# Service Workers初探

## Service Workers出现的背景

有一个困扰web用户多年的难题：网络不可连接（离线）。即使是世界上最好的web app，如果你下载不了它，用户体验基本是毁了。已经有很多种技术尝试，来解决这一问题。随着离线页面的出现，一些问题已经得到了解决。但是，最重要的问题是，仍然没有一个好的统筹机制，来对缓存和网络请求进行控制。

Service workers应该最终解决了这些问题。Service Workers可以使用JavaScript，更加灵活和细粒度地控制你的应用的缓存资源。有了它，你可以解决目前离线应用的问题，同时也可以做更多的事。使用Service Workers可以使你的应用先访问本地缓存，所以在离线状态时，在没有通过网络接收到更多的数据前，仍可以提供基本的功能体验（一般称之为 Offline First）。这是原生APP本来就支持的功能，这也是相比于 web app，原生app更受青睐主要原因之一。

## Service Worker 是什么？

一个 services worker s是一段运行在浏览器后台进程里的脚本，它独立于当前页面，提供了那些不需要与web页面交互的功能在网页背后悄悄执行的能力。在将来，基于它可以实现消息推送，静默更新以及地理围栏等服务。

在service workers提出之前，另外一个提供开发者离线体验的API叫做App Cache。然而App Cache有些局限性，例如它可以很容易地解决单页应用的问题，但是在多页应用上会很麻烦，而Service workers的出现正是为了解决App Cache的痛点。

## Service Workers的生命周期

Service workers拥有一个完全独立于Web页面的生命周期。

要让一个service workers在你的网站上生效，你需要先在你的网页中注册它。注册一个service workers之后，浏览器会在后台默默启动一个service workers的安装过程。

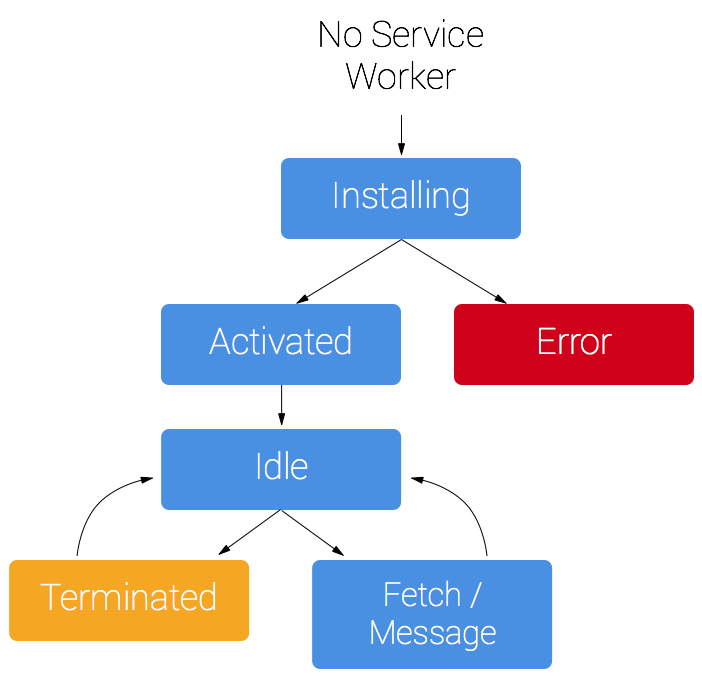
在安装过程中，浏览器会加载并缓存一些静态资源。如果所有的文件被缓存成功，service workers就安装成功了。如果有任何文件加载或缓存失败，那么安装过程就会失败，service workers就不能被激活（也即没能安装成功）。如果发生这样的问题，别担心，它会在下次再尝试安装。

当安装完成后，service workers的下一步是激活。

在激活之后，service workers将接管所有在自己管辖域范围内的页面，但是如果一个页面是刚刚注册了service workers，那么它这一次不会被接管，到下一次加载页面的时候，service worker才会生效。

当service workers接管了页面之后，它可能有两种状态：要么被终止以节省内存，要么会处理fetch和message事件，这两个事件分别产生于一个网络请求出现或者页面上发送了一个消息。

下图是一个简化了的service worker初次安装的生命周期：

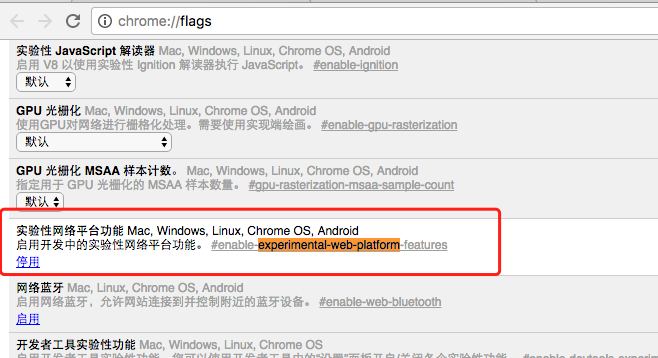


## 使用Service Workers前的设置

在已经支持serivce workers的浏览器的较新版本中，很多serivce workers的特性默认没有开启支持。如果你发现示例代码在代当前版本的浏览器中怎么样都无法正常运行，你可能需要开启一下浏览器的相关配置：

Firefox Nightly: 访问 about:config 并设置 dom.serviceWorkers.enabled 的值为 true; 重启浏览器；

Chrome Canary: 访问 chrome://flags 并开启 experimental-web-platform-features; 重启浏览器 (注意：有些特性在Chrome中没有默认开房支持)；



Opera: 访问 opera://flags 并开启 Service Workers的支持; 重启浏览器。

另外，你需要通过HTTPS来访问你的页面代码 — 出于安全原因，Service Workers严格要求要在HTTPS下才能运行。

## 注册Service Workers

要安装service workers，你需要在你的页面上注册它。这个步骤告诉浏览器你的service workers脚本在哪里。

if ('serviceWorker' in navigator) {

navigator.serviceWorker.register('/sw.js').then(function(registration) {

// Registration was successful

console.log('ServiceWorker registration successful with scope: ', registration.scope);

}).catch(function(err) {

// registration failed :(

console.log('ServiceWorker registration failed: ', err);

});

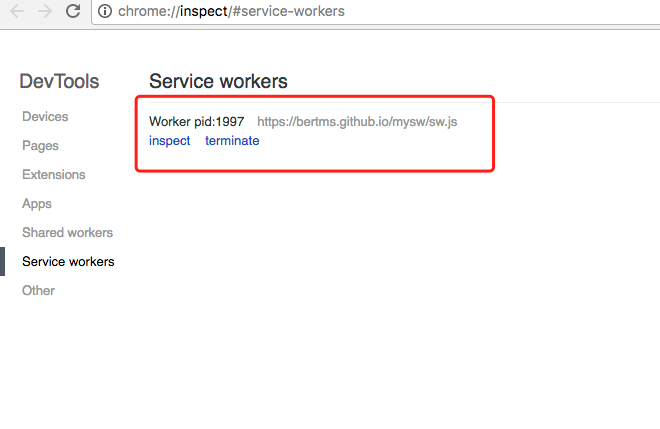
}

上面的代码检查service workers API是否可用，如果可用，service workers /sw.js 被注册。

如果这个service workers已经被注册过，浏览器会自动忽略上面的代码。

有一个需要特别说明的是service workers文件的路径，你一定注意到了在这个例子中，service workers文件被放在这个域的根目录下，这意味着service workers和网站同源。换句话说，这个service works将会收到这个域下的所有fetch事件。如果我将service workers文件注册为/example/sw.js，那么，service workers只能收到/example/路径下的fetch事件（例如： /example/page1/, /example/page2/）。

现在你可以到 chrome://inspect/#service-workers 检查service workers是否对你的网站启用了。



## Service Workers的安装步骤

在页面上完成注册步骤之后，让我们把注意力转到service workers的脚本里来，在这里面，我们要完成它的安装步骤。

在最基本的例子中，你需要为install事件定义一个callback，并决定哪些文件你想要缓存。

// The files we want to cache

var urlsToCache = [

'/',

'/styles/main.css',

'/script/main.js'

];

// Set the callback for the install step

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

// Perform install steps

});

在我们的install callback中，我们需要执行以下步骤：

1开启一个缓存

2缓存我们的文件

3决定是否所有的资源是否要被缓存

var CACHE\_NAME = 'my-site-cache-v1';

var urlsToCache = [

'/',

'/styles/main.css',

'/script/main.js'

];

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

// Perform install steps

event.waitUntil(

caches.open(CACHE\_NAME)

.then(function(cache) {

console.log('Opened cache');

return cache.addAll(urlsToCache);

})

);

});

上面的代码中，我们通过caches.open打开我们指定的cache文件名，然后我们调用cache.addAll并传入我们的文件数组。这是通过一连串promise（caches.open 和 cache.addAll）完成的。event.waitUntil拿到一个promise并使用它来获得安装耗费的时间以及是否安装成功。

如果所有的文件都被缓存成功了，那么service workers就安装成功了。如果任何一个文件下载失败，那么安装步骤就会失败。这个方式允许你依赖于你自己指定的所有资源，但是这意味着你需要非常谨慎地决定哪些文件需要在安装步骤中被缓存。指定了太多的文件的话，就会增加安装失败率。

上面只是一个简单的例子，你可以在install事件中执行其他操作或者甚至忽略install事件。

## 怎样缓存和返回Request

你已经安装了service workers，你现在可以返回你缓存的请求了。

当service workers被安装成功并且用户浏览了另一个页面或者刷新了当前的页面，service workers将开始接收到fetch事件。下面是一个例子：

self.addEventListener('fetch', function(event) {

event.respondWith(

caches.match(event.request)

.then(function(response) {

// Cache hit - return response

if (response) {

return response;

}

return fetch(event.request);

}

)

);

});

上面的代码里我们定义了fetchs事件，在event.respondWith里，我们传入了一个由caches.match产生的promise.caches.match 查找request中被service worker缓存命中的response。

如果我们有一个命中的response，我们返回被缓存的值，否则我们返回一个实时从网络请求fetch的结果。这是一个非常简单的例子，使用所有在install步骤下被缓存的资源。

## 如何更新一个Service Workers

你的service workers总有需要更新的那一天。当那一天到来的时候，你需要按照如下步骤来更新：

更新你的service workers的JavaScript文件

当用户浏览你的网站，浏览器尝试在后台下载service workers的脚本文件。只要服务器上的文件和本地文件有一个字节不同，它们就被判定为需要更新。

更新后的service workers将开始运作，install event被重新触发。

在这个时间节点上，当前页面生效的依然是老版本的service workers，新的servicer workers将进入"waiting"状态。

当前页面被关闭之后，老的service workers进程被杀死，新的servicer workers正式生效。

一旦新的service workers生效，它的activate事件被触发。

代码更新后，通常需要在activate的callback中执行一个管理cache的操作。因为你会需要清除掉之前旧的数据。我们在activate而不是install的时候执行这个操作是因为如果我们在install的时候立马执行它，那么依然在运行的旧版本的数据就坏了。

之前我们只使用了一个缓存，叫做my-site-cache-v1，其实我们也可以使用多个缓存的，例如一个给页面使用，一个给blog的内容提交使用。这意味着，在install步骤里，我们可以创建两个缓存，pages-cache-v1和blog-posts-cache-v1，在activite步骤里，我们可以删除旧的my-site-cache-v1。

下面的代码能够循环所有的缓存，删除掉所有不在白名单中的缓存。

self.addEventListener('activate', function(event) {

var cacheWhitelist = ['pages-cache-v1', 'blog-posts-cache-v1'];

event.waitUntil(

caches.keys().then(function(cacheNames) {

return Promise.all(

cacheNames.map(function(cacheName) {

if (cacheWhitelist.indexOf(cacheName) === -1) {

return caches.delete(cacheName);

}

})

);

})

);

});

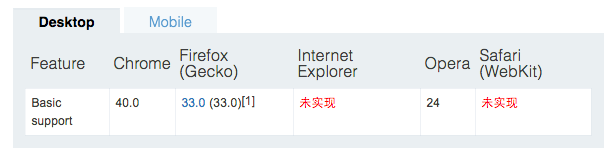
现在已经有很多用到sevice workers的项目， 例如：The Washington Post，Flip Board，AliExpress，Wikipedia，Gmail，Booking 等等，有兴趣的可以去安装体验一下。

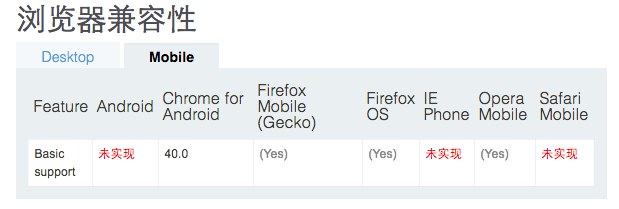
## Service workers 存在的一些问题

1 默认情况下，从第三方URL跨域得到一个资源将会失败，除非对方支持了CORS。

2 不支持重定向。

3 兼容性还不够全面。





# 谢谢大家