浅谈 Android 登录系统设计

DSonline 郭霖 2022-03-22 08:00

/ 今日科技快讯 /

近日，我国首个地铁北斗定位系统在北京地铁首都机场线开工建设，这是我国目前规模最大的室内空间导航定位系统。预计在今年内建成。

北斗卫星导航系统能够提供定位、导航和授时服务，是我国重要的空间基础设施。但是在地下、水下、建筑物内等非暴露空间，由于受到遮挡导致卫星信号无法直接接收使用，是导航定位应用亟待拓展的领域。

/ 作者简介 /

本篇文章来自wresource的投稿，文章主要分享了他对Android登录系统设计的相关心得和经验，相信会对大家有所帮助！同时也感谢作者贡献的精彩文章。

wresource博客地址：

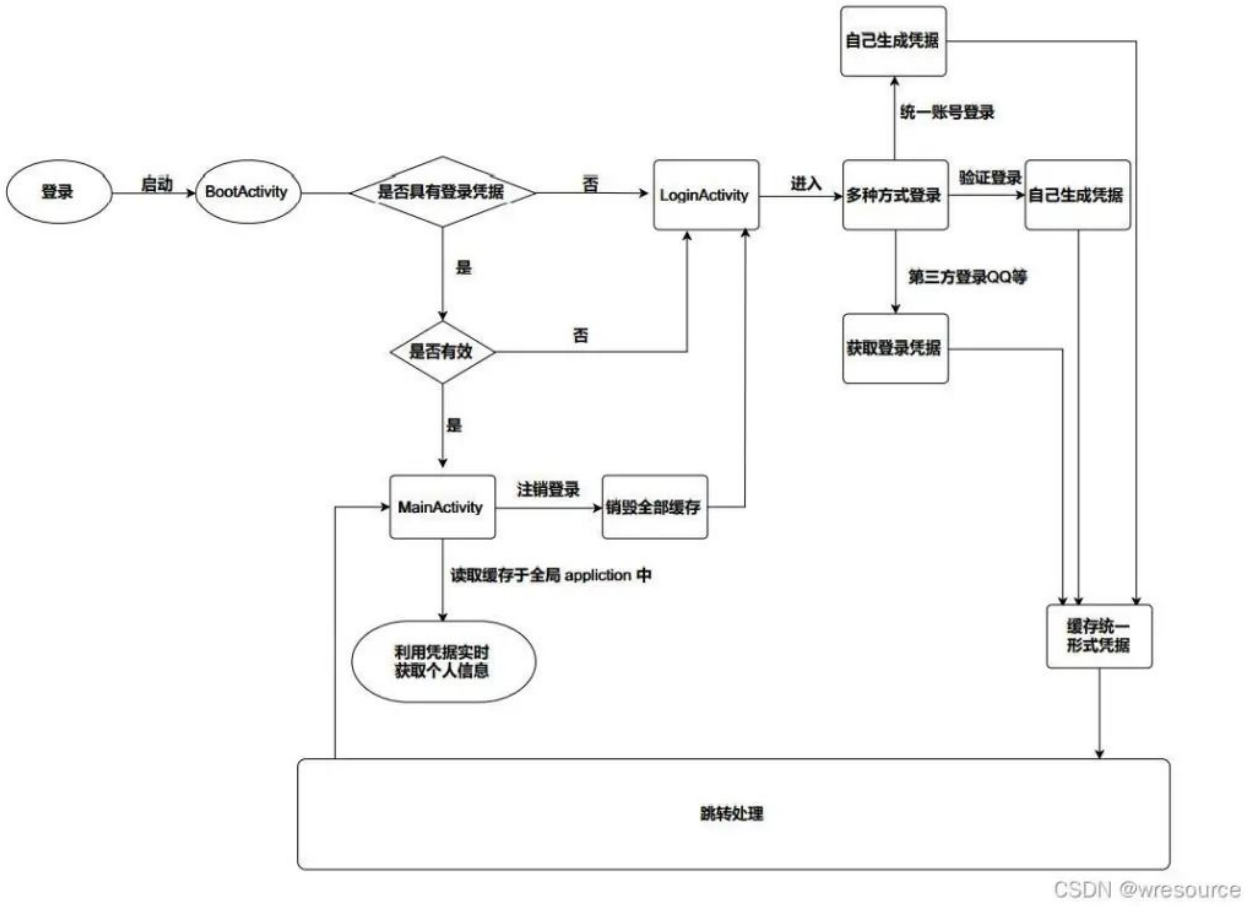
> https://dreamstudio.blog.csdn.net/

/ 前言 /

前段时间项目进入第一阶段的尾声，虽然登录方面的功能基本上已经完成开发，但是很乱，例如QQ登录等第三方登录有自己的缓存机制，本地的账号密码登录又是一种方式，邮箱手机号登录又是另一种方式，最终经过几个小时的逻辑推导，第一次在没有运行代码的情况下完成这个登录系统的开发，最终运行成功了，修改了一次没有初始化的情况，中间还出现了一些小插曲，最后完成这套系统的开发，目前app已上线谷歌应用商店，欢迎大家来体验。

/ 流程图 /

这个流程图包含了登录系统设计的全流程，也是当时模拟的时候一步步走的流程。

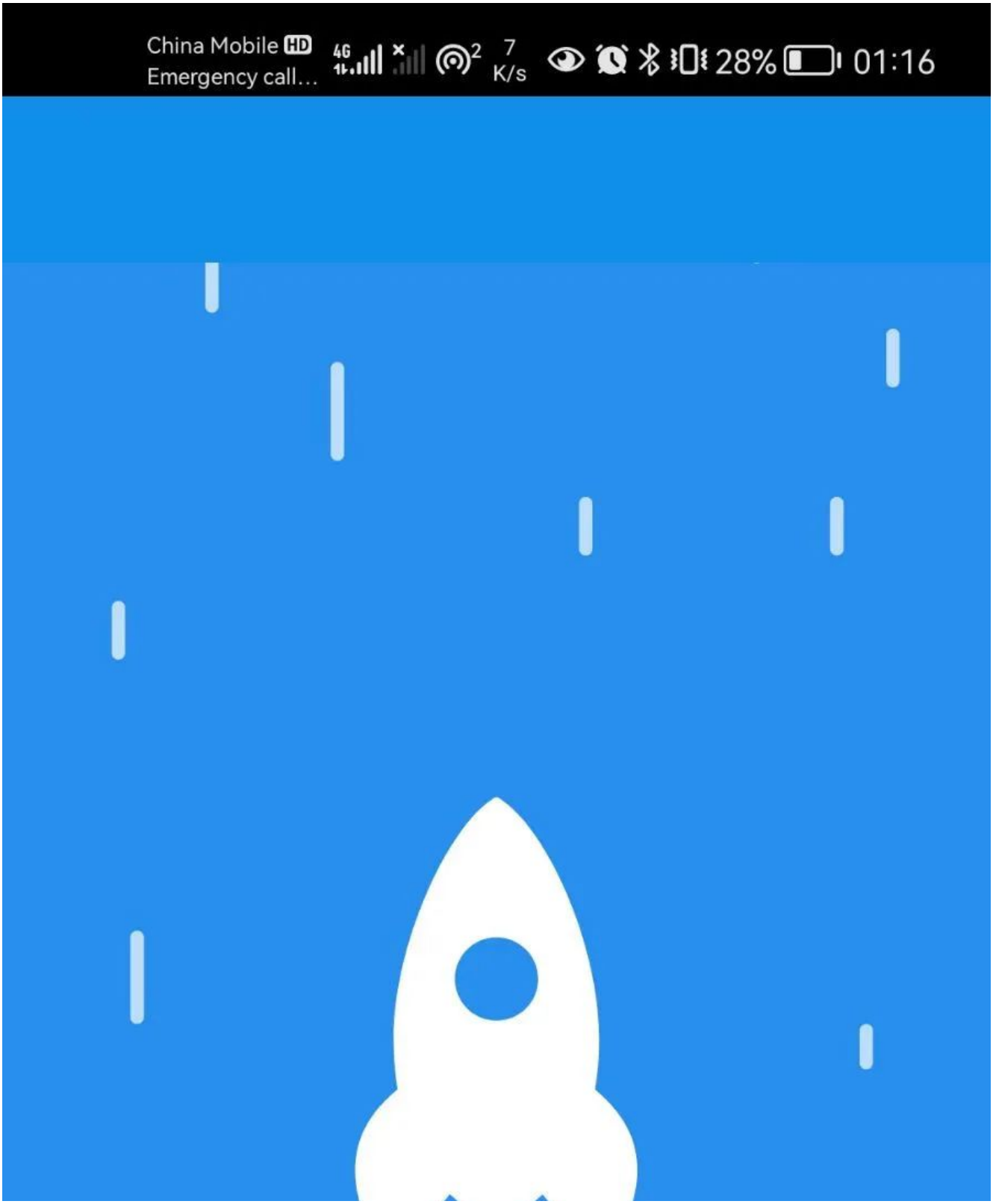


简要的用文字梳理一下登录流程。

1. 首先进入程序，需要进行判断登录缓存是否存在且有效，有效直接跳转 MainActivity，否则跳转 LoginActivity
2. 进入 LoginActivity 之后进行多种形式的缓存，这里的缓存是在各自的部分进行处理的，最后缓存成统一形式的Json数据，但有一点是必须保证的，在进入MainActivity时这些工作必须完成，因为后续即将使用到这些凭据
3. 然后跳转 MainActivity 前可以携带一些数据，或者进行一些动画的展示
4. 如果第一次登录的话进入MainActivity需要对全局application进行设置登录凭据，之后的登录在开始启动完成登录凭据的读取
5. 注销登录之后务必进行缓存的清理，包括登录缓存，退出应用时对于其他临时保存的json数据进行清理，类似个人信息之类的

6. 关于缓存是否有效，类似QQ等第三方登录有自己专门的检验机制，统一登录和邮箱登录这边我仅仅只做了时间上的验证，也可以考虑自定义设置验证机制，最后只要保证登录凭据的形式统一即可

这里关于启动界面采用Lottie view的形式展示，没有使用slash screen的设计，主要原因是想使得动画炫酷一点，然后也创建了BootActivity进行启动相关逻辑的处理，下图为启动界面图，这里由于引入动画，遇到一个坑(稍后会进行讲述)



/ 主要设计部分分析 /

由于个人开发限制，下列登录方式的方式以QQ，邮箱，统一账号登录为例。

登录缓存设计

统一登录行为是这个系统设计的核心，所以这部分的设计以简单和信息最小化为原则进行设计，下面这四个字段足够用了，也可以自行扩展。

```
//首先是唯一id字段，这个字段可以是第三方的openId也可以是其他的数据，只要保证数据唯一即可
//过期时间这个是可以设置的，登录的途径，与程序启动和进行统一账号的生成有关
data class LoginInfo(var id:String = "", var expireTime:Long = 0,
            var loginWay:String = "",var loginToken:String = "")
```

设计这样一个统一的缓存bean，然后利用 MMKV 进行键值存放相关的 json 数据，这里简单封装了Gson使用Kotlin的扩展函数完成json和对象之间的转化。

```
//String.toMyObject<>()为转化成对象的形式
//Any.toMyJson()为转化成json数据的封装

val kv = MMKV.defaultMMKV()
val loginInfo = LoginInfo(...)
kv.encode("login_info",loginInfo.toMyJson())
```

这个部分在进行各种途径登录成功之后完成缓存操作。

数据通讯设计

login 缓存

这个相当重要，设计缓存很多时候就是方便各个activity进行通信的，之前采用的是activity传值的方式进行的，每次传递都要编写一次代码，而且还容易出错，考虑到 login 这部分缓存具有很强的复用性，所以这部分需要放在全局的application中，然后需要的时候直接调用application即可。所以login 这部分的缓存是长期存放的数据，在登录成功之后进行设置application相关的缓存，同时也要考虑第一次登录或者注销登录之后的操作。

```
companion object{
    @SuppressLint("StaticFieldLeak")
    ...
    lateinit var login_info:LoginInfo
    ...
}

override fun onCreate() {
    super.onCreate()
    ...
    login_info = if (MMKV.defaultMMKV().containsKey("login_info")){
        //包含即可认为不为空直接读取
        login_info = MMKV.defaultMMKV().decodeString("login_info").toMyObject<LoginInfo>()
    }else{
        //否则初始化对象,这里如果未初始化会报错
        login_info = LoginInfo()
    }
    ...
}
```

login部分的衍生缓存

这部分需要在MainActivity里面进行操作，确保有login部分的缓存之后进行的操作，这部分可以根据自己的需求进行，我的做法是一旦失败回到登录界面，当然无网络或者弱网也会触发这个，这个就属于自己的个性化选择了，目前我自己的这个项目没有网络几乎不能完成任何操作，所以选择了这个做法。衍生缓存也可以根据自己的需要进行持久化缓存或者只存在在内存中，我这里的做法同login部分的，只是每次退出应用后都销毁这部分缓存（这部分主要是为了信息方面的安全考虑的），login部分不销毁。

例如这个QQ的获取个人信息，需要qqToken才能获取，即之前登录凭据中的accessToken。

```
private fun getQQInfo(){
   val qqToken = mTencent.qqToken
   val info = UserInfo(Application.context,qqToken)
   info.getUserInfo(object : DefaultUiListener(mTencent){
      override fun onComplete(response: Any?){
         kv.encode("qq_info",response.toString())
      }
   })
}
```

当退出MainActivity时，移除这部分的缓存即可，如果有一些不可变的数据，可以考虑持久化，这里使用kv当时是为了方便，可以全局通信，使用全局的application也是可以的，效果一样，如果想要后续进行一些其他的持久化可以考虑MMKV。

```
kv.remove("qq_info")
```

数据异常处理

关于BootActivity的一些坑

这里主要是考虑这样一个场景，当用户在等待启动动画的时候，由于意外切换到了后台，然后再次进入，由于动画的跳转是有条件的，即跳转逻辑还没处理完毕，导致退出之后，动画一直在循环(跳转逻辑这部分被跳过了)，这里的解决方案是调整逻辑处理的位置，放在onStart里面进行处理，如果onPause启动了，并且onStart部分的登录验证有效，就直接跳转

MainActivity，省略动画(其实还可以记录时间给予最佳体验)，如果无效则重新启动onStart中的逻辑，用到了一些简单的逻辑判断。

```
private var loginValid = false
private var onPause = false
...
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   ...
   //这里onCreate建议完成一些轻量级的任务，否则很有再次出现这种情况
}
override fun onStart() {
   super.onStart()
   if (!onPause){
      //跳转逻辑
      //跳转逻辑中的登录有效则
      //loginValid = true
      //否则不处理
   }else{
      //这里的startActivity参考郭神的第一行代码里面的重载函数
      startActivity<MainActivity>(Application.context){}
   }
}

override fun onPause() {
   super.onPause()
   if (loginValid){
      onPause = true
   }
}
```

login缓存部分数据被破坏

这部分的处理比较简单，可能由于MMKV缓存出错(听说这个概率极低，但还是以防万一)，一旦比如进行某种请求时发现数据查找不到记录，此时可以销毁key然后跳转到登录界面重新登录。

```
//这里之所以销毁全部key是因为，如果login_info这部分最重要的key出问题了，必须退出登录
kv.clearAll()
startActivity<LoginActivity>(context){}
finish()
```

如果是获取其他依赖login_info的数据，多次出错，可能是服务端的问题，也有可能是MMKV出问题了，此时重新登录是比较好的选择。

```
val retry = 0
//网络请求
...
if(retry > 5){
   kv.clearAll()
   startActivity<LoginActivity>(context){}
   finish()
}
```

这部分最重要的就是login_info这个key了，如果这个数据损坏只能重新登录。

涉及的activity以及它们各自的任务

BootActivity负责下一次进入的跳转处理和首次进入的初始化，同时可以放一些动画，例如Lottie View，具体见官网的使用。

```
<com.airbnb.lottie.LottieAnimationView
       android:id="@+id/animation_view"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       app:lottie_autoPlay="true"
       app:lottie_fileName="boot.json"
       app:lottie_loop="true" />
```

LoginActivity负责各种登录途径的入口处理，并不涉及具体登录逻辑。

```
class LoginActivity :BaseActivity<ActivityLoginBinding>(ActivityLoginBinding::inflate) {
   ...
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      ...
      viewBinding.qqLogin.setOnClickListener {
         ...
         //qqlogin部分处理
           ...
      }
      viewBinding.mailLogin.setOnClickListener {
         ...
            startActivity<MyLoginActivity>(FundTestApplication.context){
               putExtra("way","mailLogin")
            }
        ...
      }
      viewBinding.passwordLogin.setOnClickListener {
        ...
            startActivity<MyLoginActivity>(FundTestApplication.context){
```

```
            putExtra("way","passwordLogin")
        }
    ...
    }
  }
}
```

MyLoginActivity负责具体逻辑的处理

- QQ 这里使用 UIListener进行处理，同时sdk自带的activity完成登录具体逻辑的处理
- 邮箱和统一账号登录由此activity进行处理，需要完成两项任务

    - 本地数据的合法性检验(可以减少服务端的负担)
    - 发送本地的账号密码/邮箱验证码进行服务端那边的验证

/ 三种登录方式具体设计 /

第三方登录设计

这里以QQ举例，有兴趣的朋友可以参考上次写的QQ登录集成。

> https://dreamstudio.blog.csdn.net/article/details/122716185

QQ等第三方平台一般登录之后会生成自己的凭据，提取有用的信息作为缓存的登录凭据，比如唯一的 openId，access Token，过期时间，其他的信息如果有需要的自行进行提取，这三个是最基本的，如果只是登录的话，openId保证唯一性，accessToken保证数据的准确性，QQ平台有相应的验证登录是否有效的机制，不需要我们进行验证处理。

设置第三方登录凭据

这部分在登录成功之后即可开始。

```
val qqLogin = loginData.toMyObject<QQLogin>()

//首先设置QQ登录需要的凭据
mTencent.setAccessToken(qqLogin.access_token, qqLogin.expires_in.toString())
mTencent.openId = qqLogin.openid
```

设置缓存的登录凭据。

```
val loginInfo = LoginInfo(qqLogin.openid,qqLogin.expires_time,"qq",qqLogin.access_token)
kv.encode("login_info",loginInfo.toMyJson())
```

当首次进行登录时，需要设置到全局的application里。

```
Application.login_info = loginInfo
```

验证登录

首先验证登录的安全系数其实已经挺高了，目前主流的做法还有添加是否为机器人的检验，这部分暂时还没做，只做了验证码的验证

常见的几种形式

手机号、邮箱等，目前个人开发者只能使用邮箱进行发送验证码，手机号短信需要企业用户

验证码的生成

这里仅仅采用随机数的方法进行生成6位数字，不过这里就是服务端那边的工作了

统一登录的凭据中其他字段的生成

```
val loginInfo = LoginInfo(mail,expires_time,"mail","")
//id字段使用登录邮箱作为唯一id

//过期时间这里采用缓存时间+当前的时间生成过期时间戳，这里缓存一个月
//判断时只需要当前时间与过期时间进行比较即可
val expireTime = 30*86400*1000L + System.currentTimeMillis()

//登录途径这里即mail

//accessToken这里如果没有设计的话置空即可
```

统一账号登录

这个需要对每个第三方登录或者验证登录进来的账号进行处理，这里参考 CSDN 的id命名，途径+8位数字组成唯一id，这个id可以进行账号密码登录，所以这里没有设计账号注册系统了，主要考虑到其实大部分注册也需要邮箱手机号验证，目前也有好多平台没有自己的账号注册系统，不过自己可以给自己创建一些测试的账号便于登录。

统一账号设计

这里采用登录途径 + 8位纯数字构成统一id，重复了进行重新生成。

```
fun uid(platform:String):String{
   return "${platform}_${(Random().nextInt(99999999 - 10000000 + 1)+10000000)}"
}

//例如qq_36725737
//即代表由qq平台登录的
```

登录凭据设置

这里id即可替换成统一账号即可，都是唯一的无需担心重复，过期时间参考验证登录，途径这里设置成password即可，accessToken这里也同验证登录，同时如果时密码登录的话就不需要注册了。

然后这里提供忘记密码的选项，直接引导用户进行邮箱登录的提示，然后进入主界面里面进行设置密码，毕竟重置密码还是需要验证码的。

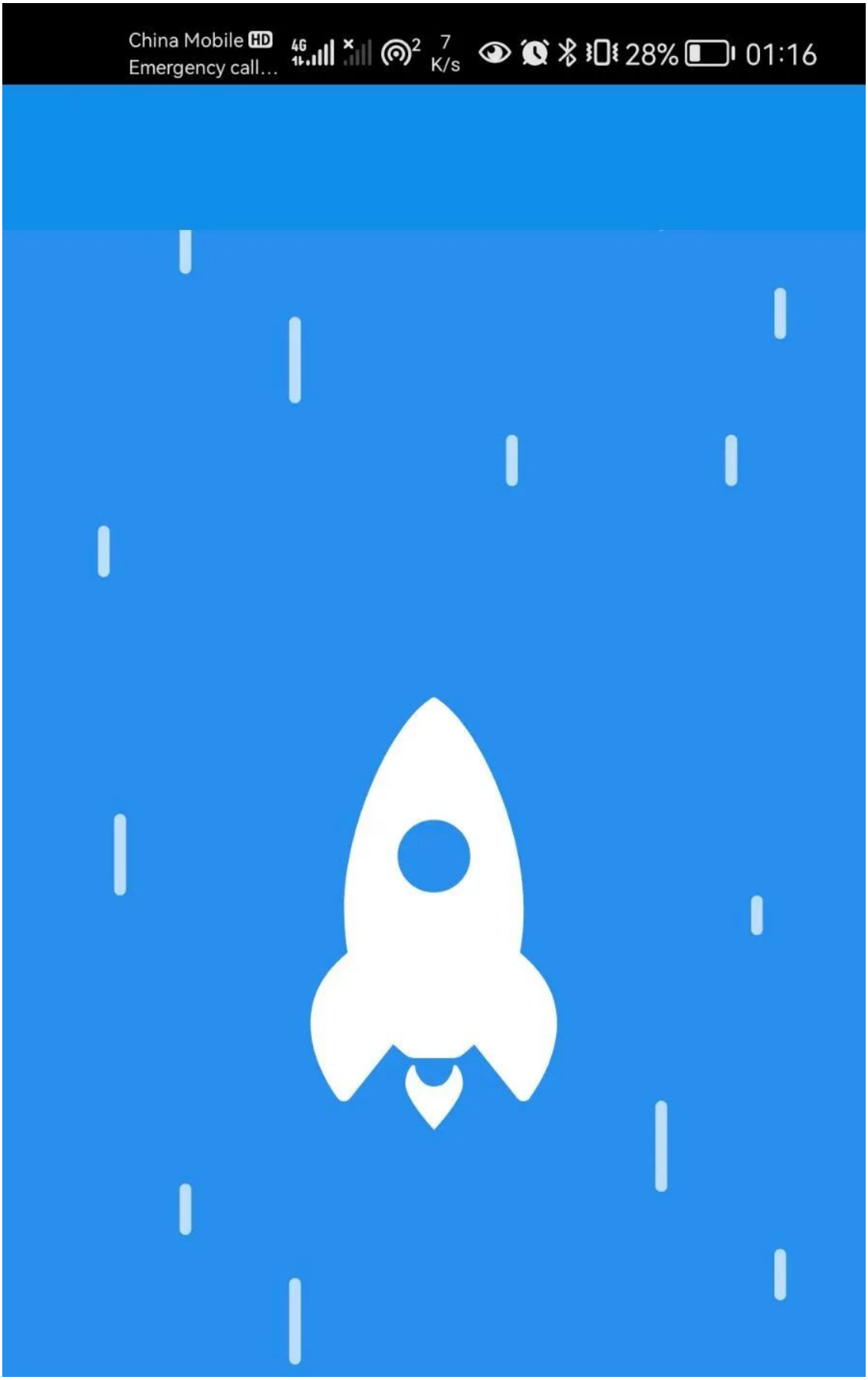
/ 总结 /

这次的登录流程的设计受益匪浅，统一的登录行为是设计中最核心的部分，可以进行扩展，同时便于进行管理，不过仍然有不足的地方，例如异地登录方面的考虑和设计，防机器人等的设计。

应用界面

目前利用此登录系统开发的app已经上线谷歌应用商店，算是上架的第一个应用，搜索VFund即可，欢迎前来体验。

部分界面展示



VFund

Fun to Real

LOGIN

MAILLOGIN

我已阅读并同意用户协议和隐私政策

邮箱登录

邮箱

验证码 发送验证码

LOGIN

Login

uid

password

FORGET

LOGIN

请使用邮箱登录

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

在微软工作365天，还你一个我眼中更加真实的微软

二维码原理解析，生成一个二维码需要这些知识

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

喜欢此内容的人还喜欢

微信语音彩铃的思考

爱偷懒的小贼猫

免费好用的PDF编辑软件，PDF转Word不限次数

271BAY帮助中心

羊了只羊最方便快捷的速通方法

小斗鱼大世界