通用程序设计

45、将局部变量的作用域最小化

(1) 最强有力的做法就是在第一次使用变量的地方对其进行声明。

(2) 几乎每个局部变量的声明都应该包含一个初始化表达式.

46、for-each 循环优于传统的 for 循环

(1) for-each 循环和传统的 for 循环相比的优点：

a.简洁性

b.预防 bug 的优势

b.无性能损失。

因此能够用 for-each 的尽量用 for-each，不要用 for 循环。collectios 和 arrays 都能使用 for-each。只要事先了 iterable 接口的类都能使用 for-each。

(2)有些情况下不能使用 for-each:

a.过滤，要删除指定的元素时，不能用 for-each。

b.转换，但需要遍历 list 或者 array 修改部分或者全部数值的时候，需要记录起始位置和结束为止的迭代器或索引变量。

c.并行迭代，当需要并行迭代不同的集合时，需要迭代器变量或者索引时候，不能用 for-each

47、了解和使用类库

使用标准类库的好处

(1)通过标准类库，可以充分利用这些编写标准类库的知识，较少错误；

(2)不必浪费时间自己去实现类库中已有的功能

(3)标准类库的性能会随着新功能的加入的而不断提高

(4)可以使自己的代码融入主流。

48、如果需要精确的答案，请避免使用 float 和 double

对于需要精确答案的计算，不能使用 float 或者 double，BigDecimal 允许完全控制舍入，如果业务要求涉及多种舍入方式，使用 BigDecimal 很方便，如果性能很关键，涉及的数值不大，就可以使用 int 或者 float，如果数值范围没有超过 9 位十进制数字，可以使用 int，如果不超过 18 位数字，使用 long，如果数值可能超过 18 位，就必须用 Bi**gDecima**

49、基本类型优先于装箱基本类型

(1)Java 1.5 增加自动装箱和自动拆箱，对应基本类型 int、double、boolean，装箱基本类型是 Integer、Double、Boolean。这两种类型之间差别。

(2)基本类型和装箱基本类型之间的三个主要区别：

a.基本类型只有值，而装箱基本类型具有与它们的值不同的同一性（两个装箱基本类型可以具有相同的值和不同的同一性）

b.基本类型只有功能完备的值，而每个装箱基本类型除了它对应的基本类型的所有功能值之外，还有个非功能值：null

c.基本类型通常比装箱基本类型更节省空间和时间。

50、如果其他类型更适合，则尽量避免使用字符串

(1)字符串不适合代替其他的值类型

如果数据本身是数值，就应该是 int float 或者 BIgInteger 之类的，如果是一个 是-或-否，这种问题，应该转化为 Boolean。

(2)字符串不适合替代枚举类型

(3)字符串不适合替代聚集类型

如果一个实体有多个组件，用字符串来表示这个实体通常不恰当，如使用 String compundKey = className + "#" + i.next()

很不恰当，显然缺点很多，用分隔符导致结果混乱，访问简单的域也要解析字符串，过程慢而且繁琐，容易出错，无法提供 equals、toString 或者 compareTo 方法

51、当心字符串连接的性能

(1)字符串连接操作符（+）是把多个字符串合并为一个字符串的便利途径。要想产生单独一行的输出，或者构建一个字符串来表示一个较小的、大小固定的对象，使用连接符操作符是非常合适的，但是它不适合运用在大规模的场景中。未连接 n 个字符串而重复使用字符串连接操作符，需要 n 的平方级的时间。这是由于字符串不可变而导致的不幸结果。当两个字符串被连接在一起时，他们的内容都要被拷贝。

(2)使用 StringBuilder 代替 string

52、通过接口引用对象

应该优先使用接口而不是类来引用对象，如下，考虑 Vector 的情况。

```
List<Subscriber> subscribers = new Vector<Subscriber>();

Vector<Subscriber> subscribers = new Vector<Subscriber>();
```

应该使用第一种方式，如果使用接口作为类型，程序将会更加灵活，当决定更换实现时，只需改变构造器中的类的名称。

53、接口优先于反射机制

(1)核心反射机制 java.lang.reflect 提供了“通过程序来访问关于已装载的类的信息”的能力，给定一个 Class 实例，可以获得 Constructor、Method、Field 实例，这些对象提供“通过程序来访问类的成员名称、域类型、方法签名等信息”的能力。

(2)反射机制存在的代价：

a.失去编译时类型检查的好处，包括异常检查。

b.执行反射访问所需的代码很长。

c.性能上的损失。

(3) 简而言之，反射机制是一种功能强大的机制，对于特定的复杂系统编程任务，它是非常必要的，但是它也有一些缺点。如果你编写的程序必须要和你编写的程序编译时未知的类一起工作，如有可能，就应该使用反射机制来实例化对象，而访问对象时则使用编译时已知的某个接口或者超类

54、谨慎地使用本地方法

从历史上看，使用 JNI 主要有三个用途：

(1)由于 Java 程序是运行在虚拟机之上的，虚拟机作为中间件，带来的平台无关性的好处的同时，也使得那些要求访问 OS 甚至硬件的底层操作变得无所适从。通过 JNI 可以调用 C/C++等编写的代码，来提 Java 完成，比如读写 Windows 的注册表，取得硬盘的序列号等。

(2)由于一些“古老”的资源要使用“古老”的代码库，为其再开发一个 Java 版本是不划算的。于是 JNI 又派上用场了。

(3)最后是为了性能。Java 的性能可能是所有语言中几乎最慢的了，关键性的部分可以通过 JNI 调用性能好的语言，如 C/C++/Colob 等。但是，随着 Java 版本的升级，性能在不断提高，**如今绝大多数地方已经不值得使用 JNI 来提高性能了。**

然而，使用 JNI 是有着相当的弊端的。就是从此失去了 Java 的平台无关性。对于第一点中的原因，很好理解吧，你通过 JNI 调用了一段 C++的代码去读 Windows 的注册表。如果移植到其他 OS 上，何来的注册表？

55、谨慎地进行优化

(1)任何优化都存在风险，有时候弄不好反而带来其他的问题

(2)并不是性能优先。努力编写好的程序而不是快的程序。

(3)对前人，尤其是类似于 Java API 这样的成熟代码，进行优化，是不明智的

56、遵守普遍接受的命名惯例

不严格的讲，命名惯例分为两大类：**字面的和语法的。**

(1)字面的命名惯例比较少，但也涉及包，类，方法，域和类型变量。

(2)包的名称应该是层次状的，用“.”分割每个部分。任何将在你的组织之外使用的包，其名称都应该以你的组织的 Internet 域名开头，并且将顶级域名放在前面，例如 com.sun ，gov.nsa。标准类库和一些可选的包，其名称以 java 和 javax 开头，这属于这一规则的例外。

(3)类和接口的名称，包括枚举和注解类型的名称，都应该包括一个或者多个单词，每个单词的首字母大写，例如 Timer 和 TimerTask。应该尽量避免用缩写，除非是一些首字母缩写和一些通用的缩写，比如 max 和 min。

(4)方法和域的名称与类和接口的名称一样，都遵守相同的字面惯例，只不过方法或者域的名称的第一个字母应该小写，例如 remove，ensureCapacity。

(5)唯一的例外是“常量域”，它的名称应该包含一个或者多个大写的单词，中间用下划线隔开，例如 VALUES，NEGATIVE_INFINITY。常量域是静态

(6)类型参数名称通常由单个字母组成。这个字母通常是以下五种类型之一：

T 代表任意的类型 ， E 表示集合的元素类型 ， K 和 V 表示映射的键和值的类型 ， X 表示异常的类型。