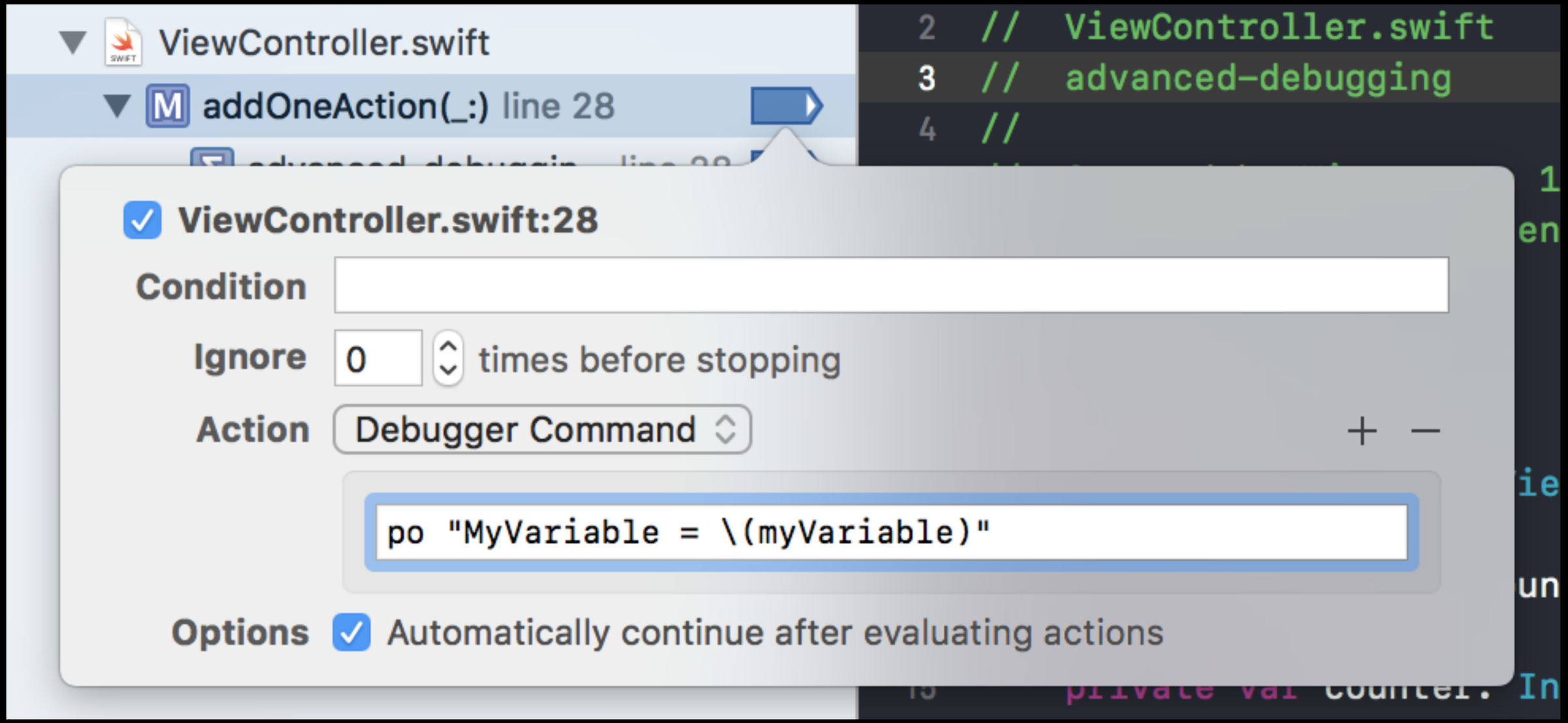
Looking Through The Mirror 🪞

Vincent Pradeilles (@v_pradeilles) – Worldline 🇫🇷

I’m Vincent 👋 🇫🇷

Let’s talk of debugging 


ViewController.swift
addOneAction(_:) line 28
ViewController.swift:28
Condition
Ignore
0
times before stopping
Action
Debugger Command
po "MyVariable = \(myVariable)"
Options
Automatically continue after evaluating actions
2 // ViewController.swift
3 // advanced-debugging
4 //


CAREFULLY CRAFTED
BREAKPOINT
PRINT()
imgflip.com

As iOS developers, we're indeed very familiar with `print()`!

But did you know that Swift also has another very similar function?

Introducing `dump()`!

So what’s the difference between them?

Let’s take a look a some examples!

```
struct User {
    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

print(user)
```

```
struct User {
    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

print(user)
```

```
> User(name: "Vincent", age: 31)
```

```
struct User {
    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

dump(user)
```

```swift
struct User {
    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

dump(user)
```

```
> ▿ User
    - name: "Vincent"
    - age: 31
```

```
print(user)

> User(name: "Vincent", age: 31)

dump(user)

> ▿ User
    - name: "Vincent"
    - age: 31
```

```
print(user)

> User(name: "Vincent", age: 31)

dump(user)

> ▿ User
    - name: "Vincent"
    - age: 31
```

Different formatting,
but otherwise pretty similar content

Now let’s see what happens with classes

```
class User {

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

print(user)
```

```
class User {

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

print(user)

$ User
```

```
class User {

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

dump(user)
```

```
class User {

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    let name: String
    let age: Int
}

let user = User(name: "Vincent", age: 31)

dump(user)

$ ▿ User #0
    - name: "Vincent"
    - age: 31
```

```
print(user)

$ User

dump(user)

$ ▿ User #0
    - name: "Vincent"
    - age: 31
```

```
print(user)

$ User

dump(user)

$ ▿ User #0
      - name: "Vincent"
      - age: 31
```

Ok now that’s an interesting difference!

Why do `print()` and `dump()` behave so differently?

Summary

Dumps the given object's contents using its mirror to standard output.

Declaration

```
@discardableResult func dump<T>(_ value: T, name: String? = nil,
indent: Int = 0, maxDepth: Int = .max, maxItems: Int = .max) -> T
```

Summary

Dumps the given object's contents using its mirror to standard output.

Declaration

```
@discardableResult func dump<T>(_ value: T, name: String? = nil,
indent: Int = 0, maxDepth: Int = .max, maxItems: Int = .max) -> T
```

So what’s this "Mirror"?

Mirrors are Swift mechanism to implement introspection

What’s introspection?

It’s the ability to reflexively interact with our code at runtime

It’s going to be much clearer with an example!

```swift
let user = User(name: "Vincent", age: 31)

let mirror = Mirror(reflecting: user)
```

```
let user = User(name: "Vincent", age: 31)

let mirror = Mirror(reflecting: user)

for child in mirror.children {

}
```

```swift
let user = User(name: "Vincent", age: 31)

let mirror = Mirror(reflecting: user)

for child in mirror.children {
    print("\(child.label): \(child.value)")
}
```

```swift
let user = User(name: "Vincent", age: 31)

let mirror = Mirror(reflecting: user)

for child in mirror.children {
    print("\(child.label): \(child.value)")
}
```

```
$ Optional("name"): Vincent
$ Optional("age"): 31
```

Through a Mirror, we can iterate over the properties of an instance!

So now it’s time to talk about
use cases that leverage this ability

#1
self-loading plugins

```
class AnalyticsSDK {
    func start(with appId: String) {
        print ("Starting Analytics for appId: \(appId)")
    }
}


class CrashReportingSDK {
    func start() {
        print ("Starting Crash Reporting")
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        analytics.start(with: "myAppId")
        crashReporting.start()

        return true
    }
}
```

```
protocol Loadable {
    func load()
}
```

```
protocol Loadable {
    func load()
}
```

```
protocol Loadable {
    func load()
}

extension AnalyticsSDK: Loadable {
    func load() {
        start(with: "myAppId")
    }
}

extension CrashReportingSDK: Loadable {
    func load() {
        start()
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        analytics.start(with: "myAppId")
        crashReporting.start()

        return true
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        analytics.load()
        crashReporting.load()

        return true
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        self.loadPlugins()

        return true
    }

    func loadPlugins() {
        let mirror = Mirror(reflecting: self)

        for child in mirror.children {
            if let loadable = child.value as? Loadable {
                loadable.load()
            }
        }
    }
}
```

```
class OtherSDK {
    func initialize(with apiKey: String) {
        print("Initializing with apiKev: \(apiKey)")
    }
}

extension OtherSDK: Loadable {
    func load() {
        initialize(with: "myApiKey")
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        self.loadPlugins()

        return true
    }

    func loadPlugins() {
        let mirror = Mirror(reflecting: self)

        for child in mirror.children {
            if let loadable = child.value as? Loadable {
                loadable.load()
            }
        }
    }
}
```

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var analytics = AnalyticsSDK()
    var crashReporting = CrashReportingSDK()
    var otherSDK = OtherSDK()

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        self.loadPlugins()

        return true
    }

    func loadPlugins() {
        let mirror = Mirror(reflecting: self)

        for child in mirror.children {
            if let loadable = child.value as? Loadable {
                loadable.load()
            }
        }
    }
}
```

#2
accessing private properties

```
class Service {
    func doService() {
        print("I'm doing a service")
    }
}

class ViewModel {
    private var service: Service

    init(service: Service) {
        self.service = service
    }
}
```

```
@testable import Private

class PrivateTests: XCTestCase {

    func testDependencyInjection() {
        let service = Service()

        let viewModel = ViewModel(service: service)

        XCTAssertIdentical(service, viewModel.service)
    }
}
```

```swift
@testable import Private

class PrivateTests: XCTestCase {

    func testDependencyInjection() {
        let service = Service()

        let viewModel = ViewModel(service: service)

❌      XCTAssertIdentical(service, viewModel.service)
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
extension Mirror {
    func firstChild<T>(named name: String) -> T? {
        children.compactMap {
            guard let value = $0.value as? T else { return nil }

            return $0.label == name ? value : nil
        }.first
    }
}
```

```swift
@testable import Private

class PrivateTests: XCTestCase {

    func testDependencyInjection() {
        let service = Service()

        let viewModel = ViewModel(service: service)

❌      XCTAssertIdentical(service, viewModel.service)
    }
}
```

```
@testable import Private

class PrivateTests: XCTestCase {

    func testDependencyInjection() {
        let service = Service()

        let viewModel = ViewModel(service: service)

        let mirror = Mirror(reflecting: viewModel)

✅      XCTAssertIdentical(service, mirror.firstChild(named: "service"))
    }
}
```

#3
runtime code generation

Wouldn’t that syntax be cool?

```
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

Well let’s implement it!

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```
protocol HTTPRequestable {
    var path: String { get }
}
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
@propertyWrapper
struct PathArgument {
    let name: String
    var value: String = ""

    init(_ name: String) {
        self.name = name
    }

    var wrappedValue: String {
        get { return value }
        set { value = newValue }
    }

    var projectedValue: (name: String, value: String) {
        return (name: name, value: value)
    }
}
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
@propertyWrapper
struct QueryArgument {
    let name: String
    var value: String = ""

    init(_ name: String) {
        self.name = name
    }

    var wrappedValue: String {
        get { return value }
        set { value = newValue }
    }

    var projectedValue: URLQueryItem {
        return URLQueryItem(name: name, value: value)
    }
}
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
extension HTTPRequestable {
    var requestURL: URL {
        let baseUrl = "https://api.myservice.com/"

        let mirror = Mirror(reflecting: self)

        var evaluatedPath = path

        for child in mirror.children {
            if let pathArgument = child.value as? PathArgument {
                evaluatedPath = evaluatedPath.replacingOccurrences(of: "{\(pathArgument.name)}",
                                                                   with: pathArgument.value)
            }
        }

        let fullUrl = baseUrl + evaluatedPath
        var urlComponents = URLComponents(string: fullUrl)!

        var queryItems = [URLQueryItem]()

        for child in mirror.children {
            if let queryArgument = child.value as? QueryArgument {
                queryItems.append(queryArgument.projectedValue)
            }
        }

        urlComponents.queryItems = queryItems

        return urlComponents.url!
    }
}
```

```swift
struct GetUserDataRequest: HTTPRequestable {
    let path: String = "group/{id}/users"
    @PathArgument("id") var groupId: String
    @QueryArgument("sort") var sort: String

    init(groupId: String, sort: String) {
        self.groupId = groupId
        self.sort = sort
    }
}

let userDataRequest = GetUserDataRequest(groupId: "42", sort: "desc")

userDataRequest.requestURL // https://api.myservice.com/group/42/users?sort=desc
```

Time for a recap!

recap

recap

- `Mirror` is Swift's reflection API

recap

- `Mirror` is Swift's reflection API
- Reflection opens up the way for some pretty powerful runtime use cases!

recap

- `Mirror` is Swift's reflection API
- Reflection opens up the way for some pretty powerful runtime use cases!
- There are some limits though: for instance, mirrors are read-only

recap

- `Mirror` is Swift's reflection API
- Reflection opens up the way for some pretty powerful runtime use cases!
- There are some limits though: for instance, mirrors are read-only
- Also, reflection easily leads to confusing and unpredictable code

recap

- `Mirror` is Swift's reflection API
- Reflection opens up the way for some pretty powerful runtime use cases!
- There are some limits though: for instance, mirrors are read-only
- Also, reflection easily leads to confusing and unpredictable code
- Think twice to make sure that it is indeed worth the added complexity!


That's all Folks!

Thank You! 

Twitter

YouTube