JAVA后端开发规范（供参考）

注：本文参考阿里巴巴JAVA开发手册

一、命名规约

1. **【强制】**：代码中的命名均不能以**下划线或者美元符号**开始或结束。
2. **【强制】**：命名中严禁使用**拼音与英文混合**的方式，更不允许**使用中文**
3. **【强制】**：类命名使用**`UpperCamelCase`**风格，必须遵循**驼峰模式**，但以下情形可以例外：（领域模型的相关命名） `DO/BO/DTO/VO`等。
4. **【强制】**：方法名、参数名、成员变量、局部变量都统一使用**`lowerCamelCase`**风格,必须遵循驼峰形式。
5. **【强制】**：常量命名全部大写，单词间用下划线隔开，力求语义完整清楚，不要嫌名字长。
6. **【强制】**：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名称开始，以**Test**结尾。
7. **【强制】**：中括号是数组类型的一部分，数组定义如下：`String[] args`;而不应该这样定义：`String args[]`;
8. **【强制】**：`POJO`类中布尔类型的变量都不要加**`is`**，否则可能引起序列化错误。
9. **【强制】**：包名统一使用**小写**，点分隔符之间有且仅有一个自然语义的英语单词。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。
10. **【推荐】**：如果使用到了设计模式，建议在类名中体现出具体模式。如代理模式就命名为：类名+Proxy
11. **【推荐】**：接口类中的方法和属性不要加任何修饰符号（public 也不要加）。尽量不要在接口里定义变量。
12. **【推荐】**：枚举类名最好带上`Enum`后缀，枚举成员变量名需要全部大写，单词间用下划线隔开。
13. **【参考】**：各层命名规约：

    A)`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。

    B)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`。

二、格式规约

1. **【强制】**：大括号使用约定。如果大括号内为空，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；表示终止右大括号必须换行。

2. **【强制】**：任何一个二元运算符左右必须加一个空格。

3. **【强制】**：缩进采用4个空格，禁止使用tab字符。

   说明：如果非要使用tab缩进，必须设置1个tab键为四个空格。IDEA设置tab为4个空格时，请勿勾选`Use tab character`；

4. **【强制】**：单行字符限制不超过120个，超出需要换行。

5. **【强制】**：没有必要增加若干空格来使某一行地字符与上一行地响应字符对齐。

   比如：

   ```
   int a = 3;
   long b = 4L; // 注意long必须使用L，不能使用l,l与1太过相似。
   StringBuffer sb = new StringBuffer();
   ```

6. **【推荐】**：方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义 之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。

三、OOP规约

1. **【强制】**：不要通过一个类的对象引用访问此类的静态变量或者静态方法，否则会增加编译器的解析成本，直接用类名访问即可。

2. **【强制】**：所有的重写方法，必须加@Override注解。

3. **【强制】**：对外暴露的接口签名，原则上不允许修改方法签名，避免对接口调用方产生影响。接口过时时必须加@Deprecated注解，并清晰地说明采用的新接口或者新服务是什么。

4. **【强制】**：不能使用过时的类或者方法。

5. **【强制】**：所有的相同类型的包装对象之间值的比较，全部使用equals方法进行比较。（考虑JAVA对象的缓存策略）

6. **【强制】**：关于基本数据类型和包装数据类型的使用标准如下:

   1. 所有的`POJO`类属性必须使用包装数据类型。
   2. `RPC`方法的返回值和参数必须使用包装数据类型

7. 所有的局部变量**推荐**基本数据类型

8. **【强制】**：定义`DO/DTO/VO`等`POJO`类时，不要设定任何属性默认值。

9. **【强制】**：序列化类新增属性时，不能更改`serialVersionUID`字段，避免反序列化失败；如果完全不兼容升级，避免序列化混乱，那么请修改`serialVersionUID`的值。

10. **【强制】**：构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，需放在init方法中。

11. **【推荐】**：`POJO`类必须写`toString()`方法。

12. **【推荐】**：getter和setter方法中不应该有业务逻辑

13. **【推荐】**：进行字符串的循环拼接使用`Stringbuilder`的append方法进行拓展。

14. 【推荐】: final可提高程序的响应效率，声明成final的情况：
    1. 不需要重新赋值的变量，包括类属性、局部变量。
    2. 对象参数前加final,表示不允许修改引用的指向
15. 类方法确定不允许被重写。
16. 【推荐】： 慎用Object的clone方法来拷贝对象。

    说明：对象的clone方法默认是浅拷贝，若想实现深拷贝需要重写clone方法实现属性对象的拷贝。

四、集合处理

1. 【强制】： 关于`hashCode`和equals的处理，遵循如下规则：
    1. 只要重写equals，就必须重写`hashCode`。
    2. 因为Set存储的是不重复的对象，依据`hashCode`和equals进行判断，所以Set存储的对象必须重写这两个方法。
2. 如果自定义对象做为Map的键，那么必须重写`hashCode`和equals。
3. 【强制】： `ArrayList`的`subList`结果不可以强转成`ArrayList`，否则会抛出`ClassCastException`。

    说明：`subList`返回的是`ArrayList`的内部类`SubList`，并不是`ArrayList`，对于`SubList`子列表的所有操作最终会反映到原列表上。
4. 【强制】： 在`SubList`场景中，须高度注意对原集合元素个数的修改，会导致子列表的遍历、增加、删除均会产生错误。
5. 【强制】： 不要在`foreach`循环中进行remove/add操作。remove元素请使用Iterator方式，如果并发操作，需要对Iterator对象加锁。

```
//错误示例
List<String> a = new ArrayList<String>();
a.add("1");
a.add("2");
for(String temp:a){
    if("1".equals(temp)){
        a.remove(temp);
    }
}

//正确示例
Iterator<String> it = a.iterator();
while(it.hasNext()){
    String temp = it.next();
    if(删除元素的条件){
        it.remove();
    }
}
```

6. 【推荐】： 使用`entrySet`遍历Map类集合`KV`，而不是`keySet`方式进行遍历。因为前者效率更高。
7. 【推荐】： 高度注意Map类集合K/V能不能存储null值得情况，如下表：

集合类	Key	Value	Super	说明
`Hashtable`	not null	not null	Dictionary	线程安全
`ConcurrentHashMap`	not null	not null	`AbstractMap`	分段锁技术
`TreeMap`	not null	允许为null	`AbstractMap`	线程不安全
`HashMap`	允许为null	允许为null	`AbstractMap`	线程不安全

五、注释规约

1. **【强制】**：类、类属性、类方法得注释必须使用`Javadoc`规范，使用`**/内容*/格式`,不得使用`//xxx`方法。

   说明：**任何字段如果为非负数，必须是unsigned**

2. **【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/* */，注意与代码对齐。

3. **【强制】**：所有枚举类型字段必须要有注释，说明每个数据项的用途。

4. **【推荐】**：与其用别人看不懂的英文来注释，不如用中文把注释把问题说清楚。专有名词和关键子保持英文即可。

5. **【推荐】**：代码修改的同时，请务必考虑注释是否需要更改，如果需要就必须立即更改，否则会导致函数的调用错误地使用该方法等。

6. **【推荐】**：写错的代码要么删除，要么注释掉并**同时**说明错误的原因，以便后面的更新与维护，而不能只简单地注释掉。

7. **【推荐】**：注释要能充分表达函数等的功能、如何使用、何时使用、以及使用注意点等，当然，在命名准确清楚的前提下，注释可视情况删减。同时，也要避免不必要的注释。

8. **【推荐】**：待办事宜务必用`TODO`标记，错误代码段（不能正常工作的代码）请务必用`FIXME`标记。

六、`MySQL`规约

建表规约

1. **【强制】**：表达是与否的字段，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)。

2. **【强制】**：表名、字段名必须使用小写字母或数字，禁止数字开头，禁止两个下划线中间只出现数字(比如a_3_b是不符合规范的）。数据库字段名修改的代价很大，字段名称一定要慎重考虑。

3. **【强制】**：表名不适用复数名词

4. **【强制】**：禁用`MySQL`保留字

5. **【强制】**：唯一索引名为`uk_字段名`；普通索引名则为`idx_字段名`

6. **【强制】**：小数类型为decimal，禁止使用float和double。

7. **【强制】**：存储定长字符串的字段使用char定长字符串类型。

8. **【强制】**：表必备三个字段：id,`gmt_create`（创建时间）,`gmt_modified`（最后一次修改的时间）。

9. **【推荐】**：合适的字段存储长度，不但节约数据库空间，节约索引存储，更重要的是提升检锁速度。

   说明：比如人的年龄用`unsigned tinyint`(表示范围为0-255，人的寿命不会超过255岁);

建索引规约

10. **【强制】**：业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。

    说明：不要以为唯一索引降低了insert的速度，这个速度损耗是可以忽略的，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验和控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

11. **【强制】**：超过三个表禁止join。多表关联查询时，须保证被关联的字段需要有索引。
12. **【强制】**：页面搜索严禁左模糊或者全模糊，如果特别有需要请使用搜索引擎来解决。
13. **【推荐】**：使用order_by时请务必注意利用索引的有序性

`ORM` 规约

14. **【强制】**：在表查询时，一律不要使用 * 作为查询的字段列表，需要哪些字段必须写明确。

    说明：1）增加查询分析器解析成本 2）增减字段容易与 `resultMap` 不一致

15. **【强制】**：`POJO` 类的boolean属性不能加is，而数据库字段必须加id_，要求在 `resultMap` 中进行字段与属性之间的映射。
16. **【强制】**：不允许直接拿 `HashMap` 与 `Hashtable` 作为查询结果集输出的。这样非常不明了！！
17. **【强制】**：@Transactional 事务不能乱用，事务会影响数据库的 `QPS` （每秒查询率）。另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等！

    注意：请自行了解 `QPS` 与 `TPS` 及其区别

七、异常日志

异常规约

1. **【强制】**：异常不要用来做流程控制，条件控制，因为异常处理效率比条件分支低许多！
2. **【强制】**：不要对一大段代码使用try-catch，try-catch精确使用可以不仅可以让流程更加清晰，而且利于维护，否则出问题了还要在一大段代码中找问题。
3. **【强制】**：有try块放到了事务代码中，catch异常后，如果需要回滚事务，一定要注意**回滚事务。**
4. **【强制】**：finally块必须对资源对象、流对象进行关闭（否则可能出现意想不到的问题），有异常也要try-catch
5. **【推荐】**：防止 `NPE` ，是程序员的自我修养，注意 `NPE` 产生的场景：

   1）返回类型为包装数据类型，有可能是null，返回int值时注意判空。

   反例：public int f(){ return Integer 对象}；如果为null，自动解箱抛NPE。

   2）数据库的查询结果可能为null。

   3）集合里的元素即使 `isNotEmpty` ，取出的数据元素也可能为null。

   说明：比如 `treeMap` 和 `hashMap` 的value允许为null

   4）远程调用返回对象，一律要求进行 `NPE` 判断。

   5）对于Session中获取的数据，建议进行 `NPE` 检查，避免空指针

日志规约

6. **【强制】**对 trace/debug/info级别的日志输出，必须使用条件输出形式或者使用占位符的方式，不应该直接使用字符串拼接

   说明：`logger.debug("Processing trade with id: " + id + " symbol: " + symbol)`；如果日志级别是 warn，上述日志不会打印，但是会执行字符串拼接操作，如果 symbol 是对象， 会执行 `toString()` 方法，浪费了系统资源，执行了上述操作，最终日志却没有打印。

正确写法：

```
if (logger.isDebugEnabled()) {
logger.debug("Processing trade with id: " + id + " symbol: " + symbol);
}

// 或者
logger.debug("Processing trade with id: {} symbol : {} ", id, symbol);
```

7. **【强制】**：谨慎使用记录日志。比如生产环境禁止输出debug日志；注意日志输出的量，避免占用大量磁盘资源。