**代码编写规范**

**编号：XZSKC-21**

**编写： 罗娇 日期： 2022/2/4**

**审核： 朱绍攀 日期： 2022/2/12**

**批准： 朱绍攀 日期： 2022/2/12**

**徐州市勘察测绘研究院有限公司**

**文档修订记录**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 版本号 | \*变化  状态 | 简要说明 | 日期 | 变更人 | 批准日期 | 批准人 |
| V1.0 | C | 创建 | 2021/12/2 | 罗娇 | 2021/12/19 | 朱绍攀 |
| V1.1 | M | 添加Idea配置注释模板 | 2021/12/26 | 罗娇 | 2021/12/26 | 朱绍攀 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**目录**

[1. 代码规范 2](#_Toc42246804)

[1.1 编程规约 2](#_Toc42246805)

[1.2 异常日志 5](#_Toc42246806)

[2. 单元测试规范 6](#_Toc42246807)

[3. 代码注释 7](#_Toc42246808)

[3.1 注意事项 7](#_Toc42246811)

[3.2 Idea配置注释模板 8](#_Toc42246812)

# 1.代码规范

## 编程规约

* **命名风格**
* 代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

反例: \_name / \_\_name / $name / name\_ / name$ /

* 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

正例: alibaba / taobao国际通用的名称，可视同英文。

反例: getPingfenByName() [评分]

* 类名使用UpperCamelCase风格

正例:JavaServerlessPlatform

反例:javaserverlessplatform

* 方法名、参数名、成员变量、局部变量都统一使用lowerCamelCase风格，必须遵 从驼峰形式。

正例: localValue / getHttpMessage() / inputUserId

* POJO类中布尔类型变量都不要加is前缀，否则部分框架解析会引起序列化错误。
* 任何自定义编程元素在命名时，使用尽量完整的单词 组合来表达其意。

正例:在 JDK 中，表达原子更新的类为:AtomicReferenceFieldUpdater。

反例:int a 的随意命名方式。

* 对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用

Impl 的后缀与接口区别。

正例:CacheServiceImpl 实现 CacheService 接口。

* 各层命名规约
* 获取单个对象的方法用 get 做前缀。  
  获取多个对象的方法用 list 做前缀，复数形式结尾如:listObjects。   
  获取统计值的方法用 count 做前缀。  
  插入的方法用 save/insert 做前缀。  
  删除的方法用 remove/delete 做前缀。  
  修改的方法用 update 做前缀。
* 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

正例:MAX\_STOCK\_COUNT

反例:MAX\_COUNT

* 抽象类命名使用Abstract或Base开头;异常类命名使用Exception结尾;测试类 命名以它要测试的类的名称开始，以 Test 结尾。
* 接口类中的方法和属性不要加任何修饰符号(public 也不要加)

正例:接口方法签名 void commit();

反例:接口方法定义 public abstract void f();

* **OOP规约**
* Object的equals方法容易抛空指针异常，应使用常量或确定有值的对象来调用 equals。

正例:"test".equals(object);

反例:object.equals("test");

* 所有整型包装类对象之间值的比较，全部使用equals方法比较。
* 定义 DO/DTO/VO 等 POJO 类时，不要设定任何属性默认值。

反例：POJO 类的 getCreate 默认值为 new Date();但是这个属性在数据提取时并没有置入具体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

* 在getter/setter 方法中，不要增加业务逻辑，增加排查问题的难度。
* 循环体内，字符串的连接方式，使用 StringBuilder 的 append 方法进行扩展。

反例:  
String str = "start";  
for (int i = 0; i < 100; i++) {  
str = str + "hello"; }

* 关于基本数据类型与包装数据类型的使用标准如下

【强制】所有的 POJO 类属性必须使用包装数据类型。  
【强制】RPC 方法的返回值和参数必须使用包装数据类型。  
【推荐】所有的局部变量使用基本数据类型。

局部变量不涉及传输问题，比较和计算等都是直接使用值，基本类型最方便也最高效。

* 构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。
* POJO 类必须写 toString 方法。如果继承了另一个 POJO 类，注意在前面加一下 super.toString。

说明:方便排查问题

* **集合处理**
* 在使用 Collection 接口任何实现类的 addAll()方法时，都要对输入的集合参数进行NPE 判断。
* 不要在 foreach 循环里进行元素的 remove/add 操作。remove 元素请使用Iterator 方式
* 集合初始化时，指定集合初始值大小。(无法确认设置为16)

说明:没有设置容量初始大小，随着元素不断增加，容量 被迫扩大，resize 需要重建 hash 表，严重影响性能。

* 利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的 contains 方法进行遍历、对比、去重操作。(去重使用Set)
* 使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。

说明:keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出 key 所对应 的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效率更高。

* **控制语句**
* 当switch括号内的变量类型为String并且此变量为外部参数时，必须先进行null 判断。
* 在高并发场景中,避免使用“等于”判断作为中断或退出的条件

反例:判断剩余奖品数量等于 0 时，终止发放奖品，但因为并发处理错误导致奖品数量瞬间变成了负数，  
这样的话，活动无法终止。

* **其他**
* 后台输送给页面的变量必须加$!{var}——中间的感叹号。

说明:如果 var 等于 null 或者不存在，那么${var}会直接显示在页面上。

* 获取当前毫秒数 System.currentTimeMillis(); 而不是 new Date().getTime();

说明:Date本身调用了System.currentTimeMillis()方法,降低了效率

* 在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。

说明:不要在方法体内定义:Pattern pattern = Pattern.compile(“规则”);

## 异常日志

* **异常处理**
* finally块必须对资源对象、流对象进行关闭，有异常也要做try-catch。
* Java 类库中定义的可以通过预检查方式规避的 RuntimeException 异常不应该通过 catch 的方式来处理，比如:NullPointerException，IndexOutOfBoundsException 等。

说明:无法通过预检查的异常除外，比如，在解析字符串形式的数字时，可能存在数字格式错误，不得不  
通过 catch NumberFormatException 来实现。

正例:if (obj != null) {...}

反例:try { obj.method(); } catch (NullPointerException e) {...}

* 应用中不可直接使用日志系统(Log4j、Logback)中的API，而应依赖使用日志框架 SLF4J 中的 API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
private static final Logger logger = LoggerFactory.getLogger(Test.class);

* 在日志输出时，字符串变量之间的拼接使用占位符的方式。

因为 String 字符串的拼接会使用 StringBuilder 的 append()方式，有一定的性能损耗。使用占位符 仅是替换动作，可以有效提升性能。

正例:logger.debug("Processing trade with id: {} and symbol: {}", id, symbol);

* 避免重复打印日志，浪费磁盘空间，务必在 log4j.xml 中设置 additivity=false。

# 2.单元测试规范

单元测试需要DAO层，service层，controller（有逻辑的）层和工具类需要写单元测试。

开发人员提交代码前，必须保证代码的单元测试能够正确运行。

项目发布时，自动执行所有测试用例，保证都能通过。

注意事项：

* 单元测试需要把middleware中WebSocketConfig上的3个注解注释掉。
* 单元测试应遵守自动化、独立性、可重复执行的要求。
* 单元测试应该是全自动执行的，并且非交互式的。测试用例通常是被定期执行的，执。
* 行过程必须完全自动化才有意义。单元测试中应使用assert验证结果是否正确。
* 为了保证单元测试稳定可靠且便于维护，单元测试用例之间决不能互相调用，也不能依赖执行的先后次序。
* 对于单元测试，要保证测试颗粒度足够小。
* 核心业务、核心应用、核心模块的增量代码确保单元测试通过。新增代码及时补充或修改单元测试。
* 单元测试统一写在src/test中，测试类路径与项目类路径相同，命名规则为在待测类名后加Test，如HomepageService与HomepageServiceTest。
* 对于数据库相关的查询，更新，删除等操作，不能假设数据库里的数据是存在的，或者直接操作数据库把数据插入进去，请使用程序插入或者导入数据的方式来准备数据。
* 和数据库相关的单元测试，可以设定自动回滚机制，不给数据库造成脏数据。或者对单元测试产生的数据有明确的前后缀标识。

单元测试作为一种质量保障手段，不建议项目发布后补充单元测试用例，建议在项目提测前完成单元测试。

现阶段，我们需要在新增模块和修改代码时需要加上对应的单元测试。

不忙的时候，需要把系统中核心代码部分加上单元测试。

# 3.代码注释



## 注意事项

* 类、类属性、类方法的注释必须使用 Javadoc 规范，使用/*\*内容*/格式，不得使用

// xxx 方式。

* 所有的类都必须添加创建者和创建日期。
* 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释

使用/ *\*\* \**/注释，注意与代码对齐。

* 所有的枚举类型字段必须要有注释，说明每个数据项的用途。

格式/\*\* \*/

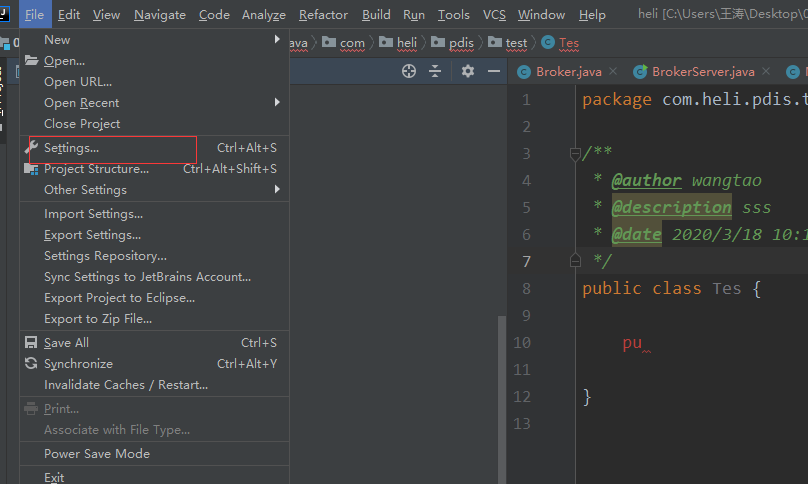
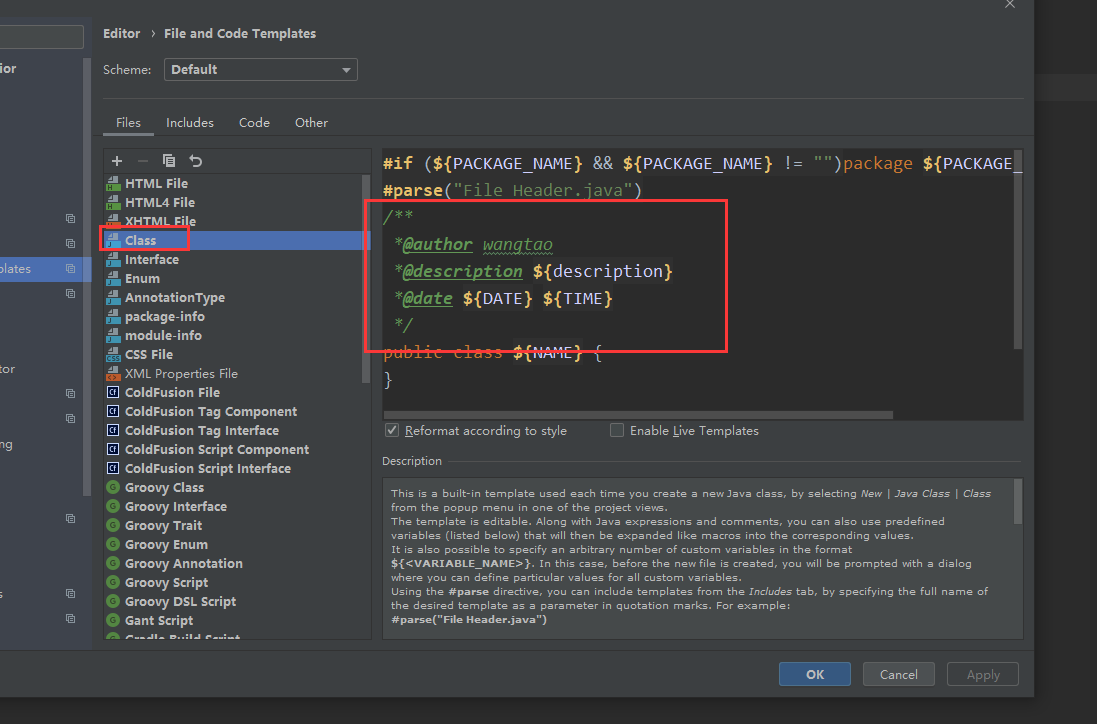
* 代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、核心逻辑  
  等的修改。
* 对于注释的要求：第一、能够准确反应设计思想和代码逻辑 ； 第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路 ； 注释也是给继任者看的，使其能够快速接替自己的工作。
* 特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。

1. TODO

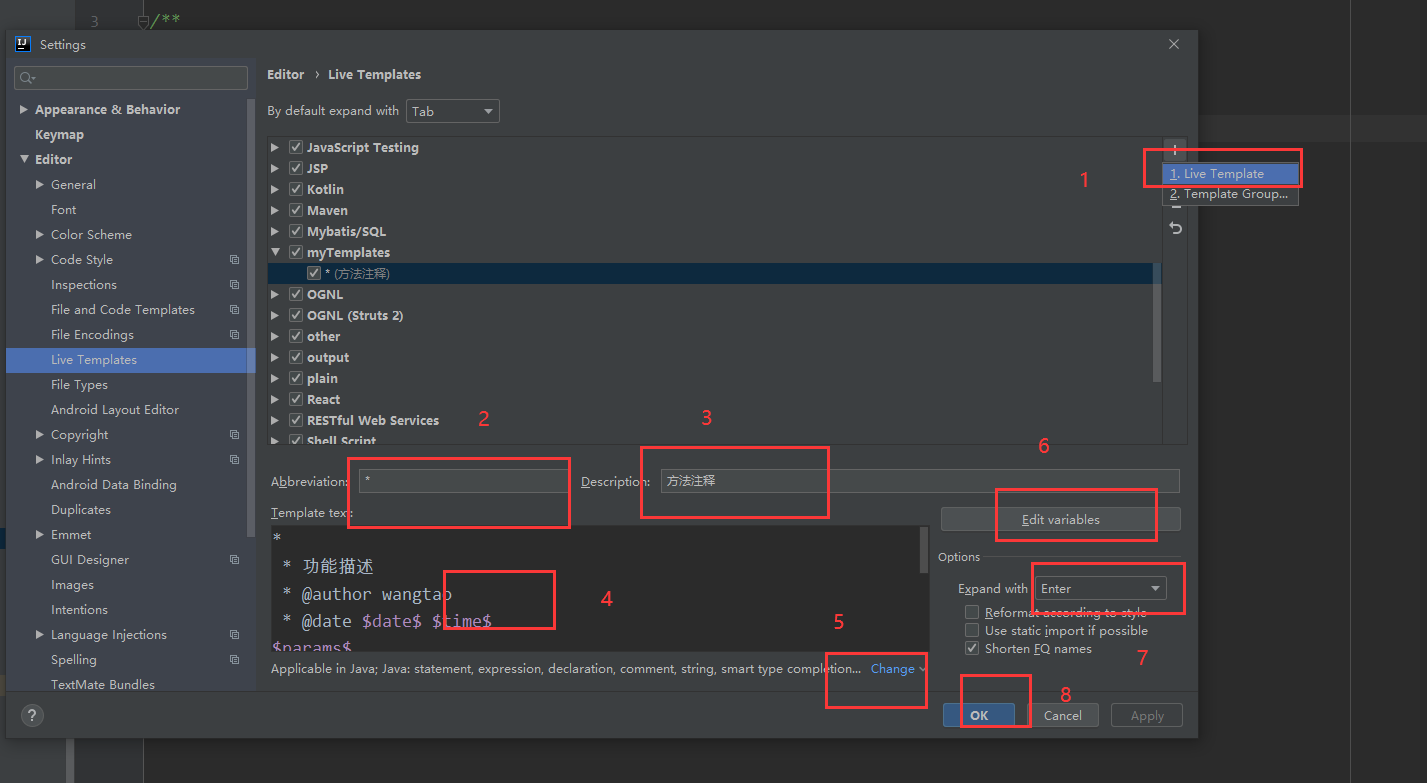
2）FIXME

## Idea配置注释模板

* Idea配置类注释



* Idea配置方法注释的模板



* 新建一个模板
* 修改模板的名字
* 修改模板描述
* 编辑模板

\*

\* 功能描述

\* @author wangtao

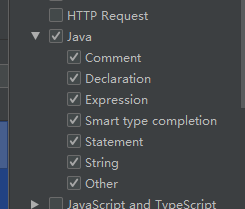
\* @date $date$ $time$

$params$

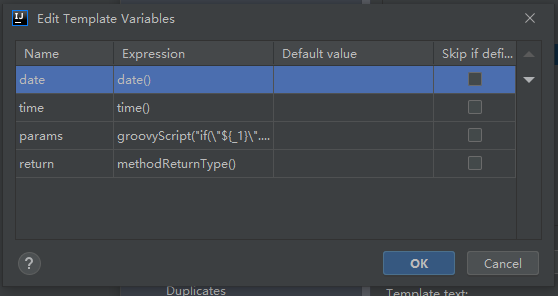
\* @return $return$

\*/

* 修改上图中底部红色字体 No applicable contexts yet. Define  ，点击Define 打开后，设置如下：



* 点击 Edit variables，弹出窗口 如下图所示：



其中params中的值为

|  |
| --- |
| groovyScript("if(\"${\_1}\".length() == 2) {return '';} else {def result=''; def params=\"${\_1}\".replaceAll('[\\\\[|\\\\]|\\\\s]', '').split(',').toList();for(i = 0; i < params.size(); i++) {if(i<(params.size()-1)){result+=' \* @param ' + params[i] + ' : ' + '\\n'}else{result+=' \* @param ' + params[i] + ' : '}}; return result;}", methodParameters()); |

* Expand with 修改为enter
* 点击应用 和 保存

使用方法：在方法上输入/\*\* 点击回车