Нативная сериализация в iOS

Дмитрий Иванов

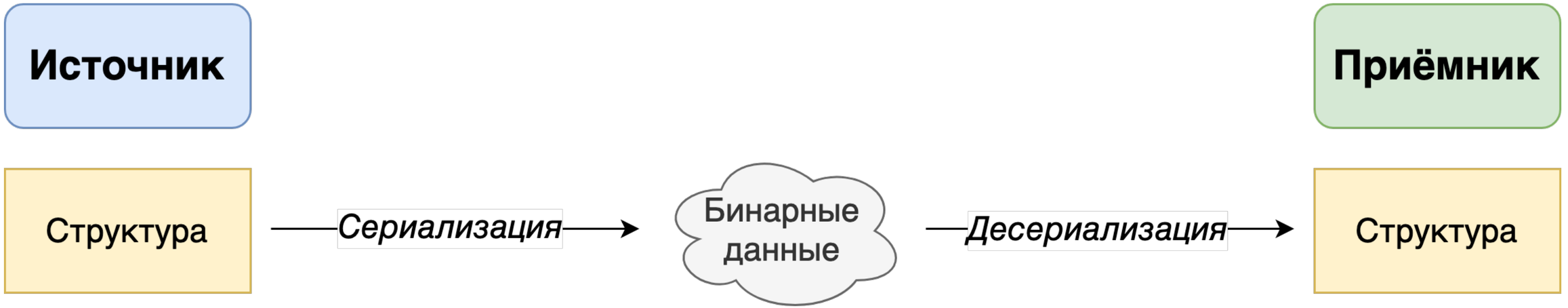
План

1. Суть явления
2. Передача и сохранение Foundation-моделей
3. Передача и сохранение кастомных моделей
4. NSCoding, NSCoder, NSKeyedArchiver
5. Codable
6. Смешаная сериализация
7. Сериализация энумов
8. Сериализация графа объектов
9. Сохранение состояния UI
10. NSCoding vs. Codable

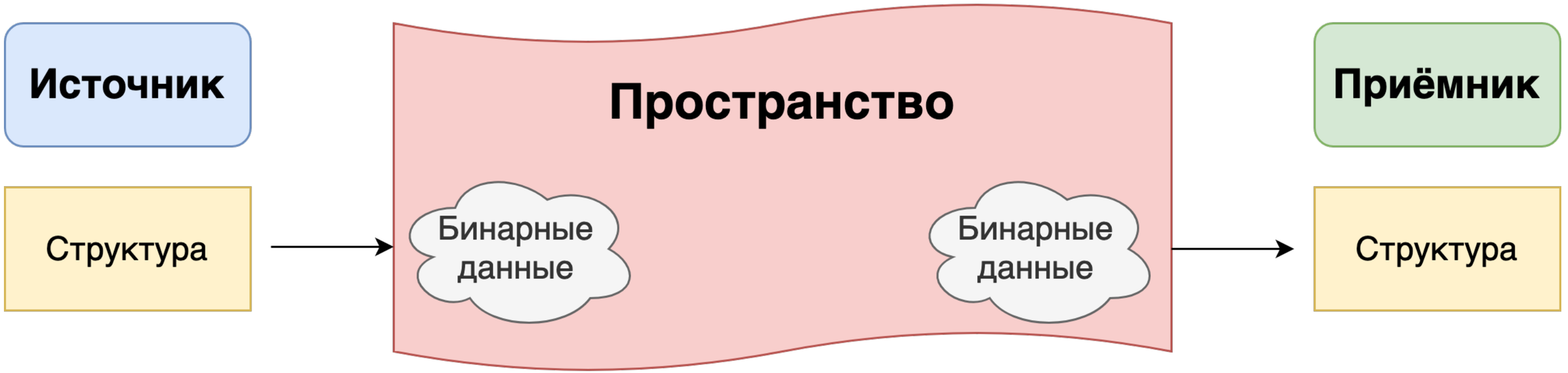
1. Суть явления

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов.

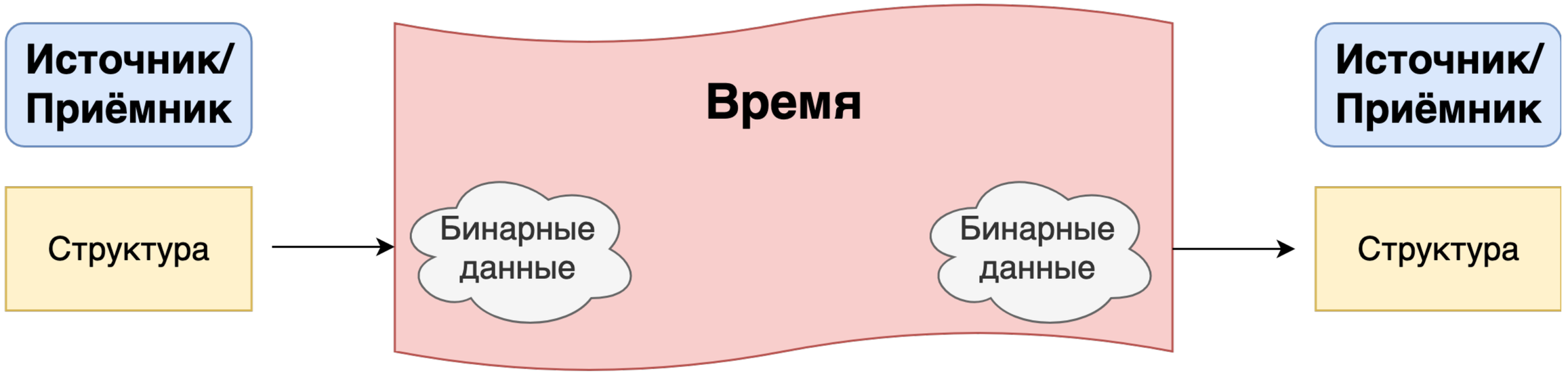
Десериализация — восстановление начального состояния структуры данных из битовой последовательности.


Источник
Приёмник
Структура
Сериализация
Бинарные данные
Десериализация
Структура

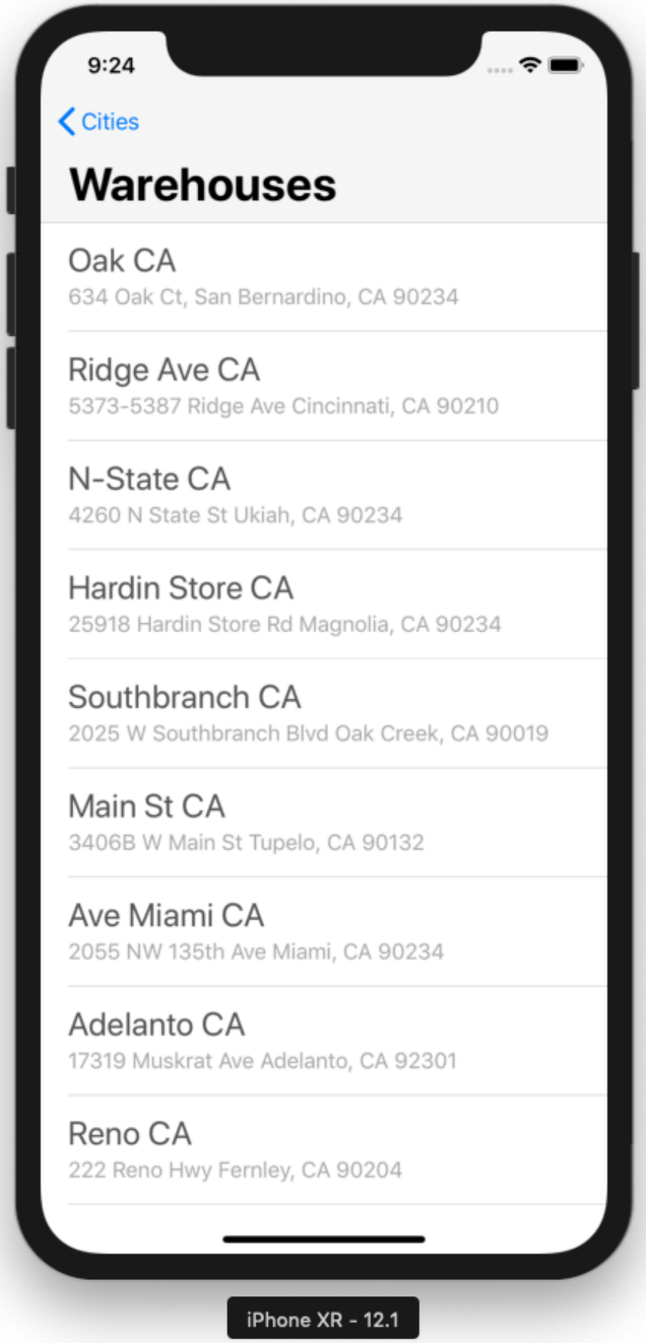
Передача данных



Сохранение данных



2. Передача и сохранение Foundation-моделей



ASN.1 **Bond** Bale **BSON** CBOR
CSV Colfer **FlatBuffers** Feather
HOCON **JSON** MNIP **MessagePack**
Plist OGDL **Protobuf** Pickle
SCaViS **SOAP** Smile **Thrift**
UBJSON VPack **XML** XDR **YAML**

JSON

NSJSONSerialization *(ObjC)*
JSONSerialization *(Swift)*

Ограничение Foundation-модели

Ограничение Foundation-модели

- Корневой объект: NSDictionary/NSArray (сериализация)

Ограничение Foundation-модели

- Корневой объект: NSDictionary/NSArray (сериализация)
- Разрешённые типы: NSString, NSNumber, NSArray, NSDictionary, NSNull

Ограничение Foundation-модели

- Корневой объект: NSDictionary/NSArray (сериализация)
- Разрешённые типы: NSString, NSNumber, NSArray, NSDictionary, NSNull
- Ключи в NSDictionary только NSString

Ограничение Foundation-модели

- Корневой объект: NSDictionary/NSArray (сериализация)
- Разрешённые типы: NSString, NSNumber, NSArray, NSDictionary, NSNull
- Ключи в NSDictionary только NSString
- Числа не должны быть NaN или бесконечность

```
// parsing
NSArray *models = [NSJSONSerialization JSONObjectWithData:jsonData
                                                  options:0
                                                    error:nil];

// cellForRowAtIndexPath
NSDictionary *model = models[indexPath.row];
cell.textLabel?.text = model[@"name"]
cell.detailTextLabel?.text = model[@"address"]
```

```objc
// saving raw JSON data
NSData *jsonData = ... ;
[jsonData writeToURL:url atomically:YES];

// saving parsed JSON data
NSArray *models = [NSJSONSerialization JSONObjectWithData:jsonData
                                                  options:0
                                                    error:nil];
NSPropertyListFormat format = NSPropertyListBinaryFormat_v1_0;
NSData *modelsToSave = [NSPropertyListSerialization dataWithPropertyList:models
                                                                  format:format
                                                                 options:0
                                                                   error:nil];
[modelsToSave writeToURL:url atomically:YES];

// saving NSArray
[models writeToURL:url atomically:YES];
```

```objc
// saving raw JSON data
NSData *jsonData = ... ;
[jsonData writeToURL:url atomically:YES];

// saving parsed JSON data
NSArray *models = [NSJSONSerialization JSONObjectWithData:jsonData
                                                  options:0
                                                    error:nil];
NSPropertyListFormat format = NSPropertyListBinaryFormat_v1_0;
NSData *modelsToSave = [NSPropertyListSerialization dataWithPropertyList:models
                                                                  format:format
                                                                 options:0
                                                                   error:nil];
[modelsToSave writeToURL:url atomically:YES];

// saving NSArray
[models writeToURL:url atomically:YES];
```

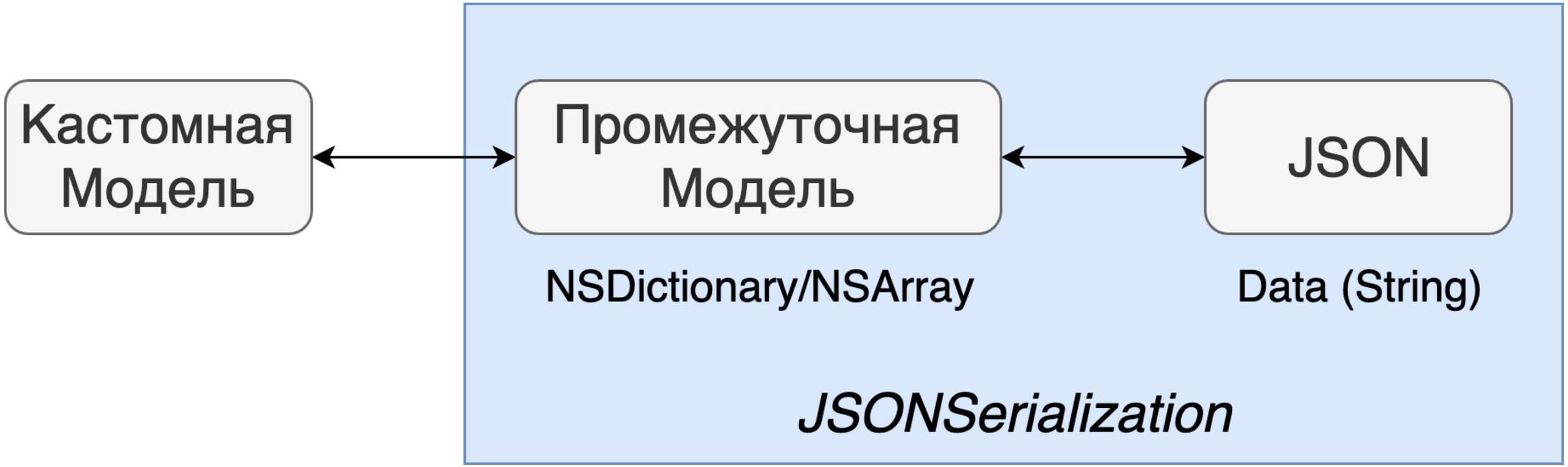
```objc
// saving raw JSON data
NSData *jsonData = ... ;
[jsonData writeToURL:url atomically:YES];

// saving parsed JSON data
NSArray *models = [NSJSONSerialization JSONObjectWithData:jsonData
                                                  options:0
                                                    error:nil];
NSPropertyListFormat format = NSPropertyListBinaryFormat_v1_0;
NSData *modelsToSave = [NSPropertyListSerialization dataWithPropertyList:models
                                                                  format:format
                                                                 options:0
                                                                   error:nil];
[modelsToSave writeToURL:url atomically:YES];

// saving NSArray
[models writeToURL:url atomically:YES];
```

```objc
// saving raw JSON data
NSData *jsonData = ... ;
[jsonData writeToURL:url atomically:YES];

// saving parsed JSON data
NSArray *models = [NSJSONSerialization JSONObjectWithData:jsonData
                                                  options:0
                                                    error:nil];
NSPropertyListFormat format = NSPropertyListBinaryFormat_v1_0;
NSData *modelsToSave = [NSPropertyListSerialization dataWithPropertyList:models
                                                                  format:format
                                                                 options:0
                                                                   error:nil];
[modelsToSave writeToURL:url atomically:YES];

// saving NSArray
[models writeToURL:url atomically:YES];
```

```
// NSDictionary, NSArray, NSString, NSData (iOS 2+)
- (BOOL)writeToURL:(NSURL *)url
        atomically:(BOOL)atomically;

// NSData (iOS 2+)
- (BOOL)writeToURL:(NSURL *)url
           options:(NSDataWritingOptions)writeOptionsMask
             error:(NSError * _Nullable *)errorPtr;

// NSDictionary, NSArray (iOS 11+)
- (BOOL)writeToURL:(NSURL *)url
             error:(NSError * _Nullable *)error;
```

3. Передача и сохранение кастомной модели

```
// Foundation-model
NSDictionary *warehouse = @{
                            @"name" : @"Oak CA",
                            @"address" : @"634 Oak Ct, San Bernardino, CA 92410",
                            @"area" : @(3578)
                            };

// custom model
@interface Warehouse : NSObject

@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *address;
@property (nonatomic, assign) NSInteger *area;

@end
```


Кастомная Модель
Промежуточная Модель
JSON
NSDictionary/NSArray
Data (String)
JSONSerialization

Создание промежуточной модели:

Ручной мапинг

```
- (NSDictionary *)jsonDict {
    return @{
        @"name" : self.name,
        @"address" : self.address,
        @"area" : @(self.area)
    };
}
```

Создание промежуточной модели:

Ручной мапинг

Плюсы:

- просто и понятно

Создание промежуточной модели:

Ручной мапинг

Плюсы:

- просто и понятно

Минусы:

- ручная работа со строками (нет проверки компилятором)
- однотипный код

Создание промежуточной модели:

Автоматический мапинг

```
#import <objc/runtime.h>
...
unsigned int outCount, i;
objc_property_t *properties = class_copyPropertyList([self class], &outCount);
for(i = 0; i < outCount; i++) {
    objc_property_t property = properties[i];
    fprintf(stdout, "%s %s\n", property_getName(property),
    property_getAttributes(property));
}
free(properties);
```

Создание промежуточной модели:

Автоматический мапинг

Плюсы:

- автогенерация промежуточной модели

Создание промежуточной модели:

Автоматический мапинг

Плюсы:

- автогенерация промежуточной модели

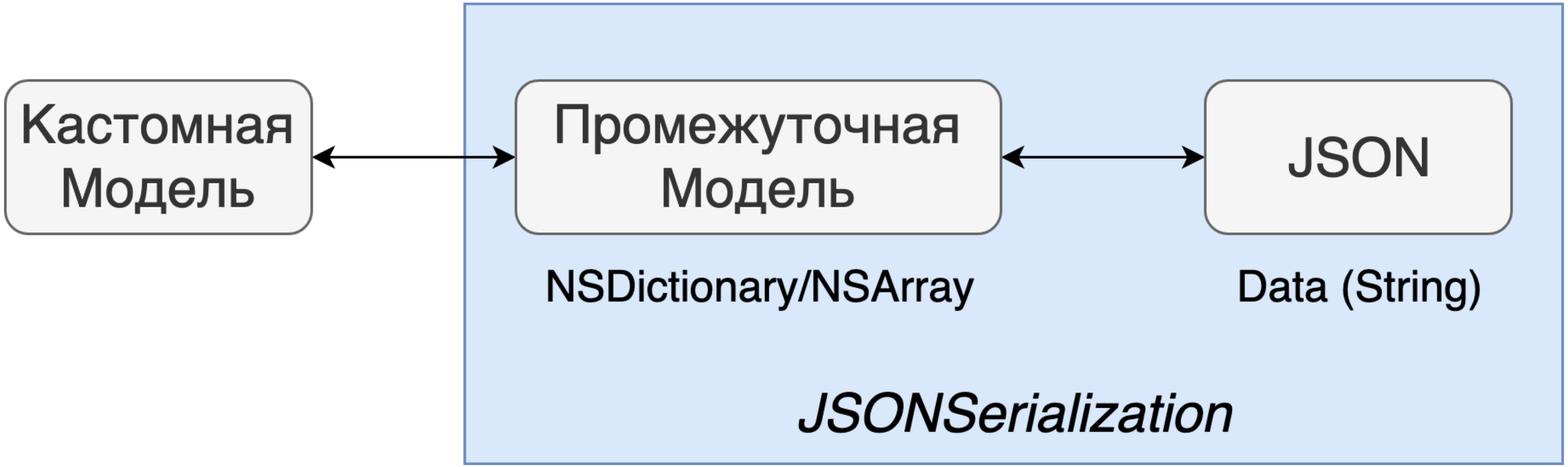
Минусы:

- нужно написать немало кода или использовать стороннее решение

Mantle: *https://github.com/Mantle/Mantle*
JSONModel: *https://github.com/JSONModel/JSONModel*

```
// JSONModel usage

@interface Warehouse : JSONModel

@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *address;
@property (nonatomic, assign) NSInteger *area;

@end

// usage
Warehouse *warehouse = [[Warehouse alloc] initWithString:jsonString error:&error];
```

4. NSCoding, NSCoder, NSKeyedArchiver


Кастомная Модель
Промежуточная Модель
JSON
NSDictionary/NSArray
Data (String)
JSONSerialization

NSCoding

```
public protocol NSCoding {
    func encode(with aCoder: NSCoder)
    init?(coder aDecoder: NSCoder) // NS_DESIGNATED_INITIALIZER
}
```

```
public protocol NSCoding {
    func encode(with aCoder: NSCoder)
    init?(coder aDecoder: NSCoder) // NS_DESIGNATED_INITIALIZER
}

open class NSCoder : NSObject {
    func encode(_ data: Data)
    func decodeData() -> Data?
    // ...
    func encode(_ value: Int32, forKey key: String)
    func encode(CGRect, forKey: String)
    // ...
    func decodeUIEdgeInsets(forKey: String) -> UIEdgeInsets
    func decodeTimeMapping(forKey: String) -> CMTimeMapping
}
```

```
open class NSKeyedArchiver : NSCoder {}

open class NSKeyedUnarchiver : NSCoder {}

let data = try NSKeyedArchiver.archivedData(
    withRootObject: model,
    requiringSecureCoding: false
)

try? data.write(to: fileUrl)

UserDefaults.standard.set(data, forKey: "YourKey")
```

```
open class NSKeyedArchiver : NSCoder {}

open class NSKeyedUnarchiver : NSCoder {}

let data = try NSKeyedArchiver.archivedData(
    withRootObject: model,
    requiringSecureCoding: false
)

try? data.write(to: fileUrl)

UserDefaults.standard.set(data, forKey: "YourKey")
```

```
let archiever = NSKeyedArchiver(requiringSecureCoding: false)
archiever.outputFormat = .xml
archiever.encodeRootObject(model)
let data = archiever.encodedData
```

```
class SimpleClass: NSObject, NSCoding {
    var simpleProperty = "test property"

    func encode(with aCoder: NSCoder) {
        aCoder.encode(
            simpleProperty, forKey: "simpleProperty")
    }

    required init?(coder aDecoder: NSCoder) {
        self.simpleProperty = aDecoder.decodeObject(
            forKey: "simpleProperty") as! String
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>simpleProperty</key>
    <string>test property</string>
</dict>
</plist>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>$archiver</key>
    <string>NSKeyedArchiver</string>
    <key>$objects</key>
    <array>
        <string>$null</string>
        <dict>
            <key>$class</key>
            <dict>
                <key>CF$UID</key>
                <integer>3</integer>
            </dict>
            <key>simpleProperty</key>
            <dict>
                <key>CF$UID</key>
                <integer>2</integer>
            </dict>
        </dict>
        <string>test property</string>
        <dict>
            <key>$classes</key>
            <array>
                <string>Test.SimpleClass</string>
                <string>NSObject</string>
            </array>
            <key>$classname</key>
            <string>Test.SimpleClass</string>
        </dict>
    </array>
    <key>$top</key>
    <dict>
        <key>$0</key>
        <dict>
            <key>CF$UID</key>
            <integer>1</integer>
        </dict>
    </dict>
    <key>$version</key>
    <integer>100000</integer>
</dict>
</plist>
```

Сравнительные размеры (байты)

JSON	Plist (bin)	Plist (xml)	Archiver (bin)	Archiver (xml)
34	75	247	269	904

JSONSerialization

PropertyListSerialization

NSCoding

NSKeyedArchiver

Проблемы при использовании в Swift:

- **NSCoding:** Нет поддержки для `struct` и `enum`
- **JSON/Plist:** отсутствие маппинга в промежуточную модель
- Отсутствие строгой типизации
- Необходимость писать много однотипного кода

5. Codable

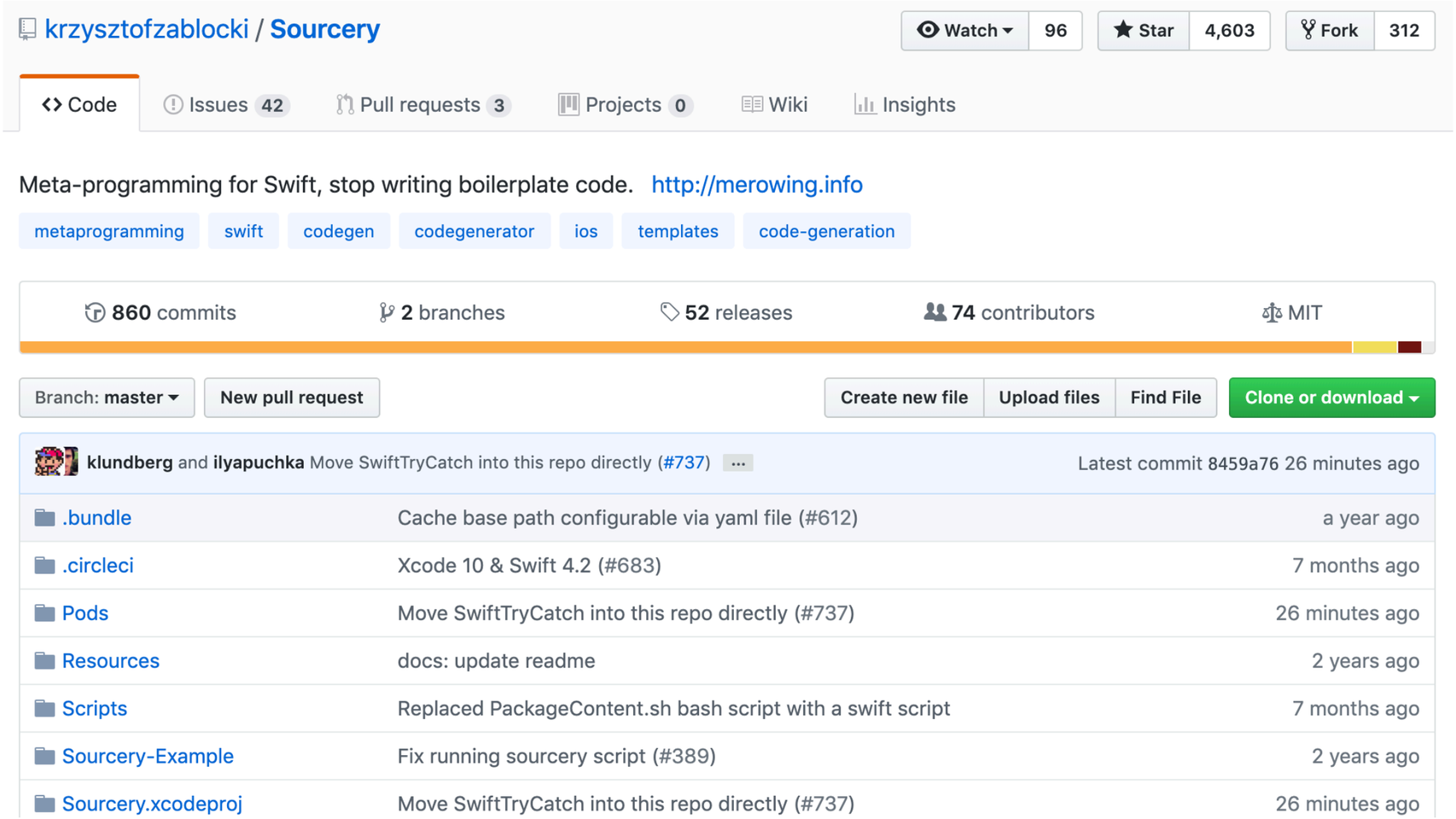
```
typealias Codable = Decodable & Encodable

public protocol Decodable {
    init(from decoder: Decoder) throws
}

public protocol Encodable {
    func encode(to encoder: Encoder) throws
}
```

```
typealias Codable = Decodable & Encodable

public protocol Decodable {
    init(from decoder: Decoder) throws
}

public protocol Encodable {
    func encode(to encoder: Encoder) throws
}
```

```
struct Image: Codable {
    let url: String
    let size: CGSize
}

struct Profile: Codable {
    let name: String
    let profileImage: Image
}
```

```
struct Coordinate: Codable {
    var latitude: Double
    var longitude: Double

    enum CodingKeys: String, CodingKey {
        case latitude
        case longitude
    }

    init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        latitude = try values.decode(Double.self, forKey: .latitude)
        longitude = try values.decode(Double.self, forKey: .longitude)
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(latitude, forKey: .latitude)
        try container.encode(longitude, forKey: .longitude)
    }
}
```

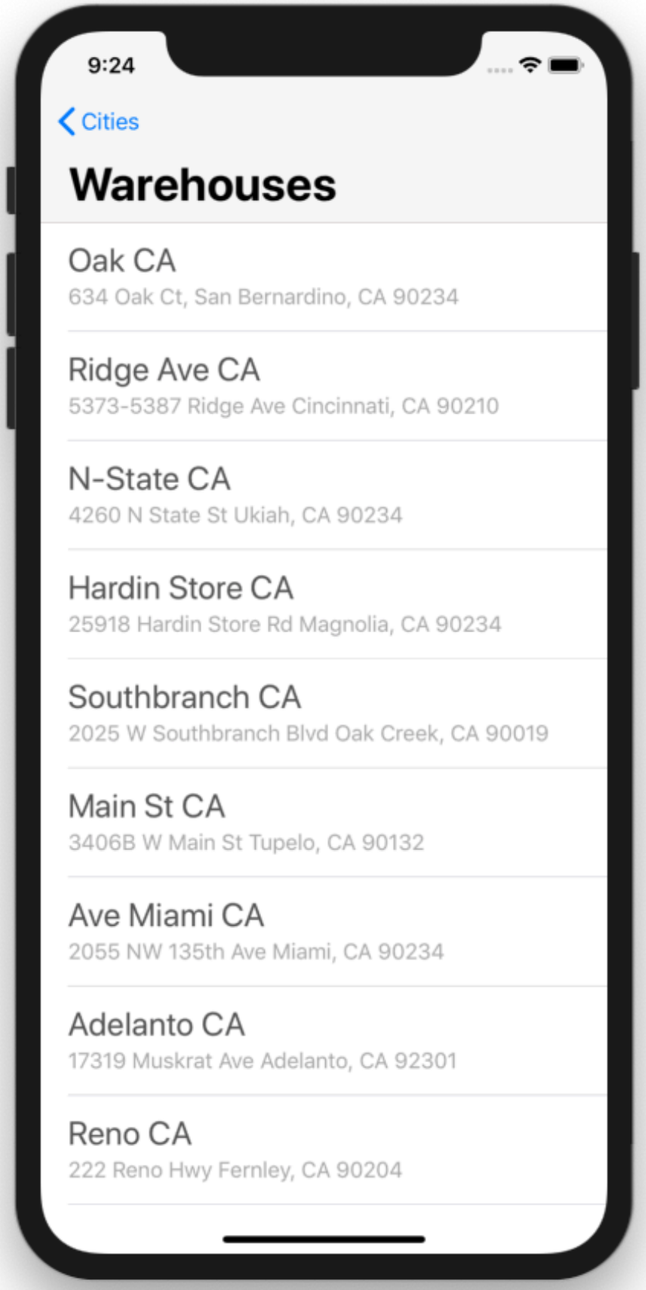
https://github.com/krzysztofzablocki/Sourcery



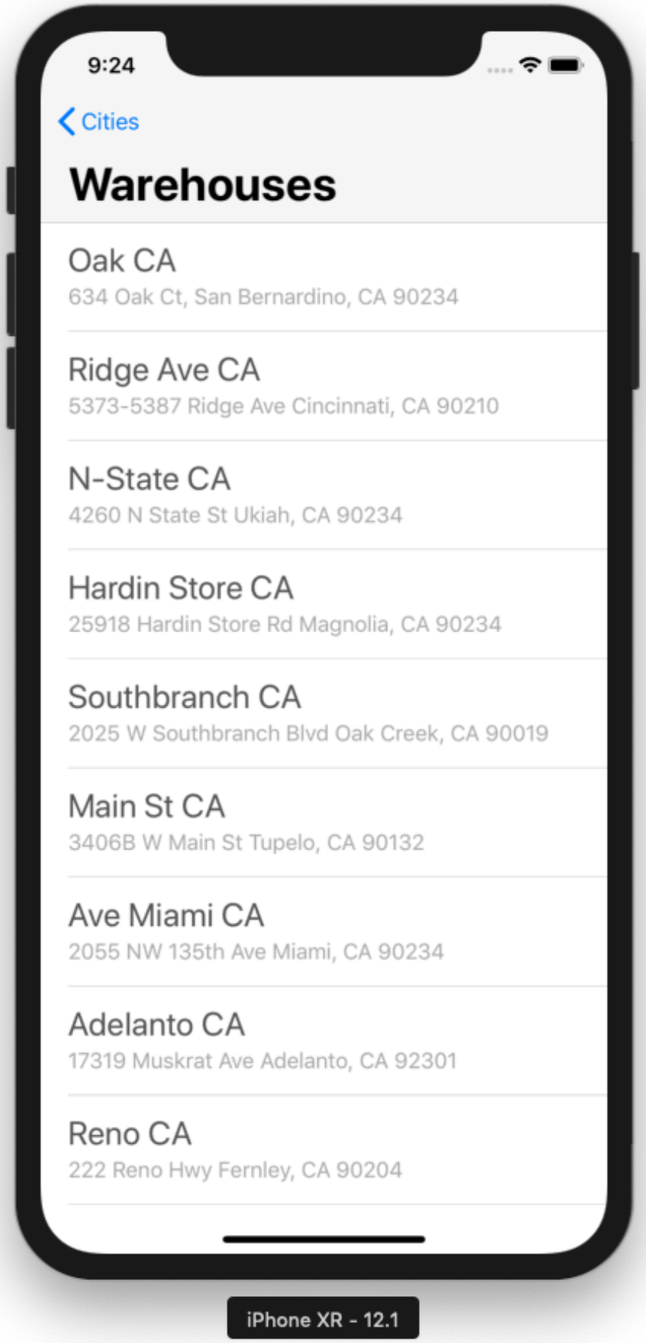
```
public protocol Decodable {}
public protocol Encodable {}
public protocol Encoder {}
public protocol CodingKey:
    CustomDebugStringConvertible,
    CustomStringConvertible {}
public protocol KeyedEncodingContainerProtocol {}
public protocol KeyedDecodingContainerProtocol {}
public protocol UnkeyedEncodingContainer {}
public protocol UnkeyedDecodingContainer {}
public protocol SingleValueEncodingContainer {}
public protocol SingleValueDecodingContainer {}
```

JSONEncoder

PropertyListEncoder



iPhone XR - 12.1



JSON -> binary (MessagePack)

```
class CustomDecoder: Decoder {
    // implementation
}

class CustomEncoder: Encoder {
    // implementation
}
```

Posted at 2017-07-28 12:44 | RSS feed (Full text feed) | Blog Index
Next article: Friday Q&A 2017-08-11: Swift.Unmanaged
Previous article: Friday Q&A 2017-07-14: Swift.Codable
Tags: fridayqna serialization swift

Friday Q&A 2017-07-28: A Binary Coder for Swift

by Mike Ash

This article is also available in Hungarian (translation by Zsolt Boros).

In my last article I discussed the basics of Swift's new `Codable` protocol, briefly discussed how to implement your own encoder and decoder, and promised another article about a custom binary coder I've been working on. Today, I'm going to present that binary coder.

Source Code

As usual, the source code is available on GitHub:

https://github.com/mikeash/BinaryCoder/tree/887cecd70c070d86f338065f59ed027c13952c83

Concept and Approach

This coder serializes fields by writing them out sequentially as raw bytes, with no metadata. For example:

Flight-School / **Codable-DIY-Kit**

Watch 1 | Star 173 | Fork 6

Code | Issues 1 | Pull requests 0 | Wiki | Insights

A template for creating your own Swift Codable encoders and decoders https://flight.school/books/codable

swift codable encoder decoder

4 commits | **1** branch | **0** releases | **2** contributors | MIT

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find File** | **Clone or download**

mattt Update README.md — Latest commit `add7b9e` on 31 Jan

Sources	Initial commit	a year ago
Tests	Initial commit	a year ago
.gitignore	Initial commit	a year ago
LICENSE.md	Initial commit	a year ago
Package.swift	Add 'path' property to Swift 4 manifest.	6 months ago
README.md	Update README.md	3 months ago

README.md

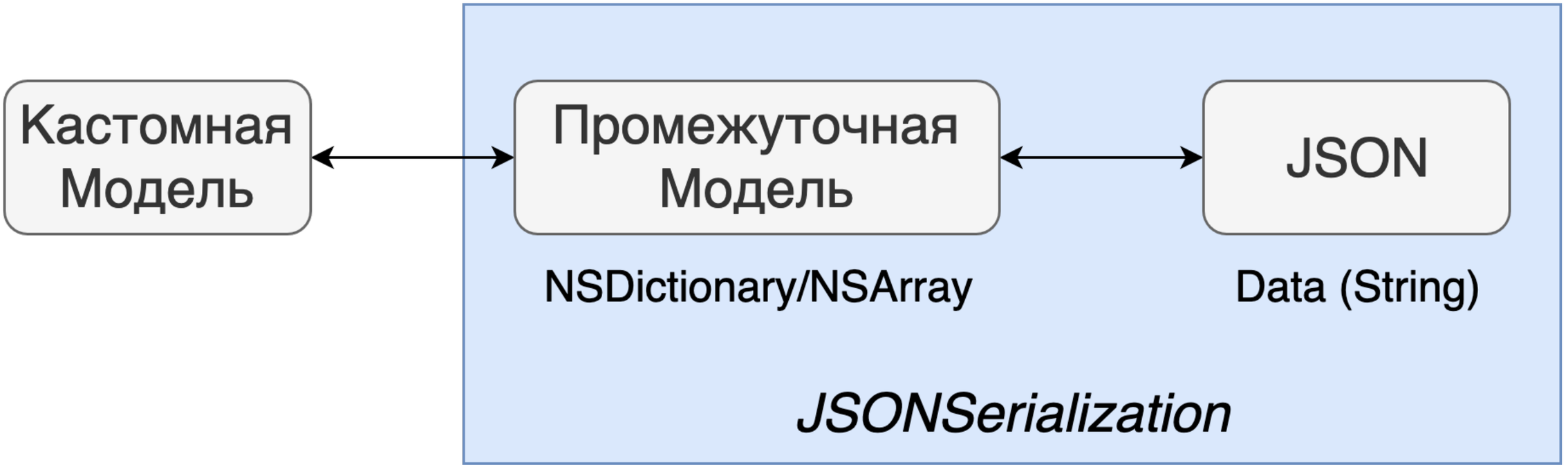
DIY Codable Encoder / Decoder Kit

Codable - native swift API

Codable - native swift API

…or not?

```
func box(_ value: UInt64) -> NSObject { return NSNumber(value: value) }
func box(_ value: String) -> NSObject { return NSString(string: value) }

func box(_ value: Encodable) throws -> NSObject {
    return try self.box_(value) ?? NSDictionary()
}

func pushKeyedContainer() -> NSMutableDictionary { ... }
func pushUnkeyedContainer() -> NSMutableArray { ... }

func decode<T : Decodable>(_ type: T.Type, from data: Data) throws -> T {
    let topLevel: Any
    do {
        topLevel = try JSONSerialization.jsonObject(with: data)
    } catch {
            // ...
    }
    // ...
}
```




```
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    try container.encode(latitude, forKey: .latitude)
    try container.encode(longitude, forKey: .longitude)
}
```

```
struct Warehouse: Codable {
    // ...
}

let jsonData = try? JSONEncoder().encode(warehouse)

let plistData = try? PropertyListEncoder().encode(warehouse)

let customFormatData = try? SomeCustomEncoder().encode(warehouse)
```

Objective-C	Swift
NSCoding (NSKeyedArchiver) JSONSerialization PropertyListSerializaion	Codable

6. Смешаная сериализация

```
class State: NSObject, NSCoding {
    let code: String
    let customersAmount: Int

    // new property
    let warehouses: [Warehouse]
}

struct Warehouse: Codable {
    let name: String
    let address: String
    let area: Int // in square feet
}
```

```
let currentState: State = ...

// Codable?
let data = try? PropertyListEncoder().encode(currentState)

// NSCoding?
let data = try NSKeyedArchiver.archivedData(
            withRootObject: currentState,
            requiringSecureCoding: false
            )
```

```
class State: NSObject, NSCoding {
    let code: String
    let customersAmount: Int
    let warehouses: [Warehouse]

    func encode(with aCoder: NSCoder) {
        aCoder.encode(code, forKey: "code")
        aCoder.encode(customersAmount, forKey: "customersAmount")
        let warehousesData = try! PropertyListEncoder().encode(warehouses)
        aCoder.encode(warehousesData, forKey: "warehouses")
    }
}
```

```
struct Warehouse: Codable {
    let address: String
    let manager: Person // legacy ObjC class

    enum CodingKeys: String, CodingKey {
        case address
        case manager
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(address, forKey: . address)
        let managerData = try NSKeyedArchiver.archivedData(
            withRootObject: manager,
            requiringSecureCoding: false
        )
        try container.encode(managerData, forKey: .manager)
    }
}
```

```
class Wrapper<T: NSCoding>: Codable {
    let value: T
    // ...

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        let valueData = try NSKeyedArchiver.archivedData(
            withRootObject: value,
            requiringSecureCoding: false
        )
        try container.encode(valueData, forKey: .value)
    }

    required init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        let valueData = try values.decode(Data.self, forKey: .value)
        value = NSKeyedUnarchiver.unarchiveObject(with: valueData) as! T
    }
}
```

```
// Obj-C
@interface Person: NSObject<NSCoding>
// ...
@end

// Swift
class Wrapper<T: NSCoding>: Codable {
    // ...
}

let person = Person()
let data = try! JSONEncoder().encode(Wrapper(value: person))

let wrapper = try! JSONDecoder().decode(Wrapper<Person>.self,
                        from: data)
let unarchivedPerson = wrapper.value
```

7. Сериализация энумов

```
enum StateCode: String, Codable {
    case alabama    = "AL"
    case alaska     = "AK"
    case arizona    = "AZ"
    // ...
}

// serializing one state
let topSallingState: StateCode
do {
    let encoder = PropertyListEncoder()
    let data = try encoder.encode(topSallingState)
    try data.write(to: fileUrl)
} catch {
    print(error)
}
```

```
enum StateCode: String, Codable {
    case alabama    = "AL"
    case alaska     = "AK"
    case arizona    = "AZ"
    // ...
}

// serializing one state
let topSallingState: StateCode
do {
    let encoder = PropertyListEncoder()
    let data = try encoder.encode(topSallingState)
    try data.write(to: fileUrl)
} catch {
    print(error)
}
```

Top-level StateCode encoded as string property list fragment.

JSONSerialisation / PropertyListSerialization требуют контейнеризации

Контейнеризация примитивного типа:

Контейнеризация примитивного типа:

- `[value]`, `["key" : value]`

Контейнеризация примитивного типа:

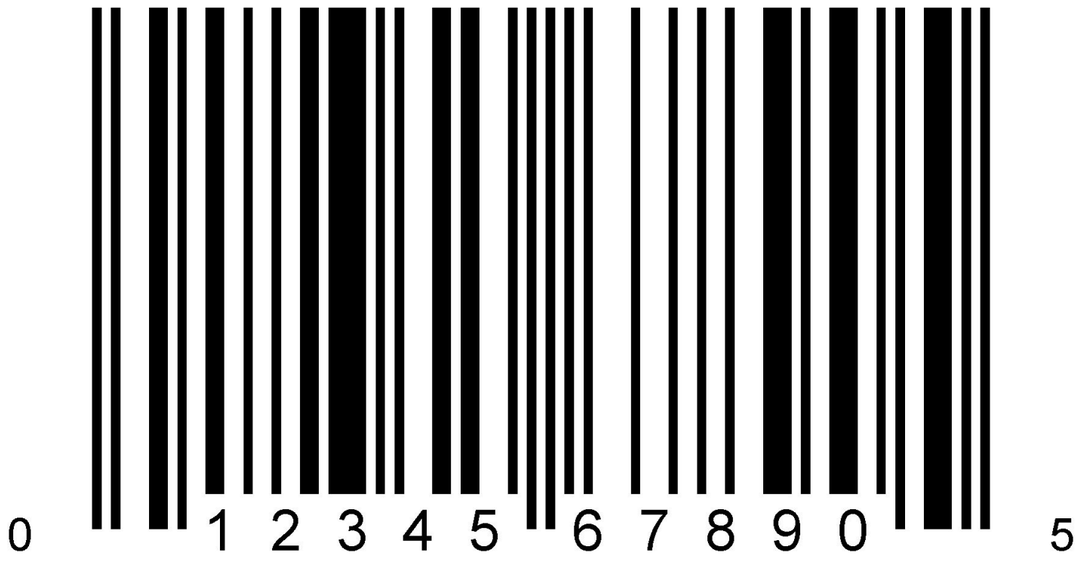
- `[value]`, `["key" : value]`
- Свойство другого объекта

Контейнеризация примитивного типа:

- `[value]`, `["key" : value]`
- Свойство другого объекта
- Реализация методов Codable

Контейнеризация примитивного типа:

- `[value]`, `["key" : value]`
- Свойство другого объекта
- Реализация методов Codable
- NSKeyedArchiver (нет ограничений на контейнеризацию)


0 1 2 3 4 5 6 7 8 9 0 5

```swift
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}

// usage
let productCode: Barcode = scaner.getCode()

switch productCode {
case .upc(let numberSystem, let manufacturer, let product, let check):
    print("UPC: \(numberSystem), \(manufacturer), \(product), \(check).")
case .qrCode(let productCode):
    print("QR code: \(productCode).")
}
```

```
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}

// an array of codes
let codes: [Barcode] = warehouse.availableProductCodes()

// we need to serialize the array
let codesData = ...
```

```
enum Barcode: Codable {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}

// an array of codes
let codes: [Barcode] = warehouse.availableProductCodes()

// we need to serialize the array
let codesData = ...
```

```
enum Barcode: Codable {                    2 Type 'Barcode' does not conform to protocol 'Decodable'
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}
```

Automatic Codable conformance for enums with associated values that themselves conform to Codable

Evolution Pitches

jaredsinclair 2 Mar '18

At least as of Swift 4.1, an enum with one or more cases with associated values cannot participate in synthesized `Codable` conformance. For example, the following will yield compiler errors:

```
enum MyOtherEnum: Codable { // errors: does not conform to Encodable/Decodab
    case foo
    case bar(label: Int)
}
```

This is true even if all the associated values in said enum themselves conform to `Codable`. In a similar fashion as the synthesized `Hashable` / `Equatable` implementations for enums with associated values that themselves conform to `Hashable` / `Equatable`, it would be useful for the Swift compiler to synthesize `Codable` conformance automatically for any enum with associated values so long as every unique occurrence of an associated value type itself already conforms to `Codable`.

In broad strokes, I'd expect the above code snippet to yield a synthesized implementation much like this:

```
enum MyEnum: Codable {
    case foo
    case bar(label: Int)
}
```

```
enum Barcode: Codable {
    case upc(Int, Int, Int, Int)
    case qrCode(String)

    init(from decoder: Decoder) throws {
        // ...
    }

    func encode(to encoder: Encoder) throws {
        // ...
    }
}
```

```
enum Barcode: Codable {
    case upc(Int, Int, Int, Int)
    case qrCode(String)

    enum CodingKeys: CodingKey {
        case upcSystem, upcManufacturer, upcProduct, upcCheck
        case qrString
    }

    // ...
}
```

```
{ “upc” :  {
        “system”  : 7,
        “manufacturer”  : 367854,
        “product”  : 67854,
        “check”  : 13
    }
}

{ “qrCode” : stringValue }
```

```
{ “upcSystem” : 7 }
{ “upcManufacturer” : 367854 }
{ “upcProduct” : 67854 }
{ “upcCheck” : 13 }

{ “qrCode” : stringValue }
```

```
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    switch self {

    // UPC
    case .upc(let system, let manufacturer, let product, let check):
        try container.encode(system, forKey: .upcSystem)
        try container.encode(manufacturer, forKey: .upcManufacturer)
        try container.encode(product, forKey: .upcProduct)
        try container.encode(check, forKey: .upcCheck)

    // QR
    case .qrCode(let stringValue):
        try container.encode(stringValue, forKey: .qrString)
    }
}
```

```
init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)

    // QR
    if let qrString = try? values.decode(String.self, forKey: .qrString) {
        self = .qrCode(qrString)
        return

    // UPC
    } else if let upsSystem = try? values.decode(Int.self, forKey: .upsSystem),
        let upcManufacturer = try? values.decode(Int.self, forKey: .upcManufacturer)
        let upcProduct = try? values.decode(Int.self, forKey: .upcProduct)
        let upcCheck = try? values.decode(Int.self, forKey: .upcCheck) {
        self = .qrCode(upsSystem , upcManufacturer, upcProduct, upcCheck)
        return

    } else {
        throw EncodingError.dataCorrupted
    }
}
```

Enum with Associated types (Sourcery)

- Добавить промежуточные протоколы

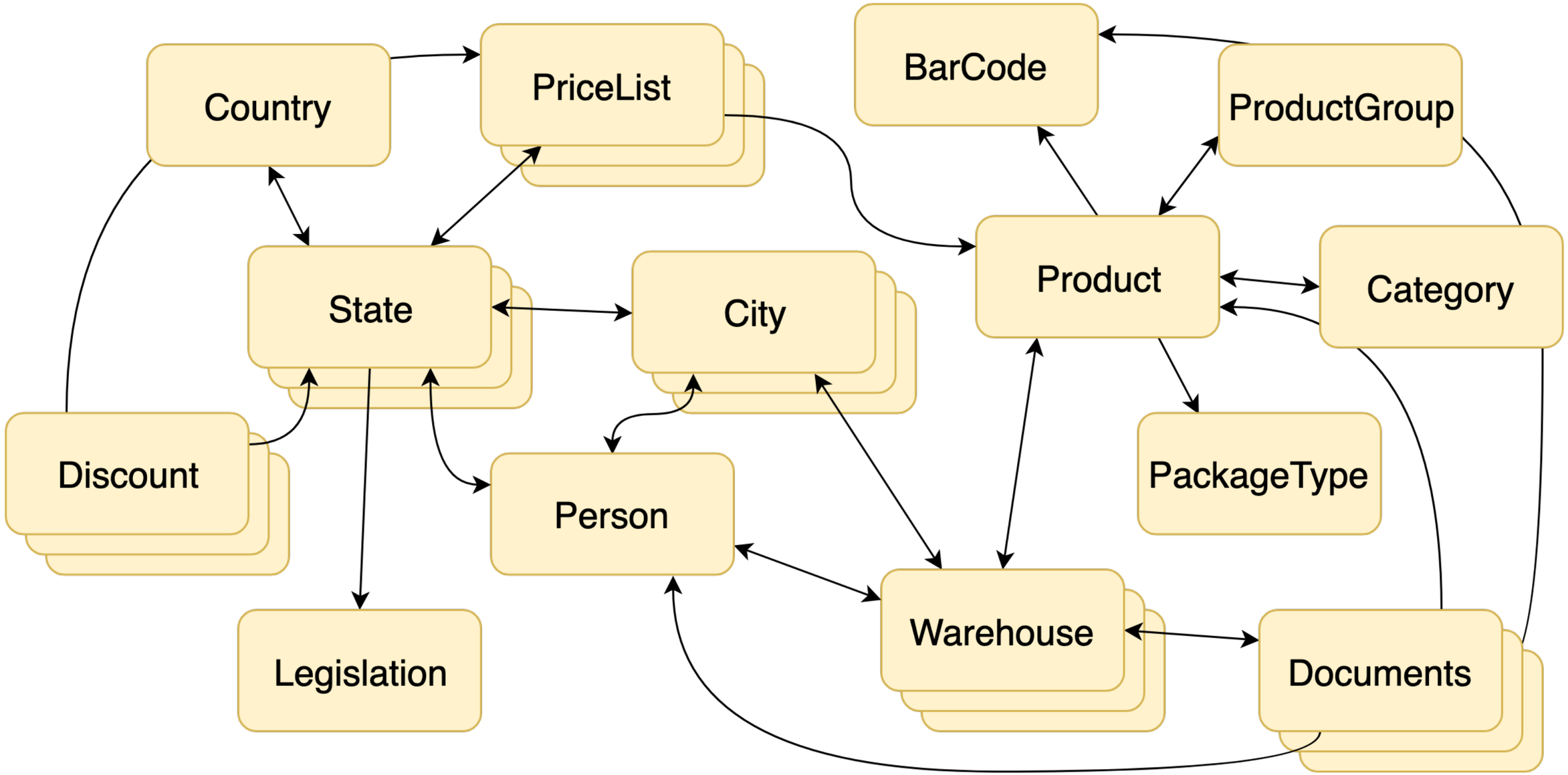
```
protocol AutoDecodable: Decodable {}
protocol AutoEncodable: Encodable {}
protocol AutoCodable: AutoDecodable, AutoEncodable {}
```

- Обозначить реализацию протокола

```
enum Barcode: AutoCodable {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}
```

- Импортировать шаблон `AutoCodable.swifttemplate`
- Сгенерировать расширение и добавить файл в проект

8. Сериализация графа объектов


Country
PriceList
BarCode
ProductGroup
State
City
Product
Category
Discount
PackageType
Person
Warehouse
Legislation
Documents

Сохранение графа объектов:

Сохранение графа объектов:

1. CoreData, Realm

Сохранение графа объектов:

1. CoreData, Realm
2. SQLite (+ORM)

Сохранение графа объектов:

1. CoreData, Realm
2. SQLite (+ORM)
3. Бинарные данные (файл, UserDefaults)

Core Data

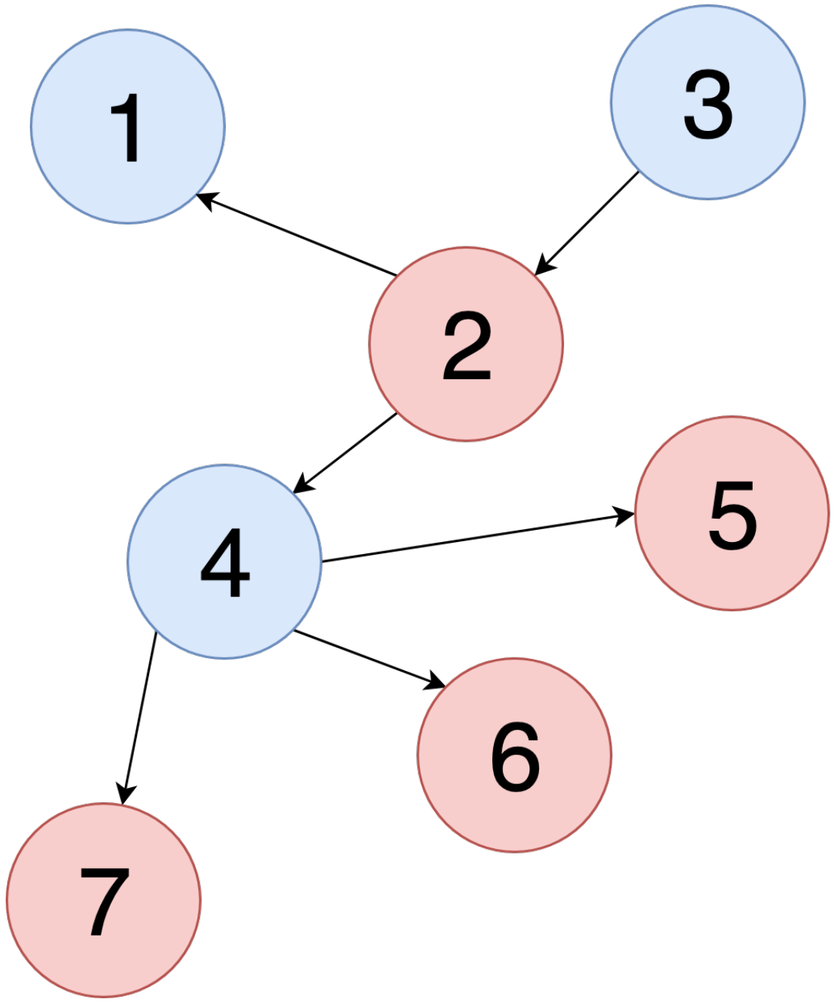
vs

NSKeyedArchiver

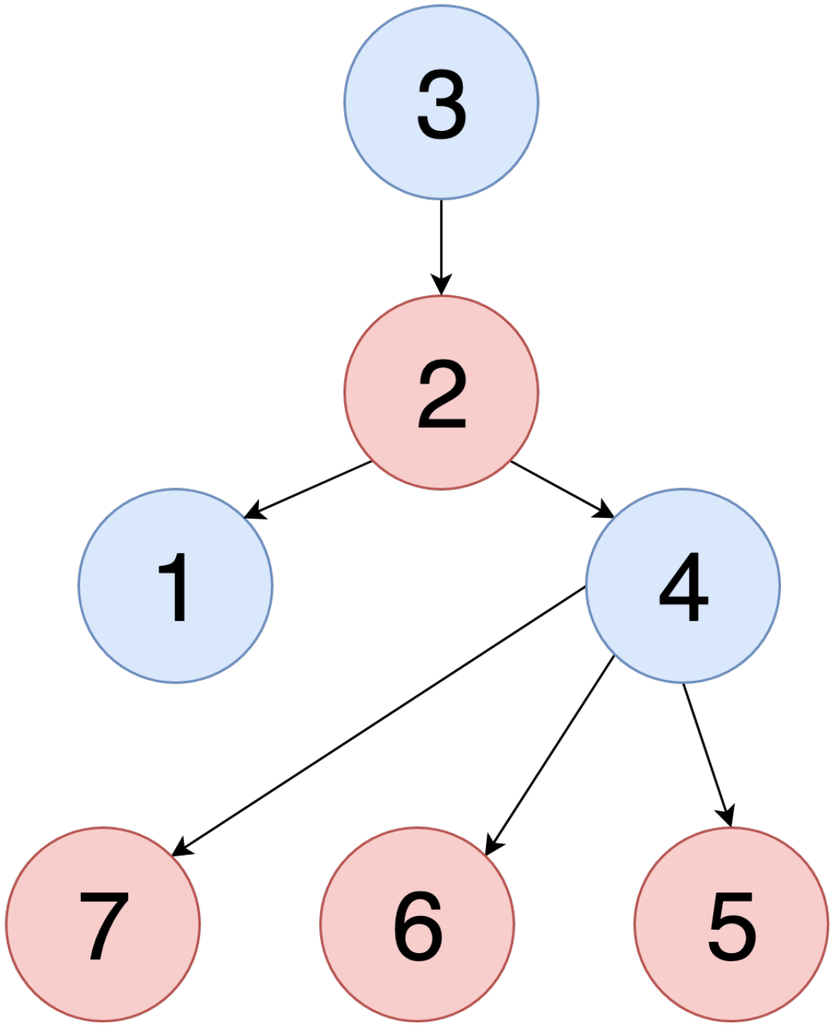
	Core Data	NSKeyedArchiver
Entity Modeling	Yes	No
Querying	Yes	No
Speed	Fast	Slow
Serialization Format	SQLite, XML, or NSData	NSData
Migrations	Automatic	Manual
Undo Manager	Automatic	Manual

	Core Data	NSKeyedArchiver
Persists State	Yes	Yes
Pain in the Ass	Yes	No

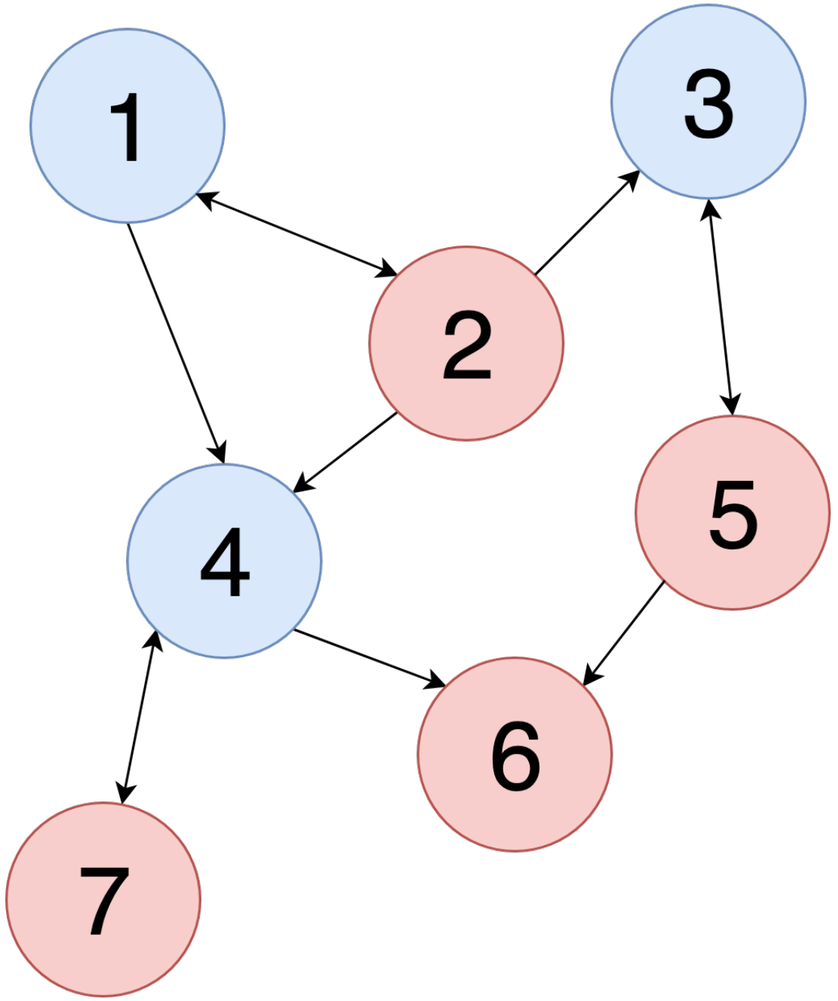
ГРАФ



ДЕРЕВО



СЛОЖНЫЙ ГРАФ



Трудности при сериализации сложного графа

Трудности при сериализации сложного графа

- Наличие циклических ссылок

Трудности при сериализации сложного графа

- Наличие циклических ссылок
- Наличие нескольких ссылок на один и тот же объект

Трудности при сериализации сложного графа

- Наличие циклических ссылок
- Наличие нескольких ссылок на один и тот же объект
- Сериализация условных объектов

Трудности при сериализации сложного графа

- Наличие циклических ссылок
- Наличие нескольких ссылок на один и тот же объект
- Сериализация условных объектов

NSKeyedArchiver: NSCoder

Сериализация объекта через NSCoder

```
encodeObject:forKey:
encodeRootObject:
encodeConditionalObject:forKey:
encodeBycopyObject:
encodeByrefObject:
```

Работа со сложным графом

Codable + JSONEncoder
Codable + PropertyListEncoder

Работа со сложным графом

Codable + JSONEncoder
Codable + PropertyListEncoder

Нет поддержки

Codable with references?

Using Swift | codable

BigZaphod Sean Heber — Jun '18

Does the automatic Codable synthesis not support references? I just built out a very large graph I was going to serialize using the automatic Codable stuff, but near as I can tell it ends up in an infinite loop when attempting to encode it. I'm assuming this is probably because of the various references and back references going on in the data. Does this mean I need to write my own Encoder/Decoder for some made up binary format that can handle references or something?

1 ♡ Reply

created	last reply	replies	views	users	likes	links
Jun '18	Feb 6	66	3.2k	11	23	15

Frequent Posters

Popular Links

31 GitHub - Flight-School/Codable-DIY-Kit: A template for creating your own Swift Codabl... github.com

30 GitHub - greg/CyclicCoding: Encodes & decodes Codable object graphs containing duplic... github.com

25 GitHub - BigZaphod/Archivable: A custom Swift 4.2 Archiver that supports reference ty... github.com

21 Codable != Archivable swift.org

Swift

Codable != Archivable

Evolution | Discussion

QuinceyMorris — Mar '18

In SE-0167 8 we find this statement about `NSKeyedArchiver` and `NSKeyedUnarchiver`:

> Although our primary objectives for this new API revolve around Swift, we would like to make it easy for current consumers to make the transition to Codable where appropriate. As part of this, we would like to bridge compatibility between new Codable types (or newly-Codable-adopting types) and existing NSCoding types.
>
> To do this, we want to introduce changes to NSKeyedArchiver and NSKeyedUnarchiver in Swift that allow archival of Codable types intermixed with NSCoding types

(Yes, I'm squarely in Cocoa-land, because that's one prominent scenario where the problem shows up. However, this is a definitely a Swift issue, not Cocoa.)

The problem is that this stated goal isn't achievable by `encodeEncodable` and `decodeDecodable` as currently implemented. As a practical exercise, I took a very ordinary, simple data model that was previously being archived for `NSDocument`, and changed all the `NSCoding` conformances to `Codable`. When I tried to save the document, my app crashed — crashed big, with infinite recursion leading to a stack overflow.

It turns out that archiving/unarching via `Codable` through keyed archivers/unarchivers doesn't respect the normal archiver convention of object reference identity. That is, reference type instances in `NSCoding` are unique within the archive as a whole. `Codable`, on the other hand, archives or unarchives a new instance at every reference encountered in the object graph. It crashes because

NSKeyedArchiver (работа с Codable)

```
extension NSKeyedArchiver {
    @available(OSX 10.11, iOS 9.0, *)
    func encodeEncodable<T>(_ value: T, forKey key: String) throws
        where T : Encodable
}

extension NSKeyedUnarchiver {
    @available(OSX 10.11, iOS 9.0, *)
    func decodeDecodable<T>(_ type: T.Type, forKey key: String) -> T?
        where T : Decodable
}
```

NSKeyedArchiver (работа с Codable)

```
let archiver = NSKeyedArchiver(requiringSecureCoding: false)
try? archiver.encodeEncodable(
    ourGraph,
    forKey: NSKeyedArchiveRootObjectKey
)
archiver.finishEncoding()
data = archiver.encodedData
try data!.write(to: fileUrl)
```


I DONT UNDERSTAND

Реализация (swift source code)

```
extension NSKeyedArchiver {
    func encodeEncodable<T : Encodable>
        (_ value: T, forKey key: String) throws {
        let plistEncoder = PropertyListEncoder()
        let plist = try plistEncoder.encodeToTopLevelContainer(value)
        self.encode(plist, forKey: key)
    }
}
```

Сериализация сложного графа Swift

Сериализация сложного графа Swift

- NSKeyedArchiver + NSCoding

Сериализация сложного графа Swift

- NSKeyedArchiver + NSCoding
- Добавить необходимую функциональность к JSONEncoder/ PropertyListEncoder

Сериализация сложного графа Swift

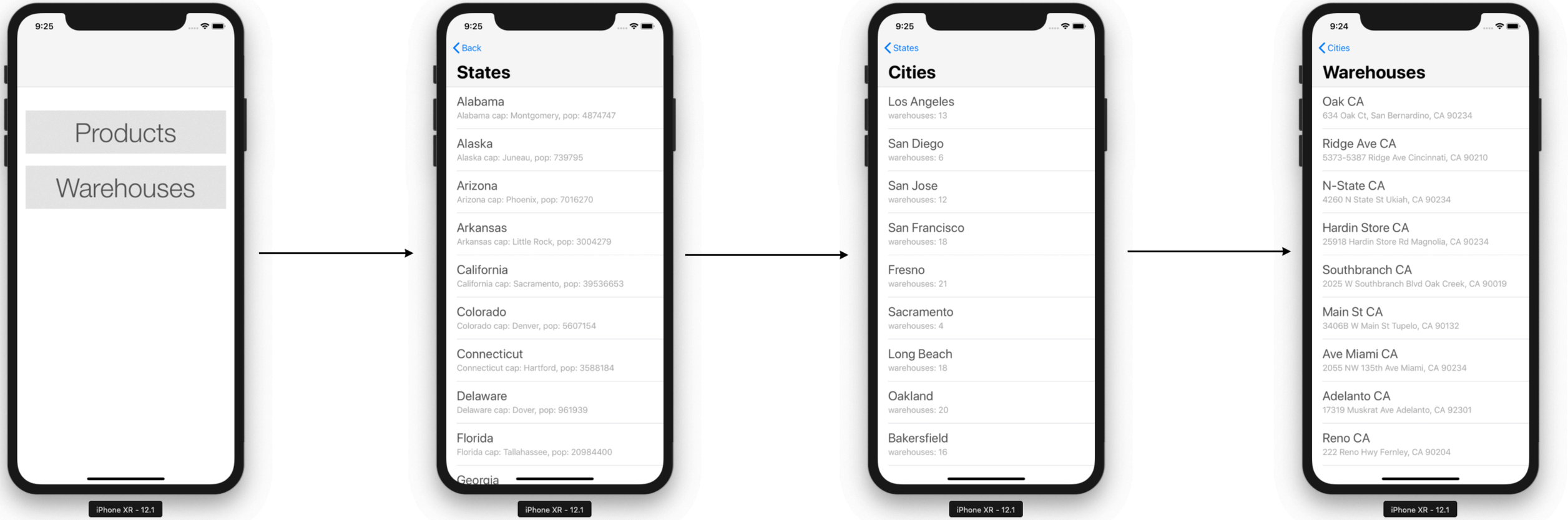
- NSKeyedArchiver + NSCoding
- Добавить необходимую функциональность к JSONEncoder/PropertyListEncoder
- Создать свою пару кодер/декодер реализующую Encoder/Decoder протоколы

Сериализация сложного графа Swift

- Сторонние библиотеки.

https://github.com/BigZaphod/Archivable
https://github.com/cherrywoods/swift-meta-serialization
https://github.com/greg/CyclicCoding
...

9. Сохранение состояния UI


9:25
Products
Warehouses
iPhone XR - 12.1
9:25
Back
States
Alabama
Alabama cap: Montgomery, pop: 4874747
Alaska
Alaska cap: Juneau, pop: 739795
Arizona
Arizona cap: Phoenix, pop: 7016270
Arkansas
Arkansas cap: Little Rock, pop: 3004279
California
California cap: Sacramento, pop: 39536653
Colorado
Colorado cap: Denver, pop: 5607154
Connecticut
Connecticut cap: Hartford, pop: 3588184
Delaware
Delaware cap: Dover, pop: 961939
Florida
Florida cap: Tallahassee, pop: 20984400
Georgia
iPhone XR - 12.1
9:25
States
Cities
Los Angeles
warehouses: 13
San Diego
warehouses: 6
San Jose
warehouses: 12
San Francisco
warehouses: 18
Fresno
warehouses: 21
Sacramento
warehouses: 4
Long Beach
warehouses: 18
Oakland
warehouses: 20
Bakersfield
warehouses: 16
iPhone XR - 12.1
9:24
Cities
Warehouses
Oak CA
634 Oak Ct, San Bernardino, CA 90234
Ridge Ave CA
5373-5387 Ridge Ave Cincinnati, CA 90210
N-State CA
4260 N State St Ukiah, CA 90234
Hardin Store CA
25918 Hardin Store Rd Magnolia, CA 90234
Southbranch CA
2025 W Southbranch Blvd Oak Creek, CA 90019
Main St CA
3406B W Main St Tupelo, CA 90132
Ave Miami CA
2055 NW 135th Ave Miami, CA 90234
Adelanto CA
17319 Muskrat Ave Adelanto, CA 92301
Reno CA
222 Reno Hwy Fernley, CA 90204
iPhone XR - 12.1

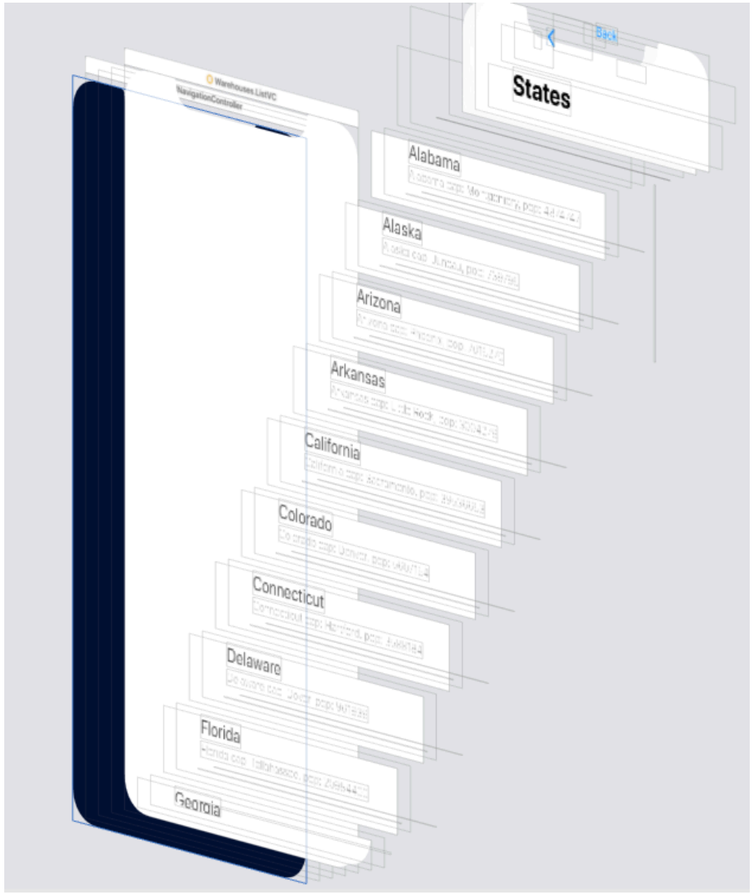
Сохранение типа экрана

```
enum ScreenType: Int {
    case main, states, cities, warehouses
}

let currentScreen = ScreenType.warehouses

func saveSurrentScreen() {
    UserDefaults.standard.set(currentScreen.rawValue, forKey: "kCurrentScreen")
}
```

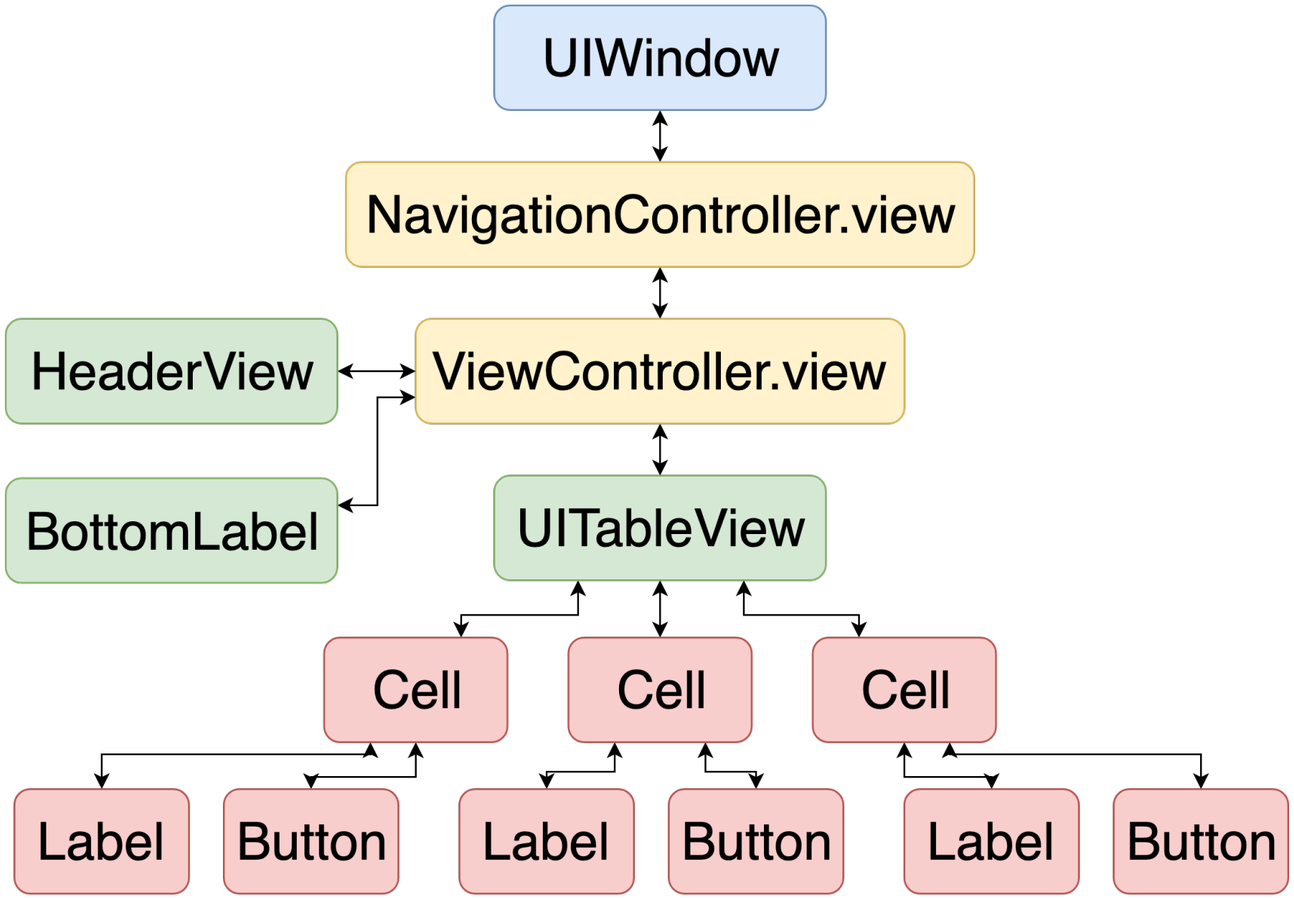
Сохранение пути к экрану

```
var currentScreen = URL(string: "root/states/cities")!

currentScreen.appendPathComponent("warehouses")

func saveSurrentScreen() {
    UserDefaults.standard.set(currentScreen, forKey: "kCurrentScreen")
}
```

Сохранение графа объектов



Сохранение графа объектов



Варианты сохранения UI

1. Сохранение типа экрана
2. Сохранение пути к экрану (deep link)
3. Сохранение графа объектов (UI + models)

Article

Preserving Your App's UI Across Launches

Return your app to its previous state after it is terminated by the system.

Framework

UIKit

On This Page

Overview

Preserving your app’s user interface helps maintain the illusion that your app is always running. Interruptions can occur frequently on iOS devices, and a prolonged interruption might cause the system to terminate your app to free up resources. However, users do not know that your app has been terminated and will not expect the state of your app to change. Instead, they expect your app to be in the same state as when they left it. State preservation and restoration ensures that your app returns to its previous state when it launches again.

At appropriate times, UIKit preserves the state of your app’s views and view controllers to an encrypted file on disk. When your app is terminated and relaunched later, UIKit reconstructs

Article

Preserving Your App's UI Across Launches

Return your app to its previous state after it is terminated by the system.

Framework

UIKit

NSKeyedArchiver + NSCoding

On This Page

Overview

Topics

See Also

Overview

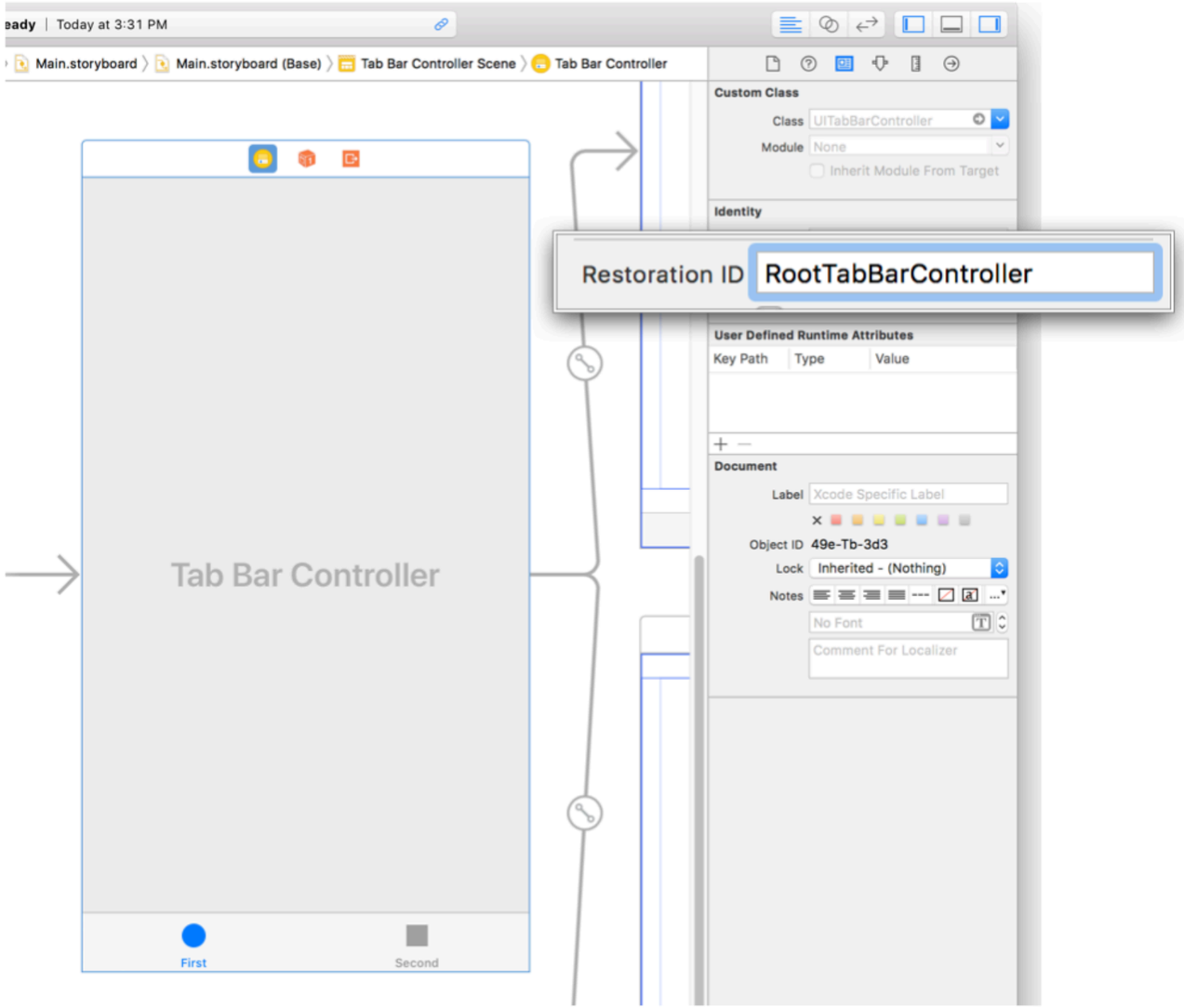
Preserving your app’s user interface helps maintain the illusion that your app is always running. Interruptions can occur frequently on iOS devices, and a prolonged interruption might cause the system to terminate your app to free up resources. However, users do not know that your app has been terminated and will not expect the state of your app to change. Instead, they expect your app to be in the same state as when they left it. State preservation and restoration ensures that your app returns to its previous state when it launches again.

At appropriate times, UIKit preserves the state of your app’s views and view controllers to an encrypted file on disk. When your app is terminated and relaunched later, UIKit reconstructs

1. Реализовать методы AppDelegate

```
func application(UIApplication,
        shouldSaveApplicationState: NSCoder) -> Bool

func application(UIApplication,
        shouldRestoreApplicationState: NSCoder) -> Bool
```

2. Добавить идентификаторы сохранения вью-контроллеров



3. Добавить код по сериализации связанного с контроллером состояния.

```
override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if firstNameField!.isFirstResponder {
        coder.encode(firstNameField?.text, forKey: "EditedText")
        coder.encode(Int32(1), forKey: "EditField")
    }
    else if lastNameField!.isFirstResponder {
        coder.encode(lastNameField?.text, forKey: "EditedText")
        coder.encode(Int32(2), forKey: "EditField")
    }
    else {
        // No editing was in progress.
        coder.encode(Int32(0), forKey: "EditField")
    }
}
```

Недостатки подхода

Недостатки подхода

1. Всё работает идеально если сториборды

Недостатки подхода

1. Всё работает идеально если сториборды
2. Параллельная логика инициализации дерева объектов.

Недостатки подхода

1. Всё работает идеально если сториборды

2. Параллельная логика инициализации дерева объектов.

Init UI -> ViewControllers -> View -> UI State
Init Model -> Model graph, Business logic, services, ...

10. NSCoding vs. Codable

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	

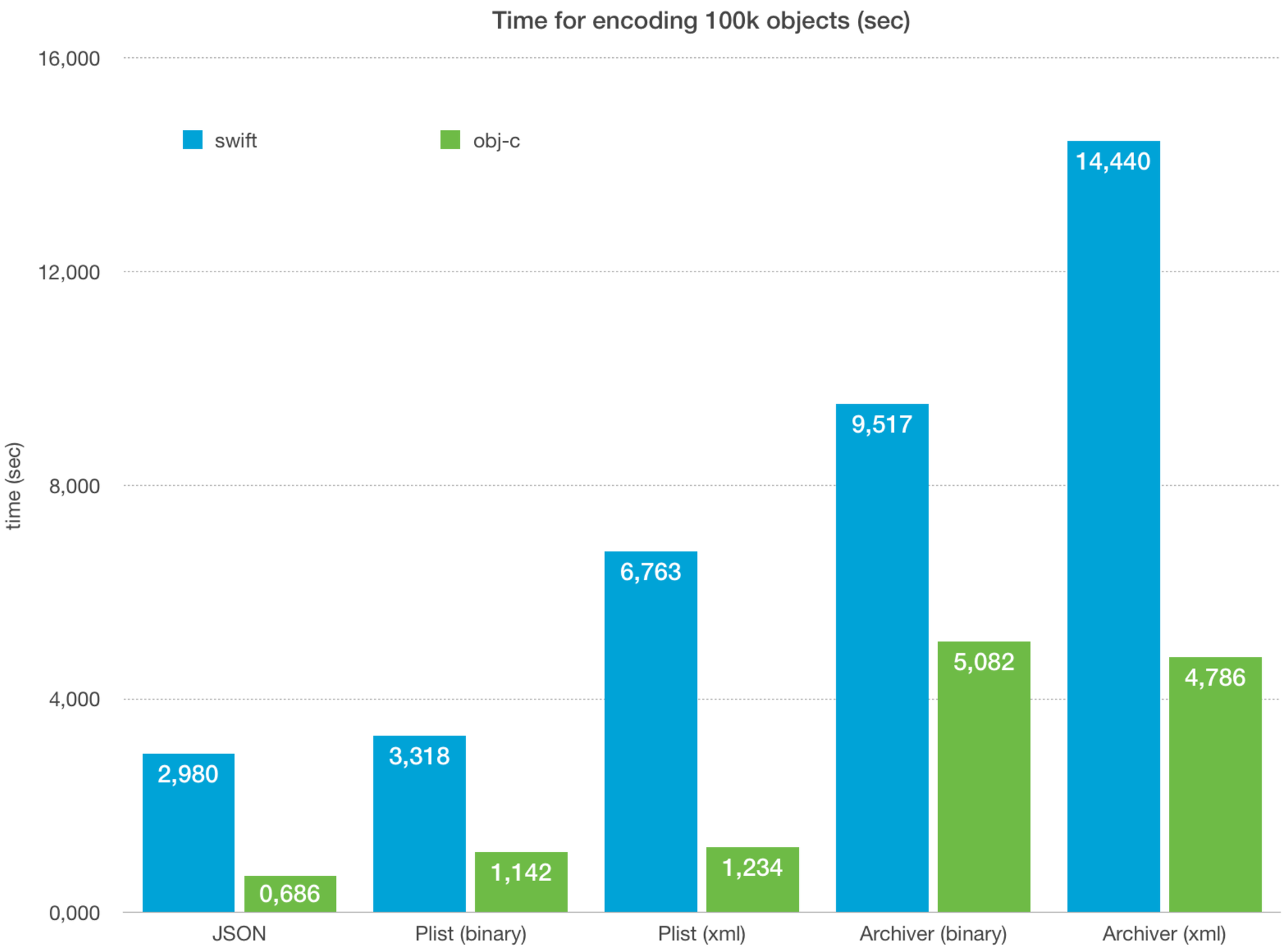
NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	Сохранение, Передача (XML, bin, JSON, Plist, ...)

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	Сохранение, Передача (XML, bin, JSON, Plist, ...)
-	Кодогенерация

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	Сохранение, Передача (XML, bin, JSON, Plist, ...)
-	Кодогенерация
Сериализация сложных графов	-

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	Сохранение, Передача (XML, bin, JSON, Plist, ...)
-	Кодогенерация
Сериализация сложных графов	-
ObjC, Swift (не struct, enum)	Swift

NSCoder	Codable
Реализован большинством объектов Foundation и UIKit	AffineTransform, Calendar, CharacterSet, Date, DateComponents, DateInterval, Decimal, IndexPath, IndexSet, Locale, Measurement, Notification, PersonNameComponents, TimeZone, URL, URLComponents, URLRequest, UUID


Time for encoding 100k objects (sec)
16,000
12,000
8,000
4,000
0,000
time (sec)
swift
obj-c
2,980
0,686
3,318
1,142
6,763
1,234
9,517
5,082
14,440
4,786
JSON
Plist (binary)
Plist (xml)
Archiver (binary)
Archiver (xml)

NSCoder	Codable
Сериализация через контейнер	Сериализация через контейнер
Больше ограничений при создании кодеров	Гибкая система создания кодеров
Сохранение (XML, bin)	Сохранение, Передача (XML, bin, JSON, Plist, ...)
-	Кодогенерация
Сериализация сложных графов	-
ObjC, Swift (не struct, enum)	Swift
Широкая реализация Foundation и UIKit	Ограниченный набор классов Foundation и UIKit
Быстрее	Медленнее

Заключение

- Сериализация для передачи (JSON)
- Сериализация для сохранения (Plist, XML)
- NSCoding, NSCoder, NSKeyedArchiver
- Codable (JSONEncoder, PropertyListEncoder, Codable + NSKeyedArchiever)
- NSCoder vs Codable

Заключение

- Создание промежуточной модели (Obj-C JSON)
- Сериализация энума с ассоциативными типами (Swift)
- Вложенная сериализация (Codable + NSCoding)
- Сериализация сложных графов
- Сохранение состояния UI

Нативная сериализация в iOS

Дмитрий Иванов

twitter: dmtopolog
email: dmtopolog@gmail.com
blog: dmtopolog.com