Android的窗口管理是C/S模式的。Android中的Window是表示TopLevel等顶级窗口的概念。DecorView是Window的Top-LevelView，这个View我称之为主View，DecorView会缺省的attach到Activity的主窗口中。主View被加入到WindowManager中，WM使用WindowState与这个主View对应。  
  
Activity建立一个主窗口后，在将主窗口添加到WindowManager时，首先要建立WindowManager代理对象，并打开一个会话（实现IWindowSessionAIDL接口），并维持该会话。Activity将通过该会话与WindowManager建立联系，这个Session是C/S体系的基础，Client通过WindowSession将window加入到WindowManager中。  
一个完整的窗口概念横跨了View，ViewRoot，WindowManagerService。Window，DecorView，View，IWindow,ISession，WindowState之间的关系如下  
  
Client端的Activity通过Session与WindowManager建立对话，而WindowManager则通过IWindow接口访问Client，将消息传递到Client端，通过消息分发渠道，将消息传递到处理函数OnXXX。  
后面我们将通过Client，WMService分别加以分析。  
2Client端  
我一致认为在Android中Window的概念并不是个很重要的概念。他的Window类，只是在PhoneWindow和MidWindow中使用。而PhoneWindow只是做了一个具体跟手机功能相关的公用事件的处理，所以在Android中PhoneWindow并不是一个抽象的纯正概念，而是一个跟手机系统相关的一个特别窗口概念，例如按键的默认动作处理，按键音的发出等等。  
2.1View  
在Activity中真正重要的概念是View，以下是Google官方对View的定义：  
Thisclassrepresentsthebasicbuildingblockforuserinterfacecomponents.AViewoccupiesarectangularareaonthescreenandisresponsiblefordrawingandeventhandling.Viewisthebaseclassforwidgets,whichareusedtocreateinteractiveUIcomponents(buttons,textfields,etc.).The{@linkandroid.view.ViewGroup}subclassisthebaseclassforlayouts,whichareinvisiblecontainersthatholdotherViews(orotherViewGroups)anddefinetheirlayoutproperties.  
我对View不做翻译，翻译成视图好像不太佳，View在Android中，View比视图具有广的外延。View包含了用户交互，包含了显示，视图在中文中仅仅表示了静态的显示。对于View的理解应该从最容易的理解开始。我们使用过编辑器，在Android中这个编辑器就是一个View，这个编辑器需要显示文字，需要接收用户的键盘输入和鼠标选择，但是一个屏幕上有多个编辑器，如何管理，如何切换焦点编辑器，这些都是需要管理的。  
客户端的组成：（Window，View，ViewRoot，WindowManagerProxy）  
  
在Activity在performLaunchActivity时，会使用Activity.attach（）建立一个PhoneWindow主窗口。这个主窗口的建立并不是一个重点。handleResumeActivity真正要启动一个Activity时候，将主窗口加入到WindowManager，当然并不是将主窗口本身，而是**将主窗口的DecorView加入到WindowManager中**。  
真正Window核心的抽象概念存在于View，ViewRoot，WindowManger中的WindowState。为了描述概念的方便性，我特别提出主View这个概念，这个主View就是Top-LevelViewofthewindow.主View与View想对，突出主View是attatch到主窗口上的。而一般的View则是存在于主View中的。主窗口这个概念，我讲的主窗口实际上就是Android提到的TopLevelWindow。  
我们所提到的概念：View，GroupView，DecorView，ViewRoot都是存在于Client端，只有WindowState这个概念存在于WindowManagerService端。  
DecorView实际上是一个ViewGroup。在依存关系上来讲，对看个主窗口来讲，DecorView是Top-LevelView.View并不是关注的重点，重要的是我们如何需要知道分发路径是建立在什么关系上的。View的成员变量mParent用来管理View上级关系的。而ViewGroup顾名思义就是一组View的管理，于是在ViewGroup构建了**焦点管理和子View节点数组**。这样通过**View的mParent和ViewGroup的mChildren构建了Android中View直接的关系网**。  
  
2.2FocusPath  
所谓的**FoucsPath就是我们的KeyEvent传递的路线**。一般的我们的KeyEvent在主循环中主View通过View的焦点记录关系传递到焦点View上。例如下图，View22是焦点，我们从最顶层的View通过mFcous的关系链找到最后所形成的路径就是FocusPath。  
  
2.3ViewRoot，WindowManagerProxy  
ViewRoot与WindowManager的核心是IWindowSession和IWindow。ViewRoot通过IWindowSession添加窗口到WindowManager。而IWindow这是WindowManager分发消息给ClientViewRoot的渠道。利用AIDL接口进行进程间通信。  
  
ViewRoot实际是一个Handler，ViewRoot建立主View与WindowsManger通讯的桥梁。ViewRoot在本质上一个Handler。我们知道Handler的基本功能就是处理回调，发送消息。  
Activity在使用getSystemService获取WindowManagerImpl，建立了一个WindowManagerImpl实例，即WindowManager服务的代理：  
wm=(WindowManagerImpl)context.getSystemService(Context.WINDOW\_SERVICE);并调用wm.addview添加窗口到WMService中。  
这个过程在客户端建立了什么样的管理框架，并如何这个会话？在WindowManagerProxy中建立了View，Layout,ViewRoot三者的对应关系表。构造一个ViewRoot就会打开一个session,并利用IWindowSession建立会话上下文。  
  
4WindowManagerService  
本次对于WindowManagerService的研究仅限于FocusWindow,消息系统。其他的部分将在后面的专门章节讨论。  
WindowManager管理的窗口是应用程序的Top-level窗口，我这里参照Window的概念叫主窗口。主窗口为什么要放在在Service这边来管理呢？为什么不放在Client那边？主窗口放置在一起管理是为了计算Z-order序列，根据应用程序的状态来显隐应用程序的窗口。我想Android设计者在考虑设计窗口系统的时候，一定首先考虑：  
窗口z-order序的管理活动窗口的计算，及其变化通知窗口归属（属于哪个应用）输入法管理  
WindowService大体上实现了如下的功能：，  
（1）Z-ordered的维护函数  
（2）输入法管理  
（3）AddWindow/RemoveWindow  
（4）Layerout  
（5）Token管理，AppToken  
（6）活动窗口管理（FocusWindow）  
（7）活动应用管理（FocusAPP）  
（8）转场动画  
（9）系统消息收集线程  
（11）系统消息分发线程  
在服务端的窗口对象叫做WindowState。在Service维护了一个mWindow数组，这个mWindow就是Window的Z-order序数组。mWindowMap用于记录。  
WindowState有一个叫做mClient成员变量来记录客户端IWindow实例，通过IWindow接口实例，Service可以访问客户端的信息，说以IWindow是Service连接View桥梁。  
（1）FocusWindow活动窗口如何计算?  
基本原理就是查找前景应用（FousActivity），并同Z-Order序中找出属于该FousActivity（AppToken）的主窗口，这个窗口就是计算出来的FocusWindow。  
（2）为什么要提出Token这个概念呢？  
一个应用程序要管理自己的窗口，那么如何来标识该窗口是属于某个Activity，Andoid设计者提出了AppToken这个概念。AppToken在本质上的描述：，通过Token找到属于该Token的allWindows。使用Token开始完成该应用程序的所有窗口的显示和隐藏。  
（3）系统消息收集与处理  
我们下面重点研究Ｓｅｒｖｉｃｅ中的系统消息收集模式及其分发模式。Ｓｅｒｖｉｃｅ使用ＫｅｙＱ作为专门的消息队列。  
ＫeyEvent  
TouchEvent  
TrackballEvent  
系统有两个线程：  
KeyQ线程，通过Navite函数readEvent轮询设备,将读取的结果放置在KeyQ队列中。系统dispatcher等待在KeyQ消息队列上，一旦从消息队列中获取到消息，就通过分发函数通过mClient传递到Client端。  
[Android核心分析（14）------AndroidGWES之输入系统](http://bbs.bianjibu.net/)  
**Android输入系统**  
依照惯例，在研究Android输入系统之前给出输入系统的本质描述：从哲学的观点来看，输入系统就是解决从哪里来又将到哪里去问题。输入的本质上的工作就是收集用户输入信息并放置到目标位置。  
Android在源代码分类上，并没有输入系统分类。本章的输入系统研究是一个综合的分析，前面的GWES的分析，特别是View的FocusPath以及WindowManagerProxy是本章分析的基础,如果没有理解，请参阅前面的窗口管理的相关章节。  
**Android输入系统的组成**  
  
**输入系统由如下几部分组成：**  
1）后台窗口管理服务  
2）FocusActivity  
3）FocusWindow  
4）FocusView：用来接收键盘消息  
从输入系统这个角度去看Android的WindowManager服务解决了用户信息输入收集，而FocusActvitiy，FocusWindow、FocusView这些概念的设计是为了解决用户输入应该放到哪里去这个问题。在整个Android系统中，同时只有一个一个FocusWindow，而属于该Window的FocusView才是真正的FocusView。  
在Android系统中，在设计上要求多个Actvitiy同时存在运行。在实现中，每次把Actvitiy变成FocusedActvitiy时（setFocusedActivity@ActivityManagerService.java）激活程序的时候，就把该Activity的主窗口设置成前景窗口,即系统中的顶层窗口，AppToken概念的引进就是为了解决窗口对象的归属问题。在这个过程中，在逻辑上看，我们挑选了一个Activity作为了FocusActivity来接收系统的消息，实质上这个FocusActivity的Focus窗口就是前景窗口。  
Focus窗口的改变将改变焦点View，前景窗口的改变也将引起焦点View的变化。焦点和光标的概念用于管理输入设备和输入事件的传送。光标是一个绘制在屏幕之上的小位图，指示当前的输入位置。键盘输入有类似的输入焦点和键盘输入插入符的概念。只有具有输入焦点的窗口才能获取键盘事件。改变窗口的焦点通常由特殊的按键组合或者TouchEvent事件完成。具有输入焦点的窗口通常绘制有一个键盘插入符。该插入符的存在、形式、位置，以及该插入符的控制完全是由窗口的事件处理例程完成的。  
现在站在更宏观的位置来看Actvitiy的输入系统，可以从LinuxDriver开始到输入框结束的整个链条，我这里给出大输入系统的概念，Android大输入系统包含：Linuxdriver,WindowManager,MessageSystem,ViewFocusPath，FocusView。  
**Android输入系统架构图**  
  
现在从Android的代码分析的角度，来看看输入系统的组成。这个过程从代码中分析处理：  
在WindowManagerService端  
readEvent@com\_android\_server\_KeyInputQueue.cpp  
KeyQ@WindowMangerService.java  
KeyInputQ@KeyInputeQueue.java  
InputDispatcherThread@WindowMangerService.java  
在Client端  
IWindow@ViewRoot.Java  
ViewRoot@ViewRoot.java  
KeyInputQ在WindowMangerService中建立一个独立的线程InputDeviceReader，使用Native函数readEvent来读取LinuxDriver的数据构建RawEvent，并放入到KeyQ消息队列中。  
InputDispatcherThread从KeyQ中读取Events，找到WindowManager中的FocusWindow，通过FocusWindow记录的mClient接口，将Events专递到Client端。Client端在根据自己的FocusPath传递事件，直到事件被处理。