Service Worker大纲

简介

什么是 ServiceWorker

ServiceWorker提供了一个一对一的代理服务器，它可以拦截浏览器的请求，然后根据自己的逻辑来处理这些请求，比如可以直接返回缓存的资源，或者从网络上获取资源，然后将资源缓存起来，再返回给浏览器。

与webworker的不同：

Service Worker是浏览器进程，Web Worker是浏览器内核进程

不会虽页面关闭而销毁

可以多页面使用

Service Worker只能在https或者localhost环境下使用

使用场景

网络请求缓存，如静态资源（HTML、CSS、JavaScript、图片等），接口请求；

消息通知

PWA

WebRTC

浏览器插件，不依赖任何一个网页tab，作为一个单独的线程可以长期在后台运行

使用

注册 ServiceWorker

ServiceWorker 生命周期

service worker 是一个特殊的 web Worker，因此他与页面通信和 worker 是一样的，同样不能访问 DOM。特殊在于他是由事件驱动的具有生命周期的 worker，并且可以拦截处理页面的网络请求(fetch)，可以访问 cache 和 IndexDB。service Worker 可以让开发者自己控制管理缓存的内容以及版本

ServiceWorke的生命周期有三个阶段，分别是安装install、激活activate和运行fetch。

安装install

安装阶段是在ServiceWorker注册成功之后，浏览器开始下载ServiceWorker脚本的阶段；这个阶段是一个异步的过程，我们可以在install事件中监听它，它的回调函数会接收到一个event对象；我们可以通过event.waitUntil来监听它的完成状态，当它完成之后，我们需要调用event.waitUntil的参数, 可以做一些缓存的操作.

注意这个参数是一个Promise对象，当这个Promise对象完成之后，浏览器才会进入下一个阶段，它会阻塞浏览器的安装，如果你的Promise对象一直没有完成，那么浏览器就会一直处于安装的状态，这样会影响到浏览器的正常使用。

激活activate

激活阶段是在安装完成之后，浏览器开始激活ServiceWorker的阶段；

这个阶段也是一个异步的过程，我们可以在activate事件中监听它，它的回调函数会接收到一个event对象；不同于安装阶段，激活阶段不需要等待event.waitUntil的传递的Permise对象完成，它会立即进入下一个阶段。但是如果传递一个一直处于pending状态的Promise对象会导致ServiceWorker 一直处在某一个状态而无法响应，导致浏览器卡死。

运行fetch

这个阶段是一个长期存在的过程，我们可以在fetch事件中监听它，它的回调函数会接收到一个event对象；任何请求拦截都是在这个阶段进行的，我们可以在这个阶段中对请求进行拦截，然后返回我们自己的响应。

ServiceWorker 请求拦截

ServiceWorker的优先级是最高的，它会拦截所有的请求，包括插件的请求。

ServiceWorker 监听事件

install：安装事件，当ServiceWorker安装成功后，就会触发这个事件，这个事件只会触发一次。

activate：激活事件，当ServiceWorker激活成功后，就会触发这个事件，这个事件只会触发一次。

fetch：网络请求事件，当页面发起网络请求时，就会触发这个事件。

push：推送事件，当页面发起推送请求时，就会触发这个事件。

sync：同步事件，当页面发起同步请求时，就会触发这个事件。

message：消息事件，当页面发起消息请求时，就会触发这个事件。

messageerror：消息错误事件，当页面发起消息错误请求时，就会触发这个事件。

error：错误事件，当页面发起错误请求时，就会触发这个事件。

ServiceWorker 缓存

ServiceWorker来缓存我们的静态资源， ServiceWorker的缓存是基于CacheStorage的，它是一个Promise对象，我们可以通过caches来获取。

添加缓存：cache.put来添加缓存，它接收两个参数，第一个参数是Request对象，第二个参数是Response对象；

获取缓存：cache.match来获取缓存，它接收一个参数，这个参数可以是Request对象，也可以是URL字符串；

删除缓存：cache.delete来删除缓存，它接收一个参数，这个参数可以是Request对象，也可以是URL字符串；

清空缓存：cache.keys来获取缓存的key，然后通过cache.delete来删除缓存；

ServiceWorker 缓存策略

ServiceWorker的缓存策略是基于fetch事件的，我们可以在fetch事件中监听请求，然后对请求进行拦截，然后返回我们自己的响应；

缓存资源

通常我们会在install事件中缓存一些资源，因为install事件只会触发一次，并且会阻塞activate事件，所以我们可以在install事件中缓存一些资源，然后在activate事件中删除一些旧的资源；缓存的所有资源一定都是确定的存在的，不能出现除状态码为 200 以外的其他状态码，否则缓存会失败；使用caches.match来匹配请求，如果匹配到了，那么就返回缓存的资源，如果没有匹配到，那么就从网络中获取资源，这也就是我们刚才提到的缓存策略：缓存优先。

缓存更新

push消息推送

按照正常的消息推送流程，我们需要经历下面几个步骤：

注册ServiceWorker获得registration对象

通过registration对象获得PushManager对象

通过PushManager对象订阅消息推送，获得subscription对象

将subscription对象发送给服务器，由服务器保存

服务器通过subscription对象推送消息

ServiceWorker通过监听push事件，获得推送的消息

ServiceWorker通过showNotification方法，显示消息

// 1. 注册`ServiceWorker`获得`registration`对象

// 2. 通过`registration`对象获得`PushManager`对象

// 3. 通过`PushManager`对象订阅消息推送，获得`subscription`对象

// 4. 将`subscription`对象发送给服务器，由服务器保存

// 5. 服务器通过`subscription`对象推送消息

// 6. `ServiceWorker`通过监听`push`事件，获得推送的消息

// 7. `ServiceWorker`通过`showNotification`方法，显示消息

消息订阅到服务器推送流程：

服务和浏览器之间是没有直接的通信的，而是通过谷歌FCM服务器来进行通信的。

离线推送：

如果你的网页或者浏览器不在线的时候，推送消息就会被FCM服务器保存起来，等你的网页或者浏览器上线的时候，FCM服务器会通过notificationcallback回调来推送消息到你的网页或者浏览器

sync后台同步

sync事件是用来同步数据的，sync的使用场景有很多，如离线数据同步、数据备份、数据恢复、数据同步

我们在index.html中请求后台查询数据是否有变化，如果有变化，那么就注册一个sync事件，然后在Service Worker中监听这个事件，当事件触发的时候，就会执行syncData方法，这个方法中会请求后台获取数据，然后更新缓存。

Demo