

第6讲 | 动态代理是基于什么原理？

2018-05-17 杨晓峰

Java核心技术36讲

[进入课程 >](#)



讲述：黄洲君

时长 11:24 大小 5.23M



编程语言通常有各种不同的分类角度，动态类型和静态类型就是其中一种分类角度，简单区分就是语言类型信息是在运行时检查，还是编译期检查。

与其近似的还有一个对比，就是所谓强类型和弱类型，就是不同类型变量赋值时，是否需要显式地（强制）进行类型转换。

那么，如何分类 Java 语言呢？通常认为，Java 是静态的强类型语言，但是因为提供了类似反射等机制，也具备了部分动态类型语言的能力。

言归正传，今天我要问你的问题是，[谈谈 Java 反射机制，动态代理是基于什么原理？](#)

典型回答

反射机制是 Java 语言提供的一种基础功能，赋予程序在运行时**自省**（introspect，官方用语）的能力。通过反射我们可以直接操作类或者对象，比如获取某个对象的类定义，获取类声明的属性和方法，调用方法或者构造对象，甚至可以运行时修改类定义。

动态代理是一种方便运行时动态构建代理、动态处理代理方法调用的机制，很多场景都是利用类似机制做到的，比如用来包装 RPC 调用、面向切面的编程（AOP）。

实现动态代理的方式很多，比如 JDK 自身提供的动态代理，就是主要利用了上面提到的反射机制。还有其他的实现方式，比如利用传说中更高性能的字节码操作机制，类似 ASM、cglib（基于 ASM）、Javassist 等。

考点分析

这个题目给我的第一印象是稍微有点诱导的嫌疑，可能会下意识地以为动态代理就是利用反射机制实现的，这么说也不算错但稍微有些不全面。功能才是目的，实现的方法有很多。总的来说，这道题目考察的是 Java 语言的另外一种基础机制：反射，它就像是一种魔法，引入运行时自省能力，赋予了 Java 语言令人意外的活力，通过运行时操作元数据或对象，Java 可以灵活地操作运行时才能确定的信息。而动态代理，则是延伸出来的一种广泛应用于产品开发中的技术，很多繁琐的重复编程，都可以被动态代理机制优雅地解决。

从考察知识点的角度，这道题涉及的知识点比较庞杂，所以面试官能够扩展或者深挖的内容非常多，比如：

考察你对反射机制的了解和掌握程度。

动态代理解决了什么问题，在你业务系统中的应用场景是什么？

JDK 动态代理在设计和实现上与 cglib 等方式有什么不同，进而如何取舍？

这些考点似乎不是短短一篇文章能够囊括的，我会在知识扩展部分尽量梳理一下。

知识扩展

1. 反射机制及其演进

对于 Java 语言的反射机制本身，如果你去看一下 java.lang 或 java.lang.reflect 包下的相关抽象，就会有一个很直观的印象了。Class、Field、Method、Constructor 等，这些完


全就是我们去操作类和对象的元数据对应。反射各种典型用例的编程，相信有太多文章或书籍进行过详细的介绍，我就不再赘述了，至少你需要掌握基本场景编程，这里是官方提供的参考文档：<https://docs.oracle.com/javase/tutorial/reflect/index.html>。

关于反射，有一点我需要特意提一下，就是反射提供的 `AccessibleObject.setAccessible` (boolean flag)。它的子类也大都重写了这个方法，这里的所谓 accessible 可以理解成修饰成员的 `public`、`protected`、`private`，这意味着我们可以在运行时修改成员访问限制！

`setAccessible` 的应用场景非常普遍，遍布我们的日常开发、测试、依赖注入等各种框架中。比如，在 O/R Mapping 框架中，我们为一个 Java 实体对象，运行时自动生成 setter、getter 的逻辑，这是加载或者持久化数据非常必要的，框架通常可以利用反射做这个事情，而不需要开发者手动写类似的重复代码。


另一个典型场景就是绕过 API 访问控制。我们日常开发时可能被迫要调用内部 API 去做些事情，比如，自定义的高性能 NIO 框架需要显式地释放 `DirectBuffer`，使用反射绕开限制是一种常见办法。

但是，在 Java 9 以后，这个方法的使用可能会存在一些争议，因为 Jigsaw 项目新增的模块化系统，出于强封装性的考虑，对反射访问进行了限制。Jigsaw 引入了所谓 Open 的概念，只有当被反射操作的模块和指定的包对反射调用者模块 Open，才能使用 `setAccessible`；否则，被认为是不合法 (illegal) 操作。如果我们的实体类是定义在模块里面，我们需要在模块描述符中明确声明：

 复制代码

```
1 module MyEntities {
2     // Open for reflection
3     opens com.mycorp to java.persistence;
4 }
```

因为反射机制使用广泛，根据社区讨论，目前，Java 9 仍然保留了兼容 Java 8 的行为，但是很有可能在未来版本，完全启用前面提到的针对 `setAccessible` 的限制，即只有当被反射操作的模块和指定的包对反射调用者模块 Open，才能使用 `setAccessible`，我们可以使用下面参数显式设置。

 复制代码

```
1 --illegal-access={ permit | warn | deny }
```

2. 动态代理


前面的问题问到了动态代理，我们一起看看，它到底是解决什么问题？

首先，它是一个**代理机制**。如果熟悉设计模式中的代理模式，我们会知道，代理可以看作是对调用目标的一个包装，这样我们对目标代码的调用不是直接发生的，而是通过代理完成。其实很多动态代理场景，我认为也可以看作是装饰器（Decorator）模式的应用，我会在后面的专栏设计模式主题予以补充。

通过代理可以让调用者与实现者之间**解耦**。比如进行 RPC 调用，框架内部的寻址、序列化、反序列化等，对于调用者往往是没有太大意义的，通过代理，可以提供更加友善的界面。

代理的发展经历了静态到动态的过程，源于静态代理引入的额外工作。类似早期的 RMI 之类古董技术，还需要 rmic 之类工具生成静态 stub 等各种文件，增加了很多繁琐的准备工作，而这又和我们的业务逻辑没有关系。利用动态代理机制，相应的 stub 等类，可以在运行时生成，对应的调用操作也是动态完成，极大地提高了我们的生产力。改进后的 RMI 已经不再需要手动去准备这些了，虽然它仍然是相对古老落后的技术，未来也许会逐步被移除。

这么说可能不够直观，我们可以看 JDK 动态代理的一个简单例子。下面只是加了一句 print，在生产系统中，我们可以轻松扩展类似逻辑进行诊断、限流等。

 复制代码

```
1 public class MyDynamicProxy {
2     public static void main (String[] args) {
3         HelloImpl hello = new HelloImpl();
4         MyInvocationHandler handler = new MyInvocationHandler(hello);
5         // 构造代码实例
6         Hello proxyHello = (Hello) Proxy.newProxyInstance(HelloImpl.class.getClassLoader(),
7             new Class[] { Hello.class }, handler);
8         proxyHello.sayHello();
9     }
10 }
11 interface Hello {
12     void sayHello();
13 }
```

```
13 }
14 class HelloImpl implements Hello {
15     @Override
16     public void sayHello() {
17         System.out.println("Hello World");
18     }
19 }
20 class MyInvocationHandler implements InvocationHandler {
21     private Object target;
22     public MyInvocationHandler(Object target) {
23         this.target = target;
24     }
25     @Override
26     public Object invoke(Object proxy, Method method, Object[] args)
27         throws Throwable {
28         System.out.println("Invoking sayHello");
29         Object result = method.invoke(target, args);
30         return result;
31     }
32 }
33
```

上面的 JDK Proxy 例子，非常简单地实现了动态代理的构建和代理操作。首先，实现对应的 `InvocationHandler`；然后，以接口 `Hello` 为纽带，为被调用目标构建代理对象，进而应用程序就可以使用代理对象间接运行调用目标的逻辑，代理为应用插入额外逻辑（这里是 `println`）提供了便利的入口。

从 API 设计和实现的角度，这种实现仍然有局限性，因为它是以接口为中心的，相当于添加了一种对于被调用者没有太大意义的限制。我们实例化的是 Proxy 对象，而不是真正的被调用类型，这在实践中还是可能带来各种不便和能力退化。

如果被调用者没有实现接口，而我们还是希望利用动态代理机制，那么可以考虑其他方式。我们知道 Spring AOP 支持两种模式的动态代理，JDK Proxy 或者 cglib，如果我们选择 cglib 方式，你会发现对接口的依赖被克服了。

cglib 动态代理采取的是创建目标类的子类的方式，因为是子类化，我们可以达到近似使用被调用者本身的效果。在 Spring 编程中，框架通常会处理这种情况，当然我们也可以[显式指定](#)。关于类似方案的实现细节，我就不再详细讨论了。

那我们在开发中怎样选择呢？我来简单对比下两种方式各自优势。

JDK Proxy 的优势：

最小化依赖关系，减少依赖意味着简化开发和维护，JDK 本身的支持，可能比 cglib 更加可靠。

平滑进行 JDK 版本升级，而字节码类库通常需要进行更新以保证在新版 Java 上能够使用。

代码实现简单。

基于类似 cglib 框架的优势：

有的时候调用目标可能不便实现额外接口，从某种角度看，限定调用者实现接口是有些侵入性的实践，类似 cglib 动态代理就没有这种限制。

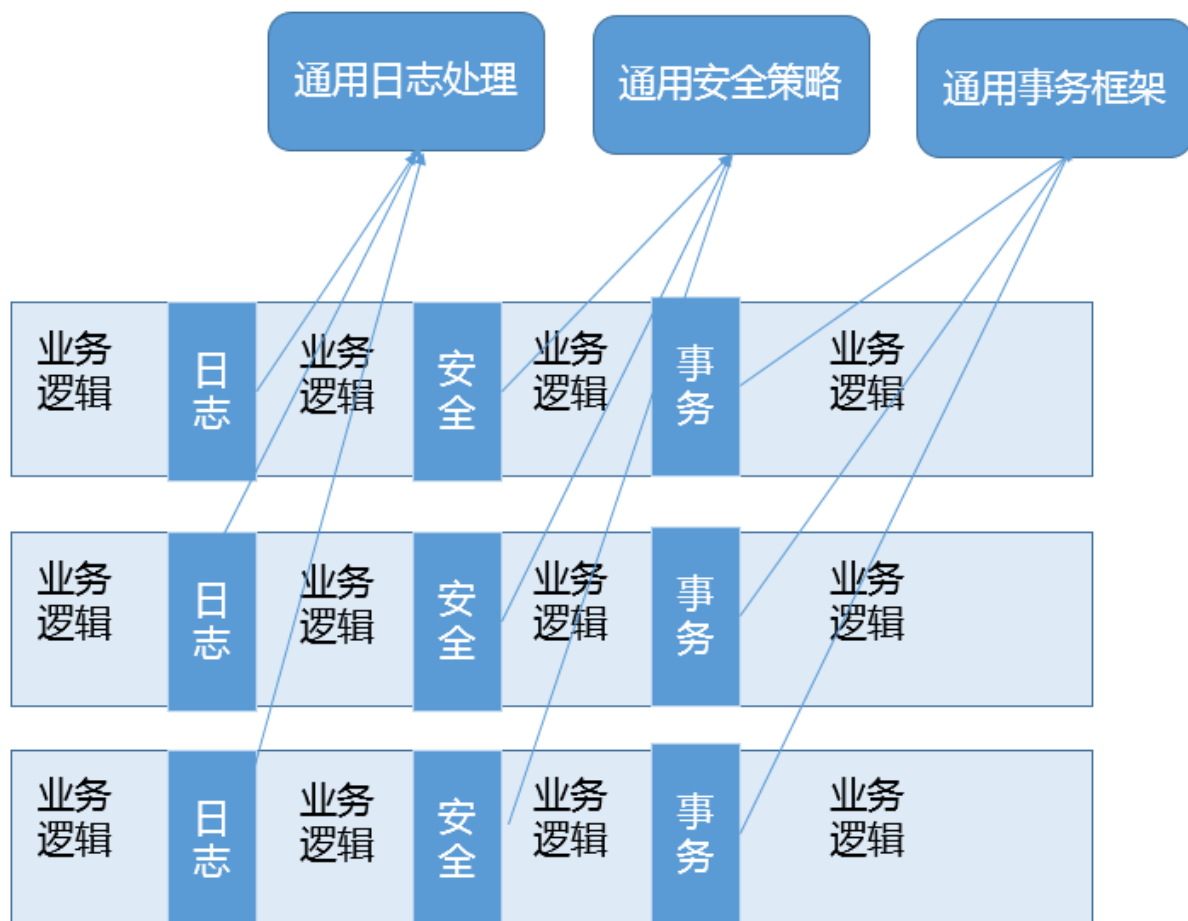
只操作我们关心的类，而不必为其他相关类增加工作量。

高性能。

另外，从性能角度，我想补充几句。记得有人曾经得出结论说 JDK Proxy 比 cglib 或者 Javassist 慢几十倍。坦白说，不去争论具体的 benchmark 细节，在主流 JDK 版本中，JDK Proxy 在典型场景可以提供对等的性能水平，数量级的差距基本上不是广泛存在的。而且，反射机制性能在现代 JDK 中，自身已经得到了极大的改进和优化，同时，JDK 很多功能也不完全是反射，同样使用了 ASM 进行字节码操作。

我们在选型中，性能未必是唯一考量，可靠性、可维护性、编程工作量等往往是更主要的考虑因素，毕竟标准类库和反射编程的门槛要低得多，代码量也是更加可控的，如果我们比较下不同开源项目在动态代理开发上的投入，也能看到这一点。

动态代理应用非常广泛，虽然最初多是因为 RPC 等使用进入我们视线，但是动态代理的使用场景远远不仅如此，它完美符合 Spring AOP 等切面编程。我在后面的专栏还会进一步详细分析 AOP 的目的和能力。简单来说它可以看作是对 OOP 的一个补充，因为 OOP 对于跨越不同对象或类的分散、纠缠逻辑表现力不够，比如在不同模块的特定阶段做一些事情，类似日志、用户鉴权、全局性异常处理、性能监控，甚至事务处理等，你可以参考下面这张图。



AOP 通过（动态）代理机制可以让开发者从这些繁琐事项中抽身出来，大幅度提高了代码的抽象程度和复用度。从逻辑上来说，我们在软件设计和实现中的类似代理，如 Facade、Observer 等很多设计目的，都可以通过动态代理优雅地实现。

今天我简要回顾了反射机制，谈了反射在 Java 语言演进中正在发生的变化，并且进一步探讨了动态代理机制和相关的切面编程，分析了其解决的问题，并探讨了生产实践中的选择考量。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？留一道思考题给你，你在工作中哪些场景使用到了动态代理？相应选择了什么实现技术？选择的依据是什么？

请你在留言区写写你对这个问题的思考，我会选出经过认真思考的留言，送给你一份学习鼓励金，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。

Java 核心技术36讲

—— 前 Oracle 首席工程师
带你修炼 Java 内功 ——

杨晓峰 前 Oracle 首席工程师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第5讲 | String、StringBuffer、StringBuilder有什么区别？

下一篇 第7讲 | int和Integer有什么区别？

精选留言 (58)

写留言



肖一林

2018-05-17

360

提一些建议：应该从两条线讲这个问题，一条从代理模式，一条从反射机制。不要老担心篇幅限制讲不清问题，废话砍掉一些，深层次的内在原理多讲些（比如asm），容易自学的扩展知识可以用链接代替

代理模式（通过代理静默地解决一些业务无关的问题，比如远程、安全、事务、日志、资源关闭.....让应用开发者可以只关心他的业务）...

展开 ▾

作者回复: 谢谢反馈，类似ASM这种字节码操纵是有单独章节覆盖的，前面基础篇有个整体印象，免得陷入细节；Java内部动态生成还有其他领域，比如Lambda实现机制，个人认为一起分析会连贯一些



公号-代码...

2018-05-17

👍 159

反射与动态代理原理

1 关于反射

反射最大的作用之一就在于我们可以不在编译时知道某个对象的类型，而在运行时通过提供完整的“包名+类名.class”得到。注意：不是在编译时，而是在运行时。...

展开 ▾



刘方杰

2018-05-18

👍 87

哎，阅读越来越困难了，我离核心是不是太远了。

展开 ▾



天使

2018-05-17

👍 81

没涉及到原理，proxy到底是如何实现的，运行时拦截？cglib是编译时对类进行织入？要是更细一些就好了



云学

2018-06-09

👍 32

看了好多篇文章，总体感觉是比较累，无论读者是否具有java背景，都应该让他看懂，而不是越看越糊涂，疑问反而更多了

展开 ▾

作者回复: 非常感谢，读者基础不同，我尽量兼顾并增加基础的介绍，因为也有反馈希望可以更全面、深入...

有好的建议请不吝赐教



jimforcode

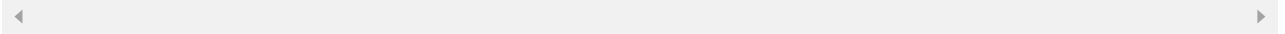
2018-05-17

👍 22

好像和啥原理没啥关系吧，总结来说就是jdk 自身的反射机制或用第三方库，哪哪看到的都

这样说，一笔带过

作者回复: 谢谢反馈，字节码操作、运行时拦截、加载期编织、Java agent等，会和Aop单独介绍，那些内容不是几句话说得完



ai~ wen

2018-05-21

👍 17

对于我这初学者，读着就懵了

展开 ▾



jimforcode

2018-05-17

👍 16

比较期待大神讲讲动态代理的原理是什么，对性能会带来什么影响，有没有什么问题需要规避，谢谢



言志

2018-05-17

👍 13

采用cglib方式的动态代理还有个缺点：不能应用到被代理对象的final方法上。
我在多数据源项目中自动切换数据源功能用到了

展开 ▾



bigfish

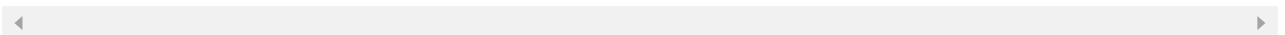
2018-07-08

👍 9

本来资质愚笨，看不懂很多东西的原理.想进来学习一下，jdk动态代理的原理和cglib代理原理等一些原理性的东西（其他章节也是如此）.发现听到原理性的东西不多都是一带而过.其实您做的课件针对很多点都是一带而过，听到某个名词一下来了兴趣继续一听结束了.我们都知道Java很大很多可以研究的，其实我们想听的很多是一些点的原理，讲完原理在结合实际应用阐述一下.也许我们就会有些豁然明朗的感觉.希望能理解一下！！！！

展开 ▾

作者回复: 谢谢反馈，后面类加载章节介绍了两者底层机制，照顾下不同基础的读者





雷赞

2018-05-22

👍 9

何为反射？

反射是指计算机程序在运行时可以访问、检测和修改它本身状态或行为的一种能力。比喻来说，反射就是程序在运行的时候能够“观察”并且修改自己的行为。

...

展开 ▾



约书亚

2018-05-17

👍 7

先回答问题：

99%的Java程序员应该都间接使用了AOP。自己项目里直接编写的，比如调用追踪，通用日志，自动重试

反射和AOP真是双刃剑效果拔群的技术。从MVC开始约定胜过配置的开发理念大行其道，ORM自动映射，plugin模式，到现在的spring + boot + cloud 的declarative编程大量...

展开 ▾

作者回复: 1.基本如此；反射是相对保证类型安全的，我觉得要比较也是和methodhandle之类对比，那个更是接近jvm内部的黑盒，性能更好

2.你是说lambda？也是需要jvm生成call site，然后invokedynamic之类调用，所以首次调用开销明显，C#不了解，不过动态生成的感觉都是如此吧；

这东西目前没有cache，如果你说的是存储在文件系统；未来，嘿嘿.....

这些太零碎，说过了会有单独章节介绍，不然没基础的就晕了，还用不上

◀ ▶



正光

2018-08-04

👍 6

整体感觉讲的太浅，水上漂的感觉

展开 ▾

作者回复: 具体底层细节在jvm字节码操纵那一讲，难度把握不准请见谅

◀ ▶



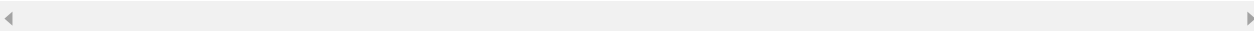
TWO STRIN...

2018-05-17

👍 6

老师可以在分享结束时推荐一些好的文章，书籍甚至演讲之类的么？

作者回复: 没问题，喜欢底层，去查查JavaOne，FOSDEM，jvmsummit等



灰飞灰猪不...

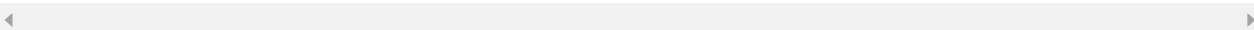
2018-05-17

👍 4

cglib是怎么实现对目标对象的拦截的呢？

展开 ▾

作者回复: 计划单独介绍



yoo

2018-08-08

👍 3

您好，我一直关注您的文章，但是每次都觉得不是很理解。可不可以在以后的文章中讲一些实际的应用场景。



Loren

2018-12-12

👍 2

我把文章中的代理的代码copy到本地运行，debug运行时我打了几个断点，运行后输出了很多次Invoking sayHello是为什么

展开 ▾



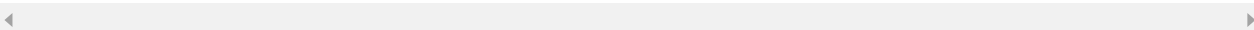
蒙奇D路飞

2018-05-28

👍 2

感觉细节层面缺少具体描述，希望后续对底层原理的描述更细致些~

作者回复: 谢谢反馈，有章节介绍类似字节码操纵之类底层技术，照顾不同基础





嗨!

2019-03-04

👍 1

比如，在 O/R Mapping 框架中，我们为一个 Java 实体对象，运行时自动生成 setter、getter 的逻辑，这是加载或者持久化数据非常必要的，框架通常可以利用反射做这个事情，而不需要开发者手动写类似的重复代码。

您好，请问利用反射自动生成get和set方法的实际工作场景是什么呢？我都是自己生成的啊，难道不生成也可以吗？

展开 ∨



godtrue

2018-12-13

👍 1

恩，如大部分同学的感受一样，不够细致，一些概念点到为止了

当然，对于开阔眼界是大大的帮助的，动态代理这个平时编码没直接写过，各种框架的实现是基本用到的，感觉都是针对字节码进行的操作，可能操作的方式有些不同。JAVA源码是写个人看的，是编译器的输入材料，Java字节码是准备给JVM来加载识别的，是机器码的待转换材料，主要是JVM来操作。...

展开 ∨