تعامل کامپوننت

در این آموزش به دستورالعمل های حالت های رایج ارتباطات کامپوننت می پردازیم. به گونه ای که در این حالت ها دو یا چند کامپوننت اطلاعات خود را با یکدیگر به اشتراک می گذارند.

رد کردن داده ها از مادر به فرزند به کمک مقیدسازی ورودی

HeroChildComponent دو ویژگی ورودی دارد که معمولا با دکوراتور @Input نمایش داده می شوند.

component-interaction/src/app/hero-child.component.ts

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

import { Hero } from './hero';

@Component({
selector: 'app-hero-child',
template: `
<h3>{{hero.name}} says:</h3>
<p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
`
})
export class HeroChildComponent {
@Input() hero: Hero;
@Input('master') masterName: string;
}
```

دومین @Input اسم ویژگی کامپوننت فرزند را از masterName به اسم مستعار 'master' تغییر می دهد.

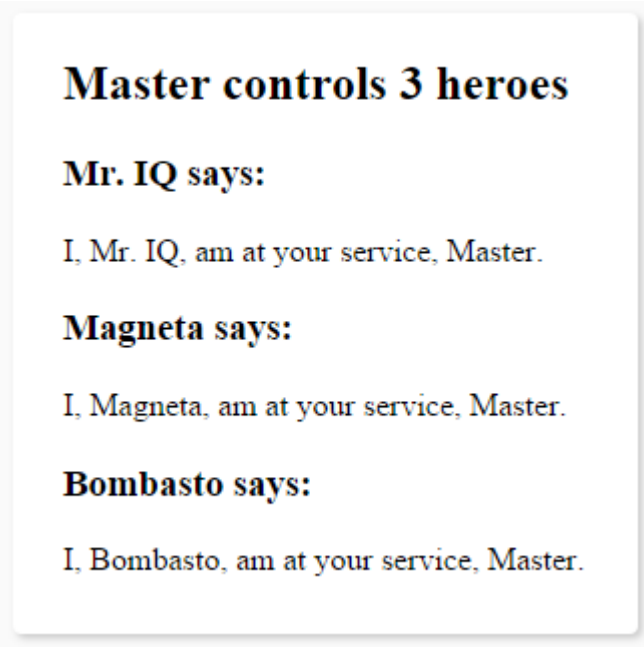
HeroChildComponent ، HeroParentComponent فرزند را به صورت تو در تو داخل یک حلقه ی تکرار *ngFor قرار می دهد و ویژگی رشته ی master آن را به اسم مستعار master فرزند و هر یک از نمونه های hero حلقه ی تکرار را به ویژگی hero فرزند مقید می کند.

component-interaction/src/app/hero-parent.component.ts

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

import { HEROES } from './hero';

```

```
@Component({
selector: 'app-hero-parent',
template: `
<h2>{{master}} controls {{heroes.length}} heroes</h2>
<app-hero-child *ngFor="let hero of heroes"
[hero]="hero"
[master]="master">
</app-hero-child>
`
})
export class HeroParentComponent {
heroes = HEROES;
master = 'Master';
}
```

نرم افزار بعد از اجرا سه هیروی زیر را نمایش می دهد:



برنامه را امتحان کنید

E2E آزمایش می کند که تمامی فرزندان طبق پیش بینی نمونه سازی و نمایش داده شوند:

component-interaction/e2e/src/app.e2e-spec.ts

```
// ...
```

```
let _heroNames = ['Mr. IQ', 'Magneta', 'Bombasto'];
let _masterName = 'Master';

it('should pass properties to children properly', function () {
let parent = element.all(by.tagName('app-hero-parent')).get(0);
let heroes = parent.all(by.tagName('app-hero-child'));

for (let i = 0; i < _heroNames.length; i++) {
let childTitle = heroes.get(i).element(by.tagName('h3')).getText();
let childDetail = heroes.get(i).element(by.tagName('p')).getText();
expect(childTitle).toEqual(_heroNames[i] + ' says:');
expect(childDetail).toContain(_masterName);
}
});
// ...
```

به کمک یک تنظیم کننده از تغییرات ویژگی ورودی جلوگیری کنید

برای قطع کردن یک مقدار حاصل از مادر و انجام اقدامات لازم برای آن از یک تنظیم کننده ی ویژگی ورودی کمک بگیرید.

تنظیم کننده ی ویژگی ورودی name موجود در NameChildComponent فرزند، فضای سفید حاصل از یک اسم را درست می کند و جای مقدار خالی را با متن پیش فرض عوض می کند.

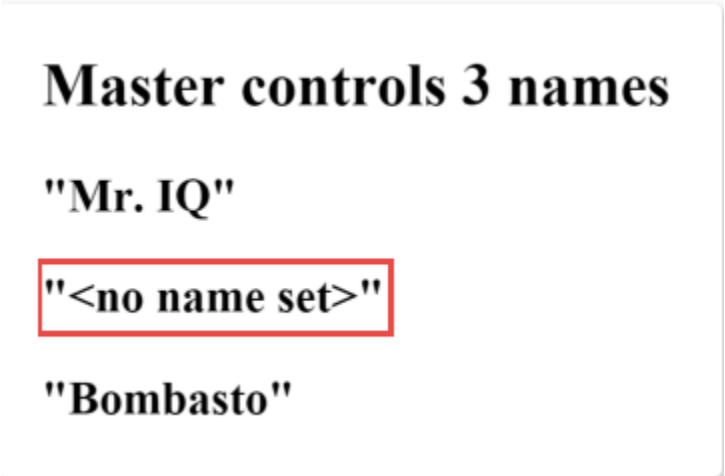
component-interaction/src/app/name-child.component.ts

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

@Component({
selector: 'app-name-child',
template: '<h3>"{{name}}"</h3>'
})
export class NameChildComponent {
private _name = '';

@Input()
set name(name: string) {
this._name = (name && name.trim()) || '<no name set>';
}

get name(): string { return this._name; }
}
```

در ادامه می توانید NameParentComponent را مشاهده کنید که تغییرات اسم شامل یک اسم به همراه تمامی فواصل را نمایش می دهد.

component-interaction/src/app/name-parent.component.ts

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

@Component({
selector: 'app-name-parent',
template: `
<h2>Master controls {{names.length}} names</h2>
<app-name-child *ngFor="let name of names" [name]="name"></app-name-child>
`
})
export class NameParentComponent {
// Displays 'Mr. IQ', '<no name set>', 'Bombasto'
names = ['Mr. IQ', ' ', ' Bombasto '];
}
```



برنامه را امتحان کنید

E2E تنظیم کننده ی ویژگی ورودی را به همراه اسامی خالی و غیرخالی امتحان می کند.

component-interaction/e2e/src/app.e2e-spec.ts

```
// ...
```

```
it('should display trimmed, non-empty names', function () {
let _nonEmptyNameIndex = 0;
let _nonEmptyName = '"Mr. IQ"';
let parent = element.all(by.tagName('app-name-parent')).get(0);
let hero = parent.all(by.tagName('app-name-child')).get(_nonEmptyNameIndex);

let displayName = hero.element(by.tagName('h3')).getText();
expect(displayName).toEqual(_nonEmptyName);
});

it('should replace empty name with default name', function () {
let _emptyNameIndex = 1;
let _defaultName = '"<no name set>"';
let parent = element.all(by.tagName('app-name-parent')).get(0);
let hero = parent.all(by.tagName('app-name-child')).get(_emptyNameIndex);

let displayName = hero.element(by.tagName('h3')).getText();
expect(displayName).toEqual(_defaultName);
});
// ...
```

جلوگیری از تغییرات ویژگی ورودی به کمک ngOnChanges()

به کمک متد ngOnChanges() رابط هوک چرخه ی عمر OnChanges مقادیر ویژگی ورودی را شناسایی کنید و اقدامات لازم را در قبال آن انجام دهید.

اگر با ویژگی های ورودی تعاملی متعددی سروکار دارید، بهتر است از این روش استفاده کنید.

در فصل «هوک های چرخه ی عمر» در رابطه با ngOnChanges() بیشتر بیاموزید.

VersionChildComponent تغییرات اعمال شده بر ویژگی های ورودی major و minor را شناسایی می کند و پیامی را می نویسد که این تغییرات در آن گزارش شده است:

component-interaction/src/app/version-child.component.ts

```
import { Component, Input, OnChanges, SimpleChange } from '@angular/core';

@Component({
selector: 'app-version-child',
template: `
<h3>Version {{major}}.{{minor}}</h3>
<h4>Change log:</h4>
<ul>
```

```
<li *ngFor="let change of changeLog">{{change}}</li>
</ul>
`
})
export class VersionChildComponent implements OnChanges {
@Input() major: number;
@Input() minor: number;
changeLog: string[] = [];

ngOnChanges(changes: {[propKey: string]: SimpleChange}) {
let log: string[] = [];
for (let propName in changes) {
let changedProp = changes[propName];
let to = JSON.stringify(changedProp.currentValue);
if (changedProp.isFirstChange()) {
log.push(`Initial value of ${propName} set to ${to}`);
} else {
let from = JSON.stringify(changedProp.previousValue);
log.push(`${propName} changed from ${from} to ${to}`);
}
}
this.changeLog.push(log.join(', '));
}
}
```

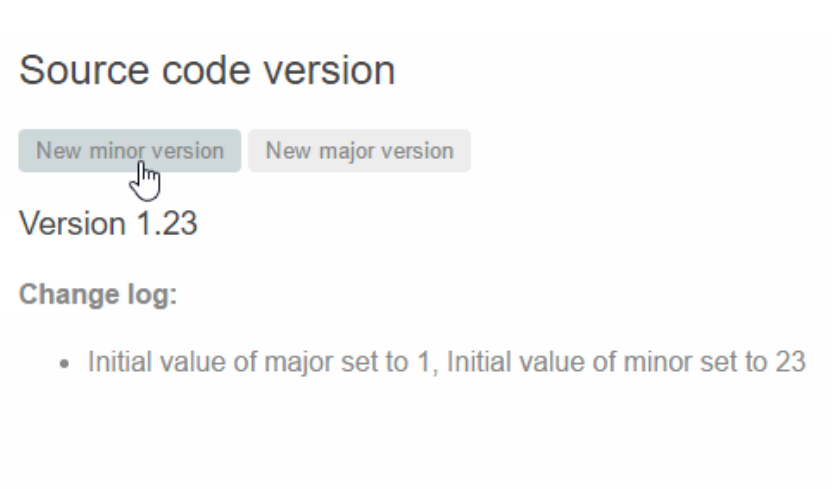
VersionParentComponent مقادیر minor و major را فراهم می کند و دکمه ها را به متدهایی مقید می کند که این مقادیر را تغییر می دهند.

component-interaction/src/app/version-parent.component.ts

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

@Component({
selector: 'app-version-parent',
template: `
<h2>Source code version</h2>
<button (click)="newMinor()">New minor version</button>
<button (click)="newMajor()">New major version</button>
<app-version-child [major]="major" [minor]="minor"></app-version-child>
`
})
export class VersionParentComponent {
major = 1;
minor = 23;

```

```
16. newMinor() {
17. this.minor++;
18. }
19.
20. newMajor() {
21. this.major++;
22. this.minor = 0;
23. }
24. }
```

در اینجا می توانید نتیجه ی فشردن یک دکمه به صورت متوالی را مشاهده کنید:



برنامه را امتحان کنید

امتحان کنید که هر دو ویژگی های ورودی در ابتدا تنظیم شده باشند و کلیک دکمه باعث فعال شدن مقادیر و فراخوان های ngOnChanges مورد انتظار شود:

component-interaction/e2e/src/app.e2e-spec.ts

```
1. // ...
2. // Test must all execute in this exact order
3. it('should set expected initial values', function () {
4. let actual = getActual();
5.
6. let initialLabel = 'Version 1.23';
7. let initialLog = 'Initial value of major set to 1, Initial value of minor set to 23';
8.
9. expect(actual.label).toBe(initialLabel);
10. expect(actual.count).toBe(1);
```

```
11. expect(actual.logs.get(0).getText()).toBe(initialLog);
12. });
13.
14. it('should set expected values after clicking \'Minor\' twice', function () {
15. let repoTag = element(by.tagName('app-version-parent'));
16. let newMinorButton = repoTag.all(by.tagName('button')).get(0);
17.
18. newMinorButton.click().then(function() {
19. newMinorButton.click().then(function() {
20. let actual = getActual();
21.
22. let labelAfter2Minor = 'Version 1.25';
23. let logAfter2Minor = 'minor changed from 24 to 25';
24.
25. expect(actual.label).toBe(labelAfter2Minor);
26. expect(actual.count).toBe(3);
27. expect(actual.logs.get(2).getText()).toBe(logAfter2Minor);
28. });
29. });
30. });
31.
32. it('should set expected values after clicking \'Major\' once', function () {
33. let repoTag = element(by.tagName('app-version-parent'));
34. let newMajorButton = repoTag.all(by.tagName('button')).get(1);
35.
36. newMajorButton.click().then(function() {
37. let actual = getActual();
38.
39. let labelAfterMajor = 'Version 2.0';
40. let logAfterMajor = 'major changed from 1 to 2, minor changed from 25 to 0';
41.
42. expect(actual.label).toBe(labelAfterMajor);
43. expect(actual.count).toBe(4);
44. expect(actual.logs.get(3).getText()).toBe(logAfterMajor);
45. });
46. });
47.
48. function getActual() {
49. let versionTag = element(by.tagName('app-version-child'));
50. let label = versionTag.element(by.tagName('h3')).getText();
51. let ul = versionTag.element((by.tagName('ul')));
52. let logs = ul.all(by.tagName('li'));
53.
54. return {
55. label: label,
```

```
logs: logs,
count: logs.count()
};
}
// ...
```

مادر به رویداد فرزند توجه می‌کند

کامپوننت فرزند یک ویژگی EventEmitter را نمایش می‌دهد و به کمک این ویژگی زمانی که اتفاقی می افتد رویدادهای emits را منتشر می‌کند. مادر به ویژگی این رویداد مقید شده و نسبت به این رویدادها واکنش نشان می‌دهد.

همان طور که در VoterComponent زیر مشاهده می‌کنید، ویژگی EventEmitter فرزند یک ویژگی خروجی است که معمولاً با @Output نشان داده می‌شود:

component-interaction/src/app/voter.component.ts

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

@Component({
selector: 'app-voter',
template: `
<h4>{{name}}</h4>
<button (click)="vote(true)" [disabled]="didVote">Agree</button>
<button (click)="vote(false)" [disabled]="didVote">Disagree</button>
`
})
export class VoterComponent {
@Input() name: string;
@Output() voted = new EventEmitter<boolean>();
didVote = false;

vote(agreed: boolean) {
this.voted.emit(agreed);
this.didVote = true;
}
}
```

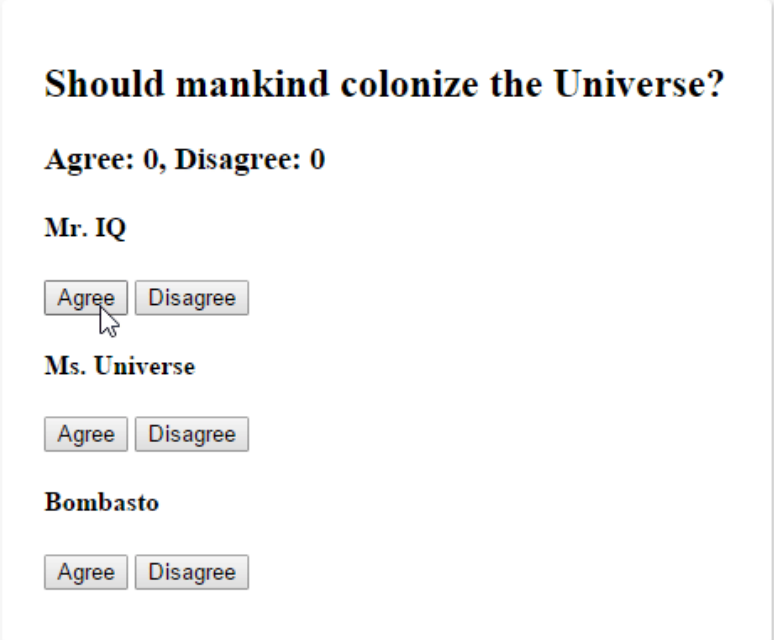
در صورتی که بر روی دکمه‌ای کلیک شود، payload بولی true یا false منتشر می‌شود.

VoteTakerComponent مادر یک event handler به نام ()onVoted را مقید می‌کند که کار آن پاسخ دادن به پی لود $event رویداد فرزند و به روز رسانی یک شمارشگر می‌باشد.

Component-interaction/src/app/votetaker.component.ts

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

@Component({
selector: 'app-vote-taker',
template: `
<h2>Should mankind colonize the Universe?</h2>
<h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
<app-voter *ngFor="let voter of voters"
[name]="voter"
(voted)="onVoted($event)">
</app-voter>
`
})
export class VoteTakerComponent {
agreed = 0;
disagreed = 0;
voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];

onVoted(agreed: 10oolean) {
agreed? this.agreed++: this.disagreed++;
}
}
```

این چهارچوب آرگومان رویداد (که توسط $event ارائه می‌شود) را به متد handler می‌دهد و این متد آن را پردازش می‌کند:



برنامه را امتحان کنید

امتحان کنید که کلیک کردن بر روی دکمه‌های *Agree* و *Disagree* باعث به روز شدن شمارشگرهای مناسب می‌شود:

component-interaction/e2e/src/app.e2e-spec.ts

```
// ...
it('should not emit the event initially', function () {
let voteLabel = element(by.tagName('app-vote-taker'))
.element(by.tagName('h3')).getText();
expect(voteLabel).toBe('Agree: 0, Disagree: 0');
});

it('should process Agree vote', function () {
let agreeButton1 = element.all(by.tagName('app-voter')).get(0)
.all(by.tagName('button')).get(0);
agreeButton1.click().then(function() {
let voteLabel = element(by.tagName('app-vote-taker'))
.element(by.tagName('h3')).getText();
expect(voteLabel).toBe('Agree: 1, Disagree: 0');
});
});
```

```
17.
18. it('should process Disagree vote', function () {
19. let agreeButton1 = element.all(by.tagName('app-voter')).get(1)
20. .all(by.tagName('button')).get(1);
21. agreeButton1.click().then(function() {
22. let voteLabel = element(by.tagName('app-vote-taker'))
23. .element(by.tagName('h3')).getText();
24. expect(voteLabel).toBe('Agree: 1, Disagree: 1');
25. });
26. });
27. // ...
```

مادر از طریق متغیر محلی با فرزند تعامل می‌کند

کامپوننت های مادر برای خواندن ویژگی‌های فرزند، یا احضار متدهای فرزند نمی‌توانند از مقیدسازی داده استفاده کنند. شما می‌توانید با ایجاد یک متغیر مرجع قالب برای عنصر فرزند و بعد از آن با اشاره به این متغیر داخل قالب مادر مانند مثال زیر هر دو کار را انجام دهید.

در مثال زیر می‌توانید CountdownTimerComponent فرزندی را مشاهده کنید که به صورت مکرر شمارش معکوس را انجام می‌دهد و موشکی را پرتاب می‌کند. این کامپوننت دارای متدهای start و stop است که کار آن‌ها کنترل کردن زمان سنج است و این کامپوننت در قالب خود پیام وضعیت شمارش معکوس را نمایش می‌دهد.

component-interaction/src/app/countdown-timer.component.ts

```
1. import { Component, OnDestroy, OnInit } from '@angular/core';
2.
3. @Component({
4. selector: 'app-countdown-timer',
5. template: '<p>{{message}}</p>'
```

```
})
export class CountdownTimerComponent implements OnInit, OnDestroy {

intervalId = 0;
message = '';
seconds = 11;

clearTimer() { clearInterval(this.intervalId); }

ngOnInit() { this.start(); }
ngOnDestroy() { this.clearTimer(); }

start() { this.countDown(); }
stop() {
this.clearTimer();
this.message = `Holding at T-${this.seconds} seconds`;
}

private countDown() {
this.clearTimer();
this.intervalId = window.setInterval(() => {
this.seconds -= 1;
if (this.seconds === 0) {
this.message = 'Blast off!';
} else {
if (this.seconds < 0) { this.seconds = 10; } // reset
this.message = `T-${this.seconds} seconds and counting`;
}
}, 1000);
}
}
```

CountdownLocalVarParentComponent که کامپوننت زمان سنج را میزبانی می‌کند را می‌توانید در زیر مشاهده کنید:

component-interaction/src/app/countdown-parent.component.ts

```
import { Component } from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';

@Component({
selector: 'app-countdown-parent-lv',
template: `
```

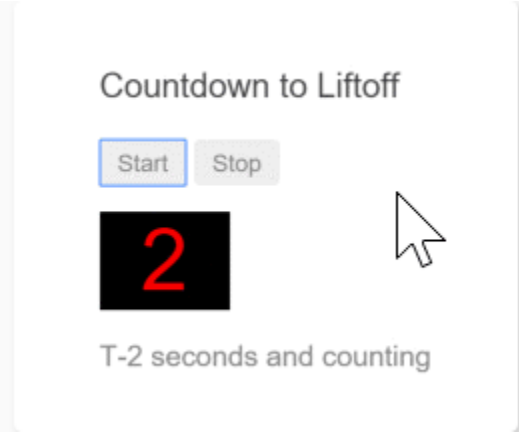
```
<h3>Countdown to Liftoff (via local variable)</h3>
<button (click)="timer.start()">Start</button>
<button (click)="timer.stop()">Stop</button>
<div class="seconds">{{timer.seconds}}</div>
<app-countdown-timer #timer></app-countdown-timer>
`,
styleUrls: ['../assets/demo.css']
})
export class CountdownLocalVarParentComponent { }
```

کامپوننت مادر نه می‌تواند به متدهای start و stop مقید شوند و نه به ویژگی seconds خود.

می‌توانید به نمایندگی از کامپوننت فرزند در تگ <countdown-timer> متغیر محلی #timer را قرار دهید. با این کار می‌توانید به کامپوننت فرزند اشاره کنید و از داخل قالب مادر به تمامی متدها یا ویژگی‌های آن دسترسی پیدا کنید.

این مثال دکمه‌های مادر را به start و stop فرزند متصل می‌کند و برای نمایش ویژگی seconds فرزند، از میانگیری استفاده می‌کند.

در اینجا می‌توانید همکاری مادر و فرزند را مشاهده کنید.



برنامه را امتحان کنید

امتحان کنید که ثانیه‌هایی که در قالب مادر نمایش داده می‌شوند، با ثانیه‌های نمایش داده شده در پیام وضعیت فرزند، مطابقت داشته باشند. همچنین امتحان کنید که کلیک بر روی دکمه‌ی stop باعث متوقف شدن شمارش معکوس شود.

component-interaction/e2e/src/app.e2e-spec.ts

```
// ...
it('timer and parent seconds should match', function () {
let parent = element(by.tagName(parentTag));
let message = parent.element(by.tagName('app-countdown-timer')).getText();
browser.sleep(10); // give `seconds` a chance to catchup with `message`
let seconds = parent.element(by.className('seconds')).getText();
expect(message).toContain(seconds);
});

it('should stop the countdown', function () {
let parent = element(by.tagName(parentTag));
let stopButton = parent.all(by.tagName('button')).get(1);

stopButton.click().then(function() {
let message = parent.element(by.tagName('app-countdown-timer')).getText();
expect(message).toContain('Holding');
});
});
// ...
```

مادر یک @ViewChild() را فراخوانی می‌کند

رویکرد متغیر محلی ساده و راحت و در عین حال محدود است. زیرا اتصال بین مادر و فرزند باید به صورت کامل داخل قالب مادر انجام شود. کامپوننت مادر به خودی خود هیچ دسترسی‌ای به فرزند ندارد.

اگر نمونه‌ای از کلاس کامپوننت مادر بایستگی مقادیر کامپوننت فرزند را بخواند یا بنویسد و یا باید متدهای کامپوننت فرزند را فراخوانی کند، در این صورت نمی‌توانید از روش متغیر محلی استفاده کنید.

زمانی که کلاس کامپوننت مادر به این نوع از دسترسی نیاز دارد، کامپوننت فرزند را به صورت *ViewChild* داخل مادر تزریق کنید.

با کمک مثال شمارش معکوس بالا این روش در مثال زیر توضیح داده شده است، نه ظاهر و نه رفتار این مثال تغییر نکرده است. همچنین CountdownTimerComponent فرزند تغییری نکرده است.

تغییر رویه از متغیر محلی به روش *ViewChild* صرفاً برای مقاصد آموزشی انجام شده است.

در ادامه می‌توانید مادر (CountdownViewChildParentComponent) را مشاهده کنید:

component-interaction/src/app/countdown-parent.component.ts

```
import { AfterViewInit, ViewChild } from '@angular/core';
import { Component } from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';

@Component({
selector: 'app-countdown-parent-vc',
template: `
<h3>Countdown to Liftoff (via ViewChild)</h3>
<button (click)="start()">Start</button>
<button (click)="stop()">Stop</button>
<div class="seconds">{{ seconds() }}</div>
<app-countdown-timer></app-countdown-timer>
`,
styleUrls: ['../assets/demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {

@ViewChild(CountdownTimerComponent)
private timerComponent: CountdownTimerComponent;

seconds() { return 0; }

ngAfterViewInit() {
// Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
// but wait a tick first to avoid one-time devMode
// unidirectional-data-flow-violation error
setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
}

start() { this.timerComponent.start(); }
stop() { this.timerComponent.stop(); }
}
```

برای آن که بتوانیم view فرزند را وارد کلاس کامپوننت مادر کنیم، به یک مقدار کار بیشتری نیاز داریم.

اول، شما باید اشاره‌های به دکوراتور ViewChild و هوک چرخه‌ی عمر AfterViewInit را import کنید.

سپس، CountdownTimerComponent فرزند را از طریق ویژگی ViewChild@ داخل ویژگی خصوصی timerComponent تزریق کنید.

متغیر محلی #timer از متادیتای کامپوننت رفته است. در عوض دکمه‌ها را به متدهای start و stop خود کامپوننت مادر مقید کنید و صدای تیک تیک ثانیه شمار را در یک میانگیری و حول متد seconds کامپوننت مادر ارائه کنید.

این متدها به صورت مستقیم به کامپوننت تایمر تزریق شده دسترسی دارند.

هوک چرخه‌ی عمر ngAfterViewInit() اطلاعات مهمی را در بر دارد. کامپوننت تایمر تا بعد از نمایش دادن view مادر توسط انگولار، در دسترس قرار نمی‌گیرد. به همین دلیل ثانیه‌ی اول تایمر از صفر شروع می‌شود.

سپس انگولار هوک چرخه‌ی عمر ngAfterViewInit را در زمانی فراخوانی می‌کند که دیگر به روز کردن نمایش view مادر ثانیه‌های شمارش معکوس دیر شده است. قانون جریان یک طرفه‌ی داده در انگولار از به روز رسانی view مادر در یک چرخه جلوگیری می‌کند. در این صورت برنامه برای آن که بتواند ثانیه‌ها را نمایش دهد، مجبور است یک دور صبر کند.

برای صبر کردن به اندازه‌ی یک تیک از setTimeout() استفاده کنید و سپس متد seconds() را به گونه‌ای بازبینی کنید که بتواند در آینده مقادیر را از کامپوننت تایمر دریافت کند.

برنامه را امتحان کنید

برای امتحان این بخش از همان شیوه‌ی امتحان شمارش معکوس که در بالا آمده است استفاده کنید.

مادر و فرزندان از طریق سرویس باهم ارتباط برقرار می‌کنند

یک کامپوننت مادر و فرزندان آن، سرویسی را به اشتراک می‌گذارند که رابط آن برقراری ارتباط دو طرفه درون خانواده را ممکن می‌کند.

حیطه‌ی نمونه‌ی سرویس، کامپوننت مادر و فرزندان آن است. کامپوننت هایی که خارج از زیر درخت این کامپوننت قرار دارند، هیچ دسترسی‌ای به این سرویس یا ارتباطات آن‌ها ندارند.

این MissionService، MissionControlComponent را به چندین فرزند AstronautComponent متصل می‌کند.

component-interaction/src/app/mission.service.ts

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

@Injectable()
export class MissionService {

// Observable string sources
private missionAnnouncedSource = new Subject<string>();
private missionConfirmedSource = new Subject<string>();

// Observable string streams
missionAnnounced$ = this.missionAnnouncedSource.asObservable();
missionConfirmed$ = this.missionConfirmedSource.asObservable();

// Service message commands
announceMission(mission: string) {
this.missionAnnouncedSource.next(mission);
}

confirmMission(astronaut: string) {
this.missionConfirmedSource.next(astronaut);
}
}
```

MissionControlComponent هم نمونه‌ی سرویسی را فراهم می‌کند که با فرزندان خود به اشتراک گذاشته است (از طریق آرایه‌ی متادیتای providers) و هم این نمونه را از طریق سازنده‌ی خود داخل خود تزریق می‌کند:

component-interaction/src/app/missioncontrol.component.ts

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

import { MissionService } from './mission.service';

@Component({
selector: 'app-mission-control',
template: `
<h2>Mission Control</h2>
```

```
<button (click)="announce()">Announce mission</button>
<app-astronaut *ngFor="let astronaut of astronauts"
[astronaut]="astronaut">
</app-astronaut>
<h3>History</h3>
<ul>
<li *ngFor="let event of history">{{event}}</li>
</ul>
`,
providers: [MissionService]
})
export class MissionControlComponent {
astronauts = ['Lovell', 'Swigert', 'Haise'];
history: string[] = [];
missions = ['Fly to the moon!',
'Fly to mars!',
'Fly to Vegas!'];
nextMission = 0;

constructor(private missionService: MissionService) {
missionService.missionConfirmed$.subscribe(
astronaut => {
this.history.push(`${astronaut} confirmed the mission`);
});
}

announce() {
let mission = this.missions[this.nextMission++];
this.missionService.announceMission(mission);
this.history.push(`Mission "${mission}" announced`);
if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
}
}
```

AstronautComponent نیز این سرویس را داخل سازنده‌ی خودش تزریق می‌کند. هر یک از AstronautComponent ها فرزندی برای MissionControlComponent هستند و به همین دلیل نمونه‌ی سرویس مادر خود را دریافت می‌کنند:

component-interaction/src/app/astronaut.component.ts

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

```

```
import { MissionService } from './mission.service';
import { Subscription } from 'rxjs';

@Component({
selector: 'app-astronaut',
template: `
<p>
{{astronaut}}: <strong>{{mission}}</strong>
<button
(click)="confirm()"
[disabled]="!announced || confirmed">
Confirm
</button>
</p>
`
})
export class AstronautComponent implements OnDestroy {
@Input() astronaut: string;
mission = '<no mission announced>';
confirmed = false;
announced = false;
subscription: Subscription;

constructor(private missionService: MissionService) {
this.subscription = missionService.missionAnnounced$.subscribe(
mission => {
this.mission = mission;
this.announced = true;
this.confirmed = false;
});
}

confirm() {
this.confirmed = true;
this.missionService.confirmMission(this.astronaut);
}

ngOnDestroy() {
// prevent memory leak when component destroyed
this.subscription.unsubscribe();
}
}
```

توجه داشته باشید زمانی که AstronautComponent از بین می‌رود، این مثال subscription و unsubscribe() را ضبط می‌کند. این کار باعث می‌شود از نشت حافظه جلوگیری شود. هیچ خطر جدی‌ای این برنامه را تهدید نمی‌کند، زیرا عمر AstronautComponent برابر با عمر خود برنامه است. البته این قضیه همیشه در برنامه‌های پیچیده‌تر صادق نخواهد بود.

شما از این شیوه‌ی مقابله‌ای در MissionControlComponent استفاده نمی‌کنید زیرا این کامپوننت درست مانند مادر عمر MissionService را کنترل می‌کند.

تاریخچه‌ی گزارشات نشان می‌دهد که پیام‌ها می‌توانند در هر دو مسیر بین MissionControlComponent مادر و AstronautComponent های فرزند رفت و آمد کنند که این فرآیند توسط این سرویس تسهیل شده است:

برنامه را امتحان کنید

کلیک بر روی دکمه‌های MissionControlComponent مادر و AstronautComponent های فرزند را امتحان کنید و مطمئن شوید که سابقه‌ی گزارش انتظارات شما را برآورده می‌کند:

component-interaction/e2e/src/app.e2e-spec.ts

```
// ...
it('should announce a mission', function () {
let missionControl = element(by.tagName('app-mission-control'));
let announceButton = missionControl.all(by.tagName('button')).get(0);
announceButton.click().then(function () {
let history = missionControl.all(by.tagName('li'));
expect(history.count()).toBe(1);
expect(history.get(0).getText()).toMatch(/Mission.* announced/);
});
});

it('should confirm the mission by Lovell', function () {
testConfirmMission(1, 2, 'Lovell');
});

it('should confirm the mission by Haise', function () {
testConfirmMission(3, 3, 'Haise');
});

it('should confirm the mission by Swigert', function () {
testConfirmMission(2, 4, 'Swigert');
});

function testConfirmMission(buttonIndex: number, expectedLogCount: number, astronaut: string) {
let _confirmedLog = ' confirmed the mission';
let missionControl = element(by.tagName('app-mission-control'));
let confirmButton = missionControl.all(by.tagName('button')).get(buttonIndex);
confirmButton.click().then(function () {
let history = missionControl.all(by.tagName('li'));
expect(history.count()).toBe(expectedLogCount);
expect(history.get(expectedLogCount - 1).getText()).toBe(astronaut + _confirmedLog);
});
}
// ...
```