx
// 将entry_point送到%rsi中
mov     -0x10(%rbp),%rsi
// 将调用者的栈顶指针保存到%r13中
mov     %rsp,%r13
// 调用Java方法
callq   *%rsi
```

注意调用callq指令后，会将callq指令的下一条指令的地址压栈，再跳转到第1操作数指定的地址，也就是*%rsi表示的地址。压入下一条指令的地址是为了让函数能通过跳转到栈上的地址从子函数返回。

callq指令调用的是entry point。entry point在后面会详细介绍。