**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>

**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'

])

})

)

})