前言

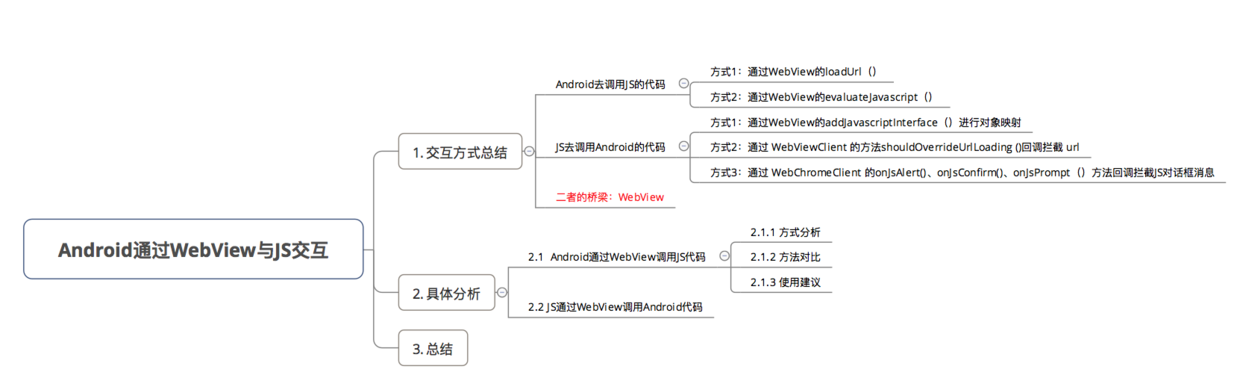
* 现在很多App里都内置了Web网页（Hyprid App），比如说很多电商平台，淘宝、京东、聚划算等等，如下图



* 上述功能是由Android的WebView实现的，其中涉及到Android客户端与Web网页交互的实现
* 今天我将全面介绍**Android通过WebView与JS交互**的全面方式

阅读本文前请先阅读：[Android开发：最全面、最易懂的Webview详解](http://blog.csdn.net/carson_ho/article/details/52693322" \t "_blank)

目录



1. 交互方式总结

Android与JS通过WebView互相调用方法，实际上是：

* Android去调用JS的代码
* JS去调用Android的代码

二者沟通的桥梁是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（）

方式1：通过WebView的loadUrl（）

* 实例介绍：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**

1. 为了方便展示，本文是采用Andorid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>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

**步骤2：在Android里通过WebView设置调用JS代码**

*Android代码：MainActivity.java*

注释已经非常清楚

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) {

// 必须另开线程进行JS方法调用(否则无法调用)

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;

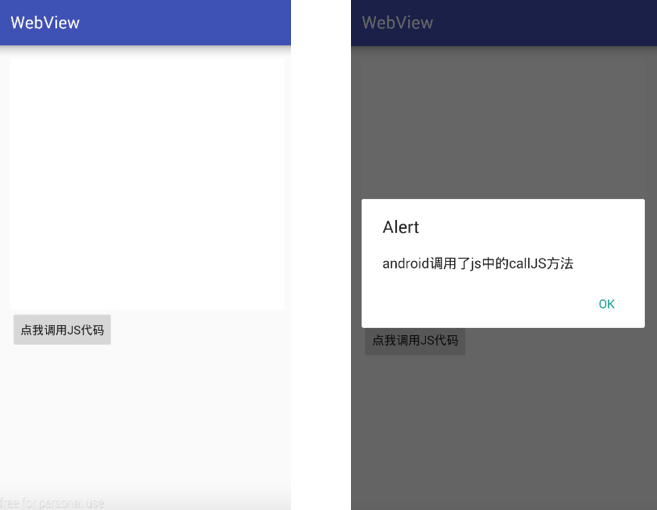
}

});

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69



**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。**

onPageFinished()属于WebViewClient类的方法，主要在页面加载结束时调用

方式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

}

});

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

2.1.2 方法对比



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

}

});

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

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*（注释已经非常清楚）

// 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");

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

**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>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

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

详细请看注释

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

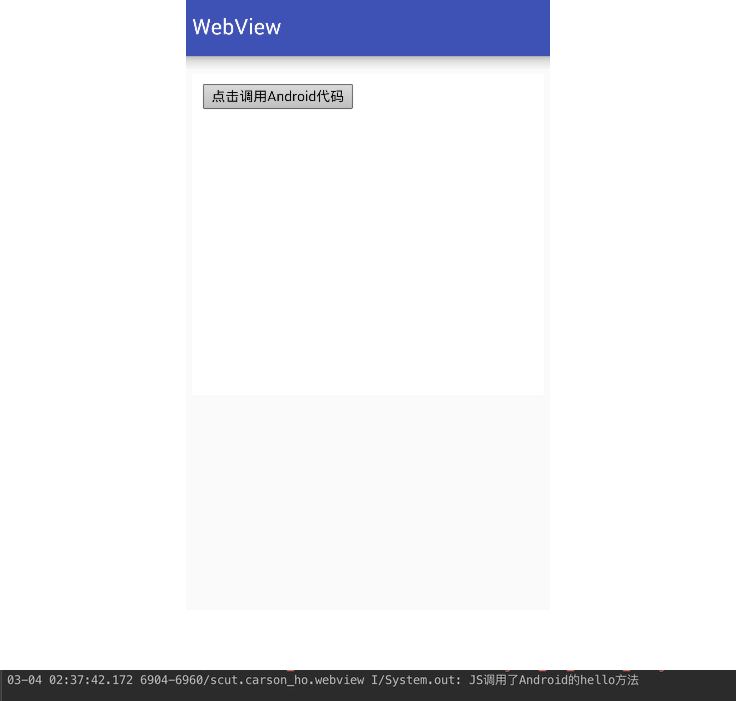
mWebView.addJavascriptInterface(new AndroidtoJs(), "test");//AndroidtoJS类对象映射到js的test对象

// 加载JS代码

// 格式规定为:file:///android\_asset/文件名.html

mWebView.loadUrl("file:///android\_asset/javascript.html");

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24



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，具体请看文章：[你不知道的 Android WebView 使用漏洞](http://blog.csdn.net/carson_ho/article/details/64904635)

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*

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 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>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

当该JS通过Android的mWebView.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);

}

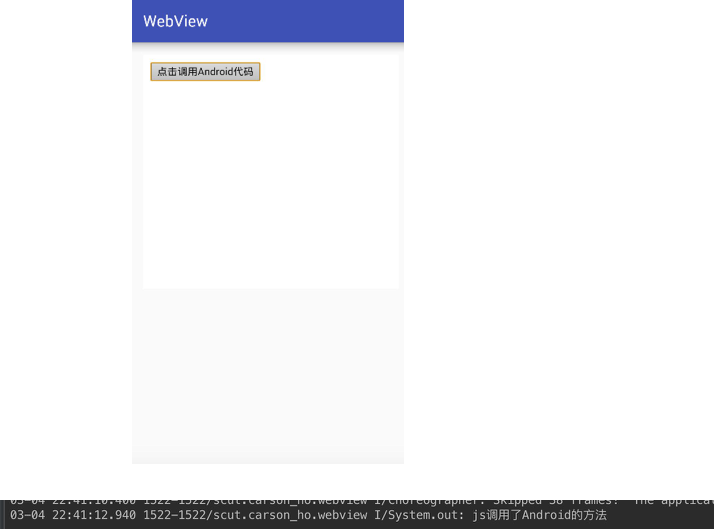
}

);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60



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);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

方式3：通过 WebChromeClient的onJsAlert()、onJsConfirm()、onJsPrompt（）方法回调拦截JS对话框alert()、confirm()、prompt（） 消息

在JS中，有三个常用的对话框方法：



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），得到他们的消息内容，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>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22

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alert()），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 defaultValue, JsPromptResult result) {

// 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);

}

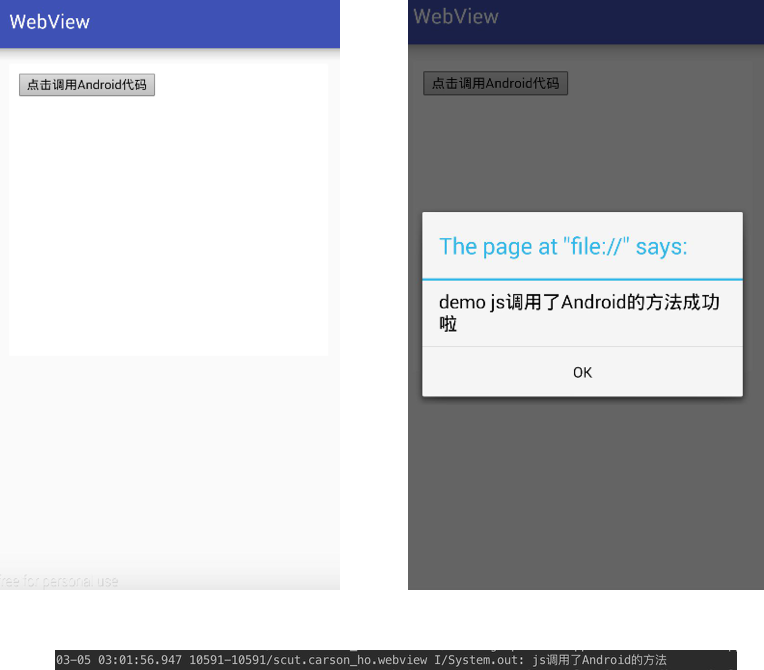
}

);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78



2.2.2 三种方式的对比 & 使用场景

