Ui显示

<http://aijiawang-126-com.iteye.com/blog/2393863>

# TODO

IStatusBarService.Stub

StatusBarShellCommand

showTvPictureInPictureMenu

Android之framework修改底部导航栏NavigationBar动态显示和隐藏

<http://www.bubuko.com/infodetail-806975.html>

提高SystemServer到SystemUI响应速度业务实战，模拟锁屏的效果

void toggleRecentApps();——void toggleFn();

# StatusBarMS启动过程

/services/core/java/com/android/server/statusbar/StatusBarManagerService.java

StatusBarManagerService extends IStatusBarService.Stub

StatusBarManagerService是IStatusBarService的实现者。由于状态栏导航栏与它的关系十分密切，因此需要对其有所了解。

与WindowManagerService、InputManagerService等系统服务一样，StatusBarManagerService在ServerThread中创建。参考如下代码：

## SystemServer

[SystemServer.java-->ServerThread.run()]

public void run()

try {

/\* 创建一个StatusBarManagerService的实例，并注册到ServiceManager中使其成为 一个系统服务 \*/

statusBar = new StatusBarManagerService(context, wm);

ServiceManager.addService(Context.STATUS\_BAR\_SERVICE, statusBar);

} catch(Throwable e) {......}

}

StatusBarManagerService extends IStatusBarService.Stub

## StatusBarManagerService()

public StatusBarManagerService(Context context, WindowManagerService windowManager) {

mContext = context;

mWindowManager = windowManager;

LocalServices.addService(StatusBarManagerInternal.class, mInternalService);

}

这基本上是系统服务中最简单的构造函数了，在这里并没有发现能够揭示StatusBarManagerService的工作原理的线索（由此也可以预见StatusBarManagerService的实现十分简单）。

接下来参考StatusBarManagerService.registerStatusBar()的实现。这个方法由SystemUI中的BaseStatusBar调用，用于建立其与StatusBarManagerService的通信连接，并取回保存在其中的信息副本。

## registerStatusBar()

public void registerStatusBar(IStatusBar bar,StatusBarIconList iconList,

List<IBinder> notificationKeys,List<StatusBarNotification> notifications,

intswitches[], List<IBinder> binders) {

**enforceStatusBarService();**

/\* ① 将bar参数保存到mBar成员中。bar的类型是IStatusBar，它即是BaseStatusBar中的 CommandQueue的Bp端。从此之后，StatusBarManagerService将通过mBar与BaseStatusBar 进行通信。因此可以理解mBar就是SystemUI中的状态栏与导航栏 \*/

mBar =bar;

// ② 接下来依次为调用者返回信息副本

// 系统状态区的图标列表

synchronized (mIcons) { iconList.copyFrom(mIcons); }

// 通知区的通知信息

synchronized (mNotifications) {

for(Map.Entry<IBinder,StatusBarNotification> e: mNotifications.entrySet()) {

notificationKeys.add(e.getKey());

notifications.add(e.getValue());

}

}

//switches中的杂项

synchronized (mLock) {

switches[0] = gatherDisableActionsLocked(mCurrentUserId);

......

}

......

}

### enforceStatusBarService

/\* 首先是权限检查。状态栏与导航栏是Android系统中一个十分重要的组件，因此必须避免其他应用调用此方法对状态栏与导航栏进行偷梁换柱。因此要求方法的调用者必须具有一个签名级的权限 android.permission.STATUS\_BAR\_SERVICE\*/

private void enforceStatusBarService() {

mContext.enforceCallingOrSelfPermission(android.Manifest.permission.STATUS\_BAR\_SERVICE,

"StatusBarManagerService");

}

## setIcon

registerStatusBar()的实现也十分简单。.主要是保存BaseStatusBar中的CommandQueue的Bp端到mBar成员之中，然后再把信息副本填充到参数里去。尽管简单，但是从其实现中可以预料到StatusBarManagerService的工作方式：当它接受到操作状态栏与导航栏的请求时，首先将请求信息保存到副本之中，然后再将这一请求通过mBar发送给BaseStatusBar。以设置系统状态区图标这一操作为例，参考如下代码：

public void setIcon(String slot, StringiconPackage, int iconId, int iconLevel,

String contentDescription) {

/\* 首先一样是权限检查，与registerStatusBar()不同，这次要求的是一个系统级别的权限android.permission.STATUS\_BAR。因为设置系统状态区图标的操作不允许普通应用程序进行。其他的操作诸如添加一条通知则不需要此权限 \*/

enforceStatusBar();

synchronized (mIcons) {

intindex = mIcons.getSlotIndex(slot);

......

StatusBarIcon icon = new StatusBarIcon(iconPackage, UserHandle.OWNER,iconId,

iconLevel, 0,

contentDescription);

// ① 将图标信息保存在副本之中

mIcons.setIcon(index, icon);

// ② 将设置请求发送给BaseStatusBar

if(mBar != null) {

try {

mBar.setIcon(index, icon);

} catch (RemoteException ex) {......}

}

}

}

## 几者的关系如

纵观StatusBarManagerService中的其他方法，会发现它们与setIcon()方法的实现十分类似。从而可以得知StatusBarManagerService的作用与工作原理如下：

它是SystemUI中的状态栏与导航栏在system\_server中的代理。所有对状态栏或导航来有需求的对象都可以通过获取StatusBarManagerService的实例或Bp端达到其目的。只不过使用者必须拥有能够完成操作的相应权限。

它保存了状态栏／导航栏所需的信息副本，用于在SystemUI意外退出之后的恢复。

它的注册方法做一些数据的初始化

这几者的关系如下



# StatusBar通信机制

public abstract class BaseStatusBar extends SystemUI implements

CommandQueue.Callbacks

## BaseStatusBar.start方法

BaseStatusBar进行一些设置,获取了IStatusBarService实例并注册一些信息到IStatusBarService中,IStatusBarService是一个系统服务,BaseStatusBar将自己注册到IStatusBarService之中,IStatusBarService会把操作状态栏和导航栏的请求转发给BaseStatusBar

为了保证SystemUI意外退出后不会发生信息丢失，IStatusBarService保存了所有需要状态栏与导航栏进行显示或处理的信息副本。在注册时将一个继承自IStatusBar.Stub的CommandQueue的实例注册到IStatusBarService以建立通信，并将信息副本取回。

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **abstract** **class** BaseStatusBar **extends** SystemUI **implements**
2. CommandQueue.Callbacks
3. {
4. **public** **void** start() {
5. /\* // 向 SM 获取 SBMS：实例化IStatusBarService,    随后BaseStatusBar将自己注册到IStatusBarService之中。以此声明本实例才是状态栏的真正实现者，IStatusBarService会将其所接受到的请求转发给本实例。 IStatusBarService会保存SystemUi的状态信息,避免SystemUi崩溃而造成信息的丢失
6. \*/
7. mBarService = IStatusBarService.Stub.asInterface(
8. ServiceManager.getService(Context.STATUS\_BAR\_SERVICE));
9. //IStatusBarService与BaseStatusBar进行通信的桥梁。
10. mCommandQueue = **new** CommandQueue(**this**);
12. /\*switches则存储了一些杂项：禁用功能列表，SystemUIVisiblity，是否在导航栏中显示虚拟的菜单键，输入法窗口是否可见、输入法窗口是否消费BACK键、是否接入了实体键盘、实体键盘是否被启用。
13. \*/
14. **int**[] switches = **new** **int**[9];
15. ArrayList<IBinder> binders = **new** ArrayList<IBinder>();
16. /\*它保存了用于显示在状态栏的系统状态区中的状态图标列表。在完成注册之后，IStatusBarService将会在其中填充两个数组，一个字符串 数组用于表示状态的名称，一个StatusBarIcon类型的数组用于存储需要显示的图标资源。 \*/
17. ArrayList<String> iconSlots = **new** ArrayList<>();
18. ArrayList<StatusBarIcon> icons = **new** ArrayList<>();
19. //调用 SBMS 的注册接口注册：IStatusBarService注册一些信息
20. mBarService.registerStatusBar(mCommandQueue, iconSlots, icons, switches, binders,  fullscreenStackBounds, dockedStackBounds);
22. }
23. }

获取IStatusBarService的实例。\*\*IStatusBarService是一个系统服务，由ServerThread

启动并常驻system\_server进程中。IStatusBarService为那些对状态栏感兴趣的其他系统服务定义了一系列API，然而对SystemUI而言，它更像是一个客户端。因为IStatusBarService会将操作状态栏的请求发送给SystemUI，并由后者完成请求

随后BaseStatusBar将自己注册到IStatusBarService之中。以此声明本实例才是状态栏的真正实现者，IStatusBarService会将其所接受到的请求转发给本实例。

“天有不测风云”，SystemUI难免会因为某些原因使得其意外终止。而状态栏中所显示的信息并不属于状态栏自己，而是属于其他的应用程序或是其他的系统服务。因此当SystemUI重新启动时，便需要恢复其终止前所显示的信息以避免信息的丢失。为此，IStatusBarService中保存了所有的需要状态栏进行显示的信息的副本，并在新的状态栏实例启动后，这些副本将会伴随着注册的过程传递给状态栏并进行显示，从而避免了信息的丢失。 从代码分析的角度来看，这一从IstatusBarService中取回信息副本的过程正好完整地体现了状态栏 所能显示的信息的类型

iconList是向IStatusBarService进行注册的参数之一。它保存了用于显示在状态栏的系统状态区中的状态图标列表。在完成注册之后，IStatusBarService将会在其中填充两个数组，一个字符串数组用于表示状态的名称，一个StatusBarIcon类型的数组用于存储需要显示的图标资源。

mCommandQueue是CommandQueue类的一个实例。CommandQueue继承自IStatusBar.Stub。因此它是IStatusBar的Bn端。在完成注册后，这一Binder对象的Bp端将会保存IStatusBarService之中。因此它是IStatusBarService与BaseStatusBar进行通信的桥梁。

public class CommandQueue extends IStatusBar.Stub {

IStatusBarService的真身是StatusBarManagerService

可见，状态栏与导航栏的启动分为如下几个过程：

* 获取IStatusBarService，IStatusBarService是运行于system\_server的一个系统服务，它接受操作状态栏/导航栏的请求并将其转发给BaseStatusBar。为了保证SystemUI意外退出后不会发生信息丢失，IStatusBarService保存了所有需要状态栏与导航栏进行显示或处理的信息副本。
* 将一个继承自IStatusBar.Stub的CommandQueue的实例注册到IStatusBarService以建立通信，并将信息副本取回。
* 通过调用子类的createAndAddWindows()方法完成状态栏与导航栏的控件树及窗口的创建与显示。
* 使用从IStatusBarService取回的信息副本。

# StatusBarManager

StatusBarManager提供了一些有用的接口，像disable()方法正是我们需要的，一些系统级的应用也是调用的该方法禁止StatusBar下拉的，比如电话、锁屏模块。想要调用该方法，你还需要以下权限：

<uses-permissionandroid:name="android.permission.STATUS\_BAR" />

<uses-permissionandroid:name="android.permission.EXPAND\_STATUS\_BAR"/>

StatusBarManager提供了一些有用的接口，像disable()方法正是我们需要的，一些系统级的应用也是调用的该方法禁止StatusBar下拉的，比如电话、锁屏模块。想要调用该方法，你还需要以下权限：

<uses-permissionandroid:name="android.permission.STATUS\_BAR" />

<uses-permissionandroid:name="android.permission.EXPAND\_STATUS\_BAR"/>

## 例子1

公司有个需求当拨打电话在前台执行时，状态栏下拉不了，而在后台执行时，状态栏则可以下拉，此时需要修改的是phone应用。

在电话的呼出流程中，我们最后需要按下拨号键，才能将电话拨打出去，那么在按下拨号键之后，我们可以看到会弹出一个界面（1），显示拨号信息以及一些其他信息，这个界面就是我们的InCallScreen界面。当然，在来电（2）的时候，弹出的界面依然是InCallScreen，在我们接通电话（3）之后显示的那个界面仍然是InCallScreen。也就是说在通话过程中，我们一直可见并操作的那个界面就是InCallScreen

在类com.android.phone. InCallScreen.java

private StatusBarManager mStatusBarManager;

mStatusBarManager = (StatusBarManager)getSystemService(Context.STATUS\_BAR\_SERVICE);

@Override

protected void onResume() {

if (DBG) log("onResume()...");

super.onResume();

mStatusBarManager.disable(StatusBarManager.DISABLE\_EXPAND);

…………………..

}

@Override

protected void onPause() {

if (DBG) log("onPause()...");

super.onPause();

mStatusBarManager.disable(StatusBarManager.DISABLE\_NONE);

…………..

}

## 例子2

公司有个需求在安全模式下状态栏下拉不了，android模式下状态栏可以下拉（两个模式的切换广播通知），这个需求我们可以使用上述方法，也可以直接在systemui中修改，此时我们使用第二种方法。

status\_bar.xml：下拉部分就是触摸这个View，因此可以在这个视图中查看触摸事件。（状态栏部分：包括浮动通知，耳机、蓝牙、电池、信号、闹钟等图标）

<com.android.systemui.statusbar.phone.PhoneStatusBarView

………………..

packagecom.android.systemui.statusbar.phone;

public classPhoneStatusBarView extends PanelBar {

@Override

public boolean panelsEnabled() { //可以在此处修改

return ((mBar.mDisabled & StatusBarManager.DISABLE\_EXPAND) == 0);

}

}

上述方法中可以修改为如下：

public boolean panelsEnabled() {

if(安全模式){

return false;

}else{

return ((mBar.mDisabled &StatusBarManager.DISABLE\_EXPAND) == 0);

}

}

packagecom.android.systemui.statusbar.phone;

public classPanelBar extends FrameLayout {

@Override

publicboolean onTouchEvent(MotionEvent event) {

// Allow subclasses to implement enable/disable semantics

if (!panelsEnabled()) {

if (event.getAction() == MotionEvent.ACTION\_DOWN) {

Slog.v(TAG,String.format("onTouch: all panels disabled, ignoring touch at(%d,%d)",

(int)event.getX(), (int) event.getY()));

}

return false;

}

public booleanpanelsEnabled() {//父类PhoneStatusBarView重写了

return true;

}

…..

}

SystemUI状态栏禁止下拉

<http://www.voidcn.com/article/p-aldyshja-sy.html>

## 物理见切换面板的显示

收起扩充面板

/\*\*

\* Expand the notifications panel.

\*/

public void expandNotificationsPanel() {

try {

final IStatusBarService svc = getService();

if (svc != null) {

svc.expandNotificationsPanel();

}

} catch (RemoteException ex) {

throw ex.rethrowFromSystemServer();

}

}

/\*\*

\* Collapse the notifications and settings panels.

\*/

public void collapsePanels() {

try {

final IStatusBarService svc = getService();

if (svc != null) {

svc.collapsePanels();

}

} catch (RemoteException ex) {

throw ex.rethrowFromSystemServer();

}

}

/\*\*

\* Expand the settings panel.

\*/

public void expandSettingsPanel() {

expandSettingsPanel(null);

}