WebKit – WebKit For Android

一、WebKit 简介

WebKit 是一个开源的浏览器网页排版引擎，包含 WebCore 排版引擎和 JSCore 引擎。
WebCore 和 JSCore 引擎来自于 KDE 项目的 KHTML 和 KJS 开源项目。Android 平台的 Web 引擎框架采用了 WebKit 项目中的 WebCore 和 JSCore 部分，上层由 Java 语言封装，并且作
为 API 提供给 Android 应用开发者，而底层使用 WebKit 核心库（WebCore 和 JSCore）进行网页排版。

二、WebKit 目录结构

Android 平台的 WebKit 模块分成 Java 和 WebKit 库两个部分，其目录结构如下表所示：
WebKit 模块目录结构(Java 层（根目录 device\java\android\android\webkit）)

1. BrowserFrame.java: BrowserFrame 对象是对 WebCore 库中的 Frame 对象的 Java 层封装，用于创建 WebCore 中定义的 Frame，及为该 Frame 对象提供 Java 层回调方法。

2. ByteArrayBuilder.java: ByteArrayBuilder 辅助对象，用于 byte 块链表的处理。

3. CachLoader.java: URL Cache 载入器对象，该对象实现 StreadLoader 抽象基类，用于通过 CacheResult 对象载入内容数据。

4. CacheManager.java: Cache 管理对象，负责 Java 层 Cache 对象管理

5. CacheSyncManager.java: Cache 同步管理对象，负责同步 RAM 和 FLASH 之间的浏览器 Cache 数据。实际的物理数据操作在 WebSyncManager 对象中完成。

6. CallbackProxy.java: 该对象是用于处理 WebCore 与 UI 线程消息的代理类。当有 Web 事件产生时 WebCore 线程会调用该回调代理类，代理类会通过消息的方式通知 UI 线程，并且调用设置的客户对象的回调函数。

7. CellList.java: CellList 定义图片集合中的 Cell，管理 Cell 图片的绘制、状态改变以及索引。

8. CookieManager.java: 根据 RFC2109 规范，管理 cookies

9. CookieSyncManager.java: Cookies 同步管理对象，该对象负责同步 RAM 和 Flash 之间的 Cookies 数据。实际的物理数据操作在基类 WebSyncManager 中完成。

10. DataLoader.java: 数据载入器对象，用于载入网页数据。

11. DateSorter.java: 尚未使用

12. DownloadListener.java: 下载侦听器接口

13. DownloadManagerCore.java: 下载管理器对象，管理下载列表。该对象运行在 WebKit 的线程中，通过 CallbackProxy 对象与 UI 线程交互。

14. FileLoader.java: 文件载入器，将文件数据载入到 Frame 中。

15. FrameLoader.java: Frame 载入器，用于载入网页 Frame 数据

16. HttpAuthHandler.java: Http 认证处理对象，该对象会作为参数传递给 BrowserCallback.displayHttpAuthDialog 方法，与用户交互。

17. HttpDataTime.java: 该对象是处理 HTTP 日期的辅助对象。

18. JsConfirmResult.java: Js 确认请求对象

19. JsPromptResult.java: Js 结果提示对象，用于向用户提示 Javascript 运行结果。

20. JsResult.java: Js 结果对象，用于用户交互

21. JWebCoreJavaBridge.java: 用 Java 与 WebCore 库中 Timer 和 Cookies 对象交互的桥接代码。

22. LoadListener.java: 载入器侦听器，用于处理载入器侦听消息。

23. Network.java: 该对象封装网络连接逻辑，为调用者提供更为高级的网络连接接口。

24. PanZoom.java: 用于处理图片缩放、移动等操作

25. PanZoomCellList.java: 用于保存移动、缩放图片的 Cell

26. PerfChecker.java: 用于效率测试的功能对象？

27. SslErrorHandler.java: 用于处理 SSL 错误消息。

28. StreamLoader.java: StreamLoader 抽象类是所有内容载入器对象的基类。该类是通过消息方式控制的状态机，用于将数据载入到 Frame 中。

29. TextDialog.java: 用于处理 html 中文本区域叠加情况，可以使用标准的文本编辑而定义的特殊 EditText 控件。

30. URLUtil.java: URL 处理功能函数，用于编码、解码 URL 字符串，以及提供附加的 URL 类型分析功能。

31. WebBackForwardList.java: 该对象包含 WebView 对象中显示的历史数据。

32. WebBackForwardListClient.java :浏览历史处理的客户接口类，所有需要接收浏览历史改变的类都需要实现该接口。

33. WebChromeClient.java: Chrome 客户基类，Chrome 客户对象在浏览器文档标题、进度条、图标改变时候会得到通知。

34. WebHistoryItem.java: 该对象用于保存一条网页历史数据

35. WebIconDataBase.java: 图表数据库管理对象，所有的 WebView 均请求相同的图标数据库对象。

36. WebSettings.java: WebView 的管理设置数据，该对象数据是通过 JNI 接口从底层获取。

37. WebSyncManager.java: 数据同步对象，用于 RAM 数据和 FLASH 数据的同步操作。

38. WebView.java: Web 视图对象，用于基本的网页数据载入、显示等 UI 操作。

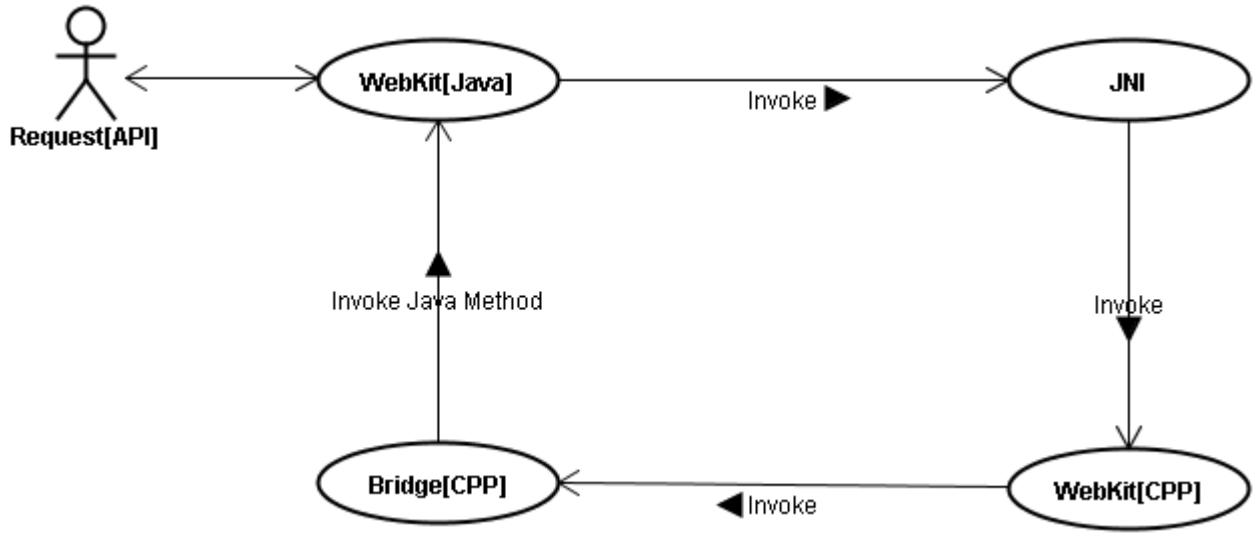
39. WebViewClient.java: Web 视图客户对象，在 Web 视图中有事件产生时，该对象可以获得通知。

40. WebViewCore.java: 该对象对 WebCore 库进行封装，将 UI 线程中的数据请求发送给 WebCore 处理，并通过 CallbackProxy 的方式，通过消息通知 UI 线程数据处理结果。

41. WebViewDatabase.java: 该对象使用 SQLiteDatabase 为 WebCore 模块提供数据存取操作。

三、WebKit 模块框架

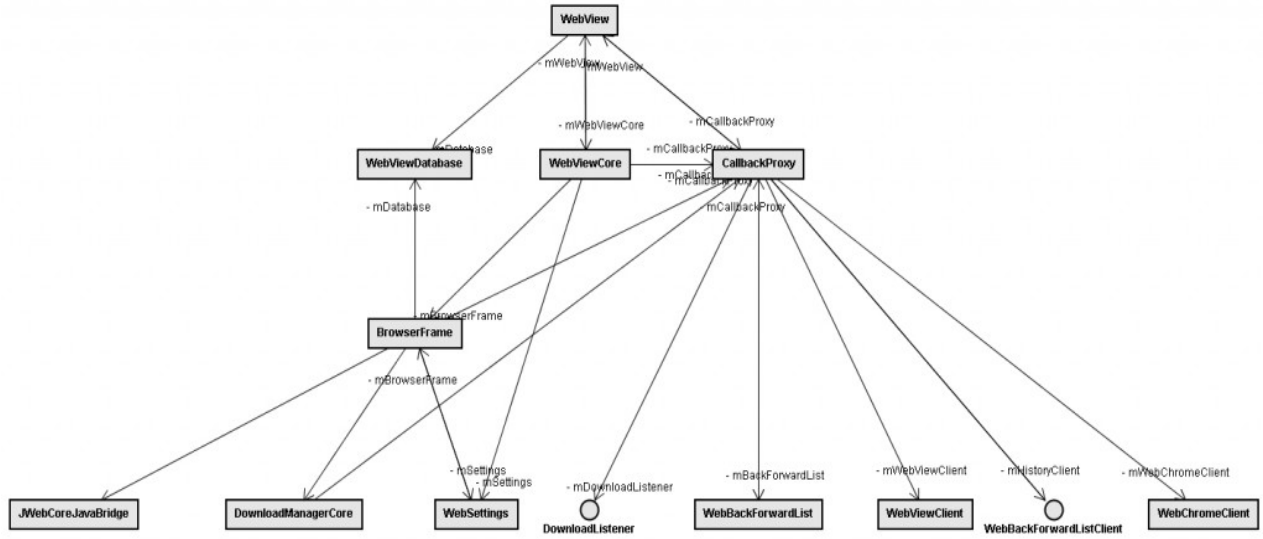
Android 平台的 WebKit 模块由 Java 层和 WebKit 库两个部分组成，Java 层负责与 Android 应用程序进行通信，而 WebKit 类库负责实际的网页排版处理。Java 层和 C 层库之间通过 JNI 和 Bridge 相互调用，如下图所示：



3.1 Java 层框架

3.1.1 主要类关系

WebKit 模块的 Java 层一共由 41 个文件组成，其中主要的类关系如下图所示：



1. WebView

WebView 类是 WebKit 模块 Java 层的视图类，所有需要使用 Web 浏览功能的 Android 应用程序都要创建该视图对象显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 javascript 请求。WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。

2. WebViewDatabase

WebViewDatabase 是 WebKit 模块中针对 SQLiteDatabase 对象的封装，用于存储和获取运行时浏览器保存的缓冲数据、历史访问数据、浏览器配置数据等。该对象是一个单实例对象，通过 getInstance 方

法获取 WebViewDatabase 的实例。WebViewDatabase 是 WebKit 模块中的内部对象，仅供 WebKit 框架内部使用。

3. WebViewCore

WebViewCore 类是 Java 层与 C 层 WebKit 核心库的交互类，客户程序调用 WebView 的网页浏览相关操作会转发给 BrowserFrame 对象。当 WebKit 核心库完成实际的数据分析和处理后会回调 WebViweCore 中定义的一系列 JNI 接口，这些接口会通过 CallbackProxy 将相关事件通知相应的 UI 对象。

4. CallbackProxy

CallbackProxy 是一个代理类，用于 UI 线程和 WebCore 线程交互。该类定义了一系列与用户相关的通知方法，当 WebCore 完成相应的数据处理，则会调用 CallbackProxy 类中对应的方法，这些方法通过消息方式间接调用相应处理对象的处理方法。详细的处理流程在下文中会具体分析。

5. BrowserFrame

BrowserFrame 类负责 URL 资源的载入、访问历史的维护、数据缓存等操作，该类会通过 JNI 接口直接与 WebKit C 层库交互。

6. JWebCoreJavaBridge

该类为 Java 层 WebKit 代码提供与 C 层 WebKit 核心部分的 Timer 和 Cookies 操作相关的方法。

7. DownloadManagerCore

下载管理核心类，该类负责管理网络资源下载，所有的 Web 下载操作均有该类同一管理。该类实例运行在 WebKit 线程当中，与 UI 线程的交互是通过调用 CallbackProxy 对象中相应的方法完成。

8. WebSettings

该对象描述了 WEB 浏览器访问相关的用户配置信息。

9. DownloadListener

下载侦听接口，如果客户代码实现该接口，则在下载开始、失败、挂起、完成等情况下，DownloadManagerCore 对象会调用客户代码中实现的 DownloadListener 方法。

10. WebBackForwardList

WebBackForwarList 对象维护着用户访问历史记录，该类为客户程序提供操作访问浏览器历史数据的相关方法。

11. WebViewClient

WebViewClient 类定义了一系列事件方法，如果 Android 应用程序设置了 WebViewClient 派生对象，则在页面载入、资源载入、页面访问错误等情况发生时，该派生对象的相应方法会被调用。

12. WebBackForwardListClient

WebBackForwardListClient 对象定义了对访问历史操作时可能产生的事件接口，当用户实现了该接口，则在操作访问历史时（访问历史移除、访问历史清空等）用户会得到通知。

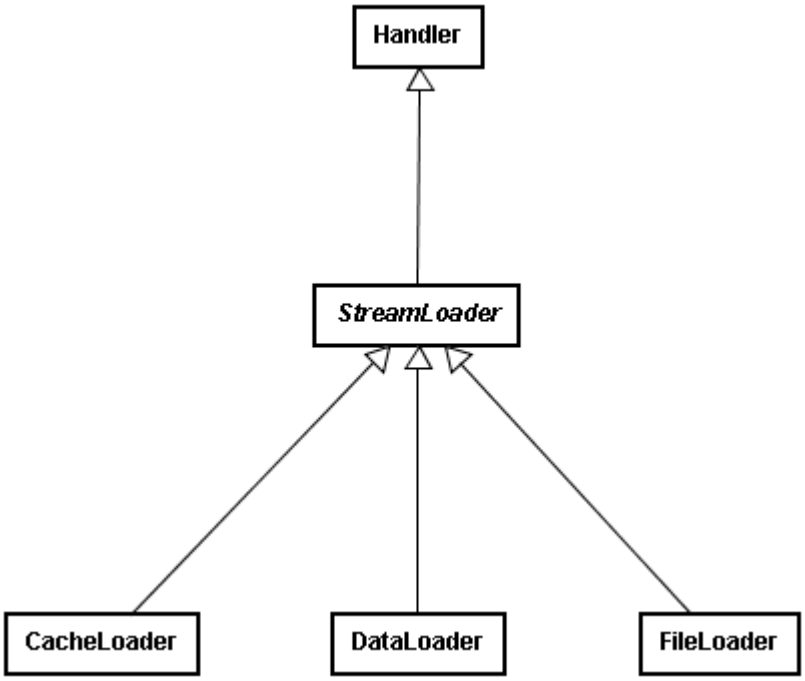
13. WebChromeClient

WebChromeClient 类定义了与浏览窗口修饰相关的事件。例如接收到 Title、接收到 Icon、进度变化时，WebChromeClient 的相应方法会被调用。

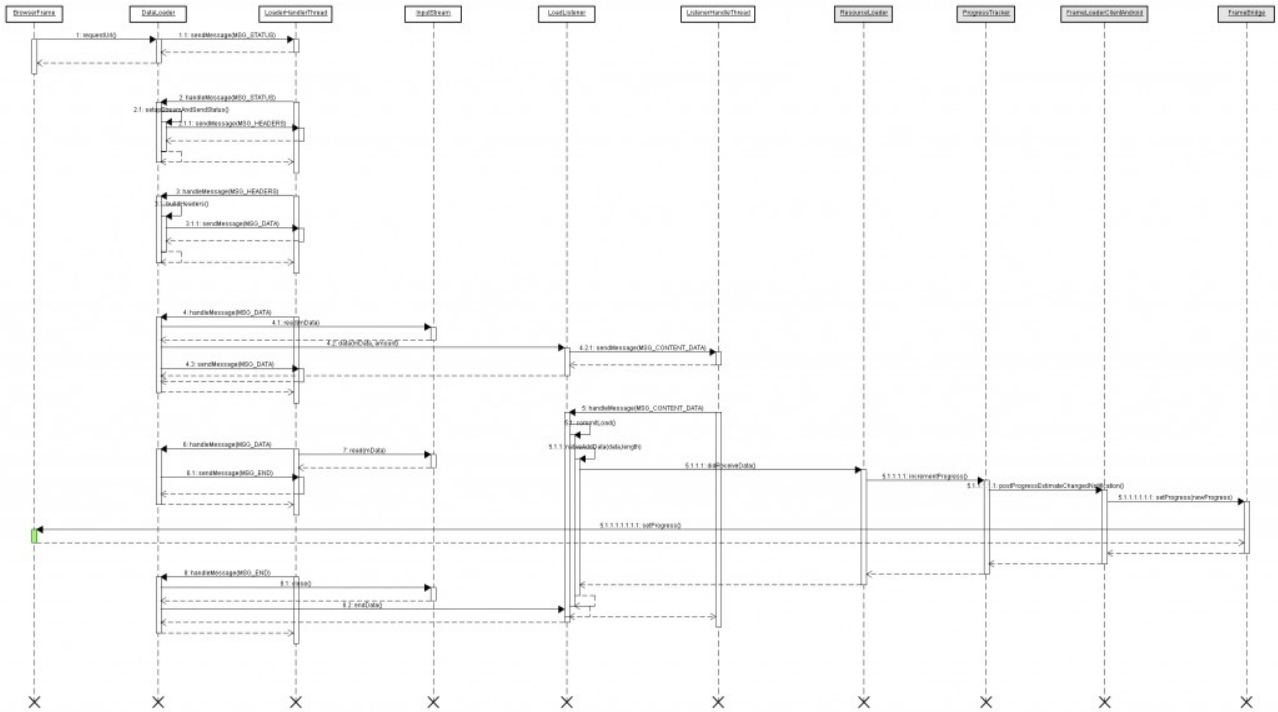
3.1.2 主要类的设计

3.1.2.1 数据载入器的设计

WebKit 模块的 Java 部分框架中使用数据载入器来加载相应类型的数据，目前有 CacheLoader、DataLoader 以及 FileLoader 三类载入器，他们分别用于处理缓存数据、内存据，以及文件数据的载入操作。Java 层（WebKit 模块）所有的载入器都从 StreamLoader 继承（其父类为 Handler），由于 StreamLoader 类的基类为 Handler 类，因此在构造载入器时，会开启一个事件处理线程，该线程负责实际的数据载入操作，而请求线程通过消息的方式驱动数据的载入。下图是数据载入器相关类的类图结构：



StreamLoader 类定义了 4 个不同的消息（MSG_STATUS、MSG_HEADERS、MSG_DATA、MSG_END），分别表示发送状态消息、发送消息头消息、发送数据消息以及数据发送完毕消息。该类提供了 2 个抽象保护方法以及一个共有方法：setupStreamAndSendStatus 保护方法主要是用于构造与通信协议相关的数据流，以及向 LoadListener 发送状态。buildHeaders 方法是向子类提供构造特定协议消息头功能。所有载入器只有一个共有方法（load），因此当需要载入数据时，调用该方法即可。与数据载入流程相关的类还有 LoaderListener 以及 BrowserFrame，当数据载入事件发生时， WebKit C 库会更新载入进度，并且会通知 BrowserFrame，BroserFrame 接收到进度条变更事件后会通过 CallbackProxy 对象，通知 View 类进度条数据变更。下面以 DataLoader 类为例子，说明数据载入以及与 UI 交互过程：



上图中绿色部分是 BrowserFrame 处理进度变更事件时，调用 CallbackProxy 对象通知视图变更状态的操作，在这里省略。途中灰色部分表示 C 层代码，而白色部分表示 Java 层代码。

3.2 C 层框架

3.2.1 C 类与 Java 类的关系

1．BrowserFrame

与 BrowserFrame Java 类相对应的 C++类为 FrameBridge，该类为 Dalvik 虚拟机回调 BrowserFrame 类中定义的本地方法进行了封装。与 BrowserFrame 中回调函数（Java 层）相对应的 C 层结构定义如下：

```
struct FrameBridge::JavaBrowserFrame
{
    JavaVM*     mJVM;
    jobject     mObj;
    jmethodID   mStartLoadingResource;
    jmethodID   mLoadStarted;
    jmethodID   mUpdateHistoryForCommit;
    jmethodID   mUpdateCurrentHistoryData;
    jmethodID   mReportError;
    jmethodID   mSetTitle;
    jmethodID   mWindowObjectCleared;
    jmethodID   mSetProgress;
    jmethodID   mDidReceiveIcon;
    jmethodID   mUpdateVisitedHistory;
    jmethodID   mHandleUrl;
    jmethodID   mCreateWindow;
    jmethodID   mCloseWindow;
    jmethodID   mDecidePolicyForFormResubmission;
};
```

该结构作为 FrameBridge（C 层）的一个成员变量（mJavaFrame），在 FrameBridge 构造函数中，

用 BrowserFrame（Java 层）类的回调方法的偏移量初始化 JavaBrowserFrame 结构的各个域。初始后，当 WebCore（C 层）在剖析网页数据时，有 Frame 相关的资源改变，比如 WEB 页面的主题变化，则会通过 mJavaFrame 结构，调用指定 BrowserFrame 对象的相应方法，通知 Java 层处理。

2．JWebCoreJavaBridge

与该对象相对应的 C 层对象为 JavaBridge，JavaBridge 对象继承了 TimerClient 和 CookieClient 类，负责 WebCore 中的定时器和 Cookie 管理。与 Java 层 JWebCoreJavaBridge 类中方法偏移量相关的是 JavaBridege 中几个成员变量，在构造 JavaBridge 对象时，会初始化这些成员变量，之后有 Timer 或者 Cookies 事件产生，WebCore 会通过这些 ID 值，回调对应 JWebCoreJavaBridge 的相应方法。

3．LoadListener

与该对象相关的 C 层结构是 struct resourceloader_t，该结构保存了 LoadListener 对象 ID、CancelMethod ID 以及 DownloadFiledMethod ID 值。当有 Cancel 或者 Download 事件产生，WebCore 会回调 LoadListener 类中的 CancelMethod 或者 DownloadFileMethod。

4．WebViewCore

与 WebViewCore 相关的 C 类是 WebCoreViewImpl，WebViewCoreImpl 类有个 JavaGlue 对象作为成员变量，在构建 WebCoreViewImpl 对象时，用 WebViewCore（Java 层）中的方法 ID 值初始化该成员变量。并且会将构建的 WebCoreViewImpl 对象指针复制给 WebViewCore（Java 层）的 mNativeClass，这样将 WebViewCore（Java 层）和 WebViewCoreImple（C 层）关联起来。

5．WebSettings

与 WebSettings 相关的 C 层结构是 struct FieldIds，该结构保存了 WebSettings 类中定义的属性 ID 以及方法 ID，在 WebCore 初始化时（WebViewCore 的静态方法中使用 System.loadLibrary 载入）会设置这些方法和属性的 ID 值。

6．WebView

与 WebView 相关的 C 层类是 WebViewNative，该类中的 mJavaGlue 中保存着 WebView 中定义的属性和方法 ID，在 WebViewNative 构造方法中初始化，并且将构造的 WebViewNative 对象的指针，赋值给 WebView 类的 mNativeClass 变量，这样 WebView 和 WebViewNative 对象建立了关系。

3.2.2 主要类关系

与 Java 层相关的 C 层类如下表所示：
类　　功能描述

(1) ChromeClientAndroid
该类主要处理 WebCore 中与 Frame 装饰相关的操作。例如设置状态栏、滚动条、Javascript 脚本提示框等。当浏览器中有相关事件产生，ChromeClientAndroid 类的相应方法会被调用，该类会将相关的 UI 事件通过 Bridge 传递给 Java 层，由 Java 层负责绘制以及用户交互方面的处理。

(2) EditorClientAndroid
该类负责处理页面中文本相关的处理，比如文本输入、取消、输入法数据处理、文本黏贴、文本编辑等操作。不过目前该类只对按键相关的时间进行了处理，其他操作均未支持。

(3) ContextMenuClient
该类提供页面相关的功能菜单，比如图片拷贝、朗读、查找等功能。但是，目前项目中未实现具体功能。

(4) DragClient
该类定义了与页面拖拽相关的处理，但是目前该类没有实现具体功能。

(5) FrameLoaderClientAndroid
该类提供与 Frame 加载相关的操作，当用户请求加载一个页面时，WebCore 分析完网页数据后，会通过该类调用 Java 层的回调方法，通知 UI 相关的组件处理。

(6) InspectorClientAndroid
该类提供与窗口相关的操作，比如窗口显示、关闭窗口、附加窗口等。不过目前该类的各个方法均为空实现。

以上六个类的实现均可以在 external/webkit/WebKit/android/WebCoreSupport 下找到。

(7) Page
该类提供与页面相关的操作，比如网页页面的前进、后退等操作。

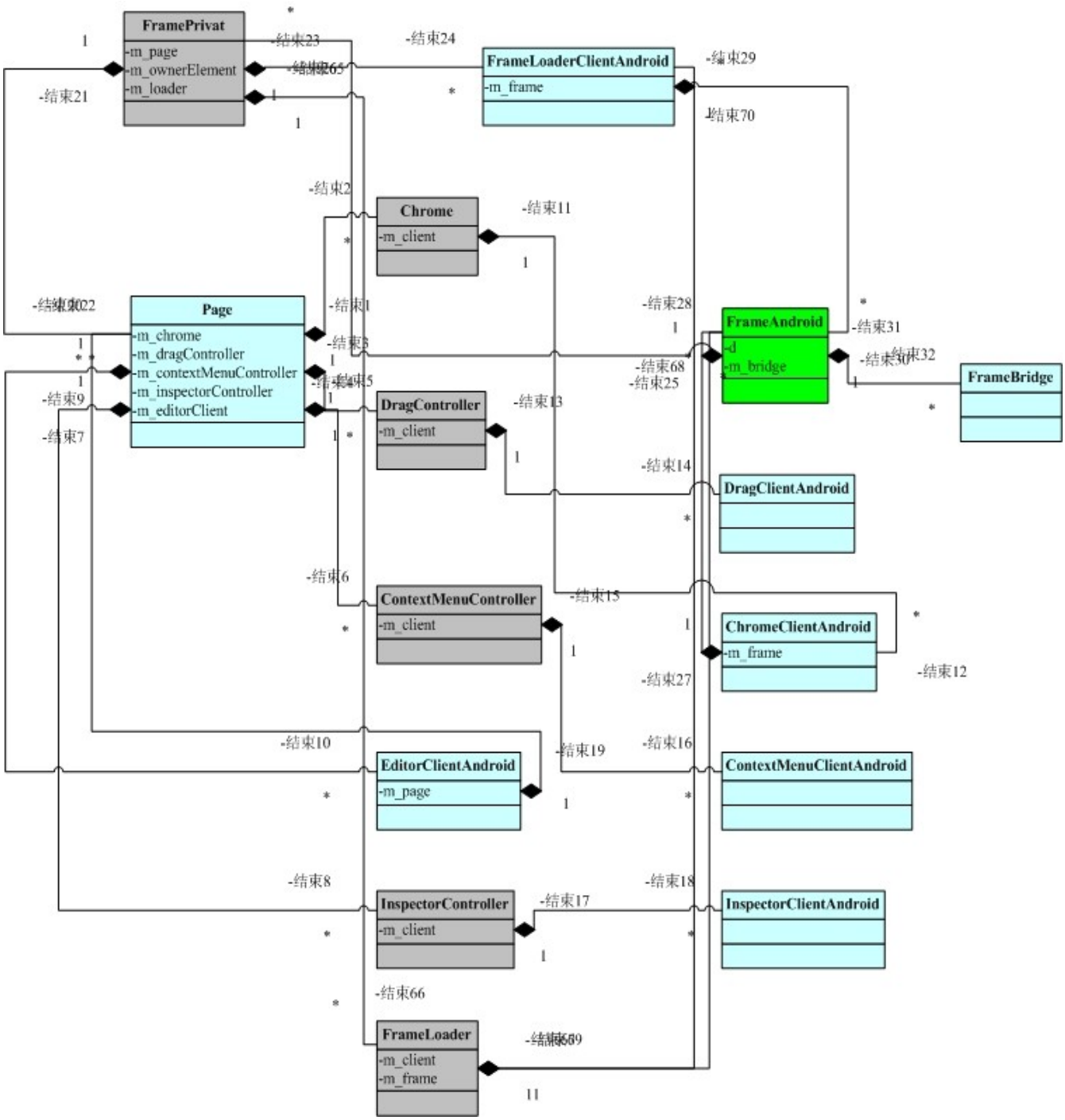
(8) FrameAndroid
该类为 Android 提供 Frame 管理。

(9) FrameBridge
该类对 Frame 相关的 Java 层方法进行了封装，当有 Frame 事件产生时，WebCore 通过 FrameBridge 回调 Java 的回调函数，完成用户交互过程。

(10) AssetManager
该类为浏览器提供本地资源访问功能。

(11) RenderSkinAndroid
该类与控件绘制相关，所有的须绘制控件都需要从该类派生，目前 WebKit 模块中有 Button、Combo、Radio 三类控件。

以上几个类会在 Java 层请求创建 Web Frame 的时候被建立，他们的关系如下图所示：



上图中标注为深绿色的 FrameAndroid 是浏览器 Frame，一个 BrowserFrame 对象对应着一个 FrameAndroid 对象。而其他 8 个标注为淡绿色的类，是与该 Frame 显示、布局等相关的类。WebKit 模块中所有 WebCore 核心代码与用户交互的操作使用 FrameAndroid 对象中的 Bridge 处理（回调相应的 Java 方法）。

四、基本操作分析

4.1 WebKit 模块初始化

Android SDK 中提供了 WebView 类，该类为客户提供客户化浏览显示的功能，如果客户需要加入浏览器的支持，可将该类的实例或者派生类的实例作为视图，调用 Activity 类的 setContentView 显示给用户。当客户代码中生成第一次生成 WebView 对象时，会初始化 WebKit 库（包括 Java 层和 C 层两个部分），之后用户可以操作 WebView 对象完成网络或者本地资源的访问。

WebView 对象的生成主要涉及 3 个类 CallbackProxy、WebViewCore 以及 WebViewDatabase。其中 CallbackProxy 对象为 WebKit 模块中 UI 线程和 WebKit 类库提供交互功能，WebViewCore 是 WebKit 的核心层，负责与 C 层交互以及 WebKit 模块 C 层类库初始化，而 WebViewDatabase 为 WebKit 模块运行时缓存、数据存储提供支持。WebKit 模块初始化流程如下：

WebView

+–创建 CallbackProxy 对象

+–创建 WebViewCore 对象

1–调用 System.loadLibrary 载入 webcore 相关类库（C 层）

2–如果是第一次初始化 WebViewCore 对象，创建 WebCoreTherad 线程

3–创建 EventHub 对象，处理 WebViewCore 事件

4–获取 WebIconDatabase 对象实例

5–向 WebCoreThread 发送初始化消息

+–获取 WebViewDatabase 实例

如上所叙，第一步调用 System.loadLibrary 方法载入 webcore 相关类库，该过程由 Dalvik 虚拟机完成，它会从动态链接库目录中寻找 libWebCore.so 类库，载入到内存中，并且调用 WebKit 初始化模块的 JNI_OnLoad 方法。WebKit 模块的 JNI_OnLoad 方法中完成了如下初始化操作：

a) 初始化 framebridge[register_android_webcore_framebridge]

初始化 gFrameAndroidField 静态变量，以及注册 BrowserFrame 类中的本地方法表。

b) 初始化 javabridge[register_android_webcore_javabridge]

初始化 gJavaBridge.mObject 对象，以及注册 JWebCoreJavaBridge 类中的本地方法

c) 初始化资源 loader[register_android_webcore_resource_loader]

初始化 gResourceLoader 静态变量，以及注册 LoadListener 类的本地方法

d) 初始化 webviewcore[register_android_webkit_webviewcore]

初始化 gWebCoreViewImplField 静态变量，以及注册 WebViewCore 类的本地方法

e) 初始化 webhistory[register_android_webkit_webhistory]

初始化 gWebHistoryItem 结构，以及注册 WebBackForwardList 和 WebHistoryItem 类的本地方法

f) 初始化 webicondatabase[register_android_webkit_webicondatabase]

注册 WebIconDatabase 类的本地方法

g) 初始化 websettings[register_android_webkit_websettings]

初始化 gFieldIds 静态变量，以及注册 WebSettings 类的本地方法

h) 初始化 webview[register_android_webkit_webview]

初始化 gWebViewNativeField 静态变量，以及注册 WebView 类的本地方法

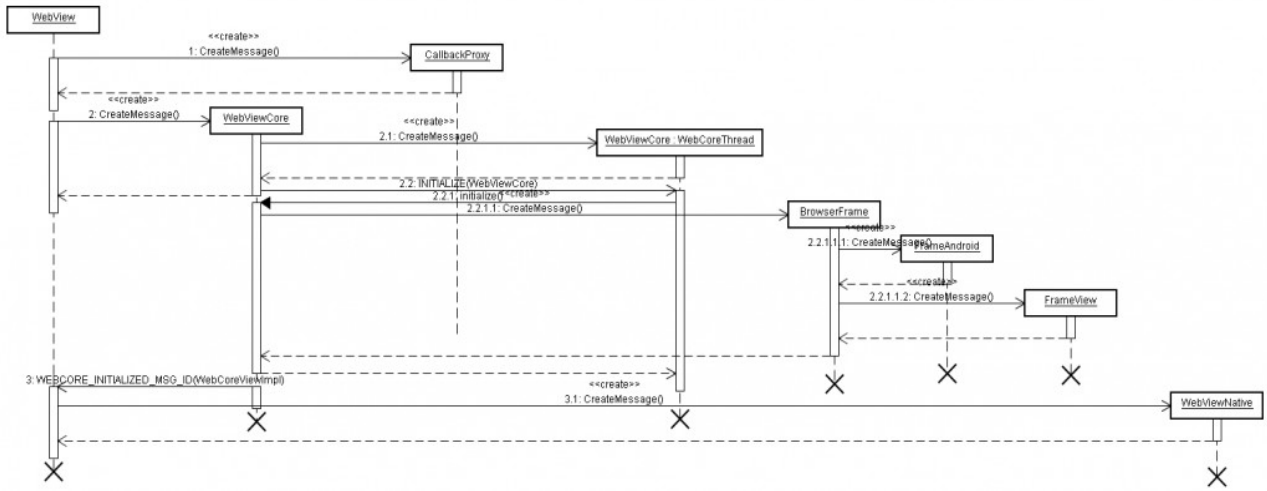
第二步是 WebCoreThread 初始化，该初始化只在第一次创建 WebViewCore 对象时完成，当用户代码第一次生成 WebView 对象，会在初始化 WebViewCore 类时创建 WebCoreThread 线程，该线程负责处理 WebCore 初始化事件。此时 WebViewCore 构造函数会被阻塞，直到一个 WebView 初始化请求完毕时，会在 WebCoreThread 线程中唤醒。

第三步创建 EventStub 对象，该对象处理 WebView 类的事件，当 WebCore 初始化完成后会向 WebView 对象发送事件，WebView 类的 EventStub 对象处理该事件，并且完成后续初始化工作。

第四步获取 WebIconDatabase 对象实例。

第五步向 WebViewCore 发送 INITIALIZE 事件，并且将 this 指针作为消息内容传递。WebView 类主要负责处理 UI 相关的事件，而 WebViewCore 主要负责与 WebCore 库交互。在运行时期，UI 线程和 WebCore 数据处理线程是运行在两个独立的线程当中。WebCoreThread 线程接收到 INITIALIZE 线程后，会调用消息对象参数的 initialize 方法，而后唤醒阻塞的 WebViewCore Java 线程（该线程在 WebViewCore 的构造函数中被阻塞）。不同的 WebView 对象实例有不同的 WebViewCore 对象实例，因此通过消息的方式可以使得 UI 线程和 WebViewCore 线程解耦合。WebCoreThread 的事件处理函数，处理 INITIALIZE 消息时，调用的是不同 WebView 中 WebViewCore 实例的 initialize 方法。WebViewCore 类中的 initialize 方法中会创建 BrowserFrame 对象（该对象管理整个 WEB 窗体，以 frame 相关事件），并且向 WebView 对象发送 WEBCORE_INITIALIZED_MSG_ID 消息。WebView 消息处理函数，会根据消息参数 1 初始化指定的 WebViewCore 对象，并且更新 WebViewCore 的 Frame 缓冲。

初始化过程的序列图如下图所示：



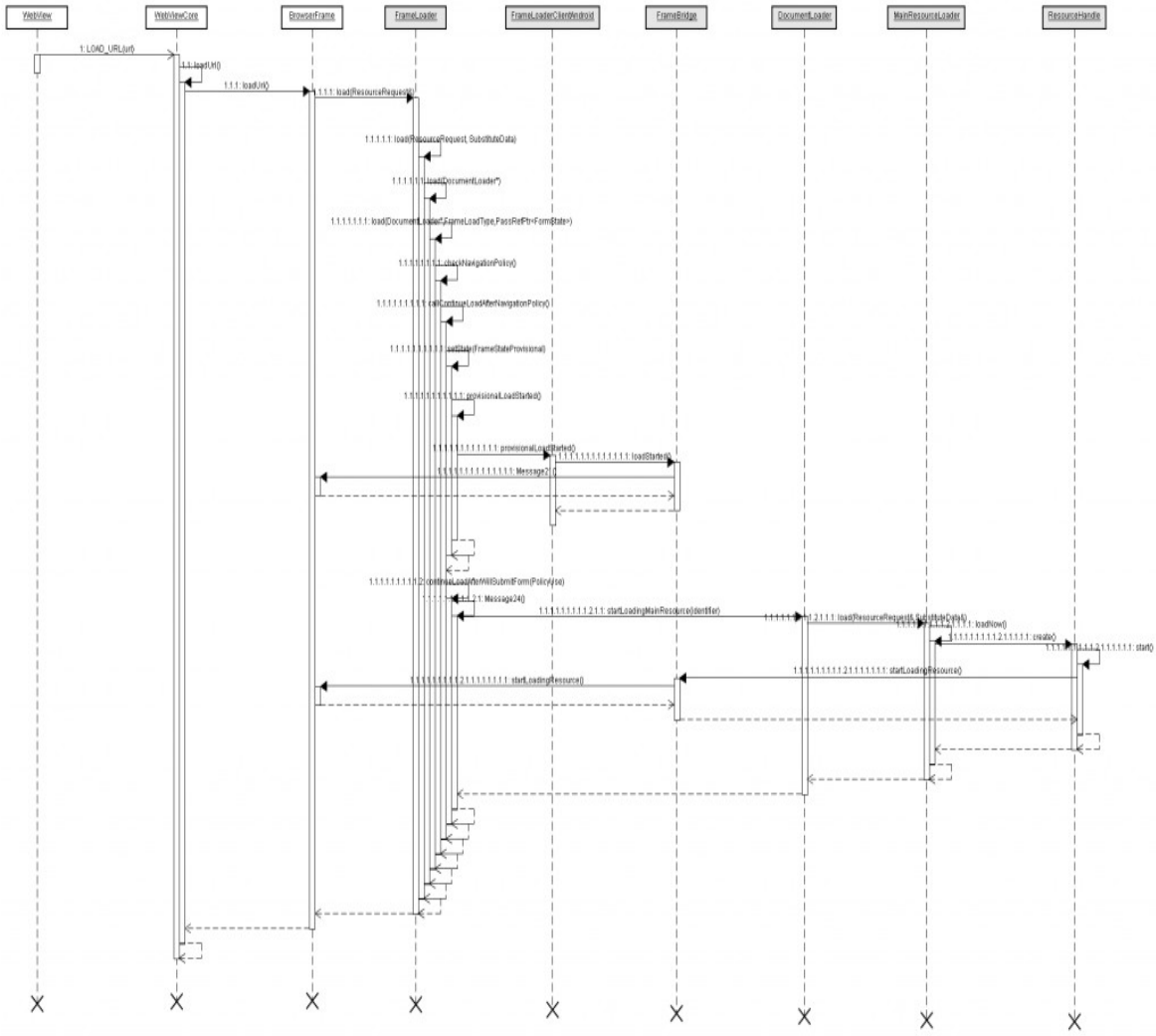
初始化完成后 Java 层和 C 层类图关系如下图所示:

上图中淡绿色的类表示 Java 层，而灰色类表示 C 层。

4.2 数据载入

4.2.1 载入网络数据

客户代码中可以使用 WebView 类的 loadUrl 方法，请求访问指定的 URL 网页数据。WebView 对象中保存着 WebViewCore 的引用，由于 WebView 属于 UI 线程，而 WebViewCore 属于后台线程，因此 WebView 对象的 loadUrl 被调用时，会通过消息的方式将 URL 信息传递给 WebViewCore 对象，该对象会调用成员变量 mBrowserFrame 的 loadUrl 方法，进而调用 WebKit 库完成数据的载入。其调用函数序列如下所示：



网络数据的载入分别由 Java 层和 C 层共同完成，Java 层完成用户交互、资源下载等操作，而 C 层主要完成数据分析（建立 DOM 树、分析页面元素等）操作。由于 UI 线程和 WebCore 线程运行在不同的两个线程中，因此当用户请求访问网络资源时，通过消息的方式向 WebViewCore 对象发送载入资源请求。在 Java 层的 WebKit 模块中，所有与资源载入相关的操作都是由 BrowserFrame 类中对应的方法完成，这些方法是本地方法，会直接调用 WebCore 库的 C 层函数完成数据载入请求，以及资源分析等操作。如上图所示，C 层的 FrameLoader 类是浏览框架的资源载入器，该类负责检查访问策略以及向 Java 层发送下载资源请求等功能。在 FrameLoader 中，当用户请求网络资源时，经过一系列的策略检查后会调用 FrameBridge 的 startLoadingResource 方法，该方法会回调 BrowserFrame（Java）类的 startLoadingResource 方法，完成网络数据的下载，而后 BrowserFrame （Java）类的 startLoadingResource 方法会返回一个 LoadListener 的对象，FrameLoader 会删除原有的 FrameLoader 对象，将 LoadListener 对象封装成 ResourceLoadHandler 对象，并且将其设置为新的 FrameLoader。到此完成了一次资源访问请求，接下来的任务即是 WebCore 库会根据资源数据进行分析和构建 DOM，以及相关的数据结构。

4.2.2 载入本地数据

本地数据是以 data://开头的 URL 表示，载入过程和网络数据一样，只不过在执行 FrameLoader 类的 executeLoad 方法时，会根据 URL 的 SCHEME 类型区分，调用 DataLoader 的 requestUrl 方法（参看 3.1.2.1 节对载入器的分析），而不是调用 handleHTTPLoad 建立实际的网络通信连接。

4.2.3 载入文件数据

文件数据是以 file://开头的 URL，载入的基本流程与网络数据载入流程基本一致，不同的是在运行 FrameLoader 类的 executeLoad 方法时，根据 SCHEME 类型，调用 FileLoader 的 requestUrl 方法，完成数据加载（参看 3.1.2.1 节对载入器的分析）。

4.3 刷新绘制

当用户拖动滚动条、有窗口遮盖、或者有页面事件触发都会向 WebViewCore（Java 层）对象发送背景重绘消息，该消息会引起网页数据的绘制操作。WebKit 的数据绘制可能出于效率上的考虑，没有通过 Java 层，而是直接在 C 层使用 SGL 库完成。与 Java 层图形绘制相关的 Java 对象有如下几个：

Picture 类

该类对 SGL 封装，其中变量 mNativePicture 实际上是保存着 SkPicture 对象的指针。WebViewCore 中定义了两个 Picture 对象，当作双缓冲处理，在调用 webKitDraw 方法时，会交换两个缓冲区，加速刷新速度。

WebView 类

该类接受用户交互相关的操作，当有滚屏、窗口遮盖、用户点击页面按钮等相关操作时，WebView 对象会与之相关的 WebViewCore 对象发送 VIEW_SIZE_CHANGED 消息。当 WebViewCore 对象接收到该消息后，将构建时建立的 mContentPictureB 刷新到屏幕上，然后将 mContentPictureA 与之交换。

WebViewCore 类

该类封装了 WebKit C 层代码，为视图类提供对 WebKit 的操作接口，所有对 WebKit 库的用户请求均由该类处理，并且该类还为视图类提供了两个 Picture 对象，用于图形数据刷新。

下面以 Web 页面被鼠标拖拽的情况为例子，分析网页数据刷新过程。当用户使用手指点击触摸屏，并且移动手指，则会引发 touch 事件的产生，Android 平台会将 touch 事件传递给最前端的视图相应（dispatchTouchEvent 方法处理）。在 WebView 类中定义了 5 种 touch 模式，在手指拖动 Web 页面的情况下，会触发 mMotionDragMode，并且会调用 View 类的 scrollBy 方法，触发滚屏事件以及使视图无效（重绘，会调用 View 的 onDraw 方法）。WebView 视图中的滚屏事件由 onScrollChanged 方法响应，该方法向 WebViewCore 对象发送 SET_VISIBLE_RECT 事件。

WebViewCore 对象接收到 SET_VISIBLE_RECT 事件后，将消息参数中保存的新视图的矩形区域大小传递给 nativeSetVisibleRect 方法，通知 WebCoreViewImpl 对象（C 层）视图矩形变更（WebCoreViewImpl::setVisibleRect 方法）。在 setVisibleRect 方法中，会通过虚拟机调用 WebViewCore 的 contentInvalidate 方法，该方法会引发 webkitDraw 方法的调用（通过 WEBKIT_DRAW 消息）。在 webkitDraw 方法里，首先会将 mContentPictureB 对象传递给本地方法 nativeDraw 绘制，而后将 mContentPictureB 的内容与 mContentPictureA 的内容对调。在这里 mContentPictureA 缓冲区是供给 WebViewCore 的 draw 方法使用，如果用户选择某个控件，绘制焦点框时候 WebViewCore 对象的 draw 方法会调用，绘制的内容保存在 mContentPictureA 中，之后会通过 Canvas 对象（Java 层）的 drawPicture 方法将其绘制到屏幕上，而 mContentPictureB 缓冲区是用于 built 操作的，nativeDraw 方法中首先会将传递的 mContentPictureB 对象数据重置，而后在重新构建的 mContentPictureB 画布上，将层上相关的元素绘制到该画布上。上面提到，之后会将 mContentPictureB 和 mContentPictureA 的内容对调，这样一次重绘事件产生时（会调用 WebView.onDraw 方法）会将 mContentPictureA 的数据使用 Canvas 类的 drawPicture 绘制到屏幕上。当 webkitDraw 方法将 mContentPictureA 与 mContentPictureB 指针对调后，会向 WebView 对象发送 NEW_PICTURE_MSG_ID 消息，该消息会引发 WebViewCore 的

VIEW_SIZE_CHANGED 消息的产生，并且会使当前视图无效产生重绘事件(invalidate())，引发 onDraw 方法的调用，完成一次网页数据的绘制过程。

原文摘自：
http://littlefermat.blog.163.com/blog/static/5977116720106268453093 9/