# Java代码走查规范

|  |  |  |  |
| --- | --- | --- | --- |
| 分类 | 重要性 | 检查项 | 备注 |
| 命名 |  |  |  |
|  | 重要 | 命名规则是否与所采用的规范保持一致？ | 成员变量，方法参数等需要使用首字母小写，其余单词首字母大写的命名方式，禁止使用下划线(\_)数字等方式命名 不要出现局部变量，成员变量大写字母开头等问题 |
|  | 一般 | 是否遵循了最小长度最多信息原则？ | 各种命名尽可能短，表意准确，除2代替‘to’，4代替‘for’外，不建议使用数字在命名中 |
|  | 重要 | has/can/is前缀的函数是否返回布尔型？ | 成员变量，方法参数，局部变量等为布尔型时，如果出现has/can/is开头，则将这些词去掉 |
|  | 重要 | 类名是否存在重名问题？ | 自己实现的类尽量不要和别人的类重名，尽管不在同一个包下，特别是子类和父类重名的情况 |
| 注释 |  |  |  |
|  | 重要 | 注释是否较清晰且必要？ | 方法JAVADOC注释中需要说明各参数、返回值及异常说明，参数说明需按照参数名称及用意对应标注 |
|  | 重要 | 复杂的分支流程是否已经被注释？ |  |
|  | 一般 | 距离较远的}是否已经被注释？ |  |
|  | 重要 | 函数是否已经有文档注释？（功能、输入、返回及其他可选） | 文件，类（含接口，枚举等），成员变量，方法前需要有JAVADOC的注释 |
|  | 一般 | 特殊用法是否被注释？ |  |
| 声明、空白、缩进 |  |  |  |
|  | 一般 | 每行是否只声明了一个变量？（特别是那些可能出错的类型） |  |
|  | 重要 | 变量是否已经在定义的同时初始化？ |  |
|  | 重要 | 类属性是否都执行了初始化？ |  |
|  | 一般 | 代码段落是否被合适地以空行分隔？ |  |
|  | 一般 | 是否合理地使用了空格使程序更清晰？ | 基本代码格式中的空格符不可缺少，这些空格出现在?,:,+,-,\*,/,=,==,>,<,>=,<=,!=,及各种括号附近 |
|  | 提示 | 代码行长度是否在要求之内？ | 每行不得超过120个字符 |
|  | 重要 | controller，service, dao 中不要声明有状态的变量。 | 此变量不能被修改。如果要进行修改，必须通过锁进行控制。 |
|  | 一般 | 折行是否恰当？ |  |
|  | 一般 | 集合是否被定义为泛型类型？ | 定义集合时，建议定义其泛型类型，减少类型转换和警告错误 |
| 语句/功能分布/规模 |  |  |  |
|  | 一般 | 包含复合语句的{}是否成对出现并符合规范？ |  |
|  | 重要 | 是否给单个的循环、条件语句也加了{}？ | if,else,else if,while,for,case等代码块必须用{}包围 |
|  | 一般 | 单个变量是否只做单个用途？ |  |
|  | 重要 | 单行是否只有单个功能？（不要使用；进行多行合并） |  |
|  | 重要 | 单个函数是否执行了单个功能并与其命名相符？ |  |
|  | 一般 | 操作符＋＋和— —操作符的应用是否符合规范？ |  |
| 规模 |  |  |  |
|  | 重要 | 单个函数不超过规定行数？ |  |
|  | 重要 | 缩进层数是否不超过规定？ |  |
| 可靠性（总则/变量和语句） |  |  |  |
|  | 重要 | 是否已经消除了所有警告？ | 开发工具的警告 |
|  | 重要 | 常数变量是否声明为final？ |  |
|  | 重要 | 对象使用前是否进行了检查？ |  |
|  | 重要 | 成员变量，局部变量是否在使用前被赋值？ | 对象初始化为null的对象被调用前必须被重新赋值，如果赋值语句在try块中，调用操作必须在try块中 |
|  | 一般 | 局部对象变量使用后是否被复位为NULL？ | 特别是 数组 集合 Map |
|  | 重要 | 对数组的访问是否是安全的？（合法的index取值为[0, MAX\_SIZE-1]）。 |  |
|  | 重要 | 是否确认没有同名变量局部重复定义问题? | 严禁局部变量名称和类或对象成员变量同名 |
|  | 一般 | 程序中是否只使用了简单的表达式？ |  |
|  | 重要 | 是否已经用（）使操作符优先级明确化？ |  |
|  | 重要 | 所有判断是否都使用了（常量==变量 或者 常量.equals(变量)）的形式？ | 常量放在比较符前可以有效降低比较符写成赋值语句 ，减少空指针异常 |
|  | 重要 | 是否每个if-else if-else语句都有最后一个else以确保处理了全集？ |  |
|  | 重要 | 是否每个switch-case语句都有最后一个default以确保处理了全集？ |  |
|  | 一般 | for循环是否都使用了包含下限不包含上限的形式？（k=0; k<MAX） |  |
|  | 重要 | XML标记书写是否完整，字符串的拼写是否正确? |  |
|  | 重要 | 对于流操作代码的异常捕获是否有finally操作以关闭流对象? | 关闭前需要判断 流对象是否为空 |
|  | 提示 | 退出代码段时是否对临时对象做了释放处理? |  |
|  | 重要 | 对浮点数值的相等判断是否是恰当的？ | 严禁使用==直接判断浮点数值 。提供通用方法 |
|  | 重要 | 是否对象比较都使用了equals？ | 对象（包括包装类）比较必须使用equals，而不是使用==或!=操作 |
|  | 重要 | 使用equals进行比较时是否确保比较的两个对象类型一致？ | equals方法比较的对象在对象类型确定的前提下，建议是同一类型的，例如Integer和""使用equals是不提倡的 |
|  | 一般 | 操作Map或Properties结构对象，用于传值时是否将Key定义为常量？ | Session，Request等对象的setAttribute，getAttribute方法的key建议使用常量，不得手工输入字符串 |
|  | 重要 | 是否在类型转换前确保了类型的兼容？ | 除非明确保证对象类型 |
|  | 重要 | 包装类做简单预算前是否保证非空？ 建议都使用包装类。 | 包装类进行操作前，建议进行非空（null != xx）判断，防止发生空指针异常 |
|  | 重要 | 对象属性在使用前是否确保被准确赋值？ | 只读属性（只提供get方法的成员变量）除非特意返回固定值，否则必须提供set方法或在其他方法调用时将其赋值 |
|  | 重要 | 方法调用前是否有非空判断？ | 对参数的非空判断必须出现在方法调用之前，否则说明前面可能导致空指针或者后者判断是没有必要的，非空判断，默认由调用者提供 |
|  | 重要 | 非线程安全的对象是否被正确保证线程安全？ | DateFormat实例的format方法调用不是线程安全，类似的情况不适合使用static定义，建议使用ThreadLocal方式实现，参看UnifiedCodeGenerator |
|  | 一般 | 相同用意的成员变量是否使用了相同的命名？ | 不同实体Entity、VO、BO之间表示同一含义的成员变量，建议使用相同的名称，尽量不要出现，有的地方用username，有的地方用userName这样的情况 |
| 可靠性（函数） |  |  |  |
|  | 重要 | 入口对象是否都被进行了判断不为空？ |  |
|  | 重要 | 入口数据的合法范围是否都被进行了判断？ |  |
|  | 重要 | 是否对有异常抛出的方法都执行了try...catch保护？ |  |
|  | 重要 | 是否函数的所有分支都有返回值？ |  |
|  | 重要 | int的返回值是否合理？（负值为失败，非负值成功） |  |
|  | 一般 | 对于反复进行了int返回值判断是否定义了函数来处理？ |  |
|  | 一般 | 关键代码是否做了捕获异常处理? |  |
|  | 一般 | 字典表定义是否用枚举，或者有一个统一的定义？ |  |
|  | 重要 | 是否对方法返回值对象做了null检查，该返回值定义时是否被初始化? |  |
|  | 重要 | 是否对同步对象的遍历访问做了代码同步? |  |
|  | 重要 | 是否确认在对Map对象使用迭代遍历过程中没有做增减元素操作？ | Map遍历时执行增减元素操作将抛出ConcurrentModificationException，对集合对象遍历时建议都不要进行增减元素操作。 |
|  | 重要 | 线程处理函数循环内部是否有异常捕获处理，防止线程抛出异常而退出? |  |
|  | 重要 | 原子操作代码异常中断，使用的相关外部变量是否恢复先前状态? |  |
|  | 重要 | 函数对错误的处理是恰当的？ |  |
|  | 重要 | 异常捕获后是否进行了日志记录或异常继续抛出？ | 异常捕获后如果无法处理需要继续抛出，如果可以处理，建议将异常日志进行记录 |
|  | 重要 | 是否构造方法中不调用当前对象的构造方法 | 严禁在构造方法中new一个当前对象 |
| 可维护性 |  |  |  |
|  | 重要 | 实现代码中是否消除了直接常数？（用于计数起点的简单常数例外） |  |
|  | 重要 | 是否消除了导致结构模糊的连续赋值？（如a= (b=d+c )） |  |
|  | 重要 | 是否正确使用了日志记录? |  |
|  | 一般 | 是否有冗余判断语句？（如：if (b) return true; else return false;） | “if (b) return true; else return false;”==》“return b;”；禁止使用类似“if/while(表达式 == true)或if/while(表达式 == false)”的判断 |
|  | 重要 | 是否把方法中的重复代码抽象成私有函数? |  |
| 代码警告 |  |  |  |
|  | 一般 | 是否清除了多余导入的包或类？ |  |
|  | 重要 | 是否清除了只定义未使用的局部变量？ | 严禁局部变量被定义或者初始化而未被使用，这种情况需要删除该局部变量 |
|  | 一般 | 是否将魔鬼数字修改为常量使用？ | 不允许直接使用除-2,-1,0,1,2,3,4,5,6,7,8,9,10外的数字，除此外的数字需要定义常量使用 |
|  | 提示 | 常量定义是否为static final格式？ | 常量定义格式为public/protected//private static final Type TYPE，static和final顺序要保持一致 |
|  | 提示 | 实现序列化的对象是否定义了serialVersionUID？ | 建议实现Serializable的类需要增加“private static final long serialVersionUID = 1L;” |
| 可读性 |  |  |  |
|  | 一般 | 是否用if else结构替换了三元运算符？ | 表达式复杂情况下 不要使用(flag?exp1:exp2)语句，该语句需要修改为if else结构 |
|  | 一般 | 代码注释率是否结余30%~60%之间？ | 代码注释率应落在30%~60%之间 |
| 性能 |  |  |  |
|  | 重要 | 日志记录的Log,Logger对象是否定义为常量？ | 用于记录日志的Log,Logger对象在类中定义必须是static final的，建议定义为private的，因为这类对象初始化比较耗时，不利系统运行 |
| 日志 |  |  |  |
|  | 重要 | 打印信息是否都用日志管理？ | 代码中建议不要使用System.out.println打印信息，只有在系统启动或系统即将退出时使用，其余部分全部用日志记录 |
| 圈复杂度 |  |  |  |
|  | 重要 | 单个类行数是否不大于500行？ | 单个类建议行数小于500行，最多不超过1000行 |
|  | 重要 | 方法参数个数是否在7个以内？ | 方法参数个数建议不大于5个，最多不超过7个 |
|  | 重要 | 单个方法函数是否不大于30行？ | 单个方法建议函数不大于30行，做多不超过60行 |
|  | 重要 | 单方法中try/for/while/switch/if最深深度是否不大于5？ | 单方法中try/for/while/switch/if最深深度不允许大于5 |
|  | 重要 | 方法调用最深深度是否不大于15？ | 方法内部+内部调用累计深度不允许大于15 |
| SQL空格 |  |  |  |
|  | 一般 | 连接符or、in、and、以及＝、<=、>=等前后加上一个空格。 |  |
|  | 一般 | 逗号之后必须接一个空格。 |  |
|  | 一般 | 关键字、保留字和左括号之间必须有一个空格。 |  |
| SQL注释 |  |  |  |
|  | 重要 | 对较为复杂的SQL语句加上注释，说明算法、功能。注释风格：注释单独成行、放在语句前面。 |  |
|  | 重要 | 对重要的计算应说明其功能。 | SQL中尽量少涉及业务逻辑 |
|  | 一般 | 可采用单行/多行注释。（-- 或 /\* \*/方式）。 |  |
| SQL优化性能建议 |  |  |  |
|  |  | 1 书写SQL语句优化细则 |  |
|  | 重要 | 1) 尽量避免相同语句由于书写格式的不同，而导致多次语法分析。 |  |
|  | 重要 | 2) 多表连接时，使用表的别名来引用列。 | 建议最多5个连接 |
|  | 重要 | 3) 不要在任何代码中使用 SELECT \*。 |  |
|  | 重要 | 4) where条件中尽量减少使用常量比较，改用参数变量。 |  |
|  | 重要 | 5) 尽量少用嵌套查询。如必须，请用not exist代替not in子句。 |  |
|  | 重要 | 6) 用多表连接代替EXISTS子句。 |  |
|  | 重要 | 7) 使用UNION ALL提高性能 。 |  |
|  | 重要 | 8) in、or子句常会使用工作表，使索引失效；如果不产生大量重复值，可以考虑把子句拆开；拆开的子句中应该包含索引。 |  |
|  |  | 2 排序注意事项 |  |
|  | 重要 | 1) 大量的排序操作影响系统性能，所以尽量减少orderby和group by排序操作。如必须使用排序操作，请遵循如下规则： |  |
|  | 重要 | a. 排序尽量建立在有索引的列上。 |  |
|  | 重要 | b. 如结果集不需唯一，使用union all代替union。 |  |
|  |  | 3 选用索引注意事项 |  |
|  | 重要 | 1) 对于复合索引，SQL语句必须使用主索引列。 |  |
|  | 重要 | 2) 索引中，尽量避免使用NULL。 |  |
|  | 重要 | 3) 对于索引的比较，尽量避免使用NOT=（!=）。 |  |
|  | 重要 | 4) 查询列和排序列与索引列次序保持一致。 |  |
|  |  | 4 其他经验性规则 |  |
|  | 重要 | 1) 任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。 |  |