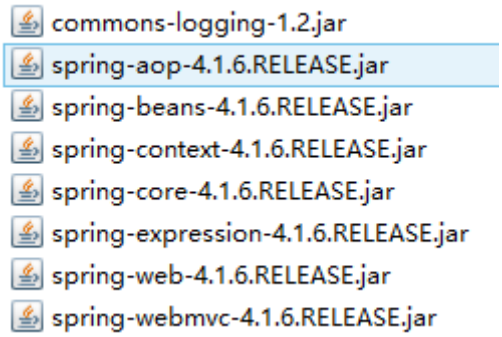
一、SpringMVC 基础入门，创建一个 HelloWorld 程序

1. 首先，导入 SpringMVC 需要的 jar 包。



2. 添加 Web.xml 配置文件中关于 SpringMVC 的配置

```
1
2
3
<!--configure the setting of springmvcDispatcherServlet and configure
4
the mapping-->
5
<servlet>
6
        <servlet-name>springmvc</servlet-name>
7
        <servlet-class>org.springframework.web.servlet.DispatcherSe
8
rvlet</servlet-class>
9
        <init-param>
1
                    <param-name>contextConfigLocation</param-name>
0
                    <param-value>classpath:springmvc-servlet.xml</p
1
aram-value>
1
            </init-param>
1
            <!-- <load-on-startup>1</load-on-startup> -->
2
</servlet>
1
3
<servlet-mapping>
1
        <servlet-name>springmvc</servlet-name>
4
        <url-pattern>/</url-pattern>
1
</servlet-mapping>
5
```

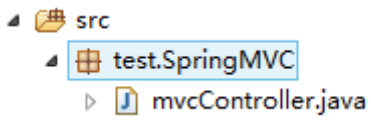
3. 在 src 下添加 springmvc-servlet.xml 配置文件

```
1<?xml version="1.0" encoding="UTF-8"?>
2<beans xmlns="http://www.springframework.org/schema/beans"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xmlns:context="http://www.springframework.org/schema/contex
```

```
5 t"
6         xmlns:mvc="http://www.springframework.org/schema/mvc"
7         xsi:schemaLocation="http://www.springframework.org/schema/b
8 eans http://www.springframework.org/schema/beans/spring-beans.xsd
9                 http://www.springframework.org/schema/context
1 http://www.springframework.org/schema/context/spring-context-4.1.xs
0 d
1                 http://www.springframework.org/schema/mvc
1 http://www.springframework.org/schema/mvc/spring-mvc-4.1.xsd">
1
2
1         <!-- scan the package and the sub package -->
3         <context:component-scan base-package="test.SpringMVC"/>
1
4         <!-- don't handle the static resource -->
1         <mvc:default-servlet-handler />
5
1         <!-- if you use annotation you must configure following setting
6 -->
1         <mvc:annotation-driven />
7
1         <!-- configure the InternalResourceViewResolver -->
8         <bean
1 class="org.springframework.web.servlet.view.InternalResourceViewRes
9 olver"
2                         id="internalResourceViewResolver">
0                 <!-- 前缀 -->
2                 <property name="prefix" value="/WEB-INF/jsp/" />
1                 <!-- 后缀 -->
2                 <property name="suffix" value=".jsp" />
2         </bean>
2 </beans>
3
2
4
2
5
2
6
2
7
```

4.在 WEB-INF 文件夹下创建名为 jsp 的文件夹，用来存放 jsp 视图。创建一个 hello.jsp，在 body 中添加“Hello World”。

5.建立包及 Controller，如下所示



6.编写 Controller 代码

```
1@Controller
2@RequestMapping("/mvc")
3public class mvcController {
4
5        @RequestMapping("/hello")
6        public String hello(){
7                return "hello";
8        }
9}
```

7.启动服务器，键入 http://localhost:8080/项目名/mvc/hello

二、配置解析

1.Dispatcherservlet

DispatcherServlet 是前置控制器，配置在 web.xml 文件中的。拦截匹配的请求，Servlet 拦截匹配规则要自己定义，把拦截下来的请求，依据相应的规则分发到目标 Controller 来处理，是配置 spring MVC 的第一步。

2.InternalResourceViewResolver

视图名称解析器

3.以上出现的注解

@Controller 负责注册一个 bean 到 spring 上下文中

@RequestMapping 注解为控制器指定可以处理哪些 URL 请求

三、SpringMVC 常用注解

@Controller

负责注册一个 bean 到 spring 上下文中

@RequestMapping

注解为控制器指定可以处理哪些 URL 请求

@RequestBody

该注解用于读取 Request 请求的 body 部分数据，使用系统默认配置的 HttpMessageConverter 进行解析，然后把相应的数据绑定 到要返回的对象上 ，再把 HttpMessageConverter 返回的对象数据绑定到 controller 中方法的参数上

@ResponseBody

该注解用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区

@ModelAttribute

在方法定义上使用 @ModelAttribute 注解：Spring MVC 在调用目标处理方法前，会先逐个调用在方法级上标注了@ModelAttribute 的方法

在方法的入参前使用 @ModelAttribute 注解：可以从隐含对象中获取隐含的模型数据中获取对象，再将请求参数 - 绑定到对象中，再传入入参将方法入参对象添加到模型中

@RequestParam

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

@PathVariable

绑定 URL 占位符到入参

@ExceptionHandler

注解到方法上，出现异常时会执行该方法

@ControllerAdvice

使一个 Contoller 成为全局的异常处理类，类中用@ExceptionHandler 方法注解的方法可以处理所有 Controller 发生的异常

四、自动匹配参数

```
1//match automatically
2@RequestMapping("/person")
3public String toPerson(String name,double age){
4        System.out.println(name+" "+age);
5        return "hello";
6}
```

五、自动装箱

1.编写一个 Person 实体类

```
1 package test.SpringMVC.model;
2
3 public class Person {
4         public String getName() {
5                 return name;
6         }
7         public void setName(String name) {
8                 this.name = name;
9         }
10        public int getAge() {
11                return age;
12        }
13        public void setAge(int age) {
14                this.age = age;
15        }
16        private String name;
17        private int age;
18
19}
```

2.在 Controller 里编写方法

```
1//boxing automatically
2@RequestMapping("/person1")
3public String toPerson(Person p){
4        System.out.println(p.getName()+" "+p.getAge());
5        return "hello";
6}
```

六、使用 InitBinder 来处理 Date 类型的参数

```
1 //the parameter was converted in initBinder
```

```
2 @RequestMapping("/date")
3 public String date(Date date){
4         System.out.println(date);
5         return "hello";
6 }
7
8 //At the time of initialization,convert the type "String" to type "date"
9 @InitBinder
10public void initBinder(ServletRequestDataBinder binder){
11        binder.registerCustomEditor(Date.class, new
12CustomDateEditor(new SimpleDateFormat("yyyy-MM-dd"),
13                        true));
  }
```

七、向前台传递参数

```
1//pass the parameters to front-end
2@RequestMapping("/show")
3public String showPerson(Map<String,Object> map){
4        Person p =new Person();
5        map.put("p", p);
6        p.setAge(20);
7        p.setName("jayjay");
8        return "show";
9}
```

前台可在 Request 域中取到"p"

八、使用 Ajax 调用

```
1//pass the parameters to front-end using ajax
2@RequestMapping("/getPerson")
3public void getPerson(String name,PrintWriter pw){
4        pw.write("hello,"+name);
5}
6@RequestMapping("/name")
7public String sayHello(){
8        return "name";
9}
```

前台用下面的 Jquery 代码调用

```
1$(function(){
```

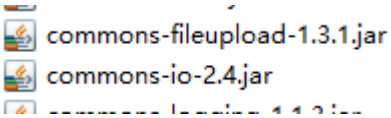
```
2        $("#btn").click(function(){
3              $.post("mvc/getPerson",{name:$("#name").val()},functio
4n(data){
5                        alert(data);
6                 });
7        });
 });
```

九、在 Controller 中使用 redirect 方式处理请求

```
1//redirect
2@RequestMapping("/redirect")
3public String redirect(){
4        return "redirect:hello";
5}
```

十、文件上传

1.需要导入两个 jar 包



2.在 SpringMVC 配置文件中加入

```
 <!-- upload settings -->
1<bean
2id="multipartResolver"   class="org.springframework.web.multipart.co
3mmons.CommonsMultipartResolver">
4        <property name="maxUploadSize" value="102400000"></property>
 </bean>
```

3.方法代码

```
1 @RequestMapping(value="/upload",method=RequestMethod.POST)
2 public String upload(HttpServletRequest req) throws Exception{
3         MultipartHttpServletRequest mreq =
4 (MultipartHttpServletRequest)req;
5         MultipartFile file = mreq.getFile("file");
6         String fileName = file.getOriginalFilename();
7         SimpleDateFormat sdf = new
8 SimpleDateFormat("yyyyMMddHHmmss");
```

```
9         FileOutputStream fos = new
1 FileOutputStream(req.getSession().getServletContext().getRealPath("
0 /")+
1                         "upload/"+sdf.format(new
1 Date())+fileName.substring(fileName.lastIndexOf('.')));
1         fos.write(file.getBytes());
2         fos.flush();
1         fos.close();
3
1         return "hello";
4 }
```

4.前台 form 表单

```
1 <form action="mvc/upload" method="post"
2 enctype="multipart/form-data">
3         <input type="file" name="file"><br>
4         <input type="submit" value="submit">
  </form>
```

十一、使用@RequestParam 注解指定参数的 name

```
1  @Controller
2  @RequestMapping("/test")
3  public class mvcController1 {
4         @RequestMapping(value="/param")
5         public String testRequestParam(@RequestParam(value="id")
6  Integer id,
7                         @RequestParam(value="name")String name){
8                 System.out.println(id+" "+name);
9                 return "/hello";
10        }
  }
```

十二、RESTFul 风格的 SringMVC

1.RestController

```
1 @Controller
2 @RequestMapping("/rest")
3 public class RestController {
4         @RequestMapping(value="/user/{id}",method=RequestMethod.GET
5 )
```

```
6         public String get(@PathVariable("id") Integer id){
7                 System.out.println("get"+id);
8                 return "/hello";
9         }
1
0         @RequestMapping(value="/user/{id}",method=RequestMethod.POS
1 T)
1         public String post(@PathVariable("id") Integer id){
1                 System.out.println("post"+id);
2                 return "/hello";
1         }
3
1         @RequestMapping(value="/user/{id}",method=RequestMethod.PUT
4 )
1         public String put(@PathVariable("id") Integer id){
5                 System.out.println("put"+id);
1                 return "/hello";
6         }
1
7         @RequestMapping(value="/user/{id}",method=RequestMethod.DEL
1 ETE)
8         public String delete(@PathVariable("id") Integer id){
1                 System.out.println("delete"+id);
9                 return "/hello";
2         }
0
2 }
1
2
2
2
3
2
4
2
5
2
6
2
7
2
8
```

2.form 表单发送 put 和 delete 请求

在 web.xml 中配置

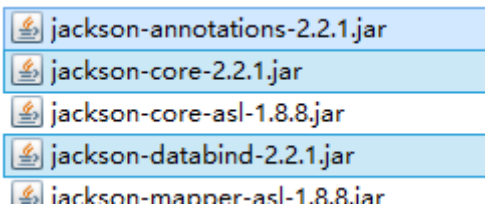
```
<!-- configure the HiddenHttpMethodFilter,convert the post method to put
lor delete -->
2<filter>
3        <filter-name>HiddenHttpMethodFilter</filter-name>
4        <filter-class>org.springframework.web.filter.HiddenHttpMetho
5dFilter</filter-class>
6</filter>
7<filter-mapping>
8        <filter-name>HiddenHttpMethodFilter</filter-name>
9        <url-pattern>/*</url-pattern>
</filter-mapping>
```

在前台可以用以下代码产生请求

```
1 <form action="rest/user/1" method="post">
2         <input type="hidden" name="_method" value="PUT">
3         <input type="submit" value="put">
4 </form>
5
6 <form action="rest/user/1" method="post">
7         <input type="submit" value="post">
8 </form>
9
10<form action="rest/user/1" method="get">
11        <input type="submit" value="get">
12</form>
13
14<form action="rest/user/1" method="post">
15        <input type="hidden" name="_method" value="DELETE">
16        <input type="submit" value="delete">
17</form>
```

十三、返回 json 格式的字符串

1.导入以下 jar 包



2.方法代码

```
@Controller
@RequestMapping("/json")
public class jsonController {

        @ResponseBody
        @RequestMapping("/user")
        public   User get(){
                User u = new User();
                u.setId(1);
                u.setName("jaylay");
                u.setBirth(new Date());
                return u;
        }
}
```

十四、异常的处理

1.处理局部异常（Controller 内）

```
@ExceptionHandler
public ModelAndView exceptionHandler(Exception ex){
        ModelAndView mv = new ModelAndView("error");
        mv.addObject("exception", ex);
        System.out.println("in testExceptionHandler");
        return mv;
}

@RequestMapping("/error")
public String error(){
        int i = 5/0;
        return "hello";
}
```

2.处理全局异常（所有 Controller）

```
@ControllerAdvice
public class testControllerAdvice {
        @ExceptionHandler
        public ModelAndView exceptionHandler(Exception ex){
                ModelAndView mv = new ModelAndView("error");
                mv.addObject("exception", ex);
                System.out.println("in testControllerAdvice");
```

```
8                   return mv;
9           }
10}
```

3.另一种处理全局异常的方法

在 SpringMVC 配置文件中配置

```
 <!-- configure SimpleMappingExceptionResolver -->
 <bean
1 class="org.springframework.web.servlet.handler.SimpleMappingExceptio
2 nResolver">
3           <property name="exceptionMappings">
4                   <props>
5                           <prop
6 key="java.lang.ArithmeticException">error</prop>
7                   </props>
8           </property>
 </bean>
```

error 是出错页面

十五、设置一个自定义拦截器

1.创建一个 MyInterceptor 类，并实现 HandlerInterceptor 接口

```
1 public class MyInterceptor implements HandlerInterceptor {
2
3           @Override
4           public void afterCompletion(HttpServletRequest arg0,
5                           HttpServletResponse arg1, Object arg2,
6 Exception arg3)
7                           throws Exception {
8                   System.out.println("afterCompletion");
9           }
10
11          @Override
12          public void postHandle(HttpServletRequest arg0,
13HttpServletResponse arg1,
14                          Object arg2, ModelAndView arg3) throws
15Exception {
16                  System.out.println("postHandle");
17          }
18
```

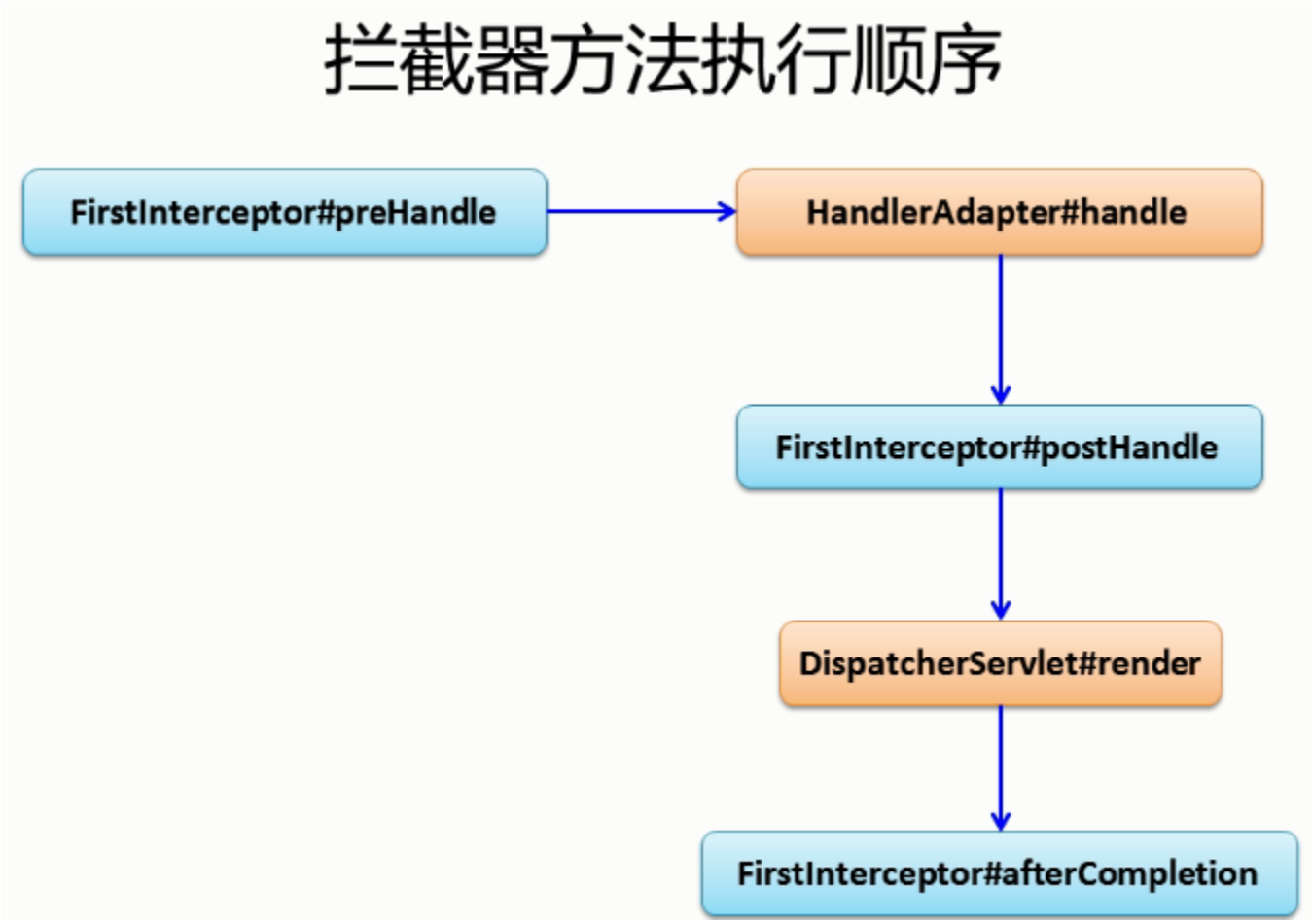
```
19        @Override
20        public boolean preHandle(HttpServletRequest arg0,
21HttpServletResponse arg1,
22                        Object arg2) throws Exception {
23                System.out.println("preHandle");
                  return true;
          }

  }
```

2.在 SpringMVC 的配置文件中配置

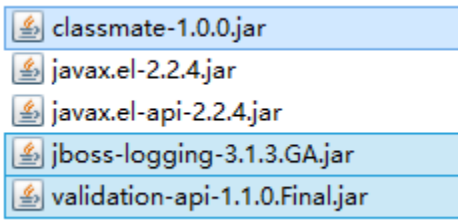
```
1<!-- interceptor setting -->
2<mvc:interceptors>
3        <mvc:interceptor>
4                <mvc:mapping path="/mvc/**"/>
5                <bean
6class="test.SpringMVC.Interceptor.MyInterceptor"></bean>
7        </mvc:interceptor>
</mvc:interceptors>
```

3.拦截器执行顺序

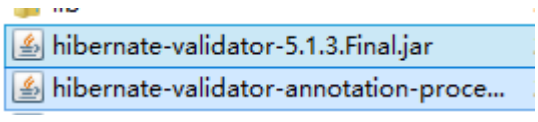


十六、表单的验证（使用 Hibernate-validate）及国际化

1. 导入 Hibernate-validate 需要的 jar 包



（未选中不用导入）



2.编写实体类 User 并加上验证注解

```
public class User {
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Date getBirth() {
                return birth;
        }
        public void setBirth(Date birth) {
                this.birth = birth;
        }
        @Override
        public String toString() {
                return "User [id=" + id + ", name=" + name + ", birth="
+ birth + "]";
        }
        private int id;
        @NotEmpty
        private String name;

        @Past
        @DateTimeFormat(pattern="yyyy-MM-dd")
        private Date birth;
}
```

ps:@Past 表示时间必须是一个过去值

3.在 jsp 中使用 SpringMVC 的 form 表单

```
<form:form action="form/add" method="post" modelAttribute="user">
        id:<form:input path="id"/><form:errors path="id"/><br>
        name:<form:input path="name"/><form:errors path="name"/><br>
        birth:<form:input path="birth"/><form:errors path="birth"/>
        <input type="submit" value="submit">
</form:form>
```

ps:path 对应 name

4.Controller 中代码

```
@Controller
@RequestMapping("/form")
public class formController {
        @RequestMapping(value="/add",method=RequestMethod.POST)

        public String add(@Valid User u,BindingResult br){
                if(br.getErrorCount()>0){
                        return "addUser";
                }
                return "showUser";
        }

        @RequestMapping(value="/add",method=RequestMethod.GET)
        public String add(Map<String,Object> map){
                map.put("user",new User());
                return "addUser";
        }
}
```

ps:

1.因为 jsp 中使用了 modelAttribute 属性，所以必须在 request 域中有一个"user".

2.@Valid 表示按照在实体上标记的注解验证参数

3.返回到原页面错误信息回回显，表单也会回显

5.错误信息自定义

在 src 目录下添加 locale.properties

```
NotEmpty.user.name=name can't not be empty
Past.user.birth=birth should be a past value
```

```
DateTimeFormat.user.birth=the format of input is wrong
typeMismatch.user.birth=the format of input is wrong
typeMismatch.user.id=the format of input is wrong
```

在 SpringMVC 配置文件中配置

```
 <!-- configure the locale resource -->
1<bean id="messageSource"
2class="org.springframework.context.support.ResourceBundleMessageSour
3ce">
4        <property name="basename" value="locale"></property>
 </bean>
```

6.国际化显示

在 src 下添加 locale_zh_CN.properties

```
username=账号
password=密码
```

locale.properties 中添加

```
username=user name
password=password
```

创建一个 locale.jsp

```
1<body>
2    <fmt:message key="username"></fmt:message>
3    <fmt:message key="password"></fmt:message>
4</body>
```

在 SpringMVC 中配置

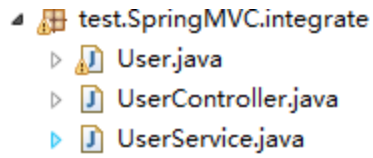
```
1<!-- make the jsp page can be visited -->
2<mvc:view-controller path="/locale" view-name="locale"/>
```

让 locale.jsp 在 WEB-INF 下也能直接访问

最后，访问 locale.jsp，切换浏览器语言，能看到账号和密码的语言也切换了

十七、压轴大戏--整合 SpringIOC 和 SpringMVC

1.创建一个 test.SpringMVC.integrate 的包用来演示整合，并创建各类



2.User 实体类

```
public class User {
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Date getBirth() {
                return birth;
        }
        public void setBirth(Date birth) {
                this.birth = birth;
        }
        @Override
        public String toString() {
                return "User [id=" + id + ", name=" + name + ", birth="
+ birth + "]";
        }
        private int id;
        @NotEmpty
        private String name;

        @Past
        @DateTimeFormat(pattern="yyyy-MM-dd")
        private Date birth;
}
```

3.UserService 类

```
@Component
public class UserService {
        public UserService(){
```

```
4                 System.out.println("UserService
5 Constructor...\n\n\n\n\n\n");
6         }
7
8         public void save(){
9                 System.out.println("save");
10        }
  }
```

4.UserController

```
1 @Controller
2 @RequestMapping("/integrate")
3 public class UserController {
4         @Autowired
5         private UserService userService;
6
7         @RequestMapping("/user")
8         public String saveUser(@RequestBody @ModelAttribute User u){
9                 System.out.println(u);
10                userService.save();
11                return "hello";
12        }
13}
```

5.Spring 配置文件

在 src 目录下创建 SpringIOC 的配置文件 applicationContext.xml

```
1<?xml version="1.0" encoding="UTF-8"?>
2<beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://www.springframework.org/schema/b
5eans
6                 http://www.springframework.org/schema/beans/spring-
7beans.xsd
8                 http://www.springframework.org/schema/util
9                 http://www.springframework.org/schema/util/spring-u
1til-4.0.xsd
0                 http://www.springframework.org/schema/context
1                 http://www.springframework.org/schema/context/sprin
1g-context.xsd
1                 "
2                 xmlns:util="http://www.springframework.org/schema/u
1til"
```

```
3                 xmlns:p="http://www.springframework.org/schema/p"
1                 xmlns:context="http://www.springframework.org/schem
4a/context"
1                 >
5         <context:component-scan
1base-package="test.SpringMVC.integrate">
6                 <context:exclude-filter type="annotation"
1                         expression="org.springframework.stereotype.
7Controller"/>
1                 <context:exclude-filter type="annotation"
8                         expression="org.springframework.web.bind.an
1notation.ControllerAdvice"/>
9         </context:component-scan>
2
0</beans>
2
1
2
2
```

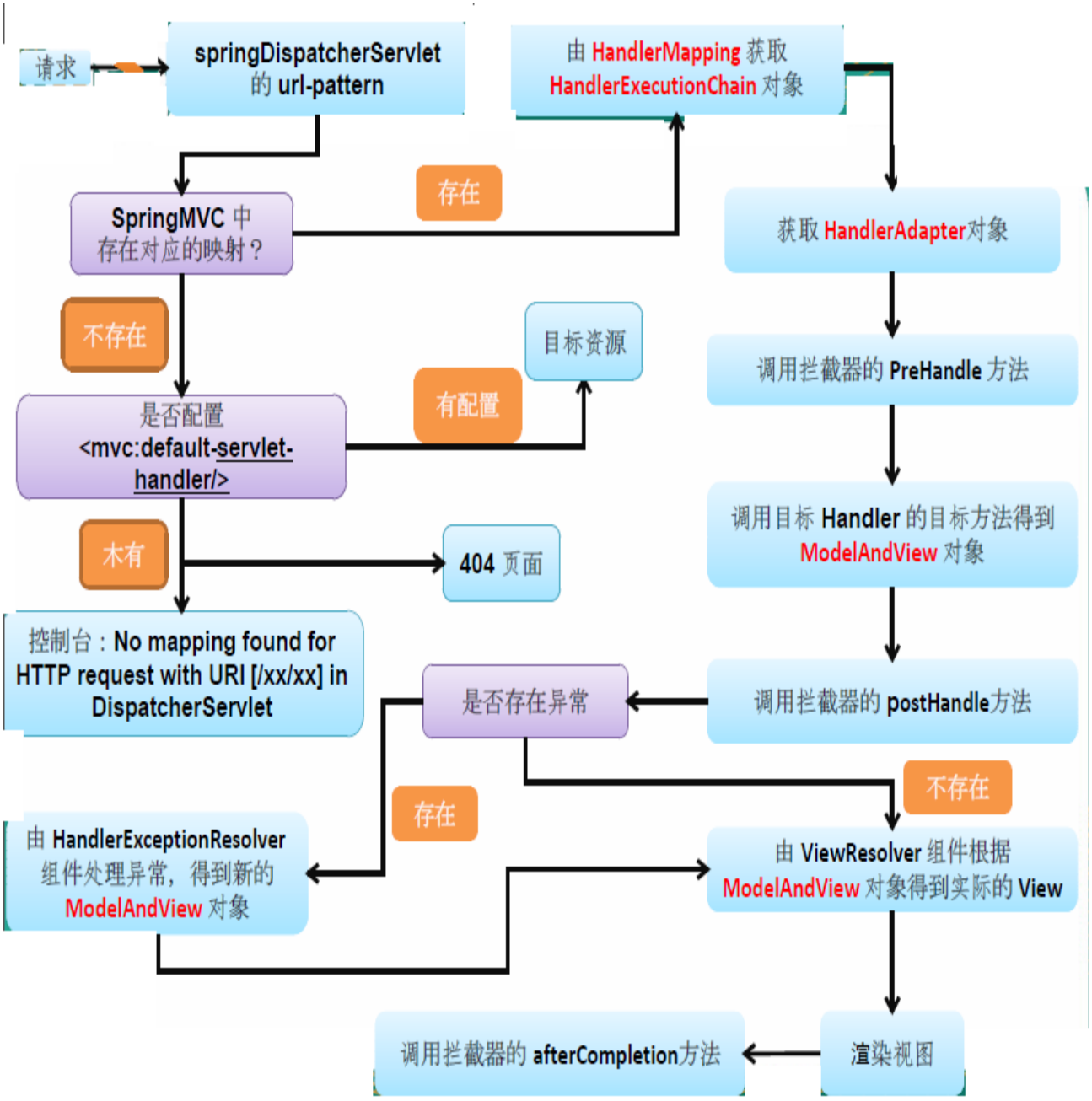
在 Web.xml 中添加配置

```
<!-- configure the springIOC -->
1<listener>
2        <listener-class>org.springframework.web.context.ContextLoade
3rListener</listener-class>
4</listener>
5<context-param>
6    <param-name>contextConfigLocation</param-name>
7    <param-value>classpath:applicationContext.xml</param-value>
8</context-param>
```

6.在 SpringMVC 中进行一些配置，防止 SpringMVC 和 SpringIOC 对同一个对象的管理重合

```
<!-- scan the package and the sub package -->
        <context:component-scan
1base-package="test.SpringMVC.integrate">
2                <context:include-filter type="annotation"
3                        expression="org.springframework.stereotype.C
4ontroller"/>
5                <context:include-filter type="annotation"
6                        expression="org.springframework.web.bind.ann
7otation.ControllerAdvice"/>
        </context:component-scan>
```

十八、SpringMVC 详细运行流程图



十九、SpringMVC 与 struts2 的区别

1、springmvc 基于方法开发的，struts2 基于类开发的。springmvc 将 url 和 controller 里的方法映射。映射成功后 springmvc 生成一个 Handler 对象，对象中只包括了一个 method。方法执行结束，形参数据销毁。springmvc 的 controller 开发类似 web service 开发。

2、springmvc 可以进行单例开发，并且建议使用单例开发，struts2 通过类的成员变量接收参数，无法使用单例，只能使用多例。

3、经过实际测试，struts2 速度慢，在于使用 struts 标签，如果使用 struts 建议使用 jstl。