هوک های چرخه‌ی عمر

کامپوننت ها چرخه‌ی عمری دارند که توسط Angular مدیریت می‌شوند.

Angular این کامپوننت ها را ایجاد و رندر می‌کند، فرزندان آن را ایجاد و رندر می‌کند، زمانی که ویژگی‌های مقید به داده‌ی آن تغییر می‌کنند، این کامپوننت ها را بررسی می‌کند و قبل از حذف کردن آن‌ها از DOM، آن‌ها را از بین می‌برد.

Angular هوک های چرخه‌ی عمری را ارائه می‌کند که با کمک آن‌ها می‌توان لحظات کلیدی چرخه‌ی عمر را دید و در این لحظات کاری انجام داد.

دستورالعمل‌ها دارای همین مجموعه از هوک های چرخه‌ی عمر هستند.

مرور کلی هوک های چرخه‌ی عمر کامپوننت

نمونه‌های کامپوننت و دستورالعمل دارای چرخه‌ی عمری هستند؛ زیرا Angular آن‌ها را با ایجاد، به روز و نابود می‌کند. برنامه نویسان می‌توانند با اجرا کردن یک یا چند رابط هوک چرخه‌ی عمر در کتابخانه‌ی Angular core این لحظات مهم را مشاهده کنند.

هر یک از این رابط‌ها دارای تنها یک متد هوک است که اسم این متدها همان اسم رابط است که یک ng قبل از آن قرار گرفته است. برای مثال رابط OnInit دارای متد هوکی به نام ngOnInit() است که Angular اندکی بعد از ایجاد کردن کامپوننت آن را فراخوانی می‌کند.

```
peek-a-boo.component.ts (excerpt)
export class PeekABoo implements OnInit {
 constructor(private logger: LoggerService) { }

// implement OnInit's `ngOnInit` method
 ngOnInit() { this.logIt(`OnInit`); }
 logIt(msg: string) {
 this.logger.log(`#${nextId++} ${msg}`);
 }
}
```

هیچ دستورالعمل یا کامپوننتی وجود ندارد که بتواند تمامی هوک های چرخه‌ی عمر را اجرا کند. Angular تنها متد هوک کامپوننت یا دستورالعملی را فراخوانی می‌کند که تعریف شده باشد.

ترتیب چرخه‌ی عمر

بعد از ایجاد یک دستورالعمل یا کامپوننت توسط فراخوانی کردن constructor آن، Angular متدهای هوک چرخه‌ی عمر را به ترتیب زیر و در لحظات مشخصی فراخوانی می‌کند.

هوک	هدف و زمان بندی
ngOnChanges()	زمانی که Angular ویژگی‌های ورودی مقید به داده را (مجدداً) تنظیم می‌کند، واکنش نشان می‌دهد. این متد شیء SimpleChanges مقادیر ویژگی قبلی و فعلی را دریافت می‌کند. قبل از ngOnInit() و هر زمان که یک یا چند ویژگی ورودی مقید به داده تغییر کنند، فراخوانی می‌شود.
ngOnInit()	بعد از آن که Angular ویژگی‌های مقید به داده را نمایش می‌دهد و ویژگی‌های ورودی کامپوننت یا دستورالعمل را تنظیم می‌کند، این دستورالعمل یا کامپوننت را مقداردهی اولیه می‌کند. یک بار و بعد از اولین ngOnChanges() فراخوانی می‌شود.
ngDoCheck()	تغییراتی که Angular نتواند و یا نخواهد شناسایی کند را شناسایی می‌کند و در قبال آن کاری انجام می‌دهد. طی هربار تغییر شناسایی و بلافاصله بعد از ngOnChanges() و ngOnInit() فراخوانی می‌شود.

ngAfterContentInit()	بعد از محتوای خارجی پروژه‌های Angular موجود در view کامپوننت یا view ای که دستورالعملی در آن قرار دارد، واکنش نشان می‌دهد. یک بار و بعد از اولین ngDoCheck() فراخوانی می‌شود.
ngAfterContentChecked()	بعد از آن که Angular محتوای طرح ریزی شده داخل کامپوننت یا دستورالعمل را بررسی می‌کند، واکنش نشان می‌دهد. بعد از ngAfterContentInit() و تمامی ngDoCheck() های بعد از آن، فراخوانی می‌شود.
ngAfterViewInit()	بعد از آن که view Angular های کامپوننت و view های فرزند یا view ای که دستورالعملی در آن قرار دارد را مقداردهی اولیه می‌کند، واکنش نشان می‌دهد. یک بار بعد از اولین ngAfterContentChecked() فراخوانی می‌شود.
ngAfterViewChecked()	بعد از آن که view Angular ها و view های فرزند کامپوننت یا view ای که دستورالعملی در آن قرار دارد را بررسی می‌کند، واکنش نشان می‌دهد. بعد از ngAfterViewInit و تمام ngAfterContentChecked() های پس از آن فراخوانی می‌شود.
ngOnDestroy()	درست قبل از آن که Angular دستورالعمل یا کامپوننت مشخص را از بین ببرد، کار پاک سازی را انجام می‌دهد. جهت جلوگیری از نشتی حافظه observable ها را از حالت اشتراک خارج کنید و تمامی event handler ها را جدا کنید. درست قبل از آن که Angular دستورالعمل یا کامپوننت را از بین ببرد، فراخوانی می‌شود.

رابط‌ها (از نظر فنی) اختیاری هستند

اگر صرفاً از نظر تکنیکال به قضیه نگاه کنیم، برنامه نویسان تایپ اسکریپت و جاوا اسکریپت مجبور به استفاده از رابط‌ها نیستند. زبان جاوا اسکریپت رابط ندارد. Angular در زمان اجرا نمی‌تواند رابط‌های تایپ اسکریپت را مشاهده کند، زیرا این رابط‌ها از جاوا اسکریپت ترنسپایل شده ناپدید می‌شوند.

خوشبختانه لزومی برای استفاده از رابط‌ها وجود ندارد. برای استفاده از هوک ها شما مجبور به افزودن رابط‌های هوک چرخه‌ی عمر به دستورالعمل‌ها و کامپوننت ها نیستید.

در عوض Angular کلاس‌های کامپوننت و دستورالعمل را زیر نظر می‌گیرد و متدهای هوک را در صورتی که تعریف شده باشند، فراخوانی می‌کند. Angular چه با رابط‌ها و چه بدون آن‌ها متدهایی مانند ngOnInit() را پیدا می‌کند و فراخوانی می‌کند.

با این وجود، برای آن که بتوان از قابلیت‌های strong typing و editor tooling استفاده کرد، بهتر است رابط‌ها را به کلاس‌های دستورالعمل تایپ اسکریپت اضافه کرد.

هوک های چرخه‌ی عمر دیگر در Angular

به غیر از هوک های کامپوننت بیان شده، زیرسیستم‌های دیگر Angular ممکن است هوک های چرخه‌ی عمر مخصوص به خود را داشته باشند.

کتابخانه‌های سوم شخص ممکن است برای آن که این امکان را به برنامه نویسان بدهند تا بر روی چگونگی استفاده از این کتابخانه‌ها کنترل بیشتری داشته باشند، این هوک ها را به کار می‌گیرند.

مثال‌هایی از چرخه‌ی عمر

در این لینک از طریق مجموعه‌ای از تمرینات که به صورت کامپوننت های تحت کنترل AppComponent ریشه‌ای ارائه شده‌اند به تبیین هوک های چرخه‌ی عمر پرداخته شده است.

این تمرینات از الگوی مشترکی پیروی می‌کنند؛ یک کامپوننت مادر که برای یک کامپوننت فرزند نقش تجهیزات آزمایشی را ایفا می‌کند و یک یا چند متد هوک چرخه‌ی عمر را تبیین می‌کند.

توضیح مختصری از هر یک از این تمرینات را می‌توانید در زیر مشاهده کنید:

کامپوننت	توضیحات
Peek-a-boo	تمامی هوک های چرخه‌ی عمر را نمایش می‌دهد. هر یک از متدهای هوک در یک گزارش on-screen نوشته می‌شوند.
Spy	دستورالعمل‌ها نیز دارای هوک های چرخه‌ی عمر هستند. یک SpyDirective می‌تواند زمانی گزارشات را ثبت کند که عنصری که جاسوسی‌اش را می‌کند با استفاده از هوک های ngOnInit و ngOnDestroy ایجاد یا نابود شود. در این مثال، در یک و داخل یک حلقه‌ی تکرار ngFor هیرو از SpyDirective استفاده شده است و توسط SpyComponent مادر مدیریت می‌شود.
OnChanges	دقت کنید که هر بار که یکی از ویژگی‌های ورودی کامپوننت تغییر می‌کند، Angular چگونه هوک ngOnChanges() را به همراه شیء changes فراخوانی می‌کند.
DoCheck	یک متد ngDoCheck() را به همراه شناسایی اختصاصی تغییر به کار می‌گیرد. دقت کنید که Angular چند بار یک بار این هوک را فراخوانی می‌کند و ثبت تغییرات آن را زیر نظر بگیرید.
AfterView	نشان می‌دهد که منظور Angular از یک view چیست. هوک های ngAfterViewInit و ngAfterViewChecked را نمایش می‌دهد.
AfterContent	چگونگی طرح ریزی محتواهای خارجی را داخل یک کامپوننت و چگونگی تشخیص تفاوت بین محتوای طرح ریزی شده از یک view فرزند کامپوننت نمایش می‌دهد. هوک های ngAfterContentInit و ngAfterContentChecked را نمایش می‌دهد.
Counter	ترکیبی از یک کامپوننت و یک دستورالعمل را نمایش می‌دهد؛ به گونه‌ای که هریک دارای هوک مخصوص به خود هستند. در این مثال هربار که کامپوننت مادر ویژگی شمارش گر ورودی خود را یک واحد افزایش می‌دهد، یک CounterComponent تغییری را (از طریق ngOnChanges) ثبت می‌کند. در همین حال SpyDirective بیان شده در مثال قبل در ثبت وقایع CounterComponent به کار گرفته می‌شود و در همین مکان گزارش مربوط به ورودی‌هایی که ایجاد و نابود می‌شوند را زیر نظر می‌گیرد.

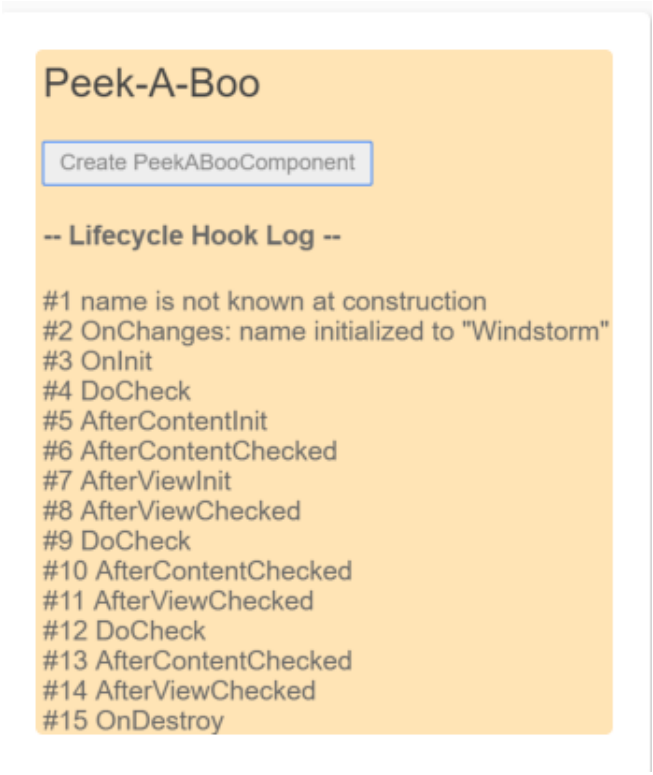
در بخش‌های باقی مانده‌ی این صفحه به صورت دقیق‌تر به این تمرینات می‌پردازیم.

Peek-a-boo: تمام هوک ها

PeekABooComponent تمامی هوک ها را در یک کامپوننت نمایش می‌دهد.

به ندرت پیش می‌آید (اگر پیش بیاید) کل رابط را مانند الان به کار بگیرید. هدف از وجود peek-a-boo نشان دادن چگونگی فراخوانی هوک ها توسط Angular و با ترتیب مورد انتظار است.

در تصویر زیر حالت گزارش را بعد از آن که کاربر بر روی دکمه‌ی *Create* و پس از آن *Destroy* کلیک کرده است، نشان می‌دهد.



ترتیب پیام‌های گزارش از ترتیب فراخوانی تعیین شده‌ی هوک ها پیروی می‌کند:

OnChanges ، OnInit ، DoCheck (3 مرتبه)، AfterContentInit، AfterContentChecked (3 مرتبه)، AfterViewInit، AfterViewChecked (3 مرتبه) و OnDestroy.

سازنده (constructor) در Angular به خودی خود یک هوک نیست. در زمان ساخت گزارش تأیید می‌کند که ویژگی‌های ورودی (برای حالت ما ویژگی name) هیچ مقدار تخصیص داده شده‌ای ندارند.

با کلیک کاربر بر روی دکمه‌ی *Update Hero* گزارش OnChanges و دو AfterContentChecked ، AfterViewChecked و DoCheck سه مرتبه‌ای بیشتر را نمایش می‌دهد. به وضوح مشخص است که اغلب این سه هوک fire می‌شوند. منطق به کار رفته در این هوک ها را تا حد امکان مختصر و مفید نگه دارید.

در مثال‌های بعدی بر جزئیات هوک ها تمرکز می‌شود.

جاسوسی کردن OnInit و OnDestroy

به کمک این دو هوک جاسوسی به صورت مخفی زمان نابودی یا به وجود آمدن یک عنصر را کشف کنید.

این شیوه برای یک دستورالعمل بهترین کار نفوذی است؛ زیرا هیچ وقت هیروها متوجه تحت نظر بودن خود نمی‌شوند.

از شوخی گذشته، به این دو نکته‌ی مهم توجه کنید:

1. Angular هم برای دستورالعمل‌ها و هم برای کامپوننت ها، متدهای هوک را فراخوانی می‌کنند.
2. یک دستورالعمل جاسوسی می‌تواند درباره‌ی یک شیء DOM اطلاعاتی را فراهم کند که شما نمی‌توانید آن را به صورت مستقیم تغییر دهید. بدیهی است که شما نمی‌توانید پیاده سازی یک بومی را دستکاری کنید. یک کامپوننت سوم شخص را هم نمی‌توانید تغییر دهید؛ اما این کار را می‌توانید انجام دهید که هر دوی آن‌ها را به کمک یک دستورالعمل زیر نظر بگیرید.

دستورالعمل جاسوسی آب زیر کاه پیچیدگی خاصی ندارد. این دستورالعمل تقریباً تمامی هوک های ngOnInit() و ngOnDestroy() که پیام‌هایی را از طریق یک LoggerService تزریق شده به parent گزارش می‌دهند را شامل می‌شود.

src/app/spy.directive.ts

```
// Spy on any element to which it is applied.
// Usage: <div mySpy>...</div>
@Directive({selector: '[mySpy]'})
export class SpyDirective implements OnInit, OnDestroy {

 constructor(private logger: LoggerService) { }

 ngOnInit() { this.logIt(`onInit`); }
```

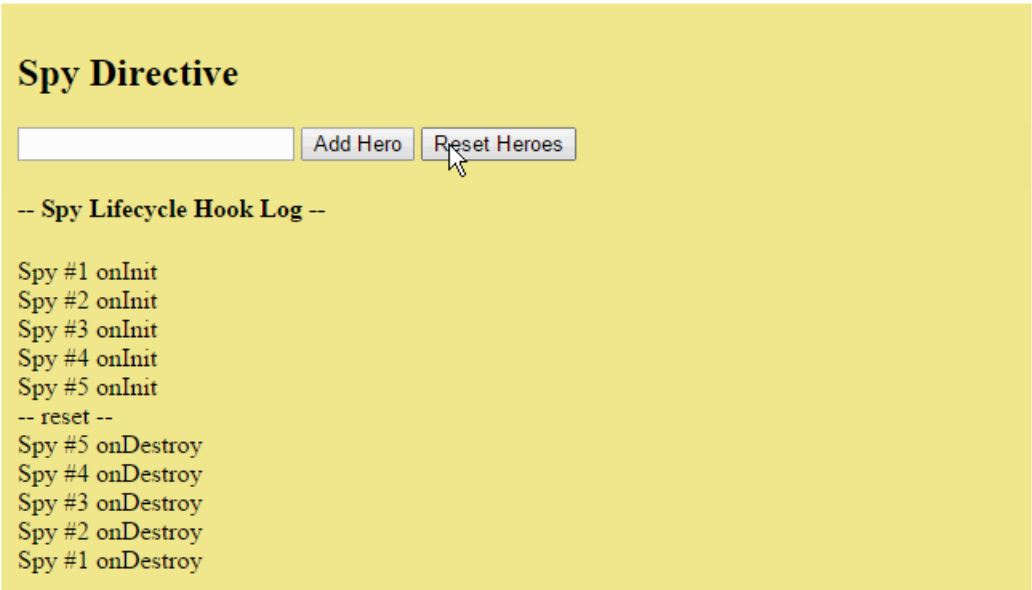
```
ngOnDestroy() { this.logIt(`onDestroy`); }


private logIt(msg: string) {
this.logger.log(`Spy #${nextId++} ${msg}`);
}
}
```

این جاسوس را می‌توانید در تمامی عناصر کامپوننت یا بومی به کار ببرید. بعد از انجام این کار، این جاسوس هم زمان با این عنصر ایجاد و نابود می‌شود. در زیر می‌توانید مقید شدن آن به یک پی در پی هیرو را مشاهده کنید:

src/app/spy.component.html

```
<div *ngFor="let hero of heroes" mySpy class="heroes">
{{hero}}
</div>
```

مرگ و تولد هر یک از جاسوس‌ها مشخص کننده‌ی مرگ و تولد هیروی متصل شده به همراه یک ورودی موجود در Hook Log زیر است:



اضافه کردن یک هیرو منجر به یک هیروی جدید می‌شود. ngOnInit() جاسوس این رویداد را ثبت می‌کند.

دکمه‌ی ریست لیست heroes را پاک می‌کند. Angular تمامی عناصر هیرو از DOM را حذف می‌کند و به صورت همزمان دستورالعمل‌های جاسوس آن‌ها را نابود می‌کند. متد ngOnDestroy() جاسوس، لحظات آخر آن را گزارش می‌دهد.

نقش اصلی و مهم‌تر متدهای ngOnInit() و ngOnDestroy() در برنامه‌های واقعی بیشتر مشخص می‌شود.

OnInit()

به دو دلیل اصلی زیر می‌توان از ngOnInit() استفاده کرد:

1. برای انجام مقداردهی‌های اولیه‌ی پیچیده کمی بعد از فرآیند ساخت.
2. برای راه اندازی کامپوننت بعد از آن که Angular ویژگی‌های ورودی را تنظیم می‌کند.

برنامه نویسان با تجربه همه بر سر این موضوع توافق دارند که ساخت کامپوننت ها باید ارزان و امن صورت بگیرد.

میسکو هوری، رهبر گروه Angular در اینجا توضیح می‌دهد که چرا نباید منطق سازنده‌ها پیچیده باشد.

در سازنده‌ی یک کامپوننت داده‌ها را دریافت نکنید. نیازی نیست بابت ارتباط یک کامپوننت با سرور از راه دور در زمانی که این کامپوننت تحت آزمایش ایجاد می‌شود و یا قبل از آن که شما بخواهید آن را نمایش دهید، نگران باشید. تنها کاری که سازنده‌ها باید انجام دهند، این است که متغیرهای محلی اولیه را بر روی مقادیر ساده تنظیم کنند.

یک ngOnInit() بهترین مکان برای یک کامپوننت است تا بتواند داده‌های اولیه‌ی خود را دریافت کند. در آموزش « Tour of Heroes» چگونگی انجام این کار نشان داده شده است.

همچنین به خاطر داشته باشید که تا بعد از فرآیند ساخت، ویژگی‌های ورودی مقید به داده‌ی یک دستورالعمل تنظیم نمی‌شوند. اگر نیاز داشته باشید که بر اساس این ویژگی‌ها این دستورالعمل را مقداردهی اولیه کنید، به مشکل بر می‌خورید. این ویژگی‌ها زمانی که ngOnInit() اجرا شود، تنظیم خواهند شد.

متد ngOnChanges() اولین فرصت برای دسترسی به این ویژگی‌ها است. Angular قبل و چندین بار بعد از ngOnInit() ، ngOnChanges() را فراخوانی می‌کند. Angular تنها یک بار ngOnInit() را فراخوانی می‌کند.

بعد از ایجاد کامپوننت برای فراخوانی ngOnInit() می‌توانید بر روی Angular حساب باز کنید. این نقطه همان جایی است که منطق مقداردهی اولیه‌ی سنگینی به آن تعلق دارد.

OnDestroy()

منطق پاک سازی را درون ngOnDestroy() قرار دهید. این منطق، منطقی است که باید قبل از آن که Angular دستورالعمل را از بین ببرد، اجرا شود.

در این زمان است که باید به بخش دیگر برنامه اطلاع داد که کامپوننت در حال از بین رفتن است.

اینجا همان مکانی است که باید منابعی که به طور خودکار به صورت زباله‌ای گردآوری نمی‌شوند را رها کرد. اشتراک خود را از observable ها و رویدادهای DOM باطل کنید. زمان سنج‌های فاصله‌ای را متوقف کنید. تمامی callback هایی که این دستورالعمل به کمک سرویس‌های برنامه یا سراسری ثبت کرده است را از حالت ثبت شده خارج کنید. اگر این کار را نکنید ممکن است با خطر نشت حافظه مواجه شوید.

OnChanges()

Angular هر زمان که تغییرات اعمال شده بر ویژگی‌های ورودی کامپوننت (دستورالعمل) را شناسایی کند، متد ngOnChanges() خود را فراخوانی می‌کند. مثال زیر بر هوک OnChanges نظارت می‌کند.

```
on-changes.component.ts (excerpt)
ngOnChanges(changes: SimpleChanges) {
 for (let propName in changes) {
 let chng = changes[propName];
 let cur = JSON.stringify(chng.currentValue);
 let prev = JSON.stringify(chng.previousValue);
 this.changeLog.push(`${propName}: currentValue = ${cur}, previousValue = ${prev}`);
 }
}
```

متد ngOnChanges() شیئی را می‌گیرد که هر یک از اسامی ویژگی‌های تغییر یافته را در یک شیء SimpleChange نگاشت می‌کند؛ به گونه‌ای که این شیء مقادیر ویژگی قبلی و فعلی را نگهداری می‌کند. این هوک ویژگی‌های تغییر یافته را تکرار می‌کند و گزارش آن‌ها را ثبت می‌کند.

در نمونه کامپوننت OnChangesComponent زیر دو ویژگی ورودی وجود دارد: hero و power.

src/app/on-changes.component.ts

```
@Input() hero: Hero;
@Input() power: string;
```

OnChangesParentComponent میزبان مانند زیر به آن‌ها مقید می‌شود:

src/app/on-changes-parent.component.html

```
<on-changes [hero]="hero" [power]="power"></on-changes>
```

این نمونه را می‌توانید هم زمان با اعمال تغییرات توسط کاربر، در عمل مشاهده کنید.

ورودی‌های این گزارش به صورت مقدار رشته‌ای تغییرات ویژگی power ظاهر می‌شوند؛ اما ngOnChanges تغییرات اعمال شده بر hero.name که در ابتدا تعجب آور هستند را نمی‌گیرد.

Angular تنها زمانی هوک را فراخوانی می‌کند که مقدار ویژگی ورودی تغییر کند. مقدار ویژگی hero مرجع شیء hero است. برای Angular اهمیتی ندارد که ویژگی name خود هیرو تغییر کرده است. از زاویه‌ی دید Angular، مرجع شیء hero تغییر نکرده است؛ به همین دلیل چیزی برای Angular وجود ندارد که گزارش کند.

DoCheck()

برای شناسایی تغییراتی که Angular به خودی خود آن‌ها را درک نمی‌کند، همچنین برای انجام کارهای لازم در قبال این تغییرات، از هوک DoCheck استفاده کنید.

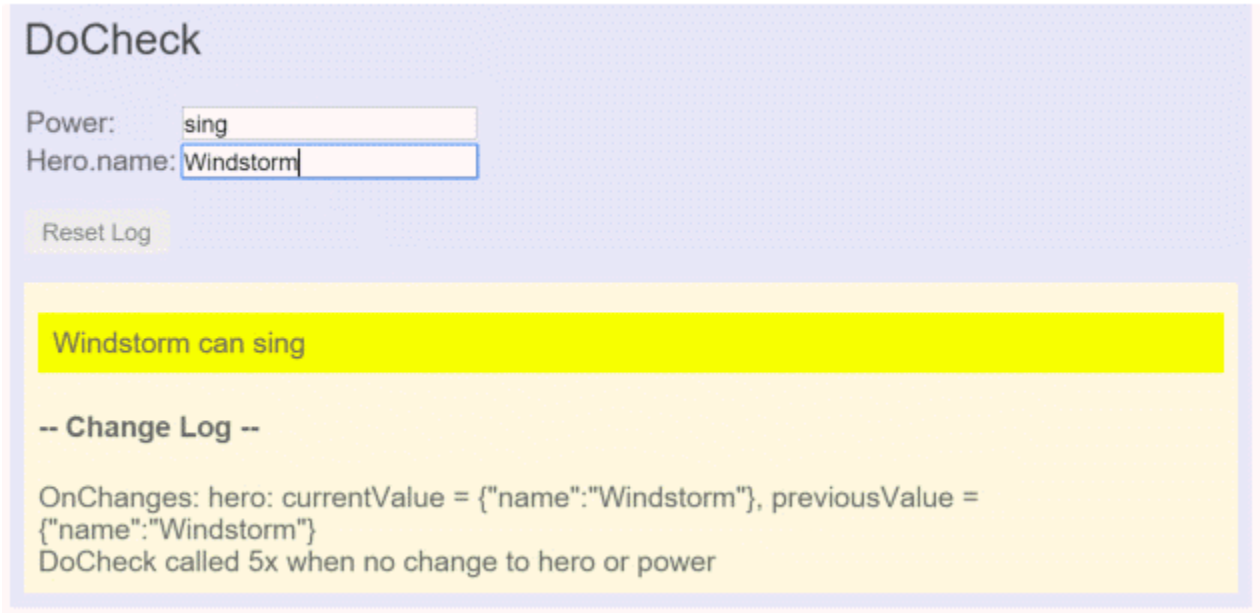
از این متد برای شناسایی تغییری استفاده کنید که Angular از آن چشم پوشی کرده است.

در نمونه‌ی DoCheck ، نمونه‌ی OnChanges به همراه هوک ngDoCheck() بسط داده شده است:

```
DoCheckComponent (ngDoCheck)
ngDoCheck() {

 if (this.hero.name!== this.oldHeroName) {
 this.changeDetected = true;
 this.changeLog.push(`DoCheck: Hero name changed to "${this.hero.name}" from "${this.oldHeroName}"`);
 this.oldHeroName = this.hero.name;
 }

 if (this.power!== this.oldPower) {
 this.changeDetected = true;
 this.changeLog.push(`DoCheck: Power changed to "${this.power}" from "${this.oldPower}"`);
 this.oldPower = this.power;
 }

 if (this.changeDetected) {
 this.noChangeCount = 0;
 } else {
// log that hook was called when there was no relevant change.
 let count = this.noChangeCount += 1;
 let noChangeMsg = `DoCheck called ${count}x when no change to hero or power`;
 if (count === 1) {
// add new "no change" message
```

```
this.changeLog.push(noChangeMsg);
} else {
// update last "no change" message
this.changeLog[this.changeLog.length - 1] = noChangeMsg;
}
}

this.changeDetected = false;
}
```

این کد مقادیر خاص مورد نظرتان را زیر نظر می‌گیرد و حالت فعلی آن‌ها را ضبط می‌کند و آن را با مقادیر قبلی مقایسه می‌کند. این کد زمانی که در hero و power تغییر خاصی اعمال نشده است، پیام ویژه‌ای را در بخش گزارش می‌نویسد تا شما بتوانید تعداد دفعات فراخوانی DoCheck را ببینید. نتایج در زیر مشهود هستند:



با وجود اینکه هوک ngDoCheck() می‌تواند زمان تغییر name هیرو را تشخیص دهد، اما این کار بهای وحشتناکی دارد. این هوک به دفعات مکرر بعد از هر چرخه از تشخیص تغییرات و صرف نظر از محل رخ داد این تغییرات فراخوانی می‌شود. در این مثال قبل از آن که کاربر بتواند کاری انجام دهد، این هوک بیش از 20 مرتبه فراخوانی می‌شود.

بسیاری از این بررسی‌های اولیه توسط Angular، زمانی فعال می‌شوند که Angular برای بار اول داده‌های نامرتبط را در جای دیگری از صفحه رندر می‌کند. صرفاً حرکت موس به سمت <input> دیگر باعث فعال شدن فرآیند فراخوانی می‌شود.

نسبتاً کمتر فراخوانی را می‌توان یافت که تغییرات واقعی اعمال شده بر داده‌های مرتبط را آشکار کنند. بدون شک پیاده سازی ما باید بسیار سبک باشد، در غیر این صورت تجربه‌ی کاربری دچار آسیب می‌شود.

AfterView

نمونه‌ی *AfterView* هوک های ()AfterViewInit و ()AfterViewChecked که Angular بعد از ایجاد view های فرزند یک کامپوننت فراخوانی می‌کند را بررسی می‌کند.

در ادامه می‌توانید یک view فرزند را مشاهده کنید که اسم یک هیرو را در یک <input> نمایش می‌دهد:

```
ChildComponent
@Component({
 selector: 'app-child-view',
 template: '<input [(ngModel)]="hero">'
})
export class ChildViewComponent {
 hero = 'Magneta';
}
```

AfterViewComponent این view فرزند را داخل قالب خود نمایش می‌دهد:

```
AfterViewComponent (template)
template: `
 <div>-- child view begins --</div>
 <app-child-view></app-child-view>
 <div>-- child view ends --</div>`
```

هوک های زیر بر اساس تغییر مقادیر داخل این view فرزند اقدامات لازم را انجام می‌دهند. تنها راهی که می‌توان از طریق آن به این view دسترسی پیدا کرد، این است که این view فرزند را از طریق ویژگی نشان شده توسط @ViewChild پرس و جو کرد.

```
AfterViewComponent (class excerpts)
export class AfterViewComponent implements AfterViewChecked, AfterViewInit {
 private prevHero = '';
```

```
// Query for a VIEW child of type `ChildViewComponent`
 @ViewChild(ChildViewComponent) viewChild: ChildViewComponent;


 ngAfterViewInit() {
// viewChild is set after the view has been initialized
 this.logIt('AfterViewInit');
 this.doSomething();
 }


 ngAfterViewChecked() {
// viewChild is updated after the view has been checked
 if (this.prevHero === this.viewChild.hero) {
 this.logIt('AfterViewChecked (no change)');
 } else {
 this.prevHero = this.viewChild.hero;
 this.logIt('AfterViewChecked');
 this.doSomething();
 }
 }
// ...
}
```

به قوانین جریان یک طرفه‌ی داده وفادار بمانید

متد doSomething() زمانی که تعداد کاراکترهای اسم هیرو از 10 عدد بیشتر شود، صفحه نمایش را به روز می‌کند.

```
AfterViewComponent (doSomething)
// This surrogate for real business logic sets the `comment`
private doSomething() {
 let c = this.viewChild.hero.length > 10? `That's a long name`: '';
 if (c!== this.comment) {
```

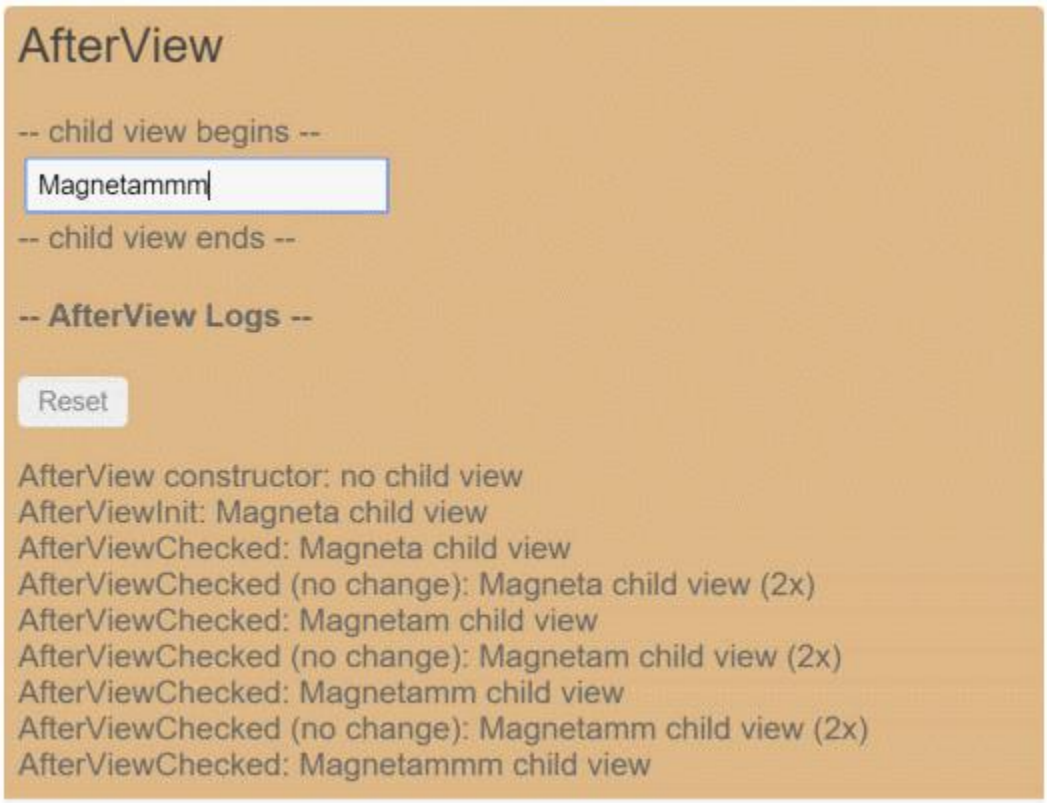
```
// Wait a tick because the component's view has already been checked
 this.logger.tick_then(() => this.comment = c);
 }
}
```

چرا متد doSomething() قبل از به روز رسانی comment باید اندکی صبر کند؟

قانون جریان یک طرفه‌ی داده در Angular، به روز کردن view پس از تکمیل ساخت آن را منع می‌کند. هر دوی این هوک ها پس از تشکیل view کامپوننت می‌سوزند.

اگر هوک ها بلافاصله ویژگی comment مقید به داده‌ی کامپوننت را به روز رسانی کند، Angular خطا می‌دهد (امتحان کنید!). LoggerService.tick_then() به روز رسانی گزارش را به مدت یک دور از چرخه‌ی جاوا اسکریپت مرورگر به عقب می‌اندازد. این مقدار از زمان به اندازه‌ی کافی طولانی است.

در ادامه می‌توانید AfterView را در عمل مشاهده کنید:



توجه داشته باشید که Angular بارها و اغلب زمانی که هیچ تغییر مطلوبی وجود نداشته باشد، AfterViewChecked() را فراخوانی می‌کند. برای جلوگیری از مشکلات مربوط به کارایی، متدهای هوک را به صورت کم حجم بنویسید.

AfterContent

نمونه‌ی *AfterContent* هوک های AfterContentInit() و AfterContentChecked() را بررسی می‌کند. Angular این دو هوک را بعد از طرح ریزی محتواهای خارجی در کامپوننت فراخوانی می‌کند.

طرح ریزی محتوا

طرح ریزی محتوا روشی برای وارد کردن محتوای HTML از خارج از کامپوننت و درج این محتوا داخل قالب کامپوننت و در مکانی تعیین شده است.

برنامه نویسان AngularJS این روش را به عنوان *transclusion* می‌شناسند.

این تغییر را در مثال قبل *AfterView* در نظر بگیرید. این مثال این بار به جای استفاده از view فرزند داخل قالب، محتوا را از مادر AfterContentComponent وارد می‌کند. در ادامه می‌توانید قالب مادر را مشاهده کنید:

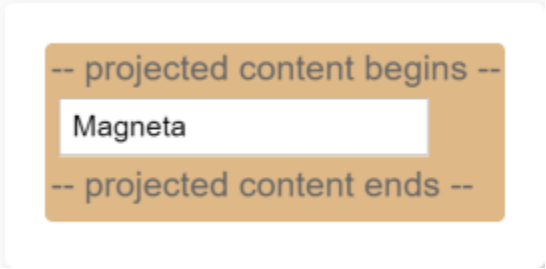
```
AfterContentParentComponent (template excerpt)
`<after-content>
 <app-child></app-child>
 </after-content>`
```

توجه داشته باشید که تگ <app-child> بین تگ‌های <after-content> قرار گرفته است. هیچ وقت محتوا را بین تگ‌های عنصر یک کامپوننت قرار ندهید مگر آن که بخواهید این محتوا را داخل کامپوننت طرح ریزی کنید.

حالا به قالب این کامپوننت نگاه کنید:

```
AfterContentComponent (template)
template: `
 <div>-- projected content begins --</div>
 <ng-content></ng-content>
 <div>-- projected content ends --</div>`
```

تگ <ng-content> یک نگهدارنده ی مکان برای محتوای خارجی است. این تگ مکان درج این محتوا را به Angular می‌گوید. در این حالت محتوای طرح ریزی شده، <app-child> حاصل از مادر است.



علائم خبرچینی طرح ریزی محتوا دو مورد زیر هستند:

- وجود HTML بین تگ‌های عنصر کامپوننت.
- حضور تگ‌های <ng-content> در قالب کامپوننت.

هوک های AfterContent

این هوک ها شبیه به هوک های *AfterView* عمل می‌کنند. تفاوت اصلی آن‌ها در کامپوننت فرزند نهفته است.

- هوک های *AfterView* به ViewChildren یا همان کامپوننت های فرزندی که تگ‌های عنصر آن‌ها داخل قالب کامپوننت ظاهر می‌شوند، رسیدگی می‌کنند.
- هوک های *AfterContent* به ContentChildren یا همان کامپوننت های فرزندی که Angular آن‌ها را داخل کامپوننت طرح ریزی کرده است، رسیدگی می‌کنند.

هوک های *AfterContent* بر اساس تغییر مقادیر در یک محتوای فرزند اقدامات لازم را انجام می‌دهند. تنها راه دسترسی به این هوک ها پرس و جوی آن‌ها توسط ویژگی نشان شده با @ContentChild است.

AfterContentComponent (class excerpts)

```
export class AfterContentComponent implements AfterContentChecked, AfterContentInit {
 private prevHero = '';
 comment = '';

// Query for a CONTENT child of type `ChildComponent`
 @ContentChild(ChildComponent) contentChild: ChildComponent;
```

```
 ngAfterContentInit() {
// contentChild is set after the content has been initialized
 this.logIt('AfterContentInit');
 this.doSomething();
 }

 ngAfterContentChecked() {
// contentChild is updated after the content has been checked
 if (this.prevHero === this.contentChild.hero) {
 this.logIt('AfterContentChecked (no change)');
 } else {
 this.prevHero = this.contentChild.hero;
 this.logIt('AfterContentChecked');
 this.doSomething();
 }
 }
// ...
}
```

هیچ جریان یک طرفه‌ای نسبت به *AfterContent* نگرانی ندارد

متد doSomething() این کامپوننت، بلافاصله ویژگی comment مقید به داده‌ی کامپوننت را به روز می‌کند و هیچ نیازی به صبر کردن ندارد.

به خاطر آورید که Angular قبل از فراخوانی هر دو هوک AfterView هر دو هوک AfterContent را فراخوانی می‌کند. Angular قبل از اتمام ساخت view این کامپوننت، کار ساخت محتوای طرح ریزی شده را تمام می‌کند. بین هوک های AfterContent... و AfterView... روزنه‌ی کوچکی وجود دارد تا بتوان view میزبان را اصلاح کرد.