代码规范

指导原则

原则 1：变量、函数和对象应该是易读的、容易理解的。
原则 2：函数和对象应该符合单一职责原则，是高内聚的。

注意：遇到任何与这两条原则冲突的规范，都以这两条原则为准。

通用规范

1. 【强制】变量的命名需要表达出变量的用途。
2. 【强制】所有情况都不使用单字母。

   正例：namesIndex
   反例：i / j / k

3. 【强制】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 / youku / Hangzhou/ name 等国际通用的名称，可视同英文。
   反例：DaZhePromotion [打折] / mingzi / ren5 / int 某变量 = 3

4. 【强制】变量命名时，不要加变量类型。

   正例：name / task
   反例：intName / stringTask / arrNames / addressStr

5. 【强制】缩写词能在百度里查到才能使用。

   正例：id / AI
   反例：connStr / str

6. 【强制】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 / $name / name_ / name$ / name__

7. 【强制】代码每行最多 130 个字符，超过需要换行。
8. 【强制】函数体不超过 30 行。
9. 【强制】类公开方法不超过 15 个。
10. 【强制】类不超过 800 行 (所有行)。
11. 【强制】函数参数不多于 5 个、函数体不超过 30 行(不含空行)、函数复杂度不超过 7。
12. 【强制】代码中添加 TODO 时,需按照 TODO(姓名):内容的格式书写。

    正例：TODO(张三):要 TODO 的内容。

13. 【强制】数据库时间存取格式统一为“yyyy-MM-dd hh:mm:ss.fff”。

14. 【强制】路径分隔符统一使用正（左）斜杠’/’。

C#代码规范

通用规范

1. 【强制】函数和类型采用大驼峰命名。

   正例：GetTaskId / GetTaskNumber
   反例：gettasks / TaskNumber

2. 【强制】除了对象字段，最好不要出现非数字和字母的字符。

   反例：_name / __name / $name / name_ / name$ / name__

表达式规范

1. 【强制】语句体即使只有一句也不省略{}。

   正例：if (flag) { flag=false; }
   反例：if (flag) flag=false;

2. 【强制】if 语句中的条件表达式的逻辑运算不要超过 3 个。

   正例：if(isDeleted){return;}
   if (flag || (value != “123” && s.Equals("123"))) { return ;}
   反例：if (isDeleted || flag || (value != “123” && s.Equals("123"))) { return ; }

函数规范

1. 【强制】函数以”动词”为核心的词组命名。

   正例：GetUserName() / SetUserName() / Login()
   反例：UserBehavior()

2. 【推荐】回调函数可以采用”On+对象+事件”形式命名。

   正例：OnTextBoxChanged() / OnButtonClicked()

3. 【强制】函数的形参和局部变量采用小驼峰命名。

   正例：taskId / taskNumber
   反例：taskid / TaskNumbe

类与对象规范

1. 【强制】类采用以”名词”为核心的词组命名。

   正例：BankAccount / BankAccountLoader
   反例：SupportInitialize / BankAccountLoad

2. 【强制】类的属性采用以”名词”为核心的词组命名。

   正例：NameAppender
   反例：NameAppend

3. 【强制】类的实例字段以”m_”开头，静态字段以”s_”开头，后面的部分使用小驼峰的形式命名，如果是控件类型，需要在最后加上控件类型的全名。

   正例：string m_userName / static string s_userAge / TextBox m_userAddress
   反例：string UserName / static string UserAge / TextBox m_userAddressTextBox

4. 【强制】类常量或者只读字段不加前缀，且使用大驼峰命名。

   正例：const string UserName / const TextBox UserAddressTextBox
   反例：const string m_userName / const TextBox s_userAddressTextBox

5. 【强制】类的成员函数(包含普通函数、索引、属性)需参照函数的规范。

6. 【强制】类不允许有非常量的公开字段，如果确实需要，则需用属性来代替。

   正例：public string UserName{get;set;}
   反例：public string m_userName

7. 【强制】接口要以”I”开头，以”形容词”或者”名词”命名，且接口的成员不超过 5 个。

   正例：IBankAccount / ISupportInitialize
   反例：BankAccount / SupportInitialize / SetName

Java 代码规范

通用规范

1. 【强制】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

2. 【强制】包名统一使用小写，点分隔符之间有不超过两个自然语义的单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。包名命名层次机构：com.fooww.{module}.{service}

   正例：应用工具类包名为 com . fooww . bigdata . util 、类名为 MessageUtils

3. 【强制】不得使用 System.out.println()控制台输出，使用 log 代替。因为控制台输出无法控制。

4. 【强制】 IDE 的 text file encoding 设置为 UTF -8 ; IDE 中文件的换行符使用 Unix 格式，不要使用 Windows 格式。

5. 【强制】在 long 或者 Long 赋值时，数值后使用大写的 L ，不能是小写的 l ，小写容易跟数字 1 混淆，造成误解。

6. 【建议】项目 groupId 使用 com.fooww.research。artifactId 使用 project-module 形式

表达式规范

1. 【强制】在一个 switch 块内，必须包含一个 default 放在最后。

2. 【强制】在 if / else / for / while / do 语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式。

   反例：if (condition) statements；

3. 【强制】if 判断中，条件不应超过 3 条逻辑。若超过 3 条逻辑判断，请拆分。

4. 【强制】大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行，如果是非空代码块则：

   1 ）左大括号前不换行。

2 ）左大括号后换行。
3 ）右大括号前换行。
4 ）右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。

正例：public void getName(){
        log.info(“frank”);
    }
反例：public void getName()
    {
        log.info(“frank”);
    }

5. 【强制】 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 ");

6. 【强制】判断表达式中，null 应该放在最前面。一方面避免漏写=，另一方面防止空指针异常。

正例：null==object&&" test " .equals(object);
反例：object==null ; object.equals(" test ")

7. 【强制】所有的相同类型的包装类对象（Integer、Float 等）之间值的比较，全部使用 equals 方法比较。

函数规范

1. 【强制】避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。

2. 【强制】相同参数类型，相同业务含义，才可以使用 Java 的可变参数，避免使用 Object。说明：可变参数必须放置在参数列表的最后。（ 提倡同学们尽量不用可变参数编程 ）

正例：public List<User> listUsers(String type, Long... ids) {...}
反例：public List<User> listUsers(Object...objs,String type){...}

3. 【强制】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

反例：GetUserName() / TmpValue

4. 【强制】方法的入参和返回值不得使用基本数据类型，全部使用 Integer、Double 等类代替。

   正例：Integer getUserAge(Byte gender)
   反例：int getUserAge(byte gender)

5. 【强制】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。

6. 【推荐】回调方法可以采用”On+对象+事件”形式命名。

   正例：OnTextBoxChanged() / OnButtonClicked()

类与对象规范

1. 【强制】类名使用 UpperCamelCase 风格，但以下情形例外：DO / BO / DTO / VO / AO / PO / UID 等。

   正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion
   反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

2. 【强制】实体类的每个属性，必须使用 // 加以注释。若实体与数据库严格对应，且数据库有注释，实体类属性的注释可以省略。建议使用 2 个 tab 进行分隔。

3. 【强制】接口和实现类命名需保持一致，实现类后缀加 impl。

   正例：ClickServiceImpl 实现 ClickService 接口

注释规范

1. 【强制】类、类方法、接口的注释必须使用 Javadoc 规范，使用/**内容*/格式，不得使用// xxx 方式。说明：在 IDE 编辑窗口中， Javadoc 方式会提示相关注释，生成 Javadoc 可以正确输出相应注释；在 IDE 中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高阅读效率。

2. 【强制】接口方法修改时，尤其入参与返回值发生修改时，需要对注释进行相应的修改。

Python 代码规范

通用规范

1. 【强制】包名、模块名、函数名、局部变量名全部使用小写，单词间用下划线连接。

   正例：
   ```
   nltkproject
       house_rent
           test_test.py
   ```
   反例：sqlServerHouseRent.py

2. 【强制】常量通常定义在模块级，通过下划线分隔单词，全部大写。

   正例：MAX_OVERFLOW / TOTAL
   反例：MaxOverflow

函数规范

1. 【强制】使用"self"名作为实例方法的第一个参数。使用"cls"名作为类方法的第一个参数。

   正例：
   ```
   class Shape:
       def set_shape(self):
           pass

       @classmethod
       def display_shape(cls):
           pass
   ```

类与对象规范

1. 【强制】类名使用大驼峰命名。

   正例：class HouseDataClean:
   反例：class housedataclean:

排版规范

1. 【强制】每一级缩进使用 4 个空格。续行应该与其包裹元素对齐，要么使用圆括号、方括号和花括号内的隐式行连接来垂直对齐，要么使用挂行缩进对齐。当使用挂行缩进时，应该考虑到第一行不应该有参数，以及使用缩进以区分自己是续行。

正例：

```
def long_function_name(
        var_one, two, three,
        four):
    print(var_one)

foo = def long_function_name(
      var_one, two, three,
      four)

foo = def long_function_name(var_one, two, three,
                             four)
```

反例：

```
    foo = def long_function_name(var_one, two, three,
          four)

    def long_function_name(
          var_one, two, three,
          four):
          print(var_one)
```

2. 【强制】顶层函数和类的定义，前后用两个空行隔开。类里的方法定义用一个空行隔开。

正例：

```
class Shape:
    def set_shape(self):
        pass
    @classmethod
    def display_shape(cls):
        pass


def get_image():
    pass
```

3. 【强制】一行导入一个, 同一个包可以导入多个。

正例：import os
import sys
from subprocess import Popen, PIPE

反例：import sys, os

4. 【强制】导入应该按照以下顺序：标准包导入，相关第三方库导入，本地应用/库特定导入，在每一组导入之间加入空行。

正例：

```
Import time
Import os

Import numpy

Import my_project
```

MYSQL 规范

建表规范

1. 表达是与否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint（1 表示是，0 表示否）。

   正例：表示逻辑删除的字段名 is_deleted，1 表示删除，0 表示未删除。

2. 表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只出现数字，数字必须跟单词放在一起，使用下划线分隔多个单词。

   正例：run_result / testcase_result / level3_name
   反例：RunResult / testcaseResult / level_3_name

3. 表名和字段名不使用复数名词。

   正例：task / task_todo
   反例：tasks / tasks_todos

4. 表必备三字段：

   1) Id：主键，类型为无符号 int，自动递增

   2) create_time：类型为 datetime，值由业务层生成

   3) modified_time：类型为 datetime，值由业务层生成

5. 临时库、表名必须以 tmp 为前缀，并以日期为后缀。

6. 备份库、表必须以 bak 为前缀，并以日期为后缀。

7. 表名要与系统当中的某个模块保持一致（业务）。

   正例：Task 模块的 task、task_todo、task_keyresult

8. 数据库名与项目名一致。

   正例：解决方案 ResearchHome 数据库名 research_home

9. 创建表的时候添加一些业务相关的备注。

   正例：所属模块：任务模块
   作用：存储任务的基本信息
   备注：xxxxx

10. 字段添加备注（如枚举值说明）

正例：status 的值代表的含义如下： 1=未开始 2=执行中 3=测试

11. 禁用保留字，如 desc、range、match、delayed 等，详情参考 MySQL 官方保留字。

12. 表字段有实际意义的，应该设为 NOT NULL, 定义不为空的字段时必须定义默认值。

正例：int/bigint/tinyint => 0
datetime => 1900-01-01
varchar => ""

13. 小数类型总是定义为 decimal，禁止使用 float 和 double，防止丢失精度(长度 20, 4)

14. 不在数据库中存储图片、文件等大数据，只需在数据库存储文件路径。

15. 不得使用外键与级联，一切外键概念必须在应用层解决。

16. 禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

索引规范

1. 业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。

2. 所有新增的主键索引、唯一索引和普通索引分别命名为 pk_字段名、uk_字段名和 idx_字段名。

SQL 语句规范

1. 总是使用 COUNT(*)来统计行数；COUNT(distinct col)计算该列除 NULL 之外的不重复行数，注意 COUNT(dustinct col1，col2)如果其中一列全为 NULL，即使另一列有不同的值，也返回 0；当某一列的值全是 NULL 时，COUNT(col)的返回结果为 0，但 SUM(col)的返回结果为 NULL，因此使用 SUM()时需注意 Null Pointer Exception 问题。

正例：可以使用如下方式避免 SUM()的 Null Pointer Exception 问题：
SELECT IFNULL(SUM(g),0) AS totalg FROM table

2. 在表查询中，一律不要使用 * 作为查询的字段列表，需要哪些字段必须明确写明。

正例：SELECT colA,colB,colC FROM table_name。

反例：SELECT * FROM table_name。

3. SELECT 语句中包含表达式的需要对该表达式起别名。

   正例：SELECT SUM(price) AS total_price FROM table_name。

4. 总是使用 ISNULL()来判断是否为 NULL 值。

   NULL<>NULL 的返回结果是 NULL，而不是 false。
   NULL=NULL 的返回结果是 NULL，而不是 true。
   NULL<>1 的返回结果是 NULL，而不是 true。

5. 在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。

6. 在查询编辑器里做修改或删除记录时，要先 SELECT，避免出现误操作，确认无误才能执行更新语句，语句总是带上 Where 条件，防止出现全局修改或删除。

7. SQL 语句的关键字总是大写。

   正例：SELECT COUNT(*) FROM table_name WHERE id = 1。
   反例：select count(*) from table_name where id = 1。

8. WHERE 语句中涉及时间字段，总是先转换为标准格式“yyyy-MM-dd HH:mm:ss.fff”。

9. SQL 语句中表名与字段名的大小写保持一致。

   数据库表名：users　字段名：id, name, create_time

   正例：SELECT id,name,create_time FROM users WHERE id = 1。
   反例：SELECT ID,Name,Create_Time FROM Users WHERE id = 1。

10. INSERT 语句指定具体字段名称，不要写成 INSERT INTO t1 VALUES(…)。

    正例：INSERT INTO table1 (col1, col2) VALUES (value1, value2)。
    反例：INSERT INTO table1 VALUES (value1, value2)。