链滴

uni-app 多端登录解决方案

作者：YYJeffrey
原文链接：https://ld246.com/article/1650096252777
来源网站：链滴

uni-app 是一个跨端开发的前端框架，可以使用一套代码同时开发 IOS、安卓、小程序的前端，是快开发移动应用的不错的选择。

在现实应用场景中，一个用户在使用当前移动应用时可能会用 IOS、安卓或小程序的任意一种。大部情况下，这个用户无论在使用何种设备环境下，他都是唯一标识的用户，设备分区那些应用除外（例腾讯的部分游戏）。

现有方案

那么如何标识出一个用户在不同设备环境下均为同一个用户呢？ 通常有以下几种方案：

1. 使用用户名和密码作为登录系统：该方案需要用户主动注册并设置用户名和密码，那么他在不同设环境下，使用自己账户登录即是唯一的。

2. 使用第三方登录作为登录系统：例如微信授权登录，在IOS和安卓应用中均可以实现跳转微信授权而小程序端如果只是微信小程序，那么在同一主体下理论上可以通过unionid标识出该用户，但支付或其他类型的小程序则无法应用该方案。

3. 使用手机号作为登录系统：用户手机号可以作为唯一标识用户的方法，且使用手机号授权的方式，比方案一使用账号密码的方案更加方便。

综上分析，第三种方案可能是最为普遍的方案，本文会以第三种方案使用手机号授权为例，来详细地述该方案的实行过程。

IOS、安卓一键登录

APP 端在使用手机号授权时会比较方便，因为 uni-app 提供了一键登录方案，可以按照官方给出的键登录文档 进行开发，笔者在该过程中排了一些坑，可以按照下文操作。

登录界面

第一步，最好是能先把这个一键登录的界面先能实现出来

文档中也提供了 uni.login() 方式即可

```
uni.login({
    provider: 'univerify',
    univerifyStyle: {
            fullScreen: true
    }
})
```

可以通过修改 univerifyStyle 属性个性化定制页面，具体也可以参考文档中 univerifyStyle 数据结构分。

```
authApp() {
    uni.login({
        provider: 'univerify',
        univerifyStyle: {
            fullScreen: true,
            otherLoginButton: {
                visible: false
            },
            icon: {
                path: "static/logo.png"            },
        },
        success: res => {
            console.log(res)
            if (res.errMsg === 'login:ok') {
                // 登录处理
            }
        }
    })
}
```

```
<center>
    <img src="https://b3logfile.com/file/2022/04/IMG_2745-0597e649.jpg" width="350" />
</center>
```

云函数换取手机号

在文档中可以发现，其提供了三种方式换取手机号，在点击授权后会收到一个 access_token 和一个 penid，此时需要使用 access_token 换取手机号。

用access_token换手机号

客户端获取到 `access_token` 后，传递给uniCloud云函数，云函数中通过 `uniCloud.getPhoneNumber` 方法获取真正的手机号。

这一步有3种方式：

1. uni-app项目开通uniCloud 服务，在前端直接写 `uniCloud.callFunction` ，将 `access_token` 传给指定的云函数。
2. 使用普通ajax请求提交 `access_token` 给uniCloud的云函数。这种方式uni-app和5+App、wap2app均可使用，但uniCloud上的云函数需要做URL化。
3. 使用普通ajax请求提交 `access_token` 给自己的传统服务器，通过自己的传统服务器再转发给 uniCloud 云函数。这种方式uni-app和5+App、wap2app均可使用，但uniCloud上的云函数需要做URL化。

简单地讲，它有三种换取手机号的方案：

1. 前端单纯使用云函数调用
2. 前端将参数 access_token 提交给云函数
3. 前端将参数 access_token 提交给自己的服务器，自己的服务器再调用云函数

这三种方案，可以发现一个共同的特点，就是没法离开云函数，开始笔者也想找其他方案，但未果。

综合考虑官方给出的三种方案，如果不实用其提供的云数据库的话，第3种方案会是一个好的选择，**官方还提到了URL化，实测其实可以不需要。**

开发者配置

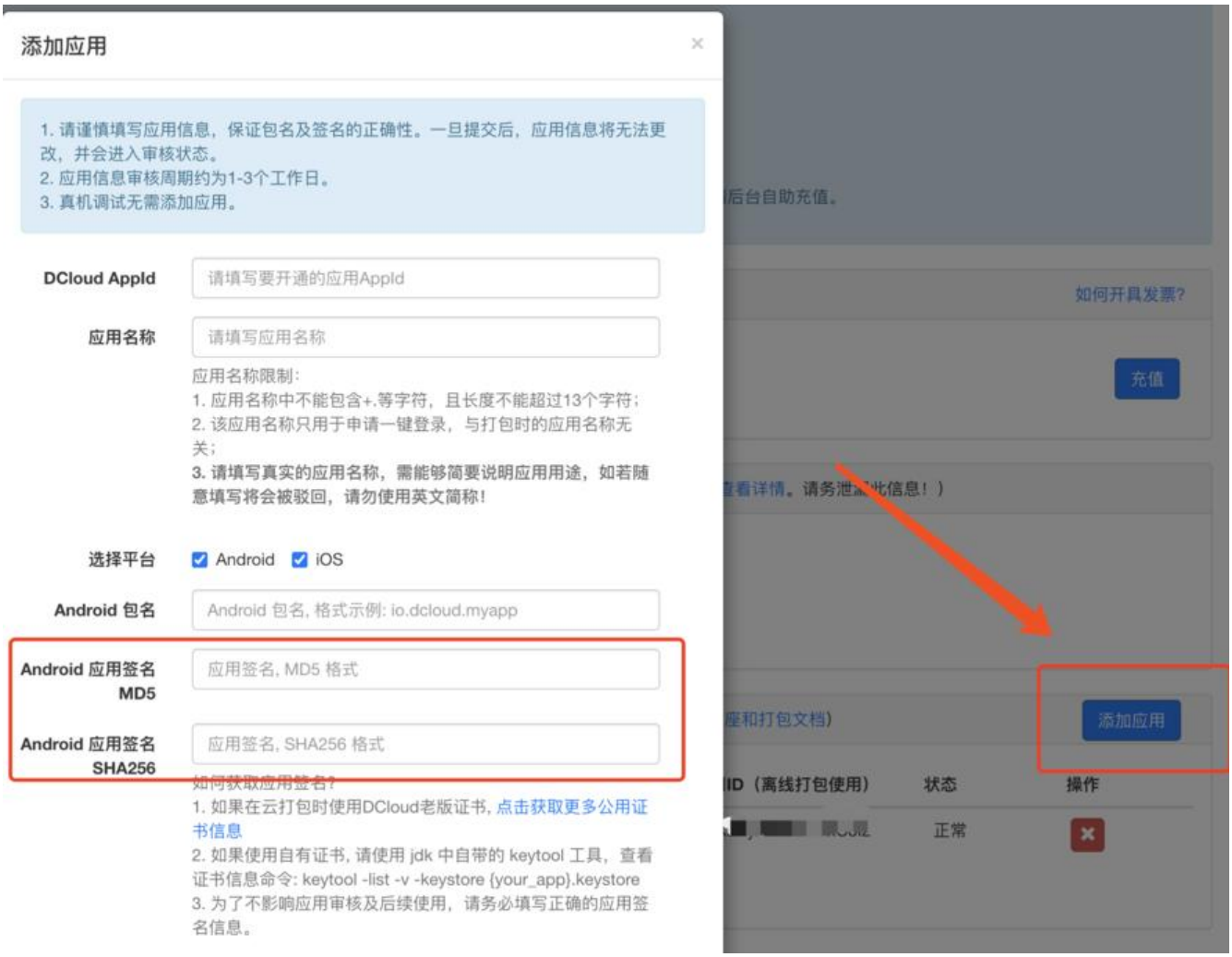
写云函数之前需要先进入 开发者中心 进行配置。

首先需要创建一个应用，如果你的应用已经创建则这部无需重复创建。

开通一键登录，开通后会生成 ApiKey 和 ApiSecret，后续在云开发时会使用到。

原文链接：uni-app 多端登录解决方案

顺便再添加应用，这部分是在打包后需要用到的，一般在生成签名时需要注意，可以参考 Android签证书生成 ，这个比较详细，按他的方案下来基本可以成功。



云函数配置及编写

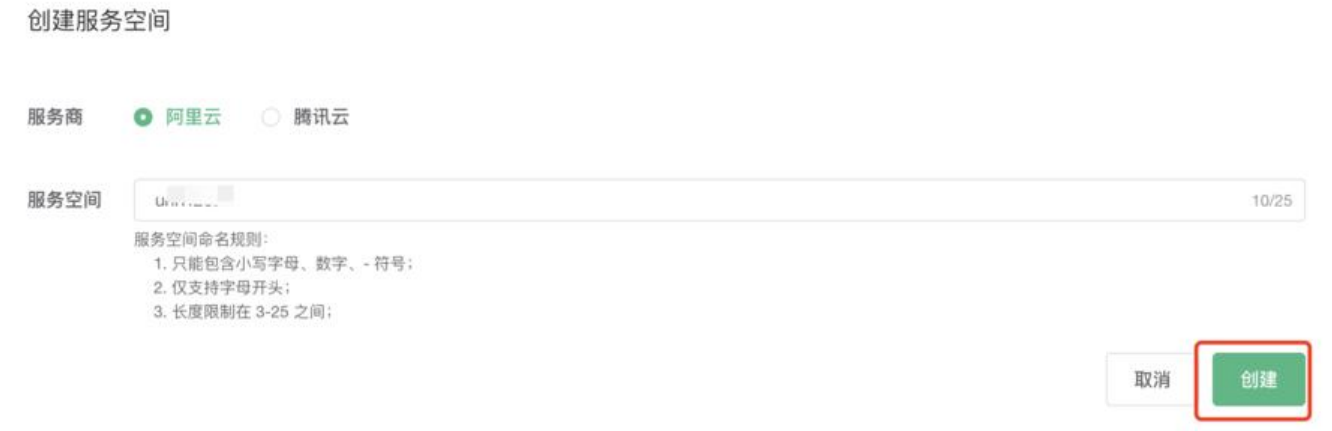
右键项目创建云开发环境

创建后再右键关联云服务空间

原文链接：uni-app 多端登录解决方案

点击新建

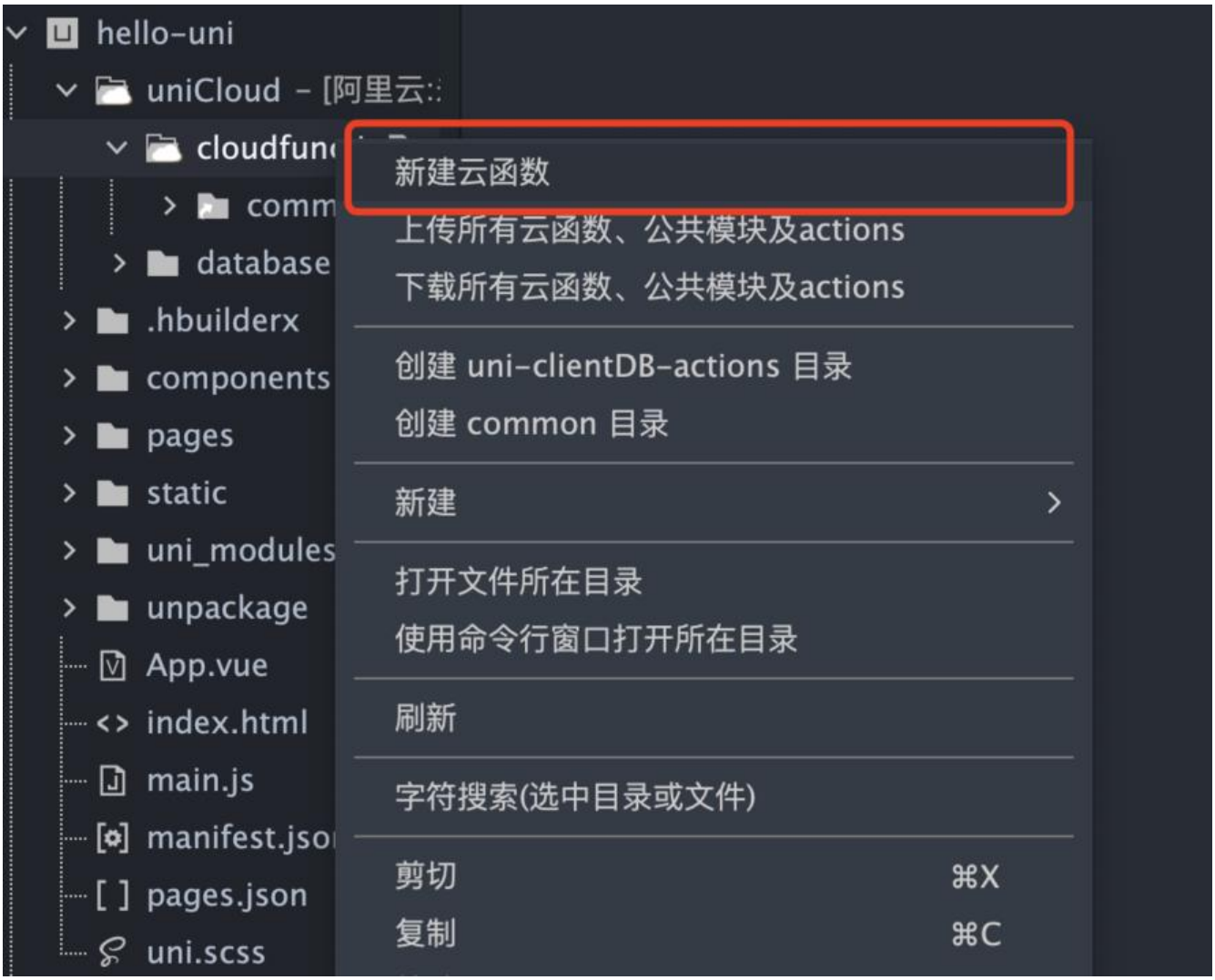
跳转到云服务空间后点击创建



返回到IDE中，选择刚刚创建的云空间并关联


关联云服务空间或项目
关联云服务空间
绑定其他项目的服务空间（适于多项目共用服务空间，详见）
新建
刷新
如何切换云服务商，详见
关联
取消

新建一个云函数



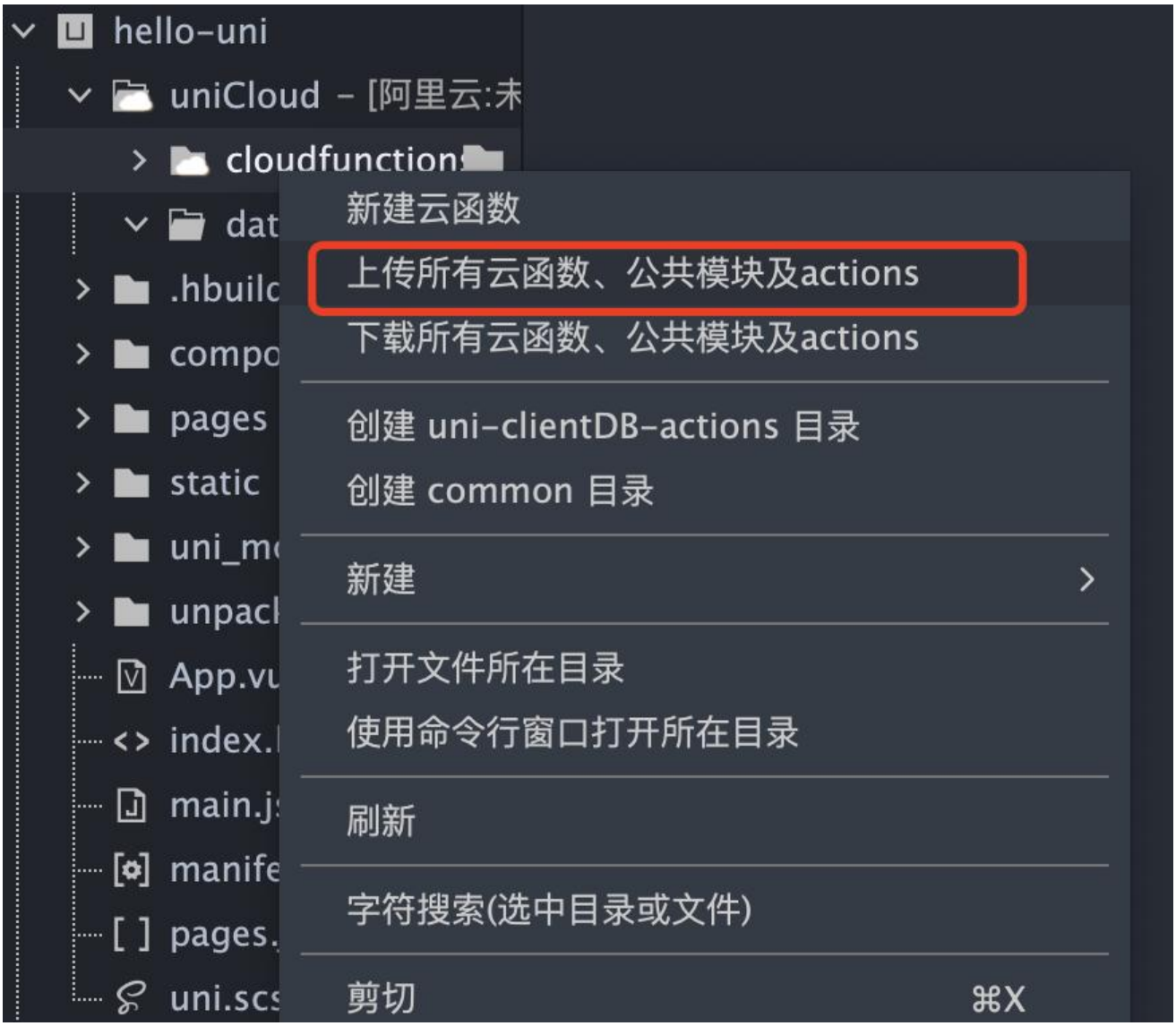
按如下代码编写一个换取手机号的云函数，此处没有按照官方的方式使用URL化，因为云函数是由服器后端接口去调用的，不对外暴露，无需再做其他复杂的操作。

修改 appid、apiKey、apiSecret。

```
'use strict';

exports.main = async (event) => {
    const params = event.queryStringParameters
    const {
        access_token,
        openid
    } = params
    const res = await uniCloud.getPhoneNumber({
        provider: 'univerify',
        appid: '__UNI__xxx',
        apiKey: 'xxx',
        apiSecret: 'xxx',
        access_token: access_token,
        openid: openid
    })
    return res
}
```

上传写好的云函数



进入云函数控制台，配置这个云函数的路径，自定义即可。

最后可以测试一下， uni.login() 后会打印出参数，之后将参数通过 PostMan 传递给你的云函数，果成功返回手机号，则可以继续下面的步骤了。

Java后端调用云函数

后端接受前端传来的，access_token 和 openid，之后通过后端去调用云函数后，最后可以将手机号 openid 进行用户信息入库。

入库后，后续的登录直接可以通过 openid 去查找用户，因为 openid 是唯一的，无需再调用云函数取手机号，毕竟换一次需要2分钱呢。

以Java为例，调用接口可以使用 RestTemplate、HttpClient或其他工具，笔者用的是一个开源工具 http-request 属实非常的方便。

```
@Component
public class UniCloud {
    @Value("${unicloud-url}")
    private String uniCloudUrl;

    /**
     * UniCloud 获取手机号
     */
    public String getPhoneNumber(String accessToken, String openid) {
        String url = this.uniCloudUrl + "auth?access_token=" + accessToken + "&openid=" + op
nid;
        String res = HttpRequest.get(url).body();
        Gson gson = new Gson();
        JsonObject jsonObject = gson.fromJson(res, JsonObject.class);
        if (jsonObject.has("success") && jsonObject.get("success").getAsBoolean()) {
            return jsonObject.get("phoneNumber").getAsString();
        }
        throw new UniCloudAPIError();
    }
}
```

最后还有一点需要注意，后面是使用小程序的方案登录，如果在通过上述 APP 的 openid 没有找到户时，还是需要换取一次手机号的，因为这个用户可能已经通过小程序注册了，并且在这一次请求中把 APP 的 openid 也入库。

小程序手机号登录

在结束了 APP 一键登录授权后，还需要实现一下小程序的手机号登录，因为一键登录，仅仅适用于 AP，不适用于小程序。

登录界面

类似的可以做一个和一键登录差不多的页面，此处按钮需要绑定获取手机号的事件。

```
<center>
    <img src="https://b3logfile.com/file/2022/04/IMG_2746-78207635.jpg" width="350" />
</center>
```

调用后端接口

此接口我还会传一个 openidCode，这个 openidCode 其实就是小程序 login() 后返回的 code，一传到后端可以获得用户的 openid。

```
export default {
    data() {
        return {
            appName: null,
            openidCode: null
        }
    },
    onShow() {
        uni.login({
            success: res => {
                this.openidCode = res.code
            }
        })
    },
    methods: {
        getphonenumber(e) {
            uni.showLoading({
                title: '正在授权中...'
            })
            const data = {
                'code': e.detail.code,
                'openid_code': this.openidCode
            }
            // 此处使用 uni.request 返送请求到后端，传的参数为 data
        }
    }
}
```

Java后端换取手机号和openid

小程序后端换取手机号和 openid 的方法比较多，这里推荐使用 WxJava 一个可以直接调用微信小程序、微信公众号等API接口的工具，也是相当好用。

```
@Component
public class WechatMp {
    private final WxMaService wxMaService;

    @Autowired
    public WechatMp(WxMaService wxMaService) {
        this.wxMaService = wxMaService;
    }

    /**
     * 获取手机号
     */
    public String getPhoneNumber(String code) {
        try {
            WxMaPhoneNumberInfo newPhoneNoInfo = this.wxMaService.getUserService().getN
wPhoneNoInfo(code);
            return newPhoneNoInfo.getPurePhoneNumber();
        } catch (WxErrorException e) {
            throw new WxAPIError();
        }
    }
```

```
    /**
     * 获取openid
     */
    public String getOpenid(String code) {
        try {
            return this.wxMaService.getUserService().getSessionInfo(code).getOpenid();
        } catch (WxErrorException e) {
            throw new WxAPIError();
        }
    }
}
```

最后，就写到这里吧，过程中还省略了前端发送请求，后端接受请求的方法，可以自行根据业务补充。