**概要设计文档**

**V1.0**

[1引言 3](#_Toc242606401)

[1.1编写目的 3](#_Toc242606402)

[1.2背景 3](#_Toc242606403)

[1.3术语定义 3](#_Toc242606404)

[1.4参考资料 4](#_Toc242606405)

[2总体设计 4](#_Toc242606406)

[2.1需求规定 4](#_Toc242606407)

[2.1.1功能需求 4](#_Toc242606408)

[2.1.2性能需求 4](#_Toc242606409)

[2.1.3输入输出要求 5](#_Toc242606410)

[2.1.4数据管理能力要求 5](#_Toc242606411)

[2.1.5故障处理要求 5](#_Toc242606412)

[2.1.6其他专门要求 5](#_Toc242606413)

[2.2运行环境 6](#_Toc242606414)

[2.2.1设备 6](#_Toc242606415)

[2.2.2支持软件 6](#_Toc242606416)

[2.3基本设计概念和处理流程 6](#_Toc242606417)

[2.3.1 上报管理 8](#_Toc242606418)

[2.3.2 审核/批管理 8](#_Toc242606419)

[2.3.3 偿还报销管理 10](#_Toc242606420)

[2.4总体结构 12](#_Toc242606421)

[2.4.1系统整体架构 12](#_Toc242606422)

[2.4.2系统部署结构 13](#_Toc242606423)

[2.4.3子模块结构 13](#_Toc242606424)

[2.5人工处理过程 15](#_Toc242606425)

[2.6尚未解决的问题 15](#_Toc242606426)

[3接口设计 16](#_Toc242606427)

[3.1用户接口 16](#_Toc242606428)

[3.2外部接口 16](#_Toc242606429)

[3.3内部接口 16](#_Toc242606430)

[4运行设计 17](#_Toc242606431)

[4.1运行模块组合 17](#_Toc242606432)

[4.2运行控制 17](#_Toc242606433)

[4.3运行时间 17](#_Toc242606434)

[5系统出错处理设计 17](#_Toc242606435)

[5.1出错信息 17](#_Toc242606436)

[5.2补救措施 18](#_Toc242606437)

[5.3系统维护设计 18](#_Toc242606438)

# 1引言

## 1.1编写目的

本文档的编写目的是对Openstack云计算系统监控的架构进行说明，为后继的详细设计等工作提供参考和依据。

## 1.2背景

云计算使用共享的硬件基础架构提供计算服务，包括计算、存储、网络等资源。云计算使得客户不需要自己搭建基础设施，只需要通过按需付费的方式就能使用计算资源。这降低了客户的花费，也使得用户的应用程序更容易扩展、可维护性更强，加速了应用的开发速度。如今已经有众多的大型IT公司和创业型企业相继推出了自己的云计算服务，并实现了业务的快速增长，云计算市场呈现出欣欣向荣的景象。在这些公司里面有的使用自己开发的系统提供云服务，如亚马逊、微软、阿里云，但更多的公司则是使用Openstack作为提供云计算服务的基础架构。

Openstack起源于2010年，是Rackspace和NASA的合作发起的开源项目，通常被用于提供IaaS服务。目前Openstack在全球众多公司和开发者的支持下不断发展，成为最受欢迎的开源云计算管理系统。Openstack是一个管理、定义、使用云计算资源的框架。Openstack可以管理物理机和虚拟机、网络、存储系统，使用租户、配额、用户角色来使得管理更加高效，并提供了控制资源的统一接口。

监控是云计算重要的部分。云服务要求极高的可用性和稳定性，因而需要对各种资源进行监控。通过观察监控数据，运维人员可以对数据中心的资源使用情况有总体的了解、发现性能瓶颈、获知系统发生故障等意外情况，在这基础上才能优化系统、处理故障、提高系统可用性。此外，监控还能记录客户的资源使用情况，进而进行计费或者建议客户购买更大的配额。

## 1.3术语定义

1. Nagios：一个开源的监控框架，简单易扩展。

2. Ganglia：一个开源的监控工具，能够获取大量节点的性能信息。

3. Graphite：一个开源的性能数据存储软件。

4. Grafana：一个开源的展示时序图表的仪表盘。

5. flask：一个python web微框架。

## 1.4参考资料

1. Kevin Jackson, Openstack Cloud Computing Cookbook Third Edition, Packt Publishing Ltd., 2015
2. V. K. Cody Bumgardner, Openstack In Action, Manning Publications Co., 2016
3. David Josephsen，Nagios系统监控实践，机械工业出版社，2014
4. Massie，Ganglia系统监控，机械工业出版社，2013
5. 詹姆斯 特恩布尔,第一本docker书，人民邮电出版社，2014
6. 格林布尔，Flask Web开发：基于Python的Web应用开发实战，人民邮电出版社，2015
7. Lorin Hochstein, Ansible: Up and runing, O’Reilly Media Inc., 2015

Jason Myers&Rick Copeland, Essential SQLAlchemy, O’Reilly Media Inc., 2016

# 2架构设计

## 2.1需求规定

### 2.1.1功能需求

* 定义要监控的主机
* 定义主机上要监控的服务
* 在仪表盘上增加或减少监控的图表
* 调节图表的大小和样式

### 2.1.2其他需求

(1) 易用性：系统应该能够提供直观友好的界面、简单的交互方式。

(2) 可扩展性：系统应能允许使用者根据自身的需求进行扩展，监控更多的指标。

(3) 可靠性：系统应提供稳定的服务。

## 2.2运行环境

操作系统：Ubuntu server 14.04；

数据库：MySQL 5.6；

运行环境：Docker容器

## 2.3基本处理流程

管理员在web界面设置要监控的机器和服务，点击确认后，系统在后台对管理员的设置进行处理，把这些操作同步到监控系统中。监控系统按照配置获取对应的机器和服务的性能信息，并把信息存储起来。管理员要查看图表的时候，系统读取这些数据，并在页面上绘制图表。

## 2.4 系统架构

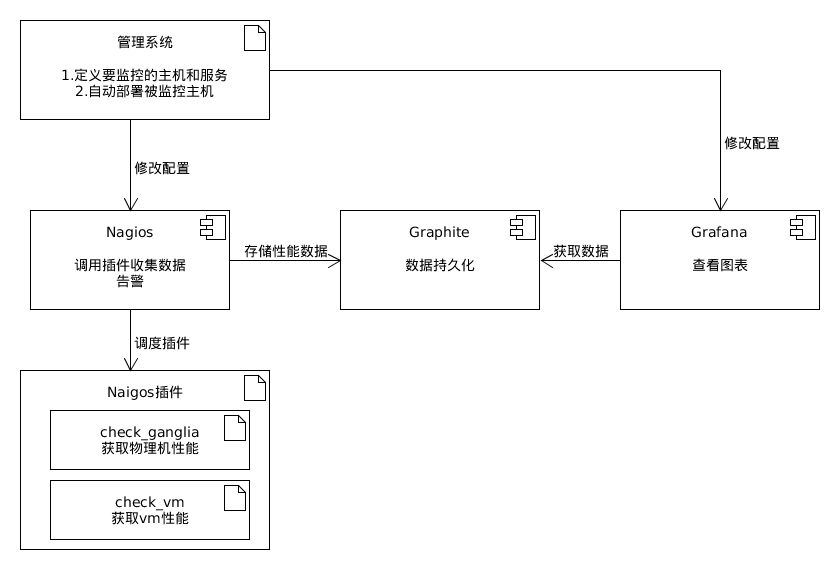


图4-1：架构图

管理系统提供了用户友好的界面方便用户定义要监控的主机和服务，并将用户的定义同步到Nagios的对象配置文件和Grafana的仪表盘。Nagios、Graphite、Grafana协作共同监控Openstack云平台的任务。Nagios负责调用插件check\_ganglia和check\_vm采集物理机和虚拟机的性能信息，并将性能信息存放到Graphite的数据库。Grafana负责将Graphite的数据展示到web页面的仪表盘上。

# 3接口设计

## 3.1用户接口

系统通过web页面为用户提供服务。Web页面设计风格简洁、重点突出、操作简单。用户可以通过web页面添加主机、服务，也可以批量添加一个IP段的主机，避免了大量的重复输入。用户可以在web页面查看仪表盘，还可以用所见即所得的方式编辑仪表盘的内容、样式。

## 3.2外部接口

该系统使用了多个开源软件，下面是系统与这些软件的交互方式：

Nagios：修改Nagios的配置文件，通过命令行重启Nagios服务。

Grafana：使用Grafana的API修改仪表盘。

Mysql：使用sqlalchemy来执行数据库操作。

## 3.3内部接口

内部基于功能划分为多个模块，向顶层隐藏了实现的细节。总共分为数据库操作模块、外部系统调用模块、web服务模块。数据库模块提供了数据库的操作、保证了管理系统内部的数据一致性。外部系统调用模块负责实现与外部系统的交互。Web服务模块负责渲染页面和处理用户的请求。

# 4运行设计

## 4.1运行模块组合

系统运行需要后台数据库、.Net Remoting、系统总控、完成特定数据管理功能程序模块和Winform显示控制几个部分协同工作。

## 4.2运行控制

系统需要先启动数据库服务器，启动无误后，各个客户端的用户通过实现获取服务器端的IP地址和端口号，就可以登录进入系统开始各种操作。

## 4.3运行时间

后台数据库服务器和应用服务器可以共同部署在一台服务器上，也可以各自占用一台机器，三个客户端可以在一台机器上，亦可以各自分开，通过局域网与服务器进行连接。

在运行是，应用服务器和数据库服务器必须同时开启，各个客户端则可以根据需要随时运行。

# 5系统出错处理设计

## 5.1出错信息

系统中的各种提示如表5-1所示：

表5-1 系统出错提示

|  |  |  |  |
| --- | --- | --- | --- |
| 故障或提示 | 系统提示信息 | 含义 | 处理方法 |
| 不能提交 | 不允许为空，请输入 | 必选项未填 | 重新输入 |
| 不能提交 | 不合法，请重新输入 | 输入数据格式不合法 | 重新输入 |
| 不能提交 | 数据项已经存在，请重新输入 | 所选数据记录在数据库中已经存在 | 重新输入 |
| 删除确认 | 是否确认删除 | 确认是否删除 | 根据需要选择 |
| 作废确认 | 是否确认作废 | 确认是否作废 | 根据需要选择 |
| 登陆失败 | 用户不存在或口令不正确，请重新输入 | 用户名或密码 | 重新返回登陆界面 |
| 数据库文件备份成功 | 数据库文件备份成功 | 成功备份数据库问价 | 无 |
| 数据库文件恢复成功 | 数据库文件恢复成功 | 成功恢复数据库文件 | 无 |
| 客户端连接不成 | 连接不成功，请检查网络连接 | 客户端不能连接上服务器端 | 检查网络状况 |
| 连接不上数据库 | 数据库连接失败 | 服务器连接不上数据库引擎 | 检查数据库连接字符串 |
| 借款请求 | X条借款请求 | 科室上报客户端提交了借款请求 | 根据实际情况操作 |
| 直接报销请求 | X条直接报销请求 | 科室上报客户端提交了直接申报请求 | 根据实际情况操作 |
| 偿还报销请求 | X条偿还报销请求 | 科室上报客户端提交了偿还报销请求 | 根据实际请款操作 |
| 申请完成提示 | 你提交的请求X已经被X审核通过 | 上报请求通过审核 | 无 |

## 5.2补救措施

1) 采用磁盘做备份准备，使用SQL Server 2005的Backup Server（备份服务）对数据库数据进行备份，如果系统遭到破坏，用备份的数据进行还原，数据的备份和还原可以通过应用程序实现，也可以通过系统管理员直接使用SQL Server 2005的Backup Server进行备份。建议用户每天对数据库中的数据进行备份；

2) 当系统运行效率过低时，通过重新启动可以重新组织数据库索引，提高系统运行效率。

3) 在系统运行的过程中，可能会突发一些不可预测的故障，如断电、死机等。为了提高系统的安全性，我们采用了基于挂接操作系统接口的服务器自身监控安全模型。在本系统的服务器操作系统中，通过远程DLL注入技术，修改操作系统中进程的导入地址表，挂接Windows操作系统的关机函数，截获Windows的关机消息，从而实现在服务器每次系统关机时，自动检测当前是否有正在运行的财务业务，保证所有业务都已顺利结束，并自动备份一次数据库，再转回Windows操作系统的关机执行。从而保障了系统服务器的业务稳定性，和数据完整性，提高了系统的安全性和稳定性。

## 5.3系统维护设计

系统采用了分层的结构进行设计，使系统各个部分分割开来，提高了系统灵活性和可扩展性。系统在三层架构的基础上，增加了一层公共层，将系统中通用的部分抽取出来，以便于系统的维护。在设计逻辑层时，我们采用了Façade模式，Facade模式基本框图如下：

客户端

客户端

客户端

门面Facade

门面Facade

网络

Facade模式

图5-1 Façade结构

其中小圆代表业务逻辑层中的小的功能，系统子模块通过“门面Facade”来

自己获取所需的功能，实现了“高内聚，低耦合”的设计要求。在系统维护的过程中，我们可以通过测试各个层次之间的接口即可达到系统维护的要求。