# 多账号统一登陆，账号模块的系统设计

现在几乎大部分的 App 都支持使用多个第三方账号进行登录，如：微信、QQ、微博等，我们把此称为多账号统一登陆。而这些账号的表设计，流程设计至关重要，不然后续扩展性贼差。

## ****一、 自建的登陆体系****

#### 1.1.1 手机号登陆注册

该设计的思路是每个手机号对应一个用户，手机号为必填项。

**流程：**

1. 首先输入手机号，然后发送到服务端。先判断该手机号是否存在账号，如果没有，就会生成随机验证码，将手机号和验证码绑定到 Redis中，并设置一定的过期时间（过期时间一般是5分钟，这就是我们一般手机验证码的有效期），最后将验证码通过短信发送给用户。
2. 用户接收到验证码后，在界面填写验证码以及密码等基础信息，然后将这些数据发送服务端。服务端收到后，先判断在 Redis里面这个手机号对应的验证码是否一致，，失败就返回错误码，成功就给用户创建一个账号和保存密码。
3. 注册成功后，用户即可通过自己的 手机号+密码进行登陆。

**问题：**

1. 用户体验差，需要完成获取验证码，填写验证码/密码/用户名等诸多的信息完成注册，然后才能使用；
2. 容易遗忘密码，遗忘后，只能通过忘记密码来重新设置密码。

#### 1.1.2 优化注册登陆

该方案的思路是弱化密码的必填性，即无论用户是否注册过，可通过 手机号+验证码 直接进行登陆(保留 手机号+密码登录的方式)。

**流程：**

1. 输入手机号，然后发送到服务端。服务端生成随机验证码，将手机号和验证码绑定到 Redis中，并设置一定的过期时间（过期时间一般是5分钟，这就是我们一般手机验证码的有效期），最后将验证码通过短信发送给用户。
2. 用户接收到验证码后，在界面只需填写收到的验证码，提交到服务端。服务端收到后，先判断在 Redis里面这个手机号对应的验证码是否一致，失败就返回错误码，成功就直接登录。如果是老用户，直接拉取用户信息；如果是新用户，则提示他可以完善用户信息（不强制）。
3. 用户通过 手机号+验证码登录后，也可选择设置密码，然后就可以通过 手机号+密码的方式登录，即：密码是非必填项。

**用户表设计：**

| **id** | **user\_name** | **user\_password** | **user\_mobile** | **state** | **more** |
| --- | --- | --- | --- | --- | --- |
| 用户id | 用户名 | 用户密码 | 手机号码 | 账号状态 | 其他信息 |

### 1.2 引入第三方账户方案

#### 1.2.1 微博登录

进入 Web2.0 时代 ,微博开放了第三方网站登录, 产品说, 这个我们得要, 加个用微博帐号就能登录我们的 App吧，而且得和我们自己的用户表关联。

**流程：**

1. 客户端调用微博登录的界面，进行输入用户名、密码，登录成功后，会返回 access\_token,通过 access\_token调取 API接口获取用户信息。
2. 服务端通过用户信息在我们用户表创建一个账号，以后，该第三方账号即可通过该微博账号直接进行登陆。

**微博用户信息表设计：**

| **id** | **user\_id** | **uid** | **access\_token** |
| --- | --- | --- | --- |
| 主键id | 用户id | 微博唯一id | 授权码 |

#### 1.2.2 噩梦来临

紧接着, QQ又开放用户登录了, 微信开放用户登录了，网易开发用户登录了。。。。。。一下子要接入好多家第三方登录了, 只能按照 “微博用户信息表” 新建一个表，重写一套各个第三方登录。

## ****二、 优化账号体系****

#### 2.1 原账号体系分析

1. **自建登陆体系：**无论 手机号+密码 , 还是 手机号+验证码 , 都是一种 用户信息+密码 的验证形式；
2. **第三方登录**：也是 用户信息+密码 的形式, 用户信息即第三方系统中的 ID(第三方系统中的唯一标识), 密码即 access\_token, 只不过是一种有使用时效定期修改的密码。

#### 2.2 新的账号体系

##### 2.2.1 数据表设计

**用户基础信息表：**

| **id** | **nickname** | **avatar** | **more** |
| --- | --- | --- | --- |
| 用户id | 昵称 | 头像 | 其他信息 |

**用户授权信息表：**

| **id** | **user\_id** | **identity\_type** | **identifier** | **credential** |
| --- | --- | --- | --- | --- |
| 主键id | 用户id | 登录类型(手机号/邮箱) 或第三方应用名称 (微信/微博等) | 手机号/邮箱/第三方的唯一标识 | 密码凭证 (自建账号的保存密码, 第三方的保存 token) |

**说明：**

1. 用户表分为 用户基础信息表 + 用户授权信息表；
2. 用户信息表不保存任何密码, 不保存任何登录信息(如用户名, 手机号, 邮箱), 只留有昵称、头像等基础信息; 所有和授权相关,都放在用户信息授权表, **用户信息表和用户授权表是一对多的关系**。

##### 2.2.2 登录流程

* **手机号+验证码**

沿用之前的方案。

* **邮箱/手机号+密码:**

用户填写 邮箱/手机号+密码; 请求登录的时候, 先判断类型, 如手机号登录为例：

使用 type='phone' 结合 identifier='手机号' 查找, 如有, 取出并判断 password\_hash(密码)是否和该条目的 credential 相符, 相符则通过验证, 随后通过 user\_id 获取用户信息;

* **第三方登录, 如微信登录：**

查询 type='weixin' 结合 identifier='微信 openId', 如果有记录, 则直接登录成功, 并更新 token; 假设与微信服务器通信不被劫持的情况下无需判断凭证问题。

##### 2.2.3 优缺点

###### 优点:

1. 登录类型无限扩展, 新增登录类型的开发成本显著降低;
2. 原来条件下, 应用需要验证手机号是否已验证和邮箱是否已验证, 需要相对应多一个字段如 phone\_verified 和 email\_verified, 如今只要在 用户授权信息表 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;
3. 在 用户授权信息表 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, 比如:已经不使用微博登录两年多, 已经绑定微信 300天;
4. 如果你说邮箱和手机号就是用户信息的组成部分, users 表尽管拓展, users 表里依然有email , phone , 但他们仅仅作为“展示用途”,和昵称,头像或者性别这些属性没有本质区别;
5. 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。当然你也可以限制一种登录方式只有一条记录;

###### 缺点 :

1. 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+旧密码都可以登录, 肯定是很诡异的情况;
2. 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；
   5. 该微博已经注册, 用户又已使用该帐号登录, 为何他重复绑定自己;
   6. 该微博已经在本站存在, 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?