Sharing Data between Angular Components - Four Methods

Parent to Child: Sharing Data via Input
parent.component.ts
child.component.ts

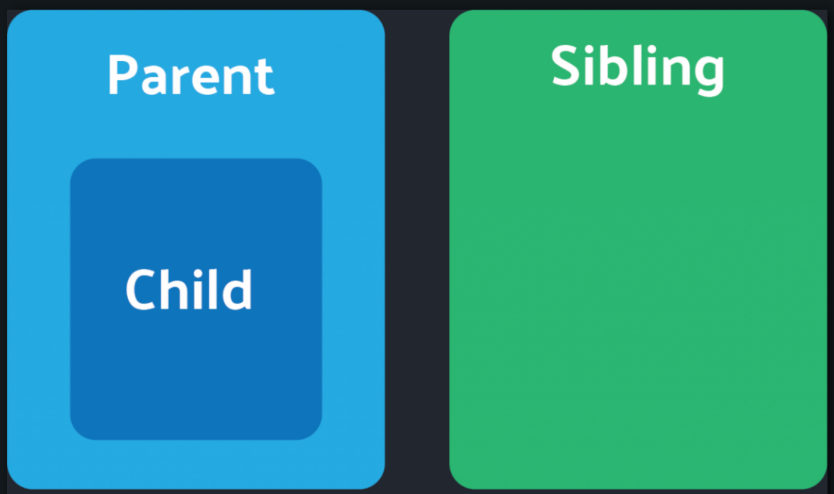
Child to Parent: Sharing Data via ViewChild
parent.component.ts
child.component.ts

Child to Parent: Sharing Data via Output() and EventEmitter
and EventEmitter
parent.component.ts
child.component.ts

Unrelated Components: Sharing Data with a Service
data.service.ts
parent.component.ts

Learning Angular? Check out the full **Angular 9 Course**

Data sharing is an essential concept to understand before diving into your first Angular project. In this lesson, I provide four different methods for sharing data between Angular components.



The Parent-Child-Sibling structure of our Angular app.

Parent to Child: Sharing Data via Input

This is probably the most common and straightforward method of sharing data. It works by using the `@Input()` **decorator** to allow data to be passed via the template.

parent.component.ts

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

@Component({
  selector: 'app-parent',
  template: `
    <app-child [childMessage]="parentMessage"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  parentMessage = "message from parent"
  constructor() { }
}
```

child.component.ts

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

@Component({
  selector: 'app-child',
  template: `
      Say {{ message }}
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  @Input() childMessage: string;

  constructor() { }

}
```

Child to Parent: Sharing Data via ViewChild

ViewChild allows a one component to be injected into another, giving the parent access to its attributes and functions. One caveat, however, is that child won't be available until after the view has been initialized. This means we need to implement the AfterViewInit lifecycle hook to receive the data from the child.

parent.component.ts

```
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { ChildComponent } from "../child/child.component";

@Component({
  selector: 'app-parent',
  template: `
    Message: {{ message }}
    <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
```

```
})
export class ParentComponent implements AfterViewInit {

  @ViewChild(ChildComponent) child;

  constructor() { }

  message:string;

  ngAfterViewInit() {
    this.message = this.child.message
  }
}
```

child.component.ts

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

@Component({
  selector: 'app-child',
  template: `
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  message = 'Hola Mundo!';

  constructor() { }

}
```

Child to Parent: Sharing Data via Output() and EventEmitter

Another way to share data is to emit data from the child, which can be listed to by the parent. This approach is ideal when you want to share data changes that occur on things like button clicks, form entires, and other user events.

In the parent, we create a function to receive the message and set it equal to the message variable.

In the child, we declare a messageEvent variable with the Output decorator and set it equal to a new event emitter. Then we create a function named sendMessage that calls emit on this event with the message we want to send. Lastly, we create a button to trigger this function.

The parent can now subscribe to this messageEvent that's outputted by the child component, then run the receive message function whenever this event occurs.

parent.component.ts

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

@Component({
  selector: 'app-parent',
  template: `
    Message: {{message}}
    <app-child (messageEvent)="receiveMessage($event)"></app-child>
```

```
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {

  constructor() { }

  message:string;

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

child.component.ts

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

@Component({
  selector: 'app-child',
  template: `
      <button (click)="sendMessage()">Send Message</button>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  message: string = "Hola Mundo!"

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

  constructor() { }

  sendMessage() {
    this.messageEvent.emit(this.message)
  }
}
```

Unrelated Components: Sharing Data with a Service

When passing data between components that lack a direct connection, such as siblings, grandchildren, etc, you should you a shared service. When you have data that should aways been in sync, I find the **RxJS BehaviorSubject** very useful in this situation.

You can also use a regular RxJS Subject for sharing data via the service, but here's why I prefer a BehaviorSubject.

- It will always return the current value on subscription - there is no need to call `onnext`
- It has a `getValue()` function to extract the last value as raw data.
- It ensures that the component always receives the most recent data.

In the service, we create a private BehaviorSubject that will hold the current value of the message. We define a currentMessage variable handle this data stream as an observable that will be used by the components. Lastly, we create function that calls next on the BehaviorSubject to change its value.

The parent, child, and sibling components all receive the same treatment. We inject the

DataService in the constructor, then subscribe to the currentMessage observable and set its value equal to the message variable.

Now if we create a function in any one of these components that changes the value of the message. when this function is executed the new data it's automatically broadcast to all other components.

data.service.ts

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable()
export class DataService {

  private messageSource = new BehaviorSubject('default message');
  currentMessage = this.messageSource.asObservable();

  constructor() { }

  changeMessage(message: string) {
    this.messageSource.next(message)
  }

}
```

parent.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";

@Component({
  selector: 'app-parent',
  template: `
    {{message}}
  `,
  styleUrls: ['./sibling.component.css']
})
export class ParentComponent implements OnInit {

  message:string;

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.currentMessage.subscribe(message => this.message = message)
  }

}
```

sibling.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";

@Component({
  selector: 'app-sibling',
  template: `
    {{message}}
    <button (click)="newMessage()">New Message</button>
  `,
```

```
  styleUrls: ['./sibling.component.css']
})
export class SiblingComponent implements OnInit {

  message:string;

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.currentMessage.subscribe(message => this.message = message)
  }

  newMessage() {
    this.data.changeMessage("Hello from Sibling")
  }

}
```

Q&A Chat

Have Questions? Let's chat about this post

SIGNUP FOR SLACK Copy the link below and paste it into the **#questions** channel in Slack 👇

`@Question https://fireship.io/lessons/sharing-data-between-angular-components-four-methods/ your question...?` COPY AND ASK