文章编号：2095-2163(2019)01-0277-04 中图分类号：TP311 文献标志码：A

基于 pytest 和 JMeter 的自动化测试系统设计和实现

李一风

（上海你我贷互联网金融信息服务有限公司，上海 200120）

摘 要：JMeter 作为一款优秀的测试工具被广泛应用于接口测试和性能测试，但是由于 JMeter 不是一种测试框架，在面对复杂的测试计划和众多测试用例的测试集时，用例管理和用例调度需要测试人员手工介入，大大影响了测试效率。本文设计并实现了一种基于 pytest 的自动化框架，实现了 pytest 和 JMeter 的有机结合，可以实现用例的管理、分布式执行、失败重试等功能，提高了 JMeter 测试用例的执行效率，降低了人为操作的出错概率。

关键词：自动化测试；分布式；pytest；JMeter

Design and implementation of automated testing system based on pytest and JMeter

LI Yifeng

（Niwodai Internet Financial Information Service Co. Ltd. , Shanghai 200120, China）

【Abstract】 As an excellent test tool, JMeter is widely used in interface testing and performance testing. But JMeter is not a testing framework, so in the face of complex test plan and test suite that contains many test cases, case management and use case scheduling require manual actions by testers, which will affects the test efficiency greatly. This paper designs and implements an automated framework based on pytest. The combination of pytest and JMeter is realized, which can achieve the management of test cases, distributed testing and re-run on failure. This testing system can improve the execution efficiency of JMeter test cases and reduce the error probability of human operation.

【Key words】 automated testing; distributed; pytest; JMeter

0 引 言

自动化测试是软件测试未来的发展方向，自动化测试中往往会遇到测试脚本管理困难、测试效率低等问题[1-4]。如何选用合适的测试工具和搭建有效的测试框架是测试工程师一直在研究的问题。JMeter 作为一种功能丰富的测试工具受到了广泛关注和研究[2-5]。Apache JMeter 是一个基于 Java 语言的开源工具，可以用于软件的功能测试和性能测试。在被测系统功能越来越多，结构越来越复杂的今天，测试用例往往非常庞大，一个回归测试就需要几十或上百个 JMeter 脚本组成的测试集。单纯使用 JMeter 时会存在以下问题。

（1）测试用例的管理。JMeter 本身不提供测试集的管理功能，多个 JMeter 脚本的执行一般需要测试人员手动整理出测试计划中应该包含的测试用例。手动在命令行启动 JMeter 命令，并在发生错误时进行人工排查，达不到一键执行和无人值守。

（2）大量脚本的执行效率问题。JMeter 自带远程执行功能只是针对一个脚本中的多个步骤并行，不支持多个 JMeter 脚本的并行执行。需要实现多个测试用例的并行执行时，需要自行编写复杂的调度程序或者测试人员人工介入。前者成本较高，后者无法实现完全的自动化。

本文设计并实现了一种基于 pytest 的自动化测试系统，具有如下特点：

（1）充分利用 pytest 本身的功能和丰富的插件，只需编写少量代码、简单轻量、可以快速部署。

（2）实现了多 JMeter 脚本的管理，可以一键执行测试集，并在脚本执行中自动进行异常重试，实现了无人值守。

（3）进行远程分布式执行，使得在遇到测试资源瓶颈时，快速进行测试执行机的横向扩展来提高测试计划的执行效率。

（4）自动生成 JUnit 格式报告，可以方便的与持续集成系统进行集成，生成各种测试报告和图表。

1 基于 pytest 的自动化测试系统

1.1 pytest

pytest 是基于 Python 语言的一种自动化测试框

作者简介：李一风（1986-），女，硕士，工程师，主要研究方向：计算机软件研发、自动化测试架构。

收稿日期：2018-11-02

架,本身具有 Python 语言的良好的跨平台性和简单易上手的优点[6-9]。pytest 可以用于单元测试或功能测试,相比其它测试框架,具有以下优点:

(1)简单灵活,容易上手。

(2)支持测试用例的参数化。

(3)能支持简单的单元测试和复杂的功能测试。

(4)具有较多第三方插件,并可以自定义扩展。

(5)执行测试过程中可将某些测试跳过,或对某些预期失败的用例标记成失败。

(6)支持多种格式的测试报告,可很好地与持续系统集成。

基于上述优点,pytest 已经成为 Python 中最流行的测试框架之一。

1.2 系统结构介绍

该框架主要分为 4 个模块,自动化测试系统结构如图 1 所示。

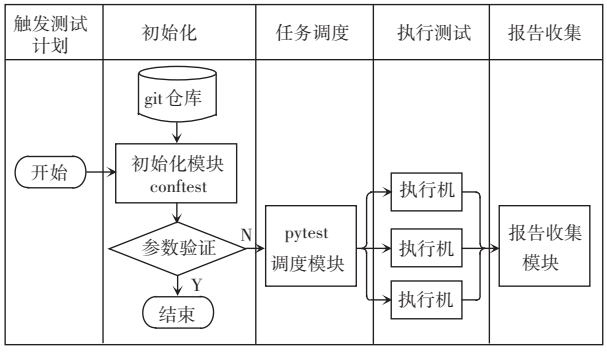

图 1 分布式自动化测试系统结构图

Fig. 1 Structure diagram of distributed automated testing system

(1)初始化模块。用于读取配置文件,获取测试要执行的各种参数;从 Git 仓库拉取最新的 JMeter 脚本文件,转换脚本文件为 pytest 的测试用例参数;建立与测试执行机的 SSH 连接,同步要执行的 JMeter 脚本到各个测试执行机;在测试执行机上创建子进程等待测试任务的分配。

(2)任务调度模块。根据调度算法,分配测试脚本到不同的执行机上的子进程。

(3)测试执行模块。在子进程中执行测试脚本,通过解析 JMeter 生成的 xml 格式的报告,判断测试执行是否成功。

(4)报告收集模块。将所有测试执行机生成的多个测试脚本的报告收集回调度机,生成整个测试计划的报告。

1.3 基于 pytest 的自动化测试流程

本文的自动化测试系统工作流程如图 2 所示。

(1)调用入口程序触发一次测试计划的执行,进行测试计划配置文件的读取,得到要执行的脚本 Git 仓库地址和测试用例目录、测试环境等参数,并启动 pytest 主程序。

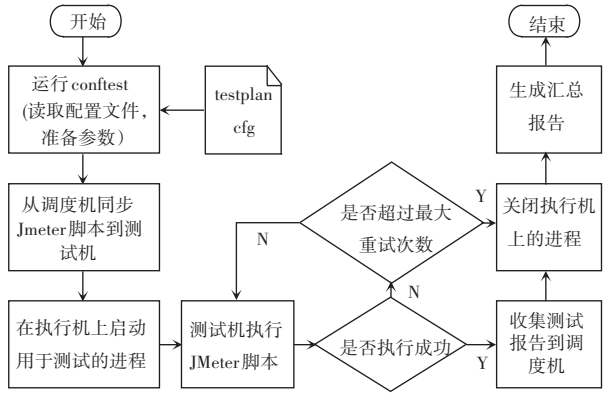

图 2 自动化测试流程图

Fig. 2 Flow chart of automated testing

(2)conftest.py 作为 pytest 主程序的默认参数化文件。本系统在其中实现了测试用例的获取和转换。包括遍历测试目录、得到测试脚本的名称和路径、排除不执行的用例(skip case 和 known error case)、组成 pytest 需要的参数化变量。

(3)pytest 主进程开始执行,首先从调度机同步需要执行的测试脚本文件到各个执行机,进行测试任务的调度和分发。

(4)测试执行机启动 JMeter 进程,执行 JMeter 脚本,并通过解析 xml 判断是否执行成功。为了排除测试环境不稳定造成的测试失败,本系统实现了失败重跑机制,即每个测试有一个最大重试次数,当测试失败时,会判断是否已经达到最大的重试次数。当没有达到时,会对测试用例进行重复执行,取成功的一次执行作为最终结果,直到达到最大重试次数。实践表明,这种机制可以降低测试环境不稳定时测试结果的误报率。另外,为了防止单个测试用例执行时间过长导致整个测试计划执行时间过长,本系统实现了超时结束机制,即设置单个测试用例执行时间的最大值,当某个测试用例的执行时间超过这个最大值时,杀掉当前测试用例的子进程,并标记测试结果为失败。

(5)所有用例执行完成时,由调度机收集在测试执行机上生成的测试用例报告,并合并多个测试用例报告为整个测试计划的汇总报告。

1.4 自动化测试系统的主要技术

1.4.1 用例的管理

通过使用 pytest 的参数化用例方法实现 JMeter 脚本到 pytest 测试用例的转变。由 pytest 实现测试

用例的管理，可以方便实现从指定的目录中筛选用例，根据脚本的名称筛选用例，指定要 skip 或者期望失败的用例等。要实现用例管理，关键问题是如何将以 JMeter 脚本形式存在的用例转变为 pytest 管理的用例。这就用到了 pytest 参数化测试用例功能。下面介绍参数化测试用例的用法。

在 pytest 中使用装饰器@ pytest.mark.parametrize 来实现用例的参数化。一个典型的例子是：

```
@pytest.mark.parametrize("test_input,expected", [
    ("3+5", 8),
    ("2+4", 6),
    ("6+9", 42),])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

在这里，@ parametrize 装饰器定义了 3 组不同的输入值和期望值（即 3 个用例），test_eval 参数将会使用这 3 组值执行 3 次。执行的输出：1 failed, 2 passed in 0.06 seconds 表示执行了 3 个用例。

同理，在手动执行单个 JMeter 测试用例时传入用例名和用例目录进行单个测试的执行。在本系统中，通过自动遍历测试目录，找到所有要执行的 JMeter 脚本，解析出目录和文件名作为 pytest 的测试参数传入，在测试函数中自动调用 JMeter 命令来执行测试。减少了手动介入，实现了用例的高效管理。

1.4.2　测试用例执行结果的判断

在传统方法中，一般需要测试人员通过查看每个 JMeter 脚本生成的结果树来判断断言是否成功，并且单个测试报告无法与持续集成系统集成显示为整个测试计划的报告。本系统通过解析 xml 报告来并通过 pytest 的断言来实现自动化判断测试用例结果，并形成多个 JMeter 测试用例的汇总报告。为了说明工作原理，下面以一个有 2 个步骤的测试用例的报告来说明如何判断测试用例的执行结果。

```
<xml version="1.0" encoding="UTF-8">
<testResults version="1.2">
<httpSample t="40316" it="0" s="true" lb="步骤 1" rc="200" rm="OK" tn="测试 1">
  <assertionResult>
    <name>响应断言-判断步骤 1 是否成功</name>
    <failure>false</failure>
  </assertionResult>
</httpSample>
<samplet="118" it="0" s="true" lb="步骤 2" rc="200" rm="OK" tn="测试 1">
  <assertionResult>
    <name>响应断言-判断步骤 2 是否成功</name>
    <failure>false</failure>
  </assertionResult>
</sample>
</testResults>
```

httpSample 和 sample 表示 2 个采样器的执行结果。例子中采样结果 s="true" 表示为执行成功。通过解析使用 Python 的第三方库 lxml 对 xml 文件进行解析。判断的标准是，如果所有的采样器结果都为成功，标记这个测试用例的结果为成功，否则为失败。使用 xpath 语法获取用例中失败的采样器数目：

"count(/testResults/*[attribute::s='false'])"，如果这个值大于 0，则表示有至少一个采样器执行失败，此时用 pytest 的断言把用例标记为失败。如果等于 0，则标记为成功。

1.4.3　测试用例的分布式执行

当被测应用的功能较多，测试计划包含的测试集很大时，执行一次测试计划的时间就会很长，这时候就需要使用分布式执行。JMeter 本身是支持分布式执行的，其原理是调度机把脚本发送到台执行机上，执行机执行完成后，把结果回传给调度机，调度机收集所有执行机的信息并汇总成一个报告。但是这种分布式是把单个脚本发送到执行机上执行，本质是一个测试用例的多个步骤级别上的分布式执行。无法把测试用例集中的多个测试用例分布到不同的执行机上执行。为了实现测试用例级别上的分布式执行，本文引入了 pytest 的 xdist 插件。

xdist 可以实现用例级别的分布式执行。其中 2 个关键的参数是 rsync 和 tx。本系统使用 rsync 把框架程序和测试脚本同步到执行机上。在本系统中使用负载均衡策略，即按照执行机的当前负载情况进行用例分配。执行流程如下：同步框架脚本和测试用例脚本到执行机，读取测试用例，在执行机上创建等待执行的进程，将测试用例分布到不同的进程中执行。

为了验证分布式系统对测试效率的提升效果，进行以下实验。实验场景：创建 10 个 JMeter 脚本，每个执行时间是 5 s。由 3 台配置相同的虚拟机组成一个测试集群，在虚拟机的数目为 1、2、3 时，执行 10 个测试脚本组成的测试集，记录测试集的执行时间，每次执行 3 次，记录 3 次的平均用时，执行用时见表 1。

（下转封三）