# **http://www.cnblogs.com/hubcarl/p/4202784.html**

# **[Cordova android框架详解](http://www.cnblogs.com/hubcarl/p/4202784.html)**

**一、Cordova 核心java类说明**

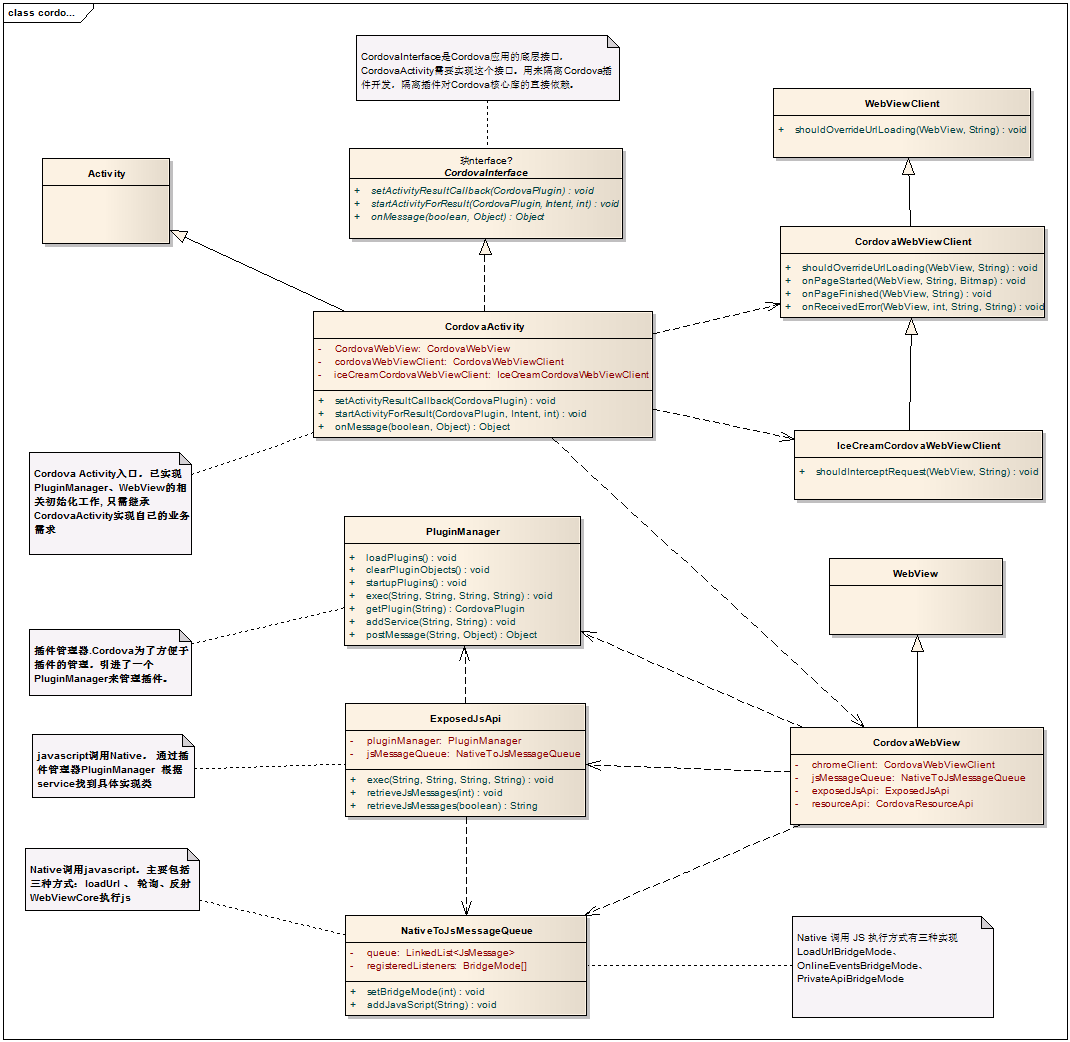
**CordovaActivity**：Cordova Activity入口，已实现PluginManager、WebView的相关初始化工作, 只需继承CordovaActivity实现自己的业务需求。

**PluginManager**： 插件管理器

**ExposedJsApi** ：javascript调用Native， 通过插件管理器PluginManager 根据service找到具体实现类。

**NativeToJsMessageQueue**：Native调用javascript，主要包括三种方式：loadUrl 、 轮询、反射WebViewCore执行js

二、 **Cordova框架类图**

****

**三、Cordova框架启动**

**当实现了DroidGap或者CordovaInterface接口的Activity的onCreate方法中调用DroidGap的loadUrl方法即启动了Cordova框架。**

Cordova提供了一个Class（**DroidGap** extends **CordovaActivity**）和一个interface（**CordovaInterface**）来让Android开发者开发Cordova。

一般情况下实现DroidGap即可，因为DroidGap类已经做了很多准备工作，可以说DroidGap类是Cordova框架的一个重要部分；如果在必要的情况下实现CordovaInterface接口，那么这个类中很多DroidGap的功能需要自己去实现。继承了DroidGap或者CordovaInterface的Activity就是一个独立的Cordova模块，独立的Cordova模块指的是每个实现了DroidGap或者CordovaInterface接口的Activity都对应一套独立的WebView，Plugin，PluginManager，没有共享的。

在初始化完CordovaWebView后调用CordovaWebView.loadUrl()。此时完成Cordova的启动。

**1.Cordova关联对象初始化**

在实例化CordovaWebView的时候, CordovaWebView对象会去创建一个属于当前CordovaWebView对象的插件管理器PluginManager对象，一个消息队列NativeToJsMessageQueue对象，一个JavascriptInterface对象ExposedJsApi，并将ExposedJsApi对象添加到CordovaWebView中，JavascriptInterface名字为：\_cordovaNative。

**2. Cordova的JavascriptInterface**  
在创建ExposedJsApi时需要CordovaWebView的PluginManager对象和NativeToJsMessageQueue对象。因为所有的JS端与Android native代码交互都是通过ExposedJsApi对象的exec方法。在exec方法中执行PluginManager的exec方法，PluginManager去查找具体的Plugin并实例化然后再执行Plugin的execute方法，并根据同步标识判断是同步返回给JS消息还是异步。由NativeToJsMessageQueue统一管理返回给JS的消息。

**3. 何时加载Plugin，如何加载**  
 Cordova在启动每个Activity的时候都会将配置文件中的所有plugin加载到PluginManager。那么是什么时候将这些plugin加载到PluginManager的呢？在b中说了最后会调用CordovaWebView.loadUrl()，对，就在这个时候会去初始化PluginManager并加载plugin。PluginManager在加载plugin的时候并不是马上实例化plugin对象，而是只是将plugin的Class名字保存到一个hashmap中，用service名字作为key值。  
当JS端通过JavascriptInterface接口的ExposedJsApi对象请求Android时，PluginManager会从hashmap中查找到plugin，如果该plugin还未实例化，利用java反射机制实例化该plugin，并执行plugin的execute方法。

**4．Cordova的数据返回**

Cordova中通过exec()函数请求android插件，数据的返回可同步也可以异步于exec()函数的请求。在开发android插件的时候可以重写public boolean isSynch(String action)方法来决定是同步还是异步。Cordova在android端使用了一个队列(NativeToJsMessageQueue)来专门管理返回给JS的数据。

1）同步  
   Cordova在执行完exec()后，android会马上返回数据，但不一定就是该次请求的数据，可能是前面某次请求的数据；因为当exec()请求的插件是允许同步返回数据的情况下，Cordova也是从NativeToJsMessageQueue队列头pop头数据并返回。然后再根据callbackID反向查找某个JS请求，并将数据返回给该请求的success函数。  
     
2）异步  
   Cordova在执行完exec()后并不会同步得到一个返回数据。Cordova在执行exec()的同时启动了一个XMLHttpRequest对象方式或者prompt()函数方式的循环函数来不停的去获取NativeToJsMessageQueue队列中的数据，并根据callbackID反向查找到相对应的JS请求，并将该数据交给success函数。  
   注：Cordova对本地的HTML文件(file:// 开头的URL)或者手机设置有代理的情况下使用XMLHttpRequest方式获取返回数据，其他则使用prompt()函数方式获取返回数据。

**5、webView.sendJavascript 发送到js队列，onNativeToJsMessageAvailable 负责执行js.**

Native 调用 JS 执行方式有三种实现 LoadUrlBridgeMode、 OnlineEventsBridgeMode、PrivateApiBridgeMode

1、webView.sendJavascript 发送js方法到JS队列

2、onJsPrompt 方法拦截，获取调用方式

》》如果是gap\_bridge\_mode，则执行 appView.exposedJsApi.setNativeToJsBridgeMode(Integer.parseInt(message));  
》》如果是gap\_poll, 则执行 appView.exposedJsApi.retrieveJsMessages("1".equals(message));

3、调用setBridgeMode 方法调用onNativeToJsMessageAvailable 执行javascript调用

**四、Native调用javascript 方式：NativeToJsMessageQueue**

**1、loadUrl javascript 调用方式**

|  |  |
| --- | --- |
| 1  2  3  4  5 | private class LoadUrlBridgeMode extends BridgeMode    if (url.startsWith("file://") || url.startsWith("javascript:") || Config.isUrlWhiteListed(url)) {    } |

**2、Navitive事件通知javascript轮询获取Navitive数据**

|  |  |
| --- | --- |
| 1  2  3 | private class OnlineEventsBridgeMode extends BridgeMode |

**3、通过Java反射获取webview 的sendMessage 方法执行js， 支持 Android 3.2.4之上(包含)**

---可以解决loadUrl 隐藏键盘的问题：当你的焦点在输入，如果这通过loadUrl调用js，会导致键盘隐藏

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | private class PrivateApiBridgeMode extends BridgeMode        Field f = webViewClass.getDeclaredField("mProvider");      f.setAccessible(true);      webViewObject = f.get(webView);      webViewClass = webViewObject.getClass();        Field f = webViewClass.getDeclaredField("mWebViewCore");      f.setAccessible(true);      webViewCore = f.get(webViewObject);       if (webViewCore != null) {          sendMessageMethod =       webViewCore.getClass().getDeclaredMethod("sendMessage", Message.class);         sendMessageMethod.setAccessible(true);      }         Message execJsMessage = Message.obtain(null, EXECUTE\_JS, url);       sendMessageMethod.invoke(webViewCore, execJsMessage); |

**4、Native注册javascript接口 \_cordovaNative**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | boolean isHoneycomb = (SDK\_INT >= Build.VERSION\_CODES.HONEYCOMB && SDK\_INT <= Build.VERSION\_CODES.HONEYCOMB\_MR2);  // Bug being that Java Strings do not get converted to JS strings automatically.This isn't hard to work-around on the JS side, but it's easier to just use the prompt bridge instead.  if (isHoneycomb || (SDK\_INT < Build.VERSION\_CODES.GINGERBREAD)) {  Log.i(TAG, "Disabled addJavascriptInterface() bridge since Android version is old.");  return;  } else if (SDK\_INT < Build.VERSION\_CODES.HONEYCOMB && Build.MANUFACTURER.equals("unknown")) {  // addJavascriptInterface crashes on the 2.3 emulator.  Log.i(TAG, "Disabled addJavascriptInterface() bridge callback due to a bug on the 2.3 emulator");  return;  }  this.addJavascriptInterface(exposedJsApi, "\_cordovaNative"); |