Service Worker - Caching, Offline, Native Features

개요

- PWA 의 오프라인 경험, 네이티브 기능의 구현 기반이 되는 서비스워커 학습
- 기존 워커들과 차별화되는 서비스워커의 특징 이해 및 배경 소개
- 서비스워커를 구현하기 위한 등록, 설치, 활성화, 업데이트 학습 및 실습
- 서비스워커 구현을 위한 라이프싸이클 이해 및 학습
- 서비스워커 보조 라이브러리 소개 및 사용법 학습

목차

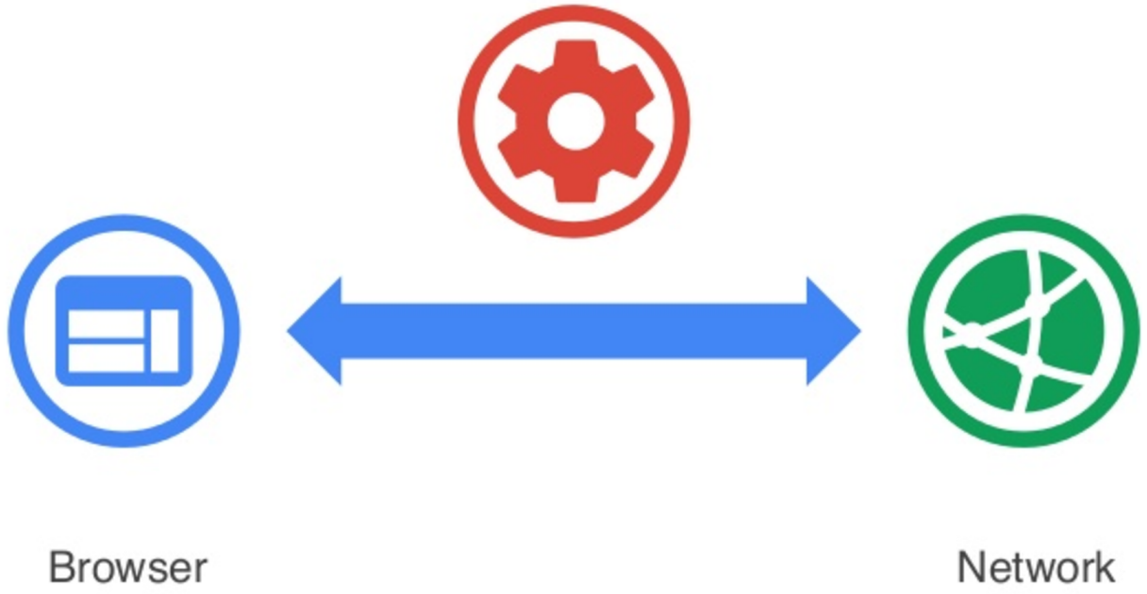
- 서비스 워커 활성화 및 업데이트
- 서비스 워커 라이프 싸이클
- 서비스 워커 캐싱
- 서비스 워커 디버깅
- 서비스 워커 실습 #4 - 과제
- 서비스 워커 푸시

Service Worker 시연

Offline Experience

Service Worker 소개

- **브라우저와 서버 사이의 미들웨어 역할을 하는 스크립트 파일**
- PWA 에서 가장 중요한 역할을 하고, Offline Experience 과 Mobile & Web Push 의 기반기술



Service Worker **특징**

1. **브라우저의 백그라운드에서 실행**되며 웹 페이지와 별개의 라이프 싸이클을 가짐
    - Javascript UI 쓰레드랑 별도로 동작하는 또 다른 쓰레드
2. **네트워크 요청을 가로챌 수 있어** 해당 자원에 대한 캐쉬 제공 또는 서버에 자원 요청
    - 프로그래밍 가능한 네트워크 프록시
3. **브라우저 종속적인 생명주기로 백그라운드 동기화 기능 제공**
    - Push 알람의 진입점을 제공
4. Web & Mobile Push 수신이 가능하도록 Notification 제공

5. navigator.serviceworker 로 접근

6. 기존 Javascript 와의 **별개의 자체 스코프**를 가짐

    - 크롬 개발자 도구의 Console 과의 별개의 서비스워커 전용 Console 존재

7. DOM 에 직접적으로 접근이 불가능 - postMessage() 이용

8. 사용하지 않을 때 **자체적으로 종료**, **필요시에 다시 동작** (event-driven 방식)

Service Worker **배경**

기존에 이미 존재하던 기술들을 **보완** -> **그리고 진화**

AppCache

- 오프라인 경험을 제공하기 위한 캐시 제공, HTML 표준
- 복수 페이지 앱에서 오동작, 파일 변화에 대해 둔감한 캐싱등의 문제

```
<html manifest="example.appcache">
</html>
```

```
// 서버에 추가 설정 필요 mime-type = text/cache-manifest
CACHE MANIFEST
# 2010-06-18:v3

# Explicitly cached entries
index.html
css/style.css

# Additional resources to cache
CACHE:
images/logo1.png
```

Workers

- 특정 작업을 병렬 스크립트로 백그라운드에서 실행 및 처리하기 위한 수단, HTML 표준
- 종류 :
  - Dedicated Workers, 라이프싸이클 - 페이지 종속적
  - Shared Workers, 브라우징 (브라우저) 컨텍스트

Shared Worker

- Javascript UI 쓰레드와 별개의 쓰레드. Global script scope
- 페이지에 비종속적 (페이지 라이프 싸이클과 별개)
- 직접적인 DOM 접근 불가

그리하여 Service Worker 가 등장합니다.

> A service worker is a type of web worker. - W3C Spec -

> Service workers are a new browser feature that provide event-driven scripts that run independently of web pages - W3C Spec repo -

Service Worker 등록

- 브라우저에 존재 유무를 확인 후 `register()` 사용

```
if ('serviceWorker' in navigator) {
        // 간단한 실행
  navigator.serviceWorker.register('/service-worker.js');
        // Promise 이용
  navigator.serviceWorker.register('/service-worker.js').
    then(function (reg) {
      // 성공하면
      console.log('Okay it worked!', reg);
    }).catch(function (err) {
      // 실패해서 에러가 발생하면
      console.log('Oops, an error occured', err);
    })
}
```

then? catch? - Promise

> 기억하시죠? 초기 렌더링에 방해되는 리소스는 최대한 뒤로 미룹니다 :)

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    // sw 의 `install` 에 필요한 자원 네트워크 요청이, 초기 렌더링에 필요한 자원 요청보다 나중에 일어나
    navigator.serviceWorker.register('/service-worker.js');
  });
}
```

load? DOMContentLoaded? 직접 눈으로 확인

- 서비스워커가 동작할 url scope 지정하기

```
navigator.serviceWorker.register('/service-worker.js', {
  scope: './myApp'
});
```

> 서비스워커의 위치는 주 도메인과 같은 위치에 있어야 합니다.

실습 #1 - Service Worker 등록

1. Node & Express 설치 후 간단한 로컬 서버 구성
2. HTML, Image, CSS, JS 파일로 간단한 웹 페이지 생성
3. Service Worker 스크립트 파일 생성
4. 브라우저의 Service Worker 지원여부 확인
5. `navigator.serviceWorker.register()` 를 HTML 파일에 추가
6. 등록 여부 콘솔로 확인

Service Worker 설치

- `register()` 에서 등록한 스크립트 파일에서 `install()` 호출

```
self.addEventListener('install', function(event) {
  // 캐쉬 등록 또는 기타 로직 수행
});
```

```
var CACHE_NAME = 'cache-v1';
var filesToCache = [
  '/',
  '/js/app.js',
  '/css/base.css'
];
```

- `CACHE_NAME` : 캐쉬를 담을 파일명 정의
- `filesToCache` : 캐쉬할 웹 자원들 정의

```
self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(CACHE_NAME).then(function(cache) {
      // 위에 지정한 캐쉬 목록을 `cache-v1` 캐쉬에 추가
      return cache.addAll(filesToCache);
    })
  );
});
```

- `waitUntil()` : 안의 로직이 수행될 때 까지 대기

주의 : 캐쉬할 파일 중 한개라도 실패하면 전체 실패. 이를 해결하기 위해 sw-toolbox **사용 가능**

실습 #2 - Service Worker 설치

1. `self.addEventListener('install', fn)` 로 서비스워커 설치
2. cache 명 및 cache 할 파일 목록 지정
3. `install` 시에 위에서 지정한 캐쉬 등록
4. 개발자 도구로 서비스워커 설치 및 캐쉬 정상 등록 여부 확인

Service Worker **네트워크 요청 응답**

- 서비스워커 설치 후 캐쉬된 자원에 대한 네트워크 요청이 있을 때는 캐쉬로 돌려준다.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request).then(function(response) {
      if (response) {
        return response;
      }
      return fetch(event.request);
    })
  );
});
```

- `respondWith()` : 안의 로직에서 반환된 값으로 화면에 돌려줌
- `match()` : 해당 request 에 상응하는 캐쉬가 있으면 찾아서 돌려주고 아니면 `fetch()` 로 자원획득

실습 #3 - Service Worker 캐쉬 응답

1. `self.addEventListener('fetch', fn)` 로 캐쉬 응답 설정
2. `event.respondWith()` 로 네트워크 요청 가로채기 및 캐쉬 응답
3. 개발자 도구 Network 패널의 offline 체크 후 캐쉬응답 확인

Service Worker **활성화 및 업데이트**

- 새로운 서비스워커가 설치되면 활성화 단계로 넘어온다.
- 이전에 사용하던 서비스워커와 이전 캐쉬는 모두 삭제하는 작업 진행

```
self.addEventListener('activate', function(event) {
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          // 새로운 서비스 워커에서 사용할 캐쉬 이외의 캐쉬는 모두 삭제
          if (cacheWhitelist.indexOf(cacheName) === -1) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

> 기존에 실행 중인 서비스워커와 사이즈를 비교하여 1 바이트라도 차이나면 새걸로 간주

sw-toolbox

- **네트워크 요청과 캐쉬 관리에 추가적인 옵션(만료기한 등)을 제공**해주는 서비스워커 보조 라이브러리

```
# 설치
npm install --save sw-toolbox
```

- 사용방법은 아래와 같이 단순하다.

```
// service-worker.js
importScripts('bower_components/sw-toolbox/sw-toolbox.js');

// 참고 - importScripts
importScripts('a.js', 'b.js', ...); // 복수 라이브러리 로딩 가능
```

- 상세한 사용법은 여기서

SW Toolbox 동작 시연

sw-precache in Gulp

- 웹 자원을 런타임 시점 이전에 **사전 캐싱 가능한 서비스워커 생성 모듈**
- sw toolbox 라이브러리와 같이 사용 가능
- 캐싱 시점을 런타임 이전 또는 런타임 시로 변경 가능
- Cache First Strategy

```
# 설치
npm install --save-dev sw-precache
npm install --global sw-precache
```

사용조건

1. HTTPS - 서비스워커 조건
2. sw-precache NPM modules - 기본적인 설치
3. Registered Service Worker - top 레벨에 위치

사용방법

Gulp task 로 사용하는 방법

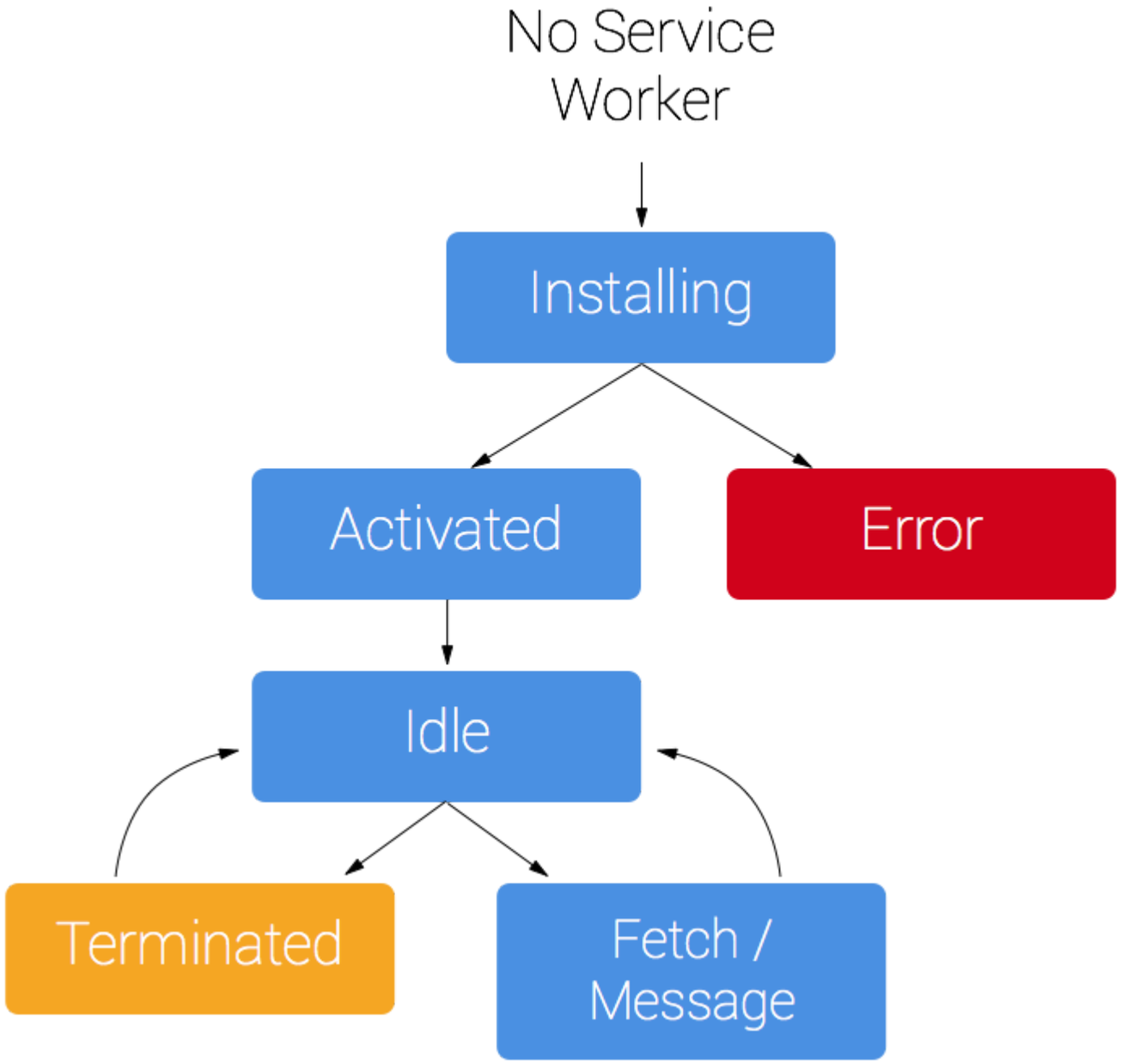
```
var swPrecache = require('sw-precache');

gulp.task('generate-service-worker', function(callback) {
  swPrecache.write(`${rootDir}/service-worker.js`, {
    staticFileGlobs: [rootDir + '/**/*.{js,html,css,png,jpg,gif,svg,eot,ttf,woff}'], //
    stripPrefix: 'app' // 빌드시점과 런타임 시점에 다를 수 있는 상대경로를 위해 앞 특정 문자열 제거
  }, callback);
});
```

CLI 를 이용하여 사용 가능

```
sw-precache
sw-precache --root=dist --static-file-globs='dist/**/*.html'
```

Service Worker **라이프 싸이클**

- **서비스워커는 웹 페이지와 별개의 생명주기**
- 서비스워커 등록 & 설치 & 활성화 과정을 잠깐 보면
    i. **웹페이지 에서** 서비스워커 스크립트 **호출**
    ii. 브라우저 **백그라운드** 에서 서비스워커 **설치**
    iii. **설치 과정에서** 정적 자원 **캐싱** (Cache 실패시 Install 실패)
    iv. **설치 후 활성화**. 네트워크 요청에 대한 가로채기 가능
- 사용하지 않을 때는 휴지상태. 필요시에만 해당 기능 수행
- 메모리 상태에 따라 자체적으로 종료하는 영리함


No Service Worker
Installing
Activated
Error
Idle
Terminated
Fetch / Message

Service Worker 디버깅

- 크롬 개발자 도구의 `Application` 패널에서 Service Workers
- 주소창에 `chrome://inspect/#service-workers`
- 주소창에 `chrome://serviceworker-internals`
- 주의사항 : 등록 과정에서 실패하였더라도 콘솔에 로그가 찍히지 않는 경우가 있음
    - 디버깅 권고 사항 - event.waitUntil() 내부 콜백 로직 확인

실습 #4 - Offline Web Service 제작

서비스워커를 이용하여 오프라인 동작가능한 웹 페이지를 제작 후 Github Page 에 호스팅. 참고

실습절차

1. 간단한 웹 페이지 생성 (HTML, JS, CSS, Images)
2. Service Worker 파일 생성
3. Register, Install, Register, Activate, Fetch 구현
4. 인터넷 미연결 상태에서 동작 확인

조건 : 실습에 포함할 파일

- html 1 개
- css 1 개
- js 1 개 이상
- 최소 image 1 개 이상

주제 : 자유

- ex) 신문 기사, Instargram Feed, 날씨 예보 등

> 제출 : jangkeehyo@gmail.com 로 해당 페이지 호스팅된 링크 보내주세요.

참고

- Service Worker Spec
- Service Worker Intro
- Instant and Offline Apps, Google - Slide Share
- Service Worker Explainer
- Offline Cookbook, Google
- Cache Storage - MDN
- Fetch() Spec
- Workbox, Google

끝