**达内郑州嵩山路中心java方向各阶段重点知识点(四)** 2017.8.29

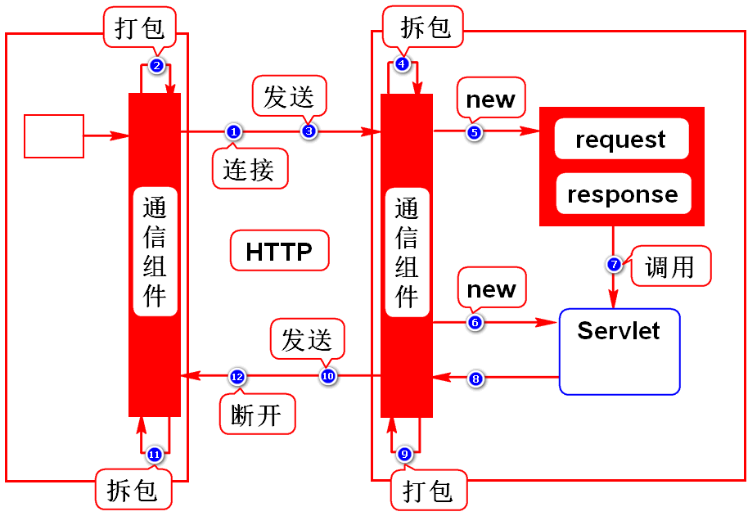
**该文档由达内郑州嵩山路中心java项目经理何老师进行整理,供学员学习使用,其他中心或机构使用请保留该出处,您身边如有想要学习技术的请联系嵩山路java项目经理:何老师,电话:13838585543**

**注:文档中带有 "面" 标识的为重点面试题,未标注的为各阶段重点知识点**

1. **什么是Servlet？**

Servlet是SUN公司推出的在服务器端处理HTTP协议的组件规范.

1. **Servlet的运行原理?(面)**

****

1. 浏览器通过向服务器发送的ip地址,端口号等信息与服务器建立连接
2. 建立连接后浏览器端通信组件将请求数据打包,然后将请求数据发送给服务器.
3. 服务器接收要用户请求数据后服务器端通信组件将数据拆包.
4. 创建request请求对象和response响应对象.
5. 创建Servlet实例对象后调用该实例对象的doGet/doPost/service方法传入request请求对象和response响应对象.
6. request请求对象接收请求信息数据,处理完毕后由response响应对象将处理结果进行响应.
7. 服务器端通信组件将响应信息打包并发送给浏览器.
8. 浏览器接收到响应信息后通信组件将其拆包并将数据展示在浏览器页面
9. 响应结束与服务器端口连接.
10. **Servlet生命周期? (面)**

**实例化**:在第一次访问或启动tomcat时，tomcat会调用此无参构造方法实例化servlet

**初始化**:tomcat在实例化此servlet后，会立刻调用init方法初始化servlet

**就绪**:容器收到请求后会调用servlet的service方法来处理请求

**销毁**:容器依据自身算法删除servlet对象，删除前会调用destroy方法

其中实例化，初始化和销毁只执行一次，service方法执行多次,默认情况下servlet 是在第一次接收到用户请求的情况下才会被实例化,可以在web.xml中的<servlet></servlet>标签内添加一个<load-on-startup>1</load-on-startup>配置,此时在启动tomcat时就会创建servlet实例.

1. **如何实现一个自定义的servlet? (面)**

1）写一个Servlet类extends HttpServlet 并重写doPost或doGet或service方法2）在web.xml中进行部署

<servlet>

<servlet-name>别名</servlet-name>

<servlet-class>包名.项目名</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>别名</servlet-name>

<url-pattern>/网络路径名</url-pattern>

</servlet-mapping>

1. **HttpServlet抽象类中有两个service()的区别?**

public void service(ServletRequest req, ServletResponse res){}

protected void service(HttpServletRequest req, HttpServletResponse resp){}

第一个访问权限public的service方法是有tomcat调用的, 它将接收的客户端请求转交给HttpServlet中的第二个访问权限为protected的service方法.

第二个访问权限为protected的service方法再把将请求依据请求类型将get请求分发给doGet（）方法,将post请求分发给doPost（）方法进行处理.

当我们自定义一个Servlet时继承HttpServlet重写的是访问权限为protected的service()方法来替代doGet()和doPost()方法.因为只会调用重写后的方法.

1. **重写HttpServlet抽象类中doGet(),doPost(),service()方法后使用中的区别?**

若自定义一个Servlet时继承HttpServlet抽象类时,如果重写doGet()和doPost()方法后,会先执行HttpServlet类中的两个service方法,然后根据请求的类型调用我们自定义类中重写的doGet和doPost方法,若请求的类型是get请求,将会执行重写后的doGet()方法,若请求类型是post请求,将会执行重写后的doPost()方法.

若在自定义一个Servlet时继承HttpServlet抽象类时,重写了访问权限是protected的service方法,那么会执行HttpServlet类访问权限是public的service,然后在调用我们重写后的service方法,不再将请求分发给doPost和doGet方法,因为我们自己重写的service方法即可处理get请求也可以处理post请求.

1. **servlet中不同状态码对应的错误问题? (面)**

200:请求成功

404:找不到请求资源:

请求地址错误,web.xml文件中两个servlet-name不一致,工程未部署

405:web服务器找不到service方法

service方法名,参数类型,返回值类型,异常等有误

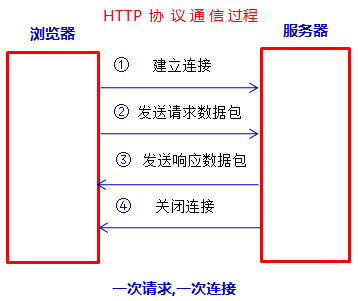
500:程序运行出错

未继承HttpServlet抽象类,web.xml文件中servlet-class类路径名错误,service

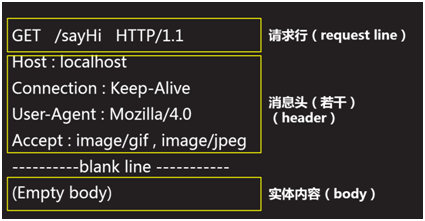
方法中代码有错误

1. **Http协议是什么?**

Http协议也叫超文本传输协议:由w3c制订的一种网络应用层协议，规定了浏览器与web服务器之间如何通信及相应的数据格式。



**请求数据包:Request Message**

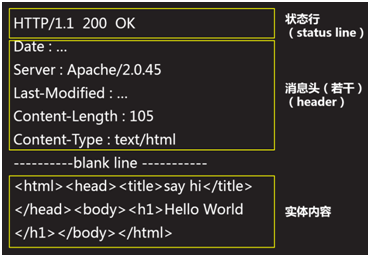


请求行:包含请求类型GET/POST、请求路径、协议类型/版本

消息头:访问主机的IP和端口号,浏览器版本,接受的资源类型等

实体内容:GET请求时为空,POST请求时为请求时，实体内容为要提交的表单数据

**响应数据包:Response Message**



状态行: 包括协议类型/版本、状态码、状态描述

消息头:响应给浏览器的相关信息:响应数据类型,数据大小,服务器版本信息,响应时间等

实体内容:响应给浏览器的具体数据,浏览器逐行解析.

1. **浏览器向服务器的请求方式中GET和POST请求有什么区别? (面)**

请求方式分为:get请求和post请求.

get请求:浏览器发送请求默认都是get请求,使用请求路径传参,将参数附加在路径上发送服务器, 由于请求路径是在请求数据包中的请求行中,所以参数在传递过程中可见，隐私性差, 只能携带少量参数(2k左右).

post请求: 在form表单上增加method="post"时,请求方式为post请求, 将参数存放在请求数据包的实体内容中通过实体内容传参, 参数在传递过程中不可见，隐私性好,传递的参数大小没有限制.

一般向服务器索取(查询)数据时用get, 向服务器提交数据时用post.

1. **浏览器向服务器请求时乱码和服务器向浏览器响应时乱码解决方式? (面)**

由于服务器通信组件在拆包和打包时默认是按照ISO8859-1编码进行编解码,若浏览器提交和解析时编码为UTF-8或其他编码,则在解析时会出现乱码.

浏览器向服务器请求时乱码:

get请求:在tomcat的server.xml中65行添加:URIEncoding="UTF-8";

post请求:在获取请求参数的最前边添加request.setCharacterEncoding("UTF-8");

服务器响应给浏览器时乱码:

在响应时添加:response.setContentType("text/html;charset=UTF-8");告知浏览器解码的编码.

1. **servletConfig对象和servletContext对象的区别? (面)**

**在web.xml中有两个配置:**

<servlet>

<servlet-name>log</servlet-name>

<servlet-class>day04.LoginServlet</servlet-class>

<init-param>

<param-name>maxOnline</param-name>

<param-value>2000</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>log</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<context-param>

<param-name>size</param-name>

<param-value>10</param-value>

</context-param>

<init-param>...</init-param>配置在某个<servlet></servlet>标签之内,为某个servlet配置一些初始化参数, web容器在创建servlet实例对象时，会自动将这些初始化参数封装到ServletConfig对象中，并在调用servlet的init(ServletConfig config)方法时，将ServletConfig对象传递给servlet,ServletConfig是一对一的.

在服务端service方法中获取配置的初始化参数方法为:

ServletConfig cfg=this.getServletConfig();

String max=cfg.getInitParameter("maxOnline");

System.***out***.println(max);

<context-param><context-param>配置于<servlet>标签平级,容器启动时会为整个web应用配置一个ServletContext对象,配置的参数为整个web应用中的所有servlet共享,属于一对多.

在服务端service方法中获取配置的参数方式为:

ServletContext ctx= this.getServletContext();

String size=ctx.getInitParameter("size");

System.***out***.println(size);

1. **getAttribute()与getParamer()的区别？(面)**

getAttribute()表示从request等作用域对象范围内取得设置的属性，必须要先通过setAttribute属性值绑定在request对象或者其他作用域对象上，才能通过getAttribute来取得绑定的属性值，设置与取得属性值为Object对象类型 .

request.setAttribute("绑定名",绑定值);

request.getAttribut("绑定名");

getParameter()表示接收参数，参数为页面提交的参数，包括：表单中设置name属性对应的提交参数、URL重写（就是xxx?id=1中的id）传的参数等，接收参数返回的是String类型。

request.getParameter("username");

1. **转发和重定向的区别？(面)**

转发和重定向都是解决WEB组件之间跳转问题的

转发：一次请求，地址不改变，共用1个request对象，只能转发到项目内组件。

request.getRequestDispatcher("emp\_list.jsp").forward(req, res);

重定向：二次请求，向浏览器响应信息时会响应302状态码和要重定向的地址,地址会发生改变，两个请求中不能共享request对象，可以重定向到项目外的组件。

通常查询时候用转发，增删改之后重定向到查询。

response.sendRedirect("findEmp");

1. **JSP和Servlet的区别? (面)**

Servlet主要是控制的处理，如调用业务层，跳转不同的jsp页面。

JSP主要在于页面的显示动态生成页面，可以与html标记一起使用,本质上还是servlet.

JSP在第一次接收到用户请求后会先将JSP编译成.java文件,然后再编译成.class文件,最终被JVM加载执行.

1. **JSP九大内置对象? (面)**



1. **JSP九大内置对象中的4个作用域及作用域对象? (面)**

|  |  |  |
| --- | --- | --- |
| 作用域 | 作用域范围 | 作用域对象 |
| page | 在当前JSP页面有效 | pageContext |
| request | 在当前请求中有效 | request |
| session | 当前会话中有效 | session |
| application | 在所有web应用中有效 | application |

四个作用域范围由小到大依次取值的顺序为:page,request,session,application,

如果在page中未取到值,则会继续在request中寻找取值,若依然未取到值则向下一个对象中寻找,依次类推.

除了默认由小到大的顺序外,也可以指定从某个作用域范围内取值:假设分别从session,request中取值:

sessionScope.绑定名

requestScope.绑定名

1. **JSP页面的标签有哪些？(面)**

<% %>:JSP脚本，能够java代码片段。

<%= %>:JSP表达式，变量结合运算符进行运算。

<%! %>:JSP声明，可声明成员属性和成员方法。

<%@ %>:JSP指令

1. **EL表达式和JSTL标准标签库? (面)**

EL表达式是可以直接在JSP页面使用简化代码的技术,可以从四个作用域对象中取值.

语法:

${ 绑定属性名 }

相当于: request.getAttribute("绑定属性名");

假设在Servlet中定义一个变量,然后将数据绑定在request中,然后转发到JSP页面并通过EL表达式获取绑定的属性值.

int num=10;

request.setAttribute("number",num);

request.getRequestDispatcher("find.jsp").forward(request, response);

在JSP页面可以使用EL表达式获取绑定的属性值:

${number}

结果为:10;

EL表达式还可以在作用域对象中获取绑定的JavaBean对象,数组,集合中的值:

比如有一个Student对象,将其绑定在request对象中,然后转发到JSP页面并通过EL表达式获取值.

Student stu=new Student();

stu.setName("张三");

stu.setAge(10);

request.setAttribute("student",stu);

request.getRequestDispatcher("find.jsp").forward(request, response);

在JSP页面通过EL表达式获取绑定的属性值:

${student.name}

${student.age}

结果为:张三 10

JSTL 是SUN公司开发的JSP标准标签库,是一个JSP标签集合，它封装了JSP应用的通用核心功能.

使用JSTL需要先导入jstl.jar包,然后在JSP页面添加指令,如下:

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*>

使用JSTL可以实现if分支,forEach循环等功能

1)if标签:

if标签的test属性必须是一个boolean类型的值，如果test的值为true，那么执行if

标签的内容，否则不执行,例如:

<c:if test=*"*${stu.sex=='M' }*"*>男</c:if>

<c:if test=*"*${stu.sex=='F' }*"*>女</c:if>

如果test属性的EL表达式表示的性别是M,执行标签中的内容:男

如果test属性的EL表达式表示的性别是F,执行标签中的内容:女

2) choose标签:

choose可以实现类似于java中的if ... else ..多路分支结构

当when标签的test为true时，会执行这个when的内容。当所有when标签的test

都为false时，才会执行otherwise标签的内容,例如:

<c:choose>

<c:when test=*"*${stu.sex=='M' }*"*>男</c:when>

<c:otherwise>女</c:otherwise>

</c:choose>

当test属性所表示的EL表达式为性别是M时执行when标签中的内容:男,否则执行

otherwise标签中的女

3)forEach标签:

forEach可以实现循环遍历, 几个常用属性:

items:要遍历的数据

var:遍历的数据保存的变量

forEach双标签中间写每次遍历要输出的数据

假设在Servlet中声明一个数组,转发到JSP页面,通过forEach标签遍历该数组:

String [] name={"张三","李四","王五"};

request.setAttribute("name",name);

request.getRequestDispatcher("find.jsp").forward(request, response);

在JSP页面接收并遍历该数组:

<c:forEach items=*"*${name }*"* var=*"n"*>

${n }

</c:forEach>

items标签中通过EL表达式获取要遍历的数组,声明一个n接收遍历的结果,在forEach

标签内使用EL表达式输出结果.

1. **Session和Cookie的区别? (面)**

Cookie和Session都是负责状态管理的,因为HTTP协议是无状态协议,需要利用Cookie和Session来记录状态.

Cookie默认保存在浏览器端,随着浏览器的关闭而生命结束,可以调用setMaxAge(int seconds)方法来设置Cookie的生命时长,以秒为单位:

当seconds>0时保存在本地硬盘

当seconds<0时保存在浏览器,默认保存在浏览器

当seconds=0时立刻结束Cookie的生命

另外存储的数据大小只能为4K左右的数据,安全性较低, 别人可以分析存放在本地的Cookie并进行Cookie欺骗.

Session保存在服务器,当第一次用户请求时,服务器为其分配一个内存来保存session对象,并生成一个SID,在响应时将该SID以Cookie的形式发送给浏览器被保存在内存中,当同一用户再次发起请求时会携带SID到服务器,服务器会依据SID来找到对应的session并使用该session对象中保存的数据,由于数据保存的服务器,数据相对安全,但保存的session较多时会影响服务器性能.

session默认在服务器端保存30分钟,可以调用setMaxInactiveInterval（int seconds)方法来设置生命时长,seconds单位为秒.

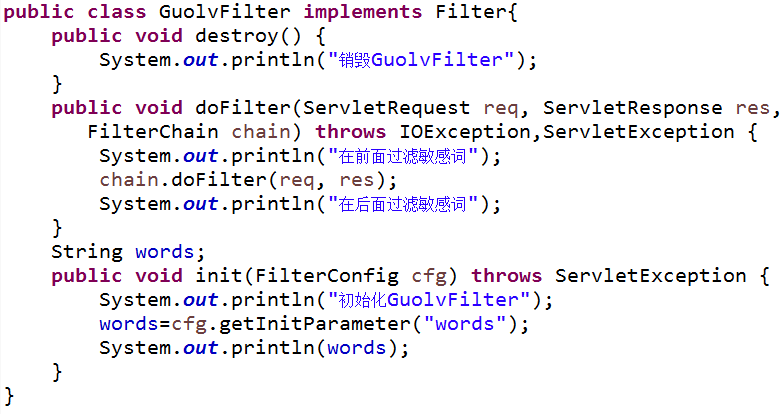
想要删除session可以使用invalidate()进行删除.

1. **什么是过滤器? (面)**

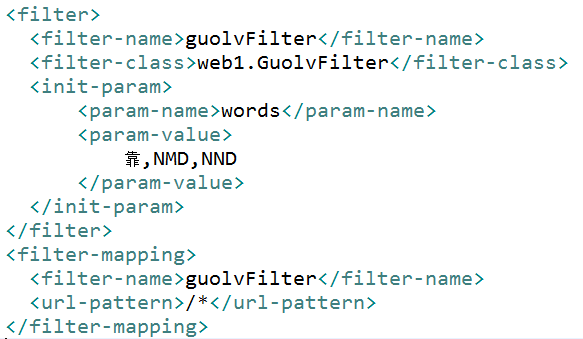
过滤器是Servlet中的一个组件,在Servlet之前执行,由于执行时机比Servlet要早,所以可以用来做一些过滤验证,比如:记录日志,过滤敏感词,权限检查等.

**自定义一个过滤器步骤:**

1.写一个类实现Filter接口,重写init, doFilter, destroy方法,其中init是初始化方法,doFilter是过滤逻辑方法,destroy是销毁过滤器方法.



2.在web.xml中写过滤器的配置信息



如上案例执行过程:

1.实例化:在启动服务器时会创建Filter实例,

2.初始化:在实例化后会调用init方法来初始化Filter,如果配置的有init-param初始化参数,则会把初始化参数保存在FilterConfig对象中,调用带参数的init方法来传递初始化参数.

3.就绪执行:初始化完毕后会调用doFilter方法来执行具体的过滤逻辑,doFilter中的FilterChain chain是过滤链,通过chain.doFilter(req,res)来向后调用servlet或者其他Filter,调用的servlet或者其他Filter执行完毕后不会立即将响应返回给浏览器,而是返回到doFilter,由doFilter来进行响应.

4.销毁:请求完毕会调用destroy来销毁过滤器.

注意:如果有多个过滤器的话,会按照web.xml中的配置顺序由上至下依次执行.

如果是过滤器和Servlet同时存在,会先执行过滤器,再执行Servlet.

1. **MVC模式？**

**M Model模型**

模型（Model）负责业务逻辑，包含两层：业务数据和业务逻辑，比如实体类，DAO，Service属于模型层。

**V View视图**

视图（View）负责显示界面和用户交互，收集用户信息，属于视图的组件不包含业务逻辑和控制逻辑的JSP。

**C Controller控制器**

控制器属于模型层M和视图层V之间的桥梁，用于控制流程，比如：在Servlet项目中的单一控制器ActionServlet.

1. **java设计模式的分类?**

总体来说设计模式分为三大类，共计23种设计模式：

**创建型模式**，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

**结构型模式**，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

**行为型模式**，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

1. **设计模式之单例模式？**

单例模式的核心结构中只包含一个被称为单例的特殊类。通过单例模式可以保证系统中

一个类只有一个实例, 可以减少频繁创建对象对内存的开销,保证核心业务只有一个对

象在进行处理

**单例模式的特点:**

1. 初始化为null私有的静态自身引用，且被private修饰，其它类不得直接访问。

2. 单例模式的类还需要有private的构造方法，这一点不难理解，如果构造方法是public的，那么类外部可以直接调用该类的构造方法，如此一来便不具备单例的特性.

3. 公有的静态get方法,返回单个实例对象给调用者.

**案例：请将如下代码务必写熟练（笔试题）**

1) **懒汉式单例类**:在第一次调用的时候实例化自己

**public** **class** Singleton {

//1.初始化为null私有的静态自身引用

**private** **static** Singleton *single*=**null**;

//2.private的构造方法

**private** Singleton(){

}

//3.public的静态获取方法

**public** **static** Singleton getSingleton(){

//如果当前的实例为null则创建对象

**if**(*single*==**null**){

*single*=**new** Singleton();

}

**return** *single*;

}

}

2) **饿汉式单例类**:在类初始化时，已经自行实例化

**public** **class** Singleton1 {

//1.类加载时初始化自行进行实例化

**private** **static** **final** Singleton1 ***sin***=**new** Singleton1();

//2.私有的构造方法

**private** Singleton1(){

}

//3.公有的获取方法

**public** **static** Singleton1 getSin(){

**return** ***sin***;

}

}

3) **线程安全单例类:**对获取方法加锁,但性能稍慢

**public** **class** SingletonSync {

**private** **static** SingletonSync *sin*=**null**;

**private** SingletonSync(){

}

/\*\*对创建对象的方法加锁,保证线程安全,但性能稍慢\*/

**public** **synchronized** **static** SingletonSync getSin(){

**if**(*sin*==**null**){

*sin*=**new** SingletonSync();

}

**return** *sin*;

}

}

}

4) **双检锁线程安全单例类:**使用同步块并进行双重判断

**public** **class** SingletonSync {

**private** **static** SingletonSync *sin*=**null**;

**private** SingletonSync(){

}

/\*\*该方法可保证线程安全下的性能问题\*/

**public** SingletonSync getSin1(){

**if**(*sin*==**null**){

**synchronized**(*sin*){

**if**(*sin*==**null**){

*sin*=**new** SingletonSync();

}

}

}

**return** *sin*;

}

}

1. **设计模式之工厂设计模式？**

小故事：话说十年前，有一个暴发户，他家有三辆汽车——Benz奔驰、Bmw宝马、Audi

奥迪，还雇了司机为他开车。不过，暴发户坐车时总是怪怪的：上Benz车后跟司机说“开

奔驰车！”，坐上Bmw后他说“开宝马车！”，坐上Audi说“开奥迪车！”,你一定说：

这人有病！直接说开车不就行了？！

通过以上小故事可以看出,如果按照这种方式去设计程序是存在弊端的,为了解决这种弊端

我们可以使用工厂模式,最终解决的方案就是:暴发户直接说开车就行了.

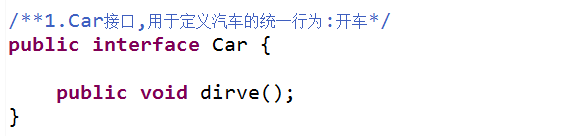
**工厂模式分为:普通工厂模式,抽象工厂模式,工厂方法模式.**

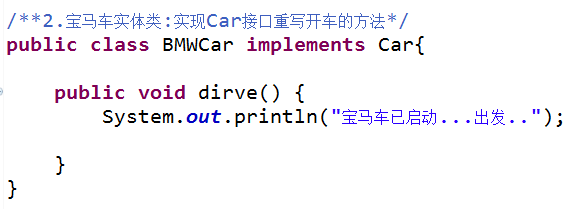
如下以普通工厂模式模式(也叫普通静态工厂模式)来解决以上小故事的弊端:

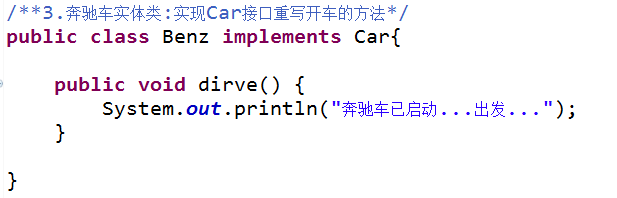
1.创建一个Car接口,封装开车的方法,让老板下达统一的开车命令,而非下达开某种车命令

2.创建不同品牌的具体汽车实例类来实现Car接口,重写开车的方法

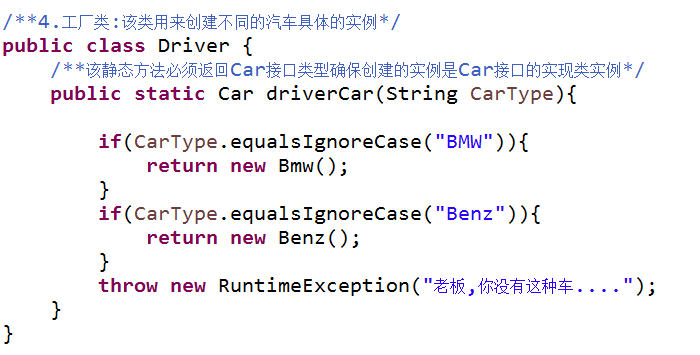
3.创建一个工厂类,本案例的核心,通过该工厂类来实例化不同品牌汽车的具体实例

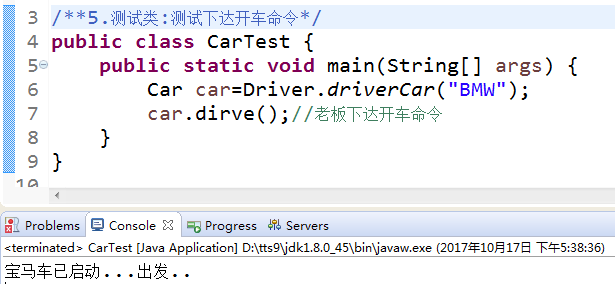






**重点出场:工厂类**

****

****

1. **Spring是什么? (面)**

是一个开源的用来简化开发的应用程序开发框架, Spring帮我们管理对象之间的关系，便于解耦和程序的维护, Spring还可以集成其它的一些框架(比如，可以集成mybatis等),简单的说,Spring可以实现简化开发,解耦,集成.

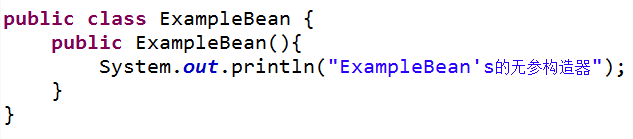
1. **Spring容器是什么? (面)**

Spring容器是Spring框架的一个核心模块，用来管理对象.

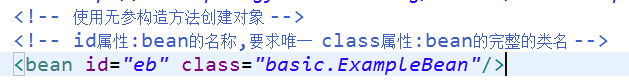
Spring容器有BeanFactory和ApplicationContext两种类型,其中ApplicationContext继承于BeanFactory接口.

使用Spring容器创建管理对象:(无参构造方法创建)

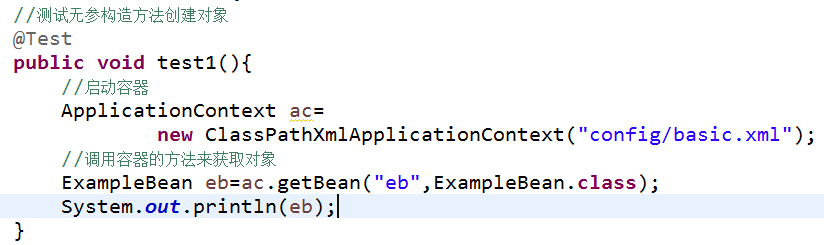
1.先写一个类,并创建无参构造方法



2.在配置文件中写Bean配置



3.启动容器并调用getBean()方法来创建对象



注意:默认情况下，bean的scope属性是单例的，容器只会创建一个实例。

如果设置scope属性="prototype"，调用getBean时，会创建一个新的实例。

默认情况下，容器启动之后，会将所有作用域为"singleton"的bean创建好。如果设置lazy-init属性值为true,表示延迟加载, 即容器启动之后，对于作用域为singleton的bean, 不再创建相应的实例,只有调用getBean方法时才会创建实例,另外关于销毁实例时调用close()时,只有作用域为singleton的bean才能被销毁.

省略:静态工厂方法和实例工厂方法创建对象省略.

1. **什么是IOC和DI? (面)**

假设现在有两个类A类和B类,在A类中execute()方法中调用B类中的f1()方法,以往的操作时在A类中自己去new B()对象,然后通过调用B类的f1()方法,现在不需要自己去new,降低程序之间的耦合度,交给Spring容器来管理对象并将B对象注入到A类中.

IOC是控制反转:对象之间的依赖关系交给容器来建立.

DI是依赖注入:容器通过调用对象的set方法或者构造方法来建立对象之间的依赖关系.

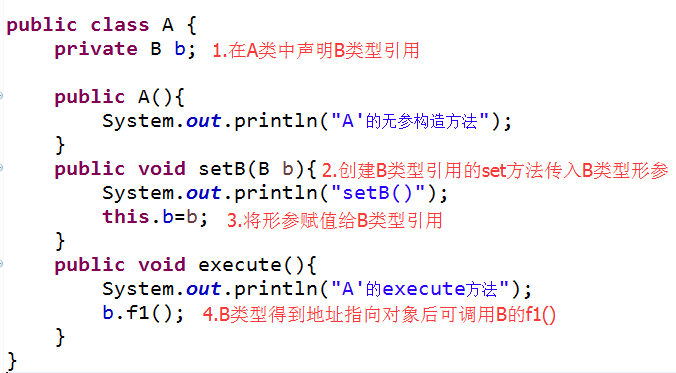
注:IOC是目标,DI是手段.

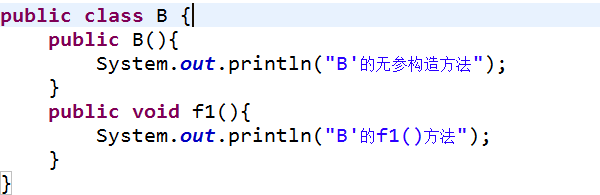
依赖注入的方式有三种:

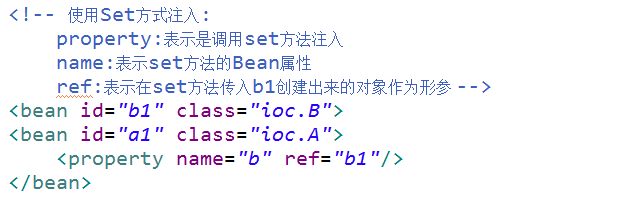
1.通过set方法注入 2.通过构造方法注入 3.通过自动装配

1)set 方法注入:

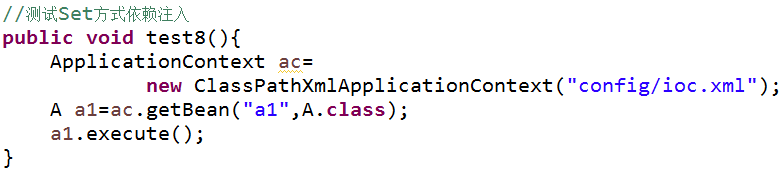
假设现在有两个类A类和B类,在A类中execute()方法中调用B类中的f1()方法,需要在A类中声明B类型的引用,在创建该引用的set方法,并声明B类型参数,在set方法内部将参数赋值给B类型的引用.



写配置文件:



启动容器创建对象并调用方法:



执行过程:

1.启动容器是先读取配置文件,将singleton的bean创建实例.

2.读配置文件时发现有property标签则取找该bean的set方法,具体找那个set方法看name的Bean属性值,并且需要通过ref来执行在该set方法中传入的谁的实例作为参数,比如:setB(B b):则是调用setB方法传入B的引用作为形参.

3.setB(B b)方法内部将形参b的引用赋值给引用类型变量this.b

4.this.b得到对象的地址指向对象.

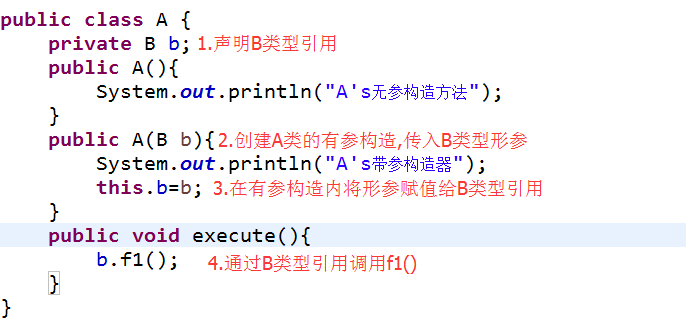
5.通过容器的getBean()方法,传入bean id和类对象,在容器中找到对应的实例返回.

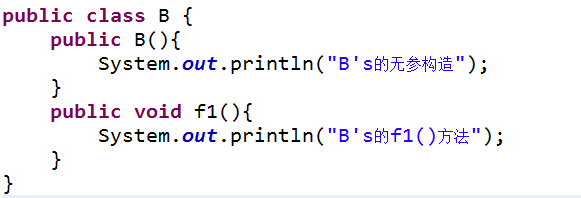
6.通过实例的引用调用execute()方法,方法内部通过this.b引用调用了f1()方法

以上配置过程繁琐,后期可以被@Resource注解替换,简化开发.

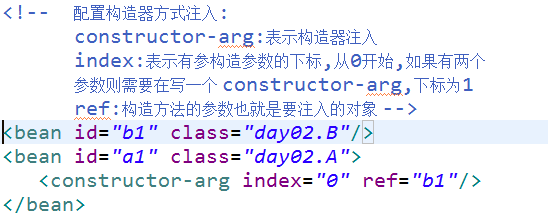
2)构造器注入:

假设现在有两个类A类和B类,在A类中execute()方法中调用B类中的f1()方法,需要在A类中声明B类型的引用,创建A类的有参构造器,并声明B类型参数,在构造方法内部将参数赋值给B类型的引用.





写配置文件:



启动容器进行测试:省略

执行过程:

1.启动容器是先读取配置文件,将singleton的bean创建实例.

2.读配置文件时发现有constructor-arg标签则取找该bean的有参构造方法,ref是要注入的对象,注入给谁参考index的下标.

3.有参构造方法内部将形参b的引用赋值给引用类型变量this.b

4.this.b得到对象的地址指向对象.

5.通过容器的getBean()方法,传入bean id和类对象,在容器中找到对应的实例返回.

6.通过实例的引用调用execute()方法,方法内部通过this.b引用调用了f1()方法

以上配置过程繁琐,后期可以被@Autowired注解替换,简化开发.

3)自动装配:(了解)

默认情况下，容器禁止自动装配。

设置autowire属性让容器自动装配。比如:autowire="byType"

byName:容器查找id等于属性名称的bean,然后调用对应 的set方法来完成注入。

注意：如果找不到对应的bean, 会注入null值。

byType:容器查找与属性类型一致的bean,然后调用对应 的set方法来完成注入。

注意：如果找不到对应的bean, 会注入null值。如果找到多个对应的bean,会出错。

constructor：类似byType,只不过调用的是构造器来 完成注入。

1. **什么是组件扫描?**

容器会扫描指定包及其子包下面的所有的类，如果该类包 含有特定的注解(比如@Component),则容器会将该类纳入 容器进行管理,创建对象并管理对象.

注:相当于配置文件当中有一个<bean id=" " class=" "/>的配置。

组件扫描注解有4个,写在类名上面:

@Component 通用组件

@Service 业务层组件

@Respository 持久层组件

@Controller 控制层组件

需要在配置文件当中，配置组件扫描。

<context:component-scan base-package="包名"/>

1. **依赖注入的相关注解有哪些?**

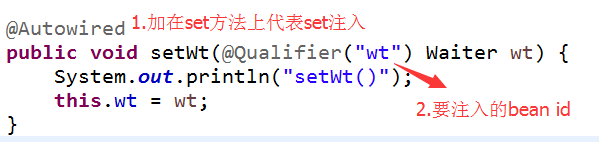
**1) @Autowired和@Qualifier**

此注解支持set注入和构造器注入两种方式:

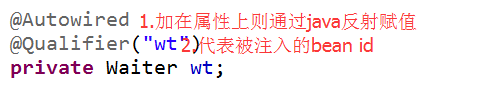
1.1)set方式注入:

1.1.1) @Autowired加在set方法上代表set注入

@Qualifier指明要注入的bean id



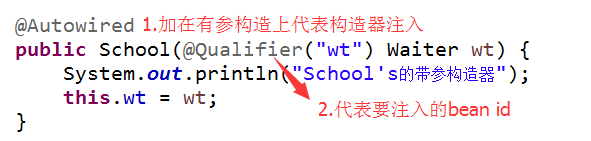
1.1.2) @Autowired加在属性上: 加在属性上则是通过java反射直接赋值. @Qualifier指明要注入的bean id



1.2)构造器注入

@Autowired加在构造方法上代表构造器注入

@Qualifier指明要注入的bean id



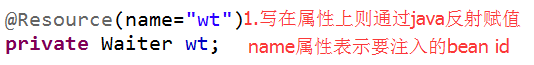
注: set注入支持加上属性上和set方法上和构造方法上,加在属性上则是通过java反射直 接赋值.

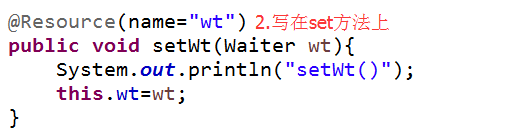
另外需要注意:若不写@Qualifier则默认按照属性的类型byType注入.

**2) @Resource**

此注解仅支持set注入,可加在属性上,也可以加在set方法上

加在属性上通过反射直接赋值,加在set方法上通过byName方式注入,若name属性不写,默认先按照ByName进行查找,如果没有找到会再按照ByType方式注入.



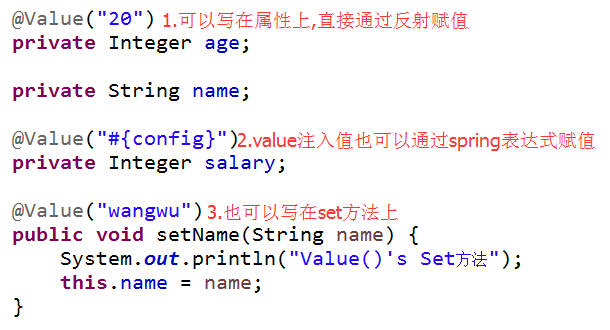


1. **@VALUE注解的作用?**

@VALUE可以设置在属性和set方法上,可以给基本类型(spring对基本类型的定义包含

我们所说的引用类型)属性注入值

VALUE要注入的值也可以通过spring表达式获得再注入.



1. **SpringMVC的五大核心组件及工作流程?**

**五大核心组件分别是:**

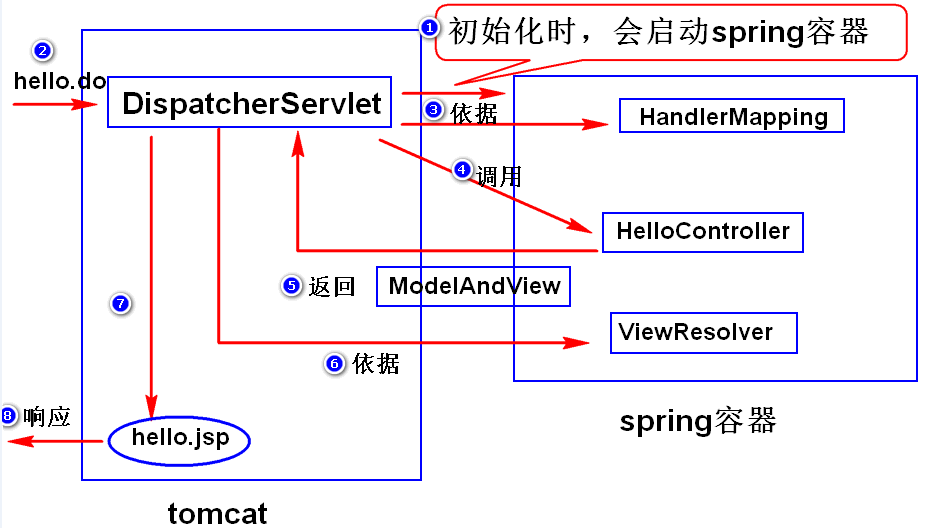
DispatcherServlet (前端控制器)

HandlerMapping (映射处理器))

Controller (处理器)

ModelAndView

ViewResolver 视图解析器

****

**工作流程:**

1.在tomcat启动时会实例化一个DispatcherServlet对象,该对象负责接收请求. 2.DispatcherServlet接收到用户请求后依据HandlerMapping中的配置路径与请求 路径进行比对,比对成功调用相应的Controller进行业务处理.

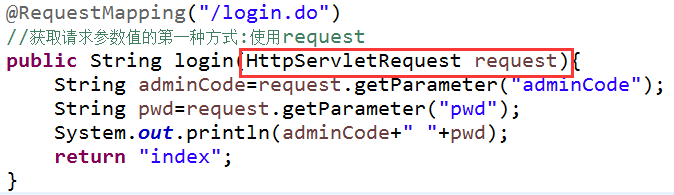
3.处理完毕后将结果封装到ModelAndView对象中(视图名和响应数据)返回给

DispatcherServlet, DispatcherServlet依据ViewResolver的解析，将视图名与 ViewResolver视图解析器中的前缀和后缀进行拼接得到一个具体的视图信息.比如 hello.jsp, 调用相应的视图对象(比如jsp)来生成相应的页面.

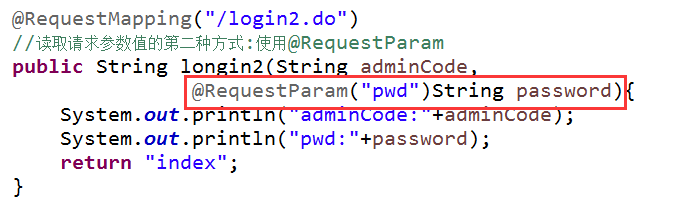
4.然后DispatcherServlet将该页面响应给浏览器.

1. **SpringMVC读取页面请求参数值的方式?**

**方式一：request对象**:只要将request对象作为方法的入参即可。



**方式二：@RequestParam:**将@RequestParam添加到参数前。



注:如果请求参数和浏览器传递的参数一致, DispatcherServlet在调用处理前的方法前

或利用java反射机制分析方法的结构,再调用request.getParamter()方法获取值.

如果请求参数和浏览器参数的参数不一致就需要在参数前加@RequestParam()传入

浏览器参数来和请求参数建立对应关系.最好在每个参数前都加@RequestParam()

**方式三：封装成javabean**:写一个java类，将该java类作为方法的入参即可。要求：

a. 属性名与请求参数名一致 b. 属性有对应的get/set方法。



1. **SpringMVC向页面传递的方式有哪些?**

**方式一: 将数据绑订到request对象。**注：springmvc默认使用转发机制来跳转。



**方式二: 返回ModelAndView对象。**

注:ModelAndView第一个参数为视图名,第二个参数要求传入Map类型数据



**方式三: 将ModelMap作为方法的入参。**

****

注:ModelMap绑定数据调用的是addAttribute("绑定名",绑定值);

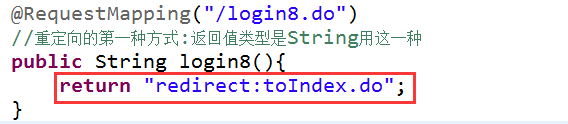
**方式四: 将数据绑订到session对象。**

****

1. **SpringMVC重定向的两种方式?**

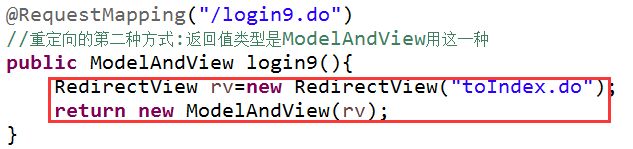
**方式一：如果方法的返回值是String,在重定向地址前添加"redirect:"即可。**

比如 return "redirect:toIndex.do";



**方式二：如果方法的返回值是ModelAndView：**

创建RedirectView重定向视图对象,将该对象封装到ModelAndView中进行重定向.



1. **如何配置Controller基于注解的SpringMVC应用方式？**

1）写一个Controller.java类。

2）在类名前添加@Controller。

3) 在类名和方法名前添加@RequestMapping("路径名")。

4）写xxxx.jsp文件。

5）配置组件扫描：

<context:component-scan base-package="包名"/>

6）配置springmvc注解扫描：让spring容器能够识别@RequestMapping.

<mvc:annotation-driven/>

7）配置视图解析器：不同的视图有不同的解析器类,此处class用省略号表示

<bean class="org.springframeword.web.servlet......">

<property name="prefix" value="/WEB-INF/jsp/">

<property name="suffix" value=".jsp"/>

</bean>

1. **表单中文参数值的乱码问题**

在web.xml文件当中，配置springmvc提供的一个过滤器 (CharacterEncodingFilter)。

a.表单提交方式必须设置为"post" b.过滤器的编码设置与表单的编码要一致。

如下:

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>

org.springframework.web.filter.CharacterEncodingFilter

</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

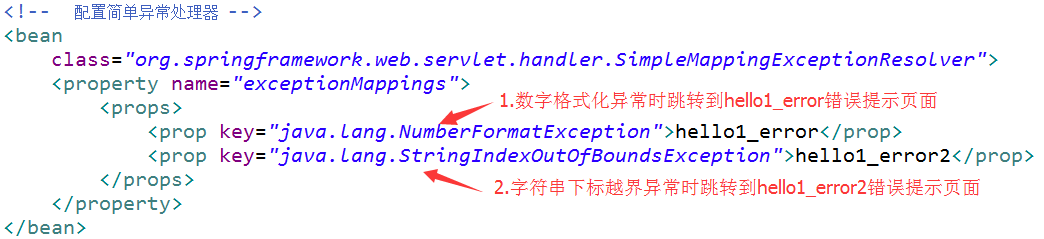
</filter-mapping>

1. **spring框架处理异常的方式?**

将异常抛给spring，由spring框架依据相应的配置，来 处理异常。

方式一： 使用简单异常处理器。

step1. 配置简单异常处理器。



step2. 编写异常处理页面。

hello1\_error1.jsp

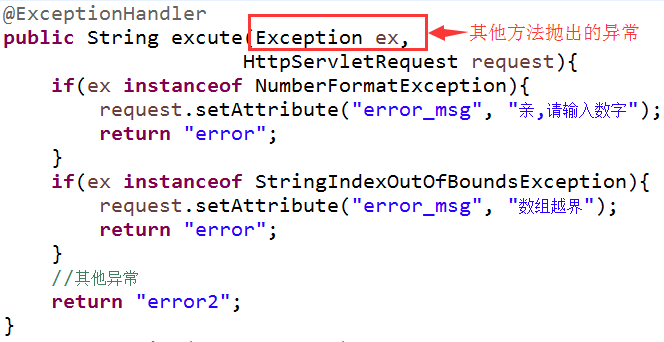
hello1.error2.jsp

方式二: 使用@ExceptionHandler。

step1.创建一个异常处理类,添加一个异常处理的方法,在异常处理方法前面添加@ExceptionHandler。

step2. 在异常处理方法里面，依据异常类型，分别进行 不同的处理。

step3.其他的Controller继承该异常处理类即可.

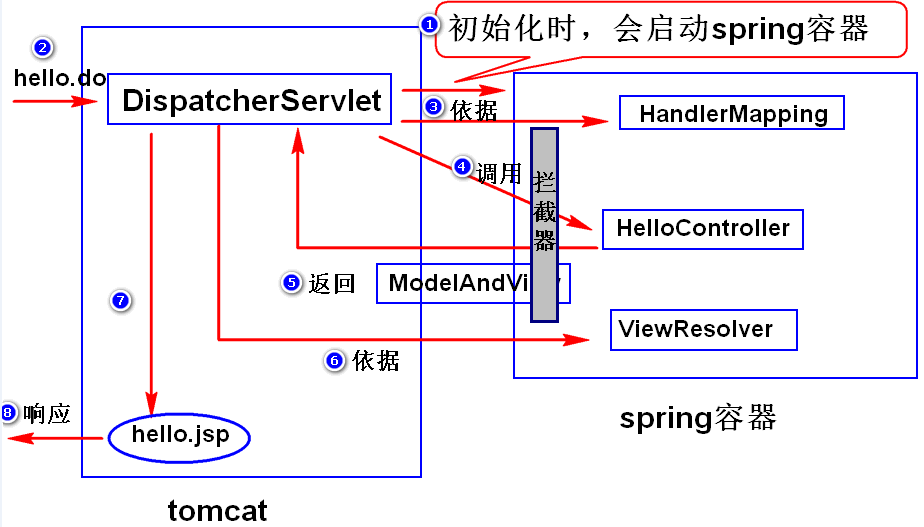


扩展:有同学会问即使写一个公共的父类让所有Controller继承,但是在方法内需要用很多个if分支,上述案例只是演示针对性的异常处理,那么还可以进行简化:如下:



1. **什么是拦截器?**

拦截器 是spring框架定义的组件, Spring的HandlerMapping处理器支持拦截器应用.前端控制器在调用处理器之前，如果发现有拦截器存在， 则会先调用拦截器，然后再调用处理器,可以用来进行身份验证等.

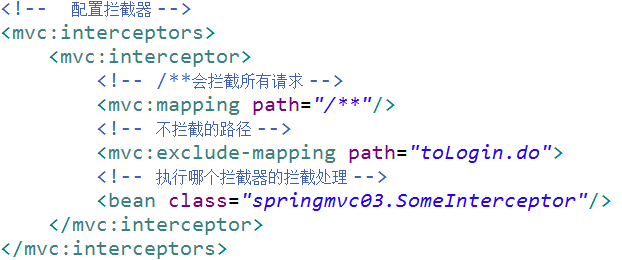
step1. 写一个java类，实现HandlerInterceptor接口。

step2. 将拦截处理逻辑写在相应的接口方法里面:

**preHandle方法**：前端控制器先调用拦截器的preHandle方法， 如果该方法返回值为true,则拦截器继续向后调用；如果 该方法返回值为false,则拦截器不再向后调用，请求处理 完成。

**postHandle方法**：处理器方法已经执行完成，正准备将 ModelAndView对象返回给前端控制器之前执行。可以在 该方法里面，修改ModelAndView对象。**afterCompletion方法**：最后执行的方法, 可以处理器所抛出的异常，如果 preHandle方法返回值为false,该方法不会执行。

step3. 配置拦截器。

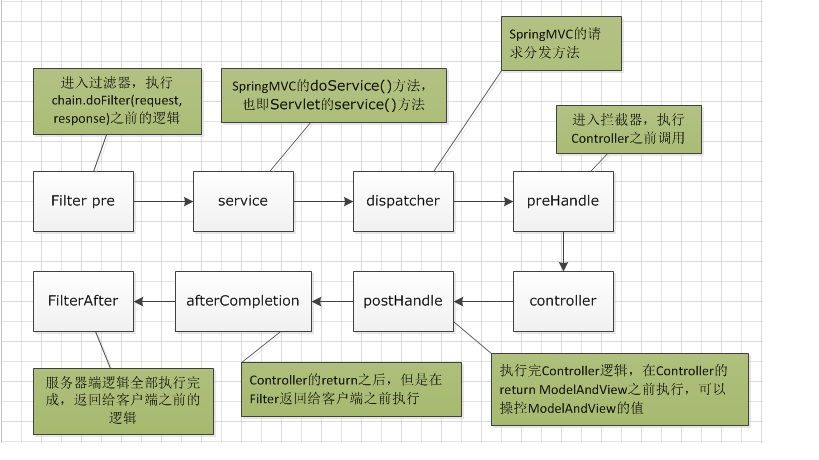


1. **过滤器和拦截器的区别?**

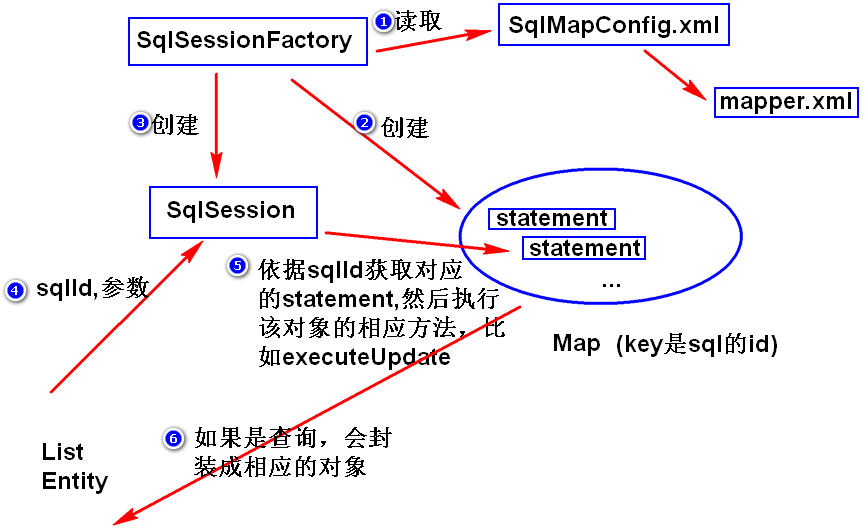
过滤器是Servlet的组件,可以对所有的请求进行拦截,比如.html,.do,.mp3,.jpg等.

拦截器是Spring框架的组件,只能拦截action请求,注:action指的就是xx.do这类请求.

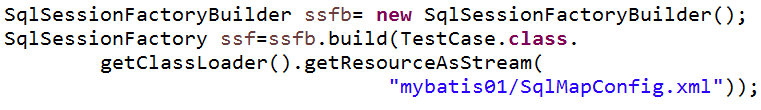
另外还有拦截时机的不同:拦截器,servlet和过滤器,controller同时存在的情况下,运行过程如下图:



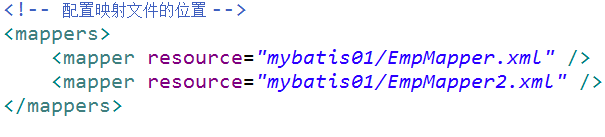
1. **mybatis的运行原理?**

****

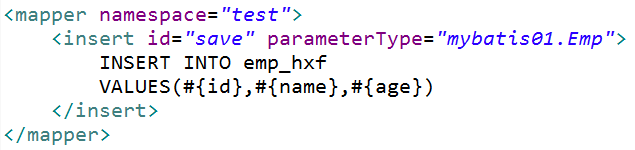
1) 先创建SqlSessionFactory的创建者对象SqlSessionFactoryBuilder,然后再创建SqlSessionFactory工厂对象来读取配置文件(sqlMapConfig.xml),再依据配置文件中的配置找打映射文件(mapper.xml);



2)读取映射文件中的SQL语句创建预编译的PreparedStatement SQL语句,由于预编译的SQL语句较多,会将他们保持到Map中,以sql id作为key,以sql语句作为value



3) SqlSessionFactory工厂对象再调用openSession()返回一个SqlSession对象,该对像封装了执行预编译SQL语句的相关方法.



4)SqlSession调用具体执行方法,传入sql id和参数.



5)依据sql id去Map中找对应sql id的value(预编译SQL),将参数传入预编译的SQL中并调用PreparedStatement对象的excuteUpdate()方法执行该SQL语句.



6)如果是插入,更新操作会返回影响的行数,是个int值

7) 如果是查询语句可以返回list,entity,map,以map为例,则将查询结果保存在结果集Map中,key是表中的字段名,value是字段值,多个返回结果时保存到多个Map中,然后调用结果集的getInt(),getString()等方法从Map中通过key取出value, 然后调用实体对象的set方法进行赋值实现映射关系.



1. **Mybatis映射文件中#{},${}有什么区别?**

select \* from user where name = "dato";

select \* from user where name = #{name};

select \* from user where name = ${name};

#{} 在解析时会解析成JDBC中PreparedStatement预编译sql,防止sql注入:

select \* from user where name = ？;

${}在解析时会解析成字符串参数,类似于JDBC中Statement sql,无法防止sql注入:

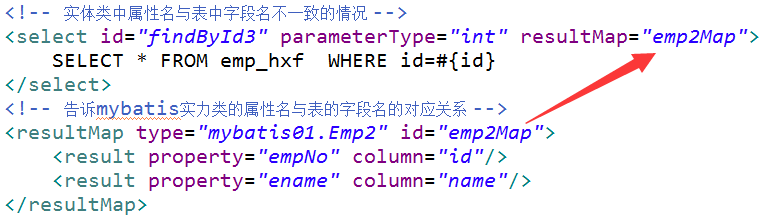
select \* from user where name = "dato";

在程序中能用#{}就不用${},不过有时你只是想直接在 SQL 语句中插入一个不改变的字符串,比如一个数据对象中的表名等,或者ORDER BY排序时可以使用${}.

1. **使用MyBatis时实体类属性和数据库表字段名不一致解决方式?**

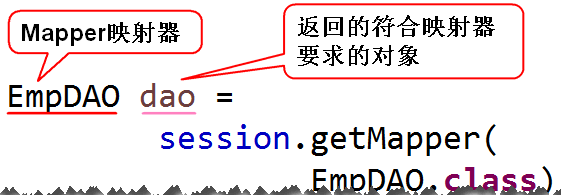
1)使用别名

2)使用resultMap属性



1. **什么是Mapper映射器?**

Mapper映射器是符合Mybatis特定要求(符合映射文件的要求)的接口， Mybatis会依据这些接口通过SqlSession对象的getMapper 方法来生成相应的实现类对象。

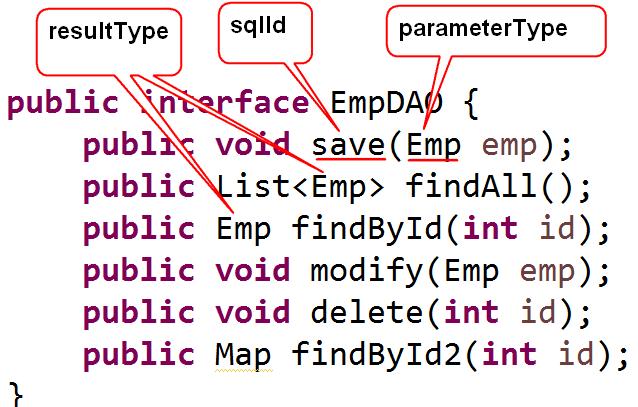


映射器接口与映射文件中SQL语句的要求:

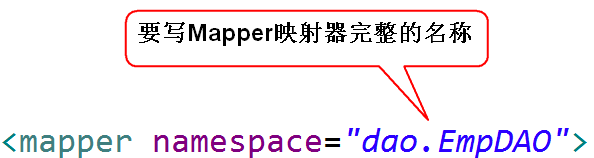
a. 方法名必须与sql语句的id一样。

b. 方法的返回类型与sql语句的返回类型一致。

c. 方法的参数类型与sql语句的参数类型一致。



此外映射文件的namespace必须写Mapper映射器接口的完整的名称:



1. **spring集成mybatis时流程及配置?**

1)导包:spring-webmvc,spring-jdbc,mybatis, spring-mybatis,ojdbc,dbcp,junit。

2)写实体类

3)写映射文件

4)写Mapper映射器接口:可以不写@Repositoty("useDAO")组件扫描

5)添加spring配置文件:

* 配置数据库连接池

<!-- 读取db.properties文件 -->

<util:properties id=*"db"* location=*"classpath:db.properties"*/>

<!-- 配置DataSource -->

<bean id=*"ds"* class=*"org.apache.commons.dbcp.BasicDataSource"*

destroy-method=*"close"*>

<property name=*"driverClassName"* value=*"#{db.driver}"*/>

<property name=*"url"* value=*"#{db.url}"*/>

<property name=*"username"* value=*"#{db.user}"*/>

<property name=*"password"* value=*"#{db.pwd}"*/>

</bean>

* SqlSessionFactoryBean的配置:

该配置的作用是通过连接池与数据库建立连接,由容器创建SqlSession对象来执行映射文件中的SQL语句.

dataSource:指定连接池

mapperLocations:指定映射文件位置, value应该以 "classpath:"开头。

<!-- 配置SqlSessionFactoryBean-->

<bean class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 指定连接池 -->

<property name=*"dataSource"* ref=*"ds"*/>

<!-- 指定映射文件的位置 -->

<property name=*"mapperLocations"*

value=*"classpath:mybatis1/\*.xml"*/>

</bean>

* MapperScannerConfigurer的配置:

该bean负责扫描某个包下面的所有的Mapper映射器， 然后生成相应的实现对象(其实是调用了SqlSession. getMapper方法),并且将这些实现对象放到容器里面, 默认的id是 映射器名(首字母小写）。

<!-- 配置MapperScannerConfigurer -->

<bean class=

*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"basePackage"* value=*"dao"*/>

</bean>

1. **Ajax是什么?有哪些常用属性,方法和事件?**

ajax是一种用来改善用户体验的技术，是利用浏览器提供的XMLHttpRequest对象,也可以称为ajax对象异步地向服务器发送请求,服务器返回部分数据，浏览器利用这些数据对当前页面做部分更新,整个过程，页面无刷新，不打断用户的操作。

**常用属性和事件:**

onreadystatechange:绑订一个事件处理函数，用来 处理readystatechange事件。

注：当readystate的值发生了任何的改变(比如，从0 变成了1），就会产生readystatechange事件。

readyState:有五个值(0,1,2,3,4),表示ajax对象与服务器通信的进展。

0:请求尚未初始化

1:正在发送请求

2:请求发送完成

3:请求成功,正在接收服务器返回数据

4:表示ajax对象已经获得了服务器返回的所有的数据。

responseText:获得服务器返回的文本数据。

responseXML:获得服务器返回的xml数据。

status:获得服务器返回的状态码:200,404,500等

常用方法:

open("请求类型","请求地址",true/false):创建请求,准备发送请求

注:true是异步,false是同步,默认为true是异步请求

send(null):发送请求

注: get请求时null不加也可以,但是个别浏览器不加不行,为了兼容性,还是考虑加上

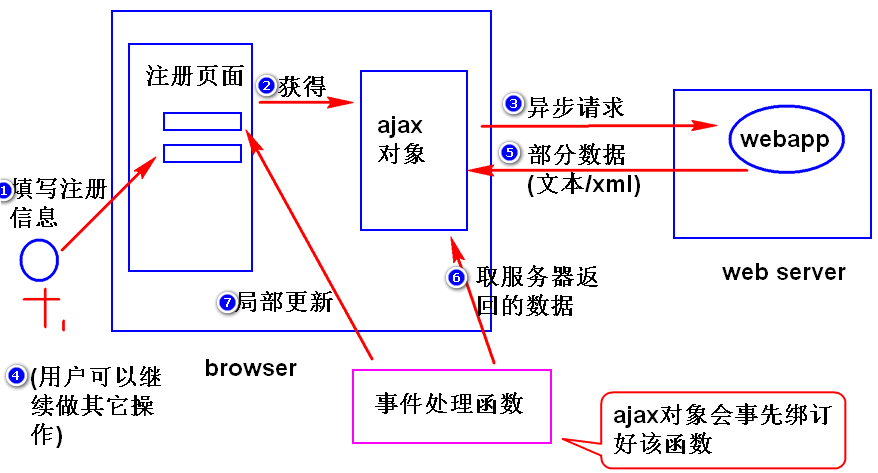
post请求时参数为参数及参数值,比如:send('username='+'zhangsan');

setRequestHeader('content-type',

'application/x-www-form-urlencoded'):指定请求消息头

注:post请求时需要添加该方法来执定请求消息头,get请求时不需要

**运行流程图如下:**



1. **怎么获取Ajax对象?**

由于目前市面上的浏览器类型分为IE和非IE,并且之间不兼容,所以要采取不同的方式获取不同的对象.

**function** getXhr() {

**var** xhr = **null**;

**if** (window.XMLHttpRequest) {

// 非ie浏览器

xhr = **new** XMLHttpRequest();

} **else** {

// ie浏览器

xhr = **new** ActiveXObject(

'Microsoft.XMLHttp');

}

**return** xhr;

}

1. **使用Ajax发起get和post请求的方式?**

**两种方式的区别:**

get请求时请求数据保存请求行,send()方法传入null值, 不用指定请求消息头.

post请求时请求数据保存在实体内容中,send()方法要传入参数和值,需要指定请求消息头.

**get请求方式:**

**function** check\_uname() {

//1.获取ajax对象

**var** xhr=getXhr();

//2.准备发送请求

**var** uri='check.do?adminCode='+$F('adminCode');

xhr.open('get',encodeURI(uri),**true**);

xhr.onreadystatechange=**function**(){

//4.处理服务器返回的数据

**if**(xhr.readyState==4&&xhr.status==200){

//只有当ajax对象已经获得了服务器返回的所有的

//数据,并且服务端没有发生异常

**var** txt=xhr.responseText;

//更新页面

$('username\_msg').innerHTML=txt;

}

};

//3.发送请求

xhr.send(**null**);

}

注: encodeURI():js内置的函数,对中文进行编码(默认utf-8 编码).

**post请求方式:**

**function** check\_uname() {

//1.获取ajax对象

**var** xhr=getXhr();

//2.准备发送请求

xhr.open("post","check.do",**true**);

//添加content-type消息头

xhr.setRequestHeader(

'content-type','application/x-www-form-urlencoded');

xhr.onreadystatechange=**function**(){

//4.处理服务器返回的数据

**if**(xhr.readyState==4&&xhr.status==200){

**var** txt=xhr.responseText;

$('username\_msg').innerHTML=txt;

}

};

//3.发送请求

xhr.send('adminCode='+$F('adminCode'));

}

1. **JSON是什么?**

JSON是一种轻量级的数据交换格式。将要交换的数据转换成一种与平台无关的数据，然后发送给接收方来处理,有解析速度快,JSON文档小等优点.

语法:

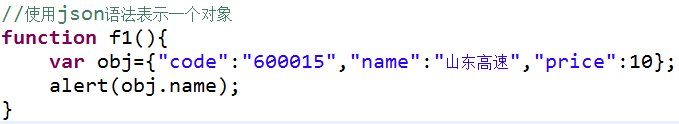
{属性名:属性值,属性名:属性值,....}

注意:

a. 属性名必须使用双引号括起来。

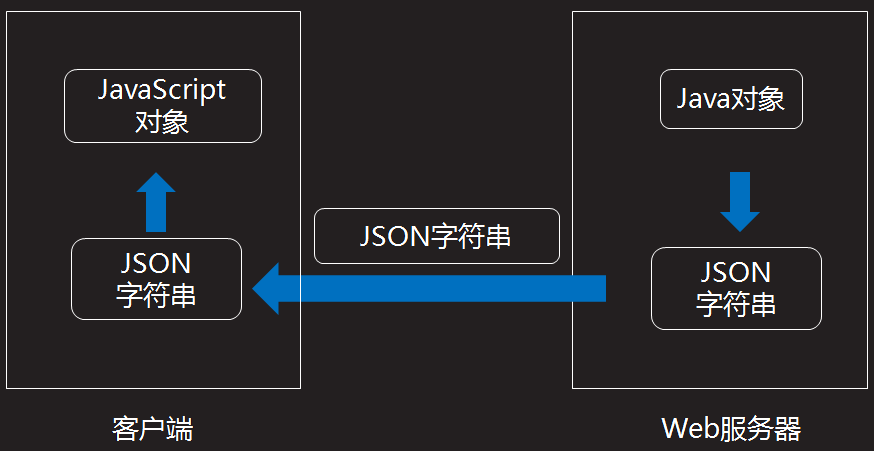
b. 属性值可以string,number,null,true/false,object。

c. 属性值如果是string,必须使用双引号括起来。



在我们的实际应用中,数据交换指的是在服务端将java对象转换成JSON字符串对象,然后将数据发送给客户端浏览器,客户端拿到JSON字符串对象后再将其转换成JS对象.

原理如下图:



**1)将java对象转换成JSON字符串对象:**

相关的API有很多,本案例演示实验jackson的ObjectMapper对象的writeValueAsString();

Stock s=**new** Stock();

s.setCode("600015");

s.setName("山东高速");

s.setPrice(10);

//将java对象转换成json字串

ObjectMapper om=**new** ObjectMapper();

String str=om.writeValueAsString(s);

System.***out***.println(str);

输出结果如下:

{"code":"600015","name":"山东高速","price":10}

另外,也可以将多个对象组成的数组或集合转成JSON字符串:

List<Stock>list=**new** ArrayList<Stock>();

**for**(**int** i=0;i<3;i++){

Stock s=**new** Stock();

s.setCode("60001"+i);

s.setName("山东高速"+i);

s.setPrice(10+i);

list.add(s);

}

//将集合转换成json字符串

ObjectMapper om=**new** ObjectMapper();

String str=om.writeValueAsString(list);

System.***out***.println(str);

输出结果为:

[{"code":"600010","name":"山东高速0","price":10},{......}]

**2)将JSON字符串转换为JS对象:**

使用prototype提供的evalJSON函数来做转换,需要引入prototype-1.6.0.3.js文件

//将json字符串转换成javascript对象

**function** f4(){

**var** str='{"name":"Tom","age":22}';

//使用prototype提供的evalJSON函数来做转换

**var** obj=str.evalJSON();

alert(obj.name);

}

另还有其他方式:

JSON提供的parse():JSON.parse(str);等

1. **JQuery对ajax的支持提供了哪些常用函数?**

**$.ajax()**

用法:  
$.ajax({});其中,{}表示一个对象,用来封装选项参数。 常用的选项参数如下。  
url : 请求地址   
type: 请求类型 (get/post)   
data: 请求参数,有两种格式:  
第一种: 请求字符串，比如 "username=tom&age=20"   
第二种: 对象，比如 {"username":"tom","age":20}   
dataType: 服务器返回的数据类型,常用的有:   
 text: 文本 html: 文档 json: json字符串 xml: xml文档

script: javascript脚本。  
success: 当服务器处理正确，用来处理服务器返回的 数据的函数。  
error:当服务器发生异常，用来处理服务器返回的数据 的函数。  
async: 同步还是异步（缺省是异步）。true表示异步， false表示同步。

**案例如下:**

**function** data(){

$.ajax({url:"demo/hello.do",

type:"post",

dataType:"json",

success:**function**(data){

alert(data);

$("#s1").html(data);

}

});

};

**load():**

作用：

向服务器发送异步请求，将服务器返回的数据直接 添加到符合要求的节点之上。

用法：

$obj.load(url,[data]);

url:请求地址。

data:请求参数。同上。

**案例如下:**

**var** url="team/editUI.do"

$("#modal-dialog .modal-body").load(url,**function**(){

$(".modal-title").html(title);

$("#modal-dialog").modal("show");

});

另外还有:**$.get(); $.post(); $.getJSON();**

以$.get ()为例:

作用:$.ajax的简写方式,作用相同

用法:

$.get(请求路径,请求参数,回调函数,返回数据类型):

其中返回数据类型可选,默认ajax将根据返回的数据智能判断.

**var** url='note/list.do';

**var** param = {notebookId:id};

$.get(url, param, **function**(result){

**if**(result.state==SUCCESS){

**var** notes=result.data;

model.updateNotesView(notes);

}**else**{

alert(result.message);

}

});

1. **Ajax+SpringMVC实现跨域访问?**

**跨域访问**:让不同域名,端口,协议之间的项目可以进行访问.

Ajax本身不支持跨域访问,并且很多浏览器需要遵守**同源策略**,目的是处于安全性考虑,但是实际项目中很多时候需要跨域访问,此时可以借助于JSONP, CORS等技术.

**同源策略**: 就是浏览器限制脚本程序只能和同协议、同域名、同端口的脚本进行交互，这包括共享和传递变量等。cookie的传递也是遵从同源策略.



JSONP:并非JSON,非官方协议,可以通过该协议进行跨域访问,在Ajax中仅支持get请求.

原理:1)在客户端注册callback函数,把callback函数的名字发送给服务器.

2)在服务器生成JSON数据,以JS语法的格式生成一个function

3)将返回的数据以入参的形式放到function中返回.

4)客户端回调函数接收服务端返回的function的中的数据

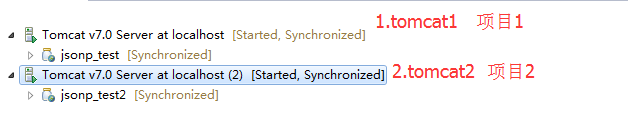
模拟案例如下:

1)分别创建两个项目和开启两个tomcat,tomcat2的端口号等三项信息进行修改,避免和

tomcat1冲突,如下两个网址端口不一致,非同源

http://localhost:8080/jsonp\_test/test\_jsonp.html

http://localhost:8081/jsonp\_test/test\_jsonp.html



2) 在项目1中创建test\_jsonp.html文件.引入jquery-3.2.1.min.js文件,并创建jsonp.js

文件并引入

<script type=*"text/javascript"*

src=*"jquery/jquery-3.2.1.min.js"*></script>

<script type=*"text/javascript"* src=*"js/jsonp.js"*></script>

<body>

<div>

<input type=*"button"* id=*"btn1"* value=*"我能跨域"*/>

<span id=*"show1"*></span>

</div>

<div>

<input type=*"button"* id=*"btn2"* value=*"我也能跨域"*/>

<span id=*"show2"*></span>

</div>

<div>

<input type=*"button"* id=*"btn3"* value=*"post跨域"*/>

<span id=*"show3"*></span>

</div>

</body>

3) 在jsonp.js中创建三个事件函数,分别对应$.ajax,$.getJSON,$.ajax的post请求方式:

$(**function** (){

$('#btn1').click(gotoJsonpTest1);

$('#btn2').click(gotoJsonpTest2);

$('#btn3').click(gotoJsonpTest3);

});

**function** gotoJsonpTest1(){

alert("测试1");

$.ajax({

url:"http://localhost:8081/jsonp\_test2/test1.do",

type:"get",

dataType:"jsonp",

//自定义的jsonp回调函数名,默认为jQuery自动生成的随机函数

jsonpCallback:"jsonpCallback1",

//传递给请求处理程序或页面的,用以获得jsonp回调函数名的参 //数名(默认为callback)

jsonp:"callbackparam",

success:**function**(result){

console.log(result);

console.log(result.age+","+result.name);

$('#show1').html(result.age).css("color","red");

}

});

}

**function** gotoJsonpTest2(){

alert("测试2");

**var** url=

"http://localhost:8081/jsonp\_test2/test2.do?jsopCallback2=?";

$.getJSON(url,**function**(result){

console.log(result);

console.log(result.age+","+result.name);

$('#show2').html(result.age).css("color","red");

});

}

**function** gotoJsonpTest3(){

alert("测试3");

$.ajax({

url:"http://localhost:8081/jsonp\_test2/test3.do",

type:"post",

dataType:"json",

data:{"name":"我叫张三"},

success:**function**(result){

console.log(result);

console.log(result.name);

$('#show3').html(result.name).css("color","red");

}

});

}

3) 在项目2中创建JsonpController.java,并创建三个接收请求的方法来对应jsonp.js中的

请求.

@RequestMapping(value= "/test1",

produces="text/html;charset=UTF-8")

@ResponseBody

**public** String jsonpTest(String callbackparam){

System.***out***.println(callbackparam);

String str="{'age':12,'name':'张三'}";

str=callbackparam+"("+str+")";

System.***out***.println(str);

**return** str;

}

@RequestMapping(value = "/test2",

produces="text/html;charset=UTF-8")

@ResponseBody

**public** String getJson(@RequestParam("jsopCallback2")

String callbackparam){

System.***out***.println(callbackparam);

String str="{'age':15,'name':'王五'}";

str=callbackparam+"("+str+")";

System.***out***.println(str);

**return** str;

}

@RequestMapping(value = "/test3")

@ResponseBody

**public** Map<String,Object> jsonpTest(String

name,HttpServletResponse res){

//用于ajax post跨域（\*，最好指定确定的http等协议+ip+端口号）:

//http://localhost:8080

res.addHeader("Access-Control-Allow-Origin", "\*");

System.***out***.println(name);

Map<String,Object>map=**new** HashMap<String,Object>();

map.put("name", "李四");

System.***out***.println(map);

**return** map;

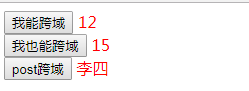
}

注: produces="text/html;charset=UTF-8":是解决get请求返回中文数据乱码

res.addHeader("Access-Control-Allow-Origin", "\*"):是post请求时跨域需要添加的

消息头.

5)测试:最终不管是get还是post请求都可以实现跨域访问了



1. **Spring中的AOP是什么?**

AOP是面向横切面编程,在不改变原有功能的情况下横向织入扩展功能代码或扩展横切功能.