###### Java虚拟机有多少个？？

在Android系统中，java虚拟机到底有多少个？？

例如：整个系统只有一个虚拟机？还是一个进程对应一个虚拟机。

解释：名词：Java 虚拟机，jvm实例。每个java程序都是JVM实例。

Java虚拟机对程序具有解释字节码和执行程序的能力。简单来说，jvm 实例就是一个解释器。它能解释并执行指令。与Linux 中的shell 脚本解释器一样。

###### 当我们运行一个java程序时，jvm 虚拟机是怎么工作的？？

当我们运行Run命令时，系统会先复制一个jvm 实例（可以看成是一个对象）。然后这个jvm实例会找到程序入口main方法，并执行里面的代码逻辑。

（具体jvm里面是怎么实现工作的我们就不知道了，不要试图是探索它，这是几十代人的结晶）

注：因此，当我们分析一个新的程序时，重要的是找到程序的入口。

###### jvm 实例 是否有运算能力？？

例如：对变量的自加 i++ ，自加运算时。在不离开jvm内存中，它是否能对变量自加？还是说必须依靠CPU的计算能力？？

答：目测来说，java虚拟机，是具有运算能力的。原因是：据说，jvm 具有自己的精简指令。

但是，我们根据数字电路知识知道，所有的软件都必须依赖硬件。只有计数器才具有计算的能力。所有的命令的执行，都是二进制的相加。而整个软件系统的核心和 计算器 之间的核心就是 指令集。

Jvm 也有指令集，因此，按道理来说，jvm 是具有运算能力的

###### 如何理解Android 中的AMS

我们知道AMS 可以启动 或 停止 一个进程。还可以进程的保活。

AMS 是怎么管理Activity、service组件的？？这些组件的对象是在哪个进程中的？？

首先，强调一点：AMS只是一个对象，java中一切皆对象。我们知道AMS只是System\_server进程中的一个stub 类型的对象。具体这个对象是怎么被线程调用的，我们不分析。

进程与进程之间是独立，而且进程是在底层的，就连jvm 都是用C++写的。更直接点， jvm 其实也是native 程序。也就是说AMS 凭什么管理其他进程？？？

答案是：管理不了，它只是提供了一个接口，从它是stub 对象就可以知道，它是利用了Binder通讯，至于它找上了谁，我们无法得知。同理，任何进程的四大组件，都是在自己的进程中生成对象的。它们只是在AMS中有一个影子，AMS通过这个影子，控制了它们的生命周期。

AMS能管理的是四大组件影子(代理)。

###### 内核 和 驱动 的关系

###### Android 系统应该关注那些进程？？？

Jvm虚拟机（包括Zygote进程和 Zygote 分裂出来的 system\_server进程）

Init 进程 以及和它启动的native 进程。

Kernel 层

注：kthread 是kernel层的鼻祖，init 是native层的鼻祖，zygote是java层的鼻祖。

###### 如何理解Android里面的handler?? 引发出如何阅读Android 源码

###### Android 的主线程也用了Looper.loop为什么不会阻塞？？？

* 1. 首先要知道4个类，Message 、Loop、MessageQueue、Handler
  2. 名词：消息投递。

1. 别的线程来投递
2. 本身线程死循环处理。
   1. 不同线程，能使用同一个handler 的原因，所有对象都是在堆内存的，线程的交互，就转化为——》拿到对象的引用。
   2. 主线程也使用了Looper.loop 为什么不会ANR？？？？消息驱动

如何理解系统

从源码（理解源码各个模块的功能和目录位置）——》到编译产物的目录—》烧录到emmc（理解分区概念）---》再到系统的启动（理解挂载概念）--🡪最后到系统启动完成（理解目录树概念）

###### 如何阅读android源码不懵逼？？

我们知道java中一切皆对象。要有对象，就必须先有类。阅读源码，其实就是了解每个类的功能（即接口的使用，学习我们能用这些方法做什么）。而往往对于一个功能，可能要许多类来辅助完成、以及各种继承关系。在对象与对象 、对象的引用 和 类的概念，不断的伦换、交替之间，我们很容易迷失自己。有时候，会突然发现，我为什么来阅读这个类？？我到底要干什么呢？？？根本就想不起，阅读的目的是什么。

**改变思维，用对象、进程的思维来阅读源码。**

**我们知道，系统是由许许多多 进程 构成的。进程就是——不断执行代码的线程在跑。线程在跑，就会调用对象的方法。**

**因此，我们要有 进程 和 对象 的观念。**

平时，我们通常因为项目时间太紧，为了达到某一个功能。直接使用某些类。因为目的性够强，我们反而不容易迷茫。大多数迷茫的情况，通常是我们自动想去学习一些新知识时。

阅读源码，当我们感觉到迷茫时，一定要及时退出阅读。此时，大脑已达到了接受的最大程序。放松、放松。我们来做一些回忆。

我是谁？？

我在哪里？？

我在干什么？？

我原来想干什么？？

我是谁？？告诉自己，你叫什么名字。 例如：我叫XXX

我在哪里？？告诉自己，你正在阅读那个类？ 例如：我在阅读XX类，XX方法。

我在干什么？？告诉自己，你怎么跑到这个类来的？为什么跳转 到这个类？ 例如：我上一个类是XX类，因为上一个类，用到了XX功能，我又跑到这个类来看。

我原来想干什么？？告诉自己，你最初想要了解什么。例如：我想了解XX功能。结果，看着，看着。跑到这里来了。

###### 我目录了解了那些android源码 或者说模块。

System\_server

SystemUI

shell脚本

###### C++中数据类型

基础数据类型、数组、结构体、类

常见名词：

变量（一般指基础数据类型 修饰的变量） 、指针变量、函数指针、引用变量。

1. 引用变量 占会内存吗？？

有引用，就会有“引用变量”一词。引用变量不占用内存。所以当我们定义一个引用变量后，取地址指针比较，会发现两者数值相同。

(思考：无论是何种类型的变量，它们是否占内存？？答案是：不占，变量名和它的值。不要和指针变量混淆了。)

指针 在函数调用中，也是值传递。为什么它能在函数中，交换两个字，是因为它解指针了。

如何判断呢？我们可以，修改指针的地址。

1. 指针 和 引用？？？？

指针也是值传递，传递的是地址。为什么这么说，是因为C语言中本身就有指针的概念，而引用是在C++中才有的。

对基础数据类型的分类：(和java的四类8种类似)

1. 整型-------int 、short、long、long long
2. bool
3. char
4. 浮点型------float 和 double

C++中，对于不同类型的数据，占用的字节不同。C++只会规范，至少占多少个字节。无法准确规定占用多少字节。

注意：sizeof 是操作符。不是函数。new 是关键字。

思考：操作符 和 关键字 是不是同一个概念？？还是两个不同的东西。

关键字就是已被:语言本身使用, 不能作其它用途使用的字

操作符是在表达式中用于连接不同对象的运算符

思考：操作符重载中，能否对两个不同class 对象相加？？

###### C++ 中的--🡪指针、引用 和 void 类型

* 指针类型的书写格式为*T \**，其中*T*为某种数据类型。
* 引用类型的书写格式为*T &*，其中*T*为某种数据类型。
* void代表空类型，也就是无类型。这种类型只能用于定义指针变量，比如*void\**。当我们确实不关注内存中存储的数据到底是什么类型的话，就可以定义一个void\*类型的指针来指向这块内存。
* C++11开始，空指针由新关键字*nullptr*[[④]](file:///C:\\Users\\innost\\workspace\\Android-Internal\\%E4%B8%93%E9%A2%98%E5%8D%B7\\%E8%99%9A%E6%8B%9F%E6%9C%BA%E4%B8%93%E9%A2%98\\C++11%E5%92%8C%E7%BC%96%E8%AF%91%E6%8A%80%E6%9C%AF%E7%AF%87\\%E7%AC%AC%E4%BA%94%E7%AB%A0C++11%E5%AD%A6%E4%B9%A0.htm" \l "_ftn4" \o "" \t "_blank)表示，类似于Java中的*null*

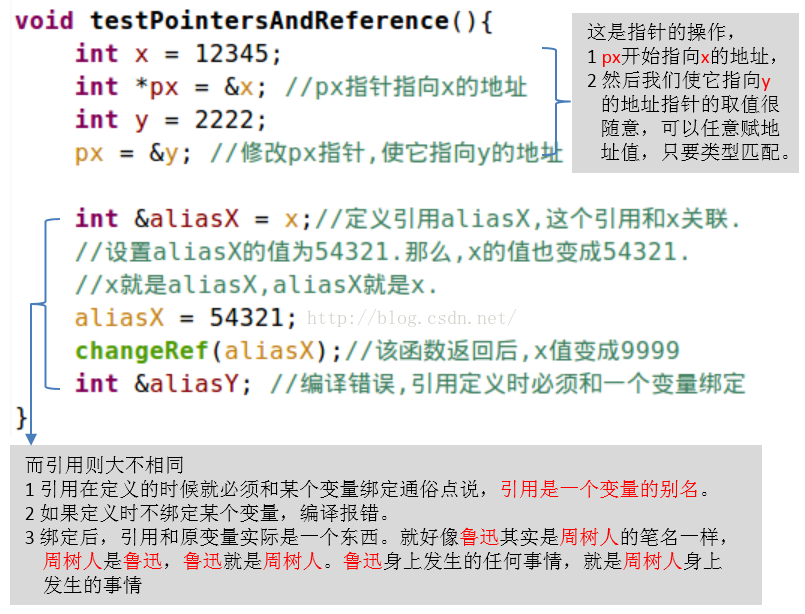
如何理解指针。记住这句话：指针就是内存地址

指针可以指向数据 也可以 指向代码。由此，指针引发的一些名词：指针变量、函数指针。

**如何在C++中，声明——函数指针变量**

* 对于数据类型的指针，*解引用*意味着获取对应地址中内存的内容。
* 对于函数指针，*解引用*意味着调用这个函数。

引用



注意：任何变量都是不占内存的。是他们的值占用了内存。严格来说，变量是一种编程的称呼。而引用变量，也是一样的，是一种别名。

C++使用引用的目的 是引用传递比指针方便。

由此引发一个名词：引用传递 和值传递。

* 引用只是变量的别名。由于是别名，所以C++要求在定义引用型变量时就必须将它和实际变量绑定。
* 引用型变量绑定实际变量之后，这两个变量（原变量和它的引用变量）其实就代表同一个东西了。图7中（1）以鲁迅为例，“鲁迅”和“周树人”都是同一个人。

字符 和 字符串

思考：在C语言中字符串 是怎么表示的？？ 有没有字符指针变量？？字符串指针变量？？？

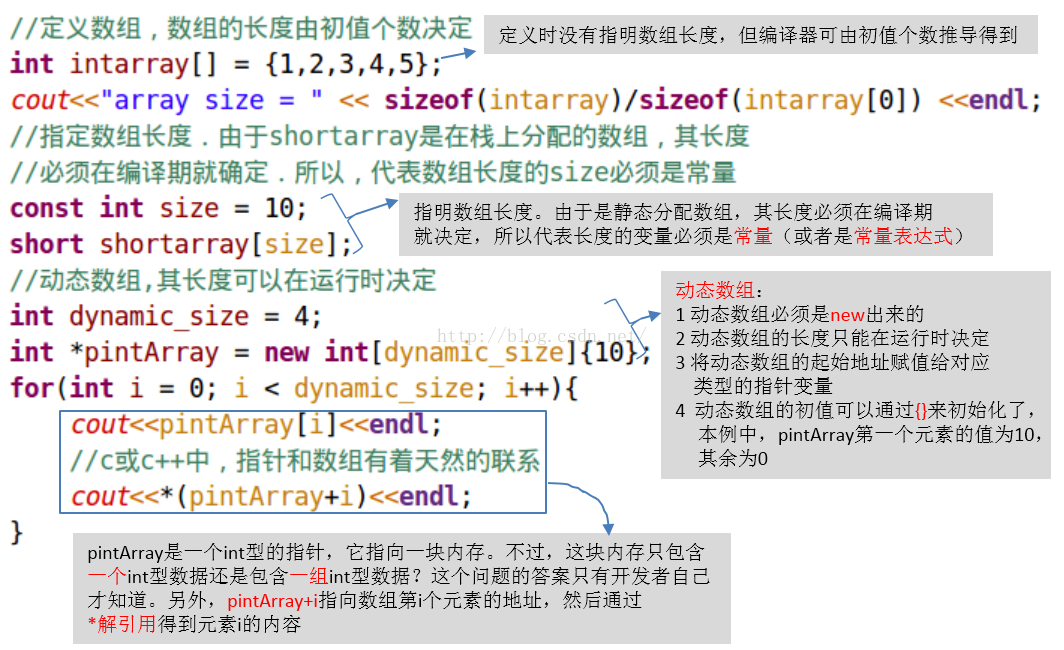
指针 可以指向一个数组的首地址。

**一个指针变量，它是指向一个数组 还是 字符数组（即字符串） 是无法区别的。(是否正确)**

char \*argv：理解为字符串

char \*\*argv：理解为字符串指针

char \*argv[]：字符串指针数组



什么是动态数组？？？数组的长度不是不可以变得么？既然数组也可以new 创建出来，是否可以认为数组，也是一种类？？？

###### C++ 编译中的一些名词：

Clang 是工具包

一个Makefile 是命令的组合（都有哪些命令？语法？？）

GCC编译器，C++用g++编译器

预编译、编译、链接

源文件、编译目标文件obj、二进制可执行程序

###### 类

类在java中的一些概念：

怎么定义一个类？

内部类、匿名内部类、finaly、修饰符、static

子类、父类、重载、覆写、super、属性、构造方法、方法。对象的创建 new 、反射。

父类声明，子类实现。向上自动转型，向下强制转型。泛型编程

包名

C++中有

命名空间——命名空间 和 类有什么区别？？名词：作用域??

命名空间的意义是什么？？？能否定义两个相同的类名？？能，依靠命名空间区别。那不是每定义一个类，就要使用一个命名空间？？？

命名空间类似于java中的package

只有在头文件中生命的，外部的c++文件才可以用，否则只能在文件中使用。

注意：因为任何头文件中都可以使用namespace 来定义命名空间。而任何C++文件都可以引用头文件，因此，在一个项目中不能存在相同的命名空间。

C++中的标准库

一些名词：

成员变量、构造函数、成员函数、析构函数

Java中，我们关心的也是这三方面：

* 1. 构造方法
  2. 成员变量
  3. 方法

同时，C++对类的声明 和 实现分两步走。一个是头文件，一个是实现文件。而java 简单了好多，即是声明，也是实现。

相比于java，而C++中，特有的：

对于构造器：

拷贝构造函数

移动构造函数

**(如果扣去构造两字，那就是拷贝函数，移动函数)**

拷贝函数的格式必须为：

类名(const 类& other)

例如：Base(const Base& other)

拷贝构造，即从一个已有的对象拷贝其内容，然后构造出一个新的对象。

注意：拷贝构造函数中，一定要注意使用何种拷贝方式：值拷贝、内容拷贝（即所谓的：浅拷贝、深拷贝）。一般来说，涉及到指针变量的，都到深拷贝。

**对于普通函数多了以下两项：**

1. **拷贝赋值函数 (什么鬼？又拷贝又赋值的)**
2. **移动赋值函数**

注意，const是C++中的常量修饰符，与Java的final类似。

C写程序的时候，是面向过程的思维方法，考虑的是函数和函数之间的调用和跳转关系。C++出现后，我们看待问题和解决问题的思路发生了很大的变化，更多考虑是设计合适的类并处理对象和对象之间的关系。

思考：

C++中那么都构造函数，我们在java中是怎么区别构造方法 和普通方法的呢？？有无返回值。

关于C++ 和java中一些关键字的总结：

Java 中

static 可以修饰属性、方法。表示的是类方法、类变量。

final 可以修饰属性 和 方法。

C++中

static

const ---------------->对标java中的final

const 是constant 的英文缩写，译文：恒定不变的。

**研究C++的时候，最大的难处是：如果不涉及class类的时候，怎么表达这些常量。总是要不断的去区分。我们现在研究的是C++，那么所有的方式，应该都是通用的。应该并不仅仅是使用在class类上。并不需要区分是带class 还是普通的Cpp文件。(以后再研究)**

Java中this是本对象的的代表。在C++ this 代表的是指针。

既然是指针，那么就无法直接调用方法，必须先解指针。

（引用是变量的别名，指针是地址。函数可以返回一个指针，也可以返回一个引用。当要求返回的是一个引用时，我们要返回的是指针还是对象呢？）

例如：我们声明了一个指针变量。

然后，又声明一个引用，这个引用是这个指针变量的别名。(头疼)

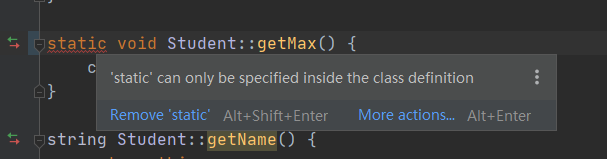
思考：

我们知道，java中的多态的使用是：父类声明，子类实现。但是在C++中，我们一旦声明一个类变量，实例就会被创建。那我们怎么让这个类，具有全局性呢？？头文件里面声明，是不是不会创建对象？？？

C++中父类声明，子类实现 比java 中的复杂。

注意：

Virtual、override、static只能出现在声明中，不能在实现接口里面。



如何禁止函数被子类重写

阻止虚函数被override

C++中，也可以阻止某个虚函数被override，方法和Java类似，就是在函数声明后添加final关键词，比如

virtual void test1(boolean test) final;//如此，test1将不能被派生类override了

思考：像final 关键字的位置能不能换到前面呢？我们习惯java的语法在前面。

思考：java中有向上转型、向下转型。C++的向上转型、和向下转型 又是怎么样的？？

思考：java中子类的创建，会先加载父类。通过代码测试，发现会先调用父类的构造方法。Java中理解这种行为为：类的加载机制。而在C++中，创建子类时，也会先调用父类的构造函数。而在销毁对象时，则相反，先调用自己的，再调用父类的。非常有意思的事情。

**补充内容：**

**如果派生类含有类类型的成员变量时，调用次序将变成：**

**构造函数：基类构造->派生类中类类型成员变量构造->派生类构造**

**析构函数：派生类析构->派生类中类类型成员变量析构->基类析构**

**多重派生的话，基类按照派生列表的顺序/反序构造或析构**

**思考：子类要不要写构造函数？？**

我们知道java中，一旦父类重定义的构造方法，java会强制要求开发者给子类提供构造方法。

C++会不会有这种情况？？？

感慨：我拉个擦，C++的构造函数，真JB 多。**默认构造，普通构造、拷贝构造、移动构造**

**图24的规矩可简单总结为：**

**如果程序员定义了任何一种类型的构造函数（拷贝构造、移动构造，默认构造，普通构造），则编译器将不再隐式创建默认构造函数。**

**如果程序没有定义拷贝（拷贝赋值或拷贝构造）函数或析构函数，则编译器将隐式合成对应的函数。**

**如果程序没有定义移动（移动赋值或移动构造）函数，并且，程序没有定义析构函数或拷贝函数（拷贝构造和拷贝赋值），则编译器将合成对应的移动函数。**

**一脸懵逼**

**我想知道的是：平时写的时候，我一定要写那些？？那些不需要我管？？**

**思考：C++中，如何代码复用？？**

**C#用“base”， Java用“super”，C++用什么？**

**情景是这样的：有时候，我们重写一个方法时，我们不希望再重写父方法里面的逻辑。而且，有些变量只有父类才可见。此时，要怎么调用父类的方法。**

**答案是：A是B的父类，fun()是B继承的A的，在B中调用A的fun()则是A::fun()**

C++中的泛型编程

###### 友元 和 类的前向声明

友元的作用无非是：

**提供一种方式，使得类外某些函数或者某些类能够访问一个类的私有成员变量或成员函数。对被访问的类而言，这些类外函数或类，就是被访问的类的朋友。**

**一些关键字：friend**

**一些名词：**

**友元函数、友元类。**

**思考：友元的两个问题：**

1. **谁是谁的好朋友？？例如：你把别人当朋友，别人并不一定把你当朋友。答案是：狐朋狗友。**
2. **好朋友直接，是不是都能互相访问对方的私有？答案：不能。**
3. **朋友关系不能继承。例如：爸爸的朋友，不是儿子的朋友。但是，如果函数是，父类声明，传的是子类实例的话。则爸爸的朋友，可以访问儿子继承过来的，爸爸中定义的非公开成员，不能访问儿子定义的非公开成员。（注：非公开成员，连儿子都不能访问）**

友元这种玩法也只有在类里面才存在。因为普通的函数，完全可以通过include包含。没有访问权限控制。

谁把别人当朋友，谁就在自己的类里面声明。

友元的使用步骤：

1. 父类声明朋友。关键字：friend
2. 朋友自己实现函数（要求传入父类的形参）
3. 朋友的函数被调用了（传入一个父类的实例。）

**类的前向声明有意思的用法：**

**类的前向声明的用法，即在头文件里进行类的前向声明，在源文件里去包含该类的头文件。**

**思考：**

**我们是否可以这么玩，所以使用到其他类的头文件，我们都不再采用include头文件，而是采用 前向声明声明来玩呢？答案是：不能。**

**理由如下：**

前向声明好处很多，但同时也有限制。以Obj为例，在看到Obj完整定义之前，不能声明Obj类型的变量（包括类的成员变量），但是可以定义Obj引用类型或Obj指针类型的变量。比如，你无法在图26中class Obj类代码之前定义ObjaObj这样的变量。只能定义Obj& refObj或Obj\* pObj。之所以有这个限制，是因为定义Obj类型变量的时候，编译器必须确定该变量的大小以分配内存，由于没有见到Obj的完整定义，所以编译器无法确定其大小，但引用或者指针则不存在此问题。读者不妨一试。

**explicit构造函数**

**类型的隐式转换。。。。。。。**

**感慨：我操，这又是什么鬼？？我现在连构造函数怎么用都不知道了.......**

###### C++中的struct

struct是C语言中的古老成员了，在C中它叫结构体。不过到了C++世界，struct不再是C语言中结构体了，它升级成了class。即C++中的struct就是一种class，它拥有类的全部特征。不过，struct和普通class也有一点区别，那就是struct的成员（包含函数和变量）默认都是public的访问权限。

**我已经疯了，没法学了**