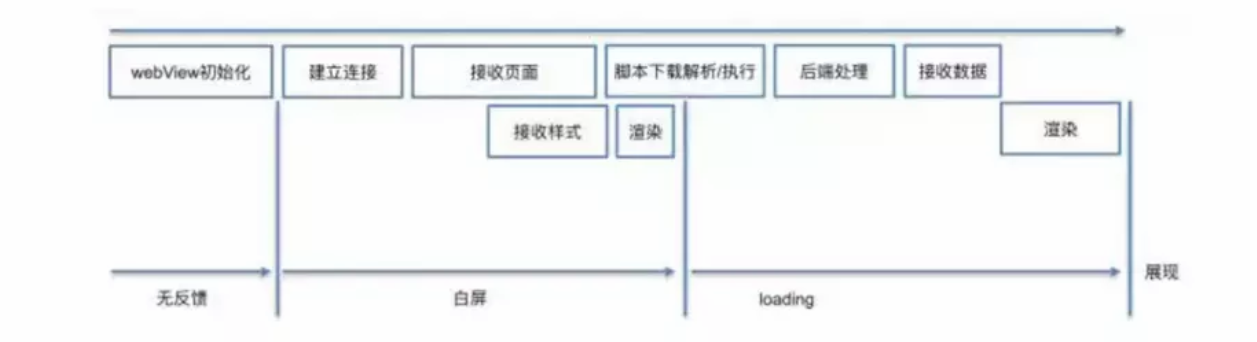
Android WebView优化

对于WebView的性能，给人最直观的莫过于：打开速度比native慢。当我们打开一个WebView页面，页面往往会慢吞吞的loading很久，若干秒后才出现你所需要看到的页面。

对于一个普通用户来讲，打开一个WebView通常会经历以下几个阶段：

1. 交互无反馈
2. 到达新的页面，页面白屏
3. 页面基本框架出现，但是没有数据；页面处于loading状态
4. 出现所需的数据

如果从程序上观察，WebView启动过程大概分为以下几个阶段：



Markdown

在流量常驻通知栏apk中打开小飞读报的一个链接

大致分析一下：

* 整个页面从进入到加载完成消耗1800ms，将近2s。
* 从进入WebView页面到开始加载网页链接内容大致消耗: 200ms
* 从开始加载网页链接内容到加载加载结束大致消耗：1600ms

这中间请求网页资源可以大致拆分一下：

* + 网页css和js请求大概消耗400ms
  + 网页各种图片的加载大致消耗1200ms

如何缩短这些过程的时间，就成了优化WebView性能的关键。接下来我们逐一分析各个阶段的耗时情况，以及需要注意的优化点。

(1)webview初始化优化

针对WebView的初始化时间，我们从以下几个方面入手：

* 首次初始化时间：客户端冷启动后，第一次打开WebView，从开始创建WebView到开始建立网络连接之间的时间。
* 二次初始化时间：在打开过WebView后，退出WebView，再重新打开WebView，从开始创建WebView到开始建立网络连接之间的时间。
* 初始化完毕后，WebView加载冗余内容导致的缓慢。如一些复杂的css文件，js文件，png图片等。

#### 1 .WebView初始化时间（）

​ 一般我们测量一个网页打开的快慢为标准，都是以网络连接开始作为起点的。但是WebView中用户体验到的打开时间需要再增加70~700ms。这是因为webview需要进行一系列的初始化操作。这也是为什么在Webview中会感觉慢。

* 在浏览器中，我们输入地址时（甚至在之前），浏览器就可以开始加载页面。
* 而在客户端中，客户端需要先花费时间初始化WebView完成后，才开始加载。而这段时间，由于WebView还不存在，所有后续的过程是完全阻塞的。

优化方法：

​ 由于这段过程发生在native的代码中，单纯靠前端代码是无法优化的；大部分的方案都是前端和客户端协作完成。

1.1**全局WebView**

​ 方法：

* 在客户端刚启动时，就初始化一个全局的WebView待用，并隐藏；
* 当用户访问了WebView时，直接使用这个WebView加载对应网页，并展示。

这种方法可以比较有效的减少WebView在App中的首次打开时间。当用户访问页面时，不需要初始化WebView的时间。

缺点：

* 额外的内存消耗。
* 页面间跳转需要清空上一个页面的痕迹，更容易内存泄露。

1.2.**客户端代理数据请求**

方法：

* 在客户端初始化WebView的同时，直接由native开始网络请求数据；
* 当页面初始化完成后，向native获取其代理请求的数据。

这个方法虽然不能减小WebView初始化时间，但数据请求和WebView初始化可以并行进行，总体的页面加载时间就缩短了；缩短总体的页面加载时间。

#### 2.与服务器连接的建立

在页面请求的数据返回之前，主要有以下过程耗费时间。

* DNS
* connection
* 服务器处理

优化方案：

DNS采用和客户端API相同的域名，当我们初次打开App时：

* 客户端首次打开都会请求hflldr.voicecloud.cn，其DNS将会被系统缓存。
* 然而当打开WebView的时候，由于请求了不同的域名，需要重新获取xiaofeiddubao.com的IP。

如果WebView的域名与App的API域名统一，则可以让WebView的DNS时间全部达到1.3ms的量级。静态资源同理，最好与客户端的资源域名保持一致。

#### 3.页面加载和渲染

上面小飞读报加载时可以看到一大部分时间都花在了图片资源的请求上，因此好的解决方案就是延迟这些图片的加载，先加载并展示非图片内容。

解决方案：WebView有一个setting配置方法：setBlockNetworkImage(boolean) 该方法的作用是是否屏蔽图片的加载。可以利用这个方法来实现图片的延迟加载。

@Override

public void onPageStarted(WebView view, String url, Bitmap favicon) {

super.onPageStarted(view, url, favicon);

DebugLog.d(TAG,"onPageStarted: "+url);

mCommonWv.getSettings().setBlockNetworkImage(true);

}

@Override

public void onPageFinished(WebView view, String url) {

super.onPageFinished(view, url);

DebugLog.i(TAG ,"onPageFinished: "+url);

mCommonWv.getSettings().setBlockNetworkImage(false);

if (!mCommonWv.getSettings().getLoadsImagesAutomatically()) {

//设置wenView加载图片资源

mCommonWv.getSettings().setBlockNetworkImage(false);

mCommonWv.getSettings().setLoadsImagesAutomatically(true);

}

}

​

#### 4. 使用缓存

​ Android 支持对页面进行资源缓存，可以在下一次从新进入该页面时直接调用缓存资源，减少网络请求消耗时间。

WebSettings websetting = webview.getSettings();

websetting.setJavaScriptEnabled(true); //与js交互

websetting.setDomStorageEnabled(true); //开启DOM形式存储

websetting.setDatabaseEnabled(true); //开启数据库形式存储

String appCacheDir = this.getApplicationContext().getDir("cache", Context.MODE\_PRIVATE).getPath(); //缓存数据的存储地址

websetting.setAppCachePath(appCacheDir);

websetting.setAppCacheEnabled(true); //开启缓存功能

websetting.setCacheMode(WebSettings.LOAD\_CACHE\_ELSE\_NETWORK); //缓存模式

websetting.setAllowFileAccess(true);

websetting.setAppCacheMaxSize(size); //设置缓存文件大小，但现在已不再提倡这个方法

关于缓存模式，有五种，根据不同需求可以进行设置：

LOAD\_CACHE\_ONLY: 不使用网络，只读取本地缓存数据

LOAD\_DEFAULT: 根据cache-control决定是否从网络上取数据。

LOAD\_CACHE\_NORMAL: API level 17中已经废弃, 从API level 11开始作用同LOAD\_DEFAULT模式

LOAD\_NO\_CACHE: 不使用缓存，只从网络获取数据.

LOAD\_CACHE\_ELSE\_NETWORK，只要本地有，无论是否过期，或者no-cache，都使用缓存中的数据。

#### 5. 前端支持

​ 除此之外，前端页面也需要控制好资源的压缩，css，js的加载顺序等。