20 单元测试的知识储备

更新时间：2020-06-03 15:43:23

> 人只有献身于社会，才能找出那短暂而有风险的生命的意义。——爱因斯坦

1. 前言

单元测试作为编码质量的重要保障手段，是编码的一个非常重要的环节。

《手册》 第三部分对单元测试进行了描述 1，包括：单元测试必须遵守自动化、独立性和可重复性原则；单元测试的粒度一般是方法级别，最多也是类级别；核心业务、核心应用。核心模块的增量代码确保单元测试通过等。

那么接下来我们思考几个问题：

什么是单元测试？

为什么要编写单元测试？

什么是好的单元测试？

单元测试的常用框架有哪些？

这些都是本节将探讨的重点内容。

2. 单元测试相关概念

2.1 单元测试和集成测试

很多人一直在写单元测试，却不知道单元测试和集成测试的区别，认为：“使用能够编写单元测试的框架编写的测试就是单元测试”，这种认识是不全面的。

接下来，我们先了解单元测试和集成测试的概念和主要区别。

《单元测试的艺术》 第一章 单元测试的基础对集成测试和单元测试进行了描述 2：

单元测试：

> 一个单元测试是一段代码，这段代码调用一个工作单元，并检验该工作单元的一个具体的最终结果。
>
> 如果关于这个最终结果的假设是错误的，单元测试就失败了。
>
> 一个单元测试的范围可以小到一个方法，大到一个类。

集成测试：

> “任何测试，如果它运行速度不快，结果不稳定，或者要用到被测试单元的一个或者多个真实依赖，我就认为它是集成测试。”
>
> 集成测试是对一个工作单元进行测试，这个测试对被测试的工作单元没有完全控制，并使用单元的一个或多个真实依赖物，例如时间、网络、数据库、线程或随机数产生器等。

两者的主要联系和区别：

集成测试和单元测试同样都很重要，都是验证系统功能的重要手段。

但是集成测试运行通常更慢，很难编写，很难做到自动化，需要配置，通常一次测试的东西过多。

集成测试会使用真实的依赖，而单元测试则把被测试的单元和其依赖隔离，以保证单元测试的高度稳定，还可以轻易控制和模拟被测试单元的行为方面。

因此**单元测试和集成测试最主要的区别之一就是测试中是否依赖“真实环境”**。

2.2 单元测试的重要性

很多人经常以 “时间紧，任务重” 或者 “单元测试没用” 为借口来拒绝编写单元测试。

但是 BUG 在软件的生命周期越早阶段发现，付出的代价越少。

单元测试可以让很多 BUG 在编码阶段就能够及时发现并解决，而不需要交给测试人员兜底，如果测试人员兜底失败，可能造成线上故障。

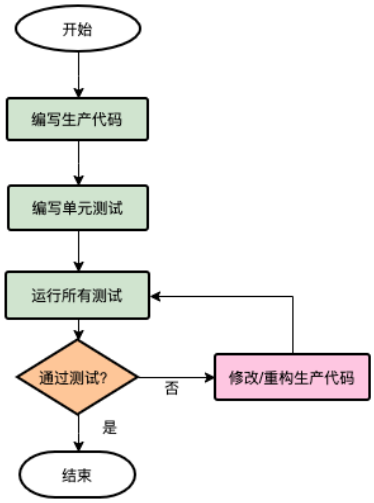
有了单元测试作保障，我们还可以放心对函数进行重构，如果重构代码导致单元测试运行失败，则说明重构的代码有问题。

长远来看，单元测试对编码的益处（如提高代码质量和避免 BUG）远比编写单元测试的投入所花费的代价要大的多。

2.3 单元测试的方法

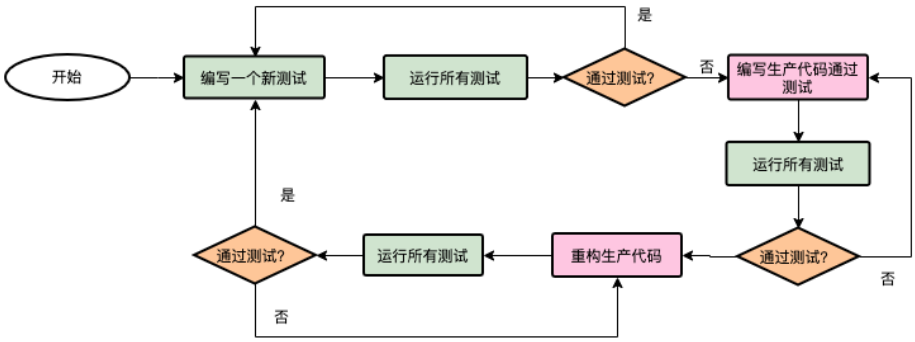
从我个人的理解来看，编写单元测试通常有两种方法，包括传统的单测方法和测试驱动开发，两种略有不同。

单元测试的传统方法大致流程如下：



在国内，很多团队采用传统的方式，即先编写好代码，然后再编写单元测试来验证该段代码是否正确。

也有一些团队采用如下图所示的方式，即先编写单元测试，然后编写代码让通过测试。这种开发方式被称为**测试驱动开发（Test-Driven Development, TDD）**。



TDD 体现了 “以终为始” 的思想，即先制定目标，然后去验证是否实现了目标，而不是先做再去 “思考目标”。

实践 TDD 的关键步骤

1. 编写一个失败的测试来证明产品中代码和功能的缺陷；
2. 编写符合测试预期的产品代码，使测试通过；
3. 重构代码。

虽然 TDD 更 “出名”，具体采用哪种方法主要看团队约定和个人编程习惯。

由于团队没有强制约定，或者开发前参数不容易确定等原因，传统的单元测试方法依然被很多开发人员采用。

2.4 何为优秀的单元测试？

既然要编写单元测试，那么好的单元测试标准是什么呢？

参考众多单元测试相关资料，我们得总结出优秀的单元测试应该具有以下几个特征：

> 满足功能：被检验的函数或类的逻辑行为满足预期功能；
>
> 满足 AIR 原则：单元测试应该可以自动执行；单元测试的用例之间要保持彼此独立；单元测试可以重复执行。
>
> 优秀的单元测试还应该具有编写容易，运行快速的特点。

在学习和开发过程中，看到很多人依然通过打印语句输出结果，通过“肉眼”来测试，这样如果对被测试的类或函数做出修改而无法满足功能要求，单元测试也会运行通过，就失去了单元测试的意义。

因此，建议大家在学习和工作开发过程中要遵循上述指导原则，编写出优秀的单元测试。

3. Java 单元测试工具

常用的 Java 单元测试有: JUnit、TestNG。

TestNG 受 JUnit 和 NUnit 的启发，功能相似，但是比 JUnit 更强大。TestNG 不只为单元测试而设计，其框架的设计目标是支持单元测试、公共能测试，端对端测试，集成测试等。

JUnit 具体用法比较简单，如果想系统学习可官方使用指南，参考《JUnit 实战 (第 2 版)》, TestNG 和 JUnit 非常相似，如果想深入学习，首推 TestNG 官方文档。

主流的 Java mock 框架有: Mockito, JMockit, Easy Mock 。

根据《What are the best mock framteworks for Java》一文 3，我们可以看到三者的特点和优劣。

Mockito 简洁易用，有 PowerMock 拓展，允许静态函数测试，社区强大，对结果的验证和异常处理非常简洁、灵活。缺点是框架本身不支持 static 和 private 函数的 mock。

JMockit 简单易用；可以 mock “一切”，包括 final 类， final/private/static 函数，而其他 mock 框架往往只支持其中一部分；缺点是社区支持不够活跃，3 个 contributers 介乎只有一个在干活，学习曲线比较陡峭。

Easy Mock 上手简单，文档清晰；同样的社区较小，导致更多人选择其它的 mock 框架。

还有很多其他配合单元测试的框架，如强大的构造随机 Java 对象的 Easy Random ，构造随机字符串的 Java Faker 等。

4. 总结

本节主要介绍了单元测试的概念、单元测试和集成测试的区别以及单元测试的必要性、主要步骤、主要框架等。

希望通过本节的学习，大家对单元测试能够有一个初步的理解。

下一节我们将介绍单元测试的范例。

参考资料

阿里巴巴与 Java 社区开发者.《 Java 开发手册 1.5.0》. 华山版 .2019 □□

[以] Roy Osherove. 《单元测试的艺术》[M]. [译] 金迎。人民邮电出版社.2014 □□

《What are the best mock framteworks for Java》 □□

}

19 日志学习和使用的正确姿势

21 单元测试构造数据的正确姿势