UNTANGLING CALLBACK SPAGHETTI

WITH REACTIVECOCOA

Sam Ritchie

codesplice

```swift
func merge<T: Comparable>(left: [T], _ right:[T]) -> [T] {
    guard left.count > 0 else { return right }
    guard right.count > 0 else { return left }
    if left[0] < right[0] {
        return [left[0]] + merge(Array(left.suffixFrom(1)), right)
    } else {
        return [right[0]] + merge(left, Array(right.suffixFrom(1)))
    }
}

extension Array where Element: Comparable {
    func foo() -> [Element] {
        guard count > 1 else { return self }

        let left = Array(prefix(count / 2))
        let right = Array(suffixFrom(count / 2))
        return merge(left.foo(),right.foo())
    }
}
```

#include "sorting.h"

```
func locationManager(manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
    if let loc = locations.first {
        WebApi.sharedApi.updateLocationOnServer(loc, success: {
            print("Updated Location!")
        }, error: { e in
            print("Error!")
        })
    }
}
```

```
let throttleInterval = NSTimeInterval(60)
var lastLocationUpdate: NSDate?

func locationManager(manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
    if let loc = locations.first {
        if lastLocationUpdate == nil ||
lastLocationUpdate!.timeIntervalSinceNow < -throttleInterval {
            lastLocationUpdate = NSDate()
            WebApi.sharedApi.updateLocationOnServer(loc, success: {
                print("Updated Location!")
            }, error: { e in
                print("Error!")
            })
        }
    }
}
```

```
let throttleInterval = NSTimeInterval(60)
var lastLocationUpdate: NSDate?

func locationManager(manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        if lastLocationUpdate == nil ||
lastLocationUpdate!.timeIntervalSinceNow < -throttleInterval {
            lastLocationUpdate = NSDate()
            WebApi.sharedApi.updateLocationOnServer(loc, success: {
                print("Updated Location!")
            }, error: { e in
                print("Error!")
            })
        }
    }
}
```

```
let throttleInterval = NSTimeInterval(60)
var lastLocationUpdate: NSDate?

func locationManager(manager: CLLocationManager, didUpdateLocations locations:
[CLLocation]) {
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        if lastLocationUpdate == nil ||
lastLocationUpdate!.timeIntervalSinceNow < -throttleInterval {
            lastLocationUpdate = NSDate()
            WebApi.sharedApi.updateLocationOnServer(loc, success: {
                print("Updated Location!")
            }, error: { e in
                print("Error!")
                self.lastLocationUpdate = nil
            })
        }
    }
}
```

```swift
let throttleInterval = NSTimeInterval(60)
var lastLocationUpdate: NSDate?
var apiErrors = 0

func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
{
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        if lastLocationUpdate == nil || lastLocationUpdate!.timeIntervalSinceNow < -
throttleInterval {
            lastLocationUpdate = NSDate()
            WebApi.sharedApi.updateLocationOnServer(loc, success: {
                print("Updated Location!”)
                self.apiErrors = 0
            }, error: { e in
                print("Error!")
                self.apiErrors++
                if self.apiErrors < 3 { self.lastLocationUpdate = nil }
            })
        }
    }
}
```

```swift
let throttleInterval = NSTimeInterval(60)
var lastLocation: CLLocation?
var apiErrors = 0
var timer: NSTimer!

timer = NSTimer.scheduledTimerWithTimeInterval(throttleInterval, target: self, selector: "timerDidFire:",
userInfo: nil, repeats: true)

func timerDidFire(timer: NSTimer) {
    WebApi.sharedApi.updateLocationOnServer(self.lastLocation, success: {
        print("Updated Location!")
        self.apiErrors = 0
    }, error: { e in
        print("Error!")
        self.apiErrors++
        if self.apiErrors < 3 { timer.fire() }
    })
}

func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        lastLocation = loc
    } else {
        lastLocation = nil
    }
}
```

```swift
let throttleInterval = NSTimeInterval(60)
var lastLocation: CLLocation?
var apiErrors = 0
var timer: NSTimer!

timer = NSTimer.scheduledTimerWithTimeInterval(throttleInterval, target: self, selector: "timerDidFire:",
userInfo: nil, repeats: true)

func timerDidFire(timer: NSTimer) {
    WebApi.sharedApi.updateLocationOnServer(self.lastLocation, success: {
        print("Updated Location!")
        self.apiErrors = 0
    }, error: { e in
        print("Error!")
        self.apiErrors++
        if self.apiErrors < 3 { timer.fire() }
    })
}

func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        lastLocation = loc
    }  else if let loc = lastLocation where loc.timestamp.timeIntervalSinceNow < -throttleInterval {
        lastLocation = nil
    }
}
```

```
let throttleInterval = NSTimeInterval(60)
var lastLocation: CLLocation?
var apiErrors = 0
var timer: NSTimer!

timer = NSTimer.scheduledTimerWithTimeInterval(throttleInterval, target: self, selector: "timerDidFire:", userInfo: nil,
repeats: true)

func timerDidFire(timer: NSTimer) {
    if let loc = lastLocation where loc.timestamp.timeIntervalSinceNow < -throttleInterval {
        lastLocation = nil
    }
    WebApi.sharedApi.updateLocationOnServer(self.lastLocation, success: {
        print("Updated Location!")
        self.apiErrors = 0
    }, error: { e in
        print("Error!")
        self.apiErrors++
        if self.apiErrors < 3 { timer.fire() }
    })
}

func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    if let loc = locations.first where loc.horizontalAccuracy < 100 {
        lastLocation = loc
    }  else if let loc = lastLocation where loc.timestamp.timeIntervalSinceNow < -throttleInterval {
        lastLocation = nil
    }
}
```

WHO'S DOWN WITH FRP?

REACTIVECOCOA

SIGNALS

DEMONSTRATION

RESOURCES

- reactivecocoa.io, or github.com/ReactiveCocoa/ReactiveCocoa
- blog.scottlogic.com
- www.reactivemanifesto.org
- Principles of Reactive Programming (Coursera)

THANKS!

@FakeSamRitchie

http://codesplice.com.au