上文提到创建用户之后，需要通过切换用户来抵达用户。细心的朋友会发现标题从UserManagerService变成了UserController。这是因为切换的代码主要在UserController实现，有意思的是，切换的入口也不是在UserManager而是在ActivityManagerService。

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

而进入ActivityManagerService里面查看switchUser代码，会发现他实际调用的是UserContoller的代码，而实际上整个切换流程的核心流程就是在UserContoller里面。

```
public boolean switchUser(final int targetUserId) {
        return mUserController.switchUser(targetUserId);
    }
```

简介一下切换的大致流程。

- 1：发起切换流程，确定满足切换要求
- 2：弹出正在切换的提示框并冻屏
- 3：启动目标用户的最佳应用（如同每次开机都要启动主用户的最佳应用）且发起锁屏
- 4：解除冻屏，并通告其他模块，用户已切换

也就是说作为框架，主要做了启动应用和发起锁屏两件事儿，其他的是通过观察者模式通知各个模块来完成。

本章主要讲解，从发起用户切换到真正用户切换的准备工作。

下面进行源码详解，首先是切换源码,其做了3件事情，通过UserId获取UserInfo，判断是否满足切换条件，开始切换：

```
boolean switchUser(final int targetUserId) {
        enforceShellRestriction(UserManager.DISALLOW_DEBUGGING_FEATURES, targetUserId);
        //通过TargetUserId获取targetUserInfo
        int currentUserId = getCurrentUserId();
        UserInfo targetUserInfo = getUserInfo(targetUserId);
        //以下是几种不切换的常见
        //不可切换场景1：切换的目标用户就是当前用户
        if (targetUserId == currentUserId) {
            Slog.i(TAG, "user #" + targetUserId + " is already the current user");
            return true;
        }
        //不可切换场景2：切换的目标用户不存在
        if (targetUserInfo == null) {
            Slog.w(TAG, "No user info for user #" + targetUserId);
            return false;
```

```
        }
        //不可切换场景3：目标用户是属于可进入的
        if (!targetUserInfo.supportsSwitchTo()) {
            Slog.w(TAG, "Cannot switch to User #" + targetUserId + ": not supported");
            return false;
        }
        //不可切换场景4：目标用户是工作用户，无需切换
        if (targetUserInfo.isManagedProfile()) {
            Slog.w(TAG, "Cannot switch to User #" + targetUserId + ": not a full user");
            return false;
        }
        //到此出则表示可以切换，于是锁定切换目标的UserId
        synchronized (mLock) {
            mTargetUserId = targetUserId;
        }
        //根据切换的目标，选择有提示的切换和无提示的切换。其中START_USER_SWITCH_UI_MSG表示有提示的切换。START_USER_SWITCH_FG_
        是无提示的切换。
        if (mUserSwitchUiEnabled) {
            UserInfo currentUserInfo = getUserInfo(currentUserId);
            Pair<UserInfo, UserInfo> userNames = new Pair<>(currentUserInfo, targetUserInfo);
            mUiHandler.removeMessages(START_USER_SWITCH_UI_MSG);
            mUiHandler.sendMessage(mHandler.obtainMessage(
                    START_USER_SWITCH_UI_MSG, userNames));
        } else {
            mHandler.removeMessages(START_USER_SWITCH_FG_MSG);
            mHandler.sendMessage(mHandler.obtainMessage(
                    START_USER_SWITCH_FG_MSG, targetUserId, 0));
        }
        return true;
    }
```

到此出，可以看到还处于切换的准备工作，还没到切换的流程中。代码走到最后的关键代码是两种切换。下面详细看看两种切换，首先看简单的无提示切换
START_USER_SWITCH_FG_MSG，该切换方式是调用startUserInForeground方法，并传入一个参数，这个参数实际上就是targetUserId。

```
public boolean handleMessage(Message msg) {
        switch (msg.what) {
            case START_USER_SWITCH_FG_MSG:
```

```
                startUserInForeground(msg.arg1);
                break;
```

记住这个startUserInForeground方法，因为有提示切换最终也是走到startUserInForeground方法。

下面看有提示切换是如何一步步走到startUserInForeground方法的。

```
 case START_USER_SWITCH_UI_MSG:
                showUserSwitchDialog((Pair<UserInfo, UserInfo>) msg.obj);
                break;
```

首先根据message锁定实际调用了showUserSwitchDialog方法，顾名思义用dialog来显示切换的提示

而后进一步跟踪showUserSwitchDialog方法，发现其调用了showUserSwitchingDialog方法。

```
private void showUserSwitchDialog(Pair<UserInfo, UserInfo> fromToUserPair) {
        mInjector.showUserSwitchingDialog(fromToUserPair.first, fromToUserPair.second,
                getSwitchingFromSystemUserMessage(), getSwitchingToSystemUserMessage());
    }
```

在进一步跟踪showUserSwitchingDialog方法，发现其做了两件事，一个是创建UserSwitchingDialog，另一个是调用UserSwitchingDialog的show方法。

```
 void showUserSwitchingDialog(UserInfo fromUser, UserInfo toUser,
    String switchingFromSystemUserMessage, String switchingToSystemUserMessage) {
    Dialog d;
    if (!mService.mContext.getPackageManager()
          .hasSystemFeature(PackageManager.FEATURE_AUTOMOTIVE)) {
      //创建UserSwitchingDialog，目的是调用它的show方法
      d = new UserSwitchingDialog(mService, mService.mContext, fromUser, toUser,
          true /* above system */, switchingFromSystemUserMessage,
          switchingToSystemUserMessage);
    } else {
      //创建 CarUserSwitchingDialog，目的是调用它的show方法。实际上CarUserSwitchingDialog是UserSwitchingDialog的子类，
      d = new CarUserSwitchingDialog(mService, mService.mContext, fromUser, toUser,
          true /* above system */, switchingFromSystemUserMessage,
          switchingToSystemUserMessage);
    }
    //最终是为了调用它的show方法。
    d.show();
```

```
    }
```

到这里我们还是一头雾水，这里我们记住有提示的切换用户，最终是在UserSwitchingDialog这个类里面实现的。于是，我们接着进入到UserSwitchingDialog
类里面去看它的show方法：

```
    public void show() {
    //这个super.show()是看代码时很容易忽略的一行代买，然而在这里非常重要，它是将自身也就是dialog显示到桌面上。
    //关于dialog的定义，请大家自己到UserSwitchingDialog类里面去查看，显示的内容属于切换流程的分支就不展开讲了。
        super.show();
        final View decorView = getWindow().getDecorView();
        if (decorView != null) {
             decorView.getViewTreeObserver().addOnWindowShownListener(this);
        }
       //延时发送MSG_START_USER信息
        mHandler.sendMessageDelayed(mHandler.obtainMessage(MSG_START_USER),
                WINDOW_SHOWN_TIMEOUT_MS);
    }
```

这里我们看到核心是一个MSG_START_USER的延时推送，于是接着跟上看MSG_START_USER具体的实现，而后我们就会发现这里的handleMessage就此一个case，
这就充分说明这个message的目的就是延迟发送，那么继续关注UserSwitchingDialog类里面的startUser方法：

```
  case MSG_START_USER:
                    startUser();
                    break;
```

走到startUser方法，终于看到我们的目标代码startUserInForeground了，同时还有dismiss来取消自身的dialog，

```
void startUser() {
        synchronized (this) {
            if (!mStartedUser) {
          //此处是核心代码，调用startUserInForeground
                mService.mUserController.startUserInForeground(mUserId);
           //此处是易被忽略的代码，dismiss和show是成对出现，分别控制dialog的显示和消失。
                dismiss();
                mStartedUser = true;
                final View decorView = getWindow().getDecorView();
                if (decorView != null) {
```

```
                    decorView.getViewTreeObserver().removeOnWindowShownListener(this);
                }
                mHandler.removeMessages(MSG_START_USER);
            }
        }
    }
```

于是到此可以看到有提示切换用户和无提示切换用户的区别在于，无提示切换用直接调用到startUserInForeground。而有提示则使用UserSwitchingDialog后
调用到startUserInForeground

接来下看看两者共同的目标方法 startUserInForeground

```
void startUserInForeground(final int targetUserId) {
//引入UserController的startUser方法
        boolean success = startUser(targetUserId, /* foreground */ true);
        if (!success) {
            mInjector.getWindowManager().setSwitchingUser(false);
        }
  }
```

这里是调用两个参数的startUser从而最终调用到三个参数的startUser。

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

```
    return startUser(userId, foreground, null);
}
```

小结，第三方app要切换用户，是通过ActivityManagerService的SwtichUser方法，该方法只需要给出正确的UserId就能确保可以切换到目标用户。
真正在使用时，需要首先明确该用户已经被创建且被可以被切换。不然在SwitchUser的准备阶段就会被干掉（return）。 而后根据定义，觉得切换用户
是否需要显示dialog提示，而后最终走到切换用户的最关键方法： UserController里面的startUser方法。