

命名规约

1. **关键字** 禁止使用中文；
2. 为了降低系统代码的耦合性，对数据库访问层和业务逻辑层进行严格区分：
 - 数据库访问层使用 `XxxDAO/XxxDaoImpl`（`Xxx` 表示数据库表名）命名
 - 业务逻辑接口层使用 `XxxService/XxxServiceImpl`（`Xxx` 表示业务模块）形式命名
 更多理解，请参考：《DAO和服务层的一些解释》
3. 为了规范不同层之间的数据传递，对 `PO/DTO/VO` 等制定规范要求（基本类型除外）：

- 持久化层 → 业务层：，采用 `PO`（Persistant Object）
- 业务层 → 控制层（controller）：，采用 `DTO`（Data Transfer Object）
- 类似Dubbo的RPC之间：采用 `DTO`（Data Transfer Object）
- 控制层 → 视图层（view）：，采用 `VO`（View Object）

关于上层向下层传递参数（基本类型除外）：

- 视图层 → 控制层（controller）：如果是更新数据，请使用 `VO`（View Object），如果是查询条件，请使用 `XxxParams.java`
- 控制层 → 业务逻辑层：如果是更新数据，请使用 `DTO`（Data Transfer Object），如果是查询条件，请使用 `XxxParams.java`
- 类似Dubbo的RPC之间：如果是更新数据，请使用 `DTO`（Data Transfer Object），如果是查询条件，请使用 `XxxParams.java`
- 业务逻辑层 → 持久化层：如果是更新数据，请使用 `PO`（Persistant Object），如果是查询条件，请使用 `XxxQuary.java`

注意：`PO/DTO/VO` 不要使用驼峰形式，如：正确形式是 `UserDAO`，而不是 `UserDao`；不同层之间如果有数据拷贝（如：`PO` 中的数据拷贝到 `DTO` 中），请使用 `org.springframework.beans.BeanUtils.copyProperties` 方法。为了保持良好的松耦合，请务必遵守规范，不要交叉使用 `PO/DTO/VO`。禁止使用 `POJO`。

4. 拒绝 **不规范的缩写**，不要嫌名字长，力求望文知意；
5. 实体类布尔属性禁止使用 `isXxxx` 形式的命名；数据库布尔型字段，禁止添加 `is_` 前缀；注意：禁止添加 `is_` 前缀一点可能与《阿里巴巴Java开发手册》有出入，但添加 `is_` 前缀后会增加持久层与业务层之间（框架）的复杂性，并且添加后也并没有明显增加数据库的可维护性和开发效率，所以建议不添加 `is_` 前缀；
6. 方法命名：
 - 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

格式规约

1. 单行字符数限制不超过 120 个，超出必须换行；算术表达式换行时，请在运算符前换行，如：

```
Long longName1 = longName2 * (longName3 + longName4 - longName5)
    + 4 * longname6;
```

2. IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式;
3. 单个方法的总行数不宜太多;

OOP规约

1. 外部正在调用或者二方库依赖的接口，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。
2. 推荐使用 `java.util.Objects#equals`
3. 所有整型包装类对象之间值的比较，全部使用 `equals` 方法比较
4. 浮点数之间的等值判断，基本数据类型不能用 `==` 来比较，包装数据类型不能用 `equals` 来判断，正确做法：

```
// 指定一个误差范围，两个浮点数的差值在此范围之内，则认为是相等的。
float a = 1.0f - 0.9f;
float b = 0.9f - 0.8f;
float diff = 1e-6f;
if (Math.abs(a - b) < diff) {
    System.out.println("true");
}
```

5. 为了防止精度损失，禁止使用构造方法 `BigDecimal(double)` 的方式把 `double` 值转化为 `BigDecimal` 对象
6. RPC 方法的返回值和参数必须使用包装数据类型；所有的局部变量使用基本数据类型；
7. 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

并发规约

1. 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

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
2. 所有的类都必须添加创建者和创建日期
3. 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
4. 在注释中用 `FIXME` 标记某代码是错误的，而且不能工作，需要及时纠正的情况（标记人，标记时间，[预计处理时间]）
5. 请在接口类的方法上进行详细的 javadoc 注释，实现类可以不用写；

日志规约

1. 统一使用 `slf4j` 的 api；（`log4j`、`logback` 等都是 `slf4j` 的实现）
2. 有数据更新的操作，必须进行日志记录，如数据库插入/更新数据等；
3. 关键的流程参数进行日志记录

请使用自定义注解（`cn.com.blumoon.scm.common.annotation.LoggerMethodParams`）需要通过日志记录参数的方法；（见项目 `scm-center-platform`）

4. 捕获的异常，请使用日志记录异常原因；
5. 在日志输出时，字符串变量之间的拼接使用占位符的方式：

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

6. 对于 `trace/debug/info` 级别的日志输出，必须进行日志级别的开关判断。

安全规约

1. 禁止 sql 拼接查询；

2. 用户请求传入的任何参数必须做有效性验证;
3. 数据有效性、合法性必须在后端校验;

数据库规约

1. 数据库名、表名、字段名, 都不允许出现任何大写字母;
2. 表必备三字段: `id`, `create_time`, `update_time`;
3. 页面搜索严禁左模糊或者全模糊, 如果需要请走搜索引擎来解决

说明: 索引文件具有 B-Tree 的最左前缀匹配特性, 如果左边的值未确定, 那么无法使用此索引。

4. 利用延迟关联或者子查询优化超多分页场景, 如正例, 先快速定位需要获取的 `id` 段, 然后再关联:

```
SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b.id
```

5. 防止因字段类型不同造成的隐式转换, 导致索引失效。
6. 不要使用 `count(列名)`或 `count(常量)`来替代 `count()`, `count()`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。
7. 注意 `count(distinct col1, col2)` 如果其中一列全为 NULL, 那么即使另一列有不同的值, 也返回为 0。
8. 表示人名的字段最大长度建议设置为63;

Dubbo规约

1. 在 Provider 端尽量多配置 Consumer 端属性; [参考](#)

其他规约

1. 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
2. 不要在视图模板中加入任何复杂的逻辑
3. `catch`尽可能地区分异常类型;
4. 不要在 `finally` 块中使用`return`
5. 在调用 RPC、二方包、或动态生成类的相关方法时, 捕捉异常必须使用 `Throwable`类来进行拦截:

说明: 通过反射机制来调用方法, 如果找不到方法, 抛出 `NoSuchMethodException`。什么情况会抛出 `NoSuchMethodError` 呢? 二方包在类冲突时, 仲裁机制可能导致引入非预期的版本使类的方法签名不匹配, 或者在字节码修改框架 (比如: ASM) 动态创建或修改类时, 修改了相应的方法签名。这些情况, 即使代码编译期是正确的, 但在代码运行期时, 会抛出 `NoSuchMethodError`。

6. 方法的返回值可以为 `null`, 最好对`null`情况进行说明