# Spring之旅

本章从宏观层面上介绍Spring。

## 1.1 简化java开发

Spring最根本的使命：简化java开发，策略：基于POJO的轻量级和最小侵入式编程、通过依赖注入与AOP实现松耦合、基于切面与惯例进行声明式编程、透过切面和模板减少样板式代码。

### 1.1.1 激发POJO的潜能

很多框架都会要求应用继承基类或者实现接口，这种侵入式编程导致应用与框架绑死，Spring中的POJO完全看不出来使用了Spring，实际上没有Spring，POJO仍然是一个合法的类。

### 1.1.2 依赖注入

# 装配Bean

传统方法创建的对象之间的关系高度耦合，难以复用与测试，创建应用对象之间的关系叫做装配。

## 2.1 Spring配置的可选方案

3种：XML配置、注解配置、自动装配。选择那种根据使用场景，也可以同时使用多种，最好用自动装配，配置越少越好。必须要明显配置的时候用注解配置，比如源代码不是自己维护的时候，如果注解不能满足要求，则用XML，比如便利的XML命名空间。

## 2.2 自动化装配Bean

Spring实现自动装配的步骤：1.component scanning组件扫描Bean，知道有哪些Bean对象；2.autowiring自动装配，满足Bean之间的依赖关系。

### 2.2.1 创建可被发现的Bean

使用@Component注册组件类，告知Spring要创建这个Bean，可被扫描发现；组件扫面默认是不启用的，需要配置启用才能扫描到@Component，在一个普通类上加入@Configuration与@ComponentScan就代表开启了组件扫描，默认情况下，会扫描这个配置类所在包以及子包的@Component组件，在XML文件中是通过<context:component-scan base-package=”包名” />实现同样的功能的。

### 2.2.2 为组件扫描的Bean命名

Spring应用上下文中的Bean都有名字，不指定默认名字是类名第一个字母小写后的名字，可以设置ID，比如@Component(“name”)，也可以使用java依赖注入的规范，@Named(“name”),2个注解功能一样，大多数场景都可以相互替换。

### 2.2.3 设置组件扫描的基础包

@ComponentScan默认扫描本包极其子包，如果要扫描其他包可以指定；指定方法有：

1. @ComponentScan(“package”)，value指定；
2. @ComponentScan(basePackages=”package”)或者@ComponentScan(basePa ckages={“package1”,”package2”})包名指定，这种方法不安全，加入包名修改了怎么办；
3. @ComponentScan(basePackage Classes={Class1.class,Class2.class}),所在类指定，为了拓展性，这个类一般指定为一个空标记接口，避免了引用实际代码。

### 2.2.4 通过为Bean添加注解实现自动装配

自动装配就是要让Spring自动满足bean依赖，使用@Autowired注解，@Autowired可以用在类的任何方法与属性上，在方法上时，寻找匹配参数的依赖Bean，如果有多个报歧义异常，如果没有也会报异常，如果有@Autowired(required=false)，则不会报异常，此时处于未装配状态，可以使用@Inject注解代替它。

### 2.2.5 验证自动装配

验证通过。

## 2.3 通过Java代码装配Bean

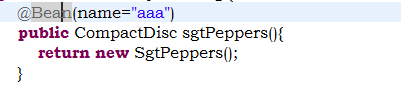
自动化组件扫描有时不行，比如需要装配第三方库类时，没法写上@Component等注解，所以需要在配置类中显示配置Bean，配置类一般只用于配置Bean，所以不要有业务逻辑代码，通常是放入单独的包中。

### 2.3.1 创建配置类

在配置类上加入@Configuration注解。

### 2.3.2 声明简单的Bean

声明一个Bean需要在配置类中声明一个方法，类似下面这个：



@Bean会告诉Spring这个方法返回的对象注册成为一个Bean，方法体中是Bean的产生逻辑，name指定对象的名字，没有name指定的话名字是方法名。

### 2.3.3 借助JavaConfig实现注入

准配Bean最简单的方法是引用创建Bean的方法，如下：

@Bean

Public CDPlayer cdplayer(){

Return new CDPlayer(sgtPeppers());

}

才是创建CDPlayerBean是使用的带参数的构造器创建的，传入的参数是sgtPeppers()的返回对象，由于是Bean，每次调用Spring会拦截，返回相同的Bean对象，不是每次调用都创建新对象；

另一种写法：

@Bean

Public CDPlayer cdplayer(CompactDisc compactDisc){

Return new CDPlayer(compactDisc);

}

当要创建这个bean时，Spring自动为参数装配一个CompactDisc 类型的对象；不用使用方法，也不用将2个方法声明在同一个配置类中，CompactDisc 可以是其他任何方式生成的Bean；使用setter方法注入的写法也是类似的：

@Bean

Public CDPlayer cdplayer(CompactDisc compactDisc){

CDPlayer cdplayer= new CDPlayer();

cdplayer.setCompactDisc(compactDisc);

Return cdplayer;

}

带有Bean注解的方法可以采用任何的java功能开生成Bean实例。

## 2.4 通过XML装配bean

XML是Spring的最古老的配置方式，建议使用自动化配置与注解配置。

### 2.4.1 创建XML配置规范

Spring XML文件在顶部声明元素的模式文件，根元素是<beans>。

### 2.4.2 声明一个简单的<bean>

在XML文件中声明一个Bean用<bean>元素，比如<bean class=”class1”/>,

class指定类的全限定名，调用无参构造函数构造，没有指定id的话名是class1#0这种形式，后面的是序号，缺点：创建Bean的方式太单一不如注解多样，没有编写期类型检查，注解可以。

### 2.4.3 借助构造器注入初始化bean

构造器注入分为使用<constructor-arg>元素与c命名空间2种，构造器注入要求类有带参数的构造函数，写法如下：

<bean class=”class1”>

<constructor-arg ref=”class2”(value=””这种注入基本值)>

</bean>

c命名空间的写法是<bean class=”class1” c:参数名-ref=”class2”/>,参数名写法可以用位置写法代替比如 c:\_0-ref这种形式，如果只有一个参数，可以直接使用c:\_-ref这种形式，如果是基本值的注入的话去掉-ref这个后缀就可以了，用法基本相同的。<null/>给参数装配null值，装配集合就是使用集合对应的元素比如<list>,<set>等，c命名空间无法装配集合。

使用setter方法注入依赖关系，使用<property>元素代替<constructor-arg>元素，其他基本不变。Setter注入的命名空间方式是p。p也不能装配集合，可以使用util空间的list制造集合，进行注入。

## 2.5 导入和混合配置

Spring的各种配置方案不互斥，所有方案创建的Bean都是共用的，平等的。

### 2.5.1 在注解配置中引用XML配置

假设一个Spring的配置类中的bean太多，拆分了多个配置类，可以通过在一个配置类上加入@Import({class1,class2})把配置类加载进来一起生成Bean，如果想把XML中的配置加载进来，使用@ImportResource(“XML路径”)把XML配置文件加载进来，获取Bean的配置情况，建议建立一个根配置的配置类，专门加载子配置。

### 2.5.2 在XML配置中引用注解配置

XML配置文件之间可以用<import>元素加载配置，resource指定文件路径，如果要加载注解配置类，可以在在XML中创建一个Bean，如：

<bean class=”主机配置类”></bean>，

建议把import与注解配置bean放到一个跟配置XML文件中，在这个XML文件中可以启用自动扫描等。

## 2.6 小结

# 高级装配

很重要，价值很高。

## 3.1 环境与profile

应用程序进行环境迁移时，有些模块会发生变化，不能写死程序代码，要留有可变化的模块配置。

### 3.1.1 配置profile bean

Spring会在运行依据环境决定Bean的创建。这种状态的判断是使用bean profile实现的，将bean定义整理到一个或者多个的profile中，对应激活的profile的bean将会被创建；使用@Profile注解指定bean属于哪个profile，这个注解可以用在注解配置类上，那么这个注解配置类配置的bean都将属于这个profile下，也可以在创建Bean的方法级别上使用这个注解，没有指定profile的bean，始终都会被创建；也可以在XML配置方法中配置bean的profile属性，可以在<beans>元素中指定属性profile=”dev”，那么这个配置文件的bean就归属该profile，所有的配置文件都存在与包中，但是只有profile处于激活状态的配置文件或者没归属的配置文件将会被Spring使用；<beans>元素下面可以嵌套<beans>元素，为该子beans元素指定profile就可以把多个XML的文件配置到一个XML文件中。

### 3.1.2 激活profile

确定目前激活的profile，需要用到spring的2个独立的属性spring.profiles.active与spring.profiles.default；优先读取spring.profiles.active，没有再读取spring.profiles.default，再没有，则只创建那些没有归属profile的bean。

设置这2个属性的方式：作为dispatcherServlet的初始化参数、作为Web应用的上下文参数、作为JNDI条目、作为环境变量、作为JVM的系统属性、在集成测试类上，使用@ActiveProfiles注解设置。

## 3.2 条件化的bean

假设某个Bean的创建条件化依赖与其他Bean的创建，或者某个环境变量存在才创建，此时可以使用@Conditional注解，可以用到带有@Bean注解的方法上，如果@Conditional指定的类的特定方法返回true，则创建Bean，false不创建，指定的类要实现Condition接口，里面有个返回boolen类性的matches方法，需要在里面实现我们自己的逻辑boolean matches(ConditionContext ctct,AnnotatedTypeMetadata metadata)，ConditionContext接口的方法有：

beanDefinitionRegistry getRegistry()检查Bean定义,ConfigurableListableBea nFactory getBeanFactory(),检查Bean存在即bean属性，Environment getEnvironment() 检查环境变量的存在即值，ClassLoader getClassLoader(),加载及检查类是否存在，AnnotatedTypeMetadata可以检查除了@Bean注解外的其他注解，这个接口的方法有：

Boolean isAnnotated(String annotatedType)检查是不是还有其他特定的注解，

Map<String,Object> getAnnotationAttributes(...)等。

@Profile注解就是通过@Conditional注解实现的。

## 3.3 处理自动装配的歧义性

当容器内有多个Bean的类型满足注入的类型，则会发生注入异常。

### 3.3.1 标识首选的Bean

在使用@Component与@Bean注解时使用@Primary注解标识首选Bean，那么这个bean会被优先注入，在XML配置中，使用primary=”true”，假设有多个首选Bean依然无法正长工作。

### 3.3.2 限定自动装配的Bean

Spring的限定符可以更好的限定Bean的选取，限定符就是@Qualifier注解，可以与@Autowaired与@Inject一同使用，比如@Qualifier(“Bean的ID”)，实际上Spring为每个bean都指定了限定符名字，没有特殊指定的话，这个名字与Bean的ID相同，所以@Qualifier里面的值就是Bean的ID，这种紧耦合会造成重构问题，所以可以指定Bean的限定符名字，也是@Qualifier(“限定符名字”)这种形式的，如果指定的限定符名字相同，可以创建自定义的限定符注解，这个注解要使用@Qualifier注解来标识，则它们具有了@Qualifier注解的特性，名字我们可以随便取。

## 3.4 bean的作用域

对象会被状态污染，所以Spring定义了多种作用域：单例，整个应用只创建一个实例；原型：每次注入或者通过Spring上下文获取的时候都会创建一个新的实例；会话，为每个会话创建一个bean实例；请求，为每个请求创建一个bean实例；使用@Scope指定作用域，可以跟@Componet与@Bean注解一起使用，XML方式下使用scope属性指定。

### 3.4.1 使用会话和请求作用域

购物车bean不能使用单例与多例bean，使用会话作用范围的bean最合适。

比如形式如下：

@Component

@Scope(value=WebApplicationContext.SCOPE\_SESSION,proxyMode=ScopedProxyMode.INTERFACES)

Public class ShoppingCart{

}

这个注解会给每个会话创建一个购物车，面临的问题：加入这个bean要注入到单例的storedService bean中，如下：

@Component

Public class StoredService{

@Autowaired

Public void setShoppingCart(ShoppingCart shoopingCart){}

}

由于这个bean是在Spring上下文加载时初始化，此时购物车bean还不存在，希望的是将每次会话产生的bean注入到里面，Spring采用代理模式，Spring并不会将shoopingCart bean注入，而是注入一个这个bean的代理，代理有与这个bean相同的方法，当里面调用shoopingCart的方法时，代理会进行懒解析并将调用委托给会话作用域真正的shoopingCart，proxyMode=ScopedProxyMode.INTERF ACES，就是为了解决这种问题的，这个注解（接口代理）表明代理要方法参数中ShoppingCart是一个接口，代理实现这个接口，并将调用委托给真正的bean，如果ShoppingCart不是接口而是类，Spring没法创建基于接口的代理，要使用CGLib来生成基于类的代理，proxyMode=ScopedProxyMode.TARGET\_CLASS，request作用域同理。

### 3.4.2 在XML中声明作用域代理

XML要实现跟作用域注解代理这样的功能是使用<aop>命名空间里面的元素<aop:scoped-proxy/>指定的是基于类的代理，加上proxy-target-class=”false”属性后指定基于接口的代理，必须现在上面的命名空间里声明aop空间。这些内容也是面向切面编程的一部分。

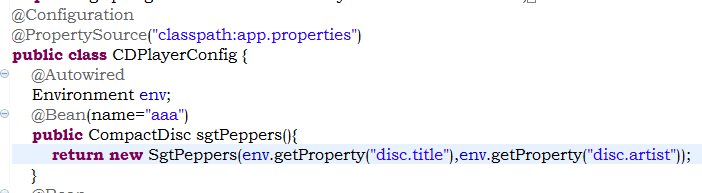
## 3.5 运行时注入

配置Spring的注入时，注入基本值，基本都是将基本值写入到代码里注入，这种一旦写入，很难更改，XML配置也是这样的，如果想要应用在运行时才确定值，可以采用属性占位符与SpEL2种方式。

### 3.5.1 注入外部的值

处理外部值的最简单方式声明属性源并通过Spring的Environment来检索属性。

比如如下：



使用外部属性来专配bean的初始值。Environment相当于一个Properties对象，有4个主要的方法：String getProperty(String key),String getProperty(String key,String defautValue)，T getProperty(String key,Class<T> type)，T getProperty (String key,Class<T> type，T defaultValue)，顾名思义。getRequiredProperty()，属性必须有，没有抛异常，前面的只是返回null，containsProperty，检查属性是否存在，getPropertyAsClass()将属性解析为类，还提供了一些检查profile信息的方法。Spring使用占位符的形式将外部属性值注入到bean中，占位符形式为${属性key}，XML与注解配置的都是直接使用${}注入，当使用组件扫描的方法时，由于没有专门的配置文件，要死使用@Value（“${}”）注解为相关属性注入。为了使用占位符，要配置一个PropertyPlaceholderConfigurer bean或者PropertySourcesPlaceholderConfigurer bean，XML中使用元素<context:property-placeholder>配置这种bean。

### 3.5.2 使用Spring表达式语言进行装配

SpEL特性：使用bean的ID来引用bean，调用方法和访问对象的属性，对值进行算术、关系和逻辑计算，正则表达式匹配、集合操作。

SpEL表达式要放在#{}中，场景：

1. 表示字面值：#{1}标识数值类型、String、boolean都可以；
2. 引用bean及其内部属性：#{bean}、#{bean.attribute}、#{bean.method}、#{bean.method()?.toUpperCase},返回值在调用方法等，也可以引用系统属性；
3. 在表达式中使用类型：#{T(System).currentTimeMillis()}标识不是字符串，要解析为类；

SpEL支持非常广泛的运算符与集合运算操作，这里可以看书。

# 面向切面编程

散布于应用程序中多处的功能是横切关注点，是与业务逻辑无关的，分离他们是AOP要解决的问题，DI完成依赖对象之间的解耦，AOP完成横切关注点与业务逻辑的解耦，横纵向的解耦，

## 4.1 什么是面向切面编程

切面就是提炼出来的横切关注点，重复的功能多处用到的话一种是继承，一种是委托，继承容易造成对象关系混乱，委托需要调用，面向切面编程提供了解决方案，在切面定义通用功能，在更切关注点声明使用，完成了通用功能与业务逻辑的解耦。

### 4.1.1 定义AOP术语

行话：切面（通知，Advice）：定义了通用功能完成的工作以及何时使用，根据使用的时机分为5种：Before，前置通知，在目标方法调用前调用切面；After后置通知，在目标方法调用后调用切面，不关心方法的输出；After-returning返回通知，成功执行后调用切面；异常通知，抛出异常后调用切面；环绕通知，包裹了目标方法，可以在之前或者之后自定义行为。

连接点（join point）：应用执行过程中能够插入切面的点，切面代码可以利用这些点插入到应用的正常流程中，并添加新的行为。

切点（Pointcut）：切点定义了在哪个连接点织入切面，实际就是连接点的筛选条件，定义了何处的问题。

切面（Aspect）：通知与切点的结合，定义了做什么、何时做、在哪里做。

引入（Introduction）：向现有的类添加方法或者属性。

织入（Weaving）：把切面应用到目标对象并创建新的代理对象的过程，在目标对象的生命周期中的织入时机：编译器，在目标类编译时织入，需要特殊的编译器；类加载期，在目标类加载到JVM时织入，需要特殊的类加载器；运行期，在运行的某个时机被织入此时AOP会为目标对象动态的创建一个代理对象。

### 4.1.2 Spring对AOP支持

Spring提供了4种AOP支持，基于代理的经典Spring AOP、纯POJO切面、@AspectJ注解驱动的切面、注入式AspectJ切面；Spring AOP构建在动态代理之上，所以只支持方法拦截；纯POJO是使用的XML方式配置成切面处理；注解的与AspectJ注解通用。有3点需要重点注意：

1. 切面&通知是用java编写的；2.Spring在运行时织入通知，代理类封装了目标类，并拦截对目标类的方法访问；3.Spring只支持方法级别的连接点。

## 4.2 通过切点来选择连接点

切点定位应用通知的地点，在Spring中使用Aspect切点表达式语言书写切点，由于是代理方式，所以与代理无关的AspectJ切点表达式语言是不能使用的，具体能使用的可以看书，其中execution是最重要的表达式，其他的都是辅助。

### 4.2.1 编写切点

切点一般是描述目标对象方法的表达式，比如下面：

Execution(\* concert.Performance.perform( .. )&&within(concert.\*)

当需要执行目标对象的concert.Performance.perform方法时，将拦截执行切面，第一个\* 标识匹配任何返回类型，（..）标识匹配任意类型，within表示只匹配该包下的内容，&&标识与同样的还有||与！；在XML中&有特殊含义，使用and、or、not代替。

### 4.2.2 在切点中选择bean

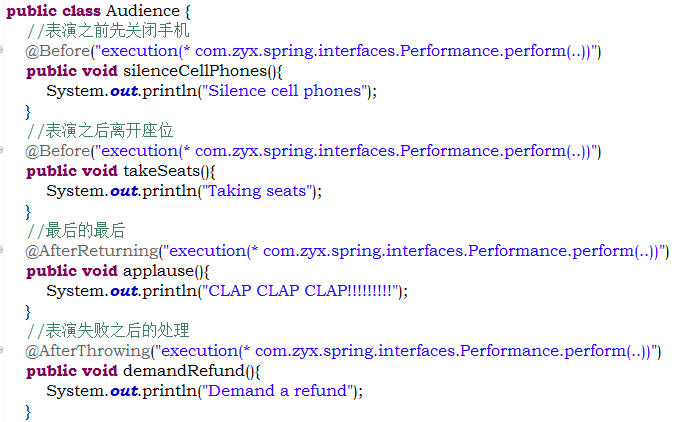
bean()表达式限定只能匹配特定的bean，里面是bean的ID或者名称。使用！后就是相反的情况。

## 4.3 使用注解创建切面

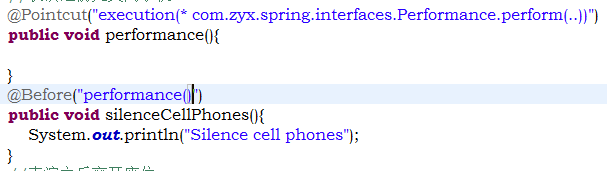
Aspect5之后可以使用注解将普通类转换为切面，之前有特殊的写法。

### 4.3.1 定义切面

以表演为例，观众的操作定义为切面类。如下形式：



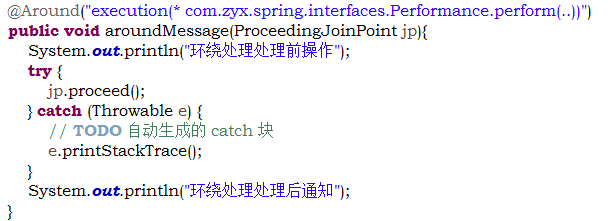
切面中方法的应用时机有5种，顾名思义：@After、@AfterReturning、@AfterThrowing、@Around、@Before；切点表达式都是相同的，可以提炼出来，使用@Pointcut表示；如下：



此时这个类只是一个普通的java类，需要将它转换为切面类，在注解配置中，在配置类上加入@EnableAspectJAutoProxy注解启用自动代理功能，在XML文件中，使用<aop:aspectj-autoproxy>启用自动代理，会为@AspectJ表示的bean中匹配的切点bean创建代理。

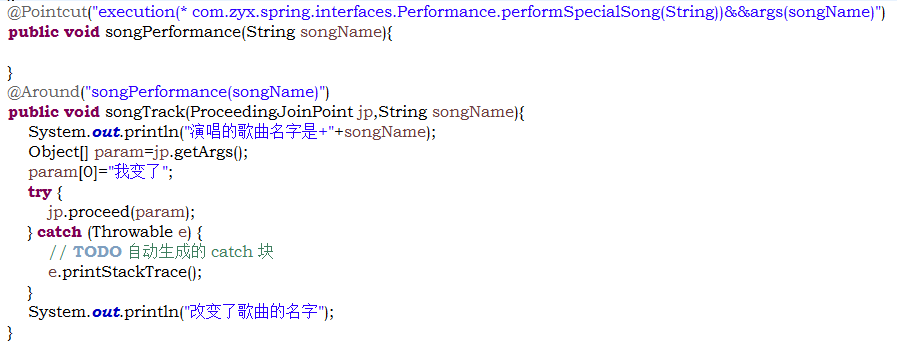
### 4.3.2 环绕通知

环绕通知使用@Around标识，形式如下：



注意 ProceedingJoinPoint参数，这个代表了目标Bean的上下文，通过它来调用目标对象的目标方法，可以不调用或者多次调用。

### 4.3.3 处理通知中的参数



传递参数需要在用到args切点表达式来传参数，args表达式标识传递给目标方法的参数也要传递到切面方法中参数的名称要匹配。

### 4.3.4 通过注解引入新功能

切面在生成目标对象的代理对象时，代理对象内也可以加入其他的接口或者类实现，在Spring中，代理是接口代理，所以代理对象代理了其他的接口对象的话，就为代理对象加入了新的功能。



当引入的接口方法被调用时，被委托给了某个实现了这个接口的对象调用；使用@DeclareParents(value=”concert.Performance+”,defaultImpl=DefaultEncoreab le.class)

Public static Encoreable encoreable；

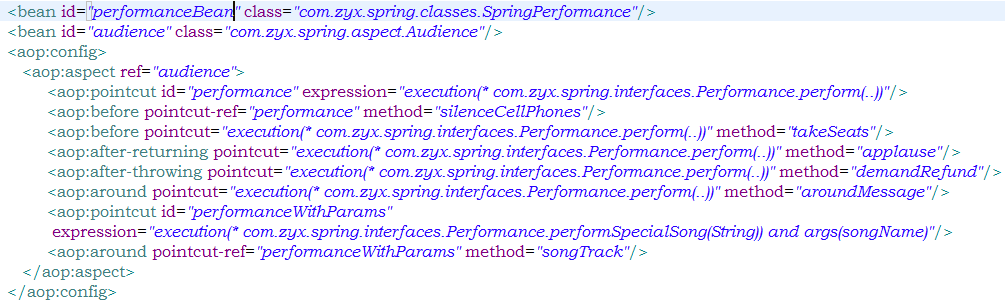
Value指定了那种类型的bean要引入该接口，+号代表子类型；DefaultEncoreable.class是实现了Encoreable 接口的一个类，后面定义的属性指定了要引入的接口。

注解的切面声明必须要有源码，如果没有源码的话使用XML声明。

## 4.4 在XML中声明切面

Spring的aop命名空间定义了很多有关AOP的元素这里可以看书。元素名与功能与注解的都差不多。

### 4.4.1 声明前置与后置



AOP元素在<aop:config>内使用，在这个元素内，可以声明很多的切面、切点等；使用<aop:aspect>声明切面，ref指向切面类bean ID，切点的定义如上图。Pointcut重复后也可以使用<aop:pointcut>单独提炼出来。

### 4.4.2 声明环绕通知

定义的方法与注解相同，XML配制方法也与其他一样。

### 4.4.3 为通知传递参数

XML配置方法一样，切点表达式加上参数就OK了。注意&&要写为and。

### 4.4.4 通过切面引入新的功能

<aop:declare-parents

Types-matching=”concert.Performance+”

Implement-interface=”concert.Encoreable”

Default-impl=”concert.DefaultEncoreable”/>  
 声明引入新的接口，Types-matching是代理的目标对象的接口，Implement-interface是引入的接口，Default-impl是引入接口的实现类，也可以使用delegate-ref=”bean ID”。

## 4.5 注入AspectJ切面

就是为Aspect bean注入其中要用到的对象。

# 构建Spring Web应用程序

Spring MVC是MVC框架。

## 5.1 Spring MVC起步

先看一个完整的流程。

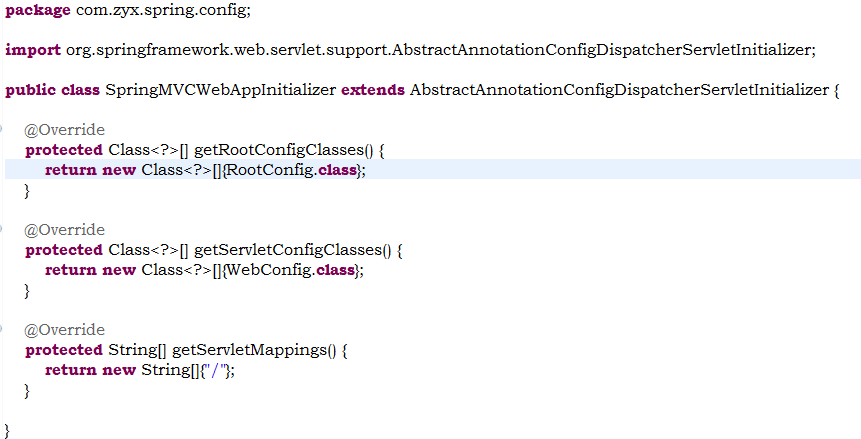
### 5.1.1 跟踪Spring MVC的请求



1. 将用户请求信息URL提交过来；2.到达前端控制器DispatcherServlet，是单例的，委托给其他组件处理，DispatcherServlet将请求发送给SpringMVC控制器，需要知道发送给哪个控制器进行处理，所以要查找映射确定控制器；3.知道控制器后将信息发送给控制器处理，并等待结果；4.处理后产生一些数据信息，需要返回给客户端，称为模型(Model)，还要返回展示的方式，也就是逻辑视图名，模型数据与视图名打包返回；5.逻辑视图名不是真正的物理视图，需要查找，所以需要视图解析器，找到真正的物理视图，并执行视图代码，得到渲染后的HTML输出；6.返回视图输出。

### 5.1.2 搭建Spring MVC

DispatchServlet是Spring MVC的核心，可以配置在Web.xml中，也可以在类中配置。需要支持Servlet3并且Spring的版本要在3.1以上。配置DispatchServlet的java代码如下：



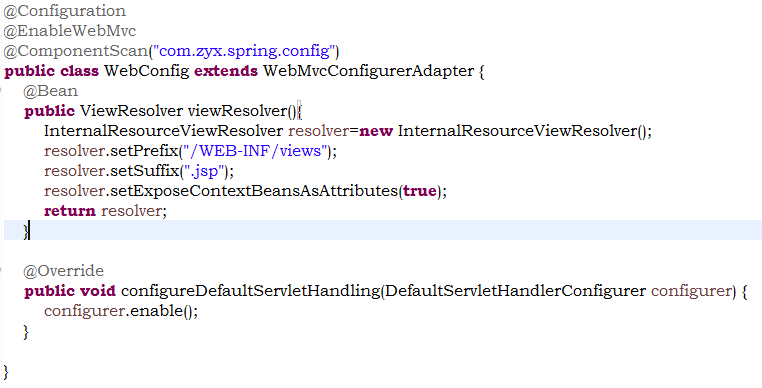
只需要知道继承AbstractAnnotationConfigDispatcherServletInitializer的任意类都会自动的配置DispatcherServlet和Spring的应用上下文。Spring的上下文会位于Servlet的上下文中。

原理：在支持Servlet3.0的Web容器中，容器会在类路径中需找实现了接口ServletContainerInitializer的类，Spring提供了这个接口的实现SpringServletContai nerInitializer，这个类中会查找实现了WebApplicationInitializer接口的类，并把配置任务交给它来执行，Spring3.2提供了这个接口的基本实现类AbstractAnnotatio nConfigDispatcherServletInitializer（中间还有一连串的继承等），继承这个类，就能在启动时被Web容器发现并执行了。里面由3个方法：

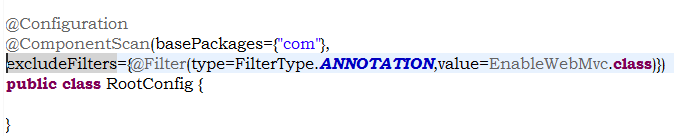
protected String[] getServletMappings()，将路径映射到该DispatchServlet上，就是规定哪种路径要经过它路由，”/”规定所有路径都可以经过它。

AbstractAnnotationConfigDispatcherServletInitializer获得执行是，会同时创建DispatcherServlet与ContextLoaderListener2个对象，2个对象作为底层对象都要负责创建应用的Bean，从良好设计的角度来说DispatcherServlet负责创建Web相关的Bean，比如控制器、视图解析器即映射等，ContextLoaderListener负责业务逻辑相关的bean的创建，根据上图，DispatcherServlet会根据getServletCon figClasses() 方法返回的@Configuration的配置类创建Bean，ContextLoaderLis tener会根据getRootConfigClasses()返回的@Configuration的配置类创建Bean。如果web容器不支持Servlet3.0那就只能使用Web.xml方式了。

开启Spring MVC的功能需要有一个类带有@EnableWebMvc注解（这个注解会帮助配置扫描到@Contoller配置的类）。我们配置在WebConfig类中；如果没有配视图解析器的话，Spring默认使用BeanName ViewResolver，根据视图名查找Bean ID（实现View接口）的解析器；没有启用组件扫描的话，只能找到配置类中显式声明的控制器，InternalResourceVi ewResolver定义了个视图解析器，configureDefaultServletHandling方法将静态资源的请求转发到其他的Servlet上，不让DispatchServlet处理。

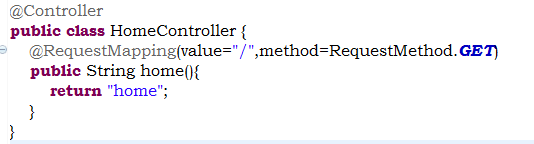


RootConfig类的代码如下：



## 5.2 编写基本的控制器

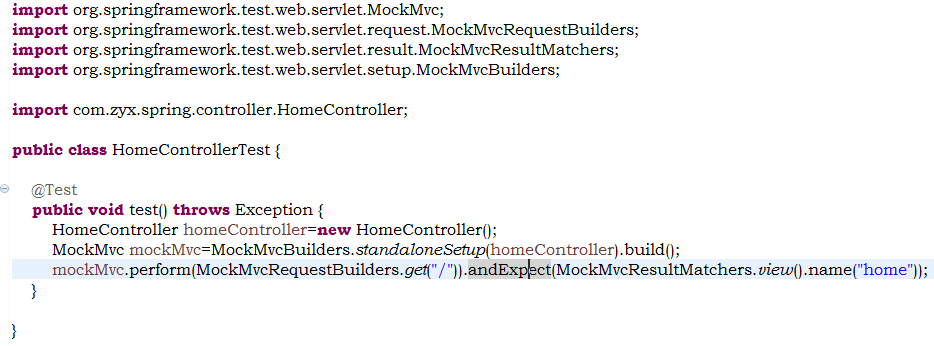
控制器是添加了@Controller注解的类，同时方法上添加了@RequestMapping注解。



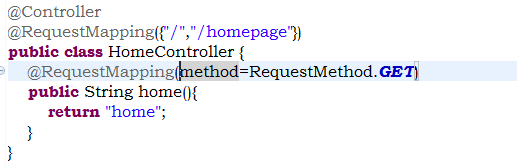
@Controller注解影响不大，是@Component注解的拓展，实现组件扫描，完全可以替换，只是为了表达更清楚，采用的@Controller，@RequestMapping指定了请求路径与请求方法。

### 5.2.1 测试控制器

传统的junit测试能创建controller对象，但是并不能模拟HTTP访问，所以Spring提供了mock Spring MVC功能来模拟HTTP访问测试。

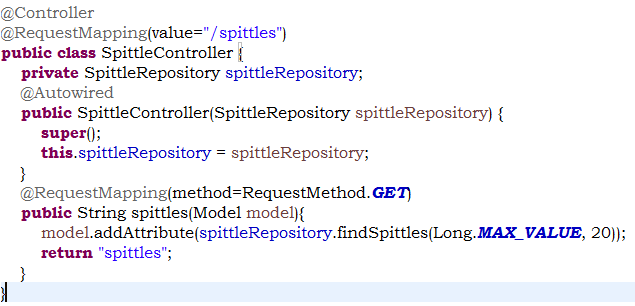


### 5.2.2 定义类级别的请求处理



@RequestMapping注解可以放在类上，对所有的方法都起作用，与方法上的@RequestMapping注解一起拼合成路径起作用，而且value路径可以接收一个String数组。

### 5.2.3 传递模型数据到视图中



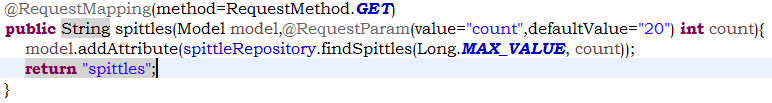
与前面的基本是一致的，此处注入了一个Service层的组件，用于产生数据，Model就是一个Map类型拓展，此处也可以用Map，这个Map将会传递到request对象中，addAttribute()如果没有指定key，根据对象类型生成key，不如List<Spittle>会生成spittleList这种key。也可以不返回视图名，直接返回Model数据，此时逻辑视图名会根据请求路径推断，最终在JSP试图中可以使用JSTL访问request里面的这个对象。此处Mock有特殊的用法，需要看书。

## 5.3 接受请求的输入

Spring MVC允许有多种方式将客户端中年的数据传送到控制器的处理器方法中，包括查询参数、表单参数、路径变量。

### 5.3.1 处理查询参数

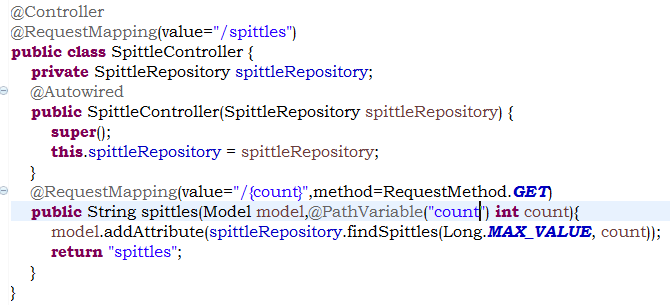
针对请求路径上的get参数？Key=value等；使用注解@RequestParam获取。



@RequestParam获取请求的参数值，如果没有这参数，那么使用默认值defaultValue，类型转换在赋值时自动完成。

### 5.3.2 通过路径参数接受输入

输入的信息作为URL的一部分，但不是？后面的参数形式，这种是一种资源的方式，@RequestMapping的占位符解决了这个问题。

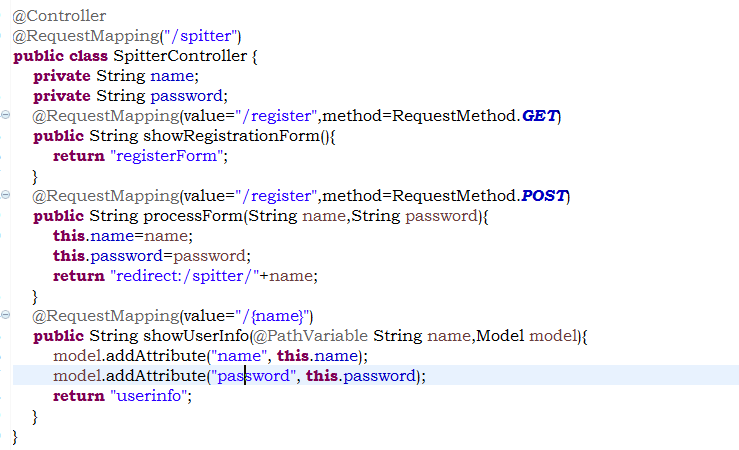


访问的URL路径上的count处的内容由count标识；使用｛｝表示占位符，在函数的变量中使用@PathVariable应用占位符处的值，并赋给参数。参数count与占位符相同，可以去掉@PathVariable中的value。

## 5.4 处理表单

处理表单需要有controller与含有form的视图。如果form指定action那么提交时提交到当前路径处理。

### 5.4.1 编写处理表单的控制器



可以看到form的参数会根据名字注入到方法中的对应的参数中，返回的字符串含有redirect，此时解析为重定向，而不是逻辑视图名，会重定向到对应页面，可以看到重定向后显示的页面为userinfo，同时数据也渲染进去了，注意资源路径的name的中文问题。

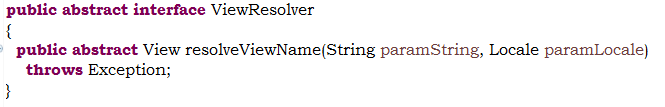
### 5.4.2 校验表单

from表单提交的数据应该被校验防止不合理数据，可以在前端做，也可以在控制器里做，自己实现校验费时费力，Spring已经提供了对java对象的校验机制，校验机制使用定义在属性上的注解规定属性满足的规则注解可以，包含的注解有@Null、@NotNull、@Size等。在传入的参数变量上设置@Valid注解，后面跟随一个Errors变量，当校验出现问题后，后面Errors变量会存储这个错误。

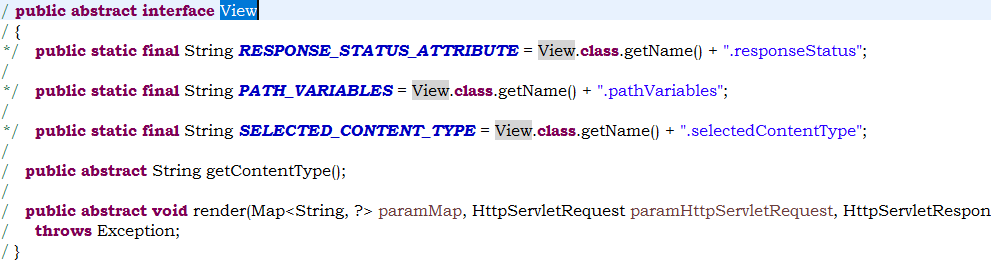
# 渲染Web视图

## 6.1 理解视图解析器

Spring MVC所有的视图解析器都要实现一个接口：



ParamString为传入的逻辑试图名称，paramLocale是Locale对象。返回的View接口对象如下：



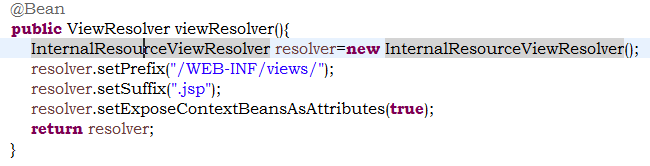
接受模型数据、request、response对象。将模型model数据放入response中到前端界面渲染成HTML。Spring MVC提供了很多实现好的视图解析器。

|  |  |
| --- | --- |
| 视图解析器 | 描述 |
| BeanNameViewResolver | 将逻辑视图与bean ID匹配解析为bean |
| ContentNegotiatingViewResolver | 根据客户端需要的类型解析逻辑视图，委托给另外的能够产生这种类型的视图解析器解析 |
| FreeMarkViewResolver | 解析为FreeMark模板 |
| InternalResourceViewResolver | 解析为Web应用的内部资源（JSP） |
| JasperReportsViewResolver | 解析为JasperReports定义 |
| ResourceBundleViewResolver | 解析为资源bundle |
| TilesViewResolver | 解析为Apache tile定义 |
| UrlBasedViewResolver | 逻辑视图名直接匹配物理视图名 |
| VelocityLayoutViewResolver | 解析为Velocity布局 |
| VelocityViewResolver | 解析为Velocity模板 |
| XmlViewResolver | 解析为xml中的bean定义 |

## 6.2 创建JSP视图

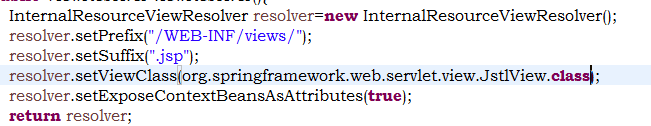
Spring为支持JSP视图需要：1.配置InternalResourceViewResolver；2.JSTL与SpEL。

### 6.2.1 配置适用与JSP的视图解析器



XML配置方式如下：

<bean id=”viewResolver” class=”org.springframework.web.servlet.view.InternalRe sourceViewResolver” p:prefix=”/WEB-INF/views/” p:suffix=”.jsp”>，逻辑视图名也可以添加斜线，可以实现目录分级。InternalResourceViewResolver会将逻辑视图名解析为InternalResourceView这个实例，这个实例引用了JSP文件，如果JSP使用了JSTL的话需要将视图解析为JstlView，需要设置viewClass属性。



<bean id=”viewResolver” class=”org.springframework.web.servlet.view.Internal ResourceViewResolver” p:prefix=”/WEB-INF/views/” p:suffix=”.jsp” p:viewClas s=”org.springframework.web.servlet.view.JstlView”>。

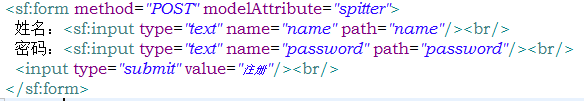
### 6.2.2 使用Spring的JSP库

标签库可以避免在JSP中直接使用java代码。非常有用。Spring提供了2类标签：1.表单处理相关标签；2.工具类标签。

一、表单处理标签：表单标签的value可以跟model里面的对象绑定。首先需要将标签声明进来。

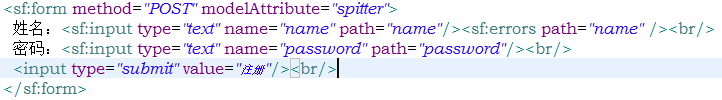
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="sf" %>

前缀可以是任意的形式，相关的一些标签可以看书。



modelAttribute为模型中的对象名字，path为对象中绑定值出现的位置。

表单中提交后进行校验后如果存在错误信息的话，还可以使用特殊的标签展现出来。



<sf:errors>将显示校验信息，path指定显示哪个属性的校验信息。比如：



会在指定位置替换为一个span标签，为了突出显示可以设置cssClass=”error”，生成的<span>有一个值为error的class样式定义，只要重新定义error的CSS就可以改变错误显示的样式，还有一种错误信息的布局方式是将所有的错误信息显示到一起，<sf:errors path=”\*” element=div .....> path里面用通配符代表所有的属性，并且设置了展现错误消息的标签为div，不是默认的span了，这个有一个cssClass为errors的样式，改变它的定义，也可以更改显示错误的信息的样式。<sf:label>标签也能够进行错误信息展示的处理。

同时也可以定义国际化的properties文件，在属性的注解校验上使用message属性指定国际化文件中的key，来完成不同地域的错误信息展示。

1. Spring通用的标签库：通用标签库很早就存在，包含一些标签，所有的标签需要看书，几个重要的标签：1.展现国际化信息的<s:message>标签；2.创建URL的标签<s:url>，3.转义标签<s:escapeBody>。

## 6.3 使用Apache Tiles视图定义布局

大型页面的布局繁琐复杂，纯代码书写不利于后续维护，是能够用布局引擎解决这个问题，Spring能够将逻辑视图名解析为Tile定义。

### 6.3.1 配置Tiles视图解析器

需要一个TilesConfigurer的Bean与一个TilesViewResolver的视图解析器，通过加载XML布局文件完成整个物理视图的架构设计，配置的方式也可以选择XML的配置。

## 6.4 使用Thymeleaf