**小课堂（知识点）**

平台开发规范

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [ ] 正式发布  [√] 正在修改 | 文件标识： | LQ-Platform-1-1 |
| 当前版本： | Version 1.0 |
| 作者： | 李强 |
| 完成日期： | 2017-06-01 |

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1.5 | 李强 | 增加规范 | 2017-06-01 | 搭建知识点框架与编写小课堂平台开发规范 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 概述 5](#_Toc504487591)

[2. 适应范围 5](#_Toc504487592)

[3. 平台架构 5](#_Toc504487593)

[3.1. 架构概述 5](#_Toc504487594)

[3.2. 架构框架图 5](#_Toc504487595)

[3.3. 平台基础模块 6](#_Toc504487596)

[3.3.1. lq-plat-core平台核心模块 6](#_Toc504487597)

[3.3.2. lq-plat-model平台模型模块 6](#_Toc504487598)

[3.3.3. lq-plat-entity 平台实体模块 6](#_Toc504487599)

[3.3.4. lq-plat-base 平台基本模块 6](#_Toc504487600)

[3.3.5. lq-plat-cache 平台缓存管理模块 6](#_Toc504487601)

[3.3.6. k-plat-captcha平台图片验证码模块 6](#_Toc504487602)

[3.3.7. lq-plat-mail平台邮件发送模块 6](#_Toc504487603)

[3.3.8. lq-plat-mq平台异步通信模块 6](#_Toc504487604)

[3.3.9. lq-plat-repository平台数据持久化模块 7](#_Toc504487605)

[3.3.10. lq-plat-repository-mongodb平台mongodb持久化模块 7](#_Toc504487606)

[3.3.11. lq-plat-search平台搜索模块 7](#_Toc504487607)

[3.3.12. lq-plat-security平台安全模块 7](#_Toc504487608)

[4. 开发规范 7](#_Toc504487609)

[4.1. 基本规范 7](#_Toc504487610)

[4.1.1. 代码组织与风格 7](#_Toc504487611)

[4.1.2. 代码模板 13](#_Toc504487612)

[4.2. 命名规范 15](#_Toc504487613)

[4.2.1. Package的命名 15](#_Toc504487614)

[4.2.2. Class的命名 16](#_Toc504487615)

[4.2.3. 类变量的命名 16](#_Toc504487616)

[4.2.4. Static Final变量的命名 16](#_Toc504487617)

[4.2.5. 参数的命名 16](#_Toc504487618)

[4.2.6. 数组的命名 16](#_Toc504487619)

[4.3. 平台模块规范 16](#_Toc504487620)

[4.3.1. 切分原则 16](#_Toc504487621)

[4.3.2. 定义原则 16](#_Toc504487622)

[4.3.3. 引用原则 16](#_Toc504487623)

[4.4. 功能模块规范 17](#_Toc504487624)

[4.5. 接口规范 17](#_Toc504487625)

[4.5.1. 设计原则 17](#_Toc504487626)

[4.5.2. 接口定义规范 18](#_Toc504487627)

[4.5.3. 接口开发规范 19](#_Toc504487628)

[4.5.4. 接口在线测试 20](#_Toc504487629)

[4.6. Service开发规范 21](#_Toc504487630)

[4.7. Repository开发规范 22](#_Toc504487631)

[4.8. Model开发规范 22](#_Toc504487632)

[4.9. Entity开发规范 22](#_Toc504487633)

[4.10. 日志记录规范 22](#_Toc504487634)

[5. 测试规范 23](#_Toc504487635)

[5.1. 单元测试 23](#_Toc504487636)

[6. 开发环境 23](#_Toc504487637)

[6.1. 开发工具 23](#_Toc504487638)

[6.2. 环境配置 23](#_Toc504487639)

[6.2.1. 系统环境 23](#_Toc504487640)

[6.2.2. 开发工具 24](#_Toc504487641)

[6.3. 代码提交规范 29](#_Toc504487642)

# 概述

为打造小课堂技术平台，为统一开发技术规范，缩短开发周期，提高平台质量，降低后期维护难度，需要一个完善、专业、有效的开发规范。 本规范的目的是使平台JAVA开发能以统一、标准的、规范的方式设计和编码。通过建立开发规范，使每个开发人员养成良好的开发风格和习惯；提高程序的可靠性、可读性、可维护性和一致性等，提高开发人员的开发水平，增进团队间的交流，并保证平台产品的质量。

# 适应范围

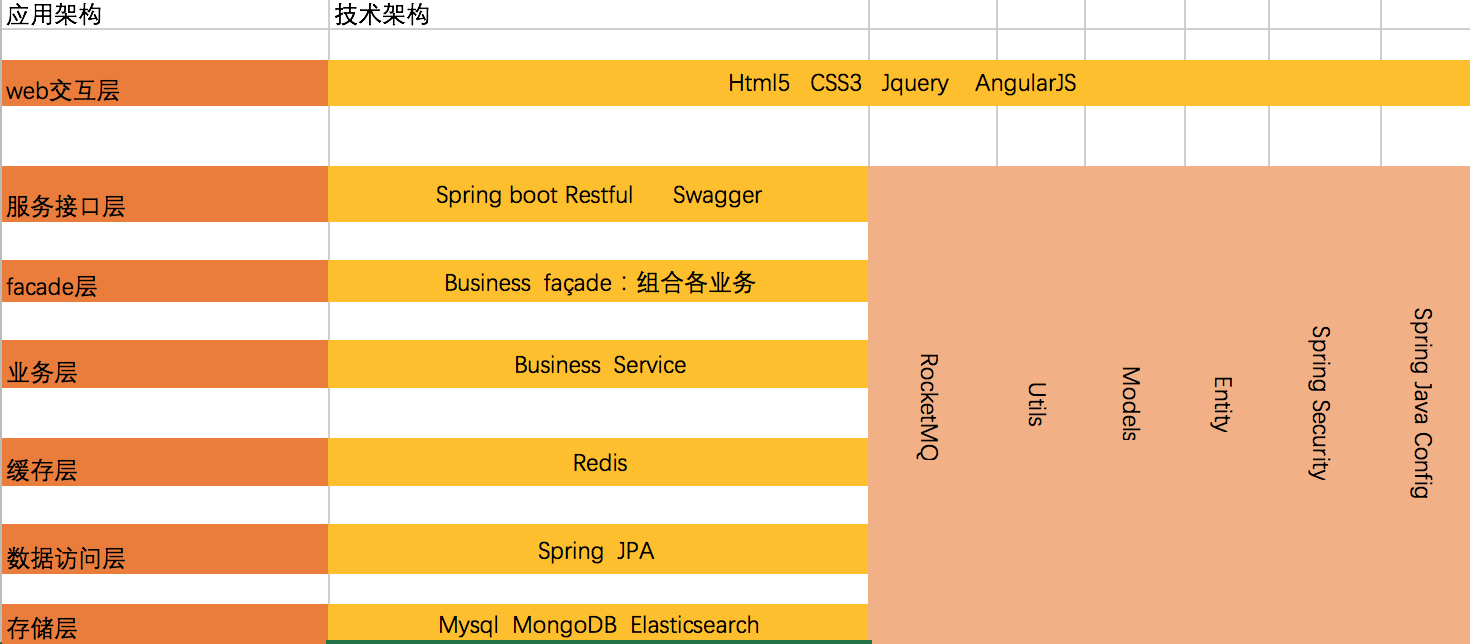
本规范适用于本公司所有JAVA软件项目、产品等的设计、开发以及维护等。所有JAVA软件开发人员在整个软件开发过程中必须遵循此规范。

# 平台架构

## 架构概述

本公司JAVA开发的软件产品多为以Restful方式提供服务的平台结构，涉及多个层面的开发，为便于逻辑分离及项目分工，开发的软件产品应当符合平台模块化的设计模式及高内聚低耦合的组件化开发方式并能提供结构层次清晰的API。

## 架构框架图



## 平台基础模块

* 平台基础模块说明地址：[http://www.cuixiaomei.com/:8080/web/swagger-ui.html](http://www.cuixiaomei.com/:8080/web/)
* 平台API地址：<http://www.cuixiaomei.com/:8080/web/v2/apidocs/>
* 平台演示地址：[http://](http://192.168.1.84:8080/lk-plat-demo/rest/)[www.cuixiaomei.com:21000/know/default.html](http://192.168.1.84:8080/lk-plat-demo/rest/)

### lq-plat-core平台核心模块

核心工具类模块

### lq-plat-model平台模型模块

平台的通用业务模型定义及接口模型定义

### lq-plat-entity 平台实体模块

平台通用的业务实体定义

### lq-plat-base 平台基本模块

包含平台基本的配置，接口服务父类，业务服务父类，通用的数据服务父类，平台常量，异常定义及统一的异常处理

### lq-plat-cache 平台缓存管理模块

平台通用的缓存管理

### lq-plat-captcha平台图片验证码模块

平台图片验证码服务

### lq-plat-mail平台邮件发送模块

按通用的模板发送邮件

### lq-plat-mq平台异步通信模块

提供平台间，平台模块间的异步通信能力支持

### lq-plat-repository平台数据持久化模块

平台通用的持久化模块

### lq-plat-repository-mongodb平台mongodb持久化模块

包含对mongodb通用的操作（新增，修改，删除，查找）

### lq-plat-search平台搜索模块

提供对中英文内容的各种搜索服务

### lq-plat-security平台安全模块

平台通用的安全管理模块

### lq-plat-im 平台即使通讯模块

平台通用的即使通讯模块

# 开发规范

所有代码文件均使用UTF-8编码。

## 基本规范

### 代码组织与风格

采用阿里巴巴JAVA开发手册对应编程规范

插件[点此下载](https://github.com/alibaba/p3c)

## 命名规范

所有的命名都要能完整的表达名称的含义或者使用约定俗成的缩写。未提及的规范尊照java官方规范。

### Package的命名

Package应该都是由一个小写单词组成且按功能模块分包定义。

所有的包定义都以com.lq.业务平台缩写.类型.功能.子功能的组织方式。

如：

com.lq.link.web.service，com.lq.link.web.service.user等。

### Class的命名

Class使用驼峰风格(CamelCase)命名，且第一个字母大写。

### 类变量的命名

类变量使用驼峰风格(CamelCase)命名且第一个字母小写。

### Static Final变量的命名

Static Final变量使下划线风格(UnderScoreCase)应该都大写且需要表达完整的意义。

### 参数的命名

参数的命名使用驼峰风格(CamelCase)命名且第一个字母小写且能表达完整的意义。

### 数组的命名

数组应该总是用下面的办法来命名：byte[] buffer，而不是：byte buffer[];

## 平台模块规范

我们公司的平台分为基础平台和业务平台两部份，其中基础平台按模块化提供很多底层的封装和基本的配置及一些常用的工具类，业务平台则是基于基础平台构建的面向用户使用的平台化产品。

### 切分原则

模块按功能性定义进行切分，每个模块解决一类需求。

### 定义原则

* 模块的分组命名：com.lq.业务平台缩写，如com.lq.plat.base
* 模块的命名格式：lq-业务平台缩写-功能，如lq-plat-base

### 引用原则

当模块需要引用其他模块时，如需要引用多个模块，则按最小化引用原则。去掉已经由引入的模块所引入的模块的定义。

## 功能模块规范

功能模块化是尽可能将一个较大的功能分解为一个个较小的、相对独立但又相关的模块化代码来实现。在功能模块开发中遵循：

* 不要使用全局变量
* 内容明确
* 不要有代码重复
* 单一职责原则
* 使用组合代替继承
* 抽象出通用类

## 接口规范

### 设计原则

借鉴RESTful设计原则，而REST的核心原则是将API拆分为逻辑上的资源。这些资源通过http被操作（GET ,POST,PUT,DELETE）, 即使内部数据模型和资源已经有了很好的对应，API设计的时候不需要把它们一对一的都暴露出来。这里的关键是隐藏内部资源，暴露必需的外部资源。

一旦定义好了要暴露的资源，可以定义资源上允许的操作，以及这些操作和API的对应关系：

* GET /tickets # 获取ticket列表
* GET /tickets/12 # 查看某个具体的ticket
* POST /tickets # 新建一个ticket
* PUT /tickets/12 # 更新ticket 12.
* DELETE /tickets/12 #删除ticekt 12

使用REST的好处在于可以充分利用http的强大实现对资源的CURD功能。而这里只需要一个endpoint：/tickets。

endpoint的单数复数，一个可以遵从的规则是：虽然看起来使用复数来描述某一个资源实例看起来别扭，但是统一所有的endpoint，使用复数使得你的URL更加规整。这让API使用者更加容易理解，对开发者来说也更容易实现。

如何处理关联？关于如何处理资源之间的管理REST原则也有相关的描述：

* GET /tickets/12/messages- Retrieves list of messages for ticket #12
* GET /tickets/12/messages/5- Retrieves message #5 for ticket #12
* POST /tickets/12/messages- Creates a new message in ticket #12
* PUT /tickets/12/messages/5- Updates message #5 for ticket #12
* PATCH /tickets/12/messages/5- Partially updates message #5 for ticket #12
* DELETE /tickets/12/messages/5- Deletes message #5 for ticket #12

其中，如果这种关联和资源独立，那么我们可以在资源的输出表示中保存相应资源的endpoint。然后API的使用者就可以通过点击链接找到相关的资源。如果关联和资源联系紧密。资源的输出表示就应该直接保存相应资源信息。（例如这里如果message资源是独立存在的，那么上面 GET /tickets/12/messages就会返回相应message的链接；相反的如果message不独立存在，他和ticket依附存在，则上面的API调用返回直接返回message信息）

不符合CURD的操作，对这个令人困惑的问题，下面是一些解决方法：

重构行为action。当行为不需要参数的时候，可以把active对应到activated这个资源，（更新使用patch）.

以子资源对待。例如:GitHub上，对一个gists加星操作：PUT /gists/:id/star 并且取消星操作：DELETE /gists/:id/star.

有时候action实在没有难以和某个资源对应上例如search。那就这么办吧。我认为API的使用者对于/search这种url也不会有太大意见的（毕竟他很容易理解）。只要注意在文档中写清楚就可以了。

### 接口定义规范

所有接口都使用UTF-8编码，并使用json数据格式。

#### 接口URL定义

{domain}/api/{version}/{client}/{channel}/{url}

* **domain ：网址**
* **version ：版本号 v1**
* **client ：客户端**
* **channel：渠道名称**
* **url :业务路径**

1. client定义

|  |  |  |
| --- | --- | --- |
| **platform** | type | identification |
| **Android系统** | 手机 | **a1** |
| 平板 | **a2** |
| **IOS系统** | 手机 | **i1** |
| 平板 | **i2** |
| **Web网站** | 前端服务 | **w1** |
| 运维后台 | **w2** |
| 第3方网站 | **w3** |

1. channel定义

|  |  |
| --- | --- |
| channel | identification |
| 微信 | **wechat** |
| 微博 | **weibo** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. url定义

业务平台名称/模块名称/子模块名称

如: link/web/admin/infousers/info

#### 接口参数定义

* 接口接收参数对像名称统一以Para结尾且放在models包或者子包下，如：com.lq.plat.model.user.UserInfoPara
* 接口返回对像如果是定义的值对像，名称统一以Dto结尾且放在model包或者子包下，如：com.lq.plat.model.user.UserInfoDto

### 接口开发规范

* 接口主要为输入输出服务，不应当含有业务逻辑代码
* 所有接口均继承com.lq.plat.controller.PlatformController，并放入com.lq.平台名.controller下或者子包下;
* 在接口类上添加@RestController注解
* 在接口类上添加@Api(value = "接口模块名称", description = "接口模块描述")
* 接口方法定义结构：
  + @ApiOperation注解用于定义此接口的名称及功能说明
  + @RequestMapping注解用于定义些接口的业务url地址及服务操作的类型(get,post,put,delete)
  + 返回类型定义为：WebAsyncTask<ResponseEntity<PlatformResult>>，并使用ApiUtils中提供的静态方法来按需要返回，其中可以自定义返回数据和消息编码及http状态码和http头信息。
  + url中的参数在url中名称定义在{}里，在方法中使用@PathVariable来对应定义此参数，如果是url通过参数列表的方式的请使用@RequestParam注解
  + 获取请求体中提交的json数据使用@RequestBody注解，检验方法使用@Valid注解并声明一个检验结果绑定参数BindingResult result，在方法体内通过if (result.hasErrors()){return ApiUtils.validError(result);}来判断是否检验通过
  + 接口的服务逻辑写在Callable方法体内，并只限于输入输出逻辑的编写，业务逻辑部分是调用相关的service来实现
  + 接口返回调用父类的getWebAsyncTask()方法，默认10秒超时由服务器端自动断开当前请求并释放占用的资源，也可以指定超时时间

### 接口在线测试

按接口规范定义的接口都可以在线进行测试，测试url为当前应用rest服务url：

http://localhost:8080/web/sagger-ui.html

## Service开发规范

* 所有service类均继承com.lq.plat.service.PlatformService，并放入com.lq.plat.repository.mongo包下或者子包下；
* Service服务类是实现业务逻辑的主要地方，一个Service解决一类事情，不应当含有过多的交叉逻辑及service内的method结构需清晰且尽量原子化，不同的业务service类通过引用的方式来调用（@Autowired）；
* 所有Service中需要通用数据库操作逻辑的引入或者通过继承扩展com.lq.plat.repository.mongo.MongoRepository类来处理；

## Repository开发规范

* 所有Repository接口继承com.lq.plat.repository.mongo.PlatformMongoRepository，并放入com.lq.平台名.repository.mongo包下或者子包下；
* 接口声明规范：<http://docs.spring.io/spring-data/data-mongo/docs/1.7.0.RELEASE/reference/html/#mongodb.repositories.queries>

## Model开发规范

* 所有model继承com.lq.plat.models.PlatformModel，并放入com.lq.平台名.models包下或者子包下;
* Model从业务逻辑设计上定义了业务数据结构模型，可用于转化成实体及接口参数和业务类方法参数等；
* Model与Model之间的关系可通过继承与引用（将其他Model当成一个属性）的方式来处理
* 接口json格式参数，较复杂的方法参数及返回对像都必须使用model；
* 接口model参数必须添加注解检验；

## Entity开发规范

* 所有entity继承com.lq.plat.entity.PlatformMongoEntity，并放入com.lq.平台名.entity包下或者子包下；
* Entity为最终持久化到数据库并装载数据库数据使用；
* 所有entity由PlatformMongoEntity来定义主键（ID），维护创建时间（createdDate）和最后修改时间（lastModifiedDate）
* 所有entity类必须加上@Document注解
* 可以使用bid来创建复合值

## 日志记录规范

* 继承了平台提供的父类的直接使用log变量即可
* 继承了平台提供的父类的使用

private final Logger log = LoggerFactory.getLogger(getClass());来定义

* 日志记录中使用”{}”占位符来定义要记录的变量数值，如log.info(“ret:{}”,ret);
* 日志记录中如果使用了debug方法，则必须使用以下方式：

if(log.isDebugEnabled()){

log.debug();

}

* 根据实际情况酌情使用warn,info级别记录，在明确的错误及异常处理中务必使用error级别并定义明确的日志关键词

# 测试规范

## 单元测试

* 所有service方法务必添加单元测试用例。
* 所有测试用例放在src/test/java目录里
* 所有测试用例继承com.lq.plat.test.AbstractPlatformTest

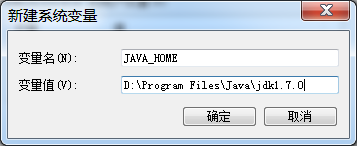
# 开发环境

## 开发工具

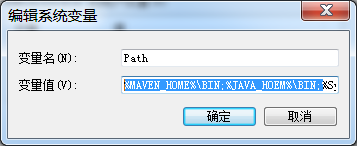
IDEA MAVEN JDK

## 环境配置

### 系统环境







### 开发工具

IDEA

## 代码提交规范

配置忽略以下内容

* .project
* .classpath
* .settings
* site
* target
* log

在提交时必须编写说明并可引用以下关键词来同步更新redmine中的指定编号的状态，如：refs #2

* refs解决某个需求
* IssueID添加功能
* fixes修复bug
* closes代码编写完毕



**注：**请在redmine知识库中详细阅读：Redmine及SVN的使用.ppt