注意点：

静态方法：可以直接用类名.方法调用，跟静态内部类区分开

内部类：

成员内部类：成员内部类不能包含静态变量和方法

静态内部类：

局部内部类：在一个方法内定义的类

匿名内部类

☆参数传递的问题：对于基本类型的参数，传递的是实参的值；引用类型的参数，传递的存放在堆里面的引用

Java中只有一种参数传递方式：值传递

基本类型传递的是实参的值，引用类型（对象、数组）传递的是引用地址

**public** **class** Test {

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

**int** x = 10;

**int** y[] = **new** **int**[10];

System.***out***.println("调用前:x="+x+",y="+y[0]);

*m*(x, y);

System.***out***.println("调用后:x="+x+",y="+y[0]);

}

 **public** **static** **void** m(**int** num, **int**[] nums){

num = 100;

nums[0] = 200;

}

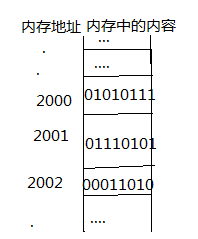
}

Jvm中堆栈的问题

Java把内存分成两种：栈内存和堆内存

栈内存：基本类型的变量和对象的引用变量，当定义一个变量就会在栈中分配内存空间，超过作用域后，java就会自动释放内存。

堆内存：存放new创建的对象和数组，在堆中的内存由java虚拟机自动垃圾回收其管理，没有引用变量指向的时候，这个对象就会变成垃圾，在随后一个不确定时间被垃圾回收其回收和释放（这是java比较占内存的主要原因）

计算机内存（RAM）：每个字节由8**位**（0，1）组成，每个字节都有唯一的地址，地址可以确定字节的位置，便于存储和提取

单目运算符：只有一个变量，如自增a++；双目运算符，2个变量

boolean布尔运算符也叫逻辑运算符，所以 &&——逻辑与(条件与，短路与)，||——逻辑或(条件或，短路或)；&和|（无条件与和无条件或）的不同点在于，p1&p2，p1|p2两个运算对象都要计算

条件运算符 x > y ? a : b 也叫三目运算符，java中唯一一个三目运算符

Java数据类型：基本类型和引用类型

栈中直接分配的有基本数据类型，数据的引用是在栈中，但是他的对象在堆中

|  |  |  |
| --- | --- | --- |
| 基本类型 | 数值型 | 整数类型byte,short,int,long |
| 浮点型 float,double |
| 非数值型 | 字符型 char |
| 逻辑（布尔）型 boolean |
| 引用类型 | | 类 class |
| 接口 interface |
| 数组 |

Byte字节型，在内存中占用1个字节，范围-128~~127

Short短数型，在内存中占用2个字节，范围-32768~~32767

Int整数型（默认），在内存中占用4个字节

Long长整数型，在内存中占用8个字节，声明时后面要加上l或L

Char字符型，在内存中占用2个字节

Float单精度浮点型，在内存中占用4个字节，声明时后面要加上f或F

Double双精度浮点型（默认），在内存中占用8个字节

一个字节=8位二进制=256个数（-128--127） kb -🡪kBytes

类型转换：小可自动转大，大转小需要强制转换，而且会失去精度

Byte -----> short,char -----> int -----> long（表达式在计算中，如+-之类运算，所有byte,short,char自动转成int）

Float -----> double

Int -----> float

Long -----> double

基本类型包装器：基本类型在java中已预定义，不是对象，所以就基本类型的包装器来转成d对象处理，jdk1.5后加入了自动装箱和拆箱功能

使用泛型的目的：1.将运行时的异常转成编译错误，减少运行时异常数量（提交编译器能力）2.解决模版编程的问题

List<String> list = new ArrayList<String>();

String str = list.get(0);//这里就不用做类型转换

Java中对象和类，面向对象程序设计（OOP）

对象是现实世界的一个实体（一个学生，一辆车，一个圆），一个对象是类的一个实例，创建实例称为实例化，构造方法是为了进行初始化对象

静态变量，静态方法，实例变量，实例方法

Public class Circle{

double radius;

Private static int numOfObj;

}

静态变量：类中所有实例共享数据，变量值存储在类的公用内存中。如Circle c1 = new Circle()和Circle c2 = new Circle()，c1和c2的mumOfObj都是公用

实例变量：实例化后，实例变量radius会根据不同的实例c1和c2存储在不同的内存空间，其中一个变化不会相互影响

实例方法和实例变量属于实例，只能实例化后才能使用，通过引用变量来调用Circle c = new Circle()；如c1.getArea()，c1.radius = 9等。而静态方法和静态变量可以用引用变量或者类名直接调用。静态变量和静态方法可以在类的实例方法或者静态方法中使用，而实例变量和实例方法只能在实例方法中使用

引用变量：在栈中该变量指向放在堆内存中对象的引用地址

Ps：数组是看作对象，用new创建的，数组的引用变量包含数组引用变量的地址

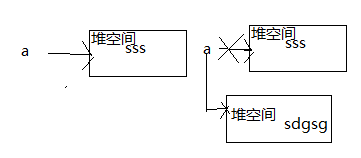
作用域

类的实例变量和静态变量称为类变量或者数据域

Java中，在构造方法中this语句出现在任何其他语句前

字符串，在java中，字符串是对象

String str =”哈哈哈” ----------------🡪String str = new String(“哈哈哈”)

String对象是不可变的，String a=”sss”; a=”sdgsg”。这样a就指向新的String对象，原来的String对象就变成垃圾

String类型的==，比较的是对象的引用地址，equals()才是比较内容（值）

字符串和数组间的转换

String.toCharArray()------🡪字符串转换成字符数组

String,StringBuilder和StringBuffer

StringBuilder和StringBuffer可以在字符串缓冲区中添加、插入或追加新的内容，多线程访问的话用StringBuffer，单线程用StringBuilder

继承和多态

Super：指向使用它的类的父类，用处：1.调用父类的构造方法 2.调用父类的方法

调用父类的构造方法必须使用super，而且必须是在子类构造方法的第一条语句

抽象类和接口

包含抽象方法的类必须声明为抽象类（但是抽象类不一定有抽象方法），抽象类不能用new操作符来创建它的实例

接口中所有的数据域（变量）都隐式声明为public final static，方法都是public abstract。接口中的常量可以用接口名.常量名访问(T1.str)。接口可以多重继承public interface myInterface extends Interface1，Interface2

I/O流

字节流：以stream结尾；字符流：以Writer和Reader)结尾。按功能分：节点流，直接跟数据源相连FileInputStream，构造方法的参数是一个文件或者其他数据源new FileInputStream(File file)；过滤流，基本上是跟节点流连接，构造方法的参数是另外一个流,继承FilterInputstream或者FilterOutputStream，如new DataInputStream(InputStream in)

桥接流：如InputStreamReader用处1.将字节流转成字符流 2.设置编码和解码格式

缓冲流：将需要读写的数据先写到缓存(内存)，等缓冲区满了再一次性打开io来操作，提高性能。用缓冲流必须要flush()，不然可能会因为缓冲区没满，io操作就还没出现

数据流：直接对8个基本类型和字符串的读写

打印流：PrintWriter

序列化：将一个对象用流的方式输出到文件或者网络等数据源，这个过程叫序列化

线程：在一个进程中运行的多条分支，这些分支会被cpu轮流切换执行，能互相访问，所以也容易造成数据冲突

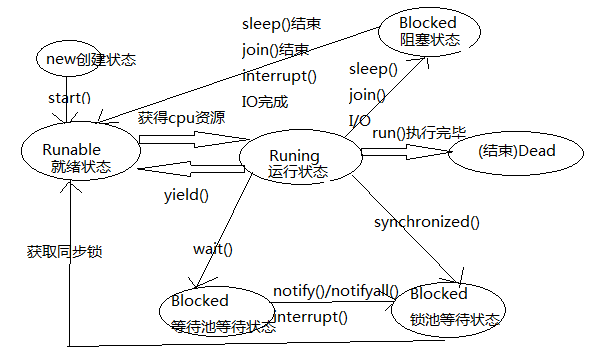
进程：一个完整的程序应用，由操作系统去创建执行，进程间相互独立，各自的内存空间,不能互相访问

实现线程的两种方式：1.继承Thread 2.实现Runable接口 MyRunable myRun = new MyRunable;Thread thread = new Thread(myRun);thread.start();

因为java只能单继承，所以用实现Runable()接口比较好，留一个继承机会

线程的名字：在构造方法中用super()来初始化线程名字；获得当前线程对象Thread.currentThread()

线程的优先级setPriority()，在1—10内，越高优先级越高，默认是5。垃圾回收器是低优先级线程

线程生命周期：

1.创建状态：刚刚new出来的对象，在堆内存分配空间

2.就绪状态：可以被cpu调用，等待时间片调度，调用了start()

3.运行状态：获得cpu调度，运行run()

4.阻塞状态：被动放弃cpu资源，可能是io阻塞或者其他线程合并（就会等其他线程结束才会变成就绪状态）导致

5.等待池等待状态:主动放弃cpu资源，调用了wait()，等待其他线程通知notify或者唤醒interrupt

6.锁池等待状态：防止数据冲突问题synchronized代码块或者synchronized方法

死锁：发生在锁嵌套中，已经获取了第一把锁，还想获取第二把锁

等待池等待wait：只有在synchroized锁住的对象才可以调用wait()，调用后会释放锁，让其他线程获得锁

Class c = Class.forName(cls);

Object o = c.newInstance()

Bean.put(id,o);

//然后获取bean对象的属性值