Java后台开发规范

前言：一个好的程序员肯定是要能书写可维护的代码，而不是一次性的代码，怎么能让团队当中的其他人，甚至过一段时间之后的你，再看自己某个时期写的代码，依然能看懂？这就涉及到规范你的代码了。

**一、规范代码的好处**

**1、从根本上降低开发成本：**

提高代码整体的可读性、可维护性、可复用性。

**2、保证代码的一致性：**

软件系统中最重要的因素之一就是编码的一致性。如果编码风格一致，也更加易于维护，因为团队内任何人都可以快速理解并修改。

**3、提升团队整体效率：**

开发人员通常需要花费大量的时间来解决代码质量问题，如果都按照规范编写，也有助于团队尽早发现问题，这将提高整个交付过程的效率。

**二、不规范代码的弊端**

**1、增加团队成员间的协作负担：**

由于缺乏规范，导致代码风格不一，极端情况下，某段代码只有某个人能修改。

**2、团队间协作更加困难：**

由于开发人员要适应不同的风格，会导致效率低下。

**3、回顾困难：**

在review期间，可能经常为类似的事情做过多的讨论。

**4、影响降低团队整体效率：**

影响团队的生产力和质量，严重的甚至会影响团队和谐。

**三、为什么很多团队缺乏规范**

1、当开发人员被要求在短时间内完成任务时，通常会回避质量标准。

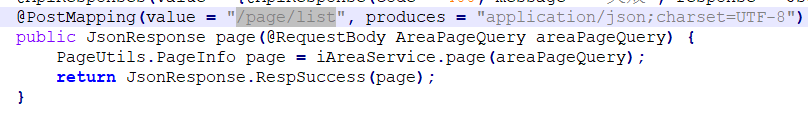
2、长时间养成的开发习惯很难在短时间内去改变。

3、有的时候虽然达成了一致，但在开发中依旧我行我素。

规范：

### 1、controller控制层接口规范

### （1）接口命名尽量简单，例如分页接口



(2) 统一返回ResponseJson

(3) Controller尽量不使用map,json等动态传参，尽量使用javabean对象。

原因：使用动态传参虽然灵活，但代码可读性差，不利于后期维护。

### （4）controller层需要做参数校验

### (5) controller层不打印日志，日志放到aop统一打印

### 尽量使用post请求方式

### controller层接口必须添加注释，说明接口用途

### 项目迭代过程中，有需求变更，需要废弃原接口的，必须在controller层接口添加注释，/\*\*

### \*接口废弃，20xx年x月

### \*/

2、命名规范

**2.1、变量、成员、方法名统一采用驼峰命名(lowerCamelCase),做到见语知其义**

例子：变量——用户数据(userList)、方法——getUserData(int type)等。  
说明：正常变量定义使用驼峰命名,特殊的如DTO\VO\DO等除外。

2.2、类名的定义

（1）普通类名采用大写字母开始;  
（2）抽象类采用Abstract或Base开头。 例子：普通类——class  
UserModel,抽象类——abstract class AbstractUserDefinition等。

## 2.3、常量、类型、接口、子类的定义

（1）常量使用全大写且单词之间用"\_“隔开;

（2）boolean变量不能使用is开头;

（3）接口尽量不要修饰符、子类紧跟接口追加Impl。

例子：常量——SORT\_TYPE,布尔类型——flag,接口——UserService,实现类——UserServiceImpl等。

说明：常量不可组装,需要原子性定义,不能出现"KEY”+SORT\_TYPE这种内部出现。

## 2.4、包名、异常、枚举、方法名称的定义

（1）包名一律采用小写;

（2）异常都采用\_Exception结尾;

（3）枚举都是以Enum结尾;

（4）方法名称——根据方法内容采用如插入insert-\*。

例子：异常——UserException,包名——com.test,枚举——UserEnum,方法名称——insertUser等。

## 2.5、领域模型定义规范：主要是以VO\DTO\DO等结尾

javaBean

数据对象；XxxxEntity，要求：

（1）以 Entity 为结尾（阿里是为 DO 为结尾）

（2）Xxxx 与数据库表名保持一致

（3）类中字段要与数据库字段保持一致，不能缺失或者多余

（4） 类中的每个字段添加注释，并与数据库注释保持一致

传输对象；XxxxDTO，要求：

（1）不可以继承自 Entity

（2） DTO 可以继承、组合其他 DTO，VO，BO 等对象

from；前端请求参数实体类

3、 事务规约

3.1、【强制】数据变更类操作务必增加事务控制

3.2、【强制】事务注解增加 rollbackFor = Exception.class ，避免抛出受检异常，事务不回滚

3.3、【强制】避免大事务（如避免事务中处理过多数据、执行不必要的查询、执行耗时的操作），原则是能不放在事务中处理的操作就不放在事务中。

3.4、【强制】注意导致事务失效的几种场景

4、java程序规范

4.1、重写方法规范：重写方法必须添加注解@Override避免方法名不一致

4.2、参数如果相同、含义也相同,使用可变参数替换逐个定义

4.3、不能使用过时的类或方法

4.4、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

4.5、禁止使用构造方法BigDecimal(double)的方式把double值转化为BigDecimal对象

4.6、涉及金额必须使用BigDecimal计算（double会有精度丢失问题）

**4.7、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。**

说明：并发控制不好,会导致无法中断,如判断==0退出循环,并发会导致负数,中断不了循环。

**4.8、尽量不要在for循环中调用数据库**

通用问题解决方案：

1.针对大数据量导出，建议使用先分页在导出的方式，避免一次性查询大量数据，导致内存溢出。  
 2.针对count统计等查询过慢的问题，建议优先考虑数据库索引和优化sql；若效果不明显，如果数据实时性要求不高，且查询动态条件不多，可考虑数据缓存；如果不满足缓存条件，可使用第三方表的方式，定时统计并将结果存入三方表中，提高查询速度。  
 3.返回各种率的百分比数据，需要乘以100，一般保留两位小数（具体以需求为准）。  
 4.使用group by xxx with rollup统计时可能和count( DISTINCT xxx）结果不一致。<https://blog.csdn.net/u011066470/article/details/89739712>