link: null
title: 珠峰架构师成长计划
description: webapp用户体验差（不能离线访问），用户粘性低（无法保存入口），pwa就是为了解决这一系列问题（Progressive Web Apps）,让webapp具有快速，可靠，安全等特点
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=59 sentences=120, words=1086

什么是PWA#

webapp用户体验差（不能离线访问），用户粘性低（无法保存入口），pwa就是为了解决这一系列问题（Progressive Web Apps）,让webapp具有快速，可靠，安全等特点

PWA一系列用到的技术

- Web App Manifest
- Service Worker
- Push Api & Notification Api
- App Shell & App Skeleton
- ...

Web App Manifest

将网站添加到桌面、更类似native的体验

Web App Manifest设置

```
<link rel="manifest" href="/manifest.json">
{
    "name":"&#x73E0;&#x5CF0;&#x8BFE;&#x5802;", // &#x5E94;&#x7528;&#x540D;&#x79F0;
    "short_name":"&#x8BFE;&#x5802;", // &#x684C;&#x9762;&#x5E94;&#x7528;&#x7684;&#x540D;&#x79F0;  &#x2713;
    "display":"standalone", // fullScreen (standalone) minimal-ui browser &#x2713;
    "start_url":"", // &#x6253;&#x5F00;&#x65F6;&#x7684;&#x7F51;&#x5740;  &#x2713;
    "icons":[], // &#x8BBE;&#x7F6E;&#x684C;&#x9762;&#x56FE;&#x7247; icon&#x56FE;&#x6807;
&#x4FEE;&#x6539;&#x56FE;&#x6807;&#x9700;&#x8981;&#x91CD;&#x65B0;&#x6DFB;&#x52A0;&#x5230;&#x684C;&#x9762;icons:[{src,sizes,type}]
    "background_color":"#aaa", // &#x542F;&#x52A8;&#x753B;&#x9762;&#x989C;&#x8272;
    "theme_color":"#aaa" // &#x72B6;&#x6001;&#x680F;&#x7684;&#x989C;&#x8272;
}
```

```
&#x56FE;&#x6807;icon
<link rel="apple-touch-icon" href="apple-touch-icon-iphone.png">
&#x6DFB;&#x52A0;&#x5230;&#x4E3B;&#x5C4F;&#x540E;&#x7684;&#x6807;&#x9898; &#x548C; short_name&#x4E00;&#x81F4;
<meta name="apple-mobile-web-app-title" content="&#x6807;&#x9898;">
&#x9690;&#x85CF;safari&#x5730;&#x5740;&#x680F; standalone&#x6A21;&#x5F0F;&#x4E0B;&#x9ED8;&#x8BA4;&#x9690;&#x85CF;
<meta name="apple-mobile-web-app-capable" content="yes">
&#x8BBE;&#x7F6E;&#x72B6;&#x6001;&#x680F;&#x989C;&#x8272;
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
```

横幅安装：用户在浏览器中访问至少两次，两次访问间隔至少时间为五分钟（safari不支持横幅）

Service Worker

为了提升用户体验

Service Worker特点：

- 不能访问 / 操作dom
- 会自动休眠，不会随浏览器关闭所失效(必须手动卸载)
- 离线缓存内容开发者可控
- 必须在https或者localhost下使用
- 所有的api都基于promise

- 安装(installing)：这个状态发生在 Service Worker 注册之后，表示开始安装，触发 install 事件回调指定一些静态资源进行离线缓存。
- 安装后(installed)：Service Worker 已经完成了安装，并且等待其他的 Service Worker 线程被关闭。
- 激活(activating)：在这个状态下没有被其他的 Service Worker 控制的客户端，允许当前的 worker 完成安装，并且清除了其他的 worker 以及关联缓存的旧缓存资源，等待新的 Service Worker 线程被激活。
- 激活后(activated)：在这个状态会处理 activate 事件回调 (提供了更新缓存策略的机会)。并可以处理功能性的事件 fetch (请求)、sync (后台同步)、push (推送)。
- 废弃状态 (redundant)：这个状态表示一个 Service Worker 的生命周期结束。

serviceWorker中的方法

- self.skipWaiting():表示强制当前处在 waiting 状态的 Service Worker 进入 activate 状态
- event.waitUntil()：传入一个 Promise 为参数，等到该 Promise 为 resolve 状态为止。
- self.clients.claim()：在 activate 事件回调中执行该方法表示取得页面的控制权, 这样之后打开页面都会使用版本更新的缓存。旧的 Service Worker 脚本不再控制着页面，之后会被停止。

实现浏览器离线缓存的功能

- 注册缓存

```
window.addEventListener('load',function(){
  // &#x9875;&#x9762;&#x52A0;&#x8F7D;&#x5B8C;&#x6210;&#x540E; &#x6CE8;&#x518C;serviceWorker
  if('serviceWorker' in navigator){
      navigator.serviceWorker.register('./sw.js').then(registeration=>{
          console.log(registeration.scope);
      });
      navigator.serviceWorker.addEventListener('controllerchange',()=>{
          console.log('change')
      })
  }
  if(!navigator.onLine){
      window.addEventListener('online',()=>{
          console.log('&#x66F4;&#x65B0;');
      })
  }
});
```

- 离线缓存 应用cache缓存请求

```
// self &#x5F53;&#x524D;&#x7EBF;&#x7A0B;&#x4E2D;&#x7684;this
// &#x62E6;&#x622A;&#x7528;&#x6237;&#x53D1;&#x9001;&#x7684;&#x6240;&#x6709;&#x8BF7;&#x6C42;
let CACHE_NAME = `cache_version_` + 81;
let CACHAE_LIST = [
    '/',
    '/index.css',
    '/index.html',
    'main.js',
    '/getImage'
];
// &#x72EC;&#x7ACB;&#x7684;&#x7EBF;&#x7A0B; &#x53EF;&#x4EE5;&#x4F7F;&#x7528;fetch &#x4F46;&#x662F;&#x4E0D;&#x80FD;&#x4F7F;&#x7528;ajax
function fetchAndSave(req){
    return fetch(req).then(res=>{ // res &#x662F;&#x6D41;
        // &#x505A;&#x7F13;&#x5B58;&#x64CD;&#x4F5C;
        let r = res.clone();
        caches.open(CACHE_NAME).then(cache=>cache.put(req,r));
        return res;
    })
}
self.addEventListener('fetch', e => {
    // &#x7528;&#x76F8;&#x5E94;&#x6765;&#x66FF;&#x6362; &#x5982;&#x679C;&#x83B7;&#x53D6;&#x4E0D;&#x5230;&#x624D;&#x7528;&#x7F13;&#x5B58;
    let url = new URL(e.request.url);
    if(url.origin !== self.origin){
        return;
    }
    if(e.request.url.includes('/getImage')){ // &#x8C03;&#x7528;&#x4E86;&#x63A5;&#x53E3;
        // &#x5982;&#x679C;&#x9047;&#x5230;&#x4E86;&#x63A5;&#x53E3;  &#x66F4;&#x65B0;&#x7F13;&#x5B58;
        e.respondWith(
            fetchAndSave(e.request).catch(err=>{
                //  &#x5982;&#x679C;&#x6CA1;&#x7F51; &#x5728;&#x7F13;&#x5B58;&#x4E2D; &#x5339;&#x914D;&#x7ED3;&#x679C; &#x8FD4;&#x56DE;&#x8BF7;&#x6C42;
                return caches.match(e.request);
            })
        )
        return;
    }
    e.respondWith(
        fetch(e.request).catch(err=>{
            //  &#x5982;&#x679C;&#x6CA1;&#x7F51; &#x5728;&#x7F13;&#x5B58;&#x4E2D; &#x5339;&#x914D;&#x7ED3;&#x679C; &#x8FD4;&#x56DE;&#x8BF7;&#x6C42;
            return caches.match(e.request);
        })
    )
}); // &#x7528;&#x7F13;&#x5B58;&#x66FF;&#x6362;

// serviceWorker&#x5B89;&#x88C5;&#x7684;&#x9636;&#x6BB5;
function preCache() {
    return caches.open(CACHE_NAME).then(cache => {
        return cache.addAll(CACHAE_LIST);
    })
}
self.addEventListener('install', (e) => {
    // &#x5B89;&#x88C5;&#x7684;&#x8FC7;&#x7A0B;&#x4E2D;&#x9700;&#x8981;&#x7F13;&#x5B58;
    e.waitUntil(
        preCache().then(skipWaiting)
    )
});
function clearCache() {
    return caches.keys().then(keys => {
        return Promise.all(keys.map(key => {
            if (key !== CACHE_NAME) {
                return caches.delete(key);
            }
        }))
    })
}
self.addEventListener('activate', (e) => {
    e.waitUntil(
        Promise.all([
            clearCache(),
            self.clients.claim() // &#x7ACB;&#x5373;&#x4F7F;serviceWorker&#x751F;&#x6548;
        ])
    )
})
```

在vue中使用pwa

https://github.com/vuejs/vue-cli/tree/dev/packages/%40vue/cli-plugin-pwa (https://github.com/vuejs/vue-cli/tree/dev/packages/%40vue/cli-plugin-pwa)

```
vue create pwa-project
npm run build // &#x624D;&#x5177;&#x5907;pwa&#x6548;&#x679C;
```

vue-cli3.0配置pwa

在public目录下可以更改manifest配置文件

```
module.exports = {
  pwa: {
    name: 'My App',
    themeColor: '#f2f2f2',
    msTileColor: '#aaaaa',
    appleMobileWebAppCapable: 'yes',
    appleMobileWebAppStatusBarStyle: 'black',

    workboxPluginMode: 'InjectManifest',
    workboxOptions: {
      // swSrc is required in InjectManifest mode.

      swSrc: 'dev/sw.js',
    }
  }
}
```

需要更改registerServiceWorker文件 更改为sw.js

```
// &#x8BBE;&#x7F6E;&#x7F13;&#x5B58;&#x524D;&#x7F00;
workbox.core.setCacheNameDetails({prefix: "pwa-project"});
// &#x8BBE;&#x7F6E;&#x9884;&#x7F13;&#x5B58;&#x5217;&#x8868;
self.__precacheManifest = [].concat(self.__precacheManifest || []);
workbox.precaching.suppressWarnings();
// &#x589E;&#x52A0;&#x7F13;&#x5B58;&#x5217;&#x8868;&#x7B56;&#x7565;
workbox.precaching.precacheAndRoute(self.__precacheManifest, {});
```

基于workbox 缓存工具包 https://developers.google.com/web/tools/workbox (https://developers.google.com/web/tools/workbox)

- 内置manifest.json
- 内置serviceWorker
- 内置了缓存策略
    - cachefirst 缓存优先
    - cacheonly 仅缓存
    - networkfirst 网络优先
    - networkonly 仅网络
    - StaleWhileRevalidate 从缓存取，用网络数据更新缓存

增加缓存策略

```
workbox.routing.registerRoute(
    function(obj){
        // &#x5305;&#x6DB5;api&#x7684;&#x5C31;&#x7F13;&#x5B58;&#x4E0B;&#x6765;
        return obj.url.href.includes('/user')
    },
    workbox.strategies.staleWhileRevalidate()
);
```

app-skeleton

配置webpack插件 vue-skeleton-webpack-plugin

单页骨架屏幕

```
import Vue from 'vue';
import Skeleton from './Skeleton.vue';
export default new Vue({
    components: {
        Skeleton:Skeleton
    },
    template: `
        <skeleton></skeleton>
    `
});

plugins: [
    new SkeletonWebpackPlugin({
        webpackConfig: {
            entry: {
                app: resolve('./src/entry-skeleton.js')
            }
        }
    })
]
```

带路由的骨架屏，编写skeleton.js文件

```
import Vue from 'vue';
import Skeleton1 from './Skeleton1';
import Skeleton2 from './Skeleton2';

export default new Vue({
    components: {
        Skeleton1,
        Skeleton2
    },
    template: `

    `
});
```

```
new SkeletonWebpackPlugin({
    webpackConfig: {
        entry: {
            app: path.join(__dirname, './src/skeleton.js'),
        },
    },
    router: {
        mode: 'history',
        routes: [
            {
                path: '/',
                skeletonId: 'skeleton1'
            },
            {
                path: '/about',
                skeletonId: 'skeleton2'
            },
        ]
    },
    minimize: true,
    quiet: true,
})
```

实现骨架屏插件

```
class MyPlugin {
    apply(compiler) {
        compiler.plugin('compilation', (compilation) => {
            compilation.plugin(
                'html-webpack-plugin-before-html-processing',
                (data) => {
                    data.html = data.html.replace(``, `
                        <div id="app">
                            <div id="home" style="display:none">Home</div>
                            <div id="about" style="display:none">About</div>
                        </div>
                        <script>
                            if(window.location.hash == '#/about' || location.pathname=='/about'){
                                document.getElementById('about').style.display="block"
                            }else{
                                document.getElementById('home').style.display="block"
                            }
                        </script>
                    `);
                    return data;
                }
            )
        });
    }
}
```

vue的预渲染插件

```
npm install prerender-spa-plugin
const PrerenderSPAPlugin = require('prerender-spa-plugin')

plugins: [
    new PrerenderSPAPlugin({
        staticDir: path.join(__dirname, 'dist'),
        routes: [ '/', '/about',],
    })
]
```

Notification & Push Api

...