Android与H5交互

参考：<https://www.jb51.net/article/100758.htm>

<https://blog.csdn.net/qq_24530405/article/details/52067474>

<https://blog.csdn.net/carson_ho/article/details/52693322>

<https://www.cnblogs.com/lee0oo0/p/4026774.html>

Android实现H5和Native交互的两种方法：一种是拦截Url（设置WebViewClient并重写shouldOverrideUrlLoading方法）；另一种是JavaScript注入（api 17之前有漏洞）

**WebView**

1. 获取到webview之后，设置webview属性  
   private void settingProperty(){

myJavascriptInterface = new MyJavascriptInterface();

WebSettings webSettings = mWebView.getSettings();

//支持js

webSettings.setJavaScriptEnabled(true);

//支持通过JS打开新窗口

webSettings.setJavaScriptCanOpenWindowsAutomatically(true);

//缩放至屏幕的大小

webSettings.setLoadWithOverviewMode(true);

//将图片调整到适合webview的大小

webSettings.setUseWideViewPort(false);

//设置缓存方式

webSettings.setCacheMode(WebSettings.LOAD\_DEFAULT);

//开启DOM storage API功能

webSettings.setDomStorageEnabled(true);

//开启 database storage API 功能

webSettings.setDatabaseEnabled(true);

String cacheDirPath = getCacheDir().getAbsolutePath()+ "/webViewCache";

//设置数据库缓存路径

webSettings.setDatabasePath(cacheDirPath);

//开启Application H5 Caches 功能

webSettings.setAppCacheEnabled(true);

//设置Application Caches 缓存目录

webSettings.setAppCachePath(cacheDirPath);

/\*\*对系统API在19以上的版本作了兼容。因为4.4以上系统onPageFinished在恢复图片加载时, 如果存在多张图片引用的是相同的src时，会只有一个image标签得到加载，因而对于这样的系统我们就先直接加载 \*/

if (Build.VERSION.SDK\_INT >= 19){

webSettings.setLoadsImagesAutomatically(true);

} else {

webSettings.setLoadsImagesAutomatically(false);

}

}

1. 重写WebViewClient处理webview的各种通知与请求事件

public class MyWebViewClient extends WebViewClient {  
 */\*\*  
 \** ***@return*** *true：打开新的url（重定向）时，webview不会加载该url，需要自己处理；false：系统继续加载该url；  
 \*/* @Override  
 public boolean shouldOverrideUrlLoading(WebView view, String url) {  
 //*TODO 重写此方法，实现对网页中超链接的拦截* view.loadUrl(url);  
 return true;  
 }  
 // 网页开始加载  
 @Override  
 public void onPageStarted(WebView view, String url, Bitmap favicon) {  
 super.onPageStarted(view, url, favicon);  
 }  
 // 网页加载完成  
 @Override  
 public void onPageFinished(WebView view, String url) {  
 super.onPageFinished(view, url);  
 }  
 // 网页加载错误  
 @Override  
 public void onReceivedError(WebView view, int errorCode, String description, String failingUrl) {  
 super.onReceivedError(view, errorCode, description, failingUrl);  
 }  
}

1. 若没有设置 WebViewClient ，url重定向时则由系统（Activity Manager）处理该 url，通常是使用浏览器打开或弹出浏览器选择对话框。
2. 若设置 WebViewClient 且shouldOverrideUrlLoading返回 true ，url重定向时应由应用的代码处理该 url，WebView 不处理，也就是程序员自己做处理。
3. 若设置 WebViewClient 且shouldOverrideUrlLoading返回 false，url重定向时则说明由 WebView 处理该 url，即用 WebView 加载该 url。

注：url不需要重定向时，没有设置 WebViewClient也可以在WebView中打开，且与shouldOverrideUrlLoading返回值无关。

1. 重写WebChromeClient，辅助webview处理javascript对话框，网页的icon和title，以及加载进度条等。

*/\*\*  
 \* 重写webview的浏览器客户端  
 \* 主要功能是辅助webview处理javascript对话框，网页的icon和title，以及加载进度条等  
 \*/*public class MyWebChromeClient extends WebChromeClient {  
 // 监听页面加载进度  
 @Override  
 public void onProgressChanged(WebView view, int newProgress) {  
 super.onProgressChanged(view, newProgress);  
 }  
 // 监听网页标题  
 @Override  
 public void onReceivedTitle(WebView view, String title) {  
 super.onReceivedTitle(view, title);  
 //*TODO 接收html的title标签，可根据html的title改变而改变* }  
 // 监听网页图标  
 @Override  
 public void onReceivedIcon(WebView view, Bitmap icon) {  
 super.onReceivedIcon(view, icon);  
 }  
 @Override  
 public boolean onJsAlert(WebView view, String url, String message, JsResult result) {  
 return super.onJsAlert(view, url, message, result);  
 }  
 @Override  
 public boolean onJsConfirm(WebView view, String url, String message, JsResult result) {  
 return super.onJsConfirm(view, url, message, result);  
 }  
 @Override  
 public boolean onJsPrompt(WebView view, String url, String message, String defaultValue, JsPromptResult result) {  
 return s

}  
}

1. Js调用Android native方法：
2. 提供接口供js调用(@JavascriptInterface注解标注该方法是供js调用) （同步调用）

WebView.addJavascriptInterface(new MyJavascriptInterface(), “android”); // “android”是为js接口取的名字

public class MyJavascriptInterface{

@JavascriptInterface

public String h5NativtCallback(){

LogUtils.i(TAG, "h5NativeCallback");

return "js 调用 android 的回调 "+(num\_android++);

}

}

注：H5调用android native方法：window.android.h5NativtCallback();即window.js接口名.加方法名

1. 通过WebviewClient拦截url协议（异步调用）
2. Android native和h5定好协议；
3. 通过WebviewClient的shouldOverrideUrlLoading拦截指定协议头
4. 通过WebChromeClient的onJsAlert()、onJsConfirm()和onJsPrompt()分别拦截js的alert()、confirm()和prompt()方法（同步调用）
5. android调用js方法
6. mWebView.loadUrl("javascript:callbackFromNative('the data is from android! ‘)"

loadUrl方法中的字符串参数构成：”javascript:”+js方法，其中js方法中的参数是传递给js方法的参数

1. mWebview.evaluateJavascript("javascript:callbackFromNative('data from android!‘)“, new ValueCallback<String>() {

@Override

public void onReceiveValue(String value) { }

});

1. 区别：loadUrl需要刷新页面，且需要回调时要通过@JavaScriptInterface注解的方式实现；而evaluateJavascript不需要刷新页面，自带回调。
2. Webview使用中容易出现的问题
3. 闪屏
4. 内存泄漏