Swift 快速参考

类型、Optional、Protocol、错误处理要点

基础

Hello World

```
import Foundation
print("Hello, World!")
```

常量与变量

```
let name = "Swift"       // constant (immutable)
var count = 0            // variable (mutable)
count += 1
let pi: Double = 3.14   // explicit type annotation
```

注释

```
// single-line comment
/* multi-line
   comment */
/// documentation comment (Markdown supported)
```

类型

基本类型

Int	平台大小的整数（现代系统为 64 位）
Double	64 位浮点数（优先于 Float）
Float	32 位浮点数
Bool	**true** / **false**
String	Unicode 字符串，值类型
Character	单个扩展字形簇

类型推断与转换

```
let score = 95              // inferred as Int
let gpa = 3.8               // inferred as Double
let total = Double(score) + gpa  // explicit conversion
let label = "Score: \(score)"    // string interpolation
```

元组

```
let point = (x: 3, y: 5)
print(point.x)           // named access
let (x, y) = point       // decompose
let (first, _) = point   // ignore second value
```

类型别名

```
typealias Coordinate = (Double, Double)
let origin: Coordinate = (0.0, 0.0)
```

控制流

If / Else

```
if score > 90 { print("A") }
else if score > 80 { print("B") }
else { print("C") }
```

Switch

```
switch grade {
case "A": print("excellent")
case "B", "C": print("passing")
default: print("unknown")
}
```

循环

```
for i in 0..<5 { }           // half-open range
for name in names { }         // collection
for (i, val) in list.enumerated() { }
while condition { }
repeat { } while condition    // do-while
```

Guard

```
func process(value: Int?) {
    guard let v = value, v > 0 else { return }
    print(v)   // v is unwrapped and in scope
}
```

函数

基础函数

```
func greet(name: String) -> String {
    return "Hello, \(name)!"
}
greet(name: "Alice")
```

参数标签

```
func move(from start: Int, to end: Int) -> Int {
    return end - start
}
move(from: 0, to: 10)  // external labels
func add(_ a: Int, _ b: Int) -> Int { a + b }
```

默认参数与可变参数

```
func join(_ items: String..., separator: String = ", ") -> String {
    items.joined(separator: separator)
}
join("a", "b", "c")
```

inout 参数

```
func double(_ x: inout Int) { x *= 2 }
var num = 5
double(&num)   // num is now 10
```

闭包

闭包语法

```
let double = { (x: Int) -> Int in return x * 2 }
let nums = [3, 1, 2]
let sorted = nums.sorted { $0 < $1 }
let mapped = nums.map { $0 * 10 }
```

尾随闭包

```
UIView.animate(withDuration: 0.3) {
    view.alpha = 0.0
}
```

捕获值

```
func makeCounter() -> () -> Int {
    var count = 0
    return { count += 1; return count }
}
let counter = makeCounter()  // counter() => 1, 2, ...
```

类与结构体

结构体（值类型）

```
struct Point {
    var x: Double
    var y: Double
}
var p = Point(x: 1, y: 2)  // auto memberwise init
```

类（引用类型）

```
class Vehicle {
    var speed: Double = 0
    init(speed: Double) { self.speed = speed }
}
class Car: Vehicle { var gear: Int = 1 }
```

结构体 vs 类

struct	值类型，赋值时复制，不支持继承
class	引用类型，按引用共享，支持继承
mutating	修改 **self** 的结构体方法必须标注
deinit	仅类支持的析构器（释放前调用）

Protocol

定义与遵从

```
protocol Drawable {
    var description: String { get }
    func draw()
}
struct Circle: Drawable { /* implement required members */ }
```

Protocol 扩展

```
extension Drawable {
    func log() { print("Drawing: \(description)") }
}
// all Drawable conformers get log() for free
```

常用 Protocol

Equatable	**==** 和 **!=** 比较
Comparable	**<**、**>**、**<=**、**>=** 排序
Hashable	可用作字典键或 Set 元素
Codable	Encodable + Decodable（JSON、Plist）
CustomStringConvertible	自定义 **description** 属性
Identifiable	需要 **id** 属性（SwiftUI）

Optional

声明 Optional

```
var name: String? = "Alice"  // may contain String or nil
var age: Int? = nil          // currently nil
let count: Int = 5           // non-optional, never nil
```

解包

```
if let n = name { print(n) }       // optional binding
guard let n = name else { return }  // guard
let n = name ?? "Unknown"           // nil coalescing
let n = name!                       // force unwrap (crashes if nil)
```

可选链

```
let count = user?.address?.zip?.count
// returns nil if any link in the chain is nil
user?.save()   // called only if user is non-nil
```

Optional map

```
let length = name.map { $0.count }   // Int?
let upper = name.flatMap { $0.isEmpty ? nil : $0.uppercased() }
```

Swift 快速参考

枚举

基础枚举

```
enum Direction {
    case north, south, east, west
}
var heading = Direction.north
heading = .east   // type inferred
```

关联值

```
enum Result {
    case success(data: String)
    case failure(code: Int, message: String)
}
if case .failure(let code, _) = r { print(code) }
```

原始值

```
enum Planet: Int {
    case mercury = 1, venus, earth, mars
}
let p = Planet(rawValue: 3)   // Optional(.earth)
print(Planet.earth.rawValue)  // 3
```

枚举方法

```
enum Suit: String, CaseIterable {
    case hearts, diamonds, clubs, spades
}
Suit.allCases.forEach { print($0.rawValue) }
```

错误处理

定义错误

```
enum NetworkError: Error {
    case badURL
    case timeout(seconds: Int)
    case serverError(code: Int)
}
```

抛出与捕获

```
func fetch(url: String) throws -> Data {
    guard url.hasPrefix("https") else { throw NetworkError.badURL }
    return Data()
}
do { let data = try fetch(url: "https://example.com") }
catch { print("Error: \(error)") }
```

try 变体

try	必须在 **do-catch** 内，向上传播错误
try?	返回 Optional，错误时为 **nil**
try!	强制 try，出错时崩溃
throws	函数可以抛出错误
rethrows	仅当闭包参数抛出时才抛出