###### 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、引用变量。

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。

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*

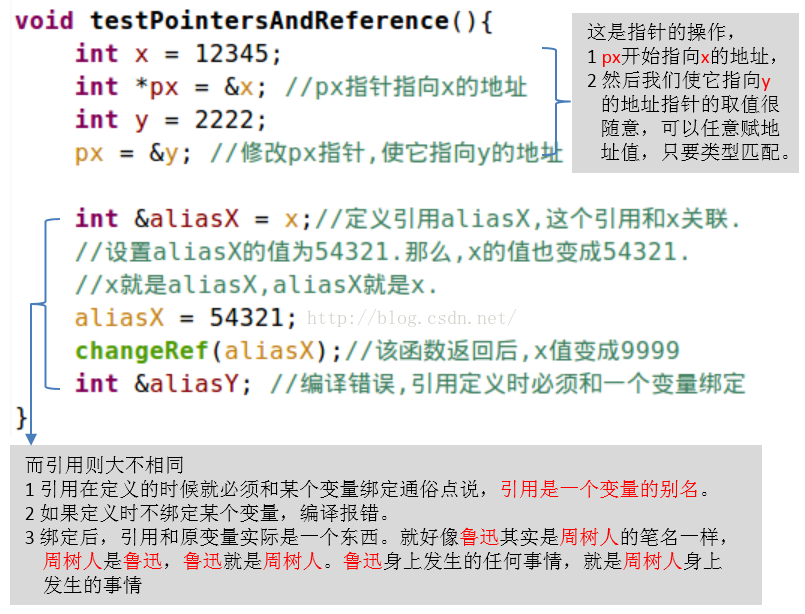
如何理解指针。记住这句话：指针就是内存地址

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？？