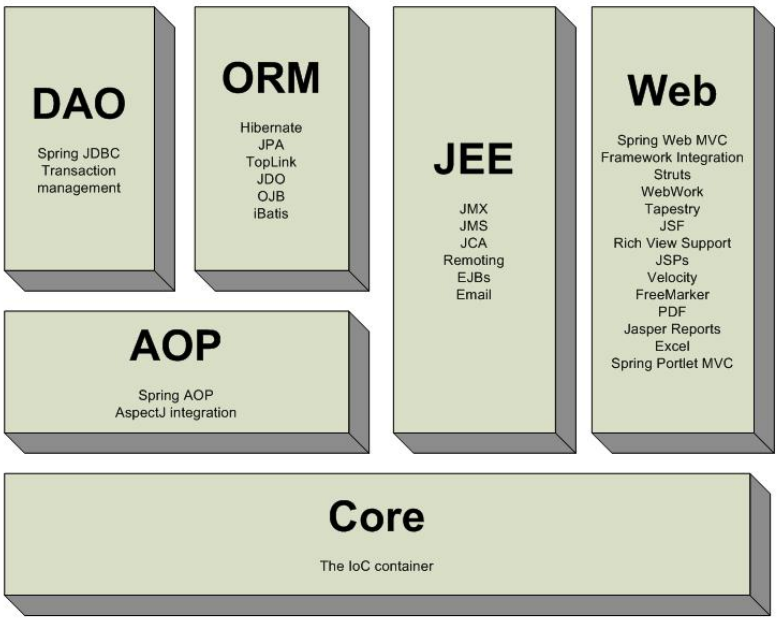
本章学习目标

- SpringMVC 简介
- SpringMVC 的入门案例
- SpringMVC 流程分析
- 配置注解映射器和适配器
- 配置视图解析器
- @RequestMapping 注解的使用
- 使用不同方式的跳转页面

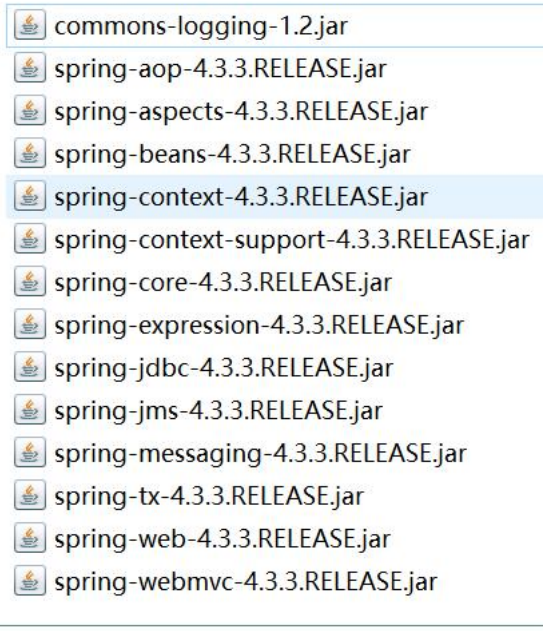
1. SpringMVC 简介

Spring web mvc 和 Struts2 都属于表现层的框架,它是 Spring 框架的一部分,我们可以从 Spring 的整体结构中看得出来：



2. SpringMVC 的入门案例

2.1. 建立 web 项目，导入 SpringMVC 相关支持 jar 包



2.2. 配置 web.xml（重点）

关键的步骤：配置核心控制器：Servlet

```
<!-- 配置 SpringMVC 的核心控制器 -->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</se
rvlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
```

2.3. 编写 spring-mvc.xml

建议放在 classpath 下面

有一个 mvc 的名称空间：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">


</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:contenxt="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context

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


    <!-- 扫描 Spring 的注解 -->
    <contenxt:component-scan
```

```
base-package="cn.sm1234.controller"></contenxt:component-scan>


</beans>
```

修改 web.xml 文件，

```
 <!-- 配置 SpringMVC 的核心控制器 -->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>


    <servlet-class>org.springframework.web.servlet.DispatcherServlet</se
rvlet-class>
    <!-- 参数 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
```

2.4. 编写 Controller 类

```
@Controller
public class HelloController {


    @RequestMapping("/hello")
    public ModelAndView hello(){
```

```
        System.out.println("执行 HelloController 的 hello 方法");
        //把数据保存到 ModelAndView（相当于保存 request 域对象）
        ModelAndView mv = new ModelAndView();
        mv.addObject("name", "Spring MVC");

        //返回物理路径
        mv.setViewName("/WEB-INF/jsp/success.jsp");
        return mv;
    }
}
```

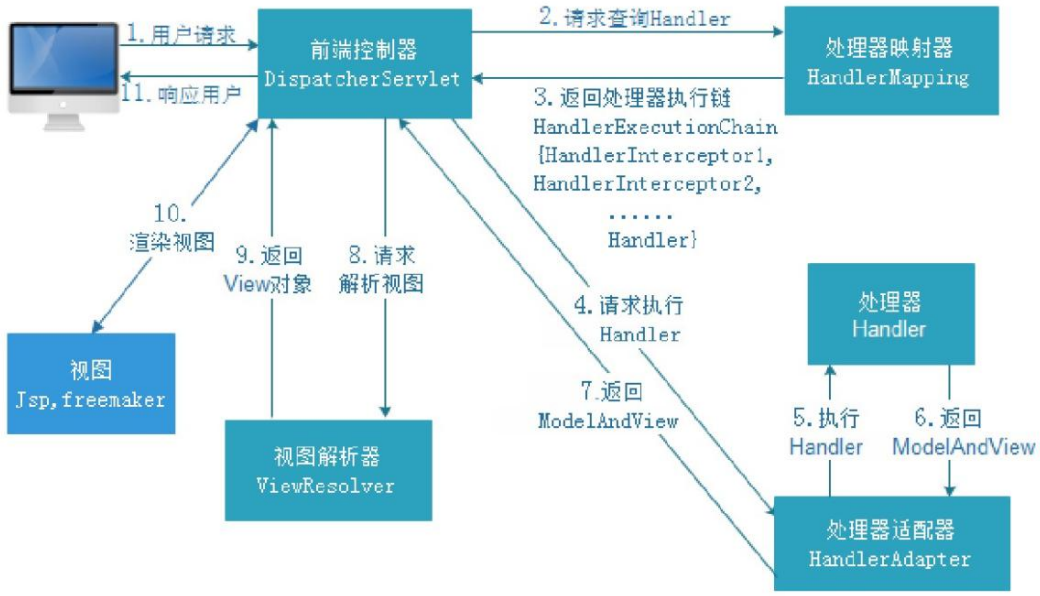
2.5. 编写页面

```
<body>
     第一个 SpringMVC 程序
    <hr/>
    ${name}
  </body>
```

2.6. 访问测试

访问路径：http://localhost:8080/ch01.spring-mvc/hello.action

3. SpringMVC 流程分析



4. 配置注解映射器和适配器

- 为什么需要配置配置注解映射器和适配器

默认情况下，SpringMVC 使用默认映射器（HandlerMapping）和适配器（HandlerAdpter）。

在 Spring3.1 之后，默认的 HandlerMapping 和 HandllerAdpter 是过时的。不建议使用。

默认的 HandlerMapping：

```
org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping
```

默认的 HandlerAdapter:

```
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter
```

Spring3.1 之后，建议使用：

新的 HandlerMapping:

```
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping
```

新的 HandlerAdapter：

```
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter
```

4.1. 配置新的 HandlerMapping 和 HandlerAdapter

方式一：（推荐）

```
<!-- 配置新的 HandlerMapping 和 HandlerAdapter -->
    <mvc:annotation-driven></mvc:annotation-driven>
```

方式二：

```
<!--  配置新的 HandlerMapping 和 HandlerAdapter-->
    <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"></bean>
    <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"></bean>
```

5. 配置视图解析器

视图解析器，用于解析视图页面，简化视图的编写。

在 spring-mvc.xml 配置：

```
<!-- 视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
        <!-- 前缀访问路径 -->
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <!-- 后缀访问路径 -->
        <property name="suffix" value=".jsp"></property>
    </bean>
```

Controller 代码：

```
@RequestMapping("/view")
    public ModelAndView view(){
        //把数据保存到 ModelAndView（相当于保存 request 域对象）
        ModelAndView mv = new ModelAndView();
        mv.addObject("name", "Spring MVC");

        //返回逻辑视图名称
        mv.setViewName("success");
        return mv;
    }
```

最终的访问路径= 前缀+逻辑视图名称+后缀

6. @RequestMapping 注解的使用

@RequestMapping: 声明 SpringMVC 控制器的类或方法的访问路径。

6.1. value 属性（*）

访问路径，用在类或方法上面

```
@Controller
@RequestMapping(value="/requestMapping")
public class RequestMappingController {


    @RequestMapping(value="/test1")
    public ModelAndView test1(){
        System.out.println("RequestMappingController 的 test1");
        ModelAndView mv = new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
}
```

6.2. method 属性（*）

该控制器的方法支持哪些 HTTP 的请求方式。（GET，POST，PUT，DELETE）

```
/**
     * method 属性
     * @return
     */
```

```
    @RequestMapping(value="/test2",method=RequestMethod.POST)
    public ModelAndView test2(){
        System.out.println("RequestMappingController 的 test2");
        ModelAndView mv = new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
```

6.3. params 属性

声明控制器的方法可以接受什么参数。

```
/**
     * params 属性
     * @return
     */
    @RequestMapping(value="/test3",method=RequestMethod.GET,params={"nam
e"})
    public ModelAndView test3(){
        System.out.println("RequestMappingController 的 test3");
        ModelAndView mv = new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
```

producers 属性： 该控制器的方法返回值的数据类型（xml/json）

consumes 属性：该控制器的方法接受参数的数据类型（xml/json）

headers 属性：该控制器的方法接受包含什么请求头的请求

7. 使用不同方式的跳转页面

7.1. 控制器方法返回 void

相当于 Servlet 的方式的页面跳转

注意：

可以在控制器方法参数里面传入：

HttpServletRequest

HttpServletResponse

HttpSession

```
    /**
     * 7.1.控制器方法返回 void
     * @throws Exception
     * @throws ServletException
     */
    @RequestMapping("/test1")
    public void test1(HttpServletRequest request,HttpServletResponse
response,HttpSession session) throws Exception{
        System.out.println("ViewControlller 的 test1");


        //转发


    //request.getRequestDispatcher("/WEB-INF/jsp/success.jsp").forward(r
equest, response);
        //重定向
        response.sendRedirect(request.getContextPath()+"/index.jsp");
    }
```

7.2. 控制器方法返回 ModelAndView

```
@RequestMapping("/test2")
    public ModelAndView test2(){
        ModelAndView mv = new ModelAndView("success");
        //如果配置了视图解析器，那么该 viewName 就是逻辑名称，否则是物理路径（真实的路径）
        //mv.setViewName("success");
        return mv;
    }
```

7.3. 控制器方法返回 String

```
/**
     * 7.3.控制器方法返回 String （转发）（推荐使用）
     */
    @RequestMapping("/test3")
    public String test3(Model model){
        System.out.println("ViewControlller 的 test3");
        model.addAttribute("name", "Spring MVC");
        //返回逻辑名称
        return "success";
    }

    /**
     * 7.3.控制器方法返回 String （重定向）（推荐使用）
     */
    @RequestMapping("/test4")
    public String test4(){
```

```
        System.out.println("ViewControlller 的 test4");
        return "redirect:/index.jsp";
    }
```