# cfg.json

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\judge\cfg.json

## 报警间隔

alarm中有一个minInterval的配置，单位是秒，默认是300秒，表示同一个event，如果配置报警多次，那么两个报警之间至少间隔300秒。  
这是个经验值，我们觉得报警太频繁没有意义，对工程师来说是干扰。收到报警之后拿出电脑、开机、连上vpn就差不多要3分钟了……

"alarm": {  
 "enabled": **true**,  
 "minInterval": 300,  
 "queuePattern": "event:p%v",  
 "redis": {  
 "dsn": "192.168.99.100:6379",  
 "maxIdle": 5,  
 "connTimeout": 5000,  
 "readTimeout": 5000,  
 "writeTimeout": 5000  
 }  
}

# 告警时间不一致

原因为192.168.0.179和192.168.0.181时间不一致，将时间同步即可解决

相差时间为两分钟



yum install -y chrony

systemctl start chronyd

systemctl enable chronyd

systemclt status chronyd

## Judge

### Main.go

Main.go入口函数

**go** http.Start()

**go** rpc.Start()

**go** cron.SyncStrategies()

**go** cron.CleanStale()

### 搭建redis测试环境

Cfg.json

"redis"**:** **{**

"dsn"**:** "192.168.99.100:6379"**,**

"maxIdle"**:** 5**,**

"connTimeout"**:** 5000**,**

"readTimeout"**:** 5000**,**

"writeTimeout"**:** 5000

**}**

docker pull index.alauda.cn/library/redis

docker run --name redis -d -p 6379:6379 index.alauda.cn/library/redis redis-server --appendonly yes

测试例子

lpush event:p0 "{\"id\":\"s\_1\_6666cd76f96956469e7be39d750cc7d9\",\"strategy\":{\"id\":7,\"metric\":\"cpu.busy\",\"tags\":{},\"func\":\"all(#3)\",\"operator\":\"\\u003e=\",\"rightValue\":90,\"maxStep\":3,\"priority\":0,\"note\":\"\",\"tpl\":{\"id\":7,\"name\":\"cpu.test\",\"parentId\":0,\"actionId\":2,\"creator\":\"liyang\"}},\"expression\":null,\"status\":\"PROBLEM\",\"endpoint\":\"192.168.1.55\",\"leftValue\":1,\"currentStep\":0,\"eventTime\":0,\"pushedTags\":null}"

json

{"id":"s\_1\_6666cd76f96956469e7be39d750cc7d9","strategy":{"id":1,"metric":"cpu.idle","tags":{},"func":"all(#3)","operator":"\u003e=","rightValue":90,"maxStep":3,"priority":0,"note":"","tpl":{"id":0,"name":"","parentId":0,"actionId":0,"creator":""}},"expression":null,"status":"","endpoint":"","leftValue":1,"currentStep":0,"eventTime":0,"pushedTags":null}

Alarm控制台

<Endpoint:192.168.1.55, Status:PROBLEM, Strategy:<Id:7, Metric:cpu.busy, Tags:map[], all(#3)>=1 MaxStep:3, P0, , <Id:7, Name:cpu.test, ParentId:0, ActionId:2, Creator:liyang>>, Expression:<nil>, LeftValue:2, CurrentStep:3, PushedTags:map[], TS:2016-08-09 09:07:00>

### cron包strategy.go

获取远程缓存

**func** syncStrategies() {

**var** strategiesResponse model.StrategiesResponse

    err := g.HbsClient.Call("Hbs.GetStrategies", model.NullRpcRequest{}, &strategiesResponse)

**if** err != nil {

        log.Println("[ERROR] Hbs.GetStrategies:", err)

**return**

    }

    rebuildStrategyMap(&strategiesResponse)

}

Hbs.GetStrategies调用hbs中的rpc模块hbs.go中GetStrategies

### Transfer\sender\send\_tasks.go Sender包send\_tasks.go

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\transfer\sender\send\_tasks.go

// Judge定时任务, 将 Judge发送缓存中的数据 通过rpc连接池 发送到Judge

forward2JudgeTask

//调用远程的方法

JudgeConnPools.Call(addr, "Judge.Send", judgeItems, resp)

### Store包history.go

history.go中PushFrontAndMaintain具体发送

linkedlist.go 中PushFrontAndMaintain决定是否发送报警

needJudge := linkedList.PushFrontAndMaintain(val, maxCount)

**if** needJudge {

    Judge(linkedList, val, now)

}

初始化HistoryBigMap

**var** HistoryBigMap = make(**map**[string]\*JudgeItemMap)

### store\judge.go Store包judge.go

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\github.com\open-falcon\judge\store\judge.go

具体组装发送Event

sendEvent

// send to redis

redisKey := fmt.Sprintf(g.Config().Alarm.QueuePattern, event.Priority())

rc := g.RedisConnPool.Get()

**defer** rc.Close()

rc.Do("LPUSH", redisKey, string(bs))

"queuePattern"**:** "event:p%v"**,**

# 同步更新judge中的strategies和expression

## Strategy.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\

src\judge\cron\strategy.go

### SyncStrategies

**func** SyncStrategies() {  
 duration := time.Duration(g.Config().Hbs.Interval) \* time.*Second* **for** {  
 syncStrategies()  
 syncExpression()  
 time.Sleep(duration)  
 }  
}

### syncStrategies

通过rpc调用Hbs.GetStrategies

**func** syncStrategies() {  
 **var** strategiesResponse model.StrategiesResponse  
 err := g.HbsClient.Call("Hbs.GetStrategies", model.NullRpcRequest{}, &strategiesResponse)  
 **if** err != nil {  
 log.Println("[ERROR] Hbs.GetStrategies:", err)  
 **return** }  
  
 rebuildStrategyMap(&strategiesResponse)  
}

### rebuildStrategyMap

g.StrategyMap.ReInit(m)重新初始化

**func** rebuildStrategyMap(strategiesResponse \*model.StrategiesResponse) {  
 // endpoint:metric => [strategy1, strategy2 ...]  
 m := make(**map**[string][]model.Strategy)  
 **for** \_, hs := **range** strategiesResponse.HostStrategies {  
 hostname := hs.Hostname  
 **if** g.Config().Debug && hostname == g.Config().DebugHost {  
 log.Println(hostname, "strategies:")  
 bs, \_ := json.Marshal(hs.Strategies)  
 fmt.Println(string(bs))  
 }  
 **for** \_, strategy := **range** hs.Strategies {  
 key := fmt.Sprintf("%s/%s", hostname, strategy.Metric)  
 **if** \_, exists := m[key]; exists {  
 m[key] = append(m[key], strategy)  
 } **else** {  
 m[key] = []model.Strategy{strategy}  
 }  
 }  
 }  
  
 g.StrategyMap.ReInit(m)  
}

## hbs.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\

src\hbs\rpc\hbs.go

### GetStrategies

**func** (t \*Hbs) GetStrategies(req model.NullRpcRequest, reply \*model.StrategiesResponse) error {  
 reply.HostStrategies = []\*model.HostStrategy{}  
 // 一个机器ID对应多个模板ID  
 hidTids := cache.HostTemplateIds.GetMap()  
 sz := len(hidTids)  
 **if** sz == 0 {  
 **return** nil  
 }  
  
 // Judge需要的是hostname，此处要把HostId转换为hostname  
 // 查出的hosts，是不处于维护时间内的  
 hosts := cache.MonitoredHosts.Get()  
 **if** len(hosts) == 0 {  
 // 所有机器都处于维护状态，汗  
 **return** nil  
 }  
  
 tpls := cache.TemplateCache.GetMap()  
 **if** len(tpls) == 0 {  
 **return** nil  
 }  
  
 strategies := cache.Strategies.GetMap()  
 **if** len(strategies) == 0 {  
 **return** nil  
 }  
  
 // 做个索引，给一个tplId，可以很方便的找到对应了哪些Strategy  
 tpl2Strategies := Tpl2Strategies(strategies)  
  
 hostStrategies := make([]\*model.HostStrategy, 0, sz)  
 **for** hostId, tplIds := **range** hidTids {  
  
 h, exists := hosts[hostId]  
 **if** !exists {  
 **continue** }  
  
 // 计算当前host配置了哪些监控策略  
 ss := CalcInheritStrategies(tpls, tplIds, tpl2Strategies)  
 **if** len(ss) <= 0 {  
 **continue** }  
  
 hs := model.HostStrategy{  
 Hostname: h.Name,  
 Strategies: ss,  
 }  
  
 hostStrategies = append(hostStrategies, &hs)  
  
 }  
  
 reply.HostStrategies = hostStrategies  
 **return** nil  
}

# 判断进行告警

## receiver.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\

src\github.com\open-falcon\judge\rpc\receiver.go

### Send

Rpc包receiver.go

负责将transfer发送过来的请求发送到alarm

**func** (this \*Judge) Send(items []\*model.JudgeItem, resp \*model.SimpleRpcResponse) error {

    remain := g.Config().Remain

    // 把当前时间的计算放在最外层，是为了减少获取时间时的系统调用开销

    now := time.Now().Unix()

**for** \_, item := **range** items {

        pk := item.PrimaryKey()

        store.HistoryBigMap[pk[0:2]].PushFrontAndMaintain(pk, item, remain, now)

    }

**return** nil

}

common\model\judge.go 定义了PrimaryKey方法

//最终调用store包linkedlist.go的PushFrontAndMaintain方法

store.HistoryBigMap[pk[0:2]].PushFrontAndMaintain(pk, item, remain, now)

## history.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\

src\github.com\open-falcon\judge\store\history.go

### PushFrontAndMaintain

**func** (this \*JudgeItemMap) PushFrontAndMaintain(key string, val \*model.JudgeItem, maxCount int, now int64) {  
 **if** linkedList, exists := this.Get(key); exists {  
 needJudge := linkedList.PushFrontAndMaintain(val, maxCount)  
 **if** needJudge {  
 Judge(linkedList, val, now)  
 }  
 } **else** {  
 NL := list.New()  
 NL.PushFront(val)  
 safeList := &SafeLinkedList{L: NL}  
 this.Set(key, safeList)  
 Judge(safeList, val, now)  
 }  
}