ทำความรู้จักกับ และใช้งาน Lifecycle Hook ใน Angular

09 January 2018 By Ninenik Narkdee (https://plus.google.com/u/0/107577507182658847616?rel=author)

lifecycle hook (tag_คำสั่ง_วิธีการ_ใช้งาน_การประยุกต์_ตัวอย่าง_เกี่ยวกับ-lifecycle_hook) angular (tag_คำสั่ง_วิธีการ_ใช้งาน_การประยุกต์_ตัวอย่าง_เกี่ยวกับ-angular)

คำสั่ง การ กำหนด รูปแบบ ตัวอย่าง เทคนิค ลูกเล่น การประยุกต์ การใช้งาน เกี่ยวกับ lifecycle hook angular

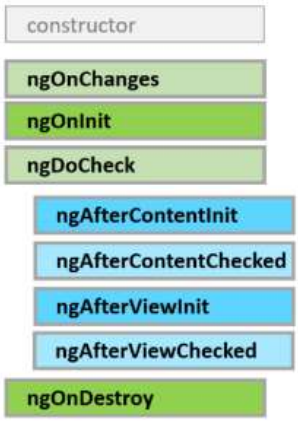
Copy ไปที่

http://niik.in/858

ในการใช้งาน Component ใน Angular จะมีระบบจัดการที่เรียกว่า Lifecycle ที่สามารถกำหนดให้ทำงาน
ในขณะที่มีการสร้างและแสดง component หลักและ component ย่อยขึ้นมา
หรือในขณะที่มีการเปลี่ยนแปลงค่า property ของ component รวมไปถึงในขณะที่มีการลบหรือล้างค่า
ก่อนจะทำการลบ component นั้นออกจาก DOM ไป

Angular ใช้งาน lifecycle hook ที่ทำให้เราสามารถเห็นภาพการเปลี่ยนแปลงและทำคำสั่งใดๆ ที่ต้องการ
ในระหว่างเกิดเหตุการณ์นั้นๆ ได้
โดยใน Directive จะมีรูปแบบ lifecycle hook ที่คล้ายกับ Component เพียงแต่ว่าใน Directive จะไม่มีในส่วนของ
การจัดการที่เกี่ยวกับ content และ view

ดูโครงสร้างลำดับการทำงานของ lifecycle hook จะมีรูปแบบตามรูปด้านล่าง



ใน Component มีการจัดการเกี่ยวกับ lifecycle โดย Angular เริ่มตั้งแต่
- เมื่อมีการสร้างและแสดงผล component
- สร้างและแสดงผล component ลูกภายใน
- ตรวจสอบเมื่อข้อมูลที่เชื่อมโยงมีการเปลี่ยนแปลงค่า
- และสุดท้ายทำลาย component ก่อนที่จะนำ component นั้นๆ ออกจาก DOM

ใน Directive จะมี lifecycle hook เหมือนกัน แต่จะไม่มีในส่วนที่เฉพาะสำหรับใน component ซึ่งได้แก่
- ngAfterContentInit()
- ngAfterContentChecked()
- ngAfterViewInit()
- ngAfterViewChecked()

โดยคำสั่ง lifecycle hook ทั้ง 4 รายการข้างต้นจะมีเฉพาะใน component แต่ไม่มีใน directive

ภาพรวมของ Component lifecycle hook

Directive และ Component จะเกิด lifecycle เมื่อ Angular สร้าง เปลี่ยนแปลงค่า และทำลาย directive และ component โดยเราสามารถที่จะเข้าไปจัดการในช่วงจังหวะเวลาที่เกิดขึ้นใน lifecycle โดยใช้ lifecycle hook interface ใน Angular core ไลบรารี่ ซึ่งเราต้อง import เข้ามาใช้งาน เมื่อจะมีการใช้งาน lifecycle hook interface ก็คือลักษณะหน้าตา ของ Object หนึ่งๆ จะคล้ายกับ class แต่ไม่ใช่ class เช่น อย่าง class จะมีคำสั่งและการทำงาน ส่วน interface จะมีแค่การกำหนดรูปร่างว่ามี property และ method แบบไหน ยกตัวอย่าง interface OnInit จะมีลักษณะ ดังนี้

```
interface OnInit {
  ngOnInit() : void
}
```

OnInit จะมีหน้าตาของคำสั่ง ngOnInit() ที่ไม่มีการคืนค่าใดๆ หรือ void
เวลาที่เราจะใช้งาน คำสั่ง ngOnInit() เราต้อง implements หรือใช้รูปแบบของ ngOnInit() จาก interface OnInit โดย import เข้ามาใช้งาน เป็นต้น

ในแต่ละ interface ของ lifecycle แต่ละตัว จะมีคำสั่งภายใน คำสั่งเดียว คล้ายๆ กับ OnInit interface โดยคำสั่งจะขึ้นต้นด้วย ng อย่าง interface "OnInit" จะมีคำสั่งเป็น ngOnInit()

ตัวอย่างการ import OnInit interface มาใช้งาน และการ implement เพื่อนำคำสั่ง ngOnInit() มาใช้งานใน AppComponent class

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

@Component({
  selector: 'my-app',
  template: `
  <span>Hellow</span>
  `,
})
export class AppComponent implements OnInit{

  constructor(){}
  ngOnInit(){

  }
}
```

Directive หรือ Component จะไม่มีการ implement lifecycle hook ทั้งหมดมาใช้งาน และบางคำสั่งก็เหมาะสำหรับใช้งานเฉพาะกับ component อย่างที่เราได้กล่าวไปแล้วข้างต้น
Angular จะเรียกใช้งานคำสั่ง lifecycle hook ใน directive หรือ component เมื่อมีการกำหนดเพื่อใช้งานเท่านั้น และ คำสั่ง constructor() ไม่ใช่ส่วนของ lifecycle hook

ลำดับการทำงานของ Lifecycle

หลังจากมีการสร้าง directive หรือ component ด้วยคำสั่ง constructor แล้ว Angular จะเรียกใช้งาน lifecycel hook ตามลำดับดังนี้

คำสั่ง ngOnChanges()

ถูกเรียกใช้งานเมื่อมีการตั้งค่าหรือกำหนดค่าให้กับการเชื่อมโยงข้อมูลของ input property โดยในคำสั่งจะมีการรับค่า parameter เป็น SimpleChanges Object ซึ่งจะเป็น object ที่เก็บค่าข้อมูลของ input property ที่เป็นค่าใหม่ที่ถูกส่งเข้ามา และข้อมูลของ input property ที่เป็นค่าเดิม ก่อนเปลี่ยนค่า
โดยคำสั่งนี้จะถูกเรียกใช้งาน 1 ครั้งก่อนคำสั่ง ngOnInit() และ จะถูกเรียกใช้งานอีกทุกครั้งที่มีการเปลี่ยนแปลงค่าของ input property เกิดขึ้น

คำสั่ง ngOnInit()

ถูกเรียกใช้งานเมื่อเริ่มต้น directive หรือ component หลังจากแสดง property ที่มีการเชื่อมโยงข้อมูล และกำหนดค่าให้กับ input property ของ directive หรือ component เรียบร้อยแล้ว
โดยจะถูกเรียกใช้ครั้งเดียว ต่อจาก คำสั่ง ngOnChanges() ที่ถูกเรียกใช้ครั้งแรก

คำสั่ง ngDoCheck()

ถูกเรียกใช้งานเมื่อมีการตรวจพบการเปลี่ยนแปลงที่ไม่ได้เกิดจาก การทำงานของ Angular หรือ Angular ไม่สามารถตรวจพบการเปลี่ยนแปลงนั้นได้ โดยจะถูกเรียกใช้งานทุกๆ ครั้งที่มีการตรวจพบการเปลี่ยนแปลงเกิดขึ้น
โดย เมื่อเริ่มต้น ngDoCheck() จะทำงานในทันทีหลังจากทำคำสั่ง ngOnChanges() และ ngOnInit() แล้ว
และจะมีการทำงานอีกในทุกๆ ครั้งที่มีการตรวจพบการเปลี่ยนแปลงเกิดขึ้น

คำสั่ง ngAfterContentInit()

ถูกเรียกใช้หลังจากมีการนำข้อมูลใดๆ มาแสดงใน component โดยจะถูกเรียกใช้งานครั้งเดียวหลังจาก ทำคำสั่ง ngDoCheck() ในครั้งแรกไปแล้ว
* ใช้งานสำหรับ component เท่านั้น

คำสั่ง ngAfterContentChecked()

ถูกเรียกใช้งานหลังจากมีการตรวจสอบข้อมูลที่นำมาแสดงใน component โดยจะถูกเรียกใช้งานหลังจากทำคำสั่ง ngAfterContentInit() และถูกเรียกใช้งานทุกๆ ครั้งที่มีการทำคำสั่งย่อย ngDoCheck()
* ใช้งานสำหรับ component เท่านั้น

คำสั่ง ngAfterViewInit()

ถูกเรียกใช้งานหลังจากมีการกำหนดค่าเริ่มต้นการแสดงของ component หลัก และ component ย่อยแล้ว
โดยถูกเรียกใช้งานเพียงครั้งเดียว หลังจากทำคำสั่ง ngAfterContentChecked() ในครั้งแรก
* ใช้งานสำหรับ component เท่านั้น

คำสั่ง ngAfterViewChecked()

ถูกเรียกใช้งานหลังจากมีการตรวจสอบการแสดงของ component หลัก และ component ย่อยแล้ว
โดยจะถูกเรียกใช้งานหลังทำคำสั่ง ngAfterViewInit() และถูกเรียกใช้งานทุกๆ ครั้งที่มีการทำคำสั่งย่อย
ของ ngAfterContentChecked()
* ใช้งานสำหรับ component เท่านั้น

คำสั่ง ngOnDestroy()

ถูกเรียกใช้ให้ทำงานต่างๆ ก่อนที่จะมีการลบหรือทำลาย directive/component ยกตัวอย่างเช่น ทำการ Unsubscribe Observable และยกเลิกการจัดการกับ event เพื่อคืนค่าหน่วยความจำ หลีกเลี่ยงปัญหา memory ไม่เพียงพอ เหล่านี้เป็นต้น

ตัวอย่างการใช้งาน Lifecycle Hook

โดยปกตเมื่อเรามีการใช้งาน Angular cli ในการสร้างไฟล์ต่างๆ เราจะเห็นว่ามีการ implement OnInit มาให้อัตโนมัติ
เรามาดูไฟล์ home.componet.ts ดังนี้

ไฟล์ home.componet.ts

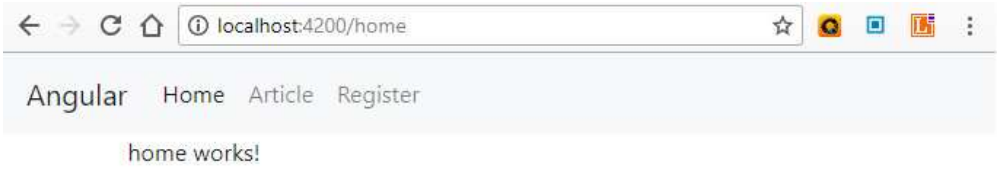
```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    // selector: 'app-home',
5    templateUrl: './home.component.html',
6    styleUrls: ['./home.component.css']
7  })
8  export class HomeComponent implements OnInit {
9
10   constructor() {
11     console.log("constructor work..");
12    }
13
14   ngOnInit() {
15     console.log("OnInit work..");
16   }
17
18 }
```

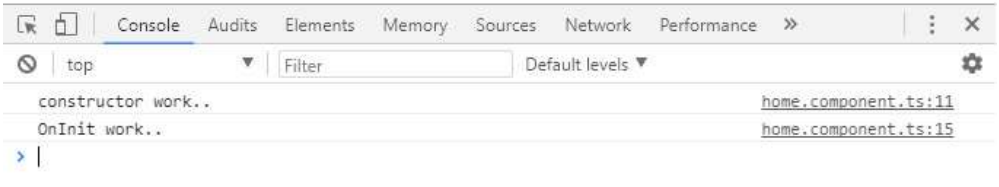
เมื่อรัน app และตรวจสอบค่าผ่าน console เราจะพบว่า มีการทำงานในส่วนของ constructor ก่อน จากนั้นก็ทำงานในส่วนของ ngOnInit() ตามลำดับ
เราลองเพิ่ม ngOnDestroy() เข้าไป โดยในส่วนของ OnDestroy ให้เราทำการ import และทำการ implement ตามโค้ดด้านล่าง

ไฟล์ home.componet.ts

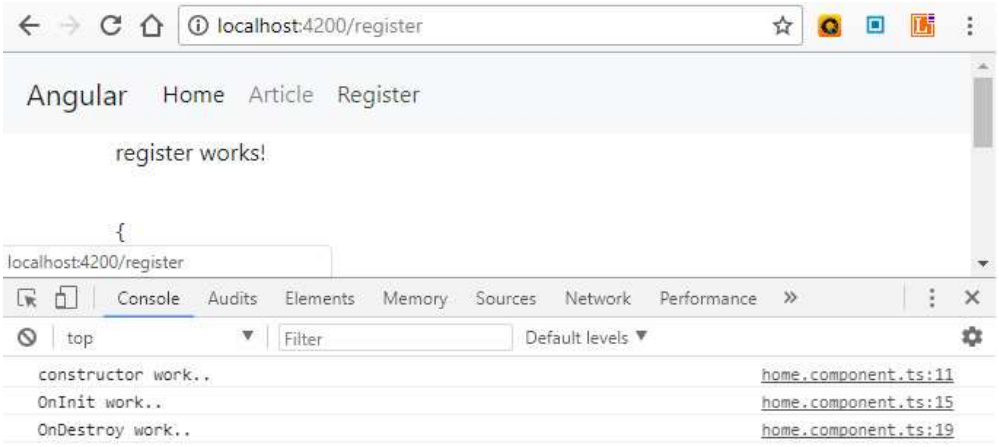
```
import { Component, OnInit, OnDestroy } from '@angular/core';

@Component({
  // selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit,OnDestroy {

  constructor() {
    console.log("constructor work..");
   }

  ngOnInit() {
    console.log("OnInit work..");
  }

  ngOnDestroy(){
    console.log("OnDestroy work..");
  }

}
```

เมื่อเราคลิกมายังหน้า home ในส่วนของ constructor (* constructor ไม่ใช่ lifecycle hook) และ ngOnInit จะทำงาน พอเราคลิกไปหน้าอื่นเช่นหน้า register ngOnDestroy ก็จะทำงานตามรูปแสดงด้านล่าง





เมื่อคลิกไปหน้า register ส่วนของ OnDestroy ในไฟล์ home.component.ts ก็จะทำงาน ก่อนที่จะลบหรือทำลาย HomeComponent

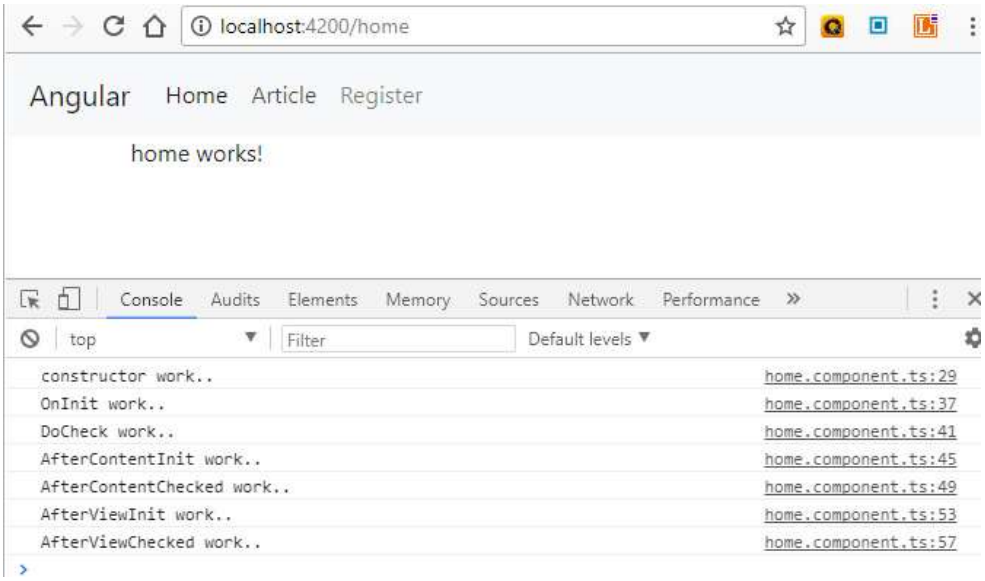


ทีนี้เรามาลอง implement lifecycle hook ทั้งหมดในไฟล์ HomeComponent กันดูดังนี้

ไฟล์ home.componet.ts

```
import {
  Component,
  SimpleChanges,
  OnChanges,
  OnInit,
  DoCheck,
  AfterContentInit,
  AfterContentChecked,
  AfterViewInit,
  AfterViewChecked,
  OnDestroy
} from '@angular/core';

@Component({
  // selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnChanges,
  OnInit,
  DoCheck,
  AfterContentInit,
  AfterContentChecked,
  AfterViewInit,
  AfterViewChecked,
  OnDestroy {

  constructor() {
    console.log("constructor work..");
  }

  ngOnChanges(changes: SimpleChanges) {
    console.log(`OnChanges work..`);
  }

  ngOnInit() {
    console.log("OnInit work..");
  }

  ngDoCheck() {
    console.log("DoCheck work..")
  }

  ngAfterContentInit() {
    console.log("AfterContentInit work..");
  }

  ngAfterContentChecked() {
    console.log("AfterContentChecked work..");
  }

  ngAfterViewInit() {
    console.log("AfterViewInit work..");
  }

  ngAfterViewChecked() {
    console.log("AfterViewChecked work..");
  }

  ngOnDestroy() {
    console.log("OnDestroy work..");
  }

}
```

มาดูการทำงานเมื่อเปิดมาหน้า home ซึ่งมีการใช้งาน HomeComponent จะเห็นการทำงานของ lifecycle hook ตามลำดับดังรูป โดยส่วนของ ngOnChanges และ ngOnDestroy จะไม่ทำงาน ซึ่ง ngOnChanges ไม่ทำงานเพราะยังไม่เข้าเงื่อนไขที่จะถูกเรียกใช้งาน เช่นเดียวกับ ngOnDestroy ที่จะทำงานเมื่อเราคลิกไปหน้าอื่น



Lifecycle hook ที่เกิดขึ้นทั้งหมด เกิดกับ HomeComponent ทั้งที่เกิดกับ HomeComponent โดยตรง หรือเกิดขึ้นกับ Child ที่อยู่ภายใน component ก็ตาม เพื่อให้เห็นการทำงานของ ngDoCheck() , ngAfterContentChecked() และ ngAfterViewChecked() เมื่อมีการตรวจพบการเปลี่ยนแปลงเกิดขึ้นใน HomeComponent เราจะสร้าง input text และ button เข้ามาใน HomeComponent โดยมีการเชื่อมโยงข้อมูลระหว่าง view กับ component ผ่าน keyup และ click event โดยให้ปรับไฟล์ home.component.html เป็นดังนี้

ไฟล์ home.component.html

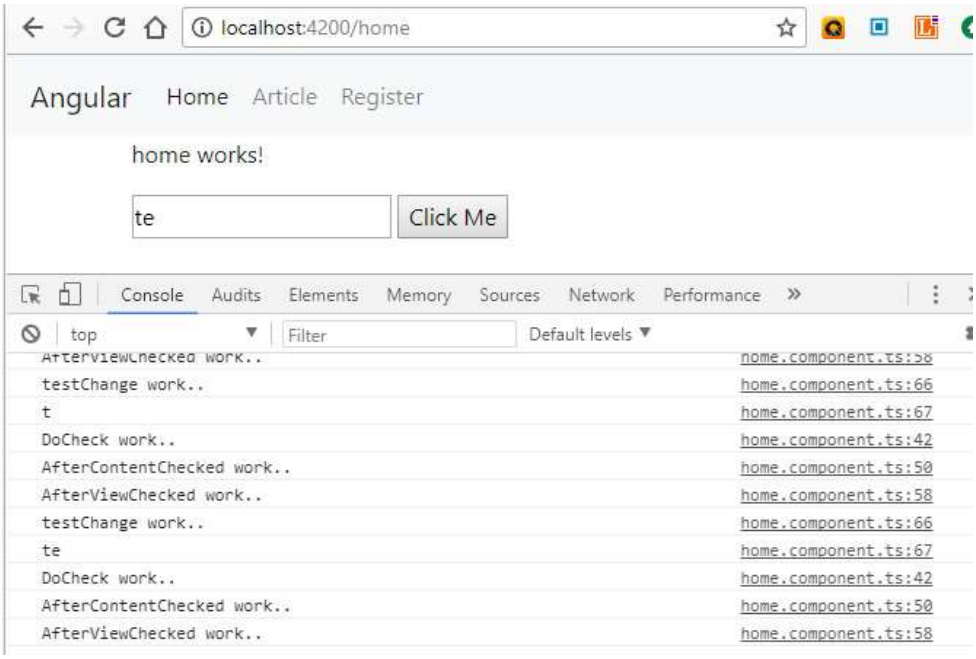
```
<p>  home works!</p>
<input type="text" #mytextbox (keyup)="testChange(mytextbox)" />
<button type="button" (click)="testClick()" >Click Me</button>
```

ส่วนในไฟล์ home.component.ts ให้ปรับเป็นดังนี้

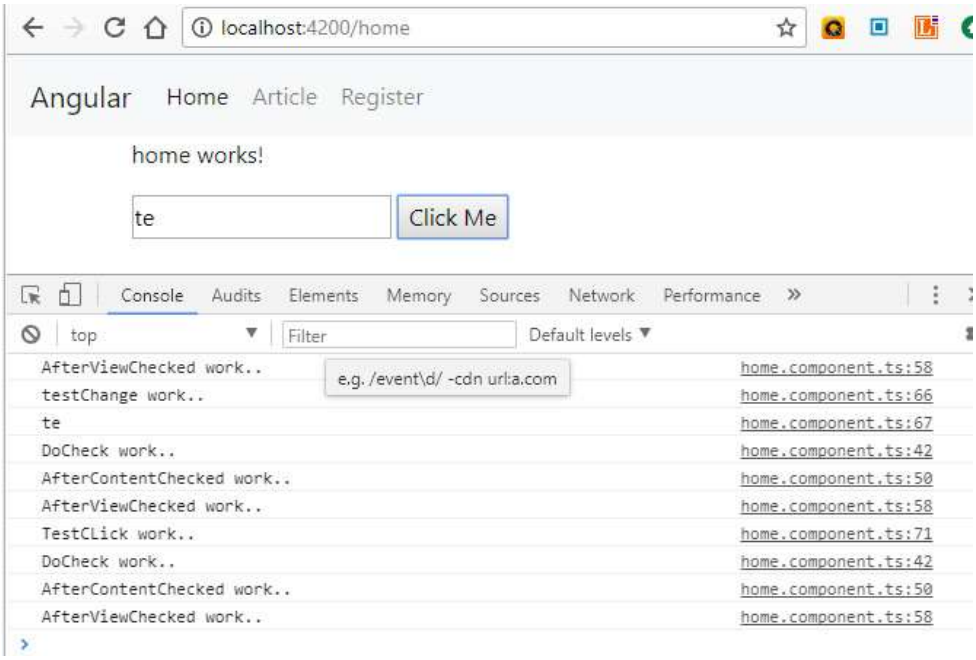
ไฟล์ home.component.ts (ขอยกโค้ดเฉพาะส่วนของ class มาอธิบาย)

```
export class HomeComponent implements OnChanges,
  OnInit,
  DoCheck,
  AfterContentInit,
  AfterContentChecked,
  AfterViewInit,
  AfterViewChecked,
  OnDestroy {

  constructor() {
    console.log("constructor work..");
  }

  ngOnChanges(changes: SimpleChanges) {
    console.log(`OnChanges work..`);
  }

  ngOnInit() {
    console.log("OnInit work..");
  }

  ngDoCheck() {
    console.log("DoCheck work..")
  }

  ngAfterContentInit() {
    console.log("AfterContentInit work..");
  }

  ngAfterContentChecked() {
    console.log("AfterContentChecked work..");
  }

  ngAfterViewInit() {
    console.log("AfterViewInit work..");
  }

  ngAfterViewChecked() {
    console.log("AfterViewChecked work..");
  }

  ngOnDestroy() {
    console.log("OnDestroy work..");
  }

  testChange(myinput){
    console.log("testChange work..");
    console.log(myinput.value);
  }

  testClick(){
    console.log("TestCLick work..");
  }

}
```

จะเห็นว่าเรามีการเพิ่มส่วนของฟังก์ชั่น testChange() และ testClick() เพื่อใช้สำหรับทดสอบ จากนั้นทดสอบกรอกข้อมูล หรือทดสอบคลิกที่ปุ่มที่สร้างมา จะได้ผลลัพธ์ดังรูปด้านล่าง



จะเห็นว่าเราทดสอบพิมพ์คำว่า 'te' ทุกครั้งที่มีการพิมพ์ตัวอักษร จะมีการเปลี่ยนแปลงเกิดขึ้นกับ HomeComponent ทำให้ DoCheck, AfterContentChecked และ AfterViewChecked ถูกเรียกใช้งานเช่นเดียวกัน



เช่นเดียวกัน เมื่อเราคลิกที่ปุ่ม ทำให้มีการทำงานในส่วนของฟังก์ชัน testClick() เกิดขึ้นกับ HomeComponent DoCheck, AfterContentChecked และ AfterViewChecked ก็ถูกเรียกใช้งานเช่นเดียวกัน

ตอนนี้เรายังไม่เห็น ngOnChanges() ถูกเรียกใช้งาน ทั้งนี้ก็เพราะเงื่อนไขการถูกเรียกใช้งานของ ngOnChanges() นั้น จะเกิดขึ้นเมื่อมีการเปลี่ยนแปลงของการเชื่อมโยงข้อมูลผ่าน input property (@Input) เกี่ยวกับ input property เพิ่มเติมสามารถดูได้เนื้อหาตามลิ้งค์ด้านล่าง

Input Output และ ตัวดำเนินการนิพจน์ของ template ใน Angular http://niik.in/789
http://www.ninenik.com/content.php?arti_id=789 (http://www.ninenik.com/content.php?arti_id=789) via @ninenik

เนื่องจาก HomeComponent ของเราไม่มีการใช้งาน input property จึงไม่มีการเรียกใช้ ngOnChanges()
เพื่อให้เห็นการทำงานของ ngOnChanges() เราจะทำการสร้าง attribute directive ขึ้นมาใช้งาน ด้วยคำสั่งดังนี้

```
C:\projects\httpclient>ng g directive /home/highlight
```

สามารถดูเพิ่มเติมเกี่ยวกับการสร้าง Attribute Directive ได้ที่ลิ้งค์ด้านล่าง
การสร้าง Attribute Directive สำหรับใช้งาน ใน Angular http://niik.in/791
http://www.ninenik.com/content.php?arti_id=791 (http://www.ninenik.com/content.php?arti_id=791) via @ninenik

จากนั้นแก้ไขไฟล์ highlight.directive.ts เป็นดังนี้

ไฟล์ highlight.directive.ts

```
import { Directive, ElementRef, HostListener, Input, SimpleChanges } from '@angular/core';

@Directive({
    selector: '[myHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef) { }

  @Input() defaultColor: string;
  @Input('myHighlight') highlightColor: string;

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight(this.highlightColor || this.defaultColor || 'red');
  }
  @HostListener('mouseleave') onMouseLeave() {
    this.highlight(null);
  }
  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }

  ngOnChanges(changes: SimpleChanges) {
    console.log(`OnChanges work..`);
    console.log(changes);
  }

  ngOnInit() {
    console.log("OnInit work..");
  }

  ngDoCheck() {
    console.log("DoCheck work..")
  }

  ngOnDestroy() {
    console.log("OnDestroy work..");
  }

}
```

อย่างที่เราทราบไปแล้วว่า directive และ component นั้นจะมี lifecycle hook คล้ายๆ กันแต่ไม่เหมือนกันเสียทีเดียว โดยใน directive จะไม่มีในส่วนของ content hook ตามตัวอย่าง attribute directive ด้านบน
Attribute directive ข้างบนนี้มีการใช้งาน input property ทำให้เราสามารถทดสอบ ngOnChange() ได้ highlightColor คือ input property โดย attribute นี้จะทำหน้าที่เปลี่ยนสีพื้นหลังเมื่อเลื่อนเมาส์ไปอยู่เหนือ element ที่กำหนด หรือเลื่อนเมาส์ออกจาก element ที่กำหนด มีสีพื้นหลังค่าเริ่มต้นเป็นสีแดง

ต่อไปให้เราปรับไฟล์ home.component.html เพื่อใช้งาน attribute directive ที่เราสร้างขึ้นกับ input text ดังนี้

ไฟล์ home.component.html

```
<p>  home works!</p>
<input [myHighlight]="color" type="text"  />
<button type="button" (click)="testClick()" >Click Me</button>
```

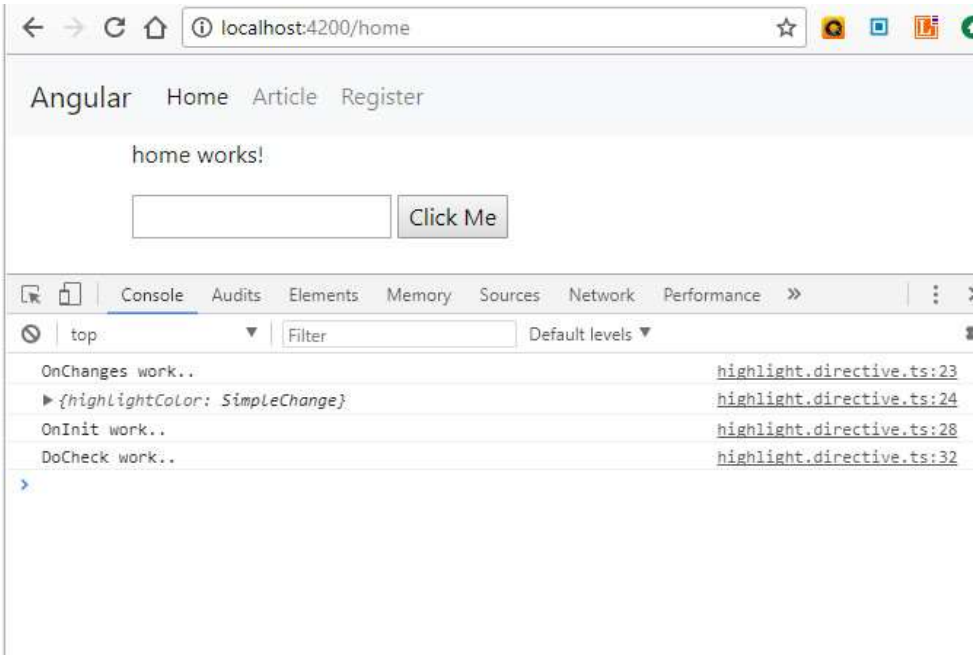
จากนั้นแก้ไขไฟล์ home.component.ts โดยเราจะเหลือไว้แค่ onInit() และฟังก์ชั่นสำหรับเปลี่ยนสลับสีของพื้นหลัง input text ที่มีการใช้งาน attribute directive ที่ชื่อ [myHighlight] เมื่อมีการคลิกที่ปุ่ม

ไฟล์ home.component.ts

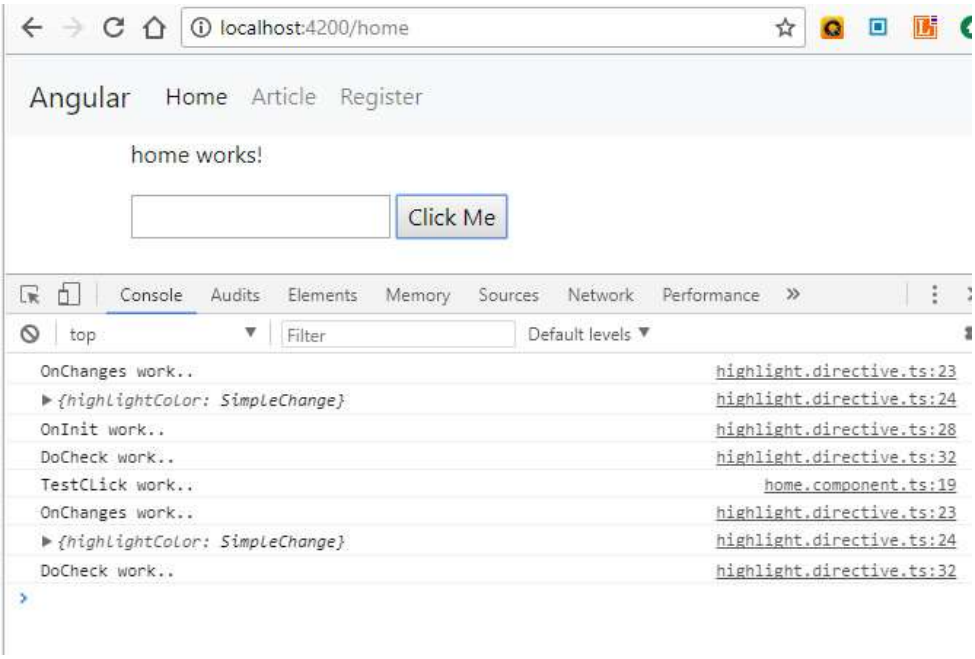
```
import { Component, OnInit } from '@angular/core';

@Component({
  // selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  private color:string;

  constructor() {  }

  ngOnInit() {

  }

  testClick(){
    console.log("TestCLick work..");
    this.color = this.color=='red'?'yellow':'red';
  }

}
```

เมื่อเราคลิกที่ปุ่ม จะทำให้ค่า color เปลี่ยนสลับไปมาระหว่างสีแดงและสีเหลือง การที่ค่าของ color เปลี่ยนทำให้ตัว highlightColor ซึ่ง input property ที่ใช้ตัวแปรชื่อเรียกแทนเป็น myHighlight มีการเปลี่ยนแปลงเกิดขึ้น ส่งผลให้เกิดการเรียกใช้ ngOnChange()

ดูผลลัพธ์จากรูปด้านล่าง ดังนี้

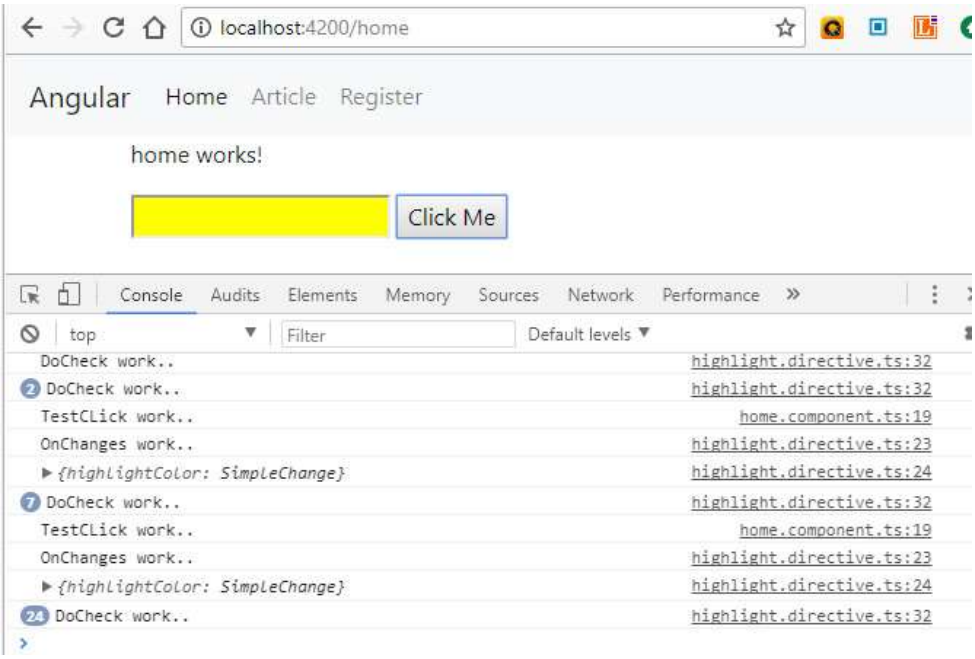


จะเห็นว่าเมื่อเราคลิกมาหน้า home ซึ่งมีการใช้งาน attribute directive ใน input text ทำให้ lifecycle hook ของ HighlightDirective ทำงาน โดยเริ่มจาก OnChange, OnInit และ DoCheck ตามลำดับ

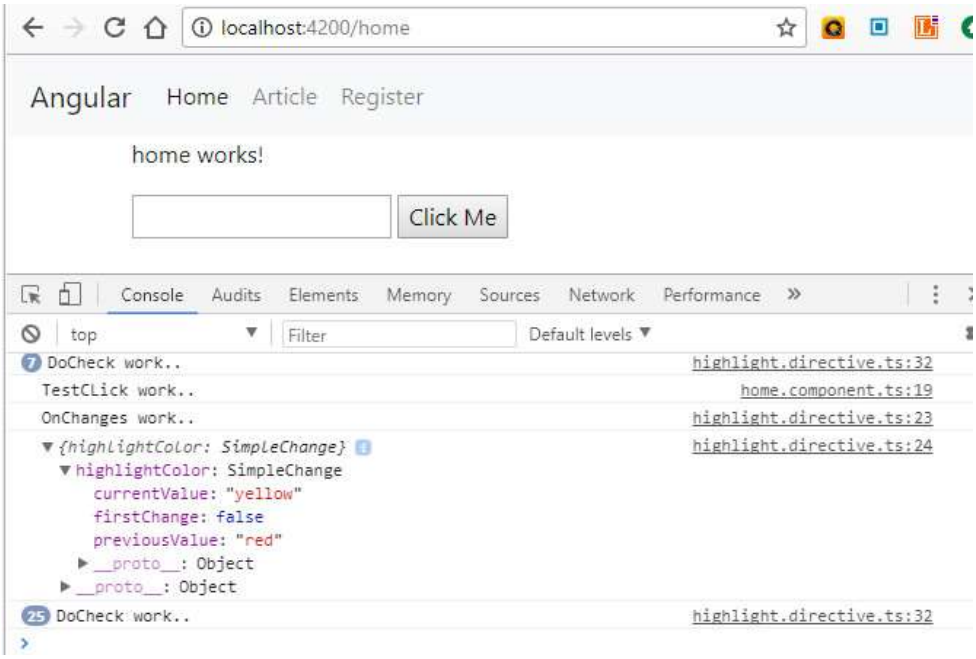


พอเราคลิกที่ปุ่ม ซึ่งเราได้กำหนดให้เปลี่ยนสลับสีพื้นหลังของ input text ก็จะทำให้ค่า highlightColor ซึ่ง input property เปลี่ยนแปลง OnChange จึงถูกเรียกใช้งาน ตามรูปด้านบน

นอกจากนั้นการเปลี่ยนแปลงค่าของ highlightColor นั้นเกิดขึ้นจากการเลื่อนเมาส์ไว้เหนือ และเลื่อนเมาส์ออกจาก input text ด้วย ลองดูหลังจากเรากดปุ่มแล้ว สีพื้นหลังสลับเป็นสีเหลือง เราลองเอาเมาส์ไว้วางอยู่เหนือ จะได้ผลลัพธ์ตามรูปด้านล่าง



ใน OnChange จะมีการส่งค่า SimpleChanges object ที่มีค่าปัจจุบันของ input property และค่าเดิมก่อนหน้าของ input property เพื่อใช้งานด้วย ดูค่าผ่าน console ตามรูปด้านล่าง



จะเห็นว่าสีพื้นหลังปัจจุบันเป็นสีเหลือง และสีก่อนหน้าเป็นสีแดง ทุกครั้งที่มีการเปลี่ยนแปลง ค่าตรงนี้ก็จะสลับไปสลับมา ตามรูปแบบหรือค่าที่เรากำหนดส่งเข้ามาใช้งาน

ตอนนี้เราได้รู้จักการใช้งาน lifecycle hook ไปบ้างแล้ว ในส่วนของการใช้งานจริงๆ นั้น จะขึ้นอยู่เราเองว่า เราจะหยิบส่วนไหนมาใช้ และใช้วัตถุประสงค์ใด lifecycle hook เป็นส่วนที่ทำให้เราสามารถทำคำสั่งที่ต้องการแทรกเข้ามา เพื่อทำหน้าที่บางอย่างเฉพาะได้

สำหรับเนื้อหาในตอนหน้าจะเป็นเกี่ยวกับอะไร รอติดตาม

เนื้อหาที่เกี่ยวข้อง

10 APR 2017 Input Output และ ตัวดำเนินการนิพจน์ของ template ใน Angular (https://www.ninenik.com/Input_Output_และ_ตัวดำเนินการนิพจน์ของ_template_ใน_Angular-789.html) **อ่าน 1,091**
เนื้อหาต่อไปนี้เป็นตอนสุดท้ายของบทความเกี่ยวกับ template syntax เบื้องต้น ซึ่

13 APR 2017 การสร้าง Attribute Directive สำหรับใช้งาน ใน Angular (https://www.ninenik.com/การสร้าง_Attribute_Directive_สำหรับใช้งาน_ใน_Angular-791.html) **อ่าน 844**
Attribute directive จะใช้สำหรับผูกพฤติการณ์หรือการทำงานต่างๆ เข้าไปใน element

Tags:: lifecycle hook (tag_คำสั่ง_วิธีการ_ใช้งาน_การประยุกต์_ตัวอย่าง_เกี่ยวกับ-lifecycle_hook) angular (tag_คำสั่ง_วิธีการ_ใช้งาน_การประยุกต์_ตัวอย่าง_เกี่ยวกับ-angular)

URL สำหรับอ้างอิง Save to Facebook http://niik.in/858 Top Copy

ทำความรู้จักกับ และใช้งาน Lifecycle Hook ใน Angular http://niik.in/858
http://www.ninenik.com/content.php?arti_id=858 via @ninenik

Author Ninenik Narkdee (https://plus.google.com/u/0/107577507182658847616?rel=author)