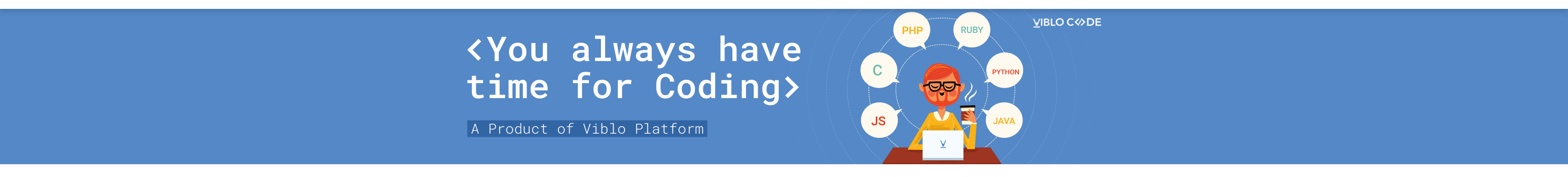

<You always have time for Coding>
A Product of Viblo Platform
VIBLO CODE
PHP
RUBY
C
PYTHON
JS
JAVA

Vinh Nhon @thevinh92 Theo dõi
323 19 36

Đã đăng vào thg 12 25, 2017 9:23 AM - 5 phút đọc
174 1 5

Local Reasoning in Swift

+3

Bài đăng này đã được cập nhật cách đây **3 năm** kể từ khi nó được cập nhật lần cuối.

Theo tài liệu từ medium Swift là 1 một ngôn ngữ mạnh mẽ và hàm xúc ngắn gọn, sau đây chúng ta sẽ cùng nói về các đặc điểm của swift mà có thể khiến cho code của bạn dễ đọc hơn.

Better Buttons

Khi bạn đọc 1 tutorial nào đó về UIButton, bạn sẽ thường thấy hướng dẫn code như sau khi muốn print ra một message khi 1 button được press:

```
override func viewDidLoad()
    let button = UIButton()
    button.addTarget(self, action: #selector(uiButtonAction), for: .touchUp
Inside)
    view.addSubview(button)
}

@objc private func uiButtonAction() {
    print("UIButton pressed!")
}
```

Mặc dù đoạn code trên không có vấn đề gì, nhưng ta có thể thấy rằng: có quá nhiều code để liên hệ giữa đoạn khai báo selector và action của button. Nếu 1 người đọc code của bạn lần đầu, họ sẽ phải mất thời gian để track xem method `uiButtonAction()` ở đâu và làm gì. Thay vào đó, nếu ta viết code như sau:

```
override func viewDidLoad() {
    let button = MyButton()
    button.action = {
        print("MyButton pressed!")
    }
    view.addSubview(button)
}
```

Sử dụng một closure để gán action cho button. Mọi thứ ngắn gọn và dễ hiểu. Tuy nhiên code này không chạy đc =)), `UIKit` chưa cho phép ta viết ntn, nhưng bạn có thể khiến một `UIButton` hoạt động theo kiểu như vậy. Ngoài việc tất cả code đều chứa trong 1 functions thì việc sử dụng closure cũng khá rõ ràng trong việc thể hiện: điều gì sẽ xảy ra khi tap vào button. Implement 1 closure cho việc tap vào button là khá đơn giản. Tạo 1 subclass của `UIButton` với một `action` property là một closure.

```
class MyButton: UIButton {

    var action: (() -> ())?

    override init(frame: CGRect) {
        super.init(frame: frame)
        sharedInit()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        sharedInit()
    }

    private func sharedInit() {
        addTarget(self, action: #selector(touchUpInside), for: .touchUpInsi
de)
    }

    @objc private func touchUpInside() {
        action?()
    }

}
```

Với cách làm này bạn có thể mở rộng thêm với các closures cho các action khác như: `touchDown`, `touchDragExit`,...

Local Reasoning

Ý tưởng của `Local Reasoning` là: người khác khi đọc code của bạn sẽ có thể hiểu được trược tiếp ngay lập tức mà không cần phải đi truy tìm xem ở các nơi khác nữa để biết code hoạt động ntn. Tại WWDC 2016, Các kỹ sư của Apple đã có một bài nói chuyện với tựa đề "Protocol and Value Oriented Programming in UIKit Apps" họ có thảo luận về khái niệm `Local Reasoning`. Họ trình bày về protocols và cách sử dụng các lợi ích của nó so với phong cách kế thừa truyền thống. Thật khó để gọi cách tổ chức code của ai đó là không hợp lý hay là đần độn vì nó thường dựa trên sở thích cá nhân. Tuy nhiên, chúng ta đều đồng ý rằng các tổ chức code tốt nhất là khi mà người đọc có thể hiểu được code mà không phải đi qua quá nhiều codebase.

Lazy Closures

Hãy cùng xem xét một design pattern khác của Swift mà tập trung vào việc cải thiện `Local Reasoning` Hãy xem ví dụ về việc setup trong method `viewDidLoad`

```
var myCustomView = UIView()

override func viewDidLoad() {
    myCustomView.backgroundColor = UIColor.blue
    myCustomView.layer.cornerRadius = myCustomView.bounds.width / 2
}
```

Giống như ví dụ về button, đoạn code phía trên vẫn chạy bình thường. Nhưng hãy chú ý rằng chúng ta đã chia ra làm 2 phần: khai báo cho `stored property` và setup `myCustomView` ở bên trong một function. Đây chings à vấn đề: việc tìm kiếm tất cả code liên quan đến view thường phải bắt buộc search. Thay vào đó, ta có thể sử dụng closure để init view:

```
lazy var myCustomView: UIView = {
    let view = UIView()
    view.backgroundColor = UIColor.blue
    view.layer.cornerRadius = myCustomView.bounds.width / 2
    return view
}()
```

closure được marked là `lazy` để `self` có thể truy nhập đc vào bên trong closure. Điều này cho phép properties có thể reference đến bất kỳ 1 constants nào khai báo trong self và set target action cho bất cứ control nào. Cách tiếp cận này cũng hoạt động cả với storyboard outlets. Thay vì sử dụng closure, ta dùng `didSet` property observer mà sẽ được gọi khi outlet được set bởi storyboard:

```
@IBOutlet weak var myCustomView: UIView! {
    didSet {
        view.backgroundColor = UIColor.blue
        view.layer.cornerRadius = myCustomView.bounds.width / 2
    }
}
```

Cái này đặc biệt hữu dụng cho những thuộc tính không thể set trong storyboard như `corner radiant`, `gradients`,... Nó sẽ `localizes our changes to the view itself` Thay vì chia nhỏ code trong file, chúng có thể đặt nó ở cùng 1 chỗ.

Protocol Conformance Extensions

Protocols thường được sử dụng trong thư viện standard của Swift hoặc API của Cocoa Touch. Nó cung cấp 1 sự đảm bảo rằng trong `compile-time`, 1 đối tượng nào đó sẽ chắc chắn có 1 property hoặc functions nào đó. Chúng ta sẽ cùng xem xét tiếp 1 pattern phổ biến khác: implement delegate và data source của collection view trong viewcontroller:

```
class MyViewController: UIViewController, UICollectionViewDelegate, UIColle
ctionViewDataSource {

    var foods = ["🍎", "🍌", "🍓"]

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
 indexPath: IndexPath) {
        print("selected item at \(indexPath)")
    }

    func collectionView(_ collectionView: UICollectionView, numberOfItemsIn
Section section: Int) -> Int {
        return foods.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt i
ndexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
"food", for: indexPath) as? FoodCollectionViewCell
        cell?.food = foods[indexPath.item]
        return cell
    }

}
```

Dĩ nhiên nó vẫn hoạt động, tuy nhiên những functions của delegate khá là dài và có xu hướng tăng kích thước của đối tượng kiểm soát chúng. Khá là khó để nhìn vào functions thuộc về protocol nào... Thay vào đó, chúng ta khai báo `protocol conformance` ở trong extension để group code lại với nhau tốt hơn.

```
extension ViewController: UICollectionViewDelegate {

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
 indexPath: IndexPath) {
        print("selected item at \(indexPath)")
    }

}

extension ViewController: UICollectionViewDataSource {

    func collectionView(_ collectionView: UICollectionView, numberOfItemsIn
Section section: Int) -> Int {
        return foods.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt i
ndexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
"food", for: indexPath) as? FoodCollectionViewCell
        cell?.food = foods[indexPath.item]
        return cell
    }

}
```

Đã đăng ký Bản quyền

MỤC LỤC

Better Buttons
Local Reasoning
Lazy Closures
Protocol Conformance Extensions

CÁC TỔ CHỨC ĐƯỢC ĐỀ XUẤT

Sun* AI Research Team
290 29 463

Sun* Cyber Security Team
176 10 269

Avengers Group
55 35 119

Bài viết liên quan

Tạm biệt OOP, chào đón POP
Phạm Quang Huy
4 phút đọc
3577 7 2 7

Reference Cycle for Closures
Phan Huynh Thien An
3 phút đọc
454 4 0 1

Weak attribute và ứng dụng thực tế trong iOS
Lê Văn Tuấn
3 phút đọc
766 2 0 2

Sử dụng Protocols, Delegates với UITableView
Vu Ngoc Anh
4 phút đọc
1072 2 0 1

Bài viết khác từ Vinh Nhon

Swift Tutorial: Cách thích hợp để request người dùng revie...
Vinh Nhon
5 phút đọc
346 5 0 2

Reactive <~ Closure
Vinh Nhon
5 phút đọc
157 0 0 0

The secret world of NSTimer
Vinh Nhon
11 phút đọc
190 1 0 1

Clean iOS Localizable Files
Vinh Nhon
8 phút đọc
192 2 1 2

Bình luận

Viết Xem trước

Viết phản hồi...

Bình luận

Quoc Nguyen @quocnb thg 1 5, 2018 4:08 PM

bài hay a ơi

0 | Trả lời Chia sẻ

© Viblo 2021