

文章编号:1001-9081(2009)03-0699-03

基于 STAF 的软件自动化测试系统的研究和实现

李夏安, 陈志泊

(北京林业大学 信息学院, 北京 100083)

(zhibo@bjfu.edu.cn)

摘 要:设计并实现了一种基于 STAF 的软件自动化测试系统,能够较好地对自动化测试过程进行管理,并且支持多平台测试网络环境下对自动化测试任务的执行和监控。通过该系统的实施,能够提高软件测试环节中的自动化程度,从而提高测试效率,缩短测试周期,同时增加测试结果的可信程度。

关键词:软件测试;自动化测试;STAF

中图分类号: TP311.5 **文献标志码:** A

Study and implementation of software automatic testing system based on STAF

LI Xia-an, CHEN Zhi-bo

(School of Information, Beijing Forestry University, Beijing 100083, China)

Abstract: A software automatic testing system based on Software Testing Automation Framework (STAF) was designed and implemented to solve the problems in current automatic testing solutions. This system could be employed to manage the process of automatic testing, and support the execution and monitoring of automatic testing tasks in multi-platform network environment. The system was implemented to improve the degree of automation, enhance the efficiency of testing, shorten the testing period, and increase the trustworthiness of the testing results.

Key words: software testing; automatic testing; Software Testing Automation Framework (STAF)

0 引言

软件测试在软件开发生命周期中扮演着越来越重要的作用。伴随着软件规模的不断增大,软件对于运行时环境的要求也日益复杂,这意味着测试人员为了运行测试用例,必须花费更多的时间用于准备测试环境。自动化测试通过减少在测试过程中需要人工操作的步骤,使测试人员得以分配更多的时间用于产生测试工作核心价值的行为,如寻找软件错误和缺陷、验证业务逻辑等。提高软件测试环节中的自动化程度,对于提高整个软件测试环节的生产率具有重要的意义。

文献[1]实现了一种测试数据和测试驱动分离的自动化测试框架,文献[2]则提出了由测试用例生成自动化测试脚本的方案,均可以提高测试用例的复用率,从而减少开发、维护测试用例的成本,但由于它们的底层测试驱动模块都使用脚本语言 Perl 来开发,所以只适用于单机测试环境。文献[3]设计了一种 C/S 模式的自动化测试网络模型,实现了对自动化测试任务的远程执行和监控,但在此系统执行的自动化测试任务必须使用 CORBA 实现通信功能,这无疑增加了测试工作的复杂度,而且测试任务也很难被复用。

STAF^[4]是开源的自动化测试框架,它封装了不同平台和不同语言间通信的复杂性,提供了消息、互斥、同步、日志等各种可复用的服务,使用户可以在此基础上构建自动化测试解决方案。STAX(STAF Execution Engine)是运行在 STAF 之上的可解析、执行 XML 格式自动化测试任务的一种服务。与自动化测试的其他技术(脚本、CORBA)相比,STAF/STAX 具有跨平台、功能强大、复用性强等特点。本文提出并实现了一种构建在 STAF/STAX 基础之上的软件自动化测试系统,该系统

可对自动化测试过程进行有效管理,同时支持在分布式多平台测试网络中对自动化测试任务的部署、执行和监控。

1 设计和实现

1.1 体系结构

基于 STAF 的软件自动化测试系统的体系结构如图 1 所示。借助于 STAF,自动化测试控制服务器与测试代理群组成了 C/S 模式的多平台自动化测试网络。自动化测试控制服务器并不在自身执行具体的测试操作,它主要起到管理和控制两大作用,即一方面利用高层 B/S 模式的 Web 应用程序提供自动化测试信息管理服务并接收工作站操作命令;另一方面利用底层的 STAF,向目标测试代理提交自动化测试任务,同时接收测试代理反馈的执行过程、结果等信息。测试代理指安装了 STAF 的测试机器,负责监听并接收自动化测试控制服务器发送过来的测试任务信息,并执行相应的测试操作,同时把测试过程和结果反馈给自动化测试控制服务器。

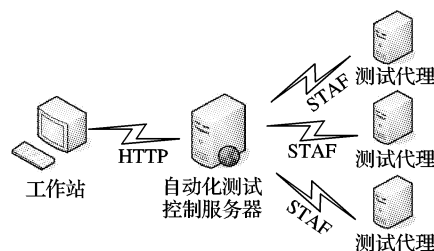


图 1 基于 STAF 的软件自动化测试系统体系结构

1.2 自动化测试任务

基于 STAF 的软件自动化测试系统中运行的测试样例为符合 STAX 语法的 XML 文件,称为自动化测试任务,具有以

收稿日期:2008-09-16;修回日期:2008-11-03。 基金项目:国家“十一五”科技支撑计划项目(2006BAD10A03)。

作者简介:李夏安(1985-),男,安徽太湖人,硕士研究生,主要研究方向:数据库、计算机软件与理论; 陈志泊(1967-),男,山东莒县人,副教授,博士,主要研究方向:数据库、计算机软件与理论。

下特点:

1) 可调用底层 STAF 提供的所有服务,拥有强大的功能来保障自动测试的实现和执行。

2) 可接收参数,从而实现了测试数据和测试任务的分离。

3) 自动化测试任务的基本单元为函数,可为其他任务所调用,提高了测试任务的复用程度。

以下的代码片段说明了如何利用 STAX 在远程机器上静默安装 Oracle 10G,该函数封装了平台的差异性,其他的自动化测试任务可以调用此函数以参数的形式传入安装路径和参数文件的位置,STAF 在远程目标机器运行安装脚本后取回其运行结果,若安装成功该函数返回 0,否则返回错误号码和错误信息。

```
<function name = "SlienceInstallOracle10G" >
  <function-list-args >
    <function-required-arg name = "target"/ >
    <function-required-arg name = "installerDir"/ >
    <function-required-arg name = "responseFile"/ >
  </function-list-args >
  <sequence >
    <stafcmd >
      <location > target </location >
      <service > 'var' </service >
      <request > 'resolve string { STAF/Config/OS/Name}'
      </request >
    </stafcmd >
    <script >
      osType = STAFResult
      installer = 'runInstaller.sh'
      fileSeparator = '/'
      if osType.startswith('Win'):
        installer = 'setup.exe'
        fileSeparator = '\'
      cmd = '% s% s% s - silent - responseFile %s'
        % (installerDir, fileSeparator, installer,
          responseFile)
    </script >
    <process >
      <location > target </location >
      <command mode = "'shell'" > cmd </command >
      <stderr mode = "'stdout'" > <returnstdout / >
    </process >
    <if expr = "RC != 0" >
      <return > RC, STAXResult </return >
    </if >
    <return > 0 </return >
  </sequence >
</function >
```

1.3 自动化测试流程

整个自动化测试流程如图 2 所示,用户通过 Web 界面选择目标测试代理和测试任务,再通过参数传入测试数据,准备就绪后提交给底层的 STAF 执行,最后同样在 Web 界面上观察实时日志和测试报告来监控自动化测试任务。更进一步,用户可以创建测试计划和触发此测试计划的条件——触发器,系统在检测到触发器所需条件成立的时候,将自动提交和执行对应的测试计划,从而实现无需人工干预的全自动化测试流程。

1.4 自动化测试控制服务器端软件架构

自动化测试控制服务器端软件架构如图 3 所示,从逻辑

上自顶而下分为三层:应用层、代理层和实现层。

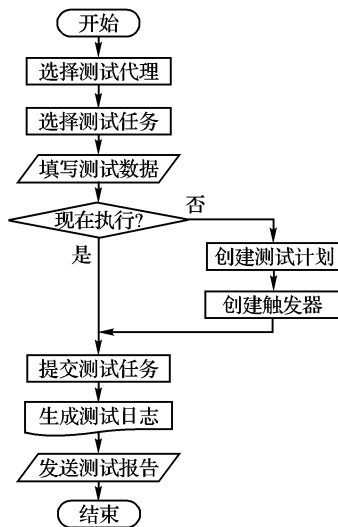


图 2 自动化测试流程

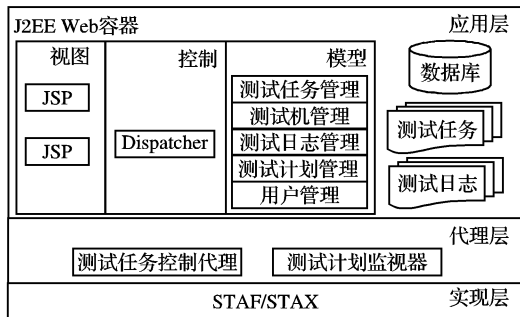


图 3 自动化测试控制服务器端软件架构

1.4.1 应用层

最顶端的应用层为运行在 Java EE Web 容器中的应用程序——自动化测试信息管理系统,该系统利用关系型数据库和文件系统来管理自动化测试过程中的数据和信息,并提供友好的 Web 界面与用户交互。

按照 MVC 应用模型的设计思想,自动化测试信息管理系统由视图、控制器和模型三个部分组成,视图层为 JSP 页面,负责展现内容和接收用户输入;控制器负责分析用户请求并调用合适的模型来处理;模型层由各种 Java Bean 组成,分别封装了各种业务逻辑。这样可以把展现和逻辑完全分离开来,增强了系统的可拓展性和可维护性。自动化测试信息管理系统主要包括以下功能模块。

1) 用户管理。管理系统用户信息,负责系统访问权限的控制。

2) 测试代理资源管理。管理整个测试系统中的测试机资源,对这些资源进行合理分配利用。

3) 自动化测试任务管理。包括自动化测试任务的上传、分类、查询、提交、中断等测试流程中的一系列操作。

4) 测试日志管理。管理包括自动化测试任务的实时运行日志、测试报告、历史记录等信息。

5) 测试计划管理。负责对自动化测试计划及触发器的管理。

按照松耦合的设计原则,应用层应不依赖于底层的具体实现,即应用层并不直接与底层通信,而是使用数据库作为和底层通信的媒介。如要提交执行一个自动化测试任务,应用层只需要把任务信息、测试数据保存在相应的数据表中,并将其状态变更为“等待执行”就算完成了提交操作,接下来只需

以一定的时间间隔从数据表中获取该条任务记录的最新状态,同时从该任务对应的文件系统目录中获取实时产生的执行日志,至于任务的具体执行过程,对应用层来说,是完全不可见的。

1.4.2 代理层

代理层运行在应用层和实现层中间,由任务控制代理和测试计划监控器两个背景程序组成,均在后台运行,前者负责执行用户从 Web 界面中提交的自动化测试任务,后者则负责监控和自动提交在系统中注册的自动化测试计划。

任务控制代理的详细算法流程如图4所示:首先向底层的 STAF 注册一个标志此 Agent 进程唯一身份的句柄,记为 STAFHandle,接着以一定的时间间隔不断地监测数据库中的数据变化,获取所有处于“等待执行”状态的任务列表,然后调用类 AutoTestJobExecutor 来逐个执行这些任务。

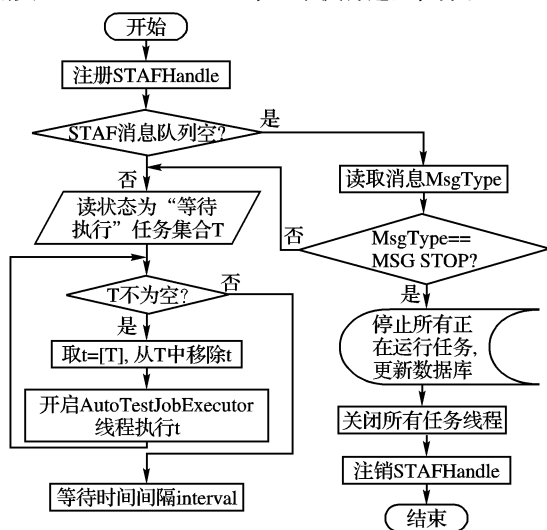


图4 任务控制代理算法流程

在此过程中,任务控制代理会监听 STAF 消息队列,一旦接收到用户发送的 MSG_STOP 消息,任务控制代理在退出自身进程前,会通知底层 STAF 停止正在运行的所有任务,并更新这些任务在数据库中的状态为“已中止”,相比于直接终止任务控制代理进程的方式更为安全,确保了数据的完整性。

任务控制代理使用多线程的方法来支持多任务并发执行,每一个线程都独立处理一个自动化测试任务的执行过程。但多线程也会带来临界资源的互斥使用问题,即在通常情况下,一台测试代理在任意时刻最多只能为一个自动化测试任务所占用。为了防止并发的多个任务使用同一台测试代理,在执行任务之前需利用 STAF 的信号量服务来申请对目标机器的使用权限,在任务结束时再释放该资源,从而确保在执行任务的过程中,目标测试代理不会被其他的任务使用。类 AutoTestJobExecutor 封装了执行某个自动化测试任务的所有细节,执行自动化测试任务的具体过程描绘如下:

- 1) 根据测试任务编号 jobId 从数据库读取任务信息,包括目标机器、任务 XML 文件、入口函数、所有参数值(测试数据)。
- 2) 注册一个 STAF 句柄,若失败转 12)。
- 3) 申请对目标机器的 MUTEX 访问资源,若失败转 11)。
- 4) 记录任务开始时刻,记为 t_1 。
- 5) 向底层 STAF 的 STAX Service 提交任务,若提交失败,更改数据库中该任务状态为“执行失败”,转 10)。
- 6) 更新数据库,更改该任务状态为“正在执行”。
- 7) 向 STAF 提出等待,直到 STAX 返回任务结果 RC 或超

过最长等待时间 t_{max} ,超时则更新数据状态为“等待超时”,转 10)。

8) 记当前时刻 t_2 ,统计任务执行时间为 $t = t_2 - t_1$ 。

9) 若 RC 为 0,更改该任务状态为“成功”,否则更改该任务状态为“失败”。

10) 释放 MUTEX 资源。

11) 注销 STAF 句柄。

12) 结束。

自动化测试计划为一个二元组 $\langle P, T \rangle$, P 指提前规划好的测试任务和测试数据, T 指对应触发器,也是一个普通的 STAX 任务,它说明了执行 P 的必要条件,如时间、测试目标准备就绪等。自动化测试计划监视器负责监控在系统中注册的所有触发器的状态,一旦发现处于激活状态的触发器,即自动提交执行相应的测试计划,算法流程如图5所示。

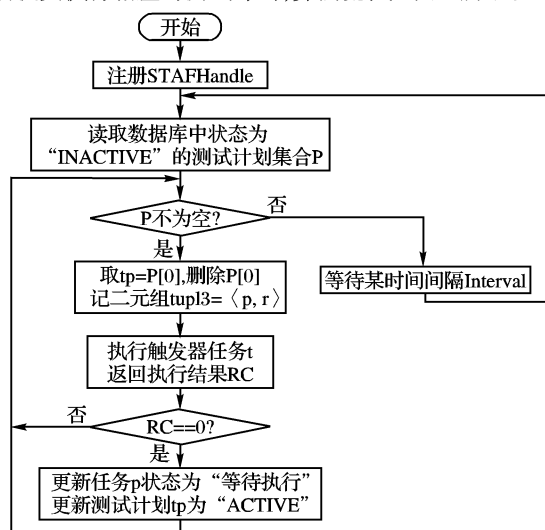


图5 自动化测试计划监视算法流程

1.4.3 实现层

实现层指运行在自动化测试控制服务器最底层的 STAF 进程及其提供的包括 STAX 在内的各种服务,封装了自动化测试控制服务器与各种平台的测试代理间通信、执行各种自动化测试具体操作的复杂性。对于高层来说,实现层可以当作是透明的。

2 基于 STAF 的软件自动化测试系统应用

按照软件开发生命周期的迭代模型,软件测试和软件开发工作是并行进行的,“每日构建”(Daily build)模式作为软件开发的最佳实践之一,为不少软件公司采纳。Tomcat 是一个开源的轻量级的 Java EE Web 服务器,支持多种平台。现以 Tomcat Daily Build 的用户接受度测试(User Acceptance Test, UAT)为例,应用基于 STAF 的软件自动化测试系统来实施自动化测试。首先设 Tomcat UAT 环节包含下列步骤:

- 1) 在各种不同的操作系统平台上检测能否安装成功。
- 2) 对关键配置进行更改,重启 Tomcat 验证是否生效。包括更改端口、配置访问权限、配置数据库连接池等。
- 3) 尝试部署一个应用程序,检测能否正常工作。

应用本文提出的自动化测试系统对以上环节进行了自动化测试,与人工测试的方式同步运行 20 天,对两种模式下完成测试任务过程中平均人机交互时间和出错次数进行了统计, Tomcat UAT 过程中比较结果如表 1 所示。

(下转第 704 页)

2.4 应用示例

假设 ATM 有两个银行团体 (B1、B2), 分别提供 A1、A2 和 A3、A4 共四种方法。根据本文模型, 图 6 给出了 ATM 的连接结构。

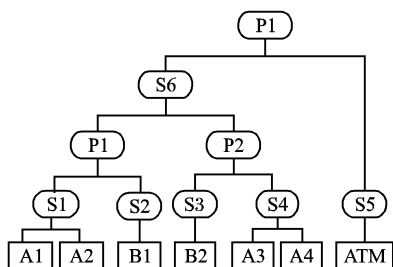


图 6 ATM 连接器结构

首先实现最底层构件, 用构件创建第 0 层子连接器, 最后借用构件和子连接器实现高层连接器。连接构件或子连接器时, 可以用顺序、选择、重复三种操作方式选择构件或连接器, 通过继承父类 Superconnector 获得连接方式, 实例化父类时收到以下参数: tget, op 和 par。tget 指出哪些连接器被连接; op 指出连接器的操作集; par 给出操作过程中的参数集。

为使每个构件的方法得到调用, 因此第 0 层所有构件都要进行连接器连接; 调用子连接器构建高层连接器。系统从最高层连接器开始执行, 经过调用子连接器, 实现构件方法的调用。

第 1 层中, S1 根据用户的需求, 从 A1 和 A2 中选择方法。同样, 通过 S2 从 A3 和 A4 中选择相应的方法。从一个银行中传递数值时需要管道连接器, 因此在第 2 层中, 用 P2 和 P3 分别连接 B1 和 B2。第 3 层中, 选择连接器 S6 从 P1 和 P2 中选择。第 4 层中, 管道连接器 P1 把用户需求和信息传递

给 ATM, 调用 ATM 方法, 并将结果传递给连接器 S3。

例如, A3 中某用户需要从账户中取款。首先, 用户插入卡并输入 PIN 码; 管道连接器 P1 向 ATM 传递信息, 连接 ATM 的连接器 S5 接到 P1 传递的用户信息及取款需求, 调用 ATM 中方法验证用户信息。如果该用户合法, P1 将用户信息及取款需求传递给 S6, S6 选择管道连接器 P2, 连接 B2。P2 用来调用 B2 中验证用户银行分支, 并返回结果; 连接器 S4 选择 A3 并调用取款函数, 最后返回结果。

3 结语

在分析当前构件模型及连接器基本任务基础之上, 提出一个更细粒度的连接器模型。该模型减少构件的复杂度, 使连接器真正成为构件间交互桥梁。最后, 描述了连接器的演化过程, 并给出相关实例。

参考文献:

- [1] WANG GUI-JUN, UNGAR L, KLAUITTER D. Component assembly for OO distributed systems[J]. IEEE Computer, 1999, 32(7): 71-78.
- [2] SZYPERSKI C. Component software - Beyond object-oriented programming[M]. 2nd ed. New York: ACM Press, 2002.
- [3] BURES T, PLASIL F. Scalable element based connectors[C]// Proceedings of SERA, LNCS 3026. Berlin: Springer, 2003: 198-204.
- [4] 许峰, 刘英, 黄皓, 等. 基于软件体系结构连接器的构件组装技术研究[J]. 计算机应用, 2006, 26(4): 836-839.
- [5] 谢文彬, 陈榕. 构件类别的扩展与应用[J]. 计算机工程与应用, 2005, 41(9): 99-101.
- [6] 许毅, 彭鑫, 赵文标. 基于通用连接器模型的复合构件的组装[J]. 计算机工程, 2006, 32(23): 61-63.

(上接第 701 页)

表 1 自动化测试与人工测试比较

测试步骤	平均人机交互 时间/(min/d)		平均出错次数 /d ⁻¹	
	人工 测试	自动化 测试	人工 测试	自动化 测试
开发、调试自动化测试脚本	0	9	-	-
安 Windows XP	8	2	0	0
装 Windows 2003	9	2	0.05	0.05
测 Redhat Linux 9	15	2	0.10	0
试 Solaris 10	16	2	0.15	0.05
关键配置测试	45	2	0.2	0
部署测试	10	2	0.1	0
总计	103	21	0.70	0.10

注: 开发自动化测试脚本的时间成本被分摊到整个测试周期: 180/20 = 9。

由表 1 可以看出, 虽然开发和调试自动化测试任务需要占用一定的时间, 但是整个测试周期都将因此而受益: 首先, 利用自动化测试系统, 用户只用在 Web 界面提交任务和查看任务运行结果, 剩下的耗时、重复的工作均交由机器完成, 甚至可以在不同平台同时执行测试任务, 从而节约了大量的时间; 其次, 由于多平台的复杂性使得人工测试的出错率很高, 自动化测试系统则分离了测试实现和测试数据, 即每次测试运行的都是同一份测试驱动程序, 只是更改了参数值, 这样既提高了测试任务的复用率, 又可使测试工作更加稳定可信赖;

最后, 自动化测试系统自动存储管理所有的测试记录, 用户可以通过 Web 界面方便地进行查询和统计, 减少了人工维护这些历史记录的额外成本。

3 结语

论文研究并实现了一种基于 STAF 的软件自动化测试系统, 实现了测试任务和数据的分离, 提高了测试任务的复用性, 并提供方便友好的交互界面对自动化测试任务、资源、结果等进行统一高效的管理。

目前系统侧重于解决面向单个用户的测试任务的自动化, 测试任务之间相对独立, 而现实工作中测试工作通常由团队完成, 团队成员之间进行的工作存在各种逻辑关系, 如依赖、顺序、并行等。下一步可以在现有系统基础之上, 结合工作流的理念来拓展应用层, 开发出支持多人协作自动化测试任务的测试系统。

参考文献:

- [1] 朱菊, 王志坚, 杨雪. 基于数据驱动的软件自动化测试框架[J]. 计算机技术与发展, 2006, 16(5): 68-70.
- [2] 朱芳, 李曦, 赵振西. 一种多平台自动化测试工具的设计和实现[J]. 计算机工程, 2004, 30(24): 186-188.
- [3] 严少清, 陈革, 万年红. 软件测试自动化管理系统的设计与实现[J]. 计算机工程, 2002, 28(9): 152-218.
- [4] RANKIN C. The Software Testing Automation Framework[EB/OL]. [2000-09-01]. <http://www.research.ibm.com/journal/sj/411/rankin.pdf>.