**A计划**

目录

[1 A计划说明 1](#_Toc524842449)

[2 Android 1](#_Toc524842450)

[2.1 登录 1](#_Toc524842451)

[3 Java 1](#_Toc524842452)

[3.1 登录 1](#_Toc524842453)

[4 C 和 C++ 2](#_Toc524842454)

[4.1 登录 2](#_Toc524842455)

[5 算法 2](#_Toc524842456)

[5.1 登录 2](#_Toc524842457)

[6 前端 2](#_Toc524842458)

[6.1 登录 2](#_Toc524842459)

[7 实践项目 2](#_Toc524842460)

[7.1 登录 2](#_Toc524842461)

[8 书籍 2](#_Toc524842462)

[8.1 登录 2](#_Toc524842463)

[9 行业流行语 3](#_Toc524842464)

[9.1 登录 3](#_Toc524842465)

[10 常用工具和命令 3](#_Toc524842466)

[10.1 Git 3](#_Toc524842467)

[10.2 Adb 3](#_Toc524842468)

[10.3 Gradle 3](#_Toc524842469)

[10.4 Linux 3](#_Toc524842470)

[10.5 Tomcat 3](#_Toc524842471)

[10.6 Tomcat 3](#_Toc524842472)

# A计划说明

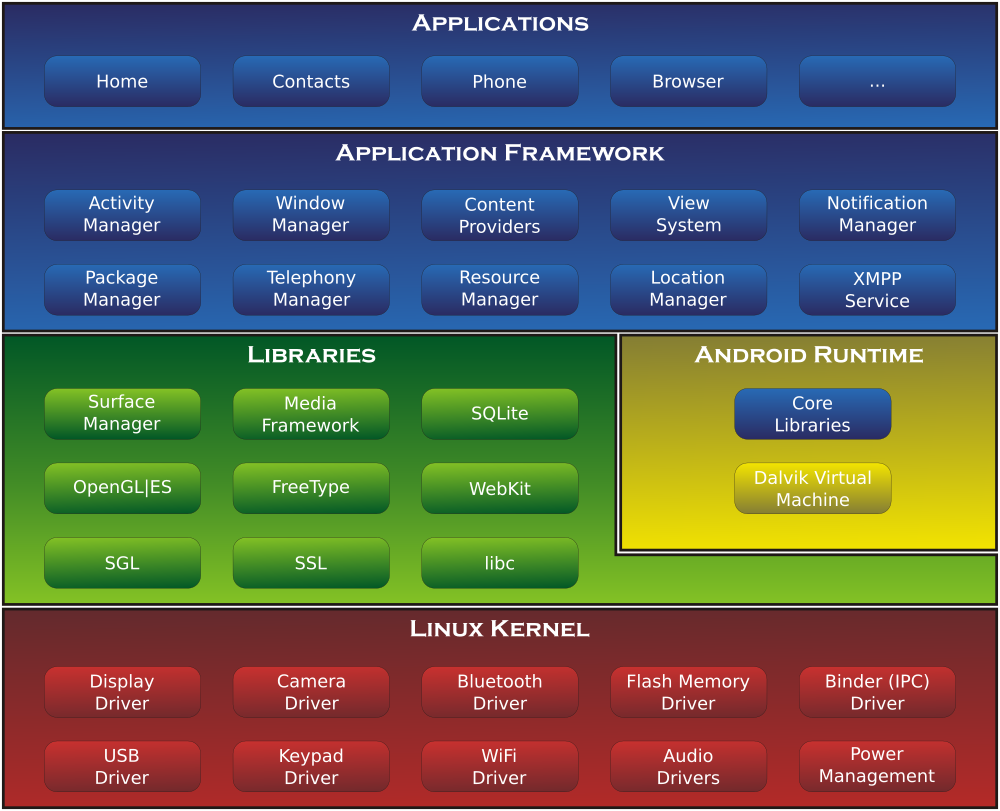
**进阶之路**

# Android

绝技，降龙十八掌

## 问：android 五大应用开发框架是什么?

答：Android采用层次化系统架构，官方公布的标准架构如下图所示。



Android由底层往上分为4个主要功能层，分别是**linux内核层**（Linux Kernel），**系统运行时库层**（Libraries和Android Runtime），**应用程序架构层**（Application Framework）和**应用程序层**（Applications）。

接下来对这几个层面进行逐个分析：

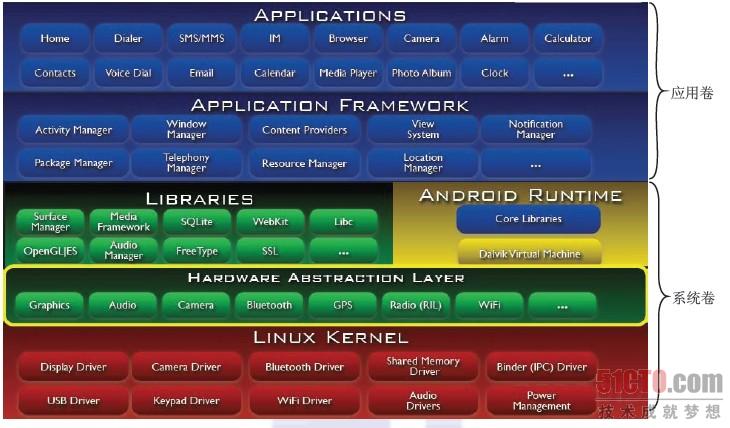
### Linux内核层

Android以Linux操作系统内核为基础，借助Linux内核服务实现硬件设备驱动，进程和内存管理，网络协议栈，电源管理，无线通信等核心功能。Android4.0版本之前基于Linux2.6系列内核，4.0及之后的版本使用更新的Linux3.X内核，并且两个开源项目开始有了互通。Linux3.3内核中正式包括一些Android代码，可以直接引导进入Android。Linux3.4增添了电源管理等更多功能，以增加与Android的硬件兼容性，使Android在更多设备上得到支持。直到现在最新的android6.0仍然继续延用着linux3.4.0，而linux最新的版本已经到了4.3系列，那么为什么android没有继续去更新Linux kernel的版本也是一个值得探讨的课题。   
Android内核 对Linux内核进行了增强，增加了一些面向移动计算的特有功能。例如，**低内存管理器LMK**（Low Memory Keller），**匿名共享内存**（Ashmem）,以及**轻量级的进程间通信Binder机制**等。这些内核的增强使Android在继承Linux内核安全机制的同时，进一步提升了内存管理，进程间通信等方面的安全性。下表列举了Android内核的主要驱动模块：

|  |  |
| --- | --- |
| 驱动名称 | 说明 |
| Android电源管理（Power Management） | 针对嵌入式设备的，基于标准Linux电源管理系统的，轻量级的电源管理驱动 |
| 低内存管理器（Low Memory Keller） | 低内存管理器（Low Memory Keller） 可以根据需要杀死进程来释放需要的内存。扩展了Linux的OOM机制，形成独特的LMK机制 |
| 匿名共享内存（Ashmem） | 为进程之间提供共享内存资源，同时为内核提供回收和管理内存的机制 |
| 日志（Android Logger） | 一个轻量级的日志设备 |
| 定时器（Anroid Alarm） | 提供了一个定时器用于把设备从睡眠状态唤醒 |
| 物理内存映射管理（Android PMEM） | DSP及其他设备只能工作在连续的物理内存上，PMEM用于向用户空间提供 连续的物理内存区域映射 |
| Android定时设备（Android Timed device） | 可以执行对设备的定时控制功能 |
| Yaffs2文件系统 | Android采用大容量的NAND闪存作为存储设备，使用Yaffs2作为文件系统管理大容量MTD NAND Flash；Yaffs2占用内存小，垃圾回收简洁迅速。 |
| Android Paranoid网络 | 对Linux内核的网络代码进行了改动，增加了网络认证机制。可在IPV4，IPV6和蓝牙中设置，由ANDROID\_PARANOID\_NETWORK宏来启用此特性。 |

### 硬件抽象层

内核驱动和用户软件之间还存在所谓的硬件抽象层（Hardware Abstract Layer,**HAL**），它是对硬件设备的具体实现加以抽象。HAL没有在Android官方系统架构图中标明，下图标出了硬件抽象层在android系统中的位置：



鉴于许多硬件设备厂商不希望公开其设备驱动的源代码，如果能将android的应用框架层与linux系统内核的设备驱动隔离，使应用程序框架的开发尽量独立于具体的驱动程序，则android将减少对Linux内核的依赖。HAL由此而生，它是对Linux内核驱动程序进行的封装，将硬件抽象化，屏蔽掉了底层的实现细节。HAL规定了一套应用层对硬件层读写和配置的统一接口，本质上就是将硬件的驱动分为用户空间和内核空间两个层面；Linux内核驱动程序运行于内核空间，硬件抽象层运行于用户空间。

### 系统运行库层

官方的系统架构图中，位于Linux内核层之上的系统运行库层是应用程序框架的支撑，为Android系统中的各个组件提供服务。系统运行库层由系统类库和Android运行时构成。

* 系统类库

系统类库大部分由C/C++编写，所提供的功能通过Android应用程序框架为开发者所使用。主要的系统类库及说明如下表：

|  |  |
| --- | --- |
| 系统类库名称 | 说明 |
| Surface Manager | 执行多个应用程序时，管理子系统的显示，另外也对2D和3D图形提供支持 |
| Media Framework | 基于PacketVideoOpenCore的多媒体库，支持多种常用的音频和视频格式的录制和回放，所支持的编码格式包括MPEG4，MP3，H264，AAC，ARM |
| SQLite | 本地小型关系数据库，Android提供了一些新的SQLite数据库API，以替代传统的耗费资源的JDBC API |
| OpenGL|ES | 基于OpenGL ES 1.0API标准实现的3D跨平台图形库 |
| FreeType | 用于显示位图和矢量字体 |
| WebKit | Web浏览器的软件引擎 |
| SGL | 底层的2D图形引擎 |
| Libc（bionic l ibc） | 继承自BSD的C函数库bionic libc，更适合基于嵌入式Linux的移动设备 |
| SSL | 安全套接层，是为网络通信提供安全及数据完整性的一种安全协议 |

以上的很多类库，例如SQlite,WebKit,SSL都在会在日常开发中有用到。除上表列举的主要系统类库之外，**Android NDK**（Native Development Kit），即**Android原生库**，也十分重要。NDK为开发者提供了直接使用Android系统资源，并采用C或C++语言编写程序的接口。因此，第三方应用程序可以不依赖于Dalvik虚拟机进行开发。实际上，NDK提供了一系列从C或C++生成原生代码所需要的工具，为开发者快速开发C或C++的动态库提供方便，并能自动将生成的动态库和java应用程序一起打包成应用程序包文件，即.apk文件。   
注意，使用原生库无法访问应用框架层API，兼容性可能无法保障。而且从安全性角度考虑，Android原生库用非类型安全的程序语言C,C++编写，更容易产生安全漏洞，原生库的缺陷（bug）也可能更容易直接影响应用程序的安全性。

* 运行时

Android运行时包含核心库和Dalvik虚拟机两部分。

* 核心库：核心库提供了Java5 se API的多数功能，并提供Android的核心API，如android.os，android.net，android.media等。
* Dalvik虚拟机：Dalvik虚拟机是基于apache的java虚拟机，并被改进以适应低内存，低处理器速度的移动设备环境。Dalvik虚拟机依赖于Linux内核，实现进程隔离与线程调试管理，安全和异常管理，垃圾回收等重要功能。

本质而言，Dalvik虚拟机并非传统意义上的java虚拟机（JVM）。Dalvik虚拟机不仅不按照Java虚拟机的规范来实现，而且两者不兼容。Dalvik和标准Java虚拟机有以下主要区别：

* Dalvik基于寄存器，而JVM基于栈。一般认为，基于寄存器的实现虽然更多依赖于具体的CPU结构，硬件通用性稍差，但其使用等长指令，在效率速度上较传统JVM更有优势。
* Dalvik经过优化，允许在有限的内存中同时高效地运行多个虚拟机的实例，并且每一个Dalvik应用作为一个独立的Linux进程执行，都拥有一个独立的Dalvik虚拟机实例。Android这种基于Linux的进程“沙箱”机制，是整个安全设计的基础之一。
* Dalvik虚拟机从DEX（Dalvik Executable）格式的文件中读取指令与数据，进行解释运行。DEX文件由传统的，编译产生的CLASS文件，经dx工具软件处理后生成。
* Dalvik的DEX文件还可以进一步优化，提高运行性能。通常，OEM的应用程序可以在系统编译后，直接生成优化文件（.ODEX）； 第三方的应用程序则可在运行时在缓存中优化与保存，优化后的格式为DEY（.dey文件）。

这部分内容，即从android4.4开始就出现了ART（android runtime），但是这个ART并不是指这一节的主题，而是一种用来代替Dalvik的新型运行环境。当然在4.4的正式环境中用的还是Dalvik，真正开始用ART取代Dalvik是从android5.0开始的。另外这一节中有提到NDK,相信对于开发者而言SDK和NDK都是必要要接触和了解的东西，那么先从下图来看看sdk和ndk的关系。   
很显然地，ndk可以通过native code跨过使用dalvik runtime,直接调用到android内核资源，而sdk则需要在dalvik runtime环境下才能调用到内核资源。然而两者并不是各司其职，各不相关。android提供了**JNI**(java native interface)使两者可以进行相互调用和通信。

### 应用程序框架层

应用程序框架层提供开发Android应用程序所需的一系列类库，使开发人员可以进行快速的应用程序开发，方便重用组件，也可以通过继承实现个性化的扩展。具体包括的模块如表：

|  |  |
| --- | --- |
| 应用程序框架层类库名称 | 功能 |
| 活动管理器（Activity Mananger） | 管理各个应用程序生命周期并提供常用的导航回退功能，为所有程序的窗口提供交互的接口 |
| 窗口管理器（Window Manager） | 对所有开启的窗口程序进行管理 |
| 内容提供器（Content Provider） | 提供一个应用程序访问另一个应用程序数据的功能，或者实现应用程序之间的数据共享 |
| 视图系统（View System） | 创建应用程序的基本组件，包括列表（lists），网格（grids），文本框（text boxes），按钮（buttons），还有可嵌入的web浏览器。 |
| 通知管理器（Notification Manager） | 使应用程序可以在状态栏中显示自定义的客户提示信息 |
| 包管理器（Package Manager） | 对应用程序进行管理，提供的功能诸如安装应用程序，卸载应用程序，查询相关权限信息等。 |
| 资源管理器（Resource Manager） | 提供各种非代码资源供应用程序使用，如本地化字符串，图片，音频等 |
| 位置管理器（Location Manager） | 提供位置服务 |
| 电话管理器（Telephony Manager） | 管理所有的移动设备功能 |
| XMPP服务 | 是Google在线即时交流软件中一个通用的进程，提供后台推送服务 |

### 应用层

Android平台的应用层上包括各类与用户直接交互的应用程序，或由java语言编写的运行于后台的服务程序。例如，智能手机上实现的常见基本功能 程序，诸如SMS短信，电话拨号，图片浏览器，日历，游戏，地图，web浏览器等程序，以及开发人员开发的其他应用程序。

将android的基本架构进行了一个总体的分析和罗列，我们可以发现，平时开发中最常接触和用到的一定是application层，但是我们也不难发现，一些application层应用到的东西都能在系统层找到对应的踪迹，例如sqlite,webkit,甚至alarm。他们是怎么从底层到达application层供我们日常开发所用，这个也是需要去了解和研究的。

## 问：Android布局相关？

声明Android程序布局有两种方式：

1) 使用XML文件描述界面布局；

2) 在Java代码中通过调用方法进行控制。

我们既可以使用任何一种声明界面布局的方式，也可以同时使用两种方式。

使用XML文件声明有以下3个特点：

1) 将程序的表现层和控制层分离；

2) 在后期修改用户界面时，无须更改程序的源程序；

3) 可通过WYSIWYG可视化工具直接看到所设计的用户界面，有利于加快界面设计的过程。

建议尽量采用XML文件声明界面元素布局。在程序运行时动态添加界面布局会大大降低应用响应速度，但依然可以在必要时动态改变屏幕内容。

### LinearLayout线性布局

LinearLayout容器中的组件一个挨一个排列，通过控制android:orientation属性，可控制各组件是横向排列还是纵向排列。

#### LinearLayout的常用XML属性及相关方法

| XML属性 | 相关方法 | 说明 |
| --- | --- | --- |
| android:gravity | setGravity(int) | 设置布局管理器内组件的对齐方式 |
| android:orientation | setOrientation(int) | 设置布局管理器内组件的排列方式，可以设置为horizontal、vertical两个值之一 |

其中，gravity属性支持top, left, right, center\_vertical, fill\_vertical, center\_horizontal, fill\_horizontal, center, fill, clip\_vertical, clip\_horizontal。也可以同时指定多种对齐方式的组合。

#### LinearLayout子元素支持的常用XML属性及方法

| XML属性 | 说明 |
| --- | --- |
| android:layout\_gravity | 指定该子元素在LinearLayout中的对齐方式 |
| android:layout\_weight | 指定子元素在LinearLayout中所占的权重 |

### TableLayout表格布局

TableLayout继承自Linearout，本质上仍然是线性布局管理器。表格布局采用行、列的形式来管理UI组件，并不需要明确地声明包含多少行、多少列，而是通过添加TableRow、其他组件来控制表格的行数和列数。

每向TableLayout中添加一个TableRow就代表一行；

每向TableRow中添加一个一个子组件就表示一列；

如果直接向TableLayout添加组件，那么该组件将直接占用一行；

在表格布局中，可以为单元格设置如下三种行为方式：

Shrinkable：该列的所有单元格的宽度可以被收缩，以保证该表格能适应父容器的宽度；

Strentchable：该列所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间；

Collapsed：如果该列被设置为Collapsed，那么该列的所有单元格会被隐藏；

#### TableLayout的常用XML属性及方法

| XML属性 | 相关方法 | 说明 |
| --- | --- | --- |
| android:collapseColumns | setColumns(int, boolean) | 设置需要被隐藏的列的序号，多个序号间用逗号分隔 |
| android:shrinkColumns | setShrinkAllColumns(boolean) | 设置需要被收缩的列的序号 |
| android:stretchColumns | setStretchAllColumns(boolean) | 设置允许被拉伸的列的序号 |

### FrameLayout帧布局

FrameLayout直接继承自ViewGroup组件。帧布局为每个加入其中的组件创建一个空白的区域(称为一帧)，每个子组件占据一帧，这些帧会根据gravity属性执行自动对齐。

#### FrameLayout的常用XM了属性及方法

| XML属性 | 相关方法 | 说明 |
| --- | --- | --- |
| android:foreground | setForeground(Drawable) | 设置该帧布局容器的前景图像 |
| android:foregroundGravity | setForeGroundGraity(int) | 定义绘制前景图像的gravity属性 |

### RelativeLayout相对布局

#### RelativeLayout的XML属性及相关方法说明

| XML属性 | 相关方法 | 说明 |
| --- | --- | --- |
| android:gravity | setGravity(int) |  |
| android:ignoreGravity | setIgnoreGravity(int) | 设置哪个组件不受gravity属性的影响 |

为了控制该布局容器的各子组件的布局分布，RelativeLayout提供了一个内部类：RelativeLayout.LayoutParams。

#### RelativeLayout.LayoutParams里只能设为boolean的XML属性

| XML属性 | 说明 |
| --- | --- |
| android:layout\_centerHorizontal | 设置该子组件是否位于布局容器的水平居中 |
| android:layout\_centerVertical |  |
| android:layout\_centerParent |  |
| android:layout\_alignParentBottom |  |
| android:layout\_alignParentLeft |  |
| android:layout\_alignParentRight |  |
| android:layout\_alignParentTop |  |

#### RelativeLayout.LayoutParams里属性值为其他UI组件ID的XML属性

| XML属性 | 说明 |
| --- | --- |
| android:layout\_toRightOf | 控制该子组件位于给出ID组件的右侧 |
| android:layout\_toLeftOf |  |
| android:layout\_above |  |
| android:layout\_below |  |
| android:layout\_alignTop |  |
| android:layout\_alignBottom |  |
| android:layout\_alignRight |  |
| android:layout\_alignLeft |  |

### Android 4.0新增的网格布局GridLayout

GridLayout是Android4.0增加的网格布局控件，与之前的TableLayout有些相似，它把整个容器划分为rows × columns个网格，每个网格可以放置一个组件。性能及功能都要比tablelayout好，比如GridLayout布局中的单元格可以跨越多行，而tablelayout则不行，此外，其渲染速度也比tablelayout要快。

GridLayout提供了setRowCount(int)和setColumnCount(int)方法来控制该网格的行和列的数量。

### GridLayout常用的XML属性和方法说明

| XML属性 | 相关方法 | 说明 |
| --- | --- | --- |
| android:alignmentMode | setAlignmentMode(int) | 设置该布局管理器采用的对齐模式 |
| android:columnCount | setColumnCount(int) | 设置该网格的列数量 |
| android:columnOrderPreserved | setColumnOrderPreserved(boolean) | 设置该网格容器是否保留序列号 |
| android:roeCount | setRowCount(int) | 设置该网格的行数量 |
| android:rowOrderPreserved | setRowOrderPreserved(boolean) | 设置该网格容器是否保留行序号 |
| android:useDefaultMargins | setUseDefaultMargins(boolean) | 设置该布局管理器是否使用默认的页边距 |

为了控制GridLayout布局容器中各子组件的布局分布，GridLayout提供了一个内部类：GridLayout.LayoutParams，来控制Gridlayout布局容器中子组件的布局分布。

#### GridLayout.LayoutParams常用的XML属性和方法说明

| XML属性 | 说明 |
| --- | --- |
| android:layout\_column | 设置该组件在GridLayout的第几列 |
| android:layout\_columnSpan | 设置该子组件在GridLayout横向上跨几列 |
| android:layout\_gravity | 设置该子组件采用何种方式占据该网格的空间 |
| android:layout\_row | 设置该子组件在GridLayout的第几行 |
| android:layout\_rowSpan | 设置该子组件在GridLayout纵向上跨几行 |

### AbsoluteLayout绝对布局

即Android不提供任何布局控制，而是由开发人员自己通过X坐标、Y坐标来控制组件的位置。每个组件都可指定如下两个XML属性：

layour\_x;

layout\_y;

绝对布局已经过时，不应使用或少使用。

### 界面布局类型的选择和性能优化

首先得明确，界面布局类型的嵌套越多越深越复杂，会使布局实例化变慢，使Activity的展开时间延长。建议尽量减少布局嵌套，尽量减少创建View对象的数量。

1 . 减少布局层次，可考虑用RelativeLayout来代替LinearLayout。通过Relative的相对其他元素的位置来布局，可减少块状嵌套；

2 . 另一种减少布局层次的技巧是使用 <merge /> 标签来合并布局；

3 . 重用布局。Android支持在XML中使用 <include /> 标签， <include /> 通过指定android:layout属性来指定要包含的另一个XML布局。

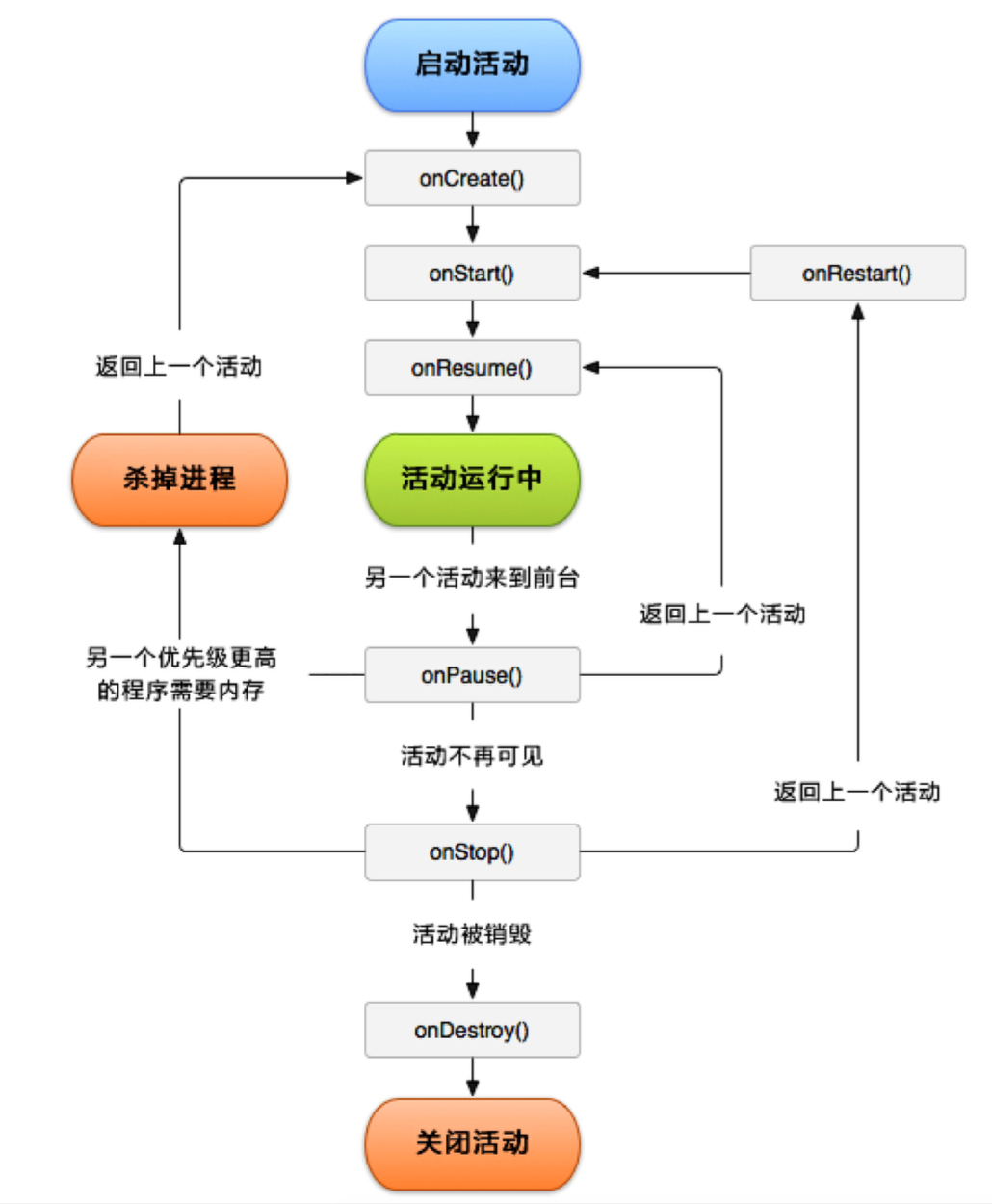
如：

1. <include android:id="@+id/id1" android:layout="@layout/mylayout">
2. <include android:id="@+id/id2" android:layout="@layout/mylayout">
3. <include android:id="@+id/id3" android:layout="@layout/mylayout">

## 问：四大组件是什么？

答：Activity、Service、Broadcast Receiver、Content Provider

## 问：Activity生命周期，及几种特殊情况下的生命周期？



在正常情况下，一个Activity从启动到结束会以如下顺序经历整个生命周期：

onCreate()->onStart()->onResume()->onPause()->onStop()->onDestory()。包含了六个部分，还有一个onRestart()没有调用，下面我们一一介绍这七部分内容。

(1) onCreate()：当 Activity 第一次创建时会被调用。这是生命周期的第一个方法。在这个方法中，可以做一些初始化工作，比如调用setContentView去加载界面布局资源，初始化Activity所需的数据。当然也可借助onCreate()方法中的Bundle对象来回复异常情况下Activity结束时的状态（后面会介绍）。

(2) onRestart()：表示Activity正在重新启动。一般情况下，当前Activity从不可见重新变为可见状态时，onRestart就会被调用。这种情形一般是用户行为导致的，比如用户按Home键切换到桌面或打开了另一个新的Activity，接着用户又回到了这个Actvity。（关于这部分生命周期的历经过程，后面会介绍。）

(3) onStart(): 表示Activity正在被启动，即将开始，这时Activity已经出现了，但是还没有出现在前台，无法与用户交互。这个时候可以理解为Activity已经显示出来，但是我们还看不到。

(4) onResume():表示Activity已经可见了，并且出现在前台并开始活动。需要和onStart()对比，onStart的时候Activity还在后台，onResume的时候Activity才显示到前台。

(5) onPause():表示 Activity正在停止，仍可见，正常情况下，紧接着onStop就会被调用。在特殊情况下，如果这个时候快速地回到当前Activity，那么onResume就会被调用（极端情况）。onPause中不能进行耗时操作，会影响到新Activity的显示。因为onPause必须执行完，新的Activity的onResume才会执行。

(6) onStop():表示Activity即将停止，不可见，位于后台。可以做稍微重量级的回收工作，同样不能太耗时。

(7) onDestory():表示Activity即将销毁，这是Activity生命周期的最后一个回调，可以做一些回收工作和最终的资源回收。

在平常的开发中，我们经常用到的就是 onCreate()和onDestory()，做一些初始化和回收操作。

### 生命周期的几种普通情况

1. 针对一个特定的Activity，第一次启动，回调如下：onCreate()->onStart()->onResume()

②用户打开新的Activity的时候，上述Activity的回调如下：onPause()->onStop()

③再次回到原Activity时，回调如下：onRestart()->onStart()->onResume()

1. 按back键回退时，回调如下：onPause()->onStop()->onDestory()
2. 按Home键切换到桌面后又回到该Activity，回调如下：onPause()->onStop()->onRestart()->onStart()->onResume()
3. 调用finish()方法后，回调如下：onDestory()(以在onCreate()方法中调用为例，不同方法中回调不同，通常都是在onCreate()方法中调用)

### 特殊情况下的生命周期

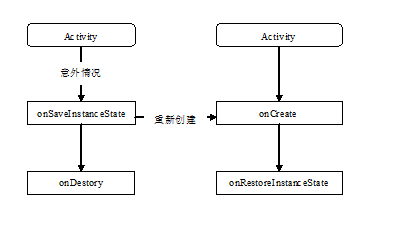
上面是普通情况下Activity生命周期的一些流程，但是在一些特殊情况下，Activity的生命周期的经历有些异常，下面就是两种特殊情况。

①横竖屏切换

在横竖屏切换的过程中，会发生Activity被销毁并重建的过程。

两个回调：onSaveInstanceState和onRestoreInstanceState。

在Activity由于异常情况下终止时，系统会调用onSaveInstanceState来保存当前Activity的状态。这个方法的调用是在onStop之前，它和onPause没有既定的时序关系，该方法只在Activity被异常终止的情况下调用。当异常终止的Activity被重建以后，系统会调用onRestoreInstanceState，并且把Activity销毁时onSaveInstanceState方法所保存的Bundle对象参数同时传递给onRestoreInstanceState和onCreate方法。因此，可以通过onRestoreInstanceState方法来恢复Activity的状态，该方法的调用时机是在onStart之后。其中onCreate和onRestoreInstanceState方法来恢复Activity的状态的区别： onRestoreInstanceState回调则表明其中Bundle对象非空，不用加非空判断。onCreate需要非空判断。建议使用onRestoreInstanceState。



横竖屏切换的生命周期：onPause()->onSaveInstanceState ()-> onStop()->onDestroy()->onCreate()->onStart()->onRestoreInstanceState()->onResume()

可以通过在AndroidManifest文件的Activity中指定如下属性：

android: configChanges = "orientation| screenSize"

来避免横竖屏切换时，Activity的销毁和重建，而是回调了下面的方法：

@Override

public void onConfigurationChanged (Configuration newConfig) {

super. onConfigurationChanged(newConfig);

}

### 资源内存不足导致优先级低的Activity被杀死

内存不足时，Dalvak 虚拟机会根据其内存回收规则来回收内存：

1. 先回收与其他Activity 或Service/Intent Receiver 无关的进程(即优先回收独立的Activity)因此建议,我们的一些(耗时)后台操作，最好是Service的形式

2.不可见(处于Stopped状态的)Activity

3.Service进程(除非真的没有内存可用时会被销毁)

4.非活动的可见的(Paused状态的)Activity

5.当前正在运行（Active/Running状态的）Activity

### Activity的三种运行状态

①Resumed（活动状态）

又叫Running状态，这个Activity正在屏幕上显示，并且有用户焦点。这个很好理解，就是用户正在操作的那个界面。

②Paused（暂停状态）

这是一个比较不常见的状态。这个Activity在屏幕上是可见的，但是并不是在屏幕最前端的那个Activity。比如有另一个非全屏或者透明的Activity是Resumed状态，没有完全遮盖这个Activity。

③Stopped（停止状态）

当Activity完全不可见时，此时Activity还在后台运行，仍然在内存中保留Activity的状态，并不是完全销毁。这个也很好理解，当跳转的另外一个界面，之前的界面还在后台，按回退按钮还会恢复原来的状态，大部分软件在打开的时候，直接按Home键，并不会关闭它，此时的Activity就是Stopped状态。

## 问：Activity的启动模式？

答：

### 启动模式的类别

Android提供了四种Activity启动方式：

标准模式（standard）

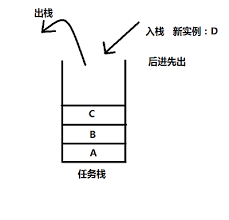
栈顶复用模式（singleTop）

栈内复用模式（singleTask）

单例模式（singleInstance）

### 启动模式的结构——栈

Activity的管理是采用任务栈的形式，任务栈采用“后进先出”的栈结构。



### Activity的LaunchMode

#### 标准模式（standard）

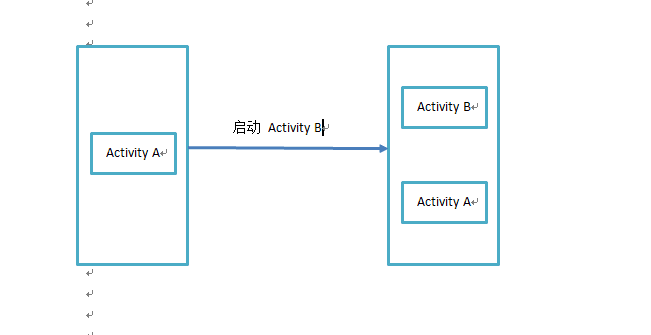
每启动一次Activity，就会创建一个新的Activity实例并置于栈顶。谁启动了这个Activity，那么这个Activity就运行在启动它的那个Activity所在的栈中。

例如：Activity A启动了Activity B，则就会在A所在的栈顶压入一个新的Activity。

特殊情况，如果在Service或Application中启动一个Activity，其并没有所谓的任务栈，可以使用标记位Flag来解决。解决办法：为待启动的Activity指定FLAG\_ACTIVITY\_NEW\_TASK标记位，创建一个新栈。

应用场景： 绝大多数Activity。如果以这种方式启动的Activity被跨进程调用，在5.0之前新启动的

Activity实例会放入发送Intent的Task的栈的顶部，尽管它们属于不同的程序，这似乎有点费解看起来也不是那么合理，所以在5.0之后，上述情景会创建一个新的Task，新启动的Activity就会放入刚创建的Task中，这样就合理的多了。



#### 栈顶复用模式（singleTop）

如果需要新建的Activity位于任务栈栈顶，那么此Activity的实例就不会重建，而是重用栈顶的实例。

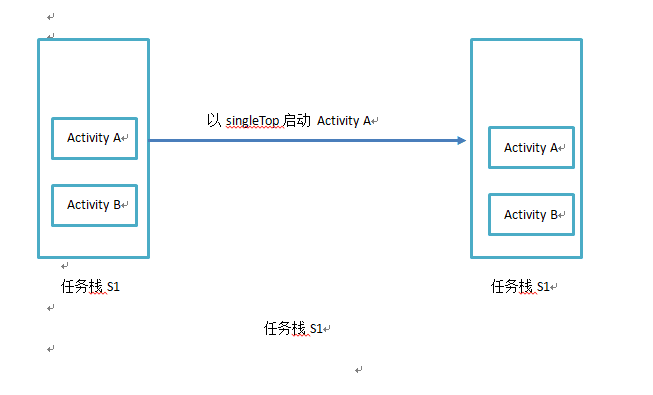
并回调如下方法：

@Override

protected void onNewIntent(Intent intent) {

super.onNewIntent(intent);

}



由于不会重建一个Activity实例，则不会回调其他生命周期方法。如果栈顶不是新建的Activity,就会创建该Activity新的实例，并放入栈顶。

应用场景： 在通知栏点击收到的通知，然后需要启动一个Activity，这个Activity就可以用singleTop，否则每次点击都会新建一个Activity。当然实际的开发过程中，测试妹纸没准给你提过这样的bug：某个场景下连续快速点击，启动了两个Activity。如果这个时候待启动的Activity使用 singleTop模式也是可以避免这个Bug的。同standard模式，如果是外部程序启动singleTop的Activity，在Android 5.0之前新创建的Activity会位于调用者的Task中，5.0及以后会放入新的Task中。

#### 栈内复用模式（singleTask）

该模式是一种单例模式，即一个栈内只有一个该Activity实例。该模式，可以通过AndroidManifest文件的Activity中指定该Activity需要加载到那个栈中，即singleTask的Activity可以指定想要加载的目标栈。singleTask和taskAffinity配合使用，指定开启的Activity加入到哪个栈中。

<activity android:name=".Activity1"

android:launchMode="singleTask"

android:taskAffinity="com.lvr.task"

android:label="@string/app\_name">

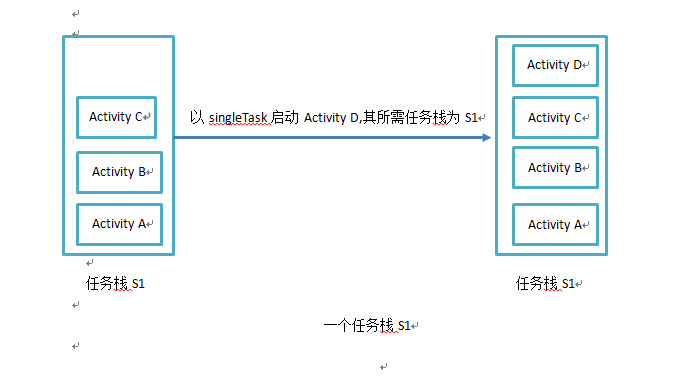
</activity>

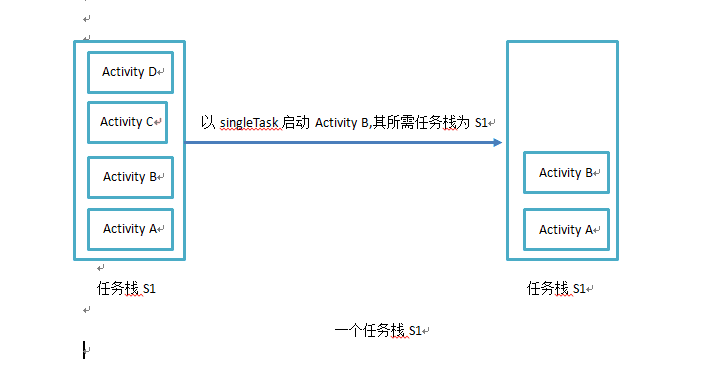
关于taskAffinity的值： 每个Activity都有taskAffinity属性，这个属性指出了它希望进入的Task。如果一个Activity没有显式的指明该Activity的taskAffinity，那么它的这个属性就等于Application指明的taskAffinity，如果Application也没有指明，那么该taskAffinity的值就等于包名。

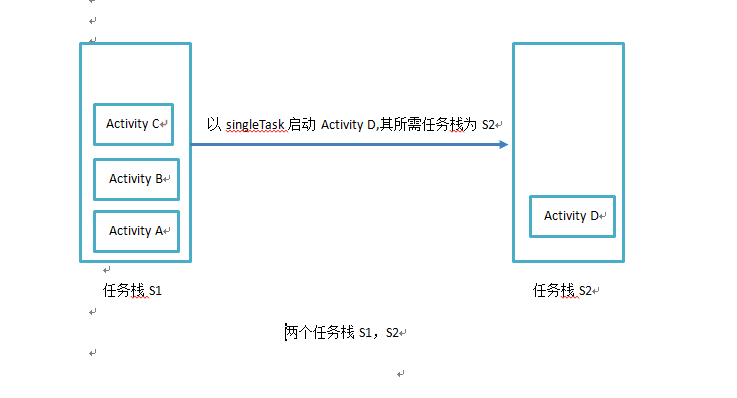
执行逻辑：

在这种模式下，如果Activity指定的栈不存在，则创建一个栈，并把创建的Activity压入栈内。如果Activity指定的栈存在，如果其中没有该Activity实例，则会创建Activity并压入栈顶，如果其中有该Activity实例，则把该Activity实例之上的Activity杀死清除出栈，重用并让该Activity实例处在栈顶，然后调用onNewIntent()方法。

对应如下三种情况：



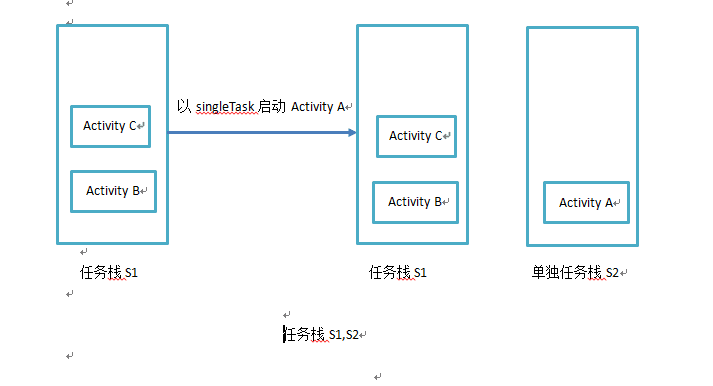




应用场景： 大多数App的主页。对于大部分应用，当我们在主界面点击回退按钮的时候都是退出应用，那么当我们第一次进入主界面之后，主界面位于栈底，以后不管我们打开了多少个Activity，只要我们再次回到主界面，都应该使用将主界面Activity上所有的Activity移除的方式来让主界面Activity处于栈顶，而不是往栈顶新加一个主界面Activity的实例，通过这种方式能够保证退出应用时所有的Activity都能报销毁。在跨应用Intent传递时，如果系统中不存在singleTask Activity的实例，那么将创建一个新的Task，然后创建SingleTask Activity的实例，将其放入新的Task中。

#### 单例模式（singleInstance）

作为栈内复用模式（singleTask）的加强版,打开该Activity时，直接创建一个新的任务栈，并创建该Activity实例放入新栈中。一旦该模式的Activity实例已经存在于某个栈中，任何应用再激活该Activity时都会重用该栈中的实例。

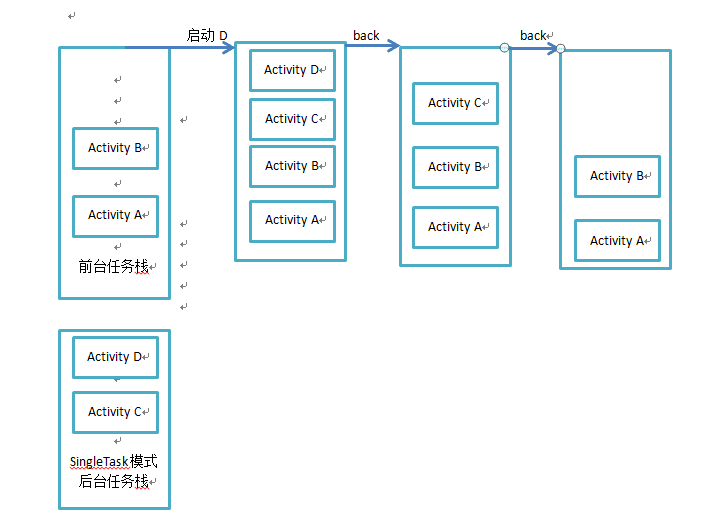


应用场景： 呼叫来电界面。这种模式的使用情况比较罕见，在Launcher中可能使用。或者你确定你需要使Activity只有一个实例。建议谨慎使用。

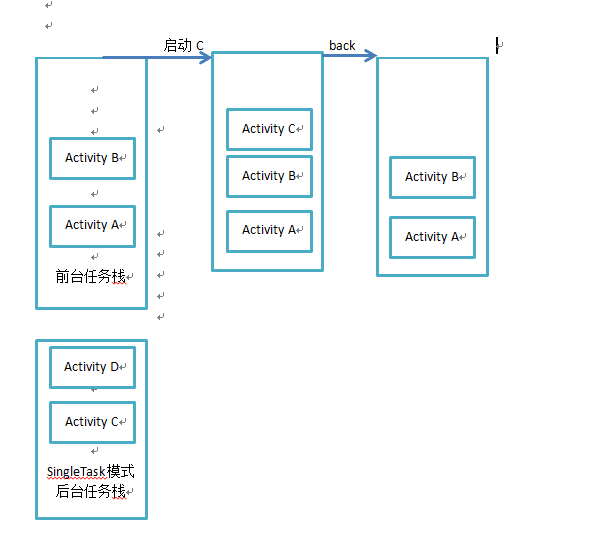
### 特殊情况——前台栈和后台栈的交互

假如目前有两个任务栈。前台任务栈为AB，后台任务栈为CD，这里假设CD的启动模式均为singleTask,现在请求启动D，那么这个后台的任务栈都会被切换到前台，这个时候整个后退列表就变

成了ABCD。当用户按back返回时，列表中的activity会一一出栈，如下图。



如果不是请求启动D而是启动C，那么情况又不一样，如下图。



调用SingleTask模式的后台任务栈中的Activity，会把整个栈的Actvity压入当前栈的栈顶。singleTask会具有clearTop特性，把之上的栈内Activity清除。

### Activity的Flags

Activity的Flags很多，这里介绍集中常用的，用于设定Activity的启动模式。可以在启动Activity时，通过Intent的addFlags()方法设置。

(1)FLAG\_ACTIVITY\_NEW\_TASK 其效果与指定Activity为singleTask模式一致。

(2)FLAG\_ACTIVITY\_SINGLE\_TOP 其效果与指定Activity为singleTop模式一致。

(3)FLAG\_ACTIVITY\_CLEAR\_TOP 具有此标记位的Activity，当它启动时，在同一个任务栈中所有位于它上面的Activity都要出栈。如果和singleTask模式一起出现，若被启动的Activity已经存在栈中，则清除其之上的Activity，并调用该Activity的onNewIntent方法。如果被启动的Activity采用standard模式，那么该Activity连同之上的所有Activity出栈，然后创建新的Activity实例并压入栈中。

## 问：Fragment的生命周期？

## 问：Activity与Fragment之间生命周期比较？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：两个Activity 之间跳转时必然会执行的是哪几个方法？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法。

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity的四种启动模式对比？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity状态保存于恢复？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Android的6大布局？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：ndk ，jni？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：android 数据存储方式？

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

## 问：Activity上有Dialog的时候按Home键时的生命周期

(1)通过后台新增账号可登录分钟系统(同一个账号可同时APP,网页端在线),登录成功未绑定设备,都进入设备绑定页。

(2)客户端已绑定设备登录，进入分钟诊所主页

# Java

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容



# C 和 C++

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

# 算法

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

# 前端

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

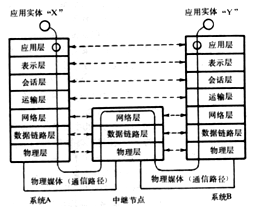
2.重置按钮,清空账户和密码内所有内容

# 网络，互联网

## 问：OSI/RM 是什么？

答：OSI/RM一般指开放系统互连参考模型

开放系统互连参考模型 (Open System Interconnect 简称OSI）是国际标准化组织(ISO)和国际电报电话咨询委员会(CCITT)联合制定的开放系统互连参考模型，为开放式互连信息系统提供了一种功能结构的框架。它从低到高分别是：物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。



OSI参考模型是计算机网路体系结构发展的产物。它的基本内容是开放系统通信功能的分层结构。这个模型把开放系统的通信功能划分为七个层次，从邻接物理媒体的层次开始，分别赋于1，2，……7层的顺序编号，相应地称之为物理层、数据链路层、网络层、运输层、会话层、表示层和应用层。每一层的功能是独立的。它利用其下一层提供的服务并为其上一层提供服务，而与其他层的具体实现无关。这里所谓的“服务”就是下一层向上一层提供的通信功能和层之间的会话规定，一般用通信原语实现。两个开放系统中的同等层之间的通信规则和约定称之为协议。通常把1～4层协议称为下层协议，5～7层协议称为上层协议。

1、国际标准化组织ISO在1979年建立了一个分委员会来专门研究一种用于开放系统的体系结构，提出了开放系统互连OSI模型，这是一个定义连接异种计算机的标准主体结构。

　　2、OSI简介：OSI采用了分层的结构化技术，共分七层，物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

　　3、OSI参考模型的特性：是一种异构系统互连的分层结构；提供了控制互连系统交互规则的标准骨架；定义一种抽象结构，而并非具体实现的描述；不同系统中相同层的实体为同等层实体；同等层实体之间通信由该层的协议管理；相信层间的接口定义了原语操作和低层向上层提供的服务；所提供的公共服务是面向连接的或无连接的数据服务；直接的数据传送仅在最低层实现；每层完成所定义的功能，修改本层的功能并不影响其他层。

　　4、物理层：提供为建立、维护和拆除物理链路所需要的机械的、电气的、功能的和规程的特性；有关的物理链路上传输非结构的位流以及故障检测指示。

　　5、数据链路层：在网络层实体间提供数据发送和接收的功能和过程；提供数据链路的流控。

　　6、网络层：控制分组传送系统的操作、路由选择、拥护控制、网络互连等功能，它的作用是将具体的物理传送对高层透明。

　　7、传输层：提供建立、维护和拆除传送连接的功能；选择网络层提供最合适的服务；在系统之间提供可靠的透明的数据传送，提供端到端的错误恢复和流量控制。

　　8、会话层：提供两进程之间建立、维护和结束会话连接的功能；提供交互会话的管理功能，如三种数据流方向的控制，即一路交互、两路交替和两路同时会话模式 。

　　9、表示层：代表应用进程协商数据表示；完成数据转换、格式化和文本压缩。

　 10、应用层：提供OSI用户服务，例如事务处理程序、文件传送协议和网络管理等。

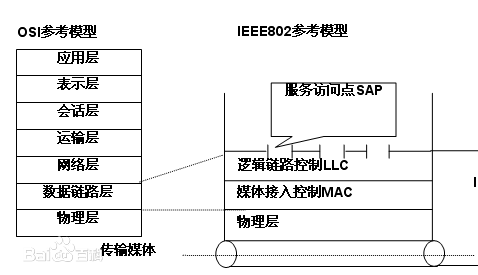
## 问：sap 是什么？

答：SAP，是Service Accessing point的缩写，意思是服务访问点，即上层访问下层所提供服务的点。

SAP（Service Access Point）是上层访问下层所提供服务的点。在计算机体系结构中，下层是为相邻上层提供服务的，而下层对它的所有上层都是透明的。

SAP是临层实体（“实体”也就是对应层的逻辑功能）间实现相互通讯的逻辑接口，位于两层边界处。从物理层开始，每一层都向上层提供服务访问点（应用层除外），每一层都有SAP，但不同层的SAP内容和表示形式是不一样的。

下面举例说明数据链路层的MAC子层和LLC子层的关系：

SAP是一个层次系统的上下层之间进行通信的接口，LLC子层为了网络层的各种协议提供服务，而上层可能运行不同协议，为区分不同上层协议的数据，要采用服务访问点。如下图：

data link layer 和 network layer 之间就是通过MAC将数据传到LLC然后由LLC通过SAP和network layer传输数据.

## 问：http,https 是什么？

## 问： 是什么？

# 实践项目

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

# 书籍

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

# 行业流行语

## 登录

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

# 常用工具和命令

## Git

1.用户使用账号和密码登录三诺分钟诊所管理系统，账号体系由三诺云平台管理，如图：

2.重置按钮,清空账户和密码内所有内容

### Git常用命令

git config --global

git status

git init 初始化

git clone 克隆

git push

git pull

git add

git commit

### git 原理

一：Git是什么？

　　Git是目前世界上最先进的分布式版本控制系统。

二：SVN与Git的最主要的区别？

　　SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就纳闷了。

Git是分布式版本控制系统，那么它就没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多个人如何协作呢？比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

## windows常用命令

Win+R

在文件夹空白处按住Shift，然后右键弹出快捷菜单，可以看到“在此处打开命令行窗口”

使用上下方向键，翻看使用过的命令

tab补齐功能

del

rem 重命名

dir 显示文件列表

tree 树形显示

Ctrl+C 强行终止当前命令或程序的执行。

cls 清除屏幕

md 创建

type 查看内容

两种通配符：问号（?）匹配文件名中的任何一个字符，星号（\*）匹配文件名中的零个或多个字符。

快捷键：

Win + 数字：任务栏中对应位置程序

Win + E: 打开资源管理器

Ctrl + W：关闭资源管理器

Win + D：跳转到桌面

Win + M：最小化所有窗口,跳到桌面

Win + Shift +M:还原最小化（所有）窗口

Win + R：打开运行

Ctrl + Alt + Delete:

在运行处输入：

1）cmd:打开DOS窗口

2）notepad：打开记事本

3）calc：打开计算器

4）mspaint：打开画图板

5）services.msc：打开系统服务

## Adb

## Gradle

## Linux

## Tomcat

## Tomcat