<https://codecraft.tv/courses/angular/angular-cli/overview/>

설치

1. Node.js깔기.
   1. npm package manager

http://cli.angular.io

node -v

npm -v

angular cli 설치

|  |
| --- |
| npm install -g @angular/cli  ng -help |

angular cli

|  |
| --- |
| ng new ProjectName -d |
| 프로젝트 생성 |

|  |
| --- |
| ng new ProjectName --skip-install |
| 프로젝트 생성 이지만 노드모듈폴더는 설치안함. |

라우터 포함 생성하기

|  |
| --- |
| ng new AngularNote --routing |

npm으로 노드 패키지모듈을 떙겨오자

|  |
| --- |
| npm i |
| 현재폴더의 package.json 으로 땡겨온다 |

시작하기

|  |
| --- |
| ng serve -o  ng serve --open |
| 서버 시작 localhost:4200  -o는 바로 브라우저 열림 |

빌드하기

|  |
| --- |
| ng build |
| ng build --prod --output-path ../../ |
| ng build -op ./webapp |
| ng build -op ./webapp -w |
| ng build --dev |

모듈은 여러개의 컴포넌트를 묶어서 관리해주는것

|  |  |
| --- | --- |
| main.ts | 프로그램에서 메인매서드같은것 |
|  |  |
|  |  |

--dev

디버그 모드

--prod

제품모드

ngStyle어트리뷰트 지시자

|  |
| --- |
| <p [ngStyle=”{color:’red’,backgroundCOlor:’yellow’}”>  내용  </p> |
| 컨포넌트에서 호출  <p [ngStyle=”{color:’red’,backgroundCOlor: getBgColor()}”>  export class NgStyleComponent imlements OnInit{  ...  //코드 비하인드에서 배경색 반환  getBgColor(): string{  return “yellow”;  //return (new Date).getSeconds() %2==0 “yellow”:”green”;  }  } |

모듈 생성

|  |
| --- |
| ng g m point -d |
| 모듈생성 |
|  |
| ng g m point –routing |
| 라우팅 까지 |
|  |
| ng g c point -m point/.point.module –spec false |
| 컴포넌트 생성하고 모듈에 적용까지 되라. |

서비스

|  |
| --- |
| ng g service point/point-data –spec false -m point/point.module -d |

클래스 생성

|  |
| --- |
| ng g class point/model |

인터페이스

|  |
| --- |
| export i nterface aaa{} |

라우팅쪽에 컴포넌트 등록

|  |
| --- |
| import {PointComponent} from “.pont.component;  cost routes :Routes =[  {  path :’point’  component : PointComponent,  data :{ title:”title”}  }  ] |

스트링 변수이용

|  |
| --- |
| this.http.get(‘${this.uri}/fff’) |

캐스팅

|  |
| --- |
| return <IPoint>response.json(); |

switch

|  |
| --- |
| this.ds.getPoint(111).subscribe(  (model:IPoint)=>{  },  (err)=>{...  },  ()=>{  //완료  }) |

subscribe

db->json 데이터 읽기

컴포넌트 사용 (구성요소)

|  |
| --- |
| ng generate component **name** –dry-run  ng g c name -d  같은 명령어 |
| name으로 컴포넌트 생성 될거라는 예시보여줌 -dry-run 이나 -d 빼면 생성됨 |
| --spec false -d  하면 스팩 파일은 빼고 처리됨 |
| ng g c header –spec false -d |
|  |
| ng g c footer –inline-template –spec false -d  ng g c footer –it –spec false -d |
| 컴포넌트 ts쪽에 인라인 템플릿쪽으로 html내용이 들어간다 (html생성안됨) |
| ng g c footer -it –inline-style –spec false -d  ng g c footer -it –is –spec false -d |
| 인라인 스타을로 만들어짐. |
|  |

가져다 쓰기

|  |
| --- |
| <app-componentname></app-componentname> |
| 셀렉터 이름을 태그로 넣으면된다. |

일반 경로로 처리할것인지 해쉬 값으로 처리할것인지 path를

|  |
| --- |
| **import** {Routes, RouterModule} **from** '@angular/router'; **import** {MainLayoutComponent} **from** "./shared/layout/app-layouts/main-layout.component"; **import** {AuthLayoutComponent} **from** "./shared/layout/app-layouts/auth-layout.component"; **import** {ModuleWithProviders} **from** "@angular/core";  **export const** routes: Routes = [  {  path: '',  component: MainLayoutComponent,  children: [  {  path: '', redirectTo: 'home', pathMatch: 'full'  },  {  path: 'home',  loadChildren: 'app/+home/home.module#HomeModule'  },  ]  },  ];  **export const** routing: ModuleWithProviders = RouterModule.*forRoot*(routes, {**useHash: true**}); |

pipe 만들기

|  |
| --- |
| ng g p +home\khh |
| **import** { Pipe, PipeTransform } **from** '@angular/core';  @Pipe({  name: 'khh' }) **export class** KhhPipe **implements** PipeTransform {   transform(value: **any**, args?: **any**): **any** {  **debugger**;  **return** value+"\_1111";  }  } |

흐름

|  |
| --- |
| index.html  </head> <body>  **<app-root>Loading...</app-root>**  <script type="text/javascript" src="inline.bundle.js"></script><script type="text/javascript" src="polyfills.bundle.js"></script><script type="text/javascript" src="styles.bundle.js"></script><script type="text/javascript" src="vendor.bundle.js"></script><script type="text/javascript" src="main.bundle.js"></script></body> </html> |
| main.ts  **import** './lib'  **import** { platformBrowserDynamic } **from** '@angular/platform-browser-dynamic'; **import** { enableProdMode } **from** '@angular/core'; **import** { environment } **from** './environments/environment'; **import** { AppModule } **from** './app/app.module';  **if** (environment.production) {  enableProdMode(); }  platformBrowserDynamic().bootstrapModule(AppModule); |
|  |

running 3가지방법

1. npm 으로 실행하는방법 package.json

2. npm으로 설치된 angular cli 설치된 플러그인으로 ng serve 처리하는방법

3.. gulp으로 angular 처리하는 방법

팁

비동기 형식 데이터 들어갈때 Directive 나 Component 쪽에 반영이 안될때

디텍팅해야된다.

|  |  |
| --- | --- |
| **import** {Component, OnInit, ChangeDetectorRef, ChangeDetectionStrategy} **from** '@angular/core'; **import** { HttpClient } **from** '@angular/common/http'; **import** { UserDetail } **from** 'app/domain/security/UserDetail'; **import** { Auth } **from** 'app/domain/security/Auth'; **import** {LoginInfoComponent} **from** "../../user/login-info/login-info.component";   @Component({   selector: 'sa-navigation',  templateUrl: './navigation.component.html',  changeDetection: ChangeDetectionStrategy.*OnPush* }) **export class** NavigationComponent **implements** OnInit {  auths:Array<Auth>;  **constructor**(**private** http: HttpClient, **private** changeDetector: ChangeDetectorRef) {  changeDetector.detach();  **this**.http.get<UserDetail>('/auth/detail',{}).subscribe(  data => {  console.log("Company: " + data);  **this**.auths = data.auths;  **this**.changeDetector.detectChanges();  // this.changeDetector.markForCheck();   },  err => {  console.log("Error occured.")  }  );  }   ngOnInit() {   }  } | **import** {  Directive,  ElementRef,  OnInit,  AfterViewInit,  Input } **from** '@angular/core';  **import** {  LayoutService } **from** "../../layout/layout.service"; **import** {  Router,  NavigationEnd } **from** "@angular/router"; **import** {  Subscription } **from** 'rxjs/Subscription'; **import** { setTimeout } **from** 'timers'; **import** {Auth} **from** "../../../domain/security/Auth";  **declare var** $: **any**;  @Directive({  selector: '[saSmartMenu]' }) **export class** SmartMenuDirective **implements** OnInit, AfterViewInit {   **private** $menu: **any**;  **private** layoutSub: Subscription;  **private** routerSub: Subscription;      **constructor**(  **private** menu: ElementRef,  **private** router: Router,  **public** layoutService: LayoutService  ) {  **this**.$menu = $(**this**.menu.nativeElement);  }   // @Input('saSmartMenu') set auths(auths:Array<Auth>){  // console.log("auth----",auths)  // // this.ngOnInit();  // // this.ngAfterViewInit();  // };    ngOnInit() {  **this**.layoutSub = **this**.layoutService.subscribe((store) => {  **this**.processLayout(store)  });   $('[routerLink]', **this**.$menu).off('click');  // collapse menu on mobiles  $('[routerLink]', **this**.$menu).on('click', () => {  **if** (**this**.layoutService.store.mobileViewActivated) {  **this**.layoutService.onCollapseMenu()  }  })   }   ngAfterViewInit() {  **this**.$menu.find('li:has(> ul)').each((i, li) => {  **let** $menuItem = $(li);  **let** $a = $menuItem.find('>a');  **let** sign = $('<b class="collapse-sign"><em class="fa fa-plus-square-o"/></b>');  $a.off('click');  $a.on('click', (e) => {  **this**.toggle($menuItem);  e.stopPropagation();  **return false**;  }).append(sign);  })   setTimeout(()=>{  **this**.processLayout(**this**.layoutService.store)  }, 200)    }   ngOnDestroy() {  **this**.layoutSub.unsubscribe();  }   **private** processLayout = (layoutStore) => {  **if** (layoutStore.menuOnTop) {  **this**.$menu.find('li.open').each((i, li) => {  **this**.toggle($(li), **false**)  })  } **else** {  **this**.$menu.find('li.active').each((i, li) => {  $(li).parents('li').each((j, parentLi) => {  **this**.toggle($(parentLi), **true**)  })  })  }   **if** (layoutStore.mobileViewActivated) {  $('body').removeClass("minified");  }  };   **private** toggle($el, condition = !$el.data('open')) {  $el.toggleClass('open', condition);   **if** (condition) {  $el.find('>ul').slideDown();  } **else** {  $el.find('>ul').slideUp();  }   $el.find('>a>.collapse-sign>em')  .toggleClass('fa-plus-square-o', !condition)  .toggleClass('fa-minus-square-o', condition);   $el.data('open', condition);   **if** (condition) {  $el.siblings('.open').each((i, it) => {  **let** sib = $(it);  **this**.toggle(sib, **false**)  })  }  }  } |
| <aside id="left-panel">   <!-- User info -->  <sa-login-info></sa-login-info>  <!-- end user info -->   <nav>  <!-- NOTE: Notice the gaps after each icon usage <i></i>..  Please note that these links work a bit different than  traditional href="" links. See documentation for details.  -->   <ul saSmartMenu>   <li \*ngFor="let auth of auths" routerLinkActive="active">  <a \*ngIf="'Y'!=auth.hddnYn && 'GET'!=auth.crudType && auth.menuLvl==1" routerLink="/home" title="Home">  <i class="{{auth.menuIcon}}"></i> <span class="menu-item-parent">[{{auth.hddnYn}}] -> {{auth.menuLvl}} {{auth.menuNm | i18n}}</span>  </a>   <ul>  <li routerLinkActive="active">  <a>vvvvvvvvvv</a>  </li>  </ul>  </li> | |

|  |
| --- |
| port 프록시 처리 |
| *##### ng server로 작업할때* **>** ./node\_modules/@angular/cli/bin/ng serve -w -o --proxy-config proxy.json |

# ㅊ[ Angular 2 : Lifecycle Hook ] 라이프 사이클

**Contents**[[show](https://blooom.co.kr/angular-2-lifecycle-hook-%EB%9D%BC%EC%9D%B4%ED%94%84-%EC%82%AC%EC%9D%B4%ED%81%B4/)]

#### 실행순서

1. ngOnChanges
2. ngOnInit
3. ngDoCheck
4. ngAfterContentInit
5. ngAfterContentChecked
6. ngAfterViewInit
7. ngAfterViewChecked

### 각 라이프 사이클 설명

#### 1. ngOnChanges

Data-bound Property 값이 변할 때.

#### 2. ngOnInit

ngOnChanges가 처음 실행되고 나서, 컴포넌트가 시작될 때. ( ngOnChanges 가 Data-bound Property 값이 변경될 때마다 실행되는 반면, ngOnInit는 첫번째 ngOnChanges 실행 이후 단 한번만 실행된다. )

#### 3. ngDoCheck

Angular 2 Change Detection Cycle 이 가동 될 때마다 실행된다. ngOnChanges 는 Data-bound Property 값이 변경될 때만 실행되지만, ngDoCheck은 Angular2의 ‘Detection Cycle’ 마다 실행되게 된다. 즉, Property에 변경 내용이 없더라도 실행이 되게 된다.

#### 4. ngAfterContentInit

Content( <ng-content> )가 삽입된 이후에 실행됨. 만약 ng-content 를 사용하지 않았다면, 신경쓸 필요가 없는 Hook 이다.

#### 5. ngAfterContentChecked

ng-content 안의 내용에 대한 Change detection 이 실행된 후. ngAfterContentInit와 마찬가지로 ng-content를 사용하지 않았다면, 해당 사항 없다.

#### 6. ngAfterViewInit

컴포넌트에 속한 모든 View 와 Child View 가 시작되고 나서. 즉, html 내용이 모두 표시되고 나서 실행된다.

#### 7. ngAfterViewChecked

컴포넌트의 View와 Child View에 대한 변경사항이 체크되고 나서 실행된다.

#### Life Cycle Test Example

app.component.ts

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. template: `
5. <h1>Life Cycle Component</h1>
6. <app-lifecycle \*ngIf="!delete"></app-lifecycle>
7. <button (click)="delete=true">Click to Delete</button>
8. `,
9. styles: [``]
10. })
11. export class AppComponent {
12. **delete** = false;
13. }

lifecycle.component.ts

1. import { Component } from '@angular/core';
2. import {
3. OnChanges,
4. OnInit,
5. DoCheck,
6. AfterContentInit,
7. AfterContentChecked,
8. AfterViewInit,
9. AfterViewChecked,
10. OnDestroy
11. } from '@angular/core';
12. @Component({
13. selector: 'app-lifecycle',
14. template: `
15. <p>
16. lifecycle Works!
17. </p>
18. `,
19. styles: []
20. })
21. export class LifecycleComponent implements
22. OnChanges,
23. OnInit,
24. DoCheck,
25. AfterContentInit,
26. AfterContentChecked,
27. AfterViewInit,
28. AfterViewChecked,
29. OnDestroy {
30. constructor() { }
31. ngOnChanges() {
32. **this**.log('ngOnChanges');
33. }
34. ngOnInit() {
35. **this**.log('ngOnInit');
36. }
37. ngDoCheck() {
38. **this**.log('ngDoCheck');
39. }
40. ngAfterContentInit() {
41. **this**.log('ngAfterContentInit');
42. }
43. ngAfterContentChecked() {
44. **this**.log('ngAfterContentChecked');
45. }
46. ngAfterViewInit() {
47. **this**.log('ngAfterViewInit');
48. }
49. ngAfterViewChecked() {
50. **this**.log('ngAfterViewChecked');
51. }
52. ngOnDestroy() {
53. **this**.log('ngOnDestroy');
54. }
55. private log(hook:string){
56. console.log(hook);
57. }
58. }

브라우저에서 실행하면 다음과 같이 콘솔창에 각 단계가 표시된다. 버튼을 누르면, lifecycle Work! 라는 내용이 사라지면서 ngDestroy 가 콘솔창에 표시될 것이다.

#### Life Cycle Test Example 2

보다 상세하게 어떻게 라이프 사이클이 작동하는 지 알아보자.

app.component.ts

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. template: `
5. <h1>Life Cycle Component</h1>
6. <app-lifecycle \*ngIf="!delete" [bindable]="boundValue">
7. <p #boundContent>{{test}}</p>
8. </app-lifecycle>
9. <button (click)="delete=true">Click to Delete</button>
10. <button (click)="test='Changed value'">Click to Change Content</button>
11. <button (click)="boundValue=2000">Click to Change Value</button>
12. `,
13. styles: [``]
14. })
15. export class AppComponent {
16. **delete** = false;
17. test = 'Starting Value';
18. boundValue = 1000;
19. }

lifecycle.component.ts

1. import { Component, Input, ViewChild, ContentChild, ElementRef } from '@angular/core';
2. import {
3. OnChanges,
4. OnInit,
5. DoCheck,
6. AfterContentInit,
7. AfterContentChecked,
8. AfterViewInit,
9. AfterViewChecked,
10. OnDestroy
11. } from '@angular/core';
12. @Component({
13. selector: 'app-lifecycle',
14. template: `
15. <ng-content></ng-content>
16. <hr>
17. <p #boundParagraph>bindable Property: {{bindable}}</p>
18. <p>{{boundParagraph.textContent}}</p>
19. `,
20. styles: []
21. })
22. export class LifecycleComponent implements
23. OnChanges,
24. OnInit,
25. DoCheck,
26. AfterContentInit,
27. AfterContentChecked,
28. AfterViewInit,
29. AfterViewChecked,
30. OnDestroy {
31. @Input() bindable = 1000;
32. @ViewChild('boundParagraph')
33. boundParagraph: ElementRef;
34. @ContentChild('boundContent')
35. boundContent: ElementRef;
36. constructor() { }
37. ngOnChanges() {
38. **this**.log('ngOnChanges');
39. }
40. ngOnInit() {
41. **this**.log('ngOnInit');
42. }
43. ngDoCheck() {
44. **this**.log('ngDoCheck');
45. }
46. ngAfterContentInit() {
47. **this**.log('ngAfterContentInit');
48. console.log(**this**.boundContent);
49. }
50. ngAfterContentChecked() {
51. **this**.log('ngAfterContentChecked');
52. }
53. ngAfterViewInit() {
54. **this**.log('ngAfterViewInit');
55. console.log(**this**.boundParagraph);
56. }
57. ngAfterViewChecked() {
58. **this**.log('ngAfterViewChecked');
59. }
60. ngOnDestroy() {
61. **this**.log('ngOnDestroy');
62. }
63. private log(hook: string) {
64. console.log(hook);
65. }
66. }

### ****Content vs View****

Content와 View를 분명히 구분할 줄 알아야 라이프 사이클을 이해할 수 있다. content 는 ng-content 처럼 바깥에서 컴포넌트 안으로 가져온 것. view는 원래 해당 컴포넌트 안에 존재하는 것.

html 안에 정의한 view child 나 content child 는 직접적으로 logic (ts files) 에서 참조해 가져올 수 없다. 따라서, @ViewChild() 나 @ContentChild() 와 같은 데코레이터를 이용해 가져온다. Dom Element 에 접근하는 법이다.

각 버튼을 실행시켜보면서 동작 상황을 살펴보자.