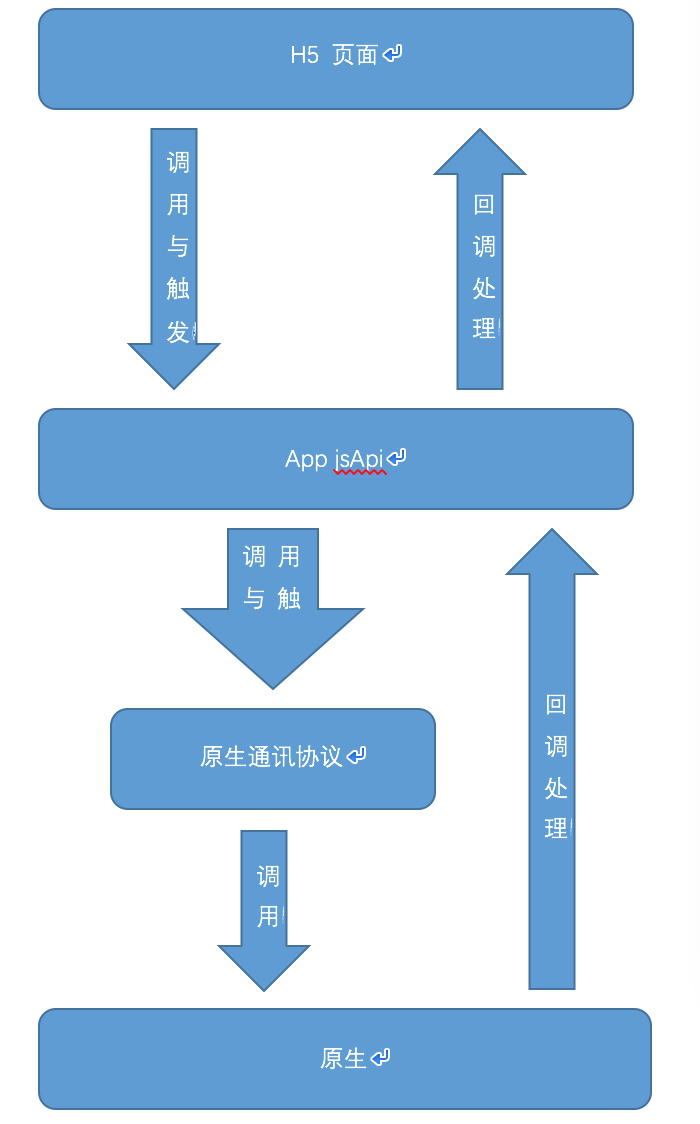
# 原理图

## H5 页面与多个App通讯

通过对各个App jsApi进一步封装成H5 jsApi。

## H5页面与单个App



# 通讯协议组装

## 原生通讯协议

①原生自定义伪通讯协议：定义成与 http 协议类似的格式：

协议名://协议类名: callbackTag/接口路径?参数1=XXX&参数2=XXX&参数3=XXX

其中：

a、协议名：app 自定义的协议名，用于H5触发行为的监控捕获， 如XYLAppJs://；

b、协议类名：原生具体协议类名；

b、回调方法标识：callbackTag回调方法标识；

b、接口路径：原生具体能力路径，不同原生能力路径不同；

c、H5传参（kv方式）：参数1=XXX&参数2=XXX&参数3=XXX

根据通讯协议规范，即可针对不同的原生能力给H5提供不同的调用地址，如：

XYLAppJs://class: h5MethodCallBackTag/method?a=2&b=3

②通讯行为——触发：能被原生监听并捕获截拦的H5行为，都可以作为原生通讯协议的触发行为。

统一使用：

location.href =' XYLAppJs://class: h5MethodCallBackTag/method?a=2&b=3 '

H5调用后，原生终端会捕获到内容：XYLAppJs://class: h5MethodCallBackTag/method?a=2&b=3

③通讯行为——调用：原生终端根据 H5 传过来的内容，解析匹配后会路由到具体处理方法，执行原生能力逻辑。

原生终端根据捕获到的协议内容，进行解析获取，若伪协议为原生指定的伪协议（“XYLAppJs”），就会根据 method 内容和 query 参数进行路由操作，寻找具体的方法执行逻辑。否则，忽略处理，按照 webview 原有跳转逻辑处理。

ios为例：

- (BOOL)webView:(UIWebView \*)webView shouldStartLoadWithRequest:(NSURLRequest \*)request navigationType:(UIWebViewNavigationType)navigationType

{

    NSURL \*url = request.URL; //url

    NSString \*scheme = url.scheme; //协议名

    NSString \*class = url.host; //协议类名

    NSString \*query = url.query; //参数

    NSNumber \*h5MethodCallBackTag = url.port; //回调编号

    NSString \*method = url.path;//接口路径

    if (url != nil && [scheme isEqualToString: @" XYLAppJs "]) {

        /\*根据method路由\*/

        NSArray \*items = @[@"/method", @"/other"];

        NSInteger item = [items indexOfObject:method];

        switch (item) {

            case 0:

                [self method];

                break;

            case 1:

                // 其它方法

                [self other];

                break;

            default:

                break;

        }

        return NO;

    } else {

        return YES;

    }

}

Android:

**public boolean** shouldOverrideUrlLoading(WebView view, String url) {  
 **if**(!TextUtils.*isEmpty*(url)){  
 Uri uri = Uri.*parse*(url);  
 String scheme = uri.getScheme();  
 String class = uri.getHost();  
 String query = uri.getQuery();  
 **if**(**"XYLAppJs"**.equals(scheme)){  
 */\*根据host, methmod路由\*/* }  
 }  
 **return true**;  
}

④通讯行为——回调：原生根据 H5 传过来的内容，捕获 js 回调函数方法标识，在原生逻辑执行结束后，将执行结果带到回调函数中并执行 js 回调函数。通过在第③步“调用”执行完后，原生会调用 js 回调函数 H5MethodCallBackTag,回调结果要求以字符串形式返回。

回调函数统一定为：H5MethodCallBack（H5MethodCallBackTag，resultData）

回调数据的格式json字符串

resultData = {

   code: 0,//0:成功，1:失败

   msg: '请求超时'//失败时候的提示，成功可为空

   body: {}//数据，无数据可以为空

};

ios为例：

/\*解析到H5的回调函数标识为h5MethodCallBackTag，回调执行js的方法\*/

webview.stringByEvaluatingJavaScriptFromString("H5MethodCallBack (‘h5MethodCallBackTag’，‘resultData)") 。

Android:

webView.loadUrl("javascript: h5MethodCallBack ('" + h5MethodCallBackTag + "'," + resultData + ")";

)

## App jsApi通讯协议

对各个不同平台的原生通讯协议的差异性进行封装，组合成App jsApi通讯协议。jsApi 完成三大行为：api 接口创建、协议 url 组装、创建 iframe 发起伪协议请求；

①api接口创建：js 函数接口封装、平台差异处理，方便H5函数调用。

比如：

method.build(' method ', {

    iOS: function(options, callback) {

        var callbackName = callback ? method.callback(callback) : null;

       method.invokeClient(' m1, m2, {

            'params': options,

            'callback': callbackName

        });

    },

    android: function(params, callback) {

       method.invokeClient(m1, m2, JSON.stringify(params), callback);

    },

    supportInvoke: true,

    support: {

        iOS: '10.3.1',

        android: '4.6.1'

    }

});

method.build 方法为 api 接口创建方法。通过传入待创建的 jsapi 方法名（method.m1.m2）和不同平台(android/ios)的差异处理配置。最终会生成H5可调用方法：

method ({

    /\*data\*/

},function(ret){

    /\*callback\*/

})

②协议url组装：从接口到 url 协议的转换、回调处理，完成协议 url 创建

第①步中，不同平台差异处理都会调用 method.invokeClient 方法，该方法实际处理的就是原理图中与原生通讯的过程。协议 url 组装的过程如下：

/\*生成回调索引\*/

tg = storeCallback(callback);

/\*协议路径组装\*/

url = 'XYLAppJs://' + encodeURIComponent(class) + ':' + tg+ '/' + encodeURIComponent(method);

/\*参数组装\*/

argus.forEach(function(a, i) {

    if (exports.isObject(a)) {

    a = JSON.stringify(a);

    }

    if (i === 0) {

        url += '?p=';

    } else {

        url += '&p' + i + '=';

    }

    url += encodeURIComponent(String(a));

});

/\*链接调用\*/

result = openURL(url, class, method);

协议 url 组装的过程实际上是对传入参数按协议规范进行拼串的过程，其中包括匿名回调函数的回调索引创建、协议名&协议路径拼串、传参循环遍历拼串。

③创建 iframe 发起伪协议请求：请求触发

/\*创建隐藏iframe\*/

var iframe = document.createElement('iframe');

iframe.style.cssText = 'display:none;width:0px;height:0px;';

function failCallback() {

    /\*错误处理\*/

}

/\*iframe协议调用\*/

iframe.onload = failCallback;

iframe.src = url;

(document.body || document.documentElement).appendChild(iframe);

/\*删除iframe\*/

setTimeout(function() {

    iframe && iframe.parentNode && iframe.parentNode.removeChild(iframe);

}, 0);

通过创建 iframe 来完成协议调用，并在调用结束后将 iframe 删除，即可在不影响原 H5 流程的情况下完成调用全过程。

## H5 jsApi通讯协议

对各个不同的App的jsApi的通讯协议的差异性进行封装，组合成H5 jsApi通讯协议。H5 jsapi 完成两大行为：app jsApi 差异请求、app jsApi 差异封装。

①app jsApi 差异请求：根据当前运行环境 App 请求具体的 App jsApi

下面以 Hybrid H5 需同时运行在App1和App2 app jsApi 差异请求处理逻辑。

<script type="text/javascript" >

    (function() {

        var ua = navigator.userAgent || "",

            isApp1 = ua.match(/App1\/([\d\.]+)/),

            isApp2 = ua.match("App2");

        if (isApp1) {

            document.write("<script src=' App1JsApi.js'><\x2Fscript>");

        } else if (isApp2) {

            document.write("<script src=' App2JsApi.js '><\x2Fscript>");

        } else {

            // 不是已兼容app，跳转到兼容app上运行

            var currentHref = window.location.href;

            /\*跳转到App1打开本页面\*/

            window.location.href = ' App1Api://forward/url?url\_prefix=' + btoa(currentHref) + '&version=1&src\_type=web';

            }, 0)

        }

    })()

</script>

除了对需兼容的 App 进行差异请求外，还应对在不兼容的 App 运行时做跳转到主兼容 App 打开当前页面的逻辑处理，并做引导性提示，保障页面的完整可用性。

②App jsApi 差异封装：根据当前具体运行的平台调用相应的 App jsApi 接口。H5 jsApi 的接口形式尽量与主运行 App 的 jsApi 保持一致。