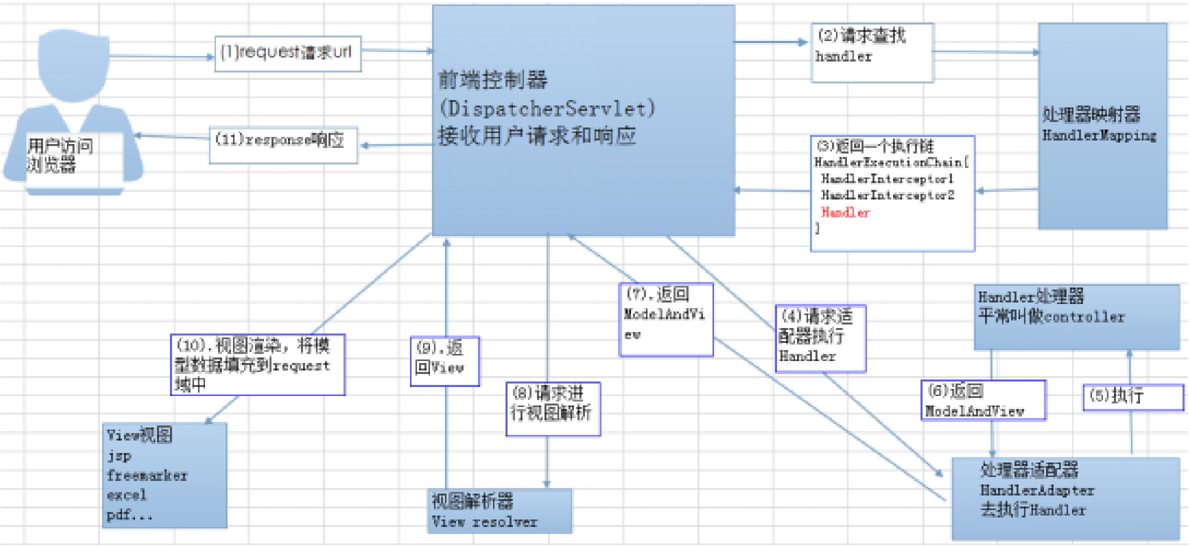
SpringMVC框架

1. SpringMVC概述

执行机制



1.1. API对象

1）DispatcherServlet: 前端控制器：写好的类需要【配置】

2）HandlerMapping: 处理器映射器,作用是根据url查找对应的处理器（Handler）写好的【配置】,返回HandlerExecutionChain。

3）HandlerExecutionChain 处理器的执行连（包含拦截器）,系统写好的API。（不需要编程关注）

4）Interceptor: 拦截器（spring的拦截器类似Filter，与FIlter有差异）,需要【自定义】，非必须

5）Handler处理器：【自定义】的Controller代码（替换Servlet的类）

6）HandlerAdapter： 处理器适配器，用于执行具体的Controller的某一个方法，返回ModeAndView。处理器适配器不需要自定义，系统已经实现了几个只需【配置】即可。

7）ModeAndView：负责管理视图和数据，直接在Controller的方法中直接使用即可。

8）ViewResolver:视图解析器只需【配置】使用即可。

9）View： 视图的对象表示 JstlView ....暂时不需要特殊的关注

1.2. 在开发过程中需要配置

1）DispatcherServlet(前端控制器，核心)：具体的类直接能够使用

2）HandlerMapping(处理器映射器)： 多个实现方案，有默认值。

3）HandlerAdapter（处理器适配器）: 多个实现方案，有默认值。

4）ViewResolver（视图解析器）： 有具体的实现多个,有默认

1.3. 在开发过程中需要自定义（自己写实现过程）

1）Interceptor(拦截器)：需要自己实现，非必须

2）Handler处理器(常常称之为Controller)：具体Controller（UserController、AccountController...等价于Servlet），必须存在。

2. 入门程序

Groupid：com.neuedu

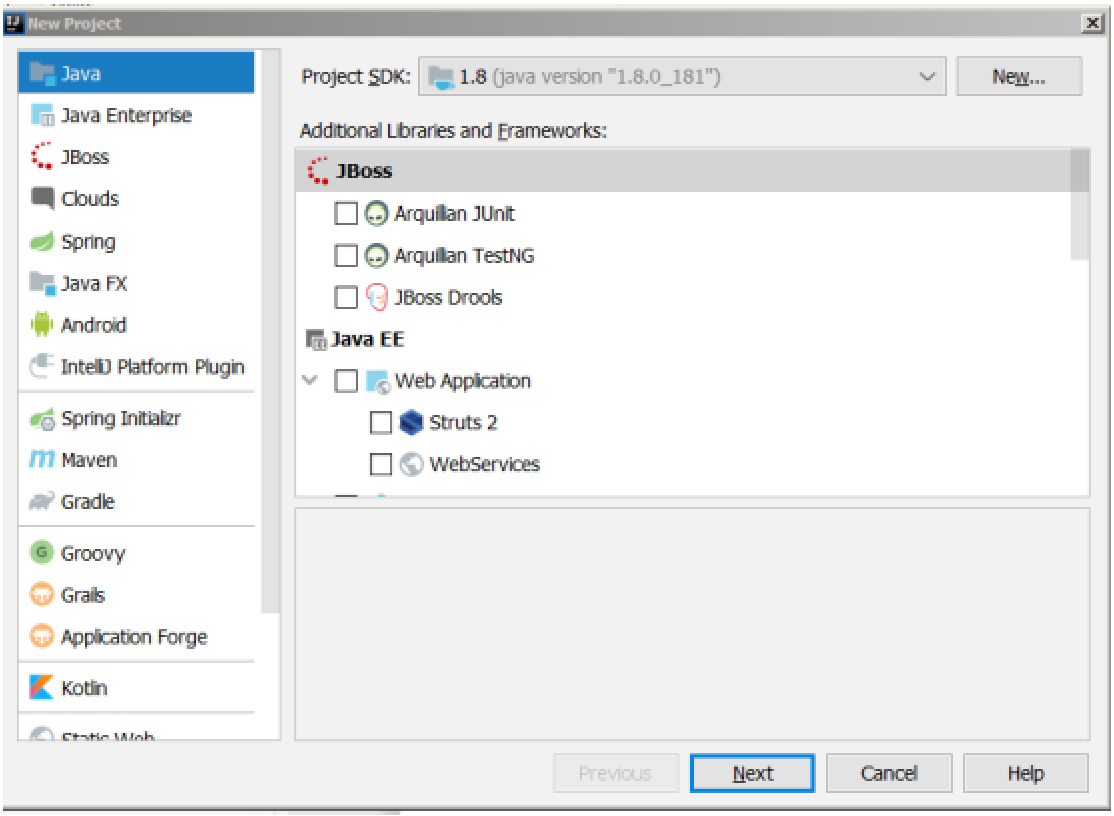
聚合项目（maven）

Spring-mvc-java1（quickstart骨架）

| springmvc-01-helloword (webapp骨架)

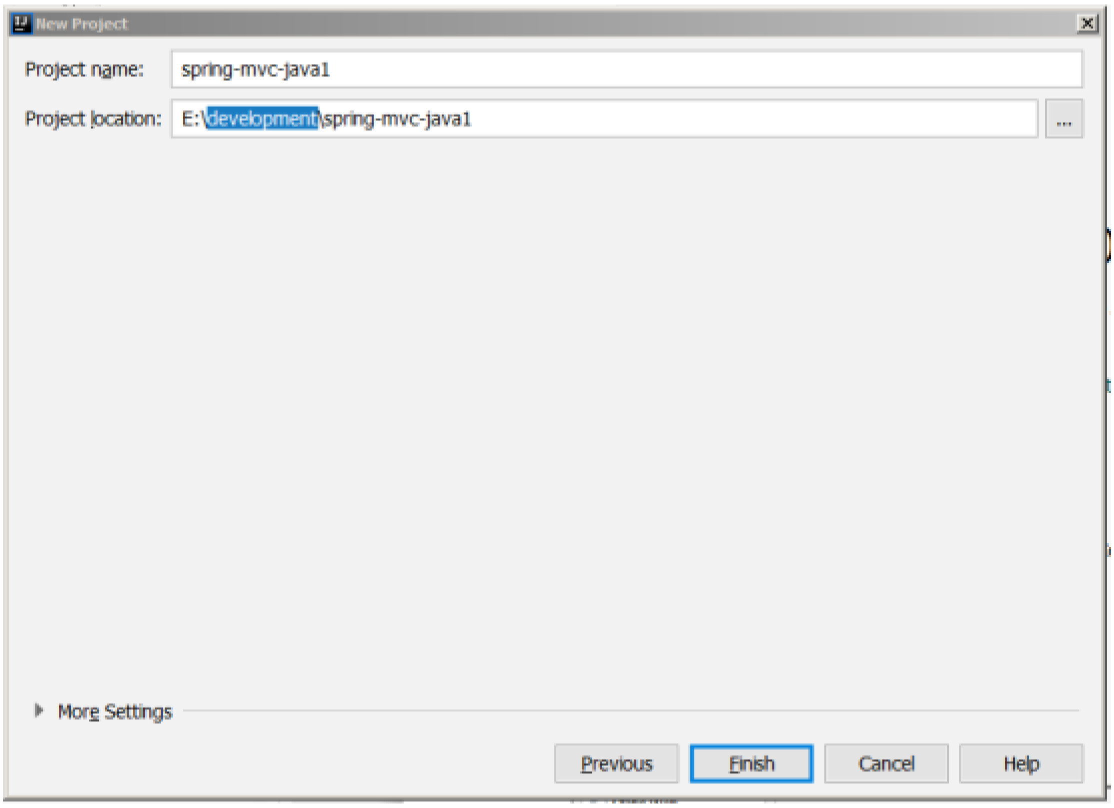
| springmvc-02-config.......... (webapp)

2.1. 创建普通的聚合工程（跟maven没有关系）



Project SDK: 1.8 (java version "1.8.0_181")

Additional Libraries and Frameworks:



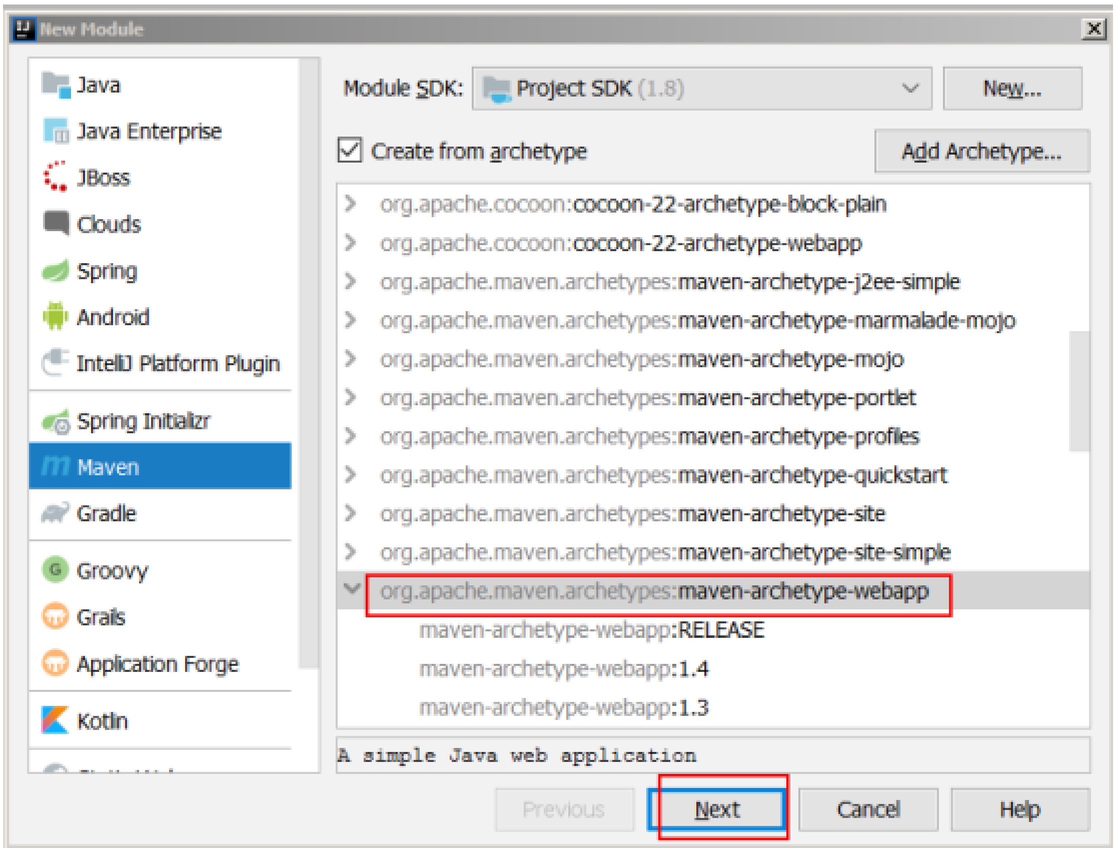
New Project

Project name: spring-mvc-java1

Project location: E:\development\spring-mvc-java1

2.2. Webapp的骨架(01-helloword)

项目的id： springmvc-01-helloword



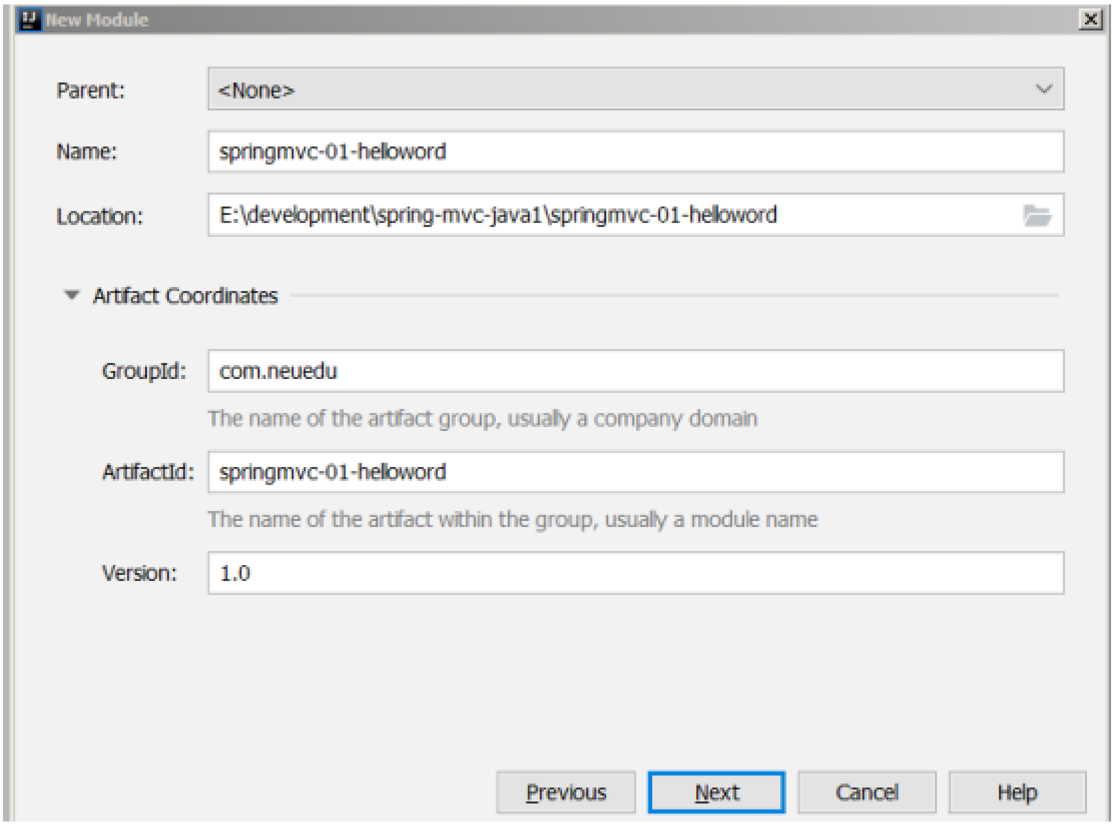
New Module

Module SDK: Project SDK (1.8)

☑ Create from archetype

- org.apache.cocoon:cocoon-22-archetype-block-plain
- org.apache.cocoon:cocoon-22-archetype-webapp
- org.apache.maven.archetypes:maven-archetype-j2ee-simple
- org.apache.maven.archetypes:maven-archetype-marmalade-mojo
- org.apache.maven.archetypes:maven-archetype-mojo
- org.apache.maven.archetypes:maven-archetype-portlet
- org.apache.maven.archetypes:maven-archetype-profiles
- org.apache.maven.archetypes:maven-archetype-quickstart
- org.apache.maven.archetypes:maven-archetype-site
- org.apache.maven.archetypes:maven-archetype-site-simple
- org.apache.maven.archetypes:maven-archetype-webapp
  - maven-archetype-webapp:RELEASE
  - maven-archetype-webapp:1.4
  - maven-archetype-webapp:1.3

```
A simple Java web application
```



Parent: <None>

Name: springmvc-01-helloword

Location: E:\development\spring-mvc-java1\springmvc-01-helloword

Artifact Coordinates

GroupId: com.neuedu

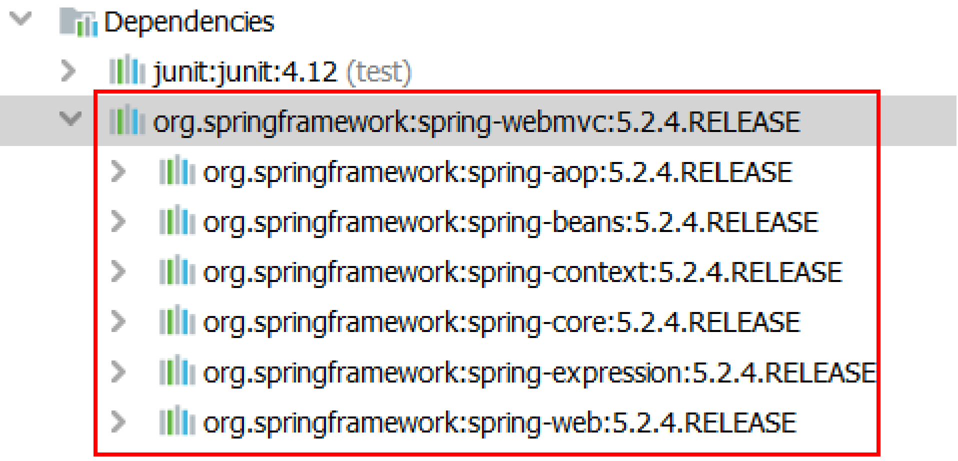
The name of the artifact group, usually a company domain

ArtifactId: springmvc-01-helloword

The name of the artifact within the group, usually a module name

Version: 1.0

2.3. 添加pom依赖



```
Dependencies
  junit:junit:4.12 (test)
  org.springframework:spring-webmvc:5.2.4.RELEASE
    org.springframework:spring-aop:5.2.4.RELEASE
    org.springframework:spring-beans:5.2.4.RELEASE
    org.springframework:spring-context:5.2.4.RELEASE
    org.springframework:spring-core:5.2.4.RELEASE
    org.springframework:spring-expression:5.2.4.RELEASE
    org.springframework:spring-web:5.2.4.RELEASE
```

```xml
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>


<dependency>
```

```
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.2.4.RELEASE</version>
</dependency>
```

2.4. 配置前端控制器

2.4.1. springmvc的配置文件

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

</beans>
```

2.4.2. Web.xml

声明 DispatcherServlet， 并指定spring的配置文件

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <!--前端控制器-->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>
  <!--默认的配置文件的名字applicationContext.xml-->
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:springmvc.xml</param-value>
</init-param>


  </servlet>

  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>


</web-app>
```

2.5. 配置处理器映射器、处理器适配器、视图解析器

有默认值，可以不配置(入门程序不配)

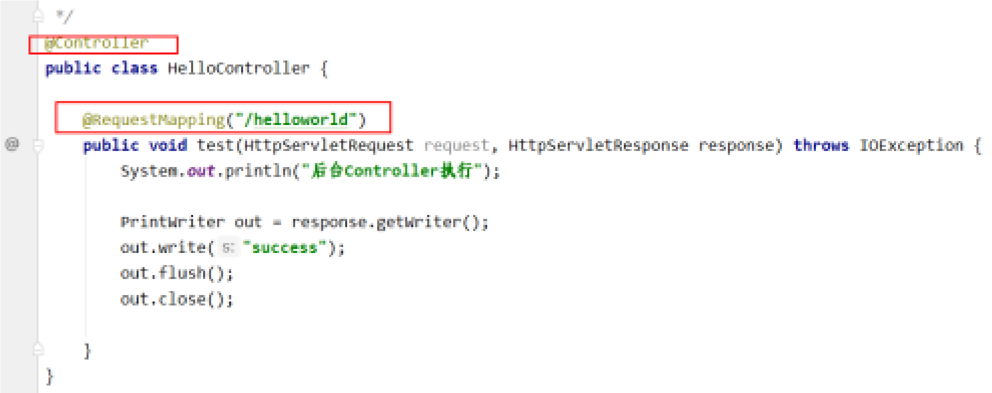
2.6. 自定义处理器(TestController)

定义一个普通类，有一个方法接受（HttpServletRequest、HTTPResponse）

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/25  14:53 25
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 测试控制器
 */
@Controller
public class HelloController {

//     --访问test方法:    http://localhost:8080/springmvc/helloworld
    @RequestMapping("/helloworld")
    public void test(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("后台Controller执行");

        PrintWriter out = response.getWriter();
        out.write("success");
        out.flush();
        out.close();

    }
}
```

2.7. 在类上配置@Controller

2.8. 在方法上配置@RequestMapping

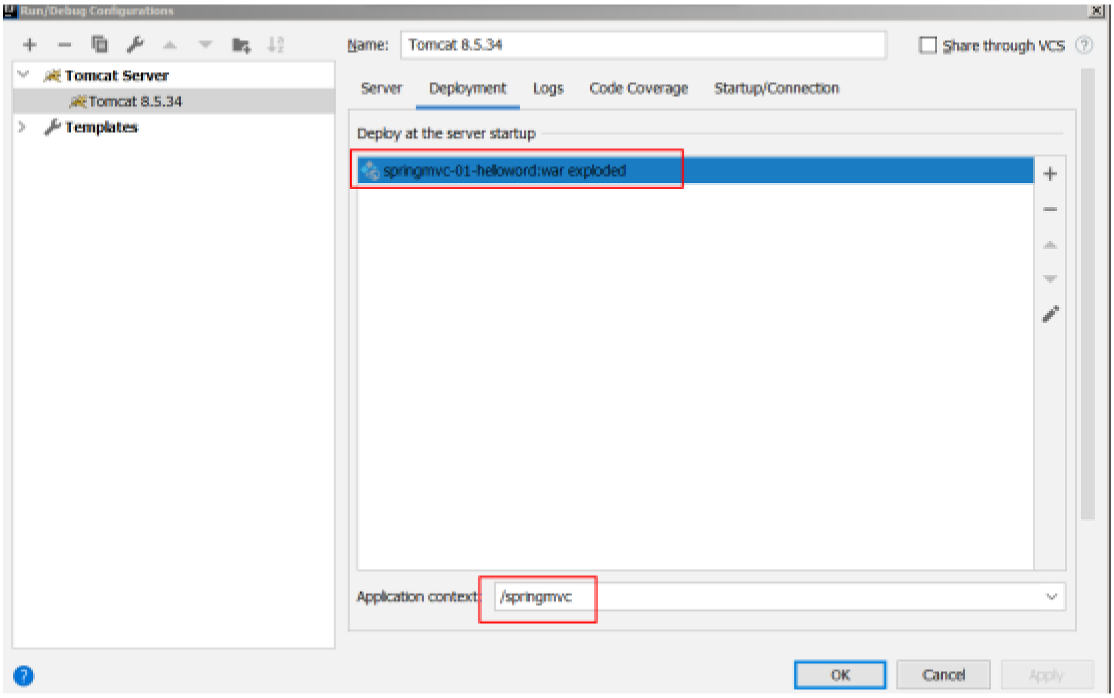


```
*/
@Controller
public class HelloController {

    @RequestMapping("/helloworld")
    public void test(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("后台Controller执行");

        PrintWriter out = response.getWriter();
        out.write( s: "success");
        out.flush();
        out.close();

    }
}
```

2.9. 在spirng-mvc.Xml中扫描controller包

```
<!--配置扫描组件-->
<context:component-scan base-package="com.neuedu.controler"/>
```

2.10. 发布tomcat进行测试



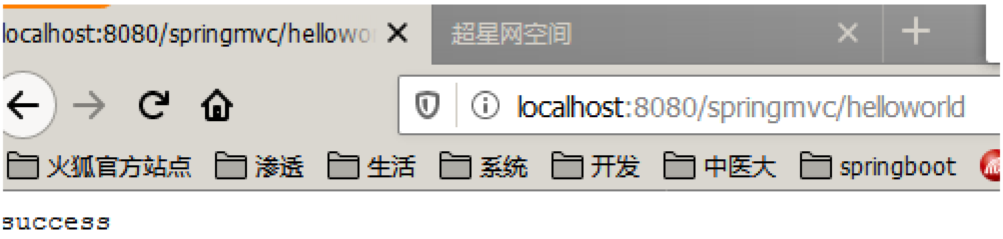
Name: Tomcat 8.5.34

Deploy at the server startup

springmvc-01-helloword:war exploded

Application context: /springmvc

测试路径： http://localhost:8080/springmvc /helloworld



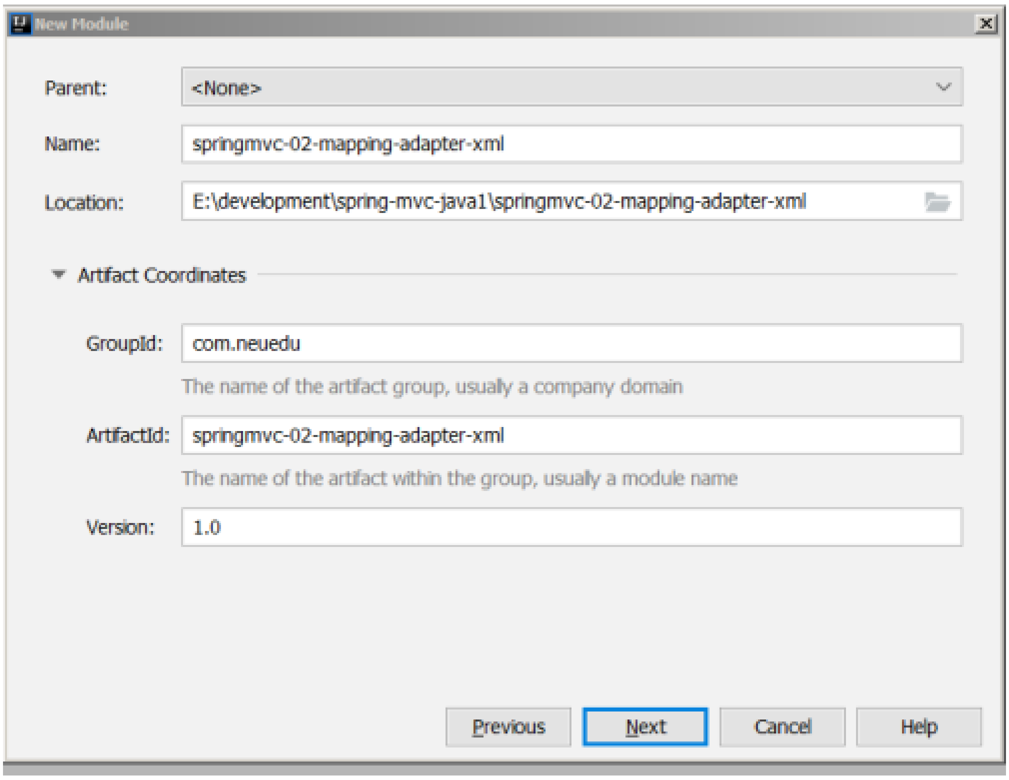
localhost:8080/springmvc/helloworld

```
success
```

3. 配置文件开发(处理器映射器、处理器适配器)

3.1. Web-app骨架的项目

Springmvc-02-mapping-adapter-xml



New Module

Parent: <None>

Name: springmvc-02-mapping-adapter-xml

Location: E:\development\spring-mvc-java1\springmvc-02-mapping-adapter-xml

Artifact Coordinates

GroupId: com.neuedu

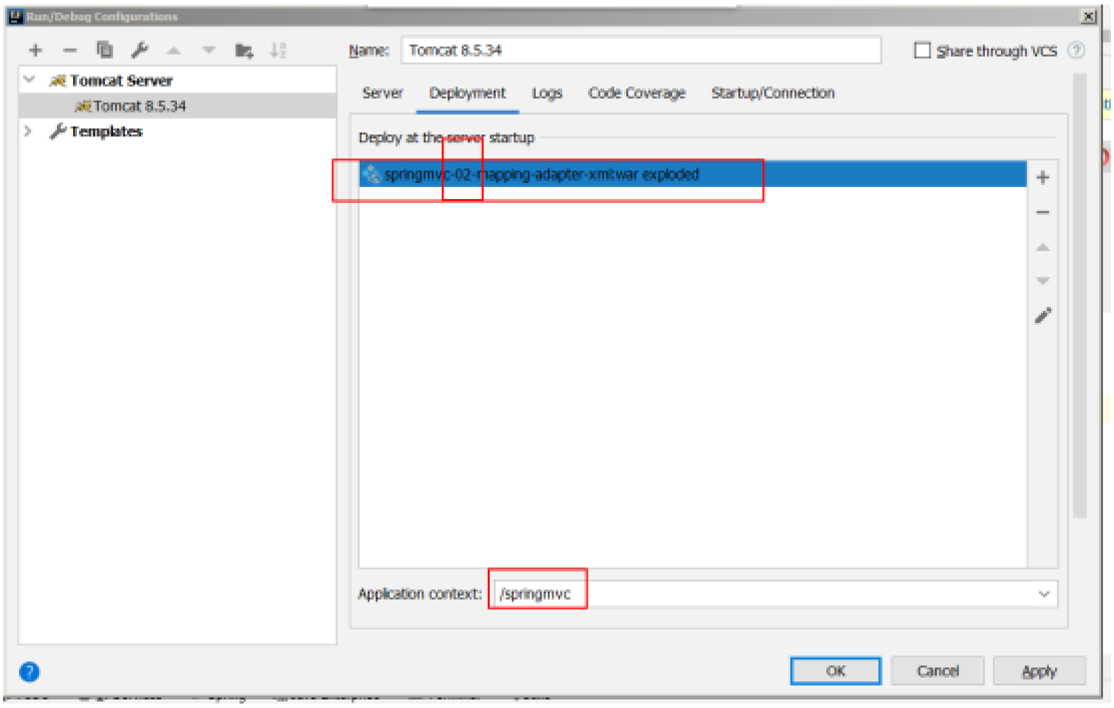
The name of the artifact group, usually a company domain

ArtifactId: springmvc-02-mapping-adapter-xml

The name of the artifact within the group, usually a module name

Version: 1.0

3.2. 重复入门程序搭建了一个helloworld



Run/Debug Configurations

Name: Tomcat 8.5.34

□ Share through VCS

Deploy at the server startup

springmvc-02-mapping-adapter-xml:war exploded

Application context: /springmvc

3.3. 测试Controller

http://127.0.0.1:8080/springmvc/helloworld

3.4. Xml形式配置处理器映射器、处理器适配器

非注解(XML)的形式配置处理器映射器、处理器适配器

3.4.1. Controller

```java
import org.springframework.web.HttpRequestHandler;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  9:23 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : BeanNameUrlHandlerMapping
 */
public class BeanNameUrlController implements HttpRequestHandler {


    @Override
    public void handleRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("后台BeanNameUrlController执行");

        PrintWriter out = response.getWriter();
        out.write("beanNameurlController");
        out.flush();
        out.close();
    }
}
```

3.4.2. 处理器映射器

【映射器】(通过什么样的形式将url和类关联起来)

3.4.2.1. BeanNameUrlHandlerMapping**

通过Bean的name和url进行匹配

```xml
  <!--使用xml的形式配置bean
-->
  <bean name="/beanname_url.action" 
class="com.neuedu.controller.BeanNameUrlController"/>

  <!--处理器映射器 -->
  <!-- 1 BeanNameUrlHandlerMapping
   作用是查找是否存在 bean的name属性 跟url一致即可找到类(Controller)
   http://localhost:8080/springmv/【beanname_url.action】
   http://localhost:8080/springmv/beanname_url.action
  -->
  <bean 
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

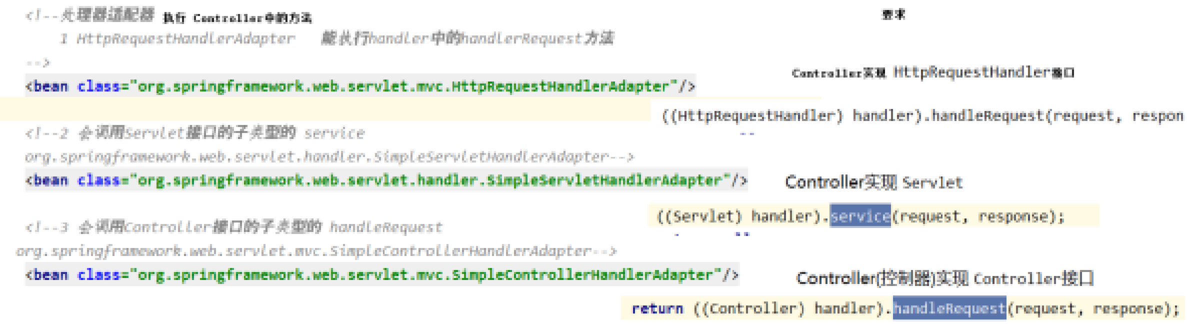
3.4.2.2. SimpleUrlHandlerMapping

```xml
<!--2  org.springframework.web.servlet.handler.SimpleUrlHandlerMapping
    可以配置属性，对应关系的属性 将url和不同的 bean对象关联起来
-->
<bean id="userController" class="com.neuedu.controller.UserController"/>
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
           <!-- http://localhost:8080/springmvc/userquery.action-->
            <prop key="/userquery.action">userController</prop>

            <!-- http://localhost:8080/springmvc/userquery2.action-->
            <prop key="/userquery2.action">userController</prop>
            <prop key="/userquery3.action">userController</prop>
            <prop key="/userquery4.action">userController</prop>
        </props>

    </property>

</bean>
```

3.4.3. 处理器适配器

处理器【适配器】（找到Controller如何执行类中的方法、执行哪个方法）

HandlerAdapter子类型



```xml
<!--处理器适配器 执行 Controller中的方法
    1 HttpRequestHandlerAdapter    能执行handler中的handlerRequest方法
-->
<bean class="org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"/>

<!--2 会调用Servlet接口的子类型的 service
org.springframework.web.servlet.handler.SimpleServletHandlerAdapter-->
<bean class="org.springframework.web.servlet.handler.SimpleServletHandlerAdapter"/>

<!--3 会调用Controller接口的子类型的 handleRequest
org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter-->
<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>
```

要求

Controller实现 HttpRequestHandler接口

```java
((HttpRequestHandler) handler).handleRequest(request, respon
```

Controller实现 Servlet

```java
((Servlet) handler).service(request, response);
```

Controller(控制器)实现 Controller接口

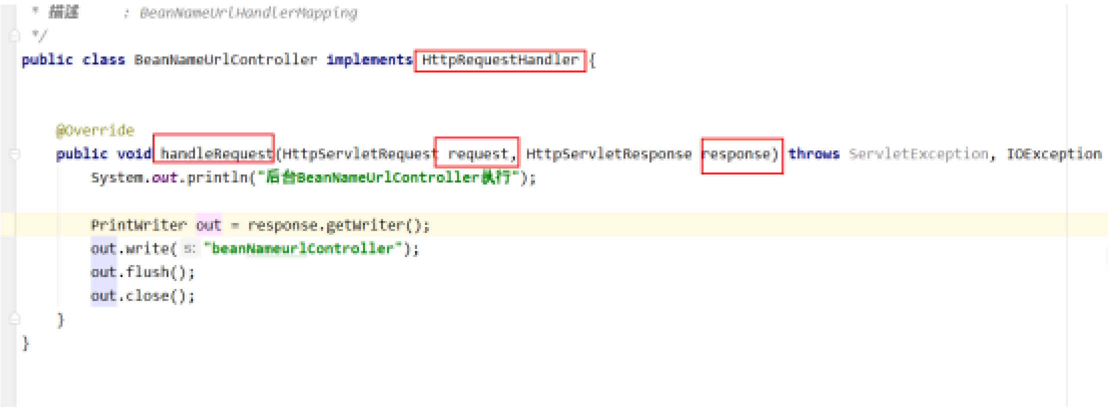
```java
return ((Controller) handler).handleRequest(request, response);
```

3.4.3.1. HttpRequestHandlerAdapter

处理器【适配器】通过适配器调用对应的Handler方法,handleRequest方法,要求类必须实现HTTPRequestHandler接口，并实现上述的抽象方法(handleRequest)

```
<!--处理器适配器
    1 HttpRequestHandlerAdapter   能执行handler中的handlerRequest方法
-->
<bean class="org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"/>
```

执行Controller中的



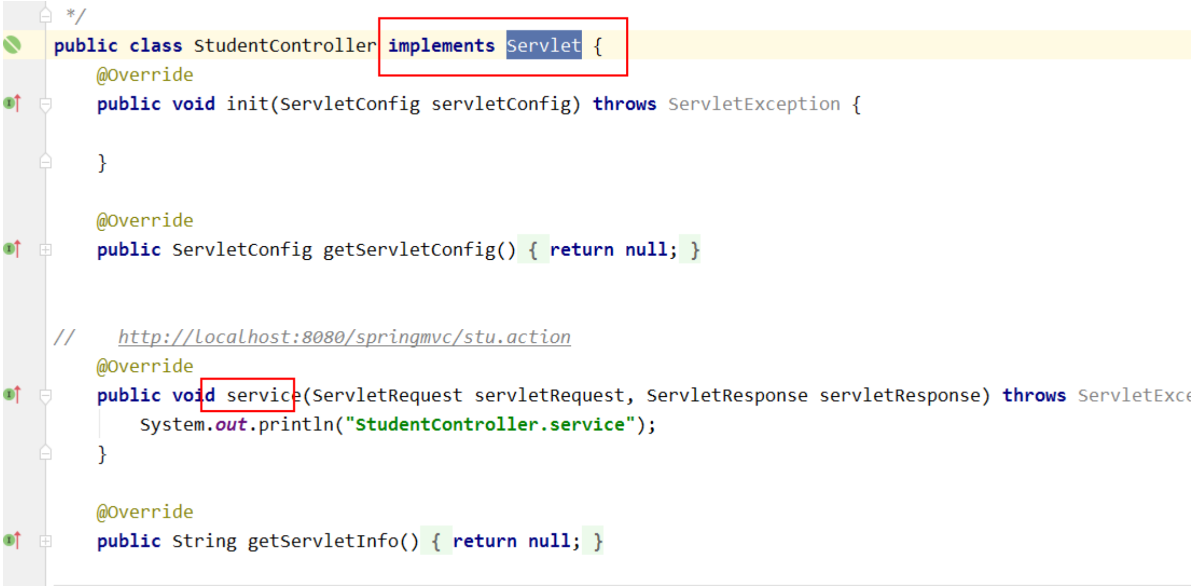
```
 * 描述      : BeanNameUrlHandlerMapping
 */
public class BeanNameUrlController implements HttpRequestHandler {


    @Override
    public void handleRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
        System.out.println("后台BeanNameUrlController执行");

        PrintWriter out = response.getWriter();
        out.write( s: "beanNameurlController");
        out.flush();
        out.close();
    }
}
```

3.4.3.2. SimpleServletHandlerAdapter

找到Controller中的service方法执行

```
import javax.servlet.*;
import java.io.IOException;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  10:34 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     :
 */
public class StudentController implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }


//    http://localhost:8080/springmvc/stu.action
    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("StudentController.service");
    }

    @Override
```

```java
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {

    }
}
```

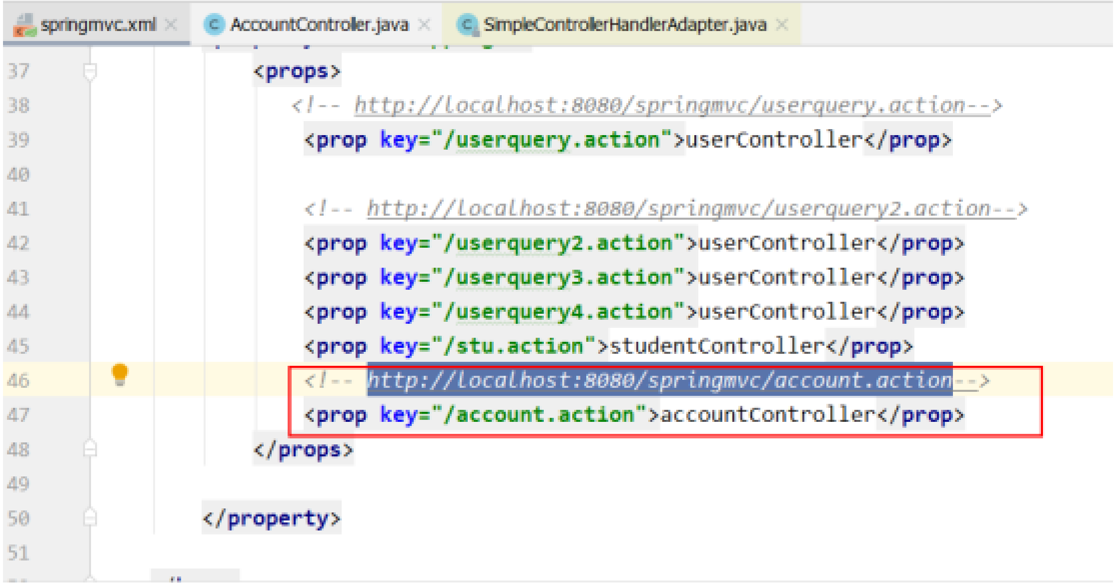


```java
*/
public class StudentController implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() { return null; }


//    http://localhost:8080/springmvc/stu.action
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletExce
        System.out.println("StudentController.service");
    }

    @Override
    public String getServletInfo() { return null; }
```

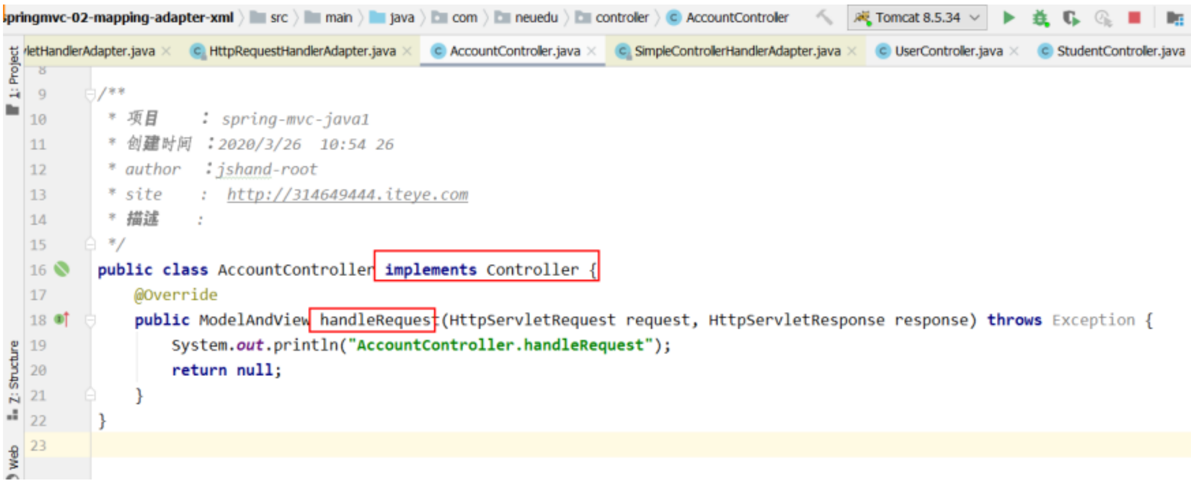
3.4.3.3. SimpleControllerHandlerAdapter**

执行类中实现自Controller接口的handleRequest方法,需要类实现Controller接口

```java
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  10:54 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     :
 */
public class AccountController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        System.out.println("AccountController.handleRequest");
        return null;
    }
}
 <!--3 会调用Controller接口的子类型的 handleRequest org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter-->
 <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>
```



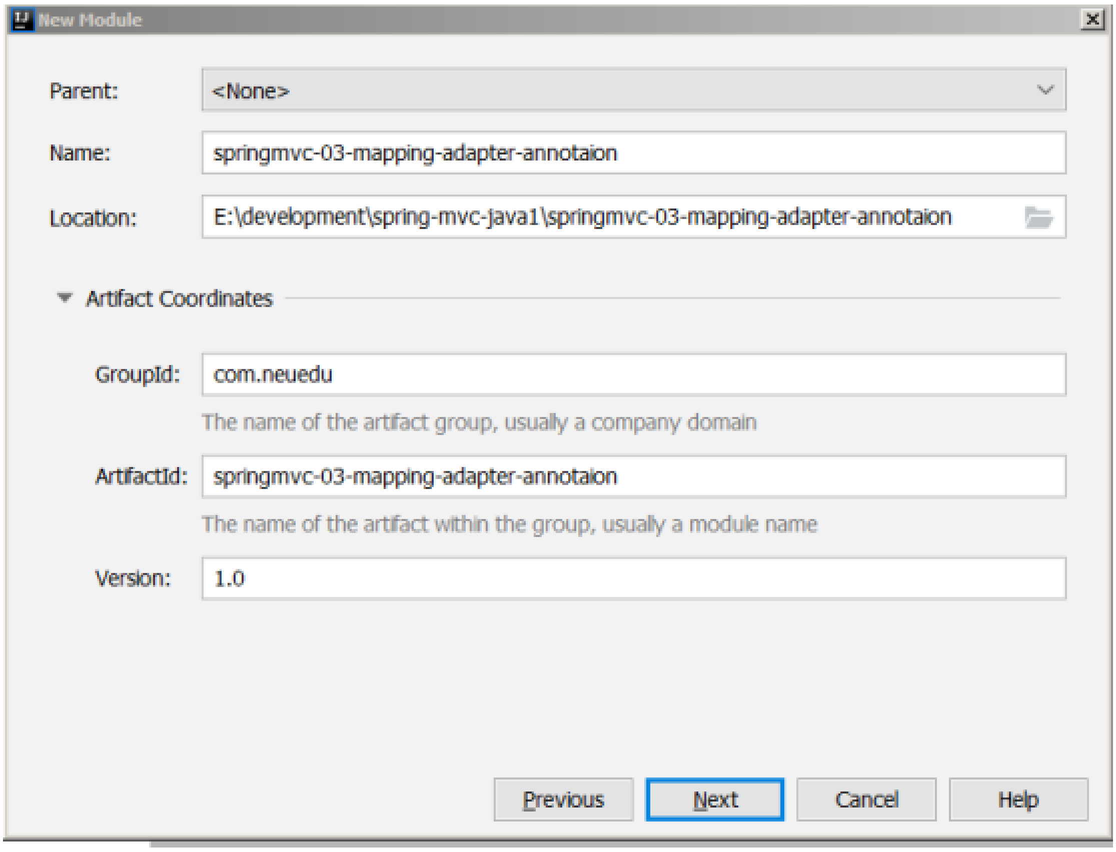
```
        <props>
            <!-- http://localhost:8080/springmvc/userquery.action-->
            <prop key="/userquery.action">userController</prop>

            <!-- http://localhost:8080/springmvc/userquery2.action-->
            <prop key="/userquery2.action">userController</prop>
            <prop key="/userquery3.action">userController</prop>
            <prop key="/userquery4.action">userController</prop>
            <prop key="/stu.action">studentController</prop>
            <!-- http://localhost:8080/springmvc/account.action-->
            <prop key="/account.action">accountController</prop>
        </props>

    </property>
```



```
/**
 * 项目     : spring-mvc-java1
 * 创建时间  ：2020/3/26  10:54 26
 * author   ：jshand-root
 * site     :  http://314649444.iteye.com
 * 描述     :
 */
public class AccountController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        System.out.println("AccountController.handleRequest");
        return null;
    }
}
```

4. 注解开发(处理器映射器、处理器适配器)

创建webapp骨架的项目



New Module

Parent: <None>

Name: springmvc-03-mapping-adapter-annotaion

Location: E:\development\spring-mvc-java1\springmvc-03-mapping-adapter-annotaion

Artifact Coordinates

GroupId: com.neuedu

The name of the artifact group, usually a company domain

ArtifactId: springmvc-03-mapping-adapter-annotaion

The name of the artifact within the group, usually a module name

Version: 1.0

4.1. 搭建springmvc程序

4.1.1. Pom**

4.1.2. Springmvc.xml**

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

    <!--配置扫描组件-->
    <context:component-scan base-package="com.neuedu.controller"/>


</beans>
```

4.1.3. Web.xml中配置前端控制器

```xml
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <!--前端控制器-->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <!--默认的配置文件的名字applicationContext.xml-->
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>

  </servlet>

  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

4.2. 配置注解形式的映射器、适配器

4.2.1. 配置IOC容器中的两个类

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

    <!--配置扫描组件-->
    <context:component-scan base-package="com.neuedu.controller"/>

    <!--注解形式的处理器映射器（Mapping）-->
    <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

    <!--注解形式的处理器适配器（Adapter）-->
    <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>


</beans>
```

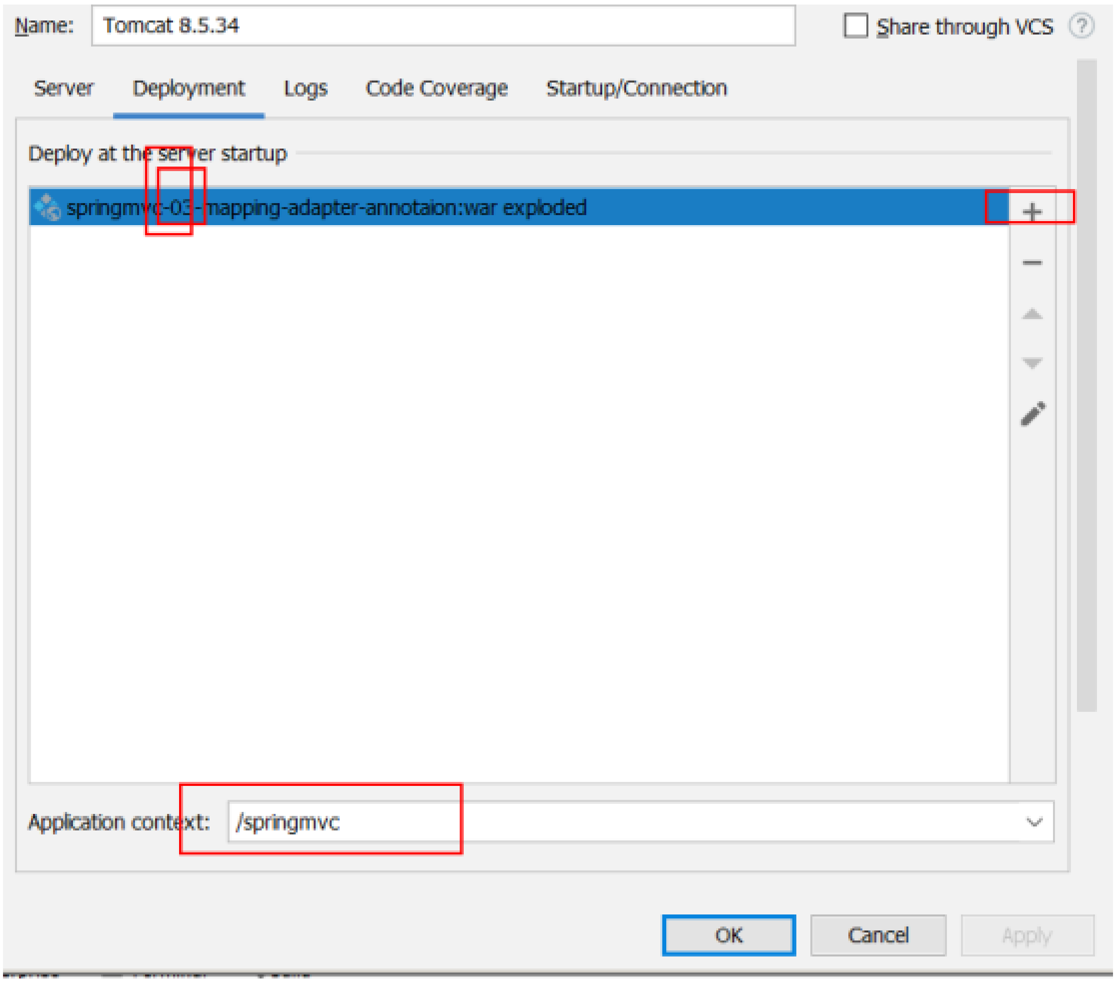
4.2.2. 使用annotation驱动的形式声明映射器、适配器

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

    <!--配置扫描组件-->
    <context:component-scan base-package="com.neuedu.controller"/>

    <!--annotation-driven 代替上述映射器 和适配器 有些额外的功能-->
    <mvc:annotation-driven/>


</beans>
```

4.2.3. 具体的Controller

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  11:33 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 注解形式
 */
@Controller
public class UserController {


    /**
     * http://ip:port/context/url
     *
     * http://127.0.0.1:8080/springmvc/test_annotation
     */
    @RequestMapping("/test_annotation")
    public void testAnnotation(){
        System.out.println("测试注解形式的方法");
    }

}
```

4.2.4. 测试：



Name: Tomcat 8.5.34

□ Share through VCS

Deploy at the server startup

springmvc-03-mapping-adapter-annotaion:war exploded

Application context: /springmvc

http://127.0.0.1:8080/springmvc/test_annotation

5. 映射请求

使用@RequestMapping注解用于在类或者方法上进行声明，类上面可以没有。如果没有那么我们请求路径：url的值即为@RequestMapping注解中的路径

http://ip:port/context/url

5.1. 使用

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  15:37 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
```

```java
 * 描述     : 测试@RequestMapping注解
 */
@Controller //让IOC容器管理组件
public class RequestController {

    //http://localhost:8080/springmvc/req1
    //http://192.168.81.3:8080/springmvc/req1
    @RequestMapping("/req1")
    public void req1(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("测试在方法中定义@RequestMapping注解");

        PrintWriter out = response.getWriter();
        out.println("req1:"+new Date().getTime());
        out.flush();
        out.close();
    }
```

5.2. 在类和方法中同时存在@RequestMapping注解

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/26  15:47 26
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 账户的控制器
 */
@Controller
@RequestMapping("/account")
public class AccountController {

//    http://localhost:8080/springmvc/account/insert
    @RequestMapping("/insert")
    public void insert(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("账户的插入");


        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("账户的插入:"+new Date().getTime());
        out.flush();
        out.close();
    }

    //http://localhost:8080/springmvc/account/update
    @RequestMapping("/update")
    public void update(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("账户的修改");
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("账户的修改:"+new Date().getTime());
        out.flush();
        out.close();
    }
}
```

5.3. @RequestMapping的其他属性

通过value匹配url，还可以配合着method、params、headers属性一起精细化的匹配

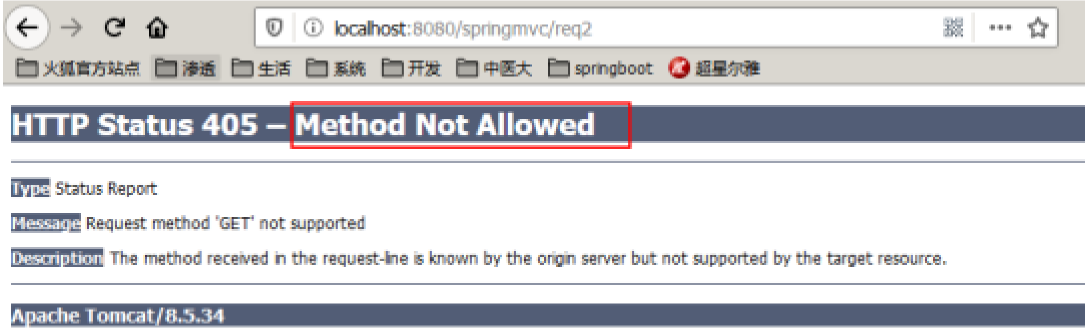
5.3.1. value

5.3.2. Method

用于匹配不同的http请求方法（POST、GET、DELETE、PUSH...7）

```
//http://localhost:8080/springmvc/req2  方法是 POST
    //http://localhost:8080/springmvc/index.jsp 上的按钮触发此次请求  方法是 POST
    @RequestMapping(value = "/req2",method ={RequestMethod.POST})
//    @RequestMapping(value = "/req2",method ={RequestMethod.POST, RequestMethod.GET})
    public void req2(HttpServletRequest request, HttpServletResponse response) throws IOException {
        System.out.println("用于支持post请求");

        PrintWriter out = response.getWriter();
        out.println("POST request :"+new Date().getTime());
        out.flush();
        out.close();
    }
<form method="post" action="req2">
    <input type="submit" value="请求后端的post方法"/>
</form>
```

出现如下问题需要考虑方法上的RequestMapping注解是否指定了method属性



HTTP Status 405 – Method Not Allowed

Type Status Report

Message Request method 'GET' not supported

Description The method received in the request-line is known by the origin server but not supported by the target resource.

Apache Tomcat/8.5.34

5.3.3. params

用于区分是否携带对应参数，对参数名字、值的匹配

param1: 表示请求必须包含名为 param1 的请求参数

!param1: 表示请求不能包含名为 param1 的请求参数

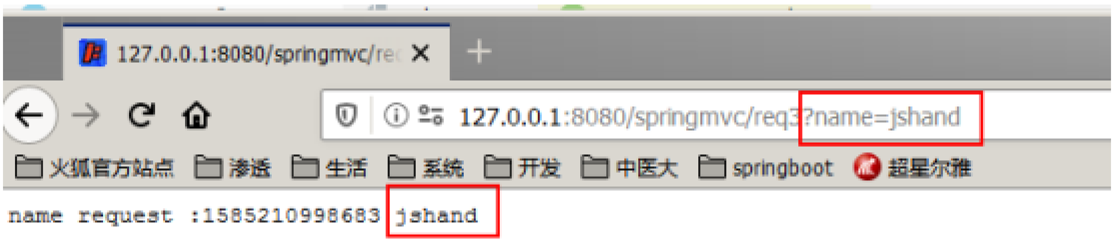
param1 != value1: 表示请求包含名为 param1 的请求参数，但其值不能为 value1

{“param1=value1”, “param2”}: 请求必须包含名为 param1 和param2 的两个请求参数，且 param1 参数的值必须为 value1

5.3.3.1. param1: 表示请求必须包含名为 param1 的请求参数

```java
//请求路径中必须包含参数名：name
//http://127.0.0.1:8080/springmvc/req3?name=jshand
@RequestMapping(value = "/req3",params ={"name"} )
public void req3(HttpServletRequest request, HttpServletResponse response) throws IOException {
    String name = request.getParameter("name");
    System.out.println("用于支持post请求"+name);

    PrintWriter out = response.getWriter();
    out.println("name request :"+new Date().getTime()+" "+name);
    out.flush();
    out.close();
}
```

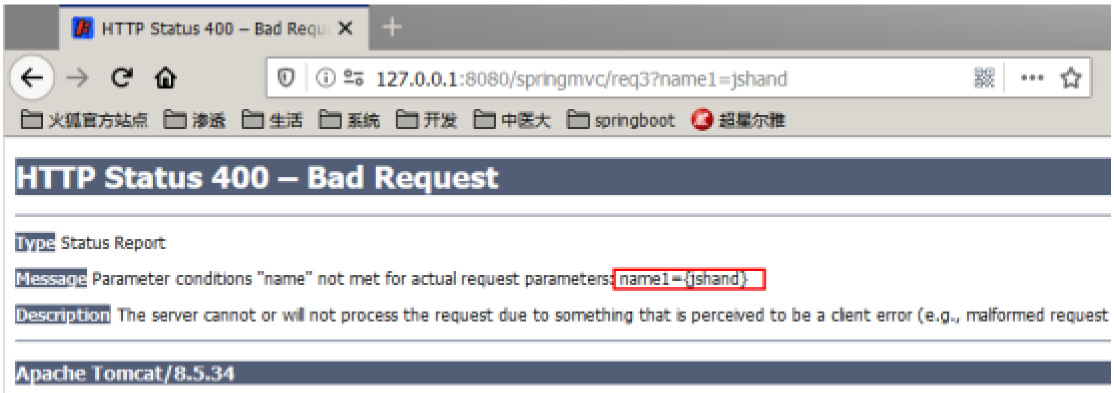
正确的情况：



127.0.0.1:8080/springmvc/req3?name=jshand

```
name request :1585210998683 jshand
```

如果不包含name参数：HTTP 400(参数、请求的问题):



127.0.0.1:8080/springmvc/req3?name1=jshand

HTTP Status 400 – Bad Request

Type Status Report

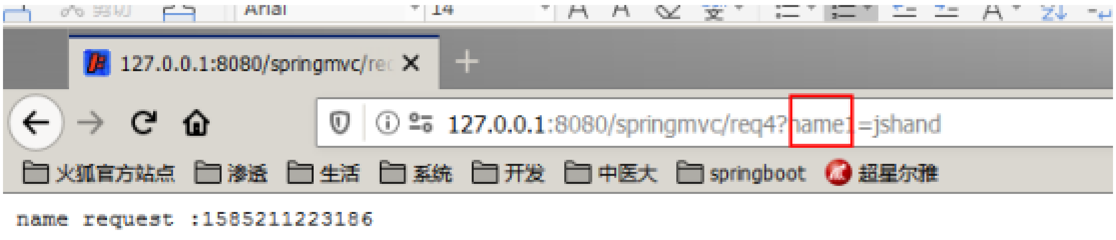
Message Parameter conditions "name" not met for actual request parameters: name1={jshand}

Description The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request

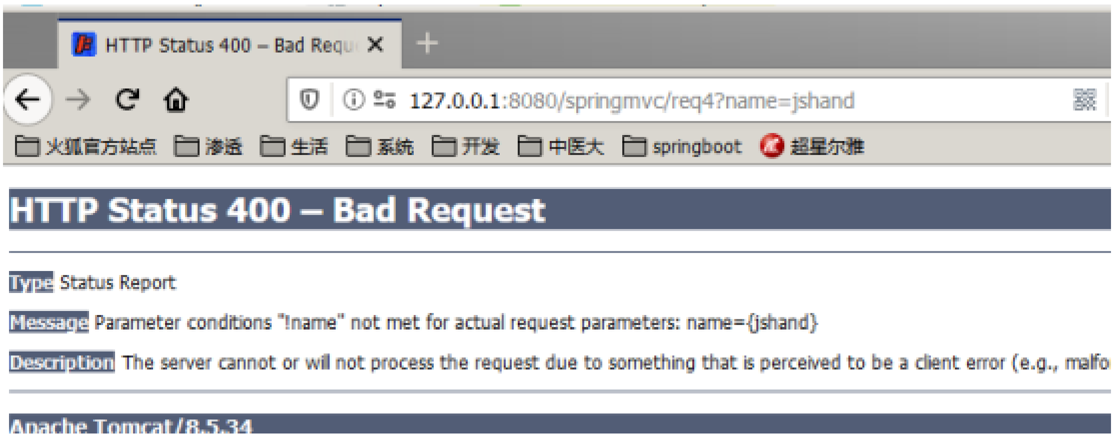
Apache Tomcat/8.5.34

5.3.3.2. !param1: 表示请求不能包含名为 param1 的请求参数

```java
//请求路径中不能出现name参数
//http://127.0.0.1:8080/springmvc/req4?name=jshand   错误
//http://127.0.0.1:8080/springmvc/req4?p1=va1        √
@RequestMapping(value = "/req4",params ={"!name"} )
public void req4(HttpServletRequest request, HttpServletResponse response) throws IOException {

    System.out.println("用于支持post请求");

    PrintWriter out = response.getWriter();
    out.println("name request :"+new Date().getTime());
    out.flush();
    out.close();
}
```


127.0.0.1:8080/springmvc/re
127.0.0.1:8080/springmvc/req4?name1=jshand
火狐官方站点
渗透
生活
系统
开发
中医大
springboot
超星尔雅

```
name request :1585211223186
```


HTTP Status 400 – Bad Requ
127.0.0.1:8080/springmvc/req4?name=jshand
火狐官方站点
渗透
生活
系统
开发
中医大
springboot
超星尔雅

HTTP Status 400 – Bad Request

Type Status Report

Message Parameter conditions "!name" not met for actual request parameters: name={jshand}

Description The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malfo

Apache Tomcat/8.5.34

5.3.3.3. param1=value1: 表示请求包含名为 param1 的请求参数且值等于value1;参数必须传

```java
//请求路径中必须传递参数name并且值需要跟jshand一直
//http://127.0.0.1:8080/springmvc/req5?name=jshand
@RequestMapping(value = "/req5",params ={"name=jshand"} )
public void req5(HttpServletRequest request, HttpServletResponse response) throws IOException {

    System.out.println("用于支持post请求");

    PrintWriter out = response.getWriter();
    out.println("name request :"+new Date().getTime());
    out.flush();
    out.close();
}
```

5.3.3.4. param1!=value1: 表示请求如果包含名为 param1 的请求参数且值不能等于value1;参数可以不传

```
//请求路径中如果传递参数name并且值不等于jshand，可以不传name参数
//http://127.0.0.1:8080/springmvc/req6?name=jshand      错误
//http://127.0.0.1:8080/springmvc/req6?name=jshand112   正确
//http://127.0.0.1:8080/springmvc/req6?                  正确
@RequestMapping(value = "/req6",params ={"name!=jshand"} )
public void req6(HttpServletRequest request, HttpServletResponse response) throws IOException {

    System.out.println("用于支持post请求");

    PrintWriter out = response.getWriter();
    out.println("name request :"+new Date().getTime());
    out.flush();
    out.close();
}
```

5.3.4. headers

```
  //http://127.0.0.1:8080/springmvc/req7    火狐浏览器（演示机器）
//    @RequestMapping(value = "/req7" )
   @RequestMapping(value = "/req7",headers ={"User-Agent=Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:74.0) Gecko/20100101 Firefox/73.0" } )
   public void req7(HttpServletRequest request, HttpServletResponse response) throws IOException {

       System.out.println("用于支持火狐请求");

       PrintWriter out = response.getWriter();
       out.println("FireFox request :"+new Date().getTime());
       out.flush();
       out.close();
}
```

5.4. RequestMapping的变种

@PostMapping 相当于是 @RequestMapping(method = {RequestMethod.POST})

@GetMapping 相当于是 @RequestMapping(method = {RequestMethod.GET})

```
@GetMapping(value = "/get_mapping" )
public void getMapping(HttpServletRequest request, HttpServletResponse response) throws IOException {

    System.out.println("doGet 请求成功 ");

    PrintWriter out = response.getWriter();
    out.println("Get request :"+new Date().getTime());
    out.flush();
    out.close();
}

@PostMapping(value = "/post_mapping" )
```

```
public void postMapping(HttpServletRequest request, HttpServletResponse response) throws IOException {

    System.out.println("doPost 请求成功 ");

    PrintWriter out = response.getWriter();
    out.println("Post request :"+new Date().getTime());
    out.flush();
    out.close();
}
```

6. 方法返回值

控制器的目标最终要给浏览器客户端进行响应（内容：html、json-js、ajax）

6.1. void

返回值是void以为着需要编程进行相应，方法入参需要显示的声明request、response.

6.1.1. 使用request转向页面，

如下：

```
request.getRequestDispatcher("页面路径").forward(request, response);
```

6.1.2. 可以通过response页面重定向：

```
response.sendRedirect("url")
```

6.1.3. 可以通过response指定响应结果，

例如响应json数据如下：

```
response.setCharacterEncoding("utf-8");
response.setContentType("application/json;charset=utf-8");
response.getWriter().write("json串");
@RequestMapping("/req1")
public void req1(HttpServletRequest request, HttpServletResponse response) throws IOException {
    System.out.println("测试在方法中定义@RequestMapping注解");
    PrintWriter out = response.getWriter();
    out.println("req1:"+new Date().getTime());
    out.flush();
    out.close();
}
```

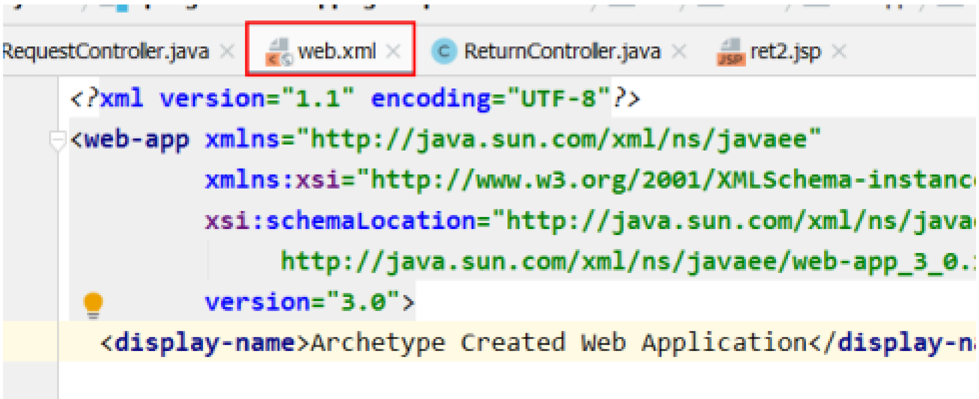
6.2. ModelAndView

自己设置mode 和视图，由【视图解析器】进行渲染响应（html）

6.2.1. 添加jstl依赖

```xml
<!-- https://mvnrepository.com/artifact/jstl/jstl -->
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

6.2.2. 使用servlet-3.0的版本



```xml
<?xml version="1.1" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
         xsi:schemaLocation="http://java.sun.com/xml/ns/java
             http://java.sun.com/xml/ns/javaee/web-app_3_0.
         version="3.0">
  <display-name>Archetype Created Web Application</display-n
```

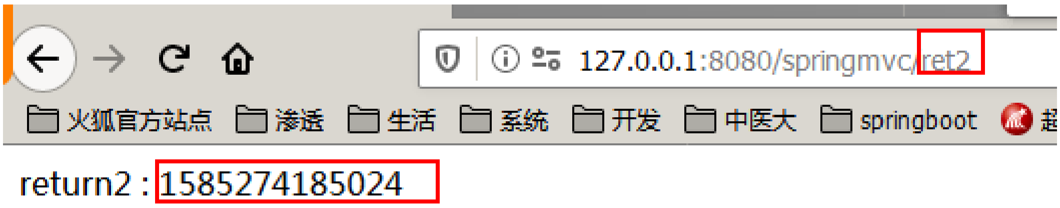
6.2.3. 映射的方法

```java
//http://127.0.0.1:8080/springmvc/ret2
@RequestMapping("/ret2")
public ModelAndView ret2() throws IOException {
    System.out.println("返回值为void");


    //代表模型和视图的对象
    ModelAndView mav = new ModelAndView();

    //类似于requet.setAttribute("attrName",'attrValue');
    mav.addObject("time",new Date().getTime());


    //模拟从数据库查询出的 用户列表(User --Map)
    List<Map> list = new ArrayList();

    for (int i = 0; i <10; i++) {
        Map user = new HashMap();
        user.put("name","name"+i);
        user.put("age",30+i);
        user.put("addres","address"+i);
        list.add(user);
    }


    mav.addObject("time",new Date().getTime());
    mav.addObject("list",list);
    //想要跳转到此位置
    mav.setViewName("/return/ret2.jsp");
    return mav;
}
```

6.2.4. 跳转的jsp

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 9:27
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java"  %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

return2 : ${time}

    <table border="1" cellspacing="0" cellpadding="0" width="100%">
        <tr>
            <td>name</td>
            <td>age</td>
            <td>addres</td>
        </tr>

        <c:forEach items="${list}" var="user">
            <tr>
                <td>${user.name}</td>
                <td>${user.age}</td>
                <td>${user.addres}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

6.2.5. 测试



127.0.0.1:8080/springmvc/ret2

```
return2 : 1585274185024
```

6.3. 视图解析器

6.4. String**

6.4.1. 代表视图名称

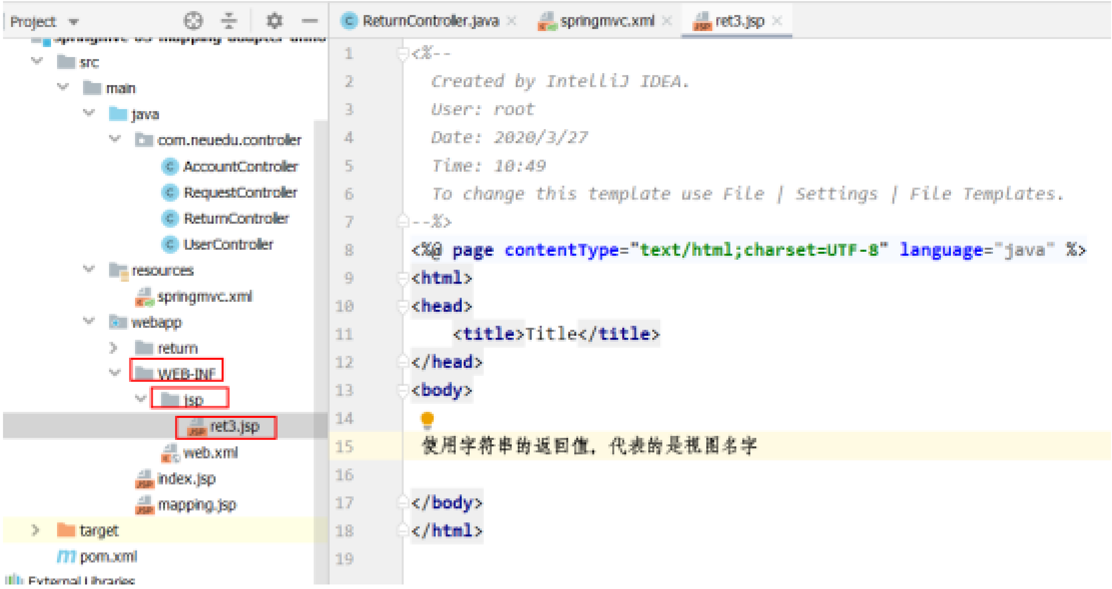
根据视图解析器配置的前缀、后缀,自动的匹配完整的路径

6.4.1.1. 映射方法

默认的是内部跳转，可以使用request共享数据，视图名称会受视图解析器的前后缀影响

```java
//http://127.0.0.1:8080/springmvc/ret3
@RequestMapping("/ret3")
public String ret3() throws IOException {
    //prefix        /WEB-INF/jsp/
    //suffix        .jsp

    // 相当于:  /WEB-INF/jsp/ret3.jsp
    return "ret3";
}
```

6.4.1.2. 跳转的jsp



```jsp
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 10:49
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

 使用字符串的返回值，代表的是视图名字

</body>
</html>
```

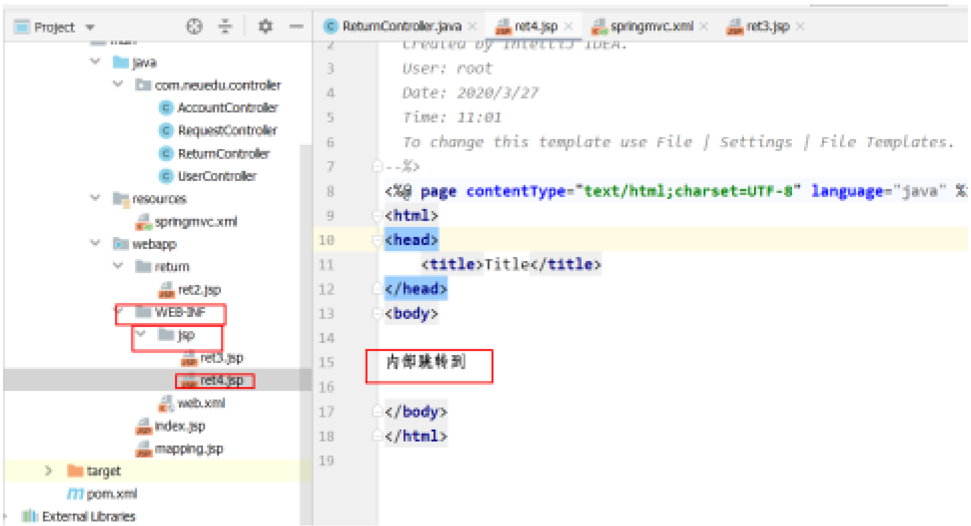
6.4.2. 内部跳转

6.4.2.1. 跳转到内部的位置

内部跳转 不会受视图解析器的前后缀影响，路径需要写完整,WEB-INF目录中的资源不能被浏览器直接访问，可以通过内部跳转的形式进行访问，目录是安全。可以在Controller和JSP中共享requst对象

```java
//http://127.0.0.1:8080/springmvc/ret4
@RequestMapping("/ret4")
public String ret4() throws IOException {
    return "forward:/WEB-INF/jsp/ret4.jsp";
}
```

6.4.2.2. Jsp位置



```
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 11:01
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %
<html>
<head>
    <title>Title</title>
</head>
<body>

内部跳转到

</body>
</html>
```

6.4.3. 重定向

重定向不能共享request

6.4.3.1. 映射方法

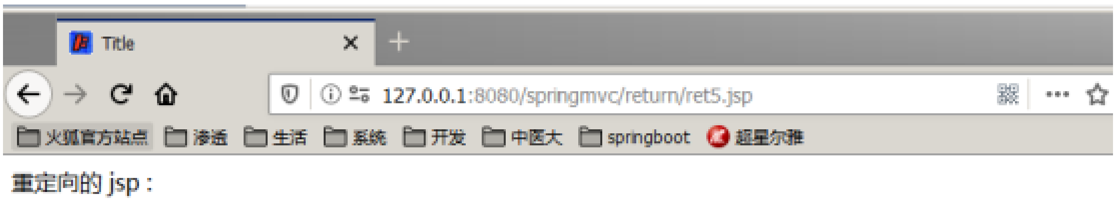
```
//http://127.0.0.1:8080/springmvc/ret5
@RequestMapping("/ret5")
public String ret5(HttpServletRequest request) throws IOException {
    request.setAttribute("time",new Date().getTime());
    return "redirect:/return/ret5.jsp"; //重定向：让浏览器重新请求 此路径
}
```

6.4.3.2. Jsp的写法

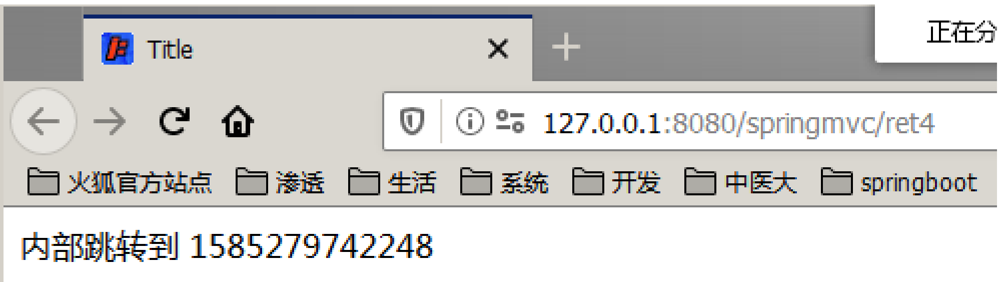
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

重定向的 jsp  : ${time}

</body>
</html>
```

6.4.3.3. 与内部跳转的对比

不能共享request



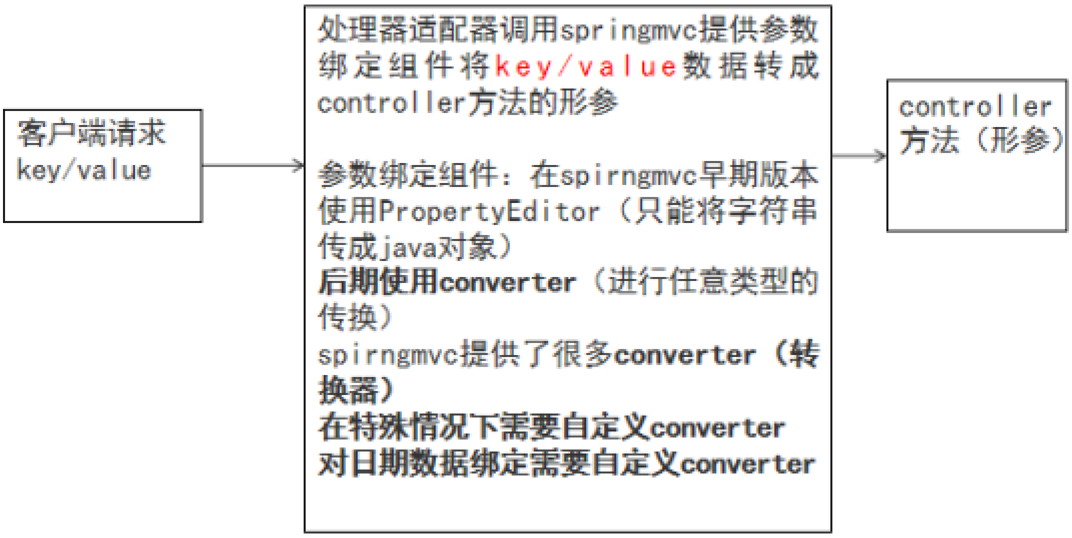
重定向的 jsp :



内部跳转到 1585279742248

7. 参数绑定

7.1. 参数绑定的过程



客户端请求key/value

处理器适配器调用springmvc提供参数绑定组件将key/value数据转成controller方法的形参

参数绑定组件：在spirngmvc早期版本使用PropertyEditor（只能将字符串传成java对象）
后期使用converter（进行任意类型的传换）
spirngmvc提供了很多converter（转换器）
在特殊情况下需要自定义converter
对日期数据绑定需要自定义converter

controller方法（形参）

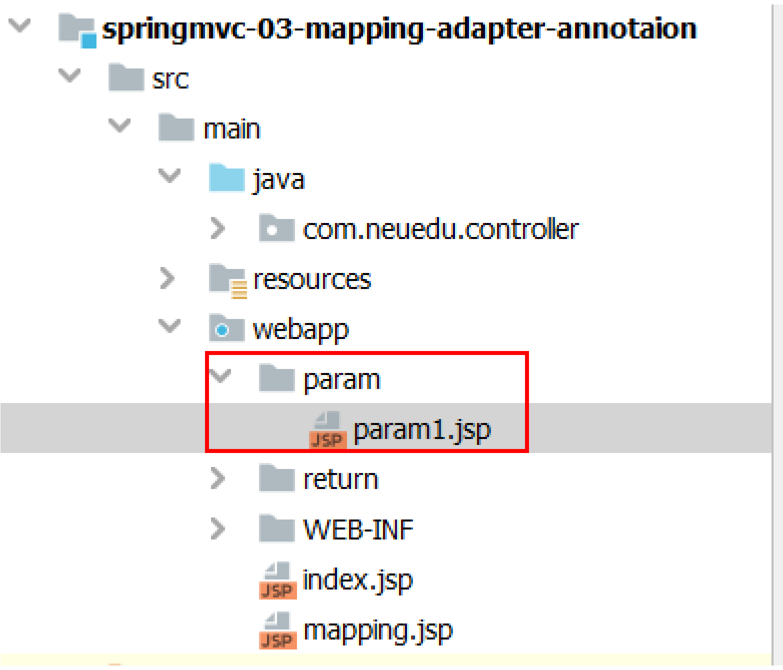
7.2. 内置的参数

7.2.1. HttpServletRequest、HttpServletResponse、HTTPSession

7.2.1.1. 映射方法

```
//http://127.0.0.1:8080/springmvc/param1
@RequestMapping("/param1")
public void param1(HttpServletRequest request, HttpServletResponse response,
HttpSession session) throws ServletException, IOException {
    request.setAttribute("attr_req","value_req");
    session.setAttribute("attr_sess","value_sess");
    request.getRequestDispatcher("/param/param1.jsp").forward(request,response);
}
```

7.2.1.2. Jsp



```
springmvc-03-mapping-adapter-annotaion
  src
    main
      java
        com.neuedu.controller
      resources
      webapp
        param
          param1.jsp
        return
        WEB-INF
        index.jsp
        mapping.jsp
```

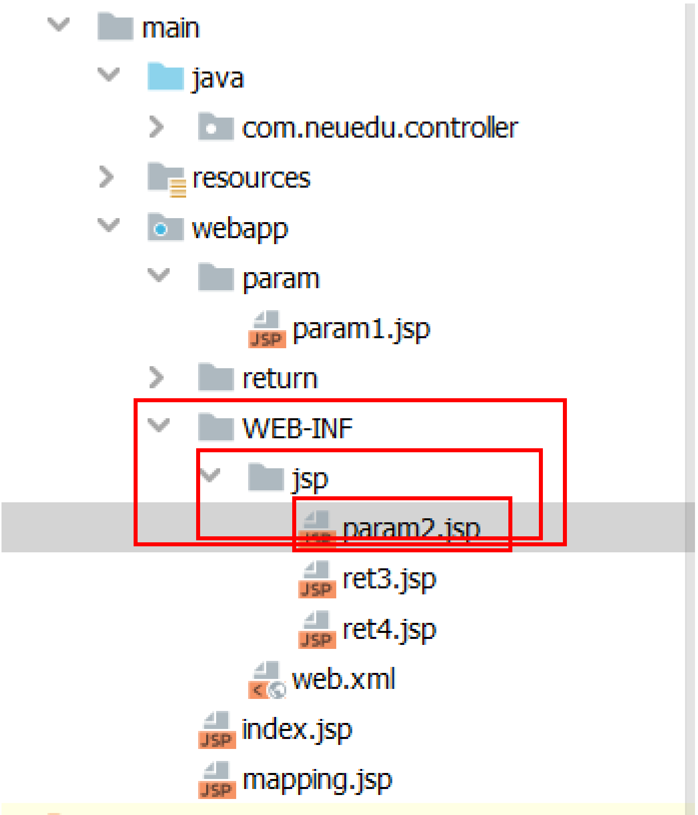
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

测试内置参数<br/>

request作用域测试： ${attr_req}<br/>
session作用域测试：${attr_sess}<br/>

</body>
</html>
```

7.2.2. Model、ModelMap

7.2.2.1. 映射方法

```
//http://127.0.0.1:8080/springmvc/param2
@RequestMapping("/param2")
//public String param2(Model model) throws ServletException, IOException {
public String param2(ModelMap model) throws ServletException, IOException {

    //相当于是向request作用域设置属性
    model.addAttribute("time",new Date().getTime());
    model.addAttribute("title","测试内置参数Model、ModelMap");

    return "param2";
}
```

7.2.2.2. Jsp



```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 13:40
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>${title}</title>
</head>
<body>

${title}<br/>
${time}

</body>
</html>
```

7.3. 基础类型

支持整型、字符串、单精度/双精度、布尔型

当请求的【参数名称】和【处理器形参名称】一致(区分大小写)时会将请求参数与形参进行绑定。

控制器的映射方法

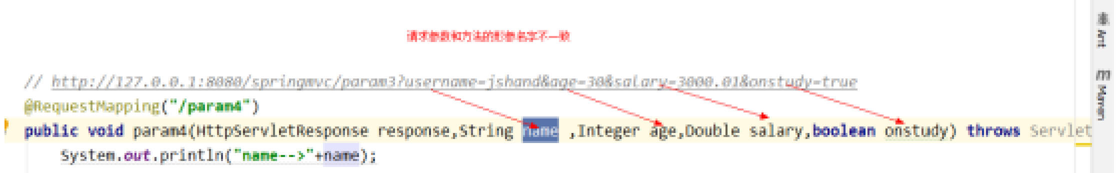
```
// http://127.0.0.1:8080/springmvc/param3?name=jshand&age=30&salary=3000.01&onstudy=true
@RequestMapping("/param3")
public void param3(HttpServletResponse response,String name ,Integer age,Double salary,boolean onstudy) throws ServletException, IOException {
    System.out.println("name-->"+name);
    System.out.println("age-->"+age);
    System.out.println("salary-->"+salary);
    System.out.println("onstudy-->"+onstudy);

    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();

    out.println("<div style='border:1px solid red'>name:"+name+"</div>");
    out.println("<div style='border:1px solid red'>age:"+age+"</div>");
    out.println("<div style='border:1px solid red'>salary:"+salary+"</div>");
    out.println("<div style='border:1px solid red'>onstudy:"+onstudy+"</div>");

    out.flush();
    out.close();
}
```

7.3.1. @RequestParam**

当请求的【参数名称】和【处理器形参名称】不一致的时候需要使用注解@RequestParam进行自定义的绑定,

声明次注解则默认该参数必须提供（必须传）,可以使用required属性=false设置为非必须。

通过defaultValue属性设置默认值



请求参数和方法的形参名字不一致

```
// http://127.0.0.1:8080/springmvc/param3?username=jshand&age=30&salary=3000.01&onstudy=true
@RequestMapping("/param4")
public void param4(HttpServletResponse response,String name ,Integer age,Double salary,boolean onstudy) throws Servlet
    System.out.println("name-->"+name);
```

```
// http://127.0.0.1:8080/springmvc/param4?username=jshand&age=30&salary=3000.01&onstudy=true
@RequestMapping("/param4")
public void param4(HttpServletResponse response, @RequestParam(value = "username",required = false,defaultValue="admin") String name , Integer age, Double salary, boolean onstudy) throws ServletException, IOException {
    System.out.println("name-->"+name);
    System.out.println("age-->"+age);
    System.out.println("salary-->"+salary);
    System.out.println("onstudy-->"+onstudy);

    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();

    out.println("<div style='border:1px solid red'>name:"+name+"</div>");
    out.println("<div style='border:1px solid red'>age:"+age+"</div>");
    out.println("<div style='border:1px solid red'>salary:"+salary+"</div>");
    out.println("<div style='border:1px solid red'>onstudy:"+onstudy+"</div>");

    out.flush();
    out.close();
}
```

7.4. Pojo类型

7.4.1. 表单

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 14:38
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

        <%--模拟用户注册，存在大量的字段--%>


    <form action="../param5">

        <!--模拟都是 30-字段 -->
        <table>
            <tr>
                <td>用户名字</td>
                <td><input type="text" name="username"></td>
            </tr>
            <tr>
                <td>常用地址</td>
                <td><input type="text" name="password"></td>
            </tr>
            <tr>
                <td>账户余额</td>
                <td><input type="text" name="amount"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="用户注册"></td>
            </tr>
        </table>


    </form>

</body>
</html>
```

7.4.2. 映射方法

```
// http://127.0.0.1:8080/springmvc/param/param5.jsp
@RequestMapping("/param5")
public void insertUser(HttpServletResponse response,
                       User user) throws IOException {
    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();

    out.println("<div style='border:1px solid red'>username:"+user.getUsername()+"</div>");
    out.println("<div style='border:1px solid red'>password:"+user.getPassword()+"</div>");
    out.println("<div style='border:1px solid red'>amount:"+user.getAmount()+"</div>");

    out.flush();
    out.close();
}
```

7.5. Pojo包装的POJO

为了解决同名参数可以使用pojo嵌套pojo解决

7.5.1. ParamVO

```
package com.neuedu.entity;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/30  9:07 30
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     :
 */
public class ParamVO {

    private User user;
    private Person person;

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    @Override
    public String toString() {
        return "ParamVO{" +
                "user=" + user +
                ", person=" + person +
                '}';
    }

}
```

7.5.2. User

```java
package com.neuedu.entity;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/27  14:47 27
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * http://127.0.0.1:8080/context?username=aaa&password=xxx
 * 描述     :
 */

public class User {
    private String username;
    private String password;
    private Double amount;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", amount=" + amount +
                '}';
    }
}
```

7.5.3. Person

```java
package com.neuedu.entity;

/**
 * 项目    ：  spring-mvc-java1
 * 创建时间 ：2020/3/30  8:54 30
 * author  ：jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 人员的实体
 */
public class Person {

    private String name;
    private Integer age;
    private Double amount;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", amount=" + amount +
                '}';
    }
}
```

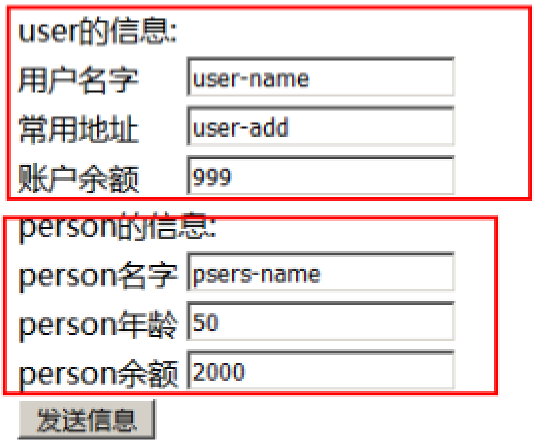
7.5.4. 控制器

```java
/**
 * 接受两张表的 信息
 * @param response

 * @throws IOException
 */
@RequestMapping("/param6")
public void param6(HttpServletResponse response, ParamVO paramVO) throws IOException {

    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();

    out.println("<div style='border:1px solid red'>user：:"+
            paramVO.getUser().toString()
            +"</div>");


    out.println("<div style='border:1px solid blue'>person：:"+
            paramVO.getPerson().toString()
            +"</div>");


    out.flush();
    out.close();
}
```

7.5.5. Form表单页面

```jsp
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 14:38
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <form action="../param6">


        <table>
            <tr>
                <td colspan="2"> user的信息:</td>

            </tr>
            <tr>
                <td>用户名字</td>
                <td><input type="text" name="user.username"></td>
            </tr>
            <tr>
                <td>常用地址</td>
                <td><input type="text" name="user.password"></td>
            </tr>
            <tr>
                <td>账户余额</td>
                <td><input type="text" name="user.amount"></td>
            </tr>
            <tr>
                <td colspan="2"> person的信息:</td>

            </tr>
            <tr>
                <td>person名字</td>
                <td><input type="text" name="person.name"></td>
            </tr>
            <tr>
                <td>person年龄</td>
                <td><input type="text" name="person.age"></td>
            </tr>
            <tr>
                <td>person余额</td>
                <td><input type="text" name="person.amount"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="发送信息"></td>
            </tr>
        </table>


    </form>

</body>
</html>
```

7.5.6. 测试效果

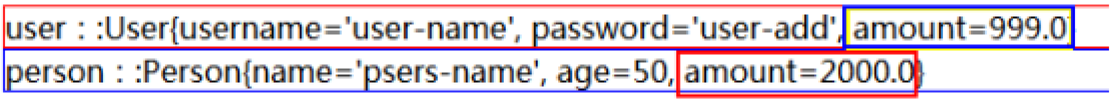


user的信息:

用户名字	user-name
常用地址	user-add
账户余额	999

person的信息:

person名字	psers-name
person年龄	50
person余额	2000



```
user : :User{username='user-name', password='user-add', amount=999.0}
person : :Person{name='psers-name', age=50, amount=2000.0}
```

7.6. 数组类型

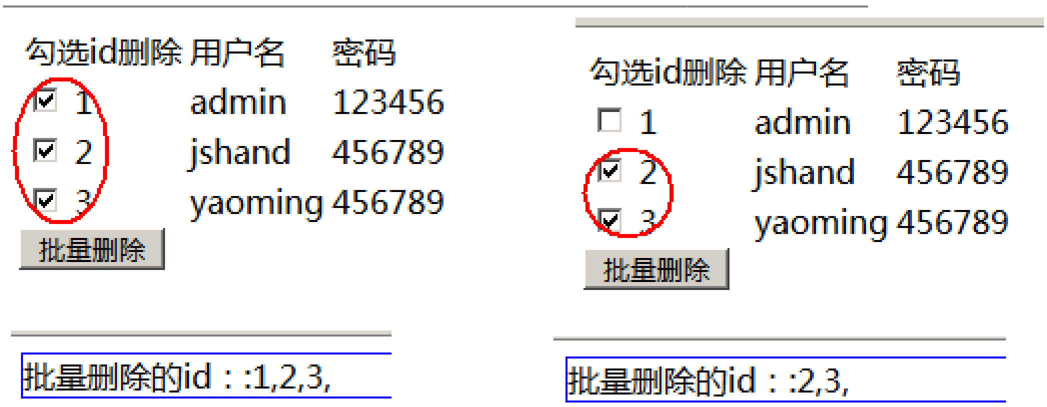
7.6.1. 控制器

```java
/***
 * 接受数组
 *
 * 批量删除用户信息   【userId、userId、userId、userId】
 *
 */
@RequestMapping("/param7")
public void param7(HttpServletResponse response, Integer[] userId) throws IOException {
    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();


    StringBuffer ids = new StringBuffer();
    for (Integer id : userId) {
        ids.append(id+",");
    }

    out.println("<div style='border:1px solid blue'>批量删除的id：:"+
            ids.toString()
            +"</div>");
    out.flush();
    out.close();
}
```

7.6.2. Form表单

```jsp
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 14:38
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <form action="../param7">


        <table>
            <tr>
                <td>勾选id删除</td>
                <td>用户名</td>
                <td>密码</td>
            </tr>

            <tr>
                <td><input type="checkbox" name="userId" value="1"/> 1</td>
                <td>admin</td>
                <td>123456</td>
            </tr>

            <tr>
                <td><input type="checkbox" name="userId" value="2"/> 2</td>
                <td>jshand</td>
                <td>456789</td>
            </tr>

            <tr>
                <td><input type="checkbox" name="userId"  value="3"/> 3</td>
                <td>yaoming</td>
                <td>456789</td>
            </tr>
        </table>

        <input type="submit" value="批量删除">

    </form>



</body>
</html>
```

7.6.3. 页面效果



勾选id删除	用户名	密码
☑ 1	admin	123456
☑ 2	jshand	456789
☑ 3	yaoming	456789

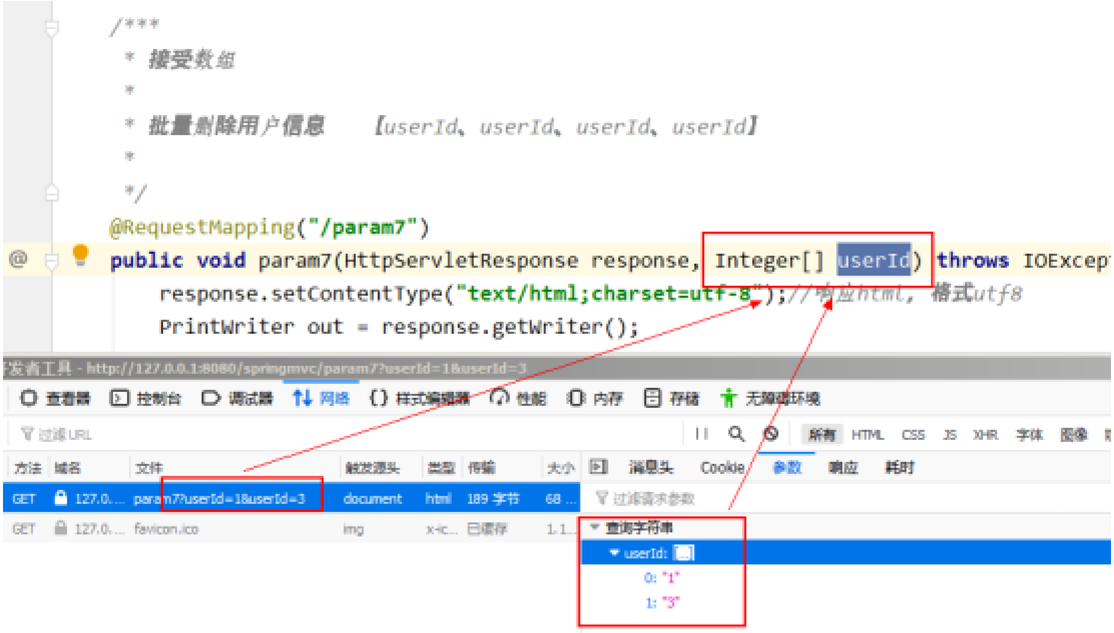
批量删除

勾选id删除	用户名	密码
☐ 1	admin	123456
☑ 2	jshand	456789
☑ 3	yaoming	456789

批量删除

批量删除的id：:1,2,3,

批量删除的id：:2,3,



```
/***
 * 接受数组
 *
 * 批量删除用户信息    【userId、userId、userId、userId】
 *
 */
@RequestMapping("/param7")
public void param7(HttpServletResponse response, Integer[] userId) throws IOExcep
    response.setContentType("text/html;charset=utf-8");//响应html, 格式utf8
    PrintWriter out = response.getWriter();
```

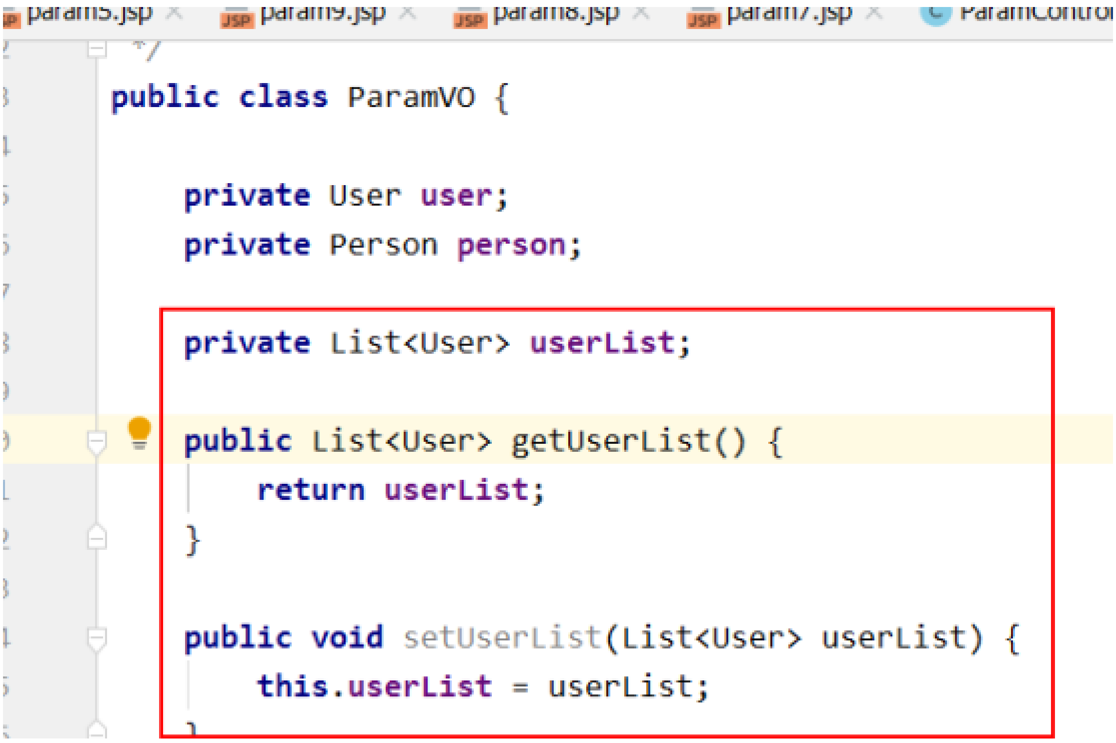
发者工具 - http://127.0.0.1:8080/springmvc/param7?userId=1&userId=3

方法	域名	文件	触发源头	类型	传输	大小
GET	127.0....	param7?userId=1&userId=3	document	html	189 字节	68 ...
GET	127.0....	favicon.ico	img	x-ic...	已缓存	1.1...

```
▼ 查询字符串
  ▼ userId: [...]
      0: "1"
      1: "3"
```

7.7. List类型封装参数

7.7.1. Vo中添加List属性 添加setter、getter方法



```
*/
public class ParamVO {

    private User user;
    private Person person;

    private List<User> userList;

    public List<User> getUserList() {
        return userList;
    }

    public void setUserList(List<User> userList) {
        this.userList = userList;
    }
```

```
package com.neuedu.entity;

import java.util.List;
import java.util.Map;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/30  9:07 30
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     :
 */
public class ParamVO {

    private User user;
    private Person person;

    private List<User> userList;

    public List<User> getUserList() {
        return userList;
    }

    public void setUserList(List<User> userList) {
        this.userList = userList;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }

    @Override
    public String toString() {
        return "ParamVO{" +
                "user=" + user +
                ", person=" + person +
                '}';
    }

}
```

7.7.2. 控制器方法

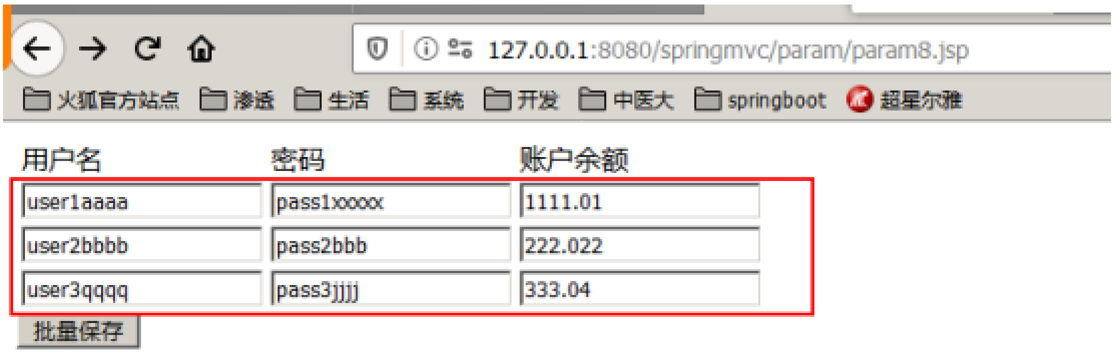
```
/**
 * 使用List接受参数
 * @param response
 * @param vo
 * @throws IOException
 */
@RequestMapping("/param8")
public void param8(HttpServletResponse response, ParamVO vo) throws IOException {
    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();


    StringBuffer userInfos = new StringBuffer();
    for (User user : vo.getUserList()) {
        userInfos.append(user.toString()+"<br/>");
    }

    out.println("<div style='border:1px solid blue'>批量添加用户的信息：:"+
            userInfos.toString()
            +"</div>");
    out.flush();
    out.close();
}
```

7.7.3. Form表单

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 14:38
  To change this template use File | Settings | File Templates.
```

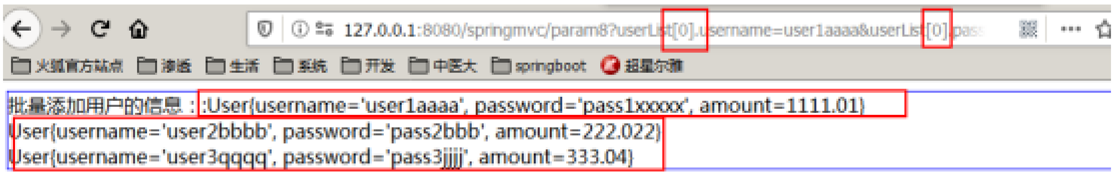
```
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <form action="../param8">


        <table>
            <tr>
                <td>用户名</td>
                <td>密码</td>
                <td>账户余额</td>
            </tr>

            <tr>
                <td><input type="text" name="userList[0].username"/></td>
                <td><input type="text" name="userList[0].password"/></td>
                <td><input type="text" name="userList[0].amount"/></td>
            </tr>
            <tr>
                <td><input type="text" name="userList[1].username"/></td>
                <td><input type="text" name="userList[1].password"/></td>
                <td><input type="text" name="userList[1].amount"/></td>
            </tr>

            <tr>
                <td><input type="text" name="userList[2].username"/></td>
                <td><input type="text" name="userList[2].password"/></td>
                <td><input type="text" name="userList[2].amount"/></td>
            </tr>

        </table>

        <input type="submit" value="批量保存">

    </form>


</body>
</html>
```

7.7.4. 测试效果



用户名	密码	账户余额
user1aaaa	pass1xxxxx	1111.01
user2bbbb	pass2bbb	222.022
user3qqqq	pass3jjjj	333.04

批量保存


127.0.0.1:8080/springmvc/param8?userList[0].username=user1aaaa&userList[0].pass
火狐官方站点
渗透
生活
系统
开发
中医大
springboot
超星尔雅

批量添加用户的信息：:User{username='user1aaaa', password='pass1xxxxx', amount=1111.01}
User{username='user2bbbb', password='pass2bbb', amount=222.022}
User{username='user3qqqq', password='pass3jjjj', amount=333.04}

7.8. Map接受参数

7.8.1. Form表单

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/27
  Time: 14:38
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>


    <form action="../param9">

        <!--
         Account{
             accountName
             amount
         }
         1 (String accountName, double amount)
         2 (Account account )
         3 (@RequestParam Map account )

         -->
        <table>
            <tr>
                <td>账户名</td>
                <td><input type="text" name="accountName"></td>
            </tr>
            <tr>
                <td>账户余额</td>
                <td><input type="text" name="amount"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="新增账户"></td>
            </tr>
        </table>


    </form>



</body>
</html>
```

7.8.2. 控制器

需要注意，在Map参数上添加@RequestParam注解才能绑定参数

```
/**
 * 使用Map接受参数
 * @param response
 * @param map  上需要添加 @RequestPara 注解
 * @throws IOException
 */
@RequestMapping("/param9")
public void param9(HttpServletResponse response,@RequestParam Map map) throws IOException {
    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();




    out.println("<div style='border:1px solid blue'>批量添加用户的信息：:"+
            map
            +"</div>");
    out.flush();
    out.close();
}
```

7.9. 自定义的参数转换

页面上传递过的基础类型（Stirng、浮点型、整数等）可以直接绑定，Date特殊Springmvc默认无法转换，需要自定义转换器

7.9.1. 自定义一个转换器

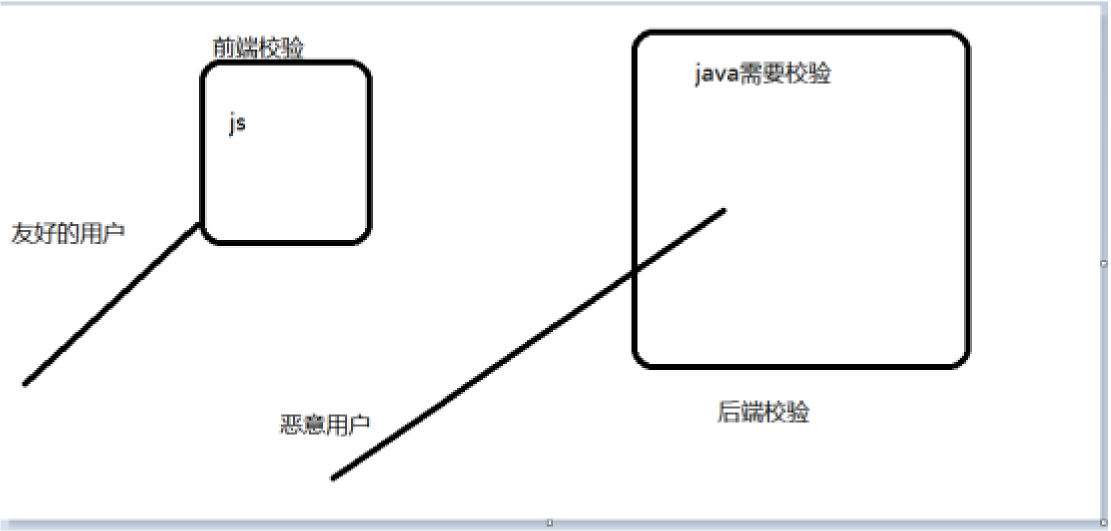
```java
package com.neuedu.converter;

import org.springframework.core.convert.converter.Converter;;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.logging.SimpleFormatter;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/31  11:42 31
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 自定义的 类型转换器
 */
public class String2DateConverter implements Converter<String, Date> {

    static List<SimpleDateFormat> sdfs = new ArrayList();
    static{
        sdfs.add(new SimpleDateFormat("yyyy-MM-dd"));
        sdfs.add(new SimpleDateFormat("yyyy/MM/dd"));
        sdfs.add(new SimpleDateFormat("yyyy-MM-dd"));
        sdfs.add(new SimpleDateFormat("yyyy/MM/dd HH:mm:ss"));
        sdfs.add(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));

    }


    @Override
    public Date convert(String s) {
        Date date = null;

        for (SimpleDateFormat sdf : sdfs) {
            try {
                date = sdf.parse(s);
                return date;

            } catch (ParseException e) {
                e.printStackTrace();
            }
        }

        return null;
    }
}
```

7.9.2. 给处理器适配器注入converService**

```
<mvc:annotation-driven conversion-service="conversionService"  />


<!-- 自定义的参数转换器  配置各种转换器 有默认值  String- stirng     stirng- Double ....
    -->
<bean id="conversionService"
 class="org.springframework.format.support.FormattingConversionServiceFactoryBean
">

    <property name="converters">
        <list>
            <!--内部的Bean声明-->
            <bean id="string2DateConverter"
class="com.neuedu.converter.String2DateConverter"/>
        </list>
    </property>
</bean>
```

8. 数据校验

校验的意义



前端校验框架比较多 参考（）

https://www.runoob.com/jquery/jquery-plugin-validate.html

8.1. Spring整合Hibernate-validation校验

8.1.1. pom.xml导入校验jar文件

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-validator -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.1.0.Final</version>
</dependency>
```

8.1.2. 配置校验器

让spring容器管理校验器

```xml
<!--声明校验器-->
    <bean id="validation" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
        <!--注入校验器的实现规则-->
        <property name="providerClass" value="org.hibernate.validator.HibernateValidator"></property>

        <!--校验失败的错误消息  文件中读取-->
        <property name="validationMessageSource" ref="validationMessageSource"/>
    </bean>

<!--     使用此类ReloadableResourceBundleMessageSource加载属性文件-->
    <bean id="validationMessageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>classpath:ValidationErrorMess</value>
            </list>
        </property>
        <!--原始文件的编码-->
        <property name="defaultEncoding" value="utf-8"/>

        <!--读取文件的编码-->
        <property name="fileEncodings" value="utf-8"/>

        <!--设置最大缓存时间 2000毫秒之后重新加载配置文件-->
<!--     <property name="cacheMillis" value="2000"/> -->
        <property name="cacheSeconds" value="2"/>

    </bean>
```

8.1.3. 校验器注入到处理器适配器中

```xml
<!--
    validator 属性的作用 将声明的校验器Bean注入到 处理器适配器 HandlerApapter
-->
<mvc:annotation-driven validator="validation" />
```

8.1.4. 添加校验规则

控制器接受参数（需要校验的参数）,例如接受用户User信息

1. @Null 被注释的元素必须为 null
2. @NotNull 被注释的元素必须不为 null
3. @AssertTrue 被注释的元素必须为 true
4. @AssertFalse 被注释的元素必须为 false
5. @Min(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
6. @Max(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
7. @DecimalMin(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
8. @DecimalMax(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
9. @Size(max=, min=) 被注释的元素的大小必须在指定的范围内

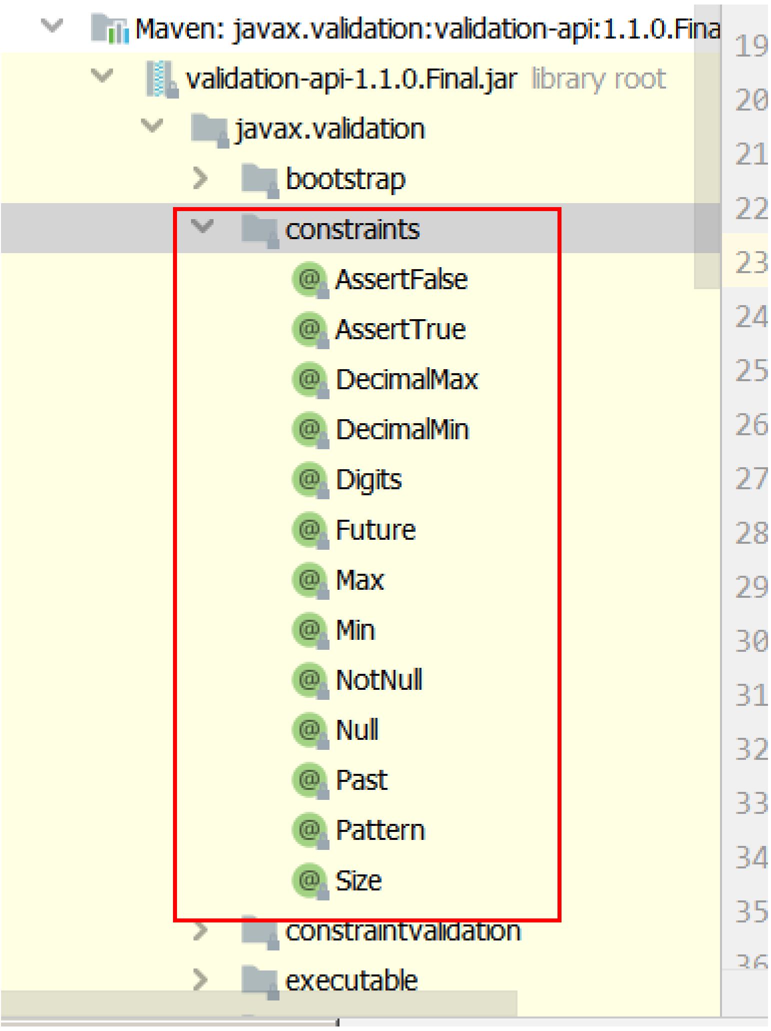
10. @Digits(integer, fraction) 被注释的元素必须是一个数字，其值必须在可接受的范围内
11. @Past 被注释的元素必须是一个过去的日期
12. @Future 被注释的元素必须是一个将来的日期
13. @Pattern(regex=,flag=) 被注释的元素必须符合指定的正则表达式

8.1.4.1. 控制的方法

```java
package com.neuedu.controller;

import com.neuedu.entity.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/30  11:27 30
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 校验规则的控制器
 */
@Controller
public class ValidatorContoller {


    /**
     * 1 写完Controller 方法测试，路径： http://127.0.0.1:8080/springmvc/validator1?username=abc&password=123456&amount=99
     * @param user
     * @param response
     * @throws IOException
     */
    @RequestMapping("/validator1")
    public void validator1(User user , HttpServletResponse response) throws IOException {


        response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
        PrintWriter out = response.getWriter();

        out.println("<div style='border:1px solid blue'>通过校验的用户参数:"+
                user
                +"</div>");

        out.flush();
        out.close();
    }
}
```

8.1.4.2. 在User对象中添加校验规则

在需要校验的字段上添加校验规则注解，如@Size



- Maven: javax.validation:validation-api:1.1.0.Fina
  - validation-api-1.1.0.Final.jar library root
    - javax.validation
      - bootstrap
      - constraints
        - AssertFalse
        - AssertTrue
        - DecimalMax
        - DecimalMin
        - Digits
        - Future
        - Max
        - Min
        - NotNull
        - Null
        - Past
        - Pattern
        - Size
      - constraintvalidation
      - executable

```java
package com.neuedu.entity;

import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/27  14:47 27
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * http://127.0.0.1:8080/context?username=aaa&password=xxx
 * 描述     :
 */

public class User {


    //用户名最短5 最长10
    @Size(min=5,max=10,message ="{mess.validate.user_length}")
    private String username;

    @Min(value = 6,message = "密码不能小于6")
    private String password;
    private Double amount;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", amount=" + amount +
                '}';
    }
}
```

8.1.5. 错误信息文件

```
mess.validate.user_length=用户名长度不正确，请检查
```

8.1.6. 捕获错误信息

```java
/**
 * 1 写完Controller 方法测试，路径： http://127.0.0.1:8080/springmvc/validator1?username=abc&password=123456&amount=99
 * 2 在需要校验单 参数上添加@Validated注解 ， 并且在此参数后面(紧挨着校验的参数)添加BindingReuslt 参数用于接收异常消息
 * @param user
 * @param response
 * @throws IOException
 */
@RequestMapping("/validator1")
public void validator1(@Validated User user ,BindingResult bindingResult , HttpServletResponse response) throws IOException {

    response.setContentType("text/html;charset=utf-8");//响应html，格式utf8
    PrintWriter out = response.getWriter();

```

```
        //当产生异常8
        if(bindingResult.getErrorCount()>0){
            //获取所有异常对象
            List<ObjectError> errs = bindingResult.getAllErrors();
            String errStr = "";
            for (ObjectError err : errs) {
                errStr += err.getDefaultMessage()+",";
            }

            out.println("<div style='border:1px solid blue'>校验不通过:"+
                    errStr
                    +"</div>");
        }else{  //没有产生异常的
            out.println("<div style='border:1px solid blue'>通过校验的用户参数:"+
                    user
                    +"</div>");
        }


        out.flush();
        out.close();
    }
```

8.1.7. 显示错误信息

```
List<ObjectError> errs = bindingResult.getAllErrors();
String errStr = "";
for (ObjectError err : errs) {
    errStr += err.getDefaultMessage()+",";
}
```

9. 数据回显

9.1. Form表单（Jsp页面）

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/3/30
  Time: 14:51
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户的添加</title>
</head>
<body>

<form action="${pageContext.request.contextPath}/validator3">

    ${err}

    <!--模拟都是 30-字段 -->
    <table>
        <tr>
            <td>用户名字</td>
            <td><input type="text" name="username" value="${user.username}"></td>
        </tr>
        <tr>
            <td>常用地址</td>
            <td><input type="text" name="password" value="${user.password}"></td>
        </tr>
        <tr>
            <td>账户余额</td>
            <td><input type="text" name="amount" value="${user.amount}"></td>
        </tr>
        <tr>
            <td colspan="2"><input type="submit" value="用户注册" onclick="func()"></td>
        </tr>
    </table>
</form>

</body>
</html>
```

9.2. 自定义代码通过request将模型设置为属性

```
* 1 写完Controller 方法测试，路径： http://127.0.0.1:8080/springmvc/validator1?username=abc&password=123456&amount=99
     * 2 在需要校验单 参数上添加@Validated注解 ，并且在此参数后面(紧挨着校验的参数)添加BindingReuslt 参数用于接收异常消息
     *
     * 需要校验 用户名
     *
     *  返回void没有经过视图解析器
     *
     * @param user
     * @param response
     * @throws IOException
     */
    @RequestMapping("/validator3")
    public void saveUser(
                           @Validated(value = ValidateGroupLogin.class )
                                     User user ,
                           BindingResult bindingResult , HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        //当产生异常
        if(bindingResult.getErrorCount()>0){
            //获取所有异常对象
            List<ObjectError> errs = bindingResult.getAllErrors();
            String errStr = "";
            for (ObjectError err : errs) {
                errStr += err.getDefaultMessage()+",";
            }
            request.setAttribute("err",errStr);
            request.setAttribute("user",user);
```

```
      //回到添加页面 ,将异常消息返回到添加页面，让用户重新修改
      request.getRequestDispatcher("/review/user_add.jsp").forward(request,response);
    }else{
      //保存到数据库
      response.setContentType("text/html;charset=utf-8");
      //跳转到一个成功页面
      PrintWriter out = response.getWriter();
      out.println("<div style='border:1px solid blue'>通过校验的用户参数:"+
          user
          +"</div>");
      out.flush();
      out.close();

    }

  }
```

9.3. 使用@ModelAttribute注解，将参数设置为属性

属性的可以为参数类名的首字母变小写如参数（User user）key 为“user”，（User peron）还是“user”作为key

```
/**
 * 1 写完Controller 方法测试，路径： http://127.0.0.1:8080/springmvc/validator1?username=abc&password=123456&amount=99
 * 2 在需要校验单 参数上添加@Validated注解 ，并且在此参数后面(紧挨着校验的参数)添加BindingReuslt 参数用于接收异常消息
 *
 * 需要校验 用户名
 *
 *  返回void没有经过视图解析器
 *        @ModelAttribute("mysuer")     === model.addAttribute("mysuer",user);
 *        @ModelAttribute  User user    == model.addAttribute("user",user);
 * @param user
 * @param response
 * @throws IOException
 */
@RequestMapping("/validator3")
public String saveUser(
                        @ModelAttribute("mysuer")
                       @Validated(value = ValidateGroupLogin.class )
                                 User user ,
                       BindingResult bindingResult , HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
  //当产生异常
  if(bindingResult.getErrorCount()>0){
    //获取所有异常对象
    List<ObjectError> errs = bindingResult.getAllErrors();
    String errStr = "";
    for (ObjectError err : errs) {
      errStr += err.getDefaultMessage()+",";
    }
```

```
        request.setAttribute("err",errStr);
        return "foward:/review/user_add.jsp";
    }else{
        //保存到数据库
        response.setContentType("text/html;charset=utf-8");
        //跳转到一个成功页面
        PrintWriter out = response.getWriter();
        out.println("<div style='border:1px solid blue'>通过校验的用户参数:"+
                user
                +"</div>");
        out.flush();
        out.close();

    }
    return null;

}
```

10. 异常处理

10.1. 定义自定义异常类

非必须

```
/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/31  9:57 31
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 自定义异常类
 */
public class BusinessException extends  Exception {

    public BusinessException(String message) {
        super(message);
    }
}
```

10.2. 定义异常处理器

当产生异常时能够进行处理，比如说跳转到一个友好错误界面

```
import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/31  9:59 31
 * author  : jshand-root
```

```java
 * site    :  http://314649444.iteye.com
 * 描述     : 自定义的异常处理器
 */

public class MyExceptionResolver implements HandlerExceptionResolver {

    /**
     * 处理异常的方法 resolveException
     * @param request
     * @param response
     * @param handler  控制器
     * @param ex      产生的异常对象
     * @return
     */
    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {

        request.setAttribute("msg",ex.getMessage());

        ModelAndView mav = new ModelAndView();

        mav.setViewName("/error/500.jsp");

        return mav;
    }
}
```

10.3. 配置异常处理器

将定义好的异常处理器配置到iOC容器中。

10.4. 编写异常信息文件

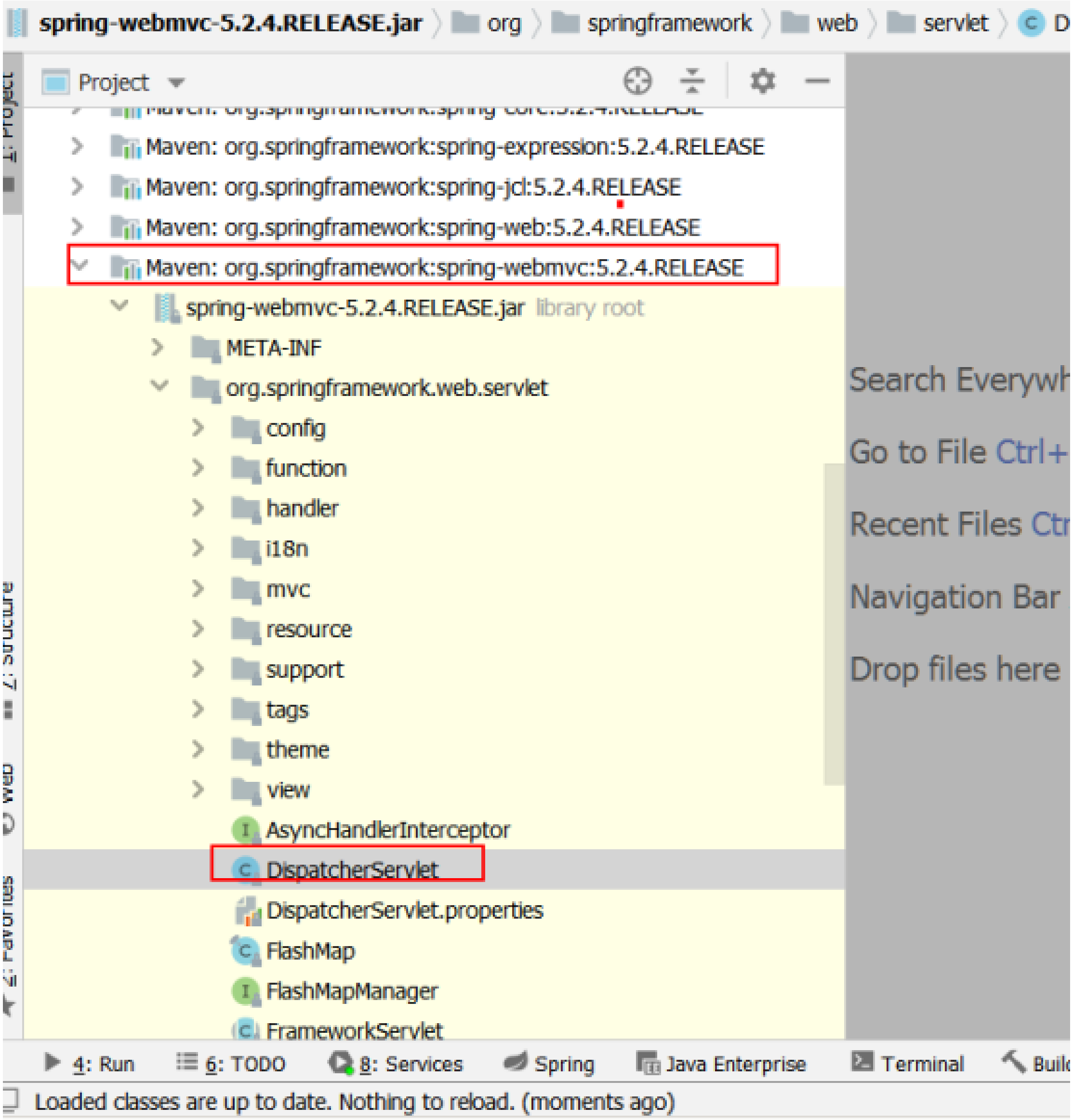
编写一个友好错误界面

10.5. 异常类应用

测试使用异常处理器

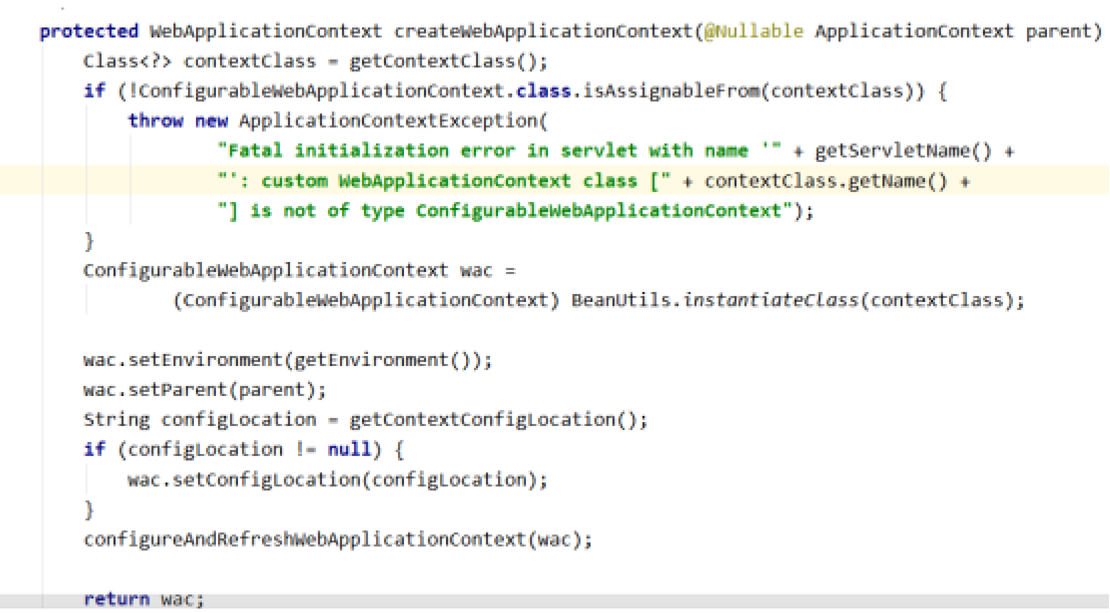
11. DispatcherServlet源码分析

11.1. 所在的包：


spring-webmvc-5.2.4.RELEASE.jar 〉 org 〉 springframework 〉 web 〉 servlet 〉 D
Project
Maven: org.springframework:spring-expression:5.2.4.RELEASE
Maven: org.springframework:spring-jcl:5.2.4.RELEASE
Maven: org.springframework:spring-web:5.2.4.RELEASE
Maven: org.springframework:spring-webmvc:5.2.4.RELEASE
spring-webmvc-5.2.4.RELEASE.jar library root
META-INF
org.springframework.web.servlet
config
function
handler
i18n
mvc
resource
support
tags
theme
view
AsyncHandlerInterceptor
DispatcherServlet
DispatcherServlet.properties
FlashMap
FlashMapManager
FrameworkServlet
Search Everywh
Go to File Ctrl+
Recent Files Ctr
Navigation Bar
Drop files here
4: Run
6: TODO
8: Services
Spring
Java Enterprise
Terminal
Buil

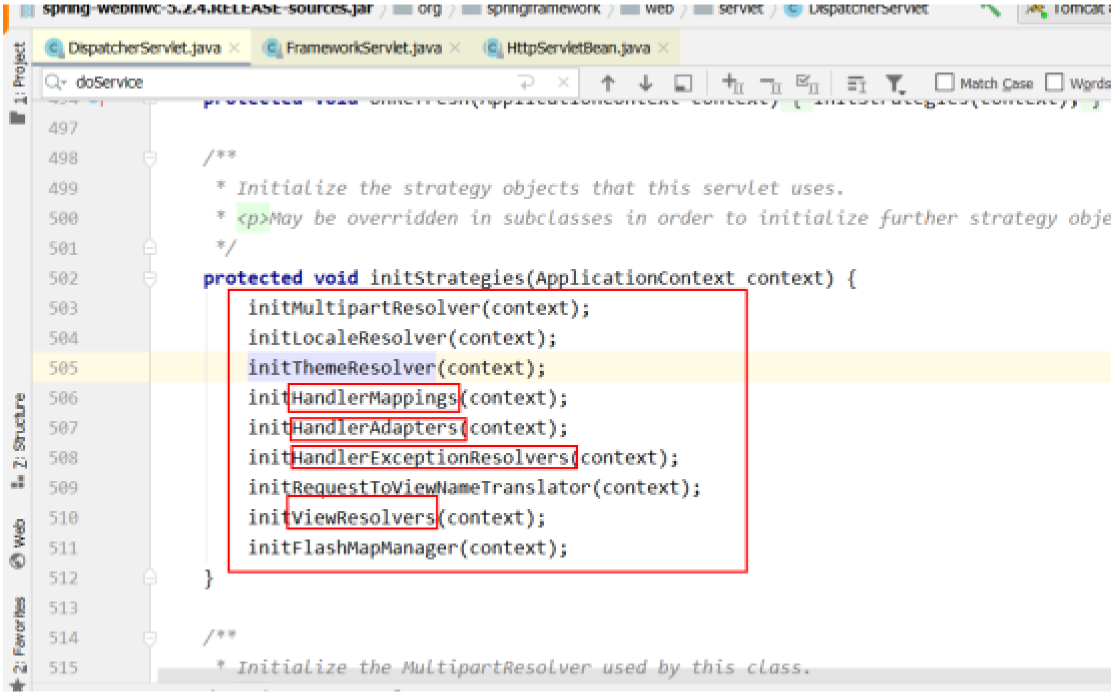
Loaded classes are up to date. Nothing to reload. (moments ago)

11.2. 创建 *Web类型的ApplicationContext*



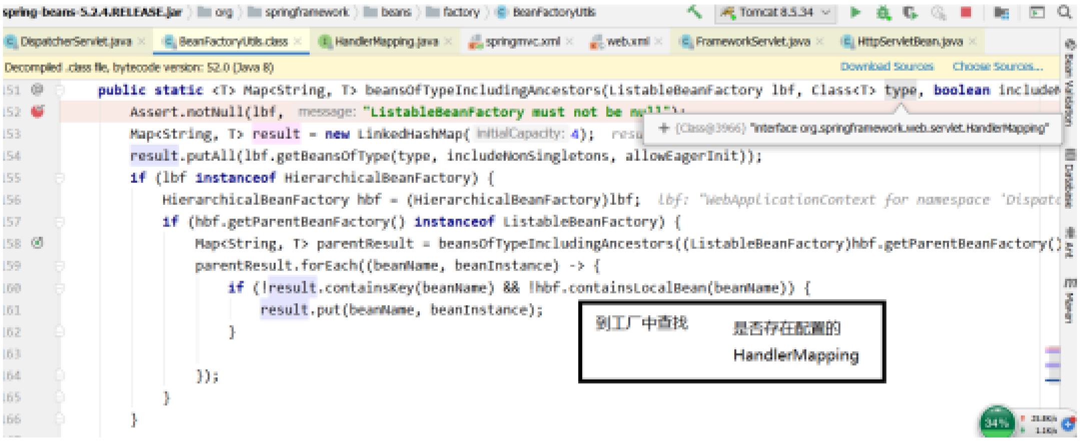
```java
protected WebApplicationContext createWebApplicationContext(@Nullable ApplicationContext parent)
    Class<?> contextClass = getContextClass();
    if (!ConfigurableWebApplicationContext.class.isAssignableFrom(contextClass)) {
        throw new ApplicationContextException(
                "Fatal initialization error in servlet with name '" + getServletName() +
                "': custom WebApplicationContext class [" + contextClass.getName() +
                "] is not of type ConfigurableWebApplicationContext");
    }
    ConfigurableWebApplicationContext wac =
            (ConfigurableWebApplicationContext) BeanUtils.instantiateClass(contextClass);

    wac.setEnvironment(getEnvironment());
    wac.setParent(parent);
    String configLocation = getContextConfigLocation();
    if (configLocation != null) {
        wac.setConfigLocation(configLocation);
    }
    configureAndRefreshWebApplicationContext(wac);

    return wac;
```

用户初始化策略



```java
    protected void onRefresh(ApplicationContext context) { initStrategies(context); }

    /**
     * Initialize the strategy objects that this servlet uses.
     * <p>May be overridden in subclasses in order to initialize further strategy obje
     */
    protected void initStrategies(ApplicationContext context) {
        initMultipartResolver(context);
        initLocaleResolver(context);
        initThemeResolver(context);
        initHandlerMappings(context);
        initHandlerAdapters(context);
        initHandlerExceptionResolvers(context);
        initRequestToViewNameTranslator(context);
        initViewResolvers(context);
        initFlashMapManager(context);
    }

    /**
     * Initialize the MultipartResolver used by this class.
```

11.3. 初始化以HandlerMapping为例

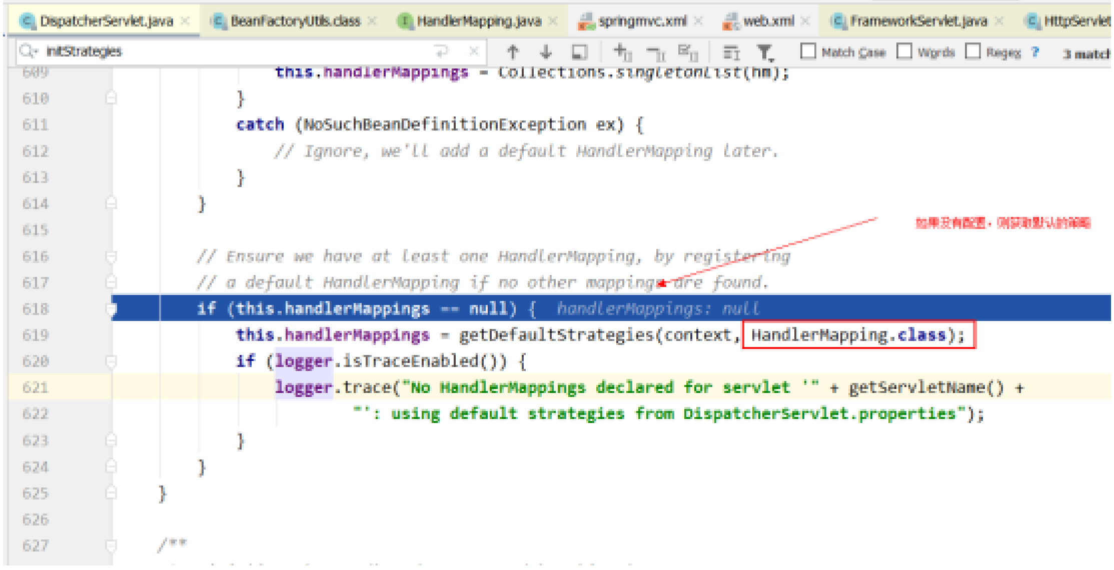


Decompiled .class file, bytecode version: 52.0 (Java 8)

```java
    public static <T> Map<String, T> beansOfTypeIncludingAncestors(ListableBeanFactory lbf, Class<T> type, boolean include
        Assert.notNull(lbf, message: "ListableBeanFactory must not be null");
        Map<String, T> result = new LinkedHashMap( initialCapacity: 4);  resu
        result.putAll(lbf.getBeansOfType(type, includeNonSingletons, allowEagerInit));
        if (lbf instanceof HierarchicalBeanFactory) {
            HierarchicalBeanFactory hbf = (HierarchicalBeanFactory)lbf;  lbf: "WebApplicationContext for namespace 'Dispatc
            if (hbf.getParentBeanFactory() instanceof ListableBeanFactory) {
                Map<String, T> parentResult = beansOfTypeIncludingAncestors((ListableBeanFactory)hbf.getParentBeanFactory()
                parentResult.forEach((beanName, beanInstance) -> {
                    if (!result.containsKey(beanName) && !hbf.containsLocalBean(beanName)) {
                        result.put(beanName, beanInstance);
                    }

                });
            }
        }
```

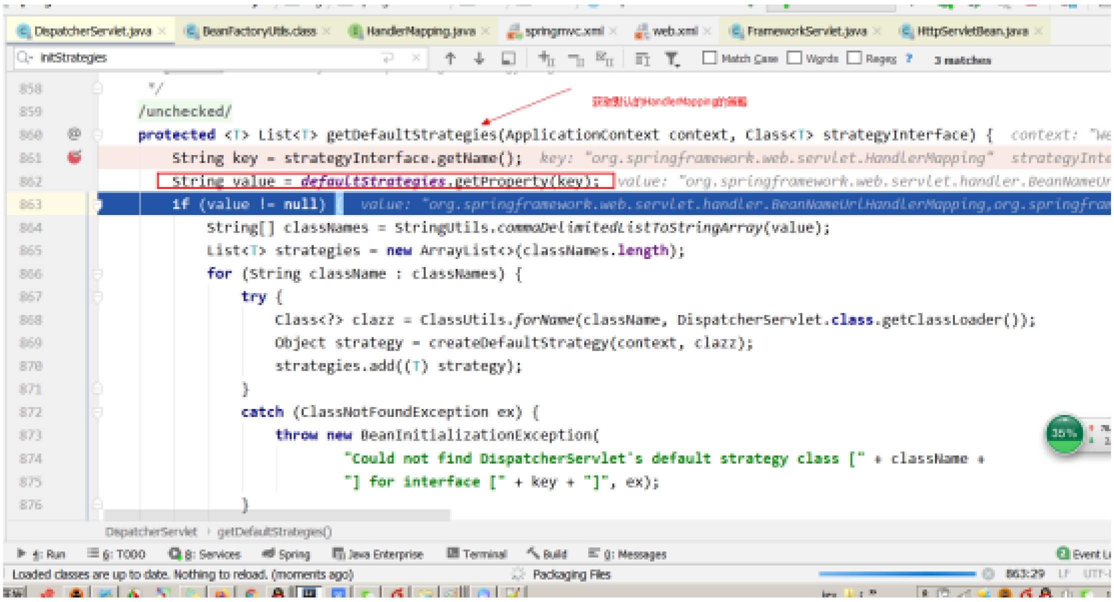
+ {Class@3966} "interface org.springframework.web.servlet.HandlerMapping"

到工厂中查找 是否存在配置的 HandlerMapping



```java
                    this.handlerMappings = Collections.singletonList(hm);
                }
            }
            catch (NoSuchBeanDefinitionException ex) {
                // Ignore, we'll add a default HandlerMapping later.
            }
        }

        // Ensure we have at least one HandlerMapping, by registering
        // a default HandlerMapping if no other mappings are found.
        if (this.handlerMappings == null) {  handlerMappings: null
            this.handlerMappings = getDefaultStrategies(context, HandlerMapping.class);
            if (logger.isTraceEnabled()) {
                logger.trace("No HandlerMappings declared for servlet '" + getServletName() +
                        "': using default strategies from DispatcherServlet.properties");
            }
        }
    }

    /**
```

如果没有配置，则获取默认的策略

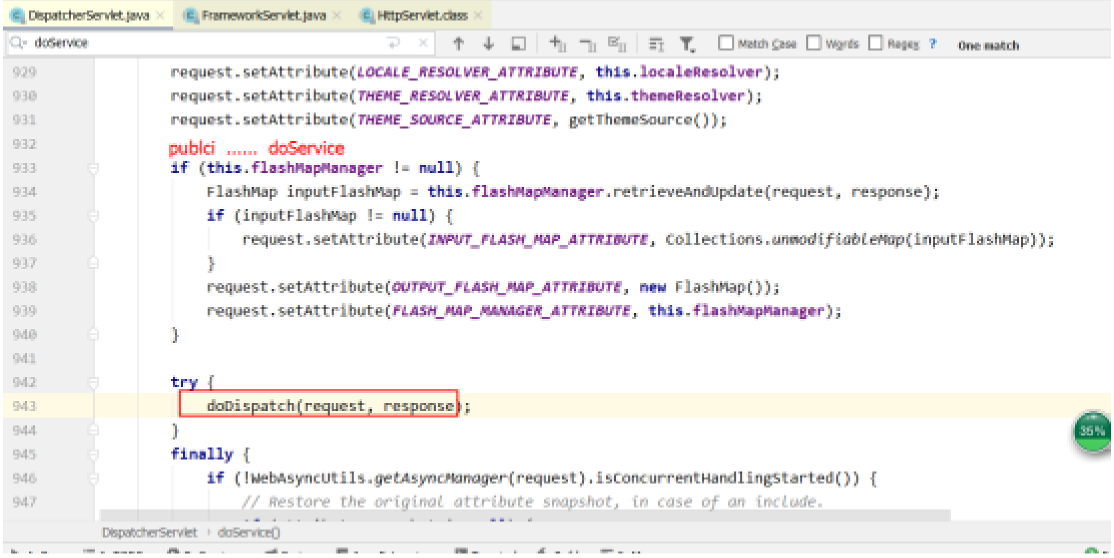


获取默认的HandlerMapping的策略

```
    */
   /unchecked/
   protected <T> List<T> getDefaultStrategies(ApplicationContext context, Class<T> strategyInterface) {   context: "We
       String key = strategyInterface.getName();   key: "org.springframework.web.servlet.HandlerMapping"   strategyInte
       String value = defaultStrategies.getProperty(key);  value: "org.springframework.web.servlet.handler.BeanNameUr
       if (value != null) {   value: "org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping,org.springfram
           String[] classNames = StringUtils.commaDelimitedListToStringArray(value);
           List<T> strategies = new ArrayList<>(classNames.length);
           for (String className : classNames) {
               try {
                   Class<?> clazz = ClassUtils.forName(className, DispatcherServlet.class.getClassLoader());
                   Object strategy = createDefaultStrategy(context, clazz);
                   strategies.add((T) strategy);
               }
               catch (ClassNotFoundException ex) {
                   throw new BeanInitializationException(
                           "Could not find DispatcherServlet's default strategy class [" + className +
                           "] for interface [" + key + "]", ex);
               }
```

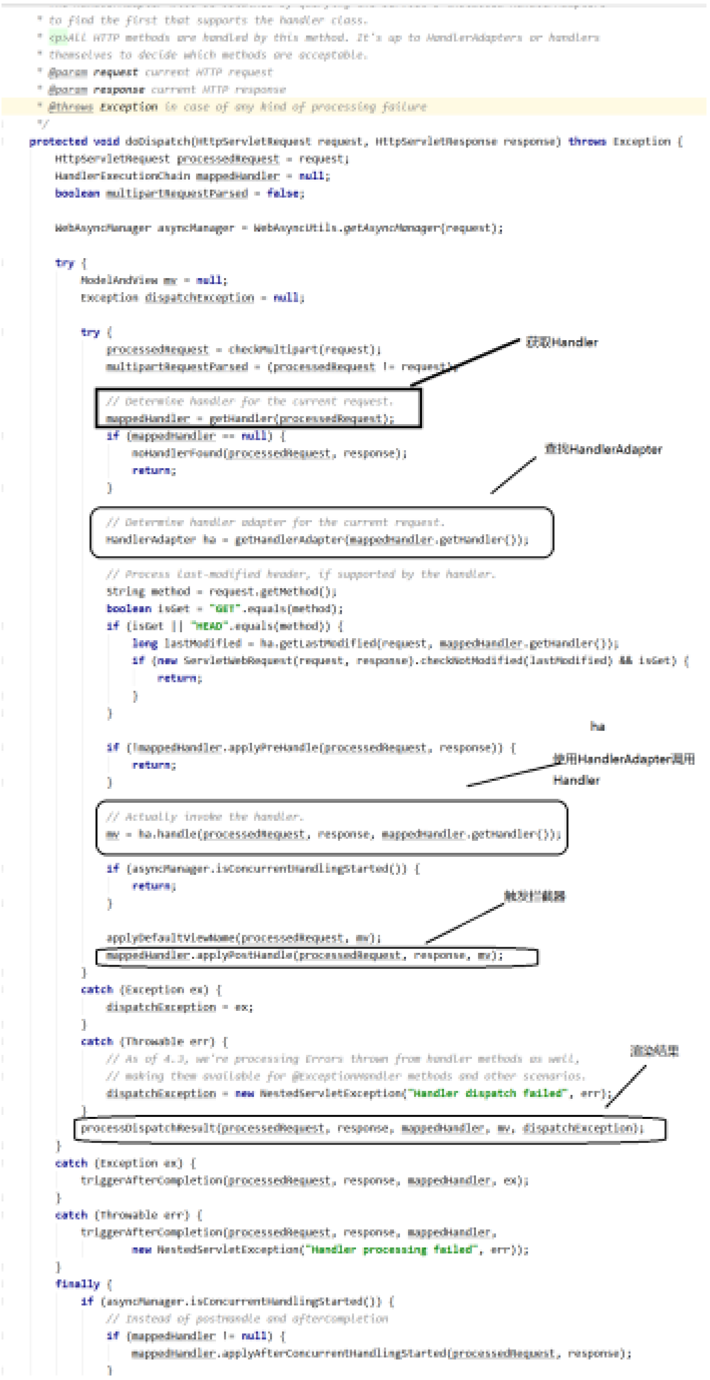
Loaded classes are up to date. Nothing to reload. (moments ago)

11.4. 处理请求的逻辑



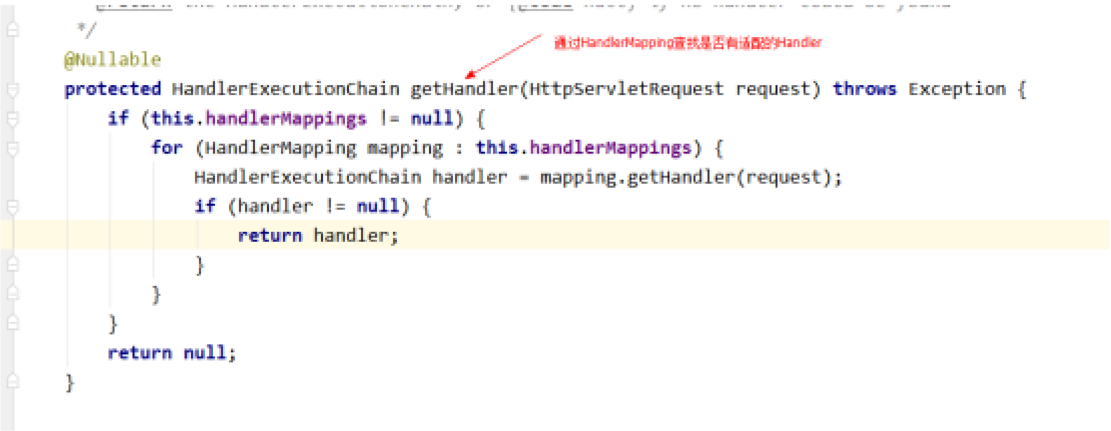
```
        request.setAttribute(LOCALE_RESOLVER_ATTRIBUTE, this.localeResolver);
        request.setAttribute(THEME_RESOLVER_ATTRIBUTE, this.themeResolver);
        request.setAttribute(THEME_SOURCE_ATTRIBUTE, getThemeSource());
        publci ...... doService
        if (this.flashMapManager != null) {
            FlashMap inputFlashMap = this.flashMapManager.retrieveAndUpdate(request, response);
            if (inputFlashMap != null) {
                request.setAttribute(INPUT_FLASH_MAP_ATTRIBUTE, Collections.unmodifiableMap(inputFlashMap));
            }
            request.setAttribute(OUTPUT_FLASH_MAP_ATTRIBUTE, new FlashMap());
            request.setAttribute(FLASH_MAP_MANAGER_ATTRIBUTE, this.flashMapManager);
        }

        try {
            doDispatch(request, response);
        }
        finally {
            if (!WebAsyncUtils.getAsyncManager(request).isConcurrentHandlingStarted()) {
                // Restore the original attribute snapshot, in case of an include.
```

11.5. doDispatch方法（外层的大方法）



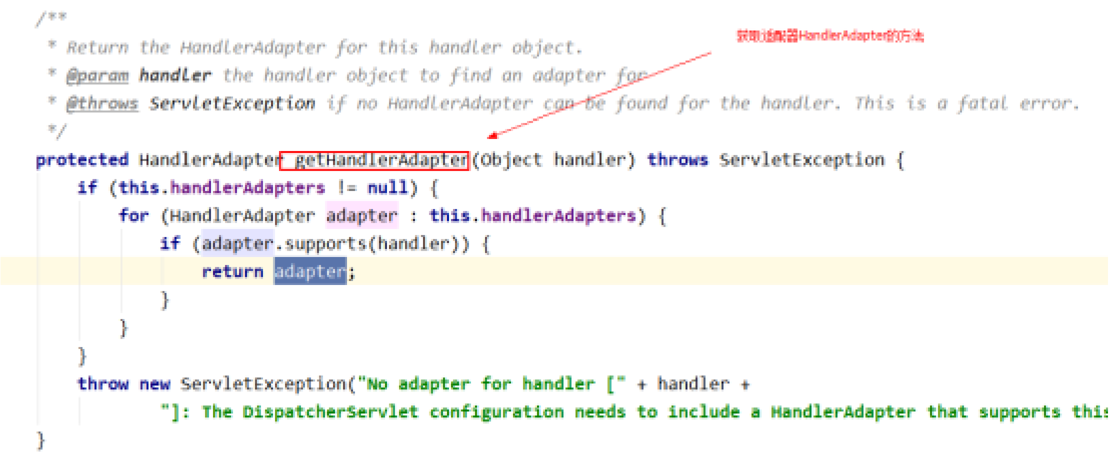
```java
 * to find the first that supports the handler class.
 * <p>All HTTP methods are handled by this method. It's up to HandlerAdapters or handlers
 * themselves to decide which methods are acceptable.
 * @param request current HTTP request
 * @param response current HTTP response
 * @throws Exception in case of any kind of processing failure
 */
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {
    HttpServletRequest processedRequest = request;
    HandlerExecutionChain mappedHandler = null;
    boolean multipartRequestParsed = false;

    WebAsyncManager asyncManager = WebAsyncUtils.getAsyncManager(request);

    try {
        ModelAndView mv = null;
        Exception dispatchException = null;

        try {
            processedRequest = checkMultipart(request);
            multipartRequestParsed = (processedRequest != request);

            // Determine handler for the current request.
            mappedHandler = getHandler(processedRequest);
            if (mappedHandler == null) {
                noHandlerFound(processedRequest, response);
                return;
            }

            // Determine handler adapter for the current request.
            HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());

            // Process last-modified header, if supported by the handler.
            String method = request.getMethod();
            boolean isGet = "GET".equals(method);
            if (isGet || "HEAD".equals(method)) {
                long lastModified = ha.getLastModified(request, mappedHandler.getHandler());
                if (new ServletWebRequest(request, response).checkNotModified(lastModified) && isGet) {
                    return;
                }
            }

            if (!mappedHandler.applyPreHandle(processedRequest, response)) {
                return;
            }

            // Actually invoke the handler.
            mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

            if (asyncManager.isConcurrentHandlingStarted()) {
                return;
            }

            applyDefaultViewName(processedRequest, mv);
            mappedHandler.applyPostHandle(processedRequest, response, mv);
        }
        catch (Exception ex) {
            dispatchException = ex;
        }
        catch (Throwable err) {
            // As of 4.3, we're processing Errors thrown from handler methods as well,
            // making them available for @ExceptionHandler methods and other scenarios.
            dispatchException = new NestedServletException("Handler dispatch failed", err);
        }
        processDispatchResult(processedRequest, response, mappedHandler, mv, dispatchException);
    }
    catch (Exception ex) {
        triggerAfterCompletion(processedRequest, response, mappedHandler, ex);
    }
    catch (Throwable err) {
        triggerAfterCompletion(processedRequest, response, mappedHandler,
                new NestedServletException("Handler processing failed", err));
    }
    finally {
        if (asyncManager.isConcurrentHandlingStarted()) {
            // Instead of postHandle and afterCompletion
            if (mappedHandler != null) {
                mappedHandler.applyAfterConcurrentHandlingStarted(processedRequest, response);
            }
```

11.5.1. 获取Handler的方法



```java
 */
@Nullable
protected HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {
    if (this.handlerMappings != null) {
        for (HandlerMapping mapping : this.handlerMappings) {
            HandlerExecutionChain handler = mapping.getHandler(request);
            if (handler != null) {
                return handler;
            }
        }
    }
    return null;
}
```

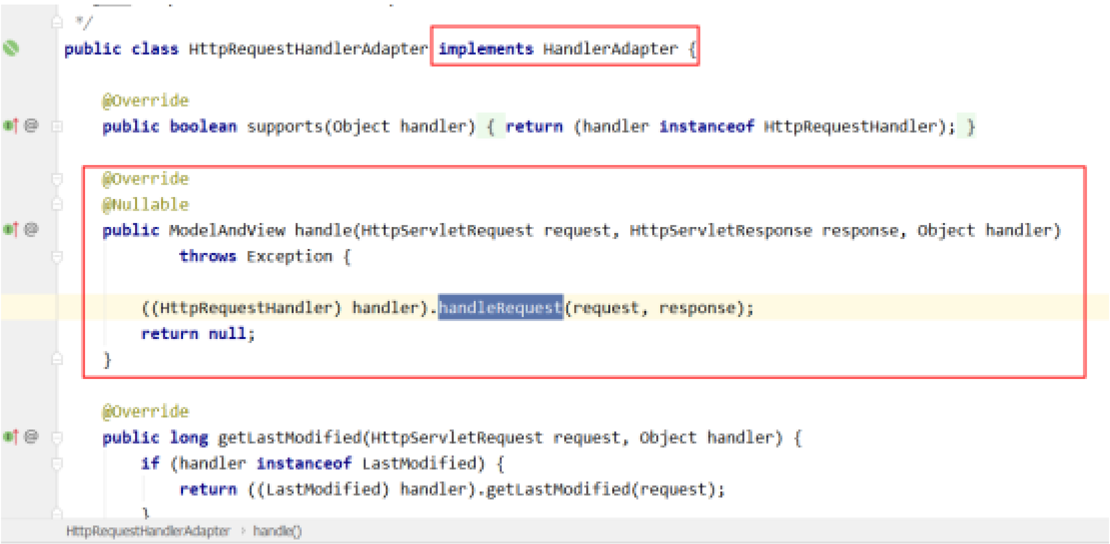
11.5.2. 获取适配器



```java
/**
 * Return the HandlerAdapter for this handler object.
 * @param handler the handler object to find an adapter for
 * @throws ServletException if no HandlerAdapter can be found for the handler. This is a fatal error.
 */
protected HandlerAdapter getHandlerAdapter(Object handler) throws ServletException {
    if (this.handlerAdapters != null) {
        for (HandlerAdapter adapter : this.handlerAdapters) {
            if (adapter.supports(handler)) {
                return adapter;
            }
        }
    }
    throw new ServletException("No adapter for handler [" + handler +
            "]: The DispatcherServlet configuration needs to include a HandlerAdapter that supports this
}
```

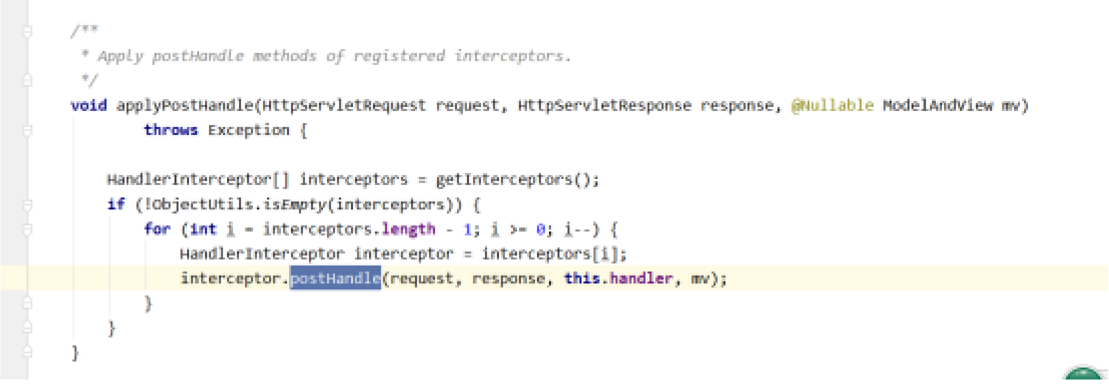
获取适配器HandlerAdapter的方法

11.5.3. 适配器执行Handler（以他为例HttpRequestHandlerAdapter



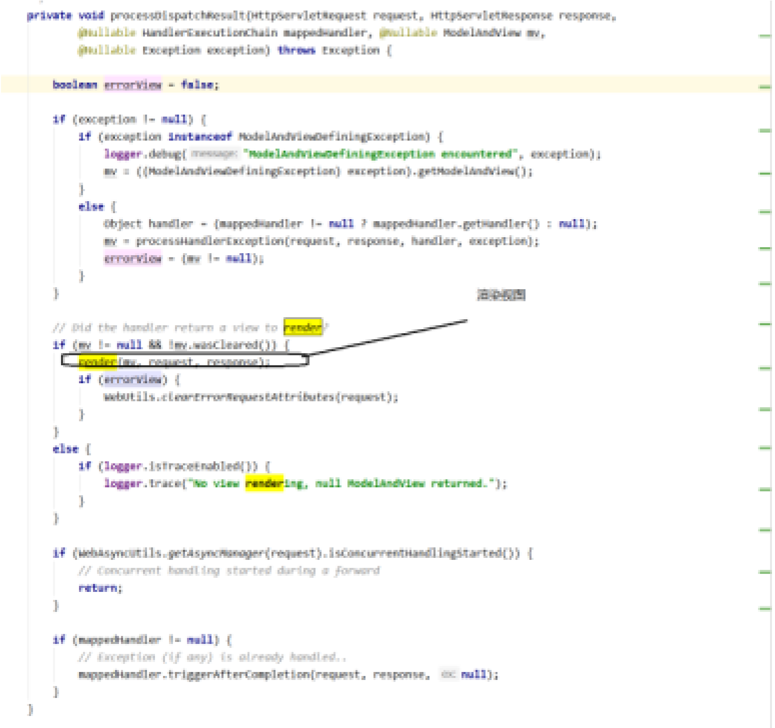
```java
 */
public class HttpRequestHandlerAdapter implements HandlerAdapter {

    @Override
    public boolean supports(Object handler) { return (handler instanceof HttpRequestHandler); }

    @Override
    @Nullable
    public ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Object handler)
            throws Exception {

        ((HttpRequestHandler) handler).handleRequest(request, response);
        return null;
    }

    @Override
    public long getLastModified(HttpServletRequest request, Object handler) {
        if (handler instanceof LastModified) {
            return ((LastModified) handler).getLastModified(request);
        }
```

11.5.4. 执行拦截器



```java
/**
 * Apply postHandle methods of registered interceptors.
 */
void applyPostHandle(HttpServletRequest request, HttpServletResponse response, @Nullable ModelAndView mv)
        throws Exception {

    HandlerInterceptor[] interceptors = getInterceptors();
    if (!ObjectUtils.isEmpty(interceptors)) {
        for (int i = interceptors.length - 1; i >= 0; i--) {
            HandlerInterceptor interceptor = interceptors[i];
            interceptor.postHandle(request, response, this.handler, mv);
        }
    }
}
```

11.5.5. 渲染视图



```java
private void processDispatchResult(HttpServletRequest request, HttpServletResponse response,
        @Nullable HandlerExecutionChain mappedHandler, @Nullable ModelAndView mv,
        @Nullable Exception exception) throws Exception {

    boolean errorView = false;

    if (exception != null) {
        if (exception instanceof ModelAndViewDefiningException) {
            logger.debug( message: "ModelAndViewDefiningException encountered", exception);
            mv = ((ModelAndViewDefiningException) exception).getModelAndView();
        }
        else {
            Object handler = (mappedHandler != null ? mappedHandler.getHandler() : null);
            mv = processHandlerException(request, response, handler, exception);
            errorView = (mv != null);
        }
    }

    // Did the handler return a view to render?
    if (mv != null && !mv.wasCleared()) {
        render(mv, request, response);
        if (errorView) {
            WebUtils.clearErrorRequestAttributes(request);
        }
    }
    else {
        if (logger.isTraceEnabled()) {
            logger.trace("No view rendering, null ModelAndView returned.");
        }
    }

    if (WebAsyncUtils.getAsyncManager(request).isConcurrentHandlingStarted()) {
        // Concurrent handling started during a forward
        return;
    }

    if (mappedHandler != null) {
        // Exception (if any) is already handled..
        mappedHandler.triggerAfterCompletion(request, response,  ex: null);
    }
}
```

渲染视图

12. 拦截器

12.1. 定义类实现拦截器接口

需要实现，并实现抽象方法HandlerInterceptor

```java
package com.neuedu.interceptor;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/3/31  13:47 31
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 登录的拦截器
 */
public class ValidateLoginInterceptor implements HandlerInterceptor {

    /**
     * 在控制器方法之前执行的
     * @param request
     * @param response
     * @param handler
     * @return   false：控制的方法不会就行执行，同时postHandle、afterCompletion 方法也都不会继续执行
     * @throws Exception
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        System.out.println("登录的拦截器:preHandle");
        return true;
    }

    /**
     * 在控制器方法之后执行 ,如果有异常不会执行
     * @param request
     * @param response
     * @param handler
     * @param modelAndView
     * @throws Exception
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("登录的拦截器:postHandle");

    }

    /**
     * 在控制器方法之后执行 有异常也正常的执行
     * @param request
     * @param response
     * @param handler
     * @param ex
     * @throws Exception
     */
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("登录的拦截器:afterCompletion");
    }
}
```

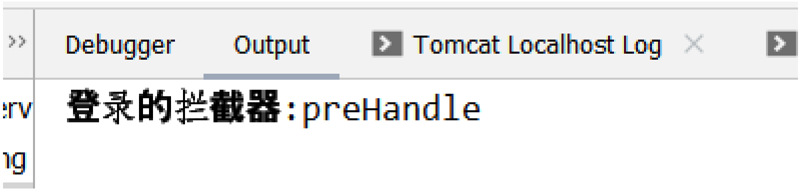
12.2. 将拦截器配置到具体的HandlerMapping上

```
<mvc:interceptors>
    <mvc:interceptor>
      <!--
        http://127.0.0.1:8080/springmvc/abc
        http://127.0.0.1:8080/springmvc/def  -->
       <!-- <mvc:mapping path="/*" />-->


        <!--
        包含子目录
        http://127.0.0.1:8080/springmvc/user/insert
        http://127.0.0.1:8080/springmvc/user/update
        -->
        <mvc:mapping path="/**" />
```

```
            <bean class="com.neuedu.interceptor.ValidateLoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>
```

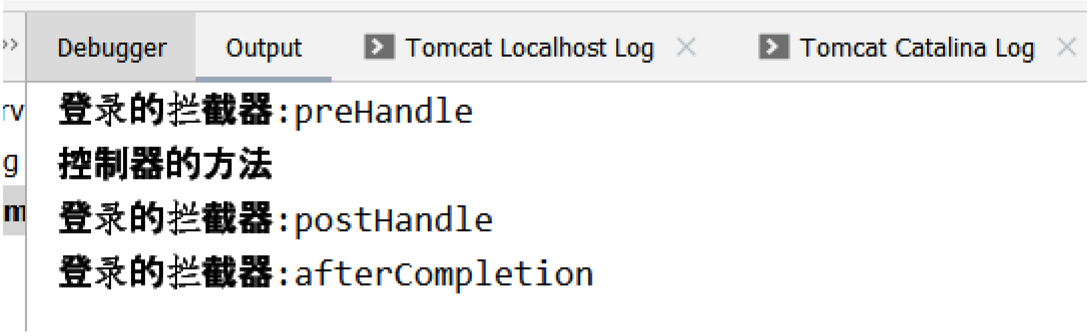
12.3. 测试

12.3.0.1. 单个拦截器并且preHandler方法返回false



```
登录的拦截器:preHandle
```

12.3.0.2. 单个拦截器并且preHandler方法返回true



```
登录的拦截器:preHandle
控制器的方法
登录的拦截器:postHandle
登录的拦截器:afterCompletion
```

12.3.1. preHandle

在控制器方法之前执行的,

返回结果

True：相当于是FIlter的放行

False：后续的拦截器方法（postHandle、afterCompletion），控制器方法都不执行

12.3.2. postHandle

如果控制器方法没有异常，则在方法之后执行postHandle，如果有异常则此方法不会执行。

12.3.3. afterCompletion

在控制器方法之后执行postHandle，无论是否存在异常。

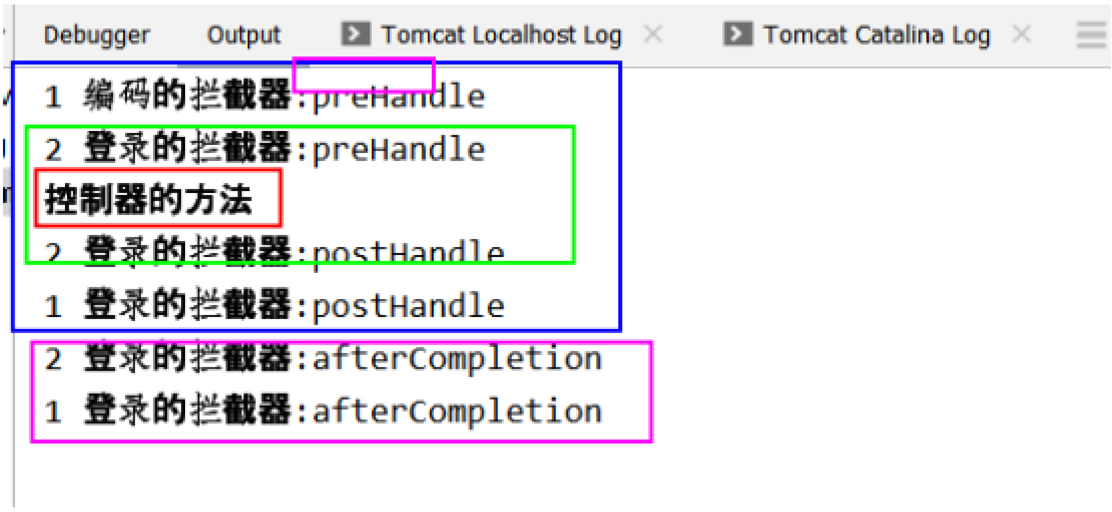
12.4. 多个拦截器

第二个拦截器

```
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```
/**
* 项目    :  spring-mvc-java1
* 创建时间 : 2020/3/31  13:47 31
* author  : jshand-root
* site    :  http://314649444.iteye.com
* 描述     : 设置中文编码的拦截器
*/
public class CharsetInterceptor implements HandlerInterceptor {

    /**
     * 在控制器方法之前执行的
     * @param request
     * @param response
     * @param handler
     * @return   false：控制的方法不会就行执行，同时postHandle、afterCompletion 方法也都不会继续执行
     * @throws Exception
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

        request.setCharacterEncoding("utf-8");
        response.setCharacterEncoding("utf-8");

        System.out.println("1 编码的拦截器:preHandle");
        return true;
    }

    /**
     * 在控制器方法之后执行 ,如果有异常不会执行
     * @param request
     * @param response
     * @param handler
     * @param modelAndView
     * @throws Exception
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("1 登录的拦截器:postHandle");

    }

    /**
     * 在控制器方法之后执行 有异常也正常的执行
     * @param request
     * @param response
     * @param handler
     * @param ex
     * @throws Exception
     */
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("1 登录的拦截器:afterCompletion");
    }
}
```

12.5. 多个拦截器preHandle方法返回true

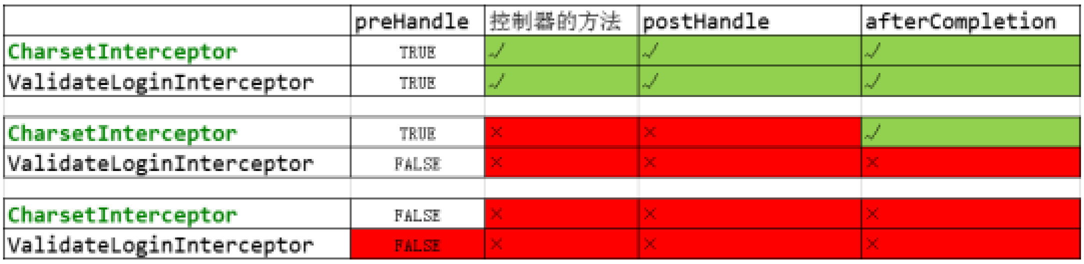


```
1 编码的拦截器:preHandle
2 登录的拦截器:preHandle
控制器的方法
2 登录的拦截器:postHandle
1 登录的拦截器:postHandle
2 登录的拦截器:afterCompletion
1 登录的拦截器:afterCompletion
```

12.6. 多拦截器的总结

当第一个拦截器返回true的时候，afterCompletion方法可以执行，第二个拦截器返回false后续的方法都不执行。只要有一个拦截器返回fasle则控制器的方法就不执行。

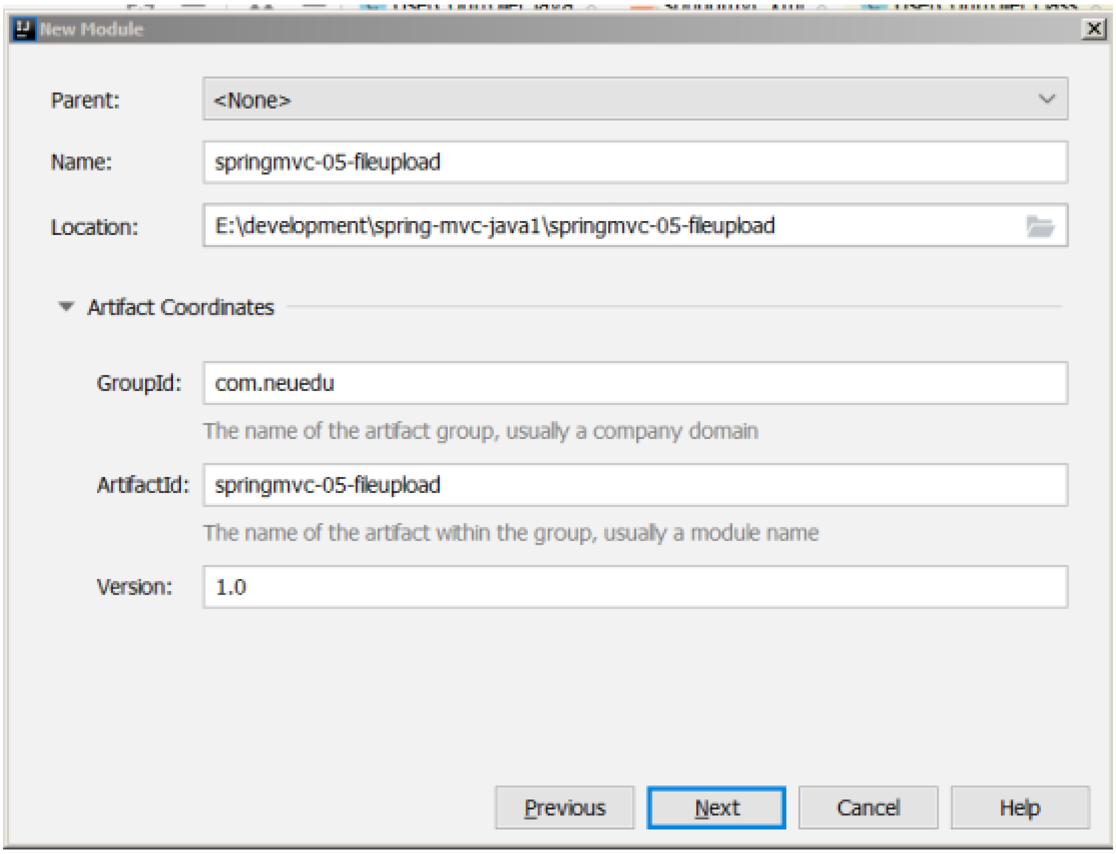
建议将必须要执行的拦截器前置（例如：无论是否登录成功都需要设置的编码的拦截器）



	preHandle	控制器的方法	postHandle	afterCompletion
CharsetInterceptor	TRUE	✓	✓	✓
ValidateLoginInterceptor	TRUE	✓	✓	✓
CharsetInterceptor	TRUE	✗	✗	✓
ValidateLoginInterceptor	FALSE	✗	✗	✗
CharsetInterceptor	FALSE	✗	✗	✗
ValidateLoginInterceptor	FALSE	✗	✗	✗

13. 文件的上传下载

13.1. 创建项目



Parent: <None>

Name: springmvc-05-fileupload

Location: E:\development\spring-mvc-java1\springmvc-05-fileupload

Artifact Coordinates

GroupId: com.neuedu

The name of the artifact group, usually a company domain

ArtifactId: springmvc-05-fileupload

The name of the artifact within the group, usually a module name

Version: 1.0

13.2. 添加依赖,修改pom.xml

添加common-fileupload(Apache)类库,从request中解析出文件内容

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.2.4.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.4</version>
</dependency>
```

13.3. 前端控制器(web.xml)

```xml
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <!--前端控制器-->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <!--默认的配置文件的名字applicationContext.xml-->
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>

  </servlet>

  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>


</web-app>
```

13.4. Springmvc.xml

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

    <!--配置扫描组件    -->
    <context:component-scan base-package="com.neuedu.controller"/>

    <mvc:annotation-driven  />

</beans>
```

13.5. 创建上传文件的表单

上传文件时form表单要求： 1 ） method ： post 2 ） enctype: multipart/form-data

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/4/1
  Time: 9:00
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<html>
<head>
    <title>上传</title>
</head>
<body>

<!--
    上传文件时form表单要求：
    1 method ：  post
    2 enctype：  multipart/form-data

 -->

    <form action="${pageContext.request.contextPath}/upload_file"  method="post" enctype="multipart/form-data">

        上传文件1:<input type="file" name="myfile" /><br/>
        上传文件2:<input type="file" name="myfile" /><br/>

        <input type="submit" value="上传"/>

    </forma>


</body>
</html>
```

13.6. 配置multipartResolver

使用什么类型的库解析request中的文件内容

1）使用Servlet原生的方式

2）common-fileupload

在springmvc中声明multipartResolver

```
!--指定使用common-fileupload类型的方式解析request中的文件内容-->
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!--设置允许上传的最大多少字节 -->
    <property name="maxUploadSize" value="2000"></property>
</bean>
```

13.7. 控制器的方法接受上传的文件

1)将表单中的文件持久化的保存到服务器硬盘中,

2)记录到数据库中。

1 xxx 2020年4月1日08:43:35 96530565d35746b19bf5a5ee3251fdf8.txt a.txt

2 xxx 2020年4月1日08:43:35 86530565d35746b19bf5a5ee3251fdf8.txt a.txt

3 xxx 2020年4月1日08:43:35 76530565d35746b19bf5a5ee3251fdf8.txt b.txt

4 xxx 2020年4月1日08:43:35 66530565d35746b19bf5a5ee3251fdf8.txt ctxt

5 xxx 2020年4月1日08:43:35 596c08719709471697d0c39df87f8d6f.txt d.txt

```java
package com.neuedu.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.IOException;
import java.util.UUID;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/4/1  8:53 01
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 文件上传下载的控制器
 */
@Controller
public class FileUploadController {

    //上传到目标文件夹
    private static final String BASE_DIR ="D:\\java1upload\\";


    @RequestMapping("/index")
    @ResponseBody
    public String success(){
        return "success";
    }


    /**
```

```
     * 接受浏览器上传文件的控制器方法
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/upload_file",method = RequestMethod.POST) //只允许post提交
    public String upload(MultipartFile[] myfile  ) throws IOException {

        //myfile 此对象代表上传到服务器的文件句柄，在临时目录,需要转储到指定的目录（D:\\java1upload）

        for (MultipartFile file : myfile) {

            //转储到指定位置
            //生成唯一的文件名
            //转储的目标文件  destFile
            File destFile = new File(BASE_DIR, getNewFileName(file.getOriginalFilename()));
            destFile.createNewFile();
            file.transferTo(destFile); //转储的方法
        }
        return "success";
    }


    public static void main(String[] args) {


        //获取新的 文件名
        for (int i = 0; i <10 ; i++) {

            System.out.println(getNewFileName("a.txt"));
        }


    }

    public static String getNewFileName(String orgName){

//        String newFileName = UUID.randomUUID().toString();
        /**
         * d36f2ac3-d634-420b-a9ef-7daa6c701b29
         * 9f249028-63a6-4843-8227-c6f40d3a73c3
         * c13d36da-4f8f-4832-9868-de87e4eed458
         * 1c2266bd-7cb9-4291-859f-0472b2f5cbe9
         * df056bfb-1657-4598-b27d-836886b72f5b
         */


        String newFileName = UUID.randomUUID().toString().replace("-","");
        /**
         * cf4e231a859247248e376ce9dd787a78
         * d7de91b743c546718138ca965d0a6a90
         * 77ee1ebc28f246408794fd9ac7240ebc
         * d937977c2c13408280ed2bbddb948af5
         * c5d69044f56c4fb5a5df07d21f8097d2
         * ca0782562bd34f17838a18b39d7b8833
         * f1bea843ff694d5aabf590849ef3d3be
         */

        newFileName += orgName.substring(orgName.lastIndexOf("."));
        /**
         * c2e15fc3b7eb4d028bcaa270c4fd817c.txt
         * aad35ed22c674f8781f60ba581b031e6.txt
         * 7aa862adc57a481aa41bcb91ae910104.txt
         * ff9d14279bf545ad99b0668c92d8828b.txt
         * 025ae8b3d61a4e6cb8516af3da0ba215.txt
         */


        return newFileName;
    }


}
```

13.8. 测试

13.9. 下载

13.9.1. 展示列表

13.9.1.1. 控制的方法

```
/**
 * 列表展示文件
 * 1   xxx   2020年4月1日08:43:35     96530565d35746b19bf5a5ee3251fdf8.txt   a.txt
 * 2   xxx   2020年4月1日08:43:35     86530565d35746b19bf5a5ee3251fdf8.txt   a.txt
 * 3   xxx   2020年4月1日08:43:35     76530565d35746b19bf5a5ee3251fdf8.txt   b.txt
 * 4   xxx   2020年4月1日08:43:35     66530565d35746b19bf5a5ee3251fdf8.txt   ctxt
 * 5   xxx   2020年4月1日08:43:35     596c08719709471697d0c39df87f8d6f.txt   d.txt
 * @param  从Controller开始  http://127.0.0.1:8080/springmvc/filelist
 */
@RequestMapping(value = "/filelist")
public String upload(Model model) throws IOException {

    //查询数据库 返回list （文件）

    //暂时直接列表 D:\java1upload
    File uploadDir = new File(BASE_DIR);

    //所有文件的数组
    File[] files = uploadDir.listFiles();
    model.addAttribute("files",files);//将集合放到作用域中。

    return "/file/file_list.jsp";
}
```

13.9.1.2. Jsp列表

13.9.2. 提供下载的方法

根据文件名、id等条件查询对象的文件句柄并提供下载功能

13.9.2.1. 编码的形式自己读取文件并通过response响应

```
/**
     * http://127.0.0.1:8080/springmvc/download?filename=596c08719709471697d0c39df87f8d6f.txt
     * @param filename
     * @return
     * @throws IOException
     */
    @RequestMapping(value = "/download")
    public void download(String filename, HttpServletResponse response) throws IOException {

//        new File("D:\\java1upload\\",filename);
        File downFile = new File(BASE_DIR, filename);

        //告诉浏览器下面向浏览器发送附件
        response.setHeader("Content-Disposition","attachment;filename="+filename);
        ServletOutputStream os = response.getOutputStream();
        FileInputStream fis = new FileInputStream(downFile);

        int len = 0;
        byte[] buffer = new byte[1024];//缓存区
        while( (len = fis.read(buffer)) != -1){
            os.write(buffer,0,len);
            os.flush();
        }

        os.close();
        fis.close();
    }
```

Spring-web模块的ResponseEntity类快速的构建一个响应内容

```
/**
 *   http://127.0.0.1:8080/springmvc/download2?filename=596c08719709471697d0c39df87f8d6f.txt
 * @param filename
 * @param response
 * @return
 * @throws IOException
 */
@RequestMapping(value = "/download2")
public ResponseEntity download2(String filename, HttpServletResponse response) throws IOException {
    File downFile = new File(BASE_DIR, filename);


    ResponseEntity entity = ResponseEntity.ok().
            //mime
            header(HttpHeaders.CONTENT_TYPE,"application/octet-stream").
            //通知浏览器以什么方式处理响应结果（直接打开，附件下载）
            header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + downFile.getName() + "\"").
            //设置body中为文件资源
            body(new FileSystemResource(downFile));
    return entity;
}
```

14. json数据交互

在数据请求中特别是ajax中，常用到json格式。

14.1. 添加依赖

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.2.4.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.4</version>
</dependency>


<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>


<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.68</version>
```

```
</dependency>


<!--解析对象为json-->
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.9</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-annotations</artifactId>
  <version>2.9.9</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.9.1</version>
</dependency>
```

14.2. 页面

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/4/1
  Time: 14:35
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script>

        function req(){

            var xmlhttp;
            if (window.XMLHttpRequest)   {
                //  IE7+，Firefox，Chrome，Opera，Safari 浏览器执行代码
                xmlhttp=new XMLHttpRequest();
            }
            else {
                // IE6，IE5 浏览器执行代码
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }

            xmlhttp.open("GET","json1",true);
            xmlhttp.send();

            xmlhttp.onreadystatechange=function(){
                if (xmlhttp.readyState==4 && xmlhttp.status==200){
                     var person = eval('('+xmlhttp.responseText+')');
                     alert(person.name)
                     alert(person.age)
                }
            }
        }

    </script>
</head>
<body >
    ajax信息 <br/>

<input type="button" value="ajax请求" onclick="req()" />

</body>
</html>
```

14.3. 原生ServletAPI的形式响应json数据

```
package com.neuedu.controller;

import com.alibaba.fastjson.JSON;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;

/**
 * 项目    :  spring-mvc-java1
 * 创建时间 : 2020/4/1  14:38 01
 * author  : jshand-root
 * site    :  http://314649444.iteye.com
 * 描述     : 使用json格式进行交互
 */
@Controller
public class JsonController {


    @RequestMapping("/json1")
    public void json1(HttpServletRequest request, HttpServletResponse response) throws IOException {

        response.setContentType("application/json;charset=utf-8");
        PrintWriter out = response.getWriter();


        Map user = new HashMap();
        user.put("name","郭靖-m");
        user.put("age","45");
        //使用 类库的形式 将待键值对的 map

        //将map对象转换成json格式的字符串
        String json = JSON.toJSONString(user);

//        out.print("{'name':'金山','age':30}");
        out.print(json);

        out.flush();
        out.close();
    }

}
```

14.4. 基础SpringMVC的机制自动的转换类型输出json

```
<%--
  Created by IntelliJ IDEA.
  User: root
  Date: 2020/4/1
  Time: 14:35
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <script>

        function req(){

            var xmlhttp;
            if (window.XMLHttpRequest)   {
                //  IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
                xmlhttp=new XMLHttpRequest();
            }
            else {
                // IE6, IE5 浏览器执行代码
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }

            xmlhttp.open("GET","json1",true);
            xmlhttp.send();

            xmlhttp.onreadystatechange=function(){
                if (xmlhttp.readyState==4 && xmlhttp.status==200){
                     var person = eval('('+xmlhttp.responseText+')');
                     alert(person.name)
                     alert(person.age)
                }
            }
        }



        function req2(){
            var xmlhttp;
            if (window.XMLHttpRequest)   {
                //  IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
                xmlhttp=new XMLHttpRequest();
            }
            else {
                // IE6, IE5 浏览器执行代码
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }

            xmlhttp.open("GET","ajax_html",true);
            xmlhttp.send();

            xmlhttp.onreadystatechange=function(){
                if (xmlhttp.readyState==4 && xmlhttp.status==200){
                    var html = xmlhttp.responseText;
                   // alert(html)
                    document.getElementById("content").innerHTML = html;
                }
            }
        }


        function req3(){

            var xmlhttp;
            if (window.XMLHttpRequest)   {
                //  IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
                xmlhttp=new XMLHttpRequest();
            }
            else {
                // IE6, IE5 浏览器执行代码
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }

            xmlhttp.open("GET","json2",true);
            xmlhttp.send();

            xmlhttp.onreadystatechange=function(){
                if (xmlhttp.readyState==4 && xmlhttp.status==200){
                    var person = eval('('+xmlhttp.responseText+')');
                    alert(person.name)
                    alert(person.age)
                }
            }
        }


    </script>
</head>
<body >
    ajax信息 <br/>
```

```

<input type="button" value="ajax请求-json" onclick="req()" />
<input type="button" value="ajax请求-html" onclick="req2()" />
    <input type="button" value="ajax请求-json-springmvc" onclick="req3()" />

<div id="content"></div>


</body>
</html>
```

14.5. 控制器的方法

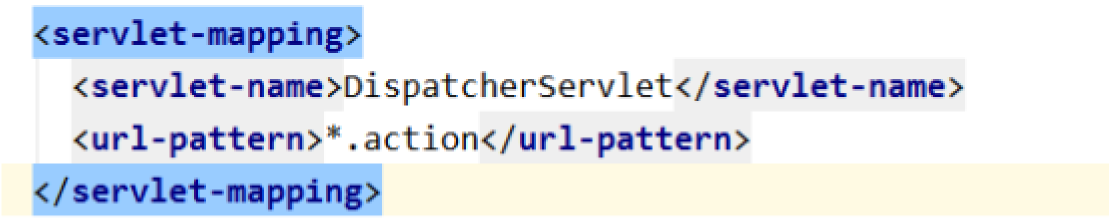
```
@ResponseBody
@RequestMapping("/json2")
public Map json2()  {

    Map user = new HashMap();
    user.put("name","郭靖-m");
    user.put("age","45");

    return user;
}
```

15. RESTful支持

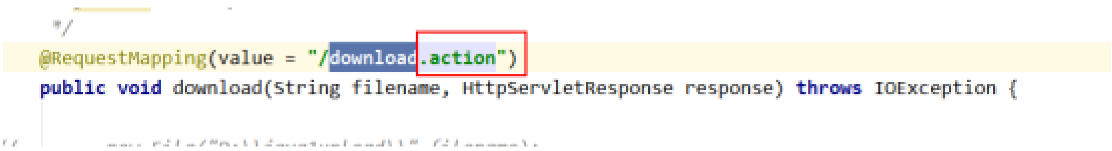
15.1. 概述：

Url：/ 静态资源404

15.1.1. 使用具体扩展名限定控制器方法的url映射 如:**



```
<servlet-mapping>
  <servlet-name>DispatcherServlet</servlet-name>
  <url-pattern>*.action</url-pattern>
</servlet-mapping>
```



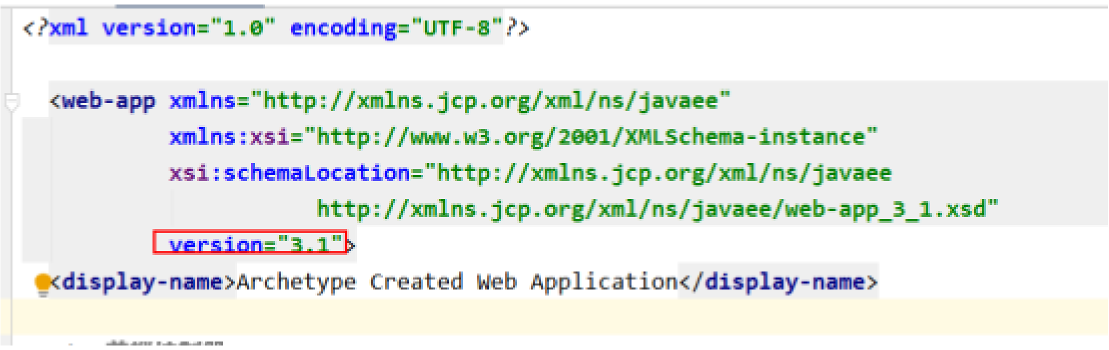
```
 */
@RequestMapping(value = "/download.action")
public void download(String filename, HttpServletResponse response) throws IOException {
```

15.1.2. 就行使用/对RESTFul风格的支持。

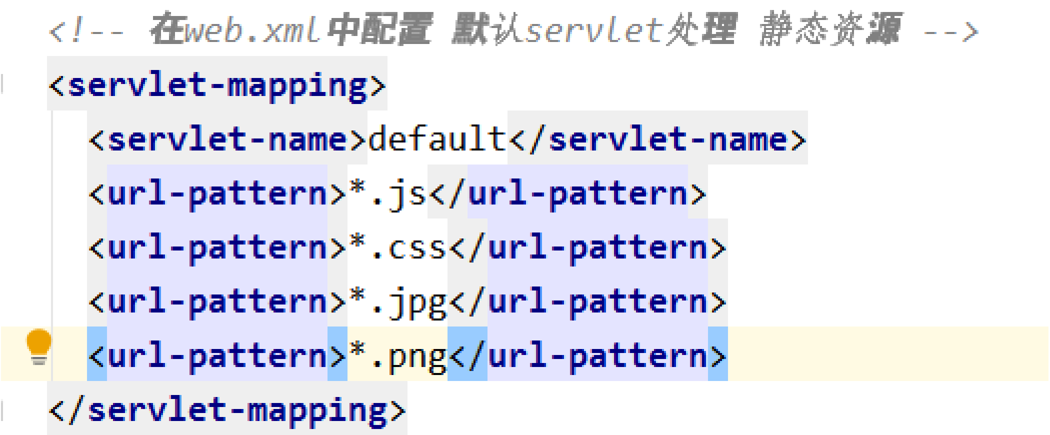
15.1.2.1. 需要解决静态资源404的问题

15.1.2.1.1. 将静态资源交还给默认servlet-default**

在web.xml中配置 默认servlet处理 静态资源

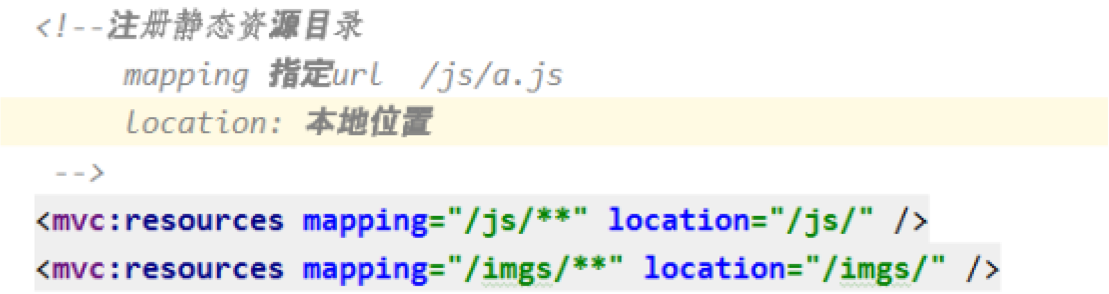


```xml
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                     http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
<display-name>Archetype Created Web Application</display-name>
```



```xml
<!-- 在web.xml中配置 默认servlet处理 静态资源 -->
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>*.js</url-pattern>
  <url-pattern>*.css</url-pattern>
  <url-pattern>*.jpg</url-pattern>
  <url-pattern>*.png</url-pattern>
</servlet-mapping>
```

15.1.2.1.2. 在springmvc的容器中声明静态资源目录

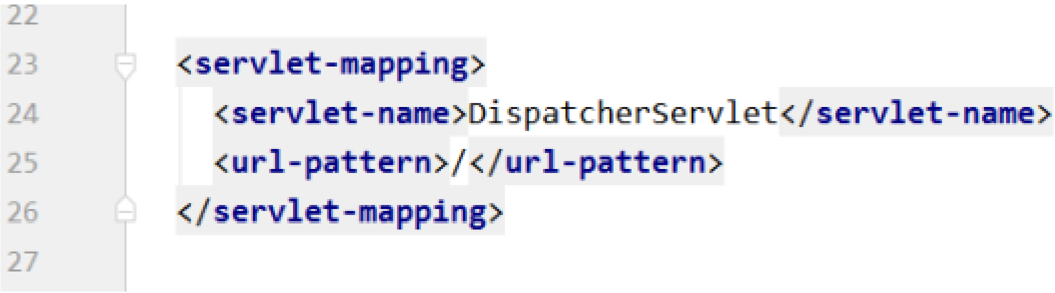


```xml
<!--注册静态资源目录
    mapping 指定url  /js/a.js
    location: 本地位置
 -->
<mvc:resources mapping="/js/**" location="/js/" />
<mvc:resources mapping="/imgs/**" location="/imgs/" />
```

15.1.2.1.3. 在springmvc容器中声明默认的Servlet处理器

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

15.2. RESTful风格的实现



```xml

<servlet-mapping>
  <servlet-name>DispatcherServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

```

15.2.1. 控制器方法

```java
package com.neuedu.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 项目    ：  spring-mvc-java1
 * 创建时间 ：2020/4/2  10:42 02
 * author  ：jshand-root
 * site    ：  http://314649444.iteye.com
 * 描述     ：RESTful风格的控制器
 */
@Controller
public class RESTfulController {


    static List<Map> userList = new ArrayList();

    /**
     * 模拟从数据库中查询出的10个用户信息
     */
    static {
        for (int i = 1; i <= 10 ; i++) {
            Map user = new HashMap();
            user.put("id",i+"");
            user.put("name","name:"+i);
            user.put("address","address:"+i);
            userList.add(user);
        }
    }


    //http://127.0.0.1:8080/springmvc/user/queryById/5
    @RequestMapping("/user/queryById/{id}")
    @ResponseBody
//    public Map queryById(String id){      绑定的参数是 ？id=10
    public Map queryById(@PathVariable("id") String id){   // context/{id}   -- String id

        for (Map user : userList) {
            if(id.equals(user.get("id"))){
                return  user;
            }
        }
        return null;
    }
}
```

15.2.2. RESTful风格的路径写法（支持通配符）

@RequestMapping不但支持标准的URL，还支持Ant风格（即?、*和**的字符，）的和带{xxx}占位符的URL。以下URL都是合法的：

- /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。

- /user/{userId}

  匹配user/123、user/abc等URL。

- /user/**/{userId}

  匹配user/aaa/bbb/123、user/aaa/456等URL。

- company/{companyId}/user/{userId}/detail

  匹配company/123/user/456/detail等的URL。

15.2.3. 实例：（跟上面有重复）

*代表至少一个字符以上的统配

** 匹配的是多级目录，字符个数可以没有，也可以有多个

? 有且匹配一个 1

```
* 代表至少一个字符以上的统配



** 匹配的是多级目录，字符个数可以没有，也可以有多个



? 有且匹配一个



1 user/*/createUser

 √ user/aa/createUser

 √ user/bb/createUser

 √ user/abdsdfasf/createUser

 √ user/a/createUser

× user/createUser



```

```


2 user/**/createUser

 √ user/aa/createUser

 √ user/createUser

 √ user/aac/bbe/createUser

3 user/createUser?

 √ user/createUsera

 √ user/createUserb

 √ user/createUserc

× user/createUser

× user/createUseraaa

× user/createUserbbb

× user/createUserabc
```