1. springmvc框架

# MVC设计模式

## 什么是设计模式

设计模式（Design Pattern）是一套被反复使用、多数人知晓的、经过分类的、代码设计经验的总结。

使用设计模式的目的：为了代码可重用性、让代码更容易被他人理解、保证代码可靠性。

设计模式使代码编写真正工程化；

设计模式是软件工程的基石脉络，如同大厦的结构一样。

设计模式就是一种模子，经过多年实践锤炼形成一套行之有效的完成某个特定任务的步骤和方式。

例如：西凤酒的酿造过程，酿造工序，前后不能变，温差不能变，这样做就是好喝，稍微改动就变味道了。

再如，北京烤鸭，就是那样做，就是那些调料腌制，变量配比，味道口感就是不如人家。

## MVC设计模式

MVC设计模式是一种通用的软件编程思想

在MVC设计模式中认为, 任何软件都可以分为三部分组成：

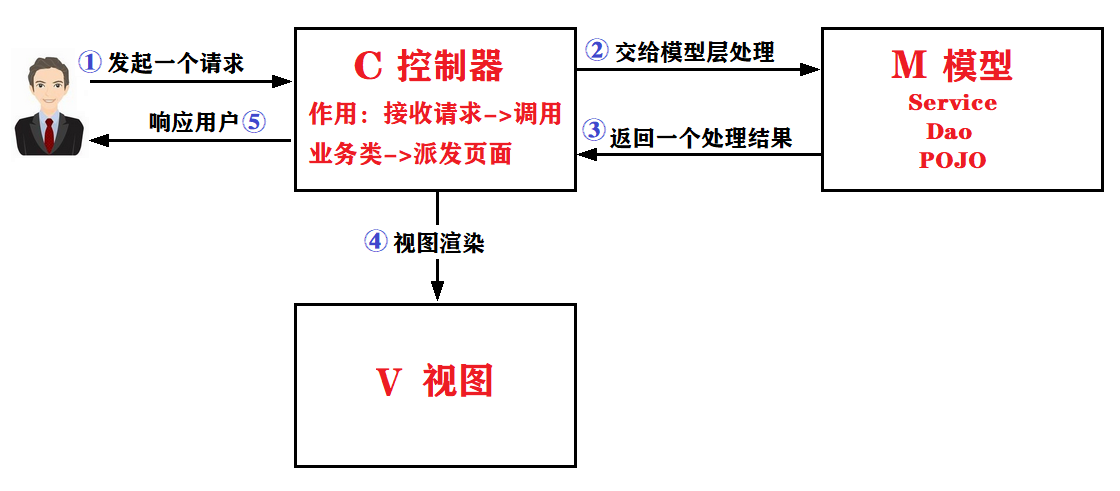
(1)控制程序流转的控制器（**Controller**）

(2)封装数据处理数据的模型（**Model**）

(3)负责展示数据的视图（**view**）

并且在MVC设计思想中要求一个符合MVC设计思想的软件应该保证上面这三部分相互独立，互不干扰，每一个部分只负责自己擅长的部分。

如果某一个模块发生变化，应该尽量做到不影响其他两个模块。这样做的好处是，软件的结构会变得更加的清晰，可读性强。有利于后期的扩展和维护，并且代码可以实现复用。



# 初识SpringMVC

## Servlet的缺点

1、通常情况下，一个Servlet类只负责处理一个请求，若项目中有成百上千个请求需要处理，就需要有成百上千个Servlet类，这样会使得项目中Servlet类的个数暴增；

2、在Servlet3.0版本之前，每一个Servlet都需要在web.xml文件中至少做八行配置信息，配置内容多且繁琐。当Servlet特别多时，web.xml配置量太多，不利于团队开发；

3、当通过客户端提交参数到服务器，通过Servlet进行接收时，无论数据本身是什么格式，在Servlet中一律按照字符串进行接收，后期需要进行类型转换，复杂类型还需要特殊处理，特别麻烦！

4、servlet具有容器依赖性，必须放在服务器中运行，不利于单元测试；

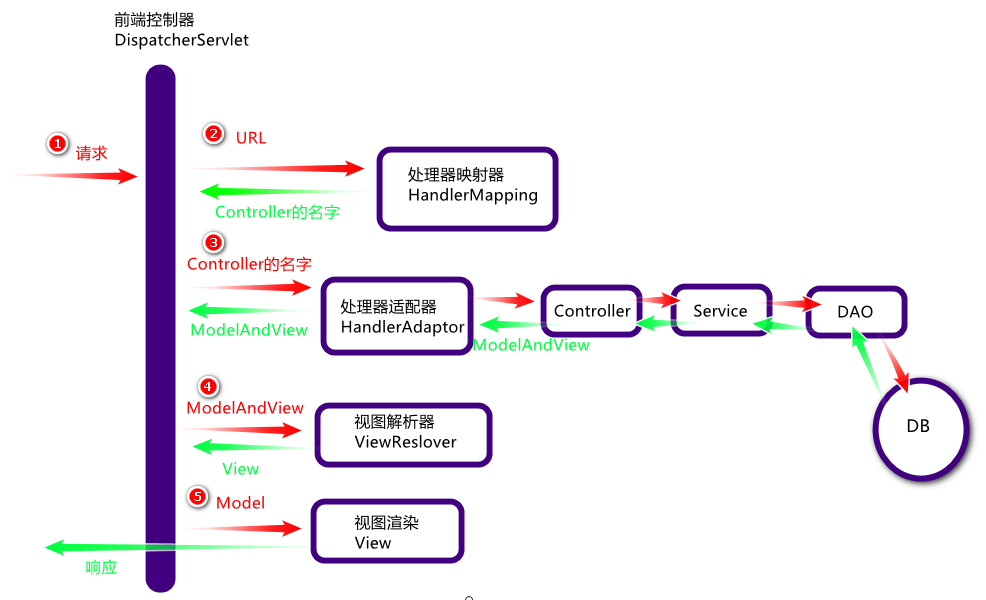
...

## SpringMVC简介

Springmvc是spring框架的一个模块，spring和springmvc无需中间整合层整合

Springmvc是一个基于mvc的web框架

## spring执行原理



(1).用户发送请求 至 前端控制器(DispatcherServlet)；

提示：DispatcherServlet的作用：接收请求，调用其它组件处理请求，响应结果，相当于转发器、中央处理器，是整个流程控制的中心

(2).前端控制器(DispatcherServlet)收到请求后调用处理器映射器(HandlerMapping)

处理器映射器(HandlerMapping)找到具体的Controller(可以根据xml配置、注解进行查找)，并将Controller返回给DispatcherServlet；

(3).前端控制器(DispatcherServlet)调用处理器适配器(HandlerAdapter)。处理器适配器经过适配调用具体的Controller；(Controller--> service --> Dao --> 数据库)

Controller执行完成后返回ModelAndView，

提示：Model(模型数据，即Controller处理的结果，Map) **View**(逻辑视图名，即负责展示结果的JSP页面的名字)

处理器适配器(HandlerAdapter)将controller执行的结果(ModelAndView)返回给前端控制器(DispatcherServlet)；

(4).前端控制器(DispatcherServlet)将执行的结果(ModelAndView)传给视图解析器(ViewReslover)

视图解析器(ViewReslover)根据View(逻辑视图名)解析后返回具体JSP页面

(5).前端控制器(DispatcherServlet)根据Model对View进行渲染（即将模型数据填充至视图中）；

前端控制器(DispatcherServlet)将填充了数据的网页响应给用户。

其中整个过程中需要开发人员编写的部分有 **Controller**、**Service**、**Dao**、**View；**

# springmvc入门案例

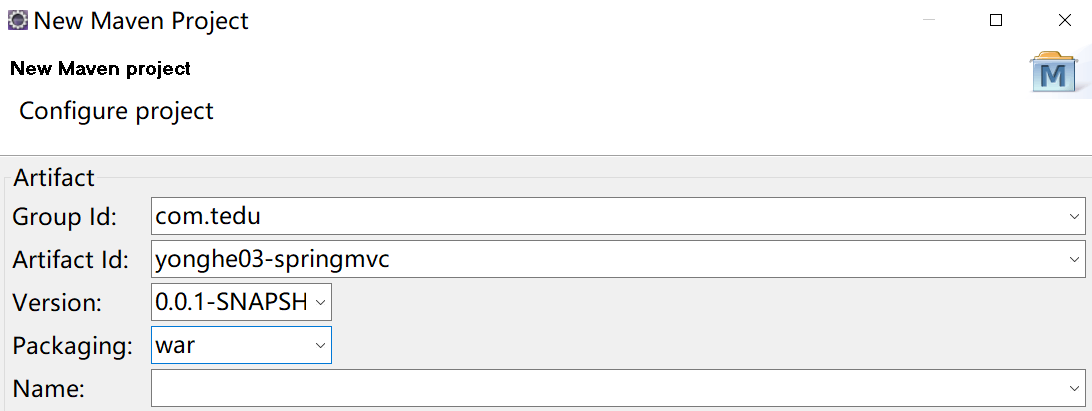
**需求：**

(1)通过浏览器访问 <http://localhost/项目名称/hello> 地址，在控制台输出 "hello springmvc"

(2)将请求转向 /WEB-INF/pages/home.jsp 页面

## 创建Maven—Javaweb工程

**1、通过Maven创建javaweb工程**



2、在pom.xml中引入springmvc所需jar包：将下面的配置直接拷贝到pom.xml中的根标签内

<!-- 集中定义依赖版本号 -->

<properties>

<junit.version>4.10</junit.version>

<spring.version>4.1.3.RELEASE</spring.version>

</properties>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

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

<scope>test</scope>

</dependency>

<!-- SpringMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

<!-- Servlet支持Request和Response -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.4</version>

<scope>provided</scope>

</dependency>

<!-- java对象转换json的工具类 -->

<dependency>

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

<artifactId>jackson-databind</artifactId>

<version>2.5.1</version>

</dependency>

</dependencies>

## 在web.xml中配置springmvc

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"WebApp\_ID"* version=*"2.5"*>

<!-- 配置springmvc前端控制器, 将所有请求交给springmvc来处理 -->

<servlet>

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

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

<!-- 配置springmvc核心配置文件的位置，默认Springmvc的配置文件是在WEB-INF目录下，默认的名字为springmvc-servlet.xml，如果要放在其他目录，则需要指定如下配置：

-->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc-config.xml</param-value>

</init-param>

</servlet>

<!-- 其中的斜杠（/）表示拦截所有请求（除JSP以外）, 所有请求都要经过springmvc前端控制器 -->

<servlet-mapping>

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

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

</servlet-mapping>

</web-app>

## 创建并配置springmvc-config.xml

直接复制下面配置文件的内容即可！

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>

<!-- 1.配置前端控制器放行静态资源(html/css/js等，否则静态资源将无法访问) -->

<mvc:default-servlet-handler/>

<!-- 2.配置注解驱动，用于识别注解（比如@Controller） -->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- 3.配置需要扫描的包：spring自动去扫描 base-package 下的类，

如果扫描到的类上有 @Controller、@Service、@Component等注解，

将会自动将类注册为bean

-->

<context:component-scan base-package=*"com.tedu.controller"*>

</context:component-scan>

<!-- 4.配置内部资源视图解析器

prefix:配置路径前缀

suffix:配置文件后缀

-->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

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

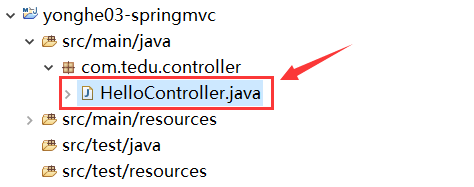
<property name=*"suffix"* value=*"****.jsp****"*/>

</bean>

</beans>

## 创建并实现HelloController类

1、创建com.tedu.controller.HelloController类



2、实现HelloController类

**package** com.tedu.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller /\* 这个注解表示当前类是属于控制层 \*/

**public** **class** HelloController {

/\* [http://localhost/项目名称**/hello**](http://localhost/项目名称/hello)\*/

@RequestMapping("**/hello**")

/\* 这个注解用于：映射请求的资源路径(/hello)和当前方法(hello)的对应关系

\* 当浏览器请求 /hello 路径时, 将会访问(执行)当前这个方法 \*/

**public** String hello() {

System.***out***.println("hello springmvc...");

**return** "**home**";

}

}

## 创建并实现home.jsp

在WEB-INF/pages/目录下，创建home.jsp页面。

**WEB-INF/pages/home.jsp**

<%@ page pageEncoding=*"utf-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"utf-8"*/>

</head>

<body>

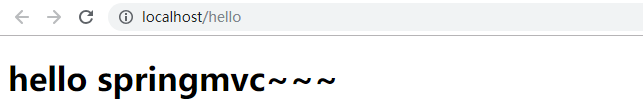
<h1>hello springmvc~~~</h1>

</body>

</html>

## 访问测试

打开浏览器，输入url地址：<http://localhost/hello> 地址，访问结果如下：



# springmvc参数绑定

当项目中引入springmvc框架后，所有的请求流转将由springmvc进行控制，当客户端发送的请求中包含数据（也就是请求参数）时，那么该如何在controller层获取这些参数呢？

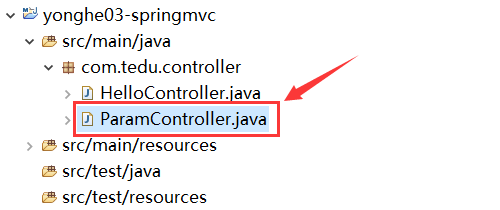
springmvc会自动的将请求中包含的参数和方法的参数进行匹配，也就是说只要保证，请求中的参数名称和方法中的参数名称相对应（另，参数的格式也要正确），在方法中就可以使用这些参数—即请求中的参数。

## 基本类型参数绑定

当需要获取客户端发送过来的少量数据时，可以在Controller中声明的方法上，通过声明方法参数对这些参数一个一个进行接收，具体示例如下：

**需求：通过浏览器发请求访问Controller，并在请求中携带name、age数据访问服务器，在服务器端的 Controller中获取这些数据。**

1、创建com.tedu.controller.ParamController类



2、在ParamController类中添加param1方法，用于接收基本类型的参数，代码实现如下：

**package** com.tedu.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller /\* 这个注解表示当前类是属于控制层 \*/

**public** **class** ParamController {

/\* 测试：接收基本类型的类型的参数 \*/

@RequestMapping("param1")

**public** String param1(String name, Integer age){

System.***out***.println("name="+name);

System.***out***.println("age="+age);

**return** "home";

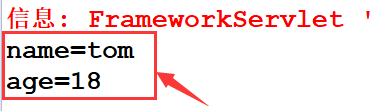
}

}

3、访问ParamController中的param1方法，在访问时，注意将name和age参数一起发送给服务器:



控制台输出结果为：



## 包装类型参数绑定

当需要获取客户端发送过来的多个数据时，可以在Controller中声明的方法上，通过声明方法参数对这些数据一个一个进行接收较麻烦，可以在方法上声明对象类型的参数，通过对这些数据统一进行接受，springmvc会自动将接收过来的参数封装在对象中，具体示例如下：

1、在ParamController类中添加param2方法，用于接收对象类型的参数，代码实现如下：

**package** com.tedu.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller /\* 这个注解表示当前类是属于控制层 \*/

**public** **class** ParamController {

/\* 测试：接收对象类型的参数 \*/

@RequestMapping("param2")

**public** String param2(User user){

System.***out***.println("user="+user);

**return** "home";

}

@RequestMapping("param1")

**public** String param1(String name, Integer age){

System.***out***.println("name="+name);

System.***out***.println("age="+age);

**return** "home";

}

}

3、创建User类，声明name和age属性，提供对应的set和get方法，并重写toString方法

**package** com.tedu.controller;

**public** **class** User {

**/\* 声明name、age属性 \*/**

**private** String name;

**private** Integer age;

**/\* 声明name、age属性对应的set和get方法 \*/**

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

**/\* 重写toString方法 \*/**

@Override

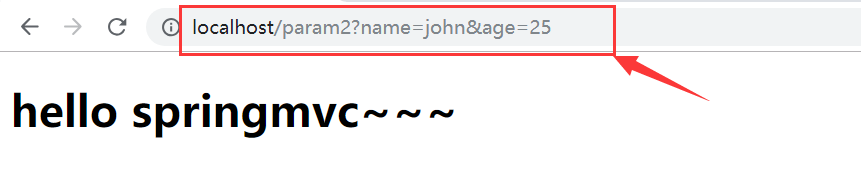
**public** String toString() {

**return** "User [name=" + name + ", age=" + age + "]";

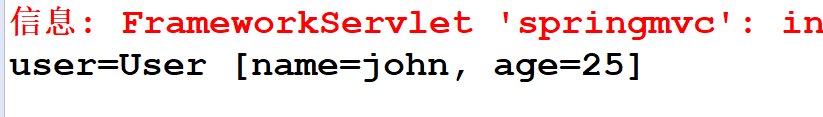
}

}

4、访问ParamController中的param2方法，在访问时，注意将name和age参数一起发送给服务器:



控制台输出结果为：



## 日期类型参数绑定

1、在ParamController类中添加param3方法，代码实现如下：

**package** com.tedu.controller;

**import** java.util.Date;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

@Controller /\* 这个注解表示当前类是属于控制层 \*/

**public** **class** ParamController {

/\* 测试：接收日期类型的参数 \*/

@RequestMapping("param3")

**public** String param3(Date date){

System.***out***.println("date="+date.~~toLocaleString~~());

**return** "home";

}

@RequestMapping("param2")

**public** String param2(User user){

System.***out***.println("user="+user);

**return** "home";

}

@RequestMapping("param1")

**public** String param1(String name, Integer age){

System.***out***.println("name="+name);

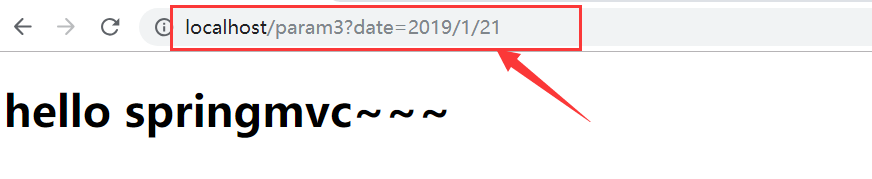
System.***out***.println("age="+age);

**return** "home";

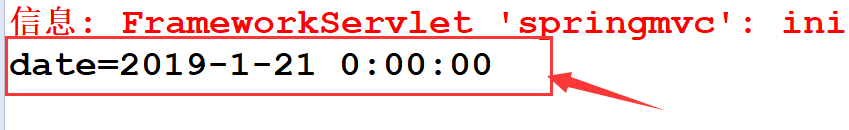
}

}

2、访问ParamController中的param3方法，在访问时，注意将date参数一起发送给服务器:



控制台输出结果为：

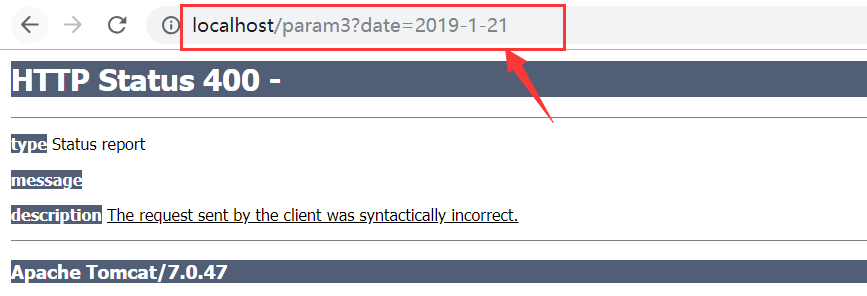


**==================================================================**

**常见问题：**

**==================================================================**

1、当访问ParamController中的param3方法，如果传递给服务器的日期数据是如下格式：



从图中可以看出，如果日期参数是 yyyy-MM-dd 格式就会出现400错误，其实是因为参数格式匹配错误，由于springmvc默认的日期格式是yyyy/MM/dd，因此如果日期参数不是 yyyy-MM-dd 格式，就会出现400错误！！

**2、解决方案：**

在springmvc中，提供了@InitBinder注解，用于指定自定义的日期转换格式，因此，我们只需要在Controller类中添加下面的代码即可，在接受日期类型的参数时，会自动按照自定义的日期格式进行转换。

**public** **class** ParamController {

@RequestMapping("param3")

**public** String param3(Date date){

System.***out***.println("date="+date.~~toLocaleString~~());

**return** "home";

}

/\* 自定义日期转换格式 \*/

@InitBinder

**public** **void** InitBinder (ServletRequestDataBinder binder){

binder.registerCustomEditor(java.util.Date.**class**,

**new** CustomDateEditor(**new** SimpleDateFormat("yyyy-MM-dd"), **true**)

);

}

...

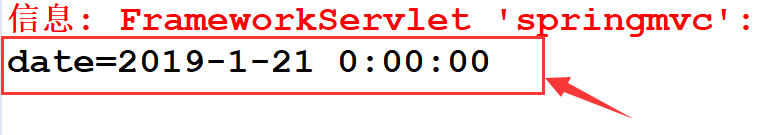
...

}

3、再次测试：



控制台输出结果为：



# 跳转和乱码处理

## 实现转发(forward)

在前面request对象的学习中，通过request对象可以实现请求转发（即资源的跳转）。

同样的，springmvc也提供了请求转发的方式，具体实现如下：

需求：通过浏览器访问 testForward方法，执行testForward方法后，将请求转发到 （HelloController）hello， 也就是home.jsp页面。

1、在HelloController中，提供testForward方法，代码实现如下：

/\* 测试请求转发(forward) \*/

@RequestMapping("testForward")

**public** String testForward(){

System.***out***.println("测试请求转发(forward)...");

**return** "**forward**:hello";

}

2、打开浏览器，在浏览器中输入：<http://localhost/testForward> 地址，访问效果如下:



forward方式相当于:

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

转发是一次请求，一次响应；

转发后地址栏地址没有发生变化（还是访问testForward的地址）；

转发前后的request和response对象也是同一个。

## 实现重定向(redirect)

在前面response对象的学习中，通过response对象可以实现请求重定向（即资源的跳转）。

同样的，springmvc也提供了请求重定向的方式，具体实现如下：

需求：通过浏览器访问 testRedirect方法，执行testRedirect方法后，将请求重定向到 （HelloController）hello， 也就是home.jsp页面。

1、在HelloController中，提供testRedirect方法，代码实现如下：

/\* 测试请求重定向(redirect) \*/

@RequestMapping("testRedirect")

**public** String testRedirect(){

System.***out***.println("测试请求重定向(redirect)...");

**return** "redirect:hello";

}

2、打开浏览器，在浏览器中输入：<http://localhost/testForward> 地址，访问效果如下:



redirect方式相当于：

response.sendRedirect(url);

重定向是两次请求，两次响应；

重定向后地址栏地址发生了变化(变为转发后的地址)；

并且在重定向前后，request和response对象不是同一个。

## 乱码处理

在前面的Servlet学习中，我们学习了GET和POST请求参数乱码如何解决。

springmvc也提供了解决请求参数乱码的方案，就是在web.xml中加入如下代码，可以解决POST提交参数乱码：

<filter>

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

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

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

<param-value>UTF8</param-value>

</init-param>

</filter>

<filter-mapping>

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

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

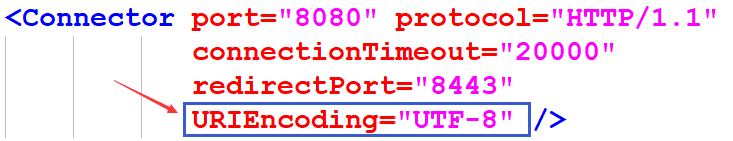
</filter-mapping>

Servlet中，两种请求方式乱码解决方案回顾：

(1)如果请求方式为**POST**提交，必然会出现乱码，解决方式是在任何获取参数的代码之前，添加如下代码：

request.setCharacterEncoding("utf-8");

(2)如果请求方式为**GET**提交，tomcat8及之后的版本已经解决了中文参数乱码的问题，不需要处理，tomcat7及之前的版本只需要在 [tomcat]/conf/server.xml中添加如下配置也可以解决乱码问题。



# springmvc响应数据

## Model的使用

当请求发起访问Controller中的方法时，可以通过参数声明，在方法内使用Model。

@RequestMapping("/doorList")

**public** String doorList(**Model** **model**){}

Model对象实际上是一个Map集合，例如：往model中添加一个属性

model.addAttribute(String name, Object value);

其中，addAttribute方法会将属性保存到request域中，再通过转发将属性数据带到相应的JSP中，通过${}取出并显示。

示例，往Model中添加属性：

@RequestMapping("/testModel")

**public** String testModel(Model model){

/\* 往Model添加属性 \*/

model.addAttribute("name", "刘德华");

model.addAttribute("age", 20);

**return** "home";

}

在home.jsp中取出属性并显示：

<body>

<h1>hello springmvc~~~</h1>

${ name } <br/>

${ age }

</body>

## 返回JSON数据

1、什么是JSON？

JSON（JavaScript Object Notation）是一种JS提供的轻量级的数据交换格式。

JSON在项目开发中是一种非常流行的数据交换格式。

例如:在JS中可以通过下面的形式，声明一个person对象

var person = {

"name" : "张飞",

"age" : 18,

"friends" :["关羽", "刘备"],

"sayHi" : function(){ alert("person.sayHi()..") }

};

可以通过person对象访问其中的属性或方法:

person.name; // 张飞

person.age; // 18

person.friends; // ["关羽", "刘备"]

person.sayHi(); // 弹框提示 person.sayHi()..

上面是JS中声明对象的一种常用方式，也是JSON的格式。

2、由于JSON格式简单, 并且可以通过JS非常方便的访问JSON中的数据。因此，在服务器响应时，经常会返回一个JSON数据：

@RequestMapping("/testJson")

**@ResponseBody**

**public** List<User> testJson(){

//模拟查询所有用户,将所有用户信息封装到List<User>集合中

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

list.add( **new** User("张三", 18) );

list.add( **new** User("李四", 20) );

list.add( **new** User("王五", 22) );

//将所有用户的List<User>集合以JSON格式响应

**return** list;

}

返回的结果为：

[{

"name": "张三",

"age": 18

}, {

"name": "李四",

"age": 20

}, {

"name": "王五",

"age": 22

}]

# 扩展内容

## springmvc前端控制器拦截静态资源的解决办法

在配置SpringMVC开发环境时，会在web.xml文件中配置SpringMVC的前端控制器，将所有请求交给前端控制器处理，因此在url-pattern中配置了斜杠（/），

<!-- 1.配置springmvc前端控制器, 并将所有请求交给springmvc处理 -->

<servlet>

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

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

<!-- 配置springmvc核心配置文件的位置，默认Springmvc的配置文件是在WEB-INF目录下，默认的名字为springmvc-servlet.xml，如果要放在其他目录，则需要指定如下配置：

-->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc-config.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

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

<!-- 斜杠表示拦截所有请求（除JSP以外） -->

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

</servlet-mapping>

url-pattern中配置的斜杠（/）表示将除了JSP以外的其他请求都拦截下来，交给spring的前端控制器来处理。

但是这样配置，会将对静态资源的访问也拦截下来，导致访问静态资源时，出现404（资源找不到），因为spring的前端控制器将对静态资源的访问也当成了一个controller请求，去配置对应的映射路径，这当然找不到。

比如访问：<http://localhost/day15-springmvc/home.html>，由于配置的是斜杠（/），所以此时会拦截静态资源，到controller中匹配路径为/home.html的方法，此时自然是匹配不到的。

如果需要访问到静态资源，让前端控制器对静态资源的请求放行。此时可以在springmvc-config.xml文件中添加放行静态资源的配置:

<!-- 1.配置前端控制器放行静态资源(html/css/js等，否则静态资源将无法访问) -->

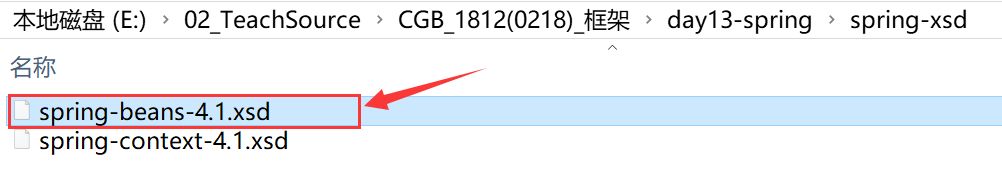
<mvc:default-servlet-handler/>

## spring/springmvc配置文件没有提示的解决办法

### applicationContext.xml没有提示的解决办法

1、配置spring-beans-4.1.xsd文件

(1)找到spring-beans-4.1.xsd的文件的位置，例如：

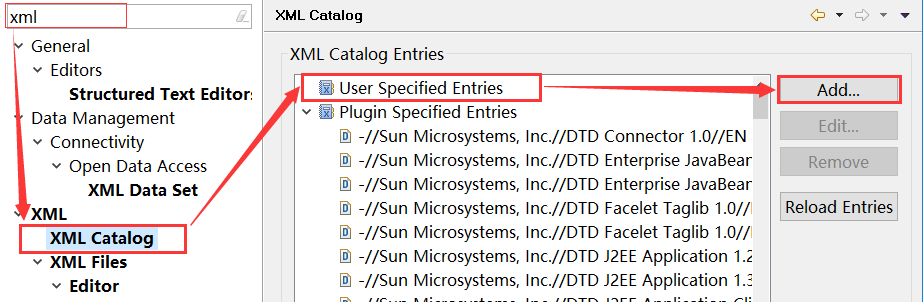


(2)复制下面的url地址：

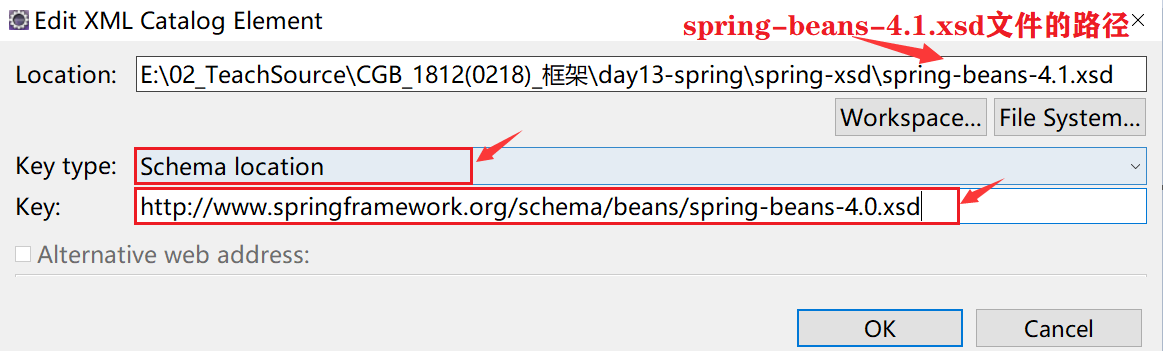
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

(3)在eclipse菜单栏中: window --> Preferences --> 在搜索框中搜索 [ xml ]

XML --> XML Catalog --> User Specified Entries --> Add…

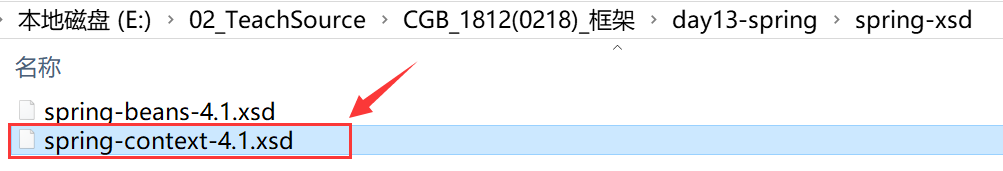


(4)在弹出的窗口中:



2、配置spring-context-4.0.xsd文件

(1)找到spring-context-4.0.xsd的文件的位置，例如：

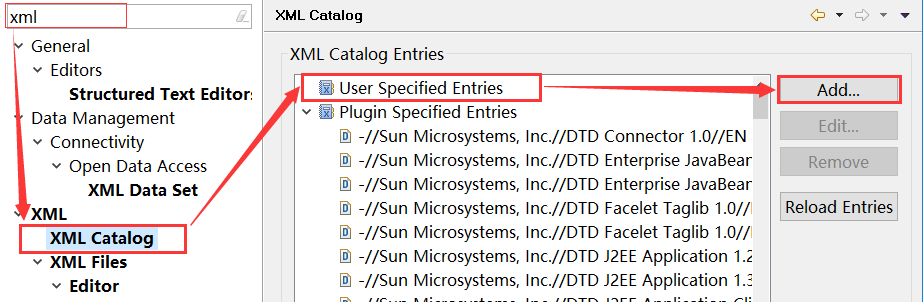


(2)复制下面的url地址：

http://www.springframework.org/schema/context/spring-context-4.0.xsd

(3)在eclipse菜单栏中: window --> Preferences --> 在搜索框中搜索 [ xml ]

XML --> XML Catalog --> User Specified Entries --> Add…



(4)在弹出的窗口中:

