目录

[1. https://www.jianshu.com/p/c70989bd5f29?from=message&isappinstalled=0 1](#_Toc47032467)

[2. java中==和equals和hashCode的区别 1](#_Toc47032468)

[3. int、char、long各占多少字节数 2](#_Toc47032469)

[4. int与integer的区别 2](#_Toc47032470)

[5. 谈谈对java多态的理解 4](#_Toc47032471)

[6. String、StringBuffer、StringBuilder区别 5](#_Toc47032472)

[7. 什么是内部类？内部类的作用 6](#_Toc47032473)

[8. jvm内存模型 7](#_Toc47032474)

[9. 抽象类和接口区别 8](#_Toc47032475)

[10. 抽象类的意义 9](#_Toc47032476)

[11. 在使用中抽象类和接口我们该如何选择？ 9](#_Toc47032477)

[12. 抽象类是否可以没有方法和属性？ 9](#_Toc47032478)

[13. 泛型中extends和super的区别 9](#_Toc47032479)

[14. 父类的静态方法能否被子类重写 9](#_Toc47032480)

[15. 进程和线程的区别 11](#_Toc47032481)

[16. final，finally，finalize的区别 12](#_Toc47032482)

[17. 序列化的方式 14](#_Toc47032483)

[18. Serializable 和Parcelable 的区别 15](#_Toc47032484)

[19. 静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？ 15](#_Toc47032485)

[20. 静态内部类的设计意图 17](#_Toc47032486)

### <https://www.jianshu.com/p/c70989bd5f29?from=message&isappinstalled=0>

### java中==和equals和hashCode的区别

“==”：

　　==是运算符，用来比较两个值、两个对象的内存地址是否相等；

“equals()”：

　　equals是Object类的方法，默认情况下比较两个对象是否是同一个对象，内部实现是通过“==”来实现的。

如果想比较两个对象的其他内容，则可以通过重写equals方法，

例如：String类就重写了equals方法，改成了对象的内容是否相等。

具体实现可以参考：<https://www.cnblogs.com/hujinshui/p/10038044.html>

“hashCode()”：

　　hashCode也是Object类里面的方法，返回值是一个对象的哈希码，同一个对象哈希码一定相等，但不同对象哈希码也有可能相等。

如果两个对象通过equals方法比较相等，那么他的hashCode一定相等；

如果两个对象通过equals方法比较不相等，那么他的hashCode有可能相等；

例如以下程序：

class Untitled {

public static void main(String[] args) {

System.out.println("AaAa".hashCode());//2031744

System.out.println("BBBB".hashCode());//2031744

}

}

### int、char、long各占多少字节数

Java基本类型占用的字节数：  
1字节： byte , boolean  
2字节： short , char  
4字节： int , float  
8字节： long , double

编码与中文：  
Unicode/GBK： 中文2字节  
UTF-8： 中文通常3字节，在拓展B区之后的是4字节  
综上，中文字符在编码中占用的字节数一般是2-4个字节。

C语言中short通常是2字节，具体可能和平台编译器相关。

### int与integer的区别

一、区别：

1.Integer是int的包装类，int则是java的一种基本的数据类型；

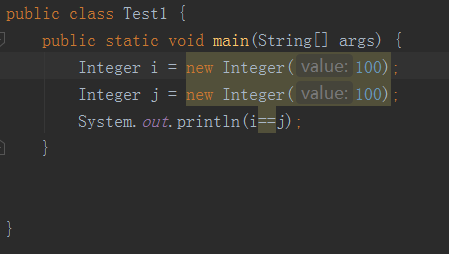
2.Integer变量必须实例化之后才能使用，而int变量不需要实例化；

3.Integer实际是对象的引用，当new一个Integer时，实际上生成一个指针指向对象，而int则直接存储数值

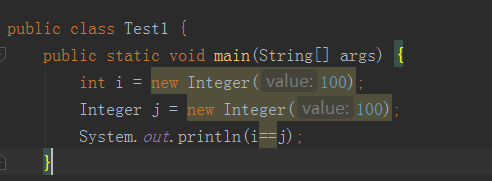
4.Integer的默认值是null，而int的默认值是0。

二、Integer和int的比较

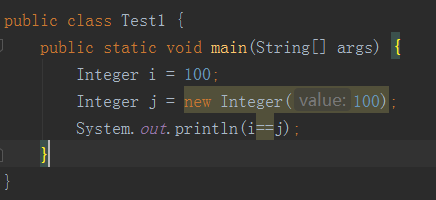
　　1.由于Integer实际是对一个Integer对象的引用，所以两个通过new生成的Integer变量永远是不相同的，因为New生成的是两个不同的对象，其内存地址不同。下面运行的结果为false



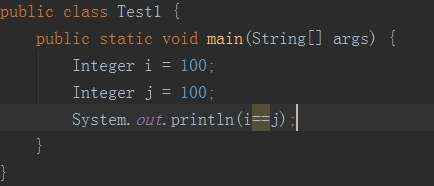
　　2.Integer变量和int变量进行比较时，只要两个变量的值相等，则结果就为True，(因为包装类Integer和基本数据类型比较的时候，java会自动拆箱为int，然后进行比较，实际上就是两个int变量进行比较)，下面运行的结果为true



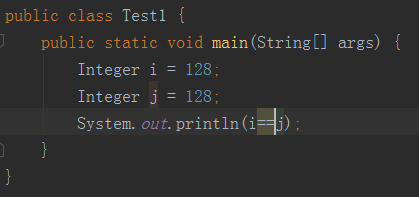
　　3.非new生成的Integer变量和new Integer生成的Integer变量比较的时候，结果为false(因为非new生成的Integer变量指向的是Java常量池中的对象，而new出来的对象指向的是堆中新建的对象，两者内存地址不同)，下面返回的是false



　　4.两个非new出来的Integer对象，进行比较的时候，如果两个变量的值区间在-127~128之间的时候，则返回的结果为true，如果两个变量的变量值不在这个区间，则比较的结果为false。下面返回的是true



　　下面返回的是false



三、java 基本类型与引用类型的区别：

　　1.基本数据类型保存原始值，引用数据类型保存的是引用值(引用值就是指在对象中所处的地理位置)

### 谈谈对java多态的理解

<https://www.cnblogs.com/eyesCentre/archive/2019/04/12/10697487.html>

1.什么是java的多态

浏览了别人博客中的一些介绍多态的文章，发现大家的描述有点不一样，主要区别在于是否把方法的重写算做多态。一种我比较认同的说法如下：

多态分为两种

a. 编译时多态：方法的重载；

b. **运行时多态**：JAVA运行时系统根据调用该方法的实例的类型来决定选择调用哪个方法则被称为运行时多态。（我们平时说得多的事运行时多态，所以多态主要也是指运行时多态）；

上述描述认为重载也是多态的一种表现，不过多态主要指运行时多态。

2.运行时多态

**a. 面向对象的三大特性**：封装、继承、多态。从一定角度来看，封装和继承几乎都是为多态而准备的。这是我们最后一个概念，也是最重要的知识点。

**b. 多态的定义**：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

**c. 实现多态的技术称为**：动态绑定（dynamic binding），是指在**执行期间**判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

**d. 多态的作用**：消除类型之间的耦合关系。

**e. 现实中，关于多态的例子不胜枚举**。比方说按下 F1 键这个动作，如果当前在 Flash 界面下弹出的就是 AS 3 的帮助文档；如果当前在 Word 下弹出的就是 Word 帮助；在 Windows 下弹出的就是 Windows 帮助和支持。同一个事件发生在不同的对象上会产生不同的结果。

下面是多态存在的三个必要条件，要求大家做梦时都能背出来！

多态存在的三个必要条件  
一、要有继承；  
二、要有重写；  
三、父类引用指向子类对象。

**多态的好处**：

1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。  
2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。  
3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图8.3 所示。图中超类Shape规定了两个实现多态的接口方法，computeArea()以及computeVolume()。子类，如Circle和Sphere为了实现多态，完善或者覆盖这两个接口方法。  
4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。  
5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

### String、StringBuffer、StringBuilder区别

**String** 类不可变，内部维护的char[] 数组长度不可变，为final修饰，String类也是final修饰，不存在扩容。字符串拼接，截取，都会生成一个新的对象。频繁操作字符串效率低下，因为每次都会生成新的对象。

**StringBuilder** 类内部维护可变长度char[] ， 初始化数组容量为16，存在扩容， 其append拼接字符串方法内部调用System的native方法，进行数组的拷贝，不会重新生成新的StringBuilder对象。

它是非线程安全的字符串操作类， 其每次调用 toString方法而重新生成的String对象，不会共享StringBuilder对象内部的char[]，会进行一次char[]的copy操作。

**StringBuffer** 类内部维护可变长度char[]， 基本上与StringBuilder一致，但其为线程安全的字符串操作类，大部分方法都采用了Synchronized关键字修改，以此来实现在多线程下的操作字符串的安全性。

其toString方法而重新生成的String对象，会共享StringBuffer对象中的toStringCache属性（char[]），但是每次的StringBuffer对象修改，都会置null该属性值。

### 什么是内部类？内部类的作用

**什么是内部类**：  
将一个类定义在另一个类里面或者一个方法里面，这样的类称为内部类。

**内部类的作用：**

1.成员内部类  
成员内部类可以无条件访问外部类的所有成员属性和成员方法（包括private成员和静态成员）。  
当成员内部类拥有和外部类同名的成员变量或者方法时，会发生隐藏现象，即默认情况下访问的是成员内部类的成员。

2.局部内部类  
局部内部类是定义在一个方法或者一个作用域里面的类，它和成员内部类的区别在于局部内部类的访问仅限于方法内或者该作用域内。

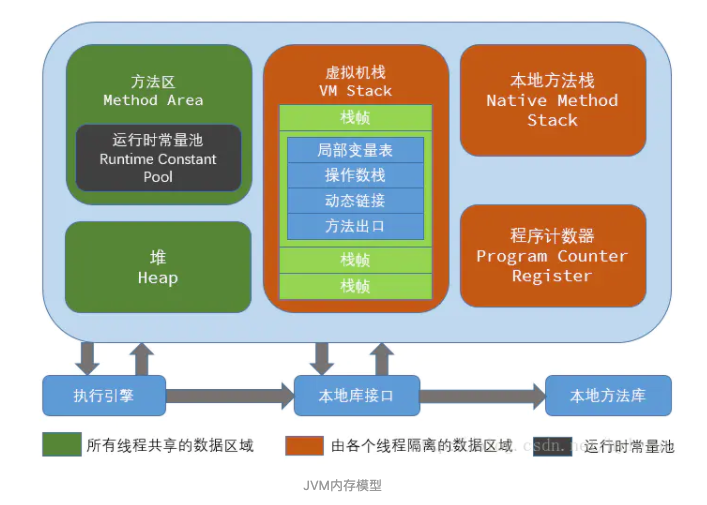
3.匿名内部类  
匿名内部类就是没有名字的内部类

4.静态内部类  
指被声明为static的内部类，他可以不依赖内部类而实例，而通常的内部类需要实例化外部类，从而实例化。静态内部类不可以有与外部类有相同的类名。不能访问外部类的普通成员变量，但是可以访问静态成员变量和静态方法（包括私有类型）  
一个 静态内部类去掉static 就是成员内部类，他可以自由的引用外部类的属性和方法，无论是静态还是非静态。但是不可以有静态属性和方法、

**作用**  
1.每个内部类都能独立的继承一个接口的实现，所以无论外部类是否已经继承了某个(接口的)实现，对于内部类都没有影响。内部类使得多继承的解决方案变得完整， 　　  
2.方便将存在一定逻辑关系的类组织在一起，又可以对外界隐藏。 　　  
3.方便编写事件驱动程序 　　  
4.方便编写线程代码

### jvm内存模型

<https://www.jianshu.com/p/0ecf020614cb>





### 抽象类和接口区别

一 什么时候使用抽象类和接口

* 如果你拥有一些方法并且想让它们中的一些有默认实现，那么使用抽象类吧。
* 如果你想实现多重继承，那么你必须使用接口。由于Java不支持多继承，子类不能够继承多个类，但可以实现多个接口。因此你就可以使用接口来解决它。
* 如果基本功能在不断改变，那么就需要使用抽象类。如果不断改变基本功能并且使用接口，那么就需要改变所有实现了该接口的类。

<https://www.cnblogs.com/songhuiqiang/p/10647835.html>



### 抽象类的意义

1. 因为抽象类不能实例化对象，所以必须要有子类来实现它之后才能使用。这样就可以把一些具有相同属性和方法的组件进行抽象，这样更有利于代码和程序的维护。
2. 当又有一个具有相似的组件产生时，只需要实现该抽象类就可以获得该抽象类的那些属性和方法。

所以，Java中抽象类对于代码的维护和重用有很好的帮助，也是Java面向对象的一个重要体现

### 在使用中抽象类和接口我们该如何选择？

\* 如果你拥有一些方法并且想让它们中的一些有默认实现，那么使用抽象类吧。

\* 如果你想实现多重继承，那么你必须使用接口。由于Java不支持多继承，子类不能够继承多个类，但可以实现多个接口。因此你就可以使用接口来解决它。

\* 如果基本功能在不断改变，那么就需要使用抽象类。如果不断改变基本功能并且使用接口，那么就需要改变所有实现了该接口的类。

### 抽象类是否可以没有方法和属性？

答案是肯定的，可以。

    抽象类中可以没有抽象方法，但有抽象方法的一定是抽象类。所以，java中 抽象类里面可以没有抽象方法。注意即使是没有抽象方法和属性的抽象类，也不能被实例化。

### 泛型中extends和super的区别

<https://blog.csdn.net/qq_36898043/article/details/79655309>

<? extends T>和<? super T>是Java泛型中的“通配符（Wildcards）”和“边界（Bounds）”的概念。

* <? extends T>：是指 “上界通配符（Upper Bounds Wildcards）”
* <? super T>：是指 “下界通配符（Lower Bounds Wildcards）”

PECS原则

最后看一下什么是PECS（Producer Extends Consumer Super）原则，已经很好理解了：

* 频繁往外读取内容的，适合用上界Extends。
* 经常往里插入的，适合用下界Super。

### 父类的静态方法能否被子类重写

<https://www.cnblogs.com/sunbr/p/11545010.html>

静态：

　　在编译时所分配的内存会一直存在（不会被回收），直到程序退出内存才会释放这个空间，在实例化之前这个方法就已经存在于内存，跟类的对象没什么关系。子类中如果定义了跟父类相同名称、相同的形参列表、相同返回值类型的静态方法，就满足重写的条件，于是在内存中又分配了一块给子类的静态方法，重写了父类方法。

父类代码

[制代码](javascript:void(0);)

1 public class Fu {

2 public static void show() {

3 System.out.println("父类的静态方法");

4 }

5 public void method() {

6 System.out.println("父类的一般方法");

7 }

8 }

[制代码](javascript:void(0);)

子类代码

[制代码](javascript:void(0);)

public class Zi extends Fu {

public static void main(String[] args) {

Fu fu = new Zi();

fu.show();

fu.method();

}

public static void show() {

System.out.println("子类的静态");

}

public void method() {

System.out.println("子类的一般方法");

}

}

[制代码](javascript:void(0);)

 输出结果是：

 父类的静态方法

 子类的一般方法

　　当父类引用指向子类对象，只会调用父类的静态方法，此行为并不具有多态性！所以子类重写父类的静态方法，并没有实际意义！

### 进程和线程的区别

<https://www.jianshu.com/p/2dc01727be45>

<https://www.cnblogs.com/zhehan54/p/6130030.html>

一、进程与线程的区别：

1、进程是资源分配的最小单位，线程是程序执行的最小单位（资源调度的最小单位）  
2、进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。  
而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。  
3、线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。  
4、但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。  
参考：<https://www.cnblogs.com/zhehan54/p/6130030.html>

#### 二、进程与线程的资源：

1、堆与栈

堆：　是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。  
栈：是个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立，因此，栈是　thread safe的。操作系统在切换线程的时候会自动的切换栈，就是切换　ＳＳ／ＥＳＰ寄存器。栈空间不需要在高级语言里面显式的分配和释放。

2、其他

线程共享的环境包括：进程代码段、进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)、进程打开的文件描述符、信号的处理器、进程的当前目录和进程用户ID与进程组ID。  
进程拥有这许多共性的同时，还拥有自己的个性。有了这些个性，线程才能实现并发性。这些个性包括：  
1.线程ID  
每个线程都有自己的线程ID，这个ID在本进程中是唯一的。进程用此来标  
识线程。  
2.寄存器组的值  
由于线程间是并发运行的，每个线程有自己不同的运行线索，当从一个线  
程切换到另一个线程上时，必须将原有的线程的寄存器集合的状态保存，以便  
将来该线程在被重新切换到时能得以恢复。  
3.线程的堆栈  
堆栈是保证线程独立运行所必须的。  
线程函数可以调用函数，而被调用函数中又是可以层层嵌套的，所以线程  
必须拥有自己的函数堆栈，使得函数调用可以正常执行，不受其他线程的影  
响。  
4.错误返回码  
由于同一个进程中有很多个线程在同时运行，可能某个线程进行系统调用  
后设置了errno值，而在该线程还没有处理这个错误，另外一个线程就在此时  
被调度器投入运行，这样错误值就有可能被修改。  
所以，不同的线程应该拥有自己的错误返回码变量。  
5.线程的信号屏蔽码  
由于每个线程所感兴趣的信号不同，所以线程的信号屏蔽码应该由线程自  
己管理。但所有的线程都共享同样的信号处理器。  
6.线程的优先级  
由于线程需要像进程那样能够被调度，那么就必须要有可供调度使用的参  
数，这个参数就是线程的优先级。  
参考：<https://blog.csdn.net/shuilan0066/article/details/7683315>

3、进程与线程的同步

进程：无名管道、有名管道、信号、共享内存、消息队列、信号量  
进程：互斥量、读写锁、自旋锁、线程信号、条件变量

### final，finally，finalize的区别

<https://www.cnblogs.com/ktao/p/8586966.html>

1. final

　　在java中，final可以用来修饰类，方法和变量（成员变量或局部变量）。下面将对其详细介绍。

1.1 修饰类

　　当用final修饰类的时，表明该类不能被其他类所继承。当我们需要让一个类永远不被继承，此时就可以用final修饰，但要注意：

final类中所有的成员方法都会隐式的定义为final方法。

1.2 修饰方法

使用final方法的原因主要有两个：

　　(1) 把方法锁定，以防止继承类对其进行更改。

　　(2) 效率，在早期的java版本中，会将final方法转为内嵌调用。但若方法过于庞大，可能在性能上不会有多大提升。因此在最近版本中，不需要final方法进行这些优化了。

final方法意味着“最后的、最终的”含义，即此方法不能被重写。

**注意：若父类中final方法的访问权限为private，将导致子类中不能直接继承该方法，因此，此时可以在子类中定义相同方法名的函数，此时不会与重写final的矛盾，而是在子类中重新地定义了新方法。**

class A{

private final void getName(){

}

}

public class B extends A{

public void getName(){

}

public static void main(String[]args){

System.out.println("OK");

}

}

1.3 修饰变量

 　　final成员变量表示常量，只能被赋值一次，赋值后其值不再改变。类似于C++中的const。

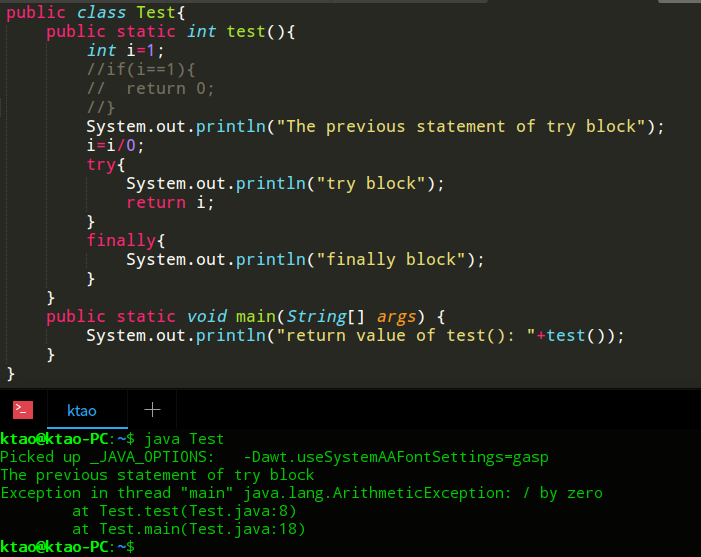
　　当final修饰一个基本数据类型时，表示该基本数据类型的值一旦在初始化后便不能发生变化；如果final修饰一个引用类型时，则在对其初始化之后便不能再让其指向其他对象了，但该引用所指向的对象的内容是可以发生变化的。本质上是一回事，因为引用的值是一个地址，final要求值，即地址的值不发生变化。

　　final修饰一个成员变量（属性），必须要显示初始化。这里有两种初始化方式，一种是在变量声明的时候初始化；第二种方法是在声明变量的时候不赋初值，但是要在这个变量所在的类的所有的构造函数中对这个变量赋初值。

当函数的参数类型声明为final时，说明该参数是只读型的。即你可以读取使用该参数，但是无法改变该参数的值。

2. finally

 　　finally作为异常处理的一部分，它只能用在try/catch语句中，并且附带一个语句块，表示这段语句最终一定会被执行（不管有没有抛出异常），经常被用在需要释放资源的情况下。（×）（这句话其实存在一定的问题）



3. finalize

　　finalize()是在java.lang.Object里定义的，也就是说每一个对象都有这么个方法。这个方法在gc启动，该对象被回收的时候被调用。其实gc可以回收大部分的对象（凡是new出来的对象，gc都能搞定，一般情况下我们又不会用new以外的方式去创建对象），所以一般是不需要程序员去实现finalize的。   
特殊情况下，需要程序员实现finalize，当对象被回收的时候释放一些资源，比如：一个socket链接，在对象初始化时创建，整个生命周期内有效，那么就需要实现finalize，关闭这个链接。   
　　使用finalize还需要注意一个事，调用super.finalize();

### 序列化的方式

<https://blog.csdn.net/ahuqihua/article/details/81331316>

**序列化和反序列化**  
序列化：可以将对象转化成一个字节序列，便于存储。  
反序列化：将序列化的字节序列还原  
优点：可以实现对象的"持久性”， 所谓持久性就是指对象的生命周期不取决于程序。  
  
序列化需要：  
所需类：ObjectInputStream和ObjectOutputStream  
方法： readObject()和writeObject();

**序列化方式一： 实现Serializable接口(隐式序列化)**通过实现Serializable接口，这种是隐式序列化(不需要手动)，这种是最简单的序列化方式，会自动序列化所有非static和 transient关键字修饰的成员变量。

**序列化方式二：实现Externalizable接口。(显式序列化)**  
Externalizable接口继承自Serializable, 我们在实现该接口时，必须实现writeExternal()和readExternal()方法，而且只能通过手动进行序列化，并且两个方法是自动调用的，因此，这个序列化过程是可控的，可以自己选择哪些部分序列化

**序列化方式三：实现Serializable接口+添加writeObject()和readObject()方法。(显+隐序列化)**  
如果想将方式一和方式二的优点都用到的话，可以采用方式三， 先实现Serializable接口，并且添加writeObject()和readObject()方法。注意这里是添加，不是重写或者覆盖。但是添加的这两个方法必须有相应的格式。

* 1，方法必须要被private修饰                                ----->才能被调用
* 2，第一行调用默认的defaultRead/WriteObject(); ----->隐式序列化非static和transient
* 3，调用read/writeObject()将获得的值赋给相应的值  --->显式序列化

### Serializable 和Parcelable 的区别

<https://www.cnblogs.com/berylqliu/p/6261479.html>

Parcelable和Serializable的区别：  
    android自定义对象可序列化有两个选择一个是Serializable和Parcelable  
    一、对象为什么需要序列化  
        1.永久性保存对象，保存对象的字节序列到本地文件。  
        2.通过序列化对象在网络中传递对象。  
        3.通过序列化对象在进程间传递对象。  
    二、当对象需要被序列化时如何选择所使用的接口  
        1.在使用内存的时候Parcelable比Serializable的性能高。  
        2.Serializable在序列化的时候会产生大量的临时变量，从而引起频繁的GC（内存回收）。  
        3.Parcelable不能使用在将对象存储在磁盘上这种情况，因为在外界的变化下Parcelable不能很好的保证数据的持续性。

### 静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？

<https://blog.csdn.net/qdh186/article/details/79807057>

1)先说是否可以被继承

public class One {

//静态属性和静态方法是否可以被继承？

public static String one\_1 = "one";

public static void oneFn() {

System.out.println("oneFn");

}

}

public class Two extends One{

//空

}

public class MyTest {

//静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？

public static void main(String[] args) {

One one = new Two();

one.oneFn();

String one\_1 = One.one\_1;

System.out.println("One.one\_1>>>>>>>"+one\_1);

String one\_12 = one.one\_1;

System.out.println("one.one\_1>>>>>>>"+one\_12);

}

}

//打印结果如下

oneFn

One.one\_1>>>>>>>one

one.one\_1>>>>>>>one

以上打印说明了：  
父类的静态属性和方法可以被子类继承

2）是否可以被重写？  
再来看这段代码

public class One {

//静态属性和静态方法是否可以被重写？以及原因？

public static String one\_1 = "one";

public static void oneFn() {

System.out.println("oneFn");

}

}

public class Two extends One {

public static String one\_1 = "two";

public static void oneFn() {

System.out.println("TwoFn");

}

}

public class MyTest {

//静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？

public static void main(String[] args) {

One one = new Two();

one.oneFn();

String one\_1 = One.one\_1;

System.out.println("One.one\_1>>>>>>>"+one\_1);

String one\_12 = one.one\_1;

System.out.println("one.one\_1>>>>>>>"+one\_12);

}

}

//打印结果如下

//oneFn

//One.one\_1>>>>>>>one

//one.one\_1>>>>>>>one

以上打印说明  
当父类的引用指向子类时，使用对象调用静态方法或者静态变量，是调用的父类中的方法或者变量。并没有被子类改写。  
所以我认为不可以被子类重写。

3）原因  
static修饰函数/变量时，其实是全局函数/变量，它只是因为java强调对象的要挂，它与任何类都没有关系。靠这个类的好处就是这个类的成员函数调用static方法不用带类名。

注意：static关键字可以用修饰代码块.static代码块可以置于类中的任何一个位置，并可以有多个static代码块。在类初次被加载时，会按照静态代码块的顺序来执行，并且只会执行一次。

### 静态内部类的设计意图

在成员内部类中要注意两点：

1. 成员内部类中不能存在任何static的变量和方法；
2. 成员内部类是依附于外围类的，所以只有先创建了外围类才能够创建内部类。

**静态内部类与非静态内部类之间存在一个最大的区别：**

非静态内部类在编译完成之后会隐含地保存着一个引用，该引用是指向创建它的外围内，但是静态内部类却没有。

没有这个引用就意味着：

1. 它的创建是不需要依赖于外围类的。
2. 它不能使用任何外围类的非static成员变量和方法。

### 闭包和局部内部类的区别

比较常见的对闭包的定义是：引用了自由变量的一段代码或函数，被引用的自由变量和函数（一段代码）共同存在，即使离开了创造它的环境