借助Service Worker和cacheStorage缓存及离线开发

这篇文章发布于 2017年07月12日，星期三，00:59，归类于 JS实例。 阅读 29109 次, 今日 16 次 35 条评论

一、缓存和离线开发

说得HTML5离线开发，我们通常第一反应是使用html5 manifest缓存技术，此技术已经出现很多年了，我以前多次了解过，也见过一些实践案例，但是却从未在博客中介绍过，因为并不看好。

为什么不看好呢？用一句话解释就是“投入产出比有些低”。

对于web应用，掉线不能使用是理所当然的，绝不会有哪个开发人员会因为网页在没网的时候打不开被测试MM提bug，或者被用户投诉，所以，我们的web页面不支持离线完全不会有什么影响。但如果我们希望支持离线，会发现，我投入的精力和成本啊还真不小，就说一点，html5 manifest缓存技术需要服务端配合直接就让成本蹭蹭蹭的上去了，因为已经不仅仅是前端这一个工种的工作了，很可能就需要跨团队协作，这事情立马一下子就啰嗦了，开发时候啰嗦，以后维护也啰嗦。

而支持离线的收益，我估算了下，比支持无障碍访问收益还要低。所以，站在商业的角度讲，如果一项技术成本比较高，收益比较小，是并不建议实际应用的。开发人员喜欢在重要的项目中玩些时髦的酷酷的新技术自嗨是可以理解的，但前提是收益明显。如果只是自嗨，留下烂摊，有必要警惕了，业务意识和职业素养说明还有待提高。

铺垫这么多，就是要转折说下本文要介绍的缓存技术。

本文所要介绍的基于Service Worker和cacheStorage缓存及离线开发，套路非常固定，无侵入，且语言纯正，直接复制粘贴就可以实现缓存和离线功能，纯前端，无需服务器配合。一个看上去很酷的功能只要复制粘贴就可以实现，绝对是成本极低的，小白中的小白也能上手。

由于成本几乎可以忽略不计，此时离线功能支持的投入产出比相当高了，经验告诉我这种技术以后流行和普及的可能性非常高，于是迫不及待分享给大家。

在开始案例前，我们有必要先了解几个概念。

二、通俗易懂的方式介绍Service Worker

Service Worker直白翻译就是“服务人员”，看似不经意的翻译，实际上却正道出了Service Worker的精髓所在。

举个例子，如果我们冲着叶修的集卡去麦当劳消费，如下图所示：

实际流程都是需要一个“服务人员”，客户点餐，付钱，服务人员提供食物和叶修卡，回到客户手上。

如果从最大化利用角度而言，这里的服务人员其实是多余的，客户直接付钱拿货更快捷，而这种直接请求的策略就是web请求的做法，客户端发送请求，服务器返回数据，客户端再显示。

这么多年下来，似乎web的这种数据请求策略没有任何问题，那为何现在要在中间加一个“服务人员”-Service Worker呢？

主要是用户应付一些特殊场景和需求，比方说离线处理（客官，这个卖完了），比方说消息推送（客官，你的汉堡好了……）等。而离线应用和消息推送正是目前native app相对于web app的优势所在。

所以，Service Worker出现的目的是让web app可以和native app开始真正意义上的竞争。

Service Worker概念和用法

我们平常浏览器窗口中跑的页面运行的是主JavaScript线程，DOM和window全局变量都是可以访问的。而Service Worker是走的另外的线程，可以理解为在浏览器背后默默运行的一个线程，脱离浏览器窗体，因此，window以及DOM都是不能访问的，此时我们可以使用 `self` 访问全局上下文，可参见这篇短文：“了解JS中的全局对象window.self和全局作用域self”。

由于Service Worker走的是另外的线程，因此，就算这个线程翻江倒海也不会阻塞主JavaScript线程，也就是不会引起浏览器页面加载的卡顿之类。同时，由于Service Worker设计为完全异步，同步API（如 `XHR` 和 `localStorage` ）不能在Service Worker中使用。

除了上面的些限制外，Service Worker对我们的协议也有要求，就是必须是 `https` 协议的，但本地开发也弄个 `https` 协议是很麻烦的，好在还算人性化，Service Worker在 `http://localhost` 或者 `http://127.0.0.1` 这种本地环境下的时候也是可以跑起来的。如果我们想做个demo之类的页面给其他小伙伴看，我们可以借助Github（因为是 `https` 协议的），例如我就专门建了个https-demo的小项目，专门用来放置一些需要 `https` 协议的demo页面，相信离不开 `https` 的技术场景以后会越来越多。

最后，Service workers大量使用 `Promise` ，因为通常它们会等待响应后继续，并根据响应返回一个成功或者失败的操作，这些场景非常适合 `Promise` 。如果对 `Promise` 不是很了解，可以先看我之前文章：“ES6 JavaScript Promise的感性认知”，然后再看：“JavaScript Promise迷你书（中文版）”。

Service Worker的生命周期

我们直接看一个例子吧，如下HTML和JS代码：

```
<h3>一些提示信息</h3>
<ul>
    <li>浏览器是否支持：<span id="isSupport"></span></li>
    <li>service worker是否注册成功：<span id="isSuccess"></span></li>
    <li>当前注册状态：<span id="state"></span></li>
    <li>当前service worker状态：<span id="swState"></span></li>
</ul>
```

```
<script src="./static/jquery.min.js"></script>
<script>
if ('serviceWorker' in navigator) {
    $('#isSupport').text('支持');

    // 开始注册service workers
    navigator.serviceWorker.register('./sw-demo-cache.js', {
        scope: './'
    }).then(function (registration) {
        $('#isSuccess').text('注册成功');

        var serviceWorker;
        if (registration.installing) {
            serviceWorker = registration.installing;
            $('#state').text('installing');
        } else if (registration.waiting) {
            serviceWorker = registration.waiting;
            $('#state').text('waiting');
        } else if (registration.active) {
            serviceWorker = registration.active;
            $('#state').text('active');
        }
        if (serviceWorker) {
            $('#swState').text(serviceWorker.state);
            serviceWorker.addEventListener('statechange', function (e) {
                $('#swState').append(' 状态变化为' + e.target.state);
            });
        }
    }).catch (function (error) {
        $('#isSuccess').text('注册没有成功');
    });
} else {
    $('#isSupport').text('不支持');
}
</script>
```

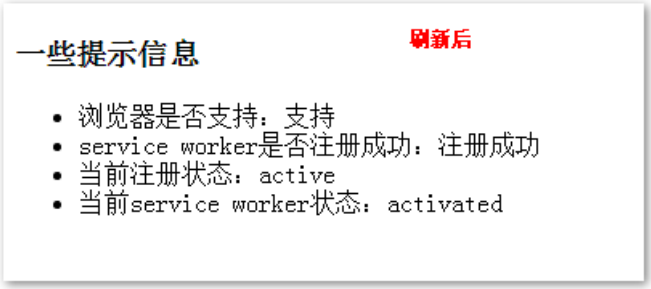
代码作用很简单，判断浏览器是否支持Service Worker以及记录Service Worker的生命周期（当前状态）。

在Chrome浏览器下，当我们第一次访问含有上面代码的页面时候，结果会是这样：

会看到：installing → installed → activating → activated。

这个状态变化过程实际上就是Service Worker生命周期的反应。

当我们再次刷新此页面，结果又会是这样：



直接显示注册成功状态。

Service Worker注册时候的生命周期是这样的：

1. Download – 下载注册的JS文件
2. Install – 安装
3. Activate – 激活

一旦安装完成，如何注册的JS没有变化，则直接显示当前激活态。

然而，实际的开发场景要更加复杂，使得Service Worker还有其它一些状态。例如下图这样：

出现了`waiting`，这是怎么出现的呢？我们修改了Service Worker注册JS，然后重载的时候旧的Service Worker还在跑，新的Service Worker已经安装等待激活。我们打开开发者工具面板，Application → Service Workers，可能就会如下图这样：

此时，我们页面强刷下会变成这样，进行了激活：

再次刷新又回到注册完毕状态。

然后，这些对应的状态，Service Worker是有对应的事件名进行捕获的，为：

```
self.addEventListener('install', function(event) { /* 安装后... */ });
```

```
self.addEventListener('activate', function(event) { /* 激活后... */ });
```

最后，Service Worker还支持`fetch`事件，来响应和拦截各种请求。

```
self.addEventListener('fetch', function(event) { /* 请求后... */ });
```

基本上，目前Service Worker的所有应用都是基于上面3个事件的，例如，本文要介绍的缓存和离线开发，`'install'`用来缓存文件，`'activate'`用来缓存更新，`'fetch'`用来拦截请求直接返回缓存数据。三者齐心，构成了完成的缓存控制结构。

Service Worker的兼容性

桌面端Chrome和Firefox可用，IE不可用。移动端Chrome可用，以后估计会快速支持。

三、了解Cache和CacheStorage

`Cache`和`CacheStorage`都是Service Worker API下的接口，截图如下：

其中，`Cache`直接和请求打交道，`CacheStorage`和Cache对象打交道，我们可以直接使用全局的`caches`属性访问CacheStorage，例如，虽然API上显示的是`CacheStorage.open()`，但我们实际使用的时候，直接`caches.open()`就可以了。

`Cache`和`CacheStorage`的出现让浏览器的缓存类型又多了一个：之前有memoryCache和diskCache，现在又多了个ServiceWorker cache。

至于`Cache`和`CacheStorage`具体的增删改查API直接去这里一个一个找，Service Worker API的知识体量实在惊人，若想要系统学习，那可要做好充足的心理准备了。

但是，如果我们只是希望在实际项目中应用一两点实用技巧，则要轻松很多。例如，我们希望在我们的PC项目或者移动端页面上渐进增强支持离线开发和更灵活的缓存控制，直接参照下面的套路即可！

四、借助Service Worker和cacheStorage离线开发的固定套路

1. 页面上注册一个Service Worker，例如：

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('./sw-demo-cache.js');
}
```

2. `sw-demo-cache.js`这个JS中复制如下代码：

```
var VERSION = 'v1';

// 缓存
self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(VERSION).then(function(cache) {
      return cache.addAll([
        './start.html',
        './static/jquery.min.js',
        './static/mm1.jpg'
      ]);
    })
  );
});

// 缓存更新
self.addEventListener('activate', function(event) {
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          // 如果当前版本和缓存版本不一致
          if (cacheName !== VERSION) {
            return caches.delete(cacheName);
          }
        })
      );
    })
```

```
  );
});

// 捕获请求并返回缓存数据
self.addEventListener('fetch', function(event) {
  event.respondWith(caches.match(event.request).catch(function() {
    return fetch(event.request);
  }).then(function(response) {
    caches.open(VERSION).then(function(cache) {
      cache.put(event.request, response);
    });
    return response.clone();
  }).catch(function() {
    return caches.match('./static/mm1.jpg');
  }));
});
```

3. 把 `cache.addAll()` 方法中缓存文件数组换成你希望缓存的文件数组。

恭喜你，简单3步曲，复制、粘贴、改路径。你的网站已经支持Service Worker缓存，甚至离线也可以自如访问，支持各种网站，PC和Mobile通杀，不支持的浏览器没有任何影响，支持的浏览器天然离线支持，想要更新缓存，`v1` 换成 `v2` 就可以，so easy!。

眼见为实，您可以狠狠地点击这里：借助Service Worker和cacheStorage缓存及离线开发demo

进入页面，我们勾选network中的Offline，如下图：

结果刷新的时候，页面依然正常加载，如下gif截屏：

我们离线功能就此达成，出乎意料的简单与实用。

五、和PWA技术的关系

PWA全称为“Progressive Web Apps”，渐进式网页应用。功效显著，收益明显，如下图：

PWA的核心技术包括：

- 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，之前有写过“简单了解HTML5中的Web Notification桌面通知”

有此可见，Service Worker仅仅是PWA技术中的一部分，但是又独立于PWA。也就是虽然PWA技术面向移动端，但是并不影响我们在桌面端渐进使用Service Worker，考虑到自己厂子用户70%~80%都是Chrome内核，收益或许会比预期的要高。

六、Service Worker更多的应用场景

Service Worker除了可以缓存和离线开发，其可以应用的场景还有很多，举几个例子（参考自MDN）：

- 后台数据同步

- 响应来自其它源的资源请求，
- 集中接收计算成本高的数据更新，比如地理位置和陀螺仪信息，这样多个页面就可以利用同一组数据
- 在客户端进行CoffeeScript，LESS，CJS/AMD等模块编译和依赖管理（用于开发目的）
- 后台服务钩子
- 自定义模板用于特定URL模式
- 性能增强，比如预取用户可能需要的资源，比如相册中的后面数张图片

开动自己的脑力，很多想法直接就可以前端实现。

例如我想到了一个：重构人员写高保真原型的时候，模拟请求伪造数据的时候，可以不用依赖web环境，直接在Service Worker中拦截和返回模拟数据，于是整个项目只有干净的HTML、CSS和JS。

好了，就这么多。

内容较多，行为仓促，有错误在所难免，欢迎指正！

也欢迎交流，感谢阅读！

> 《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« 了解JS中的全局对象window.self和全局作用域self

HTML5 indexedDB前端本地存储数据库实例教程 »

猜你喜欢

- 了解JS中的全局对象window.self和全局作用域self
- Service Worker实现浏览器端页面渲染或CSS,JS编译
- HTML5 indexedDB前端本地存储数据库实例教程
- 翻译：清除各个浏览器中的数据研究
- 简单了解HTML5中的Web Notification桌面通知
- 翻译 - CSS Sprites:实用技术还是生厌之物?
- 翻译：web制作、开发人员需知的Web缓存知识
- 突破本地离线存储5M限制的JS库localforage简介

分享到：1

标签： app shell, cache, CacheStorage, fetch, manifest, Notification, pwa, Service Worker, 离线存储, 缓存

发表评论（目前35条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. **leto说道：**
2018年10月16日 20:10

博主写的简单明了，顶一下。但是 fetch 中确实有错误，百度网站给的这段代码不错：https://lavas.baidu.com/pwa/offline-and-cache-loading/service-worker/how-to-use-service-worker

回复

2. **leafront说道：**
2018年10月8日 16:04

缓存start.html页面会不会导致start.html做了修改，不会更新内容。

回复

> **张 鑫旭说道：**
> 2018年10月9日 10:15
>
> 会，此时需要你update缓存版本，拉取新资源缓存。
>
> 回复

3. **7qin说道：**
2018年09月11日 22:51

无法访问此网站
网址为 https://zhangxinxu.github.io/https-demo/cache/start.html 的网页可能暂时无法连接，或者它已永久性地移动到了新网址。
ERR_FAILED

回复

> **7qin说道：**
> 2018年09月11日 22:53
>
> sw-demo-cache.js:38 Uncaught (in promise) TypeError: Failed to execute ‘put’ on ‘Cache’: parameter 2 is not of type ‘Response’.
> at sw-demo-cache.js:38
>
> 回复

> **张 鑫旭说道：**
> 2018年09月17日 14:29
>
> 再试试，github地址有时候不稳定
>
> 回复

4. **webphate说道：**
2018年08月22日 12:01

caches.match(event.request).catch(function() { // 在我本地跑demo，如果caches.match没有匹配到，会返回resolve（undefiined），catch捕获不到
return fetch(event.request);
}).then(function(response) { // response 为undefined会报错
caches.open(VERSION).then(function(cache) {
cache.put(event.request, response);
});

回复

> **webphate说道：**
> 2018年08月22日 12:05
>
> 原来已经有人已经在别的评论中回复了，
> “bobo说道：
> 2018年05月21日 16:37”
>
> 回复

5. **shenzm说道：**
2018年05月11日 18:21

例子中的 fetch事件的实现是有问题的。进入catch里的话，说明缓存没有匹配到，这个时候是去服务器上获取资源，之后并没有添加到缓存中去。

这样如果在清楚掉cache时，以后就都不会缓存住了

回复

> **bobo说道：**
> 2018年05月21日 16:37
>
> this.addEventListener(‘fetch’, function(event) {
> event.respondWith(
> caches.match(event.request).then(function(resp) {
> return resp || fetch(event.request).then(function(response) {
> return caches.open(VERSION).then(function(cache) {
> cache.put(event.request, response.clone())
> return response
> })
> })
> })
>)
> })
>
> 回复

6. **Victor说道：**
2018年04月26日 19:55

我现在有个需求想实现，不知道借助ServiceWorker能否实现： 我需要用js代码在本地不断的缓存二进制数据（音视频流转的Blob数据，非常大，可能几个G），完了之后能够取到这些数据并生成本地文件下载或者上传到服务器。。
现在浏览器会自动缓存blob但是是临时的，刷新页面或关闭浏览器就没有了，而我需要能一直存储，只能js或者用户手动删除。。哪位大神有什么好的建议吗?

回复

7. **janmi说道：**
2017年11月30日 18:24

在更新sw.js或注销sw，只有第二次访问才会更新资源，想请教一下贵厂是怎么处理这种问题的?

回复

dq说道：
2018年02月11日 15:16

self.skipWaiting 不行么

回复

8. 王晓聪说道：
2017年11月3日 08:56

请教一个问题，静态的html页面如何做更新？比如start.html做了部分修改，好像做不到更新，加载的还会是缓存中的页面

回复

9. 阿不晕说道：
2017年10月24日 12:53

可以缓存绝对路径么

回复

10. 飘过IP说道：
2017年09月14日 16:18

报这个错误：Uncaught (in promise) DOMException: Only secure origins are allowed

回复

张 鑫旭说道：
2017年09月15日 23:08

应该是因为页面不是https协议

回复

11. 飘过IP说道：
2017年09月14日 15:25

service worker是否注册成功：注册没有成功
当前注册状态：缺省
当前service worker状态：缺省

请问什么原因?

回复

哼哈大将说道：
2017年12月28日 15:22

我也是遇到这样的问题，球解答

回复

dq说道：
2018年02月11日 15:17

https证书不是可信的，不会注册sw的

回复

12. xiaobin说道：
2017年08月28日 15:21

简单易懂，很厉害

回复

13. **Dolphin说道：**
2017年08月9日 04:20

说到用 service worker 模拟 API，就不得不安利下我们的 service mocker 了：https://github.com/service-mocker/service-mocker

回复

14. **mfk说道：**
2017年07月21日 13:52

大哥，你那个demo为什么第一次打开会非常的慢？？

回复

> 张 鑫旭说道：
> 2017年07月22日 10:45
>
> 因为github慢~
>
> 回复

15. **前端说道：**
2017年07月20日 20:56

html5 manifest也可以纯前端！

回复

16. **神秘博士说道：**
2017年07月19日 13:34

CacheStorage 最大容量是多大?
Html manifest缓存最大才5MB

回复

17. **Web前端之家说道：**
2017年07月18日 16:35

离线开发现在是潮流啊。。。这个知识点分享得不错，学习。。。

回复

18. **zero说道：**
2017年07月13日 16:58

跟web worker有什么区别?

回复

19. **lucy杨说道：**
2017年07月13日 15:28

(011).toString(2) 与 (11).toString(2) 不理解这其中的猫腻

回复

> **lyj说道：**
> 2017年07月19日 09:53
>
> (011)是八进制的先转换成十进制的9然后转成二进制的1001；而11就直接是十进制的11，转成二进制的1011
>
> 回复

20. **moocss**说道：
2017年07月12日 17:22

看好PWA。

回复

21. **ClayIdols**说道：
2017年07月12日 14:47

赞，做 WebApp 主页面可以缓存起来，体验好很多

回复

22. **GIN**说道：
2017年07月12日 09:39

好顶赞!

回复

23. 亚里士朱德说道：
2017年07月12日 08:57

用websocket和后台同步数据不是更好么?

回复

> **r**说道：
> 2017年07月12日 11:21
>
> 
>
> 根本不是一回事
>
> 回复

最新文章

- 常见的CSS图形绘制合集
- 粉丝群第1期CSS小测点评与答疑
- 分享三个纯CSS实现26个英文字母的案例
- 小tips: 纯CSS实现打字动画效果
- CSS/CSS3 box-decoration-break属性简介
- CSS :placeholder-shown伪类实现Material Design占位符交互效果
- 从天猫某活动视频不必要的3次请求说起
- CSS vector-effect与SVG stroke描边缩放
- CSS ::backdrop伪元素是干嘛用的?
- 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

- 常见的CSS图形绘制合集 (190)
- 未来必热：SVG Sprite技术介绍 (119)
- 粉丝群第1期CSS小测点评与答疑 (115)
- HTML5终极备忘大全（图片版+文字版） (93)
- 让所有浏览器支持HTML5 video视频标签 (86)
- Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
- CSS3下的147个颜色名称及对应颜色值 (79)
- 视区相关单位vw, vh..简介以及可实际应用场景 (76)
- 写给自己看的display: flex布局教程 (76)
- 小tips: 纯CSS实现打字动画效果 (76)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 了解JS中的全局对象window.self和全局作用域self
- Service Worker实现浏览器端页面渲染或CSS,JS编译
- HTML5 indexedDB前端本地存储数据库实例教程
- 翻译：清除各个浏览器中的数据研究
- 简单了解HTML5中的Web Notification桌面通知
- 翻译 - CSS Sprites:实用技术还是生厌之物?
- 翻译：web制作、开发人员需知的Web缓存知识
- 突破本地离线存储5M限制的JS库localforage简介