

项目 3

代码级接口测试自动化

项目简介

被测程序(ArrayCompare)实现如下功能。

输入一个以逗号(或其他字符)分隔的字符串,程序将解析该字符串并得到一个数组。以同样的方式输入第二个字符串,并解析成数组。

对输入的字符分隔的每一个值进行判断,必须为数值类型,否则程序将不做任何处理。

如果输入合法,将按如下顺序进行判断。

- (1) 如果数组长度为零,将直接输出信息“结果: 数组长度为零.”。
- (2) 如果两个数组长度不相等,将直接输出信息“结果: 数组长度不一致.”。
- (3) 如果两个数组不经过任何排序,自然相等,输出信息为“结果: 数组相同.”。
- (4) 如果两个数组经过排序后比较,是相等的,输出信息为“结果: 数组排序后相同.”。
- (5) 如果两个数组经过排序后比较,不相等,输出信息为“结果: 数组不同.”。

程序不需要设计专门的 GUI 界面,直接使用命令行即可。

☆代码路径: \bookworkspace\CodeTest

项目展示

实验结果如图 3-1 所示。

The screenshot shows a Windows Command Prompt window titled '命令提示符'. The command entered is 'java com.woniuxy.compare/MainRun'. The window displays two sets of user inputs and their corresponding results:

```
D:\Workspace\bookworkspace\CodeTest\bin>java com.woniuxy.compare/MainRun
请输入一个字符串:
11,22,33,44,55
请输入一个字符串:
11,22,33,44
结果: 数组长度不一致.

D:\Workspace\bookworkspace\CodeTest\bin>java com.woniuxy.compare/MainRun
请输入一个字符串:
11,22,33,44
请输入一个字符串:
22,33,44,11
结果: 数组排序后相同.

D:\Workspace\bookworkspace\CodeTest\bin>
```

图 3-1 数组判断

项目目标

- (1) 理解路径覆盖和条件覆盖对测试用例设计的指导价值。

- (2) 熟练运用基于代码级的接口测试自动化技术。
- (3) 对代码级自动化测试框架 JUnit 和 TestNG 有深入理解。
- (4) 理解代码覆盖率对代码级自动测试的价值。
- (5) 深入理解并熟练运用代码级自动化测试技术,突破自动化测试技术难题。

3.1 预备知识: 深入理解接口测试及白盒测试

实验简介

随着移动互联网的触角深入人们生活的每个角落,伴随而来的便是企业不断对其软件系统接口定义和研发,以便于进行数据传输和交换。由此导致目前企业急需大量专业接口测试工程师,因为接口测试天然具备自动化测试的可行性。所以本项目专门介绍接口测试的各种存在形式。

实验目的

- (1) 理解接口测试的含义与作用。
- (2) 理解白盒测试与灰盒测试。
- (3) 理解代码级接口测试与协议级接口测试。

实验流程

1. 关于接口测试

接口测试是一个比较宽泛的概念,近几年在国内受到很多企业和测试从业者的追捧。事实上,无论通过何种方式运行一段程序,人们都必须依赖于调用该程序的接口才能实现。

比如,假设现在用户通过登录界面输入用户名和密码,单击“登录”按钮,最终该操作会被组装为一个 HTTP 请求进而发送给后台服务器端,后台服务器则会直接调用实现登录的接口,通过该接口运行登录的实际代码,如图 3-2 所示。

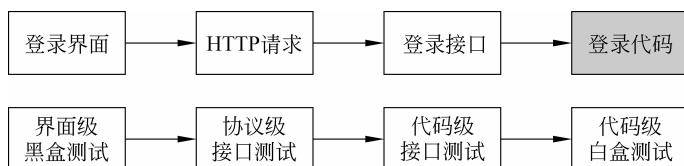


图 3-2 通过接口运行登录的实际代码

在这个过程中,单击“登录”按钮动作发生在前端界面,如果通过该方法观察其最终运行状态,就称之为界面级的黑盒测试。人们也可以利用各种协议发送工具甚至用代码发送协议数据包给后台服务器进而观察其运行状态,此时,就称之为协议级的接口测试。当然,人们也可以从代码层面直接调用该登录接口,这样就称之为代码级的接口测试。最后,人们还可以直接深入代码实现层,对代码的实现逻辑进行详细测试,这样就称之为白盒测试或单元测试。

那么问题来了,这整个过程唯一的区别仅在于人们调用该登录代码的方式不一样,最终

真正工作的都是同一段代码。所以,任何一种调用的方式,都是在驱动程序运行而已。从本质上来说,人们所做的事情没有任何区别,那么为什么还要基于界面测试登录的功能呢?这里完全可以基于其接口进行测试达到同样的目的,而且这样的做法都是可以自动化的,也是可以重用的。这也是为什么接口测试越来越受到重视的原因。

当然,也正是因为接口测试的“接口”,是一个不太容易下定义的概念,所以读者千万不能盲目地认为协议级的测试就是接口测试,或者代码级的测试就是接口测试。

2. 关于白盒测试

本书在前面将自动化测试进行了归类:代码级、协议级和界面级。本书所讲的白盒测试,统指代码级的测试。白盒测试是相对于黑盒测试而言的一种测试方法而已,是指人们可以基于系统的代码层逻辑实现非常有针对性的测试,其参考文档主要是系统的详细设计文档,甚至可以精确到算法层实现,也可以向上提升到代码接口层。

白盒测试的核心就是利用程序测试程序(如某个函数或方法),所以白盒测试默认情况下自带自动化测试属性。从接口测试的定义来看,白盒测试自然也是通过调用其函数或方法的接口才能完成测试的执行。所以,本书后续所介绍的白盒测试,均是指基于代码级接口实现的测试,既关注接口规范,也关注代码的逻辑实现。

白盒测试既然天然属于自动化测试,那么可以将白盒测试、灰盒测试和黑盒测试有效地结合起来,各自完成不同的测试。比如利用白盒测试完成对基本功能点的测试或部分性能测试;利用灰盒测试如协议级测试完成可靠性、安全性、性能等测试工作;利用黑盒测试完成用户体验、兼容性方面的测试。通常情况下,这样的组合会让人们的测试工作变得更加有效率。建议读者不妨在工作当中尝试这样的组合。

3. 代码级接口测试实施价值

代码级测试在预防软件产品的 Bug 方面是非常有效的。如果将其发挥好,从组织和技术层面做好协调和规范,其价值不容小觑。简单将其特点归纳为如下几点。

(1) 容易上手。只要对代码有一定了解,都可以轻易完成针对代码的专项测试。即使是一个没有受过正统程序设计培训的人,也可以比较容易地按照标准流程和模板完成测试脚本的开发和测试数据的准备。

(2) 容易实施。由于代码级测试工作直接使用测试代码调用被测代码(通过其开放出来的接口进行调用),所以实施过程非常容易。只需要通过简单的判断就可以确定该项测试是通过还是失败。

(3) 容易维护。通常情况下,在软件研发的过程中,一旦代码的接口确定,变动相对较少,所以维护该测试脚本的工作量相对较低,测试脚本也会比较稳定。

(4) 容易见效。一旦将代码级测试工作实施起来,可以立即看到测试脚本所产生的效果。

(5) 容易定位。由于测试脚本直接调用被测代码,所以一旦测试脚本无法运行通过,很容易定位该问题,同样,修复该问题的成本也会更小。

(6) 增强质量意识。事实上,很多企业的代码级测试工作会由测试团队和程序员团队共同负责,这将非常有助于程序员团队在编写代码时增强其代码的质量意识,为全员质量意识打下坚实的基础。

4. 代码级接口测试实施难题

既然代码级测试这么有价值,为什么现在很多企业还是做不好呢?当然,这当中是有技术的原因,但更重要的是人为的原因。笔者就这些年的经验,总结一下为什么很难将代码级测试实施的原因。

(1) 程序员不习惯。程序员习惯了直接上手写代码,并没有养成写代码之前或之后还要再写测试代码的习惯。

(2) 测试人员编码能力差。很多测试工程师在编码方面的能力的确不敢恭维,而且国内存在着这样一种现象就是写不好代码的人才去做软件测试。

(3) 程序接口不规范。代码级测试能够实施好,需要一个规范的设计文档和接口说明,甚至清晰的算法实现。但是在很多研发团队中,能把客户的需求分析清晰、形成文档已经不易,根本没有时间设计接口规范。所以程序编写的过程就是一个打补丁的过程,导致代码级测试工作很难实施。

(4) 利用调试代替测试。程序员团队都会将自己的代码进行调试,为了能尽早发现程序中可能存在的 Bug。这本身并没有错,错的是误认为这就是测试。事实上,调试是单次的、随机的行为,不具备可重用性,比如程序员每一次调试输入的数据可能都是随机的,而且这个数据很有可能没有很好地覆盖代码逻辑。而代码级测试则是严谨的、可重复运行的。无论程序怎么修改,只要能顺利通过代码级测试,都是可以接受的,除非测试程序有 Bug。

(5) 项目时间紧迫。这应该是每个团队都会遇到的问题。由于时间紧迫,团队没有时间完整地测试;由于时间紧迫,团队没有时间写测试脚本。但是这样真的可以提高效率吗?无数失败或延期的项目经验告诉人们,如果不具备规范和严谨的流程,以及全员质量意识,即使项目赶出来了,客户也不一定认可。

(6) 只关心用户看得到的东西。有的团队只做黑盒测试,以为将用户的操作过程模拟好就行,这样用户就不会发现问题了。但是,实际上,往往用户什么问题都可以发现,所以千万不要抱有侥幸心理。

(7) 测试程序复杂度高。有时候程序员为了能调用一个被测代码,需要准备大量的测试环境和测试数据,这是代码级测试经常遇到的问题。事实上,这个问题需要辩证地来看待,针对测试环境非常复杂的情况,无论白盒、黑盒,可能测试起来都会比较复杂。通常笔者会建议代码级测试更多关注在代码的算法层或逻辑实现层(即层次更低的代码)。而协议级或界面级的测试,更多关注于控制层代码。

事实上,笔者并不是在说代码级测试多么完美,多么有价值,但是一定不能无视它的存在。如果研发团队能把代码级测试的工作多做一些,系统测试的工作就可以少做一些,而且根据缺陷放大模型,把问题扼杀在摇篮中,更能看到其价值。

思考练习

- (1) 请理解单元测试与白盒测试、代码级接口测试与灰盒测试在作用上的区别。
- (2) 自学白盒测试中的路径覆盖、条件覆盖等测试用例设计方法。

3.2 核心实验

3.2.1 实现被测程序 ArrayCompare 代码

实验简介

本实验主要为读者梳理本测试项目的目标程序 ArrayCompare 的功能结构以及具体的代码实现。

实验目的

- (1) 掌握概要设计与接口设计的基本方法。
- (2) 能够熟练快速地定位和梳理代码逻辑。
- (3) 对代码功能进行解读,进而达到测试的目的。

实验流程

1. 代码结构图

ArrayCompare 代码结构如图 3-3 所示。

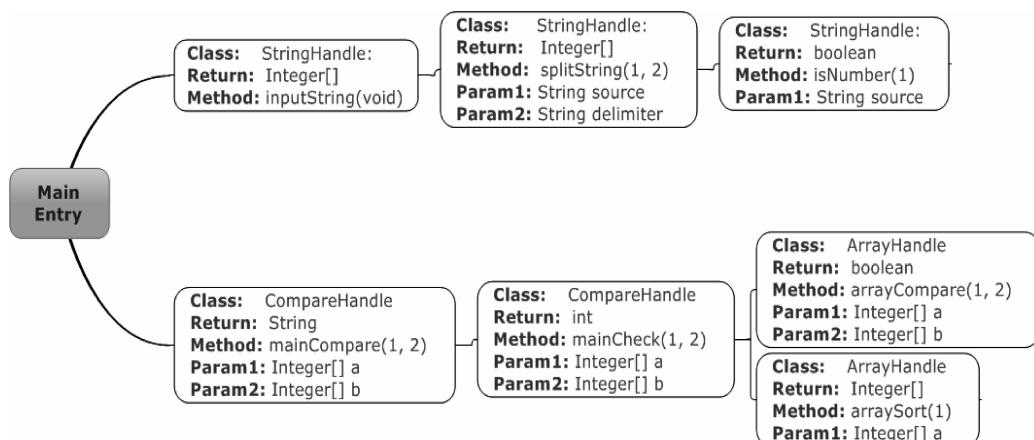


图 3-3 ArrayCompare 代码结构图

2. 代码实现原则

本书将基于如下思路进行程序的整体架构和接口设计。

- (1) 合理使用面向对象特性,保证可测试性和模块的独立性。
- (2) 高重用,低耦合。
- (3) 结构合理,方法之间的调用深度不超过 5 层。
- (4) 命名规范合理,见名知意。
- (5) 不使用多态性,不过度使用面向对象设计原则。
- (6) 不使用太多内置 API,尽量自主实现所有功能。