Angular

Junghoo Cho

cho@cs.ucla.edu

Angular Overview

- Web front-end development framework developed by Goog
  - Supports development of complex single-page app
  - Provides easy-to-use end-to-end development tool-chain
  - Encourages modular development (through *components* and *servic*
- One of three most popular Web front-end development framework/library
  - Together with React.js and Vue.js

What We Will Learn

- Angular command-line interface (CLI)
- Angular directive: custom HTML extension
- Angular component: custom HTML element
    - template, class, CSS
    - Data binding: Template-class interaction
        - Interpolation, property binding, event binding
- Angular service and dependency injection
- We reimplement "Google Suggest" using Angular as exampl

Angular CLI (1)

- Angular Command-Line Interface (CLI)
- `$ ng new <app-name>`
  - Generates initial skeleton code
  - The main code is in `src/app`

```
$ ng new google-suggest --skip-install
$ cd google-suggest
$ ls src/app
app.component.css   app.component.spec.ts  app.module.ts
app.component.html  app.component.ts
```

  - Angular CLI creates the top-level “app component”
    - `app.component.ts`, `app.component.html`, `app.component.css` are componen template and CSS file
    - All other components become children of the app component
  - `app.module.ts` is the “root module” of the app

Angular CLI (2)

- `$ ng build --prod`
  - Build the final production .html, .css, .js files in `dist/`
  - Produced files can be deployed to any static HTTP server
- `$ ng serve --host 0.0.0.0`
  - Monitor source-code change and dynamically recompile and serve internal HTTP server
  - Helps avoiding manual recompilation and deployment
  - Extremely useful during development
- Remember: *Angular code runs in the browser*, not server!
  - `ng serve` is only for development not for deployment
- `$ ng new google-suggest --minimal`
  - html, css, ts merged into a single file

Angular Component

- In Angular, app is split into modular components
  - Each component is developed independently with unit test
- *Component*: specific part of an app responsible for certain UI interaction
  - Label list, search box, email list, …
- Each component consists of
  1. Class object
  2. HTML *template*
  3. CSS style

Generating Components

- Q: How should we split our Google Suggest app into compo
- A: For illustration, we split out app into
  - SearchBoxComponent
  - DisplayComponent
- `$ ng generate component <component-name>`
  - Generate skeleton code for component

```
$ ng generate component search-box
$ ng generate component display
$ ls src/app
app.component.css       app.component.ts  search-box/
app.component.html      app.module.ts
app.component.spec.ts   display/
```

- We need to develop HTML, CSS and TS for each component

Developing HTML Template

- *Template*: “HTML view” of component
  - Determines what a component displays on the page
- Q: What should each component have in their template?

```
<!-- search-box.component.html -->
<form action="http://www.google.com/search">
    <input type="text" name="q"><input type="submit">
</form>
```

```
<!-- display.component.html -->
<div>Suggestion here</div>
```

- Q: Why don’t they show up in my app?

Directive

- Q: How can I include SearchBox and Display components in application?
- A: Add *component directives* to app component template

```
<!-- app.component.html -->
<app-search-box></app-search-box>
<app-display></app-display>
```

  - *Component directive*: custom “HTML tag” of component
- *Directive*: “HTML extension” by Angular
  - Technically incorrect, but good enough for beginners
  - *Component directive*, *attribute directive*, *structural directive*
  - e.g., `app-search-box`, `app-display`, …

Component Directive

- Q: How does Angular know that `<app-search-box>` tag corr
  `SearchBoxComponent`?
- `@Component` decorator:

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

Data Binding

- Component template *does not* interact with its class so far
- Q: How can a component template and its class interact?
  - Q: Can we call a class method when user presses submit button?
- *Data binding*: ways to make a template and its class interact
- We learn three data binding mechanisms
  - *Interpolation*
  - *Property binding*
  - *event binding*

Interpolation

- Q: Can I display `title` property of `AppComponent` class in its t

```
// app.component.ts
title = "google-suggest";
```

- *Interpolation*

```
<!-- app.component.html -->
<h1>{{ title }}</h1>
```

  - Syntax: `{{ expression }}`
    - `expression` is replaced with the result of the expression
    - `expression` should have *no side effect*

Property Binding

- Q: Can I set the (default) value of input query box with data f component class?

```
// search-box.component.ts
defaultQuery = "UCLA";
```

- *Property binding*

```
<!-- search-box.component.html -->
<input type="text" name="q" [value]="defaultQuery">
```

  - Syntax: `[property]="expression"`
    1. Evaluate the result of `expression`
    2. Set the result as the value of DOM `property`
  - Whenever the value of `expression` changes, the `property` value is *c updated*

Event Binding

- Q: Can we call a class method when user presses submit butt
- A: Yes, use *event binding*

```
<!-- search-box.component.html -->
<input type="submit" (click)="showAlert();">
```

  - Syntax: `(event)="statement;"`
    - Execute `statement` when `event` is triggered
    - `statement` may have side effect

```
// search-box.component.ts
showAlert() { alert("No USC Please!"); }
```

Attribute Directives in Component

- So far, property and event bindings are done on standard HT elements

```
<input type="submit" (click)="showAlert();">
```

- Angular components, like `<app-search-box>` can also
  - have its own property,
  - throw events, and
  - support property and event bindings!

Attribute Directives in Component

- Example

```
<!-- app.component.html -->
<app-search-box [query]="title" (input)="handleInput($event);"
</app-search-box>
```

  - Assign `title` value of `AppComponent` to `query` property of `SearchBo`
  - Call `handleInput` method of `AppComponent` when `input` event is thr `SearchBoxComponent`
- Angular components looks and behaves like a standard HTM
  - Property and event bindings are “APIs” of the component
  - Allows parent and child components interact

HTML Element vs Angular Component

- Angular component is like a user-defined HTML DOM eleme

```
<input type="submit" onclick="callback();">
<search-box [query]="'UCLA'" (click)="callback();"></search-bo
```

- Almost one-to-one mapping between the two

HTML Element	**Angular Component**
HTML Tag `<p>`	Component directive `<app-display>`
`attr="val"`	Property binding `[prop]="expr"`
`onevent="f();"`	Event binding `(event)="stmt;"`
DOM object	Component class object
DOM property	Component class property
DOM method	Component class method

Component as User-Defined HTML Ele

- “API” of a Component
  - “Name”: component directive
  - “Input”: property binding
  - “Output”: event binding
- When developing a component, design its “API” first
  - What “input property” should it have?
  - What “output events” should it throw?

More on Property Binding (1)

- Example

```
<!-- app.component.html -->
<app-search-box [defaultQuery]="title"></app-search-box>
```

  - Assign `title` value (of `AppComponent`) to `defaultQuery` property (of `SearchBoxComponent`)
  - `defaultQuery` is an input of `SearchBoxComponent`

- Add `@Input()` decorator to allow property binding

```
// search-box.component.ts
import { Input } from '@angular/core';

@Input() defaultQuery: string;
```

More on Property Binding (2)

- Note the difference

```
<input type="text" name="q" value="query">
```

vs

```
<input type="text" name="q" [value]="query">
```

vs

```
<input type="text" name="q" [value]="'query'">
```

More on Event Binding

- Example

```
<!-- app.component.html -->
<app-search-box (input)="handleInput($event);"></app-search-bo
```

  - Set `handleInput()` of `AppComponent` as the `input` event handler fro `SearchBoxComponent`
  - `$event` is the standard DOM `event` object in this case
- Q: What events can be thrown from a component?
- A1: All standard DOM events within `SearchBoxComponent`, lil "bubble up" through the component
- A2: A component can *throw its own custom event*, not just bu standard DOM events!

Throwing Custom Event

- Example

```
<!-- app.component.html -->
<app-search-box (advice)="handleAdvice($event);"></app-search-
```

  - When `advice` event is thrown, call `handleAdvice($event)` of `AppCor`

Throwing Custom Event

- Component can throw a “custom event” by
  1. Creating an `EventEmitter` object and assign it to a property
     - Add `@Output()` decorator to make it available for event binding
  2. Calling `emit(obj)` on the property
     - Property name becomes event name
     - `obj` is passed as the `$event` object

```
// search-box.component.ts
import { EventEmitter, Output } from '@angular/core';
...
@Output() advice = new EventEmitter<string[]>();
...
this.advice.emit(["Yes UCLA", "No USC"]);
```

Angular $event Object

```
<!-- app.component.html -->
<app-search-box (advice)="handleAdvice($event);"></app-search-box
```

- In event binding, `$event` is *Angular event object*
- `$event` object can be
  - the standard DOM event object

    `e.g., <input type="submit" (click)="showAlert($event);">`
  - “custom event object” emitted by `EventEmitter`

    `e.g., <app-search-box (advice)="handleAdvice($event);"></app-search-bo`
- Remark: Custom events ***do not*** bubble up
  - This is different from standard DOM events
  - Only its direct parent can catch custom events

Structural Directive

- Q: Can we show different HTML elements depending on a cl
  property value?
- Structural directives: `*ngIf`, `*ngFor`, `*ngSwitch`

Structural Directive: `*ngIf`

```
<img [src]="imgUrl" *ngIf="imgUrl">
```

- Syntax: `*ngIf="expression"`
  - Create element (and its descendants) only if `expression` is “true”

Structural Directive: *ngFor

```
<ul>
    <li *ngFor="let item of items">{{ item.name }}</li>
</ul>
```

- Syntax: `*ngFor="let a of list"`
  - Create one DOM element per each element in `list`
  - `a`: *template input variable*
  - If name conflict, template variables has precedence over class prope

Structural Directive: `ngSwitch`

```
<ng-container [ngSwitch]="media.type">
    <img [src]="media.url" *ngSwitchCase="'image'">
    <video [src]="media.url" *ngSwitchCase="'video'"></video>
    <embed [src]="media.url" *ngSwitchDefault>
</ng-container>
```

- Syntax:

  ```
  <e [ngSwitch]="expression">
      <e1 *ngSwitchCase="case_expression1">
      ...
      <en *ngSwitchDefault>
  </e>
  ```

  - Create child element(s) with `expression == case_expression`
  - Create `default` element(s) if no match
  - Our example used `<ng-container>` to group multiple elements

Summary So Far

- Angular provides extensions for HTML and DOM!
- Angular component: “Extended HTML element”
  - Template, class, CSS
  - Data binding
    - Interpolation, property binding, event binding
  - `@Input`, `@Output`, `EventEmitter`
- Angular directive: “Extended HTML keyword”
  - Component directive: HTML tag
  - Attribute directive: HTML attribute
    - Input property binding, Output event binding
  - Structural directive: New keywords for conditional structure

Back to Google Suggest

- Our Google Suggest app consists of `AppComponent` and two
  - `SearchBoxComponent`
  - `DisplayComponent`
- Components are "dumb" UI elements (like HTML elements)
  - They just display what they are asked to display
  - They throw events that they are asked to throw
  - They have "no global picture" of the app

Functions of Two Components

- Q: What are the functions of `SearchBoxComponent` and `DisplayComponent`?
- A:
  - `SearchBoxComponent`
    - Monitors user's input events
    - Monitors submit button clicks
    - Alerts USC query
  - `DisplayComponent`
    - Displays suggestions
- Q: What should be the "API" of the two components
  - Q: What should be the "inputs" and "outputs" of the components?

Component "API"

- `DisplayComponent` takes one input property

  ```
  <app-display [suggestions]="listOfSuggestions">
  ```

  - Display the input in its template
- `SearchBoxComponent` provides two output events

  ```
  <app-search-box (userInput)="handleUserInput($event);"
                  (submit)="handleSubmit($event);">
  ```

  - Monitors and throws the two user events
  - Output `$event` objects:
    - `userInput` event: current "query" in the input box
    - `submit` event: DOM `Event` object, so that parent can "veto" query submission i

DisplayComponent

```
<!-- display.component.html -->
<ul>
    <li *ngFor="let suggestion of suggestions">{{suggestion}}</li
</ul>

// display.component.ts
import { Input } from '@angular/core';

@Input() suggestions: string[];
```

SearchBoxComponent

```
<!-- search-box.component.html -->
<form action="http://www.google.com/search">
    <input type="text" name="q" (input)="handleInput($event);">
    <input type="submit" (click)="handleSubmit($event)">
</form>

// search-box.component.ts
import { EventEmitter, Output } from '@angular/core';

query: string = "";
@Output() userInput = new EventEmitter<string>();
@Output() submit = new EventEmitter<Event>();
```

SearchBoxComponent

```
// search-box.component.ts
handleInput(event) {
    this.query = event.target.value;
    if (this.noUSC(this.query)) {
        this.userInput.emit(this.query);
    } else {
        alert("No USC query please!");
    }
}
handleSubmit(event) {
    if (this.noUSC(this.query)) {
        this.submit.emit(event);
    } else {
        alert("No USC query please!");
        event.preventDefault();
    }
}
noUSC(query) { return !(/(^| )USC($| )/i.test(query)); }
```

Next Step

- All basic UI elements have been implemented as two compc
- Let us “connect” them to implement our application logic!
- Q: What should the app do?
  - In case of `userInput` event, send query to Google suggest server an result in `DisplayComponent`

Handling userInput Event

```
<!-- app.component.html -->
<app-search-box (userInput)="getSuggestions($event)"></app-searcl
<app-display [suggestions]="suggestions"></app-display>
// app.component.ts
suggestions: string[] = [];
getSuggestions(query: string) {
    // for now, we just show user input as suggestions
    this.suggestions = [query];
}
```

Summary: Angular Component

- Components are “extended UI elements”
- An app is hierarchically “decomposed” into simpler compone
  - Components have clearly defined "API"s
  - Input and output binding
- Components are “composed” hierarchically to more complex components
  - Reduces code and development complexity
  - Encourages modular and independent app development
- Any questions?

Angular Service

- For illustration, assume that multiple components in our app interact with Google suggest server
- Q: What should we do? Copy the same code in each compor
- A: Separate out the shared functionality into a separate *servi*
- Angular service provides “services” that can be used by many components

Creating Service

- We separate code for `getSuggestions()` functionality into `SuggestionService`
  - This is for illustration only. This is not necessary for our example sinc `getSuggestions()` is used only by `AppComponent`
- `SuggestionService` API
  - `getSuggestions(query): Promise<string[]>`
  - Let any component obtain suggestions from Google server without details

```
$ ng generate service suggestion
$ ls -l
suggestion.service.spec.ts      suggestion.service.ts
```

SuggestionService: Implementatio

```
// suggestion.service.ts
async getSuggestions(query: string): Promise<string[]> {
    let res = await fetch("http://oak.cs.ucla.edu/classes/cs144/e
    let text = await res.text();
    let parser = new DOMParser();
    let xml = parser.parseFromString(text,"text/xml");
    let s = xml.getElementsByTagName('suggestion');
    let suggestions = [];
    for (let i = 0; i < s.length; i++) {
        suggestions.push(s[i].getAttribute("data"));
    }
    return suggestions; // automatically converted to Promise wi
}
```

Dependency Injection (1)

- Q: If multiple components, say `A` and `B`, need to use `SuggestionService` who should “create” and “own” `SuggestionService`? `A` or `B`?
- A: Service is an independent entity from components
  - Service does not “belong to” a single component
  - No individual component should create and own a service
  - Instead, the framework should create it and make it available to wh

Dependency Injection (2)

- Any component can "request" a service by listing it in constr parameter
  - `constructor(..., serv: NeededService, ...)`
- When app starts, the framework creates the requested servic passes it as constructor parameter
  - *Dependency injection*: Component's "dependency" is automatically " framework
  - *One* service instance is shared by *all* requesting components

Dependency Injection: Example

```
// app.component.ts
import { SuggestionService } from './suggestion.service';
...
public suggestionService: SuggestionService;

constructor(suggestionService: SuggestionService) {
    this.suggestionService = suggestionService;
}
```

```
// app.component.ts
import { SuggestionService } from './suggestion.service';
...
constructor(public suggestionService: SuggestionService) {}
```

- Above two are exactly the same

Using Service

```
// app.component.ts
import { SuggestionService } from './suggestion.service';

constructor(public suggestionService: SuggestionService) {}

async getSuggestions(query) {
    if (query.length > 0) {
        this.suggestions = await this.suggestionService.getSugges
    }
}
```

- Final Code

What We Learned

- Angular provides custom “extensions” to HTML and DOM
  - Component: Custom HTML DOM elements
  - Directive: Custom HTML keywords
- Angular app is developed as a hierarchy of components and
  - Component: Dumb UI elements
  - Service: Service code shared by many components
- Angular encourages *modular* and *reusable* development
  - Clear separation of UI elements from application logic
  - Dependency injection as basic “plumbing” mechanism

Other Topics for Self Study

- Tutorial on building “traditional” web site with Angular
- Angular routing & navigation
- Angular module system
- Angular component life cycle hooks

References

- Angular tutorial: https://angular.io/tutorial
- More extensive book on Angular (free): https://codecraft.tv/courses/angular/
- Official Angular documentation: https://angular.io/guide/arc