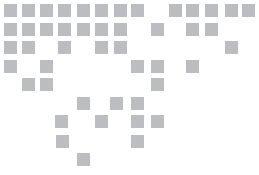
第一部分 Part 1

基础篇

- 第 1 章　系统基础信息模块详解
- 第 2 章　业务服务监控详解
- 第 3 章　定制业务质量报表详解
- 第 4 章　Python 与系统安全



Chapter 1

第 1 章 系统基础信息模块详解

系统基础信息采集模块作为监控模块的重要组成部分，能够帮助运维人员了解当前系统的健康程度，同时也是衡量业务的服务质量的依据，比如系统资源吃紧，会直接影响业务的服务质量及用户体验，另外获取设备的流量信息，也可以让运维人员更好地评估带宽、设备资源是否应该扩容。本章通过运用 Python 第三方系统基础模块，可以轻松获取服务关键运营指标数据，包括 Linux 基本性能、块设备、网卡接口、系统信息、网络地址库等信息。在采集到这些数据后，我们就可以全方位了解系统服务的状态，再结合告警机制，可以在第一时间响应，将异常出现在苗头时就得以处理。

本章通过具体的示例来帮助读者学习、理解并掌握。在本章接下来的内容当中，我们的示例将在一个连续的 Python 交互环境中进行。

进入 Python 终端，执行 python 命令进入交互式的 Python 环境，像这样：

```
# python
Python 2.6.6 (r266:84292, Nov 22 2013, 12:16:22)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1.1 系统性能信息模块 psutil

psutil 是一个跨平台库（http://code.google.com/p/psutil/），能够轻松实现获取系统运行的

进程和系统利用率（包括 CPU、内存、磁盘、网络等）信息。它主要应用于系统监控，分析和限制系统资源及进程的管理。它实现了同等命令行工具提供的功能，如 ps、top、lsof、netstat、ifconfig、who、df、kill、free、nice、ionice、iostat、iotop、uptime、pidof、tty、taskset、pmap 等。目前支持 32 位和 64 位的 Linux、Windows、OS X、FreeBSD 和 Sun Solaris 等操作系统，支持从 2.4 到 3.4 的 Python 版本，目前最新版本为 2.0.0。通常我们获取操作系统信息往往采用编写 shell 来实现，如获取当前物理内存总大小及已使用大小，shell 命令如下：

```
物理内存 total 值：free -m | grepMem | awk '{print $2}'
物理内存 used 值：free -m | grepMem | awk '{print $3}'
```

相比较而言，使用 psutil 库实现则更加简单明了。psutil 大小单位一般都采用字节，如下：

```
>>> import psutil
>>>mem = psutil.virtual_memory()
>>>mem.total,mem.used
(506277888L, 500367360L)
```

psutil 的源码安装步骤如下：

```
#wget  https://pypi.python.org/packages/source/p/psutil/psutil-2.0.0.tar.gz
--no-check-certificate
# tar -xzvf psutil-2.0.0.tar.gz
# cd psutil-2.0.0
# python setup.py install
```

1.1.1　获取系统性能信息

采集系统的基本性能信息包括 CPU、内存、磁盘、网络等，可以完整描述当前系统的运行状态及质量。psutil 模块已经封装了这些方法，用户可以根据自身的应用场景，调用相应的方法来满足需求，非常简单实用。

（1）CPU 信息

Linux 操作系统的 CPU 利用率有以下几个部分：

- ❑ User Time，执行用户进程的时间百分比；
- ❑ System Time，执行内核进程和中断的时间百分比；
- ❑ Wait IO，由于 IO 等待而使 CPU 处于 idle（空闲）状态的时间百分比；
- ❑ Idle，CPU 处于 idle 状态的时间百分比。

我们使用 Python 的 psutil.cpu_times() 方法可以非常简单地得到这些信息，同时也可以获取 CPU 的硬件相关信息，比如 CPU 的物理个数与逻辑个数，具体见下面的操作例子：

```
>>> import psutil
>>>psutil.cpu_times()# 使用 cpu_times 方法获取 CPU 完整信息，需要显示所有逻辑 CPU 信息，
>>># 指定方法变量 percpu=True 即可，如 psutil.cpu_times(percpu=True)
scputimes(user=38.039999999999999, nice=0.01, system=110.88, idle=177062.59,
iowait=53.399999999999999, irq=2.9100000000000001, softirq=79.579999999999998,
steal=0.0, guest=0.0)
>>>psutil.cpu_times().user     # 获取单项数据信息，如用户 user 的 CPU 时间比
38.0
>>>psutil.cpu_count()     # 获取 CPU 的逻辑个数，默认 logical=True4
>>>psutil.cpu_count(logical=False)     # 获取 CPU 的物理个数
2
>>>
```

（2）内存信息

Linux 系统的内存利用率信息涉及 total（内存总数）、used（已使用的内存数）、free（空闲内存数）、buffers（缓冲使用数）、cache（缓存使用数）、swap（交换分区使用数）等，分别使用 psutil.virtual_memory() 与 psutil.swap_memory() 方法获取这些信息，具体见下面的操作例子：

```
>>> import psutil
>>>mem = psutil.virtual_memory()    # 使用 psutil.virtual_memory 方法获取内存完整信息
>>>mem
svmem(total=506277888L, available=204951552L, percent=59.5, used=499867648L,
free=6410240L, active=245858304, inactive=163733504, buffers=117035008L,
cached=81506304)
>>>mem.total     # 获取内存总数
506277888L
>>>mem.free      # 获取空闲内存数
6410240L
>>>psutil.swap_memory()      # 获取 SWAP 分区信息 sswap(total=1073733632L, used=0L,
free=1073733632L, percent=0.0, sin=0, sout=0)
>>>
```

（3）磁盘信息

在系统的所有磁盘信息中，我们更加关注磁盘的利用率及 IO 信息，其中磁盘利用率使用 psutil.disk_usage 方法获取。磁盘 IO 信息包括 read_count（读 IO 数）、write_count（写 IO 数）、read_bytes（IO 读字节数）、write_bytes（IO 写字节数）、read_time（磁盘读时间）、write_time（磁盘写时间）等。这些 IO 信息可以使用 psutil.disk_io_counters() 获取，具体见下面的操作例子：

```
>>>psutil.disk_partitions()     # 使用 psutil.disk_partitions 方法获取磁盘完整信息
[sdiskpart(device='/dev/sda1', mountpoint='/', fstype='ext4', opts='rw'),
sdiskpart(device='/dev/sda3', mountpoint='/data', fstype='ext4', opts='rw')]
>>>
>>>psutil.disk_usage('/')  # 使用 psutil.disk_usage 方法获取分区（参数）的使用情况
```

```
sdiskusage(total=15481577472, used=4008087552, free=10687057920,
percent=25.899999999999999)
>>>
>>>psutil.disk_io_counters()     # 使用 psutil.disk_io_counters 获取硬盘总的 IO 个数、
                                 # 读写信息
sdiskio(read_count=9424, write_count=35824, read_bytes=128006144, write_
bytes=204312576, read_time=72266, write_time=182485)
>>>
>>>psutil.disk_io_counters(perdisk=True)  #“perdisk=True”参数获取单个分区 IO 个数、
                                          # 读写信息
{'sda2': sdiskio(read_count=322, write_count=0, read_bytes=1445888, write_
bytes=0, read_time=445, write_time=0), 'sda3': sdiskio(read_count=618, write_
count=3, read_bytes=2855936, write_bytes=12288, read_time=871, write_time=155),
'sda1': sdiskio(read_count=8484, write_count=35821, read_bytes=123704320,
write_bytes=204300288, read_time=70950, write_time=182330)}
```

（4）网络信息

系统的网络信息与磁盘IO类似，涉及几个关键点，包括bytes_sent（发送字节数）、bytes_recv=28220119（接收字节数）、packets_sent=200978（发送数据包数）、packets_recv=212672（接收数据包数）等。这些网络信息使用psutil.net_io_counters()方法获取，具体见下面的操作例子：

```
>>>psutil.net_io_counters()     # 使用 psutil.net_io_counters 获取网络总的 IO 信息，默
                                # 认 pernic=False
snetio(bytes_sent=27098178, bytes_recv=28220119, packets_sent=200978, packets_
recv=212672, errin=0, errout=0, dropin=0, dropout=0)
>>>psutil.net_io_counters(pernic=True)   #pernic=True 输出每个网络接口的 IO 信息
{'lo': snetio(bytes_sent=26406824, bytes_recv=26406824, packets_sent=198526,
packets_recv=198526, errin=0, errout=0, dropin=0, dropout=0), 'eth0':
snetio(bytes_sent=694750, bytes_recv=1816743, packets_sent=2478, packets_
recv=14175, errin=0, errout=0, dropin=0, dropout=0)}
>>>
```

（5）其他系统信息

除了前面介绍的几个获取系统基本信息的方法，psutil模块还支持获取用户登录、开机时间等信息，具体见下面的操作例子：

```
>>>psutil.users()     # 使用 psutil.users 方法返回当前登录系统的用户信息
[suser(name='root', terminal='pts/0', host='192.168.1.103',
started=1394638720.0), suser(name='root', terminal='pts/1',
host='192.168.1.103', started=1394723840.0)]
>>> import psutil, datetime
>>>psutil.boot_time()     # 使用 psutil.boot_time 方法获取开机时间，以 Linux 时间戳格式返回
1389563460.0
>>>datetime.datetime.fromtimestamp(psutil.boot_time()).strftime("%Y-%m-%d
%H:%M:%S")
```

```
'2014-01-12 22:51:00'     # 转换成自然时间格式
```

1.1.2 系统进程管理方法

获得当前系统的进程信息，可以让运维人员得知应用程序的运行状态，包括进程的启动时间、查看或设置 CPU 亲和度、内存使用率、IO 信息、socket 连接、线程数等，这些信息可以呈现出指定进程是否存活、资源利用情况，为开发人员的代码优化、问题定位提供很好的数据参考。

（1）进程信息

psutil 模块在获取进程信息方面也提供了很好的支持，包括使用 psutil.pids() 方法获取所有进程 PID，使用 psutil.Process() 方法获取单个进程的名称、路径、状态、系统资源利用率等信息，具体见下面的操作例子：

```
>>> import psutil
>>>psutil.pids()    # 列出所有进程 PID
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19……]
>>> p = psutil.Process(2424)  # 实例化一个 Process 对象，参数为一进程 PID
>>> p.name()  # 进程名
'java'
>>> p.exe()  # 进程 bin 路径
'/usr/java/jdk1.6.0_45/bin/java'
>>>p.cwd()  # 进程工作目录绝对路径
'/usr/local/hadoop-1.2.1'
>>>p.status()     # 进程状态
'sleeping'
>>>p.create_time()     # 进程创建时间，时间戳格式
1394852592.6900001
>>>p.uids()     # 进程 uid 信息
puids(real=0, effective=0, saved=0)
>>>p.gids()     # 进程 gid 信息
pgids(real=0, effective=0, saved=0)
>>>p.cpu_times()     # 进程 CPU 时间信息，包括 user、system 两个 CPU 时间
pcputimes(user=9.0500000000000007, system=20.25)
>>>p.cpu_affinity()     #get 进程 CPU 亲和度，如要设置进程 CPU 亲和度，将 CPU 号作为参数即可
[0, 1]
>>>p.memory_percent()     # 进程内存利用率
14.147714861289776
>>>p.memory_info()     # 进程内存 rss、vms 信息
pmem(rss=71626752, vms=1575665664)
>>>p.io_counters()     # 进程 IO 信息，包括读写 IO 数及字节数
pio(read_count=41133, write_count=16811, read_bytes=37023744, write_
bytes=4722688)
>>>p.connections()     # 返回打开进程 socket 的 namedutples 列表，包括 fs、family、laddr
                       # 等信息
[pconn(fd=65, family=10, type=1, laddr=('::ffff:192.168.1.20', 9000),
```

```
raddr=(),……]
>>>p.num_threads()     # 进程开启的线程数
33
```

（2）popen 类的使用

psutil 提供的 popen 类的作用是获取用户启动的应用程序进程信息，以便跟踪程序进程的运行状态。具体实现方法如下：

```
>>> import psutil
>>>from subprocess import PIPE
# 通过 psutil 的 Popen 方法启动的应用程序，可以跟踪该程序运行的所有相关信息
>>> p = psutil.Popen(["/usr/bin/python", "-c", "print('hello')"], stdout=PIPE)
>>>p.name()
'python'
>>>p.username()
'root'
>>>p.communicate()
('hello\n', None)
>>>p.cpu_times()     # 得到进程运行的 CPU 时间，更多方法见上一小节
pcputimes(user=0.01, system=0.040000000000000001)
```

参考提示
- ❑ 1.1.1 节示例参考 https://github.com/giampaolo/psutil。
- ❑ 1.1.1 节模块说明参考官网 http://psutil.readthedocs.org/en/latest/。

1.2　实用的 IP 地址处理模块 IPy

IP 地址规划是网络设计中非常重要的一个环节，规划的好坏会直接影响路由协议算法的效率，包括网络性能、可扩展性等方面，在这个过程当中，免不了要计算大量的 IP 地址，包括网段、网络掩码、广播地址、子网数、IP 类型等。Python 提供了一个强大的第三方模块 IPy（https://github.com/haypo/python-ipy/），最新版本为 V0.81。IPy 模块可以很好地辅助我们高效完成 IP 的规划工作，下面进行详细介绍。

以下是 IPy 模块的安装，这里采用源码的安装方式：

```
# wget https://pypi.python.org/packages/source/I/IPy/IPy-0.81.tar.gz --no-
check-certificate
# tar -zxvf IPy-0.81.tar.gz
# cd IPy-0.81
# python setup.py install
```

1.2.1 IP 地址、网段的基本处理

IPy 模块包含 IP 类，使用它可以方便处理绝大部分格式为 IPv6 及 IPv4 的网络和地址。比如通过 version 方法就可以区分出 IPv4 与 IPv6，如：

```
>>>IP('10.0.0.0/8').version()
4    #4 代表 IPv4 类型
>>>IP('::1').version()
6    #6 代表 IPv6 类型
```

通过指定的网段输出该网段的 IP 个数及所有 IP 地址清单，代码如下：

```
from IPy import IP
ip = IP('192.168.0.0/16')
print ip.len()     # 输出 192.168.0.0/16 网段的 IP 个数
for x in ip:    # 输出 192.168.0.0/16 网段的所有 IP 清单
    print(x)
```

执行结果如下：

```
65536
192.168.0.0
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.7
192.168.0.8
……
```

下面介绍 IP 类几个常见的方法，包括反向解析名称、IP 类型、IP 转换等。

```
>>>from IPy import IP
>>>ip = IP('192.168.1.20')
>>>ip.reverseNames()           # 反向解析地址格式
['20.1.168.192.in-addr.arpa.']
>>>ip.iptype()    #192.168.1.20 为私网类型 'PRIVATE'
>>> IP('8.8.8.8').iptype()    #8.8.8.8 为公网类型
'PUBLIC'
>>> IP("8.8.8.8").int()       # 转换成整型格式
134744072
>>> IP('8.8.8.8').strHex()    # 转换成十六进制格式
'0x8080808'
>>> IP('8.8.8.8').strBin()    # 转换成二进制格式
'00001000000010000000100000001000'
>>> print(IP(0x8080808))      # 十六进制转成 IP 格式
8.8.8.8
```

IP 方法也支持网络地址的转换，例如根据 IP 与掩码生产网段格式，如下：

```
>>>from IPy import IP
>>>print(IP('192.168.1.0').make_net('255.255.255.0'))
192.168.1.0/24
>>>print(IP('192.168.1.0/255.255.255.0', make_net=True))
192.168.1.0/24
>>>print(IP('192.168.1.0-192.168.1.255', make_net=True))
192.168.1.0/24
```

也可以通过 strNormal 方法指定不同 wantprefixlen 参数值以定制不同输出类型的网段。输出类型为字符串，如下：

```
>>>IP('192.168.1.0/24').strNormal(0)
'192.168.1.0'
>>>IP('192.168.1.0/24').strNormal(1)
'192.168.1.0/24'
>>>IP('192.168.1.0/24').strNormal(2)
'192.168.1.0/255.255.255.0'
>>>IP('192.168.1.0/24').strNormal(3)
'192.168.1.0-192.168.1.255'
```

wantprefixlen 的取值及含义：

- ❑ wantprefixlen = 0，无返回，如 192.168.1.0；
- ❑ wantprefixlen = 1，prefix 格式，如 192.168.1.0/24；
- ❑ wantprefixlen = 2，decimalnetmask 格式，如 192.168.1.0/255.255.255.0；
- ❑ wantprefixlen = 3，lastIP 格式，如 192.168.1.0-192.168.1.255。

1.2.2　多网络计算方法详解

有时候我们想比较两个网段是否存在包含、重叠等关系，比如同网络但不同 prefixlen 会认为是不相等的网段，如 10.0.0.0/16 不等于 10.0.0.0/24，另外即使具有相同的 prefixlen 但处于不同的网络地址，同样也视为不相等，如 10.0.0.0/16 不等于 192.0.0.0/16。IPy 支持类似于数值型数据的比较，以帮助 IP 对象进行比较，如：

```
>>>IP('10.0.0.0/24') < IP('12.0.0.0/24')
True
```

判断 IP 地址和网段是否包含于另一个网段中，如下：

```
>>> '192.168.1.100' in IP('192.168.1.0/24')
True
>>>IP('192.168.1.0/24') in IP('192.168.0.0/16')
True
```

判断两个网段是否存在重叠，采用 IPy 提供的 overlaps 方法，如：

```
>>>IP('192.168.0.0/23').overlaps('192.168.1.0/24')
1    # 返回 1 代表存在重叠
>>>IP('192.168.1.0/24').overlaps('192.168.2.0')
0    # 返回 0 代表不存在重叠
```

示例 根据输入的 IP 或子网返回网络、掩码、广播、反向解析、子网数、IP 类型等信息。

```
#!/usr/bin/env python
from IPy import IP

ip_s = raw_input('Please input an IP or net-range: ')     # 接收用户输入，参数为 IP
地址或网段地址
ips = IP(ip_s)
if len(ips) > 1:    # 为一个网络地址
    print('net: %s' % ips.net())    # 输出网络地址
    print('netmask: %s' % ips.netmask())    # 输出网络掩码地址
    print('broadcast: %s' % ips.broadcast())    # 输出网络广播地址
    print('reverse address: %s' % ips.reverseNames()[0])    # 输出地址反向解析
    print('subnet: %s' % len(ips))    # 输出网络子网数
else:    # 为单个 IP 地址
    print('reverse address: %s' % ips.reverseNames()[0])    # 输出 IP 反向解析

print('hexadecimal: %s' % ips.strHex())    # 输出十六进制地址
print('binary ip: %s' % ips.strBin())    # 输出二进制地址
print('iptype: %s' % ips.iptype())    # 输出地址类型，如 PRIVATE、PUBLIC、LOOPBACK 等
```

分别输入网段、IP 地址的运行返回结果如下：

```
# python simple1.py
Please input an IP or net-range: 192.168.1.0/24
net: 192.168.1.0
netmask: 255.255.255.0
broadcast: 192.168.1.255
reverse address: 1.168.192.in-addr.arpa.
subnet: 256
hexadecimal: 0xc0a80100
binaryip: 11000000101010000000000100000000
iptype: PRIVATE

# python simple1.py
Please input an IP or net-range: 192.168.1.20
reverse address: 20.1.168.192.in-addr.arpa.
hexadecimal: 0xc0a80114
binaryip: 11000000101010000000000100010100
iptype: PRIVATE
```

参考提示
- 1.2.1 节官网文档与示例参考 https://github.com/haypo/python-ipy/。
- 1.2.2 节示例1参考 http://blog.philippklaus.de/2012/12/ip-address-analysis-using-python/ 和 http://www.sourcecodebrowser.com/ipy/0.62/class_i_py_1_1_i_pint.html 等文章的 IPy 类说明。

1.3　DNS 处理模块 dnspython

dnspython（http://www.dnspython.org/）是 Python 实现的一个 DNS 工具包，它支持几乎所有的记录类型，可以用于查询、传输并动态更新 ZONE 信息，同时支持 TSIG（事务签名）验证消息和 EDNS0（扩展 DNS）。在系统管理方面，我们可以利用其查询功能来实现 DNS 服务监控以及解析结果的校验，可以代替 nslookup 及 dig 等工具，轻松做到与现有平台的整合，下面进行详细介绍。

首先介绍 dnspython 模块的安装，这里采用源码的安装方式，最新版本为 1.9.4，如下：

```
# http://www.dnspython.org/kits/1.9.4/dnspython-1.9.4.tar.gz
# tar -zxvf dnspython-1.9.4.tar.gz
# cd dnspython-1.9.4
# python setup.py install
```

1.3.1　模块域名解析方法详解

dnspython 模块提供了大量的 DNS 处理方法，最常用的方法是域名查询。dnspython 提供了一个 DNS 解析器类——resolver，使用它的 query 方法来实现域名的查询功能。query 方法的定义如下：

```
query(self, qname, rdtype=1, rdclass=1, tcp=False, source=None, raise_on_no_
answer=True, source_port=0)
```

其中，qname 参数为查询的域名。rdtype 参数用来指定 RR 资源的类型，常用的有以下几种：

- A 记录，将主机名转换成 IP 地址；
- MX 记录，邮件交换记录，定义邮件服务器的域名；
- CNAME 记录，指别名记录，实现域名间的映射；
- NS 记录，标记区域的域名服务器及授权子域；
- PTR 记录，反向解析，与 A 记录相反，将 IP 转换成主机名；
- SOA 记录，SOA 标记，一个起始授权区的定义。

rdclass 参数用于指定网络类型，可选的值有 IN、CH 与 HS，其中 IN 为默认，使用最广泛。tcp 参数用于指定查询是否启用 TCP 协议，默认为 False（不启用）。source 与 source_port 参数作为指定查询源地址与端口，默认值为查询设备 IP 地址和 0。raise_on_no_answer 参数用于指定当查询无应答时是否触发异常，默认为 True。

1.3.2 常见解析类型示例说明

常见的 DNS 解析类型包括 A、MX、NS、CNAME 等。利用 dnspython 的 dns.resolver.query 方法可以简单实现这些 DNS 类型的查询，为后面要实现的功能提供数据来源，比如对一个使用 DNS 轮循业务的域名进行可用性监控，需要得到当前的解析结果。下面一一进行介绍。

（1）A 记录

实现 A 记录查询方法源码。

【 /home/test/dnspython/simple1.py 】

```
#!/usr/bin/env python
import dns.resolver

domain = raw_input('Please input an domain: ')     # 输入域名地址
A = dns.resolver.query(domain, 'A')     # 指定查询类型为 A 记录
for i in A.response.answer:     # 通过 response.answer 方法获取查询回应信息
    for j in i.items:     # 遍历回应信息
printj.address
```

运行代码查看结果，这里以 www.google.com 域名为例：

```
# python simple1.py
Please input an domain: www.google.com
173.194.127.180
173.194.127.178
173.194.127.176
173.194.127.179
173.194.127.177
```

（2）MX 记录

实现 MX 记录查询方法源码。

【 /home/test/dnspython/ simple2.py 】

```
#!/usr/bin/env python
import dns.resolver
```

```
domain = raw_input('Please input an domain: ')
MX = dns.resolver.query(domain, 'MX')     # 指定查询类型为 MX 记录
for i in MX:     # 遍历回应结果，输出 MX 记录的 preference 及 exchanger 信息
   print 'MX preference =', i.preference, 'mail exchanger =', i.exchange
```

运行代码查看结果，这里以 163.com 域名为例：

```
# python simple2.py
Please input an domain: 163.com
MX preference = 10 mail exchanger = 163mx03.mxmail.netease.com.
MX preference = 50 mail exchanger = 163mx00.mxmail.netease.com.
MX preference = 10 mail exchanger = 163mx01.mxmail.netease.com.
MX preference = 10 mail exchanger = 163mx02.mxmail.netease.com.
```

（3）NS 记录

实现 NS 记录查询方法源码。

【/home/test/dnspython/ simple3.py】

```
#!/usr/bin/env python
import dns.resolver

domain = raw_input('Please input an domain: ')
ns = dns.resolver.query(domain, 'NS')     # 指定查询类型为 NS 记录
for i in ns.response.answer:
   for j in i.items:
      print j.to_text()
```

只限输入一级域名，如 baidu.com。如果输入二级或多级域名，如 www.baidu.com，则是错误的。

```
# python simple3.py
Please input an domain: baidu.com
ns4.baidu.com.
dns.baidu.com.
ns2.baidu.com.
ns7.baidu.com.
ns3.baidu.com.
```

（4）CNAME 记录

实现 CNAME 记录查询方法源码。

【/home/test/dnspython/ simple4.py】

```
#!/usr/bin/env python
import dns.resolver
```

```
domain = raw_input('Please input an domain: ')
cname = dns.resolver.query(domain, 'CNAME')    # 指定查询类型为 CNAME 记录
for i in cname.response.answer:    # 结果将回应 cname 后的目标域名
   for j in i.items:
      print j.to_text()
```

结果将返回 cname 后的目标域名。

1.3.3 实践：DNS 域名轮循业务监控

大部分的 DNS 解析都是一个域名对应一个 IP 地址，但是通过 DNS 轮循技术可以做到一个域名对应多个 IP，从而实现最简单且高效的负载平衡，不过此方案最大的弊端是目标主机不可用时无法被自动剔除，因此做好业务主机的服务可用监控至关重要。本示例通过分析当前域名的解析 IP，再结合服务端口探测来实现自动监控，在域名解析中添加、删除 IP 时，无须对监控脚本进行更改。实现架构图如图 1-1 所示。

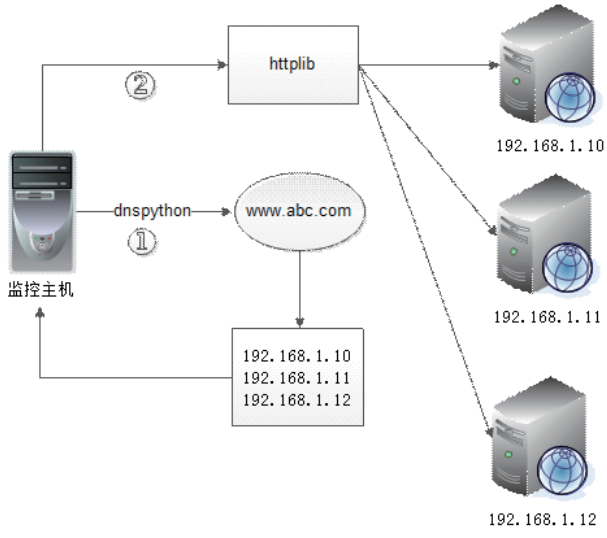


图 1-1 DNS 多域名业务服务监控架构图

1. 步骤

1）实现域名的解析，获取域名所有的 A 记录解析 IP 列表；

2）对 IP 列表进行 HTTP 级别的探测。

2. 代码解析

本示例第一步通过 dns.resolver.query() 方法获取业务域名 A 记录信息，查询出所有 IP 地

址列表，再使用 httplib 模块的 request() 方法以 GET 方式请求监控页面，监控业务所有服务的 IP 是否服务正常。

【 /home/test/dnspython/simple5.py 】

```
#!/usr/bin/python
import dns.resolver
import os
import httplib

iplist=[]    #定义域名 IP 列表变量
appdomain="www.google.com.hk"    #定义业务域名

def get_iplist(domain=""):    #域名解析函数，解析成功 IP 将被追加到 iplist
    try:
        A = dns.resolver.query(domain, 'A')    #解析 A 记录类型
    except Exception,e:
        print "dns resolver error:"+str(e)
        return
    for i in A.response.answer:
        for j in i.items:
            iplist.append(j.address)    #追加到 iplist
    return True

def checkip(ip):
    checkurl=ip+":80"
    getcontent=""
    httplib.socket.setdefaulttimeout(5)    #定义 http 连接超时时间（5 秒）
    conn=httplib.HTTPConnection(checkurl)    #创建 http 连接对象

    try:
        conn.request("GET", "/",headers = {"Host": appdomain})  #发起 URL 请求，添
                                                                #加 host 主机头
        r=conn.getresponse()
        getcontent =r.read(15)   #获取 URL 页面前 15 个字符，以便做可用性校验
    finally:
        if getcontent=="<!doctype html>":  #监控 URL 页的内容一般是事先定义好的，比如
                                           #“HTTP200”等
            print ip+" [OK]"
        else:
            print ip+" [Error]"    #此处可放告警程序，可以是邮件、短信通知

if __name__=="__main__":
    if get_iplist(appdomain) and len(iplist)>0:    #条件：域名解析正确且至少返回一个 IP
        for ip in iplist:
            checkip(ip)
    else:
        print "dns resolver error."
```

我们可以将此脚本放到 crontab 中定时运行，再结合告警程序，这样一个基于域名轮循

的业务监控已完成。运行程序，显示结果如下：

```
# python simple5.py
74.125.31.94 [OK]
74.125.128.199 [OK]
173.194.72.94 [OK]
```

从结果可以看出，域名 www.google.com.hk 解析出 3 个 IP 地址，并且服务都是正常的。

第 2 章 Chapter 2

业务服务监控详解

业务服务监控是运维体系中最重要的环节，是保证业务服务质量的关键手段。如何更有效地实现业务服务，是每个运维人员应该思考的问题，不同业务场景需定制不同的监控策略。Python 在监控方面提供了大量的第三方工具，可以帮助我们快速、有效地开发企业级服务监控平台，为我们的业务保驾护航。本章涉及文件与目录差异对比方法、HTTP 质量监控、邮件告警等内容。

2.1 文件内容差异对比方法

本节介绍如何通过 difflib 模块实现文件内容差异对比。difflib 作为 Python 的标准库模块，无需安装，作用是对比文本之间的差异，且支持输出可读性比较强的 HTML 文档，与 Linux 下的 diff 命令相似。我们可以使用 difflib 对比代码、配置文件的差别，在版本控制方面是非常有用。Python 2.3 或更高版本默认自带 difflib 模块，无需额外安装，我们先通过一个简单的示例进行了解。

2.1.1 示例 1：两个字符串的差异对比

本示例通过使用 difflib 模块实现两个字符串的差异对比，然后以版本控制风格进行输出。

【 /home/test/difflib/simple1.py 】

```
#!/usr/bin/python
```

```
import difflib
text1 = """text1:    # 定义字符串 1
This module provides classes and functions for comparing sequences.
including HTML and context and unified diffs.
difflib document v7.4
add string
"""
text1_lines = text1.splitlines()     # 以行进行分隔，以便进行对比
text2 = """text2:    # 定义字符串 2
This module provides classes and functions for Comparing sequences.
including HTML and context and unified diffs.
difflib document v7.5"""
text2_lines = text2.splitlines()
d = difflib.Differ()     # 创建 Differ() 对象
diff = d.compare(text1_lines, text2_lines)     # 采用 compare 方法对字符串进行比较
print '\n'.join(list(diff))
```

本示例采用 Differ() 类对两个字符串进行比较，另外 difflib 的 SequenceMatcher() 类支持任意类型序列的比较，HtmlDiff() 类支持将比较结果输出为 HTML 格式，示例运行结果如图 2-1 所示。

```
[root@SN2013-08-020 difflib]# python simple1.py
- text1:
?     ^

+ text2:
?     ^

- This module provides classes and functions for comparing sequences.
?                                                 ^

+ This module provides classes and functions for Comparing sequences.
?                                                 ^

  including HTML and context and unified diffs.
- difflib document v7.4
?                    ^

+ difflib document v7.5
?                    ^

- add string
```

图 2-1 示例运行结果

为方便大家理解差异关系符号，表 2-1 对各符号含义进行说明。

表 2-1　符号含义说明

符号	含义
'-'	包含在第一个序列行中，但不包含在第二个序列行
'+'	包含在第二个序列行中，但不包含在第一个序列行
' '	两个序列行一致
'?'	标志两个序列行存在增量差异
'^'	标志出两个序列行存在的差异字符

2.1.2　生成美观的对比 HTML 格式文档

采用 HtmlDiff() 类的 make_file() 方法就可以生成美观的 HTML 文档，对示例 1 中代码按以下进行修改：

```
d = difflib.Differ()
diff = d.compare(text1_lines, text2_lines)
print '\n'.join(list(diff))
```

替换成：

```
d = difflib.HtmlDiff()
print d.make_file(text1_lines, text2_lines)
```

将新文件命名为 simple2.py，运行 # python simple2.py > diff.html，再使用浏览器打开 diff.html 文件，结果如图示 2-2 所示，HTML 文档包括了行号、差异标志、图例等信息，可读性增强了许多。

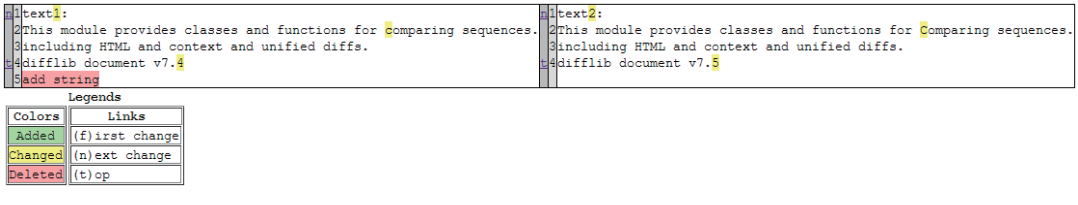


图 2-2　在浏览器中打开 diff.html 文件

2.1.3　示例 2：对比 Nginx 配置文件差异

当我们维护多个 Nginx 配置时，时常会对比不同版本配置文件的差异，使运维人员更加清晰地了解不同版本迭代后的更新项，实现的思路是读取两个需对比的配置文件，再以换行符作为分隔符，调用 difflib.HtmlDiff() 生成 HTML 格式的差异文档。实现代码如下：

【 /home/test/difflib/simple3.py 】

```
#!/usr/bin/python
import difflib
import sys

try:
    textfile1=sys.argv[1]     # 第一个配置文件路径参数
    textfile2=sys.argv[2]     # 第二个配置文件路径参数
except Exception,e:
    print "Error:"+str(e)
    print "Usage: simple3.py filename1 filename2"
    sys.exit()

def readfile(filename):     # 文件读取分隔函数
    try:
        fileHandle = open (filename, 'rb' )
        text=fileHandle.read().splitlines()     # 读取后以行进行分隔
        fileHandle.close()
        return text
    except IOError as error:
       print('Read file Error:'+str(error))
       sys.exit()

if textfile1=="" or textfile2=="":
    print "Usage: simple3.py filename1 filename2"
    sys.exit()

text1_lines = readfile(textfile1)     # 调用 readfile 函数，获取分隔后的字符串
text2_lines = readfile(textfile2)

d = difflib.HtmlDiff()     # 创建 HtmlDiff() 类对象
print d.make_file(text1_lines, text2_lines)     # 通过 make_file 方法输出 HTML 格式的比对结果
```

运行如下代码：

```
# python simple3.py nginx.conf.v1 nginx.conf.v2 > diff.html
```

从图 2-3 中可以看出 nginx.conf.v1 与 nginx.conf.v2 配置文件存在的差异。

参考提示 2.1 节示例参考官网文档 http://docs.python.org/2/library/difflib.html。

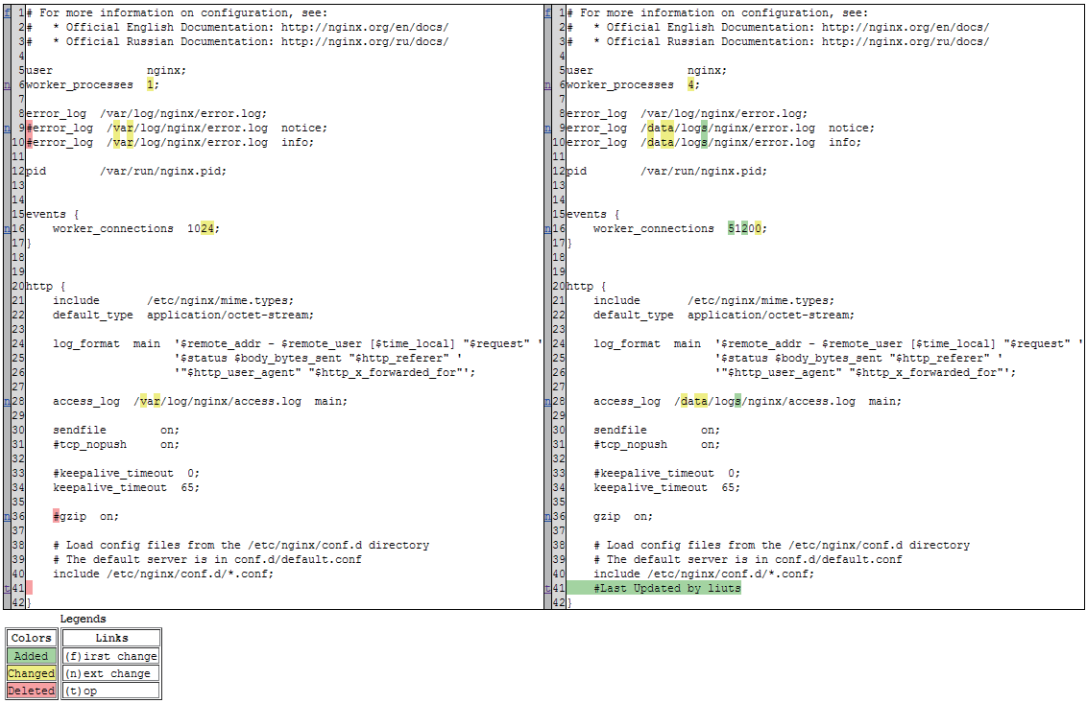


图 2-3　nginx.conf.v1 与 nginx.conf.v2 配置文件对比结果

2.2　文件与目录差异对比方法

当我们进行代码审计或校验备份结果时，往往需要检查原始与目标目录的文件一致性，Python 的标准库已经自带了满足此需求的模块 filecmp。filecmp 可以实现文件、目录、遍历子目录的差异对比功能。比如报告中输出目标目录比原始多出的文件或子目录，即使文件同名也会判断是否为同一个文件（内容级对比）等，Python 2.3 或更高版本默认自带 filecmp 模块，无需额外安装，下面进行详细介绍。

2.2.1　模块常用方法说明

filecmp 提供了三个操作方法，分别为 cmp（单文件对比）、cmpfiles（多文件对比）、dircmp（目录对比），下面逐一进行介绍：

- **单文件对比，**采用 filecmp.cmp(f1, f2[, shallow]) 方法，比较文件名为 f1 和 f2 的文件，相同返回 True，不相同返回 False，shallow 默认为 True，意思是只根据 os.stat() 方法返回的文件基本信息进行对比，比如最后访问时间、修改时间、状态改变时间等，会

忽略文件内容的对比。当 shallow 为 False 时，则 os.stat() 与文件内容同时进行校验。

示例：比较单文件的差异。

```
>>> filecmp.cmp("/home/test/filecmp/f1","/home/test/filecmp/f3")
True
>>> filecmp.cmp("/home/test/filecmp/f1","/home/test/filecmp/f2")
False
```

- **多文件对比，**采用 filecmp.cmpfiles(dir1, dir2, common[, shallow]) 方法，对比 dir1 与 dir2 目录给定的文件清单。该方法返回文件名的三个列表，分别为匹配、不匹配、错误。匹配为包含匹配的文件的列表，不匹配反之，错误列表包括了目录不存在文件、不具备读权限或其他原因导致的不能比较的文件清单。

示例：dir1 与 dir2 目录中指定文件清单对比。

两目录下文件的 md5 信息如下，其中 f1、f2 文件匹配；f3 不匹配；f4、f5 对应目录中不存在，无法比较。

```
[root@SN2013-08-020 dir2]# md5sum *
d9dfc198c249bb4ac341198a752b9458  f1
aa9aa0cac0ffc655ce9232e720bf1b9f  f2
33d2119b71f717ef4b981e9364530a39  f3
d9dfc198c249bb4ac341198a752b9458  f5
 [root@SN2013-08-020 dir1]# md5sum *
d9dfc198c249bb4ac341198a752b9458  f1
aa9aa0cac0ffc655ce9232e720bf1b9f  f2
d9dfc198c249bb4ac341198a752b9458  f3
410d6a485bcf5d2d2d223f2ada9b9c52  f4
```

使用 cmpfiles 对比的结果如下，符合我们的预期。

```
>>>filecmp.cmpfiles("/home/test/filecmp/dir1","/home/test/filecmp/dir2",['f1','f2',
'f3','f4','f5'])
(['f1', 'f2'], ['f3'], ['f4', 'f5'])
```

- **目录对比，**通过 dircmp(a, b[, ignore[, hide]]) 类创建一个目录比较对象，其中 a 和 b 是参加比较的目录名。ignore 代表文件名忽略的列表，并默认为 ['RCS', 'CVS', 'tags'] ；hide 代表隐藏的列表，默认为 [os.curdir，os.pardir]。dircmp 类可以获得目录比较的详细信息，如只有在 a 目录中包括的文件、a 与 b 都存在的子目录、匹配的文件等，同时支持递归。

dircmp 提供了三个输出报告的方法：

- report()，比较当前指定目录中的内容；
- report_partial_closure()，比较当前指定目录及第一级子目录中的内容；

❑ report_full_closure()，递归比较所有指定目录的内容。

为输出更加详细的比较结果，dircmp 类还提供了以下属性：

❑ left，左目录，如类定义中的 a；
❑ right，右目录，如类定义中的 b；
❑ left_list，左目录中的文件及目录列表；
❑ right_list，右目录中的文件及目录列表；
❑ common，两边目录共同存在的文件或目录；
❑ left_only，只在左目录中的文件或目录；
❑ right_only，只在右目录中的文件或目录；
❑ common_dirs，两边目录都存在的子目录；
❑ common_files，两边目录都存在的子文件；
❑ common_funny，两边目录都存在的子目录（不同目录类型或 os.stat() 记录的错误）；
❑ same_files，匹配相同的文件；
❑ diff_files，不匹配的文件；
❑ funny_files，两边目录中都存在，但无法比较的文件；
❑ subdirs，将 common_dirs 目录名映射到新的 dircmp 对象，格式为字典类型。

示例：对比 dir1 与 dir2 的目录差异。

通过调用 dircmp() 方法实现目录差异对比功能，同时输出目录对比对象所有属性信息。

【 /home/test/filecmp/ simple1.py 】

```
import filecmp
a="/home/test/filecmp/dir1"     # 定义左目录
b="/home/test/filecmp/dir2"     # 定义右目录
dirobj=filecmp.dircmp(a,b,['test.py'])     # 目录比较，忽略 test.py 文件
# 输出对比结果数据报表，详细说明请参考 filecmp 类方法及属性信息
dirobj.report()
dirobj.report_partial_closure()
dirobj.report_full_closure()
print "left_list:"+ str(dirobj.left_list)
print "right_list:"+ str(dirobj.right_list)
print "common:"+ str(dirobj.common)
print "left_only:"+ str(dirobj.left_only)
print "right_only:"+ str(dirobj.right_only)
print "common_dirs:"+ str(dirobj.common_dirs)
print "common_files:"+ str(dirobj.common_files)
print "common_funny:"+ str(dirobj.common_funny)
print "same_file:"+ str(dirobj.same_files)
print "diff_files:"+ str(dirobj.diff_files)
print "funny_files:"+ str(dirobj.funny_files)
```

为方便理解，通过 tree 命令输出两个目录的树结构，如图 2-4 所示。

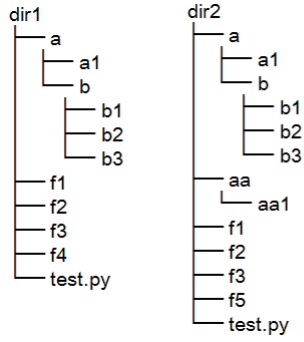


图 2-4 通过 tree 命令输出的两个目录

运行前面的代码并输出，结果如下：

```
# python simple1.py
-------------------report---------------------
diff /home/test/filecmp/dir1 /home/test/filecmp/dir2
Only in /home/test/filecmp/dir1 : ['f4']
Only in /home/test/filecmp/dir2 : ['aa', 'f5']
Identical files : ['f1', 'f2']
Differing files : ['f3']
Common subdirectories : ['a']
-------------report_partial_closure-----------
diff /home/test/filecmp/dir1 /home/test/filecmp/dir2
Only in /home/test/filecmp/dir1 : ['f4']
Only in /home/test/filecmp/dir2 : ['aa', 'f5']
Identical files : ['f1', 'f2']
Differing files : ['f3']
Common subdirectories : ['a']
diff /home/test/filecmp/dir1/a /home/test/filecmp/dir2/a
Identical files : ['a1']
Common subdirectories : ['b']
-------------report_full_closure--------------
diff /home/test/filecmp/dir1 /home/test/filecmp/dir2
Only in /home/test/filecmp/dir1 : ['f4']
Only in /home/test/filecmp/dir2 : ['aa', 'f5']
Identical files : ['f1', 'f2']
Differing files : ['f3']
Common subdirectories : ['a']
diff /home/test/filecmp/dir1/a /home/test/filecmp/dir2/a
Identical files : ['a1']
Common subdirectories : ['b']
diff /home/test/filecmp/dir1/a/b /home/test/filecmp/dir2/a/b
Identical files : ['b1', 'b2', 'b3']
left_list:['a', 'f1', 'f2', 'f3', 'f4']
```

```
right_list:['a', 'aa', 'f1', 'f2', 'f3', 'f5']
common:['a', 'f1', 'f2', 'f3']
left_only:['f4']
right_only:['aa', 'f5']
common_dirs:['a']
common_files:['f1', 'f2', 'f3']
common_funny:[]
same_file:['f1', 'f2']
diff_files:['f3']
funny_files:[]
```

2.2.2　实践：校验源与备份目录差异

有时候我们无法确认备份目录与源目录文件是否保持一致，包括源目录中的新文件或目录、更新文件或目录有无成功同步，定期进行校验，没有成功则希望有针对性地进行补备份。本示例使用了 filecmp 模块的 left_only、diff_files 方法递归获取源目录的更新项，再通过 shutil.copyfile、os.makedirs 方法对更新项进行复制，最终保持一致状态。详细源码如下：

【 /home/test/filecmp/simple2.py 】

```
#!/usr/bin/env python
import os, sys
import filecmp
import re
import shutil
holderlist=[]

def compareme(dir1, dir2):    # 递归获取更新项函数
    dircomp=filecmp.dircmp(dir1,dir2)
    only_in_one=dircomp.left_only    # 源目录新文件或目录
    diff_in_one=dircomp.diff_files    # 不匹配文件，源目录文件已发生变化
    dirpath=os.path.abspath(dir1)    # 定义源目录绝对路径
    # 将更新文件名或目录追加到 holderlist
    [holderlist.append(os.path.abspath(os.path.join(dir1,x))) for x in only_in_one]
    [holderlist.append(os.path.abspath(os.path.join(dir1,x))) for x in diff_in_one]
    if len(dircomp.common_dirs) > 0:    # 判断是否存在相同子目录，以便递归
        for item in dircomp.common_dirs:    # 递归子目录
            compareme(os.path.abspath(os.path.join(dir1,item)), \
            os.path.abspath(os.path.join(dir2,item)))
        return holderlist

def main():
    if len(sys.argv) > 2:    # 要求输入源目录与备份目录
        dir1=sys.argv[1]
        dir2=sys.argv[2]
    else:
        print "Usage: ", sys.argv[0], "datadir backupdir"
        sys.exit()
```

```
    source_files=compareme(dir1,dir2)     # 对比源目录与备份目录
    dir1=os.path.abspath(dir1)

    if not dir2.endswith('/'): dir2=dir2+'/'     # 备份目录路径加“/”符
    dir2=os.path.abspath(dir2)
    destination_files=[]
    createdir_bool=False

    for item in source_files:     # 遍历返回的差异文件或目录清单
        destination_dir=re.sub(dir1, dir2, item)     # 将源目录差异路径清单对应替换成
                                                     # 备份目录
        destination_files.append(destination_dir)
        if os.path.isdir(item):     # 如果差异路径为目录且不存在，则在备份目录中创建
            if not os.path.exists(destination_dir):
                os.makedirs(destination_dir)
                createdir_bool=True     # 再次调用 compareme 函数标记

    if createdir_bool:     # 重新调用 compareme 函数，重新遍历新创建目录的内容
        destination_files=[]
        source_files=[]
        source_files=compareme(dir1,dir2)     # 调用 compareme 函数
        for item in source_files:     # 获取源目录差异路径清单，对应替换成备份目录
            destination_dir=re.sub(dir1, dir2, item)
            destination_files.append(destination_dir)

    print "update item:"
    print source_files     # 输出更新项列表清单
    copy_pair=zip(source_files,destination_files)     # 将源目录与备份目录文件清单拆分成元组
    for item in copy_pair:
        if os.path.isfile(item[0]):     # 判断是否为文件，是则进行复制操作
            shutil.copyfile(item[0], item[1])

if __name__ == '__main__':
    main()
```

更新源目录 dir1 中的 f4、code/f3 文件后，运行程序结果如下：

```
# python simple2.py /home/test/filecmp/dir1 /home/test/filecmp/dir2
update item:
['/home/test/filecmp/dir1/f4', '/home/test/filecmp/dir1/code/f3']
# python simple2.py /home/test/filecmp/dir1 /home/test/filecmp/dir2
update item:
[]     # 再次运行时已经没有更新项了
```

- 2.2.1 节模块方法说明参考 http://docs.python.org/2/library/filecmp.html。
- 2.2.2 节示例参考 http://linuxfreelancer.com/how-do-you-compare-two-folders-and-copy-the-difference-to-a-third-folder。

2.3　发送电子邮件模块 smtplib

电子邮件是最流行的互联网应用之一。在系统管理领域，我们常常使用邮件来发送告警信息、业务质量报表等，方便运维人员第一时间了解业务的服务状态。本节通过 Python 的 smtplib 模块来实现邮件的发送功能，模拟一个 smtp 客户端，通过与 smtp 服务器交互来实现邮件发送的功能，这可以理解成 Foxmail 的发邮件功能，在第一次使用之前我们需要配置 smtp 主机地址、邮箱账号及密码等信息，Python 2.3 或更高版本默认自带 smtplib 模块，无需额外安装。下面详细进行介绍。

2.3.1　smtplib 模块的常用类与方法

SMTP 类定义：smtplib.SMTP([host[, port[, local_hostname[, timeout]]]])，作为 SMTP 的构造函数，功能是与 smtp 服务器建立连接，在连接成功后，就可以向服务器发送相关请求，比如登录、校验、发送、退出等。host 参数为远程 smtp 主机地址，比如 smtp.163.com ；port 为连接端口，默认为 25 ；local_hostname 的作用是在本地主机的 FQDN（完整的域名）发送 HELO/EHLO（标识用户身份）指令，timeout 为连接或尝试在多少秒超时。SMTP 类具有如下方法：

- SMTP.connect([host[, port]]) 方法，连接远程 smtp 主机方法，host 为远程主机地址，port 为远程主机 smtp 端口，默认 25，也可以直接使用 host:port 形式来表示，例如：SMTP.connect（“smtp.163.com”，“25”）。
- SMTP.login(user, password) 方法，远程 smtp 主机的校验方法，参数为用户名与密码，如 SMTP.login（“python_2014@163.com”，“sdjkg358”）。
- SMTP.sendmail(from_addr, to_addrs, msg[, mail_options, rcpt_options]) 方法，实现邮件的发送功能，参数依次为是发件人、收件人、邮件内容，例如：SMTP.sendmail（“python_2014@163.com”，“demo@domail.com”，body），其中 body 内容定义如下：

```
"""From: python_2014@163.com
To: demo@domail.com
Subject: test mail

test mail body"""
```

- SMTP.starttls([keyfile[, certfile]]) 方法，启用 TLS（安全传输）模式，所有 SMTP 指令都将加密传输，例如使用 gmail 的 smtp 服务时需要启动此项才能正常发送邮件，如 SMTP.starttls()。
- SMTP.quit() 方法，断开 smtp 服务器的连接。

下面通过一个简单示例帮助大家理解，目的是使用 gmail 向 QQ 邮箱发送测试邮件，代

码如下：

```
#!/usr/bin/python
import smtplib
import string

HOST = "smtp.gmail.com"     # 定义 smtp 主机
SUBJECT = "Test email from Python"     # 定义邮件主题
TO = "testmail@qq.com"     # 定义邮件收件人
FROM = "mymail@gmail.com"     # 定义邮件发件人
text = "Python rules them all!"     # 邮件内容
BODY = string.join((     # 组装 sendmail 方法的邮件主体内容，各段以 "\r\n" 进行分隔
        "From: %s" % FROM,
        "To: %s" % TO,
        "Subject: %s" % SUBJECT ,
        "",
        text
        ), "\r\n")
server = smtplib.SMTP()     # 创建一个 SMTP() 对象
server.connect(HOST,"25")     # 通过 connect 方法连接 smtp 主机
server.starttls()     # 启动安全传输模式
server.login("mymail@gmail.com","mypassword")     # 邮箱账号登录校验
server.sendmail(FROM, [TO], BODY)     # 邮件发送
server.quit()     # 断开 smtp 连接
```

我们将收到一封这样的邮件，如图 2-5 所示。

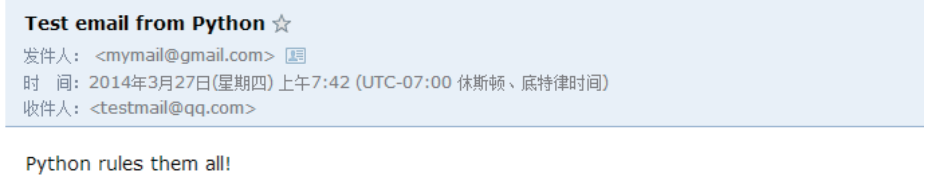


图 2-5 收到的邮件

2.3.2 定制个性化的邮件格式方法

通过邮件传输简单的文本已经无法满足我们的需求，比如我们时常会定制业务质量报表，在邮件主体中会包含 HTML、图像、声音以及附件格式等，MIME（Multipurpose Internet Mail Extensions，多用途互联网邮件扩展）作为一种新的扩展邮件格式很好地补充了这一点，更多 MIME 知识见 http://zh.wikipedia.org/wiki/MIME。下面介绍几个 Python 中常用的 MIME 实现类：

- email.mime.multipart.MIMEMultipart([_subtype[, boundary[, _subparts[, _params]]]])，作用是生成包含多个部分的邮件体的 MIME 对象，参数 _subtype 指定要添加到 "Content-type:multipart/subtype" 报头的可选的三种子类型，分别为 mixed、related、

alternative，默认值为 mixed。定义 mixed 实现构建一个带附件的邮件体；定义 related 实现构建内嵌资源的邮件体；定义 alternative 则实现构建纯文本与超文本共存的邮件体。

- email.mime.audio.MIMEAudio (_audiodata[, _subtype[, _encoder[, **_params]]])，创建包含音频数据的邮件体，_audiodata 包含原始二进制音频数据的字节字符串。
- email.mime.image.MIMEImage(_imagedata[, _subtype[, _encoder[, **_params]]])，创建包含图片数据的邮件体，_imagedata 是包含原始图片数据的字节字符串。
- email.mime.text.MIMEText (_text[, _subtype[, _charset]])，创建包含文本数据的邮件体，_text 是包含消息负载的字符串，_subtype 指定文本类型，支持 plain（默认值）或 html 类型的字符串。

2.3.3　定制常用邮件格式示例详解

前面两小节介绍了 Python 的 smtplib 及 email 模块的常用方法，那么两者在邮件定制到发送过程中是如何分工的？我们可以将 email.mime 理解成 smtplib 模块邮件内容主体的扩展，从原先默认只支持纯文本格式扩展到 HTML，同时支持附件、音频、图像等格式，smtplib 只负责邮件的投递即可。下面介绍在日常运营工作中邮件应用的几个示例。

示例 1：实现 HTML 格式的数据报表邮件。

纯文本的邮件内容已经不能满足我们多样化的需求，本示例通过引入 email.mime 的 MIMEText 类来实现支持 HTML 格式的邮件，支持所有 HTML 元素，包含表格、图片、动画、CSS 样式、表单等。本示例使用 HTML 的表格定制美观的业务流量报表，实现代码如下：

【 /home/test/smtplib/simple2.py 】

```
#coding: utf-8
import smtplib
from email.mime.text import MIMEText     # 导入 MIMEText 类

HOST = "smtp.gmail.com"     # 定义 smtp 主机
SUBJECT = u" 官网流量数据报表 "     # 定义邮件主题
TO = "testmail@qq.com"     # 定义邮件收件人
FROM = "mymail@gmail.com"     # 定义邮件发件人
msg = MIMEText("""     # 创建一个 MIMEText 对象，分别指定 HTML 内容、类型（文本或 html）、字
                       # 符编码
    <table width="800" border="0" cellspacing="0" cellpadding="4">
      <tr>
         <td bgcolor="#CECFAD" height="20" style="font-size:14px">* 官网数据   <a
href="monitor.domain.com"> 更多 >></a></td>
      </tr>
      <tr>
        <td bgcolor="#EFEBDE" height="100" style="font-size:13px">
```

```
        1）日访问量 :<font color=red>152433</font>  访问次数 :23651 页面浏览量 :45123
点击数 :545122  数据流量 :504Mb<br>
        2）状态码信息 <br>
          500:105  404:3264  503:214<br>
        3）访客浏览器信息 <br>
          IE:50%  firefox:10% chrome:30% other:10%<br>
        4）页面信息 <br>
          /index.php 42153<br>
          /view.php 21451<br>
          /login.php 5112<br>
        </td>
      </tr>
    </table>""","html","utf-8")
msg['Subject'] = SUBJECT     # 邮件主题
msg['From']=FROM     # 邮件发件人，邮件头部可见
msg['To']=TO     # 邮件收件人，邮件头部可见
try:
    server = smtplib.SMTP()     # 创建一个 SMTP() 对象
    server.connect(HOST,"25")     # 通过 connect 方法连接 smtp 主机
    server.starttls()     # 启动安全传输模式
    server.login("mymail@gmail.com","mypassword")     # 邮箱账号登录校验
    server.sendmail(FROM, TO, msg.as_string())     # 邮件发送
    server.quit()     # 断开 smtp 连接
    print "邮件发送成功！"
except Exception, e:
    print "失败: "+str(e)
```

代码运行结果如图 2-6 所示，我们将业务日志分析结果定期推送给管理员，以方便管理员了解业务的服务情况。

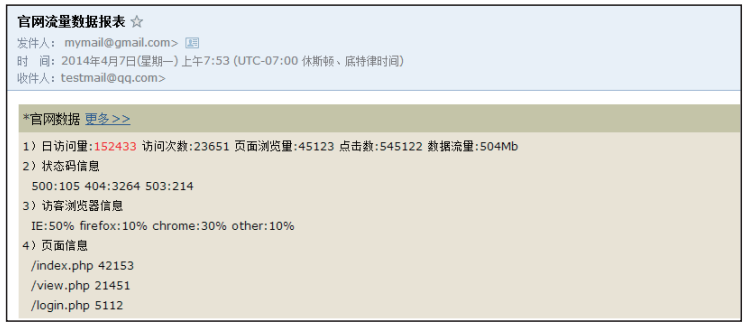


图 2-6　示例 1 运行结果

示例 2：实现图文格式的服务器性能报表邮件。

示例 1 通过 MIMEText 类来实现 HTML 格式的邮件，当要求包含图片数据的邮件内容时，需要引用 MIMEImage 类，若邮件主体由多个 MIME 对象组成，则同时需引用 MIMEMultipart 类来进行组装。本示例通过 MIMEText 与 MIMEImage 类的组合来实现图文格式的服务器性能报表邮件的定制，实现代码如下：

【 /home/test/smtplib/simple3.py 】

```
#coding: utf-8
import smtplib
from email.mime.multipart import MIMEMultipart     # 导入 MIMEMultipart 类
from email.mime.text import MIMEText     # 导入 MIMEText 类
from email.mime.image import MIMEImage     # 导入 MIMEImage 类

HOST = "smtp.gmail.com"     # 定义 smtp 主机
SUBJECT = u" 业务性能数据报表 "     # 定义邮件主题
TO = "testmail@qq.com"     # 定义邮件收件人
FROM = "mymail@gmail.com"     # 定义邮件发件人

def addimg(src,imgid):     # 添加图片函数，参数 1：图片路径，参数 2：图片 id
    fp = open(src, 'rb')     # 打开文件
    msgImage = MIMEImage(fp.read())     # 创建 MIMEImage 对象，读取图片内容并作为参数
    fp.close()     # 关闭文件
    msgImage.add_header('Content-ID', imgid)     # 指定图片文件的 Content-ID,<img>
                                                 # 标签 src 用到
    return msgImage     # 返回 msgImage 对象

msg = MIMEMultipart('related')     # 创建 MIMEMultipart 对象，采用 related 定义内嵌资源
                                   # 的邮件体
msgtext = MIMEText("""     # 创建一个 MIMEText 对象，HTML 元素包括表格 <table> 及图片 <img>
<table width="600" border="0" cellspacing="0" cellpadding="4">
      <tr bgcolor="#CECFAD" height="20" style="font-size:14px">
        <td colspan=2>* 官网性能数据  <a href="monitor.domain.com"> 更多 >></a></td>
      </tr>
      <tr bgcolor="#EFEBDE" height="100" style="font-size:13px">
        <td>
         <img src="cid:io"></td><td>
         <img src="cid:key_hit"></td>
      </tr>
      <tr bgcolor="#EFEBDE" height="100" style="font-size:13px">
         <td>
         <img src="cid:men"></td><td>
         <img src="cid:swap"></td>
      </tr>
    </table>""","html","utf-8")     #<img> 标签的 src 属性是通过 Content-ID 来引用的

msg.attach(msgtext)     #MIMEMultipart 对象附加 MIMEText 的内容
msg.attach(addimg("img/bytes_io.png","io"))     # 使用 MIMEMultipart 对象附加 MIMEImage
                                                # 的内容
msg.attach(addimg("img/myisam_key_hit.png","key_hit"))
msg.attach(addimg("img/os_mem.png","men"))
msg.attach(addimg("img/os_swap.png","swap"))
msg['Subject'] = SUBJECT     # 邮件主题
msg['From']=FROM     # 邮件发件人，邮件头部可见
msg['To']=TO     # 邮件收件人，邮件头部可见
try:
```

```
    server = smtplib.SMTP()     # 创建一个 SMTP() 对象
    server.connect(HOST,"25")     # 通过 connect 方法连接 smtp 主机
    server.starttls()     # 启动安全传输模式
    server.login("mymail@gmail.com","mypassword")     # 邮箱账号登录校验
    server.sendmail(FROM, TO, msg.as_string())     # 邮件发送
    server.quit()     # 断开 smtp 连接
    print "邮件发送成功！"
except Exception, e:
    print "失败: "+str(e)
```

代码运行结果如图 2-7 所示，我们将业务服务器性能数据定期推送给管理员，以方便管理员了解业务的服务情况。

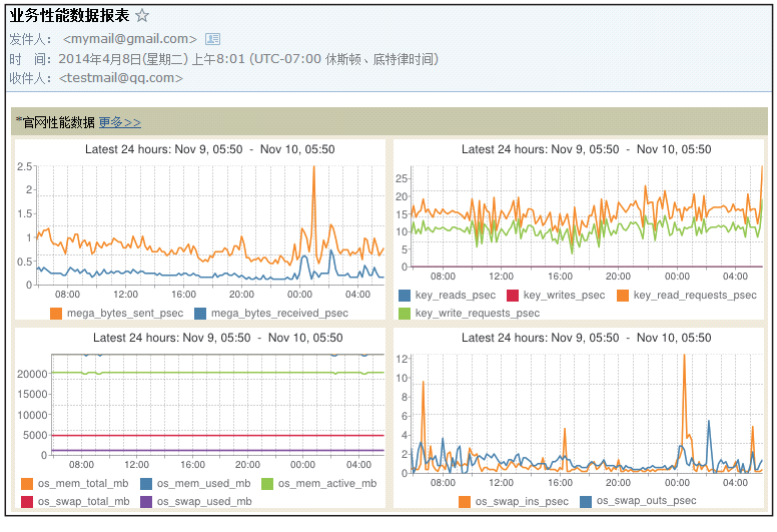


图 2-7 示例 2 运行结果

示例 3：实现带附件格式的业务服务质量周报邮件。

本示例通过 MIMEText 与 MIMEImage 类的组合，实现图文邮件格式。另通过 MIMEText 类再定义 Content-Disposition 属性来实现带附件的邮件。我们可以利用这些丰富的特性来定制周报邮件，如业务服务质量周报。实现代码如下：

【 /home/test/smtplib/simple4.py 】

```
#coding: utf-8
import smtplib
from email.mime.multipart import MIMEMultipart     # 导入 MIMEMultipart 类
from email.mime.text import MIMEText     # 导入 MIMEText 类
from email.mime.image import MIMEImage     # 导入 MIMEImage 类
HOST = "smtp.gmail.com"     # 定义 smtp 主机
```

```
SUBJECT = u"官网业务服务质量周报"     # 定义邮件主题
TO = "testmail@qq.com"     # 定义邮件接收人
FROM = "mymail@gmail.com"     # 定义邮件发件人

def addimg(src,imgid):     # 添加图片函数，参数 1：图片路径，参数 2：图片 id
    fp = open(src, 'rb')     # 打开文件
    msgImage = MIMEImage(fp.read())     # 创建 MIMEImage 对象，读取图片内容作为参数
    fp.close()     # 关闭文件
    msgImage.add_header('Content-ID', imgid)     # 指定图片文件的 Content-ID,<img>
                                                 # 标签 src 用到
    return msgImage     # 返回 msgImage 对象

msg = MIMEMultipart('related')     # 创建 MIMEMultipart 对象，采用 related 定义内嵌资源
                                   # 的邮件体
# 创建一个 MIMEText 对象，HTML 元素包括文字与图片 <img>
msgtext = MIMEText("<font color=red>官网业务周平均延时图表 :<br><img src=\"cid:weekly\"
border=\"1\"><br> 详细内容见附件。</font>","html","utf-8")
msg.attach(msgtext)     #MIMEMultipart 对象附加 MIMEText 的内容
msg.attach(addimg("img/weekly.png","weekly"))     # 使用 MIMEMultipart 对象附加
                                                  # MIMEImage 的内容

# 创建一个 MIMEText 对象，附加 week_report.xlsx 文档
attach = MIMEText(open("doc/week_report.xlsx", "rb").read(), "base64", "utf-8")
attach["Content-Type"] = "application/octet-stream"     # 指定文件格式类型
# 指定 Content-Disposition 值为 attachment 则出现下载保存对话框，保存的默认文件名使用
#filename 指定
# 由于 qqmail 使用 gb18030 页面编码，为保证中文文件名不出现乱码，对文件名进行编码转换
attach["Content-Disposition"] = "attachment; filename=\"业务服务质量周报 (12 周 ).xlsx\"".
decode("utf-8").encode("gb18030")

msg.attach(attach)     #MIMEMultipart 对象附加 MIMEText 附件内容
msg['Subject'] = SUBJECT     # 邮件主题
msg['From']=FROM     # 邮件发件人，邮件头部可见
msg['To']=TO     # 邮件收件人，邮件头部可见
try:
    server = smtplib.SMTP()     # 创建一个 SMTP() 对象
    server.connect(HOST,"25")     # 通过 connect 方法连接 smtp 主机
    server.starttls()     # 启动安全传输模式
    server.login("mymail@gmail.com","mypassword")     # 邮箱账号登录校验
    server.sendmail(FROM, TO, msg.as_string())     # 邮件发送
    server.quit()     # 断开 smtp 连接
    print "邮件发送成功！"
except Exception, e:
    print "失败："+str(e)
```

代码运行结果如图 2-8 所示，实现了发送业务服务质量周报的邮件功能。

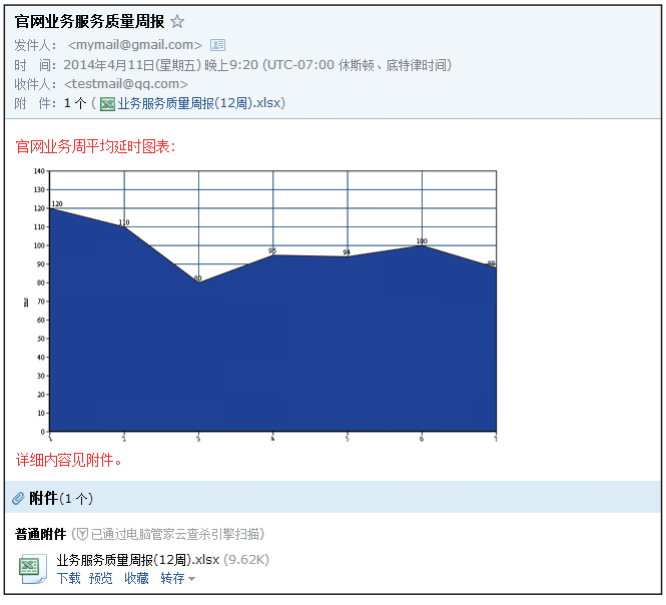


图 2-8 示例 3 的运行结果

参考提示

- 2.3.1 节 smtplib 模块的常用类与方法内容参考 https://docs.python.org/2.7/library/smtplib.html。
- 2.3.2 节 email.mime 常用类定义内容参考 https://docs.python.org/2.7/library/email.mime.html。

2.4 探测 Web 服务质量方法

pycurl（http://pycurl.sourceforge.net）是一个用 C 语言写的 libcurl Python 实现，功能非常强大，支持的操作协议有 FTP、HTTP、HTTPS、TELNET 等，可以理解成 Linux 下 curl 命令功能的 Python 封装，简单易用。本节通过调用 pycurl 提供的方法，实现探测 Web 服务质量的情况，比如响应的 HTTP 状态码、请求延时、HTTP 头信息、下载速度等，利用这些信息可以定位服务响应慢的具体环节，下面详细进行说明。

pycurl 模块的安装方法如下：

```
easy_install pycurl     #pip 安装方法
pip install pycurl     #easy_install 安装方法
```

```
# 源码安装方法
# 要求 curl-config 包支持，需要源码方式重新安装 curl
# wget http://curl.haxx.se/download/curl-7.36.0.tar.gz
# tar -zxvf curl-7.36.0.tar.gz
# cd curl-7.36.0
# ./configure
# make && make install
# export LD_LIBRARY_PATH=/usr/local/lib
#
# wget https://pypi.python.org/packages/source/p/pycurl/pycurl-7.19.3.1.tar.gz
--no-check-certificate
# tar -zxvf pycurl-7.19.3.1.tar.gz
# cd pycurl-7.19.3.1
# python setup.py install --curl-config=/usr/local/bin/curl-config
```

校验安装结果如下：

```
>>> import pycurl
>>> pycurl.version
'PycURL/7.19.3.1 libcurl/7.36.0 OpenSSL/1.0.1e zlib/1.2.3'
```

2.4.1　模块常用方法说明

pycurl.Curl() 类实现创建一个 libcurl 包的 Curl 句柄对象，无参数。更多关于 libcurl 包的介绍见 http://curl.haxx.se/libcurl/c/libcurl-tutorial.html。下面介绍 Curl 对象几个常用的方法。

- close() 方法，对应 libcurl 包中的 curl_easy_cleanup 方法，无参数，实现关闭、回收 Curl 对象。
- perform() 方法，对应 libcurl 包中的 curl_easy_perform 方法，无参数，实现 Curl 对象请求的提交。
- setopt(option, value) 方法，对应 libcurl 包中的 curl_easy_setopt 方法，参数 option 是通过 libcurl 的常量来指定的，参数 value 的值会依赖 option，可以是一个字符串、整型、长整型、文件对象、列表或函数等。下面列举常用的常量列表：

```
c = pycurl.Curl()     # 创建一个 curl 对象
c.setopt(pycurl.CONNECTTIMEOUT, 5)     # 连接的等待时间，设置为 0 则不等待
c.setopt(pycurl.TIMEOUT, 5)     # 请求超时时间
c.setopt(pycurl.NOPROGRESS, 0)     # 是否屏蔽下载进度条，非 0 则屏蔽
c.setopt(pycurl.MAXREDIRS, 5)     # 指定 HTTP 重定向的最大数
c.setopt(pycurl.FORBID_REUSE, 1)     # 完成交互后强制断开连接，不重用
c.setopt(pycurl.FRESH_CONNECT,1)     # 强制获取新的连接，即替代缓存中的连接
c.setopt(pycurl.DNS_CACHE_TIMEOUT,60)     # 设置保存 DNS 信息的时间，默认为 120 秒
c.setopt(pycurl.URL,"http://www.baidu.com")     # 指定请求的 URL
c.setopt(pycurl.USERAGENT,"Mozilla/5.2 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50324)")     # 配置请求 HTTP 头的 User-Agent
c.setopt(pycurl.HEADERFUNCTION, getheader)  # 将返回的 HTTP HEADER 定向到回调函数 getheader
```

```
c.setopt(pycurl.WRITEFUNCTION, getbody)     # 将返回的内容定向到回调函数 getbody
c.setopt(pycurl.WRITEHEADER, fileobj)      # 将返回的 HTTP HEADER 定向到 fileobj 文件对象
c.setopt(pycurl.WRITEDATA, fileobj)     # 将返回的 HTML 内容定向到 fileobj 文件对象
```

- getinfo(option) 方法，对应 libcurl 包中的 curl_easy_getinfo 方法，参数 option 是通过 libcurl 的常量来指定的。下面列举常用的常量列表：

```
c = pycurl.Curl()     # 创建一个 curl 对象
c.getinfo(pycurl.HTTP_CODE)     # 返回的 HTTP 状态码
c.getinfo(pycurl.TOTAL_TIME)     # 传输结束所消耗的总时间
c.getinfo(pycurl.NAMELOOKUP_TIME)     #DNS 解析所消耗的时间
c.getinfo(pycurl.CONNECT_TIME)     # 建立连接所消耗的时间
c.getinfo(pycurl.PRETRANSFER_TIME)     # 从建立连接到准备传输所消耗的时间
c.getinfo(pycurl.STARTTRANSFER_TIME)     # 从建立连接到传输开始消耗的时间
c.getinfo(pycurl.REDIRECT_TIME)     # 重定向所消耗的时间
c.getinfo(pycurl.SIZE_UPLOAD)     # 上传数据包大小
c.getinfo(pycurl.SIZE_DOWNLOAD)     # 下载数据包大小
c.getinfo(pycurl.SPEED_DOWNLOAD)     # 平均下载速度
c.getinfo(pycurl.SPEED_UPLOAD)     # 平均上传速度
c.getinfo(pycurl.HEADER_SIZE)     #HTTP 头部大小
```

我们利用 libcurl 包提供的这些常量值来达到探测 Web 服务质量的目的。

2.4.2 实践：实现探测 Web 服务质量

HTTP 服务是最流行的互联网应用之一，服务质量的好坏关系到用户体验以及网站的运营服务水平，最常用的有两个标准，一为服务的可用性，比如是否处于正常提供服务状态，而不是出现 404 页面未找到或 500 页面错误等；二为服务的响应速度，比如静态类文件下载时间都控制在毫秒级，动态 CGI 为秒级。本示例使用 pycurl 的 setopt 与 getinfo 方法实现 HTTP 服务质量的探测，获取监控 URL 返回的 HTTP 状态码，HTTP 状态码采用 pycurl. HTTP_CODE 常量得到，以及从 HTTP 请求到完成下载期间各环节的响应时间，通过 pycurl. NAMELOOKUP_TIME、pycurl. CONNECT_TIME、pycurl. PRETRANSFER_TIME、pycurl. R 等常量来实现。另外通过 pycurl.WRITEHEADER、pycurl.WRITEDATA 常量得到目标 URL 的 HTTP 响应头部及页面内容。实现源码如下：

【 /home/test/pycurl/simple1.py 】

```
# -*- coding: utf-8 -*-
import os,sys
import time
import sys
import pycurl

URL="http://www.google.com.hk"     # 探测的目标 URL
c = pycurl.Curl()     # 创建一个 Curl 对象
```

```
    c.setopt(pycurl.URL, URL)     # 定义请求的 URL 常量
    c.setopt(pycurl.CONNECTTIMEOUT, 5)     # 定义请求连接的等待时间
    c.setopt(pycurl.TIMEOUT, 5)     # 定义请求超时时间
    c.setopt(pycurl.NOPROGRESS, 1)     # 屏蔽下载进度条
    c.setopt(pycurl.FORBID_REUSE, 1)     # 完成交互后强制断开连接，不重用
    c.setopt(pycurl.MAXREDIRS, 1)     # 指定 HTTP 重定向的最大数为 1
    c.setopt(pycurl.DNS_CACHE_TIMEOUT,30)     # 设置保存 DNS 信息的时间为 30 秒
    # 创建一个文件对象，以”wb”方式打开，用来存储返回的 http 头部及页面内容
    indexfile = open(os.path.dirname(os.path.realpath(__file__))+"/content.txt",
"wb")
    c.setopt(pycurl.WRITEHEADER, indexfile)      # 将返回的 HTTP HEADER 定向到 indexfile 文件
对象
    c.setopt(pycurl.WRITEDATA, indexfile)     # 将返回的 HTML 内容定向到 indexfile 文件对象
    try:
        c.perform()     # 提交请求
    except Exception,e:
        print "connecion error:"+str(e)
        indexfile.close()
        c.close()
    sys.exit()

    NAMELOOKUP_TIME =  c.getinfo(c.NAMELOOKUP_TIME)     # 获取 DNS 解析时间
    CONNECT_TIME =  c.getinfo(c.CONNECT_TIME)     # 获取建立连接时间
    PRETRANSFER_TIME =   c.getinfo(c.PRETRANSFER_TIME)     # 获取从建立连接到准备传输所消
                                                           # 耗的时间
    STARTTRANSFER_TIME = c.getinfo(c.STARTTRANSFER_TIME)     # 获取从建立连接到传输开始消
                                                             # 耗的时间
    TOTAL_TIME = c.getinfo(c.TOTAL_TIME)     # 获取传输的总时间
    HTTP_CODE =  c.getinfo(c.HTTP_CODE)     # 获取 HTTP 状态码
    SIZE_DOWNLOAD =  c.getinfo(c.SIZE_DOWNLOAD)     # 获取下载数据包大小
    HEADER_SIZE = c.getinfo(c.HEADER_SIZE)     # 获取 HTTP 头部大小
    SPEED_DOWNLOAD=c.getinfo(c.SPEED_DOWNLOAD)     # 获取平均下载速度
    # 打印输出相关数据
    print "HTTP 状态码：%s" %(HTTP_CODE)
    print "DNS 解析时间：%.2f ms"%(NAMELOOKUP_TIME*1000)
    print "建立连接时间：%.2f ms" %(CONNECT_TIME*1000)
    print "准备传输时间：%.2f ms" %(PRETRANSFER_TIME*1000)
    print "传输开始时间：%.2f ms" %(STARTTRANSFER_TIME*1000)
    print "传输结束总时间：%.2f ms" %(TOTAL_TIME*1000)
    print "下载数据包大小：%d bytes/s" %(SIZE_DOWNLOAD)
    print "HTTP 头部大小：%d byte" %(HEADER_SIZE)
    print "平均下载速度：%d bytes/s" %(SPEED_DOWNLOAD)
    # 关闭文件及 Curl 对象
    indexfile.close()
    c.close()
```

代码的执行结果如图 2-9 所示。

```
[root@SN2013-08-020 pycurl]# python simple1.py
HTTP状态码：200
DNS解析时间：113.18 ms
建立连接时间：300.70 ms
准备传输时间：301.06 ms
传输开始时间：507.36 ms
传输结束总时间：507.52 ms
下载数据包大小：12006 bytes/s
HTTP头部大小：798 byte
平均下载速度：23656 bytes/s
```

图 2-9 探测到的 Web 服务质量

查看获取的 HTTP 文件头部及页面内容文件 content.txt，如图 2-10 所示。

```
HTTP/1.1 200 OK^M
Date: Wed, 23 Apr 2014 15:19:04 GMT^M                    HTTP HEADER
Expires: -1^M
Cache-Control: private, max-age=0^M
Content-Type: text/html; charset=Big5^M
Set-Cookie: PREF=ID=e2c1021e3b4d36e8:FF=0:NW=1:TM=1398266344:LM=1398266344:S=X1ev4S
Set-Cookie: NID=67=P9W3T5im7sfXTfZVXP9mOSQq9SB3MlQqtA6SnLxh_4bbdN6iY3Q2vKOciXTmhaG7u
5:19:04 GMT; path=/; domain=.google.com.hk; HttpOnly^M
P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/an
Server: gws^M
X-XSS-Protection: 1; mode=block^M
X-Frame-Options: SAMEORIGIN^M
Alternate-Protocol: 80:quic^M
Transfer-Encoding: chunked^M
^M
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="zh-HK">
<script>(function(){                                     HTTP BODY
window.google={kEI:"6NlXU-TQEMy6kAXeu4CoCA",getEI:function(a){for(var b;a&&(!a.getA
{return"https:"==window.location.protocol},kEXPI:"17259,4000116,4007661,4007830,4008
,4012373,4012504,4013374,4013414,4013591,4013723,4013758,4013787,4013823,4013967,401
9,4015155,4015234,4015260,4015342,4015519,4015550,4015635,4015638,4015639,4015772,40
25,4016466,4016479,4016487,4016623,4016703,4016730,4016767,4016800,4016824,4016851,4
177,4017201,4017205,4017261,4017336,8300015,8300017,8500165,8500223,8500240,8500252
```

图 2-10 content.txt 截图

参考提示 ❑ 2.4.1 节 pycurl 模块的常用类与方法说明参考官网 http://pycurl.sourceforge.net/doc/index.html。

第 3 章 Chapter 3

定制业务质量报表详解

在日常运维工作当中，会涉及大量不同来源的数据，比如每天的服务器性能数据、平台监控数据、自定义业务上报数据等，需要根据不同时段，周期性地输出数据报表，以方便管理员更加清晰、及时地了解业务的运营情况。在业务监控过程中，也需要更加直观地展示报表，以便快速定位问题。本章介绍 Excel 操作模块、rrdtool 数据报表、scapy 包处理等，相关知识点运用到运营平台中将起到增色添彩的作用。

3.1 数据报表之 Excel 操作模块

Excel 是当今最流行的电子表格处理软件，支持丰富的计算函数及图表，在系统运营方面广泛用于运营数据报表，比如业务质量、资源利用、安全扫描等报表，同时也是应用系统常见的文件导出格式，以便数据使用人员做进一步加工处理。本节主要讲述利用 Python 操作 Excel 的模块 XlsxWriter（https://xlsxwriter.readthedocs.org），可以操作多个工作表的文字、数字、公式、图表等。XlsxWriter 模块具有以下功能：

- 100% 兼容的 Excel XLSX 文件，支持 Excel 2003、Excel 2007 等版本；
- 支持所有 Excel 单元格数据格式；
- 单元格合并、批注、自动筛选、丰富多格式字符串等；
- 支持工作表 PNG、JPEG 图像，自定义图表；
- 内存优化模式支持写入大文件。

XlsxWriter 模块的安装方法如下：

```
# pip install XlsxWriter     #pip 安装方法
# easy_install XlsxWriter     #easy_install 安装方法

# 源码安装方法
# curl -O -L http://github.com/jmcnamara/XlsxWriter/archive/master.tar.gz
# tar zxvf master.tar.gz
# cd XlsxWriter-master/
# sudo python setup.py install
```

下面通过一个简单的功能演示示例，实现插入文字（中英字符）、数字（求和计算）、图片、单元格格式等，代码如下：

【 /home/test/XlsxWriter/simple1.py 】

```
#coding: utf-8
import xlsxwriter

workbook = xlsxwriter.Workbook('demo1.xlsx')     # 创建一个 Excel 文件
worksheet = workbook.add_worksheet()     # 创建一个工作表对象

worksheet.set_column('A:A', 20)     # 设定第一列（A）宽度为 20 像素
bold = workbook.add_format({'bold': True})     # 定义一个加粗的格式对象

worksheet.write('A1', 'Hello')     #A1 单元格写入 'Hello'
worksheet.write('A2', 'World', bold)     #A2 单元格写入 'World' 并引用加粗格式对象 bold
worksheet.write('B2', u' 中文测试 ', bold)     #B2 单元格写入中文并引用加粗格式对象 bold

worksheet.write(2, 0, 32)     # 用行列表示法写入数字 '32' 与 '35.5'
worksheet.write(3, 0, 35.5)     # 行列表示法的单元格下标以 0 作为起始值，'3,0' 等价于 'A3'
worksheet.write(4, 0, '=SUM(A3:A4)')     # 求 A3:A4 的和，并将结果写入 '4，0'，即 'A5'

worksheet.insert_image('B5', 'img/python-logo.png')     # 在 B5 单元格插入图片
workbook.close()     # 关闭 Excel 文件
```

程序生成的 demo1.xlsx 文档截图如图 3-1 所示。

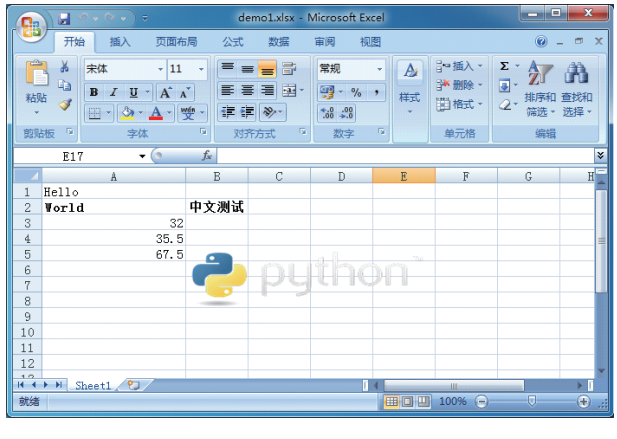


图 3-1　demo1.xlsx 文档截图

3.1.1　模块常用方法说明

1. Workbook 类

Workbook 类定义：Workbook(filename[, options])，该类实现创建一个 XlsxWriter 的 Workbook 对象。Workbook 类代表整个电子表格文件，并且存储在磁盘上。参数 filename（String 类型）为创建的 Excel 文件存储路径；参数 options（Dict 类型）为可选的 Workbook 参数，一般作为初始化工作表内容格式，例如值为 {'strings_to_numbers': True} 表示使用 worksheet.write() 方法时激活字符串转换数字。

- add_worksheet([sheetname]) 方法，作用是添加一个新的工作表，参数 sheetname（String 类型）为可选的工作表名称，默认为 Sheet1。例如，下面的代码对应的效果图如图 3-2 所示。

```
worksheet1 = workbook.add_worksheet()              # Sheet1
worksheet2 = workbook.add_worksheet('Foglio2')     # Foglio2
worksheet3 = workbook.add_worksheet('Data')        # Data
worksheet4 = workbook.add_worksheet()              # Sheet4
```

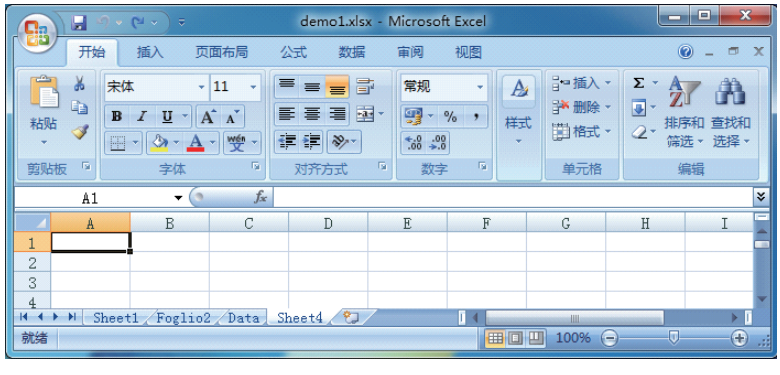


图 3-2　添加新工作表

- add_format([properties]) 方法，作用是在工作表中创建一个新的格式对象来格式化单元格。参数 properties（dict 类型）为指定一个格式属性的字典，例如设置一个加粗的格式对象，workbook.add_format({'bold': True})。通过 Format methods（格式化方法）也可以实现格式的设置，等价的设置加粗格式代码如下：

```
bold = workbook.add_format()
bold.set_bold()
```

更多格式化方法见 http://xlsxwriter.readthedocs.org/working_with_formats.html。

- add_chart（options）方法，作用是在工作表中创建一个图表对象，内部是通过 insert_chart() 方法来实现，参数 options（dict 类型）为图表指定一个字典属性，例如设置一个线条类型的图表对象，代码为 chart = workbook.add_chart({'type': 'line'})。
- close() 方法，作用是关闭工作表文件，如 workbook.close()。

2. Worksheet 类

Worksheet 类代表了一个 Excel 工作表，是 XlsxWriter 模块操作 Excel 内容最核心的一个类，例如将数据写入单元格或工作表格式布局等。Worksheet 对象不能直接实例化，取而代之的是通过 Workbook 对象调用 add_worksheet() 方法来创建。Worksheet 类提供了非常丰富的操作 Excel 内容的方法，其中几个常用的方法如下：

- write(row, col, *args) 方法，作用是写普通数据到工作表的单元格，参数 row 为行坐标，col 为列坐标，坐标索引起始值为 0；*args 无名字参数为数据内容，可以为数字、公式、字符串或格式对象。为了简化不同数据类型的写入过程，write 方法已经作为其他更加具体数据类型方法的别名，包括：
  - write_string()　写入字符串类型数据，如：

```
worksheet.write_string(0, 0, 'Your text here');
```

  - write_number()　写入数字类型数据，如：

```
worksheet.write_number('A2', 2.3451);
```

  - write_blank()　写入空类型数据，如：

```
worksheet.write('A2', None);
```

  - write_formula()　写入公式类型数据，如：

```
worksheet.write_formula(2, 0, '=SUM(B1:B5)');
```

  - write_datetime()　写入日期类型数据，如：

```
worksheet.write_datetime(7, 0,datetime.datetime.strptime('2013-01-23',
'%Y-%m-%d'),workbook.add_format({'num_format': 'yyyy-mm-dd'}));
```

  - write_boolean()　写入逻辑类型数据，如：

```
worksheet.write_boolean(0, 0, True);
```

  - write_url()　写入超链接类型数据，如：

worksheet.write_url('A1', 'ftp://www.python.org/')。

下列通过具体的示例来观察别名 write 方法与数据类型方法的对应关系，代码如下：

```
worksheet.write(0, 0, 'Hello')          # write_string()
worksheet.write(1, 0, 'World')          # write_string()
worksheet.write(2, 0, 2)                # write_number()
worksheet.write(3, 0, 3.00001)          # write_number()
worksheet.write(4, 0, '=SIN(PI()/4)')   # write_formula()
worksheet.write(5, 0, '')               # write_blank()
worksheet.write(6, 0, None)             # write_blank()
```

上述示例将创建一个如图 3-3 所示的工作表。

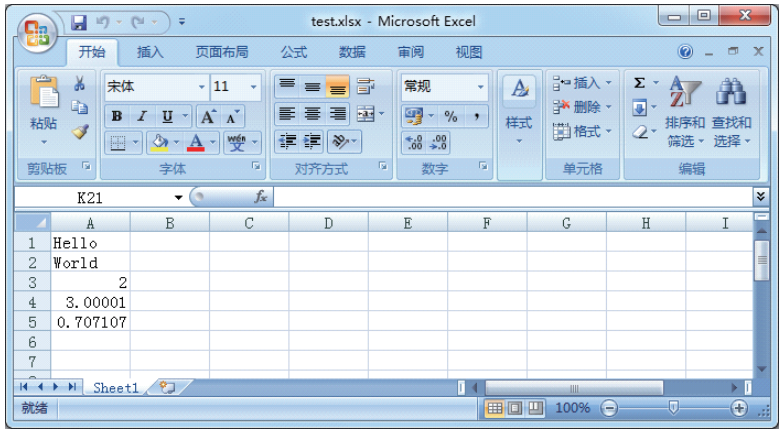


图 3-3　创建单元格并写入数据的工作表

- set_row（row, height, cell_format, options）方法，作用是设置行单元格的属性。参数 row（int 类型）指定行位置，起始下标为 0；参数 height（float 类型）设置行高，单位像素；参数 cell_format（format 类型）指定格式对象；参数 options（dict 类型）设置行 hidden（隐藏）、level（组合分级）、collapsed（折叠）。操作示例如下：

```
worksheet.write('A1', 'Hello')      # 在 A1 单元格写入 'Hellow' 字符串
cell_format = workbook.add_format({'bold': True})     # 定义一个加粗的格式对象
worksheet.set_row(0, 40, cell_format)     # 设置第 1 行单元格高度为 40 像素，且引用加粗
                                          # 格式对象
worksheet.set_row(1, None, None, {'hidden': True})     # 隐藏第 2 行单元格
```

上述示例将创建一个如图 3-4 所示的工作表。

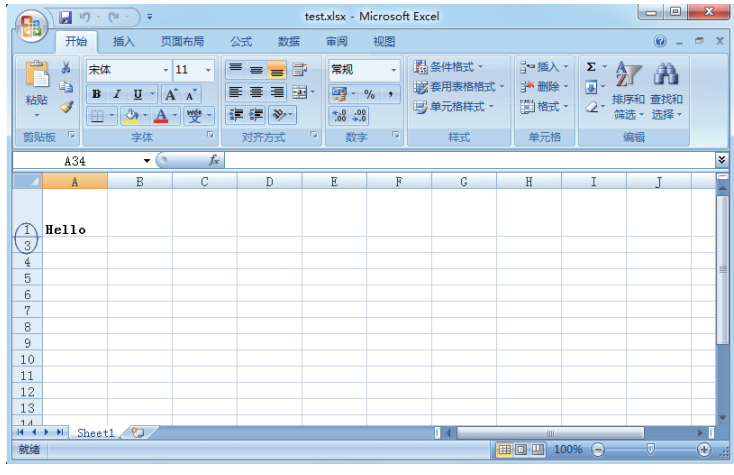


图 3-4 设置行单元格属性后的效果

❑ set_column（first_col, last_col, width, cell_format, options）方法，作用为设置一列或多列单元格属性。参数 first_col（int 类型）指定开始列位置，起始下标为 0；参数 last_col（int 类型）指定结束列位置，起始下标为 0，可以设置成与 first_col 一样；参数 width（float 类型）设置列宽；参数 cell_format（Format 类型）指定格式对象；参数 options（dict 类型）设置行 hidden（隐藏）、level（组合分级）、collapsed（折叠）。操作示例如下：

```
worksheet.write('A1', 'Hello')      # 在 A1 单元格写入 'Hello' 字符串
worksheet.write('B1', 'World')      # 在 B1 单元格写入 'World' 字符串
cell_format = workbook.add_format({'bold': True})     # 定义一个加粗的格式对象
                                    # 设置 0 到 1 即 (A 到 B) 列单元格宽度为 10 像素，
                                     且引用加粗格式对象
worksheet.set_column(0,1, 10,cell_format)
worksheet.set_column('C:D', 20)     # 设置 C 到 D 列单元格宽度为 20 像素
worksheet.set_column('E:G', None, None, {'hidden': 1})     # 隐藏 E 到 G 列单元格
```

上述示例将创建一个如图 3-5 所示的工作表。

❑ insert_image(row, col, image[, options]) 方法，作用是插入图片到指定单元格，支持 PNG、JPEG、BMP 等图片格式。参数 row 为行坐标，col 为列坐标，坐标索引起始值为 0；参数 image（string 类型）为图片路径；参数 options（dict 类型）为可选参数，作用是指定图片的位置、比例、链接 URL 等信息。操作示例如下：

```
# 在 B5 单元格插入 python-logo.png 图片，图片超级链接为 http://python.org
worksheet.insert_image('B5', 'img/python-logo.png', {'url': 'http://python.org'})
```

上述示例将创建一个如图 3-6 所示的工作表。

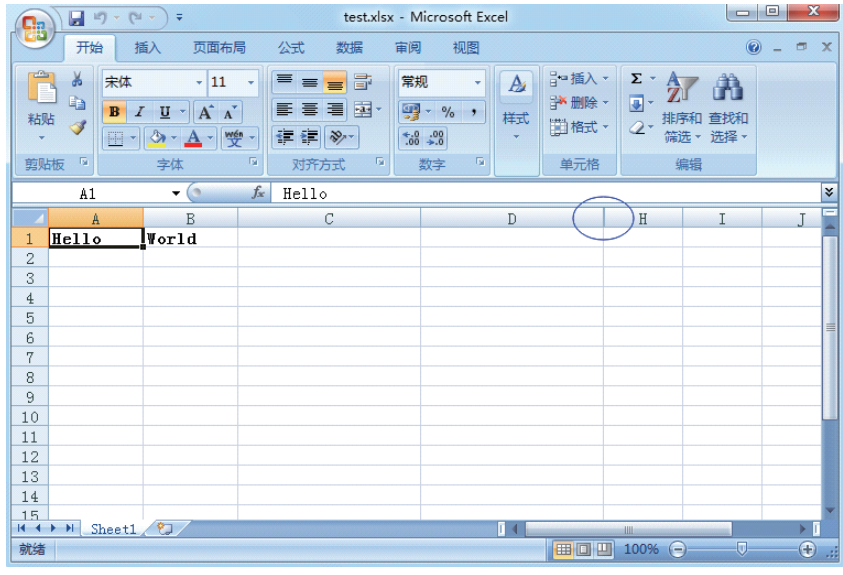


图 3-5　设置列单元格属性后的效果

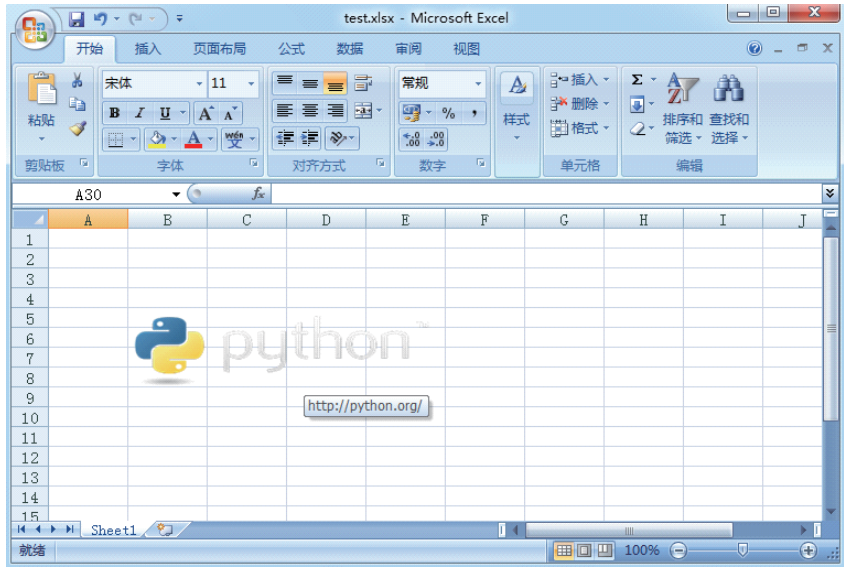


图 3-6　插入图片到单元格的效果

3. Chart 类

Chart 类实现在 XlsxWriter 模块中图表组件的基类，支持的图表类型包括面积、条形图、柱形图、折线图、饼图、散点图、股票和雷达等，一个图表对象是通过 Workbook（工作簿）

的 add_chart 方法创建，通过 {type,' 图表类型 '} 字典参数指定图表的类型，语句如下：

```
chart = workbook.add_chart({type, 'column'})     # 创建一个 column（柱形）图表
```

更多图表类型说明：

- ❍ area：创建一个面积样式的图表；
- ❍ bar：创建一个条形样式的图表；
- ❍ column：创建一个柱形样式的图表；
- ❍ line：创建一个线条样式的图表；
- ❍ pie：创建一个饼图样式的图表；
- ❍ scatter：创建一个散点样式的图表；
- ❍ stock：创建一个股票样式的图表；
- ❍ radar：创建一个雷达样式的图表。

然后再通过 Worksheet（工作表）的 insert_chart() 方法插入到指定位置，语句如下：

```
worksheet.insert_chart('A7', chart)     # 在 A7 单元格插入图表
```

下面介绍 chart 类的几个常用方法。

- ❑ chart.add_series (options) 方法，作用为添加一个数据系列到图表，参数 options (dict 类型) 设置图表系列选项的字典，操作示例如下：

```
chart.add_series({
    'categories': '=Sheet1!$A$1:$A$5',
    'values':     '=Sheet1!$B$1:$B$5',
    'line':       {'color': 'red'},
})
```

add_series 方法最常用的三个选项为 categories、values、line，其中 categories 作为是设置图表类别标签范围；values 为设置图表数据范围；line 为设置图表线条属性，包括颜色、宽度等。

- ❑ 其他常用方法及示例。
  - ❍ set_x_axis(options) 方法，设置图表 X 轴选项，示例代码如下，效果图如图 3-7 所示。

```
chart.set_x_axis({
    'name': 'Earnings per Quarter',     # 设置 X 轴标题名称
    'name_font': {'size': 14, 'bold': True},  # 设置 X 轴标题字体属性
    'num_font':  {'italic': True },     # 设置 X 轴数字字体属性
})
```

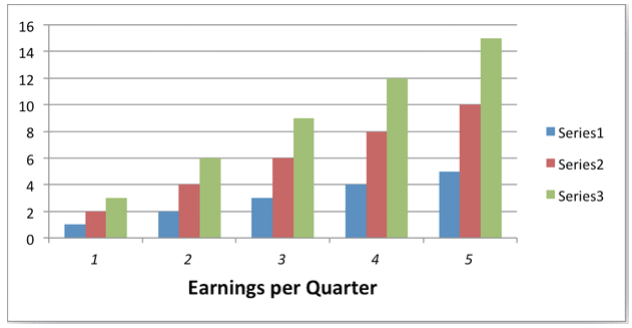


图 3-7　设置图表 X 轴选项

❍ set_size(options) 方法，设置图表大小，如 chart.set_size({'width': 720, 'height': 576})，其中 width 为宽度，height 为高度。

❍ set_title(options) 方法，设置图表标题，如 chart.set_title({'name': 'Year End Results'})，效果图如图 3-8 所示。

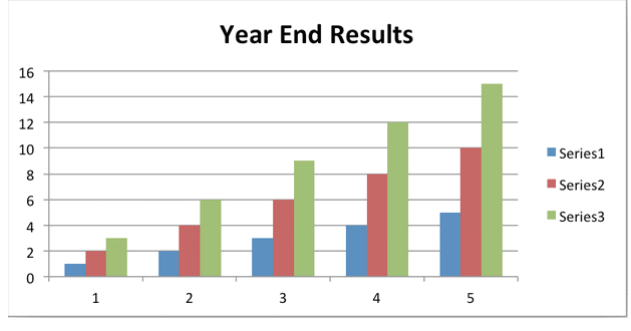


图 3-8　设置图表标题

❍ set_style(style_id) 方法，设置图表样式，style_id 为不同数字则代表不同样式，如 chart.set_style(37)，效果图如图 3-9 所示。

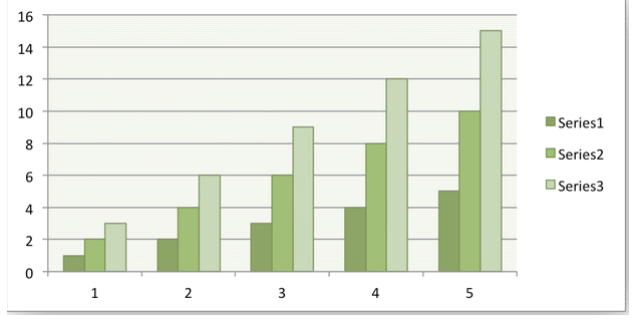


图 3-9　设置图表样式

❍ set_table(options) 方法，设置 X 轴为数据表格形式，如 chart.set_table()，效果图如图 3-10 所示。

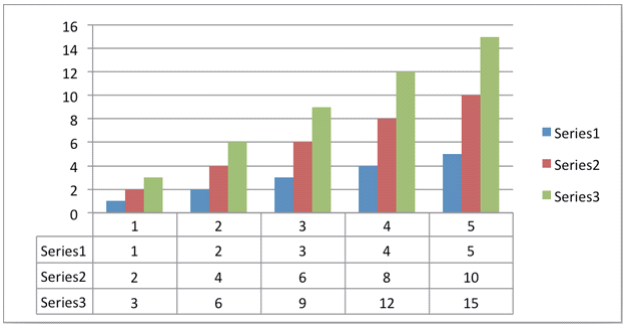


图 3-10 设置 X 轴为数据表格形式

3.1.2 实践：定制自动化业务流量报表周报

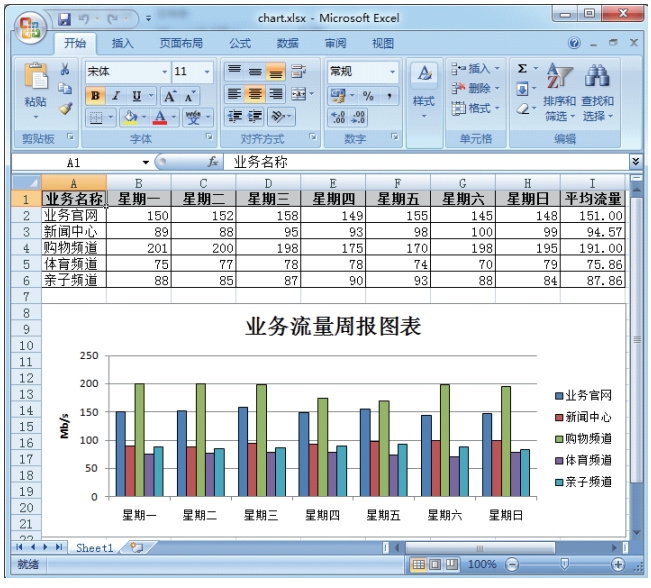
本次实践通过定制网站 5 个频道的流量报表周报，通过 XlsxWriter 模块将流量数据写入 Excel 文档，同时自动计算各频道周平均流量，再生成数据图表。具体是通过 workbook.add_chart({'type': 'column'}) 方法指定图表类型为柱形，使用 write_row、write_column 方法分别以行、列方式写入数据，使用 add_format() 方法定制表头、表体的显示风格，使用 add_series() 方法将数据添加到图表，同时使用 chart.set_size、set_title、set_y_axis 设置图表的大小及标题属性，最后通过 insert_chart 方法将图表插入工作表中。我们可以结合 2.3 节的内容来实现周报的邮件推送，本示例略去此功能。实现的代码如下：

【 /home/test/XlsxWriter/simple2.py 】

```
#coding: utf-8
import xlsxwriter

workbook = xlsxwriter.Workbook('chart.xlsx')     # 创建一个 Excel 文件
worksheet = workbook.add_worksheet()     # 创建一个工作表对象
chart = workbook.add_chart({'type': 'column'})     # 创建一个图表对象
# 定义数据表头列表
title = [u'业务名称',u'星期一',u'星期二',u'星期三',u'星期四',u'星期五',u'星期六',u'星期日',u'平均流量']
buname= [u'业务官网',u'新闻中心',u'购物频道',u'体育频道',u'亲子频道']     # 定义频道名称
# 定义 5 频道一周 7 天流量数据列表
data = [
    [150,152,158,149,155,145,148],
    [89,88,95,93,98,100,99],
    [201,200,198,175,170,198,195],
    [75,77,78,78,74,70,79],
```

```
    [88,85,87,90,93,88,84],
]
format=workbook.add_format()     # 定义 format 格式对象
format.set_border(1)     # 定义 format 对象单元格边框加粗 (1 像素 ) 的格式

format_title=workbook.add_format()     # 定义 format_title 格式对象
format_title.set_border(1)    # 定义 format_title 对象单元格边框加粗 (1 像素 ) 的格式
format_title.set_bg_color('#cccccc')    # 定义 format_title 对象单元格背景颜色为
                                        #'#cccccc' 的格式
format_title.set_align('center')     # 定义 format_title 对象单元格居中对齐的格式
format_title.set_bold()     # 定义 format_title 对象单元格内容加粗的格式

format_ave=workbook.add_format()     # 定义 format_ave 格式对象
format_ave.set_border(1)     # 定义 format_ave 对象单元格边框加粗 (1 像素 ) 的格式
format_ave.set_num_format('0.00')    # 定义 format_ave 对象单元格数字类别显示格式

# 下面分别以行或列写入方式将标题、业务名称、流量数据写入起初单元格，同时引用不同格式对象
worksheet.write_row('A1',title,format_title)
worksheet.write_column('A2', buname,format)
worksheet.write_row('B2', data[0],format)
worksheet.write_row('B3', data[1],format)
worksheet.write_row('B4', data[2],format)
worksheet.write_row('B5', data[3],format)
worksheet.write_row('B6', data[4],format)

# 定义图表数据系列函数
def chart_series(cur_row):
    worksheet.write_formula('I'+cur_row, \
     '=AVERAGE(B'+cur_row+':H'+cur_row+')',format_ave)     # 计算 (AVERAGE 函数) 频
                                                          # 道周平均流量
    chart.add_series({
        'categories': '=Sheet1!$B$1:$H$1',     # 将“星期一至星期日”作为图表数据标签 (X 轴 )
        'values':     '=Sheet1!$B$'+cur_row+':$H$'+cur_row,     # 频道一周所有数据作
                                                                # 为数据区域
        'line':       {'color': 'black'},     # 线条颜色定义为 black( 黑色 )
        'name': '=Sheet1!$A$'+cur_row,     # 引用业务名称为图例项
    })

for row in range(2, 7):     # 数据域以第 2 ~ 6 行进行图表数据系列函数调用
    chart_series(str(row))

#chart.set_table()     # 设置 X 轴表格格式，本示例不启用
#chart.set_style(30)     # 设置图表样式，本示例不启用
chart.set_size({'width': 577, 'height': 287})     # 设置图表大小
chart.set_title ({'name': u' 业务流量周报图表 '})     # 设置图表 (上方) 大标题
chart.set_y_axis({'name': 'Mb/s'})     # 设置 y 轴 (左侧) 小标题

worksheet.insert_chart('A8', chart)     # 在 A8 单元格插入图表
workbook.close()     # 关闭 Excel 文档
```

上述示例将创建一个如图 3-11 所示的工作表。



	A	B	C	D	E	F	G	H	I
1	业务名称	星期一	星期二	星期三	星期四	星期五	星期六	星期日	平均流量
2	业务官网	150	152	158	149	155	145	148	151.00
3	新闻中心	89	88	95	93	98	100	99	94.57
4	购物频道	201	200	198	175	170	198	195	191.00
5	体育频道	75	77	78	78	74	70	79	75.86
6	亲子频道	88	85	87	90	93	88	84	87.86

图 3-11 业务流量周报图表工作表

参考提示 3.4.1 节 XlsxWrite 模块的常用类与方法说明参考官网 http://xlsxwriter.readthedocs.org。

3.2 Python 与 rrdtool 的结合模块

rrdtool（round robin database）工具为环状数据库的存储格式，round robin 是一种处理定量数据以及当前元素指针的技术。rrdtool 主要用来跟踪对象的变化情况，生成这些变化的走势图，比如业务的访问流量、系统性能、磁盘利用率等趋势图，很多流行监控平台都使用到 rrdtool，比较有名的为 Cacti、Ganglia、Monitorix 等。更多 rrdtool 介绍见官网 http://oss.oetiker.ch/rrdtool/。rrdtool 是一个复杂的工具，涉及较多参数概念，本节主要通过 Python 的 rrdtool 模块对 rrdtool 的几个常用方法进行封装，包括 create、fetch、graph、info、update 等方法，本节对 rrdtool 的基本知识不展开说明，重点放在 Python rrdtool 模块的常用方法使用介绍上。

rrdtool 模块的安装方法如下：

```
easy_install python-rrdtool     #pip 安装方法
pip install python-rrdtool     #easy_install 安装方法
```

```
# 需要 rrdtool 工具及其他类包支持，CentOS 环境推荐使用 yum 安装方法
# yum install rrdtool-python
```

3.2.1　rrdtool 模块常用方法说明

下面介绍 rrdtool 模块常用的几个方法，包括 create（创建 rrd）、update（更新 rrd）、graph（绘图）、fetch（查询 rrd）等。

1. Create 方法

create filename [--start|-b start time] [--step|-s step] [DS:ds-name:DST:heartbeat:min:max] [RRA:CF:xff:steps:rows] 方法，创建一个后缀为 rrd 的 rrdtool 数据库，参数说明如下：

- filename 创建的 rrdtool 数据库文件名，默认后缀为 .rrd；
- --start 指定 rrdtool 第一条记录的起始时间，必须是 timestamp 的格式；
- --step 指定 rrdtool 每隔多长时间就收到一个值，默认为 5 分钟；
- DS 用于定义数据源，用于存放脚本的结果的变量；
- DST 用于定义数据源类型，rrdtool 支持 COUNTER（递增类型）、DERIVE（可递增可递减类型）、ABSOLUTE（假定前一个时间间隔的值为 0，再计算平均值）、GUAGE（收到值后直接存入 RRA）、COMPUTE（定义一个表达式，引用 DS 并自动计算出某个值）5 种，比如网卡流量属于计数器型，应该选择 COUNTER；
- RRA 用于指定数据如何存放，我们可以把一个 RRA 看成一个表，保存不同间隔的统计结果数据，为 CF 做数据合并提供依据，定义格式为：[RRA:CF:xff:steps:rows]；
- CF 统计合并数据，支持 AVERAGE（平均值）、MAX（最大值）、MIN（最小值）、LAST（最新值）4 种方式。

2. update 方法

update filename [--template|-t ds-name[:ds-name]...] N|timestamp:value[:value...] [timestamp:value[:value...] ...] 方法，存储一个新值到 rrdtool 数据库，updatev 和 update 类似，区别是每次插入后会返回一个状态码，以便了解是否成功（updatev 用 0 表示成功，–1 表示失败）。参数说明如下：

- filename 指定存储数据到的目标 rrd 文件名；
- -t ds-name[:ds-name] 指定需要更新的 DS 名称；
- N|Timestamp 表示数据采集的时间戳，N 表示当前时间戳；
- value[:value...] 更新的数据值，多个 DS 则多个值。

3. graph 方法

graph filename [-s|--start seconds] [-e|--end seconds] [-x|--x-grid x-axis grid and label] [-y|--y-grid y-axis grid and label] [--alt-y-grid] [--alt-y-mrtg] [--alt-autoscale] [--alt-autoscale-max] [--units-exponent] value [-v|--vertical-label text] [-w|--width pixels] [-h|--height pixels] [-i|--interlaced] [-f|--imginfo formatstring] [-a|--imgformat GIF|PNG|GD] [-B|--background value] [-O|--overlay value] [-U|--unit value] [-z|--lazy] [-o|--logarithmic] [-u|--upper-limit value] [-l|--lower-limit value] [-g|--no-legend] [-r|--rigid] [--step value] [-b|--base value] [-c|--color COLORTAG#rrggbb] [-t|--title title] [DEF:vname=rrd:ds-name:CF] [CDEF:vname=rpn-expression] [PRINT:vname:CF:format] [GPRINT:vname:CF:format] [COMMENT:text] [HRULE:value#rrggbb[:legend]] [VRULE:time#rrggbb[:legend]] [LINE{1|2|3}:vname[#rrggbb[:legend]]] [AREA:vname[#rrggbb[:legend]]] [STACK:vname[#rrggbb[:legend]]] 方法，根据指定的 rrdtool 数据库进行绘图，关键参数说明如下：

- ❑ filename 指定输出图像的文件名，默认是 PNG 格式；
- ❑ --start 指定起始时间；
- ❑ --end 指定结束时间；
- ❑ --x-grid 控制 X 轴网格线刻度、标签的位置；
- ❑ --y-grid 控制 Y 轴网格线刻度、标签的位置；
- ❑ --vertical-label 指定 Y 轴的说明文字；
- ❑ --width pixels 指定图表宽度（像素）；
- ❑ --height pixels 指定图表高度（像素）；
- ❑ --imgformat 指定图像格式（GIF|PNG|GD）；
- ❑ --background 指定图像背景颜色，支持 #rrggbb 表示法；
- ❑ --upper-limit 指定 Y 轴数据值上限；
- ❑ --lower-limit 指定 Y 轴数据值下限；
- ❑ --no-legend 取消图表下方的图例；
- ❑ --rigid 严格按照 upper-limit 与 lower-limit 来绘制；
- ❑ --title 图表顶部的标题；
- ❑ DEF:vname=rrd:ds-name:CF 指定绘图用到的数据源；
- ❑ CDEF:vname=rpn-expression 合并多个值；
- ❑ GPRINT:vname:CF:format 图表的下方输出最大值、最小值、平均值等；
- ❑ COMMENT:text 指定图表中输出的一些字符串；
- ❑ HRULE:value#rrggbb 用于在图表上面绘制水平线；
- ❑ VRULE:time#rrggbb 用于在图表上面绘制垂直线；
- ❑ LINE{1|2|3}:vname 使用线条来绘制数据图表，{1|2|3} 表示线条的粗细；

❑ AREA:vname 使用面积图来绘制数据图表。

4. fetch 方法

fetch filename CF [--resolution|-r resolution] [--start|-s start] [--end|-e end] 方法，根据指定的 rrdtool 数据库进行查询，关键参数说明如下：

❑ filename 指定要查询的 rrd 文件名；
❑ CF 包括 AVERAGE、MAX、MIN、LAST，要求必须是建库时 RRA 中定义的类型，否则会报错；
❑ --start --end 指定查询记录的开始与结束时间，默认可省略。

3.2.2　实践：实现网卡流量图表绘制

在日常运营工作当中，观察数据的变化趋势有利于了解我们的服务质量，比如在系统监控方面，网络流量趋势图直接展现了当前网络的吞吐。 CPU、内存、磁盘空间利用率趋势则反映了服务器运行健康状态。通过这些数据图表管理员可以提前做好应急预案，对可能存在的风险点做好防范。本次实践通过 rrdtool 模块实现服务器网卡流量趋势图的绘制，即先通过 create 方法创建一个 rrd 数据库，再通过 update 方法实现数据的写入，最后可以通过 graph 方法实现图表的绘制，以及提供 last、first、info、fetch 方法的查询。图 3-12 为 rrd 创建到输出图表的过程。

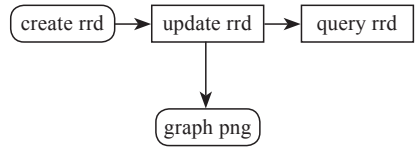


图 3-12　创建、更新 rrd 及输出图表流程

第一步　采用 create 方法创建 rrd 数据库，参数指定了一个 rrd 文件、更新频率 setp、起始时间 --start、数据源 DS、数据源类型 DST、数据周期定义 RRA 等，详细源码如下：

【 /home/test/rrdtool/create.py 】

```
# -*- coding: utf-8 -*-
#!/usr/bin/python
import rrdtool
import time

cur_time=str(int(time.time()))     # 获取当前 Linux 时间戳作为 rrd 起始时间
# 数据写频率 --step 为 300 秒（即 5 分钟一个数据点）
rrd=rrdtool.create('Flow.rrd','--step','300','--start',cur_time,
# 定义数据源 eth0_in（入流量）、eth0_out（出流量）；类型都为 COUNTER（递增）；600 秒为心跳值，
# 其含义是 600 秒没有收到值，则会用 UNKNOWN 代替；0 为最小值；最大值用 U 代替，表示不确定
  'DS:eth0_in:COUNTER:600:0:U',
  'DS:eth0_out:COUNTER:600:0:U',
```

```
  #RRA 定义格式为 [RRA:CF:xff:steps:rows]，CF 定义了 AVERAGE、MAX、MIN 三种数据合并方式
  #xff 定义为 0.5，表示一个 CDP 中的 PDP 值如超过一半值为 UNKNOWN，则该 CDP 的值就被标为 UNKNOWN
  # 下列前 4 个 RRA 的定义说明如下，其他定义与 AVERAGE 方式相似，区别是存最大值与最小值
  # 每隔 5 分钟 (1*300 秒 ) 存一次数据的平均值，存 600 笔，即 2.08 天
  # 每隔 30 分钟 (6*300 秒 ) 存一次数据的平均值，存 700 笔，即 14.58 天 ( 2 周 )
  # 每隔 2 小时 (24*300 秒 ) 存一次数据的平均值，存 775 笔，即 64.58 天 ( 2 个月 )
  # 每隔 24 小时 (288*300 秒 ) 存一次数据的平均值，存 797 笔，即 797 天 (2 年 )
  'RRA:AVERAGE:0.5:1:600',
  'RRA:AVERAGE:0.5:6:700',
  'RRA:AVERAGE:0.5:24:775',
  'RRA:AVERAGE:0.5:288:797',
  'RRA:MAX:0.5:1:600',
  'RRA:MAX:0.5:6:700',
  'RRA:MAX:0.5:24:775',
  'RRA:MAX:0.5:444:797',
  'RRA:MIN:0.5:1:600',
  'RRA:MIN:0.5:6:700',
  'RRA:MIN:0.5:24:775',
  'RRA:MIN:0.5:444:797')
if rrd:
    print rrdtool.error()
```

第二步 采用 updatev 方法更新 rrd 数据库，参数指定了当前的 Linux 时间戳，以及指定 eth0_in、eth0_out 值（当前网卡的出入流量），网卡流量我们通过 psutil 模块来获取，如 psutil.net_io_counters()[1] 为入流量，关于 psutil 模块的介绍见第 1.1。详细源码如下：

【 /home/test/rrdtool/update.py 】

```
# -*- coding: utf-8 -*-
#!/usr/bin/python
import rrdtool
import time,psutil

total_input_traffic = psutil.net_io_counters()[1]     # 获取网卡入流量
total_output_traffic = psutil.net_io_counters()[0]     # 获取网卡出流量
starttime=int(time.time())     # 获取当前 Linux 时间戳
# 将获取到的三个数据作为 updatev 的参数，返回 {'return_value': 0L} 则说明更新成功，反之失败
update=rrdtool.updatev('/home/test/rrdtool/Flow.rrd','%s:%s:%s'  %
(str(starttime),str(total_input_traffic),str(total_output_traffic)))
print update
```

将代码加入 crontab，并配置 5 分钟作为采集频率，crontab 配置如下：

```
*/5 * * * * /usr/bin/python /home/test/rrdtool/update.py > /dev/null 2>&1
```

第三步 采用 graph 方法绘制图表，此示例中关键参数使用了 --x-grid 定义 X 轴网格刻度；DEF 指定数据源；使用 CDEF 合并数据；HRULE 绘制水平线（告警线）；GPRINT 输出最大值、最小值、平均值等。详细源码如下：

【 /home/test/rrdtool/graph.py 】

```
# -*- coding: utf-8 -*-
#!/usr/bin/python
import rrdtool
import time
# 定义图表上方大标题
title="Server network  traffic flow ("+time.strftime('%Y-%m-%d', \
time.localtime(time.time()))+")"
# 重点解释 "--x-grid","MINUTE:12:HOUR:1:HOUR:1:0:%H" 参数的作用（从左往右进行分解）
“MINUTE:12”表示控制每隔 12 分钟放置一根次要格线
“HOUR:1”表示控制每隔 1 小时放置一根主要格线
“HOUR:1”表示控制 1 个小时输出一个 label 标签
“0:%H”0 表示数字对齐格线，%H 表示标签以小时显示
rrdtool.graph( "Flow.png", "--start", "-1d","--vertical-label=Bytes/s",\
"--x-grid","MINUTE:12:HOUR:1:HOUR:1:0:%H",\
 "--width","650","--height","230","--title",title,
 "DEF:inoctets=Flow.rrd:eth0_in:AVERAGE",     # 指定网卡入流量数据源 DS 及 CF
 "DEF:outoctets=Flow.rrd:eth0_out:AVERAGE",     # 指定网卡出流量数据源 DS 及 CF
 "CDEF:total=inoctets,outoctets,+",     # 通过 CDEF 合并网卡出入流量，得出总流量 total

"LINE1:total#FF8833:Total traffic",     # 以线条方式绘制总流量
 "AREA:inoctets#00FF00:In traffic",     # 以面积方式绘制入流量
 "LINE1:outoctets#0000FF:Out traffic",     # 以线条方式绘制出流量
 "HRULE:6144#FF0000:Alarm value\\r",     # 绘制水平线，作为告警线，阈值为 6.1k
 "CDEF:inbits=inoctets,8,*",     # 将入流量换算成 bit，即 *8，计算结果给 inbits
 "CDEF:outbits=outoctets,8,*",     # 将出流量换算成 bit，即 *8，计算结果给 outbits
"COMMENT:\\r",     # 在网格下方输出一个换行符
 "COMMENT:\\r",
 "GPRINT:inbits:AVERAGE:Avg In traffic\: %6.2lf %Sbps",     # 绘制入流量平均值
 "COMMENT:   ",
 "GPRINT:inbits:MAX:Max In traffic\: %6.2lf %Sbps",     # 绘制入流量最大值
 "COMMENT:  ",
 "GPRINT:inbits:MIN:MIN In traffic\: %6.2lf %Sbps\\r",     # 绘制入流量最小值
 "COMMENT: ",
 "GPRINT:outbits:AVERAGE:Avg Out traffic\: %6.2lf %Sbps",     # 绘制出流量平均值
 "COMMENT: ",
 "GPRINT:outbits:MAX:Max Out traffic\: %6.2lf %Sbps",     # 绘制出流量最大值
 "COMMENT: ",
 "GPRINT:outbits:MIN:MIN Out traffic\: %6.2lf %Sbps\\r")     # 绘制出流量最小值
```

以上代码将生成一个 Flow.png 文件，如图 3-13 所示。

提示　查看 rrd 文件内容有利于观察数据的结构、更新等情况，rrdtool 提供几个常用命令：

- ❑ info 查看 rrd 文件的结构信息，如 rrdtool info Flow.rrd；
- ❑ first 查看 rrd 文件第一个数据的更新时间，如 rrdtool first Flow.rrd；
- ❑ last 查看 rrd 文件最近一次更新的时间，如 rrdtool last Flow.rrd；
- ❑ fetch 根据指定时间、CF 查询 rrd 文件，如 rrdtool fetch Flow.rrd AVERAGE。

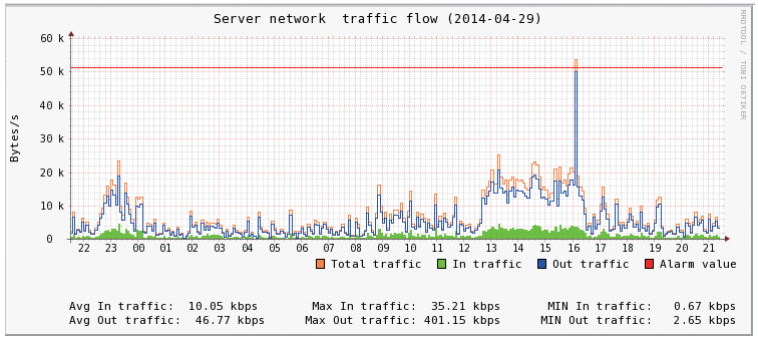


图 3-13 graph.py 执行输出图表

参考提示 3.2.1rrdtool 参数说明参考 http://bbs.chinaunix.net/thread-2150417-1-1.html 和 http://oss. oetiker.ch/rrdtool/doc/index.en.html。

3.3 生成动态路由轨迹图

scapy（http://www.secdev.org/projects/scapy/）是一个强大的交互式数据包处理程序，它能够对数据包进行伪造或解包，包括发送数据包、包嗅探、应答和反馈匹配等功能。可以用在处理网络扫描、路由跟踪、服务探测、单元测试等方面，本节主要针对 scapy 的路由跟踪功能，实现 TCP 协议方式对服务可用性的探测，比如常用的 80（HTTP）与 443（HTTPS）服务，并生成美观的路由线路图报表，让管理员清晰了解探测点到目标主机的服务状态、骨干路由节点所处的 IDC 位置、经过的运营商路由节点等信息。下面详细进行介绍。

scapy 模块的安装方法如下：

```
# scapy 模板需要 tcpdump 程序支持，生成报表需要 graphviz、ImageMagick 图像处理包支持
# yum -y install tcpdump graphviz ImageMagick

# 源码安装
# wget http://www.secdev.org/projects/scapy/files/scapy-2.2.0.tar.gz
# tar -zxvf scapy-2.2.0.tar.gz
# cd scapy-2.2.0
# python setup.py install
```

3.3.1 模块常用方法说明

scapy 模块提供了众多网络数据包操作的方法，包括发包 send()、SYN\ ACK 扫描、嗅探

sniff()、抓包 wrpcap()、TCP 路由跟踪 traceroute() 等，本节主要关注服务监控内容接下来详细介绍 traceroute() 方法，其具体定义如下：

traceroute(target, dport=80, minttl=1, maxttl=30, sport=<RandShort>, l4=None, filter=None, timeout=2, verbose=None, **kargs)

该方法实现 TCP 跟踪路由功能，关键参数说明如下：

- target：跟踪的目标对象，可以是域名或 IP，类型为列表，支持同时指定多个目标，如 ["www.qq.com","www.baidu.com","www.google.com.hk"]；
- dport：目标端口，类型为列表，支持同时指定多个端口，如 [80,443]；
- minttl：指定路由跟踪的最小跳数（节点数）；
- maxttl：指定路由跟踪的最大跳数（节点数）。

3.3.2　实践：实现 TCP 探测目标服务路由轨迹

在此次实践中，通过 scapy 的 traceroute() 方法实现探测机到目标服务器的路由轨迹，整个过程的原理见图 3-14，首先通过探测机以 SYN 方式进行 TCP 服务扫描，同时启动 tcpdump 进行抓包，捕获扫描过程经过的所有路由点，再通过 graph() 方法进行路由 IP 轨迹绘制，中间调用 ASN 映射查询 IP 地理信息并生成 svg 流程文档，最后使用 ImageMagick 工具将 svg 格式转换成 png，流程结束。

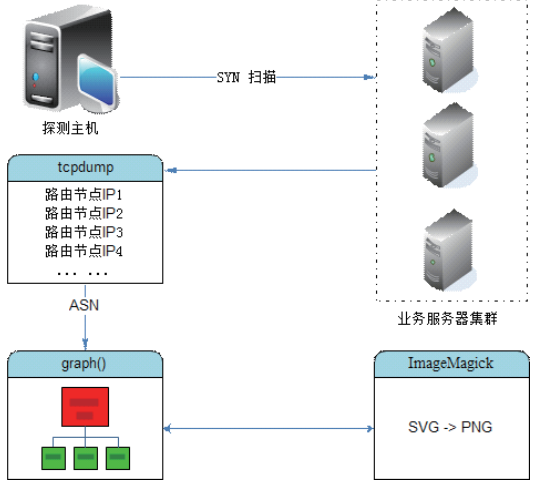


图 3-14　TCP 探测目标服务路由轨迹原理图

本次实践通过 traceroute() 方法实现路由的跟踪，跟踪结果动态生成图片格式。功能实现源码如下：

【 /home/test/scapy/simple1.py 】

```
# -*- coding: utf-8 -*-
import os,sys,time,subprocess
import warnings,logging
warnings.filterwarnings("ignore", category=DeprecationWarning) # 屏蔽 scapy 无用告警信息
logging.getLogger("scapy.runtime").setLevel(logging.ERROR) # 屏蔽模块 IPv6 多余告警
from scapy.all import traceroute

domains = raw_input('Please input one or more IP/domain: ') # 接受输入的域名或 IP
target =  domains.split(' ')
dport = [80]    # 扫描的端口列表

if len(target) >= 1 and target[0]!='':
    res,unans = traceroute(target,dport=dport,retry=-2) # 启动路由跟踪
    res.graph(target="> test.svg")    # 生成 svg 矢量图形
    time.sleep(1)
    subprocess.Popen("/usr/bin/convert test.svg test.png", shell=True) #svg 转 png 格式
else:
    print "IP/domain number of errors,exit"
```

代码运行结果见图 3-15，“ -”表示路由节点无回应或超时；“11”表示扫描的指定服务无回应；“SA”表示扫描的指定服务有回应，一般是最后一个主机 IP。

```
Received 73 packets, got 39 answers, remaining 21 packets
   113.108.238.121:tcp80 180.96.12.11:tcp80
1  192.168.1.1     11    192.168.1.1     11
2  114.116.64.1    11    114.116.64.1    11
3  10.145.209.26   11    10.145.209.26   11
4  10.144.12.74    11    10.145.209.25   11
5  10.144.10.66    11    10.144.10.66    11
6  10.144.10.206   11    10.144.10.206   11
7  10.144.12.153   11    10.144.12.153   11
8  10.83.64.1      11    10.83.64.1      11
11 10.252.253.1    11    10.252.253.1    11
18 -                     202.97.49.221   11
20 113.108.238.121 SA    202.102.69.22   11
21 113.108.238.121 SA    -
22 113.108.238.121 SA    -
23 113.108.238.121 SA    180.96.12.11    SA
24 113.108.238.121 SA    180.96.12.11    SA
25 113.108.238.121 SA    180.96.12.11    SA
26 113.108.238.121 SA    180.96.12.11    SA
27 113.108.238.121 SA    180.96.12.11    SA
28 113.108.238.121 SA    180.96.12.11    SA
29 113.108.238.121 SA    180.96.12.11    SA
30 113.108.238.121 SA    180.96.12.11    SA
```

图 3-15　代码运行结果

生成的路由轨迹图见图 3-16（仅局部），“ -”将使用 unk* 单元代替，重点路由节点将通过 ASN 获取所处的运营商或 IDC 位置，如 IP“202.102.69.210”为“ CHINANET-JS-AS-AP AS Number for CHINANET jiangsu province backbone,CN”意思为该 IP 所处中国电信江苏省骨干网。

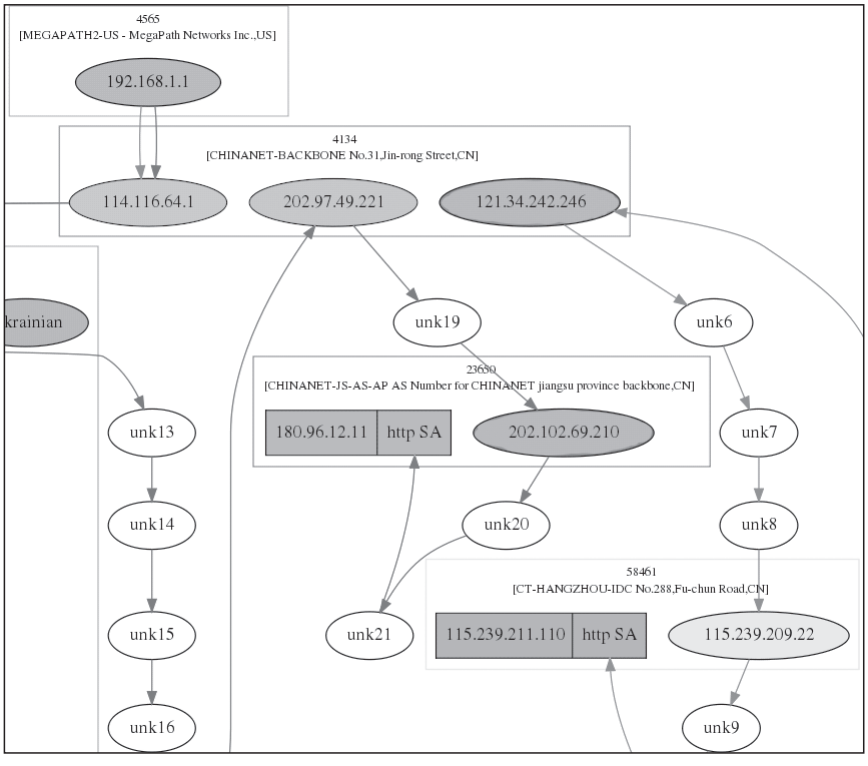


图 3-16　路由轨迹图

通过路由轨迹图，我们可以非常清晰地看到探测点到目标节点的路由走向，运营商时常会做路由节点分流，不排除会造成选择的路由线路不是最优的，该视图可以帮助我们了解到这个信息。另外 IE8 以上及 chrome 浏览器都已支持 SVG 格式文件，可以直接浏览，无需转换成 png 或其他格式，可以轻松整合到我们的运营平台当中。

参考提示　3.3.1 节 scapy 方法参数说明参考 http://www.secdev.org/projects/scapy/doc/usage.html。