<epam>

Angular Performance optimization.

What’s the problem

The most common situation in enterprise development is developing for IE and you have to develop your application for IE. As you know, IE performance is really poor.

So the main problem in angular development is optimizing for IE.

Also you may need some of this advices to boost your application performance, if it too large to work as fast as you need.

Basic optimization

Before we start to optimize application, we should think about whole picture. Application optimization it is a complex thing.

Before optimizing the code itself, you should think about the infrastructure and environment.

Some technics allow you speed up your application before code optimization:

1) Amount of http requests should be decreased (bundling).

2) All data that could be cached, must be cached.

3) All compiled code must be minified

4) All unused code must be removed (tree-shaking)

5) AOT compilation

6) Pre-fetching resources and Lazy loading

7) Use web workers

8) Server side compilation, where it could be used

Quick tip

In fact, what do you need to speed up your application is:

- ChangeDetectionStrategy.OnPush,
- using trackBy in *ngFor,
- using manual manipulations with ChangeDetectorRef,
- denouncing your Rx streams,
- using enableProdMode in production,
- using aot compilation,
- using manual change detection in ngDoCheck to reduce heavy calculation.

Using trackBy in *ngFor

If track by is not specified, then the default is "track by $ id (item)" which generates "$$ hashKey".

The bottom line is that Angular needs to somehow map the array element and the DOM, as a result, an array identifier (note.id / hashkey / $ index / ...) is used for the array element. DOM for this element of the array, if the DOM element is not found, then it is considered that the element of the array is new and under it creates its own DOM and scope, which again bind to the identifier.

As a result, so that there would be a minimum of DOM rebuilds, you need to do so that the identifiers are saved from $ digest to $ digest, for this there are several track by modes.

```
@Component({
  selector: 'my-app',
  template: '<li *ngFor="let item of list; trackBy:identify">{{item.name}}</li>'
})
export class App {
  list:[{name: 'Gustavo'}, {name: ‘Ivan'}];
  public identify(index, item) {
    return item.name;
  }
}
```

Denouncing your Rx streams and managing subsciptions

To consume a stream we need to subscribe to that stream. When we subscribe to that stream a subscription will be created. That subscription will keep on living until the stream is completed or until we unsubscribe manually from that stream. Managing subscriptions is very important and in a number of cases we will have to manually unsubscribe an existing subscription to avoid memory leaks.

RxJS streams are very resource intensive, so do not create useless streams, manage your subscriptions and your application will be fast.

Using enableProdMode in production

Removes debug info from console also optimizes your application build.

enableProdMod() is already activated when you build your application with –prod flag

ChangeDetectionStrategy.OnPush

Every time something changed, Angular runs Change detection cycle FOR ALL COMPONENTS. This cycle is called dirty checking. The essence of the test is that Angular compares the new values with the old ones and updates the view if they are not equal.

Now imagine a large application with a huge number of components and a variety of conditions. If we allow Angular, each time the change detection cycle starts, to check each of these conditions, it will negatively affect performance.

Despite the fact that Angular is well optimized, as the application grows, it will have to work more and more.

OnPush strategy allow to not run checking every time, but run this checking manually only when we need it.

ngDoCheck hook

A lifecycle hook that invokes a custom change-detection function for a directive, in addition to the check performed by the default change-detector.

Note that ngDoCheck runs only for the top-level component with the OnPush strategy. It does not start for these child components.

AOT vs JIT compilation

JIT COMPILATION

Compile TypeScript just in time for executing it.

- Compiled in the browser.
- Each file compiled separately.
- No need to build after changing your code and before reloading the browser page.
- Suitable for local development.

AOT COMPILATION

AOT - Compile TypeScript during build phase.

- Compiled by the machine itself, via the command line (Faster).
- All code compiled together, inlining HTML/CSS in the scripts.
- No need to deploy the compiler (Half of Angular size).
- More secure, original source not disclosed.
- Suitable for production builds.

Additional info

- https://blog.strongbrew.io/rxjs-best-practices-in-angular/
- https://habr.com/ru/company/infopulse/blog/358860/
- https://stackoverflow.com/questions/41450226/angular-2-just-in-time-jit-vs-ahead-of-time-aot-compilation
- https://blog.angularindepth.com/everything-you-need-to-know-about-change-detection-in-angular-8006c51d206f