命名规则定义文档

> **项目成员：**
>
> 王世泽 - S202175111
>
> 杨悦 - S202175112
>
> 张思远 - S202175103
>
> 肖杏梅 - S202175098
>
> 呼佳嘉 - S202175019
>
> **创建日期：** 2021-10-16
>
> **更新日期：** 2021-10-16
>
> **当前版本：** v1.1

日期	作者	版本	修改记录
2021-10-16	王世泽	v1.0	新建版本
2021-10-16	王世泽	v1.1	完成代码规约与数据库规约

1. 代码规约

1.1 命名风格

1. 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 / $object / name_ / name$ / Object$
```

2. 代码中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。

```
//反例：
    DaZhePromotion [打折] / getPingfenByName()  [评分] / int某变量 = 3
//正例：
    alibaba / taobao / youku / hangzhou //等国际通用的名称，可视同英文。
```

3. 类名使用UpperCamelCase风格，必须遵从驼峰形式，但以下情形例外：领域模型的相关命名，如DO / BO / DTO / VO等。

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

4. 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
```

5. 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
```

6. 中括号是数组类型的一部分，数组定义如下：

```
//正例：
    String[] args;
//反例
    String args[];
```

7. 各层命名规约：

   - Service/DAO层方法命名规约

     1. 获取单个对象的方法用get做前缀。
     2. 获取多个对象的方法用list做前缀。
     3. 获取统计值的方法用count做前缀。
     4. 插入的方法用save（推荐）或insert做前缀。
     5. 删除的方法用remove（推荐）或delete做前缀。
     6. 修改的方法用update做前缀。

   - 领域模型命名规约

     1. 数据对象：xxxDO，xxx即为数据表名。
     2. 数据传输对象：xxxDTO，xxx为业务领域相关的名称。
     3. 展示对象：xxxVO，xxx一般为网页名称。
     4. POJO是DO/DTO/BO/VO的统称，禁止命名成xxxPOJO。

1.2 常量定义

1. 不允许出现任何魔法值（即未经定义的常量）直接出现在代码中。

```
//反例：
String key="Id#taobao_"+tradeId;
cache.put(key,  value);
```

2. long或者Long初始赋值时，必须使用大写的L，不能是小写的l，小写容易跟数字1混淆，造成误解。

   > 说明：Long a = 2l;写的是数字的21，还是Long型的2?

3. 不要使用一个常量类维护所有常量，应该按常量功能进行归类，分开维护。

   如：缓存相关的常量放在类：CacheConsts下；系统配置相关的常量放在类：ConfigConsts下。

1.3 代码格式

1. 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：
    - 左大括号前不换行。
    - 左大括号后换行。
    - 右大括号前换行。
    - 右大括号后还有else等代码则不换行；表示终止右大括号后必须换行。
2. 缩进采用4个空格，禁止使用tab字符。

    说明：如果使用tab缩进，必须设置1个tab为4个空格。IDEA设置tab为4个空格时，请勿勾选Use tab character；而在eclipse中，必须勾选insert spaces for tabs。

```
//正例：
public static void main(String args[]) {
    //缩进4个空格
    String say = "hello";
    //运算符的左右必须有一个空格
    int flag = 0;
    //关键词if与括号之间必须有一个空格，括号内的f与左括号，0与右括号不需要空格
    if (flag == 0) {
        System.out.println(say);
    }
    //左大括号前加空格且不换行；左大括号后换行
    if (flag == 1) {
        System.out.println("world");
        //右大括号前换行，右大括号后有else，不用换行
    } else {
        System.out.println("ok");
        //在右大括号后直接结束，则必须换行
    }
}
```

3. 单行字符数限制不超过 120个，超出需要换行，换行时遵循如下原则：
    - 第二行相对第一行缩进 4个空格，从第三行开始，不再继续缩进，参考示例。
    - 运算符与下文一起换行。
    - 方法调用的点符号与下文一起换行。
    - 在多个参数超长，逗号后进行换行。
    - 在括号前不要换行，见反例。

```
//正例：
StringBuffer sb = new StringBuffer();
//超过120个字符的情况下，换行缩进4个空格，并且方法前的点符号一起换行
sb.append("zi").append("xin")...
    .append("huang")...
    .append("huang")...
    .append("huang");
//反例：
StringBuffer sb = new StringBuffer();
//超过120个字符的情况下，不要在括号前换行
sb.append("zi").append("xin")...append
    ("huang");
//参数很多的方法调用可能超过120个字符，不要在逗号前换行
method(args1, args2, args3, ...
    , argsX);
```

1.4OOP 规约

1. 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. 所有的覆写方法，必须加@Override注解。
3. 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");
//说明：推荐使用java.util.Objects#equals（JDK7引入的工具类）
```

4. 所有的相同类型的包装类对象之间值的比较，全部使用equals方法比较。
5. 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。
6. setter方法中，参数名称与类成员变量名称一致，this.成员名=参数名。在getter/setter方法中，尽量不要增加业务逻辑，增加排查问题的难度。

```
//反例：
public Integer getData(){
    if(true)  {
        return data + 100;
    } else {
        return data - 100;
    }
}
```

7. 循环体内，字符串的联接方式，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";
}
//说明：反编译出的字节码文件显示每次循环都会new出一个StringBuilder对象，然后进行append操作，最后通过toString方法返回String对象，造成内存资源浪费。
```

1.5 注释规约

1. 所有的抽象方法（包括接口中的方法）必须要用Javadoc注释、除了返回值、参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。

```
public interface ICusModelService
{
    /**
     * 查询模型信息管理
     *
     * @param modelId 模型信息管理主键
     * @return 模型信息管理
     */
    public CusModel selectCusModelByModelId(Long modelId);
    ...
}
```

2. 所有的类都必须添加创建者信息。

```
/**
 * 模型信息管理Service业务层处理
 *
 * @author kamiu
 * @date 2021-09-17
 */
@Service
public class CusModelServiceImpl implements ICusModelService {...}
```

2. 数据库规约

2.1 建表规约

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unsignedtinyint（1表示是，0表示否），此规则同样适用于odps建表。
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。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。

```
--正例：
getter_admin，task_config，level3_name
--反例：
GetterAdmin，taskConfig，level_3_name
```

3. 表名不使用复数名词。
4. 小数类型为decimal，禁止使用float和double。

   说明：float和double在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。如果存储的数据范围超过decimal的范围，建议将数据拆成整数和小数分开存储。
5. varchar是可变长字符串，不预先分配存储空间，长度不要超过5000，如果存储长度大于此值，定义字段类型为text，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

2.2 sql语句规约

1. 不要使用count(列名)或count(常量)来替代count()，*count()*就是SQL92定义的标准统计行数的语法，跟数据库无关，跟NULL和非NULL无关。

说明：count(*)会统计值为NULL的行，而count(列名)不会统计此列为NULL值的行。

2. 使用ISNULL()来判断是否为NULL值。注意：NULL与任何值的直接比较都为NULL。

   说明：

   1） NULL<>NULL的返回结果是NULL，而不是false。

   2） NULL=NULL的返回结果是NULL，而不是true。

   3） NULL<>1的返回结果是NULL，而不是true。

3. 不得使用外键与级联，一切外键概念必须在应用层解决。

   说明：（概念解释）学生表中的student_id是主键，那么成绩表中的student_id则为外键。

   如果更新学生表中的student_id，同时触发成绩表中的student_id更新，则为级联更新。

   外键与级联更新适用于单机低并发，不适合分布式、高并发集群；

   级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

4. 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。