

# Обработка ошибок и управление сложностью в Swift

Презентация 3

10 октября 2025 г.

# Содержание

## Обработка ошибок

Исключения

Встроенные средства отладки (assert)

## Управление сложностью кода

Пространства имен

Модули

Пакеты

Импорт и экспорт

## Заключение

# Обработка ошибок в Swift

- ▶ Swift использует типобезопасную систему обработки ошибок
- ▶ Ошибки представлены типами, соответствующими протоколу Error
- ▶ Используются опциональные типы и Result для обработки ошибок
- ▶ Поддержка do-catch блоков для перехвата исключений

# Определение ошибок

```
1 // Определение пользовательских ошибок
2 enum NetworkError: Error {
3     case noConnection
4     case timeout
5     case invalidURL
6     case serverError(Int)
7 }
8
9 enum ValidationError: Error {
10    case emptyField
11    case invalidFormat
12    case tooShort(minLength: Int)
13    case tooLong(maxLength: Int)
14 }
15
16 // Функция, которая может выбросить ошибку
17 func fetchData(from url: String) throws -> String {
18     guard !url.isEmpty else {
19         throw NetworkError.invalidURL
20     }
21
22     // Симуляция сетевого запроса
23     if url.contains("timeout") {
24         throw NetworkError.timeout
25     }
26
27     return "Данные получены"
28 }
```

# Обработка ошибок с do-catch

```
1 // Обработка ошибок с помощью do-catch
2 func processNetworkRequest() {
3     do {
4         let data = try fetchData(from: "https://api.example.com")
5         print("Успешно: \(data)")
6     } catch NetworkError.noConnection {
7         print("Нет подключения к интернету")
8     } catch NetworkError.timeout {
9         print("Превышено время ожидания")
10    } catch NetworkError.invalidURL {
11        print("Неверный URL")
12    } catch NetworkError.serverError(let code) {
13        print("Ошибка сервера: \(code)")
14    } catch {
15        print("Неизвестная ошибка: \(error)")
16    }
17 }
18
19 // Обработка с помощью try?
20 let result = try? fetchData(from: "https://api.example.com")
21 if let data = result {
22     print("Данные: \(data)")
23 } else {
24     print("Ошибка при получении данных")
25 }
```

# Propagating ошибок

```
1 // Передача ошибок дальше по цепочке вызовов
2 func validateEmail(_ email: String) throws -> Bool {
3     guard !email.isEmpty else {
4         throw ValidationError.emptyField
5     }
6
7     guard email.contains "@" else {
8         throw ValidationError.invalidFormat
9     }
10
11    guard email.count >= 5 else {
12        throw ValidationError.tooShort(minLength: 5)
13    }
14
15    return true
16 }
17
18 func processUser(_ email: String) throws -> String {
19     let isValid = try validateEmail(email)
20     return "Пользователь обработан: \(email)"
21 }
22
23 // Использование propagating ошибок
24 do {
25     let message = try processUser("user@example.com")
26     print(message)
27 } catch ValidationError.emptyField {
28     print("Поле email не может быть пустым")
29 } catch ValidationError.invalidFormat {
30     print("Неверный формат email")
31 } catch {
32     print("Ошибка валидации: \(error)")
33 }
```

# Assert и Precondition

```
1 // Assert для отладочных проверок
2 func divide(_ a: Int, by b: Int) -> Int {
3     assert(b != 0, "Деление на ноль!")
4     return a / b
5 }
6
7 let result = divide(10, by: 2) // Работает normally
8 // let result2 = divide(10, by: 0) // Assertion failed в Debug
9
10 // Precondition для проверок в Release
11 func safeDivide(_ a: Int, by b: Int) -> Int {
12     precondition(b != 0, "Деление на ноль недопустимо")
13     return a / b
14 }
15
16 // AssertionFailure для недостижимого кода
17 func processValue(_ value: Int) {
18     switch value {
19     case 1...10:
20         print("Малое значение")
21     case 11...100:
22         print("Среднее значение")
23     default:
24         assertionFailure("Неожиданное значение: \(value)")
25     }
26 }
```

# Дополнительные средства отладки

```
1 // FatalError для критических ошибок
2 func loadConfiguration() -> String {
3     guard let config = Bundle.main.object(forInfoDictionaryKey: "Config") as?
4         String else {
5         fatalError("Конфигурация не найдена!")
6     }
7     return config
8 }
9 // print и debugPrint для отладки
10 func debugFunction() {
11     let data = [1, 2, 3, 4, 5]
12
13     print("Обычный вывод: \(data)")
14     debugPrint("Отладочный вывод: \(data)")
15
16     // dump для детального вывода
17     dump(data)
18 }
19 // Условная компиляция для отладки
20 #if DEBUG
21     print("Отладочная информация")
22 #else
23     print("Продакшн режим")
24 #endif
25 // Условные проверки
26 func processArray(_ array: [Int]) {
27     precondition(!array.isEmpty, "Массив не может быть пустым")
28     assert(array.allSatisfy { $0 > 0 }, "Все элементы должны быть положительными
29
30     ")
31     let sum = array.reduce(0, +)
32     print("Сумма: \(sum)")
33 }
```



# Управление сложностью кода

- ▶ Пространства имен для организации кода
- ▶ Модули для инкапсуляции функциональности
- ▶ Пакеты для управления зависимостями
- ▶ Импорт и экспорт для контроля доступа

# Пространства имен через структуры

```
1 // Пространство имен через структуры
2 struct MathUtils {
3     static func add(_ a: Int, _ b: Int) -> Int {
4         return a + b
5     }
6
7     static func multiply(_ a: Int, _ b: Int) -> Int {
8         return a * b
9     }
10
11    static let pi = 3.14159
12 }
13
14 struct StringUtils {
15     static func capitalize(_ string: String) -> String {
16         return string.capitalized
17     }
18
19     static func reverse(_ string: String) -> String {
20         return String(string.reversed())
21     }
22 }
23
24 // Использование пространств имен
25 let sum = MathUtils.add(5, 10)
26 let product = MathUtils.multiply(3, 4)
27 let capitalized = StringUtils.capitalize("hello world")
```

# Вложенные пространства имен

```
1 // Вложенные пространства имен
2 struct Network {
3     struct Request {
4         static func get(url: String) -> String {
5             return "GET запрос к \url"
6         }
7
8         static func post(url: String, data: String) -> String {
9             return "POST запрос к \url с данными: \data"
10        }
11    }
12
13    struct Response {
14        static func parse(_ data: String) -> [String: Any] {
15            return ["status": "success", "data": data]
16        }
17
18        static func validate(_ response: [String: Any]) -> Bool {
19            return response["status"] as? String == "success"
20        }
21    }
22 }
23
24 // Использование вложенных пространств имен
25 let request = Network.Request.get(url: "https://api.example.com")
26 let response = Network.Response.parse("some data")
27 let isValid = Network.Response.validate(response)
```

# Создание модулей

```
1 // Модуль для работы с пользователями
2 public struct User {
3     public let id: Int
4     public let name: String
5     public let email: String
6
7     public init(id: Int, name: String, email: String) {
8         self.id = id
9         self.name = name
10        self.email = email
11    }
12 }
13
14 public class UserManager {
15     private var users: [User] = []
16
17     public init() {}
18
19     public func addUser(_ user: User) {
20         users.append(user)
21     }
22
23     public func getUser(by id: Int) -> User? {
24         return users.first { $0.id == id }
25     }
26
27     public func getAllUsers() -> [User] {
28         return users
29     }
30 }
31
32 // Внутренние функции модуля (не экспортируются)
33 internal func validateUser(_ user: User) -> Bool {
34     return !user.name.isEmpty && user.email.contains("@")
35 }
```



# Уровни доступа в модулях

```
1 // Различные уровни доступа
2 public class PublicClass {
3     public var publicProperty: String = "public"
4     internal var internalProperty: String = "internal"
5     private var privateProperty: String = "private"
6
7     public init() {}
8
9     public func publicMethod() {
10         print("Публичный метод")
11     }
12
13     internal func internalMethod() {
14         print("Внутренний метод")
15     }
16
17     private func privateMethod() {
18         print("Приватный метод")
19     }
20 }
21
22 // Файл-приватные элементы
23 fileprivate class FilePrivateClass {
24     fileprivate var property: String = "fileprivate"
25 }
26
27 // Открытые классы для наследования
28 open class OpenClass {
29     open func openMethod() {
30         print("Открытый метод для переопределения")
31     }
32
33     public func publicMethod() {
34         print("Публичный метод")
35     }
36 }
```



# Swift Package Manager

```
1 // Package.swift файл
2 // swift-tools-version:5.7
3 import PackageDescription
4
5 let package = Package(
6     name: "MySwiftPackage",
7     platforms: [
8         .iOS(.v13),
9         .macOS(.v10_15)
10    ],
11    products: [
12        .library(
13            name: "MySwiftPackage",
14            targets: ["MySwiftPackage"]
15        ),
16    ],
17    dependencies: [
18        .package(url: "https://github.com/apple/swift-algorithms", from: "1.0.0")
19        ),
20        .package(url: "https://github.com/apple/swift-collections", from: "1.0.0"
21        )
22    ],
23    targets: [
24        .target(
25            name: "MySwiftPackage",
26            dependencies: [
27                .product(name: "Algorithms", package: "swift-algorithms"),
28                .product(name: "Collections", package: "swift-collections")
29            ]
30        ),
31        .testTarget(
32            name: "MySwiftPackageTests",
33            dependencies: ["MySwiftPackage"]
34        ),
35    ]
36)
```



# Структура пакета

```
1 // Sources/MySwiftPackage/Models/User.swift
2 public struct User {
3     public let id: UUID
4     public let name: String
5     public let email: String
6
7     public init(name: String, email: String) {
8         self.id = UUID()
9         self.name = name
10        self.email = email
11    }
12 }
13
14 // Sources/MySwiftPackage/Services/UserService.swift
15 import Algorithms
16 import Collections
17
18 public class UserService {
19     private var users: OrderedSet<User> = []
20
21     public init() {}
22
23     public func addUser(_ user: User) {
24         users.append(user)
25     }
26
27     public func getUsers() -> [User] {
28         return Array(users)
29     }
30
31     public func findUser(by name: String) -> User? {
32         return users.first { $0.name == name }
33     }
34 }
```



# Импорт модулей

```
1 // Импорт стандартных модулей
2 import Foundation
3 import UIKit
4 import SwiftUI
5
6 // Импорт сторонних пакетов
7 import Alamofire
8 import SwiftyJSON
9
10 // Условный импорт
11 #if canImport(UIKit)
12     import UIKit
13     typealias View = UIView
14 #elseif canImport(AppKit)
15     import AppKit
16     typealias View = NSView
17 #endif
18
19 // Импорт с переименованием
20 import Foundation as NSFoundation
21
22 // Импорт конкретных типов
23 import struct Foundation.URL
24 import class Foundation.URLSession
25
26 // Использование импортированных модулей
27 func makeNetworkRequest() {
28     let url = URL(string: "https://api.example.com")!
29     let session = URLSession.shared
30
31     let task = session.dataTask(with: url) { data, response, error in
32         // Обработка ответа
33     }
34     task.resume()
35 }
```



# Экспорт и публичные API

```
1 // Публичный API модуля
2 public protocol DataProvider {
3     func fetchData() async throws -> Data
4 }
5
6 public class NetworkDataProvider: DataProvider {
7     private let session: URLSession
8
9     public init(session: URLSession = .shared) {
10         self.session = session
11     }
12
13     public func fetchData() async throws -> Data {
14         let url = URL(string: "https://api.example.com")!
15         let (data, _) = try await session.data(from: url)
16         return data
17     }
18 }
19
20 // Внутренние типы (не экспортируются)
21 internal class CacheManager {
22     private var cache: [String: Data] = [:]
23
24     internal func store(_ data: Data, for key: String) {
25         cache[key] = data
26     }
27
28     internal func retrieve(for key: String) -> Data? {
29         return cache[key]
30     }
31 }
```

## Ключевые моменты

- ▶ Обработка ошибок обеспечивает надежность приложения
- ▶ Исключения позволяют элегантно обрабатывать ошибки
- ▶ **Assert** и **Precondition** помогают в отладке
- ▶ Пространства имен организуют код логически
- ▶ Модули инкапсулируют функциональность
- ▶ Пакеты управляют зависимостями
- ▶ Импорт/экспорт контролируют доступ к API

## Практические рекомендации

- ▶ Используйте `do-catch` для обработки ошибок
- ▶ Применяйте `assert` только для отладочных проверок
- ▶ Организуйте код в логические пространства имен
- ▶ Создавайте модули с четкими границами ответственности
- ▶ Используйте Swift Package Manager для управления зависимостями
- ▶ Экспортируйте только необходимые публичные API