

Patterns in Swift

ishkawa



Yosuke Ishikawa

ishikawa

Developer Program Member

LINE

Tokyo, Japan

y@ishikawa.org

<http://ishikawa.org>

Joined on Aug 9, 2011

260

307

44

Followers

Starred

Following

Organizations



Contributions

Repositories

Public activity

Edit profile

Popular repositories

[APIKit](#)

A networking library for building type safe web...

352



[ISRefreshControl](#)

iOS4-compatible UIRefreshControl

235



[ISColumnsController](#)

paginated container view controller.

121



[ISDiskCache](#)

LRU disk cache for iOS.

71



[UIViewControllerAnimatedTransitioning](#)

extension for custom transition by blocks.

54



Repositories contributed to

[CocoaPods/Specs](#)

The CocoaPods Master Repo

2,527



[antitypical/Result](#)

Swift type modelling the success/failure of arbit...

470



[AliSoftware/OHHTTPStubs](#)

1,686



Stub your network requests easily! Test your a...

[Alamofire/Alamofire](#)

Elegant HTTP Networking in Swift

10,595



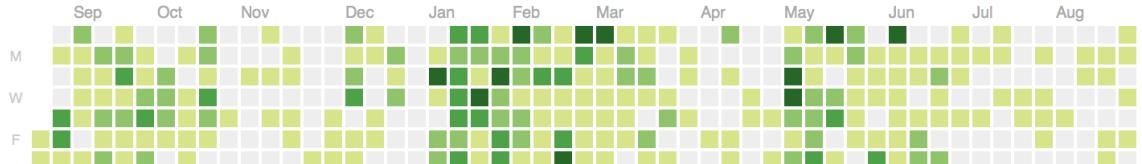
[matteocrippa/awesome-swift](#)

A collaborative list of awesome swift resource...

4,236



Contributions



Summary of pull requests, issues opened, and commits. [Learn how we count contributions.](#)

Less More

Contributions in the last year

1,699 total

Aug 29, 2014 – Aug 29, 2015

Longest streak

64 days

January 9 – March 13

Current streak

0 days

Last contributed a day ago

Patterns in Swift

- if-case
- switch-case
- for-case

```
let something = ...
if case let foo as Foo = something {
    ...
}
```

- do-catch

```
do {
    try doSomething()
} catch let error as MyError {
    print(error)
}
```

問題点

- 文法がよくわからん

例 1

enumのassociated value

```
enum Either<A, B> {
    case Left(A)
    case Right(B)
}
```

```
let either: Either<A, B> = ...

switch either {
case .Left(let a):
    print("A: \(a)")

case .Right(let b):
    print("B: \(b)")
}
```

これでもいいの？

```
enum Either<A, B> {
    case Left(A)
    case Right(B)
}
```

```
let either: Either<A, B> = ...

switch either {
case let .Left(a):
    print("A: \(a)")

case let .Right(b):
    print("B: \(b)")
}
```

例 2

Optional

```
let tuple: (A?, B) = ...  
  
switch tuple {  
case (.Some(let a), let b):  
    print("A: \(a), B: \(b)")  
  
case (.None, let b):  
    print("A: nil, B: \(b)")  
}
```

これでもいいの？

```
let tuple: (A?, B) = ...  
  
switch tuple {  
case (let a?, let b):  
    print("A: \(a), B: \(b)")  
  
case (.None, let b):  
    print("A: nil, B: \(b)")  
}
```

これも同じ？？？？

```
let tuple: (A?, B) = ...  
  
switch tuple {  
case (let a?, let b):  
    print("A: \(a), B: \(b)")  
  
case (let a, let b):  
    print("A: \(a), B: \(b)")  
}
```


パターンの文法を知らないと…

- 込み入ったswitch文が読めない
- 愚直なswitch文しか書けない
- ググってみるけどよくわからなくて時間を浪費する

結果的に劣等感に悩まされる

パターンの文法はそんなに難しくないけど

初見で察することができるほど簡単ではない

あらためてパターンってなに？

- 具体的な値ではなく値の性質を表すもの
- 具体的な値がマッチしているかどうかテストできる
- 数種類のパターンがあり組み合わせが可能
 - Wildcard Pattern
 - Identifier Pattern
 - Value-Binding Pattern
 - Tuple Pattern
 - Enumeration Case Pattern
 - Optional Pattern
 - Type-Casting Pattern
 - Expression Pattern

組み合わせ可能

```
let tuple: (A?, B) = ...  
  
switch tuple {  
case (let a?, let b):  
    print("A: \(a), B: \(b)")  
  
case (let a, let b):  
    print("A: \(a), B: \(b)")  
}
```

`case (let a?, let b)`では以下が組み合わされている

- Tuple Pattern
- Identifier Pattern
- Value-Binding Pattern
- Optional Pattern

各パターンの仕様

書くのが大変なので公式を参照します

- The Swift Programming Language
 - Language Reference
 - Patterns

どれも結構使う

よくわからなかつたやつを振り返ると...

例1: パターンの組み合わせの順序が異なる

```
let either: Either<A, B> = ...  
  
switch either {  
    case .Left(let a):  
        print("A: \(a)")  
  
    case .Right(let b):  
        print("B: \(b)")  
}
```

```
switch either {  
    case let .Left(a):  
        print("A: \(a)")  
  
    case let .Right(b):  
        print("B: \(b)")  
}
```

例2: 同じもの違うパターンで表現している

```
let tuple: (A?, B) = ...
switch tuple {
case (.Some(let a), let b):
    print("A: \(a), B: \(b)")

case (.None, let b):
    print("A: nil, B: \(b)")
}
```

```
switch tuple {
case (let a?, let b):
    print("A: \(a), B: \(b)")

case (.None, let b):
    print("A: nil, B: \(b)")
}
```

```
switch tuple {
case (let a?, let b):
    print("A: \(a), B: \(b)")

case (let a, let b):
    print("A: \(a), B: \(b)")
}
```

ところで、普通のswitchはどうなってたの？

IntとかRange<Int>とか

```
let int: Int = ...  
  
switch int {  
case 1:  
    print("1")  
  
case 3..  
    print("3..  
}  
    <10":  
    print("3..<10")  
}
```

どのパターンが使われているかわかりますか？

正解はexpression pattern

```
func ~=<T: Equatable>(a: T, b: T) -> Bool  
func ~=<I: ForwardIndexType where I: Comparable>  
    (pattern: Range<I>, value: I) -> Bool
```

標準ライブラリで`~=`が定義されている

アプリ開発での使いどころ

UITableViewController

```
class MyTableViewController: UITableViewController {
    // セルがA, B, Cの三種類あるとする
    enum CellType {
        case A
        case B
        case C

        var reuseIdentifier: String {
            switch self {
            case .A: return "A"
            case .B: return "B"
            case .C: return "C"
            }
        }
    }
    ...
}
```

セルの種類が複数あると期待通りの型のセルがdequeueできたか
確認するのもだるいし安全にキャストしていくのもだるい

UITableViewController

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cellType = /* indexPathなどからセルの種類を決める */
    let cell = tableView.dequeueReusableCellWithIdentifier(
        cellType reuseIdentifier, forIndexPath: indexPath)

    switch (cellType, cell) {
    case (.A, let cell as CellA):
        // cellはCellAとして扱える
        cell.cellALabel.text = "Hello!"

    default:
        fatalError("Unexpected cell dequeued.")
    }

    return cell
}
```

期待している種類と実際にdequeueされたセルの型を
同時にマッチングしてあげればそこまでだるくない

まとめ

- パターンの文法は初見では難しいが1度理解すれば簡単
- 複数のパターンの組み合わせで値の性質を表現する
- 標準ライブラリでいくつかexpression patternが用意されている
- アプリ開発でも使いどころはある