## Rrdlite

一个轻量级的rrdtool工具包，线程安全，解除librrd依赖,只提供create,update,fetch,info

### 从rrd文件中获取数据FetchResult

rrd\_c.go

**func** Fetch(filename, cf string, start, end time.Time, step time.Duration) (FetchResult, error) {

rrd.go

**type** FetchResult **struct** {

    Filename string

    Cf string

    Start time.Time

    End time.Time

    Step time.Duration

    DsNames []string

    RowCnt int

    values []float64

}

### 时间序列数据获取

rrdtool /rrdtool.go/ fetch

**for** i, val := **range** values {

    ts := start\_ts + int64(i+1)\*int64(step\_s)

    d := &cmodel.RRDData{

        Timestamp: ts,

        Value: cmodel.JsonFloat(val),

    }

    ret[i] = d

}

{

"cf" : "AVERAGE",

"code" : 0,

"data" : [{

"name" : null,

"endpoint" : "192.168.1.136",

"counter" : "cpu.busy",

"data" : [{

"x" : 1471933440000,

"y" : 1.010101

}, {

"x" : 1471933500000,

"y" : 2.010050

}

### 创建rrd数据库

rrdtool /rrdtool.go/ reate

#测试例子

rrdlite/rrd\_test.go at master · yubo/rrdlite

https://github.com/yubo/rrdlite/blob/master/rrd\_test.go

**func** create(filename string, item \*cmodel.GraphItem) error {

    now := time.Now()

    start := now.Add(time.Duration(-24) \* time.Hour)

    step := uint(item.Step)

    c := rrdlite.NewCreator(filename, start, step)

    c.DS("metric", item.DsType, item.Heartbeat, item.Min, item.Max)

    // 设置各种归档策略

    // 1分钟一个点存 12小时

    c.RRA("AVERAGE", 0.5, 1, RRA1PointCnt)

    // 5m一个点存2d

    c.RRA("AVERAGE", 0.5, 5, RRA5PointCnt)

    c.RRA("MAX", 0.5, 5, RRA5PointCnt)

    c.RRA("MIN", 0.5, 5, RRA5PointCnt)

    // 20m一个点存7d

    c.RRA("AVERAGE", 0.5, 20, RRA20PointCnt)

    c.RRA("MAX", 0.5, 20, RRA20PointCnt)

    c.RRA("MIN", 0.5, 20, RRA20PointCnt)

    // 3小时一个点存3个月

    c.RRA("AVERAGE", 0.5, 180, RRA180PointCnt)

    c.RRA("MAX", 0.5, 180, RRA180PointCnt)

    c.RRA("MIN", 0.5, 180, RRA180PointCnt)

    // 12小时一个点存1year

    c.RRA("AVERAGE", 0.5, 720, RRA720PointCnt)

    c.RRA("MAX", 0.5, 720, RRA720PointCnt)

    c.RRA("MIN", 0.5, 720, RRA720PointCnt)

**return** c.Create(true)

}

### Rrd\_test.go 指定dbfile获取数据

根据endpoint和metric获取文件名

package rrdlite

import (

"fmt"

"github.com/open-falcon/rrdlite"

//"github.com/open-falcon/graph/index"

cutils "github.com/open-falcon/common/utils"

"runtime"

"sync"

"testing"

"time"

)

const (

dbfile = "/opt/open-falcon/data/6070/13/13feb2f6d65266c7c2273223d6561b47\_GAUGE\_60.rrd"

step = 1 \* 3600 \* 12

heartbeat = 2 \* step

b\_size = 100000

work\_size = 10

)

var now time.Time

var wg sync.WaitGroup

func init() {

now = time.Now()

runtime.GOMAXPROCS(runtime.NumCPU())

}

func TestEcho(t \* testing.T){

t.Fatal("test")

t.Error("test")

fmt.Println("test")

}

func TestAll(t \*testing.T) {

pk := cutils.Md5(fmt.Sprintf("%s/%s", "192.168.1.136","cpu.busy"))

// Info

//dbfile = "/opt/open-falcon/data/6070/13/13feb2f6d65266c7c2273223d6561b47\_GAUGE\_60.rrd"

var file = "/opt/open-falcon/data/6070/"+pk[0:2]+"/"+pk+"\_GAUGE\_60.rrd"

fmt.Printf("dbfile: %s\t;%s\n", dbfile)

inf, err := rrdlite.Info(file)

if err != nil {

t.Fatal(err)

}

for k, v := range inf {

fmt.Printf("%s (%T): %v\n", k, v, v)

}

// Fetch

end := time.Unix(int64(inf["last\_update"].(uint)), 0)

start := end.Add(-1 \* step \* time.Second)

fmt.Printf("Fetch Params:\n")

fmt.Printf("Start: %s\n", start)

fmt.Printf("End: %s\n", end)

fmt.Printf("Step: %s\n", step\*time.Second)

fetchRes, err := rrdlite.Fetch(dbfile, "AVERAGE", start, end, step\*time.Second)

if err != nil {

t.Fatal(err)

}

defer fetchRes.FreeValues()

fmt.Printf("FetchResult:\n")

fmt.Printf("Start: %s\n", fetchRes.Start)

fmt.Printf("End: %s\n", fetchRes.End)

fmt.Printf("Step: %s\n", fetchRes.Step)

for \_, dsName := range fetchRes.DsNames {

fmt.Printf("\t%s", dsName)

}

fmt.Printf("\n")

row := 0

for ti := fetchRes.Start.Add(fetchRes.Step); ti.Before(end) || ti.Equal(end); ti = ti.Add(fetchRes.Step) {

fmt.Printf("%s / %d", ti, ti.Unix())

for i := 0; i < len(fetchRes.DsNames); i++ {

v := fetchRes.ValueAt(i, row)

fmt.Printf("\t%e", v)

}

fmt.Printf("\n")

row++

}

/\*dsType, step, exists := index.GetTypeAndStep("192.168.1.136", "cpu.busy")

fmt.Printf("dsType: %s\n", dsType)

fmt.Printf("step: %s\n", step)

fmt.Printf("exists: %s\n", exists)\*/

//pk := cutils.Md5(fmt.Sprintf("%s/%s", "192.168.1.136","cpu.busy"))

}