Angular: how to use events in components

Angular is a framework developed by Google that facilitates the creation of dynamic and complex web applications. One of the key features of Angular is the event management system, which allows components to communicate with each other effectively. Component events in Angular offer a powerful way to manage interaction between different parts of your application. In this article, we will explore how to create and use component events in Angular.

Creating Events in Components

To create an event in an Angular component, we need to follow some key steps. Suppose we have a parent component (ParentComponent) and a child component (ChildComponent), and we want to make the child component emit an event when something happens inside it.

Definition of the Event in the Child Component

In the child component, we need to import the `EventEmitter` module from `@angular/core` and create an instance of it inside the component. Next, we will define an event using the `@Output()` decorator. Let's see an example:

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

@Component({
   selector: 'app-child',
```

```
   template: '<button
(click)="onButtonClick()">Click</button>',
})
export class ChildComponent {
   @Output() buttonClicked: EventEmitter<void> = new
EventEmitter<void>();

   onButtonClick(): void {
     this.buttonClicked.emit();
   }
}
```

In the example above, we defined an event called `buttonClicked` which is an instance of `EventEmitter`. The event is fired when the button in the template is clicked.

Event Handling in the Parent Component

Now that we have created the event in the child component, we need to handle it in the parent component. To do this, we should listen to the event in the HTML of the parent component using the `(event)="handler()"` event binding hook. Here's an example:

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

@Component({
   selector: 'app-parent',
   template: '<app-child
(buttonClicked)="handleButtonClick()"></app-child>',
})
export class ParentComponent {
   handleButtonClick(): void {
```

```
    console.log('The button was clicked in the child
component.');
    // You can perform further actions here.
  }
}
```

In the parent component template, we are listening for the child component's `buttonClicked` event and calling the `handleButtonClick()` method when the event occurs.

Use of Events in the Services

Events can also be used to communicate between services and components. We can extend this concept by using a service to manage communication between components.

Creating a Service with Events

```
import { Injectable, EventEmitter } from
'@angular/core';

@Injectable({
  providedIn: 'root',
})
export class EventService {
  buttonClicked: EventEmitter<void> = new
EventEmitter<void>();
}
```

Use of the Service in the Components

```
import { Component } from '@angular/core';
import { EventService } from './event.service';

@Component({
   selector: 'app-parent',
   template: '<button
(click)="onButtonClick()">Click</button>',
})
export class ParentComponent {
   constructor(private eventService: EventService) {}

   onButtonClick(): void {
     this.eventService.buttonClicked.emit();
   }
}

@Component({
   selector: 'app-child',
   template: '<p>Child Component</p>',
})
export class ChildComponent {
   constructor(private eventService: EventService) {
     this.eventService.buttonClicked.subscribe(() =>
{
       console.log('The button was clicked in the
child component.');
       // You can perform further actions here.
     });
   }
}
```

In this example, we have created a service (`EventService`) that emits a `buttonClicked` event. The parent component uses the service to emit the event when the button is clicked, and the child component uses the same service to subscribe to that event and handle it when it occurs.

Conclusions

Component events in Angular provide a powerful mechanism for managing communication between different elements of an application. Creating and using events in components can greatly improve the modularity and maintainability of your code. Experiment with the provided examples and adapt them to the specific needs of your application to maximize the effectiveness of events in your Angular projects.