سینتکس قالب

برنامه ی Angular چیزهایی که کاربر می بیند و کارهایی که می تواند انجام دهد را مدیریت می کند. Angular این کار را از طریق تعامل میان یک نمونه ی کلاس کامپوننت (خود کامپوننت) و قالب کاربری آن انجام می دهد.

احتمالا از تجربه ای که از model-view-controller (MVC) یا model-view-viewmodel (MVVM) کسب کرده اید، با دوگانه ی کامپوننت – قالب آشنا شده اید. در Angular کامپوننت نقش controller/viewmodel و قالب نقش view را ایفا می کند.

این صفحه از نظر فنی مرجع کاملی برای زبان قالب Angular است. در این صفحه به اصول اولیه ی زبان قالب پرداخته می شود و به اغلب سینتکس هایی که در بخش های دیگر این آموزش ممکن است با آن ها مواجه شوید پرداخته می شود.

بسیاری از تکه کدها، نکات و مفاهیم را توضیح می دهند که نسخه ی کامل آن ها را می توانید در این لینک مشاهده کنید.

HTML در قالب ها

HTML زبان قالب Angular است. تقریبا کل سینتکس HTML در سینتکس قالب معتبر است. البته عنصر <script> استثنای جالب توجهی است. استفاده از آن ممنوع است تا بتوان از خطرات حملات تزریق <script> جلوگیری کرد. در عمل <script> نادیده گرفته می شود و در کنسول مرورگر هشداری نمایش داده می شود. برای اطلاعات بیشتر به صفحه ی امنیت مراجعه کنید.

برخی از HTML های مجاز مفهوم خاصی در قالب ندارند. عناصر <body>, <html> و <base> نقش مفیدی ندارند. تقریبا می توان از عناصر دیگر نیز به این صورت انتقاد کرد.

برای گسترش فهرست لغات HTML قالب های خود می توانید از دستورالعمل ها و کامپوننت هایی استفاده کنید که به صورت صفات و عناصر جدیدی ظاهر می شوند. در بخش های زیر چگونگی دریافت و تنظیم مقادیر DOM به صورت پویا از طریق مقیدسازی داده را خواهید آموخت.

کار خود را با اولین نوع از مقیدسازی داده (میانگیری) آغاز کنید تا بیشتر قدر HTML قالب را بدانید.

میانگیری ({{...}})

پیش از این در این آموزش با میانگیری دو آکولادی {{ and }} آشنا شده اید.

src/app/app.component.html

```
<p>My current hero is {{currentHero.name}}</p>
```

برای ساختن رشت های محاسبه شده داخل متونی بین تگ های عناصر HTML و درون تخصیص صفات می توانید از میانگیری استفاده کنید.

src/app/app.component.html

```
<h3>
 {{title}}
 <img src="{{heroImageUrl}}" style="height:30px">
</h3>
```

متنی که داخل آکولادها قرار می گیرد، اغلب اسم یک ویژگی کامپوننت است. Angular جای این اسم را با مقدار رشته ای ویژگی کامپوننت متناظر عوض می کند. در مثال بالا Angular ویژگی های title و heroImageUrl را ارزیابی می کند و جاهای خالی را پر می کند. به این صورت که ابتدا عنوان برنامه به صورت بولد و پس از آن تصویر مربوط به هیرو را نمایش می دهد.

به طور کلی تر متن بین آکولادها یک عبارتی از قالب است که Angular ابتدا آن را ارزیابی می کند و سپس آن را به یک رشته تبدیل می کند. در میانگیری زیر این مطلب با جمع دو عدد نشان داده شده است:

src/app/app.component.html

```
<!-- "The sum of 1 + 1 is 2" -->
<p>The sum of 1 + 1 is {{1 + 1}}</p>
```

این عبارت می تواند متدهای کامپوننت میزبان مانند ()getVal که در زیر آمده است را احضار کند:

src/app/app.component.html

```
<!-- "The sum of 1 + 1 is not 4" -->
<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}</p>
```

Angular تمامی عبارت هایی که داخل دو آکولاد قرار دارند را ارزیابی می کند، نتایج عبارت را به رشته هایی تبدیل می کند و آن ها را به رشته های لفظی مجاور پیوند می زند. در نهایت Angular این نتیجه ی ترکیبی میانگیری شده را به یک ویژگی دستورالعمل یا عنصر تخصیص می دهد.

شاید در ظاهر این طور به نظر برسد که شما نتیجه را بین تگ های عنصر درج می کنید و آن را به صفاتی تخصیص می دهید. این طور به قضیه نگاه کردن کارتان را راحت می کند و در آینده به ندرت با مشکل مواجه خواهید شد اما با این وجود کاملا درست نیست. میانگیری سینتکس ویژه ای است که همان طور که در زیر توضیح داده شده است، Angular آن را به یک مقیدسازی ویژگی تبدیل می کند.

اما ابتدا اجازه دهید نگاه دقیق تری به دستورات و عبارت های قالب بیاندازیم.

عبارت های قالب

کار یک عبارت قالب این است که مقداری را تولید کند. Angular این عبارت را اجرا می کند و آن را به یک ویژگی از هدف binding تخصیص می دهد. این هدف می تواند یک عنصر HTML، کامپوننت یا دستورالعمل باشد.

آکولادهای میانگیری موجود در {{1 + 1}} اطراف عبارت قالب 1+1 را احاطه می کنند. در بخش مقیدسازی ویژگی که در زیر آمده است، یک عبارت قالب داخل علامت نقل قول و در سمت راست علامت مساوی، ظاهر می شود. به این صورت: [property]="expression"

این عبارت ها در زبانی نوشته می شوند که شبیه به جاوا اسکریپت است. از بسیاری از عبارت های جاوا اسکریپت می توان به عنوان عبارت های قالب استفاده کرد.

استفاده از عبارت های جاوا اسکریپت که عوارض جانبی به دنبال دارند و یا به نوعی باعث بروز این عوارض می شوند، ممنوع است. از جمله:

- تخصیص دهی (=, +=, -=, ...)
- new
- عبارت های زنجیره ای به همراه ; یا ,
- عملگرهای کاهش و افزایش پله ای (++ و --)

از جمله دیگر تفاوت هایی که این سینتکس ها با سینتکس های جاوا اسکریپت دارند، می توان به موارد زیر اشاره کرد:

- عدم پشتیبانی از عملگرهای بیتی | و &
- عملگرهای عبارتی قالب جدید مانند |, ?. و !.

زمینه ی عبارت

زمینه ی عبارت معمولا به نمونه ی کامپوننت گفته می شود. در تکه کدهای زیر title موجود در دو آکولاد و isUnchanged داخل نقل قول به ویژگی های AppComponent مربوط هستند.

src/app/app.component.html

```
{{title}}
<span [hidden]="isUnchanged">changed</span>
```

ممکن است عبارت ها به ویژگی های زمینه ی قالب نیز اشاره داشته باشند، مانند یک متغیر ورودی قالب (let hero) یا متغیر مرجع قالب (#heroInput).

src/app/app.component.html

```
<div *ngFor="let hero of heroes">{{hero.name}}</div>
<input #heroInput> {{heroInput.value}}
```

زمینه ی مربوط به واژه های موجود در یک عبارت، ترکیبی از متغیرهای قالب، شیء زمینه ی دستورالعمل (در صورت وجود) و اعضای کامپوننت است. اگر به اسمی اشاره کنید که به بیش از یکی از این فضای نام ها تعلق داشته باشد، در این صورت اسم متغیر قالب به همراه اسمی در زمینه ی دستورالعمل و درنهایت اسامی اعضای کامپوننت اولویت پیدا می کند.

در مثال قبل به چنین تلاقی های اسمی اشاره شده است. کامپوننت دارای ویژگی hero است و *ngFor یک متغیر قالب hero را تعریف می کند. hero موجود در {{hero.name}} به متغیر ورودی قالب و نه ویژگی کامپوننت اشاره دارد.

عبارت های قالب در فضای نام سراسری نمی توانند به هیچ چیزی اشاره کنند (غیر از undefined). آن ها نمی توانند به window یا document اشاره کنند و یا console.log یا Math.max را فراخوانی کنند. حتی در اشاره به اعضای زمینه ی عبارت محدودیت دارند.

راهبردهای عبارت

عبارت های قالب نقش اصلی در کارکرد درست و یا نادرست برنامه دارند. لطفا راهبردهای زیر را دنبال کنید:

- عدم وجودعوارض جانبی مشهود
- اجرای سریع
- سادگی
- تکرار شوندگی

تنها در شرایطی خاص و با آگاهی کامل می توانید از این راهبردها صرف نظر کنید.

عدم وجودعوارض جانبی مشهود

عبارت های قالب به غیر از مقدار ویژگی هدف، هیچ حالتی از برنامه را نباید تغییر دهند.

این قانون در سیاست های مربوط به «جریان یک طرفه ی داده» در Angular ضروری است. شما هیچ وقت نباید نگران باشید که خواندن یک مقدار کامپوننت باعث تغییر دیگر مقادیر نمایش داده شده می شود یا نمی شود. view طی یک مرحله رندر شدن باید کاملا پایدار باشد.

اجرای سریع

Angular بعد از هر بار تغییر چرخه ی شناسایی، عبارت های قالب را اجرا می کند. این چرخه ها توسط فعالیت های ناهمگام زیادی فعال می شوند. مانند promise resolution، نتایج HTTP، رویدادهای تایمر، فشردن دکمه و حرکت موس.

عبارت ها باید سریعا تمام شوند و گرنه ممکن است تجربه ی کاربری drag شود، مخصوصا در دستگاه های کندتر. اگر هزینه ی محاسبه ی مقادیر زیاد است، حتما از آن ها cache بگیرید.

سادگی

با وجود اینکه در نوشتن عبارت های قالب کاملا پیچیده آزاد هستید، اما بهتر است این کار را نکنید.

فراخوان یک متد و یا اسم ویژگی بهتر است عادی باشد. هر از چند گاهی استفاده از کاراکتر نفی بولی (!) ایرادی ندارد. در غیر این صورت برنامه و منطق کسب و کار را در خود کامپوننت محصور کنید تا توسعه و آزمایش آن آسان تر شود.

تکرار شوندگی

عبارت های تکرار شوندگی به این دلیل مناسب هستند که عاری از عوارض جانبی بوده و عملکرد شناسایی تغییر Angular را ارتقا می دهند.

به زبان Angular یکی عبارت تکرار شوندگی همواره و دقیقا همان چیز قبلی را برگشت می دهد تا زمانی که یکی از مقادیر وابستگی آن تغییر کند.

طی تنها یک دوره از حلقه ی رویداد، مقادیر وابستگی نباید تغییر کنند. اگر یک عبارت تکرار شوندگی، رشته و یا عددی را برگشت دهد، این عبارت اگر دوبار پشت سرهم فراخوانی شود، همان عدد و یا رشته را برگشت می دهد. اگر این عبارت شیئی را برگشت دهد (از جمله یک آرایه)، بعد از آن که دو بار پشت سرهم فراخوانی شود، همان مرجع شیء را برگشت می دهد.

دستورات قالب

این دستورات به رویداد برآمده از یک هدف مقیدسازی مانند یک عنصر، کامپوننت یا دستورالعمل پاسخ می دهند. همان طور که مشاهده می کنید دستورات قالب موجود در بخش مقیدسازی رویداد در علامت های نقل قول و در سمت راست علامت مساوی قرار می گیرند. مانند "statement"=(event).

src/app/app.component.html

```
<button (click)="deleteHero()">Delete hero</button>
```

دستورات قالب عوارض جانبی دارند. البته کل نکته ی یک رویداد مربوط به همین مطلب است. زیرا به این صورت می توانید از طریق اعمال کاربر، برنامه را به روز رسانی کنید.

پاسخ دادن به رویدادها، جنبه ی دیگری از جریان یک طرفه ی داده ها درAngular است. طی این دور از حلقه ی رویداد، شما آزاد هستید تا هر چیزی را در هر جایی تغییر دهید.

دستورات قالب، مانند عبارت های قالب از زبانی مانند زبان جاوا اسکریپت استفاده می کنند. تجزیه کننده ی دستورات قالب با تجزیه کننده ی عبارت های قالب متفاوت است و صریحا از تخصیص دهی پایه ای (=) و عبارت های زنجیره ای (همراه با ; یا ,) پشتیبانی می کند.

به هر حال، امکان استفاده از برخی از سینتکس های جاوا اسکریپت وجود ندارد:

- New
- عملگرهای افزایش و کاهش پله ای ++ و --
- عملگر تخصیصی مانند =+ و =-
- عملگرهای بیتی | و &
- عملگرهای عبارت قالب

زمینه ی دستور

دستورات مانند عبارت ها، تنها می توانند به چیزی اشاره کنند که داخل زمینه ی دستورات هستند. مانند متد event handling مربوط به نمونه ی کامپوننت.

زمینه ی دستور معمولا نمونه ی کامپوننت است. *deleteHero* داخل "deleteHero()"=(click) متدی از کامپوننت مقید به داده است.

src/app/app.component.html

```
<button (click)="deleteHero()">Delete hero</button>
```

زمینه ی دستور نیز به ویژگی های زمینه ی خود قالب اشاره می کند. در مثال های زیر شیء $event قالب یک متغیر ورودی قالب (let hero) و یک متغیر مرجع قالب (#heroForm) به متد event handling کامپوننت داده می شود.

src/app/app.component.html

```
<button (click)="onSave($event)">Save</button>
<button *ngFor="let hero of heroes" (click)="deleteHero(hero)">{{hero.name}}</button>
<form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form>
```

اسامی زمینه ی قالب نسبت به اسامی زمینه ی کامپوننت اولویت بیشتری دارند. در deleteHero(hero) بالا، hero متغیر ورودی قالب و نه ویژگی hero کامپوننت است.

دستورات قالب نمی توانند به چیزی در فضای نام سراسری اشاره کنند. همچنین آن ها قادر نیستند به window یا document اشاره کنند و نمی توانند console.log یا Math.max را فراخوانی کنند.

راهبردهای دستور

درست مانند عبارت ها، سعی کنید از نوشتن دستورات پیچیده ی قالب پرهیز کنید. یک فراخوان متد و یا تخصیص دهی ساده ی ویژگی می تواند حد استاندارد باشد.

حالا که با عبارت ها و دستورات قالب آشنا شده اید، دیگر می توانید در رابطه با انواع مختلفی از سینتکس هاس مقیدسازی داده که فراتر از میانگیری هستند، اطلاعات کسب کنید.

مروری بر سینتکس مقیدسازی

مقیدسازی داده مکانیزمی است که به کمک آن می توان چیزهایی که کاربر می بیند را با مقادیر داده های برنامه هماهنگ کرد. درست است که می توانید مقادیر را به HTML ارسال کنید و یا از آن دریافت کنید، اما اگر این کارها را به چهارچوب مقیدسازی بسپارید در این صورت خواندن، نوشتن و نگهداری برنامه تان آسان تر خواهد شد. در این حالت تنها کاری که انجام می دهید این است که مقیدسازی ها را بین منابع مقیدسازی و عناصر HTML هدف اعلان می کنید و باقی کار را به این چهارچوب می سپارید.

Angular، انواع مختلفی از مقیدسازی داده را فراهم می کند. در این آموزش بعد از بررسی دقیق مقیدسازی داده در Angular و سینتکس آن به اغلب آن ها می پردازیم.

انواع مقیدسازی را می توان به 3 دسته تقسیم کرد و آن ها را بر اساس مسیر جریان داده تفکیک نمود: از منبع به view ، از view به منبع و به صورت دنباله ای دو طرفه از view به منبع به view:

جهت داده ها	سینتکس	نوع
یک طرفه از منبع داده به هدف view	{{expression}} [target]="expression" "bind-target="expression	میانگیری ویژگی صفت کلاس سبک
یک طرفه از هدف view به منبع داده	(target)="statement" on-target="statement"	رویداد
دو طرفه	[(target)]="expression" bindon-target="expression"	دو طرفه

انواع مقیدسازی به غیر از میانگیری در سمت چپ علامت تساوی دارای یک اسم هدف هستند که این اسم یا داخل علامت (() , []) قرار دارد و یا بعد از پیشوند (bind-, on-, bindon-) آمده است.

اسم هدف، اسم یک ویژگی است. ممکن است در ظاهر اسم یک صفت باشد، اما این طور نیست. برای آن که این تفاوت را درک کنید، باید از زاویه ی جدیدی به HTML قالب نگاه کنید.

یک مدل ذهنی جدید

با توجه به توانایی بالای مقیدسازی داده در توسعه ی فهرست لغات HTML به کمک markup های اختصاصی، در نظر گرفتن HTML قالب به عنوان HTML Plus وسوسه انگیز است.

این واقعا HTML Plus است، اما تفاوت زیادی با HTML ای دارد که شما به آن عادت کرده اید. زیرا به مدل ذهنی جدیدی نیاز دارد.

در دوره ی معمولی توسعه ی HTML شما یک ساختار بصری را به همراه عناصر HTML ایجاد می کنید و با قرار دادن صفات این عناصر بر روی ثابت های رشته ای آن ها را تغییر می دهید.

src/app/app.component.html

```
<div class="special">Mental Model</div>
<img src="assets/images/hero.png">
<button disabled>Save</button>
```

شما هنوز به این روش در قالب های Angular، ساختارها را ایجاد می کنید و صفات را مقداردهی اولیه می کنید.

سپس می آموزید به کمک کامپوننت هایی عناصر جدیدی را ایجاد کنید، که این کامپوننت ها HTML را درون خود جای می دهند و آن ها را درست مانند حالتی که جزء عناصر بومی HTML هستند درون قالب ها رها می کنند.

src/app/app.component.html

```
<!-- Normal HTML -->
<div class="special">Mental Model</div>
<!-- Wow! A new element! -->
<app-hero-detail></app-hero-detail>
```

این همان HTML Plus است.

حالا در رابطه با مقیدسازی داده اطلاعات بیشتری کسب می کنید. اولین مقیدسازی ای که شما با آن مواجه می شوید، چیزی مانند زیر است:

src/app/app.component.html

```
<!-- Bind button disabled state to `isUnchanged` property -->
<button [disabled]="isUnchanged">Save</button>
```

به زودی به علامت کروشه ی عجیب بالا می پردازیم. صرف نظر از این کروشه، برداشت شما این است که این کد در حال مقید شدن به صفت disabled و تنظیم آن بر روی مقدار فعلی ویژگی isUnchanged کامپوننت است.

برداشت شما نادرست است! مدل ذهنی همیشگی شما در رابطه با HTML گمراه کننده است. چیزی که در حقیقت در حال رخ دادن است این است که بعد از آن که شما مقیدسازی داده را آغاز کنید، دیگر کاری به صفات HTML و تنظیم صفات ندارید؛ بلکه تنها ویژگی های عناصر DOM ، کامپوننت ها و دستورالعمل ها را تنظیم می کنید.

صفت HTML در برابر ویژگی DOM

برای درک نحوه ی مقیدسازی توسط Angular تشخیص تفاوت بین صفت HTML و ویژگی DOM حیاتی است.

صفات توسط HTML و ویژگی ها توسط DOM تعریف می شوند.

- تعداد کمی از صفات HTML در برابر ویژگی ها نگاشت 1:1 دارند. id یکی از آن ها است.
- برخی از صفات HTML دارای ویژگی متناظر نیستند، مانند colspan.
- برخی از ویژگی های DOM دارای صفات متناظر نیستند، مانند textContent.
- بسیاری از صفات HTML به ویژگی ها نگاشت می شوند... اما نه به آن صورتی که شما احتمالا فکر می کنید.

تا زمانی که قانون کلی زیر را متوجه نشوید، نمی توانید دسته ی آخر را درک کنید:

صفت ها تنها کاری که با ویژگی های DOM دارند این است که آن ها را مقداردهی اولیه می کنند و دیگر کاری با آن ها ندارند.

برای مثال، زمانی که مرورگر <input type="text" value="Bob"> را رندر می کند، به همراه یک ویژگی value که مقدار اولیه ی آن برابر با "Bob" است، یک گره ی DOM متناظر را ایجاد می کند.

زمانی که کاربر "Sally" را درون کادر ورودی وارد می کند، ویژگی value مربوط به عنصر DOM به "Sally" تبدیل می شود، اما همان طور که از عنصر ورودی (input.getAttribute('value') پیداست صفت value مربوط به HTML تغییری نمی کند و (input.getAttribute('value') ، "Bob" را برگشت می دهد.

Value صفت HTML مقدار اولیه را مشخص می کند؛ ویژگی value مربوط به DOM مقدار فعلی آن است. صفت disabled مثال عجیب دیگری است. ویژگی disabled یک دکمه به صورت پیش فرض false است تا این دکمه فعال باشد. زمانی که صفت disabled را اضافه می کنید، تنها حضور آن ویژگی disabled دکمه را بر روی مقدار true قرار می دهد و دکمه غیرفعال می شود.

اضافه و حذف کردن صفت disabled باعث غیرفعال و فعال شدن دکمه می شود. مقدار این صفت بی ربط است. به همین دلیل شما نمی توانید با نوشتن <button disabled="false">Still Disabled</button> دکمه ای را فعال کنید.

تنظیم ویژگی disabled دکمه (مثلا به کمک یک مقیدسازی Angular) باعث می شود که دکمه فعال و یا غیرفعال شود. چیزی که اهمیت دارد، مقدار ویژگی است.

صفت HTML و ویژگی DOM یکسان نیستند، حتی زمانی که اسم یکسانی داشته باشند.

این حقیقت ارزش تکرار کردن دارد. مقیدسازی قالب در کنار ویژگی ها و رویدادها کار می کند، نه صفات.

دنیای بدون صفات

در دنیای Angular، تنها نقشی که صفت ها دارند این است که عنصر و حالت دستورالعمل را مقداردهی اولیه کنند. زمانی که شما اقدام به نوشتن یک مقیدسازی داده می کنید، منحصرا با ویژگی ها و رویدادهای شیء هدف سروکار دارید. صفات HTML عملا حذف می شوند.

حالا یا در نظر داشتن این مدل در ذهنتان، به خواندن ادامه دهید تا در رابطه با هدف های مقیدسازی بیشتر بیاموزید.

هدف های مقیدسازی

هدف مقیدسازی به چیزی گفته می شود که داخل DOM باشد. این هدف با توجه به نوع مقیدسازی می تواند یک ویژگی (عنصر | کامپوننت | دستورالعمل)، یک رویداد (عنصر | کامپوننت | دستورالعمل) و یا (به ندرت) یک اسم صفت باشد. این موارد به صورت مختصر در جدول زیر بیان شده اند:

نوع	هدف	مثال ها
ویژگی	ویژگی عنصر ویژگی کامپوننت ویژگی دستورالعمل	src/app/app.component.html <img [src]="heroImageUrl"> <app-hero-detail [hero]="currentHero"></app-hero-detail> <div [ngClass]="{'special': isSpecial}">
رویداد	رویداد عنصر رویداد کامپوننت	src/app/app.component.html <button (click)="onSave()">Save</button>

<app-hero-detail (deleteRequest)="deleteHero()"></app-hero- detail> div (myClick)="clicked=$event" clickable>click > <me</div	رویداد دستورالعمل	
src/app/app.component.html <"input [(ngModel)]="name>	رویداد و ویژگی	دو طرفه
src/app/app.component.html <button [attr.aria-label]="help">help</button>	صفت (استثنا)	صفت
src/app/app.component.html <div [class.special]="isSpecial">Special	ویژگی class	کلاس
src/app/app.component.html button [style.color]="isSpecial ? 'red' : > <""green	ویژگی style	سبک

با در نظر داشتن این دید کلی، حالا دیگر می توانید به صورت دقیق تر به یادگیری انواع مقیدسازی بپردازید.

مقیدسازی ویژگی ([property])

یک مقیدسازی ویژگی قالب را بنویسید تا ویژگی یک عنصر view تنظیم شود. این مقیدسازی مقدار ویژگی را برابر با مقدار یک عبارت قالب قرار می دهد.

رایج ترین مقیدسازی ویژگی یک ویژگی عنصر را بر روی یک مقدار ویژگی کامپوننت تنظیم می کند. به عنوان مثال می توان به مقیدسازی ویژگی src مربوط به عنصر تصویر به یک ویژگی heroImageUrl کامپوننت اشاره کرد:

src/app/app.component.html

```
<img [src]="heroImageUrl">
```

مثال دیگر مربوط به غیرفعال کردن یک دکمه است، در زمانی که کامپوننت isUnchanged است:

src/app/app.component.html

```
<button [disabled]="isUnchanged">Cancel is disabled</button>
```

مثال دیگر تنظیم یکی از ویژگی های یک دستورالعمل است:

src/app/app.component.html

```
<div [ngClass]="classes">[ngClass] binding to the classes property</div>
```

و مثال آخر تنظیم ویژگی مدل یک کامپوننت اختصاصی است (یکی از بهترین روش ها برای کامپوننت های فرزند و مادر است تا بتوانند با یکدیگر ارتباط برقرار کنند):

src/app/app.component.html

```
<app-hero-detail [hero]="currentHero"></app-hero-detail>
```

ورود یک طرفه

افراد اغلب مقیدسازی ویژگی را به عنوان یک مقیدسازی یک طرفه ی داده در نظر می گیرند زیرا طی آن مقداری در یک جهت از ویژگی داده ی یک کامپوننت به سمت ویژگی یک عنصر هدف جریان پیدا می کند.

نمی توانید برای خارج کردن مقادیر از عنصر هدف از مقیدسازی ویژگی استفاده کنید، نمی توانید برای خواندن یک ویژگی عنصر هدف به آن مقید شوید، تنها می توانید آن را تنظیم کنید.

به طور مشابه برای فراخوانی یک متد در عنصر هدف نمی توانید از مقیدسازی ویژگی استفاده کنید.

اگر این عنصر باعث بروز رویدادهایی شود، می توانید به کمک مقیدسازی رویداد به آن ها توجه کنید.

اگر حتما باید ویژگی یک عنصر هدف را بخوانید یا یکی از متدهای آن را فراخوانی کنید، به روش متفاوتی نیاز دارید. برای اطلاعات بیشتر به مرجع API مربوط به ViewChild و ContentChild مراجعه کنید.

هدف مقیدسازی

ویژگی عنصری که بین دو کروشه قرار می گیرد، ویژگی هدف را مشخص می کند. ویژگی src عنصر تصویر در کد زیر همان ویژگی هدف است.

src/app/app.component.html

```
<img [src]="heroImageUrl">
```

برخی از افراد به عنوان جایگزین، پیشوند -bind را ترجیح می دهند که به این شیوه صورت متعارف گفته می شود:

src/app/app.component.html

```
<img bind-src="heroImageUrl">
```

اسم هدف همیشه اسم یک ویژگی است، حتی اگر ظاهر آن شبیه به اسم چیز دیگری باشد. ممکن است از نظر شما src اسم یک صفت باشد، اما این طور نیست؛ بلکه اسم ویژگی عنصر یک تصویر است.

ویژگی های عناصر معمولا هدف های رایج تری هستند، اما Angular در ابتدا به دنبال اسمی می گردد که ویژگی یک دستورالعمل شناخته شده باشد. مانند مثال زیر:

src/app/app.component.html

```
<div [ngClass]="classes">[ngClass] binding to the classes property</div>
```

از نظر فنی Angular این اسم را با یک ورودی دستورالعمل تطبیق می دهد. این ورودی یکی از اسامی ویژگی های فهرست شده در آرایه ی inputs این دستورالعمل یا یک دکوراتور @Input() است. چنین ورودی هایی به ویژگی های خود دستورالعمل نگاشت می شوند.

اگر این اسم نتواند ویژگی یک دستورالعمل یا عنصر شناخته شده را تطبیق دهد، در این صورت Angular خطای "unknown directive" می دهد.

پرهیز از اثرات جانبی

همان طور که قبلا نیز بیان شد، ارزیابی یک عبارت قالب نباید هیچ اثر جانبی مشهودی داشته باشد. برای اینکه از این بابت خیال شما راحت باشد، زبان عبارت کار خود را انجام می دهد. شما نه می توانید مقداری را به چیزی در یک عبارت مقیدسازی ویژگی تخصیص دهید و نه می توانید از عملگرهای افزایش و کاهش پله ای استفاده کنید.

البته ممکن است که این عبارت ویژگی و یا متدی را احضار کند که دارای اثرات جانبی باشد. Angular برای شناسایی این اثرات و یا متوقف کردن شما راهی را نمی شناسد.

این عبارت می تواند چیزی مانند ()getFoo را فراخوانی کند. این تنها شما هستید که می دانید ()getFoo چه کاری انجام می دهد. اگر ()getFoo چیزی را تغییر دهد و شما به صورت اتفاقی در حال مقیدسازی به آن چیز باشید، ممکن است با تجربه ی ناخوشایندی روبرو شوید. تضمینی وجود ندارد که Angular این مقدار تغییر یافته را نمایش دهد. Angular ممکن است این تغییر را شناسایی کند و خطای هشداری بدهد. به صورت کلی از ویژگی های داده ها و متدهایی استفاده کنید که تنها مقادیر را برگشت می دهند و نه بیشتر.

برگشت دادن نوع صحیح

عبارت قالب باید با نوعی از مقداری ارزیابی شود که این مقدار مورد انتظار ویژگی هدف است. اگر ویژگی هدف انتظار یک رشته را دارد، همان رشته را برگشت دهید. اگر ویژگی هدف انتظار یک عدد را دارد، همان عدد را برگشت دهید و اگر ویژگی هدف انتظار یک شیء را دارد، همان شیء را برگشت دهید.

ویژگی hero کامپوننت HeroDetail انتظار یک شیء hero را دارد که این دقیقا همان چیزی است که شما در مقیدسازی ویژگی در حال ارسال آن هستید.

src/app/app.component.html

```
<app-hero-detail [hero]="currentHero"></app-hero-detail>
```

براکت ها را فراموش نکنید

براکت ها به Angular می گویند که عبارت قالب را ارزیابی کند. اگر این براکت ها را از قلم بیاندازید، Angular این رشته را به عنوان یک ثابت در نظر می گیرد و ویژگی هدف را با این رشته مقداردهی اولیه می کند. Angular این رشته را ارزیابی نمی کند!

اشتباه زیر را نکنید:

src/app/app.component.html

```
<!-- ERROR: HeroDetailComponent.hero expects a
   Hero object, not the string "currentHero" -->
```

```
<app-hero-detail hero="currentHero"></app-hero-detail>
```

مقداردهی اولیه ی رشته به صورت یک باره

اگر تمامی موارد زیر برقرار باشند، شما می توانید از براکت ها صرف نظر کنید:

- **ویژگی هدف یک مقدار رشته ای را بپذیرد.**
- **این رشته مقدار ثابتی باشد که شما بتوانید آن را داخل قالب آماده کنید.**
- **این مقدار اولیه هیچ وقت تغییر نکند.**

شما به صورت روتین به این صورت در HTML استاندارد صفات را مقداردهی اولیه می کنید. همچنین این روش برای مقداردهی اولیه ی ویژگی کامپوننت و دستورالعمل به خوبی جواب می دهد. در مثال زیر ویژگی prefix مربوط به HeroDetailComponent بر روی مقدار اولیه و ثابت یک رشته و نه یک عبارت قالب تنظیم شده است. Angular آن را تنظیم می کند و دیگر کاری با آن ندارد.

src/app/app.component.html

```
<app-hero-detail prefix="You are my" [hero]="currentHero"></app-hero-detail>
```

از سوی دیگر [hero] به ویژگی currentHero کامپوننت همچنان مقید باقی می ماند.

مقیدسازی ویژگی یا میانگیری؟

شما اغلب بین این دو حق انتخاب دارید. دو روش مقید سازی زیر یک کار را انجام می دهند:

src/app/app.component.html

```
<p><img src="{{heroImageUrl}}"> is the <i>interpolated</i> image.</p>
<p><img [src]="heroImageUrl"> is the <i>property bound</i> image.</p>

<p><span>"{{title}}" is the <i>interpolated</i> title.</span></p>
<p>"<span [innerHTML]="title"></span>" is the <i>property bound</i> title.</p>
```

در بسیاری از موارد، میانگیری جایگزین راحت تری برای مقیدسازی ویژگی می باشد.

اگر در حال رندر کردن مقادیر داده های رشته ای هستید، هیچ دلیل فنی برای ترجیح یکی بر دیگری وجود ندارد، اما اگر خوانا بودن برای تان مهم است بهتر است از میانگیری استفاده کنید. بین این دو آن را انتخاب کنید که هم

از این قوانین کد نویسی ای که در ذهنتان است پیروی کند و هم بتوانید به کمک آن به راحت ترین حالت ممکن کار خود را انجام دهید.

در مواقعی که می خواهید مقدار ویژگی یک عنصر را برابر با یک مقدار غیر رشته ای قرار دهید، باید از مقیدسازی ویژگی استفاده کنید.

امنیت محتوا

محتوای آلوده ی زیر را در نظر بگیرید:

src/app/app.component.ts

```
evilTitle = 'Template <script>alert("evil never sleeps")</script>Syntax';
```

خوشبختانه مقیدسازی داده ی Angular نسبت به چنین HTML های خطرناکی در حالت آماده باش قرار دارد. Angular قبل از این که آن ها را نمایش دهد، این مقادیر را پاک سازی می کند و اجازه نمی دهد HTML هایی که دارای تگ های اسکریپت هستند به داخل مرورگر راه پیدا کنند. این کار نه در میانگیری و نه در مقیدسازی ویژگی رخ می دهد.

src/app/app.component.html

```
<!--
 Angular generates warnings for these two lines as it sanitizes them
 WARNING: sanitizing HTML stripped some content (see http://g.co/ng/security#xss).
-->
<p><span>"{{evilTitle}}" is the <i>interpolated</i> evil title.</span></p>
<p>"<span [innerHTML]="evilTitle"></span>" is the <i>property bound</i> evil title.</p>
```

شیوه ی رسیدگی میانگیری به تگ های اسکریپت با مقیدسازی ویژگی متفاوت است. اما هر دو بدون هیچ ضرری این محتوا را رندر می کنند.

"Template <script>alert("evil never sleeps")</script>Syntax" is the *interpolated* evil title.

"Template Syntax" is the *property bound* evil title.

مقیدسازی صفت، کلاس و سبک

سینتکس قالب برای موقعیت هایی که به خوبی نمی توان در آن ها از مقیدسازی ویژگی استفاده کرد، مقیدسازی های یک طرفه ی ویژه ای را ارائه می کند.

مقیدسازی صفت

برای تنظیم مقدار یک صفت می توانید مستقیما از مقیدسازی صفت استفاده کنید.

این نوع از مقیدسازی تنها استثناء موجود برای قانونی است که در آن گفته می شود مقیدسازی ویژگی هدف را تنظیم می کند. تنها در این مقیدسازی است که یک صفت ایجاد و تنظیم می شود.

در این آموزش بارها تاکید شده است که تنظیم ویژگی یک عنصر به کمک مقیدسازی ویژگی همیشه بر تنظیم صفت به کمک یک رشته اولویت دارد. حالا چرا Angular مقیدسازی صفت را ارائه می کند؟

در مواقعی که هیچ ویژگی عنصری برای مقیدسازی وجود ندارد، باید از مقیدسازی صفت استفاده کنید.

صفت های table span، SVG و ARIA را در نظر بگیرید. این ها صفت های خالص هستند و متناظر با ویژگی های عناصر نیستند و ویژگی های عناصر را تنظیم نمی کنند. همچنین در آن ها هیچ ویژگی هدفی برای مقیدسازی وجود ندارد.

اگر کدی مانند زیر را بنویسید، این حقیقت به صورت عذاب آوری آشکار می شود.

```
<tr><td colspan="{{1 + 1}}">Three-Four</td></tr>
```

و با این خطا مواجه می شوید:

```
Template parse errors:
Can't bind to 'colspan' since it isn't a known native property
```

همان طور که از پیام بالا مشخص است، عنصر <td> دارای ویژگی colspan نیست. این عنصر دارای صفت "colspan" است اما مساله این است که میانگیری و مقیدسازی ویژگی تنها می توانند ویژگی ها را تنظیم کنند و نه صفات را .

برای ایجاد و مقید شدن به چنین صفاتی، شما نیاز به مقیدسازی صفات دارید.

مقیدسازی صفت چیزی شبیه به مقیدسازی ویژگی است. با این تفاوت که به جای آن که با ویژگی یک عنصر بین براکت آغاز شود، با پیشوند attr به همراه یک نقطه (.) و اسم صفت آغاز می شود. سپس مقدار این صفت را با استفاده از عبارتی که به یک رشته تجزیه می شود تنظیم کنید.

[attr.colspan] را به یک مقدار محاسبه شده مقید کنید:

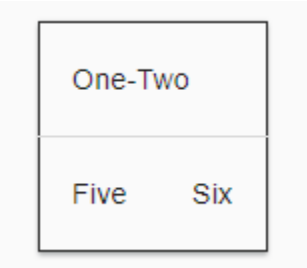
src/app/app.component.html

```
<table border=1>
 <!--  expression calculates colspan=2 -->
 <tr><td [attr.colspan]="1 + 1">One-Two</td></tr>

 <!-- ERROR: There is no `colspan` property to set!
  <tr><td colspan="{{1 + 1}}">Three-Four</td></tr>
 -->

 <tr><td>Five</td><td>Six</td></tr>
</table>
```

بعد از رندر شدن، جدول به شکل زیر در می آید:



یکی از اصلی ترین کاربردهای مقیدسازی صفت، تنظیم صفت های ARIA است. مانند مثال زیر:

src/app/app.component.html

```
<!-- create and set an aria attribute for assistive technology -->
<button [attr.aria-label]="actionName">{{actionName}} with Aria</button>
```

مقیدسازی کلاس

به کمک مقیدسازی کلاس می توانید اسامی کلاس های CSS را از صفت class یک عنصر حذف کنید و یا به آن اضافه کنید.

سینتکس مقیدسازی کلاس شبیه به مقیدسازی ویژگی است، با این تفاوت که به جای ویژگی یک عنصر داخل براکت، سینتکس آن با پیشوند class، در صورت تمایل همراه به یک نقطه (.) و اسم کلاس CSS آغاز می شود: [class.class-name].

در مثال های زیر چگونگی اضافه و حذف کردن کلاس "special" برنامه به کمک مقیدسازی کلاس نشان داده شده است. به صورت زیر می توانید بدون استفاده از مقیدسازی این صفت را تنظیم کنید:

src/app/app.component.html

```
<!-- standard class attribute setting  -->
<div class="bad curly special">Bad curly special</div>
```

می توانید جای آن را با یک مقیدسازی به رشته ای از اسامی کلاس های مورد نظر عوض کنید. این شیوه از جا به جایی یا به صورت کامل انجام می شود و یا اصلا انجام نمی شود.

src/app/app.component.html

```
<!-- reset/override all class names with a binding  -->
<div class="bad curly special"
   [class]="badCurly">Bad curly</div>
```

در نهایت می توانید به اسم کلاس مشخصی مقید شوید. زمانی که نتیجه ی ارزیابی عبارت قالب true شود، Angular این کلاس را اضافه می کند و اگر نتیجه false شود، آن را حذف می کند.

src/app/app.component.html

```
<!-- toggle the "special" class on/off with a property -->
<div [class.special]="isSpecial">The class binding is special</div>


<!-- binding to `class.special` trumps the class attribute -->
<div class="special"
```

[class.special]="!isSpecial">This one is not so special

با وجود این که این روش برای فعال کردن تنها یک اسم کلاس مناسب است، اما دستورالعمل NgClass در مواقعی که بخواهید چندین اسم کلاس را به صورت هم زمان مدیریت کنید اولویت دارد.

مقیدسازی سبک

سبک ها را می توانید به صورت درون خطی به همراه مقیدسازی آن ها تنظیم کنید.

سینتکس مقیدسازی سبک شبیه به مقیدسازی ویژگی است، با این تفاوت که به جای ویژگی یک عنصر داخل براکت، سینتکس آن با پیشوند style به همراه یک نقطه (.) و اسم ویژگی یک سبک CSS آغاز می شود: [style.style-property].

src/app/app.component.html

```
<button [style.color]="isSpecial ? 'red' : 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan' : 'grey'" >Save</button>
```

برخی از سبک های موجود در مقیدسازی سبک دارای فرمت واحدی هستند. در مثال زیر به صورت شرطی اندازه ی فونت بر روی واحدهای "em" و "%" تنظیم می شود.

src/app/app.component.html

```
<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>
```

با وجود اینکه این روش برای تنظیم تنها یک سبک مناسب است، اما برای تنظیم چندین سبک درون خطی به صورت هم زمان عموما بهتر است از دستورالعمل NgStyle استفاده کنید.

توجه داشته باشید که اسم ویژگی یک سبک را هم می توان با – (مانند بالا) و هم به صورت نگارش شتری (مانند fontSize) نوشت.

مقیدسازی رویداد ((event))

تا به اینجای کار دستورالعمل های مقیدسازی ای که با آن ها آشنا شده اید، داده ها را در یک جهت از یک کامپوننت به یک عنصر به جریان می انداخته اند.

کاربران فقط به صفحه نگاه نمی کنند، بلکه متن هایی را داخل کادرهای ورودی وارد می کنند، آیتم هایی را از فهرست ها انتخاب می کنند و بر روی دکمه هایی کلیک می کنند. این کارها ممکن است منجر به جریان داده در جهت مخالف شود؛ یعنی از عنصر به یک کامپوننت.

تنها راهی که می توان از طریق آن به عمل کاربر پی برد این است که به رویدادهای خاصی توجه کرد. مانند فشردن کلید، حرکت موس، کلیک کردن موس یا لمس صفحه. شما از طریق مقیدسازی رویداد در Angular می توانید توجه خود را به اعمال کاربر اعلان کنید.

سینتکس مقیدسازی رویداد، شامل یک اسم رویداد هدف داخل پرانتز در سمت چپ تساوی و یک دستور قالب نقل قول شده در سمت راست تساوی است.مقیدسازی رویداد زیر به رویدادهای کلیک دکمه توجه می کند و هر زمان که کلیکی رخ دهد، متد ()onSave کامپوننت را فراخوانی می کند:

src/app/app.component.html

```
<button (click)="onSave()">Save</button>
```

رویداد هدف

اسمی که داخل پرانتز قرار می گیرد (برای مثال (click)) بیانگر رویداد هدف است. در مثال زیر رویداد هدف، رویداد کلیک دکمه است.

src/app/app.component.html

```
<button (click)="onSave()">Save</button>
```

برخی از افراد ترجیح می دهند به عنوان جایگزین از پیشوند - on استفاده کنند که به این شیوه صورت متعارف گفته می شود:

src/app/app.component.html

```
<button on-click="onSave()">On Save</button>
```

رویدادهای عناصر معمولا هدف های رایج تری هستند، اما Angular در ابتدا به دنبال اسمی می گردد که ویژگی رویداد یک دستورالعمل شناخته شده باشد. مانند مثال زیر:

src/app/app.component.html

```
<!-- `myClick` is an event on the custom `ClickDirective` -->
```

```
<div (myClick)="clickMessage=$event" clickable>click with myClick</div>
```

به دستورالعمل myClick در بخش «دادن نام مستعار به ویژگی های ورودی و خروجی» بیشتر می پردازیم.

اگر این اسم نتواند رویداد یک عنصر یا ویژگی خروجی یک دستورالعمل شناخته شده را تطبیق دهد، در این صورت Angular خطای "unknown directive" می دهد.

دستورات $event و event handling

طی مقیدسازی رویداد، Angular برای رویداد هدف یک event handler را تنظیم می کند.

زمانی که رویداد پیش بیاید، در این صورت handler دستور قالب را اجرا می کند. معمولا این دستور شامل یک دریافت کننده است که کار آن اجرا کردن عملی در پاسخ به این رویداد است، مانند ذخیره کردن یک مقدار از کنترل HTML داخل یک مدل.

این نوع از مقیدسازی اطلاعات مربوط به رویداد از جمله مقادیر داده ها را از طریق شیء رویدادی به نام $event هدایت می کند.

شکل شیء رویداد توسط رویداد هدف مشخص می شود. اگر رویداد هدف یک رویداد عنصر بومی DOM باشد، در این صورت $event همراه با ویژگی هایی مانند target و target.value به یک شیء رویداد DOM تبدیل می شود.

مثال زیر را در نظر بگیرید:

src/app/app.component.html

```
<input [value]="currentHero.name"
    (input)="currentHero.name=$event.target.value" >
```

این کد با مقید شدن به ویژگی name ویژگی value کادر ورودی را تنظیم می کند. این کد برای توجه کردن به تغییرات اعمال شده بر این مقدار، به رویداد input کادر ورودی مقید می شود. زمانی که کاربر تغییرات را اعمال می کند، رویداد input پیش می آید و این مقیدسازی دستور بالا را داخل زمینه ای که شامل شیء رویداد DOM یا همان $event است را اجرا می کند.

متن تغییر یافته برای به روز کردن ویژگی name ، با دنبال کردن مسیر $event.target.value بازیابی می شود.

اگر این رویداد به دستورالعملی متعلق باشد (به خاطر آورید که کامپوننت ها همان دستورالعمل ها هستند)، در این صورت $event همان شکلی را به خود می گیرد که این دستورالعمل تصمیم دارد آن را تولید کند.

رویدادهای اختصاصی با EventEmitter

معمولا دستورالعمل ها با استفاده از EventEmitter رویدادهای اختصاصی را پیش می آورند. این دستورالعمل یک EventEmitter را تولید می کند و آن را به عنوان یک ویژگی نمایش می دهد. این دستورالعمل برای fire کردن یک رویداد، EventEmitter.emit(payload) را فراخوانی می کند و payload که می تواند هر چیزی باشد را به آن می دهد. دستورالعمل های مادر با مقید شدن به این ویژگی و دسترسی به این payload از طریق شیء $event به این رویداد توجه می کنند.

یک HeroDetailComponent را در نظر بگیرید که کار آن ارائه ی اطلاعات هیرو و پاسخ به اعمال کاربر است. با وجود اینکه HeroDetailComponent دارای دکمه ی delete است اما به خودی خود نمی داند چگونه این هیرو را حذف کند. بهترین کاری که می تواند انجام دهد این است که رویدادی را پیش آورد که در خواست پاک کردن کاربر را گزارش دهد.

در زیر بخش های منتخبی از این HeroDetailComponent را مشاهده می کنید:

src/app/hero-detail.component.ts (template)

```
template: `
<div>
 <img src="{{heroImageUrl}}">
 <span [style.text-decoration]="lineThrough">
  {{prefix}} {{hero?.name}}
 </span>
 <button (click)="delete()">Delete</button>
</div>`
```

src/app/hero-detail.component.ts (deleteRequest)

```
// This component makes a request but it can't actually delete a hero.
deleteRequest = new EventEmitter<Hero>();


delete() {
 this.deleteRequest.emit(this.hero);
}
```

این کامپوننت یک ویژگی deleteRequest را تعریف می کند که EventEmitter را برگشت می دهد. هر زمان که کاربر بر روی دکمه ی *delete* کلیک می کند، این کامپوننت متد ()delete را احضار می کند و به EventEmitter می گوید که یک شیء Hero را از خود خارج کند.

حالا یک کامپوننت مادر و میزبان را در نظر بگیرید که به رویداد deleteRequest مربوط به HeroDetailComponent مقید می شود.

src/app/app.component.html (event-binding-to-component)

```
<app-hero-detail (deleteRequest)="deleteHero($event)" [hero]="currentHero"></app-hero-detail>
```

زمانی که رویداد deleteRequest فایر می شود، Angular متد deleteHero مربوط به کامپوننت مادر را فرا می خواند و هیرویی که قرار است حذف شود را (خارج شده از HeroDetail) به متغیر $event می دهد.

دستورات قالب اثرات جانبی دارند

متد deleteHero اثرات جانبی دارد. زیرا یکی از هیروها را پاک می کند. اثرات جانبی دستور قالب قابل قبول و در عین حال دور از انتظار نیستند.

حذف این هیرو باعث می شود مدل به روز رسانی شود و حتی شاید تغییرات دیگر از جمله پرس و جوها و فایل های ذخیره ی موجود در یکی از سرورهای از راه دور فعال شود. این تغییرات از طریق سیستم به بخش های دیگر نفوذ می کنند و در نهایت در این view و view های دیگر نمایش داده می شوند.

مقیدسازی دو طرفه ([(...)])

اغلب اوقات ممکن است بخواهید به صورت هم زمان ویژگی یک داده را نمایش دهید و زمانی که کاربر تغییرات را اعمال می کند، این ویژگی را به روز کنید.

سمت عنصر ترکیبی از تنظیمات یک ویژگی عنصر مشخص گرفته می شود و به رویداد تغییر یک عنصر توجه می شود.

Angular برای این منظور سینتکس مقیدسازی دوطرفه ی ویژه ای [(x)] را ارائه می کند. سینتکس [(x)] براکت های مقیدسازی ویژگی ([x]) را با پرانتزهای مقیدسازی رویداد (x) ترکیب می کند.

[( )] = موز داخل جعبه

برای آن که بهتر در خاطرتان بماند که پرانتزها داخل براکت ها می روند، موزی را داخل یک جعبه تصور کنید.

زمانی که عنصر ویژگی قابل تنظیمی به نام x و رویداد متناظری به نام xChange را داشته باشد، بیان سینتکس [(x)] آسان می شود. در ادامه یک SizerComponent را می بینید که اندازه ی الگو را متناسب می کند. SizerComponent دارای یک ویژگی مقدار size به همراه یک رویداد sizeChange است.

src/app/sizer.component.ts

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
selector: 'app-sizer',
template: `
<div>
<button (click)="dec()" title="smaller">-</button>
<button (click)="inc()" title="bigger">+</button>
<label [style.font-size.px]="size">FontSize: {{size}}px</label>
</div>`
})
export class SizerComponent {
@Input() size: number | string;
@Output() sizeChange = new EventEmitter<number>();

dec() { this.resize(-1); }
inc() { this.resize(+1); }
```

```
18.
19. resize(delta: number) {
20. this.size = Math.min(40, Math.max(8, +this.size + delta));
21. this.sizeChange.emit(this.size);
22. }
23. }
```

size اولیه یکی از مقادیر ورودی از مقیدسازی ویژگی است. کلیک بر روی دکمه ها باعث می شود که size داخل حدود min و max افزایش یا کاهش یابد و بعد از آن رویداد sizeChange به همراه اندازه ی تنظیم شده پیش آید.

در ادامه مثالی را مشاهده می کنید که در آن AppComponent.fontSizePx قید دوطرفه ی SizerComponent است:

src/app/app.component.html (two-way-1)

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
<div [style.font-size.px]="fontSizePx">Resizable Text</div>
```

AppComponent.fontSizePx مقدار اولیه ی SizerComponent را تشکیل می دهد. کلیک بر روی دکمه ها باعث می شود AppComponent.fontSizePx از طریق مقیدسازی دوطرفه به روز شود. مقدار اصلاح شده ی AppComponent.fontSizePx در سراسر مقیدسازی سبک جریان می یابد و باعث می شود متن نمایش داده شده بزرگ تر یا کوچک تر شود.

سینتکس مقیدسازی دوطرفه برای مقیدسازی ویژگی و یک مقیدسازی رویداد مانند یک قند سینتکسی عمل می کند؛ به این صورت که Angular قندهای مقیدسازی SizerComponent را تجزیه می کند به صورتی که نتیجه ی زیر حاصل می شود:

src/app/app.component.html (two-way-2)

```
<app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>
```

متغیر $event ، payload رویداد SizerComponent.sizeChange را در بر می گیرد. Angular زمانی که کاربر بر روی دکمه ها کلیک می کند، مقدار $event را به AppComponent.fontSizePx تخصیص می دهد.

به وضوح مشخص است که سینتکس مقیدسازی دوطرفه در مقایسه با مقیدسازی مجزای رویداد و ویژگی آسان تر است.

استفاده از مقیدسازی دوطرفه در کنار عناصر فرم HTML مانند <input> و <select> می تواند کار شما را آسان کند، هرچند که هیچ عنصر HTML بومی ای مقدار x و الگوی رویداد xChange را درک نمی کند.

خوشبختانه دستورالعمل NgModel Angular پلی است که شما می توانید به کمک آن برای تشکیل عناصر از موقعیت سازی دو طرفه استفاده کنید.

دستورالعمل های پیش فرض

ورژن های قدیمی تر Angular به صورت پیش فرض چیزی بالغ بر 70 دستورالعمل را شامل می شد. از آن زمان تا به حال جامعه ی برنامه نویسان مشارکت بیشتری کرده اند و دستورالعمل های خصوصی بیشماری برای برنامه های داخلی ایجاد شده اند.

به بسیاری از این دستورالعمل ها در Angular نیاز ندارید. به کمک سیستم مقیدسازی کارامدتر و گویاتر Angular می توانید به همین نتایج دست پیدا کنید. چه لزومی دارد که برای مدیریت یک کلیک، یک دستورالعمل جدید ایجاد کنید، وقتی می توانید مانند زیر یک مقیدسازی ساده را بنویسید؟

src/app/app.component.html

```
<button (click)="onSave()">Save</button>
```

همچنان برای ساده کردن کارهای پیچیده می توانید ازاین دستورالعمل ها استفاده کنید. Angular هنوز هم به همراه دستورالعمل های پیش فرض به فروش می رسد. هرچند که تعداد آن ها زیاد نیست. شما به زودی دستورالعمل های مخصوص به خود را خواهید نوشت، هر چند که بازهم تعداد آن ها نیز چندان زیاد نیست.

در این بخش به مرور برخی از دستورالعمل های پیش فرض پرکاربرد می پردازیم و آن ها را به دو دسته ی دستورالعمل های صفتی و دستورالعمل های ساختاری تقسیم می کنیم.

دستورالعمل های صفتی پیش فرض

این دستورالعمل ها به رفتار عناصر HTML، صفت ها، ویژگی ها و کامپوننت ها توجه می کنند و آن ها را تغییر می دهند. این دستورالعمل ها معمولا بر روی عناصر اعمال می شوند؛ درست مانند حالتی که صفت HTML و به دنبال آن اسم HTML باشند.

در آموزش دستورالعمل های صفت به جزییات بسیاری پرداخته شده است. بسیاری از NgModule ها مانند RouterModule و FormModule دستورالعمل های صفتی مخصوص به خود را تعریف می کنند. دراین بخش به معرفی پرکاربردترین دستورالعمل های صفتی می پردازیم:

- NgClass : اضافه و حذف مجموعه ای از کلاس های CSS
- NgStyle : اضافه و حذف مجموعه ای از سبک های HTML
- NgModel : مقید سازی دو طرفه به یک عنصر فرم HTML

NgClass

برای آنکه بتوانید چگونگی نمایش عناصر را کنترل کنید، باید به صورت پویا کلاس های CSS را حذف یا اضافه کنید. برای اضافه یا حذف چندین کلاس به صورت همزمان می توانید به ngClass مقید شوید.

مقیدسازی کلاس روشی مناسب برای حذف یا اضافه کردن تنها یک کلاس است.

src/app/app.component.html

```
<!-- toggle the "special" class on/off with a property -->
<div [class.special]="isSpecial">The class binding is special</div>
```

برای حذف یا اضافه کردن چندین کلاس CSS به صورت همزمان، دستورالعمل NgClass انتخاب مناسب تری است.

سعی کنید ngClass را به یک شیء کنترلی کلید:مقدار مقید کنید. هر یک از کلیدهای شیء ، یک اسم کلاس CSS هستند و اگر این کلاس ها بنا باشد که اضافه شوند مقدار این کلید true و اگر بنا باشد که حذف شوند مقدار این کلید False می شود.

متد کامپوننت setCurrentClasses را در نظر بگیرید. در این متد ویژگی یک ویژگی کامپوننت (currentClasses) به همراه شیئی ایجاد می شود که این شیء بر اساس حالت true/false مربوط به سه ویژگی کامپوننت دیگر، این سه کلاس را حذف یا اضافه می کند.

src/app/app.component.ts

```
currentClasses: {};
setCurrentClasses() {
```

```
 // CSS classes: added/removed per current state of component properties
 this.currentClasses = {
  'saveable': this.canSave,
  'modified': !this.isUnchanged,
  'special':  this.isSpecial
 };
}
```

اضافه کردن مقیدسازی ویژگی ngClass به currentClasses باعث می شود که کلاس های این عنصر به صورت زیر تنظیم شوند :

src/app/app.component.html

```
<div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special</div>
```

فراخوانی کردن یا نکردن ()setCurrentClasses چه در ابتدا یا زمانی که ویژگی های وابستگی تغییر می کنند دست شماست.

NgStyle

شما می توانید بر اساس حالت کامپوننت به صورت پویا سبک های درون خطی را تنظیم کنید. همچنین به کمک NgStyle می توانید چندین سبک درون خطی را به صورت همزمان تنظیم کنید.

مقید سازی سبک روشی آسان برای تنظیم تنها یک مقدار سبک است.

src/app/app.component.html

```
<div [style.font-size]="isSpecial ? 'x-large' : 'smaller'" >
  This div is x-large or smaller.
</div>
```

برای تنظیم چندین سبک درون خطی به صورت همزمان، دستورالعمل NgStyle گزینه ی مناسب تری است.

سعی کنید ngStyle را به یک شیء کنترلی کلید:مقدار مقید کنید. هر یک از کلیدهای این شیء ، یک اسم سبک هستند و اگر این سبک ها بنا باشد که اضافه شوند مقدار این کلید true و اگر بنا باشد که حذف شوند مقدار این کلید False می شود.

متد کامپوننت setCurrentStyles را در نظر بگیرید. در این متد ویژگی یک ویژگی کامپوننت (currentStyles) به همراه شیئی ایجاد می شود که این شیء بر اساس حالت true/false مربوط به سه ویژگی کامپوننت دیگر، این سه سبک را حذف یا اضافه می کند.

src/app/app.component.ts

```
currentStyles: {};
setCurrentStyles() {
 // CSS styles: set per current state of component properties
 this.currentStyles = {
  'font-style': this.canSave     ? 'italic' : 'normal',
  'font-weight': !this.isUnchanged ? 'bold'   : 'normal',
  'font-size':  this.isSpecial   ? '24px'   : '12px'
 };
}
```

اضافه کردن یک مقیدسازی ویژگی ngStyle به currentStyles باعث می شود که سبک های عنصر به صورت زیر تنظیم شوند :

src/app/app.component.html

```
<div [ngStyle]="currentStyles">
 This div is initially italic, normal weight, and extra large (24px).
</div>
```

فراخوانی کردن یا نکردن ()setCurrentStyles چه در ابتدا و چه در زمان تغییر ویژگی های وابستگی دست شماست.