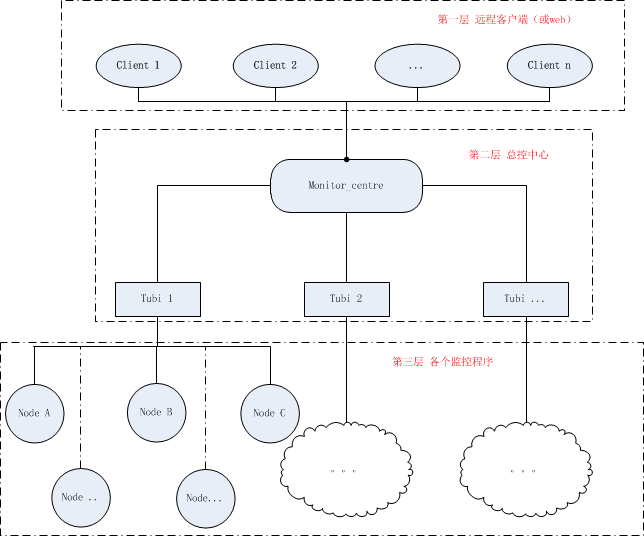
一. 项目需求

二. 组织架构

Ⅰ. 架构说明

本服务是一套监控系统，分三层结构。第一层，远程客户端端访问；第二层，总控中心；第三层，各个PC机上部署的监控程序。（见下图）

****

**Client：** 供用户配置、使用、控制的远程客户端

**Monitor\_centre:** 总控中心，下载client上的配置文件分发给Tubi，收集每个Tubi的数据并 反馈给client

**Tubi：** 分发配置文件和ip（路由）给旗下的每个Node，并收集Node返回的数据

**Node：** 监控程序

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

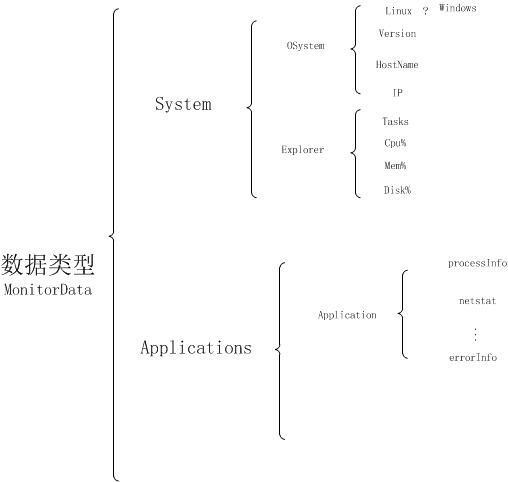
第三层，考虑到要和操作系统打交道，且要有强大的逻辑处理、字符处理、网络通信、和面向对象的编程需求，故用python + go实现

三. Monitor Node

Ⅰ. 通信协议

考虑到node与tubi之间需要长连接全双工的实时交互，故采用基于tcp/ip协议的websocket

Ⅱ. 数据结构



ResultRec：

|  |  |  |
| --- | --- | --- |
| 结构体 | 字段 | 类型 |
| ResultRec | Name | String |
| Data | String |

每一个Application下的每一条监控记录。Data为python处理后返回的数据，Name为Record 名。

ResultApp：

|  |  |  |
| --- | --- | --- |
| 结构体 | 字段 | 类型 |
| ResultApp | Name | String |
| Recs | []ResultRec |

每一个Application下的所有数据，Name 为该Application名。

Explorer：

|  |  |  |
| --- | --- | --- |
| 结构体 | 字段 | 类型 |
| Explorer | Tasks | String |
| Cpu | String |
| Mem | String |
| Swap | String |

分别是系统进程、Cpu、内存、交换空间等一系列信息。

Osystem：

|  |  |  |
| --- | --- | --- |
| 结构体 | 字段 | 类型 |
| Explorer | Sys | String |
| User | String |
| IP | String |
| Version | String |
| Platform | String |

分别是操作系统、用户、IP、系统版本、平台的一系列信息。

**最终的数据返回结构为：**

|  |  |  |
| --- | --- | --- |
| 结构体 | 字段 | 类型 |
| MonitorData | Apps | []ResultApp |
| Exp | Explorer |
| Osys | Osystem |
| Time | String |

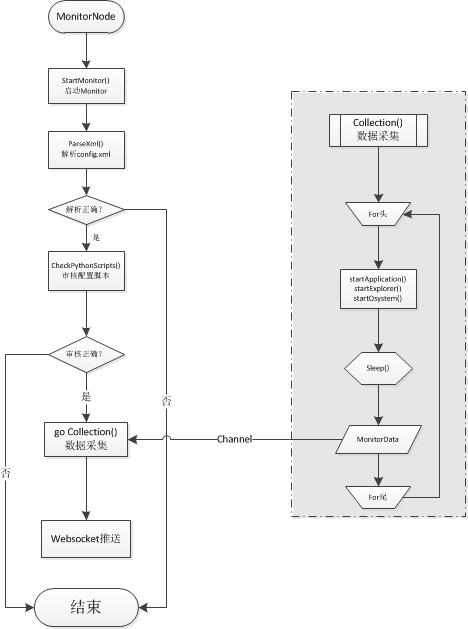
Ⅲ. 序列化与反序列化

目前暂时用encoding/json包来做序列化和反序列化

json.Marshal(v)

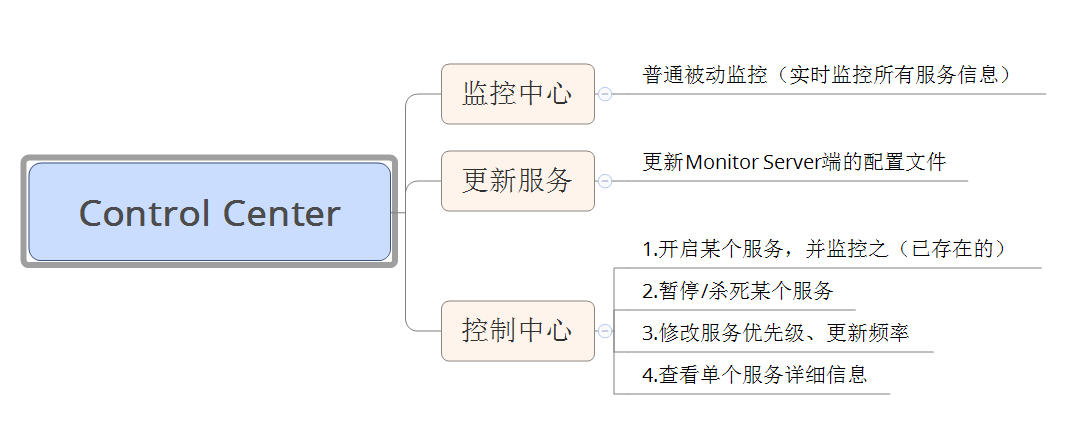
json.Unmarshal([]bytes, v)

Ⅳ.流程图

****

四. Monitor Tubi

五. Monitor Centre

****

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

1. **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" : ""

}

2. **Monitor Tubi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。

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

......