# Java技术管理与编码规范

**规范级别定义**：

**[强制]**：必须遵守的规则，不允许任何形式的违反。代码审查（Code Review）时，违反此级别规则的代码将被拒绝合并。

**[推荐]**：强烈建议遵守的最佳实践，但在特定场景下，若有充分理由，可以不遵守。若要违反，需在代码审查中进行说明。

**[允许]**：表示某项实践是被接受的，通常用于澄清某些有争议或有多种选择的场景。

### 1. 命名规范 (Naming Conventions)

1. **[强制]** 包名（Package）：全部小写，使用反向域名格式，且不含下划线。例如：com.google.common。
2. **[强制]** 类名与接口名（Class/Interface）：采用大驼峰命名法（UpperCamelCase）。例如：LoginService, UserDao。
3. **[强制]** 方法名（Method）：采用小驼峰命名法（lowerCamelCase）。例如：getUserById, calculateScore。
4. **[强制]** 常量名（Constant）：全大写，单词间用下划线分隔。例如：MAX\_CONNECTIONS, DEFAULT\_TIMEOUT。
5. **[强制]** 变量名（非常量字段、参数、局部变量）：采用小驼峰命名法（lowerCamelCase）。例如：userName, requestCount。
6. **[强制]** 类型变量（Type Variable）：单个大写字母，或以大写字母T结尾的大驼峰。例如：T, E, K, V, RequestT。
7. **[推荐]** 抽象类名（Abstract Class）：以 Abstract 或 Base 开头。例如：AbstractController。
8. **[推荐]** 异常类名（Exception Class）：以 Exception 结尾。例如：DuplicateUsernameException。
9. **[推荐]** 测试类名（Test Class）：以被测试类的名称开头，以 Test 结尾。例如：UserServiceTest。
10. **[推荐]** 布尔类型的变量名：避免使用 is 或 has 作为前缀，因为这可能与Getter方法冲突。例如，使用 enabled 而非 isEnabled。

### 2. 格式化规范 (Formatting)

1. **[强制]** 缩进：使用4个空格进行缩进，严禁使用Tab字符。
2. **[强制]** 行长度：每行代码长度原则上不超过120个字符。
3. **[强制]** 大括号使用：if, else, for, do, while 语句，即使其代码块只有一行或为空，也必须使用大括号 {}。
4. **[强制]** 大括号位置：左大括号 { 不换行，紧跟在关键字或方法签名后；右大括号 } 单独占一行。
5. **[推荐]** 空格使用：在所有二元运算符（如=, +, \*, &&）两边都添加一个空格。
6. **[推荐]** 垂直空白：方法与方法之间、逻辑代码块之间，使用一个空行进行分隔。
7. **[推荐]** 导包顺序：import 语句应按以下顺序分组，组内按字母排序：静态导入、java、javax、第三方包（如com, org）、本项目包。
8. **[允许]** 小括号分组：在复杂的表达式中，允许使用额外的括号来明确运算优先级，提升可读性。

### 3. 注释规范 (Comments)

1. **[强制]** Javadoc：所有public的类、接口、方法和非私有字段都必须有Javadoc注释。
2. **[强制]** 禁止空catch块：捕获异常后，绝不能留空catch块。如果确认无需处理，必须添加注释说明原因。
3. **[强制]** TODO注释：必须遵循 // TODO(你的用户名): 说明要做的事情 的格式，以便跟踪。
4. **[推荐]** 注释内容：注释应解释“为什么”这么做，而不是“做了什么”。代码本身应清晰地说明“做了什么”。
5. **[推荐]** 非公开方法：对于复杂的private方法，建议也添加简短的注释。
6. **[允许]** 单行注释：在方法内部，允许使用 // 进行实现细节的注释。

### 4. 编程实践 (Programming Practices)

1. **[强制]** 使用@Override注解：当一个方法意图覆盖父类方法时，必须使用@Override注解。
2. **[强制]** 禁止使用原生类型：禁止使用原生泛型（Raw Types），如 List，必须明确指定泛型类型，如List<String>。
3. **[强制]** 字符串比较：必须使用 .equals() 方法比较字符串内容，严禁使用 ==。
4. **[强制]** 资源关闭：必须使用 try-with-resources 语句来保证InputStream、Connection等可关闭资源的正确释放。
5. **[推荐]** Optional的使用：返回可能为null的单个对象时，推荐使用 Optional<T>，以明确表达“可能没有值”的语义。
6. **[推荐]** 接口编程：变量声明、方法参数和返回值应优先使用接口类型，而不是具体的实现类。例如，使用 List<User> 而非 ArrayList<User>。
7. **[推荐]** 不变性：优先设计不可变类（Immutable Class），对非必须修改的字段使用 final 关键字。
8. **[推荐]** 使用java.time包：处理日期和时间，应使用 java.time 包（如 LocalDate, LocalDateTime），而不是过时的 Date 和 Calendar。
9. **[推荐]** 魔法值：禁止在代码中直接使用魔法值（未定义的字面量），应将其定义为命名良好的常量。
10. **[推荐]** 判空：对于集合或数组类型的返回值，如果没有结果，应返回空集合或空数组，而不是null。
11. **[允许]** 使用var：在Java 10+中，当局部变量的类型可以从右侧表达式清晰推断出时，允许使用var来提高代码简洁性。

### 5. 异常处理 (Exception Handling)

1. **[强制]** 禁止捕获Exception或Throwable：除了在顶层通用处理器或测试代码中，严禁捕获通用的Exception、RuntimeException或Throwable，应捕获具体的子类异常。
2. **[推荐]** 异常包装：在转换或重新抛出异常时，推荐将原始异常（cause）包装进新的异常中，以保留完整的堆栈跟踪信息。
3. **[推荐]** 受检与非受检异常：对于可预见的、调用方需要处理的异常情况（如文件不存在），使用受检异常（Checked Exception）。对于编程错误（如空指针），使用非受检异常（Unchecked Exception）。

### 6. 并发处理 (Concurrency)

1. **[强制]** 线程创建：禁止直接 new Thread()，应使用线程池（ExecutorService）来管理线程生命周期，防止资源耗尽。
2. **[强制]** 锁的释放：在使用锁（Lock）时，必须在finally块中调用unlock()方法，以确保锁一定被释放。
3. **[推荐]** 使用java.util.concurrent：优先使用J.U.C包提供的高级并发工具（如Atomic\*类, ConcurrentHashMap, CountDownLatch）代替底层的wait/notify和synchronized。

### 7. 依赖管理 (Dependency Management)

1. **[强制]** 使用构建工具：项目必须使用Maven或Gradle进行依赖管理，禁止手动在项目中添加JAR包。
2. **[推荐]** 明确依赖版本：显式声明所有直接依赖的版本号，避免使用动态版本（如LATEST或版本范围），以保证构建的可重复性。
3. **[推荐]** 最小化依赖范围：为依赖项指定最合适的范围（scope），例如测试相关的库应使用 test 范围