此文章已于 13:07:42 2017/12/15 发布到 HealerJean梦想博客

JAVA代码优化

类别 java

# 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、对象和变量相关

### 1、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++)

{...}

### 2、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);

}

### 3、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。

### 1、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

### 2、切记以常量定义的方式替代魔鬼数字，魔鬼数字的存在将极大地降低代码可读性，字符串常量是否使用常量定义可以视情况而定

#### 1、（我之前工作的地方，全他妈是魔鬼数字都想哭了）

#### 2、比如数据字典的使用，利用常量

### 3、尽可能使用局部变量（栈存的方式，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

## 2、集合数组相关

### 1、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

### 2、不要将数组声明为public static final，属于傻B性质的浪费资源

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

### 3、不要对数组使用toString()方法

看一下对数组使用toString()打印出来的是什么：

public static void main(String[] args)

{

int[] is = new int[]{1, 2, 3};

System.out.println(is.toString());

}

结果是：

[I@18a992f

本意是想打印出数组内容，却有可能因为数组引用is为空而导致空指针异常。不过虽然对数组toString()没有意义，但是对集合toString()是可以打印出集合里面的内容的，因为集合的父类AbstractCollections<E>重写了Object的toString()方法。

## 3、线程相关

### 1、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。

## 4、基本数据

### 1、把一个基本数据类型转为字符串，基本数据类型.toString()是最快的方式、String.valueOf(数据)次之、数据+""最慢

### 2、****long或者Long初始赋值时，使用大写的L而不是小写的l，因为字母l极易与数字1混淆，这个点非常细节，值得注意****

## 5、继承相关

### 1、所有重写的方法必须保留@Override注解

#### 1、清楚地可以知道这个方法由父类继承而来

#### 2、getObject()和get0bject()方法，前者第四个字母是"O"，后者第四个子母是"0"，加了@Override注解可以马上判断是否重写成功

#### 3、在抽象类中对方法签名进行修改，实现类会马上报出编译错误

## 6、程序准备

### 1、程序运行过程中避免使用反射

反射是Java提供给用户一个很强大的功能，功能强大往往意味着效率不高。不建议在程序运行过程中使用尤其是频繁使用反射机制，特别是Method的invoke方法，如果确实有必要，一种建议性的做法是将那些需要通过反射加载的类在项目启动的时候通过反射实例化出一个对象并放入内存----用户只关心和对端交互的时候获取最快的响应速度，并不关心对端的项目启动花多久时间。

### 2、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(有点废话，都不能继承了，哪里来的方法重写，肯定是final啊笨，)。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%。

### 3、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。

### 4、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。能不使用异常，就不使用

### 5、对资源的close()建议分开操作

意思是，比如我有这么一段代码：

[复制代码](javascript:void(0);)

try

{

XXX.close();

YYY.close();

}

catch (Exception e)

{

...

}

[复制代码](javascript:void(0);)

建议修改为：

[复制代码](javascript:void(0);)

try

{

XXX.close();

}

catch (Exception e)

{

...

}

try

{

YYY.close();

}

catch (Exception e)

{

...

}

[复制代码](javascript:void(0);)

### 解释：虽然有些麻烦，却能避免资源泄露。我们想，如果没有修改过的代码，万一XXX.close()抛异常了，那么就进入了catch块中了，YYY.close()不会执行，YYY这块资源就不会回收了，一直占用着，这样的代码一多，是可能引起资源句柄泄露的。而改为下面的写法之后，就保证了无论如何XXX和YYY都会被close掉

**6、不要创建一些不使用的对象，不要导入一些不使用的类**

这毫无意义，如果代码中出现"The value of the local variable i is not used"、"The import java.util is never used"，那么请删除这些无用的内容

# 二、代码性优化

## 1、****字符串变量和字符串常量equals****的****时候将字符串常量写在前面****

### 理由：避免空指针异常

这是一个比较常见的小技巧了，如果有以下代码：

String str = "123";

if (str.equals("123"))

{  
 ...  
}

建议修改为：

String str = "123";

if ("123".equals(str))

{

...

}