

分类号_____

学校代码 **10487**

学号 **M201176073**

密级_____

华中科技大学

硕士学位论文

UEFI BIOS 的自动化测试框架

的设计与实现

学位申请人：安 婷

学 科 专 业：软件工程

指 导 教 师：黄立群 副教授

答 辩 日 期：2014.1.7

**A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree for the Master of Engineering**

**Design and Implementation of UEFI BIOS
Automation Test Framework**

Candidate : An Ting

Major : Software Engineering

Supervisor : Assoc. Prof. Huang Liqun

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

January, 2014

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于 ☐ 保密， 在_____年解密后适用本授权书。
☐ 不保密。

（请在以上方框内打“√”）

学位论文作者签名：

指导教师签名：

日期： 年 月 日

日期： 年 月 日

摘要

随着信息技术的发展,自动化无疑是各大企业进行软件测试的必然趋势,虽然互联网上有许多免费的自动化测试框架,但是对于企业级的用户而言,考虑到安全性,以及测试对象的特殊性,尤其是诸如 BIOS 这种需要涉及到很多底层硬件的产品,自主开发自动化测试框架会是一个更好的选择。本系统研究的目的是创建一个在多个 UEFI BIOS 平台提供并发控制的可靠的自动化测试框架。

FiTT (Firmware in-depth Testing Technology,即固件深入测试技术)是一个为 BIOS 开发和测试服务的框架。本系统的硬件开发环境部署在两台能够通过网络端口进行通信的服务器上,采用 Windows Socket 机制实现服务器之间的通信。开发所使用的操作系统为 Window2k8 R2 Server 平台,选择的开发语言是 Python+XML,并调用了许多经过广泛测试的可扩展的开源库,它们为 Python 提供诸如串口处理的定制化的服务;为了实现用同一套代码在不同类型的服务器平台提供并发控制的测试,本系统设计了一个平台代理(Platform Proxy)的架构,将控制平台的工具抽象化,为实际的和虚拟的服务器平台的配置提供一个通用的 API,从而简化了解决方案的集成;并且使用 Django 的技术为用户提供功能丰富的 WEB 服务,包括邮件报告功能;最重要的结合 sikuli 图像比对和 autoItX 等技术实现 Python 模拟鼠标键盘的操作,本课题所实现的系统为 BIOS 开发组实现了一整套完全自动化的测试用例。

本课题从自动化测试技术的发展现状着手,为目标系统做出完整的需求分析,进而从硬件组织和软件结构两个方向进行系统设计,然后着重论述 3 个主要的功能模块的实现方式,包括核心框架的平台加载模块和 WEB 服务模块,以及测试用例模块,最后提出四个性能指标,通过截图展示 FiTT 的执行流程,分析测试的结果,并对系统的可靠性进行评估。

关键词: 自动化测试 平台代理 图像比对

Abstract

With the rapid development of information era and software technology, automation is undoubtedly an inevitable trend for software testing of major enterprise. Although there are many free automated testing framework on the Internet, for enterprise-level users, they need take into account the safety and specificity of the test object, especially products involving a lot of underlying hardware, like BIOS, developing own automated testing framework would be a better choice. The purpose of this study is to create automated testing framework to provide concurrency control and reliable system across multiple UEFI BIOS platforms.

FiTT (Firmware in-depth Testing Technology) is a framework which provides automated testing service for BIOS develop team. The hardware environment needs two servers which can communicate with each other through port. The system is written in Python and XML, and call lots of scalable open source library which have been through extensive testing. It constructed a Platform Proxy architecture providing an abstraction over the tools used to control the platform, Platform Proxy provide a common API which simplify workaroud integration. FiTT use Django technology provides a feature-rich WEB services for testers. Above all, by binding images comparison technology of sikuli and using autoItX to simulate keyboards and mouse, FiTT achieve a suite of fully automated test cases for BIOS develop team.

Generally, in this paper, firstly go through the current situation for automated testing technology, then make a complete analysis for the purpose system, design system organization from both hardware configuration and software architecture, and the focus on the implementation of three major functional blocks, including Platform Proxy block, WEB service block and fully automated test cases block. In the end, the paper raises the four performance indicators, by showing the flow of execution screenshot of FiTT, I analyze the test results, and assess the reliability of the system.

Key words: Automated test Platfom Proxy Sikuli

目 录

| | |
|--------------------------|------|
| 摘 要..... | I |
| Abstract..... | II |
| 1 绪论 | |
| 1.1 研究背景..... | (1) |
| 1.2 研究现状..... | (3) |
| 1.3 课题背景及研究内容 | (6) |
| 1.4 预计涉及的关键技术 | (7) |
| 1.5 本文的主要内容和框架实现 | (13) |
| 2 需求分析和系统总体设计 | |
| 2.1 需求分析..... | (14) |
| 2.2 系统总体设计 | (17) |
| 2.3 交互流程设计 | (23) |
| 2.4 本章小结..... | (24) |
| 3 系统的详细设计与实现 | |
| 3.1 开发环境..... | (25) |
| 3.2 执行框架模块的详细设计与实现 | (25) |
| 3.3 测试用例模块的详细设计与实现 | (39) |
| 3.4 本章小结..... | (50) |
| 4 系统的性能测试与分析 | |
| 4.1 系统性能的评价指标 | (51) |
| 4.2 功能测试..... | (52) |
| 4.3 测试结果分析 | (55) |
| 4.4 本章小结..... | (57) |

华中科技大学硕士学位论文

5 总结与展望

5.1 全文总结.....(58)

5.2 展望.....(60)

致 谢.....(61)

参考文献.....(62)

1 绪论

自动化测试能将开发和测试人员从重复枯燥的手动测试中解放出来，能在降低人力资源成本的同时，保证更可靠更高效的测试，因此，实现自动化毫无疑问是企业进行产品开发和测试的一个必然发展方向。本课题的研究的自动化测试框架 FiTT(Firmware in depth testing technology)来源于英特尔 BIOS 开发组，为基于 Grantley 平台的 UEFI BIOS 服务，因此在展开课题前有必要对本框架服务的对象——UEFI BIOS 做个简单的了解，以便更好地完成本课题要实现的自动化测试框架的需求分析。

1.1 研究背景

一台计算机幕后的工作原理，普通用户并不知道。开发者耗费大量的精力将组成计算机的零散部件整合成一个符合逻辑的整体，而 BIOS 是整个计算机系统最重要的底层系统软件。BIOS 的全称是 Basic Input/Output System，即“基本输入/输出系统”，是一种所谓的“可启动固件”，承担的任务主要包括在开机时初始化硬件和自检，并担任操作系统控制硬件时的中介。

最早的电脑没有可启动固件，用户手动操作电脑前面板上的开关来输入一个启动程序。Gary Kildall 等人在 70 年代提出一个可启动固件的概念，将其视为计算机硬件与操作系统之间的一个抽象层^[1]。随着 IBM PC 机的产生，开发者们应用了一个与 Gary 等人提出的可启动固件类似的扩展概念，叫做 BIOS，即基本输入输出系统，其作用是初始化硬件，检测硬件的运行，最重要的是引导操作系统。后来为了解决 Legacy BIOS 的兼容性会降低处理器性能的问题，英特尔提出了 EFI (Extensible Firmware Interface，即可扩展的固件接口)的构想。其目的在于为下一代的 BIOS 树立一个关于开发的全新框架，EFI 是一套完整的接口规范，而非具体的软件，作用在操作系统与平台固件之间，用来定义很多关键的数据结构和系统服务^[2]。可扩展固件接口(EFI)最初是由英特尔开发，并在 2005 年将其交给 UEFI 论坛

进行推广与发展，后来名称改成 Unified EFI(UEFI)，即统一的可扩展的固件接口。

随着计算机硬件和操作系统的不断发展，BIOS 的发展经历了 Legacy BIOS——EFI BIOS——UEFI BIOS，从 16 位汇编到 C 语言，从静态链接到动态链接，从内存固定编址模式到实模式，从 PC 厂商各自为阵到统一的标准，可启动固件的复杂性与日俱增，目前基本可以与底层硬件及其加载的驱动，软件的复杂性并驾齐驱。传统式 (Legacy) BIOS 已经成为进步的绊脚石，对于新的 UEFI，按照传统 BIOS 的观点，未来计算机的发展会是一个“没有特定 BIOS”的时代^[3]。

本课题所要实现的自动化测试框架 FiTT 就是为 UEFI BIOS 开发和测试服务的。它要实现的测试用例非常多，包括主板上电，POST 启动，OS 引导等等，归根到底，就是要测试 UEFI BIOS 的 Boot Flow，主要分为如下几个阶段：

(1) SEC Phase: 这是 UEFI 启动过程的第一个阶段，是系统上电/重启的入口，处理平台重新启动相关的所有事件，这个阶段还会创建供系统内存初始化使用的临时内存区，验证 PEI Foundation 的安全性等；

(2) PEI Phase: PEI 的全称是 Pre-EFI Initialization，本阶段需要承担初始化的工作，包括 Memory 和一些必要的 CPU、Chipset 等。PEI Phase 还要精简还没有压缩的代码，确定启动路径，初始化并描述最小的 System RAM；

(3) DXE Phase: Driver Execution Environment，即驱动执行环境阶段，DXE Phase 由以下三部分组成来完成平台的初始化：DXE Core 模块产生一些提供引导服务的模块，DXE Dispatcher 负责发现并按照正确的顺序执行 DXE Drivers，DXE Drivers 负责 CPU，Chipset 的初始化，为系统组件和启动设备等提供系统概要；

(4) BDS Phase: Boot Device Select，即引导设备选择，BDS 阶段要尝试加载列在环境变量上的驱动，初始化环境变量的控制台设备，尝试从启动设备列表启动；

此外，还要测试 EFI Shell，引导到 OS 之后的 Run Time 阶段以及设计重启或者休眠的 AL 阶段^[4]。

众所周知，企业级的代码需要涉及到比较复杂的版本管理，BIOS 开发组每天都会 Build 新的 BIOS image，由于 UEFI 的 BIOS 有着非常庞大的测试用例 (Test Case)，通常 validation team 完成一个完整回归测试可能需要几周的时间，基于这种

情况，BIOS 开发组的开发人员必须至少保证一些基本的 feature 每天都能够 Pass，比如 BIOS 的更新，PowerOn，Boot 到 DOS/EFI Shell/OS，OS 的安装，simics 模拟硬件测试，以及一些基本 feature 的测试，之前这些测试都是由开发人员手动完成，需要耗费大量的时间，比如在一个平台上做基本的 Smoke Test 大概需要耗费两个小时，Simics 测试需要一个半小时，而目前 BIOS 开发组需要覆盖至少两个平台，而这些测试每天都是重复操作^[5]，基于这种情况，本课题需要实现一个基于固件摄入测试技术的自动化测试框架 FiTT，将开发人员从沉闷冗余的测试中解放出来，而且不依赖于人为因素更加高效。

1.2 研究现状

自动化毫无疑问是企业进行产品开发和测试的必然的发展方向，而成熟自动化测试框架是保证测试用例可靠执行的基础。随着测试技术的发展，很多公司都发布了自己所开发的自动化测试工具。通过调研发现，目前比较流行的自动化测试工具或解决方案主要有如下几种：

（1）以 QTP 为核心的框架

QTP (quicktest Professional) 是最目前常见的用于自动化测试的工具之一。QTP 遵循关键字驱动的思想来简化对测试用例脚本的处理，提供符合应用软件环境的测试用例的自动化，主要是功能测试以及回归测试。普通用户预先录制屏幕上的操作流程，QTP 自动生成测试用例。对于专业人员而言，改变调试环境和内置的脚本，能够完全控制测试以及测试对象的属性^[6]。目前不少公司都是以 QTP 为本企业自动化测试框架的核心。以 QTP 为核心的自动化测试框架主要优点在于适用性好，操作简单，脚本也通俗易懂，甚至无代码基础的测试人员也可以加入脚本录制和调试。不过，对于专业的测试人员而言，QTP 有一个很明显的缺点主要是基于关键字的框架在灵活度方面表现很差^[7-8]。

（2）RFT

IBM 的 Rational Functional Tester 为基于 Web 的应用程序提供录制和回放的功能，它基于 Java、.NET 的对象技术。记录脚本时，RFT 会为被测的应用程序自动

建立一个测试对象地图，并且允许测试人员在记录过程中将验证点插入到脚本中，从而对应用程序建立过程中对象所处的状态进行确认^[9]。它同时适用于普通测试人员和专业图形化用户界面开发人员。普通测试人员可以直接使用 RFT 将复杂的任务进行简化；专业测试人员还可以用标准化的脚本语言，对复杂功能进行个性化定制。为了解决对象识别问题，RFT 采用动态查找对象的方法，所以比 QTP 框架灵活度要高。此外，还允许测试人员用其他方式代替基于模式的识别，比如一个数值范围或正则表达式。但是，RFT 不仅价格高昂，在使用起来，效率也比较低，网上关于该框架的问题解决方案非常少^[10-12]。

（3）开源框架

目前网上有一些发展得越来越成熟的开源的框架，目前比较热门的有 Selenium, ant, testng, hudson，很多自动化测试框架设计的核心思想都是模拟，从终端用户的角度出发，编写模仿手工操作的脚本来测试应用程序。他们针对的产品略有差别，功能性和稳定性各有千秋，以 Selenium 为例，因为更加入了 xpath 加上配合 IDE 进行定位等，它比 RFT 更加灵活，而且资料更全面。参考开源框架可以方便用户根据自己的产品需求进行二次开发以实现定制化的功能。但是开源框架对测试人员的编码水平有一定要求，大多数人执行这个框架前需要有较长时间学习适应^[13-15]。一般开源项目开发时间比较短，由于使用人群还不是很普及，不如 QTP 之类的成熟框架完善，但是我们可以期待其未来发展。

以上就是目前对于实现自动化测试常见的解决方案，在选择如何实现项目组的自动化测试之前，有必要对其核心技术做个深入了解。目前比较流行的自动化测试框架技术大致可以归为四类：

（1）数据驱动测试框架（The Data-Driven Testing Framework）

数据驱动测试框架读取数据文件的输入和输出数值，比如数据池，ODBC 源等，然后赋给变量，包括工具自己捕获的和测试人员在代码中定义的变量。其主要做法是，无论是输入数值，还是输出验证数值，都设为变量，编写的测试脚本代码要实现四个功能模块，包括读取数据文件，记录信息日志和记录测试状态，以及实现贯穿程序的导航^[16]。

数据驱动测试框架是所有测试架构中最简单的一种，它实现的仅仅是在脚本中分离测试数据^[17]。它的优点在于可以单独维护测试数据，但是缺点非常明显，被测程序的变更会带来很多额外的工作量，维护成本最高。

（2）测试脚本模块化框架（The Test Script Modularity Framework）

测试脚本模块化框架下，开发人员要为测试目标应用程序的模块和函数编写独立的脚本，如果想要实现特定的测试用例，可以将这些小脚本用分层的方法合并成更大的测试。这个框架最容易熟练掌握。它采用了一个很好的编程策略，为部件创建一个抽象层将应用程序的其他部分隐藏，使用抽象或封装的原则，可以增强测试框架的可扩展性和可维护性^[18]。

测试脚本模块化框架的优点很明显，但也存在一些问题，例如，自动化测试开发工程师要承担大部分的变更引起的工作量；没有对控件识别和业务逻辑进行很好的抽象封装，尽管他们属于不同区域^[19]。

（3）测试库构架框架（The Test Library Architecture Framework）

测试库构架框架类似于前一个测试脚本模块化框架，但是它分离的不是脚本，而是正在执行测试的应用程序，将其分为过程同函数。它要求创建一个库文件，用来包含被测应用程序的模块，函数和零件。测试用例脚本能够直接调用这个库文件。框架的开发人员负责编写测试脚本，维护业务逻辑，测试人员只需维护测试数据。

测试库构架框架有两个主要的优点：①如果只有被测试系统的某一层改变，只需要对应的人员处理变更维护；②将业务逻辑和控件识别操作分别进行抽象。但是，变更引起的工作量还是需要自动化测试开发工程师来承担^[20-21]。

（4）关键字驱动或表驱动测试框架（The Keyword-Driven or Table-Driven Testing Framework）

关键字驱动（或者说表格驱动测试）框架需要对两个方面进行开发：用来运行的自动化工具，以及独立的应用程序和脚本的数据表和关键字。关键字驱动测试类似于模拟手工操作。关键字测试的表格同测试用例的详细指引会描述应用程序的功能特性^[22]。

前面提到过以 QTP 为核心的框架是基于关键字驱动的典型代表。关键字驱动

或表驱动测试框架在很大程度上减轻了自动化开发工程师的工作量；普通测试人员也能很好的维护本模块的测试数据和用例。不过，较高的框架的抽象程度对开发阶段人员的能力也要比较高^[23]。

面上有很多免费或者收费的框架采用上面提到的四个类型的自动化测试框架技术，比如 IBM 的 RPT，HP 的 LoadRunner，但是对于企业级的用户而言，考虑到安全性，以及测试对象的特殊性，尤其是诸如 BIOS 这种需要涉及到很多底层硬件的产品，自主开发自动化测试框架会是一个更好的选择^[24-25]。

在本系统中，我们的框架并不是完全选择了哪一种框架技术，而是根据需要，而是综合了好几种框架技术，测试脚本模块化和测试库构架更是能在很多场景下得到体现，比如，在 FiTT 中，每一个大的 Stored job 都是由许多个独立功能的 Task 组合在一起，我们可以根据需要随意地组合 task，构成不同功能的测试，极大地提高了代码的重用性。再比如，将一些常用的功能函数封装成一个库，方便之余让代码看起来也非常简洁。

1.3 课题背景及研究内容

本课题来源于英特尔数据中心的一个实际项目，该项目主要有 5 个人参与开发，本框架服务于围绕 Harswell 处理器的服务器平台上的 BIOS 芯片。事实上，自动化测试框架的开发可以分为三种类型，第一种是负责纯的框架的构建，第二种是负责测试用例的执行，第三种介于两者之间，而我的角色属于第三种，我的工作内容包括 BIOS 开发组的自动化测试框架的定制化，功能的完善，代码兼容性的提高以及 BIOS 开发组测试脚本的实现。

本课题的研究内容包括如下：

(1) 软硬件环境的构成。BIOS 芯片的测试需要涉及到非常多的底层硬件，而且独立于操作系统，因此如何实现服务器平台上芯片的自动化测试框架，会有非常多的考量。对于硬件环境，本系统至少需要三台服务器，两块服务器主板，两套 RSC2+KVM 的配置；对于软件环境，FiTT 自动化测试框架大部分的代码都是通过 Python 实现的，涉及到了很多提供服务的软件和库，比如 PIL，Sikuli，Pyserial，zope 等。

(2) 系统核心框架的设计。核心框架 core 是自动化测试平台的大脑，其重要性不言而喻。如何控制整个流程的执行，各个模块以何种方式协作，文件的组织形式，框架要实现哪些主要的功能以及如何实现等，都需要在核心框架中设计清楚。

(3) 主要功能的详细设计和实现。这一部分会主要介绍 FiTT 自动化测试平台的一些主要功能的算法和具体实现，是对核心框架价值的体现。

1.4 预计涉及的关键技术

本系统的开发基于两台能通过端口进行通信的服务器。基于 Window2k8 R2 Server 平台，我选择的开发语言是 Python+XML，其中 Python 用于测试用例具体编码的实现，XML 用于实现参数信息的 descriptor 文件。

本课题的核心问题主要是执行框架和核心测试用例的实现，执行框架的难点在于为了实现并发控制不同服务器平台的测试，系统需要提供一个将实际硬件平台的硬件抽象出来的平台代理；在此基础上，本自动化测试框架还要提供一个供用户访问的 GUI 界面，最后要实现许多 BIOS 测试的测试用例，这些测试用例需要。预计本文的关键技术有如下几个：

1) Django 技术。本课题要实现一个为测试开发人员提供一个同自动化框架交互的 WEB 访问接口^[26]。

2) Platform Proxy。本课题研究的对象是一个企业级的框架，它需要为不同的硬件服务器平台提供并发控制的服务，因此有必要设计一个平台代理的架构将底层硬件的平台抽象，设计一个 Board 基类，然后针对不同的实际平台通过继承展现多态。

此外，对于测试用例的实现还有很多细节的处理，针对需求，本系统主要用到如下技术方案来提供服务。

(1) 通过 Sikuli 进行图像对比

FiTT 使用 Sikuli 做图像比对，以实现自动化测试中一些关键的 checkpoint 的判断。用户只需掌握基础的 Python 语言知识，图形用户界面的截图元素封装了很多功能函数，它们可以帮助测试人员自动的完成编程任务。

Sikuli 是一种基于 Jython 的新型的图形化编程技术，既是一种脚本语言，也是一种集成开发环境。Sikuli 的意思是“上帝之眼”，它基于图像检索技术，Sikuli 在过程预录制之后能让电脑能像人眼一样“看”这个“真实世界”^[27]。

Sikuli 预先实现一些截图的 GUI 功能函数，比如 find(), exist(), 点击函数名，它会自动被插入到编辑区，若需使用该函数的截图作为参数，则自动转入屏幕截图状态，这种方式可以让我们在编写脚本的过程中时，不需要获取 Web 内容对象，也不必关心应用程序相关的 API。执行脚本时，根据图像检索算法，Sikuli 对当前屏幕中预录制的控件进行分析和匹配，然后执行脚本所写的键鼠操作。

Sikuli 赋予脚本人的视角，让测试工具不仅可以获得后台的接口数据及其返回值，而且能够捕捉真实展现的 GUI。基于 Sikuli 的 GUI 自动化在很大程度上降低了对应用的内部程序的依赖；忽略获取或者操作非标准控件和标准控件的差异性；适用的操作系统需要有图形用户界面。Sikuli 的扩展性很强，因为它采用 Python 语法模式，此外，开源也让 Sikuli 本身获得了更多的发展^[28]。

Sikuli 的优点非常明显，但在实际应用中也会存在一些会直接影响到我们测试脚本的编写的问题：

① 兼容性不好，太依赖屏幕截图，在不同的操作系统上，不同的浏览器中，甚至是不同的显示器分辨率下，图形源文件是存在差异的，这会对自动化测试跨平台的能力造成障碍。

② 脚本易受外界干扰。由于 Sikuli 太过于依赖于实时显示的桌面，对于截图的检索过程，如果出现逻辑定义之外的，诸如弹出窗口一类的意外界面，无论是遮挡还是切换焦点，程序判断都会受到影响。

因此，如果想要通过 Sikuli 独立完成较大的自动化测试项目，肯定会遇到一些问题，不过如果将它作为现有测试工具的一个有效补充，发挥 Sikuli 的优势，还是非常方便的，本系统就使用 Sikuli 来进行图片对比，这样的一种实现策略使得自动化测试脚本和手工测试用例的差距变得越来越小，从而让测试用例自动转化变得非常简单。

2) 通过 RSC2+KVM 控制服务器的电源和跳线

在英特尔公司内部,开发人员使用 RSC2 来控制服务器主板的电源和跳线。RSC2 的全称是 Remote System Controller 2,即远程系统控制器(以下简称 RSC2),通过这个设备,用户可以在任何地方通过互联网远程控制某个测试中的 SUT (System Under Test)。RSC2 能够控制 SUT 基板上的 8 处跳线,比如 Clear CMOS Jumper, BMC Force Update Jumper 等,并提供两路交流供电的功能。RSC2 有一个 USB 接口可以在服务器和测试主板之间共享,执行 AD/DC 动力循环(Power Cycling)测试,它可以作为服务器 HOST 和测试主板 SUT 之间通信的介质,比如可以将文件从 HOST 拷贝到 SUT,甚至能够实现远程安装操作系统^[29]。

RSC2 是一套硬件设备,它有对应的客户端软件,RSC2 客户端软件需要 .NET 3.5 SP1 Framework 的支持,下载必需的软件程序,运行安装程序,驱动将会被同时安装入系统中。在安装完毕后,RSC2 Client 同时提供了很多 Python Library,通过这些库,开发人员可以通过可编程接口连接 RSC2 的硬件来控制 and 监控测试主板的按钮,跳线和 LED 灯。

RSC2 和 IP-KVM 配套使用,可以供远程的用户来控制一个图形化的界面^[30],KVM 映射的界面就如同一个窗口一样显示在桌面。这一过程的实现需要安装 JRE,通过访问 KVM 的 IP 来获取 SUT 所处的 OS 的界面,比如 BIOS Setup, EFI Shell, DOS 和其他的 OS。

3) AutoItX 实现 Python 模拟鼠标键盘

AutoIt v3 是一款免费软件,可以用来编写一些脚本语言,它的作用是在 Windows 图形化用户界面中,通过对键盘鼠标的操作进行模拟,从而帮助实现自动化^[31]。本课题实现的系统,在很多地方都使用了 AutoItX 的库函数,比如 send()。

在如何实现 Python 模拟键盘和鼠标的功能,我做了很多的研究工作,一般而言,比较通用的操作可以有三种方法。

(1) 通过使用 win32api 模块和 win32con 模块,python 可以方便地调用 keyboard_event 函数操作键位码来实现键盘输入模拟。比如按下然后释放 ctrl+v

```
import win32api  
  
import win32con
```



```
win32api.keybd_event(17,0,0,0) #ctrl 键位码是 17
win32api.keybd_event(86,0,0,0) #v 键位码是 86
win32api.keybd_event(86,0,win32con.KEYEVENTF_KEYUP,0) #释放按键
win32api.keybd_event(17,0,win32con.KEYEVENTF_KEYUP,0)
```

(2) 使用 `windll.user32` 实现鼠标模拟。

Python 可以调用 windows 下的 dll。`ctypes` 是一个基于 Python 的模块，它的作用是实现在 Python 中对 C 语言的数据类型进行新建和操作，可以在动态链接库中给 C 函数传递参数^[32]。`Ctypes` 支持多种操作系统。下面介绍如何调用 `ctypes` 模块实现一个简单的‘Hello world’。

```
from ctypes import *
MessageBox =windll.user32.MessageBoxA
MessageBox(0, '你好， Ann! ', 'Your first programmer', 0) # 调用函数
```

这段程序的结果是弹出一个 windows 的消息框。在对 `windll.user32` 进行操作的时候，程序会自动加载 windows 动态库：`user32.dll`。

操作鼠标常用到的函数有：

取得鼠标位置，`get_mpos()`

移动鼠标，`set_mpos()`

鼠标左键点击，`move_click()`

(3) 借助 `autoItX` 来实现 python 模拟鼠标键盘。

通过引入 `win32api` 和 `win32con`，可以让 python 实现单个的按键，但是对于复杂的字符串或者较长的命令，用 `win32api` 来处理显得不是很方便，可以借助 `AutoIt` 来实现。安装 `AutoItX` 后需要进行注册，将 `AutoItX3.dll` 文件拷贝到 window，在命令行中执行如下命令：

```
regsvr32.exe AutoItX3.dll
from win32com.client import Dispatch
```

4) Django 技术提供 Web 访问接口

作为一个开源的框架，Django 基于 Python，遵循 MVC 的设计模式提供 WEB

服务。MVC (Model View Controller) 采用的软件设计模式很典型, 即分离业务逻辑和数据, 前提是将业务逻辑封装在部件中, 用户同界面之间进行数据交互的行为能在不重写业务逻辑的条件下被改进和实现个性化定制^[33]。

Django 基于 MVC 的设计的特点有如下几方面: ①对象关系映射 ORM (object-relational mapping) 连接模型和关系数据库, 数据库 API 的使用变容易; ②使用正则表达式匹配 URL 的方式没有框架的特定限定, 程序员可以轻易实现个性化服务; ③采用模版系统分隔内容、设计和 Python 代码, 语言可继承; ④有丰富的表单模型; ⑤不需要在人员管理的创建和内容更新上花费太多时间, 这一部分可以由 Django 自动完成^[34-35]。

Django 框架的 URLconf 机制匹配 URL 是通过正则表达式, 匹配之后再调用对应的函数, 从而实现 MVC 控制器部分。URLconf 并不限制 URL 的风格, 开发人员可以设计个性化的 URL: 传统的, RESTful 的, 或者是另类的。控制层由 Django 自动完成的最大好处降低了开发人员的工作量, 程序员编写比较少的代码就可以完成很多事情, 提高了工作效率^[36]。

将控制器交给框架自动处理, 使得开发的重点可以放在模型 (Model)、模板 (Template) 和视图 (Views) 上。模型是数据存取层, 负责所有设计数据的事务, 比如存取数据, 定义数据之间的关系等; 模板是表现层, 决定在文档或页面中的显示方式; 视图是业务逻辑层, 作为模型与模板之间的桥梁, 负责存取模型及调取恰当模板的相关逻辑^[37-38]。

可以看出, Django 视图并不处理用户输入, 它只决定展现哪些数据给用户, 如何展现这些由视图所指定的数据, 则交给 Django 模板决定。简单来说, Django 将视图进一步拆为两个模块: Django 视图和 Django 模板, 它们分别决定“展现什么”和“如何展现”, 这种思想使得 Django 不受限于内置模板, 而是根据需要可以随时更改^[39]。

本课题研究的自动化测试的框架 FiTT 就是利用 Django 提供 Web service。其具体的工作机制如下:

(1) 启用 Django 服务器要执行 `manage.py runserver` 命令, 系统会自动加载本

目录下的配置文件 `settings.py`。`settings.py` 包含了项目的配置信息，比如，`ROOT_URLCONF`，它告诉 Django 的 `URLConf` 要调用哪个模块，一般默认 `urls.py` 工作机制。

(2) 访问 `url` 时，根据 `ROOT_URLCONF` 的设置，Django 装载 `URLConf`。

(3) 接着，按顺序逐个匹配 `URLConf` 中的 `URLpatterns`。如果找到符合的对象就会调用对应的视图函数，并把 `HttpRequest` 对象 `request` 设置成第一个参数。

(4) 最后，`view` 函数会返回一个 `HttpResponse` 对象^[40-42]。

5) WAIK 实现 OS 无人值守自动安装

本自动化测试框架有一个非常重要的需求是实现 windows 无人值守的自动安装。这里用到了微软提供的一个工具 **WAIK**，它的全称是 **Windows® Automated Installation Kit (AIK)**，即 Windows 自动安装工具包 (**Windows AIK**)，它是一组用于支持 Windows 操作系统的配置以及部署的工具^[43]。

BIOS 开发组为了验证 **baseline** 的 BIOS 对 windows 的支持，每天都需要验证 Windows 是否能够正常安装，为了实现 Windows2012 无人值守的自动安装，本自动化系统可以借助 **WAIK** 来实现。其实现原理是用 **WAIK** 软件来构建一个定制化的应答文件，将这个应答文件保存在 U 盘里面，放在主板上，安装系统的时候，光盘会自动去搜索应答文件，从而载入具体的参数设置信息，在安装时，用户不需要手动选择。

应答文件是一个 `.xml` 的文件，它记录着安装过程所需的所有参数信息。简单的答案文件包括欢迎使用的自定义和基本的安装配置信息。还可以载入特定的驱动程序或应用程序。“Windows 映像”窗格的“组件”节点会包含可用的设置。展开列表，通过右键选择正确的配置阶段，将系统所需的组件，比如语言，分区选择，密码设置等，添加到答案文件中的指定配置阶段，在安装的过程中，Windows 在不同的配置传送中安装操作系统的不同部分^[44]。

值得一提的是，无人值守的安装方式中，Windows 安装中的 `setup.exe` 通过两种方式来调用应答文件：明确指定和模糊搜索。这两种情况下应答文件的保存是有区别的：如果采用明确指定的方式指定应答文件的位置和名称，那么应答文件名可以随

意命名；如果采用模糊搜索的方式，必须将应答文件命名为“Autounattend.xml”。在运行 setup.exe 安装 OS 时，系统会自动去检测所有 detect 到的设备的根目录。

1.5 本文的主要内容和框架实现

本文各章节的主要内容为：

第一章绪论部分首先介绍了课题产生的背景，为需求分析做准备，然后通过自动化测试框架技术的研究现状，探讨适合本公司项目的技术方案，并对课题背景和研究内容作了简要的介绍，预计本系统可能涉及到的关键技术；

第二章主要是需求分析和系统总体设计，只有明确需求才能完善设计，在需求分析模块将对 BIOS 的功能做更详细的介绍，以便实现后面的测试用例。在系统设计模块，会从软硬件架构两方面详细介绍 FiTT 的核心框架的设计，主要包括硬件架构，软件组织结构以及整个框架的运作流程；

第三章是主要功能的详细设计和实现。这里会介绍本系统的开发环境，然后透过代码，从执行框架和测试功能两个方面来介绍 FiTT 框架所实现的各个功能模块，包括主要算法，以及其实现的技巧。本章主要介绍了 FiTT 的平台加载模块，WEB GUI 模块，测试用例的实现模块，主要有 FetchBinary，FlashUpdate，BootToDOS/EFI/Win2012，RSC2 的控制等等，并且提供 GUI(图形化用户界面)和 Command line(命令行)两种模式供用户使用；

第四章是系统性能测试和分析，本章给出了四个评价系统性能的指标，通过一个典型的测试执行的流程，向大家直观展示 FiTT 实现的功能，并根据四个指标分别对 FiTT 进行结果总结和性能分析。

第五章是对本文的工作的总结，并对 FiTT 自动化测试下一步的优化和完善提出意见。

最后是致谢和参考文献。

2 需求分析和系统总体设计

本章会首先对 FiTT 整个自动化测试框架做完整的需求分析，然后从硬件环境和软件模块架构两方面做一个总体的概要设计，最后对测试人员同 FiTT 交互的流程设计做个简单介绍。

本文档使用了如下四个常用缩写词用来描述 FiTT 的几个主要设备：

RSC2——Remote System Controller II，通过 RSC2 连接 ITP HOST 和 SUT，可以控制服务器主板的电源以及板载的一些跳线模式；

ITP HOST——保存测试脚本的服务器，通过 USB 接口控制 RSC2 的主机系统。该主机中安装的 OS 必须支持远程桌面功能；

Local Server——控制整个 FiTT 执行流程的服务器，它和 ITP HOST 通过 7654 端口进行通信，用来发送执行任务的指令，保存测试结果等。

SUT——测试中的系统。这就是 RSC2 通过跳线控制的测试中的系统。

2.1 需求分析

2.1.1 FiTT 系统概述

FiTT 的全称是 Firmware in-depth Testing Technology,也就是固件深入测试技术。它是一个为 UEFI BIOS 开发和测试提供服务的框架。FiTT 的目标在于：保持和提高 BIOS 发布的稳定性，减少沉闷重复的测试，FiTT 执行起来非常简单，在开发的过程中，FiTT 能够做到最大化代码重用。本系统研究的最终目的是创建一个在不同 BIOS 芯片项目通用的框架。

FiTT 是由 Python 编写的，并调用了许多经过广泛测试的可扩展的开源的库。FiTT 基于开放的标准和良好的文件记录的接口。它运用 Web2.0 的接口允许 web 用户访问。FiTT 提供了两种方式供用户提交任务：WEB GUI 和 Command Line。

2.1.2 开发背景

FiTT 自动化测试框架是为英特尔 BIOS 开发和测试部门服务的一套框架，是一个企业级别的分布式架构，希望能够部署在不同型号的服务器平台上。主要用来测

试 UEFI BIOS 的一些基本功能和特性。

2.1.3 系统主要功能模块

UEFI BIOS 想要实现完整的自动化测试需要考虑 4 个要素：开发测试人员，执行框架，测试脚本，测试专用的自动化测试工具。首先必须得有一个提供自动化的执行测试的框架，它负责对测试过程进行管理，包括测试执行的管理和测试报告的管理等几个部分，在整个测试流程里面，自动化框架协调个功能模块的工作流程，管理和追踪测试过程，最后向测试人员通过邮件等方式报告测试反馈的结果。其次是测试用例，它是实现 UEFI BIOS 自动化测试平台的最终目的，执行框架也是为测试用例脚本而服务的。本系统从执行框架和测试脚本两个方面来看，可以把主要功能模块进行如下划分，如图 2-1 所示。

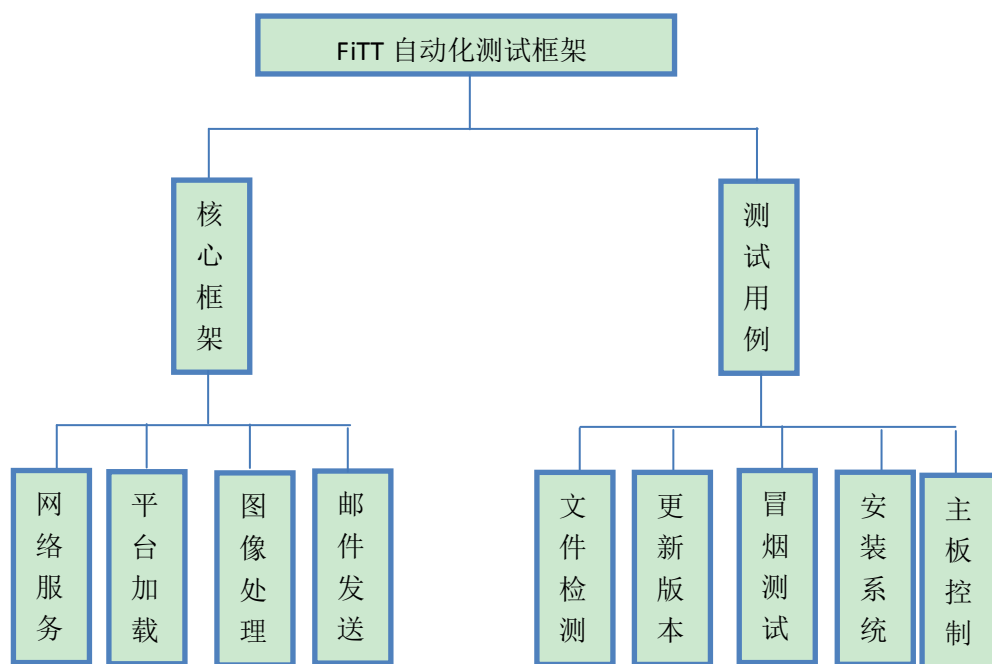


图 2-1 FiTT 的功能模块组织图

下面分别对各个模块要实现的功能进行简要介绍：

网络服务（Web service）模块：WEB GUI 是测试人员同 FiTT 自动化测试执行平台交互的窗口。本系统结合 Django 的技术给用户提供了一个非常优美的 GUI 界面，并且包括丰富的窗口功能，其中，DASHBOARD 界面向用户展示了首页的信

息包括一些欢迎的信息, FiTT 的版本信息, 最近一次执行的测试用例的信息, 所有执行过的测试结果的统计信息等。REMOTE HOSTS 主要提供目前 FiTT 控制的 SUT 的详细信息, CREATE JOB 和 STORED JOB 是测试人员提交任务的窗口。WEB 服务模块还提供供管理人员负责权限处理的界面。

平台加载模块: Platform Proxy 这个 API 是为了简化 task 的开发而设计的一个提供基于实际硬件的一个抽象的 API。Platform Proxy 的目标有如下几个: ①通过一个统一的系统替代传统的控制类; ②为 platform configuration (平台配置) 提供一个通用的 API; ③为状态和配置提供永久存储; ④允许对服务器主板进行具体的代码和任务的并行开发; ⑤简化了解决方案的集成; ⑥为实际平台和虚拟平台提供一个单一的视角。

图像处理模块: 图像处理是自动化测试中至关重要的一个环节, 它需要代替人眼在一些关键的检验点作出判断并给出执行指令。本系统使用 Sikuli 进行图像比对, 并且用到了 PIL(python image library)来进行图像处理, 比如截图。

邮件模块: HOST 在执行完测试后, 需要将测试结果反馈给 Local Server, Local Server 在搜集测试的结果之后将邮件发送给测试人员, 测试人员只需查看邮件就可以得到测试的结果。

新的文件检测模块: 为了实现完全自动化测试, 在 FiTT 的框架里面, 我们加入了一个 BIOS 检测的模块。BIOS 开发组每天都会 build 新的 BIOS image, 将其放在某个服务器上面, 这个新的 BIOS 检测模块会永真执行, 去检测指定服务器上的某个文件夹是否有新的 BIOS image 产生, 如果有, 就会启动一个执行 Daily Smoke Tests 的进程, 否则, 继续等待新的 image。

版本更新模块: 对于 daily 的 Smoke Test, BIOS 自动更新模块就显得非常重要。这个模块出现在新的 BIOS 检测模块之后, 在收到指令后, FiTT 会去服务器上自动抓取新的 BIOS image, 然后拷贝到本地文件夹, 通过 RSC2 的 U 盘映射到 SUT 上, 然后启动到 EFI Shell 通过专门的测试工具进行 BIOS 的更新。

冒烟测试模块: 冒烟测试也叫 Smoke Test, 是每个企业级别的产品开发不能缺少的一块, 对于开发人员来讲, 通过一些小的冒烟测试, 能够保证新 build 的产品可以正常的使用。在 BIOS 的 Smoke test 里面, 我们主要包括了几个小的测试 Power On, Boot To DOS/EFI Shell/Windows 等。

OS 无人值守的自动安装模块：由于 BIOS 是操作系统与底层硬件之间的桥梁，对于普通用户来讲，BIOS 的最大作用就是对操作系统提供 driver 的支撑，。在本系统中，我们用到了微软提供的一个 WAIK 工具，通过定制化的应答文件，结合 FiTT 的脚本实现了 OS 无人值守自动安装功能。

主板控制模块：在 BIOS 的自动化测试里面，我们通过 RSC2 来控制服务器主板的电源以及跳线，RSC2 提供了一些 python 的扩展库，调用这些库函数来控制 SUT。由于在很多测试用例里面都涉及到 RSC2 的控制，而且其相似性很高，为了提高代码的重用性，我们把这一块独立成一个类似于组件的模块。

2.2 系统总体设计

本模块将从硬件架构和软件模块两方面对系统体系结构进行设计。

2.2.1 硬件架构

FiTT 的物理硬件组成主要包括三个部分：分别用于做 Local Server，ITP Host 的两台服务器以及对应的测试主板。此外，还需要用于控制测试主板电源的 RSC2 和控制视频输出的 KVM。如图 2-2 所示。

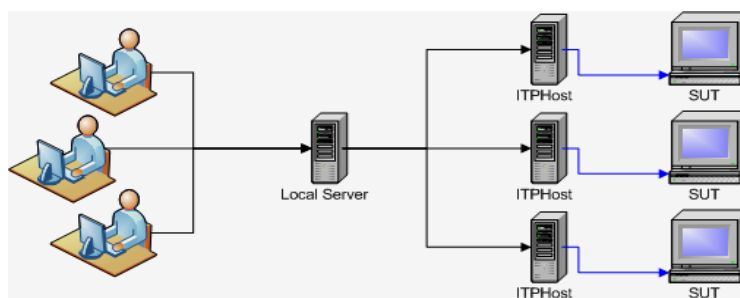
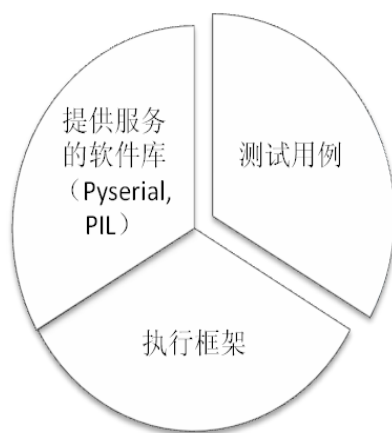


图 2-2 FiTT 的物理组成

从图 2-2 中可以看出，FiTT 是一个多任务多用户的框架，一台 Local server 可以对应多台 ITP Host,每台 ITP Host 对应一台 SUT。Local server 和 ITP Host 是两台独立的服务器，通过网络端口 7654 进行通信。Local Server 控制执行分布式部署的各个 ITP Hosts,控制执行流程,保存结果,处理资源或者与安全相关的任务。ITP Host 存放真正的 Python 的 task 脚本，与 Local server 进行通信，向 SUT 发送执行任务的指令。SUT 就是测试主板，通过 RSC2 与 ITP Host 相连。

2.2.2 软件模块结构

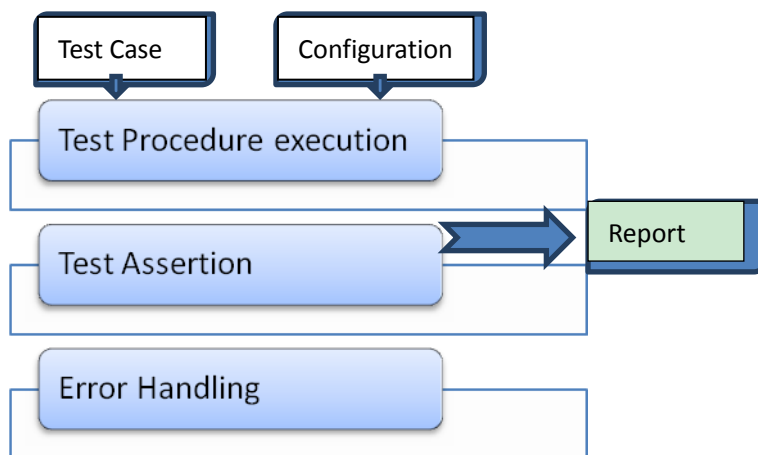
抽象来看，FiTT 的框架主要由三个部分组成:FiTT 的执行框架，FiTT 所涉及到的提供服务的一些软件或者库,比如 PythonITP,RSC2,第三个部分就是根据项目需要设计的 Test Case。如图 2-3 所示。



下面分别对图中的各个模块进行详细的设计。

(1) 执行框架

执行框架是自动化测试平台的核心部分。它需要协调各个自动化测试用例的工作，控制整个测试的流程，追踪并处理测试的结果，其组成如图 2-4 所示。



在测试用例开始执行后，系统首先进行初始化，读取之前的配置和预定义的宏变量，在测试执行的过程中，FiTT 主要有三个功能模块在运行：测试过程执行模块，中断模块和错误处理模块。

测试过程执行模块控制整个测试的流程，协调各个测试用例有序工作，管理并追踪测试过程，最后报告测试结果，是最核心的部分。

测试中断模块负责中断处理，在某个测试用例出现错误或者执行失败后，FiTT 需要能够中断测试，反馈错误结果，继续后一个测试。

错误处理模块负责容错机制，在发生异常时能够捕获，对程序中的错误能够进行正确分析，并且能作出相应的处理，比如抛出异常，或者将分析结果反馈给测试人员，并且给出提示信息以供测试人员参考。

为了实现对不同型号的非 OS 的硬件平台进行自动化测试，FiTT 需要定义一个通用的 API,并且有一些规格文档定义了一些通用的方法，规定 test case 如何编写，如何产生和保存结果。Platform Proxy API 是在 FiTT0.98 中引进的一个新的 API,这个 API 是为了简化 task 的开发而设计的一个提供基于实际硬件的一个抽象的 API。用 FiTT 的 API 结合 Django 来收集结果，并且提供了一个 web 的接口，用户也可以直接从 Web 上提交测试用例，查看和下载测试的结果。

具体来看，FiTT 核心框架的架构可以分为五层八个模块，如图 2-5 所示。

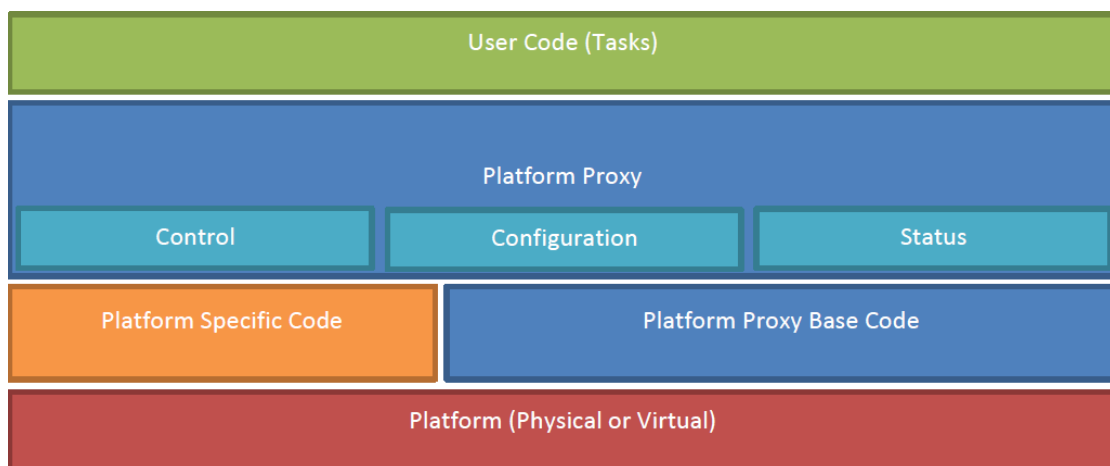


图 2-5 Platform Proxy 的架构

至上而下的模块分别是：

User code 主要是 Task, task 可以被形容成执行一个 Job 的步骤, 也就是一些常用功能和测试用例的实现, 比如 fetch binary, 传递文件来改变预定义的值等。

Platform Proxy 模块: 控制平台的工具抽象化, 同时提供一些通用的可以被 task writer 利用的服务。Platform ProxyAPI 是为了简化 task 的开发而设计的一个提供基于实际硬件的一个抽象的 API。Platform Proxy 的目标在于通过一个统一的系统替代传统的控制类; 为 platform configuration (平台配置) 提供一个通用的 API; 为状态和配置提供永久存储; 允许对服务器主板进行具体的代码和任务的并行开发; 简化了解决方案的集成。

控制组件 (Control Component) 负责抽象与平台交互相关的所有活动, 它旨在通过提供一个简单通用的接口来替代所有的传统的控制模块, 它也能够提供平台定制化的功能。最基本的 Control 组件提供如下几个功能: 电源控制 (AC,DC), 重启, LED 控制。默认情况下, Control 组件会从主板上获得实时的信息。

配置组件 (Configuration Component) 负责追踪测试平台当前的配置信息, 也可以设置新的配置, 这个组件旨在保存一直的配置信息 (要么直接获得配置信息, 要么通过 Task 传入配置值), 而不是为了实施获得信息, 配置组件只能包含可变的配置信息, 换句话说, 只能包含在启动的时候可变的参数。比如说, enabled sockets, 线程数, 分区, 硬盘的模式等等。

状态组件 (Status Component) 用来追踪一些极度不稳定的状态, 比如目前加载的 OS 的名称或者屏幕分辨率的设置。与 Configuration 组件不同之处在于, status 组件是为了尽可能地反应服务器平台的状态, 它需要遵循一些规则。Status 允许和 Task 之间有比较弱的联系, 但是在设计的时候应该尽可能低加强它们之间的联系, 这样若没有设置特定的状态, task 就不能被执行, 测试的可靠性更高。

最底的两层是 Platform 的抽象, FiTT 会关注加载正确的 board 对象, 通过运用保存在 SUT 数据库中的信息并将这些信息通知给 task, 任何 error 的产生都会被记录并报告出来。FiTT 提供了基本的实现加载 board 的步骤以及相关的一些模块。对于无参的 Board, FiTT 也提供了基本的方法和参数。每一个项目都需要扩展这个基

本的 Board 类来添加项目所需的功能。如果项目不需要添加任何功能，对于不希望使用 Platform Proxy 的项目的功能实现而言，那么默认的 Board implementation 就会被用作一个备选。

(2) 提供服务的软件和库

FiTT 主要用到如下软件来提供服务：

通过 RSC2 来控制电源和跳线，要运行 RSC2 首先必须安装 dotNet。此外，由于 RSC2 的库还调用的 VC runtime library，所以需要安装一个 vcredist_x86.exe，如果缺少这个库，python 在 import RSC2 的时候就会报错。

通过 KVM 来获得服务器主板的视频输出，这个视频的界面会被被当做一个窗口投放在 ITP HOST 的界面中，KVM 需要连接到互联网中，用户通过访问 KVM 的 IP 来获得视频的界面。KVM 运行在 JAVA 环境下，所以还需要安装 JRE。

通过 Sikuli 来进行图像比对，以获得一些关键校验点的图像判断。

通过 AutoIt 来实现 python 模拟鼠标键盘，可以很方便地实现对字符串的操作，比如向 EFI Shell 传递字符串或者指令。

Django 的框架帮助 FiTT 提供了一个 web 的接口，通过 Django 可以帮助 Local server 管理 ITP HOST，定制 daily 的测试让用例每天定时执行。

WAIK 帮助完成 windows 无人值守的自动化安装。

FiTT 用到的库主要有如下几个：

PySerial: PySerial 是用来支持串口通讯的 python 模块，它封装了访问串口的类和方法，为 python 提供后端。

Pywin32: Python 并不自带访问 Window API 的库，需要借助 Pywin32，它允许程序员以类似于 VC 的做法来采用 Python 开发 win32 的应用。

PythonITP: 可以通过 ITP 进行 debug。

COMtypes: 由于在 Python 中不能直接访问 COM 对象接口，在开发 FiTT 时需要调用专门访问的 COM 包: comtypes，可以帮助程序员非常容易地实现自定义的访问以及基于 COM 接口和调度。

Readline: readline 是一个提供交互式的文本编辑功能的库，所有使用 Readline 的程序都能使用类似的方法操作命令行，使得命令行编辑变得非常方便。

Twisted12: Twisted 是一个支持多个平台的网络开发框架,效率很高,而且基于 python。在 FiTT 的开发中,我们可以借助 Twisted 实现定制化的网络应用。

Pylzma: PyLZMA 通过 LZMA 库提供了一种独立于平台的方法读写那些经过压缩和解压的数据。

Zope: Zope 是一个开源的采用了现代设计模式的 Web 应用服务器,是一个基于组件架构,允许开发者只使用浏览器就可以创建 web 应用程序。

PIL: PIL 的全称是 Python Imaging Library。它的作用在于为 Python 添加图像处理能力。在 FiTT 中,我使用 PIL 来进行截图为测试人员提供 log 信息。

(4) 测试用例

对于本课题要实现的自动化测试系统,所有的测试用例都是围绕 BIOS 的功能展开的。根据 BIOS 开发的需要,有大量的 feature 的测试,特别是有一些每天都必须跑的测试非常繁重枯燥冗余,需要耗费大量的时间,通过 FiTT 的框架,编写相应的脚本,实现了很多测试的自动化执行,开发人员或者测试人员只需要通过 FiTT 的 WEB GUI 或者 Command line 两种方式提交任务就可以远程控制让 BIOS 自动执行所需测试。

目前根据需求要实现的测试用例主要有如下几个:

CheckNewBIOSDaily 永真循环检测服务器上是否有新的 image 产生。

FetchBinary 拷贝 BIOS image 到 ITP Host 上。

Flash update 做 BIOS 版本的更新,参数需要有 debug 和 release 两种模式。

BootToDOS, BootToEFI, BootToWindows(包括 Win2k8 R2 和 Wind2012)主要做 BIOS setup, DOS,EFI Shell 和 windows 的引导,这个过程检测 BIOS 能否正常启动,能否对 OS 提供正确的 driver 的支持,键鼠设备是否能够正常工作等。

OSinstall 实现 OS 无人值守的自动安装,包括 win2k8 R2 和 win2012。

RSC2 control 实现服务器主板电源的控制,包括 AC ON/OFF, DC ON/OFF, Reset。此外,根据开发人员 debug 的需求还有一些定制化的功能,比如 Recovery。

FiTT 有一个很大的特点就是代码重用率很高,在 FiTT 的编写规定里面,task 和 job 是两个很不同的概念,task 用来描述测试的每一个独立封装起来的步骤,它

是一个单独的.py 文件，而 jobs 是由多个 task 串起来的，是整个算法的实现，是一个.xml 文件，这个文件描述了各个 task 的参数。目前实现的每天全自动执行的 daily smoke test 里面囊括了常用的测试用例。其流程如下：FiTT 永真执行一个脚本去检测服务器上是否有新的 BIOS image 产生，如果没有，则继续等待，如果有，那么这个.py 文件就会启动一个进程，向 Local server 发送执行 daily smoke test 的命令，Local server 收到请求后，会通过 7654 端口向 ITP HOST 发送执行任务的命令，然后 HOST 通过 RSC2 和 KVM 控制 SUT，做 BIOS 更新，如果更新失败，保存 log 信息，终止测试，如更新成功，SUT 继续启动到 DOS/EFI Shell/OS，这个过程检测 BIOS 能否提供正确的 driver 的支持。此外，测试用例还包括 Windows(Win2k8 R2 和 Win2012)无人值守自动安装，以及 BIOS 的回滚测试。

2.3 交互流程设计

在 FiTT 自动化测试框架中，要涉及到四个交互的对象：测试人员，Local Server，ITP Host 以及 SUT，这四个元素的交互构成了 FiTT 的工作流程，本节对各个对象进行交互的方式进行设计，FiTT 的工作流程为：

(1) 测试人员通过 8080 端口访问 Local server 的 IP 来获得 web service，通过 FiTT 的 GUI 界面选择执行的 task，这个 job 就会发送给 Local server；

(2) Local server 和 ITP host 是通过 7654 端口进行通信，Local server 收到任务，就会把这个执行命令发给 ITP host；

(3) ITP host 的 7654 端口侦听到对方发送过来的 job 的包，然后开始执行测试用例，通过 RSC2 控制服务器，也可能是不涉及实际硬件的模拟平台测试；

(4) 执行结束，ITP host 会把结果反馈给 Local server，然后 Localserver 会搜集测试结果，并产生一个结果集的 receipts 文件，然后将这个关于 Fail/Pass/Block 的 xml 文件通过邮件 report 给用户；

(5) 最后用户通过访问 web 来查看结果。

此外，在后期开发中，我们还引入了 Command Line。通过一系列 API 和一个封装起来得 Prompt.py 类，程序员可以通过命令行提交任务。

2.4 本章小结

本章首先对 FiTT 自动化测试框架进行需求分析，在开始介绍了 FiTT 的概念以及它产生的背景，着重对系统的主要功能模块做了详细的分析。第二个部分是系统总体设计，从硬件架构和软件模块两个方面对 FiTT 进行概要设计，着重介绍了软件模块的三个架构，执行框架，调用的服务和库，要实现的测试用例。最后介绍了测试人员与 FiTT 交互的方式，即 FiTT 的执行流程。

3 系统的详细设计与实现

本章主要介绍 FiTT 自动化测试框架详细设计，包括开发环境以及主要功能模块的实现细节，此处主要从两方面来论述，执行框架方面的模块着重介绍平台加载模块的实现和 WEB GUI 界面的实现；测试脚本方面从 DailySmokeTest 的流程介绍了主要的测试用例及其实现方法。

3.1 开发环境

FiTT 采用的语言是 Python，在代码的编写方面处处体现了面向对象的编程思想。FiTT 将一些独立的功能都封装在一个类里面，

本系统的开发环境：

硬件环境：两台服务器，一台用作 Local Server，一台用作 ITP Host；

RSC2+KVM；

一台 SUT:Grantley 的 CRB/WCP/KNP 的服务器主板。

板载硬件：2 颗 Haswell 的 QDD1 的 CPU；

至少两根内存条；

2 个硬盘；

1 个 SATA 接口的 CDROM；

2 个 USB key，一个用来装 DOS 系统。

操作系统：ITP Host: Windows 2008 R2 企业版；

Local Server: Window server 2003 企业版。

开发平台：Python2.6； eclipse3.5 ； jdk1.6 ； Sikuli。

3.2 执行框架模块的详细设计与实现

3.2.1 平台加载模块

FiTT 在对平台加载初始化的处理上使用了 Platform Proxy 的概念。Platform

Proxy 是在 FiTT0.98 中引进的一个新的 API,这个 API 是为了简化 task 的开发而设计的一个提供基于实际硬件的一个抽象的 API。Platform Proxy 的目标是通过一个统一的系统替代传统的控制类;为平台配置提供一个通用的 API;为状态和配置提供永久存储;允许对服务器主板进行具体的代码和任务的并行开发;简化了解决方案的集成;为实际平台和虚拟平台提供一个单一的视角。

Platform Proxy 的工作原理是,将控制平台的工具抽象化,同时提供一些通用的可以被 task writer 利用的服务。在 Platform Proxy 中最基本的抽象是类 Board, Board 这个类封装了所有的由 Platform Proxy 默认实现的函数和 platform 规定的由用户提供的代码。

Board 包括了一些诸如 name, project, type, SKU 的信息,以及如下 4 个组件:

控制组件 (Control component):用来控制 platform (比如 power, feature 等)。替代了传统的平台控制模块,一般封装成 control.py;

配置组件(Configuration component):用来存储执行期间平台的配置(比如 enable processor,分区等等),而且它可以扩展至更改配置参数(比如 BIOS Setup 里面 enable sockets, virtual platform configuration),一般封装成 config.py;

状态组件 (status component):用来追踪平台实时的状态,比如加载 OS,任务的执行状况等,一般封装成 status.py;

测试机信息数据库 (SUTInfo object) 的备份。

FiTT 会关注加载正确的 board 对象,通过运用保存在 SUT 数据库中的信息并将这些信息通知给 task,任何 error 的产生都会被记录并报告出来。

值得注意的是,board 的选择和加载是在 runtime 执行期间,这意味着 Board 的行为可能会被主机系统的配置改变,Platform 设计的目标就是提高实际的执行的准确度,这样会使得实际情形的结果更加的弹性化,而且调试起来简单,但是,它又很依赖于 Board 代码使用 platform Proxy 的方式。在下面的指导中我们使用了一些通用的准则来让 API 做的更好。

(1) 模块的文件组织形式

这个模块全部是用 Python 实现的。本系统的设计采用了面向对象的思想,将各

个模块封装起来。所有与平台加载模块的代码都放在在 `core` 文件夹下面的 `platformproxy` 包里面。FiTT 的编写规定了一个顶层目录 `task` 文件夹需要包含的所有包的形式。在本 `platformproxy` 包里面包含如图 3-1 所示的文件。



图 3-1 平台加载模块的文件结构

Init.py: python 的每个模块的包中，都有一个 `__init__.py` 文件，有了这个文件，我们才能导入这个目录下的 `module`。如果你精通 Python 的话，你一定知道 `__init__.py` 会把这个项目目录变成一个 Python 的包（`package`）——相关 Python 模块的一个集合。

Platformloader.py: 负责平台的加载处理，包括初始化，选择匹配的 `board`，错误处理。

Base package: 是默认的 `platform`，它包括四个文件 `board.py`, `config.py`, `status.py`, `control.py`。前面提到过，在 `Platform Proxy` 中最基本的抽象是类 `Board`, `Board` 这个类封装了所有的由 `Platform Proxy` 默认实现的函数和 `platform` 规定的由用户提供的代码。

Rsc2 package 和 **simics package** 属于内建的 `platform`，`rsc2` 允许控制实际服务器平台的电源和跳线，在 `FiTT` 里面已经将其封装成一个服务。**Simics** 属于一个仿真平台，用来模拟实际硬件的测试。

Feature package:这个类是对功能的扩展，它提供了一些额外的 feature。

(2) 功能的设计与实现

FiTT 提供了基本的实现加载 board 的步骤以及相关的一些模块。对于无参的 Board, FiTT 也提供了基本的方法和参数。每一个项目都需要扩展这个基本的 Board 类来添加项目所需的功能。如果项目不需要添加任何功能,对于不希望使用 Platform Proxy 的项目的功能实现而言,那么默认的 Board implementation 就会被用作一个备选。平台加载功能模块的流程如图 3-2 所示。

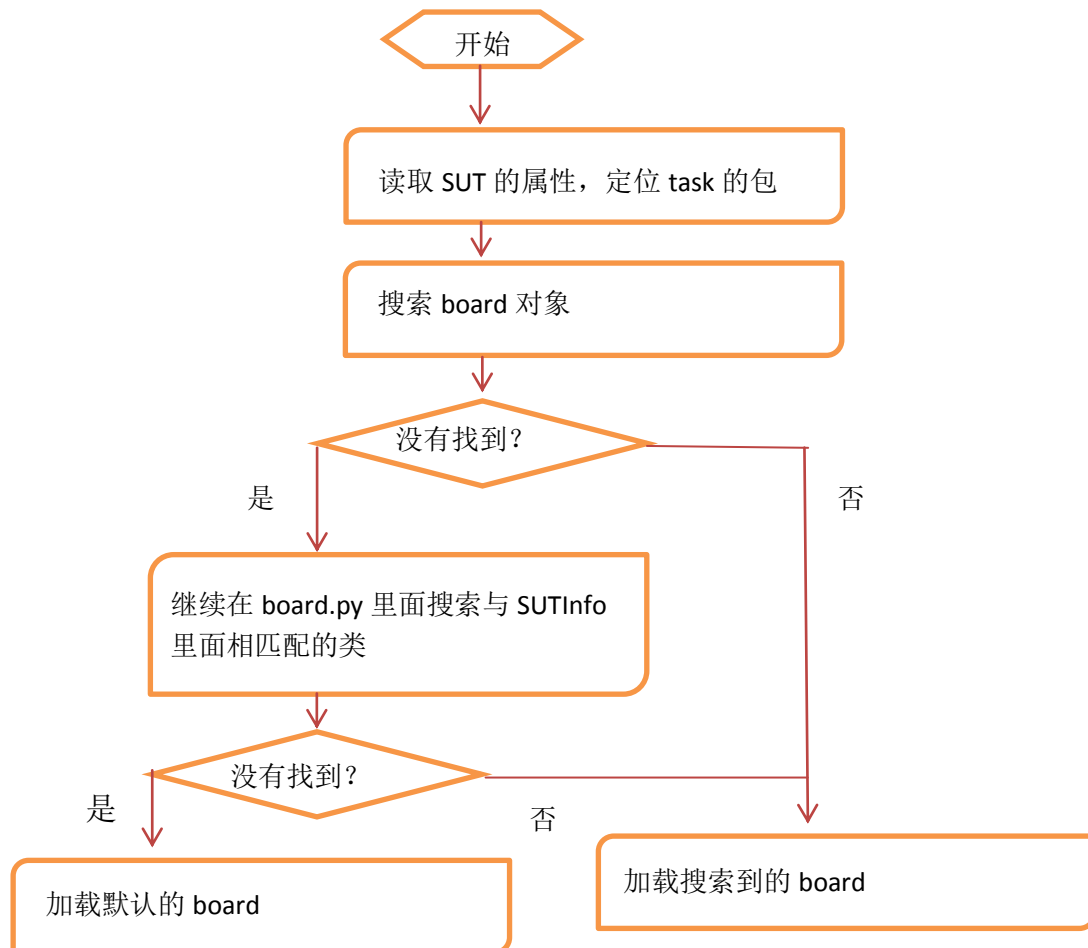


图 3-2 平台加载流程

从图 3-2 可以看出，平台加载的过程如下：

(1) SUT 的项目的一些属性（保存在 SUTInfo 对象中）会被读取，并用来定位一个 FiTT 的 task 所在的 package;

(2) 在挑选出来的 package 中搜索 boards.py, 这个模块可能包含了一个 selectBoard 函数, 这个函数可能会返回一个 Board 对象名, 如果没有, 返回 error;

(3) 如果没有返回 board 对象 (或者没有 selectBoard 函数), 那么程序会在 boards.py 里面搜索一个与 SUTInfo 对象的 board 域相名字符合的类, 这个类应该是一个存在的 board;

(4) 如果没有找到 class, 那么就用 default 的 Board;

⑤如果在有 error 发生, 那么记录堆栈踪迹, 使用 default Board 来实现。

由于 core 实现的代码太多, 尤其很多都是变量的初始化, 在这里我主要列出一些关键的步骤或函数的实现方式。

结合代码, 对此流程进行剖析。

(1) SUT 初始化

```
def loadForSUT(sutInfo):  
    project = sutInfo.project  
    boardname = sutInfo.sutBoard  
    taskpath = GlobalConfigurationManager.taskpath  
    board = None  
    if project not in TaskFactory.packages:  
        raise core.exceptions.platformLoadError(boardname, "Package for %s does not  
exist" % project)
```

```
    configs=GlobalConfigurationManager.getPackageConfigs(packages[project])
```

这一步首先从 SUTInfo 对象中读取 SUT 的属性, 然后检测这个 project 是否存在 FiTT 的 package 里面, 如果不存在, 那么就抛出异常信息“该 package 不存在”, 如果存在, 那么获得配置信息。在 Python 里面使用 raise 抛出异常。

(2) 尝试搜索 project 和 boards

```
projectpath = os.path.join(taskpath, project)  
if not os.path.exists(projectpath):  
    raise core.exceptions.platformLoadError    boardSel = None
```

```
try:
    boardSel = instantiate(project, "boards", "selectBoard")
except core.exceptions.systemError, e:
    log.info("Board selector does not exist")
    log.info(str(e))
```

在挑选出来的 package 中搜索 boards.py, 这个模块可能包含了一个 selectBoard 函数, 这个函数可能会返回一个 Board 对象名, 如果没有, 那么返回 error。在这里用到了 os 模块自带的路径操作函数: os.path.join(path1[, path2[, ...]])将多个路径组合后返回。os.path.exists(path)检查路径是否存在, 如果 path 存在, 返回 True; 如果 path 不存在, 返回 False。SelectBoard 函数会返回一个 board 实例, 其原型是: selectBoard(info,config), 其中 info 指的是 SUTInfo 对象, config 是获得的配置信息。Instantiate()是在 Board.py 里面定义的一个初始化函数, info()函数是一个消息函数, 通过 FiTT 控制台发送提示信息。

(3) 在 board.py 里面搜索与 SUTInfo 里面相匹配的类

```
if boardSel != None:
    try:
        board = boardSel(sutInfo, configs)
    except Exception,e:
        log.error("Error while running board selector")
        log.exception(e)
    if board is not None:
        if isinstance(board, BaseBoard):
            board.configurations = configs
            return board
        else:
            raise core.exceptions.platformLoadError
```

程序会在 boards.py 里面搜索一个与 SUTInfo 对象的 board 域相名字符合的类,

这个类应该是一个存在的 board。如果没有找到 class，那么就用 default 的 Board；如果在这个过程中有 error 发生，那么记录堆栈踪迹，使用 default Board。

这几块代码就是实现加载，初始化，从接收到的 sutInfo 对象里面返回 platform，如果 platform 不能被加载，那么一个 PlatformLoadError 会被捕获。最后程序的结果是返回一个 baseBoard 的实例或者返回捕获到的异常。

3.2.2 WEB Service 模块

FiTT 选择 Django 作为 Web 框架开发的首选，是因为它简洁的开发模式有非常多的优点。首先，Django 可用于快速设计和开发具有 MVC 结构的 Web 服务，因为它拥有完善的对象关系映射机制，使用模板机制，并且具备动态创建后台管理界面的功能；其次，Django 有一个用来做快速开发的内置的服务器，不用安装 Apache、Lighttpd，或者其他的 WEB 服务器软件；再次，Django 的 URL 分发不会在链接中产生一些杂乱的字符，非常直观；最后，Django 可扩展的内置模板可以将模型层、控制层与页面模板完全独立，分别进行编码，有利于开发人员协同合作。

Django 将很多复杂的操作都集成在直观简洁的命令提示符里面，调用 Django 的控制台命令 startproject 就可以新建应用，以 DASHBOARD 模块为例，运行：

```
django-admin.py startproject dashboard
```

命令执行的结果是生成一个 dashboard 的文件夹，里面有如下四个子文件：

__init__.py 文件主要用来向 Python 编译器解释，它表明本目录下的文件是基于 Python 的工程模块。

Manage.py 文件是同本项目协作的一个 Django 的工具，它同 django-admin.py 一起负责对建立的工程进行配置管理，django-admin.py 也是一个命令行工具。

Settings.py 文件是配置文件，用来设置数据库和应用模块的全局配置信息。在项目里安装的所有应用都具有 settings 的访问权限。

Urls.py 文件的作用是配置 URL 的地址映射，管理 URL 的地址格式。Django 里常说的 URLconf 指的就是 urls.py，它是一个配置文件，负责将 URL 模式映射到

应用程序，URLconf 是 Django 最重要的特性之一。

1) 模块的文件组织形式

对于本系统要实现的 WEB GUI 界面，除了上面提到的一个项目所包含的一些配置文件以外，还需要创建如表 3-1 所示应用包。

表 3-1 WEB service 模块的文件组织形式

| 文件 | 功能 |
|-------------------|-----------------------------------|
| Dashboard 包 | FiTT 的首页，显示欢迎信息，版本信息，测试结果统计信息等 |
| Cron 包 | 实现定时执行任务设置的 GUI 界面 |
| PackageManager 包 | 获取 Local server 上已经安装的包的信息 |
| FAQ 包 | 提供常见问题的解答 |
| ResultCollector 包 | 搜集 ITP HOST 上面任务的执行结果，并显示在 GUI 界面 |
| SutManagerDjango | 负责管理分布式系统所有 SUT 的信息 |
| Templates 包 | 负责将各个模块传递进来的信息显示出来 |
| ReportLoaders 包 | 保存 report 信息 |

2) 功能的设计与实现

下面根据 Django 框架实现的流程简要介绍其实现的主要方法：

(1) 创建应用 app

Manage.py 工具是一个简单的封装，可以告诉 Django-admin.py 读入特定项目的设置文件。比如，要新建一个叫做 cron 的应用程序（或 app），可以执行：

```
./manage.py startapp cron
```

这个命令会生成一个应用程序 cron 的框架，它主要包括两个 Python 模块：模型和视图。cron 目录中包含以下文件：__init__.py，models.py，views.py。此时 models.py 和 views.py 只是抢先占据位置，它们并不包含真正的代码。

提供项目中应用程序的位置并不是必需的，它只是为新的 Django 程序员建立的一种约定，如果需要在过个项目中使用同一个应用程序，可以在项目的命名空间中存放该程序，然后绑定配置文件和主 URL 文件。

如果想要让 Django 识别新应用程序，开发人员需要把 app 以模块的形式添到

settings.py 文件的 INSTALLED_APPS 元组里，用逗号结尾。Django 区分系统不同部分的配置是采用使用 INSTALLED_APPS，包括自动化的管理应用、测试框架。

比如，对于新建的 cron app，需要在元组中添加字符串 FiTT.cron。

为了实现 FiTT 所需要的界面，本系统在 settings.py 中添加了如下记录：

```
INSTALLED_APPS = [  
.....  
'djangoCode.FiTT.sutManagerDjango',  
'djangoCode.FiTT.jobRecordsDjango',  
'djangoCode.FiTT.resultCollector',  
'djangoCode.FiTT.faq',  
'djangoCode.FiTT.dashboard',  
'djangoCode.FiTT.cron',  
'djangoCode.FiTT.reportLoader'  
]
```

（2）设计 Model

Models.py 文件的作用是定义应用（app）的数据结构。Django 会根据 DRY 原则尽可能使用程序所提供给应用程序的关于模型的信息。

以 CRON 模块为例。用 idle.pyw 打开 cron 文件夹下的 models.py，删掉用来占位的文本，编写代码创建 CRON 模块专属的 models。代码如下：

```
class ScheduledJob(models.Model):  
    name = models.CharField(max_length=256)  
    description = models.TextField()  
    parameters = models.TextField()  
    owner = models.CharField(max_length=256)  
    mailingList = models.TextField(default="")  
    repeat = models.IntegerField(default=0)  
    monday = models.BooleanField(default = False)
```


...

```
time = models.TimeField()
```

在这个例子中描述的完整的模型，实现了一个包含 15 个变量的“ScheduledJob”对象，默认情况下，Django 会自动给所有的 model 加上一个唯一的、自增的 id 变量，所以严格来说，“ScheduledJob”对象应该包含 16 个变量。这个新建的 ScheduledJob 类是 `django.db.models.Model` 的一个子类，它是一个数据 model 的标准基类。所有的变量，比如 `owner`，`mailinglist`，都被定义成某个特定的变量类（field class）的实例，和普通的类属性一样。

在 WEB GUI 下面，这个 `scheduledJob` 类的显示型式如图 3-3 所示。

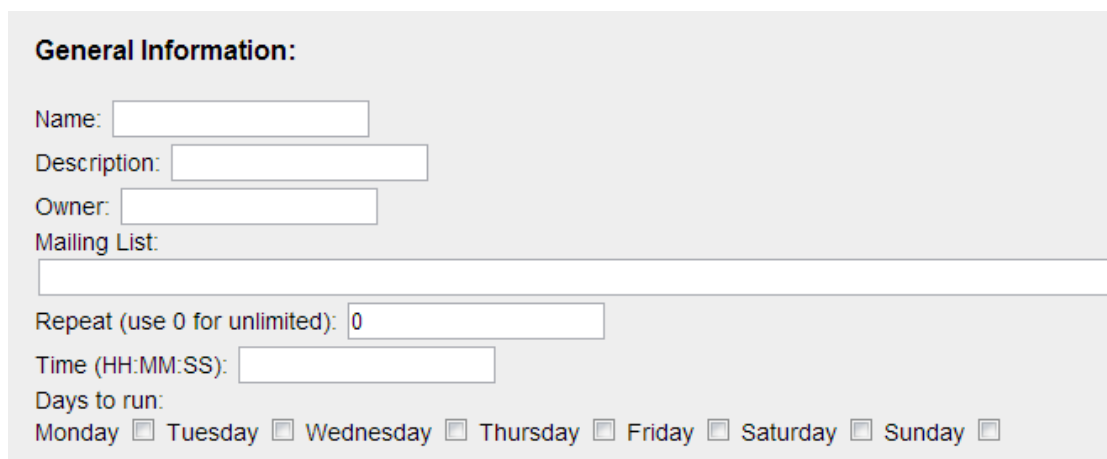


图 3-3 Scheduled job 的 GUI 界面

图 3-3 包含了 Scheduled job 界面的一些设置信息，比如任务的名称，描述，邮件接收人和设定的时间等。

（3）设置数据库

本系统以 SQLite 作为 FITT 平台的后台数据库。SQLite 是一个不需要进行任何配置的轻量级数据库，存储介质使用本地文件系统，访问控制使用原生的文件系统权限，它在磁盘上的存在形式是简单的文件。要使用 SQLite，可以简单地使用 `setuptools` 来安装 `pysqlite`：

```
easy_install pysqlite
```

上面这条指令可以创建一个（空的）数据库，在使用模型之前，需要在设置文

件中配置数据库。通过改写 settings.py 文件来通知 Django 如何使用这个新建的数据库。对于 SQLite，仅仅只用表明使用的数据库名和数据库引擎。

本系统在 settings.py 中配置数据库的代码如下：

```
DATABASES = {  
    "default": {  
        'NAME': os.environ["FITT_DB"],  
        'ENGINE': 'django.db.backends.sqlite3',  
        'USER': '',  
        'PASSWORD': ''  
    }  
}
```

想要用 python 获得一些有关系统的各种信息的时候就不得不想到 os 的 environ，os.environ 返回一个包括当前系统及用户的环境变量的 mapping object，其格式为 environ({变量名: 变量值...})。获得变量值主要有两种方法：使用 os.environ[变量名] 或者使用 os.getenv(变量名)。值得说明的是，import os 就会映射环境变量，如果在这短时间内，环境被改变了（包括系统和用户），os.environ 值并不会发生变化，只能直接修改 os.environ 的值。

设置完数据库之后，可以通过提供的连接信息通知 Django 连接数据库，同时设置应用程序所需的表，执行如下命令：

```
./manage.py syncdb
```

该命令执行时，Django 会查找每一个存在于 INSTALLED_APPS 里面的 models.py 文件，分别为其创建数据库表。INSTALLED_APPS 里的所有默认应用都拥有 model，在运行 syncdb 时 Django 会为每个 app 都创建一个或多个表。

（4）建立页面的公共部分

Django 的页面包含三个典型的模块：功能为显示传递进来的信息的模板（template）；负责从数据库里获取需要显示信息的视图（view）；用来向视图传递参数，或者负责匹配请求和视图函数的 URL 模式。

本模块将通过 DASHBOARD 页面来展示建立页面公共部分的具体实现办法。

① 建立模板

Django 的模板语言是一个加上一些大括号中的特殊模板标签的 HTML，非常简单。这些标签都是变量标签，作用是对模板接收到的数据进行显示。想要访问传递给模板对象的属性，可以在变量标签中使用 python 风格的点记号。如 DASHBOARD 页面的部分模板代码：

```
<div id="dashboardLeft">
<div class="dashboardBox" id="welcomeDiv">
<span class="boxTitle">
Welcome
</span>
Welcome {{ user }},<br />
{% if totalJobs > 0 %}
You currently have {{ queuedJobs }} job(s) queued and {{ runningJobs }} job(s)
running.<br />
The status of your last job is {{ lastStatus }}<br />
{% else %}
You have not run any test.
{% endif %}
```

其在 DASHBOARD 页面左上角的直观展示如图 3-4 所示。

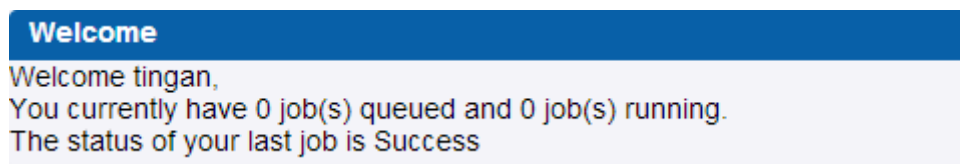


图 3-4 欢迎界面

这是显示欢迎信息的代码。这个模板实现的功能是在首页显示欢迎，并告诉用户目前正在执行的任务是哪一个，以及上一个测试用例执行的结果。

② 建立视图

视图是一个方法，收到请求对象之后，视图负责完成如下模块：直接或间接的所有业务逻辑、拥有模板数据的上下文字典、表示模板、响应对象（返回结果）。

在 Django 中，视图（view）也是 Python 方法，它在一个收到 URL 请求时被调用，模板（template）是视图加载之后显示的页面。视图的作用是从数据库得到需要被展示的消息。本处通过 DASHBOARD 界面的显示版本信息的模块来向大家展示视图函数的编写方式：

```
def showVersion(request):
    systems = getSutsFromDB()
    localVersion = "%s %s" % (core.version, core.devtage)
    try:
        for dirRelease in [dRelease for dRelease in os.listdir(releasepath) if
os.path.isdir(os.path.join(releasepath,dRelease))]:
            print dirRelease
            rd = os.path.join(releasepath, dirRelease)
            rls = os.listdir(rd)
            rls.sort()
            releases.extend([dirRelease+"/"+rl.replace(".zip", "", 1) for rl in rls if ".zip" in rl])
    except WindowsError:
        pass
    t = loader.get_template('update/showVersion.html')
    c = Context({
        'systems': systems,
        'localversion': localVersion,
        'releases': releases
    })
    return HttpResponse(t.render(c))
```

第一行的 `showVersion(request)` 函数，所有的 Django 视图函数都把 `django.http.HttpRequest` 对象设为第一个参数，此外，还可以借用 `URLconf` 接受其他的参数，在 FiTT 的自动化框架 WEB 服务的实现模块里面。

第二行到异常结束，实现从 `getSutsFromDB()` 函数从数据库读取信息并打印。

`t=loader.get_template('update/showVersion.html')`，想要创建模板对象 `t`，只需要告诉 Django 模板的名字即可，为了使 Django 快速找到该模板对象，我们将 `t` 保存在 `app` 的 `templates` 文件夹中。

在 `c=Context()` 里面有三对键和值，Django 模板通过 `Context` 提供渲染的数据，`Context` 是一个字典类的对象。

最后一行，Django 视图函数会返回一个 `django.http.HttpResponse` 对象。给该对象的构造函数传递字符串是最简单的方式。整个函数实现的功能在 DASHBOARD 页面的直观展示如图 3-5。

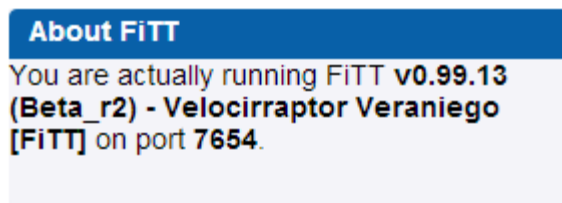


图 3-5 FiTT 介绍信息

这个函数的功能是显示 FiTT 的 core code 的版本信息。

③ 创建 URL

正则表达式是 Django 的 URL 分发系统配置模块的方式，通过将 URL 字符串模式映射成视图 `views`，允许 URL 独立于底层代码，实现控制灵活性最大化。

创建 `urls.py` 模块后，通过 `settings.py` 模块中的 `ROOT_URLCONF` 值，把 URL 配置的默认起点设为 `urls.py`。URL 配置文件需要包含一个对象，用来定义 `urlpatterns` 模式。

以 DASHBOARD 页面为例，应用程序启动时会打开详细视图以及索引，访问方式可以通过如代码所示的 URL 映射来实现：

```
from django.conf.urls.defaults import patterns, include
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('djangoCode.FiTT',
    (r'^$', 'dashboard.views.dashboard'),
    (r'^dashboard/$', 'dashboard.views.dashboard'),
    (r'^update/$', 'dashboard.views.showVersion'),
```

第一行导入 `django.conf.urls.defaults` 下的 `patterns` 和 `include` 模块，第二行和第三行用来 `enable admin`，然后调用 `patterns()` 函数并将结果赋给 `urlpatterns` 变量，视图函数的通用前缀是通过 `patterns()` 函数中的第一个字符串来表示。`urlpatterns` 变量用来定义 URL 以及负责 `handle` 这些 URL 代码的映射关系。在 `FiTT` 里面得到各个 `app` 的 URL 模式，然后放入应用程序中，可以降低项目与应用之间的耦合性，提高代码重用。

3.3 测试用例模块的详细设计与实现

`FiTT` 自动化测试平台目前实现所有的测试用例都是围绕 BIOS 的功能展开的。根据 BIOS 开发的需要，有大量的 `feature` 的测试，比如 `Power on`，对 `OS driver` 的支持，键鼠的支持，`EFI Shell` 下测试的执行，BIOS 的更新等等。

`FiTT` 有一个很大的特点就是代码重用率很高，在 `FiTT` 的编写规定里面，`task` 和 `job` 是两个很不同的概念，`task` 用来描述测试的每一个独立封装起来的步骤，它是一个单独的 `.py` 文件，而 `jobs` 是由多个 `task` 串起来的，是整个算法的实现，是一个 `.xml` 文件，这个文件描述了各个 `task` 的参数，名称，位置，状态等等。目前实现的每天全自动执行的 `daily smoke test` 里面囊括了常用的测试用例。

在 `FiTT` 的文件夹模块中有一个命名为 `Task` 文件夹专门存放测试用例，其中 `Grantley` 包下面的所有测试用例都是服务于 `Grantley` 平台上面的 `UEFI BIOS` 的。上面已经大致列出了实现的测试用例，在文件组织形式方面唯一值得补充的地方是 `FiTT` 的规格说明文档规定了 `task` 文件的组织形式 `FiTT` 有一个专门规定的 `Task` 文件

夹作为所有测试用例任务包的最高层目录。而 Grantley 平台的所有测试脚本都放在 Grantley 文件夹下。

按照 task write guide 的要求，本模块的文件组织形式如图 3-6 所示。

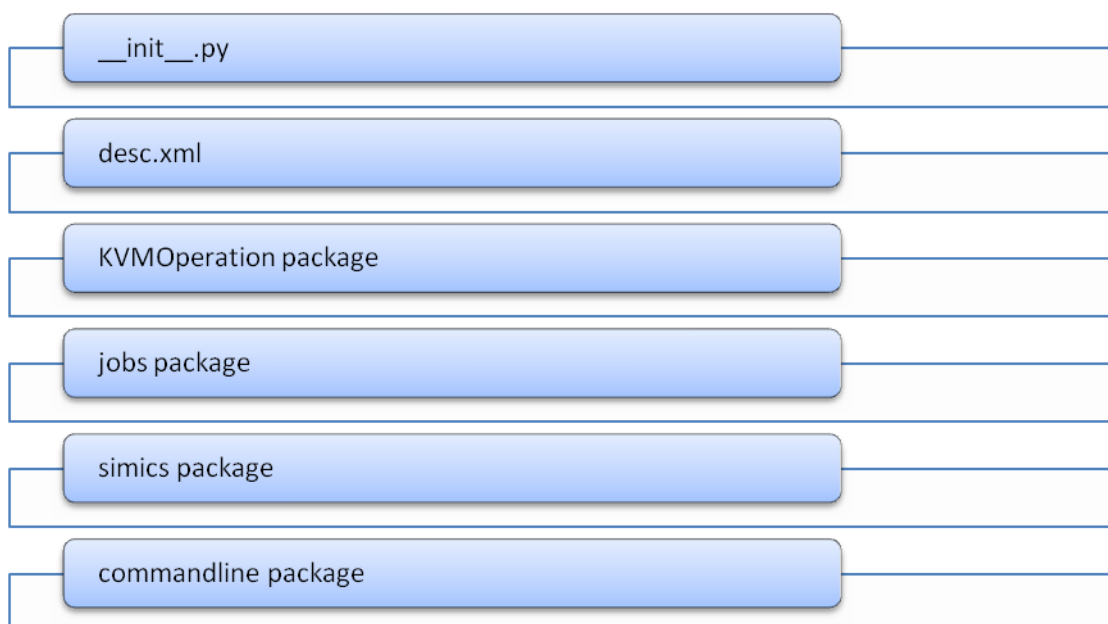


图 3-6 Task/Grantley 文件夹的组织形式

`__init__.py` 文件，前面屡次提到过；

`Desc.xml` 文件，它包含了本目录下的所有 task 的信息，比如名称，参数，位置，变量值等等；

`KVMOperation` 包，都是 Python 代码，即任务实现的脚本，包括 `.py` 和 `.sikuli` 文件。比如启动到 DOS 的测试脚本有三个，`BootToDOS.py`，`BootToDOS.sikuli` 和 `VerifyDOS.sikuli`，`FiTT` 使用 `sikuli` 做图像比对，每个 task 的 `.py` 文件会调用为它服务的 `.sikuli` 文件。

`Job` 文件夹，里面保存所有的 stored job，stored job 是一个 `.xml` 文件，它将一个个小的 task 穿成一个完整的大测试。

`Commandline` 包里面包括命令行执行的一些测试脚本，可以通过命令行和访问 WEB GUI 两种方式提交任务。

此外，可能还包括一些 task 执行所需的支持文件，比如脚本，图片等，这部分是可选的。

目前 FiTT 实现一套全自动执行的测试 daily smoke test，它里面囊括了 FiTT 实现的常用的大量的测试用例，根据 BIOS 的特性，这里面各个用例的执行顺序有着微妙的设计。其流程如图 3-7 所示。

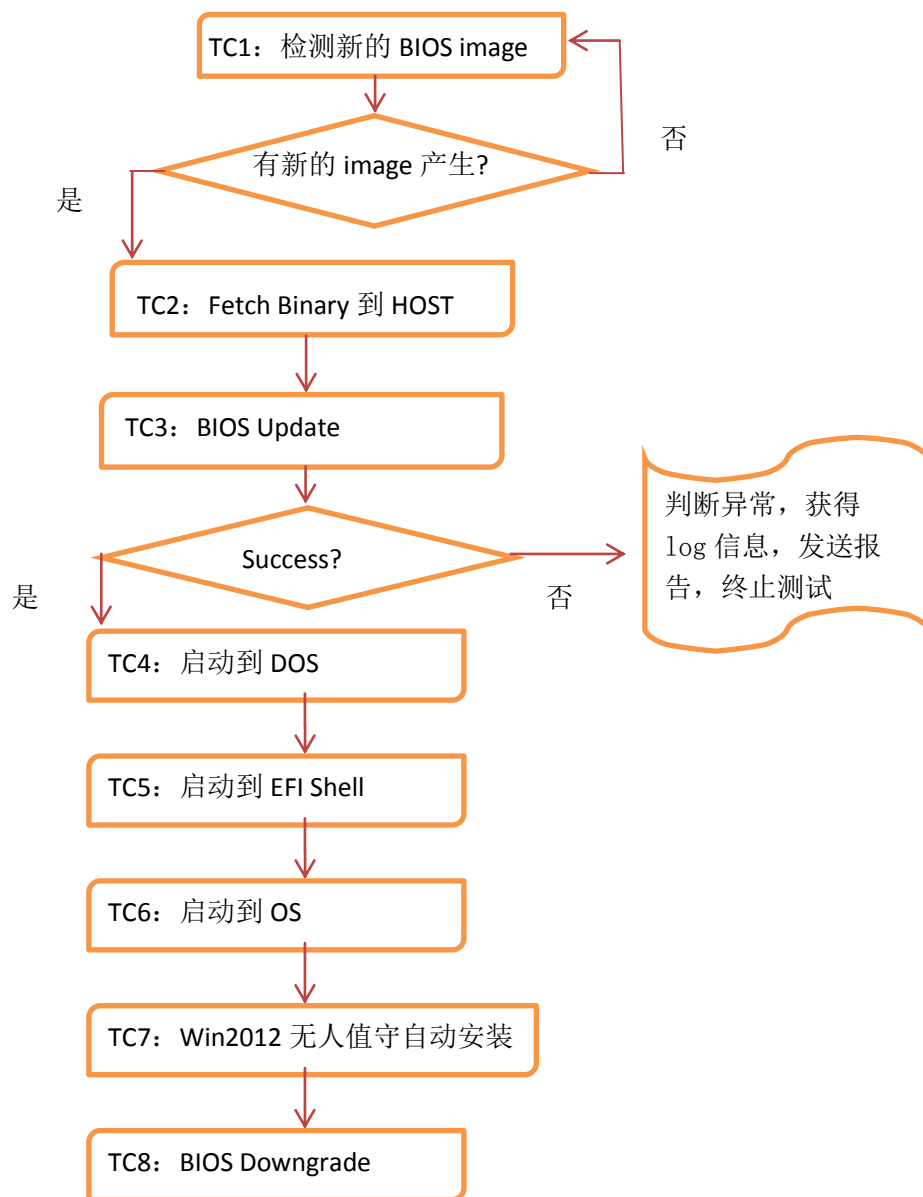


图 3-7 Daily Smoke Test 的执行流程

如流程所示, FiTT 永真执行一个脚本去检测服务器上是否有新的 BIOS image 产生, 如果没有, 则继续等待, 如果有, 那么这个.py 文件就会启动一个进程, 向 Local server 发送执行 daily smoke test 的命令, Local server 收到请求后, 会通过 7654 端口向 ITP HOST 发送执行任务的命令, 然后 HOST 通过 RSC2 和 KVM 控制 SUT, 做 BIOS 更新, 如果更新失败, 保存 log 信息, 终止测试, 如果更新成功, SUT 继续启动到 DOS/EFI Shell/OS, 这个过程检测 BIOS 能否正常启动, 能否对 OS 提供正确的 driver 的支持, 键鼠设备是否能够正常工作等等。第七个测试用例是做 windows2012 无人值守的自动安装, 第八个测试是做 BIOS 的回滚测试。在这几个测试里面, TC1, TC2, TC3 的结果会影响到后面的测试, TC4 到 TC8 是相互独立的。

由于此模块涉及的代码太多, 下面我主要列出一些关键的函数或功能的实现。

3.3.1 检测新的 BIOS 模块

由于 FiTT 对于测试人员提交的任务采用的是一种先进先出的管理机制, 这意味着, 如果某台 HOST 上面一个程序没有执行完, 那么后面提交的关于该 ITP HOST 的所有任务只能处于等待状态。但是, BIOS develop team 每天 build 新的 BIOS image 的时间并不是固定的, 这样如果 WEB GUI 界面的 GRON 目录下设置一个固定的时间去检测有没有新的 image 产生, 这样可能会漏掉一些之前 release 出来的 BIOS 版本, 而且如果一直没有 image 产生, 这个脚本会一直等待, 这样占用 FiTT 的线程和资源。此处我们采用 command line 的方式来永真执行一个检测是否有新的 BIOS image 的脚本。

本系统将检测是否有新的 BIOS image 产生的功能全部封装在一个.py 文件里面来实现: CheckNewBIOSDaily.py, 它永真执行, 循环检测服务器上是否有新的 image 产生。

其核心算法实现的代码如下:

```
StartDirList = os.listdir(MonitorDir)

while 1:
```

```
try:
    CurDirList = os.listdir(MonitorDir)
    for name in CurDirList:
        if name in StartDirList:
            pass
        else:
            for SutName in SutList:
                JobName='DailySmokeTest_'+BiosName+'_'+SutName
                cmd = 'python prompt.py --runjob --destination='+LocalServer+' --port=7654
--identifier='+TaskName+' --parameters='+Parameter1Name+':'+BiosName+'
--name='+JobName+' --owner=swang112 --sut='+SutName+' --mailinglist='+MailList
                PuttyProcess = subprocess.Popen(cmd)
                flag = True
                break
```

代码剖析：os.listdir("path")函数的作用是获取服务器指定目录中的文件及子目录的列表，程序执行开始，会获得服务器中现有的 BIOS image 的文件夹，然后等待一段时间会再次去获取服务器中的文件内容，如果没有新的文件产生，那么等待，一分钟之后再次检索，如果发现有新的文件夹产生，那么程序就会通过 subprocess.Popen(cmd)来调用一个系统进程，通知 Local server 给所有的目的 ITP host 发送一个执行 Daily smoke teste 的命令，然后跳出循环，重新刷新目录，再次循环等待新的 BIOS image 产生。

3.3.2 Fetch Binary 模块

FetchBinary 模块实现了将 BIOS image 拷贝到 ITP Host 上，它是一个独立的模块，可以通过 Command line 方式和 WEB GUI 两种方式加参数（BIOS 的版本号）的提交任务。如果是通过 CheckNewBIOSDaily.py 自动提交的进程，那么这个模块回去拷贝最新产生的一版 BIOS，如果是通过手动的方式提交任务，那么 FetchBinary 模块回去拷贝指定的 BIOS 版本。

本模块只包含一个 `FetchBinary.py` 文件，这个部分代码看似简单，但是实际上涉及到了许多 OS 模块自带的路径处理函数：

```
def copydir(self, srcdir, dstdir):  
  
    dirname = os.path.basename(srcdir)  
  
    if os.path.exists(os.path.join(dstdir, dirname)):  
  
        shutil.rmtree(os.path.join(dstdir, dirname))  
  
        shutil.copytree(srcdir, os.path.join(dstdir,  
dirname), ignore=shutil.ignore_patterns("*.bin", "*.zip", "*.rar", "*.7z"))
```

代码剖析：此处实现了一个拷贝函数。功能虽然简单，但调用了很多函数。`os.path.basename(path)`会返回 `path` 最后的文件名，即去掉路径，只要 BIOS image 的文件名；`os.path.exists(path)`查看路径是否存在，如果 `path` 存在，返回 `True`；如果 `path` 不存在，返回 `False`；`os.path.join(path1[, path2[, ...]])`将多个路径组合后返回，第一个绝对路径之前的参数将被忽略。此外，非常重要的是用到了 python 的 `shutil` 模块，`shutil -- High-level file operations` 是一种高层次的文件操作工具，类似于高级 API，而且主要强大之处在于其对文件的复制与删除操作更是比较支持好。可使用 `shutil.rmtree()` 函数递归删除目录的内容，即删除一整个目录，同理，`shutil.copytree(src,dst)`函数则是拷贝整个目录，此处还通过 `shutil.ignore_patterns` 设定了筛选条件。

3.3.3 自动更新 BIOS 版本模块

Flash update 做 BIOS 版本的更新，根据参数的设置，需要有 `debug` 和 `release` 两种模式。自动更新 BIOS 模块的脚本包括 `Flashupdate.py` 和 `Flashupdate.sikuli`，它的一整个流程是首先通过 RSC2 usb 接口将 BIOS images 从 ITP Host 映射到 SUT 上面，然后启动到 EFI Shell 里面，用专门的 `iflash` 工具，在 Shell 下面运行一个更新 BIOS 的命令。

`Flashupdate.py` 这个脚本的代码超过 400 行，还不包括相关的 `sikuli` 脚本，此处仅介绍关键的功能函数实现代码。

(1) USB 切换

USB 切换调用了 RSC2 的 library, 通过 `host = rsc2.Host('localhost')` 初始化, 就可以调用 RSC2 的一些库函数了。主要代码如下:

```
try:
    host.getBox(0).setUsbMux(rsc2.MUX_TO_HOST)
    time.sleep(15)
except:
    print "cannot switch rsc2 usb to host"
    self.stopTest()
##此处省略从 ITP 拷贝 BIOS image 到 u 盘的动作
host.getBox(0).setUsbMux(rsc2.MUX_TO_SUT)
```

`host.getBox(0).setUsbMux('para')` 用来操作 U 盘, 如果参数是 `rsc2.MUX_DISCONNECTED`, 那么执行卸载的动作, 参数 `rsc2.MUX_TO_HOST` 将 u 盘连接到 HOST 上面, `rsc2.MUX_TO_SUT` 将 u 盘映射到 SUT 上。一整个流程就是, u 盘先从 HOST 上面拷贝文件, 然后在映射到 SUT 上, 这样, 测试主板上的 BIOS 在启动的时候就可以检测到 u 盘里面的文件了。

(2) 给 SUT 上电

给服务器主板上电的过程包括 AC off (拔掉电源), AC on (插上电源), DC on (相当于按下 power on 键), 此处同样用到了 RSC2 的库函数。

```
host.getBox(0).getSignal(rsc2.ID_AC_1).setSigAssertionState(rsc2.AC_OFF)
host.getBox(0).getSignal(rsc2.ID_AC_1).setSigAssertionState(rsc2.AC_ON)
host.getBox(0).getSignal(rsc2.ID_FPBT_PWR).setSigAssertionState(rsc2.BUTTON_PRESSED)
host.getBox(0).getSignal(rsc2.ID_FPBT_PWR).setSigAssertionState(rsc2.BUTTON_RELEASED)
```

`getSignal('button')` 可以获得 power button 的信号, 包括 AC 和 DC; `setSigAssertionState()` 用来设置上电的过程。

(3) 启动到 EFI shell 下面执行刷新的动作

启动到 EFI Shell 动作下一小节会详细介绍, 这里不再赘述, 本节着重介绍更新的操作。我们需要实现的是模仿键盘在 EFI Shell 里面敲出一行命令并执行:

```
Iflash32.efi -u -ni -nopc BIOSImage.cap
```

前面已经介绍过, 为了实现 python 模拟鼠标键盘, 我们借用了 autoItX, 可以通过 autoItX.send()函数来实现:

```
AutoItX.WinActivate("Lantronix SLSP KVM")
```

```
AutoItX.send("map -r{ENTER}")
```

```
AutoItX.send(UdriveLetterInEFI + "{ENTER}")
```

```
AutoItX.send("cd BIOSIMG{ENTER}")
```

```
AutoItX.send("Iflash32.efi -u -ni -nopc BIOSImage.cap{ENTER}")
```

这一程序实现的是在 BIOS 进入到 EFI Shell 里面之后, 用 map -r 命令查看映射到 SUT 的设备信息, 然后进入到 RSC2 上的 u 盘即 UdriveLetterInEFI, 然后用 cd 命令打开 BIOSIMG 文件夹, 调用 iflash32 工具执行更新的命令

但是由于在 EFI Shell 这一步, KVM 偶尔会有屏幕刷新延迟的问题, autoItX.send()发给 EFI Shell 的命令可能会被漏掉, 这样会造成 FiTT 误判。

值得一提的是, EFI shell 有自己的特性, 那便是, 如果启动到 EFI Shell 在五秒内不按任何键, 那么 Shell 会自动去检测 u 盘里面有没有 startup.nsh 文件, 如果有, 那么自动执行该文件。为了解决 autoItX.send()漏发的问题, 我想了一个变通的方法, 便是借用 BIOS 自己的特性。我将 iflash 的动作通过一个 startup.nsh 来实现, 这是一个类似于批处理的文件, 代码非常简单:

```
Fs0
```

```
Cd BIOSIMG
```

```
Iflash32.efi -u -ni -nopc BIOSImage.cap
```

这个 workaround 的方法不仅保证了 BIOS 更新动作的可靠性, 而且顺便检测了 EFI Shell 本身功能的特性。

3.3.4 启动到 DOS/EFI Shell/OS 的模块

这个模块主要是实现 DOS/EFI Shell/OS 的自动引导, BootToDOS, BootToEFI, BootToWindows(includWin2k8 R2 和 Windows 2012)主要做 BIOS setup, DOS,EFI Shell 和 windows 的引导, 这个过程检测 BIOS 能否正常启动, 能否对 OS 提供正确的 driver 的支持, 键鼠设备是否能够正常工作等。

在代码实现方面, 这是三个独立的模块, 由于启动到 DOS/EFI Shell/OS(windows 或者 redhat)有很多关键的动作都是相似的, 这里放在一起介绍, 以启动到 Win2012 为例。

启动到 Windows 的模块的文件包括 BootToWindows.py, BootToWin2012.sikuli 和 VerifyWin2012.sikuli 三个文件, 函数的入口在 BootToWindows.py 里面, 脚本首先会打开供开发人员调试用的 putty 工具以截获 log 信息, 然后调用 RSC2 的库函数实现给服务器主板上电, 然后调用 BootToWin2012.sikuli 脚本引导到 Win2012 系统, 最后通过 VerifyWin2012.sikuli 中预定义的设置分析判断是否成功引导到 Windows, 并在 CMD.tool 中输出一些信息来检测鼠标键盘是否正常工作。下面介绍关键的实现步骤:

(1) 打开串口信息

这里用到了一个开发人员常用来 debug 的工具 putty, 通过它来获得 BIOS 启动过程中抛出的信息以供开发人员调试用。具体的代码如下

```
cmd = os.path.join("c:\\FITT\\UserFiles\\", "putty.exe") + " -load " +
"UDT_Console"

if os.path.isfile(os.path.join("c:\\FITT\\UserFiles\\", "putty.exe")) == False:
    print "putty.exe doesn't exist"
    self.receipt.messages.append("Failed [putty.exe doesn't exist]")
    PuttyProcess = subprocess.Popen(cmd)
```

此处先定位 putty.exe 工具所在的位置, 并设置加载的参数, UDT_Console 模式配置了 putty 打开的 port, 以及串口配置信息, 然后调用了系统函数 subprocess.Popen() 来打开进程。

(2) 等待 SUT power on

```
i = SikuliLoader.loadScriptFromPackage(self.configuration.package,
"KVMOperation.WaitForPowerOn", {}, {}, log)
```

```
i.run()
```

```
if i.resultCode == "FAIL":
```

```
self.receipt.messages.append("cannot find power on screen.")
```

```
lib.Capturer.CaptureScreen('Fail', LogDir)
```

```
self.receipt.addResult(Result.logResult("Fail:", "Fail.png"))
```

SikuliLoader.loadScriptFromPackage 是 python 调用 sikuli 脚本的方法, 此处调用 sikuli 已经编好的脚本 WaitForPowerOn.sikuli。其主要代码如下:

```
switchApp("Lantronix SLSP KVM")
```

```
Count = 0
```

```
Target = Pattern("1375182484393.png").similar(0.59)
```

```
while(exists(Target) == None and Count < 300):
```

```
Count = Count + 1
```

```
if (Count >= 3000):
```

```
fail()
```

```
else:
```

```
success()
```

这个脚本会检测上电之后 5 分钟内 kvm 中显示的 BIOS 启动进程是否会出现 BIOS POST SCREEN 的界面, 如果是, 那么表示服务器主板被正常上电, 返回 success, 否则 fail。

(3) 启动到 Win2012

```
for i in range(150):
```

```
time.sleep(0.1)
```

```
win32api.keybd_event(117, 0, 0, 0)
```

```
time.sleep(0.2)
```

```
win32api.keybd_event(117, 0, win32con.KEYEVENTF_KEYUP, 0)

i = SikuliLoader.loadScriptFromPackage(self.configuration.package,
"KVMOperation.BootToWin2012", {}, {}, log)

i.run()
```

此处是前面介绍过的非常重要的 python 通过 win32 api 模拟键盘敲击的动作，前五行是在 150s 内以 0.1 秒/次的平率狂按 F6，来进入 BIOS 的启动选项。SikuliLoader.loadScriptFromPackage()调用 BootToWin2012.sikuli，这个脚本实现了查找系统盘启动的标签，然后按下 ENTER 键。

(4) 键鼠模拟

最后调用 VerifyWin2012.sikuli 检测是否会出现 Windows2012 的 Shell command，如果找到该工具，用鼠标单击，并输出 Smoke Test 来检测键盘是否能正常工作。最后采用命令行的方式安全关机。

```
AutoItX.WinActivate("Lantronix SLSP KVM")
AutoItX.send("echo Smoke Test{ENTER}")
AutoItX.send("shutdown -s{ENTER}")
```

这里调用了 AutoItX 的两个函数，WinActivate（）实现聚焦，将键鼠操作实现在 KVM 上，然后调用 send()发送两个命令。

(5) 结果处理

VerifyWin2012.sikuli 执行完之后会返回一个结果，success 或者 fail，FiTT 需要对测试结果做一些处理，判断测试是否正确执行，并应当给出相应的 log 信息：对当前状态屏幕抓取的快照，sikuli 脚本执行的 log，putty 的 log，关掉 putty 的进程，停止测试。

下面给出关键步骤的代码：

```
self.receipt.messages.append("Window hang or Blue screen of Death happen.")
lib.Capturer.CaptureScreen('Fail', LogDir)
self.receipt.addResult(Result.logResult("Fail:", "Fail.png"))
PuttyProcess.terminate()
```



```
LogPath = os.path.join(self.job.path, "tmp")
PuttyLogPath = os.path.join("c:\\FiTT\\UserFiles\\", "putty.log")
shutil.copy(PuttyLogPath, LogPath)
self.receipt.addResult(Result.logResult("Putty Log:", "putty.log"))
self.receipt.addResult(Result.logResult("sikulilog", "BootToWindows.log"))
self.receipt.resultCode = self.receipt.FAIL
self.stopTest()
```

此处列举了 failed 的情况，如果 BootToWin2012.sikuli 返回 fail，那么 FiTT 会将其作出的分析发送给 Local server 作为一个提示信息，然后对 KVM 所显示的 BIOS 目前的状态做一个快照，保存图片并发送给 Local server，其中 lib.Capturer.CaptureScreen() 是在执行框架里面加入的一个模块，这个模块借用 PIL 库实现了截图的功能。然后调用系统的 terminate() 关掉 putty 的进程，并将 putty 的 log 和 sikuli 的 log 发送给 Local server，最后通过 self.receipt.resultCode = self.receipt.FAIL 语句将结果设置为 fail，最后执行 self.stopTest() 停止测试。

3.4 本章小结

本章主要介绍 FiTT 自动化测试框架的主要模块功能的实现细节。主要三个模块来论述，执行框架方面的模块着重介绍平台加载模块的实现和 WEB GUI 界面的实现；测试脚本方面从 DailySmokeTest 的流程介绍了主要的测试用例及其实现方法。

4 系统的性能测试与分析

本章介绍 FiTT 自动化测试框架的测试方法和性能分析结果，这个环节非常重要，一方面对根据度量分析的结果进行分析可以保证测试活动的可靠性和有效性；而另一方面，通过对 FiTT 的性能进行分析总结，可以不断改进测试流程中的环节，提高测试的质量和效率。本章首先设定系统性能评价的指标，然后描述测试的过程，最后对其性能进行分析。

4.1 系统性能的评价指标

FiTT 的目标是为英特尔 UEFI BIOS 开发和测试提供稳定可靠的自动化服务，这是一个企业级的应用，因此对系统的稳定性，测试的准确性都有很高的要求。软件能力成熟度模型 CMMI4 体系给出了 4 个衡量标准：覆盖率，执行率，执行通过率，缺陷解决率。本文以其作为自己的性能验收指标：

（1）测试覆盖率

测试覆盖率反映对需求的测试覆盖状况。要从广度和深度两方面看，广度覆盖考察需求规格说明书，是否涉及了每个需求项的测试用例。深度覆盖是说测试设计不能太过浅显，测试用例能否透过已知需求，对可能产生问题的情形进行深入挖掘。FiTT 的测试用例必须尽可能地覆盖 BIOS 开发组所需的功能测试，包括不同的测试平台，不同的项目。自动化测试平台必须能够准确捕捉 BIOS 执行过程中可能遇到的各种情况，是正常通过测试，是测试失败，还是出现异常。

（2）测试的执行率

测试的执行率代表测试用例的实际执行比例。由于在实际的测试开发过程里面，经常发生许多相同的情形。一种情形是企业系统常使用的开发模式是迭代，每次测试的重点都略有不同；二是受测试资源的局限，很难每次都全部执行完设计的测试用例。因而在执行各个阶段的测试时，开发测试人员安排测试活动会根据不同的内容和测试重点，这就是“测试执行率”指标产生的缘由，它是检验一个自动化框架的所有用例使用频率的标准。

(3) 测试执行通过率

测试用例的执行结果主要有：通过，失败，阻塞，忽略。通过与失败反映实际测试结果与预期是否一致，阻塞是表明不同测试用例之间的耦合性，前面的测试是否会对后面的测试产生一定的影响，忽略是由于测试用例根本不适用系统需求。有关自动化测试，这个指标非常重要，测试执行的通过率不仅能够验证测试用例的正确与否，而且能够反映框架本身的健壮性。

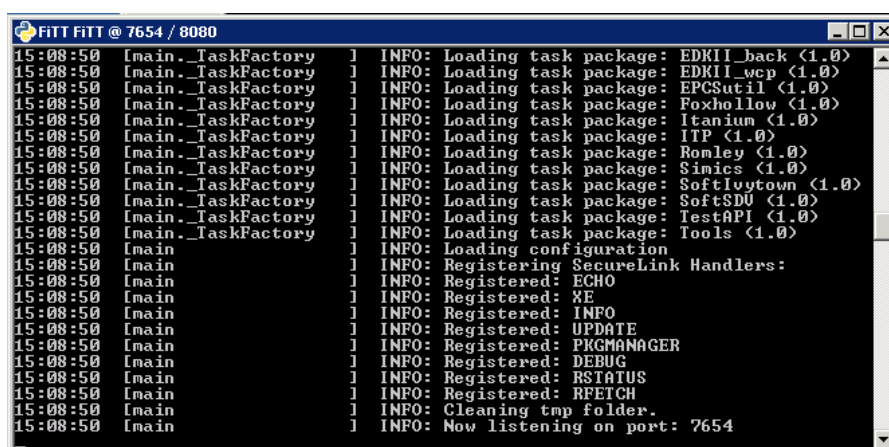
(4) 缺陷解决率

缺陷解决率是指某个时间段内已经解决的缺陷在总缺陷中所占的比例。没有完美的测试，测试框架和测试用例常常本身就可能是存在缺陷的，就算是测试用例在当前的平台环境下都可以通过，但是随着 BIOS 功能的更新和改变，也难保不会出现新状况。通过缺陷解决率，可以检验出本自动化测试框架的稳定性与可靠性。

4.2 功能测试

本节展示测试 FiTT 自动化测试框架在 Grantley 平台上连续执行多个测试用例的过程。

第一步，测试 FiTT 是不是可以成功启动。通过双击 start.bat 文件同时启动 ITP HOST 和 Local server 两端的 FiTT。出现如图 4-1 所示的界面。



```
FiTT FiTT @ 7654 / 8080
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: EDKII_back (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: EDKII_wcp (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: EPCSutil (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: Foxhollow (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: Itanium (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: ITP (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: Romley (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: Simics (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: SoftIvytown (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: SoftSDU (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: TestAPI (1.0)
15:08:50 [main._TaskFactory] 1 INFO: Loading task package: Tools (1.0)
15:08:50 [main] 1 INFO: Loading configuration
15:08:50 [main] 1 INFO: Registering SecureLink Handlers:
15:08:50 [main] 1 INFO: Registered: ECHO
15:08:50 [main] 1 INFO: Registered: XE
15:08:50 [main] 1 INFO: Registered: INFO
15:08:50 [main] 1 INFO: Registered: UPDATE
15:08:50 [main] 1 INFO: Registered: PRGMANAGER
15:08:50 [main] 1 INFO: Registered: DEBUG
15:08:50 [main] 1 INFO: Registered: RSTATUS
15:08:50 [main] 1 INFO: Registered: RFEICH
15:08:50 [main] 1 INFO: Cleaning tmp folder.
15:08:50 [main] 1 INFO: Now listening on port: 7654
```

图 4-1 FiTT Console

在成功启动 FiTT 后，ITP HOST 和 Local server 会不断地监听 7654 端口。等待任务的命令。

第二步，通过 8080 端口访问 local server 的 IP 来获得 FiTT 提供的 WEB 服务。这一步测验 WEB GUI 的服务。WEB GUI 界面如图 4-2 所示。

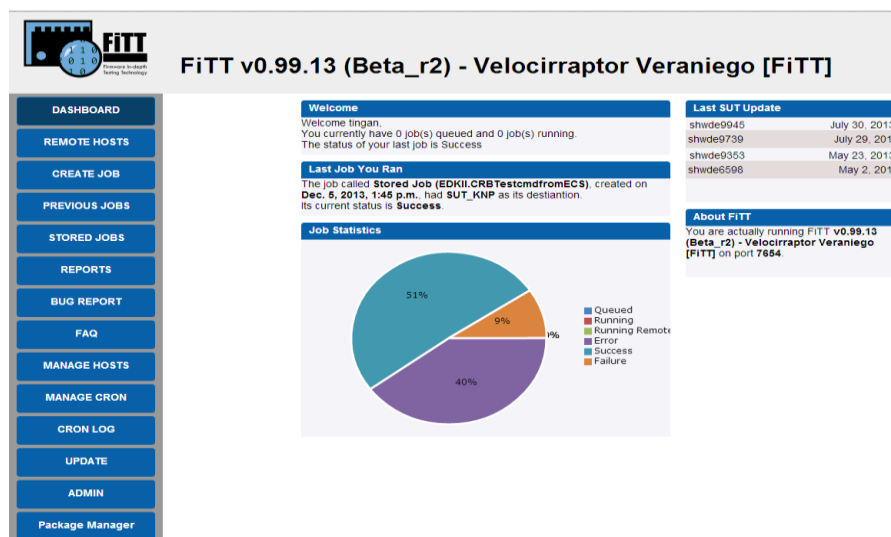


图 4-2 Local server 的 WEB 界面

DASHBOARD 是 FiTT 自动化测试框架的 WEB 界面的首页，它显示欢迎信息，上一个执行的测试用例的信息所有测试用例执行的结果，Local server 分布式控制的所有平台，以及 FiTT 的版本信息。

第三步，单击左边面板的 STORED JOBS 项目，选择要提交的任务，填写参数信息，邮件信息。这一步检测是否能够正常提交测试。出现如图 4-3 所示界面。

Execute Stored Jobs

Stored Job:
Identifier:

Job Data (optional):
Job Name:
Comma separated list of emails:

Description:
Check binary from command line and call smoke test

Parameters:
Version:

Destination SUT:

☐ shwde6598
☒ shwde9353

图 4-3 提交测试用例

选择测试用例，添加邮件接收者，给出参数，选择要执行测试的主板，单击 Execute 后，Local Server 收到指令，解析，然后向指定的 ITP Host 发送命令。FiTT 开始执行测试。

第四步，搜集测试结果。

在测试执行完后，FiTT 搜集测试结果，并发出 report 邮件，如图 4-4 所示。

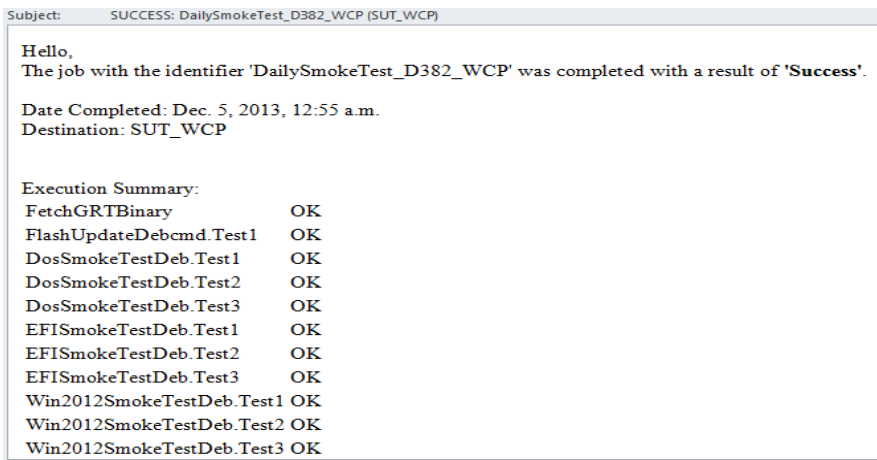


图 4-4 report 邮件的内容

邮件会详细罗列每一个 task 的执行结果。并且会给出详细 log 的链接。

第五步，打开邮件中的链接，会进入到 WEB GUI 界面，可以查看测试结果。

执行的详细信息显示在 WEB GUI 面板的 PREVIOUS JOBS 一栏，如图 4-5 所示。

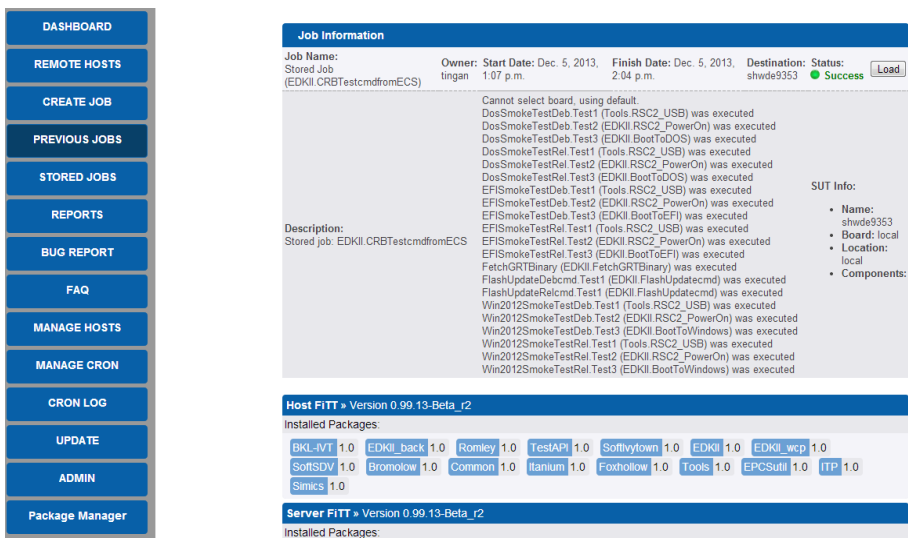


图 4-5 测试结果报告

这里给出了测试执行的详细信息，包括测试名称，开始时间，结束时间，执行的是哪一台 ITP Host，以及测试的详细描述。如果将滑动条往下拉，还可以看到各个测试用例的执行结果，以及相关的 log。如图 4-6 所示。



图 4-6 log 信息

这是更新 BIOS 版本的测试结果，它给出了详细的执行时间，FiTT 的判断结果，以及每个关键校验点的 log 信息，比如 BIOS 更新之前，更新之后的界面截图，Sikuli 执行的每一个动作的痕迹，以及 debug 信息。可以通过 VIEW File 按钮来下载相应的 log。

4.3 测试结果分析

上面通过向大家展示一个测试用例的执行来了解 FiTT 的工作流程。下面分别对评价该系统的四个性能指标进行分析。

(1) 测试的覆盖率

目前 FiTT 的测试用例基本上覆盖了 BIOS 开发组每天都需要做的 daily smoke test。覆盖率超过 85%。CheckNewBIOSDaily 永真循环检测服务器上是否有新的 image 产生，FetchBinary 拷贝 BIOS image 到 ITP Host 上。Flash update 做 BIOS 版

本的更新，根据参数的设置，需要有 debug 和 release 两种模式。BootToDOS，BootToEFI，BootToWindows(includWin2k8 R2 和 Windows 2012)主要做引导，这个过程检测 BIOS 能否正常启动，能否对 OS 提供 driver 的支持，键鼠设备是否正常工作等。OSinstall 实现 OS 无人值守的自动安装，目前实现的是 windows2k8 R2 和 windows2012，RSC2 control 实现服务器主板电源的控制，包括 AC/DC 和重启。此外，根据开发人员的需求还有一些定制化的功能，目前还在不断开发中。

（2）测试的执行率

对于设计的 daily smoke test, FiTT 的执行率是按照 BIOS 开发组 build 新的 BIOS image 的频率来执行的，即，只要有新的 BIOS image 产生，FiTT 立刻将其抓取过来做 BIOS develop team 所需的 smoke test。不会漏过任何版本。对于 BIOS 一些其他的功能的测试，比如 recovery 跳线的测试，是根据测试开发人员的需要，只要 FiTT 是空闲状态，需要测的时候就可以测。

（3）测试执行通过率

FiTT 尽可能体现完全自动化这一概念。在 FiTT 的框架里面加入了一个 job statistics 模块来自动统计迄今为止 FiTT 的所有测试用例执行通过率的状况，打开 FiTT 的 WEB GUI 界面，在 DASHBOARD 页面就可以看到，如图 4-7 所示。

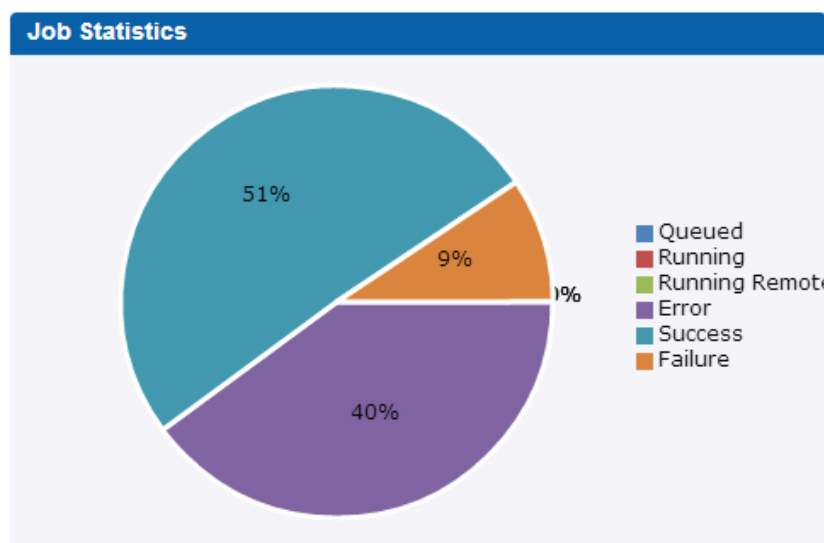


图 4-7 FiTT 的测试结果统计图

到目前为止, FiTT 所有执行过的测试用例通过的比例占 51%, 测试报错的比例是 40%, 这里面的错误包括在 debug 本自动化测试框架时手动终止测试执行造成的错误和 BIOS feature 本身不符合测试用例预期输出的错误, 此外, 9% 的 failure 主要是 BIOS 测试的结果不符合预期, 可能代表 BIOS 存在的 bug。

(4) 缺陷解决率

FiTT 目前已经开发的版本已经到 0.99.13 (Beta_r2), 在每一个阶段我们都会遇到各种各样的 bug。比如漏掉 BIOS 的某个版本不能测, 邮件不能正常发送, kvm 刷新平率不稳定, autoItX 丢包, 无法做 BIOS 版本的 downgrade 等, 这些问题基本上都得到了解决, 比如 autoItX 丢包, 借用 BIOS EFI Shell 的特性, 换了一个 workaround 的手法来处理 BIOS 的更新; 再比如无法做 BIOS 版本的 downgrade, 我们把 BIOS 的版本作为一个变量剥离出来, 发在 desc.xml 里面, 这样就可以传递版本参数了; 对于 BIOS 某个版本不能测的问题, 我们选择绕开 FiTT 的 WEB GUI 里面的 scheduled job 在固定时间触发测试的功能, 而换成后台独立运行的一个脚本。

目前还需要解决的问题是, FiTT 对硬件的依赖还是比较高, 比如 KVM 和 RSC2, 而且 sikuli 对分辨率的依赖过高, 不利于将本项目部署到其他部门。

4.4 本章小结

本章首先提出了性能分析的四个指标, 测试的覆盖率, 测试的执行率, 测试执行通过率, 缺陷解决率。然后通过一个测试用例的执行来介绍 FiTT 的执行流程, 从另一方面检测 FiTT 的稳定性, 最后对测试结果进行总结, 从四个指标的方面来进行分析。

5 总结与展望

5.1 全文总结

本文针对 BIOS 项目开发的需求,创新性地研究出一套针对 UEFI BIOS 开发和测试服务的自动化测试框架 FiTT (Firmware in-depth Testing Technology,也就是固件深入测试技术),FiTT 是由 Python 编写的,并调用了许多经过广泛测试的可扩展的开源的库。FiTT 基于开放的标准和良好的文件记录的接口。它运用 Web2.0 的接口允许任何 web 用户访问。FiTT 的实现将 BIOS 开发人员从沉闷重复的 daily test 中解放出来,执行简单,尽可能最大化资源利用,而且更加稳定可靠。

该系统的主要特征有如下几点:

(1)FiTT 在对平台加载初始化的处理上使用了 Platform Proxy 的概念。Platform Proxy 是为了简化 task 的开发而设计的一个提供基于实际硬件的一个抽象的 API。Platform Proxy 的工作原理是,将控制平台的工具抽象化,同时提供一些通用的可以被 task writer 利用的服务。在 Platform Proxy 中最基本的抽象是类 Board, Board 这个类封装了所有的由 Platform Proxy 默认实现的函数和 platform 规定的由用户提供的 code。FiTT 会关注加载正确的 board 对象,通过运用保存在 SUT 数据库中的信息并将这些信息通知给 task,任何 error 的产生都会被记录并报告出来 Platform Proxy 通过一个统一的系统替代传统的控制类,并且为 platform configuration (平台配置)提供一个通用的 API,它为状态和配置提供永久存储,允许对服务器主板进行具体的代码和任务的并行开发,同时简化了解决方案的集成,为实际和虚拟的服务器平台提供一个单一的视角。

(2) FiTT 实现了许多功能,框架方面包括提供 WEB 服务,邮件服务,并且实现了一整套完全自动化的测试用例,FiTT 永真执行一个脚本去检测服务器上是否有新的 BIOS image 产生,如果没有,则继续等待,如果有,那么这个.py 文件就会启动一个进程,向 Local server 发送执行 daily smoke test 的命令,Local server 收到请求后,会通过 7654 端口向 ITP HOST 发送执行任务的命令,然后

HOST 通过 RSC2 和 KVM 控制 SUT, 做 BIOS 更新, 如果更新失败, 保存 log 信息, 终止测试, 如果更新成功, SUT 继续启动到 DOS/EFI Shell/OS, 这个过程检测 BIOS 能否正常启动, 能否对 OS 提供正确的 driver 的支持, 键鼠设备是否能够正常工作等等。此外, 还包括 windows2012 无人值守的自动安装, 以及 BIOS 的 Downgrade 的测试等。

本文的工作内容包括:

第一章绪论首先介绍了 FiTT 的研究背景, 了解 FiTT 自动化测试的对象, 为需求分析做准备, 然后通过自动化测试框架技术的研究现状, 探讨适合本公司项目的技术方案, 并对课题背景和研究内容作了简要的介绍, 最后就是全文框架介绍;

第二章主要介绍了 FiTT 的开发环境, 详细介绍本自动化测试框架所涉及到的技术, 比如 Sikuli, Django, RSC2, AutoItX, 如何实现 Python 模拟鼠标键盘, 如何结合 WAIK 实现 windows 无人值守自动安装, 本系统所涉及到的分布式的概念, 由于本系统涉及到大量的提供服务的软件和库, 因此对其中的典型, 也有必要做一个简单的介绍;

第三章主要是需求分析和系统设计, 只有明确需求才能完善设计, 在需求分析模块将对 BIOS 的功能做更详细的介绍, 以便实现后面的测试用例。在系统设计模块, 会从软硬件架构两方面详细介绍 FiTT 的核心框架的设计, 主要包括硬件架构, 软件组织结构以及整个框架的运作流程;

第四章是主要功能的详细设计和实现。这里会透过代码来介绍 FiTT 框架所实现的各个功能模块, 包括主要算法, 以及其实现的技巧。本章主要介绍了 FiTT 的平台加载模块, WEB GUI 模块, 测试用例的实现模块, 主要有 FetchBinary, FlashUpdate, BootToDOS/EFI/Win2012, OS 无人值守的自动安装, RSC2 的控制等等, 并且提供 GUI(图形化用户界面)和 Command line(命令行)两种模式供用户使用;

第五章是系统性能测试和分析, 本章给出了四个评价系统性能的指标, 通过一个典型的测试执行的流程, 向大家直观展示 FiTT 实现的功能, 并从四个指标分别对 FiTT 进行结果总结和性能分析。

5.2 展望

FiTT 目前 release 出去版本已经是 FiTT v0.99.13 (Beta_r2)，在每一个阶段我们都会遇到各种各样的 bug。比如漏掉 BIOS 的某个版本不能测，邮件不能正常发送，kvm 刷新率不稳定，autoItX 丢包，无法做 BIOS 版本的 downgrade 等，这些问题基本上都得到了解决，比如 autoItX 丢包，借用 BIOS EFI Shell 的特性，换了一个 workaround 的手法来处理 BIOS 的更新；再比如无法做 BIOS 版本的 downgrade，我们把 BIOS 的版本作为一个变量剥离出来，发在 desc.xml 里面，这样就可以传递版本参数了；对于 BIOS 某个版本不能测的问题，我们选择绕开 FiTT 的 WEB GUI 里面的 scheduled job 在固定时间触发测试的功能，而换成后台独立运行的一个脚本。

目前还需要解决的问题是，FiTT 对硬件的依赖还是比较高，比如 KVM 和 RSC2，而且 sikuli 对分辨率的依赖过高，不利于将本项目部署到其他部门。因此对未来我们希望作出的改进是：

- (1) 探索 console redirection，尽量减少 FiTT 测试结果对硬件的依赖；
- (2) 还可以对功能测试进行进一步开发，比如实现 BIOS 参数的检测；

(3) FiTT 作为一个分布式的系统，目前已经在 WCP, KNP, CRB 三套平台上实现了，但是由于硬件的差异和 BIOS 的差异，这几台 ITP host 上的 FiTT 脚本有一些小的差异，特别是图像处理和时间控制方面，我们还可以做更高的抽象，然后封装起来，将这套系统部署到其他部门。

致 谢

在英特尔亚太研发中心实习的一年里，我在技术能力和沟通能力方面都取得了很大的进步，在实习将近一年的时间里，我能够全身心地专注在 FiTT 自动化测试框架的项目上，这与学校老师同学和企业同时的帮助是分不开的，在本文即将结束之际，我想要对一些给我特别帮助的人表示感谢。

首先，要感谢我的导师黄立群老师，我为自己能有这样一位学识渊博而又和蔼可亲的导师感到非常幸运，黄老师是我在如何参与一个大型项目的启蒙老师，在实验室参与的第一个工业实时监控项目，黄老师耐心教导了我许多专业方面的知识，在嵌入式和计算机网络方面，我学到了很多有用的知识，而且更重要的是锻炼了我如何组织以整个项目的流程，如何进行分工合作等。在工程实践期间，黄老师也常常督促我们学习，在工作和学习方面给了我很多指导和帮助。

其次，要感谢英特尔 BIOS 开发组的所有同事，他们在技术上给了我许多专业的指导。感谢我的经理 Lucia，她在工作上给了我很多支持和鼓励，她在项目上给了很大空间让我发挥，并尽可能的提供硬件和技术上的支持；感谢我的 Buddy Cloud，他是我在英特尔的导师，为我制定了非常好的 training plan，并且在工作中教会了我很多 BIOS 相关的知识；感谢 FiTT 项目中的同事刘铮，Sanyo 和 Carlos，在整个系统的实现过程中，我遇到了非常多的问题，他们总能给我一些非常有建设性的指导。

最后，我要感谢学校的老师同学们，各位老师教给我的专业知识在工作中非常有用，无论是语言，面向对象，还是测试方面的知识，都让我在今后的工作中获益匪浅。我要感谢熊导和廖老师，由于远在上海实习，学校的工作比较难以顾及，她们在生活上给了我很大指导和帮助；最重要的是我要感谢我的室友张艺轩，在实习期间，她帮我处理了很多学校的事情，在我找工作期间给了我很多的鼓励。

参考文献

- [1] Vincent Z., Michael R., Robert H. Beyond BIOS. Intel Press, 2006: 3-15
- [2] 余志超, 朱泽民. 新一代 BIOS——EFI、CSSBIOS 技术研究. 科技信息(学术研究). 2006(5): 12-13
- [3] 王宇飞. 基于 UEFI 的底层 API 的性能分析及其功能测试的研究与设计: [硕士学位论文]. 吉林: 吉林大学图书馆, 2010
- [4] Framework Open Source Community. Edk Getting Started Guide, 2005: 15-21
- [5] MCCONNELL S. Best Practices: Daily Build and Smoke Test. IEEE Software, 1996: 143-144
- [6] 邓慧琴. Web 考试系统的自动化测试方案. 齐齐哈尔大学学报(自然科学版), 2012(1): 12-14
- [7] Zhi Q. Z. , Scholz B. , Denaro G. Automated software testing and analysis: Techniques, practices and tools: Proceedings of the Annual Hawaii International Conference on System Sciences, 2007: 22-24
- [8] Elfriede Dustin. Effective Software Testing: 50 Specific Ways to Improve Your Testing. Addison-Wesley Educational Publishers Inc, 2002: 34-45
- [9] 杨承川. 基于 GUI 的自动化测试框架的研究与实现: [硕士学位论文]. 上海: 东华大学图书馆, 2008
- [10] Berner S. , Weber R. , Keller R. K. Observations and lessons learned from automated testing: Proceedings-27th International Conference on Software Engineering, ICSE05, 2005: 8-9
- [11] 潘雪婷. 基于 Python 的控件分析模型的实现: [硕士学位论文]. 北京: 中国地质大学(北京)图书馆, 2010
- [12] Bouquet F. , Grandpierre C. , Legeard B. , et al. A test generation solution to automate software testing. Proceedings of 3rd international workshop on

华中科技大学硕士学位论文

- Automation of software test, May, 2008: 46-47
- [13] 金虎. 自动化软件测试技术研究: [硕士学位论文]. 成都: 四川大学图书馆, 2006
- [14] 王世俊. 软件自动化测试框架的研究和实现: [硕士学位论文]. 上海: 华东师范大学图书馆, 2006
- [15] Chernonozhkin S. Automated Test Generation and Static Analysis. Program. Compute. Software, 2001, 2(27): 86-94
- [16] 杨平. 基于 OpenOffice 办公软件的自动化测试框架设计和实现: [硕士学位论文]. 天津: 天津工业大学图书馆, 2009
- [17] Han Mei. The Research and Application of IBM Rational Functional Tester. 上海: 同济大学图书馆, 2008
- [18] 李季. 敏捷测试实践与相关问题研究: [硕士学位论文]. 北京: 北京交通大学图书馆, 2010
- [19] 沈偕荫. 基于迭代开发模型的 GUI 自动化测试框架的构建: [硕士学位论文]. 上海: 复旦大学图书馆, 2010
- [20] 彭璇. 面向 Web 应用的自动化功能测试架构研究与实现: [硕士学位论文]. 广州: 华南理工大学图书馆, 2012
- [21] 孙杨. 软件自动化测试集成系统的研究与实现: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2012
- [22] Marty Alchin. Pro Django. Apress, 2009: 8-13
- [23] 夏长虹, 尹排, 陈文博. 组件对象模型 Web 开发的软件工程方法. 计算机世界, 1999. (11): 204-209
- [24] 张磊. UI 软件自动化测试方法和应用的研究: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2008
- [25] Surhone, Lambert M, Timpledon, Miriam T, Marseken, Susan F. PCI Express. Betascript Publishing, 2010: 34-65
- [26] Chernonozhkin S. Automated Test Generation and Static Analysis. Program.
-

- Compute. Software, 2001, 2(27): 86-94
- [28] 杜春华. 人脸特征点定位及识别的研究: [硕士学位论文]. 上海: 上海交通大学图书馆, 2008
- [28] Paul Viola, Michael J. Jones. Robust Real-Time Face Detection. International Journal of Computer Vision, 2004(2): 12-13
- [29] 刘铮. BIOS 自动化测试框架的设计与实现: [硕士学位论文]. 武汉: 华中科技大学图书馆, 2009
- [30] 阎晓弟, 耶健, 邵晶. KVM-over-IP 技术及其在图书馆中心机房远程监控管理的应用. 现代图书情报技术, 2006(4): 31-33
- [31] 王芳. 针对欢跃平台软件 EFEI 的自动化测试系统的设计与实现: [硕士学位论文]. 武汉: 华中科技大学图书馆, 2007
- [32] Jeff Forcier, Paul Bissex, Wesley Chun. Python Web Development with Django. America: Addison-Wesley Professional , November 3, 2008: 14-32
- [33] 周吉波. 基于 Django Web 框架的车载管理系统的设计与应用: [硕士学位论文]. 杭州: 浙江工业大学图书馆, 2012
- [34] 郎芳. 基于 Django 技术的自动化测试工具设计与实现: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2012
- [35] 王冉阳. 基于 Django 和 Python 的 Web 开发. 电脑编程技巧与维护, 2009(2): 11-12
- [36] Rainsberger J. , Stirling S. JUnit Recipes-Practical Methods for Programmer Testing. Manning Publications, 2006: 209-245
- [37] 刘班. 基于 Django 快速开发 Web 应用. 电脑知识与技术, 2009(7): 25-26
- [38] Li Jun, Li Ling. Comparative research on Python speed optimization strategies. Intelligent Computing and Integrated Systems (ICISS) 2010 International Conference, 2010
- [39] 康计良. Python 语言的可视化编程环境的设计与实现: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2012
-

华中科技大学硕士学位论文

- [40] 李迎辉. Python 开发 Rails 框架——Django 框架介绍. 程序员, 2006(11): 14-15
- [41] Dana M. , Raymond B. , William. Web 2. 0 Programming with Django and TurboGears Wright, 2007: 14-15
- [42] Hourieh A. Learning Website Development with Django, Packt Publishing Limited, 2008: 66-71
- [43] M. L. Liu 著, 顾铁成等译. 分布式计算原理与应用. 北京: 清华大学出版社, 2004: 24-27
- [44] 王智慧. 基于自然语言理解的自动应答技术及应用研究: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2008