Sharing data between child and parent directives and components

A common pattern in Angular is sharing data between a parent component and one or more child components. Implement this pattern with the `@Input()` and `@Output()` decorators.

Consider the following hierarchy:

```
content_copy<parent-component>

  <child-component></child-component>

</parent-component>
```

The `<parent-component>` serves as the context for the `<child-component>`.

`@Input()` and `@Output()` give a child component a way to communicate with its parent component. `@Input()` lets a parent component update data in the child component. Conversely, `@Output()` lets the child send data to a parent component.

Sending data to a child component

The `@Input()` decorator in a child component or directive signifies that the property can receive its value from its parent component.

To use @Input(), you must configure the parent and child.

Configuring the child component

To use the @Input() decorator in a child component class, first import Input and then decorate the property with @Input(), as in the following example.

src/app/item-detail/item-detail.component.ts

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

export class ItemDetailComponent {

  @Input() item = ''; // decorate the property with @Input()

}
```

In this case, @Input() decorates the property `item`, which has a type of `string`, however, @Input() properties can have any type, such as `number`, `string`, `boolean`, or `object`. The value for `item` comes from the parent component.

Next, in the child component template, add the following:

src/app/item-detail/item-detail.component.html

```
content_copy<p>

  Today's item: {{item}}

</p>
```

Configuring the parent component

The next step is to bind the property in the parent component's template. In this example, the parent component template is `app.component.html`.

1. Use the child's selector, here `<app-item-detail>`, as a directive within the parent component template.
2. Use property binding to bind the `item` property in the child to the `currentItem` property of the parent.
   src/app/app.component.html

   ```
   content_copy<app-item-detail [item]="currentItem"></app-item-detail>
   ```

3. In the parent component class, designate a value for `currentItem`:
   src/app/app.component.ts

```
content_copyexport class AppComponent {
  currentItem = 'Television';
}
```

With `@Input()`, Angular passes the value for `currentItem` to the child so that `item` renders as `Television`.

The following diagram shows this structure:

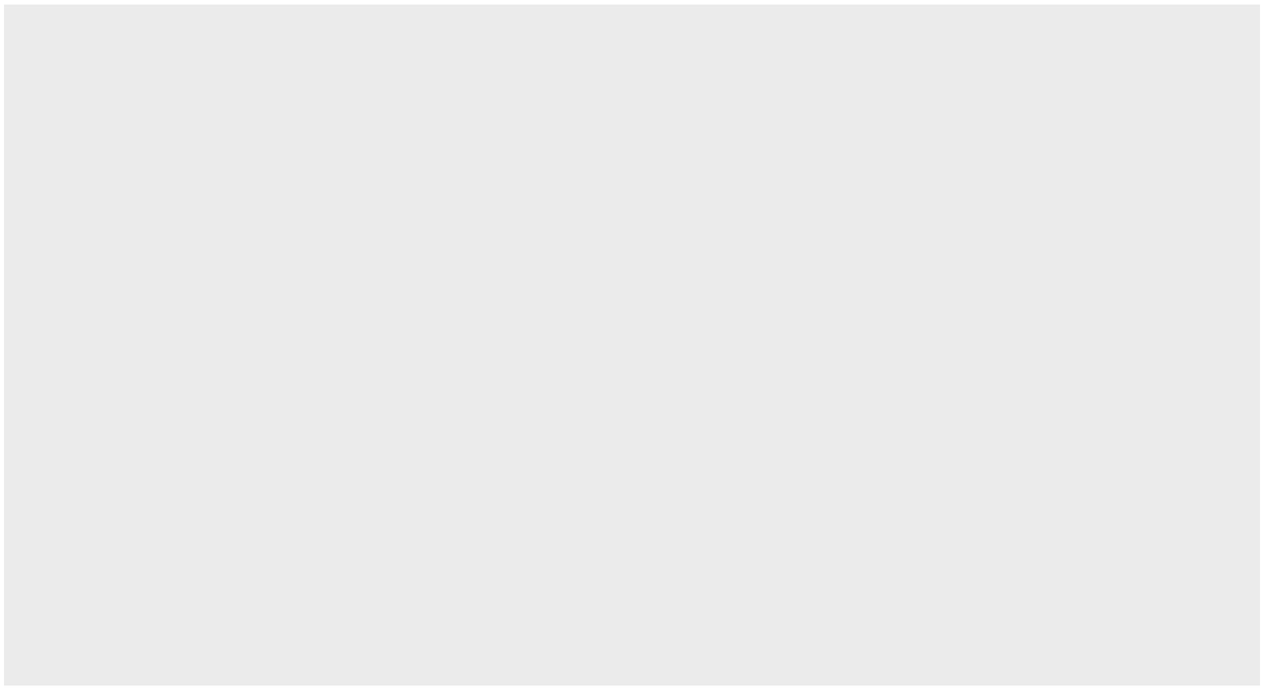
The target in the square brackets, `[]`, is the property you decorate with `@Input()` in the child component. The binding source, the part to the right of the equal sign, is the data that the parent component passes to the nested component.

Watching for `@Input()` changes

To watch for changes on an `@Input()` property, use OnChanges, one of Angular's lifecycle hooks. See the `OnChanges` section of the Lifecycle Hooks guide for more details and examples.

Sending data to a parent component

The `@Output()` decorator in a child component or directive lets data flow from the child to the parent.



@Output() marks a property in a child component as a doorway through which data can travel from the child to the parent.

The child component uses the @Output() property to raise an event to notify the parent of the change. To raise an event, an @Output() must have the type of EventEmitter, which is a class in @angular/core that you use to emit custom events.

The following example shows how to set up an @Output() in a child component that pushes data from an HTML <input> to an array in the parent component.

To use @Output(), you must configure the parent and child.

Configuring the child component

The following example features an <input> where a user can enter a value and click a <button> that raises an event. The EventEmitter then relays the data to the parent component.

1. Import Output and EventEmitter in the child component class:

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

2. In the component class, decorate a property with @Output(). The following example newItemEvent @Output() has a type of EventEmitter, which means it's an event.
   src/app/item-output/item-output.component.ts

```
content_copy@Output() newItemEvent = new EventEmitter<string>();
```

The different parts of the preceding declaration are as follows:

DECLARATION PARTS	DETAILS
`@Output()`	A decorator function marking the property as a way for data to go from the child to the parent.
`newItemEvent`	The name of the `@Output()`.
`EventEmitter<string>`	The `@Output()`'s type.
`new EventEmitter<string>()`	Tells Angular to create a new event emitter and that the data it emits is of type string.

For more information on EventEmitter, see the EventEmitter API documentation.

3. Create an `addNewItem()` method in the same component class:
src/app/item-output/item-output.component.ts

```
content_copyexport class ItemOutputComponent {


  @Output() newItemEvent = new EventEmitter<string>();


  addNewItem(value: string) {

    this.newItemEvent.emit(value);

  }
```

```
    }
```

The addNewItem() function uses the @Output(), newItemEvent, to raise an event with the value the user types into the <input>.

Configuring the child's template

The child's template has two controls. The first is an HTML <input> with a template reference variable, #newItem, where the user types in an item name. The value property of the #newItem variable stores what the user types into the <input>.

src/app/item-output/item-output.component.html

```
content_copy<label for="item-input">Add an item:</label>

<input type="text" id="item-input" #newItem>

<button type="button" (click)="addNewItem(newItem.value)">Add to
parent's list</button>
```

The second element is a <button> with a click event binding.

The (click) event is bound to the addNewItem() method in the child component class. The addNewItem() method takes as its argument the value of the #newItem.value property.

Configuring the parent component

The AppComponent in this example features a list of items in an array and a method for adding more items to the array.

src/app/app.component.ts

```
content_copyexport class AppComponent {

  items = ['item1', 'item2', 'item3', 'item4'];


  addItem(newItem: string) {

    this.items.push(newItem);

  }

}
```

The addItem() method takes an argument in the form of a string and then adds that string to the items array.

Configuring the parent's template

1. In the parent's template, bind the parent's method to the child's event.
2. Put the child selector, here `<app-item-output>`, within the parent component's template, `app.component.html`.
   src/app/app.component.html

   ```
   content_copy<app-item-output
   (newItemEvent)="addItem($event)"></app-item-output>
   ```

   The event binding, `(newItemEvent)='addItem($event)'`, connects the event in the child, `newItemEvent`, to the method in the parent, `addItem()`.
   The `$event` contains the data that the user types into the `<input>` in the child template UI.
   To see the `@Output()` working, add the following to the parent's template:

   ```
   content_copy<ul>

     <li *ngFor="let item of items">{{item}}</li>

   </ul>
   ```

   The `*ngFor` iterates over the items in the `items` array. When you enter a value in the child's `<input>` and click the button, the child emits the event and the parent's `addItem()` method pushes the value to the `items` array and new item renders in the list.

Using `@Input()` and `@Output()` together

Use `@Input()` and `@Output()` on the same child component as follows:

src/app/app.component.html

```
content_copy<app-input-output

  [item]="currentItem"

  (deleteRequest)="crossOffItem($event)">

</app-input-output>
```

The target, `item`, which is an `@Input()` property in the child component class, receives its value from the parent's property, `currentItem`. When you click delete, the child component raises an event, `deleteRequest`, which is the argument for the parent's `crossOffItem()` method.

The following diagram shows the different parts of the `@Input()` and `@Output()` on the `<app-input-output>` child component.

The child selector is `<app-input-output>` with `item` and `deleteRequest` being `@Input()` and `@Output()` properties in the child component class. The property `currentItem` and the method `crossOffItem()` are both in the parent component class.

To combine property and event bindings using the banana-in-a-box syntax, `[()]`, see Two-way Binding.