Multithreading, Networking and Codable

March 6, 2018

* Adopted from Stanford's CS193p Slides

Multithreading

Queues

- Multithreading is mostly about "queues" in iOS.
- Functions (usually closures) are simply lined up in a queue (like at the movies!).
- Then those functions are pulled off the queue and executed on an associated thread(s).
- Queues can be "serial" (one closure a time) or "concurrent" (multiple threads servicing it).

Multithreading

Main Queue

```swift
let mainQueue = DispatchQueue.main
```

- Special serial queue
- All UI activity must occur on main queue and main queue only
- All non-UI activity that is time consuming should not occur on main queue
- Functions are pulled off and worked on in the main queue only when it is quiet

Multithreading

Global Queues

```
let backgroundQueue = DispatchQueue.global(qos: DispatchQoS)
```

- Shared, global, concurrent
- Ensures UI to be highly responsive
- Commonly used for non-main-queue work instead of custom queues
- Almost always what you will use to get activity off the main queue

Multithreading

Placing Code On Queue

```
queue.async { ... }
queue.sync  { ... }
```

- Multithreading is the process of putting closures into these queues
- Two primary ways of putting a closure onto a queue
  - Async (Preferered): Plop a closure onto a queue and keep running on the current queue
  - Sync: Block the current queue waiting until the closure finishes on that other queue

Multithreading

Multithreaded Networking API

```swift
let session = URLSession(configuration: .default)
if let url = URL(string: "https://iosgatech.xyz/...") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        // do something
    }
    task.resume()
}
```

- API lets you fetch the contents of an http or https URL into a Data off the main queue

Multithreading

Multithreaded Networking API

```
let session = URLSession(configuration: .default)
if let url = URL(string: "https://iosgatech.xyz/...") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        // I want to do UI things here
        // with the data of the download
        // can I?
    }
    task.resume()
}
```

- The code will be run off the main queue
- Use a variant of this API that lets you specify the queue to run on (main queue)

Multithreading

Multithreaded Networking API

```
let session = URLSession(configuration: .default)
if let url = URL(string: "https://iosgatech.xyz/...") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        DispatchQueue.main.async {
            // do UI stuff here
        }
    }
    task.resume()
}
```

- UI code has been dispatched back to the main queue
- Can legally do UI stuff on main queue

Multithreading

Order

```
0  let session = URLSession(configuration: .default)
1  if let url = URL(string: "https://iosgatech.xyz/...") {
2      let task = session.dataTask(with: url) { (data: Data?, response, error) in
5          // parse the data
6          DispatchQueue.main.async {
8              // do UI stuff here
7          }
3          print("Finished parsing the data, but no UI updates yet")
       }
4      task.resume()
   }
```

Data to String

Create String object out of fetched data

```
let jsonString = String(data: jsonData!, encoding: .utf8)
// JSON is always utf8
```

Codable

Convert from JSON to Codable object or struct

```
if let myObject: MyType = try? JSONDecoder().decode(MyType.self, from: jsonData!) {
}
```

Codable

JSON is not strongly typed

- Date or URL are just strings
- Swift handles all this automatically and is even configurable

```
let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
decoder.dateDecodingStrategy = .secondsSince1970
```

Codable

Catch errors during a decoding

- Decode throws an enum of type DecodingError.
- We can get the associated values of the enum similar to how we do with switch.

```
do {
    let object = try JSONDecoder().decode(MyType.self, from: jsonData!)
    // success, do something with object
} catch DecodingError.keyNotFound(let key, let context) {
    print("couldn’t find key \(key) in JSON: \(context.debugDescription)")
} catch DecodingError.valueNotFound(let type, let context) {
} catch DecodingError.typeMismatch(let type, let context) {
} catch DecodingError.dataCorrupted(let context) {
}
```

Codable

Make Types Codable

- If your vars are all also Codable (standard types all are), then you’re done!

```
{
    "someDate" : "2017-11-05T16:30:00Z",
    "someString" : "Hello",
    "other" : <whatever SomeOtherType looks like in JSON>
}
```

```swift
struct MyType : Codable {
    var someDate: Date
    var someString: String
    var other: SomeOtherType // SomeOtherType has to be Codable too!
}
```

Codable

Make Types Codable

- JSON keys might have different names than your var names (or not be included).
- someDate might be some_date
- Configure by adding a private enum to your type called CodingKeys like this

```
struct MyType : Codable {
    var someDate: Date
    var someString: String
    var other: SomeOtherType // SomeOtherType has to be Codable too!

    private enum CodingKeys : String, CodingKey {
        case someDate = "some_date"
        // note that the someString var will now not be included in the JSON
        case other // this key is also called "other" in JSON
   }
}
```

Codable

- You can participate directly in the decoding by implementing the decoding initializer …
- Note that this init throws, so we don’t need do { } inside it (it will just rethrow).
- Also note the "keys" are from the CodingKeys enum on the previous slide (e.g. .someDate).
- Don’t call super.init with your own decoder (use your container’s superDecoder()).

```
class MyType : Codable {
  var someDate: Date
  var someString: String
  var other: SomeOtherType // SomeOtherType has to be Codable too!

  init(from decoder: Decoder) throws {
    let container = try decoder.container(keyedBy: CodingKeys.self)
    someDate = try container.decode(Date.self, forKey: .someDate)
    // process rest of vars, perhaps validating input, etc. …
    let superDecoder = try container.superDecoder()
    try super.init(from: superDecoder) // only if class
  }
}
```