Sun*

Get Started with RxSwift

Outline

1. What is RxSwift?
2. Introduction to asynchronous programming
3. Cocoa and UIKIt asynchronous APIs
4. Asynchronous programming glossary

1. What is RxSwift?

- **RxSwift** is a library for composing asynchronous and event-based code by using observable sequences and functional style operators, allowing for parameterized execution via schedulers.

- In its essence, simplifies developing asynchronous programs by allowing your code to react to new data and process it in a sequential, isolated manner.

2. Introduction to asynchronous programming

1. Problem with synchronous
2. Asynchronous programming

2.1. Problem with synchronous

- Normally, a given program's code runs straight along, with only one thing happening at once. If a function relies on the result of another function, it has to wait for the other function to finish and return, and until that happens, the entire program is essentially stopped from the perspective of the user.

- This is a frustrating experience and isn't a good use of computer processing power — especially in an era in which computers have multiple processor cores available. There's no sense sitting there waiting for something when you could let the other task chug along on another processor core and let you know when it's done.

2.2. Asynchronous programming

- Asynchronous programming is a means of parallel programming in which a unit of work runs separately from the main application thread and notifies the calling thread of its completion, failure or progress.

- Writing code that truly runs in parallel, however, is rather complex, especially when different bits of code need to work with the same pieces of data. It’s hard to argue about which piece of code updates the data first, or which code read the latest value.

3. Cocoa and UIKit asynchronous APIs

- Apple provides lots of APIs and the iOS SDK that help you write asynchronous code:
  - **NotificationCenter**: To execute a piece of code any time an event of interest happens, such as the user changing the orientation of the device or the software keyboard showing or hiding on the screen.
  - **The delegate pattern**: Lets you define an object that acts on behalf, or in coordination with, another object.
  - **Grand Central Dispatch**: To help you abstract the execution of pieces of work. You can schedule code to be executed sequentially in a serial queue, or run a multitude of tasks concurrently on different queues with different priorities.
  - **Closures**: To create detached pieces of code that you can pass around in your code, so other objects can decide whether to execute it or not, how many times, and in what context.

4. Asynchronous programming glossary

1. State, and specifically, shared mutable state
2. Imperative programming
3. Side effects
4. Declarative code
5. Reactive systems

4.1. State, and specifically, shared mutable state

- **State** is somewhat difficult to define. For instance when you start your laptop, it runs fine,but after you use it for a few days or even weeks ,it might start behaving weirdly or abruptly hang and refuse to speak to you ,no any response whatever you send any orders,it stops working for you, but once you restart your laptop, perhaps it works just fine .That's **State**. The data in memory ,the one stored on disk, all the artifacts of reacting to users input, all traces that remain after fetching data from server - the sum of these is the state of your laptop.
- Managing the state of your app, especially when shared between multiple asynchronous components, is one of the issues.

4.2. Imperative programming

- **Imperative programming** is a programming paradigm that uses statements to change the program's state.

- **Imperative code** is similar to the code that your computer understands. All the CPU does is follow lengthy sequences of simple instructions. The issue is that it gets challenging for humans to write imperative code for complex, asynchronous apps — especially when shared mutable state is involved.

4.3. Side effects

- **Side effect** are any change to the state outside of the current scope.

- **Side effects** are not bad in themselves. After all, causing side effects is the ultimate goal of *any* program!

- The important aspect of producing side effects is doing so in a *controlled* way. You need to be able to determine which pieces of code cause side effects, and which simply process and output data.

4.4. Declarative code

- In functional programming, you aim to minimize the code that causes side effects.

- **Declarative code** lets you define pieces of behavior. RxSwift will run these behaviors any time there's a relevant event and provide an immutable, isolated piece of data to work with.

4.5. Reactive systems

- **Reactive systems** is a rather abstract term and covers web or iOS apps that exhibit most or all of the following qualities:
  - **Responsive**: Always keep the UI up to date, representing the latest app state.
  - **Resilient**: Each behavior is defined in isolation and provides for flexible error recovery.
  - **Elastic**: The code handles varied workload, often implementing features such as lazy pull-driven data collections, event throttling, and resource sharing.
  - **Message driven**: Components use message-based communication for improved reusability and isolation, decoupling the lifecycle and implementation of classes.

Question & Answer?