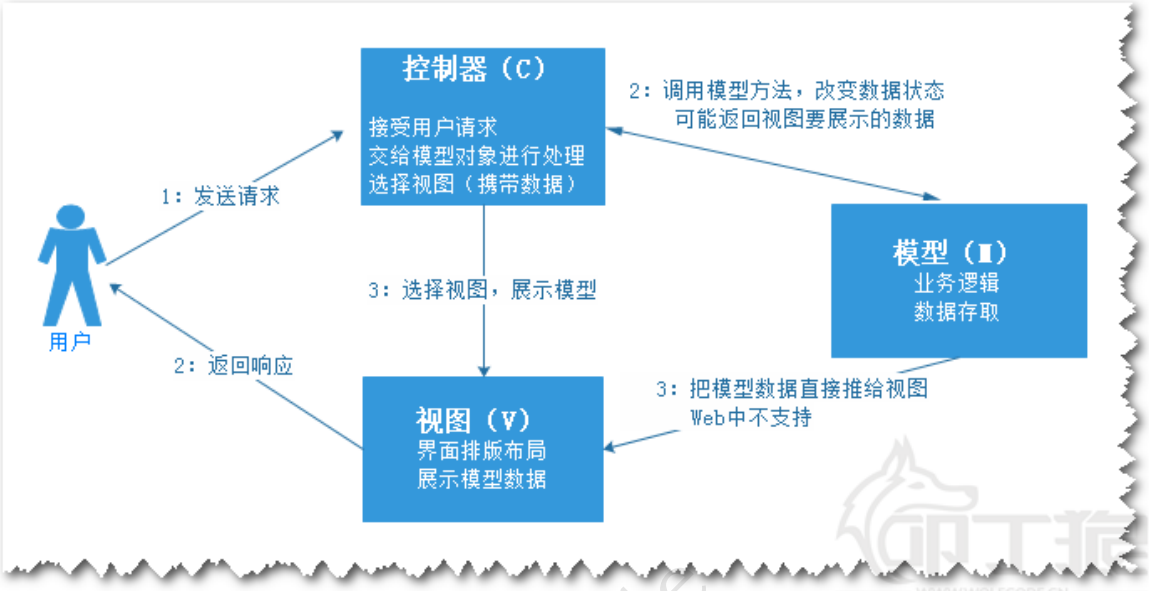
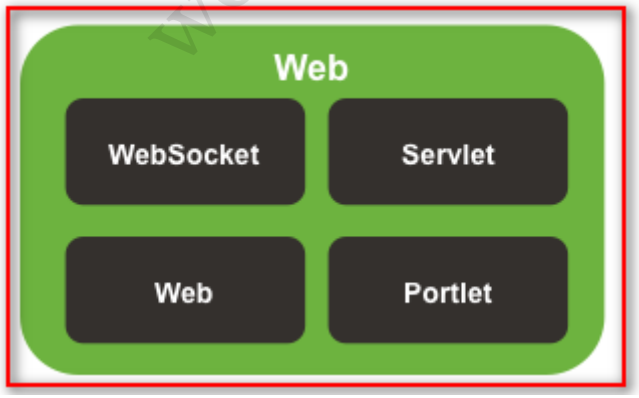
一、Spring Web 框架介绍

1、MVC 思想



JavaBean、JSP、Servlet，减少耦合，提高可维护性。

2、Spring MVC



- Servlet：Web 服务的模块，包含对 MVC 与 REST 的实现，Spring MVC。
- Web：提供与 Web 的集成，基于 Web 应用程序上下文。
- WebSocket：实现客户端与服务端主动通信。
- Portlet：提供了在 Portlet 环境中实现 MVC。

Spring MVC 是 Spring 对 MVC 思想实现，其有以下好处：

- 它解决 Web 开发中常见的问题（参数接收、文件上传、表单验证、国际化等），而且使用简单，与 Spring 无缝集成。
- Spring3.0 后全面超越 Struts2，成为最优秀的 MVC 框架 （更安全，性能更好，更简单）。
- 支持 RESTful 风格的 URL 请求，非常容易与其他视图技术集成，如 Velocity、FreeMarker、JSP 等。
- 采用了松散耦合可插拔组件结构，比其他 MVC 框架更具扩展性和灵活性。

3、其它 MVC 框架

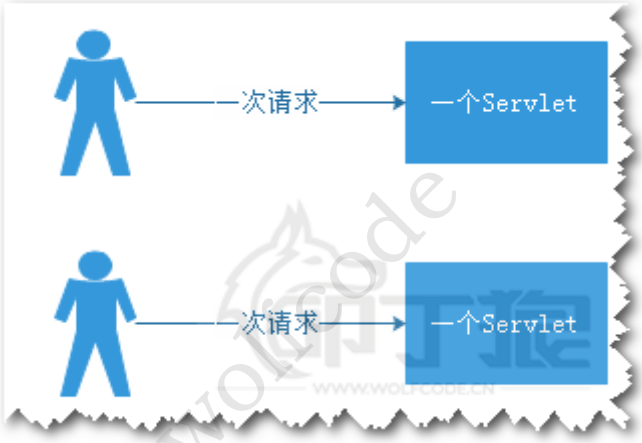
- Struts1 --> WebWork --> Struts2
- Spring MVC
- JSF

二、前端控制器

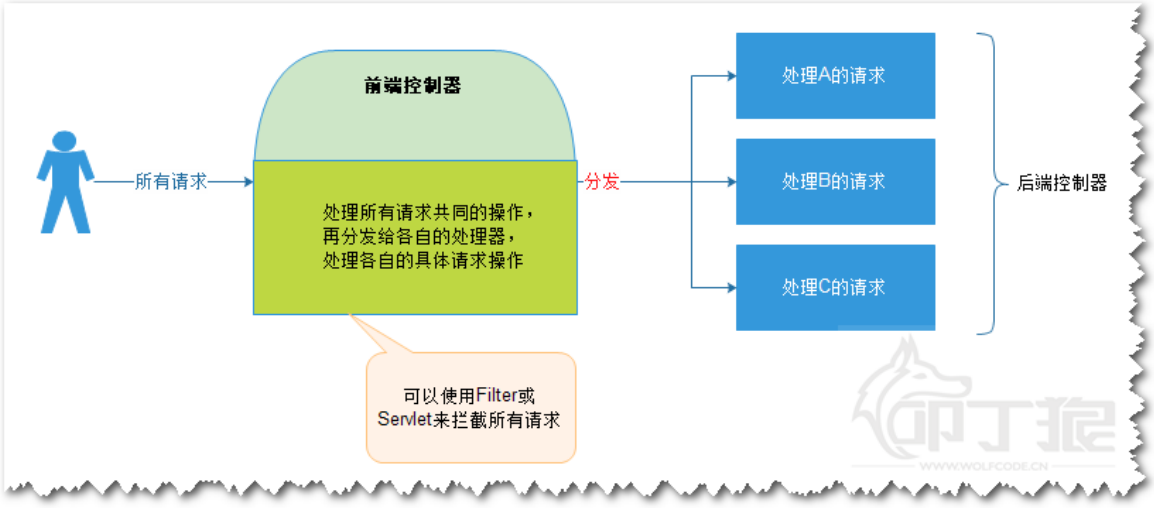
1、什么是前端控制器

在 MVC 框架中都存在一个前端控制器，在 WEB 应用的前端（Front）设置一个入口控制器（Controller），是用来提供一个集中的请求处理机制，所有的请求都被发往该控制器统一处理，然后把请求分发给各自相应的处理程序。一般用来做一个共同的处理，如权限检查，授权，日志记录等。因为前端控制的集中处理请求的能力，因此提高了可重用性和可拓展性。

1.1、没有前端控制器



1.2、有前端控制器



2、Spring MVC 中的前端控制器

Spring MVC 已经提供了一个 DispatcherServlet 类作为前端控制器，所以要使用 Spring MVC 必须在 web.xml 中配置前端控制器。

把处理请求的对象称之为处理器或后端控制器，Spring MVC 中习惯称之为 Controller，如处理员工请求的就会命名为 EmployeeController。

三、使用注解方式开发 HelloWorld 程序

1、新建 Maven 项目

项目名 springmvc-demo，注意打包方式是 war。拷贝依赖和插件：

```
<groupId>cn.wolfcode</groupId>
<artifactId>springmvc-demo</artifactId>
<version>1.0.0</version>
<packaging>war</packaging>

<properties>
    <spring.version>5.0.8.RELEASE</spring.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.0.1</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.1</version>
            <configuration>
                <port>80</port>
                <path>/</path>
                <uriEncoding>UTF-8</uriEncoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```

wolfcode

2、配置前端控制器

在 main 目录新建 webapp/WEB-INF/web.xml，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```
                      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

    <!-- Spring MVC 前端控制器-->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 指定 Spring MVC 加载的配置文件-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:mvc.xml</param-value>
        </init-param>
        <!-- Tomcat 启动初始化 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

3、编写一个处理器类及 JSP

```
package cn.wolfcode.web.controller;

@Controller
public class HelloController {
    // 提供方法处理请求，在浏览器地址栏输入如下 localhost/hello，就会执行下面的方法
    @RequestMapping("/hello")
    public ModelAndView save() {
        ModelAndView mv = new ModelAndView();
        // 往作用域或者模型中存入数据
        mv.addObject("msg", "Hello Spring MVC");
        // 找视图
        mv.setViewName("/WEB-INF/views/welcome.jsp");
        return mv;
    }
}
```

对应上面的代码路径建 JSP 文件，使用 EL 表达式取值即可。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>入门</title>
</head>
<body>
    ${msg}
</body>
</html>
```

4、编写 Spring MVC 配置文件

在 resources 目录下新建 mvc.xml，配置如下：

```
<!-- 配置 IoC DI 注解解析器，让 Spring 我们创建 HelloController 类的对象 -->
<context:component-scan base-package="cn.wolfcode.web.controller"/>

<!-- MVC 注解解析器 -->
<mvc:annotation-driven/>
```

5、测试

打开浏览器，在地址栏收入 localhost/hello，访问看下效果。

四、HelloWorld程序释疑

1、前端控制器初始化

load-on-startup 元素是可选的：若值为 0 或者大于 0 时，表示容器在应用启动时就构建 Servlet 并调用其 init 方法做初始化操作（非负数的值越小，启动该 Servlet 的优先级越高）；若值为一个负数时或者没有指定时，则在第一次请求该 Servlet 才加载。

配置的话，就可以让 SpringMVC 初始化的工作在容器启动的时候完成，而不是丢给用户请求去完成，提高用户访问的体验性。

2、Spring MVC 的配置文件的位置

DispatcherServlet 默认会去 WEB-INF 目录下去按照 servletName-servlet.xml 路径去加载 Spring MVC 配置文件。若 servlet-name 元素文本内容是　hello，则去 WEB-INF 目录下寻找 hello-servlet.xml 文件。

一般地，我们从 classpath 路径，显示去加载 SpringMVC 的配置文件。

```
<init-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath:mvc.xml</param-value>
</init-param>
```

3、MVC 注解

3.1、RequestMapping 注解

RequestMapping 注解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。还可以限定请求类型和响应类型等。

3.2、MVC 注解解析器

配置了会往容器中存入 RequestMappingHandlerMapping、RequestMappingHandlerAdapter、ExceptionHandlerExceptionResolver 等三个 bean。除此之外：

- 支持使用 ConversionService 实例对表单参数进行类型转换。
- 支持使用 @NumberFormat、@DateTimeFormat 注解完成数据格式化操作。
- 支持使用 @Valid 注解对 JavaBean 实例进行 JSR303 验证。
- 支持使用 @RequestBody 和 @ResponseBody 注解读写 JSON。

五、前端控制器映射路径

1、映射路径

- 配置如 *.do*、.htm 是最传统方式，可以访问静态文件（图片、 JS、 CSS 等），但不支持 RESTful 风格。
- 配置成 /，可以支持流行的 RESTful 风格，但会导致静态文件（图片、 JS、 CSS 等）被拦截后不能访问。
- 配置成 /*，是错误的方式，可以请求到 Controller 中，但跳转到调转到 JSP 时被拦截，不能渲染 JSP 视图，也会导致静资源访问不了。

可以在 webapp 目录下新建 static/demo.html，访问验证上面的说法。

2、访问静态资源和 JSP 被拦截的原因

Tomcat 容器处理静态资源是交由内置 DefaultServlet 来处理的（拦截路径是 /），处理 JSP 资源是交由内置的 JspServlet 处理的（拦截路径是*.jsp | *.jspx）。

启动项目时，先加载容器的 web.xml，而后加载项目中的 web.xml。当拦截路径在两者文件中配置的一样，后面会覆盖掉前者。

所以前端控制器配置拦截路径是 / 的所有静态资源都会交由前端控制器处理，而拦截路径配置 /*，所有静态资源和 JSP 都会交由前端控制器处理。

wolfcode

3、怎么让静态资源可以访问到

3.1、方式一

在 web.xml 中修改，修改前端控制器的映射路径如下：

```
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

但注意，访问控制器里的处理方法时，请求路径须携带 .do。

3.2、方式二

在 mvc.xml，配置如下即可：

```
<mvc:default-servlet-handler/>
```

上述配置会在 Spring MVC 上下文中创建存入一个 DefaultServletHttpRequestHandler 的 bean，它会对进入 DispatcherServlet 的请求进行筛查，若不是映射的请求，就将该请求交由容器默认的 Servlet 处理。

六、处理响应

响应处理是指怎么编写控制器里面的处理方法接受请求做响应，找视图文件和往作用域中存入数据。要处理方法要做响应，一般处理方法返回的类型为 ModelAndView 和 String。

1、返回 ModelAndView

1.1、编写处理方法

方法中返回 ModelAndView 对象，此对象中设置模型数据并指定视图。

```
package cn.wolfcode.web.controller;

@Controller
public class ResponseController {
    // 提供方法处理请求，localhost/resp1.do
    @RequestMapping("/resp1")
    public ModelAndView resp1() {
        // 通过创建这个类对象，告诉 Spring MVC 找什么视图文件， 往作用域或者说往模型中存入什么数据
        ModelAndView mv = new ModelAndView();
        // 往作用域或者模型中存入数据
        mv.addObject("msg", "方法返回类型是 ModelAndView");
        // 找视图
        mv.setViewName("/WEB-INF/views/resp.jsp");
        return mv;
    }
```

1.2、编写 JSP

在对应路径下新建 resp.jsp，内容如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>响应</title>
</head>
<body>
    ${msg}
</body>
</html>
```

wolfcode

1.3、添加数据到模型方法对比

- addObject(String key, Object value)：设置共享数据的 key 和 value。
- addObject(Object value)：设置共享数据的 value，key 为该 value 类型首字母小写。

2、返回 String

返回 String 类型（使用广泛），要共享数据，此时和 Model 参数组合使用，用其往作用域或模型中存入数据。

2.1、编写处理方法

```java
package cn.wolfcode.web.controller;

@Controller
public class ResponseController {
    // 提供方法处理请求，localhost/resp2.do
    @RequestMapping("/resp2")
    public String resp2(Model model) {
        // 往作用域或者模型中存入数据
        model.addAttribute("msg", "方法返回类型是 String");
        // 返回视图名
        return "/WEB-INF/views/resp.jsp";
    }
}
```

3、消除视图前缀和后缀

因为视图会固定放在一个位置，又因为统一选用一种视图技术，所以视图后缀名也是一样的，那么在处理方法设置视图路径的代码随着项目变大，存在着大量重复。而 Spring MVC 提供可通过配置的方法消除视图路径重复，只需要在 mvc.xml，配置如下即可：

```xml
<!--
    配置视图解析器
    配置这个，那么 Spring MVC 找视图的路径就是：前缀 + 逻辑视图名（处理方法设置或返回视图名）
+ 后缀名
-->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- 视图前缀 -->
    <property name="prefix" value="/WEB-INF/views/"/>
    <!-- 视图后缀 -->
    <property name="suffix" value=".jsp"/>
</bean>
```

wolfcode

七、请求转发及 URL 重定向

1、请求转发和 URL 重定向对比

	地址栏	WEB-INF	共享请求数据	表达重复提交
请求转发	不变	可以访问	共享	有
URL 重定向	变	不可访问	不共享	无

2、请求转发

加上 forward 前缀方式，表示请求转发，相当于 request.getRequestDispatcher().forward(request,response)，转发后浏览器地址栏不变，共享之前请求中的数据。

2.1、编写处理方法

```java
package cn.wolfcode.web.controller;

@Controller
public class ResponseController {
    // localhost/f.do
    @RequestMapping("/f")
    public String forward() {
        return "forward:/WEB-INF/views/welcome.jsp";
    }
}
```

3、URL 重定向

加上 redirect 前缀方式，表示重定向，相当于 response.sendRedirect()，重定向后浏览器地址栏变为重定向后的地址，不共享之前请求的数据。

3.1、编写处理方法

```java
package cn.wolfcode.web.controller;

@Controller
public class ResponseController {
    // localhost/r.do
    @RequestMapping("/r")
    public String redirect() {
        return "redirect:/static/demo.html";
    }
}
```

wolfcode

4、路径问题

在请求转发或者 URL 重定向时，若路径：

- 加了 /，使用是绝对路径（推荐使用），从项目根路径找；
- 没加 /，使用是相对路径，相对于上一次访问上下文路径的上一级找。

/response/test6.do ---> "redirect:/hello.html" ---> localhost:/hello.html

/response/test6.do ---> "redirect:hello.html" ---> localhost:/response/hello.html

八、处理简单类型请求参数

处理方法如何获取请求中的参数，而这些参数值是期望用基本数据类型及其包装类、String 和 BigDecimal 等形参接收。

1、请求参数名和处理方法方法的形参同名

比如请求路径 /req1.do?username=xx&age=18，就需要编写如下处理方法来接受请求的参数。

```java
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req1")
    public ModelAndView resp1(String username, int age) {
        System.out.println(username);
        System.out.println(age);
        return null;
    }
}
```

2、请求参数名和处理方法方法的形参不同名

比如请求路径 /req2.do?username=xx&age=18，就需要编写如下处理方法来接受请求的参数。

```java
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req2")
    // 此时请求参数为 name，而形参名为 username
    public ModelAndView resp2(@RequestParam("username")String name, int age) {
        System.out.println(name);
        System.out.println(age);
        return null;
    }
}
```

wolfcode

3、乱码处理

3.1、GET 方式传递中文参数乱码问题

Tomcat7 及之前手动修改配置文件，使用代码手动转；但若使用 Maven 的话，可以 pom.xml 中的 Tomcat 插件配置一行也可以解决：

```xml
<uriEncoding>UTF-8</uriEncoding>
```

Tomcat8 已处理这个问题，不需要额外配置。

3.2、POST 方法传递中文乱问题

在 web.xml 直接使用 Spring MVC 内置的过滤器来处理。

```
<!-- 编码过滤器，仅针对 POST 方式 -->
<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>
```

九、处理复合类型请求参数

处理方法如何获取请求中的参数，而这些参数值是期望用自定义类型、数组类型等形参接收。

1、数组类型

比如请求路径 /req3.do?ids=1&ids=2&ids=3，就需要编写如下处理方法来接受请求的参数。

```
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req3")
        public ModelAndView resp3(Long[] ids) {
        System.out.println(Arrays.toString(ids));
        return null;
    }
}
```

wolfcode

2、自定义类型

比如有个保存用户的需求，请求时携带用户名和密码参数，后端期望创建一个用户对象来封装这些参数值。

2.1、自定义类

用来封装参数值。

```
package cn.wolfcode.domain;

public class User {
    private Long id;
    private String Username;
    private String password;

    // 省略 setter getter toString
}
```

2.2、编写处理方法

比如请求路径 /req4.do?username=hehe&password=666，**注意传递参数名与封装对象的属性名一致**，就需要编写如下处理方法。

```
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req4")
    public String resp4(User user) {
        System.out.println(user);
        return null
    }
}
```

底层 Spring MVC 根据请求地址对应调用处理方法，调用方法时发现要传递 User 类型的实参，Spring MVC 会反射创建 User 对象，之后通过请求参数名找对应的属性，给对象的属性设置对应的参数值。

十、处理日期类型请求参数

1、引入

开发中处理时间字符串的格式是比较常遇到的问题，之前你们是怎么做的？那使用 Spring MVC 肯定帮我们简化很多工作，但有些东西是简化不了的，你们觉得是什么？

2、处理日期格式参数

2.1、在处理方法的 Date 类型的形参贴上 @DateTimeFormat

比如请求路径 /req5.do?date=2020-05-19，就需要编写如下处理方法来接受请求的参数。

```
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req5")
    // 注意形参的类型为 java.util.Date
    public ModelAndView resp5(@DateTimeFormat(pattern="yyyy-MM-dd")Date date) {
        System.out.println(date.toLocaleString());
        return null;
    }
}
```

2.2、在封装参数类的 Date 类型的字段上贴 @DateTimeFormat

2.2.1、修改 User 类

```java
package cn.wolfcode.domain;

public class User {
    private Long id;
    private String Username;
    private String password;
    // 增加下面这个字段，并贴注解
    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date date;

    // 省略 setter getter toString
}
```

2.2.2、编写处理方法

比如请求路径 /req6.do?date=2020-05-19，就需要编写如下处理方法来接受请求的参数。

```java
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req6")
    public ModelAndView resp6(User user) {
        System.out.println(user);
        return null;
    }
}
```

wolfcode

十一、ModelAttribute 注解使用

可以贴在方法和形参上，这里重点讲贴在形参。形参前提是自定义类型，会存到模型中，可在视图中获取到。**后期项目用于查询条件回显用。**

1、未使用注解

1.1、编写处理方法

```java
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req7")
    public String resp7(User user) {
        return "m";
    }
}
```

1.2、编写 JSP

新建 webapp/WEB-INF/views/m.jsp，文件内容如下：

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>ModelAttribute 注解</title>
</head>
<body>
    ${user}
</body>
</html>
```

1.3、访问测试

访问路径 请求路径 /req7.do?username=hehe&password=666

2、使用注解

1.1、修改处理方法

```java
package cn.wolfcode.web.controller;

@Controller
public class RequestController {
    @RequestMapping("/req7")
    // 给形参贴 @ModelAttribute，用来修改存在模型中使用的 key
    public String resp7(@ModelAttribute("u")User user) {
        return "m";
    }
}
```

wolfcode

1.2、修改 JSP

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>ModelAttribute 注解</title>
</head>
<body>
    ${u}
</body>
</html>
```

1.3、访问测试

访问路径 请求路径 /req7.do?username=hehe&password=666

十二、文件上传

1、引入

回顾之前使用 Servlet3.0 来解决文件上传的问题，编写上传表单（POST、multipart/form-data），还在处理方法 doPost 中编写解析上传文件的代码。但现在我们使用 Spring MVC，其可以帮我们简化开发。你们觉得能简化什么地方?

2、准备上传表单

新建 webapp/static/upload.html，注意请求数据类型必须是：multipart/form-data，且请求方式是 POST。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>文件上传</title>
</head>
<body>
    <form action="/upload.do" method="POST" enctype="multipart/form-data">
        文件:<input type="file" name="pic"><br>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

3、修改 web.xml

设置文件上传大小等。

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
    <multipart-config>
        <max-file-size>52428800</max-file-size>
        <max-request-size>52428800</max-request-size>
    </multipart-config>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

4、配置上传解析器

在 mvc.xml，配置上传解析器，注意上传解析器这个 **bean 名称是固定的，必须为 multipartResolver**。

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.support.StandardServletMultipartResolve
r"/>
```

5、编写上传控制器

```java
package cn.wolfcode.web.controller;

@Controller
public class UploadController {

    // Spring 容器存在 ServletContext 类型的对象，所以定义好 ServletContext 类型字段贴 @Autowired 注解即可获取到
    @Autowired
    private ServletContext servletContext;

    @RequestMapping("/upload")
    public ModelAndView upload(Part pic) throws Exception {
        System.out.println(pic.getContentType());       // 文件类型
        System.out.println(pic.getName());              // 文件参数名
        System.out.println(pic.getSize());              // 文件大小
        System.out.println(pic.getInputStream());       // 文件输入流

        // FileCopyUtils.copy(in, out)，一个 Spring 提供的拷贝方法

        // 获取项目 webapp 目录下 uploadDir 目录的绝对路径
        System.out.println(servletContext.getRealPath("/uploadDir"));

        return null;
    }
}
```

十三、拦截器

1、定义及作用

Spring MVC 提供的，若配置在 Spring MVC 配置文件中，则被 Spring 容器管理，主要用于对请求访问控制器的方法进行拦截，在后面的项目中用于登录与权限判断。

2、自定义拦截器

2.1、定义一个类实现 HandlerInterceptor

```java
package cn.wolfcode.web.interceptor;

public class MyInterceptor implements HandlerInterceptor {

    // 处理方法之前
    // 返回 false，就被拦截；返回 true 就放行
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        System.out.println("preHandle");
        return true;
    }

    // 处理方法之后，渲染视图之前
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
```

wolfcode

```
        System.out.println("postHandle");
    }

    // 渲染视图之后
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        System.out.println("afterCompletion");
    }
}
```

2.2、配置拦截器

在 mvc.xml，配置如下

```
<mvc:interceptors>
    <!-- 配置拦截器 -->
    <mvc:interceptor>
        <!-- 拦截的路径 -->
        <mvc:mapping path="/**"/>
        <!-- 被排除的路径
        <mvc:exclude-mapping path="排除的路径"/> -->
        <!-- 拦截器类 -->
        <bean class="cn.wolfcode.web.interceptor.MyInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

2.3、测试

在浏览器输入地址访问某个控制器中某个方法，看下控制台的打印结果。

十四、核心组件说明（拓展）

1、核心组件

1.1、前端控制器 DispatcherServlet

不需要我们开发，由框架提供，需要在 web.xml 中配置。作用：接受请求，处理响应结果，转发器，中央处理器。

1.2、处理器映射器 HandlerMapping

不需要我们开发，由框架提供。作用，更具请求的 URL 找到对应的 Handler。

1.3、处理器适配器 HandlerAdapter

不要我们开发，由框架提供。作用：调用处理器（Handler/Controller）的方法。

1.4、处理器 Handler（又名Controller）

需要我们开发，必须按照 HandlerAdapter 的规范去开发。作用：接收用户请求数据，调用业务方法处理请求。

1.5、视图解析器 ViewResolver

不需要我们开发，有框架或者第三方提供。作用：视图解析，把逻辑视图名称解析成真正的物理视图。支持多种视图技术 Velocity、FreeMarker、JSP 等。

1.6、视图

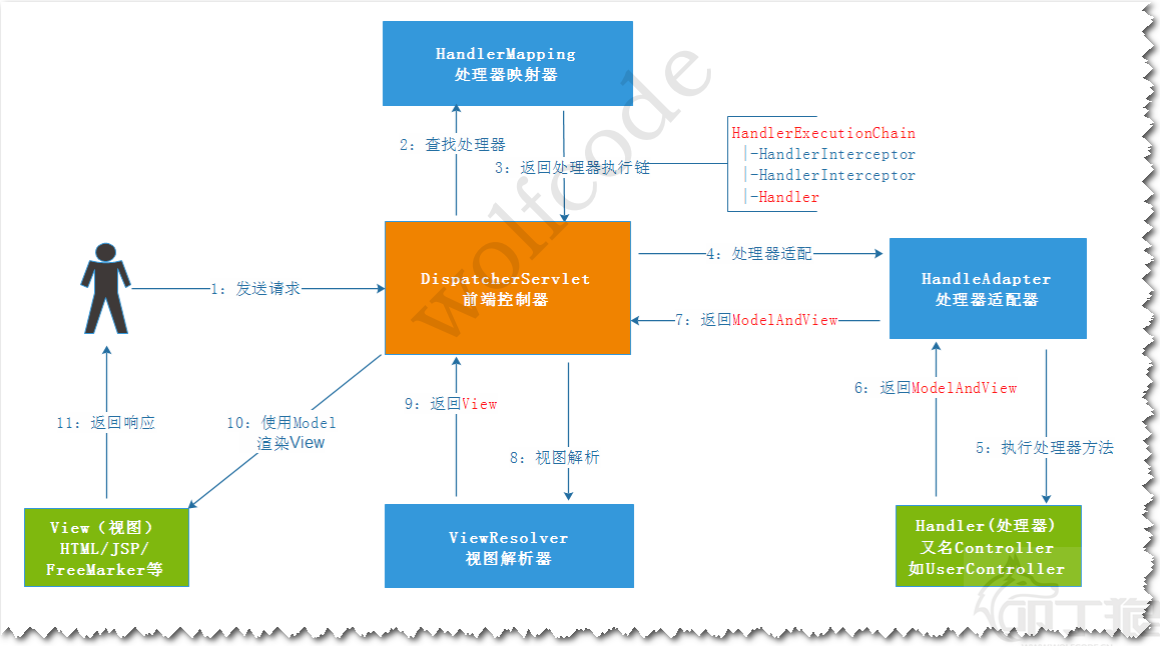
需要我们开发。作用：把数据展现给用户。

2、开发步骤

- 在 web.xml 配置前端控制器
- 在 mvc.xml 配置处理器映射器、处理适配器、视图解析器（用默认的可不配置）。
- 需要开发（结合需求）：
  - 先开发 Contoller，再配置。
  - 开发 JSP。

十五、执行流程分析（拓展）

1、图示流程



2、文字描述

- 用户发送出请求到前端控制器 DispatcherServlet。
- DispatcherServlet 收到请求调用 HandlerMapping（处理器映射器）。
- HandlerMapping 找到具体的处理器（通过 XML 或注解配置），生成处理器对象及处理器拦截器（若有），再一起返回给 DispatcherServlet。
- DispatcherServlet 调用 HandlerAdapter（处理器适配器）。
- HandlerAdapter 经过适配调用具体的处理器的某个方法（Handler/Controller）。
- Controller 执行完成返回 ModelAndView 对象。
- HandlerAdapter 将 Controller 返回的 ModelAndView 再返回给 DispatcherServlet。
- DispatcherServlet 将 ModelAndView 传给 ViewReslover（视图解析器）。
- ViewReslover 解析后返回具体 View（视图）。
- DispatcherServlet 根据 View 进行渲染视图（即将模型数据填充至视图中）。
- DispatcherServlet 响应用户。

十六、源码分析执行流程（拓展）

即通过代码来证明上面图的流程。

wolfcode