JavaScript - Frameworks - Angular - Http Requests

# HttpClient

Angular provides a simple HTTP client, the HttpClient service class in @angular/common/http. The service provides:

The HTTP client service offers the following major features.

* The ability to request typed response objects.
* Streamlined error handling.
* Testability features.
* Request and response interception.

## Fetch Data

The HttpClient.get() method can be used to send an AJAX request and return an observable. The return data depends on the observe and responseType values:

options: {

headers?: HttpHeaders | {[header: string]: string | string[]},

observe?: 'body' | 'events' | 'response',

params?: HttpParams|{[param: string]: string | string[]},

reportProgress?: boolean,

responseType?: 'arraybuffer'|'blob'|'json'|'text',

withCredentials?: boolean,

}

Import HttpHeaders and HttpParams from "@angular/common/http". An example set of params is: { params: new HttpParams().set('name', term) }

When defining the options object, do not infer the values, since it will not match the string union type expected. Either use 'as const' or set the option variable to the correct interface.

Specify the return typescript type when calling get(), for example assigning the <Config> interface:

this.http.get<Config>('/api/url');

Then subscribe to the observer and process the response data, specifiy the response data shape type as an interface in the processing method. Subscribe must be called, otherwise the request will not be made.

## Add to App

Import the HttpClientModule into the app module, after the browser module.

Create and inject the HttpClient service as a dependency of the application class:

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError, retry } from 'rxjs/operators';

@Injectable()

export class ConfigService {

constructor(private http: HttpClient) { }

getConfigResponse(): Observable<HttpResponse<Config>> {

return this.http.get<Config>(

this.configUrl, { observe: 'response' });

}

}

Then request:

showConfig() {

this.configService.getConfig()

// clone the data object, using its known Config shape

.subscribe((data: Config) => this.config = { ...data });

}

## Error Handling

If a request fails on the server, HttpClient returns an error object instead of a successful response. Therefore, the service which requests should also perform error handling.

### Error Types

There are three types of typical error:

* 400s - bad request - error response
* 500s - server error - error response
* client network fail - JS ErrorEvent

HttpClient will capture both types of error in HttpErrorResponse. In the service create a error handling function:

private handleError(error: HttpErrorResponse) {

if (error.error instanceof ErrorEvent) {

// A client-side or network error occurred. Handle it accordingly.

console.error('An error occurred:', error.error.message);

} else {

// The backend returned an unsuccessful response code.

// The response body may contain clues as to what went wrong.

console.error(

`Backend returned code ${error.status}, ` +

`body was: ${error.error}`);

}

// Return an observable with a user-facing error message.

return throwError(

'Something bad happened; please try again later.');

}

Then add the catchRrror pipe to the .get() observable, and retry a few times:

getConfig() {

return this.http.get<Config>(this.configUrl)

.pipe(

retry(3),

catchError(this.handleError)

);

}

## Send Data

Send data using .post(), .put(), or .delete(). To add body data, use the second parameter. For example:

addHero(hero: Hero): Observable<Hero> {

return this.http.post<Hero>(this.heroesUrl, hero, httpOptions)

.pipe(

catchError(this.handleError('addHero', hero))

);

}