# 多账号登录

目录

[多账号登录 1](#_Toc13572990)

[前言 2](#_Toc13572991)

[总结需求 2](#_Toc13572992)

[前端设计 3](#_Toc13572993)

[账号密码优先 3](#_Toc13572994)

[手机号快捷登陆优先 3](#_Toc13572995)

[第三方授权登陆优先 3](#_Toc13572996)

[服务端设计 4](#_Toc13572997)

[简单的用户登录 7](#_Toc13572998)

[添加微信登录 7](#_Toc13572999)

[如果又要添加QQ登录 8](#_Toc13573000)

[那么怎样才能设计灵活的登录 8](#_Toc13573001)

前言

用户系统是很多产品最基础的构成之一，但是越是基础越是开源设计想要完善也更难。在设计用户系统的时候，首先想到的关键词是注册和登录。但并不是有这两者就足够了，更加完善用户系统本身还需要考虑：多平台账号打通，同平台账号之间绑定与解绑，账号安全等及需要怎样的前端设计才是满足这个产品本身定位和用户操作的设计。

用户系统的实现简单来说有两种方式：一是自己构建用户系统；二是用第三方授权。第三方授权获得的用户信息有限且受制于人，而自己构建用户系统研发及用户使用成本均不如第三方授权。所以更多的是两者并存，相互补充。

由于工作需要从0到1设计一个用户系统，本人不是强技术型产品，所以在定义服务端字段或需求有不完善和不专业的地方，希望大家多提意见，共同完善。

总结需求

1.用户系统基本注册／登录功能及前端页面设计

2.多平台账号打通，即在单一app注册的用户，能够使用此账号登陆系统内所有app

3.用户相对独立，对于单一app来说用户身份唯一



前端设计

账号密码优先

账号密码优先是最常见的一种登录注册设计，适用于普遍场景，鼓励用户用注册方式登录，有利于产品引导用户完善更多的资料，留存自己的用户信息。例如知乎是以账号密码登录为最优先，且会隐藏第三方授权登录。现在的账号密码登录都会以用户注册方式代替系统生成的userid作为账号。纯账号密码登录是较为早期的设计，例如早期qq和飞信。

手机号快捷登陆优先

手机号快捷登陆，也叫免密登录／短信验证码登录，适用于用户不登录能够完成大部分行为，只有在某种场景下必须获得用户身份时才需要用户登录，且此时用户的想要完成的行为是被要求登录操作打断的。常见的如团购类产品，用户在应用内进行了商品的浏览、筛选、添加到购物车，当要进行最后一步付款操作时，发现未登录。这时繁琐的注册或者登录都有可能导致这笔订单甚至这个用户的流失。所以这时获取用户身份的方式一定要尽可能便捷。

第三方授权登陆优先

第三方授权登陆，适用于对用户资料和权限要求不高快速解约开发成本的产品。建议在应用构建用户系统的前期可以首先接入第三方，快速的实现登录功能。但是若想建设自身关系链还是需要完善自己的用户系统。

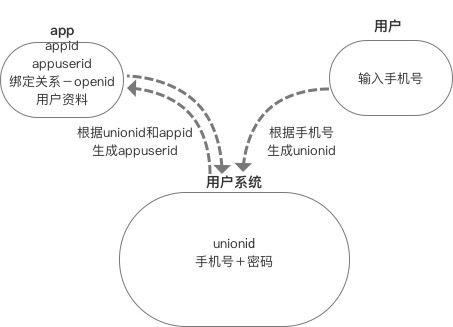
用户资料实际也属于用户系统等设计的内容。是否要增加此项的判断原则是根据这个产品对用户资料的需求程度决定用户注册时是否要增加资料填写页，资料填写页是强制阻断性的还是可跳过的，必填的资料项有哪些，希望填的有哪些。例如是需要关系链的那么注册的时候就应该强制用户去填写资料设置必要的昵称和头像，这样的用户对于此类应用来说才属于有效用户，不然在app内用户点进资料页，全是系统自动生成的垃圾信息，对于建设关系链和留存伤害较大。

服务端设计

简单的用户系统服务端的基本功能需求为：判断账号身份（注册/未注册），账号身份生成（新用户分配id），账号密码验证；这里要设计的并不满足于注册登录，需要考虑多平台账号打通的用户系统并且要和在打通情况下单一平台或多个平台之间，用户的多个账号之间绑定于解绑。

现在先说一下多平台账号打通需要考虑哪些问题：

1. 用户系统身份的创建，因为我们是多平台的系统，所以用户身份只能纳入手机号注册的用户，若第三方授权注册的也算用户系统用户，在账号绑定的那一关则会出现混乱；
2. 实现多平台账号打通，要实现多平台账号打通，即所有接入多平台都能够查询到此用户身份；
3. 平台间用户身份独立，要实现平台间用户身份独立，则需要在用户系统用户身份的基础上创建一个平台的用户身份。



**用户系统多平台打通**

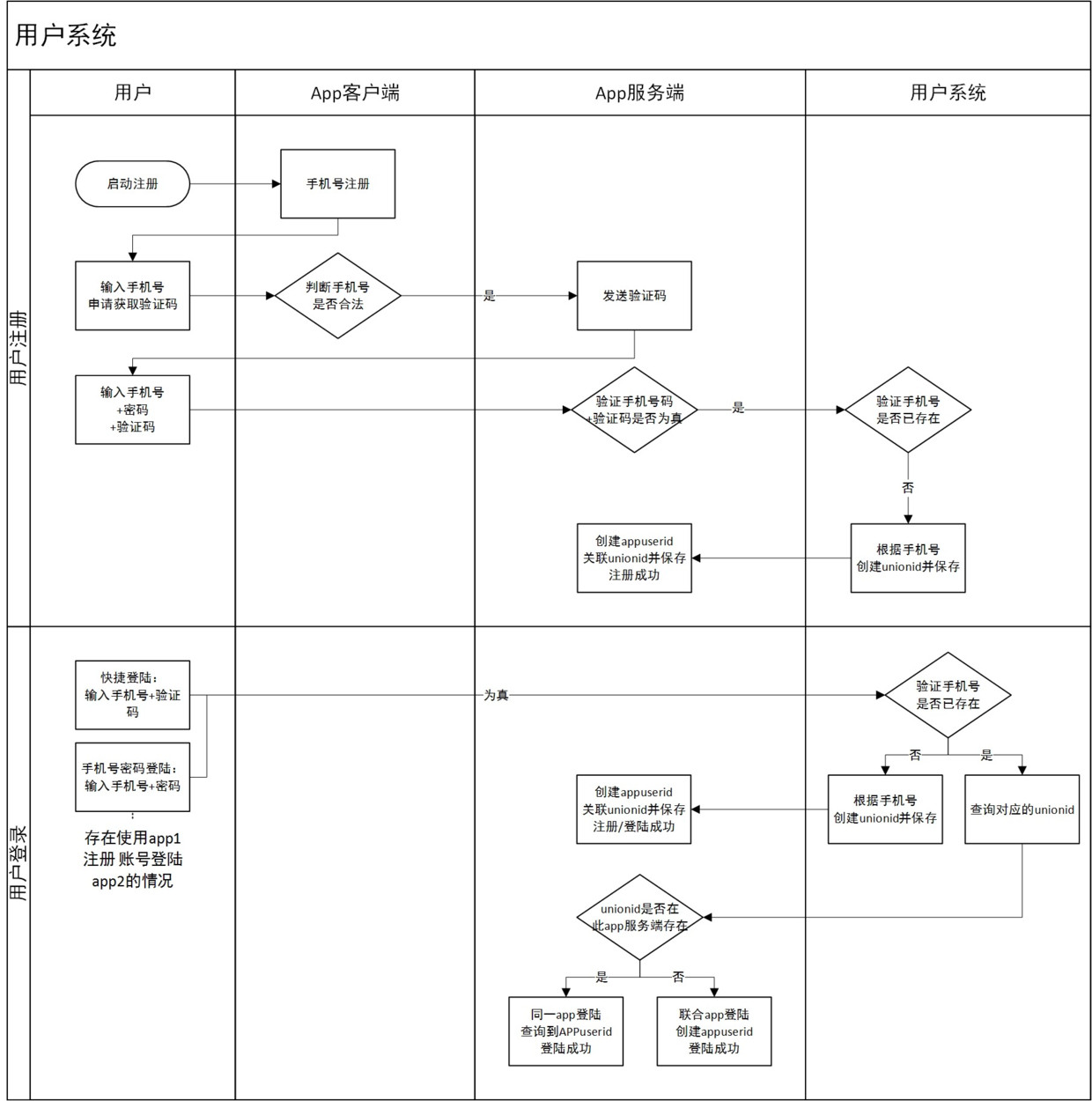
**名词解释**

1. Appid：接入用户系统时首先分配，用于区别接入的各个app；
2. Unionid：用户手机注册时，由用户系统根据手机号创建，在用户系统作为用户唯一身份标识；
3. Appuserid：用户注册时，由app服务端根据union或者第三方授权的openid创建，在app内作为用户唯一的身份标识。

**基本原则**

1. 手机号作为用户系统账号的注册的唯一途径，根据手机号在用户系统服务端创建并保存unionid；app服务端根据unionid创建并保存appuserid，且将unionid对应保存；
2. 用户系统同一unionid可对应不同的appuserid；
3. 使用第三方openid授权的注册用户不计入用户系统仅存在app服务端作为单app用户，即unioid为空只生成appuserid；第三方授权包括微博微信，QQ，Facebook，Twitter。

主线流程图：



* **手机号注册主流程为**：用户注册时，用户系统服务端需要验证手机号＋验证码是否为真，此手机号是否已有对应unionid；若有提示已注册，请登录；若无创建对应unionid，app服务端根据unionid创建appuserid。至此成功生成了用户系统身份及当前app用户身份。
* **手机号登陆主流程为**：用户登录时，用户系统服务的验证手机号＋密码是否为真，此手机号是否有对应unionid，若有，则说明此用户有用户系统身份。

还需要app服务端需要查询是否有对应的appuserid，若有说明此用户有此app身份，直接用其appuserid登录；若无则说明是用户系统内其他联合app注册用户根据unionid创建此app的用户身份，至此登录成功。

用户系统是大多数app都会有多构成，单一的用户系统也并不那么复杂，但是若要构建产品矩阵进行多平台间的用户打通，加上帐号绑定与解绑，并不是一时半会能够想清楚的需求，若大家感兴趣为会继续补充帐号绑定和账号安全产品应该关心和设计的事情。

简单的用户登录

原来系统的登录功能为用户使用登录名LoginName进行登录,登录数据库设计

|  |  |  |
| --- | --- | --- |
| 用户登录表(Login) | | |
| id | 主键ID |  |
| loginName | 登录名 |  |
| password | 登录密码 |  |

表数据如下

|  |  |  |
| --- | --- | --- |
| id | loginName | password |
| 1 | admin | admin |
| 2 | adminUser | admin |

用户登录时,如果loginName是admin,password是admin,查找Login表

|  |
| --- |
| select \* from Login where loginName =’admin’ and password =’admin’; |
| 如果查找结果不为Null,登录成功 |
| 否则,登录错误 |

添加微信登录

此时有个需求为添加微信登录,微信登录还会存在过期时间等问题

此时可以修改用户登录表(Login)如下

|  |  |  |
| --- | --- | --- |
| 用户登录表(Login) | | |
| id | 主键ID |  |
| loginName | 登录名 |  |
| password | 登录密码 |  |
| wxOpenId | 微信登录openId |  |
| wxExpire | 微信登录过期时间等 |  |

登录用户表信息如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | loginName | password | wxOpenId | wxExpire |
| 1 | admin | admin | 123\*\*\*\*\*\* | … |
| 2 | adminUser | admin | 124\*\*\*\*\*\* | … |

此时修改用户登录代码

|  |
| --- |
| 如果登录类型为用户LoginName登录,逻辑如上 |
| 如果登录类型为用户wxOpenId登录, |
| select \* from Login where wxOpenId=’admin’; |
| 如果查找结果不为Null,登录成功, 否则,登录错误 |

如果又要添加QQ登录

如果又需要添加QQ登录或者微博登录方式,那么按照现在的添加方式,那么需要修改表结构如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | loginName | password | wxOpenId | wxExpire | qqOpenId | weiboOpenId |
| 1 | admin | admin | 123\*\*\*\*\*\* | … | \*\*\* | \*\*\* |
| 2 | adminUser | admin | 124\*\*\*\*\*\* | … | \*\*\* | \*\*\* |

这么扩展,需要不断的维护表结构,

那么怎样才能设计灵活的登录

当用户使用任意方式登录成功后,我们获取的总是User表的一行记录,它实际表示用户的个人资料信息(UserInfo),而用户登录只是为了认证用户(Authenticate),所以无论使用本地账号密码(LoginName)或者第三方账号密码,本质都是为了认证

所以,把用户信息和认证信息分开,数据库表设计

|  |  |  |
| --- | --- | --- |
| 用户信息表(User) | | |
| id | 主键ID |  |
| username | 用户姓名 |  |
| phone | 手机号 |  |
| email | 邮箱 |  |
|  | | |
| 用户登录表(Login) | | |
| id | 主键ID |  |
| userId | 用户ID |  |
| oauthType | 登录方式  QQ weibo weixin … |  |
| oauthId | 认证号 |  |
| oauthToken | 登录Token |  |
| oauthExpire | 失效时间 |  |
| status | 状态 |  |

数据库保存信息如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | userId | oauthType | oauthId | oauthToken | oauthExpire | status |
| 1 | 1 | NAME | admin | admin |  | normal |
| 2 | 1 | QQ | 123\*\*\*\* |  |  | normal |
| 3 | 1 | WEIBO | 1\*\*\*\*\*\*\* |  |  | normal |
| 4 | 1 | WEIXIN | 1\*\*\*\*\*\*\* |  |  | normal |
| 5 | 2 | NAME | UserAdmin | admin |  | normal |

通过这种表结构设计，使许多原来纠结的问题瞬间解决。

　　优点：

　　1、站内登录类型无限拓展，代码改动小。如果真要支持身份证登录了，只要少许几处改动，无需修改表结构。

　　2、第三方登录类型可用工场模式批量拓展，新增第三方登录类型的开发成本降到最低。

　　3、在USER\_AUTHS表中增加一个统一的verified字段，每种登录方式都可以直观看到是否已验证情况。基于信任第三方登录的数据准确性，默认第三方登录都是已验证。如果用户修改登录手机号或登录邮箱，也能清晰跟踪每一步的完成度。

　　4、可按需绑定任意数量的同类型登录方式，即一个用户可以绑定多个微信，可以有多个邮箱，可以有多个手机号，是不是很赞！当然你也可以限制一种登录方式只有一条记录。

　　5、在USER\_AUTHS添加相应的时间和IP地址，就可以更加完整地跟踪用户的使用习惯（此部分应该单独建立日志表）。比如，已经不使用微博登录两年多，已经绑定微信300天。

　　6、即使完全使用第三方帐号登录，可在前端做到“无需注册本站帐号”的效果。过去许多网站虽然支持第三方帐号登录，但出于留存用户等原因，第一次微博登录回来，让你再填写一套他们网站的邮箱、密码等信息，也就失去了微博登录的最大意义。

　　从技术上说，原有的结构导致除了在微博用户表建立一个条目外，必须在用户表建立一条对应的条目，而且一般情况下不能让用户表里的邮箱或者用户名和密码留空。用户体验好的，邮箱自动生成 微博ID@id.weibo.sina.com ，密码则随机生成。至于体验不好的，只能说早知道还不如不用微博登录呢！现在呢，我们的这个用户表结构则完全没有这样的困扰，只要微博提供的昵称和头像地址就可以生成这个用户，再关联他的微博登录记录。而且我们的表结构意味着，用户可以解除他的所有登录方式，于是这个账户便彻底变成了没法登录的僵尸（解决办法是在代码里加一个限制，至少保留一条USER\_AUTHS的记录）。如果你非要得到用户的邮箱，那么每次登录的时候看到他不存在一条IDENTITYTYPE为email的记录，则弹窗弹死他，让他赶快填邮箱，否则啥都别干。

　　7、提升了逻辑思维能力。抽象出事物本质是码农必备职业素养。

　　8、如果你说邮箱和手机号就是用户信息的组成部分，他们依然需要体现在USERS表中作为前端展示。USERS表尽管拓展，USERS表里依然有email，phone。但它们仅仅作为“展示用途”，这和昵称、头像、或者性别这些属性没有本质区别。在用户信息表与用户授权登录拆分后，用户信息表可以随时增加任意字段，加星座，加生日，都没问题，只需要在前端展示时多几个输入框，录入时多几行代码，与用户登录相关的问题做到最大程度解耦。

　　缺点：

　　1、原先的用户判断由1次SQL变成2次SQL请求。  
　　2、用户同时存在邮箱、用户名、手机号等多种站内登录方式时，改密码时必须一起改，否则就变成了邮箱+新密码，手机号+旧密码访问了，肯定是很诡异的情况。如果考虑到这一点，又要在USER\_AUTHS表中新增一个表示站内登录方式或第三方登录方式的标识字段。  
　　3、代码量增加了，有些情况下逻辑判断增加了，难度增大了。

　　举个例子，无论用户是否已登录，无论用户是否已注册过，都是点击同一链接前往微博第三方授权后返回，可能出现几种情况：

　　1）该微博在本站未注册过，很好，直接给他注册关联并登录；

　　2）该微博已经在本站存在，当前用户未登录，直接登录成功；

　　3）该微博未在本站注册，但当前用户已经登录并关联的是另一个微博帐号，作何处理取决于是否允许绑定多个微博帐号；

　　4）该微博未在本站注册过，当前用户已登录，尝试进行绑定操作；