**移动端Web加载性能优化调研**

# 背景

目前微信，QQ空间等大量App都内嵌了H5，混合开发已经是目前的趋势，使用H5实现的功能能够在不升级App的情况下动态更新，而且可以在Android或iOS的App上同时运行，节约了成本，提高了开发效率。但是网络状况不好或者H5页面超级复杂的情况下，用户等待的时间过长就会失去耐心，所以H5页面的加载速度对用户体验影响很大。对于电商App来说这种影响更为致命，大部分市场、运营的活动都会通过H5直接展现给用户，过长的加载时间势必会影响App的推广。

# APP通用优化加载方案

1. H5缓存，需要讨论合理的缓存时间以及缓存清除的时机。
2. 常用资源预加载。

# H5缓存优化方案解析

1. **H5缓存机制介绍**
2. H5，即 HTML5，是新一代的 HTML 标准，加入很多新的特性。离线存储（也可称为缓存机制）是其中一个非常重要的特性。H5 引入的离线存储，意味着应用中的Web页面可进行缓存，并可在没有因特网连接时进行访问。
3. H5 应用程序缓存为应用带来三个优势：  
   离线浏览：用户可在离线时使用它们。  
   速度：已缓存资源加载的更快。  
   减少服务器负载：浏览器将只从服务器下载更新过或更改过的资源。
4. H5的6中缓存机制：浏览器缓存机制、Dom Storgage（Web Storage）存储机制、Web SQL Database 存储机制、Application Cache（AppCache）机制、Indexed Database （IndexedDB）、File System API。
5. **H5缓存机制原理分析**
6. **浏览器缓存机制**

浏览器缓存机制是通过 HTTP 协议头里的 Cache-Control和 Last-Modified等字段来控制文件缓存的机制。  
  
Cache-Control用于控制文件在本地缓存有效时长，比如服务器回包：Cache-Control:max-age=600 表示文件在本地应该缓存，且有效时长是600秒（从发出请求算起）。在接下来600秒内如果再次请求这个资源，浏览器不会发出 HTTP 请求，而是直接使用本地缓存的文件。

Last-Modified 是标识文件在服务器上的最新更新时间。下次请求时，如果文件缓存过期，浏览器通过 If-Modified-Since 字段带上这个时间，发送给服务器，由服务器比较时间戳来判断文件是否有修改。如果没有修改，服务器返回304告诉浏览器继续使用缓存；如果有修改，则返回200，同时返回最新的文件。

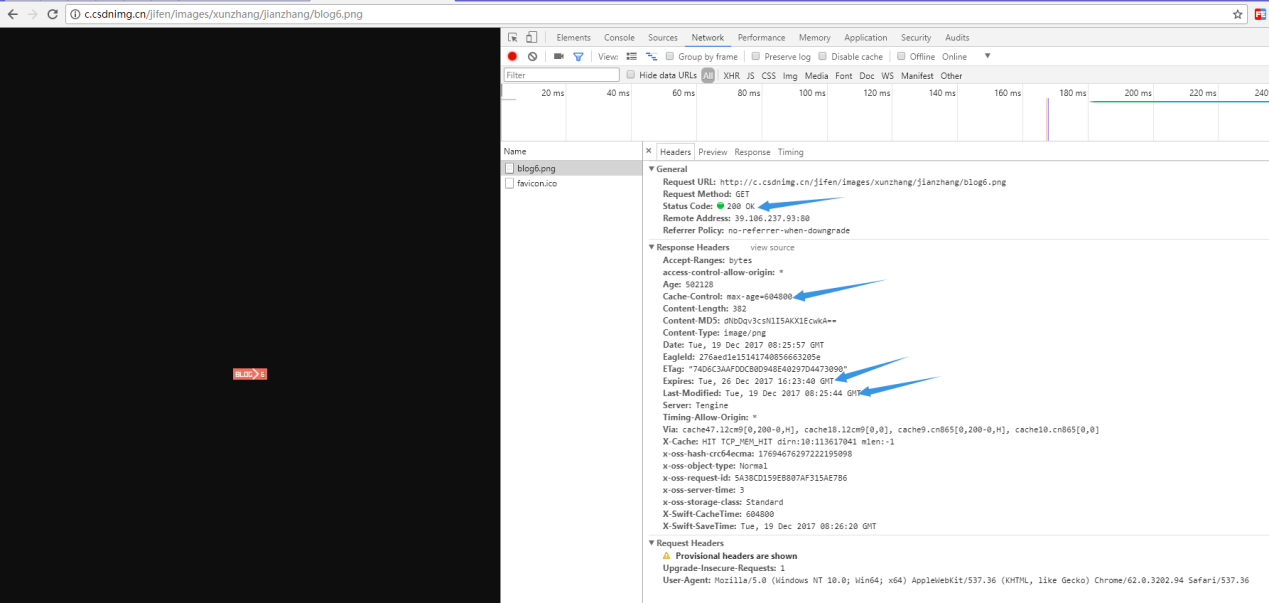
另外有两种特殊的情况：

用户手动刷新页面，在请求中加上字段：Cache-Control:max-age=0，发包向服务器查询是否有文件是否有更新，浏览器会直接认为缓存已经过期（可能缓存还没有过期）。

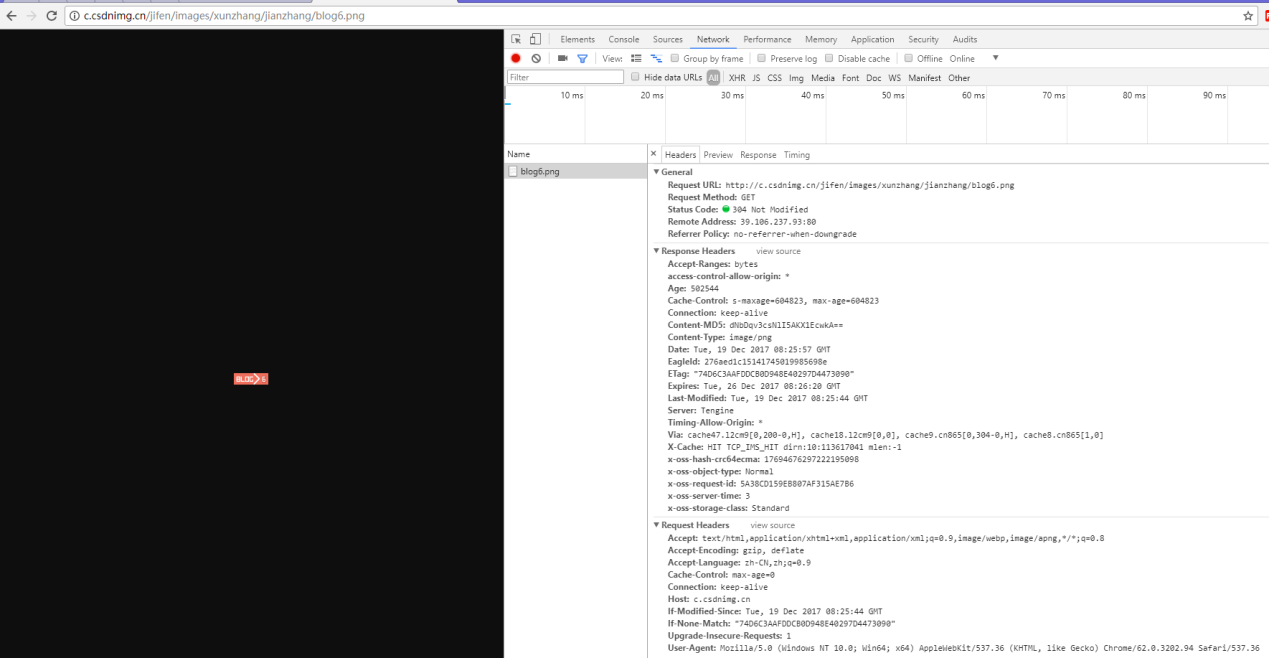
强制刷新页面，在请求中加上字段：Cache-Control:no-cache（或 Pragma:no-cache），发包向服务重新拉取文件，浏览器会直接忽略本地的缓存。

下面是通过 Google Chrome 浏览器自带的开发者工具，对一个资源文件不同情况请求与回包的截图。

首次请求：200，并且返回了过期的时间和上次修改的时间。



请求头中带上If-Modified-Since字段的请求：

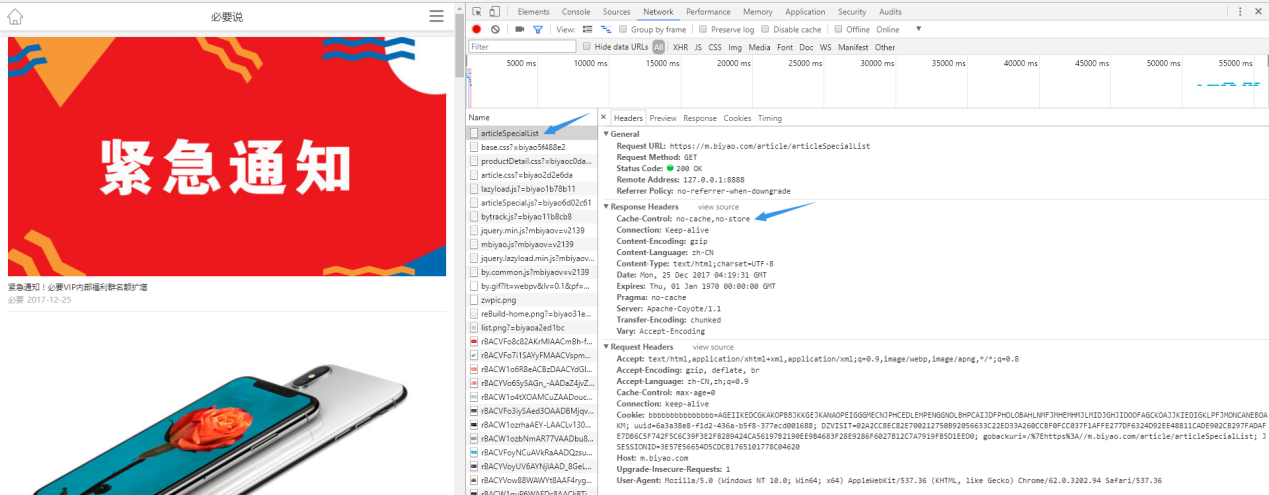


如果请求头中没有If-Modified-Since字段，在缓存有效期内不会发起网络请求，直接使用本地缓存。  
  
在实际应用中，为了解决 Cache-Control 缓存时长不好设置的问题，以及为了”消灭304“，Web前端采用的方式是：

在要缓存的资源文件名中加上版本号或文件 MD5值字串，如 common.d5d02a02.js，common.v1.js，同时设置 Cache-Control:max-age=31536000，也就是一年。在一年时间内，资源文件如果本地有缓存，就会使用缓存；也就不会有304的回包。

如果资源文件有修改，则更新文件内容，同时修改资源文件名，如 common.v2.js，html页面也会引用新的资源文件名。

线上必要说列表目前没有开启缓存策略，每刷新一次，所有的图片，css等全部重新请求，抓包图片如下：

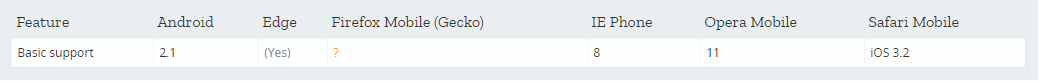


1. **Dom Storage 存储机制**Dom Storage 是通过存储字符串的 Key/Value 对来提供的，并提供 5MB的存储空间，而Cookies的存储空间才4KB。DOM Storage 分为 sessionStorage 和 localStorage，sessionStorage 用来存储与页面相关的数据，它在页面关闭后无法使用。而 localStorage 则持久存在，在页面关闭后也可以使用。  
     
   Dom Storage 给 Web 提供了一种更灵活的数据存储方式，存储空间更大（相对 Cookies)，方便存储服务器或本地的一些简单临时数据，如果要存储结构化的数据，则需要将要存储的对象转为 JSON 字串。不太适合存储比较复杂或存储空间要求比较大的数据，也不适合存储静态的文件等。

因在移动端浏览器存在例如Safari无痕模式不支持Web Storage的情况，在要求大范围 兼容的情况下无法使用，但是Native Webview不存在不兼容的问题，可设计使用存储 一些实时性不是很高的数据，避免过多的网络请求造成的性能问题。例如天猫搜索采用 Session Storage存储每次搜索数据和分页数据。

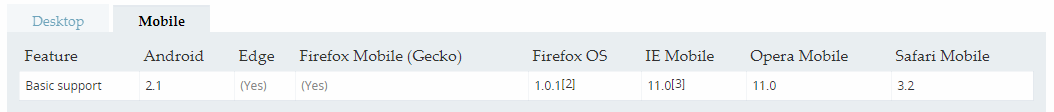
在桌面访问情况下天猫搜索接口返回数据时间为：

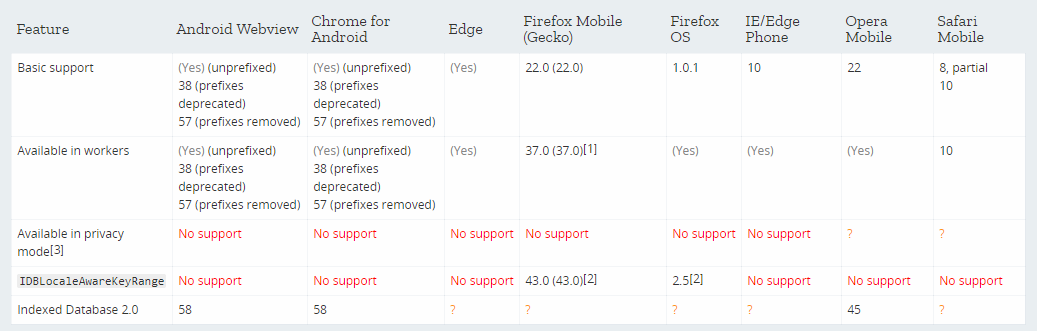


利用Session Storage缓存数据后数据由本地抓取，避免网络请求消耗。  
  
MDN提供兼容范围如图：  


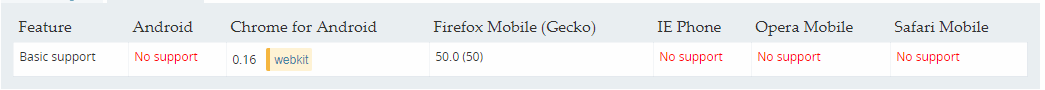
1. **Web SQL Database存储机制**基于 SQL 的数据库存储机制用于存储适合数据库的结构化数据。根据官方的标准文档，Web SQL Database 存储机制不再推荐使用，将来也不再维护，而是推荐使用 AppCache 和 IndexedDB。  
     
   SQL Database 的主要优势在于能够存储结构复杂的数据，能充分利用数据库的优势，可方便对数据进行增加、删除、修改、查询，但是SQL Database 不太适合做静态文件的缓存。
2. **Application Cache 机制**Application Cache是为支持 Web App 离线使用而开发的缓存机制，是以文件为单位进行缓存，且文件有一定更新机制。  
     
   AppCache 的原理有两个关键点：manifest 属性和 manifest 文件。HTML 在头中通过 manifest 属性引用 manifest 文件。manifest 文件是一个以 appcache 结尾的文件，里面列出了需要缓存的文件以及需要网络请求的文件。  
     
   一旦应用被缓存，它就会保持缓存直到发生下列情况：  
   用户清空浏览器缓存。  
   manifest 文件被修改。  
   由程序来更新应用缓存。

MDN提供兼容范围如图：



1. **Indexed Database存储机制**  
     
   IndexedDB 是一种灵活且功能强大的数据存储机制，它集合了 Dom Storage 和 Web SQL Database 的优点，用于存储大块或复杂结构的数据，提供更大的存储空间，使用起来也比较简单，并且操作数据库的API都是异步的，操作的结果都是在回调中返回，可以作为 Web SQL Database 的替代，但是不太适合静态文件的缓存。  
     
   优点如下：  
   以key-value 的方式存取对象，可以是任何类型值或对象，包括二进制。  
   可以对对象任何属性生成索引，方便查询。  
   较大的存储空间，默认推荐250MB(分 HOST)，比 Dom Storage 的5MB 要大的多。  
   通过数据库的事务（tranction）机制进行数据操作，保证数据一致性。  
   异步的 API 调用，避免造成等待而影响体验。  
   MDN提供兼容范围如图：  
   

支持基础API，具体兼容情况还需实际确认。

1. File System API  
     
   Web应用对比桌面应用来说一直存在一个被大家诟病的问题：它们无法在一个有层次的文件夹结构体即文件系统中互动和组织。但是HTML5 File System API为Web开发者提供了一个可以创建本地文件夹结构的接口，window.requestFileSystem(type, size, successCallback, opt\_errorCallback)。开发者可以根据入参指定文件系统大小和生命周期，以及文件系统创建成功或者失败的回调。同时HTML5为开发者提供了一系列的操作此文件夹系统的API，包括增删改查和修改文件目录等，但是此API存在严重兼容问题。  
     
   目前MDN给到的兼容范围见下图：  
   
2. **缓存机制适用场景**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 缓存机制 | 优势 | 适用场景 | android是否支持 | ios是否支持 |
| 浏览器缓存机制 | HTTP协议层支持 | 静态文件的缓存 |  |  |
| Dom Storage | 较大存储空间、使用简单 | 临时简单数据的缓存、Cookies的扩展 | 支持 |  |
| SQL Database | 存储、管理复杂结构数据 | 用IndexedDB替代、不推荐使用 | 支持 |  |
| AppCache | 方便构建离线App | 离线App、静态文件缓存 | 支持，但是官方已经不太推荐 |  |
| IndexedDB | 存储任何类型数据、使用简单、支持索引 | 结构关系复杂的数据存储SQL Database的替代 | Android4.4开始支持 |  |
| File System API | 支持文件系统的操作 | 数据适合以文件进行管理的场景 | 不支持 |  |

1. **单独开启浏览器缓存机制加载时间对比**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 首次 | 185ms | 196ms | 216ms | 385ms | 289ms | 200ms | 292ms | 337ms | 183ms | 181ms |
| 非首次 | 87ms | 68ms | 88ms | 88ms | 92ms | 105ms | 83ms | 82ms | 92ms | 103ms |

测试页面为：http://192.168.97.169:8888/index.html。

其中的manifest.css、jquery.js文件比较大、加载时间较长。

10次首次启动中去除最高值385ms，去除最低值181ms后的平均加载时间为：237.25ms

10次非首次启动中去除最高值105ms，去除最低值68ms后的平均加载时间为：89.375ms

1. **单独开启Application Cache加载时间对比**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 首次 | 256ms | 243ms | 218ms | 239ms | 232ms | 224ms | 222ms | 229ms | 215ms | 246ms |
| 非首次 | 134ms | 72ms | 94ms | 98ms | 130ms | 96ms | 168ms | 93ms | 92ms | 105ms |

测试页面为：<http://192.168.97.169:8888/manifest.htm>。

其中的manifest.css、jquery.js文件比较大、加载时间较长。

10次首次启动中去除最高值256ms，去除最低值215ms后的平均加载时间为：231.625ms

10次非首次启动中去除最高值168ms，去除最低值72ms后的平均加载时间为：105.25ms

1. **缓存的时间**

其中的manifest.css、jquery.js文件比较大、加载时间较长。

1. **缓存清除的时机**

其中的manifest.css、jquery.js文件比较大、加载时间较长。

# APP优化H5加载速度方案

1. WebView先不要自动加载图片，等页面finish后再发起图片加载。

具体方法：  
在WebView初始化时设置如下代码：

public void int () {

if(Build.VERSION.SDK\_INT >= 19) {

webView.getSettings().setLoadsImagesAutomatically(true);

} else {

webView.getSettings().setLoadsImagesAutomatically(false);

}

}

同时在WebView的WebViewClient实例中的onPageFinished()方法添加如下代码：

@Override

public void onPageFinished(WebView view, String url) {

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

webView.getSettings().setLoadsImagesAutomatically(true);

}

}

1. 3.0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，但是WebView开启硬件加速后，可能导致WebView白屏和闪烁，所以废弃此方案。
2. 缓存WebView数据，需要讨论合理的缓存时间以及缓存清除的时机。  
   具体方法：  
   WebSettings s = webvi.getSettings();  
   s.setDomStorageEnabled(true);  
   s.setDatabaseEnabled(true);  
   s.setDatabasePath(BYApplication.getContext().getDir("database",

Context.MODE\_PRIVATE).getPath());  
s.setAppCacheEnabled(true);  
缓存策略：webSettings.setCacheMode(WebSettings.LOAD\_CACHE\_ELSE\_NETWORK);  
清除时机：退出必要APP时清除WebView的缓存数据。

1. 常用JS本地化。  
   具体方法：  
   APP端将常用的 JS 脚本本地化，直接打包放入 apk 中。比如 H5 页面获取用户信息，设置标题等通用方法，就可以直接写入一个 JS 文件，放入 asserts 文件夹，在 WebView 调用了onPageFinished() 方法后进行加载。
2. 常用资源预加载。  
   缓存技术，能优化二次启动 WebView 的加载速度。而首次加载过程会有许多外部依赖的 JS、CSS、图片等资源需要下载，而这些资源的下载严重影响H5的加载速度。  
   具体方法：  
   在应用启动时，用户在 wifi 的网络环境下， 提前下载 H5 页面需要的资源，在WebView的加载过程中通过shouldInterceptRequest方法拦截资源的加载过程，如果本地存在资源直接返回本地的资源，否则从网络上下载。
3. 使用第三方WebView内核：腾讯X5。  
   解决了Android众多厂商WebView定制化的兼容性问题，SDK只有212KB，可以共享微信和手机 QQ 的 X5 内核，可以降低内核的冷启动几率，加快WebView加载速度。  
   网页浏览能力

Web页面crash率降低75%

页面打开速度提升35%

流量节省60%

阅读模式

去除网页中广告等杂质

优化文章的阅读体验

文件打开能力

包括会话页的互传文件及邮件中附件

支持doc、ppt、xls、pdf等办公格式

支持jpg、gif、png、bmp等图片格式

支持zip、rar等压缩文件

支持mp3、mp4、RMVB等音视频格式

视频菜单能力

支持屏幕调节等常规视频菜单功能

灵活切换全屏&小窗功能

1. 提高渲染的优先级。  
   webView.getSettings().setRenderPriority(RenderPriority.HIGH)。

# 交互标准

1. Android中调用WebView中的js脚本非常方便，只需要调用WebView的loadUrl()方法即可。  
   具体方法：  
   contentWebView.loadUrl("javascript:alert('hello')");
2. WebView中js调用本地java方法。  
   具体方法：  
   Android端实现：  
   webView.addJavascriptInterface(new WebAppInterface(this),"Android");  
   public class WebAppInterface {

Context mContext;

WebAppInterface(Context c) {

mContext = c;

}

@JavascriptInterface

public void showToast(String toast) {

Toast.makeText(mContext, toast, Toast.LENGTH\_SHORT).show();

}  
H5端实现：  
<input type="button" value="Say hello"

onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">

function showAndroidToast(toast) {

Android.showToast(toast); }

</script>

# 线上js交互数据结构

1. js交互数据结构  
   现app中都是js通过js桥调用native代码，并且传递数据给app，js桥统一调用本地LinkAndroidApp(String jsStr)方法，其中传入的jsStr为json格式的字符串。

jsStr的通用json格式：

{

"type" : 19, //操作的类型，app判断不同类型执行不同操作

"values" : {xxxx} //操作的具体数据，不同类型数据不同

}

1. 各类型数据  
   type：2 //吊起native登录

values:

type：11 //吊起native设置页面

values

type：14、15 //关闭当前web页面

values

type：17 //图片放大展示

values：

{

"images" : [

{

"imgSrc" : "http://biyao.com/image.jpg",

"selected" : true

}

]

}

type：19 //签名web页面填写的签名信息回传给app

values：

{

"extendInfo" : {

"productSigns" : [

{

"carve\_content" : "123"

}

]

},

"glasstype" : "2",

"performExtends" : {

"halitus" : [

{

"sm" : 1300220055

}

],

"sign" : []

}

}

------------------------------

{

"extendInfo" : {

"productSigns" : [

{

"carve\_content" : "123"

}

]

},

"glasstype" : "1"

}

--------------------------------

{

"extendInfo" : {

"productSigns" : [

{

"carve\_content" : "15885555"

}

]

},

"performExtends" : {

"halitus" : [],

"sign" : [

{

"pr" : 0,

"sm" : 1300600002

}

]

}

}

type：20 //中间页面跳转scheme的形式跳转本地页面

values：

{

"isModel" : 0,

"title" : "标题",

"type" : 0,

"value" : "xxxx"

}

type：21 //标题条标题通过js获取web页面标题设置到原生标题条上

values：

{

"title" :“web页面标题”

}

# 新版js交互数据结构

1. js交互数据结构，废弃type，功能操作都改为路由字符串。  
   {

"routerUrl":""

}