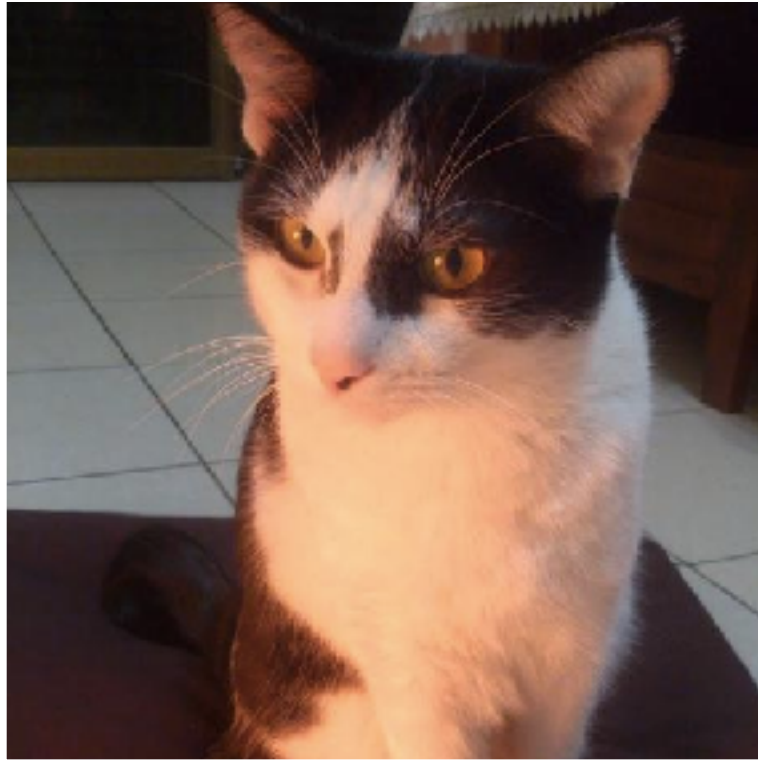


# Progressive Web Apps & Web Performance

# About Me



本名：吳俊賢

公司用：Henry Wu

網路用：CodinCat



**TREND**  
MICRO™

最難講的部分講完了

# Progressive Web Apps & Web Performance

# 先講 Progressive Web Apps

## (簡稱 PWA)

# Progressive Web Apps

懶人包

像 Native App 的 Web App

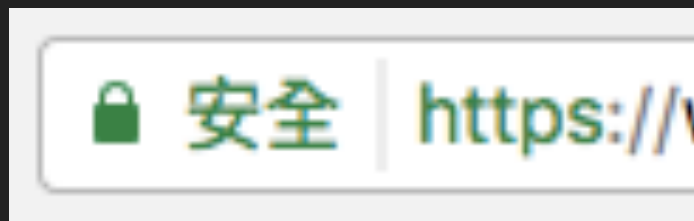
而且要很快

很快、很快

越快越好

# Progressive Web Apps

最低標準 - Alex Russel 版



Offline Support\*

Manifest

# Offline Support





# Progressive Web Apps

## 最低標準 - Google 官方版

HTTPS、Offline Support、Manifest  
Responsive

Chrome, Edge, Firefox and Safari

Nexus 5 + 3G, Time to Interactive < 10s

換頁要流暢，不應受網路影響

每個頁面都要有獨立的 URL

# Progressive Web Apps

先撇開 Performance 相關

HTTPS、RWD → 幾乎已是標配

獨立 URL → React Router, Vue Router...

跨瀏覽器 → 沒 IE，小事

只差加個 Manifest

# Manifest (.json)

為了把 Web App 偽裝成 Native App

我們需要一些設定

如 Icon 等



92% 上午12:59

<https://mobile.twitter.com>

1



# Welcome to Twitter

See what's happening in the  
world right now.



Twitter

twitter.com



加到主畫面





87% 上午1:06



1:06 上午

12°C 新北市 - 2月12日, 週日



LINE



React HN



Twitter



Facebook



Chrome



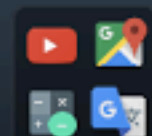
Wi-Fi



行動數據



Messenger



```
{
  "name": "Hello World",
  "short_name": "Hi",
  "icons": [
    {
      "src": "xxx.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "xxxxxx.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#000000",
  "theme_color": "#FFDDEE"
}
```

搞定

其實真的沒什麼

剩下的就是你 App 的功能



主要是 Web 功能越來越強  
以前 Native App 才能做的事  
現在 Web App 都能做到

# 例如

- Web Bluetooth
- Payment Request API
- Background Sync
- Push Notifications
- IndexedDB

...

結束  
謝謝大家

開玩笑的

別忘了剛才略過一樣東西

# Performance

(含 Offline Support)

**講到 Performance...**

Cache

# HTTP Caching



# 長這樣

Status	Protocol	Size	Time	Cache-Control
200	h2	(from disk cache)	17 ms	public,max-age=31536000
200	h2	(from disk cache)	9 ms	public,max-age=31536000
200	h2	(from disk cache)	9 ms	public,max-age=31536000
200	h2	(from disk cache)	9 ms	public,max-age=31536000
200	h2	(from disk cache)	9 ms	public,max-age=31536000
200	h2	(from disk cache)	7 ms	public,max-age=31536000
200	h2	(from disk cache)	7 ms	public,max-age=31536000
200	h2	(from disk cache)	6 ms	public,max-age=31536000
200	h2	(from disk cache)	7 ms	public,max-age=31536000

在快取期限內  
不會發出 request  
且 Status Code 為 200

# 那 304 (Not Modified)

## 是什麼？

Status	Protocol	Size	Time	Cache-Control
304	h2	558 B	57 ms	max-age=5
304	h2	558 B	57 ms	max-age=5
304	h2	558 B	58 ms	max-age=5
304	h2	557 B	60 ms	max-age=5

快取已經過期了

但 Browser 有存 ETag 等資訊

帶去 Server 比對後發現無需更新

這沒什麼

就是 **Server** 的設定

在 PWA 裡更強調的是  
**Service Worker**


# Service Worker

可以攔截所有 request

並傳回任意 response

```
self.addListener( 'fetch', event => {  
    event.respondWith( new Response( 'Hi' ) )  
})
```



Name	✕	Headers	Preview	Response
 jquery.min.js	1	Hi		

整個掉包的過程  
網頁端是察覺不到的

所以再說一次  
**HTTPS 是必要條件**

攔下 request 之後  
就能利用 Cache API 來做事  
一般會有幾種快取策略

**Cache Only**

**Cache First**

**Network Only**

**Network First**

**Fastest**

# 自己寫最基本的 Service Worker

## 大概像這樣

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.open('v1').then(cache =>
      cache.match(event.request).then(response =>
        response || fetch(event.request).then(response => {
          cache.put(event.request, response.clone())
          return response
        })
      )
    )
  )
})
```

我也看不下去  
別說什麼 Strategy 了

別擔心

Google 都幫我們搞定了



**sw-toolbox**

**sw-precache**

**Workbox** (最新，二合一)









可以透過 cli  
或是整合 webpack 等  
自動幫你生出來

```
module.exports = {  
  // ...  
  plugins: [  
    // ...  
    new workboxPlugin({  
      globDirectory: DIST_DIR,  
      globPatterns: ['**/*.{html,js,css}'],  
      swDest: path.join(DIST_DIR, 'sw.js')  
    })  
  ]  
}
```







如果想要 100% 掌控  
可以 import 進來自己寫

```
importScripts( 'workbox-sw.js' )

const workboxSW = new WorkboxSW( )
workboxSW.router.registerRoute(
  /\//images\/(.*\/)?.*\.(png|jpg|gif)/,
  workboxSW.strategies.cacheFirst( )
)
```

Status	Type	Size	Time	Waterfall
200	stylesheet	(from ServiceWorker)	22 ms	
200	stylesheet	(from ServiceWorker)	15 ms	
200	stylesheet	(from ServiceWorker)	20 ms	
200	stylesheet	(from ServiceWorker)	33 ms	
200	stylesheet	(from ServiceWorker)	32 ms	
200	stylesheet	(from ServiceWorker)	38 ms	
200	font	(from ServiceWorker)	46 ms	
200	font	(from ServiceWorker)	47 ms	
200	font	(from ServiceWorker)	85 ms	
200	script	(from ServiceWorker)	80 ms	

除非是用 Cache Only  
你會發現多了一堆 request  
且前面有個齒輪

Name	Status	Type	Size	Time
<input type="checkbox"/>  style.min.css	304	text/css	244 B	10 ms
<input type="checkbox"/>  glyphs.css	304	text/css	242 B	6 ms
<input type="checkbox"/>  main.css	304	text/css	243 B	9 ms
<input type="checkbox"/>  796b62e0-5149...	304	x-font-ttf	244 B	5 ms
<input type="checkbox"/>  glyphs-half...	304	font-woff2	243 B	2 ms
<input type="checkbox"/>  fontawesome-...	304	font-woff2	244 B	11 ms



正常現象請勿驚慌

齒輪代表是 **Service Worker**  
所發出的 **request**，  
用來更新快取

要檢視 **Service Worker** 和 **Cache**  
請切至 **DevTools** 的 **Application** 標籤

## Application

Manifest

Service Workers

Clear storage

## Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

## Cache

Cache Storage

workbox-precaching-revisio

workbox-precaching-revisio

workbox-runtime-caching-h

images - https://workboxjs.

Application Cache

## Service Workers

☐ Offline ☐ Update on reload ☐ Bypass for network ☐ Show all

<https://workboxjs.org/examples/workbox-sw/>

Source [sw.js](#)

Received 2017/7/11 下午3:53:29

Status ● #18 activated and is running [stop](#) [inspect](#)

Clients <https://workboxjs.org/examples/workbox-sw/> [focus](#)

<https://workboxjs.org/examples/workbox-sw/> [focus](#)

<https://workboxjs.org/>

Source [sw.js](#)

Received 2017/7/11 下午3:25:37

Status ● #2 activated and is running [stop](#) [inspect](#)

有了這麼先進的快取  
網站應該變得超級快了吧

...請別忘了

僅限於第 2、3、4... 次訪問

加速第一次訪問

最大的重點就是  
不要讓非必要的東西擋到我們



# Render-Blocking

```
<html>
```

```
<head>
```

```
  <title>Hello</title>
```

```
  <link href="style.css" rel="stylesheet">
```

```
</head>
```

```
<body>
```

```
  . . .
```

```
</body>
```

```
</html>
```



## ! 可能的最佳化做法

清除前幾行內容中的禁止轉譯 JavaScript 和 CSS

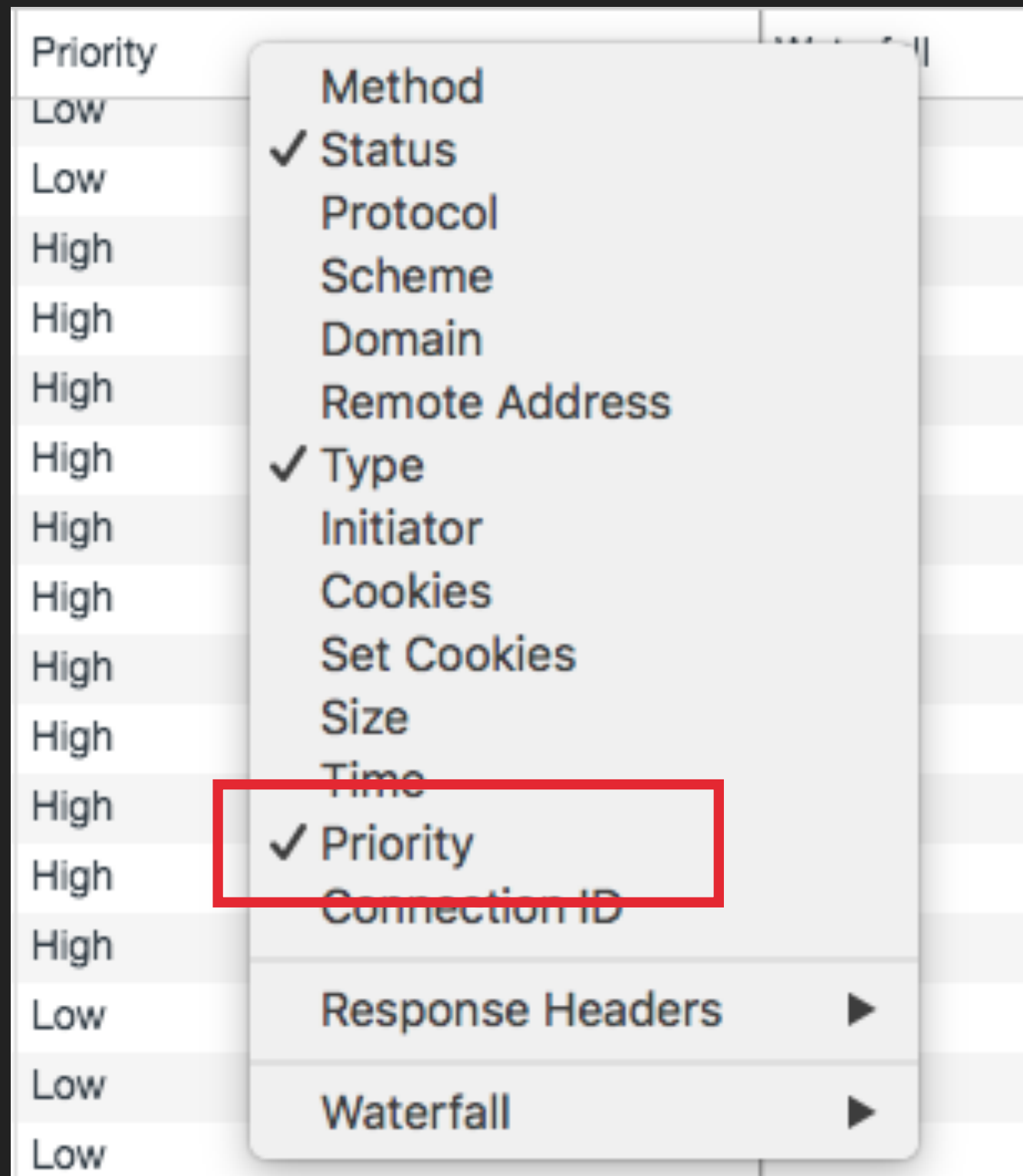
您的網頁含有 17 項禁止轉譯指令碼資源和 7 項禁止轉譯 CSS 資源，對網頁的轉譯作業造成延遲。

預設情況下

**CSS 的 Priority 是最高的**  
**甚至在 CSSOM 建置完成之前**  
**任何 JavaScript 都無法執行**

Name	Status	Type		Priority
 bootstrap.min.css	200	stylesheet		Highest
 font-awesome.min...	200	stylesheet		Highest

# 插播



# 插播

**Highest** - 例如 HTML、CSS

**High** - 例如 JS\*

**Medium** - 例如 Favicon

**Low** - 例如圖片

**Lowest** - 例如 prefetch 的資源

回到剛才的 CSS

有沒有辦法讓不重要的 CSS 別  
擋著我們？



```
<link
```

```
media="print"
```

```
href="print.css" rel="stylesheet">
```

Name	Status	Type	Priority
 print.css	200	stylesheet	Lowest

就算不合一樣會被下載  
但 **priority** 會降至最低  
且不再是 **Blocking**

# 小技巧

## 讓 IE 相關的 CSS 不要浪費大家的時間

```
<link
```

```
rel="stylesheet" href="css/ie.css"
```

```
media="(-ms-high-contrast: active), (-ms-high-contrast: none)">
```

廢話不多說，換 JS

JS 預設也是 **Blocking**

—遇到就要等它載完、執行完

類似剛才的手法

獨立的 `scripts`

可以加上 `async` 或 `defer`

```
<script async src="foo.js"></script>
```

```
<script defer src="bar.js"></script>
```

**async** 載完後就會馬上執行  
**defer** 則是等 DOM parse 完之  
後才會執行

**Priority 都會降至 Low**  
**且都不再是 Blocking**



但現實中很常見的狀況是  
一大包 JS、一大包 CSS

`async`、`defer` 根本用不到

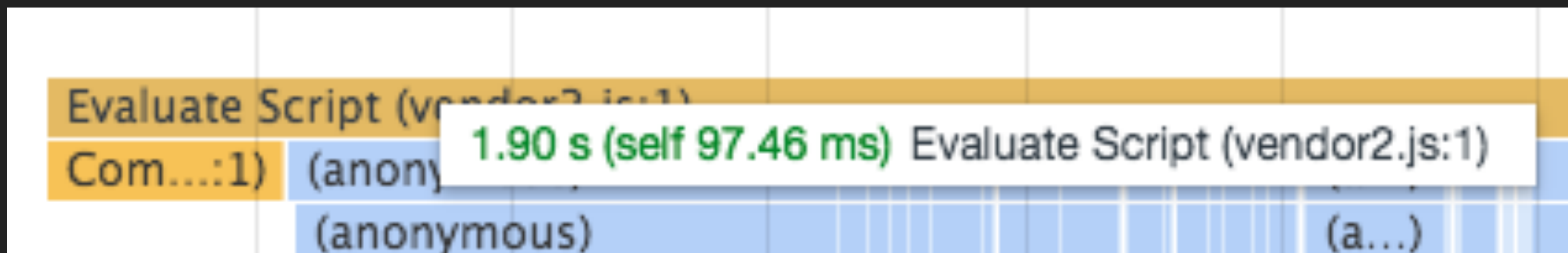
同時還要再補充一點

不要以為有快取就一定會超快

Status	Type	Initiator	Size	Time	Waterfall	1.00 s		
200	document	Other	(from ServiceWorker)	6 ms				
200	stylesheet	(index)	(from ServiceWorker)	23 ms				
200	stylesheet	(index)	(from ServiceWorker)	20 ms				
200	stylesheet	(index)	(from ServiceWorker)	22 ms				
200	stylesheet	(index)	(from ServiceWorker)	28 ms				
200	script	(index)	(from memory cache)	0 ms				
200	stylesheet	(index)	(from ServiceWorker)	23 ms				
200	stylesheet	(index)	(from ServiceWorker)	25 ms				
200	font	(index)	(from ServiceWorker)	32 ms				
200	font	(index)	(from ServiceWorker)	28 ms				
200	font	(index)	(from ServiceWorker)	31 ms				
200	script	(index)	(from ServiceWorker)	29 ms				
200	script	(index)	(from ServiceWorker)	31 ms				
200	script	(index)	(from ServiceWorker)	82 ms				
200	script	(index)	(from ServiceWorker)	37 ms				
200	script	(index)	(from ServiceWorker)	38 ms				

onLoad 事件

所有資源都有快取了  
且在 100ms 內就全部下載完成  
但卻在 1s 左右時才 load 完



要看懂、要執行 JS 都需要時間  
尤其在較低階的裝置上

再複習一次

剛才說過最大的重點就是  
不要讓非必要的東西擋到我們

# Code Splitting

# Code Splitting

把第三方 modules 或你自己的 JS

切成多個 .js 檔

且只在需要的時候才動態載進來

其實很簡單







```
import( 'fatLibrary' ).then(fatLibrary => {  
    console.log(fatLibrary)  
})
```

最早是 **require.ensure()**

後來改為 **System.import()**

最新規格只需 **import()**

Name ▲	Type	Size
 1.js	script	259 KB
 2.js	script	232 KB
 4.js	script	50.6 KB
 7.js	script	468 KB

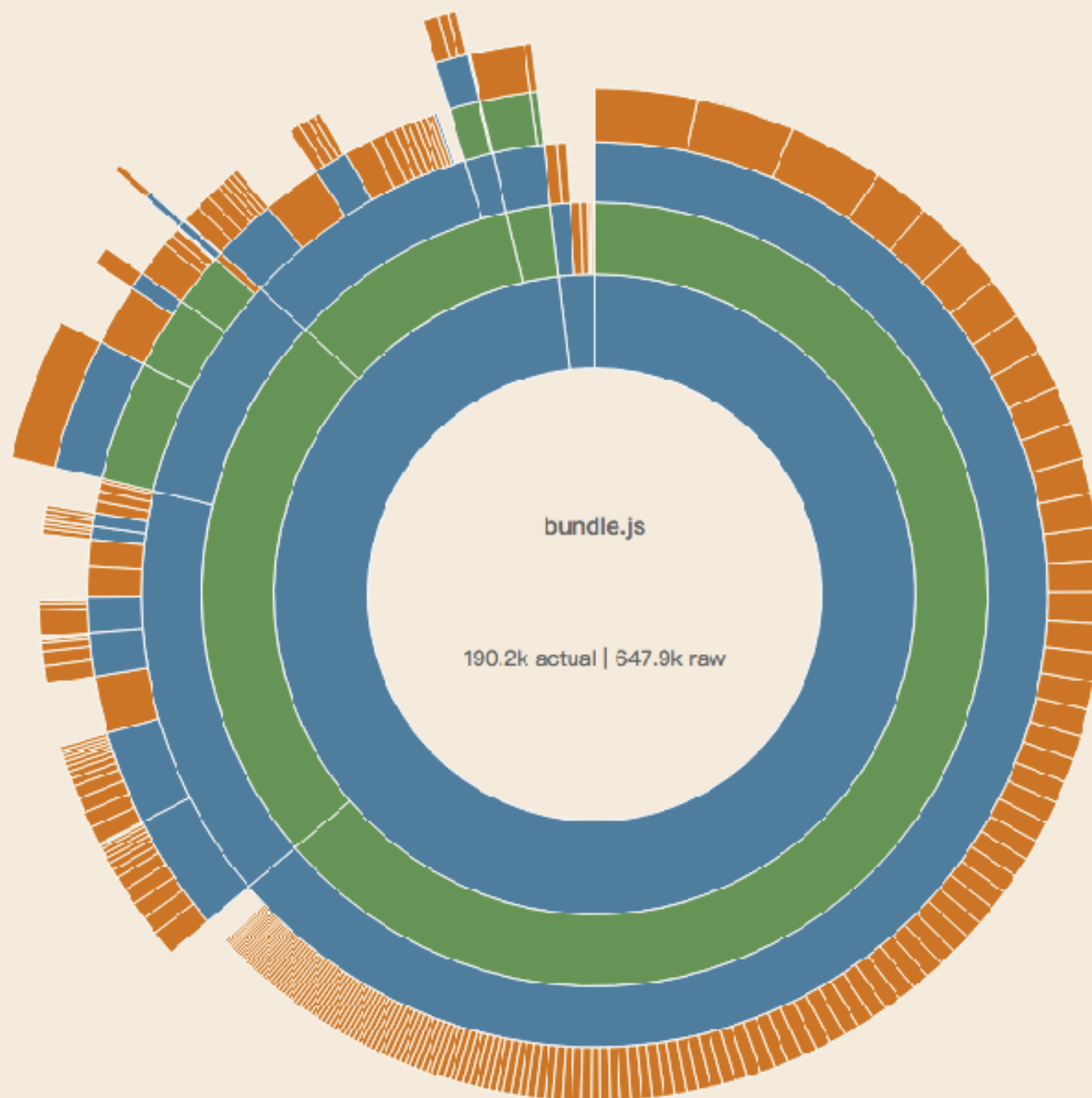
(預設檔名為流水號)

就這樣

有 **babel** 和 **webpack** 之後

不需再加任何額外設定

# WEBPACK VISUALIZER



**React Router 、Vue Router**  
**等都支援 route based 的**  
**code splitting**

**拿 Vue Router 來示範**

**(因為 React Router 4 之後比較囉嗦)**

```
const router = new VueRouter({  
  routes: [{  
    path: '/foo',  
    component: () => import('./Foo.vue')  
  }]  
})
```

**Foo.vue 還有它的 dependencies 只有在  
進入 /foo 頁面後才會被下載及載入**

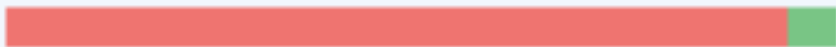
別忘了還有 CSS



案例

你載入了 bootstrap

但你用了多少？

	Type	Total Bytes	Unused Bytes	
css/bootstrap.min.css	CSS	121 200	115 657 95.4 %	

很少看到有網站

用到 30% 以上的 bootstrap

其實有自訂下載，用不到的東西可以砍掉

# 插播

⋮ Console Coverage x				
● ↺ ⌵				
URL	Type	Total Bytes	Unused By...	
https://v4-alpha.getbootstrap.com/dist/css/bootstrap.min.css	CSS	150 996	140 253 92.9 %	<div><div></div></div>
chrome-extension://jabopobgcpjmedljpbcaablpmlmfcogm/jquery.min.js	JS	94 871	66 962 70.6 %	<div><div></div></div>
https://code.jquery.com/jquery-3.1.1.slim.min.js	JS	69 309	40 299 58.1 %	<div><div></div></div>
https://v4-alpha.getbootstrap.com/dist/js/bootstrap.min.js	JS	46 653	28 677 61.5 %	<div><div></div></div>
chrome-extension://jabopobgcpjmedljpbcaablpmlm... /html2canvas.min.js	JS	28 598	27 387 95.8 %	<div><div></div></div>
https://v4-alpha.getbootstrap.com/assets/js/vendor/holder.min.js	JS	32 280	25 403 78.7 %	<div><div></div></div>
https://v4-alpha.getbootstrap.com/assets/js/vendor/tether.min.js	JS	24 632	20 997 85.2 %	<div><div></div></div>
chrome-extension://jabopobgcpjmedljpbcaablpmlmfcogm/wf_chrome.js	JS	17 209	15 127 87.9 %	<div><div></div></div>
chrome-extension://lmhkpmbekcpmknkioeibfkpmmfi... /content.bundle.js	JS	29 689	15 028 50.6 %	<div><div></div></div>
https://v4-alpha.getbootstrap.com/assets/css/docs.min.css	CSS	18 247	14 187 77.7 %	<div><div></div></div>
https://www.google-analytics.com/analytics.js	JS	30 071	9 638 32.1 %	<div><div></div></div>
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/inject.js	JS	10 233	8 239 80.5 %	<div><div></div></div>
extensions::event_bindings	JS	19 021	7 427 39.0 %	<div><div></div></div>
chrome-extension://fjnbnpbmkenffdngjfgmeleoegfcffe/src/inject/apply.js	JS	9 265	5 825 62.9 %	<div><div></div></div>
extensions::json_schema	JS	18 051	5 031 27.9 %	<div><div></div></div>
https://v4-alpha.getbootstrap.com/assets/js/vendor/clipboard.min.js	JS	10 285	4 792 46.6 %	<div><div></div></div>
extensions::sendRequest	JS	5 267	3 470 65.9 %	<div><div></div></div>
extensions::lastError	JS	5 125	2 838 55.4 %	<div><div></div></div>
extensions::binding	JS	22 306	2 717 12.2 %	<div><div></div></div>
extensions::messaging	JS	16 405	2 601 15.9 %	<div><div></div></div>

# 插播

**Chrome  $\geq 59$**

**(Chrome  $\geq 60$ )**

**cmd + shift + p : coverage**

現在，  
非初次載入時必要的東西  
我們都拆分開來了

但使用者是會操作的

例如換頁

這時又要再下載及載入這些資源

# Precache

就是字面上的意思  
雖然現在還沒用到  
但我先偷偷 `cache` 起來



**回到 Service Worker**

**回到 Workbox**

```
importScripts( 'workbox-sw.js' )
```

```
const workboxSW = new WorkboxSW( )
```

```
workboxSW.precache( [{  
  url: ' /dist/2.js',  
  revision: '4301192...d3c2cb'  
}] )
```

(這通常不會手動寫

**cli 或 webpack 可以幫你生出來)**

**再回到 PWA**

**有一項規則是 Offline Support**

最最最基本的方式就是  
把 `offline.html` 之類的東西  
`precache` 起來

然後在請求 `index.html` 失敗時

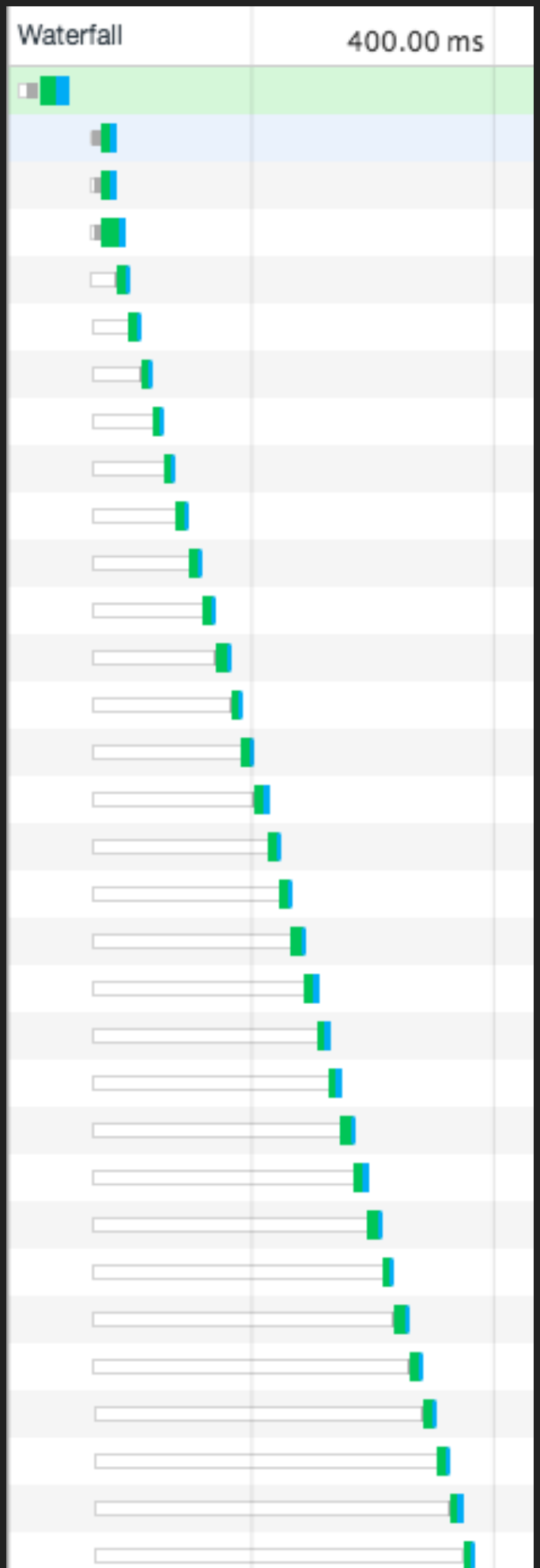
改回傳 `offline.html`

最低限度的 `Offline Support`

就搞定了

隨著資源越來越多  
有個 CP 值超高的東西必加

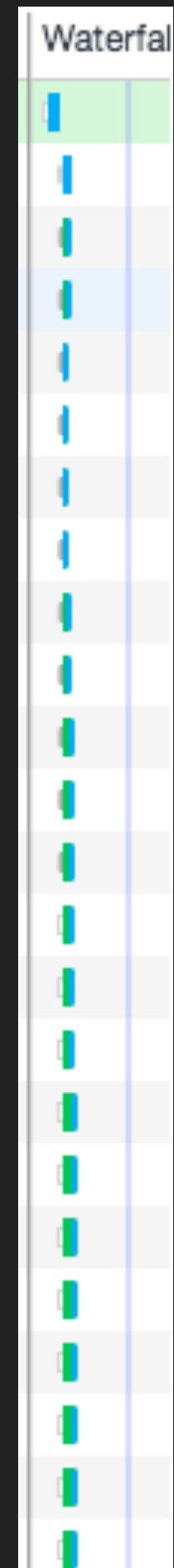
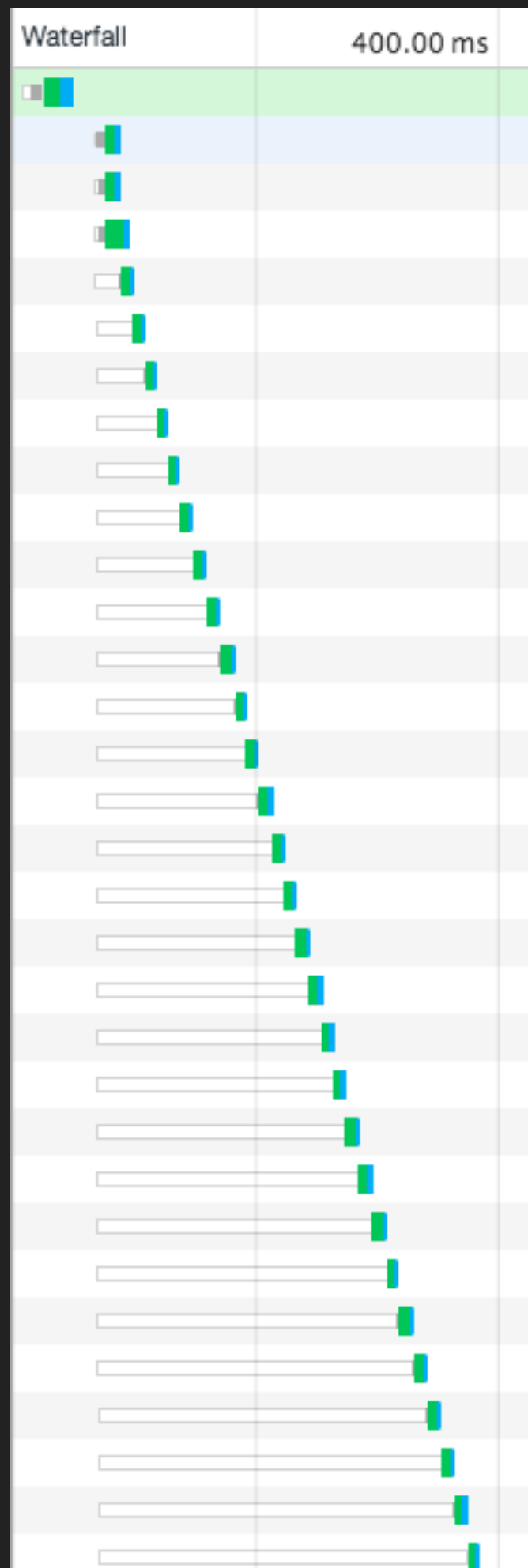
否則



**HTTP/2**



# 懶人包



現在初次載入很快  
而且還有快取了  
但別忘了還有...

# **Runtime Performance**

以 React 來說  
多數情況下不需刻意優化

**適時使用 PureComponent  
或者你有使用 React Redux**

**很少需要自己實作**

**shouldComponentUpdate**

當然某些時候還是需要優化

# 案例

若需要用到操作 Real DOM 的 Library  
例如 Highcharts、D3.js 或 jQuery 等  
我們通常會放在 `componentDidUpdate` 之  
類的地方，會在 React render 完後才執行



當我們 mount 一個 component 時：

**componentWillMount()**

**render()**

**Browser paints**

**componentDidMount()**

這樣對嗎？

假設在 cDM 裡還執行了 setState() :

**componentWillMount()**

**render()**

**Browser paints**

**componentDidMount() → setState()**

**shouldComponentUpdate()**

**componentWillUpdate()**

**render()**

**Browser paints**

**componentDidUpdate()**

# 都不對

**(React lifecycle 的順序是對的沒錯)**

執行 `render()` 不代表  
`browser` 會開始把東西畫上去

**mount 一個 component 時：**

**componentWillMount()**

**render()**

**componentDidMount()**

**Browser paints**

在 cDM 裡再執行 setState() :

**componentWillMount()**

**render()**

**componentDidMount() → setState()**

**shouldComponentUpdate()**

**componentWillUpdate()**

**render()**

**componentDidUpdate()**

**Browser paints**

別忘了還包括

底下所有子 components

問題在哪？



一般來說直接操作 Real DOM  
的都比較貴一點

**意思就是**

**若你在 cDM 或 cDU 之中**

**使用 D3.js 或 Highcharts 等做了許多事情  
從 mount 這個 component 到 browser 把  
它畫出來為止會花很多時間**

實際例如，用 **React Router** 切換頁面時  
使用者點擊連結，從 **A 頁** 切到 **B 頁**  
而 **B 頁** 裡面有許多這類的 **components**  
所有 **components** 必須 **mount** 完、執行完  
之後頁面才會出現  
延遲感可能非常明顯

# 在 cDM 裡再執行 setState() 的行為

# mount + render

# update + render

[illegible]

paint

各花個 0.1 秒，你的換頁就要 0.2 秒以上

最簡單暴力的解法

```
componentDidUpdate() {  
  setTimeout(() => {  
    this.expensiveFunc()  
  }, 0)  
}
```

**setTimeout 會在這輪  
執行完、畫完之後才觸發**

**解法2：Twitter 的解法**  
**用 Higher Order Component**  
**延遲載入 component**



```
import hoistStatics from 'hoist-non-react-statics'
import React from 'react'

export default function deferComponentRender(WrappedComponent) {
  class DeferredRenderWrapper extends React.Component {
    state = { shouldRender: false }

    componentDidMount() {
      window.requestAnimationFrame(() => {
        window.requestAnimationFrame(() =>
          this.setState({ shouldRender: true })
        )
      })
    }

    render() {
      return this.state.shouldRender
        ? <WrappedComponent {...this.props} />
        : null
    }
  }

  return hoistStatics(DeferredRenderWrapper, WrappedComponent)
}
```

```
const DeferredComponent =  
  deferComponentRender(HeavyComponent)
```

其實就是包裝起來而已  
用起來比較漂亮

然後這裡是用

**requestAnimationFrame**

這個狀況下兩者結果一樣

所以剛才的 `setTimeout` 等同

```
componentDidUpdate() {  
  requestAnimationFrame(() => {  
    requestAnimationFrame(this.expensiveFunc)  
  })  
}
```

總之就是延遲較慢的操作

儘快回饋 user

這概念不是只有 React 適用

# Summary

Cache

CSS Media Query

JS Async, Defer

Code Splitting

Precache

HTTP/2

延遲 React Components



還有很多沒時間提

**preload**

**prefetch**

**IndexedDB**

**HTTP/2 Server Push**

**SSR**

...

你不需要全部都做  
要動手一定是從 CP 值高的先

# 80/20 法則

大概做 20% 的優化就能達到

80% 的優化度

**快取**

**壓縮 (gzip, brotli)**

**HTTP/2**

**不用一個下午，成效又超高**

**React 或 Vue 之類的 library**  
**請確定是 production build**

應用越來越複雜

低階裝置有點吃力

特殊的 case

或你吃飽太閒

再繼續一步一步優化



**Alex Russell** @slightlylate · 4月22日



Need to dig into this and explain a few things:

1. Perf doesn't get good by magic. Never one thing, it's always many little things

*Perf doesn't get good by magic. Never one thing, it's always many little things*

**Thanks**



# Refs

- ▶ <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/>
- ▶ <https://developers.google.com/web/progressive-web-apps/checklist>
- ▶ <https://workboxjs.org/>
- ▶ <https://medium.com/reloading/preload-prefetch-and-priorities-in-chrome-776165961bbf>
- ▶ <https://medium.com/@paularmstrong/twitter-lite-and-high-performance-react-progressive-web-apps-at-scale-d28a00e780a3>