Android WebView与JS交互

1.交互方式总结

Android与JS通过WebView相互调用方法，实际上是：

- Android去调用JS的代码
- JS去调用Android的代码

两者的沟通桥梁是WebView

对于Android调用JS代码的方法有2种：

1. 通过`WebView`的`loadUrl()`
2. 通过`WebView`的`evaluateJavascript()`

对于JS调用Android代码的方法有3种：

1. 通过`WebView`的`addJavascriptInterface()`进行对象映射
2. 通过`WebViewClient`的`shouldOverrideUrlLoading()`方法回调拦截url
3. 通过`WebChromeClient`的`onJsAlert()`、`onJsConfirm()`、`onJsPrompt()`方法回调拦截JS对话框`alert()`、`confirm()`、`prompt()`消息

2.具体分析

2.1 Android通过WebView调用JS代码

对于Android调用JS代码的方法有2种：

1. 通过`WebView`的`loadUrl()`
2. 通过`WebView`的`evaluateJavascript()`

2.1.1 方法分析

方法1:通过`WebView`的`loadUrl()`

- 示例介绍：点击Android按钮，即调用WebView JS(文本名为javascript)中`callJS()`
- 具体使用：
  步骤1：将需要调用的JS代码以.html格式放到src/main/assets文件夹里
  注：
  1. 为了方便展示，本文是采用Android调用本地JS代码说明；
  2. 实际情况时，Android更多的是调用远程JS代码，即将加载的JS代码路径改成url即可；

  需要加载JS代码：javascript.html

```
// 文本名：javascript
<!DOCTYPE html>
<html>

  <head>
```

```
    <meta charset="utf-8">
    <title>Carson_Ho</title>

// JS代码
    <script>
// Android需要调用的方法
  function callJS(){
     alert("Android调用了JS的callJS方法");
  }
</script>

  </head>

</html>
```

步骤2：在Android里通过WebView设置调用JS代码

Android代码：MainActivity.java

```
public class MainActivity extends AppCompatActivity {

   WebView mWebView;
   Button button;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mWebView =(WebView) findViewById(R.id.webview);

       WebSettings webSettings = mWebView.getSettings();

       // 设置与Js交互的权限
       webSettings.setJavaScriptEnabled(true);
       // 设置允许JS弹窗
       webSettings.setJavaScriptCanOpenWindowsAutomatically(true);

       // 先载入JS代码
       // 格式规定为:file:///android_asset/文件名.html
       mWebView.loadUrl("file:///android_asset/javascript.html");

       button = (Button) findViewById(R.id.button);


       button.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               // 通过Handler发送消息
               mWebView.post(new Runnable() {
                   @Override
                   public void run() {

                       // 注意调用的JS方法名要对应上
                       // 调用javascript的callJS()方法
                       mWebView.loadUrl("javascript:callJS()");
                   }
               });

           }
       });

       // 由于设置了弹窗检验调用结果,所以需要支持js对话框
       // webview只是载体，内容的渲染需要使用webviewChromClient类去实现
       // 通过设置WebChromeClient对象处理JavaScript的对话框
       //设置响应js 的Alert()函数
       mWebView.setWebChromeClient(new WebChromeClient() {
           @Override
```

```
            public boolean onJsAlert(WebView view, String url, String message, final JsResult result) {
                AlertDialog.Builder b = new AlertDialog.Builder(MainActivity.this);
                b.setTitle("Alert");
                b.setMessage(message);
                b.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        result.confirm();
                    }
                });
                b.setCancelable(false);
                b.create().show();
                return true;
            }

        });


    }
}

```

特别注意：JS代码调用一定要在`onPageFinished()`回调之后才能调用，否则不会调用。
注：`onPageFinished()`属于WebViewClient类的方法，主要在页面加载结束时调用

方法2:通过`WebView`的`evaluateJavascript()`

- 优点：该方法比第一种方法效率更高、使用更简洁。
    1. 因为该方法的执行不会使页面刷新，而第一种方法（loadUrl）的执行则会；
    2. Android 4.4后才可使用

- 具体使用

```
// 只需要将第一种方法的loadUrl()换成下面该方法即可
    mWebView.evaluateJavascript ("javascript:callJS()", new ValueCallback<String>() {
        @Override
        public void onReceiveValue(String value) {
            //此处为 js 返回的结果
        }
    });
}
```

2.1.2 方法对比

调用方式	优点	缺点	使用
使用loadUrl（）	方便简洁	效率低;获取返回值麻烦	不需要获取返回值，
使用evaluateJavascript()	效率高	向下兼容性差（仅Android4.4以上可用）	Android

2.1.3 使用建议

两种方法混合使用，即Android 4.4以下使用方法1.Android 4.4以上使用方法2

```
// Android版本变量
final int version = Build.VERSION.SDK_INT;
// 因为该方法在 Android 4.4 版本才可使用，所以使用时需进行版本判断
if (version < 18) {
    mWebView.loadUrl("javascript:callJS()");

```

```
} else {
    mWebView.evaluateJavascript ("javascript:callJS()", new ValueCallback<String>() {
        @Override
        public void onReceiveValue(String value) {
            //此处为 js 返回的结果
        }
    });
}
```

2.2 JS通过WebView调用Android代码

对于JS调用Android代码的方法有3种

1. 通过 `WebView` 的 `addJavascriptInterface()` 进行对象映射
2. 通过 `WebViewClient` 的 `shouldOverrideUrlLoading()` 方法回调拦截url
3. 通过 `WebChromeClient` 的 `onJsAlert()` 、 `onJsConfirm()` 、 `onJsPrompt()` 方法回调拦截JS对话框 `alert()` 、 `confirm()` 、 `prompt()` 消息

2.2.1方法分析

方式1：通过 `WebView` 的 `addJavascriptInterface()` 进行对象映射
步骤1:定义一个与JS对象映射关系的Android类：AndroidtoJs
AndroidtoJs.java

```
// 继承自Object类
public class AndroidtoJs extends Object {

    // 定义JS需要调用的方法
    // 被JS调用的方法必须加入@JavascriptInterface注解
    @JavascriptInterface
    public void hello(String msg) {
        System.out.println("JS调用了Android的hello方法");
    }
}
```

步骤2：将需要调用的JS代码以 `.html` 格式放到src/main/assets文件夹里
需要加载JS代码：javascript.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Carson</title>
      <script>


         function callAndroid(){
        // 由于对象映射，所以调用test对象等于调用Android映射的对象
            test.hello("js调用了android中的hello方法");
         }
      </script>
   </head>
   <body>
      //点击按钮则调用callAndroid函数
      <button type="button" id="button1" onclick="callAndroid()"></button>
   </body>
</html>
```

步骤3：在Android里通过WebView设置Android类与JS代码的映射

```
public class MainActivity extends AppCompatActivity {

    WebView mWebView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mWebView = (WebView) findViewById(R.id.webview);
        WebSettings webSettings = mWebView.getSettings();

        // 设置与Js交互的权限
        webSettings.setJavaScriptEnabled(true);

        // 通过addJavascriptInterface()将Java对象映射到JS对象
        //参数1：Javascript对象名
        //参数2：Java对象名
        mWebView.addJavascriptInterface(new AndroidtoJs(), "test");//AndroidtoJS类对象映射到js的test对象

        // 加载JS代码
        // 格式规定为:file:///android_asset/文件名.html
        mWebView.loadUrl("file:///android_asset/javascript.html");

```

特点

- 优点：使用简单（仅将Android对象和JS对象映射即可）
- 缺点：存在严重的漏洞问题

方式2：通过 `WebViewClient` 的方法 `shouldOverrideUrlLoading()` 回调拦截url

- 具体原理：
    1. Android通过 `WebViewClient` 的回调方法 `shouldOverrideUrlLoading()` 拦截url
    2. 解析该url的协议
    3. 如果检测到是预先约定好的协议，就调用相应方法
       即JS需要调用Android的方法

- 具体使用
  步骤1：在JS约定所需要的url协议
  JS代码：JavaScript.html

```
<!DOCTYPE html>
<html>

  <head>
     <meta charset="utf-8">
     <title>Carson_Ho</title>

    <script>
        function callAndroid(){
           /*约定的url协议为：js://webview?arg1=111&arg2=222*/
           document.location = "js://webview?arg1=111&arg2=222";
        }
    </script>
</head>

<!-- 点击按钮则调用callAndroid（）方法  -->
  <body>
    <button type="button" id="button1" onclick="callAndroid()">点击调用Android代码</button>
  </body>
</html>
```

当该JS通过Android的`WebView.loadUrl("file:///android_asset/javascript.html")`加载后，就会回调`shouldOverrideUrlLoading()`，接下来继续看步骤2：

步骤2：在Android通过WebViewClient复写`shouldOverrideUrlLoading()`

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    WebView mWebView;
//    Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mWebView = (WebView) findViewById(R.id.webview);

        WebSettings webSettings = mWebView.getSettings();

        // 设置与Js交互的权限
        webSettings.setJavaScriptEnabled(true);
        // 设置允许JS弹窗
        webSettings.setJavaScriptCanOpenWindowsAutomatically(true);

        // 步骤1：加载JS代码
        // 格式规定为:file:///android_asset/文件名.html
        mWebView.loadUrl("file:///android_asset/javascript.html");


// 复写WebViewClient类的shouldOverrideUrlLoading方法
mWebView.setWebViewClient(new WebViewClient() {
                                    @Override
                                    public boolean shouldOverrideUrlLoading(WebView view, String url) {

                                        // 步骤2：根据协议的参数，判断是否是所需要的url
                                        // 一般根据scheme（协议格式） & authority（协议名）判断（前两个参数）
                                        //假定传入进来的 url = "js://webview?arg1=111&arg2=222"（同时也是约定好的需要拦截的）

                                        Uri uri = Uri.parse(url);
                                        // 如果url的协议 = 预先约定的 js 协议
                                        // 就解析往下解析参数
                                        if ( uri.getScheme().equals("js")) {

                                            // 如果 authority  = 预先约定协议里的 webview，即代表都符合约定的协议
                                            // 所以拦截url,下面JS开始调用Android需要的方法
                                            if (uri.getAuthority().equals("webview")) {

                                               //  步骤3：
                                                // 执行JS所需要调用的逻辑
                                                System.out.println("js调用了Android的方法");
                                                // 可以在协议上带有参数并传递到Android上
                                                HashMap<String, String> params = new HashMap<>();
                                                Set<String> collection = uri.getQueryParameterNames();


                                            }

                                            return true;
                                        }
                                        return super.shouldOverrideUrlLoading(view, url);
                                    }
                                }
        );
    }
        }
```

特点

- 优点：不存在方式1的漏洞
- 缺点：JS获取Android方法的返回值复杂
  如果JS想要得到Android方法的返回值，只能通过WebView的`loadUrl()`去执行JS方法把返回值传递回去，相关的代码如下：

```
  // Android：MainActivity.java
mWebView.loadUrl("javascript:returnResult(" + result + ")");

// JS：javascript.html
function returnResult(result){
    alert("result is" + result);
}
```

方式3：通过`WebChromeClient`的`onJsAlert()`、`onJsConfirm()`、`onJspPrompt()`方法回调拦截JS对话框`alert()`、`confirm()`、`prompt()`消息
在JS中，有三种常用的对话框方法：

方法	作用	返回值	备注
alter()	弹出警告框	没有	在文本加入"\n"可换行
confirm()	弹出确认框	两个返回值	返回布尔值；通过该值可判断点击时确认还是取消：true表示点击确认，false表示
prompt()	弹出输入框	任意设置返回值	点击“确认”:返回输入框中的值；点击“取消”：返回null

方式3的原理：Android通过`WebChromeClient`的`onJSAlert（）`、`onJsConfirm()`、`onJsPrompt()`方法回调分别拦截JS对话框（即上述三个方法），得到它们的消息内容，然后解析即可。
下面的例子将用拦截JS的输入框（即`prompt()`方法）说明：

1. 常用的拦截是：拦截JS的输入框（即`prompt()`方法）
2. 因为只有`prompt()`可以返回任意类型的值，操作最全面方便、更加灵活；而`alert()`对话框没有返回值；`confirm()`对话框只能返回两种状态（确定\取消）两个值

步骤1：加载JS代码，如下：
javascript.html(以.html格式放到src/main/assets文件夹里)

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Carson_Ho</title>

     <script>

    function clickprompt(){
    // 调用prompt（）
    var result=prompt("js://demo?arg1=111&arg2=222");
    alert("demo " + result);
}

      </script>
</head>

<!-- 点击按钮则调用clickprompt()  -->
   <body>
     <button type="button" id="button1" onclick="clickprompt()">点击调用Android代码</button>
   </body>
</html>
```

当使用`mWebView.loadUrl("file:///android_asset/javascript.html")`加载了上述JS代码后，就会触发回调`onJsPrompt()`,具体如下：

1. 如果是拦截警告框（即`alter()`）,则触发回调`onJsAlert()`；
2. 如果是拦截确认框（即`confirm()`），则触发回调`onJsconfirm()`；

步骤2：在Android通过`WebChromeClient`复写`onJsPrompt()`

```
public class MainActivity extends AppCompatActivity {

    WebView mWebView;
//    Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mWebView = (WebView) findViewById(R.id.webview);

        WebSettings webSettings = mWebView.getSettings();

        // 设置与Js交互的权限
        webSettings.setJavaScriptEnabled(true);
        // 设置允许JS弹窗
        webSettings.setJavaScriptCanOpenWindowsAutomatically(true);

// 先加载JS代码
        // 格式规定为:file:///android_asset/文件名.html
        mWebView.loadUrl("file:///android_asset/javascript.html");


        mWebView.setWebChromeClient(new WebChromeClient() {
                                    // 拦截输入框(原理同方式2)
                                    // 参数message:代表promt（）的内容（不是url）
                                    // 参数result:代表输入框的返回值
                                    @Override
                                    public boolean onJsPrompt(WebView view, String url, String message, String defaultVal
                                        // 根据协议的参数，判断是否是所需要的url(原理同方式2)
                                        // 一般根据scheme（协议格式） & authority（协议名）判断（前两个参数）
                                        //假定传入进来的 url = "js://webview?arg1=111&arg2=222"（同时也是约定好的需要拦截的）

                                        Uri uri = Uri.parse(message);
                                        // 如果url的协议 = 预先约定的 js 协议
                                        // 就解析往下解析参数
                                        if ( uri.getScheme().equals("js")) {

                                            // 如果 authority  = 预先约定协议里的 webview，即代表都符合约定的协议
                                            // 所以拦截url,下面JS开始调用Android需要的方法
                                            if (uri.getAuthority().equals("webview")) {

                                                //
                                                // 执行JS所需要调用的逻辑
                                                System.out.println("js调用了Android的方法");
                                                // 可以在协议上带有参数并传递到Android上
                                                HashMap<String, String> params = new HashMap<>();
                                                Set<String> collection = uri.getQueryParameterNames();

                                                //参数result:代表消息框的返回值(输入值)
                                                result.confirm("js调用了Android的方法成功啦");
                                            }
                                            return true;
                                        }
                                        return super.onJsPrompt(view, url, message, defaultValue, result);
                                    }

// 通过alert()和confirm()拦截的原理相同，此处不作过多讲述

                                    // 拦截JS的警告框
                                    @Override
                                    public boolean onJsAlert(WebView view, String url, String message, JsResult result) {
                                        return super.onJsAlert(view, url, message, result);
                                    }

                                    // 拦截JS的确认框
                                    @Override
                                    public boolean onJsConfirm(WebView view, String url, String message, JsResult result)
```

```
                return super.onJsConfirm(view, url, message, result);
            }
        }
    );


    }

}
```

2.2.2 三种方式的对比&使用场景

调用方式	优点	缺点	
通过addJavascriptInface()进行添加对象映射	方便简洁	Android4.2以下存在漏洞问题	Androi
通过WebViewClient的方法shouldOverrideURLLoading（ ）回调拦截url	不存在漏洞问题	使用复杂：需要进行协议的约束；从Native层往Web层传递值比较繁琐	不需要返回
通过WebChromeClient的onJsAlert()、onJsConfirm()、onJsPrompt()方法回调拦截JS对话框消息	不存在漏洞问题	使用负责：需要进行协议的约束	能满足

3.总结

- 本文主要对Android通过WebView与JS的监护方式进行了全面介绍

类型	调用方式	优点	缺点	使用场景	使用建议
Android 调用 JS	loadUrl（）	方便简洁	效率低、获取返回值麻烦	不需要获取返回值，对性能要求较低时	混合使用，即： Android 4.4以下 用方法1 Android 4.4以上 用方法2
	evaluateJavascript（）	效率高	向下兼容性差 (仅Android 4.4以上可用)	Android 4.4以上	
JS 调用 Android	通过addJavascriptInterface（）进行添加对象映射	方便简洁	Android 4.2以下存在漏洞问题	Android 4.2以上相对简单互调场景	/
	通过 WebViewClient.shouldOverrideUrlLoading ()回调拦截 url	不存在漏洞问题	使用复杂：需要进行协议的约束；从 Native 层往 Web 层传递值比较繁琐	不需要返回值情况下的互调场景 (iOS主要使用该方式)	
	通过 WebChromeClient的onJsAlert()、onJsConfirm()、onJsPrompt（）方法回调拦截JS对话框消息	不存在漏洞问题	使用复杂：需要进行协议的约束；	能满足大多数情况下的互调场景	

代码地址