Angular extra features

Ing. Luigi Brandolini

Agenda

1. Internazionalizzazione
2. Angular & Material Design
3. Angular PWAs

Internazionalizzazione

Internazionalizzazione

- Per il supporto al tema dell'internazionalizzazione, sono presenti le seguenti alternative:
  - Angular `i18n` (internal module): complesso, macchinoso e non supportato da Ionic
  - Librerie di terze parti:
    - **`ngx-translate`** (la nostra scelta)

      `https://github.com/ngx-translate/`
    - `i18next`

Ngx-translate

- Processo di integrazione rapido e semplice
- Loader built-in per il caricamento di file di traduzione partendo da un determinato path
- Supporto per la compilazione Ahead-of-Time (un'app per ogni locale)
- File di traduzione in formato JSON, leggero e semplice da utilizzare
- Supporto all'interpolazione e al markup HTML nei file di traduzione
- Traduzioni mediante service, pipe, directive
- Supporto agli event listeners (esempio: onLangChange event)
- Possibilità di definire un handler custom di traduzione
- Funzionalità extra (es: plugins che includono un router speciale, loader .po files, ..)

Ngx-translate: installazione

- Consiste di due librerie:
  - **Core**: include le funzioni di base attraverso services, pipe, directives, per convertire il contenuto nei vari linguaggi:

```
npm i @ngx-translate/core --save
```

  - **Http-Loader**: include funzioni specifiche per il fetching dei file di traduzione da web server esterno

```
npm i @ngx-translate/http-loader --save
```

Configurazione

- Sarà poi necessario importare e registrare il **`TranslateModule`** nel file `app.module.ts`:

```
import { TranslateLoader, TranslateModule } from '@ngx-translate/core';
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';

// required for AOT compilation
export function HttpLoaderFactory(http: HttpClient) {
  return new TranslateHttpLoader(http);
}

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    TranslateModule.forRoot({
      defaultLanguage: 'en',
      loader: {
          provide: TranslateLoader,
          useFactory: HttpLoaderFactory,
          deps: [HttpClient]
      }
  })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Configurazione

- Creazione dei file di traduzione all'interno del progetto in `src/assets/i18n` (percorso di default, può anche essere modificata), in base ai linguaggi che si intende supportare:
  - `src/assets/i18n/en.json`
  - `src/assets/i18n/it.json`

Utilizzo

- Utilizzo nei template delle feature di traduzione mediante pipe 'translate'

```
<h1>{{ 'homepage.title' | translate }}</h1>
```

Locale IT (it.json):

```
{
    "homepage": {
        "title": "Welcome!"
    }
}
```

Locale EN (en.json):

```
{
    "homepage": {
        "title": "Welcome!"
    }
}
```

Angular Material Design and Flex Box

Material Design

- Material Design

```
ng add @angular/material
```

- In automatico, effettuerà:
  - aggiunta dipendenze al package.json
  - aggiunta Roboto font nell'index.html
  - aggiunta icon font Material Design all'index.html
  - aggiunta di regole di stile CSS globali per:
    - rimuovere i margini dal body
    - impostare height: 100% per html e body
    - Impostare Roboto come default application font
- Creare un modulo specifico per le componenti Material
- Reference:

https://material.angular.io/

Flex Layout

- Aggiungere Flex Layout (Responsive Layout):

```
npm install @angular/flex-layout@latest --save
```

es.:

```
import { FlexLayoutModule } from '@angular/flex-layout';

@NgModule({
    declarations: [ ... ],
    imports: [ ..., FlexLayoutModule ],
})
export class AppModule { }
```

- Reference: https://github.com/angular/flex-layout

- Responsive APIs: https://github.com/angular/flex-layout/wiki/Responsive-API

Angular PWAs Features

Features of a PWA

- Progressive (funzionano indipendentemente dalla piattaforma client, OS e browser)
- Responsive (indipendentemente da size, platform e orientation)
- Connectivity Independent (funziona anche in assenza di rete)
- App-like (anche se non viene installata o scaricata da qualche store)
- Fresh (sempre aggiornate, mediante Service Worker)
- Splash Screen (viene mostrato allo start-up, in maniera del tutto simile alla applicazioni native)

Workers

- Web workers e Service workers sono due tipi di workers
- Tra essi, sussistono analogie e differenze

Analogie:

- Sono entrambi eseguiti in thread secondario, consentendo al codice JavaScript di essere eseguito senza bloccare il main thread e la user interface
- Non hanno accesso agli oggetti Window e Document, perciò non possono interagire direttamente col DOM e anche il loro accesso alle browser APIs è limitato

Workers

Differenze:

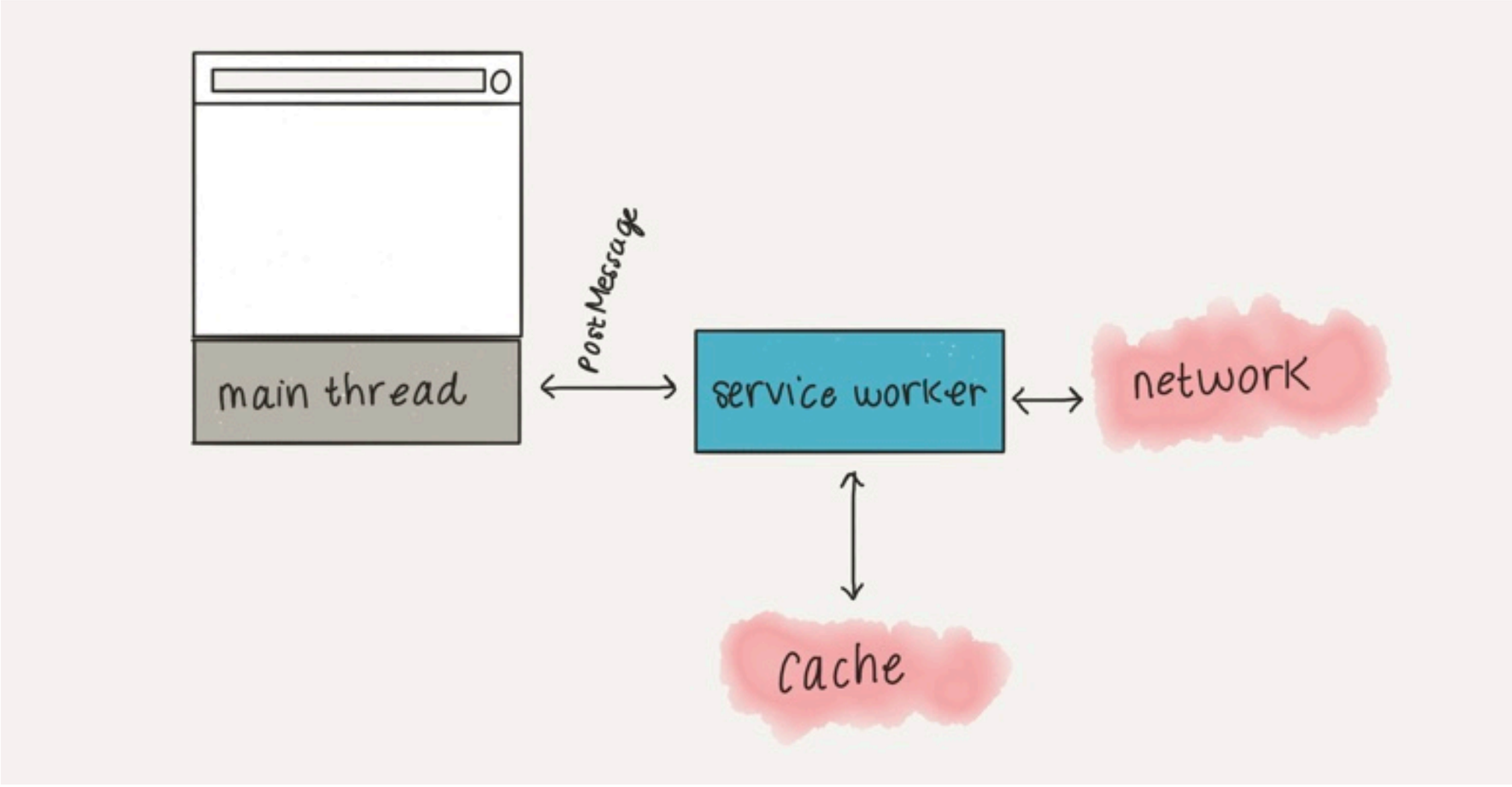
- Diversamente dai Web workers, i Service workers consentono di intercettare le network request (via *fetch* event) e di ricevere Push API events in background (via *push* event)

- Una pagina può avere più Web workers, ma un singolo Service worker controlla tutti i tab attivi, all'interno dello Scope in cui è registrato

- Il ciclo di vita di un Web worker è fortemente legato al tab a cui appartiene, mentre il ciclo di vita di un service worker è indipendente da esso (chiudendo il tab dove un Web worker è in esecuzione, quest'ultimo terminerà, mentre un Service worker potrà continuare a girare in background, anche se il sito non avrà alcun tab attivo aperto)

Web Worker

Service Worker



Setup

- Aggiunta del Service Worker:

`ng add @angular/pwa`
`(o stabile funzionante: @angular/pwa@13.3.7)`

- Questo comando:
  - Aggiunge il package al progetto `@angular/service-worker` all'interno del file `package.json`. In caso di problemi con l'aggiunta automatica del service-worker mediante `ng add`, si può procedere con l'installazione manuale:

```
npm i @angular/service-worker --force
```

  - Imposta, nel file `angular.json`, il flag "`serviceWorker`" a `true`. Grazie a questa impostazione, sarà possibile configurare nel dettaglio il funzionamento del service worker nell'applicazione
  - Importa e configura nell'app module il service worker tramite `ServiceWorkerModule`, il quale:
    - offre il servizio di SwUpdate (per la gestione degli aggiornamenti delle versioni)
    - offre il servizio di SwPush (per la gestione delle Web Push notifications)
    - carica lo script `ngsw-worker.js` nel browser
  - Aggiorna il file index.html:
    - Include un link per aggiungere il file `manifest.json`
    - Aggiunge i meta tags per il theme-color
  - Installa i file di icona per supportare la Progressive Web App (PWA) installataCrea il file di configurazione del service worker chiamato **`ngsw-config.json`**, che specifica i comportamenti di caching e altre impostazioni

ngsw-config.json (esempio)

```
{
  "$schema": "./node_modules/@angular/service-worker/config/schema.json",
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",
          "/manifest.webmanifest",
          "/*.css",
          "/*.js"
        ]
      }
    },
..,
  "dataGroups": [
    {
      "name": "books-api",
      "urls": [
        "/v1/**"
      ],
      "cacheConfig": {
        "strategy": "freshness", /* alternative: performance */
        "maxSize": 100,
        "maxAge": "3d2h",
        "timeout": "5s"
      }
    }
  ]
}
```

} **Offline capabilities**

Docs

- Per la configurazione dettagliata dell'Angular Service Worker, si rimanda a:

  https://angular.io/guide/service-worker-config

Offline caching

- Quando l'applicazione è offline, nello specifico, vengono sottoposti a caching:
  - `index.html`
  - `favicon.ico`
  - `Build artifacts (JS and CSS bundles)`
  - `Tutto ciò che si trova sotto assets`
  - `Immagini e fonts all'interno dell'outputPath configurato (per default ./dist/<project-name>/)`

Caching

- Modelli di progettazione di una PWA:
  - **Online first**: progettare l'applicazione per funzionare principalmente quando c'è connessione
  - **Offline first:** progettare l'applicazione per funzionare principalmente in assenza di connessione (più complesso)

Credits

Dott. Ing. Luigi Brandolini

Software Engineer, IT Professional Instructor

Contacts:

E-Mail: luigi.brandolini@gmail.com