Difference between Constructor and ngOnInit

Asked 3 years, 5 months ago Active 1 month ago Viewed 376k times

856

194

Angular provides life cycle hook `ngOnInit` by default.

Why should `ngOnInit` be used, if we already have a `constructor` ?

angular typescript ngoninit

edited May 29 at 1:17

Andres Gardiol
43 9

asked Mar 3 '16 at 5:14

Haseena P A
5,570 3 12 26

9 hey, check out my answer that explains the difference from the perspective of the inner workings of Angular – Max Koretskyi aka Wizard Aug 1 '17 at 6:17

2 check the article The essential difference between Constructor and ngOnInit in Angular – Max Koretskyi aka Wizard Sep 7 '17 at 10:55

22 Answers

913

The `Constructor` is a default method of the class that is executed when the class is instantiated and ensures proper initialization of fields in the class and its subclasses. Angular or better Dependency Injector (DI) analyzes the constructor parameters and when it creates a new instance by calling `new MyClass()` it tries to find providers that match the types of the constructor parameters, resolves them and passes them to the constructor like

```
new MyClass(someArg);
```

`ngOnInit` is a life cycle hook called by Angular2 to indicate that Angular is done creating the component.

We have to import `OnInit` in order to use like this (actually implementing `OnInit` is not mandatory but considered good practice):

then to use the method of `OnInit` we have to implement in the class like this.

```
export class App implements OnInit{
  constructor(){
     //called first time before the ngOnInit()
  }

  ngOnInit(){
     //called after the constructor and called  after the first ngOnChanges()
  }
}
```

> Implement this interface to execute custom initialization logic after your directive's data-bound properties have been initialized. ngOnInit is called right after the directive's data-bound properties have been checked for the first time, and before any of its children have been checked. It is invoked only once when the directive is instantiated.

Mostly we use `ngOnInit` for all the initialization/declaration and avoid stuff to work in the constructor. The constructor should only be used to initialize class members but shouldn't do actual "work".

So you should use `constructor()` to setup Dependency Injection and not much else. ngOnInit() is better place to "start" - it's where/when components' bindings are resolved.

For more information refer here:

- https://angular.io/api/core/OnInit
- Angular 2 Component Constructor Vs OnInit

edited Dec 16 '17 at 0:59

Zze
11.1k 6 47 76

answered Mar 3 '16 at 5:20

Pardeep Jain
46.5k 21 109 156

48 Exactly, most (or even all) class based languages have contructors to ensure proper initialization order especially of classes that extend other classes where some quite difficult issues can come up, like final fields (don't know if TS has them) and similar. Contstructors are not related to Angular2, they are a TypeScript feature. Lifecycle hooks are called by Angular after some initialization took place or when some event happend to allow the component act on certain situations and to give it the chance to do some tasks at proper times. – Günter Zöchbauer Mar 3 '16 at 5:24

6 There is a block quote in angular.io/docs/ts/latest/guide/server-communication.html that also explains this: "Components are easier to test and debug

3 *is a life cycle hook called by Angular2 to indicate that Angular is done creating the component.* - that's not exactly that. it signals that it's initialized the bindings. The component is created earlier. See my answer – Max Koretskyi aka Wizard Aug 1 '17 at 6:41

1 Yupp @absolutly correct – Kunvar Singh Oct 5 '17 at 13:17

13 As with all "best practices", I think it would be a good idea to also explain *why* you shouldn't be doing "work" in the constructor. This article by the Angular team lead is dense but may help: misko.hevery.com/code-reviewers-guide/… Also, less importance should be placed on the incantations required to implement OnInit (this is easy to find) and more on the critical fact that data-bindings aren't available in the constructor. – Reikim Oct 12 '17 at 20:54

117

The article The essential difference between Constructor and ngOnInit in Angular explores the difference from multiple perspectives. This answer provides the most important difference explanation related to the component initialization process which also shows the different in usage.

Angular bootstrap process consists of the two major stages:

- constructing components tree
- running change detection

The constructor of the component is called when Angular constructs components tree. All lifecycle hooks are called as part of running change detection.

When Angular constructs components tree the root module injector is already configured so you can inject any global dependencies. Also, when Angular instantiates a child component class the injector for the parent component is also already set up so you can inject providers defined on the parent component including the parent component itself. Component constructors is the only method that is called in the context of the injector so if you need any dependency that's the only place to get those dependencies.

When Angular starts change detection the components tree is constructed and the constructors for all components in the tree have been called. Also every component's template nodes are added to the DOM. The `@Input` communication mechanism is processed during change detection so you cannot expect to have the properties available in the constructor. It will be available on after `ngOnInit`.

Let's see a quick example. Suppose you have the following template:

```
<my-app>
   <child-comp [i]='prop'>
```

So Angular starts bootstrapping the application. As I said it first creates classes for each component. So it calls

element for the `child-comp` and calling `ChildComponent` constructor. At this stage it's not really concerned with the `i` input binding and any lifecycle hooks. So when this process is finished Angular ends up with the following tree of component views:

```
MyAppView
  - MyApp component instance
  - my-app host element data
      ChildCompnentView
        - ChildComponent component instance
        - child-comp host element data
```

Only then runs change detection and updates bindings for the `my-app` and calls `ngOnInit` on the MyAppComponent class. Then it proceeds to updating the bindings for the `child-comp` and calls `ngOnInit` on the ChildComponent class.

You can do your initialization logic in either constructor or `ngOnInit` depending on what you need available. For example the article Here is how to get ViewContainerRef before @ViewChild query is evaluated shows what type of initialization logic can be required to be performed in the constructor.

Here are some articles that will help you understand the topic better:

- Everything you need to know about change detection in Angular
- Angular's $digest is reborn in the newer version of Angular
- The mechanics of property bindings update in Angular

edited Sep 7 '17 at 10:54

answered Aug 1 '17 at 6:13

Max Koretskyi aka Wizard

55.9k 25 164 288

26 this should be the accepted answer. it actually explains the WHY rather than repeating mantras and stating `the constructor should only be used to inject dependencies` . – Stavm Aug 26 '17 at 12:29

8 way more accurate and interesting than the accepted answer. – yannick1976 Feb 22 '18 at 15:46

1 @yannick1976, thanks! Check out the referenced articles – Max Koretskyi aka Wizard Feb 22 '18 at 15:56

@flobacca, can you please rephrase the question, it's hard to understand what you're asking – Max Koretskyi aka Wizard Dec 11 '18 at 21:34

1 @MaxKoretskyiakaWizard you were right. I have done some mistake in my application setup. it's working as described by you. angular-c7zisx.stackblitz.io – user2485435 Feb 15 at 2:44

87

I think the best example would be using services. Let's say that I want to grab data from my server when my component gets 'Activated'. Let's say that I also want to do some additional things to the data after I get it from the server, maybe I get an error and want to log it differently.

It is really easy with ngOnInit over a constructor, it also limits how many callback layers I need to add to my application.

For Example:

```
export class Users implements OnInit{

    user_list: Array<any>;

    constructor(private _userService: UserService){
    };

    ngOnInit(){
        this.getUsers();
    };

    getUsers(){
        this._userService.getUsersFromService().subscribe(users =>  this.user_list =
users);
    };


}
```

with my constructor I could just call my _userService and populate my user_list, but maybe I want to do some extra things with it. Like make sure everything is upper_case, I am not entirely sure how my data is coming through.

So it makes it much easier to use ngOnInit.

```
export class Users implements OnInit{

    user_list: Array<any>;

    constructor(private _userService: UserService){
    };

    ngOnInit(){
        this.getUsers();
```

```
        this._userService.getUsersFromService().subscribe(users =>  this.user_list =
users);
        this.user_list.toUpperCase();
    };


}
```

It makes it much easier to see, and so I just call my function within my component when I initialize instead of having to dig for it somewhere else. Really it's just another tool you can use to make it easier to read and use in the future. Also I find it really bad practice to put function calls within a constructor!

answered Mar 3 '16 at 5:30

Morgan G

2,220 3 12 25

4 This answers the question a lot better, provides a clear example as to when not to use the constructor. – Ryder Bergerud Jul 12 '16 at 22:48

Your example could be simplified if you just set user_list to the Observable. Angular2 has the async pipe, so there wouldn't be any problems there. – DarkNeuron Feb 25 '17 at 17:34

27 As seen in the answer below this makes no difference if it's in the constructor. This is not a real answer to the purpose. – Jimmy Kane Jun 12 '17 at 16:52

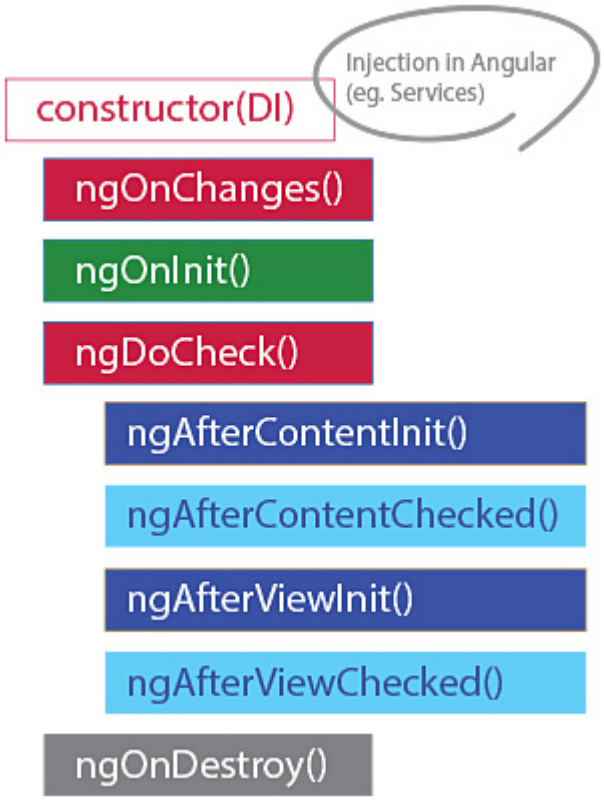
7 I don't see how this answers the question at all. Why couldn't you just put the code in the `constructor` ? – CodyBugstein Oct 26 '17 at 20:20

1 @Morgan why can't you just do `constructor(private _userService: UserService){ this.getUsers(); };` – Ashley Nov 15 '18 at 21:25

62

OK, first of all `ngOnInit` is part of **Angular lifecycle**, while `constructor` is part of **ES6** JavaScript class, so the major difference starts from right here!...

Look at the below chart I created which shows the lifecycle of Angular.



In Angular2+ we use `constructor` to do the `DI(Dependency Injection)` for us, while in Angular 1 it was happening through calling to String method and checking which dependency was injected.

As you see in the above diagram, `ngOnInit` is happening after the constructor is ready and `ngOnChnages` and get fired after the component is ready for us. All initialisation can happen in this stage, a simple sample is injecting a service and initials it on init.

OK, I also share a sample code for you to look, see how we get use of `ngOnInit` and `constructor` in the code below:

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';


@Component({
```

```
  styles: ['h1 { font-weight: normal; }']
})
class ExampleComponent implements OnInit {
  constructor(private router: Router) {} //Dependency injection in the constructor

  // ngOnInit, get called after Component initialised!
  ngOnInit() {
    console.log('Component initialised!');
  }
}
```

edited Jun 29 '18 at 7:41

answered Jun 5 '17 at 10:04

Alireza
59.5k 14 195 127

49

The first one (constructor) is related to the class instantiation and has nothing to do with Angular2. I mean a constructor can be used on any class. You can put in it some initialization processing for the newly created instance.

The second one corresponds to a lifecycle hook of Angular2 components:

Quoted from official angular's website:

> - `ngOnChanges` is called when an input or output binding value changes
> - `ngOnInit` is called after the first `ngOnChanges`

So you should use `ngOnInit` if initialization processing relies on bindings of the component (for example component parameters defined with `@Input`), otherwise the constructor would be enough...

edited Mar 3 '16 at 7:32

Günter Zöchbauer
359k 82 1142 1038

answered Mar 3 '16 at 6:36

Thierry Templier
155k 29 337 309

I will just add one important thing that was skipped in the explanations above and explains when you **MUST** use `ngOnInit` .

However, since `ngOnInit` happens once the component has been created and the checks (`ngOnChanges`) have been called you can access the DOM at this point.

```
export class App implements OnInit, AfterViewInit, AfterContentInit {
  @Input() myInput: string;
  @ViewChild() myTemplate: TemplateRef<any>;
  @ContentChild(ChildComponent) myComponent: ChildComponent;

  constructor(private elementRef: ElementRef) {
     // this.elementRef.nativeElement is undefined here
     // this.myInput is undefined here
     // this.myTemplate is undefined here
     // this.myComponent is undefine here
  }

  ngOnInit() {
     // this.elementRef.nativeElement can be used from here on
     // value of this.myInput is passed from parent scope
     // this.myTemplate and this.myComponent are still undefined
  }
  ngAfterContentInit() {
     // this.myComponent now gets projected in and can be accessed
     // this.myTemplate is still undefined
  }

  ngAfterViewInit() {
     // this.myTemplate can be used now as well
  }
}
```

edited Jan 31 at 9:35

answered Jan 29 '18 at 11:16

Miroslav Jonas

2,435 14 28

3 Nope. For `@ViewChildren` in particular, you need to use the `ngAfterViewInit` method. See here: stackoverflow.com/questions/46314734/… – AsGoodAsItGets Jan 30 at 15:57

1 Thanks, @AsGoodAsItGets for pointing it out. I have now improved the answer – Miroslav Jonas Jan 31 at 9:36

31

`Constructor` : `constructor` is a `default method` runs (*by deafult*) when component is being constructed. When you create `an instance` of a class that time also `constructor(default method)` would be called. So in other words, when component is being `constructed or/and an instance is created constructor(default method)` is called and relevant code written within is called. Basically and generally in `Angular2` it used to inject things like `services` when component is being constructed for the further use.

`OnInit` : ngOnInit is component's life cycle hook which runs first after `constructor(default method)` when component is being initialized.

So, Your constructor will be called first and Oninit will be called later after constructor method.

boot.ts

```
import {Cmomponent, OnInit} from 'angular2/core';
import {ExternalService} from '../externalService';

export class app implements OnInit{
   constructor(myService:ExternalService)
   {
           this.myService=myService;
   }

   ngOnInit(){
     // this.myService.someMethod()
   }
}
```

Resources: **LifeCycle hook**

You can check this **small demo** which shows implementation of both things.

edited Mar 3 '16 at 5:59

answered Mar 3 '16 at 5:20

micronyks

39.1k 12 80 116

4 I think "constructor is something which runs or called when component is initialized." is misleading. The constructor is a feature of the class not of the component. I'd say the instance of the class only becomes a component after the constructor was called **and** Angular did it's initialization. – Günter Zöchbauer Mar 3 '16 at 5:29

Yes changed the statement you can check now. – micronyks Mar 3 '16 at 5:31

1 Hmm, IMHO it's still the same "constructor(default method) is something which runs or called when component is constructed.". It's not only called when

2 Yes absolutely. Forgot to mention that when you create an object of a class also that time `constructor` would be called. But this answer has been written in angular2 context. To know the best answer you must be knowing OOPs basics. Still I'll update answer. – micronyks Mar 3 '16 at 5:39

@GünterZöchbauer, I don't think that it's a correct assertion that *is a feature of the class not of the component*. From the programming language perspective yes, this is correct. But I can successfully work with components without any lifecycle hooks at all. But I can't work with a component without a constructor if I need DI because that's the only injectable place. See my answer – Max Koretskyi aka Wizard Aug 1 '17 at 6:47

17

To test this, I wrote this code, borrowing from the NativeScript Tutorial:

user.ts

```
export class User {
    email: string;
    password: string;
    lastLogin: Date;

    constructor(msg:string) {
        this.email = "";
        this.password = "";
        this.lastLogin = new Date();
        console.log("*** User class constructor " + msg + " ***");
    }

    Login() {
    }
}
```

login.component.ts

```
import {Component} from "@angular/core";
import {User} from "./../../shared/user/user"

@Component({
  selector: "login-component",
  templateUrl: "pages/login/login.html",
  styleUrls: ["pages/login/login-common.css", "pages/login/login.css"]
})
export class LoginComponent {
```

```
  constructor() {
    this.user = new User("constructor");   // TWO
    console.log("*** Login Component Constructor ***");
  }

  ngOnInit() {
    this.user = new User("ngOnInit");   // THREE
    this.user.Login();
    this.isLoggingIn = true;
    console.log("*** Login Component ngOnInit ***");
  }

  submit() {
    alert("You're using: " + this.user.email + " " + this.user.lastLogin);
  }

  toggleDisplay() {
    this.isLoggingIn = !this.isLoggingIn;
  }

}
```

Console output

```
JS: *** User class constructor property ***
JS: *** User class constructor constructor ***
JS: *** Login Component Constructor ***
JS: *** User class constructor ngOnInit ***
JS: *** Login Component ngOnInit ***
```

answered May 24 '16 at 14:58

abbaf33f

1,210 1 13 35

16

The main difference between constructor and `ngOnInit` is that `ngOnInit` is lifecycle hook and runs after constructor. Component interpolated template and input initial values aren't available in constructor, but they are available in `ngOnInit` .

The practical difference is how `ngOnInit` affects how the code is structured. Most initialization code can be moved to `ngOnInit` - *as long as this doesn't create race conditions*.

A substantial amount of initialization code makes constructor method hard to extend, read and test.

A usual recipe for separating initialization logic from class constructor is to move it to another method like `init`:

```
class Some {
  constructor() {
    this.init();
  }

  init() {...}
}
```

`ngOnInit` can serve this purpose in components and directives:

```
constructor(
  public foo: Foo,
  /* verbose list of dependencies */
) {
  // time-sensitive initialization code
  this.bar = foo.getBar();
}

ngOnInit() {
  // rest of initialization code
}
```

Dependency injection

The primary role of class constructors in Angular is dependency injection. Constructors are also used for DI annotation in TypeScript. Almost all dependencies are assigned as properties to class instance.

Average component/directive constructor is already big enough because it can have multiline signature due to dependencies, putting unnecessary intialization logic to constructor body contributes to the antipattern.

Asynchronous initialization

Asynchronous initialization constructor can often be considered antipattern and have smell because class instantiation finishes before asynchronous routine does, and this can create race conditions. If it's not the case, `ngOnInit` and other lifecycle hooks are better places for this, particularly because they can benefit from `async` syntax:

```
constructor(
  public foo: Foo,
  public errorHandler: ErrorHandler
) {}

async ngOnInit() {
  try {
    await this.foo.getBar();
    await this.foo.getBazThatDependsOnBar();
  } catch (err) {
    this.errorHandler.handleError(err);
  }
}
```

If there are race conditions (including the one that a component shouldn't appear on initialization error), asynchronous initialization routine should take place before component instantiation and be moved to parent component, router guard, etc.

Unit testing

`ngOnInit` is more flexible than a constructor and provides some benefits for unit testing that are explained in detail in this answer.

Considering that `ngOnInit` isn't called automatically on component compilation in unit tests, methods that are called in `ngOnInit` can be spied or mocked after component instantiation.

In exceptional cases `ngOnInit` can be entirely stubbed to provide isolation for other component units (for instance, some template logic).

Inheritance

Child classes can only augment constructors, not replace them.

Since `this` cannot be referred before `super()`, this puts restrictions on initialization precedence.

Considering that Angular component or directive uses `ngOnInit` for time-insensitive initialization logic, child classes can chose whether `super.ngOnInit()` is called and when:

```
ngOnInit() {
  this.someMethod();
  super.ngOnInit();
}
```

edited Jun 20 '18 at 14:00

answered Feb 9 '18 at 12:35

Estus Flask

87.2k 25 141 262

15

Like a lot of other languages, you can initialize variables at the class level, the constructor, or a method. It is up to the developer to decide what is best in their particular case. But below are a list of best practices when it comes to deciding.

Class level variables

Usually, you will declare all your variables here that will be used in the rest of you component. You can initialize them if the value doesn't depend on anything else, or use const keyword to create constants if they will not change.

```
export class TestClass{
    let varA: string = "hello";
}
```

Constructor

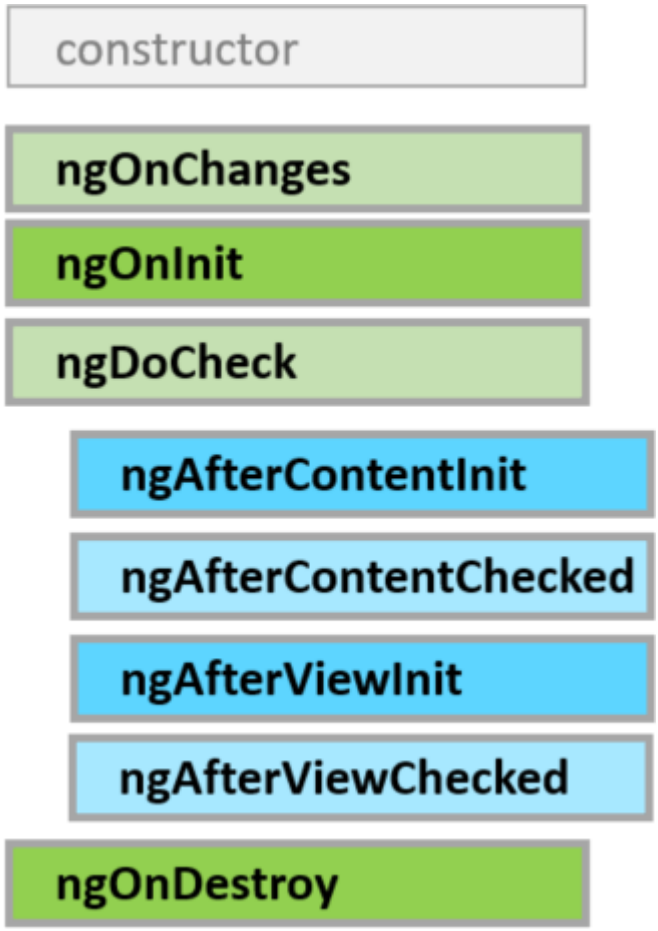
Normally it's best practice to not do anything in the constructor and just use it for classes that will be injected. Most of the time your constructor should look like this:

```
    constructor(private http: Http, private customService: CustomService) {}
```

this will automatically create the class level variables, so you will have access to `customService.myMethod()` without having to do it manually.

NgOnInit

NgOnit is a lifecycle hook provided by the Angular 2 framework. Your component must implement `OnInit` in order to use it. This lifecycle hook gets called after the constructor is called and all the variables are initialized. The bulk of your initialization should go here. You will have the certainty that Angular has initialized your component correctly and you can start doing any logic you need in `OnInit` versus doing things when your component hasn't finished loading properly.



https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

TLDR

If you are using Angular 2 framework and need to interact with certain lifecycle events, use the methods provided by the framework for this to avoid problems.

answered Mar 6 '17 at 21:59

Eduardo Dennis
9,539 10 64 87

12

The above answers don't really answer this aspect of the original question: What is a lifecycle hook? It took me a while to understand what that means until I thought of it this way.

1) Say your component is a human. Humans have lives that include many stages of living, and then we expire.

2) Our human component could have the following lifecycle script: Born, Baby, Grade School, Young Adult, Mid-age Adult, Senior Adult, Dead, Disposed of.

3) Say you want to have a function to create children. To keep this from getting complicated, and rather humorous, you want your function to only be called during the Young Adult stage of the human component life. So you develop a component that is only active when the parent component is in the Young Adult stage. Hooks help you do that by signaling that stage of life and letting your component act on it.

Fun stuff. If you let your imagination go to actually coding something like this it gets complicated, and funny.

answered Dec 9 '16 at 19:52

Preston

1,252 15 29

7

The ***constructor*** is a method in JavaScript and is considered as a feature of the class in es6 .When the class is instantiated it immediately runs the constructor whether it is used in Angular framework or not.So it is called by JavaScript engine and Angular has no control on that.

```
import {Component} from '@angular/core';
@Component({})
class CONSTRUCTORTEST {

//This is called by Javascript not the Angular.
     constructor(){
        console.log("view constructor initialised");
     }
}
```

The "ConstructorTest" class is instantiated below;So it internally calls the constructor(All these happens by JavaScript(es6) no Angular).

```
new CONSTRUCTORTEST();
```

```
import {Component} from '@angular/core';
@Component({})
class NGONINITTEST implements onInit{
   constructor(){}
   //ngOnInit calls by Angular
   ngOnInit(){
     console.log("Testing ngOnInit");
   }
}
```

First we instantiate the class as below which happen to immediate runs of constructor method.

```
let instance = new NGONINITTEST();
```

ngOnInit is called by Angular when necessary as below:

```
instance.ngOnInit();
```

But you may ask why we are using constructor in Angular?

The answer is ***dependencies injections***.As it is mentioned before, constructor calls by JavaScript engine immediately when the class is instantiated (before calling ngOnInit by Angular), so typescript helps us to get the type of the dependencies are defined in the constructor and finally tells Angular what type of dependencies we want to use in that specific component.

answered May 2 '17 at 6:14

Negin

1,397 9 19

7

Two things to observe here:

1. Constructor is called whenever an object is created of that class.
2. ngOnInit called once the component is created.

Both have different usability.


1,329 15 18

5

constructor() is the default method in the Component life cycle and is used for dependency injection. Constructor is a Typescript Feature.

ngOnInit() is called after the constructor and ngOnInit is called after the first ngOnChanges.

i.e. Constructor()->ngOnChanges()->ngOnInit()

as mentioned above ngOnChanges() is called when an input or output binding value changes.

answered Jun 8 '18 at 9:45


Shajin Chandran
425 4 6

4

Both methods have different goals/responsibilities. The task of the constructor (which is a language supported feature) is to make sure that the representation invariant holds. Otherwise stated to make sure that the instance is valid by giving correct values to the members. It is up to the developer to decide what 'correct' means.

The task of the onInit() method (which is an angular concept) is to allow method invocations on a correct object (representation invariant). Each method should in turn make sure that the representation invariant holds when the method terminates.

The constructor should be used to create 'correct' objects, the onInit method gives you the opportunity to invoke method calls at a well defined instance.

answered Jan 9 '17 at 19:20


Bruno Ranschaert
4,189 4 31 42

4

Constructor: The constructor method on an ES6 class (or TypeScript in this case) is a feature of a class itself, rather than an Angular feature. It's out of Angular's control when the constructor is invoked, which means that it's not a suitable hook to let you know when Angular has finished initialising the component. JavaScript engine calls the constructor, not Angular directly. Which is why the ngOnInit

As the constructor is initialised by the JavaScript engine, and TypeScript allows us to tell Angular what dependencies we require to be mapped against a specific property.

ngOnInit is purely there to give us a signal that Angular has finished initialising the component.

This phase includes the first pass at Change Detection against the properties that we may bind to the component itself - such as using an @Input() decorator.

Due to this, the @Input() properties are available inside ngOnInit, however are undefined inside the constructor, by design

answered Dec 20 '17 at 6:47

Vishal Gulati
4,750 6 47 71

2

Constructor is the first, and it happens sometimes when @input data is null! so we use Constructor for declare services and ngOnInit happens after. Exsample for contrutor:

```
constructor(translate: TranslateService, private oauthService: OAuthService) {
   translate.setDefaultLang('En');
       translate.use('En');}
```

Exsample for onInit:

```
ngOnInit() {
    this.items = [
      { label: 'A', icon: 'fa fa-home', routerLink: ['/'] },
      { label: 'B', icon: 'fa fa-home', routerLink: ['/'] }]
}
```

I think that onInit is like InitialComponents() in winForm .

answered Dec 5 '18 at 6:34

user1012506
1,402 15 34

1

1) Angular injector detect constructor parameter('s) and instantiate class.

2) Next angular call life-cycle

Angular Lifecycle Hooks

ngOnChanges --> Call in directive parameters binding.

ngOnInit --> Start angular rendering...

Call other method with state of angular life-cycle.

answered Nov 4 '17 at 6:02

Moslem Shahsavan

388 3 13

1

The `constructor` is called when Angular "instanciates/constructs" the component. The `ngOnInit` method is a hook which represents the initialization part of the component lifecycle. A good practice is to use it only for **service injection**:

```
constructor(private
    service1: Service1,
    service2: Service2
){};
```

Even if it is possible, you should not do some "work" inside. If you want to launch some action which have to occur at component "initialization", use `ngOnInit` :

```
ngOnInit(){
    service1.someWork();
};
```

Moreover, actions that involve **input properties**, coming from a parent component, can't be done in the contructor. They should be placed in `ngOnInit` method or another hook. It is the same for element related to the view (the DOM), for example, **viewchild elements**:

```
@Input itemFromParent: string;
@ViewChild('childView') childView:
```

```
    // childView is undefined here
};

ngOnInit(){
    console.log(itemFromParent); // OK
    // childView is undefined here, you can manipulate here
};
```

answered Jan 18 at 12:23

veben
2,529 5 17 31

1

I found the answer and I tried to translate it to english: This question still arised, even in technical interviews. In fact, there is a big resemblance between the two, but also there are some differences.

- The constructor is part of ECMAScript. On the other hand ngOnInit() is a notion of angular.
- We can call the constructors in all classes even if we do not use Angular
- LifeCycle: The constructor is called before ngOnInt ()
- In the constructor we can not call HTML elements. However, in ngOnInit () we can.
- Generally, calls of services in the ngOnInit () and not in the constructor

Source: http://www.angular-tuto.com/Angular/Component#Diff

answered Mar 13 at 8:33

doudou
11 1

0

`constructor()` is used to do dependency injection.

`ngOnInit()`, `ngOnChanges()` and `ngOnDestroy()` etc. are lifecycle methods. `ngOnChanges()` will be the first to be called, before `ngOnInit()`, when the value of a bound property changes, it will NOT be called if there is no change. `ngOnDestroy()` is called when the component is removed. To use it, `OnDestroy` needs to be `implement`ed by the class.

12.7k 10 81 139

0

Constructor is a function executed when component (or other class) is built.

ngOnInit is a function belonging to a component life-cycle method groups and they are executed in a diffrent moment of our component (that's why name life-cycle). Here is a list of all of them:

```
//Called once, after the first ngOnChanges()          ngOnInit

// Called before ngOnInit() and whenever one of       ngOnChanges
input properties change.

// Called just before Angular destroys the            ngOnDestroy
directive/component.

// Called during every change detection run           ngDoCheck

// Called after the ngAfterContentInit() and every    ngAfterContentChecked
subsequent ngDoCheck()..

// Called after the ngAfterViewInit() and every       ngAfterViewChecked
subsequent ngAfterContentChecked().

// Called once after the first ngDoCheck().           ngAfterContentInit

// Called once after the first                        ngAfterViewInit
ngAfterContentChecked()
```

answered May 28 at 15:30

Przemek Struciński
608 5 7