# 竞品分析

## 框架优化

l\_wang王磊/ffpan潘斐斐

# 1. 行业分析

## 1.1 综述

距离上次同业产品对比已过半年，这次将对上次对比过的网站再次分析，对比这半年来竞争对手的变化，知己知彼，方能百战不殆，取精去糟，方能能跑行业。

本次对比选择同行[艺龙](http://m.elong.com/)，[同程](http://touch.17u.com/)，[酷讯](http://m.kuxun.cn/)，[去哪儿](http://touch.qunar.com/)国内同行业产品，和国外[Ebay](http://m.ebay.com/)，[booking](http://www.amazon.com/gp/aw/h.html?ie=UTF8&pc_redir=1)，[一淘](http://www.expedia.com.hk/MobileHotel?langid=2057&semcid=gog_hk_expedia_brand&kword=expedia!e.ZzZz.2120000000141.0.32288984262.expedia.expedia&gclid=CKLU1pqelroCFYIn4godQ0kAaw)等知名电商，从产品，性能，用户体验等方面做对比。分析产品的不同前端架构、渲染方式、以获取行业发展最新动态，为Ctrip网站的优化或者特性的增减提供依据。

这里主要分析点会放在国内旅游网站的对比，对国外网站提供简单的分析，而国内又会主要分析携程与去哪儿的差距

## 1.2 业务线

时光俨然、半年已逝。互联网追求的是速度，不论是产品访问的速度还是业务拓展的速度，皆是分秒必争。所以我们第一步来看看各个竞争对手的业务发展。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **频道** | **携程** | **去哪儿** | **艺龙** | **同程** | **酷讯** |
| 机票 | ✔ | ✔ | ✔ | ✔ | ✔ |
| 酒店 | ✔ | ✔ | ✔ | ✔ | ✔ |
| 用车 | ✔ | ✔ | ✕ | ✕ | ✕ |
| 火车票 | ✔ | ✔ | ✔ | ✕ | ✔ |
| 团购 | ✔ | ✔ | ✔ | ✔ | ✕ |
| 旅游 | ✔ | ✔ | ✔ | ✔ | ✕ |
| 门票 | ✔ | ✔ | ✕ | ✔ | ✕ |
| 攻略社区 | ✔ | ✔ | ✔ | ✕ | ✕ |
| 用户中心 | ✔ | ✔ | ✔ | ✔ | ✕ |
| 金融支付 | ✔ | ✕ | ✕ | ✕ | ✕ |
| 夜销 | ✕ | ✔ | ✕ | ✕ | ✕ |
| 当地人 | ✕ | ✔ | ✕ | ✕ | ✕ |
| 热销榜 | ✕ | ✕ | ✕ | ✔ | ✕ |
| 最低价 | ✕ | ✕ | ✕ | ✔ | ✕ |
|  |  |  |  |  |  |

表1.2 同行产品业务线

从数据上看，业务线站在行内前面的依旧是携程与去哪儿，携程与去哪儿业务线功能基本持平，其中携程多了一个金融支付，但是去哪儿也有礼品卡，但是没有形成单独的频道。

去哪儿在酒店优惠方多了一个现实特惠项目，只能在晚上18.00开始预订，是一个特色点，与公司群狼项目有点类似，但是群狼没有用于H5站点。

去哪儿还单独有一个当地人频道，该功能被携程融合到了各个频道中。

同程的特点集中体现与门票与旅游，占了整个网站很大比重

## 1.3 技术详情

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目** | **携程** | **去哪儿** | **艺龙** | **同程** | **酷讯** |
| **单页** | ✔ | ✕ | ✕ | ✕ | ✕ |
| **框架** | RequireJS  Zepto  Underscore  Backbone | RequireJS  underscore  Zepto  扩展Zepto | 原生框架 | Jquery | Zepto |
| **结果**  **输出** | js模板 | 输出html  局部HTML模板 | 输出html | 输出html | 输出html |
| **SEO** | 1.1独立SEO  2.0整合SEO | SEO站点统一 | SEO站点统一 | SEO站点统一 | 单独SEO |
| **统计代码** | Google  UBT | 百度统计 | 无 | 百度统计 | Google统计  自带统计 |
| **统计代码加载方式** |  |  |  |  |  |

表1.3 同行产品技术一览

**模式**

以模式上看，只有携程使用了webapp模式，其余产品皆是采用服务器吐出静态HTML的方式。

webapp模式被认为是web架构的最终赢家，优势在于用户体验好，可以减轻服务器压力，消耗更少带宽，并且可以与Hybrid无缝结合，是移动端优先选择，主要缺点是SEO支持不好。

传统模式网站采用服务器完整吐出首屏HTML的方式，该方式实现较简单，SEO友好。

**框架**

以框架来说，携程使用requireJS管理文件，使用zepto操作dom、使用underscore做基础方法库、阉割Backbone实现自己的框架，并且实现自己的UI库，整体功能完善。

去哪儿使用zepto操作dom，使用requireJS组织文件，具有自己的方法库，并具有较复杂UI组件，功能较完善，因为不是单页，实现难度相对简单。

艺龙、同程、酷讯比较一致，基本都只是引入一个dom操作库，从业务功能以及组件功能来说，都相对简单，移动站点规模比起携程和去哪儿相差甚远。

**SEO**

携程1.1 SEO采用独立站点做支撑，SEO与webapp分离，2.0框架处理了SEO问题，正在线上做验证。

由于其它同行产品皆是服务器吐出完整静态HTML，所以SEO本身支持较好，除了酷讯未做特殊处理。

**统计代码**

携程、酷讯采用Google分析、以及内部系统做用户行为分析，去哪儿、同程采用百度统计，而艺龙未做站点分析

## 1.4 其它产品技术详情

|  |  |  |  |
| --- | --- | --- | --- |
| **项目** | **Booking** | **ebay** | **一淘** |
| **单页** | ✕ | ✕ | ✕ |
| **框架** | jquery | require  jquery | SaeJS  Zepto  Backbone |
| **结果**  **输出** | 输出HTML | 输出HTML | 输出HTML |
| **SEO** | 统一SEO | 统一SEO | 统一SEO |
| **统计代码** | 1. ga  2. dobuleclick  3. utag(tealium.com)  4. bttlcheck | 自带 | 淘宝自带 |

表1.4 其它产品技术一览

国外的变化不大，一淘发生了很大的变化，由原来的单页应用变成了服务器吐出HTML的模式，具体缘由与产生的效果不明。

## 1.5 去哪儿ipad版本

去哪儿新出了一套webapp模式的ipad版本，说是ipad版本，其实使用体验更加贴近手机，其实现与携程比较类似，框架引用也很类似requireJS、underscore、Backbone一个不少，dom操作库换成了jquery。

框架实现了一套完整的UI组件，满足一个频道的基本开发，view切换带有动画，只不过暂时未区分前进后退，并且切换会有白页的现象；对移动端的点击做了处理，响应迅速。

该框架相同请求返回的数据未做缓存，所以请求的数据立即再次请求也会发送请求。

所以整个来说，去哪儿pad版本，大的模块出来了，细节点处理不太细腻，可以看做是移动端webapp的尝试，假以时日旅游行业可能会出现两个webapp了。

## 1.6 单页or多页

**SPA优缺点**

SPA（single page application），即单页webapp，它具有以下优点：

* 用户体验，对于内容的改动不需要加载整个页面。这样不会出现白页情况，页面与页面无缝切换，甚至带有一定动画效果。
* 请求量少，请求内容无需服务器解析，对服务器压力较小，消耗更少的带宽。

当然，单页应用也不是完美无瑕的，他也具有以下问题：

* 由于历史原因，单页应用对SEO支持不是太好，需要对SEO做特殊处理。
* 首次加载量过大，首屏加载慢，所以首屏需要做特殊处理。
* 入门门槛高，view编码需要释放资源，以免heap值过高，对编码人员的要求较高。

**行业趋势**

webapp结构简单：首先传递HTML文档框架；然后使用JavaScript修改页面；紧接着再从服务器传递更多数据然后再修改页面，如此循环。从性能的角度看，在现代浏览器中单页面Web App已经能够和普通native应用程序相媲美，而且几乎所有的操作系统都支持现代的浏览器。

一个单页面Web应用程序就是一个Web应用程序，但结构不同。其中最重要的是：在第一次请求的时候，所有的标记语言（HTML）就已经传输到客户端，其余的请求都通过REST API获取JSON数据，数据的传输通过Web Socket API或远程过程调用。单页面应用程序可以说是分拆Web技术的最后一步——通过分离(css)内容，改进架构（XML和 XSLT）上的灵活性，调用服务器（AJAX）再到解压应用程序的导航页面结构。因此，这在Web发展中是个历史性的转折点。

目前这只是单页面Web应用开发的初期，但可以看出将单页面应用、APIs以及JavaScript结合在一起将成为许多流行应用的规范。

## 1.7 总结

对比半年前的同行网站，从业务上看，移动端变化较快的是携程与去哪儿，同程门票/旅游一块变化也很大，其它网站变化较小。

由技术上看，携程依旧是webapp独树一帜，领跑行业，国内单页应用仍然较少，其中去哪儿ipad版本采用了单页应用的模式，说是ipad版本其实用做mobile更为合理，所以这可能是去哪儿切入webapp的一个信号或者尝试。

半年时间，主要变化依旧体现在携程与去哪儿上，不论从业务的扩展或者技术的发展，双方皆是你追我赶，这种短兵相接的现象仍然会持续下去，所以携程的变化不能停下，业务线拓展要快，网站响应速度也要快，下面我们来分析携程网站优缺点，以提升我们产品的速度以及体验。

# 2 机票频道对比

## 2.1 机票首页对比

### 请求资源个数

### 请求资源大小

### 性能时间

//todo

## 2.2机票搜索结果页对比

### 请求资源个数

### 请求资源大小

### 性能时间

#### API时间

数据来源：

<http://es.ops.ctripcorp.com/#/dashboard/elasticsearch/H5%20RestAPI%20%E6%8E%92%E5%90%8D>

搜索结果页的API会请求两个

<http://m.ctrip.com/restapi/Flight/Domestic/Flight/LowestPrice/Query>

<http://m.ctrip.com/restapi/Flight/Domestic/FlightList/Query>

FlightList的Query请求时间大于0.5秒的占50%，需要优化。同时这个请求并不是首屏展示所需要的，可以放置在后续用户点击时再请求。

#### 页面性能

分别模拟了2G/3G网络(30KB)和Wifi网络(150KB)来检测页面性能

## 2.3 机票预订页对比

### 请求资源个数

### 请求资源大小

### 性能时间

## 2.4 Heap值的变化

因为只有Ctrip使用的是单页的应用，所以Heap不会随着页面的切换而释放。

以用户操作一个简单的流程来看一下同一页面的Heap变化。

操作过程：首页->搜索结果页->预订填写页->搜索结果页->筛选一次->返回首页（\*2）

可以明显看出，用户访问的页面越多，Heap值越高，得不到释放。导致页面会变卡。

建议方案

1. 释放历史页面的Dom和Event
2. 释放闭包
3. 释放历史页面的计时器

# 3 网络传输优化

## 3.1 综述

前端优化分为两个切入点：网络传输与DOM操作，而网络传输是制约一个网站速度的主要因素。

网络传输的优化要点是，零请求，无流量，其意是最大程度的减少请求数，降低请求量。对webapp模式的应用来说，首屏加载慢是一个不可避免的问题，所以提升webapp首屏加载速度是提示携程整体网站速度的关键。

## 3.2 方案一：fake页

频道对比：机票首页VS用车首页

测试环境：30kb/s VS 150kb/s

3.2 首页请求数

3.2 首页请求量

3.2 机票首页响应速度-150kb/s

3.2 机票首页响应速度-30kb/s

3.2 机票首页响应速度-150kb/s

3.2 机票首页响应速度-30kb/s

根据以上数据，机票请求数为23，用车为18；机票请求量为227kb，用车为372kb。用车主要容量集中在了一个图片上面，达到了150kb，这些数据集中体现在了两个频道的响应。

* 机票，DOM Ready平均时间为360ms/4613ms
* 用车，DOM Ready平均时间为574ms/8049ms

这个数据说明请求量的大小会直接影响到页面的响应速度。

但是我们应该避免页面长时间白页，这个时候便提出了fake页的概念。页面渲染只需要完整的HTML以及CSS，这个便是第一个优化点。

从数据请求数以及请求量来说，机票首页的响应应该比较慢，若是任由js加载完成再渲染页面，用户很有可能失去耐心。

但是从DOMContentLoaded来看，机票首页事实上页面响应比较迅速，HTML与CSS的容量是34.7kb，所以这个加载结束后页面第一屏便渲染结束，然后再异步加载js，当js改变后再动态改变dom结构中的一些关键点，机票做了一个静态HTML页面，转载首屏的基本内容，让机票首页快速显示，然后js加载结束后会马上重新渲染整个页面，这个样子，用户就可以很快的看到页面响应，给用户一个快的错觉。

这个直观感受在机票以及用车首页首次加载时可以明显感觉到机票体验好得多，给人感觉快得多。

## 方案二：降低请求数

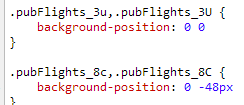
由机票首页来说，出现了重复引用，这个在线上出现是大忌，而且一个页面需要4个CSS来说，也是值得优化的。一个频道只会需要两个CSS，一个框架的，一个频道的，这些CSS需要合一，并且对内部代码进行精简，降低css方面的请求数。



这方面用车首页就基本满足要求



再看flights\_logo.css的代码片段：



这类代码在该文件中洋洋洒洒出现了500行之多，这个事实上是可消除的，从而消除该文件

## 3.4 方案三：降低请求量

* 以用车团队的car2.png为例，达到了124kb，事实上可压缩至35kb。
* 以框架库为例，除了核心包以外，不需要的UI或者功能库可以剔除，用到了再动态加载，减少首次加载量，让频道自己配置需要哪些组件。
* 以各频道为例，现在的方案是将前三页的js与html模板打包，其它页面访问时再请求，若是想提升首屏加载便可以只下载需要的页面文件。

## 3.5 方案四：缓存Ajax/localstorage

该方案的原理与前面类似，我们发送Ajax请求时候，应该缓存一些非实时数据，比如城市信息和常用联系人，但是我们只能缓存非敏感信息，这里以机票与团队游为例：

* 机票搜索页至列表页的请求数据会缓存30s-60s，若是过期时间内用户回到列表页的话不会重新请求数据，这对服务器压力，页面响应皆是有利的。
* 团队游从周边游进列表页会发一次Ajax请求，若是立刻由列表页返回首页，再点击周边游进入列表页又会发起请求，这个在30s内事实上意义不大，可以减少一次请求。

另外，携程Ajax数据请求全部是post请求，对于get和post的效率，我们曾经做过一次测试：

get100次平均耗时323ms；post100次平均耗时589ms，所以post方式是比get慢的，但post请求的优点是安全，并且参数没有长度限制。

携程选择post的原因，主要是处于安全考虑，一些字段出现在URL不安全，所以便牺牲了一定速度。

## 3.7 方案五：lazyload

只显示首屏页面，其它内容需要时再加载，比如列表页、图片lazyload

## 3.8 竞品对比

**机票首页**

Taobao和Qunar有以下几个特点

1. Qunar的框架代码放在外链中，一个文件；逻辑代码放在页面中，所以Domready时间会较长，但是减少了额外的JS请求
2. Qunar的首页和搜索结果页的JS和CSS是打包在一个文件中，请求前两个页面无需加载额外的JS，速度更快

我们需要改进的点

1. 选择城市默认会加载80kb的城市数据，包括国内和国际。建议默认加载国内的，当用户切换的时候再请求国际的数据，提升选择城市的数据。
2. 精简城市数据，只返回页面中需要的数据。
3. 精简框架代码

框架加载以下几个主要资源

-require.js

-include.libs.js

-include.ubt.js

-include.ga.js

-libs.js(libs\_r.js)

-bfa.min.js

-ga.js

-common.js

-widget.js(list)

-BU\_main.js

1. 已经确认WinPhone支持IE10以上的浏览器，所以可以删除include.libs.js的判断
2. Ubt和GA相关的统计代码统一在Lizard框架中加载，删除include.ubt.js和include.ga.js
3. Widget按照页面进行配置加载
4. 优化BU的main.js，每张页面只加载需要的逻辑代码JS

**机票列表**

Qunar和Taobao有以下几个特征

1. 机票搜索结果页默认只显示展示的最低价数据，点击某一航班之后再去请求更多的数据，首屏的速度会快；
2. Qunar的JS/CSS等资源已经在首页进行了缓存，所以此页面请求速度较快。
3. 在2G/3G网络情况下，Qunar由于是多页应用，资源需要重新加载，页面展示时间较长；Ctrip由于是单页应用，资源已经预先加载好，所以在网络较差的情况下，页面仍然能够较迅速的展示。

我们需要改进的点

1. 精简搜索列表页请求的数据，虽然请求回来的量不大，但是请求耗时较长。
2. 点击更多时再去服务器请求数据，一方面使搜索结果页首屏更快展示，另一方面可以节省用户的流量
3. \*\*建议将首页和搜索结果页两页的资源单独打包，提升用户的搜索速度。提供一个分块打包机制，根据用户的行为来进行页面集的资源打包。例如首页和搜索结果页，预订页和支付页。

# 4 DOM操作优化

## 4.1 综述

DOM操作主要分为页面渲染与资源清理（heap控制），两者之间又相辅相成，若是DOM操作一块处理不好，其产生的感觉就不再是慢，而是卡，所以DOM操作优化的主要目的就是消灭页面卡的问题，这个在移动端尤为重要。

## 4.2 方案一：优化页面渲染

### ****4.2.1关于页面渲染****

浏览器会解析三个东西：HTML、Javascript、CSS

浏览器首先会根据HTML生成DOM Tree，其次会根据CSS生成CSS Rule Tree，javascript可以通过DOM API与CSS API操作DOM Tree与CSS Rule Tree，从而引起页面变化。

浏览器解析结束会通过DOM Tree与CSS Rule Tree形成render tree，只有display不为none的元素才会形成render Tree，render Tree形成后浏览器会调用GUI绘制页面，在此之前做的一件事情便是layout或者说reflow。上面的描述简单而言可以分为以下流程：

* 生成DOM树
* 计算CSS样式
* 构建render tree
* reflow，定位元素位置大小
* 绘制页面

在这个过程中，若是javascript动态改变DOM Tree便会引起reflow

页面中的元素改变，只要不影响尺寸，比如只是颜色改变只会引起repaint不会引起回流

否则，reflow不可避免，这个时候便需要重新计算形成render Tree

reflow分为局部回流与全局回流，会影响下面的，不会影响上面的元素

reflow耗用的系统资源较大，DOM Tree中受到影响的节点皆会reflow，然后影响其子节点最坏的情况是所有节点reflow，该问题引发的现象便是低性能的电脑风扇不停的转，手机变得很热，并且非常耗电，以下操作可能引起reflow

* 操作dom结构
* 动画
* DOM样式修改
* 获取元素尺寸的API

### 4.2.2减少使用定位属性（fixed/absolute）

static元素处于文档流中，其渲染速度是最快的，我们做过一个测试：

100个absolute元素与100个static元素渲染时差在0.01-0.007ms

100000个元素渲染差距便增至30ms左右，这个微小的时差在移动端变得尤为明显，比如：

小米/三星手机（1000左右），便存在明显的渲染问题，具体表现为：

* 定位元素在手机上不能显示。
* 定位元素动画效果失效。

以上问题便是UI渲染失效多导致，最好的解决方案是减少使用定位元素，否则只能引起强烈reflow才能解决。

另外，产品经常会有fixed的相关需求，比如支付按钮一直出现在低端，这个需求会造成两个问题：

* fixed元素遭遇文本框时失效，可能会飘到页面中间阻挡输入
* 影响效率

问题一原因与移动端的实现有关，暂时没有完美的解决方案，问题二便与渲染直接关联

滚屏时，页面上所有的像素会跟着滚动，显卡对全屏幕上下移动的处理很快，但是若是出现一个fixed元素或者有元素不跟着一起滚动，那么滚动对手机浏览器来说就是一个负担，这种滚动的性能甚至体现在了iphone 4s，因为滚动可能会造成reflow，这个现象体现在：

使用absolute配合javascript模拟fixed效果时，会有断片的效果，该问题在iphone5s便不会出现这个问题。



### ****4.2.3奇技淫巧****

当然，我们不能忽略产品的需求，fixed类需求应该在技术上得到解决，还用户一个良好的体验。

**虚拟键盘导致fixed元素错位**

fixed元素一定会伴随虚拟键盘的出现，但是虚拟键盘只是“贴”在了viewport上，表面上不会对dom产生“任何”影响，但是这个时候fixed元素表现却变得怪异起来，会错位。

应用层面解决问题方案是，虚拟键盘弹出时将fixed元素设置为static，虚拟键盘消失时候设置回来。

由于虚拟键盘出现并未抛出事件，而检测scroll或者resize事件，皆会有一定延迟，会出现闪烁现象，所以现有最好的方案是setinterval定时器监控当前获取焦点元素是否为文本元素，若是是的话便需要处理，如此便可解决fixed元素错误问题。

**fixed元素滑动惯性平滑度**

我们常常遇到这种产品需求，tab标签栏开始固定，当滚动向下超过该标签栏后便会变成fixed元素，一直出现在头部，这样的需求在电脑上没有问题，但是在iPhone5s以下的手机常常会出现小范围错位或者快速移动大范围错位的问题。

这个时候我们可以引起reflow迫使浏览器重绘以解决这个问题，这里推荐一个奇怪的hack写法：同时设置三个image元素的src属性，便可以全范围解决该难题， 该方案被团队游团队证实并得到应用。



//三图片src，引发reflow，处理fixed方案惯性问题

var el = this.els.ctlc.find('img');

$(el[0]).attr("src", 'http://res.m.ctrip.com/html5/Content/images/144.png');

$(el[1]).attr("src", 'http://res.m.ctrip.com/html5/Content/images/144.png');

$(el[2]).attr("src", 'http://res.m.ctrip.com/html5/Content/images/144.png');

### 4.2.4其它

* CSS选择器尽量使用id与class，避免过度层叠
* 避免使用数值，比如：border: none不会引起渲染，而boder: 0会
* 动画时候让元素脱离文档流，以免导致大量reflow
* 避免逐条修改DOM样式，改以className实现同样功能
* 操作DOM时将display设置为none，因为这种元素不会影响渲染，或者操作fragment对象取代操作显示在页面上的DOM
* 避免将获取DOM样式属性的操作写在循环中，可能引起重复reflow

## 4.2 方案二：内存资源优化

### 4.2.1移动端的javascript

首先，移动端的性能与PC端的性能完全不在一个数量级上，比如，我一个朋友做过一个测试，使用innerHTML绘制大段，之后想获取HTML的ID节点，事实上是获取不到的，这种问题在单页模拟多页，动态创建DOM会经常发生

var element = $('<div id = "test">...大量结构...</div>');

$(root).html(element)

$('#test) //为空

这类问题匪夷所思，因为页面UI渲染与DOM操作是互斥的，但是就算出现了这个问题，一个解决方案是使用settimeout，更好的方案是使用DOMNodeRemoved事件监控页面DOM改变，将我们的DOM操作回调放入以确保渲染结束。

以上问题只是为了说明移动端的性能问题，这类性能问题会导致很多莫名其妙的问题，而且很多与渲染有关。但是这也从侧面说明了移动端资源的紧缺，若是heap值过大，会导致操作出现卡的现象，更有甚者，会引起页面假死直接退出。

webapp的模式，完全依赖于浏览器的垃圾回收，基本就是作死，因为传统页面一旦刷新页面整个资源完全释放，而webapp没有刷新这类操作，只有一个状态到两一个状态，不相关的内存会保留，资源必须手动释放，或者说，框架必须提供垃圾释放的机制。

这个由图表heap值变化可以清晰看出。

而view切换过程中，不用的资源若是不手动设置为null会导致变量得不到回收便脱离框架控制而失控了。所以我们在webapp的过程中需要注意：

* 释放没有使用的闭包
* 观察者需要得到清理
* 释放定时器
* view切换过程中，在destroy中释放view相关资源

### 4.2.2闭包陷阱

在我们工作过程中，滥用局部变量极有可能引起闭包陷阱，这个问题不止是性能问题，在逻辑上会引起错误，而且不易发现，比如，在AMD闭包中使用一个局部变量

var \_attributes = {};

callback ($.extend(\_attributes, opts));

如此操作，会改变\_ attributes对象，若是一个实例还无问题，但是两个实例的话便会发生变量污染。

这只是一个例子，但是在代码中滥用局部变量可能会引起不必要的隐忧，戒之慎之。

### 4.2.3webapp资源释放

根据前面的描述，我们可以得出一个结论：

无论是view还是UI组件我们得提供统一的destroy接口，以便让用户继承释放资源。

若是view的资源得不到释放导致heap值过高，webapp模式的网站其价值大减。这里有几点可以考虑：

* 一个频道view实例保存不超过5个，多了便释放dom结构以及内存引用
* view隐藏时释放内部资源，解除DOM事件句柄
* UI组件与view相同，需要统一释放机制

但是单页应用由于页面不会刷新，总有一些资源得不到释放，此问题任重道远，平时编写过程可以做以下优化：

* 使用函数替换逻辑

让我们的函数产生一个返回值替换函数中的大段逻辑，这样的第一个好处便是逻辑清晰，第二个好处是这些函数在不同的函数中，这个函数被使用后便会自动得到释放。

* 清理闭包引用

当一个闭包函数或者什么使用结束后，若不会再使用，便需要手动清理该变量，以便解除闭包之间的引用关系，从而释放资源。

* 使用对象属性或者方法

一个对象可以引用其他对象的属性或者方法，比如obj.foo = thatObj;这种情况下，我们可以随时删除对象解除引用关系，然后便可以清理资源。

### 4.2.4动画与假死

动画而言建议采用CSS3实现动画，CSS3中又推荐采用最新的接口，比如使用transform取代top/lelf操作，这样操作效率搞得多。

若是采用动画可以将对应元素设置为absolute以减少回流，另外最关键一点还是

避免移动DOM树过多的节点，这个时候需要驳回产品无理需求，比如：

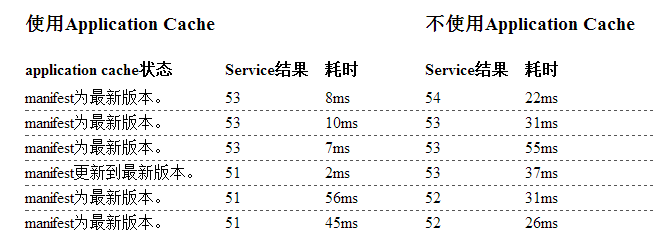
用车最初要求日期滚屏组件，显示半年的数据，这半年的数据便是180个DOM树

这个级别的DOM一旦移动整个手机会直接卡死，甚至构建DOM树，渲染页面也会出现假死现象，该问题需要规避。

### 4.2.3Application Cache

Application Cache是HTML5为webapp离线使用而增加的API，与localstorage、cookie等不同，Application Cache存储的是一系列请求资源允许浏览器在请求资源时不必通过网络，设计得当的话可以实现离线应用。

使用Application Cache主要是在网络性能上提升，有效降低了网络延迟，提升请求加速



但是也会有一些问题，比如新版本缓存不立刻生效；manifest中的请求路径相对于manifest文件，而非加载页面；更新/回滚等问题，所以使用与否还得论证。

# 5 体验优化

## 5.1 区域滚动

移动端经常需要实现区域滚动的需求，成熟的也有IScroll解决方案，但是方案却不理想。



就官方的例子便会出现以下问题：

* 头部消失
* 偶尔不能显示文本框焦点，或者焦点错位

导致的原因与组织浏览器默认事件有关，所以，我这里不太推荐各位大范围的使用区域滚动，而改在区域使用，就去哪儿的ipad版本在一个具有文本框的地方使用了IScroll，其提高的用户体验与导致的问题一样引人入胜。

# 6 Hybrid

## 6.1 多webview方案

多个webview性能没有问题，系统在切换的动画，让用户感觉不到加载的过程。

问题在于，现在的单页应用，所有JS，html都在同一个页面，载入内容过多，导致页面载入慢，而每开一个webview都需要载入这些js，html；

框架可否考虑非单页应用，减少每个页面内容，加快载入速度？

先前研究淘宝做的Hybrid的彩票，都是单页，每次跳转都是新开webview。

另外，从用户使用和操作来说，列表webapp，不太适合单页应用，至少现在我们app里面的那些列表/详情的，多刷几页列表页，点击详情，回退回来又到列表首页，基本没法用。