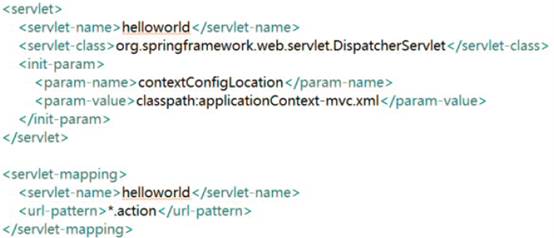
**1.配置Web.xml**

(1)在Web.xml中通过servlet-mapping的方式将指定类型的请求交给DispatcherServlet处理。并配置好Spring配置文件的目录和文件名。



如果不通过<init-param>中contextConfigLocation来指定SpringMVC配置文件的存储位置，则：

DispatcherServlet 默认加载 /WEB-INF/<servlet-name>.xml 的 SpringMVC 配置文件, 启动 WEB 层的 SpringMVC 容器。上例中就是/WEB-INF/helloWorld.xml

(2)<servlet>中还可以指定<load-on-startup>1</load-on-startup>



**2.创建MVC配置文件**

（1）配置自动扫描的包

ip_image006.jpeg

之后请求就会到该包下的类的方法中去匹配RequestMapping

 (2)配置视图解析器



此配置代表RequestMapping注解的方法中的return值对应的跳转页面为：

/WEB-INF/pages/return值.jsp

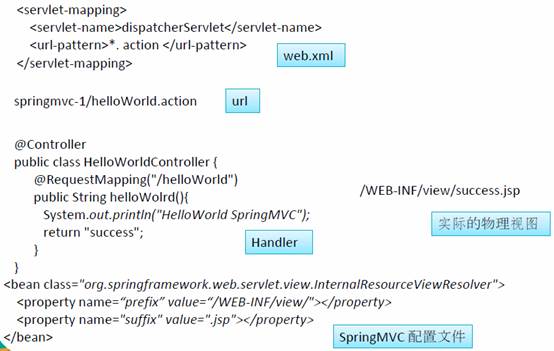
**3.请求处理器类**

请求处理器类在类上面用@Controller注解修饰。

类中方法上用 @RequestMapping("/helloWorld")来修饰，代表指定请求会交给这个方法来处理。

@RequestMapping修饰的类的return值代表着要跳转的视图。

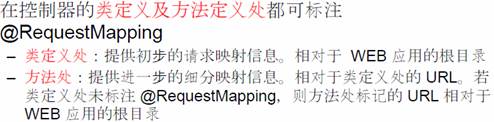
HelloWorld例子：



**4.@RequestMapping**

(1)

ip_image012.jpeg



所谓相对于WEB应用根目录是指，如果只有方法处存在@RequestMapping("helloWorld")，则对应url请求为：SpringMVCProject/helloWorld

Eclipse中WEB应用根目录就是WebRoot。

ip_image016.jpeg

（2）RequestMapping中可以配置的参数有：value，method，params，headers来精确映射请求，他们之间是“与”关系。

value：代表指定的url请求。

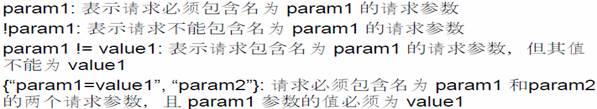
@RequestMapping(value="helloWorld") 和@RequestMapping("helloWorld") 作用相同。

method：代表请求的方式。如果不匹配则会报请求not support的错误。

@RequestMapping(value="helloWorld", method = **RequestMethod.POST**)

@RequestMapping(value="helloWorld", method = **RequestMethod.GET**)

params: 代表请求时的参数值

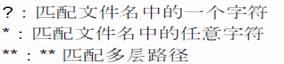


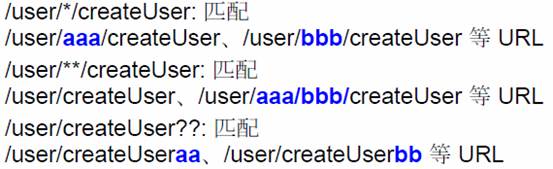
headers：代表Http请求，报文头header中特定属性的值

ip_image020.jpeg

这些参数是让RequestMapping更加精确，如果某个请求不能匹配到任何一个RequestMapping，则会报404错误。

（3）Ant风格匹配符可用在@RequestMapping的映射请求的url参数中





（4）占位符



**@PathVariable(value="占位符名")**

**注意**：此时这个参数不是普通get，post方法中传递的参数，而是将url地址的一部分作为参数。

所以对于上例可以访问到的url可以是:

<a href="MVCProject/delete/123">linkaa</a>

（5）HTTP 协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET 用来获取资源，POST 用来新建资源，PUT 用来更新资源，DELETE 用来删除资源。

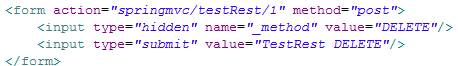
浏览器 form 表单只支持 GET 与 POST 请求，而DELETE、PUT 等 method 并不支持。

解决办法：

1.先配置一个HiddenHttpMethodFilter



2.写一个以POST方式发送请求的form表单，在表单中携带一个name="\_method"的隐藏域，值为DELETE或PUT。



3.为了能够更准确的使不同类型的请求对应不同的Controller，可以在@RequestMapping中指定method类型，例如

ip_image032.jpeg

**5.映射请求参数**

SpingMVC中可以通过使用注解将HTTP请求信息绑定到相对应的处理方法的方法参数中。注解具体有：

(1)**@RequestParam**:绑定请求参数，其中的value属性值与form表单中的name对应或者是url中参数列表的键对应。

@RequestParam(value="name",required=true,default="qiutian")

value是请求参数的名字。required代表是否必须要有这个请求参数，如果required为true但实际上没有这个请求参数，则将抛出异常。defaultValue代表请求参数不存时候的默认值。



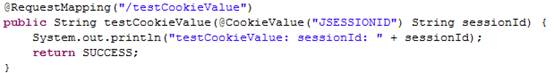
（2）**@RequestHeader**：绑定请求报头。即Http协议中Header中的属性值。

请求头包含了若干个属性，服务器可据此获知客户端的信息，通过@RequestHeader 即可将请求头中的属性值绑定到处理方法的形参中。

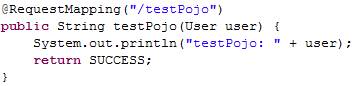
@RequestHeader(value="...", required=..., defaultValue="..." )

value对应header部分键值对中的键

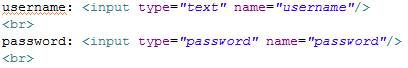
（3）**@CookieValue**：绑定请求中Cookie值



（4）POJO对象，此时不需要用注释，直接将POJO对象作为方法参数即可。



此时form表达中对应input的name值就是User类的成员属性的名字，如

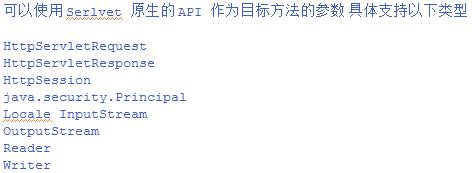


其中username，password是User类的成员变量，并配有get、set方法。

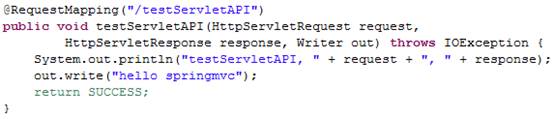
此处POJO属性支持联级，即User中成员变量可以是另一个类的对象如Address类对象address.此时，表单中对应name为 address.city，address.province

form表单中action属性就是请求地址，与RequestMapping中value值对应。

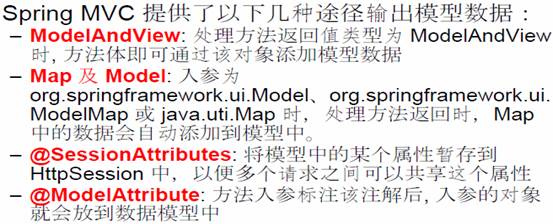
（5）ServletAPI



例：



**6.处理模型数据**



这些方法都会将模型数据放入在request域中，用于在页面上进行显示。

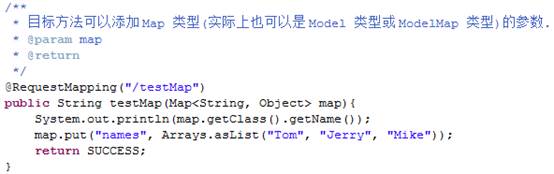
(1)ModelAndView





(2)Map及Model





当传入一个Map作为方法参数以后，就相当于得到了模型对象的引用，然后就可以用.put(key,value)往模型里面存数据，或者用.get(key)来从模型中取数据。

存的数据会存放在request域中。

ip_image056.jpeg

如果传入的是Model类型对象model，则使用model.addAttribute(key, value)可以将数据放在request域中。

（3）@SessionAttributes (注意拼写：有s)



@SessionAttributes注解包含两个属性，value和types，都是数组类型的

value：通过属性名将指定的属性存放在HttpSession中

types: 通过对象类型将指定类型的对象放入在HttpSession中，这里types的值为类型的class对象。如User.class, String.class

ip_image060.jpeg

注意：

- @SessionAttributes注解只能放在类上面，不能修饰方法

-被@SessionAttributes指定的数据，还是通过map.put,或者model.addAttribute来放在模型中，只不过这些数据不仅存在于request域中，还存在于session域中。

（4）@ModelAttribute

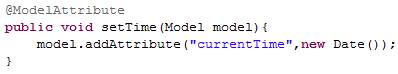
Model:SpringMVC中Model属性是一个键值对，Model中key用于与方法中参数匹配或者是用来在页面上显示。Model中value代表对应key的Model的值。

@ModelAttribute可以用来修饰方法或者用来修饰方法中的形式参数。目的是将数据放入到Model中同时也就默认存在request域中（ModelMap.addAttribute方法也是将数据放在request域中）。特殊情况，当此放入在Model中的key存在于@SessionAttributes的value中或者数据类型在types中时，这个数据也同时被放在了session域中。

**@ModelAttribute注释方法:**

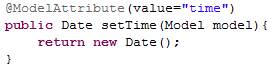
被@ModelAttribute注释的方法，会在每个目标方法执行前被SpringMVC调用，目的是将一些数据放入在**request**域中，如果还注释了@SessionAttributes则还可能在session域中。

(1)注释void返回值的方法



一般会在方法中用model或map将数据加到模型数据中。此时一个key为currentTime，value为new Date()的模型数据被放在了request域中，可以在页面用${currentTime}或${requestScope.currentTime}显示

(2)注释有返回值的方法



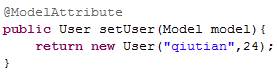
此时被放在request域中的数据模型的key值取决于：如果@ModelAttribute中有value属性则此value属性的值为key值，如果没有value属性，则将返回值类型名第一个字母小写作为key值。value值为返回值。

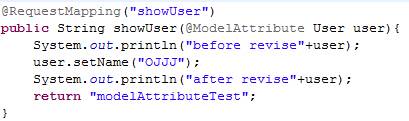
一般对应将基本类型作为返回值的都建议在@ModelAttribute中写上value属性。

**@ModelAttribute注释目标方法参数:**

（1）用@ModelAttribute注释方法参数时，首先参数值会来源于特定的被@ModelAttribute注释的方法里面存入的模型数据。之后如果与url中参数重复则会被覆盖。最后目标方法执行结束后，被@ModelAttribute注释的参数会被继续添加到模型数据中，从而存在request域中。（如果标注了SessionAttributes则可能被存在session域中）。

例：





当执行

ip_image070.jpeg

 ，

首先showUser目标方法中use对象会被setUser方法中填入model中的数据赋值，即name=qiutian，age=24。之后user对象又被url请求参数中的name，age重新赋值。最后在执行完showUser目标方法后，被修改过的user对象又被再次放在了request域中，此时user对象name为OJJJ。

（2）用@ModelAttribute注释的参数可以直接根据form表单中的name来赋值，但不写@ModelAttribute其实也可以。对于基本类型变量，变量名和表单中name匹配即可，对于pojo只要类成员属性与表单中name对应即可。

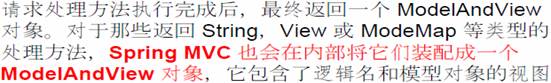


容易出现的异常：@ModelAttribute的value值存在域SessionAttributes的value中，但这个session中的model没有值。

一般@ModelAttribute用于注释POJO，当注释基本类型对象或者String时，当再次向request域存值的时候容易在前台读不到。

**7.视图和视图解析器**

(1)



ip_image076.jpeg

SpringMVC不关心最后是由何种视图来完成数据渲染工作，SprinMVC提供了一个高度抽象的接口View，所有视图包括自定义视图都是继承这个接口。SpringMVC主要工作在于生产数据模型，不关心视图渲染工作。

常用的视图实现类有：

- InternalResourceView：将JSP或其他资源封装成视图

- JstlView：如果JSP文件中使用了JSTL国际化标签，则会使用该视图

- AbstractExcelView: Excel文档视图

（2）

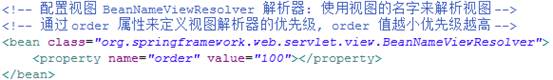
ip_image078.jpeg

ip_image080.jpeg



常用的视图解析器实现类：

ip_image084.jpeg



ip_image088.jpeg

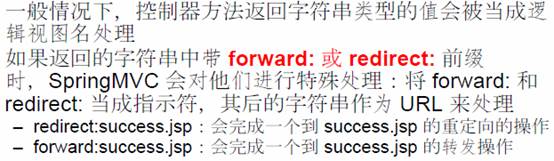
（3）如果希望直接从输入界面直接调转到视图界面，而不经过controller，则可以使用 mvc:view-controller标签



path代表定义跳转的url，view-name就类似于return的返回值，会被视图解析器解析成对应的视图。

必须同时配置<mvc:annotation-driven>不然会在普通request请求跳转到controller时报错。

(4)



**注意：**

（1）使用redirect和forward以后，视图解析器将不再会工作。所以需要明确的指定需要跳转的页面的位置和全面。

如果转发到jsp页面：则为 return "forward:/jsp/showResult.jsp"

如果只是重定向到另一个controller，则为 return "redirect:/searchInfo"

（2）forward转发：request会保留，一般用于从controller向页面传值

  redirect重定向：request不会被保留，一般用于从一个controller跳到另一个controller

关于路径，建议一般使用绝对路径，从/开始。

**8.视图和视图解析器**