**石家庄铁道大学信息科学与技术学院**

**实验报告**

**2020年----2020年 第二学期**

专 业： 软件工程

课程名称： 软件过程管理

班 级： 信1705-1

姓 名： 李秦 学号：20173522

指导教师： 王辉

# [实验任务一]：COCOMO II 软件的使用

**实验要求：**

1. 通过自己看Manual，熟悉COCOMO II软件的使用；
2. 验证**软件工作量估计\_例子.ppt**中的例子，结果如下所示：

# **cocomo_test_revised**

1. 提交扩展名为est的COCOMO项目文件，并将结果进行截图。

# 

# [实验任务三]：代码规范和评审模板的建立

**实验要求：**

1. 根据自己的编程习惯和缺陷，建立自己的**代码规范**和**代码评审标准**，并且今后务必加以使用；
2. 需要提交**代码规范**和**代码评审标准**两个表格，使用Word即可。

Java代码规范：

|  |  |  |  |
| --- | --- | --- | --- |
| **Java代码规范** | | | |
| **源码文件基础** | **文件名** | 源码文件名由它所包含的顶级class的类名（区分大小写），加上.java扩展名组成 | |
| **文件编码** | 源码文件编码应为UTF-8。 | |
| **空格字符** | 除了换行符外，ASCII水平空白字符（0x20）是源码文件中唯一支持的空格字符。这意味着： | 1，字符串和字符文字中的所有其它空格字符都将被转义。 |
| 2，制表符（Tab键）不用于缩进。 |
| **特殊转义字符串** | 任何需要转义字符串表示的字符（例如\b, \t, \n, \f, \r, \’, \等），采用这种转义字符串的方式表示，而不采用对应字符的八进制数（例如 \012）或Unicode码（例如 \u000a）表示。 | |
|
|
| **非ASCII字符** | 对于其余非ASCII字符，直接使用Unicode字符（例如 ∞），或者使用对应的Unicode码（例如 \u221e）转义，都是允许的。 | |
|
| **源码文件结构** | **Lincense 或 copyright声明信息（如有）** | 如果需要声明lincense或copyright信息，应该在文件开始时声明。 | |
|
|
| **包声明** | 包声明没有长度限制，单行长度限制规范，不适用于包声明。 | |
| **import语句** | 不应使用通配符import，不管是不是静态导入。 | |
| import语句的行，没有行长度的限制。单行长度限制规范，不适用于import语句所在行。 | |
| import的顺序如下：所有静态导入（static import）为一组；所有非静态导入为一组 | |
| 静态导入不能用于静态嵌套类。它们可以正常导入。 | |
| **类声明** | 每个源码文件中只能有一个顶级class。 | |
| 类成员的顺序对代码的易读性有很大影响，但是没有一个统一正确的标准。不同的类可以以不同的方式对其内容进行排序。 | |
|
| 当一个类有多个构造函数或多个同名的方法时，这个函数要写在一些，中间不要有其它代码。 | |
|
| **格式规范** | **大括号** | 大括号用在if,else,for,do,和while等语句。甚至当它的实现为空或者只有一句话时，也要使用。 | |
| 对于非空语句块，大括号遵循Kernighan和Ritchie的风格：  大括号前没有换行  开头大括号后换行  结束大括号前换行  如果右括号结束一个语句块或者函数体、构造函数体或者有命名的类体，则需要换行。 | |
|
|
|
|
|
| 一个空的语句块，可以在大括号开始之后真接接结束大括号，中间不需要空格或换行。但是当一个由几个语句块联合组成的语句块时，则需要换行。 | |
|
| **语句块的缩进：2空格** | 每当一个新的语句块产生，缩进就增加两个空格。当这个语句块结束时，缩进恢复到上一层级的缩进格数。缩进要求对整个语句块中的代码和注释都适用。 | |
|
| **一行最多一句代码** | 每句代码的结束都需要换行。 | |
| **行长度限制：100** | Java代码的单行限制长度为100个字符。 | |
| **长行换行** | 当一个非赋值运算的语句断行时，在运算符号之前断行。 | |
| 当一个赋值运算语句断行时，一般在赋值符号之后断行。但是也可以在之前断行。 | |
| 在调用函数或者构造函数需要断行时，与函数名相连的左括号要在一行。也就是在左括号之后断行。 | |
|
| 逗号断行时，要和逗号隔开的前面的语句断行。也就是在逗号之后断行。 | |
| 在lambda中，一行不会与箭头相邻，除非如果lambda的主体由单个无括号的表达式组成，则可以在箭头之后立即出现换行。 | |
|
| 当断行之后，在第一行之后的行，我们叫做延续行。每一个延续行在第一行的基础上至少缩进四个字符。 | |
| 当原行之后有多个延续行的情况，缩进可以大于4个字符。如果多个延续行之间由同样的语法元素断行，它们可以采用相同的缩进。 | |
|
| **垂直空白** | 单行空行在以下情况使用：  类成员间需要空行隔开;例外：成员变量之间的空白行不是必需的。一般多个成员变量中间的空行，是为了对成员变量做逻辑上的分组。  在函数内部，根据代码逻辑分组的需要，设置空白行作为间隔。  类的第一个成员之前，或者最后一个成员结束之后，用空行间隔。（可选）  本文档中其他部分介绍的需要空行的情况。 | |
|
|
|
|
|
|
|
| **水平空白** | 除了语法、其他规则、词语分隔、注释和javadoc外，水平的ASCII空格只在以下情况出现：  所有保留的关键字与紧接它之后的位于同一行的左括号之间需要用空格隔开。  所有保留的关键字与在它之前的右花括号之间需要空格隔开。  在左花括号之前都需要空格隔开。只有两种例外：  @SomeAnnotation({a, b})  String[][] x = {{"foo"}};  所有的二元运算符和三元运算符的两边，都需要空格隔开。  逗号、冒号、分号和右括号之后，需要空格隔开。  // 双斜线开始一行注释时。双斜线两边都应该用空格隔开。并且可使用多个空格，但是不做强制要求。  变量声明时，变量类型和变量名之间需要用空格隔开。  初始化一个数组时，花括号之间可以用空格隔开，也可以不使用。 | |
|
|
|
|
|
|
|
|
|
|
|
|
| **分组括号** | 非必须的分组括号只有在编写代码者和代码审核者都认为大家不会因为没有它而导致代码理解错误的时候，或者它不会使代码更易理解的时候才能省略。没有理由认为所有阅读代码的人都能记住所有java运算符的优先级。 | |
|
|
| **枚举类型** | 在遵循枚举常量的每个逗号后，换行符是可选的，还允许附加空行（通常只有一个）。 | |
| **变量声明** | 不要采用一个声明，声明多个变量。 | |
| 局部变量不应该习惯性地放在语句块的开始处声明，而应该尽量离它第一次使用的地方最近的地方声明，以减小它们的使用范围。 局部变量应该在声明的时候就进行初始化。如果不能在声明时初始化，也应该尽快完成初始化。 | |
|
|
|
| **数组** | 所有数组的初始化，都可以采用和块代码相同的格式处理。 | |
| 方括号应该是变量类型的一部分，因此不应该和变量名放在一起。 | |
| **switch语句** | switch语句是指在switch花括号中，包含了一组或多组语句块。每组语句块都由一个或多个switch标签打头。 | |
|
| switch花括号之后缩进两个字符。 每个switch标签之后，后面紧接的非标签的新行，按照花括号相同的处理方式缩进两个字符。在标签结束后，恢复到之前的缩进，类似花括号结束。 | |
|
|
| switch语句中，每个标签对应的代码执行完后，都应该通过语句结束，否则应该通过注释说明，代码需要继续向下执行下一个标签的代码。注释说明文字只要能说明代码需要继续往下执行都可以 | |
|
|
| 每个switch语句中，都需要显式声明default标签。即使没有任何代码也需要显示声明。 | |
| **Annotations** | Annotations应用到类、函数或者构造函数时，应紧接javadoc之后。每一行只有一个Annotations。 Annotations所在行不受行长度限制，也不需要增加缩进。 如果Annotations只有一个，并且不带参数。则它可以和类或方法名放在同一行。 Annotations应用到成员变量时，也是紧接javadoc之后。不同的是，多个annotations可以放在同一行。 | |
|
|
|
|
| **注释** | 注释的缩进与它所注释的代码缩进相同。可以采用 /\* / 进行注释，也可以用 // 进行注释。当使用 /\*/ 进行多行注释时，每一行都应该以 \* 开始， 并且 \* 应该上下对齐。 | |
|
|
| **修饰符** | 多个类和成员变量的修饰符，按Java Lauguage Specification中介绍的先后顺序排序。 | |
| **数字文字** | long 值整型常量使用大写L后缀，从来没有小写（避免与数字1混淆）。 | |
| **命名** | **适用于所有命名标识符的通用规范** | 标示符只应该使用ASCII字母、数字和下划线，字母大小写敏感。因此所有的标示符，都应该能匹配正则表达式 \w+ 。 | |
| **包名** | 包名全部用小写字母，通过 . 将各级连在一起。不应该使用下划线。 | |
| **类名** | 类型的命名，采用以大写字母开头的大小写字符间隔的方式 | |
| class命名一般使用名词或名词短语。interface的命名有时也可以使用形容词或形容词短语。annotation没有明确固定的规范。 | |
| 测试类的命名，应该以它所测试的类的名字为开头，并在最后加上Test结尾。 | |
| **方法名** | 方法命名，采用以小写字母开头的大小写字符间隔的方式（lowerCamelCase）。 方法命名一般使用动词或者动词短语。 在JUnit的测试方法中，可以使用下划线，用来区分测试逻辑的名字，经常使用如下的结构：test\_ 。 | |
|
|
| **常量名** | 常量命名，全部使用大写字符，词与词之间用下划线隔开。 | |
| 常 量是一个静态成员变量，但不是所有的静态成员变量都是常量。在选择使用常量命名规则给变量命名时，你需要明确这个变量是否是常量。 | |
| 常量一般使用名词或者名词短语命名。 | |
| **非常量的成员变量名** | 非常量的成员变量命名（包括静态变量和非静态变量），采用lowerCamelCase命名。一般使用名词或名词短语。 | |
| **参数名** | 参数命名采用lowerCamelCase命名。 | |
| 应该避免使用一个字符作为参数的命名方式。 | |
| **类型名** | 类型名有两种命名方式： 单独一个大写字母，有时后面再跟一个数字。 像一般的class命名一样，再在最后接一个大写字母。 | |
|
|
| **局部变量名** | 局部变量采用lowerCamelCase命名。它相对于其他类型的命名，可以采用更简短宽松的方式。 但即使如此，也应该尽量避免采用单个字母进行命名的情况，除了在循环体内使用的临时变量。 即使局部变量是final、不可改变的，它也不能被认为是常量，也不应该采用常量的命名方式去命名。 | |
|
|
| **Camel case的定义** | 有时一些短语被写成Camel case的时候可以有多种写法。例如一些缩写词汇，或者一些组合词：IPv6 或者 iOS 等。  为了统一写法，Google style给出了一种几乎可以确定为一种的写法。  将字符全部转换为ASCII字符，并且去掉 ’ 等符号。  将上一步转换的结果拆分成一个一个的词语。从空格处和从其他剩下的标点符号处划分。  注意：一些已经是Camel case的词语，也应该在这个时候被拆分。但是例如iOS之类的词语，它其实不是一个Camel case的词语，而是人们惯例使用的一个词语，因此不用做拆分。  经过上面两部后，先将所有的字母转换为小写，再把每个词语的第一个字母转换为大写。  最后，将所有词语连在一起，形成一个标示符。 | |
|
|
|
|
|
|
|
| **编程实践** | **@Override 都应该使用** | @Override annotations只要是符合语法的，都应该使用。 | |
| **异常捕获 不应该被忽略** | 一般情况下，catch住的异常不应该被忽略，而是都需要做适当的处理。 | |
| 如果这个catch住的异常确实不需要任何处理，也应该通过注释做出说明。 | |
| **静态成员的访问** | 当一个静态成员被访问时，应该通过class名去访问，而不应该使用这个class的具体实例对象。 | |
|
| **不使用Finalizers 方法** | 重载Object的finalize方法是非常非常罕见的。 | |

Java代码评审标准：

|  |  |  |  |
| --- | --- | --- | --- |
| **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。 |  |