**Android Wallpaper之设置壁纸流程**

2018年04月05日 14:36:18 [风二中](https://me.csdn.net/lj527409) 阅读数：1657

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/lj527409/article/details/79825015

**What?**

什么是壁纸？

android wallpaper包括锁屏壁纸和桌面壁纸，壁纸又区分静态和动态两种。我们每天使用手机第一眼看到的就是壁纸，好看的壁纸对于手机的颜值也有大大的提升（滑稽），就让我们对壁纸一探究竟吧。

本文基于Android 8.1源码，相关文件如下：

1./frameworks/base/services/core/java/com/android/server/wallpaper/WallpaperManagerService.java

2./frameworks/base/services/core/java/android/app/WallpaperManager.java

3./frameworks/base/core/java/android/service/wallpaper/WallpaperService.java

4./frameworks/base/core/java/android/service/wallpaper/IwallpaperService.aidl

5./frameworks/base/packages/apps/SystemUI/src/com/android/systemui/statusbar/phone/LockscreenWallpaper.java

6./frameworks/base/packages/SystemUI/src/com/android/systemui/ImageWallpaper.java

7./frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/phone/StatusBar.java

**How?**

1.作为开发者如何去设置壁纸？

2.壁纸是怎么显示的？

3.壁纸存储在什么位置？

**如何设置壁纸？举个栗子。**



咱们手机中一般有内置主题的应用，在这里可以下载使用非常多好看的壁纸，点击即可设置为锁屏或桌面壁纸。

设置方法很简单，look:

1. WallpaperManager wallpaperManager = WallpaperManager.getInstance(this);
2. try {
3. wallpaperManager.setStream(InputStream,null,true,WallpaperManager.FLAG\_LOCK);
4. } catch (IOException e) {
5. e.printStackTrace();
6. }

**三步骤：**

1.添加设置壁纸的权限

<uses-permission android:name="android.permission.SET\_WALLPAPER"/>

2.获取WallpaperManager对象

3.设置壁纸，四个参数分别对应：

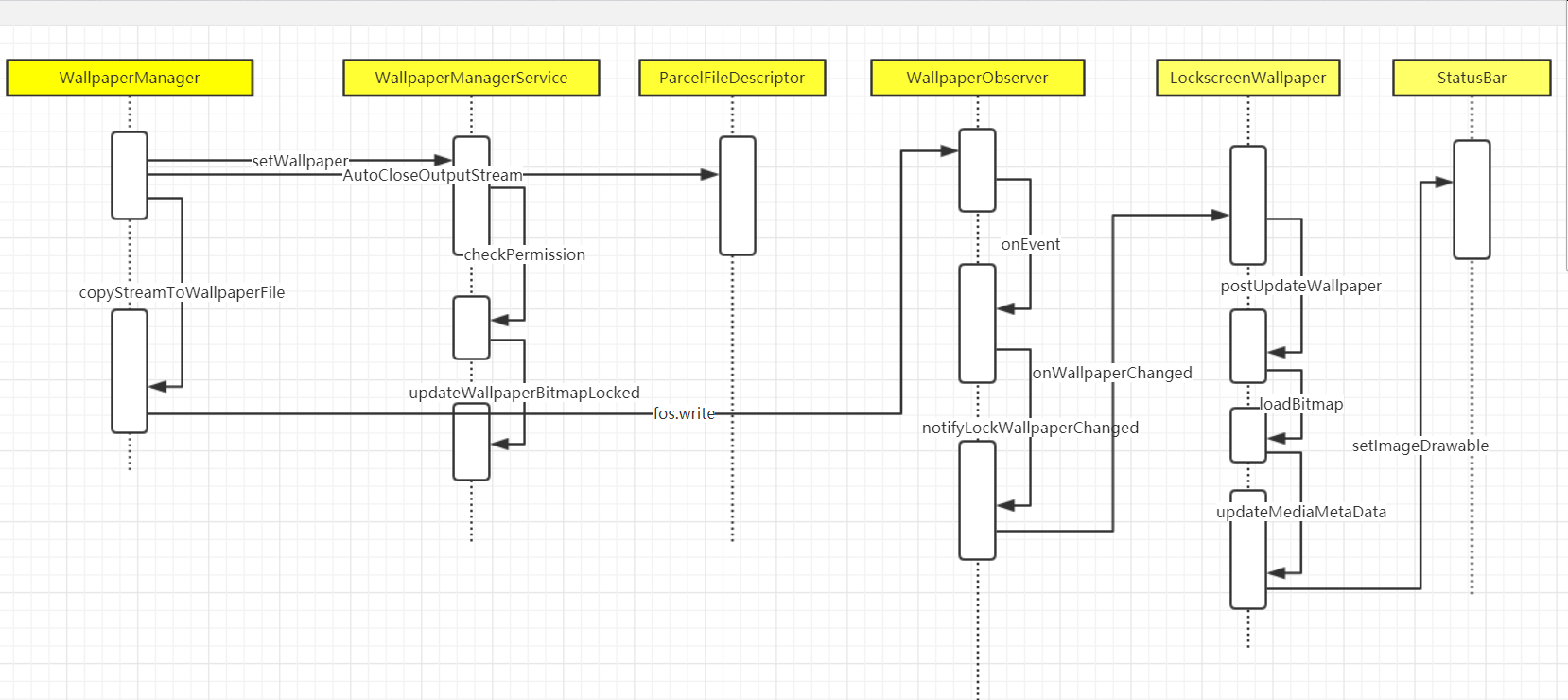
a.InputStream:图片对于的输入流

b.visibleCropHint:图片裁剪相关，一般默认为null

c.allowBack:是否允许回退

d.which:壁纸分为锁屏壁纸和桌面壁纸，所以需要设置FLGA:锁屏壁纸--WallpaperManager.FLAG\_LOCK,桌面壁纸：WallpaperManager.FLAG\_SYSTEM

**壁纸是如何显示的？**



下面就对如上图中的每个过程做一个简单的分析：

**1.setStream**

1. try {
2. *//sGlobals.mService即WallpaperManagerService*
3. ParcelFileDescriptor fd = sGlobals.mService.setWallpaper(null,
4. mContext.getOpPackageName(), visibleCropHint, allowBackup,
5. result, which, completion, UserHandle.myUserId());
6. if (fd != null) {
7. FileOutputStream fos = null;
8. try {
9. *//将壁纸copy一份并存储到对应目录，默认是/data/system/users/0/wallpaper(或wallpaper\_lock),其中0是主用户的userId，支持多用户*
10. fos = new ParcelFileDescriptor.AutoCloseOutputStream(fd);
11. copyStreamToWallpaperFile(bitmapData, fos);
12. fos.close();
13. completion.waitForCompletion();
14. } finally {
15. IoUtils.closeQuietly(fos);
16. }
17. }
18. } catch (RemoteException e) {
19. throw e.rethrowFromSystemServer();
20. }
21. }

**2.WallpaperManagerService.java#setWallpaper**

1. @Override
2. public ParcelFileDescriptor setWallpaper(String name, String callingPackage,
3. Rect cropHint, boolean allowBackup, Bundle extras, int which,
4. IWallpaperManagerCallback completion, int userId) {
6. *//检查有没有设置壁纸的权限*
7. checkPermission(android.Manifest.permission.SET\_WALLPAPER);
8. *//调用setStream方法的时候参数which必须是正确的*
9. if ((which & (FLAG\_LOCK|FLAG\_SYSTEM)) == 0) {
10. final String msg = "Must specify a valid wallpaper category to set";
11. Slog.e(TAG, msg);
12. throw new IllegalArgumentException(msg);
13. }
15. */\* If we're setting system but not lock, and lock is currently sharing the system*
16. *\* wallpaper, we need to migrate that image over to being lock-only before*
17. *\* the caller here writes new bitmap data.*
18. *\*/*
19. *//如果当前没有锁屏壁纸的话，并且是设置桌面壁纸即which == FLAG\_SYSTEM，那么同时设置为锁屏壁纸*
20. if (which == FLAG\_SYSTEM && mLockWallpaperMap.get(userId) == null) {
21. if (DEBUG) {
22. Slog.i(TAG, "Migrating system->lock to preserve");
23. }
24. migrateSystemToLockWallpaperLocked(userId);
25. }
27. ParcelFileDescriptor pfd = updateWallpaperBitmapLocked(name, wallpaper, extras);
28. }

**3.最主要的方法：WallpaperObserver#onEvent**

WallpaperObserver是WallpaperManagerservice.java的内部类，它的主要职责是监听文件变化，也就是壁纸对应的文件更新,看下源码中关于它的注释：

1. */\*\**
2. *\* Observes the wallpaper for changes and notifies all IWallpaperServiceCallbacks*
3. *\* that the wallpaper has changed. The CREATE is triggered when there is no*
4. *\* wallpaper set and is created for the first time. The CLOSE\_WRITE is triggered*
5. *\* every time the wallpaper is changed.*
6. *\*/*

监听wallpaper变化并通知IWallpaperServiceCallbacks,前文提到的LockscreenWallpaper就是继承了

IWallpaperServiceCallbacks，并重写了它的onWallppaerChanged方法，在这里更新锁屏壁纸的。

1. @Override
2. public void onEvent(int event, String path) {
3. *//如果是锁屏壁纸更新*
4. if (moved && lockWallpaperChanged) {
5. notifyLockWallpaperChanged();
6. *//android 8.0新增的一个变化，锁屏包括下拉快捷的主题会根据当前的壁纸来变化，避免壁纸和锁屏的图标颜色一致导致的显示不清问题，但是有一个缺陷就是：*
7. *//获取的是当前壁纸的主色调，而不是某个区域的主色调，这样就会导致虽然主色调是白色，比如时间的区域是黑色，这一点小米做的比较好，它是根据当前区域的壁纸的主色调来进行反色的。*
8. notifyWallpaperColorsChanged(wallpaper, FLAG\_LOCK);
9. return;
10. }
12. if (sysWallpaperChanged || lockWallpaperChanged) {
13. notifyCallbacksLocked(wallpaper);
15. }
17. if (sysWallpaperChanged) {
18. *//桌面壁纸变化，那么bind ImageWallpaper，ImageWallpaper是负责显示静态桌面壁纸的*
19. *// If this was the system wallpaper, rebind...*
20. bindWallpaperComponentLocked(mImageWallpaper, true,
21. false, wallpaper, null);
22. notifyColorsWhich |= FLAG\_SYSTEM;
23. }
25. if (lockWallpaperChanged
26. || (wallpaper.whichPending & FLAG\_LOCK) != 0) {
27. if (DEBUG) {
28. Slog.i(TAG, "Lock-relevant wallpaper changed");
29. }
30. *// either a lock-only wallpaper commit or a system+lock event.*
31. *// if it's system-plus-lock we need to wipe the lock bookkeeping;*
32. *// we're falling back to displaying the system wallpaper there.*
33. *//如果参数which是system+lock，也就是同时设置锁屏和桌面壁纸，那么remove锁屏壁纸，因为已经是同一张壁纸了*
34. if (!lockWallpaperChanged) {
35. mLockWallpaperMap.remove(wallpaper.userId);
36. }
37. *// and in any case, tell keyguard about it*
38. notifyLockWallpaperChanged();
39. notifyColorsWhich |= FLAG\_LOCK;
40. }
42. }

**先看锁屏壁纸更新这一部分：**

1. void notifyLockWallpaperChanged() {
2. final IWallpaperManagerCallback cb = mKeyguardListener;
3. if (cb != null) {
4. try {
5. cb.onWallpaperChanged();
6. } catch (RemoteException e) {
7. *// Oh well it went away; no big deal*
8. }
9. }
10. }

mKeyguardListener赋值的地方：

1. @Override
2. public boolean setLockWallpaperCallback(IWallpaperManagerCallback cb) {
3. checkPermission(android.Manifest.permission.INTERNAL\_SYSTEM\_WINDOW);
4. synchronized (mLock) {
5. mKeyguardListener = cb;
6. }
7. return true;
8. }

前面我们说过LockscreenWallpaper.java是继承了IWallpaperManagerCallback的，那么setLockWallpaperCallback调用的地方应该是在这里：

1. public LockscreenWallpaper(Context ctx, PhoneStatusBar bar, Handler h) {
3. mService = IWallpaperManager.Stub.asInterface(
4. ServiceManager.getService(Context.WALLPAPER\_SERVICE));
5. mWallpaperManager = (WallpaperManager) ctx.getSystemService(Context.WALLPAPER\_SERVICE);
6. try {
7. *//在这里给mKeyguardListener赋值的*
8. mService.setLockWallpaperCallback(this);
9. } catch (RemoteException e) {
10. Log.e(TAG, "System dead?" + e);
11. }
12. }

4.LockscreenWallpaper.java#onWallpaperChanged：

1. @Override
2. public void onWallpaperChanged() {
3. *// Called on Binder thread.*
4. mH.removeCallbacks(this);
5. mH.post(this);
6. }

**LockscreenWallpaper实现了Runnable接口的，所以看下它的run方法：**

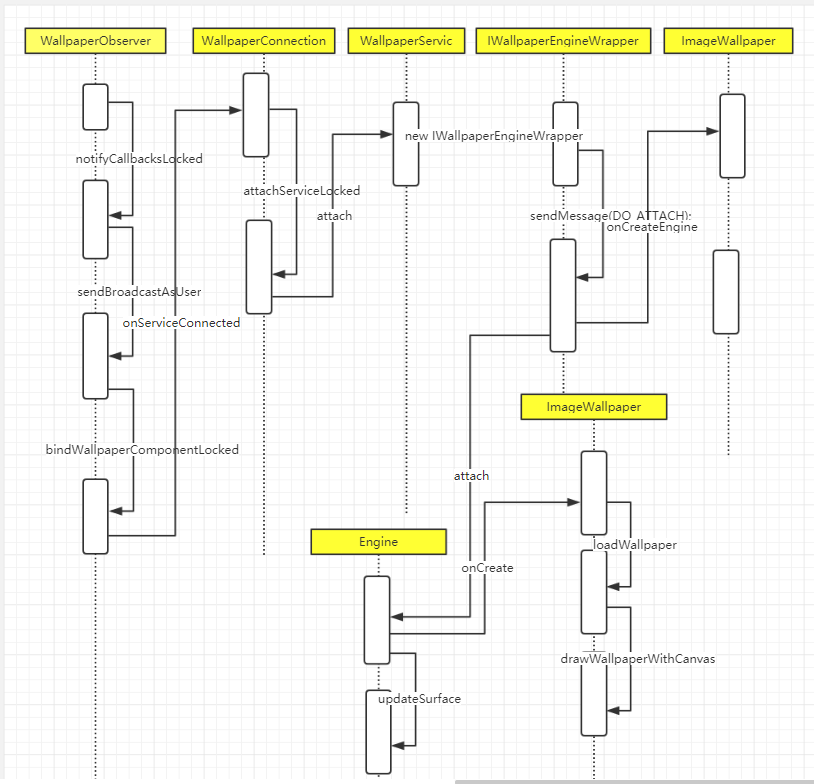
1. @Override
2. public void run() {
3. *// Called in response to onWallpaperChanged on the main thread.*
4. mLoader = new AsyncTask<Void, Void, LoaderResult>() {
5. @Override
6. protected LoaderResult doInBackground(Void... params) {
7. return loadBitmap(currentUser, selectedUser);
8. }
9. @Override
10. protected void onPostExecute(LoaderResult result) {
11. super.onPostExecute(result);
12. if (isCancelled()) {
13. return;
14. }
15. if (result.success) {
16. mCached = true;
17. mCache = result.bitmap;
18. mUpdateMonitor.setHasLockscreenWallpaper(result.bitmap != null);
19. *//通知StatsuBar更新壁纸*
20. mBar.updateMediaMetaData(
21. true */\* metaDataChanged \*/*, true */\* allowEnterAnimation \*/*);
22. }
23. mLoader = null;
24. }
25. }.executeOnExecutor(AsyncTask.THREAD\_POOL\_EXECUTOR);
26. }

异步获取壁纸，并通知StatusBar去更新壁纸。

**5.StatusBar.java#updateMediaMetaData:**

这里主要就是对锁屏壁纸所在的View做最基本的setImageBitmap。

**6.再看桌面壁纸部分：**



**6.1 bindWallpaperComponentLocked**

1. if (sysWallpaperChanged) {
2. *// If this was the system wallpaper, rebind...*
3. bindWallpaperComponentLocked(mImageWallpaper, true,
4. false, wallpaper, null);
5. notifyColorsWhich |= FLAG\_SYSTEM;
6. }

mImageWallpaper = ComponentName.unflattenFromString(  
                context.getResources().getString(R.string.image\_wallpaper\_component));

也就是一开始提到的：

/frameworks/base/packages/SystemUI/src/com/android/systemui/ImageWallpaper.java

1. boolean bindWallpaperComponentLocked(ComponentName componentName, boolean force,
2. boolean fromUser, WallpaperData wallpaper, IRemoteCallback reply) {

5. Intent intent = new Intent(WallpaperService.SERVICE\_INTERFACE);

8. WallpaperConnection newConn = new WallpaperConnection(wi, wallpaper);
9. *//componentName就是ImageWallpaper*
10. intent.setComponent(componentName);
11. intent.putExtra(Intent.EXTRA\_CLIENT\_LABEL,
12. com.android.internal.R.string.wallpaper\_binding\_label);
13. intent.putExtra(Intent.EXTRA\_CLIENT\_INTENT, PendingIntent.getActivityAsUser(
14. mContext, 0,
15. Intent.createChooser(new Intent(Intent.ACTION\_SET\_WALLPAPER),
16. mContext.getText(com.android.internal.R.string.chooser\_wallpaper)),
17. 0, null, new UserHandle(serviceUserId)));
18. }

ImageWallpaper继承了Service，既然是bindService，那么主要看下conn,也就是WallpaperConnection。

它的onServiceConnected方法：

1. @Override
2. public void onServiceConnected(ComponentName name, IBinder service) {
3. synchronized (mLock) {
4. if (mWallpaper.connection == this) {
5. mService = IWallpaperService.Stub.asInterface(service);
6. attachServiceLocked(this, mWallpaper);
7. *// XXX should probably do saveSettingsLocked() later*
8. *// when we have an engine, but I'm not sure about*
9. *// locking there and anyway we always need to be able to*
10. *// recover if there is something wrong.*
11. saveSettingsLocked(mWallpaper.userId);
12. FgThread.getHandler().removeCallbacks(mResetRunnable);
13. }
14. }
15. }

**6.2 继续看attachServcieLocked方法：**

1. void attachServiceLocked(WallpaperConnection conn, WallpaperData wallpaper) {
2. try {
3. conn.mService.attach(conn, conn.mToken,
4. TYPE\_WALLPAPER, false,
5. wallpaper.width, wallpaper.height, wallpaper.padding);
6. } catch (RemoteException e) {
7. Slog.w(TAG, "Failed attaching wallpaper; clearing", e);
8. if (!wallpaper.wallpaperUpdating) {
9. bindWallpaperComponentLocked(null, false, false, wallpaper, null);
10. }
11. }
12. }

conn.mService.attach是调用了IWallpaperServiceWrapper 的attach方法，IWallpaperServiceWrapper 继承了

IWallpaperService.Stub。

1. @Override
2. public void attach(IWallpaperConnection conn, IBinder windowToken,
3. int windowType, boolean isPreview, int reqWidth, int reqHeight, Rect padding) {
4. new IWallpaperEngineWrapper(mTarget, conn, windowToken,
5. windowType, isPreview, reqWidth, reqHeight, padding);
6. }

在看它的构造方法，发送了一个DO\_ATTACH的消息：

1. Message msg = mCaller.obtainMessage(DO\_ATTACH);
2. mCaller.sendMessage(msg);
3. case DO\_ATTACH: {
4. try {
5. mConnection.attachEngine(this);
6. } catch (RemoteException e) {
7. Log.w(TAG, "Wallpaper host disappeared", e);
8. return;
9. }
10. Engine engine = onCreateEngine();
11. mEngine = engine;
12. mActiveEngines.add(engine);
13. engine.attach(this);
14. return;
15. }

onCreateEngine也是一个抽象的方法：

1. */\*\**
2. *\* Must be implemented to return a new instance of the wallpaper's engine.*
3. *\* Note that multiple instances may be active at the same time, such as*
4. *\* when the wallpaper is currently set as the active wallpaper and the user*
5. *\* is in the wallpaper picker viewing a preview of it as well.*
6. *\*/*
7. public abstract Engine onCreateEngine();

实现的地方仍然是在ImageWallpaper.java里

1. @Override
2. public Engine onCreateEngine() {
3. mEngine = new DrawableEngine();
4. return mEngine;
5. }

DrawableEngine是自定义的继承Engine的内部类

最后调用engine.attach方法。

WallpaperService.java的attach方法：

1. void attach(IWallpaperEngineWrapper wrapper) {
2. onCreate(mSurfaceHolder);
4. mInitializing = false;
5. mReportedVisible = false;
6. updateSurface(false, false, false);
7. }

**6.3 onCreate(mSurfaceHolder)**

它是一个抽象方法

1. */\*\**
2. *\* Called once to initialize the engine. After returning, the*
3. *\* engine's surface will be created by the framework.*
4. *\*/*
5. public void onCreate(SurfaceHolder surfaceHolder) {
6. }

它是一个抽象方法，那么真正的实现是在它的子类，也就是ImageWallpaper.java里

1. @Override
2. public void onCreate(SurfaceHolder surfaceHolder) {
3. if (DEBUG) {
4. Log.d(TAG, "onCreate");
5. }
6. super.onCreate(surfaceHolder);
7. mDefaultDisplay = getSystemService(WindowManager.class).getDefaultDisplay();
8. setOffsetNotificationsEnabled(false);
9. updateSurfaceSize(surfaceHolder, getDefaultDisplayInfo(), false */\* forDraw \*/*);
10. }

surfaceHolder是在父类里初始化的，surfaceHolder = new BaseSurfaceHolder();

**6.4 updateSurfaceSize**

在这里主要是继续调用loadWallpaper方法去解析壁纸并最终绘制到surfaceHolder上。

**6.5 drawFrame**

对壁纸进行一些裁剪操作，根据是否支持硬件加速来决定绘制的方法：

1. *//支持硬件加速*
2. if (mIsHwAccelerated) {
3. if (!drawWallpaperWithOpenGL(sh, availw, availh, xPixels, yPixels)) {
4. drawWallpaperWithCanvas(sh, availw, availh, xPixels, yPixels);
5. }
6. } else {
7. drawWallpaperWithCanvas(sh, availw, availh, xPixels, yPixels);
8. if (FIXED\_SIZED\_SURFACE) {
9. *// If the surface is fixed-size, we should only need to*
10. *// draw it once and then we'll let the window manager*
11. *// position it appropriately. As such, we no longer needed*
12. *// the loaded bitmap. Yay!*
13. *// hw-accelerated renderer retains bitmap for faster rotation*
14. unloadWallpaper(false */\* forgetSize \*/*);
15. }
16. }

到这里，把壁纸的设置的简单过程基本上就讲完了，作为笔记做一个记录。

如有错误的地方，欢迎指正。