Quick/Nimble常规使用指南

本文档介绍在 iOS 项目（含 SDK、私有库、组件库）中，如何使用 **Quick/Nimble** 进行 BDD 风格单元测试，提高可读性、可维护性，并增强测试覆盖度。

---

1. Quick/Nimble 简介

Quick：一个支持 BDD（行为驱动开发）风格的测试框架。
Nimble：断言库，让测试断言更自然、更可读。

BDD 结构非常清晰：

- `describe`：描述测试对象（类/模块）
- `context`：描述场景（前置条件）
- `it`：描述行为（单个测试条件）

示例：

代码块

```
describe("UserManager") {
    context("when login succeeds") {
        it("should save token") {
            expect(token).toNot(beNil())
        }
    }
}
```

这种结构比 XCTest 更具表达力，适用于公共库与复杂逻辑的单元测试。

---

2. 项目集成方式

2.1 SPM（推荐）

在 `Package.swift` 中加入：

代码块

```
.testTarget(
```

```
        name: "MyLibraryTests",
        dependencies: [
            "MyLibrary",
            "Quick",
            "Nimble"
        ]
    )
```

2.2 CocoaPods

代码块

```
target 'MyLibraryTests' do
  inherit! :search_paths
  pod 'Quick'
  pod 'Nimble'
end
```

---

3. 测试目录结构推荐

代码块

```
MyLibrary/
│
├── Sources/
│
└── Tests/
    ├── Specs/
    │   └── UserManagerSpec.swift
    ├── Fixtures/            # JSON、文本资源文件（可选）
```

---

4. Quick 常用结构

4.1 基础用法

代码块

```
describe("Calculator") {
    it("adds numbers") {
        let result = Calculator.add(1, 2)
        expect(result).to(equal(3))
    }
}
```

4.2 beforeEach / afterEach

测试初始化与清理逻辑：

代码块

```
describe("UserManager") {

    var manager: UserManager!

    beforeEach {
        manager = UserManager()
    }

    afterEach {
        manager = nil
    }

    it("runs with fresh instance") {
        expect(manager).toNot(beNil())
    }
}
```

4.3 多层 describe / context

代码块

```
describe("Validator") {

    context("when input is empty") {
        it("returns false") {
            expect(Validator.validate("")).to(beFalse())
        }
    }

```

```
    context("when input is valid") {
        it("returns true") {
            expect(Validator.validate("abc")).to(beTrue())
        }
    }
}
```

---

5. Nimble 常用断言

基础类型

代码块

```
expect(a).to(equal(b))
expect(a).toNot(equal(b))
expect(value).to(beNil())
expect(value).toNot(beNil())
```

Bool

代码块

```
expect(flag).to(beTrue())
expect(flag).to(beFalse())
```

数组 / 集合

代码块

```
expect(list).to(contain(3))
expect(list).to(haveCount(5))
```

类型检查

代码块

```
expect(model).to(beAKindOf(User.self))
```

抛错误

代码块

```
expect { try fn() }.to(throwError())
```

---

6. async/await 测试

Quick/Nimble 完全支持 async：

代码块

```
it("parses correctly") {
    let result = try await service.load()
    expect(result.name).to(equal("Alice"))
}
```

如果需要测试抛错：

代码块

```
it("throws error") {
    await expect {
        try await service.loadFail()
    }.to(throwError())
}
```

异步断言

代码块

```
it("waits for async result") {
    var done = false

    DispatchQueue.global().asyncAfter(deadline: .now() + 0.2) {
        done = true
    }

    expect(done).toEventually(beTrue(), timeout: .seconds(1))
}
```

---

7. 完整示例

业务类

代码块

```
final class Counter {
    private(set) var value: Int = 0

    func increase() {
        value += 1
    }

    func reset() {
        value = 0
    }
}
```

QuickSpec 测试

代码块

```
import Quick
import Nimble
@testable import MyLibrary

final class CounterSpec: QuickSpec {
    override func spec() {

        describe("Counter") {

            var counter: Counter!

            beforeEach {
                counter = Counter()
            }

            it("starts from zero") {
                expect(counter.value).to(equal(0))
            }

            it("increases value") {
                counter.increase()
                expect(counter.value).to(equal(1))
```

```
            }

            it("resets value") {
                counter.increase()
                counter.reset()
                expect(counter.value).to(equal(0))
            }
        }
    }
}
```

---

8. 最佳实践建议

✔ 一个 Spec 文件只测一个类

保持测试的聚焦性。

✔ describe → context → it 层级要清晰

减少阅读成本。

✔ 每个 it 只测一件事

行为越明确，越好维护。

✔ 避免依赖顺序

测试应该完全独立。

✔ 尽量使用 beforeEach 初始化对象

保障状态干净。

✔ 测试命名描述行为（英文即可）

例如：

- `"returns correct value"`
- `"throws error for invalid input"`