HttpClient

Performs HTTP requests. This service is available as an injectable class, with methods to perform HTTP requests. Each request method has multiple signatures, and the return type varies based on the signature that is called (mainly the values of `observe` and `responseType`).

See more...

```
class HttpClient {

  request(first: string | HttpRequest<any>, url?: string, options: { body?: any; headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  delete(url: string, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; body?: any; } = {}): Observable<any>

  get(url: string, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  head(url: string, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  jsonp<T>(url: string, callbackParam: string): Observable<T>

  options(url: string, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  patch(url: string, body: any, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  post(url: string, body: any, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams | { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json"; withCredentials?: boolean; } = {}): Observable<any>

  put(url: string, body: any, options: { headers?: HttpHeaders | { [header: string]: string | string[]; }; context?: HttpContext; observe?: "body" | "events" | "response"; params?: HttpParams
| { ...; }; reportProgress?: boolean; responseType?: "arraybuffer" | ... 2 more ... | "json";
withCredentials?: boolean; } = {}): Observable<any>

}
```

See also

- HTTP Guide
- HTTP Request

Description

Note that the `responseType` *options* value is a String that identifies the single data type of the response. A single overload version of the method handles each response type. The value of `responseType` cannot be a union, as the combined signature could imply.

Further information is available in the Usage Notes...

Methods

request()
mode_edit code

Constructs an observable for a generic HTTP request that, when subscribed, fires the request through the chain of registered interceptors and on to the server.

You can pass an HttpRequest directly as the only parameter. In this case, the call returns an observable of the raw HttpEvent stream.

Alternatively you can pass an HTTP method as the first parameter, a URL string as the second, and an options hash containing the request body as the third. See `addBody()`. In this case, the specified `responseType` and `observe` options determine the type of returned observable.

- The `responseType` value determines how a successful response body is parsed.
- If `responseType` is the default `json`, you can pass a type interface for the resulting object as a type parameter to the call.

The `observe` value determines the return type, according to what you are interested in observing.

- An `observe` value of events returns an observable of the raw HttpEvent stream, including progress events by default.
- An `observe` value of response returns an observable of HttpResponse<T>, where the `T` parameter depends on the `responseType` and any optionally provided type parameter.

request()
[mode_edit](#) [code](#)

- An `observe` value of body returns an observable of `<T>` with the same `T` body type.

delete()
[mode_edit](#) [code](#)

Constructs an observable that, when subscribed, causes the configured `DELETE` request to execute on the serve
See the individual overloads for details on the return type.

15 overloads...

Show Allexpand_more

delete()
mode_edit code

get()
mode_edit code

Constructs an observable that, when subscribed, causes the configured `GET` request to execute on the server. S
the individual overloads for details on the return type.

15 overloads...
Show Allexpand_more

get()
mode_edit code

get()
mode_edit code

head()
mode_edit code

Constructs an observable that, when subscribed, causes the configured `HEAD` request to execute on the server. The `HEAD` method returns meta information about the resource without transferring the resource itself. See the individual overloads for details on the return type.

15 overloads...

Show Allexpand_more

head()

mode_edit code

head()

mode_edit code

jsonp()

mode_edit code

Constructs an `Observable` that, when subscribed, causes a request with the special method `JSONP` to be dispatched via the interceptor pipeline. The JSONP pattern works around limitations of certain API endpoints that don't support newer, and preferable CORS protocol. JSONP treats the endpoint API as a JavaScript file an tricks the browser to process the requests even if the API endpoint is not located on the same domain (origin) a the client-side application making the request. The endpoint API must support JSONP callback for JSONP requests to work. The resource API returns the JSON response wrapped in a callback function. You can pass t callback function name as one of the query parameters. Note that JSONP requests can only be used with `GET` requests.

Constructs a `JSONP` request for the given URL and name of the callback parameter.

```
jsonp(url: string, callbackParam: string): Observable<Object>
```

Parameters

`url`	`string`	The resource URL.

jsonp()

mode_edit code

`callbackParam`	`string`	The callback function name.

Returns

`Observable<Object>`: An `Observable` of the response object, with response body as an object.

Constructs a `JSONP` request for the given URL and name of the callback parameter.

```
jsonp<T>(url: string, callbackParam: string): Observable<T>
```

Parameters

`url`	`string`	The resource URL.
`callbackParam`	`string`	The callback function name. You must install a suitable interceptor, such as one provided by HttpClientJsonpModule. If no such interceptor is reached, then the `JSONP` reques can be rejected by the configured backend.

Returns

`Observable<T>`: An `Observable` of the response object, with response body in the requested type.

options()

mode_edit code

Constructs an `Observable` that, when subscribed, causes the configured `OPTIONS` request to execute on the server. This method allows the client to determine the supported HTTP methods and other capabilities of an endpoint, without implying a resource action. See the individual overloads for details on the return type.

15 overloads...

Show Allexpand_more

options()
mode_edit code

options()
mode_edit code

patch()
mode_edit code

Constructs an observable that, when subscribed, causes the configured `PATCH` request to execute on the server
See the individual overloads for details on the return type.

15 overloads...

Show Allexpand_more

patch()

mode_edit code

post()

mode_edit code

Constructs an observable that, when subscribed, causes the configured `POST` request to execute on the server. The server responds with the location of the replaced resource. See the individual overloads for details on the return type.

15 overloads...

Show Allexpand_more

post()
mode_edit code

put()
mode_edit code

Constructs an observable that, when subscribed, causes the configured `PUT` request to execute on the server. The `PUT` method replaces an existing resource with a new set of values. See the individual overloads for detail
on the return type.

15 overloads...

Show Allexpand_more

put()
mode_edit code

Usage notes

Sample HTTP requests for the Tour of Heroes application.

HTTP Request Example

```
content_copy// GET heroes whose name contains search term

searchHeroes(term: string): observable<Hero[]>{


 const params = new HttpParams({fromString: 'name=term'});

   return this.httpClient.request('GET', this.heroesUrl,
{responseType:'json', params});

}
```

Alternatively, the parameter string can be used without invoking HttpParams by directly joining to the URL.

```
content_copythis.httpClient.request('GET', this.heroesUrl + '?' +
'name=term', {responseType:'json'});
```

JSONP Example

```
content_copyrequestJsonp(url, callback = 'callback') {

 return this.httpClient.jsonp(this.heroesURL, callback);

}
```

PATCH Example

```
content_copy// PATCH one of the heroes' name

patchHero (id: number, heroName: string): Observable<{}> {

const url = `${this.heroesUrl}/${id}`;   // PATCH api/heroes/42

 return this.httpClient.patch(url, {name: heroName}, httpOptions)

   .pipe(catchError(this.handleError('patchHero')));

}
```