你写过超过2500行的方法么？通常来说，这么大的方法并不多见，一般都是使用机器辅助生成的为主，这种情况在模板编译或其它语言的自动转换中比较常见。例如，对一个复杂的JSP页面，机器有可能会为它生成一个复杂的servlet方法去实现。

然而在Hotspot上运行这种大方法，很可能会有性能问题。例如，把文章所附DEMO的play()方法的内容分别重复拷贝1、2、4、8、16、32次并依次运行，在我的机器（Hotspot\_1.6u22/Windows）上得到的play()的执行消耗时间分别是28.43、54.72、106.28、214.41、419.30、1476.40毫秒/万次。在重复拷贝1～16次时，随着代码量增加，方法执行所消耗的时间也对应成倍增加。当重复拷贝32次时，方法却多消耗了80%的时间。如果把这个play()方法拆分成play1()和play2()，让它们的方法体都是16次的重复拷贝，play1()最后再调用play2()，那么，play1()+play2()的执行消耗时间是857.75毫秒/万次，恰好是之前重复拷贝16次所消耗的时间的两倍。**为什么同样功能的一段代码放在一个方法中执行会变慢，拆分成两个方法就变快？**

大家知道，JVM一开始是以解释方式执行字节码的。当这段代码被执行的次数足够多以后，它会被动态优化并编译成机器码执行，执行速度会大大加快，这就是所谓的JIT编译。DEMO的play()方法在被统计消耗时间之前，已经预热执行了2000次，满足ClientVM的方法JIT编译阈值CompileThreshold=1500次的要求，那么，它是不是真的被JIT编译了呢？我们可以增加VM参数”-XX:+PrintCompilation”调查一下。在+PrintCompilation打开以后，列出了JVM在运行时进行过JIT编译的方法。下面是经过32次重复拷贝的play()方法的JIT编译记录（只列出需要关心的部分）：

34       HugeMethodDemo::buildTheWorld (184 bytes)

39       HugeMethodDemo::run (59 bytes)

而分成两部分的play1()+plaay2()的JIT编译记录则为：

**18       HugeMethodDemo::play1 (4999 bytes)**

**19       HugeMethodDemo::play2 (4993 bytes)**

36       HugeMethodDemo::buildTheWorld (184 bytes)

41       HugeMethodDemo::run (59 bytes)

显然，经过重复拷贝32次的play()方法没有经过JIT编译，始终采用解释方式执行，而分拆开的play1()+play2()经过JIT编译，所以难怪play()要慢80%。

为什么play()方法不受JVM青睐呢，是太长了么？这只能到Hotspot源码中去翻答案了。在[compilationPolicy.cpp](http://hg.openjdk.java.net/jdk6/jdk6/hotspot/file/2ecd0d1194d2/src/share/vm/runtime/compilationPolicy.cpp)中有写道：

// Returns true if m is allowed to be compiled

bool CompilationPolicy::canBeCompiled(methodHandle m) {

if (m->is\_abstract()) return false;

if (DontCompileHugeMethods && m->code\_size() > HugeMethodLimit) return false;

// Math intrinsics should never be compiled as this can lead to

// monotonicity problems because the interpreter will prefer the

// compiled code to the intrinsic version. This can't happen in

// production because the invocation counter can't be incremented

// but we shouldn't expose the system to this problem in testing

// modes.

if (!AbstractInterpreter::can\_be\_compiled(m)) {

return false;

}

return !m->is\_not\_compilable();

}

当DontCompileHugeMethods=true且代码长度大于HugeMethodLimit时，方法不会被编译。DontCompileHugeMethods与HugeMethodLimit的值在[globals.hpp](http://hg.openjdk.java.net/jdk6/jdk6/hotspot/file/2ecd0d1194d2/src/share/vm/runtime/globals.hpp)中定义：

product(bool, DontCompileHugeMethods, true,

"don't compile methods > HugeMethodLimit")

develop(intx, HugeMethodLimit,  8000,

"don't compile methods larger than this if +DontCompileHugeMethods")

上面两个参数说明了Hotspot对字节码超过8000字节的大方法有JIT编译限制，这就是play()杯具的原因。由于使用的是product mode的JRE，我们只能尝试关闭DontCompileHugeMethods，即增加VM参数”-XX:-DontCompileHugeMethods”来强迫JVM编译play()。再次对play()进行测试，耗时855毫秒/万次，性能终于上来了，输出的JIT编译记录也增加了一行：

16       HugeMethodDemo::play (9985 bytes)

使用”-XX:-DontCompileHugeMethods”解除大方法的编译限制，一个比较明显的缺点是JVM会尝试编译所遇到的所有大方法，者会使JIT编译任务负担更重，而且需要占用更多的Code Cache区域去保存编译后的代码。但是优点是编译后可以让大方法的执行速度变快，且[可能提高GC速度](http://blogs.sun.com/jonthecollector/entry/did_you_know)。运行时Code Cache的使用量可以通过JMX或者JConsole获得，Code Cache的大小在[globals.hpp](http://hg.openjdk.java.net/jdk6/jdk6/hotspot/file/2ecd0d1194d2/src/share/vm/runtime/globals.hpp)中定义：

define\_pd\_global(intx, ReservedCodeCacheSize, 48\*M);

product\_pd(uintx, InitialCodeCacheSize, "Initial code cache size (in bytes)")

product\_pd(uintx, ReservedCodeCacheSize, "Reserved code cache size (in bytes) - maximum code cache size")

product(uintx, CodeCacheMinimumFreeSpace, 500\*K, "When less than X space left, we stop compiling.")

一旦Code Cache满了，HotSpot会停止所有后续的编译任务，虽然已编译的代码不受影响，但是后面的所有方法都会强制停留在纯解释模式。因此，如非必要，应该尽量避免生成大方法；如果解除了大方法的编译限制，则要留意配置Code Cache区的大小，准备更多空间存放编译后的代码。

最后附上DEMO代码：

|  |  |
| --- | --- |
| import java.io.StringWriter; | |
| import java.io.Writer; |

|  |  |
| --- | --- |
| import java.util.HashMap; | |
| import java.util.Map; |

|  |
| --- |
|  |
| public class HugeMethodDemo { | |

|  |
| --- |
|  |
| public static void main(String[] args) throws Exception { | |

|  |  |
| --- | --- |
| HugeMethodDemo demo = new HugeMethodDemo(); | |
|  |

|  |  |
| --- | --- |
| int warmup = 2000; | |
| demo.run(warmup); |

|  |
| --- |
|  |
| int loop = 200000; | |

|  |
| --- |
| double total = demo.run(loop); |
| double avg = total / loop / 1e6 \* 1e4; | |

|  |
| --- |
|  |
| System.out.println(String.format( | |

|  |  |
| --- | --- |
| "Loop=%d次, " + "avg=%.2f毫秒/万次", loop, avg)); | |
| } |

|  |
| --- |
|  |
| private long run(int loop) throws Exception { | |

|  |  |
| --- | --- |
| long total = 0L; | |
|  |

|  |
| --- |
| for (int i = 0; i < loop; i++) { |
| Map theWorld = buildTheWorld(); | |

|  |  |
| --- | --- |
| StringWriter console = new StringWriter(); | |
|  |

|  |  |
| --- | --- |
| long start = System.nanoTime(); | |
| play(theWorld, console); |

|  |  |
| --- | --- |
| long end = System.nanoTime(); | |
| total += (end - start); |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| return total; | |
| } |

|  |
| --- |
|  |
| private Map buildTheWorld() { | |

|  |  |
| --- | --- |
| Map context = new HashMap(); | |
| context.put("name", "D&D"); |

|  |  |
| --- | --- |
| context.put("version", "1.0"); | |
|  |

|  |
| --- |
| Map game = new HashMap(); |
| context.put("game", game); | |

|  |
| --- |
|  |
| Map player = new HashMap(); | |

|  |  |
| --- | --- |
| game.put("player", player); | |
| player.put("level", "26"); |

|  |
| --- |
| player.put("name", "jifeng"); |
| player.put("job", "paladin"); |

|  |  |
| --- | --- |
| player.put("address", "heaven"); | |
| player.put("weapon", "sword"); |

|  |  |
| --- | --- |
| player.put("hp", 150); | |
|  |

|  |  |
| --- | --- |
| String[] bag = new String[] { "world\_map", "dagger", | |
| "magic\_1", "potion\_1", "postion\_2", "key" }; |

|  |  |
| --- | --- |
| player.put("bag", bag); | |
| return context; |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| private void play(Map theWorld, Writer console) throws Exception { | |
| // 重复拷贝的开始位置 |

|  |
| --- |
| if (true) { |
| String name = String.valueOf(theWorld.get("name")); | |

|  |
| --- |
| String version = String.valueOf(theWorld.get("version")); |
| console.append("Game ").append(name).append(" (v").append(version).append(")\n"); | |

|  |  |
| --- | --- |
| Map game = (Map) theWorld.get("game"); | |
| if (game != null) { |

|  |  |
| --- | --- |
| Map player = (Map) game.get("player"); | |
| if (player != null) { |

|  |  |
| --- | --- |
| String level = String.valueOf(player.get("level")); | |
| String job = String.valueOf(player.get("job")); |

|  |  |
| --- | --- |
| String address = String.valueOf(player.get("address")); | |
| String weapon = String.valueOf(player.get("weapon")); |

|  |
| --- |
| String hp = String.valueOf(player.get("hp")); |
| console.append("  You are a ").append(level).append(" level ").append(job) | |

|  |
| --- |
| .append(" from ").append(address).append(". \n"); |
| console.append("  Currently you have a ").append(weapon).append(" in hand, ") | |

|  |  |
| --- | --- |
| .append("your hp: ").append(hp).append(". \n"); | |
| console.append("  Here are items in your bag: \n"); |

|  |
| --- |
| for (String item : (String[]) player.get("bag")) { |
| console.append("     \* ").append(item).append("\n"); | |

|  |
| --- |
| } |
| } else { | |

|  |  |
| --- | --- |
| console.append("\tPlayer not login.\n"); | |
| } |

|  |
| --- |
| } else { |
| console.append("\tGame not start yet.\n"); | |

|  |  |
| --- | --- |
| } | |
| } |

|  |  |
| --- | --- |
| // 重复拷贝的结束位置 | |
| } |

|  |
| --- |
| } |

[/java]