我们上一节通过了一个电影的例子形象的诠释了IoC的概念，但我们也知道Spring应用IoC是通过配置文件实现的，是没有直接用Java代码侵入的，那么它是如何做到的呢？这里我们先驻足温习一下Java的知识--反射，对，Spring就是通过反射实现这一点的。

类装载器ClassLoader的概要介绍

ClassLoader是个啥玩意呢？这么说吧，反射的入口就是从它开始的，通过它指定全限定类名装载到对应的反射实例，通俗的讲它能在Java运行时把一个类装入JVM。

JVM在运行时会产生3个ClassLoader：根装载器，ExtClassLoader（扩展类装载器），AppClassLoader（应用类装载器）。其中根装载器不是ClassLoader的子类，它采用C++编写，因而Java中也看不到它，它负责装载JRE的核心类库。ExtClassLoader和AppClassLoader都是ClassLoader的子类，其中ExtClassLoader负责装载JRE的扩展目录ext下的jar包，AppClassLoader则负责装载Classpath路径下的类包。

三者之间存在父子层级关系，大致关系如图：

通过代码我们来实际看一波这三者的关系，如图：

```
public class ClassLoaderTest {
    public static void main(String[] args) {
        ClassLoader loader = Thread.currentThread().getContextClassLoader();
        System.out.println("current loader:"+loader);
        System.out.println("parent loader:"+loader.getParent());
        System.out.println("grandparent loader:"+loader.getParent(). getParent());
    }
}
```

运行以上代码，在控制台上将打印出以下信息：

```
current loader:sun.misc.Launcher$AppClassLoader@131f71a
parent loader:sun.misc.Launcher$ExtClassLoader@15601ea
//①根装载器在Java中访问不到，所以返回null
grandparent loader:null
```

输出很清晰地展示了三者之间的层级，第三个为什么返回null，那是因为根装载器在Java中时无法访问的。

ClassLoader可以在JVM层面实现代码的侵入，那么又是如何保证安全性的呢？JVM早就考虑到了，在装载类时使用了“全盘负责委托机制”。

ClassLoader类常用方法介绍

Class loadClass(String name)：name参数需赋给一个需要装载类的全限定类名，见名知意，这个方法就是装载类时使用。

ClassLoader getParent()：用来获取装载器的父装载器。

Java反射的基本用法

Constructor：通过Class#getConstructor(null)可以获取无参构造函数，再通过Constructor#newInstance()就可以实例化出类对象了，不知道是jdk更新的原因还是啥，现在甚至可以不经过这一步，通过Class#newInstance()直接实例化对象。

Field：通过Class#getDeclaredField(String name)可以获取类的变量并通过set(Object obj,Object value)赋值，若私有变量为私有变量，还需setAccessible(true)取消语言访问检查，私有方法也是这般处理。

Method：通过Class#getDeclaredMethod(String name,Class clazz)获取方法并通过invoke(Object obj,Object obj2)赋参执行。