摘　要

为了优化 OpenStack 监控模块在功能和可视化方面的不足，确保 OpenStack 实例的稳定运行，设计出一个用于 OpenStack 的监控系统。通过使用 Libvirt、Python 编程语言、Django 框架、Echarts 等开发技术，完成了拥有实时监控、实时报警功能的监控系统。该系统主要是通过每秒数据的输出绘制成图表来实现实时监控，还原当前运行资源的真实性。

最后在本地虚拟环境下搭建 OpenStack 平台进行了测试，该系统能够准确的输出实例当前的资源状态，实现报警等功能。解决了 OpenStack 监控模块在功能和数据可视化方面的缺陷，并且对于 OpenStack 监控模块从 Controller 节点获取资源数据而言，该系统直接从 Computer 节点获取资源数据，从而节省了 Controller 节点资源。

关键词：OpenStack；资源监控；虚拟化；Libvirt；Python

Abstract

In order to optimize the function and visualization of the OpenStack monitoring module and ensure the stable operation of the OpenStack instance, a monitoring system for OpenStack was designed. Through the use of Libvirt, Python programming language, Django framework, Echarts and other development technologies, a monitoring system with real-time monitoring and real-time alarm functions has been completed. The system mainly realizes real-time monitoring by drawing data output per second into a chart, restoring the authenticity of current running resources.

Finally, the OpenStack platform was built and tested in a local virtual environment. The system can accurately output the current resource status of the instance and implement alarm and other functions. Solved the defects of the OpenStack monitoring module in terms of function and data visualization, and for the OpenStack monitoring module to obtain resource data from the Controller node, the system directly obtains resource data from the Computer node, thereby saving Controller node resources.

Key words： OpenStack; resource monitoring; virtualization; Libvirt; Python

目 录

第一章 绪论

1.1 项目研究背景及研究意义

从二十世纪开始，云计算技术就受到人们的赞赏和欣赏。从谷歌的成立到亚马逊尝试时的突破，它已成为一个巨大的工业和生态链。被称为新世纪以来最大的技术进步之一。今日云计算的发展“蓬勃”并非没有理由。由于云计算可以产生较少的本地硬件和软件，所以本地计算机只能执行云计算系统的接口软件。其余工作由网络计算机、服务器和数据存储系统负责[1]。

基础架构即服务一般都包含了平台即服务与软件即服务。而绝大多数情况下基础架构即服务都是需要虚拟化技术的支持。通过用户层的虚拟化管理器，从真实硬件资源中虚拟出硬件资源，从而去新建虚拟机、删除虚拟机、迁移虚拟机。

单台主机上的虚拟机可以通过简单的命令来管理，但是在现实生产环境中往往需要考虑环境的高可用性、负载均衡以及备份等问题。因此会使用多台主机集成虚拟化平台。这样做会带来虚拟机运维上面的问题。这就需要应用软件来辅助人员去完成虚拟机任务的管理。这就是基础架构即服务系统产生的原因，也是 OpenStack、AWS 等项目所实现的价值。

而云平台资源监控是一个关键的技术支撑，它是云平台资源管理、迁移、调度、故障、分析、负载、报警与恢复的前提，监控能帮助云平台资源合理化使用、调度、定位故障、提升用户体验，大大提高了云平台的服务效果。

因此，设计出一个面向云平台的监控系统十分必要。其中 OpenStack 是当前主流的云计算技术之一，发展速度也飞快。对此本课题将设计实现出一个针对 OpenStack 的监控系统。

1.2 项目国内外研究现状

在中国，阿里云监控系统是阿里巴巴云服务的监控系统。阿里云监控的执行可以检测阿里巴巴云资源情况，使阿里巴巴云用户能够快速了解他们使用的实例的资源和性能。阿里巴巴的云监控主要可以监控自己的云服务、可以监控自己的网站和定制自己需要的监控需求来进行监控。通过监控，用户可以清楚地看到自己监控对象的数据。还可以使用资源监控报警功能，当用户设置一个报警值，若用户使用的

服务超过设置的报警值，则会通知用户。但目前阿里云监控并不是全面免费开放。

百度云亦是国内优秀的云厂商，而百度云观测是百度开发的一个监控系统，百度云观测的主要服务对象是网站的站长，它针对网站的站长设计出这一个对网站的安全监控系统，它提供给站长报警服务，并且监控功能覆盖全网站的运行状况、安全和访问速度等。并且它提供的是长期免费的服务。但主要是提供网站的监控服务。

从国外角度来看，亚马逊云观察（Amazon CloudWatch）是亚马逊开发的一个云监控系统，通过亚马逊云观察，用户可以得到亚马逊云的资源情况，能通过资源情况得到资源能力范围，并且可以通过日志发现云资源问题。亚马逊云观察能够做到很细微的监控，它能具体监控进程、数据库等。当然，亚马逊云观察也可以设置一个报警值，用来告诉用户当前云资源出现问题。

TeamViewer 是国外著名的软件公司，Monitis 是 TeamViewer 旗下的一个子公司开发出的一个监控系统，它提供了市场上比较好的云计算无代理监控功能，并且有完善的监控功能，也能支持用户自定义监控、分析数据等，重要的是它拥有一个良好的交互页面，使得管理者在管理上得心应手，相信所有管理者在第一次接触管理系统时都希望使用一个简单易上手的交互页面。

1.3 论文结构安排

第一章说明了为何要针对云计算平台开发一个监控系统。

第二章介绍了选用的云计算平台 OpenStack，使用技术 Python 与 Django 框架、Libvirt、Echarts 组件。

第三章对 OpenStack 监控系统的用户、数据模块进行需求分析。

第四章则是设计实现 OpenStack 监控系统的用户、数据模块以及模块当中的各项功能。

第五章对 OpenStack 监控系统进行测试。

第六章是对监控系统设计实现的结束总结以及系统优化改善的方法。

1.4 本章小结

本章对云计算平台发展、云监控重要性做出了分析。国内外的大厂商都争先的发展研究云计算，国内有阿里、腾讯，国外有谷歌、亚马逊。并且以上的厂商都拥有自己的云监控平台。但是开源的云监控学术研究还是寥寥可数。仍需要我们为云

监控学术做努力。本文就目前主流的开源云计算技术 OpenStack 云平台进行研究，

设计开发出一个监控功能较完善的 OpenStack 监控平台，并且可以提供个人使用。

第二章 OpenStack 与监控系统相关技术

2.1 OpenStack 介绍

2.1.1 OpenStack 概述

OpenStack 是由 Rackspace 与 NASA（美国国家航空航天局）联合研发的管理云平台操作系统，可以控制一个数据中心内的大型计算，存储以及网络资源池，所有这些资源均通过仪表盘进行管理，该仪表盘可让管理员进行控制管理，同时授权其用户通过 Web 界面配置资源[2]。作为一个开源的项目，所有人都可以为其提交代码，开发、修改功能，使其不断衍生出新的项目，对此 OpenStack 成为使用人数较多、应用范围较大的云平台之一。

2.1.2 OpenStack 架构

一般云计算平台都是由各种组件构成，OpenStack 也不例外，OpenStack 当前比较重要的组件有：

（1）Horizon（UI 交互），Horizon 提供了基于网页的 UI 交互 ，即可以通过 UI 完成对实例的操作。

（2）Glance（镜像），Glance 能够存储、检索和注册磁盘镜像。

（3）Neutron（网络），Neutron 主要能够提供网络。

（4）Keystone（认证），Keystone 主要能够给其他 OpenStack 服务进行认证以及授予权限，相当于对操作指令请求者的确定性进行鉴定。

（5）Swift（对象存储），Swift 是一种高可用的分布式对象存储服务，其可以为 OpenStack 提供可拓展的存储备份系统。

（6）Cinder（块存储），Cinder 主要是为实例提供持久性的块存储。

（7）Nova（计算），Nova 主要能够管理实例。用户操作实例也是通过它来进行。

OpenStack 各个项目之间既是互相独立又是互相紧密联系的。独立是指各个项目之间的安装部署都是相互隔离，互不干扰的。而紧密联系是指各个项目之间又是互相协调工作的。总体架构如图 2-1。

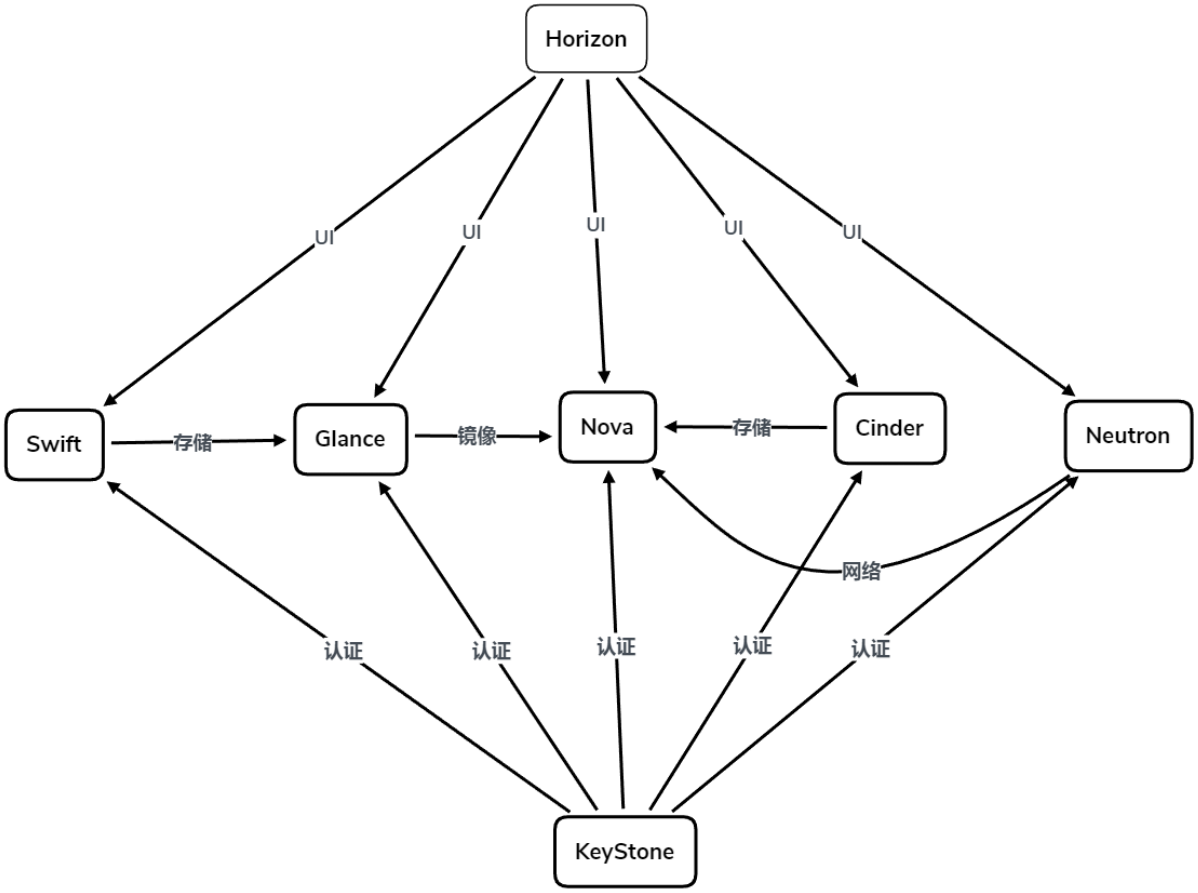

图 2-1 OpenStack 架构图

2.2 Python 与 Django 框架

2.2.1 Python 语言

Python 是一种开放的、面向对象的高级计算机编程翻译语言，第一次于 1991 年公布[3]。Python 是强大的，因为 Python 目前已经拥有一个富有而强大的模块库，因为这个库，Python 处理事情十分的快速方便。不仅如此，Python 还可以在其它语言中使用它们的各种模块 [4]。

Python 语言有以下几个特点：

（1）容易学习，不仅语法简单、结构清晰而且固定关键字少，使得 Python 拥有良好的阅读性。正式因为 Python 的简单理念，使得少量代码能完成的需求在其他语言里面可能需要上百行代码才能完成。由此可以看出，Python 语言是十分简单易学的。

（2）面向对象，采用了“以程序为导向的编程方法”，该程序包括可重复使用该代码的功能或程序。使用面向对象的编程，程序由对象构造，对象由数据和功能

组成。

（3）可扩展性，可以使用 C 或 C++来写入某些程序，以加速执行批判码，或防止发布某些重要算法，然后再用 Python 的程序来呈现。Python 语言类的模块库不仅强大，而且非常丰富，它可以很容易地将其它语言（尤其是 C 语言或 C++语言）制作的各种模块连接起来，并且可以扩展 Python 的功能。

（4）模块库，模块库也能理解成一个十分巨大的对象库，是前人为做某件事情而编写的“工具”，Python 能够引用它们，来帮助编写者快速进行编程。

2.2.2 Django 框架

Django 用于开发网站，而且遵循 MVC 的设计理念。劳伦斯出版集团开发了这一框架，最初是用于开发一个侧重于新闻内容的网站，后来才在社区开源。它强调了代码的再使用，多个组件可以很容易地以“插件”的形式为整个框架服务，Django 有许多强大的第三方插件，甚至您也可以很容易地开发自己的工具插件[5]。因为 Django 遵循了 MVC 的设计理念，因此 MVC 有的特性 Django 都有，但 Django 不满足 MVC 的开发速度，并加以改造，成为 MVT 模式。当中，V 做的是处理逻辑，T 为页面，而 M 则被封装成一个对象使用。在 MVT 处理模式中，实现了一个更符合项目开发过程的升级处理构想。在这种处理模式中，具有相对简单功能的控制器部分被封装在路径中，并且路径被用于完成请求的分发，路径通过配置实现（即 Django 框架中的 url 与 setting）。

2.3 Libvirt

Libvirt 是目前为止使用最为广泛的 KVM 虚拟机进行管理的工具和 API[6]。Libvirt 由三个部分组成，分别是编程语言的 API、进程、命令操作工具。Libvirt 支持许多语言对它进行模型包装，包装完后提供给该编程语言使用。2014 年前释放的代码包括 Python 界面，经过 2014 年的一个版本形成独立的 Libvirt-Python 库。Libvirt 还实现了 Ruby、Java、Perl 等语言的绑定。而且 Libvirt 也适用多种类的虚拟机监视器。Libvirt 可以做到：即使域不相同，还能对实例进行操作，普通操作同时包括了资源操作；通过执行本地 Libvirt-daemon，本地和远程机器可以访问和使用 Libvirt 功能。远程则是配置 SSH；可用于管理和创建虚拟网络以及管理物理和逻辑网络接口等。目前 Libvirt 已经在生产环境中广泛使用。应用程式包括各种控制线工具、图形接口工具、连续集成、Web 应用程序、IaaS 服务（其中就包括 OpenStack），从而极大地促进了开发和集成。

2.4 Echarts

Echarts 是百度的一款 Javascript 开放源软件，可以支持多种浏览器使用。Echarts 提供了 API 接口和丰富的文档，通过正确定义和组合背景下发送的 Json 数据，可以显示所需的数据图标[7]。Echarts 具有各种功能和效果以及友好的视图，提供给开发者良好的数据显示。Echarts 的视图类型特点有许多，其中包括折线、圆球、水球、区域、动态、易用性、可拓展性等。Echarts 的特性可以使数据可视化分析将提供更高的性能和更广泛的适应性。在开发使用主题数据的系统时，可以根据用户的需要设计和设计网页数据显示格式和后端数据显示的方法。这大大提高了用户与开发者控制数据的能力和用户与数据处理的能力。

2.5 小结

本章主要是对 OpenStack 监控系统需要的技术做介绍。先是对 OpenStack 做了简单的介绍，并且说明了 OpenStack 的主要核心项目，分析了 OpenStack 的基础架构。然后对 Python 与 Django 框架做了简单分析与总结。之后说明了 Libvirt 与 Echarts 的作用。了解并掌握以上的技术后，就可以对项目进行开发。后续的章节将主要讲述系统各个模块需求分析、模块设计与实现、系统测试与部署。

第三章 OpenStack 监控系统需求分析

3.1 OpenStack 监控系统模块及工作流程

3.1.1 OpenStack 监控系统模块需求分析

OpenStack 监控系统是一个面向监控人员可以方便监控 OpenStack 云平台资源使用情况的系统，该系统的主要功能是让监控人员登陆系统后能对 OpenStack 实例、OpenStack 的 Computer 节点的资源（包括 CPU 利用率、内存利用率、网络流量吞吐、硬盘使用情况等）当前状态的把控，并且可以有效的防止实例或系统状态异常所带来的一系列问题。因此对本系统的实现提出用户模块与数据模块。用户模块主要是区分监控人员与管理人员，里面包含了创建账户、登陆、修改密码等功能。数据模块主要是用于数据的应用，里面包含监控与报警功能。系统模块需求分析如图 3-1 所示。

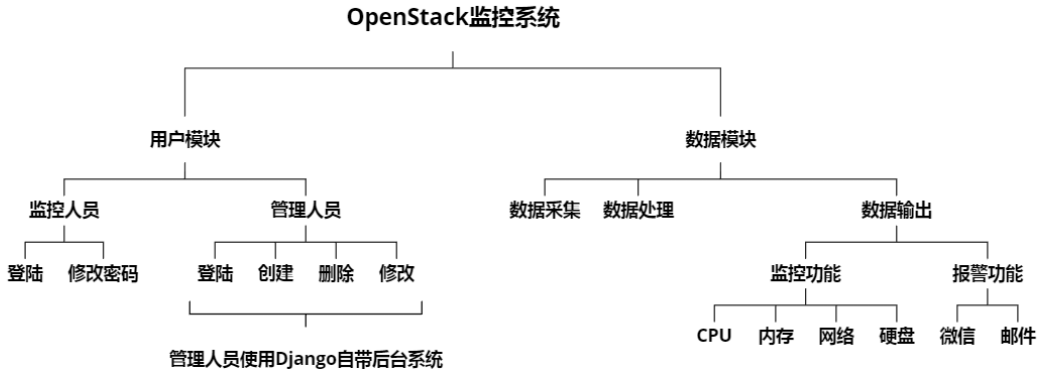

图 3-1 OpenStack 监控系统模块图

3.1.2 OpenStack 监控模块间工作流程

系统的模块间工作流程是基于 OpenStack 平台部署以及实例运行完毕后，首先由用户模块中的监控人员组在登陆模块进行登入系统，再由 OpenStack 监控系统的数

据模块采集获取所需的监控资源数据，再对采集的资源数据进行处理，之后将处理完的数据存入数据库中，最后由监控系统中数据模块的监控功能在前端页面展示，报警功能则是当数据有异常发出报警。综上所述及图 3-1 得出，系统之间两个模块及功能互相工作，密不可分。模块工作之间流程关系如图 3-2 所示。

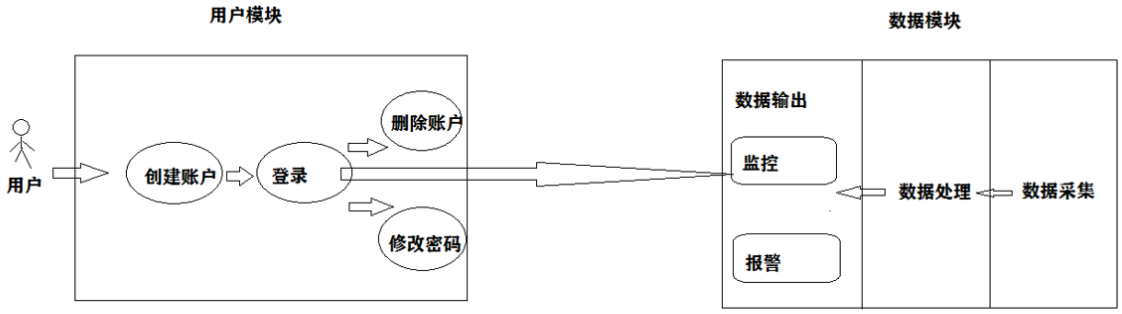

图 3-2 OpenStack 监控系统模块关系流程图

3.2 用户模块需求分析

OpenStack 监控平台用户管理模块主要是用于用户的权限区分。监控人员不需要具有注册账号的权限，注册账号由管理员进行操作。因此整个系统的登陆页面有两个，其中管理员登陆页面使用的是 Django 自带的后台登陆页面。而监控人员的登陆模块则是前端登陆页面。并且两个登陆功能的账号不互通。综上提出对用户模块的需求分析（如图 3-3 所示）：

（1）管理人员用户需求：作为系统的管理者，具有监控用户的增、删、改、查等操作权限。既可以为监控人员创建登陆账号，为监控人员初始化密码，并且可以知道监控人员账号所有信息，拥有监控用户所有权。

（2）监控人员用户需求：而监控人员不能自行注册监控账号，只能登陆指定账号对 OpenStack 实例进行监控，只拥有改、查等操作权限。即只能够修改自己账号密码，当监控人员忘记自己密码时只能联系管理员进行修改或者告知，能够看见自己账号的信息。

（3）登陆功能需求：登陆功能主要是用于监控人员登陆监控平台以及管理人员登陆后台系统。判断是否能登陆平台及后台的唯一凭证是用户名与密码。并且管理人员与监控人员拥有两个不同的登陆页面。管理人员使用的是 Django 框架自带的后台登陆页面，而监控人员则使用本系统的登陆页面。两种用户的区别是管理权限不

同。但登陆凭证一样都为自己的账号密码。管理用户以及监控用户都需要输入正确的账号密码才能登入系统以及后台。

（4）创建账号需求：监控用户不能自主创建账号，只能通过管理员发放的账号登陆。因此平台的登陆模块不需要注册功能。

（5）修改密码需求：监控人员与管理人员都能够自主修改密码。并且管理人员还能对监控人员进行密码修改，当监控人员忘记密码是，只能通过联系管理人员进行密码修改或告知。

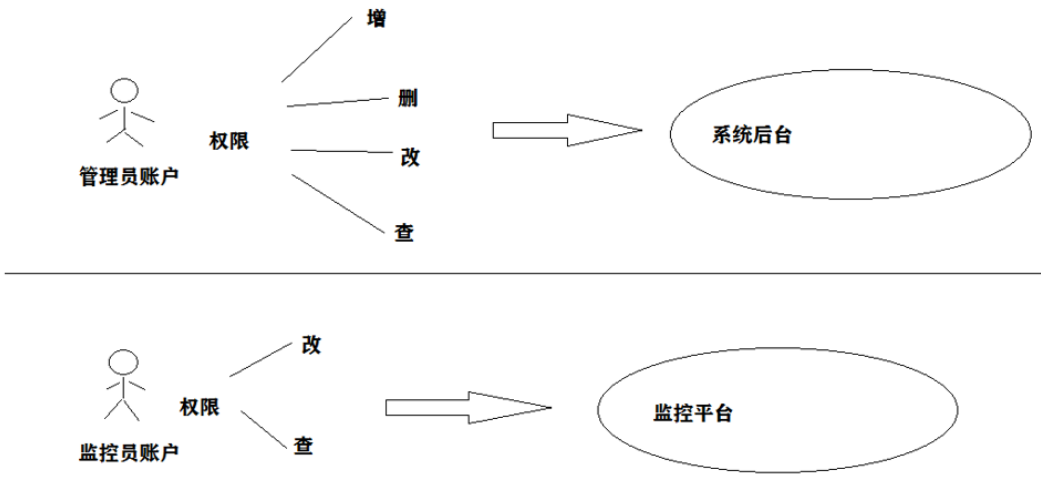

图 3-3 OpenStack 监控系统用户模块需求分析图

3.3 数据模块需求分析

OpenStack 监控平台数据模块分别实现 OpenStack 实例资源数据的采集、数据的处理以及数据的输出。因此提出数据管理模块需求分析（需求分析图可参考图 3-2）：

（1）数据的采集：监控平台需要及时准确的清楚当前实例的状态，因此 OpenStack 资源数据的采集需要准确、实时。不仅要采集 OpenStack 实例的资源，同样也需要采集物理主机的资源，因为物理主机的状态决定了实例的状态。采集时需要区分这两个区别。

（2）数据的处理：监控平台需要做到 OpenStack 实例以及物理主机实时监控，则代表采集的数据量庞大，首先应做到区分实例资源数据与主机资源数据，需要将实例资源数据与主机资源数据分开存储。在实例资源数据与主机资源数据分开存储的同时也需要将不同的资源种类也分开存储。这样更有益于我们对数据的使用。不同的资源数据可能拥有不同的特性，我们可以将采集的资源数据进行格式统一化。

这样会更加方便存储与处理应用。

（3）数据的输出：系统采集、处理过后的资源数据需要在监控平台前端输出，输出的首要方式为动态图表显示，动态图表需要能直观的观察到 OpenStack 实例以及物理主机的实时状态，因此需要简洁明了。次要方式为文字表达，用文字显示监控时期每个时刻的实际状态数据，有利于监控人员观察实际数据发现问题。并且能够下载监控时期的状态图表。

（4）监控功能：OpenStack 监控平台监控功能主要是用于查看 OpenStack 云平台上运行实例的状态。用户可以清楚的了解当前云实例的状态。监控人员需要准确知道当前实例的运行状态，因此资源数据的采集需要确保及时与准确，这是监控功能的前提。数据保存格式统一方便保存与使用。做到实例实时的监控。能够保存历时监控趋势图，并且不仅要图表显示，需要有真实文字监控数据。

（5）报警功能：当实例出现异常时，监控人员则需要精准及时的收到实例错误状态，由于每个使用者的实际情况可能不同，报警内容能够支持自定义设置，并且按实际要求设置报警阈值，达到这个阈值系统则发出报警，当然也含有不报警的选项。报警方式可以有邮件报警、微信报警。

3.4 小结

本章首先描述了 OpenStack 监控系统的整体模块结构还有工作流程。然后对系统的用户模块的登陆功能、创建功能、删除功能；数据模块的数据采集、数据处理、数据输出、监控功能、报警功能等进行需求分析。详细描述了两个模块的工作内容、两个模块所需要提供的功能、各个功能之间的联系，并且提出以及区分管理人员和监控人员两个概念，为后续的设计与实现规划了大致系统框架，并且奠定了基础。在接下来的章节，会根据本章的需求功能分析进行具体的设计与系统开发。

第四章 OpenStack 监控平台设计与实现

4.1 监控平台整体结构设计

选择 Python 编程语言进行系统开发的原因很简单，整个 OpenStack 项目就是用 Python 完成的，并且提供 HTTP 形式的 RESTful API 与 Python Client API 两种 API，而且在数据采集方面考虑到不能对 OpenStack 服务器性能影响过大的情况下，选用脚本采集数据的方法来进行数据采集，再存进数据库服务器。因此 OpenStack 监控平台选择使用 Python 语言进行开发。

根据 Django 框架的特点，满足快速开发监控系统的需求。因此监控平台选择 Django 框架作为系统的后端框架，AdminLTE[8]框架作为系统的前端框架，用作前端 Web 页面渲染。在采集数据的工作上，采用 Python 编写的脚本对 OpenStack 服务器进行数据采集，并且使用 Mysql 数据库存储数据。监控系统整体结构如图 4-1。

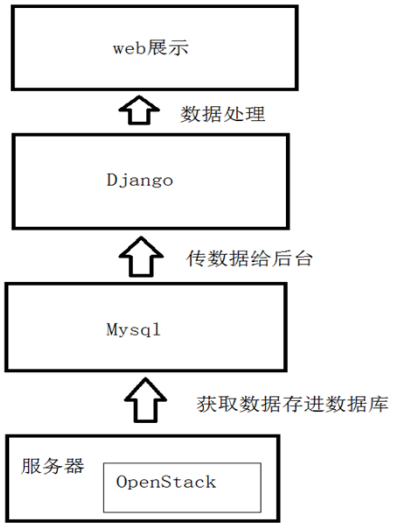

图 4-1 OpenStack 监控系统整体框架图

OpenStack 监控平台的开发都按照 Django 项目开发流程进行，监控系统项目结构如图 4-2 所示。

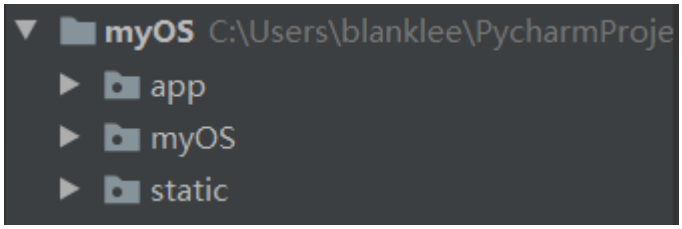

图 4-2 OpenStack 监控系统代码结构图

如图 4-2 所示，整个 OpenStack 监控系统包括三个部分。其中 app 为监控系统的具体实现部分，负责整个系统的展现与逻辑处理，里面包含了所有模块的实现代码，是整个监控系统的核心部分。myOS 为监控系统的配置文件，其中包括配置所需的数据库、中间件与监控系统入口 url 等。static 则是监控系统的前端渲染文件目录，里面包含的是 AdminLET 渲染文件。

4.2 用户模块设计与实现

4.2.1 用户创建

由于监控系统的账户由管理人员来创建，而且 Django 框架自带一个管理后台，当管理人员登陆后台后，可以为用户组新建用户，如图 4-3 所示。APP 内用户则是监控人员用户组，可以点击增加来新增监控人员账户，认证授权用户则是管理员用户组，可以增加一名管理员。

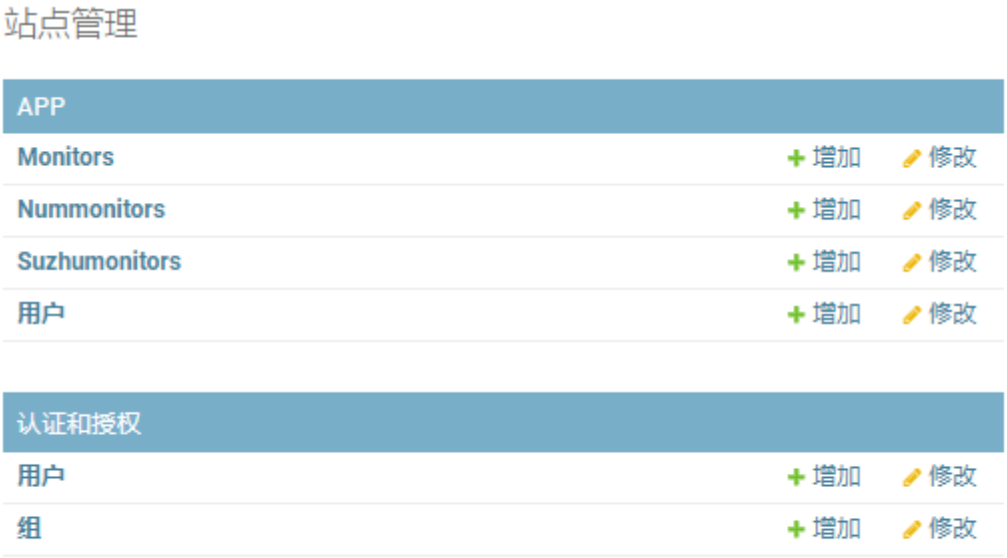

图 4-3 创建用户图

创建监控人员账号需要填写用户名、密码、邮箱、性别等基本信息，需要在 models 中添加模型代码如图 4-4 所示，实现出来创建用户过程如图 4-5 所示。实际过程就是通过后台创建一用户表，通过管理员点击增加用户完成监控员注册的部分，在系统数据库中添加一条账户信息，使得监控人员能够获得账号密码登陆监控平台。

```python
class User(models.Model):

    gender = (
        ('male', "男"),
        ('female', "女"),
    )

    name = models.CharField(max_length=128, unique=True)
    password = models.CharField(max_length=256)
    email = models.EmailField(unique=True)
    sex = models.CharField(max_length=32, choices=gender, default="男")
    c_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

    class Meta:
        ordering = ["-c_time"]
        verbose_name = "用户"
        verbose_name_plural = "用户"
```

图 4-4 创建用户模型代码

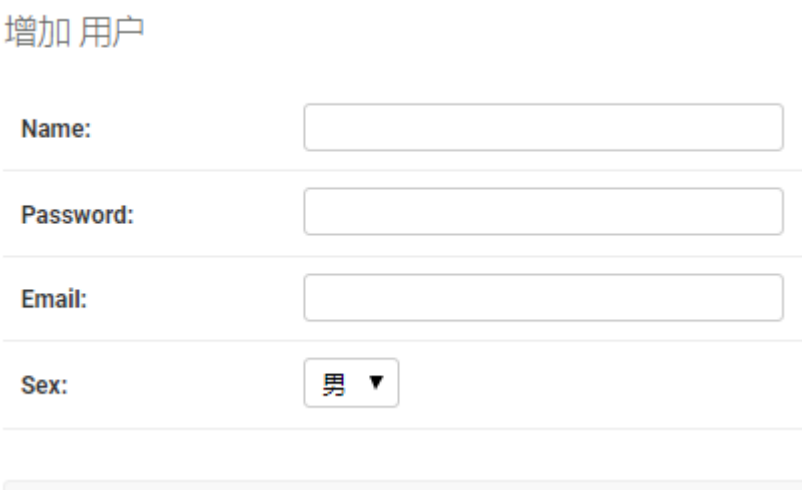

图 4-5 创建用户过程图

4.2.2 用户登录

4.2.2.1 用户登陆流程

用户登录流程图如图 4-6 所示，监控人员若进入系统登录界面进行登陆操作，系统后台会先检测监控人员是否输入账号密码，若无则提示输入。输入账号密码后，后台继续对输入的账号密码与数据库进行匹配，匹配成功则登陆成功，进入系统，匹配错误则提醒监控人员输入正确的用户名密码。

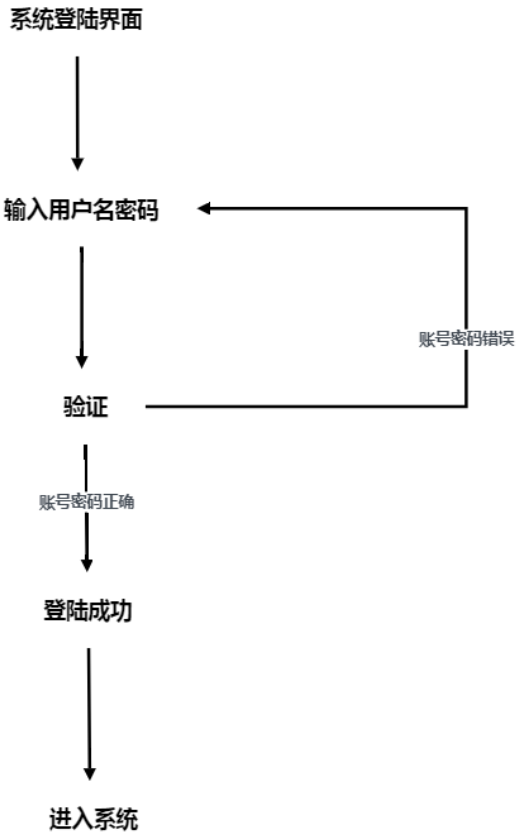

图 4-6 OpenStack 监控系统登陆流程图

4.2.2.2 用户登陆界面实现

登陆页面由一个文本表单、一个密码表单以及一个按钮构成，如图 4-7 所示。

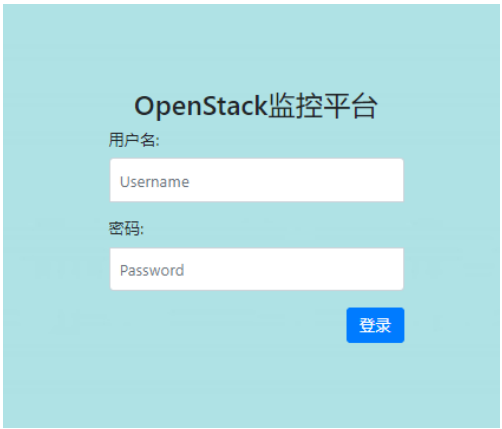

图 4-7 登陆页面

登录功能核心逻辑实现如图 4-8 所示，login_form 是先定义创建好的表单，is_valid()是 Form 表单里面封装好的校验方法，结果返回 True 或者 False，当校验到表单中有值返回 True，反之返回 False，校验过后会将表单中的值存在 cleaned_data 字典中，login_form.cleaned_data.get('username')就是获取校验过后表单中的用户名，密码同理。models.User.objects.get(name=username)是数据库匹配，使用 try 异常机制，

执行如果错误，说明数据库中不存在该 username，就转回登陆页，如果能够正常获取 username，则进行 user.password == password 密码匹配。request.session['is_login'] = True 是创建 session 字典保存当前用户登录状态为 True，下面的 user_id、user_name 同样是使用 session 字典保存当前用户的信息。整体登录逻辑流程为当监控人员输入账号密码进行登陆时，后台会先获取输入的值，若为空值则提醒用户输入正确内容。如果输入的用户名在数据库中存在，则进行下一步密码匹配，当密码匹配成功后，会分配出一个 session 字典写入用户状态和数据，session 会保存在后台，目的是用 session 记录用户登录状态，这样做的好处可以防止用户重复登陆与用户数据调用。当密码匹配不成功，则提示密码不正确。

```
    if request.session.get('is_login', None):
        return redirect('/app/dashboard/')
    if request.method == 'POST':
        login_form = forms.UserForm(request.POST)
        message = '请检查填写的内容！'
        if login_form.is_valid():
            username = login_form.cleaned_data.get('username')
            password = login_form.cleaned_data.get('password')

            try:
                user = models.User.objects.get(name=username)
            except:
                message = '用户不存在！'
                return render(request, 'app/login.html', locals())
#            if user.password == hash_code(password):
            if user.password == password:
                request.session['is_login'] = True
                request.session['user_id'] = user.id
                request.session['user_name'] = user.name
                return redirect('/app/dashboard/')
            else:
                message = '密码不正确！'
                return render(request, 'app/login.html', locals())
```

图 4-8 登陆核心逻辑代码

4.2.3 密码修改

4.2.3.1 修改密码流程

修改密码的流程如图 4-9 所示，在修改密码时系统后台会先检测监控人员是否输入正确的数值内容。当检测到有效内容值后，会继续判断输入的密码都是否输入正确。若都正确则修改成功，返回登陆页面。若错误则提示重新输入。

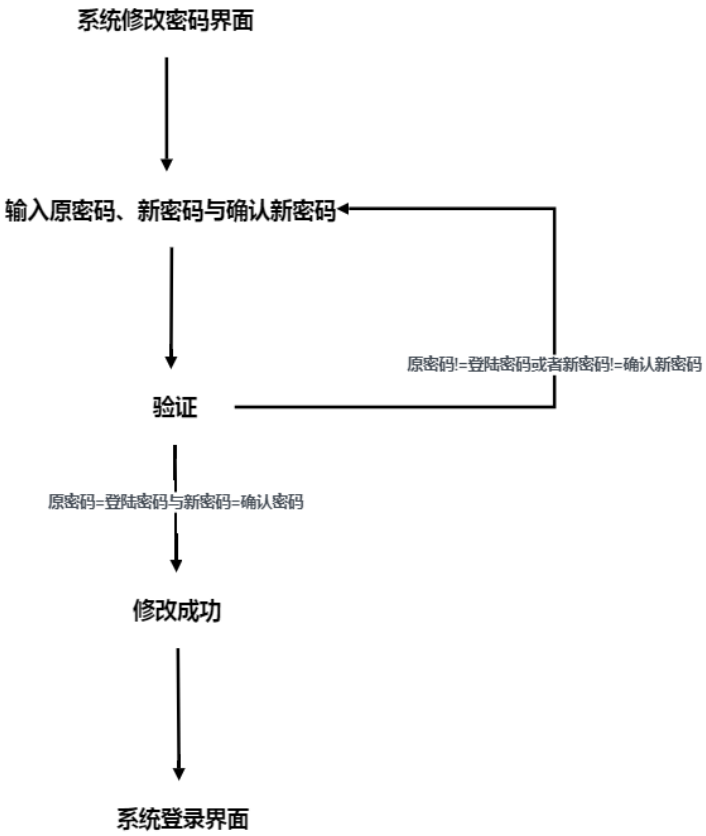

图 4-9 修改密码流程图

4.2.3.2 修改密码实现

修改密码页面由三个文本表单、一个超链接以及一个按钮所构成，如图 4-10 所示。

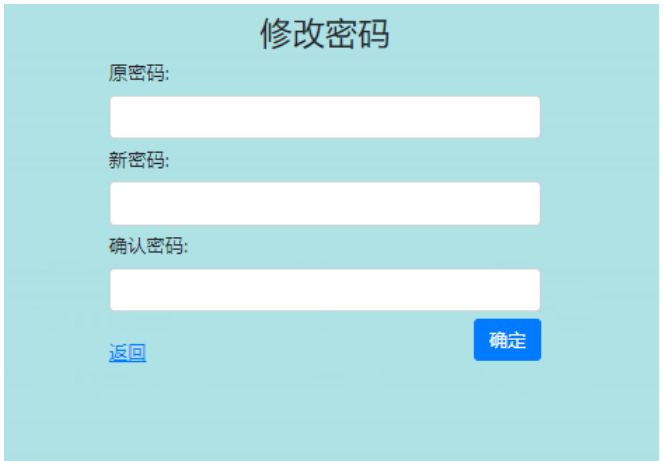

图 4-10 修改密码页面

修改密码功能核心逻辑实现如图 4-11 所示，代码实现使用的逻辑方法与登陆功能实现代码大同小异，先用 is_valid()方法校验表单，cleaned_data 字典保存表单值，再用 ifelse 语句判断密码是否相同，修改密码整体逻辑流程为：当监控人员进入修改密码功能入口时，系统会自动去到修改密码页面，如图 4-10 所示。监控人员需要按