# 29-如何在iOS中进行面向测试驱动开发和面向行为驱动开发？

你好，我是戴铭。今天，我要和你分享的话题是，如何在 iOS 中进行面向测试驱动开发和面向行为驱动开发。

每当你编写完代码后，都会编译看看运行结果是否符合预期。如果这段代码的影响范围小，你很容易就能看出结果是否符合预期，而如果验证的结果是不符合预期，那么你就会检查刚才编写的代码是否有问题。

但是，如果这段代码的影响范围比较大，这时需要检查的地方就会非常多，相应地，人工检查的时间成本也会非常大。特别是团队成员多、工程代码量大时，判断这段代码的影响面都需要耗费很多时间。那么，每次编写完代码，先判断它的影响面，然后再手动编译进行检查的开发方式，效率就非常低了，会浪费大量时间。

虽说一般公司都会有专门的测试团队对产品进行大量测试，但是如果不能在开发阶段及时发现问题，当各团队代码集成到一起，把所有问题都堆积到测试阶段去发现、解决，就会浪费大量的沟通时间，不光是开发同学和测试同学之间的沟通时间，还有开发团队之间的沟通时间也会呈指数级增加。

那么，有没有什么好的开发方式，能够提高在编写代码后及时检验结果的效率呢？

所谓好的开发方式，就是开发、测试同步进行，尽早发现问题。从测试范围和开发模式的角度，我们还可以把这种开发模式细分出更多类型。

**从测试范围上来划分的话**，软件测试可以分为单元测试、集成测试、系统测试。测试团队负责的是集成测试以及系统测试，而单元测试则是有开发者负责的。对于开发者来说，通过单元测试就可以有效提高编写代码后快速发现问题的效率。

概括来说，单元测试，也叫作模块测试，就是对单一的功能代码进行测试。这个功能代码，可能是一个类的方法，也可能是一个模块的某个函数。

单元测试会使用 Mock 方式模拟外部使用，通过编写的各种测试用例去检验代码的功能是否正常。一个系统都是由各个功能组合而成，功能模块划分得越小，功能职责就越清晰。清晰的功能职责可以确保单个功能的测试不会出现问题，是单元测试的基础。

**从开发模式划分的话，**开发方式可以分为 TDD（Test-driven development，面向测试驱动开发）和 BDD（Behavior-driven development，面向行为驱动开发）。

- TDD 的开发思路是，先编写测试用例，然后在不考虑代码优化的情况下快速编写功能实现代码，等功能开发完成后，在测试用例的保障下，再进行代码重构，以提高代码质量。
- BDD 是 TDD 的进化，基于行为进行功能测试，使用 DSL（Domain Specific Language，领域特定语言）来描述测试用例，让测试用例看起来和文档一样，更易读、更好维护。

TDD 编写的测试用例主要针对的是开发中最小单元进行测试，适合单元测试。而 BDD 的测试用例是对行为的描述，测试范围可以更大一些，在集成测试和系统测试时都可以使用。同时，不仅开发者可以使用BDD的测试用例高效地发现问题，测试团队也能够很容易参与编写。这，都得益于 BDD 可以使用易于编写行为功能测试的 DSL 语言。

接下来，我就和你详细聊聊 TDD 和 BDD。

## TDD

我刚刚也已经提到了，TDD在确定功能需求后，首先就会开始编写测试用例，用来检验每次的代码更新，能够让我们更快地发现问题，并能保正不会漏掉问题。其实，这就是通过测试用例来推动开发。

在思想上，和拿到功能需求后直接开发功能的区别是，TDD会先考虑如何对功能进行测试，然后再去考虑如何编写代码，这就给优化代码提供了更多的时间和空间，即使几个版本过后再来优化，只要能够通过先前写好的测试用例，就能够保证代码质量。

所以说，TDD 非常适合快速迭代的节奏，先尽快实现功能，然后再进行重构和优化。如果我们不使用 TDD 来进行快速迭代开发，虽然在最开始的时候开发效率会比 TDD 高，但是过几个版本再进行功能更新时，就需要在功能验证上花费大量的时间，反而得不偿失。

其实，TDD 这种开发模式和画漫画的工作方式非常类似：草稿就类似 TDD 中的测试用例，漫画家先画草稿，细节由漫画家和助手一起完成，无论助手怎么换，有了草稿的保障，内容都不会有偏差。分镜的草稿没有细节，人物眼睛、鼻子都可能没有，场景也只需要几条透视线就可以。虽然没有细节，但是草稿基本就确定了漫画完成后要表达的所有内容。

## BDD

相比 TDD，BDD更关注的是行为方式的设计，通过对行为的描述来验证功能的可用性。行为描述使用的 DSL，规范、标准而且可读性高，可以当作文档来使用。

BDD 的 Objective-C 框架有 Kiwi、Specta、Expecta等，Swift 框架有 Quick。

Kiwi框架不光有 Specta 的 DSL 模式，Expecta框架的期望语法，还有 Mocks 和 Stubs 这样的模拟存根能力。所以接下来，我就跟你说说这个iOS中非常有名并且好用的BDD框架，以及怎么用它来进行 BDD 开发。

## Kiwi

将Kiwi集成到你的App里，只需要在 Podfile 里添加 pod ‘Kiwi’ 即可。下面这段代码，是 Kiwi 的使用示例：

```
// describe 表示要测试的对象
describe(@"RSSListViewController", ^{
    // context 表示的是不同场景下的行为
    context(@"when get RSS data", ^{
        // 同一个 context 下每个 it 调用之前会调用一次 beforeEach
        beforeEach(^{
            id dataStore = [DataStore new];
        });


        // it 表示测试内容，一个 context 可以有多个 it
        it(@"load data", ^{
            // Kiwi 使用链式调用，should 表示一个期待，用来验证对象行为是否满足期望
            [[theValue(dataStore.count) shouldNot] beNil];
        });
    });
```

```
});
```

上面这代码描述的是在 RSS 列表页面，当获取 RSS 数据时去读取数据这个行为的测试用例。这段测试用例代码，包含了 Kiwi 的基本元素，也就是describe、context、it。这些元素间的关系可以表述为：

- describe 表示要测试的对象，context 表示的是不同场景下的行为，一个 describe 里可以包含多个 context。
- it表示的是需要测试的内容，同一个场景下的行为会有多个需要测试的内容，也就是说一个 context 下可以有多个 it。

测试内容使用的是 Kiwi 的 DSL 语法，采用的是链式调用。上面示例代码中 shouldNot 是期望语法，期望是用来验证对象行为是否满足期望。

期望语法可以是期望数值和数字，也可以是期望字符串的匹配，比如：

```
[[string should] containString:@"rss"];
```

should containString 语法表示的是，期望 string 包含了 rss 字符串。Kiwi 里的期望语法非常丰富，还有正则表达式匹配、数量变化、对象测试、集合、交互和消息、通知、异步调用、异常等。完整的期望语法描述，你可以查看Wiki的 Expectations 部分。

除了期望语法外，Kiwi 还支持模拟对象和存根语法。

模拟对象能够降低对象之间的依赖，可以模拟难以出现的情况。模拟对象包含了模拟 Null 对象、模拟类的实例、模拟协议的实例等。存根可以返回指定选择器或消息模式的请求，可以存根对象和模拟对象。

模拟对象和存根的详细语法定义，你可以查看Wiki 的 Mocks and Stubs 部分。

## 小结

按照 TDD 和 BDD 方式开发，有助于更好地进行模块化设计，划清模块边界，让代码更容易维护。TDD 在测试用例的保障下更容易进行代码重构优化，减少 debug 时间。而使用 BDD 编写的测试用例，则更是好的文档，可读性非常强。通过这些测试用例，在修改代码时，我们能够更方便地了解开发 App 的工作状态。同时，修改完代码后还能够快速全面地测试验证问题。

无论是 TDD 还是 BDD，开发中对于每个实现的方法都要编写测试用例，而且要注意先编写测试用例代码，再编写方法实现代码。测试用例需要考虑到各种异常条件，以及输入输出的边界。编写完测试用例还需要检查如果输入为错时，测试用例是否会显示为错。

最后需要强调一点，好的模块化架构和 TDD 、BDD 是相辅相成的。TDD 和 BDD 开发方式会让你的代码更加模块化，而模块化的架构更容易使用 TDD 和 BDD 的方式进行开发。

在团队中推行 TDD 和 BDD 的最大困难，就是业务迭代太快时，没有时间去写测试用例。我的建议是，优先

对基础能力的功能开发使用 TDD 和 BDD，保证了基础能力的稳定，业务怎么变，底子还都是稳固的；当有了业务迭代、有了间隙时，再考虑在核心业务上采用 BDD，最大程度的保证 App 核心功能的稳定。

### 课后作业

今天我跟你聊了很多 TDD 和 BDD 的优点，但是很多团队并没有使用这样的开发方式，你觉得这其中的原因是什么呢？

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎把它分享给更多的朋友一起阅读。



### 精选留言：

- 时间都去哪了 2019-05-16 10:25:29
  TDD,OC有什么好用的推荐框架吗? [2赞]

- 不知名的iOS网友 2019-05-16 16:18:55
  课程一些笔记：https://github.com/CrusherWu/iOSRoadMap [1赞]

- Trust me ❀ 2019-05-16 09:35:44
  正在手淘推动BDD😄 [1赞]

- Geek_9d618c 2019-05-16 08:13:44
  没有采用TDD或BDD的原因： 一来很多业务迭代的比较快，没有时间是一个原因。 二来，能够TDD是建立在编写TDD的场景足够，也就是能模拟细粒度模块的外围环境，对于小项目而言想要的往往就是快速出产品，一开始就关注细粒度模块化的很少，对于大项目，受历史原因业务之间的强耦合导致很难去构建Mock场景。 挺赞同从基础模块和对外的 SDK 结合业务的发展去编写TDD可能更合适。
  [1赞]

- yasuoyuhao 2019-05-16 07:53:10
  很多情況都是一開始都想說先實現幾項功能再補測試，然後就一直補一直寫，要避免這樣的情況最好使用 TDD / BDD 同步測試代碼，確保整體質量，也管理測試用例。也為 CI 做好充足的準備。