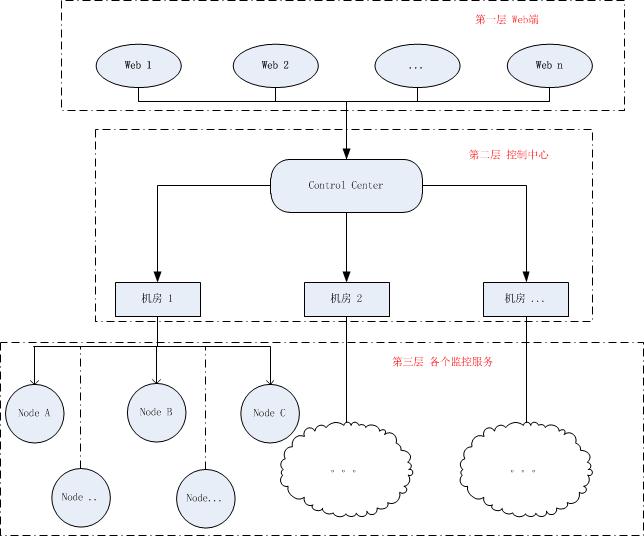
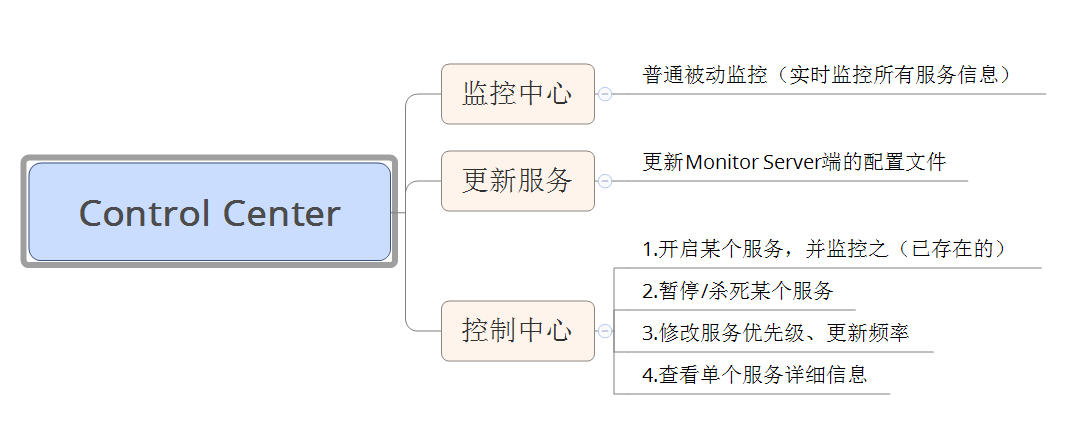
一. 项目需求

二. 组织架构

**Ⅰ. 架构说明**

本服务是一套监控系统，如上图分三层结构。第一层，远程web端访问；第二层， 中心；第三层，各个服务器（PC机）部署的监控服务。（见下图）

**1.Control Center:**

****

Control Center提供给web的有两个功能模块：监控、控制（更新服务不对外开放）。

Control Center从web端获取配置文件、或控制命令、或监控命令推送到机房，机房再分别分发给各个Node。

其中“机房”扮演了一个ip再分配的路由角色

**2.Monitor Server(Node)**

一个Node在一个PC机上，就是一个子监控系统

**Ⅱ. 语言选择**

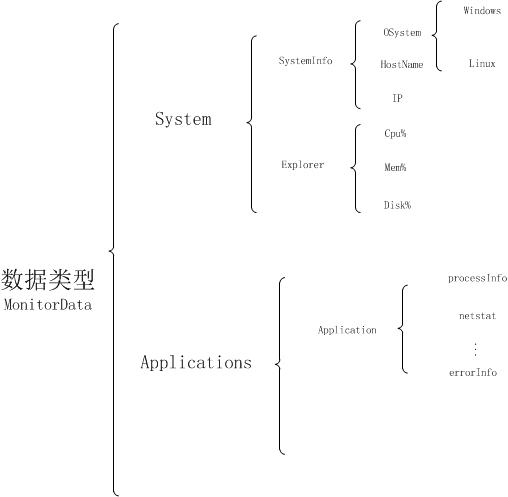
第一层，不限语言，实现图形化界面。

第二层，考虑到高并发和做服务器的优势，用golang1.7来实现

第三层，考虑到要和操作系统打交道，且要有强大的逻辑处理、字符处理、网络通信、和面向对象的编程需求，故用python3.5实现

三.数据结构

1. Monitor Server（node）



**System结构体：**

a. Explorer（资源管理器） （top |head -5）

系统CPU（%）

type SysCPU struct ｛

us float //用户空间

sy float //内核空间

io float //io等待

id float //空闲

｝

系统Mem（k）

type SysMem struct ｛

total int //总共

free int //空闲

use int //已用

cache int //缓存

｝

系统task（个）

type SysTask struct ｛

total int

running int

sleep int

stop int

zombie int

｝

文件系统Disk （df -h）

type FileSystem struct {

diskShow string

}

b. SystemInfo cat /etc/issue

系统信息，ubuntu？ redhat？ debian？ win7？ win10？

type SystemInfo struct {

oSystem string //linux? or windows?

ip string

}

最终的系统信息结构如下：

type System struct ｛

cpu SysCPU

mem SysMem

task SysTask

disk FileSystem

sysInfo SystemInfo

｝

**Applications结构体：**

type Process struct {

user string

pid string

cpu string

mem string

vsz string //程序完全驻留在内存, 需要占用的内存空间

rss string //实际占用的内存空间

tty string

stat string //当前进程状态

start string

time string

ppid string

command string

}

stat 中的参数意义如下：

   D 不可中断 Uninterruptible（usually IO）

   R 正在运行，或在队列中的进程

   S 处于休眠状态

   T 停止或被追踪

   Z 僵尸进程

   W 进入内存交换（从内核2.6开始无效）

   X   死掉的进程

   < 高优先级

   n   低优先级

   s   包含子进程

   +   位于后台的进程组

type Application struct ｛

process Process

net Netstat

log ErrLog

......

｝

type Applications struct {

Application

...

}

最终发送的数据结构MonitorData:

type MonitorData struct {

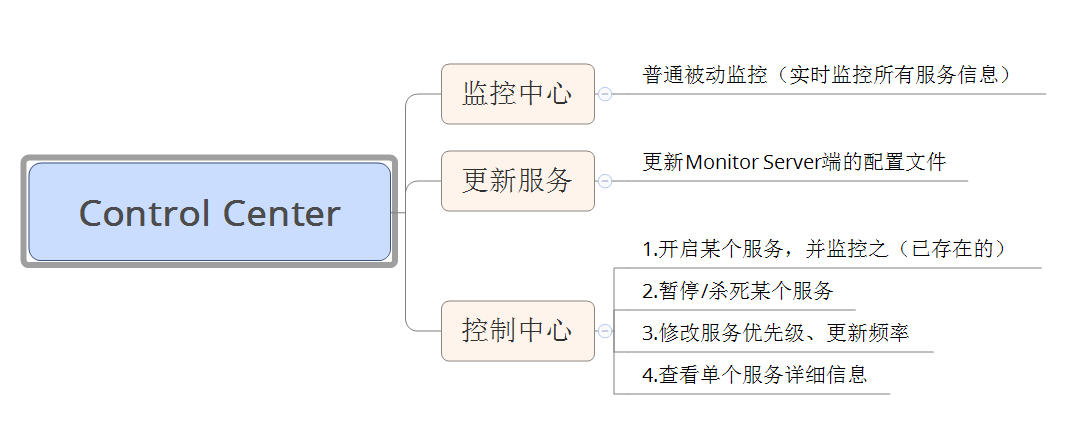
sysInfo System

appInfo Applications

}

四. 通信协议

1. **如下图所示，Control Center大致分为监控、控制、更新三大功能模块。**



2. **根据上图，设置数据发送结构：**

type SendCommand struct {

mode int

name string

command string

xml string

}

参数说明：

mode： 选择操作类型。0，1，2， 分别代表监控、控制、更新。默认mode=0

name： 服务名称（控制使用），默认为空。

command： start、stop、update、info（控制使用），默认为空。

xml： 本地配置文件路径（更新服务使用），默认为空。

3. **Post请求格式如下**：

POST /api/hostname/post HTTP/1.1

Host: 192.168.9.222:9090

Content-Type: application/json

Cache-Control: no-cache

Postman-Token: 7333090b-5d0c-0ee0-a6c4-bf352a5f0299

{

"mode" : 0,

"name" : "redis-server",

"command" : “stop”，

"xml" : ""

}

4. **Monitor Server返回数据格式：**

type ResultJson struct {

Code int `json: "code"`

Data string `json: "data"`

Message string `json: message`

}

参数说明：

code: 返回状态码，0表示成功，其他未定。

data： 查询到相应服务的json数据，map-csv-json。

message： 错误信息，或其他补充说明。

五. shell命令

**1. ps -aux | grep name //根据服务名，获取进程信息**

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

树形结构显示进程: **ps -axjf**

ps命令很强大很丰富，能以表头字段进行排序，可以根据通配符、正则表达式删选。

比如获取redis-server的pid：**ps -aux | grep redis-server | awk '{print $2}'**

**2. ifconfig**

**ifconfig ens33 | grep "inet addr"**

inet addr:192.168.9.222 Bcast:192.168.9.255 Mask:255.255.255.0

**ifconfig ens33 | grep "inet addr" | awk '{print $2}'**

addr:192.168.9.222

**ifconfig ens33 | grep "inet addr" | awk '{print $2}' | awk -F: '{print $2}'**

192.168.9.222

通过这种方式可获取系统的ip、网关

**3. hostname**

**4. netstat**

Netstat 命令用于显示各种网络相关信息，如网络连接，路由表，接口状态 (Interface Statistics)，masquerade 连接，多播成员 (Multicast Memberships) 等等.

**netstat -a 列出所有端口**

**netstat -at tcp**

**netstat -au udp**

**netstat -l 只显示监听端口**

**netstat -r 显示核心路由信息**

**netstat -i 显示网络接口列表**

**5. cat path > file.txt**

重定向服务出错日志

......