SpringMVC 总结

项目搭建步骤

1) 导入jar 包

```
分类
    1) spring-mvc (web.Spring-core)
    2) jackson
    3) 上传相关的
    4) 日志,测试
    5) 跨服务器上传
```

2) web.xml 配置

```
编码过滤器,
核心控制器(并配置加载springmvc.xml文件)
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <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>

  <!--配置解决中文乱码的过滤器-->
  <filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>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>
```

3 springmvc.xml

```
 <!-- 1开启注解扫描 -->
    <context:component-scan base-package="cn.itcast"/>
<!-- 2 视图解析器对象 -->
    <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

<!--3 前端控制器，哪些静态资源不拦截-->
    <mvc:resources location="/css/" mapping="/css/**"/>
    <mvc:resources location="/images/" mapping="/images/**"/>
    <mvc:resources location="/js/" mapping="/js/**"/>
<!--4,配置文件解析器对象-->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="maxUploadSize" value="10485760" />
    </bean>
<!--5,配置拦截器-->
    <mvc:interceptors>
        <!--配置拦截器-->
        <mvc:interceptor>
            <!--要拦截的具体的方法-->
            <mvc:mapping path="/user/*"/>
            <!--不要拦截的方法
            <mvc:exclude-mapping path=""/>
            -->
            <!--配置拦截器对象-->
            <bean class="cn.itcast.controller.cn.itcast.interceptor.MyInterceptor1" />
        </mvc:interceptor>

        <!--配置第二个拦截器-->
        <mvc:interceptor>
            <!--要拦截的具体的方法-->
            <mvc:mapping path="/**"/>
            <!--不要拦截的方法
            <mvc:exclude-mapping path=""/>
            -->
            <!--配置拦截器对象-->
            <bean class="cn.itcast.controller.cn.itcast.interceptor.MyInterceptor2" />
        </mvc:interceptor>
    </mvc:interceptors>
<!--6,配置自定义类型转换器-->
    <bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactoryBean">
        <property name="converters">
        <set>
            <bean class="cn.itcast.utils.StringToDateConverter"/>
        </set>
    </property>
    </bean>
```

```
<!--7, 开启SpringMVC框架注解的支持 -->
<mvc:annotation-driven conversion-service="conversionService"/>
<!--8, 异常处理器 -->
 <bean id="sysExceptionResolver" class="cn.itcast.exception.SysExceptionResolver"/>

```

4) 编写Controller

常用注解

1) RequestMapping

2) RequestBody (不适用于post 请求)

3)@RequestParam

请求参数封装

1) 基本类型 和前端表单字段名称对应

2) 封装类型 : 属性名称和 表单字段名称对应

```
用户姓名: <input type="text" name="user.uname" /><br/>
用户年龄: <input type="text" name="user.age" /><br/>
```

3) 集合:

```
用户姓名: <input type="text" name="list[0].uname" /><br/>
用户年龄: <input type="text" name="list[0].age" /><br/>

用户姓名: <input type="text" name="map['one'].uname" /><br/>
用户年龄: <input type="text" name="map['one'].age" /><br/>

  private List<User> user;
  private Map<String,User> map;
注意:
    数组形式的注意角标的使用
    map 集合封装统一个对象key 要相同
```

响应:

请求json 格式数据

```

```

```
接收即响应json
@RequestMapping("/testAjax")
    public  @ResponseBody  User testAjax(@RequestBody User user){
        System.out.println("testAjax方法执行了...");
        // 客户端发送ajax的请求，传的是json字符串，后端把json字符串封装到user对象中
        System.out.println(user);
        // 做响应，模拟查询数据库
        user.setUsername("haha");
        user.setAge(40);
        // 做响应
        return user;
    }
```

ModelAndView

```
 @RequestMapping("/testModelAndView")
    public ModelAndView testModelAndView(){
        // 创建ModelAndView对象
        ModelAndView mv = new ModelAndView();
        System.out.println("testModelAndView方法执行了...");
        // 模拟从数据库中查询出User对象
        User user = new User();
        user.setUsername("小凤");
        user.setPassword("456");
        user.setAge(30);

        // 把user对象存储到mv对象中，也会把user对象存入到request对象
        mv.addObject("user",user);

        // 跳转到哪个页面
        mv.setViewName("/index.jsp");

        return mv;
    }
```

请求重定向及转发

```
请求转发
return "forward:/WEB-INF/pages/success.jsp";
请求重定向
return "redirect:/index.jsp";
```

文件上传

```
文件上传的三个必要前提
    A form表单的enctype取值必须是：multipart/form-data
        (默认值是:application/x-www-form-urlencoded) enctype:是表单请求正文的类型
    B method属性取值必须是Post
    C 提供一个文件选择域<input type=”file” />
```

普通模式

```
<form action="/user/fileupload1" method="post" enctype="multipart/form-data">
        选择文件: <input type="file" name="upload" /><br/>
        <input type="submit" value="上传" />
</form>


@RequestMapping("/fileupload1")
    public String fileuoload1(HttpServletRequest request) throws Exception {
        System.out.println("文件上传...");

        // 使用fileupload组件完成文件上传
        // 上传的位置
        String path =
request.getSession().getServletContext().getRealPath("/uploads/");
        // 判断，该路径是否存在
        File file = new File(path);
        if(!file.exists()){
            // 创建该文件夹
            file.mkdirs();
        }

        // 解析request对象，获取上传文件项
        DiskFileItemFactory factory = new DiskFileItemFactory();
        ServletFileUpload upload = new ServletFileUpload(factory);
        // 解析request
        List<FileItem> items = upload.parseRequest(request);
        // 遍历
        for(FileItem item:items){
            // 进行判断，当前item对象是否是上传文件项
            if(item.isFormField()){
                // 说明普通表单向
            }else{
                // 说明上传文件项
                // 获取上传文件的名称
                String filename = item.getName();
                // 把文件的名称设置唯一值，uuid
                String uuid = UUID.randomUUID().toString().replace("-", "");
                filename = uuid+"_"+filename;
                // 完成文件上传
                item.write(new File(path,filename));
                // 删除临时文件
                item.delete();
            }
        }

        return "success";
    }
```

```
springmvc 文件上传
@RequestMapping("/fileupload2")
```

```
public String fileuoload2(HttpServletRequest request, MultipartFile upload) throws Exception {
        System.out.println("springmvc文件上传...");
        // 使用fileupload组件完成文件上传
        // 上传的位置
        String path = request.getSession().getServletContext().getRealPath("/uploads/");
        // 判断，该路径是否存在
        File file = new File(path);
        if(!file.exists()){
            // 创建该文件夹
            file.mkdirs();
        }
        // 说明上传文件项
        // 获取上传文件的名称
        String filename = upload.getOriginalFilename();
        // 把文件的名称设置唯一值，uuid
        String uuid = UUID.randomUUID().toString().replace("-", "");
        filename = uuid+"_"+filename;
        // 完成文件上传
        upload.transferTo(new File(path,filename));
        return "success";
    }
```

异常处理器

```
1) 自定义异常
2) 自定义异常处理器
    public class SysExceptionResolver implements HandlerExceptionResolver{
        public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {
            // 获取到异常对象
            SysException e = null;
            if(ex instanceof SysException){
                e = (SysException)ex;
            }else{
                e = new SysException("系统正在维护....");
            }
            // 创建ModelAndView对象
            ModelAndView mv = new ModelAndView();
            mv.addObject("errorMsg",e.getMessage());
            mv.setViewName("error");
            return mv;
    }}
   3) 配置异常处理器
   <bean id="sysExceptionResolver" class="cn.itcast.exception.SysExceptionResolver"/>
```

日期类型转换器

```
public class StringToDateConverter implements Converter<String,Date>{
    public Date convert(String source) {
        if(source == null){throw new RuntimeException("请您传入数据");}
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
        try {// 把字符串转换日期
            return df.parse(source);
        } catch (Exception e) {
            throw new RuntimeException("数据类型转换出现错误");
}}}
```

拦截器

```
拦截器的概念:
    Spring MVC 的处理器拦截器类似于Servlet开发中的过滤器Filter，用于对处理器进行预处理和后处理。
拦截器和过滤器的区别
    过滤器是servlet规范中的一部分，任何java web工程都可以使用。
        拦截器是SpringMVC框架自己的，只有使用了SpringMVC框架的工程才能用。
    过滤器在url-pattern中配置了/*之后，可以对所有要访问的资源拦截。
        拦截器它是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。


我们要想自定义拦截器，  要求必须实现：HandlerInterceptor接口。
```

```
/**
 * 自定义拦截器
 */
public class MyInterceptor1 implements HandlerInterceptor{

    /**
     * 预处理，controller方法执行前
     * return true 放行，执行下一个拦截器，如果没有，执行controller中的方法
     * return false不放行
     */
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        System.out.println("MyInterceptor1执行了...前1111");
        // request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request,response);
        return true;
    }
```

```xml
<!--配置拦截器-->
    <mvc:interceptors>
        <!--配置拦截器-->
        <mvc:interceptor>
        <!--要拦截的具体的方法-->
        <mvc:mapping path="/user/*"/>
        <!--不要拦截的方法
        <mvc:exclude-mapping path=""/>
        -->
        <!--配置拦截器对象-->
        <bean class="cn.itcast.controller.cn.itcast.interceptor.MyInterceptor1" />
    </mvc:interceptor>
```

```
多个拦截器方法间执行的顺序

```