Jonathan Roland

Web 425 Angular with TypeScript

Prof. Richard Krasso

1/22/2021

Discussion 4.1 – Observables and Observable Streams

Reactive programming seeks to optimize the responsiveness of event-driven applications. This builds from the Observer/Observable pattern. This pattern describes the asynchronous relationship between a data stream and a subscriber. The key to reactive programming is understanding the elements in this model and tuning our business logic to react to events in the most optimal way.

An observer can also be called a subscriber. Its purpose is to listen for a particular data stream then carry out actions based on what the stream is doing. An observable is a function that can asynchronously emit a data stream. A stream is a serial transmission of data from a data source. Observables can only emit one of three things in a stream: the next element, an error, or an end-of stream signal. Observers can thus only fulfill three purposes: handling the next emitted element, handling an error, and handling the closing of a stream.

Observables can be described as ‘hot’ or ‘cold’. Hot observables will start streaming data even if there are no subscribers to listen for it, while a cold observable will only stream when it has subscribers. In Angular, subscribers use the subscribe() and unsubscribe() functions to set this relationship with cold observables.

To optimize streams, observable objects can use map() and filter() functions to prepare the streamed data for the observer. The map() function allows each element in an array to have a transformative function mapped to it. The filter() function applies to each element, similarly to the map() function, but is used to apply custom filter functions to the stream so that only those filtered elements are transmitted.

Putting this all together, subscribing to a stream (an observer listening to an observable) creates a relationship where your code shows interest in receiving data from that stream. When it subscribes, it declares what operations should be done to that data before emitting. This saves the observer from having to process the data on receipt. The observer will at least handle the received data in some way and may have logic to handle errors or the end of the stream. By planning ahead for the data we want the stream to transmit and how to handle it, we can optimize the responsiveness of our application and build it to react to the behavior of the stream.

**References:**

Fain, Y., & Moiseev, A. (2016). *Angular 2 Development with TypeScript* (1st ed.). Manning Publications.