朗鹰API（LangYing API）开发手册

目录

[主要相关技术与框架 4](#_Toc452133312)

[环境 4](#_Toc452133313)

[开发环境搭建 4](#_Toc452133314)

[更改配置 4](#_Toc452133315)

[运行/调试 4](#_Toc452133316)

[测试 5](#_Toc452133317)

[目录结构 5](#_Toc452133318)

[模块总览 5](#_Toc452133319)

[框架结构说明 5](#_Toc452133320)

[代码编写说明 6](#_Toc452133321)

[创建/更改数据结构 6](#_Toc452133322)

[生成Model和Mybatis Mapper 6](#_Toc452133323)

[Model添加Swagger注解 7](#_Toc452133324)

[修改Mybatis Mapper 7](#_Toc452133325)

[Service层编写 8](#_Toc452133326)

[API接口实现 10](#_Toc452133327)

[公用类 11](#_Toc452133328)

[基类 11](#_Toc452133329)

[公用类 11](#_Toc452133330)

[约定 12](#_Toc452133331)

[事务管理 12](#_Toc452133332)

[数据分页实现 13](#_Toc452133333)

[数据验证 13](#_Toc452133334)

[访问控制 13](#_Toc452133335)

[缓存 15](#_Toc452133336)

[日志 15](#_Toc452133337)

[错误处理 16](#_Toc452133338)

[新增公用类 17](#_Toc452133339)

[新增常量/配置项 17](#_Toc452133340)

[代码注释 17](#_Toc452133341)

[Swagger 17](#_Toc452133342)

[其它 18](#_Toc452133343)

[配置文件 18](#_Toc452133344)

[打包 18](#_Toc452133345)

[代码分支管理 18](#_Toc452133346)

[公司内部Maven仓库 19](#_Toc452133347)

## 主要相关技术与框架

|  |  |
| --- | --- |
| Spring/Springboot | Java bean的管理；实现Restful API |
| Mybatis | 数据持久层框架 |
| MySQL | 数据库管理系统 |
| Redis | 缓存框架 |
| Gradle | 构建工具 |
| Swagger | Restful API文档和测试框架 |

## 环境

### 开发环境搭建

1. 下载并安装JDK 8（<http://www.oracle.com/technetwork/java/javase/downloads/index.html>）
2. 下载并安装IntelliJ IDEA（<https://www.jetbrains.com/idea/>）
3. 下载并安装Gradle并加bin目录添加进系统的path（<https://gradle.org>）
4. 下载项目源代码（<http://192.168.12.243/langying/langyingapi>）
   1. 实际开发中，需根据项目需要选择恰当的分支
5. 打开IntelliJ IDEA导入项目
   1. 选择Import Project
   2. 选择Import project from existing sources并指定代码存放目录
   3. 选择Import project from external model / gradle
   4. 指定gradle home
   5. 点击finish，等待gradle下载相关jar包并编译项目
6. 后续设置
   1. File -> Project Structure -> Project中指定JDK
   2. File -> Settings -> Editor -> File Encodings，将所有的encoding都指定为UTF8

### 更改配置

GIT中取到的默认的配置允许个人开发环境连接到测试环境的数据库运行。如有必要，参见配置部分说明。

### 运行/调试

运行gradle任务（tasks -> application -> bootRun）即可启动内置tomcat运行LangYing API应用。根据需要选择Run…或者Debug…即可。

应用入口为haozuowen\src\main\java\com\langying\Application.java

**提示：**如果要设置mybatis执行SQL的断点，可以在org.apache.ibatis.executor中寻找。如果只是要查看SQL，可观察在console中打印出的日志。

### 测试

API的测试使用Swagger

* 打开Swagger [http://192.168.12.240/swagger/](http://192.168.12.240:8081/swagger/)
* 在Swagger中输入本地应用访问地址。默认为<http://127.0.0.1:8088/v2/api-docs>
* 点击浏览，即可查看API目录
* 选择相应的API进行测试

注：生产环境和准生产环境不提供API目录信息，无法使用Swagger测试。可以用Swagger连接测试环境获得API信息后，用postman测试。

## 目录结构

### 模块总览

|  |  |
| --- | --- |
| **模块名** | **包含内容** |
| haozuowen | 包含所有API接口的实现类，及非基础模块的Model、Service和Mybatis Mapper类 |
| common | 基类，公用类，AOP定义等 |
| usermodule | 用户相关的Model、Service和Mybatis Mapper类 |
| lyresource | 朗鹰资源（如阅读内容）相关的Model、Service和Mybatis Mapper类 |
| payment | 支付相关的Model、Service类 |

即所有的API入口在haozuowen模块中，Model/Service/Mybatis Mapper分散在各模块中。common模块为业务无关的公用类。

### 框架结构说明

业务代码开发主要结构如下

|  |  |
| --- | --- |
| **层** | **位置** |
| Controller (action) | {模块}\src\main\java\com\langying\resources |
| Service | {模块}\src\main\java\com\langying\controller\service |
| Domain (model) | {模块}\src\main\java\com\langying\models |
| Persistence (Mybatis Mapper) | {模块}\src\main\java\com\langying\controller\mapper (Java)  {模块}\src\main\resources\mapper (XML) |

注：controller只存在于haozuowen模块中。

当客户端发出API请求时

* 相应的Controller将拦截到用户的Request，调用Service方法后相应给出Response。
* Service方法将通过Mybatis Mapper与数据库交互填充Model，执行业务逻辑后将数据返回给Controller。

## 代码编写说明

### 创建/更改数据结构

如果有数据结构更改的需求（新建或修改表结构），可以在测试环境中做相应修改。

如修改比较大并和当前测试环境的应用版本不兼容（理论上尽量避免），考虑启用单独的开发专用数据库。

数据更改相关SQL提交到GIT中，langyingapi\script.sql

### 生成Model和Mybatis Mapper

如果有新增表或表结构有大幅度调整，建议使用Mybatis Generator自动生成Model和Mybatis Mapper。

1. **修改Mybatis Generator配置文件加入目标表名字**

**例如：**usermodule\src\test\resources\generatorOldLangyYingConfig.xml，在加入context节点下增加

|  |
| --- |
| <table tableName="r\_user\_gold\_detail" enableSelectByExample="false" enableDeleteByExample="false" enableCountByExample="false" enableUpdateByExample="false" >  <generatedKey column="user\_gold\_detail\_id" sqlStatement="MySql" identity="true" /> </table> |

1. **运行Gradle任务**

对应之前的操作，运行gradle任务usermodule -> Tasks -> other -> mybatisGenerateOldLangYing，Mybatis Generator将自动生成model(RUserGoldDetail.java)、Java mapper(RUserGoldDetailMapper.java)和XML mapper(RUserGoldDetailMapper.xml)到相应的路径下

如果运行失败，可能是因为LangYing API开发中用的mybatis generator是经过定制的，之前获取lib时没有从公司内部服务器取得准确版本。

修复方法：从langyingapi\lib下获取mybatis-generator-core-1.3.1.jar，在IDEA中找到该jar包并定位到资源管理器中，替换。

**Gradle的Mybatis Generator的任务与配置信息如下：**

* 数据库连接信息：{模块}\src\test\resources\db.properties
* Generator配置：{模块}\src\test\resources\generator{数据库名}Config.xml

|  |  |
| --- | --- |
| Gradle任务名 | 对应数据库 |
| mybatisGenerateOldLangYing | LangYing (用户信息和主数据) |
| mybatisGenerateReading | LyResource (阅读资源库) |
| MybatisGenerateResource | LyResource (阅读资源库) |

注：每个模块下支持的Gradle Mybatis Generator任务不完全相同，视模块dependency而定

自动生成的文件存放路径为

|  |  |
| --- | --- |
| Model | {模块}\src\main\java\com\langying\models |
| Java Mapper | {模块}\src\main\java\com\langying\controller\mapper |
| XML Mapper | {模块}\src\main\resources\mapper (XML) |

注：如果目标文件本来存在，XML文件的生成将在原来文件上追加，Java文件将直接覆盖。

### Model添加Swagger注解

在Model中增加Swagger注解有助于Swagger生成相关文档，例如

|  |
| --- |
| @ApiModel(**"用户信息"**) **public class** UUser **implements** Serializable {  @ApiModelProperty(**"用户编号"**)  **private** Integer **userId**; |

### 修改Mybatis Mapper

在Mybatis Generator自动生成文件后，需要手工修改Java Mapper类，删除自动生成的方法，并使之继承相应的接口

**例如**

|  |
| --- |
| **public interface** RUserGoldDetailMapper **extends** IOldLangYingMapper<RUserGoldDetail>{} |

各基本接口与数据库对应关系如下

|  |  |  |
| --- | --- | --- |
| 接口 | 相关Annotation Interface | 数据库 |
| IOldLangYingMapper | LangYingSource | LangYing |
| IResourcesMapper | ReadingSource | LyResource |

IOldLangYingMapper与IResourceMapper都继承于IBaseMapper，提供了以下默认的方法

|  |
| --- |
| */\*\*  \* 更新记录(有效字段,即非空字段)  \*/* **int** updateByPrimaryKeySelective(T obj);  */\*\*  \* 根据主键 返回记录  \*/* T selectByPrimaryKey(Object seq);  */\*\*  \* 根据 条件返回记录  \*/* T selectByParams(@Param(value = **"params"**) Map<String, Object> params);  */\*\*  \* 查询 符合条件的记录总数  \*/* **int** selectCountByParams(@Param(value = **"params"**) Map<String, Object> params);   */\*\*  \* 分页查询 记录，分页条件为null，返回所有  \*/* List<T> selectListByParams(@Param(value = **"params"**) Map<String, Object> params,  @Param(value = **"pageOffset"**) Integer pageOffset,  @Param(value = **"pageSize"**) Integer pageSize,  @Param(value = **"orderParam"**) String orderParam); |

在此基础上，根据业务需要相应增加/修改

* Java Mapper文件中定义接口
* XML Mapper中定义具体的SQL

### Service层编写

完成Mybatis Mapper与Model的生成和编写之后，需要编写Service实现。

1. Service层实现应安置于com.{模块}.controller.service中
2. 选择相关的Service类，或新建Service类
3. 新建Service类时
   1. 用@Service标注Service类
   2. 继承于BaseService<T>
   3. @Autowired方法以注入baseMapper
   4. 创建/修改对应的Interface并申明需要暴露给外部的方法，Service类实现该Interface。
   5. Interface的命名"I" + Service类名，并继承于IBaseService<M>
   6. 如果该Service不符合常用的CRUD模式，可以不继承BaseService/IBaseService
4. 所需的Mapper使用@Autowired注入
5. Service类中添加方法前，检查BaseService是否已有类似实现

例:

*ClassesService.java*

|  |
| --- |
| @Service **public class** ClassesService **extends** BaseService<UClasses> **implements** IClassesService{  UClassesMapper **classesMapper**;   @Autowired  **void** setClassesMapper(UClassesMapper classesMapper) {  **this**.**classesMapper** = classesMapper;  **super**.**baseMapper**=classesMapper;  } |

*IClassesService.java*

|  |
| --- |
| **public interface** IClassesService **extends** IBaseService<UClasses>{ |

**IBaseService中定义了以下方法**

BaseService中实现了部分（不包括addBasic，modifyBasic和delBasic）

|  |
| --- |
| */\*\*  \* 增加记录  \*/* **public** <K> K addBasic(M obj) **throws** Exception;  */\*\*  \* 修改记录  \*/* **public void** modifyBasic(M obj) **throws** Exception;  */\*\*  \* 删除记录  \*/* **public void** delBasic(M obj) **throws** Exception;  */\*\*  \* 根据主键查询记录  \*/* **public** <K> M findObjByKey(K seq) **throws** Exception;  */\*\*  \* 根据条件查询记录  \*/* **public** Object findObj(Map<String, Object> params) **throws** Exception;   */\*\*  \* 根据条件查询 ，返回记录总数  \*/* **public** Integer findCountByParams(Map<String, Object> params);  */\*\*  \* 根据条件查询列表  \*/* **public** List<M> findListByParams(Map<String, Object> params, Order order) **throws** Exception;  */\*\*  \* 根据条件查询列表  \*/* **public** List<M> findListByParams(Map<String, Object> params, Order order, Integer maxRecord) **throws** Exception;  */\*\*  \* 根据条件查询 列表（分页查询）  \*/* **public** RollPage<M> findListPageByParams(Map<String, Object> params, Order order, Integer pageNum, Integer pageSize)**throws** Exception; |

注：见下文中的约定部分了解更多规范（事务处理、异常、日志等）

### API接口实现

Controller类实现了对用户Restful API请求的相应，并调用Service层实现业务逻辑

1. 所有Controller类安置于haozuowen\src\main\java\com\langying\resources中
2. 类名以Action结尾（CustomErrorController为特例，因为只被spring内部实现调用）
3. API以业务相关分类，每一类对应一个\*Action类，例如UserAction，TradeAction。期中CommonAction中实现跨业务模块的公用API，比如sendMessage()，securityCode()等。
4. 类使用@RestController标注
5. 需要使用的Service类用@Autowired注入
6. URL Mapping定义
   1. 每一个Action类定义第一层URL映射，比如@RequestMapping(**"/user"**)
   2. 每一个API方法定义第二层URL映射和支持的HTTP方法，比如@RequestMapping(value = **"/modifyUserInfo"**, method = RequestMethod.***POST***)
7. 每一个API方法使用注解@ResponseBody以指定返回数据写入body区域
8. 方法返回时，返回*ResponseHandler.getResponse*(object)。该方法将包装Response信息，加入标准结构，如response code, timestamp等。如果不需要返回对象，则直接使用*ResponseHandler.getResponse()*
9. 如遇错误（包括数据验证错误和异常），抛出ApiException，细节见下文中关于错误处理的约定。
10. 如果同一个model在多个API的返回中被引用，但是返回的字段不同，可以在Model中定义View的Interface，在相应字段上用@JsonView注解，在API定义时也使用@JsonView注解。参见UUser.java，LyTaskAction.java
11. Swagger文档支持
    1. 每一个Action类标注分类信息，如返回对象则同时标注类信息，比如@ApiOperation(value = **"用户信息"**, response = UUser.**class**)
    2. 每一个API方法头标注API用途，比如@ApiOperation(value = **"修改用户信息"**, notes = **"修改用户信息userGender(性别) 1:男,0:女"**)
    3. 每一个方法中的parameter标注parameter信息，比如@ApiParam(**"姓名"**) @RequestParam(value = **"userRealName"**

## 公用类

### 基类

目前共三个基类，都在common模块中

|  |  |
| --- | --- |
| com.langying.controller.mapper.IBaseMapper | Mapper公用接口，参见代码编写说明#修改Mybatis Mapper |
| com.langying.controller.service.IBaseService | Service公用接口，参见代码编写说明#Service层编写 |
| com.langying.controller.service.BaseService | Service实现基类，参见代码编写说明#Service层编写 |

### 公用类

**DataSource**

1. DataSource的Annotation定义存放于common\src\main\java\com\langying\common\models\。该定义连接了连接池配置信息与mapper类。
2. 连接池配置类存放于{模块}\src\main\java\com\langying\config\。其中环境相关的配置信息位于haozuowen\src\main\resources\application.yml
3. 每一个DataSource Annotation都有一个对应的公用interface，该interface同时继承于IBaseMapper。原则上Mapper类应该继承该公用interface，而不是直接继承于原始的annotation interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Source Annotation** | **对应的数据库** | **对应的继承了IBaseMapper的Interface** | **对应的Data Source Config类** |
| LangYingSource | LangYing | IOldLangYingMapper （usermodule模块中） | LangYingConfig |
| ReadingSource | LyResource | IResourcesMapper  （lyresource模块中） | ResourceDataSourceConfig |

**Filter类**

haozuowen\src\main\java\com\langying\filter\CorsFilter.java

针对每一个Request做通用处理，目前主要是将一些通用参数统一放到requestAttribute中，无论实际请求通过什么方式传递

**AOP**

haozuowen\src\main\java\com\langying\aspect\LogAspect.java

该类拦截了API实现方法和其它一些通用方法（定义参见类中的@Pointcut

定义），记录了通用日志信息，包括IP，用户信息，参数，使用时间等。

**Util类**

系统中的Util类如下

* common\src\main\java\com\langying\handler\PasswordHelper
* common\src\main\java\com\langying\handler\ResponseHandler: 用于API返回JSON格式Response，参见代码编写说明#API接口实现
* common\src\main\java\com\langying\toolbox\security\\*
* common\src\main\java\com\langying\toolbox\utils\\*

**注：其它公用类将在后续相关的约定中进行描述。**

## 约定

### 事务管理

框架使用了基于DataSource的事务管理，在DataSource Config类中声明了DataSourceTransactionManager并传入相应的DataSource。

在开发Service类时

1. 针对每个有数据更改的Service方法，应该使用@Transactional进行标注。
2. Transactional的参数无须特别标注（这意味着各参数将采用默认值，比如propagation将采用Propagation.REQUIRED）
3. 为提升性能，应该控制每个事务的长度，业务允许情况下尽量避免过大的事务并在同一事务中先执行读操作再执行写操作

**TODO**

|  |
| --- |
| 1. Spring事务管理默认只针对RuntimeException Rollback。目前Service代码有部分方法会抛出ApiException。须结合异常处理规范讨论 2. 需要讨论是否有必要在DataSource Config中将isolation level设置为READ\_UNCOMMITTED，并显式将DefaultAutoCommit设置为False 3. 需要讨论是否应该在类级别用@Transactional，不需要写操作的方法用readonly=true    1. 需要验证readonly = true的行为 (数据库如何处理，对性能影响，遇到read\_uncommitted 会如何，如果有修改数据的SQL会如何)    2. 需要验证readonly 的方法中调用非read only的，会如何 |

### 数据分页实现

数据分页的实现

1. 首选采用BaseService的findListByParams()
2. 其次选使用RollPage类实现，实现时需要传入Order类作为参数

### 数据验证

1. Required验证通过Controller的具体方法的参数定义使用实现，例如
   1. 如果客户端请求没有传入Required参数，系统将自动调用haozuowen\src\main\java\com\langying\exception\GlobalExceptionHandler.java，将异常转换为ApiException后返回
2. 数据格式验证，复杂的可以抽取到单独的类中实现（例如：haozuowen\src\main\java\com\langying\controller\Verification\RequestVerify.java），简单的可以直接在Controller中做判断
   1. 如果验证出错，抛出ApiException

关于ApiException的使用将在下文详细描述。

### 访问控制

框架基于Spring Security实现访问控制，主要相关的实现类为

* haozuowen\src\main\java\com\langying\config\SecurityConfig.java
* haozuowen\src\main\java\com\langying\config\SessionConfig.java
* haozuowen\src\main\java\com\langying\controller\security\AuthenticationProviderCustom.java
* haozuowen\src\main\java\com\langying\controller\security\ SimpleAuthenticationManager.java
* haozuowen\src\main\java\com\langying\controller\security\ UserSecurityService.java
* haozuowen\src\main\java\com\langying\controller\common\UserDetailHelper.java
* haozuowen\src\main\java\com\langying\resources\CustomErrorController.java

客户端请求的流程为

1. 将{用户名:密码}采用base64编码
2. 访问/user/userInfo，header中传入authentication=“Basic 用户名密码的base64编码”
3. 验证通过后，返回信息中获取token值（注：理论上带着authentication header访问任何需要访问控制的URL，都可以触发验证逻辑，但只有userInfo会返回token值）
4. 在后续的请求中，将token放入header一起发送，服务端将根据token得到用户信息供后续操作（比如验证用户是否可以访问某资源）

服务端处理流程如下

该URL是否需要访问控制？（SecurityConfig#config()中指定）

Header中是否有token

Redis中查找Token

Header中是否有authentication

Token是否有效

抛出异常，最终由CustomErrorController返回出错的JSON字串

AuthenticationProviderCustom调用UserSecurityService查询用户信息

是否有匹配的用户？

生成token，并和用户信息放入Redis

后续的API实现代码的执行。如已验证并需要使用用户信息，可调用UserDetailHelper #getUserDetail

N

Y

Y

N

N

N

Y

Y

Y

N

框架使用Redis存储token信息

* SessionConfig中指定RedisHttpSession
* SessionConfig中通过@EnableRedisHttpSession(maxInactiveIntervalInSeconds=43200)指定token过期时间

### 缓存

框架采用Redis作为缓存框架

|  |
| --- |
| **class** RedisConfig **extends** CachingConfigurerSupport{ |

**框架代码**

|  |  |
| --- | --- |
| haozuowen\src\main\java\com\langying\config\RedisConfig.java | 通知Spring采用Redis作为缓存框架  读入application.yml中定义的各个cacheName对应的expire时间并设置 |
| common\src\main\java\com\langying\common\cache\RedisCacheExt.java  common\src\main\java\com\langying\common\cache\RedisCacheManagerExt.java | 主要作用为Cache被调用后自动延长过期时间 |

**业务代码**

1. 使用Cache时，用@CachePut(cacheNames =**" "**,key=**"''+# "**)来定义如何存放数据到Cache中，用@Cacheable(cacheNames =**" "**,key=**"+# "**)来指明从Cache中获取数据。示例可参见haozuowen\src\main\java\com\langying\controller\service\HzwReportService.java
2. 一旦使用Cache，在相关的更改数据的Service方法中，一定要清空所有相关的Cache。如果是跨应用的Cache应用场景要特别注意对其它系统的影响（比如学术工具修改阅读数据，将对个人阅读系统有影响）。
3. 适合使用Cache的场景包括：
   1. 不存数据库但是需要在上下文中应用的（比如手机动态码等）
   2. 相对静态并被大量调用的数据
   3. 在某一用户使用流程中反复使用的数据
4. 如上文所述，Redis用于存放访问控制中的token和用户信息

**配置**

Redis相关配置位于haozuowen\src\main\resources\application.yml中

* Redis服务器的地址和端口
* 每个CacheName相关的失效时长

目前每个环境部署了一台Redis服务器，理想情况下应该每个环境至少部署两台并组成集群

### 日志

系统采用slf4j + logback作为日志框架。

**公用日志**

系统的公用日志包括

* 通过AOP实现的记录每个请求/方法的IP/用户/参数/耗时等。参见公用类#AOP部分
* MyBatis提供的SQL日志（在logback.xml中设置<**logger name="org.apache.ibatis" level="debug"**/>）

**业务日志**

1. 原则上系统中不应该包含System.out.println()或ex.printStacktrace()
2. 在每一个类中定义logger。例如**private static final** Logger ***logger*** = LoggerFactory.*getLogger*(GoldService.**class**);注意引用的Logger和LoggerFactory应属于org.slf4j
3. 在需要输出日志的地方，使用相应的Logger方法，比如error(msg, throwable)或info(msg)等
4. 适当的输出日志信息，将有助开发和生产环境中遇到问题时进行诊断。
5. 请恰当使用不同的Log级别。error: 出错；warning: 错误但不影响用户；info：在生产环境也应该输出的日志；debug：仅在开发时输出的日志
6. 如果输出日志级别为debug，加入前置判断**if** (***logger***.isDebugEnabled())

**配置**

1. application.yml中可以更改log的输出地址。其中logging:file用于log文件命名；logging:path用于压缩后的log存放路径
2. logback.xml配置log级别以及文件滚动规则

### 错误处理

**ApiException**

1. Controller层应将所有的错误（包括验证错误，逻辑错误，有特殊含义的异常）转换为ApiException
2. 新增错误类型时，应在haozuowen\src\main\resources\message-resCode.xml中登记code和description，在抛出异常时传入相应code，例如**throw new** ApiException(**"2001"**);
3. GlobalExceptionHandler将捕捉到异常，并调用ApiException.*getResPonse*(ex)转换为JSON的响应。
4. GlobalExceptionHandler也会把一些特定的未捕捉到的异常转换为ApiException，例如MissingServletRequestParameterException、MethodArgumentNotValidException、BindException

注：message-resCode.xml中存放的code-description mapping信息被存放在Redis中，该部分由com.langying.common.cache.ResCodeMessageCache和com.langying.config.MessageConfig实现

**异常处理原则**

1. 原则上Service和Mapper层不要忽略异常，如果捕捉了异常，应该重新抛出，到Controller层转换为ApiException
   1. 如果经过分析后确认该异常可以忽略（即应用可以继续执行后续代码，不会造成导致用户疑惑的结果，也不会破坏数据完整性），记录warning级别日志
2. 所有的异常捕捉后，应该记录日志（参见日志部分规范）

**TODO**

|  |
| --- |
| 1. 需要讨论ApiException的code的编码规范/编码区间（是否考虑每个模块有前缀） 2. 目前Service层有少许代码也抛出ApiException，一方面造成了Service层对ApiException的依赖（从命名与实现上更加适合Controller层专用），另一方面会导致事务管理中（如果按默认设置）不rollback。需要讨论是否建立特定的Servcie层异常，到Controller层再转换为ApiException |

### 新增公用类

如果需要添加公用方法/类

* 所有模块都可以使用的，创建于common\src\main\java\com\langying\toolbox\utils
* 如果是模块公用的，创建于{模块}\src\main\java\com\langying\toolbox\utils

### 新增常量/配置项

**重要！**代码中不应该直接使用Magic Number和字符串

数字或字符串常量

1. 根据使用场景定义在恰当的位置
   1. 如果是所有模块公用的，定义在common\src\main\java\com\langying\common\contant\CommonConstant.java
   2. 如果只在一个模块中公用的，定义在{模块}\src\main\java\com\langying\common\contant\CommonConstant.java
   3. 如果确实只在一个类中使用的，抽取在类的头部，采用常量命名方式
2. 如果可能，尽量定义为Java enum
3. 如果是环境相关或后续可能变化的（但不是机密信息），在application.yml中定义数值，在程序中使用@value读取。例如UserAction中@Value(**"${reading.registerUser.gold.initNum}"**)

### 代码注释

1. 方法名与变量名应尽量做到自解释
2. API文档参见Swagger部分说明
3. 内部代码中，鼓励增加适量的注释（使用//，尽量不要使用块注释）。注释的专注点应该为
   1. 比较复杂的逻辑
   2. 大段逻辑的总结
   3. 比较特殊的逻辑

### Swagger

1. Swagger 是RESTFUL接口的文档在线自动生成工具。
2. Swagger要求应用代码中按规范给出注解，参见代码编写说明#API接口实现
3. com.langying.config.DocketApiConfig将Swagger支持注入框架并提供配置信息（如apiInfo()）。
4. com.langying.Application头部的@EnableSwagger2指示Swagger扫描类并生成文档索引页（位于<http://host:port/v2/api-docs>）
5. 公司内Swagger服务位于[http://192.168.12.240/swagger/](http://192.168.12.240:8081/swagger/)，提供API文档浏览和测试功能
   1. 该Swagger服务经过了定制，允许传入token
   2. 在打包到准生产环境和生产环境时，将去除Application中的@EnableSwager2，以屏蔽Swagger的文档目录（即不再生成/v2/api-docs页面）。如果需要测试准生产环境和生产环境，可以使用postman等RESTFUL API测试工具

### 其它

以下各项无特殊约定，但在开发过程中应给予足够考虑

1. 命名规范
2. 资源的关闭（比如文件的关闭等——如果框架没有提供自动管理）
3. 线程安全
4. 性能

## 配置文件

1. 系统配置文件存放haozuowen\src\main\resources
2. Application.yml为主要配置文件，包括服务端口、数据库信息、Redis信息、日志路径和业务系统所需可配置项
3. logback.xml为日志的配置（日志级别，滚动策略等）
4. message-resCode.xml定义所有ApiException引用的错误代码和信息
5. standard.properties提供好作文报告所需的评分标准

## 打包

Jenkins为系统部署（到生产环境和准生产环境）打包时，根据环境不同，将

* 复制environment\haozuowen\{环境}\application.yml到相应目录，以覆盖环境相关设置
* 复制Application.java到相应目录，以禁止使用Swagger生成API目录文档

## 代码分支管理

所有的代码和SQL和语句应提交到GIT中，版本管理原则如下

1. 开发新模块或修改使，应从master生成新的分支，所有参与的开发都基于该分支开发，默认为”developer”。如果有多个开发任务并行进行中并关联到不同的开发计划中，则需要生成多个分支。
2. 在迭代周期中需要部署到测试环境时，应当基于开发的分支
3. 在准备正式发布时
   1. 合并到master分支
   2. 从master分支打包并部署到准生产环境，验证
   3. 用同样的包（但不同的配置文件）部署到生产环境，验证
   4. 删除原有开发分支

## 公司内部Maven仓库

为提高下载第三方类库的速度，公司内部架设了一个Maven仓库<http://192.168.12.248:8081/nexus/content/groups/public/>

如果该Maven仓库不可用，也可以考虑使用<http://maven.oschina.net/content/groups/public/，在langyingapi\build.gradle>中更新mavenLocal()的定义即可