Springmvc02

返回值分类

返回值是字符串

返回值是 void

返回值是 ModelAndView

```java
@Controller
@RequestMapping(value = "/user")
public class UserController {

    @RequestMapping(value = "/test")
    public String test(){
        System.out.println("欢迎访问UserController类的test方法！");
        return "success";
    }
    // 返回字符串（表单回显）
    @RequestMapping(value = "/userUpdate")
    public String userUpdate(Model model){
        System.out.println("欢迎访问UserController类的userUpdate方法！");
        // 封装到User的对象中
        User user = new User();
        user.setUsername("张三");
        user.setPassword("123");
        user.setAge(22);
        model.addAttribute("user",user); // 相当于存放到Request域
        return "update";
    }
    // 返回字符串（保存）
    @RequestMapping(value = "/update")
    public String update(User user){
        System.out.println("欢迎访问UserController类的update方法！user:"+user);
        return "update";
    }
    // 无返回值（保存）
    @RequestMapping(value = "/testVoid")
    public void testVoid(HttpServletRequest request, HttpServletResponse response) throws Exception {
        System.out.println("欢迎访问UserController类的testVoid方法！");
        // 1:转发，取代默认跳转的页面 /springmvc_day02_response/WEB-INF/pages/user/testVoid.jsp
        // request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request,response);
        // 2:重定向,
        // response.sendRedirect(request.getContextPath()+"/WEB-INF/pages/success.jsp");
        // response.sendRedirect(request.getContextPath()+"/user/test");
        // 3:直接相应数据，不会执行视图解析器
        response.setContentType("text/html;charset=utf-8");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().println("你好！");
    }
```

```java
// ModelAndView
@RequestMapping(value = "/testModelAndView")
public ModelAndView testModelAndView() throws Exception {
    System.out.println("欢迎访问UserController类的testModelAndView方法！");
    ModelAndView modelAndView = new ModelAndView();
    // 模型
    List<User> list = new ArrayList<>();
    User user1 = new User();
    user1.setUsername("张三");
    user1.setPassword("123");
    User user2 = new User();
    user2.setUsername("李四");
    user2.setPassword("123");
    list.add(user1);
    list.add(user2);
    modelAndView.addObject("list",list);
    // 视图
    modelAndView.setViewName("success"); // WEB-INF/pages/success.jsp
    return modelAndView;
}
// 转发和重定向的关键字
@RequestMapping(value = "/testForwardOrRedirect")
public String testForwardOrRedirect() throws Exception {
    System.out.println("欢迎访问UserController类的testForwardOrRedirect方法！");

    //return "success"; // 默认转发（没有使用关键字）
    //return "forward:/WEB-INF/pages/success.jsp";
    return "redirect:/user/test";// 重定向到/user/test-->success.jsp
}
// 测试ajax请求
@RequestMapping(value = "/testAjax")
@ResponseBody
public User testAjax(@RequestBody User user) throws Exception {
    System.out.println("欢迎访问UserController类的testAjax方法！user:"+user);
    // 响应对象对象（集合）
    User u = new User();
    u.setUsername("哈哈");
    u.setPassword("456");
    u.setAge(20);
    return u; // 让对象转换成json形式返回，不能执行视图解析器
}
```

去掉对静态资源的拦截

springmvc.xml 配置，去掉静态资源

```
<!--4：静态资源不过滤-->
<!--方案一
    mvc:resources mapping="" ：页面指定的src映射路径 (<script src="js/jquery.min.js"></script>)
    location="" ：项目中路径 (webapp/js/jquery.min.js)
-->
<!--<mvc:resources mapping="/js/**" location="/js/"></mvc:resources>-->
<!--<mvc:resources mapping="/css/**" location="/css/"></mvc:resources>-->
<!--<mvc:resources mapping="/img/**" location="/img/"></mvc:resources>-->
<!--4：静态资源不过滤-->
<!--方案二-->
<mvc:default-servlet-handler></mvc:default-servlet-handler>
```

普通文件上传

```
<%--
   enctype="multipart/form-data"把请求体分成多个部分上传
--%>
<form action="user/testFileUpload1" method="post" enctype="multipart/form-data">
    选择文件：<input type="file" name="upload"><br/>
    <input type="submit" value="上传"/><br/>
</form>
```

```
// 文件上传
@RequestMapping(path="/testFileUpload1")
public String testFileUpload1(HttpServletRequest request) throws Exception {
    System.out.println("执行了testFileUpload1方法！");
    // 1：上传的位置，获取到项目根目录下的uploads文件夹绝对路径
    String path = request.getSession().getServletContext().getRealPath("/uploads");
    // 创建file对象
    File file = new File(path);
    // 判断是否存在
    if(!file.exists()) {
        // 创建目录
        file.mkdirs();
    }
    //2：创建ServletFileUpload和磁盘文件项工厂
    DiskFileItemFactory factory = new DiskFileItemFactory();
    ServletFileUpload fileUpload = new ServletFileUpload(factory);
    // 3：解析request，遍历FileItem对象
    List<FileItem> list = fileUpload.parseRequest(request);
    // 遍历，获取到每一个文件项的对象
    for (FileItem fileItem : list) {
        // 判断，当前fileItem是否是文件项
        if(fileItem.isFormField()) {
            // 说明是普通的表单 ， 文本框,下拉框等
        }else {
            // 文件上传项 fileItem
            // 获取到文件的名称
            String filename = fileItem.getName();
            // 生成唯一标识
            String uuid = UUID.randomUUID().toString().replace("-", "").toUpperCase();
            filename = uuid+"_"+filename;
            // 4：上传文件，调用write()方法
            fileItem.write(new File(file, filename));
            // 删除临时文件
            fileItem.delete();
        }
    }
    return "success";
}
```

SpringMVC 传统方式文件上传

简化 ServletFileUpload 的写法。

SpringMVC 框架提供了 MultipartFile 对象，该对象表示上传的文件，要求变量名称必须和表单 file 标签的 name 属性名称相同。

第一步：springmvc.xml 中配置

配置文件解析器对象

```
<!-- 配置文件解析器对象，要求id名称必须是multipartResolver -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!--文件大小限制：10*1024*1024字节，表示10M-->
    <property name="maxUploadSize" value="10485760"/>
</bean>
```

第二步：index.jsp

```
<hr>
<form action="user/testFileUpload2" method="post" enctype="multipart/form-data">
    选择文件：<input type="file" name="upload"><br/>
    <input type="submit" value="上传"/><br/>
</form>
```

第三步：UserController.java

```
// springmvc提供的文件上传
/**
 * 1：在springmvc.xml中，添加文件上传解析器，解析request
 * 2：在Controller类的方法上使用解析完的属性MultipartFile upload,upload必须对应表单name的属性值
 */
@RequestMapping(value = "/testFileUpload2")
public String testFileUpload2(HttpServletRequest request, MultipartFile upload) throws Exception {
    System.out.println("欢迎访问UserController类的testFileUpload2方法！");
    // 1：指定上传文件的位置，获取存储文件的upload路径
    String path = request.getSession().getServletContext().getRealPath("/upload");
    File file = new File(path);
    if(!file.exists()){
        file.mkdirs();
    }
    // 2：上传文件名
    String fileName = upload.getOriginalFilename();
    fileName = UUID.randomUUID().toString().replace("-","").toUpperCase()+"_"+fileName;
    // 3：上传文件
    upload.transferTo(new File(file,fileName));
    return "success";
}
```

springmvc 跨服务器方式的文件上传

```
/**
 * 跨服务器的文件上传
 */
@RequestMapping(value = "/testFileUpload3")
public String testFileUpload3(HttpServletRequest request, MultipartFile upload) throws Exception {
    System.out.println("欢迎访问UserController类的testFileUpload3方法！");
    // 1：指定上传文件的位置，获取存储文件的upload路径
    String path = "http://localhost:9090/springmvc_day02_fileuploadserver/upload";

    // 2：上传文件名
    String fileName = upload.getOriginalFilename();
    fileName = UUID.randomUUID().toString().replace("-","").toUpperCase()+"_"+fileName;
    // 3：上传文件
    Client client = Client.create();
    WebResource webResource = client.resource(path + "/" + fileName);
    // 表示获取上传文件的字节，根据WebResource，传递远程服务器上
    webResource.put(upload.getBytes());
    // 传递文件名
    request.setAttribute("fileName",fileName);
    return "success";
}
```

SpringMVC 的异常处理

第一步：编写异常处理类

```
public class SysExceptionResolver implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, @Nullable Object o, Exception e) {
        e.printStackTrace();
        SysException se = null;
        // 获取到异常对象
        if(e instanceof SysException) {
            se = (SysException) e;
        }else {
            se = new SysException("请联系管理员");
        }
        ModelAndView mv = new ModelAndView();
        // 存入错误的提示信息
        mv.addObject("message", se.getMessage());
        // 跳转的Jsp页面
        mv.setViewName("error");// 跳转到WEB-INF/pages/error.jsp
        return mv;
    }
}
```

第二步：配置异常处理类

```
<!-- 配置异常处理器 -->
<bean id="sysExceptionResolver" class="com.itheima.exception.SysExceptionResolver"/>
```

自定义拦截器

编写自定义拦截器

```
public class MyInterceptor implements HandlerInterceptor {
    /** 表示在访问Controller类之前，先执行的方法
     * boolean：true：放行，可以访问Controller类的方法
     *          false：不放行，不可以访问Controller类的方法
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
        Object handler) throws Exception {
        System.out.println("【访问Controller类的方法之前执行拦截器...】");
//        // 使用转发或者重定向，指定错误页面
//        request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request,response);
        return true;
    }
    // 表示在访问Controller类之后，再执行的方法
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
        Object handler, @Nullable ModelAndView modelAndView) throws Exception {
//        // 使用转发或者重定向，指定错误页面
//        request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request,response);
        System.out.println("【访问Controller类的方法之后执行拦截器...，在访问success.jsp页面之前】");
    }
    // 表示在访问success.jsp页面之后，执行的方法（最终的方法）
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
        Object handler, @Nullable Exception ex) throws Exception {
        System.out.println("【访问success.jsp页面之后...】");
    }
}
```

```
<!-- (3) 配置拦截器-->
<mvc:interceptors>
    <mvc:interceptor>
        <!--拦截器处理的路径，处理user下的所有请求路径-->
        <mvc:mapping path="/user/**"/>
        <!--拦截器处理的路径，排除不执行拦截器的请求路径，/user/save不会执行拦截器-->
        <mvc:exclude-mapping path="/user/save"></mvc:exclude-mapping>
        <bean class="com.itheima.interceptor.MyInterceptor"></bean>
    </mvc:interceptor>
    <mvc:interceptor>
        <!--拦截器处理的路径，处理user下的所有请求路径-->
        <mvc:mapping path="/**"/>
        <!--拦截器处理的路径，排除不执行拦截器的请求路径，/user/save不会执行拦截器-->
        <mvc:exclude-mapping path="/user/save"></mvc:exclude-mapping>
        <bean class="com.itheima.interceptor.MyInterceptor2"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```