代码优化

# 代码优化的目标是：

### 1、减小代码的体积

### 2、提高代码运行的效率

## 1、尽量指定类、方法的final修饰符

带有final修饰符的类是不可派生的。在Java核心API中，有许多应用final的例子，例如java.lang.String，整个类都是final的。为类指定final修饰符可以让类不可以被继承，为方法指定final修饰符可以让方法不可以被重写。如果指定了一个类为final，则该类所有的方法都是final的。Java编译器会寻找机会内联所有的final方法，内联对于提升Java运行效率作用重大，具体参见[Java运行期优化](http://www.cnblogs.com/xrq730/p/4857820.html)。此举能够使性能平均提高50%。

## 2、尽量重用对象 （使用同一个对象）

特别是String对象的使用，出现字符串连接时应该使用StringBuilder/StringBuffer代替。由于Java虚拟机不仅要花时间生成对象，以后可能还需要花时间对这些对象进行垃圾回收和处理，因此，生成过多的对象将会给程序的性能带来很大的影响。

## 3、尽可能使用局部变量（栈去，随着方法运行结束，这些内容会消失）

调用方法时传递的参数以及在调用中创建的临时变量都保存在栈（函数的参数值，局部变量）中，速度较快，栈中创建的变量，随着方法的运行结束，这些内容就没了，不需要额外的垃圾回收。

其他变量，如静态变量、实例变量等，都在堆中创建，由程序员分配和释放，速度较慢。另外

## 4、及时关闭流

Java编程过程中，进行数据库连接、I/O流操作时务必小心，在使用完毕后，及时关闭以释放资源。因为对这些大对象的操作会造成系统大的开销，稍有不慎，将会导致严重的后果。

## 5、尽量减少对变量的重复计算

for (int i = 0; i < list.size(); i++)

{...}

建议替换为：

length = list.size()

for (int i = 0; i < length; i++)

{...}

## 6、尽量采用懒加载的策略，即在需要的时候才创建

例如：

String str = "aaa";

if (i == 1)

{

　　list.add(str);

}

建议替换为：

if (i == 1)

{

　　String str = "aaa";

　　list.add(str);

}

## 7、慎用异常

异常对性能不利。抛出异常首先要创建一个新的对象，Throwable接口的构造函数调用名为fillInStackTrace()的本地同步方法，fillInStackTrace()方法检查堆栈，收集调用跟踪信息。只要有异常被抛出，Java虚拟机就必须调整调用堆栈，因为在处理过程中创建了一个新的对象。异常只能用于错误处理，不应该用来控制程序流程。

## 8、尽量使用HashMap、ArrayList、StringBuilder，除非线程安全需要，否则不推荐使用Hashtable、Vector、StringBuffer，后三者由于使用同步机制而导致了性能开销

## 9、不要将数组声明为public static final

因为这毫无意义，这样只是定义了引用为static final，数组的内容还是可以随意改变的，将数组声明为public更是一个安全漏洞，这意味着这个数组可以被外部类所改变

**10、使用同步代码块替代同步方法**

这点在多线程模块中的[synchronized锁方法块](http://www.cnblogs.com/xrq730/p/4851530.html)一文中已经讲得很清楚了，除非能确定一整个方法都是需要进行同步的，否则尽量使用同步代码块，避免对那些不需要进行同步的代码也进行了同步，影响了代码执行效率。

## 11、将常量声明为static final，并以大写命名

这样在编译期间就可以把这些内容放入常量池中，避免运行期间计算生成常量的值。另外，将常量的名字以大写命名也可以方便区分出常量与变量