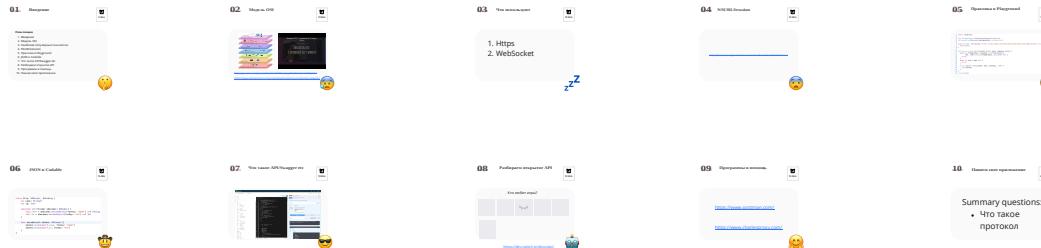


```
1 import Foundation
2
3 let configuration = URLSessionConfiguration.default
4 let session = URLSession(configuration: configuration)
5
6 guard let url = URL(string: "https://itunes.apple.com/search?media=music&entity=song&term=cohen") else {
7     fatalError()
8 }
9
10 let task = session.dataTask(with: url) { data, response, error in
11     guard let httpResponse = response as? HTTPURLResponse,
12         (200..<300).contains(httpResponse.statusCode) else {
13         return
14     }
15     guard let data = data else {
16         return
17     }
18     if let result = String(data: data, encoding: .utf8) {
19         print(result)
20     }
21 }
22 task.resume()
```



HTTP

- текстовый протокол
- передача информации в Интернете
- де-факто основной транспорт в текущий момент времени

HTTP. Методы

- OPTIONS** - получение информации о веб-сервере
- GET** - получение содержимого ресурса, параметры идут после «?».
- HEAD** - тот же GET, но без тела в ответе. Нужен чтобы проверить наличие ресурса или дату его изменения
- POST** - модификация ресурса
- PUT** - создание нового ресурса, либо обновление существующего
- PATCH** - тот же PUT, но для обновления фрагмента ресурса
- DELETE** - удаление ресурса
- TRACE** - проверка изменения заголовка промежуточными серверами

HTTP. Идемпотентность

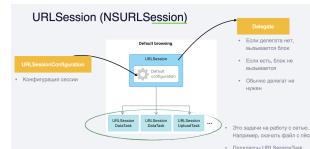
- это страшное слово означает способность метода вернуть один и тот же результат при многократных запросах с одним и тем же параметром
- идемпотентные методы HTTP: **GET, HEAD, PUT, DELETE**
- идемпотентные методы **МОЖНО КОШИРОВАТЬ** на клиенте (что это делается фреймворками и непрозрачно для разработчика)
- POST** не кешируется

HTTP. Коды состояний

- информационные (**1xx**)-практически не используются
- успешно(**2xx**)-запрос успешно выполнен
- перенаправление(**3xx**)-переадресация на другую страницу
- ошибка клиента(**4xx**)-что то пошло не так на стороне пользователя
- ошибка сервера(**5xx**)-что то поломалось на стороне сервера

HTTPS

- обычный HTTP работающий поверх SSL/TLS
- SSL/TLS - прослойка между транспортным и прикладным уровнями, обеспечивающая шифрованное соединение



NSURLSessionConfiguration

- let config = URLSessionConfiguration.default // Текущие настройки конфигурации для уже не используемого URLSession
- let session = URLSession(configuration: config) // Получение URLSession, настроенного с конфигурацией config
- Создание пакета, который содержит тело запроса, требований к содержимому, дополнительные HTTP-запросы, а также URL-путь доступа к сети и многое другое
- task:
 - default - по умолчанию
 - asynchronous - отправляет на диск файл, storage или любые другие данные
 - streaming - Хранит в кешере
- Задающее пользователю ограничения и начиная давать приложению время для выполнения
- NSURLSession.sharedSession - для базовых запросов. Не конфигурируется

Первый вариант (старый) - разобрать пришедшний JSON

```
let task = URLSession.shared.dataTask(with: url, completionHandler: { (data, response, error) in
    if let err = error {
        print("Error: \(err.localizedDescription)")
    }
    if let data = data {
        let json = JSON(data: data)
        let title = json["title"].string ?? "Unknown"
        let subtitle = json["subtitle"].string ?? "Unknown"
        print("Title: \(title), Subtitle: \(subtitle)")
    }
})
task.resume()
```

Второй вариант - Decodable

```
struct Post: Decodable {
    var id: Int
    var title: String
    var subtitle: String
}

do {
    let posts = try JSONDecoder().decode(Post.self, from: jsonData)
} catch {
    print(error)
}
```

Decodable со своими ключами

```
struct Post: Decodable {
    var id: Int
    var title: String
    var subtitle: String
}

enum CodingKeys: String, CodingKey {
    case id = "id"
    case title = "title"
    case subtitle = "subtitle"
}

extension Post: Encodable {
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        container.encode(id, forKey: .id)
        container.encode(title, forKey: .title)
        container.encode(subtitle, forKey: .subtitle)
    }
}
```

Тогда для того, чтобы закодировать - Encodable

```
struct PhotoCategory: Encodable {
    var name: String
    var id: Int
    var photoCount: Int
    var photo: Photo?
}

enum CodingKeys: String, CodingKey {
    case name = "name"
    case id = "id"
    case photoCount = "photoCount"
    case photo = "photo"
}

extension PhotoCategory: Encodable {
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        container.encode(name, forKey: .name)
        container.encode(id, forKey: .id)
        container.encode(photoCount, forKey: .photoCount)
        container.encode(photo, forKey: .photo)
    }
}
```

В реальном мире используйте ёщё и DTO

```
struct PhotoCategory {
    var name: String
    var id: Int
    var photoCount: Int
    var photo: Photo?
}

enum PhotoCategoryType: String {
    case photo
    case video
}

var photoCategory: PhotoCategory {
    var name: String = "Photo"
    var id: Int = 1
    var photoCount: Int = 0
    var photo: Photo?
}

// когда
var photoCategory: PhotoCategory {
    var id: Int = 1
    var photoCount: Int = 0
    var photo: Photo?
}

init(id: Int, photoCount: Int, photo: Photo?) {
    self.id = id
    self.photoCount = photoCount
    self.photo = photo
}
```



NSURLSessionTask

- Создание задачи на работу с интернетом:
- let url = URL(string: "https://developer.apple.com/documentation")!
- let task = URLSession.shared.dataTask(with: url, completionHandler: { (data, response, error) in
 // то, что будет дано после выполнения задачи
 })
- После создания:
- task.cancel() // отменяет задачу на выполнении
- task.cancel() // отменяет задачу
- task.cancel() // ставит задачу на паузу

NSURLSessionTask

- Может подождать пока не перед URL, и с помощью URLRequest:
- let session = URLSession(configuration: config)
let task = session.dataTask(with: URL(string: "https://developer.apple.com/documentation")!)
let request = URLRequest(url: URL(string: "https://developer.apple.com/documentation")!)
task.request = request
task.resume()
// то, что будет дано после выполнения задачи

Что в итоге получилось:

```
let session = URLSession(configuration: config)
let url = URL(string: "https://developer.apple.com/documentation")
let request = URLRequest(url: url)
let task = URLSession.shared.dataTask(with: url, completionHandler: { (data, response, error) in
    if let err = error {
        print("Error: \(err.localizedDescription)")
    }
    if let data = data {
        let json = JSON(data: data)
        let title = json["title"].string ?? "Unknown"
        let subtitle = json["subtitle"].string ?? "Unknown"
        print("Title: \(title), Subtitle: \(subtitle)")
    }
})
task.resume()
```



01

Введение



5 Min

План лекции

- 1. Введение
- 2. Выбираем 3 добровольцев
- 3. Добавляем зависимость руками
- 4. CocoaPods
- 5. Carthage
- 6. SPM

- 1) Никита
- 2) Михаил
- 3) Саша

Package managers

План лекции

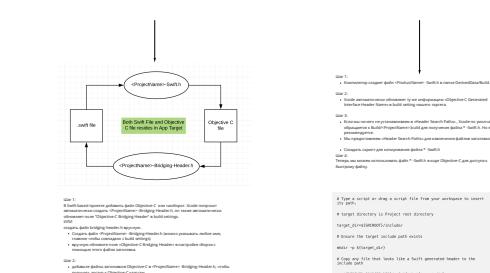
1. Введение
2. Что такое интеропрототип?
3. Разберем возможные сценарии на практике
4. Доказка



Swift and Objective-C



Код Swift и Objective-C находятся в одном таргете приложения. Как код Objective-C используется в классах Swift?



Swift and Objective-C



Код Swift и Objective-C находятся в одном таргете приложения. Как Swift код используется в классах Objective-C?



Есть статическая библиотека Swift. Как использовать ее в таргете приложения?

Шаг 1: Создать Swift-модуль. Swift-модуль может содержать только один файл, но может содержать несколько классов, функций и т.д.
Шаг 2: Добавить файл в таргет приложения. Для этого в меню `File -> Add Files to "YourProjectName"`. Выберите `Swift module` и выберите созданный в шаге 1 файл.

Шаг 3: Установите зависимости в `Build Phases -> Link Binary With Libraries`. Выберите `Swift module` и выберите созданный в шаге 1 файл.

Шаг 4: Установите зависимости в `Build Phases -> Copy Headers`.

Шаг 5: Установите зависимости в `Build Phases -> Preprocessor Macros`.

Шаг 6: Установите зависимости в `Build Phases -> Swift Compiler`.



Есть статическая библиотека Objective-C. Как использовать ее в таргете приложения?

Шаг 1: Создать Objective-C-модуль. Objective-C-модуль может содержать только один файл, но может содержать несколько классов, функций и т.д.
Шаг 2: Добавить файл в таргет приложения. Для этого в меню `File -> Add Files to "YourProjectName"`. Выберите `Objective-C module` и выберите созданный в шаге 1 файл.

Шаг 3: Установите зависимости в `Build Phases -> Link Binary With Libraries`. Выберите `Objective-C module` и выберите созданный в шаге 1 файл.

Шаг 4: Установите зависимости в `Build Phases -> Copy Headers`.

Шаг 5: Установите зависимости в `Build Phases -> Preprocessor Macros`.

Шаг 6: Установите зависимости в `Build Phases -> Objective-C Compiler`.



Как статическая библиотека Swift используется в другой статической библиотеке Swift/Objective-C?



Как статическая библиотека Objective-C используется в другой статической библиотеке Swift/Objective-C?

Swift and Objective-C



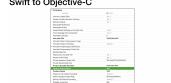
Bridging-header file



Objective-C to Swift



Swift to Objective-C



Swift to Objective-C



Swift to Objective-C



Swift to Objective-C



Swift to Objective-C



Swift to Objective-C



Swift to Objective-C



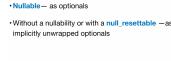
Objective-C Nullability



Objective-C Nullability



Objective-C Nullability



Objective-C Nullability



Objective-C Nullability



Objective-C Nullability



Read



1. View Construction
2. Compared to UIKit
3. View Updates
4. Environment
5. Layout

- Для созданияью в SwiftUI мы создаем дерево значений ящиков, описывающее, что должно быть на экране.
- Чтобы изменить то, что отображается на экране мы меняем состояние состояния (`State`), и вычисляется новое дерево значений.
- Затем SwiftUI обновляет экран, чтобы отразить эти новые значения

- В UIKit долголежущие экземпляры `UIView` и `UIViewController`. Их значения можно сбрасывать.
- В SwiftUI View это протокол, которому соответствуют структуры, их значения не делятся.
- Мы не можем напрямую обновить то, что отображается на экране. Вместо этого мы должны изменять свойства состояния

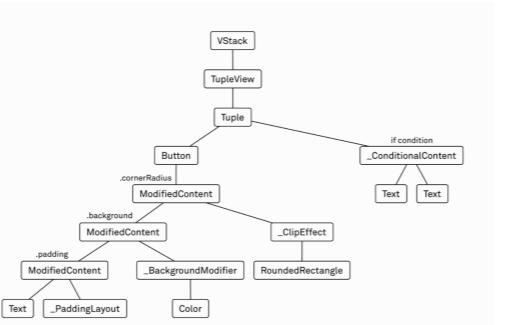
- Свойства, которые должны транслировать обновления ящиков, завернуты в `PropertyWrapper`, `@ObservedObject`, `@StateObject`
- В SwiftUI View это протокол, которому соответствуют структуры, их значения не делятся.
- Мы не можем напрямую обновить то, что отображается на экране. Вместо этого мы должны изменять свойства состояния
- Когда нам нужно сохранять и наблюдать за состоянием вне элемента управления, мы можем использовать `State` и `@Binding` и делаем его публичным.
- Когда мы хотим преобразовать значение ящика для него габаритное можно использовать `Environment`.
- Можно рассматривать как механизм DI, но пока так никто не делает (не мозги, пожалуйста, языки и UIKit)
- Еще есть `Preference`, они пропагируются не ящика, а наружу

- Когда нам нужно сохранять и наблюдать за состоянием вне элемента управления, мы можем использовать `State` и `@Binding` и делаем его публичным.
- Когда мы хотим преобразовать значение ящика для него габаритное можно использовать `Environment`.
- Можно рассматривать как механизм DI, но пока так никто не делает (не мозги, пожалуйста, языки и UIKit)
- Еще есть `Preference`, они пропагируются не ящика, а наружу

Demo Time

```
import SwiftUI

struct ContentView: View {
    @State var counter = 0
    var body: some View {
        VStack {
            Button(action: { counter += 1 }, label: {
                Text("Tap me!")
                    .padding()
                    .background(Color.tertiarySystemFill)
                    .cornerRadius(5)
            })
            if counter > 0 {
                Text("You've tapped \(counter) times")
            } else {
                Text("You've not yet tapped")
            }
        }
    }
}
```



```
var body: some View {
    VStack {
        TupleView<
            Button<
                ModifiedContent<
                    ModifiedContent<
                        Text,
                        _PaddingLayout
                    >,
                    _BackgroundModifier<Color>
                >,
                _ClipEffect<RoundedRectangle>
            >
        >,
        ConditionalContent<Text, Text>
    }
}
```

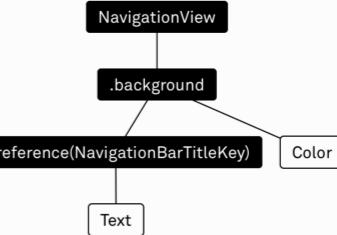
```
var body: some View {
    VStack {
        Text("Hello, world!")
            .transformEnvironment(\.font) { dump($0) }
    }
}
```

```
// ...
// - style: SwiftUI.Font.TextStyle.headline
}
/* 
ModifiedContent<
VStack<Text>,
_EnvironmentKeyWritingModifier<Optional<Font>>
>
*/

```

```
NavigationView {
    Text("Hello")
        .navigationBarTitle("Root View")
        .background(Color.gray)
}

```



#View Controllers

UIKit	SwiftUI	Note
UIViewController	View	
UITableViewController	List	You can also use <code>ScrollView</code> with <code>LazyHStack</code> or <code>LazyStack</code>
UICollectionViewController	LazyVGrid and LazyHGrid	Currently there is no SwiftUI view replacement for this, but you can simulate some layout with composing of List as in Composing Complex Interfaces's tutorial . In iOS 14, we now have <code>LazyVGrid</code> and <code>LazyHGrid</code> .
UISplitViewController	NavigationView	
UINavigationController	NavigationView	
UIPageViewController	TabView	A style of <code>TabView</code> in iOS 14
UITabBarController	TabView	
UISearchController	-	
UIImagePickerController	-	
UIVideoEditorController	-	
UIActivityViewController	-	
UIAlertController	Alert	

#Views and Controls

UIKit	SwiftUI	Note
UILabel	Text, Label	
UITabBar	TabView	
UITabBarItem	TabView	.tabItem under TabView
UITextField	TextField	For password (<code>isSecureTextEntry</code>) use <code>SecureField</code>
UITextView	TextEditor	iOS 14
UITableView	List	also <code>VStack</code> and <code>Form</code>
UINavigationBar	NavigationView	Part of <code>NavigationView</code>
UINavigationItem	ToolbarItem	iOS 14
UIBarButtonItem	NavigationView	.navigationItems in <code>NavigationView</code>
UICollectionView	LazyVGrid and LazyHGrid	iOS 14
UIStackView	HStack, LazyHStack	.axis == .Horizontal
UIStackView	VStack, LazyVStack	.axis == .Vertical
UIScrollView	ScrollView	
UIActivityIndicatorView	ProgressView with <code>CircularProgressViewStyle</code>	iOS 14
UIImageView	Image	
UIPickerView	Picker	
UIButton	Button, Link	
UIDatePicker	DatePicker	
UIPageControl		iOS 14. Auto add to <code>TabView</code> with <code>pageTabBarStyle</code> style. You can control its appearance by <code>indexViewStyle</code> .
UIProgressView	ProgressView	iOS 14
UISegmentedControl	Picker	A style (<code>SegmentedPickerStyle</code>) of <code>Picker</code>
UISlider	Slider	
UIStepper	Stepper	
UISwitch	Toggle	
UIToolbar	NavigationView with <code>.toolbar</code>	iOS 14
MKMapView	Map	import <code>MKMapView</code> to use this view

#Framework Integration - SwiftUI in UIKit

Integrate SwiftUI views into existing apps, and embed UIKit views and controllers into SwiftUI view hierarchies.

UIKit	SwiftUI	Note
UIview (UIHostingController)	View	There is no direct convert to <code>UIview</code> , but you can use container view to add view from <code>UIViewController</code> into view hierarchy
UIViewController (UIHostingController)	View	

#Framework Integration - UIKit in SwiftUI

Integrate UIKit views into existing apps, and embed UIKit views and controllers into SwiftUI view hierarchies.

UIKit	SwiftUI	Note
UIview (UIHostingController)	View	

#Pure SwiftUI

iOS 14

In iOS 14, you can write the whole app without a need for UIKit. Checkout [App essentials in SwiftUI](#) session from WWDC2020.

UIKit	SwiftUI	Note
UIApplicationDelegate	App	
UIWindowSceneDelegate	Scene	