第 1 章

自动化测试概述

自动化测试是一个软件测试工程师追求技术、挑战自我、进阶升级的必备技能，因为在软件测试中引入了自动化，可以节省大量的资源，包括人力、财力、时间，提高测试效率。本章将从什么是自动化测试、熟悉自动化测试理论、自动化测试常用工具和建立自动化测试学习体系 4 个方面介绍自动化测试基础和如何学习自动化测试。

1.1 什么是软件自动化测试

软件自动化测试英文是 Software Automated Testing，是指采用程序模拟人的操作检验被测程序或应用可以正常运行的一种软件测试技术，旨在提高测试效率，保证软件质量。

1.1.1 定义

百度百科中对自动化测试的定义是：自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程。在此，自动化测试一般指的是软件自动化测试，也就是测试工程师设置一定的条件，然后通过工具或编写脚本模拟人的行为运行被测系统或应用程序，并且以断言方式将执行结果与预期结果进行对比，进而评估测试结果，最后将整个测试报告输出，输出的测试报告通常包括运行环境、运行时间、成功/失败的测试用例等内容。

软件自动化测试就是将软件测试工软件师编写的手工功能测试用例通过工具或脚本实现、组织、自动执行的一个过程。也有一些人将自动化测试定义为，凡是通过机器、工具、脚本等进行的非人为操作代替手工测试，便认为是自动化测试。

1.1.2 应用条件

实现自动化测试并不是所有的软件或程序都适用，它具有一定的局限性。如果要对某个应用程序实现自动化测试，那么执行者就需要考虑实现自动化测试带来的价值、投入与产出比等一系列现实因素。一般情况下，可以实现自动化测试的软件或程序需具备以下特点：

1. 需求稳定

只有在产品需求稳定，变更不太频繁的情况下，才值得开展自动化测试。自动化测试用例是根据产品的特性来设计的，一旦产品有所变动，测试用例就需要跟着做出调整，那么之前写的自动化测试用例就相当于报废了。如果产品需求明确，或者某一部分功能模块稳定，则可以考虑对明确的产品或稳定的功能模块引入自动化测试。

2. 项目开发周期比较长

进行自动化测试的产品一定要有较长的开发周期。实现自动化测试是为了方便在以后的测试中，确保已经验证过的稳定功能是正确的，而不仅仅是为了发现 BUG。自动化测试本身是对产品进行需求分析、编写自动化测试用例、执行测试的一个开发过程，本身就需要一个较长的时间对测试脚本进行调整和维护。如果项目开发周期短，手工测试就足够了，没必要再投入资源实现自动化测试。

3. 有大量的重复性测试工作

自动化测试具有两大显著特点，那就是减少大量的重复性工作和提高测试效率。软件版本迭代中，每一次迭代都需要重复执行大量的测试用例，如果采用人工执行，会消耗大量的时间。随着产品功能的增强，测试用例的数量也会增加，执行时间会更长，此时就可以看出自动化测试具有明显的优越性。

4. 手工测试难以胜任的工作

某些测试通过工具或编写代码更容易实现，手工测试则难以完成。例如性能测试中对 CPU、IO、内存等的监控，长时间运行过程中的变化曲线。此类测试建议优先考虑自动化测试。

5. 各方面的支持

实现自动化测试需要得到公司和项目领导的支持。首先是自动化测试对人员要求比较高，公司需要培养自动化测试工程师或从社会上招聘，需要消耗公司财力；其次自动化测试还需要运行环境，以及测试人员、开发人员、运维人员等的配合，消耗其他人员的宝贵时间；最后自动化测试项目短时间很难完成，长时间又不会像手工测试一样产出很多 BUG，可能会让各层级领导觉得没有非常明显的价值。因此，自动化测试的实现需要得到各方面的支持。

以上是对适合引入自动化测试的项目做了介绍。可见，如果项目需求变更频繁、投入周期短、投入后产出的价值低、人物财力资源又不足等，那么，是不太适合引入自动化测试的。

1.1.3　对比手工测试

自动化测试比起手工测试能带来许多优势，比如节约时间、提高测试用例执行效率等，但实施自动化测试也有一定的要求和限制，自动化测试并不能完全代替手工测试，两种测试只能作为一种互补。下面对自动化测试和手工测试做一个简单的对比。

- 测试对象：自动化测试的功能模块一般都比较稳定，不会有太大的变动；手工测试的对象比较任意，不管是稳定的，还是新开发或修改的功能模块都可以进行测试。
- 测试效率：同等级的测试用例，自动化测试执行效率更高。
- 测试用例编写：自动化测试需要脚本编写、调试、思考与其他脚本的关系等，而手工测试只需要用自然语言按照一定的格式写出来就行。
- 结果可靠性：自动化测试的结果判断由机器进行，非常可靠；手工测试的结果判断容易受到测试人员的影响，没有机器那么可靠。
- 资源利用：自动化测试可以在任意时间执行，而手工测试只能在人员上班时间进行。因此，自动化测试可以更充分地利用资源。
- 人员要求：手工测试人员在掌握测试基础和理解产品业务后便可开展工作；自动化测试人员不但需要掌握手工测试人员的技能，而且还要将测试用例通过工具或脚本进行实现。因此对自动化测试人员的技能要求更高。

自动化测试与手工测试的对比如表 1-1 所示。

表1-1　自动化测试与手工测试对比

对　比　项	自动化测试	手工测试
测试对象	稳定的功能模块	稳定的、新开发或修改的功能模块
测试效率	快	慢
测试用例编写	用时较长，速度较慢	速度比较快
结果可靠性	可靠	容易受到测试人员的影响
资源利用	充分利用	只有在人员上班时间才会使用
人员要求	比较高	一般

自动化测试与手工测试的关系

自动化测试运行的测试用例是既定的，目的是快速执行，保障产品稳定的功能是正确的。故自动化测试执行效率快，范围有限。

手工测试在执行既定用例时，由于带有人的主观思想，因此可以延伸发散，可以对产品完成更加深入的测试，但手工执行效率远远落后于机器的效率。

1.1.4　分类

这里我们根据被测对象和测试内容对自动化测试来进行分类。

根据被测对象可以分为单元测试自动化、代码库测试自动化、接口测试自动化、浏览器端测试自动化、桌面应用程序测试自动化、移动端测试自动化。

- 单元测试自动化：被测对象是类或方法，关注的是代码的实现与逻辑。
- 代码库测试自动化：被测对象是代码库提供的对象、方法和属性，关注的是代码库提供

的功能。

- 接口测试自动化：被测对象是接口，主要指前后端交互的接口，关注的是传递的数据。
- 浏览器端测试自动化：被测对象是一个 Web 系统，主要指 Web 页面，关注的是 UI 界面和系统的业务逻辑。
- 桌面应用程序测试自动化：被测对象是一个 C/S 架构的应用程序，关注的是应用程序的自身布局和业务逻辑。
- 移动端测试自动化：被测对象是一个移动设备上的应用程序，关注的是应用程序的自身布局、业务逻辑和与移动设备的交互。

根据测试内容可以分为功能测试自动化、接口测试自动化、性能测试自动化、大数据测试自动化、AI 测试自动化。

- 功能测试自动化：测试应用程序提供的功能和业务逻辑。
- 接口测试自动化：测试应用程序与自身的接口、提供给第三方的接口和使用第三方的接口之间的数据交互。
- 性能测试自动化：通过自动化测试工具或者代码手段，来模拟正常、峰值以及异常负载访问被测系统，来观测系统各项性能指标是否合格的测试过程。
- 大数据测试自动化：测试系统对海量数据的处理，包括数据的创建、存储、建模、检索、计算和分析。
- AI 测试自动化：结合 AI 的架构、算法和应用场景做针对性的测试。

1.1.5　价值

自动化测试的实施具有诸多限制，但是许多项目还在大量地投入资源进行建设，价值何在？下面我们来看一个案例。

案　例

A 项目初期，有 1 个测试人员，300 条测试用例，两个月发布一次版本，一天执行 15 条测试用例，需要 20 天执行完所有的测试用例。工作可以完成。

半年后，随着功能的增长，有两个测试人员，800 条测试用例，一个半月发布一次版本，每人每天执行 15 条测试用例，需要 27 天执行完所有的测试用例。工作可以完成。

一年后，功能更多了，用户也增多了，用户的需求也多了，需要一个月发布一次大版本，这个时候测试人员增加到 4 个，2000 条测试用例，每人每天执行 15 条测试用例，需要 33 天执行完所有的测试用例。勉强可以完成工作。

在此期间，如果每 10 天还需要发布一次小版本，修复用户的 BUG，那么还能完成测试工作吗？

假如一直依靠人力解决该问题，显然是不合理的、不科学的。我们可以试想一下，在半年后

就开始投入人力将测试用例自动化，那么后续的版本迭代中，对已有的稳定功能进行自动化测试验证，对新开发的或有变动的功能模块进行人工测试，测试工作就不会非常紧张，而且功能测试覆盖率也会比较高。

我们来看上述案例的另一个版本

A 项目初期，有 1 个测试人员，300 条测试用例，两个月发布一次版本，一天执行 15 条用例，需要 20 天执行完所有的测试用例。工作可以完成。

半年后，随着功能的增加，有两个测试人员，1 个自动化测试人员和 1 个手工功能测试人员，800 条测试用例，其中 600 条实现了自动化测试用例，一个半月发布一次版本，每人每天执行 15 条测试用例，需要 14 天执行完所有的测试用例。工作可以完成。

一年后，功能更多了，用户也增多了，用户的需求也多了，需要 1 个月发布一次大版本，这个时候测试人员增加到 4 个，1.5 个自动化测试人员和 2.5 个手工功能测试人员（其中一个人在版本发布时做手工测试工作，平时维护自动化测试用例，也就是版本发布时会有 3 名手工功能测试人员），2000 条测试用例，其中 1600 条实现了自动化测试用例，每人每天执行 15 条测试用例，需要 9 天执行完所有的测试用例。工作可以完成。

在此期间，如果 10 天还需要发布一次小版本，修复用户 BUG，那么小版本发布时使用自动化测试脚本保障产品稳定功能，手工测试人员对优先级高和用户 BUG 及 BUG 附近的功能进行验证，那么小版本的发布对产品质量完全可以得到保障。

我们可以看到，投入自动化测试带来的效果是显著的。下面对实施自动化测试可以带来的价值做一个总结：

- 提高测试效率，缩短测试时间。
- 减少受人为因素的影响。人工测试对结果的判断受人为因素的影响，使用自动化测试可以规避人为因素的影响。
- 保障测试覆盖率。每一次版本迭代都需要执行全部的测试用例，如果测试用例数量非常大，单靠人力很难全覆盖，使用自动化测试则很容易对已有的功能模块进行全覆盖。
- 提高可靠性。自动化测试中的断言比人工断言更严谨，例如字符串“abc ”，人工可能会忽略后面的空格认为是“abc”，而自动化测试中则不会。
- 更好地利用资源。自动化测试的执行可以在任意时间，比如晚上两点钟。例如一台测试机器，白天由测试人员使用，晚上执行自动化测试用例，可以更充分地利用公司资源。
- 减少测试人员烦躁的心情。一直反复枯燥的人工点点点工作，测试人员难免会产生厌烦。实行自动化测试可以避免通过人力对用例多次反复的执行，同时做手工测试的人员也可以接触自动化测试，掌握新技能，个人能力得到提升。

1.2　自动化测试的概念

在学习自动化测试之前，我们需要先掌握一些基本概念，然后带着这些概念学习自动化测试，如此才能快速理清思路，遇到问题也能够找到解决方法。

1.2.1　流程

自动化测试流程是我们从 0 开始实现一个自动化测试项目，直至测试结束的一个过程，与手工测试流程相同。

自动化测试流程大体可以分为 8 大步骤，即分析测试需求、制定测试计划、设计测试用例、搭建测试环境、编写测试脚本、执行并分析结果、跟踪测试问题、出具测试报告。如图 1-1 所示。

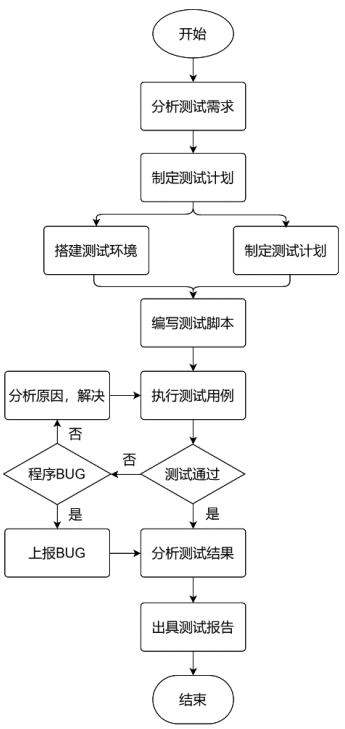

图 1-1　自动化测试流程

1. 分析测试需求

测试的最终目标是保障产品质量，确保产品符合用户需求，所以需求仍然是第一位的。通过分析需求，了解测试要点、用例覆盖点、测试颗粒度等，测试人员会有个初步的方案，都测试哪些内容，什么时间开始和结束，实现哪些类型（API 测试、单元测试等）自动化测试，实现到什么程度，会遇到哪些阻碍和问题。最终，测试工程师要知道测什么？怎么测？什么时候测？

2. 制定测试计划

测试计划是对测试进度的一个监管。测试计划是一个描述了要进行的测试活动的范围、方法、资源和进度的文档，明确了测试对象、测试目的、测试项目内容、测试方法以及所需的资源分配、人员协调，可以有效预防计划的风险，保障测试的顺利实施。

3. 设计测试用例

测试用例是通过分析测试需求，提取测试点，编写形成可执行的测试用例文档，此时的测试用例还只是文档性的手工可执行的测试用例，需要后面通过编写测试脚本转换为自动化测试用例。

4. 搭建测试环境

搭建自动化测试环境是非常重要的一个环节，是后续脚本开发的基础。这是一个大框架，包括采用什么工具、什么样的运行环境、网络等，都是需要着重考虑的。

5. 编写测试脚本

编写测试脚本实际上就是将手工测试用例转换为自动化测试用例的一个过程，在此过程中，要运用所学的自动化测试框架、编程设计模式、面向对象/过程编程思想等合理地编写脚本，使得最终自动化测试项目易用、易维护，结果方便查看。

6. 执行并分析结果

执行自动化测试项目并分析测试结果，对测试环境不稳定的地方进行调整，对脚本逻辑不严密的地方进行优化，对发现的项目 BUG 及时上报。

7. 跟踪测试问题

测试问题要持续跟踪，直至解决闭环。如果是脚本问题，就需要记录，在合适的时间解决；如果是项目 BUG 就需要提交 BUG 单，并在自动化测试用例上关联或做标记。

8. 出具测试报告

一般来说，自动化测试报告是集成在自动化测试项目中，不需要单独作为一个步骤，但测试报告是自动化执行结果的体现，因此在此还是作为一个单独的步骤来说明。自动化测试报告是自动化测试的一个总结，其包括报告执行环境、测试用例成功/失败情况、结果分析等内容。

1.2.2　原则

自动化测试原则是经过前辈们长期自动化测试经验总结所得出的具有指导意义的、合理化的自动化测试准则，为后来人提供便捷、减少弯路的自动化脚本开发的指导思想。下面简单地介绍一些自动化测试原则。

1. 了解自动化测试的目的

自动化测试并不是需要实现所有的手工测试用例。我们需要了解做自动化的目的，如果将自动化测试只是定位在冒烟测试和回归测试，那么它的目的就是验证最核心的几个功能没有问题，自动化测试用例就只需要实现冒烟测试和回归测试用例即可。

如果一个项目要通过自动化测试完成产品的测试，实现测试用例时，首先开展的也往往是核心业务或重复执行率较高的部分，抓住重点内容，后续才向边缘扩充。

2. 验证程序而非发现问题

自动化测试主要是用来验证程序没有问题，而非发现问题，这是一个思想上的认识。在编写自动化测试用例时，要抓住稳定的功能模块，减少开发动荡模块的测试用例。对于动荡模块来说，非常不稳定且易变更，开发人员稍微做出一点点调整，自动化测试用例就有可能全部推翻重来，比起手工测试花费的成本可就太大了。

3. 一条测试用例是一个完整场景

一条自动化测试用例应该只有且只能验证一个测试场景，该测试场景也不宜过于复杂。例如验证登录后新增学生，那么一条测试用例就只验证新增学生的一个点，不能将登录验证也添加其中，登录应该登录验证的测试用例，在此登录也应该是前置条件。

4. 用例的独立性

自动化测试中非常关注测试用例的独立性，每一条测试用例都可以拿出来单独运行，与其他用例不产生依赖或影响。

5. 正向为主

一个测试点会有非常多的逆向逻辑验证。例如登录场景，正向逻辑是输入正确的用户名和密码登录，逆向逻辑就有用户名错误、用户名为空、密码错误、密码为空等很多种。而用户操作最多的则是正向逻辑，因此自动化测试要以正向逻辑用例为主。再就是逆向逻辑情况较多、验证复杂，需要投入非常高的成本，在编写自动化用例时以辅助为主。

6. 回归原点

自动化测试需要回归原点，无论是配置文件、测试步骤还是测试数据。回归原点后，可以保证每轮测试执行都可顺利执行，例如删除 Id=3 的学生，本次测试删除后下次还需要删除 Id=3 的学生，那么就需要在每次删除测试完成后将 Id=3 的学生进行恢复。

1.2.3　测试模型

自动化测试模型是自动化测试框架与工具的设计思想，在自动化测试项目中，它指导着测试工程师合理地排版布局、设计模块。目前探索出的设计模型有线性模型、模块化模型、数据模型、

关键字模型、行为模型以及混合模型。

1. 线性模型

通过录制或编写脚本与应用程序的操作步骤对应起来，每个测试脚本都是相对独立的，且不产生其他依赖与调用，单纯的模拟用户操作场景。线性模型脚本开发时不需要考虑任何技巧，单个脚本开发速度非常快，但对于整个测试项目，这种测试脚本在开发和维护上都会出现大量的重复操作，成本是非常高的，首先脚本不能重复利用，其次当操作有所改变时相关的脚本都需要修改。因此，线性脚本比较适合临时、一次性的测试场景。

2. 模块化模型

将重复的操作独立出来，封装成公共模块，在编写测试脚本时如果需要便可以直接调用，提高代码的复用性。模块化模型是对线性模型上缺陷的纠正，它消除了线性脚本中不能重复利用的弊端，从而提高了自动化测试项目的可读性和可维护性。

模块化模型经过发展，衍生出了页面对象模型（Page Object Model，简称 PO 或 POM 模型），在 Web UI 自动化测试中应用非常广泛。通过面向对象的方式，封装页面定位和页面动作操作，与测试逻辑分离，实现页面对象与测试逻辑解耦，方便后续维护。

3. 数据模型

输入数据的改变驱动自动化测试的执行，最终引起测试结果的改变，简单理解就是数据参数化。装载数据的方式可以是列表、字典、Excel、数据库、XML 等外部文件，实现了数据与脚本的分离。

数据模型最大的益处就是数据与脚本分离，一套测试程序通过不同的测试数据产生不同的测试结果，可快速应对操作步骤相近，大批量测试数据的测试场景。

4. 关键字模型

通过关键字的改变引起测试结果的改变，也称为表驱动测试或基于动作词的测试。关键字驱动模型将测试用例分为测试步骤、测试对象、对象操作和对象数据 4 个部分，是对测试步骤的分解，将每一个操作步骤都分解成测试对象、对象操作和对象数据三个内容。测试步骤是该步骤的动作描述，可以理解为操作名称；测试对象是操作的元素；对象操作是对象动作，对象行为，例如单击、输入；对象数据是测试对象所需要的数据。

5. 行为模型

行为模型是一种敏捷开发方法，从用户的需求出发强调系统行为。通过使用自然描述语言确定自动化测试脚本的模型，自然描述语言即人类常用交流的语言，例如汉语、英语等。

行为模型可以使更多的人参与到自动化测试用例的开发中，测试用例的写法和手工功能测试用例的写法类似， 例如输入“登录系统”测试程序就执行登录系统操作。

6. 混合模型

混合模型是将模块化模型、数据模型和关键字模型混合在一起，取长补短，针对不同的场景灵活运用，以达到效率的最大化。

自动化测试模型是在实际工作中探索发展起来的。如果将测试模型比作一台榨油机，那么线性模型就是一次性成型的榨油机；模块化模型是由许多零部件组装的榨油机；数据模型是由许多零部件组装的榨油机，不但可以榨菜籽油还可以榨大豆油、花生油、芝麻油等；关键字模型是将榨油机器每个提炼油的工序都分解开来，例如放入大豆 10 斤→机器转动 5 分钟→停止机器→收取油和残渣；行为模型是通过一定的内容输入自动榨油，更高级点则是语音控制，例如语音输入榨取 10 斤大豆油，机器便会自动执行并最终会获取 10 斤大豆油。

1.2.4　度量模型

度量模型也称为成熟度模型。自动化度量模型，也可以称为对一个自动化项目或一个自动化团队的度量模型。不同的人或不同的项目根据自己的实际情况对度量模型的理解也不同，产生的度量标准也不尽相同，现从自动化测试的发展阶段、评估内容、建设能力三个维度对度量模型进行说明。

1. 自动化测试的发展阶段

从自动化的发展阶段来归纳，大致可以分为 5 个阶段，每个阶段均代表一种状态，如图 1-2 所示。

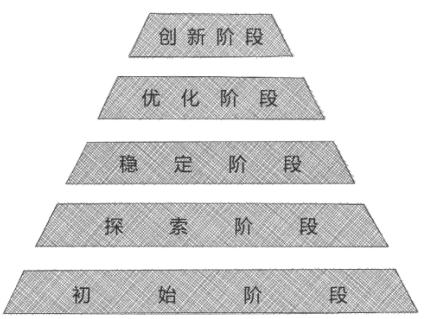

图 1-2　自动化发展阶段模型

- 初始阶段：团队几乎没有任何自动化方面的技术背景或经验，也没有在项目中做任何与自动化测试相关的尝试。该阶段，团队或项目在自动化测试方面一片空白。
- 探索阶段：听过一些技术分享，或团队成员有相关的技术基础，结合被测项目有自动化测试方向的尝试，想通过自动化测试提高测试效率，减少重复的工作。该阶段，团队正在积极探索，构建自动化测试图谱。
- 稳定阶段：经历了探索阶段，自动化理论基础、技术技能等已得到稳固，能成熟运用各

种自动化技术实践，编写的自动化测试脚本也可以稳定运行。

- 优化阶段：优化阶段一定是在稳定阶段的基础上，当一切都稳定了之后，根据团队和当下情况进行优化改进，使自动化测试项目更有利于运行、结果追踪、问题定位等。对于一般团队，优化阶段可以说就是顶峰阶段。
- 创新阶段：创新阶段是创新、领导、先行，带领着自动化测试技术进步。已经超越了企业，是在整个自动化测试生态圈突破壁垒，拓展空间。

2. 评估内容

每一个阶段或每一个自动化项目都可以从覆盖率、有效性、效率、过程质量、体验、发现率 6 个方向来评估做得怎样，产生的效果如何，如图 1-3 所示。

- 覆盖率：包括业务功能覆盖率、异常场景覆盖率和代码覆盖率。
- 有效性：包括用例有效性、代码有效性和测试有效性。
- 效率：包括用例执行效率、冒烟测试效率、回归测试效率、投入资源与产出价值效率。
- 过程质量：包括用例测试质量、测试脚本质量与任务构建质量。
- 体验：包括测试人员体验、研发人员体验、运维人员体验与其他人员体验。
- 发现率：指应用代码和功能被自动化用例检测出缺陷的概率。

通过评估，可以得到接口测试的一些结果数据，对下一阶段自动化测试的开展有参考意义，也有助于不断提高测试的质量和效益。

3. 建设能力

以上两个维度只是对自动化测试当前项目的发展和评估。为了更好地发展，还应该对团队人员和项目建设有所考虑。因此，需要从建设能力维度做进一步考量，如图 1-4 所示。

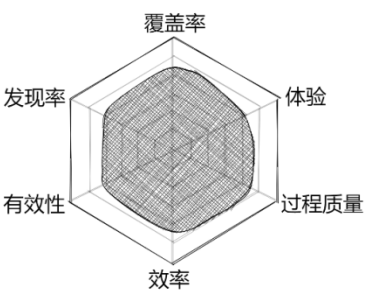

图 1-3　自动化评估模型

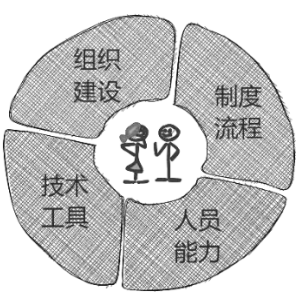

图 1-4　自动化建设能力模型

- 组织建设：保证自动化测试项目的顺利开展。
- 制度流程：管理或标准化自动化测试工作流程。
- 人员能力：人员技术水平、思维严谨等。
- 技术工具：采用什么技术或借助什么工具。

以上从三个维度对自动化测试度量模型做了概述，从这三个维度可以对团队或项目当前的现状进行评估，指导未来的改进空间。

1.3　自动化测试常用工具

实现自动化测试项目，需要借助许多工具来完成。借助这些工具，测试人员可以更好地编写自动化测试用例，运行和查看测试结果。相比手动测试，减少了很多人为错误，也很大程度地节省了人力和时间。

不同类型自动化测试项目使用的工具也不相同，下面根据自动化测试类型介绍一些常用的测试工具。

1.3.1　单元测试工具

单元自动化测试工具经常使用的是一些测试框架，测试框架是对整个或部分系统测试时提供的可重用设计，表现为一组抽象构件及构件实例间交互的方法。单元测试框架是为测试系统中的单元而定制的应用测试框架。在自动化测试中，基本都会使用单元测试框架来组织、管理和执行那些独立的自动化测试用例，并统计测试结果。

不同的工具或编程语言有着不同的单元测试框架，这些框架的结构、用法也都不尽相同，下面介绍一些比较有影响力的单元测试框架。

1. unittest

unittest 是 Python 内置的一个单元测试模块。它的设计受到 JUnit 的启发，与其他语言中的主流单元测试框架有着相似的风格，支持将测试样例聚合到测试集中，并将测试与报告框架独立。

2. pytest

pytest 是一个非常流行的 Python 第三方单元测试框架，简洁高效且支持 315 种以上的插件，同时兼容 unittest 框架，是一个功能齐全的 Python 测试工具。该工具可以帮助我们编写更好的程序，不仅可以编写小测试，还可以扩展到复杂的功能测试。拥有简单灵活、容易上手、支持参数化、可标记测试功能与属性等诸多特点。

3. JUnit

JUnit 是一个 Java 语言的单元测试框架。由 Kent Beck 和 Erich Gamma 建立，逐渐成为源于 Kent Beck 的 sUnit 的 xUnit 家族中最为成功的一个。JUnit 是一个开放资源框架，用于编写和运行可重复的测试，且提供了注释识别测试方法和断言测试预期结果。

4. TestNG

TestNG 旨在简化广泛的测试需求，是一个 Java 语言的测试框架，其灵感来自 JUnit 和 NUnit，

但同时引入了一些新的功能，变得更强大、创新、可扩展、灵活。编写测试通常需要三个步骤：①编写测试的业务逻辑，并在代码中插入 TestNG 注解；②在 testng.xml 文件或 build.xml 中添加有关测试的信息；③运行 TestNG。

5. Go-testing

Go-testing 是 Go 语言自带的测试框架，可以直接进行单元测试、性能分析、输出结果验证等，使用非常简单。

单元测试工具非常多，除了上面介绍的常用工具之外，还有其他非常多优秀的工具，如 Mocha、Karma、Spock、CppUnit、C++Test 等，感兴趣的读者可查看相关资料，此处就不详细介绍了。

1.3.2　代码包测试工具

大部分测试人员很少会接触到代码包测试，但时时都在使用代码包接口。代码（code）是一组由字符、符号或信号码元以离散形式表示信息的明确的规则体系，是程序员用开发工具所支持的语言写出来的源文件。代码包（code package）则是这些源文件的一个集合，是一个有机整体，由许多必要的功能组成，每个功能里面又封装了若干个类和方法，这些类和方法都围绕着核心功能展开，并且提供了相关的操作和数据信息。

在软件开发中，开发人员经常会将一些具有特定功能的公用代码抽离成一个独立模块，其他地方需要时直接集成该模块，可很大程度提高开发效率。将这种独立模块打包共享，在编程语言中就形成了“模块共享”机制。为了方便表述，人们便将这种具有特定功能的独立模块称为包，也称代码包。

代码包是一个完善的代码接口体系，编程人员可以通过它提供的接口实现一些特定功能。很多编程语言中都支持代码包，比如，Python 语言可以在软件库（https://pypi.org/）查看已经发布的 Python 包。

代码包测试的主体是代码包提供给外部可以调用的一系列 API，例如 Selenium 包，测试主体就是 Selenium 提供给编程人员调用的 webdriver.Chrome()、find_element(by, value)、find_elements(by, value)、click()、quit()等类、方法或属性，其本身也是代码。我们可以使用单元测试工具实现代码包接口自动化测试项目。

1.3.3　接口测试工具

现在绝大部分项目都开展了接口测试，有的采用 Postman、JMeter 等可视化工具实现，有的通过 Python、Java 等编程脚本实现，还有的自研了接口测试平台。无论通过哪种方式实现接口测试，其目的都是为适合自身项目需求，产生最有效的价值输出。下面介绍一些在接口测试中常用的工具。

1. Postman

Postman 是一款可以运行在 MacOS、Windows、Linux 系统上的模拟发送 HTTP 接口请求的工具，几乎支持所有类型的 HTTP 请求，提供可视化操作界面，使用简单且容易上手。许多功能

设计都非常切近使用者的习惯，利于项目接口的测试，例如接口分类管理、人员协同、内容分享、发布设计的接口、自动化生成接口文档、在线存储数据和任务定时等。

2. SoapUI

SoapUI 通过 SOAP/HTTP 来检查、调用、实现 Web Service 的功能/负载/符合性测试。该工具既可作为一个单独的测试软件使用，也可利用插件集成到 Eclipse、Maven2.X、Netbeans 和 IntelliJ 中使用。SoapUI 分为开源版和专业版，开源版（OpenSource）可以免费下载和使用，专业版（Pro）需要购买才能使用。对于开发人员和测试人员来说，SoapUI 的简单性和易用性可以加快 REST、SOAP 和 GraphQL 的接口交付。

3. JMeter

Apache JMeter 是 Apache 组织基于 Java 开发的压力测试工具，测试人员也经常用它来做接口测试。它是一个免费开源的、带有图形界面的应用程序，支持多种协议接口，例如 HTTP、JDBC、SOAP、LDAP、JMS、FTP 等。

4. Python-Requests

Requests 是 Python 语言进行标准 HTTP 请求的第三方库。它将请求背后的复杂性抽象成一个简单的 API，以便使用者可以专注于与服务交互和在应用程序中使用数据。Requests 简单便捷，功能丰富，可以满足日常测试需求。

除此之外，我们还会经常用到一些抓包工具，对接口进行分析。常用的抓包工具有 Fiddler、Wireshark、Charles、浏览器开发者工具等，这些抓包工具可以辅助我们更好地开展接口测试工作，提高测试效率。

1.3.4　Web UI 测试工具

近几年，Web 端 UI 自动化测试越来越受欢迎，同时也涌现出许多优秀的框架和工具。不同的工具采用不同的思想策略，实现的功能也各有侧重点。下面介绍一些比较常见的框架或工具。

1. Katalon

Katalon Studio 是一款非常优秀的自动化测试工具，可以运行在 Windows、MacOS、Linux 系统。构建在 Selenium 和 Appium 框架之上，提供了一整套功能支持 Web 系统、App 应用和 API 测试自动化。使用简单，对于不会编程的测试人员，也可以非常容易地开始一个项目的自动化，且支持 CI/CD（持续集成/持续发布）流程集成。

2. Selenium

Selenium 是 ThoughtWorks 公司研发的一个 Web UI 自动化测试框架，直接运行在浏览器中，就像真正的用户在操作一样。其支持在 Windows、macOS、Linux 多平台上运行，支持操作 IE、

FireFox、Safari、Opera、Chrome 等多种浏览器，支持 C#、Java、Ruby、Python 等多种语言。开源免费，且拥有成熟的社区、大量的文档，是目前最火的一款 Web UI 自动化工具。许多工具和测试平台都是建立在 Selenium 基础上。

3. Airtest

Airtest 是一个跨平台的 UI 自动化测试框架，几乎支持在所有平台（Android、iOS、Windows、Unity、Cocos2dx、白鹭引擎、微信小程序）上执行 App 自动化测试。它使用图像识别技术来定位 UI 元素，因此无须嵌入任何代码，即可对游戏和应用进行自动化操作。Airtest 有一个强大的 GUI 工具 AirtestIDE，可以帮助测试人员录制和调试测试脚本。

4. Cucumber

Cucumber 是一个可以用自然语言（自然语言通常指一种自然地随文化演化的语言，例如汉语、英语、日语）描述测试用例的行为驱动（BDD）自动化测试工具。在测试中，将执行步骤和断言信息使用自然语言定义好，编写测试用例脚本时，测试人员只需要根据定义好的文字进行即可完成，与手工功能测试编写用例非常相似，易于理解和维护。

5. Robot Framework

Robot Framework 是一个基于 Python 语言开发的关键字驱动测试自动化框架，功能丰富并且扩展性强，主要用于轮次很多的验收测试和验收测试驱动开发（ATDD）。它的测试功能可以通过 Python 或 Java 实现的测试库进行扩展，用户可以使用与创建测试用例相同的语法，从现有的关键字中创建新的更高级别的关键字。测试用例的编写就像填表格一样，简单易上手。

Web UI 自动化测试框架和工具多彩纷呈，还有一些比较有创意的工具，例如龙测、Cypress、TestCraft、AutonomIQ、Functionize、TestIM 等，都为自动化测试的发展提供了新颖的思路，值得我们学习。

1.3.5　App 测试工具

移动设备的大量使用，也带动了 App 测试技术的长足发展。与此同时，也出现了许多 App 自动化测试工具，例如 Appium、Robotium、Espresso、Calabash、Airtest 等，下面介绍一些常用的 App 自动化测试工具。

1. Appium

Appium 是一个开源、跨平台的移动端自动化测试工具，支持 Android、iOS 和 Windows 平台上的原生、macOS 平台应用、移动 Web 和混合应用，可以使用 Java、Python、JS、PHP 等多种语言编写。Appium 使用 Client-Server 的架构设计，并采用标准的 HTTP 通信协议，Server 端负责与 iOS、Android 原生测试框架交互。当 Client 端向 Server 端发送请求后，Server 端接收请求并进行解析，然后驱动 iOS/Android 设备执行相应的操作，最后将执行结果放在 HTTP 响应中返回客户端。

2. UI Automator

UI Automator 是谷歌推出的一个 Android 自动化测试工具，基于 Accessibility 服务，功能很强，基本上支持所有的 Android 事件操作。但是该工具只支持在 SDK16（Android 4.1）及以上版本运行，并且不支持移动浏览器和混合应用。同时，测试脚本只能使用 Java 语言，然后打包成 JAR 或 APK 包上传到设备上才可以运行。

3. Monkey

Monkey 是 Android SDK 自带的一个测试工具，使用 Java 语言编写，在 Android 文件系统中存放的路径是/system/framework/monkey.jar。使用 Monkey 工具可向系统发送伪随机的用户事件流，例如按键输入、触摸屏输入、手势输入等，检测程序长时间的稳定性。在测试中还会将日志输出，方便问题定位。但也有一定的局限性，即测试对象仅为应用程序包，生成的事件是随机的且不能自定义。

4. Robotium

Robotium 是一款 Android 自动化测试框架，主要用于 Android 平台的应用进行黑盒自动化测试，它提供了模拟各种手势操作（单击、长 按、滑动等）、查找和断言机制的 API，能够对各种控件进行操作。Robotium 拥有许多优点，例如轻松识别元素，API 使用简单，执行速度快，支持原生和混合应用。也可以和 Maven、Gradle 以及 Ant 工具结合，进行持续集成测试。

5. Espresso

Espresso 框架是谷歌官方大力推荐的一套测试框架，相比 Robotium 框架，Espresso 提供的 API 更加精确，编写测试代码简单，容易快速上手。

除了上面介绍的一些工具外，还有许多非常友好的 App 自动化测试工具，例如 KIF、XCTest、ATX、Macaca、Calabash 等。这些工具采用了不同的设计理念，从不同的方向和角度打造，目的都是更好地辅助 App 测试。

1.3.6　性能测试工具

性能测试也是当今测试人员发展的一个主流方向，要想深入理解性能测试，掌握基本的性能测试工具很有必要，其中 JMeter 和 LoadRunner 两款工具最有代表性。它们可用以模拟多并发用户，向服务器施加压力，帮助业务快速定位产品性能瓶颈，准确验证系统能力，全面提升稳定性。常见的性能测试工具如下：

1. JMeter

Apache JMeter 是 Apache 组织基于 Java 开发的压力测试工具，测试人员也经常用它来做接口测试。它是一个免费开源的，带有图形界面的应用程序，支持多种协议接口，例如 HTTP、JDBC、SOAP、LDAP、JMS、FTP。

2. LoadRunner

LoadRunner 是一款适用于大型企业应用程序的性能和负载测试。支持多种协议和技术，具有强大的分析和报告功能，被广泛使用，拥有丰富的资源和支持。

3. Locust

Locust 是用 Python 语言编写的开源、轻量级负载测试工具，可用于测试系统的承载能力和性能。适用于灵活定制性能测试脚本的场景。

4. WebLOAD

WebLOAD 是一款专业的性能测试工具，用于测试 Web 和移动应用程序的性能。它支持多种协议和技术，包括 HTTP、Ajax、WebSocket 等，并提供实时监控、性能分析和报告等功能。WebLOAD 还可以与 CI/CD 工具集成，实现自动化的性能测试流程。

当然，还有其他一些非常优秀的性能测试工具，例如 Gatling、PerfDog 等，都有各自的特色，如果有精力，也可以了解一下。

1.4　如何入门自动化测试

入门自动化测试是一个过程，需要学习者实现行为和思维两方面的转变，这也恰恰是许多即将入行人员最关心的内容。本节将结合笔者多年的从业实践，谈一下这方面的经验。

1.4.1　入门是基础

入门自动化测试需要拥有两个基本功，测试基础理论、经验和自动化工具的基本使用。首先，需要了解软件测试的基本概念、原理和流程，掌握测试用例的编写、执行、记录等基础能力。同时，还需要学习测试用例的设计原则和方法，对测试对象进行全面的了解。其次是掌握像 Postman、Selenium、Robot Framework、Appium 等自动化测试工具的使用。自动化测试是建立在测试基础、手工测试之上实现的，需要夯实的测试理论基础和大量的实践经验，因此在入门过程中手工测试经验积累是重中之重。入门自动化测试大致需要 1~3 年时间，在学习自动化测试的这段时间内，要并行开展手工测试以积累经验，掌握常用测试工具的使用。手工测试是本职工作，掌握常用测试工具是自我提升，要从行动上落实。

当初步掌握自动化测试工具的使用后，可通过项目进行尝试，体验自动化测试带来的价值。例如学完 Robot Framework，尝试将项目中优先级高的手工功能用例实现自动化，初步在项目中应用。在接触过程中逐渐体会自动化测试思维的转变，例如自动化测试更多在于检查已有功能的稳定，而非验证新功能的正确。此时，要考虑用例的实现，还要考虑脚本稳定、函数复用等。

入门自动化测试后会有一个非常直观的感受，任意一个常见的场景，可以瞬间给出可能的测试用例并列出高优先级用例，思考学过的自动化测试工具并给出可能实现的方案，此方案可能不

是最优，但可以实现该场景的高优先级用例。

1.4.2　入行是起点

入行是从零开始，通过大量自动化测试经验教训的累积，形成自己的解决方式。例如本书实践篇第 9 章至第 15 章，就是笔者自动化测试经验的沉淀和思想的表现。入行的起点是模仿，基于入门基础，模仿他人的笔记完成当前工作，在模仿的过程中揣摩他人的写作思路，吸收精华，找出不足并思考如何改进；接着形成自我形态，在不打破现有自动化项目结构、规则、模型的前提下，实现自己思路；最后思考整个自动化项目的优缺点，提出改进方案，与前辈探讨可行性。当能够考虑整个自动化测试项目的时候，那么恭喜你已经成为自动化测试行业的骨干人员。

入行自动化测试后会有两种自信的表现，一种是基于他人项目模型，在熟悉后任意内容可快速实现自动化测试用例；另一种是承担新项目自动化测试工作，搭建一套基础框架并给出样例模型，他人模仿样例很容易补充自动化用例。

1.4.3　入职是挑战

入职一家新公司，接触一个新项目对自身来说都是一种挑战，首先是团队氛围，交流方式都与之前不同，要学会融入；其次是团队工作方式的改变，例如之前是被动式工作，领导安排什么活就干什么活，新团队是主动式工作，领导不会安排具体的工作，只罗列了范围、期望结果，需要自行管理过程，自我主动思考、尝试解决；最后是项目的变化，例如之前是合同管理项目，现在是类似于 Excel 的一个平台，那么在自动化测试实现的方式上就需要合理规划各种控件的自动化脚本实现。进入一个新团队，首先要做到的就是融入，其次才是思考自动化的实现。如果新项目已有自动化工作的积累，则先模仿补充，逐步优化；如果未有自动化测试的积累，则请教其他成员（例如开发、运维）团队的资源调度、工作规划等，结合团队当前已有的再有序拓展。入职新的团队，始终要保持空杯心态，自信而不失礼节地开展工作。

自动化测试入门简单，只需要熟悉测试理论，一定的手工功能测试经验和基本的相关工具使用。但入行就需要大量的经验积累，是一个长期的实践过程。本书作为一本软件自动化测试入门指南，希望带领读者快速入门，深入思考。

1.5　思　考　题

1. 什么是自动化测试？
2. 实施自动化测试能带来哪些价值？
3. 你都知道哪些自动化测试模型？
4. 开展一个自动化测试项目，需要经过哪些步骤？
5. 评估自动化测试项目，应从哪几个方面入手？