Observables in Angular

Angular makes use of observables as an interface to handle a variety of common asynchronous operations. For example:

- You can define custom events that send observable output data from a child to a parent component.
- The HTTP module uses observables to handle AJAX requests and responses.
- The Router and Forms modules use observables to listen for and respond to user-input events.

Transmitting data between components

Angular provides an `EventEmitter` class that is used when publishing values from a component through the `@Output()` decorator. `EventEmitter` extends RxJS `Subject`, adding an `emit()` method so it can send arbitrary values. When you call `emit()`, it passes the emitted value to the `next()` method of any subscribed observer.

A good example of usage can be found in the EventEmitter documentation. Here is the example component that listens for open and close events:

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Here is the component definition:

EventEmitter

```
@Component({
  selector: 'zippy',
  template: `
  <div class="zippy">
    <div (click)="toggle()">Toggle</div>
    <div [hidden]="!visible">
      <ng-content></ng-content>
    </div>
  </div>`})

export class ZippyComponent {
  visible = true;
  @Output() open = new EventEmitter<any>();
  @Output() close = new EventEmitter<any>();

  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

HTTP

Angular's `HttpClient` returns observables from HTTP method calls. For instance, `http.get('/api')` returns an observable. This provides several advantages over promise-based HTTP APIs:

- Observables do not mutate the server response (as can occur through chained `.then()` calls on promises). Instead, you can use a series of operators to transform values as needed.
- HTTP requests are cancellable through the `unsubscribe()` method.
- Requests can be configured to get progress event updates.
- Failed requests can be retried easily.

Async pipe

The AsyncPipe subscribes to an observable or promise and returns the latest value it has emitted. When a new value is emitted, the pipe marks the component to be checked for changes.

The following example binds the `time` observable to the component's view. The observable continuously updates the view with the current time.

Using async pipe

```
@Component({
  selector: 'async-observable-pipe',
  template: `<div><code>observable|async</code>:
       Time: {{ time | async }}</div>`
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>(observer => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Router

`Router.events` provides events as observables. You can use the `filter()` operator from RxJS to look for events of interest, and subscribe to them in order to make decisions based on the sequence of events in the navigation process. Here's an example:

Router events

```
import { Router, NavigationStart } from '@angular/router';
import { filter } from 'rxjs/operators';

@Component({
  selector: 'app-routable',
  templateUrl: './routable.component.html',
  styleUrls: ['./routable.component.css']
})
export class Routable1Component implements OnInit {

  navStart: Observable<NavigationStart>;

  constructor(private router: Router) {
    // Create a new Observable that publishes only the NavigationStart event
    this.navStart = router.events.pipe(
      filter(evt => evt instanceof NavigationStart)
    ) as Observable<NavigationStart>;
  }

  ngOnInit() {
    this.navStart.subscribe(evt => console.log('Navigation Started!'));
  }
}
```

The ActivatedRoute is an injected router service that makes use of observables to get information about a route path and parameters. For example, `ActivatedRoute.url` contains an observable that reports the route path or paths. Here's an example:

ActivatedRoute

```
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-routable',
  templateUrl: './routable.component.html',
  styleUrls: ['./routable.component.css']
})
export class Routable2Component implements OnInit {
  constructor(private activatedRoute: ActivatedRoute) {}

  ngOnInit() {
    this.activatedRoute.url
      .subscribe(url => console.log('The URL changed to: ' + url));
  }
}
```

Reactive forms

Reactive forms have properties that use observables to monitor form control values. The `FormControl` properties `valueChanges` and `statusChanges` contain observables that raise change events. Subscribing to an observable form-control property is a way of triggering application logic within the component class. For example:

Reactive forms

```
import { FormGroup } from '@angular/forms';

@Component({
  selector: 'my-component',
  template: 'MyComponent Template'
})
export class MyComponent implements OnInit {
  nameChangeLog: string[] = [];
  heroForm: FormGroup;

  ngOnInit() {
    this.logNameChange();
  }
  logNameChange() {
    const nameControl = this.heroForm.get('name');
    nameControl.valueChanges.forEach(
      (value: string) => this.nameChangeLog.push(value)
    );
  }
}
```