4.8 SpringMVC之SpringMVC入门

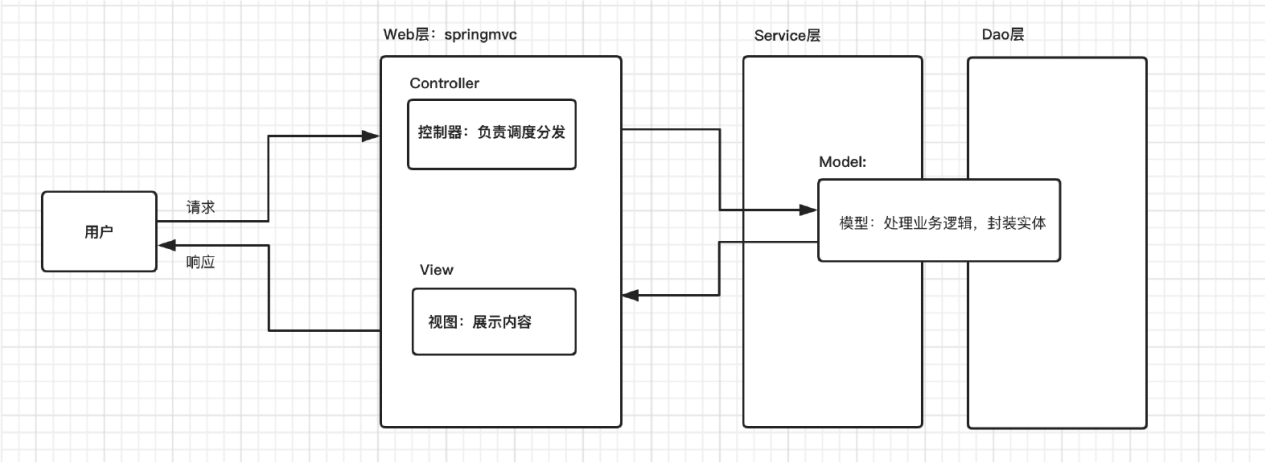
4.8.2 SpringMVC概述

MVC是软件工程中的一种软件架构模式，它是一种分离业务逻辑与显示界面的开发思想。

M模型：处理业务逻辑，封装实体

V视图：展示内容

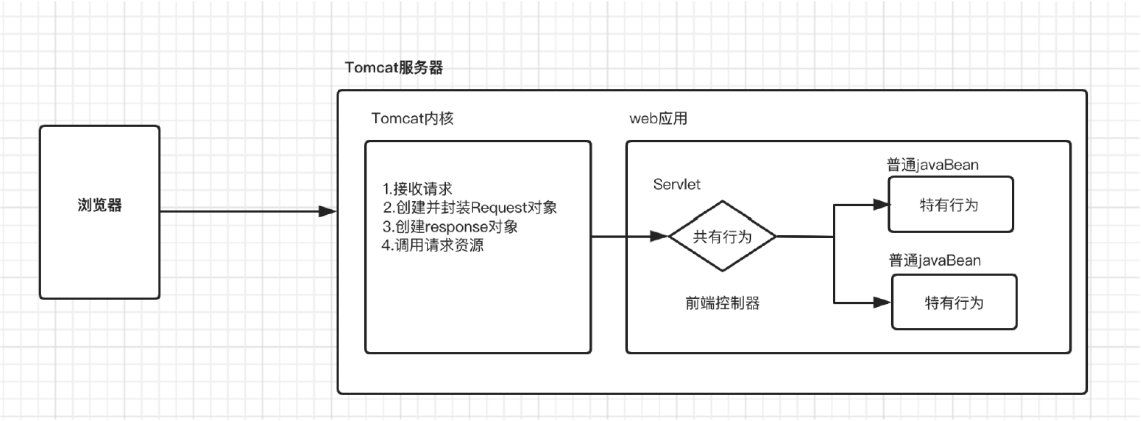
C控制器：负责调度分发，例如，接受请求，调用模型，转发视图



SpringMVC

SpringMVC是一种基于Java来实现MVC设计模式的**Web框架，支持Restful编程风格**。

4.8.4 JavaWeb执行流程

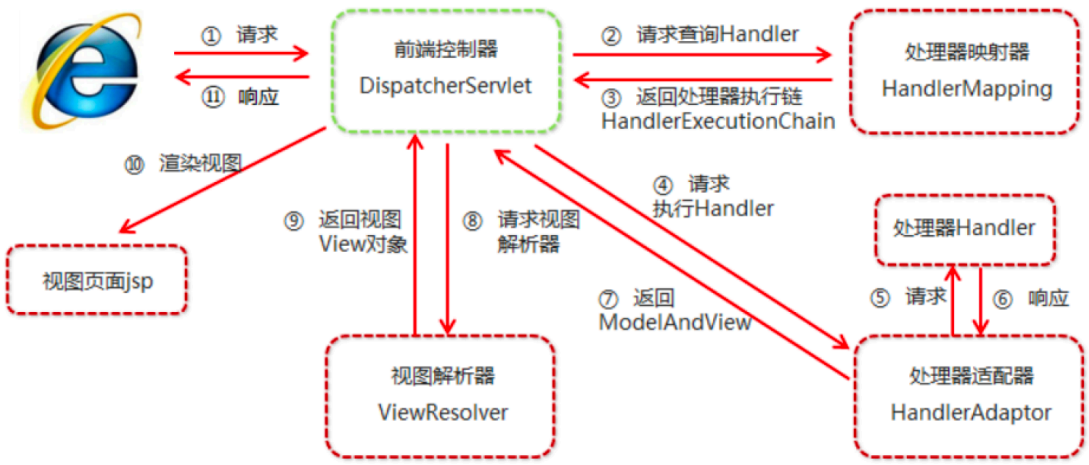


http://localhost:8080/springmvc_quickstart/quick 发给tomcat服务器

1. tomcat内核会接受请求，解析请求资源地址，创建并封装Request对象，创建Response对象，调用请求资源
2. 在web.xml文件中，根据servlet-mapping调用对应的servlet。本案例中，就会调用DispatcherServlet前端控制器，DispatcherServler就会解析映射地址，找到对应控制器中标有@RequestMapping的方法。

总结：由DispatcherServlet前端控制器解析请求资源，决定执行哪个controller中的方法

4.8.5 SpringMVC执行流程



1. 用户发送请求，tomcat服务器接收并解析请求，然后交给前端控制器DispatcherServlet
2. DispatcherServlet调用处理器映射器HandlerMapping来查找对应的控制器Controller
3. 找到后，会返回处理器执行链，这样DispatcherSerlvet就知道对应的控制器Controller
4. 然后发送给处理器适配器HandlerAdaptor这个组件
5. HandlerAdapter就会调用具体的控制器
6. 控制器执行完成返回ModelAndView对象
7. HandlerAdapator将ModelAndView对象返还给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器
9. ViewReslover解析后返回具体View
10. DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）
11. DispatcherServlet将渲染后的视图响应响应用户

4.8.6 SpringMVC组件解析

前端控制器：DispatcherServlet

用户请求到达前端控制器，它就相当于 MVC 模式中的 C，DispatcherServlet 是整个流程控制的中心，由它调用其它组件处理用户的请求，DispatcherServlet 的存在降低了组件之间的耦合性。

处理器映射器：HandlerMapping

HandlerMapping 负责根据用户请求找到 Handler 即处理器Controller，SpringMVC 提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

处理器适配器：HandlerAdapter

通过 HandlerAdapter 对处理器Controller进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

处理器：Handler 开发者编写的Controller文件

它就是我们开发中要编写的具体业务控制器。由 DispatcherServlet 把用户请求转发到Handler。由Handler对具体的用户请求进行处理。

视图解析器：ViewResolver

ViewResolver 负责将处理结果生成 View 视图，ViewResolver 首先根据逻辑视图名解析成物理视图名，即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。

视图：View 开发者编写

SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView等。最常用的视图就是 jsp。一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

手动配置HandlerMapping，HandlerAdapter，ViewResolver

```
<!--处理器映射器和处理器适配器功能增强，进行了功能的增强，支持json-->
<mvc:annotation-driven></mvc:annotation-driven>

<!--视图解析器-->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/pages/"></property>
  <property name="suffix" value=".jsp"></property>
</bean>
```

4.8.7 SpringMVC注解解析

@Controller

SpringMVC基于Spring容器，所以在进行SpringMVC操作时，需要将Controller存储到Spring容器中

@RequestMapping

作用：用于建立**请求 URL** 和**处理请求方法**之间的对应关系

位置：

1. 类上：请求URL的第一级访问目录。此处不写的话，就相当于应用的根目录。写的话需要以/开头。它出现的目的是为了使我们的URL可以按照模块化管理
2. 方法上：请求URL的第二级访问目录，和一级目录组成一个完整的 URL 路径。

属性：

1. value：用于指定请求的URL。它和path属性的作用是一样的
2. method：用来限定请求的方式
3. params：用来限定请求参数的条件

   例如：params={"accountName"} 表示请求参数中必须有accountName

   pramss={"money!100"} 表示请求参数中money不能是100

4.8.8 请求参数类型介绍

SpringMVC可以接收如下类型的参数：

1. 基本类型参数
2. 对象类型参数
3. 数组类型参数
4. 集合类型参数

4.8.9 获取基本类型请求参数

Controller中的业务方法的参数名称要与请求参数的key值一致，参数值会自动映射匹配。并且能自动做类型转换；自动的类型转换是指从String向其他类型的转换。

```
<a href="${pageContext.request.contextPath}/user/simpleParam?id=1&username=杰克">
  基本类型
</a>
```

```
@RequestMapping("/simpleParam")
public String simpleParam(Integer id,String username) {
  System.out.println(id);
  System.out.println(username);
  return "success";
}
```

4.8.10 获取对象类型请求参数

1. 首先定义一个对应的类
2. 其次确保类中的成员变量名要和请求的参数一致

```
<form action="${pageContext.request.contextPath}/user/pojoParam" method="post">
  编号：<input type="text" name="id"> <br>
  用户名：<input type="text" name="username"> <br>
  <input type="submit" value="对象类型">
</form>
```

```
public class User {
  Integer id;
  String username;
  // setter getter...
}
```

```
@RequestMapping("/pojoParam")
public String pojoParam(User user) {
  System.out.println(user);
  return "success";
}
```

当post请求时，数据会出现乱码，我们可以设置一个过滤器来进行编码的过滤。

```
<!--配置全局过滤的filter-->
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filterclass>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

4.8.11 获取数组类型请求参数

Controller中的业务方法数组名称与请求参数的key一致，参数值会自动映射匹配

```
<form action="${pageContext.request.contextPath}/user/arrayParam">
  编号：<br>
  <input type="checkbox" name="ids" value="1">1<br>
  <input type="checkbox" name="ids" value="2">2<br>
  <input type="checkbox" name="ids" value="3">3<br>
  <input type="checkbox" name="ids" value="4">4<br>
  <input type="checkbox" name="ids" value="5">5<br>
  <input type="submit" value="数组类型">
</form>
```

```
@RequestMapping("/arrayParam")
public String arrayParam(Integer[] ids) {
  System.out.println(Arrays.toString(ids));
  return "success";
}
```

4.8.12 获取集合类型请求参数

获得集合参数时，要将集合参数包装到一个POJO中才可以

```
<form action="${pageContext.request.contextPath}/user/queryParam" method="post">
  搜索关键字：
    <input type="text" name="keyword"> <br>

  user对象：
    <input type="text" name="user.id" placeholder="编号">
    <input type="text" name="user.username" placeholder="姓名"><br>

  list集合：
    第一个元素：
    <input type="text" name="userList[0].id" placeholder="编号">
    <input type="text" name="userList[0].username" placeholder="姓名"><br>
    第二个元素：
    <input type="text" name="userList[1].id" placeholder="编号">
    <input type="text" name="userList[1].username" placeholder="姓名"><br>

  map集合
    第一个元素：
    <input type="text" name="userMap['u1'].id" placeholder="编号">
    <input type="text" name="userMap['u1'].username" placeholder="姓名"><br>
```

```
    第二个元素:
    <input type="text" name="userMap['u2'].id" placeholder="编号">
    <input type="text" name="userMap['u2'].username" placeholder="姓名"><br>

  <input type="submit" value="复杂类型">
</form>
```

```
public class QueryVo {
  private String keyword;
  private User user;
  private List<User> userList;
  private Map<String, User> userMap;
}
```

```
@RequestMapping("/queryParam")
public String queryParam(QueryVo queryVo) {
  System.out.println(queryVo);
  return "success";
}
```

4.8.13 自定义类型转换器

1. 创建一个类实现Converter接口

   ```
   public class DateConverter implements Converter<String, Date>{}
   ```

2. 重写convert方法

   ```
   //s就是表单传递过来的请求参数
   @Override
   public Date convert(String s){

   }
   ```

3. 在配置文件中进行配置

```
<bean id="conversionFactory"
class="org.springframework.context.support.ConversionServiceFactoryBean">
  <property name="converters">
    <set>
      <bean class="com.zichen.converter.DateConverter"></bean>
    </set>
  </property>
</bean>
```

4. 并把配置信息添加到

```
<!--处理器映射器和适配器增强-->
<mvc:annotation-driven conversion-service="conversionFactory">
</mvc:annotationdriven>
```

4.8.14 @RequestParam注解

当请求的参数key名称与Controller的业务方法参数名称**不一致**时，就需要通过@RequestParam注解显示的绑定

```
<a href="${pageContext.request.contextPath}/user/findByPage?pageNo=2">
  分页查询
</a>
```

```
/*
@RequestParam() 注解
defaultValue 设置参数默认值
name 匹配页面传递参数的名称
required 设置是否必须传递参数，默认值为true；如果设置了默认值，值自动改为false
*/
@RequestMapping("/findByPage")
public String findByPage(@RequestParam(name = "pageNo", defaultValue = "1")
Integer pageNum,
                         @RequestParam(defaultValue = "5") Integer pageSize) {
  System.out.println(pageNum);
  System.out.println(pageSize);
  return "success";
}
```

4.8.15 @RequestHeader, @CookieValue

@RequestHeader: 获取请求头的数据。

@CookieValue: 获取cookie中的数据。

4.8.16 获取Servlet相关API

SpringMVC支持使用原始ServletAPI对象作为控制器方法的参数进行注入，常用的对象如下

```
@RequestMapping("/servletAPI")
public String servletAPI(HttpServletRequest request, HttpServletResponse response,
HttpSession session) {
  System.out.println(request);
  System.out.println(response);
  System.out.println(session);
  return "success";
}
```

4.8.17 响应方式介绍

页面跳转

1. 返回字符串逻辑视图
2. void原始ServletAPI：借助request，response对象
3. ModelAndView对象

返回数据

1. 直接返回字符串数据
2. 将对象或集合转为json格式返回

4.8.18 页面转发和重定向

1. 返回字符串逻辑视图：本质是请求转发，地址栏不会发生改变
2. 原始ServletAPI实现页面跳转

```
@RequestMapping("/forward")
public void returnVoid(HttpServletRequest request, HttpServletResponse
response){
  //借助request对象完成请求转发

  //借助response对象完成重定向

}
```

如果是请求转发，地址栏不会发生改变，前面可以加**forward关键字**，既可以转发到jsp，也可以转发到其他的控制器方法

```
@RequestMapping("/forward")
public String forward(Model model) {
  //return "forward:/product/findAll";
  model.addAttribute("username", "拉勾招聘");
  return "forward:/WEB-INF/pages/success.jsp";
}
```

如果是重定向，地址栏会发生改变，可以使用**redirect关键字**，我们可以不写虚拟目录，springMVC框架会自动拼接，并且将Model中的数据拼接到url地址上

```
@RequestMapping("/redirect")
public String redirect(Model model) {
  //底层使用的还是request.setAtrribute("username","拉勾教育")域范围；一次请求
  model.addAttribute("username", "拉勾教育");
  return "redirect:/index.jsp";
}
```

4.8.19 ModelAndView应用

方式一：在Controller中，方法创建并返回ModelAndView对象，并且设置视图名称

mode/l模型：用来封装存放数据

view/视图：用来展示数据

```
@RequestMapping("/returnModelAndView1")
public ModelAndView returnModelAndView1() {

  ModelAndView modelAndView = new ModelAndView();
  //设置模型数据
  modelAndView.addObject("username", " lagou");
  //设置视图名称
  modelAndView.setViewName("success");
  return modelAndView;
}
```

方式二：在Controller中方法形参上直接声明ModelAndView，无需在方法中自己创建，在方法中直接使用该对象设置视图，同样可以跳转页面

```
@RequestMapping("/returnModelAndView2")
public ModelAndView returnModelAndView2(ModelAndView modelAndView) {
  //设置模型数据
  modelAndView.addObject("username", "itheima");
  //设置视图名称
  modelAndView.setViewName("success");
  return modelAndView;
}
```

返回数据

1. 直接返回字符串数据

```
public void returnVoid(HttpServletRequest request, HttpServletResponse
response){
  response.setContentType("text/html;charset=utf-8");
  response.getWriter().write("拉勾网");
}
```

4.8.20 @SessionAtribute注解

如果在多个请求之间共用数据，则可以在控制器类上标注一个 @SessionAttributes,配置需要在session中存放的数据范围，Spring MVC将存放在model中对应的数据暂存到 HttpSession 中。

4.8.21 开启静态资源访问

当有静态资源需要加载时，比如jquery文件，通过谷歌开发者工具抓包发现，没有加载到jquery文件，原因是SpringMVC的前端控制器DispatcherServlet的url-pattern配置的是 /（缺省）,代表对所有的静态资源都进行处理操作，这样就不会执行Tomcat内置DefaultServlet处理，我们可以通过以下两种方式指定放行静态资源：

在spring-mvc配置文件中进行配置

方式一

```
<!--在springmvc配置文件中指定放行资源-->
<!--mapping：放行的映射路径; location：静态资源所在的目录 -->
<mvc:resources mapping="/js/**" location="/js/"/>
<mvc:resources mapping="/css/**" location="/css/"/>
<mvc:resources mapping="/img/**" location="/img/"/>
```

方式二

```
<!--在springmvc配置文件中开启DefaultServlet处理静态资源-->
<mvc:default-servlet-handler/>
```

4.9 SpringMVC之SpringMVC进阶

4.9.2 Ajax异步交互

前端和后端交互格式：json格式；

1. 后端怎么把json转换成对象进行处理?
2. 前端怎么把对象转化成json进行处理?

SpringMVC默认使用**MappingJackson2HttpMessageConverter**对json数据进行转换：

1. 需要加入jackson包
2. 同时使用<mvc: annotation-driven/>

@RequestBody: json -> 对象

该注解用于Controller的方法的形参声明，当使用**ajax提交并指定contentType为json形式**时，通过HttpMessageConverter接口转换为对应的POJO对象。

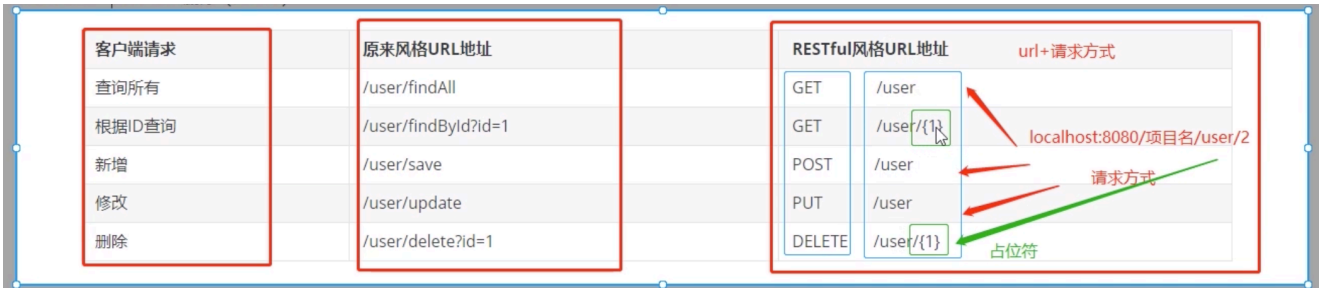
@ResponseBody: 对象 -> xml/json

该注解用于将Controller的方法返回的对象，通过HttpMessageConverter接口转换为指定格式的数据如：json,xml等，通过Response响应给客户端。

4.9.3 Restful服务

Restful是一种编程风格(url+请求格式)，主要用于客户端与服务器交互

1. GET -> Read
2. POST -> Create
3. PUT -> Update
4. DELETE -> Delete



@PathVariable

用来接收RESTful风格请求地址中占位符的值。占位符的值要和参数名一致

@RestController

RESTful风格多用于前后端分离项目开发，前端通过ajax与服务器进行异步交互，我们处理器通常返回的是json数据所以使用@RestController来替代@Controller和@ResponseBody两个注解。

```
@RestController //@Controller + @ResponseBody
@RequestMapping("/restful")
public class RestfulController {
//  @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    @GetMapping(value = "/user/{id}")
    public String findById(@PathVariable Integer id){
        return "findById:"+id;
    }

    @PostMapping("/user")
    public String post(){
        return "post";
    }

    @PutMapping("/user")
    public String put(){
        return "put";
    }

    @DeleteMapping("/user/{id}")
    public String delete(@PathVariable Integer id){
        return "delete:"+id;
    }
}
```

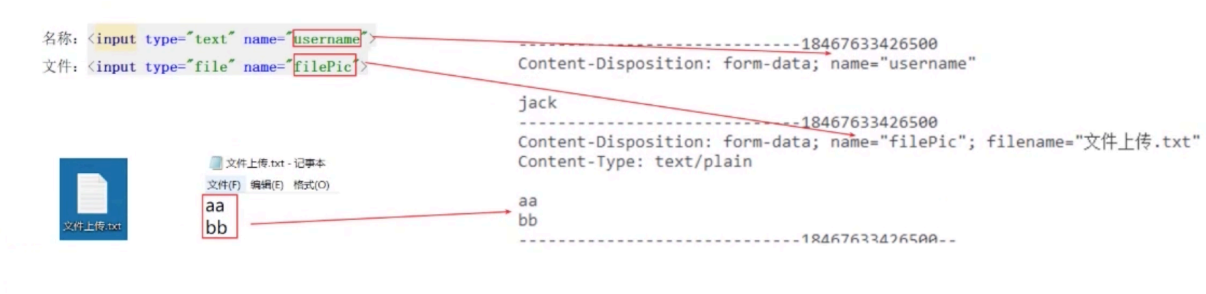
4.9.4 文件上传

文件上传三要素：

1. 表单项type=“file”
2. 表单的提交方式method=“post”
3. 表单的enctype属性是多部分表单形式 enctype=“multipart/form-data"

文件上传原理：

1. 当form表单的enctype取值为application/x-www-form-urlencoded 时，form表单的正文内容格式是： name=value&name=value
2. 主要是把表单的enctype属性改成“multipart/form-data"，这样请求正文内容就变成多部分形式。



4.9.5 单文件上传（springmvc_ajax）

1. 导入fileupload和io坐标
2. 配置文件上传解析器

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <!-- 文件上传最大值,5MB 5*1024*1024 -->
  <property name="maxUploadSize" value="5242880"></property>

  <!-- 设定文件上传时写入内存的最大值，如果小于这个参数不会生成临时文件，默认为10240 -->
  <property name="maxInMemorySize" value="40960"></property>
</bean>
```

3. 编写文件上传代码

```
<form action="${pageContext.request.contextPath}/fileupload" method="post"
enctype="multipart/form-data">
  名称：<input type="text" name="username"> <br>
  文件：<input type="file" name="file"> <br>
  <input type="submit" value="单文件上传">
</form>
```

```
@RequestMapping("/fileupload")
public String fileUpload(String username, MultipartFile file){
  System.out.println(username);
  String originalFileName = file.getOriginalFilename();
  try {
    file.transferTo(new File("/Users/zichenfu/Documents/大数据课程/阶段二/4.模块四/fileUpload/"+originalFileName                    ));
  } catch (IOException e) {
    e.printStackTrace();
  }
  return "success";
}
```

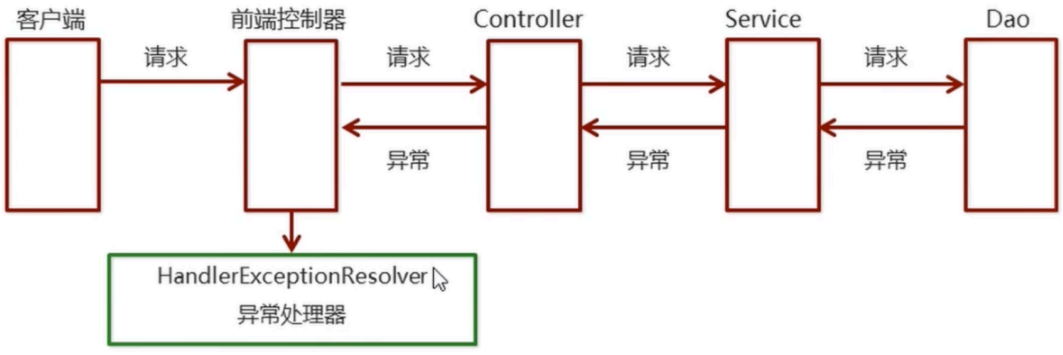
4.9.6 多文件上传

利用MultipartFile类型的数组来实现

4.9.7 异常处理思路 HandlerExceptionResolver

在Java中，对于异常的处理一般有两种方式:

1. 一种是当前方法捕获处理（try-catch），这种处理方式会造成业务代码和异常处理代码的耦合。
2. 另一种是自己不处理，而是抛给调用者处理（throws），调用者再抛给它的调用者，也就是一直向上抛。在这种方法的基础上，衍生出了SpringMVC的异常处理机制。**系统的dao、service、controller出现都通过throws Exception向上抛出，最后由springmvc前端控制器交由异常处理器进行异常处理，如下图**:



4.9.8 自定义异常处理器（springmvc_ajax）

1. 创建异常处理器类实现HandlerExceptionResolver
2. 配置异常处理器
3. 编写异常页面
4. 测试异常跳转

如果出现异常就会经过异常处理器

4.9.9 Web异常处理机制

```
<error-page>
  <error-code>500</error-code>
  <location>/500.jsp</location>
</error-page>

<error-page>
  <error-code>404</error-code>
  <location>/404.jsp</location>
</error-page>
```

4.9.10 拦截器概念

Spring MVC的拦截器和Servlet开发中的过滤器Filter类似，用于对处理器进行预处理和后处理。

将拦截器按一定的顺序联结成一条链，这条链称为拦截器链（InterceptorChain）。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。拦截器也是AOP思想的具体实现。

拦截器和过滤器的区别：

区别	过滤器	拦截器
使用范围	是 servlet 规范中的一部分，任何 Java Web 工程都可以使用	是 SpringMVC 框架自己的，只有使用了 SpringMVC 框架的工程才能用
拦截范围	在 url-pattern 中配置了/*之后，可以对所有要访问的资源拦截	只会拦截访问的控制器方法，如果访问的是 jsp，html,css,image 或者 js 是不会进行拦截的

4.9.11（springmvc_ajax）

1. 创建拦截器类实现HandlerInterceptor接口
2. 配置拦截器
3. 测试拦截器的拦截效果

4.9.12 拦截器链

END