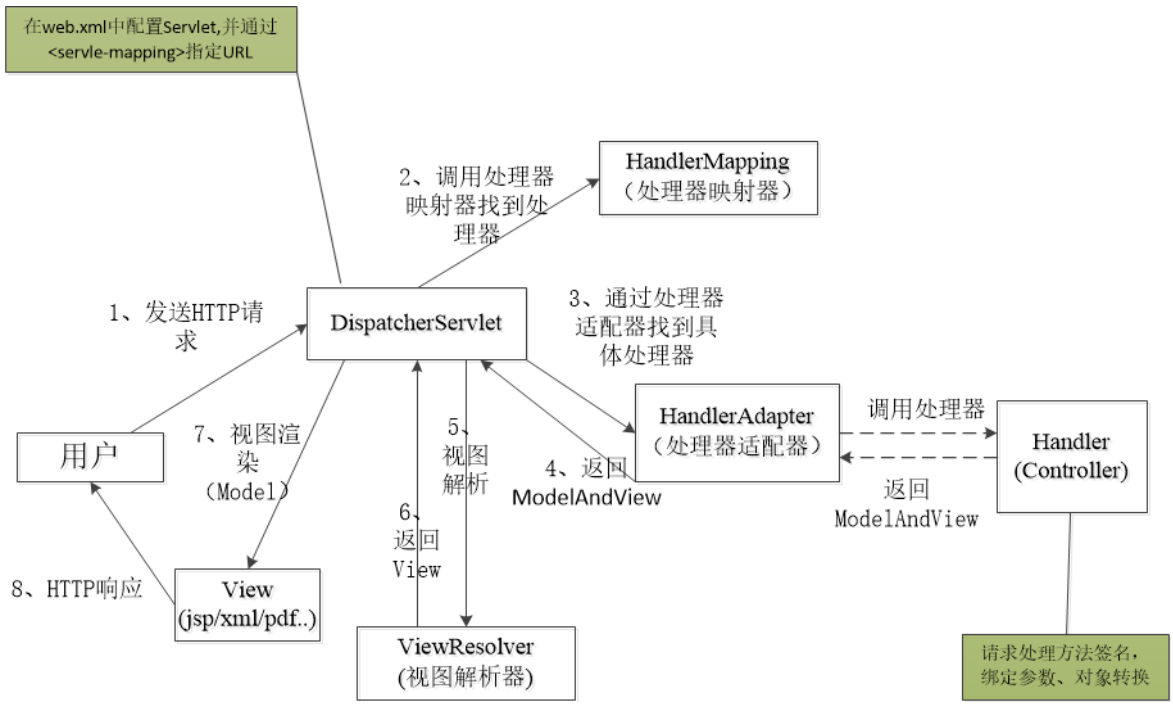
Spring MVC

1 工作原理（体系架构）



步骤描述：

第一步：用户发起HTTP请求，web服务器收到请求、若匹配到DispatcherServlet的请求映射路径（在web.xml中指定），则web容器将请求转交给DispatcherServlet处理。
第二步：DispatcherServlet根据请求信息，及HandlerMapping配置（可以根据xml配置、注解进行查找）查找Handler
第三步：DispatcherServlet根据HandlerMapping得到请求对应的Handler后，通过HandlerAdapter去调用执行Handler
第四步：Handler执行完成给适配器返回ModelAndView，HandlerAdapter向DispatcherServlet返回ModelAndView，ModelAndView是spring mvc框架的一个底层对象，包括 Model（模型）和view（视图逻辑名）
第五步：ModelAndView中包含的是“逻辑视图名”而非真正的视图对象，DispatcherServlet通过视图解析器完成视图名到视图对象的解析工作
第六步：视图解析器向DispatcherServlet返回视图对象View
第七步：DispatcherServlet使用View对象对ModelAndView中的模型数据进行视图渲染，视图渲染将模型数据(在ModelAndView对象中)填充到request域
第十一步：DispatcherServlet向用户响应结果

2 配置DispatcherServlet

```
<servlet>
    <servlet-name>smart</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>

    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:smart-servlet.xml
        </param-value>
        <!-- 默认是/WEB-INF/[servlet名字]-servlet.xml -->
    </init-param>
    <load-on-startup>1</load-on-startup>
    <!--启动优先级-->
</servlet>

<servlet-mapping>
    <servlet-name>smart</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

声明DispatcherServlet，默认自动加载/WEB-INF/smart-servlet.xml的spring配置文件。

配置参数

namespace: 命名空间，默认为-servlet,指定属性后WEB-INF/.xml

contextConfigLocation: 指定spring配置文件的得资源路径。

指定DispatcherServlet处理所有以.html为后缀的Http请求。

2.1 一个简单的实例（大概认识）

通过简单实例讲解SpringMVC开发过程

（1）配置web.xml，指定业务层对应的Spring配置文件，定义DispatcherServlet

（2）编写处理请求的控制器

（3）编写视图对象，这里使用jsp视图对象

（4）配置Spring MVC的配置文件，使控制器、视图解析器等生效

(a)负责用户处理的控制器：UserController

```
@Controller
@RequestMapping("user")
public class UserController {
    @Autowired
    private UserService userService;

    @RequestMapping("/register.html")
    public String register(){
        return "register";
    }

    @RequestMapping(method = RequestMethod.POST)
    public ModelAndView createUser(User user){
        userService.createUser(user);
```

```
        ModelAndView mav=new ModelAndView();
        mav.setViewName("creatSuccess");
        mav.addObject("user",user);
        return mav;
    }

}
```

(b)视图对象：

```
<form method="post" action="<c:url value="/user.html"/>">
    <table>
    <tr>
    <td>用户名：</td>
    <td><input type="text" name="userName"></td>
    </tr>
    <tr>
    <td>密码：</td>
    <td><input type="password" name="password"></td>
    </tr>
    <tr>
       <td>姓名：</td>
       <td><input type="text" name="realName"></td>
    </tr>
     <tr>
    <td colspan="2"><input type="submit" name="提交"/> </td>
```

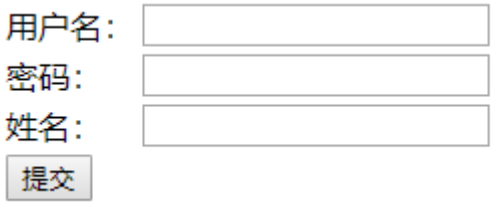
(c)创建成功页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户创建成功</title>
</head>
<body>
恭喜，用户${user.userName}创建成功
</body>
</html>
```

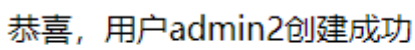
(d)springmvc配置文件

```
<context:component-scan base-package="com.smart.web"/>
<!--配置视图解析器，将ModelAndView及字符串解析为具体页面-->
<bean
        class="org.springframework.web.servlet.view.InternalResourceV
        p:viewClass="org.springframework.web.servlet.view.JstlView"
        p:prefix="/WEB-INF/jsp/"
        p:suffix=".jsp"/>
```

(e)登录页面：



(f)注册成功页面:



2.2 注解驱动的控制器

类处的注解@RequestMapping("user")指定的URL相对于WEB应用的部署路径，而方法处指定的URL相对于类定义处指定的URL.

其他映射请求，也可以根据请求方法及参数进行映射。

URL中的{xxx}占位符可以通过@PathVariable("xxx")绑定到操作方法的入参中。

```
@RequestMapping(path="/delete",method = RequestMethod.POST,params = "userId")
@RequestMapping(headers = "content-type=text/*")
```

3 请求处理方法签名

```
//1、请求参数按名称匹配的方法绑定到方法入参中，方法返回的字符串代表逻辑视图名

 @RequestMapping（Path="/handle1"）
 public String handle1(@RequestParam("username") String username,
                       @RequestParam("password") String password,
                       @RequestParam("realName") String realName){
     return "main";
 }
 //2、请求参数按名称匹配的方法绑定到user属性中，方法返回的字符串代表逻辑视图名
 @RequestMapping（Path="/handle2"）
 public ModelAndView createUser(User user) {

     return mv;
 }
 //3、直接将Http请求对象传递给处理方法，方法返回的字符串代表逻辑视图名
 @RequestMapping（Path="/handle3"）
 public String handle3(HttpServlet request){
     return "main";
 }
```

1、RequestMappingHandlerAdapter默认已经装配了HttpMessageConverter(对象转换):

StringHttpMessageConverter, ByteArrayHttpMessageConverter, SourceHttpMessageConverter, AllEncompassingFormHttpMessageConverter

途径：@ResponseBody和@RequestBody对处理方法进行标注

HttpEntity/ResponseEntity作为处理方法的入参或返回值。

2、 使用@RestController和AsyncRestTemplate

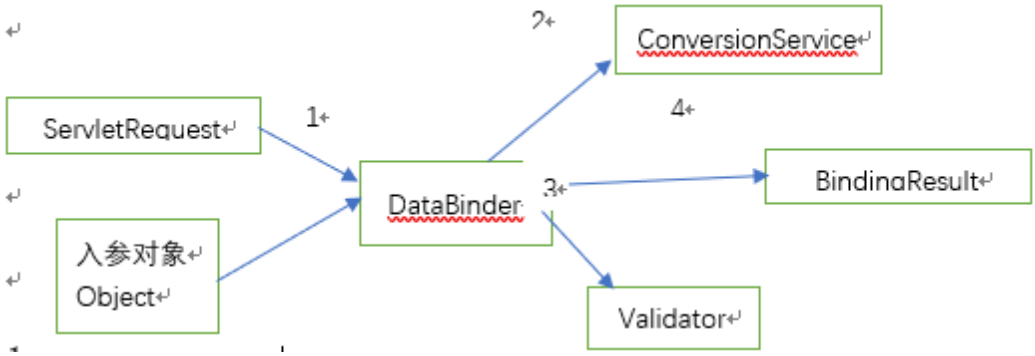
@RestController=@ResponseBody+@Controller(疑问：什么时候使用@ResponseBody和@RequestBody?)

4 处理模型数据

输出模型数据途径：

```
1 ModelAndView:处理方法返回值类型为ModelAndView时，方法体即可通过该对象添加模型数据
//添加模型
ModelAndView addObject()
ModelAndView addAllObjects()
//设置视图
void setView()
void setViewName()
2 @ModelAttribute:在方法入参标注该注解后，入参的对象就会放到数据模型中
//在方法入参前使用@ModelAttribute
@RequestMapping(path = "/handle6")
public String handle6(@ModelAttribute("user") User user)
3 Map及Model:如果方法入参为...ModelMap或..Map,处理方法返回Map中的数据会自动添加到模型中
//根据处理方法有Map或Model类型
@RequestMapping(path = "/handle6")
public String handle6(ModelMap modelMap){
modelMap.addAttribute("testAttr","valuel");
User user=(User)modelMap.get("user");
4 @SessionAttribute:将模型中的某个属性暂存到Httpsession中，以便多个请求之间可以共享这个属性。
//标注@SessionAttributes，SpringMVC会将模型中对应属性暂存到HttpSession
@SessionAttribute("user")
public class test {
//@SessionAttribute("user")会自动将本处理器中的任何处理方法属性名为user的模型属性透明的存储到HttpSession中。
```

5 处理方法的数据绑定



5.1 ConversionService

完成JAVA类型转换工作：

```
<!-- 装配自定义conversionService-->
<mvc:annotation-driven conversion-service="conversionService"/>
<bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactoryBean">
    <property name="converters">
        <list>
            <bean class="com.smart.domain.User"/>
        </list>
    </property>
</bean>
```

数据格式化：装配FormattingConversionServiceFactoryBean

```
<mvc:annotation-driven conversion-service="conversionService"/>
<bean id="conversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean
"*>     <property name="converters">
```

举例：

```
@DateTimeFormat(pattern = "yyyy-MM-dd")
private Date birthday;
@NumberFormat(pattern = "#,###.#")
private long salary;
```

5.2 数据校验

Spring校验框架：定义Bean

```
<bean id="validator"
class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"/
>
```

SpringMVC校验;

通过mvc:annotation-driven会默认装配LocalValidatorFactoryBean，只需在方法入参标注@Valid

获取校验结果：BindingResult bindingResult

```
public String handle62(@Valid @ModelAttribute("user") User user,
                       BindingResult bindingResult)
```

在页面中显示错误：

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<form:errors path="username" cssClass="errorClass"/>
```

6 视图和视图解析器

InternalResourceViewResolver//解析为URL文件，如JSP

模板视图：freeMarker,Excal,PDF,xml,json.

6.1 freeMarker

文件名：***.ftl。

显示页面：<#list userList as user> </#list>

装配FreeMarker:

```
<bean class="…"
P:templateLoaderPath="WEB/INF/ftl"
P:defaultEncoding="UTF-8">
<property name="freemakersetting">
<props>
<prop kry="classic_compatible">true</prop>
</props>
```

6.2 Excel

继承AbstractXlsView类 //注意。Excel文档名称必须编码为iso-8859-1.

在***-servlet.xml文件配置Bean.

6.3 输出XML

XML形式的视图对象为:MarshallingView

在***-servlet.xml文件配置Bean.

```
<bean id="userListXML"
class="org.springframework.web.servlet.view.xml.MarshallingView"
       p:modelKey="userList"
       p:marshaller-ref="xmlMarshaller"/> <bean id="xmlMarshaller"
class="org.springframework.oxm.xstream.XStreamMarshaller">
</bean>
```

7 本地化解析

7.1 cookieLocaleResolver

```
  <bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver"
      p:cookieName="clientLanguage"
      p:cookieMaxAge="1000000"
      p:cookiePath="/"
      p:defaultLocale="zh_CN"/>
```

7.2 sessionLocaleResolver

```
<bean id="localeResolver"
 class="org.springframework.web.servlet.i18n.SessionLocaleResolver"/>
```

7.3 localeChangeInterceptor

```
<mvc:interceptors>
<bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"*/>
</mvc:interceptors>
```