**代码规范**

1.变量命名的规则

|  |  |
| --- | --- |
| 序号 | 规则 |
| 1 | 变量的命名规则要求用“匈牙利法则”。即开头字母用变量的类型，其余部分用变量的英文意思或其英文意思的缩写,尽量避免用中文的拼音,要求单词的第一个字母应大写。即：变量名=变量类型+变量的英文意思（或缩写） |
| 2 | 全局变量用g\_开头,如一个全局的长型变量定义为g\_lFailCount,即：变量名=g\_+变量类型+变量的英文意思（或缩写） |
| 3 | 静态变量用s\_开头,如一个静态的指针变量定义为s\_plPerv\_Inst,即：     变量名=s\_+变量类型+变量的英文意思（或缩写） |
| 4 | 成员变量用m\_开头,如一个长型成员变量定义为m\_lCount;即：变量名=m\_+变量类型+变量的英文意思（或缩写） |
| 5 | 对枚举类型（enum）中的变量，要求用枚举变量或其缩写做前缀。并且要求用大写。 |
| 6 | 对struct、union、class变量的命名要求定义的类型用大写。并要加上前缀，其内部变量的命名规则与变量命名规则一致。 |
| 7 | 对常量（包括错误的编码）命名，要求常量名用大写，常量名用英文表达其意 思。 如：#define CM\_FILE\_NOT\_FOUND CMMAKEHR(0X20B) 其中 CM 表示类别。 |
| 8 | 对 const的变量要求在变量的命名规则前加入 c\_,即： c\_+变量命名规则；例 如： const char\* c\_szFileName; |

2.函数的命名规范

|  |  |
| --- | --- |
| 序号 | 规则 |
| 1 | 函数的命名应该尽量用英文表达出函数完成的功能。 |
| 2 | 遵循动宾结构的命名法则，函数名中动词在前,并在命名前加入函数的前缀，函数名的长度不得少于 8 个字母。 |

3.函数参数规范

|  |  |
| --- | --- |
| 序号 | 规则 |
| 1 | 参数名称的命名参照变量命名规范。 |
| 2 | 为了提高程序的运行效率，减少参数占用的堆栈，传递大结构的参数，一律采 用指针或引用方式传递。 |

注释规范

|  |  |
| --- | --- |
| 序号 | 规则 |
| 1 | 函数头的注释对于函数，应该从“功能 ”，“参数”，“返回值 ” 、“主要思路 ”、“调用方法 ” 、“日期 ”六个方 面注释 |
| 2 | 变量的注释： 对于变量的注释紧跟在变量的后面说明变量的作用。原则上对于每个变量应该注释，但 对于意义非常明显的变量， |
| 3 | 文件的注释： |
| 4 | 其他注释： 在函数内我们不需要注释每一行语句。但必须在各功能模块的每一主要部分之前添加块 注释，注释每一组语句，在循环、流程的各分支等，尽可能多加以注释。 |

**代码评审清单(Code Checklist)**

1.类清单

|  |  |  |
| --- | --- | --- |
| 被检查类名 | 模块名称 | 产品名称 |
|  |  |  |
|  |  |  |
|  |  |  |

2.检查项

|  |  |  |
| --- | --- | --- |
| 常规 | 是 | 否 |
| 程序是否明确地注释了其实现功能 |  |  |
| 注释格式是否正确 |  |  |
| 程序单元是否承担单一职责 |  |  |
| 复杂程序是否合理地分解为子程序 |  |  |
| 程序中接口定义是否明晰 |  |  |
| 异常处理是否符合规范 |  |  |
| 包名、类名命名是否符合规范 |  |  |
| 方法名命名是否符合规范 |  |  |

|  |  |  |
| --- | --- | --- |
| 变量 /常量 | 是 | 否 |
| 相关系列变量是否统一集中声明，自定义数据类型变量是否进行了必要而充分的描述 |  |  |
| 变量是否按相关规范进行了命名（有意义的命名、数据类型的区分等） |  |  |
| 变量是否按其命名含义承担了单一的使用目的，必要时是否清晰定义了额外变量 |  |  |
| 变量引用是否正确关闭 |  |  |
| 常量是否使用 getter/setter方法访问 |  |  |
| 是否应用良好定义的枚举类型替代了简单的字符标识或布尔变量使用 |  |  |
| UFBoolean 是否使用 UFBoolean 构造函数进行构造 |  |  |
| 处理可变 String 时是否未使用 StringBuffer |  |  |
| 是否优先使用 Java标准库提供的容器（ ArrayList 、HashSet、HashMap、LinkList ）来处 理顺序结构、集合、关联数组合堆栈队列 |  |  |
| 对数组的使用场景是否正确 |  |  |
| 对于 public 尽量谨慎使用 |  |  |
| 不同作用域变量名称尽量不要相同 |  |  |

|  |  |  |
| --- | --- | --- |
| 算法 | 是 | 否 |
| 数据结构是否合理、精简 |  |  |
| 算法是否可以独立测试，是否与数据库和其他算法隔离 |  |  |
| 是否尽量避免了嵌套的运用 |  |  |
| 复杂逻辑是否进行了必要而充分的注释 |  |  |

|  |  |  |
| --- | --- | --- |
| 控制 | 是 | 否 |
| 代码执行路径是否清晰 |  |  |
| Switch 语句是否有缺省分支 |  |  |
| 控制逻辑复杂度是否合理，是否进行了必要而充分的注释 |  |  |
| 每个循环体是否仅执行了单一而明确的功能 |  |  |
| 与常数比较需要将常数放在比较表达式的前面 |  |  |

|  |  |  |
| --- | --- | --- |
| 布局 | 是 | 否 |
| 代码是否良好格式化并能体现其逻辑结构 |  |  |

|  |  |  |
| --- | --- | --- |
| 设计 | 是 | 否 |
| 程序是否可读、可扩展并健壮 |  |  |
| 具体实现细节是否已尽可能的“隐藏” |  |  |
| 是否优先使用接口而不是抽象类或具体类 |  |  |
| 方法参数是否在 5个以内 |  |  |
| 方法慎用 public 修饰符 |  |  |
| 对于不需要子类来重载的类尽量使用 final |  |  |
| 对象作用域是否过大 |  |  |

|  |  |  |
| --- | --- | --- |
| 效率 | 是 | 否 |
| 尽量不要在循环内出现远程调用 |  |  |
| 每个业务动作远程调用次数是否小于 3 次 |  |  |
| 远程调用数据传输是否有不必要的冗余数据 |  |  |

|  |  |  |
| --- | --- | --- |
| SQL | 是 | 否 |
| Sql 语句小写 |  |  |
| 引用字符使用单引号 |  |  |
| 严禁使用 select \* 形式的语句，必须指出具体字段 |  |  |
| 严禁使用 insert into table values （?,?,?），必须指出具体要赋值的字段 |  |  |
| 避免隐含的类型转换（不同数据类型字段相加） |  |  |
| 子查询前后必须加上括号 |  |  |
| 避免在 where 使用’ 1=1’,’1=2’这种表达方式作为部分条件 |  |  |
| 禁止使用视图 |  |  |
| 禁止使用 XX in () or XX in ()(in 中的元素个数不应超过 500) |  |  |
| 禁止使用 or 超过 500 个 |  |  |
| 禁止使用 not in，建议使用 not exist |  |  |
| 禁止在一条 sql 语句中使用 3 层以上的嵌套 |  |  |
| 如果有多表连接时，应该有主从之分，尽量从一个表取数 |  |  |
| Where 子句过滤条件，索引列或过滤记录最多的条件应该放在前面 |  |  |
| 字符串连接必须使用“ ||” |  |  |
| Case when语句中只能出现 =、>=、<=以及 is null 运算符 |  |  |
| 左连接写法必须带” outer”关键字 |  |  |
| Sql 中函数是否在指定范围内 |  |  |
| Join 与 on 必须严格匹配 |  |  |

3.问题清单

|  |  |  |  |
| --- | --- | --- | --- |
| 问题描述 | 预计修改日期 | 修改日期 | 检查人签字 |
|  |  |  |  |