#MDCC 2016

Swift 链式语法应用

陈乘方@ENJOY

关于我

- Swift 开发者
- 「ENJOY」iOS 客户端负责人
- 两年 Swift 实际项目开发经验
- 微博 ID: webfrogs
- Twitter: nswebfrog

Writing code is always easy,
the hard part is reading it.

链式语法？

链式语法

可以连续不断地进行方法调用的一种语法形式

Objective-C 链式语法

RAC

```
[[[[client logIn]
    then:^{
        return [client loadCachedMessages];
    }]
    flattenMap:^(NSArray *messages) {
        return [client fetchMessagesAfterMessage:
                                messages.lastObject];
    }]
    subscribeError:^(NSError *error) {
        [self presentError:error];
    } completed:^{
        NSLog(@"Fetched all messages.");
    }];
```

Masonry

```
make.top.equalTo(self.view).with.offset(80);
```

Swift 链式语法

Optional Chaining

```
let dic: [String: String] = ["url": "http://www.apple.com"]
let request = dic["url"]
    .flatMap { URL(string: $0) }
    .map { URLRequest(url: $0) }
```

链式语法优点

- 简洁
- 高复用性
- 高可读性
- 减少中间变量

实现方式

函数的返回值类型与自身类型相同

举例:

```
extension Int {
    func add(_ value: Int) -> Int {
        return self + value
    }
}
```

```
let sum = 3.add(2).add(6)
```

应用实例

- 链式 UI 代码
- 链式网络结果处理

UI 代码

```
let firstLabel = UILabel()
view.addSubview(firstLabel)
firstLabel.snp.makeConstraints { (make) in
    make.top.equalTo(self.view).offset(80)
    make.centerX.equalTo(self.view)
}
firstLabel.backgroundColor = .clear
firstLabel.font = UIFont.systemFont(ofSize: 20)
firstLabel.textColor = .red
firstLabel.text = "firstLabel"

let secondLabel = UILabel()
view.addSubview(secondLabel)
secondLabel.snp.makeConstraints { (make) in
    make.top.equalTo(firstLabel.snp.bottom).offset(20)
    make.centerX.equalTo(self.view)
}
secondLabel.backgroundColor = .clear
secondLabel.font = UIFont.systemFont(ofSize: 20)
secondLabel.textColor = .red
secondLabel.text = "secondLabel"
```

UI 代码

```
let firstLabel = UILabel()
view.addSubview(firstLabel)
firstLabel.snp.makeConstraints { (make) in
    make.top.equalTo(self.view).offset(80)
    make.centerX.equalTo(self.view)
}
firstLabel.backgroundColor = .clear
firstLabel.font = UIFont.systemFont(ofSize: 20)
firstLabel.textColor = .red
firstLabel.text = "firstLabel"

let secondLabel = UILabel()
view.addSubview(secondLabel)
secondLabel.snp.makeConstraints { (make) in
    make.top.equalTo(firstLabel.snp.bottom).offset(20)
    make.centerX.equalTo(self.view)
}
secondLabel.backgroundColor = .clear
secondLabel.font = UIFont.systemFont(ofSize: 20)
secondLabel.textColor = .red
secondLabel.text = "secondLabel"
```

UI 代码

1. 创建 view
2. 添加到父视图
3. 设置约束
4. 配置 view 属性

```
let firstLabel = UILabel()
view.addSubview(firstLabel)
firstLabel.snp.makeConstraints { (make) in
    make.top.equalTo(self.view).offset(80)
    make.centerX.equalTo(self.view)
}
firstLabel.backgroundColor = .clear
firstLabel.font = UIFont.systemFont(ofSize: 20)
firstLabel.textColor = .red
firstLabel.text = "firstLabel"

let secondLabel = UILabel()
view.addSubview(secondLabel)
secondLabel.snp.makeConstraints { (make) in
    make.top.equalTo(firstLabel.snp.bottom).offset(20)
    make.centerX.equalTo(self.view)
}
secondLabel.backgroundColor = .clear
secondLabel.font = UIFont.systemFont(ofSize: 20)
secondLabel.textColor = .red
secondLabel.text = "secondLabel"
```

链式 UI 代码

```
let firstLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(self.view).offset(80)
        make.centerX.equalTo(self.view)
    }
    .ccf_config { (label) in
        label.backgroundColor = .clear
        label.font = UIFont.systemFont(ofSize: 20)
        label.textColor = .red
        label.text = "firstLabel"
    }
let secondLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(firstLabel.snp.bottom).offset(20)
        make.centerX.equalTo(self.view)
    }
    .ccf_config { (label) in
        label.backgroundColor = .clear
        label.font = UIFont.systemFont(ofSize: 20)
        label.textColor = .red
        label.text = "secondLabel"
    }
```

链式 UI 代码

```
let commonLabelConfig: (UILabel) -> Void = { label in
    label.backgroundColor = .clear
    label.font = UIFont.systemFont(ofSize: 20)
    label.textColor = .red
}
let firstLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(self.view).offset(80)
        make.centerX.equalTo(self.view)
    }
    .ccf_config(commonLabelConfig)
    .ccf_config { (label) in
        label.text = "firstLabel"
    }
let secondLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(firstLabel.snp.bottom).offset(20)
        make.centerX.equalTo(self.view)
    }
    .ccf_config(commonLabelConfig)
    .ccf_config { (label) in
        label.text = "secondLabel"
    }
```

链式 UI 代码

```
let commonLabelConfig: (String?) -> ((UILabel) -> Void) = { text in
    return { label in
        label.backgroundColor = .clear
        label.font = UIFont.systemFont(ofSize: 20)
        label.textColor = .red
        label.text = text
    }
}
let firstLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(self.view).offset(80)
        make.centerX.equalTo(self.view)
    }
    .ccf_config(commonLabelConfig("firstLabel"))
let secondLabel = UILabel
    .ccf_create(withSuperView: view)
    .ccf_layout { (make) in
        make.top.equalTo(firstLabel.snp.bottom).offset(20)
        make.centerX.equalTo(self.view)
    }
    .ccf_config(commonLabelConfig("secondLabel"))
```

链式 UI 代码

Demo

Talk is cheap. Show me the code.

Linus Torvalds

应用实例

- 链式 UI 代码
- 链式网络结果处理

网络结果处理

目的：JSON 数据转换为 Model 对象

网络结果处理

```
let jsonData: Any = ["key": "value"]
guard let jsonDic = jsonData as? [String: Any] else {
    // TODO: data format error
    return
}
guard let model = DemoModel(jsonDic: jsonDic) else {
    // TODO: data format error
    return
}
// do something
```

链式网络结果处理

- 封装网络结果
- 抽象数据处理过程
- 提供基础处理组件

链式网络结果处理

```
demoResult
    .flatMap(responseDataToDic)
    .flatMap(responseDicToAPIModel)
    .successHandler { (data: DemoResponseModel) in
        // TODO: success
        print(data)
    }
    .failureHandler { (error) in
        // TODO: failure
        print(error)
    }
```

链式网络结果处理

网络结果封装：

```
enum APIResult<T> {
    case success(T)
    case failure(Error)
}
```

链式网络结果处理

```
extension APIResult {
    func flatMap<U>(_ transform: (T) -> APIResult<U>) -> APIResult<U> {
        let result: APIResult<U>
        switch self {
        case let .success(value):
            result = transform(value)
        case let .failure(error):
            result = .failure(error)
        }
        return result
    }
    func map<U>(_ transform: (T) -> U) -> APIResult<U> {
        let result: APIResult<U>
        switch self {
        case let .success(value):
            result = .success(transform(value))
        case let .failure(error):
            result = .failure(error)
        }
        return result
    }
}
```

链式网络结果处理

```
extension APIResult {
    func successHandler(_ success: (T) -> Void) -> APIResult<T> {
        if case let .success(value) = self {
            success(value)
        }
        return self
    }

    func failureHandler(_ failure: (Error) -> Void) -> APIResult<T> {
        if case let .failure(error) = self {
            failure(error)
        }
        return self
    }
}
```

链式网络结果处理

Demo

Talk is cheap. Show me the code.

Linus Torvalds

命名空间式扩展

- 去除 Objective-C 时代的前缀式扩展方法命名
- 目前几乎所有主流的 Swift 三方框架都已支持

SnapKit：

```
view.snp_makeConstraints { (make) in
}
```

```
view.snp.makeConstraints { (make) in
}
```

命名空间式扩展

```
protocol NamespaceCompatible {
    associatedtype CompatibleType
    var ccf: CompatibleType { get }
    static var ccf: CompatibleType.Type { get }
}

struct Namespace<Base> {
    let base: Base

    init(_ base: Base) {
        self.base = base
    }
}
extension NamespaceCompatible {
    var ccf: Namespace<Self> {
        return Namespace(self)
    }

    static var ccf: Namespace<Self>.Type {
        return Namespace.self
    }
}

extension NSObject: NamespaceCompatible { }
```

命名空间式扩展

- UI 链式代码的适配
- 对 Struct 进行扩展比较麻烦

命名空间式扩展

Demo

Talk is cheap. Show me the code.

Linus Torvalds

总结

- 了解链式代码
- 如何编写自己的链式代码
- 链式代码实例
- Swifty 的命名空间方式扩展

Demo 工程地址：https://github.com/MDCC2016/SwiftChainingDemo

思考

- 如何在实际的编程中应用链式语法
- 链式代码与 RxSwift 结合
- 如何让自己的代码更加 Swifty

扩展资料

- WWDC 2016 session 403: Swift API Design Guidelines
- WWDC 2015 session 408: Protocol-Oriented Programming in Swift
- 读 Swift 源码，理解 Monad

ENJOY Swift

≈

Thanks!