Spring 学习

Chapter1:

为了降低Java开发的复杂性，Spring采取了以下4种关键策略：

基于POJO的轻量级和最小侵入性编程；

通过依赖注入和面向接口实现松耦合；

基于切面和惯例进行声明式编程；

通过切面和模板减少样板式代码。

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。

通过DI，对象的依赖关系将由系统中负责协调各对象的第三方组件在创建对象的时候进行设定。对象无需自行创建或管理它们的依赖关系

依赖注入的方式之一：

构造器注入(constructor injection)

DI所带来的最大收益——松耦合。如果一个对象只通过接口（而不是具体实现或初始化过程）来表明依赖关系，那么这种依赖就能够在对象本身毫不知情的情况下，用不同的具体实现进行替换

创建应用组件之间协作的行为通常称为装配（wiring）Spring有多种装配bean的方式，采用XML是很常见的一种装配方式

Spring通过应用上下文（Application Context）装载bean的定义并把它们组装起来。Spring应用上下文全权负责对象的创建和组装。Spring自带了多种应用上下文的实现，它们之间主要的区别仅仅在于如何加载配置。

ClassPathXmlApplicationContext该类加载位于应用程序类路径下的一个或多个XML配置文件

DI能够让相互协作的软件组件保持松散耦合，而面向切面编程（aspect-orientedprogramming，AOP）允许你把遍布应用各处的功能分离出来形成可重用的组件。

面向切面编程往往被定义为促使软件系统实现关注点的分离一项技术。系统由许多不同的组件组成，每一个组件各负责一块特定功能。除了实现自身核心的功能之外，这些组件还经常承担着额外的职责。诸如日志、事务管理和安全这样的系统服务经常融入到自身具有核心业务逻辑的组件中去，这些系统服务通常被称为横切关注点，因为它们会跨越系统的多个组件。

如果将这些关注点分散到多个组件中去，你的代码将会带来双重的复杂性。实现系统关注点功能的代码将会重复出现在多个组件中。这意味着如果你要改变这些关注点的逻辑，必须修改各个模块中的相关实现。即使你把这些关注点抽象为一个独立的模块，其他模块只是调用它的方法，但方法的调用还是会重复出现在各个模块中。组件会因为那些与自身核心业务无关的代码而变得混乱。一个向地址簿增加地址条目的方法应该只关注如何添加地址，而不应该关注它是不是安全的或者是否需要支持事务。

AOP能够使这些服务模块化，并以声明的方式将它们应用到它们需要影响的组件中去。所造成的结果就是这些组件会具有更高的内聚性并且会更加关注自身的业务，完全不需要了解涉及系统服务所带来复杂性。总之，AOP能够确保POJO的简单性。

我们可以把切面想象为覆盖在很多组件之上的一个外壳。应用是由那些实现各自业务功能的模块组成的。借助AOP，可以使用各种功能层去包裹核心业务层。这些层以声明的方式灵活地应用到系统中，你的核心应用甚至根本不知道它们的存在。这是一个非常强大的理念，可以将安全、事务和日志关注点与核心业务逻辑相分离。

Spring旨在通过模板封装来消除样板式代码。Spring的JdbcTemplate使得执行数据库操作时，避免传统的JDBC样板代码成为了可能。

容器是Spring框架的核心。Spring容器使用DI管理构成应用的组件，它会创建相互协作的组件之间的关联。毫无疑问，这些对象更简单干净，更易于理解，更易于重用并且更易于进行单元测试。

Spring容器并不是只有一个。Spring自带了多个容器实现，可以归为两种不同的类型。bean工厂（由org.springframework. beans. factory.beanFactory接口定义）是最简单

的容器，提供基本的DI支持。应用上下文（由org.springframework.context.ApplicationContext接口定义）基于BeanFactory构建，并提供应用框架级别的服务，例如从属性文件解析文本信息以及发布应用事件给感兴趣的事件监听者。

Spring自带了多种类型的应用上下文。

AnnotationConfigApplicationContext：从一个或多个基于Java的配置类中加载Spring应用上下文。

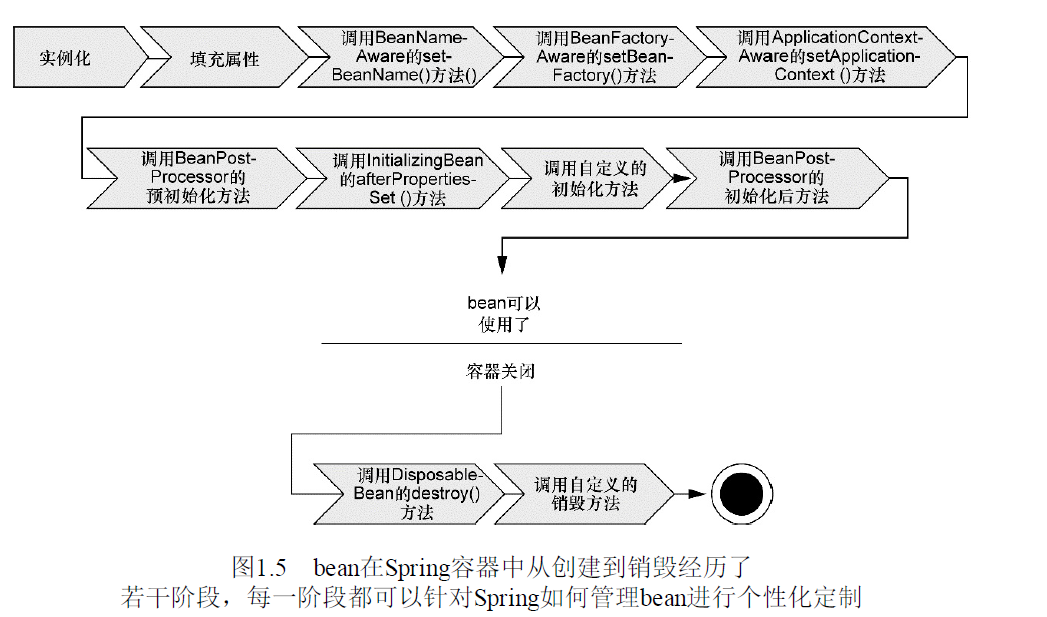
AnnotationConfigWebApplicationContext：从一个或多个基于Java的配置类中加载Spring Web应用上下文。

ClassPathXmlApplicationContext：从类路径下的一个或多个XML配置文件中加载上下文定义，把应用上下文的定义文件作为类资源。

FileSystemXmlapplicationcontext：从文件系统下的一个或多个XML配置文件中加载上下文定义。

XmlWebApplicationContext：从Web应用下的一个或多个XML配置文件中加载上下文定义。

Spring 的生命周期



1．Spring对bean进行实例化；

2．Spring将值和bean的引用注入到bean对应的属性中；

3．如果bean实现了BeanNameAware接口，Spring将bean的ID传递给setBeanName()方法；

4．如果bean实现了BeanFactoryAware接口，Spring将调用setBeanFactory()方法，将BeanFactory容器实例传入；

5．如果bean实现了ApplicationContextAware接口，Spring将调用setApplicationContext()方法，将bean所在的应用上下文的引用传入进来；

6．如果bean实现了BeanPostProcessor接口，Spring将调用它们的postProcessBeforeInitialization()方法；

7．如果bean实现了InitializingBean接口，Spring将调用它们的afterPropertiesSet()方法。类似地，如果bean使用init-method声明了初始化方法，该方法也会被调用；

8．如果bean实现了BeanPostProcessor接口，Spring将调用它们的postProcessAfterInitialization()方法；

9．此时，bean已经准备就绪，可以被应用程序使用了，它们将一直驻留在应用上下文中，

直到该应用上下文被销毁；

10．如果bean实现了DisposableBean接口，Spring将调用它的destroy()接口方法。同

样，如果bean使用destroy-method声明了销毁方法，该方法也会被调用。

Chapter2 装配bean

创建应用对象之间协作关系的行为通常称为装配（wiring），这也是依赖注入（DI）的本质

Spring 提供的三种主要的装配机制

在XML中进行显式配置。

在Java中进行显式配置。

隐式的bean发现机制和自动装配。

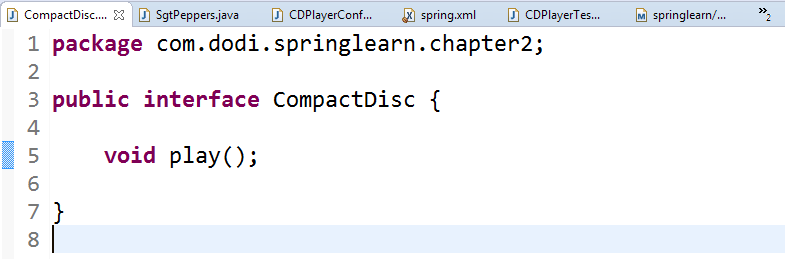
自动化装配bean

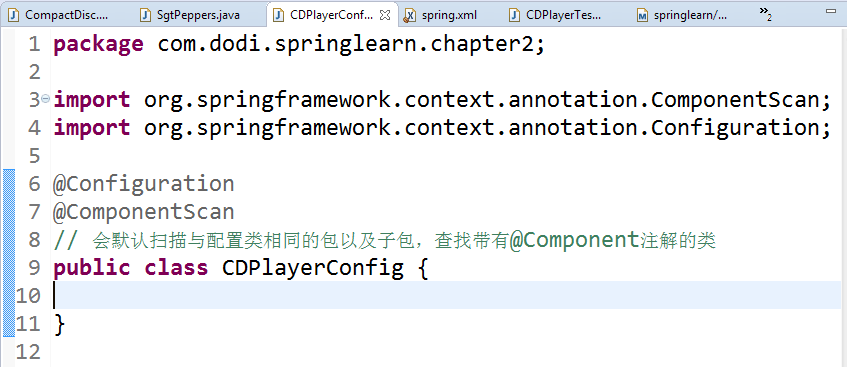
Spring从两个角度来实现自动化装配：

组件扫描（component scanning）：Spring会自动发现应用上下文中所创建的bean。

自动装配（autowiring）：Spring自动满足bean之间的依赖。

程序清单





@Component

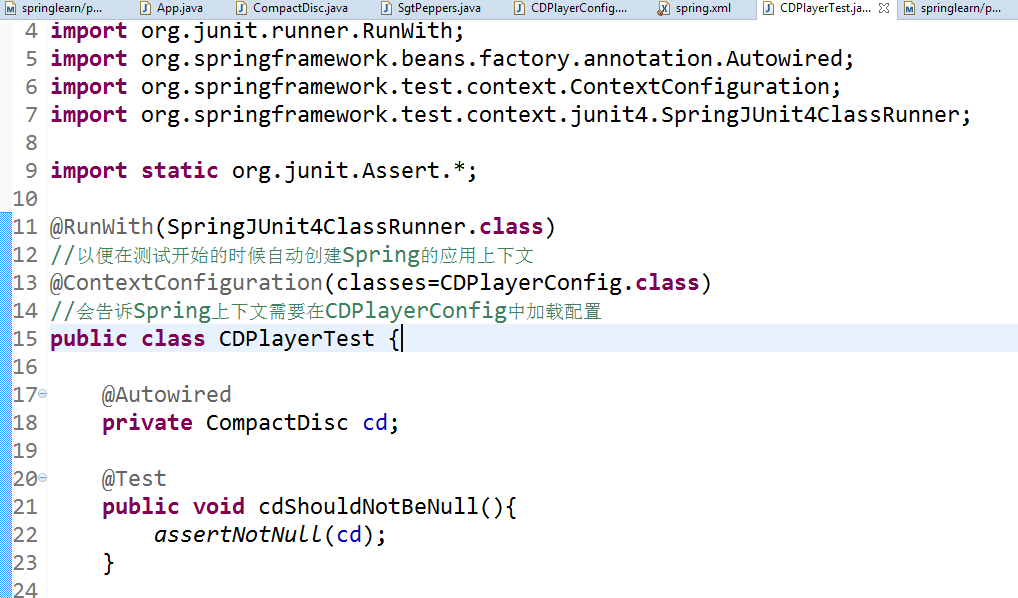
// 这个注解表明该类会作为组件类，并且告诉Spring要为这个类创建bean

没有必要显示配置SgtPeppersBean， 因为这个类使用了@Component 注解，Spring会把事情处理妥当

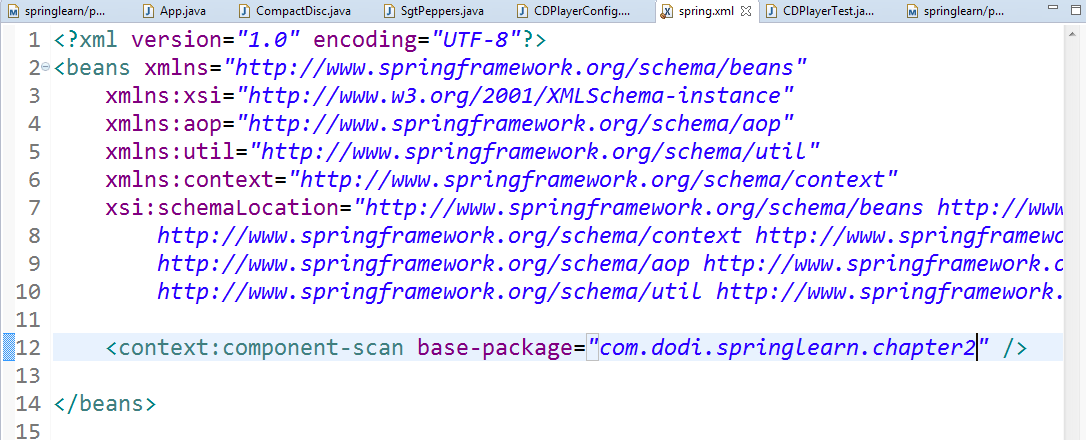
不过，组件扫描默认是不启用的。我们还需要显式配置一下Spring，从而命令它去寻找带有@Component注解的类，并为其创建bean



如果没有其他配置的话，@ComponentScan默认会扫描与配置类相同的包。因为CDPlayerConfig类位于com.dodi.springlearn.chapter2包中，因此Spring将会扫描这个包以及这个包下的所有子包，查找带有@Component注解的类。这样的话，就能发现CompactDisc，并且会在Spring中自动为其创建一个bean。



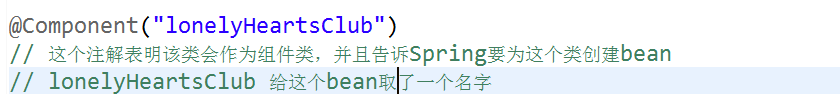
我们也可以使用XML来启用组件扫描



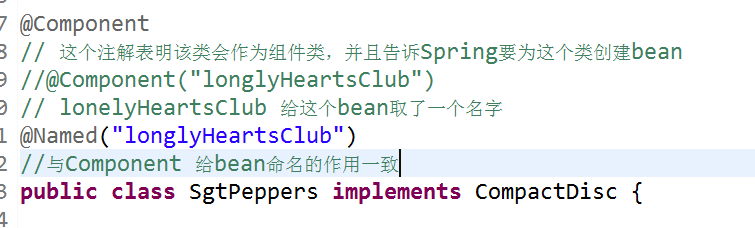
为组件扫描的bean命名

Spring应用上下文中所有的bean都会给定一个ID。在前面的例子中，尽管我们没有明确地为SgtPeppers bean设置ID，但Spring会根据类名为其指定一个ID。具体来讲，这个bean所给定的ID为sgtPeppers，也就是将类名的第一个字母变为小写。

如果想为这个bean设置不同的ID，你所要做的就是将期望的ID作为值传递给@Component注解。比如说，如果想将这个bean标识为lonelyHeartsClub，那么你需要将SgtPeppers类的@Component注解配置为如下所示：



还有另外一种为bean命名的方式，这种方式不使用@Component注解，而是使用Java依赖注入规范（Java Dependency Injection）中所提供的@Named注解来为bean设置ID：



**设置扫描组件的基础包**

为了指定不同的基础包，你所需要做的就是在@ComponentScan的value属性中指明包的名称



在上面的例子中，所设置的基础包是以String类型表示的。我认为这是可以的，但这种方

法是类型不安全（not type-safe）的。如果你重构代码的话，那么所指定的基础包可能就会出现错误了。

除了将包设置为简单的String类型之外，@ComponentScan还提供了另外一种方法，那就是将其指定为包中所包含的类或接口：



可以看到，basePackages属性被替换成了basePackageClasses。同时，我们不是再使用String类型的名称来指定包，改为basePackageClasses属性所设置的数组中包含了类。这些类所在的包将会作为组件扫描的基础包。

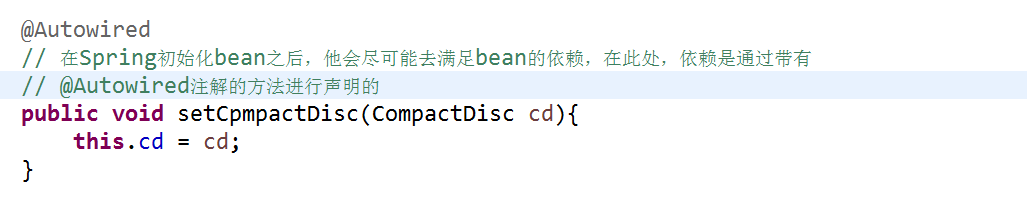
**通过为bean添加注解实现自动装配**

简单来说，自动装配就是让Spring自动满足bean依赖的一种方法，在满足依赖的过中，会在Spring应用上下文中寻找匹配某个bean需求的其他bean。为了声明要进行自动装配，我们可以借助Spring的@Autowired注解。



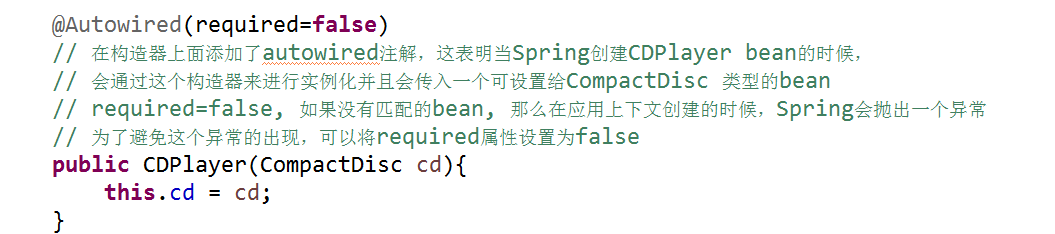
@Autowired注解不仅能够用在构造器上，还能用在属性的Setter方法上。比如说，如果

CDPlayer有一个setCompactDisc()方法，那么可以采用如下的注解形式进行自动装配：



不管是构造器、Setter方法还是其他的方法，Spring都会尝试满足方法参数上所声明的依赖。假如有且只有一个bean匹配依赖需求的话，那么这个bean将会被装配进来。

如果没有匹配的bean，那么在应用上下文创建的时候，Spring会抛出一个异常。为了避免异常的出现，你可以将@Autowired的required属性设置为false：



将required属性设置为false时，Spring会尝试执行自动装配，但是如果没有匹配的bean

的话，Spring将会让这个bean处于未装配的状态。但是，把required属性设置为false时，你需要谨慎对待。如果在你的代码中没有进行null检查的话，这个处于未装配状态的属性有可能会出现NullPointerException。

如果有多个bean都能满足依赖关系的话，Spring将会抛出一个异常，表明没有明确指定要选择哪个bean进行自动装配