PWA 在阿里巴巴交易履约业务中的实践

阿里巴巴 ICBU 事业部 林燕燕

2021.11.27

大纲

01 业务简介

02 核心场景

03 PWA

04 具体实践

01

业务简介

ICBU 事业部

ICBU 是阿里巴巴的国际事业部，主要负责阿里国际站（Alibaba.com）业务，是全球最大的跨境贸易与服务平台。

交易履约业务

阿里巴巴国际站上成交一笔交易之后，通过遍布全国的合作伙伴“一拍档”，为卖家提供通关、外汇、物流、金融、退税等外贸综合服务。

赋能拍档

为拍档提供数字化能力，帮助他们触达客户，更好提供服务。

服务考核

对拍档服务客户的过程加以考核，以保证服务的高质量完成。

02

核心场景

上门拜访
VISIT

1. 浏览客户信息，确定即将拜访客户

2. 导航到客户工厂地址，上报定位进行打卡

3. 完成客户拜访，在系统中录入拜访记录

痛点问题
PROBLEM

工厂地处郊区
地址偏远

无法获取用户
定位

弱网
网络请求不畅

设备离线
无法访问系统

问题总结

弱网、无网时
如何完成拜访

PWA

03

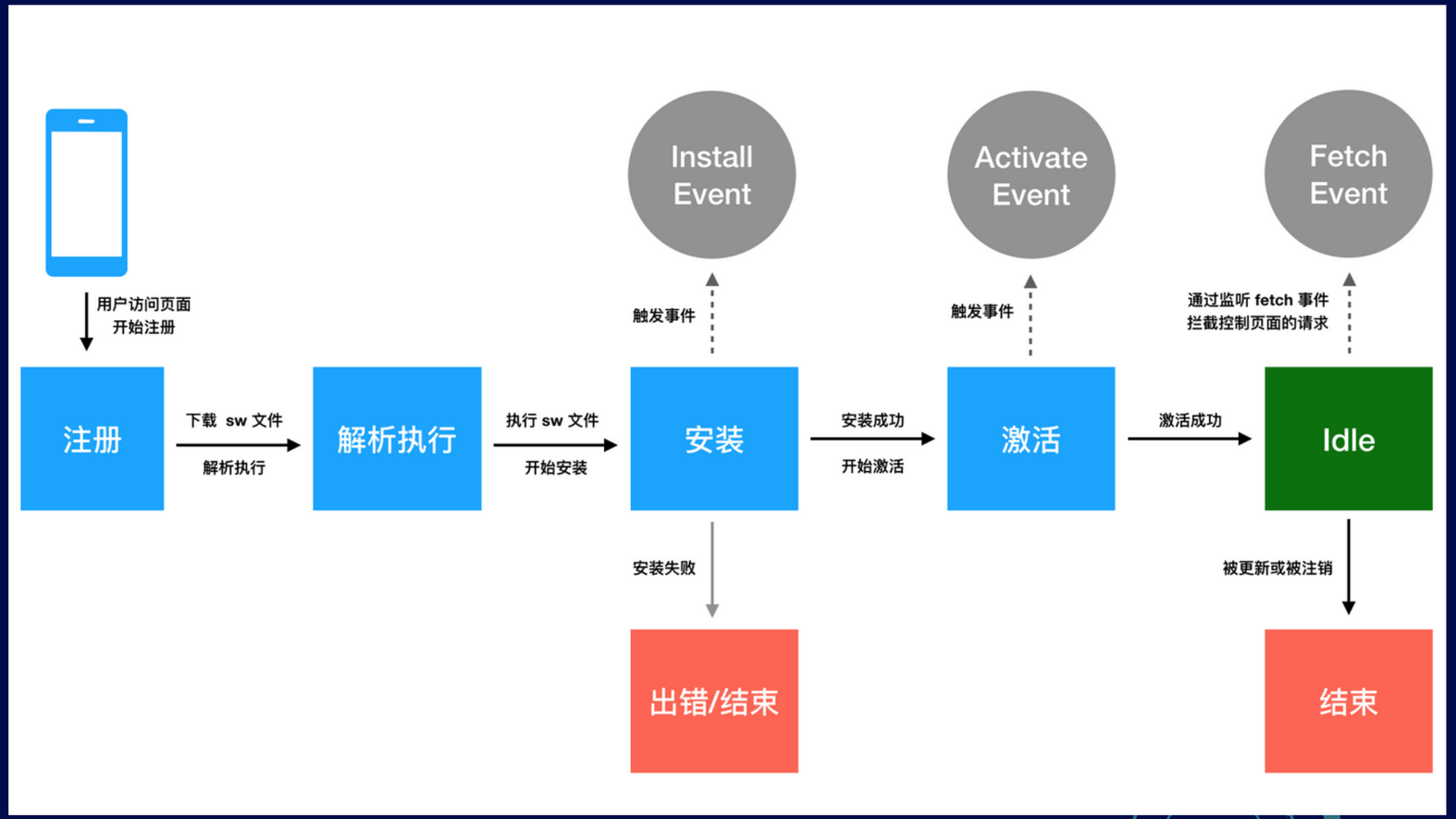
PWA

Progressive Web App

1. Service Worker
   缓存，站点离线化
2. 可拦截请求
   离线操作
3. 网络恢复
   数据自动同步
4. 可安装
   享独立入口

Service Worker

Service Worker (线程)是独立于浏览器主线程的工作线程，完全隔离，独立 Context



Workbox

Google Chrome 官方提供的一套可方便使用 Service Worker 离线能力的解决方案

```
importScripts('/workbox-sw.min.js');
```

workbox.routing.registerRoute (capture, handler, method)

路由请求缓存

- 通过 capture 请求路由匹配到指定待缓存请求
- 通过 handler 回调函数决定用何种策略来缓存匹配上的文件
- HTTP method, 默认 GET

workbox.strategies

缓存策略

- 针对多种应用场景的缓存策略
- 相对于裸写 service worker 逻辑来说，更轻松更可读

workbox.backgroundSync.Queue (QueueName)

重试队列

- 主要用于处理离线状态下的 POST 请求
- 可将失败请求添加到 IndexedDB 中，待网络恢复后依次出队，重新发起请求。

04

具体实践

非全站 PWA

整个站点有多个子应用，但有离线需求的只有上门拜访相关的场景，所以只针对其做 PWA 改造，不侵入其他应用。

SPA 页面

只围绕「上门拜访」这一个场景提供服务，SPA 接入更简单，且能带来更好的用户交互体验。

HTTPS

Service Worker 仅运行于 HTTPS 协议下。

sw.js

需要部署在与页面同 host 下才能顺利引入。

manifest.json

部署到核心场景对应的 scope 下，限定其生效的作用域，其余页面仍通过浏览器方式显示。

workbox

Google 官方离线缓存解决方案。

链路重新梳理

浏览客户信息 → 前往客户地址 → 离线定位 获准打卡 → 录入拜访数据

桌面入口
静默缓存

资料展示

经纬度校验

联网同步

有网 / 弱网
online

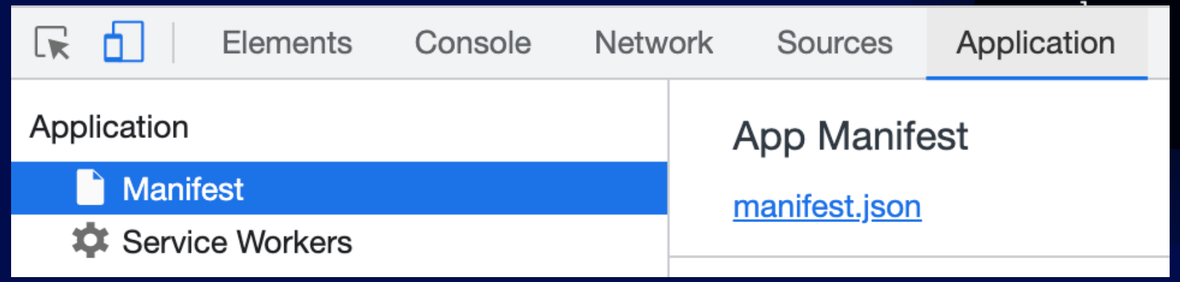
离线
offline

manifest.json

启动入口。
可配置名称、启动地址、图标、主题等。

由于业务入口较深，外层应用非我们维护，所以这种直接入口显得尤为重要。

```
{
    "name": "上门拜访",
    "short_name": "上门拜访",
    "start_url": "./index.htm",
    "display": "minimal-ui",
    "theme_color": "#fff",
    "background_color": "#fff",
    "icons": [
        {
            "src": "https://img.alicdn.com/imgextra/i2/O1CN0192-192.png",
            "sizes": "192x192",
            "type": "image/png"
        }
```



sw.js 实时更新

Service Worker 有 Diff 机制，但 sw.js 可能被浏览器缓存，导致 Diff 不能实时进行。

```
{
  'Cache-Control': 'no-store',
  'Pragma': 'no-cache'
}
```

服务端控制不缓存

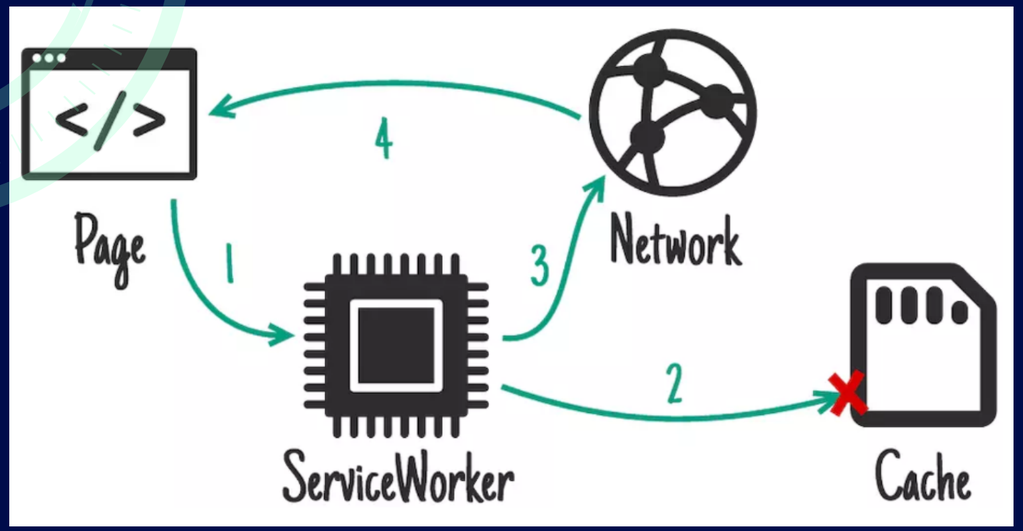
监听到 sw.js 更新，调用 skipWaiting 强制新老 SW 替换，替换完成后主动刷新页面，保证逻辑最新。

```
const skipWaiting = (event) => {
    wb.addEventListener('controlling', () => {
        window.location.reload();
    });

    wb.messageSkipWaiting();
}

wb.addEventListener('waiting', skipWaiting);
```

新 sw.js 替换老的

分策略缓存

Cache First

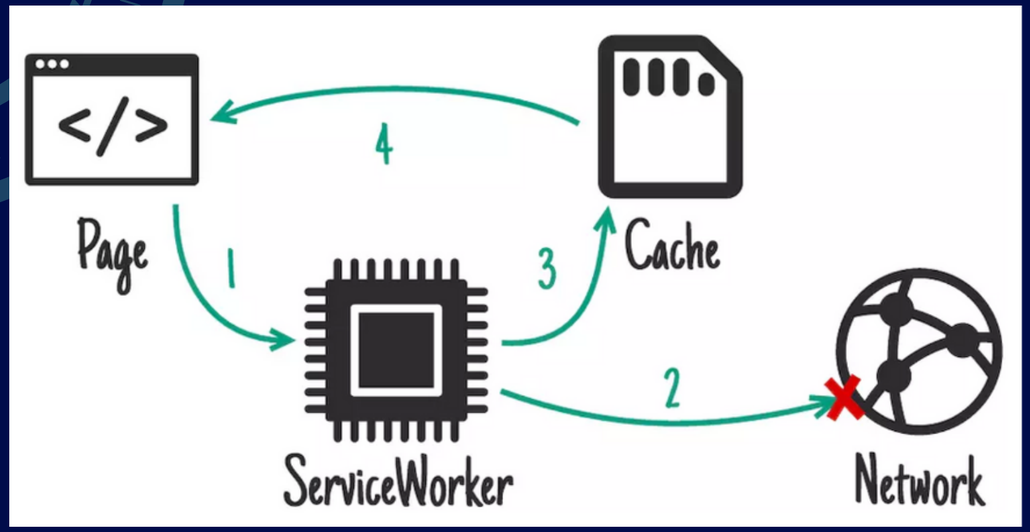


适用于缓存不常变动，对实时性要求不高的资源，如通用 js/css

```
workbox.routing.registerRoute(
    new RegExp('.*common\.(?:js|css)'),
    new workbox.strategies.CacheFirst({
        matchOptions: {
            // 此类资源不需要实时更新，所以不判断其 search
            ignoreSearch: true
        }
    })
);
```

分策略缓存

Network First

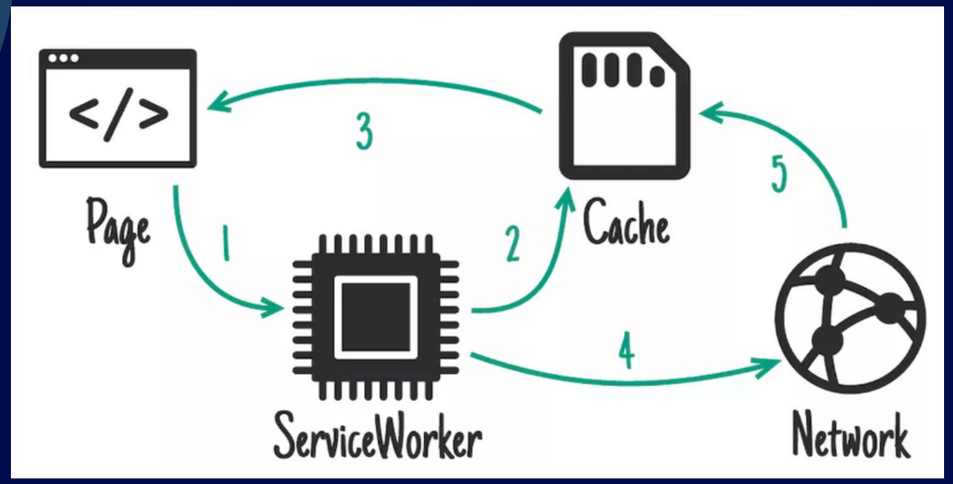


适用于缓存经常变动，对实时性有要求的文件，比如页面 HTML

```
workbox.routing.registerRoute(
  new RegExp('.*\.html'),
  new workbox.strategies.NetworkFirst()
);
```

分策略缓存

Stale While Revalidate



适用于需要保证页面响应速度，又有一定实时性要求的情况，

比如 CDN 上的资源文件

```
workbox.routing.registerRoute(
    new RegExp('https://your\.cdn\.com/'),
    new workbox.strategies.StaleWhileRevalidate({
        plugins: [
            // 默认并不支持第三方请求缓存(因为状态不可预测)，所以需要使用插件判断资源状态
            new workbox.cacheableResponse.CacheableResponsePlugin({
                statuses: [0, 200]
            })
        ]
    })
);
```

离线定位

HTML 5 原生 API 获取用户设备当前定位

- 需经过用户授权之后方可获取
- 离线模式下需设置把超时时间设置长一些

```
navigator.geolocation.getCurrentPosition(res => {
    console.log(`latitude: ${res.coords && res.coords.latitude}`);
    console.log(`longitude: ${res.coords && res.coords.longitude}`);
}, err => {
    console.log(`err: ${err.message}`);
}, { timeout: 8000 })
```

联网同步

用户进行打卡的请求，以及拜访完成之后，录拜访的请求，都可能受限于网络而无法完成。

针对这类 POST 请求进行监听并拦截，请求失败时将整个请求添加到队列中，待网络恢复后重试。由此实现离线时可操作，联网后即时同步。

```
const queue = new workbox.backgroundSync.Queue('myQueue');

self.addEventListener('fetch', (event) => {
    // 只处理 POST 请求
    if (event.request.method !== 'POST') {
        return;
    }

    const syncReq = async () => {
        try {
            const res = await fetch(event.request.clone());
            return res;
        } catch (error) {
            // 请求失败时，添加进队列
            await queue.pushRequest({ request: event.request });
            return error;
        }
    }
    event.respondWith(syncReq());
});
```

```
workbox Background sync for tag 'workbox-background-sync:myQueue' has been received
workbox Request for '[redacted].json?[redacted]' has been replayed in queue 'myQueue'
workbox All requests in queue 'myQueue' have successfully replayed; the queue is now empty!
```

Tips

1. Chrome 控制台
   灵活调试

2. manifest.json
   display 模式

3. 缓存控制
   最小可用原则

4. backgroundSync
   调试勿使用 offline

THANKS

Q&A

2021.11.27