RRDtool 的前期规划

RRDtool 的前期规划相对多一点，因为 RRDtool 很多东西需要自己设定。除了上述 MRTG 考虑的几点之外，我一般还考虑以下几点 ：

A）是一个 RRD 文件中包括多个监测对象（DS），还是分成多个 RRD 文件 ？RRDtool 提供了 tune 操作，可以增加监测对象或者删除 RRD 文

件中的某个对象，而且绘图时也可以指定要画的是那个对象，这点看个人喜欢而定。

B）如何统计取得的数据 ： MRTG 是固定的，5 分钟、20 分钟、2 小时、1 天。RRDtool 则可以自己设置

C）如何保存/统计这些数据 ： 这是和 MRTG 不同的地方。MRTG log 的建立和维护是自动的，RRDtool 的数据存放

则需要自己定义。但我们可以参照 MRTG 的方式：

每日统计图（5 分钟平均） ： 600 个，大约 2 天的时间

每周统计图（20 分钟平均） ： 600 个，大约 8 天的时间

每月统计图 （2 小时平均） ： 600 个，50 天的时间

每年统计图 （1 天平均） ：730 个， 2 年的时间

D 要以什么方式绘图 ： MRTG 只有曲线（LINE）和方块（AREA）两种；RRDtool 除了这两种外，还有一种是 STACK 方式。就是在前一个曲线或者方

块的基础上绘图图，而不是直接从 X 轴开始绘图。这样绘制出来的图比较清晰，不会出现交叉的现象，但此时 Y 轴的值等于当前对象的值加上前一

个绘图对象的值。例如前一个对象（cpu 的系统进程利用率）的值是 10，采用的是 AREA 方式绘图。当前对象（cpu 的用户级进程的利用率）是 5，

采用的是 STACK 方式，则“cpu 的用户级进程利用率”对应的 Y 轴刻度是 10+5=15；所以如果不加说明，别人可能会误解。

三）实际例子

A）搞清楚究竟想要监测什么对象 ： 监测本地主机的网络流量。包括 eth0 和 lo 接口的流量。

B）想要以什么方法来取得数据 ：sar 也可以统计网卡接口的流量。但这里我们用 SNMP ，访问 ifInOctets 和 ifOutOctets 。

假设脚本名称是 get_eth0_traffic.sh 和 get_lo_traffic.sh

C）每个对象的监测时间是多长时间一次 ： 5 分钟

D）是采用一个 RRD 文件还是多个 ： 2 个 RRD 文件，一个是 eth0.rrd，一个是 lo.rrd

E）[color=bl;ue]为每个监测对象起名 ： 分别是 eth0_in ,eth0_out ,lo_in ,lo_out

F）统计频率 ： 5 分钟、20 分钟、2 小时、1 天

G）如何保存统计数据 ： 600 个、600 个、600 个、730 个

H）要以什么方式绘图 ： 目前暂不考虑该问题。等到实际绘图时再体验。

注 ： 实际上我们可以把数据的插入、绘图一起做到 get_eth0_traffic.sh 和 get_lo_traffic.sh 中，但目前这两个脚本只是负责取数据并输出而已，

到最后我们再把这些功能合并到一起。

四）下面是脚本的内容

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# cat get_eth0_traffic.sh
#!/bin/bash

# 首先取得 eth0 接口的 ifIndex
index=$(snmpwalk -IR localhost RFC1213-MIB::ifDescr |grep eth0|cut -d '=' -f 1|cut -d '.' -f 2)

# 再通过 snmp 协议取得 ififInOctets 和 ifOutOctets 的值
# 由于在 /etc/snmp.conf 中配置了 defVersion 和 defCommunity ，所以 snmpget 命令不用指定这两个参数

eth0_in=$(snmpget -IR -Os localhost ifInOctets.${index}|cut -d ':' -f 2|tr -d '[:blank:]')
eth0_out=$(snmpget -IR -Os localhost ifOutOctets.${index}|cut -d ':' -f 2 |tr -d '[:blank:]')
echo $eth0_in
echo $eth0_out

[root@dns1 bob]#
```

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# cat get_lo_traffic.sh
#!/bin/bash
# 首先取得 eth0 接口的 ifIndex

index=$(snmpwalk -IR localhost RFC1213-MIB::ifDescr |grep lo|cut -d '=' -f 1|cut -d '.' -f 2)
lo_in=$(snmpget -IR -Os localhost ifInOctets.${index}|cut -d ':' -f 2|tr -d '[:blank:]')
lo_out=$(snmpget -IR -Os localhost ifOutOctets.${index}|cut -d ':' -f 2 |tr -d '[:blank:]')
echo $lo_in

echo $lo_out
[root@dns1 bob]#
```

再把这 2 个脚本放入 crontab 中，每 5 分钟执行一次

[Copy to clipboard] [-]

CODE:

```
*/5 * * * * /home/bob/get_eth0_traffic.sh
*/5 * * * * /home/bob/get_lo_traffic.sh
```

不过这样会有讨厌的邮件产生，也可以在脚本中用 while true 循环，配合 sleep 300 让脚本一直运行，而不是重复启动脚本。具体选择那样你自己决定。
当所有的准备工作都完成后，就可以开始考虑建库了。

建立 RRD 数据库

****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****

准备工作都做完了，脚本也写完了，就可以开始建库了。建库实际上就是建立后缀名为 .rrd 的 RRD 文件。

一）语法格式

[Copy to clipboard] [-]

CODE:

```
rrdtool create filename [--start|-b start time] [--step|-s step]
                        [DS:ds-name:DST:dst arguments]
                        [RRA:CF:cf arguments]
```

其中 filename 、DS 部分和 RRA 部分是必须的。其他两个参数可免。

二）参数解释

A）**<filename>** ：默认是以 .rrd 结尾，但也以随你设定。
B） **--step** ：就是 RRDtool“期望”每隔多长时间就收到一个值。和 MRTG 的 interval 同样含义。默认是 5 分钟。我们的脚本也应该是
每 5 分钟运行一次。
C） **--start** ：给出 RRDtool 的第一个记录的起始时间。RRDtool 不会接受任何采样时间小于或者等于指定时间的数据。也就是说 --start
指定了数据库最早的那个记录是从什么时候开始的。如果 update 操作中给出的时间在 --start 之前，则 RRDtool 拒绝接受。--satrt 选项也是

可选的。按照 我们在前一篇中的设定，则默认是当前时间减去 600*300 秒，也就是 50 个小时前。 如果你想指定--start 为 1 天前，可以用

[Copy to clipboard] [-]

CODE:

```
--start $(date -d '1 days aog' +%s)
```

注意，--start 选项的值必须是 timestamp 的格式。

D） **DS** ：DS 用于定义 Data Soure 。也就是用于存放脚本的结果的变量名(DSN)。

就是我们前面提到的 eth0_in ,eth0_out, lo_in , lo_out 。DSN 从 1-19 个字符，必须是 0-9,a-z,A-Z 。

E） **DST** ：DST 就是 Data Source Type 的意思。有 COUNTER、GUAGE、DERIVE、ABSOLUTE、COMPUTE 5 种。

由于网卡流量属于计数器型，所以这里应该为 COUNTER 。

F） **RRA** ：RRA 用于指定数据如何存放。我们可以把一个 RRA 看成一个表，各保存不同 interval 的统计结果

G）**PDP** ：Primary Data Point 。正常情况下每个 interval RRDtool 都会收到一个值；RRDtool 在收到脚本给来的值后 会计算出另外

一个值（例如平均值），这个 值就是 PDP ；这个值代表的一般是"xxx/秒"的含义。注意，该值不一定等于 RRDtool 收到的那个值。除非是

GAUGE ，可以看下面的例子就知道了

H） **CF** ：CF 就是 Consolidation Function 的缩写。也就是合并（统计）功能。有 AVERAGE、MAX、MIN、LAST 四种

分别表示对多个 PDP 进行取平均、取最大值、取最小值、取当前值四种类型。具体作用等到 update 操作时 再说。

I） CDP ：Consolidation Data Point 。RRDtool 使用多个 PDP 合并为（计算出）一个 CDP。也就是执行上面 的 CF 操作后的结果。这个值就是存入 RRA

的数据，绘图时使用的也是这些数据。

三）再说 DST

DST 的选择是十分重要的，如果选错了 DST ，即使你的脚本取的数据是对的，放入 RRDtool 后也是错误的，更不用提画出来的图是否有意义了。

如何选择 DST 看下面的描述 ：

A）COUNTER ：必须是递增的，除非是计数器溢出（overflows）。在这种情况下，RRDtool 会自动修改收到的值。例如网络接口流量、收到的

packets 数量都属于这一类型。

B）DERIVE：和 COUNTER 类似。但可以是递增，也可以递减，或者一会增加一会儿减少。

C）ABSOLUTE ：ABSOLUTE 比较特殊，它每次都假定前一个 interval 的值是 0，再计算平均值。

D）GAUGE ：GAGUE 和上面三种不同，它没有"平均"的概念，RRDtool 收到值之后字节存入 RRA 中

E）COMPUTE ：COMPUTE 比较特殊，它并不接受输入，它的定义是一个表达式，能够引用其他 DS 并自动计算出某个值。例如

[Copy to clipboard] [-]

CODE:

```
DS:eth0_bytes:COUNTER:600:0:U DS:eth0_bits:COMPUTE:bytes,8,*
```

则 eth0_bytes 每得到一个值，eth0_bits 会自动计算出它的值：将 eth0_bytes 的值乘以 8 。不过 COMPUTE 型的 DS 有个限制，只能应用

它所在的 RRD 的 DS ，不能引用其他 RRD 的 DS。 COMPUTE 型 DS 是新版本的 RRDtool 才有的，你也可以用 CDEF 来实现该功能。

F）AVERAGE 类型适合于看"平均"情况，例如一天的平均流量，。所以 AVERAGE 适用于需要知道 'xxx/秒' 这样的需求。但采用 AVERAGE 型时，你并不知道

在每个 CDP 中（假设 30 分钟平均，6 个 PDP 组成）之中，流量具体是如何变化的，什么时候高，什么时候低。这于需要用到别的统计类型了

G）MAXIMUM 、MINIMUM 不适用想知道"xxx/秒"这样的需求，而是适用于想知道某个对象在各个不同时刻的表现的需求，也就是着重点在于各个时间点。

也就是所谓的"趋势"了，还是上面的例子，如果采用 MAXIMUM 或者 MINIMUM 的 CF ，可以看出接口在每个 CDP 的周期内最高是达到多少，最低又是多少，如果是 AVERAGE 的话，有可能前 5 个 PDP 都很均匀，但最后一个 PDP 的值发生很大的突变。这时候如果用 AVERAGE 可能是看不出来的，因为突变的部分被平均分配到整个时间段内了,所以看不出突变这一现象;但如果用 MAXIMUM 就可以清楚的知道在该 CDP 的周期内，曾经有达到某个值的时候。所以用
MAXIMUM 或者 MINIMUM 就可以知道某个对象在某个时间段内最大达到多少，最低低到什么程度。

例如要看某个接口在一天内有没有超过 50Mb 流量的时候就要用 MAXIMUM.例如要看磁盘空间的空闲率在一天内有没有低于 20% 的时候就要用 MINIMUM

H）LAST 类型适用于 "累计"的概念，例如从 xxx 时候到目前共累计 xxxx 这样的需求。例如邮件数量，可以用 LAST 来表示 30 分钟内总共收到多少个邮件，同样 LAST 也没有平均的概念，也就是说不适用于 'xxx/秒' 这样的需求，例如你不能说平均每秒钟多少封邮件这样的说法；同样也不适用于看每个周期内的变化，例如 30 分钟内共收到 100 封邮件，分别是 ：第一个 5 分钟 20 封，第二个 5 分钟 30 封，第三个 5 分钟没有，第 4 个 5 分钟 10 封，第 5 个 5 分钟也没有，第 6 个 5 分钟 40 封。如果用 MAXIMUM 或者 MINIMUM 就不知道在 30 分钟内共收到 100 封邮件，而是得出 30 和 0。所以 LAST 适用于每隔一段时间被观察 对象就会复位的情况。例如每 30 分钟就收一次邮件，邮件数量就是 LAST 值，同时现有的新邮件数量就被清零；到下一个 30 分钟再收一次邮件，又得到一个 30 分钟的 LAST 值。这样就可以得得出"距离上一次操作后到目前为止共 xxx"的需求。（例如距离上一次收取邮件后又共收到 100 封新邮件）

四）DST 实例说明

这样说可能还是比较模糊，可以看下面的例子，体会一下什么是 DST 和 PDP ：

QUOTE:

```
Values = 300, 600, 900, 1200        # 假设 RRDtool 收到 4 个值，分别是 300，600，900，1200

Step = 300 seconds                  # step 为 300

COUNTER = 1,1, 1,1                      # （300-0）/300，（600-300）/300，（900-600）/300，
（1200-900）/300 ，所以结果为 1，1，1，1
```

```
DERIVE = 1,1,1,1                    # 同上

ABSOLUTE = 1,2,3,4                  # (300-0)/300,(600-0)/300 , (900-0)/300, (1200-0)/300，
所以结果为 1，2，3，4

GAUGE = 300,600,900,1200            # 300 , 600 ,900 ,1200 不做运算，直接存入数据库
```

所以第一行的 values 并不是 PDP ，后面 4 行才是 PDP

五）开始建库

[Copy to clipboard] [-]

CODE:

```
[root@dns1 root]# rrdtool create eth0.rrd \
> --start $(date –d '1 days ago' +%s) \
> --step 300 \
> DS:eth0_in:COUNTER:600:0:12500000 \        # 600 是 heartbeat；0 是最小值；12500000 表示最大值；
> DS:eth0_out:COUNER:600:0:12500000 \        # 如果没有最小值/最大值，可以用 U 代替，例如 U:U
> RRA:AVERAGE:0.5:1:600 \          # 1 表示对 1 个 PDP 取平均。实际上就等于 PDP 的值
> RRA:AVERAGE:0.5:4:600 \          # 4 表示每 4 个 PDP 合成为一个 CDP，也就是 20 分钟。方法是对 4 个 PDP
取平均，
> RRA:AVERAGE:0.5:24:600 \ # 同上，但改为 24 个，也就是 24*5=120 分钟=2 小时。
> RRA:AVERAGE:0.5:288:730          # 同上，但改为 288 个，也就是 288*5=1440 分钟=1 天
[root@dns1 root]#
```

注：上面第 2-4 个 RRA 的记录数实际上应该是 700，775，790，而不是 600，600，730。
600 samples of 5 minutes (2 days and 2 hours)= 180000 秒 （2.08 天）
700 samples of 30 minutes (2 days and 2 hours, plus 12.5 days)= 1260000 秒 （14.58 天 ， 2 周）
775 samples of 2 hours (above + 50 days) = 5580000 秒 （64.58 天，2 个月）
797 samples of 1 day (above + 732 days, rounded up to 797) = 68860800 秒（2 年）
可以看出每个 RRA 都存储了相应单位 2 倍时间的数据，例如每天的 RRA 存储 2 天的数据，每周的 RRA 存储 2 周的数据，每月的 RRA 存储 2 个月的数据，每年的 RRA 存储 2 年的数据

检查一下结果

[Copy to clipboard] [-]

CODE:

```
root@dns1 bob]# ll -h eth0.rrd
-rw-r--r--    1 root    root          41K 11 月  19 23:16 eth0.rrd
[root@dns1 bob]#
```

有的人可能会问，上面有两个 DS，那 RRA 中究竟存的是那个 DS 的数据呢？实际上，这些 RRA 是共用的，你只需建立一个 RRA，它就可以用于全部的 DS 。

所以在定义 RRA 时不需要指定是给那个 DS 用的。

六）什么是 CF

以第 2 个 RRA 和 4，2，1，3 这 4 个 PDP 为例
AVERAGE ：则结果为 (4+2+1+3)/4=2.5
MAX ：结果为 4 个数中的最大值 4
MIN ：结果为 4 个数中的最小值 1
LAST ：结果为 4 个数中的最后一个 3
同理，第三个 RRA 和第 4 个 RRA 则是每 24 个 PDP、每 288 个 PDP 合成为 1 个 CDP

七）解释度（Resolution）

这里要提到一个 Resolution 的概念，在官方文档中多处提到 resolution 一词。Resolution 究竟是什么？Resolutino 有什么用？

举个例子，如果我们要绘制 1 小时的数据，也就是 60 分钟，那么我们可以从第一个 RRA 中取出 12 个 CDP 来绘图；也可以从第 2 个 RRA

中取出 2 个 CDP 来绘图。到底 RRDtool 会使用那个呢？

让我们看一下 RRA 的定义 ：RRA:AVERAGE:0.5:4:600 。

Resolution 就等于 4 * step = 4 * 300 = 1200 ，也就是说 ，resolution 是每个 CDP 所代表的时间范围，或者说 RRA 中每个 CDP（记录）

之间的时间间隔。所以第一个 RRA 的 resolution 是 1* step=300，第 2 是 1200，第三个是 24*300=7200，第 4 个 RRA 是 86400 。

默认情况下，RRDtool 会自动挑选合适的 resolution 的那个 RRA 的数据来绘图。我们大可不必关心它。但如果自己想取特定 RRA 的数据，就需要用到它了。

关于 Resolution 我们还会在 fetch 和 graph 操作中提到它。

八）xff 字段

细心的朋友可能会发现，在 RRA 的定义中有一个数值，固定是 0.5 ，这个到底是什么东东呢？
这个称为 xff 字段，是 **xfile factor** 的缩写。让我们来看它的定义 ：

QUOTE:
The xfiles factor defines what part of a consolidation interval may be made up from *UNKNOWN* data while

the consolidated value is still regarded as known. It is given as the ratio of allowed *UNKNOWN* PDPs to

the number of PDPs in the interval. Thus, it ranges from 0 to 1 (exclusive)

这个看起来有点头晕，我们举个简单的例子 ：例如

[Copy to clipboard] [-]

CODE:
```
RRA:AVERAGE:0.5:24:600
```

这个 RRA 中，每 24 个 PDP（共两小时）就合成为一个 CDP，如果这 24 个 PDP 中有部分值是 UNKNOWN（原因可以很多），例如 1 个，那么这个 CDP

合成的结果如何呢？是否就为 UNKNOWN 呢？

不是的，这要看 xff 字段而定。Xff 字段实际就是一个比例值。0.5 表示一个 CDP 中的所有 PDP 如果超过一半的值为 UNKNOWN ，则该 CDP 的值就被标为

UNKNOWN。也就是说，如果 24 个 PDP 中有 12 个或者超过 12 个 PDP 的值是 UNKNOWN ，则该 CPD 就无法合成，或者合成的结果为 UNKNOWN；

如果是 11 个 PDP 的值为 UNKNOWN ，则该 CDP 的值等于剩下 13 个 PDP 的平均值。

如果一个 CDP 是有 2 个 PDP 组成，xff 为 0.5 ，那么只要有一个 PDP 为 UNKNOWN ，则该 PDP 所对应的 CDP 的值就是 UNKNOWN 了

获取 RRD 文件的信息

****
****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****
****

一）前言

可能你已经颇不亟待的想知道如何往 RRD 文件插入数据、如何绘图了吧？hoho，先别急，在你做这些事情之前，最好先思考以下几个问题 ：

A）如果给你一个 RRD 文件，你能知道它的第一次/最后一次 update 的时间是在什么时候吗？
B）如果你很久之前建立了一个 RRD 文件，现在因为工作原因需要对该 RRD 文件进行一些修改。不过遗憾的是，你已经不记得得当初设置的具体选项和参数了， 这时候该怎么办呢？

这两个问题就对应今天要讲的两个操作 ：first/last 、info 。first 就是用于查看该 RRD 文件中某个 RRA 的第一个数据是在什么时候插入的（或者说第一次更新）；
last 就是查看该 RRD 文件的最近一次更新；
info 就是查看 rrd 文件的结构信息。
下面就以实际例子来看一下该怎么用这三个命令 ：

二）如何查询一个 RRD 文件的结构信息

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool info eth0.rrd          （由于输出信息较多，截取了一部分）
filename = "eth0.rrd"
rrd_version = "0003"
step = 300                                  # RRDtool 希望每 5 分钟收到一个数据
last_update = 1163862985                # 这是最近一次更新的 timestamp 。可以用 date 转换为具体的时间
ds[eth0_in].type = "COUNTER"                # 有一个名为 eth0_in 的 DS，DST 是 COUNTER
ds[eth0_in].minimal_heartbeat = 600       # hearbeat 时间是 600 秒
ds[eth0_in].min = 0.0000000000e+00        # eth0_in 的最小值是 0 （bytes）
```

```
ds[eth0_in].max = 1.2500000000e+07          # eth0_in 的最大值是 1250000000 （bytes）
ds[eth0_in].last_ds = "UNKN"
ds[eth0_in].value = 0.0000000000e+00
ds[eth0_in].unknown_sec = 85
ds[eth0_out].type = "COUNTER"
ds[eth0_out].minimal_heartbeat = 600
ds[eth0_out].min = 0.0000000000e+00
ds[eth0_out].max = 1.2500000000e+07
ds[eth0_out].last_ds = "UNKN"
ds[eth0_out].value = 0.0000000000e+00
ds[eth0_out].unknown_sec = 85
rra[0].cf = "AVERAGE"                               # 第一个 RRA 的编号是 0，不是 1。
rra[0].rows = 600                                   # 共保存 600 个记录
rra[0].pdp_per_row = 1                      # 每个 CDP 由一个 PDP 统计得出
rra[0].xff = 5.0000000000e-01               # 只要当前 interval 的 PDP 为 unknown ，则该 CDP 的值也是
unknown
rra[0].cdp_prep[0].value = NaN
rra[0].cdp_prep[0].unknown_datapoints = 0
rra[0].cdp_prep[1].value = NaN
rra[0].cdp_prep[1].unknown_datapoints = 0
rra[1].cf = "AVERAGE"                               # 第二个 RRA 的编号是 1。同样也是 AVERAGE 型。
rra[1].rows = 600                                   # 也是保存 600 个记录
rra[1].pdp_per_row = 4                      # 每个 CDP 由 4 个 PDP 的求平均值得出
rra[1].xff = 5.0000000000e-01               # 每个 CDP 最多允许 2 个 PDP 为 unknown ，超过则该 CDP 为
unknown
rra[1].cdp_prep[0].value = NaN
rra[1].cdp_prep[0].unknown_datapoints = 3
rra[1].cdp_prep[1].value = NaN
rra[1].cdp_prep[1].unknown_datapoints = 3
[root@dns1 bob]#
```

由于信息太长，这里截取了后面 2 个 RRA 的信息。

三）第一次更新/最近一次更新

如果想知道最近一次更新发生在什么时候，除了可以用上面的 info 操作，还可以用 last 操作

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool last eth0.rrd
1163862985
[root@dns1 bob]#
```

如果转换成具体的时间就是 ：

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool last eth0.rrd |xargs -i date -d '1970-01-01 {} sec utc'
六 11 月 18 23:16:25 CST 2006
[root@dns1 bob]#
```

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool first eth0.rrd
1163683200
[root@dns1 bob]#
```

如果换成具体的时间就是 ：

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# [root@dns1 bob]# rrdtool first eth0.rrd |xargs -i date -d '1970-01-01 {} sec utc'
四 11 月 16 21:20:00 CST 2006
[root@dns1 bob]#
```

这三个命令的语法都非常简单，但并不可以因此小看它们的功能，尤其是 info 操作。日后如果需要对 RRD 文件进行调整，是经常需要用到的。

更新 RRD 文件

****

注 ： 该教程参考了如下内容 ：

A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html

B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool

http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool

作者 ： ailms <ailms{@}263{dot}net>

版本 ： v1

最后修改 ： 2006/11/17 17:35

****

一）前言

写了 N 多东东，总算到了 update 部分了。这里有必要比较一下 RRDtool 和 MRTG 在 update 方面的差别。

A）MRTG 可以通过 SNMP 协议直接访问 SNMP 对象，你只需要在 cfg 文件中的 Target 指定 OID ，MRTG 就可以自动替你取回数据。

例如 Target[as1_eth0]: 2:n7css@172.17.64.11:::::1 ，表示使用 SNMP v1 协议访问 172.17.64.11 主机上 index 为 2 的那个接口，默认是取 ifInOctets 和 ifOutOctets 这两个对象的值。RRDtool 则没有这个功能，只能你自己写脚本取数据。

B)MRTG 只支持 COUNTER 和 GAUGE 类型的 Target ；RRDtool 则还可以使用 DERIVE、ABSOLUTE、COMPUTE
C）由于上面的原因，MRTG 无法识别小数，负数。例如你给 MRTG 一个 -1 的值，它会解释为 1 ；这点可以通过 LOG 看出来。小数也不行。例如 .72 （bc 的输出）会被识别为 72 ，而不是 0.72。
D）MRTG 每次 update 每次运行只更新一次，或者说只插入一行记录。但 RRDtool 可以在一个 updat 操作中插入多个记录。
E）MRTG 一次要求 2 个值，RRDtool 则没有该方面的限制。
F）最大的一个区别是， MRTG 在收到一个值后会自动得出 timestamp ，并记录在 log 的第一个字段；而 RRDtool 是需要你给出一个 timestamp ， 表示该数据是什么时间采集的。

二）update 操作的语法格式

[Copy to clipboard] [-]

CODE:

```
rrdtool {update | updatev} filename
        [--template|-t ds-name[:ds-name]...]
        N|timestamp:value[:value...]
        at-timestamp@value[:value...]
        [timestamp:value[:value...] ...]
```

N 表示 now 的意思，会被 RRDtool 替换为当前的 timestamp ，也就是 date +%s 的结果。Timestamp 部分比较灵活，可以是数字形式，也可以是

AT-风格的时间（参考 at 命令的 manual），有点类似于自然语言的风格。

三）手工方式 update 数据库

我们先学习一下如何手工 update 数据库。Update 命令分成两部分 ：

A）时间戳 （timestamp）：表示该数据是在那个时间点采集的。Timestamp 的格式可以非常灵活 ：

数字形式 ：例如 1164419418 ，表示 “六 11 月 25 09:50:18 CST 2006”。通常用于手工插入的方式。

快捷方式 ：N 。字母 N 表示当前时间（Now）。如果是通过 crontab 的方式来运行 update 操作，这是最实用的方式。

AT-风格 ：所谓的 AT 风格的时间，可以参考 at 命令的 manual。例如 now、yesterday、now-1hour、now+5min 都是 AT 风格的时间。

要注意的是，如果使用 AT 风格的时间，则时间和第一个 value 之间使用 @ 分隔，而不是“:”

B）数值部分 ：一个 RRD 文件可以有多个 DS ，所以一次 update 可以给出多个 value 。多个 value 之间用“:”分隔。不过是不是所有的 DS 都必须

给出值呢？不一定。有时候你只想给出一部分 DS 的值，甚至有时候某些类型的 DS 是不允许赋值的，例如 COMPUTE 型的 DS 就是这样一个例子。

四）实际操作

实例一 ：一个错误的例子

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool update eth0.rrd 1163862980:1:2
ERROR: illegal attempt to update using time 0 when last update time is 1163862985 (minimum one second step)
[root@dns1 bob]#
```

咦？为什么出错了呢？是语法错误吗？不是的，RRDtool 提示最近一次更新是在 1163862985 这个时候。也就是说，update 给出的时间戳必须大于该值。

不能等于或者小于该时刻。因为数据一旦插入到 RRA 中，就不允许再次修改。所以 update 会检查给出的时间戳是否大于最后一次更新的时间戳，如果不是

则报错，不予执行。那如何才能知道最近一次更新的时间戳呢？还记得前一篇**“如何获取 RRD 文件的信息”**中介绍的 last 和 info 命令吗？对了！就是它们。

执行一下看看是什么结果 ？

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool last eth0.rrd
1163862985
[root@dns1 bob]#
```

last 操作显示的时间戳和上面的报错信息给出的值一样。这个时间是

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# date -d '1970-01-01 1163862985 sec utc'
六 11 月 18 23:16:25 CST 2006
[root@dns1 bob]#
```

总之如果要 update 数据库，则 update 操作给出的时间戳必须晚于最后一次 update 的时间。

实例二 ：还是一个错误的例子

我们挑 23:16 的下一个 5 分钟 23:20 作为时间戳吧。

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# date -d '2006-11-18 23:20' +%s
1163863200
[root@dns1 bob]#
```

所以 update 命令为 ：

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool update eth0.rrd 1163863200:1
ERROR: expected 2 data source readings (got 1) from 1163863200:1:...
[root@dns1 bob]#
```

还是不行？！！

仔细看错误信息，原来是我们给少了一个值。还记得 info 命令吗？这会儿它派上用场了。

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool info eth0.rrd
filename = "eth0.rrd"
…..(省略)
last_update = 1163862985
ds[eth0_in].type = "COUNTER"
ds[eth0_out].type = "COUNTER"
```

原来是 2 个 DS ，怪不得 RRDtool 会报错了

实例三 ：这次应该成功了吧？

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool update eth0.rrd 1163863200:1:2
[root@dns1 bob]#
```

这会倒是没有错误信息，那究竟数据是否别插入到 RRA 中了呢？

对于该问题，有两个方法，一个是通过 fetch 操作，从 RRA 中提取数据；但这个我们下一篇再讲。

还有一种方法就是用 updatev 操作来代替 update 。updatev 的 v 表示 verbose 的意思，现在就来看 updatev 的作用 ：

实例四 ：updatev 的好处

我们执行多个 update 操作

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool last eth0.rrd
1163864400
[root@dns1 bob]#
[root@dns1 bob]# rrdtool update eth0.rrd 1163864700:3000:4000
[root@dns1 bob]# rrdtool updatev eth0.rrd 1163865000:3300:4600
return_value = 0
[1163865000]RRA[AVERAGE][1]DS[eth0_in] = 1.0000000000e+00
[1163865000]RRA[AVERAGE][1]DS[eth0_out] = 2.0000000000e+00
```

可以看到 return value 是 0，这个 return value 你可以理解为 shell 编程中的 exit status 。updatev 用 0 表示成功，-1 表示失败。

不过我们插入的值明明是 3300 和 4600 ，为什么出来的结果是 1 和 2 呢？

这是因为 eth0_in 和 eth0_out 都是 COUNTER 型的 DS，所以 RRDtool 在收到 3300 和 4600 后，会作一个运算，就是

（3300-3000）/ step （300）= 1，（4600-4000）/step（300）=2 ，这就是 1 和 2 这两个值的来源了。还记得前面提到的 PDP 吗？

这两个值 （1 和 2）就是 PDP 了，而不是 3300 和 4600 。这点要搞清楚。

实例五 ：另外一个 updatev 的例子

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]#
[root@dns1 bob]# rrdtool updatev eth0.rrd 1163865300:3300:4600
return_value = 0
[1163865300]RRA[AVERAGE][1]DS[eth0_in] = 0.0000000000e+00
[1163865300]RRA[AVERAGE][1]DS[eth0_out] = 0.0000000000e+00
[root@dns1 bob]#
```

在 1163865300 这个时刻我们给出的值和上次一样，所以 eth0_in 和 eth0_out 的 PDP 都为 0

搞清楚了 PDP 的概念，现在我们来看什么是 CDP ，以及 CDP 是如何计算的

实例六 ：通过 updatev 掌握 CF 的概念

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool updatev eth0.rrd 1163865600:4000:5000
return_value = 0
[1163865600]RRA[AVERAGE][1]DS[eth0_in] = 2.3333333333e+00
[1163865600]RRA[AVERAGE][1]DS[eth0_out] = 1.3333333333e+00
[1163865600]RRA[AVERAGE][4]DS[eth0_in] = 1.6666666667e+00
[1163865600]RRA[AVERAGE][4]DS[eth0_out] = 1.6666666667e+00
[1163865600]RRA[AVERAGE][24]DS[eth0_in] = NaN
[1163865600]RRA[AVERAGE][24]DS[eth0_out] = NaN
[root@dns1 bob]#
```

这次的输出和上次又不一样了。这次 update 操作影响到几个 RRA ，看到 [] 中的 1，4，24 了吗？它们就是代表不同的 RRA 中每个 CDP 所包含的 PDP 数量。

1 就是 1 个 CDP 包含 1 个 PDP，4 就是 一个 CDP 包含 4 个 PDP（20 分钟）、24 就是 一个 CDP 包含 24 个 PDP（2 小时）。

不过为什么没有 288 呢？eth0.rrd 的第 4 个 RRA 不是规定每 288 个 PDP 合并为一个 CDP 吗？

因为这个时候还轮不到它出场。1163865600 / 7200 = 161648 , 也就是说刚好 1163865600 是在 7200 的某个周期上（161648）。

但 1163865600 / 86400 ≈ 13470.66 ，说明 1163865600 还不到 86400 的周期。

必须等到 13471* 86400=1163894400 才会出现 [288] 的 CDP，那这个时间戳代表的时间是什么时候呢？看下面的 date 命令就知道了 ：

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# date -d '1970-01-01 1163894400 sec utc'
日 11 月 19 08:00:00 CST 2006
[root@dns1 bob]#
[root@dns1 root]# date -d '1970-01-01 1163865600 sec utc'
日 11 月 19 00:00:00 CST 2006
[root@dns1 root]#
```

这样不就是刚好相差 1 天的时间了吗？你可能会觉得很奇怪，为什么不是 00:00 而是 08:00 呢？

还记得 create 操作的语法吗？其中有一个 --start 参数吗？不记得了？没关系，那就得用 first 操作来重新找出来，

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool first eth0.rrd --rraindex 3
1100822400
[root@dns1 bob]#
[root@dns1 bob]# date -d '1970-01-01 1100822400 sec utc'
五 11 月 19 08:00:00 CST 2004
[root@dns1 bob]#
```

看到了吗？是 2004 年的 11 月 19 日早上 8 点正，距离 2006-11-19 刚好是 2 年，也就是 730 天,因为 eth0.rrd 的第 4 个 RRA

只保存 730 个记录。每个记录时间上相差 1 天。也就是第一个记录是 2004/11/19 8:00 ,第二个记录是 2004/11/20 8:00 ，

第三个记录代表 2004/11/21 8:00 ，依次类推。所以离 1163865600 最近的下一个记录是发生在 2006/11/19 8:00 。

所以严格意义上来说，RRDtool 中的一天并不一定是从 0:00 开始的，但可以保证的就是两个记录之间肯定相差 86400 秒（1 天）。

四）自动更新数据库

其实这些都只不过是手工 update 时需要注意的一些地方，如果是自动更新数据库，时间戳方面就交给 RRDtool 去处理吧，我们不用操心了。

前面我们已经写好了一个脚本，现在就用它来更新

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# cat get_eth0_traffic.sh
#!/bin/bash

# 首先取得 eth0 接口的 ifIndex

index=$(snmpwalk -IR localhost RFC1213-MIB::ifDescr |grep eth0|cut -d '=' -f 1|cut -d '.' -f 2)

# 再通过 snmp 协议取得 ififInOctets 和 ifOutOctets 的值

# 由于在 /etc/snmp.conf 中配置了 defVersion 和 defCommunity ，所以 snmpget 命令不用指定这两个参数

eth0_in=$(snmpget -IR -Os localhost ifInOctets.${index}|cut -d ':' -f 2|tr -d '[:blank:]')

eth0_out=$(snmpget -IR -Os localhost ifOutOctets.${index}|cut -d ':' -f 2 |tr -d '[:blank:]')

echo ${eth0_in}

echo ${eth0_out}

# 需要我要用这些数据来更新 eth0.rrd ，注意 update 时的 timestamp 我们用的是 N

/usr/local/rrdtool-1.2.14/bin/rrdtool updatev /home/bob/eth0.rrd N:${eth0_in}:${eth0_out}

[root@dns1 bob]#
```

五）接下来是什么呢？

有了数据，下面该学什么了呢？是绘图吗？

不是！^_^!! （估计有人快疯了吧）

在绘图之前，你有没有想过 RRDtool 在绘图时如何取数据的呢？

例如我想画 2 小时内的数据，那么我们有 4 个 RRA ，其 resolution 分别是 300，1200，7200，86400

（还记得什么是 resolution 吗？就是每个 RRA 中两个 CDP 相隔的时间）。是从第一个 RRA 取出 7200/300=24 个记录，

还是从第二个 RRA 取出 7200/1200=6 个记录呢？或者是从第三个 RRA 中取出 1 个记录就可以呢？

这些问题我们就留待下一篇再学习吧。这里给大家留几个问题 ：

QUOTE:

A） 如果 eth0.rrd 在 5 分钟内收到不止 1 个更新，结果会怎样？提示：用 updatev 就可以看出来了

B） 如果过了 eth0.rrd 在 5 分钟内没有收到脚本返回的值，是否立即就用 UNKNOWN 作为 PDP 的值？

C） 参考上面的例子，搞清楚 heartbeat 的含义

D） 在搞清楚 heartbeat 后，再想一下 heartbeat 和 step 之间的关系。

从 RRD 文件中提取数据

****
****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****
****

一）前言

RRD 是 Round Robin Database 的意思，那么是否可以象普通的数据库进行查询操作呢？

答案是可以的。fetch 就是用来做这种事情的工具。当然 fetch 不能和 select 语句相比，它只是根据用户指定的时间，

从合适的 RRA 中取出数据，并加以格式化。不过和 MRTG 相比，已经好很多了，至少你不用取看该死的 log 文件。

实际上，fetch 操作其实可以不学，因为 RRDtool 会自动帮你选好数据。但你如何确定 RRDtool 取的数据就是你想要的呢？

或者说你如何证明 RRDtool 绘制出来的图是正确的呢？

废话少说，下面开始正文

二）fetch 操作的语法

[Copy to clipboard] [-]

CODE:

```
rrdtool fetch filename CF [--resolution|-r resolution] [--start|-s start] [--end|-e end]
```

其中 --start、--end、-r 都是可选的 。RRDtool 默认的 --end 是 now ，--start 是 end-1day ，也就是 1 天前。

CF 可以是 AVERAGE、MAX、MIN、LAST ，当然必须建库时有该 CF 类型的 RRA 才可以查，否则会报错。

三）fetch 如何取数据

在确定了时间范围后，RRDtool 会从多个 RRA 中挑选最佳的那个 RRA 的数据。至于什么是"最佳"，则从两个方面考虑 ：

A）第一是该 RRA 的数据要尽可能的覆盖所请求的时间范围。如何计算一个 RRA 的覆盖时间呢？以 eth0.rrd 的第一个 RRA 为例，

有 600 个记录，每个记录相隔 300 秒，则总的时间覆盖范围是 180000 秒≈2 天，所以如果 --start 和 --end 规定的时间范围

大于 2 天，则 RRDtool 不会从该 RRA 中取数据。

B）第二是 resolution 的要求。还是上面的例子，如果是要画 3 天的数据，从时间覆盖范围上来讲，第 2、3、4 个 RRA 都符合要求。

那究竟挑选那个 RRA 的数据呢？如果 fetch 中有指定 -r 选项，则挑选 resolution 等于 -r 指定的值那个 RRA 的数据。如果没有

-r 选项，则从第一个合适的 RRA 中取数据。

C）fetch 如果不加 --start、--end、-r ，则默认输出 resolution 最小的那个 RRA 的数据。就像下面的例子 1 一样。

四）实际例子

实例一 ：默认情况

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE |more
                         eth0_in             eth0_out

1164467700: 1.1337243905e+01 9.6323712631e-02
1164468000: 1.7896453039e+01 0.0000000000e+00
1164468300: 1.8469136234e+01 1.2215723119e+00
。。。。（中间省略）
1164553800: 6.9634610564e+01 4.9644415243e+01
1164554100: nan nan
```

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# date                              （当前时间）
日 11 月 26 23:11:12 CST 2006

[root@dns1 bob]# date -d '1970-1-1 1164554100 sec utc'              (最后一个记录的时间)
日 11 月 26 23:15:00 CST 2006
[root@dns1 bob]#

[root@dns1 bob]# date -d '1970-1-1 1164467700 sec utc'              （第一个记录的时间）
六 11 月 25 23:15:00 CST 2006
[root@dns1 bob]#
```

fetch 输出的第一列是 timestamp ，表示后面的数据是在什么时间收到的。":" 后面就是 DS 的值。fetch 不能指定只取那个 DS 的数据，

只能一次性取出全部 DS 的值。可以看到，eth0.rrd 有两个 DS ：eth0_in 和 eth0_out ，每个 DS 的值用 空格进行分隔，一律采用科学记数法的格式。

如果 fetch 不指定 --start 和 --end ，则默认取从当前时刻算起，往前 1 天的数据（289 个记录）。因为现在是 23:11 ，还不到 23:15,所以最后一个记

录的值是 NaN（Not a Number），也就是 UNKNOWN 的意思。可以看到，两个记录之间的时间间隔是 300。

实例二：使用 --start 和 --end 指定时间范围

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE --start 1164467700 --end 1164553800 |more
                     eth0_in             eth0_out

1164468000: 1.7896453039e+01 0.0000000000e+00
1164468300: 1.8469136234e+01 1.2215723119e+00
1164468600: 1.5988336199e+01 1.4417769382e-01
。。。。。（中间省略很多）
1164553800: 6.9634610564e+01 4.9644415243e+01
1164554100: 1.7481962958e+02 2.3086574912e+02
[root@dns1 bob]#
```

可以看到第一个记录和最后一个记录都比 --start 和 --end 晚了 300 秒。

实例三 ：使用 AT 风格的时间

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE --start end-1day --end 1164553800 |more
                     eth0_in             eth0_out

1164467700: 1.1337243905e+01 9.6323712631e-02
1164468000: 1.7896453039e+01 0.0000000000e+00
。。。。。。。（中间省略很多）
1164554100: 1.7481962958e+02 2.3086574912e+02
[root@dns1 bob]#
```

注意 --start 的值是 end-1day ，这就是 AT 风格的时间。end 就是 --end 中给出的 1164553800 。具体的时间范围是表示起始时间从 1164553800 往前 1 天 。

可以看到，现在第一个记录和实例二相比，提前了 300 秒。和例 2 中的 –start 一致了。所以能够用 AT 风格的时间的时候还是用 AT 风格的时间比较方便。

可以省去计算的麻烦，别人也比较容易看。

实例四 ：提取指定 resolution 的数据

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE --start 1164467700 --end start+1day -r 1200 |more
                     eth0_in             eth0_out

1164468000: 1.7899370295e+01 3.8782610300e+00
```

```
1164469200: 2.0828335735e+01 3.4166666667e-01
1164470400: 1.4581351504e+01 3.5000000000e-02
。。。。。。（中间省略很多）
1164554400: 9.4367707174e+01 9.4866775629e+01
[root@dns1 bob]
```

可以看到，现在记录两两之间的时间间隔变成了 1200 了。输出的行数为（86400/1200）+1=73（72+1）。

实例五：如果指定一个不存在的 resolution 呢？

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE --start 1164467700 --end start+1day -r 1000 |more
                     eth0_in             eth0_out

1164468000: 1.7899370295e+01 3.8782610300e+00
1164469200: 2.0828335735e+01 3.4166666667e-01
。。。。。。（中间省略很多）
1164554400: 9.4367707174e+01 9.4866775629e+01
[root@dns1 bob]#
```

我们指定的 resolution 是 1000 ，但并没有那个 RRA 的 resolution 为 1000，所以 RRDtool 挑选了第一个合适的 resolution ，

也就是 1200 的那个 RRA 的数据作为结果输出。**注意，RRDtool 只会挑选 resolution 比 –r 指定的值相等或者更高的 RRA ，不会挑**

选比 –r 指定的值小的 RRA 。例如在该例子中，RRDtool 就不会挑选 resolution=300 的第一个 RRA 。为什么呢？

各位可以自己根据第三部分“fetch 如何提取数据”的两个准则考虑一下

实例六 ：再来看一个 –r 的例子

如果我不想指定 –-start 或者 –end ，就想看 resolution 为 1200 呢？

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE -r 1200
                     eth0_in             eth0_out

1164470400: 1.4581351504e+01 3.5000000000e-02
1164471600: 1.9312781373e+01 3.5000000000e-02
```

```
。。。。。（中间省略很多）
1164555600: 8.5249300043e+01 7.0171152327e+01
1164556800: nan nan
[root@dns1 bob]#
```

咦？为什么还是使用记录的时间间隔还是 300 秒呢？我们不是指定了 -r 1200 吗？

老实说，这种方法 90% 以上的机率是不会成功吗？那应该怎么办呢？

实例七 ：正确使用 -r 的方式

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE -r 1200 --end $(($($(date +%s)/1200)*1200)) |more
                     eth0_in             eth0_out

1164470400: 1.4581351504e+01 3.5000000000e-02
1164471600: 1.9312781373e+01 3.5000000000e-02
1164472800: 1.7383358822e+01 3.5000000000e-02
1164474000: 1.4781054841e+01 3.3225406191e-01
。。。。。。（中间省略很多）
1164555600: 8.5249300043e+01 7.0171152327e+01
1164556800: nan nan
[root@dns1 bob]#
```

现在 resolution 已经变成 1200 的了。同理，如果想看 7200，86400 resolution 的 RRA，只要把 end 部分的 （ ）中的数字替换为相应的值就可以了。

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE -r 7200 --end $(($($(date +%s)/7200)*7200))

[root@dns1 bob]# rrdtool fetch eth0.rrd AVERAGE -r 86400 --end $(($($(date +%s)/86400)*86400))
```

实例八 ：关于 fetch 提取数据准则 1 的测试

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd --start now-3day AVERAGE |more
                     eth0_in             eth0_out

1164298800: nan nan
1164300000: nan nan
。。。。。。（中间省略很多）
1164556800: 6.4118014239e+01 1.8871145267e+01
```

```
1164558000: nan nan
[root@dns1 bob]#
```

和第一个例子不同，这次的 resolution 是 1200 了？为什么呢？因为我们指定的时间范围是 3 天，而第一个 RRA 只保存 2 天的数据多一点，所以 RRDtool 不会从

该 RRA 取数据，那么会从那个 RRA 取数据呢？由于我们没有指定 –r 选项，所以 RRDtool 选择 1200 的那个 RRA

实例九 ：关于 fetch 提取数据准则 2 的测试

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool fetch eth0.rrd --start now-3day AVERAGE -r 7200 |more
                     eth0_in             eth0_out

1164304800: nan nan
1164312000: nan nan
1164319200: nan nan
。。。。。。（中间省略很多）
1164549600: 5.1899602485e+01 4.3073128067e-01
1164556800: 7.9766222122e+01 4.0644151093e+01
1164564000: nan nan
[root@dns1 bob]#
```

现在 resolution 不再是 1200 ，而是指定的 7200 了。

因为虽然 resolution=1200 的 RRA 就可以满足 --start 和 --end 的需求，

但因为 -r 指定 resolution=7200 ，所以从第 3 个 RRA 中取数据

五）总结

从上面我们可以看出，fetch 实际上是非常复杂的一个命令，如果想要输出你所要的数据，就必须考虑好几个因素 ：

A）第一是具体想输出的时间范围?

B）第二是计算好 –-start 和 –end。建议至少给出一个，最好 2 个都给出

C）第三是如果有多个 RRA 符合条件，则使用 –r 指定具体的 resolution

使用 RRDtool 进行绘图（一）

****

****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****
****

一）前言

使用 RRDtool 我们最关心什么？当然是如何把数据画出来了。虽然前面谈了很多，但这些都是基础来的。掌握好了，可以让你在绘图时更加得心应手。

本来还有 RPN （反向波兰表达式）一节的，但考虑一下，觉得还是放到后面，先从基本的绘图讲起。

这一节的内容虽然很多，但基本都是实验性的内容，只要多试几次就可以了。

二、graph 操作的语法

[Copy to clipboard] [-]

CODE:

```
rrdtool graph filename [option ...]
  [data definition ...]
  [data calculation ...]
  [variable definition ...]
  [graph element ...]
  [print element ...]
```

其中的 data definiton、variable definition 、data calculation、分别是下面的格式

[Copy to clipboard] [-]

CODE:

```
DEF:<vname>=<rrdfile>:<ds-name>:<CF>[:step=<step>][:start=<time>][:end=<time>][:reduce=<CF>]
VDEF:vname=RPN expression
CDEF:vname=RPN expression
```

其中 filename 就是你想要生成的图片文件的名称，默认是 png 。你可以通过选项修改图片的类型，可以有 PNG、SVG、EPS、PDF 四种。

A）DEF 是 Definition （定义）的意思。定义什么呢？你要绘图，总要有数据源吧？DEF 就是告诉 RRDtool 从那个 RRD 中取出指定

DS（eth0_in、eth0_out）的某个类型的统计值（还可以指定 resolution、start、end），并把这一切放入到一个变量 <vname>中 。

你可能会感到奇怪，为什么还有一个 CF 字段？因为 RRA 有多种 CF 类型，有些 RRA 可能用来保存平均值、有些 RRA 可能用于统计最大值、

最小值等等。所以你必须同时指定使用什么 CF 类型的 RRA 的数据。至于 :start 和 :end 、:reduce 则用得比较少，最常用的就是 :step 了，

它可以让你控制 RRDtool 从那个 RRA 中取数据。

B）VDEF 是 Variable Definition （变量定义）的意思。定义什么呢？记得 MRTG 在图表的下面有一个称之为 Legend 的部分吗？

那里显示了 1 个或者 2 个 DS（MRTG 没有 DS 一说,这里是借用 RRDtool 的)的 "最大值"、"平均值"、"当前值"。这些值是如何

计算的呢？

RRDtool 中用 VDEF 来定义。这个变量专门存放某个 DS 某种类型的值，例如 eth0_in 的最大值、eht0_out 的当前值等。当你需要象

MRTG 一样输出数字报表（Legend） 时，就可以在 GPRINT 子句（sub clause）中调用它。

同样它也需要用一个变量来存放数值。要注意的是，旧版 的 RRDtool 中是用另外一种格式来达到相同的目的。新版的 RRDtool 则推荐使用

VDEF 语句。但在使用过程中，却发现 VDEF 的使用反而造成了困扰。例如你有 5 个 DS 要画，每个 DS 你都想输出最大值、最小值、平均值

、当前值。 如果使用 VDEF ，则需要 4 * 5 = 20 个 VDEF 语句，这会造成极大的困扰。具体例子可以看第十一节"数字报表"部分。

C)CDEF 是 Calculation Define 的意思。使用过 MRTG 的都会体会到一点，MRTG 的计算能力实在太差了。例如你有两个 Target ，

一个是 eth0_in ， 一个是 eth0_out,如果要把它们相加起来，再除以 8，得出 bytes 为单位的值，如何计算呢？或者说你只想看

eth0_in 中超过 10Mb/s 的那部分， 其余的不关心，又如何实现呢？因为 MRTG 不能对它从 log 取出来的数据进行修改，只能原

原本本的表现，所以很难满足我们的要求。而使用 CDEF ， 这一切都可以很容易的实现。CDEF 支持很多数学运算，甚至还支持简

单的逻辑运算 if-then-else ，可以解决前面提到的第 2 个问题：如何只绘制你所关 心的数据。不过这一切都需要熟悉 RPN 的语法，

所以我们放到下一节介绍，这一节就介绍如何把 RRDtool 中的数据以图表的方式显示出来。

三）选项分类

本部分我们按照官方文档的方式，把选项分成几大类，分为 ：

A）**Time range** ： 用于控制图表的 X 轴显示的起始/结束时间，也包括从 RRA 中提取指定时间的数据。

B）**Labels** ：用于控制 X/Y 轴的说明文字。

C）**Size** ：用于控制图片的大小。

D）**Limits** ：用于控制 Y 轴的上下限。

E）**Grid** ：用于控制 X/Y 轴的刻度如何显示。

F）**Miscellaneous** ：其他选项。例如显示中文、水印效果等等。

G）**Report** ：数字报表

需要说明的是，本篇当中并不是列出了所有选项的用法，只是列出较为常用的选项，如果想查看所有选项的的用法，可以到官方站点下载文档，

这里就不一一列出了，望各位见谅。

四）时间范围控制（**Time Range**）

[Copy to clipboard] [-]

CODE:

```
[-s|--start time] [-e|--end time] [-S|--step seconds]
```

既然要绘图，就应该有一个起始/结束的时间。Graph 操作中有 –s ， -e 选项。这两个选项即可以用于控制图表的 X 轴显示的时间范围，也可以用

于控制 RRDtool 从 RRA 中提取对应时间的数据。如果没有指定 --end ，默认为 now；如果没有指定 --start，则默认为 1 天前。如果两者都没有

指定，则图表默认显示从当前算起 1 天内的。数
回头看一下 DEF 中，也有 :start ,:end , :step ，这些和 --start、--end、--step 之间有什么关系呢？

让我们先看 :step 和 –step 之间的关系是如何的。

下面以 eth0.rrrd 为例，假设要绘制的时间范围 **range 等于 end -start[**

A）如果 0 < ragne < 180000 （第一个 RRA 的时间覆盖范围），则默认从第 1 个 RRA 中取数据 ：

如果 DEF 中给出的 :step > 300 ，例如 1000 ，则从 resolution= 1000 的或者第一个高于 1000 的 RRA 中挑选数据，由于 eth0.rrd 中没有

resolution = 1000 的 RRA，则 RRDtool 会从 resolution = 1200 的第 2 RRA 中取数据。

如果 DEF 中给出的 :step <= 300 ，例如 200 ，则 RRDtool 会忽略该设定，还是从第一个 RRA 中取数据。

B）如果 180000 < range < 720000 ，由于第一个 RRA 只能保存 2 天的数据，所以默认从第 2 个 RRA 中取数据 ：

如果 DEF 中给出的 :step > 1200 ，例如 1800，则 RRDtool 会从 resolution = 7200 的第 3 RRA 中取数据

如果 DEF 中给出 :step<= 1200，例如 300 ，则 RRDtool 会忽略，还是从第 2 个 RRA 中取数据

C）如果 720000 < range < 4320000 ，则默认从第三个 RRA 中取数据 ：

如果 DEF 中给出的 :step > 7200 ，例如 10000 ，则从第 4 个 RRA 中取数据

如果 DEF 中给出的 :step <= 7200 ， 例如 1200 ，则忽略该值，并还是从第 3 个 RRA 中取数据

D）如果 4320000 < range < 63072000 ，则默认从第 4 个 RRA 中取数据 ：

如果 DEF 中给出的 :step > 86400 ，则行为未知

如果 DEF 中给出的 :step <=86400 ，则从第 4 个 RRA 中取数据

E）-S 选项可以直接控制 RRDtool 如何挑选 RRA 。

例如 -S 1200 ，即使 DEF 中不加 :setp ，则 RRDtool 会从第 2 个 RRA 中取数据，即使 rang <

180000

如果 -S 和 :step 同时出现，则 :step 优先。

F）DEF 中的 :start 和 :end 可以覆盖 --start 和 --end 的值。

默认情况下，如果 DEF 中不加 :start 和 :end ，则等于 --start 和 –end

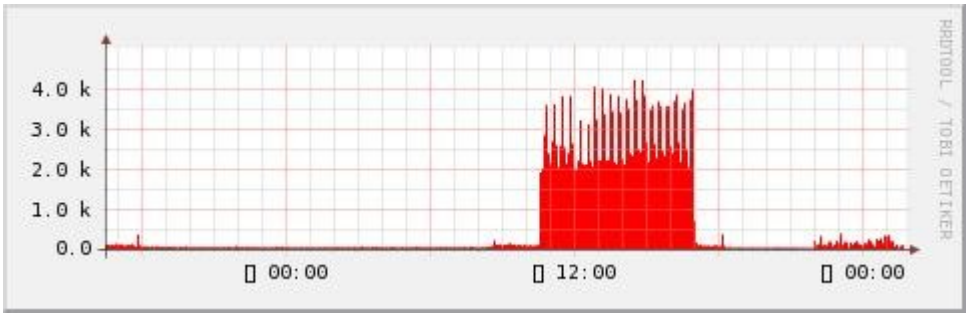
如果 DEF 中定义了 :start 和 :end ，则以 :start 和 :end 为准。

实例 1 ：使用 --start 指定 X 轴的起始时间

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-120000 \          # 表示起始时间是当前时间往前 120000 秒，也就是 33 个小时左右
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \          # 从 eth0.rrd 中取出 eth0_in 的数据，CF 类型为 AVERAGE
> AREA:value1#ff0000          # 用“方块”的形式来绘制 value1 ，注意这里是用 value1 ，不是用 eth0_in
481x154                       # 如果 RRDtool 有绘图方面的语句，则这里显示图片大小，否则为 0x0。
[root@dns1 bob]#
```



可以看到 X 轴的文字有些是乱码，不过不要紧，你可以临时已用 env LANG=C rrdtool xxxx 来解决该问题，或者在后面用

–n 来设定 RRDtool 使用中文字体，就不会出现这样的情况了

实例 2 ：使用 :step 从第 2 个 RRA 中取数据

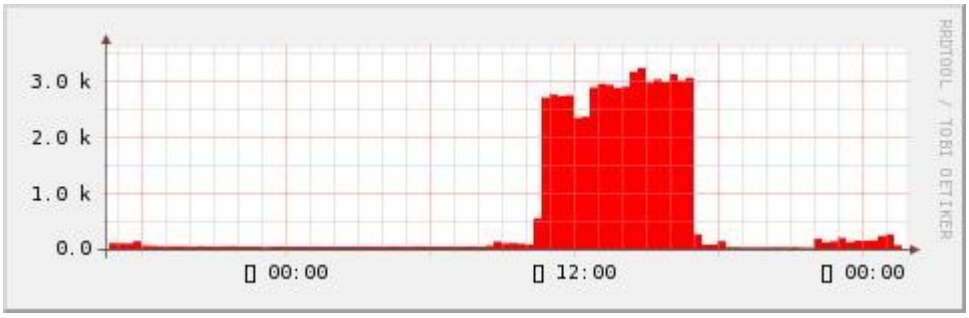
[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 2.png \
> --start now-120000 \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:step=1000 \          # :step 指定 resolution=1000
> AREA:value1#ff0000
```

```
481x154
[root@dns1 bob]#
```

这里是 :step=1000，但 RRDtool 会取 :step=1200 的 第 2 个 RRA 的数据来绘图,可以和上面的 1.png 比较，发现比较平滑。

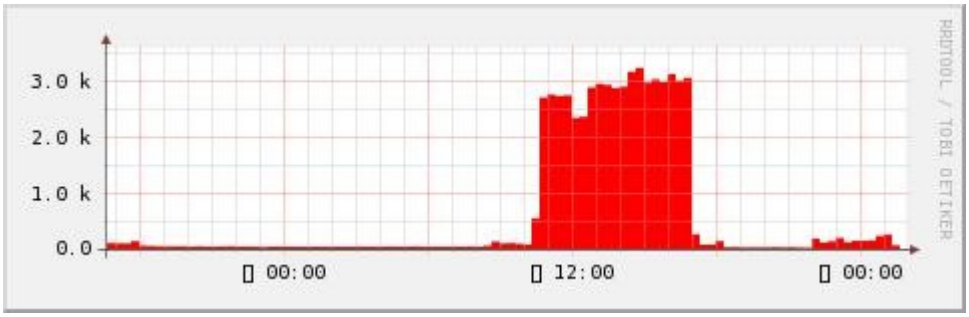


实例 3 ： 使用 –S 从第 2 个 RRA 中取数据

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 4.png
> -S 1200 \                    # 使用 –S 控制 RRDtool 从 resolution=1200 的 RRA 中取数据
> --start now-120000 \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> AREA:value1#ff0000
481x154
[root@dns1 bob]#
```



可以看到和上面的图一样，说明 RRDtool 的确按照 -S 的设置从第 2 个 RRA 中取数据了

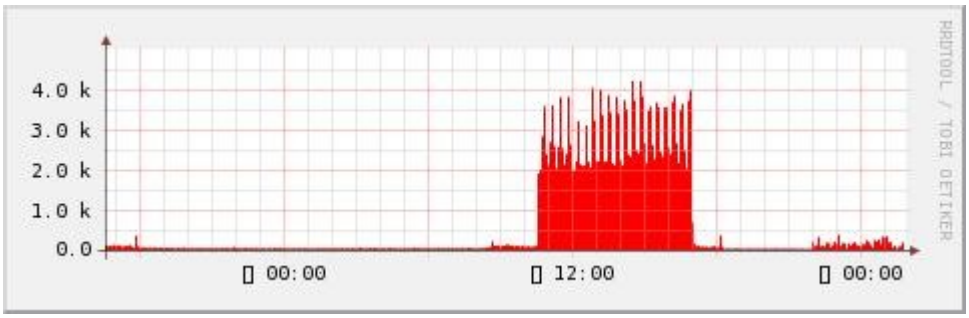
使用 –S 可以对 DEF 中所有的 DS 都使用相同的 resolution，等于在每个 DEF 后都加上 :step=<value> ，value 是 –S 的值

实例 4 ： 同时使用 –S 和 :step

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 5.png
> -S 1200 \                                    # -S 指定 resolution=1200
> --start now-120000 \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:step=300 \          # :step 指定 resolution=300
> AREA:value1#ff0000
481x154
[root@dns1 bob]#
```



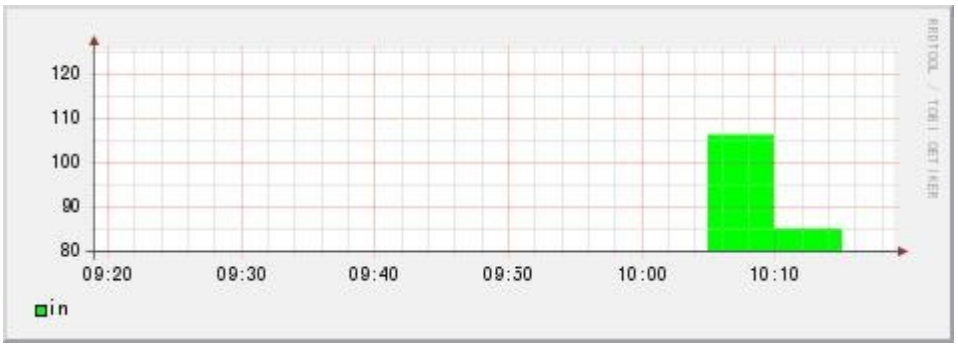
可以看到 5.png 和 1.png 是一样的，也就是说 –S 1200 并没有起作用，而是 :step=300 起作用了

实例 5 ：使用 :start 和 :end 只显示指定时间内的数据

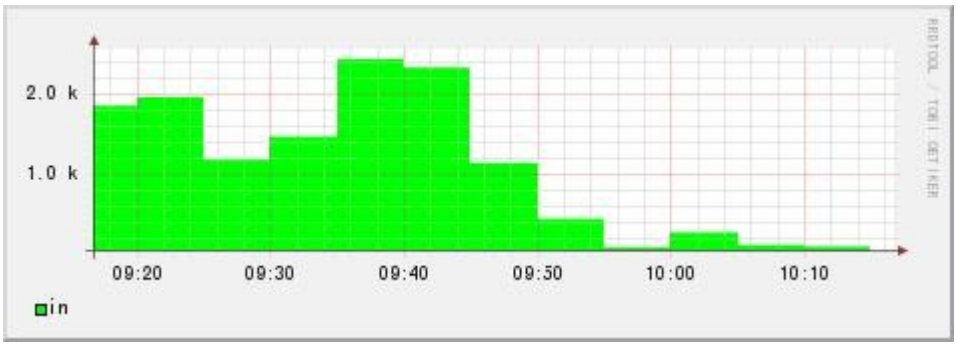
[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-1h \                             # X 轴显示 1 个小时的长度
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:start=now-600:end=now-300 \          # 但只取 10 分钟前到 5 分钟前的这部分
> AREA:value1#00ff00:in
475x168
[root@dns1 bob]#
```



如果我们不加 :start 和 :end ，则效果如下 ：



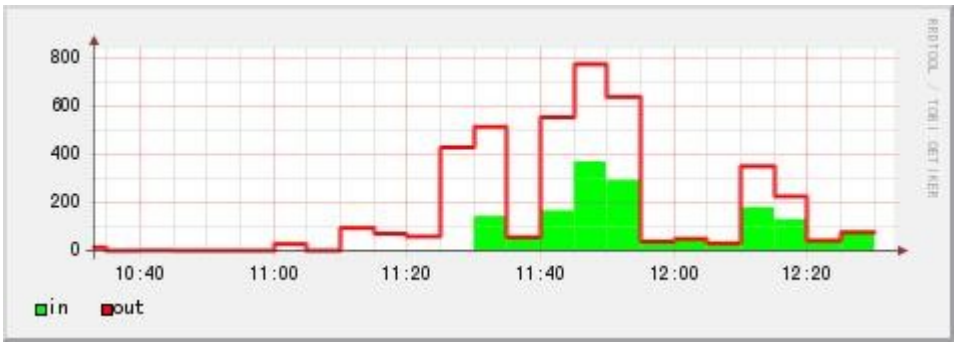
我们甚至可以让两个对象显示不同的时间，例如

实例 6 ：让两个对象显示不同时间段的数据

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-2h \                    # 规定时间为 2 小时内
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:end=now:start=end-1h \          # 规定时间为 1 小时内
> DEF:value2=eth0.rrd:eth0_out:AVERAGE \          # 没有指定 :start 和 :end，默认和 –-start 一样也是 2 小时
> AREA:value1#00ff00:in \
> LINE2:value2#ff0000:out:STACK
475x168
[root@dns1 bob]#
```



实例 7 ：把一段时间分为几段分别显示 ：

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:end=now:start=end-1h \          # 当前 1 小时内
> DEF:value2=eth0.rrd:eth0_in:AVERAGE:end=now-1h:start=now-2h \          # 2 小时前
> DEF:value3=eth0.rrd:eth0_in:AVERAGE:end=now-2h:start=now-3h \          # 3 小时前
> LINE1:value1#00ff00:"1 hours ago" \
```

```
> LINE2:value2#ff0000:"2 hours ago" \
> LINE3:value3#000000:"3 hours ago"
475x168
[root@dns1 bob]
```

我们把 3 个小时内的数据用三种不同粗细、不同颜色的曲线画了出来。

看到了吗，out 部分（红色）显示了 2 个小时内的流量，而 in 部分（绿色）则只显示了 1 个小时内的部分

在这里要提一点，虽然我们指定了 --start 或者 --end ,或者 :start , :end，但并不意味着曲线就一定会从指定的时间点开始和结束。

例如我们上面指定了 :start=now-600:end=now-300 ，也就是只显示 5 分钟的数据。但图表出来的效果却是 10(10:05-10:15)分钟

的数据，这是因为我们挑选的时间当中"不慎"横垮了两个周期(10:05-10:10,10:10-10:15)，所以 RRDtool 会把它们全部画出来，而

不是只画其中的 5 分钟。

使用 RRDtool 进行绘图_（二）

****
****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****

****

五）说明文字（Label）

[Copy to clipboard] [-]

CODE:

```
[-t|--title string] [-v|--vertical-label string]
```

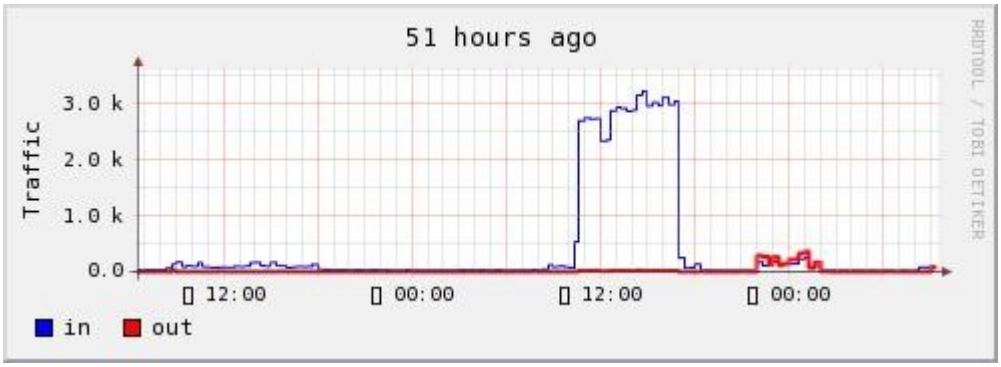
-t 是用于图表上方的标题，-v 是用于 Y 轴的说明文字

实例 1 ：给图表增加标题

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-183600 \                    # 从当前开始往前 51 个小时
> -t "51 hours ago" -v "Traffic" \        # 标题是 “51 hours ago”，Y 轴的说明文字是 “Traffic”
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> DEF:value2=eth0.rrd:eth0_out:AVERAGE \
> LINE1:value1#0000ff:in \          # 注释 ：以 1 个像素宽的曲线画出 value1，颜色是蓝色，图例的说明文字是“in”
> LINE2:value2#ff0000:out           # 注释 ：以 2 个像素宽的曲线画出 value2，颜色是红色，图例的说明文字是 “out”
497x179
[root@dns1 bob]#
```



现在我们用的是 LINE 的方式来绘图。LINE 可以有 3 种，分别是 LINE1|2|3,也就是线条的粗细。还有一种是 STACK 方式下面再介绍。

可以看到流入的流量比流出的流量稍大，这样看的话，out 流量比较难看，是否可以有别的方式呢？RRDtool 还提供了

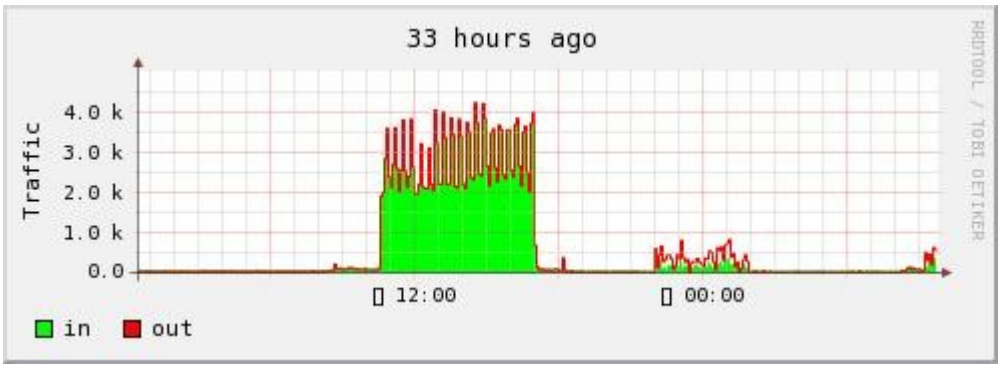
另外一种格式，就是 STACK 。意思就是在前一个对象的基础（把前一个对象的值当成 X 轴）上绘图，而不是从 X 轴开始。

实例 2 ：使用 STACK 方式绘图

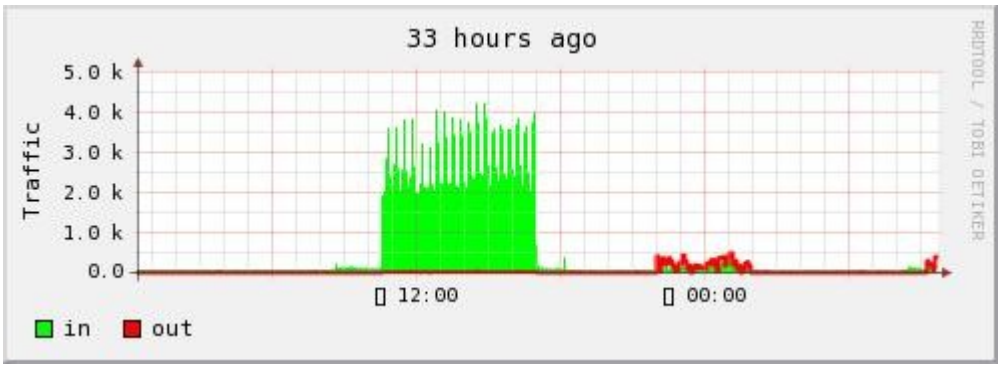
[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 3.png \
> --start now-120000 \
> -t "33 hours ago" \
> -v "Traffic" \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> DEF:value2=eth0.rrd:eth0_out:AVERAGE \
> AREA:value1#00ff00:in \
> LINE:value2#ff0000:out:STACK                    # 注意最后的 “STACK” ，表示在 value1 的基础上绘图
497x179
[root@dns1 bob]#
```



这是没有采用 STACK 方式绘图的效果 ：



可以看得出上面采用 STACK 方式的比较清晰，但要注意，采用 STACK 方式后，在读取 out 流量时，Y 轴的刻度不再是 out 的值，

应该用刻度值减去 in 的值，才是 out 真正的值。这点比较麻烦。需要配合 GPRINT 语句才能达到一定的效果。

六）图表大小（**Size**）

[Copy to clipboard] [-]

CODE:

[-w|--width pixels] [-h|--height pixels]

这里说图表大小而不是图片大小，是因为 –w ，-h 控制的是 X/Y 轴共同围起来的那部分大小，而不是整个图片的大小，这点在前面就可以看出了。

默认的图表大小是 400 （长）x 100 （高），但一般会返回 497x179 这样的数字，这个才是图片的大小。

RRDtool 比 MRTG 好的一个地方就是 MRTG 一放大图片，就会变得模糊。RRDtool 则不会。

在官方文档中，-w 似乎是一个比较敏感的参数，我们看下面的描述 ：

QUOTE:

First it makes sure that the RRA covers as much of the graphing time frame as possible. Second it looks at the resolution of the

RRA compared to the resolution of the graph. It tries to find one which has the same or higher better resolution. With the ``-r''

option you can force RRDtool to assume a different resolution than the one calculated from the pixel width of the graph.

QUOTE:

By default, rrdtool graph calculates the width of one pixel in the time domain and tries to get data from an RRA with that resolution.

With the step option you can alter this behaviour. If you want rrdtool graph to get data at a one-hour resolution from the RRD, set

step to 3'600. Note: a step smaller than one pixel will silently be ignored

这两段话分别是从 rrd-beginners 和 rrd_graph 文档中摘出来的。似乎看起来 –w 会影响到图表的 resolution ，进一步影响到 RRDtool 如何选择 RRA ，

但经过实验却发现并非如此。

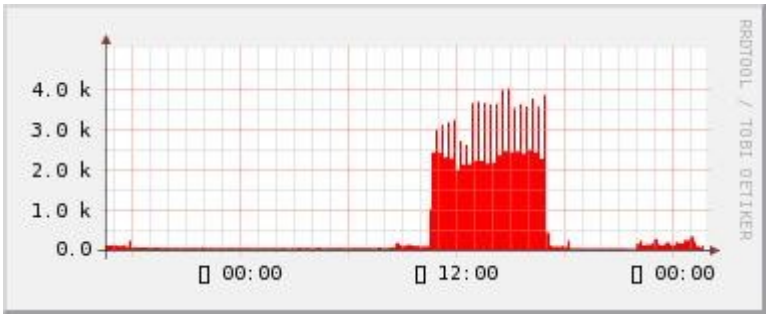
我对这两段话中的图表的 resolution 一词不知如何理解和计算，希望好心的朋友能够指点一下 ^_^ .

实例 1 ：使用 –w 设定图表大小为 300 像素

[Copy to clipboard] [-]

CODE:

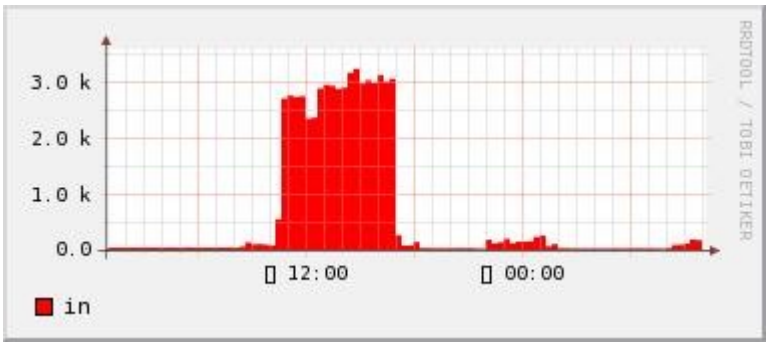
```
[root@dns1 bob]# rrdtool graph 3.png
> -w 300 \                                  # 设定 size 为 300 pixel
> --start now-120000 \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> AREA:value1#ff0000
381x154
[root@dns1 bob]#
```



可以看到图表是不是变小了呢？而且整个图片的大小也变小了。

如果用前面的话来推理，由于 120000/300（-w 的值）= 400 > 300 （step）,由于没有 resolution=400 的 RRA，

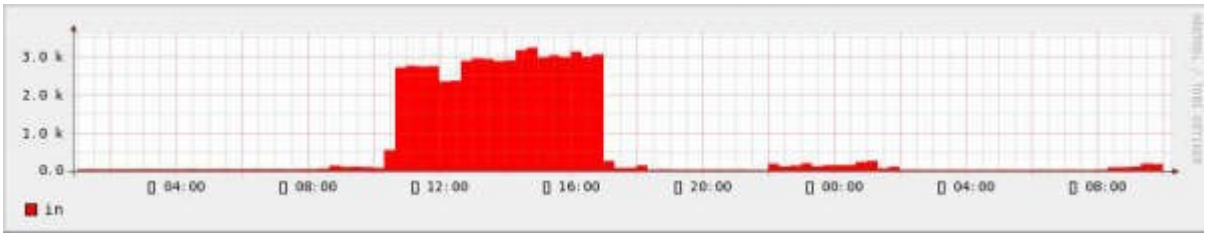
所以应该采用 resolution=7200 的第 2 个 RRA 的数据来绘图，但实际上不是。



上面这个才是 300 pixel 宽，resolution=7200 的效果

所以我觉得 –w 和 –h 并不能影响 RRDtool 如何选择 RRA ，只能起到缩小放大的作用。

BTW：当你绘制的时间范围较大时，可以使用 –w 增大图表大小，这样看起来比较"舒服"



七） Y 轴上下限（Limits）

[Copy to clipboard] [-]

CODE:

```
[-u|--upper-limit value] [-l|--lower-limit value] [-r|--rigid]
```

默认情况下，RRDtool 会和 MRTG 一样自动调整 Y 轴的数字，来配合当前的数值大小。如果想禁止该特性，可以通过 –upper-limit 和

--lower-limit 来做限制，表示 Y 轴显示的值从多少到多少。如果没有指定 –rigid ，则在图表的上下边界处还是会有一些延伸，但如果指定了

--rigid ，则严格按照 --upper-limit 和 --lower-limit 绘制。

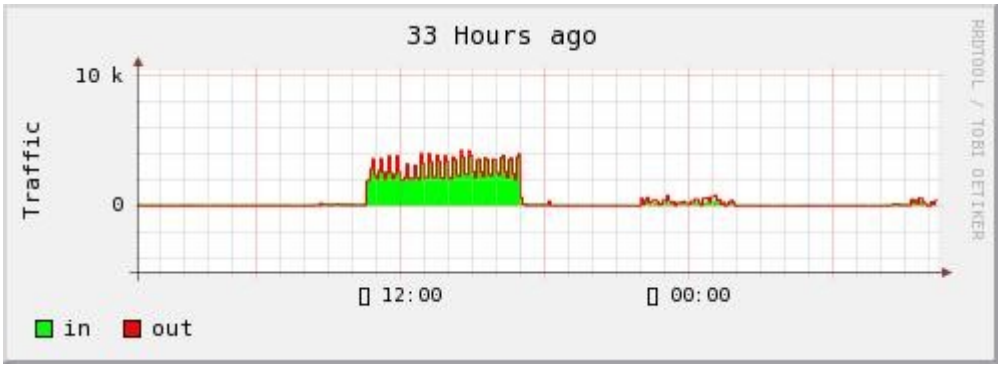
在使用 –lower-limit 时要注意，如果数据中有负数，如果 --lower-limit 为 0，则负数部分是显示不出来的。

实例 1 ：使用 --upper-limit 和 --lower-limit 限制 Y 轴的上下限

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-120000 \
> -v "Traffic" -t "33 Hours ago" \
> --lower-limit -5000 \                    # 限制 Y 轴下限为 -5000
> --upper-limit 10000 \                    # 上限为 10000
> --rigid \                                          # 严格按照上面的规定来画
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> DEF:value2=eth0.rrd:eth0_out:AVERAGE \
> AREA:value1#00ff00:in \
> LINE1:value2#ff0000:out:STACK
497x179
[root@dns1 bob]#
```



八） X/Y 轴刻度（Grid）

[Copy to clipboard] [-]

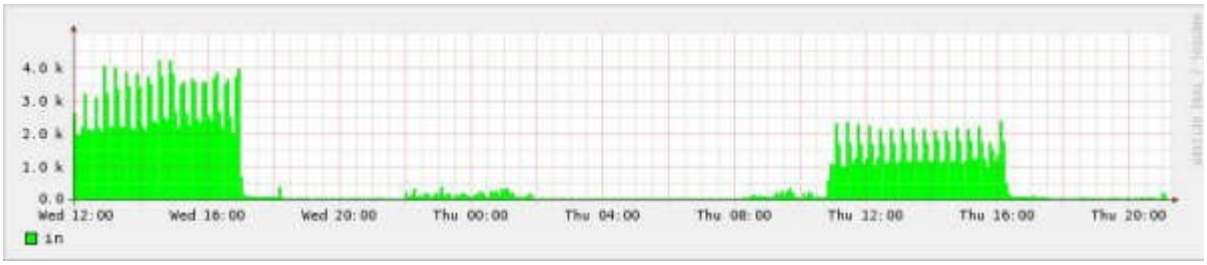
CODE:

```
[-x|--x-grid GTM:GST:MTM:MST:LTM:LST:LPR:LFM]
[-x|--x-grid none]
[-y|--y-grid grid step:label factor]
[-y|--y-grid none]
[-Y|--alt-y-grid]
[-X|--units-exponent value]
```

RRDtool 中设置 X 轴的刻度比较复杂，如果没有必要，可以交给 RRDtool 自动去处理。

例如上面的图，33 小时的情况下，X 轴只有 2 个值，显得很不足。这时候有两种方法 ：

A）一是使用 –w 增大图表的宽度，这样 RRDtool 会自动加多一些刻度上去。



不过需要增加多大才会有上面的这种效果不得而知，所以这种方法不是很方便。

B）二是通过上面的选项自己设置 X/Y 轴的刻度如何显示。首先看上图，在垂直的线中，红色的线称为 Major Grid（主要网格线），

灰色的线称为 Base Grid （次要网格线）（这里是借用 EXCEL 中的概念 ^_^ ）。 X 轴下面的时间文字成为 Label 。了解这

三样东西后，就可以动手调整刻度了。

C）有两种方法可以快速去掉 X/Y 轴的刻度，就是 --x-grid none 和 --y-grid none

D）**GTM:GST** ：控制次要格网线的位置。GTM 是一个时间单位，可以是 SECOND、MINUTE、HOUR、DAY 、WEEK、MONTH、YEAR 。

GST 则是一个数字，控制每隔多长时间放置一根次要格线。例如我们要画一个 1 天的图表，决定每 15 分钟一根次要网格线，则格式为 MINUTE:15

E）**MTM:MST** ：控制主要网格线的位置。MTM 同样是时间单位，MST 是一个数字。接上面的例子，决定一个小时 1 根主要网格线。则格式为 HOUR:1

LTM:LST ：控制每隔多长时间输出一个 label 。决定为 1 小时 1 个 label 。则格式为 HOUR:1

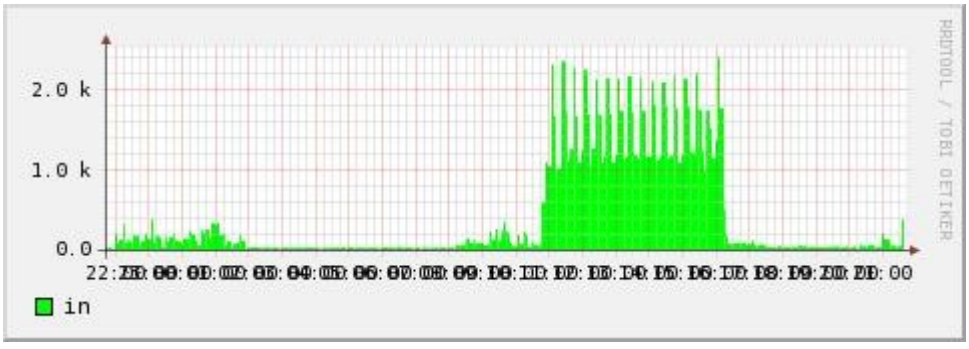
G）**LPR:LFM** ：LTM:LST 只是决定了 label 的显示位置了，没有指定要显示什么内容。LPR 指的是如何放置 label 。如果 LPR 为 0，则数字对齐格线

（适用于显示时间）。如果不为 0，则会做一些偏移（适用于显示星期几、月份等）。至于 LFM 则需要熟悉一下 date 命令的参数，常用的有 %a（星期几）、

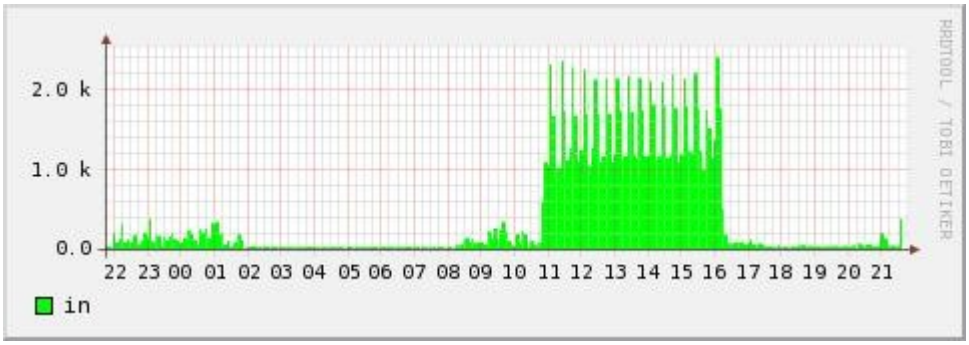
%b（月份）、%d（天）、%H（小时）、%M（分）、%Y（年）。我们决定显示小时和分，所以用 %H%M

H）综合起来，X 轴的刻度定义就是 --x-grid MINUTE:15:HOUR:1:HOUR:1:0:'%H:%M'。最好把 %H:%M 括起来

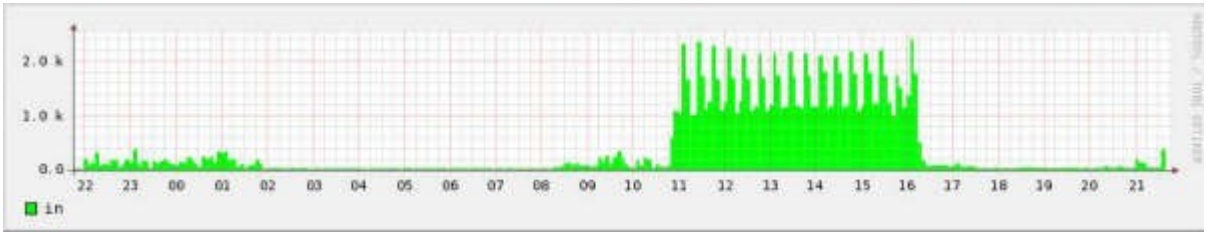
建议 MST 是 GST 的 2-6 倍，MST 和 LST 相同。这样画出来的图比较美观一些



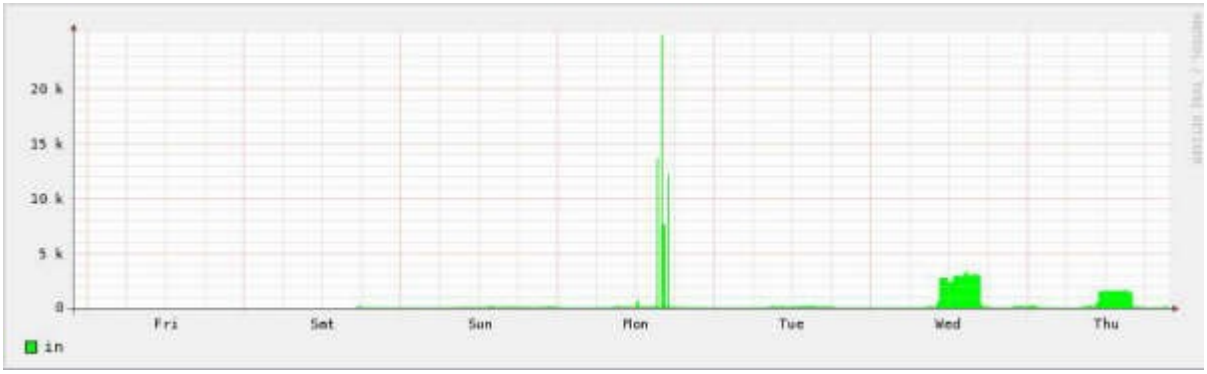
这明显就是图片太小了，不够显示。把上面的 :%M 去掉就可以了，只显示小时，不显示分钟



如果把图片放大一点就更好了 (-w 800)



所以在设置 X 轴的刻度时，要记得不要显示太多东西，否则需要增大图片的大小

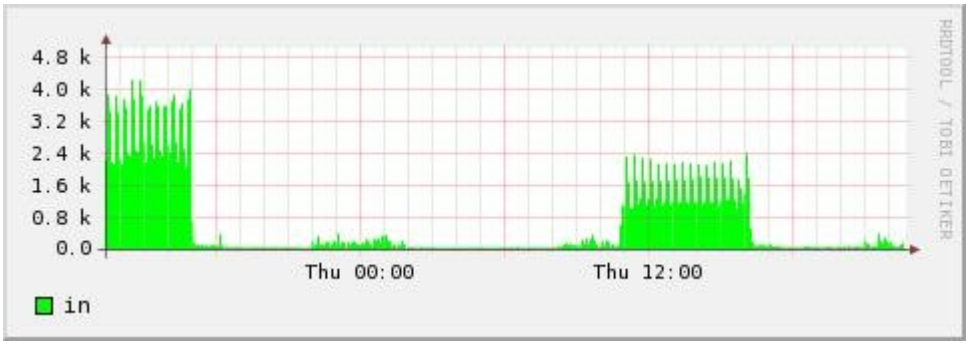


I）Y 轴刻度的设置又不一样了

grid step ：用于控制 Y 轴每隔多少显示一根水平线

label factor ：默认为 1，也就是在每根水平线的高度那里显示一个值。

例如下面就是一个例子 （每隔 800 显示一根水平线）



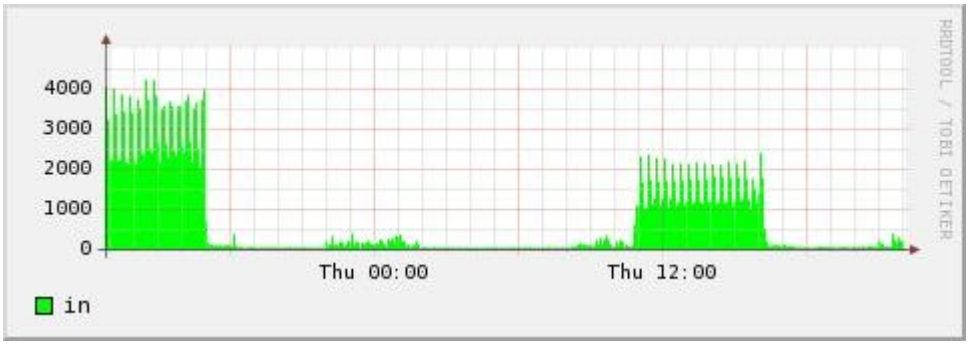
J）Y 轴还有一个很方便的选项就是 –Y ，它可以最大限度的优化 Y 轴的刻度，建议每次绘图都加上去。

K）Y 轴另外一个有用的选项就是 –X （虽然选项名是 -X ，但确实是用来设置 Y 轴刻度值的）。在上面的图我们看到 RRDtool 自动对 Y 轴的值进行调整，

以 k 为单位显示。但如果你不想以 k 显示，而是想固定以某个单位来显示（M，b）该怎么办呢？这就要用到 –X 选项了。-X 后面跟一个参数，参数值

范围是 -18、-15、-12、-9、-6 、-3、0、3、6、9、12、15、18 。0 表示以原值显示，3 表示数值除以 1000，也就是以 k 为单位显示，6 就是以

M 显示，9 就是以 G 显示。如果你给出 1 或者 2，则 RRDtool 也可以接受，但会被"静悄悄"的改为 0。下面就是一个以原值（-X 0）显示的例子



使用 RRDtool 进行绘图_（三）

****

注 ： 该教程参考了如下内容 ：

A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html

B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****
****

九、 其他（Miscellaneous）

[Copy to clipboard] [-]

CODE:

```
[-n|--font FONTTAG:size:[font]]
[-g|--no-legend]
[-b|--base value]
```

A）-n | --font 是一个有意思的选项。CU 的 abel 兄曾提供了一个中文 patch 可以实现显示中文的效果。但对于我这等对 C 一窍不通的家伙，就不知道怎么用了。

不过幸好 –n 选项可以实现这个目的，只需要中文字体的文件就可以搞定了。

首先你要找出一个中文的字体文件。例如/usr/share/fonts/zh_CN/TrueType/gkai00mp.ttf 。你也可以把 Windows 上的 C:\Windows\Fonts 下面的中

文字体拷贝到 Linux 上，例如 home/bob/Fonts/simhei.ttf （黑体）效果不错，其他的则不太行，会出现模糊或者重叠的情况。建议就使用黑体算了。

其次是确定字体大小。中文字体不宜小于 7 ， 否则看不清楚

确定你要修改的是图表的那个部分。有 DEFAULT（全部），TITLE （标题）、AXIS（坐标轴字体）、UNIT（Y 轴单位字体）、LEGEND（图例字体）几种。

下面就以实际的例子来说明如何显示中文 ：

实例 1 ： 使用 –n 让 RRDtool 显示中文

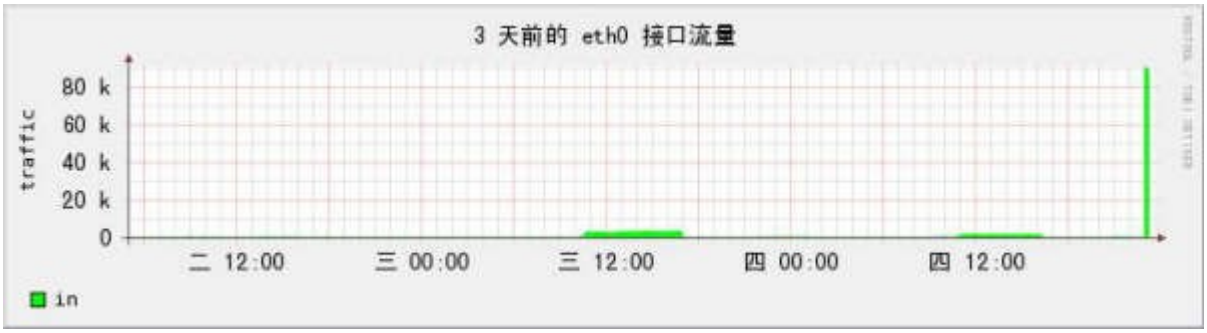
[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 4.png \
> -n TITLE:10:'/home/bob/Fonts/simhei.ttf' \                    # 修改标题的字体为黑体
> -n AXIS:10:'/home/bob/Fonts/simhei.ttf' \                     # 修改 X 轴的字体为黑体
> --start now-240000 \                    # 大于 3 天的数据
```

```
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> AREA:value1#00ff00:in -t "33 小时前的 eth0 接口流量" -v "traffic"
> -Y –w 600          # 图表宽度为 600 pixel
503x190
[root@dns1 bob]#
```

这就是最终的效果了，可以看到标题和 X 轴都是中文的，但 Y 轴的字体还是默认的字体。



B）-g |--no-legend 用于取消图表下方的图例，不过不建议这么做。

C）-b|--base value 在 MRTG 和 RRDtool 中，默认 1k=1000 ，使用 –b 可以进行调整，例如 –b 1024

十）数字报表

看看上面的图表，是不是觉得还少了些什么呢？对了，就是只有图，没有文字说明。如何象 MRTG 那样能够显示“最大值”、“平均值”、“当前值”呢？

这就需要用到 GRPINT 和 COMMENT 语句了。
GPRINT 就是在图表的下方（仍然属于图片的内部）输出最大值、最小值、平均值这些东东；COMMENT 就是用来输出一些字符串，例如报表的表头。

A）GPRINT 的格式是 GPRINT:vname:CF:format 。由于 format 部分太多参数了，我这里就用最常用的那个 ： %x.ylf 。

B）COMMENT 的格式是 COMMENT:text 。要注意 COMMENT 默认是不输出换行的，如果要输出换行，必须用 “\n” 。

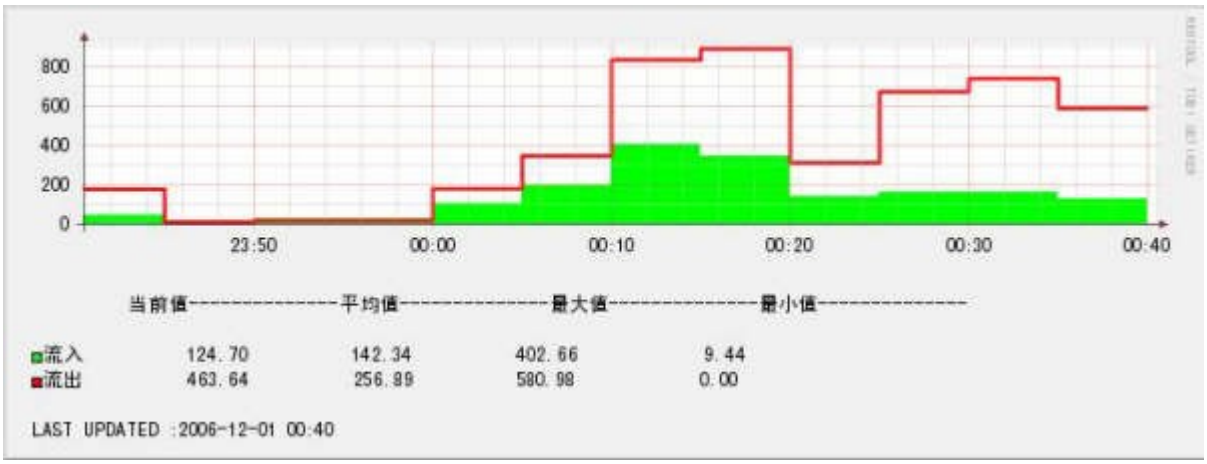
下面就以一个实例来说明如何打印报表 ： 绘制 1 小时前的流量图，并打印数字报表（参照 abel 兄给出的例子）

[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 1.png \
> --start now-1h -w 600 -n DEFAULT:8 \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE \
> DEF:value2=eth0.rrd:eth0_out:AVERAGE \
```

```
> COMMENT:" \n" \
> COMMENT:"          当前值--------------平均值--------------最大值--------------最小值
--------------\n"                                  > COMMENT:" \n"
> AREA:value1#00FF00:"流入" \
> GPRINT:value1:LAST:'%13.2lf' \
> GPRINT:value1:AVERAGE:%13.2lf \
> GPRINT:value1:MAX:%13.2lf \
> GPRINT:value1:MIN:%13.2lf \
> COMMENT:" \n" \
> LINE2:value2#ff0000:"流出":STACK \          # 注意这里是 STACK 方式
> GPRINT:value2:LAST:%13.2lf \
> GPRINT:value2:AVERAGE:%13.2lf \
> GPRINT:value2:MAX:%13.2lf \
> GPRINT:value2:MIN:%13.2lf \
> COMMENT:" \n" \
> COMMENT:" \n" \
> COMMENT:"LAST UPDATED \:$(date '+%Y-%m-%d %H\:%M')\n" -Y
687x270
[root@dns1 bob]#
```



注意比较 Y 轴刻度值和"流出"部分的值的关系， Y 轴刻度值 —"流入" ="流出"

由于时间精力有限，关于对齐方面的工作就大家自己试验吧.如果绘制的对象数量不是很多，可以用横向报表，不要用这种垂直的格式，

这种格式的好处是便于比较各个对象的值。不过我可以肯定，如何让这些数字和上面的表头对齐是一个会令你极度抓狂的工作的！！！

上面的 COMMENT 一是输出表头，二是输出空行。注意，要用 COMMENT 输出空行，必须用 COMMENT:'\n'。

注意到'\'前面的空格吗？这个是不可以漏的，否则就不会有空行的效果了。

十一）特殊功能

[Copy to clipboard] [-]

CODE:

```
VRULE:time#color[:legend]
HRULE:value#color[:legend]
SHIFT:vname:offset
```

A）VRULE/HRULE 可以用于在图表上面绘制垂直线/水平线。例如我们想要在图表上面标出最大值，可以用 HRULE 在 Y 轴的指定刻度值

那里绘制一根水平线，例如 HRUE:100000#ff0000:"最大值" 在 100k 处画一根水平线，并指出这是最大值。

SHIFT 可以用来移动数据，例如 abel 兄曾经在 "[教學]中的教學(二) RRDTOOL 1.2 更新項目"中提到过一个问题，

就是"xx 同期相比"如何画？下面就以如何比较 3 天的数据。

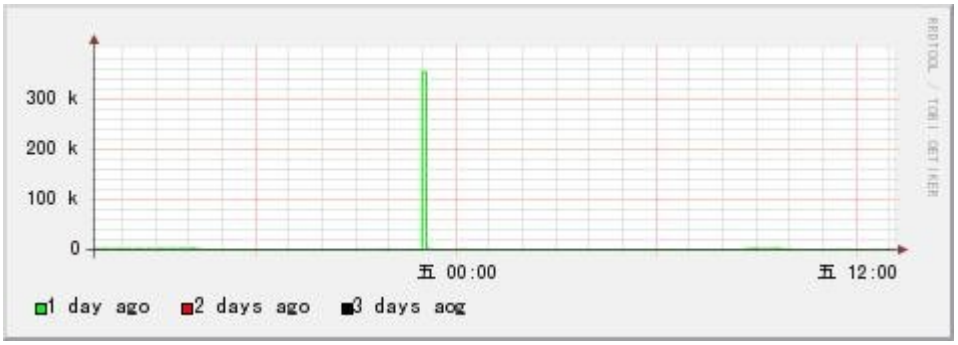
实例 1 ：绘制连续 3 天的数据

[Copy to clipboard] [-]

CODE:

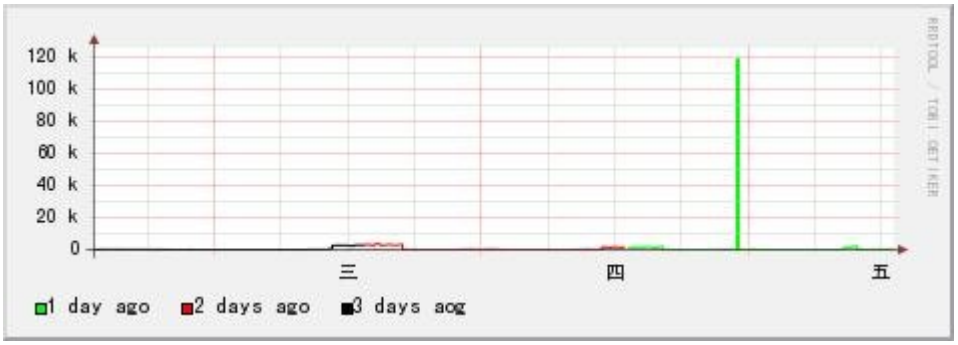
```
[root@dns1 bob]# rrdtool graph 1.png \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:end=now:start=now-1d \          # 1 天前
> DEF:value2=eth0.rrd:eth0_in:AVERAGE:end=now-1d:start=now-2d \        # 2 天前
> DEF:value3=eth0.rrd:eth0_in:AVERAGE:end=now-2d:start=now-3d \        # 3 天前
> LINE1:value1#00ff00:"1 day ago" \
> LINE1:value2#ff0000:"2 days ago" \
> LINE1:value3#000000:"3 days aog" \
```

```
> -Y
475x168
You have new mail in /var/spool/mail/root
[root@dns1 bob]#
```



为什么只有 1 天前的数据呢？因为我们没有指定 --start ，RRDtool 默认只绘制 1 天前的数据。由于这里覆盖了 3 天，

所以我们可以把 --start 定义为 --start now-3d 就可以了。



现在是不是 3 天的数据都画出来了呢？不过由于它们是横向排列的，所以要比较同个时间段的并不容易，能否把它们

按"垂直"的方式排列呢？这就要用到 SHIFT 了！

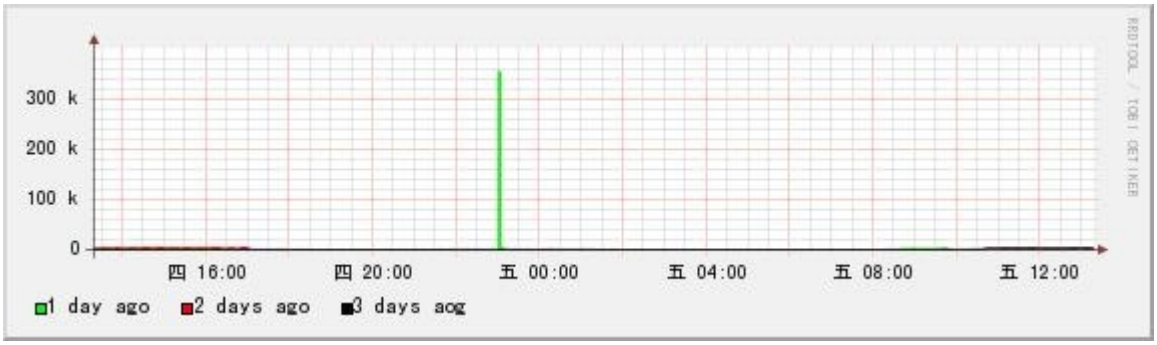
[Copy to clipboard] [-]

CODE:

```
[root@dns1 bob]# rrdtool graph 3.png \
> DEF:value1=eth0.rrd:eth0_in:AVERAGE:end=now:start=now-1d \              # 1 天前
> DEF:value2=eth0.rrd:eth0_in:AVERAGE:end=now-1d:start=now-2d \             # 2 天前
> DEF:value3=eth0.rrd:eth0_in:AVERAGE:end=now-2d:start=now-3d \             # 3 天前
> LINE1:value1#00ff00:"1 day ago"  \
> SHIFT:value2:86400 LINE1:value2#ff0000:"2 days ago" \                  #  把曲线向右移动 1 天
> SHIFT:value3:172800 LINE1:value3#000000:"3 days ago" \                  #  把曲线向右移动 2 天
```

```
> -Y –w 600
475x168
```

[root@dns1 bob]#



和上面的图表比较，是否可以发现 X 轴不同了，不再是 3 天，而是 1 天多 1 点了。而且 3 根曲线重叠在一起了，可以看出在这三天中，

只有 1 天前的 23 点左右有一点流量之外，其余绝大部分都没有流量。

这就是 SHIFT 的功能了，可以把曲线/方块沿着 X 轴移动（左右都可以），我们达到比较同期数据的目的。是不是很好用呢？

十二）总结

这次的内容可真够多的，足足写了 19 页。不过工具性的东西就是这样 ：别看内容 N 多，你只要动手画出 1 个图之后，就会觉得一

切都很简单了。以后只要套用就可以了。关键是如何更好的把你想要的数据以合适的发给你是呈现出来。

上面这些内容都是我通过实验得出的，由于具体的环境不同，可能会跟大家的不同，或者出现错误。我希望大家不要客气，有错误

的地方就指正，有什么好的发现也提出来，一起完善 RRDtool 的文档。这样就可以让越来越多的人了解、掌握 RRDtool 了 。

RRDtool 简体中文教程_9：如何使用 RPN
如何使用 RPN

****
****
注 ： 该教程参考了如下内容 ：
A）官方文档 ： http://oss.oetiker.ch/rrdtool/doc/index.en.html
B）abel 兄的大作 ： http://bbs.chinaunix.net/viewthread.php?tid=552224&highlight=rrdtool
http://bbs.chinaunix.net/viewthread.php?tid=552220&highlight=rrdtool
作者 ： ailms <ailms{@}263{dot}net>
版本 ： v1
最后修改 ： 2006/11/17 17:35
****
****

一）前言

RPN 代表逆波兰式（Reverse Polish Notation）。逆波兰式最早于 1920 年由 Jan Lukasiewicz 发明，最神奇的地方是用它来表示数学表达式，

完全不需要括号。而且 RPN 不像普通的数学表达式那样，操作符在操作数的中间，而是在操作数的右边。例如 3+2 用 RPN 表示就是 3,2,+ ；

3+（2X5） 用 RPN 表示就是 3,2,5,x,+ 最后运算的部分（加法部分）的操作符放在最后，乘号放在前面，表示先执行 2 x 5 ，在把结果和 3 相加。

在 RRDtool 中，RPN 还可以用来表示 if-then-else 关系。这点在绘图中很有用。例如你要看 eth0 接口在一天当中流量 ≥ 10Mb/s 的部分，“隐藏”

其他低于 10Mb/s 的部分，则可以用到这个功能。

二）操作符

什么是 RPN

A）RPN 是 Reverse Polish Notation 的缩写，是用于表示算术运算和逻辑运算的一种语法格式。

B）RRDtool 的 CDEF 语句中就经常使用 RPN 来对 DEF 取出来的数据进行运算。

C）RPN 的特点是操作数和操作符出现的顺序和运算的顺序一致，这样就不需要使用括号了

D）RPN 的格式是 <value1>,<value2>,<operator> ,[<value1>,<value2>,<operator>]… 可以看出是操作数在前，操作符在后的格式

E）RPN 需要提到堆栈的概念（stack）。堆栈是用来存储操作数和操作符的。

F）当堆栈中压入（push）一个操作符时，就从堆栈中取出（pop）所需要的操作数进行计算（根据操作的不同 pop 出不同数量的操作数）。

结果再返回（push）堆栈，最终整个 RPN 应该只返回一个值，或者说堆栈中只有一个元素

G）在 CDEF 中书写 RPN 操作符，要一律以大写的格式出现

H） RPN 中，如果某个部分的运算结果非 0，则被认为是 true ，只有 0 才被认为是 false

三）RPN 操作符的分类

A）布尔操作符 ：GT、GE、LT、LE、EQ、NE、

B）特殊值比较符 ：UN、ISINF、

C）条件操作符 ：IF

D）比较操作符 ：MIN、MAX、LIMIT

E）算术操作符 ：+ 、-、*、/、%、SIN, COS, LOG, EXP, SQRT、FLOOR, CEIL、ATAN、ATAN2、DEG2RAD, RAD2DEG

F）数据集操作符 ：所谓数据集（sets），就是指多个数据。SORT、REV、AVG、TREND

G）特殊值 ：UNKN、INF、NEINF、PREV、COUNT

H）时间操作符 ：NOW、TIME、LTIME

I）堆栈操作符 ：POP、DUP、EXC

四）RPN 操作的结果

A）布尔操作符 ：从堆栈中 pop 出两个元素，并根据比较结果返回 0 （false） 或者 1 。任何同 UNKNOWN 或者 INF 、NEINF 比较的都为 0

B）特殊值比较符 ：从堆栈中 pop 出 1 个元素，并同 UNKNOWN 或者 INF、NEINF 比较。结果为 0 或者 1

C）条件操作符 ：从堆栈中 pop 出 3 个元素，如果最后 pop 出的那个元素不为 0（条件部分为 true），则第 2 次 pop 出的那个元素被重新入栈（then 部分）；

否则第一次 pop 出的元素重新入栈（else 部分）。结果为 then 部分或者 else 部分返回的值，不一定为 0 或者 1

D）比较操作符 ：

对于 MIN/MAX 操作符来说，从堆栈中 pop 出两个操作符，并把较大/小的那个重新入栈。如果其中有一个 unknown ，则结果为 unknonw

对于 LIMIT 操作符来说，先从堆栈中 pop 出 2 个操作数，作为边界的定义；再 pop 出 1 个操作数，比较该操作数是否落在前面定义的范围内。

如果是则把最后 pop 出的那个元素重新入栈；否则把 UNKN 值入栈；注意，是 UNKN ，不是 0

E）算术操作符 ：根据操作符 pop 出所需数量的操作数，并把算术运算的结果重新入栈

F）数据集操作符 ：

对于 SORT、REV 来说，先从堆栈中 POP 出一个元素，该元素的值就是下面要 pop 出的元素的数量。然后对堆栈从上到下的若干个元素

由第一次 pop 的出的那个元素的值决定）进行排序/反向排序。结果再重新入栈。

注意 ：由于堆栈的特点是后进先出，所以要操作的元素是从 SORT 操作符往左方向计数。例如 v1,v2,v3,v4,3,SORT 是对 v2~v4 排序，

不是对 v1~v3 排序。 这点在书写 RPN 时要特别注意。

注意 ：SORT 操作是最小值在堆栈的最顶部；REV 则相反，最小值是在堆栈的最顶部。

对于 AVG 操作来说，同样是先 pop 出 1 个元素，并按照指定的数量对后续的若干个操作数进行操作，但结果只有一个数值，并入栈。

G）特殊值 ：

UNKN 表示压入一个 UNKN 值；INF、NEINF 分别表示把 INF、NEINF 值压入堆栈

H）时间操作符 ：

TIME 返回当前所提取的记录的 timestamp ，注意 TIME 直接返回当前记录的 timestamp ，不用任何

参数

NOW 返回当前时间，同样 NOW 不用任何参数

I）堆栈操作符 ：

POP ：弹出堆栈的最顶部的那个元素

EXC ：交换堆栈顶部的第一个和第二个元素的值

五）如何阅读 RPN

A）首先按照从左到右的顺序，找出第一个 RPN 操作符，并根据上一节的内容，对相应的操作数进行操作

B）操作结果分成两种 ：

如果是一个值，直接替换掉该部分 RPN

如果是多个值（数据集操作，但 AVG 操作只返回一个值），则结果可能为多个数值。则把这若干个数值用 ‘,’ 隔开，替换原来那部分 RPN

C）如此循环，一直到整个 RPN 只返回一个值为止

六、RPN 实例

A）布尔型操作符 ：2,1,GE 表示 2>=1 ；

B）特殊值比较符 ：mydata,UN 表示 mydata == UNKNOWN

C）条件操作符 ：mydata,UN,0,mydata,IF 表示如果 mydata 等于 UNKNOWN ，则返回 0；否则还是返回 mydata 本身

D）比较操作符 ：mydata,20,MAX 表示返回 mydata，20 这两个数值中较大的一个；alpha,0,100,LIMIT 表示测试 alpha 的值是否小于等于 0，大于等于 100；

E）算术操作符 ：1,2,- 表示 1-2=-1

F）数据集操作符 ：

v1,v2,v3,v4,v5,4,SORT 表示对 v1~v4 进行正向排序，结果堆栈中还是有 5 个元素；

v1,v2,v3,v4,3,AVG,+,2,/ 表示对 v4,v3,v2 进行求平均值，并把结果入栈。假设 v2~v4 的结果为 k ，

则为 v1,k,+,2,/ 也就是返回 (v1+k)/2

G）特殊值 ：mydata,0,GT,UNKN,mydata,IF 表示如果 mydata 大于 0 则返回 UNKNOWN ，否则还是 mydata

H）时间操作符 ： TIME,`date –d "2006-10-01 10:00" +%s`,GT,0,1,IF 表示如果当前记录的采集时间是在 2006-10-01 10:00 之后就返回 1，否则返回 0

I）堆栈操作符 ：POP 就立即弹出第一个元素

七）如何表示 AND、OR 关系

A）我们知道 RPN 表达式的值除非 0，否则都认为是 true

B）我们可以利用 加法操作和乘法操作来实现 OR 和 AND 的逻辑关系；如果两个 RPN 表达式的值相加不等于 0，就一定为 true ；

如果两个 RPN 表达式的值相乘不等于 0，就一定为 true

C）AND 关系的例子 ：例如要比较某个值（15，9）是否在特定范围内可以用 ：

15,10,GT,15,20,LT,* ，结果就是（15>10）*（15 < 20）= 1 * 1 =1 ，所以为 true

9,10,GT,9,20,LT,* ，结果就是 （9 >10）* （ 9 < 20）= 0 * 1 =0 ，所以为 false

D）OR 关系的例子：同上例如要比较某个值(7,15)小于 10，或者大于 20：

7,10,LT,7,20,GT,+ ，结果为 （7 < 10）+ （7 > 20） = 1+0 = 1,所以为 true

15,10,LT,15,20,GT,+ ，结果为 (15 < 10) + (15 > 20) = 0 + 0 =0 ，所以为 false

E）不过使用 + 需要注意一个地方 ：相加的双方都必须是正数，否则可能出现问题，例如一个正数（-5，true）和另外一个正数（5，true）相加为 0（false）

如果是按照 OR 的关系，应该是 true 的，但结果变成 0（false），所以在使用 + 来表示 OR 的关系时，要注意该问题

F）使用 * 则没有该问题了，正数 * 负数 = 负数 （true）。所以如果遇到 OR 关系的时候，可以转换为 AND 关系来计算。

例如要表达 (x < a) OR (x > b) 的关系，可以改为 (x >a) AND (x < b) ，诀窍就是把比较操作符调反方向，把 + 改为 *

八）实例

实例 1：例如要看 eth0 的总流量，可以用如下的定义

[Copy to clipboard] [-]

CODE:

```
DEF:value1=eth0.rrd:eth0_in:AVERAGE \
DEF:value2=eth0.rrd:eth0_out:AVERAGE \
CDEF:value3=value1,value2,+ \
AREA:value3#ff0000:”total”
```

实例 2 ：假设我们要把 eth0 和 lo 的流入流量相加，得出总的流入流量

[Copy to clipboard] [-]

CODE:

```
DEF:value1=eth0.rrd:eth0_in:AVERAGE \
DEF:value2=lo.rrd:lo_out:AVERAGE \
CDEF:value3=value2,UN,0,value2,IF \
CDEF:value4=value1,value3,+ \
AREA:value4#00ff00:”total in”
```

由于 lo.rrd 一直没有数据插入，所以一直都是 NaN ，如果直接把 value1 和 value2 相加，由于 value2 是 UNKNOWN，

所以相加的结果也是 UNKNOWN 。图表上将什么都不显示，所以需要对 value2 进行判断，如果 value2 的值 UNKNOWN （value2,UN），

则返回 0，否则返回 value2 本身。然后把这个值赋予变量 value3 ，最后把 value1 和 value3 相加，才得出真正入流量

实例 3 ：只看 eth0 中流量大于 10Mb/s 的部分，其余不看

[Copy to clipboard] [-]

CODE:

```
DEF:value1=eth0.rrd:eth0_in:AVERAGE \
DEF:value2=eth0.rrd:eth0-_out:AVERAGE \
CDEF:value3=value1,1000000,GT,value1,UNKN,IF \
CDEF:value4=value2,1000000,GT,value2,UNKN,IF \
AREA:value3#00ff00:”traffic_in \> 10M\/s” \
AREA:value4#ff0000:”traffic_out \> 10Mb\/s”:STACK
```

实例 4 ：只绘制特定时间段（在 2006/11/29 10:30 ~ 2006/11/29 12:30）的数据

[Copy to clipboard] [-]

CODE:

```
DEF:value1=eth0.rrd:eth0_in:AVERAGE \
DEF:value2=eth0.rrd:eth0_out:AVERAGE \
CDEF:value3=TIME,$(date –d ‘2006-11-29 10:30’ +%s),GT,TIME,$(date –d '2006-11-29 12:30'
+%s),LT,*,value1,UNKN,IF \
CDEF:value4=TIME,$(date –d ‘2006-11-29 12:30’ +%s),GT,TIME,$(date –d '2006-11-29 13:30'
+%s),LT,*,value2,UNKN,IF \
AREA:value3#00ff00:”traffic_in” \
AREA:value4#ff0000:”traffic_out”:STACK
```

九、完结

相信到目前为止，大家对 RRDtool 的认识应该更深了吧。一定要多做实验，这样才能做到熟能生巧，灵活应用。

其实剩下的还有 xport、dump、restore、resize、tune、rrdcgi 几个操作没讲，而且有一些应用经验方面的东西也没有提到，

不过想要全部写出来，可能太耗时间和精力了，这些东西足足写了我 2 个星期才写完。中间还要不断的做实验以验证正确性，怕误导了大家。

如果需要的话，可以自己下载官方文档学习，或者能有热心的朋友补充就更好了，^_^ 。

十、 本人的一点学习体会

本人从开始看 RRDtool 官方文档到开始写这篇教程，差不多用了 2 个月。RRDtool 比学习 MRTG 难多了，资料少，RRDtool 的中文资料目前就只有 abel

兄写的那一篇教程，如果没有实际的上机操作，是不可能看懂的，所以 abel 兄也特别交代这点。如果只一心想速成，到头来反而吃亏的是自己。

象 sendmail、bind 这些服务器的配置，随便在 google 上都可以搜到一大把所谓的“快速入门”，很多人也都照着做了。但明明别人可以的，为什么轮到自己

却失败呢？相信这是很多人心中曾有的郁闷经历。其实归根到底就是基础的问题，再深入一些就是学习心态的问题。“不积跬步，无以至千里；不积小流,无以成江海”。

配置一个服务器并不是照抄配置就可以的。环境的不同，需求的不同这些因素都要考虑在内。怎么可能做到完全一样呢？同一个语句换个环境可能就不行了。所以我很

少看那些所谓的快速入门，要么看 manual ，要么看书（说到这里，感觉 O’Reilly 的书真是不错！^_^），如果是象 RRDtool 这种的，就只好看官方文档了。

学习的同时也要注意选择好的教材。有时候一本好书能带给无穷的好处。这点在我第一次看 O'Reilly 的《dns & bind 4th》就有感觉，老外的书很注重循序渐进，

通常他们都是从某个实际工作环境的一个小例子说起，逐步引入各个命令、配置语句。然后随着需求的壮大，不断引入新的内容，最后形成一个总体。这样看完后会心中会

有一个整体的框架和概念。不象国内一些书，毫不顾及条理，一上来就讲语法、命令，搞得读者很快都没有兴趣。这样的书可谓害人不浅。

同时也建议大家读英文原版的书。为什么呢？虽然中文的看起来快一些，但学习不是竞走比赛。不是比谁看的快，而是比谁学的牢。英文书的词汇其实都是专业词汇，

只要看多了，自然记住了。实在记不住，可以用金山词霸等工具辅助。俺的英文水平只有二级，但并不妨碍我看书。况且看英文书，有一个"英文→中文"的转换的过程。就

是揣摩作者这句话的含义，或者说这句话应该如何翻译好。有些人觉得这个没有什么，但我觉得这个过程是你弄清作者思想的重要步骤。在你不断的揣摩中，可能会有不同

的理解，直到你认为这是最正确的那一种解释为止。如果是看中文书，可能会由于惰性，比较容易就接受作者的想法，而失去这个主动我思考的过程。

一时有感而发，胡乱写了一通，请各位朋友见谅了。