PWA

1. 概念

PWA (Progressive Web Apps) 是一种 Web App 新模型，并不是具体指某一种前沿的技术或者某一个单一的知识点，我们从英文缩写来看就能看出来，这是一个渐进式的 Web App，是通过一系列新的 Web 特性，配合优秀的 UI 交互设计，逐步的增强 Web App 的用户体验。

用户的手机现在几乎被各种大大小小形形色色的 App 给攻占了，手机的容量是有上限的，用户的时间成本也是有上限的，我们如何让 App 做到用户需要就能立马得到，用户不需要不占用手机资源呢？显然 Native App 是根本做不到这点的，用户能做的只会在抱怨和留舍纠结中一遍一遍的重复着安装和卸载。而另一方面 Native App 没法做到平台兼容，对于企业和开发者来说相对于 Web App 的平台兼容性以及可维护行来说 Native App 的开发运维成本太高。

PWA 工程的解决方案中借助了 [service worker](https://lavas.baidu.com/doc/offline-and-cache-loading/service-worker/service-worker-introduction) 的离线存储能力，消息推送能力以及系统的添加桌面能力，从而形成一个完善的 Web App 解决方案，帮助我们在 Web 端低成本的开发和维护一个逐步类 Native App 化的 Web App

* Web App Manifest – 在主屏幕添加app图标，定义手机标题栏颜色之类
* Service Worker – 缓存，离线开发，以及地理位置信息处理等
* App Shell – 先显示APP的主结构，再填充主数据，更快显示更好体验
* Push Notification – 消息推送

1. Web App Manifest

2.1 创建manifest.json

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

{  
 "scope": "/",  
 "name": "victorGuo",  
 "short\_name": "victorGuo",  
 "start\_url": "./index.html",  
 "display": "standalone",  
 "description": "victorGuo",  
 "dir": "rtl",  
 "lang": "cn",  
 "orientation": "portrait",  
 "theme\_color": "#3f51b5",  
 "background\_color": "#fff",  
 "icons": [{  
 "src": "./image/all.png",  
 "sizes": "80x80",  
 "type": "image/png"  
 },  
 {  
 "src": "./image/all.png",  
 "sizes": "120x120",  
 "type": "image/png"  
 },  
 {  
 "src": "./image/all.png",  
 "sizes": "200x200",  
 "type": "image/png"  
 }  
 ]  
}

1. 生命周期

install -> installed -> actvating -> Active -> Activated -> Redundant

1. service worker

3.1 service worker注册

if ('serviceWorker' in navigator) {  
 navigator.serviceWorker  
 .register('./sw.js', {scope: './'})  
 .then(*registration* => console.log('ServiceWorker 注册成功！作用域为: ', *registration*.scope))  
 .catch(*err* => console.log('ServiceWorker 注册失败: ', *err*));  
}

3.2 service安装

 列举要默认缓存的静态资源，一般用于离线使用 ，建议缓存文件不要太大。太大的文件会增加安装失败率

const *version* = 'gt\_v5';  
const *offlineResources* = [  
 './',  
 './offline.html',  
 './jquery.min.js'  
 // '/wp-content/themes/Kratos/images/default.jpg'  
];

function onInstall(*event*) {

// event.waitUtil 用于在安装成功之前执行一些预装逻辑

// 但是建议只做一些轻量级和非常重要资源的缓存，减少安装失败的概率  
 *event*.waitUntil(  
 caches.open(cacheKey('offline'))  
 .then(*cache* => *cache*.addAll(*offlineResources*))  
 .then(() => log('installation complete! version: ' + *version*))  
 .then(() => self.skipWaiting())  
 );  
}

3.3 service worker 更新

如果 sw.js 文件的内容有改动，当访问网站页面时浏览器获取了新的文件，它会认为有更新，于是会安装新的文件并触发 install 事件。但是此时已经处于激活状态的旧的 Service Worker 还在运行，新的 Service Worker 完成安装后会进入 waiting 状态。直到所有已打开的页面都关闭，旧的 Service Worker 自动停止，新的 Service Worker 才会在接下来打开的页面里生效。

如果希望在有了新版本时，所有的页面都得到及时更新怎么办呢？

可以在 install 事件中执行 skipWaiting 方法跳过 waiting 状态，然后会直接进入 activate 阶段。接着在 activate 事件发生时，通过执行 clients.claim 方法，更新所有客户端上

### 3.4 Service Worker 相关事件处理

### 3.4.1 install 事件

当前脚本被安装时，会触发 install 事件

### 3.4.2 fetch 事件

这里实现了一个缓存优先、降级处理的策略逻辑：监控所有 http 请求，当请求资源已经在缓存里了，直接返回缓存里的内容；否则使用 fetch API 继续请求，如果是 图片或 css、js 资源，请求成功后将他们加入缓存中；如果是离线状态或请求出错，则降级返回预缓存的离线内容。