SpringMVC 总结

一、三层架构和 MVC

1、三层架构：

开发架构一般基于两种形式，一种式 C/S 架构，也就是客户端/服务器；另一种是 B/S 架构，也就是浏览器/服务器

JAVAEE 中几乎全部基于 B/S 架构开发

三层架构：

表现层：web 层，它负责接收客户端发送的请求，向客户端响应结果。(SpringMVC)

业务层：service 层，它负责处理逻辑业务。(Spring)

持久层：dao 层，它负责对数据库的访问，对数据进行持久化。(MyBatis)

2、MVC

全名 Model View Controller，模型(javabean)-视图(jsp)-控制层(servlet)

二、SpringMVC 入门

1、概述：

是一种基于 JAVA 的实现 MVC 设计模式的请求驱动类型的轻量级 web 框架。是现在主流的 MVC 框架之一，全面超越 struts2。它通过一套注解使一个简单的 JAVA 类成为处理请求的控制器，而无需任何接口。

2、SpringMVC 的优势：

清晰的角色划分：

控制器(controller)

验证器(validator)

命令对象(command obect)、

表单对象(form object)

模型对象(model object)

Servlet 分发器(DispatcherServlet)

处理器映射(handler mapping)

试图解析器(view resoler)

每一个角色都可以由一个专门的对象来实现。

3、SpringMVC 和 Struts2d 的对比：

共同点：它们都是表现层框架，都是基于 MVC 模型写的

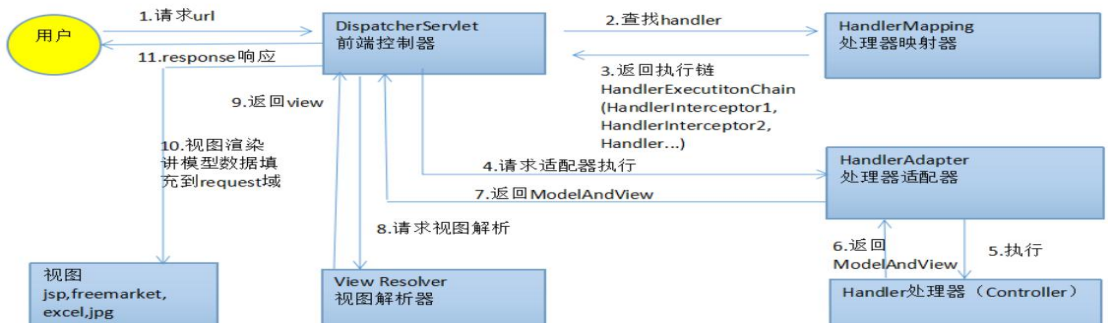
底层都离不开 ServletAPI

它们处理请求的的机制都是一个核心控制器

区别：SpringMVC 的入口是 Servlet；Struts2 的入口是 Filter

SpringMVC 是基于方法设计的(单例)；Struts2 是基于类(多例)；

4、SpringMVC 执行流程：



5、入门程序：

流程：(1)启动服务器加载一些配置

* DispatcherServlet 对象创建
* SpringMVC.xml 被加载
* Hello(控制器类)创建成对象
* InternalResourceViewResolver 视图解析器对象创建

(2)发送请求，后台处理请求

* eg：hello
* 请求到达 DispatcherServlet (控制作用 指挥中心)
* @RequestMapping(path = "/H") (请求找到需要执行的方法)

```
public String A(){
    System.out.println("张翼麒");
    return "success";
}
```

* 通过 InternalResourceViewResolver -->跳转到 success.jsp

详细步骤：(1)配置 web.xml

```
<servlet>
    <servlet-name>nb</servlet-name>
    //配置前端控制器 Servlet 对象
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        //初始化 SpringMVC.xml
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:SpringMVC.xml</param-value>
    </init-param>
    //启动服务器时候就创建前端控制器
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>nb</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

(2)配置 SpringMVC.xml

```
//开启 spring 注解
<context:component-scan base-package="com.zyq"></context:component-scan>
//创建视图解析器对象并配置属性
<bean id="f" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/page/"></property>
    <property name="suffix" value=".jsp"></property>
```

```
</bean>
//开启 SpringMVC 注解【自动配置了处理器映射器、处理器适配器】
<mvc:annotation-driven></mvc:annotation-driven>
```

(3)创建控制器(一个普通 JAVA 类)

```
//注释这是一个控制器
@Controller
public class Hello {
//通过前台能映射调用这个方法
@RequestMapping(path = "/H")
    public String A(){
        System.out.println("牛逼");
        return "success";
    }
}
```

(4)响应结果

返回的 string 通过视图解析器解析之后得到一个需要显示的页面

eg：success.jsp

6、SpringMVC 注解开发：

(1)@RequestMapping(path = "/H")

适用对象：

方法、类(若配置了类，调用方法需要带上类的路径 eg：T/T1)

注解属性：

path(value)：指定请求的 URL；eg:path ="T1"(path 为 string 数组)

method :指定访问此方法的请求方式 eg:RequestMethod.POST (RequestMethod 是枚举类直接.属性就行)

params: 指定必须传的参数；eg:params = "user=张翼麒" T1

headers :指定必须含有的请求头

三、请求参数的绑定

1、请求参数的绑定说明：

(1)表单提交的数据都是键值对的形式(p=v); user=123&pwd=123;

(2)SpringMVC 的绑定过程是把表单提交的数据，作为控制器中方法的参数进行绑定

注意：提交的参数名称==方法参数名称

(3)支持的数据类型：基本数据类型和 String JavaBean 集合类型

2、基本数据类型和 String：

提交表单数据名称==方法参数名称

区分大小写

3、绑定 JavaBean(pojo)类型：

表单传递参数的 name 名==pojo 属性名

如果 pojo 类型内包含另外的 pojo 类型，在表单传递时候需要用【pojo 属性名.属性】

4、绑定集合类型：

当一个 pojo 类型中有 list map 等集合类型属性时候；

eg:List<User> list; Map<String,User> map;

前台传递参数：

list：name="list[0].uid" name="list[0].uname"

map: name="map['one'].uid" name="map['one'].uname"

5、解决中文乱码问题：

配置 web.xml

```
<!--配置过滤器-->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <!--初始化-->
        <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>
```

6、类型转换：

默认情况下 SpringMVC 会自动处理从前台传递的数据---前台传过来的全部是 String 类型；

但是经过 SpringMVC 的自动类型转换---可以将 String 转成其他的数据类型；

特殊情况 eg:

StingMVC 会把 String(yyyy/MM/dd)--->Date

但是 StingMVC 不会把 String(yyyy-MM-dd)--->Date

【这时候就需要自定义类型转换】

(1)创建 JAVA 类继承 Converter<String, Date>由 string-->Date 自定义转换

(2)配置 SpringMVC.xml 文件

```
<!--自定义类型转换器      -->
    <bean id="ZDY" class="org.springframework.context.support.ConversionServiceFactoryBean">
        <property name="converters">
            <set>
```

```
                    //自定义类的全限定路径
                        <bean class="com.zyq.utils.StringToDate"></bean>
                    </set>
                </property>
            </bean>

        //创建了自定义类型转换器后还需要使其生效：
        <mvc:annotation-driven conversion-service="ZDY">
```

【还有一个简单的方式--注解】 加在 JavaBean Date 属性类型上即可

```
@DateTimeFormat(pattern = "yyyy-MM-dd")
```

7、在控制器中获取 ServletAPI：

```
public String T5(HttpServletRequest request, HttpServletResponse response)
```

只需要传入参数 HttpServletRequest、HttpServletResponse

四、常用注解：

1、@RequestParam

内置属性：

```
@AliasFor("name")
    String value() default "";
@AliasFor("value")
    String name() default "";
boolean required() default true;   //此属性默认 true  默认一旦配置@RequestParam 注解
                                                    前台数据 name 必须等于注解配置 name
```

场景：适用于当前台上传数据 name!=控制器方法参数名

使用：@RequestParam(name="name"); //name 和前台一致

2、@RequestBody

作用：直接获取【请求体】的内容，得到 key=value&key=value..的数据

注意：get 请求方式不适用(get 方式没有请求体)

使用：eg：public String T2(@RequestBody String body)

3、@PathVariable

作用：用于绑定 URL 中的占位符；

{uid}、{uname}就是占位符

是 SpringMVC 支持 REST 风格的 URL

使用：用于接收是哟个 REST 风格传递的数据;<a href="Anno/T3/15/张翼麒">

不用像之前一样：<a href="Anno/T3?uid=15&uname=张翼麒">

eg： @RequestMapping(path = "/T3/{uid}/{uname}")

接收：eg：@PathVariable("uid") String id

注意：接收时的"uid"==path 中占位符

4、@RequestHeader

作用：用于获取请求头的值

属性：value--提供请求头的名称

注意：不常用

5、@CookieValue

作用：把指定 cookie 的值传入控制器方法的参数

属性：value--指定 cookie 的值 c

使用：eg：@CookieValue("one") String c　作为方法参数

取出"one"cookie 的值，前提是先存"one"进入 cookie

6、@ModelAttribute

作用：修饰【方法】或者【参数】

配置在方法上：该方法会在控制器中【所有】执行的方法之前执行【慎用】

属性：value--用于获取数据的 key，key 可以是 pojo 的属性，也可以是 map 的 key

适用场景：当提交表单不完整的时候，保证没有提交的数据为数据库的数据，为不是没提交就为 null

一般在这个方法里面查询数据库，查询出没有上传的数据

@ModelAttribute--修饰的方法可以【有返回值】也可以【没有返回值】

但没有返回值时候需要使用【Map】存 POJO eg：Map<String,User>

然后在请求方法参数使用@ModelAttribute("Key")修饰传入的 pojo 参数

注意：一般用于更新，参数没有传齐，需要先有一个方法查出来没齐的参数，在进行更新方法

7、@SessionAttributes

作用：用于控制器方法参数之间的共享

属性：value--用于指定存入的 key

问题:原来使用 request 域存在问题:本身使用没问题,但程序耦合度过高,SpringMVC 提供了一个类存取值 类似 session

【Model】--接口： 方法：eg:　model.addAttribute("1","张翼麒");

使用：配置在【类】上：eg：@SessionAttributes(names = {"1"})

可以把 Model 域中的数据也存在@SessionAttributes 中

这样就可以在其他的控制器方法中使用 key--"1"的数据

【ModelMap】获取：

eg：modelMap.get("1")

五、响应数据

1、返回 String

直接利用视图解析器帮助跳转页面；

eg return"success";

视图解析器回去找到配置路径下的 success.jsp 文件

2、返回 void

需要手动跳转：HttpServletResponse HttpServletRequest

请求转发：request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request,response);

请求重定向：response.sendRedirect(request.getContextPath()+"/S.jsp");

直接跳转：使用 PrintWriter writer = response.getWriter();直接打印输出

3、返回 ModelAndView

和 Model 类似

返回值 String 底层也是用的 ModelAndView 所以我们可以直接使用这个 两个类似

ModelAndView 底层把数据存入 request 域中

使用： ModelAndView mv=new ModelAndView();

存入 pojo： mv.addObject("pojo",pojo);

设置跳转(利用视图解析器)： mv.setViewName("success");

4、使用关键字跳转

请求转发：

return "forward:/WEB-INF/pages/success.jsp";

重定向：

return "redirect:/S.jsp";

六、响应 JSON 数据

1、前台发送 json 数据：

```
$.ajax({            url:"T/T5",
                    contentType:"application/json;charset=utf-8",
                    data:'{"age":20,"uname":"张翼麒","upad":"123456"}',
                    dataType:"json",
                    type:"post",
                    success:function (data) {
                        alert(data);
                        alert(data.age);
                        alert(data.uname);
                        alert(data.upad);
                    }
                });
```

2、后台接收 json 数据：

(1)public void Pojo T5(@RequestBody String body)

@RequestBody 注解：可以获取请求体的全部内容；

(2)public void Pojo T5(@RequestBody int age,String uname)

可以直接获取数据：参数名==json 属性名

(3)public void Pojo T5(@RequestBody Pojo pojo)

当 json 属性对应 pojo 时候可以用 pojo 来接收

3、响应 json 数据：

(1)当查询到数据封装到 pojo 后，需要以 json 格式返回；

(2)引入 json 依赖

```
<!--Ajax-->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.7.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>2.7.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.7.0</version>
        </dependency>
```

(3)返回：可以返回一个 pojo

也可以返回集合对象

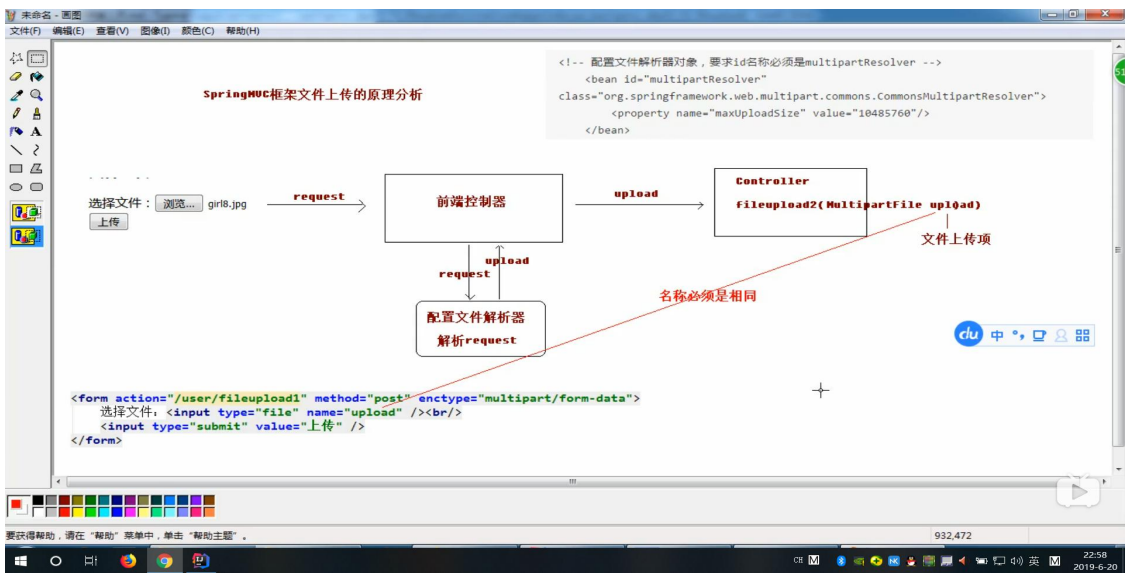
需要在【方法】上或者【返回值】标注 @ResponseBody

public @ResponseBody Pojo T5(@RequestBody Pojo pojo)

会自动转换 pojo 为 json 格式

4、前台接收 json;

七：文件上传



1、必要前提：

(1)form 表单的 enctype 取值必须是：multipart/form-data　　enctype 是表单请求正文的类型

(2)必须是 POST 方式

(3)提供文件域：<input type="file">

2、原理分析：

(1)当 enctype 的取值不是默认值后【request.getParameter()将失效】

(2)当 enctype 的取值是默认值 application/x-www-form-urlencoded 时候 form 表单正文内容为

key=value&key=value&key=value

(3)当 enctype 的取值不是默认值后；分隔符分成一部分一部分；

3、借助第三方组件实现上传：

依赖：

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>
```

4、JAVAweb 传统方式：

```
@RequestMapping(path = "/T1")
public String T1(HttpServletRequest request) throws Exception {
    System.out.println("上传文件~~~");
    //上传路径
    String path=request.getSession().getServletContext().getRealPath("/uploads");
    File file=new File(path);
    if (!file.exists()){//判断文件是否存在  不存在就创建
        file.mkdir();
    }
    //解析 request 对象，获取文件上传项　获取解析工厂
    DiskFileItemFactory factory=new DiskFileItemFactory();
    ServletFileUpload upload=new ServletFileUpload(factory);
    //开始解析 request
    List<FileItem> fileItems = upload.parseRequest(request);
    for (FileItem fileItem : fileItems) {
        //判断当前上传的是普通表单还是文件
        if (fileItem.isFormField()){
```

```
                }else{
                    String n= UUID.randomUUID().toString().replace("-","");
                    String name = n+"_"+fileItem.getName();
                    fileItem.write(new File(path,name));
                    //上传文件大于 kb 会产生临时文件  需要删除
                    fileItem.delete();
                }
            }
            return "success";
        }
```

5、SpringMVC 上传：

```
    @RequestMapping(path = "/T2")
        public String T2(HttpServletRequest request, MultipartFile up) throws Exception {
            System.out.println("SpringMVC 上传文件~~~");
            //上传路径
            String path = request.getSession().getServletContext().getRealPath("/uploads");
            File file = new File(path);
            if (!file.exists()) {//判断文件是否存在  不存在就创建
                file.mkdir();
            }
            String n = UUID.randomUUID().toString().replace("-", "");
            String name = n + "_" + up.getOriginalFilename();
            //上传
           up.transferTo(new File(path,name));
            return "success";
        }
```

注意：MultipartFile 参数名必须和前台 name 一致

6、跨服务器上传：

(1)引入依赖

```
 <!--跨服务器上传-->
            <dependency>
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-client</artifactId>
                <version>1.19</version>
            </dependency>
            <dependency>
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-core</artifactId>
                <version>1.19</version>
```

```
        </dependency>
```

(2)配置 Tomact web.xml

默认情况下 Tomact 不能跨服务器上传

```
<init-param>
    <param-name>readonly</param-name>
    <param-value>false</param-value>
</init-param>
```

(3)配置两个服务器

(4) @RequestMapping(path = "/T3")

```
public String T3(MultipartFile up) throws IOException {
    System.out.println("跨服务器上传~~~");
    //定义目标服务器的路径
    String path="http://localhost:9090/SpringMVC_03_UpLoad/UpLoad/";

    String n = UUID.randomUUID().toString().replace("-", "");
    //扩展名
    String originalFilename = up.getOriginalFilename();
    String lastname=originalFilename.substring(originalFilename.lastIndexOf("."));
    System.out.println(lastname);
    //文件名
    String filename=n+lastname;
    System.out.println(filename);
    //创建客户端对象
    Client client = Client.create();
    //和图片服务器连接
    WebResource resource = client.resource(path + filename);
    //上传
    resource.put(String.class,up.getBytes());
    return "success";
}
```

(5)启动两个服务器

从当前服务器上传文件到另一个服务器

八、SpringMVC 异常处理

1、异常处理流程图

2、自定义异常类

```
public class Error extends Exception{
    private String exception;
            ……get/set
                                        }
```

3、定义异常处理器：

```
@Component   //记得标上此主键
public class ErrorResolver implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,
Exception ex) {
        Error error = null;
        if (ex instanceof Error) {
            error = (Error) ex;
        } else {
            error = new Error("系统正在维护~~");
            System.out.println(ex.getMessage());
            System.out.println(ex.toString());
        }
        //创建 ModelAndView 对象
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("err", error.getException());
        //配置跳转
        modelAndView.setViewName("error");
        return modelAndView;
    }
}
```

4、配置错误需要跳转的页面

通过异常处理器的 ModelAndView 可以存取【异常信息】并且【跳转到自定义错误页面】

5、在 controller 层抛出异常

可以是 try-catch 捕获并且抛出自定义异常

也可以直接抛出 Exception

6、自定义异常页面使用 EL 表达式接收 ModelAndView 的数据

九、SpringMVC 拦截器

1、概述：

SpringMVC 的拦截器类似于 Servlet 开发中的过滤器 Filter,用于对 Controller 进行【预处理和后处理】

【拦截器链](Interceptor Chain): 将拦截器按一定的顺序连接成一条链。

在访问被拦截的方法或者字段时，拦截器链中的拦截器就会按照之前定义的顺序被调用。

2、拦截器和过滤器的区别

过滤器：(1)是 Servlet 的技术，适用于 JAVAWEB 的所有工程

(2)在 url-pattern 中配置了/*之后，可以对[所有]要访问的资源进行拦截

拦截器：(1)是 SpringMVC 的技术，只有使用 SpringMVC 框架在能使用

(2)只会拦截访问 Controller 的方法，如果是访问 JS\HTML\IMG\CSS 不会拦截

3、自定义拦截器类---实现 implements org.springframework.web.servlet.HandlerInterceptor

包含 3 个方法：

(1)preHandle:执行 Controller 方法前执行

返回值：true--放行

false--不放行---不放行就可以执行其他操作 eg:请求转发到其他页面

(2)postHandle：执行完 Controller 方法后，响应页面前执行

(3)afterCompletion：执行完响应页面之后执行

4、SpringMVC 配置拦截器

```
<!--配置拦截器-->
    <mvc:interceptors>
        <mvc:interceptor>
            <!--拦截哪些方法     /**拦截所有    /T/*  拦截 T 请求后的方法  -->
            <mvc:mapping path="/**"/>
            <bean class="com.zyq.interceptor.HandlerInterceptor"></bean>
            <!--不用拦截哪些方法   一般配上面即可-->
            <!--<mvc:exclude-mapping path=""></mvc:exclude-mapping>    -->
        </mvc:interceptor>
    </mvc:interceptors>
```