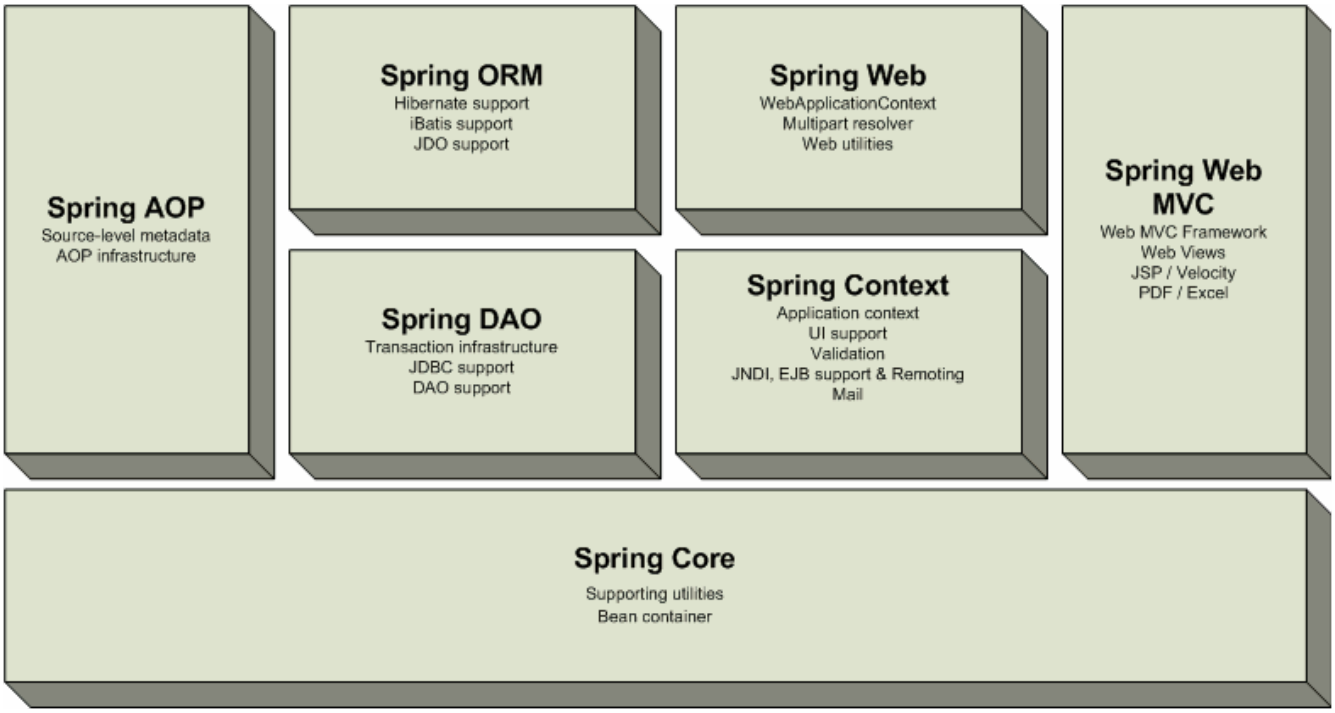
Spring架构



Spring MVC

1 SpringMVC 概述

Spring 为展现层提供的基于 MVC 设计理念的优秀的 Web 框架，是目前最主流的 MVC 框架之一

Spring3.0 后全面超越 Struts2，成为最优秀的 MVC 框架

Spring MVC 通过一套 MVC 注解，让 POJO 成为处 理请求的控制器，而无须实现任何接口。

支持 REST 风格的 URL请求

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

2 Spring Web MVC是什么

Spring Web MVC是一个MVC的开源框架，SpringMVC是spring的一个后续产品，是spring在原有基础上，又提供了web应用的MVC模块，可以简单的把springMVC理解为是spring的一个模块（类似AOP，IOC这样的模块），springMVC和spring无缝集成，(其实springMVC就是spring的一个子模块，所以根本不需要同spring进行整合)。

Spring Web MVC是一种基于Java的实现了Web MVC设计模式的请求驱动类型的轻量级Web框架，即使用了MVC架构模式的思想，将web层进行职责解耦，基于请求驱动指的就是使用请求-响应模型，框架的目的就是帮助我们简化开发，Spring Web MVC也是要简化我们日常Web开发的。

3 Spring Web MVC能帮我们做什么

让我们能非常简单的设计出干净的Web层和薄薄的Web层； √进行更简洁的Web层的开发； √天生与Spring框架集成（如IOC容器、AOP等）； √提供强大的约定大于配置的契约式编程支持； √能简单的进行Web层的单元测试； √支持灵活的URL到页面控制器的映射； √非常容易与其他视图技术集成，如Velocity、FreeMarker等等，因为模型数据不放在特定的API里，而是放在一个Model里（Map数据结构实现，因此很容易被其他框架使用）； √非常灵活的数据验证、格式化和数据绑定机制，能使用任何对象进行数据绑定，不必实现特定框架的API； √提供一套强大的JSP标签库，简化JSP开发； √支持灵活的本地化、主题等解析； √更加简单的异常处理； √对静态资源的支持； √支持Restful风格。

4 搭建的第一个SpringMVC框架

1. 加入 jar 包
2. 在 web.xml 中配置 DispatcherServlet
3. 加入 Spring MVC 的配置文件
4. 编写处理请求的处理器，并标识为处理器
5. 编写视图

4.1 加入 jar 包

commons-logging-1.1.3.jar spring-aop-4.3.0.RELEASE.jar spring-beans-4.3.0.RELEASE.jar spring-context-4.3.0.RELEASE.jar spring-core-4.3.0.RELEASE.jar spring-expression-4.3.0.RELEASE.jar **spring-web-4.3.0.RELEASE.jar spring-webmvc-4.3.0.RELEASE.jar**

4.2 Web.xml配置DispatcherServlet

配置 DispatcherServlet： DispatcherServlet 默认加载 /WEB- INF/.xml 的 Spring 配置文件, 启动 WEB 层 的 Spring 容器。

可以通过 contextConfigLocation 初始化参数自定 义配置文件的位置和名称

```
<!-- SpringMVC Servlet DispatcherServlet -->
    <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:spring_mvc.xml</param-value>
        </init-param>
        <!-- <load-on-startup>1</load-on-startup> -->
    </servlet>
    <servlet-mapping>
    <!-- /:拦截所有的请求（.jsp除外）
        /*:拦截所有的请求（包括JSP）
     -->
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

4.3 加入 Spring MVC 的配置文件

```
<!-- 默认情况下，DispatcherServlet 会 /WEB-INF/Servet名称-servlet.xml [/WEB-
INF/dispatcherServlet-servlet.xml]
            自定义配置文件目录位置：contextConfigLocation -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring_mvc.xml</param-value>
        </init-param>
        <!-- <load-on-startup>1</load-on-startup> -->
```

4.4 编写处理请求的处理器，并标识为处理器

```
/*
 * RequestMapping在类定义处指定的URL相对于web应用的部署路径 请求如：项目名/users
 * 在方法处指定的URL则相对于类定义
 */
@Controller
@RequestMapping(value="/users")
public class UsersAction {

    @RequestMapping("/add.action")    //请求的URL为：项目名/users/add.action
    public String add(){
        System.out.println("请求进来啦...");
        return "/users/add.jsp";   //返回到页面  /从应用根目录查找页面
    }
    @RequestMapping("/list.action")   //请求的URL为：项目名/users/list.action
    public String list(){
        System.out.println("请求进入list...");
        return "/users/list.jsp";
    }
```

5 @RequestMapping 映射请求

@RequestMapping可用于修饰在： –**类定义处**：提供初步的请求映射信息。相对于 WEB 应用的根目录 –**方法处**：提供进一步的细分映射信息。相对于类定义处的 URL。若类定义处未标注 @RequestMapping，则方法处标记的 URL 相对于WEB 应用的根目录

在处理方法入参处使用 @RequestMapping注解为控制器指定处理URL请求 – value：指定请求的实际地址(请求 URL) – method： 指定请求的method类型， GET、POST、PUT、DELETE等(请求方法) – params： 与 method 相类似，作用为了细化映射。只有当 URL 中包含与 params 值相匹配的参数的请求，处理方法才会被调用 例：params= {"uname", "age!=50"} – headers： 用于细化映射。只有当请求的 Request Headers 中包含与 heanders 值相匹配的参数，处理方法才会被调用(请求头)

@RequestMapping(method)：指定页面请求方式(method 的值一旦指定，那么，处理方法就只对指定的 http method 类型的请求进行处理)

- produces = "application/json; charset=utf-8"

在spring mvc中，@ResponseBody返回的默认编码为ISO-8859-1，但是实际需要的是UTF-8，所以需要转换

使用 @RequestMapping 映射请求 @RequestMapping 支持 Ant 风格的 URL•： • Ant 风格资源地址支持 3 种匹配符： ?：匹配文件名中的一个字符 *：匹配文件名中的任意字符 **： 匹配多层路径

```
/user/*/createUser: 匹配 –
/user/aaa/createUser、/user/bbb/createUser 等 URL
/user/**/createUser: 匹配 –
/user/createUser、/user/aaa/bbb/createUser 等 URL
/user/createUser??: 匹配 –
/user/createUseraa、/user/createUserbb 等 URL
```

6 请求处理方法签名

Spring MVC 通过分析处理方法的签名，将 HTTP 请求信 息绑定到处理方法的相应参数中。 Spring MVC 对控制器处理方法签名的限制是很宽松的， 几乎可以按喜欢的任何方式对方法进行签名。

必要时可以对方法及方法入参标注相应的注解（@RequestParam、@PathVariable等） Spring MVC 框架会将 HTTP 请求的信息绑定到相应的方法入参 中，并根据方法的返回值类型做出相应的后续处理。

@RequestParam

使用 @RequestParam 绑定请求参数值

在处理方法入参处使用 @RequestParam 可以把请求参 数传递给请求方法 – value或name：请求参数的参数名 – required：是否必须。默认为 true, 表示请求参数中必须包含对应 的参数，若不存在，将抛出异常 – defaultValue：请求参数的默认值

@PathVariable

注解：映射 URL 绑定的占位符

带占位符的 URL 是 Spring3.0 新增的功能，该功能在 SpringMVC 向 REST 目标挺进发展过程中具有里程碑的 意义

通过 @PathVariable 可以将 URL 中占位符参数绑定到控 制器处理方法的入参中：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("/delete/{id}")
public String delete(@PathVariable("id") Integer id){
    UserDao.delete(id);
    return "redirect:/user/list.action";
}
```

POJO

使用 POJO 对象绑定请求参数值

Spring MVC 会按请求参数名和 POJO 属性名进行自动匹 配，自动为该对象填充属性值。支持级联属性。 如：uname/ grade.gname等

```java
/*
 * 原理：通过名称，查找对应setXxxx()方法
 */
@RequestMapping("test2")
public String test2(Users user){
    System.out.println(user);
    System.out.println(user.getGrade().getGname()+"\t"+user.getUname());
    return "/users/list.jsp";
}
```

Servlet API

Spring MVC中，控制器类可以不依赖于ServletAPI对象，但是SpringMVC并不阻止用户使用。ServletAPI可以同时和其它入参同时使用。且位置顺序没有要求。（如果处理方法自行使用response返回响应，则处理方法返回值设为void即可）

HttpServletRequest HttpServletResponse HttpSession java.security.Principal Locale InputStream OutputStream Reader Writer

关于重定向

一般情况下，控制器方法返回字符串类型的值会被当成(物理)逻辑视图名处理 如果返回的字符串中带 **forward:** 或 **redirect:** 前缀 时，SpringMVC 会对他们进行特殊处理：将 forward: 和 redirect: 当成指示符，其后的字符串作为 URL 来处理 redirect:success.jsp：会完成一个到 success.jsp 的重定向的操作 forward:success.jsp：会完成一个到 success.jsp 的转发操作

```java
@RequestMapping("t2") // user/t2
public String t2(){
    System.out.println("forward....");
    return "forward:/index.jsp"; //默认是以转发的方式运行
}
@RequestMapping("t3") // user/t3
public String t3(){
    System.out.println("redircet....");
    return "redirect:/index.jsp"; //默认是以重定向的方式运行
}
```

处理模型数据

Spring MVC 提供了以下几种途径输出模型数据：

Map 及 Model: 入参为 org.springframework.ui.Model、org.springframework.ui. ModelMap 或 java.uti.Map 时，处理方法返回时，Map 中的数据会自动添加到模型中

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

Map 及 Model

Spring MVC 在调用方法前会创建一个隐含的模型对象作为模型数据的存储容器。
如果方法的入参为 Map 或 Model 类 型，Spring MVC 会将隐含模型的引用传递给这些入参。
在方法体内，开发者可以 通过这个入参对象访问到模型中的所有数 据，也可以向模型中添加新的属性数据

```
@RequestMapping(value="model/{id}",method=RequestMethod.GET) // user/model/1
public String selectByIdModel(@PathVariable("id")int id,HttpServletRequest request,
                              Model model){
    User user=userService.selectById(id);
    request.setAttribute("requestMsg","Request消息数据 ");
    model.addAttribute("modelMsg", "Model消息数据");
    model.addAttribute("user",user);

    return "/user/user-view.jsp"; //转发的方式
}
```

ModelAndView

控制器处理方法的返回值如果为 ModelAndView，则其既 包含视图信息，也包含模型数据信息。
- 添加模型数据：

```
MoelAndView addObject(String attributeName, Object attributeValue)
ModelAndView addAllObject(Map<String, ?> modelMap)
```

- 设置视图：

```
void setView(View view)
void setViewName(String viewName)
```

示例：

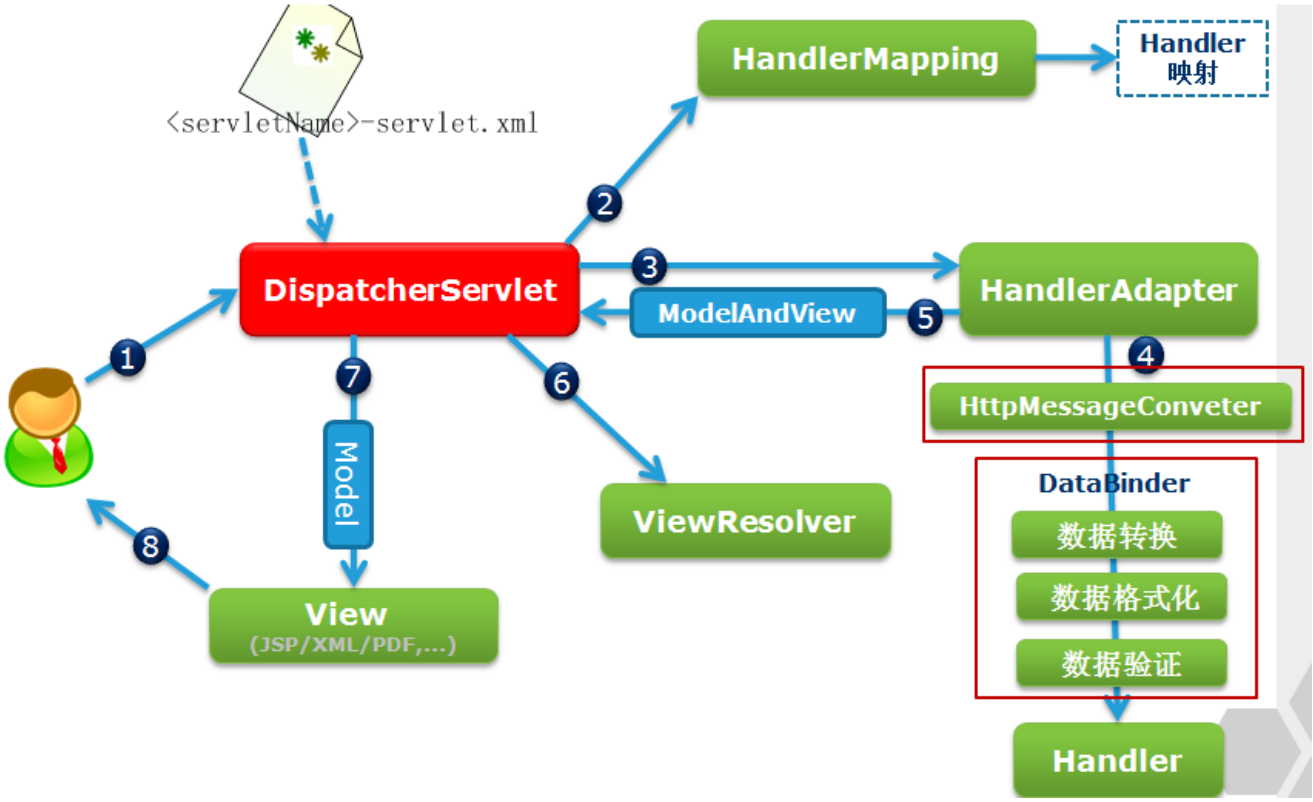
```
//ModelAndView  则其既 包含视图信息，也包含模型数据信息。
@RequestMapping(value="mv/{id}") // user/mv/11
public ModelAndView modelview(@PathVariable("id") int id){
    ModelAndView model=new ModelAndView();
    User user=userService.selectById(id);

    model.addObject("modelAndViewMsg", "ModelAndView消息数据");
    model.addObject("user", user);
    model.setViewName("/user/user-view.jsp");//视图信息
    return model;
}
```

7 SpringMVC编码格式 web.xml

```
<!-- SpringMVC编码格式处理UTF-8 -->
    <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>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

8 SpringMVC执行流程



1.用户发送请求至前端控制器DispatcherServlet 2.DispatcherServlet收到请求调用处理器映射器HandlerMapping。 3.处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain(包括处理器对象和处理器拦截器)一并返回给DispatcherServlet。 4.DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作 5.执行处理器Handler(Controller，也叫页面控制器)。 6.Handler执行完成返回ModelAndView 7.HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet 8.DispatcherServlet将ModelAndView传给ViewReslover视图解析器 9.ViewReslover解析后返回具体View 10.DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。 11.DispatcherServlet响应用户。