spring字幕-邱明

代码地址：https://github.com/mingqcn/JavaEEPlatform.git

# 1.0 Spring框架核心--前言

the spring 框架的核心。

spring的核心，我们主要讲5个内容。

1.

那我们首先会探讨什么是**软件的模块化**。

软件的模块化是用来解决软件复杂性的一个最基本的方法。

啊，通过这个方法，我们可以把软件切成这个部分。

但不同的部分有不同的人，或者在不同的时间来实现。

2.

然后我们会来介绍spring5.0的两个技术栈。

这两个技术栈分别是我们最常用的传统的。

**阻塞式的servlet 技术栈**和更加高效的**非阻塞的reactive 技术栈**。

3.

第三，我们会介绍**spring的容器**的概念。spring容器是是spring 框架的最核心的内容。它负责创建和管理。

spring 中间的所有对象的。

呃，全生命周期。

4.

最后我们会来介绍,在spring容器中间一个非常重要的特性 **控制反转**，也就是由spring容器。

来管理对象之间的依赖和关联。

# 1.1 Spring框架核心--软件模块化

我们都知道，复杂性是软件的一个基本特征。简单的软件其实并不需要特别的软件工程的方法去解决它，但是那些复杂的工业及的软件。他们有着数以百计称之数以千万的数据，共成千上万的人依赖于这些软件来完成他的工作。这些软件的复杂性已经超过了单个开发者所能理解的范围通常来说，他需要几十甚至几百的程序员投入几个月，甚至一年的时间才可以完成这样的一个软件的开发。模块化的方法是用来控制软件复杂性的一个最基本的方法，通过模块化我们可以把软件划分成为多个部分，把每个部分分配给不同的人去做或者把不同的部分在不同的时间来做，从而使得简单的软件可以由多人协作在一段时间内来完成。

MVC:

软件模块化的方法有面向过程的方法和面向对象的方法，但是，无论哪种方法在模块化之前，他都会首先把软件划分成一个层次性的结构。

最简单的这个层次性的划分的方法是把软件切分成2层，1层是用来实现操作界面的**视图层**，它是整个程序的外壳，而另1层呢，则是包含程序的业务逻辑，程序的数据访问以及数据存储的**模型层**，它是整个程序的核心。在这个2层里头界面层相对比较单薄，而且简单通常我们会用面向过程的设计方法来直接设计界面层，而模型层呢？会比较复杂，而且内容比较多，同时他也容易发生改变，所以我们往往会有面向对象的方法来设计和实现模型层的部分。呃由于采用那两种不同的设计方法来设计了视图层和模型层，所以这2层之间，我们往往会插入了1层**控制器层**，用来把模型层的对象转换，成为适合被视图层的代码所使用的API。

这就是我们一个最基本的分层结构，把整个的系统分成了。视图层、控制器层和模型层，形成一个3层的结构。当然，由于模型层相对比较复杂，我们往往也可以把模型层分成更新小的层比如说，我们可以把模型层，把它分成用于负责业务逻辑的**服务层**，用来负责存储数据，组装成对象模型的**数据访问层**，以及负责对象模型和关系数据库转换的**映射层**，还有我们的数据载体，**实体层**。

分层的这样的一种方式，使得每1层都有特定的职责，采用层次以后，我们的任何一个功能被划分到多个层里头。这样我们就得到了一个比原有的模块划分，更细的一个结果，使得整体的这个系统的结构，可以简单化，也可以让不同技能的程序员负责不同的层，而且每1层相互独立。通常来说，我们在层次结构上，头会要求上层的代码调用下层的代码而不允许下层代码去调用上层的代码，所以我们可以把每1层把它分隔开，进行独立测试。代码的变更，也有可能控制在1层之内，不会影响上层或者是下层的其他的代码。

同时，由于采用多层的结构，我们可以在不同的层使用不同的技术来实现。

比如，在视图层，我们可以用Vue框架或者微信小程序来实现用户的界面。

在控制器层，我们可以用阻塞式的springMVC来实现控制器层的API。

在我们的服务层以及数据访问层，包括映射层，我们往往会借助Spring容器中间的Bean对象来实现，其中的数据访问层，为了提高速度，我们可以在这1层使用Redis来实现对象的缓存.在映射层啊，我们也可以选用Mybatis或者 HIBERNATE来实现对象模型向关系模型的映射。

最后1层的这个实体层，它是不依赖于任何的技术的，通常我们把它包装成为一个POJO对象，也就是Plain Old Java Object。

分层的结构，我们可以把它部署在不同的物理服务器上去，比如说我们可以把，视图层部署在web服务器上。把其他的层部署在这个应用服务器上，数据存储在数据库服务器上。这样的话，我们就形成了一个由三台服务器构成的应用系统，可以大大地提高我们系统的吞吐能力。

# 1.2 Spring框架核心--Spring技术栈

Spring5.0中间包含了很多的技术框架，其中它的外部框架就包含了两个技术栈，分别是传统的阻塞式Servlet技术栈和非阻塞式的Reative技术栈。

**Servlet技术栈**是传统的基于Tomcat和Netty Servlet容器的这样的一个技术栈，他通过这些容器所提供的Servlet API来实现服务器端的应用。

当客户端向服务器端发起请求的时候，服务器的会针对每个请求开启一个线程来处理，那当然有多个客户端同时访问服务器的时候，服务器就会开启多个线程，并行的处理，每一个请求在处理完成了以后，才会把线程空闲出来给下一个请求来使用。

由于服务器的内存是有限的，我们通常都会在服务器上设定一个线程的上限，所以当同时来的请求大于了服务器所设定的线程上限的时候，请求就会被阻塞起来，就是处于等待状态。直到有线程被空闲出来，他才会继续运行，所以这样的一个技术，我们把它称之为阻塞式的Servlet技术栈。

而**非阻塞式的Reative技术栈**，采用了完全不同的办法，它采用了**异步**和回调的方式。当客户端发过来请求以后，他不会用一个独立的线程去处理这个请求，而是会把它转成异步的方式，把结果通过消息或者回调的方式，来返回给返回者。所以请求并不需要在服务器里去等待结果的产生，而是采用异步的方式，使得我们对于服务器的开销可以减少很多。这样，通过使用Reative的技术栈，我们只需要用极少的容器的线程，就可以处理非常多的并发用户的请求，而且所有的请求都不会阻塞。

这两种技术栈，由于它实现的方式是完全不同的。分别是我们传统阻塞式的这种方式和基于异步和回调的这种非阻塞的方式，所以技术栈里头的所有的**技术框架都是不同的**。

在Servlet技术栈中间，采用了SpringMVC来实现我们服务器端的控制器层，来处理HTTPRequest和HTTPResponse的这样的一个请求和回应。在数据层，它采用了我们熟悉的JDBC和JPA去访问关系数据库，也可以去访问Redis和Mango的NoSQL数据库。在这种阻塞式Servlet的技术栈中间，一个请求发送到服务器端，交由SpringMVC去处理，那SpringMVC会把处理的业务，交由后面的服务层或者或者数据访问层去访问数据库，或者访问Redis，去拿到相应的数据返回回来，然后通过SpringMVC产生HTTPResponse，返回给客户端，这就是一个完整的从请求到回应的过程。在全部过程中间，服务器都会用一个独立的线程来处理请求和产生这个结果，直到结果产生以后，这个线程才会被释放出来。

那Reactive栈的这一边，它采用了Spring WebFlux作为控制器层，他就不再需要说请求过来，去等待请求执行完成以后，再把结果返回回来，而是请求过来以后，调用者就可以直接去做其他的事情，他可以等到结果产生以后，再把它发送给调用者。由于是这样的一种机制，所以说，他不能去访问我们传统的关系数据库，只能去访问NoSQL的数据库。

在这两种技术栈中间，其实我们在大多数情况下，使用的是Servlet技术栈，因为这样的一个技术栈比较符合我们传统的习惯。但是在少数的场合，特别是在一些高并发的场合，我们访问的数据，可以存储在Redis或者NoSQL的数据库里，那我们可以采用这种WebFlux，这种非阻塞的技术栈来实现我们的服务端应用。在本课中间，我们主要介绍的是Servlet技术栈。

# 1.3 Spring框架核心--Spring容器

Spring的容器。Spring容器是Spring核心的部分。它主要的作用是帮助程序员处理大量的繁琐事务，很多程序员只需要用代码去实现他所关注的业务，而不需要去处理其他的事情，在spirng的框架中间，程序员其实是不需要自己去创建**对象**的，也不需要去建立**对象的关联**。所有的这些东西，他都需要通过配置的信息告诉Spring的容器。

由Spring容器负责去创建对象，去建立对象的关联。Spring的容器不仅仅会创建对象，而且能建立对象的关联，而且它还能够管理这些对象的全生命周期，从创建到销毁。

例如说我们的系统的服务层，分成了用户服务，运费服务，权限服务，还有商品服务四个对象，那我们知道这四个对象是有相互关联的，比如说权限的服务和用户服务是有着密切的关联，我们只需要提供给Spring的容器这四个类的代码，用注解或者是配置文件告诉Spring的容器这四个服务之间的关联关系。Spring的容器在启动的时候，他就会负责创建好这四个对象，而且把这些对象进行相互的关联。那这样的话，我们就可以直接从Spring的容器中间拿到这些对象，而且可以很方便地去使用它。

从刚才我们可以看到，我们需要用配置信息告诉Spring容器，如何去创建对象以及如何去建立对象的关联，那当然，我们可以有**多种方式**去告诉Spring容器这些**配置信息**。

在最早期的时候，我们会采用XML，XML的这样的一种配置信息，它的好处在于说他的配置信息跟代码是分离的，是一个放在外部的独立文件，所以说，我们可以直接去修改配置，而不需要去编译代码。那当然，弊病就是XML的配置比较繁琐，而且它的配置信息跟代码是分离的，同样，我们也没办法去检查出XML中间配置的错误，只有当程序运行到使用配置的这一部分的时候，我们才会发现配置的错误，这样的话，对我们的测试其实产生了很大的压力。

所以说，在后期我们主要采用了另外两种方式来告诉Spring容器我们的配置信息，他就是，我们可以用**JAVA代码**来写配置信息，也可以用**注解**来写配置信息。那当然，我们知道，因为JAVA代码和注解，是在写代码的过程中，就能分析出他的错误，所以说，我们在测试的时候也可以用很多工具去判断，我们的测试是否有覆盖到了这部分的代码，或者说注解，所以说，这是我们目前来说，主要的主流的来写配置信息的方式。

当然用JAVA代码或者用注解来写配置。我们还是会需要写很多的配置，所以在Spring中间，它提供了一个SpringBoot，大量采用了这种默认的配置，减少了我们写配置的工作量。所以很多的配置，你不再需要写了，可以用SpringBoot，用默认方式去指定，我们要写的只是说特别的，不能用默认去表示的这些配置，我们需要用JAVA代码或者说用注解来告诉Spring的容器。

我们来看几个简单的用注解告诉Spring容器的例子，我们前面说了我们先要告诉Spring容器，类是要由Spring容器创建出来的对象，我们怎么告诉他呢，我们可以用注解的方式也可以用代码的方式。我们在这个课上讲的例子是用注解的方式，也就是说我们在类前面可以加一个注解**@Component**，那Spring容器在读到这样的一个注解的时候，**他就知道，这个类是需要创建出来的对象**，对象创建出来，在Spring中是有一个ID的。ID，当然可以作为@component一个属性告诉他，但是，很多时候我们都不会去写这个属性。那他的ID就默认为首字母是小写的类名。

除@Component以外，我们其实还有很多其他的名字告诉Spring容器，这是一个他需要创建的对象。比如说对于控制器的对象，因为它是一种很特别的对象，他要负责处理HTTPRequest、HTTPResponse的请求，所以我们需要在类前面加上一个**@Controller**标签来**告诉Spring的容器这是一个控制器的对象**。那对于其他层的对象。其实，他就是一个普通的Spring的对象，比如说服务层的对象，比如说数据存储层的对象。当然，我们可以用@Component来标志说，这就是一个Spring的普通的对象，但是，为了让代码的可读性更好SPRING的框架提供了@Service，@Repository这些标签来，分别用来描述Service层的对象或者数据存储层的对象，但是它的作用是跟@Component完全一样的。

前面我们说了Spring容器是Spring的最核心的部分。SPRING的框架提供了**两种容器**分别是BeanFactory和ApplicationContext（由BeanFactory派生出来）。

BeanFactory是Spring框架的最基本的容器，我们前面说的读取Bean的对象的配置信息，把Bean对象实例化出来，加载上来，然后把Bean对象实例化出来，维护这些Bean对象之间的关联，然后要负责销毁这些Bean对象，这些事情都是由BeanFactory来完成的。那**BeanFactory逻辑是，当他第一次访问到一个对象的时候，他才会创建对象。**

**在大多数情况下，我们不会使用BeanFactory这样的类对象，而是会使用ApplicationContext**，ApplicationContext是由BeanFactory派生出来的容器。我们都知道派生的概念，所以BeanFactory所有具备的特性ApplicationContext也具备，但是他BeanFactory这些特性的基础上，增加了一些高级的功能，比如说MessageResource，MessageResource，国际化的接口，以及比如说ResourceLoader，可以去加载外面资源的接口，比如说还有一些ApplicationEvent的publish这样的一些接口，应用事件发布的这些接口。所以说它是在BeanFactory的功能基础上，增加了很多面向实际的这些功能。

那他与BeanFactory最大的一个不同是，**他不是在第一次使用对象时创建对象，而是在容器启动的时候，就会创建所有的对象**，就是在配置定义中所有找到的对象，他都会创建出来。这样的话，其实可以有效地提高我们系统运行的效率，当系统启动的时候可能稍微慢一点，当系统启动起来以后，所有的对象都已经创建出来，所以当你用的时候，我就不需要再去花时间创建对象了，直接就可以用了。

我们说Bean的容器，他会负责从对象的创建到销毁的全生命周期，如果说我们需要知道对象的创建过程是个什么样的，Spring的容器其实也提供了一系列的接口，让我们可以在整个对象的生命周期的不同的阶段，去做一些处理，那这些接口被分成两类，**一类是Bean级的接口，一类是容器级的接口**，所谓Bean级的接口，是说他只会影响特定的一个Bean的生命周期，就是在特定的一个Bean的生命周期里头才会去作用。而容器级的接口则是在这个容器中间，所有的Bean都会起作用。

我们首先介绍一下Bean生命周期的接口，在Spring容器中间Bean生命周期的接口，一共有5个：

BeanNameAware：用于获得Bean对象的名称

BeanFactoryAware：用来得到BeanFactory容器对象

ApplicationContextAware：用来得到ApplicationContext的容器对象

InitializingBean：Bean对象的创建的时候的接口

DisposableBean：被废弃时候的接口:

这五个Bean级的生命周期接口，只是针对特定的Bean对象才会起作用的，不会对所有的Bean对象去起作用。

那如果说某个Bean对象是需要使用这些接口的，那Bean对象就可以去实现这些接口接口，那自然就会有些接口中间的方法，这些接口中间的方法，就会在生命周期的不同的阶段，会被调用，从而你可以去做一些你在阶段想要做的事情。

另外一种接口是容器级的生命周期接口，容器级的生命周期的接口主要是两个，一个**InstantiationAwareBeanPostProcessorAdapter**，这个名字比较长，而另外一个呢，**BeanPostProcessor**，容器级的接口，是对在容器中间所有的Bean对象都会起作用的。

那在生命周期的中间，这两类接口是按照一个什么样的顺序调用的呢，当我们去实例化一个对象的时候：

首先被调用的是容器级的InstantiationAwareBeanPostProcessorAdapter，这个接口中间的postProcessBeforeInstantiation这个接口。

那这个接口被调用以后，他则会去调用Bean对象自身的构造函数。

Bean自身的构造函数调用完了以后，他在会去调用另外一个容器级的接口叫InstantiationAwareBeanPostProcessorAdapter的postProcessPropertyValue（这个接口已经废弃）。

那这个接口完成了以后，就会对bean对象的属性来进行赋值，Bean对象的属性往往是来自于，我们在配置中间告诉他的，比如说，它的关联，或者说某些配置文件中间的信息是需要放到bean的属性里去的。

当Bean属性赋值完成了以后，他就会调用我们的Bean级的接口：

BeanNameAware的setBeanName，这个可以告诉你Bean对象的名称是什么

然后是BeanFactoryAware的setBeanFactory这个方法上，你可以拿到BeanFactory对象

再是ApplicationContextAware的setApplicationContext接口，这样你可以拿到ApplicationContext对象。

这三个Aware的接口完成了以后，他所调用的是另外一个Bean级的接口，叫做BeanPostProcessor的postProcessBeforeInitialization的接口

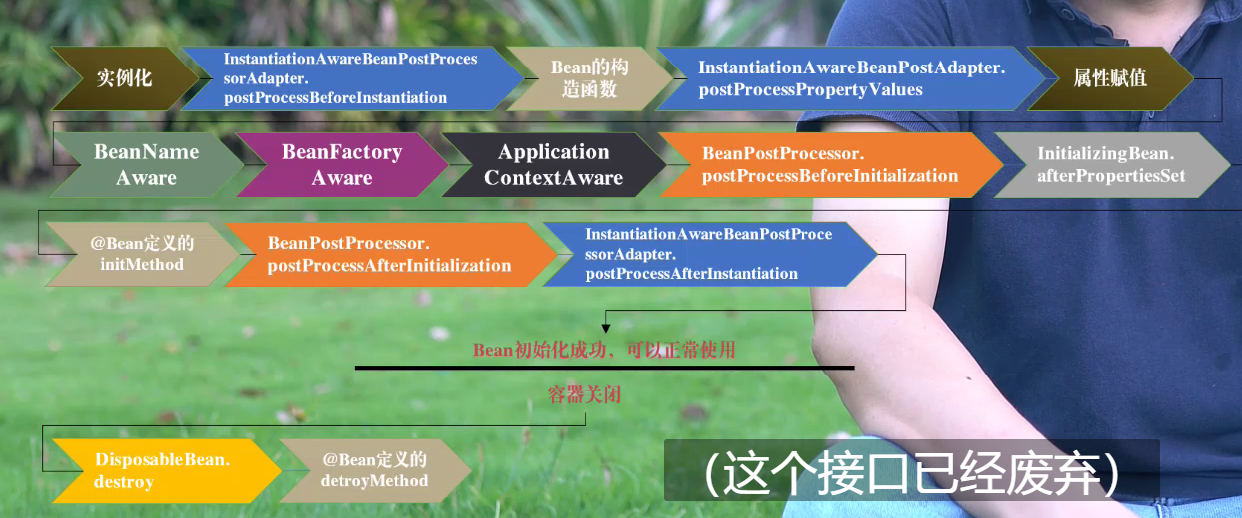
然后完成了以后，则是另外一个bean级的接口InitializingBean的afterPropertiesSet的这样的一个接口。

当这些方法都调完了以后，他就会调用。我们在@Bean的注解中间所指定的initMethod的这个方法。

然后就是我们的容器级的接口BeanPostProcessor的postProcessAfterInitialization的这个接口。

然后就是另外一个容器级的接口InstantiationAwareBeanPostProcessorAdapter的postProcessAfterInstantiation的这个接口。

那两个容器级被调完以后，这个Bean对象就是已经创建成了，你就可以正常的去使用这个Bean对象，那当这个容器被终止的时候，容器中间的Bean对象也会被销毁。他被销毁，它首先调用的是DisposableBean的destroy的方法，然后是我们在@Bean的注解中间所描述的destroyMethod的属性所定义的方法，然后才是最后把这个Bean对象给他销毁掉。



## 1.3.1 Spring框架核心--Spring容器示例

//解释了注解/标签，配置文件的作用

//实例代码展示了bean对象生命周期流程中调用了哪些方法。

下面我们用一个例子代码来看一下，Bean的生命周期的接口是怎样工作的。这个代码是基于Maven和SpringBoot完成的，这两部分内容我们在下一讲中间才会讲到，所以我们先看一下这个代码，不用去管 SpringBoot和Maven的部分。

这个代码非常的简单，我们做了4个类，其中2个类是Spring的Bean对象类，分别是Car和Boss，Boss这个类，我们做的内容相对比较多，我们让它实现了5个Bean级的生命周期接口

，而Car这个类就比较简单，我们只是让它做了一个构造函数。

我们另外做了两个容器级的接口的类，它分别实现了两个容器级的接口，我们来看一下这一些类实现了Bean级的生命周期接口和容器级生命周期接口的这些方法，在这两个Bean的实例化的过程中间，是按照一个什么样的顺序被调用的。

我们来看一下代码，它的代码在我们的例子的工程里头的SpringFramework这一章里头，它的工程名叫做beanlifecycle，我们说这个例子，是用Maven和SpringBoot来写的，我们先不用去管Maven和SpringBoot。

我们直接来看我们所写的两个Bean对象，第一个Bean对象相对比较简单，是我们的Car对象，大家可以看到它只写了一个构造函数，在构造函数中间，就仅仅是用System.out去打印出来一个它的构造函数被调用。我们对象，我们通过配置信息，让Spring容器知道它是个Bean对象，我们的办法是用注解的方式，用@Component这样的一个注解，来告诉Spring容器，这是一个Bean对象。

我们看一下另外一个Bean对象，另外一个Bean对象是Boss，Boss对象我们做的相对比较复杂一点，我们让它实现了5个Bean级的生命周期接口，就是我们三个Aware接口，分别是BeanNameAware，BeanFactoryAware和ApplicationContextAware，还有2个InitializingBean和DisposableBean。这5个Bean级的生命周期接口，有5个对应的方法，所以我们写在了后面，这5个@Overide的方法，就是我们上面5个Bean级生命周期接口所对应的方法。

其他的方法我们可以看到，我们有写了一个属性，在Bean对象里头叫做bossName，我们的 bossName，是有一个对应的get和set方法，其中在它的set方法前面，我们写了一个注解叫@Value，然后里头的名字是$，这个注解的含义是告诉Spring容器，说我的**方法的参数**，就是它这个参数bossName，我**要从配置文件中间读**，所以Spring容器在读到说，它的Bean对象中间的某一个，也可以写到属性前面，那直接设到属性上去了，我们现在把它写在方法前面，它就会从配置文件中间，读取信息，把它设置到这个方法的参数上面。

我们配置信息写在哪儿呢？配置信息写在了application.yaml文件里头，这个文件是在resources目录下，resources目录下放的都是一些配置文件，或者说是静态文件，我们可以看一下，在application.yaml中，我们写了一个配置项，就是Boss.bossName，因为是用yaml来写的，所以说它是用分级的方式来写的，如果说用properties来写的话，它就直接写，Boss.bossName，我们的属性给它设定成为Jack，所以Spring容器在实例化一个Boss对象的时候，它就会从配置文件中间读取值，把它赋值给setBossName的参数上，里头的代码，我们可以看到，它是会把它打印出来，实际上它是一个属性，这是它的 bossName的属性，还有它对应的get/set方法。

另外我们看到它的构造函数，我们还有两个方法分别是myPostConstruct和myPreDestory这两个方法，我们是准备放在@Bean的定义中间来写的。

我们注意一下，这一个Bean对象在类前面是**没有用注解来写**的，我们怎样告诉Spring容器，这是一个Bean对象呢，在我们的Spring配置的Java代码中间，我们前面说过，可以用Java代码来写配置，在这个工程中，它的配置的Java代码，是写在BeanLifeCycleApplication类，因为它前面加了一个注解叫做@SpringBootApplication，这个注解的具体的含义我们在后面的章节再讲，但是这个注解中间，其实也包含了配置的信息，就是它包含了一个配置的内容，所以Spring容器会去读，有加了配置的注解的类，然后去读取里头的内容，在这个类中间，我们写了一个方法，这个方法是返回值是Boss类型，它的方法名是boss，然后里头的内容方法里的内容，就是创建了一个Boss对象，这个方法起什么样的作用，这个方法起的作用是我们前面加了一个@Bean的注解，加了一个@Bean的注解，Spring容器就知道，我需要创建一个Bean对象，Bean对象的类型是什么就是这个方法的返回类型，Boss类型，Bean对象的id是什么，它的ID就是这个方法的名称，就是小写的boss，所以它就会创建一个Boss类型的id是小写的boss的这样的一个Bean对象。通过@Bean的注解，写在这个方法前面，当然这个Bean对象，就是我们在这个方法里头，用代码写出来的，构造出来的对象，这就是我们说的这种，**用Java代码来告诉Spring的框架，告诉Spring的容器，怎样来做配置**，这种做法，在这里@Bean的注解里头，我们定义了它的两个属性，一是initMethod的是我们的myPostContruct，destroyMethod的是myPreDestroy，这就是我们在Boss类里头的两个方法。

我们可以看到，用Java代码来写Spring的配置，信息其实更加的灵活，比如在这个例子中间，我们的Boss的Spring对象，是我们在这个方法中间，用方法来写，用代码来写的，所以假如说Boss是有一些子类的话，其实我们可以根据配置的信息，在这个方法中间去实例化出Boss的某一个子类对象，然后返给Spring的容器，所以说它的灵活性，其实是大于注解的方式。当然如果说，我们不需要这样的灵活性，需求上不需要这样的灵活性的话，其实我们是没必要用Java代码来写的，因为用注解写来的比较简单，这是两种写Spring的框架的配置的方法。

我们来跑，除了这2个Bean对象以外，因为1个Boss里头又定义了5个Bean级的生命周期，另外1个Car里头，其实什么都没有定义，我们还定义了两个容器级的接口，因为它不是针对特定的Bean对象，所以你需要独立的写一个类，来实现**容器级**的接口。我们首先看第一个实现了InstantiationAwareBeanPostProcessor这样的一个接口的类，这个类里头，我们就直接实现了接口所要求的两个方法，在方法里头都是打印出来对应的信息。另外一个的类也类似，我们也是实现了BeanPostProcessor所定义的两个方法，去打印出它的内容。

我们把整个工程跑起来，因为我们前面说过，容器在启动的时候，它会把里头所找到的这些Bean对象，全部实例化出来，那么我们就可以看到，我们所定义的这些生命周期的接口，在整个Bean对象的实例化过程中间，是按照一个什么样的顺序来运行的。

跑的时候，我们要依赖于SpringBoot的，因为我们在版本中间有定义了一个Plugin是SpringBoot的Plugin，所以我们在Maven，具体的我们在下次课再会讲，我们这次只管用了就好了，我们在Maven这一边找到这个对应的工程，找到它的Plugins，然后找到spring-boot，找到spring-boot:run，我们让它跑起来，因为我们写的是服务器端的程序，其实它是没有界面的，我们这些信息都需要通过命令行来看到。我们在程序中间所跑出来的这些，所打印出来的这些信息，当你看到SpringBoot的提示的时候，就说明SpringBoot已经正常地运行起来了，所以我们可以看一下，运行起来的SpringBoot，打印出了一些什么样的信息。

有很多，因为我们这里定义了，两个容器级的接口对象，因为我们容器级的接口对象，是对Spring中间的所有的Bean对象都会起作用的，Spring容器里头不仅仅有我们定义的两个Bean对象，其实它内部的很多的东西也是Bean对象。我们可以看到，这里有很多提示，就是因为我们容器级的这个接口，是对它自己内部定义的这些Bean对象都起了作用，所以就打出了一大堆的提示。

我们关心的就是我们自己的两个Bean对象的提示，所以我们找一下这个Car，在前面Car的比较简单，因为Car里头没有定义Bean级的生命周期接口，所以我们可以看到它的顺序，首先是InstantiationAwareBeanPostProcessorAdapter的postProcessBeforeInstantiation的接口，然后是它的构造函数，然后再是同样接口的postProcessAfterInstantiation的接口，然后再是BeanPostProcessor的两个接口，一个是它的postProcessBeforeInitialization和postProcessAfterInitialization这两个接口，这样就是Car，这我们定义的，因为你可以看到它所起作用的方法都是容器级的，那两个接口的4个方法被调用。

另外1个是Boss的因为我们定义了5个Bean级的生命周期接口，所以你就能看到它的过程，首先跟前面的Car一样的是这个InstantiationAwareBeanPostProcessorAdapter的postProcessBeforeInstantiation，这是Boss的，然后是它的构造函数，然后是同样接口的postProcessAfterInstantiation，在这个接口完成以后，我们可以看到 Boss的Bean对象的属性，是从配置文件里面读进来的，所以这时候你能看到它的setBossName被调用，因为我们在前面配置的一个需要从配置文件读取属性的这样的一个方法，所以你可以看到这个方法被调用了，它调用以后设的bossName是什么，然后是我们的三个Aware的方法BeanFactoryAware的方法和ApplicationContextAware的方法就会还有BeanNameAware的方法。首先是BeanNameAware的方法BeanFactoryAware的方法和ApplicationContextAware这三个Aware的接口，被接连调用。然后就是容器级的BeanPostProcessor的postProcessBeforeInitialization的方法被调用，之后是我们的InitializationBean的bean级接口afterPropertiesSet的接口，然后是它的@Bean中间所定义的initialMethod的方法myPostConstruct，再然后就是另外一个容器级的BeanPostProcessor的另外一个方法，postProcessAfterInitialization被调用了。这样的话，Boss对象就被创建完成了。

下面看到的其实是它内部的一些Bean对象，因为我们定义了两个容器级的接口，所以你可以看到这些容器级的接口，对于不仅仅对于我们定义的Bean对象会起效果，其实对于在Spring容器中间它自己的一些内部的Bean对象，也会起效果，所以说我们称之为容器级的。

最后我们可以看到Bean对象的销毁过程，这Bean对象销毁的时候，我们的DisposableBean因为只有在Boss里头定义了一个Bean级的DisposableBean的这样的一个接口，它的destroy方法被调用了，这个在前，然后是我们在@Bean中间所定义的 destroyMethod的myPreDestroy的方法被调用，这样的话，对象就被摧毁了。

# 1.4 Spring框架核心--控制反转

控制反转，是Spring容器的一个重要的特征。所谓控制反转，是指在Spring容器中间，Bean对象的依赖，其实是不需要自己来管理的。我们之前说的，我们可以通过配置信息，来告诉Spring容器，Bean对象之间的依赖是什么，但很多时候，Spring容器是可以基于某一种规则，去自动的管理Spring容器中的所有Bean对象之间的依赖的。

Spring容器是怎样来管理这些Bean对象之间的依赖的呢，它采用的是**依赖注入的方式**，来实现控制反转的，它怎样采用依赖注入的方式，我们需要通过**注解的方式**，来告诉Spring的容器，说这些**Bean对象之间，是会相互关联的**。我们最常用的一个注解，是用@Autowired。在我们的Bean对象的属性，或者是在Bean对象的方法的前面，用@Autowired的这样的一个注解，来告诉Spring的容器说这样的一个属性或者说在方法里头的参数，是需要由Spring容器，来注入对象。

Spring容器会首先去查询对应的，类型是相同的Bean对象，如果同一个类型的Bean对象，只有一个，它就会把Bean对象，装配给我们的属性，或者说装配给我们这个方法的参数。如果说@Autowired所绑定的数据类型，其实它得到的结果，不是一个的话，@Autowired会根据变量的名称来查找，也就是如果说，我们绑定的属性的名称，或者说在方法中间的参数的名称，跟Bean对象的id是一样的，它就会把它对上来，我们通过代码来看一下，@Autowired是怎样来使用的。

我们来看一个Spring容器自动装配的例子，我们在这个例子中间，做了三种，都做了几个

让它自动装配的类，分别是Boss，Car和Office，我们把Car和Office，装到Boss里去。

我们在前面说到，装配的方式可以有多种，我们可以把它写在属性的前面，我们可以把它写在构造函数的前面，我们可以把它写在set方法前面，所以我们的Boss做了三个类，分别对应了三种不同的，把Bean对象把它装配上来的这种方式。

而Car对象和Office对象，我们做了一点特别的处理，对于Office对象来说它就比较简单，它就是一个普通的对象，但是Car我们实际上做的是一个接口，我们在Car底下做了两个实现类，分别是Toyota和Haval做这两个子类的目的是，因为我们想看一下，在自动装配的时候，如果说在Car这个类型下头，有两个不同类型的实例的话，它是怎样来装配的，我们在课件中说到，它是通过id，来确认它的，来确认最后装配的是哪一个对象。

好，我们来看一下这个代码，自动装配的这个例子。在第一章的SpringFramework里头的autowiredemo这个工程底下，我们建了几个Bean。

首先来说，我们建了一个非常简单的Bean叫做Office，我们是用注解来表现他的，这个Bean里头没有任何的实现。

第二个我们建了一个接口，我们建了一个接口叫做Car，因为它是个接口，所以你大家可以看到，前面是没有任何注解的。我们有两个类来实现了Car的接口，分别是Toyota，大家可以看到是个Component，我们都没写任何内容，因为我们只是想看一下，它的自动绑定的过程，它实现了Car的接口。另外一个是Haval，哈弗汽车，它也实现了Car的接口。

我们是要把这两个两种对象，就是Car和Office，要注入到Boss里头，Boss我们写了三个类，分别用来演示，它的三种不同的注入方式，第一个我们看到的是，Boss\_constructor，因为我们想通过它的构造函数，注进来，在这个例子里面，我们首先看到的参数是用的Car类型，我们在构造函数中间，有两个参数，一个是car，一个是 office，在这个构造函数中间，如果是Car类型的话（而且参数名是car的话），我们知道在代码中间，它其实会出问题的，因为你跑起来的时候，它其实并不知道，你是要把Toyota注射进去，还是要把Haval注射进去。

所以我们先跑一下这段代码，看看它的这个问题在什么地方，跑一下，所以跑起来的时候，你会发现它会报错，它报的错是什么呢，报的错是说它的APPLICATION FAILED TO START，应用起不来，失败的原因是什么，是因为Parameter 0 of Constructor，我们Boss\_constructor的，这个方法的，第一个参数，它需要一个，需要一个Bean对象，但是在Spring容器中间，按照配置信息，它能找到两个，分别是我们的toyota，和我们的haval，所以这样的话，它就没办法把它绑定上去了，我们可以把这代码略微改一下，我们不改它的类型，我们只是把它的变量名称，改成toyota。

因为我们在这里头，它其实没有，所有的Bean对象都没有定义id，所以它的id，就是类名的首字母小写，所以Toyota类型的，Bean对象的id应该就是，toyota，我们把这个改一下，改一下以后我们再跑，再来，spring-boot:run去跑一下，去跑的话，你就发现，它这个就没有问题了，它跑起来，不会再报错，这Boss\_constructor。

其实其他的也类似，我们看一下，它通过property来注进去的，property来注进去的话，它是不需要写构造函数，写在构造函数前的，不需要写在方法前头，它直接就写在了属性前面。这样的话，当它实例化出来的时候，它会把根据这个类型去找对应的，所以我们在这个例子里面，我们可以看到，我们的car，注射的不是toyota，注射的是haval，这个跑起来就不会出错的。

第三个是通过set函数，set函数的话，我们又是注入的toyota，在setCar前面，虽然它是Car的类型，在它的属性中间，它也是Car的类型（属性名称也是car），但是我们在它set函数的参数上头，给它的名称叫做toyota，这样它就会用id去找，car类型，下有id，叫做toyota的，Bean对象把它注入进来。

这个是我们的自动装配的，这样一个过程，我们现在已经没有错了，但是跑起来以后，我们怎么知道说它的自动装配，是不是OK的，我们利用了，我们在下一章，要讲的一个Actuator的接口，我们通过了另外一个工程，一个有界面的工程，叫做得Admin UI，这个部分，我们再下次再仔细讲，我们今天先只是看一下它的结果。

我们把界面的工程给它跑起来，界面的工程，我们在配置文件里面，有定义了，说它的端口在8081，因为我们的autowired这个端口，要放在8080，他们两个别打架了，所以我们再把autowired这个端口，也这个工程也跑起来，然后跑到了8080。

好，两个工程都跑起来以后，我们就可以看一下，8081端口上头的，我们，这个是Admin UI的这个工具，能够看到说，这个服务器集群中间，我们到底跑了多少这个，SpringBoot的Application，当然现在就有一个，你看到这个Application（应用数），只有一个我们要看的，其实是这个Bean对象，因为它是利用 Actuator的接口，就可以看到这个Bean对象的一些配置，所以我们就看一下，到它的WallBoard（应用墙）的里头去看一下，我们Application的配置是什么，我们关心的是，它这里头有有绑定了多少Bean，又起来了多少Bean，所以有它内部的Bean，也有我们定义的。

我们首先看到的是，boss\_Setter的Bean，我们可以看到，它绑上去的是toyota，虽然它的Bean中间，定义的类型是Car类型，是接口类型，但是因为它用id，找到了这个toyota对象，所以它绑定的是toyota，boss\_constructor绑上去的，也是toyota， boss\_property，绑上去的是，是我们的havel的这样的一个对象，这样就能看到说，它最后把绑定上去的东西，都是不同的类型的对象，根据id去找上去的。

# 2.1 Maven简介

maven是apaqi底下的一个一个纯Java的开源项目，它主要是用来管理一个项目的构建、依赖、以及生成报告和文档等步骤。Maven是与IDE无关的，所以无论在哪个IDE的平台上，Maven都是提供了一样的操作接口。

Maven最主要的作用，就是用来编译打包我们的应用程序，它的第二个作用呢，只是用来去管理我们在程序中间，所需要用到的Jar包。

Maven管理Jar包的方式是，它提供了一个中央仓库和本地仓库。我们用Maven来管理Jar包，并不需要把Jar包引入到我们工程里头，只需要在去描述它需要使用什么样的Jar包。Maven会从中央仓库中间，把Jar包下载到本地仓库里，然后从本地仓库里头，关联到我们的工程里头。所以如果说Jar包在我们曾经用过，在本地仓库里头有，我们就不需要从中央仓库中间下载了，如果说本地没有的话，它是需要从中央仓库下载的本地的Maven仓库，默认在我们每个用户的用户目录的.m2目录底下。所以如果在Maven的Jar包出现了混乱的时候，我们可以选择把本地仓库，也就是用户目录中的.m2目录删除掉，这样的话，Maven就会从中央仓库，重新来下载这些Jar包过来。

对于在我们项目中间的所有的文件，Maven提供了一个标准的目录结构，这个目录结构会规定说，我们的生产代码，测试代码，放在什么样的目录底下，以及这些目录的命名方式。这样所有的程序员，在协同工作的时候，大家所有的东西，都会按照一定的规定来存放。Maven的默认的目录结构是这样的，它的目录主要是分为两个目录，一个是src，这个是所有的源代码，以及资源文件，都放在source目录底下，而另外一个目录是target目录，它所有编译出来的代码，以及所有的要用的资源文件，都会拷贝到 target的对应目录上头去，然后最后打出来的Jar包，或者War包，也在target的目录里头。除了这两个目录以外，它还需要一个POM文件，POM文件位于整个目录的根目录底下，所以从根目录进去看的话，一个Maven的工程就是一个，POM文件，和一个src文件夹，如果说它有经过编译，打包了以后，它还会出现一个target的目录。

我们来详细看看，src目录底下的结构，点开src的目录，你就能看到，一个main目录和一个test目录，main目录放生产代码，和生产的资源文件的，而test目录，是用来放测试代码，和测试用的资源文件，在main底下再点开看的话，就能看到一个java的文件夹，和一个resources文件夹，所有的java源代码，都放到了 java的文件夹里头，而resources文件夹里头放的是各种各样的配置文件，以及一些静态文件，都会在resources的这个目录底下。Maven的工程在编译的时候，会把resources目录里头的文件和Java目录中的Java代码编译出来的.class文件，进行合并，最后，根据它的各自的目录结构，拷贝到一起，然后最后打包形成我们的Jar包，或者是War包，这个是src目录。

看一下test目录，test这个目录，你可以看到，它的结构跟src是一样的，java底下放的是测试的代码，而resources的目录底下，放的是测试所用的资源，这是Maven标准的目录结构。

Maven管理这么多东西，但是它所有的信息，都是放在POM文件中间的，POM文件定义了项目的一些最基本的信息，描述了用户如何去构建、打包这样的一个项目，以及声明说，在这个项目中间，会要引入多少Jar包，也就是我们说项目的依赖，使用这样的一种POM文件，可以大大简化我们的开发和构建的过程，使得我们的整个的项目管理和具体的IDE无关。

我们来看一下autowiredemo工程的POM文件，因为它是一个非常简单的工程，

所以里头的POM文件的内容不多，我们主要来看一下，POM文件的一个主要的结构，在autowiredemo的工程的POM文件中间，我详细的把主要的标签的注释都写了，我们一行一行的来看，我们首先看到的是一个标记，这个标记标识本文件，是一个XML文件，它的编码是用UTF-8来写的，POM文件的内容，都在project标记里头，project头不要去动它的，下面modelVersion，也不要去动它标识的是版本，是4.0.0，我们真正写的内容从后面开始，后面我们看到parent，它定义的是，POM文件的继承关系，POM文件是可以相互继承的，它可以继承父POM文件中间的所有的定义，然后再当前POM文件，可以去修改，以及增加新的定义，跟面向对象的定义，是完全一样的，在parent的后面，我们看到的是，groupId以及artifactId，这两个Id合起来，构成了我们这个工程的唯一的标识，然后是它的版本号version，我们每次发布新的版本都需要给它一个版本号，之后是一些描述信息，描述信息后面我们看到的是打包的方式，对于SpringBoot的工程来说，它主要是打成两种格式，一种是带了Servlet容器的Jar格式。还有一种是不带Servlet容器的War格式，War包的话，你是需要，把它部署到Servlet应用服务器里头的packaging后面，我们看到的是，properties的定义，property为了去定义一些在POM文件中间，我们会重复用到的一些值，比如说版本号等等，所以这里用property定义以后，后面就可以直接去引用property了，当然我们可以在这里去设定，Maven的一些默认的属性，这里我们可以看到，我们设定的工程构建的时候的字符集的格式，用UTF-8，POM体后面看到的是依赖关系的定义，Maven的一个主要的工作，就是来管理，在工程中间所需要用到的各种各样的Jar包，所以在这个过程中间，所需要用到的Jar包，

都在dependencies里头写了，每个dependency的是就是一个独立的Jar包，dependencies后面看到的是Maven的插件，我们知道Maven是依赖于插件，来完成各种各样的功能的，所以说你需要用到哪些插件，就在plugins里头，把它写进去。最后看到的这个是仓库的定义，如果不加这部分的话，用的是在国外的中央仓库，会使得我们的Jar包下载的比较慢，所以我们一般会定义一个国内的Maven仓库的镜像，我们在这个例子中间看到的，定义的是阿里云的

Maven的仓库镜像，这就是一个最简单的。

# 2.2 Maven生命周期

我们前面说了，Maven是用来负责管理项目的构建的，这是它的最主要的功能之一，Maven是怎样来管理，这个项目的构建过程，它主要通过生命周期，来定义我们项目的构建顺序。

在Maven中间，它定义了三个生命周期。

**第一个**是Clean生命周期，这个周期主要是用来做清理工作，也就是删除我们上一次构建中间，所产生的各种各样的文件，比如说我们知道，这些文件，都存在于target的文件夹底下，默认情况下都是这样的。

**第二个**生命周期是构建生命周期，也叫Default生命周期，这个是Maven的最主要的生命周期，我们的项目的编译、测试、打包、安装部署等等，都在这个生命周期里头。

**第三个**就是Site生命周期，是用来生成我们的项目的一些报告的，比如说我们的测试报告，比如说我们API文档等等，我们可以把我们项目的测试报告，API文档，发布到某一个Web服务器上去，Maven可以根据我们在POM中间所包含的这些定义信息，生成我们所需要的文档，以及把这些文档，发布到一个指定的Web服务器上，方便团队的相互的交流。

每一个生命周期中间，包含了一系列的阶段，这些阶段是按照顺序来执行的。

Clean生命周期比较简单，它只包含了三个阶段，分别是pre-clean. clean和post-clean，clean阶段是最主要的阶段，因为在这个阶段中间，它会清理上一个，构建的时候，所生成的文件，而pre-clean是在clean之前的，它是做一些预备工作，post-clean是在clean之后的，它是做一些善后的工作。



Default生命周期的阶段比较多，因为它是Maven的主要的生命周期，我们一个个的把它的阶段，给大家介绍一下。

第一个阶段是validate阶段，这个阶段是来验证项目中间，每个目录底下，它所需要的资源，是否是可用的。

如果是可用的，它就会进行第二个阶段，initialize，initialize的作用，是做一些初始化的工作，比如说设定一些属性，创建一些比如说以后需要的目录。

第三个阶段是generate-sources，在这个阶段，可以用一些插件，去生成一些额外的源代码，比如说我们在后面会看到，我们的POJO对象，我们的一些模型对象，都会需要get/set方法，但是我们不想在每一个模型对象中间写一堆的get/set方法，所以我们会用一个插件，叫做Lombok这样的一个插件，去生成这些POJO对象的get/set的这些代码，这个插件就是用在generate-sources这个阶段，用来去生成源代码。

process-sources的主要作用，是去处理源代码，它会对源代码里头的一些值进行替换，等等这样的一些操作。

generate-resources只是用来生成资源文件的，资源文件，是我们的一些配置文件，比如说在Mybatis中间，所需要的XML，这都统称为资源文件，所以如果说这些资源文件，要用插件自动生成的话，在generate-resources这个阶段产生。

process-resources的作用相似，它就是用来处理这些生成的资源文件的，可以做替换，可以为打包去做一些准备等等，生成了源代码生成了资源文件以后，我们有了这两个东西，我们就要来做编译。

所以我们看到下一个阶段，就是compile，这个阶段，就是把我们的源代码编译成为.class文件，编译成为.class文件以后

我们的下个阶段称之为，process-classes，这个阶段是干什么，这个阶段的作用，我们是可以对编译出来的文件去做一些处理的，比如说，大家有兴趣的话可以去查一下，比如说可以对.class文件，作为一些字节码的增强处理，可以增强它的一些功能，这样一些插件，就要放在process-classes这个阶段，来运行

然后我们测试部分，分为generate-test-resources，这是用来生成测试代码，测试的源代码，

process-test-sources，是对这些源代码做一些处理，比如说替换一些值，

generate-test-resources，是用来生成测试用的资源文件，

process-test-resources，说则是处理，我们的这些资源文件，

test-compile是用来编译我们的项目的单元测试代码的，

process-test-classes，这一个阶段都是跟前面类似，是对我们编译产生的，test的class文件，就是测试的class文件，来进行一些后处理，比如说做字节码的增强操作等等。

完成了这些test的阶段以后，我们就会来到真正的 test的阶段，这是来做单元测试的，我们会使用一些单元测试的框架，来进行单元测试，这里要注意一点，在这个阶段，测试代码只是在运行，它并不会被打包，也不会被部署到我们的服务器上去。

之后如果测试完成了以后就是prepare-package，这是打包之前的预备工作，通常来说是做解压，比如说处理包的版本

然后是打包的阶段package，这个阶段，我们会根据我们在POM中间的定义，创建Jar包或者是War包

打包完成以后是pre-integration-test，这个阶段，主要是做一些集成测试之前的预处理工作，包括建立相应的环境等等

而之后的阶段，就是integration-test，这个就是把我们的Jar包，部署到我们的服务器上去，在测试环境中进行集成测试，

之后是post-integration-test，这是测试之后的环境的清理工作，

然后是verify这个阶段，这个阶段，是对集成测试的结果进行检查，看它是否质量达标

如果质量达标的话，我们就进入到install的阶段，这个阶段是，把这个项目打包的项目，部署到本地的Maven仓库里头，也就是我们的用户，目录的.m2文件夹里头，这样的话，在本地的其他的项目，就可以在本地仓库中间找到，我们打上去的Jar包。

最后一个阶段是deploy，也是可以把我们的jar包，部署到远端的仓库里去，整个项目组的其他的成员，就可以访问到，我们做好了Jar包的Maven依赖了。



第三个生命周期，Site的生命周期，它一共包含4个阶段，分别是，pre-site，post-site，site和site-deploy，这4个阶段， pre-site和post-site，是分别做一些，生成项目文档的，之前和之后的处理工作，但真正有意义的是site这个阶段，是用来生成这个项目的相关的文档，而site-deploy这个阶段，就是把生成的项目文档，发布到我们指定的Web服务器上去。



我们可以看到，为什么，Maven要定义三个生命周期，因为它的每一个生命周期中间，它的阶段是有先后顺序的，当我们运行，当我们执行一个阶段的时候，在生命周期之前的所有的阶段，都会执行，那这三个生命周期之间，是相互独立的。

比如说我们在执行 Default生命周期，中间的阶段的时候，它不会去执行，在Clean生命周期中的阶段，以及在Site生命周期的阶段，它只会执行在Default生命周期，这个阶段之前的那些阶段。

所以我们说三个生命周期，是线性独立的，这样的一个设定，就会使得说，我们的管理会变得比较简单。

比如说我们需要做清理的时候，我们就需要去执行，在Clean生命周期中间的阶段，我们需要去发布项目文档的时候，我们就执行在Site生命周期的阶段

如果说我们需要编译打包和部署的话，我们就去执行，在Default生命周期中间的这样的一个阶段。

这样的话就大大减少了我们，执行命令的这样的一个输入，我们只需要指定一个阶段，它之前的阶段就都会执行。

在Maven中，生命周期的阶段，定义了干活的顺序，但其实它的阶段里头，并没有真正去干活，真正干活的是什么，真正干活的是我们在Maven中间的插件，生命周期的每一个阶段，都会跟插件进行绑定啊，所以说当我们去说，执行到这个阶段的时候，它其实是执行这个阶段，所绑定的插件的功能，那插件有些什么样的功能呢，我们在下一段再来讲。

# 2.3 Maven插件

springboot是是maven的一个插件。

Maven的插件，我们前面说Maven，其实真正干活的，是靠插件来干活的，在生命周期中间的阶段，只是定义了一个顺序，每个阶段是需要跟插件，绑定到一起，再来完成相应的任务的。

Maven中提供了很多的插件，每个插件，可以完成多个功能，所以我们把这个插件的每一个功能，我们把它称之为**插件的目标**，叫做Plugin Goal，以我们最常用的插件spring-boot-maven-plugin来说，它里头就包含了多个功能，也就是多个目标。

我们最常用的目标，就是springboot:run，冒号后面的这个run，这个功能主要是运行，SpringBoot的应用程序，所以说它会在，Maven的标准的目录的，结构上头，去找到Jar的文件，也就是我们在target底下找到jar文件，然后把它运行起来。

springboot:repackage，它是进行一个重打包，它就是会重新打jar包，或者war包，把它变成一个可执行的包。

springboot:help，这个是用来去显示，springboot plugin的一些帮助信息。

另外两个功能分别是springboot:start和springboot:stop，它的功能跟springboot:run不同，它是用来启动SpringBoot应用程序，但是它不会阻塞，我们在用springboot:run的时候，我们会发现，我们会运行一个jar包，但是当前的目标是被阻塞了，它就会停在那，等到你执行完了以后，你把它退出来，它才会继续，才会放出来，start是不会阻塞的，它允许你在运行的应用程序，有再去运行其他的应用程序，stop只是用来去停止，我们用start起来的一个spring的应用程序。

这是我们常用的Maven的springboot的plugin，我们一般用的时候，我们其实并没有把它，跟我们的生命周期中的阶段绑定起来，我们只是在Maven的plugin中间，把它引进来，所以我们在，比如在idea中间，我们在Plugins，中间就能看到这个plugin，所以我们就可以在里头，直接去点，它的相应的目标，去完成它的相应的功能。

但是在很多时候，我们实际上是会把plugin和生命周期中的阶段绑定起来的，绑定有两种方式，一种是内置绑定，一种是自定义绑定，内置绑定就是Maven中，它已经绑定好了的，比如说我们在Clean生命周期中间的，clean这个阶段它就被绑定到，默认绑定到，maven-clean-plugin:clean目标里头

Default生命周期，它就比较复杂，因为它要会根据你的package的类型，打包是打jar包还是，打war包，决定这些阶段是怎样绑定的，我们以jar包来说，如果打jar包或者war包的话，这两个是一样的，它的process-resources这个阶段，绑定到maven-resources-plugin:resources功能上，目标上，compile阶段绑定的则是，maven-compile-plugin:compile，这个目标，process-test-resources，这个阶段，绑定的是

maven-resources-plugin:testResources，test-compile这个阶段，绑定的是maven-compile-plugin:testCompile，test这个阶段，绑定的是，maven-surefire-plugin:teste，这个阶段，打包，因为我们现在是打jar包，或者war包，如果是jar包的话，绑定的maven-jar-plugin:jar，如果是war包的话，则是maven-war-plugin:war， install的话，绑定的是，maven-install-plugin:install目标，deploy绑定的是maven-deploy-plugin:deploy的目标，这是当我们要打jar包，或者war包的时候，它Default的生命周期的阶段的绑定的情况

如果说我们的packaging，不是要打jar包或者war包，如果说我们是一个pom，这种情况其实是用在，父子项目管理的时候，它在父项目中间，其实是不需要去打包的，它只需要去，定义在所有子项目中所共享的一些信息，这时候它的packaging的值是pom，具体我们在后面碰到例子的时候，我们再来讲这样的一种Maven的工程，怎样去写POM文件

好，我们可以看到，如果说它packaging的值是pom的时候，它其实所有的这些阶段，都没有绑起来，除了install和deploy以外，因为在父POM文件定义的结构中间，其实它既不需要编译，也不需要做任何其他的事的，它只是绑定了它的install和deploy，分别绑定到maven-install-plugin:install和maven-deploy-plugin:deploy

对于Site生命周期来说，它的site阶段绑定的是maven-site-plugin:site的目标，site-deploy，则绑定到同样一个plugin的deploy目标上，这是Maven的内置的绑定，

所以说，我们如果在plugins中，不做任何定义的话，Maven的这些所有的阶段，就是这样绑定到对应的插件上，当然有时候我们需要说，在Maven的不同阶段，去做一些我们自己定义的，这样的一个动作，定义的功能，我们就需要在plugins中去自定义阶段和插件的绑定关系，当然我们在这个阶段，暂时先不给大家看这样的例子，我们在后面，逐步的把这种自定义的绑定的例子，讲给大家来听。

我们看一下在idea，怎样把一个工程，从github上头，把它下载下来，我们选择idea的VCS的菜单，底下的Get from Version Control，这样我们就可以从git服务器上，把我们要的工程下载下来，Version control选择Git的版本，然后在URL上头，输入我们的JavaEEPlatform工程的git地址（https://github.com/mingqcn/JavaEEPlatform.git），然后选择clone，这样的话，它就会从服务器上，把JavaEE的示例代码，全部把它下载下来。

在下载下来的文件夹里头，我们就能看到，Maven的标准的目录结构了，但是这时候idea，并不会把它认为是一个项目的工程，我们还需要在idea中间，把它import成为一个 idea的，在Maven的根目录的POM文件上头点右键，选择Add as Maven Project，这样使得idea的IDE，去读取POM文件里头的相关的定义，然后把它变成一个，在Idea的工程，当它变成Idea工程以后，你可以看到，它的相应的目录的颜色，都会发生改变，因为在Idea中间，它会认出来哪些是，生产代码的目录，哪些是资源目录，哪些是测试代码的目录。

同时在右边的TAB里头，你可以看到，它出现了一个Maven的TAB，我们点开这个Maven的TAB，你能看到，已经识别出来的工程项目，autowiredemo这里头它有三个Folder。第一个是Lifecycle，这个就是我们的生命周期这个阶段，第二个是Plugins，也就是我们在pom文件中定义的plugin，Maven自己默认的plugin，第三个是Dependencies，这个是我们在POM里面定义的所有的Denpendencies的jar包，都在Denpendencies里头。我们主要使用的是，Lifecycle和Plugins，主要是用这两个东西。

我们首先看一下Lifecycle，我们可以看到，在Idea中间的Lifecycle中，并没有把所有的阶段，都放到了Lifecycle中，我们可以看到，它放了主要是，在Clean周期中间的clean阶段，在Site生命周期中间的，site阶段，以及在Default生命周期中间的，validate，compile，test，package, verify，install和deploy，这几个阶段。，为什么会把这些阶段放上，因为这是我们在使用Maven的时候，主要的阶段，比如说我们不太会去使用，generate-sources，和process-resources，这些阶段常使用的阶段，在idea的界面中间，它都把它直接列了出来。

我们来看一下怎么样来使用，我们知道这些阶段，其实是有绑定了插件的，目标的，我们可以看到clean的，我们首先要做个打包，打包的package的阶段是在Default生命周期里头的，如果说你在界面上，可以看到，运行package的阶段的话，它前面的validate，cmpile和test，这些阶段都会要跟着运行，所以我们可以运行一下package，我们可以看一下，它是不是这样。

运行完了以后，我们可以看到这些，plugin的目标，都在一一的执行，我们首先看到的是，maven-resources-plugin:resources，的目标，这个目标我们可以回忆一下，我们绑定的是在，Default生命周期中间的，process-resources，这样的一个阶段

后面我们可以看到maven-complie-plugin:compile目标，它绑定的是我们的Default生命周期的compile阶段，这上面已经写了。

第三个我们可以看到的是这个maven-resources-plugin:testResource，它绑定的是Default生命周期的testResource这个阶段。

第4个我们看到的是maven-compile-plugin:TestCompile绑定的是Default生命周期中的，testCompile的这个阶段然后是maven-surefire-plugin:test，绑定是Default的test这个阶段。

所以当我们运行package的时候，它其实是会运行这些前面的阶段所绑定的，那些plugin的目标，然后是执行测试的结果，然后我们可以看到，jar和repackage这两个目标，执行，这两个目标都是，绑定在package的阶段里头的

我们过来看一下目录上的，我们也看到，它生成了一个红色的target，目录（不会上传Git的目录），在target目录中间，我们可以看到，我们编译出来的.class文件，我们可以看到，我们打出来的jar包，这就是我们在package阶段，最后执行起来的结果。

我们执行一下，clean这个阶段，是在Clean周期的，它的主要目的是为了清掉所有的这些构建出来的文件，我们可以看到，提示它运行的是maven-clean-plugin:clean，是在Clean生命周期的clean阶段的，绑定的插件目标，运行完这个插件目标以后，你可以看到target的目录，整个就被删掉，所以我们通过idea的，右边的Maven的标签，可以很方便地使用这个idea，由于我们的工程，是使用Maven，进行工程管理的，所以我们一般不会使用，idea里头的自身的功能，去进行程序的运行和打包，而是借助于Maven来实现，因为我们如果要修改配置的话，我们也是改的Maven的配置，它不会去改Idea里头的配置，所以这样的话，就把Idea退化，成为我们一个写代码的工具，整个工程的打包编译的过程，我们交给了Maven来做。

# 2.4 SpringBoot

Spring Boot的主要的作用是，用来去构建一个，独立的应用程序，SpringBoot的应用程序中，已经包含了Tomcat Jetty，和Undertow这样的Servlet容器，所以我们不再需要，在服务器上去装一个Tomcat，或者Jetty的Servlet容器，然后把我们的应用程序，部署到这样的容器上。因为SpringBoot中，已经包含了这种容器，可以打出一个，包含这些内嵌的，Servlet容器的这样的一个jar包，这个jar包里，只有我们的这样的一个应用程序，然后运行jar包，使得我们的容器，和我们的应用程序，把它运行起来。这样的一个过程，会使得我们的部署，变得比较简单。

SpringBoot的第二个作用，是它大量采用了starter，去简化Maven的这样一个工程，而starter其实采用了一种约定的方式，把我们常用的一些jar包，把它组合起来，使得我们可以，把一组一组的jar包，引入到我们的Maven的工程中间去。

SpringBoot的第三个特性是，它提供了一个产品级的指标，监控的功能，就是我们Actuator，通过Actuator我们可以监控，我们的应用服务器的运行状况，以及它的相关的配置信息。

我们首先来讲SpringBoot的starter，SpringBoot中间，它把一些常用的jar包，把它变成starter.

在没有starter之前，我们的工作顺序是这样的，比如说我们在Maven中，需要引入一个jar包，我们会把jar包，引入到Maven工程中，然后编译，然后发现，这个jar包还会依赖于另外一个jar包，所以我们把第二个jar包，又把它加进来，再编译，然后发现它会引用第三个jar包，所以这样的过程周而复始，直到我们把所有的jar包，引入进来.

有了starter以后，我们只需要引入一个starter，它其实是一个jar包的集合，还有它们的配置，通过引入 starter，把所有相关的jar包，以及它的配置信息，全部引进来。

starter是怎样做到这一点的呢，比如说我们最常用的starter,是spring-boot-starter-web，如果说不用starter，我们需要引入，比如说Spring Web，SpringMVC等多个jar包，才能完成我们日常的Web应用所需要的那些功能。spring-boot-starter-web里头，把这些所有必要的jar包，都已经包含在里头了，所以说我们可以把这样的一个starter认为是一个Maven的依赖组，引入了这一个组，就引入了里头所有的依赖。

在我们的课程设计中间，我们会常用到的starter，有这么一些。

首先我们会用SpringBoot的核心的starter，叫做spring-boot-starter，这一个starter里头，包含了自动配置，如何做日志，以及YAML的支持。

第二个，因为我们的应用程序，是一个Web的应用程序，所以我们会包含spring-boot-starter-web，来支持Web有关的功能，比如说RestFul API，比如说是SpringMVC等等。

第三个，我们会用到面向方面编程，所以我们会引入spring-boot-starter-aop，这一个starter里头，主要是支持面向方面编程，以及AspectJ。‘

第三，为了提速，我们会用到Redis这样的，NoSQL数据库，所以我们也会引入spring-boot-starter-data-redis，这里头包含了Redis一系列的库。

最后我们的程序，是会要做严格的测试的，所以我们会引入，spring-boot-starter-test，这一个starter，中间包含了JUnit，Mockito，等单元测试的包，来给我们做单元测试用。

我们以autowiredemo，来看在Maven中间，如何定义一个SpringBoot的工程。

首先我们看到的是，一个SpringBoot的工程，它需要继承，spring-boot-starter-parent，因为SpringBoot，有一些公用的配置，设置信息是写在这个parent里头的，所以我们一般来说，就直接继承它的spring-boot-starter-parent。（pom.xml）

第二个部分可以看到说，因为我们这个工程非常的简单，所以说它这里头只写了五个，starter，第一个starter是，spring-boot-start，这里头包含了一系列SpringBoot，所要的jar包，还有它的配置信息。

第二个是spring-boot-starter-web，其实这个部分，不是特别的需要写，因为我们本身的工程里头，其实是，没有这个部分内容的，但是因为我们要使用Actuator，所以说因为Actuator是有Web的，所以我们需要加入，spring-boot-starter-web这个部分

第三个是我们用来做测试的，这个代码中间也没有测试，其实测试的部分也可以不写

第4个我们看到的是Actuator

第5个部分，我们看到的是，我们要使用的，Actuator的监控界面，它需要引入一个spring-boot-admin-starter-client的jar包

还有是我们在这个代码中间的一些，简单的要用到一些其它的jar包，把它写到这头里面

我们的plugin就直接使用spring-boot-maven-plugin，能够使得整个工程，按照SpringBoot的要求，把它运行起来。

这是它的POM部分。

我们看一下它的源代码（.java），作为一个SpringBoot的工程，它需要有一个最基本的配置信息，它的配置信息，写到了根目录底下，我们的根目录下，写了一个AutowiredemoApplication，这里头写了一个main函数，所有的SpringBoot工程，都需要一个main函数的，main函数里头都比较简单，它直接就是调，SpringApplcation.run，把当前Class跑起来，带着命令行参数把它跑起来，这一行代码，在所有的SpringBoot工程中间，都是一样的，我们其实只是把它拷贝过来，它所有的SpringBoot工程都是一样。

我们这里要特别注意的是，类前面加了一个注解叫做@SpringBootApplication，这个注解其实是一个复合注解，它里头包含了三个注解，第一个是@Configuration，@Configuration表示说，这个类会是被Spring容器，认为可以去读配置，读取配置信息的一个类，第二个是注解是，@EnableAutoConfiguration，这个注解的作用，是告诉Spring容器，它要使用自动的配置信息，第三个注解是@ComponentScan，@ComponentScan的意思是，让Spring容器，从当前目录以下，去找，所有的Bean对象是什么，并且Bean对象是有，做了@Component注解的那些类，是我们的Bean对象，所以@SpringBootApplication，这个标签，就是我们由这三个标签复合起来的。因为SpringBoot的，要让所有的配置简单化，所以说它把它复合起来，变成了@SpringBootApplication，这样我们只要写一个标签就可以了。

最后我们看一下它的配置信息（application.yaml），它的配置信息写在了application.yaml，文件里头，这里头，跟我们SpringBoot的有关的事，我们定义了一个设备的端口，把它定义了8080端口，其他的部分，其他的两个配置，一个配置是Actuator，另一个是我们界面服务器做登记的。

# 2.5 Actuator

其实除了我们前面所提到的常见的Starter以外，SpringBoot还提供了一个特别的starter，叫做spring-boot-starter-actuator。

这个starter不是用来去完成特定的业务功能，它的**主要的功能是用来做监控的**。也就是我们基本上，在这个starter的jar包上头，是不会直接引用去写任何代码的。引入了这个starter，我们的这样的一个SpringBoot的应用，就具备了监控的功能。监控是我们这样的一个，多服务器的这样的一个结构，就是我们微服务体系中间，必不可少的一个最基本的性能。

SpringBoot，对于每一个微服务，都提供了这样的一个监控的支持，底层的支持。所以说，当我们把这样的一个starter引进来以后，我们的部署上去的，这样的一个SpringBoot的应用，就具备了这样的一个监控，健康检查，审计，以及一些控制的这样的一些功能。

我们具体的功能我们来看，我们加上了spring-boot-starter-actuator，这样一个starter以后，程序会变成什么样子。以及给大家介绍一个，可以去简单管理所有的，SpringBoot应用的，这样的一个，AdminUI这样的一个程序。

我们在autowiredemo，上面看一下，带了Actuator的SpringBoot应用，跑起来是个什么样子。所以我们在Maven中间，首先把它做一下打包。打包完成以后，我们用springboot:run，把它跑起来，跑起来以后，我们可以在浏览器中偏输入，localhost:8080/actuator，我们就能看到说这个服务器，的Actuator所返回的一些相关的信息。(ps：我这里输入localhost没反应)

当然这样的信息，其实是不太适合看的，所以我们在使用了，另外一个包，也就是spring-boot-admin-starter-client，这样的一个，这样的一个工程来搜集，SpringBoot的Actuator的信息，然后以Web的方式，把它显示出来。

工程在我们的AdminUI目录底下，所以我们打开，我们的AdminUI的目录。这也是一个Maven的工程，所以如果没有import的话，先要把它Add as Maven Project。

如果已经加过的话，我们可以看一下它的POM文件，它的POM文件非常的简单，它的POM文件里面用了，spring-boot-admin-starter-server，这样的一个jar包，然后它的Web服务器，用的是非阻塞式的，技术栈的WebFlux，所以说它不是用的spring-boot-starter-web，而是spring-boot-starter-webflux，这是引入的是非阻塞的，然后它有spring-boot-starter，以及spring-boot-starter-actuator。

我们可以看到，它的代码里头，其实没有，任何的代码，就是一个Application，然后前面加了@SpringBootApplication，然后它加了一个自己的注解，叫做@EnableAdminServer，这个注解的作用是，使得我们的Admin Server能够起来。

它的配置文件中间，只写了它的端口号，是在8081，所以我们刚才在autowiredemo，的例子中间，我们可以看到，它的配置文件中间，它写了一项spring.boot.admin.client.url: http://localhost:8081，这是因为我们两个应用，都装在一台机器上头了，所以它会向本机的，8081的端口，去登记它的信息。

好，我们把AdminUI把它跑起来，先把它编译打包（package），然后用springboot:run，把它跑起来，跑起来以后，我们在本地的8081端口，就能看到说它的管理的界面，在管理界面中间，我们能看到说，有一个应用已经登记上来了，是在运行在8080端口的，这样的一个应用，我们点进去的话，是可以看到，应用的一些相关的信息的，我们点这个wallboard应用墙菜单，这里头如果有多个应用的话，就会有多个出来，我们现在只有一个。(记得选择正确项目的springboot:run，在本机实验能正常成功)

点进去的话，我们就能看到它在，Actuator里头所能看到那些信息，但是这样的显示，就变得更加的友好，比如说我们能看到它的环境信息，比如说我们能看到，它的所有的Bean的信息，我们能看到它配置的配置信息，以及它的一些定时任务等等，另外我们还能看到，它的各个包的日志的配置，它的JVM等等这些相关的信息，这就是我们的AdminUI。

# 3.1 RESTful API

RESTful API是一种网络应用程序接口的设计风格，RESTful API用资源模型的方式，来描述业务的接口，主要的目的是让整个接口的描述，描述更加容易理解，具备自解释性，而且这样的一种描述的方式，可以让程序员或者使用者，很方便的找到某一个接口，是否已经定义过，从而极大的提升了，接口的易用性和可维护性。

RESTful API的接口最基本的特征是它是面向资源的。**资源**是构成应用系统的最基本的概念，它的定义和我们之前，在领域模型中间所说的领域的概念，是同样的一个含义。RESTful API的接口中间，用一个URL代表一种资源。

在这种资源上，是可以去做若干的操作的，而RESTful的操作，是依赖于HTTP的协议的，也就是它利用HTTP协议的五种的操作，GET，POST，PUT，DELETE和PATCH，这5种操作，来形成对于资源的相关的操作。

在RESTful接口中间，所有的信息可以用XML或者JSON来表述的，由于JSON的表述更加的简短，而且具备人的可读性，所以目前来说，RESTful的接口，基本会采用JSON的这样的一种表述的方式。

这就是我们说的，RESTful风格的最基本的三个特征，面向资源的、采用HTTP协议的、用JSON的方式来做表述的。

我们首先来看一下，RESTful的资源模型。RESTful的资源模型，可以支持单个的资源对象，资源的对象集，或者资源的集合对象，以及可以表述资源之间的从属关系。我们前面说到过RESTful API，是用URL来表示资源的，我们通过一些例子，来看一下，我们怎样用URL来表述，在应用系统中的一个资源。

比如说我们要描述在系统中的商品，我们可以用一个URL：https://demo.xmu.edu.cn，来表示这样的一个系统，在这个系统中间的商品goods，我们就用它的路径/goods来表示。所以这样的一个URL，就可以表述在系统中间的所有的商品。比如说我们的品牌，我们可以同样用https://demo.xmu.edu.cn/brands，来表述在系统中间的品牌。我们的商品的分类，也可以用同样的URL，但是路径是/categories来表示说，商品的分类。

我们一般来说，跟描述领域模型一样的，我们是用**名词**来描述资源的，所以在资源中间，我们不会把动词用进来，同样为了表述简单，因为我们描述的资源，都是多个资源，所以我们一般会用**复数形式**，来描述资源。

比如说我们会用，/categories，描述所有的一级商品分类，要描述在我们系统中间的一个分类的话，我们就用/categories/1来表示id为1的一个一级商品分类。那二级分类呢，我们可以用，/categories/1/subcategories表示说一个，id为1的一级商品分类，下面的所有的二级分类，其中的某一个二级分类，我们就可以直接用/subcategories/2，表示id为2的一个二级分类。如果说我们想描述，在二级分类下面的所有的商品，因为我们的商品是在分类底下的，我们可以用/subcategories/2/goods，来表示说，一个id唯2的二级商品，分类下面的所有的商品。同样一级分类，下面的所有的商品，我们可以/categories/1/goods，表示说，在一级分类底下的所有的商品。

我们在电子商城的系统中间，在课程设计的电子商城系统中间，一级分类底下，其实是没有商品的，我们的分类，也就是采用两级分类的方式，所以所有的商品，其实都是在二级分类底下，/categories/1/goods则表示说，在id为1的一级商品，分类底下的，所有的二级，分类里头的商品，所以说它是一个非常多的，商品的集合。同样我们如果说想要，拿到一个单独的商品，我们可以用/goods/5，拿到一个id为5的商品。

**JSON是RESTful API用来进行数据交换的格式**，也就是资源的属性值，资源的内容，是用JSON来描述的。JSON它是一种完全独立于编程语言的，用文本方式，来描述数据的这样的一种格式。JSON所具备的最大的特征，在于说，它在便于人去理解的同时，而且也方便于机器去解析或者生成，而且由于JSON的描述，是相对比较简短，所以说，它是有利于提高网络传输的效率。所以今天我们在RESTful API中间，大量使用JSON，来作为数据描述的语言。JSON的构成的值，可以由对象、数组，数字和字符串，以及我们的两个布尔值，false true还有空（null）来构成。

我们以一个对象为例，我们要描述一个人，一个用户的信息，以及他的住址，我们的用户的name，属性是John Doe，他的age是18，他的地址，则是另外一个对象，它的country是China，zip-code，邮编是1000，我们用这样的一个JSON字串，可以用很短的这样的一个长度，来描述出来一个，人也可以理解，机器也可以解析的，这样的一个信息。这是对象，对象的话，大家可以看到，它全是用花括号{}来括起来的。

第二种是，它也可以用来表示**数组**，数组的话，它直接用这个值放在[]里面，比如说我们一串整数的数组，用方括号括起来，直接把值放在里面，而不需要再放属性的，这个名称了。

这样的一种方式，我们可以看到，JSON是既简短，而且又便于被人和机器理解，这也就是为什么RESTful API，会采用JSON的方式，来描述我们的资源。

# 3.2 RESTful操作

RESTful API,是利用HTTP的，5种请求的方法，来完成相关的操作的。5种请求方法，我们前面说过，分别是HTTP的GET，POST，PUT、PATCH和DELETE。

这5种方法分别对应的获得一个，资源的信息，新建一个新的资源信息，修改资源信息的所有属性，修改资源信息的特定属性，以及删除一个资源，所以利用这5种方法，看起来，好像只能完成，资源的增删改查，我们知道对于针对资源的操作，在系统中间是很复杂的，我们怎么样用这5种操作，配合资源的定义，来完成我们在系统中间，各种各样复杂的工作，我们来看一下。我们在这一部分来说，RESTful的这种API，是怎样来实现的。

我们首先说一下，这5种操作的返回值，它的返回值，会利用HTTP的状态码。

我们知道在HTTP的状态码中，对我们来说比较有意义的，是以200开头的这些状态码，200开头的状态码，表示的是操作成功，它有200，201，202。

300开头的的状态码，其实对我们来说，没有什么太大的意义，因为300开头的表示是重定向。

400开头的表述，说是客户端的错误，也就是我们前端，往后端发请求的时候，如果请求包含了语法错误，比如说方法的名不对，它就会返回404，比如说方法名对的，但是参数不符合要求，或者参数不对，它就会返回400，所以400开头的错误，主要是用来描述客户端的错误。

500开头的错误，主要是描述服务器的错误，就是服务器在处理的时候，内部会产生错误，比如说，内部的数据库Down掉了，那就会发生500的错误。

我们利用这些HTTP的状态码，把系统的一些错误的返回值，或者正确的状况，返回给前端，但是在所有的HTTP的状态码中间，我们并不是所有的都会用的，我们会用到的状态码，主要有这么一些。我们首先说，5种方法啊都会用到的状态。

首先来说是400，因为400开头的这个状态码，表示的是客户端的错误，如果客户端的参数格式错误，就是用400来表示，401用来表示说，用户是需要进行登录的（才能访问这个URL的），403，是表示说，这一个API用户，是没有权限来访问的，409，是表示说资源是冲突的，往往是由于说资源被某个用户所，独占造成的。

500的错误，表示的是服务器端的问题，500表示的是通用错误，也就是服务器程序出错，我们一般都会返回，500通常比如数据库down掉了，等等这种原因就会返回500。503是表示说，服务器当前无法处理请求，往往是因为服务器的负载很大，它超过延时了，我们就会返回503的错误，这是所有的请求，都会返回的共用错误。

我们来看一个个的请求，我们首先来说一下GET的请求，我们前面说过，GET请求是用来查询，或者获得特定的资源的，比如说我们要获得，在系统中间的，所有的一级分类，我们就会用GET /categories，这样的一个请求，去获得在系统中间的，所有的一级商品分类。

如果说要获得一个特定的商品分类，就是我们用GET请求，去对应着它的资源的定义，GET /categories/12就是用来，获得id为12的一个一级商品分类。

如果说要获得在一个分类底下的，所有的商品，比如说我们要获得，在一级分类底下的，所有的商品，GET /categories/12/goods，通过这样的方式，就能够获得id为12的一级，分类底下的所有的商品。

大家知道，如果说在我们系统中间，商品是非常多的，直接获得一级分类，底下的所有的商品，其实会造成系统的失败的，所以说像获取这么大的数据，我们通常在后面，还需要加上它的分页，至于分页怎么定义，我们可以看一下后面的例子。

如果说要得到一个一级分类，底下的二级分类，我们可以用GET请求，去访问/categories/12/subcategories，这样的一个资源，这样就会得到，id为12的一级分类，底下的所有的二级分类。

如果说我们想，得到符合某些条件的，这样的一个商品，我们可以在/goods后面，

去加一个问号，后面跟的是过滤条件，或者叫查询条件，我们用，用一个查询条件，name等于电视机，然后我们会跟上它的排序，按照price按照价格排序，用&的符号，用&的符号来分割多个条件，来获得一个名称，为电视机的商品，价格，按照升序排序的，这样的一个东西，这就是我们说，用问号，加上查询，或者过滤关键字以后，来获得的一个资源。

同样我们比如说，要获得所有的秒杀商品，我们就像前面获得一个一级分类，底下的所有商品也是类似的，我们需要加上它的分页，第几页以及每页多少个商品，所以我们GET /hot-goods，然后用问号后面跟上两个，条件page等于2，然后&pagesize等于10，这样可以分页的方式，获得秒杀商品的第二页，以每页10个商品的方式，把它显示出来。

GET的这样的一种方式，我们可以看到，它是用来获取一个资源的，所以一个GET所获得的资源，通常是我们的最后一个名词。

比如说我们前面例子中间看到的，要获得一级分类，底下的二级分类，所以它是\categories\12\subcategories，它所返回的是我们在整个URL中间的，最后的名词，我们要获得的东西。

GET的返回值，如果说是获得对象成功的话，它的 HTTP的返回状态码，是用了200，表示说请求成功了，结果已经在这个，会放到Response里头返回。

POST的请求，是用来创建一个资源的，而且会把创建的资源，新的资源返回回来的，我们知道，当我们创建一个新的资源的，时候啊，是会得到一个，由系统生成的，唯一的id的，id是要放到返回的新的资源里头，一起带回来的。POST的方式，它是可以携带请求体的，它会是把要新建的对象的信息，以JSON的方式，放到请求体里头。

比如说我们要创建一个一级分类，我们可以用，POST /categories，这样创建的，就是一个一级分类，比如说我们要创建一个二级分类，我们可以用，POST /categories/12/subcategories，这样的一个定义其实它是创建了一个二级分类，而且会把二级分类跟id为12的一级商品分类，建立了一个从属关系。

比如说我们要创建一个，二级分类底下的商品，这个商品是之前已经创建过的，之前这个商品，或者说是没有分类的，或者说已经在某个二级分类的，当我们做这样的一个请求

POST /subcategories/19/goods/1，则表示说，要把这个商品，加到id为，19的二级商品分类，它的结果就是，如果这个商品之前，在其他的分类中间，它就会从其他的分类中间，移除，加到 id为19的二级分类底下，如果没有的话，就是直接加上来了。

关于付款，我们也可以用POST的部分来完成，我们可以用，POST /orders/1000/payment 为id为1000的订单，来完成这样的一个付款的动作。

所以我们可以看到，POST的定义是新建一个资源，它不仅仅可以用来新建一个资源，也可以用来新建一个，资源和资源的从属关系，比如说我们前面说了，把一个商品加到一个二级分类里头去。

当然有些时候，我们也会利用，POST的这样的一个请求，来完成我们所要的一个功能，比如说登录，登录这样的一个动作，其实并没有新建任何的资源，但是登录这样一个动作，需要把用户名和口令传到服务器端，而用户名和口令，需要利用，Request Body，就是请求体来传到服务端，因为POST，具备这样的一个特性，所以说我们，用POST的这样一个请求，来完成我们的登陆操作，就是POST /login，来完成一个用户登录操作。

我们可以看到这一个请求，或者说这样的一个定义，其实违背了我们在前面所说的，RESTful API的，资源的定义，因为RESTful API的资源，必须用名词来定义，但是login是一个什么，login是一个动词，因为我们没有办法，把登录这样的一个动作，描述到某一个特定的资源上头，它是登录这个系统，而这个系统本身不是一个资源的，我们只是把系统中间的概念，来建立的资源，所以我们没办法，去描述登录的，这样的一个动作的，我们只能用一个动词，在POST里头，去描述这样一个东西，所以RESTful的风格，虽然它有定义了一系列的原则，但是我们在很多时候是没有办法，完全按照RESTful风格的原则，来定义我们的接口的，我们再把这些特例的东西，需要把它单独拎出来，如果说除了这些极少数特例以外，大部分的系统的请求，都可以通过，大部分系统的功能，都可以通过资源，以及对资源的请求，来完成。

我们来看一下POST的返回码，POST的请求如果成功的话，它返回的不是200，而是201，因为201用来表示，一个新资源的成功的建立，它在数据的Response body中间，会带回新建的对象，同样这个对象是有id的，这是一个同步的调用。

如果是一个异步的调用的话，当客户端发出这个请求的时候，其实它不会即时返回新建的对象，这时候它返回的是202，202表示说，服务器已经接受了这个请求，但是它这个请求没有完成，客户端就可以继续做其他的事情，但是请求完成以后，不见得会，通知客户端，也许客户端就不需要管这个请求，比如说发个邮件，这样的一个动作，它其实就不需要管说，这个邮件是否有发成功，交给服务器端去发就好了，所以这时候它会返回202。

第三个我们要讲的是PUT的请求，PUT的请求是用来去修改，我们的资源的属性的，或者说我们修改资源对象的，PUT的修改的值，同样放在请求体里头，Request Body，

比如说我们要修改，id为12的商品一级分类，我们可以用PUT /categories/12，来完成这样的一个修改的动作。

比如说我们要修改二级分类，我们可以用POST /subcategories/13，去修改id为13的二级分类。

我们除了用PUT，来完成修改的动作以外，其实我们还能用PUT，去完成我们在系统中间的一些，功能，比如说我们的商品，是要上架和下架的，我们可以用PUT的请求，来描述，这样的一个功能，比如说上架我们可以把它描述成为，PUT /gooods/5/onshelves，这表示说把id为5的商品上架，下架我们可以用，PUT /gooods/5/offshelves，这样表示说，把id为5的商品下架。所以很多的功能，我们都可以通过，这样的一种，资源的定义的方式，通过PUT的请求来完成（便于控制权限）。

PUT请求的返回的状态码，如果说这样的一个请求是成功的，它会返回200的这样的一个值，按照HTTP的协议来说，PUT的请求是要把修改好的对象带回来的，但这样会有安全性的问题，比如说，我只是修改对象的一个属性，但带回的话，会把修改对象的所有属性，全部带回来，所以我们一般来说，PUT的请求，只会返回一个状态，而不会把修改成功的对象，给带回来。如果说是一个异步的操作的话，跟前面的POST一样，我们用202来表示说，服务器端已经接受了，这样的一个请求，但是它的这个处理没有完成。

第4个是DELETE的请求，DELETE请求，是用来删除一个资源的，比如说我们要删除一个，id为12的商品一级分类，我们可以用，DELETE /categories/12来完成，要注意这样的一个动作，不仅仅是删除了一个资源，还需要把资源底下的，把一级商品分类底下的所有的，二级商品分类要一并删除，然后还要把商品的，归属于，所有一级分类底下的，二级分类的，所有的商品，全部变成，没有分类的商品。

比如说我们要从，一级分类中间拿掉一个商品，我们可以用DELETE /subcategories/12，DELETE /subcategories/12/goods/5，这样的话，它会把id为5的商品，从一级分类里头，下面的某一个二级分类里头，给它拿掉，所以DELETE，不仅仅是可以删除一个，资源也可以去掉一个资源的，从属关系。

DELETE返回的状态码，跟前面的PUT是一样的，200表示删除成功，202表示说，这个请求是已经接受，但是尚未完成。

# 3.3 SpringMVC

SpringMVC采用了一种松散的，可插拔的组件结构，它通过用注解的方式，使得我们的一个Java的类，变成我们的控制器，控制器不需要实现任何的接口，它是一个POJO。SpringMVC是可以支持RESTful风格的，控制器的，它的整个的结构，是围绕着DispatcherServlet类的，也就是我们称之为，前端控制器为中心来展开的，DispatcherServlet，是SpringMVC的框架，已经做好Servlet对象，它会放在我们的Servlet容器中间，所有客户端发过来的请求，都是通过DispatcherServlet，分发给在Spring框架中间的，其他的控制器对象的，所以说DispatcherServlet是SpringMVC，的中枢，它负责从客户端，获取用户的请求，然后根据用户请求的URL，也就是它的RESTful资源的描述，来分派给对应的控制器来访问，同样它会把从控制器返回的结果，来决定说使用哪个视图，

把最后的界面呈现给用户，我们来看一下，SpringMVC，的这样的一个架构，是怎样处理前端的请求，以及返回回应的，第一步当客户端，发起一个请求到DispatcherServlet，也就是前端控制器的时候，前端控制器是会把请求，交给，Handler Mapping，去看应该有哪一个，Controller来负责这样的一个请求，Handler Mapping，是会根据配置信息，目前来说主要是根据注解，来在整个Spring容器中间去找说，有哪个Controller所定义的URL，是跟客户端发过来的， URL是一致的，Handler Mapping，处理器的映射器，它会把Controller返回给前端控制器，前端控制器，则会去执行Controller对应的这个方法，Controller方法，在执行完成以后，Controller方法其实会去调用，后面的服务层的方法，以及Dao层的方法，Mapper层的方法，去完成整个的逻辑，当调用完成了以后，它会返回一个ModelandView的结构，这里头既包含了所返回的数据，也包含了这个数据所需要，用来展现的是图是哪一个，所以我们称之为，ModelandView，这样的一个结构，DispatcherServlet，会把其中的View提取出来，去从它的View Resolver里头去看，应该有哪一个视图，来完成这样的一个数据的渲染，所以它会根据视图的名称，返回不同的视图的解析器，来把这个视图解释成为一个View对象，再返回给DispatcherServlet，DispatcherServlet则把模型，数据和这个视图渲染到一起，

也就是把模型数据填到视图上头，然后返回给客户端，客户端就拿到了一个有数据的，这样的一个视图，这是传统的方式，在今天我们使用RESTful的这样的，一种风格的时候，我们的服务器，是不需要向前端去返回一个已经渲染好的界面的，我们只需要把控制器成，所返回的数据，直接把它包装成为JSON的格式，通过HTTP的Response返回给，客户端就好了，所以后头的，View Resolver，就是视图的解析器，以及视图的渲染过程，在RESTful Controller，中间是没有的，DispatcherServlet找Handler Mapping，去看这样的一个URL，由哪个控制器来负责，哪个控制器的哪个方法来负责，去调对应的控制器的方法，然后把控制器方法返回来的值，变成JSON，然后都把控制器返回来的值，丢还给前端，就完成了整个过程，所以它会更加的简单，我们在SpringMVC中间，通过一系列的注解来标识说，控制器，以及控制器的方法，该处理什么样的URL，我们今天大量使用的是，RESTful控制器，就是控制器的所有的请求，都是RESTful风格的，所以我们用，@RestController，这样的一个注解，来标识说，控制器是提供RESTful风格，API的控制器，对于控制器中间的，对于这个Controller的中间的每一个方法，包括Controller本身，我们可以用，@GetMapping，@PostMapping，@PutMapping和@DeleteMapping，去标识说URL的，是什么URL，以及它到底是哪一种请求方式，当然我们还有第5种方式，就是用@RequestMapping，@RequestMapping这种方式，其实就是不指定说，到底是哪种请求，通常我们会用来定义，在定义在Controller，前面，用来表示说类的所有的方法的，共同的这样的一个路径，我们可以看到，在前面的两个部分，我们都没有提到，在HTTP协议中间的，第5种请求方法PATCH，PATCH是用来去修改，某个对象的特定属性的，但是我们今天，更加习惯，用PUT的这样的一个动作，这样的请求来完成，对于对象的某些属性的修改，原来的定义来说，PUT是对对象的属性的全部，的修改，今天我们一般在实现的时候，会有这样的一个约定，当我们用PUT请求的时候，如果对象里头的属性是空的，那就说明这个属性是不需要修改的，我们所要修改的，PUT的发过来的对象中间，属性值不为空的这样的一个属性，就是要修改的，所以这样的话，我们在定义这样的，一个对象的属性的时候，我们往往会采用对象型的，比如说我们不会用int去，定义一个整数型的，而是会用Integer这样的一个对象，去定义这样的一个属性，这样就方便说，我们可以，通过PUT这样的一个请求，来完成对于对象部分，对于资源部分属性的，这样的一个修改，我们首先来看一下，SpringMVC的这些注解，是怎样使用的，这个例子是电子商城中间，商品和商品规格的例子，我们知道在电子商城中间，一个商品是有多个规格的，而且这种规格，是可以组合式的，来选的，比如说像红米的4X，手机有颜色，有内存有机身，存储，这三个选项可以选，所以用户当选了三个选项以后，就形成了红米4X手机的，一个具体的规格，手机的价格，它的重量，等等，都跟这个规格，是有着密切的关系的，所以我们在SpringMVC上头，基于这样的一个商品和商品的规格，实现了一些简单的新增商品，查询商品之类的这些功能，通过这些功能，我们可以看到，SpringMVC的注解，是怎样来使用的，这个例子是在JavaEEPlatform的2. SpringMVC，这个目录底下，这个目录底下有一个RestfulDemo，这个例子，我们先不去看它的配置，还有它的总体的内容，我们先看一下，我们刚才说的S'p'ri'n'g'MVC的这些注解，是怎样用来去标识一个Controller，这个例子非常简单，我们只做了一个Controller在restfuldemo包，底下的controller的包里面，点下这个包以后，你大家可以看到说，它里头有一个GoodsController，所以我们点开这个类，类里头的注解写了很多，但是我们不打算一次，把这些注解全讲给大家，我们先讲就是在SpringMVC中间，@RestController，以及5个Mapping，在这个类的前面，我们看到它有一个@RestController的注解，这个注解就是告诉Spring容器说，类，是一个Controller，而且还是个RESTful的Controller，那意味着说，它送过来的参数，是可以被以JSON的方式送过来，而它的返回值，可以用JSON的方式回去，它的每一个方法是可以映射到一个URL的HTTP的操作上面，所以它这里定义了，对于整个类定义了一个根的路径，我们叫做@RequestMapping，因为在这类中间，既有GET也有PUT有POST的，所以说它在根目录底下，就没有去指定所请求的类型，用了@RequestMapping这样一个注解value定义的goods就说明，这个类底下的所有的方法，如果自己不指定，绝对路径的话，它就是会从goods开始，往后接它的相对路径，故事后面有个produces，内容是application/json;charset=UTF-8，这个是在描述说，这个方法的返回的Response它的Content类型，它的内容的类型是什么，因为我们现在全是RESTful的这样的一种风格的API，所以说它的API返回的类型，全是JSON的格式，因为结果中间是有中文的，所以我们还需要特别设定说它的charset是UTF-8，否则的话中文会出现乱码，这是produces的作用，然后我们看底下的，看底下的每一个方法，我们首先看到的第一个方法是getGoodsById这个方法，但这个方法的前面是有一个@GetMapping表示就是说这个方法是一个，是映射到一个GET请求上头，里头的值里头，大家可以看到，写的一个，这个表示什么，这个表示是说，在类的路径的后面，然后再加上一级id，表示的说明，它是一个变量，所以我们在它的方法的参数里头，我们看到了一个注解，叫做@PathVariable，这个注解的作用就是说，要从路径上面去拿到变量，变量的名称，就是后面写的id，然后把这个值赋到参数的，id里去，所以比如说\goods\5这样的一个资源，我们知道这是定位到，在系统中间的id为5的一个商品，其中的5就是我们用{}表示，id会被映射到Integer的id，上面去，这是第一个方法，第二个方法，我们看到的也是一个，@GetMapping，映射的是/goods/search，我们看一个POST的第三个方法，是新建一个新的商品，它用的是@PostMapping，@PostMapping，因为后面是，它是没有内容， 就是POST /goods，就会使得说，createGood的方法，是被映射到POST \goods上头，因为新建的商品的内容，是在Request Body里头传过来的，所以我们注意看它的参数里头有一个注解，叫做@RequestBody，这就说明说，后面的这个是一个对象，我们称之为VO对象（Value Object）就是用来，去接收，从前端送过来的数据的，因为它前端送过来的是一个商品对象的信息VO的对象，我们就在controller下有一个vo包，我们可以点开看一下，VO的对象，这个VO的对象里头，定义了说从前端发过来的一个，新建的商品，里头有些什么样的属性，我们可以看到，它里头有商品的名称，商品的描述，商品的单位以及商品的可选规格，也就是我们说的颜色，运行内存，机身存储这些，还有多少的规格，是可以选的，还有一个就是，根据这些可选的规格，它最后组合出来的话，到底有多少个不同的产品，它不同的规格的具体的产品，每一个规格的产品，我们用一个ProductVo去表示的，所以我们可以看一下，ProductVo里头，描述的是每一个产品，它是有个编号序号，序号其实就是来自于，我们前面的可选规格的组合，把它按照一定的规则组成序号，第二个是规格的描述，然后我们知道在每个规格上头，它的促销价和零售价是不一样的，另外还有它的重量不一样，所以我们写了一个这样的，ProductVo的这样一个类，所以POST的方法来说，createGood方法来说，前端是会把一个商品，商品的可选的规格，以及由可选规格所产生的所有可能的规格的组合，把它建成了一个这样的对象，它GoodsVo和ProductVo的，一对多关联的这样一个对象，然后把它传到，我们的方法里头，我们的传过来的方式，是通过，Request Body过来的，所以我们要在这个方法的参数前面，加了一个@RequestBody，我们还能看到说，它上面还有一个注解叫做，@ResponseStatus(HttpStatus.CREATED)，我们记得我们前面说过，POST的方法，如果处理成功的话，它返回码不是200，它的返还码是201，我们可以看到在前面的Get方法，中间，其实我们不用特别去标出，返回的 HTTP的，返回码是什么，因为你不标的话，它默认成功的话，都会返回200，对于POST的方法，因为我们的HTTP的返回码，要返回，201，所以我们专门写了一个注解，叫做@ResponseStatus，使得这个方法在执行完了以后，HTTP的返回码是，201，也就是HttpStatus.CREATED，这是POST的方法，我们再在看一个PUT方法，PUT方法在这儿是用来去修改对象，所以它同样的从路径里头，把用@PutMapping("")的，把id放到路径里头，然后又把路径里头的值，去放它的第一个参数里去，我们可以看到这里它的，@PathVariable里，没有去写它的值，因为这时候认为，参数的名称，和在Path里的变量的名称，是一致的，所以就会会对过去，同样我们知道，PUT要修改的对象的属性，是放在Request Body里的，所以我们同样用一个，GoodsVo来去接从前端送过来的，需要修改的这样的一个对象的属性，这是PUT的方法，最后看个DELETE方法，是用来去删除一个商品的，同样id是写在路径里头的，我们把它映射到，我们的参数里，去用@PathVariable的这样一个，变量，这样来实现了，说一个DELETE的请求，怎么映射到，我们的delGoods的方法参数里去，这就是我们在SpringMVC中间的几个主要的注解，来实现了我们，对一个URL的请求，通过我们前面说的，Handler Mapping，来看这些注解，来把它映射到

# 3.4 合法性检查

合法性检查，我们知道控制器另外一个主要的作用，是对前端发过来的数，进行合法性检查以及封装，因为我们的控制，是直接面向客户端，所以虽然前端会对输入数据的进行合法性检查，但是客户端出于安全性的必要，还是应该去做合法性检查。

通常来说，如果对于每一个输入的参数，做合法性检查，是一个繁琐而且很麻烦的事，所以我们现在一般来，会借助javax.validation来完成对于输入参数的合法检查。

javax.validation的合法性检查是JSR303的标准，它是一套对于JavaBean的参数检验的标准，它定义了很多的注解，可让我们可以在输入参数的对象上定义很多注解，我们一般来说，把所有的这些输入参数，都把定义成为VO，就是Value Object，我们可以在这些VO上头，加上这些javax.validation的注解，通过这些注解，来帮助我们，来完成合法性的检查。

javax.validation是什么呢，它有蛮多注解的，我们介绍一些比较常用的。

比如说我们要判断输入的值，是true还是false，我们可以分别用@AssertTrue，和@AssertFalse，来看注解的，属性值是true还是false。

要判空，或者说不空的话，我们可以用，@Null和@NotNull来判断注解的，属性值是空还是非空。

对于字符串来说，我们可以用这个，@NotBlank，来判断它是不是，空字串。所以，@NotBlank，它不仅仅能判断一个字串是不是空，而且也能判断这个字串里头的，所有的字符是不是空格。

对于整型的数值来说，我们可以用，@Max和@Min，去判断这个值，去验证这个值，必须要大于或者小于，对于浮点型的，它其实也有对应的一组注解（@DecimalMax和@DecimalMin），大家可以去查一下。

对于日期型的，我们可以用@Past和，@Future，来判断属性值，比服务器的当前的日期早，或者晚。

对于EMAIL，我们可以用@Email来判断说是否符合EMAIL格式的，当然EMAIL后面可以加pattern，我们可以限定这个EMAIL，必须是来自于某个机构的，可以在pattern中间去做描述。

如果说这样的一个注解，加到了VO对象上去，输入的参数在，不符合注解的要求的话，我们的Controller就会抛出，Exception，这个Exception叫做，MethodArgumentNotValidException，我们是需要来处理这个Exception的。在Spring容器中间， SpringMvC的框架中间，我们可以定义一个，RestControllerAdvice，来统一，处理异常。

我们来看一下，这一段的代码，是怎么样来写的。

我们主要用，javax.validation的这一组的注解，用来去做，输入参数的合法性检查。所以我们可以看见，这一组注解，我们是加在了，Controller的VO对象上面。

我们这个例子里面，写了两个VO对象，我们首先来看，GoodsVo对象的注解的情况。我们对于商品的名称，是因为输入的时候，商品的名称是不能为空的，所以我们加了一个，@NotBlank，注解，如果出现问题以后，它会提示，向前端提示这样的一个信息，因为这是一个字符串的对象，所以我们加了@NotBlank，以后，如果这个字符串是null，如果做字符串全是空格，会抛出一个错误，说输入的商品名称是不能为空的，这是在商品的上面。

我们看一下在规格上面，规格上面，首先来说，对于每个规格来说，规格的序号是不能为空的，所以同样，因为它也是一个String类型的字串，所以我们只需要前面加@NotBlank，就可以了，另外一个我们加了限制的，东西是它的价格，它的促销价和零售价，我们可以看到，在这里，我们是用整数来表现它的价格的，因为我们的最小单位是分，所以这样的话，它会比浮点型，在计算的速度上都会快一些，所以我们用整型来表达的价格，对于两个价格来说，我们是不允许出现负数的，如果出现负数的话，意味着你每卖一个东西，都要赔钱给别人，所以我们给它定义了一个最小的，限制@Min，对于重量，也是我们在这个上面，同样加了一个最小限制的，这样的一个最小的限制@Min，它不能是个负数的重量，这就是我们在VO对象上面，所加的一个这样的限制。

我们在给这些VO对象，定义了，javax.validation的注解以后，我们看Controller里面，大家可以看见，当我们传进来的东西，要用VO对象来接的时候，我们可以控制，要不要来做这样的一个合法性，检查，比如说在POST的方法中间，createGood，这个方法中间，因为我们要对输入的对象做，合法性检查，就是商品的那些属性，是否是合法的，所以我们在它的输入参数的goodsVo，前面加了一个注解，叫做@Validated，这样的话，就会使得我们定义在，VO里头的那些，合法性检查就起作用了，加了这样的一个@Validated，注解以后，加了这样的一个注解以后，我们就可以使得它的，Controller，在接收到这样的一个，goodsVo，对象的时候，就可以对里头的属性，来进行合法性检查了。如果说它的合法性检查的结果，是，不满足我们的要求的，这样的一个Controller，就会抛出一个，Exception, 类型是，这样的一个，叫做方法的参数，不合法的，这样的一个异常，我们是需要来处理异常的，因为我们只有处理这个异常，我们才会知道说，是哪一个方法出了问题啊，而且我们要返回一个，400的这样的一个返回码，表示说输入的参数是不正确的。

在SpringMVC的框架中间，我们怎样来去处理，这样的一个异常呢，我们可以定义一个新的类，这个类叫做处理异常的类，我们在类前面加了一个注解，245，这个注解的意思是说，它是一个Advice类，Advice是我们后面的，AOP（面向方面编程），中间要讲的一个概念，我们现在因为要用到这个部分，我们简单讲一下说，Advice这个类是用来干嘛的，Advice这个类，其实是用来说，当，一个异常出现的时候，它是怎样去处理这样的异常，就在这里来说，用来去处理说，当Controller的一个异常出现的时候，该怎样去处理这样的一个异常，所以我们可以看到，在@RestControllerAdvice的后面，我们也加了一个，annotations=RestController.class，这其实是我们限制了，Advice的适用的场合，它不是在所有的地方，都会去调用Advice的，它仅仅是有加了，@RestController注解的那些类，上头如果出现了异常，都会去调用ControllerExceptionHandler，里头所定义的方法。

我们在往下面看，我们的方法，写了一个叫做，methodArgumentNotValid的方法，这个方法的参数，就是我们要捕捉的异常，我们怎么知道，这个方法是要捕捉异常的，我们是靠在这方法，前面的注解，287，这个注解里头的value写了，说我这个方法是用来处理，291，这样的一个异常的，所以我们的参数就是在，RestController的中间，所抛出来异常。

我们看一下里头的代码，这个代码的里面，我们定义了一个，返回的对象，ReturnObject，ReturnObject是，我们在非Controller层，都会使用到的一个返回的对象，它跟我们在Controller层，要返回的对象，是非常类似的，里头是有三个部分，一个部分是errNo，一个部分是errMsg还有一个就是它，的返回值，这是它的返回值，它的返回值的部分，是用一个模板来定的，因为我们可能在这个部分，会要，把不同的对象带回去，所以我们用了一个模板的，这样的概念，用到这个地方。

在这个方法里头的代码，我们把Exception中间的，所有的Message，把它组到一起，我们可以看一下，到底会有多少个Message，把它组成一个StringBuffer，然后用分号分隔，然后把它作为errMsg，把它放到retObj中间，然后把它带回去，当然我们自己也定义了一个，字段不合法的这样的一个错误，叫做，FIELD\_NOTVALID，这样的一个错误，把它放到errNo里头，然后我们需要控制一下，因为我们的是返回这里，头是有中文的，所以我们需要定义一下，说，它的返回类型，345，

这一块部分的，我们要特别讲一下，因为原来我们，如果通过RestController的话，记得我们前面，在它的类前面，我们有定义了一个，355，这样一个定义，这样会使得，Controller中间的所有的方法，在返回的时候，在产生Response的时候，它的content-type，它的内容的类型，是这样的一个类型，但是如果说这个方法抛出了异常，它就不是由Controller的，这个方法来产生Response了，而是由我们的ControllerException，的这样的一个方法，去产生了Response，所以我们在这里，还是需要去定义一下，说它的返回类型，是带了中文的，application/json，而不是一个普通的application/json。

我们怎样来设定这个类型，我们可以看到，我们，让Spring容器，把一个HttpServletResponse注入到类里面来，所以我们注意看，它的属性里头，我们有定义了一个，@Autowired的属性，@Autowired注入进来的属性，叫做，httpServletResponse，的对象，这个对象会有Spring容器，在创建这个东西的时候，把它注入进来，所以我们虽然这里头没有写，@Component，但是我们应该知道，399，是Spring一个Bean对象，所以Spring Bean对象，Spring容器，在创建Bean对象以后，它会把当前，正在使用的，HttpServletResponse对象，给它注入到这个对象里头来，这样的话，我们就可以直接用注入进来的，HttpServletResponse，对象，去设它的，content-type，使得它是我们要的，能够输入中文的JSON的格式，然后把对象反馈，这样的话，我们最后返回给前端的就是一个，我们的ReturnObject，给它变成了JSON对象，而且它的content-type，是我们要的带UTF-8的这样，一个content-type。

还有一点的话，我们这个是需要返回400的，返回码的，而不是200的返回码，所以我们可以看到这个方法前面，我们定义了一个@ResponseStatus，这样控制这个方法，最后返回的就是一个400的，返回码，这就是我们用来处理这个Controller的，输入的参数，不合法的时候，抛出了异常处理，处理这个异常的，的Advice，其实Advice不仅仅可以用来，处理参数不合法，抛出的异常，其实只要从，RestController方法中间，抛出的所有的异常，我们可以都可以统一的写在，Advice的类中间，就是不同的异常，用一个不同的方法去写，前面，加上不同的注解，所以它是一个集中去处理异常的，这样的一种方式，这是我们的RestControllerAdvice怎样去，处理，输入的参数，不合法的这样的一个异常。

# 3.5 跨域访问

跨源的服务调用，是我们今天很普遍的一个现象，因为今天我们的服务器，已经是把前端和后端分离了，所以前端可能会用一台，独立的Web服务器，去存储前端的HTML和，JavaScript代码，或者是说小程序，而后端的话，会用另外一台服务器，去运行Servlet容器，以及在Servlet容器上头的，Spring框架，这样的话，由于前端的静态的页面，和JavaScript和后端的，提供服务的API的服务器，不是同一台服务器，所以说，就会产生跨源服务调用的问题。（跨域是指跨域名的访问）

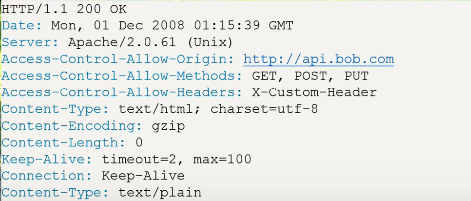
我们知道在浏览器中，间，为了安全性的起因，大家可以去查一下相关的资料，它是不允许跨源服务调用的，就是不允许说，来自于A服务器的JavaScript代码，向另外一台B服务器，去发出请求调用的，但是在我们今天的，这样的一个应用场景中间，我们怎样让跨源的服务调用，变成可能，我们是通过这样的一种方式，在HTTP协议上，通过这样一种方式来完成的。

跨源的服务的，处理方式，其实有两种，一种是简单请求，一种是非简单请求，由于我们的应用环境，是来回传的，是 application/json的格式，所以说它是归属于非简单请求的。

非简单请求，就是客户端在向浏览器，在向服务器发出真正的请求之前，它先要做一次询问，我们称之为一个询问，叫做预检请求，它是OPTION的这样的一个，操作，对于一个URL发出OPTION的操作，OPTION操作中间，就包含了这样的一些内容，它包含了，当前的域名的服务器，就是它静态的网页，HTML，和JavaScript来源的这台服务器的域名，以及说它是，想要做的什么样的操作。

我们可以看到一个这样的例子，这样的例子，发出的请求是，我的来源是api.bob.com，打算用PUT的方式，去访问api.alice.com，这样的一个服务器，api.alice.com这台服务器，接到了这样的一个预警的请求，以后它会检查，Origin就是OPTION，请求中间的Origin，和访问的方法，还有可以有的包头，是否在我的服务器设置的，允许跨源的请求范围内，如果说我的服务器允许Origin，以这样的一种Request的方式，请求的方式，包含这样的一个包头，来访问我的服务器的话，它就会给出回应，如果说我的服务器，不允许，这样的一个访问的话，它就不会给出回应。





在回应中间，我们可以看到，它有一个Access-Control-Allow-Origin，来源的服务器，api.bob.com这，台服务器，它允许的请求的方式，Access-Control-Allow-Methods，包括了GET, POST和PUT，它允许客户请求的时候，包含了一个叫做X-Custom-Header的，包头，

服务器通过了这样的一个，110，客户端就可以向服务器发出正式，的，跨源的请求了，我们称之为CORS请求，在每一个跨源的请求，CORS请求中间，它都需要包含一个，Origin的包头，来标识说这个请求的，来源是什么地方，通过检查这个头，以及它的方法，服务器来决定说，它是否能对这个请求来进行回应，这是我们从网络协议的角度来，描述，跨源服务请求，是怎样来完成的。

具体到SpringMVC中间，我们怎样来配置，说服务器端允许哪些，Origin它的来源，以及用什么样的方法，以及它可以用什么样的头，来访问这台服务器

SpringMVC是通过一个叫做，WebMvcConfigurer，的一个接口，来配置跨源请求的，所以你需要实现一个，实现了这个接口的对象，配置到Spring框架中间，告诉我们这台服务器允许，从哪个来源来的，用什么样的请求，包含了什么样的包头的，这样的一个请求，来执行跨源的请求服务。

我们具体来看一下，这个代码是怎样来写的，我们其实可以通过多种方式，来配置跨域访问，一种方式是用@CrossOrigin，可以在类的，前面，甚至在方法的前面，去设定说一个类，或者说一个方法，所允许的，来源的服务器是什么，用@CrossOrigin，注解来写。

但是很多情况下，其实我们是要对整个，服务器上的所有的请求，都设定一个统一的来源，服务器的地址的，所以这时候我们一般会使用，WebMvcConfigurer，这样的一个接口，这个接口，它其实是可以设定一个全局的，一个跨域访问的规则，所以我们在代码中间，就是采用了这样的方式，来实现的，跨域访问的配置。

在我们的代码中间，我们写了一个实现了WebMvcConfigurer，接口的，这样的一个对象，对象放到了config包底下，一般我们所有的配置的类，都放到了这个包底下，在类的前面，大家可以看到，写了一个注解，189，SpringBoot会，在整个的目录里头，去找，有配了@Configuration，这样的注解的类，会认为这个类，是用来做配置信息的，所以说注解的作用，就是告诉Spring容器，说，这个类做的是配置信息。

我们可以看到这个类，CorsConfig第一个实现了我们，204，这样的一个接口，这个接口里头大家去看的话，会有非常多的，可以实现的方法，可以用来去配置我们SpringMVC，的，各方面的，这样的一个东西，但是我们在这个例子中间，我们仅仅是用了它的，addCorsMapping，这样的一个方法，就实现了这样的一个方法，去设定它的跨域访问的，这样的一个配置。

我们可以看到这个方法，是有一个参数的，叫做CorsRegistry，其实我们就是把，跨域的设置，通过这个参数，把它设到我们的系统中间去，在这个方法中间，我们可以看到，231，它的addMapping的话，实际上它是可以用来去设定说，我这样的一个跨域访问，是对哪些URL是有效的，我们现在用的是\*\*的，对所有的URL都有效，第二个是allowedOrigins，这个其实是设定说，从哪些来源的过来，我们现在为了简单，也给了一个\*，那是所有的来源都可以，但我们知道，其实真实的系统这样是不安全的，其实是应该去设定说，它具体的IP，或者具体的域名，第三个是allowedHeaders，就是说，可以加什么样的头，我们现在同样也是全部放开的，第4条就是allowed，我们限定了GET, POST，DELETE, PUT，还有OPTIONS和HEAD的这几种，操作，是可以通过跨源来请求的，最后一个maxAge，是有效的时间，通过，重写了这样的一个方法，我们就可以对，因为我们现在是对所有的路径都，有效的，实际上，对我们系统中间，所有的跨源访问，都把它设定成，这样一个全局的系统配置

# 4.1 测试方法

测试是我们开发过程中间，一个非常频繁的过程，非常频繁的事情，测试是我们开发过程中间，必须要做的一件事情，我们可以说，开发代码的30%，甚至50%的时间，是用来去写测试代码的，所以我们在开发第一个程序的时候，在做SpringMVC的开发的时候，我们就要跟大家讲，如何来做测试。

我们通常来说，以测试的方式，来驱动，整个代码的开发过程，有些极端的说法来说，是应该先写测试，再来实现我们的生产代码，但是我一般的习惯来说，是我会先写一个，Bean对象，或者一个我们的生产代码，在里头去增加一些方法，每写完一些方法，就去写这些方法的测试，验证这些方法能够通过测试，然后再进行下一个周期。

所以这样的一个时间，通常来说控制在一天之内，也就是你一天之内，所有写好的生产代码，也要把它们的测试代码写完，然后把测试完成，才完成了这一天的工作。

我们主要从两个角度来讲，说这个测试怎么做，我们首先来讲说测试的方法，大家都知道，测试的方法主要有两种，一种是白盒测试啊，一种是黑盒测试，白盒测试主要是测代码的，内部结构和处理过程，它其实并不是测的软件的功能，它主要是测的是，我们的测试，对于所有代码的覆盖面，也就是我们的测试，应该把所有的代码都有测试到，那当然这并不意味着说，所有代码都测试到了，你的功能，就是全部都测试完了，因为有可能你的功能没用代码写，这是白盒测试。我们在这个学期的过程中间，我们会建议大家，会推荐大家使用jacoco，来看测试，对于大家代码的覆盖面。

而另外一个测试则是黑盒测试，黑盒测试则是根据软件的功能，需求，对软件来进行测试的，所以说它并不是去看你这个测试，对于代码的覆盖面，而是通过各种输入和输出，去发现，这个代码的功能，是否是跟我们想象的功能，约定的功能是一致的，从而来找到代码的缺陷，我们这学期的期末测试，则是以一种黑盒测试的方式，来完成的，我们的公开测试用例，和不公开测试用例，其实主要是测大家的，黑盒测试测大家的功能，是否满足我们在需求里头，所描述的功能，大家的自测的部分，就是你们自己写的，测试的部分，主要是实现的是白盒测试，你们每一个生产代码，一定都需要用测试代码，把它覆盖住啊，所以白盒测试是黑盒测试的基础，如果没有白盒测试，没有测试那些代码，黑盒测试时那些代码，很有可能就会出错，但是白盒测试是不能替代，黑盒测试两种测试，都是需要，用不同的测试方法，来完成的，这是测试的方法。（白盒测试走遍代码的每个分支，黑盒测试测功能是否正确实现）

第二个我们要来讲的是测试的层次，我们的测试的层次，其实主要分为三种，一种是单元测试，就是每一个部分的单独测试，另外一种叫做切片测试，就是把一个部分的系统，把它跟其他的这种隔离开，来进行切片测试，第三种测试则是集成测试，我们的学期主要在期末检查大家，的时候，做的是集成测试，比如说我们会把整个系统，作为一个整体来，对大家的系统来进行测试，自己，每个小组在自己做测试的时候，可能是单元测试，也可能是集成测试，甚至有些人会去做切片测试，所以说，三种测试方式，你们自己都会用到，但是我们在期末检查的时候，是来做的是集成测试，所以集成测试的话，它是基于黑盒的角度来测试的，所以它不可能，也许不会测到你的每一行代码，但是它主要是测试你的功能，代码的覆盖面的话，主要依靠大家自己做的单元测试，切片测试和一部分集成测试，来完成这个代码的，覆盖面的这样的一个，白盒测试

（网上关于切片测试的资料很少，切片测试和单元测试有什么区别呢）

# 4.2 单元测试

测试这一部分，我们主要结合代码，来给大家讲，如何进行单元测试，集成测试，以及性能测试。单元测试的部分来说，是测我们整个代码的最小的部分，所以我们最基本的模型对象，就是我们的POJO，就各种Vo各种，在我们里头的业务对象，如果里头有方法的话，是需要进行单元测试的。现在我们这个例子中间，我们的GoodsVo，和ProductVo里头是有写了，一个，由Vo对象来生成这个业务对象，就是生成Goods对象，和Products对象的方法，所以我们在这个部分来看一下，单元测试是如何在，这些模型对象，和VO对象上头来完成的。

第二个我们会看到的是，我们会看到的是切片测试，因为我们现在这个例子，中间的Service层，完全是一个假的Service层，因为它，并没有去访问数据库，所以我们这个代码中间，其实有意义的是Controller层，因为我们可以看到，controler层完成了restfulapi，这样的一个实现，以及说它能够对输入的数据，来进行合法性检查，所以我们重点是来去测试，Controller层的RESTful API，以及它输入参数的合法性，这样的话，我们用切片测试的方法，可以把我们的Controller API，和我们没有意义的Service层，把它切开，也就是我们只测，Controller层的代码，所以这里我们会用到，Mockito的MockBean，SpringBoot的环境下头，去把Service层，做成一个模拟的，Spring的Bean对象，然后来实现对于Controller层的测试.

第三个部分是集成测试，也就是我们会把整个系统，作为一个整体，从它的Controller层作为入口，来测试它的各个部分功能，虽然我们在这个例子中间，其实这样的检测是没有意义的，但是我们可以看到这样的一个，集成测试的方法，就是我们通过这个例子，去给大家展现一个，把整个系统作为一个，整体，去做集成测试的方法。

第4个部分，我们会选择其中的某些特定的，API，比如说我们创建商品的API，来做一下性能测试，因为我们知道，在创建商品的API上头，是有做了输入参数的，合法性检查的，所以我们通过这样的一个性能，测试来看，合法性检查，通过Exception的方式，来做合法性检查，会有什么样的问题。

好，我们来借助代码，来逐个的讲，这些测试是怎样完成的，我们的测试代码，都在RestFulDemo的，test目录底下，一般来说我们建代码的时候，我们的测试代码的目录结构，和我们的生产代码的结构，是保持一致的，比如说我们要测GoodsVo，GoodsVo是，在生产代码中间的，controller的包底下的，vo的包里头，我们的测试代码，也同样建了一个同样的包，而且我们的测试类的命名，跟生产代码的命名，是有对应关系的，这里要注意一下，我们的所有的测试代码的命名，需要以Test的结尾，如果说你不写Test的结尾的，话，surefire-plugin其实是，默认不会去找其他类名的类，它只会去运行Test结尾的类。

所以我们的类名叫做GoodsVoTest，说明我们是去测那个GoodsVo的，GoodsVo里头，我们知道它有一个方法，我们需要测试，叫做createGoods，所以在这个测试类中间，我们同样也写了这样的一个方法，createGoodsTest（方法前用@Test标注），这里头就是用来去测，GoodsVo对象中的，createGoods，这个方法。

我们在这里，用了一个工厂的设计模式，因为我们在测试对象之前，首先要把一个VO对象，给创建出来，所以我们用了工厂的设计模式，去创建了一个VO对象，至于这个代码是怎么写的，大家可以具体去看GoodsFactory类，这里要特别说一下，我们可以看到GoodsFactory这个类，因为它是为了测试，而来产生VO对象的，所以我们把这个类放到test的util包，底下，但是在生产代码中间，其实是没有这个类的，所以最后在打包的过程中间，大家可以看到，最后打出的jar包里头，是没有这些测试代码的，会没有，我们在这个测试代码中间，放的GoodsFactory，当然也没有这个XXTest类，GoodsFactory主要作用，就是用来去创建一个Vo

创出了，Vo以后，我们就去调Vo对象的，createGoods出来的Goods对象，我们需要知道说，这个Goods对象，跟我们预期的值，是不是一致的，所以我们这里使用了，在JUnit5里头，中间的，一个方法叫做assertEquals，这个包里有assertEquals, assertTrue等等，一系列的方法，主要我们这里用的是assertEquals， assertEquals的话，我们会把我们对于Goods中间的，name，和我们觉得的，它的预期值，assertEquals去做对比，所以我们可以看到，这里比较了name比较了brief，比较了unit，比较了categoryId，比较了brandId，当然你如果想去比较，specList里头的每一个东西，也是可以的，但是我们这里，做了一下节省，所以我们只比较了，它的size应该是三个，这样的一个单元测试。

我们可以看到它的，其实是不依赖于我们框架的，它因为我们测试的对象，GoodsVo，它是一个POJO对象（spring是bean对象），它并不是由Spring的容器，去管理的对象，所以这样的一个测试，就是完全跟Spring容器没关的，你可以不依赖Spring的容器，把测试把它跑完，我们把这样的一种测试，就是纯的，单元测试，基于JUnit的单元测试，当然我们也可以在Spring的框架里，头，去做单元测试，比如说我们要，单独测试它的一个dao对象，或者单独测试它的一个service对象，也称为单元测试，但是那样单元测试，其实就不是很，不是很纯净了，因为它其实是依赖于Spring的环境，我们看到GoodsVo的，这样的一个单元测试，就是最单纯的这样的一个单元测试

# 4.3 切片测试与集成测试

切片测试和集成测试，在Spring工程中间，切片测试和集成测试，其实用了相同的技术，它们都会依赖于Spring的框架，以及Spring的环境，来进行测试。

但是切片测试和集成测试的差别，在于说，切片测试可以，在Spring的环境中间，把某一部分给切开，切的时候既可以横向切，也可以纵向切。我们在这个例子里面，给大家看到的是横向切，也就是把看Controller层和Service层切开，当然我们也可以纵向切，把相互依赖的，比如说Service的对象给它切开，来单独进行测试。

我们来看一下，切片测试和集成测试是怎样做的，下面我们来看一下切片测试，我们前面说过，在我们这个例子中间，我们想把它的controller代码，跟它的service代码分开，来单独做切片测试。所以因为我们这里只有一个controller，GoodsController，所以我们做了一个叫做，GoodsControllerTest类，这个类就放到了controller，这个包底下。

我们来看一下，这个类里头，前面有两个注解，第一个注解是，@SpringBootTest，这个注解的主要的目的是要去，说我们当前的类是一个测试类，而且一个SpringBoot的测试类，所以它后面有一个，classes = DemoApplication.class，因为跟之前我们讲的Model对象，或者VO对象的测试不一样，我们要测试Controller的话，它需要依赖于这个，Spring容器的环境，Spring容器的环境在哪呢，我们的后面这个定义，其实就是说，我们要用，DemoApplication那个文件中间，所描述的是对应的环境，这个文件其实就是我们生产的，环境的，应用的环境是一致的，因为我们的根是定义在，DemoApplication里的，所以第一句话，所描述的内容只有两个，第一个是我们当前的类是一个，测试类，第二测试类所用的环境，是跟我们的生产的环境，是完全一样的。

第二个注解写的是，关于它的MVC的部分，因为我们测试的是它的Controller，如果说严格去测试Controller的话，是应该把Servlet容器起来，向Servlet容器，去发HTTP的请求，然后去看它的HTTP的返回，是不是，我们想预期的结果，但是这样测试很慢，因为这个过程，虽然你在单机上来执行，但是它还是通过网络的，端口的这样的一个，过程，而且还要依赖于Servlet容器，所以我们在测控制器的时候，我们往往采用了一种模拟的方式，就是它并不真正的去启动Servlet容器，而是模拟的这样一个测试过程，使得我们通过模拟的过程，去发HTTP的request，其实也没发，就是模拟了一个HTTP的，request/reponse的过程，所以是完成这样的一个测试，我们称这样的一个东西，叫做MockMVC。我们在下面，可以看到，MockMVC是被注入进来，MockMVC是有很多的配置，我们第二个注解，AutoConfigureMockMVC其实，就是，去自动配置，模拟MVC的相关的配置，我们现在没有给，它的值，其实它可以，在后面配一些值的，这是我们在测试类前面的两个，注解。

测试类里面我们可以看到，注进了两个对象，一个对象是MockMVC这个，就是我们用来去模拟， MVC的这样的一个控制器，的，第二个是MockBean，因为我们的goodsService，需要跟Controller，给它切割开，因为我们GoodsController，129，所以我们用一个@MockBean，去标记了一个GoodsService对象，加了这两个注解以后，Spring容器，会把这两个对象注射进来，所以通过这一点来说，我们以也能够看出来说，GoodsControllerTest，其实是一个Spring的Bean对象，Spring的容器注进来，这两个对象的话，其中我们的goodsService对象，其实并不是去注射的，是，我们真正的，生产代码中间的goodsService对象，而是注进来一个模拟的对象，148，相同的行为。

那它怎么样来完成GoodsService，所需要实现的那些方法，我们在后面例子，你可以看到，它可以提供了一个given的方式，来模拟，所有的方法的返回的值，所以我们看第一个，第一个测试方法，同样这个方法前面，要加一个@Test，这个@Test的注解，跟我们前面在JUnit里的注解，是一个意思，就表示说，方法是一个测试的方法，这个方法里头，我们可以看到，第一句话就是，我们对于注进来的模拟的，goodsService，我们要定义它的findById的方法的，行为是什么，因为我们知道，我们目前要测controller的，176，它是有调了GoodsService的，findById这个方法的，我们在生产代码里已经看到，我们通过given，这样的一个定义了，说findById，如果它的参数值输进来是1的话，willReturn就是说，它会返回一个什么样的值呢，它会返回的，就是，我们从GoodsFactory，中间，去创建出来的一个id，为1的对象，大家可以去看GoodsFactory，相关的方法，后面的部分就是，有了这样的一个前提，我们就可以通过MockMVC，去发出这样的一个get的请求，这样的一个get请求，发出来以后，我们后面可以看到，perform(get( ... ))这样的，就是发出一个get请求，andExpect，所描述的就是我们期望说，这样的一个请求的返回值，它的status().OK就是200，然后我们也期望说，content-type，是application/json;charset=UTF-8，然后我们要把它真正的返回值拿，过来，所以我们andReturn().getResponse().getContentAsString()，作为一个字符串拿，过来，然后我们写了一个，我们期望的字符串，我们把期望的字符串，因为我们知道，拿回来的，是一个JSON的字串，所以我们把期望的值，也用JSON的字串写起来。

然后我们再用了JSONAssert，这也是一个包我们引入的，叫做JSONAssert，的这样的一个包，里头的方法，这个方法的主要作用，就是去比较两个JSON字串的，是不是一样的，第一个参数，是我们期望的JSON字串，第二个是它返回的JSON字串，第三个参数，其实是说做严格比较，还是不严格，比较，true的话是严格比较，而且两个字串的东西，是完全一样的，不严格比较，我们可以允许说，responseString里头的值，多于expectedResponse的值，就是expectedResponse的JSON的项可以少一些，但是responseString的项多一些，那么我们只要responseString里头返回的，expectedResponse中间有的那些值，在responseString中也有，这样的一个结果就是对的，所以如果第三个数是false，的话，就是这样的一个定义。

我们看完了第一个方法，后面的方法其实类似，第二个方法，同样，因为这个方法中间调了，goodsService的，searchByName，方法，所以我们去定义了，这个参数，是墨迹的话，我们就会给它一个，这样的返回值创建了，带了墨迹的Goods.

创建对象的部分来说，我们可以看到，我们的given，会去描述了goodsService，的createGoods，这样的一个方法，在这个地方，因为在createGoods中间，它传进去的是，一个GoodsVo的对象，但是我们在外头建的就是，我们在测试代码中心，建的GoodsVo的对象，和GoodService所收到的GoodsVo对象，不见得是同一个对象，所以我们这里用了一个any()，但其实用了一个any()的话，就是只要去调，createGoods，它返回的就是我们看到Goods对象，就是return，我们建出来的Goods对象，这是createGoods。

modiGoodTest和delGoodsTest，比较简单，因为我们其实就是，直接让它返回来true这样，把我们的GoodsService的对象，和我们的Controller的对象，给它切开了，使得我们的测试代码，就只会测Controller的对象。

最后我们看一下，集成测试代码，集成测试代码，在GoodsControllerIntegrationTest里头，集成测试的代码，跟我们的切片测试代码是一样的，只是说，我们没有去模拟一个goodsService，所以大家可以看到，注入对象中间，我们没有去注入，goodsService对象，在所有的方法之前，我们也没有去定义说，goodsService对象，将会返回什么样的值，所以我们就直接去通过，MockMVC去调它对应的Controller，的方法，然后去看它的值，是不是我们想要的值，所以这就是测的是一个整体，我们把它称之为集成测试。

# 4.4 测试的例子

我们来看一下我们的例子，整体跑起来是什么样，这个例子和我们前面的，autowiredemo的例子相比，它要复杂的很多，它不仅仅包括了，我们在前面所讲的，是SpringMVC的内容，但更重要的是，它包含了

各种测试方法，就是我们前面说的，单元测试，切片测试和集成测试

而且它还包含了把项目文档，部署到我们的文档服务器上去的过程

好，我们来结合代码来看一下，这个例子，是怎样来实现，我们所说的这些特性的。

我们来看一下，RestFulDemo的POM文件，这个POM文件比我们之前，看到的autowiredemo，的POM文件要复杂的很多，因为这里头，我们可以看到，这个过程中间，我们有加入了，合法性检查，以及测试的很多的内容，我们还需要在测试完成了以后，生成各种各样的测试报告，然后把测试的结果，包括我们工程的整个的结果，发布到我们的Web服务器上，我们的项目工程的服务器上。

所以POM模型中间，跟autowiredemo相比，我们可以看到，首先它增加了一个Validation的Starter， 这个Validation的Starter，就是我们，在javax.validation的那，一组注解，要用的话，它是需要用这个Starter。

第二个是，我们可以看到，我们引入了一个lombok的这样的一个Dependency。因为我们，这里头有Vo对象，也有Model对象，在Vo对象和Model对象中间，我们可以看到，我们并没有去写set/get方法，包括toString的方法，而是引入了一系列的lombok的注解，让它去自动生成这些代码，这就是我们前面说的，在Maven的生命周期的，generate-source的阶段，它会调用Project，lombok，去生成，我们所需要的get/set的方法，包括toString的方法，还有equals的方法，所以这是我们引入的Project，lombok，Maven的依赖，在Idea中间，因为我们在代码中间，是没有那些set/get，包括toString的方法的，所以说，在Idea中间，会发现它是有编译的红线的，警告的，但是因为我们的编译，和打包的过程，是依赖于Maven的，所以Maven有这样的一个依赖，在Maven编译打包的时候，就不会出错，为了把这个Idea中间，这个讨厌的红线，给它去掉，可以在Idea中去装一个，lombok plugin装上去以后，你就发现它这个都没有了，这是关于的lombok。

然后因为我们要把结果产生，结果是产生JSON丢回去，这里头我们会用到，fastxml.jackson，其中我们这里引入的这几个，jackson-datatype-jsr310，以及jackson-datatype-jdk8，这是因为在库里头，在fastxml库里头，对于日期型的处理是有问题的，原来的标准库里头，是有问题的，所以它后面做了一个补丁，我们需要去额外引入，fastxml的，jsr310和jdk8，这样使它的日期形式，能够写到这个JSON里头去的。

再往后头是两个，跟我们的，API的文档Swagger，API文档是有关的，我们在代码中间可以看到，我们写了大量的，swagger的注解，这些注解，是用来生成swagger ，API的，所以我们需要引入，这两个Dependency，去生成它的API。

最后两个Dependency，是跟我们的日志有关的，我们的工程中间，引入了日志的机制，在合法性检查出错的时候，会写一个日志，我们的日志不是用的，默认的日志，我们用的是logback，logback日志，是目前各种日志中间最快的一种，所以我们需要在Dependency中心，引入了，logback，然后在resources目录底下，我们可以看见，我们有一个XML叫做，logback.xml这个就是logback日志的配置，文件，配置文件的详细的内容，我们就不去讲，我们配置文件，其实是有配了两项，一项是在我们看到在控制台输出，就是STDOUT，标准输出上，所看到的一些日志，另外我们也可以把日志，写在文件里头去，因为在部署到服务器上之后，你不可能时时盯着那个服务器，去看它的输出的日志，所以我们是需要把日志，写到文件里的，特别在服务器上调试的时候，以及在之后，如果说运行的时候会出错的话，都需要去看它在服务器上生成，一个文件，所以大家可以自己去研究一下，到网上去查一下相关的资料，去了解一下，logback.xml的配置是怎么写的，当然我们也可以在里头去控制，说，它的输出的级别，正常情况下，我们可能输出的级别，在INFO甚至更高的级别，比如说我们可以控制到，WARN输出，或者甚至到ERROR输出，那样的话，级别越高，它输出的内容就越少，如果说是在出错的情况下，头我们可能把这个级别，降低到INFO，因为你的级别降低，降的越低，你的输出的日志就越多，输出的日志，其实对整个系统的性能，是有影响的，所以在正常情况下，我们一般会把，日志的输出级别，调到一个比较高的这个档次，使得它输出的日志比较少，这是我们在这个工程中间，所引入的一些新的依赖包，在之后的工程中间，其实这一些的依赖包，是我们的标配，都会引进进来的，所以我们在第一次，加入了这些东西以后，先做了一个简要的介绍。

看一下在build的部分，build的部分，我们加入了一个新的东西，大家可以看到，加入了一个叫做jacoco的，东西，jacoco的，plugin的主要作用，就是用来去产生，测试的时候，代码覆盖率的报告，所以我们可以看到，我们引入了这个jacoco以后，我们后面定义了一个executions，executions是说我们引入了，这个plugin，它要在我们的生命周期的，哪一个阶段去运行，所以我们可以看到，我们定义了第一个execution，这是一定要去运行的，这是它在运行之前，是要去做一下准备的，第二个我们可以看到，我们定义了一个post-unit-test的，这个有定义来说，它运行的阶段是什么，在test阶段，所以说，当我们的Default的test阶段，在运行的时候，除了它默认的test以外，就是surefire:test以外，它还需要运行什么，运行，我们在jacoco中间，所定义的report目标，所以说，它在阶段运行完了以后，在surefire的的测试运行完了以后，后面有定义的时候，report产生的东西在哪儿，它把所有的数据，它放到jacoco.exec的，文件里头，当然这个数据我们没办法去看，但是我们要看的，实际上是，它所产生的HTML的报告，我们把所有的报告，放到target/site/jacoco-ut，这个目录底下，当这个测试完成了以后，我们就可以看到，在target/site/jacoco-ut里头，有我们测试覆盖率的，HTML的报告，之后，我们要把site目录里的所有，内容，给发布到我们的，项目文档的服务器上去，就是我们的site服务器上去，这是第二个。

第三个我们可以看到，我们加入了一个，site的plugin在build里头，是因为我们需要去用site的plugin，去把文档做发布，这里我们也定义了它的，的字符集，就是它的语言类型，所以我们是中文UTF-8的，这样一个，格式，这里有依赖了两个东西，因为我们要把工程，依然发布到一个，Nginx的WebDAV的方式，提供出来的一个文件，共享的服务上面，所以我们这里用到了，wagon-webdav-jackrabbit，这样的一个包，以及包所依赖的，slf4j，这样的一个log的日志输出，因为我们在工程中心，用的日志是logback，我们前面说了，但是它在这儿用的是slf4j，所以我们要把slf4j，引进来，在这个plugin中用，不是在我们所有的工程中用，这是我们在build的中间，我们可以看到，因为我们要，除了是正常的build以外，我们要去做代码的覆盖率测试，以及要把我们最后的site，的内容，把它定义成为，说我们生成报告以后，我们要把它发布到，我们的Nginx的WebDAV，的服务上去，所以我们加入了两个plugin，的定义。

最后我们看到的是，我们新加进来的一个reporting，的定义，reporting的定义是什么，我们在site中间，我们其实要去定义说一些更细的，报告的生成的内容，除了说Maven site，自己所提供的内容以外，我们可以看到，我们去定义了它的，project-info-report-plugin，的版本，因为这个版本其实你不定义也可以，因为这个plugin其实是有，默认在site的里头的，但是它会出Warning，所以我们定义了一下它的版本号，我们定义了说，我们要生成javadoc，javadoc是我们写的代码，前面的javadoc，的注释，注释了，每一个类和每一个方法的，定义是什么，我们用了，maven-javadoc-plugin，生成了javadoc的文档，javadoc的文档，默认是在site里头，它在site的apidoc，的目录底下，同样这个文档被生成以后，我们也是要发布到，项目工程里头去的。

这里我们定义了，说，它的输出是用的，project.reporting.outputEncoding，这个定义，我们写在了前面的属性里头，所以在前面的属性里头我们有，定义了，说project.reporting.outputEncoding为UTF-8，因为我们的注释中间是有中文的，所以要加这个定义，否则在输出时候是会报错的，会报错的。

最后我们要把我们的单元测试，内容，是依靠Maven内置的，surefire-plugin，虽然我们在，在build那部分，没有去定义它，但是默认是它，所以是不用去定义的，但是产生的是一系列的，txt的文件和XML文件，在网上是不太好看的，因为我们最终要把所有的测试，结果，发布到我们的WebDAV，的服务器上去，所以我们需要引入一个新的，plugin叫做，maven-surefire-report-plugin，这个plugin的作用，它是可以把前面maven-surefire-plugin，所生成的数据的文件，把它产生成一个，HTML的，这样格式的东西，然后把它放到，site的目录底下去，所以我们可以看到，我们定义了，这个plugin，然后定义了说它的运行的目标，运行的方式，它其实是可以带测试的，但是我们测试用了前面那个的，所以我们让这个只是，由数据去产生报告，这是report-only，这是加入了一系列，的reporting。

reporting是一个新的部分，reporting的定义是仅在，Site的生命周期里，头才会去使用，site的阶段里头，才会去使用的，其实我们在reporting中间的，所有的plugin，都可以放到前面的build，当你把它放到，前面的build的时候，你会发现，在我们的Maven的tab中，就会发现多了一个，plugin，就多了一项，就是我们放进去这一项，所以你如果想知道说，这个东西在site中间，它会运行，成一个什么样的结果，你其实是可以先放到，build中间去，去单独运行一下，因为你放到build里头，你其实就可以单独去运行，但是它运行OK了以后，你再把它放到reporting来，这样在做site，那个阶段的时候，它就会运行，我们在reporting中间，所定义的这些plugin，去称之为生成，我们所需要的，那些项目文档。

最后我们新加的部分，是distributionManagment，这个里头定义的是什么，就定义的是我们要发布的，项目文档的服务器，我们这里头也定义了一个，site，就是一个服务器，它的id siteserver，以及它的url， url其实我们是，标明了说协议用的DAV，的协议，在HTTP的基础上的DAV的，协议，以及它的IP地址，以及它要发布去的路径是什么，特别注意一下，site的定义，中间的id是要跟我们，最后，在Maven中间，因为我们最后要靠，Maven把这个东西，发布到服务器上去，所以这个服务器是有用户名和，密码的，我们在Maven的配置，中间就是Maven的，settings.xml去定义了，说这台服务器的用户名和密码是，什么，所以要跟在，Maven的settings.xml中间的，声明的id是一样的，这样的话它才知道，发布的时候，用Mavenl配置中间的用户名和，密码，发布到我们这里，所定义的，url的服务器上去，这是我们的POM文件。

有了POM文件以后，我们把工程去跑一下，跑的时候，我们其实是要分成几个阶段来跑，的，第一个是你可以先clean一下，就会把我们之前的，target里头东西全部删掉，所以先做一下clean，我们之前的实验，在讲Maven的那一节时候，已经看到了，这样的话，它只能目录就被清空了。第二个部分，我们其实是不需要打包的，因为我们这个阶段，只是要做一下测试，所以我们跑一下测试的，阶段，跑测试这个阶段的时候，大家可以注意看一下，你就可以看到说，我们在前面所定义的，那些单元测试，集成测试，以及切片测试都会在去跑，因为我们定义了三个，我们定了看多个测试类不止三个，分别是在controller底下，Vo的最简单的，JUnit的单元测试，以及说对于它的Service的测试，代码，以及说对于它的Controller的，集成测试和切片测试。

单元测试的结果可以看到，它前面没有启动，一个SpringBoot的标志，说明说它不依赖于Spring的容器，来完成测试，而切片测试的集成测试，大家可以看到，它启动了一个，Spring的容器，SpringBoot，所以你能看到，前面有SpringBoot的标志，然后才开始这个测试，所以这是两种测试的差别。

我们这个测试结果，最后它是会生成surefire的，为report文件，我们最后来运行 site这个，阶段，site阶段，我们会生成所有的项目文档，把它生成在target的目录下，如果是第一次运行的话，就会比较慢，因为它会要从服务器上，下载很多的包下来，这会生成说，我们的整个的项目的文档，我们在前面测试的时候，产生的结果，生成了一个HTML的文件，放到了site的目录底下，叫做surefire-report.html，这样一个文件，还有说我们生成了，jacoco的文件，jacoco文件在之前已经生成了，因为我们定义在test的阶段，所以test的阶段完成以后，我们去site目录，这样去看，jacoco-ut的，目录就已经产生了，这时候主要是产生的项目的，项目文档，还有我们根据surefire的单元测试，所产生的单元测试的文档。

最后我们把这个例子，用是spring boot:run，把它跑起来，跑起来以后，我们在localhost:8080/swagger-ui.html，打开我们生成，的，Swagger的API的文档，这样我们就能看到说，用swagger的注解所产生的，API的接口文档

# 4.5 性能测试

性能测试，性能测试这一部分，我们主要讲怎样用JMeter，来对我们的系统，来做性能测试，在这一个部分，我们先简要介绍一下，什么是JMeter。

JMeter是一个基于Java的，开源的性能测试的工具，它的功能非常强大，但是我们在这一部分，只是对JMeter做一个初步的认识。JMeter里头，分为几个主要的概念。

第一个是测试计划（Test Plan），在测试计划底下，它可以去建Thread Group，所谓Thread Group，就是我们可以用一组线程，来测我们的一个或者说几个URL，在Thread Group底下，它可以为所有的HTTP的请求，去建一些默认的设置，包括HTTP请求的IP，它的端口，协议类型等等，以及，HTTP头的，默认设置，还有它的Cookie的默认设置等等。在这些默认设置的基础上，在Thread Group底下，我们可以去建，多个HTTP Request通常来说，我们在一个Thread Group底下，所建的，HTTP Request，只是针对一个URL的，就是针对某一个特定的，URL的测试。

我们今天这个例子中间，我们可以看到，我们在Thread Group底下，我们针对\goods的，POST这样的一个操作，我们建了两个HTTP，Request，一个是正常的HTTP Request，一个是不正常的HTTP Request。当然，如果严格去做的话，其实针对正常的HTTP Request要建，一个自己的Thread Group，我们在对不正常的，也要去见另外一个Thread Group，这样的话，才可以对比说，系统在正常的请求，和不正常的请求的的状况下头，这个服务器的反应是什么样子。

建好了HTTP Request以后，我们需要对HTTP的返回值，来做一些判断，所以在HTTP Request，底下我们会建Goods，我们今天所要建的Goods，主要是两种，一种是Response Goods，就是，响应的Goods，我们主要去判断，它的Response Code，就是它的状态码，另外一种是JSON Goods，因为我们是RESTful API，所以我们要去判断，它返回的JSON的值，是不是我们所要的值，这是针对一个HTTP，Request的设置，以及对于它的返回值的，这样的一个判断。

我们在这些测试的基础上头，我们希望看到，我们的测试的结果，所以在JMeter中间，我们需要定义Listener，Listener可以定义在HTTP Request，底下，当然这样的一个Listener，就是关于HTTP Request的报告，它也可以定义在，Thread Group底下，那就是在Thread Group底下，所有的HTTP Request的，这样的一个报告。今天这个例子，我们是定义在Thread Group下，因为我们Thread Group上，定义了两个HTTP Request，这样我们可以看到，两个HTTP Request的结果，会显示在我们的最终的结果报告中间，Listener中间。

在JMeter中间它有多种Listener了，我们今天主要会用到三个Listener的，一个是最简单的View Result Tree，它会把结果显示成为树状的方式，我们可以看到每一个结果的请求，的格式，以及它返回的格式和返回的值，这便于我们去调试，整个系统的设置。

在做性能测试的时候，其实我们是要在，几秒钟之内，发出几百，甚至上千或者过万的请求的，所以我们也不会去一一地去看，它每一个请求的格式是什么，所以在大多情况下，我们更需要的是一种集合报告。所以我们今天会看到，第二种Listener，就叫做Aggregation Report，集合报告会把，一次中间的所有的结果，把它汇总起来，展现给我们。

如果想更直观的看到，这样的一个报告的话，我们可以用图的方式表现出来，所以我们今天会看到，第三种报告，叫做Graph Results。

好，我们来看一下JMeter是，怎样来做性能测试的，我们以RestFulDemo为例，来看一下如何用JMeter，来进行它的性能测试，我们在RestFulDemo，的目录下，先用git pull把代码更新成最新的，因为我的代码已经是最新的，所以我就进到target目录，中间，用java -jar，把RestFulDemo运行起来。

然后到JMeter的目录里头，我们可以看一下，在它的目录里头，有一个JMeter的执行文件，我们用./JMeter，把JMeter跑起来，我们装的JMeter的版本，是5.3版本的JMeter，首先把我们的，Test Plan的名字改，一下，然后我们先建一个，Thread Group，Thread Group就是我们准备，用一组线程来跑的群组，我们把，Thread Group，叫做，Goods-test，啊所以我们把上面的测试计划，改成叫做Web-test，在Thread Group里头，我们可以看到，它其实有很多的东西可以加的。

首先我们给它加一个，Config Element，我们给它加一个配置，第一个配置加的是 HTTP的，请求的默认设置，因为我们的协议，用的是HTTP，我们给所有的请求的，协议都是HTTP，然后我们的地址，因为所有的请求都是同一个地址，端口号8080，这是在这个组底下，所有的HTTP请求的默认设置。

然后我们再给它加一个，HTTP的头的设置，同样在Config Element里头，我们有一个HTTP Header Manager，我们要加一个头，因为我们是RESTful API，所以我们要把所有的头，都加上，Content-Type:application/json;charset=UTF-8的字符集。

在我们要在建请求，我们是想测的是，新增商品的 API，所以我们新建一个Sample，Sample是用来做测试取样的，我们的第一个Sample建的是，的，是一个HTTP，Request的，Sample，是用来发HTTP请求的，我们的名字叫做PostGoods-success，就是一个正常的，Post Goods的请求，它的Port和，Server的IP，我们就不用再输入了，因为我们已经是默认的，所以它的Request的类型，是POST，然后它的路径是\goods，然后它的Body Data里头，输入我们一个JSON，这是一个正常的Request。

我们期望它的返回值，首先我们期望它的返回码，应该是201，所以我们建了一个，在Goods下面，建了一个Response Goods的事情，我们希望它的Response Code，是等于我们加一个值，201，然后我们再加另外一个Goods（assertion）.

因为我们不仅仅，它的，Response Code是201，我们还要看它的，返回的JSON，字串，我们加一个JSON的，Goods，JSON的Goods，可以用来判断说它的返回值的，我们把它的名字改一下，因为我们这是用来判断它的，errno，所以我们把JSON的Goods，改成errno，这里有一个JSON Path的定义，JSON Path的定义，非常像XML的XPath，$表示根， .表示一级，因为我们在根下面有一个errno，所以我们就起了一个，$.errno，我们期望在这个地方，如果是正确的话，errno返回的值是0，给我们输入一个0，这是，我们关于第一个请求的，一个Response Goods，和JSON的Goods，分别用来判断它的，HTTP Status Code，和，它返回的JSON里头的值.

然后我们建另外一个HTTP，Request，前面那个HTTP Request是一个正常的，成功的，Post Goods的HTTP的，Request，我们现在建一个PostGoods-NoName，然后它的Goods的名字是没有的，的，这样的一个HTTP Request因为我们，知道，这个请求会抛出异常的，同样它的协议和IP不用去设，我们要默认的 HTTP请求的，方式是POST，路径是\goods，Body Data中间，我们把数据中间的，name的部分给它删掉，这样就不给它送name，过去它是空，再给它加Goods，它的Assertion首先，Response Code等于400，因为它是会出格式错误，的客户端格式错.

它的返回的JSON的值，同样加JSON的Assertion，我们也是去判断它的errno，同样是$.errno，我们期望它的值是，出错的错误是503，好，现在我们给第二个，HTTP Request，也建好了它的请求，以及我们期望，判断它的返回的值。

我们还需要建一个Listener，来显示我们的测试的结果，我们就在Thread Group下头，去建Listener了，当然我们也可以在HTTP，Request去建Listener，建在Thread Group下头，就下面的所有的HTTP的，Request的返回值，都会显示在Listener里，我们选择的建的是一个，View Results Tree，的Listener，这样就会把结果，用树状的方式来显示，这是最简单的，也是，我们首先一般都会建的一个Listener，了，我们先把我们整个的，JMeter的设置去存一个盘。

好，我们就可以跑一下，因为我们现在用的是设置默认，默认设置，其实它测试只会跑一次的，所以我们按一下可以看到，它跑了两个HTTP，Request一部分了两，其中一个是绿色的，因为那是正常的，它返回是200，另外一个是红色的，因为它的返回值是400，我们可以看到，在这里可以看到，它的Request和Response Body各自的值。

我们改一下它的errno把第二个的改一下，我们改成，然后把它的Response Code也改，一下，我们改成201，再来跑一下，我们先清掉之前的跑的结果，再来跑一下，大家可以看到，第一条是没有问题的，但第二条底下会出现了，两个Assertion的值，分别告诉你说，第一个Response Assertion它的值，跟它预想的不对，errono也是一样的，它的值它运行的也不对，所以这就是Assertion的作用，如果说它的返回值，跟你定义的Assertion是不一样的话，就像我们在JUnit一样，它会告诉你说，这个测试是不正确的。

我们再把这个值改回来，这个改回400，这样的话，用我说的，去清一下再跑一下，因为View Results Tree的话，它只能看到，我们的，每一次的跑的结果，当然我们需要去做一些，跑的结果的汇总，特别是我们在做性能测试的时候，会要把同样的测试跑很多的遍，很多遍，所以我们需要加一些新的Listener，我们把显示的加了一个，聚集报告，Aggregation Report，把所有的结果，把它汇集起来的一个。

然后我们再加另外一个Listener，其实Listener都是各种各样显示，你的测试结果的，这样的一种集合，我们加一个Graph Results，用图的方式去显示，这样的话，我们的结果，就会在三个Listener上，显示就会有三种显示的方式，一个是分别用Tree一个是用集合，报告，一个是用图。

然后我们把前面的，Thread Group给它改一下，我们现在让这个测试跑100遍，然后在5秒钟之内把它跑完，也就是在5秒钟之内，它会把整个测试，跑100遍，所以我们看到它的集合，可以看到每个Request各跑了100次，总计跑了500次，这是它的图的方式。

我们要起一个窗口啊去看一下，服务器的负载是怎么样，所以我们再起一个窗口，用SSH连上我们的服务器，我们用top命令，把这个服务器的性能，把它显示在窗口里头，top命令的话可以显示，当前每个线程每个进程，所跑的状况，以及总共的CPU的占有率，还有内存的占用的情况等等。

我们可以看一下，在用JMeter性能测试的时候，对这台服务器的测试压力有多大，我们把这个结果给它清掉，然后我们再把第三项，Loop Count给它设，2，这样的话，它就会把100个例子，跑两遍，我们先跑，一遍，这就跑了，每一个人Request各跑了200次，我们来看一下，测试的结果测试，对这个服务器的状态，然后把这个窗口拿到上面去，然后再跑一遍，让它可以看见，这个窗口拿下来，我们再跑一遍，这样又跑了200次，这个就能看到说，这个服务器的占用，达到了，最高达到了百分之五十几啊，也就是现在这个服务器的性能，我们在5秒钟之内，跑100次，跑100次，各跑100次，实际上是5秒钟之内，跑了200个请求，应该是，没有什么太大的压力，因为聚合报告这边我没有清，所以我们之前跑了100次，然后跑了两次，200次的，所以所有的测试，总共跑了500次，清空以后它就会重新清理，好，这就是我们关于JMeter的

# 5.1 MyBatis

MyBatis，市面上存在着两种对象模型，和关系数据库的映射框架，一个是Hibernate，而另外一个就是，我们在本门课程中间，会要介绍到的MyBatis。

Hibernate，是一种自动化的对象模型，和关系数据库的映射框架，它对于数据库的操作，提供了完整的封装，所以它在开发的效率上头，会比较高，我们只需要设计系统的，对象模型Hibernate会负责，根据我们设计的对象模型，去生成对应的表结构，以及去根据我们，对于对象模型的操作转换成为，对于关系数据库的SQL的，操作，当然由于这个过程，是由Hibernate自动完成的，所以说它自动生成的SQL的，语句，并不是最优的语句，当然Hibernate也提供了强大的功能，使得你可以自己去定义，对象模型和关系，数据库的映射关系，以及，通过SQL语句，直接去超过数据库，如果说这样去使用Hibernate的话，就不如我们，在本门课程中间，介绍的MyBatis方便。

MyBatis与Hibernate不同，它是一个半自动化的对象模型，和关系数据库的映射框架，它采用了XML和注解的方式，把对象模型和关系数据库的映射，把它写成一个配置信息，从而把对于数据库的操作，映射，成为，一个接口的方法，通过一个这样的配置信息，把对象模型的对象，映射成为关系数据库的表，把我们对于数据库的操作，映射成为接口的一个方法，从而使得我们在使用Mybatis的时候，关于JDBC参数设置，以及如何处理结果等等，这种繁琐而重复的工作，从工程中间就被移除了，所以我们只需要去管理，对象模型和关系数据库的映射，以及怎样去把，对对象模型的操作转换成为对于关系数据库的这样的一个操作.

由于说MyBatis是这样的一个半，自动化的这样的框架，所以我们在开发过程中间，是需要去写SQL语句的，同样也要去花精力，去处理这个，就是关系数据库和对象模型的，映射，以及它们操作之间的映射，这样的一个过程，虽然我们的开发效率，比这个Hibernate要慢，但由于原生的SQL语句，是由程序员自己写的，所以说使用Hibernate，可以严格地控制，SQL的执行性能，而且它的灵活度非常的高，可以对SQL做，作细粒度的优化，所以我们在本门课中间，主要介绍的就是这样的，MyBatis的这样的关系对象模型，和关系数据库的映射框架，它也是目前在国内互联网大厂，所采用的主流的对象模型，和关系数据库的映射框架.

对象模型是怎样映射到这个关系数据库的呢，我们知道在对象模型中间，存在着一系列的概念，比如说我们知道有类，对象模型中间最基本的概念就是类，然后由类我们可以实例化出一系列的对象，在类里头，我们可以定义属性，而每一个对象的属性又是不同的，同样对象和对象之间，还存在着继承关系，关联关系等等这些关系，我们当然还可以在类中间去写方法

这些对象模型的概念，是怎样映射到关系数据库的呢，在关系数据库中间，我们只有表，只有表中间可以插记录，然后在每一个表中间，可以定义它的字段，每一条的记录，它的字段是不一样的，我们可以看到，对象模型的这些概念，和关系模型的这些概念，是有着一定的相似性的，所以我们可以把类映射成为，把对象模型中间的类，映射成为关系数据库中间的表，把对象模型中的对象，映射成为关系数据库中的记录，把对象模型的属性，映射，成为表中间的字段.

由于这样的一个对象模型，是在内存中间的，而关系数据库，是存在数据库中间的，所以我们需要维持一个在内存中间的对象模型和在这个关系数据库中间的记录之间的关系，我们给每一个对象，要指定一个id，用 id作为在数据库中间的主键，从而实现对象模型的对象和关系数据库的记录之间的关联.

对于对象模型中间的关系，首先来说继承关系是在关系，数据库中间，是没有办法去支持的，但是关联关系，无论是一对一的关联关系，一对多的关联关系，和多对多的关联关系，我们在关系数据库中间，都可以用外键去实现，我们在学数据库的时候，应该知道，怎样用外键去实现一对一的一对多的，和多对多的这样的一个关系。继承模型虽然在，这个关系数据库中间不支持，但是我们可以用技术的手段，在关系数据库中间去实现，面向对象中间类的继承，以及在类的继承上头，去实现多态，我们在后面的例子中间，会给大家讲一个，如何去实现继承关系的例子。

最后是方法，由于方法是没办法写到数据库里头的，所以方法其实也不需要写到，数据库里，因为不管类的对象，是怎么不同的对象，它的方法都是一样的，都是定义在类里面的，所以我们不需要把方法写到关系数据库里去，那也就不会把这个方法映射到数据库里头去了。

MyBatis的这样的一种方式，我们前面说了，它是一个半自动的方式，所以说它的配置信息，主要是有两种方式来写，这个关系对象模型和关系数据库的配置信息，一种是用XML文件，另外一种则是用注解写在Mapper的，就是它的映射的接口上面，从而把对对象模型的操作，映射成为对数据库的操作，把数据库的把对象模型映射成为关系数据库中间的表，记录和字段。

这是MyBatis的这样的一个主要的机制，MyBatis在XML中间和注解中间，都提供了一系列的标签注解，或者说映射的标记，来描述这样的一个，操作的映射和对象模型，和关系数据库模型的映射，我们首先简要的讲一下，主要的标记。

第一个是insert，insert的标记主要是把，我们对于数据库的insert的操作，把它映射成为我们在对象模型中间的一个新增对象的这样的一个操作。Update标记主要是用来去映射Update SQL语句的，我们可以把针对对象模型的一个修改的操作，或者针对对象模型会造成它属性修改的一个操作，把它映射成为在关系数据库中间的，Update的SQL语句;Delete这个标记，是用来影射Delete的SQL语句的，当然需要把数据库中间的，把一个对象，把它彻底的销毁掉，我们就可以把它用，Delete这个标记，把它映射成为一条delete语句，select，是最常见的一个操作，select标记，是映射我们在数据库中间的select语句的，select的语句会返回很多不同的结果集，如果结果集，可以简单的映射成为，对象的话，成为类的话，我们可以直接把，这个类，作为它的返回的类型，但如果说我们有一些复杂的映射，我们可以用resultMap，这个标记，把返回的结果集映射成为，一组有关联的复杂的对象。

传参的部分来说，如果是简单的一个参数的话，你就可以直接传过去，MyBatis负责，把这个参数映射到我们在SQL语句中间的参数，如果说要传多个参数的话，我们需要定义一个，parameterMap，把多个参数放到parameterMap中间，把它传到MyBatis里，MyBatis从这里头去拿到对应的参数，去映射到SQL语句上头这个参数。

sql标记，主要是用来去做重用的，SQL代码的，我们如果有一段SQL代码，会在多处重用到，我们可以用sql标记，去把这一段的SQL代码，把它标记起来，然后再需要用到它的时候，用下面这个标签，用include的标签，把它引进进来，这是基本的MyBatis的标记。

除了MyBatis这些基本标记以外，它还有动态SQL，动态的SQL的功能，至于这些标记，以及动态SQL是，怎么来使用的，我们通过几个例子来看，这些标记和SQL语句，动态SQL语句，是怎样完成对象模型和关系数据库的映射的。

我们以RestFulDemoMybatis，的例子，的Mapper的XML文件为例，来看一下，MyBatis的这些，XML的标记是怎样来使用的，这个XML文件，是在工程的，要resources目录底下，在这个目录下，你可以看到，我们建了一个跟Mapper，的接口文件，相同的目录，因为，这个XML跟Mapper的接口，文件，最后打包时候，要打包到同一个目录底下去的，所以我们在resources底下建了的，一个相同的目录，这里头有几个简单的，跟商品有关的增删改查的，一些业务，我们看一下，我们怎样用MyBatis的，标记，来实现对象模型，和关系模型的映射，第一个我们看的是，select的findGoods，这样的一个标记，这是用的是select的标记，select的标记的，是从数据库里去查询，数据返回给对象模型，我们可以看到，它有定义了一个id ，findGoods加上上面的，XML，所定义的namespace，这个就对应到，我们的Mapper接口中间的，同样有一个方法，叫做findGoods的一个方法，parameterType，表示说，它的传过来的参数是什么样的，因为我们现在传参都是，用对象传的，所以我们不需要去特别定义，parameterMap，如果说，你要传几个参数过来，但是又不想用对象传的话，你就需要专门去定义，parameterMap，resultType，我们直接返回的是一个对象，所以我们就直接，把这个对象，作为它的resultType，在select标记里头，你可以看到，它是一个select的语句，但是我们可以看到，这里用了一个include的标签include是，一个，id，叫做Goods\_Column\_List的，SQL片段，我们在上面可以看到，这个SQL片断，它是用sql标记，来定义的，一段SQL语句，为什么定义这样的，SQL语句，是因为它有可能会重用，所以我们就用SQL，给它定义在这，如果说多次使用的话，那就可以用include的，把它引进来，这样的好处在于说，如果我们修改的话，只要修改sql里头的定义的话，所有的地方都会跟着修改，所以这里是select的字段名，我们是用一段SQL的，标记所标记的SQL语句来定义，这样用它include进来，后面大家应该看都比较简单，FROM哪个table，WHERE然后它的条件，在条件里头，我们用了一个动态，SQL语句中间的if的标记，为什么会用这个标记，原因是因为说，我们需要去看传进来的参数，中间就是一个GoodsPo中间，它不为空的属性，会作为它的查询条件的，所以我们要去判断说，它的几个属性，id，goodsSn, name，categoryId，brandId，和state我们只能用这几个条件，来做查询，那去判断这样的几个字段，它的属性是不是空，如果它不是空，也不是空字串，我们就会把它作为查询条件，把它组到SQL语句中间来，if的标记就是用来去判断属性，是不是空的，它的格式是，把判断语句写的字符串里头，test等于，在里头如果说它不为空的话，就要组上去的查询的条件，这里头我们可以看到，它用一个很特别的标记，就是#这个标记，这个标记是什么标记，表示说这个里头的值，是来自于parameter，来自于参数的，因为我们的parameterType，GoodsPo所以id就会是GoodsPo，的一个属性，把这个语句会做一个预编译，然后把#里头的东西，当做这个参数，在执行的时候，在跟预编译好的SQL语句绑定，上去，这样相对比较安全，当然我们还有另外一种符号，就是用$开头的符号，$开头的符号，实际上是用字符串的组上去的，我们知道，如果说，用$那个符号的话，很容易出问题，会产生SQL注入攻击的问题，所以我们一般都用#，符号，使得要查询的参数，或者要操作的参数，是被预编译的，然后在执行的时候，再把这个参数给它绑定过去，而不是在运行的时候，动态去组SQL语句，这是我们一般来说，写数据库应用的一个通用的原则，就是不会去组装SQL语句，而是把所有的SQL语句，都经过预编译过，然后把参数绑定上去，当然我们知道这样做的话，它的速度也会更快一些，这是select，它的返回类型是，GoodsPo，所以说，它会把返回的字段名称，对应到GoodsPo上面去，大家可能稍微注意一下，我们字段名里头，其实是有带下划线的，但是我们在GoodsPo里头的，属性，是用驼峰来写的，所以我们在配置文件中间，在application.yaml里头，其实是有一个定义的，这个定义叫做，map-underscore-to-camel-case=true，在配置里头，这个定义的作用，就是，我们会把带下划线的字段名，把它自动映射成为，驼峰的，这样的一种方式，因为这是两种系统不同的命名的，规则，这是select的标记，我们再看一个insert的标记，这是一个insert的标记，我们先看第一个比较简单的，就是我们要往数据库里头，插入一个商品信息，createGoods，传过来的参数是GoodsPo后面的，两个属性，对insert来说是很特别的，因为我们在数据库里头，有一个字段是自增字段，而且它是主键，主键自增字段，所以我们在这里定义了，keyproperty=id，就说明字段，而是要用useGeneratedKeys=true，就是要用自增的方式来，来自动产生的（也会把id设回到GoodsPo参数中），所以后面属性的名称，就是起这样的一个作用，那里头我们可以看到，一个，很正常的insert的语句，include把我们的字段名，给它，带过来，include的里头，同样也是一个用SQL定义好的，语句，后面的values，我们可以看到，同样用#，把GoodsPo类型的参数，里头的属性给它绑定上来，同样也是做预编译，的方式，update，updateGoods，是一个update的标签，它的里头很特别的地方，在于说它的set语句，因为同样的，我们在做update的时候，我们也是根据传过来的参数里头，属性不为空的，才需要update，所以我们这里，在set的这个组里头，我们同样用if的标签去，动态的SQL去判断一下，属性是不是空，如果不是空的话，才会组到SQL语句里去，在我们这个例子中间，我们没有Delete的例子，因为我们其实是不允许去做物理，删除的，我们都是逻辑删除，所以就看不到，DELETE的例子，最后我们看一个，resultMap，的定义，我们在上面，可以看到，我们定义了一个叫做GoodProductResultMap，为什么会定义这样一个，resultMap，是因为我们GoodPo，是一个有关联的对象，就是一个商品的里头，它有多个规格，所以在GoodsPo中间，有一个属性叫做productList，它其实是商品关联的规格对象，当然我们在数据库里，知道它是两张表，我们这定义了一个，resultMap，它的作用是什么，它的作用是，我们要去查询这两张表，最后形成的是一个，商品对象中间，带了多个规格对象的，这样一个有关联的对象，返回回来，我们注意看一下resultMap，它的type是说，resultMap，最后形成的对象，会是一个什么类型的对象，我们形成的是一个GoodsPo，对象，所以后面的property，全是定义的是，GoodsPo的property，以及Product里头的property，后面是id，后面我们写了一个autoMapping，因为在resultMap中间，你如果不写autoMapping的话，你其实是要一个个地去写，它的属性和字段之间的关系的，我们这里用了autoMapping以后，如果说能够自动映射的，就按照全部自动映射，但是有些东西是不能自动映射的，比如说，下面第一个id，因为我们要显式的说明说，在resultMap中间，哪个属性，是用来标识对象的id的，所以id这个变成标记，就是用来说，column id的字段，要对应到property的id的字段，这两个是，它的主键，以及它是唯一标识这个对象的，其它的我们都没有写，看见没有，其它它都是，让它去做autoMapping，自动映射的，后面的部分是collection，因为我们里头有一个属性，叫做，productList，这个属性，它是一个List类型里的，每个元素是ProductPo，所以我们后面定义的ofType，实际上是说，在collection的，每一个元素的类型，是ProductPo，这里，要把整个的包名全要写全，同样使用了autoMapping，由于说有一些的字段的名字，和属性的名字，是不能做autoMapping的，所以我们这里除了id以外，可以看到写了，4个result，这是因为这都autoMapping不上去的，字段名和属性名，这样就会定义了一个，resultMap，不是简单的返回一个GoodsPo，对象，而是返回了一个带关联的，GoodsPo对象里头，有productList的GoodsPo，对象，resultMap还可以做得很复杂，我们现在是一次，把两个东西都查出来，它也可以做成这种延迟加载的，方式，当你访问到时候，它才查，不访问的时候，它就不查，我们看一下resultMap，在这里是怎么用的，我们下面有一个是select，它是findGoodsWithProduct，resultMap就是要把，商品，和它的规格一起带出来，它的参数是GoodsPo，注意看它的返回值，不再是写的是返回的，类型resultType，而是写的resultMap，我们要用一个resultMap，来装它的返回值，当然resultMap，最后返回出来的对象，还是GoodsPo对象，因为resultMap，那边，定义的type，GoodsPo，这条SQL语句，其实是一个，两表的连表查询，所以select很长，有两个表里头的字段，然后在FROM后头，我们可以看到，LEFT JOIN，也就是左链接，我们让Goods这张表，去左链接，Product那张表，然后链接的条件是，Goods的id，等于Product那张表中间的，goods\_id，后面就是正常的条件，我们知道这样的一个，连表查询的语句，查出来以后，它是会出现多条记录的，因为一个GoodsPo是有多个ProductPo，但是我们定义了返回的值，是一个resultMap以后，你可以看到，MyBatis，把多条记录的这样的一个返回集，把它映射成为了一个，一对多的对象关系模型，这就是我们resultMap，的，一个最经典的使用，它用来去定义我们的这样一个，对象模型，跟数据库的返回值之间的对应，关系，然后MyBatis，根据这个定义关系，把数据的结果集，映射成为一个对象模型

# 5.2 Spring中的MyBatis

Spring中的MyBatis，在Spring中间，MyBatis主要体现在三个地方。

第一，我们需要写一个Mapper的接口，来定义我们对于对象模型的操作，也就是说MyBatis通过接口，去实现对于对象模型的操作映射成为对于关系数据库的操作。第二，我们需要写一个XML对应着我们前面写的Mapper的接口，然后在这个XML中间，我们写的是这样的一个操作，是怎样映射成为一个关系数据库的操作的，比如说，它是映射成为一个什么样的操作，它的值，它的对象的属性，是怎样映射到关系数据库的表的字段上头的。第三个，我们需要去写配置信息，来告诉Spring框架，到哪去找Mapper，以及到哪去找XML。

首先我们来说，Mapper的XML，Mapper的XML，中间，我们首先要定义一个，namespace，namespace，是Mapper XML的名字，这个namespace，的名字，要定义的跟我们前面说的，Mapper的接口的名字是一致的，在Mapper的XML中间，我们每一个Statement，每一个标记，都有一个id，Mapper XML的namespace它的名字，加上这个id，就是我们可以唯一标识的一个操作，而这个操作的名字，跟我们在Mappper接口中间，所定义的，Mapper的类名，和Mapper的方法名，是一一对应的，这是Mappper的XML。在Mapper接口中间，我们还需要定义，每一个接口方法的参数，和返回值，这个参数和返回值的定义，也需要和在Mappper的XML中间的，每一个Statement定义它的resultType或者resultMap，parameterType，和parameterMap是一致的。同时我们要注意，Mapper的XML文件，和Mappper的接口文件，我们需要定义在同一个目录底下。

Mappper的接口文件，是一个Java类，所以说它是定义在我们的一个包，底下的，而Mapper的XML文件，是一个静态的文件，这个文件我们通常放到resources目录底下。我们在前面讲Maven的时候，讲到过resources底下的对应目录的文件，会拷贝到我们的Java代码的.class的对应目录上去，所以意味着说我们的这些XML，需要在resources底下，去建一个，跟Mapper的接口相同的包相同的这样一个目录结构，把 Mapper的XML文件，放到这个里面来。

第三个就是它的配置信息，在Mapper中间的MyBatis的配置信息，我们都统一写在application.yaml底下，在配置信息中间，我们主要是有两项，第一个是mapper-locations，mapper-locations，主要是告诉Spring框架，到哪去找XML文件。第二个type-aliases-packages则是说到哪去扫描我们在Mapper中间所定义的类型的别名，Mapper，中间所有使用的对象类型，如果我们不指定别名的话，我们是需要完整的写它的包名，和它的类名的，这样比较繁琐，所以我们通常会用一个@Alias的注解，去标记说这些，所有的这些在MyBatis中间要用到的映射的名字的别名，这样就避免，我们去把它的完整的包名，全部写下来。我们所说的第二个配置项的作用，告诉Spring框架到哪去找这些别名。

我们看到MyBatis的信息，是定义在，Mapper接口和Mapper，XML文件中间的，这些配置信息，是怎样进入到Spring框架的内部的，我们首先要在的工程中间使用MyBatis，我们需要引入一个，mybatis-spring-boot-starter，这样的一个starter，引入进来以后，它会自动识别，在Spring框架中间，已经定义的DataSource，也就是它的连接的数据库，通过它的SqlSessionFactoryBuilder，去根据传入的配置信息，我们的配置信息，或者是在XML中间，或者是在注解中间来生成Configuration对象。

Configuration对象里头，就已经包含了我们在XML和注解中间所有写的MapperStatement的对象，每一个MapperStatement对象，都是常住在内存里头的。

根据生成出来的Configuration类型的对象，然后我们创建用于去构造数据库连接的SqlSessionFactory的对象，有了SqlSessionFactory的对象。

以后它会创建并登记一个SqlSession的对象， SqlSession的对象会开启一个数据库的会话，它会自动扫描，我们在这个前面定义中间，所定义的目录中间的所有的Mapper的接口文件，然后创建自动的Mapper的对象，并把它们登记到Spring的上下文里面，以后我们在Spring框架中间，就可以把这些登记上去的这些mapper对象，注入到我们的Spring Bean对象中间，从而可以直接使用这些mapper 对象中间所定义的接口的方法，去操作我们的数据库。

所以SqlSession的对象，它会去管理我们的数据库的连接，创建我们的mapper对象，同样，它也会控制在数据库中间的事务，以及去负责对于数据库的查询。Mapper接口，大家注意到，它是没有实例化的对象的，我们在Spring框架中间仅仅是定了一个接口，还没有定义它的实现，Spring框架中间，是怎样来把通过一个接口来实现对象模型和关系数据库的映射的。我们前面说过，在Mapper的接口文件中间，我们的Mappper的接口的名字，加上这个方法的名字，必须跟XML中间的一个标记的namespace，和它的id的名字必须是一致的。我们前面说到过，当MyBatis的框架起来的时候，它其实会读取这个XML中间的配置信息，或者注解中间的配置信息，去创建Configuration对象，也就是我们说的XML中间的namespace，加上id的名字，所构成的这样的一个MapperStatement的对象。

当Spring框架去创建了一个Mappper的接口的时候，它实际上是利用JDK的动态的代理机制，去创建了一个动态代理的对象，来实现了我们的这样的一个Mapper的接口，我们知道这是Spring框架非常常用的一种方法动态的代理对象，是根据我们调用的Mapper的名字，去到我们的Configuration对象中间，去找到对应的MapperStatement的对象，然后把我们传过来的参数，映射到MapperStatement对象的这个参数上头，执行MapperStatement对象中间的SQL语句，把它的返回值在映射回到我们的Mapper的返回值里头，最后实现了我们对象模型和关系数据库的操作的映射和数据的映射，这就是在Spring框架中间，是怎样来实现MyBatis的对象模型和关系数据库的半自动映射的这样的一个机制的

# 5.3 事务

事务，在Spring框架中间的事务，是比数据库的事务，更大的一个概念。

数据库中间的事务，是指操作数据库的，一个操作序列，它要么全部执行，要么全部不执行，这中间有一个失败的话，就全部会把它回滚回来，所以它是一个完整的整体。在Spring框架中间，它的事务的概念，是不局限于，对数据库的操作的，也就是在Spring框架中间，只要是说它的数据源是支持事务的，它可以把多个数据源的这样的一个操作，把它放到一个事务里头来，把它作为一个整体，其中有任何一个失败了，它就会把整个的序列，把它全部回滚回去。

事务所具备的最主要的特性，就是我们说的ACID的特性，首先A是代表原子性，原则性就是我们前面说的，事务中间的所有的操作，是不可分割的，它是一个整体，要不全部完成，要不全部就不执行，C表示它的一致性。一致性是说，当事务有多个事务来并行的时候，它的执行的顺序，执行的结果，一定是一致的，一致的结果跟它的某一个顺序的，执行的结果是相同的，那不会说因为说，多个事务的执行，会造成它最后执行的结果，是不一样的，第三是它的隔离性，隔离性是说在并行执行的事务中间，它是不受到其它的事务影响的，每一个事务在执行过程中间，它所看到的东西，应该是属于它自己的，也就是说它对于其它的事务，对其它的事务的执行，对于当前这个事务来说，它都是透明的。第4个就是持久性Durability，第一，持久性是说，如果事务已经结束了，它提交的数据系统，一定要保证事务的提交的数据，会把它写进去，它的修改要是不可以丢失的，即算是说当事务结束以后，系统会发生故障，这个数据都不应该是丢失的，这就是我们说的可持久性。这是我们说事务的4个特性，ACID，我们在后面展开这个Spring框架，的，事务的讨论的时候，围绕着ACID来讨论的，特别是其中的，C和I就是它的隔离性和一致性。

在Spring框架中间，你要标识一个事务，是通过注解@Transaction来标注的，@Transaction，是可以标注在Bean对象的方法的前面的，或者标在Bean对象的类的前面，你如果把它标在Bean对象的类的，前面，就表示说类的所有的方法，都是要有事务的，当然表示在方法前面，只是说，当前这个方法是要事务的。

我们经常会，把注解标在了Service层的对象的方法前面，原因是因为在前面的，Controller层的对象，主要是负责数据的合法性检查，以及返回的数据的封装的，在返回的数据封装的时候，其实不太容易出错，而数据的合法性检查，如果出现问题的话，它是在所有的其它操作之前的，所以说它就不涉及到事务的回滚的问题，正是因为这样的原因，我们就会把@Transaction的注解，把它标注在Service层对象中间。

@Transaction的注解，是有几个属性的。第一个属性，叫做propagation，它是用来去标注事务的传播方式的，第二个是isolation，这是标识事务的隔离级别，这两个属性，我们在后面会仔细的讲。

第三个是readOnly是标识，当前是一个只读事务，我们知道事务是会对数据进行写，作的，默认情况下，所有的事务都是读写操作，写的操作会对于在并行事务上面的，它的一致性和隔离性会产生问题，所以说它需要更严格的，这样的一个隔离的方式，对于读的操作来说，因为它不会对数据发生改变，所以它对一致性和隔离性的影响，是没有的，如果说你的数据是一个只读的数据，你只需要保证自己的一致性和隔离性，而不会去影响其它的事务的，一致性和隔离性的话，你可以把事务的readOnly的标签标成true,默认是读写的，把readOnly标成true以后，它会对其它的事务的影响会更小一些。

最后4个分别是用来标识，哪些异常会影响引起事务的回滚，或者不引起回滚的事务，分为两组事务1，rollbackFor/rollbackForClassName事务3，和noRollbackFor/noRollbackForClassName事务5，rollbackFor/rollbackForClassName标识事务6，会引起回滚的异常的事务，rollbackFor是用这个名字来标识的事务8，异常的名字来标识的事务9，rollbackForClassName，则是用类的对象，异常类的对象来标识的，noRollbackFor/noRollbackForClassName，是标识不会影响回滚的，异常的分别也是用名字和用类来，标识。

我们来详细讲一下，它的传播设置，propagation型和隔离级别，这两个属性的设置，首先我们来讲，propagation，在Spring框架的Transaction中间propagation，可以设这么几个，默认的值是REQUIRED，REQUIRED是说，当前如果没有事务，它就会开启一个新的事务，当前如果有事务，它就会沿用这个事务，为什么，调用这个方法的时候，当前会有事务，因为我们前面说了，我们会把注解写在了，服务层的方法的前面，服务层的方法有可能会被另外，一个，服务层的方法来调用，如果说那个服务层的方法，是有开启了事务的话，这个事务就会沿用到，当前的方法里头来。如果说你标示的是REQUIRED，如果它就会，这个方法，就会沿用前面方法的事务，不会新建一个新的事务，如果说前面那个方法，是没有事务的，REQUIRED的属性，则会新建一个新的事务，也就是说，不管怎么样，都保证当前的这一段，这个方法的代码，是在一个事务里运行的。这里特别要说一下，如果它沿用前面那个方法的事务，的话，如果这段代码出现了，会引起回滚的异常的话，它会引起前面那个方法的事务，也一起回滚，所以当前的方法的操作，和前面调用，它的方法的操作，就成为一个原子性的整体，要么全部成功，要么全部失败，这个REQUIRED是默认的设置。

第二个设置是SUPPORTS，SUPPORTS表示当前的这一段，代码，是支持事务的，所以说如果调用它的方法，是有事务的话，它就是会在这个事务中间，同样它失败，也会造成前面那个事务的所有的，操作回滚，如果当调用它的方法，没有事务的话，它也就没有事务，所以说它不象REQUIRED，会强制新建一个事务。

第三个是MANDATORY，MANDATORY是标识，这个事务，这个方法是必须要有事务的，所以当调用它的方法，有事务的时候，它就会在前面，那个方法的事务中间，去跟着执行在一起执行，如果调用它的方法，没有事务的话，它不像会新建一个事务，而是会抛出一个异常，说这个方法是需要事务的。

第4个是REQUIRES\_NEW，REQUIRES\_NEW是说方法，是要新建一个事务出来，所以不管调用它的方法，有没有事务，它都会新建一个事务了，我们特别注意一下，当调用它的方法有事务的时候，它新建的事务实际上是，在之前的事务之上的，也就是它把前面事务挂起，然后开启一个新的事务，这样的话，如果当前的方法出错的话，它所引起的回滚，只是在当前事务里头，而不会影响前面事务的相关的，操作，这是REQUIRES\_NEW。

第5个是NOT\_SUPPORTED，这表示当前事务，是，不支持事务的，所以说如果调用它的方法没事务，那就没事务，没什么问题，如果调用它的方法有事务的话，它也会以没事务的方式执行，就是把调用它的方法的事务，挂起来，这段代码是没有事务去执行的，它出错不会引起，调用它的方法的回滚。

最后一个是NEVER，NEVER是表示说，当前这个操作是不能有事务的，所以说如果调用它的方法，没事务，那就没事了，如果调用它方法有事务的话，它会去抛出异常的。

我们其实还有一个list，NESTED是表示说嵌套事务，其实NESTED的嵌套事务，它跟我们前面所说的，REQUIRED是比较相像的。

第二个我们要谈到的属性，就是隔离级别，隔离级别，跟我们在事务中间的，并发中间的问题，并发的三个问题有关，就是我们说的脏读问题，不可重复读的问题，和幻读的问题。

什么是脏读问题呢，脏读问题是说，事务和事务之间的隔离性，是没做好的，A事务，它在整个的事务过程中间，它会去修改数据，B事务会去读到A事务，在过程中间，所修改的数据，如果说A事务最后回滚了，它把它修改的数据，改回成原来一样的，但是由于B的事务，它读到了A事务的，过程中间的数据，我们就说B事务读到了，A的脏数据，所以说这是脏读的问题，脏读问题其实就是，完全没有隔离事务和事务之间。

如果说隔离了事务的，这样的一个，之间的东西，它不能读到这个事务，这中间的数据的话，我们就会出现第二个问题，叫做不可重复读，也就是，假如说A事务的时间是比较长的，它首先读了一个数据，然后它读完数据以后，我们开启了另外一个并行的事务，叫做B事务，B事务的话，它会去修改这个数据，B事务结束了以后，它会把它的修改的数据值提交，上去，A事务，如果在B事务结束以后，再去读数据的话，它就发现，它在，B事务开始之前，读的数据和B事务，完成之后的，两个数据是不一样的，这就是我们称之为，不可重复读的这个问题，要解决不可重复读的问题，我们知道最简单的方式，就是对数据来进行加锁，比如说加一个读锁，A事务在读了以后，它就对这个事务进行了读锁，那B事务就是不可以去修改它，只可以去读它，这样的话，就能够保证，不可重复读的问题。

加了读锁就能保证它的不可重复，读，但是它也不能避免我们的第三个问题，叫做幻读的问题，幻读的问题是这样的，假如说A事务，它要统计在系统中间，所有的商品的总数，所以说它在读取了系统中间，所有的商品的数量以后，它把所有的商品，都加了一个读锁，但是B事务，可以往这个商品中间，去新增一个新的商品，所以说如果A事务，再去读一遍，有商品的总数的时候，它发现多了一个商品，这就是我们称之为幻读的问题，因为你虽然对数据进行了锁，但是你不能禁止，另外一个事务，往里头去，增加一个新的数据，要解决这个问题，就是要做到完全串行，我在读数据的时候，你不能对这个数据，做任何的改动。

所以为了避免上述的这些问题，我们可以对数据库，设定事务的隔离级别，我们的默认的级别，DEFAULT，我们在属性中间，可以把它设成默认的级别，因为默认的级别，实际就是数据库所使用的隔离，级别，数据库的隔离级别，是可以在数据库里单独去设置的，MySQL口的数据库的，默认的隔离级别，就大家没有去动它的话，它是READ\_COMMITTED，只读到隔离了数据库，事务和事务之间的操作，只读到事务提交以后的数据。第二个级别是，READ\_UNCOMMITTED的，这是一个，最低的级别，最低的级别就是完全没有隔离，事务和事务之间，你可以读到任何一个其它事务，在中间过程所产生的数据，所以它会产生所有的问题，脏读问题不可重复读问题和幻读，问题。

提高一个级别，我们READ\_COMMITTED的这样隔离，了，是事务和事务之间的，这个操作，就是你不会读到另外一个事务，在中间过程中间所修改的数据，但是它会有我们的，不可重复读和幻读的问题。

第三个级别就是为REPEATABLE\_READ，我怎么读这个数据，这个都是一样的，都是一样的，所以这样的一个操作，是对这个数据进行了这个读锁，但它不能解决幻读问题。

最后就是SERIRALIZABLE，定义的级别，SERIRALIZABLE就是完全串行，完全串行以后，你就可以解决脏读，不可重复读和幻读的问题，所有问题都解决了，但是它会带来更大的问题，就是所有的数据的操作，都变成串行的，这样完全没有并行的话，这样的一个系统

# 5.4 事务的实验

我们在MySQL8.0的数据库，上面，来做4个实验，来看看我们的这4种事务的隔离，级别，是怎样处理脏读问题，不可重复读的问题，还有幻读的问题的，我们的MySQL的数据库，用的是MySQL8.0，所以首先我们来看一下，数据库的全局的事务，隔离性，可以看到它的MySQL，全局的事务，隔离性是REPEATABLE READ，我们也可以看一下，当前会话的事务隔离性，它也是REPEATABLE READ，这是两个不同的概念，在@Transaction的注解后面的，isolation的属性，所设的事务的，隔离的级别是设的会话的事务隔离，级别，也就是我们看到的第二个值，所以我们后面的所有的，实验都是在会话的事务，隔离级别上面来做，我们现在看到的是，它的会话的事务级别，跟全局的事务级别是一致的，所以我们首先把它的会话级的，事务级别，设成了READ UNCOMMITTED，我们看了两个窗口，我们把这两个窗口，会话的事务隔离级别，都设成了READ UNCOMMITTED，设成了READ UNCOMMITTED，以后我们预计说，我们在一个会话中间，在一个事务中间，对数据库的修改，即使还是我们还没有提交这个，事务，而在另外一个会话中间，就能够看见，所以设好了以后，我们分别看一下，这2个会话的事务隔离级别，当前的会话，事务隔离级别，都变成了READ UNCOMMITTED，可以看到，它变成READ UNCOMMITTED，这边看一下，其实它也变成了READ UNCOMMITTED，好，我们使用oomall\_demo中间的，oomall\_product数据表，我们首先分别开启事务，start transaction 两个会话窗口，都开启事务，所以我们先看一下，oomall\_product这张表，里头的内容，这是产品的规格信息，其中有一个是规格的库存，就是stock，我们打算来修改库存量，stock的值，我们在另外一个会话，也看一下，这时候，两个会话中看到的，product的，的信息，都是一致的，所以我们要在一个会话中间，用，update语句，去修改product的stock的值，它的库存量，我们把它的库存量设成99，只是把id=1的规格的库存量，设成99，设完了以后，我们虽然还没有提交事务，但是我们在这个会话中间，就能看到，它的值变成99了，由于现在的事务的隔离级别，是READ UNCOMMITTED的，所以我们在另外一个会话中间，去看一下它的值，已经看到的时候，它能够看到，在，第一个会话中间，还没有提交的事务，它改成了99，我们现在第一个会话，中间把事务会滚回去，所以再去看的话，它就变成了100，这样的话，实际上它的事务的最后的值，stock还是100，当然在这边也能看到，它改成100了，所以这就是READ UNCOMMITTED级别，它能够读到另外一个会话中间，在事务过程中间，还没有提交的，之前的中间值，我们再来看第二个，第二个我们把它的，session transaction isolation level，把它的事务隔离级别，设成了READ COMMITTED，这样让它只能够读到，事务提交以后的，所以我们先用，set session transaction level的命令，把它设成了READ COMMITTED的值，然后我们来看一下，它的会话的，transaction isolation的值是多少，可以看到，它已经变成了，READ COMMITTED，在这边的第二会话中间，我们同样把它的，事务的隔离级别，会话的事务的各类级别，也改成了，READ COMMITTED，所以看一下它的会话隔离级别，会话的事务隔离级别，它也改成了READ COMMITTED，然后同样，我们在两个会话中间，分别开启事务，我们同样准备去修改，oomall\_product这张表里头，的库存信息，所以我们先看一下，修改之前的，库存信息是多少，它是100，好，我们同样用update的语句，把它的库存信息，库存量修改成为我们把id=1，的规格，信息的库存量，修改成为99，在提交之前，在执行这条命令之前，我们先在第二个会话中间看一下，product的库存量是多少，这时候显示的应该是跟第一个，会话中间是相同的值，是100，所以我们这边执行一条update，语句，在第一个会话中间，这时候没有提交事务，但是在第一个会话中已经看到，它被改成了99了，第二个会话中间可以看到，它还是100，然后我们在第一个会话中间，把事务提交，然后在第二个会话中间，就能看到提交以后的数据，这就是READ COMMITTED的隔离级别，在这个隔离级别上头，我们只能中间的数据是看不到的，只能看到会话事务，提交以后的数据，第三个实验，我们把会话的隔离级别，提升到REPEATABLE READ的级别，我们其实先不用设，因为MySQL8.0的，默认的级别就是，REPEATABLE READ，所以我们就不需要，再去做那样的一个设定语句了，所以可以看一下，我们开了三个会话，三个会话的，隔离级别都是REPEATABLE READ，好，我们的三个会话中间，分别开始一个事务，然后在第一个会话中间，我们准备去修改库存量，但是在修改它的库存量之前，我们先在第二个会话中间，去查一下库存量是多少，在第一个会话中修改库存量，提交以后再来在第二个会话中间去看，一下的是什么，然后再在第三个会话中间，去看一下它是什么，所以我们先看一下，库存量，现在我们可以看到，第一个，id=1的规格的库存量是100，在第一个会话中间，和第二个会话中间，我们都来看一下，第三个的话，我们不看，我们等到它事务提交以后，再来看，所以两个会话中间，可以看到，它的库存量都是100，好，我们现在第一个，事务中间，来修改它的库存量，到99，我们把id=1的规格的库存量，修改成为99，然后把它提交把事务提交，我们在这里看一下，因为它提交以后，可以看到，它的库存量已经改成99，我们在第二个事务中间，可以看到，它看到的库存量依然是100，所以说明，它在第一个事务开始之前，看过一次，第一个事务在更新之前，看过一次是100，它在第一个事务有提交完了以后，再看的话，它还是100，就是说它，只要它的事务开始了，它看过的数据，它永远都会是一样的，所以这叫做，REPEATABLE READ，但是第三个事务，因为它之前没看过，所以我们在这时候，在第一个事务已经提交的时候，再去看的话，会发现它的库存量，已经是99了，跟第一个的会话是一样的，最后一个实验，我们来尝试一下SERIALIZABLE的，这样的一个隔离级别，串行的隔离级别，我们开启两个会话，分别把它们的会话级的，事务隔离级别都修改成为，串行的SERIALIZABLE，串行的隔离级别，主要是为了解决幻读问题，所以在这里，我们不在去修改它的数值，而是试图往里头去，插入一个新的产品规格，所以修改了完了以后，我们在2个会话之间，分别看一下，它当前会话的事务隔离级别，都改成了串行的SERIALIZABLE，好，我们在第一个，会话中间开启一个事务，然后在第二个会话中间，也开启一个事务，在第一个会话中间，我们打算插入一条新的产品规格，我们先分别看一下，两个会话中间看一下，产品规格的信息，这时候，都是三条产品规格信息，我们准备插入在第一个会话之间，我们准备插入一条，新的产品规格信息，插入的时候，你发现，这条insert的语句，是会停在这，原因是因为，我们的第二个会话，已经开始了一个事务，而且在查看这张表的信息，所以它会使得我们的第一个会话，的插入的的操作被停下来，在等待着，第二个，会话中间这个事务结束，所以我们把第二个会话的事务，结束，就可以看到，第一个会话的

# 5.5 MyBatis的例子

我们来讲一下，RestFulDemoMyBatis的例子，这个例子比之前的RestFulDemo，要复杂了很多，因为它包含了RESTful API，和MyBatis，两个部分的内容，所以这个例子，已经初步展现了，我们这样一个基于数据库的，基于MyBatis的，应用程序的一个层次体系结构，我们可以看到，在这个例子中间，我们所有的代码被分成了，Controller层、Service层，DAO层、Mapper层，以及还有一个model的包，我们不能把model包称为是某，一个层，因为model包里头对象，其实是用在上面那4层对象的，所以我们一般来说，我们这个例子中间，用了4层的这样的一个，分层的体系结构，来实现我们这样的一个应用，我们来讲一下，这4层的职责分别是什么，第一层，是我们的控制器层，controller，controller层的职责，主要是负责接收前端的数据，把这些数据传给后端，的处理，封装，要返回给前端的数据，以及处理一些，错误码、异常的处理，这是在controller层的职责，第二层是我们的Service层，Service层的主要的职责，是用来去写我们的业务代码，所以在我们这个例子中间，因为我们的业务代码，是比较单薄的，我们这个例子，主要是对于，商品和规格对象的增删改查，所以大家可以看到，在这个例子中间，Service层的代码是非常少的，基本上是没有，这是因为我们的业务，简单，现在的例子，中间业务简单所造成的，第三层在下面这一层，是我们称之为DAO层，我们称之为DAO层是，用来去访问数据的，当然我们访问数据的定义是说，它要从数据库里去拿到数据，然后把数据封装，成为我们想要的，对象的样子，所以DAO层中间，因为现在的业务，主要就是数据的增删改查，所以大家可以看到，我们现在这个例子中间，大量的代码是在，DAO层的，最后一层是Mapper层，Mapper层的主要的作用，是用来实现对象模型，和关系数据库的转换，所以这里头，主要是一个Mapper接口，还有，我们把XML，也归属到这一层里头，这是我们整个，程序的4层的体系结构，它是一个严谨的分层结构，也就是Controller层，只能调Service层，Serivice层只能调到DAO层，DAO层只能调Mapper层，我们没有让它去，跨层的这样的一个调用，除了这4个包以外，我们还有一个包叫做model，我们把model包画在边上，这个包里头，其实包含了三类的对象，一类对象是VO对象，VO对象是用在Controller层的，就是Controller层的作用，主要是两个，第一，是接收从前端过来的数据，所以我们在这里头有，叫做GoodsVo和ProductVo， 这一类的VO的里头，是有带合法性检查的，因为它要对前端接收过的数据，进行合法性检查，前端过来的数据，跟我们在数据库里存的数据，以及我们在代码中间的，在系统中间的对象模型是不，一样的，所以，我们的VO，123，完全按照前端所需要传的数据的，格式来定义的，它跟我们的对象模型，和数据库的存储，第二类的，VO我们在命名上，给它加了一个RetVo，这在这里中间有两个RetVo，一个是GoodsRetVo，一个是ProductRetVo，这两个VO的作用，是用来去封装，返回给前端的数据的，所以大家可以看到，在RetVo中间，其实是没有，没有合法性检查的，我们让RetVo的设计，就是只封装了，我们需要返回给前端的数据，就是前端想要什么样的数据，或者我们在API中间定义的是，给前端什么样的数据，就用这样的一个VO去封装，它，在model中间的第一类对象，两个VO分别用来接收，从前端传过来的数据，和返回给前端的数据，在model的第二类对象，我们称之为业务对象，有人把它称之为BO，Bussiness Object，但是在这，我们没有去给它做特别的名字，大家可以看到后面没有后缀的，比如说Goods， Product，这一类的对象，我们称之为业务对象，业务对象实际上是我们，构成对象模型的，重要的组成部分，换句话说，我们对象模型，其实就是由业务对象，来构成的，我们是要把对象模型，组件组装出来，然后去在上面去执行，去实现我们的业务逻辑的，但是因为这个例子中间，它其实没有任何的业务逻辑，就是增删改查，所以在这里中，我们并没有体现出来，BO的重要性，但是我们还是把，BO的这一类对象，就是业务对象把它建了出来，所以这里头有两个业务对象，Goods和Product，第三类对象，是我们的PO， PO的作用，是用来去这对数据库访问的，在我们这个例子中间，PO基本上跟数据库里头的表，是一一对应的，所以每一张数据库里，这个表，就会有一个对应的PO，数据库里，的字段，表里的字段，就是PO的属性，所以PO的对象，并不能体现出，面向对象的模型出来，我们的DAO层的作用就是，从PO中间，去构建出，我们的BO对象，就是我们的业务对象，然后让Service层在，这些业务对象上头去，实现它的业务逻辑，实现它的业务逻辑，这是我们在model上的三类的，对象，除了model包以外，我们其实还有一个，包叫做util, util包里，头，是我们一些通用的工具，包里头的，代码很多都是用静态方法来实现，的，所以它不是用，面向对象的思想，来实现的，所以上面的所有的代码中间，就是我们4个层，甚至model包里头代码中间，都有可能用到了util的这个包，代码，这是我们整个代码的这样的一个，结构，我们具体来讲一下，在这个例子中间的，VO、BO和PO的关系，在我们的代码中间VO是有，两种，一种是用来传对象的，一种是用来返回值的，所以传对象的那一类的，VO里头，都有一个方法，就是它可以用来去创建对应的，BO的对象，创建BO的对象，在我们这个例子中间，为了节省内存空间，我们其实把PO对象，放到了BO对象里头，把PO对象，放在了BO的对象里头，所以BO对象，其实并不是真正，存在的一个对象，虽然它是一个类，但是你们可以看到，所有的BO对象里头，都放了一个PO对象，它其实是把PO对象，用BO的方式，给它展现出来，这里其实是一个，我们称之为装饰器的设计模式，当然也有人把它称之为代理设计，模式，不管是哪种名字，反正它的作用就是它的里头，其实是一个PO但是它展现出来，的是一个BO，所以它把PO中间，不需要看的一些属性，以及在PO中间，没有的一些属性，在BO上头，把它实现出来，所以这是这样的一个结构，对于每一个BO来说，因为它最后是要返回值，返回给前端的，所以BO的话，它是可以产生一个，RetVO的，这样的一个对象的，所以我们可以看到，所有的BO对象，都实现了一个接口，叫做VoObject，接口，中间定义的方法，就可以从BO对象中，创建出来，一个需要返回给前端的，这样的一个RetVo的对象，这是我们在model的，几个对象之间的关系，我们来看一下整个工程的结构，RestFulDemoMyBatis，工程在3. MyBatis底下，这底下我们可以看到，它有一个目录叫做sql，这个sql底下，我们放了一个叫做，database.sql的文件，这个语句是用来建用户和建Database，所以当你把MySQL，装好了以后，要把这上面的这些语句，拷贝到MySQL的命令行里头，去执行它，执行的结果，就会在数据库上面，建出一个叫做oomall\_demo的数据库，然后会建出，demouser的用户，还有它的密码，然后会把oomall\_demo的，数据库的，所有的权限，都付给了demouser的用户，完成了这以后，我们首先来看一下，工程的配置文件，它的配置文件，在resources的application.yaml，可以看到跟之前的配置，面相比，它增加了很大的一块，都是关于数据库连接的，在Spring.datasource底下，我们的连接是用的JDBC，所以第一个是，它的连接的驱动，是JDBC的驱动，当然驱动要能够用的话，需要在Maven的POM，中间去引入了，JDBC的Dependency依赖，第二个是JDBC的连接字，连到是我们现在数据库，是在本机，连到了本机的上面，因为我们现在本机的数据库，装的是MySQL8.0，所以它连接字的后面，我们会把跟，MySQL5.7不一样的地方，它会加了一个serverTimezone，这个是标识说，当前的时区是，我们设的是上海时区，北京的时间，然后是我们刚才设的，数据库的用户名和密码，后面的部分type的部分，我们不是直接去连的数据库，我们利用了一个阿里的连接池，什么是连接池，每次使用SQL语句，都会去跟数据库建立一次连接，发一条SQL，语句过去，然后它返回一个结果回来，用完以后，就会把这个连接，把它关掉，为什么会要这么做，原因是因为数据库的，可以支持的，同时连接数是有限的，因为它的同时连接数支持的越多，它的内存的开销也越大，我们一般来说，MySQL装好以后，没有去动它的话，它的同时连接数应该是在200，应该是100，还是200的数里头，每一个SQL语句的执行，它就会建一次连接，然后把它关掉，建一次连接把它关掉，当然我们知道这样是比较慢的，所以我们这里引进了一个阿里的，Druid的连接池，连接池是什么意思，就是它会去把连接，先建立好一些连接，然后在使用的时候，在使用SQL语句的时候，它不用每次都去建立和关闭连接，了，它从连接池里头，拿出一个连接过来，然后去运行这条SQL语句，这就是Druid的作用，我们还有两项，一个是schema，一个是data，这两个设置，分别指向了，两个SQL的语句，放在了resources，下的schema.sql，和data.sql，schema.sql，主要是放的建表，的语句，data.sql放的是，初始化数据的，这样的一个语句，后面我们可以看到，有一堆跟Druid有关的配置，其中有它的初始连接数，整个应用一起来，它就会最开始就会建三个，连接，它的最小的连接池的数量，因为它随着连接的减少，它会删掉这个，已经存在着连接，所以说我们定了一个最小数，是3，最大数是10，所以说它最多，会建10个数据库的连接，如果说它的操作语句，超过了最大连接数的话，我们就需要去设立一个，最大等待时间，一个60000的最大等待时间，这些参数是非常有意义的，如果说我们的应用，是一个，数据库的应用，实际上你把前面的，Servlet的，线程数设得很大，没有把数据库的同时的连接数，把它设大的话，其实这块地方就会是瓶颈，所以说我们要均衡的来设置这些，参数，要让整个系统的运转，是比较流畅的，如果我们想监控，Druid的连接池的情况，我们可以配置一个，stat-view-servlet，这样的话，它就会配置一个监控的网页，让我们去看，它的连接池的工作情况，我们可以给网页设置一个用户名，和密码，就是login-username和login-password，然后enable是把它打开，url-pattern是指定监控网页，的路径，它是在/druid/目录底下，后面的话是慢SQL的配置，慢SQL的配置，是在Druid中间，非常重要的一个配置，因为我们可以通过，慢SQL的日志，去发现我们的数据库里头，运行的比较慢的SQL，从而去优化这些SQL语句，现在设置里头，它已经打开了，慢SQL的日志，它的，log-slow-sql=true就打开了它的，慢SQL的日志，然后它的慢SQL的定义，是超过100毫秒的，SQL的语句，认为是慢SQL会写到日志里去，慢SQL的日志，我们是需要把它写到，日志文件里去的，在之前的工程里头，它已经是用了logback的日志，我们在之前没有去细讲logback，日志，logback是一个非常快的日志的，框架，它的配置文件，在resources底下的，logback.xml的配置文件中间，我们来讲一下，它logback的配置文件的，配置的方法，打开logback.xml的配置文件，我们可以看到，在，第一项里头，配置的是log所写的目录名，我们当前的测试代码来说，我们就配置在当前的目录底下，会建一个叫做logs的目录，但是如果你配置到服务器上去的，话啊，你应该知道我们的，这个是打成了一个JAR包，部署到服务器上去的，所以你需要去修改配置项，把配置文件写在那台服务器的，某一个目录底下，这样的话我们就可以，在外面能够看到，整个系统的运行情况，在logback的配置文件中间，它最主要的内容就是appender，appender其实就是用来去输出，日志的，在现在这个版本中间，我们可以看到，我们定义了三个appender，一个是标准的STDOUT appender，这个是用来去输出，到我们的标准输出上，就是我们在console上面，所看到的内容，这里头定义了它的输出的格式，以及它的输出的字符集，第二个我们看到的是一个FILE，输出到文件的appender，输出的文件的appender，定义的是按照时间的这个原则，按照时间的因素，来生成文件的，来生成日志文件的，这个appender主要是用来生成我们在系统，这一级的所有的日志，所以它是按照每天的方式，来生成一个文件，而且它的所有的日志，会保留30天，30天之前的日志文件，会把它删掉，后面同样定义了，它的格式以及定义来说，它允许的每个文件的大小，就是文件大小，我们这里的定义的是10兆，所以说大于10兆的话，它就会开一个新的文件，然后第三个appender，我们看到的是，关于Druid的文件的输出，它同样定义的是一个，时间的去滚动的这样的一个方式，但是跟之前的方式不一样，它在前面定义的一个file，我们其实并不需要，按照每天去生成一个独立的，慢SQL的日志文件，我们可能更感兴趣的是，到目前为止，我们看到的，所有的慢，SQL的日志文件，所以它前面定的file，它会把当前正在记录的，慢SQL的日志，在log里头，后面的rollingPolicy，它其实是会把，历史的记录放到后面去，按照它产生的日期的值，写到一个log文件里去，我们注意看一下，它后面rollingPolicy里头，我们定义的，timeBassFileNamingAndTriggeringPolicy，就是它会触发的机制，是，它的文件大小大于了20兆，所以大于20兆的话，就把当前的文件，会产生一个新的文件，放到目录底下去，而且我们这里没有去删除，它的这样的一个限制，同样后面是它的输出的格式啊，而且我们加了一个filter，它这个日志，就只记录ERROR级别的，这样的一个日志，最后的两个是我们定义的，真正输出的logger，我们可以看到，对于druid包来说，我们定义了，它会把它输出到，这个标准的输出上看，console上，会输出到druid的文件里头，我们又定义了另外一个，logger，609，它的输出级别是INFO，然后我们把默认的定义成为，在标准输出和文件上头去做输出，这就是我们的logback的，XML的配置，由于我们在POM文件，已经引入了logback的依赖，所以说当我们的，SpringBoot的程序起来以后，它会去读取logback.xml，里头的配置信息，然后把我们的相关的信息，写到日志文件里去，好，最后我们看一下，这个程序，跑起来是个什么样子，现在已经配置了数据库，和Druid的这样的一个部分，我们首先可以来做一个测试啊，跑完以后你会发现，因为我们在测试的test类的，前面，加了@Transaction，所以它跑完以后，你去看一下数据库里头的，跑的数据会发现，它跟跑之前的数据，是完全一样的，所以这是我们在Spring框架中间，关于数据源的这样的一些，设置，我们的数据源，使用的JDBC的方式，连接了一个MySQL的数据库，然后中间，通过了一个阿里巴巴的连接池，使得我们可以同时，保持多个数据库的连接，这样也提快，我们的这样一个速度，下面的部分我们可以看到，是关于MyBatis的设置，MyBatis的设置里头，我们已经在前面的课里头，已经讲到，过了这里头的一些设置，其中有一个设置，在默认情况下，我们是把它注释掉的，就是log-impl，这个设置的主要的作用，就是因为我们知道，MySQL，把我们对于MyBatis的操作，转换成为一个SQL语句，有些时候我们可能需要看一下，产生的SQL语句是个，什么样的，也许是我们SQL语句写的有问题，所以我们需要看一下，在运行的时候，SQL语句产生的是什么样的，我们可以把注释log-impl，打开，这样的话，它就会把所有的SQL语句，都输出到控制台上，我们回头来看一下POM文件，在POM文件中间，跟之前的没有MyBatis的POM文件，相比，我们增加了这么几个内容，第一，我们增加了一个PageHelper，虽然我们现在这个，还没有用PageHelper，PageHelper是一个用来做分页，当我们要查询多条的语句的时候，我们实际上做一个分页，这样使得不会一次从数据库里查，太多的这样的一个数据，第二个我们看到的，就是我们之前说的，MySQL的JDBC的，这样的一个驱动叫什么，mysql-connector-jara，然后是MyBatis，需要引入一个，mybatis-spring-boot-starter，我们在前面的课中间讲了，然后我们看到，我们用了阿里巴巴的Druid，所以我们在Dependency中间，引入了阿里巴巴的Druid，然后其它的就跟上一个的工程是，一样的，好，最后我们看一下，这个程序，跑起来是个什么样子，现在已经配置了数据库和Druid，这样的一个部分，我们首先可以来做一个测试啊，它跑完以后，你去看一下数据库里头的，跑的数据会发现，它跟跑之前的数据，是完全一样的，这个是因为，我们在它的测试类的定义的前面，加了一个@Transaction的注解，因为它是个测试类，这个注解会使得，测试类中间的所有的测试方法，都在一个事务中间去执行，而且当每一个方法结束的时候，这个事务会回滚，测试之前，数据库里的数据是什么样，测试，完成以后，数据库里头的数据，也会恢复成，原先的样子，第二个是我们把它，用SpringBoot的把它跑起来，SpringBoot把它跑起来以后，我们可以通过localhost:8080，去访问它，我们主要是去看一下，它的状态的监控的网页，所以我们输入，localhost:8080/druid/，在这个目录下，我们就能看到Druid的监控网页，这里我们可以看到，它的SQL的日志，以及它的慢SQL的日志，以及一些信息，当然你进到网页之前，是要输入你的用户名和密码的，就是我们在application.yaml，中间所配置的

# 6.1 缓存的作用

为什么需要缓存，我们知道计算机在不同的层次，上头，它的处理的时间是不一样的，在计算机系统中间，最快的是，CPU的一级缓存和二级缓存，读取CPU的一级缓存，仅仅需要0.5纳秒，如果说在CPU的缓存，读取的代码中间，包含了转移和分支预测的话，需要5纳秒，读取二级缓存需要7纳秒， CPU的一级和二级缓存，是整个系统中间最快的部分，但是它也是容量最小的部分，所以我们一般不会把，数据存储在一级缓存和二级缓存，里头，在一级缓存和二级缓存里头，主要存储的是代码，以及在当前正在计算的数据。

数据主要存储在哪呢，我们知道数据主要是存储在，先放在内存里头，内存中间读取1兆的数据，大概需要25万纳秒，内存也是有限的，我们也知道更多的数据，是放在通过网络，或者说放在磁盘上头的，在1G的网络上，头发送一兆的数据，和读取1兆的数据是差不多的，大概都需要1000万纳秒的，时间，而在磁盘上头，这个是SSD磁盘，和机械磁盘是不一样的，我们知道SSD的磁盘，速度，会比较快一些，但是SSD的磁盘，有一个问题，就是它会有寿命，它特别是写的寿命，擦除的寿命是有限的，机械磁盘来说，它的速度会稍慢一些，但是它的寿命会比较长，所以在我们的系统中间，我们特别是在用于，存储数据的数据库上头，我们往往还是采用了，机械磁盘的这种方式。

机械磁盘的话，它的时间主要分成两个部分，我们知道一个是磁盘的寻道时间，还有一个是它，读出和写入的时间，在机械磁盘上头，磁盘的寻道时间，大概在1000万纳秒左右，从磁盘上读出1兆的数据，大概需要3000万纳秒左右，一般情况下，头我们大概在磁盘上，读出1兆的数据，大概需要4000万，纳秒左右的时间，所以它的时间，其实是比网络的，传输的速度，还要稍慢一些。

从上面的分析，我们可以知道，由于数据库，是需要通过网络去访问的，因为数据库通常是一台独立的，机器，而它又包含了大量的磁盘操作，所以说当整个系统的，访问量非常大的时候，数据库就成为了整个系统的瓶颈，为了提高数据库的处理的性能，同时处理的性能，我们可以加大数据库的连接数，和数据库的并发数，同时处理数据的能力，但是我们知道，并发加大的数量是有限的，因为最终它还是受限于，单台计算机的计算的能力，内存的资源，以及它的磁盘，和网络访问的能力。

所以说当单台的数据库服务器，达到了一个瓶颈的时候，我们就会采用一些高并发，负载均衡，以及分布式数据库，这样的一些技术，来把单台的数据库服务器，变成多台的数据库服务器，或者在数据库服务器上，进行这种分库分表的，这样的一些操作。

但我们知道这一些的操作，需要花费很大的人力，比如说我们要去做分库分表的话，实际上在程序上，需要去做一些修改的，比如说，我们要去做负载平衡的话，实际上要花相当大的精力，在配置上头，同样又需要花费大量的资金，因为它是从一台数据库，变成了多台数据库，所以在硬件和网络上的投资会，增加，所以说在采用这些，我们常用的这种，提高处理能力的，这种多数据库，或者数据库集群的，这样的技术的之前，我们往往会采用一种，更加经济，而且有效的方式，来减轻数据库的负载，这就是缓存的技术。

缓存的技术，其实我们，在整个系统的各个地方，都是可以使用到的，我们的这样的一个系统，通常可以分为这么几个部分，前端客户端，前端，或者是用基于网页的，或者说基于APP的这样，的一个前端，然后如果说，基于网页的系统的话，通常还会有一个Web服务器（接入服务器），因为我们的一些静态的数据，静态的网页，静态的图片，或者是在Web服务器（接入服务器）上，在Web服务器后面，我们会有应用服务器，应用服务器是用来运行，我们的Java程序来，执行我们的业务逻辑的计算的，再往后头，才是我们的数据库服务器。

我们所说的缓存的概念，也就是说我们在处理数据的时候，不是说所有的数据，都要从数据库里头去读取，因为我们前面知道，数据库是，当处理量大的时候，数据库就是整个系统的瓶颈，我们在前面的这些所有的地方，都可以去做一些缓存，来减轻后面系统的压力。

我们从最开始的，前端的系统开始说起，我们在前端是用，如果说基于网页的系统的话，是用Javascript来写的，JavaScript可以去，把一些常用的只读数据，把它存储在，JavaScript的对象，中间，这样的话，它就不需要频繁的，向后端，去发出请求，去读取这些数据。

如果说是APP的话，当然它本身就是用Java语言，写的，它也能够很方便的，把一些常用的只读数据，存储在前端的应用程序里头，在Web层的话，Web层的话，我们通常的构造方式，是用Nginx服务器，和Varnish服务器，来做Web服务器（接入服务器），其中Nginx服务器，主要是用来处理静态的资源的，当然Nginx服务器，也可以把我们访问的数据，当做一个静态资源，存储在这个服务器上，我们称之为静态的缓存。所以说，当第一次访问这样的一个，请求的时候，它获得的数据，Nginx会存在磁盘上，之后再去访问的时候，它就会从磁盘上，把它读出来，然后直接返回给前端，这样使得后面的机器就可以，根本就不会起来，来处理这样的一个请求。

Varnish服务器跟Nginx服务器，类似，但是它不同的地方在于说，它的缓存不是在磁盘上，而是在内存里头，我们知道内存的速度，会比磁盘上去读取速度要快很多，所以说，Varnish服务器是一个比Nginx，服务器，做缓存更，快的一个解决方案，所以说，我们往往会用Nginx去处理，静态的资源，用Varnish去处理这种，我们称为动态数据的，这样的一个缓存，当然用内存做缓存，存在的一些问题，因为内存是有限的，所以Varnish里头，有设计了更复杂的，导入和导出，就是缓存的数据，把它导出和导入的，这样的一种机制，这个是在Web服务器（接入服务器）上做缓存，在Web服务器（接入服务器）上可以做的缓存，我们通常是一些，相对比较固定的数据，比如说一些枚举的信息，比如说我们前端的一些，状态值，因为通常来说，是不太会发生改变的，我们就可以把它缓存在，Web服务器（接入服务器）上去。

应用服务器，是我们做缓存的一个主力，应用服务器做缓存的方式，主要是有这么三种，一种是我们应用服务器，本身是可以做缓存的，比如说我们用Spring框架，来做应用服务器的话，我们可以在Spring框架中间做，一个Singleton Bean，Singleton Bean里头，存储了一些，数据存储的一些数据，这样的话，我们在每次要访问这些数据的，时候，就不用去数据库里去拿了，直接可以从Singleton Bean中间，去拿到这些数据，这是一个，最简单的应用服务器的缓存的，方式。

当然应用服务器本身，在使用数据库的时候，数据库的ORM框架，比如说，MyBatis或者Hibernate的，时候，它们都有提供了，一级缓存，和二级缓存的，这样的一种机制。

当然我们还有第三种方式，就是，当我们的应用服务器，不是一台的时候，我们希望在多台服务器的之间，去共享缓存数据，也就是我们前面的第一种，那种缓存方式，是每台应用服务器有自己的缓存，因为我们有自己的缓存的话，这个会带来很多的副作用，首先我们不说，它会占用，每一台应用服务器的内存，更主要的是这些缓存，如果要更新的话，其实是非常不方便的，因为它没有统一的存在，一个地方，可能有些应用服务器的缓存被，更新了，有些就没有更新，这样会造成我们的，数据的错误，所以说对于这样的一种，有可能会被更新的这种缓存数据，我们更多的会采用，我们的第三种方案，就是用Redis的，分布式的缓存技术，为所有的应用服务器，提供一个统一的缓存服务器，然后我们把这些缓存的数据，都放在Redis服务器上去。

Redis的服务器的，缓存，是通过内存来做的，通过网络去访问，通过内存来做的，所以我们从最开始的分析来说，它会比我们通过网络，去读取磁盘上头的数据，通过网络去在磁盘上读取数据的，这个速度要更快一些，所以从前面的分析上来看，我们可以在整个系统的，各个部位，都来做缓存，我们做缓存的这样的一个依据，是这样的，通常来说，如果说只读的数据，就是或者改动，基本上是没有的数据，我们会越往前靠，也就是说，我们会把这些，只读性比较大的数据，放在前端的缓存上头，或者说放到Web服务器（接入服务器）的缓存，上头，当然我们也可以放在应用服务器，的，每一台应用服务器的内部的，数据结构上，因为它修改的可能性不大的话，其实就不会存在我们前面说的，应用服务器，自己缓存的，这样的一个副作用，越往前靠的缓存，它对于整体性能的提升，是非常明显的，越是，会修改的，有可能修改的这个东西，我们就会越往后放，比如说我们的Redis的缓存，放的就是一些，有可能会被修改的数据，对于那些会频繁修改的数据来说，我们实际上是没法把它放到缓存，里的，那些数据只能依靠数据库，来进行处理，所以说通过使用缓存，我们可以看到，我们把很多读的，这个数据，把它往前面提，从而减轻了后端的数据库的压力，从而使，整体系统的性能

# 6.2 MyBatis缓存

MyBatis缓存，我们在前面提到过，在应用服务器上头的，ORM框架MyBatis，它提供了一个，数据库查询结果的，缓存的，这样的一个功能，它可以把前面的数据库的，查询的结果，放到缓存里头，下次就不用再去查数据库了，从而降低了数据库的负载，特别是读的负载。

那么MyBatis提供了一级缓存，和二级缓存，两种不同的缓存，一级缓存的话，是在单个的事务中间，会把这个数据结果缓存下来，而二级缓存的话，是在整个应用服务器的级别上头，只要之前的查询，有查过这个数据，这个数据就被缓存到，MyBatis的数据库的，缓存里头来。

那么MyBatis是怎样知道，说查询是之前查过的呢，MyBatis认为说，以下的三个条件是完全一样的，它则认为它们是相同的查询，第一是查询的ID，就是我们在这个XML中间，以及在mapper的接口中间，所定义的，方法名，以及在XML中间所定义的ID，首先ID是一样的。第二个ID是一样的，因为我们里头，可能会写动态的SQL语句，所以它的第二个条件，是去判断说，这次查询所产生的，这个预编译的SQL语句，要是一样的。第三个它是去判断，要绑到SQL语句上头的，查询条件的值，参数值要是一样，ID一样，预编译出来的SQL语句一样，以及查询的参数值一样，MyBatis框架就认为说，这是同一个查询。

它会用一个算法，把这三个，条件生成一个键值，然后以键值和结果，存到一个HashMap里头来，如果说，后续的查询，算得的键值是完全一样的，它就从HashMap中间，把这个数据取出来，返回给应用服务器，这样就不去查数据库了。

当我们有执行的update，insert，和delete语句的时候，它都会去清空，缓存，从而使得，会重新执行数据库的查询。

我们前面说过，它有一级缓存和二级缓存，一级缓存是在一个事务里头的，所以开启事务以后，在事务里头，只要有查过的结果，查过的语句，就不会再送到数据库里头，再去做查询了，那么MyBatis，是怎样来实现一级缓存的呢，我们知道我们在前面讲，MyBatis，在上一次讲MyBatis，讲到MyBatis是靠SQLSession，是一个SQL的会话，来把我们对于MyBatis，框架的，接口的调用，转成了SQL语句，在SQL的会话中间，SQLSession中间，它有一个Executor，它就是具体负责，我们来把接口的调用，变成，在XML中间的，SQL语句的，这样的一个映射的机制，Executor里头，是有一个LocalCache的本地，缓存，所以说当用户发起查询的时候，MyBatis的SQLSession，会去生成，一个我们mapper的statement，就是我们按照前面的三个条件，生成，按照它XML中心定义，生成了一个SQL语句，然后按照我们前面的三个条件，到LocalCache中间去找，说这样的一个查询，是否有执行过，如果说有执行过，就直接从LocalCache中间，拿出来返回给客户，如果没有的话，就会到数据库去查，把查的结果返回给客户，而且同时会存到，LocalCache。

这样的一个SQLSession是，在一个事务中间去建立的，所以每一个，是在一个事务中间去建立的，我们前面说到过每一个事务会，开启一个，SQLSession事务结束了以后，就会把SQLSession关闭，所以这样的一个缓存，就只一个事务里头，所以我们称它为一级缓存，不开启事务，实际上一级缓存就是不存在。

二级缓存是什么，二级缓存是在一级缓存的基础上，来实现的，二级缓存，我们把它称之为，mapper级的缓存，也就是它的作用域，是一个namespace，mapper中间的一个，namespace，我们前面注意到，在每一个mapper的XML中间，我们都定义了一个namespace，所以在每个，namespace上头，如果说一个不同的SQLSession，执行的查询语句是相同的，它则会把上次的查询结果，提取出来，直接返回给前面，从而更大幅度的，减少了数据库的查询。

在MyBatis中间，二级缓存，是怎样来做的呢，我们知道在MyBatis中间，它其实是做了一个装饰器模式，也就是在我们的SQLSession，的前面，就是说它的后面还是一样，和前面的一级缓存做法是一样的，它是通过SQLSession，完成我们的对于接口的调用，到SQL语句的，这样的一个映射的，在它之前面它放了一个叫做，CachingExecutor的，就是在SQLSession的Executor的前面，它放了一个叫做，CachingExecutor，的装饰器的对象，所以说当一个查询，由SQLSession来执行的时候，它首先会要过CachingExecutor的，CachingExecutor的首先是在它的，namespace，级的一个缓存里头，去进行查找，去找说有没有之前，在这个namespace里头，有做过相同的查询，如果没有的话，它就会交给我们前面，在一级缓存中间提到的Executor去做，查询，在Executor中所做的操作，就一级缓存是完全一样的，如果说从Executor的返回来的结果，它又会放到CachingExecutor会放到，namespace的，二级缓存里头来，从而当下一次，在另外一个事务中间，它去执行查询的时候，CachingExecutor，就会从namespace中间，把前一次查询，所缓存下来的数据，直接返回回去，这是二级缓存的，这样的一个实现机制。

我们知道在一个事务中间，其实我们极少会把同一个查询，语句执行两次，所以一级缓存，对于我们的一般的应用程序来说，它的意义其实不大的，二级缓存的意义是比较大的，但是由于我们看到，MyBatis二级缓存，是在应用服务器来这一个级别来，做的缓存，也就是不同的应用服务器，其实会有做不同的二级缓存，而我们的通常来说，大负载的系统来说，我们是不止一台服务器的，所以这样的一个二级缓存，MyBatis二级缓存，其实没办法在，应用服务器的集群的层面上，来做缓存，这是它的第一个的缺点。

第二个缺点是二级缓存，其实是没有区分的，来做缓存，只要是对数据库的查询，它其实都会把它缓存下来，不管以后有没有会重复用到的，可能性，我们知道，这样的一种不加选择的缓存，其实会占用大量的内存空间，而且因为内存是有限的，所以它一定会用某种算法，把不需要的数据给它清除出去，所以这样也会降低了，这样的缓存的一个效果，因为缓存所用到的内存，是一个非常有限的资源，所以我们在做，应用服务器级的缓存的时候，我们往往希望，有一种更节约的方式，或者更加有效的方式，来使用这些有限的内存，这就是我们在这一个层级上头，更多的会采用，基于Redis的这样的一种，分布式的缓存的方式，来由自己控制来实现缓存

# 6.3 Redis缓存

Redis是一个开源，的，高性能的值键对，是内存数据库，它的主要的用途，它可以当做一个数据库来使用，但是它最主要的用途，是用来做缓存，作为缓存的Redis，它不但支持，把数据放到内存里头，让其它的应用服务器高效的访问，而且它还支持，把内存中间的数据，来进行持久化，这样的话，一旦万一系统掉电的话，它的数据其实是，已经被存储在磁盘上的，重启的时候，可以再次把数据读取起来，再使用。

Redis不仅仅支持我们常用的，Key-Value类型的值键的数据，其实它也可以把一个复杂的，数据结构存到值里头，比如说它支持List结构，集合的结构，以及排序集合和Hash，这样的一个，数据的结构，同样它也是一个分布式的缓存，数据库服务器，就是一个分布式的缓存服务器，它可以支持多台服务器的部署，支持主从的模式，就是我们往一台服务器上去写，可以在多台的Redis服务器上头去，读。

Redis的服务器，总体来说，它的性能是非常高的，因为它的数据，主要是存储在内存里头，我们在Redis上头，读的速度是每秒钟可以读，11万次，写的速度稍慢一点，但是每秒钟，也能写81,000多次， Redis的所有的操作，都是原子性的，所以我们在Redis的操作，做的时候它都是互斥的，当然它的所有的操作，也支持事务，我们可以把Redis的操作，放到一个，有Spring框架管理的事务里头，跟我们数据库的事务，放在同一个事务里，如果说在这个事务中间，有任何的动作失败了，它就会全部回滚回去。

Redis我们前面说过，它是个值键的数据库，它的值的部分，它支持5种数据类型。最常用的是这个字符型，也就是我们基本上，可以把所有的数据，把它，用JSON做序列化以后，把它变成字符串，存到 Redis的数据库里，去Redis的缓存里头去，这个字符串的类型，它是二进制安全的，而且它的读取的性能是非常快的，但它的缺点是在于说，它的有一个序列化，和反序列化的过程，因为你所有的，不管是什么类型的对象，你都需要把它序列化成字符串，或者从字符串反序列化成对象，所以这是有时间开销的。另外因为它把它序列化成了一个，字符串，所以说你试图去更新其中的一个，属性，其实是很不方便的，因为你要更新它的一个属性，其实你要把它再反序列化出来，然后在对象上的改完，再把它序列化成这个字符串，其实这是非常不方便的，所以对于这种字符串型的，这样的一个值键对的话，我们在Redis中间的做法，是通常直接删除，不会去做更新，因为你去做更新，其实跟把它删掉，再重新放一个进去的速度，其实是差不多的。

Redis的另外一种数据结构，就是Hash字典，Hash的这种结构适合直接把对象，存储到Redis中间去，因为它不像前面的字符串，是需要做序列化和反序列化的，所以说在这样的，直接把对象存到Redis中间去的，这样的一种Hash结构中间，我们可以在Redis中间，直接去修改，对象中间的某一个属性，所以说，你如果有要修改，某一个属性值的这样的需求的，时候，你适合把它作为Hash，把它存进去，相比字符串来说，虽然它可以直接修改属性，但是因为它存的是一个更复杂的，结构，而不是一个简单的内容，所以说它占用的存储空间，会更大。

除了字符串和Hash这两种用来存，对象的，这样的数据结构以外，Redis还支持可以，更复杂的结构，存到这个值里头，第一，它支持链表，链表，就是它会把对象，按照一定的顺序，把这个顺序，其实双向链表的，这样的一个结构，按照顺序存到值里头去，这样的一个顺序的链表，它可以支持对于链表的，增加和删除，而且提供了快速访问，比如说某一个，这个元素，或者某一段元素的，这样的一些API，这样的一个List的结构，我们可以在上面做很多的功能，比如说我们可以做，排行榜，就可以用List结构，我们甚至可以用List结构，去做一个简单的，这个消息服务器，做个消息队列。

第二个数据结构是集合，就是里头的元素是不能重复的，所以它可以，在集合的基础上头，去做集合的，交集、并集和差集这些操作，同时对于集合里头的，每个元素的访问速度是非常快的，是 O（1）的时间，复杂度的，这样的一个结构，利用集合来说，我们可以做一些，比如说，独立IP统计，就是去统计，从每一个IP来访问这个系统的，次数，有多少利用它集合的唯一性。

集合是可以做排序的，所以它有第三种结构，叫做说有序的集合，在这种排序的集合上头，每一个元素，其实它有一个参数Score，所以集合中的元素，会按照这个Score的值，来进行排序，当我们把一个元素带它的Score，插入到这样的一个排序，集合的时候，它就已经天然排序了，所以是我们可以很方便的

# 6.4 缓存有效期和淘汰策略

缓存的有效期与淘汰策略，我们知道Redis的缓存，是非常有限，所以我们实际上，是无法把所有的数据，都放到缓存里头来的，当我们把这个数据，放到缓存里的时候，我们一般是要给数据，设定一个有效期的。指定说这个数据，会在缓存里，放多久的时间，当这个数据到期了以后，就要从缓存里头，把数据给它清除掉，以腾出空间来，让新的数据，放到缓存里头来，这个就是我们称的，缓存的有效期。

淘汰策略，则是指说，当这个缓存中间的，所有的数据，都满了的时候，我们需要用一个什么样的方法，把缓存里头的数据给它拿走，对于缓存的有效期和淘汰策略的，话，我们通常有这么几个方法，来去清理缓存里的数据。

第一个来说是定时过期，这也是一般大家都会想到的一个，方法，当我们把一个数据，放到缓存里去的时候，因为大部分的数据，其实都是有有效期的，我们可以去为每一个数据，设定一个定时器，等到时间到了以后，就把这个数据，从缓存里头清除掉，我们知道这种策略的话，内存的清理是非常干净的，因为数据一到期就会被清理掉，但是这种做法来说，其实是不现实的，因为我们在缓存中间，会有非常多的数据，每个数据放进去的，缓存的时间不同，它的过期的时间也就不同，所以每一个数据，都需要一个定时器去，把它清理掉，这样的话，会大量的占用CPU的资源，从而使得我们，用来做处理缓存的，这样的一个，计算资源会受到影响，所以一般来说，我们都不会用这样的一种，定时过期的方式，来清理缓存里头的数据。

另外一种方式就是惰性过期，惰性过期的话，它是只有当你访问到一个数据的时候，它会去判断一下，这个数据的有效期是否已经到了，如果到了，我们就把这个数据清除掉，如果没到的话，这个数据拿出来给你访问，我们可以看到这样的一种方式，其实它是完全不需要定时器的，所以说它不会占用CPU的资源，但是这样的一种方式，对于内存来说是非常不友好的，因为除非你访问到这个数据，它才会被清理，如果说没有访问到这个数据的话，它就会一直呆在缓存里头，极端的情况下，头我们有些数据已经过期了，但是它从来没有被访问过，所以它就会一直待在缓存里，不清除，这样就造成大量的缓存被占用了。

第三种方式就是，我们称之为定期过期的，这样的一种方式，定期过期的方式，实际上是我们设定一个间隔，每隔一段时间，去扫描一下缓存里头的数据，把过期的数据给它拿走，这样的一种方式相比，惰性过期来说，其实它也是一种主动清理的过程，但是和定时过期相比，它可以设定一个时间间隔，所以这个时间间隔的设置的长短，也就决定了说，对于CPU的占用资源会有多少，同样它其实也可以设定说，它每次扫描的时候，只是扫描多少数据，所以间隔和扫描的数据，这两个值决定，可以用来去平衡我们的，CPU的资源的占用情况，以及我们想对内存清理的，这样的一个程度，我们知道定期过期的这种方式，其实它并不是及时清理的，所以有一些数据，过期了，它还会在缓存里头待着，等到它到扫描的时候，被扫描到了，它才会被清理。

Redis采用的，其实是惰性过期，和定期过期结合的方式，Redis是每隔100毫秒，就会去默认情况下的，每隔100毫秒，就会去扫描一下缓存，如果发现有过期的，这个值的话，就把它拿掉，当然它在100毫秒中间，它其实是不会去扫描所有的数据，的，因为缓存里的数据，有多有少，而且本身缓存可大可小，所以说，它去全部扫描缓存的数据的话，其实会占用比较多的时间，所以Redis采用了，这种，随机抽取的方式，抽取值也是可以设定的，它抽取这些值，随机抽取一些值，把这些抽取的值，中间，有过期的数据给它清理掉。

这样的一种方式，我们知道，还是会造成很多的数据，如果没有被抽取到，它就会呆在缓存里头，所以Redis也采取了，惰性过期的方式，当你访问到一个值的时候，发现这个值过期了，它就会从缓存里把它清理掉，所以结合采用了这样的一个惰性，过期和定期过期的方式，能够以尽量少的CPU资源，来清理在，缓存中间的数据，这是Redis有效期的管理。

不管怎么样去管理有效期，因为内存缓存总是有限的，所以说它总会有一天，到缓存里头的所有的的空间全占，满了，但是有一个新的数据要放进去，Redis碰到这种情况的时候，它就会启用，它的内存淘汰机制，缓存淘汰机制。

反正它其实有这么几种方法，最简单的这样的一个方法，就是，我们叫做noeviction这种方法，就是如果说，缓存不够了，放进去数据它就会报错，但是这种方法我们知道，我们一般不会采用，更多的我们会采用一种算法，从所有的数据中间把清理掉。

我们的算法主要有最少使用，最近不常使用，以及随机的算法，针对的数据有两个范围，一个范围是所有有有效期的，数据，我们称之为，volatile的，这样的一个有有效期的数据，而另外一个就是所有的数据，我们称之为allkeys，所以当我们去设定Redis的，内存淘汰机制的时候，如果我们设定的是allkeys-lru，就是用LRU的，算法，就是最近最少使用的，最近，最少使用这个算法，去把所有数据中间，最少使用的，数据给它清除掉，如果是allkeys-random的话，就是随机的从所有的数据中间，拿一个数据把它清除掉，把它拿走，volatile-lru就是用，LRU算法，在有有效期的数据中间，拿到一个数据把它清理掉，volatile-random就是用，随机的方法，在有效期的数据中间把它拿掉，把它淘汰一个。对于有有效期的数据来说，它还有一种自己的算法，叫做volatile-ttl这种算法来说，是取有效期过期时间，最早的数据给它清理掉，这样的话可以保证说，如果是有过期数据的话，它的过期时间肯定是最早的，如果没有过期数据的话啊，就是说，最近会要过期的数据，就会把它拿掉。在Redis 4.0以后，它提供了第三种算法，叫做LFU是，是找最近最不经常，使用的，这样的一个，数据把它拿掉，对于所有的数据是allkeys-lfu，对有效期数据，叫做是volatile-lfu，我们用两个例子来讲一下。

LRU和LFU的差别，LRU是去找最近，最少使用的，LFU则是去找最近，使用的频率最少的，LRU的话，它是把最近访问过的数据，插到队头时，它会最晚被淘汰，LFU则是，按照它的使用频率来进行排序，所以把使用频率最小的数据，把它清理掉。

我们用一个例子来看一下，这两种算法的运行的状况，是个什么样的状况，假如说我们有5个数据，已经在队列里头，分别是ABCDE我们首先，用LRU的算法，LRU算法，假如说我们第一个数据，被访问到的是D按照，LRU的算法D，就会被移到了队头，A到了第二位，随后我们访问到了数据，B就会到队头，那D，就到第二位，A就到了第三位，第三个数据，F这个F在队列中间是没有的，意味着我们要从，对中间淘汰一个数据，所以按照LRU算法，它队尾的E数据，因为它在最近这段时间，都没有被使用过，所以说它就被移出了队列，F这摆到了队头，所以这样的队列，就剩下了FBDAC，这样的一个顺序，其中FBD是我们最近访问的这，三个数据，这是 LRU的算法，我们知道LRU的算法，其实存在着一个问题，它其实只表现了数据，最近访问的次序，其实它没办法去，表现出来数据访问的频率，我们其实更加关心的是，最近访问的数据在未来，如果还被访问的话，它应该留在队列里头，而不被淘汰掉，所以说，如果突然在短期，有些新的数据被涌进来，它就可能会把我们之前，可能用的比较频繁的这些数据，给挤出队列里头去，所以我们就会有LFU的，算法。

LFU算法跟LRU不，一样，它不是简单的用它访问的，先后来进行排序的，而是把所有的数据，记了一个访问次数，比如说我们还是ABCDE，这5个数据，但是它们的访问次数，分别是32 30 26 26 25，在队列中间，它们就按照这个顺序，来进行排序的，当D数据首先访问的时候，D的访问次数就会被加一，就变成了27，所以D就会在整个队列中间，往前提了，一位，排在了 C的前面，第二个数据B被访问的时候，它的数据也加了一变成了31，但是它没有超过A所以说它，依然会在第二位，F进来的时候，因为F是在队列中间没有的，所以它的次数是1，它不会像LRU排在队头，而是按照顺序会排在了队尾，而把E淘汰出去了。

所以这样的一种算法来说，我们可以看到，它按照访问频度去排序的话，就会使得频度访问较小的数据，给它排除出去，当然我们知道这有一个问题，第一个问题就是，你们看到，F进来以后，它就被排在了队尾，如果下次再进来一个新的数据，就会很快把F给淘汰出去，所以LFU的算法，对于新数据来说是不友好的，通常来说，我们会把新数据，给它做一个初始值，比如做个5或者6的初始值，它不至于说，一进来，就会很快被淘汰出去。第二个是，它对于老数据是非常友好的，比如说A虽然在我们这一段，时间里，头，都没有访问，但是它依然在队头，原因是因为它之前，被访问的次数比较多，所以它一直是排在队头的。所以LFU还需要做一个定期，淘汰的，就定期衰减的这样的一个做法，每隔一段时间，把所有的数据的次数，都减去一个值，这样使得老数据，也会在对中间慢慢的被淘汰，这就是LFU的算法。

在Redis中间我们可以看到，如果说它简单地采用，LRU和LFU的话，都需要维持一个排序队列，或者是记它的访问的先后，或者是按照它的频度去做排序，这是需要占用很大的，内存空间，和CPU的资源的，而且每次数据的访问，都会造成队列的改动，所以Redis，采用了一种近似的方式，它就不去维护这样的一个队列，而是从所有的值中间，去随机选取了N个，然后按照它的，最近访问的次数，或者最近的频度，去进行一个排序。

这样的话就使得说，我们并不是需要每一次，对于数据的访问，都会去，做一个排序的动作，而是当内存的空间不够了，在发生淘汰的时候，我们才需要从所有的数据中间，提取N个，对N个来进行排序，把按照LRU或者LFU算法，把队尾的数据给它拿掉，新的数据，其实也不需要记住，它在队列中间，在什么位置，就直接放进去就好了，所以这样的一种算法，我们可以看到，它比我们在理论上，或者LRU和LFU的计算，的，这个量都要小很多，但是它也可以近似达到，LRU和LFU，的效果，而且它的资源的控制量，其实是由N值来控制的，你提取多少值，随机选取多少值来作为排序，这个就会直接决定说，你会要用多少资源来运行，LRU和LFU N越大，它的效果，越接近于理论上的LRU或者LFU，N越小，所以它就离理论值越远，但是它对于计算资源的消耗，也就越小

# 6.5 缓存的雪崩和穿透

Redis的，雪崩和穿透问题，我们知道Redis是不可能，把所有的数据，都缓存起来的，所以我们前面讲到，Redis是需要对数据，设定，有效期的，无论是采用惰性的删除方法，还是定期的删除方法，这些有效期的这些数据，都会被从内存缓存中间清除掉。

这样的一种有效期的，这样的一种方式，可能会造成我们的一个副作用，副作用是什么呢，以我们的电子商城为例来说，我们知道用户通常来说，会在一个特定的时间，去开始访问这样的一个系统，在这个时间的话，所有访问过的商品，就会被load到缓存里头来，从而使得之后的访问，不再去访问数据库，降低了数据库的负载，这是我们Redis的主要的一个，提高系统性能的方式。

但是我们可以看到，如果说这个商品，在一个集中的时间，被load到了缓存里头，因为我们对于所有的商品，设定的有效期是一样的，这样也就意味着，它在一个集中的时间会失效，所以说我们可以看到，如果发生这样的一个情况，在某一个特定的时间，这些商品就全部都失效了，所以所有的请求，都会到了数据库，需要读数据库，把这个商品信息读出来，再放到缓存里头，这样就在一个集中的时间，对于数据库产生了一个，集中访问的这样的压力，这样的一个情况，我们称之为缓存的雪崩。

怎么解决这个问题，其实也非常的简单，就是我们避免，让缓存中间的数据，同时过期，所以虽然我们会给每一个商品，设定一个有效期，一个统一的有效期，但是我们会针对每一个商品，去加上一个随机值，这样就大幅度的减少了，它们同时过期的这样的一个可能性，从而降低，缓存雪崩产生的可能，这是第一个问题。

第二个问题是缓存的穿透，缓存的穿透是这样的，我们知道，在整个系统中间的，瓶颈和短板是数据库，所以如果说一个黑客，想攻击我们的系统的话，它肯定是要攻击我们的短板，当然一个方案就是它攻击数据库，让我们的数据库的负载，变得非常的大，然后使得整个系统的速度，变得非常的慢，怎么来攻击我们的数据库，我们在数据库前面，通常会做一个缓存，使得所有的访问都会，使得很多访问，都会在缓存里的就完成，

而不会到数据库里头，但是我们有一种特定的情况，是一定会去查，数据库的，这种情况是什么呢，这种情况就是在缓存里头，找不到这个对象，它就一定会去查数据库，所以黑客就会利用这样的一个，特性，始终向我们发出一个请求，而这个请求，在缓存中间是找不到的，怎样去发出一个请求，在缓存期间找不到呢。

我们可以想象一个这样的情况，就是我们在所有的，电子商城系统中间，都会有一个最简单的功能，就是根据id，去，把一个商品的详情拿出来，然后返回给用户，当然这是一个非常高频访问的，请求，所以说它一定会把拿出来的商品，放到缓存里头来，但是我们知道黑客如果说想通过，请求，去攻击系统的话，它的做法就是，它会不停地去请求一个不存在的，id，当它请求一个不存在的id，会发生什么，首先id对应的商品，在缓存中间没有的，所以它一定会去查数据库，但查数据库的话，它查到的是一个空，在数据库里，也不会有这样的一个商品，所以说，它也不会把这样的一个东西，放到缓存里头来，所以黑客再次来访问，这样的一个请求的时候，它又得到数据库里去查，虽然它查出来还是空，但是它也依然不会放到缓存里头，来，所以这样就会使得，黑客的每一次请求，都访问到了数据库，从而对数据库产生了压力，这就是我们称的缓存的穿透。

解决的办法是什么，解决的办法是，两种方案，一种就是我们采用布隆过滤器，布隆过滤器是一种非常特别的，这样的一个数据结构，数据结构有两个特征，第一，它能够存储大量的id，第二，它可以快速的去判断，某一个id的值，是否在过滤器中间存在，当然它有一定的误判率，但是它的速度是非常快的，所以我们可以在查询的前面，先加一个布隆过滤器，把我们系统中间，所有有的商品的id，放到布隆过滤器里头，先通过布隆过滤器，去看id，是不是我们系统中间，有的一个商品，如果有的话，才去到缓存和数据库里查，没有的话就直接把它挡住，虽然会有一定的失误率，但是它能够有效的解决，黑客去查询不存在的商品，这样的一个请求。但是布隆过滤器，也有它的缺点，它的缺点就是，它会占有一定的空间，因为要把所有的id都记住，这个是会占一定的空间的。

我们还有第二个方案，第二个方案就是，我们每次从数据库里查出来的，东西，如果是空的话，我们会把空的值和id，存到缓存里，存一段时间，这样的话，当黑客再次用 id去发出请求，的时候，它就会被缓存拦住，而不会再去查数据库，这就是我们来对付缓存穿透的两种方法。

# 7.1 AOP的概念

高内聚低耦合，是面向对象设计，所追求的终极目标，但其实我们无论怎样设计，都没办法解决，这样的一些耦合问题。比如说在我们系统中间的权限，校验和写日志的问题，这些功能，它像沙子一样的，被掺到了我们的系统中间，因为系统的很多地方，都会要调用它，所以说你没办法，把这些沙子，从系统中间把它挑离出来，就是系统会跟这个校权限，和写日志，紧密的耦合在一起，如果说我们想把系统的主要的，业务，和这些校权限写日志，这些非常散的这些小功能，给它隔离开，用我们现有的程序设计的方法，无论是面向对象的方式，还是面向过程的方式，其实都是没有办法把沙子挑干净的，我们最后还是要留下一个点，那个点，调用权限，和调用日志的地方。

面向切面编程，也就是AOP编程的，这样的一种方式，其实是能够解决这一个问题的，这样的问题的，最有效的方式，它能把这些沙子，从整个系统中间，给它剥离出来，使得系统中间，只剩下我们的，主要的业务过程，没有这些沙子，没有这些沙子，这些沙子，可以定义在一个独立的地方，然后用一个定义的方式，把它在系统运行的时候，或者在系统编译的时候，再跟我们的系统给它耦合起来，那这样的一种方式，就彻底把这些，我们的琐碎的散的这些小功能，遍布到处都是的，这些散功能，和我们的系统的主要的功能，给它完全分开了，这就是我们所说的AOP，技术，或者叫做AOP编程技术。

AOP的这样一种编程的技术，其实它有两种实现方式，一种实现方式，是用一种新的编程语言，比如说AspectJ还有一种方式，就是在现有的框架上，去实现AOP的这些概念，不管是用哪一种方式，不管是用新的编程语言，还是在现有的编程框架，去实现这些AOP的概念，其它都会用到AOP的一些术语，这些术语，在编程中间我们都会碰到。

所以我们先要介绍一下，这些术语的含义是什么，首先第一个我们要讲的是Advice，Advice其实就是我们的沙子，也就是我们，插到那些主业务里头去的那些，零散的东西，比如说日志权限怎么做，它要做什么。

当然除了沙子以外，它还有在什么时候做，所以Advice里面包含两个内容，第一个是要做什么，第二个是什么时候做，什么时候做，主要是指它是在沙子的，在那个点上头，在之前做之后做，或者前后都要做，或者是在那个点抛错的时候才做，这是指做的时机，第二个概念是JointPoint，JointPoint，其实就是我们掺沙子的点，所有在系统中间要掺的地点，我们就叫做叫JointPoint。

大家知道JointPoint是非常多的，因为我们掺的是沙子，所以我们通常需要一个定义的，方式，去定义这些点是什么点，我们称这个叫做，PointCut叫做切点，它定义了一个，或者多个JointPoint，它的连接点，多个JointPoint，通常它是用一种通配符的方式，定义条件的方式，来定义这些JointPoint。

第4个是切面，就是Aspect，Aspect其实就是包含了，我们前面所说的2个概念，我们的Advice要做什么，在什么时候做。我们的PointCut，就是定义了哪些沙子的地方在哪，它的所有的切点，它的所有的连接点，在什么样的地方。

Advice和，PointCut合起来，就构成了我们的一个，Aspect，里头包含了我们说的三个内容，第一，点在什么地方，第二，这个点的什么时间，要做事，第三是要做什么啊，其中后面两个是在Advice，定义的，第一个是在PointCut的定义的。

第5个我们要讲的概念就是，Target它就是我们要掺沙子的，对象，所以AOP的，Target中间，它的目标是它要没沙子，它只有主动的业务，而把要掺进去的东西，交给了Aspect去定义，最后在编译，或者在运行的时候，再把这些沙子，掺到我们的主体系统，中间来，这是AOP的一些基本的概念和他的术语

# 7.2 Spring中的AOP

Spring中的AOP，Spring框架是怎样来实现，AOP的，Spring框架主要是通过动态代理机制，来实现AOP的。

我们知道，Spring框架，最主要的概念就是，它会创建和管理Bean对象，我们在前面也谈到过，它在创建Bean对象的时候，其实它不是直接把Bean对象，注入到我们的使用的地方，它注入的是一个动态代理，所以它正是利用这样的一个，动态代理的这样的机制，来实现我们的AOP的一些功能。也就是它注进去的，既然是动态代理，所以说它就可以在动态代理上面，去根据我们AOP Aspect，中间的定义，去把一些沙子，掺在动态代理上头，这样也就使得在Spring中间，AOP的切点，主要都是在方法上面。

AOP的框架中间，我们首先要定义一个切面，Aspect，它并没有定义一个新的语法，来定义切面，而是利用了我们类的语法，所以类和类里头的属性方法，都是可以用的，它通过了这样的一个注解，来定义说，这个类定义的是一个切面，注解，就是@Aspect，且在类的定义，前面来定义，这个类是一个切面。

在切面里头我们前面说了，它定义两个东西，一个是PointCut，一个是Advice，PointCut的看的其实是一个条件，就是到底哪些连接点，是我们要掺沙子的地方。

它用一个@Pointcut的，注解，来定义切点，它的条件就写在，注解的属性里头，但是这个注解，是要放在一个位置上头的，要么放在属性前面，要么就放在方法前面，我们通常把注解，放到一个方法前面，这个方法里头是没有内容的，这个方法的作用，就是为了去定义这样的一个切点，这样的一个Pointcut的注解。

Advice它其实就是一个方法，同样它是用一组注解来定义，Advice的执行时间，然后方法里头的内容，只是Advice要做的，这样的一个事情。

我们首先来看一下，切点pointcut的定义，切点的定义，我们前面说过，它在定义在一个方法前面的，它在定义切点的时候，它用的AspectJ的定义符，但是由于我们说，我们的切点，主要是在方法上头，所以我们最常用的，其实是4个定义符。

一个是args()，表示说，方法的参数，应该是有特定的类型的，一个是@args()表示说，这个方法的参数要加了特定的注解，就是第一个是说，方法的参数类型，是这个类型，它就满足说它是我们的连接点。

第二个是表示说，这个方法上参数，如果也加了这样的特定的注解，这个方法就是我们的连接点。

第三个就是execution()表示这个方法，本身，方法的名字，符合一定的条件，那就是我们的连接点。

当然还有最后一种就是的，@annotation()表示这个方法，前面加了一个特定的注解，这就是我们的连接点，其实我们最常使用的。

我个人最喜欢使用的，就是@annotation()，因为我不太喜欢用条件，去隐含的去定义说哪些是连接点，而是喜欢用注解，用注解去显式的，说明说这个地方，我要切一个沙子进去，所以我比较喜欢用第4个，@annotation，用注解来定义。

我们看两个例子，第一个是用这个方法名字，所以我们在@Pointcut里头，定义的是一个execution的，定义符，用引号来包含的，execution表示说，它是在restdemo.service，底下的所有的，方法，不管它参数是什么，它都会要掺沙子，都掺着这个东西，我们可以看到，它的方法里头，是没有内容的，因为这个方法，主要是作为一个载体，用来在前面去定义注解的。

第二个注解，其实是定义的是注解，我们定义了一个注解叫做@Audit，所以，有加这个注解的，就是我们要掺沙子的地方，这是我比较喜欢用的一种方式，

在Advice前面，我们用5个Advice的注解来标识，这个方法是一个Advice，这个注解其实标识了，说Adivce在什么时候，来执行。我们第一个是@Before表示说，在JointPoint执行之前，就是方法执行之前，@After是在之后方法执行之后，@AfterReturning是在这个方法，成功执行完成之后，在结果返回之后，第4个是@AfterThrowing，就是它有抛出了，某一个特定的错误，第5个是@Around，表示这个方法的执行之前和执行，之后，至于在这个点上都做什么，就是我们在Advice，这个方法里头，去写的代码，我们在方法里头，可以做很多的事情，我们可以在方法的执行之前，去判断这个方法的，参数去判断方法的，这个参数是否符合一定的要求，或者说去拿到一些值，把它设到这个方法的参数上去，我们可以在方法的执行完成之后，把方法的返回值，去做一些过滤，或者去做一些修改，再把它返回给客户端，这都是我们在Advice中间。

# 9.1 消息服务