摘要

自动化运维系统能够为运维人员提供对大规模服务器群集进行可视化运维操作的平台，通过自动化运维系统运维人员可以更直观清晰的了解和记录主机信息，并进行管理。

本文设计并实现了一个自动化运维系统。系统设计开发使用 VIM 开发工具，Python 开发语言，MySQL 数据库，使用了集中化运维管理工具 SaltStack，基于 web.py 的架构完成，该系统设计实现的功能包括：账户管理，主机管理,已加入 SaltStack 的主机信息采集，Master 监控，本系统同时提供对于远程主机的命令管理。

关键词： Python; SaltStack; 自动化运维

Abstract

The automated operation and maintenance system can provide a platform for the operation and maintenance personnel to visualize the operation and maintenance of large-scale server clusters. Through the automatic operation and maintenance system, the operation and maintenance personnel can understand and record the host information more intuitively and clearly, and manage it.

This paper designs and implements an automated operation and maintenance system. The system design and development uses VIM development tools, Python development language, MySQL database, and uses the centralized operation and maintenance management tool SaltStack, which is completed based on the web.py architecture. The functions designed and implemented by the system include: account management, host management, and has joined SaltStack The host information collection, Master monitoring, the system also provides command management for the remote host.

Key words： Python; Saltstack; Automated operations

目 录

第一章 绪论

1.1 研究意义与目的

随着互联网时代的飞速发展，特别是在发展中飞速融入了人们的生活中。在这其中产生的信息也是越来越大，越来越多，云计算与大数据也就这么随着发展，而在发展中跟上来的挑战就是所需要管理的主机集群也愈发庞大，而作为在这后面提供支持的运维人员所需面临的就是如此多的主机，而这有许多工作都是繁琐，重复的，而所需要处理的如此多且重复，那么就很容易出现错误。该自动化运维系统基于 WEB 开发，用户可以通过在该系统中看到主节点的资源信息，以及将所有的机器进行一个登记，以及对各个主机的信息可以进行查看，并且在 WEB 中通过 SaltStack 管理工具可以通过批量的命令发送来进行对旗下主机的一些重复性操作进行批量处理。因此该选题还是有必要的，因为这样就可以通过一个可视化的 WEB 界面来直观的让使用人员了解所有的主机情况，并且让一些重复繁琐的操作一次就能实现，减少出错的概率，因此这对于运维人员来说是具有实用性的[1]。

通过使用 WEB 来开发该自动化运维平台，该课题主要是有以下的几点目的：

1、在大学的时候所学的一些相关专业课程的基础上，将所学的知识与实践结合使用，来进一步巩固相关专业的知识以及实操的能力，从而达到对学习过的知识了解的更清晰，运用更熟练，这是对于已学习过的知识进行更深的挖掘；

2、随着互联网时代的飞速发展，为此提供支持的主机数量也越来越庞大，完成该选题是为了给使用人员提供一个自动化运维平台，能够减低运维人员的一些重复繁琐的工作量，以及更好的对主机集群进行更好的管理；

3、自动化运维领域目前较为热门，这能够帮助我们掌握 Python 的一部分知识，以及对于 MySQL，SaltStack 运维管理工具等进行学习，在之后的职业选择中有更多的选择，在进行相关的运维工作能有更好的帮助。

通过该选题的设计与实现，可以有效的增强我们对于新知识的学习与使用、对相关资料的查询以及使用，以及找出问题分析并解决、综合实践使用相关知识等能力，这有利于我们的学习能力、专业技能能力以及逻辑能力的进一步加强。

1.2 研究背景和现状

在互联网用户及内容的规模还不如现在的过去，为这些行为提供服务的主机数量并不多，因此运维人员通过逐台的登陆主机进行维护或者进行部署服务等都是游刃有余。而随着互联网的逐渐发展，伴随着的就是信息数据量的逐渐增多以及需要上线的项目也变得越来越多，这时候就需要逐渐的扩大主机群的规模，在面对逐渐扩大的主机集群时，运维人员则通过 shell 脚本以及 Python 来对这些主机进行维护等处理。而到了现在，又伴随这云计算大数据等的到来，以及人工智能等行业的火热，运维人员所需要面临的挑战就大了,因此就有了 Ansible、Chef、Puppet、Fabric、SaltStack 等的自动化运维工具来为人们提高效率。

为了完成该自动化运维平台，通过学习、研究、分析，本自动化运维系统主要功能有用户可以通过在该系统中看到主节点的资源信息，将所有的机器进行一个登记，以及对各个主机的信息可以进行查看，并且我们可以使用 SaltStack 管理工具通过批量的命令发送来对旗下主机的一些重复性操作进行批量处理，这有助于提高人们的工作效率，让一些重复、繁琐的工作减少出现错误。该自动化运维平台使用 Python 语言进行编码与实现，后台则采用 web.py 的框架，存储方面则使用 MySQL 数据库来为数据提供存储以及通过 SaltStack 运维工具来实现功能，通过这些相关工具和技术的结果使用来完成该系统。

1.3 系统内容综述与论文结构

1.3.1 本文主要内容

本系统是一个基于 Python 开发的 WEB 项目，是一个自动化运维的 WEB 应用。前端使用 Mako 进行开发，后台使用 web.py 进行开发,数据库使用 MySQL,并结合了 SaltStack 自动化运维工具。可以通过本系统来进行对主节点的资源信息展示的，以及远程命令、查询子节点的信息、批量部署等等功能。

1.3.2 本文主要结构

本论文的主要结构介绍如下：

第 1 章：绪论，主要是论述自动化运维平台研究的意义与目的、课题的背景与现状以及对本论文的主要内容进行简单的概述

第 2 章：系统分析与设计，主要是对该系统的功能模块来进行需求的分析，然后设计出一套符合本系统的架构模式和结构模式，介绍集中化运维工具 SaltStack 介绍以及开发本系统时使用到的一些技术还有系统所需要的环境以及数据库设计。

第 3 章：自动化运维系统的详细设计与实现，首要是详细介绍本系统的各个功能模块

的设计思路以及核心代码的实现，主要有功能实现、关键技术和关键的业务逻辑等

第 4 章：系统测试，主要写了对于整个系统进行测试

第 5 章： 总结与展望，对于本论文进行归纳总结以及提出展望。

第二章 系统分析与设计

2.1 需求分析

用户模块：

用户登陆、注册、修改信息、退出。

主页模块：

展示主节点当前各类信息。

显示当前时间。

主机登记模块：

登记所需记录的主机详细信息。

修改所需记录的主机详细信息。

删除所需记录的主机详细信息。

SaltStack 模块：

显示当前加入 SaltStack 的子节点信息。

通过 SaltStack 命令执行功能实现远程命令、批量部署等功能。

2.2 系统总体设计

2.2.1 开发结构设计

设计、创建该 WEB 的模式使用了 MVC 模式，该模式的是指 Model View Controller 即模型-视图-控制器。关于该模式作用如下：

1. Model 即模型，它表示的即是核心的对于数据的操作或者对于数据库的操作；
2. Controller 即控制器，它负责的根据收到的请求而调用所需要的 Model 来完成请求；
3. View 即视图，它则与用户进行交互并根据所接受到的数据来呈现到用户面前。

2.2.2 系统架构设计

本自动化运维系统的架构由客户端和服务端组成，客户端面对的是使用该自动化运维平台并进行交互的用户，所有功能的实现则由客户端、服务端、数据库以及SaltStack之间的交互完成。如下图2-1所示，为该自动化运维系统的总体架构：

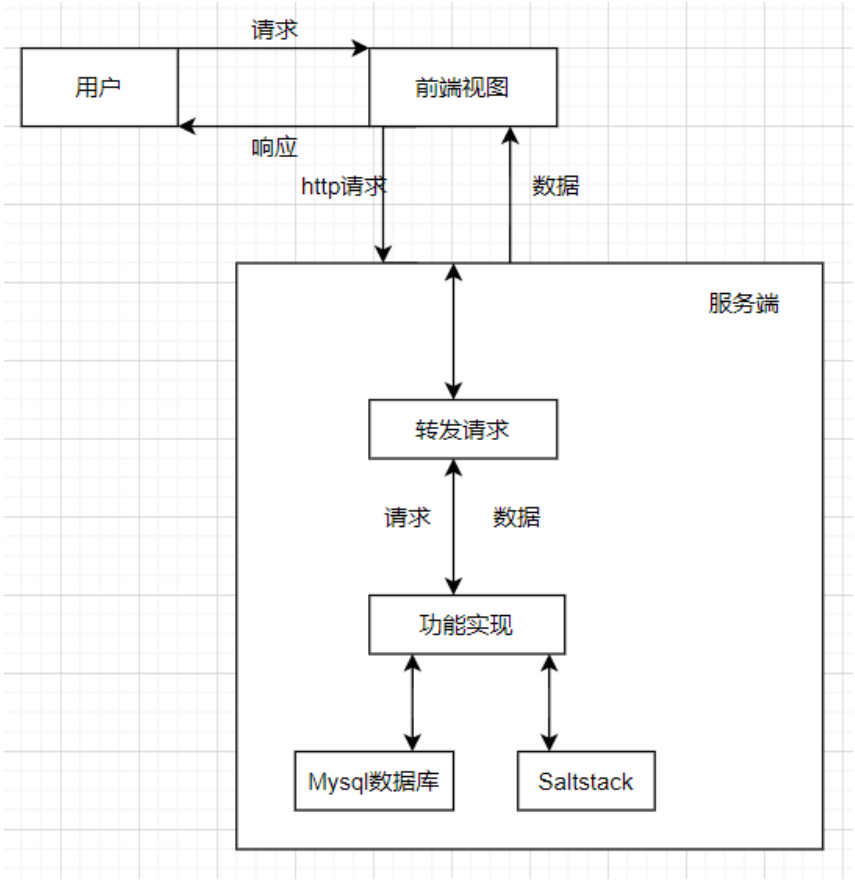

图2-1 系统架构图

2.2.3 总体功能架构

根据 2.1 的功能需求分析，如图2-2所示，此为该自动化运维系统的功能模块：

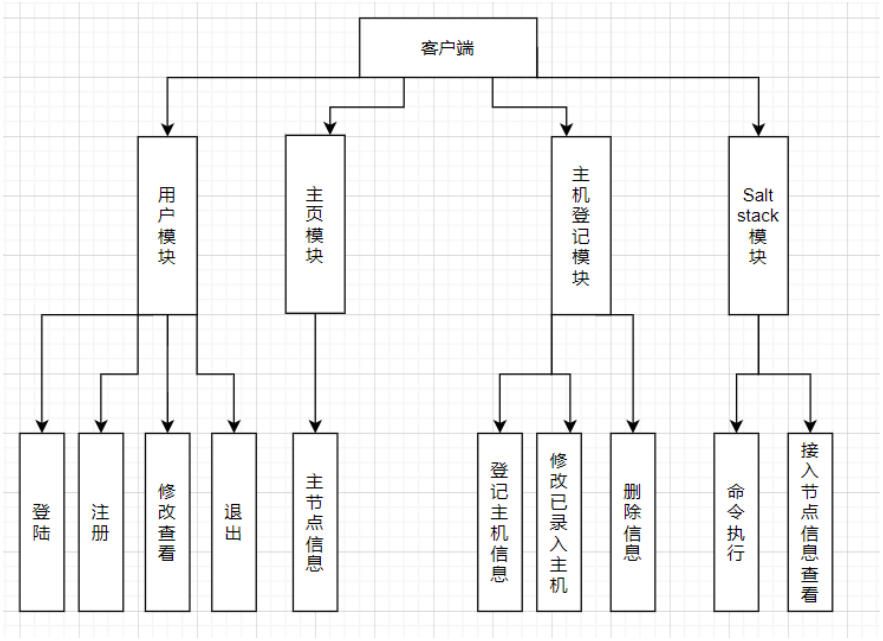
客户端
用户模块
主页模块
主机登记模块
Salt stack 模块
登陆
注册
修改查看
退出
主节点信息
登记主机信息
修改已录入主机
删除信息
命令执行
接入节点信息查看

图 2-2 系统功能模图

2.3 SaltStack 综述

2.3.1 简介

SaltStack 是一个在 2011 年建立的开源项目，这是一个功能强大，能适应与大规模的进行批量管理服务器的 C/S 自动化集中管理工具，该工具是基于 Python 语言实现，实现它的底层网络架构，进行通信的则是轻量级消息队列 ZeroMQ。通过对于 SaltStack 的部署以及使用，我们能够对大量服务器做到任务执行以及配置管理以及基于这两大功能延伸的另外的一些功能如采集服务器数据等等。

2.3.2 SaltStack 基本原理

SaltStack 所使用的模式是 C/S 模式，即客户端/服务端的模式，其中客户端是指 SaltStack 中的 minion 子节点，服务端指 Saltstack 中的 Master 主节点，主节点 Master 与子节点 minion 之间的通信是利用的轻量级消息队列 ZeroMQ。当主节点和子节点的 SaltStack 对应组件下载安装完后，配置完对应配置和 hosts 启动服务后，子节点 minion 会连接主节点 Master 将自己的 pub key 发送过去,请求它为其签发证书，而这时的主节点 Master 则通过 salt-key 命令可以查看并且接受子节点 minion 的 key,为它签发证书，而等到证书签发完毕后，就代表这主节点 Master 和子节点 minion 之间已经互相信任了。这时候我们就可以通过主节点 Master 端发送任何指令让匹配的子节点 minion 执行，子节点 minion 执行完命令后则返回结果。在这之间完成任务是通过 4505 以及 4506 两个端口实现的，其中 4505 端口对于的是轻量级消息队列 ZeroMQ 的 PUBsystem,它的功能是用来发送消息的，而 4506 端口则是 REP system,它的功能则是用来接受消息的。

SaltStack 的主节点与子节点之间的的消息是通过轻量级消息队列 ZeroMQ 的发布-订阅模式来进行传递的，它们之间的连接方式有 tcp 以及 icp 两种。SaltStack 的命令通过 salt.client.LocalClient.cmd_cli 发布到主节点 Master,并以此来获取一个 jobid，而命令执行得到的结果则是根据此 jobid 来获取的。主节点 Master 接收到该命令后，则通过 ZeroMQ 在发送到子节点 minion 上，并通过接受到的消息得到需要执行的命令在通过 minion._handle_aes 处理，其会发起一个本地线程来执行命令。在处理完命令后，调用 minion._return_pub 方法，将执行结果返回给主节点 Master。主节点 Master 接受到了子节点 minion 返回的结果后，在通过使用 Master._handle_aes 方法，将结果写的文件中。在最后获取到结果并最终输出到终端的则是 salt.client.LocalClient.cmd_cli 通过轮询获取 Job 执行结果[2]。

在 SaltStack 中的还包含了 State、Grain、Pillar、Highstate、Modules 等模块。其中，State 模块是通过在服务器 Master 中的/srv/salt 路径下编写.sls 函数关于配置管理等的一些指令，在通过命令调用该函数实现对与目标服务器的配置管理等。Grain 是在 SaltStack 中的以 key

value 形式存储的一种静态的数据库，它存储的是由客户端 minion 要返回给服务端的数据，这部分数据是静态的，包含一些如操作系统类型、版本或者一些硬件属性的不经常改变的数据。Pillar 是在 SaltStack 中的以 key value 形式存储的一种动态的数据库这里面存储的数据是比较私密的，它存储的是客户端 minion 需要向服务器索要的数据，这些数据是只有指定的某个 minion 客户端才能看到的关于自身的 Pillar 数据，而别的是无法看见的。Highstate 的功能与 State 模块功能类似，但是却是能通过 top.sls 来对多个服务器来进行更为仔细的管理。Modules 模块则是包含的 SaltStack 的指令，包括命令行 cmd 模块指令、cp 模块指令、service 模块指令等等，通过这些指令来完成 SaltStack 的功能[3]。

2.3.3 SaltStack 的部署架构

在 SsltStack 中，它包含的角色有三种，其中最重要的两个角色分别是处于中心控制系统的 Master 角色，以及被管理的客户端 minion,还有一个角色是 syndic，它的作用类似与代理，即只负责分发任务下去给客户端 minion 执行。

而基于这三种角色，SaltStack 则有四种部署的架构[4]：

第一种：Master 服务器与所有客户端 minion 直连，minion 通过消息队列直接接受来自于 Master 服务器分发下来的命令，并执行命令或者完成配置管理，具体结构图如图 2-3 所示：

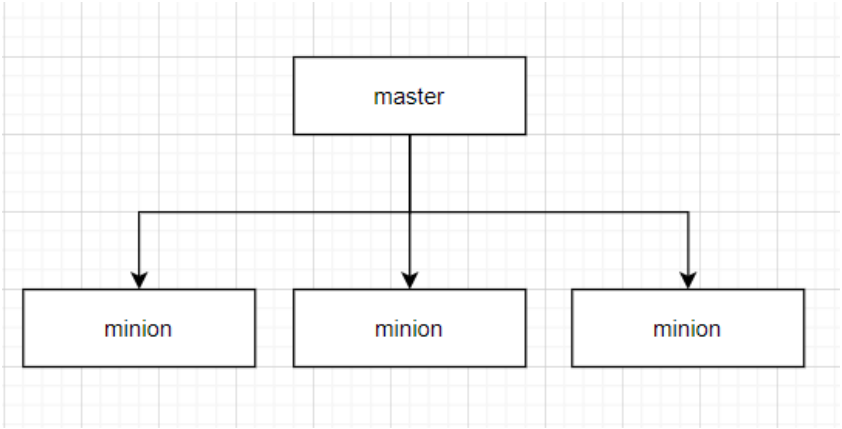

图 2-3 单 Master 直连

第二种：Master 服务器通过 syndic 节点来将命令发布给旗下的客户端 minion，在该架构中，Master 服务器通过 syndic 来对客户端进行管理，通过这种结构，则可以进行更加多级的扩展以此来满足更大规模的使用，具体结构图如图 2-4 所示：

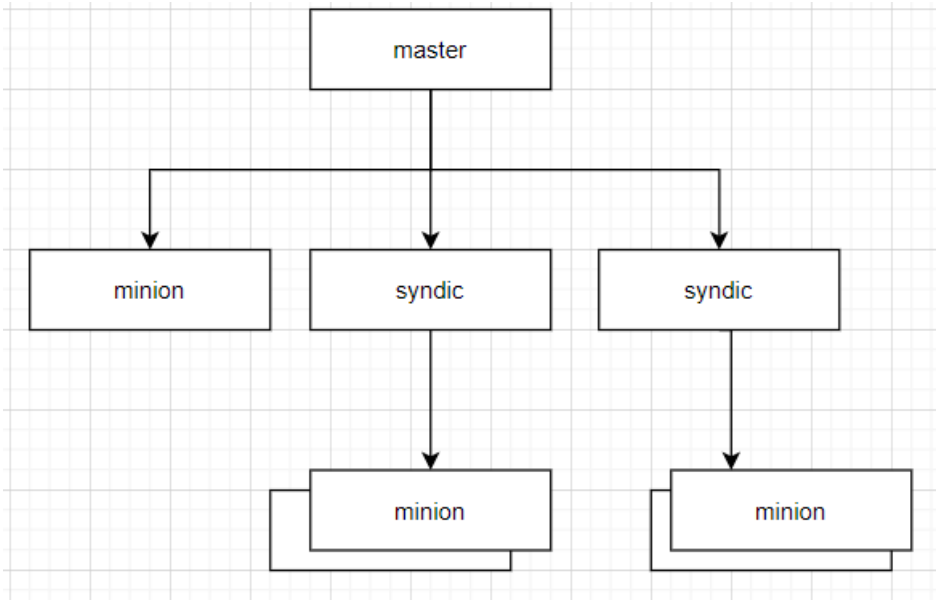

图 2-4 使用代理连接

第三种：多台 Master 服务器与所有客户端 minion 直连，minion 通过消息队列直接接受来自于 Master 服务器分发下来的命令，并执行命令或者完成配置管理，而使用这种结构的时候，客户端 minion 需要在 hosts 以及 salt 的配置文件中配置多个服务端的节点以及多个服务器 Master 之间的配置、状态、密钥文件等需要相同，以此来提高系统的高可用，解决单点问题，具体结构图如图 2-5 所示：

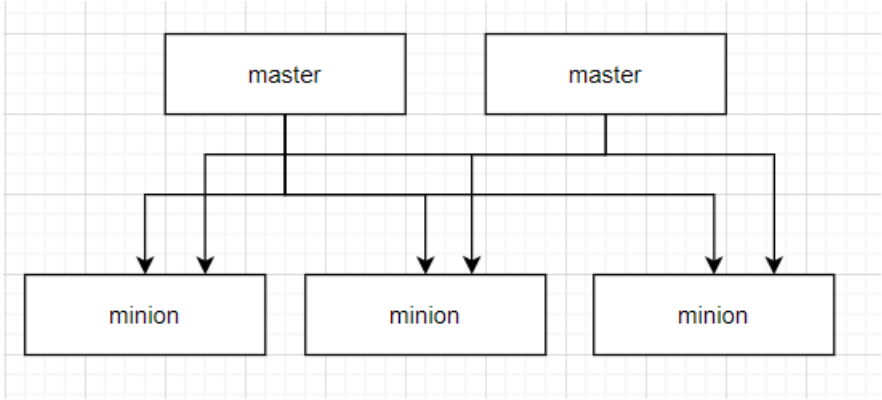

图 2-5 多 Master 连接

第四种：单 minion 客户端的无 Master 服务器模式，即单机版，在该种架构当中，客户端不受任何的服务器 Master 控制使用，只要在本地运行，一些相关的功能就能自己完成，具体结构图如图 2-6 所示：

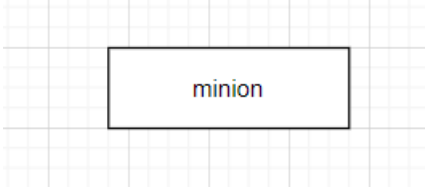

图 2-6 单机版

在本次系统中，我们采用的是第一种 Master 服务器对所有客户端 minion 进行管理的架构，在进行这小型的架构时即可满足需求。

2.3.4 为何选择 SaltStack

目前来说可以选择的自动化运维工具有很多，但是目前而言较为热门的还是Puppet、Chef、Ansible 和 SaltStack、Fabric 这几个工具。其中 Ansible、SaltStack 和 Fabric 这三者都是基于 Python 来进行开发的，而 Puppet、Chef 则是基于 Ruby 来进行开发的。和 Saltstack 相比，Ansible 和其之间功能是非常相似的，不过 SaltStack 是基于轻量级消息队列 ZeroMQ 来进行通信的，而 Ansible 则不一样，它的底层主要还是通过 SSH 来进行通信，而 ZMQ 相比，那 Ansible 它的通信速度则就慢了许多了，而当如果遇到更大规模的集群时，是不如 SaltStack 的，除此之外，它在对于支持的环境这方面而言，也是比 SaltStack 支持的操作系统的数量要更少一些，不过它相对与 SaltStack，它在进行安装部署的时候，子节点是不需要在下载安装 agent 的，相对与 SaltStack 而言则是则是更简单。而 Fabric 相对与 SaltStack 而言，它要更适合与小型的环境中，而且相比较与 SaltStack 而言，由于它的定位以及本身，会更加的容易出现单点故障，因此如果是在稍微大型些的环境中使用的话则显的不妥。Puppet 则是相对于这些而言要更成熟，但是它的库也因此显的更为笨重，除此之外，由于它是基于 Ruby 开发的，而如果要使用它则需要对此语言有更深入的了解才能很好的掌握该工具，相比较而言，它的使用门槛还是有点高的。而 Chef 与 Puppet 有点类似，但也是一样由于是基于 Ruby 开发的，想要使用该工具的话还需要通过次语言进行更多的开发以及了解才能对其进行使用，这相对与 Pupper 而言，更不是一个能简单使用的工具[5]。

因此基于多方面的考虑，我们在这里选择了使用 SaltStack 这一款工具，相比较而言，它具有较为全面的功能，以及拥有快速的速度，对于各类操作系统环境的支持也要更加全面，在大型的集群环境也不会显得不堪重用，而且使用门槛不高，对于使用人员而言更为友好，而且由于是使用 Python 开发的，因此对于该语言是相当兼容的，而 Python 中的数量庞大的库对其的使用更是提供了更为有力的支持[6]。

2.4 开发技术综述

2.4.1 Python 编程语言

Python 是当前较为热门的，能在时下火热的领域中频频出现的如人工智能等的一种面向对象的解释型高级计算机脚本语言。它在创立之初就是为了要编写 shell 脚本而出现的。目前来说，在现在的许多公司对于使用 Python 的工程师还是很有需求的，而别是在与人工智能以及运维方面等。Python 这一门语言具有许多的有点，比如简洁、拥有大量的扩展库，而正是凭借这些扩展库让 Python 拥有了许多的功能以及对于新手而言更为友好，它的使用门槛并不会特别的高，正是这等等的众多优点也让它的在现在的众多的编程语言当中能有属于它自己的地位，当然了，Python 也并不是毫无缺点的，比如相对与 C、Java 等语言来说它的速度就相对而言慢了不少了，而这也是它解释型语言的特性导致的，它在工作的时候是需要将代码解释给硬件能理解的语言，所以就会导致在速度方面会比较慢，以及还有另外的一些小的缺点，当然了，这些缺点对于 Python 而言并不会盖过它的优点[7]。

2.4.2 web.py 框架

web.py 框架是一个小巧轻量但是这不影响它功能强大的一个基于 Python 的 WEB 开发框架。它是一个很精简的框架，只是提供了基础的骨架以及 url 路由，对于用户而言通过使用这个框架对于业务的逻辑方面来说是更容易上手使用进行开发的，不过也是因为它精简的特性所导致它所拥有的功能并没有其他框架那样强悍，如果有所需要的另外一些功能则需要使用者自己在去进行开发。

2.4.3 Mako 模板

Mako 模板是一个基于 Python 的一个 WEB 模板，它是一个借鉴了 Jinjia2 等多个优秀模板而编写出来的一个简单易上手、渲染速度快，并且对于 Python 语言的引用有着优秀的支持的一个模板，虽然它的使用难度并不算很高，但是这并不影响它的对于 Python 中的 WEB 开发的优秀的支持。

2.4.4 psutil 库

psutil 库你一个可以在 Python 使用的且且跨平台的一个第三方库。它在对于对系统资源监控之一方面起到重要的作用，通过使用该库，我们使用 Python 来进行对系统管理的时候效率会的到非常大的提高，而且使用调用该库并实现功能的时候非常简单，只需要些许代码即可实现所要实现的对系统资源监控的功能，除此之外，该库所支持的操作系统数量也是不少，主流的大部分操作系统都能支持进行资源监控，psutil 绝对是一个在运维人员手中的一个重要第三方库。

2.4.5 Dmidecode 工具

Dmidecode 是一个可以获取系统硬件信息的一个工具，我们可以通过下载 Python 使用的版本来对该工具进行使用，通过该工具，我们能够对于系统的的硬件信息进行获取，这是一个对于 psutil 的补充使用。

2.4.6 MySQL 数据库

MySQL 是一个在现在也有非常多个人或企业使用的一个开源的、体量小、轻型的关系型数据库，别看它开源且体量不大，但是这写都不影响它的功能，功能强大，对于使用人员来说相对于其他数据库而言更容易上手，对于许多操作系统都是提供使用支持的，而且具有非常不错的稳定性的同时维护方面也并不困难，这些结合起来对于个人或者是一些中小企业来说，满足他们的数据存储需求也是完全足够的了，而即便是是在面对大型的数据存储的场景时，它也并不是不能使用的，这足以看出它的功能强大[8]。

2.5 系统所需环境及数据库设计

2.5.1 系统运行时所需环境

桌面虚拟软件：VMware Workstation PRO
操作系统：Centos6.5
数据库：MySQL5.1.73
SaltStack:SaltStack 2015.5.10

2.5.2 环境配置

1、Python 的相关配置：

在使用操作系统自带的 Python 的时候，还需要下载 pip 这一 Python 的包管理工具，以此来下载使用一些所需要使用的 Python 的库，在下载后，可一通过 pip -V 命令来查看 pip 的版本，并以此来查看是否安装正确。而在正确下载该管理工具后，在下载其他包后，可以是哦那个 pip -show 命令来查看所下载的包是否正确。

2、MySQL 的相关配置：

MySQL 是我在该系统中采用的用来存放一些数据的关系型数据库，在本次中所使用的是 5.1.73 版本的数据库。在下载安装后，我们在为其创建了 root 用户以及密码为 123456789，除此之外，为了能够不仅仅是在本机连接上数据库，通过 MySQL 的命令，为用户 root 赋予了远程登陆的权限，除此之外，为了防止数据库出现问题，将这里面的空账号删除。

3、SaltStack 的相关配置：

SaltStack 是我在该系统中所使用的一款自动化集中管理工具。在下载该工具前，我们需要修改 yum 源，因为如果使用默认的仓库的话，这里面是不含有该工具的，因此需要安装新的 yum 源，并更改相应的配置。在完成前面的步骤后，由于在本系统中所采用的架构是服务端 Master 直接管理客户端 minion,所以需要分别对应下载相应的 salt-master 和 salt-minion 版本。下载完相对于的版本，则分别在所有服务器中的 hosts 文件中增加相对应的 ip 到主机名的映射，然后在 minion 服务器中修改 SaltStack 的相应配置后，将所有服务器启动，在启动后，客户端 minion 会根据之前的 hosts 文件中的映射以及相应 salt 文件中的配置寻找带 Master 并发送证书请求其为自己进行签名，已达到互信，Master 可以通过 salt-key -a 的命令来接受某客户端的请求，不过如果需要接受很多 minion 的时候，则可以修改相应的 auto_accept 的配置，让 Master 自动为 minion 签发证书。如此一来，SaltStack 也就完成了相应的配置可以进行使用了。

2.5.3 数据库表分析

为了能够满足该自动化运维系统的功能需求，在此于数据库中建立了与用户信息模块、主机信息模块、配置信息模块、登陆信息模块等相关的表来实现。

用户信息模块:该表存放了用户的 id、账号、密码、昵称、手机号码、邮箱账号、用户等级、用户状态、最后登陆记录、注册时间、备注等这类信息。

主机信息模块：该表存放了关于该主机的 id、主机名称、如果存在域名记录域名信息、内网 ip、公网 ip、管理 ip、服务器型号对应 id、CPU 信息对应 id、硬盘大小对应 id、内存大小对应 id、操作系统版本、自定义编码信息、登记时间信息、上架时间信息、更改的时间、登记者 id、更改者 id、任务信息、类型、机房归属信息对应 id、所属机柜信息对应 id、标签、编号、当前状态、备注等信息。

配置信息模块：该表存放了关于该配置的 id、类型、该类型的值、组标记、当前状态、备注等信息。

登陆信息模块：该表存放了关于登陆顺序信息 id、用户 id 登陆时间、登陆 ip、登陆地方、使用浏览器类型、登录令牌、过期时间、会话状态等信息。

2.5.4 概念模型设计

根据上述的需求分许，我们可以凭此得到设计思路，并按照此思路设计出各实体关系，并用 E-R 图表现出来。各 E-R 图如图 2-7 所示：

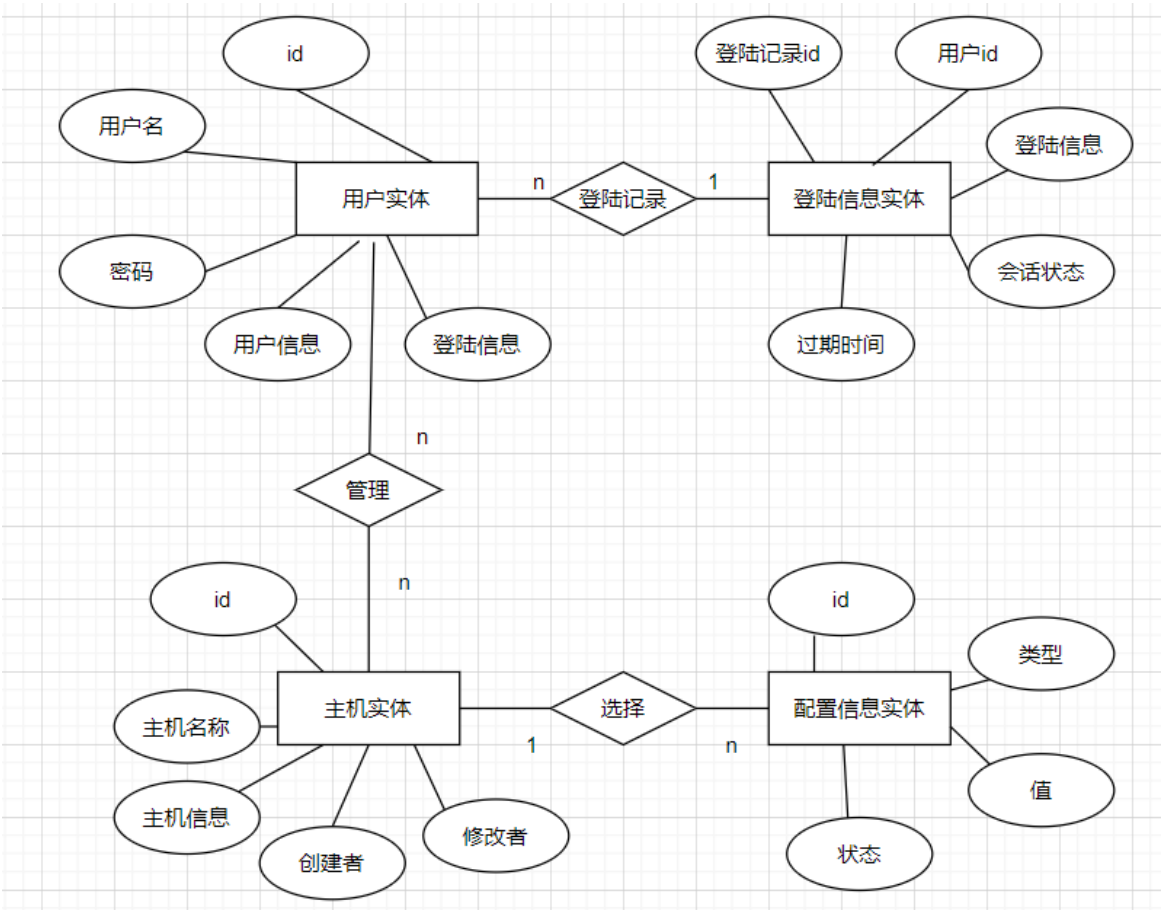

图 2-7 实体关系图

2.5.5 数据库表结构设计

当前的表结构设计如表 2-1 至 2-4 所示：

表 2-1 用户信息 users 表

字段名	类型	长度	允许空值	主键	注释
id	int	3		√	用户的 id
username	varchar	20			账号
password	varchar	60			密码
nickname	varchar	20	是		昵称
mobile	varchar	15	是		手机号码
email	varchar	50	是		邮箱账号
level	int	1			用户等级
status	varchar	3			用户状态
loginfo	int	5			最后登陆记录
regdate	timestamp	0			注册时间
comment	varchar	50	是		备注

表 2-2 主机信息 hosts 表

字段名	类型	长度	允许空值	主键	注释
id	int	3		√	用户的 id
hostname	varchar	30			主机名称
domain	varchar	30	是		如果存在域名记录域名信息
priip1	varchar	15			内网 ip
priip2	varchar	15	是		内网 ip
pubip1	varchar	15	是		公网 ip
pubip2	varchar	15	是		公网 ip
adminip	varchar	15	是		管理 ip
model	varchar	30			服务器型号对应 id
cpu	varchar	30			CPU 信息对应 id
hdd	varchar	20	是		硬盘大小对应 id
mem	varchar	20			内存大小对应 id
os	varchar	30			操作系统版本
rnum	varchar	20			自定义编码信息
storagedate	date	0			登记时间信息

startdate	date	0			上架时间信息
mdate	timestamp	0			更改的时间
creator	int	3			登记者 id
editor	int	3			更改者 id
role	varchar	20			任务信息
type	varchar	20			类型
idc	varchar	20			机房归属信息对应 id
idctag	varchar	20			所属机柜信息对应 id
stag	varchar	20			标签
snum	varchar	20			编号
status	varchar	10			当前状态
comment	varchar	20	是		备注

表 2-3 配置信息 options 表

字段名	类型	长度	允许空值	主键	注释
id	int	3		√	该配置的 id
type	varchar	15			类型
value	varchar	50			该类型的值
default	varchar	3			组标记
status	varchar	3			当前状态
comment	varchar	20	是		备注

表 2-4 登陆信息 login_logs 表

字段名	类型	长度	允许空值	主键	注释
id	int	6		√	登陆顺序信息 id
uid	int	3			用户 id
data	timestamp	0			登陆时间
ip	varchar	15			登陆 ip
location	varchar	30	是		登陆地方
agent	varchar	150			使用浏览器类型
token	varchar	64	是		登录令牌
expiry	datetime	0	是		过期时间

status	varchar	3	会话状态

第三章 自动化运维系统的详细设计与实现

3.1 用户模块的设计与实现

3.1.1 登陆注册流程

如果用户的登陆进网站后则进入登陆界面，在输入相关用户信息进行登陆后，会去数据库进行判断是否正确并将结果返回到前面，如果成功则登陆，失败则无法进入。在进入网站后可以通过用户管理的功能增加用户，并将该数据与数据库中数据进行比较，如果用户信息不存在则注册成功，存在则注册失败。其中登陆注册的流程图如图 3-1 至 3-2 所示。

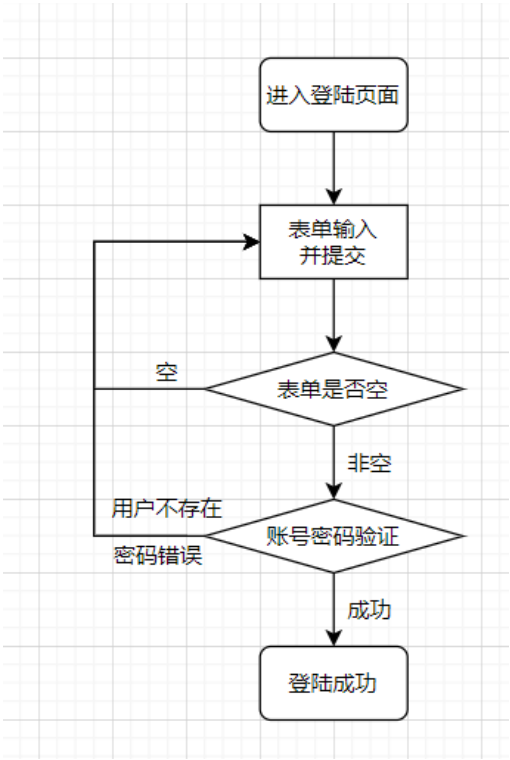

图 3-1 登陆流程设计

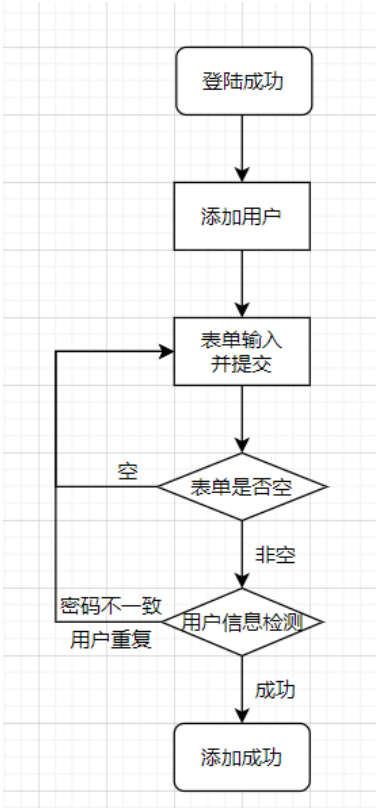

图 3-2 注册流程设计

3.1.2 用户模块的介绍与实现

用户模块中包括了登陆、注册、修改信息、删除用户、查看信息、退出组成。用户添加、删除、修改查看页面如图 3-3 至 3-4 所示。

图 3-3 登陆页面

图 3-4 用户管理页面

在登陆的时候，点击登陆，则通过一个监控事件将用户信息拿去数据库进行查询，然后通过该信息获取该用户的所有信息并保存开启一个 session 会话中在将此次会话状态设置为运行中的时候并根据此进行一个 cookie 的设置然后在进行用户的登陆，最终通过验证跳转进入主页。

而在登陆后，则可以对用户进行注册，通过新增用户，对用户的数据进行输入后，则将该数据拿去数据库中根据用户名来进行判断是否拥有重复的用户，经过判断没有后，则将新用户的数据存入数据库中完成用户注册。进行用户注册后，还可以对用户信息进行修改、查看以及删除以及对使用状态的更改。这一系列操作则是通过修改数据库中的数据属性来进行实现。

在完成了一系列操作后需要退出，则在选择登出后，通过该用户的 id 找到数据库中的数据将数据库中关于该用户的使用状态进行更改，并将该 session 会话关闭，这时候即完成了退出。该部分的主要代码逻辑如附录 1 中所示。

3.2 模块的设计与实现

3.2.1 主页功能流程

在进行登陆后，会将信息传递到主页后，在主页中将这些传递过来的信息在页面中展现出来，其中流程图如图 3-5 所示。

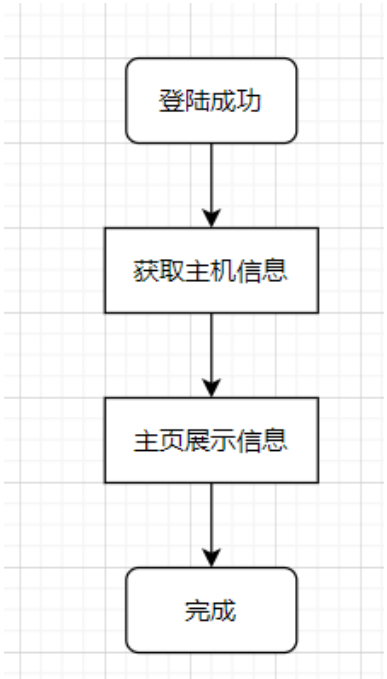

图 3-5 主页功能流程设计