代码格式

a、强制 ：

每个缩进层级使用 2 个空格，禁止使用 Tab 字符。 [1](1)

一行代码的最大长度为 100 个字符，对于注释中的文字、Javadoc 以及 URL 等除外。如果一行代码超出最大长度，必须进行折行。折行时需遵循如下规则：运算符或方法调用的点符号与下文一起折行；如果是非赋值运算符，则在该符号前断开；如果是赋值运算符或 foreach 中的分号，则在该符号后断开。 [2](2)

b、推荐 ：

主体内容层级建议不超过 3 层，整体一目了然，避免过多嵌套。 [3](3)

变量声明时尽量使用`var`，提升代码可读性，但不能滥用，如不能使用`var`声明`null`值变量、不允许推断泛型类型变量等。 [4](4)

c、允许 ：允许使用水平对齐来增加代码美观度，但不做强制要求。 [5](5)

命名规范

a、强制 ：

包名必须全部小写，连续的单词只是简单地连接起来，不使用下划线。 [6](6)

类名和接口名使用 UpperCamelCase 风格，测试类的命名以它要测试的类的名称开始，以 Test 结束。 [7](7)

方法名使用 lowerCamelCase 风格，通常是动词或动词短语。 [8](8)

常量名命名模式为 CONSTANT\_CASE，全部字母大写，用下划线分隔单词，每个常量都是一个静态 final 字段。 [9](9)

b、推荐 ：非常量字段名、参数名使用 lowerCamelCase 风格。 [10](10)

注释规范

a、强制 ：

项目中不允许出现任何解释型的 todo 注释，只允许出现以@Todo 开头的 Javadoc 风格格式。 [11](11)

禁止在代码中出现不符合中文语法的注释，如 “//易出错”“//有问题” 等。 [12](12)

b、推荐 ：

类和成员变量要使用 Javadoc 注释，充分描述功能、行为、输入、输出和异常等信息。 [13](13)

方法必须搭配 Javadoc 的@param、@return 等注释，详细说明方法的输入和输出。 [14](14)

编程实践

a、强制 ：

在敏捷新项目中，必须使用最新的稳定版本的 Java 语言进行开发，如 Java 17。 [15](15)

禁止使用已废弃的类、方法或字段。 [16](16)

禁止抛出裸指针异常，如 NullPointerException、IndexOutOfBoundsException 等。 [17](17)

b、推荐 ：

优先使用局部变量，减少类成员变量的使用，以提高代码的可维护性和可测试性。 [18](18)

避免在循环中创建对象，尽量将对象的创建放在循环外，以提高性能。 [19](19)

c、允许 ：

允许使用三元运算符，但在嵌套使用时不得超过 2 层。 [20](20)

允许使用枚举类型，但需遵循特定的格式规范。 [21](21)

异常处理

a、强制 ：

捕获异常时，必须使用具体的异常类型，禁止直接捕获 Exception 或 Throwable。 [22](22)

禁止在 finally 块中返回或抛出任何内容，否则会导致难以捉摸的错误。 [23](23)

b、推荐 ：

使用自定义异常时，建议定义自己的异常基类，以便统一处理业务逻辑异常。 [24](24)

对于可能出现异常的代码，应提前进行防御性检查，避免不必要的异常发生。 [25](25)

并发控制

a、强制 ：

使用线程池时，必须合理配置线程池参数，避免因线程池配置不当导致系统崩溃。 [26](26)

在多线程环境下，禁止使用非线程安全的集合类，如 ArrayList、HashMap 等。 [27](27)

b、推荐 ：

使用锁时，尽量避免使用重量级锁，优先使用轻量级锁或并发工具类，如 ReentrantLock、ConcurrentHashMap 等。 [28](28)

对于共享资源的访问，建议使用线程安全的容器或进行同步控制。 [29](29)

依赖管理

a、强制 ：

禁止在项目中引入重复的依赖，避免版本冲突。 [30](30)

禁止使用未经安全检测的第三方依赖，防止引入安全漏洞。 [31](31)

b、推荐 ：

尽量使用语义化版本管理，便于理解和更新依赖。 [32](32)

定期检查依赖的更新情况，及时升级到更稳定或安全的版本。 [33](33)

测试规范

a、强制 ：

项目中必须编写单元测试，单元测试的覆盖率应达到一定程度，如不低于 80%。 [34](34)

禁止在测试代码中使用与业务代码相同的日志级别，避免混淆日志输出。 [35](35)

b、推荐 ：

编写测试用例时，应尽量做到测试的独立性和可重复性，避免测试用例之间的相互影响。 [36](36)

使用模拟对象（Mock）框架来隔离依赖，提高测试的可靠性和效率。 [37](37)

性能优化

a、强制 ：

禁止在性能敏感的代码中使用低效的算法或数据结构，如频繁使用高复杂度的排序算法等。 [38](38)

禁止在循环中进行不必要的数据库查询或网络请求，应尽量在循环外处理或进行批量操作。 [39](39)

b、推荐 ：

对于大数据量的处理，建议使用分页或流式处理方式，避免一次性加载过多数据导致内存溢出。 [40](40)

部署与运维

a、强制 ：

禁止在生产环境中直接修改代码或配置文件，所有变更必须通过规范的部署流程进行。 [41](41)

禁止将敏感信息（如数据库密码、API 密钥等）硬编码在代码中，应使用配置文件或环境变量进行管理，并确保其安全存储和传输。 [42](42)

b、推荐 ：

在部署应用时，建议进行灰度发布或蓝绿部署，以降低新版本发布带来的风险。 [43](43)

建立完善的监控和告警机制，及时发现和处理应用运行中的异常情况。 [44](44)

安全规范

a、强制 ：

禁止在代码中存储或传输明文密码，必须对密码进行加密处理。 [45](45)

禁止使用不安全的加密算法或哈希函数，如 MD5 等容易被破解的算法。 [46](46)

b、推荐 ：

对用户输入的数据进行严格的验证和过滤，防止 SQL 注入、XSS 攻击等安全漏洞。 [47](47)

使用 HTTPS 协议进行数据传输，确保数据在传输过程中的安全性。 [48](48)

代码审查

a、强制 ：

项目中的代码提交必须经过代码审查，未通过审查的代码禁止合并到主分支。 [49](49)

禁止跳过代码审查流程进行强制推送或合并代码。 [50](50)

b、推荐 ：

代码审查时，应重点关注代码的可读性、可维护性、性能、安全性等方面。 [51](51)

定期进行代码审查培训，提高团队成员的代码审查能力。 [52](52)