第一部分 基础篇

第 1 章 系统基础信息模块详解

1. psutil 是一个跨平台库，能够轻松实现获取系统运行进程和系统利用率（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。获取当前物理内存总大小及已使用大小，shell 命令如下：（psutil、系统运行进程、系统利用率）

```
物理内存 total 值：free -m | grep Mem | awk '(print $2}'
物理内存 used 值：free -m | grep Mem | 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)
```

2. 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。
3. 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。

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 的物理个数也逻辑个数。 （cpu_times()、CPU 利用率）

```
>>> 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
>>>
```

4. 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：
（psutil.virtual_memory()、psutil.swap_memory()）

```
>>> import psutil
>>>mem = psutil.virtual_memory()  #使用psutil.virtual_memory获取内存完整信息
>>>mem
svmeut (total=506277888L, availabl6=204951552L, percent=59.5, used=499867648L,
free=6410240L, active=2 45858304 , inactive=l63733504, 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)
```

5. 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：　　（磁盘利用率、psutil.disk_usage、IO信息、disk_io_counters）

```
>>>psutil.disk_partitions() #使用psutil.disk_partitions方法获取磁盘完整信息
[sdiskpart(device='/dev/sdal',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
```

6. 系统的网络信息与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=200974（发送数据包数）等。这些网络信息使用psutil.net_io_counters()方法获取。（net_io_counters()、发送字节数、接收字节数）
7. ipy模块用于处理IP地址。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IPv6和IPv4，如：　　（ipy、ip地址）

```
>>>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)
```

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
'PUBLIC1'
>>> 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
```

9. IP方法也支持网络地址的转换，例如根据IP与掩码生成网段格式，如下：（转换、make_net）

```
>>>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： （strNormal、定制网段）

```
>>>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(1192.168.1.0/24').strNormal(2)
'1192.168.1.0/255.255.255.0'
>>>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。

10. 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： （比较、小于）

```
>>>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：（in、包含）

```
>>> '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
```

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

```
>>>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
```

12. ipy 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
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 IP 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
```

13. dnspython 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。（dnspython、dns 工具包）

14. 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： （resolver.query、查询）

```
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_on_answer 参数用于指定当查询无应答时是否触发异常，默认为 True。

15. 常见解析类型示例说明

(1) A 记录

```
#!/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
        print j.address
```

(2) 实现 MX 记录查询方法源码

```
#!/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
```

(3) NS 记录

```
#!/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()
```

(4) CNAME 记录

```
#!/usr/bin/env python
import dns.resolver
domain = raw_input('Please input an domain:')
cname = dns.resolver.query(domainf
    for i in cname.response.answer:
        for j in i.items:
            print j.to text()
```

第 2 章 业务监控服务详解

1. 通过标准库的 difflib 模块可以实现文件内容差异对比。 （difflib、差异对比）
2. differ()对两个字符串进行比较，SequenceMatcher()支持任意类型序列的比较，HtmlDiff()支持将比较结果输出为 html 格式。（differ、字符串、SequenceMatcher、任意类型序列、HtmlDiff、html 格式）
3. 采用 htmldiff()类的 make_file()方法就可以生成美观的 html 文档。（make_file()、html 文档）
4. 标准库中的 filecmp 可以实现文件、目录、遍历子目录的差异对比功能。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： （cmp、cmpfiles、dircmp ）

- 单文件对比，采用 filecmp.cmp(f 1 , f2[, shallow])方法，比较文件名为 f 1 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 shallcw 为 false 时，则 os.stat()与文件内容同时进行检验。
- 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、不具备志权限或其他原因导致的不能比较的文件清单。
- 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]。dir 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。

5. dircmp 类提供了以下函数：

- report()　　（比较指定目录并打印）

  比较当前指定目录中的内容并打印。

- report_partial_closure()　　（比较第一级子目录并打印）

  比较当前指定目录及第一级子目录中的内容并打印。

- report_full_closure()　　（递归比较并打印）

  递归比较所有指定目录的内容并打印。

  同时，dircmp 类提供了以下属性：

- left：左目录，如类中定义的目录 a。　（左）
- right：右目录，即目录 b。　　（右）
- left_list：左目录中的文件及目录列表。　　（左目录、列表）
- right_list：右目录中的文件及目录列表。　　（右目录、列表）
- common：两边目录共同存在的文件或目录。　　（共同存在）
- left_only：只在左目录中的文件或目录。　　（只在左目录）
- right_only：只在右目录中的文件或目录。（只在右目录）
- common_dirs：a 和 b 中共同存在的子目录。　　（共同、子目录）
- common_files：a 和 b 中共同存在的子文件。　（共同、子文件）
- common_funny：两边目录都存在的子目录（不同类型或 os.stat()记录的错误）　　（两边都存在）
- same_files：匹配相同的文件。　　（相同文件）
- diff_files：不匹配的文件。　　（不匹配）
- funny_files：两边目录中都存在，但无法比较的文件。　　（两边目录都存在）

6. 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，比如登陆、校验、发送、退出等。host 参数为远程 smtp 主机的地址，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 类具有以下方法：（建立连接、发送请求）

(1)　　SMTP.connect([host[,port]])方法，连接远程 smtp 主机，host 为远程主机地址，port 为远程主机端口，默认为 25，也可以直接使用 host:port 形式表示。（连接）

(2)　　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.（校验方法）

(3)　　SMTP.sendmail(from_addr,to_addrs,msg[,mail_options,rcpt_options])方法，实现邮件的发送功能，参数是发件人、收件人、邮件内容。　　（发送邮件）

(4)　　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。　　（安全传输模式）

(5)　　SMTP.quit()，断开 smtp 服务器的连接。

7. MIME 是一个新的扩展邮件格式，可以让人在邮件主体中会包含 HTML、图像、声音以及附件格式等。下面介绍几种 python 中常用的 MIME 类：（MIME、扩展邮件格式）

(1) 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。（MIMEMultipart、多个部分）

(2) 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。（MIMEAudio 、音频）

(3) email.mime.image.MIMEImage(_imagedata[,_subtype[,_encoder[,**_params]]])，创建包含图片数据的邮件体，_audiodata 包含原始图片数据的字节字符串。（MIMEImage、图片）

(4) 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 heml 类型的字符串。 （MIMEText、文本）

8. pycurl 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，作用类似于 request 库。下面介绍 Curl 对象几个常用的方法：（pycurl、curl 命令功能的 python 封装）

(1) 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。 （close、关闭、回收）

(2) 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。 （perform、请求提交）

(3) setopt(option,value)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、整形、长整形、文件对象、列表或函数等。

(4) 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。

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import pycurl
import time
class Test:
    def __init__(self):
        self.contents = ''
    def body_callback(self, buf):
        self.contents = self.contents + buf

sys.stderr.write("Testing %sn" % pycurl.version)
start_time = time.time()
url = 'http://www.dianping.com/hangzhou'
t = Test()
c = pycurl.Curl()
```

```
c.setopt(c.URL, url)
c.setopt(c.WRITEFUNCTION, t.body_callback)
c.perform()
end_time = time.time()
duration = end_time - start_time
print c.getinfo(pycurl.HTTP_CODE), c.getinfo(pycurl.EFFECTIVE_URL)
c.close()
print 'pycurl takes %s seconds to get %s ' % (duration, url)
print 'lenth of the content is %d' % len(t.contents)
#print(t.contents)
```

第 3 章 定制业务质量报表详解

1. 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。使用 Excel 操作模块、rrdtool 数据报表、scapy 包等可以完成这项工作。

a) 数据报表之 Excel 操作模块

1. python 操作 excel 的 XlsxWriter 模块可以操作多个工作表的文字、数字、公式、图表等。 （XlsxWriter 模块）
2. workbook 类定义 workbook(filename[,option])，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。（workbook 类、整个电子表格文件）

(1) add_worksheet([sheetname])方法，作用是添加一个新的工作表，参数为可选的工作表名称，默认为 sheet1。 （添加新的）

(2) 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。参数为指定一个格式属性的字典。 （格式化单元格）

(3) add_chart(options)方法，作用是在工作表中创建一个图表对象。 （创建图表对象）

3. worksheet 类代表一个 excel 工作表，是 XlsWriter 模块操作 Excel 内容最核心的一个类。worksheet 对象不能直接实例化，而是通过 workbook 对象调用 add_worksheet()方法来创建。worksheet 几个常用的方法如下： （worksheet、不能直接实例化）

(1) write(row,col,*args)方法：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 为数据内容，可以为数字、化工、字符串或格式对象。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()：写入字符串数据。
- write_number()：写入数字类型数据。

- write_bland()：写入空类型数据。
- write_formula()：写入公式类型数据。
- write_datatime()：写入日期类型数据。
- write_boolean()：写入逻辑类型数据。
- write_url()：写入超链接数据。

(2) set_row(row,height,cell_format,options)：设置行单元的属性。

(3) set_column(first_col,last_col,width,cell_format,options)：作用为设置一列或多列单元格属性。

(4) 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。

4. chart 类实现在 XlsxWriter 模块中图片组件的基类，支持的图表类型包括面积、条形图、柱形图、折线图、饼图等，一个图表对象是通过 Workbook（工作薄）的 add_chart 方法创建，通过｛type,’图表类型’｝参数指定图表的类型。（chart 类、图片组件）

(1) Chart.add_series(options)：添加一个数据系统列图表，参数设置图表系列选项的字典。

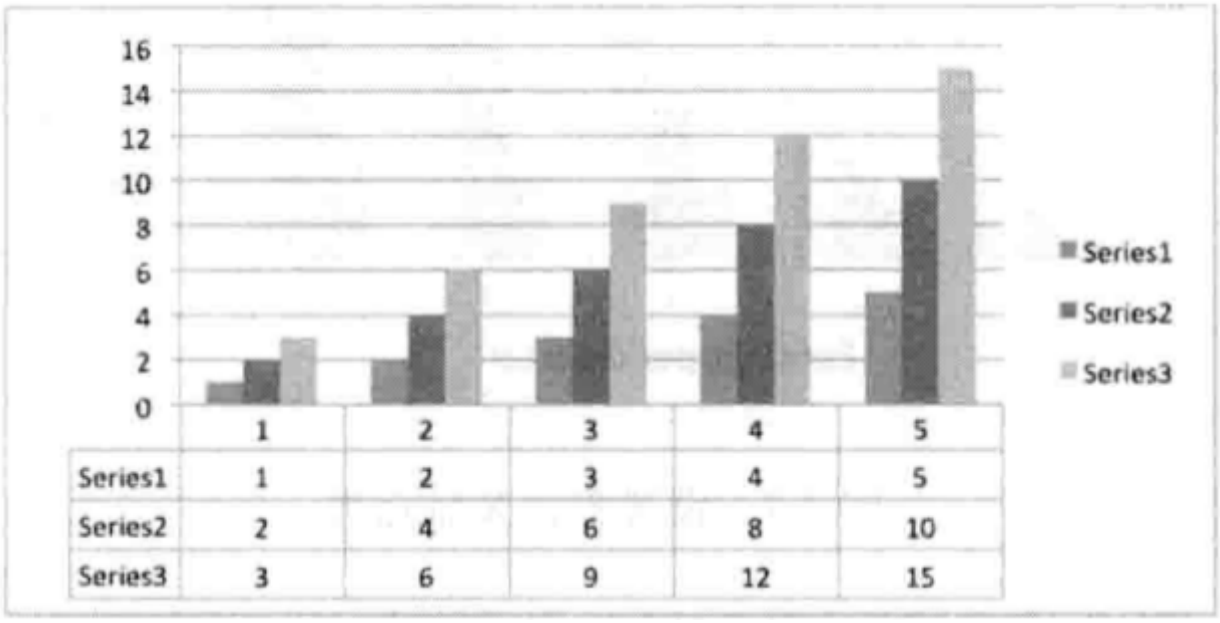
(2) set_x_axis(options)：设置图表 X 轴选项。

(3) set_size(options)：设置图表大小。

(4) set_title(options)：设置图表标题。

(5) set_style(style_id)：设置图表样式。

(6) set_table(options)：设置 x 轴为数据表格形式。如图：



	1	2	3	4	5
Series1	1	2	3	4	5
Series2	2	4	6	8	10
Series3	3	6	9	12	15

b) rrdtool

1. rrdtool 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。　　（环状数据库）

2. RRDTool 是一套监测工具，可用于存储和展示被监测对象随时间的变化情况。比如，我们在 Windows 电脑上常见的内存和 CPU 使用情况：　　（RRDTool 、监测工具）

3. RRD 全称是环型数据库。顾名思义，它是一种循环使用存储空间的数据库，适用于存储和时间序列相关的数据。RRD 数据库在被创建的时候就已经定义好了大小，当空间存储满了以后，又从头开始覆盖旧的数据，所以和其他线性增长的数据库不同，RRD 的大小可控且不用维护。假设你存储了时刻 3 的监测值，那就不能再存储时刻 2 的监测值了。　　（RRD、环型数据库）

4. create 方法：

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

表示使用 rrdtool 命令 create 创建一个名为 filename 的数据库文件，通常 RRD 数据库文件的后缀为 .rrd ，但是你随便使用文件名也不会有影响。第二行开始的为可选参数。

5. update 方法：

```
rrdtool.update filename [可选参数]
```

存储一个新值到 rrdtool 数据库。

6. grap 方法：　　（绘图）

```
Rrdtool.graph filename [可选参数]
```

根据指定的 rrdtool 数据库进行绘图。

7. fetch 方法：　　（查询）

```
Rrdtool.fetch filename CF [可选参数]
```

根据指定的 rrdtool 数据库进行查询。

c) Scapy

1. 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。 （scapy、数据包、伪造和解包）

第 4 章 python 与系统安全

1. Clam antivirus（ClamAV）是一款免费而且开放源代码的防毒软件，软件与病毒库的更新皆由社区免费发布。ClamAV 主要为 linux、unix 系统提供病毒扫描、查杀等服务。pyClamad 模块是一个 python 第三方模块，可让 python 直接使用 clamAV 病毒扫描守护进程 alamd，来实现一个高效的病毒检测功能，另外，pyClamad 模块也非常容易整合到已有的平台中。 （ClamAV、pyClamad 模块、防毒）
2. pyClamd 提供了两个关键类，一个为 ClamdNetworkSocket()类 ，实现使用网络套接字操作 clamd；另一个为 ClamdUnixSocket0 类 ，实现使用 Unix 套接字类操作 clamd。两个类定义的方法完全一样，现以 ClamdNetworkSocket()类进行说明。（ClamdNetworkSocket、ClamdUnixSocket、网络套接字操作、Unix 套接字类操作）

(1) _ init (self, host=’127.0.0.1’,port=3310, timeout=None)：是 ClamdNetworkSocket 类的初始化方法，参数 host 为连接主机 IP； 参数 port 为连接的端口，默认为 3310，与 /etc/clamd.conf 配置文件中的 TCPSocket 参数要保持一致；timeout 为连接的超时时间。

(2) contscan_file(self, file)：实现扫描指定的文件或目录，在扫描时发生错误或发现病毒将不终止，参数 file （string 类型）为指定的文件或目录的绝对路径。（扫描）

(3) multiscan_file(self，file)方法，实现多线程扫描指定的文件或目录，多核环境速度更快，在扫描时发生错误或发现病毒将不终止，参数 file （string 类型）为指定的文件或目录的绝对路径。 （多线程扫描）

(4) scan_file(self,file)方法，实现扫描指定的文件或目录，在扫描时发生错误或发现病毒将终止，参数 file （string 类型）为指定的文件或目录的绝对路径。（扫描）

(5) shutdown (self)方法，实现强制关闭 clamd 进程并退出。 （强制关闭）

(6) stats(self)方法，获取 Clamscan 的当前状态。 （当前状态）

(7) reload (self)方法，强制重载 clamd 病毒特征库，扫描前建议做 reload 操作。（强制重载）

(8) EICAR(self) 方法，返回 EICAR 测试字符串，即生成具有病毒特征的字符串，便于测试。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
import pyclamd
from threading import Thread
```

```
class Scan(Thread): #继承多线程Thread类

    def __init__ (self,IP,scan_type,file):
        """构造方法"""
        Thread.__init__(self)
        self.IP = IP
        self.scan_type=scan_type
        self.file = file
        self.connstr=""
        self.scanresult=""


    def run(self):
        """多进程run方法"""

        try:
            cd = pyclamd.ClamdNetworkSocket(self.IP,3310)
            """探测连通性"""
            if cd.ping():
                self.connstr=self.IP+" connection [OK]"
                """重载clamd病毒特征库"""
                cd.reload()
                """判断扫描模式"""
                if self.scan_type=="contscan_file":
                    self.scanresult="{0}\n".format(cd.contscan_file(self.file))
                elif self.scan_type=="multiscan_file":
                    self.scanresult="{0}\n".format(cd.multiscan_file(self.file))
                elif self.scan_type=="scan_file":
                    self.scanresult="{0}\n".format(cd.scan_file(self.file))
                time.sleep(1)
            else:
                self.connstr=self.IP+" ping error,exit"
                return
        except Exception,e:
            self.connstr=self.IP+" "+str(e)


IPs=['192.168.1.21','192.168.1.22'] #扫描主机的列表
scantype="multiscan_file" #指定扫描模式
scanfile="/data/www" #指定扫描路径
i=1
threadnum=2 #指定启动的线程数
scanlist = [] #存储Scan类线程对象列表

for ip in IPs:
```

```
    """将数据值带入类中，实例化对象"""
    currp = Scan(ip,scantype,scanfile)
    scanlist.append(currp) #追加对象到列表

"""当达到指定的线程数或IP列表数后启动线程"""
    if i%threadnum==0 or i==len(IPs):
        for task in scanlist:
            task.start() #启动线程

        for task in scanlist:
            task.join() #等待所有子线程退出，并输出扫描结果
            print task.connstr #打印服务器连接信息
            print task.scanresult #打印结果信息
        scanlist = []
    i+=1
```

3. python-nmap是一个用于端口扫描的库。python-nmap作为nmap命令的python封装，可以让python很方便地操作nmap扫描器。它可以帮助管理员完成自动扫描任务和生成报告。本节介绍PortScanner类，实现一个nmap工具的端口扫描功能封装；另一个是PortScannerHostDict类。下面介绍PortScanner类：（python-nmap、PortScanner类、端口扫描）

(1) scan(self,hosts='127.0.0.1',ports=none,argument='-sV')：实现指定主机、端口、nmap命令行参数的扫描。（扫描）

(2) command_line(self)：返回扫描方法映射到具体nmap命令行。

(3) scaninfo(self)：返回nmap扫描信息，格式为字典类型。（nmap扫描信息）

(4) all_hosts(self)：返回nmap扫描的主机清单，格式为列表类型。（主机清单）

(5) hostname(self)：返回扫描对象的主机名。（主机名）

(6) state(self)：返回扫描对象的状态，包括4种状态（up、down、unknown、skipped）。（扫描对象的状态）

(7) all_protocols(self)：返回扫描的协议。（扫描的协议）

(8) all_tcp()：返回TCP协议扫描的端口。（扫描的端口）

(9) tcp(self,port)：返回扫描TCP协议port的信息。（TCP协议port的信息）

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import nmap

scan_row=[]
input_data = raw_input('Please input hosts and port: ')
```

```
scan_row = input_data.split(" ")
if len(scan_row)!=2:
    print "Input errors,example \"192.168.1.0/24 80,443,22\""
    sys.exit(0)
hosts=scan_row[0]    #接收用户输入的主机
port=scan_row[1]    #接收用户输入的端口

try:
    nm = nmap.PortScanner()    #创建端口扫描对象
except nmap.PortScannerError:
    print('Nmap not found', sys.exc_info()[0])
    sys.exit(0)
except:
    print("Unexpected error:", sys.exc_info()[0])
    sys.exit(0)

try:
    nm.scan(hosts=hosts, arguments=' -v -sS -p '+port)    #调用扫描方法，参数指定扫描主机hosts，nmap扫描命令行参数arguments
except Exception,e:
    print "Scan erro:"+str(e)

for host in nm.all_hosts():    #遍历扫描主机
    print('----------------------------------------------------')
    print('Host : %s (%s)' % (host, nm[host].hostname()))    #输出主机及主机名
    print('State : %s' % nm[host].state())    #输出主机状态，如up、down

    for proto in nm[host].all_protocols():    #遍历扫描协议，如tcp、udp
        print('----------')
        print('Protocol : %s' % proto)    #输入协议名

        lport = nm[host][proto].keys()    #获取协议的所有扫描端口
        lport.sort()    #端口列表排序
        for port in lport:    #遍历端口及输出端口与状态
            print('port : %s\tstate : %s' % (port, nm[host][proto][port]['state']))
```

第5章 系统批量运维管理器pexpect详解

1. 通过pexpect我们可以实现对ssh、ftp、passwd、telnet等命令行进行自动交互，而无需人工干涉来达到自动化的目的。它可以用于自动化安装脚本，用于复制不同服务器上的软件包安装。 它可以用于自动化软件测试。（pexpect、自动交互、自动化测试）

a) spawn

1. spawn 类是 pexpect 的主要类接口，功能是启动和控制应用程序，以下是它的构造函数字义：　　（spawn 类、启动和控制应用程序）

```
Class
pexpect.spawn(command,args=[],timeout=30,maxread=2000,searchwindowsize=none,logfile=none,cwd=none,env=none,ignore_sighup=true)
```

其中 command 参数可以是任意已知的系统命令。例如：

```
chile=pexpect.spawn('/usr/bin/ftp')　　#启动客户端命令
```

当子程序需要参数时，可以使用 python 列表来代替参数：　　（列表代替参数）

```
chile=pexpect.spawn('/usr/bin/ssh',['user@example.com'])
```

参数 timeout 为等待结果的超时时间；参数 maxread 为 pexpect 从终端控制台一次读取的最大字节数，searchwindowsize 参数为匹配缓冲区字符串的位置，默认是从开始位置匹配。

注意：spawn() ，或者说 pexpect 并不会转译任何特殊字符，比如 *字符在 Linux 的 shell 中有特殊含义，但是在 pexpect 中不会转译它们，如果在 linux 系统中想使用这些符号的正确含义就必须加上 shell 来运行，这是很容易犯的一个错误。（不会转译任何特殊字符）

```
chile=pexpect.spawn('/bin/bash -c "ls -l | grep LOG >  logs.txt"')
```

spawn 的选项包括下面这些：

- timeout：指定程序的默认超时时间
- maxread：指定一次性试着从命令输出中读多少数据。
- searchwindowsize：是与 maxread 参数一起合作使用的，它的功能比较微妙，但可以显著减少缓存中有很多字符时的匹配时间。
- logfile：当给 logfile 参数指定了一个文件句柄时，所有从标准输入和标准输出获得的内容都会写入这个文件中（注意这个写入是 copy 方式的），如果指定了文件句柄，那么每次向程序发送指令（process.send）都会刷新这个文件（flush）。
- logfile_read：记录执行程序中返回的所有内容，也就是去掉你发出去的命令，而仅仅只包括命令结果的部分。
- cwd 用来指定命令发送的命令在哪个路径下执行，它一般是用在 send() 系列命令中，比如在 Linux 中，你想在 /etc 目录下执行 ls -l 命令，那么完全不需要用。sendline("cd /etc && ls -l") 这样的方式，而是用 sendline("ls -l", cwd="/etc") 就可以了。
- env：指定环境变量的值，这个值是一个字典，如果你发送的命令要使用一些环境变量，那么可以在这里提供。
- ignore_sighup：是否过滤 SIGHUP 信号。
- delaybeforesend：字符发送延时。

2. Expect()　　（expect() 、等待指定的关键字）

当 spawn() 启动了一个程序并返回程序控制句柄后，就可以用 expect() 方法来等待指定的关键字了。它最后会返回 0 表示匹配到了所需的关键字，如果后面的匹配关键字

是一个列表的话，就会返回一个数字表示匹配到了列表中第几个关键字，从 0 开始计算。

```
process.expect(pattern_list, timeout=-1, searchwindowsize=None)
```

在这里的 searchwindowsize 是在 expect() 方法中真正生效的，默认情况下是 None，也就是每从子进程中获取一个字符就做一次完整匹配，如果子进程的输出很多的话，性能会非常低。如果设置为其他的值，表示从子进程中读取到多少个字符才做一次匹配，这样会显著减少匹配的次数，增加性能。

```
process.expect('[Nn]ame')
```

3. expect_exact()　　（不再使用正则表达式）

它的使用和 expect() 是一样的，唯一不同的就是它的匹配列表中不再使用正则表达式。从性能上来说 expect_exact() 要更好一些。

4. expect_list()　　（只会转换一次）

使用方式和 expect() 一样，唯一不同的就是它里面接受的正则表达式列表只会转换一次。expect() 稍微有点笨，每调用一次它都会将内部的正则表达式转换一次（当然也有其他办法避免）。

5. expect_loop()　　（标准输入、获取内容）

用于从标准输入中获取内容，loop 这个词代表它会进入一个循环，必须要从标准输入中获取到关键字才会往下继续执行。

6. Send()　　（发送字符串）

作为 3 个关键操作之一，用来向程序发送指定的字符串，它的使用没什么特殊的地方，比如：

```
process.expect("ftp>")
process.send("by\n")
```

7. Sendline()　　（回车换行符）

sendline() 和 send() 唯一的区别就是在发送的字符串后面加上了回车换行符。

8. Sendcontrol()　　（发送控制字符）

向子程序发送控制字符，比如 <kbd>ctrl+C</kbd> 或者 <kbd>ctrl+D</kbd> 之类的，比如你要向子程序发送 <kbd>ctrl+G</kbd>，那么就这样写：

```
process.sendcontrol('g')
```

9. Sendeof()　　（End Of File ）

向子程序发送 End Of File 信号。

10. Sendintr()　　（SIGINT）

向子程序发送 SIGINT 信号。

11. Interact()　　（控制权限）

将控制权限交给用户（或者说标准输入）。

12. Close()

停止应用程序。

13. Terminate()

可以看作是上面 close() 的别名，因为不管是功能还是使用方法都是一样的。

14. Kill()

向子程序发送 SIGKILL 的信号。

15. Flush()

什么都不干，只是为了与文件方法兼容而已。

16. Isalive()

检查被调用的子程序是否正在运行，这个方法是运行在非阻断模式下面的。如果获得的返回是 True 表示子程序正在运行；返回 False 则表示程序运行终止。

17. Isatty()

返回 True 表示打开和连接到了一个 tty 类型的设备，或者返回 False 表示未连接。

18. Next() （下一行）

和操作文件一样，这个方法也是返回缓存中下一行的内容。

19. Read() （获取内容）

获取子程序返回的所有内容，一般情况下我们可以用 expect 来期待某些内容，然后通过 process.before 这样的方式来获取，但这种方式有一个前提：那就是必须先 expect 某些字符，然后才能用 process.before 来获取缓存中剩下的内容。

20. Readline() （返回一行输出）

返回一行输出，返回的内容包括最后的\r\n 字符。

21. Readlines() （返回一个列表）

返回一个列表，列表中的每个元素都是一行（包括\r\n 字符）。

22. Setecho()

设置子程序运行时的响应方式，一般情况下向子程序发送字符的时候，这些字符都会在标准输出上显示出来，这样你可以看到你发送出去的内容，但是有的时候，我们不需要显示，那么就可以用这个方法来设置了。

23. Setwinsize() （控制台窗口大小）

设置控制台窗口大小。

24. Wait() （等待执行完毕）

直到被调用的子程序执行完毕之前，程序都停止（或者说等待）执行。它不会从被调用的子程序中读取任何内容。

25. Write() （send、不会返回）

类似于 send()命令，只不过不会返回发送的字符数。

26. Writelines() （类似于 write()、字符串列表）

类似于 write() 命令，只不过接受的是一个字符串列表， writelines() 会向子程序一条一条的发送列表中的元素，但是不会自动在每个元素的最后加上回车换行符。

spawn 示例程序：

```
# This connects to the openbsd ftp site and
# downloads the recursive directory listing.
import pexpect
child = pexpect.spawn('ftp ftp.openbsd.org')
child.expect('Name .*: ')
child.sendline('anonymous')
child.expect('Password:')
child.sendline('noah@example.com')
child.expect('ftp> ')
child.sendline('lcd /tmp')
child.expect('ftp> ')
child.sendline('cd pub/OpenBSD')
child.expect('ftp> ')
child.sendline('get README')
child.expect('ftp> ')
child.sendline('bye')
```

b) Run

1. run 是使用 pexpect 进行封装的调用外部命令的函数，类似 os.system 或 os.popen 方法，不同的是，使用 run()可以同时获得命令的输出结果及命令的退出状态。
   run（）函数比 spawn 简单，对于快速调用程序很有用。 当您调用 run（）函数时，它执行给定的程序，然后返回输出。（run、调用外部命令）

```
Pexpect.run(command,timeout=-1,withexitstatus=False,events=None,
    extra_args=None,logfile=None, cwd=None, env=None)
```

command 是要运行的命令，例如，要执行 svn 命令：

```
from pexpect import *
run ("svn ci -m 'automatic commit' my_file.py")
```

又如：

```
from pexpect import *
child = spawn('scp foo user@example.com:.')
child.expect('(?i)password')
child.sendline(mypassword)
```

上面的代码可以替换为：

```
from pexpect import *
run('scp foo user@example.com:.', events={'(?i)password': mypassword})
```

c) Pxssh

1. pxssh 是 pexpect 的派生类，扩展了 pexpect.spawn 以专门设置 SSH 连接。 这增加了

登录，注销和期待 shell 提示的方法。（pxssh 类、专门设置 SSH 连接、登录、注销）

```
import pxssh
s = pxssh.pxssh()
```

2. Prompt()　　（寻找提示符）

正则表达式搜索寻找提示符。 如果您使用 auto_prompt_reset=False 调用 login()，则必须手动设置此属性。

3. force_password()　（禁用公钥、强制密码）

如果设置为 True，则禁用公钥身份验证，强制服务器请求密码。 请注意，sysadmin 可以禁用密码登录，在这种情况下，这将不起作用。

4. Login()　　（登陆）

登陆到给定的服务器中。

5. Logout()

发送到远程 shell 的退出。如果停止作业，则会自动发送两次退出。

6. sync_original_prompt()

尝试找到提示。基本上，按回车并记录回应；再次输入并记录回应；如果两个回应相似，则假设我们处于原始提示。

pxxsh 示例：

```
from pexpect import pxssh
import getpass
try:
    s = pxssh.pxssh()
    hostname = raw_input('hostname: ')
    username = raw_input('username: ')
    password = getpass.getpass('password: ')
    s.login(hostname, username, password)
    s.sendline('uptime')   # run a command
    s.prompt()             # match the prompt
    print(s.before)        # print everything before the prompt.
    s.sendline('ls -l')
    s.prompt()
    print(s.before)
    s.sendline('df')
    s.prompt()
    print(s.before)
    s.logout()
except pxssh.ExceptionPxssh as e:
    print("pxssh failed on login.")
    print(e)
```

第6章 系统批量运维管理器paramiko详解

1. paramiko是基于python实现的SSH2远程安全连接，支持认证及密钥方式。可以实现远程命令执行、文件传输、中间SSH代理等功能，相对于Pexpect，封装的层次更高，更贴近SSH协议的功能。它提供客户端和服务器功能。（paramiko、SSH2远程安全连接、封装的层次更高）
2. paramiko依赖第三方的crypto、ecdsa包及python-devel的支持。

a) SSHClient类

1. SSHClient类是SSH服务会话的高级表示，该类封装了传输、通道及SFTPClient的校验、建立的方法，通常用于执行远程命令。（SSHClient类、SSH服务会话）
2. Connect()　　（连接）

   实现了远程SSH连接并检验。
3. exec_command()　　（执行）

   远程命令执行方法，该命令的输入与输出流为标准输入、输出、错误输出的python文件对象。
4. load_system_host_key()　　（公钥校验文件）

   加载本地公钥校验文件，默认为~/.ssh/known_hosts，非默认路径需要手工指定。
5. set_missing_host_key_policy()（连接策略）

   设置连接的远程主机没有本地主机密钥或HostKeys对象时的策略，目录支持三种，分别是AutoAddPolicy、RejectPolicy（默认）、WarningPolicy，仅限用于SSHClient类。

b) SFTPClient类

1. SFTPClient作为一个SFTP客户端对象，根据SSH传输协议的sftp会话，实现远程文件操作，比如文件上传、下载、权限、状态等操作。（SFTPClient、SFTP客户端对象）
2. from_transport()　　（创建）

   创建一个已连通的SFTP客户端通道。
3. Put()　　（上传）

   上传本地文件到远程SFTP服务端。
4. Get()　　（下载）

   从远程SFTP服务端下载文件到本地。
5. Mkdir()　　（创建目录）

   在SFTP服务器端创建目录。
6. Remove()　　（删除目录）

   删除SFTP服务器端指定目录。
7. Rename()　　（重命名）

   重命名SFTP服务器端文件或目录。

8. Stat()　　（文件信息）

   获取远程SFTP服务器端指定文件信息。

9. Listdir()　　（目录列表）

   获取远程SFTP服务器端指定目录列表，以列表形式返回。

第7章 系统批量运维管理器Fabric详解

1. Fabric是一种用于简化SSH应用程序部署或系统管理任务的Python（2.5-2.7）库和命令行工具。（fabric、简化、SSH命令行工具）

- Fabric是一种可以通过命令行执行任意Python函数的工具;
- Fabric是一个子程序库（构建在较低级别的库之上），可以通过SSH 方便地执行shell命令和Pythonic 。

2. 首先，来看一个简单的例子：

```
#fabric.py              在当前目录下，命名为fabric.py
def hello():
    print("Hello world!")
```

使用fab命令执行，执行结果如下：

```
$ fab hello
Hello world!

Done.
```

3. fabric支持shell形式的<task name>:<arg>参数形式，例如：

```
def hello(name="world"):
    print("Hello %s!" % name)
```

在当前目录下命名为fabric.py，执行：

```
$ fab hello:name=Jeff
Hello Jeff!

Done.
```

4. fab命令作为fabric程序的命令行入口，提供了丰富的参数调用：

- -l：显示定义好的任务函数名
- -f：指定fab入口文件，默认入口文件名为fabfile.py
- -g：指定网关设备
- -H：指定目标主机，多台主机用“，”分隔
- -P：以异步并行方式运行多主机任务，默认为串行运行
- -R：指定role，以角色名区分不同业务组设备

- -t：设置设备连接超时时间（秒）
- -T：设置远程主机命令执行超时时间
- -w：当命令执行失败，发出告警，而非默认中止任务

5. fab 命令是结合 fabfile.py（其他文件名须添加-f filename 引用）来搭配使用，部分命令行参数可以通过相应的方法来代替，使之更加灵活。
6. env 对象的作用是定义 fabfile 全局设定，支持多个属性，包括目标主机、用户、密码、角色等。（env、fabfile 全局设定）

- Env.host：定义目录主机，可以用 IP 或主机名表示，以列表形式定义。（目录主机）
- Env.exclude_hosts：排除指定主机。
- Env.user：定义用户名。
- Env.port：定义目录主机端口，默认为 22。
- Env.password：定义密码。
- Env.passwords：与 password 功能一样，区别在于不同主机不同密码的应用场景，需要注意的是，配置 passwords 时需配置用户、主机、端口等信息。(不同主机不同密码)
- Env.gateway：定义网关。
- Env.deploy_release_dir：自定义全局变量。
- Env.roledefs：定义角色分组。

7. 常用 API：

(1) local：执行本地命令。

(2) lcd：切换本地目录。

(3) cd：切换远程目录。

(4) run：执行远程命令。

(5) sudo：sudo 方式执行远程命令。

(6) put：上传本地文件到远程主机。

(7) get：从远程主机下载文件到本地。

(8) prompt：获得用户输入信息。

(9) confirm：获得提示信息确认。

(10) reboot：重启远程主机。

(11) @task：函数修饰符，标识的函数为 fab 可调用，非标记对 fab 不可见，纯业务逻辑。

(12) @runs_once：函数修饰符，标识的函数只会执行一次，不受多台主机影响。

8. 完整示例：

```
from __future__ import with_statement
```

```
from fabric.api import *
from fabric.contrib.console import confirm

env.hosts = ['my_server']

def test():
    with settings(warn_only=True):
        result = local('./manage.py test my_app', capture=True)
    if result.failed and not confirm("Tests failed. Continue anyway?"):
        abort("Aborting at user request.")

def commit():
    local("git add -p && git commit")

def push():
    local("git push")

def prepare_deploy():
    test()
    commit()
    push()

def deploy():
    code_dir = '/srv/django/myproject'
    with settings(warn_only=True):
        if run("test -d %s" % code_dir).failed:
            run("git clone user@vcshost:/path/to/repo/.git %s" % code_dir)
    with cd(code_dir):
        run("git pull")
        run("touch app.wsgi")
```

第 8 章 从零开发一个轻量级 WebServer

1. configobj 是一个简单且功能强大的用于读写配置文件的 python 应用接口。提供一个简单的编程接口和一个简单的语法配置文件。（configobj、读写配置文件）
2. 读取配置文件的正常方法是给 ConfigObj 文件名：　　（文件名）

```
from configobj import ConfigObj
config = ConfigObj(filename)
```

可以将配置文件的成员作为字典访问：　　（字典访问）

```
from configobj import ConfigObj
config = ConfigObj(filename)
#
value1 = config['keyword1']
value2 = config['keyword2']
```

```
#
section1 = config['section1']
value3 = section1['keyword3']
value4 = section1['keyword4']
#
# you could also write
value3 = config['section1']['keyword3']
value4 = config['section1']['keyword4']
```

3. 创建一个空的ConfigObj，设置文件名和一些值，然后写入文件：（空的）

```
from configobj import ConfigObj
config = ConfigObj()
config.filename = filename
#
config['keyword1'] = value1
config['keyword2'] = value2
#
config['section1'] = {}
config['section1']['keyword3'] = value3
config['section1']['keyword4'] = value4
#
section2 = {
    'keyword5': value5,
    'keyword6': value6,
    'sub-section': {
        'keyword7': value7
        }
}
config['section2'] = section2
#
config['section3'] = {}
config['section3']['keyword 8'] = [value8, value9, value10]
config['section3']['keyword 9'] = [value11, value12, value13]
#
config.write()
```

4. 将配置文件写入到不同的文件：

```
from configobj import ConfigObj
#
conf_ini = "./test.ini"
config = ConfigObj(conf_ini,encoding='UTF8')
del config['client_srv']['port']
config.filename = "./test1.ini"
config.write()
```

5. Python默认自带的模块已经可以实现简单的HTTP服务器，如 BaseHTTPServer模块提供基本的Web服务和处理器类；SimpleHTTPServer模块包含 GET与 HEAD请求与处理支持；CGIHTTPServer模块包含处理 POST请求的支持。Yorserver是基于BaseHTTPServer模块Web服务类 HTTPServer扩展而来，同时也使用 CGIHTTPServer模块提供CGI程序的接收与执行。（BaseHTTPServer模块、Web服务和处理器类、SimpleHTTPServer模块、 GET与 HEAD、CGIHTTPServer模块、POST）

6. HTTP缓存机制：

(1) Expires 机制：在 HTTP/1.1协议中，Expires字段声明了一个网页或URL地址不再被浏览器缓存的时间，一旦超过了这个时间，浏览器会重新向原始服务器发起新请求。

(2) max-age 机制：客户端另一缓存机制则是利用HTTP消息头中的“cache-control”来控制，其中max-age字段实现在原始服务器返回的max-age配置的秒数内，浏览器将不会发送相关请求到服务器，而是由缓存直接提供，超过这一时间段后才向原始服务器发起请求，由服务器决定返回新数据还是仍由缓存提供。与 Expires不同，max-age是通过指定相对时间秒数来实现缓存过期，当与Expires同时存在时，max-age会覆盖Expires。

(3) Last-Modified 机制：最后一种浏览缓存机制为Last-Modified，其原理是客户端通过if-Modified-Since请求头将先前接收到服务器端文件的Last-Modified时间戳信息进行发送，目的是让服务器端进行比对验证，通过这个时间戳判断客户端的文件是否是最新，如不是最新的，则返回新的内容(HTTP 200)，如果是最新的，则返回HTTP 304告诉客户端其本地缓存的文件是最新的，无需重启下载。于是客户端就可以直接从本地加载文件了。

7. 启用HTTP内容压缩，可为我们节省不少带宽成本，并且也可以加快网页访问速度，提升用户体验。目前主流的浏览器都支持客户端解压功能。（HTTP内容压缩）

8. HTTPS是以安全为目标的 HTTP通道，可以理解成 HTTP的安全版，即 HTTP协议下加入 SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL (Secure Sockets Layer，安全套接层）。目前HTTPS广泛用于互联网上安全敏感的通信，例如电商在线交易支付方面。

第9章 集中化管理平台Ansible详解

1. Ansible 是一个系统自动化工具，可以用来做系统配置管理，批量对远程主机执行操作指令。（自动化、配置管理）

2. 安装Ansible之后，不需要启动或运行一个后台进程，或是添加一个数据库。只要在一台电脑(可以是一台笔记本)上安装好，就可以通过这台电脑管理一组远程的机器.在远程被管理的机器上，不需要安装运行任何软件，因此升级Ansible版本不会有太多问题。（不需要、只要）

3. 通常我们使用 ssh 与托管节点通信，默认使用 sftp。如果 sftp 不可用，可在ansible.cfg 配置文件中配置成 scp 的方式. 在托管节点上也需要安装 Python 2.4或以上的版本.如果版本低于 Python 2.5 ，还需要额外安装一个模块：

```
python-simplejson
```

第 11 章 统一网络控制器 Func 详解

1. Func 是由红帽子公司以 Fedora 平台构建的统一网络控制器，是为解决集群管理、监控问题而设计开发的系统管理基础框架。它是一个能有效简化多服务器系统管理工作的工具，它易于学习、使用和扩展，功能强大，只需要极少的配置和维护操作。Func 分为 master 和 slave 两部分， master 为主控端，slave 为被控端。（Func、统一网络控制器）

附录 SSL

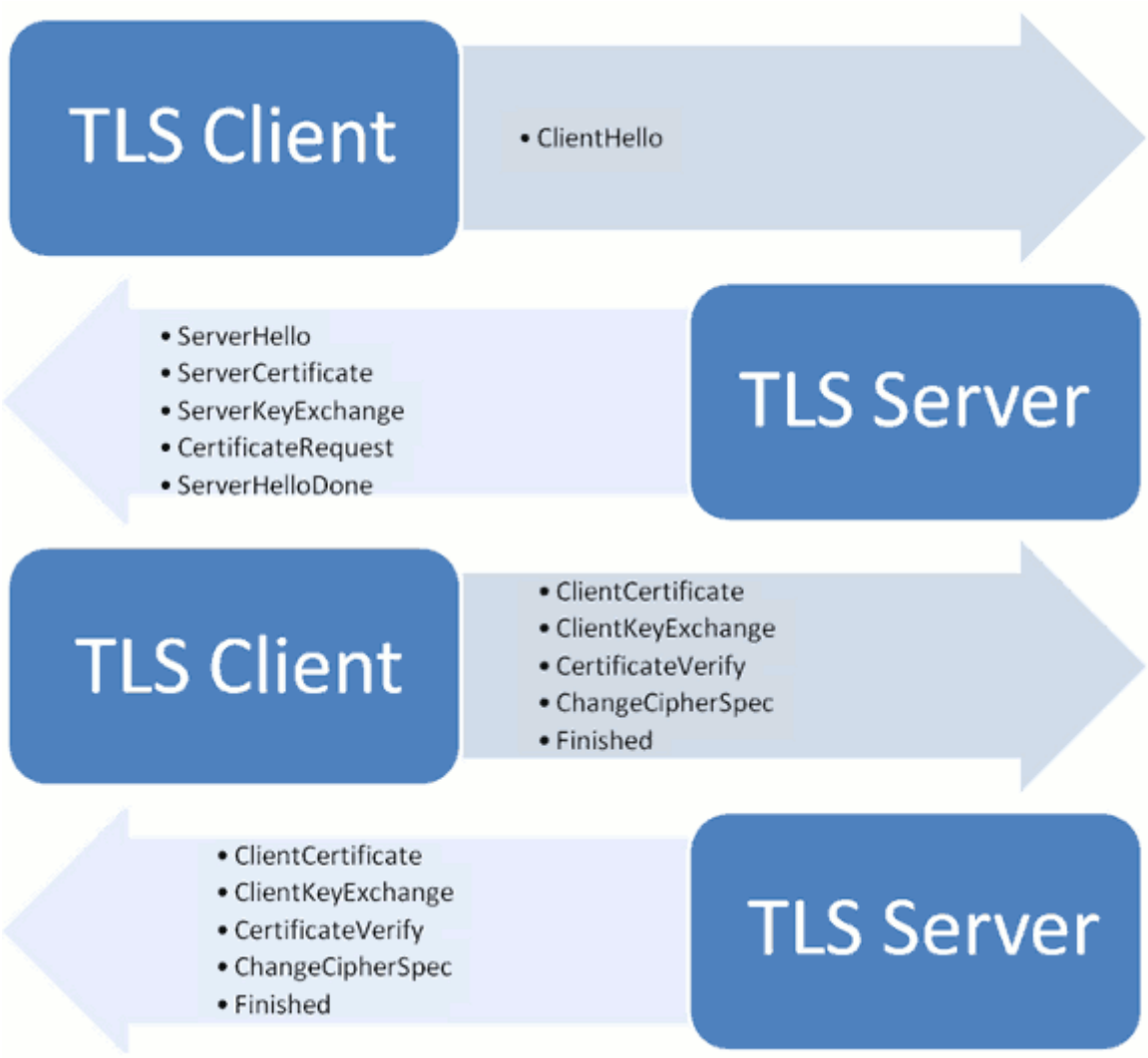
1. SSL/TLS 协议的基本思路是采用公钥加密法，也就是说，客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密。（公钥加密法、私钥解密）
2. 为了防止公钥被篡改，将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。（防止篡改、放在数字证书）
3. 公钥加密计算量太大，为了减少耗用的时间，每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。
4. 因此，SSL/TLS 协议的基本过程是这样的：

   （1） 客户端向服务器端索要并验证公钥。

   （2） 双方协商生成"对话密钥"。

   （3） 双方采用"对话密钥"进行加密通信。

   上面过程的前两步，又称为"握手阶段"（handshake）。
5. "握手阶段"涉及四次通信。需要注意的是，"握手阶段"的所有通信都是明文的。


TLS Client
• ClientHello
• ServerHello
• ServerCertificate
• ServerKeyExchange
• CertificateRequest
• ServerHelloDone
TLS Server
TLS Client
• ClientCertificate
• ClientKeyExchange
• CertificateVerify
• ChangeCipherSpec
• Finished
• ClientCertificate
• ClientKeyExchange
• CertificateVerify
• ChangeCipherSpec
• Finished
TLS Server