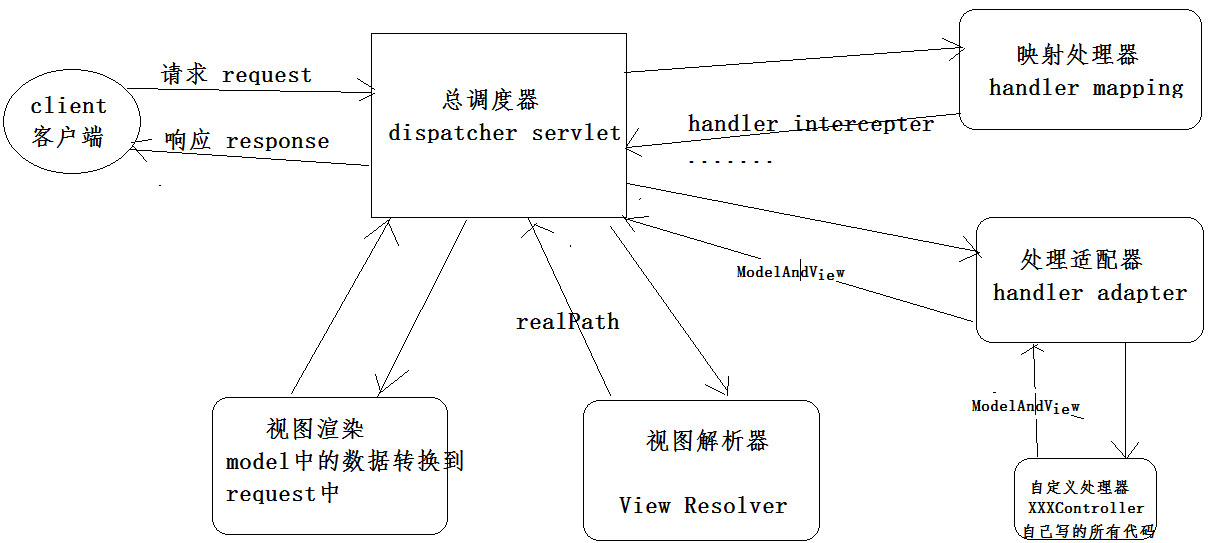
**1，什么是Spring MVC ？简单介绍下你对springMVC的理解?**

Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架，通过把Model，View，Controller分离，将web层进行职责解耦，把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

**2，SpringMVC的工作流程（工作原理）？**



1、客户端发出一个http请求给web服务器，web服务器对http请求进行解析，如果匹配 DispatcherServlet的请求映射路径（在web.xml中指定），web容器将请求转交给DispatcherServlet.

2、DipatcherServlet接收到这个请求之后将根据请求的信息以及HandlerMapping的配置找到处理请求的处理器（Handler）。

3-4、DispatcherServlet根据HandlerMapping找到对应的Handler,将处理权交给Handler（Handler将具体的处理进行封装），再由具体的HandlerAdapter对Handler进行具体的调用。

5、Handler对数据处理完成以后将返回一个ModelAndView()对象给DispatcherServlet。

6、Handler返回的ModelAndView()只是一个逻辑视图并不是一个正式的视图，DispatcherSevlet通过ViewResolver将逻辑视图转化为真正的视图View。

7、Dispatcher通过model解析出ModelAndView()中的参数进行解析最终展现出完整的view并返回给客户端。

**3，请说明一下springmvc和spring-boot区别是什么？**

总的来说，Spring 就像一个大家族，有众多衍生产品例如 Boot，Security，JPA等等。但他们的基础都是Spring 的 IOC 和 AOP，IOC提供了依赖注入的容器，而AOP解决了面向切面的编程，然后在此两者的基础上实现了其他衍生产品的高级功能；因为 Spring 的配置非常复杂，各种xml，properties处理起来比较繁琐。于是为了简化开发者的使用，Spring社区创造性地推出了Spring Boot，它遵循约定优于配置，极大降低了Spring使用门槛，但又不失Spring原本灵活强大的功能。

说得更简便一些：Spring 最初利用“工厂模式”（DI）和“代理模式”（AOP）解耦应用组件。大家觉得挺好用，于是按照这种模式搞了一个 MVC框架（一些用Spring 解耦的组件），用开发 web 应用（ SpringMVC ）。然后有发现每次开发都写很多样板代码，为了简化工作流程，于是开发出了一些“懒人整合包”（starter），这套就是 Spring Boot。

**4，SpringMVC怎么样设定重定向和转发的**

jsp或者servlet可以这样玩：

// 转发 默认带项目名称

request.getRequestDispatcher("url地址").forward(request,response);

//重定向 request.getContextPath() 获取项目名称

response.sendRedirect(request.getContextPath()+"url地址");

（1）转发：在返回值前面加"forward:"，比如"forward:url地址"

（2）重定向：在返回值前面加"redirect:"，比如"redirect:url地址"

区别：

**转发：**

地址栏不发生变化，显示的是上一个页面的地址

请求次数：只有1次请求,跳转发生在服务器端，客户是无感知

根目录：[http://localhost:8080/项目地址/，包含了项目的访问地址](http://localhost:8080/%E9%A1%B9%E7%9B%AE%E5%9C%B0%E5%9D%80/%EF%BC%8C%E5%8C%85%E5%90%AB%E4%BA%86%E9%A1%B9%E7%9B%AE%E7%9A%84%E8%AE%BF%E9%97%AE%E5%9C%B0%E5%9D%80)

请求域(request)中数据不会丢失

重定向：

1. 地址栏：显示新的地址（请求的addDept.do地址栏变化为list.do）

2.请求次数：2次,跳转发生在客户端，客户可以直接看到

3.根目录：<http://localhost:8080/> 没有项目的名字

4.请求域中的数据会丢失，因为是2次请求

**5，SpringMvc怎么和AJAX相互调用的**

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象

具体步骤如下

1.加入Jackson.jar或者fastjson.jar或者其他包

在properties里面加入：

<!-- jackson版本号 -->

<jackson.version>2.8.3</jackson.version>

在dependencies加入：

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>${jackson.version}</version>

</dependency>

2.在配置文件中配置json的映射

在springmvc的配置文件中加入：

<mvc:annotation-driven/>

3.在接受Ajax方法里面可以直接返回Object,List等,但方法前面要加上 @ResponseBody注解

@ResponseBody

@RequestMapping("testReturnJson") @GetMapping @PostMapping @PutM....

public Object testReturnJson(){

/\* Map deptMap = new HashMap();

deptMap.put("deptNo",3);

deptMap.put("dname","开发3部");

deptMap.put("loc","3楼");\*/

List<Map> deptMapList = new ArrayList<>();

Map deptMap1 = new HashMap();

deptMap1.put("deptNo",1);

deptMap1.put("dname","开发1部");

deptMap1.put("loc","1楼");

deptMapList.add(deptMap1);

Map deptMap2 = new HashMap();

deptMap2.put("deptNo",2);

deptMap2.put("dname","开发2部");

deptMap2.put("loc","2楼");

deptMapList.add(deptMap2);

return deptMapList;

}

**6，如何解决POST请求中文乱码问题，GET的又如何处理呢？**

（1）解决post请求乱码问题：

在web.xml中配置一个CharacterEncodingFilter过滤器，设置成utf-8；

<!--解决post乱码-->

<filter>

<filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

（2）get请求中文参数出现乱码解决方法有两个(tomcat7及以前版本)：

①修改tomcat配置文件添加编码与工程编码一致，如下：

（tomcat8以及以后版本，默认的get就支持中文）

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

URIEncoding="utf-8"

 ②另外一种方法对参数进行重新编码：

String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码。

**7，SpingMvc中的控制器的注解一般用那个,有没有别的注解可以替代？**

一般用@Controller注解,spring4.0之后也可以使用@RestController,@RestController注解相当于@ResponseBody ＋ @Controller,表示是表现层,除此之外，一般不用别的注解代替。

**8，怎样在方法里面得到Request,或者Session对象？**

直接在方法的形参中声明request,SpringMvc就自动把request对象传入。

public Object testReturnJson(HttpSession session, HttpServletRequest request,

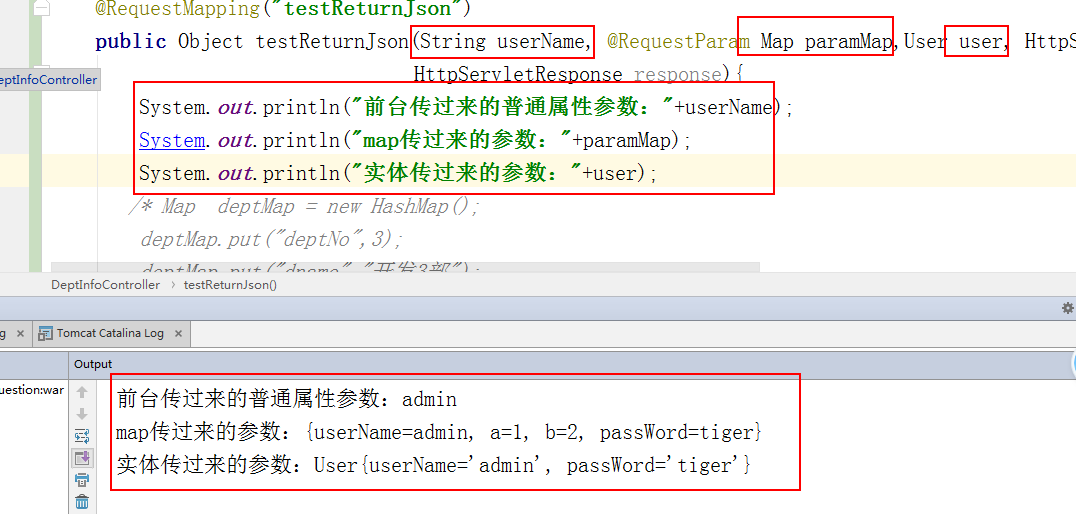
HttpServletResponse response){

**9,如果想在拦截的方法里面得到从前台传入的参数（前台有很多个参数传,入）,怎么得到？**

直接在形参里面声明这个参数就可以,但必须接受名称和传过来的参数名称一样

直接在方法中声明这个对象,SpringMvc就自动会把属性赋值到这个对象里面。

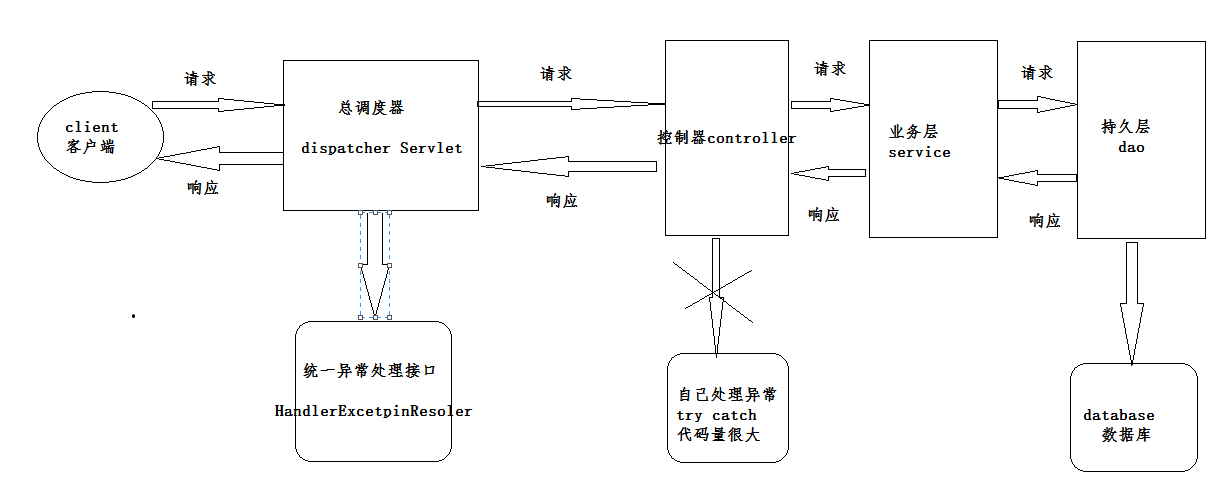
<http://localhost:8080/dept/testReturnJson.do?userName=admin&a=1&b=2&passWord=tiger>



如果使用map或者对象传入参数是ajax请求的json数据，需要使用@RequestBody接受

**10,Springmvc 如何做异常处理**

可以将异常抛给Spring框架，由Spring框架来处理；自定义实现spring的全局异常解析器HandlerExceptionResolver，在异常处理器中添视图页面即可。



在spirngmvc的配置文件中：

<!--统一异常处理-->

<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

<!--所有异常默认的异常处理页面-->

<property name="defaultErrorView" value="error"></property>

<!--异常页面获取异常信息的名称-->

<property name="exceptionAttribute" value="e"></property>

</bean>

和视图解析器配置的前缀和后缀保持一致的位置，写入上面配置的error页面：

对不起，出错了，请联系管理员，管理员联系方式：李工，电话：1666666666。。。。

<%

Exception ex = (Exception)request.getAttribute("e");

out.print("错误描述：<br>"+ex.getMessage()+"<br>");

ex.printStackTrace(new java.io.PrintWriter(out));

%>

**11,控制器方法的一些有效返回类型是什么**

Spring MVC中的控制器方法有许多可用的返回类型，这些方法由控制器内部的@RequestMapping注释。其中一些流行的是:

（1）String

（2）void request.getWrite().print("nologin")

（3）View

（4）ModelAndView (Class)

（5）Model (Interface)

（6）Map

（7）HttpEntity<?> or ResponseEntity<?>

（8）HttpHeaders

可以是objce 任意一个实体类。。。。

每种返回类型都有其特定用途。例如，如果使用的是String，则表示Controller只返回View Name，此视图名称将由ViewResolver解析。

如果不想返回任何视图名称，请提回返回类型void。如果要设置视图名称以及想要发送一些对象，请使用ModelAndView作为返回类型。

**12,Springmvc 中 如果拦截get方式提交的方法,怎么配置？**

如何限制方法只能是get提交 post提交。

可以在@RequestMapping注解里面加上method=RequestMethod.GET

或者使用@GetMapping 不是get提交时就会报406 Method Not Allowed

**13,SpringMvc里面拦截器是怎么写的**

有两种写法,一种是实现HandlerInterceptor接口，另外一种是继承适配器类（HandlerInterceptorAdapter），接着在接口方法当中，实现处理逻辑；

package com.aaa.mvc.interceptor;

import org.springframework.web.servlet.HandlerInterceptor;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* fileName:TestInterceptor

\* description:

\* author:zz

\* createTime:2020/2/17 15:51

\* version:1.0.0

\*/

public class TestInterceptor implements HandlerInterceptor {

@Override

public boolean preHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o) throws Exception {

System.out.println("处理拦截器的业务功能。。。。");

//返回值为true 程序继续运行，为false程序，程序中断

return true;

}

@Override

public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {

System.out.println("在视图渲染之前和进入业务处理方法之后。。。");

}

@Override

public void afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) throws Exception {

System.out.println("在业务处理方法完成后，最终要执行的。。。");

}

}

然后在SpringMvc的配置文件中配置拦截器即可：

<!--拦截器配置-->

<mvc:interceptors>

<!--测试拦截器-->

<mvc:interceptor>

<!--拦截器的拦截范围-->

<mvc:mapping path="/\*/\*.do"/>

<!--拦截器具体实现类-->

<bean class="com.aaa.mvc.interceptor.TestInterceptor"></bean>

</mvc:interceptor>

</mvc:interceptors>

**14, SpringMvc的控制器是不是单例模式,如果是,有什么问题,怎么解决？**

是单例模式,所以在多线程访问的时候有线程安全问题,不要用同步,会影响性能的,解决方案是在控制器里面不能写字段。

如果不想使用单例模式，可以使用前面讲过的注解

@Scope("prototype") //设置为原生模式（非单例），变为原生模式

**15,什么是DispatcherServlet以及它用于什么,DispatcherServlet如何通过应用程序上下文实例化**

DispatcherServlet是Front Controller设计模式的一个实现，它处理对Spring MVC应用程序的所有传入Web请求。前端控制器模式是Web应用程序中的常见模式，其作用是接收所有请求并将其路由到应用程序的不同组件以进行实际处理。DispatcherServlet由Tomcat或Jetty等Servlet容器实例化。必须将DispatcherServlet定义到web.xml文件中，如下所示。

可以看到load-on-startup标记为1，当Spring MVC应用程序部署到Tomcat或任何其他Servlet容器时，将实例化DispatcherServlet。在实例化期间，它会查找文件servlet-name-servlet.xml，然后初始化此文件中定义的bean。

<!--总调度器 开始-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!--拦截所有请求 -->

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

</servlet-mapping>

<!--总调度器 结束-->

不再加载自定义配置文件，而是加载默认配置文件



**16SpringMVC的注解**

1.@Controller

@Controller 用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller 对象。

2.@RequestMapping

@RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

3.@Resource和@Autowired

@Resource和@Autowired都是做bean的注入时使用，其实@Resource并不是Spring的注解，它的包是javax.annotation.Resource，需要导入，但是Spring支持该注解的注入。

相同点：两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要再写setter方法。

不同点：

@Autowired为Spring提供的注解，需要导入包

org.springframework.beans.factory.annotation.Autowired；

@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。如下：

public class TestServiceImpl {

@Autowired

@Qualifier("userDao")

private UserDao userDao;

}

2) @Resource

@Resource默认按照ByName自动注入，由J2EE提供，需要导入包javax.annotation.Resource。@Resource有两个重要的属性：name和type，而Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以，如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不制定name也不制定type属性，这时将通过反射机制使用byName自动注入策略。

注：最好是将@Resource放在setter方法上，因为这样更符合面向对象的思想，通过set、get去操作属性，而不是直接去操作属性。

4.@ModelAttribute和 @SessionAttributes

@ModelAttribute该Controller的所有方法在调用前，先执行此@ModelAttribute方法，可用于注解和方法参数中，可以把这个@ModelAttribute特性，应用在BaseController当中，所有的Controller继承BaseController，即可实现在调用Controller时，先执行@ModelAttribute方法。

 @SessionAttributes即将值放到session作用域中，写在class上面。

/\*\*

\* Spring MVC

在调用目标处理方法前，会先逐个调用在方法级上标注了

@ModelAttribute 的方法

\* 设置属性

\* @return

\*/

@ModelAttribute("myUser")

public User getUser(){

User user= new User();

user.setUserName("admin11");

user.setPassWord("tiger11");

return user;

}

在另外一个方法中，查看是否存在：

System.out.println("model中是否存在:"+model.containsAttribute("myUser"));

System.out.println("map中是否存在:"+map.get("myUser"));

在任意方法中：

model.addAttribute("testMyModelAttr","在session中看到我");

或者上面

@ModelAttribute("myUser")

public User getUser(){

User user= new User();

user.setUserName("admin11");

user.setPassWord("tiger11");

return user;

}

在类头上加入：

@SessionAttributes(value = {"myUser","testMyModelAttr"})

只要session不失效，在任何地方，都可以去到"myUser","testMyModelAttr"

session中是否包含:

<%-- ${requestScope} ${applicationScope}--%>

<br>${sessionScope.myUser} &nbsp; &nbsp; &nbsp; &nbsp;

${sessionScope.testMyModelAttr}<br>

5.@PathVariable /{a}/{b}/{c} /emp/list/1001

method (@PathVariable("a") String a ,@PathVariable String b,@PathVariable("c") String c1)

用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数。

6.@requestParam

@requestParam主要用于在SpringMVC后台控制层获取参数，类似一种是request.getParameter("name")，它有三个常用参数：defaultValue = "0", required = false, value = "isApp"；defaultValue 表示设置默认值，required 铜过boolean设置是否是必须要传入的参数，value 值表示接受的传入的参数类型。

7.@ResponseBody

作用： 该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

8.@Component

相当于通用的注解，当不知道一些类归到哪个层时使用，但是不建议。

9.@Repository

用于注解dao层，在daoImpl类上面注解。