|  |
| --- |
| abc |

# **Links**

|  |  |
| --- | --- |
| Document | <https://angular.io/docs> |
| Online IDE for 1st App | <https://stackblitz.com/angular/odpeknvxnlq?file=src%2Fapp%2Fapp.component.ts> |
| Search API | <https://angular.io/api?type=pipe> |
| Template syntax | [https://angular.io/guide/template-syntax](https://angular.io/guide/template-syntax#the-pipe-operator-) |
| Directive | Attribute Directives: <https://angular.io/guide/attribute-directives>  <input [(ngModel)]="hero.name">  <a [title]="product.name + ' details'"> |
| Structural Directives: <https://angular.io/guide/structural-directives>  \*ngFor  \*ngIf  \*ngSwitch: <https://angular.io/guide/template-syntax#ngSwitch>  \*ngStyle: <https://angular.io/guide/template-syntax#ngStyle>  \*ngClass: <https://angular.io/guide/template-syntax#ngClass> |
| Service / Dependency Injection  (share data) | <https://angular.io/guide/dependency-injection> |
| Glossary (bảng thuật ngữ) | <https://angular.io/guide/glossary> |
| CLI – Command Line Interface | <https://angular.io/cli> |
| Lifecycle hooks | <https://angular.io/guide/lifecycle-hooks> |
| Observe API response  Subcribe data | <https://angular.io/guide/http>  <https://angular.io/tutorial/toh-pt4#observable-data>  <https://rxjs-dev.firebaseapp.com/guide/overview> |
| {{ interpolation binding }} | <https://angular.io/guide/template-syntax#interpolation> |
| Dynamic component loader | <https://angular.io/guide/dynamic-component-loader#dynamic-component-loader> |

# **CLI commands**

<https://angular.io/cli>

|  |  |
| --- | --- |
| **npm install -g @angular/cli** | install the Angular CLI globally |
| **ng new** my-project-name | Create a new Angular CLI workspace project  & initial application |
| cd my-project-name  **ng build –prod** | build project |
| **ng serve –open**  ng serve –o | Run the application  <http://localhost:4200/> |
|  |  |
| **ng generate component** heroes  **ng generate component** heroes **–t**  **ng generate component** heroes **--inline-style** | generate a new ”heroes” **component**  generate component without html file – `inline template` (inlineTemplate=true)  generate component without css file – inline style |
| **ng generate service** hero | generate a new ”hero” **service** |
| **ng generate module** app-routing **--flat --module=app** | generate a new ”app-routing” **module**  **--flat** puts the file in src/app instead of its own folder.  **--module=app** tells the CLI to register it in the imports array of the AppModule. |

# **Data binding**

* Overall

<https://angular.io/guide/template-syntax#binding-syntax-an-overview>

|  |  |  |
| --- | --- | --- |
| C:\Users\phitr\OneDrive\Desktop\data binding.PNG | | |
| **Source 🡪 View** | **Source 🡨🡪 View** | **View 🡪 Source** |
| Interpolation  Property  Attribute  Class  Style | Two-way  [NgModules](https://angular.io/api/forms/DefaultValueAccessor)  [[(ngModel)]](https://angular.io/guide/template-syntax#ngmodel-two-way-binding)  [DemoForm](https://stackblitz.com/angular/bbjgpkqgpog?file=src%2Fapp%2Fapp.component.html)  [DemoFontSize](https://stackblitz.com/angular/xleymovoeld?file=src%2Fapp%2Fsizer%2Fsizer.component.html) | Event |
| [target]="expression"  bind-target="expression"  {{expression}} | [(target)]="expression"  bindon-target="expression" | (target)="statement"  on-target="statement" |
| HTML attribute vs. DOM property:  The distinction between an HTML attribute and a DOM property is key to understanding how Angular binding works. **Attributes are defined by HTML. Properties are accessed from DOM (Document Object Model) nodes.**   * A few HTML attributes have 1:1 mapping to properties; for example, id. * Some HTML attributes don't have corresponding properties; for example, aria-\*. * Some DOM properties don't have corresponding attributes; for example, textContent.   Template binding works with **properties** and **events**, not **attributes**.  **Property values can change; attribute values can't:** <https://stackblitz.com/angular/gxqmxolavag?file=src%2Fapp%2Fapp.component.ts>  HTML attributes is to initialize element and directive state | | |

* Types

|  |  |  |
| --- | --- | --- |
| [**Binding types & targets**](https://angular.io/guide/template-syntax#binding-types-and-targets) | | |
| **Type** | **Target** | **Examples** |
| Property | Element property  Component property  Directive property | <img [src]="heroImageUrl">  <app-hero-detail [hero]="currentHero"></app-hero-detail>  <div [[ngClass](https://angular.io/api/common/NgClass)]="{'special': isSpecial}"></div>  <https://angular.io/guide/template-syntax#property-binding-property> |
| Event | Element event  Component event  Directive event | <button (click)="onSave()">Save</button>  <app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail>  <div (myClick)="clicked=$event" clickable>click me</div>  <https://angular.io/guide/template-syntax#template-statements>  <button (click)="deleteHero()">Delete hero</button>  <button (click)="onSave($event)">Save</button>  <button \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes" (click)="deleteHero(hero)">{{hero.name}}</button>  <form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form> |
| Two-way | Event and property | <input [([ngModel](https://angular.io/api/forms/NgModel))]="name"> |
| Attribute | Attribute (the exception) | <https://angular.io/guide/template-syntax#attribute-binding>  <button [attr.aria-label]="help">help</button>  <tr><td [attr.colspan]="1 + 1">Three-Four</td></tr>  <tr><td [colSpan]="1 + 1">Three-Four</td></tr> |
| Class | class property | <https://angular.io/guide/template-syntax#class-binding>  Object format with class names as the keys & truthy/falsy expressions as the values  <div [class.special]="isSpecial">Special</div>  <div [class.selected]=" hero === selectedHero">selected Hero</div>  <div [class]="className className">Special</div>  <div [class]="['className', 'className']">Special</div> // array  <div [class]="{className: true, className: false}">Special</div> // object  <https://angular.io/guide/template-syntax#ngclass>  <div [[ngClass](https://angular.io/api/common/NgClass)]="isSpecial ? 'special' : ''">This div is special</div>  (NgClass no longer provides significant value, and might eventually be removed in the future)  <div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special.</div>    currentClasses: {};    setCurrentClasses() {      // CSS classes: added/removed per current state of component properties      this.currentClasses =  {        'saveable': this.canSave,        'modified': !this.isUnchanged,        'special':  this.isSpecial      };    } |
| Style | style property | <https://angular.io/guide/template-syntax#style-binding>  <https://angular.io/api/animations/style>  <button [style.color]="isSpecial ? 'red' : 'green'">  <button [style]="width: 100px; height: 100px;">  <button [style]=" ['width', '100px']"> // array  <button [style]="{width: '100px', height: '100px'}"> // object  <https://angular.io/guide/template-syntax#ngstyle>  <div [[ngStyle](https://angular.io/api/common/NgStyle)]="currentStyles">  (NgStyle no longer provides significant value, and might eventually be removed in the future)  <div [ngStyle]="currentStyles">    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'      };    } |

* Priority

<https://angular.io/guide/template-syntax#styling-precedence>

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Styling Precedence** | | |
|  | **Property binding ⇨** | **Map binding ⇨** | **Static value** |
| **Template**  **⇩** | <div [class.foo]="hasFoo">  <div [style.color]="color"> | <div [class]="classExpr">  <div [style]="styleExpr"> | <div class="foo">  <div style="color: blue"> |
| **Directive**  **⇩** | host: {'[class.foo]': 'hasFoo'}  host: {'[style.color]': 'color'} | host: {'[class]': 'classExpr'}  {'[style]': 'styleExpr'} | host: {'class': 'foo'}  host: {'style': 'color: blue'} |
| **Component** | host: {'[class.foo]': 'hasFoo'}  host: {'[style.color]': 'color'} | host: {'[class]': 'classExpr'}  host: {'[style]': 'styleExpr'} | host: {'class': 'foo'}  host: {'style': 'color: blue'} |
| <!-- The `class.special` binding will override any value for the `special` class in `classExpr`. -->  <div [class.special]="isSpecial" [class]="classExpr">Some text.</div>  <!-- The `style.color` binding will override any value for the `color` property in `styleExpr`. -->  <div [style.color]="color" [[style](https://angular.io/api/animations/style)]="styleExpr">Some text.</div> | | | |
| <!-- The `class.special` template binding will override any host binding to the `special` class set by `dirWithClassBinding` or `comp-with-host-binding`.-->  <comp-with-host-binding [class.special]="isSpecial" dirWithClassBinding>Some text.</comp-with-host-binding>  <!-- The `style.color` template binding will override any host binding to the `color` property set by `dirWithStyleBinding` or `comp-with-host-binding`. -->  <comp-with-host-binding [style.color]="color" dirWithStyleBinding>Some text.</comp-with-host-binding> | | | |
| <!-- If `classExpr` has a value for the `special` class, this value will override the `class="special"` below -->  <div class="special" [class]="classExpr">Some text.</div>  <!-- If `styleExpr` has a value for the `color` property, this value will override the `[style](https://angular.io/api/animations/style)="color: blue"` below -->  <div [style](https://angular.io/api/animations/style)="color: blue" [[style](https://angular.io/api/animations/style)]="styleExpr">Some text.</div> | | | |
| **null** 🡪 value will be removed entirely  **undefined** 🡪 lower priority value | | | |

* interpolation VS property binding

|  |  |  |
| --- | --- | --- |
| **interpolation** |  | **property binding** (for setting an element property to a non-string data value) |
| <p><img src="{{itemImageUrl}}"> is the <i>interpolated</i> image.</p> |  | <p><img [src]="itemImageUrl"> is the <i>property bound</i> image.</p> |
| <p><span>"{{interpolationTitle}}" is the <i>interpolated</i> title.</span></p> |  | <p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property bound</i> title.</p> |

* two-way binding

|  |
| --- |
| app.module.ts  import { [FormsModule](https://angular.io/api/forms/FormsModule) } from '@angular/forms';  imports: [  [BrowserModule](https://angular.io/api/platform-browser/BrowserModule),  [FormsModule](https://angular.io/api/forms/FormsModule)  ],  .html  <input [(ngModel)]="hero.name"> |

|  |
| --- |
| **Two-way binding syntax w same result** |
|  |
| <label for="example-ngModel">[([ngModel](https://angular.io/api/forms/NgModel))]:</label>  <input [([ngModel](https://angular.io/api/forms/NgModel))]="currentItem.name" id="example-ngModel"> |
| <label for="example-change">(ngModelChange)="...name=$event":</label>  <input [[ngModel](https://angular.io/api/forms/NgModel)]="currentItem.name" (ngModelChange)="currentItem.name=$event" id="example-change"> |
| <label for="without">without [NgModel](https://angular.io/api/forms/NgModel):</label>  <input [value]="currentItem.name" (input)="currentItem.name=$event.target.value" id="without"> |

# **Syntax**

* interpolation

<https://angular.io/api/core/Component#interpolation>

<https://angular.io/guide/template-syntax#interpolation->

|  |
| --- |
| <p>Current customer: {{ currentCustomer }}</p> |
| <img src="{{itemImageUrl}}"> |
| <p>The sum of 1 + 1 is not {{1 + 1 + functionName()}}</p> |
| <input #customerInput>  Inputed value: {{customerInput.value}} |
| {{item?.name}}  // ? protect a view render failure if item is null  // works with long property paths such as a?.b?.c?.d  {{item.color!.toUpperCase()}}  // Assert color is defined, even if according to the `Item` type it could be undefined  // NOT null / undefined  {{$any(this).bestByDate}}  {{$any(item).bestByDate}}  // bestByDate is not a member of the item object |

* ngIf

<https://angular.io/guide/template-syntax#ngIf>

<https://angular.io/api/common/NgIf>

~~null 🡪 err~~

|  |
| --- |
| <p \*ngIf="product.description">  {{ product.description }}  </p> |

* ngFor

<https://angular.io/guide/template-syntax#ngfor>

<https://angular.io/api/common/NgForOf>

|  |  |
| --- | --- |
| ngFor | <li \*ngFor="let product of products">  {{ product.name }}  </li> |
| ngFor  &  index | <div \*ngFor="let product of products; index as productId">  <a [title]="product.name + ' details'" [routerLink]="['/products', productId]">  {{ product.name }}  </a>  </div>  <div \*ngFor="let item of items; let i=index">{{i + 1}} - {{item.name}}</div> |
| ngfor  &  trackby | src/app/app.component.ts  trackByItems(index: number, item: Item): number {  return item.id;  }  src/app/app.component.html  <div \*ngFor="let item of items; trackBy: trackByItems">  ({{item.id}}) {{item.name}}  </div>  <https://angular.io/guide/template-syntax#ngfor-with-trackby>  If the id has already been rendered,  Angular keeps track of it & doesn't re-query the server for the same id |

* ngSwitch

<https://angular.io/guide/template-syntax#the-ngswitch-directives>

<https://angular.io/api/common/NgSwitch>

<https://angular.io/api/common/NgSwitchCase>

<https://angular.io/api/common/NgSwitchDefault>

|  |
| --- |
| <div [ngSwitch]="currentItem.feature">  <app-stout-item \*ngSwitchCase="'stout'" [item]="currentItem"></app-stout-item>  <app-device-item \*ngSwitchCase="'slim'" [item]="currentItem"></app-device-item>  <app-lost-item \*ngSwitchCase="'vintage'" [item]="currentItem"></app-lost-item>  <app-best-item \*ngSwitchCase="'bright'" [item]="currentItem"></app-best-item>  <!-- . . . -->  <app-unknown-item \*ngSwitchDefault [item]="currentItem"></app-unknown-item>  </div> |

* #var (reference variable)

<https://angular.io/guide/template-syntax#template-reference-variables-var>

|  |
| --- |
| <input #customerInput>  inputed value: {{customerInput.value}} |
| <form #itemForm="[ngForm](https://angular.io/api/forms/NgForm)" (ngSubmit)="onSubmit(itemForm)">  <label for="name" >  Name  <input class="form-control" name="name" [ngModel](https://angular.io/api/forms/NgModel) required />  </label>  <button type="submit">Submit</button>  </form>  <div [hidden]="!itemForm.form.valid">  <p>{{ submitMessage }}</p>  </div> |

A template reference variable (#phone) is not the same as a template input variable (let phone) such as in an \*ngFor.

The scope of a reference variable is the entire template.

So, don't define the same variable name more than once in the same template as the runtime value will be unpredictable.

|  |
| --- |
| [Alternative syntax](https://angular.io/guide/template-syntax#alternative-syntax): using ref-fax instead of #fax |
| <input ref-fax placeholder="fax number" />  <button (click)="callFax(fax.value)">Fax</button> |

* onChange

|  |
| --- |
| <input #searchBox id="search-box" (input)="search(searchBox.value)" /> |
| Every time the user types in the textbox, the binding calls search() with the textbox value |

* disabled

Notice that the binding is to the disabled property of the button's DOM element, not the attribute.

This applies to data-binding in general.

Data-binding works with properties of DOM elements, components, and directives, not HTML attributes.

|  |
| --- |
| <input [disabled]="true">  <input [attr.disabled]="false">  <input [disabled]="condition ? true : false">  <input [disabled]="condition ? 'disabled' : null"> |

* click

Event binding ( ): <https://angular.io/guide/template-syntax#event-binding>

|  |
| --- |
| < (click)="share()">  Share  </button> |

* pipe

The pipe operator has a higher precedence than the ternary operator (?:),

which means a ? b : c | x is parsed as a ? b : (c | x).

Nevertheless, for a number of reasons, the pipe operator cannot be used without parentheses in the first and second operands of ?:.

A good practice is to use parentheses in the third operand too.

<https://angular.io/guide/template-syntax#the-pipe-operator->

<https://angular.io/api?type=pipe>

<https://angular.io/guide/pipes>

* [JsonPipe](https://angular.io/api/common/JsonPipe)
* [UpperCasePipe](https://angular.io/api/common/UpperCasePipe)
* [CurrencyPipe](https://angular.io/api/common/CurrencyPipe)
* [DatePipe](https://angular.io/api/common/DatePipe)

|  |  |
| --- | --- |
| <b>{{ product.price | currency }}</b>  <!-- convert title to uppercase, then to lowercase -->  <p>multiple pipes: {{title | [uppercase](https://angular.io/api/common/UpperCasePipe) | [lowercase](https://angular.io/api/common/LowerCasePipe)}}</p> | uses the currency pipe to transform product.price from a number to a currency string.  A pipe is a way you can transform data in your HTML template |
| <!-- pipe with configuration argument => "February 25, 1980" -->  <p>Date format pipe: {{item.manufactureDate | [date](https://angular.io/api/common/DatePipe):'longDate'}}</p>  <p>The hero's birthday is {{ birthday | [date](https://angular.io/api/common/DatePipe):"MM/dd/yy" }} </p> | apply parameters to a pipe |
| <p>Item json pipe: {{item | [json](https://angular.io/api/common/JsonPipe)}}</p> | json pipe is particularly helpful for debugging bindings |
| <div \*ngFor="let shipping of shippingCosts | async"> | returns the latest value from a stream of data & continues to do so for the life of a given component.  When Angular destroys that component, the async pipe automatically stops  <https://angular.io/api/common/AsyncPipe> |

* XXX

XXX

|  |
| --- |
| XXX |

# **Import**

Input & Output demo: <https://stackblitz.com/angular/xleymovoeld?file=src%2Fapp%2Fsizer%2Fsizer.component.ts>

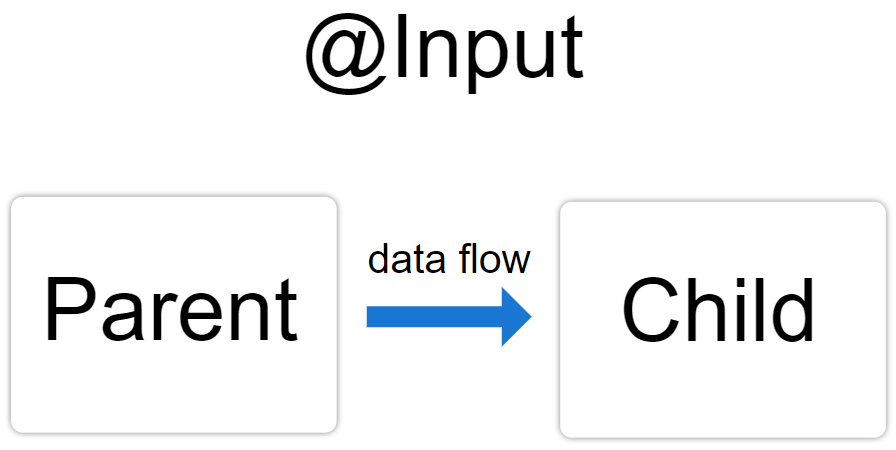
* Input (data fr parent)

[Getting Started / Your First App # Input](https://angular.io/start#input)

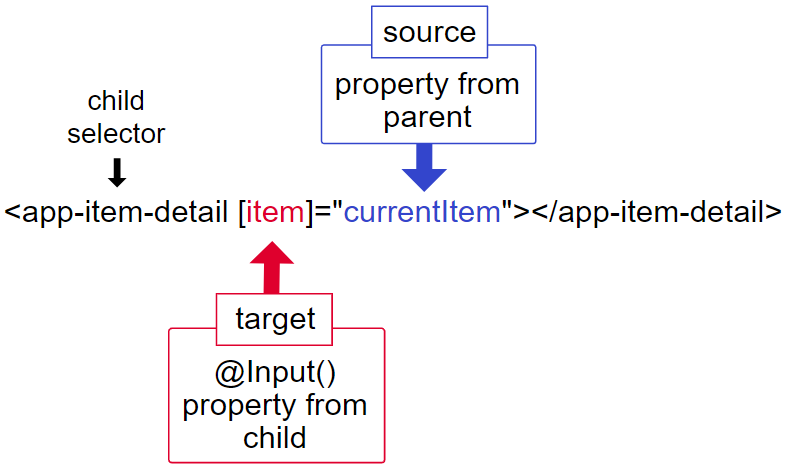
<https://angular.io/guide/template-syntax#how-to-use-input>

<https://angular.io/guide/template-syntax#input-and-output-properties>

The @[Input](https://angular.io/api/core/Input)() decorator indicates that the property value passes in from the component's parent



|  |  |
| --- | --- |
| **Parent** | **Child** |
| src/app/app.component.ts  export class AppComponent {  currentItem = 'Television';  }  src/app/app.component.html  <app-item-detail [item]="currentItem"></app-item-detail> | src/app/item-detail/item-detail.component.ts  import { [Component](https://angular.io/api/core/Component), [Input](https://angular.io/api/core/Input) } from '@angular/core'; // First, import [Input](https://angular.io/api/core/Input)  export class ItemDetailComponent {  @[Input](https://angular.io/api/core/Input)() item: string; // decorate the property with @[Input](https://angular.io/api/core/Input)()  }  src/app/item-detail/item-detail.component.html  Today's item: {{item}} |
| [OnChanges](https://angular.io/api/core/OnChanges) to watch for changes on an @[Input](https://angular.io/api/core/Input)() property  <https://angular.io/api/core/OnChanges>  <https://angular.io/guide/lifecycle-hooks#onchanges> | |



**XXX ↓**

|  |  |
| --- | --- |
| **Parent** | **Child** |
| src/app/product-list/product-list.component.html  <app-product-alerts [product]="product"> </app-product-alerts> | src/app/product-alerts/product-alerts.component.ts  import { [Input](https://angular.io/api/core/Input) } from '@angular/core';  @[Input](https://angular.io/api/core/Input)() product;  src/app/product-alerts/product-alerts.component.html  <p \*[ngIf](https://angular.io/api/common/NgIf)="product.price > 700"> <button>Notify</button> </p> |

|  |  |
| --- | --- |
| **Parent** | **Child** |
| [Pass data from parent to child with input binding](https://angular.io/guide/component-interaction#pass-data-from-parent-to-child-with-input-binding) | |
| 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';  } | 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;  } |
| [Prevent input property changes with a **setter**](https://angular.io/guide/component-interaction#intercept-input-property-changes-with-a-setter) | |
| 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 'Dr IQ', '<no name set>', 'Bombasto'  names = ['Dr IQ', ' ', ' Bombasto '];  } | 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; }  } |
| [Prevent input property changes with **ngOnChange()**](https://angular.io/guide/component-interaction#intercept-input-property-changes-with-ngonchanges) | |
| 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;  newMinor() {  this.minor++;  }  newMajor() {  this.major++;  this.minor = 0;  }  } | 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(', '));  }  } |

* Output (emit event to parent)

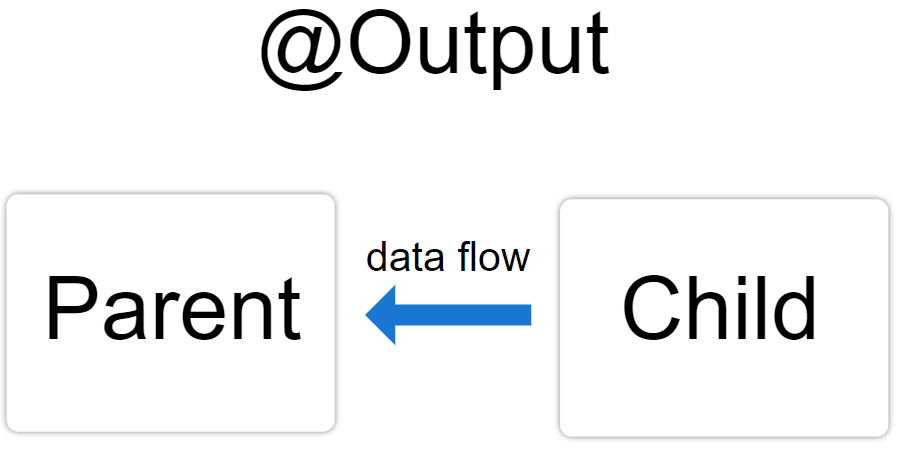
[Getting Started / Your First App # Output](https://angular.io/start#output)

<https://angular.io/guide/template-syntax#how-to-use-output>

<https://angular.io/guide/template-syntax#input-and-output-properties>

<https://angular.io/api/core/EventEmitter>

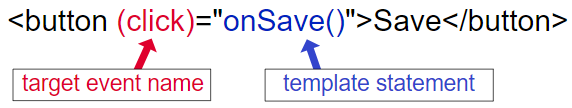
Output(emit) an event



|  |  |
| --- | --- |
| **Child** | **Parent** |
| src/app/item-output/item-output.component.ts  import { [Output](https://angular.io/api/core/Output), [EventEmitter](https://angular.io/api/core/EventEmitter) } from '@angular/core';  export class ItemOutputComponent {  @[Output](https://angular.io/api/core/Output)() newItemEvent = new [EventEmitter](https://angular.io/api/core/EventEmitter)<string>();  // new event emitter and that the data it emits is of type string  addNewItem(value: string) {  this.newItemEvent.emit(value);  }  }  src/app/item-output/item-output.component.html  <label>Add an item: <input #newItem></label>  <button (click)="addNewItem(newItem.value)">Add to parent's list</button> | src/app/app.component.ts  export class AppComponent {  items = ['item1', 'item2', 'item3', 'item4'];  addItem(newItem: string) {  this.items.push(newItem);  }  }  src/app/app.component.html  <app-item-output (newItemEvent)="addItem($event)"></app-item-output>  // $event contains the data that the user types into the <input> in the child template UI  <ul>  <li \*[ngFor](https://angular.io/api/common/NgForOf)="let item of items">  {{item}}  </li>  </ul> |

**XXX ↓**

|  |
| --- |
| Src/app/product-alerts/product-alerts.component.ts  import { [Output](https://angular.io/api/core/Output), [EventEmitter](https://angular.io/api/core/EventEmitter) } from '@angular/core';  @[Output](https://angular.io/api/core/Output)() notify = new [EventEmitter](https://angular.io/api/core/EventEmitter)();  src/app/product-alerts/product-alerts.component.html  <button (click)="notify.emit()">Notify Me</button>  src/app/product-list/product-list.component.ts  onNotify() {  window.alert('You will be notified when the product goes on sale');  }  src/app/product-list/product-list.component.html  <app-product-alerts [product]="product" (notify)="onNotify()"> </app-product-alerts> |



<https://angular.io/guide/template-syntax#event-binding-event>

Demo: <https://stackblitz.com/angular/gjvajkplmqoe?file=src%2Fapp%2Fapp.component.html>

Event types: <https://developer.mozilla.org/en-US/docs/Web/Events>

<https://angular.io/api/core/EventEmitter>

|  |  |
| --- | --- |
| <button (myClick)="clickMessage=$event" clickable>click with myClick</button> {{clickMessage}} | {{clickMessage}} = click |
| <input  [value]="currentItem.name"  (input)="currentItem.name=$event.target.value" >  without [NgModel](https://angular.io/api/forms/NgModel) | init value  onChange event |
| (window as any)['\_\_zone\_symbol\_\_PASSIVE\_EVENTS'] = ['scroll']; |  |

<https://angular.io/guide/user-input>

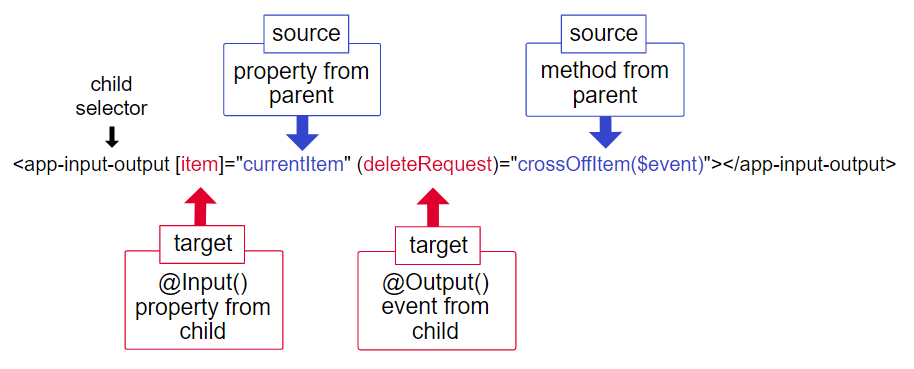
Demo: <https://stackblitz.com/angular/plnybelxrqx?file=src%2Fapp%2Flittle-tour.component.ts>

|  |  |
| --- | --- |
| [keyup](https://angular.io/guide/user-input#get-user-input-from-a-template-reference-variable) | @Component({    selector: 'app-key-up2',    template: `      <input #box (keyup)="onKey(box.value)">      <p>{{values}}</p>    `  })  export class KeyUpComponent\_v2 {    values = '';    onKey(value: string) {      this.values += value + ' | ';    }  } |
| [keyup.enter](https://angular.io/guide/user-input#key-event-filtering-with-keyenter) | @Component({    selector: 'app-key-up3',    template: `      <input #box (keyup.enter)="onEnter(box.value)">      <p>{{value}}</p>    `  })  export class KeyUpComponent\_v3 {    value = '';    onEnter(value: string) { this.value = value; }  } |
| [On blur](https://angular.io/guide/user-input#on-blur)  Click btn  Or  Enter  Or  Click outside | @Component({    selector: 'app-little-tour',    template: `      <input #newHero        (keyup.enter)="addHero(newHero.value)"        (blur)="addHero(newHero.value); newHero.value='' ">      <button (click)="addHero(newHero.value)">Add</button>      <ul><li \*ngFor="let hero of heroes">{{hero}}</li></ul>    `  })  export class LittleTourComponent {    heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];    addHero(newHero: string) {      if (newHero) {        this.heroes.push(newHero);      }    }  } |
| [Scroll](https://angular.io/guide/user-input#source-code) | 1. Create a file zone-flags.ts under src directory.  (window as any)['\_\_zone\_symbol\_\_PASSIVE\_EVENTS'] = ['scroll'];  2. In the src/polyfills.ts file, before importing zone.js, import the newly created zone-flags  import './zone-flags';  import 'zone.js/dist/zone'; // Included with Angular CLI. |

|  |  |
| --- | --- |
| **Child** | **Parent** |
| [Parent listens for child event](https://angular.io/guide/component-interaction#parent-listens-for-child-event) | |
| 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;  }  } | 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($event)">  </app-voter>  `  })  export class VoteTakerComponent {  agreed = 0;  disagreed = 0;  voters = ['Narco', 'Celeritas', 'Bombasto'];  onVoted(agreed: boolean) {  agreed ? this.agreed++ : this.disagreed++;  }  } |
| [Parent interacts with child via local variable](https://angular.io/guide/component-interaction#parent-interacts-with-child-via-local-variable) | |
| component-interaction/src/app/countdown-timer.component.ts  import { Component, OnDestroy, OnInit } from '@angular/core';  @Component({  selector: 'app-countdown-timer',  template: '<p>{{message}}</p>'  })  export class CountdownTimerComponent implements OnInit, OnDestroy {  intervalId = 0;  message = '';  seconds = 11;  clearTimer() { clearInterval(this.intervalId); }  ngOnInit() { this.start(); }  ngOnDestroy() { this.clearTimer(); }  start() { this.countDown(); }  stop() {  this.clearTimer();  this.message = `Holding at T-${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);  }  } | 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 { }    #timer: ability to access any of its ***properties*** or ***methods*** from within the parent template |
| [Parent calls an @ViewChild()](https://angular.io/guide/component-interaction#parent-calls-an-viewchild) | |
| Same as above | 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(); }  } |
| [Parent and children communicate via a service](https://angular.io/guide/component-interaction#parent-and-children-communicate-via-a-service) (data sharing) | |
| 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);  }  } | |
| 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$.subscribe(  mission => {  this.mission = mission;  this.announced = true;  this.confirmed = false;  });  }  confirm() {  this.confirmed = true;  this.missionService.confirmMission(this.astronaut);  }  ngOnDestroy() {  // prevent memory leak when component destroyed  this.subscription.unsubscribe();  }  } | component-interaction/src/app/missioncontrol.component.ts  import { Component } from '@angular/core';  import { MissionService } from './mission.service';  @Component({  selector: 'app-mission-control',  template: `  <h2>Mission Control</h2>  <button (click)="announce()">Announce mission</button>  <app-astronaut \*ngFor="let astronaut of astronauts"  [astronaut]="astronaut">  </app-astronaut>  <h3>History</h3>  <ul>  <li \*ngFor="let event of history">{{event}}</li>  </ul>  `,  providers: [MissionService]  })  export class MissionControlComponent {  astronauts = ['Lovell', 'Swigert', 'Haise'];  history: string[] = [];  missions = ['Fly to the moon!',  'Fly to mars!',  'Fly to Vegas!'];  nextMission = 0;  constructor(private missionService: MissionService) {  missionService.missionConfirmed$.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; }  }  } |

* input & output on the same child component

Demo: <https://stackblitz.com/angular/qxknmaybrny?file=src%2Fapp%2Fapp.component.ts>



**Declarations:**

<https://angular.io/guide/template-syntax#input-and-output-declarations>

<https://angular.io/guide/styleguide#decorate-input-and-output-properties>

**Aliasing in the metadata:** public name of an input/output property should be different from the internal name

<https://angular.io/guide/template-syntax#aliasing-inputs-and-outputs>

|  |
| --- |
| // tslint:disable: no-inputs-metadata-property no-outputs-metadata-property  inputs: ['clearanceItem'],  outputs: ['buyEvent']  // tslint:enable: no-inputs-metadata-property no-outputs-metadata-property  @[Input](https://angular.io/api/core/Input)() item: string;  @[Output](https://angular.io/api/core/Output)() deleteRequest = new [EventEmitter](https://angular.io/api/core/EventEmitter)<string>(); |
| // tslint:disable: no-inputs-metadata-property no-outputs-metadata-property  inputs: ['input1: saveForLaterItem'], // propertyName:alias  outputs: ['outputEvent1: saveForLaterEvent']  // tslint:disable: no-inputs-metadata-property no-outputs-metadata-property  @[Input](https://angular.io/api/core/Input)('wishListItem') input2: string; // @[Input](https://angular.io/api/core/Input)(alias)  @[Output](https://angular.io/api/core/Output)('wishEvent') outputEvent2 = new [EventEmitter](https://angular.io/api/core/EventEmitter)<string>();  // @[Output](https://angular.io/api/core/Output)(alias) propertyName = ... |

* Routing (URL params)

[Getting Started / Routing](https://angular.io/start/start-routing)

<https://angular.io/api/router/RouterModule>

<https://angular.io/api/router/RouterOutlet>

<https://angular.io/api/router/RouterLink>

URL parameters

|  |
| --- |
| src/app/app.module.ts  @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  RouterModule.forRoot([  { path: '', component: ProductListComponent },  { path: 'products/:productId', component: ProductDetailsComponent },  ])  ],  src/app/product-list/product-list.component.html  <a [routerLink]="['/products', productId]">  {{ product.name }}  </a>  src/app/product-details/product-details.component.ts  import { [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) } from '@angular/router';  constructor( private route: [ActivatedRoute](https://angular.io/api/router/ActivatedRoute), ) { }  src/app/product-details/product-details.component.ts  ngOnInit() {  this.route.paramMap.subscribe(params => {  this.product = products[+params.get('productId')];  });  } |

* Services (share data)

[Getting Started / Managing Data #Services](https://angular.io/start/start-data#services)

<https://angular.io/guide/providers>

To share data across components

When you provide the service at the root level, Angular creates a single, shared instance of HeroService and injects into any class that asks for it. Registering the provider in the @Injectable metadata also allows Angular to optimize an app by removing the service if it turns out not to be used after all.

|  |
| --- |
| src/app/cart.service.ts  import { [Injectable](https://angular.io/api/core/Injectable) } from '@angular/core';  @Injectable({  providedIn: 'root'  })  export class CartService {  items = [];  addToCart(product) {  this.items.push(product);  }  getItems() {  return this.items;  }  }  src/app/product-details/product-details.component.ts import { CartService } from '../cart.service';  export class ProductDetailsComponent implements OnInit {  constructor(  private cartService: CartService  ) { }  addToCart(product) {  this.cartService.addToCart(product);  window.alert('Your product has been added to the cart!');  }  }  src/app/product-details/product-details.component.html  <button (click)="addToCart(product)">Buy</button>  src/app/cart/cart.component.ts  import { CartService } from '../cart.service';  export class CartComponent implements OnInit {  items;  constructor(  private cartService: CartService  ) { }  ngOnInit() {  this.items = this.cartService.getItems();  }  }  src/app/cart/cart.component.html  <div class="cart-item" \*ngFor="let item of items">  <span>{{ item.name }}</span>  <span>{{ item.price | currency }}</span>  </div> |

* HttpClient (API JSON)

[Getting Started / Managing Data #HttpClient](https://angular.io/start/start-data#services)

<https://angular.io/guide/http>

<https://angular.io/api/common/http/HttpClient>

Your app can fetch data and interact with external APIs and resources like JSON

|  |
| --- |
| src/assets/shipping.json  [  {  "type": "Overnight",  "price": 25.99  },  {  "type": "2-Day",  "price": 9.99  },  {  "type": "Postal",  "price": 2.99  }  ]  src/app/app.module.ts  import { [HttpClientModule](https://angular.io/api/common/http/HttpClientModule) } from '@angular/common/[http](https://angular.io/api/common/http)';  (at the top of the file with the other imports)  @NgModule({  imports: [  BrowserModule,  HttpClientModule,  ReactiveFormsModule,  RouterModule.forRoot([  { path: '', component: ProductListComponent },  { path: 'products/:productId', component: ProductDetailsComponent },  { path: 'cart', component: CartComponent },  ])  ]  })  src/app/cart.service.ts  import { [HttpClient](https://angular.io/api/common/http/HttpClient) } from '@angular/common/[http](https://angular.io/api/common/http)';  constructor( private [http](https://angular.io/api/common/http): [HttpClient](https://angular.io/api/common/http/HttpClient) ) {}  getShippingPrices() {  return this.http.get('/assets/shipping.json');  }  src/app/shipping/shipping.component.ts  import { CartService } from '../cart.service';  export class ShippingComponent implements OnInit {  shippingCosts;  constructor(  private cartService: CartService  ) {  }  ngOnInit() {  this.shippingCosts = this.cartService.getShippingPrices();  }  }  src/app/shipping/shipping.component.html  <div class="shipping-item" \*ngFor="let shipping of shippingCosts | async">  <span>{{ shipping.type }}</span>  <span>{{ shipping.price | currency }}</span>  </div> |

* Form

[Getting Started / Forms](https://angular.io/start/start-forms#define-the-checkout-form-model)

Create custom form controls and easy validation experiences

<https://angular.io/guide/forms>

<https://angular.io/api/forms/NgForm>

<https://angular.io/api/forms/FormsModule>

<https://angular.io/api/forms/FormBuilder>

<https://angular.io/api/forms/ReactiveFormsModule>

[(ngModel)]: <https://angular.io/guide/template-syntax#ngmodel-two-way-binding>

Demo: <https://stackblitz.com/angular/bbjgpkqgpog?file=src%2Fapp%2Fapp.component.html>

|  |
| --- |
| src/app/cart/cart.component.ts  import { [FormBuilder](https://angular.io/api/forms/FormBuilder) } from '@angular/forms';  export class CartComponent implements OnInit {  items;  checkoutForm;  constructor(  private cartService: CartService,  private formBuilder: FormBuilder,  ) {  this.checkoutForm = this.formBuilder.group({  name: '',  address: ''  });  }  onSubmit(customerData) {  // Process checkout data here  this.items = this.cartService.clearCart();  this.checkoutForm.reset();  console.warn('Your order has been submitted', customerData);  }  }  src/app/cart.service.ts  clearCart() {  this.items = [];  return this.items;  }  src/app/cart/cart.component.html  <form [formGroup]="checkoutForm" (ngSubmit)="onSubmit(checkoutForm.value)">  <div>  <label for="name">  Name  </label>  <input id="name" type="text" formControlName="name">  </div>  <div>  <label for="address">  Address  </label>  <input id="address" type="text" formControlName="address">  </div>  <button class="button" type="submit">Purchase</button>  </form> |

|  |
| --- |
| <form #itemForm="[ngForm](https://angular.io/api/forms/NgForm)" (ngSubmit)="onSubmit(itemForm)">  <label for="name" >  Name  <input class="form-control" name="name" [ngModel](https://angular.io/api/forms/NgModel) required />  </label>  <button type="submit">Submit</button>  </form>  <div [hidden]="!itemForm.form.valid">  <p>{{ submitMessage }}</p>  </div> |

* Dynamic component loader

<https://angular.io/guide/dynamic-component-loader#dynamic-component-loader>

Demo: <https://run.stackblitz.com/api/angular/v1?file=src/app/app.component.ts>

A way to load a new component without a fixed reference to the component in the ad banner's template.

|  |
| --- |
| **anchor directive:** where to insert components |
| ad.directive.ts  import { Directive, ViewContainerRef } from '@angular/core';  @Directive({    selector: '[ad-host]', // to apply the directive to the element  })  export class AdDirective { // to mark valid insertion points in the template.    constructor(public viewContainerRef: ViewContainerRef) { }  // to gain access to the view container of the element that will host the dynamically added component  } |
| ad-banner.component.ts  import { Component, Input, OnInit, ViewChild, ComponentFactoryResolver, OnDestroy } from '@angular/core';  import { AdDirective } from './ad.directive';  import { AdItem }      from './ad-item';  import { AdComponent } from './ad.component';  @Component({    selector: 'app-ad-banner',    template: `                <div class="ad-banner-example">                  <h3>Advertisements</h3>                  <ng-template ad-host></ng-template>                </div>              `  })  /\*  <ng-template> is where you apply the directive you just made.      To apply the AdDirective, recall the selector from ad.directive.ts, ad-host.      Apply that to <ng-template> without the square brackets.      Now Angular knows where to dynamically load components.      It is a good choice for dynamic components because it doesn't render any additional output. \*/  export class AdBannerComponent implements OnInit, OnDestroy {    @Input() ads: AdItem[];    currentAdIndex = -1;    @ViewChild(AdDirective, {static: true}) adHost: AdDirective;    interval: any;    constructor(private componentFactoryResolver: ComponentFactoryResolver) { }    ngOnInit() {      this.loadComponent();      this.getAds();    }    ngOnDestroy() {      clearInterval(this.interval);    }    loadComponent() {      // picks an ad      this.currentAdIndex = (this.currentAdIndex + 1) % this.ads.length;      const adItem = this.ads[this.currentAdIndex];      /\*  use ComponentFactoryResolver to resolve a ComponentFactory for each specific component          ComponentFactory creates an instance of each component \*/      const componentFactory = this.componentFactoryResolver.resolveComponentFactory(adItem.component);      /\*  you're targeting the viewContainerRef that exists on this specific instance of the component.          How do you know it's this specific instance? Because it's referring to adHost and adHost is the directive you set up earlier to tell Angular where to insert dynamic components.          As you may recall, AdDirective injects ViewContainerRef into its constructor. This is how the directive accesses the element that you want to use to host the dynamic component \*/      const viewContainerRef = this.adHost.viewContainerRef;      viewContainerRef.clear();      /\*  createComponent() to add the component to the template          It returns a reference to the loaded component.          Use that reference to interact with the component by assigning to its properties or calling its methods \*/      const componentRef = viewContainerRef.createComponent(componentFactory);      (<AdComponent>componentRef.instance).data = adItem.data;    }    getAds() {      this.interval = setInterval(() => {        this.loadComponent();      }, 2000);    }  } |
| app.module.ts  import { BrowserModule }        from '@angular/platform-browser';  import { NgModule }             from '@angular/core';  import { AppComponent }         from './app.component';  import { HeroJobAdComponent }   from './hero-job-ad.component';  import { AdBannerComponent }    from './ad-banner.component';  import { HeroProfileComponent } from './hero-profile.component';  import { AdDirective }          from './ad.directive';  import { AdService }            from './ad.service';  @NgModule({    imports: [ BrowserModule ],    providers: [AdService],    declarations: [ AppComponent,                    AdBannerComponent,                    HeroJobAdComponent,                    HeroProfileComponent,                    AdDirective ],    entryComponents: [ HeroJobAdComponent, HeroProfileComponent ],    /\*    add dynamically loaded components to entryComponents array (To ensure that the compiler still generates a factory).    Angular compiler generates a ComponentFactory for any component referenced in a template.    However, there are no selector references in the templates for dynamically loaded components since they load at runtime.    \*/    bootstrap: [ AppComponent ]  })  export class AppModule {    constructor() {}  } |
| ad.service.ts  import { Injectable }           from '@angular/core';  import { HeroJobAdComponent }   from './hero-job-ad.component';  import { HeroProfileComponent } from './hero-profile.component';  import { AdItem }               from './ad-item';  @Injectable()  export class AdService {    getAds() {      return [        new AdItem(HeroProfileComponent, {name: 'Bombasto', bio: 'Brave as they come'}),        new AdItem(HeroProfileComponent, {name: 'Dr IQ', bio: 'Smart as they come'}),        new AdItem(HeroJobAdComponent,   {headline: 'Hiring for several positions',                                          body: 'Submit your resume today!'}),        new AdItem(HeroJobAdComponent,   {headline: 'Openings in all departments',                                          body: 'Apply today'}),      ];    }  } |
| **AdComponent interface:** In the ad banner, all components implement a common AdComponent interface to standardize the API for passing data to the components. |
| hero-job-ad.component.ts  import { Component, Input } from '@angular/core';  import { AdComponent }      from './ad.component';  @Component({    template: `      <div class="job-ad">        <h4>{{data.headline}}</h4>        {{data.body}}      </div>    `  })  export class HeroJobAdComponent implements AdComponent {    @Input() data: any;  }  hero-profile.component.ts  import { Component, Input }  from '@angular/core';  import { AdComponent }       from './ad.component';  @Component({    template: `      <div class="hero-profile">        <h3>Featured Hero Profile</h3>        <h4>{{data.name}}</h4>        <p>{{data.bio}}</p>        <strong>Hire this hero today!</strong>      </div>    `  })  export class HeroProfileComponent implements AdComponent {    @Input() data: any;  }  ad.component.ts  export interface AdComponent {    data: any;  } |
|  |

* Angular elements - Popup Service

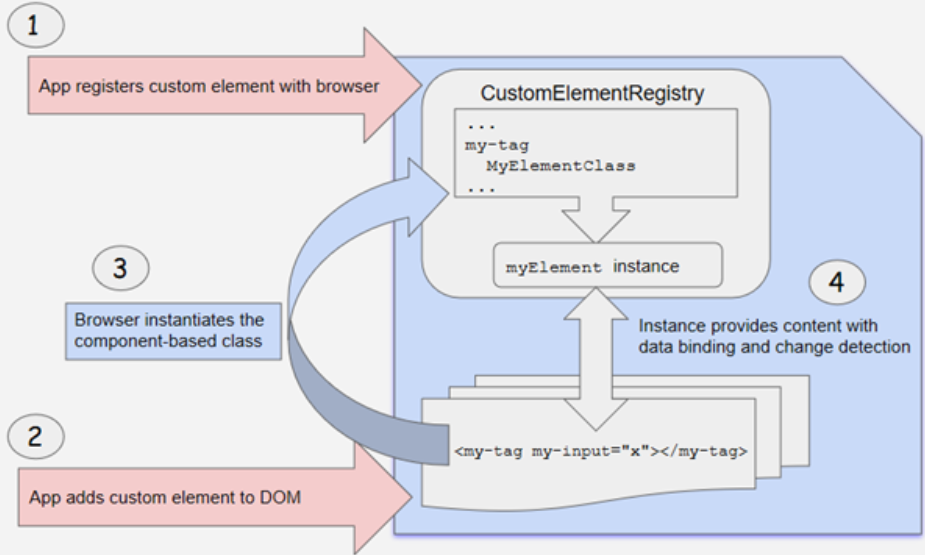
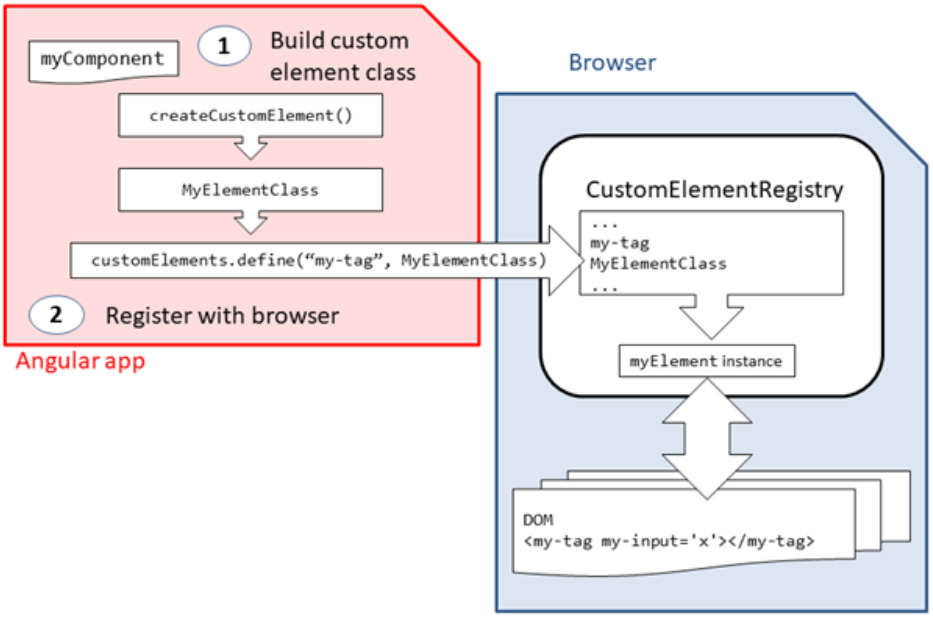
Getting Started / XXX

[https://angular.io/guide/elements#angular-elements-overview](https://angular.io/guide/elements" \l "angular-elements-overview)

[https://angular.io/guide/elements#example-a-popup-service](https://angular.io/guide/elements" \l "example-a-popup-service)

Popup Demo: <https://angular.io/generated/live-examples/elements/stackblitz>

* **createCustomElement()** 
  + convert an Angular component, together with its dependencies, to a custom element. The function collects the component's observable properties, along with the Angular functionality the browser needs to create and destroy instances, and to detect and respond to changes.
  + convert a component into a class that can be registered with the browser as a custom element. After you register your configured class with the browser's custom-element registry, you can use the new element just like a built-in HTML element in content that you add directly into the DOM: <my-popup message="Use Angular!"></my-popup>
* **customElements.define()** register the configured constructor and its associated custom-element tag with the browser's CustomElementRegistry. When the browser encounters the tag for the registered element, it uses the constructor to create a custom-element instance.

|  |
| --- |
| popup.component.ts  defines a simple pop-up element that displays an input message, with some animation and styling  import { Component, EventEmitter, Input, Output } from '@angular/core';  import { animate, state, style, transition, trigger } from '@angular/animations';  @Component({    selector: 'my-popup',    template: `      <span>Popup: {{message}}</span>      <button (click)="closed.next()">&#x2716;</button>    `,    host: {      '[@state]': 'state',    },    animations: [      trigger('state', [        state('opened', style({transform: 'translateY(0%)'})),        state('void, closed', style({transform: 'translateY(100%)', opacity: 0})),        transition('\* => \*', animate('100ms ease-in')),      ])    ],    styles: [`      :host {        position: absolute;        bottom: 0;        left: 0;        right: 0;        background: #009cff;        height: 48px;        padding: 16px;        display: flex;        justify-content: space-between;        align-items: center;        border-top: 1px solid black;        font-size: 24px;      }      button {        border-radius: 50%;      }    `]  })  export class PopupComponent {    state: 'opened' | 'closed' = 'closed';    @Input()    set message(message: string) {      this.\_message = message;      this.state = 'opened';    }    get message(): string { return this.\_message; }    \_message: string;    @Output()    closed = new EventEmitter();  } |
| popup.service.ts  creates an injectable service that provides two different ways to invoke the PopupComponent; as a dynamic component, or as a custom element.  Notice how much more setup is required for the dynamic-loading method  import { ApplicationRef, ComponentFactoryResolver, Injectable, Injector } from '@angular/core';  import { NgElement, WithProperties } from '@angular/elements';  import { PopupComponent } from './popup.component';  @Injectable()  export class PopupService {    constructor(private injector: Injector,                private applicationRef: ApplicationRef,                private componentFactoryResolver: ComponentFactoryResolver) {}    // Previous dynamic-loading method required you to set up infrastructure    // before adding the popup to the DOM.    showAsComponent(message: string) {      // Create element      const popup = document.createElement('popup-component');      // Create the component and wire it up with the element      const factory = this.componentFactoryResolver.resolveComponentFactory(PopupComponent);      const popupComponentRef = factory.create(this.injector, [], popup);      // Attach to the view so that the change detector knows to run      this.applicationRef.attachView(popupComponentRef.hostView);      // Listen to the close event      popupComponentRef.instance.closed.subscribe(() => {        document.body.removeChild(popup);        this.applicationRef.detachView(popupComponentRef.hostView);      });      // Set the message      popupComponentRef.instance.message = message;      // Add to the DOM      document.body.appendChild(popup);    }    // This uses the new custom-element method to add the popup to the DOM.    showAsElement(message: string) {      // Create element      const popupEl: NgElement & WithProperties<PopupComponent> = document.createElement('popup-element') as any;      // Listen to the close event      popupEl.addEventListener('closed', () => document.body.removeChild(popupEl));      // Set the message      popupEl.message = message;      // Add to the DOM      document.body.appendChild(popupEl);    }  } |
| app.module.ts  adds the PopupComponent in the module's entryComponents list, to exclude it from compilation and avoid startup warnings or errors  import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';  import { AppComponent } from './app.component';  import { PopupComponent } from './popup.component';  import { PopupService } from './popup.service';  // Include the `PopupService` provider,  // but exclude `PopupComponent` from compilation,  // because it will be added dynamically.  @NgModule({    imports: [BrowserModule, BrowserAnimationsModule],    providers: [PopupService],    declarations: [AppComponent, PopupComponent],    bootstrap: [AppComponent],    entryComponents: [PopupComponent],  })  export class AppModule {  } |
| app.component.ts  defines the app's root component, which uses the PopupService to add the pop-up to the DOM at run time.  When the app runs, the root component's constructor converts PopupComponent to a custom element  import { Component, Injector } from '@angular/core';  import { createCustomElement } from '@angular/elements';  import { PopupService } from './popup.service';  import { PopupComponent } from './popup.component';  @Component({    selector: 'app-root',    template: `      <input #input value="Message">      <button (click)="popup.showAsComponent(input.value)">Show as component</button>      <button (click)="popup.showAsElement(input.value)">Show as element</button>    `,  })  export class AppComponent {    constructor(injector: Injector, public popup: PopupService) {      // Convert `PopupComponent` to a custom element.      const PopupElement = createCustomElement(PopupComponent, {injector});      // Register the custom element with the browser.      customElements.define('popup-element', PopupElement);    }  } |
|  |

* XXX

Getting Started / XXX

UseForXXX

|  |
| --- |
| XXX |

# **Lifecycle hooks**

<https://angular.io/guide/lifecycle-hooks#lifecycle-sequence>

|  |  |  |
| --- | --- | --- |
| Hook | Purpose | Timing |
| Peek-a-boo (all hooks) | <https://stackblitz.com/angular/rkqqovvgdqp?file=src%2Fapp%2Fpeek-a-boo.component.ts> |  |
| ngOnChanges() | **Detects changes** to **input properties** of the component (or directive)  <https://angular.io/guide/lifecycle-hooks#onchanges>  <https://angular.io/api/core/SimpleChange> | Called before ngOnInit() and whenever one or more data-bound input properties change. |
| ngOnInit() | To perform complex initializations shortly **after construction**  To **set up the component** after Angular sets the input properties  To **fetch its initial data** (Don't fetch data in a component constructor)  <https://angular.io/guide/lifecycle-hooks#oninit> | Called once, after the first ngOnChanges(). |
| ngDoCheck() | Detect and act upon changes **same as ngOnChanges() but** Angular can't or won't detect on its own.  <https://angular.io/guide/lifecycle-hooks#docheck> | Called during every change detection run, immediately after ngOnChanges() and ngOnInit(). |
| ngAfterContentInit() | Calls after Angular projects **external content** into the component.  <https://angular.io/guide/lifecycle-hooks#aftercontent> | Called once after the first ngDoCheck(). |
| ngAfterContentChecked() | Respond after Angular **checks the content projected** into the directive/component.  <https://angular.io/guide/lifecycle-hooks#aftercontent> | Called after the ngAfterContentInit() and every subsequent ngDoCheck(). |
| ngAfterViewInit() | Calls after it **initial creates** a component's **child views**  <https://angular.io/guide/lifecycle-hooks#afterview> | Called once after the first ngAfterContentChecked(). |
| ngAfterViewChecked() | Calls after it **check creates** a component's **child views**  <https://angular.io/guide/lifecycle-hooks#afterview> | Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked(). |
| ngOnDestroy() | Cleanup just before Angular destroys the directive/component.  Unsubscribe Observables and detach event handlers to avoid memory leaks.  <https://angular.io/guide/lifecycle-hooks#ondestroy>   * the logic that must run before Angular destroys the directive * notify another part of the application that the component is going away. * Unsubscribe from Observables and DOM events. * Stop interval timers. * Unregister all callbacks that this directive registered with global or application services. You risk memory leaks if you neglect to do so. | Called just before Angular destroys the directive/component. |

<https://angular.io/guide/lifecycle-hooks#lifecycle-examples>

Demo: <https://stackblitz.com/angular/rkqqovvgdqp?file=src%2Fapp%2Fapp.component.html>

# **Style**

<https://angular.io/guide/component-styles>

Demo: <https://stackblitz.com/angular/lnpkejmmmoe?file=src%2Fapp%2Fhero-app.component.ts>

The **styles** specified in @[Component](https://angular.io/api/core/Component) metadata apply **only within** the template of that component.

They are **not inherited** by any components **nested** within the template nor by any content **projected** into the component.

* Special selectors

<https://angular.io/guide/component-styles#special-selectors>

* **:host**

style for *current component wrapper*

<https://angular.io/guide/component-styles#host>

* **:host-context**

<https://angular.io/guide/component-styles#host-context>

|  |  |
| --- | --- |
| :host-context(.theme-light) h2 {  background-color: red;  } | style for all <h2> (if some ancestor element has class theme-light) inside **current component wrapper** |

* **::ng-deep**
* **/deep/**
* **>>>**

<https://angular.io/guide/component-styles#deprecated-deep--and-ng-deep>

|  |  |  |
| --- | --- | --- |
| style for all <h3> from current component wrapper to all **child elements** | | |
| :host ::ng-deep h3 {    background: blue;  } | :host /deep/ h3 {    background: blue;  } | :host >>> h3 {    background: blue;  } |

* Style Types

|  |  |
| --- | --- |
| external css file | @Component({    selector: 'app-hero-detail',    templateUrl: './hero-detail.component.html',    styleUrls: ['./hero-detail.component.css']  }) |
| **ng generate component** heroes **--inline-style**  (generate component without css file – inline style)  Must be written in CSS (not less, sass, scss, …) | @Component({    selector: 'app-root',    template: `      <h1>Tour of Heroes</h1>      <app-hero-main [hero]="hero"></app-hero-main>    `,    styles: ['h1 { font-weight: normal; }']  }) |
| Template inline styles | @Component({    selector: 'app-hero-controls',    template: `      <style>        button {          background-color: white;          border: 1px solid #777;        }      </style>      <h3>Controls</h3>      <button (click)="activate()">Activate</button>    `  }) |
| Template link tags | @Component({    selector: 'app-hero-team',    template: `      <!-- We must use a relative URL so that the AOT compiler can find the stylesheet -->      <link rel="stylesheet" href="../assets/hero-team.component.css">      <h3>Team</h3>      <ul>        <li \*ngFor="let member of hero.team">          {{member}}        </li>      </ul>`  }) |
| @import css file | src/app/hero-details.component.css (excerpt)  @import './hero-details-box.css'; |
| **External & global style files** | <https://github.com/angular/angular-cli/wiki/stories-global-styles>  angular.json  "styles": [    "src/styles.css",    "src/more-styles.css",    { "input": "src/lazy-style.scss", "lazy": true },    { "input": "src/pre-rename-style.scss", "bundleName": "renamed-style" },  ], |

# **Definition**

<https://angular.io/guide/glossary>

* Object

objectName {

id: number;

name: string;

}

* Array

arrayName[]

* Attribute

<https://angular.io/guide/glossary#attribute-directives>

defined by HTML, **value cannot change**

<button disabled>Save</button>

* Property

accessed from DOM (Document Object Model) nodes, **value can change**

<button [disabled]="isUnchanged">Save</button>

* Component

<https://angular.io/guide/glossary#component>

XXX

<app-hero-detail></app-hero-detail>

* Directive

<https://angular.io/guide/glossary#directive>

XXX

@Directive()

@Input()

@NgModule

NgClass

NgForm

NgIf, NgFor, NgSwitch

* Module

XXX

* Lifecycle hooks

XXX

* XXX

XXX

* XXX

XXX

* XXX

XXX

* XXX

XXX

* XXX

XXX

* XXX

XXX

* XXX

XXX

# **\*\*\*NOTE**

* model-view-controller (MVC) or model-view-viewmodel (MVVM)
* a difference between a Component & a Directive in that a Component will be referenced without specifying the attribute value, and a Directive will not change the implicit reference (that is, the element)
* <script> & <html>, <body>, <base> are ignore in \*.html file
* templateUrl: can contain \*.html, [\*.svg](https://angular.io/guide/template-syntax#svg-in-templates) or HTML
* Reserve the constructor() for simple initialization such as wiring constructor parameters to properties. The constructor shouldn't *do anything*. It certainly shouldn't call a function that makes HTTP requests to a remote server as a *real* data service would.
  + **Constructor()** for **variables**
  + **ngOnInit**() for **functions**
* Instead, call getHeroes() inside the *[ngOnInit lifecycle hook](https://angular.io/guide/lifecycle-hooks)* and let Angular call ngOnInit() at an appropriate time *after* constructing a HeroesComponent instance.
* Angular only binds to **public** component properties
* Angular uses **capitalization conventions** to distinguish the names of various types, as described in the [naming guidelines section](https://angular.io/guide/styleguide#02-01) of the Style Guide. Here's a summary of the case types:
  + camelCase : Symbols, **properties**, **methods**, **pipe** names, non-component directive selectors, constants. Standard or lower camel case uses lowercase on the first letter of the item. For example, "selectedHero".
  + UpperCamelCase (or PascalCase): **Class** names, including classes that define components, **interfaces**, **NgModules**, **directives**, and **pipes**, Upper camel case uses uppercase on the first letter of the item. For example, "HeroListComponent".
  + dash-case (or "kebab-case"): Descriptive part of **file** names, component **selectors**. For example, "app-hero-list".
  + underscore\_case (or "snake\_case"): Not typically used in Angular. Snake case uses words connected with underscores. For example, "convert\_link\_mode".
  + UPPER\_UNDERSCORE\_CASE (or UPPER\_SNAKE\_CASE, or SCREAMING\_SNAKE\_CASE): Traditional for **constants** (acceptable, but prefer camelCase). Upper snake case uses words in all capital letters connected with underscores. For example, "FIX\_ME".
* Many NgModules such as the [RouterModule](https://angular.io/guide/router" \o "Routing and Navigation) and the [FormsModule](https://angular.io/guide/forms" \o "Forms) define their own attribute directives. The most common attribute directives are as follows:
  + [NgClass](https://angular.io/guide/template-syntax#ngClass)—adds and removes a set of CSS classes.
  + [NgStyle](https://angular.io/guide/template-syntax#ngStyle)—adds and removes a set of HTML styles.
  + [NgModel](https://angular.io/guide/template-syntax#ngModel)—adds two-way data binding to an HTML form element.
* Return a string if the target property expects a string, a number if it expects a number, an object if it expects an object, and so on

|  |  |
| --- | --- |
| string | src/app/app.component.html  <app-item-detail [childItem]="parentItem"></app-item-detail>  src/app/item-detail/item-detail.component.ts  @[Input](https://angular.io/api/core/Input)() childItem: string;  src/app/app.component.ts  parentItem = 'lamp'; |
| array | src/app/app.component.html  <app-item-list [items]="currentItems"></app-item-list>  src/app/item-list.component.ts  @[Input](https://angular.io/api/core/Input)() items: Item[];  src/app/item.ts  export interface Item { id: number; name: string; }  src/app.component.ts  currentItems = [{ id: 21, name: 'phone' }]; |

# **Tour of Heros App**

* Links

<https://angular.io/tutorial>

<https://stackblitz.com/angular/bbykkemeqor?file=src%2Fapp%2Fhero.service.ts>

* Visual Layout



* interface

|  |
| --- |
| export interface Hero {      id: number;      name: string;  } |

* Mock data

<https://angular.io/tutorial/toh-pt2#create-mock-heroes>

|  |
| --- |
| import { Hero } from '../hero';    hero: Hero = {      id: 1,      name: 'Windstorm'    };  export const HEROES: Hero[] = [    { id: 11, name: 'Dr Nice' },    { id: 12, name: 'Narco' },    { id: 13, name: 'Bombasto' },    { id: 14, name: 'Celeritas' },    { id: 15, name: 'Magneta' },    { id: 16, name: 'RubberMan' },    { id: 17, name: 'Dynama' },    { id: 18, name: 'Dr IQ' },    { id: 19, name: 'Magma' },    { id: 20, name: 'Tornado' },  ];  export class HeroesComponent implements OnInit {    selectedHero: Hero;    onSelect(hero: Hero): void {      this.selectedHero = hero;    }  } |
| src/app/hero.ts  export class Hero {      constructor(          public id: number,          public name: string) { }  }  src/app/app.component.ts  heroes = [      new Hero(1, 'Windstorm'),      new Hero(13, 'Bombasto'),      new Hero(15, 'Magneta'),      new Hero(20, 'Tornado')  ];  myHero = this.heroes[0]; |

* Routing config

<https://angular.io/tutorial/toh-pt5#routes>

|  |
| --- |
| const routes: Routes = [    { path: '', redirectTo: '/dashboard', pathMatch: 'full' }, // auto navigate to dashboard & change '/' path to '/dashboard'    { path: 'dashboard', component: DashboardComponent },    { path: 'heroes', component: HeroesComponent }, // tells the router to match that URL to path: 'heroes' and display the HeroesComponent when the URL is something like localhost:4200/heroes    { path: 'detail/:id', component: HeroDetailComponent }, // :id is a placeholder for a specific hero id  ]; |

* Get id From URL

<https://angular.io/tutorial/toh-pt5#routable-herodetailcomponent>

|  |
| --- |
| import { ActivatedRoute } from '@angular/router';    constructor(      private route: ActivatedRoute, // holds parameter information of URL    ) { }      const id = +this.route.snapshot.paramMap.get('id');      // route parameters are always strings      // (+) operator converts the string to a number (which is what a hero id should be)      // route.snapshot is a static image of the route information shortly after the component was created      // paramMap is a dictionary of route parameter values extracted from the URL. The "id" key returns the id of the hero to fetch |

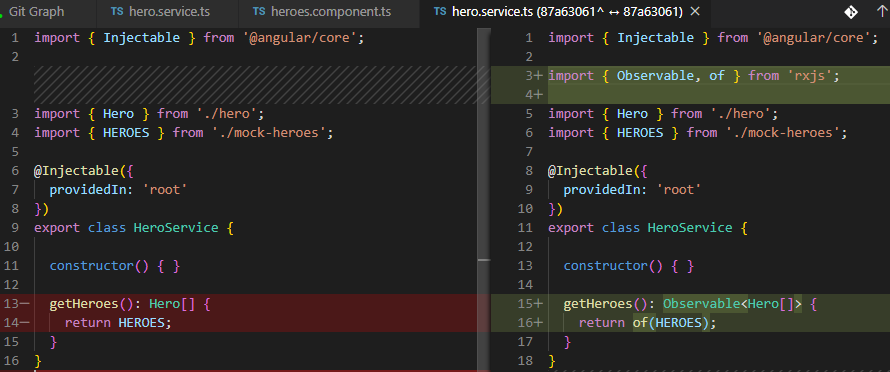
* goBack()

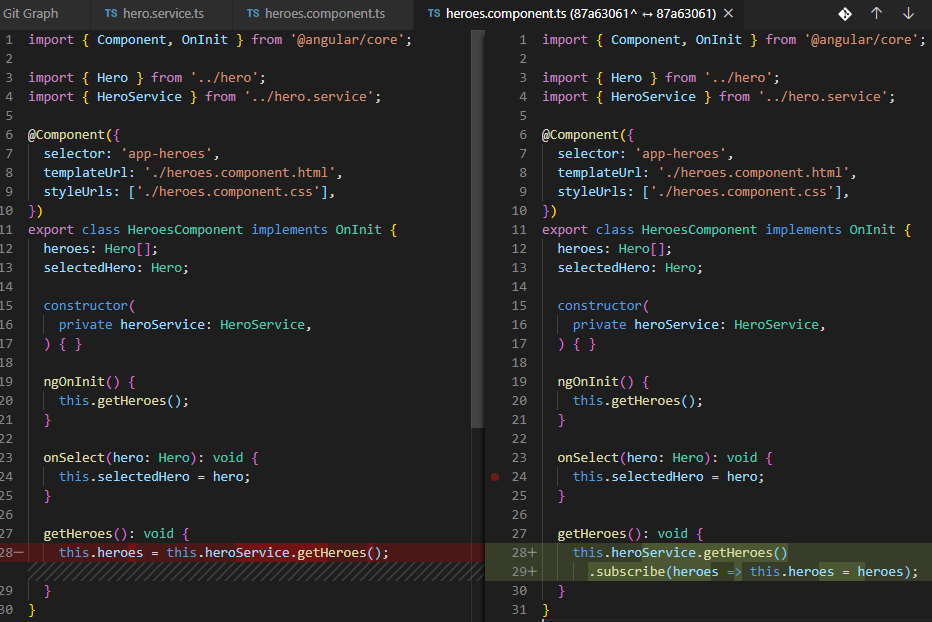
<https://angular.io/tutorial/toh-pt5#find-the-way-back>

|  |
| --- |
| import { Location } from '@angular/common';    constructor(      private location: Location, // service for interacting with the browser; to navigate back to the view that navigated here    ) { }    goBack(): void {      this.location.back();    } |

* Observable data (by RxJS library)

<https://angular.io/tutorial/toh-pt4#observable-data>





* Get data from server (by In-memory Web API module)
  + Get Heroes List

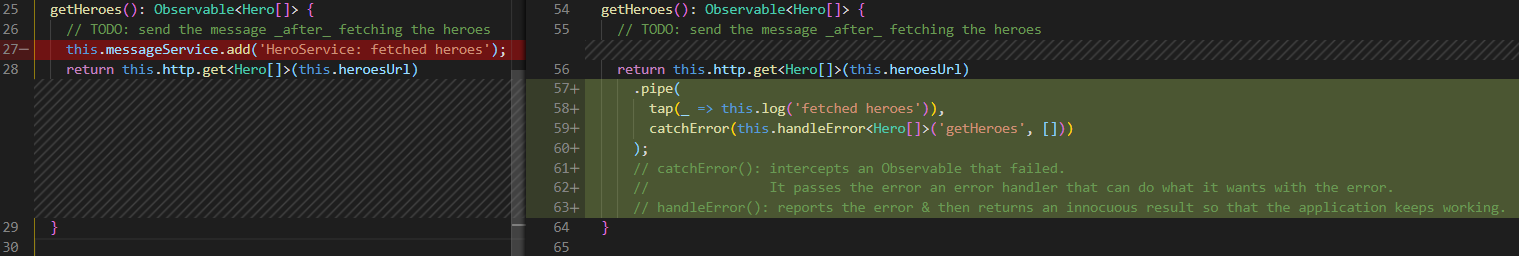
<https://angular.io/tutorial/toh-pt6#heroes-and-http>

* + Log message by RxJS tap()

<https://angular.io/tutorial/toh-pt6#handleerror>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/2b4743a2fbcf505a92e78dbc285640d06193f2f2?diff=split>

|  |
| --- |
| import { catchError, map, tap } from 'rxjs/operators';    // After reporting the error to the console,    // the handler constructs a user friendly message    // and returns a safe value to the app so the app can keep working.    /\*\*     \* Handle Http operation that failed.     \* Let the app continue.     \* @param operation - name of the operation that failed     \* @param result - optional value to return as the observable result     \*/    private handleError<T>(operation = 'operation', result?: T) {      return (error: any): Observable<T> => {        // TODO: send the error to remote logging infrastructure        console.error(error); // log to console instead        // TODO: better job of transforming error for user consumption        this.log(`${operation} failed: ${error.message}`);        // Let the app keep running by returning an empty result.        return of(result as T);      };    }    /\*\* Log a HeroService message with the MessageService \*/    private log(message: string) {      this.messageService.add(`HeroService: ${message}`);    } |



|  |
| --- |
| getHeroes(): Observable<Hero[]> {      // TODO: send the message \_after\_ fetching the heroes      return this.http.get<Hero[]>(this.heroesUrl)        .pipe(          tap(\_ => this.log('fetched heroes')),          catchError(this.handleError<Hero[]>('getHeroes', []))        );        // catchError(): intercepts an Observable that failed. It passes the error an error handler that can do what it wants with the error.        // handleError(): reports the error & then returns an innocuous result so that the application keeps working.    } |

* + Get hero by id

<https://angular.io/tutorial/toh-pt6#get-hero-by-id>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/5cc71262f82d5bdc83cd485e0fda335895ab476e>

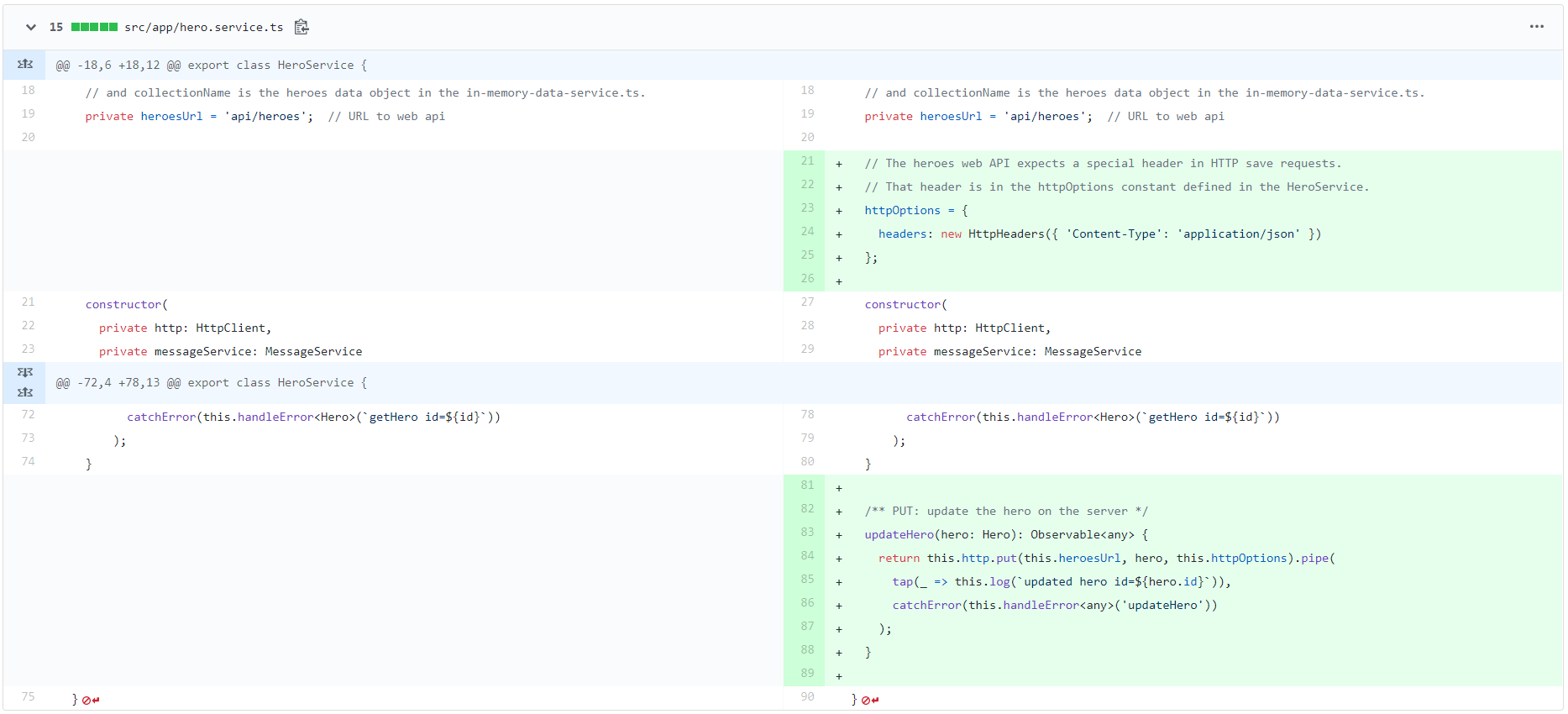
|  |
| --- |
| // GET hero by id. Will 404 if id not found    getHero(id: number): Observable<Hero> {      // TODO: send the message \_after\_ fetching the hero      const url = `${this.heroesUrl}/${id}`;      return this.http.get<Hero>(url)        .pipe(          tap(\_ => this.log(`fetched hero id=${id}`)),          catchError(this.handleError<Hero>(`getHero id=${id}`))        );    } |

* + Update hero by id

<https://angular.io/tutorial/toh-pt6#update-heroes>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/245aef04aeb463fb837e1b084f0975f26e6a4923?diff=split>





* + Add a new hero

<https://angular.io/tutorial/toh-pt6#add-a-new-hero>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/3d10a4dda7078c17f97b5b1fde2269242d7a2cbb?diff=split>

|  |
| --- |
| src/app/heroes/heroes.component.html  <div>    <label>Hero name:      <input #heroName />    </label>    <!-- (click) passes input value to add() & then clears the input -->    <button (click)="add(heroName.value); heroName.value=''">      add    </button>  </div>  src/app/heroes/heroes.component.ts    add(name: string): void {      name = name.trim();      if (!name) { return; }      this.heroService.addHero({ name } as Hero)        .subscribe(hero => {          this.heroes.push(hero);        });    }  src/app/hero.service.ts    /\*\* POST: add a new hero to the server \*/    addHero(hero: Hero): Observable<Hero> {      return this.http.post<Hero>(this.heroesUrl, hero, this.httpOptions).pipe(        tap((newHero: Hero) => this.log(`added hero w/ id=${newHero.id}`)),        catchError(this.handleError<Hero>('addHero'))      );    } |

* + Delete hero by id

<https://angular.io/tutorial/toh-pt6#delete-a-hero>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/7640e720f3e9b06286b694f837d65e1cb1e18cf4?diff=split>

|  |
| --- |
| src/app/heroes/heroes.component.ts    delete(hero: Hero): void {      this.heroes = this.heroes.filter(h => h !== hero);      this.heroService.deleteHero(hero).subscribe();      // There's nothing for the component to do with the Observable returned by heroService.delete() but it must subscribe() anyway.      // If you don't use subscribe(), the service will not send the delete request to the server.      // As a rule, an Observable does nothing until something subscribes.    }  src/app/hero.service.ts    /\*\* DELETE: delete the hero from the server \*/    deleteHero(hero: Hero | number): Observable<Hero> {      const id = typeof hero === 'number' ? hero : hero.id;      const url = `${this.heroesUrl}/${id}`;      return this.http.delete<Hero>(url, this.httpOptions).pipe(        tap(\_ => this.log(`deleted hero id=${id}`)),        catchError(this.handleError<Hero>('deleteHero'))      );    } |

* + Search Hero by name

<https://angular.io/tutorial/toh-pt6#search-by-name>

<https://github.com/TonyTran0209/angular-tour-of-heroes/commit/80e937938ecefc6889cd8054aa86ba4b9ffd25d9?diff=split>

<https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap>

|  |
| --- |
| src/app/hero-search/hero-search.component.html  <div id="search-component">    <h4><label for="search-box">Hero Search</label></h4>    <!-- As the user types in the search box, an input event binding calls the component's search() method with the new search box value. -->    <!-- Every time the user types in the textbox, the binding calls search() with the textbox value, a "search term".    The searchTerms becomes an Observable emitting a steady stream of search terms. -->    <input #searchBox id="search-box" (input)="search(searchBox.value)" />    <ul class="search-result">      <!-- The $ is a convention that indicates heroes$ is an Observable, not an array -->      <!-- async pipe identifies Angular's AsyncPipe & subscribes to an Observable automatically, so you won't have to do so in the component class -->      <li \*ngFor="let hero of heroes$ | async">        <a routerLink="/detail/{{ hero.id }}">          {{ hero.name }}        </a>      </li>    </ul>  </div>  src/app/hero-search/hero-search.component.ts  import { Component, OnInit } from '@angular/core';  import { Observable, Subject } from 'rxjs';  import {     debounceTime, distinctUntilChanged, switchMap   } from 'rxjs/operators';  import { Hero } from '../hero';  import { HeroService } from '../hero.service';  @Component({    selector: 'app-hero-search',    templateUrl: './hero-search.component.html',    styleUrls: ['./hero-search.component.css']  })  export class HeroSearchComponent implements OnInit {    heroes$: Observable<Hero[]>; // The $ is a convention that indicates heroes$ is an Observable, not an array    private searchTerms = new Subject<string>(); // is an RxJS Subject    constructor(private heroService: HeroService) {}    // Push a search term into the observable stream.    /\* A Subject is both a source of observable values & an Observable itself.    You can subscribe to a Subject as you would any Observable.    You can also push values into that Observable by calling its next(value) method as the search() method does. \*/    search(term: string): void {      this.searchTerms.next(term);    }    ngOnInit(): void {      this.heroes$ = this.searchTerms.pipe(        // wait until the flow of new string events pauses for 300ms before passing along the latest string        // You'll never make requests more frequently than 300ms        debounceTime(300),        // ensures that a request is sent only if the filter text changed        distinctUntilChanged(),        /\* switchMap() calls the search service for each search term that makes it through debounce() and distinctUntilChanged().        It cancels & discards previous search observables,        returning only the latest search service observable \*/        // https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap        switchMap((term: string) => this.heroService.searchHeroes(term)),      );    }    }  src/app/hero.service.ts    /\* GET heroes whose name contains search term \*/    searchHeroes(term: string): Observable<Hero[]> {      if (!term.trim()) {        // if not search term, return empty hero array.        return of([]);      }      return this.http.get<Hero[]>(`${this.heroesUrl}/?name=${term}`).pipe(        tap(x => x.length ?          this.log(`found heroes matching "${term}"`) :          this.log(`no heroes matching "${term}"`)),        catchError(this.handleError<Hero[]>('searchHeroes', []))      );    } |

* XXX

|  |
| --- |
| abc |