# QA

Dfg

 如何创建另外一个用户？如何区分访客用户及其他用户？各种用户的区别是什么？

 Android是怎么限制最多用户数量的？Flyme最多可以创建几个，在哪里控制了？

 切换用户做了些什么操作？第三方应用有什么办法知道用户切换了？如何知道自己当前处于哪个用户？

 用户切换后，原来的进程怎么处理的？SystemUI是新的进程吗？微信是开了两个进程吗，进程ID是怎么分配的，有什么特点？

 不同用户是如何分别存储数据的？多个用户之间如何共享数据

https://www.jianshu.com/p/0833e8a80025

# 概述

***framework/base/services/core/java/com/android/server/pm/UserManagerService.java***

***framework/base/core/java/android/os/UserManager.java***

***framework/base/core/java/android/content/pm/UserInfo.java***

***framework/base/core/java/android/os/UserHandle.java***

Android多用户模型，通过UserManagerService(以下简称为UMS)对多用户进行创建、删除、查询等管理操作。

不同用户的设置各不相同，并且不同用户安装的应用及应用数据也不相同。但是系统中和硬件相关的设置则是共用的，如网络设置等。

用户切换后前面用户运行的后台进程还可以继续运行，这样进行用户切换时无须中断一些后台进行的耗时操作（如下载）

CS架构

* Binder服务端：UserManagerService继承于IUserManager.Stub，作为binder服务端；
* Binder客户端：UserManager的成员变量mService继承于IUserManager.Stub.Proxy，该变量作为binder客户端。UserManager的大部分核心工作都是交由其成员变量mService，再通过binder调用到UserManagerService所相应的方法

### 1.1 概念

先来介绍userId, uid, appId，SharedAppGid这几个概念的关系

* userId：是指用户Id；
* appId： 是指跟用户空间无关的应用程序id；取值范围 0<= appId <100000
* uid:是指跟用户空间紧密相关的应用程序id;
* SharedAppGid:是指可共享的应用id；

转换关系：

uid = userId \* 100000 + appId

SharedAppGid = 40000 + appId

另外，PER\_USER\_RANGE = 100000， 意味着每个user空间最大可以有100000个appid。这些区间分布如下：

* system appid: [1000, 9999]
* application appid:[10000, 19999]
* Shared AppGid: [50000, 59999]
* isolated appid: [99000, 99999]

### UserHandle

### 1.2 UserHandle

常见方法：

| **方法** | **含义** |
| --- | --- |
| isSameUser | 比较两个uid的userId是否相同 |
| isSameApp | 比较两个uid的appId是否相同 |
| isApp | appId是否属于区间[10000,19999] |
| isIsolated | appId是否属于区间[99000,99999] |
| getIdentifier | 获取UserHandle所对应的userId |

常见成员变量：(UserHandle的成员变量mHandle便是userId)

| **userId** | **赋值** | **含义** |
| --- | --- | --- |
| USER\_OWNER | 0 | 拥有者 |
| USER\_SYSTEM = 0; |  |  |
| USER\_ALL | -1 | 所有用户 |
| USER\_CURRENT | -2 | 当前活动用户 |
| USER\_CURRENT\_OR\_SELF | -3 | 当前用户或者调用者所在用户 |
| USER\_NULL | -1000 | 未定义用户 |

类成员变量：

UserHandle OWNER = new UserHandle(USER\_OWNER); // 0

UserHandle ALL = new UserHandle(USER\_ALL); // -1

UserHandle CURRENT = new UserHandle(USER\_CURRENT); // -2

UserHandle CURRENT\_OR\_SELF = new UserHandle(USER\_CURRENT\_OR\_SELF); // -3

关于UID默认情况下，客户端可使用rocess.myUserHandle()； 服务端可使用UserHandle.getCallingUserId();

### 解析UID

执行adb shell ps命令输出当前系统所有进程信息，以下列举部分进程，其中第一列代表的是当前进程 所属的UID。

USER PID PPID VSIZE RSS WCHAN PC NAME

root 1 0 2108 1068 SyS\_epoll\_ 0000000000 S /init

root 2 0 0 0 kthreadd 0000000000 S kthreadd

root 275 1 13276 1640 hrtimer\_na 0000000000 S /system/bin/vold

root 319 1 4148 904 SyS\_epoll\_ 0000000000 S /system/bin/lmkd

system 320 1 3848 1072 binder\_thr 0000000000 S /system/bin/servicemanager

system 321 1 138396 10324 SyS\_epoll\_ 0000000000 S /system/bin/surfaceflinger

root 331 1 4472 792 \_\_skb\_recv 0000000000 S /system/bin/debuggerd64

radio 333 1 71240 4928 hrtimer\_na 0000000000 S /system/bin/rild

drm 334 1 20128 2048 binder\_thr 0000000000 S /system/bin/drmserver

media 336 1 122404 9536 binder\_thr 0000000000 S /system/bin/mediaserver

root 338 1 3880 736 unix\_strea 0000000000 S /system/bin/installd

root 359 1 1670784 40732 poll\_sched 0000000000 S zygote64

system 1586 359 1880668 134868 SyS\_epoll\_ 0000000000 S system\_server

radio 3341 359 1371948 55264 SyS\_epoll\_ 0000000000 S com.android.phone

media\_rw 3409 275 19676 13828 inotify\_re 0000000000 S /system/bin/sdcard

u0\_a23 4152 359 1301552 27184 SyS\_epoll\_ 0000000000 S com.android.incallui

u0\_a89 10920 360 1118036 129224 SyS\_epoll\_ 0000000000 S com.tencent.mobileqq

u0\_a94 15481 360 939456 52004 SyS\_epoll\_ 0000000000 S com.sina.weibo

..

可以看到UID有root, system,radio,media等都属于系统uid定义在在Process.java文件，如下：

| **uid** | **值** | **含义** |
| --- | --- | --- |
| ROOT\_UID | 0 | root uid |
| SYSTEM\_UID | 1000 | 用于systemserver进程 |
| PHONE\_UID | 1001 | telephony所属的uid |
| BLUETOOTH\_UID | 1002 | 蓝牙所属的uid |
| LOG\_UID | 1007 | log所属的uid |
| WIFI\_UID | 1008 |  |
| MEDIA\_UID | 1013 | 用于mediaserver进程 |
| VPN\_UID | 1016 |  |
| DRM\_UID | 1019 |  |
| MEDIA\_RW\_GID | 1023 | 具有写内部媒体存储权限的uid |
| NFC\_UID | 1027 |  |
| PACKAGE\_INFO\_GID | 1032 |  |
| SHARED\_RELRO\_UID | 1037 |  |
| SHELL\_UID | 2000 | shell uid |
| SHARED\_USER\_GID | 9997 |  |

除了系统UID，还有另一类普通的应用uid，命令以u开头，代表的是普通app的uid，例如u0\_a94代表的是uid=10094， 这个转换过程，见UserHandle.java的formatUid()方法：

**public static void** formatUid(StringBuilder sb, **int** uid) {  
 **if** (uid < Process.FIRST\_APPLICATION\_UID) {  
 sb.append(uid);  
 } **else** {  
 sb.append(**'u'**);  
 sb.append(getUserId(uid));  
 **final int** appId = getAppId(uid);  
 **if** (appId >= Process.FIRST\_ISOLATED\_UID && appId <= Process.LAST\_ISOLATED\_UID) {  
 sb.append(**'i'**);  
 sb.append(appId - Process.FIRST\_ISOLATED\_UID);  
 } **else if** (appId >= Process.FIRST\_APPLICATION\_UID) {  
 sb.append(**'a'**);  
 sb.append(appId - Process.FIRST\_APPLICATION\_UID);  
 } **else** {  
 sb.append(**'s'**);  
 sb.append(appId);  
 }  
 }  
}

举例说明：

u0i20 = 0 \* 100000 + (99000 + 20) = 99020

u1a30 = 1 \* 100000 + (10000 + 30) = 110030

u2s1001 = 2 \* 100000 + 1001 = 201001

### 1.4 UserInfo

UserInfo代表的是一个用户的信息，涉及到的flags及其含义，如下：

用户的权限类别

| **flags** | **含义** |
| --- | --- |
| FLAG\_PRIMARY | 主用户，只有一个user具有该标识 |
| FLAG\_ADMIN | 具有管理特权的用户，例如创建或删除其他用户 |
| FLAG\_GUEST | 访客用户，可能是临时的 |
| FLAG\_RESTRICTED | 限制性用户，较普通用户具有更多限制，例如禁止安装app或者管理wifi等 |
| FLAG\_INITIALIZED | 表明用户已初始化 |
| FLAG\_MANAGED\_PROFILE | 表明该用户是另一个用户的轮廓 |
| FLAG\_DISABLED | 表明该用户处于不可用状态 |

各种用户拥有不同的权限组合

**USER\_OWNER** 是 **PRIMARY** 和 **ADMIN** 的组合.

### UserState

//用户启动中

public final static int STATE\_BOOTING = 0;

//用户正常运行状态

public final static int STATE\_RUNNING = 1;

//用户正在停止中

public final static int STATE\_STOPPING = 2;

//用户处于关闭状态

public final static int STATE\_SHUTDOWN = 3;

用户生命周期线：

STATE\_BOOTING -> STATE\_RUNNING -> STATE\_STOPPING -> STATE\_SHUTDOWN.

可通过AMS.switchUser()来切换用户，并更新mCurrentUserId为新切换的用户。

# 启动流程

[-> PackageManagerService.java]

sUserManager = **new** UserManagerService(context, **this**,  
 **new** UserDataPreparer(mInstaller, mInstallLock, mContext, mOnlyCore), mPackages);

UMS是在PackageManagerService对象初始化的过程中创建。

## UserMS

UserManagerService(Context context, PackageManagerService pm, UserDataPreparer userDataPreparer,  
 Object packagesLock) {  
 **this**(context, pm, userDataPreparer, packagesLock, Environment.getDataDirectory());  
}  
  
**private** UserManagerService(Context context, PackageManagerService pm,  
 UserDataPreparer userDataPreparer, Object packagesLock, File dataDir) {  
 mContext = context;  
 mPm = pm;  
 mPackagesLock = packagesLock;  
 mHandler = **new** MainHandler();  
 mUserDataPreparer = userDataPreparer;  
 **synchronized** (mPackagesLock) {

//String USER\_INFO\_DIR = **"system"** + File.separator + **"users"**;

// //创建目录mUsersDir =/data/system/users  
 mUsersDir = **new** File(dataDir, USER\_INFO\_DIR);  
 mUsersDir.mkdirs();  
 *// Make zeroth user directory, for services to migrate their files to that location*

//创建目录/data/system/users/0File userZeroDir = **new** File(mUsersDir, String.valueOf(UserHandle.USER\_SYSTEM));  
 userZeroDir.mkdirs();  
 FileUtils.setPermissions(mUsersDir.toString(),  
 FileUtils.S\_IRWXU | FileUtils.S\_IRWXG | FileUtils.S\_IROTH | FileUtils.S\_IXOTH,  
 -1, -1);

// //mUserListFile文件路径为/data/system/users/userlist.xml  
 mUserListFile = **new** File(mUsersDir, USER\_LIST\_FILENAME);

initDefaultGuestRestrictions();

// //解析userlist.xml文件  
 readUserListLP();  
 sInstance = **this**;  
 }  
 mLocalService = **new** LocalService();  
 LocalServices.addService(UserManagerInternal.**class**, mLocalService);  
 mLockPatternUtils = **new** LockPatternUtils(mContext);  
 mUserStates.put(UserHandle.USER\_SYSTEM, UserState.STATE\_BOOTING);  
}

### initDefaultGuestRestrictions()\

// 初始化来宾账户的默认限制条件

private void initDefaultGuestRestrictions() {

    synchronized (mGuestRestrictions) {

        if (mGuestRestrictions.isEmpty()) {

        // "no\_config\_wifi"，不允许配置WiFi

            mGuestRestrictions.putBoolean(UserManager.DISALLOW\_CONFIG\_WIFI, true);

        // "no\_install\_unknown\_sources"，不允许安装未知来源的应用

            mGuestRestrictions.putBoolean(UserManager.DISALLOW\_INSTALL\_UNKNOWN\_SOURCES, true);

        // "no\_outgoing\_calls"，不允许呼叫电话

            mGuestRestrictions.putBoolean(UserManager.DISALLOW\_OUTGOING\_CALLS, true);

        // "no\_sms"，不允许收发短信

            mGuestRestrictions.putBoolean(UserManager.DISALLOW\_SMS, true);

        }

    }

}

### readUserListLP()

[java] view plain copy

// 从/data/system/users/userlist.xml文件读取用户信息

[html] view plain copy

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>

<users nextSerialNumber="10" version="5">

    <guestRestrictions>

        <restrictions no\_config\_wifi="true" no\_outgoing\_calls="true" no\_sms="true" />

    </guestRestrictions>

    <user id="0" />

</users>

---------------------

private final SparseArray<UserData> mUsers = new SparseArray<>();

private void readUserListLP() {

    // 如果文件不存在，则创建管理员用户并返回

    if (!mUserListFile.exists()) {

        fallbackToSingleUserLP();

        return;

    }

    FileInputStream fis = null;

    AtomicFile userListFile = new AtomicFile(mUserListFile);

    try {

        fis = userListFile.openRead();

        XmlPullParser parser = Xml.newPullParser();

        parser.setInput(fis, StandardCharsets.UTF\_8.name());

        int type;

        while ((type = parser.next()) != XmlPullParser.START\_TAG

                && type != XmlPullParser.END\_DOCUMENT) {

            // Skip

        }

        if (type != XmlPullParser.START\_TAG) {

            Slog.e(LOG\_TAG, "Unable to read user list");

        // 如果文件异常，则创建管理员用户并返回

            fallbackToSingleUserLP();

            return;

        }

        mNextSerialNumber = -1;

    // 解析文件

        if (parser.getName().equals(TAG\_USERS)) {

            String lastSerialNumber = parser.getAttributeValue(null, ATTR\_NEXT\_SERIAL\_NO);

            if (lastSerialNumber != null) {

                mNextSerialNumber = Integer.parseInt(lastSerialNumber);

            }

            String versionNumber = parser.getAttributeValue(null, ATTR\_USER\_VERSION);

            if (versionNumber != null) {

                mUserVersion = Integer.parseInt(versionNumber);

            }

        }

        final Bundle newDevicePolicyGlobalUserRestrictions = new Bundle();

        while ((type = parser.next()) != XmlPullParser.END\_DOCUMENT) {

            if (type == XmlPullParser.START\_TAG) {

                final String name = parser.getName();

                if (name.equals(TAG\_USER)) {

                    String id = parser.getAttributeValue(null, ATTR\_ID);

            // 初始化UserData对象保存从 /data/system/users/${id}.xml 文件中读取到的用户信息

                    UserData userData = readUserLP(Integer.parseInt(id));

                    if (userData != null) {

                        synchronized (mUsersLock) {

                // 把解析到的用户信息保存到mUsers中

                            mUsers.put(userData.info.id, userData);

                            if (mNextSerialNumber < 0

                                    || mNextSerialNumber <= userData.info.id) {

                                mNextSerialNumber = userData.info.id + 1;

                            }

                        }

                    }

                } else if (name.equals(TAG\_GUEST\_RESTRICTIONS)) {

                    while ((type = parser.next()) != XmlPullParser.END\_DOCUMENT

                            && type != XmlPullParser.END\_TAG) {

                        if (type == XmlPullParser.START\_TAG) {

                            if (parser.getName().equals(TAG\_RESTRICTIONS)) {

                                synchronized (mGuestRestrictions) {

                                    UserRestrictionsUtils

                                            .readRestrictions(parser, mGuestRestrictions);

                                }

                            } else if (parser.getName().equals(TAG\_DEVICE\_POLICY\_RESTRICTIONS)

                                    ) {

                                UserRestrictionsUtils.readRestrictions(parser,

                                        newDevicePolicyGlobalUserRestrictions);

                            }

                            break;

                        }

                    }

                } else if (name.equals(TAG\_GLOBAL\_RESTRICTION\_OWNER\_ID)) {

                    String ownerUserId = parser.getAttributeValue(null, ATTR\_ID);

                    if (ownerUserId != null) {

                        mGlobalRestrictionOwnerUserId = Integer.parseInt(ownerUserId);

                    }

                }

            }

        }

        synchronized (mRestrictionsLock) {

            mDevicePolicyGlobalUserRestrictions = newDevicePolicyGlobalUserRestrictions;

        }

    // 解析完文件后，更新用户ID

        updateUserIds();

    // 如果有必要，则升级Version

        upgradeIfNecessaryLP();

    } catch (IOException | XmlPullParserException e) {

        fallbackToSingleUserLP();

    } finally {

        IoUtils.closeQuietly(fis);

    }

}

#### fallbackToSingleUserLP

// 创建管理员用户

private void fallbackToSingleUserLP() {

    int flags = UserInfo.FLAG\_INITIALIZED;

    // In split system user mode, the admin and primary flags are assigned to the first human

    // user.

    if (!UserManager.isSplitSystemUser()) {

        flags |= UserInfo.FLAG\_ADMIN | UserInfo.FLAG\_PRIMARY;

    }

    // Create the system user

    UserInfo system = new UserInfo(UserHandle.USER\_SYSTEM, null, null, flags);

    UserData userData = new UserData();

    userData.info = system;

    synchronized (mUsersLock) {

        mUsers.put(system.id, userData);

    }

    mNextSerialNumber = MIN\_USER\_ID;

    mUserVersion = USER\_VERSION;

    Bundle restrictions = new Bundle();

    synchronized (mRestrictionsLock) {

        mBaseUserRestrictions.append(UserHandle.USER\_SYSTEM, restrictions);

    }

    // 更新用户ID

    updateUserIds();

    // 初始化来宾账户的默认限制条件

    initDefaultGuestRestrictions();

    /\*

     \* 把用户信息写到 /data/system/users/${id}.xml文件中，简单的写文件，不再看源码

     \* Writes the user file in this format:

     \*

     \* <user flags="20039023" id="0">

     \*   <name>Primary</name>

     \* </user>

     \*/

    writeUserLP(userData);

    /\*

     \* 把用户信息写到 /data/system/users/userlist.xml文件中

     \* Writes the user list file in this format:

     \*

     \* <users nextSerialNumber="3">

     \*   <user id="0"></user>

     \*   <user id="2"></user>

     \* </users>

     \*/

    writeUserListLP();

}

### UserData的定义

这样UserManagerService的初始化工作就完成了，主要的工作就是解析userlist.xml文件，并创建了mUsers列表中的UserData对象。

---------------------

作者：kaijiehui

来源：CSDN

原文：https://blog.csdn.net/u011341111/article/details/79458128

版权声明：本文为博主原创文章，转载请附上博文链接！

---------------------

作者：kaijiehui

来源：CSDN

原文：https://blog.csdn.net/u011341111/article/details/79458128

版权声明：本文为博主原创文章，转载请附上博文链接！

其中MainHandler是用于处理消息WRITE\_USER\_MSG的Handler，UMS初始化过程的主要功能：

* 创建目录/data/system/users
* 创建目录/data/system/users/0
* 解析目录/data/system/users中的userlist.xml获取所有用户id，再分别解析该目录下id.xml(比如1.xml)，并创建相应的UserInfo对象

## systemReady

在PackageManagerService. systemReady

1. @Override
2. **public** **void** systemReady() {
3. ...
4. sUserManager.systemReady();
5. ...
6. }

## 其他接口

Dsfg

### 2.UserData的定义

[java] view plain copy

private static class UserData {

    // Basic user information and properties

    UserInfo info;

    // Account name used when there is a strong association between a user and an account

    String account;

    // Account information for seeding into a newly created user. This could also be

    // used for login validation for an existing user, for updating their credentials.

    // In the latter case, data may not need to be persisted as it is only valid for the

    // current login session.

    String seedAccountName;

    String seedAccountType;

    PersistableBundle seedAccountOptions;

    // Whether to perist the seed account information to be available after a boot

    boolean persistSeedData;

    void clearSeedAccountData() {

        seedAccountName = null;

        seedAccountType = null;

        seedAccountOptions = null;

        persistSeedData = false;

    }

}

public class UserInfo implements Parcelable {

    /\*\* 8 bits for user type 用户类型\*/

    public static final int FLAG\_MASK\_USER\_TYPE = 0x000000FF;

    /\*\*

     \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NOTE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

     \* These flag values CAN NOT CHANGE because they are written

     \* directly to storage.

     \*/

    /\*\*

     \* Primary user. Only one user can have this flag set. It identifies the first human user

     \* on a device.主用户标志，通常是第一个ID为0的用户

     \*/

    public static final int FLAG\_PRIMARY = 0x00000001;

    /\*\*

     \* User with administrative privileges. Such a user can create and

     \* delete users.admin用户标志，有此标志才有创建和删除用户的权限

     \*/

    public static final int FLAG\_ADMIN   = 0x00000002;

    /\*\*

     \* Indicates a guest user that may be transient.guest用户标志

     \*/

    public static final int FLAG\_GUEST   = 0x00000004;

    /\*\*

     \* Indicates the user has restrictions in privileges, in addition to those for normal users.

     \* Exact meaning TBD. For instance, maybe they can't install apps or administer WiFi access pts.

     \* 标志权限受限的用户，具体受限功能未定

     \*/

    public static final int FLAG\_RESTRICTED = 0x00000008;

    /\*\*

     \* Indicates that this user has gone through its first-time initialization.

     \* 标志该用户是否已经初始化

     \*/

    public static final int FLAG\_INITIALIZED = 0x00000010;

    /\*\*

     \* Indicates that this user is a profile of another user, for example holding a users

     \* corporate data.标志该UserInfo是另一个用户的profile

     \*/

    public static final int FLAG\_MANAGED\_PROFILE = 0x00000020;

    /\*\*

     \* Indicates that this user is disabled.标志该用户已被禁止

     \*

     \* <p>Note: If an ephemeral user is disabled, it shouldn't be later re-enabled. Ephemeral users

     \* are disabled as their removal is in progress to indicate that they shouldn't be re-entered.

     \*/

    public static final int FLAG\_DISABLED = 0x00000040;

    public static final int FLAG\_QUIET\_MODE = 0x00000080;

    /\*\*

     \* Indicates that this user is ephemeral. I.e. the user will be removed after leaving

     \* the foreground.

     \*/

    public static final int FLAG\_EPHEMERAL = 0x00000100;

    public static final int NO\_PROFILE\_GROUP\_ID = UserHandle.USER\_NULL;

    public int id; // 用户ID

    public int serialNumber; // 用户的序列号，不会重复

    public String name; // 用户名称

    public String iconPath; // 用户头像路径

    public int flags; // 用户标志

    public long creationTime; // 创建用户的时间

    public long lastLoggedInTime; // 最后一次登录的时间

    public String lastLoggedInFingerprint; // 最后一次用指纹登录的时间

    public int profileGroupId; // 用户profile的group ID

    public int restrictedProfileParentId;

    /\*\* User is only partially created. \*/

    public boolean partial; // true表示该用户没有创建完成

    public boolean guestToRemove;

    ...

}

### 3.添加用户

UserManagerService中添加用户的方法是createUser()：

[java] view plain copy

@Override

public UserInfo createUser(String name, int flags) {

    if (DBG) Slog.i(LOG\_TAG, "createUser name " + name);

    // 检查添加用户的权限

    checkManageOrCreateUsersPermission(flags);

    return createUserInternal(name, flags, UserHandle.USER\_NULL);

}

#### createUserInternal

在该方法中首先会检查 用户是否被赋予了 **DISALLOW\_ADD\_USER** 权限，该权限禁止用户添加用户.

private UserInfo createUserInternal(String name, int flags, int parentId) {

    // 如果没有添加用户的权限则返回null

    if (hasUserRestriction(UserManager.DISALLOW\_ADD\_USER, UserHandle.getCallingUserId())) {

        Log.w(LOG\_TAG, "Cannot add user. DISALLOW\_ADD\_USER is enabled.");

        return null;

    }

    return createUserInternalUnchecked(name, flags, parentId);

}

#### createUserInternalUnchecked

private UserInfo createUserInternalUnchecked(String name, int flags, int parentId) {

    // 如果是一个低内存设备，则返回null

    if (ActivityManager.isLowRamDeviceStatic()) {

        return null;

    }

    final boolean isGuest = (flags & UserInfo.FLAG\_GUEST) != 0;

    final boolean isManagedProfile = (flags & UserInfo.FLAG\_MANAGED\_PROFILE) != 0;

    final boolean isRestricted = (flags & UserInfo.FLAG\_RESTRICTED) != 0;

    final long ident = Binder.clearCallingIdentity();

    UserInfo userInfo;

    UserData userData;

    final int userId;

    try {

        synchronized (mPackagesLock) {

            UserData parent = null;

            if (parentId != UserHandle.USER\_NULL) {

                synchronized (mUsersLock) {

            // 根据userId获取UserData信息

                    parent = getUserDataLU(parentId);

                }

                if (parent == null) return null;

            }

        // 判断是否可以添加更多profile

            if (isManagedProfile && !canAddMoreManagedProfiles(parentId, false)) {

                Log.e(LOG\_TAG, "Cannot add more managed profiles for user " + parentId);

                return null;

            }

        // 判断是否达到用户上限

            if (!isGuest && !isManagedProfile && isUserLimitReached()) {

                // If we're not adding a guest user or a managed profile and the limit has

                // been reached, cannot add a user.

                return null;

            }

            // If we're adding a guest and there already exists one, bail.

        // 如果创建的是guest用户且guest用户已经存在则返回

            if (isGuest && findCurrentGuestUser() != null) {

                return null;

            }

            // In legacy mode, restricted profile's parent can only be the owner user

            if (isRestricted && !UserManager.isSplitSystemUser()

                    && (parentId != UserHandle.USER\_SYSTEM)) {

                Log.w(LOG\_TAG, "Cannot add restricted profile - parent user must be owner");

                return null;

            }

            if (isRestricted && UserManager.isSplitSystemUser()) {

                if (parent == null) {

                    Log.w(LOG\_TAG, "Cannot add restricted profile - parent user must be "

                            + "specified");

                    return null;

                }

                if (!parent.info.canHaveProfile()) {

                    Log.w(LOG\_TAG, "Cannot add restricted profile - profiles cannot be "

                            + "created for the specified parent user id " + parentId);

                    return null;

                }

            }

            if (!UserManager.isSplitSystemUser() && (flags & UserInfo.FLAG\_EPHEMERAL) != 0) {

                Log.e(LOG\_TAG,

                        "Ephemeral users are supported on split-system-user systems only.");

                return null;

            }

            // In split system user mode, we assign the first human user the primary flag.

            // And if there is no device owner, we also assign the admin flag to primary user.

            if (UserManager.isSplitSystemUser()

                    && !isGuest && !isManagedProfile && getPrimaryUser() == null) {

                flags |= UserInfo.FLAG\_PRIMARY;

                synchronized (mUsersLock) {

                    if (!mIsDeviceManaged) {

                        flags |= UserInfo.FLAG\_ADMIN;

                    }

                }

            }

        // 获取下一个可用的userId

            userId = getNextAvailableId();

        // 创建/data/system/users/userId文件夹

            Environment.getUserSystemDirectory(userId).mkdirs();

            boolean ephemeralGuests = Resources.getSystem()

                    .getBoolean(com.android.internal.R.bool.config\_guestUserEphemeral);

            synchronized (mUsersLock) {

                // Add ephemeral flag to guests/users if required. Also inherit it from parent.

                if ((isGuest && ephemeralGuests) || mForceEphemeralUsers

                        || (parent != null && parent.info.isEphemeral())) {

                    flags |= UserInfo.FLAG\_EPHEMERAL;

                }

        // 初始化新用户

                userInfo = new UserInfo(userId, name, null, flags);

                userInfo.serialNumber = mNextSerialNumber++;

                long now = System.currentTimeMillis();

                userInfo.creationTime = (now > EPOCH\_PLUS\_30\_YEARS) ? now : 0;

        // 设置partial变量为true，表示用户还没有创建完成

                userInfo.partial = true;

                userInfo.lastLoggedInFingerprint = Build.FINGERPRINT;

                userData = new UserData();

                userData.info = userInfo;

                mUsers.put(userId, userData);

            }

        // 保存用户信息

            writeUserLP(userData);

            writeUserListLP();

            if (parent != null) {

                if (isManagedProfile) {

                    if (parent.info.profileGroupId == UserInfo.NO\_PROFILE\_GROUP\_ID) {

                        parent.info.profileGroupId = parent.info.id;

                        writeUserLP(parent);

                    }

                    userInfo.profileGroupId = parent.info.profileGroupId;

                } else if (isRestricted) {

                    if (parent.info.restrictedProfileParentId == UserInfo.NO\_PROFILE\_GROUP\_ID) {

                        parent.info.restrictedProfileParentId = parent.info.id;

                        writeUserLP(parent);

                    }

                    userInfo.restrictedProfileParentId = parent.info.restrictedProfileParentId;

                }

            }

        }

    // 为新建用户准备存储区域

        final StorageManager storage = mContext.getSystemService(StorageManager.class);

        storage.createUserKey(userId, userInfo.serialNumber, userInfo.isEphemeral());

        mPm.prepareUserData(userId, userInfo.serialNumber,

                StorageManager.FLAG\_STORAGE\_DE | StorageManager.FLAG\_STORAGE\_CE);

    // 保存所有安装应用在新建用户目录下的安装状态

        mPm.createNewUser(userId);

    // 创建完新用户后修改partial变量为false，表示用户创建完成，并重新保存用户信息

        userInfo.partial = false;

        synchronized (mPackagesLock) {

            writeUserLP(userData);

        }

        updateUserIds();

        Bundle restrictions = new Bundle();

        if (isGuest) {

            synchronized (mGuestRestrictions) {

                restrictions.putAll(mGuestRestrictions);

            }

        }

        synchronized (mRestrictionsLock) {

            mBaseUserRestrictions.append(userId, restrictions);

        }

    // 发送成功添加新用户的广播

        Intent addedIntent = new Intent(Intent.ACTION\_USER\_ADDED);

        addedIntent.putExtra(Intent.EXTRA\_USER\_HANDLE, userId);

        mContext.sendBroadcastAsUser(addedIntent, UserHandle.ALL,

                android.Manifest.permission.MANAGE\_USERS);

        MetricsLogger.count(mContext, isGuest ? TRON\_GUEST\_CREATED : TRON\_USER\_CREATED, 1);

    } finally {

        Binder.restoreCallingIdentity(ident);

    }

    return userInfo;

}

// 根据userId获取UserData信息

private UserData getUserDataLU(int userId) {

    final UserData userData = mUsers.get(userId);

    // If it is partial and not in the process of being removed, return as unknown user.

    if (userData != null && userData.info.partial && !mRemovingUserIds.get(userId)) {

        return null;

    }

    return userData;

}

final Settings mSettings;

/\*\* Called by UserManagerService \*/

// 保存所有安装应用在新建用户目录下的安装状态

void createNewUser(int userId) {

    synchronized (mInstallLock) {

    // 把所有已安装的应用数据拷贝到新建用户对应目录（/data/user/0/）下

        mSettings.createNewUserLI(this, mInstaller, userId);

    }

    synchronized (mPackages) {

    // 在/data/system/users/0/package-restrictions.xml文件中保存应用的限制信息

        scheduleWritePackageRestrictionsLocked(userId);

    // 更新/data/system/packages.list文件

        scheduleWritePackageListLocked(userId);

    // 保存默认浏览器应用，并更新/data/system/users/0/package-restrictions.xml文件

        applyFactoryDefaultBrowserLPw(userId);

    // 主要域名验证

        primeDomainVerificationsLPw(userId);

    }

}

/\*\* Map from package name to settings 每个包名对应一个PackageSetting\*/

final ArrayMap<String, PackageSetting> mPackages = new ArrayMap<>();

// 把所有已安装的系统应用数据拷贝到新建用户对应目录下

void createNewUserLI(@NonNull PackageManagerService service, @NonNull Installer installer,

        int userHandle) {

    String[] volumeUuids;

    String[] names;

    int[] appIds;

    String[] seinfos;

    int[] targetSdkVersions;

    int packagesCount;

    synchronized (mPackages) {

    // 从map中获取出所有的settings

        Collection<PackageSetting> packages = mPackages.values();

        packagesCount = packages.size();

        volumeUuids = new String[packagesCount];

        names = new String[packagesCount];

        appIds = new int[packagesCount];

        seinfos = new String[packagesCount];

        targetSdkVersions = new int[packagesCount];

        Iterator<PackageSetting> packagesIterator = packages.iterator();

    // 遍历所有的PackageSetting

        for (int i = 0; i < packagesCount; i++) {

            PackageSetting ps = packagesIterator.next();

            if (ps.pkg == null || ps.pkg.applicationInfo == null) {

                continue;

            }

            // Only system apps are initially installed.初始化时只安装系统应用

            /\*\* M: [Operator] Operator package should also be installed @{ \*/

            boolean curInstalledStatus = ps.isSystem()

                                || (ps.pkgFlagsEx & ApplicationInfo.FLAG\_EX\_OPERATOR) != 0;

        // 设置每一个应用在新创建用户下的安装状态，系统应用为true

            ps.setInstalled(curInstalledStatus, userHandle);

            /\*\* @} \*/

            // Need to create a data directory for all apps under this user. Accumulate all

            // required args and call the installer after mPackages lock has been released

            volumeUuids[i] = ps.volumeUuid;

            names[i] = ps.name;

            appIds[i] = ps.appId;

            seinfos[i] = ps.pkg.applicationInfo.seinfo;

            targetSdkVersions[i] = ps.pkg.applicationInfo.targetSdkVersion;

        }

    }

    for (int i = 0; i < packagesCount; i++) {

        if (names[i] == null) {

            continue;

        }

        // TODO: triage flags!

        final int flags = StorageManager.FLAG\_STORAGE\_CE | StorageManager.FLAG\_STORAGE\_DE;

        try {

        // 在新建用户目录（/data/user/0/）下创建每个应用的数据目录（包名命名的文件夹）

            installer.createAppData(volumeUuids[i], names[i], userHandle, flags, appIds[i],

                    seinfos[i], targetSdkVersions[i]);

        } catch (InstallerException e) {

            Slog.w(TAG, "Failed to prepare app data", e);

        }

    }

    synchronized (mPackages) {

    // 解析"etc/preferred-apps"目录下所有XML文件，XML文件中保存的是设备使用者指定的响应某个Intent

    // 的最合适的组件信息

        applyDefaultPreferredAppsLPw(service, userHandle);

    }

}

### removeUser

.删除用户

[java] view plain copy

/\*\*

 \* Removes a user and all data directories created for that user. This method should be called

 \* after the user's processes have been terminated.

 \* @param userHandle the user's id

 \*/

@Override

public boolean removeUser(int userHandle) {

    // 检查调用者是否有删除用户的权限

    checkManageOrCreateUsersPermission("Only the system can remove users");

    if (getUserRestrictions(UserHandle.getCallingUserId()).getBoolean(

            UserManager.DISALLOW\_REMOVE\_USER, false)) {

        Log.w(LOG\_TAG, "Cannot remove user. DISALLOW\_REMOVE\_USER is enabled.");

        return false;

    }

    long ident = Binder.clearCallingIdentity();

    try {

        final UserData userData;

        int currentUser = ActivityManager.getCurrentUser();

        if (currentUser == userHandle) {

            Log.w(LOG\_TAG, "Current user cannot be removed");

            return false;

        }

        synchronized (mPackagesLock) {

            synchronized (mUsersLock) {

                userData = mUsers.get(userHandle);

                if (userHandle == 0 || userData == null || mRemovingUserIds.get(userHandle)) {

                    return false;

                }

                // We remember deleted user IDs to prevent them from being

                // reused during the current boot; they can still be reused

                // after a reboot.保存要删除的userId，防止重复删除

                mRemovingUserIds.put(userHandle, true);

            }

            try {

                mAppOpsService.removeUser(userHandle);

            } catch (RemoteException e) {

                Log.w(LOG\_TAG, "Unable to notify AppOpsService of removing user", e);

            }

            // Set this to a partially created user, so that the user will be purged

            // on next startup, in case the runtime stops now before stopping and

            // removing the user completely.

        // 删除用户并没有删除相关用户文件，只是把partial变量修改为true，

        // 开机后如果该变量还是true会删除相关文件

            userData.info.partial = true;

            // Mark it as disabled, so that it isn't returned any more when

            // profiles are queried.

            userData.info.flags |= UserInfo.FLAG\_DISABLED;

        // 更新/data/system/users/${id}.xml文件

            writeUserLP(userData);

        }

        if (userData.info.profileGroupId != UserInfo.NO\_PROFILE\_GROUP\_ID

                && userData.info.isManagedProfile()) {

            // Send broadcast to notify system that the user removed was a

            // managed user.发送删除用户的广播

            sendProfileRemovedBroadcast(userData.info.profileGroupId, userData.info.id);

        }

        if (DBG) Slog.i(LOG\_TAG, "Stopping user " + userHandle);

        int res;

        try {

        // 停止正在运行的用户

            res = ActivityManagerNative.getDefault().stopUser(userHandle, /\* force= \*/ true,

            new IStopUserCallback.Stub() {

                        @Override

                        public void userStopped(int userId) {

                // 删除用户相关应用信息

                            finishRemoveUser(userId);

                        }

                        @Override

                        public void userStopAborted(int userId) {

                        }

                    });

        } catch (RemoteException e) {

            return false;

        }

        return res == ActivityManager.USER\_OP\_SUCCESS;

    } finally {

        Binder.restoreCallingIdentity(ident);

    }

}

---------------------

作者：kaijiehui

来源：CSDN

原文：https://blog.csdn.net/u011341111/article/details/79458128

版权声明：本文为博主原创文章，转载请附上博文链接！

### .多用户管理

UserManagerService主要管理用户的账号信息，运行中的用户管理由ActivityManagerService来负责。

用户的状态有5种，定义在UserState类中：

[java] view plain copy

public final class UserState {

    // User is first coming up.启动中

    public final static int STATE\_BOOTING = 0;

    // User is in the locked state.锁定

    public final static int STATE\_RUNNING\_LOCKED = 1;

    // User is in the unlocking state.未锁定

    public final static int STATE\_RUNNING\_UNLOCKING = 2;

    // User is in the running state.运行中

    public final static int STATE\_RUNNING\_UNLOCKED = 3;

    // User is in the initial process of being stopped.停止的初始过程中

    public final static int STATE\_STOPPING = 4;

    // User is in the final phase of stopping, sending Intent.ACTION\_SHUTDOWN.停止的最后阶段

    public final static int STATE\_SHUTDOWN = 5;

    ...

}

@Override

public boolean switchUser(final int targetUserId) {

    // 检查调用者是否有切换用户的权限

    enforceShellRestriction(UserManager.DISALLOW\_DEBUGGING\_FEATURES, targetUserId);

    UserInfo currentUserInfo;

    UserInfo targetUserInfo;

    synchronized (this) {

    // 获取当用用户的相关信息

        int currentUserId = mUserController.getCurrentUserIdLocked();

        currentUserInfo = mUserController.getUserInfo(currentUserId);

    // 获取切换目标用户的相关信息

        targetUserInfo = mUserController.getUserInfo(targetUserId);

        if (targetUserInfo == null) {

            Slog.w(TAG, "No user info for user #" + targetUserId);

            return false;

        }

    // 如果目标用户不支持切换，则返回

        if (!targetUserInfo.supportsSwitchTo()) {

            Slog.w(TAG, "Cannot switch to User #" + targetUserId + ": not supported");

            return false;

        }

    // 如果目标用户是另一个用户的profile，则返回

        if (targetUserInfo.isManagedProfile()) {

            Slog.w(TAG, "Cannot switch to User #" + targetUserId + ": not a full user");

            return false;

        }

        mUserController.setTargetUserIdLocked(targetUserId);

    }

    // 发送切换用户的消息

    Pair<UserInfo, UserInfo> userNames = new Pair<>(currentUserInfo, targetUserInfo);

    mUiHandler.removeMessages(START\_USER\_SWITCH\_UI\_MSG);

    mUiHandler.sendMessage(mUiHandler.obtainMessage(START\_USER\_SWITCH\_UI\_MSG, userNames));

    return true;

}

final class UiHandler extends Handler {

    public UiHandler() {

        super(com.android.server.UiThread.get().getLooper(), null, true);

    }

    @Override

    public void handleMessage(Message msg) {

        switch (msg.what) {

            ...

            case START\_USER\_SWITCH\_UI\_MSG: {

                mUserController.showUserSwitchDialog((Pair<UserInfo, UserInfo>) msg.obj);

                break;

            }

            ...

        }

    }

}

void showUserSwitchDialog(Pair<UserInfo, UserInfo> fromToUserPair) {

    // The dialog will show and then initiate the user switch by calling startUserInForeground

    Dialog d = new UserSwitchingDialog(mService, mService.mContext, fromToUserPair.first,

            fromToUserPair.second, true /\* above system \*/);

    d.show();

}

@Override

public void show() {

    // Slog.v(TAG, "show called");

    super.show();

    final View decorView = getWindow().getDecorView();

    if (decorView != null) {

        decorView.getViewTreeObserver().addOnWindowShownListener(this);

    }

    // Add a timeout as a safeguard, in case a race in screen on/off causes the window

    // callback to never come.

    mHandler.sendMessageDelayed(mHandler.obtainMessage(MSG\_START\_USER),

            WINDOW\_SHOWN\_TIMEOUT\_MS);

}

private final Handler mHandler = new Handler() {

    @Override

    public void handleMessage(Message msg) {

        switch (msg.what) {

            case MSG\_START\_USER:

        // 调用startUser()方法

                startUser();

                break;

        }

    }

};

void startUser() {

    synchronized (this) {

        if (!mStartedUser) {

        // 调用startUserInForeground方法

            mService.mUserController.startUserInForeground(mUserId, this);

            mStartedUser = true;

            final View decorView = getWindow().getDecorView();

            if (decorView != null) {

                decorView.getViewTreeObserver().removeOnWindowShownListener(this);

            }

            mHandler.removeMessages(MSG\_START\_USER);

        }

    }

}

/\*\*

 \* Start user, if its not already running, and bring it to foreground.

 \* 开启用户，如果用户没有在运行，则开启它

 \*/

boolean startUserInForeground(final int userId, Dialog dlg) {

    boolean result = startUser(userId, /\* foreground \*/ true);

    dlg.dismiss();

    return result;

}

/\*\*

 \* Start user, if its not already running.

 \* <p>The user will be brought to the foreground, if {@code foreground} parameter is set.

 \* When starting the user, multiple intents will be broadcast in the following order:</p>

 \* <ul>

 \*     <li>{@link Intent#ACTION\_USER\_STARTED} - sent to registered receivers of the new user

 \*     <li>{@link Intent#ACTION\_USER\_BACKGROUND} - sent to registered receivers of the outgoing

 \*     user and all profiles of this user. Sent only if {@code foreground} parameter is true

 \*     <li>{@link Intent#ACTION\_USER\_FOREGROUND} - sent to registered receivers of the new

 \*     user and all profiles of this user. Sent only if {@code foreground} parameter is true

 \*     <li>{@link Intent#ACTION\_USER\_SWITCHED} - sent to registered receivers of the new user.

 \*     Sent only if {@code foreground} parameter is true

 \*     <li>{@link Intent#ACTION\_USER\_STARTING} - ordered broadcast sent to registered receivers

 \*     of the new fg user

 \*     <li>{@link Intent#ACTION\_LOCKED\_BOOT\_COMPLETED} - ordered broadcast sent to receivers of

 \*     the new user

 \*     <li>{@link Intent#ACTION\_USER\_UNLOCKED} - sent to registered receivers of the new user

 \*     <li>{@link Intent#ACTION\_PRE\_BOOT\_COMPLETED} - ordered broadcast sent to receivers of the

 \*     new user. Sent only when the user is booting after a system update.

 \*     <li>{@link Intent#ACTION\_USER\_INITIALIZE} - ordered broadcast sent to receivers of the

 \*     new user. Sent only the first time a user is starting.

 \*     <li>{@link Intent#ACTION\_BOOT\_COMPLETED} - ordered broadcast sent to receivers of the new

 \*     user. Indicates that the user has finished booting.

 \* </ul>

 \*

 \* @param userId ID of the user to start

 \* @param foreground true if user should be brought to the foreground

 \* @return true if the user has been successfully started

 \*/

boolean startUser(final int userId, final boolean foreground) {

    // 检查调用者权限

    if (mService.checkCallingPermission(INTERACT\_ACROSS\_USERS\_FULL)

            != PackageManager.PERMISSION\_GRANTED) {

        String msg = "Permission Denial: switchUser() from pid="

                + Binder.getCallingPid()

                + ", uid=" + Binder.getCallingUid()

                + " requires " + INTERACT\_ACROSS\_USERS\_FULL;

        Slog.w(TAG, msg);

        throw new SecurityException(msg);

    }

    Slog.i(TAG, "Starting userid:" + userId + " fg:" + foreground);

    final long ident = Binder.clearCallingIdentity();

    try {

        synchronized (mService) {

            final int oldUserId = mCurrentUserId;

        // 如果要开启的用户已经存在，则直接返回true

            if (oldUserId == userId) {

                return true;

            }

            mService.mStackSupervisor.setLockTaskModeLocked(null,

                    ActivityManager.LOCK\_TASK\_MODE\_NONE, "startUser", false);

        // 获取目标用户的相关信息

            final UserInfo userInfo = getUserInfo(userId);

            if (userInfo == null) {

                Slog.w(TAG, "No user info for user #" + userId);

                return false;

            }

            if (foreground && userInfo.isManagedProfile()) {

                Slog.w(TAG, "Cannot switch to User #" + userId + ": not a full user");

                return false;

            }

        // 如果要把用户切到前台，则播放动画

            if (foreground) {

                mService.mWindowManager.startFreezingScreen(

                        R.anim.screen\_user\_exit, R.anim.screen\_user\_enter);

            }

            boolean needStart = false;

            // If the user we are switching to is not currently started, then

            // we need to start it now.如果目标用户不存在，则修改用户状态为正在开启

        // UserState的状态默认值是STATE\_BOOTING

            if (mStartedUsers.get(userId) == null) {

                UserState userState = new UserState(UserHandle.of(userId));

                mStartedUsers.put(userId, userState);

                getUserManagerInternal().setUserState(userId, userState.state);

        // 根据用户状态更新已经开启的用户列表

                updateStartedUserArrayLocked();

                needStart = true;

            }

            final UserState uss = mStartedUsers.get(userId);

            final Integer userIdInt = userId;

            mUserLru.remove(userIdInt);

        // 调整用户在mUserLru中的位置，当前用户位于末尾

            mUserLru.add(userIdInt);

            if (foreground) {

        // 修改当前用户的Id

                mCurrentUserId = userId;

        // 更新用户配置信息

                mService.updateUserConfigurationLocked();

                mTargetUserId = UserHandle.USER\_NULL; // reset, mCurrentUserId has caught up

        // 更新与当前用户相关的用户列表

                updateCurrentProfileIdsLocked();

                mService.mWindowManager.setCurrentUser(userId, mCurrentProfileIds);

                // Once the internal notion of the active user has switched, we lock the device

                // with the option to show the user switcher on the keyguard.

                mService.mWindowManager.lockNow(null);

            } else {

                final Integer currentUserIdInt = mCurrentUserId;

        // 更新与当前用户相关的用户列表

                updateCurrentProfileIdsLocked();

                mService.mWindowManager.setCurrentProfileIds(mCurrentProfileIds);

                mUserLru.remove(currentUserIdInt);

                mUserLru.add(currentUserIdInt);

            }

            // Make sure user is in the started state.  If it is currently

            // stopping, we need to knock that off.确保用户处于启动状态，如果处于

        // 停止的初始阶段，则中止它。如果已经发送过停止运行的广播，则重新设置用户的状态

            if (uss.state == UserState.STATE\_STOPPING) {

                // If we are stopping, we haven't sent ACTION\_SHUTDOWN,

                // so we can just fairly silently bring the user back from

                // the almost-dead.

                uss.setState(uss.lastState);

                getUserManagerInternal().setUserState(userId, uss.state);

        // 根据用户状态更新已经开启的用户列表

                updateStartedUserArrayLocked();

                needStart = true;

            } else if (uss.state == UserState.STATE\_SHUTDOWN) {

                // This means ACTION\_SHUTDOWN has been sent, so we will

                // need to treat this as a new boot of the user.

                uss.setState(UserState.STATE\_BOOTING);

                getUserManagerInternal().setUserState(userId, uss.state);

        // 根据用户状态更新已经开启的用户列表

                updateStartedUserArrayLocked();

                needStart = true;

            }

            if (uss.state == UserState.STATE\_BOOTING) {

                // Give user manager a chance to propagate user restrictions

                // to other services and prepare app storage

        // 在用户启动之前，先准备相关用户的限制及存储

                getUserManager().onBeforeStartUser(userId);

                // Booting up a new user, need to tell system services about it.

                // Note that this is on the same handler as scheduling of broadcasts,

                // which is important because it needs to go first.

                mHandler.sendMessage(mHandler.obtainMessage(SYSTEM\_USER\_START\_MSG, userId, 0));

            }

            if (foreground) {

        // 发送相关消息

                mHandler.sendMessage(mHandler.obtainMessage(SYSTEM\_USER\_CURRENT\_MSG, userId,

                        oldUserId));

                mHandler.removeMessages(REPORT\_USER\_SWITCH\_MSG);

                mHandler.removeMessages(USER\_SWITCH\_TIMEOUT\_MSG);

                mHandler.sendMessage(mHandler.obtainMessage(REPORT\_USER\_SWITCH\_MSG,

                        oldUserId, userId, uss));

                mHandler.sendMessageDelayed(mHandler.obtainMessage(USER\_SWITCH\_TIMEOUT\_MSG,

                        oldUserId, userId, uss), USER\_SWITCH\_TIMEOUT);

            }

            if (needStart) {

                // Send USER\_STARTED broadcast 如果需要开启用户，则发送相应广播

        // 用户切换牵扯到很多模块，如壁纸管理、输入法、账号管理等，都需要收到通知

                Intent intent = new Intent(Intent.ACTION\_USER\_STARTED);

                intent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY

                        | Intent.FLAG\_RECEIVER\_FOREGROUND);

                intent.putExtra(Intent.EXTRA\_USER\_HANDLE, userId);

                mService.broadcastIntentLocked(null, null, intent,

                        null, null, 0, null, null, null, AppOpsManager.OP\_NONE,

                        null, false, false, MY\_PID, SYSTEM\_UID, userId);

            }

            if (foreground) {

        // 把开启的用户设为前台用户

                moveUserToForegroundLocked(uss, oldUserId, userId);

            } else {

        // 用户启动结束，则切换用户到STATE\_RUNNING\_LOCKED状态

                mService.mUserController.finishUserBoot(uss);

            }

            if (needStart) {

        // 如果需要开启用户，则发送相应广播

                Intent intent = new Intent(Intent.ACTION\_USER\_STARTING);

                intent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY);

                intent.putExtra(Intent.EXTRA\_USER\_HANDLE, userId);

                mService.broadcastIntentLocked(null, null, intent,

                        null, new IIntentReceiver.Stub() {

                            @Override

                            public void performReceive(Intent intent, int resultCode,

                                    String data, Bundle extras, boolean ordered, boolean sticky,

                                    int sendingUser) throws RemoteException {

                            }

                        }, 0, null, null,

                        new String[] {INTERACT\_ACROSS\_USERS}, AppOpsManager.OP\_NONE,

                        null, true, false, MY\_PID, SYSTEM\_UID, UserHandle.USER\_ALL);

            }

        }

    } finally {

        Binder.restoreCallingIdentity(ident);

    }

    return true;

}

// 根据用户状态更新已经开启的用户列表mStartedUserArray

private void updateStartedUserArrayLocked() {

    int num = 0;

    for (int i = 0; i < mStartedUsers.size(); i++) {

        UserState uss = mStartedUsers.valueAt(i);

        // This list does not include stopping users.

        if (uss.state != UserState.STATE\_STOPPING

                && uss.state != UserState.STATE\_SHUTDOWN) {

            num++;

        }

    }

    mStartedUserArray = new int[num];

    num = 0;

    for (int i = 0; i < mStartedUsers.size(); i++) {

        UserState uss = mStartedUsers.valueAt(i);

        if (uss.state != UserState.STATE\_STOPPING

                && uss.state != UserState.STATE\_SHUTDOWN) {

            mStartedUserArray[num++] = mStartedUsers.keyAt(i);

        }

    }

}

发送的msg消息是在ActivityManagerService中处理的：

final class UiHandler extends Handler {

    public UiHandler() {

        super(com.android.server.UiThread.get().getLooper(), null, true);

    }

    @Override

    public void handleMessage(Message msg) {

        switch (msg.what) {

        ...

        case SYSTEM\_USER\_START\_MSG: {

            mBatteryStatsService.noteEvent(BatteryStats.HistoryItem.EVENT\_USER\_RUNNING\_START,

                    Integer.toString(msg.arg1), msg.arg1);

        // 新建用户时调用

            mSystemServiceManager.startUser(msg.arg1);

            break;

        }

        case SYSTEM\_USER\_CURRENT\_MSG: {

            mBatteryStatsService.noteEvent(

                    BatteryStats.HistoryItem.EVENT\_USER\_FOREGROUND\_FINISH,

                    Integer.toString(msg.arg2), msg.arg2);

            mBatteryStatsService.noteEvent(

                    BatteryStats.HistoryItem.EVENT\_USER\_FOREGROUND\_START,

                    Integer.toString(msg.arg1), msg.arg1);

        // 切换用户时调用

            mSystemServiceManager.switchUser(msg.arg1);

            break;

        }

        case REPORT\_USER\_SWITCH\_MSG: {

            mUserController.dispatchUserSwitch((UserState) msg.obj, msg.arg1, msg.arg2);

            break;

        }

        case CONTINUE\_USER\_SWITCH\_MSG: {

            mUserController.continueUserSwitch((UserState) msg.obj, msg.arg1, msg.arg2);

            break;

        }

        case USER\_SWITCH\_TIMEOUT\_MSG: {

            mUserController.timeoutUserSwitch((UserState) msg.obj, msg.arg1, msg.arg2);

            break;

        }

        case REPORT\_USER\_SWITCH\_COMPLETE\_MSG: {

            mUserController.dispatchUserSwitchComplete(msg.arg1);

            break;

        }

        ...

    }

};

// 该方法主要是调用mUserSwitchObservers列表中的IUserSwitchObserver对象的onUserSwitching方法

// 如果想知道用户切换，可以调用AMS的registerUserSwitchObserver()方法来注册一个观察者对象

void dispatchUserSwitch(final UserState uss, final int oldUserId, final int newUserId) {

    Slog.d(TAG, "Dispatch onUserSwitching oldUser #" + oldUserId + " newUser #" + newUserId);

    // 获取所有注册回调方法的总数

    final int observerCount = mUserSwitchObservers.beginBroadcast();

    if (observerCount > 0) {

        final IRemoteCallback callback = new IRemoteCallback.Stub() {

            int mCount = 0;

            @Override

            public void sendResult(Bundle data) throws RemoteException {

                synchronized (mService) {

                    if (mCurUserSwitchCallback == this) {

            // 收到一条回调，就加一

                        mCount++;

            // 所有注册的回调方法都执行了，发送继续处理的消息

                        if (mCount == observerCount) {

                            sendContinueUserSwitchLocked(uss, oldUserId, newUserId);

                        }

                    }

                }

            }

        };

        synchronized (mService) {

            uss.switching = true;

            mCurUserSwitchCallback = callback;

        }

    // 遍历调用所有注册回调对象的onUserSwitching方法

        for (int i = 0; i < observerCount; i++) {

            try {

                mUserSwitchObservers.getBroadcastItem(i).onUserSwitching(

                        newUserId, callback);

            } catch (RemoteException e) {

            }

        }

    } else {

        synchronized (mService) {

        // 如果没有注册回调方法的，直接调用继续执行用户切换的方法

            sendContinueUserSwitchLocked(uss, oldUserId, newUserId);

        }

    }

    mUserSwitchObservers.finishBroadcast();

}

void sendContinueUserSwitchLocked(UserState uss, int oldUserId, int newUserId) {

    mCurUserSwitchCallback = null;

    mHandler.removeMessages(USER\_SWITCH\_TIMEOUT\_MSG);

    mHandler.sendMessage(mHandler.obtainMessage(ActivityManagerService.CONTINUE\_USER\_SWITCH\_MSG,

            oldUserId, newUserId, uss));

}

void continueUserSwitch(UserState uss, int oldUserId, int newUserId) {

    Slog.d(TAG, "Continue user switch oldUser #" + oldUserId + ", newUser #" + newUserId);

    synchronized (mService) {

        mService.mWindowManager.stopFreezingScreen();

    }

    uss.switching = false;

    // 发送完成切换用户的消息

    mHandler.removeMessages(REPORT\_USER\_SWITCH\_COMPLETE\_MSG);

    mHandler.sendMessage(mHandler.obtainMessage(REPORT\_USER\_SWITCH\_COMPLETE\_MSG,

            newUserId, 0));

    // 停止切换到后台的Guest或临时用户

    stopGuestOrEphemeralUserIfBackground();

    // 强制停止后台用户

    stopBackgroundUsersIfEnforced(oldUserId);

}

/\*\* Called on handler thread \*/

void dispatchUserSwitchComplete(int userId) {

    final int observerCount = mUserSwitchObservers.beginBroadcast();

    for (int i = 0; i < observerCount; i++) {

        try {

        // 遍历调用所有观察者的onUserSwitchComplete方法

            mUserSwitchObservers.getBroadcastItem(i).onUserSwitchComplete(userId);

        } catch (RemoteException e) {

        }

    }

    mUserSwitchObservers.finishBroadcast();

}

/\*\*

 \* Stops the guest or ephemeral user if it has gone to the background.

 \* 停止切换到后台的Guest或临时用户

 \*/

private void stopGuestOrEphemeralUserIfBackground() {

    synchronized (mService) {

        final int num = mUserLru.size();

        for (int i = 0; i < num; i++) {

            Integer oldUserId = mUserLru.get(i);

            UserState oldUss = mStartedUsers.get(oldUserId);

            if (oldUserId == UserHandle.USER\_SYSTEM || oldUserId == mCurrentUserId

                    || oldUss.state == UserState.STATE\_STOPPING

                    || oldUss.state == UserState.STATE\_SHUTDOWN) {

                continue;

            }

            UserInfo userInfo = getUserInfo(oldUserId);

            if (userInfo.isEphemeral()) {

                LocalServices.getService(UserManagerInternal.class)

                        .onEphemeralUserStop(oldUserId);

            }

            if (userInfo.isGuest() || userInfo.isEphemeral()) {

                // This is a user to be stopped.

                stopUsersLocked(oldUserId, true, null);

                break;

            }

        }

    }

}

// 强制停止后台用户

private void stopBackgroundUsersIfEnforced(int oldUserId) {

    // Never stop system user

    if (oldUserId == UserHandle.USER\_SYSTEM) {

        return;

    }

    // For now, only check for user restriction. Additional checks can be added here

    boolean disallowRunInBg = hasUserRestriction(UserManager.DISALLOW\_RUN\_IN\_BACKGROUND,

            oldUserId);

    if (!disallowRunInBg) {

        return;

    }

    synchronized (mService) {

        if (DEBUG\_MU) Slog.i(TAG, "stopBackgroundUsersIfEnforced stopping " + oldUserId

                + " and related users");

        stopUsersLocked(oldUserId, false, null);

    }

}

void timeoutUserSwitch(UserState uss, int oldUserId, int newUserId) {

    synchronized (mService) {

        /// M: Change Slog.wtf to Slog.w to avoid having WTF easily after adding new user

        Slog.w(TAG, "User switch timeout: from " + oldUserId + " to " + newUserId);

        sendContinueUserSwitchLocked(uss, oldUserId, newUserId);

    }

}

Activity进入Idle状态时会调用activityIdleInternalLocked方法，该方法中会修改用户的状态到STATE\_RUNNING\_LOCKED状

---------------------

作者：kaijiehui

来源：CSDN

原文：https://blog.csdn.net/u011341111/article/details/79458128

版权声明：本文为博主原创文章，转载请附上博文链接！

# 三. userId的用处

## 3.1 客户端(ContextImpl)

**Service**

public boolean bindService(Intent service, ServiceConnection conn, int flags) {

warnIfCallingFromSystemProcess();

//此处Process.myUserHandle获取的是获取当前进程uid所属的userId来创建UserHandle对象。

return bindServiceCommon(service, conn, flags, Process.myUserHandle());

}

public ComponentName startService(Intent service) {

warnIfCallingFromSystemProcess();

//mUser也是通过Process.myUserHandle()方法获取

return startServiceCommon(service, mUser);

}

**Broadcast**

public void sendBroadcast(Intent intent) {

warnIfCallingFromSystemProcess();

String resolvedType = intent.resolveTypeIfNeeded(getContentResolver());

try {

intent.prepareToLeaveProcess();

ActivityManagerNative.getDefault().broadcastIntent(

mMainThread.getApplicationThread(), intent, resolvedType, null,

Activity.RESULT\_OK, null, null, null, AppOpsManager.OP\_NONE, null, false, false,

getUserId());

} catch (RemoteException e) {

throw new RuntimeException("Failure from system", e);

}

}

无论是广播，服务，Activity，在没有指定UserId的情况下，都采用默认的当前进程uid所对应的userId。

## 服务端(AMS)

广播，服务，Activity启动的过程，经过Binder进入system\_server进程，则都会会采用如下方法将userId进行转换：

int handleIncomingUser(int callingPid, int callingUid, int userId, boolean allowAll, int allowMode, String name, String callerPackage) {

final int callingUserId = UserHandle.getUserId(callingUid);

if (callingUserId == userId) {

return userId;

}

// userId转换【见下文】

int targetUserId = unsafeConvertIncomingUser(userId);

if (callingUid != 0 && callingUid != Process.SYSTEM\_UID) {

final boolean allow;

... //对于非system uid，则会进行各种权限检查。

}

if (!allowAll && targetUserId < 0) {

throw new IllegalArgumentException(...);

}

//检查shell权限

if (callingUid == Process.SHELL\_UID && targetUserId >= UserHandle.USER\_OWNER) {

if (mUserManager.hasUserRestriction(UserManager.DISALLOW\_DEBUGGING\_FEATURES,

targetUserId)) {

throw new SecurityException(...);

}

}

return targetUserId;

}

int unsafeConvertIncomingUser(int userId) {

return (userId == UserHandle.USER\_CURRENT || userId == UserHandle.USER\_CURRENT\_OR\_SELF)

? mCurrentUserId : userId;

}

该方法主要功能：

1. 当调用者userId等于userId时，则直接返回该userId；否则往下执行；
2. 当USER\_CURRENT或USER\_CURRENT\_OR\_SELF类型的userId，则返回mCurrentUserId；否则继续采用userId；
3. 对于非system uid，则会进行各种权限检查。

## 3.3 调试相关

dumpsys user可查看用户情况.

# 参考

多用户管理UserManager

<http://gityuan.com/2016/11/20/user_manager/>

Android多用户之UserManagerService源码分析

<https://blog.csdn.net/u011341111/article/details/79458128>