一 基本用法

WebView也是Android View的一种, 我们通常用它来在应用内部展示网页, 和以往一样, 我们先来简单看一下它的基本用法。

添加网络权限

```
<uses-permission android:name="android.permission.INTERNET"
```

在布局中添加WebView

```
<?xml version="1.0" encoding="utf-8"?>
<webview  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

使用WebView加载网页

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

以上就是WebView的简单用法, 相比大家已经十分熟悉, 下面我们就来逐一看看WebView的其他特性。

WebView基本组件

了解了基本用法, 我们对WebView就有了大致的印象, 下面我们来看看构建Web应用的三个重要组件。

WebSettings

WebSettings用来对WebView做各种设置, 你可以这样获取WebSettings:

```
WebSettings webSettings = mWebView .getSettings();
```

WebSettings的常见设置如下所示:

JS处理

- setJavaScriptEnabled(true); //支持js
- setPluginsEnabled(true); //支持插件
- setJavaScriptCanOpenWindowsAutomatically(true); //支持通过JS打开新窗口

缩放处理

- setUseWideViewPort(true); //将图片调整到适合webview的大小
- setLoadWithOverviewMode(true); // 缩放至屏幕的大小
- setSupportZoom(true); //支持缩放，默认为true。是下面那个的前提。
- setBuiltInZoomControls(true); //设置内置的缩放控件。 这个取决于setSupportZoom(), 若setSupportZoom(false)，则该WebView不可缩放，这个不管设置什么都不能缩放。
- setDisplayZoomControls(false); //隐藏原生的缩放控件

内容布局

- setLayoutAlgorithm(LayoutAlgorithm.SINGLE_COLUMN); //支持内容重新布局
- supportMultipleWindows(); //多窗口

文件缓存

- setCacheMode(WebSettings.LOAD_CACHE_ELSE_NETWORK); //关闭webview中缓存
- setAllowFileAccess(true); //设置可以访问文件

其他设置

- setNeedInitialFocus(true); //当webview调用requestFocus时为webview设置节点
- setLoadsImagesAutomatically(true); //支持自动加载图片
- setDefaultTextEncodingName("utf-8"); //设置编码格式
- setPluginState(PluginState.OFF); //设置是否支持flash插件
- setDefaultFontSize(20); //设置默认字体大小

WebViewClient

WebViewClient用来帮助WebView处理各种通知, 请求事件。我们通过继承WebViewClient并重载它的方法可以实现不同功能的定制。具体如下所示:

- shouldOverrideUrlLoading(WebView view, String url) //在网页上的所有加载都经过这个方法,这个函数我们可以做很多操作。比如获取url，查看url.contains(“add”)，进行添加操作
- shouldOverrideKeyEvent(WebView view, KeyEvent event) //处理在浏览器中的按键事件。
- onPageStarted(WebView view, String url, Bitmap favicon) //开始载入页面时调用的，我们可以设定一个loading的页面，告诉用户程序在等待网络响应。
- onPageFinished(WebView view, String url) //在页面加载结束时调用, 我们可以关闭loading 条，切换程序动作。
- onLoadResource(WebView view, String url) //在加载页面资源时会调用，每一个资源（比如图片）的加载都会调用一次。
- onReceivedError(WebView view, int errorCode, String description, String failingUrl) //报告错误信息
- doUpdateVisitedHistory(WebView view, String url, boolean isReload) //更新历史记录
- onFormResubmission(WebView view, Message dontResend, Message resend) //应用程序重新请求网页数据
- onReceivedHttpAuthRequest(WebView view, HttpAuthHandler handler, String host,String realm) //获取返回信息授权请求
- onReceivedSslError(WebView view, SslErrorHandler handler, SslError error) //让webview处理https请求。
- onScaleChanged(WebView view, float oldScale, float newScale) //WebView发生改变时调用
- onUnhandledKeyEvent(WebView view, KeyEvent event) //Key事件未被加载时调用

WebChromeClient

WebChromeClient用来帮助WebView处理JS的对话框、网址图标、网址标题和加载进度等。同样地, 通过继承WebChromeClient并重载它的方法也可以实现不同功能的定制, 如下所示:

- public void onProgressChanged(WebView view, int newProgress); //获得网页的加载进度，显示在右上角的TextView控件中
- public void onReceivedTitle(WebView view, String title); //获取Web页中的title用来设置自己界面中的title, 当加载出错的时候，比如无网络，这时onReceiveTitle中获取的标题为"找不到该网页",
- public void onReceivedIcon(WebView view, Bitmap icon); //获取Web页中的icon
- public boolean onCreateWindow(WebView view, boolean isDialog, boolean isUserGesture, Message resultMsg);
- public void onCloseWindow(WebView window);

- public boolean onJsAlert(WebView view, String url, String message, JsResult result); //处理alert弹出框，html 弹框的一种方式
- public boolean onJsPrompt(WebView view, String url, String message, String defaultValue, JsPromptResult result) //处理confirm弹出框
- public boolean onJsConfirm(WebView view, String url, String message, JsResult result); //处理prompt弹出框

WebView生命周期

onResume()

WebView为活跃状态时回调，可以正常执行网页的响应。

onPause()

WebView被切换到后台时回调, 页面被失去焦点, 变成不可见状态，onPause动作通知内核暂停所有的动作，比如DOM的解析、plugin的执行、JavaScript执行。

pauseTimers()

当应用程序被切换到后台时回调，该方法针对全应用程序的WebView，它会暂停所有webview的layout，parsing，javascripttimer。降低CPU功耗。

resumeTimers()

恢复pauseTimers时的动作。

destroy()

关闭了Activity时回调, WebView调用destory时, WebView仍绑定在Activity上.这是由于自定义WebView构建时传入了该Activity的context对象, 因此需要先从父
容器中移除WebView, 然后再销毁webview。

```
mRootLayout.removeView(webView);
mWebView.destroy();
```

WebView页面导航

页面跳转

当我们在WebView点击链接时, 默认的WebView会直接跳转到别的浏览器中, 如果想要实现在WebView内跳转就需要设置WebViewClient, 下面我们先来
说说WebView、WebViewClient、WebChromeClient三者的区别。

- WebView: 主要负责解析和渲染网页
- WebViewClient: 辅助WebView处理各种通知和请求事件
- WebChromeClient: 辅助WebView处理JavaScript中的对话框, 网址图标和标题等

如果我们想控制不同链接的跳转方式, 我们需要继承WebViewClient重写shouldOverrideUrlLoading()方法

```
    static class CustomWebViewClient extends WebViewClient {

        private Context mContext;

        public CustomWebViewClient(Context context) {
```

```
            mContext = context;
        }

        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            if (Uri.parse(url).getHost().equals("github.com/guoxiaoxing")) {
                //如果是自己站点的链接，则用本地WebView跳转
                return false;
            }
            //如果不是自己的站点则launch别的Activity来处理
            Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
            mContext.startActivity(intent);
            return true;
        }
    }
```

关于shouldOverrideUrlLoading()方法的两点说明:

1 方法返回值

返回true: Android 系统会处理URL, 一般是唤起系统浏览器。
返回false: 当前 WebView 处理URL。

由于默认放回false, 如果我们只想在WebView内处理链接跳转只需要设置
mWebView.setWebViewClient(new WebViewClient())即可

```
/**
     * Give the host application a chance to take over the control when a new
     * url is about to be loaded in the current WebView. If WebViewClient is not

     * provided, by default WebView will ask Activity Manager to choose the
     * proper handler for the url. If WebViewClient is provided, return true
     * means the host application handles the url, while return false means the
     * current WebView handles the url.
     * This method is not called for requests using the POST "method".
     *
     * @param view The WebView that is initiating the callback.
     * @param url The url to be loaded.
     * @return True if the host application wants to leave the current WebView
     *         and handle the url itself, otherwise return false.
     */
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        return false;
    }
```

2 方法deprecated问题

shouldOverrideUrlLoading()方法在API >= 24时被标记deprecated, 它的替代方法是

```
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest
request) {
            view.loadUrl(request.toString());
            return true;
        }
```

但是public boolean shouldOverrideUrlLoading(WebView view, String url)支持更广泛的API我们在使用的时候还是它,

页面回退

Android的返回键, 如果想要实现WebView内网页的回退, 可以重写onKeyEvent()方法。

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the Back button and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack()) {
        myWebView.goBack();
        return true;
    }
    // If it wasn't the Back key or there's no web page history, bubble up to the
default
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```

页面滑动

关于页面滑动, 我们在做下拉刷新等功能时, 经常会去判断WebView是否滚动到顶部或者滚动到底部。

我们先来看一看三个判断高度的方法

```
getScrollY();
```

该方法返回的是当前可见区域的顶端距整个页面顶端的距离,也就是当前内容滚动的距离.

```
getHeight();
getBottom();
```

该方法都返回当前WebView这个容器的高度

```
getContentHeight();
```

返回的是整个html的高度, 但并不等同于当前整个页面的高度, 因为WebView有缩放功能, 所以当前整个页面的高度实际上应该是原始html的高度
再乘上缩放比例. 因此, 判断方法是:

```
if (webView.getContentHeight() * webView.getScale() == (webView.getHeight() +
webView.getScrollY())) {
    //已经处于底端
}

if(webView.getScrollY() == 0){
    //处于顶端
}
```

以上这个方法也是我们常用的方法, 不过从API 17开始, mWebView.getScale()被标记为deprecated

> This method was deprecated in API level 17. This method is prone to inaccuracy due to race conditions
> between the web rendering and UI threads; prefer onScaleChanged(WebView,

因为scale的获取可以用一下方式:

```
public class CustomWebView extends WebView {

public CustomWebView(Context context) {
    super(context);
    setWebViewClient(new WebViewClient() {
        @Override
        public void onScaleChanged(WebView view, float oldScale, float newScale)
{
            super.onScaleChanged(view, oldScale, newScale);
            mCurrentScale = newScale
        }
    });
}
```

WebView缓存实现

在项目中如果使用到WebView控件, 当加载html页面时, 会在/data/data/包名目录下生成database与cache两个文件夹。
请求的url记录是保存在WebViewCache.db, 而url的内容是保存在WebViewCache文件夹下。

控制缓存行为

```
WebSettings webSettings = mWebView.getSettings();
//优先使用缓存
webSettings.setCacheMode(WebSettings.LOAD_CACHE_ELSE_NETWORK);
//只在缓存中读取
webSettings.setCacheMode(WebSettings.LOAD_CACHE_ONLY);
/不使用缓存
WwebSettings.setCacheMode(WebSettings.LOAD_NO_CACHE);
```

清除缓存

```
clearCache(true); //清除网页访问留下的缓存，由于内核缓存是全局的因此这个方法不仅仅针对
webview而是针对整个应用程序.
clearHistory (); //清除当前webview访问的历史记录，只会webview访问历史记录里的所有记录除了
当前访问记录.
clearFormData () //这个api仅仅清除自动完成填充的表单数据，并不会清除webView存储到本地的数
据。
```

WebView Cookies

添加Cookies

```
public void synCookies() {
    if (!CacheUtils.isLogin(this)) return;
    CookieSyncManager.createInstance(this);
    CookieManager cookieManager = CookieManager.getInstance();
    cookieManager.setAcceptCookie(true);
    cookieManager.removeSessionCookie();//移除
    String cookies = PreferenceHelper.readString(this, AppConfig.COOKIE_KEY,
AppConfig.COOKIE_KEY);
    KJLoger.debug(cookies);
    cookieManager.setCookie(url, cookies);
    CookieSyncManager.getInstance().sync();
}
```

清除Cookies

```
CookieManager.getInstance().removeSessionCookie();
```

WebView本地资源访问

当我们在WebView中加载出从web服务器上拿取的内容时，是无法访问本地资源的，如assets目录下的图片资源，因为这样的行为属于跨域行为（Cross-Domain），而WebView是禁止的。解决这个问题的方案是把html内容先下载到本地，然后使用loadDataWithBaseURL加载html。这样就可以在html中使用 file:///android_asset/xxx.png 的链接来引用包里面assets下的资源了。

```
private void loadWithAccessLocal(final String htmlUrl) {
    new Thread(new Runnable() {
        public void run() {
            try {
                final String htmlStr = NetService.fetchHtml(htmlUrl);
                if (htmlStr != null) {
                    TaskExecutor.runTaskOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            loadDataWithBaseURL(htmlUrl, htmlStr, "text/html",
"UTF-8", "");
                        }
                    });
                    return;
                }
            } catch (Exception e) {
                Log.e("Exception:" + e.getMessage());
            }

            TaskExecutor.runTaskOnUiThread(new Runnable() {
                @Override
                public void run() {
                    onPageLoadedError(-1, "fetch html failed");
                }
            });
        }
    }).start();
}
```

注意

- 从网络上下载html的过程应放在工作线程中
- html下载成功后渲染出html的步骤应放在UI主线程，不然WebView会报错
- html下载失败则可以使用我们前面讲述的方法来显示自定义错误界面

二 代码交互

Android原生方案

关于WebView中Java代码和JS代码的交互实现, Android给了一套原生的方案, 我们先来看看原生的用法。后面我们还会讲到其他的开源方法。

JavaScript代码和Android代码是通过addJavascriptInterface()来建立连接的, 我们来看下具体的用法。

1 设置WebView支持JavaScript

```
webView.getSettings().setJavaScriptEnabled(true);
```

2 在Android工程里定义一个接口

```
public class WebAppInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

注意: API >= 17时, 必须在被JavaScript调用的Android方法前添加@JavascriptInterface注解, 否则将无法识别。

3 在Android代码中将该接口添加到WebView

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new WebAppInterface(this), "Android");
```

这个"Android"就是我们为这个接口取的别名, 在JavaScript就可以通过Android.showToast(toast)这种方式来调用此方法。

4 在JavaScript中调用Android方法

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello
Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

在JavaScript中我们不用再去实例化WebAppInterface接口, WebView会自动帮我们完成这一工作, 使它能够为WebPage所用。

注意:

由于addJavascriptInterface()给予了JS代码控制应用的能力, 这是一项非常有用的特性, 但同时也带来了安全上的隐患,

> Using addJavascriptInterface() allows JavaScript to control your Android application. This can be a very useful feature or a dangerous
> security issue. When the HTML in the WebView is untrustworthy (for example, part or all of the HTML is provided by an unknown person or
> process), then an attacker can include HTML that executes your client-side code and possibly any code of the attacker's choosing. As such,
> you should not use addJavascriptInterface() unless you wrote all of the HTML and JavaScript that appears in your WebView. You should also
> not allow the user to navigate to other web pages that are not your own, within your WebView (instead, allow the user's default browser
> application to open foreign links—by default, the user's web browser opens all URL links, so be careful only if you handle page navigation
> as described in the following section).

下面正式引入我们在项目中常用的两套开源的替代方案

jockeyjs开源方案

jockeyjs 是一套IOS/Android双平台的Native和JS交互方法, 比较适合用在项目中。

> Library to facilitate communication between iOS apps and JS apps running inside a UIWebView

jockeyjs对Native和JS的交互做了优美的封装, 事件的发送与接收都可以通过send()和on()来完成。我们先简单的看一下Event的发送与接收。

Sending events from app to JavaScript

```
// Send an event to JavaScript, passing a payload
jockey.send("event-name", webView, payload);

//With a callback to execute after all listeners have finished
jockey.send("event-name", webView, payload, new JockeyCallback() {
    @Override
    public void call() {
        //Your execution code
    }
});
```

Receiving events from app in JavaScript

```
// Listen for an event from iOS, but don't notify iOS we've completed processing
// until an asynchronous function has finished (in this case a timeout).
Jockey.on("event-name", function(payload, complete) {
  // Example of event'ed handler.
  setTimeout(function() {
    alert("Timeout over!");
    complete();
  }, 1000);
});
```

Sending events from JavaScript to app

```
// Send an event to iOS.
Jockey.send("event-name");

// Send an event to iOS, passing an optional payload.
Jockey.send("event-name", {
  key: "value"
});

// Send an event to iOS, pass an optional payload, and catch the callback when
all the
// iOS listeners have finished processing.
Jockey.send("event-name", {
  key: "value"
}, function() {
  alert("iOS has finished processing!");
});
```

Receiving events from JavaScript in app

```
//Listen for an event from JavaScript and log a message when we have receied it.
jockey.on("event-name", new JockeyHandler() {
    @Override
    protected void doPerform(Map<Object, Object> payload) {
        Log.d("jockey", "Things are happening");
    }
});

//Listen for an event from JavaScript, but don't notify the JavaScript that the
listener has completed
//until an asynchronous function has finished
//Note: Because this method is executed in the background, if you want the method
to interact with the UI thread
//it will need to use something like a android.os.Handler to post to the UI
thread.
jockey.on("event-name", new JockeyAsyncHandler() {
    @Override
    protected void doPerform(Map<Object, Object> payload) {
        //Do something asynchronously
        //No need to called completed(), Jockey will take care of that for you!
    }
});
```

```
//We can even chain together several handlers so that they get processed in
sequence.
//Here we also see an example of the NativeOS interface which allows us to chain
some common
//system handlers to simulate native UI interactions.
jockey.on("event-name", nativeOS(this)
            .toast("Event occurred!")
            .vibrate(100), //Don't forget to grant permission
            new JockeyHandler() {
                @Override
                protected void doPerform(Map<Object, Object> payload) {
                }
            }
);

//...More Handlers


//If you would like to stop listening for a specific event
jockey.off("event-name");

//If you would like to stop listening to ALL events
jockey.clear();
```

通过上面的代码, 我们对jockeyjs的使用有了大致的理解, 下面我们具体来看一下在项目中的使用。

1 依赖配置

下载代码， 将JockeyJS.Android导入到工程中。

2 jockeyjs配置

jockeyjs有两种使用方式

方式一:

只在一个Activity中使用jockey或者多Activity共享一个jockey实例

```
//Declare an instance of Jockey
Jockey jockey;

//The WebView that we will be using, assumed to be instantiated either through
findViewById or some method of injection.
WebView webView;

WebViewClient myWebViewClient;

@Override
protected void onStart() {
    super.onStart();

    //Get the default JockeyImpl
    jockey = JockeyImpl.getDefault();

    //Configure your webView to be used with Jockey
    jockey.configure(webView);

    //Pass Jockey your custom WebViewClient
    //Notice we can do this even after our webView has been configured.
```

```
    jockey.setWebViewClient(myWebViewClient)

    //Set some event handlers
    setJockeyEvents();

    //Load your webPage
    webView.loadUrl("file:///your.url.com");
}
```

方式二:

另一种就是把jockey当成一种全局的Service来用, 这种方式下我们可以在多个Activity之间甚至整个应用内共享handler. 当然我们同样需要
把jockey的生命周期和应用的生命周期绑定在一起。

```
//First we declare the members involved in using Jockey

//A WebView to interact with
private WebView webView;

//Our instance of the Jockey interface
private Jockey jockey;

//A helper for binding services
private boolean _bound;

//A service connection for making use of the JockeyService
private ServiceConnection _connection = new ServiceConnection() {
    @Override
    public void onServiceDisconnected(ComponentName name) {
        _bound = false;
    }

    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        JockeyBinder binder = (JockeyBinder) service;

        //Retrieves the instance of the JockeyService from the binder
        jockey = binder.getService();

        //This will setup the WebView to enable JavaScript execution and provide
a custom JockeyWebViewClient
        jockey.configure(webView);

        //Make Jockey start listening for events
        setJockeyEvents();

        _bound = true;

        //Redirect the WebView to your webpage.
        webView.loadUrl("file:///android_assets/index.html");
    }

}

///....Other member variables..../////
```

```
//Then we bind the JockeyService to our activity through a helper function in our
onStart method
@Override
protected void onStart() {
    super.onStart();
    JockeyService.bind(this, _connection);
}

//In order to bind this with the Android lifecycle we need to make sure that the
service also shuts down at the appropriate time.
@Override
protected void onStop() {
    super.onStop();
    if (_bound) {
        JockeyService.unbind(this, _connection);
    }
}
```

以上便是jockeyjs的大致用法.

三 性能优化

优化网页加载速度

默认情况html代码下载到WebView后，webkit开始解析网页各个节点，发现有外部样式文件或者外部脚本文件时，会异步发起网络请求下载文件，但如果
在这之前也有解析到image节点，那势必也会发起网络请求下载相应的图片。在网络情况较差的情况下，过多的网络请求就会造成带宽紧张，影响到css或
js文件加载完成的时间，造成页面空白loading过久。解决的方法就是告诉WebView先不要自动加载图片，等页面finish后再发起图片加载。

设置WebView, 先禁止加载图片

```
WebSettings webSettings = mWebView.getSettings();

//图片加载
if(Build.VERSION.SDK_INT >= 19){
    webSettings.setLoadsImagesAutomatically(true);
}else {
    webSettings.setLoadsImagesAutomatically(false);
}
```

覆写WebViewClient的onPageFinished()方法, 页面加载结束后再加载图片

```
@Override
public void onPageFinished(WebView view, String url) {
    super.onPageFinished(view, url);
    if (!view.getSettings().getLoadsImagesAutomatically()) {
        view.getSettings().setLoadsImagesAutomatically(true);
    }
}
```

注意: 4.4以上系统在onPageFinished时再恢复图片加载时,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。

硬件加速页面闪烁问题

4.0以上的系统我们开启硬件加速后，WebView渲染页面更加快速，拖动也更加顺滑。但有个副作用就是，当WebView视图被整体遮住一块，然后突然恢复时（比如使用SlideMenu将WebView从侧边滑出来时），这个过渡期会出现白块同时界面闪烁。解决这个问题的方法是在过渡期前将WebView的硬件加速临时关闭，过渡期后再开启，如下所示:

过度前关闭硬件加速

```
if(Build.VERSION.SDK_INT > Build.VERSION_CODES.HONEYCOMB){
    mWebView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
}
```

过度前开启硬件加速

```
if(Build.VERSION.SDK_INT > Build.VERSION_CODES.HONEYCOMB){
    mWebView.setLayerType(View.LAYER_TYPE_HARDWARE, null);
}
```

以上就是本篇文章的全部内容, 大致就说这么多，在实际的项目中我们通常会自己去封装一个H5Activity用来统一显示H5页面, 下面就提供了完整的H5Activity,

封装了WebView各种特性与jockeyjs代码交互。该H5Activity提供WebView常用设置、H5页面解析、标题解析、进度条显示、错误页面展示、重新加载等功能。可以拿去稍作改造, 用于自己的项目中。

```
package com.guoxiaoxing.webview;

import android.content.Context;
import android.graphics.Bitmap;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Build;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.text.TextUtils;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.Window;
import android.webkit.JsResult;
import android.webkit.WebChromeClient;
import android.webkit.WebResourceError;
import android.webkit.WebResourceRequest;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.ProgressBar;
import com.jockeyjs.Jockey;
import com.jockeyjs.JockeyImpl;
public class H5Activity extends AppCompatActivity {

    public static final String H5_URL = "H5_URL";
    private static final String JOCKEY_EVENT_NAME = "JOCKEY_EVENT_NAME";
    private static final String TAG = H5Activity.class.getSimpleName();
```

```
    private Toolbar mToolbar;
    private ProgressBar mProgressBar;

    private Jockey mJockey;
    private WebView mWebView;
    private WebViewClient mWebViewClient;
    private WebChromeClient mWebChromeClient;

    private String mUrl;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        supportRequestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_h5);
        setupView();
        setupSettings();
    }

    @Override
    protected void onStart() {
        super.onStart();
        setupJockey();
        setupData();
    }

    private void setupView() {
        mToolbar = (Toolbar) findViewById(R.id.h5_toolbar);
        mProgressBar = (ProgressBar) findViewById(R.id.h5_progressbar);
        mWebView = (WebView) findViewById(R.id.h5_webview);
    }

    private void setupSettings() {

        mWebView.setScrollBarStyle(WebView.SCROLLBARS_INSIDE_OVERLAY);
        mWebView.setHorizontalScrollBarEnabled(false);
        mWebView.setOverScrollMode(WebView.OVER_SCROLL_NEVER);

        WebSettings mWebSettings = mWebView.getSettings();
        mWebSettings.setSupportZoom(true);
        mWebSettings.setLoadWithOverviewMode(true);
        mWebSettings.setUseWideViewPort(true);
        mWebSettings.setDefaultTextEncodingName("utf-8");
        mWebSettings.setLoadsImagesAutomatically(true);

        //JS
        mWebSettings.setJavaScriptEnabled(true);
        mWebSettings.setJavaScriptCanOpenWindowsAutomatically(true);

        mWebSettings.setAllowFileAccess(true);
        mWebSettings.setUseWideViewPort(true);
        mWebSettings.setDatabaseEnabled(true);
        mWebSettings.setLoadWithOverviewMode(true);
        mWebSettings.setDomStorageEnabled(true);


        //缓存
```

```
        ConnectivityManager connectivityManager = (ConnectivityManager)
this.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo info = connectivityManager.getActiveNetworkInfo();
        if (info != null && info.isConnected()) {
            String wvcc = info.getTypeName();
            Log.d(TAG, "current network: " + wvcc);
            mWebSettings.setCacheMode(WebSettings.LOAD_DEFAULT);
        } else {
            Log.d(TAG, "No network is connected, use cache");
            mWebSettings.setCacheMode(WebSettings.LOAD_CACHE_ELSE_NETWORK);
        }

        if (Build.VERSION.SDK_INT >= 16) {
            mWebSettings.setAllowFileAccessFromFileURLs(true);
            mWebSettings.setAllowUniversalAccessFromFileURLs(true);
        }

        if (Build.VERSION.SDK_INT >= 12) {
            mWebSettings.setAllowContentAccess(true);
        }

        setupWebViewClient();
        setupWebChromeClient();
    }

    private void setupJockey() {
        mJockey = JockeyImpl.getDefault();
        mJockey.configure(mWebView);
        mJockey.setWebViewClient(mWebViewClient);
        mJockey.setOnValidateListener(new Jockey.OnValidateListener() {
            @Override
            public boolean validate(String host) {
                return "yourdomain.com".equals(host);
            }
        });

        //TODO set your event handler
        mJockey.on(JOCKEY_EVENT_NAME, new EventHandler());
    }

    private void setupData() {
        mUrl = getIntent().getStringExtra(H5_URL);
        if (TextUtils.isEmpty(mUrl)) {
            //TODO show error page
        } else {
            mWebView.loadUrl(mUrl);
        }
    }

    private void setupWebViewClient() {
        mWebViewClient = new WebViewClient() {
            @Override
            public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request) {
                //TODO 处理URL, 例如对指定的URL做不同的处理等
                return false;
            }
```

```
            @Override
            public void onPageFinished(WebView view, String url) {
                super.onPageFinished(view, url);
            }

            @Override
            public void onPageStarted(WebView view, String url, Bitmap favicon)
{
                super.onPageStarted(view, url, favicon);
            }

            @Override
            public void onReceivedError(WebView view, WebResourceRequest request,
WebResourceError error) {
                super.onReceivedError(view, request, error);
            }
        };
        mWebView.setWebViewClient(mWebViewClient);
    }

    private void setupWebChromeClient() {
        mWebChromeClient = new WebChromeClient() {
            @Override
            public void onReceivedTitle(WebView view, String title) {
                super.onReceivedTitle(view, title);
                mToolbar.setTitle(title);

            }

            @Override
            public void onProgressChanged(WebView view, int newProgress) {
                super.onProgressChanged(view, newProgress);
                mProgressBar.setProgress(newProgress);
                if (newProgress == 100) {
                    mProgressBar.setVisibility(View.GONE);
                } else {
                    mProgressBar.setVisibility(View.VISIBLE);
                }
            }

            @Override
            public boolean onJsAlert(WebView view, String url, String message,
JsResult result) {
                return super.onJsAlert(view, url, message, result);
            }
        };
        mWebView.setWebChromeClient(mWebChromeClient);
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack()) {
            mWebView.goBack();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```