Decorators in Angular

In Angular, decorators are a special kind of declaration that can be attached to a class, method, accessor, property, or parameter. Decorators are used to attach metadata to a class or its members, which Angular uses to understand the structure and behavior of the application.

Commonly Used Angular Decorators

1. @Component: Defines a component.
2. @Directive: Defines a directive.
3. @Injectable: Marks a class as available to be provided and injected as a dependency.
4. @Input: Binds a property as an input property.
5. @Output: Binds a property as an output property.
6. @NgModule: Defines an Angular module.

Example: @Component Decorator

Step 1: Create a New Angular Component

You can generate a new component using Angular CLI:

ng generate component my-component

This command will create a new folder named my-component with the following files:
- my-component.component.ts
- my-component.component.html
- my-component.component.css
- my-component.component.spec.ts

Step 2: Understanding the @Component Decorator

Open the my-component.component.ts file. You will see the @Component decorator applied to the MyComponent class.

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

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

```
  // Component logic goes here
}
```

Breakdown of the @Component Decorator:

- selector: The CSS selector that identifies this component in a template. app-my-component means you can use this component in other templates with <app-my-component></app-my-component>.
- templateUrl: The path to the HTML file that defines the view of this component.
- styleUrls: An array of paths to CSS files that define the styles for this component.

Step 3: Implementing the Component

Modify my-component.component.html to display a simple message:

```
<h1>Hello from My Component!</h1>
```

In the my-component.component.ts file, you can add properties and methods:

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

@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html',
  styleUrls: ['./my-component.component.css']
})
export class MyComponent {
  message: string = 'Hello from My Component!';
}
```

Update my-component.component.html to use this property:

```
<h1>{{ message }}</h1>
```

Step 4: Using the Component

To use MyComponent in another component, such as AppComponent, open app.component.html and add the selector:

```
<app-my-component></app-my-component>
```

Now, when you run your Angular application (ng serve), you should see the message "Hello from My Component!" displayed on the page.

Other Decorators

@Input and @Output

These decorators are used for data binding between parent and child components.

Example:

child.component.ts

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

@Component({
 selector: 'app-child',
 template: `
  <h2>{{ childMessage }}</h2>
  <button (click)="sendMessage()">Send Message</button>
 `
})
export class ChildComponent {
 @Input() childMessage: string;
 @Output() messageEvent = new EventEmitter<string>();

 sendMessage() {
  this.messageEvent.emit('Hello from Child Component');
 }
}
```

parent.component.ts

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

@Component({
 selector: 'app-parent',
 template: `
  <app-child [childMessage]="parentMessage"
(messageEvent)="receiveMessage($event)"></app-child>
  <p>{{ message }}</p>
 `
})
export class ParentComponent {
 parentMessage = 'Message from Parent Component';
 message: string;

 receiveMessage($event) {
  this.message = $event;
 }
}
```

In this example, @Input is used to pass data from the parent to the child component, and @Output is used to send data from the child back to the parent.

Summary

Decorators in Angular are powerful tools that enable you to define the behavior and appearance of components and other parts of your application. They help in creating a more readable and maintainable codebase by clearly separating the different aspects of a component's behavior and configuration.