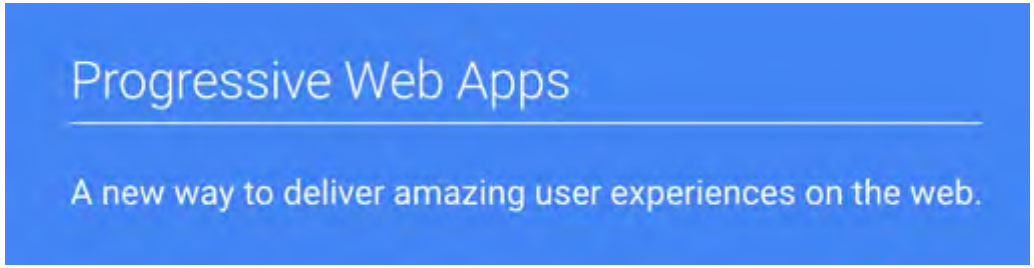
如何打造自己的PWA

钟恒
360奇舞团前端工程师

1. PWA 是什么
2. 如何打造我的 PWA
3. PWA 现状

PWA是什么？



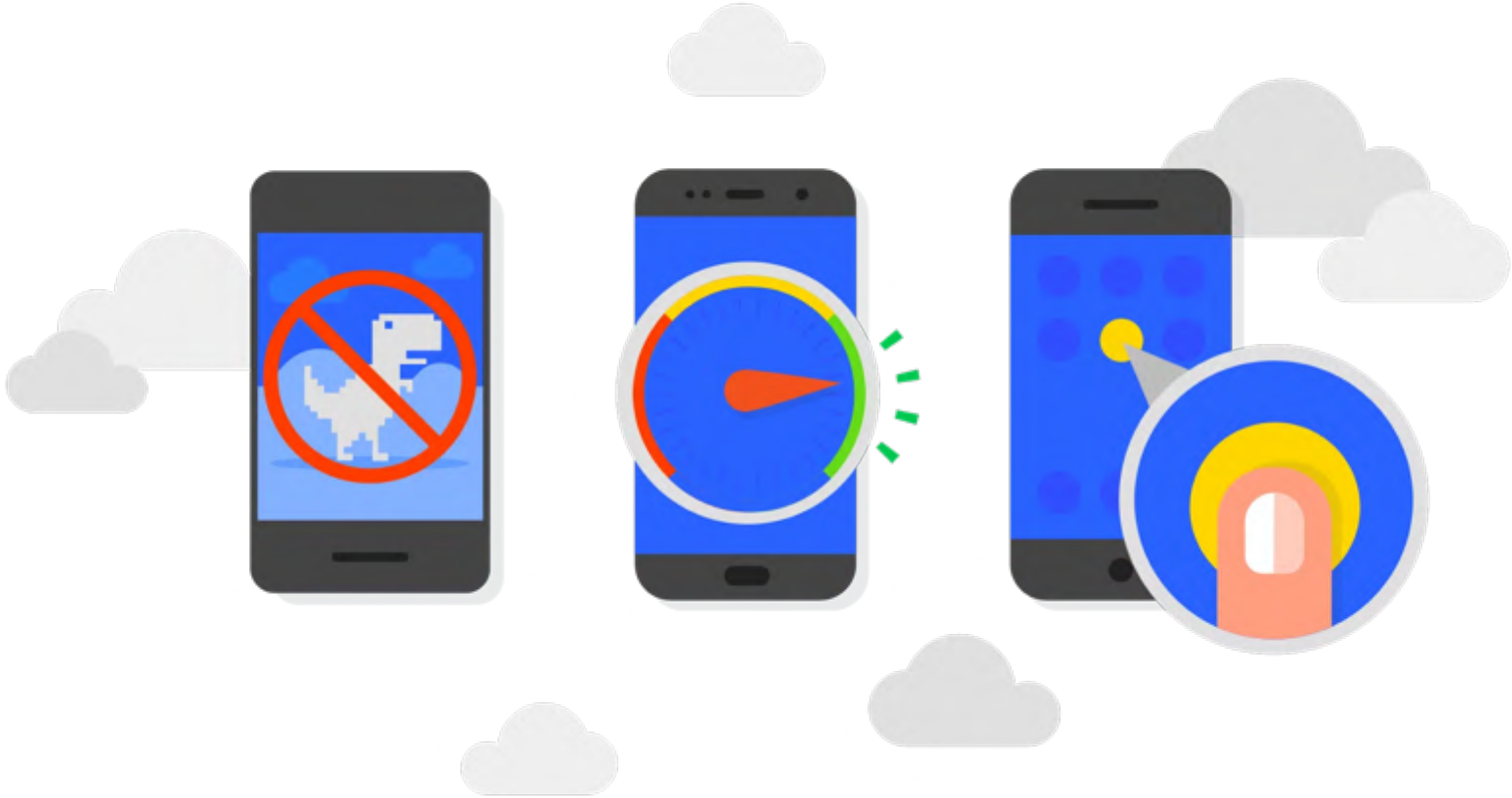
媲美原生应用的Web App

PWA三大特点

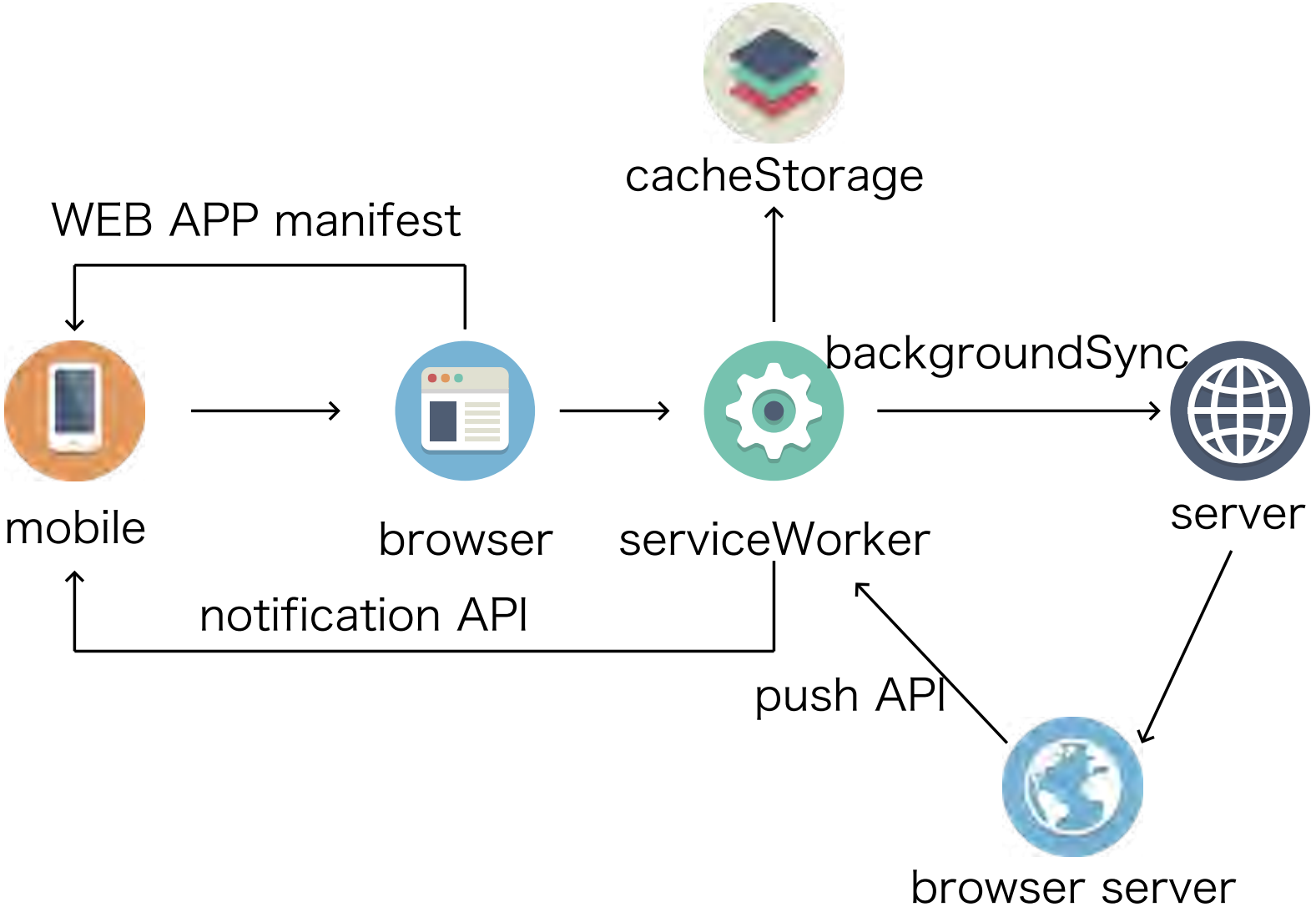
reliable
可靠

fast
快速

engaging
占领（随手可得）



PWA的技术栈


cacheStorage
WEB APP manifest
backgroundSync
mobile
browser
serviceWorker
server
notification API
push API
browser server

如何打造我的PWA

我想打造一个离线应用

serviceWorker VS appCache

	appCache	serviceWorker
编写方式	声明式文件	事件驱动式脚本
规范状况	Deprecated	Working Draft
离线使用	可以	可以
缓存加速	可以	可以
移除缓存	只允许用户清空	可以
跨域缓存	不可以	可以
错误容忍	单个文件错误会导致全量失效	可通过脚本处理文件错误
安全性	HTTP协议易于被劫持	HTTPS保证

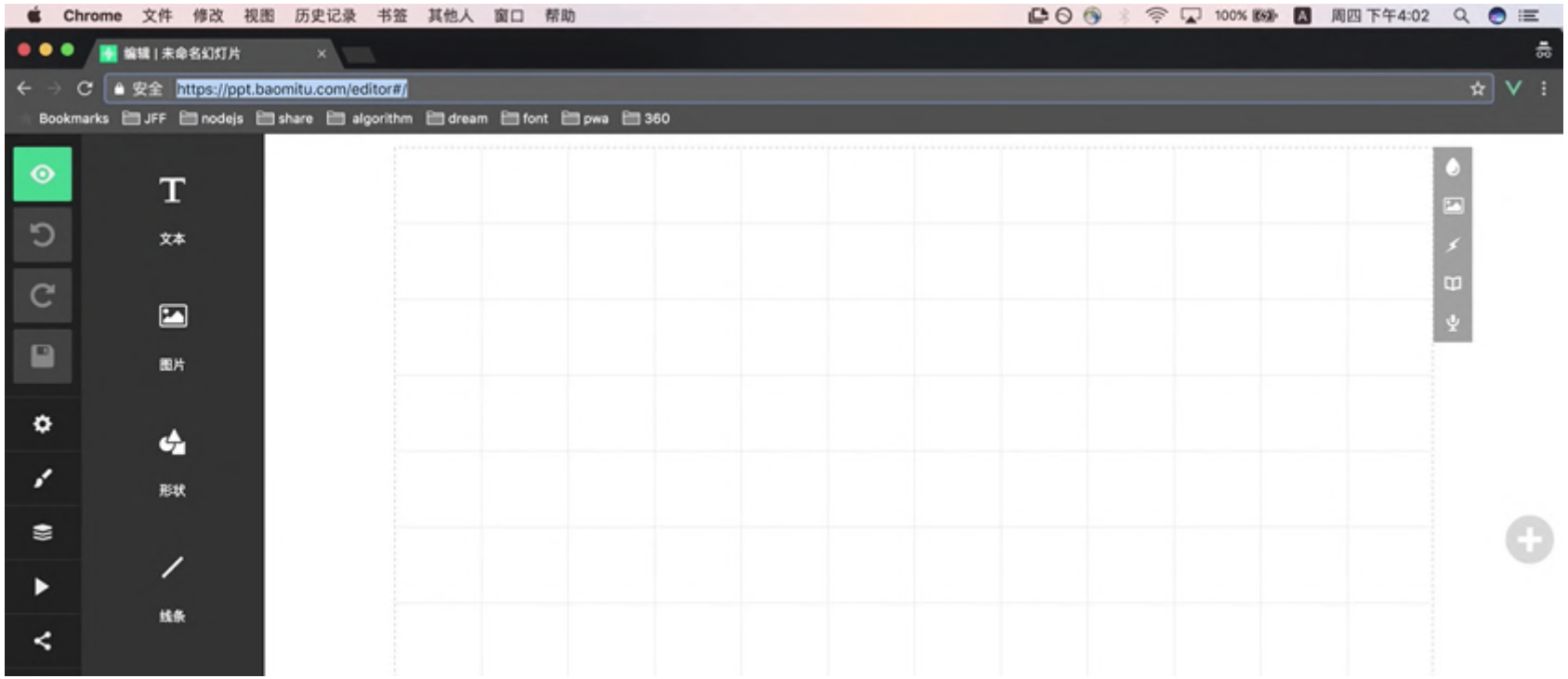
二十行代码实现离线运行

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
}
```

```
// 监听fetch
self.addEventListener('fetch', function (event) {
  const request = event.request
  // 尝试返回线上的数据
  event.respondWith(
    fetch(request)
    .then(function (response) {
      // 若成功，则缓存以备日后使用
      caches.open(cacheKeys[1])
      .then(function (cache) {
        cache.put(request, response)
      })
      // 正常返回请求内容
      return response
    })
    .catch(function () {
      // 若失败则反馈缓存中的部分
      return caches.match(request)
      .then(function (response) {
        return response || '默认缓存'
      })
    })
  )
}
```

1. 注册serviceWorker
2. 拦截请求
3. 有网时缓存响应
4. 无网时返回缓存

离线展示


Chrome 文件 修改 视图 历史记录 书签 其他人 窗口 帮助
100% 周四 下午4:02
编辑 | 未命名幻灯片
安全 https://ppt.baomitu.com/editor#/
Bookmarks JFF nodejs share algorithm dream font pwa 360
文本
图片
形状
线条

代码优化

1. 过滤请求
2. 只缓存成功响应

```
self.addEventListener('fetch', function (event) {
  const request = event.request
  const {method, url, headers} = request
  // 判断何种请求需要处理
  if(method === 'GET' &&
    (headers.get('Accept').indexOf('text/html') !== -1 ||
    url.match(/(css|js)$/) ||
    url.match(customize))
  ) {
    const noCors = !request.url.match(location.origin)
    // 尝试返回线上的数据
    event.respondWith(
    // 跨域需要采用不同的处理方式
    (noCors
      ? fetch(request.url, {'no-cors': true})
      : fetch(request)
    ).then(function (response) {
      // 将请求后的数据缓存
      const copy = response.clone()
      if(response.ok) {
        // 放入单独缓存
        caches.open(__VERSION__)
        .then(function (cache) {
          cache.put(request, copy)
        })
      }
      return response
    }).catch(function (err) {
```

我离线操作后的数据怎么上传呢?

1. 使用 online/offline 事件

```
window.addEventListener('online', e => uploadMyData())
```

用户把我的网页关掉了怎么办?

2. 每次打开页面的时候上传最新数据

用户不打开我的页面怎么办?

3. 尝试使用可靠的传输方式——backgroundSync

a method that enables web applications to synchronize data in the background.
If the page (or worker) that registered the event is running, the user agent will fire the sync event as soon as network connectivity is available.

简单的用法介绍

主线程或worker内部通过registration实例注册事件

```
navigator.serviceWorker.ready.then(function(registration) {
  registration.sync.register('outbox').then(function() {
    // registration succeeded
  }, function() {
    // registration failed
  });
});
```

serviceWorker在有网的时候执行

```
self.addEventListener('sync', function(event) {
  if (event.tag == 'outbox') {
    event.waitUntil(sendEverythingInTheOutbox());
  }
});
```

我期望可以这么用

```
// 无需等待返回时
save (slide_id, slide_config, notice) {
  ......
  if(navigator.onLine) {
    // 在线情况下直接上传云端
    return sync.POST(path, info)
  } else if(swHelper.backgroundSyncAble() && notice) {
    // 离线情况下注册backgroundSync事件进行处理
    swHelper.POST({path, param: info, id: path + slide_id, notice})
  }
  // 为了防止阻塞，我们直接返回离线错误
  return Promise.reject(new Error('offline'))
}
// 明确需要等待返回时
ask () {
  ......
  return swHelper.backgroundSyncAble()
    ? swHelper.POST({path, param: info, id: path + slide_id, notice})
    : return sync.POST(path, info)
}
```

使用postMessage辅助实现

1. 给主线程封装使用方法

```
sync (url, options, id = uuid(), notice) {
  return this.backgroundSyncAble()
    ? this.sendMessageToSw({type: 'sync', url, options, id, notice})
    : Promise.reject('no-backgroundSync')
}
POST ({path, param = {}, option = {}, id, notice} = {}) {
  const options = Object.assign({
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    credentials: 'include',
    body: isString(param) ? param : JSON.stringify(param)
  }, option)
  return this.sync(path, options, id, notice)
}
```

2. 封装双向沟通渠道

```
sendMessageToSw (msg) {
  return new Promise((resolve, reject) => {
    // 创建一个通道
    const msg_chan = new MessageChannel()

    // 处理返回的消息
    msg_chan.port1.onmessage = event => {
      if(event.data.error) {
        reject(event.data.error)
      } else {
        resolve(event.data)
      }
    }

    navigator.serviceWorker.controller.postMessage(msg, [msg_chan.port2])
  })
}
```

3. serviceWorker接收消息并注册事件

```
self.addEventListener('message', event => {
  if(isObject(event.data)) {
    if(event.data.type === 'sync') {
      // 动态分配一个tag
      const id = event.data.id || uuid()
      // 将配置临时储存
      syncStore[id] = Object.assign({port: event.ports[0]}, event.data)
      // 注册事件
      self.registration.sync.register(id)
    }
  }
})
```

4. 事件触发时发送请求并返回结果

```
self.addEventListener('sync', event => {
  // 获取信息
  const {url, options, port, notice} = syncStore[event.tag] || {}
  delete syncStore[event.tag]
  event.waitUntil(
    // 发送请求
    fetch(url, options)
    .then(response => {
      const copy = response.clone()
      // 对响应作相关处理后返回
      if(response.ok) {
        const contentType = response.headers.get('Content-Type')
        const parse = contentType.match(/json/)
          ? 'json'
          : contentType.match(/image/)
            ? 'blob'
            : 'text'
        copy[parse]()
        .then(data => {
          port.postMessage(data)
          // 如果有必要的话，我们还可以发送通知唤回用户
          if(notice) {
            const {title} = notice
            self.registration.showNotification(title, notice)
          }
        })
```

离线数据上传


Chrome 文件 修改 视图 历史记录 书签 其他人 窗口 帮助
backgroundsync - Google 搜索
编辑 | 如何打造自己的PWA应用
安全 https://ppt.baomitu.com/editor/320f21bd
应用 Bookmarks JFF nodejs share algorithm dream font pwa 360
文本
图片
形状
线条
如何打造自己的PWA应用

两步让你的Web App变得**靠谱**

1. 30行代码解决离线展示问题
2. 使用backgroundSync确保上传成功

备用方案

1. appCache
2. online/检测上传

读取缓存耗时 << 网络请求耗时

二十行代码实现提速

```
self.addEventListener('fetch', function (event) {
  const request = event.request
  // 判断是否是可以优先读取缓存的请求
  if(!remoteFirst(request)) return
  // 尝试返回缓存数据
  event.respondWith(
    caches.match(request)
    .then(function (cacheResponse) {
    // 若无缓存则请求线上数据
    return cacheResponse || fetch(request)
    .then(function (response) {
      const copy = response.clone()
      if(!response.ok) return copy
      // 若成功，则缓存以备日后使用
      caches.open(cacheKeys[1])
      .then(function (cache) {
        cache.put(request, copy)
      })
      return copy
    })
  )
}
```

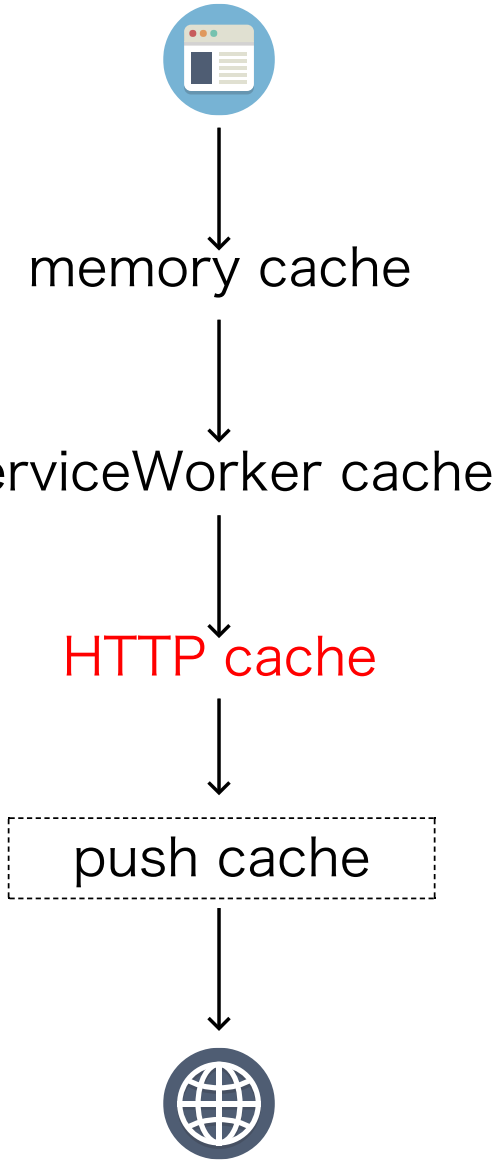
注意清理过期缓存

```
// 激活后清除无用缓存
self.addEventListener('activate', function (event) {
  caches.keys().then(keys => Promise.all(
    keys.map(key => {
      if (!expectedCaches.includes(key)) {
        return caches.delete(key);
      }
    })
  ))
  // 通知所有客户端更新
  self.clients.claim()
})
```

每当我们以为自己加了一个很炫酷的功能……

发现部分用户代码版本更新没有成功

多层次缓存的坑



切记要同时更新多个缓存

使用 URL 统一更新会是比较保险的做法

为何总是第二次使用才加速……

我们需要预存加速！！！

javaScript实现预存

常用方法

1. 使用现有的元素（如 img、script、link ）获取。
2. 使用 XMLHttpRequest 获取

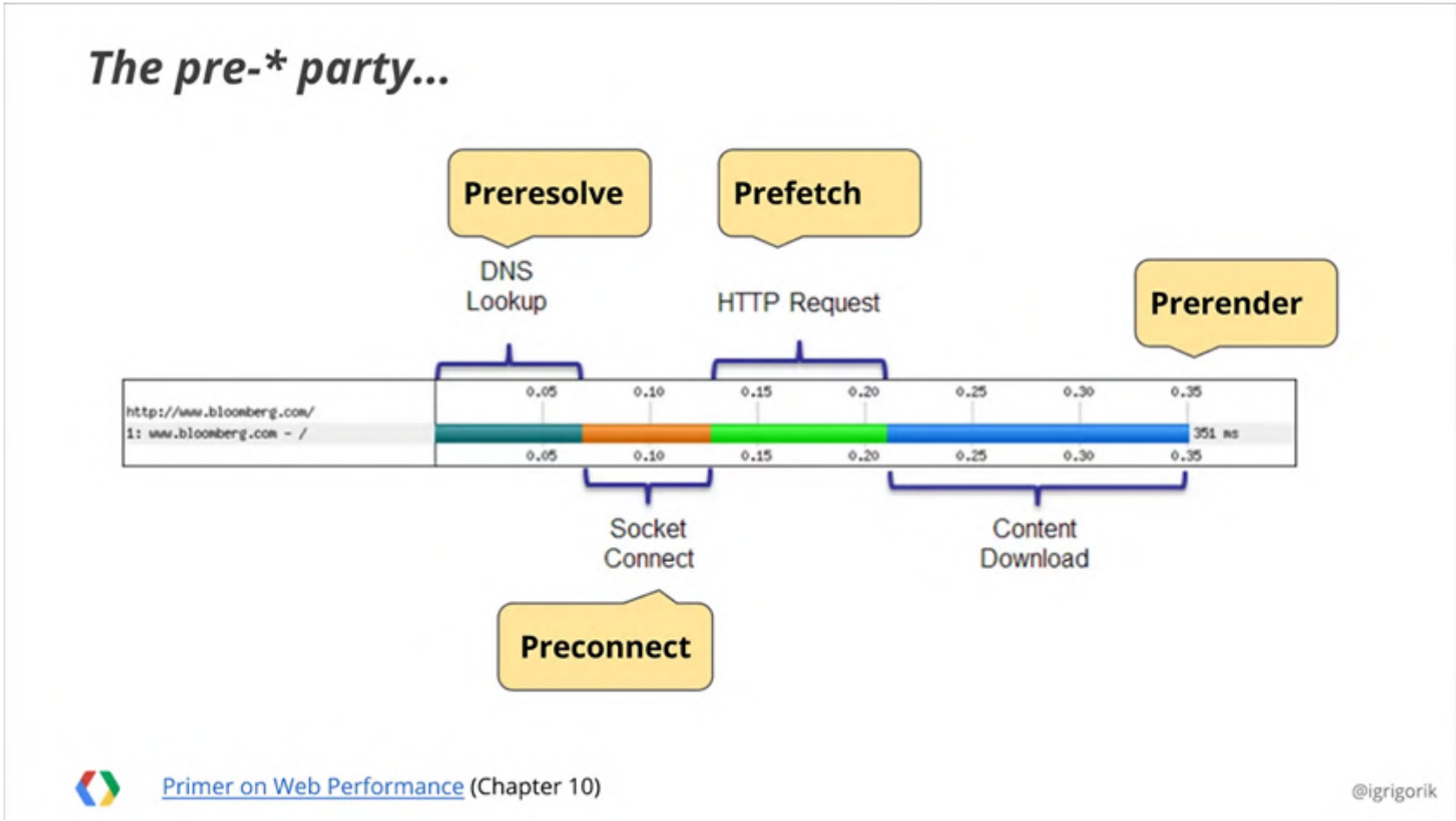
优点

1. 兼容性强
2. 可以确保缓存加载成功

潜藏问题

1. 占用主线程
2. 不能确保缓存的持久性

浏览器辅助预存


The pre-* party...
Preresolve
Prefetch
DNS Lookup
HTTP Request
Prerender
http://www.bloomberg.com/
1: www.bloomberg.com - /
0.05
0.10
0.15
0.20
0.25
0.30
0.35
351 ms
Socket Connect
Content Download
Preconnect
Primer on Web Performance (Chapter 10)
@igrigorik

<link rel="" href="//example.com">

dns-prefetch	DNS预加载
preconnect	DNS + TCP + TLS
prefetch	获取资源
prerender	后台渲染资源
preload	获取本次导航内部所需资源

优点	**潜藏问题**
1. 易用	1. 不确保加载完毕
2. 性能较好	2. 不能确保缓存的持久性

PWA 实现预存

在 serviceWorker 内部调用 cacheStorage 存储即可

在安装时进行类似声明式的安装

```
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);
      })
  );
});
```

PWA 实现预存

监听消息动态拉取

```
self.addEventListener('message', event => {
  if(isObject(event.data)) {
    if(event.data.type === 'prefetch') {
      fetch(event.data.href)
      .then(response => {
        const copy = response.clone()
        if(response.ok) {
          caches.open(__VERSION__).then(cache => {
            cache.put(href, copy)
          })
        }
      })
    }
  }
})
```

PWA 实现预存

优点

1. 在 serviceWorker 线程进行
2. 能确保资源加载成功
3. 资源是持久性储存的

相较于以往的加速方案，PWA的加速方案更为可靠、灵活

存储空间

存储空间约50M，且与localStorage、indexedDB共用

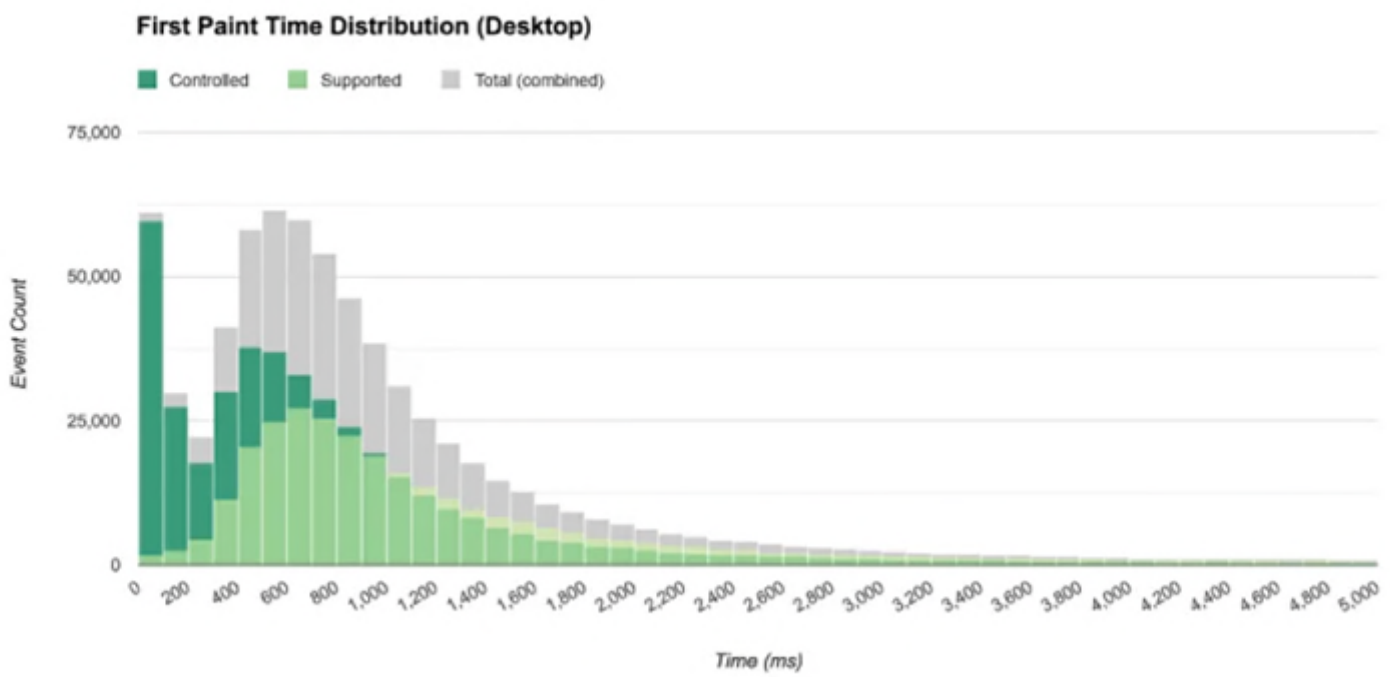
对没有存储的资源，我们也可以优化

以webp优化为例

```
self.addEventListener('fetch', function (event) {
  const request = event.request
  if (/\.jpg$|.png$/.test(request.url)) {
    // transfer the image from qhimg into webp format
    const supportWebp = event.request.headers.get('accept').includes('webp')
    if(supportWebp) {
      // Build the return URL
      const url = request.url.replace(/\.[^\.]*?$/, '.webp')
      const opt = Object.assign({}, request)
      const req = new Request(url, opt)
      return event.respondWith(
        fetch(req)
      )
    }
  }
})
```

优化效果如何?

用数据说话



官方数据显示：大量用户实现秒开，整体峰值前移

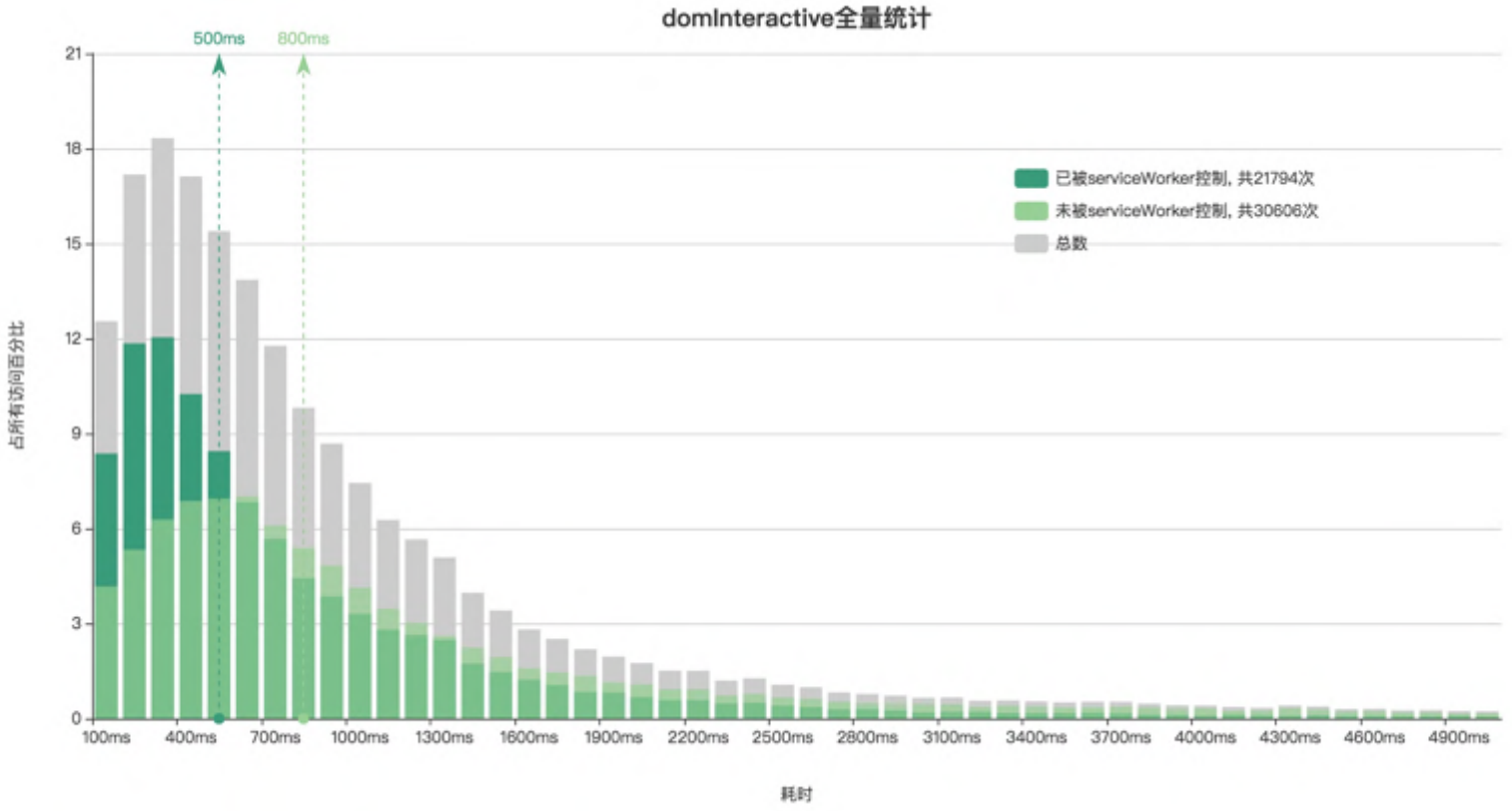
策略对比

官方策略[如SPA]

1. 缓存 html、js、css等所有必须的资源
2. 采用类似APP的方式进行更新

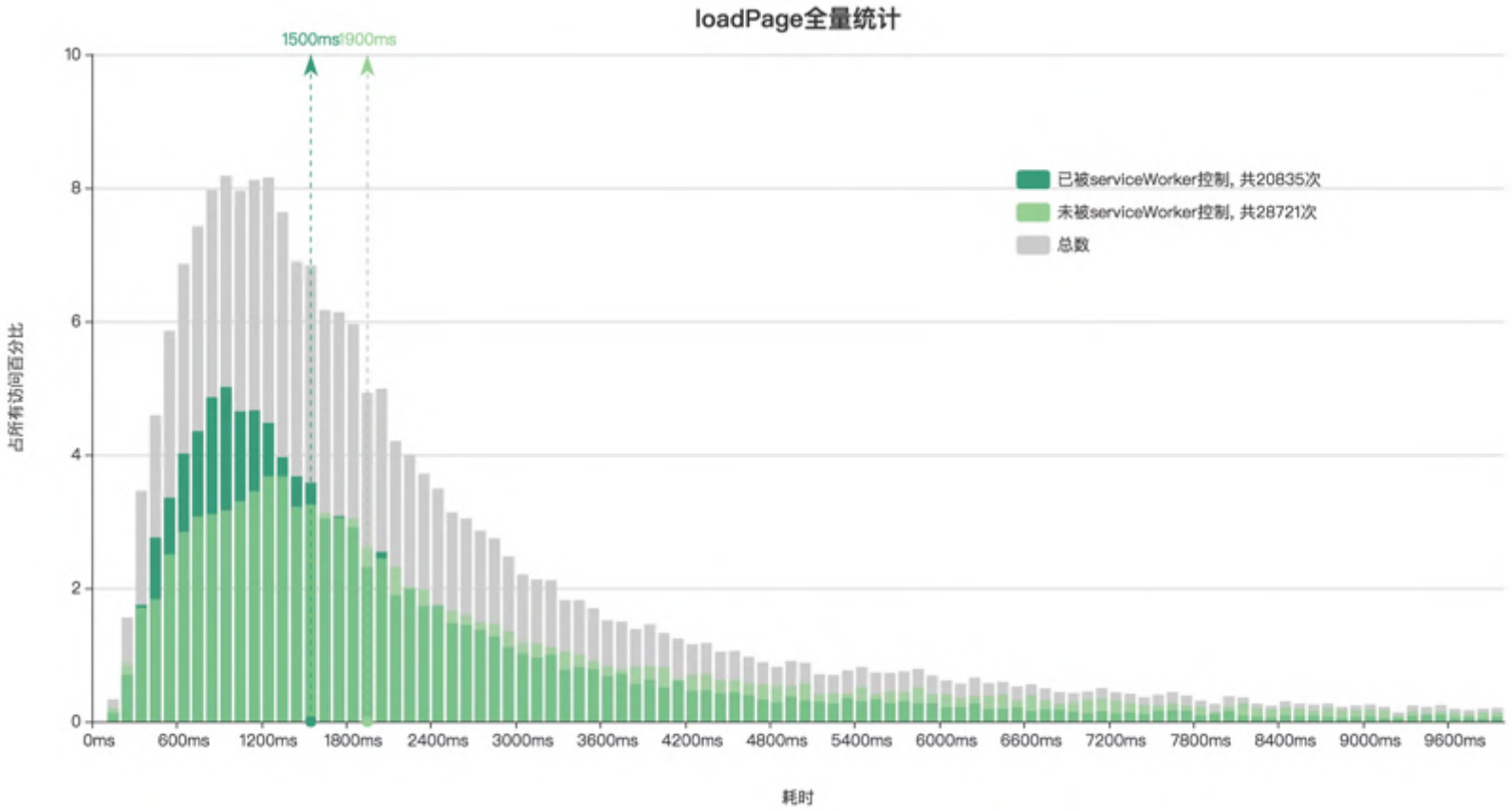
声享策略[保守]

1. 仅用 serviceWorker 缓存必要的 javaScript、CSS、font
2. 使用 webp 进行图片加速
3. 其余资源遵循HTTP cache 策略和 cdn 分发等常规策略
4. 采用 WEB 常用的即时更新方式



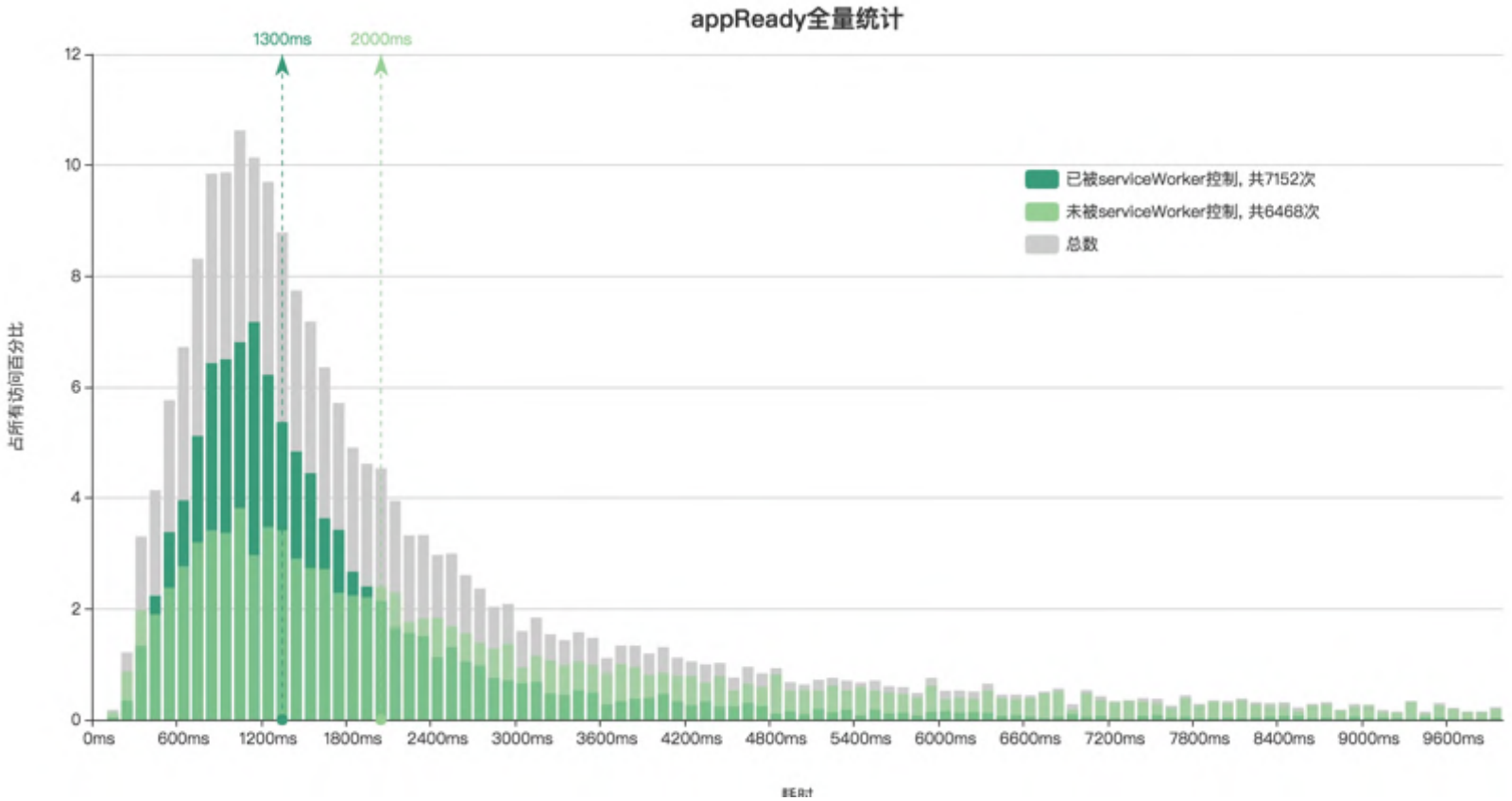
DOM可交互时间，近似于首屏时间

峰值前移明显且更为陡峭，中点前移37.5%



整体页面加载完成时间

峰值前移明显且更为陡峭，中点值前移21%



前端整体渲染结束时间（只有部分页面具有该统计）

峰值前移不明显但明显更为陡峭，中点前移35%

三步让你的Web App变得**快速**

1. 优先返回缓存内容
2. 尝试提前加载必要内容
3. 压缩其余资源

备用/同时使用方案

1. 传统的HTTP cache方案
2. javaScript驱动加载或Resource Hints驱动浏览器加载
3. picture、img的srcset等解决方案

需要一个更稳定、更可靠的打开方式——主屏幕入口

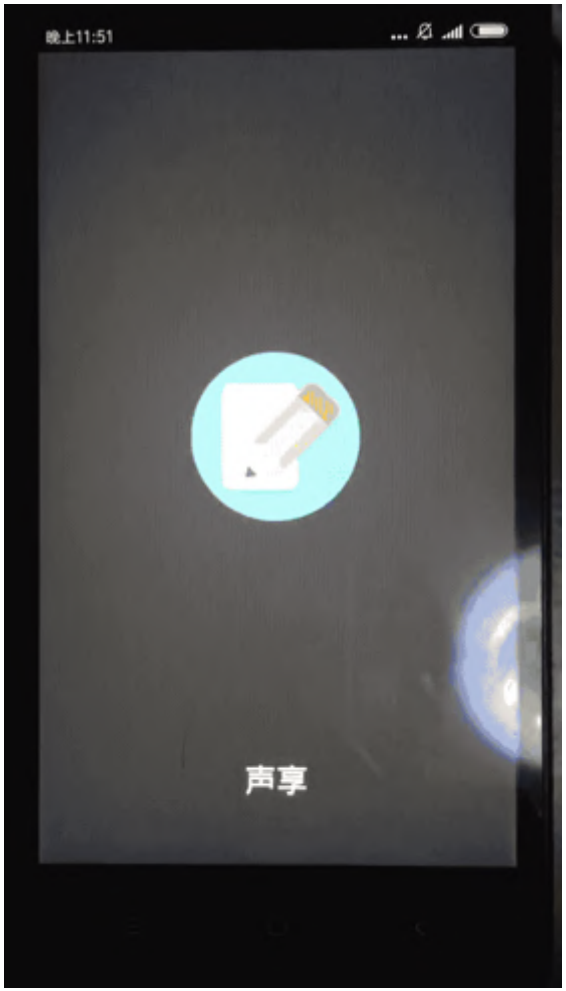
编辑 manifest 等个性化主屏配置

```
{
  "name": "声享",
  "short_name": "声享",
  "lang": "zh-CN",
  "theme_color": "#444444",
  "background_color": "#444444",
  "icons": [{
    "src": "https://p1.ssl.qhimg.com/t01a54d9106629dda0e.png"
    "sizes": "220x220",
    "type": "image/png"
  }],
  "start_url": "https://ppt.baomitu.com",
  "display": "standalone"
}
```

install banner 展示条件

- 一个manifest文件
- 通过HTTPS提供
- 注册了serviceWorker
- 被访问至少两次，且两次访问至少间隔五分钟。

Chrome



小米原生浏览器

可能大家更关注通知推送

前端订阅

```
// 订阅
subscribe () {
  const applicationServerKey = urlB64ToUint8Array(applicationServerPublicKey)
  const pushManager = this.registration.pushManager
  // 查询是否有订阅
  pushManager.getSubscription()
  .then(subscription => {
    if(subscription) return
    // 在浏览器上进行订阅
    pushManager.subscribe({
      userVisibleOnly: true,
      applicationServerKey
    })
    .then(subscription => {
      // 将订阅结果发回后端保存用于推送
      if(subscription) {
        const p256dh = btoa(String.fromCharCode.apply(null, new Uint8Array(subscription.getKey
        const auth = btoa(String.fromCharCode.apply(null, new Uint8Array(subscription.getKey('
        monitor.subscribe(subscription.endpoint, p256dh, auth)
      }
    })
  })
}
```

后端推送

```
async notificationAction () {
    const endpoints = await global.rediser.hgetall('sw-endpoints-test-1')
    const options = {
      TTL: 24 * 60 * 60,
      vapidDetails: {
        subject: '',
        publicKey: '',
        privateKey: ''
      },
      headers: {
        'Authorization': ''
      }
    }
    const tasks = []
    for(const index in endpoints) {
      const {endpoint, auth, p256dh} = JSON.parse(endpoints[index])
      const pushSubscription = {
        endpoint,
        keys: {
          auth,
          p256dh
        }
      }
      tasks.push(webpush.sendNotification(pushSubscription, this.post(), options))
    }
    return Promise.all(tasks).then(() => {this.success(endpoints)})
```

serviceWorker监听推送事件

```
self.addEventListener('push', function (event) {
  const options = event.data.json()
  event.waitUntil(self.registration.showNotification(options.title, options))
})
self.addEventListener('notificationclick', function (event) {
  event.notification.close()
  event.waitUntil(
    clients.openWindow('https://ppt.baomitu.com')
  )
})
```

Google

404. That's an error.

The requested URL /404notfound was not found on this server. That's all we know.

梦想很丰满，现实很骨感

Firefox work!!!

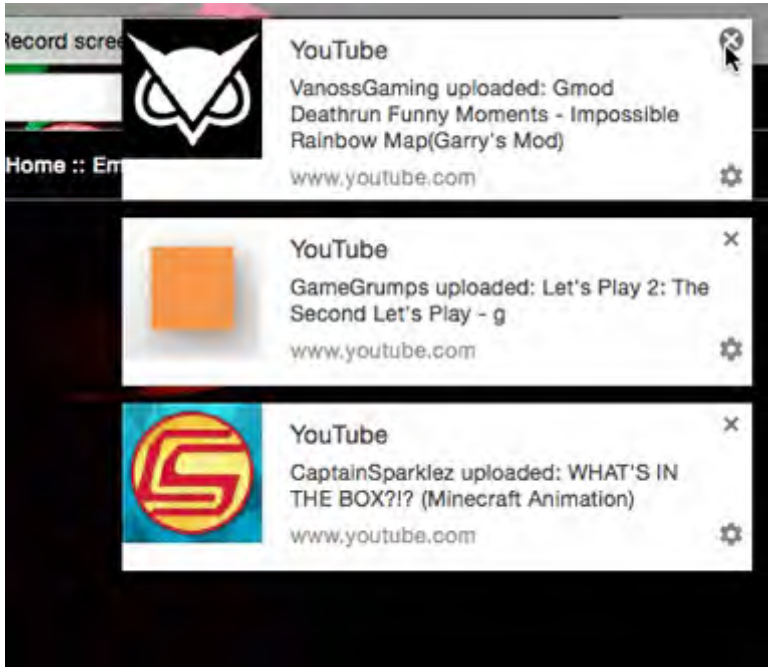
- push不可用有无替代方案?
- 除了“推”方案外，有没有更个性化的“拉”方案

periodicSync 或者 setTimeout

如何提高“用户粘性”

1. 编写 manifest 个性化主屏配置
2. 有条件的话尝试 push 或者 periodicsync

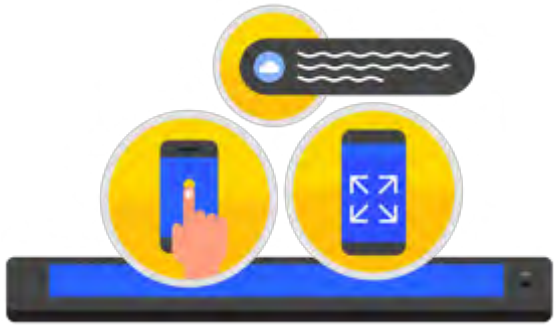
但是尽量克制点……



一款不错的PWA

1. 30行代码解决离线展示问题
2. 使用backgroundSync确保上传成功

1. 优先返回缓存内容
2. 尝试提前加载必要内容
3. 压缩其余资源



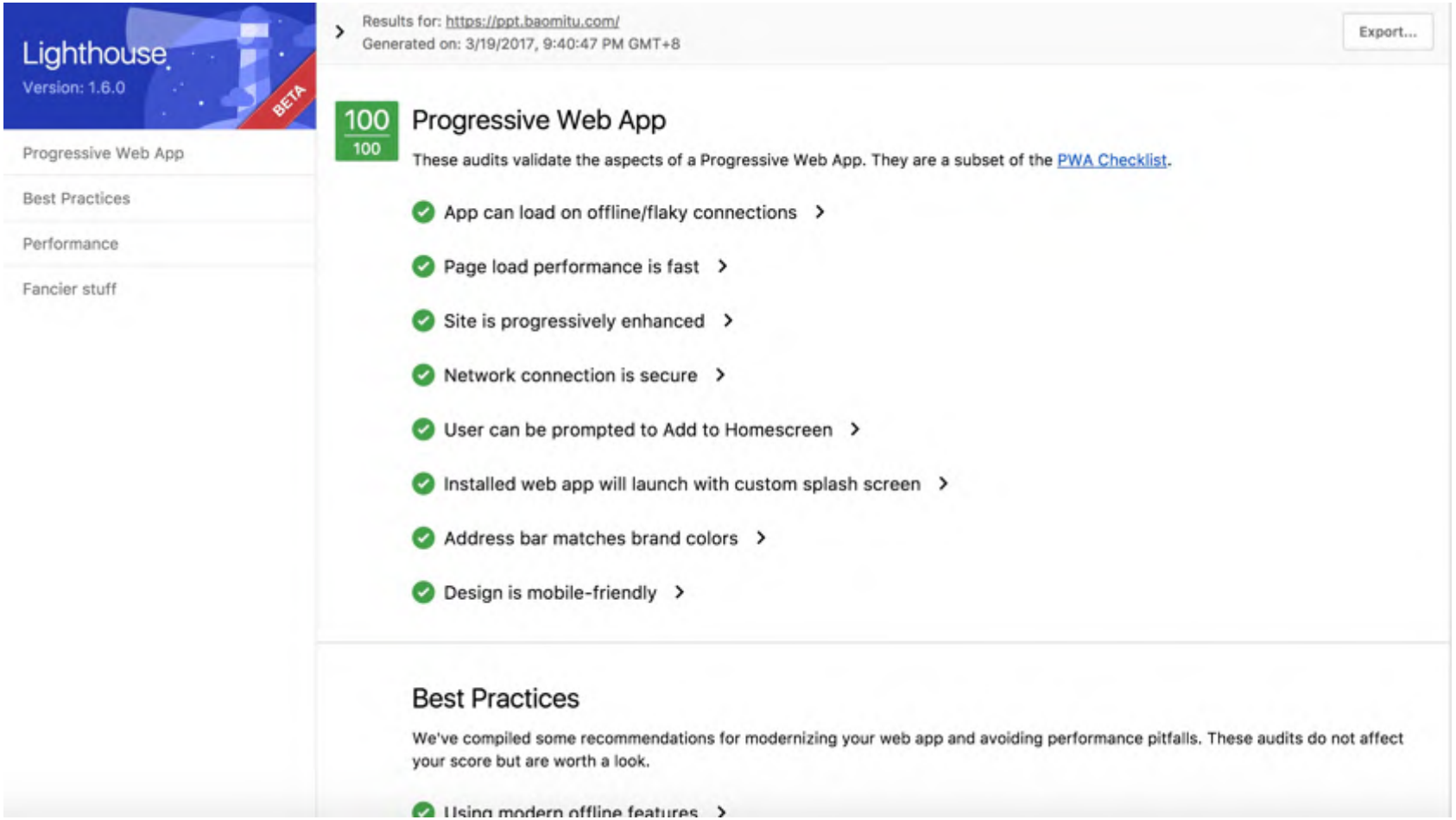
1. 编写 manifest 个性化主屏配置
2. 有条件的话尝试 push 或 periodicsync

谈谈 PWA 的现状

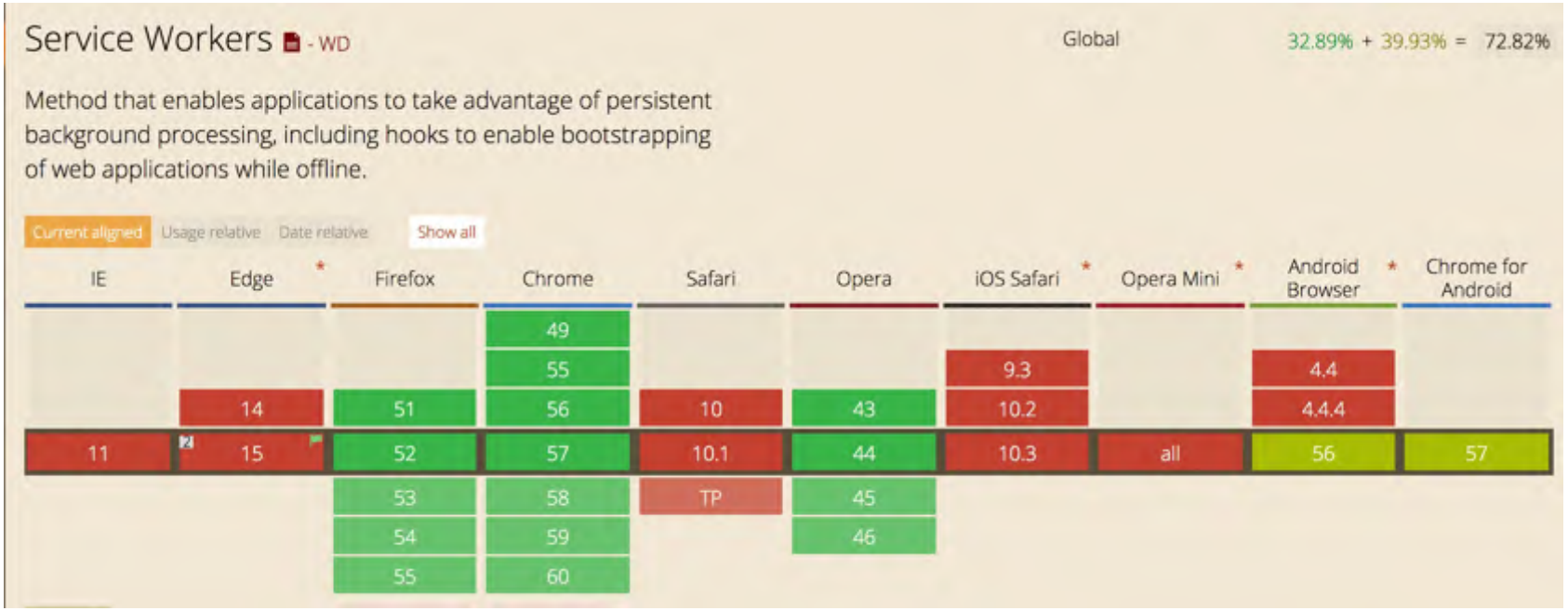
开发生态

- sw-toolbox
  - 方便易用的 serviceWorker 工具包
- sw-precache
  - 编译时帮你自动生成缓存方案
- fetch-sync
  - 使用 indexedDB 实现 backgroundSync 双向沟通的模块
- lighthouse
  - 官方提供的辅助性测试工具
- awesome-pwa
  - awesome!!!

Lighthouse 官方插件



兼容性问题


Service Workers - WD
Global
32.89% + 39.93% = 72.82%
Method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.
Current aligned
Usage relative
Date relative
Show all
IE
Edge *
Firefox
Chrome
Safari
Opera
iOS Safari *
Opera Mini *
Android Browser *
Chrome for Android
49
55
9.3
4.4
14
51
56
10
43
10.2
4.4.4
11
15
52
57
10.1
44
10.3
all
56
57
53
58
TP
45
54
59
46
55
60

- PC端
  - 以chrominum内核为主的浏览器大部分都支持了
    - chrome, 360, qq, opera
  - firefox支持
  - edge 15 可以手动打开
  - safari五年计划（已持续一年）
- 移动端
  - android 53
  - chrome, firefox
  - uc, qq, baidu最新版支持