# Springboot项目整合简易说明

背景：目前项目采用springboot多模块同时开发，最后合并部署的模式，这样大大提高了开发效率，同时也对整个项目进行了解耦，对于后期的维护和管理提供了方便。分模块开发明确每个模块的功能，让开发者很容易通过模块确定自己的职责。

问题：关于springboot多模块开发的整合目前没有详细的流程。

## springboot项目多个web项目整合

主要流程：各模块开发完成后（子模块），打包成特殊的jar，并安装到本地的maven仓库，主模块引入子模块依赖完成整合。**注：该整合目前只支持多模块同一个数据源，整合后各模块访问路径为同一个端口号。**

详细说明：

### 子模块打包成特殊jar

去除子模块默认的打包依赖

<plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <executable>true</executable>  
 </configuration>  
</plugin>

加入该打包依赖

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>8</source>  
 <target>8</target>  
 </configuration>  
 </plugin>

把项目resources目录下的mapper.xml文件转移到java目录下的mapper包中，加入资源打包目录改变配置（不改变目录打包后在主项目读取不到xml文件）

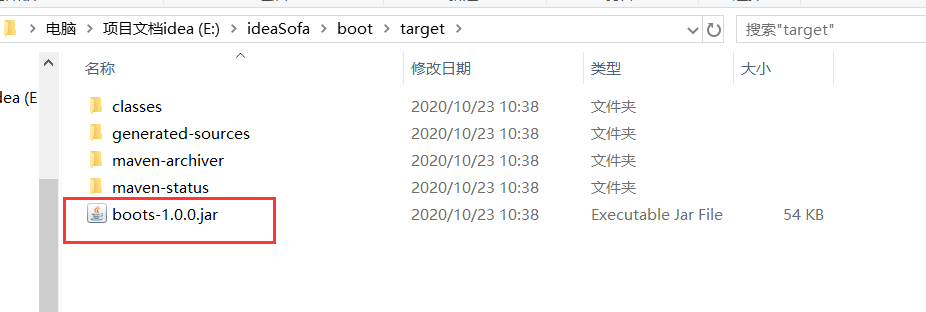
<resources>  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.xml</include>  
 </includes>  
 </resource>  
 <resource>  
 <directory>src/main/resources</directory>  
 <includes>  
 <include>\*\*/\*</include>  
 </includes>  
 </resource>  
</resources>

执行package子项目打包完成。

### 安装到本地仓库

找到子项目的打包目录：/项目名/target

示例：



在该目录cmd执行安装到本地maven命令（安装命令可自主修改，方便引入的子项目版本修改）

示例：

mvn install:install-file -DgroupId=com.example -DartifactId=boots -Dpackaging=jar  -Dversion=1.1.0 -Dfile=boots-1.0.0.jar

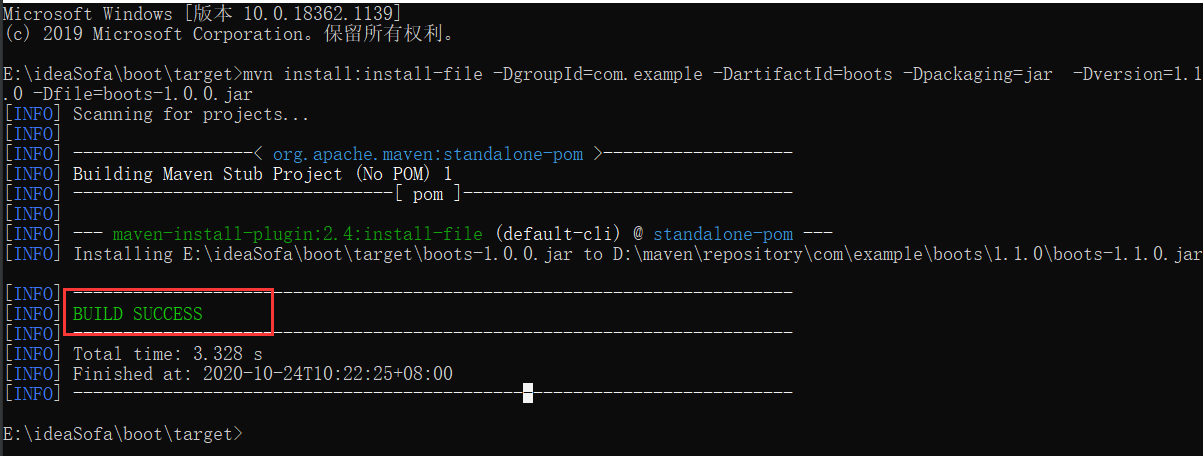
mvn install:install-file -DgroupId=com.yl -DartifactId=dlmu-core -Dpackaging=jar  -Dversion=1.2.0 -Dfile=dlmu-core-1.0.0.jar

mvn install:install-file -DgroupId=com.yl -DartifactId=dlmu-service -Dpackaging=jar  -Dversion=1.2.0 -Dfile=dlmu-service-1.0.0.jar

mvn install:install-file -DgroupId=com.yl -DartifactId=dlmu-web -Dpackaging=jar  -Dversion=1.2.0 -Dfile=dlmu-web-1.0.0.jar

mvn install:install-file -DgroupId=com.yl -DartifactId=dlmu-demo -Dpackaging=jar  -Dversion=1.2.0 -Dfile=dlmu-main-1.0.0.jar

安装完成后通过该依赖定位在本地maven的位置

<dependency>  
 <groupId>com.example</groupId> //可在安装命令修改  
 <artifactId>boots</artifactId>> //可在安装命令修改  
 <version>2.3.0</version>> //可在安装命令修改  
</dependency>

安装完成

### 主模块引用

在主项目的pom.xml中引入子模块依赖

<dependency>  
 <groupId>com.example</groupId>  
 <artifactId>boots</artifactId>  
 <version>2.3.0</version>  
</dependency>

在主模块启动项Application中，添加上扫描子模块mapper和其他包的配置

@MapperScan({"com.dlmu.\*.mapper","com.yl.boot.mapper"})  
@SpringBootApplication(scanBasePackages={"com.dlmu","com.yl.boot"})

启动项目成功，访问子项目controller成功，springboot多项目整合完成。

## springboot项目多个web项目的log整合（自定义注解aop切面log）

主要流程：打包主项目的公共工具类iap-common，该模块包含自定义注解类，安装进本地maven仓库，在子模块引入该依赖，在需要记录日志的子模块controller类上加入自定义注解@SysLogs注解。配置完成打包子项目引入主项目。

**注：该整合在主模块配置有AOP切面类Aspect，可对加入自定义注解的方法进行获取，并记录日志。**

详细说明：

### 打包公共工具类

### 详见springboot项目多web整合

### 引入子模块

### 详见springboot项目多web整合

### 配置子模块@SysLogs注解

在需要记录日志的子模块controller类方法上面加入@SysLogs注解 用title来解释该方法的作用

@GetMapping("product")  
@SysLogs(title = "依赖模块商品查询")  
public List<Product> index() {  
 //List<Product> productList=testProductService.querylist();  
 //model.addAttribute("productList", productList);  
 return testProductService.querylist();  
}

### 打包子模块引入主模块

### 详见springboot项目多web整合

## Springboot项目多个web项目的swagger2接口文档整合

主要流程:在子模块引入swagger2的依赖，在需要记录接口文档的controller类和方法上面加入swagger2注解。在主模块引入子模块依赖，在主模块配置发现子模块swagger2的注解配置。

详细说明：

### 引入swagger2依赖

引入依赖，依赖版本和主模块保持一致

<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-core</artifactId>  
 <version>2.9.2</version>  
</dependency>  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.9.2</version>  
</dependency>

### 配置子模块注解

在类上面加@Api注解 通过tags属性表明该类是什么类

@Api(tags ="其他依赖模块加入的controller")

在方法上面加@ApiOperation注解 通过value属性表明该方法作用

@ApiOperation(value = "boot模块商品查询")

配置完成

### 注入主模块

### 详见springboot项目多web整合

### 配置主模块

在主模块的swaggerConfig中配置发现多个controller配置

*/\*\*  
 \* 多个的时候 就直接添加到 pars 就可以了  
 \*/* com.google.common.base.Predicate<RequestHandler> selector1 = RequestHandlerSelectors.*basePackage*("com.dlmu.audit.controller");  
 com.google.common.base.Predicate<RequestHandler> selector2 = RequestHandlerSelectors.*basePackage*("com.dlmu.admin.controller");  
 com.google.common.base.Predicate<RequestHandler> selector3 = RequestHandlerSelectors.*basePackage*("com.yl.boot.controller");  
 pars.add(tokenPar.build());  
 //pars.add(refreshTokenPar.build());  
 return new Docket(DocumentationType.*SWAGGER\_2*)  
  
 .apiInfo(apiInfo())  
 .select()  
 //.apis(RequestHandlerSelectors.basePackage("com.dlmu.admin.controller"))  
 .apis(Predicates.*or*(selector1,selector2,selector3))  
 .paths(PathSelectors.*any*())  
 .build()  
 .globalOperationParameters(pars)  
 .enable(true)  
 ;  
}

Swagger2多个controller配置需要启动项的Application继承

SpringBootServletInitializer 类

并重写

@Override  
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler("doc.html").addResourceLocations("classpath:/META-INF/resources/");  
 registry.addResourceHandler("/webjars/\*\*").addResourceLocations("classpath:/META-INF/resources/webjars/");  
}

还需要加上

@ConditionalOnClass(SpringfoxWebMvcConfiguration.class)

注解

另外在shiro的拦截器放开

filterChainDefinitionMap.put("/doc.html", "anon");  
 filterChainDefinitionMap.put("/webjars/\*\*", "anon");  
filterChainDefinitionMap.put("/v2/api-docs", "anon");  
filterChainDefinitionMap.put("/v2/api-docs-ext", "anon");

整合完成