# Java技术管理规范文档

## 本规范将规则分为三个级别：

a、强制：必须严格遵守的规则，违反这些规则可能会导致严重的程序错误、性能问题或安全漏洞。代码审查时，违反强制性规则的代码必须被驳回。

b、推荐：建议遵循的规则，有助于提升代码质量和可维护性。在不影响核心功能和进度的前提下，应尽量采纳。

c、允许：在特定场景下可以接受的做法，或者是一些风格选择问题，团队内部可以自行决定是否采纳。

## 2. 强制性规范

### 2.1. 命名规范

1. 类名使用大驼峰命名法（UpperCamelCase），如UserService, OrderController。
2. 方法名、参数名、成员变量、局部变量都使用小驼峰命名法（lowerCamelCase），如 getUserById, orderAmount。
3. 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，如 MAX\_STOCK\_COUNT。
4. 包名统一使用小写，点分隔符之间只有一个自然语义的英语单词。包名统一使用单数形式，如com.scu.project.service。
5. 测试类的名称以它要测试的类的名称开始，以Test结尾，如UserServiceTest。
6. 抽象类命名使用Abstract或Base开头，如AbstractController。
7. 异常类命名使用Exception结尾，如UserNotFoundException。
8. 布尔类型的变量，都不要加is前缀，否则部分框架解析会引起序列化错误。反例：isSuccess。

### **2.2. 代码格式**

1. 使用4个空格进行缩进，禁止使用Tab字符。
2. 左大括号 { 不换行，在其之前保留一个空格。
3. if/else/for/while/do语句必须使用大括号 {} 包围，即使只有一行代码。
4. 在二元和三元运算符两侧必须保留一个空格，如a + b，condition ? true : false。
5. 单行字符数限制不超过120个。超出需要换行，换行时遵循：第二行相对第一行缩进8个空格，运算符与下文一起换行。

### 2.3. 注释规范

1. 类、类属性、类方法的注释必须使用Javadoc规范，使用 /\*\* ... \*/ 的形式。
2. 方法内部单行注释，使用 // 注释。
3. 对于所有public的方法，必须有Javadoc注释，说明其功能、参数（@param）、返回值（@return）和可能抛出的异常（@throws）。
4. 代码修改的同时，必须更新相应的注释，确保注释与代码逻辑一致。

### **2.4. 集合处理**

1. ArrayList的subList 结果不可强转成ArrayList，否则会抛出ClassCastException。
2. 使用集合转数组的方法，必须使用集合的toArray(T[] array)，传入类型完全一样的数组，大小为list.size()。
3. 不要在foreach循环里进行元素的remove/add 操作。remove元素请使用 Iterator方式。
4. Map/Set的遍历，应使用entrySet()方式，而不是keySet()。keySet()效率低下，相当于遍历了两次。
5. 创建HashMap时，应根据实际存储数量预估容量大小，设置初始化容量，减少动态扩容次数。

### 2.5. 并发处理

1. 获取单例对象需要保证线程安全，其中的方法也要保证线程安全。
2. 创建线程或线程池时，必须指定有意义的线程名称，方便出错时回溯。
3. 线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。
4. SimpleDateFormat是线程不安全的类，禁止定义为static变量。如果需要，请使用ThreadLocal或者java.time.format.DateTimeFormatter。
5. 高并发时，同步调用应该去考量锁的性能损耗。能用无锁数据结构，就不要用锁；能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。

### 2.6. OOP规约

1. 避免通过一个类的对象引用去访问此类的静态变量或静态方法，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。推荐使用java.util.Objects.equals。
4. 所有整型包装类对象之间值的比较，全部使用equals方法比较。
5. 任何货币金额，均以最小货币单位且整型类型来进行存储（如Long或Integer表示“分”），禁止使用float或double。

### 2.7. 异常处理

1. try-catch语句块中，catch住异常后，必须进行处理，至少要记录日志。严禁catch 块为空。
2. finally块中的代码也可能抛出异常，必须做相应的处理。
3. 不要在finally块中使用return。finally块中的return会覆盖try和catch块中的 return和异常。
4. 事务场景中，抛出异常时，必须确保事务能回滚。

## 3. 推荐性规范

### 3.1. 命名与设计

1. 接口的实现类，建议以Impl结尾，如UserServiceImpl。
2. 领域模型（POJO）中的布尔值，is开头的属性，在getter/setter时，IDE会自动去除is，保持属性名本身即可。
3. 如果模块、接口、类、方法使用了@Deprecated注解，应该清晰地说明应该使用哪个替代品。
4. 单个方法的总行数建议不超过80行。
5. 一个类中的方法不宜过多，若方法过多，应考虑拆分类。

### 3.2. 代码风格

1. 减少魔法值（未经预先定义的常量）的使用，应将其定义为常量。
2. 表达是与否概念的字段，使用boolean类型。POJO中统一使用包装类型Boolean。
3. if-else结构中，优先处理正常或简单的逻辑，将异常或复杂的逻辑放在else部分。
4. 循环体中，字符串的连接方式，使用StringBuilder的append方法，性能优于+。
5. 建议使用try-with-resources来关闭资源，即使有多个资源需要关闭。

### 3.3. 集合与数据处理

1. 集合初始化时，指定集合初始值大小。例如new ArrayList<>(16);
2. 使用isEmpty()判断集合是否为空，而不是size() == 0。isEmpty()的可读性更好，并且在某些实现中性能可能更优。
3. 对于简单的键值对，优先使用EnumMap或EnumSet，它们通常比HashMap更高效。
4. 使用Java 8的Stream API可以使集合处理代码更简洁、更具可读性。
5. POJO类必须覆写toString方法，方便日志输出和问题排查。

### 3.4. 单元测试

1. 保持单元测试的独立性。为了保证单元测试稳定可靠，单元测试用例之间不能有依赖。
2. 核心业务、核心应用、核心模块的增量代码确保单元测试通过。
3. 单元测试代码必须写在 src/test/java 目录下。
4. 单元测试用例要覆盖常用的输入组合、边界条件和异常情况。
5. 对于数据库等外部依赖，建议使用mock框架（如Mockito）进行模拟，保证测试的稳定性与速度。

### 3.5. 安全规约

1. 用户请求的参数，必须进行有效性验证。
2. 用户输入的文本，在展示时需要进行HTML转义，防止XSS攻击。
3. 在进行SQL查询时，务必使用参数化查询（PreparedStatement），严禁拼接SQL字符串，防止SQL注入。
4. 对于上传的文件，需要校验文件类型、大小，并对文件名进行重命名，防止恶意脚本执行。
5. 敏感信息（如密码、身份证号）在日志中输出时，必须进行脱敏处理。

### 3.6. 日志规约

1. 应用中不可直接使用日志系统（Log4j, Logback）的API，而应依赖使用日志框架 SLF4J中的API。
2. 日志文件推荐至少保存15天。
3. INFO级别日志应记录关键操作的上下文信息，DEBUG级别用于开发调试，WARN级别记录潜在问题，ERROR级别记录系统异常。
4. 避免重复打印日志，以免增大磁盘I/O压力。
5. 异常信息应该包含完整的堆栈信息（stack trace）。

## 4. 允许性规范

1. 在不引起歧义的情况下，对于一些非公有的、功能简单的方法，可以不写Javadoc。
2. 在团队内部达成一致的情况下，可以使用一些非标准的命名约定，例如对于DTO（Data Transfer Object），可以使用DTO后缀。
3. 简单且不会抛出检查性异常的try-catch块中，如果确认异常无需任何处理，可以在catch块中写明注释，解释为何为空。例如catch (InterruptedException e) { Thread.currentThread().interrupt(); // Restore interruption status }
4. 对于一些历史遗留代码，如果修改成本过高，可以暂时不完全遵循此规范，但新增代码必须严格遵守。
5. 使用var关键字进行局部变量类型推断，但前提是必须能清晰地从上下文推断出变量类型，不能降低代码可读性。
6. 在性能要求极高的场景下，允许编写一些非常规的优化代码，但必须附带详尽的注释和性能测试报告。
7. 接口的返回值，可以使用void，也可以返回this以支持链式调用，团队可根据场景自行选择。
8. 在一些简单的、一次性的脚本工具类中，可以使用标准输出System.out.println来打印信息，但在正式项目中应使用日志框架。
9. 对于配置类，允许使用public成员变量来简化代码，但仅限于数据容器，不应包含业务逻辑。
10. 在测试代码中，为了方便，可以使用一些简化的命名和写法，可以适当放宽对魔法值的限制。

## 参考资料:

Google Java Style Guide

阿里巴巴Java开发手册（泰山版）

Oracle Code Conventions for the Java Programming Language