PROTOCOL-ORIENTED PROGRAMMING IN NETWORKING

ISHKAWA

ISHKAWA?

Yosuke Ishikawa
ishkawa

Developer Program Member

Mercari
Tokyo, Japan
y@ishkawa.org
http://ishkawa.org
Joined on Aug 9, 2011

317 Followers **321** Starred **44** Following

Organizations

Contributions | Repositories | Public activity | Follow

Popular repositories

- **APIKit** — A networking library for building type safe web… — 525 ★
- **ISRefreshControl** — (deprecated) iOS4-compatible UIRefreshControl — 242 ★
- **ISColumnsController** — paginated container view controller. — 118 ★
- **ISDiskCache** — LRU disk cache for iOS. — 81 ★
- **UINavigationController-Transit…** — extension for custom transition by blocks. — 56 ★

Repositories contributed to

- antitypical/**Result** — Swift type modelling the success/failure of arbi… — 702 ★
- AliSoftware/**OHHTTPStubs** — Stub your network requests easily! Test your a… — 2,038 ★
- CocoaPods/**Specs** — The CocoaPods Master Repo — 2,861 ★
- kishikawakatsumi/**KeychainAcc…** — Simple Swift wrapper for Keychain that works … — 1,315 ★
- matteocrippa/**awesome-swift** — A collaborative list of awesome swift resource… — 6,680 ★

Public contributions

Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan
M W F

Summary of pull requests, issues opened, and commits. Learn how we count contributions. Less More

Contributions in the last year	Longest streak	Current streak
576 total	13 days	0 days
Feb 12, 2015 – Feb 12, 2016	February 12 – February 24	Last contributed 13 days ago

OVERVIEW

PROTOCOL-ORIENTED PROGRAMMING IN NETWORKING

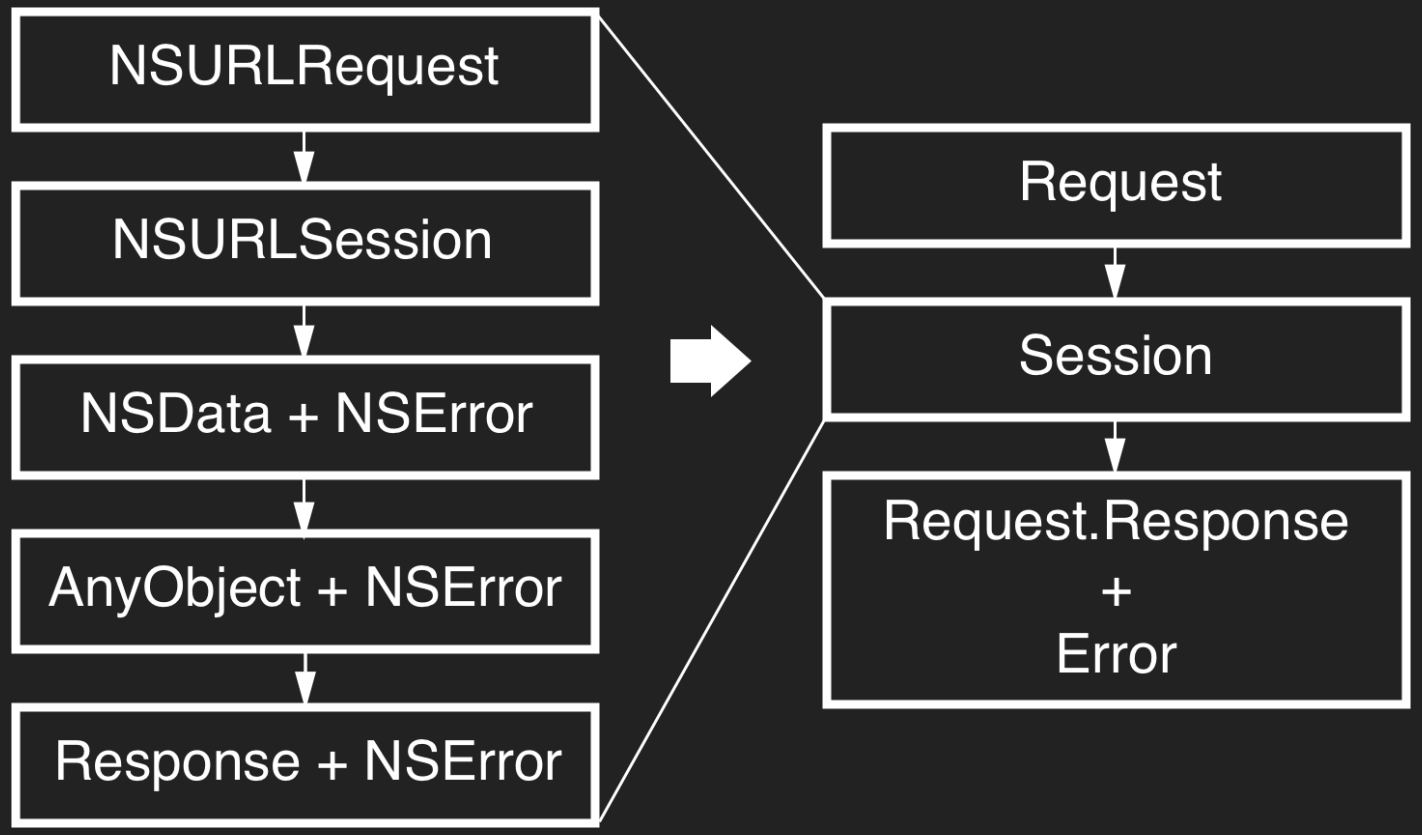
1. Wrapping NSURLSession using protocols
2. Generic programming on the protocols
3. Combination with RxSwift

TOPIC 1

Wrapping NSURLSession using protocols

- To make call-site code simpler
- To get responses type-safely

GOAL


NSURLRequest
NSURLSession
NSData + NSError
AnyObject + NSError
Response + NSError
Request
Session
Request.Response
+
Error

GOAL

```
let request = Request()

Session.sendRequest(request) { result in
    // result: Result<Request.Response, Error>
}
```

GOAL

```
let request = CreateIssueRequest(repositoryID: 123, title: "Test")

Session.sendRequest(request) { result in
    // result: Result<Issue, Error>
}
```

```
let request = SearchRepositoriesRequest(query: "swift")

Session.sendRequest(request) { result in
    // result: Result<PaginationResponse<Repository>, Error>
}
```

The type of result changes depending on the request type

WHY PROTOCOL?

1. To associate response type with request type
2. To provide flexible default implementation

WHY PROTOCOL?

1. To associate response type with request type
2. To provide flexible default implementation

ASSOCIATING RESPONSE TYPE WITH REQUEST TYPE

```
protocol RequestType {
    typealias Response

    var baseURL: NSURL { get }
    var method: HTTPMethod { get }
    var path: String { get }
    var parameters: [String: AnyObject] { get }

    func responseFromObject(
        object: AnyObject,
        URLResponse: NSHTTPURLResponse) throws -> Response
}
```

ASSOCIATING RESPONSE TYPE WITH REQUEST TYPE

```
class Session {
    ...

    func sendRequest<Request: RequestType>(
        reqeust: Request,
        handler: Result<Request.Response, Error> -> Void) {
        do {
            let data: NSData = ...
            let json: AnyObject = ...
            let response = request.responseFromObject(json...)
            handler(.Success(response)) // Request.Response
        } catch {
            handler(.Failure(error))
        }
    }
}
```

ASSOCIATING RESPONSE TYPE WITH REQUEST TYPE

```
let request = CreateIssueRequest(repositoryID: 123, title: "Test")

Session.sendRequest(request) { result in
    // result: Result<Issue, Error>
}
```

```
let request = SearchRepositoriesRequest(query: "swift")

Session.sendRequest(request) { result in
    // result: Result<PaginationResponse<Repository>, Error>
}
```

The type of result changes depending on the request type

WHY PROTOCOL?

1. To associate response type with request type
2. To provide flexible default implementation

PROVIDING DEFAULT IMPLEMENTATION

PART 1: COMMON CONFIGURATIONS

```
protocol GitHubRequestType: RequestType {

}
```

```
extension GitHubRequestType {
    var baseURL: NSURL {
        return NSURL(string: "https://api.github.com")!
    }
}
```

PROVIDING DEFAULT IMPLEMENTATION

PART 2: PROTOCOL FOR JSON DECODING

```
struct SearchRepositoriesRequest: GitHubRequestType {
    func responseFromObject(
        object: AnyObject,
        URLResponse: NSHTTPURLResponse) throws -> Response {

        guard let dictionaries = object as? [[String: AnyObject]] else {
            throw InvalidObject(object)
        }

        let repositories = dictionaries.map { try Repository($0) }
        let hasNextPage = ...

        return PaginationResponse(
            elements: repositories,
            hasNextPage: hasNextPage)
    }
```

PROVIDING DEFAULT IMPLEMENTATION

PART 2: PROTOCOL FOR JSON DECODING

```
protocol Decodable {
    static func decode(object: AnyObject) throws -> Self
}
```

PROVIDING DEFAULT IMPLEMENTATION

PART 2: PROTOCOL FOR JSON DECODING

```
extension RequestType where Response: Decodable {
    func responseFromObject(
        object: AnyObject,
        URLResponse: NSHTTPURLResponse) throws -> Response {

        return Response.decode(object)
    }
}
```

PROVIDING DEFAULT IMPLEMENTATION

PART 2: PROTOCOL FOR JSON DECODING

```
struct SearchRepositoriesRequest: GitHubRequestType {
    let query: String

    // MARK: RequestType
    typealias Response = PaginationResponse<Repository>

    var method: HTTPMethod     { return .GET }
    var path: String           { return "/search/repositories" }
    var parameters: AnyObject { return ["q": query] }
}
```

WHY PROTOCOL?

1. To associate response type with request type
    - Simpler and safer call-site
2. To provide flexible default implementation
    - Documentation-like request definition

LINKS

- APIKit: https://github.com/ishkawa/APIKit
  - RequestType, Session, etc.
- Himotoki: https://github.com/ikesyo/Himotoki
  - Decodable, etc.

TOPIC 2

Practical example of generic programming on protocols

- Example: Pagination

PAGINATION REQUEST

```
curl -v https://api.github.com/search/repositories?q=swift&page=1
```

PAGINATION REQUEST

```
protocol PaginationRequestType: RequestType {
    typealias Response: PaginationResponseType

    var page: Int { get }

    func requestWithPage(page: Int) -> Self
}
```

PAGINATION RESPONSE

```
HTTP/1.1 200 OK
Server: GitHub.com
Content-Type: application/json; charset=utf-8
Link: <https://api.github.com/...>; rel="next"

{
    "total_count": 38262,
    "incomplete_results": false,
    "items": [
        ...
    ]
}
```

PAGINATION RESPONSE

```
protocol PaginationResponseType {
    typealias Element: Decodable

    var elements: [Element] { get }
    var hasNextPage: Bool { get }
}
```

PAGINATION CLIENT

```
class PaginationClient<Request: PaginationRequestType> {
    let baseRequest: Request
    let updateHandler: Void -> Void

    init(baseRequest: Request, updateHandler: Void -> Void) { ... }

    var elements: [Request.Response.Element]
    var hasNextPage: Bool
    var page: Int

    func refresh() { ... }
    func loadNextPage() { ... }
}
```

PAGINATION CLIENT

```
func refresh() {
    let request = baseRequest.requestWithPage(1)

    Session.sendRequest(request) { result in
        switch result {
        case .Success(let response):
            self.page = page
            self.elements = response.elements
            self.hasNextPage = response.hasNextPage
            self.updateHandler()

        case .Failure(let error):
            // handle error
        }
    }
}
```

GENERIC PROGRAMMING ON PROTOCOLS

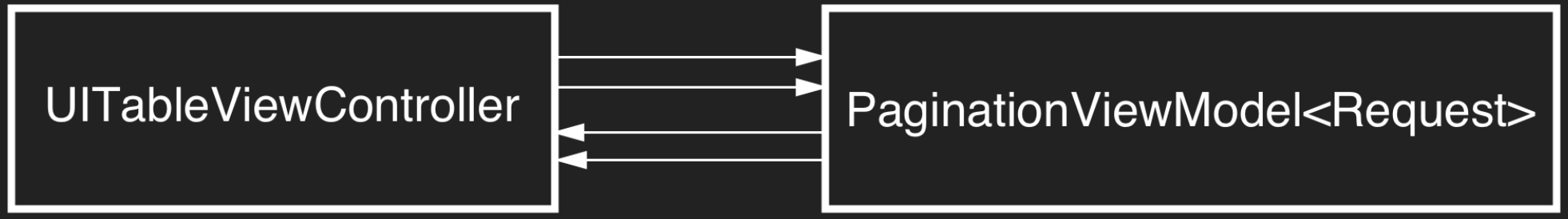
More type constraints,
more detailed implementation

TOPIC 3

Advanced example with reactive streams

RXPAGINATION

Refresh: Observable<Void>
NextPage: Observable<Void>



Elements: Observable<Request.Response>
Loading: Observable<Bool>
HasNextPage: Observable<Bool>

PAGINATION VM WITH RXSWIFT

```
class PaginationViewModel<T: PaginationRequestType> {
    init(baseRequest: T) { ... }

    // Input
    let refreshTrigger: PublishSubject<Void>
    let nextPageTrigger: PublishSubject<Void>

    // Output
    let elements: Observable<[Request.Response.Element]>
    let hasNextPage: Observable<Bool>
    let loading: Observable<Bool>
}
```

LIVE!

STEPS TO IMPLEMENT PAGINATION

- Give a base request to ViewModel
- Bind input and output streams

CONCLUSION

PROTOCOL-ORIENTED IN NETWORKING

1. Protocol is a good choice for abstraction of networking
2. More type constraints, more detailed implementation
3. Abstraction of event stream is also nice

TRY! POP