How To Create a Caching Service for Angular

Explaining the proper way to cache HTTP calls using Angular and RxJS

Yury Katkov Follow

Sep 25, 2019 · 5 min read ★

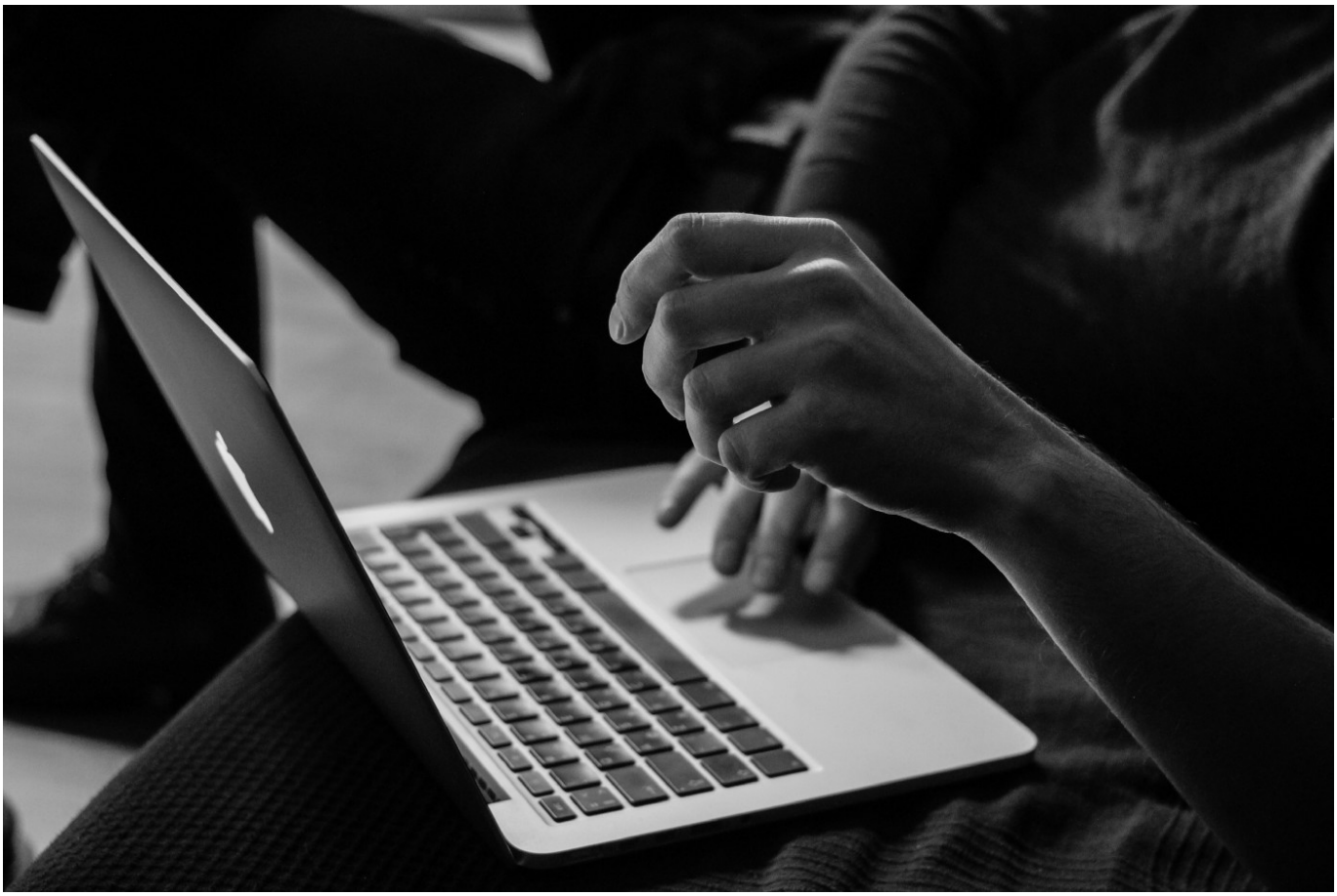


Photo by Sergey Zolkin on Unsplash

TL;DR: You need to cache the observable returned from `HttpClient` and combine it with `shareReplay` and `catchError`.

There's a lot of situations when we need to ask the server the same thing again and again. Typical examples are asking for translations for the terms, or resolving some codes using the vocabulary located on a server.

We don't want to bother the server too much, so we use caching. In this article, I will try to explain the proper way to cache HTTP calls using Angular and RxJS.

. . .

No Caching

So, suppose that we have a server that can return some nutrition information about a certain product. Let's use the Open Food Facts API for that.

I give it the code of the produce (e.g. `"7613034626844"`) and it gives me back an object with all the necessary information (e.g. `{name: "Ovomaltine milk powder"}`). Let's sketch an Angular service for that (see StackBlitz):

The service takes the code and passes it as a parameter to the API. We will create a super simple `ProductComponent` and call it in the constructor.

```
@Component({
  selector: 'my-product',
  template: `
<pre *ngIf="code">
  {{(result$|async)|json}}
</pre>
`,
})
export class ProductComponent  {
  @Input()
  code: string;
  result$: Observable<any>;

  constructor(private productService: ProductService)  {
      this.result$ = productService.resolveProduct(this.code);
  }

}
```

product-component.ts hosted with ❤️ by GitHub view raw

Now, let's use our `ProductComponent` like so in the `AppComponent`:

```
@Component({
  selector: 'my-app',
  template: `<my-product code="7613034626844"></my-product>
  <my-product code="7613034626844"></my-product>
  <my-product code="7613034626844"></my-product>`,
})
export class AppComponent  {

}
```

app.component.ts hosted with ❤️ by GitHub view raw

Sure enough, we will see that the request was performed three times, even though we were always asking the same information. Now, let's make it faster using caching.

. . .

Naïve Caching of the Value

The first idea for caching would be to actually cache the returned term, as I did here (see StackBlitz):

```
export class ProductService {

  private readonly URL = 'https://world.openfoodfacts.org/api/v0/product/';

  constructor(private http: HttpClient) {
  }

  cache = {};


  resolveProduct(code: string): Observable<any> {
    if (this.cache[code]) {
      console.log('Returning cached value!')
      return of(this.cache[code])
    }
```

```
    }
    console.log('Do the request again')
    this.http.get(this.URL+code+'.json').pipe(
      tap(resolvedValue => {
          this.cache[code] = resolvedValue;
      }));
  }
}
```

caching-article-caching-value.ts hosted with ❤️ by GitHub view raw

In theory, our service checks if the given term is presented in the cache, and, if so, returns it. If the product code is new, it asks the HTTP service to call it. When the server sends us the response, we put it into cache using the `tap` operator.

For example, a user clicks a button to see the info about the code `"7613034626844"` and our service will execute a call to a server.

When the users clicks the button next time to resolve the same term, our service will not initiate an HTTP call, using the cached value instead. At the first glance, everything works fine.

Problems with the naïve approach

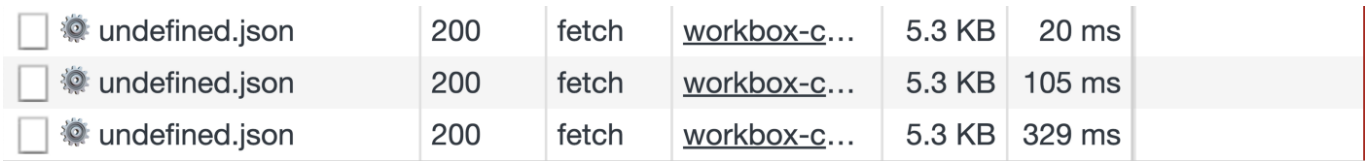
However, there is a caveat: we put the value into the cache asynchronously, and all asynchronous code will be executed after the synchronous code.

What if we call the `resolve` function in a `for` loop or in `*ngFor` or just have several components using it within the same template? It turns out that our cache doesn't do its job:

```
<my-product code="7613034626844"></my-product>
<my-product code="7613034626844"></my-product>
<my-product code="7613034626844"></my-product>
```

identical.html hosted with ❤️ by GitHub view raw

Suddenly, in the network tab, we see a lot of identical HTTP requests, so apparently, the cache can't catch up.



undefined.json	200	fetch	workbox-c…	5.3 KB	20 ms
undefined.json	200	fetch	workbox-c…	5.3 KB	105 ms
undefined.json	200	fetch	workbox-c…	5.3 KB	329 ms

The problem is that the reactions to HTTP calls happen only after all those HTTP calls are executed, because every asynchronous code is happening in the event loop.

. . .

Less Naïve Version — Cache the Observable Itself

We can fix this problem quite easily. Instead of caching the return values from the server, we can cache the observables that the Angular `HttpClient` service returns.

This way, both function calling and caching will be synchronous. The only trick is that we must use the `shareReplay` operator that will allow the subscribers to view the result of the HTTP call.

Without `shareReplay`, the observable will be kept in the `FINISHED` state after the request, and the new subscribers will not be able to get its value.

```
resolveProduct(code: string): Observable<any> {
    if (this.cache[code]) {
      console.log('Returning cached value!')
      return this.cache[code];
    }

    console.log('Do the request again')
    this.cache[code] = this.http.get(this.URL+code+'.json').pipe(
      shareReplay(1),
      catchError(err => {
          delete this.cache[code];
          return EMPTY;
      }));
}
```

non-naive.ts hosted with ❤ by GitHub view raw

non-naive.ts hosted with by GitHub

The result in the network tab looks exactly as we planned, we only have one HTTP call. Now, all that's left is to manage the exceptions.

Dealing with exceptions

What will happen if our server was down for a couple of seconds while we were asking it to resolve a term for that? The error HTTP code will raise an error in the observable.

After the observable is errored or finished, there's nothing we can do to restart it — that's a contract for the observables.

With our current approach, it means that we're caching the errors which is not a reasonable thing to do: the server will eventually start working and we would like to send it an HTTP request.

To achieve that, we can use the `catchError` operator which would delete an observable from the cache, so that, next time, the server will be called.

You can use the `delete` operator for that, or assign `false` to that key in the cache if you care about the performance more than readability.

The result: now we call the API only once.

```
Do the request again
2 Returning cached value!
```

. . .

Analysis

This particular caching approach is called *memoization* and it will serve you well in the following cases:

- The user asks for the same information a lot of times.
- This information on a server is stable by nature. Use it for vocabularies, translations, lookup services.
- Your application is not supposed to run for many days without reload.

All shortcomings of the memoization have a common reason, that is, we never clean our cached values. With that in mind, here are some restrictions:

- *Don't* use this caching method for checking real-time information: prices of stocks, weather, Twitter feed, etc. It simply won't work.
- *Don't* use this caching method if you expect your application to run for months and years without reload, because the cache size will become terribly big and you will have memory leaks.

Angular Rxjs Caching Typescript Programming