**Flood-Framework 开发指南**

作者：代明明

2019 年 10 月 19 日

目录

[1. 开始使用 4](#_Toc7316)

[1.1. API 规范 4](#_Toc27024)

[1.2. 基础模块 4](#_Toc14164)

[1.2.1. 异常处理 4](#_Toc20113)

[1.2.2. aop 日志打印服务 7](#_Toc27441)

[1.2.4. 公共报文响应处理服务（统一返回core包下的Result） 8](#_Toc636)

[1.2.5. RPC 远程调用响应对象处理服务 8](#_Toc30860)

[1.2.6. 应用日志打印增加线程号、sessionID 服务 8](#_Toc6198)

[1.2.7. 全局异常处理服务 8](#_Toc21143)

[1.2.8. 针对业务代码未捕获的错误信息，统一转义公共错误信息 9](#_Toc14982)

[1.3. 核心处理模块 9](#_Toc21491)

[该模块提供数据加密、banner、时间日期json等工具类的等功能； 9](#_Toc4278)

[1.3.1. 数据加密 9](#_Toc2367)

[1.3.2. 日期工具类（用的是jdk1.8以上的LocalDateTime） 11](#_Toc30042)

[1.3.3. json工具类 11](#_Toc18770)

[1.3.4. 图形验证码服务 11](#_Toc30784)

[1.3.5. 分布式唯一 ID 生成服务 12](#_Toc20455)

[1.3.6. 服务启动打印 Banner 服务 12](#_Toc11467)

[1.4. http 服务使用 12](#_Toc30471)

[1.5. Jdbc 服务使用 12](#_Toc4172)

[1.6. flood-sharding 服务使用 13](#_Toc19581)

[1.7. email 服务使用 13](#_Toc28703)

[1.8. 异步函数服务使用 15](#_Toc27657)

[1.9. JWTToken 16](#_Toc13705)

[2. 开发规范 16](#_Toc3134)

[2.1. 开发环境准备 16](#_Toc28077)

[2.2. 字段命名规范 17](#_Toc23335)

[2.2.1. 数据库 db 17](#_Toc22039)

[2.2.2. 实体 entity、vo 和 dto 17](#_Toc28959)

[2.2.3. 控制器 controller： 19](#_Toc15019)

[2.2.4. 服务层 service 19](#_Toc5008)

[2.2.5. 持久层 mapper 19](#_Toc6135)

[2.3. restful 接口设计 19](#_Toc3692)

[3. 企业级开发环境搭建 20](#_Toc19777)

[3.1. Eclipse 环境配置 20](#_Toc25240)

[3.2. 项目结构介绍 24](#_Toc17344)

[3.3. 新建 maven 项目 24](#_Toc14511)

[3.4. 引入 pom 资源 24](#_Toc23777)

[3.5. 新建启动模块 25](#_Toc17081)

[3.6. 新建业务模块 28](#_Toc18323)

[3.7. 单 DB 数据源配置 33](#_Toc19667)

[3.8. 多 DB 数据源配置 34](#_Toc7387)

[3.9. 主从数据源配置 36](#_Toc31704)

[3.10. 启动项目 37](#_Toc26192)

[4. 分布式架构环境搭建 39](#_Toc29321)

[4.1. RPC 远程调用 39](#_Toc1324)

[4.1.1. Client 端 39](#_Toc6346)

[4.1.2. Server 端 39](#_Toc26378)

[4.1.3. RPC 调用添加请求头 42](#_Toc24043)

1. **开始使用**

使用 flood 框架先引入框架的入口模块

|  |
| --- |
| <parent>  <groupId>cn.flood</groupId>  <artifactId>flood-dependencies-parent</artifactId>  <version>2.0.0</version>  </parent> |

## API 规范

使用使用 api 模块引入依赖工程即可

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-api</artifactId>  <version>${flood.version}</version>  </dependency> |

该模块是集成 swagger2 组件，提供可视化 UI， 模拟 http 请求，直观看到请求和响应参数；

## 1.2. 基础模块

使用 base 模块引入依赖工程即可

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-base</artifactId>  <version>${flood.version}</version>  </dependency> |

该模块提供统一信息返回、异常处理等功能；

### 1.2.1. 异常处理

* CoreException 的使用
* 使用方式一： 通过指定错误代码找到对应语种的解释翻译；

Code：

|  |
| --- |
| @RequestMapping("/add")  **public** void add(@ValidUserRegDTOuserRegDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userRegDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); **if**(ue1!= **null**) {  **throw new** CoreException(ErrorCode.*A11002*);  }  } |

Msg 字段翻译：\i18n\messages.properties

|  |
| --- |
| A11002=用户名已注册 |

前台收到报文格式如下

|  |
| --- |
| {  "\_code":"A11002",  "\_msg":"用户名已注册"  } |

* 使用方式二： 通过指定错误代码找到对应语种的解释翻译；

Code：

|  |
| --- |
| @RequestMapping("/add")  **public** void add(@ValidUserRegDTOuserRegDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userRegDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); **if**(ue1!= **null**) {  **throw new** CoreException(ErrorCode.*A11002,*new String[] {"spider"});  }  } |

Msg 字段翻译：\i18n\messages.properties

|  |
| --- |
| A11002={0}用户名已注册 |

前台收到报文格式如下

|  |
| --- |
| {  "\_code":"A11002",  "\_msg":"spider 用户名已注册"  } |

* 使用方式三： 自定义错误代码解释，不需要找配置文件解释翻译；

Code：

|  |
| --- |
| @RequestMapping("/add")  **public** void add(@RequestBody @ValidUserRegDTOuserRegDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userRegDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); **if**(ue1!= **null**) {  **throw new** CoreException(ErrorCode.*A11002,*  "用户名已注册");  }  } |

Msg 字段翻译：\i18n\messages.properties

|  |
| --- |
| A11002={0}用户名已注册 |

前台收到报文格式如下

|  |
| --- |
| {  "\_code":"A11002",  "\_msg":"用户名已注册"  } |

* HostException 的使用

HostException 继承了 CoreException，区别在于 HostException 会把错误信息直接抛到前

端去，用于接口远程调用；不在去找国际化字典翻译；使用如下：

Code：

|  |
| --- |
| @RequestMapping("/add")  **public** void add(@RequestBody @ValidUserRegDTOuserRegDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userRegDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); **if**(ue1!= **null**) {  **throw new** HostException(ErrorCode.*A11002,*  "用户名已注册");  }  } |

前台收到报文格式如下

|  |
| --- |
| {  "\_code":"A11002",  "\_msg":"用户名已注册"  } |

### 1.2.2. aop 日志打印服务

在实际应用开发中的 controller、servcie、dao 和 manager的 package 遵循包路径规则就

能打印日志,

Package 规则：组织+公司代号+平台+服务类型+业务类型，

* package：cn.flood.pay.controller.weixin

cn 代表中国，flood 代表公司，pay 代表平台，controller 代表服务类型，weixin 代表业务类型；

**Controller：**

规则：@Pointcut("execution(\* cn.\*.\*.controller.\*..\*.\*(..))")

示例：cn.flood.pay.controller.weixin

**Manager：**@Pointcut("execution(\* cn.\*.\*.manager.\*..\*.\*(..))")

**Service：**@Pointcut("execution(\* cn.\*.\*.service.\*..\*.\*(..))")

**Dao：**@Pointcut("execution(\* cn.\*.\*.mapper.\*..\*.\*(..))"

效果如下：

|  |
| --- |
| 2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.ControllerAspect - 【/user/get】【controller】【getUser】 start**  2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.impl.ControllerLog** **-** **request:UserEntity(user\_id=11111,**  **user\_name=null, nick\_name=null, mobilephone=null, create\_time=null, update\_time=null, password=null, password\_level=null, state=null, email=null, channel\_id=null, register\_ip=null, pwd\_changed=null, auth\_flag=null)**  2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.ServiceAspect - 【service】【getUser】 start**  2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.impl.ServiceLog** **-** **request:UserEntity(user\_id=11111,**  **user\_name=null, nick\_name=null, mobilephone=null, create\_time=null, update\_time=null, password=null, password\_level=null, state=1, email=null, channel\_id=null, register\_ip=null, pwd\_changed=null, auth\_flag=null)**  2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.DaoAspect - 【dao】【getUser】 start**  2018-03-09 06:32:59.707 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.impl.DaoLog - request:UserEntity(user\_id=11111, user\_name=null, nick\_name=null, mobilephone=null, create\_time=null, update\_time=null, password=null, password\_level=null, state=1, email=null, channel\_id=null, register\_ip=null, pwd\_changed=null, auth\_flag=null)**  2018-03-09 06:32:59.709 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.DaoAspect - 【dao】【getUser】 end,cost【2ms】**  2018-03-09 06:32:59.709 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.ServiceAspect - 【service】【getUser】 end,cost【2ms】**  2018-03-09 06:32:59.709 [http-nio-9013-exec-9] INFO  c6232ea0-e5fe-43ab-9059-4ff740af6410 C23F257FEA640D3F137E906B25475CE4  **cn.flood.aop.ControllerAspect - 【/user/get】【controller】【getUser】 end,cost 【2ms】** |

### 1.2.4. 公共报文响应处理服务（统一返回core包下的Result）

Code：

|  |
| --- |
| @RequestMapping(value="/login", method=RequestMethod.***POST***)  @ApiOperation(notes = "用户", value = "用户登录功能", httpMethod = "***POST***")  **public** Result login(@Valid LoginDTO dto) **throws** CoreException {  String password = RSAUtils.*decryptStringByJs*(dto.getPassword());  String userId = RSAUtils.*decryptStringByJs*(dto.getUserId());  **return** ResultWapper.*ok*(loginService.login(userId, password, dto.getVerifyCode(), dto.getTokenKey()));  } |

### 1.2.5. RPC 远程调用响应对象处理服务

Code：

|  |
| --- |
| @Autowired  **private** UserClient userClient;  Result result = userClient.login(userId);  **if**(!result.is\_succeed()){  **throw** **new** CoreException(result.get\_code(), result.get\_msg());  }  Map<String, Object> userMap = (Map<String, Object>) result.get\_data();  **if** (!password.equals(userMap.get("password"))) {  **throw** **new** CoreException(ErrorCode.*A00014*, "密码错误");  } |

### 1.2.6. 应用日志打印增加线程号、sessionID 服务

### 1.2.7. 全局异常处理服务

### 1.2.8. 针对业务代码未捕获的错误信息，统一转义公共错误信息

前台收到报文格式如下

|  |
| --- |
| {  "\_code":"S000000",  "\_msg":"系统内部错误"  } |

## 1.3. 核心处理模块

使用使用 core模块引入依赖工程即可

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-core</artifactId>  <version>${flood.version}</version>  </dependency> |

该模块提供数据加密、banner、时间日期json等工具类的等功能；

### 1.3.1. 数据加密

* BASE64 加密

|  |
| --- |
| String en\_value = Base64.*encode*("test".getBytes());  System.***out***.println(en\_value);//print dGVzdA== String source = **new** String(Base64.*decode*(en\_value)); System.***out***.println(source);// print test |

* DES 加密

|  |
| --- |
| String text = "A#flood#1";  //自定义 DES 密钥使用  DES crypt = **new** DES(*DESKEY*);  //使用框架中默认密钥  DES crypt1 = **new** DES();  String result = crypt.encrypt(text);  System.***out***.println(result);// print FcK1G+I8IyOB3a6CdL0ZRw== System.***out***.println(crypt.decrypt(result)); // print A#flood#1 |

* MD5 加密和签名

|  |
| --- |
| MD5 md5 = **new** MD5(); String value = "123";  // md5 加密，print：202cb962ac59075b964b07152d234b70 System.***out***.println(md5.*encrypt*(value));  String content = "test"; //待签名的明文数据  String key = "12dkjh4";  String charset = "utf-8";  String sign = md5.*sign*(content, key, charset); //md5 签名 //print 2c712e0e03f14cbf5bc9b82861f051fe System.***out***.println(sign);  **boolean** flag=md5.*verify*(content,sign,key,charset);//md5验签  //print ture  System.***out***.println(flag); |

* RSA 数据签名（128）

|  |
| --- |
| String str = **new** String(Base64.*encode*("18505933993".getBytes()));  System.***out***.println("Base64加密-->" + str);  String aaa = RSA.*sign*(  str,  *PRIVATE\_KEY*);  System.***out***.println("加签-->" + aaa);  System.***out***  .println("验签-->"  + RSA.*verify*(  str,  aaa, *PUBLIC\_KEY*));  System.***out***.println("原文-->" + **new** String(Base64.*decode*(str))); |

* RSA 数据签名（256）

|  |
| --- |
| String str = **new** String(Base64.*encode*("18505933993".getBytes()));  System.***out***.println("Base64加密-->" + str);  String aaa = RSA2.*sign*(  str,  *PRIVATE\_KEY*);  System.***out***.println("加签-->" + aaa);  System.***out***  .println("验签-->"  + RSA2.*verify*(  str,  aaa, *PUBLIC\_KEY*));  System.***out***.println("原文-->" + **new** String(Base64.*decode*(str))); |

### 1.3.2. 日期工具类（用的是jdk1.8以上的LocalDateTime）

* Code：(其他用法见工具类方法)

|  |
| --- |
| System.***out***.print(DateUtils.*getCurrentFormatDate*());//获取当前时间格式化日期  System.***out***.println(DateUtils.*getRemainingDays*());//计算当年剩余的天数 |

### 1.3.3. json工具类

* Code：(其他用法见工具类方法)

|  |
| --- |
| SysUser user = JsonUtils.*toObject*(userToken.getInfo(), SysUser.**class**);//json字符串转object对象  string param = JsonUtils.*toJson*(paramsMap);//将object对象转化为json数据 |

### 1.3.4. 图形验证码服务

* Code：(图形验证获取方法)

|  |
| --- |
| //注入服务  @Autowired  **private** TokenService tokenService;  /\*\*  \*  \* <p>Title: getCaptchaIcon</p>  \* <p>Description: 获取图像验证码</p>  \* **@param** driverId  \* **@return**  \*/  **public** CaptchaVO getCaptchaIcon(String driverId){  Token token = tokenService.getToken();  String tokenImg = "data:image/JPEG;base64,"+token.getBase64Image();  **boolean** flag = redisTokenService.saveToken(driverId, token, "image", 180);  ***log***.info("flag:{}",flag);  **return** CaptchaVO.*builder*().tokenKey(driverId)  .tokenImg(tokenImg).build();  } |

* Code：(图形验证校验)

|  |
| --- |
| Token token = redisTokenService.getTokenByRedisCache(tokenKey, "image");  **if**(token != **null**){  **if**(token.isValid(verifyCode)){  **try** {  password = MD5Util.*MD5*(password + userId);  } **catch** (Exception e) {  **throw** **new** CoreException("加密失败");  }  } **else** {  **throw** **new** CoreException(ErrorCode.*A00002*);// 验证码失效或错误  }  } **else** {  **throw** **new** CoreException(ErrorCode.*A00002*);// 验证码失效或错误  } |

### 1.3.5. 分布式唯一 ID 生成服务

* Code：(简单分布式方法)

|  |
| --- |
| //注入  @Autowired  **private** IdCreateService idCreateService;  **long id =** idCreateService.create(); |

### 1.3.6. 服务启动打印 Banner 服务

## 1.4. http 服务使用

pom.xml 引入 flood-okhttp， 目前集成的是 okhttp3 3.x 版本

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-okhttp</artifactId>  <version>${flood.version}</version>  </dependency> |

使用如下：（参考对应okhttp文档）

|  |
| --- |
| String responseData = HttpClient  // 请求方式和请求url  .get("http://localhost:8080/user-sys/user/list")  .asString() |

## 1.5. Jdbc 服务使用

pom.xml 引入 flood-database；

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-database</artifactId>  <version>${flood.version}</version>  </dependency> |

## 1.6. flood-sharding 服务使用

pom.xml 引入 flood-sharding，集成apache的shardingsphere用于读写分离及阿里的 druid 框架（sql 治理）和 pagehelper 分页组件；

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-sharding</artifactId>  <version>${flood.version}</version>  </dependency> |

## 1.7. email 服务使用

pom.xml 引入 flood-email

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-email</artifactId>  <version>${flood.version}</version>  </dependency> |

使用如下：注入 MailService 服务

|  |
| --- |
| @Autowired  **private** MailServicemailService;  @Autowired  **private** TemplateEnginetemplateEngine;  //发送一封简单文本邮件  @Test  **public void** testSimpleMail() **throws** Exception {  mailService.sendSimpleMail(**new** String[]{"77kkstyle@163.com"},**new**  String[]{"i77kkstyle@163.com"}, "test simple mail"," hello this is simple mail");  }    //发送一封简单 html 格式邮件  @Test  **public void** testHtmlMail() **throws** Exception {String content="<html>\n" +  "<body>\n" +   * <h3>hello world ! 这是一封 html 邮件!</h3>\n" + "</body>\n" +   "</html>";  mailService.sendHtmlMail(**new** String[]{"77kkstyle@163.com"},**new** String[]{"i77kkstyle@163.com"},"test simple mail",content);  }  //发送一封带附件的邮件  @Test  **public void** sendAttachmentsMail() {  String[] filePath = **new** String[]{"d:\\1\_jinjinstudy.jpg", "d:\\1\_jinjinstudy.jpg"};  mailService.sendAttachmentsMail(**new**  String[]{"77kkstyle@163.com"},**new** String[]{"i77kkstyle@163.com"},"主题：带附  件的邮件", "有附件，请查收！", filePath);  }  //发送一封将图片展示在邮件正文的邮件  @Test  **public void** sendInlineResourceMail() {  String rscId = "neo006";  String content="<html><body>这是有图片的邮件：<img src=\'cid:" + rscId   * "\' ></body></html>";   String imgPath = "d:\\1\_jinjinstudy.jpg";  List<MailAttachment> resource = **new** ArrayList<MailAttachment>();  MailAttachment ma = **new** MailAttachment();  ma.setAttachmentType(AttachmentType.***FILE***);  ma.setPath(imgPath);  ma.setId(rscId);  resource.add(ma);  mailService.sendInlineResourceMail(**new**  String[]{"77kkstyle@163.com"},**new** String[]{"i77kkstyle@163.com"}, "主题：这是有图片的邮件", content, resource);  }  //发送一封使用模板引擎邮件（模板引擎可以使用 thymeleaf 或 freemarker，用模板的主要原因是邮件正文内容可以通过 html 方式配置）  @Test    **public void** sendTemplateMail() {  //创建邮件正文  Context context = **new** Context();  context.setVariable("id", "006");  context.setVariable("title", "简单邮件");  String emailContent = templateEngine.process("emailTemplate", context);  mailService.sendHtmlMail(**new** String[]{"77kkstyle@163.com"},**new** String[]{"i77kkstyle@163.com"}, "主题：这是模板邮件",emailContent);  } |

### 1.8. 异步函数服务使用

pom.xml 引入 flood-async

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-async</artifactId>  <version>${flood.version}</version>  </dependency> |

使用如下：在函数中注解@Async("customAsync")即可

|  |
| --- |
| @Async("customAsync")  **public void** asyncTest() **throws** CoreException {System.***out***.println("--async-service-done");  } |

## 1.9. JWTToken

* pom.xml 引入 flood-auth 服务

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId> <artifactId>flood-auth</artifactId> <version>${flood.version}</version>  </dependency> |

* 注入 JwtAuth

|  |
| --- |
| @Autowired  **private** JwtAuthjwtAuth; |

* 创建 jwttoken

|  |
| --- |
| /\*  参数说明：  1、jwt 的唯一身份标识，主要用来作为一次性 token,从而回避重放攻击  2、jwt 所面向的用户或主题  3、jwt 签发者  4、jwt 的 token 有效期  \*/  String access\_token = jwtAuth.createJWT(user.getMobilephone(), "flood", "winsunion", 1000 \* 60 \* 3); |

* 校验 jwttoken

|  |
| --- |
| // uid 核验  CheckResult checkResult = jwtAuth.validateJWT(resetPwdDTO.getUid()); **if** (!checkResult.isSuccess()) {  **throw new** CoreException(ErrorCode.*A01005*);// TOKEN不存在或已失效  } |

1. **开发规范**

## 2.1. 开发环境准备

|  |  |
| --- | --- |
| 工具 |  |
| Eclipse（lombok工具） |  |
| Jdk1.8+ |  |
| Maven |  |
| Mysql 5.7以上 |  |
| Redis 5.x |  |

## 2.2. 字段命名规范

### 2.2.1. 数据库 db

数据库表名：全部使用小写的，多个单词用下划线分开；按照业务模块职能定义表前缀，如 ：

短信模块：sms\_template

表字段命名：user\_name

### 2.2.2. 实体 entity、vo 和 dto

类名：全部使用大写的，多个单词用下划线分开，类名使用大驼峰命名，如 UserEntity 字段：全部使用小写的，多个单词用下划线分开；如 user\_id 规则：dto 类需要必须实现 LogAspect 接口

示例：cn.flood.pay.entity.weixin.WxPayEntity.java

示例：cn.flood.pay.vo.weixin.WxPayVO.java

示例：cn.flood.pay.dto.weixin.WxPayDTO.java

说明：entity 为映射数据库表的实体类，dto 类为接受前端请求的实体类用于验证字段合法性；vo 则是经过业务处理后需要响应给前端的数据结构；

### 2.2.3. 控制器 controller：

类名：全部使用大写的，多个单词用下划线分开，类名使用大驼峰命名，如 UserController 函数名称：全部使用小写的

规则：@Pointcut("execution(\* cn.\*.\*.controller.\*..\*.\*(..))")

示例：cn.flood.pay.controller.weixin.WxPay.java

### 2.2.4. 服务层 service

类名：全部使用大写的，多个单词用下划线分开，类名使用大驼峰命名,如 UserService 函数：全部使用小写的，多个单词用下划线分开；如 save，update

规则：@Pointcut("execution(\* cn.\*.\*.service.\*..\*.\*(..))")

示例：cn.flood.pay.service.weixin.WxPayService.java

### 2.2.5. 持久层 mapper

类名：全部使用大写的，多个单词用下划线分开，类名使用大驼峰命名,如 UserMapper 函数：全部使用小写的，多个单词用下划线分开；如 save，update

规则：@Pointcut("execution(\* cn.\*.\*.mapper.\*..\*.\*(..))")

示例：cn.flood.pay.mapper.weixin.WxPayMapper.java

## 2.3. restful 接口设计

* 资源获取类型：

GET (选择 SELECT)：从服务器上获取一个具体的资源或者一个资源列表。

POST （创建 CREATE）： 在服务器上创建一个新的资源。

PUT （更新 UPDATE）：以整体的方式更新服务器上的一个资源。

PATCH （更新 UPDATE）：只更新服务器上一个资源的一个属性。

DELETE （删除 DELTE）：删除服务器上的一个资源。

还有两个不常用的 HTTP 动词：

HEAD ： 获取一个资源的元数据，如数据的哈希值或最后的更新时间。

OPTIONS：获取客户端能对资源做什么操作的信息。

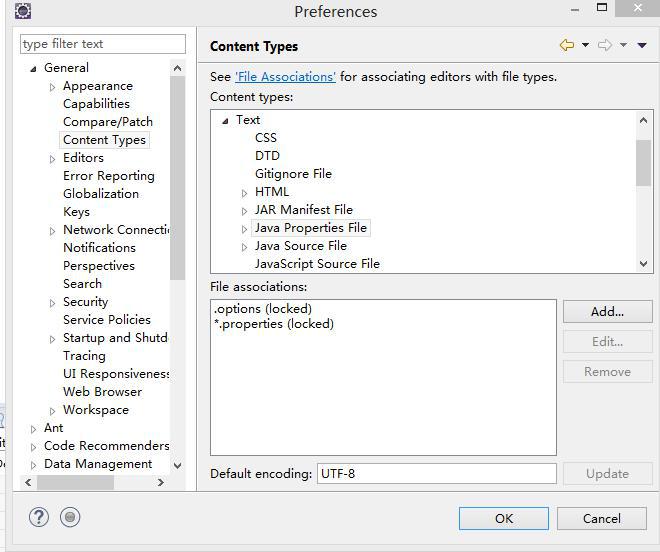
示例：

|  |
| --- |
| 用户模块-添加：POST /user  用户模块-删除：DELETE /user/{id}  用户模块-修改：PUT/user/{id}  用户模块-修改：PATCH /user/{id}  用户模块-单个查询：GET /user/{id}  用户模块-查询列表：GET /user |

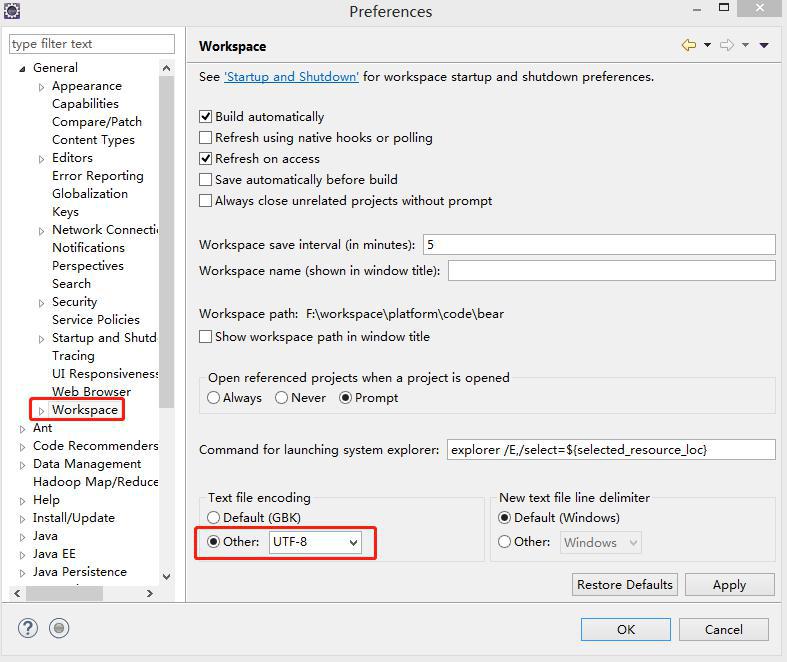
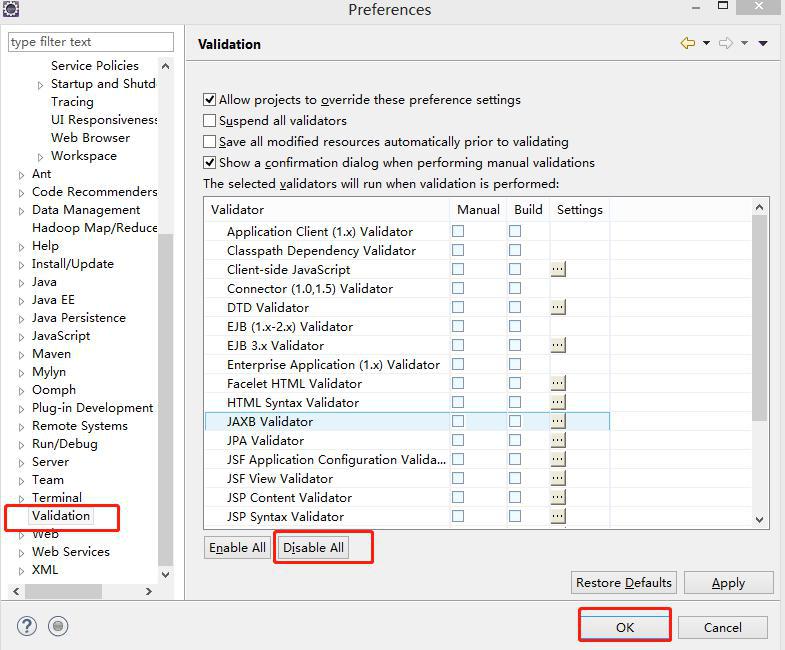
1. **企业级开发环境搭建**

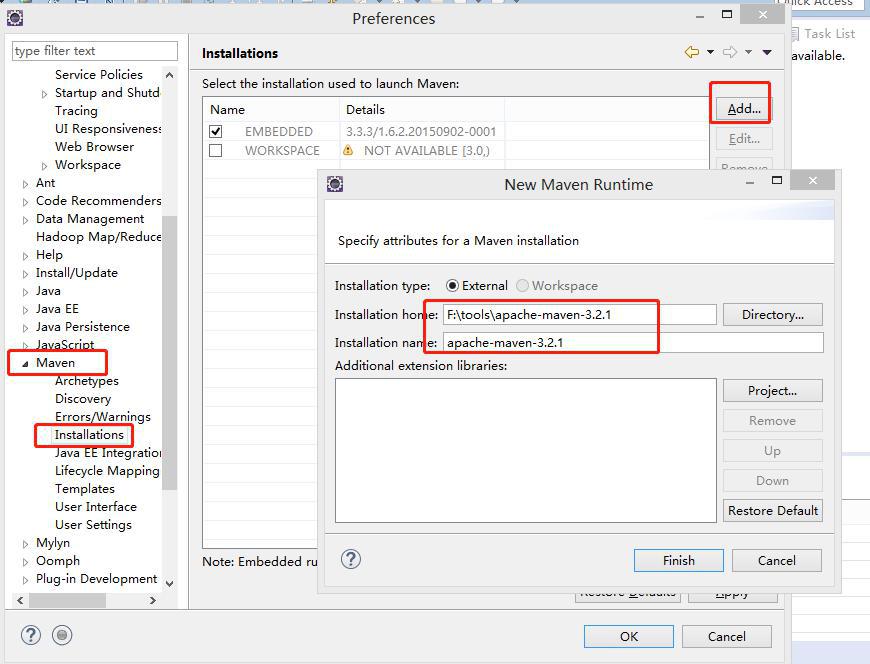
## 3.1. Eclipse 环境配置

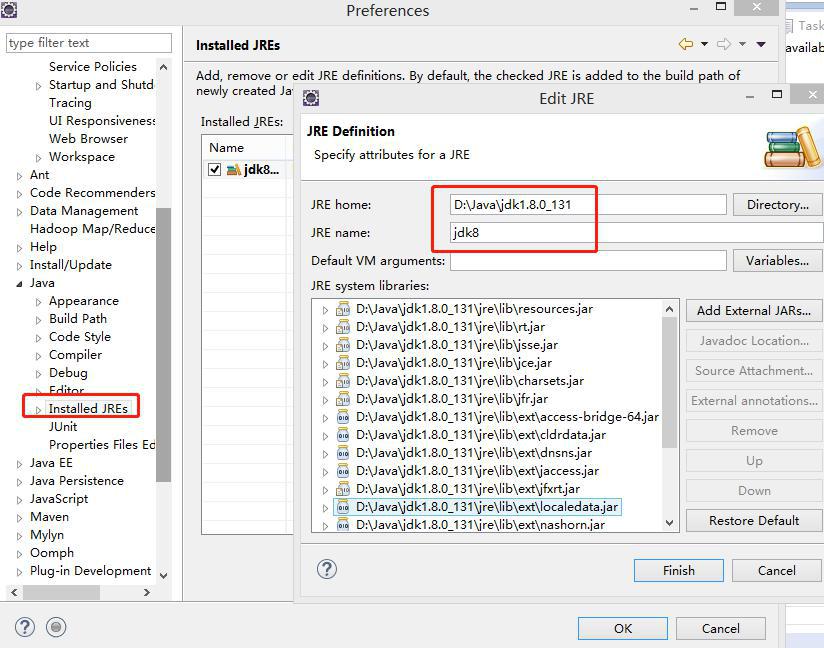
* 设置 properties 编码为 UTF-8：

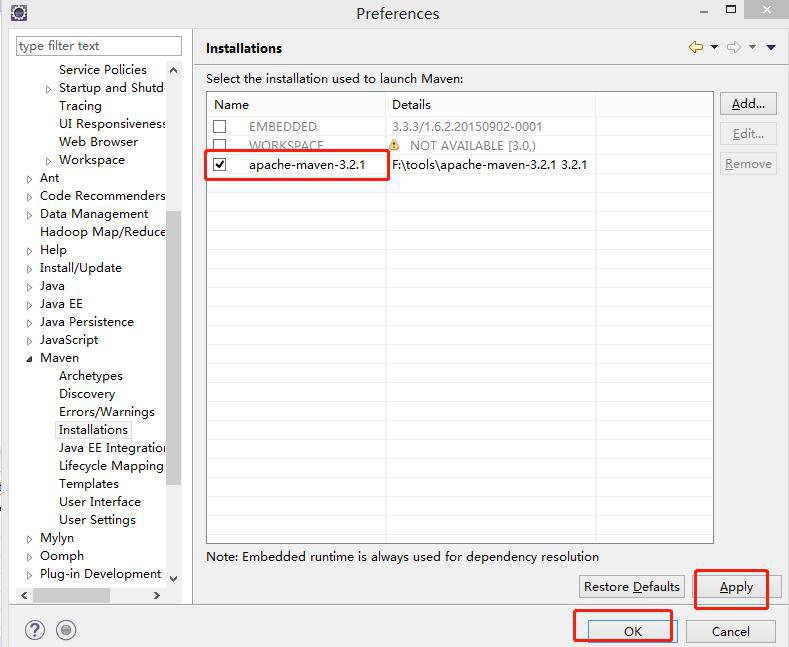
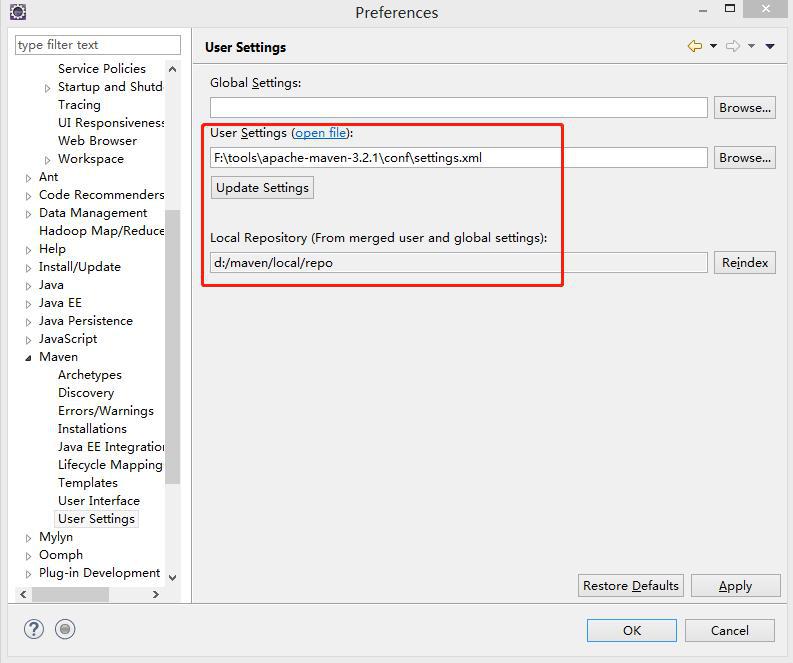


* 设置工作空间编码格式为 UTF-8



* 配置 jdk 为 1.8 或以上版本
* 配置 maven 插件
* 选择使用配置的 maven





* 配置 maven 仓库的存储目录

## 3.2. 项目结构介绍

项目结构和 package 一定要按照规范创建，否则出现日志打印不全等情况

|  |
| --- |
| bear-school  -------bear-school-server  -------bear-school-user |

## 3.3. 新建 maven 项目

|  |
| --- |
| <groupId>cn.bear.school</groupId>  <artifactId>bear-school</artifactId>  <version>1.0.0</version>  <packaging>pom</packaging> |

## 3.4. 引入 pom 资源

|  |
| --- |
| <parent>  <groupId>cn.flood</groupId>  <artifactId>flood-dependencies-parent</artifactId> <version>2.0.0</version>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncodi  ng>  <flood.version>2.0.0</flood.version>  </properties>  <dependencies>  <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-api</artifactId>  <version>${flood.version}</version>  </dependency>  <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-jdbc</artifactId>  <version>${flood.version}</version>  </dependency>  <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-core</artifactId>  <version>${flood.version}</version>  </dependency>  <dependency>  <groupId> mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <modules>  <module>bear-school-server</module>  <module>bear-school-case</module>  <module>bear-school-user</module>  <module>bear-school-discover</module>  <module>bear-school-common</module>  </modules> |

## 3.5. 新建启动模块

pom.xml 配置

|  |
| --- |
| <groupId>cn.bear.school</groupId>  <artifactId>bear-school-server</artifactId> <version>1.0.0</version> |

启动类配置

|  |
| --- |
| **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.boot.builder.SpringApplicationBuilder;  **import** org.springframework.boot.web.servlet.ServletComponentScan;  **import** org.springframework.boot.web.support.SpringBootServletInitializer;  **import** org.springframework.context.annotation.ComponentScan;  **import** org.springframework.scheduling.annotation.EnableAsync;  @MapperScan(basePackages = { "cn.bear.school.mapper"})  @ComponentScan(basePackages={"cn.flood","cn.bear.school"})  @SpringBootApplication  @EnableAsync  **public class** SchoolApplication **extends** SpringBootServletInitializer { **public static void** main(String[]args) {  **new** SpringApplicationBuilder(SchoolApplication.**class**).banner(**new** SchoolBanner()).run(args);  }  } |

启动 banner 配置

|  |
| --- |
| **import** cn.flood.banner.AbastractBanner;  **public class** SchoolBanner **extends** AbastractBanner {  **protected** String getTitle() {  **return** "(PLATFORM : BEAR-MGR-SCHOOL)";  }  } |

properties 文件配置

|  |
| --- |
| server.port=9079  #server.session.timeout=3600  #server.context-path=/servername  #server.tomcat.uri-encoding=UTF-8  #server.port=8080  #server.address= # bind to a specific NIC  #server.session-timeout= # session timeout in seconds  #the context path, defaults to '/'  #server.context-path=/spring-boot  #server.servlet-path= # the servlet path, defaults to '/'  #server.tomcat.access-log-pattern= # log pattern of the access log  #server.tomcat.access-log-enabled=false # is access logging enabled  #server.tomcat.protocol-header=x-forwarded-proto # ssl forward headers  #server.tomcat.remote-ip-header=x-forwarded-for  server.tomcat.basedir=D:/temp/school/temp  #server.tomcat.background-processor-delay=30; # in seconds  server.tomcat.max-connections=1024   * number of threads in protocol handler server.tomcat.max-threads =1024   #server.tomcat.uri-encoding = UTF-8 # character encoding to use for URLdecoding  ########################################################   * database setting.   ########################################################  spring.datasource.driverClassName=com.mysql.jdbc.Driver school.datasource.url=jdbc:mysql://127.0.0.1:3306/bear\_school?useSSL\=false &useUnicode\=true&characterEncoding\=utf-8 school.datasource.username=bear\_school school.datasource.password=A#bear\_school129  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.initialSize=2  spring.datasource.minIdle=5  spring.datasource.maxActive=10  spring.datasource.maxWait=30000  spring.datasource.timeBetweenEvictionRunsMillis=6000  spring.datasource.validationQuery=SELECT 1  spring.datasource.testWhileIdle=true  spring.datasource.testOnBorrow=false  spring.datasource.testOnReturn=false  spring.datasource.poolPreparedStatements=true  spring.datasource.maxPoolPreparedStatementPerConnectionSize=20  spring.datasource.minEvictableIdleTimeMillis=30000  spring.datasource.filters=stat,wall   * 通过 connectProperties 属性来打开 mergeSql 功能；慢 SQL 记录 spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat. slowSqlMillis=10 * 合并多个 DruidDataSource 的监控数据 #spring.datasource.useGlobalDataSourceStat=true   mybatis.mapperLocations=classpath\*:/mapper/\*Mapper.xml mybatis.configLocation=classpath:/mybatis-config.xml  ########################################################   * i18n setting.   ########################################################  #指定 message 的 basename，多个以逗号分隔，如果不加包名的话，默认从 classpath 路径开  始，默认: messages  spring.messages.basename=i18n/messages  #设定加载的资源文件缓存失效时间，-1 的话为永不过期，默认为-1 spring.messages.cache-seconds= 3600  #设定 Message bundles 的编码，默认: UTF-8  spring.messages.encoding=UTF-8  ########################################################  ### log4j setting.  ########################################################  logging.config=classpath:log4j2-dev.xml |

## 3.6. 新建业务模块

* pom.xml 配置

|  |
| --- |
| <groupId>cn.bear.school</groupId>  <artifactId>bear-school-user</artifactId>  <version>1.0.0</version> |

* 创建接口前端请求参数并验证参数合法性类：**package** cn.bear.school.dto.user;

|  |
| --- |
| **package** cn.bear.school.dto.user;  **import** java.io.Serializable;  **import** javax.validation.constraints.Pattern;  **import** cn.flood.aop.LogAspect;  **import** lombok.Data;  **import** lombok.ToString;  /\*\*  \*   * **TODO** 登录 请求验证类 * <p> * Created on 2017 年 6 月 19 日 * <p> * **@author** daimm * **@date** 2017年6月19日   \*/  @Data  @ToString  **public class** LoginDTO **implements** Serializable, LogAspect {  /\*\*  \*    \*/  **private static final long *serialVersionUID*** = -6118738380513912954L;@Pattern(regexp = "(^1[3|4|5|7|8][0-9]{9}$)", message = "手机号格式不正确  ")  **private** Stringmobilephone;//手机号码  } |

* 创建用户登录数据响应类：**package** cn.bear.school.vo.user;

|  |
| --- |
| **package** cn.bear.school.vo.user;  **import** lombok.Data;  **import** lombok.ToString;  /\*\*  \*   * **TODO** 登录 响应前端数据结构定义 * <p> * Created on 2017 年 6 月 19 日 * <p> * **@author** daimm * **@date** 2017年6月19日   \*/  @Data  @ToString  **public class** LoginVO {  **private** StringuserName;  } |

* 创建 controller 程序入口：**package** cn.bear.school.controller.user;

|  |
| --- |
| **package** cn.bear.school.controller.user;  **import** java.util.UUID;  **import** javax.servlet.http.HttpSession;  **import** javax.validation.Valid;  **import** org.springframework.beans.BeanUtils;    **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.web.bind.annotation.RequestBody; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody; **import** org.springframework.web.bind.annotation.RestController;  **import** cn.bear.school.dto.user.LoginDTO; **import** cn.bear.school.entity.user.UserEntity; **import** cn.bear.school.service.user.UserService; **import** cn.bear.school.vo.user.LoginVO;  **import** cn.flood.exception.CoreException; **import** cn.flood.service.id.IdCreateService; **import** cn.flood.service.user.BaseUserEntry; **import** cn.flood.service.user.UserRegistry; **import** lombok.extern.slf4j.Slf4j;  @RequestMapping("/pub")  @RestController  @Slf4j  **public class** PublicController {  @Autowired  **private** UserServiceuserService;  @Autowired  **private** UserRegistryuserRegistry;  @Autowired  **private** IdCreateServiceidCreateService;  @RequestMapping("/user/get")  **public** LoginVO getUser(@Valid LoginDTOloginDTO) {  ***log***.debug("id:"+idCreateService.create());  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(loginDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); LoginVO vo = **new** LoginVO(); BeanUtils.*copyProperties*(ue1, vo);  **return** vo;  }    @RequestMapping("/login")  **public** LoginVO login(@Valid LoginDTOloginDTO, HttpSessionhttpSession)  **throws** CoreException {  ***log***.info("---------------"+loginDTO.getMobilephone());  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(loginDTO, userEntity); UserEntity user = userService.login(userEntity);   * **TODO** 短信验证码校验，及密码加密user.setSource("1"); user.setUUID(UUID.*randomUUID*().toString()); user.setId(user.getUser\_id()); httpSession.setAttribute("user", user); userRegistry.setEntry(user);   LoginVO vo = **new** LoginVO();  BeanUtils.*copyProperties*(user, vo);  **return** vo;  }  @RequestMapping("/logout")  **public void** logout(HttpSessionhttpSession) **throws** CoreException {BaseUserEntry ue = (BaseUserEntry) httpSession.getAttribute("user"); **if** (ue== **null**)  **return**;  **if** (userRegistry!= **null**) {  **if** ((ue.getId() == **null** &&ue.getUUID() == **null**)   * userRegistry.containsEntry(ue.getId(), ue.getUUID()) ==   1) {  userRegistry.removeEntry(ue.getId());  httpSession.removeAttribute("user");  }  }  }  } |

* 创建用户服务类：**package** cn.bear.school.service.user;

|  |
| --- |
| **package** cn.bear.school.service.user;    **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.stereotype.Service;  **import** cn.bear.school.entity.user.UserEntity; **import** cn.bear.school.mapper.user.UserMapper;  @Service  **public class** UserService{  @Autowired  **private** UserMapperuserMapper;  **public** UserEntity getUser(UserEntityuserEntity) {  **return** userMapper.getUser(userEntity);  }  **public** UserEntity login(UserEntityuserEntity) {  **return** userMapper.getUser(userEntity);  }  } |

创建 dao 数据持久化服务：**package** cn.bear.school.mapper.user;

|  |
| --- |
| **package** cn.bear.school.mapper.user;  **import** cn.bear.school.entity.user.UserEntity;  **public interface** UserMapper {  **public** UserEntity getUser(UserEntityuserEntity);  } |

UserMapper.xml 配置路径：src\main\resources\mapper\UserMapper.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  <mapper namespace=*"cn.bear.school.mapper.user.UserMapper"* >  <resultMap id=*"getUserResult"*  type=*"cn.bear.school.entity.user.UserEntity"*>    <id property=*"user\_id"* column=*"user\_id"* />  <result property=*"user\_name"* column=*"user\_name"* />  <result property=*"nick\_name"* column=*"nick\_name"* />  <result property=*"mobilephone"* column=*"mobilephone"* />  <result property=*"create\_time"* column=*"create\_time"*  javaType=*"java.sql.Timestamp"*/>  <result property=*"password"* column=*"password"* />  <result property=*"password\_level"* column=*"password\_level"* /> <result property=*"state"* column=*"state"* /> <result property=*"email"* column=*"email"* />  <result property=*"channel\_id"* column=*"channel\_id"* /> <result property=*"register\_ip"* column=*"register\_ip"* /> <result property=*"pwd\_changed"* column=*"pwd\_changed"* /> <result property=*"auth\_flag"* column=*"auth\_flag"* />  </resultMap>  <select id=*"getUser"*  parameterType=*"cn.bear.school.entity.user.UserEntity"*  resultMap=*"getUserResult"*>  select  e.user\_id,e.user\_name,e.nick\_name,e.mobilephone,e.create\_time,e.password,e.  password\_level,e.state,e.email,e.channel\_id,e.register\_ip,e.pwd\_changed,e.a  uth\_flag  from users e where e.state=#{state}  <if test=*"mobilephone != null and mobilephone!=''"*> and e.mobilephone=#{mobilephone}</if>  <if test=*"channel\_id != null and channel\_id!=''"*> and e.channel\_id=#{channel\_id}</if>  <if test=*"user\_id != null and user\_id!=''"*> and e.user\_id=#{user\_id}</if>  <if test=*"email != null and email!=''"*> and e.email=#{email} </if> </select>  </mapper> |

## 3.7. 单 DB 数据源配置

pom.xml 引入 flood-database

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-database</artifactId>  <version>${flood.version}</version>  </dependency> |

properties 配置

|  |
| --- |
| ########################################################  ### database setting.  ########################################################  spring:  datasource:  #数据源single,multi  isSingle: true  # JDBC 配置(驱动类自动从url的mysql识别,数据源类型自动识别)  url: jdbc:mysql://10.10.10.51:3306/smeoisp\_sys?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false  username: 'win\_test'  password: 'Winsunion!@#123'  driver-class-name: com.mysql.cj.jdbc.Driver  druid:  #配置初始化大小/最小/最大  initial-size: 10  min-idle: 20  max-active: 120  #获取连接等待超时时间  max-wait: 30000  validation-query: select 1  validation-query-timeout: 5  test-on-borrow: **true**  test-on-return: **false**  test-while-idle: **true**  remove-abandoned: **true**  remove-abandoned-timeout: 120  #间隔多久进行一次检测，检测需要关闭的空闲连接  time-between-eviction-runs-millis: 30000  #一个连接在池中最小生存的时间  min-evictable-idle-time-millis: 60000  #打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false  pool-prepared-statements: **false**  max-pool-prepared-statement-per-connection-size: 20  #监控统计拦截的filters  filters: stat,wall  # 通过connectProperties属性来打开mergeSql功能；慢SQL记录  connection-properties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=10  # 合并多个DruidDataSource的监控数据  use-global-data-source-stat: **true**  pagehelper:  # 分页注册别名默认mysql（hsqldb、h2、postgresql、phoenix、mysql、mariadb、sqlite、oracle、db2、informix、sqlserver、sqlserver2012、derby、dm-达梦）  helperDialect: mysql |

## 3.8. 多 DB 数据源配置

pom.xml 引入 flood-database

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-database</artifactId>  <version>${flood.version}</version>  </dependency> |

properties 配置

|  |
| --- |
| ########################################################  ### database setting.  ########################################################  spring:  datasource:  #数据源single,multi  isSingle: false  #多数据源  sourceConfig[0]:  # JDBC 配置(驱动类自动从url的mysql识别,数据源类型自动识别)  url: jdbc:mysql://127.0.0.1:3306/winsunion\_access?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false  username: root  password: root  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.cj.jdbc.Driver  sourceConfig[1]:  # JDBC 配置(驱动类自动从url的mysql识别,数据源类型自动识别)  url: jdbc:mysql://127.0.0.1:3306/winsunion\_access?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false  username: root  password: root  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.cj.jdbc.Driver  druid:  #配置初始化大小/最小/最大  initial-size: 1  min-idle: 2  max-active: 12  #获取连接等待超时时间  max-wait: 30000  validation-query: select 1  validation-query-timeout: 5  test-on-borrow: **true**  test-on-return: **false**  test-while-idle: **true**  remove-abandoned: **true**  remove-abandoned-timeout: 120  #间隔多久进行一次检测，检测需要关闭的空闲连接  time-between-eviction-runs-millis: 30000  #一个连接在池中最小生存的时间  min-evictable-idle-time-millis: 60000  #打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false  pool-prepared-statements: **false**  max-pool-prepared-statement-per-connection-size: 20  #监控统计拦截的filters  filters: stat,wall  # 通过connectProperties属性来打开mergeSql功能；慢SQL记录  connection-properties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=10  # 合并多个DruidDataSource的监控数据  use-global-data-source-stat: **true**  pagehelper:  # 分页注册别名默认mysql（hsqldb、h2、postgresql、phoenix、mysql、mariadb、sqlite、oracle、db2、informix、sqlserver、sqlserver2012、derby、dm-达梦）  helperDialect: mysql |
| 使用说明  Mapper或者Service方法加入(对应多数据源下标)  @DataSourceAnnotation(name = "DB0") |

## 3.9. 主从数据源配置

pom.xml 引入 flood-sharding

|  |
| --- |
| <dependency>  <groupId>cn.flood</groupId>  <artifactId>flood-sharding</artifactId>  <version>${flood.version}</version>  </dependency> |

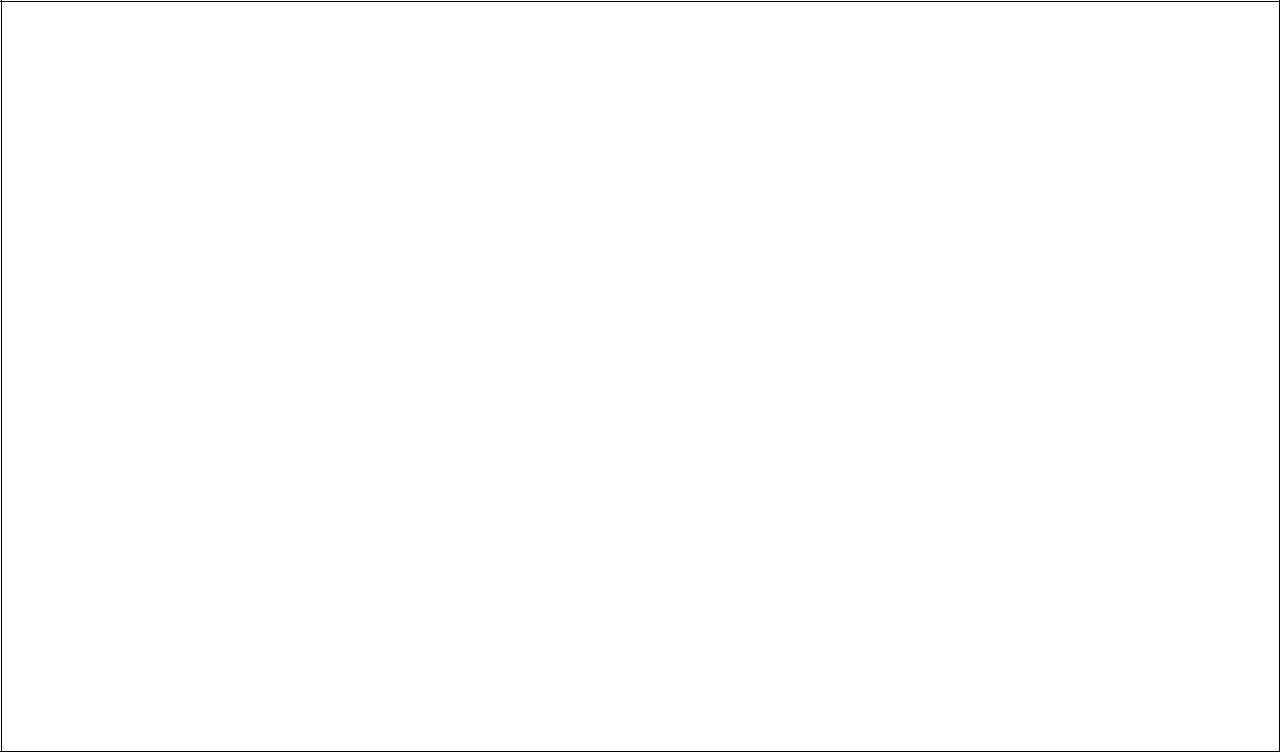
properties 配置

|  |
| --- |
| ########################################################  ###sharding database setting. 读写分离  ########################################################  sharding:  jdbc:  data-sources:  ds\_master:  driverClassName: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://localhost:3306/flood\_mgr?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false  username: 'root'  password: 'root'  ds\_slave0:  driverClassName: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://localhost:3306/sharding\_flood-mgr?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false  username: 'root'  password: 'root'  ###配置读写分离  master-slave-rule:  name: ds\_ms  ###配置从库选择策略，提供轮询与随机(ROUND\_ROBIN，RANDOM)，这里选择用轮询 如果从做了集群 查询时候做轮训查询  load-balance-algorithm-type: ROUND\_ROBIN  ####指定主数据库  masterDataSourceName: ds\_master  ####指定从数据库  slaveDataSourceNames:  - ds\_slave0  props:  sql.show: **true**  druid: #druid连接池配置  #配置初始化大小/最小/最大  initial-size: 1  min-idle: 2  max-active: 12  #获取连接等待超时时间  max-wait: 30000  validation-query: select 1  validation-query-timeout: 5  test-on-borrow: **true**  test-on-return: **false**  test-while-idle: **true**  remove-abandoned: **true**  remove-abandoned-timeout: 120  #间隔多久进行一次检测，检测需要关闭的空闲连接  time-between-eviction-runs-millis: 30000  #一个连接在池中最小生存的时间  min-evictable-idle-time-millis: 60000  #打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false  pool-prepared-statements: **false**  max-pool-prepared-statement-per-connection-size: 20  #监控统计拦截的filters  filters: stat,wall  # 通过connectProperties属性来打开mergeSql功能；慢SQL记录  connection-properties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=10  # 合并多个DruidDataSource的监控数据  use-global-data-source-stat: **true**  ---  spring:  pagehelper:  # 分页注册别名默认mysql（hsqldb、h2、postgresql、phoenix、mysql、mariadb、sqlite、oracle、db2、informix、sqlserver、sqlserver2012、derby、dm-达梦）  helperDialect: mysql  mybatis:  mapper-locations: classpath\*:/mapper/\*Mapper.xml  config-location: classpath:/mybatis-config.xml |

## 3.10. 启动项目

运行 *bear-school-server* 中的 *SchoolApplication.java* 右键 *Run as Java Application*

Banner 日志打印



(PLATFORM : BEAR-SCHOOL-SERVER)

Flood Framework Version: 1.0.0

------------------------------------------------------------

Java Home: D:\Java\jdk1.8.0\_131\jre

Java Vendor: Oracle Corporation

Java Version: 1.8.0\_131

JVM Free Memory: 121 MB

JVM Maximum Memory: 2 GB

JVM Total Memory: 151 MB

------------------------------------------------------------

OS Architecture: amd64

OS Name: Windows 8

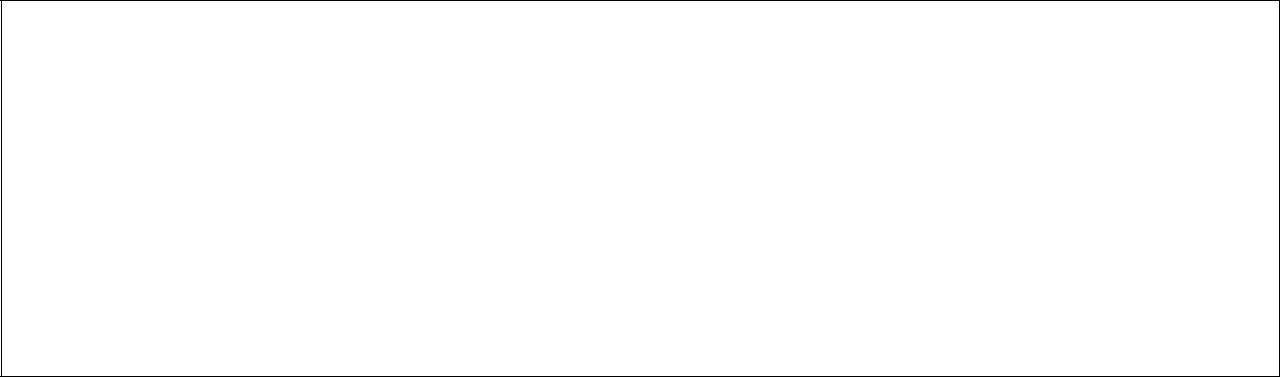
OS Version: 6.2

OS Date/Time: 2018-03-10T15:09:37.795

OS Temp Directory: C:\Users\so\AppData\Local\Temp\

------------------------------------------------------------

启动完成日志标识



2018-03-03 20:33:01.737 [main] INFO

org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainer - Tomcat started on port(s): 9011 (http)

2018-03-03 20:33:01.738 [main] INFO

org.springframework.cloud.netflix.eureka.serviceregistry.EurekaAutoServiceRegistration - Updating port to 9011

2018-03-03 20:33:01.741 [main] INFO cn.winsunion.registry.RegistryApplication - Started SchoolApplication in 6.445 seconds (JVM running for 6.928)

1. **分布式架构环境搭建**

## 4.1. RPC 远程调用

### 4.1.1. Client 端

SmsService:提供短信服务类，单独用户 Service 来包装好处是可以写一些逻辑处理，因为 rpc 服务端有响应码处理等；如果使用 SmsSendDTO 对象需要在 pom 中引入

|  |
| --- |
| <dependency>  <groupId>cn.winsunion.user</groupId>  <artifactId>winsunion-user-entity</artifactId> <version>1.0.0</version>  </dependency> |

### 4.1.2. Server 端

SmsController：

|  |
| --- |
| @RestController  @RequestMapping("/user")  **public class** UserController{  @Autowired  **private** UserServiceuserService;  @Autowired  **private** IdCreateServiceidCreateService;  @RequestMapping("/add")  **public** UserRegVO add(@RequestBody @ValidUserRegDTOuserRegDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userRegDTO, userEntity); userEntity.setState("1");  UserEntity ue1 = userService.getUser(userEntity); **if**(ue1!= **null**) {  **throw new** CoreException(ErrorCode.*A11002*);  }  UserRegVO userRegVO = **new** UserRegVO();  userEntity.setUser\_id(idCreateService.create()+""); userEntity.setState("1");  UserEntity ue = userService.add(userEntity); BeanUtils.*copyProperties*(ue, userRegVO); **return** userRegVO;  }  @RequestMapping("/login")  **public** UserLoginVO login(@RequestBody @ValidUserLoginDTOuserLoginDTO) **throws** CoreException {  UserEntity userEntity = **new** UserEntity(); BeanUtils.*copyProperties*(userLoginDTO, userEntity); userEntity.setState("1");  UserEntity ue = userService.getUser(userEntity); **if**(ue== **null**) {  **throw new** CoreException(ErrorCode.*A11001*);  }  UserLoginVO userLoginVO = **new** UserLoginVO(); BeanUtils.*copyProperties*(ue, userLoginVO); **return** userLoginVO;  }  } |

|  |
| --- |
| @Service  **public class** UserService {  @Autowired  **private** UserClientuserClient;  //用户添加  **public** Result<UserRegVO> add(UserRegDTOuserRegDTO) {  **return** userClient.add(userRegDTO);  }  //用户登录  **public** Result<UserLoginVO> login(UserLoginDTO userLoginDTO) {  **return** userClient.login(userLoginDTO);  }  } |

SmsClient：rpc 远程调用

|  |
| --- |
| @FeignClient(value = "user-server", fallback = UserHystric.**class**,  contextId = "userHystric", configuration=FeignMultipartSupportConfig.**class**)  **public interface** UserClient {  //用户添加  @RequestMapping(value = "/user/add",method = RequestMethod.***POST***) **public** Result<UserRegVO> add(UserRegDTOuserRegDTO);//用户登录  @RequestMapping(value = "/user/login",method = RequestMethod.***GET***) **public** Result<UserLoginVO> login(@SpringQueryMap UserLoginDTOuserLoginDTO);  } |

重点：@SpringQueryMap 用于fegin的get请求对象

SmsClientHystric：rpc 远程调用熔断处理逻辑

|  |
| --- |
| @Component  **public class** UserClientHystric **implements** UserClient {  **public** Result<UserRegVO> add(UserRegDTOuserRegDTO) {  **return** ResultWapper.*wrap(*"A00000", "服务器出现错误"*);*  }  **public** Result<UserLoginVO> login(UserLoginDTOuserLoginDTO) {  **return** ResultWapper.*wrap(*"A00000", "服务器出现错误"*);*  }  } |

### 4.1.3. RPC 调用添加请求头

需要实现Feign提供的一个接口RequestInterceptor，在这个类中添加请求头数据

|  |
| --- |
| @Component  **public class** FeignInterceptor **implements** RequestInterceptor{  @Override  **public void** apply(RequestTemplaterequestTemplate){requestTemplate.header("hotel\_id", "111111");  }  } |

获取请求头信息：

|  |
| --- |
| org.springframework.web.bind.annotation.RequestHeader  函数中注入@RequestHeader HttpHeaders headers 对象，从 headers 中获取 key 的值 |

|  |
| --- |
| @RequestMapping("/reg/sms/send")  **public void** reg\_sms\_send(@ValidRegSmsSendDTOregSmsSendDTO,HttpSession httpSession, @RequestHeader HttpHeaders headers)  **throws** CoreException {  //获取公共请求头信息  List<String> hotel\_id = headers.get("hotel\_id");  String str\_hotel\_id = hotel\_id.get(0);  ***log***.info("--------str\_hotel\_id:"+str\_hotel\_id);  } |