# HttpClient VS Http – El nuevo servicio http de Angular

Published on 15 noviembre, 2017

En Angular 4.3, se lanzó una nueva versión del cliente http, llamada HttpClient, que ha dejado deprecado el servicio Http original. Imagino que estás ansioso, así que voy a explicarte sus novedades. De paso, también te comento una bondad adicional que añade ahora Angular 5...

## Novedades

Aquí tienes las principales novedades que trae `HttpClient` con respecto al servicio `Http` original de Angular.

- JSON como respuesta por defecto
- Respuestas tipadas
- *Interceptors*
- Eventos de progreso
- RxJS 5.5 (en Angular 5)

## Introducción a HttpClient

`HttpClient` es un cliente con los métodos REST habituales (a continuación), que está basado en `Observables` . Básicamente es lo que vas a utilizar para hacer llamadas a una API REST y obtener resultados de la misma.

Estos son los métodos de `HttpClient` :

```
 get(url: string, options: {...}): Observable<any>
delete(url: string, options: {...}): Observable<any>
patch(url: string, body: any|null, options: {...}): Observable<any>
post(url: string, body: any|null, options: {...}): Observable<any>
put(url: string, body: any|null, options: {...}): Observable<any>
```

```
options(url: string, options: {...}): Observable<any>
jsonp<T>(url: string, callbackParam: string): Observable<T>

request(first: string|HttpRequest<any>, url?: string, options: {...}):
Observable<any>
```

Otros detalles que vale la pena recordar:

- **La suscripción se cancela al completar la petición**, así que no tienes que gestionarlo tú.
- Devuelve *cold observables*
- No se ejecuta ninguna *request* hasta que no se llama al método `subscribe()`
- Los objetos `HttpRequest`, `HttpHeaders` y `HttpParams` son inmutables.

Hechas las presentaciones... vamos a ver las novedades.

## JSON como respuesta por defecto

El nuevo servicio HttpClient te **devuelve** directamente **el body de la respuesta, parseado como JSON**. Esta aproximación simplifica el código en la mayoría casos. Eso sí, si trabajas con otro tipo de respuesta, tendrás que escribir algo más de código.

Déjame mostrarte una comparativa con el servicio antiguo.

```
//old Angular Http service
http.get('/api/items')
.map(res => res.json())
.subscribe(data => {
    console.log(data['someProperty']);
});
```

Como ves, antes tenías que mapear el resultado para convertirlo en JSON. Ahora ya no es necesario.

```
//new Angular HttpClient service
```

```
    console.log(data['someProperty']);
});
```

### ¿Y qué pasa si quiero recibir otro tipo de respuesta?

Los métodos REST de `HttpClient` aceptan un tercer parámetro (segundo en el caso `get` y `delete`) para detallar algunas opciones. Entre ellas, la propiedad `responseType` para indicar que se espera otro formato.

Las `responseType` que tienes son:

- arraybuffer
- blob
- json (es la opción por defecto)
- text

Funcionaría así:

```
//new Angular HttpClient service
http.get('/api/items', {responseType: 'text'})
.subscribe(data => {   // data is a string
    console.log(data);
});
```

### ¿Y si quiero acceder a algo más que el *body*?

Esta situación también está contemplada. Imagina que quieres acceder a los *headers*. Puedes acceder a la respuesta completa de la petición con la opción `{observe:'response'}`.

Así:

```
//new Angular HttpClient service
http.get('/api/items', {observe: 'response'})
.subscribe(response => {   // data is of type HttpResponse<Object>
    console.log(response.headers.get('X-Custom-Header');
    console.log(response.body['someProperty']); //response.body is a JSON
});
```

## Respuestas tipadas

`HttpClient` saca provecho de los *generics de TS* para *tipar* las respuestas. Si esperas un JSON con una estructura especifica, puedes indicarlo y trabajar cómodamente con tus tipos.

Mira el ejemplo:

```
 //new Angular HttpClient service

interface ItemsResponse {
  results: string[];
}

http.get<ItemsResponse>('/api/items').subscribe(data => {
  console.log(data.results); // OK, data is an instance of ItemsResponse
  console.log(data.test); // Linter & compiler error, test is not a property
of ItemsResponse

});
```

## Interceptors

Una de las cosas más criticadas del anterior servicio `Http` es que no disponía de interceptores (a diferencia de en AngularJS).

Los interceptores son un tipo de *middleware* que actúa de *proxy* entre tu cliente y el servidor. Gracias a ellos, puedes añadir funcionalidades genéricas a tu comunicación HTTP.

*- No lo pillo, ponme un ejemplo.*
*- ¡Claro, para eso estoy aquí!*

Un caso de uso muy habitual de los *interceptors* es el de añadir un token de autenticación a todas las peticiones. De este modo, centralizas la lógica en tu interceptor, y tus peticiones REST quedan mucho más limpias.

Otro ejemplo sería el de cachear las respuestas y servir desde el interceptor la respuesta de cache mientras se espera la respuesta del servidor, y una vez recuperada del servidor, actualizar la respuesta del cliente.

*- OK, mola, ¡ponme uno!*
*- Como te conozco...*

Un interceptor es una clase que implementa la interfaz `HttpInterceptor`.

Esta interfaz dispone de un método `intercept`, en el que se recibe la petición (`HttpRequest`) y una referencia al siguiente interceptor en la cadena (`HttpHandler`). Tienes que mover la *request* adelante para que pase por toda la cadena de interceptores y llegue a enviarse al servidor.

Aquí tienes un ejemplo simple de interceptor que incluye el token de autenticación:

```
 //AuthInterceptor.ts

//...some imports...
import {HttpEvent, HttpInterceptor, HttpHandler, HttpRequest} from
'@angular/common/http';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
   constructor(private auth: MyAuthService) {}

   intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
       // Get the auth header from your auth service.
       const authToken = this.auth.getAuthToken();
       const authReq = req.clone({headers: req.headers.set('Authorization',
`Token ${authToken}`)});
       return next.handle(authReq);
   }
}
```

Como ves, el interceptor en sí es muy simple. En el método intercept...
1. recojo el token de autenticación de un servicio interno
2. **clono la petición** con el header modificado
3. y propago la petición clonada (en lugar de la original) con `next.handle()`.

Aquí hay un par de detalles interesantes, relacionados con algo que decía al principio del artículo…

> Los objetos **HttpRequest**, **HttpHeaders** y **HttpParams** son inmutables.

No puedes modificar el objeto `HttpRequest` y propagarlo tal cual. Como se trata de un objeto inmutable, tienes que crear un nuevo objeto a partir del actual e incluir las modificaciones. Eso es lo que hago en `req.clone()` .

Sucede lo mismo con `req.headers` . **También es un objeto inmutable**. Si te fijas bien no modifico primero los headers y luego paso el objeto modificado. Estoy pasando directamente el resultado de `req.headers.set()` , porque `HttpHeaders.set()` no modifica el objeto `req.headers` , sino que devuelve uno nuevo con la propiedad que has añadido.

### OK. Tengo el interceptor… y ahora… ¿como lo uso?

Tienes que decirle a Angular que añada este *interceptor* al array de interceptores http. Esto lo haces en los *providers* de tu módulo principal:

```
 //app.module.ts

//...some imports...
import {NgModule} from '@angular/core';
import {HTTP_INTERCEPTORS} from '@angular/common/http';

@NgModule({
  providers: [
      {
            provide: HTTP_INTERCEPTORS,
            useClass: AuthInterceptor,
            multi: true,
     }
  ],
  //...more stuff...
})
export class AppModule {}
```

La propiedad `multi:true` , por cierto, es importante. Es lo que le indica a Angular que `HTP_INTERCEPTORS` es en realidad un array al que hay que incorporar el provider que indicas en `useClass` .

## Eventos de progreso

El servicio `HttpClient` te permite conocer el progreso tanto de la subida de tus peticiones, como de la descarga de sus respuestas. Para eso, dispones de la opción `reportProgress=true` .

Puedes usar esta opción directamente en el método `HttpClient.request()` . No obstante, si quieres usar métodos específicos como `HttpClient.get()` o `HttpClient.post()` , tendrás que acompañarla de la opción `observe="events"` , para recibir los eventos que permiten detectar el progreso.

Vamos con el código:

```
 //new Angular HttpClient service
http.get('/api/items', {observe: 'events', reportProgress: true})
    .subscribe(event=>{
      if (event.type === HttpEventType.DownloadProgress) {
        console.log(event.loaded); //downloaded bytes
        console.log(event.total); //total bytes to download
      }
      if (event.type === HttpEventType.UploadProgress) {
        console.log(event.loaded); //uploaded bytes
        console.log(event.total); //total bytes to upload
      }
      if (event.type === HttpEventType.Response) {
        console.log(event.body);
      }
    });
```

Como ves, de forma sencilla puedes comprobar el número de bytes recibidos y la progresión con respecto al total.

## RxJS 5.5

No es algo propio del servicio `HttpClient` (Angular 4.3+), pero sin duda está relacionado. Con el nuevo Angular 5, se usa por defecto una versión más actual de RxJS para tratar con *Observables*, la versión 5.5.

Esta actualización de RxJS tiene una mejora muy importante en términos de optimización (*code splitting / tree shaking*) pero también en términos de usabilidad, los ***lettable operators***.

Básicamente, esta nueva versión de RxJS utiliza los módulos de ES6 de la forma habitual, permitiéndote importar operadores de forma individual desde una librería común.

Me explico.

Antes, para importar los operadores *map* y *filter*, por ejemplo, tenías que importarlos cada uno de su propio archivo. Algo muy incómodo:

```
//rxjs old style
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/filter';
names = allUserData
.map(user => user.name)
.filter(name => name);
```

Sin embargo, con RxJS 5.5, puedes importarlos directamente de `rxjs/operators`. Eso sí, ya no puedes encadenar los operadores con `.`, sino que tienes que hacerlo mediante el nuevo método `pipe` (este es el cambio que se ha hecho por temas de optimización).

```
//rxjs 5.5 lettable operators
import { Observable } from 'rxjs/Observable';
import { map, filter } from 'rxjs/operators';
names = allUserData.pipe(
  map(user => user.name),
  filter(name => name),
);
```

**...¿como empiezo?**

## Usando HttpClient

El servicio `HttpClient` pertenece al módulo `HttpClientModule`, ambos expuestos en la librería `@angular/common/http`.

Para usarlo tienes que instalar dicha librería con `npm` o similar. Si usas una versión reciente de Angular CLI, ya lo instala al crear tu proyecto.

```
#terminal
npm install @angular/common/http
```

Una vez tienes la librería, debes importar `HttpClientModule` desde el módulo principal.

```
// app.module.ts:

//...some other imports...
import {HttpClientModule} from '@angular/common/http';

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule, // Include it after BrowserModule
    //...some more imports...
  ],
})
export class MyAppModule {}
```

El resto es coser y cantar. Importa tu servicio `HttpClient`, inyéctalo por DI, y ya estarás listo para usarlo.

```
//...some other imports...
import {HttpClient} from '@angular/common/http';

@Component(...)
export class MyComponent implements OnInit {
```

```
  ngOnInit(): void {
    //try some HTTP request:
    this.http.get('some/url').subscribe(data => {
        console.log(data);
    });
  }
}
```

## Conclusiones

He oído todo tipo de opiniones sobre este servicio. Unos muy felices, sobretodo por los *interceptors* y otros indignados por las respuestas JSON por defecto.

Disponer de interceptores es un punto positivo, pero también lo es la gestión del progreso e incluso el tipado de respuestas. Si tu servidor devuelve un formato distinto al JSON, siempre puedes indicarlo e incluso crearte un *wrapper* para no tener que repetir siempre la operación.

¿Mi veredicto? Sin duda, un cambio a mejor. ¿Y tú, qué opinas?

*Si te ha gustado este artículo, compártelo* 😉



**Relacionado**

AngularJS y Angular 2: diferencias del servicio HTTP
16 septiembre, 2016
En "Angular 2"

UPDATED - Intro a Angular (parte I): Modulo, Componente, Template y Metadatos
17 marzo, 2017
En "Angular 2"

Introducción a Angular 2 (parte I) - Modulo, Componente, Template y Metadatos
30 junio, 2016
En "Angular 2"

Published in Angular Javascript TypeScript

Previous Post
Usar componentes StencilJS desde Angular

No Newer Posts
Return to Blog

## Be First to Comment

## Deja un comentario

Introduce aquí tu comentario...