# 一、CoreJava 部分：

## 基础及语法部分：

**1、面向对象的特征有哪些方面? 【基础】**

答：面向对象的特征主要有以下几个方面：

1)抽象：抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象,二是数据抽象。

2)继承：继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。

3)封装：封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象,这些对象通过一个受保护的接口访问其他对象。

4)多态性：多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题。

**2、作用域public,private,protected,以及不写时的区别？【基础】**

答：区别如下：

作用域 当前类 同包 子孙类 其他

public √ √ √ √

protected √ √ √ ×

default √ √ × ×

private √ × × ×

不写时默认为default。

**3、String 是最基本的数据类型吗? 【基础】**

答：不是。

**4、float 型float f=3.4是否正确? 【基础】**

答:不正确；精度不准确,应该用强制类型转换，如下所示：float f=(float)3.4 。

**5、语句float f=1.3；编译能否通过？【基础】**

答:不能；应该用强制类型转换，如下所示：float f=(float)1.3； 。

**6、short s1 = 1; s1 = s1 + 1;有什么错?**

**short s1 = 1; s1 += 1;有什么错? 【基础】**

答：short s1 = 1; s1 = s1 + 1;s1+1运算结果是int 型，需要强制转换类型；short s1 = 1; s1 += 1;可以正确编译,自动类型提升。

**7、Java 有没有goto? 【基础】**

答：goto 是java 中的保留字，现在没有在java 中使用。

**8、int 和Integer 有什么区别? 【基础】**

答：Java 提供两种不同的类型：引用类型和原始类型（或内置类型）；

int 是java 的原始数据类型，Integer 是java 为int 提供的封装类。

Java 为每个原始类型提供了封装类：

原始类型: boolean,char,byte,short,int,long,float,double

封装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为null，而原始类型实例变量的缺省值与它们的类型有关。

**9、&和&&的区别？【基础】**

答：&是位运算符，表示按位与运算，&&是逻辑运算符，表示逻辑与（and）。

**10、简述逻辑操作(&,|,^)与条件操作(&&,||)的区别？【基础】**

答：区别主要有两点：a.条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型b.逻辑操作不会产生短路。

**11、heap 和stack 有什么区别？【基础】**

答：栈是一种线形集合，其添加和删除元素的操作应在同一段完成，栈按照后进先出的方式进行处理；堆是栈的一个组成元素。

**12、Math.round(11.5) 等于多少? Math.round(-11.5)等于多少? 【基础】**

答：Math.round(11.5)==12 Math.round(-11.5)==-11 round 方法返回与参数最接近的长整数，参数加1/2 后求其floor。

**13、swtich 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上? 【基础】**

答：switch（expr1）中，expr1 是一个整数表达式。因此传递给switch 和case语句的参数应该是int、short、char 或者byte。long,string 都不能作用于swtich。

**14、编程题: 用最有效率的方法算出2 乘以8 等於几? 【基础】**

答： 2 << 3。

**15、有没有length()这个方法? String 有没有length()这个方法？【基础】**

答：数组没有length()这个方法，有length 的属性。String 有length()这个方法。

**16、在JAVA 中，如何跳出当前的多重嵌套循环？【基础】**

答：在最外层循环前加label 标识,然后用break:label 方法即可跳出多重循环。

**17、构造器Constructor 是否可被override? 【基础】**

答：构造器Constructor 不能被继承，因此不能重写Overriding，但可以被重载Overloading。

**18、两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对? 【基础】**

答：不对，有相同的hash code。

**19、是否可以继承String 类? 【基础】**

答：String 类是final 类，故不可以继承。

**20、以下二条语句返回值为true 的有：**

A：“beijing”==“beijing”；

B：“beijing”.equalsIgnoreCase（new String（“beijing”））；【基础】

答：A 和B 。

**21、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递? 【基础】**

答：是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

**22、我们在web 应用开发过程中经常遇到输出某种编码的字符，如iso8859-1等，如何输出一个某种编码的字符串？【基础】**

答：public String translate(String str){

String tempStr = "";

try{

tempStr = new String(str.getBytes("ISO-8859-1"), "GBK");

tempStr = tempStr.trim();

}catch (Exception e){

System.err.println(e.getMessage());

}

return tempStr;

}

**23、String 和StringBuffer 的区别? 【基础】**

答：JAVA 平台提供了两个类：String 和StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个String 类提供了数值不可改变的字符串。而这个StringBuffer 类提供的字符串进行修改。当你知道字符数据要改变的时候你就可以使用StringBuffer。典型地，你可以使用StringBuffers 来动态构造字符数据。

**24、String, StringBuffer StringBuilder 的区别。【基础】**

答：String 的长度是不可变的；StringBuffer 的长度是可变的，如果你对字符串中的内容经常进行操作，特别是内容要修改时，那么使用StringBuffer，如果最后需要String，那么使用StringBuffer 的toString()方法；线程安全；StringBuilder 是从JDK 5 开始，为StringBuffer 该类补充了一个单个线程使用的等价类；通常应该优先使用StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。

**25、Overload 和Override 的区别。Overloaded 的方法是否可以改变返回值的类型? 【基础】**

答：方法的重写Overriding 和重载Overloading 是Java 多态性的不同表现。重写Overriding 是父类与子类之间多态性的一种表现，重载Overloading 是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写(Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded 的方法是可以改变返回值的类型。

**26、定义类A 和类B 如下：【基础】**

class A {

int a=1;

double d=2.0;

void show(){

System.out.println("Class A: a="+a +"\td="+d);

}

}

class B extends A{

float a=3.0f;

String d="Java program.";

void show(){

super.show( );

System.out.println("Class B: a="+a +"\td="+d);

}

}

(1) 若在应用程序的main 方法中有以下语句：

A a=new A();

a.show();

则输出的结果如何？

(2) 若在应用程序的main 方法中定义类B 的对象b：

A b=new B();

b.show();

则输出的结果如何？

答：输出结果为：

1）Class A: a=1 d=2.0 ；

2）Class A: a=1 d=2.0

Class B: a=3.0 d=Java program。

**27、描述一下JVM 加载class 文件的原理机制? 【基础】**

答：JVM 中类的装载是由ClassLoader 和它的子类来实现的,Java ClassLoader是一个重要的Java 运行时系统组件。它负责在运行时查找和装入类文件的类。

**28、char 型变量中能不能存贮一个中文汉字?为什么? 【基础】**

答：能够定义成为一个中文的，因为java 中以unicode 编码，一个char 占16个字节，所以放一个中文是没问题的。

**29、abstract class 和interface 有什么区别? 【基础】**

答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），。然而可以创建一个变量，其类型是一个抽象类，它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建abstract 类的实例并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类为。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。接口（interface）是抽象类的变体。新型多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，所有成员变量都是publicstatic final 的。一个类可以实现多个接口，当类实现特殊接口时，它定义（即

将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

**30、Static Nested Class 和Inner Class 的不同？【基础】**

答：Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化。

**31、java 中会存在内存泄漏吗，请简单描述。【基础】**

答：会；存在无用但可达的对象，这些对象不能被GC 回收，导致耗费内存资源。

**32、abstract 的method 是否可同时是static,是否可同时是native，是否可同时是synchronized? 【基础】**

答：都不能。

**33、静态变量和实例变量的区别？【基础】**

答：静态变量也称为类变量，归全类共有，它不依赖于某个对象，可通过类名直接访问；而实例变量必须依存于某一实例，只能通过对象才能访问到它。

**34、是否可以从一个static 方法内部发出对非static 方法的调用？【基础】**

答：不可以,如果其中包含对象的method()，不能保证对象初始化。

**35、写clone()方法时，通常都有一行代码，是什么？【基础】**

答：Clone 有缺省行为：super.clone()，他负责产生正确大小的空间，并逐位复制。

**36、GC 是什么? 为什么要有GC? 【基础】**

答：GC 是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc() 。

**37、垃圾回收的优点和原理。并考虑2 种回收机制。【基础】**

答：Java 语言中一个显著的特点就是引入了垃圾回收机制，使c++程序员最头疼的内存管理的问题迎刃而解，它使得Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

**38、垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？【基础】**

答：对于GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。可以。程序员可以手动执行System.gc()，通知GC 运行，但是Java 语言规范并不保证GC 一定会执行。

**39、String s=new String(“xyz”);创建了几个String Object? 【基础】**

答：两个对象，一个是"xyx",一个是指向"xyx"的引用对象s。

**40、接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承实体类(concrete class)? 【基础】**

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承实体类，但前提是实体类必须有明确的构造函数。

**41、Java 的接口和C++的虚类的相同和不同处。【基础】**

答：由于Java 不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面的方法或属性，现有的单继承机制就不能满足要求。与继承相比，接口有更高的灵活性，因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的方法和属性，并且接口里面的属性在默认状态下面都是public static,所有方法默认情况下是public.一个类可以实现多个接口。

**42、一个“.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？【基础】**

答：可以；必须只有一个类名与文件名相同。

**43、说出一些常用的类，包，接口，请各举5 个。【基础】**

答：常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer；

常用的包：java.lang java.awt java.io java.util java.sql；

常用的接口：Remote List Map Document NodeList

**44、Anonymous Inner Class (匿名内部类) 是否可以extends(继承)其它类？是否可以implements(实现)interface(接口)? 【基础】**

答：可以继承其他类或实现其他接口，在swing 编程中常用此方式。

**45、内部类可以引用他包含类的成员吗？有没有什么限制？【基础】**

答：一个内部类对象可以访问创建它的外部类对象的内容。

**46、java 中实现多态的机制是什么？【基础】**

答：方法的覆盖Overriding 和重载Overloading 是java 多态性的不同表现；覆盖Overriding 是父类与子类之间多态性的一种表现，重载Overloading 是一个类中多态性的一种表现。

**47、在java 中一个类被声明为final 类型，表示了什么意思？【基础】**

答：表示该类不能被继承，是顶级类。

**48、下面哪些类可以被继承? 【基础】**

1）java.lang.Thread (T)

2）java.lang.Number (T)

3）java.lang.Double (F)

4）java.lang.Math (F)

5）java.lang.Void (F)

6）java.lang.Class (F)

7）java.lang.ClassLoader (T)

答：1、2、7 可以被继承。

**49、指出下面程序的运行结果: 【基础】**

class A{

static{

System.out.print("1");

}

public A(){

System.out.print("2");

}

}

class B extends A{

static{

System.out.print("a");

}

public B(){

System.out.print("b");

}

}

public class Hello{

public static void main(String[] ars){

A ab = new B(); //执行到此处,结果: 1a2b

ab = new B(); //执行到此处,结果: 1a2b2b

}

}

答：输出结果为1a2b2b；类的static 代码段,可以看作是类首次加载(虚拟机加载)执行的代码,而对于类加载,首先要执行其基类的构造,再执行其本身的构造。

**50、继承时候类的执行顺序问题,一般都是选择题,问你将会打印出什么?【基础】**

父类：

package test;

public class FatherClass {

public FatherClass() {

System.out.println("FatherClass Create");

}

}

子类:

package test;

import test.FatherClass;

public class ChildClass extends FatherClass {

public ChildClass() {

System.out.println("ChildClass Create");

}

public static void main(String[] args) {

FatherClass fc = new FatherClass();

ChildClass cc = new ChildClass();

}

}

答：输出结果为：

FatherClass Create

FatherClass Create

ChildClass Create

**51、内部类的实现方式? 【基础】**

答：示例代码如下：

package test;

public class OuterClass {

private class InterClass {

public InterClass() {

System.out.println("InterClass Create");

}

}

public OuterClass() {

InterClass ic = new InterClass();

System.out.println("OuterClass Create");

}

public static void main(String[] args) {

OuterClass oc = new OuterClass();

}

}

输出结果为:

InterClass Create

OuterClass Create

**52、关于内部类： 【基础】**

public class OuterClass {

private double d1 = 1.0;

//insert code here

}

You need to insert an inner class declaration at line 3，Which two

inner class declarations are valid?(Choose two.)

A. class InnerOne{

public static double methoda() {return d1;}

}

B. public class InnerOne{

static double methoda() {return d1;}

}

C. private class InnerOne{

double methoda() {return d1;}

}

D. static class InnerOne{

protected double methoda() {return d1;}

}

E. abstract class InnerOne{

public abstract double methoda();

}

答：答案为C、E；说明如下：

1）静态内部类可以有静态成员，而非静态内部类则不能有静态成员；故A、B错；

2）静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量；故D 错；

3）非静态内部类的非静态成员可以访问外部类的非静态变量；故C 正确。

**53、数据类型之间的转换:**

1)如何将数值型字符转换为数字？

2)如何将数字转换为字符？

3)如何取小数点前两位并四舍五入? 【基础】

答：1)调用数值类型相应包装类中的方法parse\*\*\*(String)或valueOf(String)即可返回相应基本类型或包装类型数值；

2)将数字与空字符串相加即可获得其所对应的字符串;另外对于基本类型数字还可调用String 类中的valueOf(…)方法返回相应字符串,而对于包装类型数字则可调用其toString()方法获得相应字符串；

3)可用该数字构造一java.math.BigDecimal 对象,再利用其round()方法进行四舍五入到保留小数点后两位,再将其转换为字符串截取最后两位。

**54、字符串操作：如何实现字符串的反转及替换？【基础】**

答：可用字符串构造一StringBuffer 对象,然后调用StringBuffer 中的reverse方法即可实现字符串的反转,调用replace 方法即可实现字符串的替换。

**55、编码转换：怎样将GB2312 编码的字符串转换为ISO-8859-1 编码的字符串？【基础】**

答：示例代码如下:

String s1 = "你好";

String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1");

**56、写一个函数，要求输入一个字符串和一个字符长度，对该字符串进行分隔。【基础】**

答：函数代码如下：

public String[] split(String str, int chars){

int n = (str.length()+ chars - 1)/chars;

String ret[] = new String[n];

for(int i=0; i<n; i++){

if(i < n-1){

ret[i] = str.substring(i\*chars , (i+1)\*chars);

}else{

ret[i] = str.substring(i\*chars);

}

}

return ret;

}

**57、写一个函数，2 个参数，1 个字符串，1 个字节数，返回截取的字符串，要求字符串中的中文不能出现乱码：如（“我ABC”，4）应该截为“我AB”，输入（“我ABC 汉DEF”，6）应该输出为“我ABC”而不是“我ABC+汉的半个”。【基础】**

答：代码如下：

public String subString(String str, int subBytes) {

int bytes = 0; // 用来存储字符串的总字节数

for (int i = 0; i < str.length(); i++) {

if (bytes == subBytes) {

return str.substring(0, i);

}

char c = str.charAt(i);

if (c < 256) {

bytes += 1; // 英文字符的字节数看作1

} else {

bytes += 2; // 中文字符的字节数看作2

if(bytes - subBytes == 1){

return str.substring(0, i);

}

}

}

return str;

}

**58、日期和时间：**

**1)如何取得年月日、小时分秒？**

**2)如何取得从1970 年到现在的毫秒数？**

**3)如何取得某个日期是当月的最后一天？**

**4)如何格式化日期？【基础】**

答：1)创建java.util.Calendar 实例(Calendar.getInstance()),调用其get()方法传入不同的参数即可获得参数所对应的值,如：calendar.get(Calendar.YEAR);//获得年

2)以下方法均可获得该毫秒数:Calendar.getInstance().getTimeInMillis();System.currentTimeMillis();

3)示例代码如下:

Calendar time = Calendar.getInstance();

time.set(Calendar.DAY\_OF\_MONTH,

time.getActualMaximum(Calendar.DAY\_OF\_MONTH));

4)利用java.text.DataFormat 类中的format()方法可将日期格式化。

**59、Java 编程,打印昨天的当前时刻。【基础】**

答：public class YesterdayCurrent{

public static void main(String[] args){

Calendar cal = Calendar.getInstance();

cal.add(Calendar.DATE, -1);

System.out.println(cal.getTime());

}

}

**60、java 和javasciprt 的区别。【基础】**

答：JavaScript 与Java 是两个公司开发的不同的两个产品。Java 是SUN 公司推出的新一代面向对象的程序设计语言，特别适合于Internet 应用程序开发；而JavaScript 是Netscape 公司的产品，其目的是为了扩展Netscape Navigator功能,而开发的一种可以嵌入Web 页面中的基于对象和事件驱动的解释性语言,它的前身是Live Script；而Java 的前身是Oak 语言。下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object Based）和事件驱动（Event Driver）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用；

2）解释和编译：Java 的源代码在执行之前，必须经过编译；JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行；

3）强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型；

4）代码格式不一样。

**61、什么时候用assert？【中等难度】**

答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后， assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式计算为false，那么系统会报告一个Assertionerror。

断言用于调试目的：

assert(a > 0); // throws an Assertionerror if a <= 0

断言可以有两种形式：

assert Expression1 ;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的String 消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用source 1.4 标记：

javac -source 1.4 Test.java

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用-da 或者-disableassertions 标记。

要在系统类中启用断言，可使用-esa 或者-dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

## 异常部分

**62、Java 中的异常处理机制的简单原理和应用？【基础】**

答：当JAVA 程序违反了JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表示为一个异常。违反语义规则包括2 种情况。一种是JAVA 类库内置的语义检查。例如数组下标越界,会引发IndexOutOfBoundsException;访问null 的对象时会引发NullPointerException。另一种情况就是JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用throw 关键字引发异常。所有的异常都是java.lang.Thowable 的子类。

**63、error 和exception 有什么区别? 【基础】**

答：error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

**64、try {}里有一个return 语句，那么紧跟在这个try 后的finally {}里的code会不会被执行，什么时候被执行，在return 前还是后? 【基础】**

答：会执行，在return 前执行。

**65、JAVA 语言如何进行异常处理，关键字：throws,throw,try,catch,finally分别代表什么意义？在try 块中可以抛出异常吗？【基础】**

答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在Java 中，每个异常都是一个对象，它是Throwable 类或其它子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过5 个关键词来实现的：try、catch、throw、throws 和finally。一般情况下是用try 来执行一段程序，如果出现异常，系统会抛出（throws）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try 用来指定一块预防所有“异常”的程序；catch 子句紧跟在try 块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws 用来标明一个成员函数可能抛出的各种“异常”；Finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个try 语句，在这个成员函数内部写另一个try 语句保护其他代码。每当遇到一个try 语句，“异常”的框架就放到堆栈上面，直到所有的try 语句都完成。如果下一级的try 语句没有对某种“异常”进行处理，堆栈就会展开，直到遇到有处理这种“异常”的try 语句。

**66、运行时异常与一般异常有何异同？【基础】**

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

**67、给我一个你最常见到的runtime exception？【基础】**

答：ArithmeticException, ArrayStoreException, BufferOverflowException,

BufferUnderflowException, CannotRedoException, CannotUndoException,

ClassCastException, CMMException, ConcurrentModificationException,

DOMException, EmptyStackException, IllegalArgumentException,

IllegalMonitorStateException, IllegalPathStateException,

IllegalStateException, ImagingOpException, IndexOutOfBoundsException,

MissingResourceException, NegativeArraySizeException,

NoSuchElementException, NullPointerException, ProfileDataException,

ProviderException, RasterFormatException, SecurityException,

SystemException, UndeclaredThrowableException,

UnmodifiableSetException, UnsupportedOperationException

**68、final, finally, finalize 的区别? 【基础】**

答：final：修饰符（关键字）；如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承，因此一个类不能既被声明为abstract的，又被声明为final 的；将变量或方法声明为final，可以保证它们在使用中不被改变；被声明为final 的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改；被声明为final 的方法也同样只能使用，不能重载。finally：再异常处理时提供finally 块来执行任何清除操作；如果抛出一个异常，那么相匹配的catch 子句就会执行，然后控制就会进入finally 块（如果有的话）。finalize：方法名；Java 技术允许使用finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在Object 类中定义的，因此所有的类都继承了它。子类覆盖finalize() 方法以整理系统资源或者执行其他清理工作。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

**69、类Example A 继承Exception，类ExampleB 继承Example A；【基础】**

有如下代码片断：

try{

throw new ExampleB(“b”)；

}catch（ExampleA e）{

System.out.printfln（“ExampleA”）；

}catch（Exception e）{

System.out.printfln（“Exception”）；

}

输出的内容应该是：

A：ExampleA B：Exception C：b D：无

答：输出为A。

## 集合部分

**70、介绍JAVA 中的Collection FrameWork(及如何写自己的数据结构)【基础】**

答：Collection FrameWork 如下：

Collection

├List

│├LinkedList

│├ArrayList

│└Vector

│ └Stack

└Set

Map

├Hashtable

├HashMap

└WeakHashMap

Collection 是最基本的集合接口，一个Collection 代表一组Object，即Collection 的元素（Elements）； Map 提供key 到value 的映射。

**71、List,Set,Map 是否继承自Collection 接口？【基础】**

答：List,Set 是；Map 不是。

**72、你所知道的集合类都有哪些？主要方法？【基础】**

答：最常用的集合类是List 和Map。List 的具体实现包括ArrayList 和Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。List 适用于按数值索引访问元素的情形。Map 提供了一个更通用的元素存储方法。Map 集合类用于存储元素对（称作“键”和“值”），其中每个键映射到一个值。

**73、说出ArrayList,Vector, LinkedList 的存储性能和特性？【基础】**

答：ArrayList 和Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了synchronized 方法（线程安全），通常性能上较ArrayList 差，而LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

**74、Collection 和Collections 的区别？【基础】**

答：Collection 是java.util 下的接口，它是各种集合的父接口，继承于它的接口主要有Set 和List；Collections 是个java.util 下的类，是针对集合的帮助类，提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

**75、HashMap 和Hashtable 的区别? 【基础】**

答：二者都实现了Map 接口，是将惟一键映射到特定的值上；主要区别在于：

1)HashMap 没有排序，允许一个null 键和多个null 值,而Hashtable 不允许；

2)HashMap 把Hashtable 的contains 方法去掉了，改成containsvalue 和containsKey,因为contains 方法容易让人引起误解；

3)Hashtable 继承自Dictionary 类，HashMap 是Java1.2 引进的Map 接口的实现；

4)Hashtable 的方法是Synchronize 的，而HashMap 不是，在多个线程访问Hashtable 时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。Hashtable 和HashMap 采用的hash/rehash 算法大致一样，所以性能不会有很大的差异。

**76、Arraylist 与Vector 区别？【基础】**

答：就ArrayList 与Vector 主要从二方面来说：

1）同步性：Vector 是线程安全的（同步），而ArrayList 是线程序不安全的；

2）数据增长：当需要增长时,Vector 默认增长一倍，而ArrayList 却是一半。

**77、List、Map、Set 三个接口，存取元素时，各有什么特点？【基础】**

答：List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存key-value 值，value 可多值。

**78、Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals()? 它们有何区别? 【基础】**

答：Set 里的元素是不能重复的，用equals ()方法来区分重复与否。覆盖equals()方法用来判断对象的内容是否相同，而”==”判断地址是否相等,用来决定引用值是否指向同一对象。

**79、用程序给出随便大小的10 个数，序号为1-10，按从小到大顺序输出，并输出相应的序号。【基础】**

答：代码如下：

package test;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

public class RandomSort {

public static void printRandomBySort() {

Random random = new Random(); // 创建随机数生成器

List list = new ArrayList();

// 生成10 个随机数，并放在集合list 中

for (int i = 0; i < 10; i++) {

list.add(random.nextInt(1000));

}

Collections.sort(list); // 对集合中的元素进行排序

Iterator it = list.iterator();

int count = 0;

while (it.hasNext()) { // 顺序输出排序后集合中的元素

System.out.println(++count + ": " + it.next());

}

}

public static void main(String[] args) {

printRandomBySort();

}

}

**80、用JAVA 实现一种排序，JAVA 类实现序列化的方法？ 在COLLECTION 框架中，实现比较要实现什么样的接口？【基础】**

答：用插入法进行排序代码如下：

package test;

import java.util.\*;

class InsertSort {

ArrayList al;

public InsertSort(int num,int mod) {

al = new ArrayList(num);

Random rand = new Random();

System.out.println("The ArrayList Sort Before:");

for (int i=0;i<num ;i++ ){

al.add(new Integer(Math.abs(rand.nextInt()) % mod +

1));

System.out.println("al["+i+"]="+al.get(i));

}

}

public void SortIt(){

tempInt;

int MaxSize=1;

for(int i=1;i<al.size();i++){

tempInt = (Integer)al.remove(i);

if(tempInt.intValue() >=

((Integer)al.get(MaxSize-1)).intValue()){

al.add(MaxSize,tempInt);

MaxSize++;

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

}else{

for (int j=0;j<MaxSize ;j++ ){

if (((Integer)al.get(j)).intValue()

>=tempInt.intValue()){

al.add(j,tempInt);

MaxSize++;

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

break;

}

}

}

}

System.out.println("The ArrayList Sort After:");

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

System.out.println("al["+i+"]="+al.get(i));

}

}

public static void main(String[] args){

InsertSort is = new InsertSort(10,100);

is.SortIt();

}

}

JAVA 类实现序例化的方法是实现java.io.Serializable 接口；Collection 框架中实现比较要实现Comparable 接口和Comparator 接口。

## 线程部分

**81、sleep() 和wait() 有什么区别? 【基础】**

答：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep 不会释放对象锁。wait 是Object 类的方法，对此对象调用wait 方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify 方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

**82、当一个线程进入一个对象的一个synchronized 方法后，其它线程是否可进入此对象的其它方法? 【基础】**

答：其它线程只能访问该对象的其它非同步方法，同步方法则不能进入。

**83、请说出你所知道的线程同步的方法。【基础】**

答：wait():使一个线程处于等待状态，并且释放所持有的对象的lock；sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException 异常；notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM 确定唤醒哪个线程，而且不是按优先级；

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

**84、多线程有几种实现方法,都是什么?同步有几种实现方法,都是什么? 【基础】**

答：多线程有两种实现方法，分别是继承Thread 类与实现Runnable 接口,同步的实现方面有两种，分别是synchronized,wait 与notify。

**85、同步和异步有何异同，在什么情况下分别使用他们？举例说明。【基础】**

答：如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

**86、启动一个线程是用run()还是start()?【基础】**

答：启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM 调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

**87、线程的基本概念、线程的基本状态以及状态之间的关系？【基础】**

答：线程指在程序执行过程中，能够执行程序代码的一个执行单位，每个程序至少都有一个线程，也就是程序本身；Java 中的线程有四种状态分别是：运行、就绪、挂起、结束。

**88、简述synchronized 和java.util.concurrent.locks.Lock 的异同？【中等难度】**

答：主要相同点：Lock 能完成synchronized 所实现的所有功能；主要不同点：Lock 有比synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而Lock 一定要求程序员手工释放，并且必须在finally 从句中释放。

**89、java 中有几种方法可以实现一个线程？用什么关键字修饰同步方法?stop()和suspend()方法为何不推荐使用？【中等难度】**

答：有两种实现方法，分别是继承Thread 类与实现Runnable 接口；用synchronized 关键字修饰同步方法；反对使用stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在；suspend()方法容易发生死锁。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被“挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。故不应该使用suspend()，而应在自己的Thread 类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用wait()命其进入等待状态。若标志指出线程应当恢复，则用一个notify()重新启动线程。

**90、设计4 个线程，其中两个线程每次对j 增加1，另两个线程对j 每次减少1；写出程序。【中等难度】**

答：以下程序使用内部类实现线程，对j 增减的时候没有考虑顺序问题：

public class TestThread {

private int j;

public TestThread(int j) {this.j = j;}

private synchronized void inc(){

j++;

System.out.println(j + "--Inc--" +

Thread.currentThread().getName());

}

private synchronized void dec(){

j--;

System.out.println(j + "--Dec--" +

Thread.currentThread().getName());

}

public void run() {

(new Dec()).start();

new Thread(new Inc()).start();

(new Dec()).start();

new Thread(new Inc()).start();

}

class Dec extends Thread {

public void run() {

for(int i=0; i<100; i++){

dec();

}

}

}

class Inc implements Runnable {

public void run() {

for(int i=0; i<100; i++){

inc();

}

}

}

public static void main(String[] args) {

(new TestThread(5)).run();

}

}

# IO 流及Socket 部分

**91、什么是java 序列化，如何实现java 序列化？【基础】**

答：序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题；序列化的实现：将需要被序列化的类实现Serializable 接口，该接口没有需实现的方法，implements Serializable 只是为了标注该对象是可被序列化的，然后使用一个输出流(如FileOutputStream)来构造一个ObjectOutputStream(对象流)对象，接着，使用ObjectOutputStream 对象的writeObject(Object obj)方法就可以将参数为obj 的对象写出(即保存其状态)，要恢复的话则用输入流。

**92、java 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？【基础】**

答：字节流，字符流。字节流继承于InputStream、OutputStream，字符流继承于Reader、Writer。在java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

**93、文件和目录（IO）操作：**

1)如何列出某个目录下的所有文件？

2)如何列出某个目录下的所有子目录？

3)如何判断一个文件或目录是否存在？

4)如何读写文件？【基础】

答：1)示例代码如下:

File file = new File("e:\\总结");

File[] files = file.listFiles();

for(int i=0; i<files.length; i++){

if(files[i].isFile()) System.out.println(files[i]);

}

2)示例代码如下:

File file = new File("e:\\总结");

File[] files = file.listFiles();

for(int i=0; i<files.length; i++){

if(files[i].isDirectory()) System.out.println(files[i]);

}

3)创建File 对象,调用其exsit()方法即可返回是否存在,如:

System.out.println(new File("d:\\t.txt").exists());

4)示例代码如下:

//读文件:

FileInputStream fin = new FileInputStream("e:\\tt.txt");

byte[] bs = new byte[100];

while(true){

int len = fin.read(bs);

if(len <= 0) break;

System.out.print(new String(bs,0,len));

}

fin.close();

//写文件:

FileWriter fw = new FileWriter("e:\\test.txt");

fw.write("hello world!" + System.getProperty("line.separator"));

fw.write("你好！北京！");

fw.close();

94、写一个方法,输入一个文件名和一个字符串,统计这个字符串在这个文件中出现的次数。【基础】

答：代码如下：

public int countWords(String file, String find) throws Exception

{

int count = 0;

Reader in = new FileReader(file);

int c;

while ((c = in.read()) != -1) {

while (c == find.charAt(0)) {

for (int i = 1; i < find.length(); i++) {

c = in.read();

if (c != find.charAt(i)) break;

if (i == find.length() - 1) count++;

}

}

}

return count;

}

**95、Java 的通信编程，编程题(或问答)，用JAVA SOCKET 编程，读服务器几个字符，再写入本地显示？【基础】**

答：Server 端程序:

package test;

import java.net.\*;

import java.io.\*;

public class Server{

private ServerSocket ss;

private Socket socket;

private BufferedReader in;

private PrintWriter out;

public Server(){

try {

ss=new ServerSocket(10000);

while(true){

socket = ss.accept();

String RemoteIP =

socket.getInetAddress().getHostAddress();

String RemotePort = ":"+socket.getLocalPort();

System.out.println("A client come in!IP:"

+ RemoteIP+RemotePort);

in = new BufferedReader(new

InputStreamReader(socket.getInputStream()));

String line = in.readLine();

System.out.println("Cleint send is :" + line);

out =

new PrintWriter(socket.getOutputStream(),true);

out.println("Your Message Received!");

out.close();

in.close();

socket.close();

}

}catch (IOException e){

out.println("wrong");

}

}

public static void main(String[] args){

new Server();

}

}

Client 端程序:

package test;

import java.io.\*;

import java.net.\*;

public class Client {

Socket socket;

BufferedReader in;

PrintWriter out;

public Client(){

try {

System.out.println("Try to Connect to

127.0.0.1:10000");

socket = new Socket("127.0.0.1",10000);

System.out.println("The Server Connected!");

System.out.println("Please enter some Character:");

BufferedReader line = new BufferedReader(new

InputStreamReader(System.in));

out = new PrintWriter(socket.getOutputStream(),true);

out.println(line.readLine());

in = new BufferedReader(

new InputStreamReader(socket.getInputStream()));

System.out.println(in.readLine());

out.close();

in.close();

socket.close();

}catch(IOException e){

out.println("Wrong");

}

}

public static void main(String[] args) {

new Client();

}

}

# 二、OOA/D 与UML 部分

**96、UML 是什么？常用的几种图？【基础】**

答：UML 是标准建模语言；常用图包括：用例图,静态图(包括类图、对象图和包图),行为图,交互图(顺序图,合作图),实现图。

**97、编程题: 写一个Singleton 出来。【基础】**

答：Singleton 模式主要作用是保证在Java 应用程序中，一个类Class 只有一个实例存在。举例：定义一个类，它的构造函数为private 的，它有一个static的private 的该类变量，在类初始化时实例话，通过一个public 的getInstance

方法获取对它的引用,继而调用其中的方法。

第一种形式:

public class Singleton {

private Singleton(){}

private static Singleton instance = new Singleton();

public static Singleton getInstance(){

return instance;

}

}

第二种形式:

public class Singleton {

private static Singleton instance = null;

public static synchronized Singleton getInstance(){

if (instance==null)

instance＝new Singleton();

return instance;

}

}

其他形式: 定义一个类，它的构造函数为private 的，所有方法为static 的。一般认为第一种形式要更加安全些。

**98、说说你所熟悉或听说过的j2ee 中的几种常用模式?及对设计模式的一些看法。【中等难度】**

答：Session Facade Pattern：使用SessionBean 访问EntityBean；

Message Facade Pattern：实现异步调用；

EJB Command Pattern：使用Command JavaBeans 取代SessionBean，实现轻量级访问；

Data Transfer Object Factory：通过DTO Factory 简化EntityBean 数据提供特性；

Generic Attribute Access：通过AttibuteAccess 接口简化EntityBean数据提供特性；

Business Interface：通过远程（本地）接口和Bean 类实现相同接口规范业务逻辑一致性；

EJB 架构的设计好坏将直接影响系统的性能、可扩展性、可维护性、组件可

重用性及开发效率。项目越复杂，项目队伍越庞大则越能体现良好设计的重要性。

**99、Java 中常用的设计模式？说明工厂模式？【中等难度】**

答：Java 中的23 种设计模式：Factory（ 工厂模式），Builder（ 建造模式）， Factory Method（工厂方法模式），Prototype（原始模型模式），Singleton（单例模式），Facade（门面模式），Adapter（适配器模式）， Bridge（桥梁模式）， Composite（合成模式），Decorator（装饰模式）， Flyweight（享元模式）， Proxy（代理模式），Command（命令模式）， Interpreter（解释器模式）， Visitor（访问者模式），Iterator（迭代子模式）， Mediator（调停者模式）， Memento（备忘录模式），Observer（观察者模式），State（状态模式），Strategy（策略模式），Template Method（模板方法模式）， Chain Of Responsibleity（责任链模式）。

工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

100、开发中都用到了那些设计模式?用在什么场合? 【中等难度】

答：每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需在重复相同的工作。主要用到了MVC 的设计模式，用来开发JSP/Servlet或者J2EE 的相关应用；及简单工厂模式等。

101、你对软件开发中迭代的含义的理解；【中等难度】

答：软件开发中，各个开发阶段不是顺序执行的，应该是并行执行,也就是迭代的意思。这样对于开发中的需求变化，及人员变动都能得到更好的适应。

# 三、XML 部分

102、XML 文档定义有几种形式？它们之间有何本质区别？解析XML 文档有哪几种方式？【基础】

答：1）两种形式：dtd 以及schema；

2）本质区别：schema 本身是xml 的，可以被XML 解析器解析(这也是从DTD上发展schema 的根本目的)；

3）解析方式：有DOM,SAX,STAX 等：

DOM:处理大型文件时其性能下降的非常厉害。这个问题是由DOM 的树结构所造成的，这种结构占用的内存较多，而且DOM 必须在解析文件之前把整个文档装入内存,适合对XML 的随机访问；SAX:不同于DOM,SAX 是事件驱动型的XML 解析方式。它顺序读取XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML 文件，适合对XML 的顺序访问；

STAX:Streaming API for XML (StAX)。

103、你在项目中用到了xml 技术的哪些方面?如何实现的? 【中等难度】

答:用到了数据存贮，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成XML 文件，然后将XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同XML 文件中还原相关信息进行处理。在做软件配置时，利用XML 可以很方便的进行，软件的各种配置参数都存贮在XML 文件中。

104、用jdom 解析xml 文件时如何解决中文问题?如何解析? 【较难】

答：看如下代码,用编码方式加以解决

package test;

import java.io.\*;

public class DOMTest{

private String inFile = "c:\people.xml";

private String outFile = "c:\people.xml";

public static void main(String args[]){

new DOMTest();

}

public DOMTest(){

try{

javax.xml.parsers.DocumentBuilder builder =

javax.xml.parsers.DocumentBuilderFactory.

newInstance().newDocumentBuilder();

org.w3c.dom.Document doc = builder.newDocument();

org.w3c.dom.Element root = doc.createElement("老师");

org.w3c.dom.Element wang = doc.createElement("王");

org.w3c.dom.Element liu = doc.createElement("刘");

wang.appendChild(doc.createTextNode("我是王老师"));

root.appendChild(wang);

doc.appendChild(root);

javax.xml.transform.Transformer transformer =

javax.xml.transform.TransformerFactory.

newInstance().newTransformer();

transformer.setOutputProperty(

javax.xml.transform.OutputKeys.ENCODING,"gb2312");

transformer.setOutputProperty(

javax.xml.transform.OutputKeys.INDENT, "yes");

transformer.transform(new

javax.xml.transform.dom.DOMSource(doc),

new javax.xml.transform.stream.StreamResult(outFile));

}catch (Exception e){

System.out.println (e.getMessage());

}

}

}

105、编程用JAVA 解析XML 的方式。【较难】

答：用SAX 方式解析XML，XML 文件如下：

<?xml version="1.0" encoding="gb2312"?>

<person>

<name>王小明</name>

<college>信息学院</college>

<telephone>6258113</telephone>

<notes>男,1955 年生,博士，95 年调入海南大学</notes>

</person>

事件回调类SAXHandler.java ：

import java.io.\*;

import java.util.Hashtable;

import org.xml.sax.\*;

public class SAXHandler extends HandlerBase{

private Hashtable table = new Hashtable();

private String currentElement = null;

private String currentValue = null;

public void setTable(Hashtable table){

this.table = table;

}

public Hashtable getTable(){

return table;

}

public void startElement(String tag, AttributeList attrs)

throws SAXException{

currentElement = tag;

}

public void characters(char[] ch, int start, int length)

throws SAXException{

currentValue = new String(ch, start, length);

}

public void endElement(String name) throws SAXException{

if (currentElement.equals(name))

table.put(currentElement, currentValue);

}

}

JSP 内容显示源码,SaxXml.jsp:

<HTML>

<HEAD>

<TITLE>剖析XML 文件people.xml</TITLE>

</HEAD>

<BODY>

<%@ page errorPage="ErrPage.jsp"

contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.\*" %>

<%@ page import="java.util.Hashtable" %>

<%@ page import="org.w3c.dom.\*" %>

<%@ page import="org.xml.sax.\*" %>

<%@ page import="javax.xml.parsers.SAXParserFactory" %>

<%@ page import="javax.xml.parsers.SAXParser" %>

<%@ page import="SAXHandler" %>

<%

File file = new File("c:\people.xml");

FileReader reader = new FileReader(file);

Parser parser;

SAXParserFactory spf = SAXParserFactory.newInstance();

SAXParser sp = spf.newSAXParser();

SAXHandler handler = new SAXHandler();

sp.parse(new InputSource(reader), handler);

Hashtable hashTable = handler.getTable();

out.println("<TABLE BORDER=2><CAPTION>" +

"教师信息表</CAPTION>");

out.println("<TR><TD>姓名</TD>" + "<TD>" +

(String)hashTable.get(new String("name")) +

"</TD></TR>");

out.println("<TR><TD>学院</TD>" + "<TD>" +

(String)hashTable.get(new String("college"))

+"</TD></TR>");

out.println("<TR><TD>电话</TD>" + "<TD>" +

(String)hashTable.get(new String("telephone"))

+ "</TD></TR>");

out.println("<TR><TD>备注</TD>" + "<TD>" +

(String)hashTable.get(new String("notes"))

+ "</TD></TR>");

out.println("</TABLE>");

%>

</BODY>

</HTML>

# 数据库及SQL 部分

## Sql语句

106、有3 个表（15 分钟）：【基础】

Student 学生表(学号，姓名，性别，年龄，组织部门)

Course 课程表(编号，课程名称)

Sc 选课表(学号，课程编号，成绩)

表结构如下：

1）写一个SQL 语句，查询选修了’计算机原理’的学生学号和姓名（3 分钟）

2）写一个SQL 语句，查询’周星驰’同学选修了的课程名字（3 分钟）

3）写一个SQL 语句，查询选修了5 门课程的学生学号和姓名（9 分钟）

答：1）SQL 语句如下：

select stu.sno, stu.sname from Student stu

where (select count(\*) from sc where sno=stu.sno and cno =

(select cno from Course where cname='计算机原理')) != 0;

2）SQL 语句如下：

select cname from Course

where cno in ( select cno from sc where sno =

(select sno from Student where sname='周星驰'));

3）SQL 语句如下：

select stu.sno, stu.sname from student stu

where (select count(\*) from sc where sno=stu.sno) = 5;

107、有三张表,学生表S,课程C,学生课程表SC,学生可以选修多门课程,一门课程可以被多个学生选修,通过SC 表关联。【基础】

1）写出建表语句；

2）写出SQL 语句,查询选修了所有选修课程的学生；

3）写出SQL 语句,查询选修了至少5 门以上的课程的学生。

答：1）建表语句如下（mysql 数据库）：

create table s(id integer primary key, name varchar(20));

create table c(id integer primary key, name varchar(20));

create table sc(

sid integer references s(id),

cid integer references c(id),

primary key(sid,cid)

);

2）SQL 语句如下：

select stu.id, stu.name from s stu

where (select count(\*) from sc where sid=stu.id)

= (select count(\*) from c);

3）SQL 语句如下：

select stu.id, stu.name from s stu

where (select count(\*) from sc where sid=stu.id)>=5;

108、数据库表(Test)结构如下：【基础】

ID NAME AGE MANAGER(所属主管人ID)

106 A 30 104

109 B 19 104

104 C 20 111

107 D 35 109

112 E 25 120

119 F 45 NULL

要求:列出所有年龄比所属主管年龄大的人的ID 和名字?

答：SQL 语句如下：

select employee.name from test employee

where employee.age > (select manager.age from test manager

where manager.id=employee.manager);

109、有如下两张表：【中等难度】

表city： 表state：

CityNo CityName StateNo

BJ 北京（Null）

SH 上海（Null）

GZ 广州GD

DL 大连LN

欲得

到如下结果： City

No City Name State No State Name BJ

北京（Null） （Null） DL

大连LN 辽宁GZ

广州GD 广东SH

上海（Null） （Null） 写相

应的SQL 语句。

答：SQL 语句为：

SELECT C.CITYNO, C.CITYNAME, C.STATENO, S.STATENAME

FROM CITY C, STATE S

WHERE C.STATENO=S.STATENO(+)

ORDER BY(C.CITYNO);

## 存储过程和函数的区别

存储过程是用户定义的一系列sql语句的集合，涉及特定表或其它对象的任务，用户可以调用存储过程，而函数通常是数据库已定义的方法，它接收参数并返回某种类型的值并且不涉及特定用户表。

## 事务是什么？

事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务：

原子性事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。

一致性事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。

隔离性由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

持久性事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

## 游标的作用？如何知道游标已经到了最后？

游标用于定位结果集的行，通过判断全局变量@@FETCH\_STATUS可以判断是否到了最后，通常此变量不等于0表示出错或到了最后。

## 触发器

触发器分为事前触发和事后触发，这两种触发有和区别。语句级触发和行级触发有何区别。

事前触发器运行于触发事件发生之前，而事后触发器运行于触发事件发生之后。通常事前触发器可以获取事件之前和新的字段值。

语句级触发器可以在语句执行前或后执行，而行级触发在触发器所影响的每一行触发一次。

## 连接池的基本工作原理

### 1、基本概念及原理

由上面的分析可以看出，问题的根源就在于对数据库连接资源的低效管理。我们知道，对于共享资源，有一个很著名的设计模式：资源池（Resource Pool）。该模式正是为了解决资源的频繁分配?释放所造成的问题。为解决上述问题，可以采用数据库连接池技术。数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。我们可以通过设定连接池最大连接数来防止系统无尽的与数据库连接。更为重要的是我们可以通过连接池的管理机制监视数据库的连接的数量?使用情况，为系统开发?测试及性能调整提供依据

### 2、服务器自带的连接池

JDBC的API中没有提供连接池的方法。一些大型的WEB应用服务器如BEA的WebLogic和IBM的WebSphere等提供了连接池的机制，但是必须有其第三方的专用类方法支持连接池的用法。

## 连接池关键问题分析

### 1、并发问题

为了使连接管理服务具有最大的通用性，必须考虑多线程环境，即并发问题。这个问题相对比较好解决，因为Java语言自身提供了对并发管理的支持，使用synchronized关键字即可确保线程是同步的。使用方法为直接在类方法前面加上synchronized关键字，如：

public synchronized Connection getConnection（）

### 2、多数据库服务器和多用户

对于大型的企业级应用，常常需要同时连接不同的数据库（如连接Oracle和Sybase）。如何连接不同的数据库呢？我们采用的策略是：设计一个符合单例模式的连接池管理类，在连接池管理类的唯一实例被创建时读取一个资源文件，其中资源文件中存放着多个数据库的url地址（）?用户名（）?密码（）等信息。如tx.url=172.21.15.123：5000/tx\_it，tx.user=yang，tx.password=yang321。根据资源文件提供的信息，创建多个连接池类的实例，每一个实例都是一个特定数据库的连接池。连接池管理类实例为每个连接池实例取一个名字，通过不同的名字来管理不同的连接池。   
  
对于同一个数据库有多个用户使用不同的名称和密码访问的情况，也可以通过资源文件处理，即在资源文件中设置多个具有相同url地址，但具有不同用户名和密码的数据库连接信息

### 3、事务处理

我们知道，事务具有原子性，此时要求对数据库的操作符合“ALL-ALL-NOTHING”原则,即对于一组SQL语句要么全做，要么全不做。   
  
在Java语言中，Connection类本身提供了对事务的支持，可以通过设置Connection的AutoCommit属性为false,然后显式的调用commit或rollback方法来实现。但要高效的进行Connection复用，就必须提供相应的事务支持机制。可采用每一个事务独占一个连接来实现，这种方法可以大大降低事务管理的复杂性。

### 连接池的分配与释放

连接池的分配与释放，对系统的性能有很大的影响。合理的分配与释放，可以提高连接的复用度，从而降低建立新连接的开销，同时还可以加快用户的访问速度。   
  
对于连接的管理可使用空闲池。即把已经创建但尚未分配出去的连接按创建时间存放到一个空闲池中。每当用户请求一个连接时，系统首先检查空闲池内有没有空闲连接。如果有就把建立时间最长（通过容器的顺序存放实现）的那个连接分配给他（实际是先做连接是否有效的判断，如果可用就分配给用户，如不可用就把这个连接从空闲池删掉，重新检测空闲池是否还有连接）；如果没有则检查当前所开连接池是否达到连接池所允许的最大连接数（maxConn）,如果没有达到，就新建一个连接，如果已经达到，就等待一定的时间（timeout）。如果在等待的时间内有连接被释放出来就可以把这个连接分配给等待的用户，如果等待时间超过预定时间timeout,则返回空值（null）。系统对已经分配出去正在使用的连接只做计数，当使用完后再返还给空闲池。对于空闲连接的状态，可开辟专门的线程定时检测，这样会花费一定的系统开销，但可以保证较快的响应速度。也可采取不开辟专门线程，只是在分配前检测的方法。

### 连接池的配置与维护

连接池中到底应该放置多少连接，才能使系统的性能最佳？系统可采取设置最小连接数（minConn）和最大连接数（maxConn）来控制连接池中的连接。最小连接数是系统启动时连接池所创建的连接数。如果创建过多，则系统启动就慢，但创建后系统的响应速度会很快；如果创建过少，则系统启动的很快，响应起来却慢。这样，可以在开发时，设置较小的最小连接数，开发起来会快，而在系统实际使用时设置较大的，因为这样对访问客户来说速度会快些。最大连接数是连接池中允许连接的最大数目，具体设置多少，要看系统的访问量，可通过反复测试，找到最佳点。   
  
如何确保连接池中的最小连接数呢？有动态和静态两种策略。动态即每隔一定时间就对连接池进行检测，如果发现连接数量小于最小连接数，则补充相应数量的新连接,以保证连接池的正常运转。静态是发现空闲连接不够时再去检查。

## 连接池的实现

### 连接池模型

本文讨论的连接池包括一个连接池类（DBConnectionPool）和一个连接池管理类（DBConnetionPoolManager）。连接池类是对某一数据库所有连接的“缓冲池”，主要实现以下功能：①从连接池获取或创建可用连接；②使用完毕之后，把连接返还给连接池；③在系统关闭前，断开所有连接并释放连接占用的系统资源；④还能够处理无效连接（原来登记为可用的连接，由于某种原因不再可用，如超时，通讯问题），并能够限制连接池中的连接总数不低于某个预定值和不超过某个预定值。   
  
连接池管理类是连接池类的外覆类（wrapper）,符合单例模式，即系统中只能有一个连接池管理类的实例。其主要用于对多个连接池对象的管理，具有以下功能：①装载并注册特定数据库的JDBC驱动程序；②根据属性文件给定的信息，创建连接池对象；③为方便管理多个连接池对象，为每一个连接池对象取一个名字，实现连接池名字与其实例之间的映射；④跟踪客户使用连接情况，以便需要是关闭连接释放资源。连接池管理类的引入主要是为了方便对多个连接池的使用和管理，如系统需要连接不同的数据库，或连接相同的数据库但由于安全性问题，需要不同的用户使用不同的名称和密码。

### 连接池实现

下面给出连接池类和连接池管理类的主要属性及所要实现的基本接口：

**代码:**

public class DBConnectionPool implements TimerListener{   
private int checkedOut;//已被分配出去的连接数   
private ArrayList freeConnections=new ArrayList();  
//容器，空闲池，根据//创建时间顺序存放已创建但尚未分配出去的连接   
private int minConn;//连接池里连接的最小数量   
private int maxConn;//连接池里允许存在的最大连接数   
private String name;//为这个连接池取个名字，方便管理   
private String password;//连接数据库时需要的密码   
private String url;//所要创建连接的数据库的地址   
private String user;//连接数据库时需要的用户名   
public Timer timer;//定时器   
public DBConnectionPool(String name,String URL,String user,  
String password,int maxConn)//公开的构造函数   
public synchronized void freeConnection(Connection con)   
//使用完毕之后，把连接返还给空闲池   
public synchronized Connection getConnection(long timeout)  
//得到一个连接，timeout是等待时间   
public synchronized void release()  
//断开所有连接，释放占用的系统资源   
private Connection newConnection()  
//新建一个数据库连接   
public synchronized void TimerEvent()   
//定时器事件处理函数   
}   
public class DBConnectionManager {   
static private DBConnectionManager instance;  
//连接池管理类的唯一实例   
static private int clients;//客户数量   
private ArrayList drivers=new ArrayList();  
//容器，存放数据库驱动程序   
private HashMap pools = new HashMap();  
//以name/value的形式存取连接池对象的名字及连接池对象   
static synchronized public DBConnectionManager getInstance()  
/\*\*如果唯一的实例instance已经创建，直接返回这个实例;否则，调用私有构造函数，  
创建连接池管理类的唯一实例\*/   
private DBConnectionManager()  
//私有构造函数,在其中调用初始化函数init()   
public void freeConnection(String name,Connection con)  
//释放一个连接，name是一个连接池对象的名字   
public Connection getConnection(String name)  
//从名字为name的连接池对象中得到一个连接   
public Connection getConnection(String name,long time)  
//从名字为name的连接池对象中取得一个连接，time是等待时间   
public synchronized void release()//释放所有资源   
private void createPools(Properties props)  
//根据属性文件提供的信息，创建一个或多个连接池   
private void init()//初始化连接池管理类的唯一实例，由私有构造函数调用   
private void loadDrivers(Properties props)//装载数据库驱动程序   
}

### 连接池使用

上面所实现的连接池在程序开发时如何应用到系统中呢？下面以Servlet为例说明连接池的使用。   
  
Servlet的生命周期是：在开始建立servlet时，调用其初始化（init）方法。之后每个用户请求都导致一个调用前面建立的实例的service方法的线程。最后，当服务器决定卸载一个servlet时，它首先调用该servlet的destroy方法。   
  
　　根据servlet的特点，我们可以在初始化函数中生成连接池管理类的唯一实例（其中包括创建一个或多个连接池）。如：

**代码:**

public void init() throws ServletException   
{   
　connMgr=DBConnectionManager.getInstance();   
}

然后就可以在service方法中通过连接池名称使用连接池，执行数据库操作。最后在destroy方法中释放占用的系统资源，如：

**代码:**

public void destroy(){   
connMgr.release();   
super.destroy();

在使用JDBC进行与数据库有关的应用开发中，数据库连接的管理是一个难点。很多时候，连接的混乱管理所造成的系统资源开销过大成为制约大型企业级应用效率的瓶颈。对于众多用户访问的Web应用，采用数据库连接技术的系统在效率和稳定性上比采用传统的其他方式的系统要好很多。本文阐述了使用JDBC访问数据库的技术?讨论了基于连接池技术的数据库连接管理的关键问题并给出了一个实现模型。文章所给出的是连接池管理程序的一种基本模式，为提高系统的整体性能，在此基础上还可以进行很多有意义的扩展。

# 五、Oracle

## 1.解释冷备份和热备份的不同点以及各自的优点

　　解答:热备份针对归档模式的数据库，在数据库仍旧处于工作状态时进行备份。而冷

　　备份指在数据库关闭后，进行备份，适用于所有模式的数据库。热备份的优点在于当备份

　　时，数据库仍旧可以被使用并且可以将数据库恢复到任意一个时间点。冷备份的优点在于

　　它的备份和恢复操作相当简单，并且由于冷备份的数据库可以工作在非归档模式下,数据库

　　性能会比归档模式稍好。(因为不必将archive log写入硬盘)

## 2.你必须利用备份恢复数据库，但是你没有控制文件，该如何解决问题呢?

　　解答:重建控制文件，用带backup control file 子句的recover 命令恢复数据库。

## 3.如何转换init.ora到spfile?

　　解答:使用create spfile from pfile 命令.

　　4.解释data block , extent 和 segment的区别(这里建议用英文术语)

　　解答:data block是数据库中最小的逻辑存储单元。当数据库的对象需要更多的物理

　　存储空间时，连续的data block就组成了extent . 一个数据库对象拥有的所有extents被

　　称为该对象的segment.

## 5.给出两个检查表结构的方法

　　解答:1.DESCRIBE命令

　　2.DBMS\_METADATA.GET\_DDL 包

## 6.怎样查看数据库引擎的报错

　　解答:alert log.

## 7.比较truncate和delete 命令

　　解答:两者都可以用来删除表中所有的记录。区别在于:truncate是DDL操作，它移动

　　HWK，不需要rollback segment .而Delete是DML操作, 需要rollback segment 且花费较长

　　时间.

## 8.使用索引的理由

　　解答:快速访问表中的data block

## 9.给出在STAR SCHEMA中的两种表及它们分别含有的数据

　　解答:Fact tables 和dimension tables. fact table包含大量的主要的信息而dime

　　nsion tables 存放对fact table 某些属性描述的信息

## 10.FACT Table上需要建立何种索引?

　　解答:位图索引 (bitmap index)

## 11. 给出两种相关约束?

　　解答:主键和外键

## 12. 如何在不影响子表的前提下，重建一个母表

　　解答:子表的外键强制实效，重建母表，激活外键

## 13. 解释归档和非归档模式之间的不同和它们各自的优缺点

　　解答:归档模式是指你可以备份所有的数据库 transactions并恢复到任意一个时间点

　　。非归档模式则相反，不能恢复到任意一个时间点。但是非归档模式可以带来数据库性能

　　上的少许提高.

## 14. 如何建立一个备份控制文件?

　　解答:Alter database backup control file to trace.

## 15. 给出数据库正常启动所经历的几种状态 ?

　　解答:STARTUP NOMOUNT ?C 数据库实例启动

　　STARTUP MOUNT - 数据库装载

　　STARTUP OPEN ?C 数据库打开

## 16. 哪个column可以用来区别V$视图和GV$视图?

　　解答:INST\_ID 指明集群环境中具体的 某个instance 。

## 17. 如何生成explain plan?

　　解答:运行utlxplan.sql. 建立plan 表

　　针对特定SQL语句，使用 explain plan set statement\_id = 'tst1' into pl

　　an\_table

　　运行utlxplp.sql 或 utlxpls.sql察看explain plan

## 18. 如何增加buffer cache的命中率?

　　解答:在数据库较繁忙时，适用buffer cache advisory 工具，查询v$db\_cache\_adv

　　ice.如果有必要更改，可以使用 alter system set db\_cache\_size 命令

## 19. ORA-01555的应对方法?

　　解答:具体的出错信息是snapshot too old within rollback seg , 通常可以通过增

　　大rollback seg来解决问题。当然也需要察看一下具体造成错误的SQL文本

## 20. 解释$ORACLE\_HOME和$ORACLE\_BASE的区别?

解答:ORACLE\_BASE是oracle的根目录，ORACLE\_HOME是oracle产品的目录。

## 21. 如何判断数据库的时区？

解答：SELECT DBTIMEZONE FROM DUAL;

## 22. 解释GLOBAL\_NAMES设为TRUE的用途

解答：GLOBAL\_NAMES指明联接数据库的方式。如果这个参数设置为TRUE,

在建立数据库链接时就必须用相同的名字连结远程数据库

## 23。如何加密PL/SQL程序？

解答：WRAP

## 24. 解释FUNCTION,PROCEDURE和PACKAGE区别

解答：function 和procedure是PL/SQL代码的集合，通常为了完成

一个任务。procedure 不需要返回任何值而function将返回一个值

在另一方面，Package是为了完成一个商业功能的一组function和proceudre

的集合

## 25. 解释TABLE Function的用途

解答：TABLE Function是通过PL/SQL逻辑返回一组纪录，用于

普通的表/视图。他们也用于pipeline和ETL过程。

## 26. 举出3种可以收集three advisory statistics

解答：Buffer Cache Advice, Segment Level Statistics, Timed Statistics

## 27. Audit trace 存放在哪个oracle目录结构中?

解答：unix $ORACLE\_HOME/rdbms/audit

Windows the event viewer

## 28. 解释materialized views的作用

解答：Materialized views 用于减少那些汇总，集合和分组的

信息的集合数量。它们通常适合于数据仓库和DSS系统。

## 29. 当用户进程出错，哪个后台进程负责清理它

解答： PMON

## 30. 哪个后台进程刷新materialized views?

解答：The Job Queue Processes.

## 31. 如何判断哪个session正在连结以及它们等待的资源？

解答：V$SESSION / V$SESSION\_WAIT

## 32. 描述什么是 redo logs

解答：Redo Logs 是用于存放数据库数据改动状况的物理和逻辑结构。

可以用来修复数据库.

## 33. 如何进行强制LOG SWITCH?

解答：ALTER SYSTEM SWITCH LOGFILE;

## 34. 举出两个判断DDL改动的方法？

解答：你可以使用 Logminer 或 Streams

## 35. Coalescing做了什么？

解答：Coalescing针对于字典管理的tablespace进行碎片整理，将

临近的小extents合并成单个的大extent.

## 36. TEMPORARY tablespace和PERMANENT tablespace 的区别是？

解答：A temporary tablespace 用于临时对象例如排序结构而 permanent tablespaces

用来存储那些'真实'的对象(例如表，回滚段等）

## 37. 创建数据库时自动建立的tablespace名称？

解答：SYSTEM tablespace.

## 38. 创建用户时，需要赋予新用户什么权限才能使它联上数据库。

解答：CONNECT

## 39. 如何在tablespace里增加数据文件？

解答：ALTER TABLESPACE <tablespace\_name> ADD DATAFILE <datafile\_name> SIZE <size>

## 40. 如何变动数据文件的大小？

解答：ALTER DATABASE DATAFILE <datafile\_name> RESIZE <new\_size>;

## 41. 哪个VIEW用来检查数据文件的大小？

解答： DBA\_DATA\_FILES

## 42. 哪个VIEW用来判断tablespace的剩余空间

解答：DBA\_FREE\_SPACE

## 43. 如何判断谁往表里增加了一条纪录？

解答：auditing

## 44. 如何重构索引？

解答： ALTER INDEX <index\_name> REBUILD;

## 45. 解释什么是Partitioning（分区） 以及它的优点。

解答：Partition将大表和索引分割成更小，易于管理的分区。

## 46. 你刚刚编译了一个PL/SQL Package但是有错误报道，如何显示出错信息？

解答：SHOW ERRORS

## 47. 如何搜集表的各种状态数据？

解答： ANALYZE

The ANALYZE command.

## 48. 如何启动SESSION级别的TRACE

解答: DBMS\_SESSION.SET\_SQL\_TRACE

ALTER SESSION SET SQL\_TRACE = TRUE;

## 50。 用于网络连接的2个文件？

解答： TNSNAMES.ORA and SQLNET.ORA

**【问题】**

有下列两张表：

USERLIST表：

|  |  |  |
| --- | --- | --- |
| TELEPHONE  (Varchar(10),主键) | ACCOUNT  (varchar(10)) | RENT  (numeric(10,2)) |
| 4210001 | AAAA | 19.50 |
| 4210002 | AAAA | 20.50 |
| 4210003 | BBBB | 100.00 |
| 4210004 | CCCC | 250.00 |

CHARGE表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TELEPHONE  (Varchar(10),主键) | FEE01  (numeric(10,2)) | FEE02  (numeric(10,2)) | FEE03  (numeric(10,2)) | FEE04  (numeric(10,2)) |
| 4210001 | 11.00 | 12.00 | 13.00 | 14.00 |
| 4210002 | 21.00 | 22.00 | 23.00 | 24.00 |
| 4210003 | 31.00 | 32.00 | 33.00 | 34.00 |

请用最少的SQL语句，产生下述查询结果：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ACCOUNT  (Varchar(10),主键) | USERS  (int) | RENT  (numeric(10,2)) | FEE01  (numeric(10,2)) | FEE02  (numeric(10,2)) | FEE03  (numeric(10,2)) | FEE04  (numeric(10,2)) |
| AAAA | 2 | 40.00 | 32.00 | 34.00 | 36.00 | 38.00 |
| BBBB | 1 | 100.00 | 31.00 | 32.00 | 33.00 | 34.00 |
| CCCC | 1 | 250.00 | 0.00 | 0.00 | 0.00 | 0.00 |

其中：

数据是经过USERLIST、CHARGE表进行合适的连接后以ACCOUNT为关键字分组求和得到。

**【特别注意】**电话号码4210004 在USERLIST表中有一条记录，但是在CHARGE表中并没有记录。但结果中合同号CCCC具有一条记录。

请写出SQL语句。

SELECT A.ACCOUNT,SUM(ACCOUNT) AS USERS,A.RENT,B.FEE01,B.FEE02,B.FEE03,B.FEE04

FROM USERLIST A,CHARGE B

WHERE B.TELEPHONE= (+)A.TELEPHONE ORDER BY ACCOUNT GROUP BY ACCOUNT

## Oracle问卷（Oracle程序员应答，其他跳过）

### 【问题】

解释一下sum的作用

答：求符合条件的记录某数值字段的和

### 【问题】

Table1有两个字段，field1 varchar2(10)和field2 number。

请写出向这两个表中插入’test’和100的sql语句。

答：insert into Table1 (field1,field2) values(‘test’,100);

请写出一个sql语句，将Table1中的field1字段的值更新成’test111’：

答：update Table1 set field1=’test111’; –此为将Table1表中所有记录field1字段的值改为’test111’

update Table1 set field1=’test111’ where field2=100; –此为将Table1表中field2字段等于100的记录的field1字段的值改为’test111’

请写出一个sql语句，将Table1中的数据全部选出来：

答: select field1,field2 from Table1;

### 【问题】

Delete和truncate的区别是什么？

答: Delete删除的记录如不提交可以恢复,此命令不能删除回滚段中的数据

truncate删除的记录不能恢复,并可删除回滚段中的数据

### 【问题】

请解释一下varchar2和char的区别

答: char 最大可以容纳2000个字节

varchar2最大可以容纳4000个字节,此为可变长字段

### 【问题】

解释一下group by的作用

答: 此关键字的作用是将查询出的符合条件的记录集按某字段列表分组,在此过程中预查出的字段若为字符型,分组后只取最上面的一个,预为数值型将求和

### 【问题】

Order by有什么作用

答: 此关键字的作用是将查询出符合条件的记录集按某字段列表进行排序,默认为升序(Asc),降序为: Desc

### 【问题】

Union和union all有什么区别

答:

### 【问题】

Months\_between的用法是什么

答: 是在一个月份的范围内

### 【问题】

Oracle体系结构类：

分区表有什么作用？

答:可以将表中不同的字段或不同的记录存入不同的表空间,并且这些表空间可在不同的物理磁盘,

一般用于将比较大的字段如图片或文件类字段存入比较大的磁盘中.

请解释一下oracle表空间

答: 表空间是一个逻辑概念,但属于表空间的数据文件是存在于磁盘上的物理文件用户的数据存在于表中而表就存在于表空间中,一个表空间可以有多个属于自己的数据文件,但一个数据文件只能属于一个表空间

Oracle中表空间和数据文件有什么关系

答: 一个表空间可以有多个属于自己的数据文件,但一个数据文件只能属于一个表空间

请简单描述一下Oracle的体系结构

答: 主要有两大部分

数据库实例,包括:系统全局区(SGA),和后台进程(数据写入进程,重做日志进程,检查点进程,用户监控进程等),实例在启动时首先到初始化参数文件中找到控制文件,再从控制文件中找到日志文件和数据文件

数据库文件,包括:数据文件(.dbf)日志文件(.log)控制文件(.ctl)

其中数据文件又分 系统数据文件 用户数据文件 临时数据文件和回滚数据文件

日志文件可以有多组,一组写满换另一组,在切换时有检查点进程保证数据的完成性

控制文件是二进制文件,最大可以为64M

# 六、JDBC 及Hibernate

110、数据库，比如100 用户同时来访，要采取什么技术解决？【基础】

答：可采用连接池。

111、什么是ORM？【基础】

答：对象关系映射（Object—Relational Mapping，简称ORM）是一种为了解决面向对象与面向关系数据库存在的互不匹配的现象的技术；简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据，将java 程序中的对象自动持久化到关系数据库中；本质上就是将数据从一种形式转换到另外一种形式。

112、Hibernate 有哪5 个核心接口？【基础】

答：Configuration 接口：配置Hibernate，根据其启动hibernate，创建SessionFactory 对象；SessionFactory 接口：初始化Hibernate，充当数据存储源的代理，创建session 对象，sessionFactory 是线程安全的，意味着它的同一个实例可以被应用的多个线程共享，是重量级、二级缓存；Session 接口：负责保存、更新、删除、加载和查询对象，是线程不安全的，避免多个线程共享同一个session，是轻量级、一级缓存；Transaction 接口：管理事务；Query 和Criteria 接口：执行数据库的查询。

113、关于hibernate: 【基础】

1）在hibernate 中，在配置文件呈标题一对多，多对多的标签是什么；2）Hibernate 的二级缓存是什么；3）Hibernate 是如何处理事务的；

答：1）一对多的标签为<one-to-many> ；多对多的标签为<many-to-many>；

2）sessionFactory 的缓存为hibernate 的二级缓存；

3）Hibernate 的事务实际上是底层的JDBC Transaction 的封装或者是JTA

Transaction 的封装；默认情况下使用JDBCTransaction。

State

No

State

Name

GD 广东

LN 辽宁

SD 山东

NMG 内蒙古

114、Hibernate 的应用（Hibernate 的结构）？【基础】

答：//首先获得SessionFactory 的对象

SessionFactory sessionFactory = new Configuration().configure().

buildSessionFactory();

//然后获得session 的对象

Session session = sessionFactory.openSession();

//其次获得Transaction 的对象

Transaction tx = session.beginTransaction();

//执行相关的数据库操作:增,删,改,查

session.save(user); //增加, user 是User 类的对象

session.delete(user); //删除

session.update(user); //更新

Query query = session.createQuery(“from User”); //查询

List list = query.list();

//提交事务

tx.commit();

//如果有异常,我们还要作事务的回滚,恢复到操作之前

tx.rollback();

//最后还要关闭session,释放资源

session.close();

115、什么是重量级？什么是轻量级？【基础】

答：轻量级是指它的创建和销毁不需要消耗太多的资源，意味着可以在程序中经常创建和销毁session 的对象；重量级意味不能随意的创建和销毁它的实例，会占用很多的资源。

116、数据库的连接字符串？【基础】

答：MS SQL Server

//第二种连接方式

Class.forName(“com.microsoft.jdbc.sqlserver.SQLServerDriver”).

newInstance();

conn = DriverManager.getConnection(“jdbc:Microsoft:sqlserver

://localhost:1433;DatabaseName=pubs”,”sa”,””);

//Oracle

Class.forName(“oracle.jdbc.driver.OracleDriver”).newInstance();

conn = DriverManager.getConnection(“jdbc:oracle:thin:

@localhost:1521:sid”, uid, pwd);

//Mysql

Class.forName(“org.git.mm.mysql.Driver”).newInstance();

conn = DriverManager.getConnection(“jdbc:mysql

://localhost:3306/pubs”,”root”,””);

处理中文的问题:

jdbc:mysql://localhost:3306/pubs?useUnicode=true

&characterEncoding=GB2312

117、事务处理？【基础】

答：Connection 类中提供了3 个事务处理方法：setAutoCommit(Boolean autoCommit):设置是否自动提交事务，默认为自动提交事务，即为true，通过设置false 禁止自动提交事务；commit():提交事务；rollback():回滚事务。

118、Java 中访问数据库的步骤？Statement 和PreparedStatement 之间的区别？

【基础】

答：Java 中访问数据库的步骤如下：

1）注册驱动；

2）建立连接；

3）创建Statement；

4）执行sql 语句；

5）处理结果集（若sql 语句为查询语句）；

6）关闭连接。

PreparedStatement 被创建时即指定了SQL 语句，通常用于执行多次结构相同的SQL 语句。

119、用你熟悉的语言写一个连接ORACLE 数据库的程序,能够完成修改和查询工作。【基础】

答：JDBC 示例程序如下：

public void testJdbc(){

Connection con = null;

PreparedStatement ps = null;

ResultSet rs = null;

try{

//step1：注册驱动；

Class.forName("oracle.jdbc.driver.OracleDriver");

//step 2：获取数据库连接；

con=DriverManager.getConnection(

"jdbc:oracle:thin:@192.168.0.39:1521:TARENADB",

"sd0605","sd0605");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*查询\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//step 3：创建Statement；

String sql = "SELECT id, fname, lname, age, FROM

Person\_Tbl";

ps = con.prepareStatement(sql);

//step 4 ：执行查询语句，获取结果集；

rs = ps.executeQuery();

//step 5：处理结果集—输出结果集中保存的查询结果；

while (rs.next()){

System.out.print("id = " + rs.getLong("id"));

System.out.print(" , fname = " +

第35 页共59 页

rs.getString("fname"));

System.out.print(" , lname = " +

rs.getString("lname"));

System.out.print(" , age = " + rs.getInt("age"));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JDBC 修改\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sql = "UPDATE Person\_Tbl SET age=23 WHERE id = ?";

ps = con.prepareStatement(sql);

ps.setLong(1, 88);

int rows = ps.executeUpdate();

System.out.println(rows + " rows affected.");

} catch (Exception e){

e.printStackTrace();

} finally{

try{

con.close(); //关闭数据库连接，以释放资源。

} catch (Exception e1) {

}

}

}

120、JDBC，Hibernate 分页怎样实现？【中等难度】

答：方法分别为：

1) Hibernate 的分页：

Query query = session.createQuery("from Student");

query.setFirstResult(firstResult);//设置每页开始的记录号

query.setMaxResults(resultNumber);//设置每页显示的记录数

Collection students = query.list();

2) JDBC 的分页：根据不同的数据库采用不同的sql 分页语句

例如: Oracle 中的sql 语句为: "SELECT \* FROM (SELECT a.\*, rownum r FROM

TB\_STUDENT) WHERE r between 2 and 10" 查询从记录号2 到记录号10 之间的所有记录

121、在ORACLE 大数据量下的分页解决方法。一般用截取ID 方法，还有是三层嵌套方法。【中等难度】

答:一种分页方法

<%

int i=1;

int numPages=14;

String pages = request.getParameter("page") ;

int currentPage = 1;

currentPage = (pages==null)?(1):{Integer.parseInt(pages)}

sql = "select count(\*) from tables";

ResultSet rs = DBLink.executeQuery(sql) ;

while(rs.next()) i = rs.getInt(1) ;

int intPageCount=1;

intPageCount=(i%numPages==0)?(i/numPages):(i/numPages+1);

int nextPage ;

int upPage;

nextPage = currentPage+1;

if (nextPage>=intPageCount) nextPage=intPageCount;

upPage = currentPage-1;

if (upPage<=1) upPage=1;

rs.close();

sql="select \* from tables";

rs=DBLink.executeQuery(sql);

i=0;

while((i<numPages\*(currentPage-1))&&rs.next()){i++;}

%>

//输出内容

//输出翻页连接

合计:<%=currentPage%>/<%=intPageCount%>页

<a href="List.jsp?page=1">第一页</a>

<a href="List.jsp?page=<%=upPage%>">上一页</a>

<%

for(int j=1;j<=intPageCount;j++){

if(currentPage!=j){

%>

<a href="list.jsp?page=<%=j%>">[<%=j%>]</a>

<%

}else{

out.println(j);

}

}

%>

<a href="List.jsp?page=<%=nextPage%>">下一页</a>

<a href="List.jsp?page=<%=intPageCount%>">最后页</a>

# 七、数据结构、算法及计算机基础部分：

180、一个byte 几个单位？【基础】

答：8bit。

181、常用UNIX 命令(Linux 的常用命令)（至少10 个）【基础】

答：ls pwd mkdir rm cp mv cd ps ftp telnet ping env more echo

182、后序遍历下列二叉树，访问结点的顺序是？【基础】

A

/ \

B C

/ \ \

D E F

/ / \

G N I

/ \

J K

答：顺序为：DJGEBKNIFCA 。

183、排序都有哪几种方法？请列举。用JAVA 实现一个快速排序。【基础】

答：排序的方法有：插入排序（直接插入排序、希尔排序），交换排序（冒泡排序、快速排序），选择排序（直接选择排序、堆排序），归并排序，分配排序（箱排序、基数排序）；快速排序的伪代码：

//使用快速排序方法对a[ 0 :n- 1 ]排序

从a[ 0 :n- 1 ]中选择一个元素作为middle，该元素为支点；

把余下的元素分割为两段left 和right，使得left 中的元素都小于等于支点，

而right 中的元素都大于等于支点；

递归地使用快速排序方法对left 进行排序；

递归地使用快速排序方法对right 进行排序；

所得结果为left + middle + right。

184、写一种常见排序。【基础】

答：C++中冒泡排序：

void swap( int& a, int& b ){

int c=a; a = b; b = c;

}

void bubble( int\* p, int len ){

bool bSwapped;

do {

bSwapped = false;

for( int i=1; i<len; i++ ){

if( p[i-1]>p[i] ){

swap( p[i-1], p[i] );

bSwapped = true;

}

}

}while( bSwapped );

}

185、写一个一小段程序检查数字是否为质数；以上的程序你采用的哪种语言写的？采用该种语言的理由是什么？【基础】

答：代码如下：

#include <math.h>

bool prime( int n ){

if(n<=0) exit(0);

for( int i=2; i<=n; i++ )

for( int j=2; j<=sqrt(i); j++)

if((n%j==0) && (j!=n))

return false;

return true;

}

采用C++，因为其运行效率高。

186、编程题：设有ｎ个人依围成一圈，从第１个人开始报数，数到第ｍ个人出列，然后从出列的下一个人开始报数，数到第ｍ个人又出列，…，如此反复到所有的人全部出列为止。设ｎ个人的编号分别为1，2，…，n，打印出出列的顺序；要求用java 实现。【中等难度】

答：代码如下：

package test;

public class CountGame {

private static boolean same(int[] p,int l,int n){

for(int i=0;i<l;i++){

if(p[i]==n){

return true;

}

}

return false;

}

public static void play(int playerNum, int step){

int[] p=new int[playerNum];

int counter = 1;

while(true){

if(counter > playerNum\*step){

break;

}

for(int i=1;i<playerNum+1;i++){

while(true){

if(same(p,playerNum,i)==false) break;

else i=i+1;

}

if(i > playerNum)break;

if(counter%step==0){

System.out.print(i + " ");

p[counter/step-1]=i;

}

counter+=1;

}

}

System.out.println();

}

public static void main(String[] args) {

play(10, 7);

}

}

187、写一个方法1000 的阶乘。【较难】

答：C++的代码实现如下：

#include <iostream>

#include <iomanip>

#include <vector>

using namespace std;

class longint {

private:

vector<int> iv;

public:

longint(void) { iv.push\_back(1); }

longint& multiply(const int &);

friend ostream& operator<<(ostream &, const longint &);

};

ostream& operator<<(ostream &os, const longint &v) {

vector<int>::const\_reverse\_iterator iv\_iter = v.iv.rbegin();

os << \*iv\_iter++;

for ( ; iv\_iter < v.iv.rend(); ++iv\_iter) {

os << setfill('0') << setw(4) << \*iv\_iter;

}

return os;

}

longint& longint::multiply(const int &rv) {

vector<int>::iterator iv\_iter = iv.begin();

int overflow = 0, product = 0;

for ( ; iv\_iter < iv.end(); ++iv\_iter) {

product = (\*iv\_iter) \* rv;

product += overflow;

overflow = 0;

if (product > 10000) {

overflow = product / 10000;

product -= overflow \* 10000;

}

iv\_iter = product;

}

if (0 != overflow) {

iv.push\_back(overflow);

}

return \*this;

}

int main(int argc, char \*\*argv) {

longint result;

int l = 0;

if(argc==1){

cout << "like: multiply 1000" << endl;

exit(0);

}

sscanf(argv[1], "%d", &l);

for (int i = 2; i <= l; ++i) {

result.multiply(i);

}

cout << result << endl;

return 0;

}

# 八、C++部分

188、以下三条输出语句分别输出什么？【基础】

char str1[] = "abc";

char str2[] = "abc";

const char str3[] = "abc";

const char str4[] = "abc";

const char\* str5 = "abc";

const char\* str6 = "abc";

cout << boolalpha << (str1==str2) << endl; //输出什么?

cout << boolalpha << (str3==str4) << endl; //输出什么?

cout << boolalpha << (str5==str6) << endl; //输出什么?

答：输出为：false、false、true。

189、以下反向遍历array 数组的方法有什么错误？【基础】

vector<int> array;

array.push\_back(1);

array.push\_back(2);

array.push\_back(3);

//反向遍历array 数组：

for(vector<int>::size\_type i=array.size()-1; i>=0; --i){

cout << array[i] << endl;

}

答：for 循环中的变量i 的类型不应定义为vector<int>::size\_type，

因为该类型为无符号数值类型，故循环条件将恒成立，为死循环，应将其类型定义为有符号的int 类型。

190、以下代码有什么问题？【基础】

cout << (true ? 1 : "1") << endl;

答：运算符中两个可选值的类型不同。

191、以下代码有什么问题？【基础】

typedef vector<int> IntArray;

IntArray array;

array.push\_back(1);

array.push\_back(2);

array.push\_back(2);

array.push\_back(3);

//删除array 数组中所有的2

for(IntArray::iterator itor=array.begin(); itor!=array.end();

++itor){

if(2==\*itor) {

array.erase(itor);

}

}

答：for 循环中的if 语句后的array.erase(itor)语句，它将迭代器itor 所指向的元素删除后会自动下移一位，故应在其后加上语句：itor--;

192、以下代码中的两个sizeof 用法有问题吗？【基础】

void upperCase(char str[]){ //将str 中的小写字母转换成大写字母

for(int i=0; i<sizeof(str)/sizeof(str[0]); ++i){

if('a'<=str[i] && str[i]<='z')

str[i] -= ('a'-'A');

}

}

int main(){

char str[] = "aBcDe";

cout << "str 字符串长度为:" << sizeof(str)/sizeof(str[0]);

cout << endl;

upperCase(str);

cout << str << endl;

return 0;

}

答：在upperCase 方法中，for 循环的sizeof(str)的值将总是4，所以该方法

只能将参数中的字符串的前四个字符转换成大写字母。

193、以下代码能够编译通过吗？为什么？【基础】

unsigned int const size1 = 2;

char str1[size1];

unsigned int temp = 0;

cin >> temp;

unsigned int const size2 = temp;

char str2[size2];

答：能；

194、以下代码有什么问题？【基础】

struct Test{

Test(int){}

Test(){}

void fun(){}

};

void main(void){

Test a(1);

a.fun();

Test b();

b.fun();

}

答：main 函数的返回类型应为int；不能对b 调用fun（）方法。

195、以下代码中的输出语句输出0 吗？为什么？【基础】

struct CLS{

int m\_i;

CLS(int i):m\_i(i){ }

CLS(){ CLS(0);}

};

int main(){

CLS obj;

cout <<obj.m\_i << endl;

}

答：输出不是0；

196、C++中的空类，默认产生哪些类成员函数？【基础】

答：空类中默认包含的成员函数如下:

class Empty{

public:

Empty(); //缺省构造函数

Empty( const Empty& ); //拷贝构造函数

~Empty(); //析构函数

Empty& operator=( const Empty& ); //赋值运算符

Empty\* operator&(); //取址运算符

const Empty\* operator&() const; //取址运算符const

};

197、统计一篇文章中单词个数。【基础】

答：代码如下：

include<iostream>

#include<fstream>

using namespace std;

int main(){

ifstream fin("t.txt");

if(!fin){

cout<<"can't open file"<<endl;

return -1;

}

int count = 0;

char buf[256];

memset(buf, 0, 256);

while(1){

fin2>>buf;

if(fin2.eof())

break;

count++;

}

cout<<"The number of the words is : "<<count<<endl;

fin2.close();

return 0;

}

198、写一个函数，完成内存之间的拷贝。【中等难度】

答：代码如下：

void\* mymemcpy(void\* dest, const void\* src, size\_t count){

char\* pdest = static\_cast<char\*>(dest);

const char\* psrc = static\_cast<const char\*>(src);

if(pdest>psrc && pdest<psrc+count){ //能考虑到这种情况就行了

for(size\_t i=count-1; i!=-1; --i){

pdest[i] = psrc[i];

}

}else{

for(size\_t i=0; i<count; ++i){

pdest[i] = psrc[i];

}

}

return dest;

}

int main(){

char str[] = "0123456789";

mymemcpy(str+1, str+0, 9);

cout << str << endl; //将输出"0012345678"

return 0;

}

199、非C++内建类型A 和B，在哪几种情况下B 能隐式转化为A？【较难】

答：a）class B : public A{……}//B 公有继承自A，可以是间接继承的

b）class B{operator A();}//B 实现了隐式转化为A 的转化

c）class A{ A(const B&);}//A 实现了non-explicit 的参数为B 构造函数(可以有其他带带默认值的参数)

d）A& operator= (const A&);//赋值操作，虽不是正宗的隐式类型转换，但也可以勉强算一个

200、以下代码有什么问题？【较难】

void char2Hex(char c){ //将字符以16 进制显示

char ch = c/0x10 + '0';

if(ch>'9') ch += ('A'-'9'-1);

char cl = c%0x10 + '0';

if(cl>'9') cl += ('A'-'9'-1);

cout << ch << cl << ' ';

}

int main(){

char str[] = "I love 中国";

for(size\_t i=0; i<strlen(str); ++i)

char2Hex(str[i]);

cout << endl;

return 0;

}

答：

201、以下两条输出语句分别输出什么？【较难】

float a = 1.0f;

cout << (int)a << endl;

cout << (int&)a << endl;

cout << boolalpha << ((int)a==(int&)a) << endl; //输出什么

float b = 0.0f;

cout << (int)b << endl;

cout << (int&)b << endl;

cout << boolalpha << ((int)b==(int&)b) << endl;//输出什么

答：第一处输出false，第二处输出true。

# 九、关于堆和栈的那些事

## 问题描述

编程语言书籍中经常解释值类型被创建在栈上，引用类型被创建在堆上，但是并没有本质上解释这堆和栈是什么。我仅有高级语言编程经验，没有看过对此更清晰的解释。我的意思是我理解什么是栈，但是它们到底是什么，在哪儿呢（站在实际的计算机物理内存的角度上看）？

1. 在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？
2. 它们的作用范围是什么？
3. 它们的大小由什么决定？
4. 哪个更快？

**答案一**

栈是为执行线程留出的内存空间。当函数被调用的时候，栈顶为局部变量和一些 bookkeeping 数据预留块。当函数执行完毕，块就没有用了，可能在下次的函数调用的时候再被使用。栈通常用后进先出（LIFO）的方式预留空间；因此最近的保留块（reserved block）通常最先被释放。这么做可以使跟踪堆栈变的简单；从栈中释放块（free block）只不过是指针的偏移而已。

堆（heap）是为动态分配预留的内存空间。和栈不一样，从堆上分配和重新分配块没有固定模式；你可以在任何时候分配和释放它。这样使得跟踪哪部分堆已经被分配和被释放变的异常复杂；有许多定制的堆分配策略用来为不同的使用模式下调整堆的性能。

每一个线程都有一个栈，但是每一个应用程序通常都只有一个堆（尽管为不同类型分配内存使用多个堆的情况也是有的）。

直接回答你的问题： 1. 当线程创建的时候，操作系统（OS）为每一个系统级（system-level）的线程分配栈。通常情况下，操作系统通过调用语言的运行时（runtime）去为应用程序分配堆。 2. 栈附属于线程，因此当线程结束时栈被回收。堆通常通过运行时在应用程序启动时被分配，当应用程序（进程）退出时被回收。 3. 当线程被创建的时候，设置栈的大小。在应用程序启动的时候，设置堆的大小，但是可以在需要的时候扩展（分配器向操作系统申请更多的内存）。 4. 栈比堆要快，因为它存取模式使它可以轻松的分配和重新分配内存（指针/整型只是进行简单的递增或者递减运算），然而堆在分配和释放的时候有更多的复杂的 bookkeeping 参与。另外，在栈上的每个字节频繁的被复用也就意味着它可能映射到处理器缓存中，所以很快（译者注：局部性原理）。

**答案二**

**Stack:**

1. 和堆一样存储在计算机 RAM 中。
2. 在栈上创建变量的时候会扩展，并且会自动回收。
3. 相比堆而言在栈上分配要快的多。
4. 用数据结构中的栈实现。
5. 存储局部数据，返回地址，用做参数传递。
6. 当用栈过多时可导致栈溢出（无穷次（大量的）的递归调用，或者大量的内存分配）。
7. 在栈上的数据可以直接访问（不是非要使用指针访问）。
8. 如果你在编译之前精确的知道你需要分配数据的大小并且不是太大的时候，可以使用栈。
9. 当你程序启动时决定栈的容量上限。

**Heap：**

1. 和栈一样存储在计算机RAM。
2. 在堆上的变量必须要手动释放，不存在作用域的问题。数据可用 delete, delete[] 或者 free 来释放。
3. 相比在栈上分配内存要慢。
4. 通过程序按需分配。
5. 大量的分配和释放可造成内存碎片。
6. 在 C++ 中，在堆上创建数的据使用指针访问，用 new 或者 malloc 分配内存。
7. 如果申请的缓冲区过大的话，可能申请失败。
8. 在运行期间你不知道会需要多大的数据或者你需要分配大量的内存的时候，建议你使用堆。
9. 可能造成内存泄露。

举例：

int foo()  
{  
char \*pBuffer; //<–nothing allocated yet (excluding the pointer itself, which is allocated here on the stack).  
bool b = true; // Allocated on the stack.  
if(b)  
{  
//Create 500 bytes on the stack  
char buffer[500];

//Create 500 bytes on the heap  
pBuffer = new char[500];

}//<– buffer is deallocated here, pBuffer is not  
}//<— oops there’s a memory leak, I should have called delete[] pBuffer;

**答案三**

堆和栈是两种内存分配的两个统称。可能有很多种不同的实现方式，但是实现要符合几个基本的概念:

1.对栈而言，栈中的新加数据项放在其他数据的顶部，移除时你也只能移除最顶部的数据（不能越位获取）。



2.对堆而言，数据项位置没有固定的顺序。你可以以任何顺序插入和删除，因为他们没有“顶部”数据这一概念。



上面上个图片很好的描述了堆和栈分配内存的方式。

**在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？**

如前所述，堆和栈是一个统称，可以有很多的实现方式。计算机程序通常有一个栈叫做[调用栈](http://en.wikipedia.org/wiki/Call_stack)，用来存储当前函数调用相关的信息（比如：主调函数的地址，局部变量），因为函数调用之后需要返回给主调函数。栈通过扩展和收缩来承载信息。实际上，程序不是由运行时来控制的，它由编程语言、操作系统甚至是系统架构来决定。

堆是在任何内存中动态和随机分配的（内存的）统称；也就是无序的。内存通常由操作系统分配，通过应用程序调用 API 接口去实现分配。在管理动态分配内存上会有一些额外的开销，不过这由操作系统来处理。

**它们的作用范围是什么？**

调用栈是一个低层次的概念，就程序而言，它和“作用范围”没什么关系。如果你反汇编一些代码，你就会看到指针引用堆栈部分。就高级语言而言，语言有它自己的范围规则。一旦函数返回，函数中的局部变量会直接直接释放。你的编程语言就是依据这个工作的。

在堆中，也很难去定义。作用范围是由操作系统限定的，但是你的编程语言可能增加它自己的一些规则，去限定堆在应用程序中的范围。体系架构和操作系统是使用虚拟地址的，然后由处理器翻译到实际的物理地址中，还有页面错误等等。它们记录那个页面属于那个应用程序。不过你不用关心这些，因为你仅仅在你的编程语言中分配和释放内存，和一些错误检查（出现分配失败和释放失败的原因）。

**它们的大小由什么决定？**

依旧，依赖于语言，编译器，操作系统和架构。栈通常提前分配好了，因为栈必须是连续的内存块。语言的编译器或者操作系统决定它的大小。不要在栈上存储大块数据，这样可以保证有足够的空间不会溢出，除非出现了无限递归的情况（额，栈溢出了）或者其它不常见了编程决议。

堆是任何可以动态分配的内存的统称。这要看你怎么看待它了，它的大小是变动的。在现代处理器中和操作系统的工作方式是高度抽象的，因此你在正常情况下不需要担心它实际的大小，除非你必须要使用你还没有分配的内存或者已经释放了的内存。

**哪个更快一些？**

栈更快因为所有的空闲内存都是连续的，因此不需要对空闲内存块通过列表来维护。只是一个简单的指向当前栈顶的指针。编译器通常用一个专门的、快速的寄存器来实现。更重要的一点事是，随后的栈上操作通常集中在一个内存块的附近，这样的话有利于处理器的高速访问（译者注：局部性原理）。

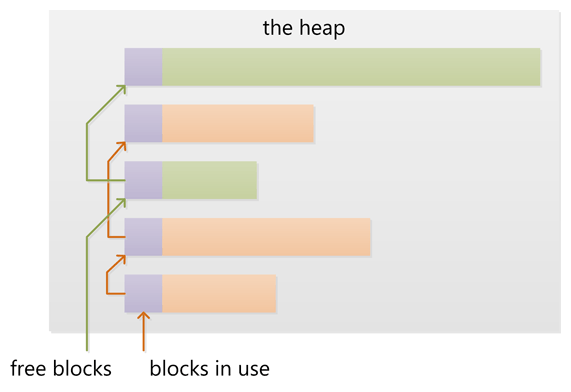
**答案四**

你问题的答案是依赖于实现的，根据不同的编译器和处理器架构而不同。下面简单的解释一下：

1. 栈和堆都是用来从底层操作系统中获取内存的。
2. 在多线程环境下每一个线程都可以有他自己完全的独立的栈，但是他们共享堆。并行存取被堆控制而不是栈。

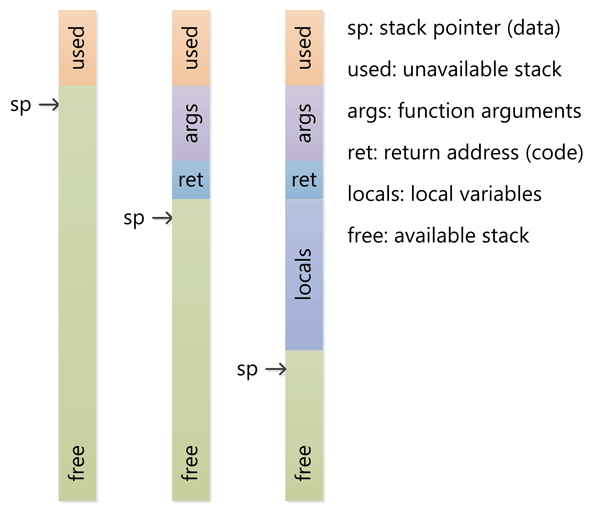
**堆:**

1. 堆包含一个链表来维护已用和空闲的内存块。在堆上新分配（用 new 或者 malloc）内存是从空闲的内存块中找到一些满足要求的合适块。这个操作会更新堆中的块链表。这些元信息也存储在堆上，经常在每个块的头部一个很小区域。
2. 堆的增加新快通常从地地址向高地址扩展。因此你可以认为堆随着内存分配而不断的增加大小。如果申请的内存大小很小的话，通常从底层操作系统中得到比申请大小要多的内存。
3. 申请和释放许多小的块可能会产生如下状态：在已用块之间存在很多小的空闲块。进而申请大块内存失败，虽然空闲块的总和足够，但是空闲的小块是零散的，不能满足申请的大小，。这叫做“堆碎片”。
4. 当旁边有空闲块的已用块被释放时，新的空闲块可能会与相邻的空闲块合并为一个大的空闲块，这样可以有效的减少“堆碎片”的产生。



**栈:**

1. 栈经常与 sp 寄存器（译者注：”stack pointer”，了解汇编的朋友应该都知道）一起工作，最初 sp 指向栈顶（栈的高地址）。
2. CPU 用 push 指令来将数据压栈，用 pop 指令来弹栈。当用 push 压栈时，sp 值减少（向低地址扩展）。当用 pop 弹栈时，sp 值增大。存储和获取数据都是 CPU 寄存器的值。
3. 当函数被调用时，CPU使用特定的指令把当前的 IP （译者注：“instruction pointer”，是一个寄存器，用来记录 CPU 指令的位置）压栈。即执行代码的地址。CPU 接下来将调用函数地址赋给 IP ，进行调用。当函数返回时，旧的 IP 被弹栈，CPU 继续去函数调用之前的代码。
4. 当进入函数时，sp 向下扩展，扩展到确保为函数的局部变量留足够大小的空间。如果函数中有一个 32-bit 的局部变量会在栈中留够四字节的空间。当函数返回时，sp 通过返回原来的位置来释放空间。
5. 如果函数有参数的话，在函数调用之前，会将参数压栈。函数中的代码通过 sp 的当前位置来定位参数并访问它们。
6. 函数嵌套调用和使用魔法一样，每一次新调用的函数都会分配函数参数，返回值地址、局部变量空间、嵌套调用的活动记录都要被压入栈中。函数返回时，按照正确方式的撤销。
7. 栈要受到内存块的限制，不断的函数嵌套/为局部变量分配太多的空间，可能会导致栈溢出。当栈中的内存区域都已经被使用完之后继续向下写（低地址），会触发一个 CPU 异常。这个异常接下会通过语言的运行时转成各种类型的栈溢出异常。（译者注：“不同语言的异常提示不同，因此通过语言运行时来转换”我想他表达的是这个含义）



\*函数的分配可以用堆来代替栈吗？

不可以的，函数的活动记录（即局部或者自动变量）被分配在栈上， 这样做不但存储了这些变量，而且可以用来嵌套函数的追踪。

堆的管理依赖于运行时环境，C 使用 malloc ，C++ 使用 new ，但是很多语言有垃圾回收机制。

栈是更低层次的特性与处理器架构紧密的结合到一起。当堆不够时可以扩展空间，这不难做到，因为可以有库函数可以调用。但是，扩展栈通常来说是不可能的，因为在栈溢出的时候，执行线程就被操作系统关闭了，这已经太晚了。

## 译者注

关于堆栈的这个帖子，对我来说，收获非常多。我之前看过一些资料，自己写代码的时候也常常思考。就这方面，也和祥子（我的大学舍友，现在北京邮电读研，技术牛人）探讨过多次了。但是终究是一个一个的知识点，这个帖子看完之后，豁然开朗，把知识点终于连接成了一个网。这种感觉，经历过的一定懂得，期间的兴奋不言而喻。

这个帖子跟帖者不少，我选了评分最高的四个。这四个之间也有一些是重复的观点。个人钟爱第四个回答者，我看的时候，瞬间高潮了，有木有？不过需要一些汇编语言、操作系统、计算机组成原理的的基础，知道那几个寄存器是干什么的，要知道计算机的流水线指令工作机制，保护/恢复现场等概念。三个回复者都涉及到了操作系统中虚拟内存；在比较速度的时候，大家一定要在脑中对“局部性原理”和计算机高速缓存有一个概念。

如果你把这篇文章看懂了，我相信你收获的不只是堆和栈，你会理解的更多！

兴奋之余，有几点还是要强调的，翻译没有逐字逐词翻译，大部分是通过我个人的知识积累和对回帖者的意图揣测而来的。请大家不要咬文嚼字，逐个推敲，我们的目的在于技术交流，不是么？达到这一目的就够了。

下面是一些不确定点:

1. 我没有听过 bookkeeping data 这种说法，故没有翻译。从上下文理解来看，可以想成是用来寄存器值？函数参数？返回地址？如果有了解具体含义的朋友，烦请告知。
2. 栈和堆栈是一回事，英文表达是 stack，堆是 heap。
3. 调用栈的概念，我是第一次听说，不太熟悉。大家可以去查查资料研究一下。

以上，送给大家，本文结束。

# 十、Java内存模型的历史变迁

## 旧的Java内存模型

Java使用的是共享内存的并发模型，在线程之间共享变量。Java语言定义了线程模型规范，通过内存模型控制线程与变量的交互，从而实现Java线程之间的通信。在JDK5之前，Java一直使用的是旧内存模型。如图1所示。变量保存在由所有线程共享的主内存中，主内存中的变量称为mastingcopy。每个线程都有一个工作内存，它保存变量的workingcopy。旧的内存模型定义了若干规则，通过这些规则来保证线程何时将主内存中的mastingcopy传送到线程的工作内存中；以及线程何时将工作内存中的workingcopy传送回主内存。旧内存模型使用8个操作来定义线程可以执行的动作。

* read（读）操作：主内存把mastingcopy传送到线程的工作内存，以供后面的load操作使用。
* load（装载）操作：线程将由read操作从主内存传送过来的值，放入工作内存中。
* use（使用）操作：线程将变量的workingcopy传送到线程执行引擎。
* assign（赋值）操作：线程将变量值从线程执行引擎传送到线程的工作内存中。
* store（存储）操作：线程将变量的workingcopy传送到主内存，供后面的write操作使用。
* write（写）操作：主内存将由store操作传送过来的值，放入主内存中。
* lock（锁定）操作：线程获得指定对象的锁。
* unlock（解锁）操作：线程释放指定对象的锁。

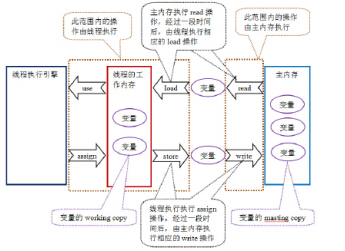


图1 Java旧内存模型

这里的关键是，由于read操作是由主内存执行，而对应的load是由线程执行，read操作和load操作之间是松散耦合的。也就是说，主内存和线程工作内存之间的变量传递是松散耦合的。同样，由于store操作是由线程执行，而对应的write是由主内存执行，store操作和write操作之间是松散耦合的。也就是说，线程工作内存和主内存之间的变量传递是松散耦合的。旧Java内存模型对Java实现如何执行变量的读/写，加锁/解锁，以及volatile变量的读/写，定义了非常严格的规则。这些规则非常复杂，具体详情请参考《JVM规范》，这里就不赘述了。旧Java内存模型通过这些复杂的规则，来保证多线程程序的线程之间，可以可靠地传递共享变量，从而保证多线程程序的正确性。

## 新的Java内存模型

从JDK5开始，Java使用新的内存模型，新内存模型完全抛弃了旧内存模型的主内存和工作内存的概念，也抛弃了旧内存模型的8个内存操作。也就是说，新内存模型完全是重新设计的。

新内存模型引入了一个新的概念，叫happens-before。happens-before的概念最初由LeslieLamport在其一篇影响深远的论文（《Time,ClocksandtheOrderingofEventsinaDistributedSystem》）中提出。LeslieLamport使用happens-before来定义分布式系统中，事件之间的一个偏序关系(partialordering)。LeslieLamport在这篇论文中给出了一个分布式算法，该算法可以将该偏序关系扩展为某种全序关系。

JSR-133使用happens-before的概念来指定两个操作（这里的操作是指程序中对变量的读/写，对锁的加锁和解锁）之间的执行顺序。新内存模型定义了如下的happens-before规则。

* 程序顺序规则：一个线程中的每个操作，happensbefore于该线程中的任意后续操作。
* 监视器锁规则：对一个锁的解锁，happens-before于随后对这个锁的加锁。
* volatile变量规则：对一个volatile域的写，happensbefore于任意后续对这个volatile域的读。
* 传递性：如果Ahappens-beforeB，且BhappensbeforeC，那么Ahappens-beforeC。
* start()规则：如果线程A执行操作ThreadB.start()（启动线程B），那么A线程的ThreadB.start()操作happensbefore于线程B中的任意操作。
* join()规则：如果线程A执行操作ThreadB.join()并成功返回，那么线程B中的任意操作happens-before于线程A从ThreadB.join()操作成功返回。

由于两个操作可以在一个线程之内，也可以是在不同线程之间。因此JMM可以通过happens-before关系向程序员提供跨线程的内存可见性保证（如果A线程的写操作a与B线程的读操作b之间存在happens-before关系，尽管a操作和b操作在不同的线程中执行，但JMM

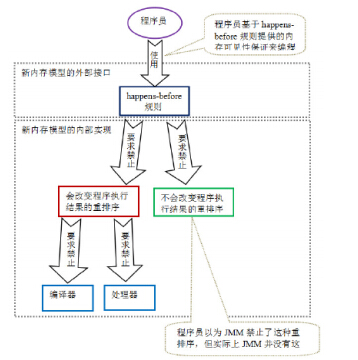


图2新内存模型的设计示意图

向程序员保证a操作将对b操作可见）。在新内存模型向程序员提供happens-before规则，程序员只需要与happens-before打交道即可，因此Java程序员的学习负担大大降低。同时，新内存模型允许不会改变程序结果的重排序，这可以最大限度地放松对编译器和处理器的束缚，新内存模型的执行性能比旧内存模型要好。

# 十一、JAVA代码效率优化

       最近在想自己编程时是否注意过代码的效率问题，得出的答案是：没有。代码只是实现了功能，至于效率高不高没怎么关注，这应该是JAVA程序员进阶的时候需要考虑的问题，不再是单纯的实现功能，也不是完全依赖GC而不关注内存中发生了什么，而要考虑到代码的性能。下面是网上找的一篇关于JAVA代码优化的文章，觉得不错，就转载了。这里面设计到了JAVA基础和J2EE方面的优化建议，有时间会整理一下，现在先转载。

       原文地址是：http://blog.163.com/user\_zhaopeng/blog/static/16602270820122105731329/

## 1、 尽量指定类的final修饰符 带有final修饰符的类是不可派生的。

       如果指定一个类为final，则该类所有的方法都是final。Java编译器会寻找机会内联（inline）所有的 final方法（这和具体的编译器实现有关）。此举能够使性能平均提高50% 。

## 2、 尽量重用对象。

       **特别是String 对象的使用中，出现字符串连接情况时应用StringBuffer 代替**。由于系统不仅要花时间生成对象，以后可能还需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。

## 尽量使用局部变量

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（Stack）中，速度较快。  其他变量，如静态变量、实例变量等，都在堆（Heap）中创建，速度较慢。另外，依赖于具体的编译器/JVM，局部变量还可能得到进一步优化。   
  
  
4、 不要重复初始化变量

        默认情况下，调用类的构造函数时， Java会把变量初始化成确定的值：所有的对象被设置成null，整数变量（byte、short、int、long）设置成0，float和 double变量设置成0.0，逻辑值设置成false。当一个类从另一个类派生时，这一点尤其应该注意，因为用new关键词创建一个对象时，构造函数链 中的所有构造函数都会被自动调用。

## 5、 在JAVA + ORACLE 的应用系统开发中，java中内嵌的SQL语句尽量使用大写的形式，以减轻ORACLE解析器的解析负担。

## I/O操作中需要及时释放资源

       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，即使关闭以释放资源。

因为对这些大对象的操作会造成系统大的开销，稍有不慎，会导致严重的后果。

## 保证过期对象的及时回收

由于JVM的有其自身的GC机制，不需要程序开发者的过多考虑，从一定程度上减轻了开发者负担，但同时也遗漏了隐患，过分的创建对象会消耗系统的大量内 存，严重时会导致内存泄露，因此，保证过期对象的及时回收具有重要意义。

JVM回收垃圾的条件是：对象不在被引用；然而，JVM的GC并非十分的机智，即使对象满足了垃圾回收的条件也不一定会被立即回收。所以，建议我们在**对象使用完毕，应手动置成null。**

## 8、 在使用同步机制时，应尽量使用方法同步代替代码块同步。

## 9、 尽量减少对变量的重复计算

## 10、尽量采用lazy loading 的策略，即在需要的时候才开始创建。

## 11、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 （Native）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，VM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。**

## 不要在循环中使用： Try { } catch() { } 应把其放置在最外层.

## StringBuffer 的使用

StringBuffer表示了可变的、可写的字符串。   
有三个构造方法 :   
StringBuffer (); //默认分配16个字符的空间   
StringBuffer (int size); //分配size个字符的空间   
StringBuffer (String str); //分配16个字符+str.length()个字符空间   
你可以通过StringBuffer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。  
  
      这里提到的构造函数是StringBuffer(int length)，length参数表示当前的StringBuffer能保持的字符数量。你也可以使用ensureCapacity(int minimumcapacity)方法在StringBuffer对象创建之后设置它的容量。首先我们看看StringBuffer的缺省行为，然后再找 出一条更好的提升性能的途径。   
StringBuffer在内部维护一个字符数组，当你使用缺省的构造函数来创建StringBuffer对象的时候，因为没有设置初始化字符长 度，StringBuffer的容量被初始化为16个字符，也就是说缺省容量就是16个字符。当StringBuffer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，**也就是（2\*旧值+2）。如果你使用缺省值，初始化之后接着往里面追加字符，在你追加到第16个字符的时候它会将容量增加到 34（2\*16+2），当追加到34个字符的时候就会将容量增加到70（2\*34+2）。无论何事只要StringBuffer到达它的最大容量它就不得 不创建一个新的字符数组然后重新将旧字符和新字符都拷贝一遍――这也太昂贵了点。所以总是给StringBuffer设置一个合理的初始化容量值是错不了 的，这样会带来立竿见影的性能增益。StringBuffer初始化过程的调整的作用由此可见一斑。所以，使用一个合适的容量值来初始化 StringBuffer永远都是一个最佳的建议。

## 14、合理的使用Java类 java.util.Vector。

     简单地说，一个Vector就是一个java.lang.Object实例的数组。Vector与数组相似，它的元素可以通过整数形式的索引访问。但是，Vector类型的对象在创建之后，对象的大小能够根据元素的增加或者删除而扩展、缩小。请考虑下面这个向Vector加入元素的例子：   
Object bj = new Object();   
Vector v = new Vector(100000);   
for(int I=0;   
I<100000; I++) { v.add(0,obj); }  
  
       除非有绝对充足的理由要求每次都把新元素插入到Vector的前面，否则上面的代码对性能不利。在默认构造函数中，Vector的初始存储能力 是10个元素，如果新元素加入时存储能力不足，则以后存储能力每次加倍。Vector类就对象StringBuffer类一样，每次扩展存储能力时，所有 现有的元素都要复制到新的存储空间之中。下面的代码片段要比前面的例子快几个数量级：   
Object bj = new Object();   
Vector v = new Vector(100000);   
for(int I=0; I<100000; I++) { v.add(obj); }  
  
       同样的规则也适用于Vector类的remove()方法。由于Vector中各个元素之间不能含有“空隙”，删除除最后一个元素之外的任意其 他元素都导致被删除元素之后的元素向前移动。也就是说，从Vector删除最后一个元素要比删除第一个元素“开销”低好几倍。  
  
       假设要从前面的Vector删除所有元素，我们可以使用这种代码：   
for(int I=0; I<100000; I++)   
{   
v.remove(0);   
}  
  
      但是，与下面的代码相比，前面的代码要慢几个数量级：   
for(int I=0; I<100000; I++)   
{   
v.remove(v.size()-1);   
}  
  
      从Vector类型的对象v删除所有元素的最好方法是：   
v.removeAllElements();  
  
      假设Vector类型的对象v包含字符串“Hello”。考虑下面的代码，它要从这个Vector中删除“Hello”字符串：   
String s = “Hello”;   
int i = v.indexOf(s);   
if(I != -1) v.remove(s);  
  
      这些代码看起来没什么错误，但它同样对性能不利。在这段代码中，indexOf()方法对v进行顺序搜索寻找字符串 “Hello”，remove(s)方法也要进行同样的顺序搜索。改进之后的版本是：   
String s = “Hello”;   
int i = v.indexOf(s);   
if(I != -1) v.remove(i);  
  
       这个版本中我们直接在remove()方法中给出待删除元素的精确索引位置，从而避免了第二次搜索。一个更好的版本是：   
String s = “Hello”; v.remove(s);  
  
       最后，我们再来看一个有关Vector类的代码片段：   
for(int I=0; I++;I < v.length)  
  
       如果v包含100,000个元素，这个代码片段将调用v.size()方法100,000次。虽然size方法是一个简单的方法，但它仍旧需要 一次方法调用的开销，至少JVM需要为它配置以及清除堆栈环境。在这里，for循环内部的代码不会以任何方式修改Vector类型对象v的大小，因此上面 的代码最好改写成下面这种形式：   
int size = v.size(); for(int I=0; I++;I<size)  
  
      虽然这是一个简单的改动，但它仍旧赢得了性能。毕竟，每一个CPU周期都是宝贵的。

## 15、当复制大量数据时，使用System.arraycopy()命令。

## 代码重构：增强代码的可读性。

## 不用new关键词创建类的实例

     用new关键词创建类的实例时，构造函数链中的所有构造函数都会被自动调用。但如果一个对象实现了Cloneable接口，我们可以调用它的 clone()方法。clone()方法不会调用任何类构造函数。   
在使用设计模式（Design Pattern）的场合，如果用Factory模式创建对象，则改用clone()方法创建新的对象实例非常简单。例如，下面是Factory模式的一个 典型实现：   
public static Credit getNewCredit() {   
return new Credit();   
}   
      改进后的代码使用clone()方法，如下所示：   
private static Credit BaseCredit = new Credit();   
public static Credit getNewCredit() {   
return (Credit) BaseCredit.clone();   
}   
上面的思路对于数组处理同样很有用。

## 乘法和除法,用移位操作替代乘法操作可以极大地提高性能。

## 在JSP页面中关闭无用的会话。

一个常见的误解是以为session在有客户端访问时就被创建，然而事实是直到某server端程序调用 HttpServletRequest.getSession(true)这样的语句时才被创建，注意如果JSP没有显示的使用 <%@pagesession=”false”%> 关闭session，则JSP文件在编译成Servlet时将会自动加上这样一条语句HttpSession session = HttpServletRequest.getSession(true);这也是JSP中隐含的session对象的来历。由于session会消耗内 存资源，因此，如果不打算使用session，应该在所有的JSP中关闭它。   
对于那些无需跟踪会话状态的页面，关闭自动创建的会话可以节省一些资源。使用如下page指令：<%@ page session=”false”%>

## JDBC与I/O

如果应用程序需要访问一个规模很大的数据集，则应当考虑使用块提取方式。默认情况下，JDBC每次提取32行数据。举例来说，假设我们要遍历一个5000 行的记录集，JDBC必须调用数据库157次才能提取到全部数据。如果把块大小改成512，则调用数据库的次数将减少到10次。

## Servlet与内存使用

许多开发者随意地把大量信息保存到用户会话之中。一些时候，保存在会话中的对象没有及时地被垃圾回收机制回收。从性能上看，典型的症状是用户感到系统周期 性地变慢，却又不能把原因归于任何一个具体的组件。如果监视JVM的堆空间，它的表现是内存占用不正常地大起大落。   
解决这类内存问题主要有二种办法。第一种办法是，在所有**作用范围为会话的Bean中实现HttpSessionBindingListener接口**。这 样，只要实现valueUnbound()方法，就可以显式地释放Bean使用的资源。  
  
另外一种办法就是尽快地把会话作废。大多数应用服务器都有设置会话作废间隔时间的选项。另外，也可以用编程的方式调用会话的 setMaxInactiveInterval()方法，该方法用来设定在作废会话之前，Servlet容器允许的客户请求的最大间隔时间，以秒计。

## 22、使用缓冲标记

一些应用服务器加入了面向JSP的缓冲标记功能。例如，BEA的WebLogic Server从6.0版本开始支持这个功能，Open Symphony工程也同样支持这个功能。JSP缓冲标记既能够缓冲页面片断，也能够缓冲整个页面。当JSP页面执行时，如果目标片断已经在缓冲之中，则 生成该片断的代码就不用再执行。页面级缓冲捕获对指定URL的请求，并缓冲整个结果页面。对于购物篮、目录以及门户网站的主页来说，这个功能极其有用。对 于这类应用，页面级缓冲能够保存页面执行的结果，供后继请求使用。

## 23、选择合适的引用机制

在典型的JSP应用系统中，页头、页脚部分往往被抽取出来，然后根据需要引入页头、页脚。当前，在JSP页面中引入外部资源的方法主要有两 种：include指令，以及include动作。   
include指令：例如<%@ include file=”copyright.html” %>。该指令在编译时引入指定的资源。在编译之前，带有include指令的页面和指定的资源被合并成一个文件。被引用的外部资源在编译时就确定， 比运行时才确定资源更高效。   
include动作：例如<jsp:include page=”copyright.jsp” />。该动作引入指定页面执行后生成的结果。由于它在运行时完成，因此对输出结果的控制更加灵活。但时，只有当被引用的内容频繁地改变时，或者在对 主页面的请求没有出现之前，被引用的页面无法确定时，使用include动作才合算。

## 24、及时清除不再需要的会话

为了清除不再活动的会话，许多应用服务器都有默认的会话超时时间，一般为30分钟。当应用服务器需要保存更多会话时，如果内存容量不足，操作系统会把部分 内存数据转移到磁盘，应用服务器也可能根据“最近最频繁使用”（Most Recently Used）算法把部分不活跃的会话转储到磁盘，甚至可能抛出“内存不足”异常。在大规模系统中，串行化会话的代价是很昂贵的。当会话不再需要时，应当及时 调用HttpSession.invalidate()方法清除会话。HttpSession.invalidate()方法通常可以在应用的退出页面调 用

## 不要将数组声明为：public static final 。

## HashMap的遍历效率讨论

经常遇到对HashMap中的key和value值对的遍历操作，有如下两种方法：Map<String, String[]> paraMap = new HashMap<String, String[]>();   
…………….//第一个循环   
Set<String> appFieldDefIds = paraMap.keySet();   
for (String appFieldDefId : appFieldDefIds) {   
String[] values = paraMap.get(appFieldDefId);   
……   
}  
  
  
//第二个循环   
for(Entry<String, String[]> entry : paraMap.entrySet()){   
String appFieldDefId = entry.getKey();   
String[] values = entry.getValue();   
…….   
}  
  
  
第一种实现明显的效率不如第二种实现。   
分析如下 Set<String> appFieldDefIds = paraMap.keySet(); 是先从HashMap中取得keySet  
  
  
代码如下：   
public Set<K> keySet() {   
Set<K> ks = keySet;   
return (ks != null ? ks : (keySet = new KeySet()));   
}  
  
  
private class KeySet extends AbstractSet<K> {   
public Iterator<K> iterator() {   
return newKeyIterator();   
}   
public int size() {   
return size;   
}   
public boolean contains(Object o) {   
return containsKey(o);   
}   
public boolean remove(Object o) {   
return HashMap.this.removeEntryForKey(o) != null;   
}   
public void clear() {   
HashMap.this.clear();   
}   
}   
其实就是返回一个私有类KeySet, 它是从AbstractSet继承而来，实现了Set接口。  
  
  
再来看看for/in循环的语法   
for(declaration : expression)   
statement  
  
  
在执行阶段被翻译成如下各式   
for(Iterator<E> #i = (expression).iterator(); #i.hashNext();){   
declaration = #i.next();   
statement   
}  
  
  
因此在第一个for语句for (String appFieldDefId : appFieldDefIds) 中调用了HashMap.keySet().iterator()  
  
  
而这个方法调用了newKeyIterator()  
  
  
Iterator<K> newKeyIterator() {   
return new KeyIterator();   
}   
private class KeyIterator extends HashIterator<K> {   
public K next() {   
return nextEntry().getKey();   
}   
}  
  
  
所以在for中还是调用了  
  
  
在第二个循环for(Entry<String, String[]> entry : paraMap.entrySet())中使用的Iterator是如下的一个内部类  
  
  
private class EntryIterator extends HashIterator<Map.Entry<K,V>> {   
public Map.Entry<K,V> next() {   
return nextEntry();   
}   
}  
  
  
此时第一个循环得到key，第二个循环得到HashMap的Entry效率就是从循环里面体现出来的第二个循环此致可以直接取key和value值 而第一个循环还是得再利用HashMap的get(Object key)来取value值现在看看HashMap的get(Object key)方法   
public V get(Object key) {   
Object k = maskNull(key);   
int hash = hash(k);   
int i = indexFor(hash, table.length); //Entry[] table   
Entry<K,V> e = table;   
while (true) {   
if (e == null)   
return null;   
if (e.hash == hash && eq(k, e.key))   
return e.value;   
e = e.next;   
}   
}   
其实就是再次利用Hash值取出相应的Entry做比较得到结果，所以使用第一中循环相当于两次进入HashMap的Entry  
  
  
中而第二个循环取得Entry的值之后直接取key和value，效率比第一个循环高。其实按照Map的概念来看也应该是用第二个循环好一点，它本 来就是key和value的值对，将key和value分开操作在这里不是个好选择。

## array(数组) 和 ArryList的使用

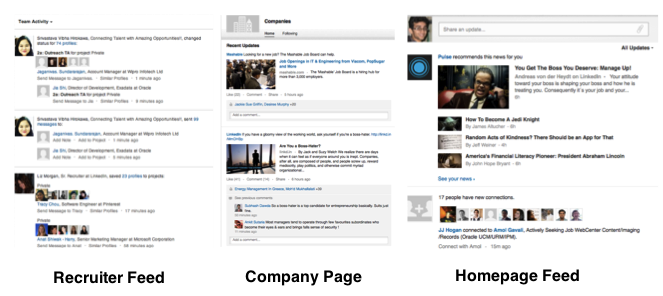
array（[]）：最高效；但是其容量固定且无法动态改变；   
ArrayList：容量可动态增长；但牺牲效率；   
**基于效率和类型检验，应尽可能使用array，无法确定数组大小时才使用ArrayList！**   
ArrayList是Array的复杂版本   
ArrayList内部封装了一个Object类型的数组，从一般的意义来说，它和数组没有本质的差别，甚至于ArrayList的许多方法，如 Index、IndexOf、Contains、Sort等都是在内部数组的基础上直接调用Array的对应方法。   
ArrayList存入对象时，抛弃类型信息，所有对象屏蔽为Object，编译时不检查类型，但是运行时会报错。   
注：jdk5中加入了对泛型的支持，已经可以在使用ArrayList时进行类型检查。   
从这一点上看来，ArrayList与数组的区别主要就是由于动态增容的效率问题了

## 28、尽量使用HashMap 和ArrayList ,除非必要，否则不推荐使用HashTable和Vector ，后者由于使用同步机制，而导致了性能的开销。

# 十二、高吞吐低延迟Java应用的垃圾回收优化

高性能应用构成了现代网络的支柱。LinkedIn有许多内部高吞吐量服务来满足每秒数千次的用户请求。要优化用户体验，低延迟地响应这些请求非常重要。

比如说，用户经常用到的一个功能是了解动态信息——不断更新的专业活动和内容的列表。动态信息在LinkedIn随处可见，包括公司页面，学校页面以及最重要的主页。基础动态信息数据平台为我们的经济图谱(会员，公司，群组等等)中各种实体的更新建立索引，它必须高吞吐低延迟地实现相关的更新。

[](http://www.importnew.com/?attachment_id=11339)

*图1 LinkedIn 动态信息*

这些高吞吐低延迟的Java应用转变为产品，开发人员必须确保应用开发周期的每个阶段一致的性能。确定优化垃圾回收(Garbage Collection,GC)的设置对达到这些指标非常关键。

本文章通过一系列步骤来明确需求并优化GC，目标读者是为实现应用的高吞吐低延迟，对使用系统方法优化GC感兴趣的开发人员。文章中的方法来自于LinkedIn构建下一代动态信息数据平台过程。这些方法包括但不局限于以下几点：[并发标记清除](http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf)(Concurrent Mark Sweep,CMS)和[G1](http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/G1GettingStarted/index.html)垃圾回收器的CPU和内存开销，避免长期存活对象引起的持续GC周期，优化GC线程任务分配使性能提升，以及GC停顿时间可预测所需的OS设置。

## 优化GC的正确时机？

GC运行随着代码级的优化和工作负载而发生变化。因此在一个已实施性能优化的接近完成的代码库上调整GC非常重要。但是在端到端的基本原型上进行初步分析也很有必要，该原型系统使用存根代码并模拟了可代表产品环境的工作负载。这样可以捕捉该架构延迟和吞吐量的真实边界，进而决定是否纵向或横向扩展。

在下一代动态信息数据平台的原型阶段，几乎实现了所有端到端的功能，并且模拟了当前产品基础架构所服务的查询负载。从中我们获得了多种用来衡量应用性能的工作负载特征和足够长时间运行情况下的GC特征。

## 优化GC的步骤

下面是为满足高吞吐，低延迟需求优化GC的总体步骤。也包括在动态信息数据平台原型实施的具体细节。可以看到在ParNew/CMS有最好的性能，但我们也实验了G1垃圾回收器。

### 1.理解GC基础知识

理解GC工作机制非常重要，因为需要调整大量的参数。Oracle的Hotspot JVM 内存管理[白皮书](http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf)是开始学习Hotspot JVM GC算法非常好的资料。了解G1垃圾回收器，请查看该[论文](http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/G1GettingStarted/index.html)。

### 2. 仔细考量GC需求

为降低应用性能的GC开销，可以优化GC的一些特征。吞吐量、延迟等这些GC特征应该长时间测试运行观察，确保特征数据来自于应用程序的处理对象数量发生变化的多个GC周期。

* Stop-the-world回收器回收垃圾时会暂停应用线程。停顿的时长和频率不应该对应用遵守SLA产生不利的影响。
* 并发GC算法与应用线程竞争CPU周期。这个开销不应该影响应用吞吐量。
* 不压缩GC算法会引起堆碎片化，导致full GC长时间Stop-the-world停顿。
* 垃圾回收工作需要占用内存。一些GC算法产生更高的内存占用。如果应用程序需要较大的堆空间，要确保GC的内存开销不能太大。
* 清晰地了解GC日志和常用的JVM参数对简单调整GC运行很有必要。GC运行随着代码复杂度增长或者工作特性变化而改变。

我们使用Linux OS的Hotspot [Java7u51](http://www.oracle.com/technetwork/java/javase/7u51-relnotes-2085002.html)，32GB堆内存，6GB新生代(young generation)和-XX:CMSInitiatingOccupancyFraction值为70(老年代GC触发时其空间占用率)开始实验。设置较大的堆内存用来维持长期存活对象的对象缓存。一旦这个缓存被填充，提升到老年代的对象比例显著下降。

使用初始的GC配置，每三秒发生一次80ms的新生代GC停顿，超过百分之99.9的应用延迟100ms。这样的GC很可能适合于SLA不太严格要求延迟的许多应用。然而，我们的目标是尽可能降低百分之99.9应用的延迟，为此GC优化是必不可少的。

### 3.理解GC指标

优化之前要先衡量。了解GC日志的详细细节(使用这些选项：-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCApplicationStoppedTime)可以对该应用的GC特征有总体的把握。

LinkedIn的内部监控和报表系统，[inGraphs](https://engineering.linkedin.com/32/eric-intern-origin-ingraphs)和[Naarad](https://github.com/linkedin/naarad)，生成了各种有用的指标可视化图形，比如GC停顿时间百分比，一次停顿最大持续时间，长时间内GC频率。除了Naarad，有很多开源工具比如[gclogviewer](https://code.google.com/p/gclogviewer/)可以从GC日志创建可视化图形。

在这个阶段，需要确定GC频率和停顿时长是否影响应用满足延迟性需求的能力。

### 4.降低GC频率

在分代GC算法中，降低回收频率可以通过：(1)降低对象分配/提升率；(2)增加代空间的大小。

在Hotspot JVM中，新生代GC停顿时间取决于一次垃圾回收后对象的数量，而不是新生代自身的大小。增加新生代大小对于应用性能的影响需要仔细评估：

* 如果更多的数据存活而且被复制到survivor区域，或者每次垃圾回收更多的数据提升到老年代，增加新生代大小可能导致更长的新生代GC停顿。
* 另一方面，如果每次垃圾回收后存活对象数量不会大幅增加，停顿时间可能不会延长。在这种情况下，减少GC频率可能使应用总体延迟降低和(或)吞吐量增加。

对于大部分为短期存活对象的应用，仅仅需要控制前面所说的参数。对于创建长期存活对象的应用，就需要注意，被提升的对象可能很长时间都不能被老年代GC周期回收。如果老年代GC触发阈值(老年代空间占用率百分比)比较低，应用将陷入不断的GC周期。设置高的GC触发阈值可避免这一问题。

由于我们的应用在堆中维持了长期存活对象的较大缓存，将老年代GC触发阈值设置为-XX:CMSInitiatingOccupancyFraction=92 -XX:+UseCMSInitiatingOccupancyOnly。我们也试图增加新生代大小来减少新生代回收频率，但是并没有采用，因为这增加了应用延迟。

### 5.缩短GC停顿时间

减少新生代大小可以缩短新生代GC停顿时间，因为这样被复制到survivor区域或者被提升的数据更少。但是，正如前面提到的，我们要观察减少新生代大小和由此导致的GC频率增加对于整体应用吞吐量和延迟的影响。新生代GC停顿时间也依赖于tenuring threshold(提升阈值)和空间大小(见第6步)。

使用CMS尝试最小化堆碎片和与之关联的老年代垃圾回收full GC停顿时间。通过控制对象提升比例和减小-XX:CMSInitiatingOccupancyFraction的值使老年代GC在低阈值时触发。所有选项的细节调整和他们相关的权衡，请查看[Web Services的Java 垃圾回收](http://engineering.linkedin.com/26/tuning-java-garbage-collection-web-services)和[Java 垃圾回收精粹](http://mechanical-sympathy.blogspot.com/2013/07/java-garbage-collection-distilled.html)。

我们观察到Eden区域的大部分新生代被回收，几乎没有对象在survivor区域死亡，所以我们将tenuring threshold从8降低到2(使用选项：-XX:MaxTenuringThreshold=2),为的是缩短新生代垃圾回收消耗在数据复制上的时间。

我们也注意到新生代回收停顿时间随着老年代空间占用率上升而延长。这意味着来自老年代的压力使得对象提升花费更多的时间。为解决这个问题，将总的堆内存大小增加到40GB，减小-XX:CMSInitiatingOccupancyFraction的值到80，更快地开始老年代回收。尽管-XX:CMSInitiatingOccupancyFraction的值减小了，增大堆内存可以避免不断的老年代GC。在本阶段，我们获得了70ms新生代回收停顿和百分之99.9延迟80ms。

6.优化GC工作线程的任务分配

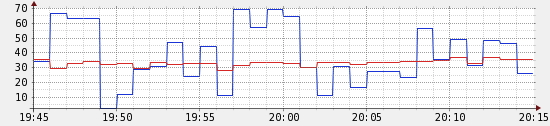
进一步缩短新生代停顿时间，我们决定研究优化与GC线程绑定任务的选项。

-XX:ParGCCardsPerStrideChunk 选项控制GC工作线程的任务粒度，可以帮助不使用[补丁](http://www.importnew.com/%5bhttp:/bugs.java.com/bugdatabase/view_bug.do?bug_id=7068625%5d(http://bugs.java.com/bugdatabase/view_bug.do?bug_id=7068625))而获得最佳性能，这个补丁用来优化新生代垃圾回收的[卡表扫描时间](http://blog.ragozin.info/2012/03/secret-hotspot-option-improving-gc.html)。有趣的是新生代GC时间随着老年代空间的增加而延长。将这个选项值设为32678，新生代回收停顿时间降低到平均50ms。此时百分之99.9应用延迟60ms。

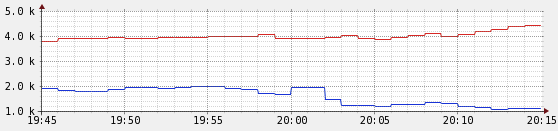
也有其他选项将任务映射到GC线程，如果OS允许的话，-XX:+BindGCTaskThreadsToCPUs选项绑定GC线程到个别的CPU核。-XX:+UseGCTaskAffinity使用affinity参数将任务分配给GC工作线程。然而，我们的应用并没有从这些选项发现任何益处。实际上，一些调查显示这些选项在Linux系统不起作用[1,2]。

7.了解GC的CPU和内存开销

并发GC通常会增加CPU的使用。我们观察了运行良好的CMS默认设置，并发GC和G1垃圾回收器共同工作引起的CPU使用增加显著降低了应用的吞吐量和延迟。与CMS相比，G1可能占用了应用更多的内存开销。对于低吞吐量的非计算密集型应用，GC的高CPU使用率可能不需要担心。

[](http://www.importnew.com/?attachment_id=11340)

*图2 ParNew/CMS和G1的CPU使用百分数%：相对来说CPU使用率变化明显的节点使用G1  
选项-XX:G1RSetUpdatingPauseTimePercent=20*

[](http://www.importnew.com/?attachment_id=11338)

*图3 ParNew/CMS和G1每秒服务的请求数：吞吐量较低的节点使用G1  
选项-XX:G1RSetUpdatingPauseTimePercent=20*

8.为GC优化系统内存和I/O管理

通常来说，GC停顿发生在(1)低用户时间，高系统时间和高时钟时间和(2)低用户时间，低系统时间和高时钟时间。这意味着基础的进程/OS设置存在问题。情况(1)可能说明Linux从JVM偷页，情况(2)可能说明清除磁盘缓存时Linux启动GC线程，等待I/O时线程陷入内核。在这些情况下如何设置参数可以参考[该PPT](http://www.slideshare.net/cuonghuutran/gc-andpagescanattacksbylinux)。

为避免运行时性能损失，启动应用时使用JVM选项-XX:+AlwaysPreTouch访问和清零页面。设置vm.swappiness为零，除非在绝对必要时，OS不会交换页面。

可能你会使用mlock将JVM页pin在内存中，使OS不换出页面。但是，如果系统用尽了所有的内存和交换空间，OS通过kill进程来回收内存。通常情况下，Linux内核会选择高驻留内存占用但还没有长时间运行的进程([OOM情况下killing进程的工作流](https://www.kernel.org/doc/gorman/html/understand/understand016.html))。对我们而言，这个进程很有可能就是我们的应用程序。一个服务具备优雅降级(适度退化)的特点会更好，服务突然故障预示着不太好的可操作性——因此，我们没有使用mlock而是vm.swappiness避免可能的交换惩罚。

## LinkedIn动态信息数据平台的GC优化

对于该平台原型系统，我们使用Hotspot JVM的两个算法优化垃圾回收：

* 新生代垃圾回收使用ParNew，老年代垃圾回收使用CMS。
* 新生代和老年代使用G1。G1用来解决堆大小为6GB或者更大时存在的低于0.5秒稳定的、可预测停顿时间的问题。在我们用G1实验过程中，尽管调整了各种参数，但没有得到像ParNew/CMS一样的GC性能或停顿时间的可预测值。我们查询了使用G1发生内存泄漏相关的一个bug[3]，但还不能确定根本原因。

使用ParNew/CMS，应用每三秒40-60ms的新生代停顿和每小时一个CMS周期。JVM选项如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // JVM sizing options  -server -Xms40g -Xmx40g -XX:MaxDirectMemorySize=4096m -XX:PermSize=256m -XX:MaxPermSize=256m  // Young generation options  -XX:NewSize=6g -XX:MaxNewSize=6g -XX:+UseParNewGC -XX:MaxTenuringThreshold=2 -XX:SurvivorRatio=8 -XX:+UnlockDiagnosticVMOptions -XX:ParGCCardsPerStrideChunk=32768  // Old generation  options  -XX:+UseConcMarkSweepGC -XX:CMSParallelRemarkEnabled -XX:+ParallelRefProcEnabled -XX:+CMSClassUnloadingEnabled  -XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly  // Other options  -XX:+AlwaysPreTouch -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCApplicationStoppedTime -XX:-OmitStackTraceInFastThrow |

使用这些选项，对于几千次读请求的吞吐量，应用百分之99.9的延迟降低到60ms。

## 参考：

[1] -XX:+BindGCTaskThreadsToCPUs似乎在Linux系统上不起作用，因为hotspot/src/os/linux/vm/os\_linux.cpp的distribute\_processes方法在JDK7或JDK8没有实现。  
[2] -XX:+UseGCTaskAffinity选项在JDK7和JDK8的所有平台似乎都不起作用，因为任务的affinity属性永远被设置为sentinel\_worker = (uint) -1。源码见hotspot/src/share/vm/gc\_implementation/parallelScavenge/{gcTaskManager.cpp，gcTaskThread.cpp, gcTaskManager.cpp}。  
[3] G1存在一些内存泄露的bug，可能Java7u51没有修改。这个[bug](http://bugs.java.com/bugdatabase/view_bug.do?bug_id=2212435)仅在Java 8修正了。

**十三**、**提高 Java 代码性能的各种技巧**

**Java 6,7,8 中的 String.intern – 字符串池**

这篇文章将要讨论 Java 6 中是如何实现 String.intern 方法的，以及这个方法在 Java 7 以及 Java 8 中做了哪些调整。

**字符串池**

字符串池（有名字符串标准化）是通过使用唯一的共享 String 对象来使用相同的值不同的地址表示字符串的过程。你可以使用自己定义的 Map<String, String> （根据需要使用 weak 引用或者 soft 引用）并使用 map 中的值作为标准值来实现这个目标，或者你也可以使用 JDK 提供的 String.intern()。

很多标准禁止在 Java 6 中使用 String.intern() 因为如果频繁使用池会市区控制，有很大的几率触发 OutOfMemoryException。Oracle Java 7 对字符串池做了很多改进，你可以通过以下地址进行了解 <http://bugs.sun.com/view_bug.do?bug_id=6962931>以及 <http://bugs.sun.com/view_bug.do?bug_id=6962930>

**Java 6 中的 String.intern()**

在美好的过去所有共享的 String 对象都存储在 PermGen 中 — 堆中固定大小的部分主要用于存储加载的类对象和字符串池。除了明确的共享字符串，PermGen 字符串池还包含所有程序中使用过的字符串（这里要注意是使用过的字符串，如果类或者方法从未加载或者被条用，在其中定义的任何常量都不会被加载）

Java 6 中字符串池的最大问题是它的位置 — PermGen。PermGen 的大小是固定的并且在运行时是无法扩展的。你可以使用 -XX:MaxPermSize=N 配置来调整它的大小。据我了解，对于不同的平台默认的 PermGen 大小在 32M 到 96M 之间。你可以扩展它的大小，不过大小使用都是固定的。这个限制需要你在使用 String.intern 时需要非常小心 — 你最好不要使用这个方法 intern 任何无法控制的用户输入。这是为什么在 JAVA6 中大部分使用手动管理 Map 来实现字符串池

**Java 7 中的 String.intern()**

Java 7 中 Oracle 的工程师对字符串池的逻辑做了很大的改变 — 字符串池的位置被调整到 heap 中了。这意味着你再也不会被固定的内存空间限制了。所有的字符串都保存在堆（heap）中同其他普通对象一样，这使得你在调优应用时仅需要调整堆大小。这 个改动使得我们有足够的理由让我们重新考虑在 Java 7 中使用 String.intern()。

**字符串池中的数据会被垃圾收集**

没错，在 JVM 字符串池中的所有字符串会被垃圾收集，如果这些值在应用中没有任何引用。这是用于所有版本的 Java，这意味着如果 interned 的字符串在作用域外并且没有任何引用 — 它将会从 JVM 的字符串池中被垃圾收集掉。

因为被重新定位到堆中以及会被垃圾收集，JVM 的字符串池看上去是存放字符串的合适位置，是吗？理论上是 — 违背使用的字符串会从池中收集掉，当外部输入一个字符传且池中存在时可以节省内存。看起来是一个完美的节省内存的策略？在你回答这个之前，可以肯定的是你 需要知道字符串池是如何实现的。

**在 Java 6，7，8 中 JVM 字符串池的实现**

字符串池是使用一个拥有固定容量的 HashMap 每个元素包含具有相同 hash 值的字符串列表。一些实现的细节可以从 Java bug 报告中获得 <http://bugs.sun.com/view_bug.do?bug_id=6962930>

默认的池大小是 1009 (出现在上面提及的 bug 报告的源码中，在 Java7u40 中增加了)。在 JAVA 6 早期版本中是一个常量，在随后的 java6u30 至 java6u41 中调整为可配置的。而在java 7中一开始就是可以配置的（至少在java7u02中是可以配置的）。你需要指定参数 -XX:StringTableSize=N,  N 是字符串池 Map 的大小。确保它是为性能调优而预先准备的大小。

在 Java 6 中这个参数没有太多帮助，因为你仍任被限制在固定的 PermGen 内存大小中。后续的讨论将直接忽略 Java 6

**Java 7 （直至 Java7u40）**

在 Java7 中，换句话说，你被限制在一个更大的堆内存中。这意味着你可以预先设置好 String 池的大小（这个值取决于你的应用程序需求）。通常说来，一旦程序开始内存消耗，内存都是成百兆的增长，在这种情况下，给一个拥有 100 万字符串对象的字符串池分配 8-16M 的内存看起来是比较适合的（不要使用1,000,000 作为 -XX:StringTaleSize 的值 – 它不是质数；使用 1,000,003代替）

你可能期待关于 String 在 Map 中的分配 — 可以阅读我之前关于 HashCode 方法调优的经验。

你必须设置一个更大的 -XX:StringTalbeSize 值(相比较默认的 1009 ),如果你希望更多的使用 String.intern() — 否则这个方法将很快递减到 0 （池大小）。

我没有注意到在 intern 小于 100 字符的字符串时的依赖情况（我认为在一个包含 50 个重复字符的字符串与现实数据并不相似，因此 100 个字符看上去是一个很好的测试限制）

下面是默认池大小的应用程序日志：第一列是已经 intern 的字符串数量，第二列 intern 10,000 个字符串所有的时间（秒）

0; time = 0.0 sec

50000; time = 0.03 sec

100000; time = 0.073 sec

150000; time = 0.13 sec

200000; time = 0.196 sec

250000; time = 0.279 sec

300000; time = 0.376 sec

350000; time = 0.471 sec

400000; time = 0.574 sec

450000; time = 0.666 sec

500000; time = 0.755 sec

550000; time = 0.854 sec

600000; time = 0.916 sec

650000; time = 1.006 sec

700000; time = 1.095 sec

750000; time = 1.273 sec

800000; time = 1.248 sec

850000; time = 1.446 sec

900000; time = 1.585 sec

950000; time = 1.635 sec

1000000; time = 1.913 sec

测试是在 Core i5-3317U@1.7Ghz CPU 设备上进行的。你可以看到，它成线性增长，并且在 JVM 字符串池包含一百万个字符串时，我仍然可以近似每秒 intern 5000 个字符串，这对于在内存中处理大量数据的应用程序来说太慢了。

现在，调整 -XX:StringTableSize=100003 参数来重新运行测试：

50000; time = 0.017 sec

100000; time = 0.009 sec

150000; time = 0.01 sec

200000; time = 0.009 sec

250000; time = 0.007 sec

300000; time = 0.008 sec

350000; time = 0.009 sec

400000; time = 0.009 sec

450000; time = 0.01 sec

500000; time = 0.013 sec

550000; time = 0.011 sec

600000; time = 0.012 sec

650000; time = 0.015 sec

700000; time = 0.015 sec

750000; time = 0.01 sec

800000; time = 0.01 sec

850000; time = 0.011 sec

900000; time = 0.011 sec

950000; time = 0.012 sec

1000000; time = 0.012 sec

可以看到，这时插入字符串的时间近似于常量（在 Map 的字符串列表中平均字符串个数不超过 10 个），下面是相同设置的结果，不过这次我们将向池中插入 1000 万个字符串（这意味着 Map 中的字符串列表平均包含 100 个字符串）

2000000; time = 0.024 sec

3000000; time = 0.028 sec

4000000; time = 0.053 sec

5000000; time = 0.051 sec

6000000; time = 0.034 sec

7000000; time = 0.041 sec

8000000; time = 0.089 sec

9000000; time = 0.111 sec

10000000; time = 0.123 sec

现在让我们将吃的大小增加到 100 万（精确的说是 1,000,003）

1000000; time = 0.005 sec

2000000; time = 0.005 sec

3000000; time = 0.005 sec

4000000; time = 0.004 sec

5000000; time = 0.004 sec

6000000; time = 0.009 sec

7000000; time = 0.01 sec

8000000; time = 0.009 sec

9000000; time = 0.009 sec

10000000; time = 0.009 sec

如你所看到的,时间非常平均，并且与 “0 到 100万” 的表没有太大差别。甚至在池大小足够大的情况下，我的笔记本也能每秒添加1,000,000个字符对象。

**我们还需要手工管理字符串池吗？**

现在我们需要对比 JVM 字符串池和 WeakHashMap<String, WeakReference<String>> 它可以用来模拟 JVM 字符串池。下面的方法用来替换 String.intern：

private static final WeakHashMap<String, WeakReference<String>> s\_manualCache =

new WeakHashMap<String, WeakReference<String>>( 100000 );

private static String manualIntern( final String str )

{

final WeakReference<String> cached = s\_manualCache.get( str );

if ( cached != null )

{

final String value = cached.get();

if ( value != null )

return value;

}

s\_manualCache.put( str, new WeakReference<String>( str ) );

return str;

}

下面针对手工池的相同测试：

0; manual time = 0.001 sec

50000; manual time = 0.03 sec

100000; manual time = 0.034 sec

150000; manual time = 0.008 sec

200000; manual time = 0.019 sec

250000; manual time = 0.011 sec

300000; manual time = 0.011 sec

350000; manual time = 0.008 sec

400000; manual time = 0.027 sec

450000; manual time = 0.008 sec

500000; manual time = 0.009 sec

550000; manual time = 0.008 sec

600000; manual time = 0.008 sec

650000; manual time = 0.008 sec

700000; manual time = 0.008 sec

750000; manual time = 0.011 sec

800000; manual time = 0.007 sec

850000; manual time = 0.008 sec

900000; manual time = 0.008 sec

950000; manual time = 0.008 sec

1000000; manual time = 0.008 sec

当 JVM 有足够内存时，手工编写的池提供了良好的性能。不过不幸的是，我的测试（保留 String.valueOf(0 < N < 1,000,000,000)）保留非常短的字符串，在使用 -Xmx1280M 参数时它允许我保留月为 2.5M 的这类字符串。JVM 字符串池 (size=1,000,003）从另一方面讲在 JVM 内存足够时提供了相同的性能特性，知道 JVM 字符串池包含 12.72M 的字符串并消耗掉所有内存（5倍多）。我认为，这非常值得你在你的应用中去掉所有手工字符串池。

**在 Java 7u40+ 以及 Java 8 中的 String.intern()**

Java7u40 版本扩展了字符串池的大小（这是组要的性能更新）到 60013.这个值允许你在池中包含大约 30000 个独立的字符串。通常来说，这对于需要保存的数据来说已经足够了，你可以通过 -XX:+PrintFlagsFinal JVM 参数获得这个值。

我尝试在原始发布的 Java 8 中运行相同的测试，Java 8 仍然支持 -XX:StringTableSize 参数来兼容 Java 7 特性。主要的区别在于 Java 8 中默认的池大小增加到 60013：

50000; time = 0.019 sec

100000; time = 0.009 sec

150000; time = 0.009 sec

200000; time = 0.009 sec

250000; time = 0.009 sec

300000; time = 0.009 sec

350000; time = 0.011 sec

400000; time = 0.012 sec

450000; time = 0.01 sec

500000; time = 0.013 sec

550000; time = 0.013 sec

600000; time = 0.014 sec

650000; time = 0.018 sec

700000; time = 0.015 sec

750000; time = 0.029 sec

800000; time = 0.018 sec

850000; time = 0.02 sec

900000; time = 0.017 sec

950000; time = 0.018 sec

1000000; time = 0.021 sec

**测试代码**

这篇文章的测试代码很简单，一个方法中循环创建并保留新字符串。你可以测量它保留 10000 个字符串所需要的时间。最好配合 -verbose:gc JVM 参数来运行这个测试，这样可以查看垃圾收集是何时以及如何发生的。另外最好使用 -Xmx 参数来执行堆的最大值。

这里有两个测试：testStringPoolGarbageCollection 将显示 JVM 字符串池被垃圾收集 — 检查垃圾收集日志消息。在 Java 6 的默认 PermGen 大小配置上，这个测试会失败，因此最好增加这个值，或者更新测试方法，或者使用 Java 7.

第二个测试显示内存中保留了多少字符串。在 Java 6 中执行需要两个不同的内存配置 比如： -Xmx128M 以及 -Xmx1280M （10 倍以上）。你可能发现这个值不会影响放入池中字符串的数量。另一方面，在 Java 7 中你能够在堆中填满你的字符串。

/\*\*

- Testing String.intern.

\*

- Run this class at least with -verbose:gc JVM parameter.

\*/

public class InternTest {

public static void main( String[] args ) {

testStringPoolGarbageCollection();

testLongLoop();

}

/\*\*

- Use this method to see where interned strings are stored

- and how many of them can you fit for the given heap size.

\*/

private static void testLongLoop()

{

test( 1000 \* 1000 \* 1000 );

//uncomment the following line to see the hand-written cache performance

//testManual( 1000 \* 1000 \* 1000 );

}

/\*\*

- Use this method to check that not used interned strings are garbage collected.

\*/

private static void testStringPoolGarbageCollection()

{

//first method call - use it as a reference

test( 1000 \* 1000 );

//we are going to clean the cache here.

System.gc();

//check the memory consumption and how long does it take to intern strings

//in the second method call.

test( 1000 \* 1000 );

}

private static void test( final int cnt )

{

final List<String> lst = new ArrayList<String>( 100 );

long start = System.currentTimeMillis();

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

{

final String str = "Very long test string, which tells you about something " +

"very-very important, definitely deserving to be interned #" + i;

//uncomment the following line to test dependency from string length

// final String str = Integer.toString( i );

lst.add( str.intern() );

if ( i % 10000 == 0 )

{

System.out.println( i + "; time = " + ( System.currentTimeMillis() - start ) / 1000.0 + " sec" );

start = System.currentTimeMillis();

}

}

System.out.println( "Total length = " + lst.size() );

}

private static final WeakHashMap<String, WeakReference<String>> s\_manualCache =

new WeakHashMap<String, WeakReference<String>>( 100000 );

private static String manualIntern( final String str )

{

final WeakReference<String> cached = s\_manualCache.get( str );

if ( cached != null )

{

final String value = cached.get();

if ( value != null )

return value;

}

s\_manualCache.put( str, new WeakReference<String>( str ) );

return str;

}

private static void testManual( final int cnt )

{

final List<String> lst = new ArrayList<String>( 100 );

long start = System.currentTimeMillis();

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

{

final String str = "Very long test string, which tells you about something " +

"very-very important, definitely deserving to be interned #" + i;

lst.add( manualIntern( str ) );

if ( i % 10000 == 0 )

{

System.out.println( i + "; manual time = " + ( System.currentTimeMillis() - start ) / 1000.0 + " sec" );

start = System.currentTimeMillis();

}

}

System.out.println( "Total length = " + lst.size() );

}

}

**总结**

* 由于 Java 6 中使用固定的内存大小（PermGen）因此不要使用 String.intern() 方法
* Java7 和 8 在堆内存中实现字符串池。这以为这字符串池的内存限制等于应用程序的内存限制。
* 在 Java 7 和 8 中使用 -XX:StringTableSize 来设置字符串池 Map 的大小。它是固定的，因为它使用 HashMap 实现。近似于你应用单独的字符串个数（你希望保留的）并且设置池的大小为最接近的质数并乘以 2 （减少碰撞的可能性）。它是的 String.intern 可以使用相同（固定）的时间并且在每次插入时消耗更小的内存(同样的任务，使用java WeakHashMap将消耗4-5倍的内存)。
* 在 Java 6 和 7（Java7u40以前） 中 -XX:StringTableSize 参数的值是 1009。Java7u40 以后这个值调整为 60013 （Java 8 中使用相同的值）
* 如果你不确定字符串池的用量，参考：-XX:+PrintStringTableStatistics JVM 参数，当你的应用挂掉时它告诉你字符串池的使用量信息。

# 十四、各种算法题

## 【程序1】菲波拉契数列

题目：古典问题：有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？     
//这是一个菲波拉契数列问题  
public class lianxi01 {  
public static void main(String[] args) {  
System.out.println("第1个月的兔子对数:    1");  
System.out.println("第2个月的兔子对数:    1");  
int f1 = 1, f2 = 1, f, M=24;  
     for(int i=3; i<=M; i++) {  
      f = f2;  
      f2 = f1 + f2;  
      f1 = f;  
      System.out.println("第" + i +"个月的兔子对数: "+f2);  
         }  
}  
}

## 【程序2】   素数

题目：判断101-200之间有多少个素数，并输出所有素数。   
程序分析：判断素数的方法：用一个数分别去除2到sqrt(这个数)，如果能被整除， 则表明此数不是素数，反之是素数。     
public class lianxi02 {  
public static void main(String[] args) {  
    int count = 0;  
    for(int i=101; i<200; i+=2) {  
     boolean b = false;  
     for(int j=2; j<=Math.sqrt(i); j++)   
     {  
        if(i % j == 0) { b = false; break; }   
         else           { b = true; }  
     }  
        if(b == true) {count ++;System.out.println(i );}  
                                  }  
    System.out.println( "素数个数是: " + count);  
}  
}

## 【程序3】   水仙花数

题目：打印出所有的 "水仙花数 "，所谓 "水仙花数 "是指一个三位数，其各位数字立方和等于该数本身。例如：153是一个 "水仙花数 "，因为153=1的三次方＋5的三次方＋3的三次方。  
public class lianxi03 {  
public static void main(String[] args) {  
     int b1, b2, b3;   
     for(int m=101; m<1000; m++) {   
      b3 = m / 100;  
      b2 = m % 100 / 10;  
      b1 = m %    10;  
      if((b3\*b3\*b3 + b2\*b2\*b2 + b1\*b1\*b1) == m) {  
      System.out.println(m+"是一个水仙花数"); }  
     }  
}  
}

## 【程序4】   分解质因数

题目：将一个正整数分解质因数。例如：输入90,打印出90=2\*3\*3\*5。     
程序分析：对n进行分解质因数，应先找到一个最小的质数k，然后按下述步骤完成：     
(1)如果这个质数恰等于n，则说明分解质因数的过程已经结束，打印出即可。     
(2)如果n <> k，但n能被k整除，则应打印出k的值，并用n除以k的商,作为新的正整数你n,重复执行第一步。     
(3)如果n不能被k整除，则用k+1作为k的值,重复执行第一步。    
import java.util.\*;  
public     class     lianxi04{   
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        System.out.print( "请键入一个正整数:     ");   
        int    n    = s.nextInt();  
        int k=2;   
        System.out.print(n + "=" );  
        while(k <= n) {  
          if(k == n) {System.out.println(n);break;}  
            else if( n % k == 0) {System.out.print(k + "\*");n = n / k; }   
                    else    k++;  
                   }  
     }  
    }

## 【程序5】

题目：有五个学生，每个学生有3门课的成绩，从键盘输入以上数据（包括学生号，姓名，三门课成绩），计算出平均成绩，把原有的数据和计算出的平均分数存放在磁盘文件 "stud "中。  
import java.io.\*;   
import java.util.\*;  
public class lianxi50 {   
public static void main(String[] args){   
   Scanner ss = new Scanner(System.in);  
   String [][] a = new String[5][6];  
   for(int i=1; i<6; i++) {  
    System.out.print("请输入第"+i+"个学生的学号：");  
    a[i-1][0] = ss.nextLine();  
    System.out.print("请输入第"+i+"个学生的姓名：");  
    a[i-1][1] = ss.nextLine();  
    for(int j=1; j<4; j++) {  
       System.out.print("请输入该学生的第"+j+"个成绩：");  
       a[i-1][j+1] = ss.nextLine();  
       }  
System.out.println("\n");  
   }  
//以下计算平均分  
float avg;  
int sum;  
for(int i=0; i<5; i++) {  
sum=0;  
   for(int j=2; j<5; j++) {  
   sum=sum+ Integer.parseInt(a[i][j]);  
      }  
   avg= (float)sum/3;  
   a[i][5]=String.valueOf(avg);  
}  
//以下写磁盘文件   
String s1;   
try {   
    File f = new File("C:\\stud");   
    if(f.exists()){   
      System.out.println("文件存在");   
      }else{   
         System.out.println("文件不存在，正在创建文件");   
          f.createNewFile();//不存在则创建   
        }   
BufferedWriter output = new BufferedWriter(new FileWriter(f));   
for(int i=0; i<5; i++) {  
for(int j=0; j<6; j++) {  
   s1=a[i][j]+"\r\n";  
   output.write(s1);      
    }  
}  
output.close();   
System.out.println("数据已写入c盘文件stud中！");  
   } catch (Exception e) {   
     e.printStackTrace();   
     }   
}  
}

## 【程序6】   最大公约数和最小公倍数

题目：输入两个正整数m和n，求其最大公约数和最小公倍数。     
/\*\*在循环中，只要除数不等于0，用较大数除以较小的数，将小的一个数作为下一轮循环的大数，取得的余数作为下一轮循环的较小的数，如此循环直到较小的数的值为0，返回较大的数，此数即为最大公约数，最小公倍数为两数之积除以最大公约数。\* /  
import java.util.\*;  
public    class     lianxi06     {   
public static void main(String[] args) {  
int     a ,b,m;  
Scanner s = new Scanner(System.in);  
System.out.print( "键入一个整数： ");   
a = s.nextInt();  
System.out.print( "再键入一个整数： ");   
b = s.nextInt();  
      deff cd = new deff();  
      m = cd.deff(a,b);  
      int n = a \* b / m;  
      System.out.println("最大公约数: " + m);  
      System.out.println("最小公倍数: " + n);  
}   
}  
class deff{  
public int deff(int x, int y) {  
     int t;  
     if(x < y) {  
      t = x;  
      x = y;  
      y = t;  
     }    
     while(y != 0) {  
      if(x == y) return x;  
      else {  
       int k = x % y;  
       x = y;  
       y = k;  
      }  
     }  
     return x;  
}  
}

## 【程序7】

题目：输入一行字符，分别统计出其中英文字母、空格、数字和其它字符的个数。

**package** WuYang;

**import** java.util.\*;

**public** **class** lianxi07 {

**public** **static** **void** main(String[] args) {

**int** abcCount=0;//英文字母个数

**int** spaceCount=0;//空格键个数

**int** numCount=0;//数字个数

**int** otherCount=0;//其他字符个数

Scanner scan = **new** Scanner(System.*in*);//扫描器接受控制台的输入信息

System.*out*.println("输入一组字符");

String str=scan.nextLine();//取出控制台的一行信息,也就是你输入的信息

**char**[] ch = str.toCharArray();//把取道的字符串变成一个char数组

**for**(**int** i=0;i<ch.length;i++){

**if**(Character.*isLetter*(ch[i])){

//判断是否字母

abcCount++;

}

**else** **if**(Character.*isDigit*(ch[i])){

//判断是否数字

numCount++;

}

**else** **if**(Character.*isSpaceChar*(ch[i])){

//判断是否空格键

spaceCount++;

}

**else**{

//以上都不是则认为是其他字符

otherCount++;

}

}

System.*out*.println("字母个数："+abcCount);

System.*out*.println("数字个数："+numCount);

System.*out*.println("空格个数："+spaceCount);

System.*out*.println("其他字符个数："+otherCount);

}

}

## 【程序8】

题目：求s=a+aa+aaa+aaaa+aa...a的值，其中a是一个数字。例如2+22+222+2222+22222(此时共有5个数相加)，几个数相加有键盘控制。     
import java.util.\*;  
public class lianxi08 {  
public static void main(String[] args) {  
     long a , b = 0, sum = 0;  
     Scanner s = new Scanner(System.in);  
     System.out.print("输入数字a的值： ");  
     a = s.nextInt();  
     System.out.print("输入相加的项数：");  
     int n = s.nextInt();  
     int i = 0;  
     while(i < n) {  
      b = b + a;  
      sum = sum + b;  
      a = a \* 10;  
      ++ i;  
     }  
      System.out.println(sum);  
}  
}

## 【程序9】   完数

题目：一个数如果恰好等于它的因子之和，这个数就称为 "完数 "。例如6=1＋2＋3.编程     找出1000以内的所有完数。     
public class lianxi09 {  
public static void main(String[] args) {  
     System.out.println("1到1000的完数有： ");  
     for(int i=1; i<1000; i++) {  
      int t = 0;  
      for(int j=1; j<= i/2; j++) {  
       if(i % j == 0) {  
        t = t + j;  
       }  
      }  
      if(t == i) {  
       System.out.print(i + "     ");  
      }  
     }  
}

}

## 【程序10】

题目：一球从100米高度自由落下，每次落地后反跳回原高度的一半；再落下，求它在 第10次落地时，共经过多少米？第10次反弹多高？   
**public** **class** lianxi10 {

**public** **static** **void** main(String[] args) {

**double** h = 100,s = 0;

**for**(**int** i=1; i<=10; i++) {

s = s + 2\*h;

h = h / 2;

}

s=s-100;

System.*out*.println("经过路程：" + s);

System.*out*.println("最后高度：" + h);

}

}

## 【程序11】

题目：有1、2、3、4四个数字，能组成多少个互不相同且一个数字中无重复数字的三位数？并把他们都输入。     
public class lianxi11 {  
public static void main(String[] args) {  
     int count = 0;  
     for(int x=1; x<5; x++) {  
      for(int y=1; y<5; y++) {  
       for(int z=1; z<5; z++) {  
        if(x != y && y != z && x != z) {  
         count ++;  
         System.out.println(x\*100 + y\*10 + z );  
        }  
       }  
      }  
     }  
     System.out.println("共有" + count + "个三位数");  
}  
}

## 【程序12】

题目：企业发放的奖金根据利润提成。利润(I)低于或等于10万元时，奖金可提10%；利润高于10万元，低于20万元时，低于10万元的部分按10%提成，高于10万元的部分，可可提成7.5%；20万到40万之间时，高于20万元的部分，可提成5%；40万到60万之间时高于40万元的部分，可提成3%；60万到100万之间时，高于60万元的部分，可提成1.5%，高于100万元时，超过100万元的部分按1%提成，从键盘输入当月利润，求应发放奖金总数？     
import java.util.\*;  
public class lianxi12 {  
public static void main(String[] args) {  
     double x = 0,y = 0;  
     System.out.print("输入当月利润（万）：");  
     Scanner s = new Scanner(System.in);  
     x = s.nextInt();  
     if(x > 0 && x <= 10) {  
     y = x \* 0.1;  
     } else if(x > 10 && x <= 20) {  
      y = 10 \* 0.1 + (x - 10) \* 0.075;  
     } else if(x > 20 && x <= 40) {  
      y = 10 \* 0.1 + 10 \* 0.075 + (x - 20) \* 0.05;  
     } else if(x > 40 && x <= 60) {  
      y = 10 \* 0.1 + 10 \* 0.075 + 20 \* 0.05 + (x - 40) \* 0.03;  
     } else if(x > 60 && x <= 100) {  
      y = 20 \* 0.175 + 20 \* 0.05 + 20 \* 0.03 + (x - 60) \* 0.015;   
     } else if(x > 100) {  
      y = 20 \* 0.175 + 40 \* 0.08 + 40 \* 0.015 + (x - 100) \* 0.01;  
     }  
     System.out.println("应该提取的奖金是 " + y + "万");  
}  
}

## 【程序13】

题目：一个整数，它加上100后是一个完全平方数，再加上168又是一个完全平方数，请问该数是多少？     
public class lianxi13 {  
public static void main(String[] args) {  
     for(int x =1; x<100000; x++) {  
      if(Math.sqrt(x+100) % 1 == 0) {  
       if(Math.sqrt(x+168) % 1 == 0) {  
        System.out.println(x + "加100是一个完全平方数，再加168又是一个完全平方数");  
       }  
      }  
     }  
}  
}  
  
/\*按题意循环应该从-100开始（整数包括正整数、负整数、零），这样会多一个满足条件的数-99。  
但是我看到大部分人解这道题目时都把题中的“整数”理解成正整数，我也就随大流了。\*/

## 【程序14】

题目：输入某年某月某日，判断这一天是这一年的第几天？     
import java.util.\*;  
public class lianxi14 {  
public static void main(String[] args) {  
     int year, month, day;  
     int days = 0;  
     int d = 0;  
     int e;  
     input fymd = new input();  
     do {  
     e = 0;  
     System.out.print("输入年：");  
     year =fymd.input();  
     System.out.print("输入月：");  
     month = fymd.input();  
     System.out.print("输入天：");  
     day = fymd.input();  
     if (year < 0 || month < 0 || month > 12 || day < 0 || day > 31) {  
     System.out.println("输入错误，请重新输入！");  
     e=1 ;   
     }  
     }while( e==1);

      for (int i=1; i <month; i++) {  
      switch (i) {  
      case 1:  
      case 3:  
      case 5:  
      case 7:  
      case 8:  
      case 10:  
      case 12:  
       days = 31;  
      break;  
      case 4:  
      case 6:  
      case 9:  
      case 11:  
       days = 30;  
      break;  
      case 2:  
       if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {  
        days = 29;  
       } else {  
        days = 28;  
       }  
       break;  
      }  
      d += days;  
      }  
     System.out.println(year + "-" + month + "-" + day + "是这年的第" + (d+day) + "天。");  
}  
}  
class input{  
public int input() {  
     int value = 0;  
     Scanner s = new Scanner(System.in);  
     value = s.nextInt();  
     return value;  
}  
}

## 【程序15】

题目：输入三个整数x,y,z，请把这三个数由小到大输出。     
import java.util.\*;  
public class lianxi15 {  
public static void main(String[] args) {  
     input fnc = new input();  
     int x=0, y=0, z=0;  
     System.out.print("输入第一个数字：");  
      x = fnc.input();  
     System.out.print("输入第二个数字：");  
      y = fnc.input();  
     System.out.print("输入第三个数字：");  
      z = fnc.input();     
    if(x > y) {  
      int t = x;  
      x = y;  
      y = t;  
     }  
    if(x > z) {  
      int t = x;  
      x = z;  
      z = t;  
     }  
    if(y > z) {  
      int t = y;  
      y = z;  
      z = t;  
     }  
    System.out.println( "三个数字由小到大排列为： "+x + " " + y + " " + z);  
}  
}  
class input{  
public int input() {  
     int value = 0;  
     Scanner s = new Scanner(System.in);  
     value = s.nextInt();  
     return value;  
}  
}

## 【程序16】

题目：计算字符串中子串出现的次数     
import java.util.\*;  
public class lianxi49 {   
public static void main(String args[]){  
Scanner s = new Scanner(System.in);  
    System.out.print("请输入字符串：");  
    String str1 = s.nextLine();  
    System.out.print("请输入子串：");  
    String str2 = s.nextLine();  
int count=0;   
if(str1.equals("")||str2.equals(""))   
   {   
   System.out.println("你没有输入字符串或子串,无法比较!");   
   System.exit(0);   
   }   
else   
   {   
    for(int i=0;i<=str1.length()-str2.length();i++)   
     {   
     if(str2.equals(str1.substring(i, str2.length()+i)))   
      //这种比法有问题，会把"aaa"看成有2个"aa"子串。   
       count++;   
       }   
System.out.println("子串在字符串中出现: "+count+" 次");   
}   
}  
}

## 【程序17】

题目：猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个     第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下     的一半零一个。到第10天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。     
public class lianxi17 {  
public static void main(String[] args) {  
     int x = 1;  
     for(int i=2; i<=10; i++) {  
      x = (x+1)\*2;  
     }  
     System.out.println("猴子第一天摘了 " + x + " 个桃子");  
}  
}

## 【程序18】

题目：两个乒乓球队进行比赛，各出三人。甲队为a,b,c三人，乙队为x,y,z三人。已抽签决定比赛名单。有人向队员打听比赛的名单。a说他不和x比，c说他不和x,z比，请编程序找出三队赛手的名单。     
public class lianxi18 {

static char[] m = { 'a', 'b', 'c' };  
static char[] n = { 'x', 'y', 'z' };

public static void main(String[] args) {

   for (int i = 0; i < m.length; i++) {  
    for (int j = 0; j < n.length; j++) {

     if (m[i] == 'a' && n[j] == 'x') {

      continue;

} else if (m[i] == 'a' && n[j] == 'y') {

      continue;

     } else if ((m[i] == 'c' && n[j] == 'x')

       || (m[i] == 'c' && n[j] == 'z')) {

      continue;

     } else if ((m[i] == 'b' && n[j] == 'z')

       || (m[i] == 'b' && n[j] == 'y')) {

      continue;

     } else

      System.out.println(m[i] + " vs " + n[j]);

    }

   }

}

}

## 【程序19】

题目：打印出如下图案（菱形）     
     \*     
   \*\*\*     
 \*\*\*\*\*     
\*\*\*\*\*\*\*     
 \*\*\*\*\*     
   \*\*\*     
    \*     
public class lianxi19 {  
public static void main(String[] args) {  
    int H = 7, W = 7;//高和宽必须是相等的奇数  
    for(int i=0; i<(H+1) / 2; i++) {  
     for(int j=0; j<W/2-i; j++) {  
      System.out.print(" ");

     }  
     for(int k=1; k<(i+1)\*2; k++) {  
      System.out.print('\*');  
     }  
     System.out.println();  
    }  
    for(int i=1; i<=H/2; i++) {  
     for(int j=1; j<=i; j++) {  
      System.out.print(" ");  
     }  
     for(int k=1; k<=W-2\*i; k++) {  
      System.out.print('\*');  
     }  
     System.out.println();  
    }  
}  
}

## 【程序20】

题目：有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前20项之和。

public class lianxi20 {  
public static void main(String[] args) {  
    int x = 2, y = 1, t;  
    double sum = 0;  
    for(int i=1; i<=20; i++) {  
     sum = sum + (double)x / y;  
     t = y;  
     y = x;  
     x = y + t;  
     }  
System.out.println("前20项相加之和是： " + sum);  
}  
}

## 【程序21】

题目：求1+2!+3!+...+20!的和     
public class lianxi21 {  
public static void main(String[] args) {  
    long sum = 0;   
    long fac = 1;  
    for(int i=1; i<=20; i++) {  
     fac = fac \* i;  
     sum += fac;  
    }  
    System.out.println(sum);  
}

}

## 【程序22】

题目：利用递归方法求5!。     
public class lianxi22 {  
public static void main(String[] args) {  
       int n = 5;  
    rec fr = new rec();  
    System.out.println(n+"! = "+fr.rec(n));  
}  
}  
class rec{  
public long rec(int n) {  
    long value = 0 ;  
    if(n ==1 ) {  
     value = 1;  
    } else   {  
     value = n \* rec(n-1);  
    }  
    return value;  
}  
}

## 【程序23】

题目：有5个人坐在一起，问第五个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第三个人，又说比第2人大两岁。问第2个人，说比第一个人大两岁。最后问第一个人，他说是10岁。请问第五个人多大？

public class lianxi23 {  
public static void main(String[] args) {  
    int age = 10;  
     for(int i=2; i<=5; i++) {  
     age =age+2;  
    }  
    System.out.println(age);  
}  
}

## 【程序24】

题目：给一个不多于5位的正整数，要求：一、求它是几位数，二、逆序打印出各位数字。     
//使用了长整型最多输入18位  
import java.util.\*;  
public class lianxi24 {  
public static void main(String[] args) {  
   Scanner s = new Scanner(System.in);  
   System.out.print("请输入一个正整数：");  
   long a = s.nextLong();  
   String ss = Long.toString(a);  
    char[] ch = ss.toCharArray();  
    int j=ch.length;  
    System.out.println(a + "是一个"+ j +"位数。");  
    System.out.print("按逆序输出是：");  
    for(int i=j-1; i>=0; i--) {  
    System.out.print(ch[i]);  
   }  
   }  
   }

## 【程序25】

题目：一个5位数，判断它是不是回文数。即12321是回文数，个位与万位相同，十位与千位相同。     
import java.util.\*;  
public class lianxi25 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    int a;  
    do{  
     System.out.print("请输入一个5位正整数：");  
      a = s.nextInt();  
      }while(a<10000||a>99999);  
     String ss =String.valueOf(a);  
     char[] ch = ss.toCharArray();  
     if(ch[0]==ch[4]&&ch[1]==ch[3]){  
     System.out.println("这是一个回文数");}  
     else {System.out.println("这不是一个回文数");}  
    }  
    }  
//这个更好，不限位数  
import java.util.\*;  
public class lianxi25a {  
public static void main(String[] args) {  
   Scanner s = new Scanner(System.in);  
   boolean is =true;  
   System.out.print("请输入一个正整数：");  
   long a = s.nextLong();  
   String ss = Long.toString(a);  
   char[] ch = ss.toCharArray();  
   int j=ch.length;  
   for(int i=0; i<j/2; i++) {  
   if(ch[i]!=ch[j-i-1]){is=false;}  
   }  
   if(is==true){System.out.println("这是一个回文数");}  
     else {System.out.println("这不是一个回文数");}  
    }  
**}**

## 【程序26】

题目：请输入星期几的第一个字母来判断一下是星期几，如果第一个字母一样，则继续   判断第二个字母。     
import java.util.\*;  
public class lianxi26 {  
public static void main(String[] args) {  
    getChar tw = new getChar();  
    System.out.println("请输入星期的第一个大写字母：");  
    char ch = tw.getChar();  
    switch(ch) {  
     case 'M':   
      System.out.println("Monday");  
      break;  
     case 'W':   
      System.out.println("Wednesday");  
      break;  
     case 'F':  
      System.out.println("Friday");  
      break;  
     case 'T': {  
      System.out.println("请输入星期的第二个字母：");  
      char ch2 = tw.getChar();  
      if(ch2 == 'U') {System.out.println("Tuesday"); }  
      else if(ch2 == 'H') {System.out.println("Thursday"); }  
      else {System.out.println("无此写法！");  
       }  
     };   
      break;  
     case 'S': {  
       System.out.println("请输入星期的第二个字母：");  
      char ch2 = tw.getChar();  
      if(ch2 == 'U') {System.out.println("Sunday"); }  
       else if(ch2 == 'A') {System.out.println("Saturday"); }  
       else {System.out.println("无此写法！");  
       }  
     };  
      break;  
default:System.out.println("无此写法！");  
}  
   }  
}  
  
class getChar{  
public char getChar() {  
    Scanner s = new Scanner(System.in);  
    String str = s.nextLine();  
    char ch = str.charAt(0);  
    if(ch<'A' || ch>'Z') {  
     System.out.println("输入错误，请重新输入");  
     ch=getChar();  
    }  
    return ch;  
}  
}

## 【程序27】

题目：求100之内的素数     
//使用除sqrt(n)的方法求出的素数不包括2和3  
public class lianxi27 {  
public static void main(String[] args) {  
    boolean b =false;  
    System.out.print(2 + " ");  
    System.out.print(3 + " ");  
    for(int i=3; i<100; i+=2) {  
     for(int j=2; j<=Math.sqrt(i); j++) {  
      if(i % j == 0) {b = false;  
                      break;  
       } else{b = true;}  
     }  
   if(b == true) {System.out.print(i + " ");}  
    }  
   }  
}  
//该程序使用除1位素数得2位方法，运行效率高通用性差。  
public class lianxi27a {  
public static void main(String[] args) {  
    int[] a = new int[]{2, 3, 5, 7};  
   for(int j=0; j<4; j++)System.out.print(a[j] + " ");  
    boolean b =false;  
    for(int i=11; i<100; i+=2) {  
     for(int j=0; j<4; j++) {  
      if(i % a[j] == 0) {b = false;  
                      break;  
       } else{b = true;}  
     }  
   if(b == true) {System.out.print(i + " ");}  
    }  
   }  
}

## 【程序28】

题目：对10个数进行排序     
import java.util.\*;  
public class lianxi28 {  
public static void main(String[] args) {  
Scanner s = new Scanner(System.in);  
   int[] a = new int[10];  
   System.out.println("请输入10个整数：");  
   for(int i=0; i<10; i++) {  
    a[i] = s.nextInt();  
   }

   for(int i=0; i<10; i++) {  
    for(int j=i+1; j<10; j++) {  
     if(a[i] > a[j]) {  
      int t = a[i];  
      a[i] = a[j];  
      a[j] = t;  
     }  
    }  
   }  
   for(int i=0; i<10; i++) {  
    System.out.print(a[i] + " ");  
   }  
}  
}

## 【程序29】

题目：求一个3\*3矩阵对角线元素之和

import java.util.\*;  
public class lianxi29 {  
public static void main(String[] args) {  
   Scanner s = new Scanner(System.in);  
   int[][] a = new int[3][3];  
System.out.println("请输入9个整数：");  
   for(int i=0; i<3; i++) {  
    for(int j=0; j<3; j++) {  
     a[i][j] = s.nextInt();  
    }  
   }  
   System.out.println("输入的3 \* 3 矩阵是:");  
   for(int i=0; i<3; i++) {  
    for(int j=0; j<3; j++) {  
     System.out.print(a[i][j] + " ");  
    }  
    System.out.println();  
   }  
   int sum = 0;  
   for(int i=0; i<3; i++) {  
    for(int j=0; j<3; j++) {  
     if(i == j) {  
      sum += a[i][j];  
     }  
    }  
   }  
   System.out.println("对角线之和是：" + sum);  
}  
}

## 【程序30】

题目：有一个已经排好序的数组。现输入一个数，要求按原来的规律将它插入数组中。      
//此程序不好，没有使用折半查找插入

**import** java.util.\*;

**public** **class** lianxi30 {

**public** **static** **void** main(String[] args) {

**int**[] a = **new** **int**[]{1, 2, 6, 14, 25, 36, 37,55};

**int**[] b = **new** **int**[a.length+1];

**int** t1=0, t2 = 0;

**int** i =0;

Scanner s= **new** Scanner(System.*in*);

System.*out*.print("请输入一个整数：");

**int** num = s.nextInt();

**if**(num >= a[a.length-1]) {

b[b.length-1] = num;

**for**(i=0; i<a.length; i++) {

b[i] = a[i];

}

} **else** {

**for**(i=0; i<a.length; i++) {

**if**(num >= a[i]) {

b[i] = a[i];

} **else** {

b[i] = num;

**break**;

}

}

**for**(**int** j=i+1; j<b.length; j++) {

b[j] = a[j-1];

}

}

**for** (i = 0; i < b.length; i++) {

System.*out*.print(b[i] + " ");

}

}

}

## 【程序31】

题目：将一个数组逆序输出。     
import java.util.\*;  
public class lianxi31 {  
public static void main(String[] args) {  
   Scanner s = new Scanner(System.in);  
   int a[] = new int[20];  
System.out.println("请输入多个正整数（输入-1表示结束）：");  
   int i=0,j;   do{  
      a[i]=s.nextInt();  
      i++;  
   }while (a[i-1]!=-1);

   System.out.println("你输入的数组为：");  
   for( j=0; j<i-1; j++) {  
    System.out.print(a[j]+"   ");  
}  
   System.out.println("\n数组逆序输出为：");  
   for( j=i-2; j>=0; j=j-1) {  
    System.out.print(a[j]+"   ");  
}  
    }  
   }

## 【程序32】

题目：取一个整数a从右端开始的4～7位。     
import java.util.\*;  
public class lianxi32 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    System.out.print("请输入一个7位以上的正整数：");  
    long a = s.nextLong();  
    String ss = Long.toString(a);  
    char[] ch = ss.toCharArray();  
    int j=ch.length;  
    if (j<7){System.out.println("输入错误！");}  
    else {  
     System.out.println("截取从右端开始的4～7位是："+ch[j-7]+ch[j-6]+ch[j-5]+ch[j-4]);  
     }  
    }  
    }

## 【程序33】

题目：打印出杨辉三角形（要求打印出10行如下图）        
            1     
          1    1     
        1    2    1     
      1    3    3    1     
    1    4    6    4    1     
1    5    10    10    5    1     
…………  
public class lianxi33 {  
public static void main(String[] args) {  
    int[][] a = new int[10][10];  
   for(int i=0; i<10; i++) {  
    a[i][i] = 1;  
    a[i][0] = 1;  
   }  
   for(int i=2; i<10; i++) {  
    for(int j=1; j<i; j++) {  
     a[i][j] = a[i-1][j-1] + a[i-1][j];  
    }  
   }  
     for(int i=0; i<10; i++) {  
    for(int k=0; k<2\*(10-i)-1; k++) {  
     System.out.print(" ");  
    }  
    for(int j=0; j<=i; j++) {  
     System.out.print(a[i][j] + "   ");  
    }  
    System.out.println();  
   }  
}  
}

## 【程序34】

题目：输入3个数a,b,c，按大小顺序输出。     
import java.util.Scanner;  
public class lianxi34 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    System.out.println("请输入3个整数：");  
    int a = s.nextInt();  
    int b = s.nextInt();  
    int c = s.nextInt();  
      if(a < b) {  
     int t = a;  
     a = b;  
     b = t;  
    }  
      if(a < c) {  
     int t = a;  
     a = c;  
     c = t;  
    }  
     if(b < c) {  
     int t = b;  
     b = c;  
     c = t;  
    }  
    System.out.println("从大到小的顺序输出:");  
    System.out.println(a + " " + b + " " + c);  
}  
}

## 【程序35】

题目：输入数组，最大的与第一个元素交换，最小的与最后一个元素交换，输出数组。

**import** java.util.\*;

**public** **class** lianxi35 {

**public** **static** **void** main(String[] args) {

**int** N = 8;

**int**[] a = **new** **int**[N];// 创建一个八个元素的数组

Scanner s = **new** Scanner(System.*in*);

**int** idx1 = 0, idx2 = 0;

System.*out*.println("请输入8个整数：");

**for** (**int** i = 0; i < N; i++) {

a[i] = s.nextInt();

}

// 获得输入的八个数字

System.*out*.println("你输入的数组为：");

**for** (**int** i = 0; i < N; i++) {

System.*out*.print(a[i] + " ");

}

// 输出输入的数组

**int** max = a[0], min = a[0];

**for** (**int** i = 0; i < N; i++) {

**if** (a[i] > max) {

max = a[i];

idx1 = i;

}

// 找出最大的数和其下标

**if** (a[i] < min) {

min = a[i];

idx2 = i;

}// 找出最大的数和其下标

}

**if** (idx1 != 0) {

**int** temp = a[0];

a[0] = a[idx1];

a[idx1] = temp;

}// 最大的数和第一个数交换位置

**if** (idx2 != N - 1) {

**int** temp = a[N - 1];

a[N - 1] = a[idx2];

a[idx2] = temp;

}// 最小的数和最后一个数交换位置

System.*out*.println("\n交换后的数组为：");

**for** (**int** i = 0; i < N; i++) {

System.*out*.print(a[i] + " ");

}

}

}

## 【程序36】

题目：有n个整数，使其前面各数顺序向后移m个位置，最后m个数变成最前面的m个数     
import java.util.Scanner;  
public class lianxi36 {  
public static void main(String[] args) {  
   int N =10;  
   int[] a = new int[N];  
   Scanner s = new Scanner(System.in);  
   System.out.println("请输入10个整数：");  
   for(int i=0; i<N; i++) {  
    a[i] = s.nextInt();  
   }  
   System.out.print("你输入的数组为：");  
   for(int i=0; i<N; i++) {  
     System.out.print(a[i] + " ");  
   }  
   System.out.print("\n请输入向后移动的位数：");  
   int m = s.nextInt();  
   int[] b = new int[m];  
   for(int i=0; i<m; i++) {  
    b[i] = a[N-m+i];  
   }  
   for(int i=N-1; i>=m; i--) {  
   a[i] = a[i-m];  
   }  
   for(int i=0; i<m; i++) {  
    a[i] = b[i];  
   }  
System.out.print("位移后的数组是：");  
   for(int i=0; i<N; i++) {  
    System.out.print(a[i] + " ");  
   }  
}  
}

## 【程序37】

题目：有n个人围成一圈，顺序排号。从第一个人开始报数（从1到3报数），凡报到3的人退出圈子，问最后留下的是原来第几号的那位。     
import java.util.Scanner;  
public class lianxi37 {  
public static void main(String[] args) {  
   Scanner s = new Scanner(System.in);  
   System.out.print("请输入排成一圈的人数：");  
   int n = s.nextInt();  
   boolean[] arr = new boolean[n];  
   for(int i=0; i<arr.length; i++) {  
    arr[i] = true;  
   }//数组赋值都是true  
   int leftCount = n;  
   int countNum = 0;  
   int index = 0;  
   while(leftCount > 1) {  
    if(arr[index] == true) {  
     countNum ++;   
     if(countNum == 3) {  
      countNum =0;  
      arr[index] = false;  
      leftCount --;  
     }  
    }  
     index ++;  
     if(index == n) {  
     index = 0;  
    }  
   }  
    for(int i=0; i<n; i++) {  
    if(arr[i] == true) {  
     System.out.println("原排在第"+(i+1)+"位的人留下了。");  
    }  
   }  
}  
}

## 【程序38】

题目：写一个函数，求一个字符串的长度，在main函数中输入字符串，并输出其长度。     
/\*………………  
\*……题目意思似乎不能用length()函数     \*/  
import java.util.\*;  
public class lianxi38 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    System.out.println("请输入一个字符串：");  
    String str = s.nextLine();  
     System.out.println("字符串的长度是："+str.length());  
    }  
    }

## 【程序39】

题目：编写一个函数，输入n为偶数时，调用函数求1/2+1/4+...+1/n,当输入n为奇数时，调用函数1/1+1/3+...+1/n(利用指针函数)     
//没有利用指针函数  
import java.util.\*;  
public class lianxi39 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    System.out.print("请输入一个正整数 n= ");  
    int n = s.nextInt();  
    System.out.println("相应数列的和为：" + sum(n));  
   }  
public static double sum(int n) {  
    double res = 0;  
    if(n % 2 == 0) {  
     for(int i=2; i<=n; i+=2) {  
      res += (double)1 / i;  
     }   
    } else {  
     for(int i=1; i<=n; i+=2) {  
      res += (double)1 / i ;  
     }  
    }  
    return res;  
}  
}

## 【程序40】

题目：字符串排序。     
public class lianxi40 {  
public static void main(String[] args) {  
   int N=5;  
   String temp = null;  
   String[] s = new String[N];  
   s[0] = "matter";  
   s[1] = "state";  
   s[2] = "solid";  
   s[3] = "liquid";  
   s[4] = "gas";  
   for(int i=0; i<N; i++) {  
    for(int j=i+1; j<N; j++) {  
     if(compare(s[i], s[j]) == false) {  
      temp = s[i];  
      s[i] = s[j];  
      s[j] = temp;  
     }  
    }  
   }  
    for(int i=0; i<N; i++) {  
    System.out.println(s[i]);  
   }  
}  
static boolean compare(String s1, String s2) {  
   boolean result = true;  
   for(int i=0; i<s1.length() && i<s2.length(); i++) {  
    if(s1.charAt(i) > s2.charAt(i)) {  
     result = false;  
     break;  
    } else if(s1.charAt(i) <s2.charAt(i)) {  
     result = true;  
     break;  
    } else {  
     if(s1.length() < s2.length()) {  
      result = true;  
     } else {  
      result = false;  
     }  
    }  
   }  
   return result;  
}  
}

## 【程序41】

**题目：海滩上有一堆桃子，五只猴子来分。第一只猴子把这堆桃子凭据分为五份，多了一个，这只猴子把多的一个扔入海中，拿走了**一份。第二只猴子把剩下的桃子又平均分成五份，又多了一个，它同样把多的一个扔入海中，拿走了一份，第三、第四、第五只猴子都是这样做的，问海滩上原来最少有多少个桃子？     
public class lianxi41 {   
public static void main (String[] args) {   
int i,m,j=0,k,count;   
for(i=4;i<10000;i+=4)   
   { count=0;   
     m=i;   
     for(k=0;k<5;k++)   
        {   
         j=i/4\*5+1;   
         i=j;   
         if(j%4==0)   
            count++;   
            else break;   
       }   
    i=m;   
if(count==4)   
{System.out.println("原有桃子 "+j+" 个");   
break;}   
}   
}   
}

## 【程序42】

题目：809\*??=800\*??+9\*??+1    其中??代表的两位数,8\*??的结果为两位数，9\*??的结果为3位数。求??代表的两位数，及809\*??后的结果。     
//题目错了！809x=800x+9x+1 这样的方程无解。去掉那个1就有解了。  
public class lianxi42 {   
public static void main (String[] args) {   
int a=809,b,i;  
for(i=10;i<13;i++)  
{b=i\*a ;  
if(8\*i<100&&9\*i>=100)  
System.out.println ("809\*"+i+"="+"800\*"+i+"+"+"9\*"+i+"="+b);}  
}  
}

## 【程序43】

题目：求0—7所能组成的奇数个数。     
//组成1位数是4个。  
//组成2位数是7\*4个。  
//组成3位数是7\*8\*4个。  
//组成4位数是7\*8\*8\*4个。  
//......  
public class lianxi43 {   
public static void main (String[] args) {   
int sum=4;  
int j;  
System.out.println("组成1位数是 "+sum+" 个");  
sum=sum\*7;  
System.out.println("组成2位数是 "+sum+" 个");  
for(j=3;j<=9;j++){  
sum=sum\*8;   
System.out.println("组成"+j+"位数是 "+sum+" 个");  
}  
}  
}

## 【程序44】

题目：一个偶数总能表示为两个素数之和。     
//由于用除sqrt(n)的方法求出的素数不包括2和3，  
//因此在判断是否是素数程序中人为添加了一个3。  
import java.util.\*;  
public class lianxi44 {  
public static void main(String[] args) {  
Scanner s = new Scanner(System.in);  
int n,i;  
do{  
     System.out.print("请输入一个大于等于6的偶数：");  
     n = s.nextInt();  
    } while(n<6||n%2!=0);   //判断输入是否是>=6偶数,不是,重新输入  
fun fc = new fun();  
    for(i=2;i<=n/2;i++){  
    if((fc.fun(i))==1&&(fc.fun(n-i)==1))   
    {int j=n-i;  
     System.out.println(n+" = "+i+" + "+j);  
     } //输出所有可能的素数对  
   }  
}  
}  
class fun{  
public int fun (int a)    //判断是否是素数的函数  
{  
int i,flag=0;  
if(a==3){flag=1;return(flag);}  
for(i=2;i<=Math.sqrt(a);i++){  
   if(a%i==0) {flag=0;break;}  
      else flag=1;}  
return (flag) ;//不是素数,返回0,是素数,返回1  
}  
}  
//解法二  
import java.util.\*;  
public class lianxi44 {  
public static void main(String[] args) {  
Scanner s = new Scanner(System.in);  
int n;  
do{  
     System.out.print("请输入一个大于等于6的偶数：");  
     n = s.nextInt();  
    } while(n<6||n%2!=0);   //判断输入是否是>=6偶数,不是,重新输入

for(int i=3;i<=n/2;i+=2){  
    if(fun(i)&&fun(n-i)) {  
      System.out.println(n+" = "+i+" + "+(n-i));  
      } //输出所有可能的素数对  
   }  
}  
static boolean fun (int a){    //判断是否是素数的函数  
boolean flag=false;  
if(a==3){flag=true;return(flag);}  
for(int i=2;i<=Math.sqrt(a);i++){  
   if(a%i==0) {flag=false;break;}  
      else flag=true;}  
return (flag) ;  
}  
}

## 【程序45】

题目：判断一个素数能被几个9整除     
//题目错了吧？能被9整除的就不是素数了！所以改成整数了。  
import java.util.\*;  
public class lianxi45 {   
public static void main (String[] args) {   
   Scanner s = new Scanner(System.in);  
   System.out.print("请输入一个整数：");  
    int num = s.nextInt();  
    int   tmp = num;  
    int count = 0;   
       for(int i = 0 ; tmp%9 == 0 ;){  
           tmp = tmp/9;  
            count ++;  
          }  
     System.out.println(num+" 能够被 "+count+" 个9整除。");  
     }  
}

## 【程序46】

题目：两个字符串连接程序     
import java.util.\*;  
public class lianxi46 {  
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    System.out.print("请输入一个字符串：");  
    String str1 = s.nextLine();  
    System.out.print("请再输入一个字符串：");  
    String str2 = s.nextLine();  
    String str = str1+str2;  
    System.out.println("连接后的字符串是："+str);  
    }  
    }

## 【程序47】

题目：读取7个数（1—50）的整数值，每读取一个值，程序打印出该值个数的＊。     
import java.util.\*;  
public class lianxi47 {  
public static void main(String[] args) {  
Scanner s = new Scanner(System.in);  
int n=1,num;  
while(n<=7){  
         do{  
          System.out.print("请输入一个1--50之间的整数：");  
             num= s.nextInt();  
           }while(num<1||num>50);  
      for(int i=1;i<=num;i++)  
      {System.out.print("\*");  
      }  
System.out.println();  
n ++;  
}  
}  
}

## 【程序48】

题目：某个公司采用公用电话传递数据，数据是四位的整数，在传递过程中是加密的，加密规则如下：每位数字都加上5,然后用和除以10的余数代替该数字，再将第一位和第四位交换，第二位和第三位交换。     
import java.util.\*;  
public class lianxi48   {   
public static void main(String args[]) {   
Scanner s = new Scanner(System.in);  
int num=0,temp;  
do{  
   System.out.print("请输入一个4位正整数：");  
      num = s.nextInt();  
     }while (num<1000||num>9999);   
int a[]=new int[4];   
a[0] = num/1000; //取千位的数字   
a[1] = (num/100)%10; //取百位的数字   
a[2] = (num/10)%10; //取十位的数字   
a[3] = num%10; //取个位的数字   
for(int j=0;j<4;j++)   
{   
a[j]+=5;   
a[j]%=10;   
}   
for(int j=0;j<=1;j++)   
    {   
    temp = a[j];   
    a[j] = a[3-j];   
    a[3-j] =temp;   
    }   
System.out.print("加密后的数字为：");   
for(int j=0;j<4;j++)   
System.out.print(a[j]);   
}   
}

# 十五、JVM相关概念

 本文首先介绍一下Java虚拟机的生存周期，然后大致介绍JVM的体系结构，最后对体系结构中的各个部分进行详细介绍。

（  首先这里澄清两个概念：JVM实例和JVM执行引擎实例，JVM实例对应了一个独立运行的java程序，而JVM执行引擎实例则对应了属于用户运行程序的线程；也就是JVM实例是进程级别，而执行引擎是线程级别的。）

## 一、 JVM的生命周期

### JVM实例的诞生：

当启动一个Java程序时，一个JVM实例就产生了，任何一个拥有public static void main(String[] args)函数的class都可以作为JVM实例运行的起点，既然如此，那么JVM如何知道是运行class A的main而不是运行class B的main呢？这就需要显式的告诉JVM类名，也就是我们平时运行java程序命令的由来，如java classA hello world,这里java是告诉os运行Sun java 2 SDK的java虚拟机，而classA则指出了运行JVM所需要的类名。

JVM实例的运行：main（）作为该程序初始线程的起点，任何其他线程均由该线程启动。JVM内部有两种线程：守护线程和非守护线程，main（）属于非守护线程，守护线程通常由JVM自己使用，java程序也可以标明自己创建的线程是守护线程。

JVM实例的消亡：当程序中的所有非守护线程都终止时，JVM才退出；若安全管理器允许，程序也可以使用Runtime类或者System.exit()来退出。

## JVM的体系结构

  粗略分来，JVM的内部体系结构分为三部分，分别是：类装载器（ClassLoader）子系统，运行时数据区，和执行引擎。

 下面将先介绍类装载器，然后是执行引擎，最后是运行时数据区

### 1，类装载器

顾名思义，就是用来装载.class文件的。JVM的两种类装载器包括：启动类装载器和用户自定义类装载器，启动类装载器是JVM实现的一部分，用户自定义类装载器则是Java程序的一部分，必须是ClassLoader类的子类。（下面所述情况是针对Sun JDK1.2）

动类装载器：只在系统类(java API的类文件)的安装路径查找要装入的类

用户自定义类装载器：

系统类装载器：在JVM启动时创建，用来在CLASSPATH目录下查找要装入的类

其他用户自定义类装载器：这里有必要先说一下ClassLoader类的几个方法，了解它们对于了解自定义类装载器如何装载.class文件至关重要。

protected final Class defineClass(String name, byte data[], int offset, int length)

protected final Class defineClass(String name, byte data[], int offset, int length, ProtectionDomain protectionDomain);

protected final Class findSystemClass(String name)

protected final void resolveClass(Class c)

defineClass用来将二进制class文件（新类型）导入到方法区,也就是这里指的类是用户自定义的类（也就是负责装载类）

findSystemClass通过类型的全限定名，先通过系统类装载器或者启动类装载器来装载，并返回Class对象。

ResolveClass: 让类装载器进行连接动作（包括验证，分配内存初始化，将类型中的符号引用解析为直接引用），这里涉及到java命名空间的问题，JVM保证被一个类装载器装载的类所引用的所有类都被这个类装载器装载，同一个类装载器装载的类之间可以相互访问，但是不同类装载器装载的类看不见对方，从而实现了有效的屏蔽。

### 2，执行引擎：它或者在执行字节码，或者执行本地方法

   要说执行引擎，就不得不的指令集，每一条指令包含一个单字节的操作码，后面跟0个或者多个操作数。

（一）指令集以栈为设计中心，而非以寄存器为中心

这种指令集设计如何满足Java体系的要求：

平台无关性：以栈为中心使得在只有很少register的机器上实现java更便利

compiler一般采用stack向连接优化器传递编译的中间结果，若指令集以stack为基础，则有利于运行时进行的优化工作与执行即时编译或者自适应优化的执行引擎结合，通俗的说就是使编译和运行用的数据结构统一，更有利于优化的开展。

网络移动性：class文件的紧凑性。

安全性：指令集中绝大部分操作码都指明了操作的类型。（在装载的时候使用数据流分析期进行一次性验证，而非在执行每条指令的时候进行验证，有利于提高执行速度）。

1. 执行技术

主要的执行技术有:解释，即时编译，自适应优化、芯片级直接执行

其中解释属于第一代JVM，即时编译JIT属于第二代JVM，自适应优化（目前Sun的HotspotJVM采用这种技术）则吸取第一代JVM和第二代JVM的经验，采用两者结合的方式

自适应优化：开始对所有的代码都采取解释执行的方式，并监视代码执行情况，然后对那些经常调用的方法启动一个后台线程，将其编译为本地代码，并进行仔细优化。若方法不再频繁使用，则取消编译过的代码，仍对其进行解释执行。

3，运行时数据区：主要包括：方法区，堆，java栈，PC寄存器，本地方法栈

1. 方法区和堆由所有线程共享   
    堆：存放所有程序在运行时创建的对象  
    方法区：当JVM的类装载器加载.class文件，并进行解析，把解析的类型信息放入方法区。
2. Java栈和PC寄存器由线程独享，在新线程创建时间里
3. 本地方法栈： 存储本地方法调用的状态

上边总体介绍了运行时数据区的主要内容，下边进行详细介绍，要介绍数据区，就不得不说明JVM中的数据类型。

JVM中的数据类型：JVM中基本的数据单元是word,而word的长度由JVM具体的实现者来决定

数据类型包括基本类型和引用类型，

1. 基本类型

包括：数值类型(包括除boolean外的所有的java基本数据类型)，boolean（在JVM中使用int来表示，0表示false，其他int值均表示true）和returnAddress（JVM的内部类型，用来实现finally子句）。

1. 引用类型包括：数组类型，类类型，接口类型

前边讲述了JVM中数据的表示，下面让我们输入到JVM的数据区

首先来看方法区：

上边已经提到，方法区主要用来存储JVM从class文件中提取的类型信息，那么类型信息是如何存储的呢？众所周知，Java使用的是大端序（big—endian:即低字节的数据存储在高位内存上，如对于1234，12是高位数据，34为低位数据，则java中的存储格式应该为12存在内存的低地址，34存在内存的高地址，x86中的存储格式与之相反）来存储数据，这实际上是在class文件中数据的存储格式，但是当数据倒入到方法区中时，JVM可以以任何方式来存储它。

类型信息：包括class的全限定名，class的直接父类，类类型还是接口类型，类的修饰符（public,等），所有直接父接口的列表，Class对象提供了访问这些信息的窗口（可通过Class.forName(“”)或instance.getClass()获得），下面是Class的方法，相信大家看了会恍然大悟，（原来如此J）

getName(), getSuperClass(), isInterface(), getInterfaces(), getClassLoader();

static变量作为类型信息的一部分保存

指向ClassLoader类的引用：在动态连接时装载该类中引用的其他类

指向Class类的引用：必然的，上边已述

该类型的常量池：包括直接常量（String，integer和float point常量）以及对其他类型、字段和方法的符号引用（注意：这里的常量池并不是普通意义上的存储常量的地方，这些符号引用可能是我们在编程中所接触到的变量），由于这些符号引用，使得常量池成为java程序动态连接中至关重要的部分

字段信息：普通意义上的类型中声明的字段

方法信息：类型中各个方法的信息

编译期常量：指用final声明或者用编译时已知的值初始化的类变量

class将所有的常量复制至其常量池或者其字节码流中。

方法表：一个数组，包括所有它的实例可能调用的实例方法的直接引用（包括从父类中继承来的）

除此之外，若某个类不是抽象和本地的，还要保存方法的字节码，操作数栈和该方法的栈帧，异常表。

举例：

class Lava{  
  private int speed = 5;  
  void flow(){}  
}  
  
  
  
class Volcano{  
  public static void main(String[] args){  
 Lava lava = new Lava();  
 lava.flow();

}

}  
运行命令java Volcano;  
（1）JVM找到Volcano.class倒入，并提取相应的类型信息到方法区。通过执行方法区中的字节码，JVM执行main（）方法，（执行时会一直保存指向Vocano类的常量池的指针）

1. Main（）中第一条指令告诉JVM需为列在常量池第一项的类分配内存（此处再次说明了常量池并非只存储常量信息），然后JVM找到常量池的第一项，发现是对Lava类的符号引用，则检查方法区，看Lava类是否装载，结果是还未装载，则查找“Lava.class”,将类型信息写入方法区，并将方法区Lava类信息的指针来替换Volcano原常量池中的符号引用，即用直接引用来替换符号引用。

（3 )JVM看到new关键字，准备为Lava分配内存，根据Volcano的常量池的第一项找到Lava在方法区的位置，并分析需要多少对空间，确定后，在堆上分配空间，并将speed变量初始为0，并将lava对象的引用压到栈中

（4调用lava的flow（）方法

好了，大致了解了方法区的内容后，让我们来看看堆

java对象的堆实现：

java对象主要由实例变量（包括自己所属的类和其父类声明的）以及指向方法区中类数据的指针，指向方法表的指针，对象锁（非必需）， 等待集合（非必需），GC相关的数据（非必需）（主要视GC算法而定，如对于标记并清除算法，需要标记对象是否被引用，以及是否已调用finalize（）方法）。

那么为什么java对象中要有指向类数据的指针呢？我们从几个方面来考虑

首先：当程序中将一个对象引用转为另一个类型时，如何检查转换是否允许？需用到类数据

其次：动态绑定时，并不是需要引用类型，而是需要运行时类型

这里的迷惑是：为什么类数据中保存的是实际类型，而非引用类型？这个问题先留下来，我想在后续的读书笔记中应该能明白

指向方法表的指针：这里和C++的VTBL是类似的，有利于提高方法调用的效率

对象锁：用来实现多个线程对共享数据的互斥访问

等待集合：用来让多个线程为完成共同目标而协调功过。（注意Object类中的wait(),notify(),notifyAll()方法）。

Java数组的堆实现：数组也拥有一个和他们的类相关联的Class实例，具有相同dimension和type的数组是同一个类的实例。数组类名的表示：如[[Ljava/lang/Object 表示Object[][]，[I表示int[]，[[[B表示byte[][][]

至此，堆已大致介绍完毕，下面来介绍程序计数器和java栈

程序计数器：为每个线程独有，在线程启动时创建，

  若thread执行java方法，则PC保存下一条执行指令的地址。

  若thread执行native方法，则Pc的值为undefined

Java栈：java栈以帧为单位保存线程的运行状态，java栈只有两种操作，帧的压栈和出栈。

每个帧代表一个方法，java方法有两种返回方式，return和抛出异常，两种方式都会导致该方法对应的帧出栈和释放内存。

帧的组成：局部变量区（包括方法参数和局部变量，对于instance方法，还要首先保存this类型，其中方法参数按照声明顺序严格放置，局部变量可以任意放置），操作数栈，帧数据区（用来帮助支持常量池的解析，正常方法返回和异常处理）。

本地方法栈：依赖于本地方法的实现，如某个JVM实现的本地方法借口使用C连接模型，则本地方法栈就是C栈，可以说某线程在调用本地方法时，就进入了一个不受JVM限制的领域，也就是JVM可以利用本地方法来动态扩展本身。

# 十六、反射相关

## 反射相关概念

运行时类型识别(Run-time Type Identification, RTTI)主要有两种方式，一种是我们在编译时和运行时已经知道了所有的类型，另外一种是功能强大的“反射”机制。

要理解RTTI在Java中的工作原理，首先必须知道类型信息在运行时是如何表示的，这项工作是由“Class对象”完成的，它包含了与类有关的信息。类是程序的重要组成部分，每个类都有一个Class对象，每当编写并编译了一个新类就会产生一个Class对象，它被保存在一个同名的.class文件中。在运行时，当我们想生成这个类的对象时，运行这个程序的Java虚拟机(JVM)会确认这个类的Class对象是否已经加载，如果尚未加载，JVM就会根据类名查找.class文件，并将其载入，一旦这个类的Class对象被载入内存，它就被用来创建这个类的所有对象。一般的RTTI形式包括三种：

1. 传统的类型转换。如“(Apple)Fruit”，由RTTI确保类型转换的正确性，如果执行了一个错误的类型转换，就会抛出一个ClassCastException异常。

2. 通过Class对象来获取对象的类型。如

Class c = Class.forName(“Apple”);

Object o = c.newInstance();

3. 通过关键字instanceof或Class.isInstance()方法来确定对象是否属于某个特定类型的实例，准确的说，应该是instanceof / Class.isInstance()可以用来确定对象是否属于某个特定类及其所有基类的实例，这和equals() / ==不一样，它们用来比较两个对象是否属于同一个类的实例，没有考虑继承关系。

如果不知道某个对象的类型，可以通过RTTI来获取，但前提是这个类型在编译时必须已知，这样才能使用RTTI来识别。即在编译时，编译器必须知道所有通过RTTI来处理的类。

使用反射机制可以不受这个限制，它主要应用于两种情况，第一个是“基于构件的编程”，在这种编程方式中，将使用某种基于快速应用开发(RAD)的应用构建工具来构建项目。这是现在最常见的可视化编程方法，通过代表不同组件的图标拖动到图板上来创建程序，然后设置构件的属性值来配置它们。这种配置要求构件都是可实例化的，并且要暴露其部分信息，使得程序员可以读取和设置构件的值。当处理GUI时间的构件时还必须暴露相关方法的细细，以便RAD环境帮助程序员覆盖这些处理事件的方法。在这里，就要用到反射的机制来检查可用的方法并返回方法名。Java通过JavaBeans提供了基于构件的编程架构。

第二种情况，在运行时获取类的信息的另外一个动机，就是希望能够提供在跨网络的远程平台上创建和运行对象的能力。这被成为远程调用(RMI)，它允许一个Java程序将对象分步在多台机器上，这种分步能力将帮助开发人员执行一些需要进行大量计算的任务，充分利用计算机资源，提高运行速度。

Class支持反射，java.lang.reflect中包含了Field/Method/Constructor类，每个类都实现了Member接口。这些类型的对象都是由JVM在运行时创建的，用来表示未知类里对应的成员。如可以用Constructor类创建新的对象，用get()和set()方法读取和修改与Field对象关联的字段，用invoke()方法调用与Method对象关联的方法。同时，还可以调用getFields()、getMethods()、getConstructors()等方法来返回表示字段、方法以及构造器的对象数组。这样，未知的对象的类信息在运行时就能被完全确定下来，而在编译时不需要知道任何信息。

另外，RTTI有时能解决效率问题。当程序中使用多态给程序的运行带来负担的时候，可以使用RTTI编写一段代码来提高效率。

## 反射机制

### 摘要

Reflection 是Java被视为动态（或准动态）语言的一个关键性质。这个机制允许程序在运行时透过Reflection APIs取得任何一个已知名称的class的内部信息，包括其modifiers（诸如public, static 等等）、superclass（例如Object）、实现之interfaces（例如Cloneable），也包括fields和methods的所有信息，并可于运行时改变fields内容或唤起methods。本文借由实例，大面积示范Reflection APIs。

### 关键词：

Introspection（内省、内观）

Reflection（反射）

有时候我们说某个语言具有很强的动态性，有时候我们会区分动态和静态的不同技术与作法。我们朗朗上口动态绑定（dynamic binding）、动态链接（dynamic linking）、动态加载（dynamic loading）等。然而“动态”一词其实没有绝对而普遍适用的严格定义，有时候甚至像对象导向当初被导入编程领域一样，一人一把号，各吹各的调。

一般而言，开发者社群说到动态语言，大致认同的一个定义是：“程序运行时，允许改变程序结构或变量类型，这种语言称为动态语言”。从这个观点看，Perl，Python，Ruby是动态语言，C++，Java，C#不是动态语言。

尽管在这样的定义与分类下Java不是动态语言，它却有着一个非常突出的动态相关机制：**Reflection**。这个字的意思是“反射、映象、倒影”，用在Java身上指的是我们可以于运行时加载、探知、使用编译期间完全未知的classes。换句话说，Java程序可以加载一个运行时才得知名称的class，获悉其完整构造（但不包括methods定义），并生成其对象实体、或对其fields设值、或唤起其methods1。这种“看透class”的能力（the ability of the program to examine itself）被称为introspection（内省、内观、反省）。Reflection和introspection是常被并提的两个术语。

Java如何能够做出上述的动态特性呢？这是一个深远话题，本文对此只简单介绍一些概念。整个篇幅最主要还是介绍Reflection APIs，也就是让读者知道如何探索class的结构、如何对某个“运行时才获知名称的class”生成一份实体、为其fields设值、调用其methods。本文将谈到java.lang.Class，以及java.lang.reflect中的Method、Field、Constructor等等classes。

### “Class”class

众所周知Java有个Object class，是所有Java classes的继承根源，其内声明了数个应该在所有Java class中被改写的methods：hashCode()、equals()、clone()、toString()、getClass()等。其中getClass()返回一个Class object。

Class class十分特殊。它和一般classes一样继承自Object，其实体用以表达Java程序运行时的classes和interfaces，也用来表达enum、array、primitive Java types（boolean, byte, char, short, int, long, float, double）以及关键词void。当一个class被加载，或当加载器（class loader）的defineClass()被JVM调用，JVM 便自动产生一个Class object。如果您想借由“修改Java标准库源码”来观察Class object的实际生成时机（例如在Class的constructor内添加一个println()），不能够！因为Class并没有public constructor（见图1）。本文最后我会拨一小块篇幅顺带谈谈Java标准库源码的改动办法。

Class是Reflection故事起源。针对任何您想探勘的class，唯有先为它产生一个Class object，接下来才能经由后者唤起为数十多个的Reflection APIs。这些APIs将在稍后的探险活动中一一亮相。

#001 public final

#002 class **Class<T>** implements java.io.Serializable,

#003 java.lang.reflect.GenericDeclaration,

#004 java.lang.reflect.Type,

#005 java.lang.reflect.AnnotatedElement {

#006 **private** Class() {}

#007 public String **toString**() {

#008 return ( isInterface() ? "interface " :

#009 (isPrimitive() ? "" : "class "))

#010 + getName();

#011 }

...

**图1：Class class片段。注意它的private empty ctor，意指不允许任何人经由编程方式产生Class object。是的，其object 只能由JVM 产生。**

**“Class” object的取得途径**

Java允许我们从多种管道为一个class生成对应的Class object。图2是一份整理。

|  |  |
| --- | --- |
| Class object 诞生管道 | 示例 |
| 运用getClass()  注：每个class 都有此函数 | String str = "abc";  Class c1 = str.getClass(); |
| 运用  Class.getSuperclass()2 | Button b = new Button();  Class c1 = b.getClass();  Class c2 = c1.getSuperclass(); |
| 运用static method  Class.forName()  （最常被使用） | Class c1 = Class.forName ("java.lang.String");  Class c2 = Class.forName ("java.awt.Button");  Class c3 = Class.forName ("java.util.LinkedList$Entry");  Class c4 = Class.forName ("I");  Class c5 = Class.forName ("[I"); |
| 运用  .class 语法 | Class c1 = String.class;  Class c2 = java.awt.Button.class;  Class c3 = Main.InnerClass.class;  Class c4 = int.class;  Class c5 = int[].class; |
| 运用  primitive wrapper classes  的TYPE 语法 | Class c1 = Boolean.TYPE;  Class c2 = Byte.TYPE;  Class c3 = Character.TYPE;  Class c4 = Short.TYPE;  Class c5 = Integer.TYPE;  Class c6 = Long.TYPE;  Class c7 = Float.TYPE;  Class c8 = Double.TYPE;  Class c9 = Void.TYPE; |

**图2：Java 允许多种管道生成Class object。**

### Java classes 组成分析

首先容我以图3的java.util.LinkedList为例，将Java class的定义大卸八块，每一块分别对应图4所示的Reflection API。图5则是“获得class各区块信息”的程序示例及执行结果，它们都取自本文示例程序的对应片段。

package java.util; //(1)

import java.lang.\*; //(2)

public class **LinkedList**<**E**> //(3)(4)(5)

extends **AbstractSequentialList**<E> //(6)

implements **List**<E>, **Queue**<E>,

Cloneable, java.io.Serializable //(7)

{

private static class Entry<E> { … }//(8)

public LinkedList() { … } //(9)

public LinkedList(Collection<? extends E> c) { … }

public E getFirst() { … } //(10)

public E getLast() { … }

private transient Entry<E> header = …; //(11)

private transient int size = 0;

}

**图3：将一个Java class 大卸八块，每块相应于一个或一组Reflection APIs（图4）。**

### Java classes 各成份所对应的Reflection APIs

图3的各个Java class成份，分别对应于图4的Reflection API，其中出现的Package、Method、Constructor、Field等等classes，都定义于java.lang.reflect。

|  |  |  |  |
| --- | --- | --- | --- |
| Java class 内部模块（参见图3） | Java class 内部模块说明 | 相应之Reflection API，多半为Class methods。 | 返回值类型(return type) |
| (1) package | class隶属哪个package | getPackage() | Package |
| (2) import | class导入哪些classes | 无直接对应之API。  解决办法见图5-2。 |  |
| (3) modifier | class（或methods, fields）的属性 | int getModifiers()  Modifier.toString(int)  Modifier.isInterface(int) | int  String  bool |
| (4) class name or interface name | class/interface | 名称getName() | String |
| (5) type parameters | 参数化类型的名称 | getTypeParameters() | TypeVariable <Class>[] |
| (6) base class | base class（只可能一个） | getSuperClass() | Class |
| (7) implemented interfaces | 实现有哪些interfaces | getInterfaces() | Class[] |
| (8) inner classes | 内部classes | getDeclaredClasses() | Class[] |
| (8') outer class | 如果我们观察的class 本身是inner classes，那么相对它就会有个outer class。 | getDeclaringClass() | Class |
| (9) constructors | 构造函数getDeclaredConstructors() | 不论 public 或private 或其它access level，皆可获得。另有功能近似之取得函数。 | Constructor[] |
| (10) methods | 操作函数getDeclaredMethods() | 不论 public 或private 或其它access level，皆可获得。另有功能近似之取得函数。 | Method[] |
| (11) fields | 字段（成员变量） | getDeclaredFields()不论 public 或private 或其它access level，皆可获得。另有功能近似之取得函数。 | Field[] |

**图4：Java class大卸八块后（如图3），每一块所对应的Reflection API。本表并非**

**Reflection APIs 的全部。**

### Java Reflection API 运用示例

图5示范图4提过的每一个Reflection API，及其执行结果。程序中出现的tName()是个辅助函数，可将其第一自变量所代表的“Java class完整路径字符串”剥除路径部分，留下class名称，储存到第二自变量所代表的一个hashtable去并返回（如果第二自变量为null，就不储存而只是返回）。

#001 Class c = null;

#002 c = Class.forName(args[0]);

#003

#004 Package p;

#005 p = c.getPackage();

#006

#007 if (p != null)

#008 System.out.println("package "+p.getName()+";");

执行结果（例）：

package java.util;

**图5-1：找出class 隶属的package。其中的c将继续沿用于以下各程序片段。**

#001 ff = c.getDeclaredFields();

#002 for (int i = 0; i < ff.length; i++)

#003 x = tName(ff[i].getType().getName(), classRef);

#004

#005 cn = c.getDeclaredConstructors();

#006 for (int i = 0; i < cn.length; i++) {

#007 Class cx[] = cn[i].getParameterTypes();

#008 for (int j = 0; j < cx.length; j++)

#009 x = tName(cx[j].getName(), classRef);

#010 }

#011

#012 mm = c.getDeclaredMethods();

#013 for (int i = 0; i < mm.length; i++) {

#014 x = tName(mm[i].getReturnType().getName(), classRef);

#015 Class cx[] = mm[i].getParameterTypes();

#016 for (int j = 0; j < cx.length; j++)

#017 x = tName(cx[j].getName(), classRef);

#018 }

#019 classRef.remove(c.getName()); //不必记录自己（不需import 自己）

执行结果（例）：

import java.util.ListIterator;

import java.lang.Object;

import java.util.LinkedList$Entry;

import java.util.Collection;

import java.io.ObjectOutputStream;

import java.io.ObjectInputStream;

**图5-2：找出导入的classes，动作细节详见内文说明。**

#001 int mod = c.getModifiers();

#002 System.out.print(Modifier.toString(mod)); //整个modifier

#003

#004 if (Modifier.isInterface(mod))

#005 System.out.print(" "); //关键词 "interface" 已含于modifier

#006 else

#007 System.out.print(" class "); //关键词 "class"

#008 System.out.print(tName(c.getName(), null)); //class 名称

执行结果（例）：

public class LinkedList

**图5-3：找出class或interface 的名称，及其属性（modifiers）。**

#001 TypeVariable<Class>[] tv;

#002 tv = c.getTypeParameters(); //warning: unchecked conversion

#003 for (int i = 0; i < tv.length; i++) {

#004 x = tName(tv[i].getName(), null); //例如 E,K,V...

#005 if (i == 0) //第一个

#006 System.out.print("<" + x);

#007 else //非第一个

#008 System.out.print("," + x);

#009 if (i == tv.length-1) //最后一个

#010 System.out.println(">");

#011 }

执行结果（例）：

public abstract interface Map<K,V>

或 public class LinkedList<E>

**图5-4：找出parameterized types 的名称**

#001 Class supClass;

#002 supClass = c.getSuperclass();

#003 if (supClass != null) //如果有super class

#004 System.out.print(" extends" +

#005 tName(supClass.getName(),classRef));

执行结果（例）：

public class LinkedList<E>

extends AbstractSequentialList,

**图5-5：找出base class。执行结果多出一个不该有的逗号于尾端。此非本处重点，为简化计，不多做处理。**

#001 Class cc[];

#002 Class ctmp;

#003 //找出所有被实现的interfaces

#004 cc = c.getInterfaces();

#005 if (cc.length != 0)

#006 System.out.print(", \r\n" + " implements "); //关键词

#007 for (Class cite : cc) //JDK1.5 新式循环写法

#008 System.out.print(tName(cite.getName(), null)+", ");

执行结果（例）：

public class LinkedList<E>

extends AbstractSequentialList,

implements List, Queue, Cloneable, Serializable,

**图5-6：找出implemented interfaces。执行结果多出一个不该有的逗号于尾端。此非本处重点，为简化计，不多做处理。**

#001 cc = c.getDeclaredClasses(); //找出inner classes

#002 for (Class cite : cc)

#003 System.out.println(tName(cite.getName(), null));

#004

#005 ctmp = c.getDeclaringClass(); //找出outer classes

#006 if (ctmp != null)

#007 System.out.println(ctmp.getName());

执行结果（例）：

LinkedList$Entry

LinkedList$ListItr

**图5-7：找出inner classes 和outer class**

#001 Constructor cn[];

#002 cn = c.getDeclaredConstructors();

#003 for (int i = 0; i < cn.length; i++) {

#004 int md = cn[i].getModifiers();

#005 System.out.print(" " + Modifier.toString(md) + " " +

#006 cn[i].getName());

#007 Class cx[] = cn[i].getParameterTypes();

#008 System.out.print("(");

#009 for (int j = 0; j < cx.length; j++) {

#010 System.out.print(tName(cx[j].getName(), null));

#011 if (j < (cx.length - 1)) System.out.print(", ");

#012 }

#013 System.out.print(")");

#014 }

执行结果（例）：

public java.util.LinkedList(Collection)

public java.util.LinkedList()

**图5-8a：找出所有constructors**

#004 System.out.println(cn[i].toGenericString());

执行结果（例）：

public java.util.LinkedList(java.util.Collection<? extends E>)

public java.util.LinkedList()

**图5-8b：找出所有constructors。本例在for 循环内使用toGenericString()，省事。**

#001 Method mm[];

#002 mm = c.getDeclaredMethods();

#003 for (int i = 0; i < mm.length; i++) {

#004 int md = mm[i].getModifiers();

#005 System.out.print(" "+Modifier.toString(md)+" "+

#006 tName(mm[i].getReturnType().getName(), null)+" "+

#007 mm[i].getName());

#008 Class cx[] = mm[i].getParameterTypes();

#009 System.out.print("(");

#010 for (int j = 0; j < cx.length; j++) {

#011 System.out.print(tName(cx[j].getName(), null));

#012 if (j < (cx.length - 1)) System.out.print(", ");

#013 }

#014 System.out.print(")");

#015 }

执行结果（例）：

public Object get(int)

public int size()

**图5-9a：找出所有methods**

#004 System.out.println(mm[i].toGenericString());

public E java.util.LinkedList.get(int)

public int java.util.LinkedList.size()

**图5-9b：找出所有methods。本例在for 循环内使用toGenericString()，省事。**

#001 Field ff[];

#002 ff = c.getDeclaredFields();

#003 for (int i = 0; i < ff.length; i++) {

#004 int md = ff[i].getModifiers();

#005 System.out.println(" "+Modifier.toString(md)+" "+

#006 tName(ff[i].getType().getName(), null) +" "+

#007 ff[i].getName()+";");

#008 }

执行结果（例）：

private transient LinkedList$Entry header;

private transient int size;

**图5-10a：找出所有fields**

#004 System.out.println("G: " + ff[i].toGenericString());

private transient java.util.LinkedList.java.util.LinkedList$Entry<E> ??

java.util.LinkedList.header

private transient int java.util.LinkedList.size

**图5-10b：找出所有fields。本例在for 循环内使用toGenericString()，省事。**

### 找出class参用（导入）的所有classes

没有直接可用的Reflection API可以为我们找出某个class参用的所有其它classes。要获得这项信息，必须做苦工，一步一脚印逐一记录。我们必须观察所有fields的类型、所有methods（包括constructors）的参数类型和回返类型，剔除重复，留下唯一。这正是为什么图5-2程序代码要为tName()指定一个hashtable（而非一个null）做为第二自变量的缘故：hashtable可为我们储存元素（本例为字符串），又保证不重复。

本文讨论至此，几乎可以还原一个class的原貌（唯有methods 和ctors的定义无法取得）。接下来讨论Reflection 的另三个动态性质：(1) 运行时生成instances，(2) 执

行期唤起methods，(3) 运行时改动fields。

### 运行时生成instances

欲生成对象实体，在Reflection 动态机制中有两种作法，一个针对“无自变量ctor”，

一个针对“带参数ctor”。图6是面对“无自变量ctor”的例子。如果欲调用的是“带参数ctor“就比较麻烦些，图7是个例子，其中不再调用Class的newInstance()，而是调用Constructor 的newInstance()。图7首先准备一个Class[]做为ctor的参数类型（本例指定为一个double和一个int），然后以此为自变量调用getConstructor()，获得一个专属ctor。接下来再准备一个Object[] 做为ctor实参值（本例指定3.14159和125），调用上述专属ctor的newInstance()。

#001 Class c = Class.forName("DynTest");

#002 Object obj = null;

#003 obj = c.newInstance(); //不带自变量

#004 System.out.println(obj);

**图6：动态生成“Class object 所对应之class”的对象实体；无自变量。**

#001 Class c = Class.forName("DynTest");

#002 Class[] pTypes = new Class[] { double.class, int.class };

#003 Constructor ctor = c.getConstructor(pTypes);

#004 //指定parameter list，便可获得特定之ctor

#005

#006 Object obj = null;

#007 Object[] arg = new Object[] {3.14159, 125}; //自变量

#008 obj = ctor.newInstance(arg);

#009 System.out.println(obj);

**图7：动态生成“Class object 对应之class”的对象实体；自变量以Object[]表示。**

### 运行时调用methods

这个动作和上述调用“带参数之ctor”相当类似。首先准备一个Class[]做为ctor的参数类型（本例指定其中一个是String，另一个是Hashtable），然后以此为自变量调用getMethod()，获得特定的Method object。接下来准备一个Object[]放置自变量，然后调用上述所得之特定Method object的invoke()，如图8。知道为什么索取Method object时不需指定回返类型吗？因为method overloading机制要求signature（署名式）必须唯一，而回返类型并非signature的一个成份。换句话说，只要指定了method名称和参数列，就一定指出了一个独一无二的method。

#001 public String **func**(String s, Hashtable ht)

#002 {

#003 …System.out.println("func invoked"); return s;

#004 }

#005 public static void main(String args[])

#006 {

#007 Class c = Class.forName("Test");

#008 Class ptypes[] = new Class[2];

#009 ptypes[0] = Class.forName("java.lang.String");

#010 ptypes[1] = Class.forName("java.util.Hashtable");

#011 **Method** m = c.**getMethod**("**func**",ptypes);

#012 Test obj = new Test();

#013 Object args[] = new Object[2];

#014 arg[0] = new String("Hello,world");

#015 arg[1] = null;

#016 Object r = m.**invoke**(obj, arg);

#017 Integer rval = (String)r;

#018 System.out.println(rval);

#019 }

**图8：动态唤起method**

### 运行时变更fields内容

与先前两个动作相比，“变更field内容”轻松多了，因为它不需要参数和自变量。首先调用Class的getField()并指定field名称。获得特定的Field object之后便可直接调用Field的get()和set()，如图9。

#001 public class Test {

#002 public double **d**;

#003

#004 public static void main(String args[])

#005 {

#006 Class c = Class.forName("Test");

#007 **Field f** = c.getField("**d**"); //指定field 名称

#008 Test obj = new Test();

#009 System.out.println("d= " + (Double)**f.get**(obj));

#010 **f.set**(obj, 12.34);

#011 System.out.println("d= " + obj.**d**);

#012 }

#013 }

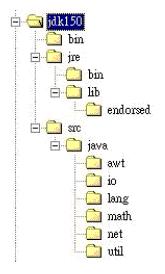
**图9：动态变更field 内容**

### Java 源码改动办法

先前我曾提到，原本想借由“改动Java标准库源码”来测知Class object的生成，但由于其ctor原始设计为private，也就是说不可能透过这个管道生成Class object（而是由class loader负责生成），因此“在ctor中打印出某种信息”的企图也就失去了意义。

这里我要谈点题外话：如何修改Java标准库源码并让它反应到我们的应用程序来。假设我想修改java.lang.Class，让它在某些情况下打印某种信息。首先必须找出标准源码！当你下载JDK 套件并安装妥当，你会发现jdk150\src\java\lang 目录（见图10）之中有Class.java，这就是我们此次行动的标准源码。备份后加以修改，编译获得Class.class。接下来准备将.class 搬移到jdk150\jre\lib\endorsed（见图10）。

这是一个十分特别的目录，class loader将优先从该处读取内含classes的.jar文件——成功的条件是.jar内的classes压缩路径必须和Java标准库的路径完全相同。为此，我们可以将刚才做出的Class.class先搬到一个为此目的而刻意做出来的\java\lang目录中，压缩为foo.zip（任意命名，唯需夹带路径java\lang），再将这个foo.zip搬到jdk150\jre\lib\endorsed并改名为foo.jar。此后你的应用程序便会优先用上这里的java.lang.Class。整个过程可写成一个批处理文件（batch file），如图11，在DOS Box中使用。



**图10：JDK1.5 安装后的目录组织。其中的endorsed 是我新建。**

del e:\java\lang\\*.class //清理干净

del c:\jdk150\jre\lib\endorsed\foo.jar //清理干净

c:

cd c:\jdk150\src\java\lang

javac -Xlint:unchecked Class.java //编译源码

javac -Xlint:unchecked ClassLoader.java //编译另一个源码（如有必要）

move \*.class e:\java\lang //搬移至刻意制造的目录中

e:

cd e:\java\lang //以下压缩至适当目录

**pkzipc** -add -path=root c:\jdk150\jre\lib\endorsed\foo.jar \*.class

cd e:\test //进入测试目录

javac -Xlint:unchecked Test.java //编译测试程序

java Test //执行测试程序

**图11：一个可在DOS Box中使用的批处理文件（batch file），用以自动化java.lang.Class**

**的修改动作。Pkzipc(.exe)是个命令列压缩工具，add和path都是其命令。**

注1用过诸如MFC这类所谓 Application Framework的程序员也许知道，MFC有所谓的dynamic creation。但它并不等同于Java的动态加载或动态辨识；所有能够在MFC程序中起作用的classes，都必须先在编译期被编译器“看见”。

# 十七、垃圾回收

## 概念

gc即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。java语言并不要求jvm有gc，也没有规定gc如何工作。不过常用的jvm都有gc，而且大多数gc都使用类似的算法管理内存和执行收集操作。

　　在充分理解了垃圾收集算法和执行过程后，才能有效的优化它的性能。有些垃圾收集专用于特殊的应用程序。比如，实时应用程序主要是为了避免垃圾收集中断，而大多数OLTP应用程序则注重整体效率。理解了应用程序的工作负荷和jvm支持的垃圾收集算法，便可以进行优化配置垃圾收集器。

　　垃圾收集的目的在于清除不再使用的对象。gc通过确定对象是否被活动对象引用来确定是否收集该对象。gc首先要判断该对象是否是时候可以收集。两种常用的方法是引用计数和对象引用遍历。

### 1.1.引用计数

　　引用计数存储对特定对象的所有引用数，也就是说，当应用程序创建引用以及引用超出范围时，jvm必须适当增减引用数。当某对象的引用数为0时，便可以进行垃圾收集。

### 1.2.对象引用遍历

　　早期的jvm使用引用计数，现在大多数jvm采用对象引用遍历。对象引用遍历从一组对象开始，沿着整个对象图上的每条链接，递归确定可到达（reachable）的对象。如果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。

　　下一步，gc要删除不可到达的对象。删除时，有些gc只是简单的扫描堆栈，删除未标记的未标记的对象，并释放它们的内存以生成新的对象，这叫做清除（sweeping）。这种方法的问题在于内存会分成好多小段，而它们不足以用于新的对象，但是组合起来却很大。因此，许多gc可以重新组织内存中的对象，并进行压缩（compact），形成可利用的空间。

　　为此，gc需要停止其他的活动活动。这种方法意味着所有与应用程序相关的工作停止，只有gc运行。结果，在响应期间增减了许多混杂请求。另外，更复杂的 gc不断增加或同时运行以减少或者清除应用程序的中断。有的gc使用单线程完成这项工作，有的则采用多线程以增加效率。

## 2.几种垃圾回收机制

### 2.1.标记－清除收集器

　　这种收集器首先遍历对象图并标记可到达的对象，然后扫描堆栈以寻找未标记对象并释放它们的内存。这种收集器一般使用单线程工作并停止其他操作。

### 2.2.标记－压缩收集器

　　有时也叫标记－清除－压缩收集器，与标记－清除收集器有相同的标记阶段。在第二阶段，则把标记对象复制到堆栈的新域中以便压缩堆栈。这种收集器也停止其他操作。

### 2.3.复制收集器

　　这种收集器将堆栈分为两个域，常称为半空间。每次仅使用一半的空间，jvm生成的新对象则放在另一半空间中。gc运行时，它把可到达对象复制到另一半空间，从而压缩了堆栈。这种方法适用于短生存期的对象，持续复制长生存期的对象则导致效率降低。

### 2.4.增量收集器

　　增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。

### 2.5.分代收集器

　　这种收集器把堆栈分为两个或多个域，用以存放不同寿命的对象。jvm生成的新对象一般放在其中的某个域中。过一段时间，继续存在的对象将获得使用期并转入更长寿命的域中。分代收集器对不同的域使用不同的算法以优化性能。

### 2.6.并发收集器

　　并发收集器与应用程序同时运行。这些收集器在某点上（比如压缩时）一般都不得不停止其他操作以完成特定的任务，但是因为其他应用程序可进行其他的后台操作，所以中断其他处理的实际时间大大降低。

### 2.7.并行收集器

　　并行收集器使用某种传统的算法并使用多线程并行的执行它们的工作。在多cpu机器上使用多线程技术可以显著的提高java应用程序的可扩展性。

## 3.Sun HotSpot

### 1.4.1 JVM堆大小的调整

　　Sun HotSpot 1.4.1使用分代收集器，它把堆分为三个主要的域：新域、旧域以及永久域。Jvm生成的所有新对象放在新域中。一旦对象经历了一定数量的垃圾收集循环后，便获得使用期并进入旧域。在永久域中jvm则存储class和method对象。就配置而言，永久域是一个独立域并且不认为是堆的一部分。

　　下面介绍如何控制这些域的大小。可使用-Xms和-Xmx 控制整个堆的原始大小或最大值。

　　下面的命令是把初始大小设置为128M：

　　java –Xms128m

　　–Xmx256m为控制新域的大小，可使用-XX:NewRatio设置新域在堆中所占的比例。

　　下面的命令把整个堆设置成128m，新域比率设置成3，即新域与旧域比例为1：3，新域为堆的1/4或32M：

java –Xms128m –Xmx128m

–XX:NewRatio =3可使用-XX:NewSize和-XX:MaxNewsize设置新域的初始值和最大值。

　　下面的命令把新域的初始值和最大值设置成64m:

java –Xms256m –Xmx256m –Xmn64m

　　永久域默认大小为4m。运行程序时，jvm会调整永久域的大小以满足需要。每次调整时，jvm会对堆进行一次完全的垃圾收集。

　　使用-XX:MaxPerSize标志来增加永久域搭大小。在WebLogic Server应用程序加载较多类时，经常需要增加永久域的最大值。当jvm加载类时，永久域中的对象急剧增加，从而使jvm不断调整永久域大小。为了避免调整，可使用-XX:PerSize标志设置初始值。

　　下面把永久域初始值设置成32m，最大值设置成64m。

java -Xms512m -Xmx512m -Xmn128m -XX:PermSize=32m -XX:MaxPermSize=64m

　　默认状态下，HotSpot在新域中使用复制收集器。该域一般分为三个部分。第一部分为Eden，用于生成新的对象。另两部分称为救助空间，当Eden 充满时，收集器停止应用程序，把所有可到达对象复制到当前的from救助空间，一旦当前的from救助空间充满，收集器则把可到达对象复制到当前的to救助空间。From和to救助空间互换角色。维持活动的对象将在救助空间不断复制，直到它们获得使用期并转入旧域。使用-XX:SurvivorRatio 可控制新域子空间的大小。

　　同NewRation一样，SurvivorRation规定某救助域与Eden空间的比值。比如，以下命令把新域设置成64m，Eden占32m，每个救助域各占16m：

java -Xms256m -Xmx256m -Xmn64m -XX:SurvivorRation =2

　　如前所述，默认状态下HotSpot对新域使用复制收集器，对旧域使用标记－清除－压缩收集器。在新域中使用复制收集器有很多意义，因为应用程序生成的大部分对象是短寿命的。理想状态下，所有过渡对象在移出Eden空间时将被收集。如果能够这样的话，并且移出Eden空间的对象是长寿命的，那么理论上可以立即把它们移进旧域，避免在救助空间反复复制。但是，应用程序不能适合这种理想状态，因为它们有一小部分中长寿命的对象。最好是保持这些中长寿命的对象并放在新域中，因为复制小部分的对象总比压缩旧域廉价。为控制新域中对象的复制，可用-XX:TargetSurvivorRatio控制救助空间的比例（该值是设置救助空间的使用比例。如救助空间位1M，该值50表示可用500K）。该值是一个百分比，默认值是50。当较大的堆栈使用较低的 sruvivorratio时，应增加该值到80至90，以更好利用救助空间。用-XX:maxtenuring threshold可控制上限。

　　为放置所有的复制全部发生以及希望对象从eden扩展到旧域，可以把MaxTenuring Threshold设置成0。设置完成后，实际上就不再使用救助空间了，因此应把SurvivorRatio设成最大值以最大化Eden空间，设置如下：

java … -XX:MaxTenuringThreshold=0 –XX:SurvivorRatio＝50000 …

## 4.BEA JRockit JVM的使用

　　Bea WebLogic 8.1使用的新的JVM用于Intel平台。在Bea安装完毕的目录下可以看到有一个类似于jrockit81sp1\_141\_03的文件夹。这就是 Bea新JVM所在目录。不同于HotSpot把Java字节码编译成本地码，它预先编译成类。JRockit还提供了更细致的功能用以观察JVM的运行状态，主要是独立的GUI控制台（只能适用于使用Jrockit才能使用jrockit81sp1\_141\_03自带的console监控一些cpu及 memory参数）或者WebLogic Server控制台。

　　Bea JRockit JVM支持4种垃圾收集器：

### 4.1.1.分代复制收集器

　　它与默认的分代收集器工作策略类似。对象在新域中分配，即JRockit文档中的nursery。这种收集器最适合单cpu机上小型堆操作。

### 4.1.2.单空间并发收集器

　　该收集器使用完整堆，并与背景线程共同工作。尽管这种收集器可以消除中断，但是收集器需花费较长的时间寻找死对象，而且处理应用程序时收集器经常运行。如果处理器不能应付应用程序产生的垃圾，它会中断应用程序并关闭收集。

　　分代并发收集器这种收集器在护理域使用排它复制收集器，在旧域中则使用并发收集器。由于它比单空间共同发生收集器中断频繁，因此它需要较少的内存，应用程序的运行效率也较高，注意，过小的护理域可以导致大量的临时对象被扩展到旧域中。这会造成收集器超负荷运作，甚至采用排它性工作方式完成收集。

### 4.1.3.并行收集器

　　该收集器也停止其他进程的工作，但使用多线程以加速收集进程。尽管它比其他的收集器易于引起长时间的中断，但一般能更好的利用内存，程序效率也较高。

　　默认状态下，JRockit使用分代并发收集器。要改变收集器，可使用-Xgc:，对应四个收集器分别为 gencopy，singlecon，gencon以及parallel。可使用-Xms和-Xmx设置堆的初始大小和最大值。要设置护理域，则使用- Xns:java –jrockit –Xms512m –Xmx512m –Xgc:gencon –Xns128m…尽管JRockit支持-verbose:gc开关，但它输出的信息会因收集器的不同而异。JRockit还支持memory、 load和codegen的输出。

　　注意 ：如果 使用JRockit JVM的话还可以使用WLS自带的console（C:\bea\jrockit81sp1\_141\_03\bin下）来监控一些数据，如cpu， memery等。要想能构监控必须在启动服务时startWeblogic.cmd中加入－Xmanagement参数。

## 5.如何从JVM中获取信息来进行调整

　　-verbose.gc开关可显示gc的操作内容。打开它，可以显示最忙和最空闲收集行为发生的时间、收集前后的内存大小、收集需要的时间等。打开- xx:+ printgcdetails开关，可以详细了解gc中的变化。打开-XX: + PrintGCTimeStamps开关，可以了解这些垃圾收集发生的时间，自jvm启动以后以秒计量。最后，通过-xx: + PrintHeapAtGC开关了解堆的更详细的信息。为了了解新域的情况，可以通过-XX:=PrintTenuringDistribution开关了解获得使用期的对象权。

## 6.Pdm系统JVM调整

### 6.1.服务器：前提内存1G 单CPU

　　可通过如下参数进行调整：－server 启用服务器模式（如果CPU多，服务器机建议使用此项）

　　－Xms,－Xmx一般设为同样大小。 800m

　　－Xmn 是将NewSize与MaxNewSize设为一致。320m

　　－XX:PerSize 64m

　　－XX:NewSize 320m 此值设大可调大新对象区，减少Full GC次数

　　－XX:MaxNewSize 320m

　　－XX:NewRato NewSize设了可不设。

　　－XX: SurvivorRatio

　　－XX:userParNewGC 可用来设置并行收集

　　－XX:ParallelGCThreads 可用来增加并行度

　　－XXUseParallelGC 设置后可以使用并行清除收集器

　　－XX：UseAdaptiveSizePolicy 与上面一个联合使用效果更好，利用它可以自动优化新域大小以及救助空间比值

### 6.2.客户机：通过在JNLP文件中设置参数来调整客户端JVM

　　JNLP中参数：initial-heap-size和max-heap-size

　　这可以在framework的RequestManager中生成JNLP文件时加入上述参数，但是这些值是要求根据客户机的硬件状态变化的（如客户机的内存大小等）。建议这两个参数值设为客户机可用内存的60％（有待测试）。为了在动态生成JNLP时以上两个参数值能够随客户机不同而不同，可靠虑获得客户机系统信息并将这些嵌到首页index.jsp中作为连接请求的参数。

　　在设置了上述参数后可以通过Visualgc 来观察垃圾回收的一些参数状态，再做相应的调整来改善性能。一般的标准是减少fullgc的次数，最好硬件支持使用并行垃圾回收（要求多CPU）。

# 十八、加速垃圾回收方式

## 1.垃圾回收

对象是使用new创建的, 但是并没有与之相对应的delete操作来回收对象占用的内存. 当我们完成对某个对象的使用时, 只需停止该对象的引用:

->将引用改变为指向其他对象

->将引用指向null

->从方法中返回, 使得该方法的局部变量不复存在

要点:

->当我们从任何可执行代码都无法到达某个对象时, 它所占用的空间就可以被回收.

->垃圾回收意味着我们永远不用担心出现虚悬引用(dangling reference). 虚悬引用, 指得是引用已经被删除的内存空间. 在那些程序员可以直接控制何时删除对象的系统中, 会存在这样的问题.

->垃圾回收器模型: 引用计数器法(不能解决循环引用), 标记-清除(mark-and-sweep)

## 2.终结

finalize方法

->在垃圾回收器确定该对象是不可达的且该对象的空间将被回收之后, 垃圾回收器就会调用这个方法.

->这个方法可以清除该对象所使用的所有非内存资源, 对每一个对象最多只能调用一次, 即使在这个方法的执行使得该对象重新变为可达之后又马上会再次变为不可达的情况下, 该方法也只能调用一次.

->finalize方法可以在任何特定的时间段内被调用, 它也可能永远不会被调用(java虚拟机结束).

覆写finalize方法

->当一个对象变成垃圾时, 它所引用的其他对象也很有可能会变成垃圾. 这些垃圾可能在调用我们编写的finalize方法之前就已经被终结了, 因此它们可能处于不可预知的状态.

->覆写finalize方法是, 加上super.finalize方法. 最好加在finally字句里面.保证其超类中声明的部分内容也可以被终结.

## 3.与垃圾回收器交互的相关类和方法

类: Runtime.getRuntime(), System

方法:gc(), runFinalization(), freeMemory(), totalMemory(), maxMemory()

System类支持静态的gc()和runFinalization()方法, 它们将调用当前Runtime对象上的相应方法.

## 4.可达性状态和引用对象

对象只有在没有任何引用指定它的时候才可以被当作垃圾回收, 但有时我们可能希望在仍旧有选定引用指向对象时, 将该对象作为垃圾回收掉.

引用对象的唯一用途就是维护对另一个被称为指称物(referent)的对象的引用. 通常我们通过字段或者局部变量来维护对对象的引用, 但是现在我们可以维护对引用对象的直接引用, 而该引用对象包装了我们实际需要的对象. 垃圾回收器可能判断出对某个对象的残留引用是否都是经由引用对象面引用到该对象的, 因此它可以决定是否要回收该对象. 引用对象的强度将决定垃圾回收器的行为, 普通的引用都是强度最大的引用.

Reference类

->包:java.lang.ref

->典型方法: get(), clear(), enqueue(), isEnqueued()

引用和可达性强度

->对象是强可达的(strongly reachable):普通的引用

->对象是软可达的(softly reachable):SoftReference

->对象是弱可达的(weakly reachable):WeakReference

->对象是虚可达的(phantom reachable):PhantomReference

->对象是不可达的:没有引用链接

一旦对象变为弱可达的(或者列弱), 它就可以被终结. 如果在终结之后该对象是不可达的, 那么它就可以被回收了.

对象可达性阶段会触发垃圾回收器对相关的引用对象类型做出适当的行为:

->软可达对象可能会任凭垃圾回收器去回收. 我们可确定的是所有对软可达对象的SoftReference都会在抛出outofMemoryError错误这前被清除.

->弱可达对象将会被垃圾回收器回收.

->虚可达对象并不是真正意义上的可达, 因为无法通过PhantomReference访问其指称对象, 其get方法总是返回null. 但是虚引用的存在可以防止对象在显式清除虚引用之前被回收. 虚引用使我们可以处理那些finalize方法已经被调用过的对象, 从而可以安全地认为它们是"死"的.