# 面试总结

## 面试开场

### 自我介绍

面试官您好，我叫田金东。现在就读于武汉东湖学院的软件工程专业。在校专业成绩良好，课外时间我也积极自学专业知识，并通过一些小的项目进行知识巩固，对常用Web框架、数据库、以及Linux项目部署都有一定的实践经历，在大二下学期的时候，参加了“蓝桥杯”软件设计大赛获得了省赛一等奖、国赛优秀奖。

生活中自己也有着广泛的兴趣爱好，课外时间积极参加乒乓球协会的活动，与同学相处的也非常融洽。

我今天想面试的职位是Java软件工程师的实习岗，实习工作也是对我人生的挑战。我会在工作中不断地完善自己，提高自己，去适应工作的需要。也希望贵公司能给我这个学习的机会。

### 你性格的优缺点是什么？

缺点：感觉自己有时候性子比较急，有什么事情想到了就想着立马去做好，如果不做心里就不踏实。有时候这个性格就会影响工作或者学习效率。很久之前我就发现了这个问题，慢慢的去改，现在这个问题改正了很多。

优点：自己在工作上比较细心负责，在校老师交给自己的任务都尽自己最大能力去完成了；自己有着良好的自我约束力，自己订的学习计划绝大部分都会不择不扣的去完成。

### 你在学校课余时间有自学专业知识吗？如果有的话你觉得自学有哪些方法？

我的业余时间大部分都用来自学了，在此之间从JavaSE基础学到了SSH和SSM框架，也学习过Linux系统。我的自学过程通常是学习视频与书籍相互结合，因为视频相对于书籍更方便理解，但是书籍有着视频更详细的知识点，所以在学习过程中这两者应该是相互促进的。然后在学习过程中要善于做笔记，我在学习的过程中有做笔记的习惯，这样利于后期的复习，毕竟不是所有知识点学了就能记住，还是需要反复的复习才行。

### 你当初为什么选择Java作为自己的学习方向，它相对于其他语言你觉得有哪些优势以及不足呢？

实话实说当初学Java就是看着Java的市场占有率去学的，当时觉得学Java短时间内不会被淘汰，再这样就给自己留下来发展空间。

现在来看Java我觉的它的优势体现在跨平台，以及生态圈的完善性。跨平台使得项目的开发与部署带来很多方便；在Java开发的大部分需求中都能找到优秀的第三方解决方案。

从Java语言本身来说它也做到了执行效率与开发效率的折中，既解决了语言的跨平台问题，又解决了传统解释语言执行效率的问题。

### 你的职业规划是什么？

我想的是3-5年内努力专研技术，争取有机会向技术总监或者架构师的方向去发展。

### 在你的理解中什么是分布式系统？

分布式系统在我的理解中就是把原先装在一台服务器上面的系统，按照功能拆分成多个模块并部署在不同的服务器中，各个模块之间通过远程调用协同工作，这样做大大增强了系统性能。

### 集群与分布式系统的区别？

集群是指在多台不同的服务器中部署相同应，构成一个集群，通过负载均衡设备对外提供服务。

### RPC框架中主要的三个角



Provider 服务提供者

Consumer 服务的消费者

Registry 注册中心

**调用关系说明：**

* 0. 服务容器负责启动，加载，运行服务提供者。
* 1. 服务提供者在启动时，向注册中心注册自己提供的服务。
* 2. 服务消费者在启动时，向注册中心订阅自己所需的服务。
* 3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
* 4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
* 5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

### 注册中心的作用是什么

在没有注册中心之前，服务消费者不知道有哪些服务提供者，也不知道服务提供者的IP与端口，此时就需要一个注册中心。

服务提供者启动后主动向注册中心注册机器ip、port以及提供的服务列表； 服务消费者启动时向注册中心获取服务提供方地址列表，这样才能通过网络远程掉用服务。

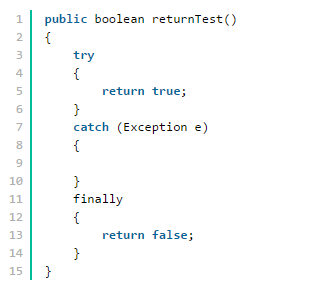
### JSP静态包含与动态包含的区别

静态包含：<%@include file="被包含页面"%>

动态包含：<jsp:include page="被包含页面"flush="true">

1. 使用静态导入（include指令）时，被包含的文件内容会原封不动的插入到包含页中，然后JSP编译器再将合成后的文件最终编译成一个Java文件；使用动态导入（include动作）包含文件时，当该标识被执行时，程序会将请求转发（不是请求重定向）到被包含的页面，并将执行结果输出到浏览器中，然后返回包含页继续执行后面的代码。因为服务器执行的是多个文件，所以JSP编译器会分别对这些文件进行编译；
2. 使用静态包含（include指令）包含文件时，由于被包含的文件最终会生成一个文件（会将导入的目标文件文件内容完整copy到本文件中），所以在被包含、包含文件中不能有重名的变量或方法；而include动作包含文件时，由于每个文件是单独编译的，所以在被包含文件和包含文件中重名的变量和方法是不相冲突的。
3. 动态导入的实质是在Servlet的service()方法中通过include方法来引入另一文件
4. 静态导入时被导入页面的编译指令会起作用；而动态导入时被导入页面的编译指令则失去作用，只是插入被导入页面的body内容。

### finally代码块



上述代码的执行结果？

答：false

### 请说出作用域public，private，protected，以及不写时的区别

private修饰的成员变量和函数只能在类本身和内部类中被访问。

protected 修饰的成员变量和函数能被类本身、子类及同一个包中的类访问。

public修饰的成员变量和函数可以被任意其他类访问。

default：即能被类本身以及同一个包中的类访问。

### 简述this关键字的用法

1、this调用本类中的属性，也就是类中的成员变量；

2、this调用本类中的其他方法；

3、this调用本类中的其他构造方法，调用时要放在构造方法的首行。

### String,StringBuffer与StringBuilder的区别?

String 字符串常量

StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

String是对象不是原始类型为不可变对象，一旦创建就不能修改它的值

对于已经存在的String的对象的修改都是重新创建一个新的对象，然后把值保存进去（旧的值会被回收）。String、StringBuffer、StringBuilder是final类，不能被继承

StringBuffer和StringBuilder是一个可变对象，它只能通过构造函数来建立。

### Java中，请说明final关键字的使用

1.用来修饰数据，包括成员变量和局部变量，该变量只能被赋值一次且它的值无法被改变。对于成员变量来讲，我们必须在声明时或者构造方法中对它赋值；

2.用来修饰方法参数，表示在变量的生存期中它的值不能被改变；

3.修饰方法，表示该方法无法被重写；

4.修饰类，表示该类无法被继承。

### 简述方法的重写和重载

**override（重写）**

1.方法名、参数、返回值相同。

2、重写方法的访问修饰符一定要大于或等于被重写的方法。

3、重写方法不能抛出新的异常或者抛出比被重写方法更宽泛的异常

4、存在于父类和子类之间。

5、被final修饰的方法不能重写

**overload（重载）**

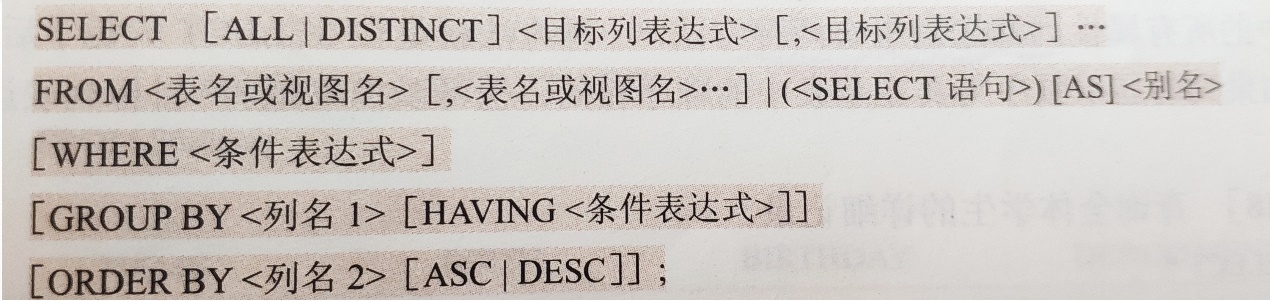
1、参数类型、个数、顺序至少有一个不相同。

2、不能重载只有返回值不同的方法名。

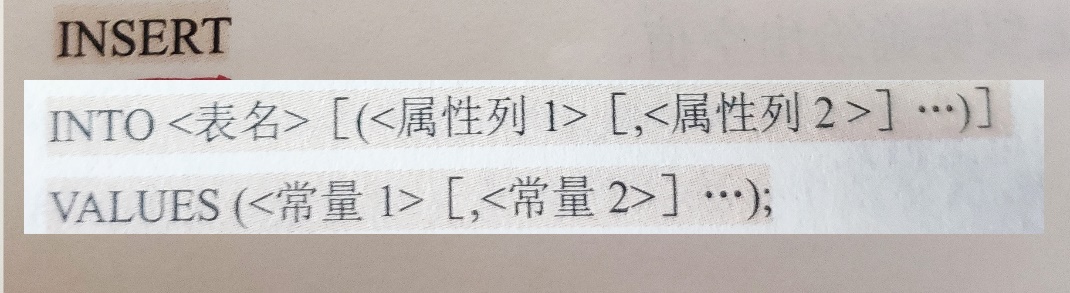
3、存在于同类中。

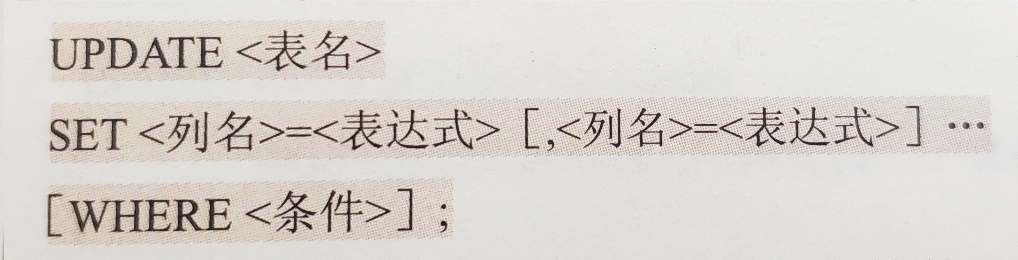
### SQL语句

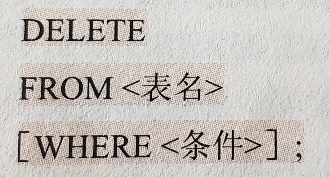
1. 查询语句



1. 插入数据



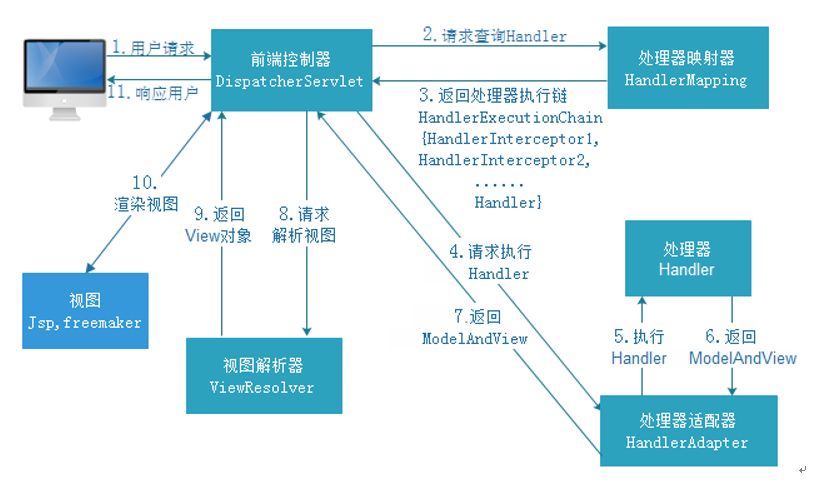
1. 修改数据  
   
2. 删除数据



## SpringMVC篇

### SpringMVC核心组件有哪些

前端控制器、处理器映射器、处理器适配器、视图解析器



### 搭建SpringMVC的步骤有哪些

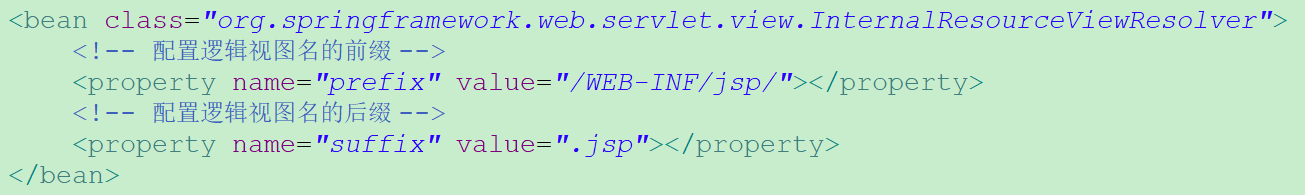
1. 创建springmvc.xml文件，在文件中添加<contxt:component>标签（开启组件扫描）
2. 在web.xml中配置前端控制器。配置前端控制器有两个要点：
   1. 需要配置，springmvc.xml配置文件的位置
   2. 需要配置拦截方式：
      1. /\* jsp js css 全部拦截
      2. \*.action 只拦截指定后缀名的请求（可以配置多个）
      3. / 除\*.jsp以外，所有东西都拦截。

（下面两个常用）

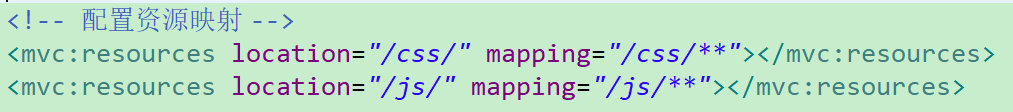
1. 编写Controller类（类上面需要加上Controller注解，方法上加上RequestMapping注解）

### springmvc.xml的其他配置

1. 配置视图解析器（配置路径前缀和后缀）



1. 配置资源映射器（如果拦截方式为“/”,那么前端控制器也会拦截对css、js文件的请求）



### 请求参数的获取

1. 默认支持的参数类型

HttpServletRequest

HttpServletResponse

HttpSession

Model/ModelMap：

ModelMap是Model接口的实现类，也可以通过ModelMap向页面传递数据。使用Model和ModelMap的效果一样，如果直接使用Model，springmvc会实例化ModelMap。

1. 接收简单类型

Integer int Integer[] int[]

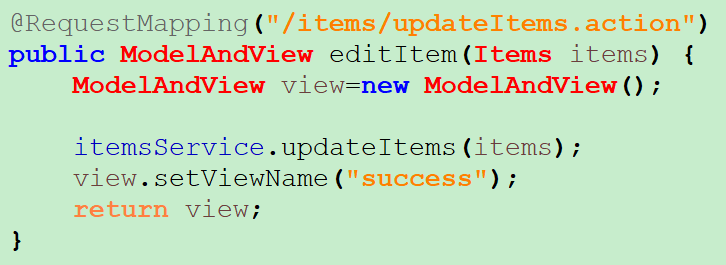
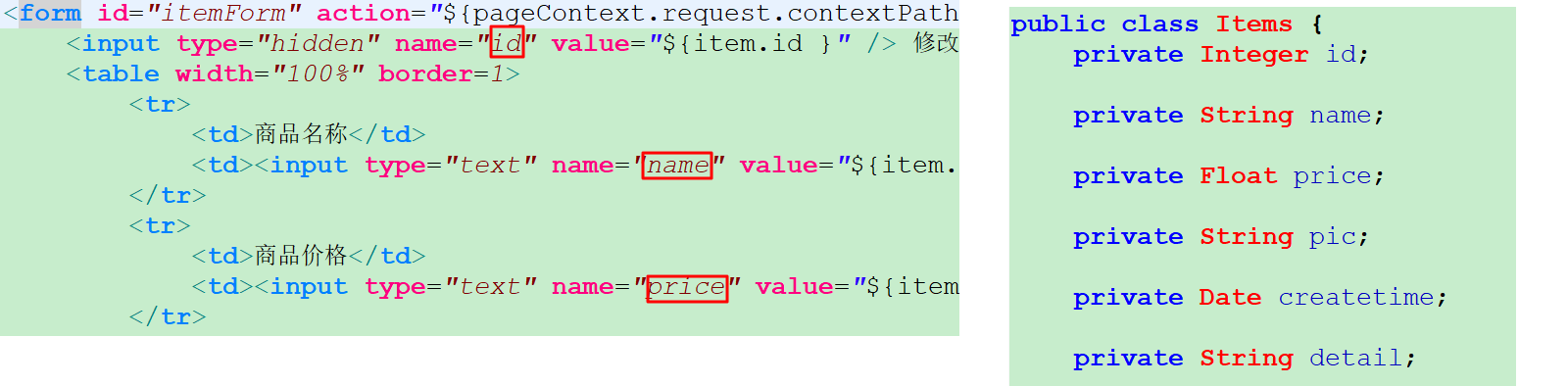
Float float Float[] float[]

Double double Double[] double[]

Boolean Boolean ：对于布尔类型的参数，请求的参数值为true或false。或者1或0

1. 接收封装类型

如果封装类型中全部都是基本类型，那么直接使用封装类型接收即可。

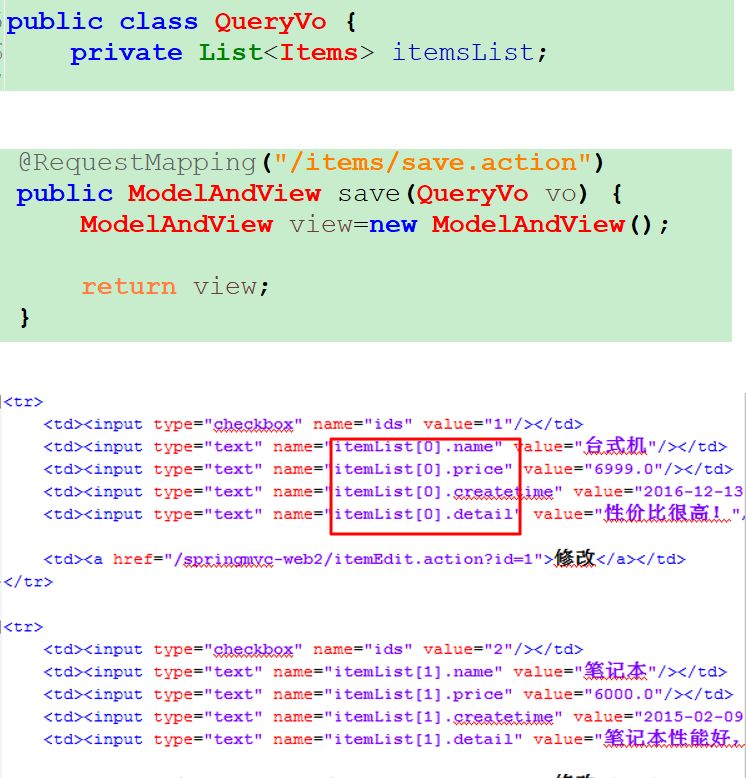


如果封装类型中，仍然有封装类型，那么此时就需要用对象导航形式了



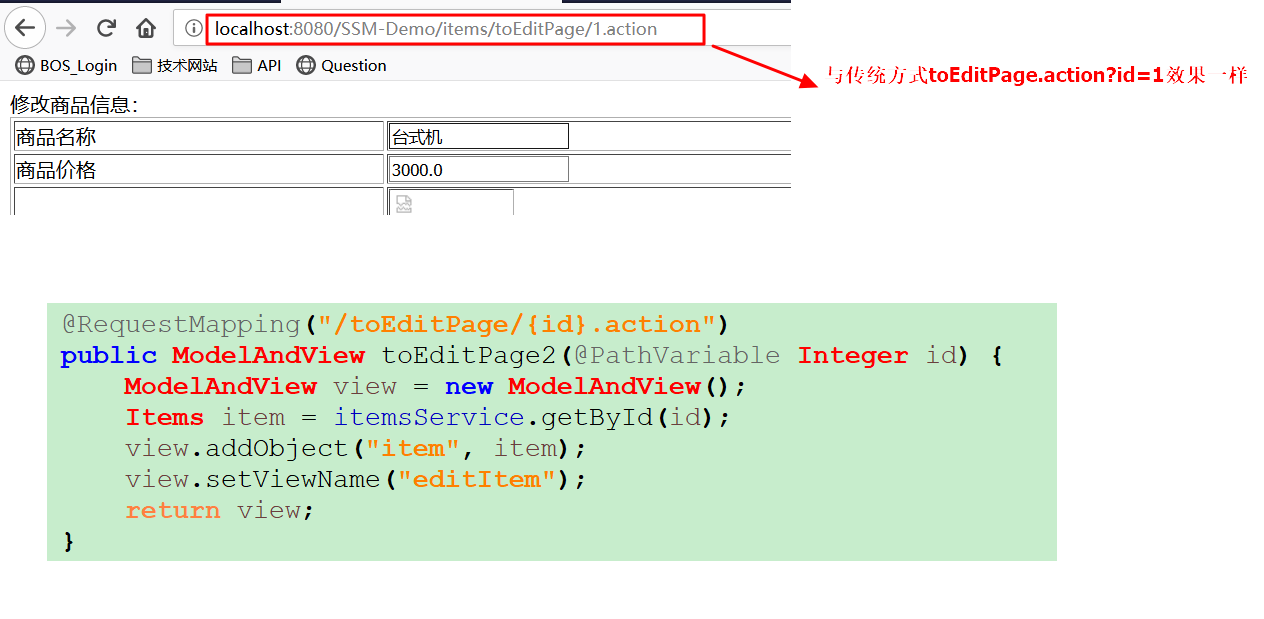
注：方式b和方式c可以混合使用，如果封装类型中的属性和局部变量的名称相同，则两个都会同时接收参数。

1. List集合封装多个对象，一次接收多批数据



List<Items>和 Items[] 只能先由一个类封装，然后由封装类接收参数，如果试图将它们直接放在方法上面接收参数则会报错。如果是基本类型的数组，直接放在方法参数上面是可以的。基本类型的List不能作为方法参数。基本类型的数组以及基本类型List可以放在封装类型中。

1. 从URL上获取参数

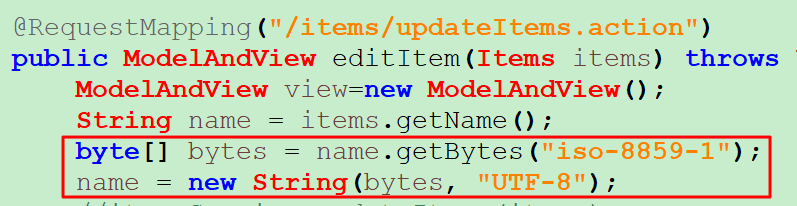


总结：方法的参数只能接收基本类型及其封装类型，以及基本类型和封装类型的数组（注意：是基本类型的数组）。如果想接收List（包括泛型为基本类型，和domain封装类）或者domain封装类的数组，必须由封装类封装，然后由封装类封装。

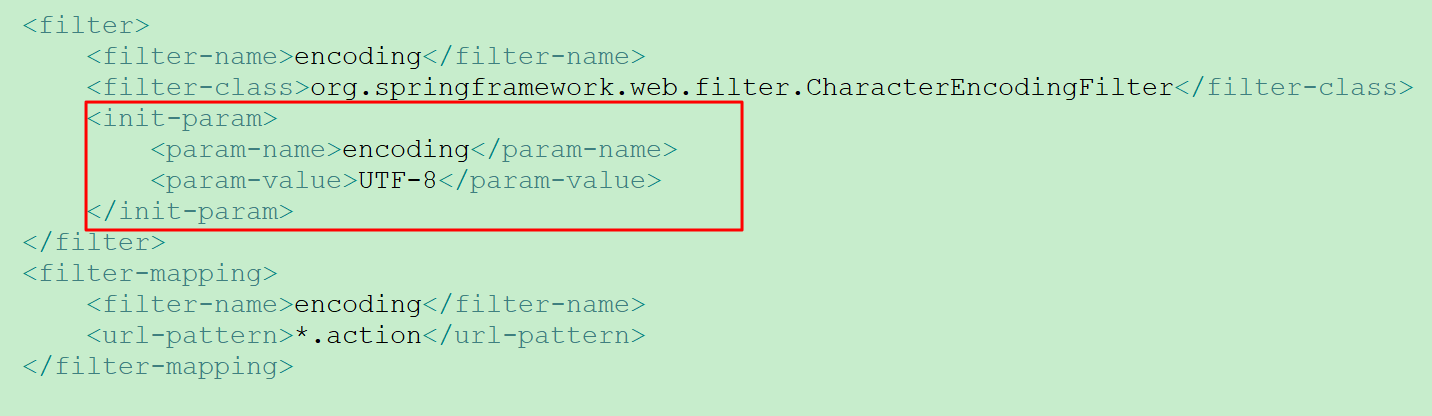
### POST乱码解决

乱码原因：前端传入的参数采用iso-8859-1的编码格式，而后台接收后采用UTF-8格式解码，所以显示的文字会乱码。

方式一：手动解决



方式二：配置Spring-web提供的过滤器



### SpringMVC常用注解

1. @RequestMapping

用途：配置请求的映射路径

属性：

1. method属性：指定前端请求的访问方式（Post还是Get）

示例：@RequestMapping(method = RequestMethod.GET)

1. value属性：用于指定该方法的访问路径，可以通过数组形式，指定多个路径。如果该标签只有value一个属性可以省略value。
2. @RequestBody

用途：在获取JSON数据时使用

1. @ResponseBody

用途：在返回Json数据时使用

注：**@ResponseBody的实质是通知程序不走逻辑视图，如果返回的是字符串就直接将字符串响应给客户端，如果方法返回的不是字符串，则视图将对象转换成JSON字符串，然后直接响应给客户端。**

1. @PathVariable

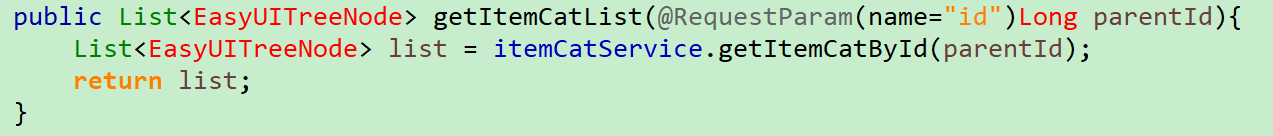
用途：在获取URL路径中的请求参数

1. @RequestParam

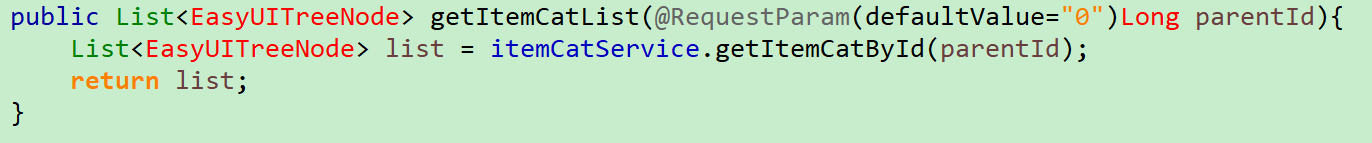
用途：指定提交的参数名，以及指定参数的默认值

属性：

1. name/value属性：用于将请求的参数列表中指定参数赋给注解所指定的方法入参。



1. defaultValue属性：指定如果该方法入参没有接收到请求值时的默认值



1. required属性：指定参数是否必须被绑定

### Controller方法的返回值

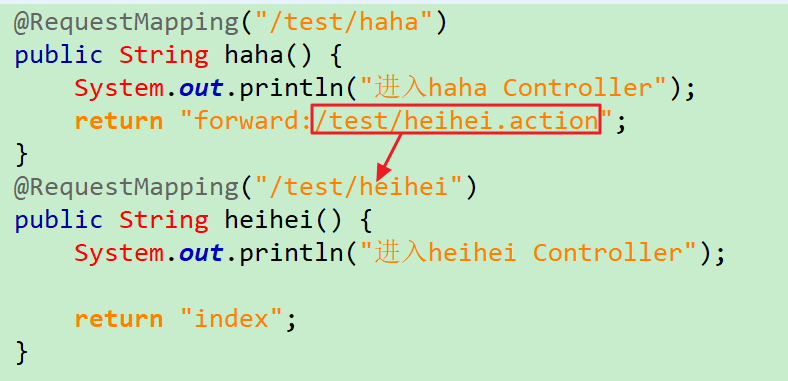
1. 返回 ModelAndView 对象

ModelAndView可以存入返回给前端页面的数据，并且保存需要跳转的视图名。

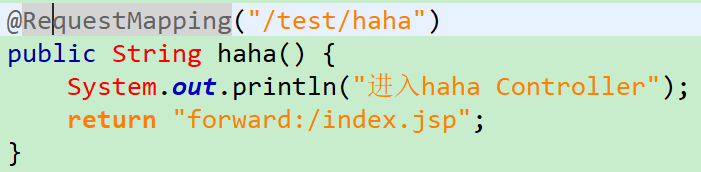
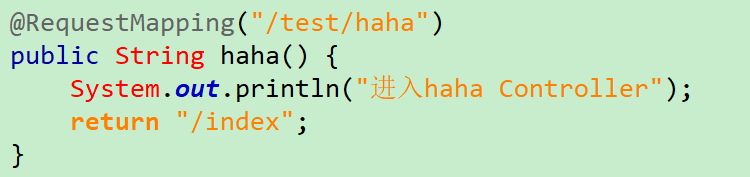
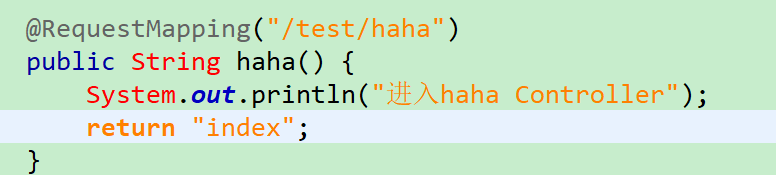
1. 返回String

返回的String就是逻辑视图名的名称。可以通过指定字符串的前缀，来指定采用何种方式跳转视图（转发/重定向）

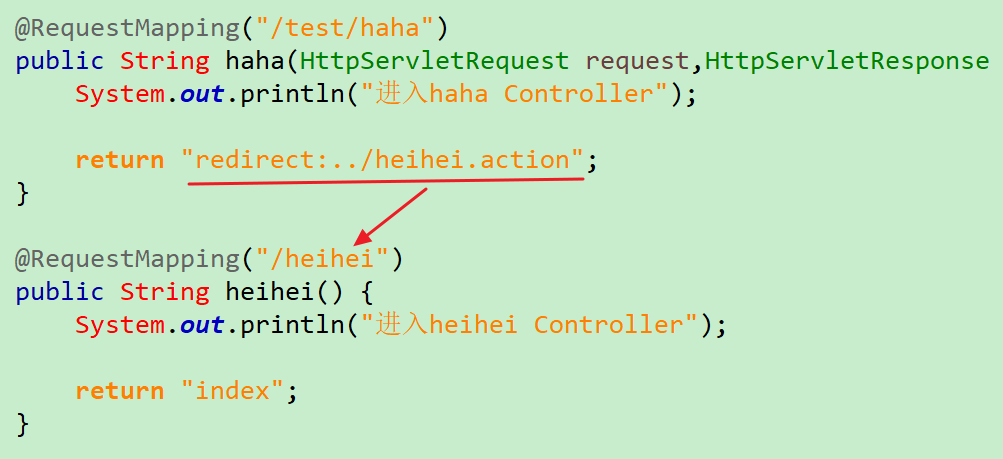
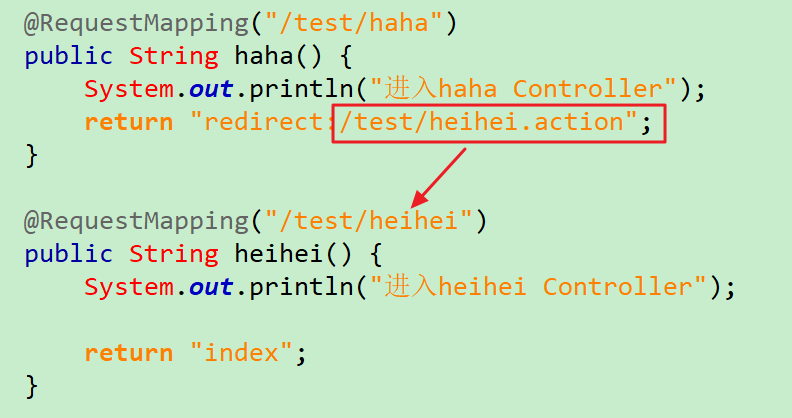
**转发到controller方法：**



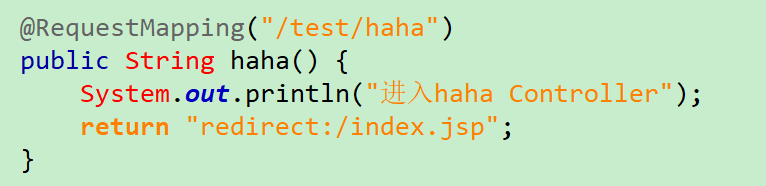
**转发到JSP：**



**重定向到Controller：**



**重定向到JSP：**



**重定向到其它服务器：**



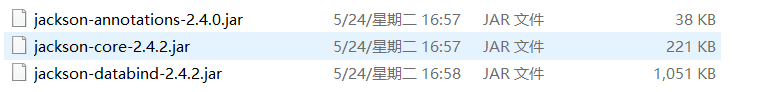
1. 返回void

如果返回void就不能像前面那样直接指定跳转的视图名了，只能通过原生Request和Response来进行转发和重定向。

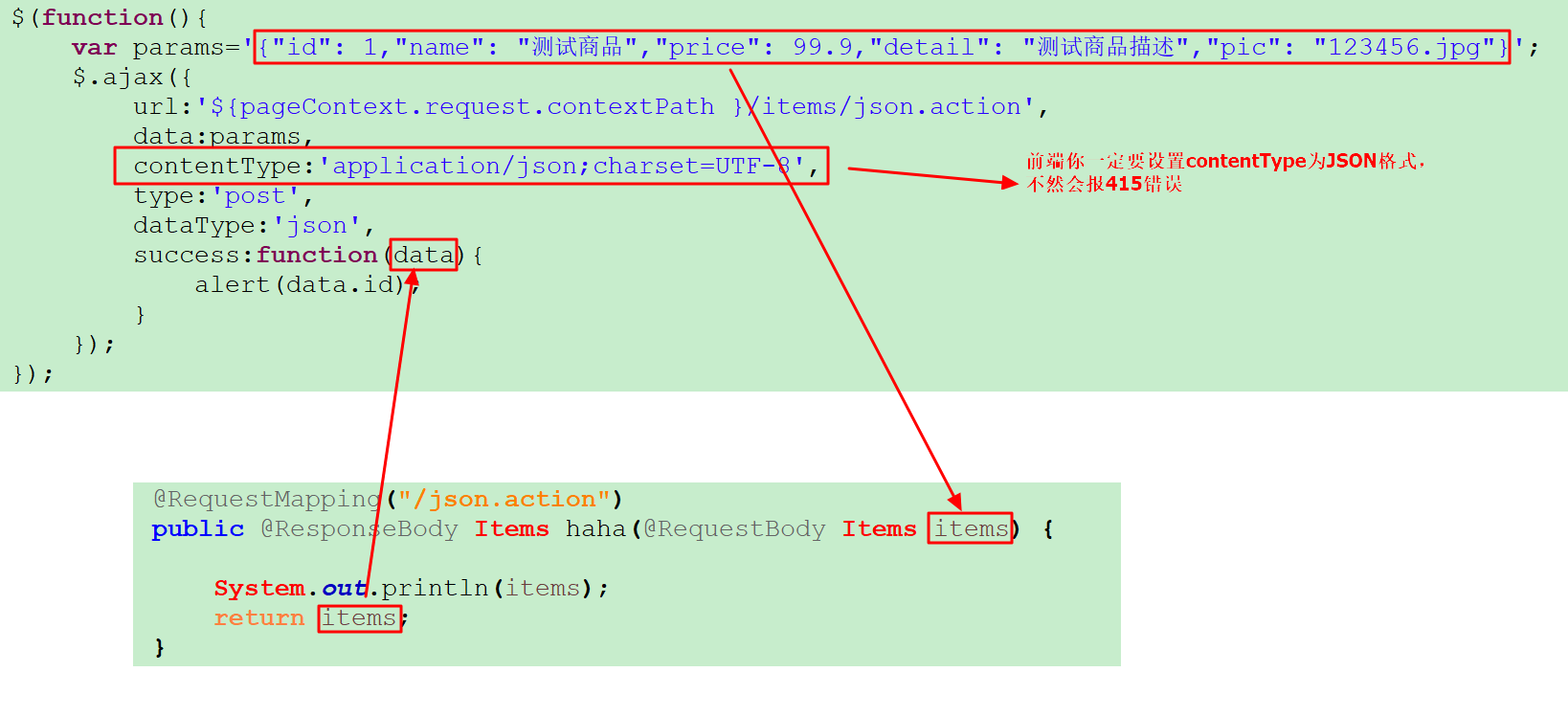
void返回类型通常与Ajax请求配合使用，因为Ajax请求不用跳转页面。

### JSON数据交互

需要做的准备：SpringMVC想要接收JSON必须导入第三方jar。



客户端常常会将请求的内容设置为JSON字符串，那前端控制器该如何接收这个JSON字符串呢，我们可以使用@RequestBody注解将json字符串转换成一个Java对象：



使用@RequestBody这个接收的前提就是，客户端发送的请求头中content-type属性值必须为application/json，否则会报415错误。

注：通常我们不管是否需要向前台返回一个对象（例如：String等），我们都在返回值前面加上@ResponseBody注解。如果Controller中的方法执行完成之后 不想给前台返回数据，则也需要加上@ResponseBody，因为即使方法返回值为void，spring也会按照前台请求过来的页面地址去找。

### 拦截器

Spring Web MVC 的处理器拦截器类似于Servlet 开发中的过滤器Filter，用于对处理器进行预处理和后处理。

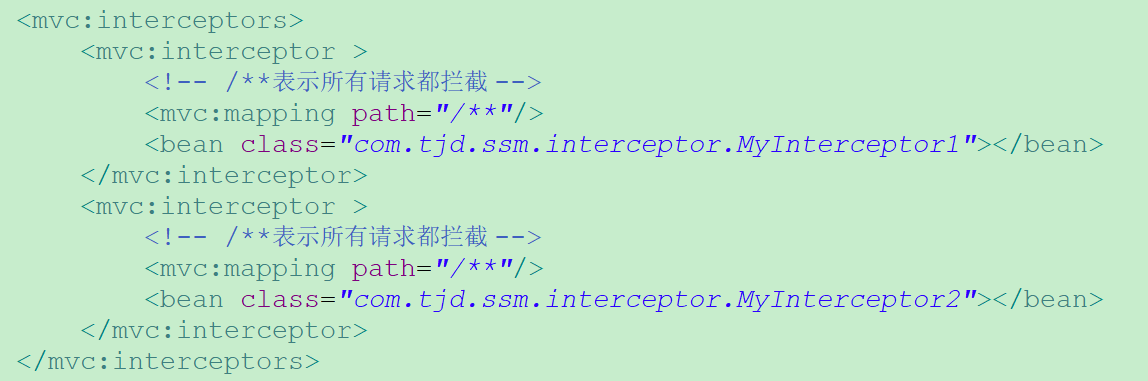
**拦截器的使用步骤**

第一步：自定义拦截器（实现**HandlerInterceptor**接口）



**preHandle方法返回true表示放行，返回false则表示截断，直接跳过Controller去执行拦截器的afterCompletion方法**

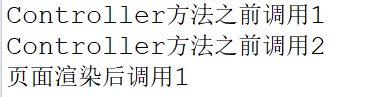
第二步：配置拦截器（springmvc.xml）



**拦截器的执行顺序**

1. preHandle按拦截器定义顺序调用，如果拦截链前面被截断则不会被调用
2. postHandler按拦截器定义逆序调用，只有在前面所有拦截器的的preHandler放行后才会被调用
3. afterCompletion 按拦截器定义逆序调用，只有当自己拦截器的preHandle方法放行（返回true）后才会执行

例：在MyInterceptor1中放行，在MyInterceptor2中拦截。访问服务器后会打印下列内容。



## Spring篇

### Spring核心思想

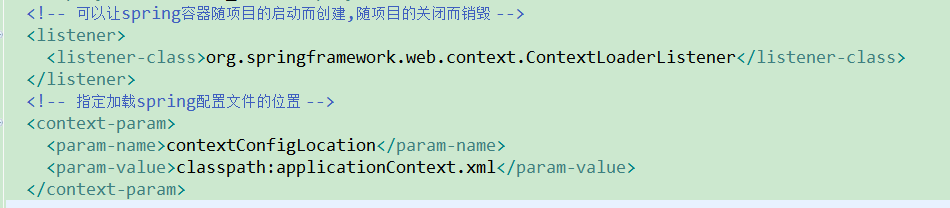
IOC（Inverse Of Control）反转控制：以前对象的创建是由开发人员自己维护的，包括依赖关系也是自己注入的。使用Spring后，对象的创建以及依赖关系的注入可以交给Spring来完成。反转控制就是将对象的创建反转给了程序自己控制。

DI（Dependency Injection）依赖注入：DI需要有IOC环境，IOC又需要DI进行依赖注入；Spring创建

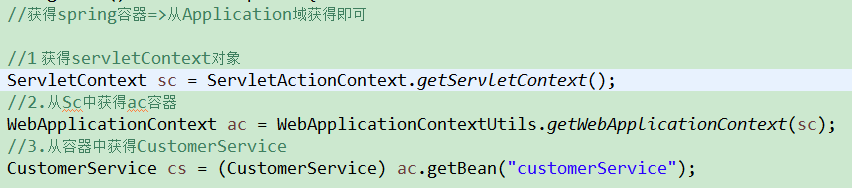
**IoC 控制反转，指将对象的创建权，反转到Spring容器 ， DI 依赖注入，指Spring创建对象的过程中，将对象依赖属性通过配置进行注入**

### WEB项目中如何使用Spring

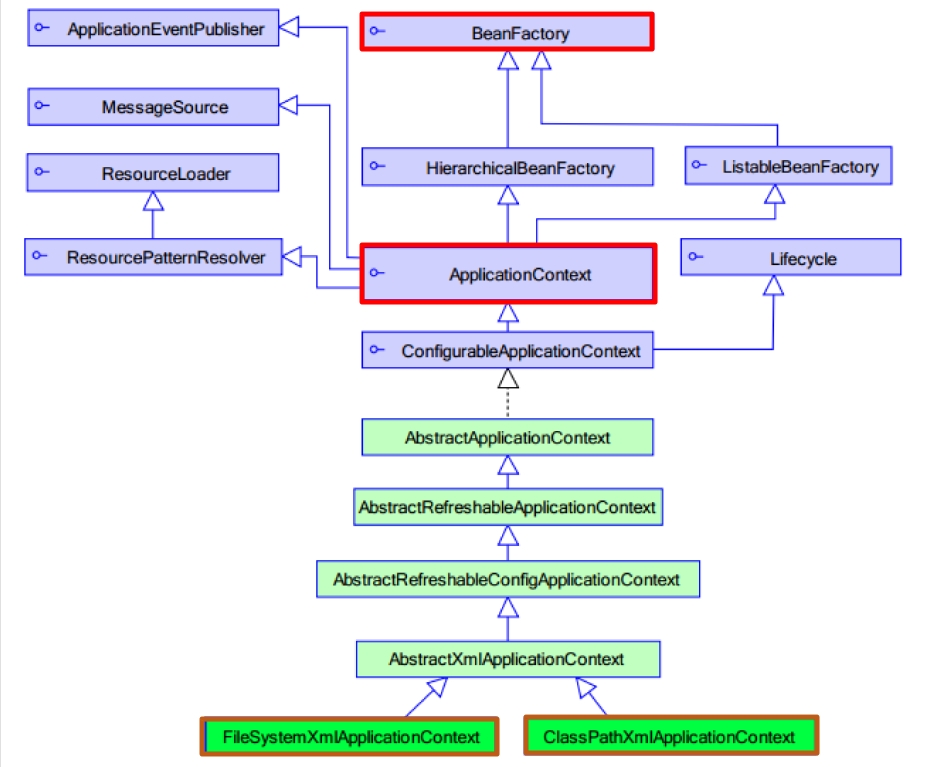
在一个web项目中，我们spring容器对象只会创建一个，该容器的生命周期应该与web项目一致，Spring框架自带了一个ServletContextListener类型的监听器，类名为ContextLoaderListener，它实现了ServletContextListener接口，所以我们只需要在web.xml文件中配置该Listener，则可以实现随项目启动。因为在项目启动时创建ServletContext时就会触发该监听器，在该监听器中自动创建了Spring容器对象(ApplicationContext)，并将该对象放入了ServletContext域中，所以我们在Web项目中都可以通过ServletContext域获得Spring容器对象。



**如何获取ServletContext域中的ApplicationContext对象？**



### Spring的核心类



ApplicationContext接口：每次容器启动时就会创建容器中配置的所有对象.并提供更多功能

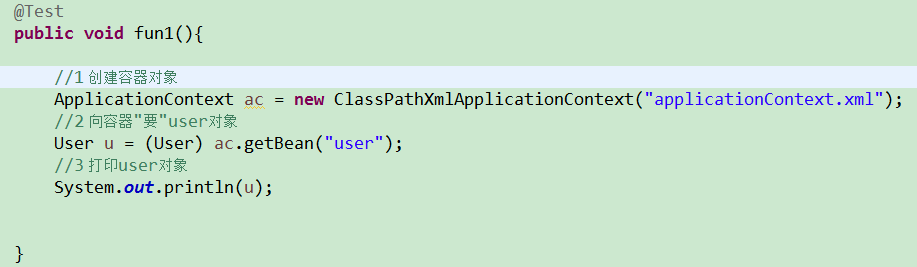
**BeanFactory接口和ApplicationContext接口有什么区别？**

①ApplicationContext 接口继承BeanFactory接口，Spring核心工厂是BeanFactory ,BeanFactory采取延迟加载，第一次getBean时才会初始化Bean, ApplicationContext是会在加载配置文件时初始化Bean。

实现类：

从类路径下加载配置文件: ClassPathXmlApplicationContext("applicationContext.xml")

从磁盘绝对路径加载配置文件：FileSystemXmlApplicationContext("d:/xxx/yyy/xxx")



### Spring Bean配置



**基础属性**：

**name**：给被管理的对象取个名字，获得对象时根据名称获得对象（name属性值可以重复，但开发中一般不重复，因为没啥意义）

**class**：被管理类的全限定名称

**id**: 功能与name属性完全相同，但id属性必须遵循id命名规范（名称不可重复，不可使用特殊字符）

**进阶属性**：

**scope属性**：

**singleton**(默认值): 单例对象.被标识为单例的对象在spring容器中只会存在一个实例

**prototype**：多例原型.被标识为多例的对象,每次再获得才会创建.每次创建都是新的对象.整合struts2时,ActionBean必须配置为多例的.

request：web环境下.对象与request生命周期一致.

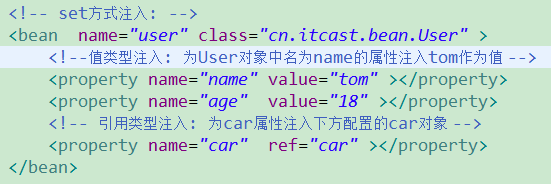
session：web环境下,对象与session生命周期一致.

**init-method属性**：配置一个方法作为生命周期初始化方法，spring会在对象创建之后立即调用。

**destory-method属性**：配置一个方法作为生命周期的销毁方法，spring容器在关闭并销毁所有容器中的对象之前调用。

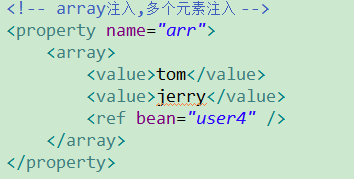
### Spring Bean对象依赖注入的方式

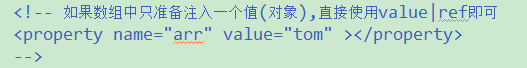
1. set方法注入



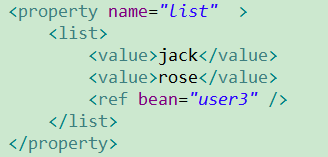
复杂类型注入：

数组：



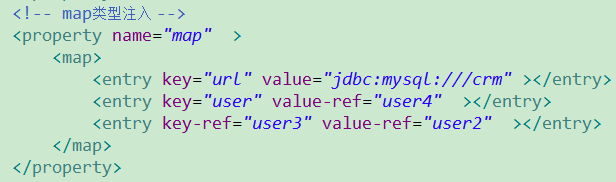


List：

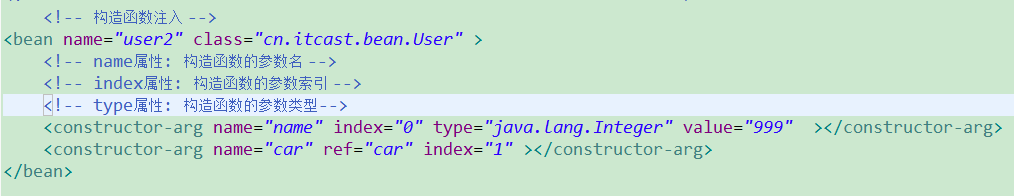




Map：

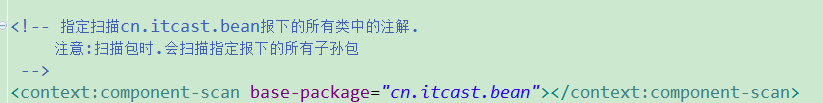


1. 构造函数注入

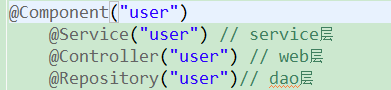


### Spring注解配置

在使用注解进行配置时，需要开启组件扫描：



**将类注册到容器中：**

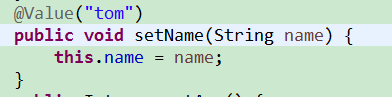


**修改对象的作用范围：**



**值类型注入：**





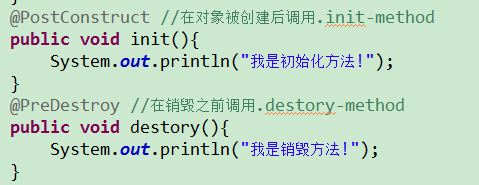
@Value注解既可以放在字段上，也可以放在set方法上。前者采用反射进行注入，后者是使用set方法进行的注入。

**引用类型注入：**





**初始化|销毁方法：**



### 什么是AOP思想

AOP Aspect Oriented Programming（面向切面编程），通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是OOP（面向对象编程）的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

### spring实现aop的原理

Spring中使用了两种动态代理技术，并且优先使用JDK自带的动态代理，如果该类没有实现接口，则使用cglib代理。

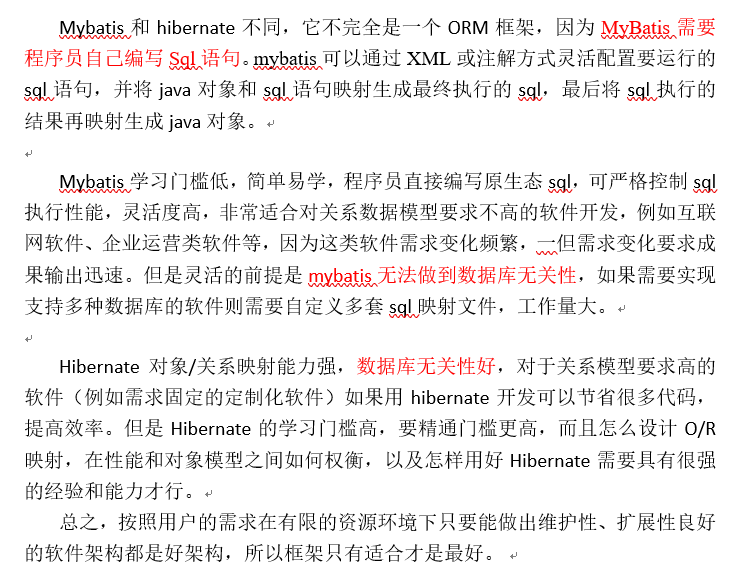
cglib代理：第三方代理技术,cglib代理.可以对任何类生成代理.代理的原理是对目标对象进行继承代理. 如果目标对象被final修饰.那么该类无法被cglib代理。这也就是为什么Hibernate技术中实体对象要求不能使用final修饰符修饰，因为修饰后就不能通过继承产生动态代理对象了。

### Spring在项目中的作用或特点

1. 我认为Spring在项目中最大的意义就是代码的解耦，如果某一个类的代码需要更改，我们只需要重新写好后，更改配置文件即可。不需要更改原先写好的代码。
2. 使用Spring可以真正的做到面向接口编程。
3. Spring可以轻松整合其它框架，如果没有Spring我们需要手动管理很多类。

## MyBatis篇

### Hibernate与MyBatis的区别



1. Hibernate可以做到数据库无关性，而MyBatis不行。
2. Hibernate完全面向对象，Hibernate由于是自动生成SQL语句，所以在性能优化上更加困难，MyBatis由于是手动编写SQL语句，在SQL优化上面更加容易。

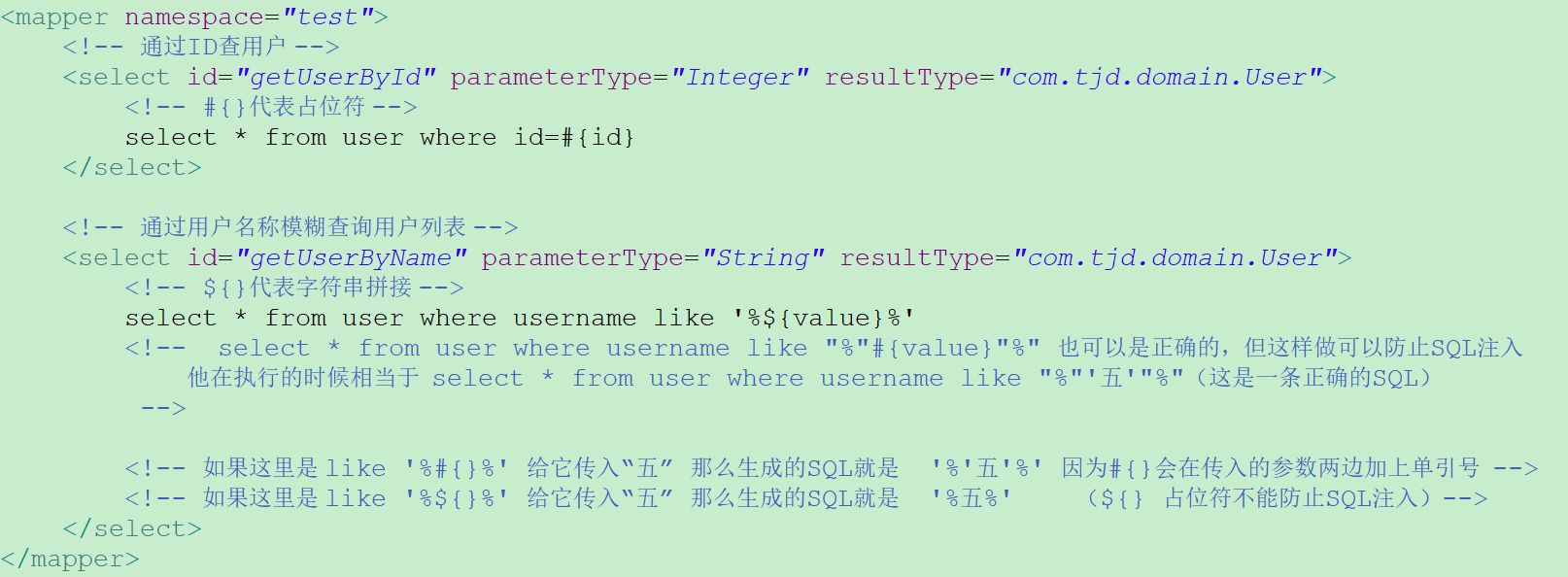
### MyBatis配置文件

**SqlMapConfig.xml（核心配置文件）**

在这个配置文件中需要配置数据库四要素（用户名、密码、URL、驱动的全限定类名）



**xxxMapper.xml（映射文件）**



### MyBatis核心类

**SqlSessionFactoryBuilder**

SqlSessionFactoryBuilder用于创建SqlSessionFacoty，SqlSessionFacoty一旦创建完成就不需要SqlSessionFactoryBuilder了，因为SqlSession是通过SqlSessionFactory创建的。所以可以将SqlSessionFactoryBuilder当成一个工具类使用，最佳使用范围是方法范围即方法体内局部变量。

**SqlSessionFactory**

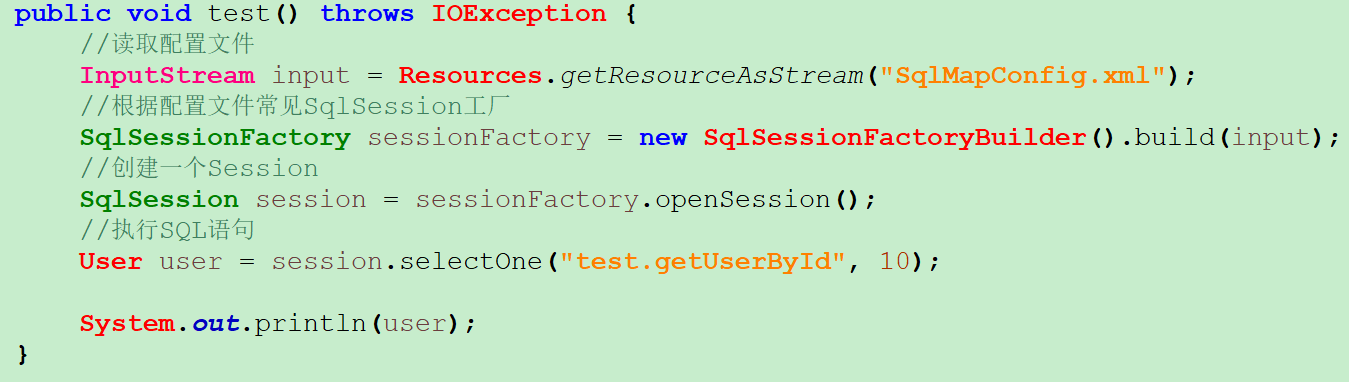
SqlSessionFactory是一个接口，接口中定义了openSession的不同重载方法，SqlSessionFactory的最佳使用范围是整个应用运行期间，一旦创建后可以重复使用，通常以单例模式管理SqlSessionFactory

**SqlSession**

SqlSession是一个面向用户的接口，sqlSession中定义了数据库操作方法。每个线程都应该有它自己的SqlSession实例。SqlSession的实例不能共享使用，它也是线程不安全的。因此最佳的范围是请求或方法范围。绝对不能将SqlSession实例的引用放在一个类的静态字段或实例字段中。

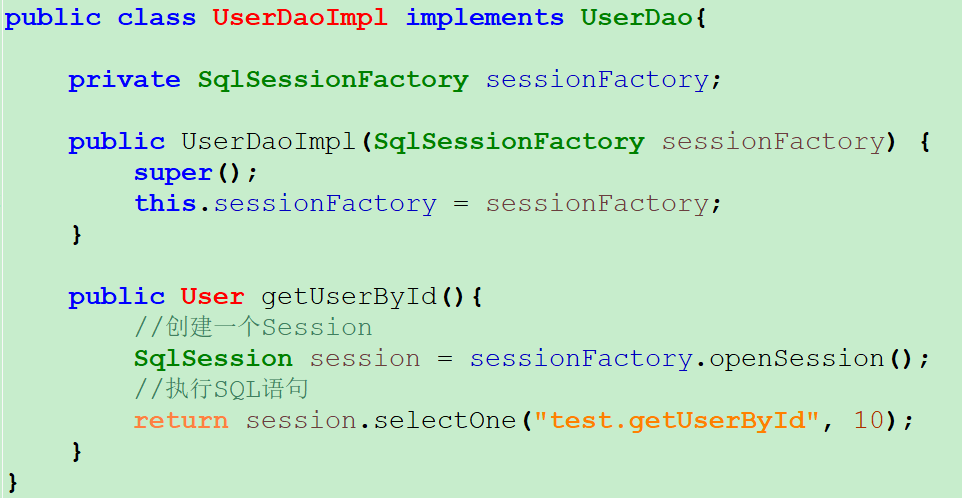
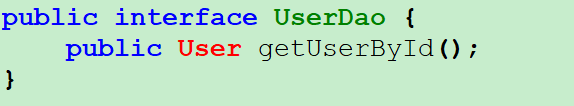
打开一个 SqlSession；使用完毕就要关闭它。通常把这个关闭操作放到 finally 块中以确保每次都能执行关闭。

**示例:**



### 开发中为什么通常使用Mapper动态代理开发

**在不使用Mapper动态代理时，开发的代码**：



弊端：

1. 我们可以发现，在UserDaoImpl的方法中，sessionFactory.openSession()显得很多余，因为如果该接口有多个方法，每个方法都需要写这句话，显得很麻烦。那此时MyBatis提供了一个Mapper动态代理的方式，帮我们实现这些方法。
2. **通过字符串去调用标签定义的SQL，这里存在硬编码。**

**Mapper动态代理开发**

Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。

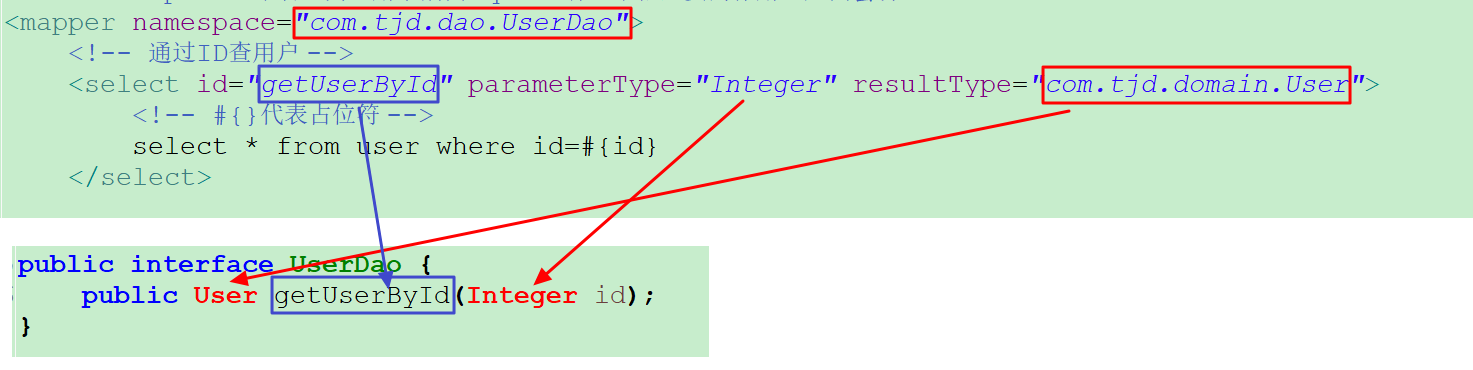
Mapper接口开发需要遵循以下规范：

1. Mapper.xml文件中的namespace与mapper接口的类路径相同。
2. Mapper接口方法名和Mapper.xml中定义的每个statement的id相同。
3. Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同。
4. Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同。

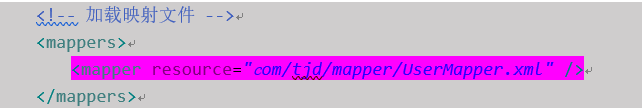
配置文件的名称不一定必须要与对应的接口名相同，但为了方便我们都是这样做的。

具体步骤：

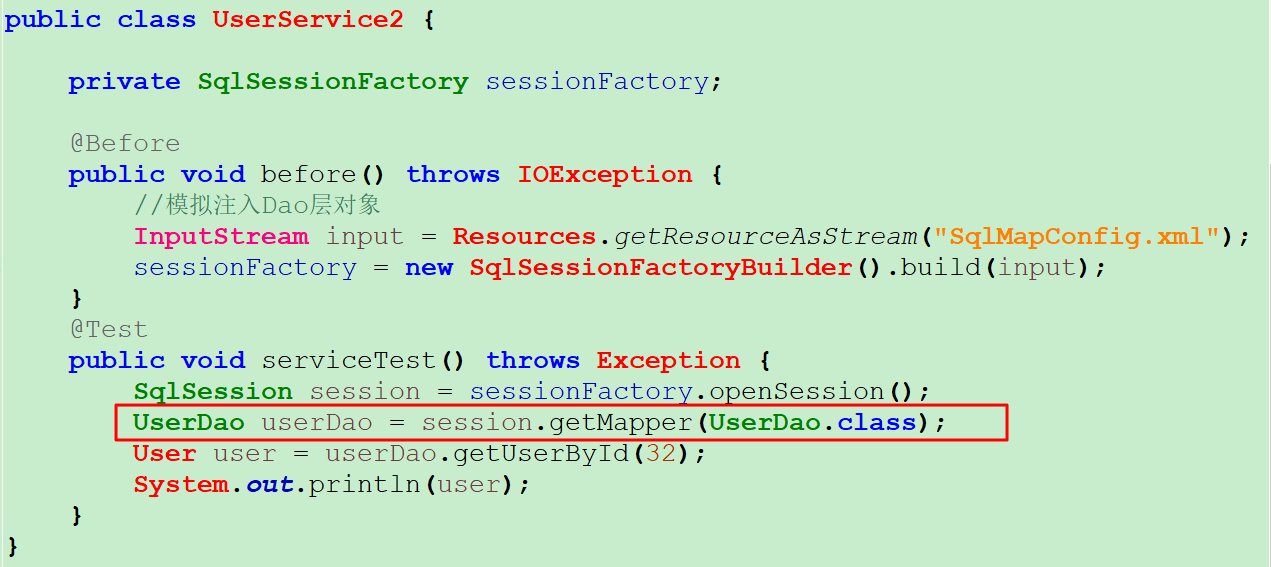
第一步：创建接口以及配置映射文件



第二步：在sqlMapConfig.xml中配置映射文件路径



测试：



**通过SqlSession的getMapper()方法获取动态代理对象。**

### SqlMapConfig.xml文件配置



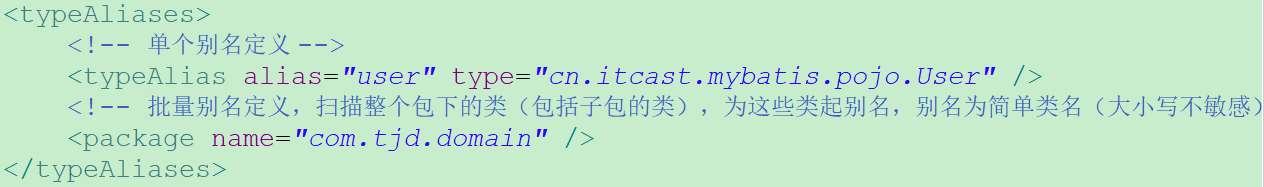
**properties标签**

用于加载外部配置文件，以及配置本地property

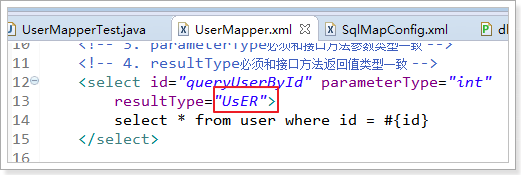


**typeAliases标签**

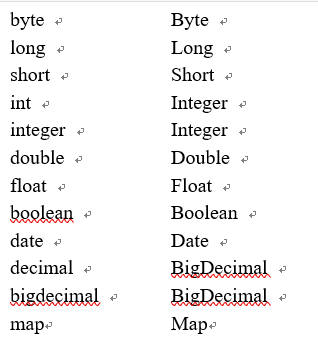
该属性用于自定义别名，在没有给POJO类起别名前，我们在配置文件必须使用全限定类名来描述这些类。如果给这些类起别名，那么我们就不需要使用全限定名称，直接写指定的别名即可。



注意：MyBatis中所有别名大小写不明感（不管起的别名是大写还是小写，在使用别名时大小写都可以任意使用）



**MyBatis自带的别名**

**mappers标签**

**<mapper resource=" " />**

使用相对于类路径的资源（现在的使用方式）

如：<mapper resource="sqlmap/User.xml" />

**<mapper class=" " />**

使用mapper接口类路径

如：<mapper class="cn.itcast.mybatis.mapper.UserMapper"/>

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。（Mapper接口就是前面提到的Dao层接口）

**<package name=""/>**

注册指定包下的所有mapper接口

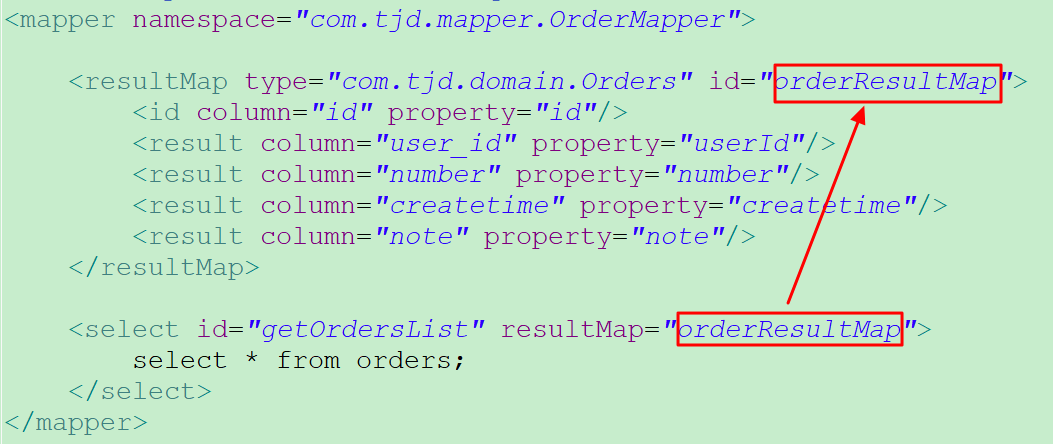
如：<package name="cn.itcast.mybatis.mapper"/>

注意：**此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。**这样做是为了在程序运行时通过接口名，找到与自己同名的映射文件。

### 映射文件配置

1. **resultMap手动映射配置**

**手动配置的意义：**在很多情况下，我们写的封装类并不是与数据库表中字段一一对应的，那么我们就需要使用手动映射这些没有对应的字段。



id标签：

用于定义主键 ,非常重要。如果是多个字段,则定义多个id。

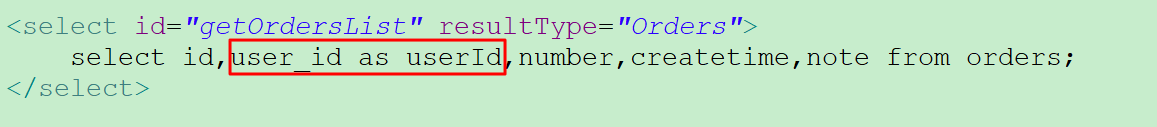
property：主键在pojo中的属性名

column：主键在数据库中的列名

result标签：

用于定义普通属性

我们不仅可以采用这种手动映射的方式，其实我们只需要在查询时给userId取一个别名，让别名与封装对象中属性的名称对应即可，即可使用传统的自动映射封装对象了。



1. **${}与#{}**

#{}：输入参数的占位符，相当于jdbc的？，该占位符可以防止SQL注入。

**#{}：**

select \* from user where username like '%#{value}%'。

上述MyBatis中配置的语句，执行时会转换成 select \* from user where username like '%'五'%' ，这样语句就执行失败（传入的参数是"五"）。在使用#{}占位符时，大括号内部可以使用任何单词取到值。

**${}：**

select \* from user where username like '%${value}%'

上述MyBatis中配置的语句，执行时会转换成 select \* from user where username like '%五%' （传入的参数是"五"）。在使用${}占位符时，大括号内部必须使用value才能取到参数的值

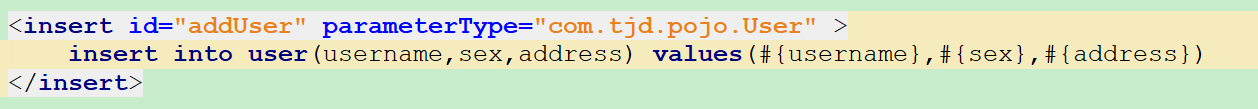
1. 当参数为一个基本类型

当使用#{}占位符时，大括号内部填写任意字符，通常为value或者v

当使用${}占位符时，大括号内部必须使用value取出参数值

1. 当入参是一个封装类型

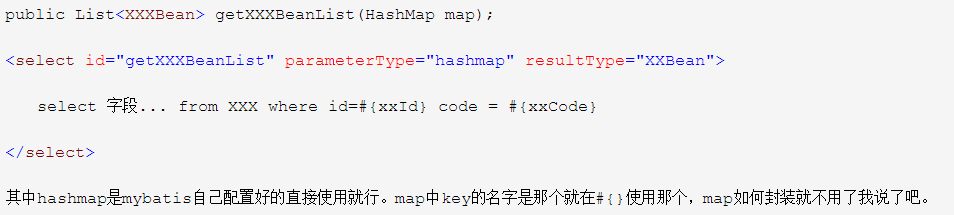
#{} 里面直接填写封装类型中的属性名即可，不需要使用xxx.属性名获取值



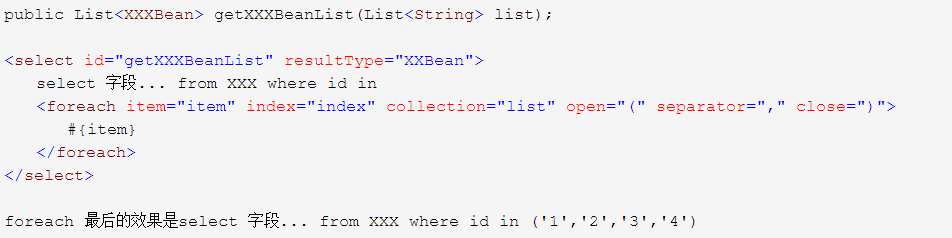
1. 传入多个参数

当使用Mapper动态代理开发时，如果方法传入多个参数，那么我们通过#{index}的形式，依次取出参数值，index从0开始。

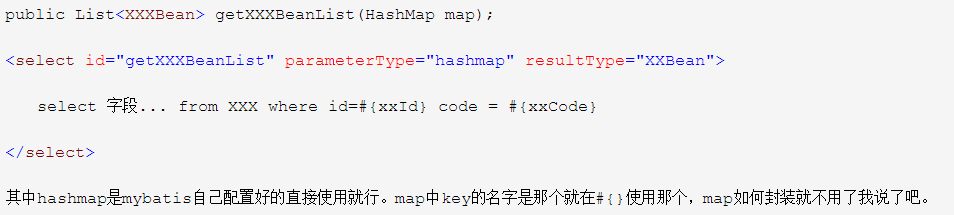
1. 当入参是一个HashMap



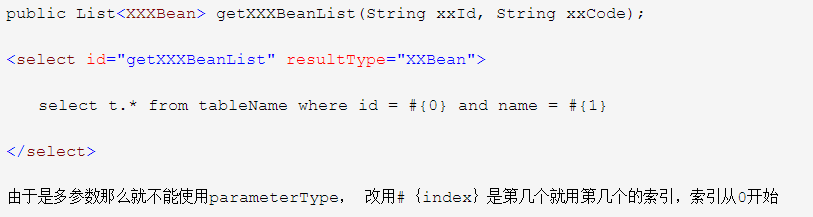
1. **parameterType的配置规则**
2. 多个参数



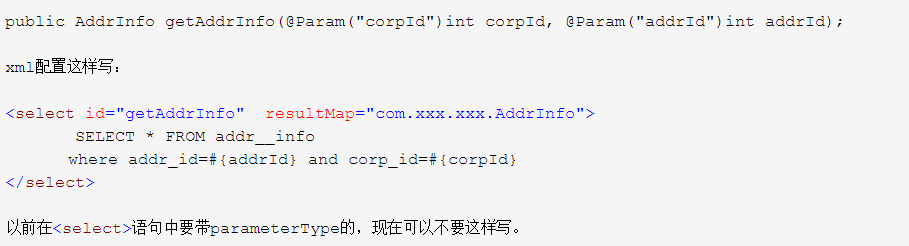
1. Map封装多个参数



1. List封装多个参数

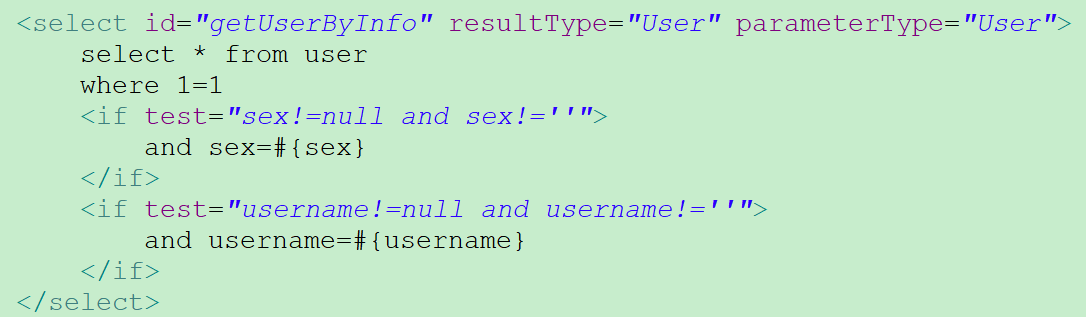


1. 通过注解传递多个参数

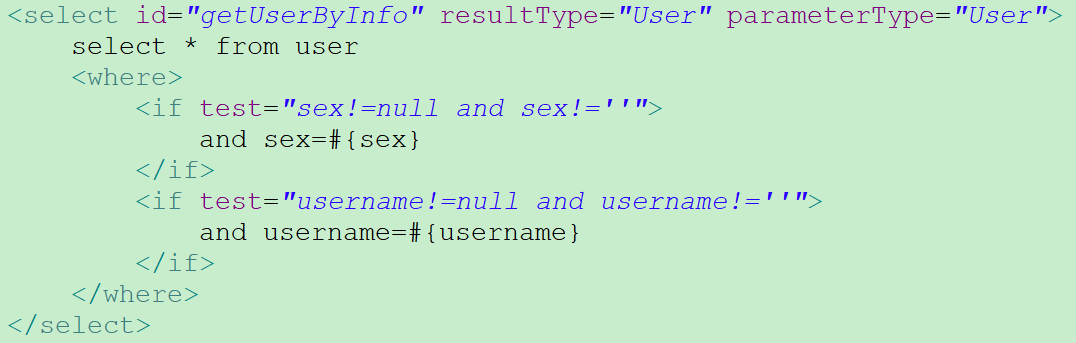


@Param注解的作用是给参数命名,参数命名后就能根据名字得到参数值,正确的将参数传入sql语句中

1. **resultType的配置规则**
2. 当返回值是一个List集合时，resultType只需要写集合的泛型即可。
3. 当返回值是一个封装对象时，直接写封装对象的全限定类名即可，或者别名。
4. **动态SQL**
5. **if标签**

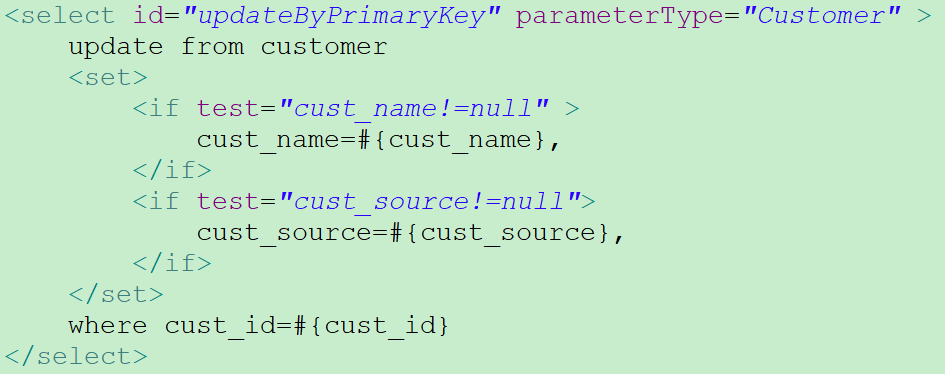


1. **where标签**



where标签可以自动添加where，同时处理where关键字后面的第一个and关键字。所以我们在每一个条件子句前面加and就行，where标签会自动删除第一个and

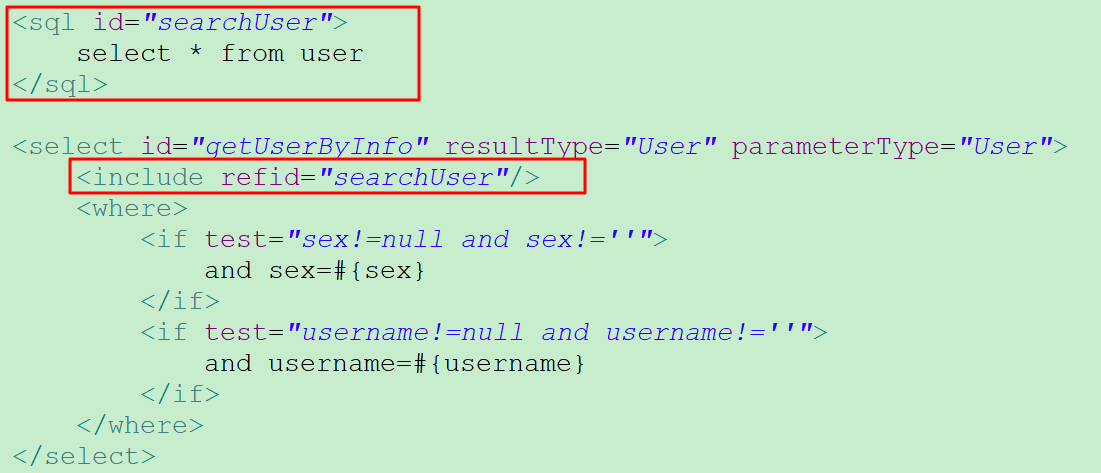
1. **set标签**



Set标签会自动删除最后的逗号，所以我们在每个子语句后面都加上逗号就行

1. **sql标签**

Sql中可将重复的sql提取出来，使用时用include引用即可，最终达到sql重用的目的。



如果要使用别的Mapper.xml配置的sql片段，可以在refid前面加上对应的Mapper.xml的namespace

1. **foreach标签**

现在有这么一个需求：我们想传入多个Id，然后查询这些Id对应的User

**foreach标签属性：**

collection：需要遍历的集合名称

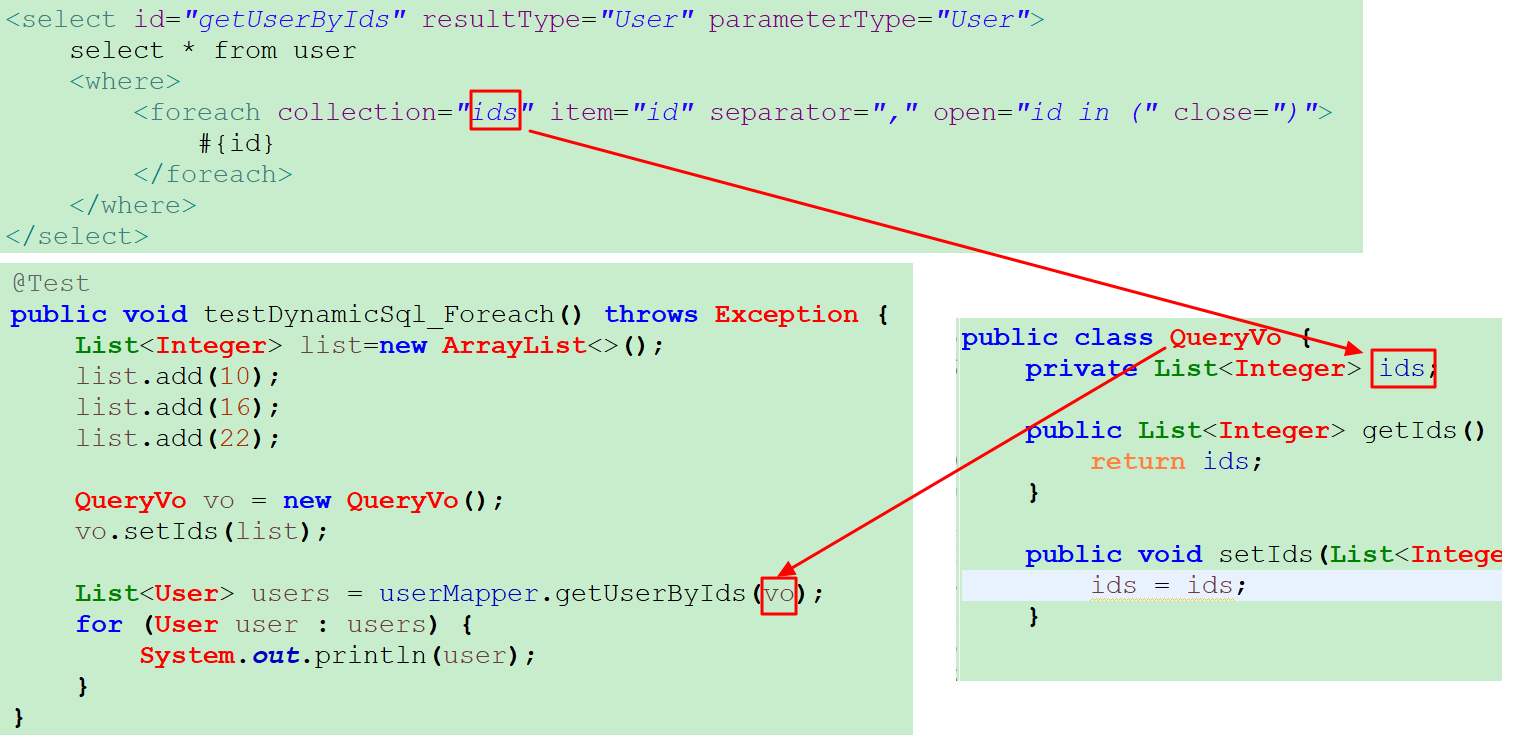
item：每次遍历得到的Item所保存的变量名

open：在前面添加的sql片段

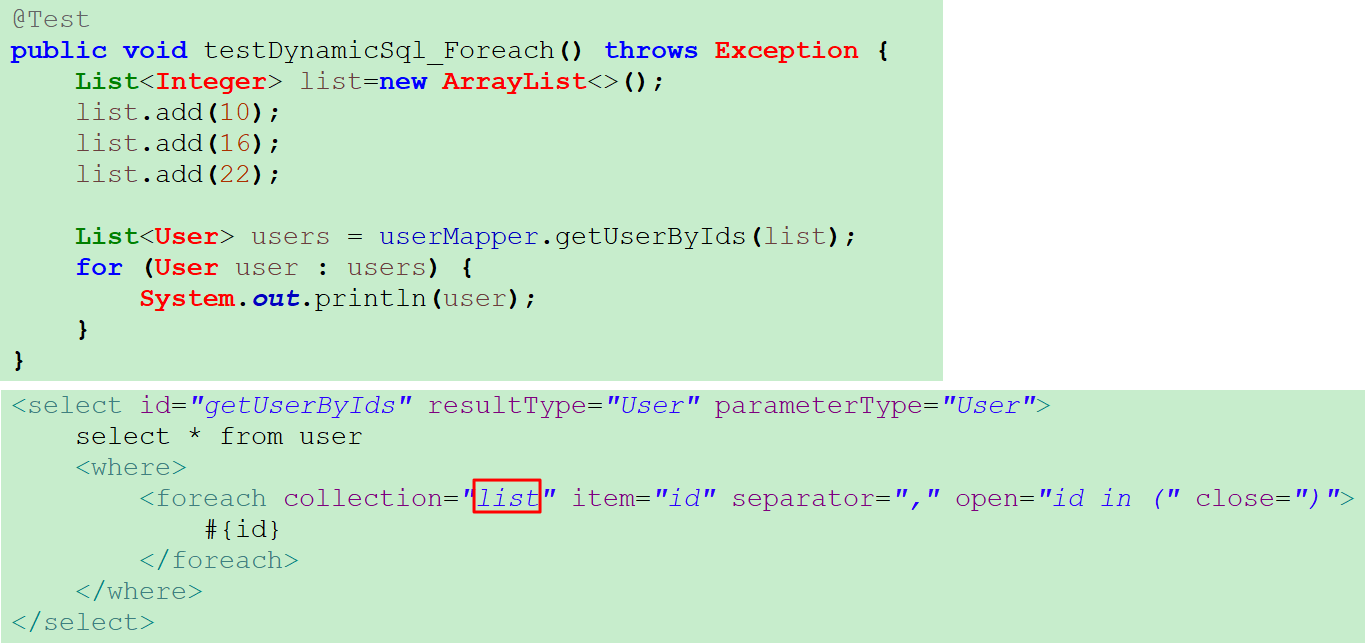
close：在结尾处添加的sql片段

separator：指定遍历的元素之间使用的分隔符

**遍历封装在对象里的集合**



**遍历集合**



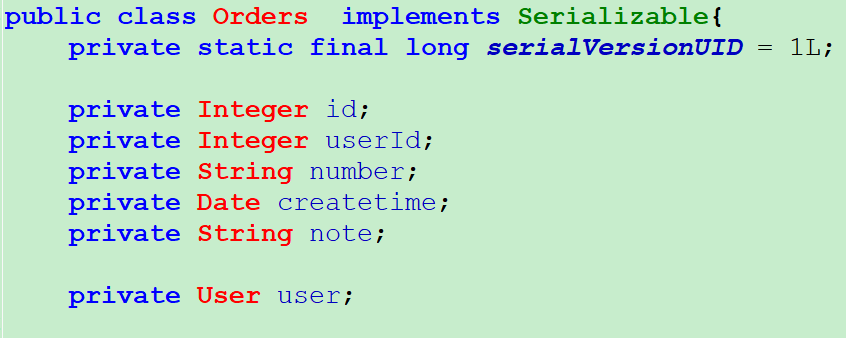
**遍历数组**

与遍历集合一样，只不过在collection属性中填array即可。

### 多表关联

**一对一映射**

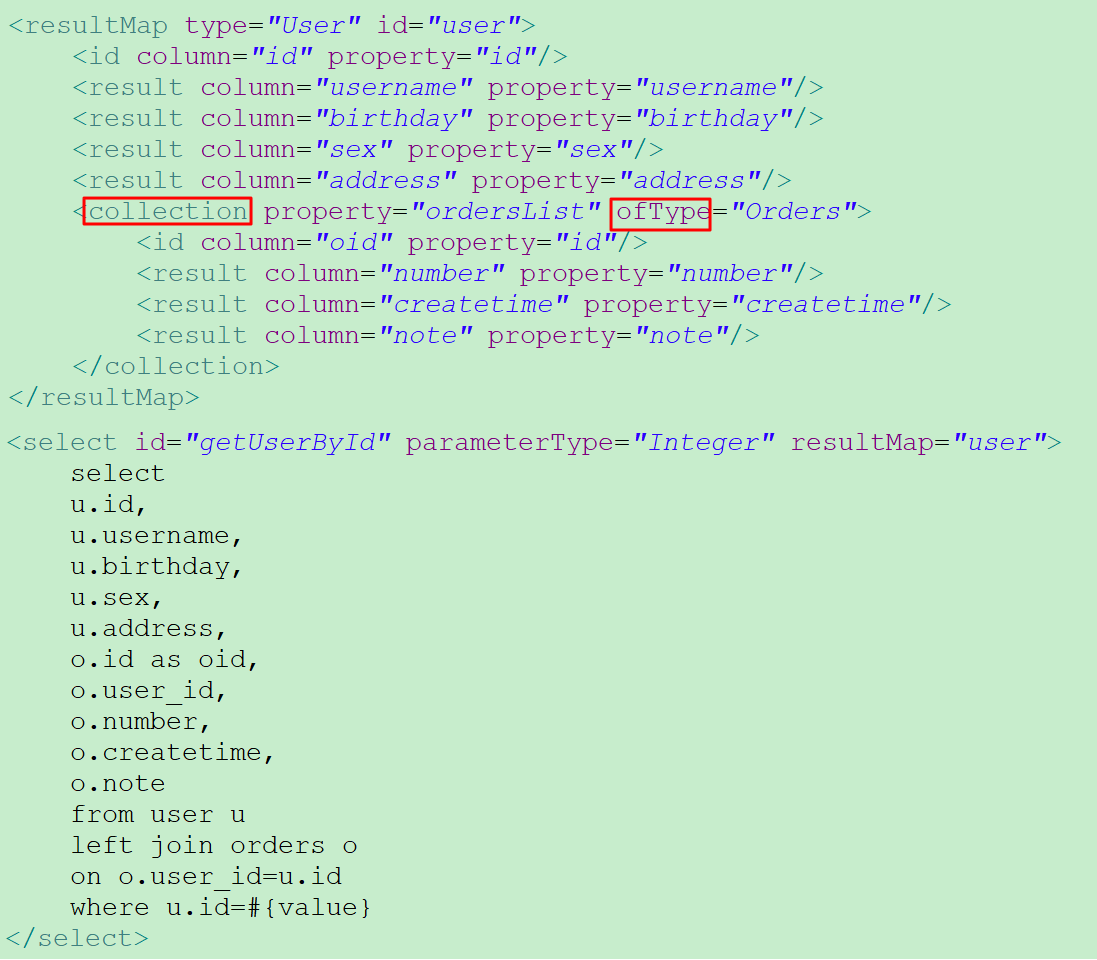


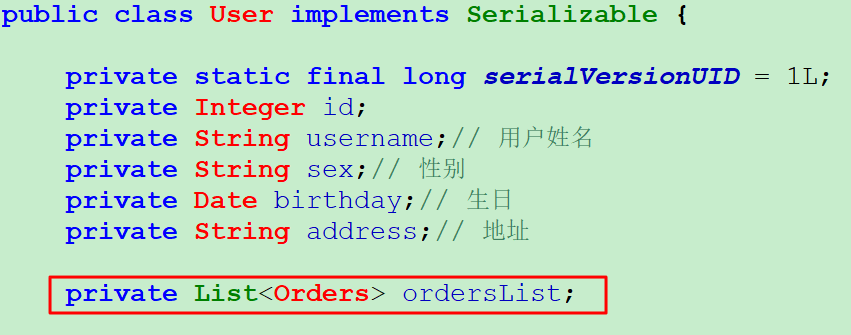


association标签：用于配置一对一嵌套映射

注意：在单表查询中，我们可以使省略POJO对象中属性与表列名相同的映射，而在一对一关联对象的映射中不能省略，必须把每个属性都一 一对应起来。

**一对多映射**





collection标签：表示一对多映射；

ofType属性：表示在封装类一对多属性中集合泛型的类型。

### MyBatis与Spring整合的思路是什么

1、SqlSessionFactory对象应该放到spring容器中作为单例存在。

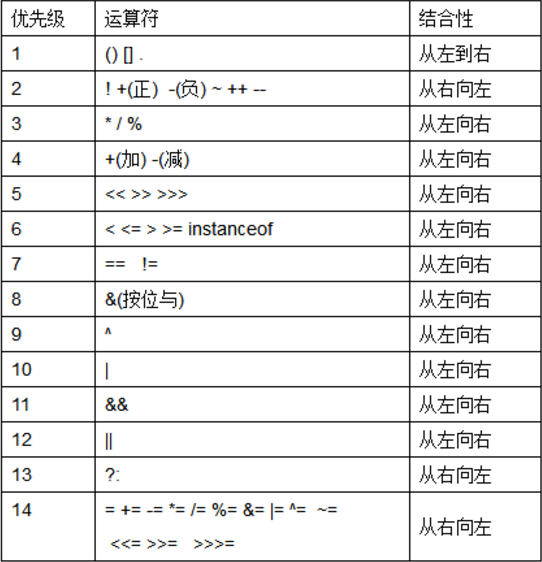
2、传统dao的开发方式中，应该从spring容器中获得sqlsession对象。

3、Mapper代理形式中，应该从spring容器中直接获得mapper的代理对象。

4、数据库的连接以及数据库连接池事务管理都交给spring容器来完成。

## Java基础

### 运算符优先级



### 导包问题

要导入java/awt/event下面的所有类，叙述正确的是？

答：只能是import java.awt.event.\*。

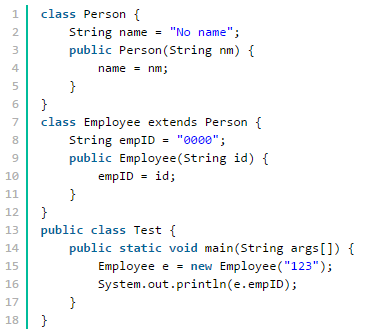
### List，Set，Map是否都继承自继承Collection接口

答：错，Map是一个顶级接口。

### 子类可以继承父类的构造函数？

答：错，只能显示调用父类构造函数。

### 下列代码为什么编译报错？

、

**父类没有无参的构造函数，所以子类需要在自己的构造函数中显式调用父类的构造函数，**  
添加  
super("nm");  
否则报错：  
Implicit super constructor Person() is undefined. Must explicitly invoke another constructor