

# springmvc-offcn

- 1.动态路径请求问题
- 2.springmvc存值机制迷惑点
- 3.三层架构和mvc模式的区别
- 4.打包动作
- 5.springmvc的一个请求流程
- 6.错误迷惑点响应json串
- 7.参数请求接收
- 8.自定义数据格式转换器
- 9.返回值类型注意点
- 10.mvc上传和下载于原生的区别
- 11.异常解析器
- 12.拦截器(把控执行时机)

## 1.动态路径请求问题

```
<servlet-mapping>
  <!--理论上 [/] 这玩意不会拦截jsp资源 但会拦截静态资源以及, 也不会拦截其他的动态映射请求-->
  <!--理论上 [//*] 这玩意不会拦截动态映射请求, 但会拦截静态资源以及jsp请求-->
  <servlet-name>dispatchServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

```
06-May-2022 15:52:42.839 警告 [http-nio-6060-exec-7] org.springframework.web.servlet.DispatcherServlet.noHandlerFound
No mapping for GET /test.jsp
06-May-2022 15:53:07.818 警告 [http-nio-6060-exec-10] org.springframework.web.servlet.DispatcherServlet.noHandlerFound
No mapping for GET /test.html
06-May-2022 15:53:27.603 警告 [http-nio-6060-exec-2] org.springframework.web.servlet.DispatcherServlet.noHandlerFound
No mapping for GET /main.jsp
null
null
1 /*访问到了动态路径,但项目路径下的资源均不可访问
2 转发也不行
06-May-2022 15:53:38.827 警告 [http-nio-6060-exec-4] org.springframework.web.servlet.DispatcherServlet.noHandlerFound
No mapping for GET /main.jsp
```

注意我们一般使用 /这玩意

## 2.springmvc存值机制迷惑点

```
}
//1.同名策略
//2.对象接收
//3.List map set 复杂对象 很简单
//存值策略 model Map ModelMap
@RequestMapping("/type")
public String type(String username, String password, Model model, ModelMap modelMap, Map map){
    model.addAttribute("username",username);
    modelMap.addAttribute("username2",username);
    map.put("username3",username);
    return "main";
}
```

- modelAndView也可以存值

### 3.三层架构和mvc模式的区别

MVC: web 工程前后端数据交互模型

- Model: 模型 数据模型
- View: 视图 前端页面 (html jsp)
- Control: 控制 控制器 (servlet)

三层架构: 适用于所有的业务软件开发

- 表现层 (View)
- 业务层 (Service)
- 持久层 (DAO)

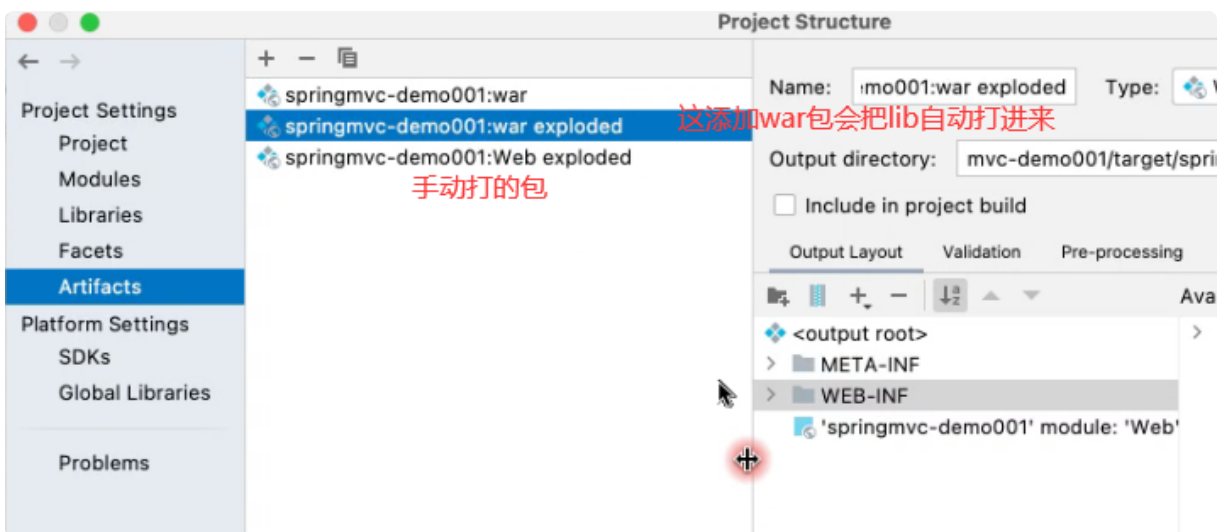
三层架构与 MVC 区别:

- Model: 数据模型
- View: 视图
- Control: 控制器

三层架构中: 业务软件 (业务开发)

- 表现层=View+Control
- Model=Service+DAO

### 4.打包动作



我个人不建议这么配置，还是用idea工具打的包

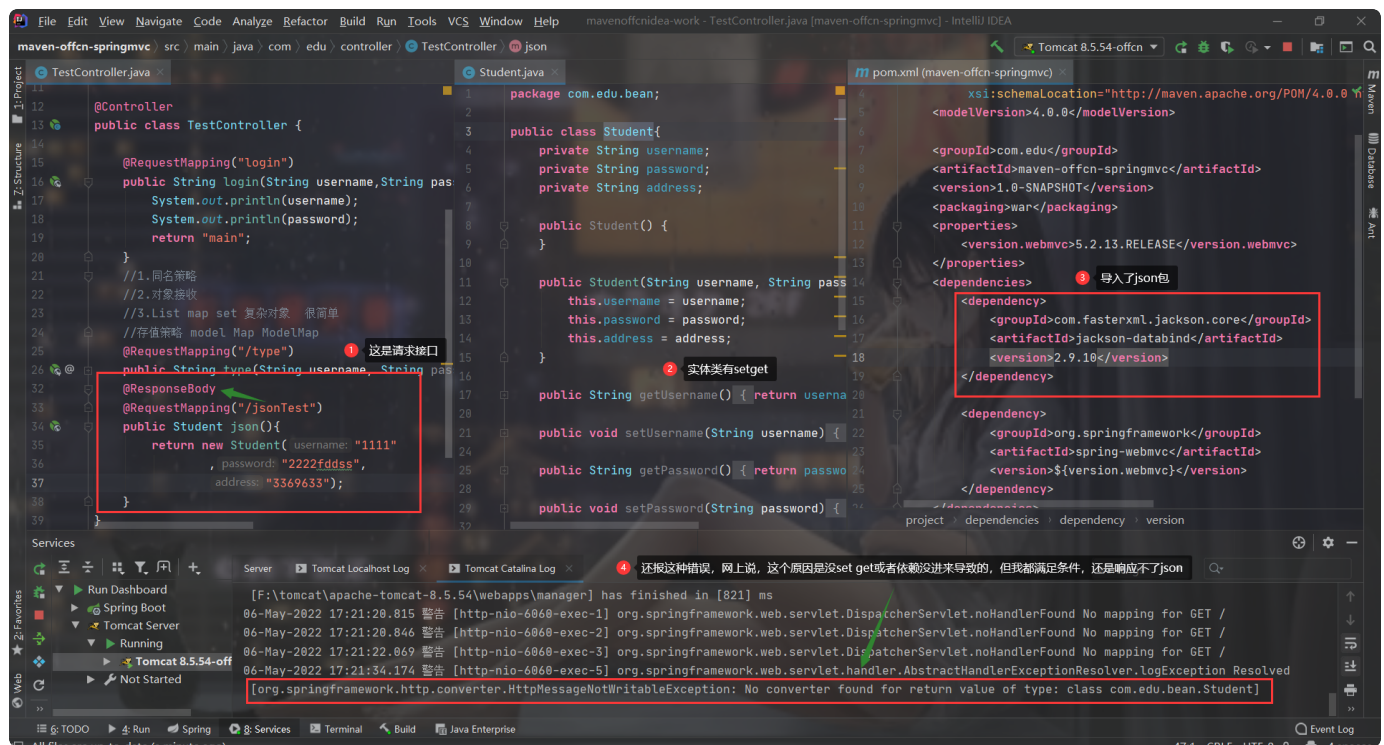
### 5.springmvc的一个请求流程

- 注意只用做到心中有数，配置变的格外简单

client发送一个请求--->dispatchServlet ----->HandlerMapper处理请求----->dispatchServlet  
 一个拦截器链----->HandlerAdpter适配器，去匹配handler----->handler----->HandlerAdpter----  
 -->dispatchServlet----->找一个modelAndView 视图解析器----->返回给dispatchServlet-----  
 -->model ----->响应

- 上面看出配置必须有这几个才能处理请求  
 dispatchServlet,HandlerMapper,HandlerAdpter,modelAndView

## 6.错误迷惑点响应json串



错误原因是没有加注解驱动；注解驱动可以替代两个handler

超说：没加注解驱动这三个地方会受到影响

Java
复制代码

1. 是配置view-controller
2. 是配置访问默认的静态资源,
3. 是处理java对象转化为json对象的时候

## 7.参数请求接收

apipost: <https://wiki.apipost.cn/document/00091641-1e36-490d-9caf-3e47cd38bcde/0d171147-ce26-4b26-bdfa-7388bf91f0d8>

```
1  @RequestMapping("/paramsEquals")
2  public String paramsEquals(String username,String password);
3  @RequestMapping("/paramsObject")
4  public String paramsObject(Student student);
5  @RequestMapping("/paramsObjects")
6  public String paramsObjects(Student student, Car car);
7  //car为student属性, 前端传值:car.name
8  @RequestMapping("/paramsStuAndCar")
9  public String paramsStuAndCar(Student student);
10 //carList为student属性, 前端传值: carList[0].name
11 @RequestMapping("/paramsStuAndCarList")
12 public String paramsStuAndCarList(Student student);
13 //carSet为student属性, 前端传值: carSet[0].name-----前提是后端先初始化好set并存
    放好对象
14 @RequestMapping("/paramsStuAndCarSet")
15 public String paramsStuAndCarSet(Student student);
16 //map为student属性, 前端传值 map["aa"].name
17 @RequestMapping("/paramsStuAndMap")
18 public String paramsStuAndMap(Student student);
19 //注解式
20 @RequestMapping(path = "/params");
21 public String params(
22     @RequestParam(name = "username") String name
23     ,@RequestParam(name = "password") String pwd
24 );
25 //map接收所有
26 @RequestMapping(path = "/paramsMap")
27 public String paramsMap(@RequestParam Map<String,String> map);
28 //路径请求
29 @GetMapping("/pathParams/{id}/{name}");
30 public String pathParams(
31     @PathVariable(name = "id",required = true) String id,
32     @PathVariable("name") String name
33 );
34 //路径map
35 @GetMapping("/pathParamsMap/{id}/{name}")
36     public String pathParamsMap(@PathVariable Map<String,String> map);
37 //json格式
38 @PostMapping("/requestBodyJson")
39     public String requestBodyJson(@RequestBody Student student);
40 //二进制格式, 但需要配置上传解析器
41 @PostMapping("/upload")
42     public String requestBodyJson(MultipartFile source);
43
```

## 8.自定义数据格式转换器

前端传过来都是字符串类型，我想转成日期类型，需要定义格式转换器，当然要在配置中注册格式转换器服务，并开启服务

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

2 开启转换器服务，可以代替两个handler,不要重复注册

```
<bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean">
    <property name="converters">
        <list>
            <bean class="com.offcn.converter.MyDateConverter"></bean>
        </list>
    </property>
</bean>
```

1 添加转换器

```
<context:component-scan base-package="com.offcn"></context:component-scan>
<mvc:annotation-driven></mvc:annotation-driven>
<mvc:default-servlet-handler></mvc:default-servlet-handler>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

1 三个handler 处理器映射器，处理器适配器，异常解析器handler

异常解析器handler handlermapper handlerAdpate



```

1 public class MyConverter implements Converter<String, Date> {
2     public Date convert(String s) {
3         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
4         Date date = null;
5         try {
6             date = simpleDateFormat.parse(s);
7         } catch (ParseException e) {
8             e.printStackTrace();
9         }
10        return date;
11    }
12 }

```

## 9.返回值类型注意点

```

@Controller
public class ForwardController {
    @RequestMapping(value = "forward")
    public String forward(){
        // 转发地址不再经过视图解析器
        return "forward:/success.jsp";
    }
}

```

① 为什么加/因为不在走视图解析器，一定是完整路径

```

//带值问题
//forward可以带request作用域的值，但重定向是不可以的
//使用session会导致服务器资源的浪费，我们采用这种方式
//http://localhost:6060/redirectAndValue
@RequestMapping("/redirectAndValue")
public String redirectAndValue(RedirectAttributes redirectAttributes){
    //redirectAttributes.addFlashAttribute("msg","li lei");
    redirectAttributes.addAttribute("msg","li lei");
    return "redirect:processRedirectAndValue";
}
@RequestMapping("/processRedirectAndValue")
public String processRedirectAndValue(@ModelAttribute(name = "msg") String msg){
    return "forward:/value.jsp";
}

```

① 调用这个接口，存一个值

② 转发到某个页面

```

@RequestMapping(value = "redirect")
public String redirect(RedirectAttributes redirectAttributes){
    redirectAttributes.addAttribute("msg", "");
    // 重定向地址不再经过视图解析器
    return "redirect:/success.jsp";
}

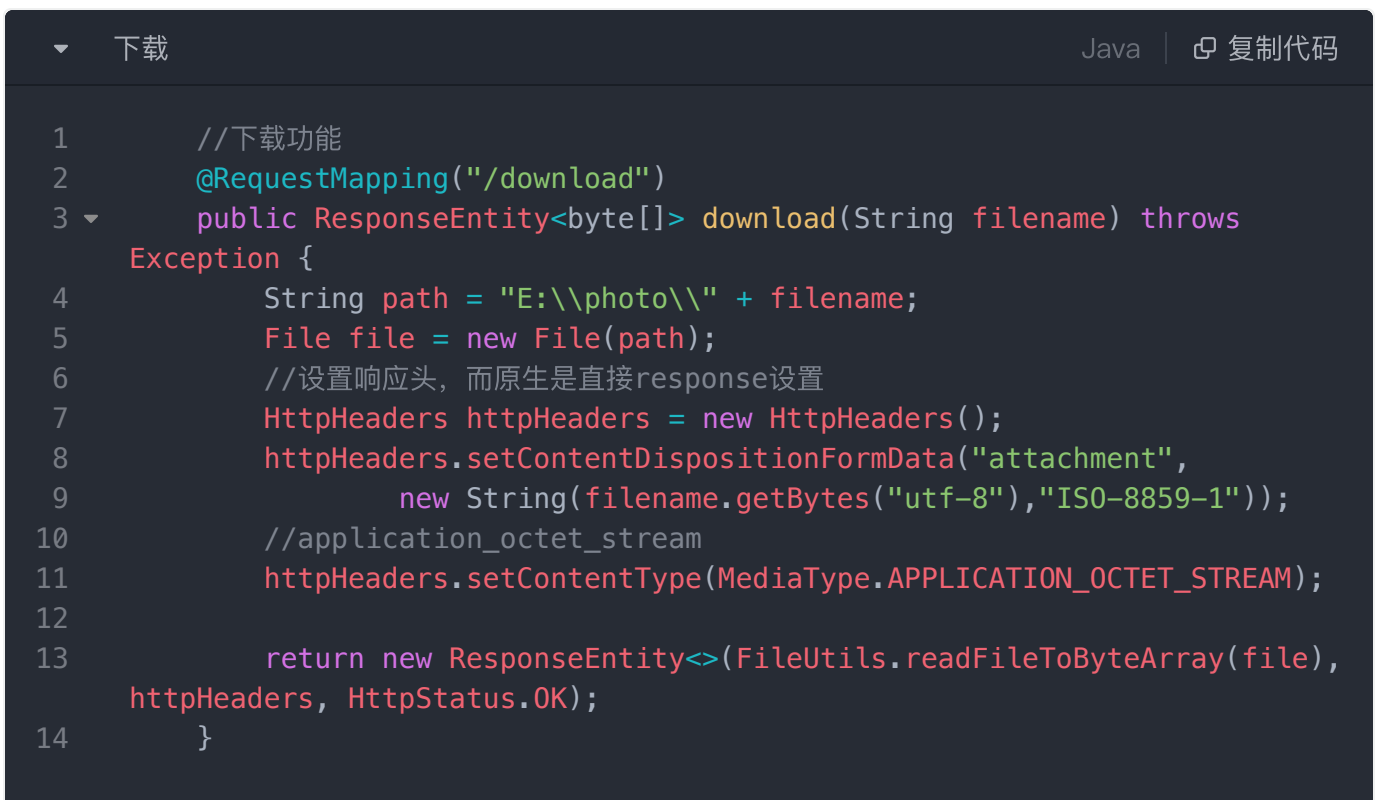
```



SUCCESS

## 10.mvc上传和下载于原生的区别

### CommonsMultipartResolver



注意如何设置响应头，如何响应前端responseEntity



```
//下载功能二
@RequestMapping("/download2")
//http://localhost:6060/download2
public ResponseEntity<byte[]> download2(String filename, HttpSession session) throws Exception {
    ServletContext servletContext = session.getServletContext();
    String realPath = servletContext.getRealPath(s: "/image/"+filename);

    String mimeType = servletContext.getMimeType(realPath);
    File file = new File(realPath);
    //设置响应头
    HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.set("Content-Type", mimeType);
    httpHeaders.setContentDispositionFormData( name: "attachment", URLEncoder.encode( s: "图片.jpg", enc: "utf-8"));
    return new ResponseEntity<>(FileUtils.readFileToByteArray(file), httpHeaders, HttpStatus.OK);
}
```

## 11.异常解析器

注解驱动已经注入了异常解析器的handler,无需我们自己在注入handler

```
@ControllerAdvice
public class MyThrowableException {
    @ExceptionHandler(value = {NullPointerException.class})
    public String nullException(){
        System.out.println("nullException");
        return "jpg";
    }
    @ExceptionHandler(value = {Throwable.class})
    public String defaultException(){
        System.out.println("defaultException");
        return "jpg";
    }
}
```

1 定义全局异常

2 与注解驱动注入的handler有关

3 先走精准异常，在走默认全局异常

## 12.拦截器(把控执行时机)

```

public class MyInterceptor implements HandlerInterceptor {
    //请求handler之前拦截
    //应用场景：权限校验
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
        HandlerMethod handlerMethod) throws Exception {
        return false;
    }

    //请求handler之后，返回之前
    //应用场景：对ModelAndView进行操作，用与给view传送公共的数据
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
        HandlerMethod handlerMethod, ModelAndView modelAndView) throws Exception {
    }

    //执行完handler返回之后
    //应用场景：统一处理异常
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
        HandlerMethod handlerMethod, Exception ex) throws Exception {
    }
}

```

```

<!--springMVC 配置文件中注册拦截器-->
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/user/login"/>
        <bean class="com.offcn.util.MyInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>

```

拦截器：/\*\*：拦截处理\*.jsp 之外的所有请求，因为 web.xml 中配置\*.jsp 执行的优先级高于/\*\*  
/\*：匹配一级，即 /add, /query 等

注意拦截器的执行顺序：可以配置多个拦截器，在配置文件中，谁在上，谁先执行。

思考：拦截器为什么不拦截jsp?

我们的tomcat开源项目中配置的加载规则，是用户项目优于，tomcat项目，用户配置了，加载用户的，用户没配置加载tomcat中web.xml的配置，所以\*.jsp的请求走的是default的servlet,而对于拦截器而言，只拦截-----dispatchServlet接收的所有请求。

```

<!-- The mapping for the default servlet -->
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- The mappings for the JSP servlet -->
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>

```