前言

最近接到Android上面开发应用双开的任务，市面上应用双开已经是很成熟的产品了。经过前期的调研，基本上实现原理都是在Android多用户的基础上面开发。所以花点时间熟悉UserManager在所难免，查看网上已经有多大牛把UserManager的流程梳理的很清楚了，我这借鉴何整理了一下，作为自己学习总结。

多用户的核心是UserManagerService，顾名思义整个用户是由它来管理，以userManagerService为核心，围绕着，UserManager、UserController、UserInfo几个关键类。

用户基本操作

关于多用户一共有3大基本操作，创建、切换和删除。

- 1.创建用户的方法

```
UserManager mUserManager = (UserManager) context.getSystemService(Context.USER_SERVICE);

UserInfo  newUserInfo = mUserManager.createUser(“用户名”, 0);
```

首先通过framework的UserManager调用createUser方法，而UserManager如同所有的manager一样是中转到service来进行实际操作。

service中创建用户源码如下，其返回值是一个UserInfo类，用于存储创建出的用户的信息

```
public UserInfo createUser(String name, int flags) {

   checkManageOrCreateUsersPermission(flags);

   return createUserInternal(name, flags, UserHandle.USER_NULL);

}
```

传入值有两个，第一个是name表示该用户的名称，flags则是表示该用户的类型，flags定义的代码在UserInfo这个类里面。

创建的用户，常用有四类：1普通用户 2访客用户 3工作用户 4其他：比如各手机自定义用户 如 隐私空间 克隆模式等等。 普通用户和访客用户，直接通过设置或状态栏就可以创建。工作用户是基于google开发的AFW（android for work）其他用户，则由各个程序员自行决定怎么创建

- 2：删除用户的方法

```
UserManager mUserManager = (UserManager) context.getSystemService(Context.USER_SERVICE);

UserInfonewUserInfo = mUserManager.removeuser(userId);
```

删除比创建用户传入的参数少，创建需要定义flag，而删除则只需指定删除的用户编号就可以。

用户编号 userId，默认从0开始，然后是10,11,12依次类推。从10开始是因为在userManagerService定义了（MIN_USER_ID = 10）多用户以10开始。

一般获取编号，主要是在createUser的时候，返回的UserInfo里面就存有UserId，把这个UserId保存好就可以轻易删掉目标用户，其次是通过ActicityManagerService的getCurrentUser获取当前Id。

- 3.切换用户的方法

```
ActivityManager.getService().switchUser(userId);
```

传入参数userId，则可以向ActicivityManagerService申请切换。该逻辑的实质是在UserController里面通过startUser方法启动目标用户，来完成切换。

获取userId的方法有很多，之前提到了createUser的时候自己保存，以及通过ActivityManagerService获取，此外，framework会主动把用户信息存在/data/system/users/userlist.xml 这个xml文件处，保证在重启手机之后，还能够查到之前创建过的用户。但是，最普遍适用的还是通过UserManagerService通过getUsers或者getUserids来获取全部的用户信息，然后自己加以筛选和判断。

以上就是关于 多用户的基本用法，供以第三方应用开发的时候查阅。

创建用户类型

上文提到创建用户的方法：

```
UserManager mUserManager = (UserManager) context.getSystemService(Context.USER_SERVICE);

UserInfo  newUserInfo = mUserManager.createUser(“用户名”, 0);
```

入口代码是UserManagerService中的createUser。

```
public UserInfo createUser(String name, int flags) {

  checkManageOrCreateUsersPermission(flags);

  return createUserInternal(name, flags, UserHandle.USER_NULL);

}
```

而最核心的方法是createUserInternalUnchecked。调用流程实际从UserManager的对外接口代码开始。APP创建用户，包括系统设置，必须通过UserManager来创建

上文提到了一个示例，而实际上在UserManager中创建用户的代码，一共有六个接口。

- 1.标准的用户创建

```
 public UserInfo createUser(String name, int flags) {

        UserInfo user = null;

        try {

            // 核心代码就调用service的createUser。

            user = mService.createUser(name, flags);


            if (user != null && !user.isAdmin() && !user.isDemo()) {

                mService.setUserRestriction(DISALLOW_SMS, true, user.id);

                mService.setUserRestriction(DISALLOW_OUTGOING_CALLS, true, user.id);

            }

        } catch (RemoteException re) {
```

```
            throw re.rethrowFromSystemServer();
        }
        return user;
    }
```

- 2.创建访客用户

```
public UserInfo createGuest(Context context, String name) {
      UserInfo guest = null;
      try {
          //同样适用了标准创建方式，只是帮助大家写好了flag，表示创建的是访客。
          guest = mService.createUser(name, UserInfo.FLAG_GUEST);
          if (guest != null) {
              Settings.Secure.putStringForUser(context.getContentResolver(),
                      Settings.Secure.SKIP_FIRST_USE_HINTS, "1", guest.id);
          }
      } catch (RemoteException re) {
          throw re.rethrowFromSystemServer();
      }
      return guest;
}
```

接下来出现了一个新的创建用户的方法，叫做创建用户的子用户。当创建子用户之后，我总结了一些概念来帮助理解子用户：

- a.用户和其子用户成对出现，user是正常用户，profile是子用户
- b.profile必须依附于user生存，也就是一个子用户一定对应一个主用户。
- c.当一个用户创建了子用户之后，该用户和子用户直接就建立了连接。我们这时候可以称这个用户为主用户。也就是主用户和子用户的成对出现的。
- d.感觉有人读到这里有点绕，这是因为代码里面User的概念太宽泛：我们通过UserId来区分每个用户，而仅仅拥有UserId的情况下，我们无法知道这个UserId对应的
  User是普通用户，还是具备子用户的主用户，或者就是子用户。在代码里面，所用用户无论子用户，主用户，乃至工作用户，统统都是User。
- e.对于一个用户我们通过getprofiles方法，可以获取该用户及其关联用户的userId。 也就是普通用户会获取一个user，而无论是主用户还是子用户，
  都会获取出主用户+子用户的user集合。
- f.对于一个子用户而言，它的parentUser是主用户。对于主用户而言，它的profile是子用户。

总结在创建的时候，我们可以知道我们是创建子用户。但是对于某个用户而言，我们需要对其参数进行判断才能知道其为哪种类型的用户。

在创建子用户的时候，需要传入正常用户的信息。

- 3.创建子用户

```
public UserInfo createProfileForUser(String name, int flags, @UserIdInt int userHandle) {
        //创建指定user的子用户，且子用户的应用均可正常使用
        return createProfileForUser(name, flags, userHandle, null);
    }
```

- 4.创建子用户，且子用户部分应用不可使用

```
public UserInfo createProfileForUser(String name, int flags, @UserIdInt int userHandle,
            String[] disallowedPackages) {
        try {
          //创建指定user的子用户，且子用户的部分应用禁止使用
            return mService.createProfileForUser(name, flags, userHandle, disallowedPackages);
        } catch (RemoteException re) {
            throw re.rethrowFromSystemServer();
        }
    }
```

- 5.在不允许创建子用户的情况下强行创建子用户，且子用户部分应用不可使用

```
    public UserInfo createProfileForUserEvenWhenDisallowed(String name, int flags,
            @UserIdInt int userHandle, String[] disallowedPackages) {
        try {
            return mService.createProfileForUserEvenWhenDisallowed(name, flags, userHandle,
                    disallowedPackages);
        } catch (RemoteException re) {
            throw re.rethrowFromSystemServer();
        }
    }
```

- 6.创建严格受管控的子用户

```
public UserInfo createRestrictedProfile(String name) {
        try {
            UserHandle parentUserHandle = Process.myUserHandle();
            UserInfo user = mService.createRestrictedProfile(name,
                    parentUserHandle.getIdentifier());
            if (user != null) {
```

```
                AccountManager.get(mContext).addSharedAccountsFromParentUser(parentUserHandle,
                        UserHandle.of(user.id));
            }
            return user;
        } catch (RemoteException re) {
            throw re.rethrowFromSystemServer();
        }
    }
```

- 小结：
  UserManager给出了6种创建用户的方式，进一步分析则发现实质上一共调用了4种

UserManagerService的创建方式:

- mService.createUser
- mService.createProfileForUser
- mService.createProfileForUserEvenWhenDisallowed
- mService.createRestrictedProfile

实际上这4种方式在UserManagerService中最终都调用到最核心的方法createUserInternalUnchecked来完成用户创建。而他们的主要区别在于传入的Flag。

具体创建流程分析

下面来跟踪这4种方式在UserManagerServices是如何一一走入createUserInternalUnchecked

- 1.最常用的createUser。其特征为创建新用户时，不对新用户进行部分应用禁用，且不是创建子用户，且需要明确是否能够创建用户。

```
    public UserInfo createUser(String name, int flags) {
     //普通模式入口，传入参数少。
        checkManageOrCreateUsersPermission(flags);
       //不创建子用户，因此不传入主用户id，即USER_NULL
        return createUserInternal(name, flags, UserHandle.USER_NULL);
    }
```

```
    private UserInfo createUserInternal(String name, int flags, int parentId) {
       //此方式表示创建的新用户，所有应用均可正常使用，于是第4个参数传入为null
        return createUserInternal(name, flags, parentId, null);
    }
```

```
    private UserInfo createUserInternal(String name, int flags, int parentId,
            String[] disallowedPackages) {
        //这是比较完整的创建用户的参数，只是需要判断一下是否可以创建用户。
        String restriction = ((flags & UserInfo.FLAG_MANAGED_PROFILE) != 0)
                ? UserManager.DISALLOW_ADD_MANAGED_PROFILE
                : UserManager.DISALLOW_ADD_USER;
        if (hasUserRestriction(restriction, UserHandle.getCallingUserId())) {
            //判断是否可以创建用户
            Log.w(LOG_TAG, "Cannot add user. " + restriction + " is enabled.");
            return null;
        }
      //最终走到createUserInternalUnchecked
        return createUserInternalUnchecked(name, flags, parentId, disallowedPackages);
    }
```

- 2.createProfileForUser和createUser基本一模一样，也是通过createUserInternal，传入参数中可以传入paretId 和 disallowedPackages。

```
     public UserInfo createProfileForUser(String name, int flags, int userId,
             String[] disallowedPackages) {
         checkManageOrCreateUsersPermission(flags);
        //和createUser的区别就是增加主用户（paretId）和应用禁用（disallowedPackages）的参数入口
         return createUserInternal(name, flags, userId, disallowedPackages);
     }
```

```
private UserInfo createUserInternal(String name, int flags, int parentId,
            String[] disallowedPackages) {
        //这是比较完整的创建用户的参数，只是需要判断一下是否可以创建用户。
        String restriction = ((flags & UserInfo.FLAG_MANAGED_PROFILE) != 0)
                ? UserManager.DISALLOW_ADD_MANAGED_PROFILE
                : UserManager.DISALLOW_ADD_USER;
        if (hasUserRestriction(restriction, UserHandle.getCallingUserId())) {
      //判断是否可以创建用户
            Log.w(LOG_TAG, "Cannot add user. " + restriction + " is enabled.");
            return null;
        }
      //最终走到createUserInternalUnchecked
```

```
            return createUserInternalUnchecked(name, flags, parentId, disallowedPackages);

        }
```

- 3.createProfileForUserEvenWhenDisallowed 也是创建子用户，不经过createUserInternal来确认是否可以创建用户，而是直接进入createUserInternalUnchecked

```
    public UserInfo createProfileForUserEvenWhenDisallowed(String name, int flags, int userId,String[] disallow
            checkManageOrCreateUsersPermission(flags);
            //最终走到createUserInternalUnchecked
            return createUserInternalUnchecked(name, flags, userId, disallowedPackages);
        }
```

- 4.createRestrictedProfile一个快捷入口，本质是调用createProfileForUser，自然最终走入createUserInternalUnchecked ，主要是锁定了flag

```
public UserInfo createRestrictedProfile(String name, int parentUserId) {
        checkManageOrCreateUsersPermission("setupRestrictedProfile");
        final UserInfo user = createProfileForUser(
                name, UserInfo.FLAG_RESTRICTED, parentUserId, null);
        if (user == null) {
            return null;
        }
        long identity = Binder.clearCallingIdentity();
        try {
            setUserRestriction(UserManager.DISALLOW_MODIFY_ACCOUNTS, true, user.id);
            android.provider.Settings.Secure.putIntForUser(mContext.getContentResolver(),
                    android.provider.Settings.Secure.LOCATION_MODE,
                    android.provider.Settings.Secure.LOCATION_MODE_OFF, user.id);
            setUserRestriction(UserManager.DISALLOW_SHARE_LOCATION, true, user.id);
        } finally {
            Binder.restoreCallingIdentity(identity);
        }
        return user;
    }
```

小结：在创建用户的流程中，最终统一走到了createUserInternalUnchecked 方法，这个方法需要传入用户名，用户特征，主用户编号，禁用应用4个参数。当主用户名为空则表示普通用户，主用户编号不为空则创建的用户为主用户编号的子用户。而从开发接口来看，必须是设定子用户的路线才能设定禁用应用。

```
private UserInfo createUserInternalUnchecked(String name, int flags, int parentId,
            String[] disallowedPackages)
```

总结，对创建用户的接口进行了与对外接口有关部分的梳理。从app可以调用的UserManager对应的6个接口，到UserManagerService际开放的4个接口，最终统一到了createUserInternalUnchecked 这一个创建用户的方法。

- 5. createUserInternalUnchecked

在此方法中要完成一个用户的创建。首先是通过flag区分需要创建用户的类型，然后不同类型在信息的使用上会有不同差别。 总的来说，创建时兼顾了各种类型的user，并且给新用户提供userId和UserInfo两个重要参数。需要注意的是，创建用户并不会使用户显示出来。要显示用户是通过多用户切换功能完成。我们平常一创建多用户就跳转到新用户的界面，那是由创建用户和切换用户共同完成。

下面是createUserInternalUnchecked 完整源码和讲解。

代码目录：

```
frameworks/base/services/core/java/com/android/server/pm/UserManagerService.java
```

```
private UserInfo createUserInternalUnchecked(String name, int flags, int parentId,String[] disallowedPackages
        //首先如同上一章所讲，传入参数为 用户的名字name ，  用户的类型/特性 flags ， 主用户的识别码 parentId， 和新创建用户时的
        //如果单纯创建一个新用户， 那么这4个参数，分别是： createUserInternalUnchecked（“自定义的名称”， 0，    USER.NULL
        //接着进行第一个判断，手机是否有足够空间支持新用户的创建，如果没有足够空间，则终止创建，return null
        DeviceStorageMonitorInternal dsm = LocalServices.getService(DeviceStorageMonitorInternal.class);
        if (dsm.isMemoryLow()) {
            Log.w(LOG_TAG, "Cannot add user. Not enough space on disk.");
            return null;
        }
        //随后对用户的特性进行一下统计，方面后续涉及到的部分进行快速判断。这里没有统计所有特性，且特性可以自己在UserInfo类里面，自己
        //特性1：访客用户
        final boolean isGuest = (flags & UserInfo.FLAG_GUEST) != 0;

        //特性2：工作用户（工作用户一定是子用户）
        final boolean isManagedProfile = (flags & UserInfo.FLAG_MANAGED_PROFILE) != 0;
        //特性3：受限制
        final boolean isRestricted = (flags & UserInfo.FLAG_RESTRICTED) != 0;
        //特性4：演示用户
        final boolean isDemo = (flags & UserInfo.FLAG_DEMO) != 0;
       // 接来下是最关注的几个对象，其中userInfo和userId是在多用户过程中，非常频繁使用的。
        final long ident = Binder.clearCallingIdentity();
        UserInfo userInfo;
        UserData userData;
```

```
        final int userId;
        try {
            synchronized (mPackagesLock) {
                UserData parent = null;
                //确认传入的主用户参数是否存在，当主用户存在，则获取parent。后面创建子用户过程中，部分信息会依赖于parent
                if (parentId != UserHandle.USER_NULL) {
                    synchronized (mUsersLock) {
                        parent = getUserDataLU(parentId);
                    }
                    if (parent == null) return null;
                }
                //确认工作用户能够被创建
                if (isManagedProfile && !canAddMoreManagedProfiles(parentId, false)) {
                    Log.e(LOG_TAG, "Cannot add more managed profiles for user " + parentId);
                    return null;
                }
                //正常用户数量是否抵达上限
                if (!isGuest && !isManagedProfile && !isDemo && isUserLimitReached()) {
                    // If we're not adding a guest/demo user or a managed profile and the limit has
                    // been reached, cannot add a user.
                    return null;
                }
                //是否在已经存在访客用户时，还创建访客用户，禁止此情况。
                if (isGuest && findCurrentGuestUser() != null) {
                    return null;
                }
                // 受管控用户必须由系统用户管控
                if (isRestricted && !UserManager.isSplitSystemUser()
                        && (parentId != UserHandle.USER_SYSTEM)) {
                    Log.w(LOG_TAG, "Cannot add restricted profile - parent user must be owner");
                    return null;
                }
                //在split system user mode下创建受管控用户必须满足的条件
                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;
                }
            }
            //在split system user mode下创建受用户必须满足的条件
            if (UserManager.isSplitSystemUser()
                    && !isGuest && !isManagedProfile && getPrimaryUser() == null) {
                flags |= UserInfo.FLAG_PRIMARY;
                synchronized (mUsersLock) {
                    if (!mIsDeviceManaged) {
                        flags |= UserInfo.FLAG_ADMIN;
                    }
                }
            }

//在进行前置条件排除之后，此处开始正式创建用户
//核心参数1：获取可用的userId，这里涉及一个方法getNextAvailableId，该方法进一步调用scanNextAvailableIdLocked方法，下面是

int getNextAvailableId() {
    int nextId = scanNextAvailableIdLocked();
    return nextId;
}
static final int MIN_USER_ID = 10;
private int scanNextAvailableIdLocked() {
    for (int i = MIN_USER_ID; i < MAX_USER_ID; i++) {
        if (mUsers.indexOfKey(i) < 0 && !mRemovingUserIds.get(i)) {
            return i;
        }
    }
    return -1;
}

//正常开机自动创建的系统用户的userId为0， 而后的userId完全靠创建顺序，从10开始，依次11,12,13增长。
```

```
userId = getNextAvailableId();
//创建用户文件夹
Environment.getUserSystemDirectory(userId).mkdirs();
// 检查是否是ephemeralGuests模式，这种模式创建用户后，会立刻切换，而当我们用这个用户离开之后，这个用户就会立刻自动销毁。
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;
    }

 //核心参数2，创建userInfo，注意该userInfo最终会返回调用createUser的app：
    userInfo = new UserInfo(userId, name, null, flags);
    userInfo.serialNumber = mNextSerialNumber++;
    long now = System.currentTimeMillis();
    userInfo.creationTime = (now > EPOCH_PLUS_30_YEARS) ? now : 0;
    userInfo.partial = true;
    userInfo.lastLoggedInFingerprint = Build.FINGERPRINT;
    if (isManagedProfile && parentId != UserHandle.USER_NULL) {
        userInfo.profileBadge = getFreeProfileBadgeLU(parentId);
    }

 //核心参数3，UserData,保持创建完成的用户数据
    userData = new UserData();
    userData.info = userInfo;
    mUsers.put(userId, userData);
    }
    //将用户数据写入本地文件
    writeUserLP(userData);
    writeUserListLP();
    //如果本次创建的用户是子用户则需要将其对应的主用户信息也保存下来。
    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;
        }
        }
      }
```

//保存userkey，也就是存下用户的一些基本参数，用于快速查找。

```
     final StorageManager storage = mContext.getSystemService(StorageManager.class);
     storage.createUserKey(userId, userInfo.serialNumber, userInfo.isEphemeral());
     mUserDataPreparer.prepareUserData(userId, userInfo.serialNumber,
    StorageManager.FLAG_STORAGE_DE | StorageManager.FLAG_STORAGE_CE);
```

//通知packageManager新用户被创建了，这样packageManager会对新的用户安装对应的应用信息，当然
在disallowedPackages里面的就不会安装到新的用户中。

```
            mPm.createNewUser(userId, disallowedPackages);
            userInfo.partial = false;
            synchronized (mPackagesLock) {
                writeUserLP(userData);
            }
       //创建成功之后，更新userId的序列，为下一个用户做准备
            updateUserIds();
       //而后进入用户的管控初始化，仅仅对于需要管控的内容进行管控
            Bundle restrictions = new Bundle();
      // 比如这里就是设置的访客现在，访客限制可以通过mGuestRestrictions查询，大致有禁止拨号，禁止短信等
            if (isGuest) {
                synchronized (mGuestRestrictions) {
                    restrictions.putAll(mGuestRestrictions);
                }
            }
            synchronized (mRestrictionsLock) {
```

```
                mBaseUserRestrictions.append(userId, restrictions);
            }
            mPm.onNewUserCreated(userId);
            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
                    : (isDemo ? TRON_DEMO_CREATED : TRON_USER_CREATED), 1);
        } finally {
            Binder.restoreCallingIdentity(ident);
        }
        //完成创建
        return userInfo;
    }
```

整个多用户的创建到此完成。回顾一下，最常见的创建方案是createUser，只需要传入用户的名称，以及0（默认的flag）。自己进行测试的时候，可以编写一个按钮，按一下就调用一个createUser，那么就可以发现，点几下就能在手机里面创建几个用户。
当对于创建用户有一些想法的时候，就要利用flag了，flag在UserInfo这个类里面进行查询。一般建议这么使用：

int flag = UserInfo.FLAG_INITIALIZED | UserInfo.FLAG_GUEST；

这样用或来组成目标flag， 以上的意思是创建不需经过开机向导的访客用户