HTTP module

Axios is richly featured HTTP client package that is widely used. Nest wraps Axios and exposes it via the built-in `HttpModule`. The `HttpModule` exports the `HttpService` class, which exposes Axios-based methods to perform HTTP requests. The library also transforms the resulting HTTP responses into `Observables`.

> info **Hint** You can also use any general purpose Node.js HTTP client library directly, including got or undici.

Installation

To begin using it, we first install required dependencies.

```
$ npm i --save @nestjs/axios axios
```

Getting started

Once the installation process is complete, to use the `HttpService`, first import `HttpModule`.

```
@Module({
  imports: [HttpModule],
  providers: [CatsService],
})
export class CatsModule {}
```

Next, inject `HttpService` using normal constructor injection.

> info **Hint** `HttpModule` and `HttpService` are imported from `@nestjs/axios` package.

```
@@filename()
@Injectable()
export class CatsService {
  constructor(private readonly httpService: HttpService) {}

  findAll(): Observable<AxiosResponse<Cat[]>> {
    return this.httpService.get('http://localhost:3000/cats');
  }
}
@@switch
@Injectable()
@Dependencies(HttpService)
export class CatsService {
  constructor(httpService) {
    this.httpService = httpService;
  }
  findAll() {
    return this.httpService.get('http://localhost:3000/cats');
  }
}
```

> info **Hint** `AxiosResponse` is an interface exported from the `axios` package (`$ npm i axios`).

All `HttpService` methods return an `AxiosResponse` wrapped in an `Observable` object.

Configuration

Axios can be configured with a variety of options to customize the behavior of the `HttpService`. Read more about them here. To configure the underlying Axios instance, pass an optional options object to the `register()` method of `HttpModule` when importing it. This options object will be passed directly to the underlying Axios constructor.

```
@Module({
  imports: [
    HttpModule.register({
      timeout: 5000,
      maxRedirects: 5,
    }),
  ],
  providers: [CatsService],
})
export class CatsModule {}
```

Async configuration

When you need to pass module options asynchronously instead of statically, use the `registerAsync()` method. As with most dynamic modules, Nest provides several techniques to deal with async configuration.

One technique is to use a factory function:

```
HttpModule.registerAsync({
  useFactory: () => ({
    timeout: 5000,
    maxRedirects: 5,
  }),
});
```

Like other factory providers, our factory function can be async and can inject dependencies through `inject`.

```
HttpModule.registerAsync({
  imports: [ConfigModule],
  useFactory: async (configService: ConfigService) => ({
    timeout: configService.get('HTTP_TIMEOUT'),
    maxRedirects: configService.get('HTTP_MAX_REDIRECTS'),
  }),
  inject: [ConfigService],
});
```

Alternatively, you can configure the `HttpModule` using a class instead of a factory, as shown below.

```
HttpModule.registerAsync({
  useClass: HttpConfigService,
});
```

The construction above instantiates `HttpConfigService` inside `HttpModule`, using it to create an options object. Note that in this example, the `HttpConfigService` has to implement `HttpModuleOptionsFactory` interface as shown below. The `HttpModule` will call the `createHttpOptions()` method on the instantiated object of the supplied class.

```
@Injectable()
class HttpConfigService implements HttpModuleOptionsFactory {
  createHttpOptions(): HttpModuleOptions {
    return {
      timeout: 5000,
      maxRedirects: 5,
    };
  }
}
```

If you want to reuse an existing options provider instead of creating a private copy inside the `HttpModule`, use the `useExisting` syntax.

```
HttpModule.registerAsync({
  imports: [ConfigModule],
  useExisting: HttpConfigService,
});
```

Using Axios directly

If you think that `HttpModule.register`'s options are not enough for you, or if you just want to access the underlying Axios instance created by `@nestjs/axios`, you can access it via `HttpService#axiosRef` as follows:

```
@Injectable()
export class CatsService {
  constructor(private readonly httpService: HttpService) {}

  findAll(): Promise<AxiosResponse<Cat[]>> {
    return this.httpService.axiosRef.get('http://localhost:3000/cats');
    //                      ^ AxiosInstance interface
  }
}
```

Full example

Since the return value of the `HttpService` methods is an Observable, we can use `rxjs` - `firstValueFrom` or `lastValueFrom` to retrieve the data of the request in the form of a promise.

```
import { catchError, firstValueFrom } from 'rxjs';

@Injectable()
export class CatsService {
  private readonly logger = new Logger(CatsService.name);
  constructor(private readonly httpService: HttpService) {}

  async findAll(): Promise<Cat[]> {
    const { data } = await firstValueFrom(
      this.httpService.get<Cat[]>('http://localhost:3000/cats').pipe(
        catchError((error: AxiosError) => {
          this.logger.error(error.response.data);
          throw 'An error happened!';
        }),
      ),
    );
    return data;
  }
}
```

> info **Hint** Visit RxJS's documentation on `firstValueFrom` and `lastValueFrom` for differences between them.