Broadview®
www.broadview.com.cn

全栈自动化测试实战

基于TestNG、HttpClient、Selenium和Appium

卢家涛 / 编著



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

全栈自动化测试实战

基于TestNG、HttpClient、Selenium和Appium

卢家涛 / 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

TestNG 作为 Java 中举足轻重的测试框架，除可以替代 JUnit 进行单元自动化测试外，还可以通过集成 Selenium、Appium 和 HttpClient 等框架做各种类型的自动化测试。本书首先对自动化测试进行了概述，接着对 TestNG 的语法进行了讲解，核心部分是使用 TestNG 进行单元自动化测试、接口自动化测试和界面自动化测试，最后介绍了持续集成、Mock 测试和代码覆盖率等扩展知识。

本书适合测试工程师、自动化测试工程师和测试管理者阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

全栈自动化测试实战：基于 TestNG、HttpClient、Selenium 和 Appium / 卢家涛编著. —北京：电子工业出版社，2020.3

ISBN 978-7-121-38382-3

Ⅰ. ①全… Ⅱ. ①卢… Ⅲ. ①软件工具－自动检测 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2020)第 021845 号

责任编辑：安　娜
印　　刷：山东华立印务有限公司
装　　订：山东华立印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　印张：15.75　字数：318.8 千字
版　　次：2020 年 3 月第 1 版
印　　次：2020 年 3 月第 1 次印刷
定　　价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前 言

写作背景

2011 年我接触了第一个自动化测试工具 DeviceAnywhere，该工具将真实手机置于云端，通过录制回放执行自动化测试脚本，最终采用图片对比技术实现断言。自此之后，我对自动化测试技术产生了浓厚的兴趣，在之后的 8 年多时间里，我不断学习自动化测试技术，并把它落实到实际项目中。

为什么是 TestNG，而不是 JUnit?

JUnit 主要用于单元测试，而 TestNG 在 JUnit 的基础上做了很多改进，更适合做全栈自动化测试的整体框架。

本书内容

本书的写作顺序不是由浅入深的，而是“自底向上”的，即按照单元自动化测试、接口自动化测试和界面自动化测试的顺序编写的。

本书首先对自动化测试进行了概述（第 1 章）；

接着对 Java 和 TestNG 语法进行了讲解（第 2 章和第 3 章）；

核心部分是使用 TestNG 进行单元自动化测试、接口自动化测试和界面自动化测试（第 4 章至第 10 章）；

最后介绍了持续集成、Mock 测试和代码覆盖率等扩展知识（第 11 章）。

对于代码基础比较薄弱的读者，建议在阅读完前三章内容后，先阅读较为简单的第 5、7、8 章，再阅读第 4、6、9、10、11 章。

适合读者

测试工程师：本书可以从简单的 Web 自动化测试（第 7 章）带你走进自动化测试的大门。

自动化测试工程师：打通各个级别的自动化测试壁垒，帮助读者全面掌握单元自动化测试、接口自动化测试和界面自动化测试。

测试管理者：了解主流自动化测试技术，指导团队开展自动化测试，把握自动化测试的方向和目标。

致谢

感谢老婆的理解和支持，让我能全身心地编写本书。

感谢老大（陈恒骥）给我进入软件测试行业的机会，没有这个机会，我不可能在这个行业走得这么远。

感谢编辑安娜在本书出版过程中耐心的指导。

虽然书中的每个技术点都曾在实际项目中实践过，但由于时间仓促难免有误，敬请读者批评指正。

笔者的 GitHub：https://github.com/lujiatao2/httpinterface/releases。

卢家涛

2019 年 12 月 20 日

目 录

第 1 章 自动化测试概述 1

1.1 自动化测试定义和分类 1
1.2 自动化测试的目的 3
1.2.1 提高软件质量 3
1.2.2 提高测试效率 5
1.3 自动化测试实施三要素 6
1.3.1 有明确的目标 6
1.3.2 有足够的资源 7
1.3.3 有合理的计划 8

第 2 章 TestNG 语法 12

2.1 TestNG 简介 12
2.2 测试前的准备工作 12
2.2.1 配置 Maven 12
2.2.2 创建工程 14
2.2.3 测试执行 16
2.3 TestNG 注解 19
2.3.1 前置条件和后置条件 20
2.3.2 数据驱动 24
2.3.3 测试用例 25
2.4 testng.xml 28
2.4.1 <package> 28

2.4.2 <include>和<exclude> 30
2.4.3 <parameter>标签 32
第 3 章 单元自动化测试 34
3.1 编写待测程序 34
3.2 手工测试用例设计 35
3.2.1 分析待测程序 35
3.2.2 测试用例设计 35
3.3 设计自动化测试用例 36
3.3.1 基于 JUnit 设计自动化测试用例 36
3.3.2 基于 TestNG 设计自动化测试用例 38
3.4 Spring 的单元自动化测试 42
3.4.1 Java 企业级应用简介 42
3.4.2 编写待测程序 44
3.4.3 单元自动化测试 50
第 4 章 HTTP 接口自动化测试 56
4.1 HTTP 简介 56
4.2 部署待测程序 57
4.3 手工测试用例设计 58
4.3.1 分析待测接口 58
4.3.2 测试用例设计 60
4.4 HttpClient 用法 62
4.4.1 HttpClient 简介 62
4.4.2 创建工程 63
4.4.3 发送 HTTP 请求 64
4.4.4 处理服务器响应 66
4.4.5 设置请求头 68
4.5 TestNG 集成 HttpClient 68
4.5.1 RESTful 接口自动化测试 69
4.5.2 SOAP 接口自动化测试 81
第 5 章 RPC 接口自动化测试 87
5.1 RPC 简介 87
5.2 部署待测程序 88
5.3 手工测试用例设计 90

5.3.1 分析待测接口 90
5.3.2 测试用例设计 91
5.4 TestNG Dubbo 接口自动化测试 92
5.4.1 基于 XML 方式 94
5.4.2 基于 API 方式 99
5.4.3 基于注解方式 100
5.4.4 泛化调用 102

第 6 章 Web 自动化测试 107

6.1 Web 自动化测试工具（框架）简介 107
6.2 部署待测程序 108
6.3 Selenium 用法 109
6.3.1 准备 109
6.3.2 元素操作 110
6.3.3 鼠标事件 115
6.3.4 键盘事件 118
6.3.5 浏览器操作 119
6.3.6 JavaScript 对话框处理 121
6.3.7 等待处理 124
6.4 TestNG 集成 Selenium 129

第 7 章 Android 自动化测试 133

7.1 Android 自动化测试工具（框架）简介 133
7.2 安装待测应用 134
7.3 Appium 用法 134
7.3.1 准备 134
7.3.2 初始化参数 136
7.3.3 元素操作 139
7.3.4 应用操作 145
7.3.5 系统操作 146
7.3.6 使用 Android 模拟器 147
7.4 TestNG 集成 Appium 148

第 8 章 iOS 自动化测试 151

8.1 iOS 自动化测试工具（框架）简介 151
8.2 待测应用开发 151

8.2.1 工程创建 152
8.2.2 界面开发 152
8.2.3 逻辑开发 155
8.3 Appium 的用法 159
8.3.1 准备 159
8.3.2 初始化参数 160
8.3.3 元素操作 161
8.3.4 应用操作 167
8.3.5 系统操作 169
8.3.6 使用 iOS 模拟器 169
8.4 TestNG 集成 Appium 171
第 9 章 自动化测试实战 174
9.1 实战项目部署安装 174
9.1.1 JForum 论坛部署 174
9.1.2 AnExplorer 文件管理器安装 177
9.2 Web 自动化测试实战 177
9.2.1 分层和解耦 177
9.2.2 公共函数和业务函数封装 180
9.2.3 自动化测试用例编写 186
9.2.4 测试数据准备 189
9.3 Android 自动化测试实战 192
9.3.1 工程准备 192
9.3.2 Page Object 设计模式 193
9.3.3 页面对象层封装 195
9.3.4 业务逻辑层封装 203
9.3.5 自动化测试用例编写 204
9.4 进一步优化 207
第 10 章 持续集成 208
10.1 持续集成、持续交付和持续部署 208
10.2 Jenkins 的重要功能简介 209
10.2.1 Jenkins 部署 209
10.2.2 任务管理 211
10.2.3 构建管理 212
10.2.4 节点管理 213

10.2.5 插件管理 213
10.2.6 用户管理 215
10.3 TestNG 集成到 Jenkins 216
10.3.1 TestNG 工程创建 216
10.3.2 SVN 部署及使用 218
10.3.3 JDK 和 Maven 配置 220
10.3.4 把 TestNG 集成到 Jenkins 221

第 11 章 Mock 测试和代码覆盖率 225

11.1 单元 Mock 测试 225
11.1.1 单元 Mock 测试简介 225
11.2.2 Mockito 用法 225
11.2 接口 Mock 测试 230
11.2.1 接口 Mock 测试简介 230
11.2.2 RAP2 用法 230
11.3 代码覆盖率简介 234
11.4 JaCoCo 用法 235
11.4.1 JaCoCo 计数器 235
11.4.2 使用 EclEmma 插件 236
11.4.3 Maven 集成 JaCoCo 239

第1章 自动化测试概述

1.1 自动化测试的定义和分类

自动化测试通俗地讲就是使用软件 A 自动执行测试用例来测试软件 B。软件 A 既可以是现有的自动化测试工具，也可以是自己编写的测试脚本。软件 B 就是待测试软件。

自动化测试的分类维度如下。

1．根据测试阶段划分

（1）单元自动化测试

单元自动化测试是指通过自动化手段对软件最小可测单元（模块）进行的测试。一个最小可测单元通常为一个方法或函数。单元自动化测试常用的工具（框架）有 JUnit、TestNG、Jtest、unittest 和 Pytest 等。

（2）集成自动化测试

集成自动化测试是指通过自动化手段把软件的模块组合起来测试。集成测试分为增量测试和非增量测试两种，这两种测试在手工或自动化的集成测试中都可以采用。

① 增量测试。

在增量测试中，如果增量方式是自顶向下的，那么测试的过程需要桩；如果增量方式是自底向上的，那么测试的过程需要驱动。在实际项目中，经常涉及的“打桩”概念就来自于增量测试。

② 非增量测试。

非增量测试不像增量测试那样有严格的增量顺序，因此在非增量测试过程中，可能同时需

要桩和驱动，也可能只需要其中一个，或者两者都不需要。

集成自动化测试的主要表现之一为接口自动化测试，用于接口自动化测试的常用工具（框架）有 JMeter、HttpClient、requests、SoapUI 和 Postman 等。

（3）系统自动化测试

系统自动化测试是指通过自动化手段，将软件、硬件、操作人员当成一个整体进行测试。系统自动化测试又可分为功能自动化测试、性能自动化测试、安全（渗透）自动化测试和兼容性自动化测试等。系统自动化测试常用的工具（框架）有很多，比如 Testin 云测平台就提供了兼容性自动化测试功能，它可以测试 App 在各种手机上运行是否正常。

2．根据测试类型划分

（1）功能自动化测试

功能自动化测试是指通过自动化手段检查软件能否达到预期功能的测试。主要表现为 Web 应用、移动应用和桌面应用等软件的界面测试。界面自动化测试的常用工具（框架）有 Selenium、Appium、Robot Framework、UFT/QTP 和 AutoIt 等。

（2）性能自动化测试

性能自动化测试是指通过自动化手段模拟各种正常、峰值和异常负载条件，从而对系统的各项性能指标进行测试。在性能自动化测试中，JMeter 和 LoadRunner 占据了绝对的统治地位。

（3）安全（渗透）自动化测试

安全（渗透）自动化测试可理解为通过自动化手段检查软件是否有安全漏洞。安全（渗透）自动化测试包含前期交互、信息收集、威胁建模、漏洞分析、渗透攻击、后渗透攻击和测试报告 7 个阶段。安全（渗透）自动化测试常用工具（框架）有 Metasploit、Burp Suite、Nessus、NMap、sqlmap、Synopsys Defensics 和 Peach Fuzzer 等。

3．根据静态和动态划分

（1）静态自动化测试

静态自动化测试是指通过自动化手段不实际运行软件的测试，比如代码审查、文档测试等。根据编程语言不同，代码审查自动化测试工具（框架）的差异很大。对 Java 而言，常用的有 FindBugs、PMD 等。

（2）动态自动化测试

动态自动化测试是指通过自动化手段实际运行软件的测试，即通常所说的“软件测试”。动态自动化测试（框架）前面已经介绍很多，这里不再赘述。

从狭义来讲，自动化测试主要指动态的功能自动化测试。

1.2　自动化测试的目的

1.2.1　提高软件质量

既然谈到软件质量，那么就有必要了解一下软件质量的度量标准。

1．需求覆盖率

需求覆盖率更多的是从产品经理的角度出发，统计产品需求文档中的需求被覆盖了多少，从而计算出需求覆盖率：

需求覆盖率 = 测试覆盖的需求数 / 需求总数 × 100%

2．代码覆盖率

开发人员可能更关心代码覆盖率。代码覆盖方式有很多种，下面举例说明。

有一个名为 demo 的 Java 方法如下：

```
public void demo(boolean a, boolean b, boolean c, boolean d) {
    if (a && b) {
        System.out.println(true);
    } else if (c || d) {
        System.out.println(false);
    }
}
```

（1）语句覆盖

语句覆盖的原则是覆盖每条语句，针对 demo 方法，一条测试用例即可完成覆盖：

Case 1：a=true，b=true，c=true，d=false

（2）分支（判定）覆盖

语句覆盖并没有考虑 if 语句为假（false）的情况，显然测试并不充分。分支（判定）覆盖可以解决这个问题。采用分支（判定）覆盖重写的测试用例如下：

Case 1：a=true，b=true，c=true，d=false

Case 2：a=true，b=false，c=false，d=false

（3）条件覆盖

分支（判定）覆盖看似比语句覆盖更加完美，但分支（判定）覆盖并没有考虑每个条件的

每个取值（即 a、b、c、d 均可以取 true 或 false 两个值）。条件覆盖能覆盖到每个条件的每个取值，采用条件覆盖重写的测试用例如下：

Case 1：a=true，b=false，c=true，d=false

Case 2：a=false，b=true，c=false，d=true

（4）分支（判定）—条件覆盖

如果能同时满足分支（判定）覆盖和条件覆盖就更好了，而分支（判定）—条件覆盖就能做到，采用分支（判定）—条件覆盖重写的测试用例如下：

Case 1：a=true，b=true，c=true，d=true

Case 2：a=false，b=false，c=false，d=false

（5）条件组合覆盖

条件组合覆盖考虑的是覆盖每个分支（判定）中每个条件的每种组合，采用条件覆盖重写的测试用例如下：

Case 1：a=true，b=true，c=true，d=true

Case 2：a=true，b=false，c=true，d=false

Case 3：a=false，b=true，c=false，d=true

Case 4：a=false，b=false，c=false，d=false

下面对这 4 条测试用例进行详细解释：

若 a && b 为 true，那么 a=true，b=true。

若 a && b 为 false，那么 a=true，b=false；或 a=false，b=true；或 a=false，b=false。

若 c && d 为 true，那么 c=true，d=true；或 c=true，d=false；或 c=false，d=true。

若 c && d 为 false，那么 c=false，d=false。

（6）路径覆盖

路径覆盖的原则是覆盖所有路径，针对 demo 方法，共包含 4 条路径：

a && b 为 true，同时 c && d 为 true。

a && b 为 true，同时 c && d 为 false。

a && b 为 false，同时 c && d 为 true。

a && b 为 false，同时 c && d 为 false。

对应测试用例如下：

Case 1：a=true，b=true，c=true，d=false

Case 2：a=true，b=true，c=false，d=false

Case 3：a=true，b=false，c=true，d=false

Case 4：a=true，b=false，c=false，d=false

如何计算代码覆盖率呢？以路径覆盖为例，假如执行了 Case 1，那么路径覆盖率为 25%（即 1 / 4 × 100%）。

关于代码覆盖的更多内容，参见本书第 12 章。

3．缺陷遗漏率

在软件质量的度量标准中，关于缺陷的不在少数，比如缺陷遗漏率、缺陷修复率、缺陷遗留率和严重缺陷率等。笔者见得最多的是缺陷遗漏率，正所谓“玉瓷之石，金刚试之”，软件质量好不好，用户说了算。当缺陷太多时，说明软件质量仍有很大的提升空间。

缺陷遗漏率 = 线上缺陷数 / 缺陷总数 × 100%

缺陷遗漏率不仅是软件质量的度量标准，也是测试人员考核的硬指标之一。

从软件质量的度量标准中无法看出自动化测试是如何提高质量的，下面举例说明。

假设某软件频繁地进行回归测试，测试人员已经身心疲惫，难以保证每次回归测试的质量，这时如果使用自动化测试代替测试人员进行回归测试，那么回归测试的质量就能得到很好的保证（机器是不知疲倦的），从而减少线上缺陷数，降低缺陷遗漏率。

1.2.2　提高测试效率

可能很多人认为：手工测试执行的用例由自动化测试代替，测试效率的提高显而易见。

但事实上不能如此片面地下结论，原因如下。

（1）自动化测试用例的编写和维护效率远低于手工测试用例

自动化测试用例一天只有几条的产出，而手工测试用例一天可以写几十条，甚至上百条。对于迭代频繁的产品，维护自动化测试用例将是测试人员的“噩梦”。

（2）自动化测试用例执行失败后需要人工分析

自动化测试用例虽然可以做断言，但执行结果仍然需要由测试人员进行人工分析，判断是自动化测试用例本身的问题还是真正的缺陷。

（3）自动化测试无法处理未知的场景

假设测试人员误删除了自动化测试用例在测试环境中创建的一条数据，那么这条自动化测

试用例就会执行失败，从而增加人工分析的时间。而手工测试则不同，如果发现创建的数据被删除了，那么再创建一条即可。

从以上几点来看，自动化测试要想提高测试效率，需要做到以下几点。

① 自动化测试建设初期需要投入更多的人力，否则自动化测试无法形成一定的体量，也就谈不上代替手工测试提高测试效率。

② 产品迭代频率不能太快，否则自动化测试用例将难以跟上维护的节奏，几个迭代过后，自动化用例将不可用。

③ 自动化测试用例质量要过关，否则在执行完成后会出现大量的“误报”，即自动化测试用例本身的问题，并非真正的缺陷。

④ 由于自动化测试对执行环境要求非常苛刻，因此不能与手工测试环境混用。

1.3 自动化测试实施三要素

1.3.1 有明确的目标

做事情如果没有目标，那就是盲目的、随意的，做自动化测试也是如此。下面分别举例说明如何制定提高软件质量和测试效率的目标。

1. 制定提高软件质量的目标

以缺陷遗漏率为例。表 1-1 为某项目的线上缺陷分析情况。

表 1-1

缺陷类别	线上缺陷数	测试发现缺陷数	缺陷总数	缺陷遗漏率
前端展示	1	28	29	3.45%
前端逻辑	4	126	130	3.08%
后端逻辑	22	276	298	7.38%
性能缺陷	0	2	2	0.00%
安全缺陷	0	1	1	0.00%
总　　计	27	433	460	5.87%

从表 1-1 可以看出，后端逻辑的线上缺陷数最高，如果能降低后端逻辑的线上缺陷数，则可有效降低缺陷遗漏率（缺陷遗漏率 = 线上缺陷数 / 缺陷总数 × 100%）。

如果采用接口自动化测试来覆盖后端逻辑，实现 50%的接口逻辑全覆盖，那么理论上线上缺陷数将减少一半，即降低为 11 个，最后可算出缺陷遗漏率为 3.56%。此时提高软件质量的目

标定为 4%比较合适，毕竟真正做到“逻辑全覆盖”是很难的。

2. 制定提高测试效率的目标

以回归测试效率为例。假设某项目的界面自动化用例约为 1500 个，项目迭代周期为半个月，平均每个迭代周期内需要做两轮回归测试，手工用例执行效率为 100 个/人/天。自动化测试用例采用 5 台执行机并发执行，开发提交测试的当天晚上配置好 5 台执行机开始执行，第二天花 1 人/天即可分析完成。但每次迭代完成后，需要花费约 2 人/天做自动化测试用例的维护，如表 1-2 所示。

表 1-2

测试类型	每迭代所需人力	每月所需人力	每季度所需人力	每年所需人力
手工测试	30	60	180	720
自动化测试	4	8	24	96
节省人力	26	52	156	624

通过表 1-2 可以看出，自动化测试明显节省了人力。下面计算对回归测试效率的提高（以每迭代为例）：

$(30-4)\ /4\ \times\ 100\%=650\%$

也就是说，当自动化用例数为 1500 个时，回归测试效率是原来的 6.5 倍（即 650%），不过有两点需要说明。

（1）前期投入

这 1500 个自动化用例的编写大约需要一年时间，平均每个工作日投入人力为 2 人，按一年 250 个工作日计算，投入人力为 500 人/天，即一天一个人实现 3 条自动化测试用例。

（2）维护成本

每迭代只需花费 2 人/天做自动化测试用例的维护，可以看出项目界面变更并不频繁。如果项目界面变更很频繁，则势必要增加维护成本。

通过这个例子可以看出，在做自动化测试之前，需要计算项目的回归测试用例数、前期投入和维护成本，才能得出真正提高测试效率的目标。对于界面变更很频繁的项目，做界面自动化测试甚至比手工测试投入的人力更多，对于这种项目，必须果断放弃界面自动化测试。笔者就经历过很多这样的项目。

1.3.2　有足够的资源

当目标明确以后，就初步确认了需要做哪种类型的自动化测试，以及待实现自动化的手工