Cheatography

Angular 2 Http Cheat Sheet

by Nathan (Nathane2005) via cheatography.com/28056/cs/8480/

Starting Up

```
@NgModule({
          dec lar ations: [],
          imp orts: [HttpM -
odule],
          boo tstrap: [],
          pro viders: []
})
```

Using Angular HTTP

```
constructor(private http: Http)
{}
```

Http Verbs

GET	get(url: string, options?: RequestOptionsArgs) : Observable<Response> Performs a request with get http method.
POST	post(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response> Performs a request with post http method.
PUT	put(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response> Performs a request with put http method.
DELETE	delete(url: string, options?: RequestOptionsArgs) : Observable<Response> Performs a request with delete http method.

Http Verbs (cont)

PATCH	patch(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response> Performs a request with patch http method.
HEAD	head(url: string, options?: RequestOptionsArgs) : Observable<Response> Performs a request with head http method.
OPTIONS	options(url: string, options?: RequestOptionsArgs) : Observable<Response> Performs a request with options http method.

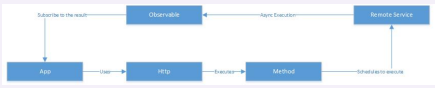
RequestOptionsArgs

```
interface Reques tOp tio nsArgs
{
   url : string
    method : string |Re que -
stM ethod
    search : string |UR LSe -
arc hParams
    headers : Headers
   body : any
    wit hCr ede ntials : boolean
    res pon seType : Respon -
seC ont entType
}
```

Headers

```
class Headers {
    sta tic fro mRe spo nse -
Hea der Str ing (he ade rsS -
tring: string) : Headers
    con str uct or( hea ders?:
Header s|{ [name: string]: any})
    app end (name: string,
value: string) : void
    del ete (name: string) :
void
    for Eac h(fn: (values:
string[], name: string, headers:
Map<st ring, string []>) =>
void) : void
    get (name: string) : string
    has (name: string) : boolean
    keys() : string[]
    set (name: string, value:
string |st ring[]) : void
    val ues() : string[][]
    toJ SON() : {[name: string]:
any}
    get All (name: string) :
string[]
    ent ries()
}
```

Observables - Flow



By **Nathan** (Nathane2005)

cheatography.com/nathane2005/

Published 17th October, 2016.
Last updated 17th October, 2016.

Angular 2 Http Cheat Sheet

by Nathan (Nathane2005) via cheatography.com/28056/cs/8480/

Sample Delegate

```
import { Injectable } from '@angular/core';
import {Http, Response, Headers, Reques tOp tions} from " @an - gul ar/ htt p";
import 'rxjs/Rx';
import {Obser vable} from " rxj - s";
import {Post} from "./h ttp /po - st.c la ss";
@Injec table()
export class HttpSe rvice {
    con str uct or( private http: Http) { }
    private reques tUrl: string = 'http: //l oca lho st: 400 0/p - osts';
   //Do all methods and observable options
    get Data(id : number) : Observ abl e<P ost> {
        return this.h ttp.get( his.re que stU rl} /${id})
            .ma p(t his.ma pRe - sponse)
            .ca tch (th is.h an - dle Error)
    }
    han dle Err or( error: any): Observ abl e<a ny> {
        con sol e.e rro r('An error occurred', error);
        return Observ abl e.t - hro w(e rro r.j son() || 'Server error');
    }
    map Res pon se( res ponse : Response) : Post {
        return respon se.j son();
    }
```

Sample Delegate (cont)

```
>  addData(body : Post) : Observable<Post> {
    let bodyString = JSON.stringify(body);
    let header = new Headers({
      'Content-Type' : 'application/json'
    });
    let options = new RequestOptions({
      headers : header
    });
    return this.http.post(${this.re que stUr l}, bodyString, options)
      .map(this.mapResponse)
      .catch(this.handleError);
  }
  updateData(body : Post) : Observable<Post> {
    let bodyString = JSON.stringify(body);
    let header = new Headers({
      'Content-Type' : 'application/json'
    });
    let options = new RequestOptions({
      headers : header
    });
    return this.http.put(${this.re que stU r l} /${ bod y.id}, bodyString, options)
      .map(this.mapResponse)
      .catch(this.handleError);
```

Sample Delegate (cont)

```
>  }
  deleteData(body : Post) : Observable<P- ost> {
    return this.http.delete(${this.re que s tU rl} /${ bod y.id})
      .map(this.mapResponse)
      .catch(this.handleError);
  }
}
```

Rx - Map

```
getData(id : number) : Observable<Post> {
        return this.h ttp.get( his.re que stU rl} /${id})
            .ma p(t his.ma pRe - sponse)
            .ca tch (th is.h an - dle Error)
    }
    map Res pon se( res ponse : Response) : Post {
        return respon se.j son();
    }
```

The map() function takes in a lambda function or a reference to a function that will execute the procedure and return the mapped result.
Accepts (res : Respone) and returns a result.

Rx - catch

```
getData(id : number) :
Observable<Post> {
        return this.h ttp.$g{et
his.re que stU rl} /${id})
           .ma p(t his.ma pRe -
sponse)
           .ca tch (th is.h an -
dle Error)
   }
    han dle Err or( error: any):
Observ abl e<a ny> {
        con sol e.e rro r('An
error occurred', error);
        return Observ abl e.t -
hro w(e rro r.j son() || 'Server
error');
   }
```

The catch reference is there to handle exceptions that are thrown. This gives you an opportunity to handle them in a graceful manner.
All rx operations return an observable.

Observable - Subscribe

observ-erOrNext	PartialObserver<T> | ((value: T) => void)
error	(error: any) => void
complete	() => void

```
this.service.getData(10).subscribe(
(data : Post) => {
this.result = data;
},
(error : any) => {
console.error(error);
}
)
```

Reference

```
private requestUrl: string =
'http://localhost:4000/posts';
    get Data(id : number) :
Observ abl e<P ost> {
        return this.h ttp.$g{et
his.re que stU rl} /${id})
           .ma p(t his.ma pRe -
sponse)
           .ca tch (th is.h an -
dle Error)
   }
```

Using the back ticks to specify internal references ``
referencing the content in ${}