举个例子，Driver类和Car类。

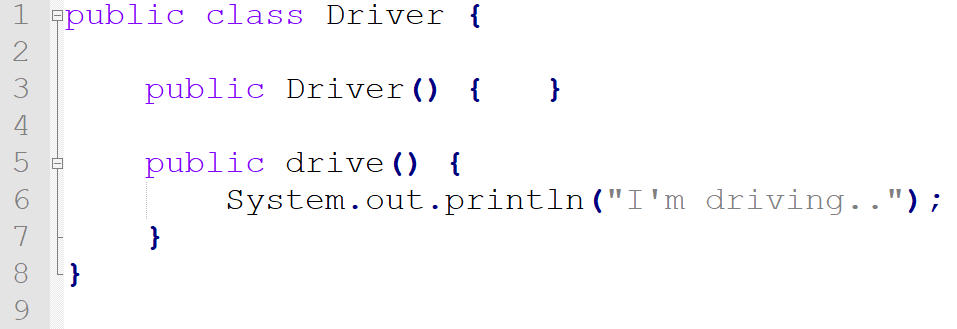


图4-1 Driver类代码

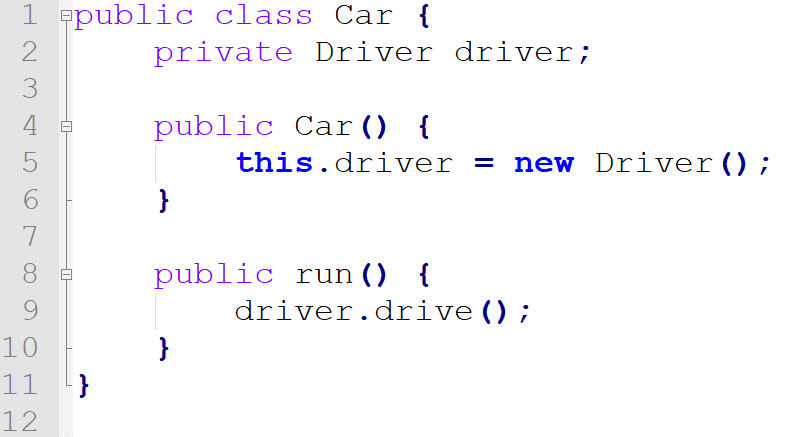


图4-2 Car类代码

Driver类中只有一个方法driver()。Car类中持有一个Driver类对象，一个run()方法。调用Car类的构造方法会实例化这个Dirver对象，调用run()方法则是直接调用Driver实例的drive()方法。

这是一个简单的例子，但已经可以说明问题。新建一个Car类对象（调用构造方法）的同时，Java虚拟机引擎会自动创建一个Driver类对象。Car类依赖Driver类而且每一个Car类的实例和一个固定的Driver实例紧紧耦合到了一起，我们甚至没有办法去改变这个Driver对象。

耦合具有两面性（two-headed beast）。一方面，紧密耦合的代码难以测试、难以复用、难以理解，并且典型地表现出“打地鼠”式的bug特性（修复一个bug，将会出现一个或者更多新的bug）。另一方面，一定程度的耦合又是必须的——完全没有耦合的代码什么也做不了。为了完成有实际意义的功能，不同的类必须以适当的方式进行交互。总而言之，耦合是必须的，但应当被小心谨慎地管理[]。

为了实现改变Driver对象的功能，新增一个方法实现其功能。

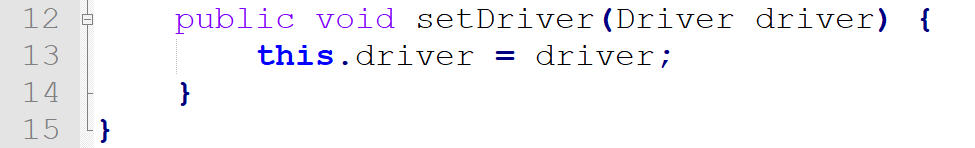


图4-3 为Car类添加的新方法

新增了一个setDriver()方法实现了此功能，这样整体减轻了耦合。还有第二种方法，改变构造方法：

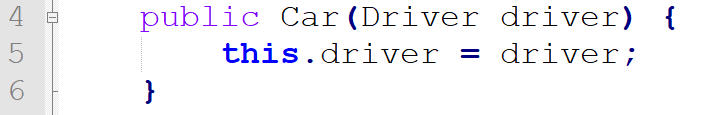


图4-4 Car类的新构造方法

这两种方法都可以减弱代码的耦合度，可以省去修改代码之苦。但是这两种方法还是需要我们手动生成Car实例，然后设置Driver实例。Spring在这两种方法的基础上，使用XML外部文件或者注解实现了完全的第三方注入。

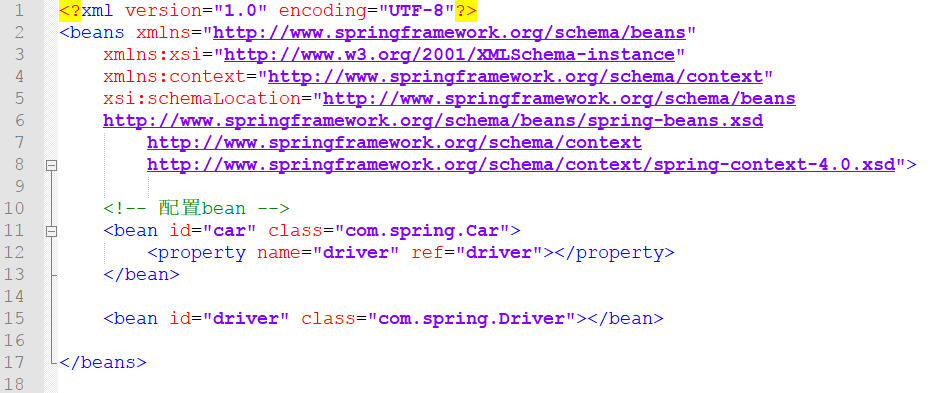


图4-5 使用XML配置注入

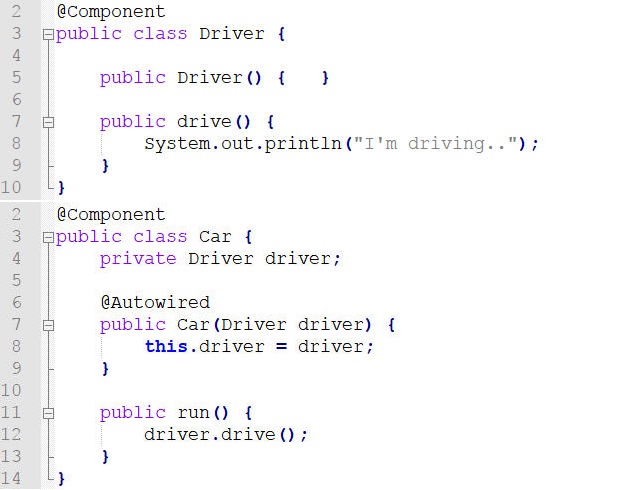


图4-6 使用注解配置注入

上面分别使用XML文件和注解（包括@Component和@Autowired）对类的实例对象进行了配置，之后无需再手动创建新的Car和Driver对象，只需使用Spring容器对应的API即可创建，完全实现了第三方注入，也就是“依赖注入”。