**web前端优化：**

dns缓存，减少时间； 网络请求**阶段**： cdn 解决网络选择和缓存的问题， cdn请求的静态资源文件发挥作用，而请求的cookie是没有用的， 因此是希望将cookie从请求头中去掉。cdn的域名不要和主站的域名一样。防止访问cdn的时候还携带主站cookie的问题。

请求过程中一些潜在的性能优化点：

dns是否可以通过缓存减少dns查询时间

网络请求的过程走最近的网络环境

相同的静态资源是否可以缓存

能否减少请求http请求大小

减少http请求

服务端渲染

**H5页面性能优化：**

<https://www.jianshu.com/p/3f45832c57a0>

<https://www.cnblogs.com/famensaodiseng/p/11820127.html>

<https://github.com/ljchen1129/SecondsOpenH5> // 主要为后端部分

<https://www.cnblogs.com/guangyun/p/8401285.html>

Pwa

Vue-ssr

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*资源加载\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

首屏加载： 是首屏加载，即在可见的屏幕范围内，内容展现完全，loading进度条消失

按需加载： 首先要明确，按需加载虽然能提升首屏加载的速度，但是可能带来更多的界面重绘，影响渲染性能，因此要评估具体的业务场景再做决定。 比如页面中使用到的图片，如果对应的视图部分没有显示出来，先不加载，等对应视图显示出来后再加载。

Eg： 下面展示一个按需加载图片的逻辑

// 页面加载完成时先调用一下，首屏内的图片。

loadImg()

document.addEventListener('scroll', function (e) {

// 跟着滚动，继续加载剩下的图片

loadImg()

})

function loadImg () {

// slice可以将类数组转化成数组对象

[].slice.call(document.querySelectorAll('[data-src]')).forEach(function (item) {

// 判断元素是否已经进入视野

if (item.getBoundingClientRect().y <= innerHeight) {

// 设置src

item.src = item.getAttribute('data-src')

// 删除data-src属性，可以减少querySelectorAll的查询次数

item.removeAttribute('data-src')

}

})

}

<img data-src="xxx"></img>

Lazyload： Lazyload，即延迟加载，这并不是一个新的技术，在PC时代也是非常常用的一种性能优化手段。这个方案的原则是让屏幕外，或者不影响整体效果显示的图片、背景等资源，在界面就绪之后再进行网络加载。

滚屏加载： 滚屏加载是一种常见的无刷新动态加载数据的方案，通常用在列表形式数据展示中。

Media Query（响应式加载）： 通过这项技术，我们能够方便地控制资源的加载与显示

第三方资源异步加载： 防止第三方资源的使用影响到页面本身的功能。

Loading进度条： 给用户视觉感受， 能够知道加载的进度

避免30\*/40\*/50\*的http status

Favicon.ico： 如果我们没有设置图标ico，则会加载默认的图标：域名目录下的favicon.ico。很多开发者没有注意到这一点，就会导致这个请求404或者500。保证图片存在， 尽可能小一些， 并设置较长的缓存过期时间。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*图片的使用\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

图片的使用格式选择： 格式选择。显示效果较好的图片格式中，有webp、jpg和png24/32这几种常见的图片格式。一般来说，webp的图片最小，但在iOS或者android4.0以下的系统中可能会有兼容性问题需要解决；Jpg是我们最常使用的方案，大小适中，解码速度快，兼容性问题也基本不存在，是我们在H5的应用中使用起来性价比最高的方案；Png24或png32，一般来说，显示效果肯定会比jpg更好，但是实际上人眼很难感知出来，所以在H5应用中要避免这种格式的大图片。对于少量的图片，推荐用[智图](http://zhitu.isux.us/)或者[tinypng](http://www.tinypng.com/)等工具来帮助自己选择合适的大小、格式。

像素控制： 在H5应用中，图片的像素要严格控制，一般来说不建议宽度超过640px

避免DataURL： h5下尽量避免使用dataUrl， 因为它的数据体积通常比二进制图片的格式大1/3，而且它不会被浏览器缓存，每次页面刷新都需要重新加载这部分数据。

使用图片的替代(css3, svg, iconfont)

### /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*域名/服务端部署\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Gzip： 服务端要开启Gzip压缩。

资源缓存，长cache

合理设置资源的过期时间，尤其对一些静态的不需要改变的资源，将其缓存过期时间设置得长一些。

分域名部署(静态资源域名)

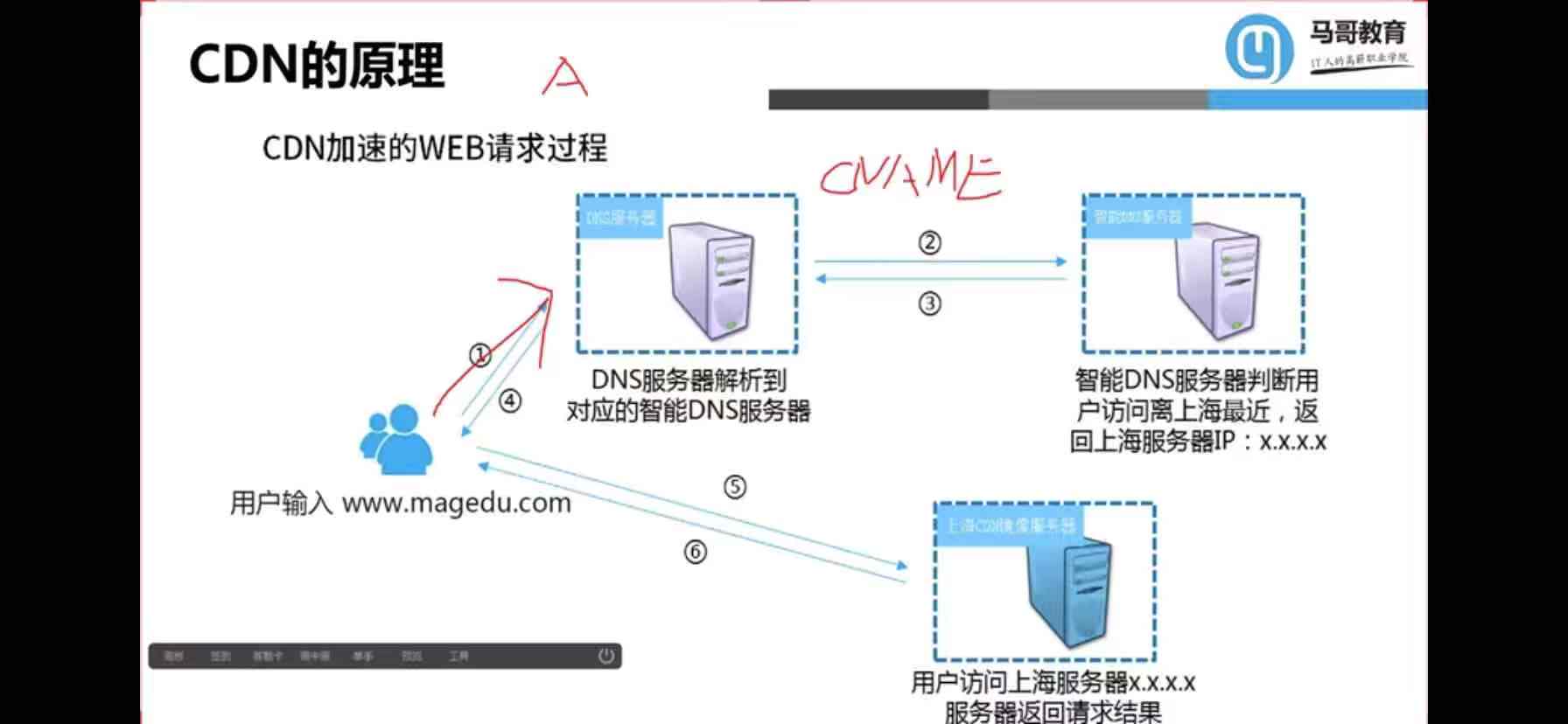
将动态资源和静态资源放置在不同的域名下，例如图片，放在自己特定的域名下。这样的好处是，静态资源请求时，不会带上动态域名中所设置的cookie头信息，从而减少http请求的大小。

减少Cookie： 尽量减少Cookie头信息的大小，因为这部分数据使用的是上行流量，上行带宽更小，所以传输速度更慢，因此要尽量精简其大小

cdn加速：部署CDN服务器，或者使用第三方的CDN加速服务，优化不同地域接入网站的带宽速度。CDN的关键技术主要有内容存储和分发技术。CDN的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求。

比如图片、视频、html等静态资源等都可以通过cdn缓存

根据用户来源地址的不同，返回不同的结果



*Tips:*

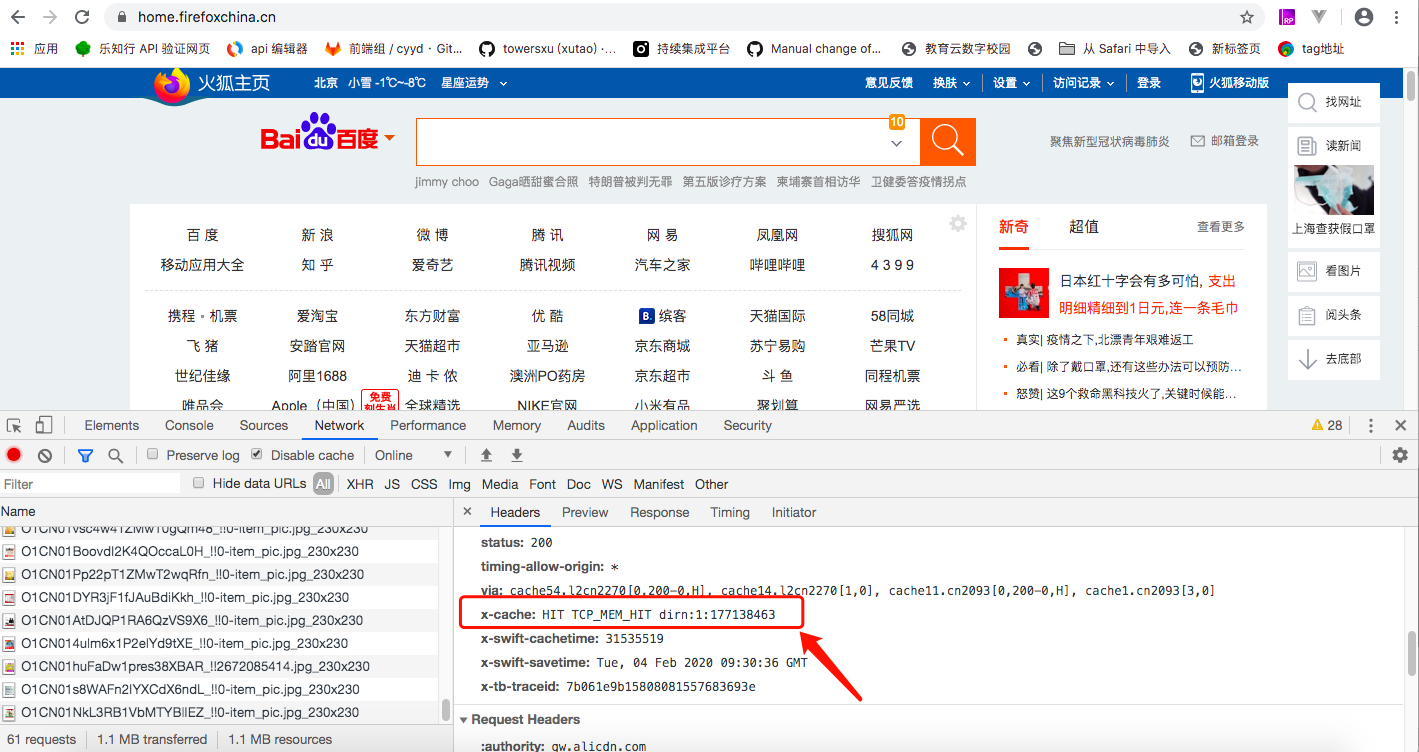
判断网站是否使用了cdn，（参考网址： <https://blog.csdn.net/Fly_hps/article/details/98486807>）1、如果多地ping同一网站，出现多个解析IP地址，那么说明使用了CDN进行内容分发； 2、 使用nslookup查看域名解析对应的IP地址，如果解析到多个IP地址则说明使用了CDN， eg： nslookup [www.baidu.com;](http://www.baidu.com;) 3、 使用CDN查询工具进行查看（该网址未成功打开）：<http://www.cdnplanet.com/tools/cdnfinder/>

判断该url下是否命中cdn： 主要查看响应头信息中的“X-Cache”字段。

显示“MISS”，说明没有命中CDN缓存，是回源的。

显示“HIT”，是命中了CDN缓存。

（via是http协议里面的一个header,通过代理和网关记录http请求x-cache是squid代理的自定义header,用来记录缓存的命中与否）



### /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*代码资源：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Javascript, CSS合并：尽量将所有的js和css合并，减少资源请求的次数。

外联使用js, css：外联使用js和css，这样可以有效地利用缓存，避免html页面刷新后重新加载这部分代码。

压缩html, js, css：压缩代码，尤其是js和css资源，压缩后的大小可以降低至原来的1/3以下，有效节约流量。

资源的版本更新：库js、css通常不会更新，但是我们的业务js和css可能会有更新，如果命中浏览器缓存，可能会让一些新的特性不能及时展现，甚至可能导致逻辑上的冲突。

因此对于这些js、css的资源引入，最好用版本号或者更新时间来作为后缀，这样的话，后缀不变，命中缓存；后缀改变，浏览器自动更新最新的代码。

Css位置

CSS要放到html代码的开头的head标签结束前。如果网页是动态生成的，那么在head代码完成后可以强制输出（例如php的flush()操作），这样的话，浏览器就会更快地解析出来head中的内容，开始下载css文件资源。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*代码规范\*\*\*\*\*\*\*\*\*\*\*\*\*/

避免空src： 导致无效请求

避免css表达式：可能会让页面多次执行计算，造成卡顿等性能问题。Eg: left: expression(document.body.offsetWidth - 180 "px");

#### （在其他浏览器中不起作用，单独针对ie设置会比较有用。 但是计算频率会很高，不建议使用）

#### 避免空css规则： 降低css渲染计算的成本

#### 避免直接设置元素style：直接设置style属性，一方面在html代码中不利于缓存，另一方面也不利于样式的复用，因此要避免，通过指定id或者class的方式，在css代码块中进行样式调整。

### /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*服务端接口\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### 接口合并：如果页面需要请求两部分以上的数据接口，建议将其合并，否则会增加一次http请求。

#### 减少接口数据量：有的时候，服务端会把一些无关紧要的数据返回回来，尤其是类似于更新时间、状态等信息，如果在客户端不影响内容的逻辑展示，不妨在接口返回的数据中直接去掉这些内容。

#### 缓存：缓存接口数据，在一些数据新旧敏感性不高的场景下很有作用，在非首次加载数据时候优先使用上次请求来的缓存数据，可以让页面更加快速地渲染出来，而不用等待一个新的http请求结束之后再渲染。这一点我们在后面还会再次提及

### **其他一些建议**

* 合理使用css
  + 正确使用Display属性 Display属性会影响页面的渲染，因此请合理使用
  + display:inline后不应该再使用width、height、margin、padding以及float
  + display:inline-block后不应该再使用float
  + display:block后不应该再使用vertical-align
  + display:table-\*后不应该再使用margin或者float
  + 不滥用float
  + 不声明过多的font-size
  + 值为0时不需要单位
  + 标准化各种浏览器前缀
  + 无前缀应放在最后
  + CSS动画只用 （-webkit- 无前缀）两种即可
  + 其它前缀为 -webkit- -moz- -ms- 无前缀 四种，（-o-Opera浏览器改用blink内核，所以淘汰）
* 选择器
  + 避免让选择符看起来像是正则表达式。高级选择器不容易读懂，执行耗时也长
  + 尽量使用ID选择器
  + 尽量使用css3动画
* 资源加载
  + 使用srcset
  + 首次加载不超过1024KB（或者可以说是越小越好）
* html和js
  + 减少重绘和回流
  + 缓存dom选择和计算
  + 缓存列表.length
  + 尽量使用事件代理,避免批量绑定事件
  + 使用touchstart，touchend代替click
  + Html使用viewport
  + 减少dom节点
  + 合理使用requestAnimationFrame动画代替setTimeOut
  + 适当使用Canvas动画
  + TouchMove, Scroll事件会导致多次渲染

使用单页架构，基本不存在页面跳转的等待时间，只需要执行js逻辑触发界面变化，最多进行一次网络请求，获得服务端数据，其他资源均不需要再次请求。

资源离线： 使用“离线包”策略。一些固定的图片、js库等，被打包放入app中（或根据需要，在app启动的时候进行下载更新）。

微应用中，网页代码里面加载网络资源的需求，就变成了直接加载本地文件，速度自然得到再一次巨大的提升。

本地数据持久化和更新机制(版本管理)： 对于一些时效性没有那么高的数据，可以考虑将接口数据缓存。而不是每次都要到服务端请求数据。弊端： 首次加载不可避免；服务端更新数据，客户端不能快速感知；数据更新后，需要重新渲染界面，界面刷新的性能消耗比正常情况更多，而且增加了程序的复杂度，容易出错。

预加载： 通过统计用户的行为信息，预判用户下一步操作，并在操作之前提前加载项目的一部分内容。

**Webapp 和nativeapp 的区别比较**

**参考网址：** <https://www.cnblogs.com/famensaodiseng/p/10763592.html>

**Pwa（Progressive Web App，渐进式增强WEB应用)**

pwa不是一项单独的技术，是应用一系列技术进行使用优化后的webapp，你更可以把它理解成是一种思想和概念，目的就是对标原生app，将Web网站通过一系列的Web技术去优化它，提升其安全性，性能，流畅性，用户体验等各方面指标，最后达到用户就像在用app一样的感觉。（目的就是在移动端利用提供的标准化框架，在网页应用中实现和原生应用相近的用户体验的渐进式网页应用）；

具体主要用到了：

Web App Manifest， 允许浏览器将站点添加到用户的手机屏幕上，添加为快捷方式，是用户有沉浸式的体验，其实是一个浏览器的内核， 但是没有地址栏，菜单栏等等ui，看上去和nativeapp基本一致

Service Worker： 浏览器在后台独立与网页运行的脚本； 拦截和处理网络请求，操作缓存； 支持 push api等； 后台同步和更新缓存

Cache API 缓存

Push&Notification 推送与通知

Background Sync 后台同步

响应式设计

主要特性：

快速，提升加载速度，减少白屏时间，实现快速加载；

可靠，允许webapp在离线时也能访问，而不是返回404 报错页面

粘性： 允许用户直接将站点添加到首屏，从而解决webapp 打开入口过深的问题

pwa的核心是提升用户体验，用户使用是并不知道是使用的是webapp还是native app。

pwa的基础是https ， 如果使用pwa的话，需要将自己的网站设置成https协议才能继续。

pwa兼容性查看网址： <https://lavas.baidu.com/ready>

Manifest.json

使用该文件需要将其通过link标签引入到项目的html文件中：

<link rel="manifest" href="static/manifest.json">

Safari目前支持通过meta/link声明的一些私有属性

文件中参数介绍 ： <https://blog.csdn.net/qq_32682137/article/details/83858800>

Eg：

{

"name": "百度天气", // 表示应用名称，启动画面中的文字

"short\_name": "standalone", // 手机主屏幕添加快捷方式后，快捷方式显示的就是这个名称

"start\_url": "",// 制定应用打开时的网址 可以添加参数用于来源统计， 如果为空则默认使用用户打开的当前页面为首屏

"icon": "../src/assets/login.png", // 快捷方式图标和启动画面图标

"background\_color": "#3E4EB8", // 启动画面的背景颜色

"theme\_color": "#2f3ba2", // 启动画面中状态栏和地址栏的颜色

"display": "standalone" // 启动画面的类型 fullscreen 应用将占满整个屏幕 standalone： 浏览器相关ui（如导航栏、工具栏等）将会被隐藏 minimal-ui： 显示形式与standal类似，不同浏览器在效果略有不同 browser： 与普通网页在浏览器中打开的显示是一致的， 对于pwa 推荐使用前两种

}

Service worker

加快资源的加载速度，尽量减少白屏时间，核心是提供更好的用户体验

**性能优化手段：**

Cdn

Css sprite

Compress/ Gzip

Async / Defer

http Cache

Service worker 不但能使页面达到秒开的功能， 而且在无网的情况下作出很好的响应。

Service worker 可以拦截网络端向服务端发送的网络请求， 然后根据条件判断是请求本地的缓存还是云端的服务。再将请求到的内容放到本地中的缓存使用。

它是一种特殊的web worker ，是浏览器运行在后台，与网页主线程独立的另一个线程，这种网络主线程的出现通常都是为了一些比较耗费性能的计算，有需要的时候再跟主线程通信，告知主线程它的计算结果，从而将计算和渲染独立开来，从而避免了阻塞的情况。

Service worker 离线缓存的新标准

Service worker 特性：

不能直接访问/操作Dom 只能使用一些特定的api， 这些api在service worker的上下文中是全局的： Promise、 Fetch API、Cache API

需要时直接唤醒，不需要时自动休眠

离线缓存内容开发者可控

一旦被安装则永远存活， 除非手动卸载

必须在https环境下工作（本地环境除外），安全考虑，避免网络传输中信息被恶意篡改

广泛使用了Promise

同域下允许注册多个不同scope的service worker， 这些service worker分管不同的域， 一般不会冲突， 并生层独立的service worker 上下文。

Service worker： 注册-》 安装-》 激活

**Service worker 注册：**

/\*\*\* \*

如果不指定scope 的话，

默认情况下service worker控制的作用域为sw-demo.js

即该文件对应的父级作用域所在的路径下，

如示例中，如果不加scope， 默认作用域为以static为

开头的页面所发出的请求

\*/

/\*\*\*

指定scope时不能越域， 否则将注册失败，

但是服务端给service worker的文件

设置头部信息 service-worker-allowed，

并给它设置最大的域， 如果注册的时候跨域了，

但是在service-worker-allowed允许的最大范围内， 注册将会成功

\*/

if('serviceWorker' in navigator) {

window.addEventListener('load', function() { // 页面资源加载结束后，调用，避免与其他资源征用带宽，从而导致页面性能下降

navigator.serviceWorker.register('/static/sw-demo.js', {scope: '/static/'}) // 具体路径根据实际情况，有可能放到ngix中，则写对应的路径

.then(function() { // 注册成功后，返回promise 对象 login was successful ， Return promise object

})

.catch(function(err) { // 注册失败 login has failed

})

})

}

**安装：**

this.addEventListener('install', function(event) {

event.waitUntil( // 控制以下代码的流程，只有以下代码resolve了，安装过程才能结束

// 安装过程中开辟一个跟service worker所对应的缓存区域， 并命名为my-cache-v1

caches.open('my-cache-v1').then(function(cache) { // 获取到缓存区域， 并命名为cache

// 调用catch.addAll 方法来缓存我们指定的文件列表

// addAll 是一个原始操作，如果所有指定的资源都缓存成功，则安装成功，否则安装失败

// 如果预缓存的文件列表过长就会增加失败的几率

return cache.addAll([

'/',

'/test.js',

'/test.css'

])

})

)

})