Angular 5 Universal – Como usar TransferState

Published on 3 enero, 2018

En artículos anteriores te explicaba los primeros pasos para usar Angular Universal. Hoy te explicaré como se usa `TransferState` para evitar el molesto parpadeo (o *flickering*) que se produce al rehidratar la vista en cliente cuando muestras datos obtenidos de forma asíncrona.

¿Qué es el *flickering*?

Con *flickering* me refiero a un breve parpadeo de la vista. Un intervalo en el que

Universal, esto se produce cuando al instanciar la vista se le añaden datos generados de forma asíncrona (a partir de una API REST, por ejemplo).

La causa del parpadeo, es que el código en cliente no tiene ni idea del estado de la vista generada en servidor, así que al rehidratar la vista, vuelve a generar los datos de forma asíncrona eliminando temporalmente los datos anteriores.

En la siguiente animación puedes ver lo que pasa con Angular Universal cuando voy a una vista que carga contactos a través de una API REST.

El resultado de este parpadeo es muy molesto y algo que por supuesto nunca debería suceder en producción. Pero no es solo eso, es que además la petición a la API REST de la que obtengo los contactos se está realizando por duplicado, una vez en servidor y otra en cliente. **¡¡¡OMFG!!!**

Esto hasta hace poco no tenía solución en Universal, pero todo cambió al lanzar Angular 5.0

TransferState

El servicio `TransferState` que ofrece Angular desde su versión 5, permite mantener el estado de un componente generado en servidor y transferirlo al cliente. De este modo se optimiza el rendimiento (evitando peticiones duplicadas) y se elimina el molesto *flickering* del que te hablaba.

Usando TransferState

Código inicial

El código del ejemplo que acabas de ver lo he generado siguiendo mi artículo anterior. El componente donde hago la petición REST (sin `TransferState` de momento), es el siguiente:

app.component.ts

```
// src/app/app.component.ts

import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import 'rxjs/add/operator/map'
import { User } from './models';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {

  public people:User[] = [];

  constructor(private http: HttpClient){
  }

  ngOnInit(){
     this.http.get<any>('https://randomuser.me/api/?results=10&seed=foobar')
      .map(data => <User[]>data.results)
```

```
        this.people = data;
      });
  }
}
```

Como ves, estoy utilizando la API abierta *randomuser.me* para obtener 10 perfiles de usuario -siempre los mismos dado que usa la misma semilla- y le asigno el resultado a mi propiedad `people` .

1. Añadiendo TransferStateModule

Lo primero es añadir los módulos `ServerTransferStateModule` y `BrowserTransferStateModule` a los módulos de servidor y cliente respectivamente.

Fíjate:

app.server.module.ts

```
// src/app/app.server.module.ts

import { NgModule } from '@angular/core';
import { ServerModule, ServerTransferStateModule } from '@angular/platform-server';
import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-loader';

import { AppModule } from './app.module';
import { AppComponent } from './app.component';

@NgModule({
  imports: [
    AppModule,
    ServerModule,
    ServerTransferStateModule,
    ModuleMapLoaderModule
  ],
  providers: [
    // Add universal-only providers here
  ],
  bootstrap: [ AppComponent ]
```

app.module.ts

```
// src/app/app.module.ts

import { BrowserModule, BrowserTransferStateModule } from
'@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { NgModule} from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule.withServerTransition({appId:'contacts'}),
    HttpClientModule,
    BrowserTransferStateModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

2. Crear claves de estado para las propiedades

Lo siguiente sería crear una clave de estado para cada propiedad que se quiera transferir. En mi caso, sería para la propiedad `people` de **app.component.ts**:

```
// src/app/app.component.ts

//...other imports...
import { makeStateKey } from '@angular/platform-browser';

const PEOPLE_KEY = makeStateKey('people');

@Component({
    //...some stuff...
})
export class AppComponent implements OnInit {
  //...component stuff...
}
```

Ahora que tengo una clave que asociar al estado, solo necesito usar `TransferState` en mi **app.component.ts** para iniciar el estado durante la primera petición (en el lado servidor) y recuperarlo en el lado cliente (sin duplicar la petición).

```
// src/app/app.component.ts

//...other imports...
import { TransferState, makeStateKey } from '@angular/platform-browser';

const PEOPLE_KEY = makeStateKey('people');

@Component({
    //...some stuff...
})
export class AppComponent implements OnInit {
  public people:User[] = [];

  constructor(
    private http: HttpClient,
    private state: TransferState
  ){ }

  ngOnInit(){
    this.people = this.state.get<User[]>(PEOPLE_KEY, null);
    if(!this.people){
      this.http.get<any>('https://randomuser.me/api/?
results=10&seed=foobar')
      .map(data => <User[]>data.results)
      .subscribe(data => {
        console.log("data retrieved from API");
        this.people = data;
        this.state.set<User[]>(PEOPLE_KEY, data);
       });
    }
  }
  //...component stuff...
}
```

Fíjate en los cambios:

1. Inyecto `TransferState` en el constructor
2. Le asigno a la propiedad `people` el estado asociado a `PEOPLE_KEY` (si ya existe), o un objeto nulo en caso contrario.

4. Al resolver la petición, asigno los datos a la propiedad `people`
5. Además, los guardo también en el estado de `PEOPLE_KEY` .

Obviamente, siguiendo este esquema, cuando se instancie de nuevo el componente en cliente, el estado de `PEOPLE_KEY` se transferirá directamente a la propiedad `people` y por tanto ya no se realizará la petición REST para cargar los datos iniciales.

Mola ¿eh?

Apuesto a que lo has probado y…

¡ZASCA!

¡*flickering* al canto!

¡No funciona!

Bueno, el tema es que falta una última cosa…

4. Rehidratar cuando el DOM esté listo

Angular Universal empieza a ejecutar el cliente en cuanto puede y seguramente el estado de `TransferState` aún no está disponible. Para evitarlo, tienes que modificar el punto de entrada en cliente (**main.ts**) y retrasar la carga hasta que el DOM esté cargado:

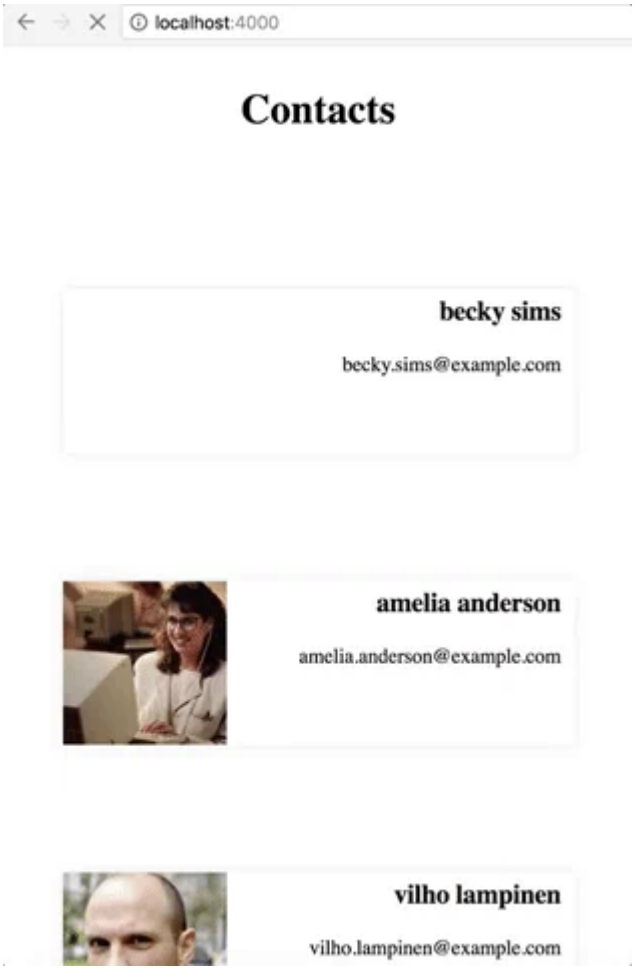
```
// src/main.ts

//...some imports...

if (environment.production) {
  enableProdMode();
}

document.addEventListener('DOMContentLoaded', ()=>{
  platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
});
```

Ahora sí, si compilas en universal y ejecutas, verás como ya no hay parpadeo.



Además, si miras la salida por consola tanto del servidor como del cliente, verás que la petición solo se ha ejecutado en el lado servidor.

Conclusiones

Gracias a la incorporación del servicio `TransferState` , Angular Universal parece estar listo por fin para su uso en producción. Y se trata de una gran incorporación.

Recuerda que el *Server Side Rendering* te permite reducir el tiempo de carga inicial de la página. Además, permite que tu página se posicione por SEO ya que los indexadores que no ejecuten JS dejarán de encontrarse una página en blanco.

Si te ha gustado este artículo, compártelo 😉

Published in Angular Javascript TypeScript

Previous Post
Angular 5 Universal: Guía rápida

No Newer Posts
Return to Blog

Be First to Comment

Deja un comentario

Introduce aquí tu comentario...