代码优化

参考地址：http://www.cnblogs.com/xrq730/p/4865416.html

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。此举能够使性能平均提高50%。

（2）尽量重用对象

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

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中，速度较快，其他变量，如静态变量、实例变量等，都在堆中创建，速度较慢。另外，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。

（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

明确一个概念，对方法的调用，即使方法中只有一句语句，也是有消耗的，包括创建栈帧、调用方法时保护现场、调用方法完毕时恢复现场等。所以例如下面的操作：

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

建议替换为：

```
for (int i = 0, length = list.size(); i < length; i++)
{...}
```

这样，在list.size()很大的时候，就减少了很多的消耗

（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）不要在循环中使用try…catch…，应该把其放在最外层

根据网友们提出的意见，这一点我认为值得商榷

9）如果能估计到待添加的内容长度，为底层以数组方式实现的集合、工具类指定初始长度

比如ArrayList、LinkedLlist、StringBuilder、StringBuffer、HashMap、HashSet等等，以StringBuilder为例：

- StringBuilder()　　// 默认分配16个字符的空间
- StringBuilder(int size)　　// 默认分配size个字符的空间
- StringBuilder(String str)　　// 默认分配16个字符+str.length()个字符空间

可以通过类（这里指的不仅仅是上面的StringBuilder）的构造函数来设定它的初始化容量，这样可以明显地提升性能。比如StringBuilder吧，length表示当前的StringBuilder能保持的字符数量。因为当StringBuilder达到最大容量的时候，它会将自身容量增加到当前的2倍再加2，无论何时只要StringBuilder达到它的最大容量，它就不得不创建一个新的字符数组然后将旧的字符数组内容拷贝到新字符数组中----这是十分耗费性能的一个操作。试想，如果能预估到字符数组中大概要存放5000个字符而不指定长度，最接近5000的2次幂是4096，每次扩容加的2不管，那么：

- 在4096 的基础上，再申请8194个大小的字符数组，加起来相当于一次申请了12290个大小的字符数组，如果一开始能指定5000个大小的字符数组，就节省了一倍以上的空间
- 把原来的4096个字符拷贝到新的的字符数组中去

这样，既浪费内存空间又降低代码运行效率。所以，给底层以数组实现的集合、工具类设置一个合理的初始化容量是错不了的，这会带来立竿见影的效果。但是，注意，**像HashMap这种是以数组+链表实现的集合，别把初始大小和你估计的大小设置得一样，因为一个table上只连接一个对象的可能性几乎为0**。初始大小建议设置为2的N次幂，如果能估计到有2000个元素，设置成new HashMap(128)、new HashMap(256)都可以。

（10）当复制大量数据时，使用System.arraycopy()命令

（11）循环内不要不断创建对象引用

例如：

```
for (int i = 1; i <= count; i++)
{
    Object obj = new Object();
}
```

这种做法会导致内存中有count份Object对象引用存在，count很大的话，就耗费内存了，建议为改为：

```
Object obj = null;
for (int i = 0; i <= count; i++)
{
    obj = new Object();
}
```

这样的话，内存中只有一份Object对象引用，每次new Object()的时候，Object对象引用指向不同的Object罢了，但是内存中只有一份，这样就大大节省了内存空间了。

（12）基于效率和类型检查的考虑，应该尽可能使用array，无法确定数组大小时才使用ArrayList