inflationThreshold 阈值，默认等于 15。当反射方法调用超过 15 次时（从 0 开始计算），会使用 ASM 生成新类，保证后面的调用比 native 要快。调用次数小于 15 次的情况下，直接用 native 的方式来调用，没有额外类的生成、校验、加载的开销。这种方式被称为 inflation 机制。

JVM 与 inflation 相关的属性有两个，一个是刚提到的阈值 sun.reflect.inflationThreshold，还有一个是是否禁用 inflation 的属性 sun.reflect.noInflation，默认值为 false。如果把 sun.reflect.noInflation 这个值设置成 true，那么从第 0 次开始就使用动态生成类的方式来调用反射方法了，而不会使用 native 的方式。增加 noInflation 选项重新执行上述 Java 代码，如下所示。

```
java -cp . -Dsun.reflect.noInflation=true ReflectionTest
```

输出结果如下所示。

```
java.lang.Exception: test#0
        at ReflectionTest.foo(ReflectionTest.java:10)
        at sun.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at ReflectionTest.main(ReflectionTest.java:18)
java.lang.Exception: test#1
        at ReflectionTest.foo(ReflectionTest.java:10)
        at sun.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at ReflectionTest.main(ReflectionTest.java:18)
```

可以看到，从第 0 次开始就已经没有使用 native 方法来调用反射方法了。

## 3.6 小结

这一章介绍了字节码稍微复杂一点的知识，我们来回顾一下重点知识：首先详细介绍了方法调用的 5 条指令的联系和区别，随后重点介绍了 invokedynamic 指令和在 Lambda 表达式上的应用，最后从字节码的角度分析了泛型擦除、synchronized 关键字和反射的原理。下一章我们会深入剖析 javac 编译原理的内容。