221)Introduction to pipes and why they are useful

Pipes is the feature built in into angular which allows you to transform your output in template. This is main purpose of pipe- to transform output. Now there are pipes for different types of output and for synchronous and asynchronous data.

Lets say we have property in our class-

let user =”Sumeet” ;

we use it in our template like this-

<p>{{user}}</p>

Now we want to display user in uppercase but we dnt want to change value as we might use it in other places. We can do this by built in pipe uppercase-

<p>{{user | uppercase}}</p>

222)Using Pipes

Pipe is responsible for changing output, so logical place to use it is in template. Here we used 2(uppercase and date) pipes in app.component.html.

223)Parametrizing pipes

Here we passed parameters to date pipe, we pass parameters by using “: ” code-

<strong>{{ server.name }}</strong> | {{ server.instanceType | uppercase }} |

{{ server.started | date: 'fullDate' }}

If our pipe accepts more than 1 parameter then we can pass them separated by :

224)where to learn more about pipes

In official documentation go to api refrence there search pipes. Here you can view all the pipes built into angular. url-

<https://angular.io/api>

here we saw splicePipe. It expects 2 arguments. Dive into theis documentation if you want to know more about pipes.

225)Chaining multiple pipes

We can chain pipes. It means output of one pipe is input to other. But note order is important. Here server.started is fetched to data pipe, then output of this is fetched as input to uppercase pipe.

Code-

<strong>{{ server.name }}</strong> | {{ server.instanceType | uppercase }} |

{{ server.started | date: 'fullDate' | uppercase}}

226)Creating a custom pipe

Sometimes we need functionality that is not in built, so we can use custom pipes for that.

We create a new file shorten.pipe.ts to implement our pipe .in this pipe we check if value is larger han 10 characters, if it is then we just return first 10 characters. now this class needs needs to have one special method to useable as a pipe. Also it is good practice to implement a interface that makes us define this special method in our class. This interface is PipeTransform.al through it is not necessary to implement this method.

This method is transform. Transform needs to receive value that should get transformed. So we pass value of type any as argument. Then we will receive list of arguments. As of now our pipe does’nt take any arguments , so we omit all other arguments.

Now transform always needs to return something. So here we use substr method on value passed and return it something. In order to use this pipe we need to add it to declaration section of app.module. so just like components and directives we also need to add pipes to declaration array.

Now we go to our custom pipe class and add @Pipe decorator and give it a name.

Code-

Shorten.pipe.ts-

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'shorten'

})

export class ShortenPipe implements PipeTransform {

transform(value: any): any {

if (value.length > 10) {

return value.substr(0, 10) + '...';

}

return value;

}

}

Now we use this pipe in app.component.html-

<strong>{{ server.name | shorten }}</strong> | {{ server.instanceType | uppercase }} |

{{ server.started | date: 'fullDate' | uppercase}}

227)Parametrizing a custom pipe

Now we want to improve our existing pipe we want user to pass inedexes upto which you want to return the string. Here we always using 10 characters which is hardcoded in pipe.

Code-

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'shorten'

})

export class ShortenPipe implements PipeTransform {

transform(value: any, limit: number): any {

if (value.length > limit) {

return value.substr(0, limit) + '...';

}

return value;

}

}

With this we allowed user to pass a parameter to the pipe because now you receive second argument in transform method. Now user may pass one parameter to the pipe. Code-

<strong>{{ server.name | shorten:5 }}</strong> | {{ server.instanceType | uppercase }} |

{{ server.started | date: 'fullDate' | uppercase}}

Our pipe will still work if we dnt pass any parameter. Also if we want to more than one parameter to pipe, just increase number of arguments to transform function.

While passing these parametrs use : ex-

228)Creating a filter Pipe

Lets now move to some advanced use cases and some limitations you might face when using pipes.

Let me show you something that might look strange at first look. We want to allow user to filter our servers by status. For ex- when you type status in text box you only see servers with stable status.

So we create text box where user can type status. Now we want to built a pipe which allow me to show only that servers which match status entered by user. So we create a new filter pipe in file- filter.pipe.ts we define transform function, here I want to implement logic which allows me to only return the elements of the array which fulfill this condition- status of server matches filterString. For this first I check if value. length(here value is array of servers, it does’nt have to be string. It can be any data you output in the end. So that’s the first important take away) is equal to 0, in this case array is empty, so we just return as it is. After that I loop through the array and check if status of array matches the filterString. Rest logic is simple check it. Code-

filter.pipe.ts-

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'filter'

})

export class FilterPipe implements PipeTransform {

transform(value: any, filterString: string, propname: string): any {

if (value.length === 0 ) {

return value;

}

const resultArray = [];

for (const item of value) {

if (item[propname] === filterString) {

resultArray.push(item);

}

}

return resultArray;

}

}

now we use our pipe in app.component. we apply it in \*ngFor loop. This might sound strange because before we only applied to string interpolation but keep in mind that I said in beginning of this module that pipes transform your output and the ngFor loop is only part of your output. Code-

app.component.html-

<li

class="list-group-item"

\*ngFor="let server of servers| filter:filteredStatus:'status'"

[ngClass]="getStatusClasses(server)">

<span

class="badge">

{{ server.status }}

</span>

<strong>{{ server.name | shorten:5 }}</strong> | {{ server.instanceType | uppercase }} |

{{ server.started | date: 'fullDate' | uppercase}}

</li>

Now our code works fine but there is a one issue that we will see in in next lecture.

229)Pure and Impure Pipes(or: how to “fix” the filter pipe)

Our pipe is working fine, but there is aa issue with it. If we allow user to add a server- we add **Add Server** button in app.component.html. now when we click on this button our server is added and we can see it on page. but if our servers are filtered by sttaus lets say by stable status, and we add stable status server then this server is not displayed. But as soon as we remove data from text field(i.e clear our filter) then we can see this server.

Reason for this behavior is that angular , thankfully I should say - not rerunning our pipe on data ,whenever this page data changes. So as soon as we change what we enter in text box, we will update our pipe or you can see rerun our pipe on data. So changing the input of the pipe will trigger recalculation , but changing the data won’t trigger it. To be precise changing arrays and objects won’t trigger it. This is good behavior otherwise angular have to rerun this pipe whenever any data on page changes. This would be bad because that would cost a lot of performance. This is reason why no built in filter pipe exists because filtering , because filtering you would say is very common task, but angular team decide against adding such a pipe because you typically have a high performance cost if you want to enforce it being updated even if you are in filter mode.

So it does’nt work here but we can force it to work but again be aware that following change will make sure that whenever we change data on page , our pipe is recalculate you could say. This could cause performance issue, that’s why I am emphasizing it. We can force this behavior by adding second property to @Pipe decorator- pure and give it value of false.

@Pipe({

name: 'filter',

pure: false

})

Now our pipe works as we wanted it to. Reason simply is that pipe now gets recalculated whenever data changes. So this what pure property is for, you can set it to false **which basically means this does not , you could say purely focusing on whether pipe arguments changed but now it will also be recalculated whenever anything changes on page**. and again this might be bad behavior I am just emphasizing this because it’s so important to understand . this might affect our performance if we have very long list here. This is our impure pipe, here we force update on each change.

230)Understanding the asyn pipe

There is a one built in pipe which I want to have closer look, which does something different than all the other pipes. It helps us in handling asynchronous data. To show how this works we create property appStatus on app.component.html. this is nt string it should get loaded after 2 seconds. To simulate this we use promise. You can imagine this data being retunred from http call or http server.

appStatus = new Promise((resolve, reject) => {

setTimeout(() => {

resolve('stable');

}, 2000);

});

So this will set appStatus to stable but after 2 seconds. Now we use this property in our template(app.component.html)-

<h2>App Status: {{appStatus}}</h2>

When we run our app, we see this-

App Status: [object Promise]

This is correct because this is a promise object. But after 2 seconds we know that it is not object. It is now a string but angular does not know because it does not watch our object. It does not see this object transforms to something else, or if if it retuns a value. it just knows that it is a promise and I am done. It’s good it behaves like this , it saves us performance.

Thankfully there is a pipe that we can use here which makes the transformation of data easier. We know that it will be string after 2 seconds. We want to output the string or it will resolve to string after 2 seconds I should say. So we do this-

<h2>App Status: {{appStatus | async}}</h2>

Now when we run our app, there is nothing in beginning but after some time we see our string there. This is what async pipe does-

It recognizes it is promise, as a side note -it also works for Observables. There it would subscribe automatically and after 2 seconds it would recognize that something changed, that the promise was resolved or in case of Observable that data was sent from the subscription and it will print that data to screen.

This is it in pipes. We will async pipe being used in http section.