————————————————语言基础———————————————————

**九种基本数据类型的大小，以及他们的封装类，占用大小。**

Int Integer 4byte 0

char Character 2bye [] null

long Long 8byte 0L

double Double 8byte 0.0d

float Float 4byte 0.0f

bool Boolean (jvm虽然定义了boolean这种数据类型，但是只对它提供了非常有限的支持。在Java虚拟机中没有任何供boolean值专用的字节码指令 4byte或1byte) false

byte Byte 1byte (byte)0

short Short 2byte (short)0

void Void

类的成员是基本类型（封装类是类不行）会有一个初始值

局部变量使用前必须初始化

**Switch能否用string做参数？**

在Java5以前，switch(expr)中，exper只能是byte，short，char，int类型。从Java5开始，java中引入了枚举类型，即enum类型。从Java7开始，exper还可以是String类型。

**equals与==的区别**

==比的是引用

Equals调用的是equals方法，如果equals没被声明则会调用父类的equals方法，object的equals比较的是引用

**Object有哪些公用方法？**

protected Object clone() 克隆一个对象

boolean equals(Object obj) 判等

protected void finalize() 垃圾回收器回收对象空间时调用

Class<?> getClass() 得到类的引用

int hashCode() 类的hash值

void notify()

void notifyAll()

String toString()

void wait()

void wait(long timeout)

void wait(long timeout, int nanos)

**Java的四种引用，强弱软虚，用到的场景。**

强引用 以前我们使用的大部分引用实际上都是强引用，这是使用最普遍的引用。

Counter counter = new Counter(); // strong reference

WeakReference<Counter> weakCounter = new **WeakReference**<Counter>(counter); //weak reference//weakCounter弱引用

SoftReference soft= new SoftReference(counter) ; //soft reference

ReferenceQueue refQueue = new ReferenceQueue(); //引用会在队列中等待被清楚

PhantomReference<Counter> phantom = new PhantomReference<Counter>( counter, refQueue);//虚引用

强引用是使用最普遍的引用。如果一个对象具有强引用，那垃圾回收器绝不会回收它。

如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；

在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。

如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。虚引用主要用来跟踪对象被垃圾回收器回收的活动。

**Hashcode的作用。**

hashCode是用于查找使用的（根据hashcode得到存储位置），而equals是用于比较两个对象的是否相等的

1、hashCode的存在主要是用于查找的快捷性，如Hashtable，HashMap等，hashCode是用来在散列存储结构中确定对象的存储地址的；

2、如果两个对象相同，就是适用于equals([Java](http://lib.csdn.net/base/java).lang.Object) 方法，那么这两个对象的hashCode一定要相同；

3、如果对象的equals方法被重写，那么对象的hashCode也尽量重写，并且产生hashCode使用的对象，一定要和equals方法中使用的一致，否则就会违反上面提到的第2点；

4、两个对象的hashCode相同，并不一定表示两个对象就相同，也就是不一定适用于equals(java.lang.Object) 方法。

**String、StringBuffer与StringBuilder的区别。**

String对象是不可变的 String类的声明是：public final 一旦我们字符串的值改变，就会在内存创建多一个空间来保存新的字符串————操作效率低。字符串是不可变的，所以在它创建的时候hashcode就被缓存了，不需要重新计算。

效率：StringBuilder > StringBuffer > String

StringBuilder：线程非安全的 StringBuffer：线程安全的

StringBuffer与StringBuilder，他们是字符串变量，是可改变的对象

1.如果要操作少量的数据用 = String

2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder

3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

**try catch finally，try里有return，finally还执行么？**

1、不管有木有出现异常，finally块中代码都会执行；

2、当try和catch中有return时，finally仍然会执行；

3、finally是在return后面的表达式运算后执行的 ，return的值会被暂存

4、finally中最好不要包含return，否则程序会提前退出，返回值不是try或catch中保存的返回值。

**Excption与Error包结构。OOM你遇到过哪些情况，SOF你遇到过哪些情况。**



1. Throwable Throwable是 Java 语言中所有错误或异常的超类。

2. Exception Exception及其子类是 Throwable 的一种形式，它指出了合理的应用程序想要捕获的条件。

3. RuntimeException RuntimeException是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类。 编译器不会检查RuntimeException异常。可以不捕获向上抛，如果一直没有处理，则jvm会自动处理（停止线程，打印异常），非运行期异常必须捕获**或**者在方法声明。

4. Error 和Exception一样， Error也是Throwable的子类。 它用于指示合理的应用程序不应该试图捕获的严重问题，大多数这样的错误都是异常条件。 和RuntimeException一样， 编译器也不会检查Error。

Oom与sof稍后补充

**Java面向对象的三个特征与含义。**

继承 封装 多态

**参数列表**

参数的类型，参数的个数，参数的顺序

**Override和Overload的含义的区别。**

Override是覆盖，代表子类重写父类的方法

覆盖

1、“**三同一不低**” 子类和父类的**方法名称**，**参数列表**，**返回类型**必须**完全相同**，子类方法的访问修饰符的权限不能比父类**低**。

2、子类方法不能抛出比父类方法**更多的异常**。

3、被覆盖的方法不能是**final**类型的。（不会覆盖只会新建一个同名的）

4、被覆盖的方法不能为**private**。（不会覆盖只会新建一个同名的

5、被覆盖的方法不能为**static**。因为静态方法是在编译的时候把静态方法和类的引用类型进行匹配。子静态父不静或反之抛错。都静态保留父类。

Overload是重载或过载

重载

重载（overload）：对于类的方法（包括从**父类中继承的方法**），**方法名相同，参数列表不同的方法之间就构成了重载关系**。重载可发生在同一个类中，也可发生在子类与父类中

**Interface与abstract类的区别**

Interface所有的方法都是抽象方法，域都是static和final的，使用方法都是public

含一个抽象方法就是Abstract类，

只能继承一个抽象类 可以实现多个接口

Abstract ————is a(复用) interface————like a（规范）

**Static class 与non static class的区别**

。

**为什么使用内部类**

**什么是擦除**

**foreach与正常for循环效率对比**

直接for循环效率最高，其次是迭代器和 ForEach操作。 其实 ForEach 编译成 字节码之后，使用的是迭代器实现的

**如何实现多重继承**

**访问权限**



**反射的作用于原理**

**多态**

在面向对象语言中，接口的多种不同的实现方式即为多态。

允许将子类类型的[指针](http://baike.baidu.com/item/%E6%8C%87%E9%92%88)赋值给父类类型的指针

把不同的子类对象都当作父类来看，可以屏蔽不同子类对象之间的差异（**后期绑定**）

**强制类型转换**

关于强转

java程序运行后,**对象的类型就不会发生改变**。而强制类型转换只不过是用某种类型去引用内存中的实体,这种类型只是编译时类型,与运行后的实体并没有联系

——————————————————Volatile——————————————————