

Angular Demos

Core Dynamic Features

`{{ expr }}, *ngFor, *ngIf
@Component ... @Input @Output
(upgraded to Angular 17)`

arnaud.nauwynck@gmail.com

Course Esilv 2024

This document:

<https://github.com/Arnaud-Nauwynck/presentations/web/angular-demos-2-ngFor-ngIf-component.pdf>

Outline

- templateHtml {{ expr }}
- *ngFor, @for() { }
- *ngIf, @if() { }
- @Component
- [] .. @Input field, input()
- () .. @Output changed, output()
- [()] .. Bidirectional Binding, model()
- router

Outline



templateHtml {{ expr }}

*ngFor, @for() { }

*ngIf, @if() { }

@Component

[] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

router

Interpolating {{ expr }} in template Html

In component.ts

```
@Component({  
  selector: 'app-demo2-interpolation',  
  templateUrl: './demo2-interpolation.component.html',  
})  
export class Demo2InterpolationComponent {  
  
  value1Text = "text1";  
}
```

In (template) component.html

```
value1Text: {{ value1Text }}
```

Result DOM

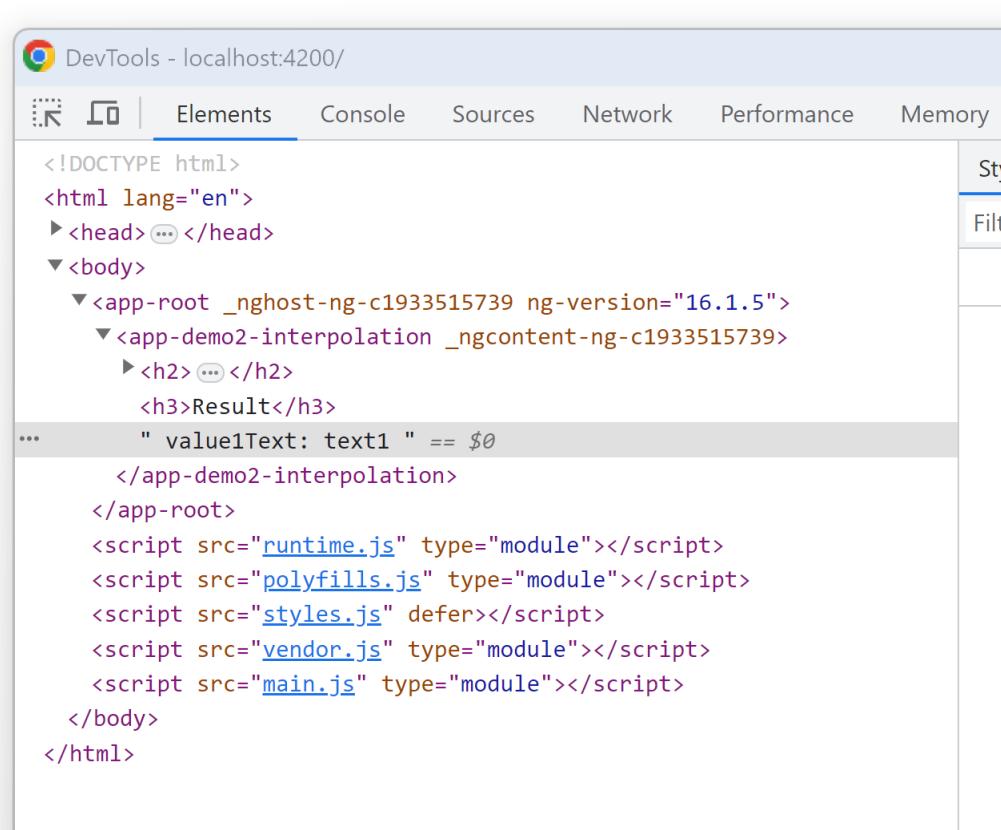
value1Text: text1

Checking with Chrome debugger (F12)

Demo {{ interpolationExpression }}

Result

value1Text: text1

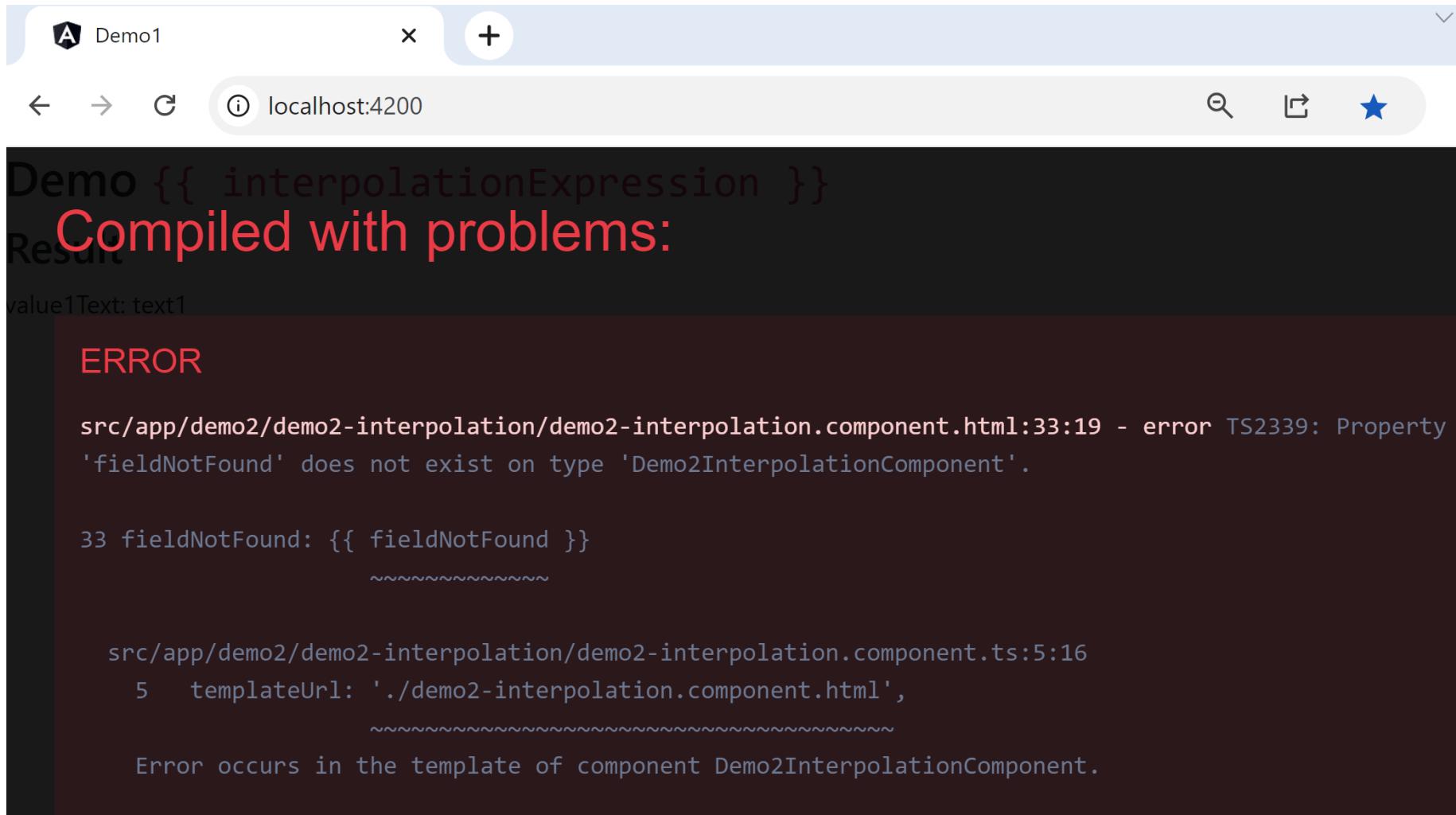


The screenshot shows the Chrome DevTools Elements tab with the following DOM structure:

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <app-root _ngc="c1933515739" ng-version="16.1.5">
      <app-demo2-interpolation _ngcontent-c1933515739>
        <h2> ...
        <h3>Result</h3>
        ...
        " value1Text: text1 " == $0
      </app-demo2-interpolation>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

The highlighted line in the DOM tree is the expression " value1Text: text1 " == \$0, which corresponds to the interpolation expression in the component template.

What if {{ fieldNotFound }} ?



Demo {{ complexExpr }}

```
value1Text = "text1";
value2Text = "text2";

value1Number = 12;
value2Number = 3.14159;

value1Bool = true;
value2Bool = false;

value10bj = { text: "objectText1", fieldInt: 1 };

get value1Getter(): string { return this.value1Text; }

function1(): string { return this.value1Text; }

{{ value1Text }}                                text1
{{ value1Number }}                             12
{{ value1Text + value2Text }}                  text1text2
{{ value1Number + value2Number }}              15.14159
{{ value1Text + ' concatenate ' + value1Number }} text1 concatenate 12
{{ (value1Number > 10)? '>10' : '<10' }}      >10
{{ value1Obj.text }}                           objectText1
{{ value1Getter }}                            text1
{{ function1() }}                            text1
change field values
value1Text:                                     text1
```

Outline



templateHtml {{ expr }}

*ngFor, @for() { }

*ngIf, @if() { }

@Component

[] .. @Input field, input()

() .. @Output changed, output()

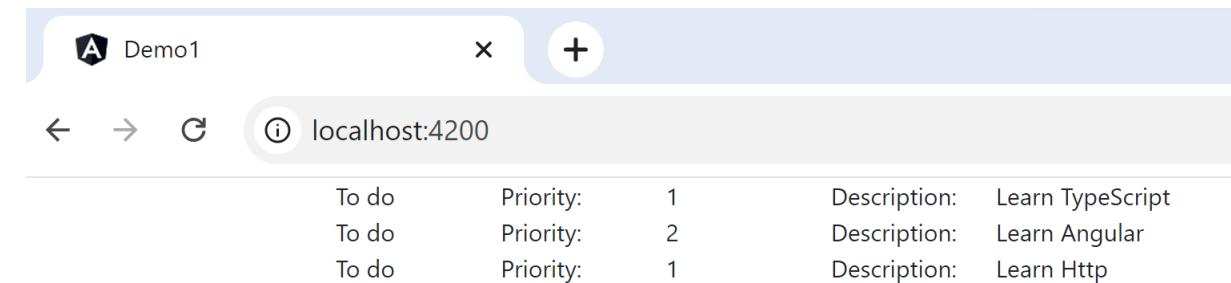
[()] .. Bidirectional Binding, model()

router

Demo *ngFor

Expected html ...
but without copy&paste

```
app.component.html x
1
2 <div class="container">
3   <div class="row">
4     <label class="col-md-1">To do</label>
5     <label class="col-md-1">Priority: </label>
6     <span class="col-md-1">1</span>
7     <label class="col-md-1">Description: </label>
8     <span class="col-md-8">Learn TypeScript</span>
9   </div>
10  <div class="row">
11    <label class="col-md-1">To do</label>
12    <label class="col-md-1">Priority: </label>
13    <span class="col-md-1">2</span>
14    <label class="col-md-1">Description: </label>
15    <span class="col-md-8">Learn Angular</span>
16  </div>
17  <div class="row">
18    <label class="col-md-1">To do</label>
19    <label class="col-md-1">Priority: </label>
20    <span class="col-md-1">1</span>
21    <label class="col-md-1">Description: </label>
22    <span class="col-md-8">Learn Http</span>
23  </div>
24 </div>
```



*ngFor: Applying MVC Design Pattern : splitting Model (json / javascript) – View (Html)

The image shows a code editor with two tabs: `app.component.ts` and `app.component.html`.

app.component.ts:

```
1 import { Component } from '@angular/core';
2
3 interface TodoItem {
4   priority: number;
5   description: string;
6 }
7
8 @Component({
9   selector: 'app-root',
10  templateUrl: './app.component.html',
11  styleUrls: ['./app.component.css']
12})
13 export class AppComponent {
14   title = 'demo1';
15
16   items: TodoItem[] = [
17     { priority: 1, description: 'learn TypeScript' },
18     { priority: 2, description: 'learn Angular' },
19     { priority: 1, description: 'learn Http' },
20   ];
}
```

app.component.html:

```
1
2
3
4
5 <div class="container">
6   <div class="row" *ngFor="let item of items">
7     <label class="col-md-1">To do</label>
8     <label class="col-md-1">Priority: </label>
9     <span class="col-md-1">{{item.priority}}</span>
10    <label class="col-md-1">Description: </label>
11    <span class="col-md-8">{{item.description}}</span>
12  </div>
13 </div>
14
15
16
17
18
19
20
```

The `*ngFor` directive in the `app.component.html` file is highlighted in yellow.

*ngFor

← → ⌂ angular.io/api/common/NgFor



DOCS

COMMUNITY

BLOG ↗

Se

Introduction

Getting started >

Understanding Angular >

Developer guides >

Best practices >

Angular tools >

Description

The `ngForOf` directive is generally used in the **shorthand form** `*ngFor`. In this form, the template to be rendered for each iteration is the content of an anchor element containing the directive.

The following example shows the shorthand syntax with some options, contained in an `` element.

```
<li *ngFor="let item of items; index as i; trackBy: trackByFn">...</li>
```



Local variable aliases:

Index, count, first / last, even / odd

trackBy:

For incremental optims and transitions

Angular 2

@for(item of list; track item.id) { ... }

```
<> todo-list.component.html ×

1 <H1>List of TODOs</H1>
2
3 <div class="container">
4   @for(item of todoService.items; track item.id) {
5     <div class='row'>
6       <label class='col-md-1'>To do</label>
7       <label class='col-md-1'>Priority: </label>
8       <span class='col-md-1'>{{item.priority}}</span>
9       <label class='col-md-1'>Description: </label>
10      <span class='col-md-5'>{{item.description}}</span>
11    </div>
12  }
13 </div>
```

Angular 7

imports: [NgForOf]

```
ts todo-list.component.ts ×

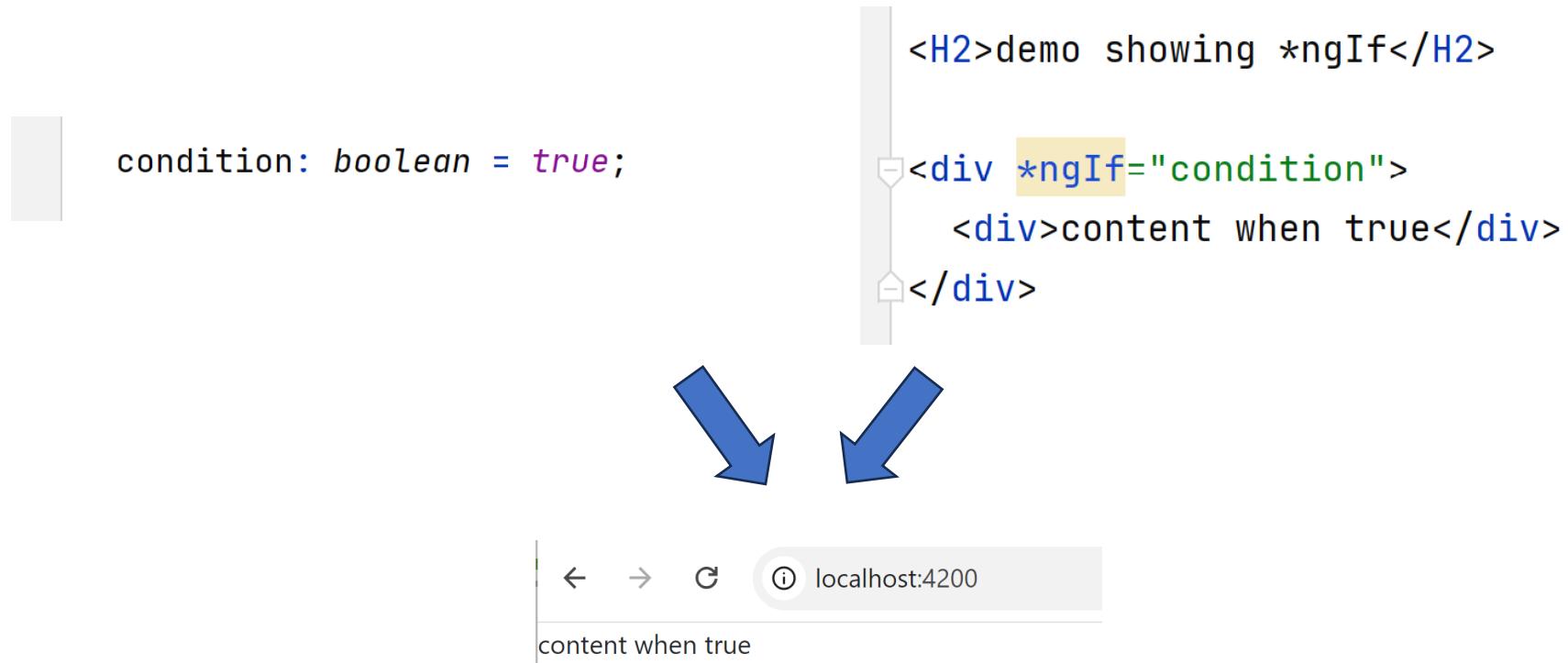
1 import { Component } from '@angular/core';
2 import { NgForOf } from "@angular/common";
3 import { TodoService } from "../todo.service";
4
5         2 usages
6 @Component({
7     selector: 'app-todo-list',
8     standalone: true,
9     imports: [ NgForOf ],
10    templateUrl: './todo-list.component.html',
11 })
12 export class TodoListComponent {
```

Outline



templateHtml {{ expr }}
*ngFor, @for() { }
***ngIf, @if() { }**
Chrome Debugger
@Component
[] .. @Input field, input()
() .. @Output changed, output()
[()] .. Bidirectional Binding, model()
router

*ngIf



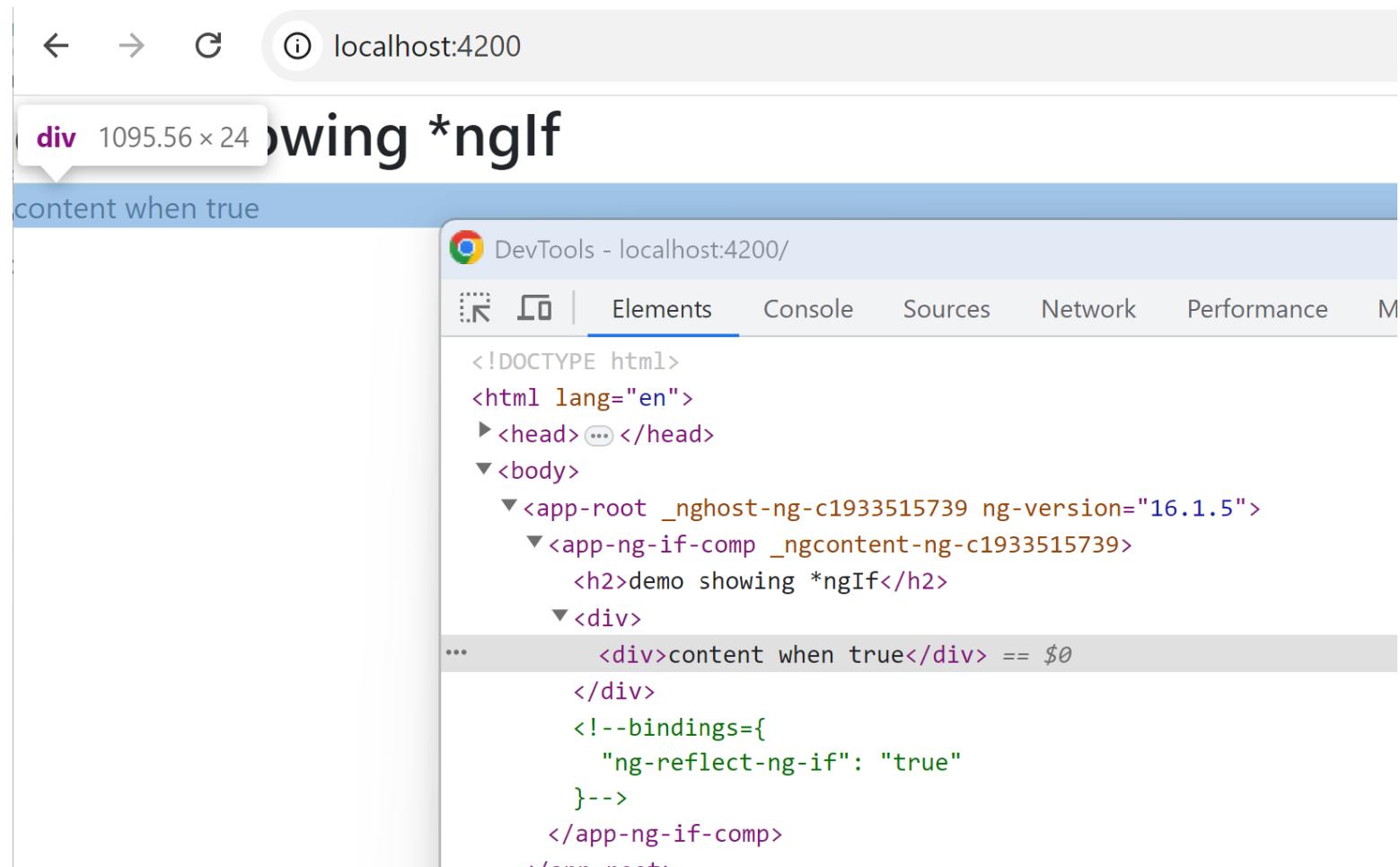
Angular 7

@if() { } @else { }

```
<> test-if.component.html <-->

1 <H2>demo *ngIf</H2>
2
3 @if(condition) {
4     <div>content when true</div>
5 } @else {
6     <div>content when false</div>
7 }
8
9 <button (click)="condition = !condition">Toggle</button>
```

Result *ngIf => attribute removed, comment added



*ngIf when false => NO content

The screenshot shows a browser window at localhost:4200. The page title is "demo showing *ngIf". Below it, there is content labeled "content when true". Another section is titled "demo showing *ngIf using !condition". To the right, the browser's DevTools Elements tab is open, displaying the DOM structure. The DOM shows an element containing an element. The first part of the element has bindings set to "true", resulting in visible content. The second part, which uses !condition, has bindings set to "false", resulting in no visible content.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <app-root _nghost-ng-c1933515739 ng-version="16.1.5">
      <app-ng-if-comp _ngcontent-ng-c1933515739>
        <h2>demo showing *ngIf</h2>
        <div>
          <div>content when true</div>
        </div>
        <!--bindings={<br/>
          "ng-reflect-ng-if": "true"<br/>
        }-->
        <h2>demo showing *ngIf using !condition</h2>
      ...
      <!--bindings={<br/>
        "ng-reflect-ng-if": "false"<br/>
      }--> == $0
    </app-ng-if-comp>
  </app-root>
```

*ngIf ... ; else contentTemplate
=> ng-template

Solution 1 : use both *ngIf="« condition »" and *ngIf="« ! condition »"

Solution 2 : use ng-template:

```
<div *ngIf="condition; else templateWhenFalse">
  <div>content when true</div>
</div>
<ng-template #templateWhenFalse>
  <div>content when false</div>
</ng-template>
```

Solution 3 : use *ngSwitch

How to avoid extra <div> on *ngIf with multiple elements content ?

```
<span>
  <div *ngIf="condition">
    <span>element 1</span>
    <span>element 2</span>
  </div>
</span>
```

Actual content:



```
<span>
  <div>
    <span>element 1</span>
    <span>element 2</span>
  </div>
</span>
```

Possible problems:
CSS, divs overhead

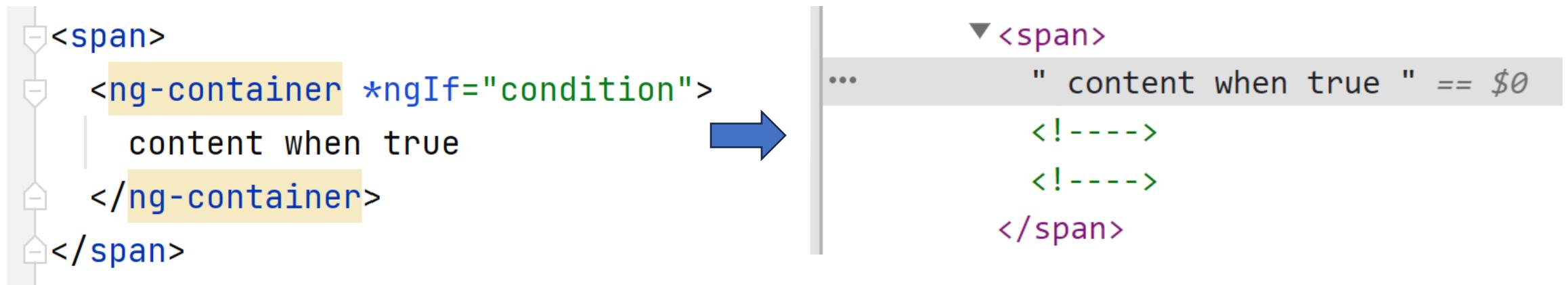
How to ???

INSTEAD... Expected content:



```
<span>
  <span>element 1</span>
  <span>element 2</span>
</span>
```

Using <ng-container *ngIf="..">



Demo *ngIf

A screenshot of a web browser window displaying a demo application for the `*ngIf` directive. The address bar shows `localhost:4200`. The page content includes several examples of `*ngIf` usage:

- demo showing *ngIf**: Shows the text "content when true".
- demo showing *ngIf using !condition**: Shows the text "content when false".
- demo showing *ngIf with else ng-template #templateId**: Shows the text "content when true".
- Button to toggle model condition**: A button labeled "Toggle condition" is shown. Below it, the text "condition = true" is displayed.
- demo showing ng-container: no extra div or span inserted**: Shows the text "content when true".

Remove <!-- bindings ..comment..--> ? « development » vs « production » conf

Default conf « development » :

```
$ ng serve  
(implicit: ng serve -c development )  
=> Has Chrome debugging comment
```

```
...      <!--bindings={  
    "ng-reflect-ng-if": "false"  
}--> == $0
```

conf « production » :

```
$ ng serve -c production  
=> Still empty placeholder comment  
but NO debugging info in it
```

```
...      <!----> == $0
```

ng serve --help ; ng serve -c production

```
$ ng serve --help  
ng serve [project]
```

Builds and serves your application, rebuilding on file changes.

Arguments:

project	The name of the project to build. Can be an application or a library.	[string] [choices: "demo1"]
---------	---	-----------------------------

Options:

-h, --help	Shows a help message for this command in the console.	[boolean]
-c, --configuration	One or more named builder configurations as a comma-separated list as specified in the "configurations" section in angular.json. The builder uses the named configurations to run the given target. For more information, see https://angular.io/guide/workspace-config#alternate-build-configurations .	[string] [choices: "development", "production"]
--allowed-hosts	List of hosts that are allowed to access the dev server.	[array]

angular.json « development » vs « production » conf

```
angular.json
61      "defaultConfiguration": "production"
62    },
63    "serve": {
64      "builder": "@angular-devkit/build-angular:dev-server",
65      "configurations": {
66        "production": {
67          "browserTarget": "demo1:build:production"
68        },
69        "development": {
70          "browserTarget": "demo1:build:development"
71        }
72      },
73      "defaultConfiguration": "development"
74    }
```

Outline

templateHtml {{ expr }}

*ngFor, @for() { }

*ngIf, @if() { }

 **@Component**

[] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

router

@Component

Goal : Avoid copy & paste,
Be able to insert re-usable html fragment
aka « component »

« component declaration » → « component re-use »

(Like « method declaration » → method call)

\$ ng generate component

(short) \$ ng g c

```
$ ng g c demo2/mycomp
CREATE src/app/demo2/mycomp/mycomp.component.html (21 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.spec.ts (559 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.ts (202 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.css (0 bytes)
UPDATE src/app/app.module.ts (793 bytes)
```

Generated @Component code

1 dir + 4 new files added

mycomp

- mycomp.component.css
- mycomp.component.html
- mycomp.component.spec.ts
- mycomp.component.ts →

+ 2 lines inserted in module
(cf next !)

app.module.ts

```
mycomp.component.html
1 <p>mycomp works!</p>

mycomp.component.ts
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-mycomp',
5   templateUrl: './mycomp.component.html',
6   styleUrls: ['./mycomp.component.css']
7 })
8 export class MycompComponent {
9
10 }
```

Using @Component

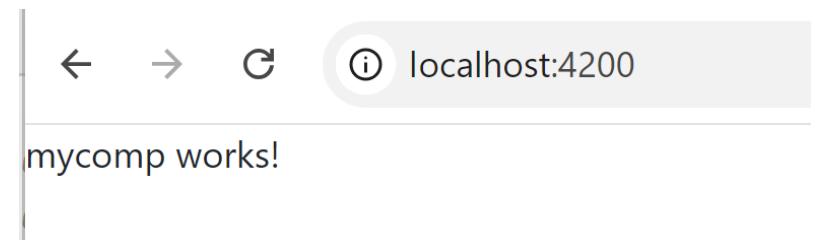
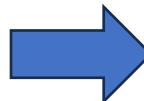
... similar to html « Custom Element »

Declare component =
assign selector (element name) to class

```
@Component({
  selector: 'app-mycomp',
  templateUrl: './mycomp.component.html',
  styleUrls: ['./mycomp.component.css']
})
export class MycompComponent {
```

Use component =
use selector as element type in html

```
<app-mycomp></app-mycomp>
```



Short @Component declaration

1/ **styleUrls** is optional (why global CSS overwrite?)



```
styleUrls: ['./mycomp.component.css']
```

2/ **templateUrl** can be inlined (why using a file for 1..2 lines?)



```
templateUrl: './mycomp.component.html',
```



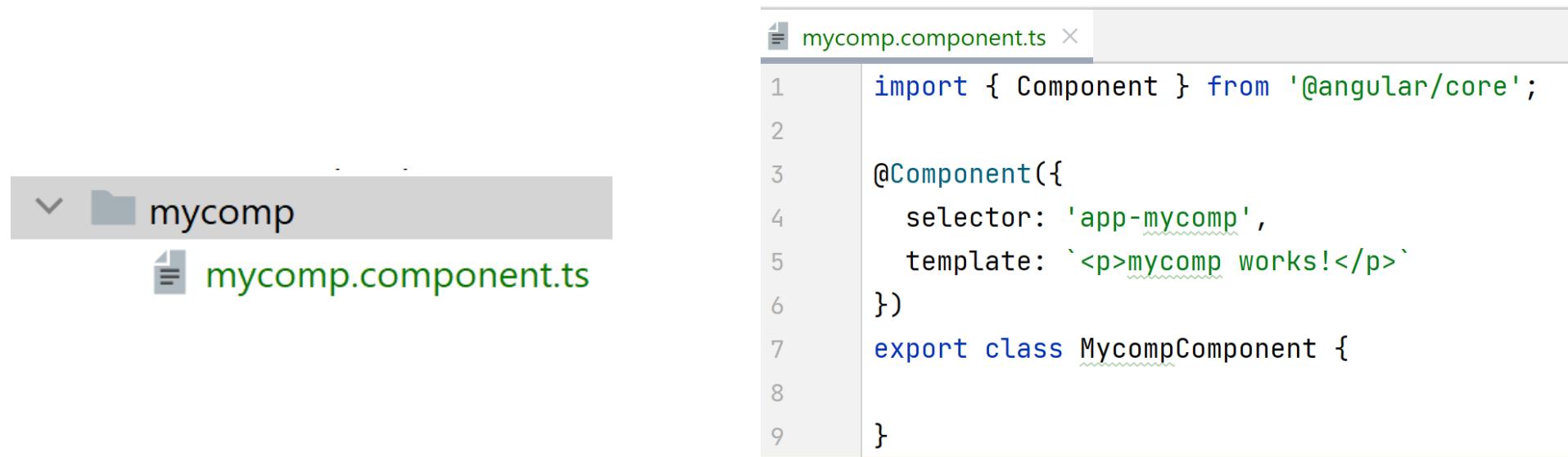
```
template: `<p>mycomp works!</p>`
```

3/ ***.spec.ts generated unit tests may be removed or completed**



(do not have assertions!)

Short @Component declaration



The image shows a code editor interface with a file tree on the left and a code editor window on the right.

File Tree:

- mycomp (selected)
- mycomp.component.ts

Code Editor (mycomp.component.ts):

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-mycomp',
5   template: `<p>mycomp works!</p>`
6 })
7 export class MycompComponent {
8
9 }
```

@Component... ADD in @NgModule declarations: [...] !!!

1 dir + 4 new files added

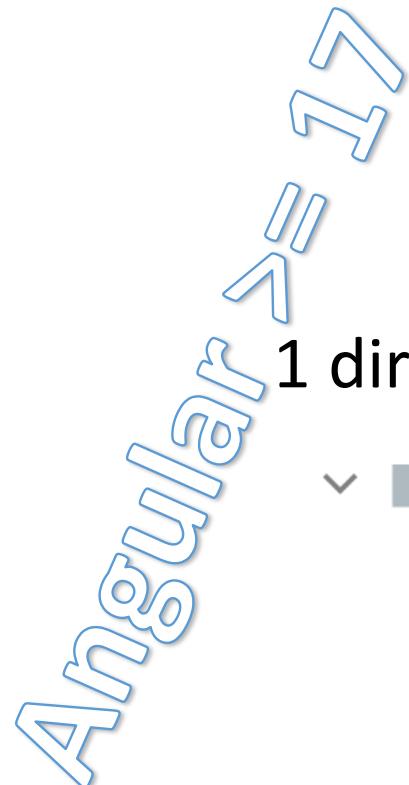
```
▼ └── mycomp
    ├── mycomp.component.css
    ├── mycomp.component.html
    ├── mycomp.component.spec.ts
    └── mycomp.component.ts
```

+ 2 lines inserted
in existing module !

app.module.ts

```
import { MycompComponent } from './demo2/mycomp/mycomp.component';

@NgModule({
  declarations: [
    AppComponent,
    MycompComponent,
  ],
  imports: [
```



@Component standalone=true

1 dir + 4 new files added

```
mycomp
  mycomp.component.css
  mycomp.component.html
  mycomp.component.spec.ts
  mycomp.component.ts
```



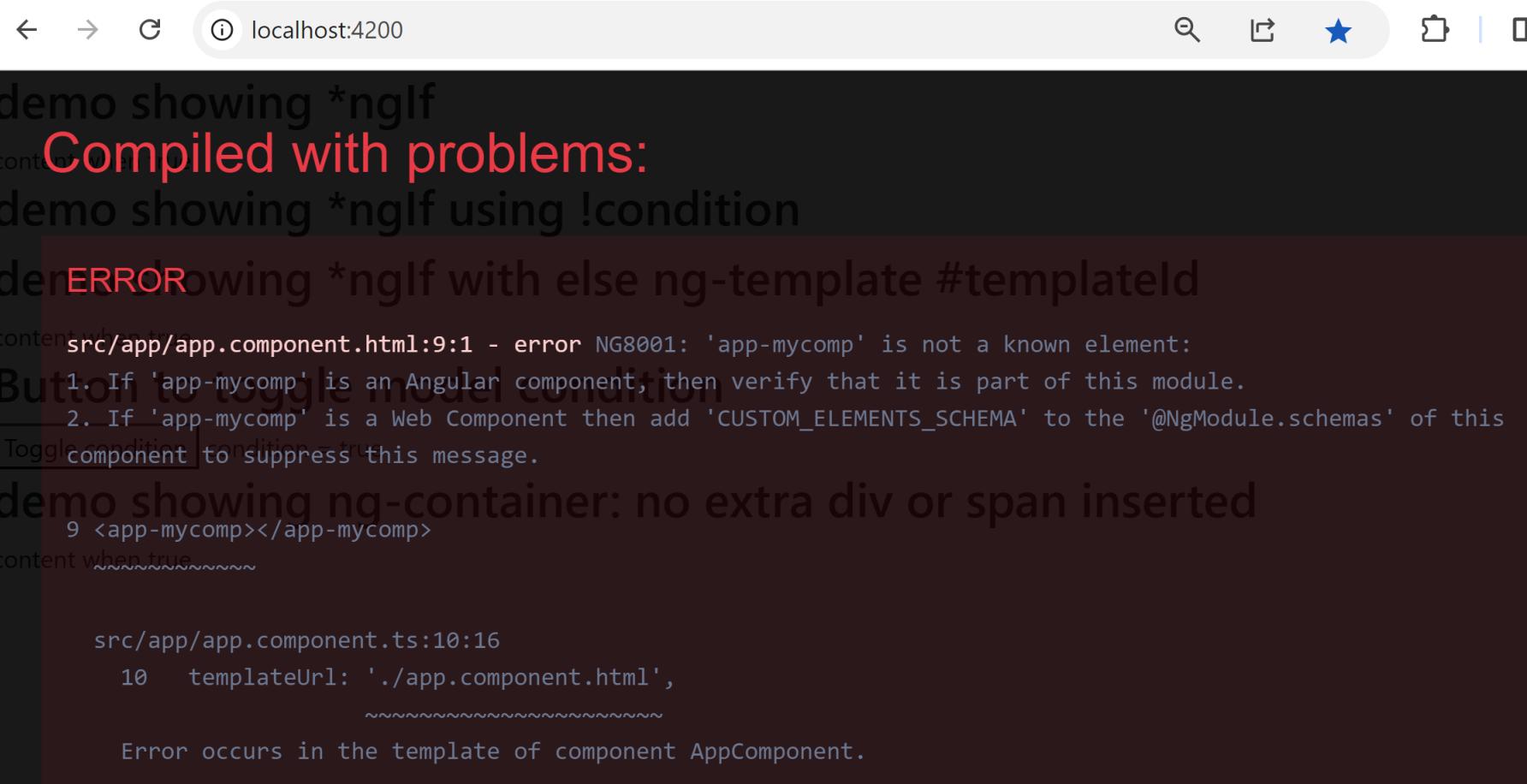
no module !

no need to add in config

app.config.ts

```
my-comp.component.ts ×
1 import { Component } from '@angular/core';
2
3 5+ usages
4 @Component({
5   selector: 'app-my-comp',
6   standalone: true,
7   imports: [],
8   templateUrl: './my-comp.component.html',
9   styleUrls: ['./my-comp.component.css']
10 })
11 export class MyCompComponent {
12 }
```

Using component, but forgetting to add in module



A screenshot of a web browser window displaying an Angular application at localhost:4200. The page title is "demo showing *ngIf". The main content area shows several examples of *ngIf usage:

- "Compiled with problems:"
- "demo showing *ngIf using !condition"
- "den~~ERRO~~ showing *ngIf with else ng-template #templateId"
- "src/app/app.component.html:9:1 - error NG8001: 'app-mycomp' is not a known element:
 - 1. If 'app-mycomp' is an Angular component, then verify that it is part of this module.
 - 2. If 'app-mycomp' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.
- "Toggle condition true~~~~~
- "demo showing ng-container: no extra div or span inserted"
- "src/app/app.component.ts:10:16
 - 10 templateUrl: './app.component.html',
~~~~~
- "Error occurs in the template of component AppComponent."

Edit, copy&paste  
... file module.ts becomes unmaintainable

⇒ Avoid modifying too many git commits / conflicts in same file

Common practices :

Solution 1/ split module in sub-modules

Solution 2/ split array declarations into constants sub-arrays

# Split declarations with array (...spreading) const

The diagram illustrates the refactoring of a component declaration array. On the left, the file `demo2-features.ts` contains a single export const declaration:

```
1 import { Comp1Component } from './comp1/comp1.component';
2 import { Demo2InterpolationComponent } from './demo2-interpolat
3 import { NgIfCompComponent } from './ng-if-comp/ng-if-comp.comp
4 import { DemoCompInputComponent } from './demo-comp-input/demo-
5 import { DemoCompInputChildComponent } from './demo-comp-input/
6
7 export const DEMO2_COMPONENTS = [
8     Demo2InterpolationComponent,
9     NgIfCompComponent,
10    MycompComponent,
11    DemoCompInputComponent,
12    Comp1Component,
13    DemoCompInputComponent, DemoCompInputChildComponent,
14];
15
```

A blue arrow points from the line `DEMO2_COMPONENTS = [` to the corresponding import statement in the code on the right. This indicates that the array is being split into two parts: a static import and a dynamic spread operator import.

On the right, the code shows the result of the refactoring:

```
import { DEMO2_COMPONENTS } from './demo2/demo2-features';
@NgModule({
  declarations: [
    AppComponent,
    ...DEMO2_COMPONENTS,
  ],
  imports: [
```

The screenshot shows a code editor window titled "app.component.html". The code contains a single line:

```
<app-my-comp></app-my-comp>
```

A tooltip has appeared over the first opening tag, containing the following message:

Component or directive matching app-my-comp element is out of scope of the current template

Below the message are two options:

- Import MyCompComponent Alt+Maj+Entrée
- More actions... Ctrl+1 Alt+Entrée

At the bottom of the tooltip, it says:

standalone component MyCompComponent

Selectors: app-my-comp

# adding <app-my-comp />

# Angular



```
import { MyComp } from '..';
@Component({ imports=[MyComp], ..
```

```
ts my-comp-parent.component.ts ×

1 import { Component } from '@angular/core';
2 import {MyCompComponent} from "../my-comp/my-comp.component";
3
4     5+ usages
5 @Component({
6     selector: 'app-my-comp-parent',
7     standalone: true,
8     imports: [
9         MyCompComponent
10    ],
11    templateUrl: './my-comp-parent.component.html',
12    styleUrls: ['./my-comp-parent.component.css'
13 })
```

```
export class MyCompParentComponent {
```

# Outline

templateHtml {{ expr }}

\*ngFor, @for() { }

\*ngIf, @if() { }

@Component

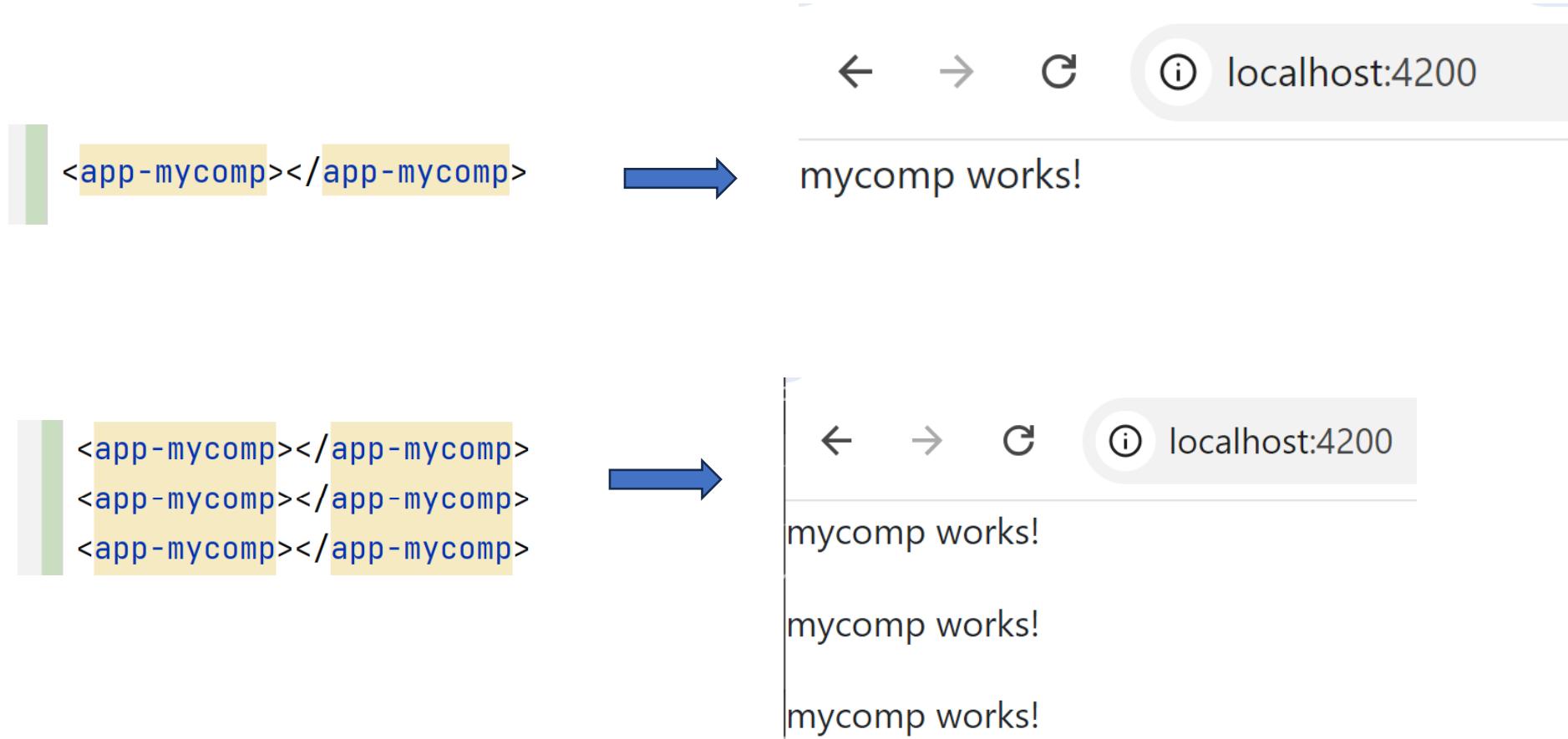
→ [] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

router

# Using repeated (static) components ... need dynamic binding



# Binding between components : pass values / capture changes

- [] One-Way data binding
  - = pass value from parent to child @input field
- ( ) change events binding
  - = handle child @output change by parent code
- [()] input+output = 2-Way data binding
  - = bind parent field to child field

# Declare @Input() field ... then use [field]=«expr»

## Child component.ts

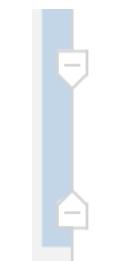
```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-demo-comp-input-child',
  templateUrl: './demo-comp-input-child.component.html'
})
export class DemoCompInputChildComponent {

  @Input()
  field: string = '';

}
```

## Parent component.html



```
<app-demo-comp-input-child  
  [field]="parentValue1"  
></app-demo-comp-input-child>
```

passing parent value to child component:  
`<child [childField]=« parentValue »></child>`

### Parent component.html

```
parentValue1: {{parentValue1}}
<app-demo-comp-input-child [field]="parentValue1">
</app-demo-comp-input-child>
```

### Parent component.ts

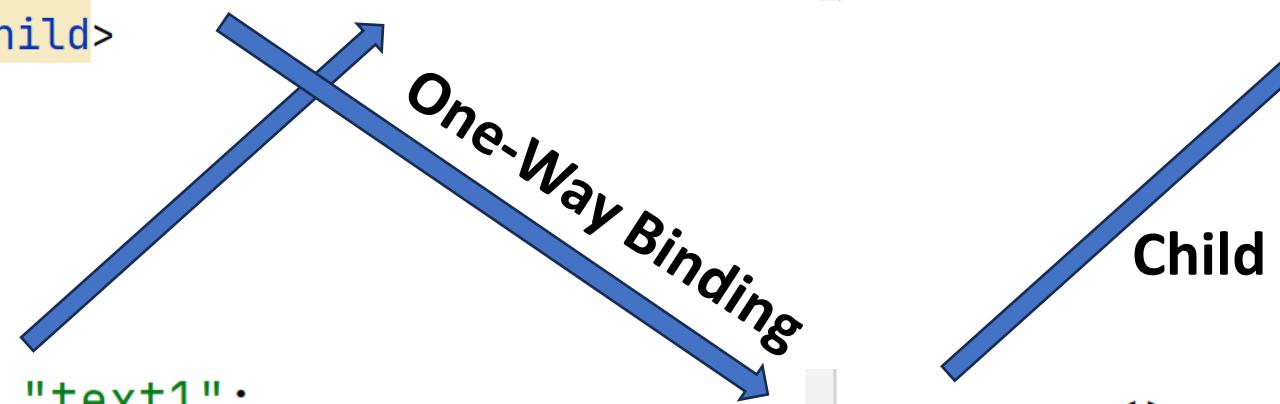
```
parentValue1 = "text1";
```

### Child component.html

```
child component, field: {{field}}
```

### Child component.ts

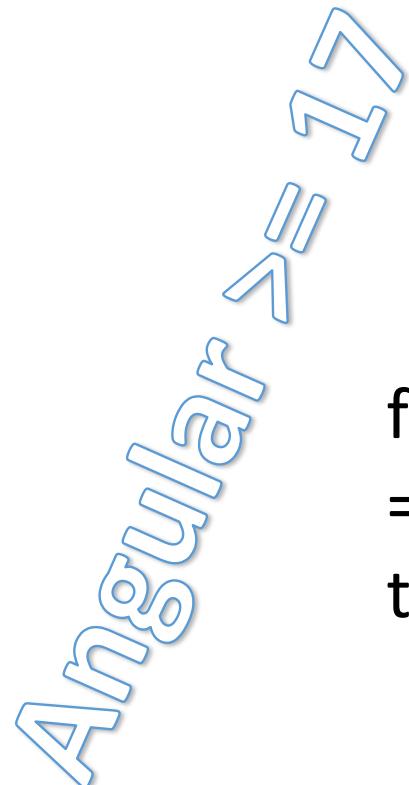
```
@Input()
field: string = '';
```



# Angular 7

instead of "@Input() field: string"  
field = input()

```
TS test-input.component.ts ×  
1 import {Component, input} from '@angular/core';  
2  
3     5+ usages  
4 @Component({  
5   selector: 'app-test-input',  
6   standalone: true,  
7   imports: [],  
8   templateUrl: './test-input.component.html',  
9 })  
10  export class TestInputComponent {  
11    field : InputSignal<string> = input(initialValue: '');  
12  }  
13
```



# :InputSignal<Type> instead of :Type

field is NOT modifiable from component  
=> it is a "read-only getter" method  
to use it, call "field()"

```
<> test-input.component.html x
1
2 field: {{field()}}
```

when forgetting parenthesis "()"

```
5 error: {{field}}
6
```

=> get  
field: hello  
error: [Input Signal: hello]

```
▲ [WARNING] NG8109: field is a function and should be invoked: field() [plugin angular-compiler]
```

```
src/app/test-input/test-input.component.html:11:9:
11 | error: {{field}}
|~~~~~
```

field = input(initialValue)

field can have initial value  
=> type is implicitly inferred

```
field2 : InputSignal<string> = input(initialValue: '');
```

text in grey is additional display by IntelliJ, but not in file ! real file content :

```
field2 = input('');
```

## field = input.required<Type>()

field can be "required"

```
field3 : InputSignal<string> = input.required<string>();
```

when forgetting to set [field3]='...'

```
<app-test-input></app-test-input>
```

```
X [ERROR] NG8008: Required input 'field3' from component TestInputComponent must be specified. [plugin angular-compiler]
```

```
src/app/test-input-parent/test-input-parent.component.html:2:0:  
2 | <app-test-input></app-test-input>  
| ~~~~~~
```

# Demo One-Way binding [childField]=« expr »

## Passing value from parent to child component: [childField]="parentValue"

```
change *child.component.ts to declare @Input field
import { Component, Input } from '@angular/core'; .. @Input() field: string = '';
```

```
change *child.component.html to use field
child component, field: {{field}}
```

```
comp using field set to parent 'parentValue1' expression:
```

```
parentValue1: text1
child component, field: text1
```

```
comp using field set to parent 'parentValue2' expression:
```

```
parentValue2: text2
child component, field: text2
```

```
comp without field set:
```

```
child component, field:
```

```
comp with field set to constant text:
```

```
child component, field: constant text
```

# Outline

templateHtml {{ expr }}

\*ngFor, @for() { }

\*ngIf, @if() { }

@Component

[] .. @Input field, input()

() .. @Output changed, output()

[()] .. Bidirectional Binding, model()

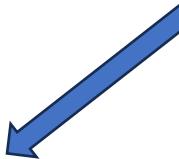
router



# Using button callback: (click)=« code... »

```
<button (click)="onClick()">Click</button>
```

```
onClick() {  
  console.log("button clicked!");  
}
```

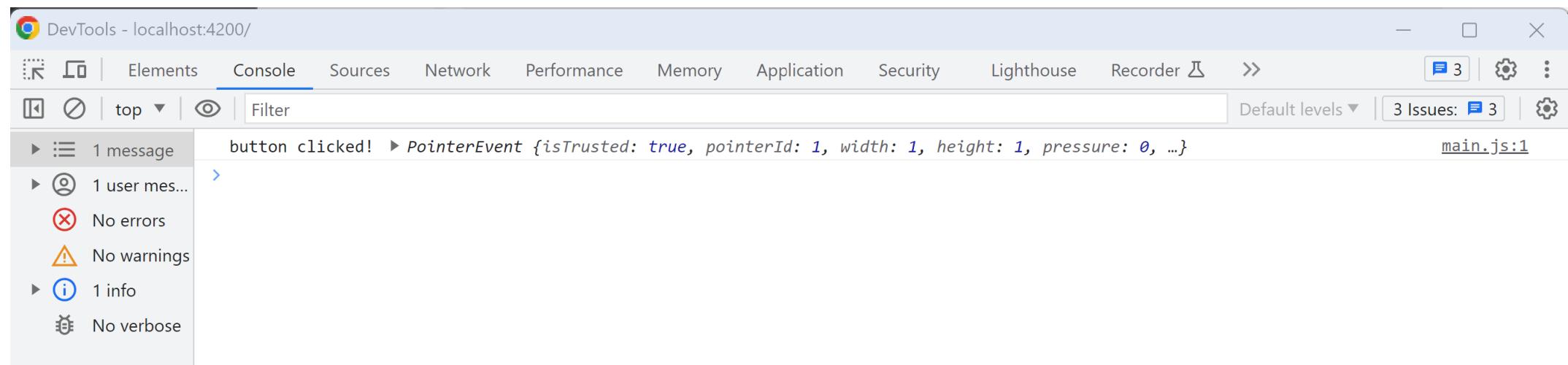


# (click) using implicit parameter « \$event »

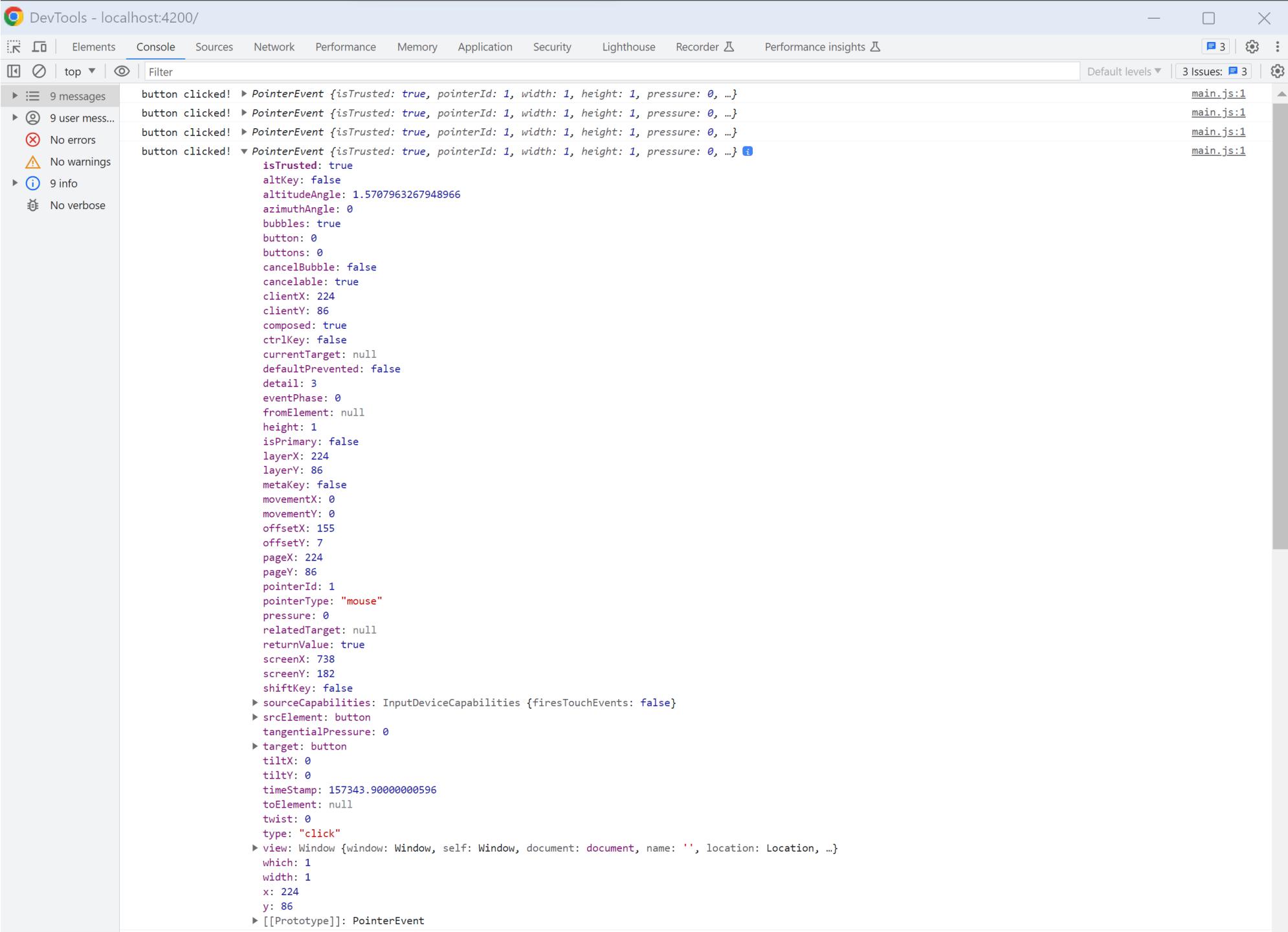
```
<button (click)="onClick($event)">Click</button>
```

```
onClick($event: Event) {  
  console.log("button clicked!", $event);  
}
```

# Click Demo ... log in Chrome F12 > Console



# \$event detail



# Chrome Debug breakpoint

The screenshot shows the Google Chrome DevTools interface with the Sources tab selected. On the left, the file tree displays the project structure under 'localhost:4200'. The file 'demo-comp-output-child.component.ts' is open, showing a TypeScript code snippet. A breakpoint is set on line 10, which contains the line: `console.log("button clicked!", $event);`. The code is syntax-highlighted in red and blue. The right side of the interface features the 'Breakpoints' sidebar. It indicates that the script is 'Paused on breakpoint'. The 'Breakpoints' section shows a single entry for 'demo-comp-output-child.component.ts' with the breakpoint line highlighted. The 'Scope' section shows the current context with variables like `this`, `$event`, and `onClick`. The 'Call Stack' section shows the execution path from the breakpoint to the global scope. Other sections include 'Watch', 'Breakpoints', 'Pause on uncaught exceptions', 'Pause on caught exceptions', and various network and performance-related tabs.

```
import { Component, Output } from '@angular/core';
@Component({
  selector: 'app-demo-comp-output-child',
  templateUrl: './demo-comp-output-child.component.html'
})
export class DemoCompOutputChildComponent {
  onClick($event: Event) {
    console.log("button clicked!", $event);
  }
}
```

DevTools - localhost:4200/

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

Page Filesystem Overrides Content scripts > :

localhost:4200

top

localhost:4200

(index)

main.js

polyfills.js

runtime.js

styles.js

vendor.js

styles.css

Selenium IDE

webpack://

node\_modules

src

app

demo2

comp1

demo2-interpolation

demo-comp-input

demo-comp-output

demo-comp-output-child.component.html

demo-comp-output-child.component.ts

demo-comp-output.component.html

demo-comp-output.component.ts

mycomp

ng-if-comp

app-routing.module.ts

app.component.html

app.component.ts

app.module.ts

main.ts

styles.css

styles.css

webpack

demo-comp-output-child.component.ts

10

Paused on breakpoint

Watch

Breakpoints

Pause on uncaught exceptions

Pause on caught exceptions

demo-comp-output-child.component.ts

console.log("button clicked!", \$event);

Scope

Local

this: DemoCompOutputChildComponent

\$event: PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1}

Closure (274)

Script

Global

Call Stack

Show ignore-listed frames

onClick

DemoCompOutputChildComponent\_Template\_button\_click\_6\_listener

DemoCompOutputChildComponent\_Template

Zone - HTMLButtonElement.addEventListener:click (async)

DemoCompOutputChildComponent\_Template

Promise.then (async)

4913

\_webpack\_exec\_

(anonymous)

(anonymous)

(anonymous)

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

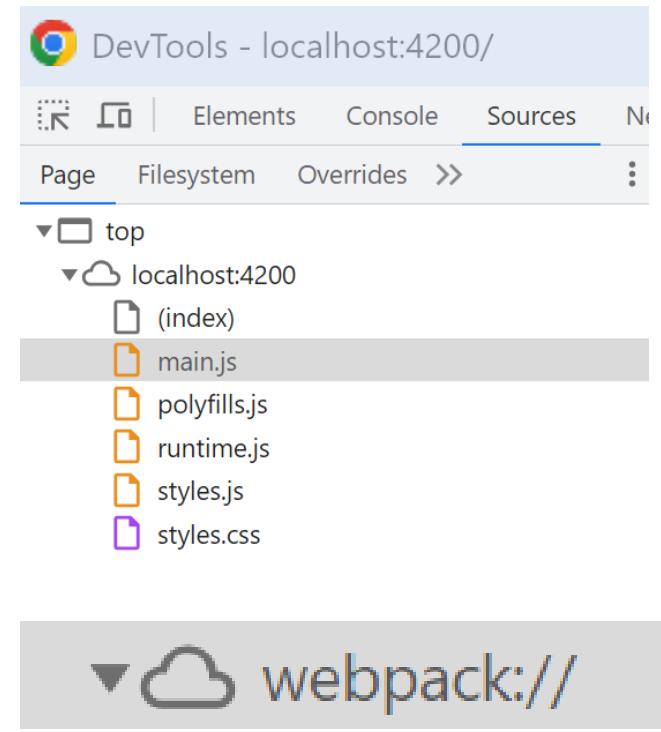
Event Listener Breakpoints

CSP Violation Breakpoints

Notice .. « ng serve -c production »  
=> NO source « .map » code !

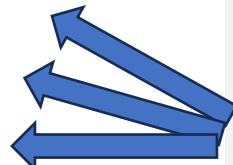
Can not browse to put breakpoint (obfuscated code)

Remark  
Source .map



# (click) on button is built-in ... you can define your own @Output()

**@Output() =  
Observable  
subscribed  
from parent  
components**



```
import { Component, Output } from '@angular/core';
import { Observable, Subject } from 'rxjs';

export interface FieldEvent {
  message: string;
}

@Component({
  selector: 'app-demo-comp-output-child',
  templateUrl: './demo-comp-output-child.component.html'
})
export class DemoCompOutputChildComponent {

  @Output()
  fieldChange = new Subject<FieldEvent>();

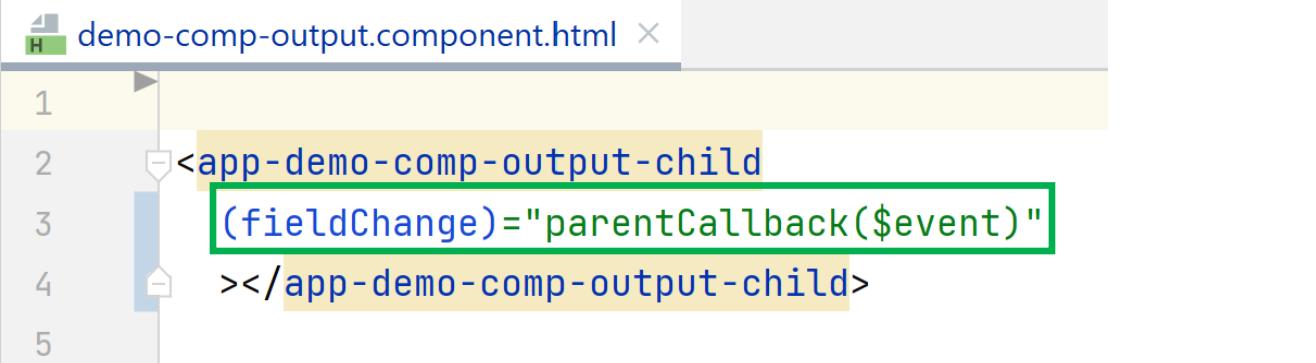
  onClickCustomEmit() {
    this.fieldChange.next({ message: "event message" });
  }

  onClick($event: Event) {
    console.log("button clicked!", $event);
  }
}
```

.next(event)

Triggered in child component (click)

# Subscribe to @Output : (childFieldChange)=« parentCallback (\$event) »



```
demo-comp-output.component.html
1
2 <app-demo-comp-output-child
3   (fieldChange)="parentCallback($event)"
4 ></app-demo-comp-output-child>
5
```

```
demo-comp-output.component.ts
1 import { Component } from '@angular/core';
2 import { FieldEvent } from './demo-comp-output-child.component';
3
4 @Component({
5   selector: 'app-demo-comp-output',
6   templateUrl: './demo-comp-output.component.html'
7 })
8 export class DemoCompOutputComponent {
9
10   parentCallback(event: FieldEvent) {
11     console.log("DemoCompOutputComponent.parentCallback", event);
12   }
13
14 }
```

# Angular 2+

## field = output<Type>()

```
TS test-output.component.ts ×

1 import {Component, output} from '@angular/core';
2
3     5+ usages
4 @Component({
5   selector: 'app-test-output',
6   standalone: true,
7   imports: [],
8   templateUrl: './test-output.component.html',
9 })
10
11 export class TestOutputComponent {
12
13   field: OutputEmitterRef<string> = output<string>();
14
15   onClick(): void {
16     console.log('click child => emit field');
17     this.field.emit({ value: 'emit event' });
18   }
19 }
```

# Chrome Debugger > « Call Stack » (parent ts)

The screenshot shows the Chrome Debugger interface with the 'Call Stack' panel highlighted by a green box. A blue arrow points from the left margin of the code editor towards the Call Stack panel.

**demo-comp-output.component.ts**

```
1 import { Component } from '@angular/core';
2 import { FieldEvent } from './demo-comp-output-child.component';
3
4 @Component({
5   selector: 'app-demo-comp-output',
6   templateUrl: './demo-comp-output.component.html'
7 })
8 export class DemoCompOutputComponent {
9
10   parentCallback(event: FieldEvent) { event = {message: 'event message'}
11     console.log("DemoCompOutputComponent.parentCallback", event);
12   }
13
14 }
15
16
17
18 // parentCallback2(event: FieldEvent) {
19 //   console.log("DemoCompOutputComponent.parentCallback2", event);
20 // }
```

**Paused on breakpoint**

demo-comp-output.component.ts

- ✓ console.log("DemoCompOutputComponent.parentCallback", event); 11

Scope

Local

- this: DemoCompOutputComponent
- event: {message: 'event message'}
- Closure (8371)
- Script
- Global

Call Stack

- parentCallback demo-comp-output.component.ts:11
  - DemoCompOutputComponent\_template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_lis... demo-comp-output.component.html:3
  - onClickCustomEmit demo-comp-output.component.ts:18
  - DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener demo-comp-output.component.html:5
- Zone - HTMLButtonElement.addEventListener:click (async)
  - DemoCompOutputChildComponent\_Template demo-comp-output.component.html:5
  - Promise.then (async)
    - 4913 main.ts:8
    - \_webpack\_exec\_ main.ts:9
    - (anonymous) main.ts:9
    - (anonymous) main.ts:9
    - (anonymous) main.js:2
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints

# Debugger Call Stack .. Parent caller[1] (parent html)

The screenshot shows a debugger interface with two main panes. On the left is a code editor with two tabs: `demo-comp-output.component.ts` and `demo-comp-outpu...component.html`. The `demo-comp-outpu...component.html` tab contains the following template code:

```
1 <app-demo-comp-output-child
2   (fieldChange)="parentCallback($event)"
3 ></app-demo-comp-output-child>
4
5
6
7
8 <!--
9 <app-demo-comp-output-child
10  (fieldChange)="parentCallback2($event)"
11 ></app-demo-comp-output-child>
12 -->
13
```

The line `(fieldChange)="parentCallback($event)"` is highlighted in blue. On the right is a debugger sidebar with the following sections:

- Paused on breakpoint**: Shows a yellow bar at the top.
- Breakpoints**: A list of breakpoints, with one entry for `demo-comp-output.component.ts` checked.
- Scope**: Shows the variable `$event` with the value `{message: 'event message'}`.
- Call Stack**: A list of frames:
  - `parentCallback` (demo-comp-output.component.ts:11)
  - `DemoCompOutputComponent_Template_app_demo_comp_output_child_fieldChange_0_listener` (demo-comp-outpu...ponent.html:3) - This frame is highlighted with a green border and has a blue arrow pointing from the code editor towards it.
  - `onClickCustomEmit` (demo-comp-outpu...ponent.ts:18)
  - `DemoCompOutputChildComponent_Template_button_click_4_listener` (demo-comp-outpu...ponent.html:5)
  - Zone - HTMLButtonElement.addEventListener:click (async)**
  - `DemoCompOutputChildComponent_Template` (demo-comp-outpu...ponent.html:5)
  - Promise.then (async)**
  - `4913` (main.ts:8)
  - `_webpack_exec_` (main.ts:9)
  - `(anonymous)` (main.ts:9)
  - `(anonymous)` (main.ts:9)
  - `(anonymous)` (main.js:2)

# Debugger Call Stack .. Parent[2] caller (child ts)

The screenshot shows a debugger interface with two main panes. The left pane is a code editor for `demo-comp-output-child.component.ts`, displaying TypeScript code for a child component. The right pane is a debugger sidebar showing the call stack.

**Code Editor (Left):**

```
1 import { Component, Output } from '@angular/core';
2 import { Observable, Subject } from 'rxjs';
3
4 export interface FieldEvent {
5   message: string;
6 }
7
8 @Component({
9   selector: 'app-demo-comp-output-child',
10  templateUrl: './demo-comp-output-child.component.html'
11})
12 export class DemoCompOutputChildComponent {
13
14   @Output()
15   fieldChange = new Subject<FieldEvent>();
16
17   onClickCustomEmit() {
18     this.fieldChange.next({ message: "event message"});
19   }
20
21   onClick($event: Event) {
22     console.log("button clicked!", $event);
23   }
24
25 }
26
```

**Debugger Sidebar (Right):**

- Paused on breakpoint
- Watch
- Breakpoints
  - Pause on uncaught exceptions
  - Pause on caught exceptions
  - demo-comp-output.component.ts
    - console.log("DemoCompOutputComponent.parentCallback", event); 11
- Scope
- Local
  - this: DemoCompOutputChildComponent
- Closure (274)
- Script
- Global
- Call Stack
  - Show ignore-listed frames
  - parentCallback demo-comp-output.component.ts:11
  - DemoCompOutputComponent\_Template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_listener demo-comp-output-child.component.html:2
  - onClickCustomEmit demo-comp-output.component.ts:18
  - DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener demo-comp-output-child.component.html:5
  - Zone - HTMLElement.addEventListener:click (async)
  - DemoCompOutputChildComponent\_Template demo-comp-output-child.component.html:5
  - Promise.then (async)
  - 4913 main.ts:8
  - \_webpack\_exec\_ main.ts:9
  - (anonymous) main.ts:9
  - (anonymous) main.ts:9
  - (anonymous) main.js:2

# Debugger Call Stack ... Parent[3] caller (child html)

The screenshot shows a debugger interface with two main panes. The left pane displays the code for `demo-comp-output-child.component.ts`, which contains a container with a row, a label, and a button. The right pane shows the debugger controls and the call stack.

**Call Stack:**

- Zone - `HTMLButtonElement.addEventListener:click (async)`
  - `DemoCompOutputChildComponent_Template` `demo-comp-output-child.component.html:5`
- `Promise.then (async)`
  - `4913` `main.ts:8`
  - `_webpack_exec_` `main.ts:9`
  - (anonymous) `main.ts:9`
  - (anonymous) `main.ts:9`
  - (anonymous) `main.js:2`

A green box highlights the entry point in the call stack, and a blue arrow points from the left pane towards the highlighted entry point in the call stack.

```
1 <div class="container">
2   <div class="row">
3     <label class="col-md-4">click to emit change</label>
4     <button type="button" class="col-md-2 btn btn-primary" (click)="onClickCustomEmit()">Emit Event</button>
5   </div>
6 </div>
7 <p>demo-comp-output-child works!</p>
8
9
10
11
12
```

**Breakpoints:**

- Paused on breakpoint
- Watch
- Breakpoints
  - Pause on uncaught exceptions
  - Pause on caught exceptions
- demo-comp-output.component.ts
  - console.log("DemoCompOutputComponent.parentCallback", event); `11`
- Scope
- Local
  - this: undefined
- Closure (`DemoCompOutputChildComponent_Template`)
- Closure (274)
- Script
- Global
- Call Stack
- Show ignore-listed frames
  - parentCallback `demo-comp-output-child.component.ts:11`
  - DemoCompOutputComponent\_Template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_listener `demo-comp-output-child.component.html:3`
  - onClickCustomEmit `demo-comp-output-child.component.ts:18`
  - DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener `demo-comp-output-child.component.html:5`

() = Change binding  
reverse way: child -> parent

### Parent component.html

```
<app-demo-comp-output-child  
  (fieldChange)="parentCallback($event)"  
></app-demo-comp-output-child>
```

### Parent component.ts

```
parentCallback(event: FieldEvent) {  
  console.log("parentCallback()", event);  
}
```

*Change Binding*

### Child component.html

```
<button (click)="onClickCustomEmit()">(
```

### Child component.ts

```
onClickCustomEmit() {  
  this.fieldChange.next({ message: "event message" });  
}  
  
@Output()  
fieldChange = new Subject<FieldEvent>();
```

# (Reminder) [] = One-Way data binding

## Parent component.html

```
parentValue1: {{parentValue1}}  
<app-demo-comp-input-child [field]="parentValue1">  
</app-demo-comp-input-child>
```

## Parent component.ts

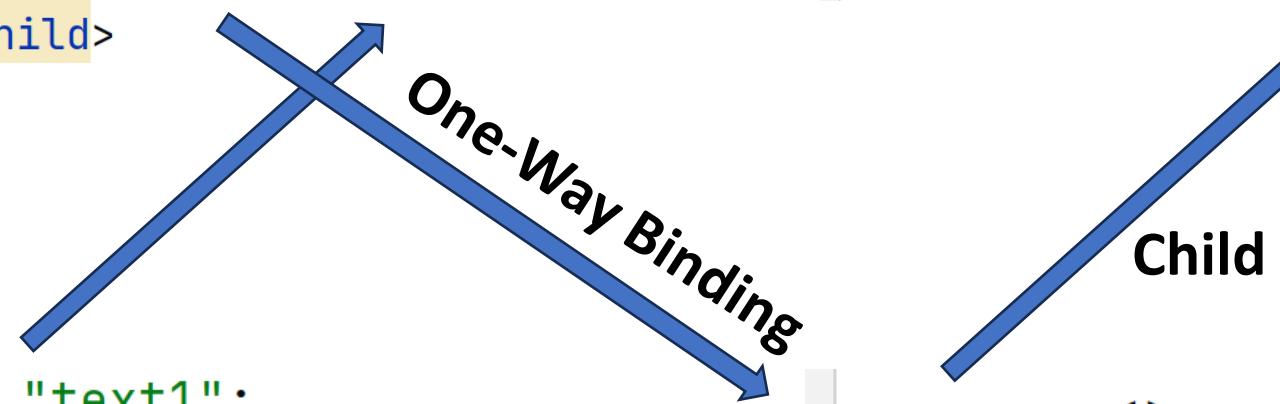
```
parentValue1 = "text1";
```

## Child component.html

```
child component, field: {{field}}
```

## Child component.ts

```
@Input()  
field: string = '';
```



# Outline

templateHtml {{ expr }}

\*ngFor, @for() { }

\*ngIf, @if() { }

@Component

[] .. @Input field, input()

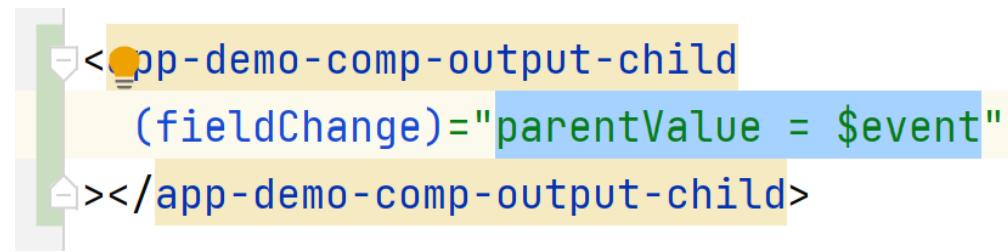
() .. @Output changed, output()

**[()] .. Bidirectional Binding, model()**

router



When event is value type (string .. From Subject<string>)  
Using callback as setter method



# [()] = 2-Way Data Binding

1

[]

[childField]=« parentField »

+

+

+

1

()

(childFieldChange)=« parentField = \$event; »

=

=

=

2

[()]

[(childField)]=« parentField »

# 2-Way Data Binding

naming Convention constraint:  
« field » <-> « fieldChange »

event type constraint:  
\$event is field value

# Mnemonic Tips: [()] instead of ([])

[()] is named = « Banana Box »

... = banana inside a box



# `[(childField)]="parentField"`

## Parent component.html

```
<app-demo2-bidirectionbinding-child  
  [(childField)]="parentField"  
></app-demo2-bidirectionbinding-child>
```

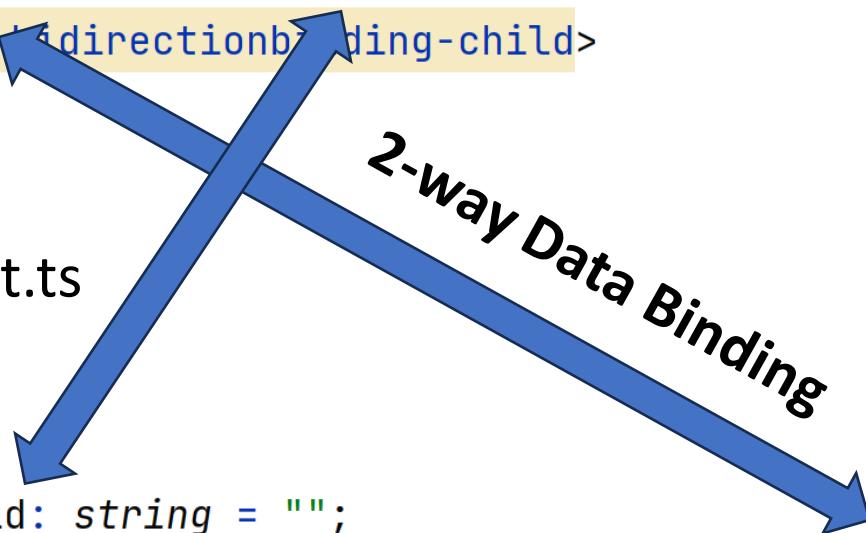
## Parent component.ts

```
parentField: string = "";
```

## Child component.html

## Child component.ts

```
@Input()  
childField: string = "";  
  
@Output()  
childFieldChange = new Subject<string>();
```



Angular 7

# field = model<Type>()

```
ts test-model.component.ts ×

1 import {Component, model} from '@angular/core';
2
3 5+ usages
4 @Component({
5   selector: 'app-test-model',
6   standalone: true,
7   imports: [],
8   templateUrl: './test-model.component.html'
9 })
10
11 export class TestModelComponent {
12
13   field : ModelSignal<string> = model<string>( initialValue: 'hello');
14
15   1 usage
16   onClick(): void {
17     this.field.set(this.field() + 'x');
18   }
19 }
```

# [(ngModel)] ... using FormsModule

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  declarations: [  
    AppComponent,  
    ...DEMO2_COMPONENTS,  
,  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
     FormsModule,  
  ]  
})
```

```
<input type="text" [(ngModel)]="field">
```

```
<input type="text" [(ngModel)]="field"  
      (ngModelChange)="onInputChange()">
```

# Demo 2-way data binding

## Parent .html

```
<input class="col-md-8" type="text"  
      [(ngModel)]="parentField">
```

```
<app-demo2-bidirectionbinding-child  
  [(childField)]="parentField"  
></app-demo2-bidirectionbinding-child>
```

## Parent .ts

```
parentField: string = "";
```

## Child .html

```
<label class="col-md-2">childField </label>  
<input class="col-md-8" type="text"  
      [(ngModel)]="childField" (ngModelChange)="onInputChange()">
```

## Child .ts

```
import { Component, Input, Output } from '@angular/core';  
import {Subject} from 'rxjs';  
  
@Component({  
  selector: 'app-demo2-bidirectionbinding-child',  
  templateUrl: './demo2-bidirectionbinding-child.component.html'  
})  
export class Demo2BidirectionbindingChildComponent {  
  
  @Input()  
  childField: string = "";  
  
  @Output()  
  childFieldChange = new Subject<string>();  
  
  onInputChange() {  
    this.childFieldChange.next(this.childField);  
  }  
}
```

# Demo 2-way data binding

A screenshot of a web browser window displaying a demonstration of two-way data binding. The address bar shows the URL `localhost:4200`. The page content includes a parent field and two child components.

parentField:

-

|                     |            |            |  |
|---------------------|------------|------------|--|
| child component (1) | childField | some value |  |
| child component (2) | childField | some value |  |

# Outline

templateHtml {{ expr }}  
\*ngFor, @for() { }  
\*ngIf, @if() { }  
@Component  
[] .. @Input field, input()  
() .. @Output changed, output()  
[()] .. Bidirectional Binding, model()



**router**

# Routes

URL **/pageX** => **@Component X**

URL **/pageY** => **@Component Y**

May use « hash »:

URL **/#/pageX** => **@Component X**

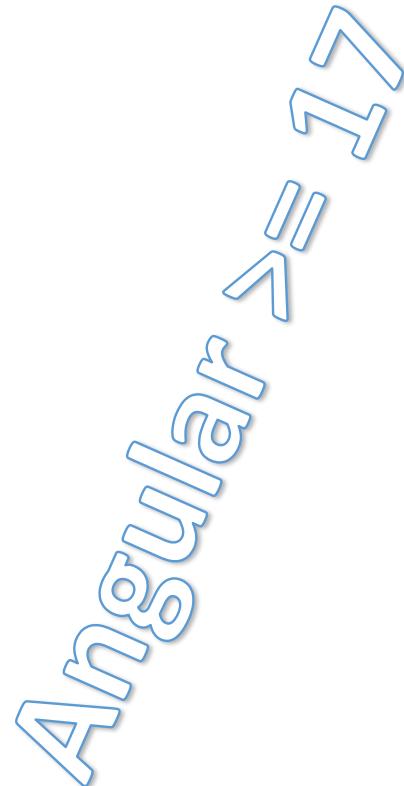
May use Child-Path and Param:

URL **/#/pageX/childY/id1234**

# RouterModule.forRoot(routes)

```
app-routing.module.ts ×
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3
4 import { Page1Component } from './demo2/page1/page1.component';
5 import { Page2Component } from './demo2/page2/page2.component';
6 import { Comp1Component } from './demo2/comp1/comp1.component';
7
8 const routes: Routes = [
9   {path: "page1", component: Page1Component },
10  {path: "page2", component: Page2Component },
11  {path: "demo2-comp1", component: Comp1Component },
12];
13
14 @NgModule({
15   imports: [RouterModule.forRoot(routes)],
16   exports: [RouterModule]
17 })
18 export class AppRoutingModule { }
```

```
app.component.html ×
1 <router-outlet></router-outlet>
```



# providers: [ provideRouter(routes), ]

```
TS app.config.ts ×

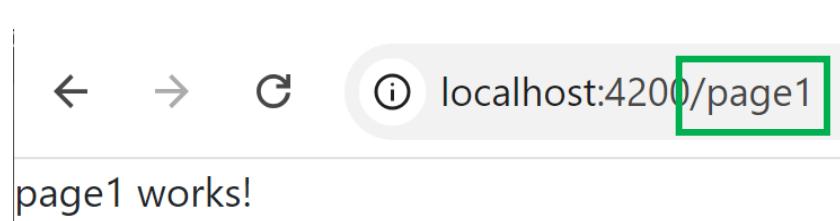
1 import {ApplicationConfig, importProvidersFrom} from '@angular/core';
2 import { provideRouter } from '@angular/router';
3
4 import { routes } from './app.routes';
5 import {provideHttpClient} from "@angular/common/http";
6
7 export const appConfig: ApplicationConfig = {
8   providers: [
9     provideRouter(routes),
10    provideHttpClient(),
11    importProvidersFrom(
12      // legacy NgModule
13    )
14  ]
15};
```

# Demo Router

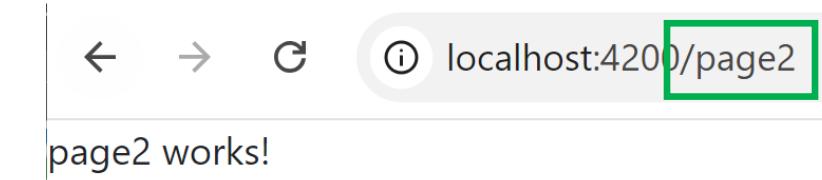
```
$ ng g c demo2/page1
CREATE src/app/demo2/page1/page1.component.html (20 bytes)
CREATE src/app/demo2/page1/page1.component.spec.ts (552 bytes)
CREATE src/app/demo2/page1/page1.component.ts (198 bytes)
CREATE src/app/demo2/page1/page1.component.css (0 bytes)
UPDATE src/app/app.module.ts (792 bytes)
```

```
$ ng g c demo2/page2
CREATE src/app/demo2/page2/page2.component.html (20 bytes)
CREATE src/app/demo2/page2/page2.component.spec.ts (552 bytes)
CREATE src/app/demo2/page2/page2.component.ts (198 bytes)
CREATE src/app/demo2/page2/page2.component.css (0 bytes)
UPDATE src/app/app.module.ts (876 bytes)
```

```
const routes: Routes = [
  {path: "page1", component: Page1Component },
  {path: "page2", component: Page2Component },
];
```



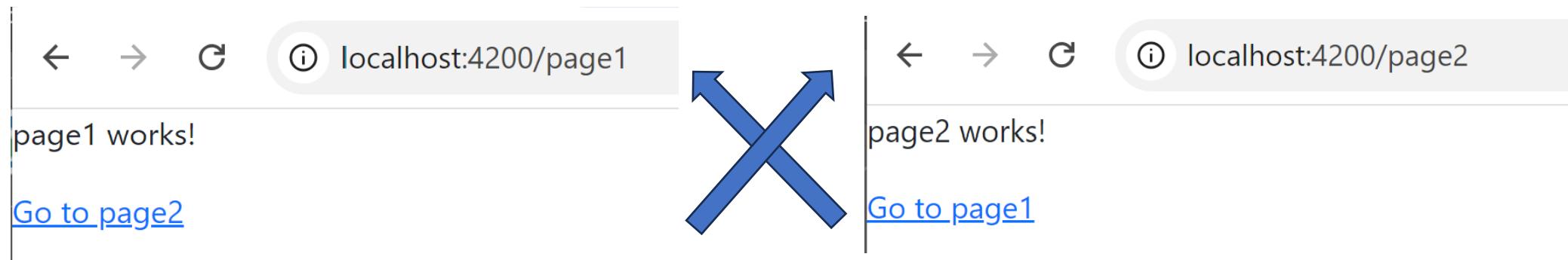
```
const routes: Routes = [
  {path: "page1", component: Page1Component },
  {path: "page2", component: Page2Component },
];
```



# Navigation : <a routerLink=<< /page >>>

```
page1.component.html ×  
1 <p>page1 works!</p>  
2  
3 <a routerLink="/page2">Go to page2</a>  
4
```

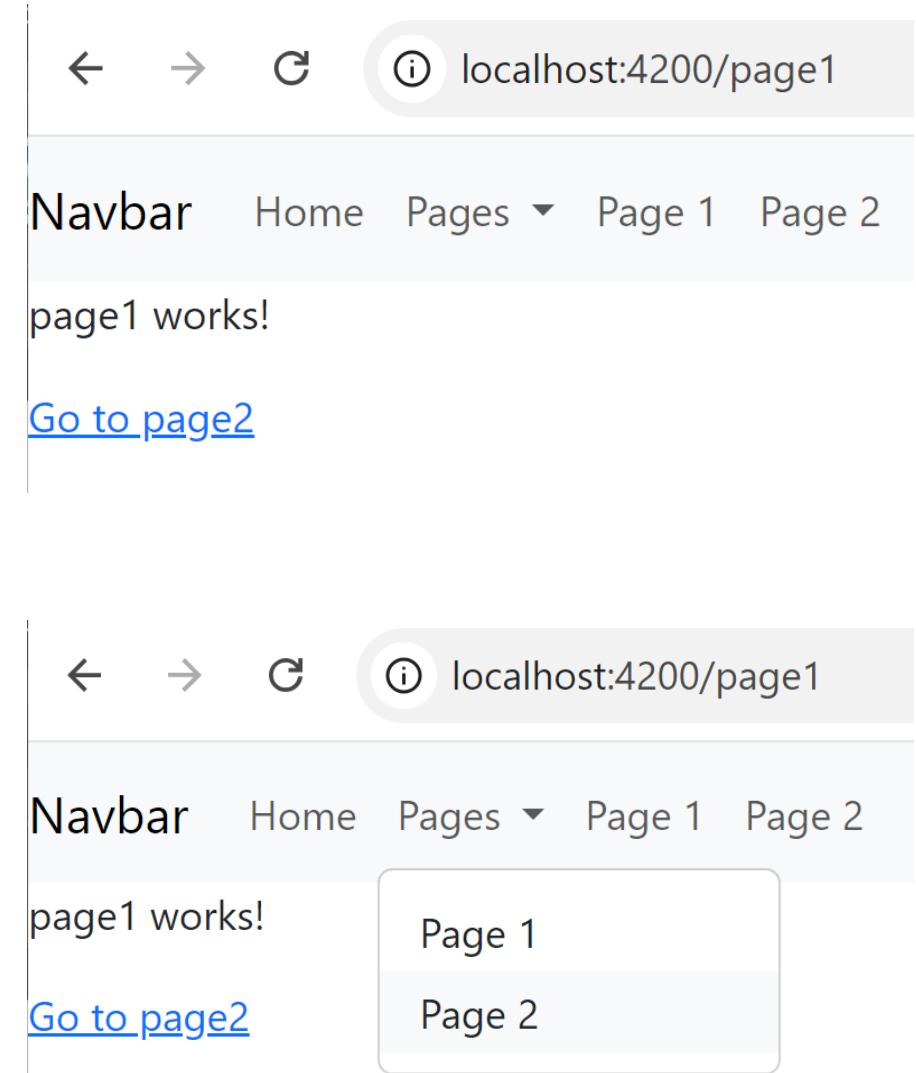
```
page2.component.html ×  
1 <p>page2 works!</p>  
2  
3 <a routerLink="/page1">Go to page1</a>  
4
```



# Main page = NavBar menu + <router-outlet> (+ footer)

```
app.component.html
```

```
16  <nav class="navbar navbar-expand-lg navbar-light bg-light">
17    <a class="navbar-brand" href="#">Navbar</a>
18    <ul class="navbar-nav mr-auto">
19      <li class="nav-item active">
20        <a class="nav-link" href="#">Home</a>
21      </li>
22      <li ngbDropdown class="nav-item">
23        <a ngbDropdownToggle id="navbarMenu1" class="nav-link" role="button">
24          Pages
25        </a>
26        <div ngbDropdownMenu aria-labelledby="navbarMenu1" class="dropdown-menu" >
27          <a class="dropdown-item" routerLink="/page1">Page 1</a>
28          <a class="dropdown-item" routerLink="/page2">Page 2</a>
29        </div>
30      </li>
31      <li class="nav-item">
32        <a class="nav-link" routerLink="/page1">Page 1</a>
33      </li>
34      <li class="nav-item">
35        <a class="nav-link" routerLink="/page2">Page 2</a>
36      </li>
37    </ul>
38  </nav>
39
40  <router-outlet></router-outlet>
41
```



# Get ActivatedRoute path & param from URL

```
import { ActivatedRoute, Router } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-page3',
  templateUrl: './page3.component.html',
  styleUrls: ['./page3.component.css']
})
export class Page3Component {

  initialId: string|null;
  currentId: string|null;
  paramSubscription!: Subscription;

  constructor(private activatedRoute: ActivatedRoute, private router: Router) {
    this.initialId = activatedRoute.snapshot.paramMap.get('id');

    console.log('Page3.constructor, initialId', this.initialId);
    this.currentId = this.initialId;

    this.paramSubscription = activatedRoute.paramMap.subscribe(paramMap => {
      this.currentId = paramMap.get('id');
      console.log("param change, currentId", this.currentId);
    });
  }
}
```

# Navigate to Route & change param

```
import { Router } from '@angular/router';

constructor(private activatedRoute: ActivatedRoute, private router: Router) {
  ...
}

onChangeValue() {
  this.router.navigate(['/page3', this.value]);
}

onClickGotoPage2() {
  this.router.navigate(['/page2']);
}
```

# Storing data in @Service ... to keep value while changing page

\$ ng generate service

```
$ ng g s appStore
CREATE src/app/app-store.service.spec.ts (368 bytes)
CREATE src/app/app-store.service.ts (137 bytes)
```



The image shows a code editor window with the tab title "app-store.service.ts". The code inside the editor is:

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class AppStoreService {
7
8   globalValue: string = 'global text';
9
10  constructor() { }
11}
```

# @Injectable Service in @Component constructor

```
service-page4.component.ts ×
1 import { Component } from '@angular/core';
2 import { AppStoreService } from '../../../../../app-store.service';
3
4 @Component({
5   selector: 'app-service-page4',
6   templateUrl: './service-page4.component.html',
7   styleUrls: ['./service-page4.component.css']
8 })
9 export class ServicePage4Component {
10
11   constructor(protected appStoreService: AppStoreService) {
12   }
13
14 }
```

Angular 7

# myService = inject(MyService)

```
TS todo-list.component.ts ×

1 import {Component, inject} from '@angular/core';
2 import { NgForOf } from "@angular/common";
3 import { TodoService } from "../todo.service";
4
5         2 usages
6 @Component({
7     selector: 'app-todo-list',
8     standalone: true,
9     imports: [ NgForOf ],
10    templateUrl: './todo-list.component.html',
11 })
12
13    public const todoService: TodoService = inject(TodoService);
14
15    // constructor(public todoService: TodoService) {
16    // }
```

# Outline

templateHtml {{ expr }}  
\*ngFor, @for() { }  
\*ngIf, @if() { }  
@Component  
[] .. @Input field, input()  
() .. @Output changed, output()  
[()] .. Bidirectional Binding, model()  
router



Next steps ?

Next steps...