# transfer\http\http.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\http\http.go

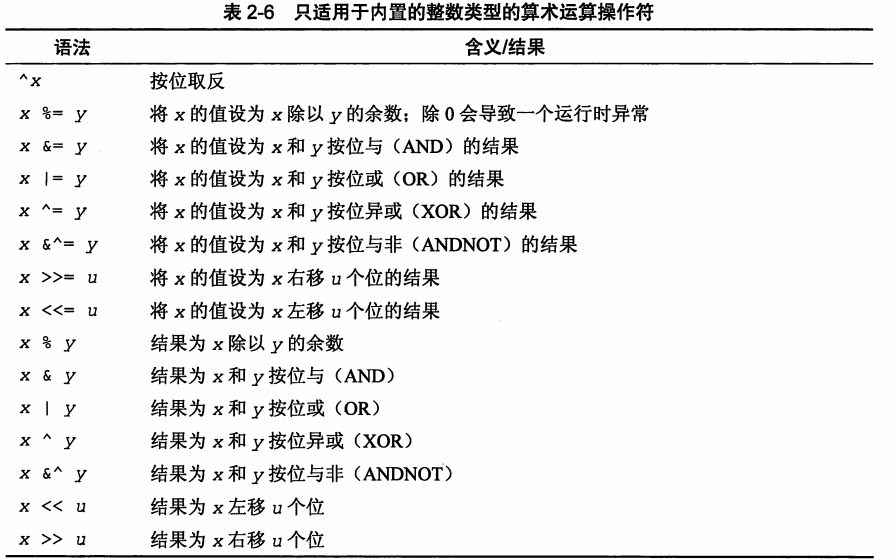
## startHttpServer

**func** startHttpServer() {  
 **if** !g.Config().Http.Enabled {  
 **return** }  
  
 addr := g.Config().Http.Listen  
 **if** addr == "" {  
 **return** }  
  
 configCommonRoutes()  
 configProcHttpRoutes()  
 configDebugHttpRoutes()  
 configApiHttpRoutes()  
  
 s := &http.Server{  
 Addr: addr,  
 MaxHeaderBytes: 1 << 30,  
 }  
  
 log.Println("http.startHttpServer ok, listening", addr)  
 log.Fatalln(s.ListenAndServe())  
}

MaxHeaderBytes

s := &http.Server{  
 Addr: addr,  
 MaxHeaderBytes: 1 << 30,  
 }

// MaxHeaderBytes controls the maximum number of bytes the  
// server will read parsing the request header's keys and  
// values, including the request line. It does not limit the  
// size of the request body.  
// If zero, DefaultMaxHeaderBytes is used.  
MaxHeaderBytes int



# transfer\http\debug\_http.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\

open-falcon\transfer\http\debug\_http.go

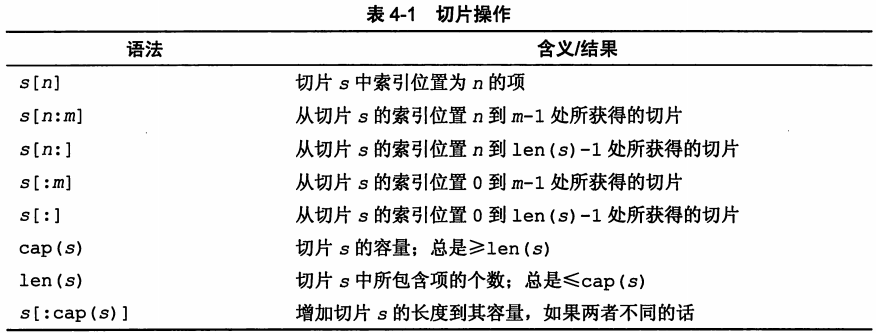
http://192.168.0.179:6060/debug/connpool/graph

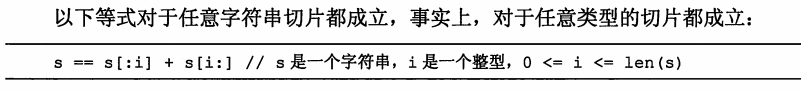
Name:127.0.0.1:6070,Cnt:6,active:6,all:6,free:6

**func** configDebugHttpRoutes() {  
 // conn pools  
 http.HandleFunc("/debug/connpool/", **func**(w http.ResponseWriter, r \*http.Request) {  
 urlParam := r.URL.Path[len("/debug/connpool/"):]  
 args := strings.Split(urlParam, "/")  
  
 argsLen := len(args)  
 **if** argsLen < 1 {  
 w.Write([]byte(fmt.Sprintf("bad args\n")))  
 **return** }  
  
 **var** result string  
 receiver := args[0]  
 **switch** receiver {  
 **case** "judge":  
 result = strings.Join(sender.JudgeConnPools.Proc(), "\n")  
 **case** "graph":  
 result = strings.Join(sender.GraphConnPools.Proc(), "\n")  
 **default**:  
 result = fmt.Sprintf("bad args, module not exist\n")  
 }  
 w.Write([]byte(result))  
 })  
}

## 字符串切片

urlParam := r.URL.Path[len("/debug/connpool/"):]





# transfer\receiver\rpc\rpc.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\receiver\rpc\rpc.go

## StartRpc

**func** StartRpc() {  
 **if** !g.Config().Rpc.Enabled {  
 **return** }  
  
 addr := g.Config().Rpc.Listen  
 tcpAddr, err := net.ResolveTCPAddr("tcp", addr)  
 **if** err != nil {  
 log.Fatalf("net.ResolveTCPAddr fail: %s", err)  
 }  
  
 listener, err := net.ListenTCP("tcp", tcpAddr)  
 **if** err != nil {  
 log.Fatalf("listen %s fail: %s", addr, err)  
 } **else** {  
 log.Println("rpc listening", addr)  
 }  
  
 server := rpc.NewServer()  
 server.Register(new(Transfer))  
  
 **for** {  
 conn, err := listener.Accept()  
 **if** err != nil {  
 log.Println("listener.Accept occur error:", err)  
 **continue** }  
 // go rpc.ServeConn(conn)  
 **go** server.ServeCodec(jsonrpc.NewServerCodec(conn))  
 }  
}

# transfer\receiver\rpc\rpc\_transfer.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\receiver\rpc\rpc\_transfer.go

## Update

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

agent\g\transfer.go

**func** updateMetrics(addr string, metrics []\*model.MetricValue, resp \*model.TransferResponse) bool {  
 TransferClientsLock.RLock()  
 **defer** TransferClientsLock.RUnlock()  
 err := TransferClients[addr].Call("Transfer.Update", metrics, resp)  
 **if** err != nil {  
 log.Println("call Transfer.Update fail", addr, err)  
 **return** false  
 }  
 **return** true  
}

接受rpc调用

**func** (t \*Transfer) Update(args []\*cmodel.MetricValue, reply \*cmodel.TransferResponse) error {  
 **return** RecvMetricValues(args, reply, "rpc")  
}

## RecvMetricValues

// process new metric values  
**func** RecvMetricValues(args []\*cmodel.MetricValue, reply \*cmodel.TransferResponse, from string) error {  
   
 **if** cfg.Graph.Enabled {  
 sender.Push2GraphSendQueue(items)  
 }  
  
 **if** cfg.Judge.Enabled {  
 sender.Push2JudgeSendQueue(items)  
 }  
  
 **if** cfg.Tsdb.Enabled {  
 sender.Push2TsdbSendQueue(items)  
 }  
  
 reply.Message = "ok"  
 reply.Total = len(args)  
 reply.Latency = (time.Now().UnixNano() - start.UnixNano()) / 1000000  
  
 **return** nil  
}

# transfer\sender\sender.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\sender\sender.go

## Push2TsdbSendQueue

// 将原始数据入到tsdb发送缓存队列  
**func** Push2TsdbSendQueue(items []\*cmodel.MetaData) {  
 **for** \_, item := **range** items {  
 tsdbItem := convert2TsdbItem(item)  
 isSuccess := TsdbQueue.PushFront(tsdbItem)  
  
 **if** !isSuccess {  
 proc.SendToTsdbDropCnt.Incr()  
 }  
 }  
}

### toolkits\container\list\safelist.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\

toolkits\container\list\safelist.go

#### SafeListLimited.PushFront

**func** (this \*SafeListLimited) PushFront(v **interface**{}) bool {  
 **if** this.SL.Len() >= this.maxSize {  
 **return** false  
 }  
  
 this.SL.PushFront(v)  
 **return** true  
}

#### SafeList.PushFront

**func** (this \*SafeList) PushFront(v **interface**{}) \*list.Element {  
 this.Lock()  
 e := this.L.PushFront(v)  
 this.Unlock()  
 **return** e  
}

### convert2TsdbItem

// 转化为tsdb格式  
**func** convert2TsdbItem(d \*cmodel.MetaData) \*cmodel.TsdbItem {  
 t := cmodel.TsdbItem{Tags: make(**map**[string]string)}  
  
 **for** k, v := **range** d.Tags {  
 t.Tags[k] = v  
 }  
 t.Tags["endpoint"] = d.Endpoint  
 t.Metric = d.Metric  
 t.Timestamp = d.Timestamp  
 t.Value = d.Value  
 **return** &t  
}

# transfer\sender\conn\_pool\conn\_pool\_manager.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\sender\conn\_pool\conn\_pool\_manager.go

## CreateSafeRpcConnPools

**func** CreateSafeRpcConnPools(maxConns, maxIdle, connTimeout, callTimeout int, cluster []string) \*SafeRpcConnPools {  
 cp := &SafeRpcConnPools{M: make(**map**[string]\*ConnPool), MaxConns: maxConns, MaxIdle: maxIdle,  
 ConnTimeout: connTimeout, CallTimeout: callTimeout}  
  
 ct := time.Duration(cp.ConnTimeout) \* time.*Millisecond* **for** \_, address := **range** cluster {  
 **if** \_, exist := cp.M[address]; exist {  
 **continue** }  
 cp.M[address] = createOnePool(address, address, ct, maxConns, maxIdle)  
 }  
  
 **return** cp  
}

## createOnePool

**func** createOnePool(name string, address string, connTimeout time.Duration, maxConns int, maxIdle int) \*ConnPool {  
 p := NewConnPool(name, address, maxConns, maxIdle)  
 p.New = **func**(connName string) (NConn, error) {  
 \_, err := net.ResolveTCPAddr("tcp", p.Address)  
 **if** err != nil {  
 //log.Println(p.Address, "format error", err)  
 **return** nil, err  
 }  
  
 conn, err := net.DialTimeout("tcp", p.Address, connTimeout)  
 **if** err != nil {  
 //log.Printf("new conn fail, addr %s, err %v", p.Address, err)  
 **return** nil, err  
 }  
  
 **return** RpcClient{cli: rpc.NewClient(conn), name: connName}, nil  
 }  
  
 **return** p  
}

net.ResolveTCPAddr

解析获得IP, port

// TCPAddr represents the address of a TCP end point.  
**type** TCPAddr **struct** {  
 IP IP  
 Port int  
 Zone string // IPv6 scoped addressing zone  
}

conn, err := net.DialTimeout("tcp", p.Address, connTimeout)

**func**[**DialTimeout**](https://github.com/golang/go/blob/master/src/net/dial.go?name=release#149)

func DialTimeout(network, address [string](http://godoc.org/builtin#string), timeout [time](http://godoc.org/time).[Duration](http://godoc.org/time#Duration)) ([Conn](http://studygolang.com/static/pkgdoc/pkg/net.htm#Conn), [error](http://godoc.org/builtin#error))

DialTimeout类似Dial但采用了超时。timeout参数如果必要可包含名称解析。

## RpcClient

// RpcCient, 要实现io.Closer接口  
**type** RpcClient **struct** {  
 cli \*rpc.Client  
 name string  
}

# transfer\sender\conn\_pool\conn\_pool.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\open-falcon\

transfer\sender\conn\_pool\conn\_pool.go

## Nconn接口

**type** NConn **interface** {  
 io.Closer  
 Name() string  
 Closed() bool  
}

# Transfer\sender\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)

## forward2JudgeTask

生成缓冲channel

nsema.NewSemaphore(concurrent)

将items传送到匿名函数中去

items := TsdbQueue.PopBackBy(batch)

**go func**(itemList []**interface**{}) {  
  
}(items)

// Tsdb定时任务, 将数据通过api发送到tsdb  
**func** forward2TsdbTask(concurrent int) {  
 batch := g.Config().Tsdb.Batch // 一次发送,最多batch条数据  
 retry := g.Config().Tsdb.MaxRetry  
 sema := nsema.NewSemaphore(concurrent)  
  
 **for** {  
 items := TsdbQueue.PopBackBy(batch)  
 **if** len(items) == 0 {  
 time.Sleep(*DefaultSendTaskSleepInterval*)  
 **continue** }  
 // 同步Call + 有限并发 进行发送  
 sema.Acquire()  
 **go func**(itemList []**interface**{}) {  
 **defer** sema.Release()  
  
 **var** tsdbBuffer bytes.Buffer  
 **for** i := 0; i < len(itemList); i++ {  
 tsdbItem := itemList[i].(\*cmodel.TsdbItem)  
 tsdbBuffer.WriteString(tsdbItem.TsdbString())  
 tsdbBuffer.WriteString("\n")  
 }  
  
 **var** err error  
 **for** i := 0; i < retry; i++ {  
 err = TsdbConnPoolHelper.Send(tsdbBuffer.Bytes())  
 **if** err == nil {  
 proc.SendToTsdbCnt.IncrBy(int64(len(itemList)))  
 **break** }  
 time.Sleep(100 \* time.*Millisecond*)  
 }  
  
 **if** err != nil {  
 proc.SendToTsdbFailCnt.IncrBy(int64(len(itemList)))  
 log.Println(err)  
 **return** }  
 }(items)  
 }  
}

### SafeListLimited.PopBackBy

**func** (this \*SafeListLimited) PopBackBy(max int) []**interface**{} {  
 **return** this.SL.PopBackBy(max)  
}

### SafeList.PopBackBy

**func** (this \*SafeList) PopBackBy(max int) []**interface**{} {  
 this.Lock()  
  
 count := this.len()  
 **if** count == 0 {  
 this.Unlock()  
 **return** []**interface**{}{}  
 }  
  
 **if** count > max {  
 count = max  
 }  
  
 items := make([]**interface**{}, 0, count)  
 **for** i := 0; i < count; i++ {  
 item := this.L.Remove(this.L.Back())  
 items = append(items, item)  
 }  
  
 this.Unlock()  
 **return** items  
}

从[SafeList.PushFront](#_SafeList.PushFront)中移除元素，并返回移除的元素

// Remove removes e from l if e is an element of list l.  
// It returns the element value e.Value.  
**func** (l \*List) Remove(e \*Element) **interface**{} {  
 **if** e.list == l {  
 // if e.list == l, l must have been initialized when e was inserted  
 // in l or l == nil (e is a zero Element) and l.remove will crash  
 l.remove(e)  
 }  
 **return** e.Value  
}

将移除的数据放到items中，并返回

items = append(items, item)

进行类型转换和类型断言

tsdbItem := itemList[i].(\*cmodel.TsdbItem)