孙聪的主页

欢迎来到【孙聪】的空间！

phone: 15257117182

E-mail: suncong@uyunsoft.cn

最近的更新

- Tornado异步编程学习
  昨天17:52:40 • 孙聪更新 • 查看变动
- ant-agent
  昨天10:26:45 • 孙聪更新 • 查看变动
- 事件台时间对齐
  昨天09:59:51 • 孙聪更新 • 查看变动
- 算法
  昨天09:04:00 • 孙聪创建
- testWeb-log.py
  2016-11-29 • 孙聪添加了附件
- MsgPack vs Json
  2016-11-29 • 孙聪创建
- compress1.png
  2016-11-29 • 孙聪添加了附件
- compress2.png
  2016-11-29 • 孙聪添加了附件
- msgpack_example.png
  2016-11-29 • 孙聪添加了附件
- ab_test.png
  2016-11-29 • 孙聪添加了附件
- tornado搭建简单服务器性能(对比Go和Node.js)
  2016-11-28 • 孙聪更新 • 查看变动
- tornado搭建简单服务器性能(对比Go和Node.js)
  2016-11-24 • 竺夏栋更新 • 查看变动
- four_process_tornado.png
  2016-11-24 • 孙聪添加了附件
- one_process_tornado.png
  2016-11-24 • 孙聪添加了附件
- Open-falcon agent
  2016-11-24 • 蒋君伟发表了评论

Navigate space

ant-agent

MsgPack vs Json

MsgPack

特点

MessagePack简称msgpack，是一种高效的二进制序列化格式，类似于Json，但是比Json更快以及更小。小的整数会被编码成一个字节，短的字符串仅仅只需要比它的长度多一字节的大小。

msgpack完全兼容json的数据格式

比json的序列化更省时间和空间

支持多种语言

为啥会小

安装官网给出的例子，有一段json数据，长度为27个字节，json为了保持其数据格式需要一些辅助字符比如 { } " : 等等，造成了数据的冗余。

```json
{"compact":true,"schema":0}
```

对于msgpack格式：

msgpack对数字，多字节字符，数组都做了优化，减少了无用的字符，压缩了存储空间。

对于无符号整型： <128，则直接1个字节；

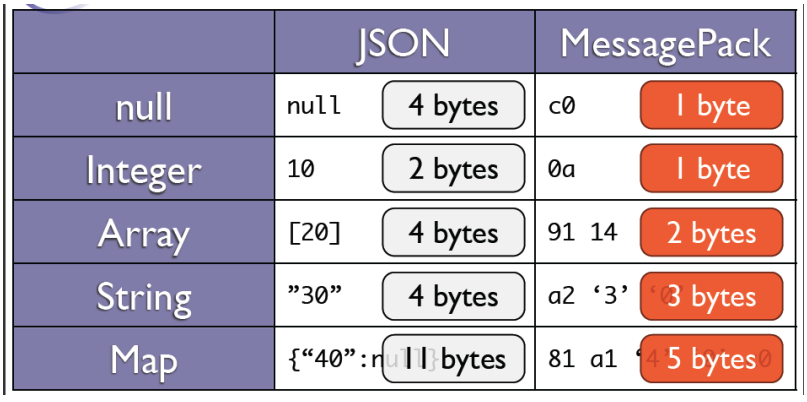
[128,255]，则{0xcc, N} 2个字节表示

[256,65535], 则{0xcd, N} 3个字节表示。（因为N介于[256,65535]， 所以需要2个字节； 总共3个字节）

[65536, 2^31-1] {0xce, N}----------5个字节

uint32_max时，{0xcf, N}----------9个字节。

	JSON	MessagePack
null	null	c0
Integer	10	0a
Array	[20]	91 14
String	”30”	a2 ‘3’ ‘0’
Map	{“40”:null}	81 a1 ‘4’ ‘0’ c0



所以，同样是上面27个字节的json数据，msgpack格式只需要18个字节。

为啥会快

JSON存储的时候是采用链表存储的，访问一颗树。每一个节点有兄妹节点，通过next/prev指针来查找，它类似双向链表；每个节点也可以有孩子节点，通过child指针来访问，进入下一层。

MessagePack
遍历过去了，从前面的数据头，就可以知道后面的是什么数据，指针应该向后移动多少，比JSON的构建链表少了很多比较的过程。

性能对比

测试环境

操作系统：ubuntu12.04 64bit

内存：2G

CPU：单核 3.2GHz

测试指标

speed

size

1.开启datadog-agent运行十分钟，分别以json和msgpack格式传输数据，自写一个小抓包工具，testWeb-log.py。观察两种方式在10分钟内产生数据包的大小。

mspack：733kb

json： 1136kb

2.随机生成包含10万个元素的字典，用sys.getsizeof方法查看字典的大小。

mspack：1050kb

json：1577kb

参考

官方网站：http://msgpack.org/

github：https://github.com/msgpack

tornado搭建简单服务器性能(对比Go和Node.js)

- 简单搭建
    - 本机环境
    - code
- 压力测试
    - 工具：siege
    - 安装：
    - 参数：
- 测试结果
    - 单进程
    - 四进程
    - ab测试结果
- 性能对比

简单搭建

本机环境

操作系统：ubuntu12.04 64bit

内存：2G

CPU：单核 3.2GHz

code

- 写了一个简单的hello world服务器，请求端口救回返回 hello tornado这个字符串；
- 定义了两个方法分别是：one_process和multi_process，分别用单进程和多进程的方式启动服务；

hello world

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

import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web

from tornado.options import define, options
define("port", default=8000, help="run on the given port", type=int)

class IndexHandler(tornado.web.RequestHandler):
    def get(self):
        greeting = self.get_argument('greeting', 'Hello')
        self.write(greeting + ', tornado!')

def one_process():
    tornado.options.parse_command_line()
    app = tornado.web.Application(handlers=[(r"/", IndexHandler)], debug = True)
    http_server = tornado.httpserver.HTTPServer(app)
    http_server.listen(options.port)
    tornado.ioloop.IOLoop.instance().start()

def multi_process():
    tornado.options.parse_command_line()
    app = tornado.web.Application(handlers=[(r"/", IndexHandler)], debug = False)
    http_server = tornado.httpserver.HTTPServer(app)
    http_server.bind(options.port, options.host)
    http_server.start(num_processes=4) # tornado将按照cpu核数来fork进程
    tornado.ioloop.IOLoop.instance().start()

if __name__ == "__main__":
    multi_process()
```

压力测试

工具：siege

安装：

install seige

```
# install siege
wget http://download.joedog.org/siege/siege-2.72.tar.gz
tar zxvf siege-2.72.tar.gz
cd siege-2.72
./configure --prefix=/usr/local/siege --mandir=/usr/local/man --with-ssl=/usr/local/ssl
mkdir -p /usr/local/siege/etc/
mkdir -p /usr/local/siege/var/
make
make install
```

参数：

argms

```
-c 200 指定并发数200
-r 5 指定测试的次数5
-f urls.txt 制定url的文件
-i internet系统，随机发送url
-b 请求无需等待 delay=0
-t 5 持续测试5分钟
# -r和-t一般不同时使用
```

测试：1000个并发请求10次

test

```
/usr/local/siege/bin/siege -c 1000 -r 10 http://localhost:12345/
```

测试结果

测试1000个并发，每个并发发送10次请求的情况。

单进程

```
** SIEGE 2.72
** Preparing 1000 concurrent users for battle.
The server is now under siege..      done.

Transactions:                  10000 hits
Availability:                 100.00 %
Elapsed time:                  21.63 secs
Data transferred:               0.14 MB
Response time:                  0.52 secs
Transaction rate:             462.32 trans/sec
Throughput:                     0.01 MB/sec
Concurrency:                  239.06
Successful transactions:       10000
Failed transactions:               0
Longest transaction:           13.50
Shortest transaction:           0.00
```

吞吐率，吞吐率特指Web服务器单位时间内处理的请求数：0.01 MB / sec

吞吐量：0.14 MB

TPS：462 trans / sec

响应时间：0.52 secs

实际处理最高并发连接数：239

四进程

```
** SIEGE 2.72
** Preparing 1000 concurrent users for battle.
The server is now under siege..      done.

Transactions:                  10000 hits
Availability:                 100.00 %
Elapsed time:                  23.28 secs
Data transferred:               0.14 MB
Response time:                  0.62 secs
Transaction rate:             429.55 trans/sec
Throughput:                     0.01 MB/sec
Concurrency:                  267.66
Successful transactions:       10000
Failed transactions:               0
Longest transaction:           15.03
Shortest transaction:           0.00
```

吞吐率，0.01 MB / sec

吞吐量：0.14 MB

TPS：429 trans / sec

响应时间：0.62 secs

实际处理最高并发连接数：267

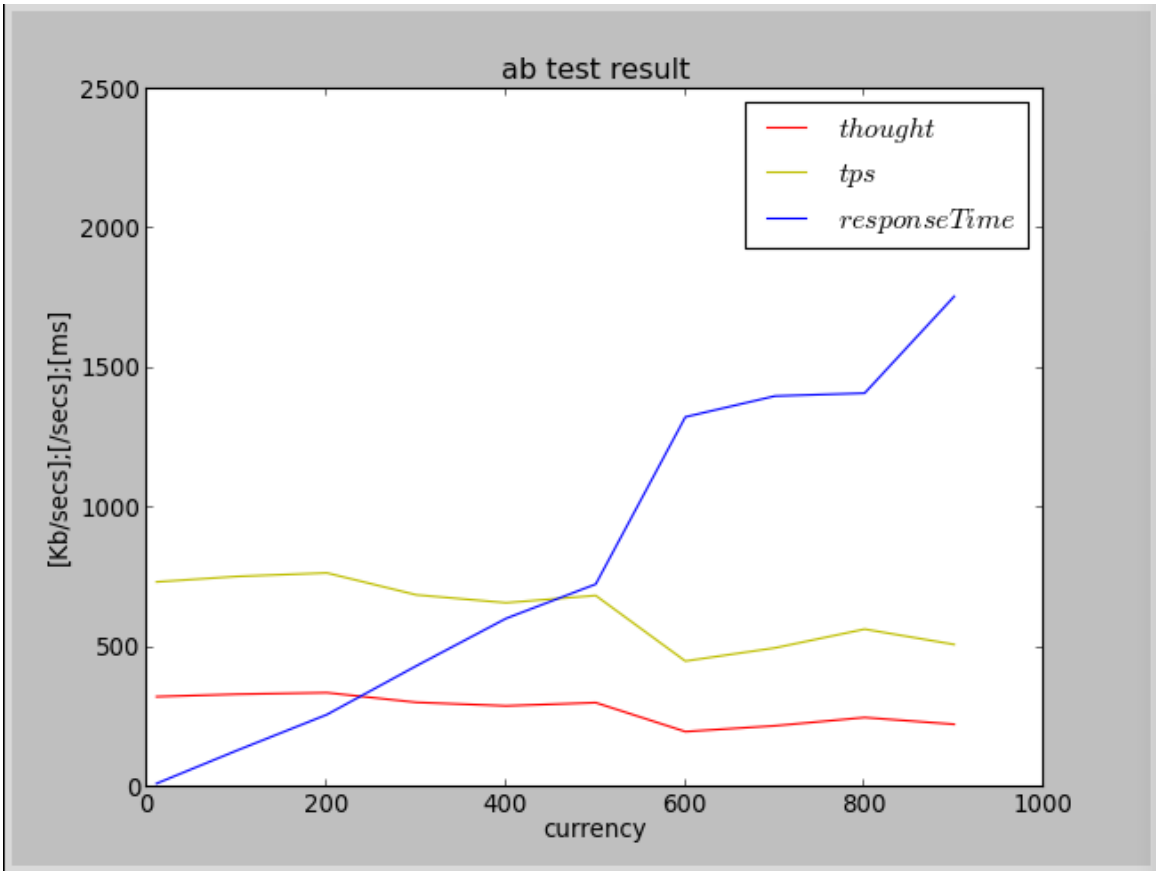
并发数	吞吐率	TPS	响应时间	
1	0.00 MB/s	1.99 trans/sec	0.00 secs	
10	0.00 MB/s	12.39 trans/sec	0.01 secs	
50	0.00 MB/s	55.01 trans/sec	0.01 secs	
100	0.00 MB/s	99.21 trans/sec	0.01 secs	
200	0.00 MB/s	218.10 trans/sec	0.03 secs	
500	0.01 MB / sec	457.88 trans/sec	0.12 secs	
800	0.01 MB / sec	405.27 trans/sec	0.53 secs	
1000	0.01 MB / sec	429 trans / sec	0.62 secs	

ab测试结果

随机生成的256个字节的get请求

并发数	吞吐率 [Kbytes/sec]	TPS [#/sec]	响应时间 [ms]
1	169.88	384.87	2.598

10	324.79	735.81	13.590
20	414.02	937.96	21.323
30	359.70	814.89	36.815
40	340.43	771.24	51.865
50	309.97	702.23	71.202
60	365.63	828.32	72.436
70	373.61	846.41	82.703
80	331.35	750.67	106.571
90	321.45	728.25	123.584
100	333.51	755.55	132.353
200	339.00	768.00	260.416
300	304.27	689.32	435.210
400	291.85	661.17	604.987
500	303.24	686.98	727.826
600	199.62	452.23	1326.759
700	220.43	499.39	1401.710
800	250.10	566.59	1411.958
900	225.87	511.71	1758.826



性能对比

没能安装Go和Node.js的环境实地

Tornado异步编程学习

tornado.ioloop

tornaod优秀的大并发处理能力得益于它底层实现了epoll的单线程异步架构。

ioloop 实际上是对 epoll 的封装，并加入了一些对上层事件的处理和 server 相关的底层处理

poll：多个连接可以统一在一定时间内轮流看一遍里面有没有数据要读写，处理多个连接了，这种方式就是 poll / select 的。

epoll：linux内核为处理大批量文件描述符而改进了poll；随着连接越来越多，轮询所花费的时间将越来越长，而服务器连接的 socket 大多不是活跃的，轮询所花费的大部分时间将是无用的。epoll在每次轮询时，会把有数据活跃的 socket 挑出来轮询，节省了大量时间。

epoll 的操作，有4个API：

- epoll_create
- epoll_ctl
- epoll_wait
- close

其中，操作 epoll 中的 event；可用参数有：

参数	含义
EPOLL_CTL_ADD	添加一个新的epoll事件
EPOLL_CTL_DEL	删除一个epoll事件
EPOLL_CTL_MOD	改变一个事件的监听方式

epoll_wait 操作则在 start()中

接口

add_handler

add_handler用于添加socket到主循环中，有三个参数，fd是socket文件描述符，handler是处理此socket的callbac函数，events是此socket注册的事件。

add_handler

```
def add_handler(self, fd, handler, events):
    self._handlers[fd] = stack_context.wrap(handler)
    self._impl.register(fd, events | self.ERROR)
```

update_handler

update_handler用于更新主循环中已存在的socker事件。

update_handler

```
def update_handler(self, fd, events):
    self._impl.modify(fd, events | self.ERROR)
```

remove_handler

remove_handler用户移除主循环已经存在的socket事件

remove_handler

```
def remove_handler(self, fd):
    self._handlers.pop(fd, None)
    self._events.pop(fd, None)
    try:
        self._impl.unregister(fd)
    except Exception:
        gen_log.debug("Error deleting fd from IOLoop", exc_info=True)
```

实例

example

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

import Queue
import socket

from functools import partial

from tornado.ioloop import IOLoop

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setblocking(0)             # 将socket设置为非阻塞

server_address = ("localhost", 10000)

sock.bind(server_address)
sock.listen(5)
#指明可接受的最大连接数

fd_map = {}             # 文件描述符到socket的映射
message_queue_map = {}  # socket到消息队列的映射

fd = sock.fileno()      # 返回套接字的文件描述符
fd_map[fd] = sock

ioloop = IOLoop.instance()

def handle_client(cli_addr, fd, event):
    s = fd_map[fd]
    if event & IOLoop.READ:
        data = s.recv(1024)   #接收最大1024字节
        if data:
            print "      received '%s' from %s" % (data, cli_addr)
            # 接收到消息更改事件为写, 用于发送数据到对端
            ioloop.update_handler(fd, IOLoop.WRITE)
            message_queue_map[s].put(data)
        else:
            print "      closing %s" % cli_addr
            ioloop.remove_handler(fd)
            s.close()
            del message_queue_map[s]

    if event & IOLoop.WRITE:
        try:
```

```
            next_msg = message_queue_map[s].get_nowait()
        except Queue.Empty:
            print "%s queue empty" % cli_addr
            ioloop.update_handler(fd, IOLoop.READ)
        else:
            print 'sending "%s" to %s' % (next_msg, cli_addr)
            s.send(next_msg)

    if event & IOLoop.ERROR:
        print " exception on %s" % cli_addr
        ioloop.remove_handler(fd)
        s.close()
        del message_queue_map[s]


def handle_server(fd, event):
    s = fd_map[fd]
    if event & IOLoop.READ:
        conn, cli_addr = s.accept()
        print "     connection %s" % cli_addr[0]
        conn.setblocking(0)
        conn_fd = conn.fileno()
        fd_map[conn_fd] = conn
        handle = partial(handle_client, cli_addr[0])   # 将cli_addr作为第一个参数
        # 将连接和handle注册为读事件加入到 tornado ioloop
        ioloop.add_handler(conn_fd, handle, IOLoop.READ)
        message_queue_map[conn] = Queue.Queue()   # 创建对应的消息队列
```

```
ioloop.add_handler(fd, handle_server, IOLoop.READ)
ioloop.start()
```

QA

1. update_handler接口既然是update，为什么参数不需要传handler函数进来。
2. socket.listen(5)指定了最大连接数是5，大于5直接拒绝么？

参考文献

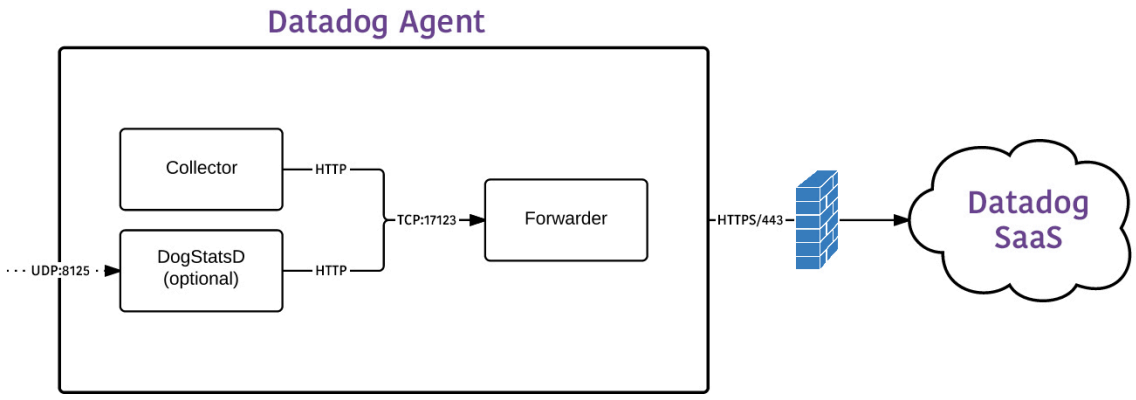
https://www.linuxzen.com/shi-yong-tornadojin-xing-wang-luo-yi-bu-bian-cheng.html

http://python.jobbole.com/85365/

dd-agent

agent forwarder缓存机制

Datadog Agent的数据流



Collector (agent.py)

Collector 会检查当前运行机器的集成环境，抓取系统性能指标，如内存和 CPU 数据。

agent.py 执行 checks.Collector.py 里面collector的run方法

Dogstatsd (dogstatsd.py)

这是 StatsD 的后台服务器，它致力于收集代码中发送出去的本地性能指标。

dogstatsd.py 的report的run方法

Forwarder (ddagent.py)

Forwarder 负责把 Dogstatsd 和 Collector 收集起来，这些数据将会被发往 Datadog server。

Forwarder 监听TCP端口，接收 Dogstatsd 和 Collector 发送来的数据

forwarder接着通过 HTTPS 转发到 Datadog 中心。

```
ddagent:Application                                    #接收各种类型的数据
    handlers = [
        (r"/intake/?", AgentInputHandler),                    #接收其他数据类型
        (r"/intake/metrics?", MetricsAgentInputHandler),      #接收metric数据类型
        (r"/intake/metadata?", MetadataAgentInputHandler),    #接收metadata数据类型
        (r"/api/v1/series/?", ApiInputHandler),               #接收series
        (r"/api/v1/check_run/?", ApiCheckRunHandler),         #接收service check
        (r"/status/?", StatusHandler),                        #反应transaction发送状态
      ]
```

ddagent.py 运行Application类的run方法来从collector和dogstatd接收数据。

transaction.py中的TransactionManager类向datadog推送数据，HTTPS协议

Forwarder 缓存

场景：Datadog server挂了或者没开机。

当forwarder与server连接正常时，forwarder将收集来的数据就缓存在内存中，直到内存中有最低数据发送量的数据才向server发送，发送完的数据会丢弃。

主要在transaction.py中的TransactionManager类实现

_MAX_WAIT_FOR_REPLAY　参数定义了前一个transaction错误后最大时间间隔，默认90秒

_MAX_QUEUE_SIZE　　参数定义了forwarder最多缓存数据的大小，达到这个阈值，老的数据就会被丢弃，默认是30MB

_THROTTLING_DELAY　　这个参数还不清楚作用，2毫秒

如果datadog
server挂掉了，那么forwarder和server间的HTTP连接会断掉，forwarder仍然会不停的封装数据并在_MAX_WAIT_FOR_REPLAY时间后想server发送，直到数据达到_MAX_QUEUE_SIZE最大值，此时就会丢弃时间最久的数据

想让agent缓存更多的数据，最简单的办法就是设置_MAX_QUEUE_SIZE参数更大一点。

agent 集成自动更新

需求

1.集成agent自动更新功能

2.agent端的文件需与server端的文件保持一致

3.服务器上agent有文件修改，添加，删除时，agent通过api自动进行更新，添加，删除操作

实现

思路

缓存上次更新文件的MD5值，以json形式存储，文件名为：last_md5.json

缓存之前agent对应的版本号，文件名为：version

step1：调用client/version api，对比和本地存储的版本号，如果一样，说明不需要更新，程序结束。如果不同执行step2；

step2：调用client/file-v2 api，下在当前server端文件的详细信息，保存为filelist.json；

step3：对比filelist.json和last_md5.json，找出其中MD5值不一样的文件(需要更新)，last_md5.json中没有的文件(需要添加)，filelist.json中没有的文件(需要删除)；

step4：根据step3的结果调用client/file/${filename} api 下载需要更新和添加的文件到本地agent；

step5：agent在本地删除last_md5.json中有filelist.json中没有的文件；

step6：更新last_md5.json文件，保证last_md5.json中的信息与当前agent所有文件信息一致。

step7：结束。

服务端

服务端提供4个接口

1. client： 获取指定client的所有fileset定义
2. client/version： 检查指定client的当前版本号
3. client/file-v2： 获取client当前所有文件清单与详细信息
4. client/file/${filename}： 下载指定文件

agent端

1. 更新部分

do_update

```
def do_update(self):
    # 1.compare version to local version
    # 2.if server's version == agent's version, then return
    # 3.else replace agent's version with server's version and do update
        version = self.download_version()
        if version == self.version:
            log.info("agent still lastest, no need to update")
            return
        try:
            f = open(self.version_file_path, 'wb')
            pickle.dump(version, f)
            f.close()
            log.info("current agent version is updated %s, now: " %version)
        except IOError:
            log.critical("could not update version, update script terminated")
            return
        try:
            self.download_file_v2()
            f = open(self.filelist_path, 'rb')
            self.filelist = pickle.load(f)
            self.filelist = eval(self.filelist)
            for file_info in self.filelist["files"]:
                f_name = file_info["name"]
                if self.last_md5.has_key(f_name) and self.last_md5[f_name] == file_info["md5"]:
                    self.next_md5[f_name] = file_info["md5"]
                    continue
                # update conf.d or checks.d file
                if "conf.d" in f_name:
                    f_dir = self.dirs["conf"]
                else:
                    f_dir = self.dirs["check"]
                f_md5 = self.get_file_md5(f_dir + f_name)
                # agent add new file
                if "File Not Exist" == f_md5:
                    log.debug("agent add new file %s" %f_name)
                    self.download_file(f_dir, f_name)
                # agent update file
                elif file_info["md5"] <> f_md5:
                    log.debug("file %s updated" %f_name)
                    self.download_file(f_dir, f_name)
                self.next_md5[f_name] = file_info["md5"]
            # agent remove old file
            self.remove_extra_file()
            self.store_md5_for_next_use()
        except KeyError:
            log.error("error on update agent, self.last_md5 has no key")
        except IOError:
            log.error("load file list error")
        except Exception,e:
            log.error("error on update agent %s" %e)
```

2. 4个api

api

```
def download_client(self):
        try:
            r = requests.get(self.urls["client"])
            return r.content
        except IOError:
            log.error("download client error")
            return "Error"

    def download_version(self):
        try:
            r = requests.get(self.urls["client_version"])
            return r.content
        except IOError:
            log.error("download version error")
            return "Error"

    def download_file_v2(self):
        try:
            r = requests.get(self.urls["client_file_v2"])
            f = open(self.filelist_path, 'wb')
            pickle.dump(r.content, f)
            f.close()
            log.debug("---------download new filelist-------------")
            return r.content
        except IOError:
            log.error("download filelist error")
            return "Error"

    def download_file(self, f_dir = None, filename = None, block_size = 10 * 1024):
        try:
            url = self.urls["cilent_download"] + base64.b64encode(filename) # BASE64 encode
            f = f_dir + filename
            r = requests.get(url, stream=True)
            with open(f, 'wb') as g:
                for part in r.iter_content(chunk_size=block_size):
                    if part:
                        g.write(part)
            log.debug("----------update %s----------" %filename)
            return "Success"
        except IOError:   # no such dir, create dir
            if "/" in filename:
                rf = f[::-1]
                pos = rf.index("/")
                dirs = f[:-pos]
                name = f[-pos:]
                if False == os.path.exists(dirs):
                    os.makedirs(dirs)
            open(name, "wb")
            self.download_file(f_dir, filename)
        except Exception,e:
            log.error("download file error, %s" %e)
            return "Error"
```

3.存储本次MD5值

md5

```
def store_md5_for_next_use(self):
    # store md5 for next use
        try:
            f = open(self.last_md5_file_path, 'wb')
            pickle.dump(self.next_md5, f)
            f.close()
            log.info("----------store md5 file completed for next use----------")
        except Exception,e:
            log.error("error on store md5. %s" %e)

    def remove_extra_file(self):
    # compare self.last_md5(store all agent file), self.next_md5 (store all server file), find the difference
        try:
            to_remove = list(set(self.last_md5.keys()).difference(set(self.next_md5)))
            for fname in to_remove:
                if "conf.d" in f_name:
                    f_dir = self.dirs["conf"]
                else:
                    f_dir = self.dirs["check"]
                os.remove(f_dir + fname)
                log.info("remove extra file %s" %fname)
        except Exception,e:
            log.error("remove extra file error, %s" %e)
```

集成

Linux (supervisord)

1. 需要为自动更新添加一个fork进程，命名为updater_process。
2. 集成到supervisord里面，由supervisord管理自动此updater_process进程。
3. updater_process每隔一段时间会运行自动更新，并且重启服务。

配置supervisor.conf，添加fork进程的配置，并且加入到datadog-agent组当中去。

supervisor.conf

```
[program:updater]
command=python updater_process.py
stdout_logfile=updater.log
redirect_stderr=true
priority=999
startsecs=3

[group:datadog-agent]
programs=forwarder,collector,dogstatsd,jmxfetch,updater
```

Windows (watchdog)

1.集成到watchdog中，在win32/agent.py中添加自动更新的守护进程。

2.由ProcessWatchDog再建一个自动更新的子进程。

3.间隔一段时间执行自动更新并且重启服务。

watchdog

```
class UpdaterProcess(multiprocessing.Process):
    def __init__(self, config):
        multiprocessing.Process.__init__(self, name='updater')
        self.is_enabled = True
        self.running = True
        self.config = config['instances'][0]
        self.interval = self.config['interval']
        self.server = self.config['server']
        self.dirs = self.config['dirs']

    def run(self):
        from config import initialize_logging
        initialize_logging('windows_updater')
        log.debug("Windows Service - Starting updater")
        while self.running:
            time.sleep(self.interval)
            try:
                a = AgentUpdater("Windows", self.server, self.dirs)
                a.do_update()
                os.system("cd C:/Program Files (x86)/Datadog/Datadog Agent/files")
                os.system("ddagent.exe restart")
                log.info("update agent success")
            except Exception,e:
                log.info("update agent failed")

    def stop(self):
        log.debug("Windows Service - Stopping updater")
        self.running = False
        pid = os.getpid()
        os.kill(pid, 9)
        log.debug("Windows Service - Stopped updater - Exit by command")
```

测试

Linux

测试点1：服务端更新，添加，删除文件，自动更新进程能否正确获取并响应；

测试点2: 是否成功集成进supervisord，定期执行自动更新并且重启；

测试点3：更新过的文件，是否可以正确运行；

日志显示测试点达成，第3点在collector添加了一条打印日志的动作也可以成功输出到日志中。

```
2016-09-21 14:06:56,373 INFO success: updater entered RUNNING state, process has stayed up for > than 5 seconds (startsecs)
2016-09-21 14:07:22,422 DEBG 'updater' stderr output:
2016-09-21 14:07:22,416 | INFO | dd.updater | update.updater(updater.py:42) | >>>>>>>>>>>>>>>>>>Linux agent start update<<<<<<<<<<<<<<<<<<
2016-09-21 14:07:22,424 DEBG 'updater' stderr output:
2016-09-21 14:07:22,424 | INFO | dd.updater | requests.packages.urllib3.connectionpool(connectionpool.py:203) | Starting new HTTP connect
2016-09-21 14:07:22,462 DEBG 'updater' stderr output:
2016-09-21 14:07:22,462 | INFO | dd.updater | update.updater(updater.py:130) | current agent version is updated "160921.140726", now:
2016-09-21 14:07:22,464 DEBG 'updater' stderr output:
2016-09-21 14:07:22,463 | INFO | dd.updater | requests.packages.urllib3.connectionpool(connectionpool.py:203) | Starting new HTTP connect
2016-09-21 14:07:22,516 DEBG 'updater' stderr output:
2016-09-21 14:07:22,516 | INFO | dd.updater | requests.packages.urllib3.connectionpool(connectionpool.py:203) | Starting new HTTP connect
2016-09-21 14:07:22,520 DEBG 'updater' stderr output:
2016-09-21 14:07:22,520 | INFO | dd.updater | update.updater(updater.py:168) | ----------store md5 file completed for next use----------
2016-09-21 14:07:22,525 DEBG 'updater' stdout output:
2016-09-21 14:07:22,525 DEBG killing updater (pid 8831) with signal SIGTERM
2016-09-21 14:07:22,526 INFO waiting for forwarder, updater, dogstatsd, collector to die
2016-09-21 14:07:22,529 DEBG fd 12 closed, stopped monitoring <POutputDispatcher at 140261005887392 for <Subprocess at 140261005774648 wi
2016-09-21 14:07:22,529 DEBG fd 14 closed, stopped monitoring <POutputDispatcher at 140261005887104 for <Subprocess at 140261005774648 wi
2016-09-21 14:07:22,529 INFO stopped: updater (terminated by SIGTERM)
2016-09-21 14:07:23,035 INFO spawned: 'updater' with pid 9042
2016-09-21 14:07:24,002 DEBG 'updater' stderr output:
2016-09-21 14:07:24,002 | INFO | dd.updater | updater(test_fork.py:41) | -----start fork ------
2016-09-21 14:07:24,210 | INFO | dd.collector | collector(agent.py:145) | suncong updated
2016-09-21 14:07:28,293 INFO success: updater entered RUNNING state, process has stayed up for > than 5 seconds (startsecs)
```

Windows

文件缓存

目的：

为了能够更好的集成进dd-agent。自动更新的缓存文件和dd-agent自己的文件隔离开来。

linux下需要更新的目录为/opt/datadog-agent/agent/和/etc/dd-agent/conf.d/；

windows下需要更新目录为C:/Program Files (x86)/Datadog/Datadog Agent/checks.d/和C:/ProgramData/Datadog/conf.d/

pickle：

参考文档：https://docs.python.org/2/library/pickle.html

pickle提供了一个简单的持久化功能。可以将对象以文件的形式存放在磁盘上。

pickle.dump(obj, file[, protocol])

序列化对象，并将结果数据流写入到文件对象中。参数protocol是序列化模式，默认值为0，表示以文本的形式序列化。protocol的值还可以是1或2，表示以二进制的形式序列化。

pickle.load(file)

反序列化对象。将文件中的数据解析为一个Python对象。注解：从file中读取一个字符串，并将它重构为原来的python对象。

实现

pickle

```
def _get_pickle_path(cls, name):
        if Platform.is_win32():
            path = os.path.join(_windows_commondata_path(), 'Datadog')
        elif os.path.isdir(PidFile.get_dir()):
            path = PidFile.get_dir()
        else:
            path = tempfile.gettempdir()
        return os.path.join(path, name + '.pickle')

-------------------------
        # load
        self.filelist_path = self._get_pickle_path("filelist")
        f = open(self.filelist_path, 'rb')
        self.filelist = pickle.load(f)
-------------------------
        #dump
        f = open(self.last_md5_file_path, 'wb')
        pickle.dump(self.next_md5, f)
        f.close()
```

问题

1. linux下自动更新集成进了supervisord，用户是dd-agent，执行更新时需要有读写权限，如果dd-agent没有写权限，更新会失败。

2. windows下面同样需要root权限才能执行。

ddagent第三方包打包和setup.py编写

linux下以adodbapi为例

1.cd ~/linux-packages/adodbapi
#进入整理好的adodbapi包代码中

2.mv 2.0.6.7 adodbapi
#将文件夹名字改为模块名，这边不改的话，Import的时候可能会找不到包名

3.vim setup.py
#编辑setup.py

4.python setup.py sdist
#执行打包命令，会创建一个dist目录，里面的tar.gz就是可以分发的包

#4.python setup.py bdist_egg
#如果打包成egg，则执行这一步

5.cd dist
6.tar -zxvf adodbapi-2.0.6.7.tar.gz
7.cd adodbapi-2.0.6.7

8.python setup.py install
#安装

在python的路径下面已经有这个包了

```
root@ubuntu:/usr/local/lib/python2.7/dist-packages# ls
adodbapi  adodbapi-2.0.6.7.egg-info  foo-1.0.egg-info  foo.py  foo.pyc
root@ubuntu:/usr/local/lib/python2.7/dist-packages#
```

进入python IDE help一下会得到这个包的详细信息

```
Help on package adodbapi:

NAME
    adodbapi - adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO

FILE
    /usr/local/lib/python2.7/dist-packages/adodbapi/__init__.py

DESCRIPTION
    Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole
    * http://sourceforge.net/projects/adodbapi
```

如何编写setup.py

```
#!/usr/bin/env python
from distutils.core import setup

setup(
    name="adodbapi",
    version="2.0.6.7",
    description = "A pure Python package implementing \
                PEP 249 DB-API using Microsoft ADO",      # description
    author = "Vernon Cole",                               # pacakge author
    author_email = "vernondcole@gmail.com",               # author's email
    url = "http://sourceforge.net/projects/adodbapi",     # download url
    packages = ['adodbapi'],                              # search adodbapi/__init__.py, tell distutils
    license = 'LGPL',                                     # license
    platforms = 'linux; Windows',                         # platforms
    classifiers=[
        'Development Status :: 5 - Production/Stable',
        'Programming Language :: Python :: 2',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.4',
        'Programming Language :: Python :: 3.5',
        'Programming Language :: Python :: Implementation :: CPython',
        'Programming Language :: Python :: Implementation :: PyPy',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Topic :: Database',
    ],
)
```

1.setup.py的执行参数

1. python setup.py build #编译
2. python setup.py install #安装
3. python setup.py sdist #制作分发包，格式为zip/tar.gz
4. python setup.py bdist_egg #制作一个二进制分发包
5. python setup.py bdist_wininst #制作windows下的分发包，exe程序
6. python setup.py bdist_rpm #制作linux下的rpm包

2.setup函数参数

1. packages #告诉Distutils需要处理那些包（包含`__init__.py`的文件夹）
2. package_dir
   #告诉Distutils哪些目录下的文件被映射到哪个源码包，感觉好像是一个相对路径的定义。一个例子：`package_dir = {'': 'lib'}`，表示以lib为主目录。
3. ext_modules #是一个包含Extension实例的列表，Extension的定义也有一些参数。
4. ext_package #定义extension的相对路径
5. requires #定义依赖哪些模块
6. provides #定义可以为哪些模块提供依赖
7. scripts #指定python源码文件，可以从命令行执行。在安装时指定`--install-script`
8. package_data #通常包含与包实现相关的一些数据文件或类似于readme的文件。

参考：

1.http://aleeee.com/pure-python_distribution_by_package.html

2.http://lingxiankong.github.io/blog/2013/12/23/python-setup/

3.http://blog.csdn.net/zhaole524/article/details/51772586

ddagent构建之依赖包

所有的rpm包都可以到http://rpm.pbone.net/搜索下载，找准适用系统 (centos等等...) 以及python下安装后的版本号：

Content of RPM :
/usr/lib/python2.7/site-packages/javapackages
/usr/lib/python2.7/site-packages/javapackages-1.0.0-py2.7.egg-info
/usr/lib/python2.7/site-packages/javapackages-1.0.0-py2.7.egg-info/PKG-INFO
/usr/lib/python2.7/site-packages/javapackages-1.0.0-py2.7.egg-info/SOURCES.txt
/usr/lib/python2.7/site-packages/javapackages-1.0.0-py2.7.egg-info/dependency_links.txt
/usr/lib/python2.7/site-packages/javapackages-1.0.0-py2.7.egg-info/top_level.txt
/usr/lib/python2.7/site-packages/javapackages/__init__.py
/usr/lib/python2.7/site-packages/javapackages/__init__.pyc
/usr/lib/python2.7/site-packages/javapackages/__init__.pyo
/usr/lib/python2.7/site-packages/javapackages/artifact.py
/usr/lib/python2.7/site-packages/javapackages/artifact.pyc
/usr/lib/python2.7/site-packages/javapackages/artifact.pyo
/usr/lib/python2.7/site-packages/javapackages/depmap.py

Linux

├── adodbapi

│ └── 2.6.0.7

├── backports.ssl-match-hostname
│ └── 3.5.0.1
├── blivet 资源https://github.com/dwlehman/blivet/releases/tag/blivet-0.18.34 已获得。

│ └── 0.18.34
├── boto
│ └── 2.36.0
├── bson 和bson相关，这是pymongo包下的一个bson包，已知pymongo版本是3.20，已获得。
│ └── 3.2
├── cffi
│ └── 1.6.0
├── chardet
│ └── 2.0.1
├── ~~concurrent~~ futures 搜索发现，这个包真名是futures，https://pythonhosted.org/futures/，经过和pypi上该包历史版本比对，确定是agent使用的版本是2.20，已获得。
│ └── 2.20
├── configobj
│ └── 4.7.2
├── configshell-fb https://github.com/open-iscsi/configshell-fb/tree/v1.1.fb11，已获得
│ └── 1.1.11
├── ~~Crypto~~ pycrypto https://pypi.python.org/pypi/pycrypto，名字错了，是pycrypto，已获得
│ └── 2.61
├── cryptography
│ └── 1.3.1
├── cupshelpers centos环境下需要下载 http://rpm.pbone.net/index.php3/stat/4/idpl/23896896/dir/opensuse/com/python-cupshelpers-1.1.12-2.6.1.x86_64.rpm.html，debian也一样吗？ 待研究，已获得
│ └── 1.0
├── datadog
│ └── 0.10.0
├── ~~dateutil~~ python-dateutil 版本号是2.5.3，https://pypi.python.org/pypi/python-dateutil/2.5.3/，已获得
│ └── ~~0.0.9~~ 2.5.3
├── decorator
│ └── 4.0.9
├── di
│ └── 0.3
├── dnspython
│ └── 1.12.0
├── docker-py
│ └── 1.3.1
├── ecdsa
│ └── 0.13
├── enum34
│ └── 1.1.3
├── eventlet
│ └── 0.19.0

├── firstboot　centos环境下需要下载 http://rpm.pbone.net/index.php3/stat/4/idpl/29071190/dir/centos_7/com/firstboot-19.9-8.el7.x86_64.rpm.html，debian也一样吗？ 待研究，已获得
|　└── 19.5
├── freeipa　根据版本号和名字看起来应该是这个 https://github.com/freeipa/freeipa/tree/release-2-0-0，已获得
|　└── 2.0.0a0
├── fros　centos环境下需要下载 https://www.rpmfind.net/linux/RPM/centos/7.2.1511/x86_64/Packages/fros-1.0-2.el7.noarch.html，debian也一样吗？ 待研究，已获得
|　└── 1.0
├── futures
|　└── 2.2.0
├── gearman
|　└── 2.0.2
├── ~~gflags~~ python-gflags　全名是这个，https://pypi.python.org/pypi/python-gflags/3.0.4/，已获得
|　└── ~~0.0.9~~ 3.0.4
├── google-apputils
|　└── 0.4.2
├── gridfs　这是pymongo包下的一个gridfs包，已知pymongo版本是3.20，已获得
|　└── 0.0.9 3.20
├── httplib2
|　└── 0.9
├── idna
|　└── 2.1
├── iniparse
|　└── 0.4
├── initial-setup centos环境下需要下载 http://rpm.pbone.net/index.php3/stat/4/idpl/27383363/dir/scientific_linux_7/com/initial-setup-0.3.9.12-1.el7.x86_64.rpm.html，debian也一样吗？ 待研究，已获得
|　└── 0.3.9.12
├── ipaddress
|　└── 1.0.16
├── ipapython　　资源 http://rpm.pbone.net/index.php3?stat=3&search=ipa-python&srodzaj=3　已获得
|　└── 3.3.3
├── IPy
|　└── 0.75
├── javapackages centos环境下需要下 http://rpm.pbone.net/index.php3/stat/4/idpl/31983635/dir/centos_7/com/python-javapackages-3.4.1-11.el7.noarch.rpm.html，debian也一样吗？ 待研究，已获得
|　└── 1.0.0
├── kafka-python
|　└── 0.9.3
├── kazoo
|　└── 1.3.1
├── kitchen
|　└── 1.1.1
├── langtable　资源https://github.com/mike-fabian/langtable/releases/tag/0.0.13　已获得
|　└── 0.0.13
├── meld3
|　└── 1.0.2
├── netaddr　资源https://pypi.python.org/pypi/netaddr/0.7.15#downloads　netaddr0.7.15　（git上最老版本0.7.10）
　　https://github.com/drkjam/netaddr/releases/tag/rel-0.7.5　netaddr-rel 0.7.5（是哪一个？）待定
|　└── 0.7.5
├── ntplib
|　└── 0.3.3
├── paramiko
|　└── 1.15.2
├── pg8000
|　└── 1.10.1
├── pkg_resources 这是pymongo包下的一个bson包，已知pymongo版本是3.20，已获得
|　└── 0.0.9
├── ply
|　└── 3.8
├── protobuf
|　└── 2.6.1
├── psutil
|　└── 3.3.0
├── psycopg2

```
|   └── 2.6
├── pyasn1
|   └── 0.1.9
├── pycparser
|   └── 2.14
├── pycrypto
|   └── 2.6.1
├── pycurl
|   └── 7.19.5.1
├── pyinotify
|   └── 0.9.4
├── pykickstart   资源https://github.com/rhinstaller/pykickstart/releases/tag/r1.99.43.10-1   已获得
|   └── 1.99.43.10
├── pymongo
|   ├── 3.2
├── PyMySQL
|   └── 0.6.6
├── pyOpenSSL
|   └── 0.14
├── pyparsing
|   └── 1.5.6
├── Pyro4
|   └── 4.35
├── py-rrdtool   资源 http://www.pudn.com/downloads328/sourcecode/unix_linux/detail1443142.html 已获得
|   └── 0.2.1
├── pysmi
|   └── 0.0.7
├── pysnmp
|   └── 4.3.2
├── pysnmp-mibs
|   └── 0.1.4
├── python-augeas
|   └── 0.4.1
├── python-dateutil
|   └── 2.5.3
├── python-gflags
|   └── 3.0.4
├── python-meh   资源https://github.com/rhinstaller/python-meh/releases/tag/r0.25-1  已获得
|   └── 0.25
├── python-memcached
|   └── 1.53
├── pytz
|   └── 2016.3
├── pyudev
|   └── 0.15
├── pyVim
|   └── 0.0.9
├── pyvmomi
|   └── 6.0.0
├── PyYAML
|   └── 3.11
├── redis
|   └── 2.10.3
├── requests
|   └── 2.6.0
├── rtslib-fb   资源https://github.com/open-iscsi/rtslib-fb/releases/tag/v2.1.fb46  已获得
|   └── 2.1.46
├── seobject   没找到任何资源       未获得
|   └── 0.1
├── serpent
|   └── 1.12
├── setuptools
|   └── 20.9.0
├── simplejson
|   └── 3.6.5
```

├── six
│   └── 1.10.0
├── slip　　资源https://git.fedorahosted.org/cgit/python-slip.git/　已获得
│   └── 0.4.0
├── slip.dbus　是slip下面的方法　　已获得
│   └── 0.4.0
├── snakebite
│   └── 1.3.11
├── SSSDConfig　资源http://rpm.pbone.net/index.php3/stat/3/limit/5/srodzaj/1/dl/40/search/python-sssdconfig/field[]/1/field[]/2
│   └── 1.11.2　　已获得
├── supervisor
│   └── 3.1.3
├── targetcli-fb　　资源在https://github.com/open-iscsi/targetcli-fb　已获得
│   └── =2.1.fb34
├── tornado
│   └── 3.2.2
├── uptime
│   └── 3.0.1
├── urlgrabber　下载资源在http://www.rpmfind.net/linux/rpm2html/search.php?query=python-urlgrabber　已获得
│   └── 3.10
├── uuid
│   └── 1.30
├── websocket-client
│   └── 0.37.0
├── yum-langpacks　资源 https://git.fedorahosted.org/git/yum-langpacks.git　已获得
│   └── 0.4.2
└── zope.interface
    └── 4.1

windows (to be continue)

├── adodbapi
│   └── 2.6.0.7
├── backports.ssl-match-hostname
│   └── 3.5.0.1
├── boto
│   └── 2.36.0
├── bson　　只找到 https://github.com/dwlehman/blivet 需要下载自行setup
│   └── 0.0.9
├── certifi　　没找到ddagent对应的版本
│   └── 0.0.9
├── checks　没找到ddagent对应的版本
│   └── 0.0.9
├── Crypto　没找到ddagent对应的版本
│   └── 0.0.9
├── cx_Oracle　　没找到ddagent对应的版本
│   └── 0.0.9
├── docker　　没找到ddagent对应的版本
│   └── 0.0.9
├── ecdsa
│   └── 0.13
├── guidata　没找到ddagent对应的版本
│   └── 0.0.9
├── httplib2
│   └── 0.9
├── _markerlib　　没找到ddagent对应的版本
│   └── 0.0.9
├── pkg_resources　没找到ddagent对应的版本
│   └── 0.0.9
├── psutil
├── pyasn1
├── pycparser
│   └── 2.14
├── pycrypto
│   └── 2.6.1

```
├── pycurl
│   └── 7.19.5.1
├── pydoc_data    没找到ddagent对应的版本
│   └── 0.0.9
├── pyinotify
│   └── 0.9.4
├── pykickstart
│   └── 1.99.43.10
├── pymongo
│   └── 3.2
├── PyMySQL
│   └── 0.6.6
├── pyparsing
│   └── 1.5.6
├── PyQt4    没找到ddagent对应的版本
│   └── 0.0.9
├── pysnmp
│   └── 4.3.2
├── pysnmp-mibs
│   └── 0.1.4
├── pyVim    没找到ddagent对应的版本
│   └── 0.0.9
├── pyvmomi
│   └── 6.0.0
├── pywin    没找到ddagent对应的版本
│   └── 0.0.9
├── PyYAML
│   └── 3.11
├── requests
│   └── 2.6.0
├── resources    没找到ddagent对应的版本
│   └── 0.0.9
├── setuptools    没找到ddagent对应的版本
│   └── 0.0.9
├── simplejson
│   └── 3.6.5
├── six
│   └── 1.10.0
├── slip
│   └── 0.4.0
├── snakebite
│   └── 1.3.11
├── spyderlib
│   └── 0.0.9
├── spyderplugins
│   └── 0.0.9
├── tornado
│   └── 3.2.2
├── unittest
│   └── 0.0.9
├── uptime
│   └── 3.0.1
├── utils    没找到ddagent对应的版本
│   └── 0.0.9
├── uuid
│   └── 1.30
├── uyun    没找到ddagent对应的版本
│   └── 0.0.9
├── websocket-client
│   └── 0.37.0
├── win32    没找到ddagent对应的版本
│   └── 0.0.9
├── win32com    没找到ddagent对应的版本
│   └── 0.0.9
├── winrandom    没找到ddagent对应的版本
```

│ └── 0.0.9
├── yum-langpacks
│ └── 0.4.2
└── zope.interface
└── 4.1

Linux

├── adodbapi

│ └── 2.6.0.7

├── backports.ssl-match-hostname
│ └── 3.5.0.1
├── blivet 只找到 https://github.com/dwlehman/blivet 需要下载自行setup github上保留的最老的版本也是0.61，未获得。

│ └── 0.18.34
├── boto
│ └── 2.36.0
├── bson 和bson相关，这是pymongo包下的一个bson包，已知pymongo版本是2.20，已获得。
│ └── 3.2
├── cffi
│ └── 1.6.0
├── chardet
│ └── 2.0.1
├── ~~concurrent~~ futures 搜索发现，这个包真名是futures，https://pythonhosted.org/futures/，经过和pypi上该包历史版本比对，确定是agent使用的版本是2.20，已获得。
│ └── 2.20
├── configobj
│ └── 4.7.2
├── configshell-fb https://github.com/open-iscsi/configshell-fb/tree/v1.1.fb11，已获得
│ └── 1.1.11
├── ~~Crypto~~

pycrypto

- 没找到ddagent对应的版本
│ └── 0.0.9
├── cryptography
│ └── 1.3.1
├── cupshelpers 没找到
│ └── 1.0
├── datadog
│ └── 0.10.0
├── dateutil 没找到ddagent对应的版本
│ └── 0.0.9
├── decorator
│ └── 4.0.9
├── di
│ └── 0.3
├── dnspython
│ └── 1.12.0
├── docker-py
│ └── 1.3.1
├── ecdsa
│ └── 0.13
├── enum34
│ └── 1.1.3
├── eventlet
│ └── 0.19.0
├── firstboot 没找到
│ └── 19.5
├── freeipa 没找到
│ └── 2.0.0a0
├── fros 没找到
│ └── 1.0

├── futures
│   └── 2.2.0
├── gearman
│   └── 2.0.2
├── gflags 没找到ddagent对应的版本
│   └── 0.0.9
├── google-apputils
│   └── 0.4.2
├── gridfs 没找到ddagent对应的版本
│   └── 0.0.9
├── httplib2
│   └── 0.9
├── idna
│   └── 2.1
├── iniparse
│   └── 0.4
├── initial-setup 没找到
│   └── 0.3.9.12
├── ipaddress
│   └── 1.0.16
├── ipapython
│   └── 3.3.3
├── IPy
│   └── 0.75
├── javapackages 没找到
│   └── 1.0.0
├── kafka-python
│   └── 0.9.3
├── kazoo
│   └── 1.3.1
├── kitchen
│   └── 1.1.1
├── langtable https://pkgs.org/download/langtable-python 版本最旧是0.0.27 未获得
│   └── 0.0.13
├── meld3
│   └── 1.0.2
├── netaddr 资源https://pypi.python.org/pypi/netaddr/0.7.15#downloads 已获得
│   └── 0.7.5
├── ntplib
│   └── 0.3.3
├── paramiko
│   └── 1.15.2
├── pg8000
│   └── 1.10.1
├── pkg_resources ddagent对应的版本未知 https://packages.debian.org/zh-cn/sid/python-pkg-resources
│   └── 0.0.9 这里的版本是20.10.1-1.1 待定
├── ply
│   └── 3.8
├── protobuf
│   └── 2.6.1
├── psutil
│   └── 3.3.0
├── psycopg2
│   └── 2.6
├── pyasn1
│   └── 0.1.9
├── pycparser
│   └── 2.14
├── pycrypto
│   └── 2.6.1
├── pycurl
│   └── 7.19.5.1
├── pyinotify
│   └── 0.9.4
├── pykickstart 资源https://github.com/rhinstaller/pykickstart 版本3.3 未获得

```
|   └── 1.99.43.10
├── pymongo
|   ├── 3.2
├── PyMySQL
|   └── 0.6.6
├── pyOpenSSL
|   └── 0.14
├── pyparsing
|   └── 1.5.6
├── Pyro4
|   └── 4.35
├── py-rrdtool   资源 http://www.pudn.com/downloads328/sourcecode/unix_linux/detail1443142.html 已获得
|   └── 0.2.1
├── pysmi
|   └── 0.0.7
├── pysnmp
|   └── 4.3.2
├── pysnmp-mibs
|   └── 0.1.4
├── python-augeas
|   └── 0.4.1
├── python-dateutil
|   └── 2.5.3
├── python-gflags
|   └── 3.0.4
├── python-meh   资源https://github.com/rhinstaller/python-meh/releases/tag/r0.25-1  已获得
|   └── 0.25
├── python-memcached
|   └── 1.53
├── pytz
|   └── 2016.3
├── pyudev
|   └── 0.15
├── pyVim
|   └── 0.0.9
├── pyvmomi
|   └── 6.0.0
├── PyYAML
|   └── 3.11
├── redis
|   └── 2.10.3
├── requests
|   └── 2.6.0
├── rtslib-fb    资源https://github.com/open-iscsi/rtslib-fb/releases/tag/v2.1.fb46  已获得
|   └── 2.1.46
├── seobject   没找到任何资源       未获得
|   └── 0.1
├── serpent
|   └── 1.12
├── setuptools
|   └── 20.9.0
├── simplejson
|   └── 3.6.5
├── six
|   └── 1.10.0
├── slip       资源https://git.fedorahosted.org/cgit/python-slip.git/   已获得
|   └── 0.4.0
├── slip.dbus    没找到任何资源       未获得
|   └── 0.4.0
├── snakebite
|   └── 1.3.11
├── SSSDConfig   资源http://rpm.pbone.net/index.php3/stat/3/limit/5/srodzaj/1/dl/40/search/python-sssdconfig/field[]/1/field[]/2
|   └── 1.11.2                                                                    已获得
├── supervisor
```

```
|   └── 3.1.3
├── targetcli-fb     资源在https://github.com/open-iscsi/targetcli-fb  已获得
|   └── =2.1.fb34
├── tornado
|   └── 3.2.2
├── uptime
|   └── 3.0.1
├── urlgrabber  下载资源在http://www.rpmfind.net/linux/rpm2html/search.php?query=python-urlgrabber   已获得
|   └── 3.10
├── uuid
|   └── 1.30
├── websocket-client
|   └── 0.37.0
├── yum-langpacks   资源 https://git.fedorahosted.org/git/yum-langpacks.git  已获得
|   └── 0.4.2
└── zope.interface
    └── 4.1
```

windows (to be continue)

```
├── adodbapi
|   └── 2.6.0.7
├── backports.ssl-match-hostname
|   └── 3.5.0.1
├── boto
|   └── 2.36.0
├── bson     只找到 https://github.com/dwlehman/blivet 需要下载自行setup
|   └── 0.0.9
├── certifi    没找到ddagent对应的版本
|   └── 0.0.9
├── checks  没找到ddagent对应的版本
|   └── 0.0.9
├── Crypto  没找到ddagent对应的版本
|   └── 0.0.9
├── cx_Oracle   没找到ddagent对应的版本
|   └── 0.0.9
├── docker    没找到ddagent对应的版本
|   └── 0.0.9
├── ecdsa
|   └── 0.13
├── guidata   没找到ddagent对应的版本
|   └── 0.0.9
├── httplib2
|   └── 0.9
├── _markerlib   没找到ddagent对应的版本
|   └── 0.0.9
├── pkg_resources  没找到ddagent对应的版本
|   └── 0.0.9
├── psutil
├── pyasn1
├── pycparser
|   └── 2.14
├── pycrypto
|   └── 2.6.1
├── pycurl
|   └── 7.19.5.1
├── pydoc_data    没找到ddagent对应的版本
|   └── 0.0.9
├── pyinotify
|   └── 0.9.4
├── pykickstart
|   └── 1.99.43.10
├── pymongo
|   └── 3.2
├── PyMySQL
```

```
|   └── 0.6.6
├── pyparsing
|   └── 1.5.6
├── PyQt4   没找到ddagent对应的版本
|   └── 0.0.9
├── pysnmp
|   └── 4.3.2
├── pysnmp-mibs
|   └── 0.1.4
├── pyVim   没找到ddagent对应的版本
|   └── 0.0.9
├── pyvmomi
|   └── 6.0.0
├── pywin   没找到ddagent对应的版本
|   └── 0.0.9
├── PyYAML
|   └── 3.11
├── requests
|   └── 2.6.0
├── resources   没找到ddagent对应的版本
|   └── 0.0.9
├── setuptools   没找到ddagent对应的版本
|   └── 0.0.9
├── simplejson
|   └── 3.6.5
├── six
|   └── 1.10.0
├── slip
|   └── 0.4.0
├── snakebite
|   └── 1.3.11
├── spyderlib
|   └── 0.0.9
├── spyderplugins
|   └── 0.0.9
├── tornado
|   └── 3.2.2
├── unittest
|   └── 0.0.9
├── uptime
|   └── 3.0.1
├── utils   没找到ddagent对应的版本
|   └── 0.0.9
├── uuid
|   └── 1.30
├── uyun   没找到ddagent对应的版本
|   └── 0.0.9
├── websocket-client
|   └── 0.37.0
├── win32   没找到ddagent对应的版本
|   └── 0.0.9
├── win32com   没找到ddagent对应的版本
|   └── 0.0.9
├── winrandom   没找到ddagent对应的版本
|   └── 0.0.9
├── yum-langpacks
|   └── 0.4.2
└── zope.interface
    └── 4.1
```

Linux

```
├── adodbapi
|   └── 2.6.0.7
```

├── backports.ssl-match-hostname
│   └── 3.5.0.1
├── blivet   只找到 https://github.com/dwlehman/blivet 需要下载自行setup

github上保留的最老的版本也是0.61

│   └── 0.18.34
├── boto
│   └── 2.36.0
├── bson   没找到ddagent对应的版本
│   └── 0.0.9
├── cffi
│   └── 1.6.0
├── chardet
│   └── 2.0.1
├── concurrent   没找到ddagent对应的版本
│   └── 0.0.9
├── configobj
│   └── 4.7.2
├── configshell-fb
│   └── 1.1.11
├── Crypto   没找到ddagent对应的版本
│   └── 0.0.9
├── cryptography
│   └── 1.3.1
├── cupshelpers 没找到
│   └── 1.0
├── datadog
│   └── 0.10.0
├── dateutil   没找到ddagent对应的版本
│   └── 0.0.9
├── decorator
│   └── 4.0.9
├── di
│   └── 0.3
├── dnspython
│   └── 1.12.0
├── docker-py
│   └── 1.3.1
├── ecdsa
│   └── 0.13
├── enum34
│   └── 1.1.3
├── eventlet
│   └── 0.19.0
├── firstboot   没找到
│   └── 19.5
├── freeipa   没找到
│   └── 2.0.0a0
├── fros   没找到
│   └── 1.0
├── futures
│   └── 2.2.0
├── gearman
│   └── 2.0.2
├── gflags   没找到ddagent对应的版本
│   └── 0.0.9
├── google-apputils
│   └── 0.4.2
├── gridfs 没找到ddagent对应的版本
│   └── 0.0.9
├── httplib2
│   └── 0.9
├── idna
│   └── 2.1

├── iniparse
│   └── 0.4
├── initial-setup 没找到
│   └── 0.3.9.12
├── ipaddress
│   └── 1.0.16
├── ipapython
│   └── 3.3.3
├── IPy
│   └── 0.75
├── javapackages 没找到
│   └── 1.0.0
├── kafka-python
│   └── 0.9.3
├── kazoo
│   └── 1.3.1
├── kitchen
│   └── 1.1.1
├── langtable 没找到
│   └── 0.0.13
├── meld3
│   └── 1.0.2
├── netaddr 只找到0.7.18
│   └── 0.7.5
├── ntplib
│   └── 0.3.3
├── paramiko
│   └── 1.15.2
├── pg8000
│   └── 1.10.1
├── pkg_resources 没找到ddagent对应的版本
│   └── 0.0.9
├── ply
│   └── 3.8
├── protobuf
│   └── 2.6.1
├── psutil
│   └── 3.3.0
├── psycopg2
│   └── 2.6
├── pyasn1
│   └── 0.1.9
├── pycparser
│   └── 2.14
├── pycrypto
│   └── 2.6.1
├── pycurl
│   └── 7.19.5.1
├── pyinotify
│   └── 0.9.4
├── pykickstart 没找到
│   └── 1.99.43.10
├── pymongo
│   ├── 3.2
├── PyMySQL
│   └── 0.6.6
├── pyOpenSSL
│   └── 0.14
├── pyparsing
│   └── 1.5.6
├── Pyro4
│   └── 4.35
├── py-rrdtool 只找到1.0b1
│   └── 0.2.1
├── pysmi

```
|   └── 0.0.7
├── pysnmp
|   └── 4.3.2
├── pysnmp-mibs
|   └── 0.1.4
├── python-augeas
|   └── 0.4.1
├── python-dateutil
|   └── 2.5.3
├── python-gflags
|   └── 3.0.4
├── python-meh    没找到
|   └── 0.25
├── python-memcached
|   └── 1.53
├── pytz
|   └── 2016.3
├── pyudev
|   └── 0.15
├── pyVim
|   └── 0.0.9
├── pyvmomi
|   └── 6.0.0
├── PyYAML
|   └── 3.11
├── redis
|   └── 2.10.3
├── requests
|   └── 2.6.0
├── rtslib-fb    只找到2.1.47
|   └── 2.1.46
├── seobject    没找到
|   └── 0.1
├── serpent
|   └── 1.12
├── setuptools
|   └── 20.9.0
├── simplejson
|   └── 3.6.5
├── six
|   └── 1.10.0
├── slip      只找到0.2
|   └── 0.4.0
├── slip.dbus    没找到
|   └── 0.4.0
├── snakebite
|   └── 1.3.11
├── SSSDConfig    没找到
|   └── 1.11.2
├── supervisor
|   └── 3.1.3
├── targetcli-fb      没找到
|   └── =2.1.fb34
├── tornado
|   └── 3.2.2
├── uptime
|   └── 3.0.1
├── urlgrabber    只找到3.9.1，官方最新版只到3.9.1
|   └── 3.10
├── uuid
|   └── 1.30
├── websocket-client
|   └── 0.37.0
├── yum-langpacks    没找到
|   └── 0.4.2
```

└── zope.interface
    └── 4.1

windows (to be continue)

├── adodbapi
│   └── 2.6.0.7
├── backports.ssl-match-hostname
│   └── 3.5.0.1
├── boto
│   └── 2.36.0
├── bson 只找到 https://github.com/dwlehman/blivet 需要下载自行setup
│   └── 0.0.9
├── certifi 没找到ddagent对应的版本
│   └── 0.0.9
├── checks 没找到ddagent对应的版本
│   └── 0.0.9
├── Crypto 没找到ddagent对应的版本
│   └── 0.0.9
├── cx_Oracle 没找到ddagent对应的版本
│   └── 0.0.9
├── docker 没找到ddagent对应的版本
│   └── 0.0.9
├── ecdsa
│   └── 0.13
├── guidata 没找到ddagent对应的版本
│   └── 0.0.9
├── httplib2
│   └── 0.9
├── _markerlib 没找到ddagent对应的版本
│   └── 0.0.9
├── pkg_resources 没找到ddagent对应的版本
│   └── 0.0.9
├── psutil
├── pyasn1
├── pycparser
│   └── 2.14
├── pycrypto
│   └── 2.6.1
├── pycurl
│   └── 7.19.5.1
├── pydoc_data 没找到ddagent对应的版本
│   └── 0.0.9
├── pyinotify
│   └── 0.9.4
├── pykickstart
│   └── 1.99.43.10
├── pymongo
│   └── 3.2
├── PyMySQL
│   └── 0.6.6
├── pyparsing
│   └── 1.5.6
├── PyQt4 没找到ddagent对应的版本
│   └── 0.0.9
├── pysnmp
│   └── 4.3.2
├── pysnmp-mibs
│   └── 0.1.4
├── pyVim 没找到ddagent对应的版本
│   └── 0.0.9
├── pyvmomi
│   └── 6.0.0
├── pywin 没找到ddagent对应的版本
│   └── 0.0.9

```
├── PyYAML
│   └── 3.11
├── requests
│   └── 2.6.0
├── resources  没找到ddagent对应的版本
│   └── 0.0.9
├── setuptools  没找到ddagent对应的版本
│   └── 0.0.9
├── simplejson
│   └── 3.6.5
├── six
│   └── 1.10.0
├── slip
│   └── 0.4.0
├── snakebite
│   └── 1.3.11
├── spyderlib
│   └── 0.0.9
├── spyderplugins
│   └── 0.0.9
├── tornado
│   └── 3.2.2
├── unittest
│   └── 0.0.9
├── uptime
│   └── 3.0.1
├── utils   没找到ddagent对应的版本
│   └── 0.0.9
├── uuid
│   └── 1.30
├── uyun   没找到ddagent对应的版本
│   └── 0.0.9
├── websocket-client
│   └── 0.37.0
├── win32   没找到ddagent对应的版本
│   └── 0.0.9
├── win32com   没找到ddagent对应的版本
│   └── 0.0.9
├── winrandom   没找到ddagent对应的版本
│   └── 0.0.9
├── yum-langpacks
│   └── 0.4.2
└── zope.interface
    └── 4.1
```

dd-agent监控MySQL指标详解

- 采集方式
    - code：
    - pymysql vs MySQLdb
- 指标
    - STATUS_VARS
    - VARIABLES_VARS
    - INNODB_VARS
    - BINLOG_VARS
    - OPTIONAL_STATUS_VARS
    - OPTIONAL_STATUS_VARS_5_6_6
    - OPTIONAL_INNODB_VARS
    - GALERA_VARS
    - PERFORMANCE_VARS
    - SCHEMA_VARS
    - REPLICA_VARS
    - SYNTHETIC_VARS
- 各项指标的采集方式
    - self._collect_metadata(db, host)
    - self._collect_metrics(host, db, tags, options, queries)
    - self._collect_system_metrics(host, db, tags)

采集方式

code：

linux为例 /opt/datadog-agent/agent/checks.d/mysql.py

pymysql vs MySQLdb

pymysql和MySQLdb都是python操作mysql数据库的模块，但是前者支持python 3.x，后者不支持，因此ddagent这边采用的是pymysql模块。

指标

STATUS_VARS

在方法 _collect_metrics中执行sql "SHOW STATUS;"可以得到的；

```
mysql> SHOW STATUS;
+-----------------------------------------+-------------+
| Variable_name                           | Value       |
+-----------------------------------------+-------------+
| Aborted_clients                         | 0           |
| Aborted_connects                        | 1           |
| Binlog_cache_disk_use                   | 0           |
| Binlog_cache_use                        | 0           |
| Binlog_stmt_cache_disk_use              | 0           |
| Binlog_stmt_cache_use                   | 0           |
| Bytes_received                          | 1159        |
| Bytes_sent                              | 1185        |
| Com_admin_commands                      | 0           |
| Com_assign_to_keycache                  | 0           |
| Com_alter_db                            | 0           |
| Com_alter_db_upgrade                    | 0           |
| Com_alter_event                         | 0           |
| Com_alter_function                      | 0           |
| Com_alter_procedure                     | 0           |
| Com_alter_server                        | 0           |
| Com_alter_table                         | 0           |
| Com_alter_tablespace                    | 0           |
| Com_analyze                             | 0           |
| Com_begin                               | 0           |
| Com_binlog                              | 0           |
| Com_call_procedure                      | 0           |
| Com_change_db                           | 0           |
| Com_change_master                       | 0           |
| Com_check                               | 0           |
| Com_checksum                            | 0           |
| Com_commit                              | 0           |
| Com_create_db                           | 0           |
| Com_create_event                        | 0           |
| Com_create_function                     | 0           |
| Com_create_index                        | 0           |
| Com_create_procedure                    | 0           |
| Com_create_server                       | 0           |
| Com_create_table                        | 0           |
| Com_create_trigger                      | 0           |
| Com_create_udf                          | 0           |
```

```
STATUS_VARS = {
    # Command Metrics
    'Slow_queries': ('mysql.performance.slow_queries', RATE),    #慢查询
    'Questions': ('mysql.performance.questions', RATE),
    'Queries': ('mysql.performance.queries', RATE),
    'Com_select': ('mysql.performance.com_select', RATE),
    'Com_insert': ('mysql.performance.com_insert', RATE),
    'Com_update': ('mysql.performance.com_update', RATE),
    'Com_delete': ('mysql.performance.com_delete', RATE),
    'Com_replace': ('mysql.performance.com_replace', RATE),
    'Com_load': ('mysql.performance.com_load', RATE),
    'Com_insert_select': ('mysql.performance.com_insert_select', RATE),
    'Com_update_multi': ('mysql.performance.com_update_multi', RATE),
    'Com_delete_multi': ('mysql.performance.com_delete_multi', RATE),
    'Com_replace_select': ('mysql.performance.com_replace_select', RATE),
    # Connection Metrics
    'Connections': ('mysql.net.connections', RATE),
    'Max_used_connections': ('mysql.net.max_connections', GAUGE),
    'Aborted_clients': ('mysql.net.aborted_clients', RATE),
    'Aborted_connects': ('mysql.net.aborted_connects', RATE),
    # Table Cache Metrics
    'Open_files': ('mysql.performance.open_files', GAUGE),
    'Open_tables': ('mysql.performance.open_tables', GAUGE),
    # Network Metrics
    'Bytes_sent': ('mysql.performance.bytes_sent', RATE),
    'Bytes_received': ('mysql.performance.bytes_received', RATE),
    # Query Cache Metrics
    'Qcache_hits': ('mysql.performance.qcache_hits', RATE),
    'Qcache_inserts': ('mysql.performance.qcache_inserts', RATE),
    'Qcache_lowmem_prunes': ('mysql.performance.qcache_lowmem_prunes', RATE),
    # Table Lock Metrics
    'Table_locks_waited': ('mysql.performance.table_locks_waited', GAUGE),
    'Table_locks_waited_rate': ('mysql.performance.table_locks_waited.rate', RATE),
    # Temporary Table Metrics
    'Created_tmp_tables': ('mysql.performance.created_tmp_tables', RATE),
    'Created_tmp_disk_tables': ('mysql.performance.created_tmp_disk_tables', RATE),
    'Created_tmp_files': ('mysql.performance.created_tmp_files', RATE),
    # Thread Metrics
    'Threads_connected': ('mysql.performance.threads_connected', GAUGE),
    'Threads_running': ('mysql.performance.threads_running', GAUGE),
    # MyISAM Metrics
    'Key_buffer_bytes_unflushed': ('mysql.myisam.key_buffer_bytes_unflushed', GAUGE),
    'Key_buffer_bytes_used': ('mysql.myisam.key_buffer_bytes_used', GAUGE),
    'Key_read_requests': ('mysql.myisam.key_read_requests', RATE),
    'Key_reads': ('mysql.myisam.key_reads', RATE),
    'Key_write_requests': ('mysql.myisam.key_write_requests', RATE),
    'Key_writes': ('mysql.myisam.key_writes', RATE),
}
```

VARIABLES_VARS

在方法 _collect_scalar中，通过执行sql "SHOW GLOBAL VARIABLES"，解析执行结果得到。

```
VARIABLES_VARS = {
    'Key_buffer_size': ('mysql.myisam.key_buffer_size', GAUGE),
    'Key_cache_utilization': ('mysql.performance.key_cache_utilization', GAUGE),
    'max_connections': ('mysql.net.max_connections_available', GAUGE),
    'query_cache_size': ('mysql.performance.qcache_size', GAUGE),
    'table_open_cache': ('mysql.performance.table_open_cache', GAUGE),
    'thread_cache_size': ('mysql.performance.thread_cache_size', GAUGE)
}
```

INNODB_VARS

InnoDB相关的指标，在方法_get_stats_from_innodb_status中，执行sql "SHOW /*!50000 ENGINE*/ INNODB STATUS"：

解析sql执行的结果，得到关于InnoDB相关指标。

例：

```
mysql> SHOW /*!50000 ENGINE*/ INNODB STATUS \G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
161114 13:30:49 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 58 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 1 1_second, 1 sleeps, 0 10_second, 1 background, 1 flush
srv_master_thread log flush and writes: 1
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 2, signal count 2
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 2, rounds 60, OS waits 2
RW-excl spins 0, rounds 0, OS waits 0
Spin rounds per wait: 0.00 mutex, 30.00 RW-shared, 0.00 RW-excl
------------
TRANSACTIONS
------------
Trx id counter 700
Purge done for trx's n:o < 530 undo n:o < 0
History list length 16
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started
MySQL thread id 38, OS thread handle 0x7ffaa2d84700, query id 117 localhost root
SHOW /*!50000 ENGINE*/ INNODB STATUS
--------
FILE I/O
--------
```

```
---
LOG
---
Log sequence number 2123696
Log flushed up to   2123696
Last checkpoint at  2123696
0 pending log writes, 0 pending chkp writes
8 log i/o's done, 0.00 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 137363456; in additional pool allocated 0
Dictionary memory allocated 37767
Buffer pool size   8192
Free buffers       8020
Database pages     172
Old database pages 0
Modified db pages  0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 172, created 0, written 0
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 172, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 1271, id 140714035697408, state: waiting for server activity
Number of rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

```
INNODB_VARS = {
    # InnoDB metrics
    'Innodb_data_reads': ('mysql.innodb.data_reads', RATE),
    'Innodb_data_writes': ('mysql.innodb.data_writes', RATE),
    'Innodb_os_log_fsyncs': ('mysql.innodb.os_log_fsyncs', RATE),
    'Innodb_mutex_spin_waits': ('mysql.innodb.mutex_spin_waits', RATE),
    'Innodb_mutex_spin_rounds': ('mysql.innodb.mutex_spin_rounds', RATE),
    'Innodb_mutex_os_waits': ('mysql.innodb.mutex_os_waits', RATE),
    'Innodb_row_lock_waits': ('mysql.innodb.row_lock_waits', RATE),
    'Innodb_row_lock_time': ('mysql.innodb.row_lock_time', RATE),
    'Innodb_row_lock_current_waits': ('mysql.innodb.row_lock_current_waits', GAUGE),
    'Innodb_current_row_locks': ('mysql.innodb.current_row_locks', GAUGE),
    'Innodb_buffer_pool_bytes_dirty': ('mysql.innodb.buffer_pool_dirty', GAUGE),
    'Innodb_buffer_pool_bytes_free': ('mysql.innodb.buffer_pool_free', GAUGE),
    'Innodb_buffer_pool_bytes_used': ('mysql.innodb.buffer_pool_used', GAUGE),
    'Innodb_buffer_pool_bytes_total': ('mysql.innodb.buffer_pool_total', GAUGE),
    'Innodb_buffer_pool_read_requests': ('mysql.innodb.buffer_pool_read_requests', RATE),
    'Innodb_buffer_pool_reads': ('mysql.innodb.buffer_pool_reads', RATE),
    'Innodb_buffer_pool_pages_utilization': ('mysql.innodb.buffer_pool_utilization', GAUGE),
}
```

BINLOG_VARS

方法 _get_binary_log_stats

根据执行sql "SHOW BINARY LOGS;"得到

```
BINLOG_VARS = {
    'Binlog_space_usage_bytes': ('mysql.binlog.disk_use', GAUGE),
}
```

OPTIONAL_STATUS_VARS

这一大堆都可以在方法 _collect_metrics中执行指令 "SHOW STATUS;"可以得到的

```
OPTIONAL_STATUS_VARS = {
    'Binlog_cache_disk_use': ('mysql.binlog.cache_disk_use', GAUGE),
    'Binlog_cache_use': ('mysql.binlog.cache_use', GAUGE),
    'Handler_commit': ('mysql.performance.handler_commit', RATE),
    'Handler_delete': ('mysql.performance.handler_delete', RATE),
    'Handler_prepare': ('mysql.performance.handler_prepare', RATE),
    'Handler_read_first': ('mysql.performance.handler_read_first', RATE),
    'Handler_read_key': ('mysql.performance.handler_read_key', RATE),
    'Handler_read_next': ('mysql.performance.handler_read_next', RATE),
    'Handler_read_prev': ('mysql.performance.handler_read_prev', RATE),
    'Handler_read_rnd': ('mysql.performance.handler_read_rnd', RATE),
    'Handler_read_rnd_next': ('mysql.performance.handler_read_rnd_next', RATE),
    'Handler_rollback': ('mysql.performance.handler_rollback', RATE),
    'Handler_update': ('mysql.performance.handler_update', RATE),
    'Handler_write': ('mysql.performance.handler_write', RATE),
    'Opened_tables': ('mysql.performance.opened_tables', RATE),
    'Qcache_total_blocks': ('mysql.performance.qcache_total_blocks', GAUGE),
    'Qcache_free_blocks': ('mysql.performance.qcache_free_blocks', GAUGE),
    'Qcache_free_memory': ('mysql.performance.qcache_free_memory', GAUGE),
    'Qcache_not_cached': ('mysql.performance.qcache_not_cached', RATE),
    'Qcache_queries_in_cache': ('mysql.performance.qcache_queries_in_cache', GAUGE),
    'Select_full_join': ('mysql.performance.select_full_join', RATE),
    'Select_full_range_join': ('mysql.performance.select_full_range_join', RATE),
    'Select_range': ('mysql.performance.select_range', RATE),
    'Select_range_check': ('mysql.performance.select_range_check', RATE),
    'Select_scan': ('mysql.performance.select_scan', RATE),
    'Sort_merge_passes': ('mysql.performance.sort_merge_passes', RATE),
    'Sort_range': ('mysql.performance.sort_range', RATE),
    'Sort_rows': ('mysql.performance.sort_rows', RATE),
    'Sort_scan': ('mysql.performance.sort_scan', RATE),
    'Table_locks_immediate': ('mysql.performance.table_locks_immediate', GAUGE),
    'Table_locks_immediate_rate': ('mysql.performance.table_locks_immediate.rate', RATE),
    'Threads_cached': ('mysql.performance.threads_cached', GAUGE),
    'Threads_created': ('mysql.performance.threads_created', MONOTONIC)
}
```

OPTIONAL_STATUS_VARS_5_6_6

mysql 5.6.6版本的两个指标，mysql版本根据 version判断。

这两个指标是Mysql5.6.6新增的指标，亦可以通过执行sql "SHOW STATUS;"得到。

```
OPTIONAL_STATUS_VARS_5_6_6 = {
    'Table_open_cache_hits': ('mysql.performance.table_cache_hits', RATE),
    'Table_open_cache_misses': ('mysql.performance.table_cache_misses', RATE),
}
```

OPTIONAL_INNODB_VARS

供选择的InnoDB监测指标，当参数extra_innodb_metrics为True的时候才会监测。

同样是在方法_get_stats_from_innodb_status中，执行sql "SHOW /*!50000 ENGINE*/ INNODB STATUS"，解析结果可以得到。

```
OPTIONAL_INNODB_VARS = {
    'Innodb_active_transactions': ('mysql.innodb.active_transactions', GAUGE),
    'Innodb_buffer_pool_bytes_data': ('mysql.innodb.buffer_pool_data', GAUGE),
    'Innodb_buffer_pool_pages_data': ('mysql.innodb.buffer_pool_pages_data', GAUGE),
    'Innodb_buffer_pool_pages_dirty': ('mysql.innodb.buffer_pool_pages_dirty', GAUGE),
    'Innodb_buffer_pool_pages_flushed': ('mysql.innodb.buffer_pool_pages_flushed', RATE),
    'Innodb_buffer_pool_pages_free': ('mysql.innodb.buffer_pool_pages_free', GAUGE),
    'Innodb_buffer_pool_pages_total': ('mysql.innodb.buffer_pool_pages_total', GAUGE),
    'Innodb_buffer_pool_read_ahead': ('mysql.innodb.buffer_pool_read_ahead', RATE),
    'Innodb_buffer_pool_read_ahead_evicted': ('mysql.innodb.buffer_pool_read_ahead_evicted', RATE),
    'Innodb_buffer_pool_read_ahead_rnd': ('mysql.innodb.buffer_pool_read_ahead_rnd', GAUGE),
    'Innodb_buffer_pool_wait_free': ('mysql.innodb.buffer_pool_wait_free', MONOTONIC),
    'Innodb_buffer_pool_write_requests': ('mysql.innodb.buffer_pool_write_requests', RATE),
    'Innodb_checkpoint_age': ('mysql.innodb.checkpoint_age', GAUGE),
    'Innodb_current_transactions': ('mysql.innodb.current_transactions', GAUGE),
    'Innodb_data_fsyncs': ('mysql.innodb.data_fsyncs', RATE),
    'Innodb_data_pending_fsyncs': ('mysql.innodb.data_pending_fsyncs', GAUGE),
    'Innodb_data_pending_reads': ('mysql.innodb.data_pending_reads', GAUGE),
    'Innodb_data_pending_writes': ('mysql.innodb.data_pending_writes', GAUGE),
    'Innodb_data_read': ('mysql.innodb.data_read', RATE),
    'Innodb_data_written': ('mysql.innodb.data_written', RATE),
    'Innodb_dblwr_pages_written': ('mysql.innodb.dblwr_pages_written', RATE),
    'Innodb_dblwr_writes': ('mysql.innodb.dblwr_writes', RATE),
    'Innodb_hash_index_cells_total': ('mysql.innodb.hash_index_cells_total', GAUGE),
    'Innodb_hash_index_cells_used': ('mysql.innodb.hash_index_cells_used', GAUGE),
    'Innodb_history_list_length': ('mysql.innodb.history_list_length', GAUGE),
    'Innodb_ibuf_free_list': ('mysql.innodb.ibuf_free_list', GAUGE),
    'Innodb_ibuf_merged': ('mysql.innodb.ibuf_merged', RATE),
    'Innodb_ibuf_merged_delete_marks': ('mysql.innodb.ibuf_merged_delete_marks', RATE),
    'Innodb_ibuf_merged_deletes': ('mysql.innodb.ibuf_merged_deletes', RATE),
    'Innodb_ibuf_merged_inserts': ('mysql.innodb.ibuf_merged_inserts', RATE),
    'Innodb_ibuf_merges': ('mysql.innodb.ibuf_merges', RATE),
    'Innodb_ibuf_segment_size': ('mysql.innodb.ibuf_segment_size', GAUGE),
    'Innodb_ibuf_size': ('mysql.innodb.ibuf_size', GAUGE),
    'Innodb_lock_structs': ('mysql.innodb.lock_structs', RATE),
    'Innodb_locked_tables': ('mysql.innodb.locked_tables', GAUGE),
    'Innodb_locked_transactions': ('mysql.innodb.locked_transactions', GAUGE),
    'Innodb_log_waits': ('mysql.innodb.log_waits', RATE),
    'Innodb_log_write_requests': ('mysql.innodb.log_write_requests', RATE),
    'Innodb_log_writes': ('mysql.innodb.log_writes', RATE),
    'Innodb_lsn_current': ('mysql.innodb.lsn_current', RATE),
    'Innodb_lsn_flushed': ('mysql.innodb.lsn_flushed', RATE),
    'Innodb_lsn_last_checkpoint': ('mysql.innodb.lsn_last_checkpoint', RATE),
    'Innodb_mem_adaptive_hash': ('mysql.innodb.mem_adaptive_hash', GAUGE),
    'Innodb_mem_additional_pool': ('mysql.innodb.mem_additional_pool', GAUGE),
    'Innodb_mem_dictionary': ('mysql.innodb.mem_dictionary', GAUGE),
    'Innodb_mem_file_system': ('mysql.innodb.mem_file_system', GAUGE),
    'Innodb_mem_lock_system': ('mysql.innodb.mem_lock_system', GAUGE),
    'Innodb_mem_page_hash': ('mysql.innodb.mem_page_hash', GAUGE),
    'Innodb_mem_recovery_system': ('mysql.innodb.mem_recovery_system', GAUGE),
    'Innodb_mem_thread_hash': ('mysql.innodb.mem_thread_hash', GAUGE),
    'Innodb_mem_total': ('mysql.innodb.mem_total', GAUGE),
    'Innodb_os_file_fsyncs': ('mysql.innodb.os_file_fsyncs', RATE),
    'Innodb_os_file_reads': ('mysql.innodb.os_file_reads', RATE),
    'Innodb_os_file_writes': ('mysql.innodb.os_file_writes', RATE),
    'Innodb_os_log_pending_fsyncs': ('mysql.innodb.os_log_pending_fsyncs', GAUGE),
    'Innodb_os_log_pending_writes': ('mysql.innodb.os_log_pending_writes', GAUGE),
    'Innodb_os_log_written': ('mysql.innodb.os_log_written', RATE),
    'Innodb_pages_created': ('mysql.innodb.pages_created', RATE),
    'Innodb_pages_read': ('mysql.innodb.pages_read', RATE),
```

```
'Innodb_pages_written': ('mysql.innodb.pages_written', RATE),
'Innodb_pending_aio_log_ios': ('mysql.innodb.pending_aio_log_ios', GAUGE),
'Innodb_pending_aio_sync_ios': ('mysql.innodb.pending_aio_sync_ios', GAUGE),
'Innodb_pending_buffer_pool_flushes': ('mysql.innodb.pending_buffer_pool_flushes', GAUGE),
'Innodb_pending_checkpoint_writes': ('mysql.innodb.pending_checkpoint_writes', GAUGE),
'Innodb_pending_ibuf_aio_reads': ('mysql.innodb.pending_ibuf_aio_reads', GAUGE),
'Innodb_pending_log_flushes': ('mysql.innodb.pending_log_flushes', GAUGE),
'Innodb_pending_log_writes': ('mysql.innodb.pending_log_writes', GAUGE),
'Innodb_pending_normal_aio_reads': ('mysql.innodb.pending_normal_aio_reads', GAUGE),
'Innodb_pending_normal_aio_writes': ('mysql.innodb.pending_normal_aio_writes', GAUGE),
'Innodb_queries_inside': ('mysql.innodb.queries_inside', GAUGE),
'Innodb_queries_queued': ('mysql.innodb.queries_queued', GAUGE),
'Innodb_read_views': ('mysql.innodb.read_views', GAUGE),
'Innodb_rows_deleted': ('mysql.innodb.rows_deleted', RATE),
'Innodb_rows_inserted': ('mysql.innodb.rows_inserted', RATE),
'Innodb_rows_read': ('mysql.innodb.rows_read', RATE),
'Innodb_rows_updated': ('mysql.innodb.rows_updated', RATE),
'Innodb_s_lock_os_waits': ('mysql.innodb.s_lock_os_waits', RATE),
'Innodb_s_lock_spin_rounds': ('mysql.innodb.s_lock_spin_rounds', RATE),
'Innodb_s_lock_spin_waits': ('mysql.innodb.s_lock_spin_waits', RATE),
'Innodb_semaphore_wait_time': ('mysql.innodb.semaphore_wait_time', GAUGE),
'Innodb_semaphore_waits': ('mysql.innodb.semaphore_waits', GAUGE),
'Innodb_tables_in_use': ('mysql.innodb.tables_in_use', GAUGE),
'Innodb_x_lock_os_waits': ('mysql.innodb.x_lock_os_waits', RATE),
```

```
    'Innodb_x_lock_spin_rounds': ('mysql.innodb.x_lock_spin_rounds', RATE),
    'Innodb_x_lock_spin_waits': ('mysql.innodb.x_lock_spin_waits', RATE),
}
```

GALERA_VARS

MySQL/Galera 是一种多主同步集群，但只限于使用 MySQL/InnoDB 引擎

```
GALERA_VARS = {
    'wsrep_cluster_size': ('mysql.galera.wsrep_cluster_size', GAUGE),
    'wsrep_local_recv_queue_avg': ('mysql.galera.wsrep_local_recv_queue_avg', GAUGE),
    'wsrep_flow_control_paused': ('mysql.galera.wsrep_flow_control_paused', GAUGE),
    'wsrep_cert_deps_distance': ('mysql.galera.wsrep_cert_deps_distance', GAUGE),
    'wsrep_local_send_queue_avg': ('mysql.galera.wsrep_local_send_queue_avg', GAUGE),
}
```

PERFORMANCE_VARS

方法 _get_query_exec_time_95th_us

方法 _query_exec_time_per_schema

```
PERFORMANCE_VARS = {
    'query_run_time_avg': ('mysql.performance.query_run_time.avg', GAUGE),
    'perf_digest_95th_percentile_avg_us': ('mysql.performance.digest_95th_percentile.avg_us', GAUGE),
}
```

```
perf_digest_95th_percentile_avg_us:
SELECT s2.avg_us avg_us, IFNULL(SUM(s1.cnt)/NULLIF( (SELECT COUNT(*) FROM
performance_schema.events_statements_summary_by_digest), 0), 0) percentile FROM (SELECT COUNT(*) cnt,
ROUND(avg_timer_wait/1000000) AS avg_us FROM
performance_schema.events_statements_summary_by_digest GROUP BY avg_us) AS s1 JOIN (SELECT COUNT(*)
cnt, ROUND(avg_timer_wait/1000000) AS avg_us FROM
performance_schema.events_statements_summary_by_digest GROUP BY avg_us) AS s2 ON s1.avg_us <=
s2.avg_us GROUP BY s2.avg_us HAVING percentile > 0.95 ORDER BY percentile LIMIT 1;

query_run_time_avg:
SELECT schema_name, SUM(count_star) cnt, ROUND(AVG(avg_timer_wait)/1000000) AS avg_us FROM
performance_schema.events_statements_summary_by_digest WHERE schema_name IS NOT NULL GROUP BY
schema_name
```

SCHEMA_VARS

方法 _query_size_per_schema :

```
SCHEMA_VARS = {
    'information_schema_size': ('mysql.info.schema.size', GAUGE),
}

# 执行sql: "SELECT table_schema,SUM(data_length+index_length)/1024/1024 AS total_mb FROM
information_schema.tables GROUP BY table_schema"
# 得到数据库相关字节大小返回，单位是mb
```

REPLICA_VARS

mysql 主从复制相关指标：

seconds_behind_master：mysql从库时延

slaves_connected：从库连接数

```
REPLICA_VARS = {
    'Seconds_Behind_Master': ('mysql.replication.seconds_behind_master', GAUGE),
    'Slaves_connected': ('mysql.replication.slaves_connected', COUNT),
}

# Slaves_connected：    SELECT THREAD_ID, NAME FROM performance_schema.threads WHERE NAME LIKE '%worker' 或者 SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST WHERE COMMAND LIKE '%Binlog dump%'，根据查看结果count得到
# Seconds_Behind_Master:
```

SYNTHETIC_VARS

mysql query catch相关指标：

在方法_compute_synthetic_results中：依赖于Qcache_hits,Qcache_inserts,Qcache_not_cached,Qcache_hits计算而得

```
SYNTHETIC_VARS = {
    'Qcache_utilization': ('mysql.performance.qcache.utilization', GAUGE),
    'Qcache_instant_utilization': ('mysql.performance.qcache.utilization.instant', GAUGE),
}

#其中：
# Qcache_utilization = (float(results['Qcache_hits']) / (int(results['Qcache_inserts']) + int(results['Qcache_not_cached']) + int(results['Qcache_hits'])) * 100)
# Qcache_instant_utilization = ((float(results['Qcache_hits']) - self._qcache_hits) / ((int(results['Qcache_inserts']) - self._qcache_inserts) + (int(results['Qcache_not_cached']) - self._qcache_not_cached) + (int(results['Qcache_hits']) - self._qcache_hits)) *  100)
```

各项指标的采集方式

mysql.py里面采集mysql指标写在check方法里面，check方法里面调用了方法分别采集：

- self._collect_metadata
- self._collect_metrics
- self._collect_system_metrics

self._collect_metadata(db, host)

- get_version函数获取mysql的version，通过sql语句SELECT VERSION()
- 调用父类AgentCheck的service_metadata方法save下来此mysql的version。

self._collect_metrics(host, db, tags, options, queries)

所有的指标都是在这个方法里面采集。具体采集方式在上面。

self._collect_system_metrics(host, db, tags)

- 首先判断监测的mysql是不是本机安装的mysql，如果是本机的mysql，获取当前mysqld的pid；
- 如果是监测远程的mysql，则根据psutil模块获取三个指标
  mysql.performance.user_time
  mysql.performance.kernel_time
  mysql.performance.cpu_time

dd-agent框架部份在windows与linux上运行

dd-agent各框架

1. collector (agent.py) Collector 会检查当前运行机器的集成环境，抓取系统性能指标，如内存和 CPU 数据。

2. Dogstatsd (dogstatsd.py) 这是StatsD的后台服务器，它致力于收集从你代码中发送出去的本地性能指标。

3. Forwarder (ddagent.py) Forwarder 负责把Dogstatsd和Collector收集到的数据推到一个队列中，这些数据将会被发往 Datadog。

4. SupervisorD
由一个单独的管理进程控制。我们把它与其他组件分隔开来，因此如果你担心资源消耗而不想运行所有组件的话（虽然我们建议你这么做），可以单独运行它。

Linux

环境：ubuntu12.04
Python: 2.7.3

一键安装运行：

```
sudo DD_API_KEY=86830b460bd64cce990def8a357c38fd REPERTORY_URL="https://monitor.uyuntest.cn" bash -c "$(curl -L https://monitor.uyuntest.cn/downloads/agent/install_agent.sh)"

 sudo /etc/init.d/datadog-agent start
 sudo /etc/init.d/datadog-agent stop
 sudo /etc/init.d/datadog-agent restart
 sudo /etc/init.d/datadog-agent info
```

源码安装运行：

```
#相关依赖安装
 wget
https://pypi.python.org/packages/fe/f6/da82dee704be089b6c3f5a7eb17a5f7c67e4fb6d030405dde392dc846714/python-etcd-0.4.3.tar.gz
 wget
https://pypi.python.org/packages/1f/30/f0455f156b68aba3b34e8d77d41c0e1f01a2a5e6d06aeaa4ee1c6db189b1/docker-py-0.5.1.tar.gz
 wget
https://pypi.python.org/packages/45/94/c7f52aa952b18b457bb03446818d284cddb355fface47d91c4d8cebd6693/websocket-client-0.4.tar.gz


 tar -zxvf python-etcd-0.4.3.tar.gz
 tar -zxvf docker-py-0.5.1.tar.gz
 tar -zxvf websocket-client-0.4.tar.gz


 cd python-etcd-0.4.3
 python setup.py install
 cd ../docker-py-0.5.1
 python setup.py install
 cd ../websocket-client-0.4
 python setup.py install


 pip install dnspython
 pip install python-consul
 pip install ntplib
 pip install uptime
 pip install pyyaml
 pip install tornado

 #下拉源代码
 git clone https://github.com/DataDog/dd-agent.git
 cd dd-agent/

 #输入配置内容
 vim datadog.conf
 #运行dd三个框架
 python agent.py start
 python dogstatsd.py
 python ddagent.py
```

三个进程已经起来了

```
root      6644  0.0  2.0 155428 20208 pts/2    Tl   15:47   0:00 python dogstatsd.py
root      6653  0.0  2.1 158064 21112 pts/2    T    15:47   0:00 python ddagent.py
root      6665  0.2  2.2  90860 22852 ?        S    15:47   0:05 python agent.py start
root      7471  0.1  1.2  61816 12732 pts/2    S+   16:19   0:00 python testWeb.py
root      7504  0.0  0.0  13580   936 pts/0    S+   16:20   0:00 grep --color=auto python
```

==========

Windows

环境：Win7

Python: 2.7.12

一键安装运行：

1. 直接下载 https://monitor.uyuntest.cn/downloads/agent/Windows/Datadog-Agent-win32.msi，下载完成后复制到D盘根目录开始下载，完成后复制到被管主机D:\根目录。
 2. msiexec /qb /i "D:\Datadog-Agent-win32.msi" APIKEY="86830b460bd64cce990def8a357c38fd" DDURL="https://monitor.uyuntest.cn/api/v2/gateway/dd-agent"

源码安装运行：

win32/agent.py：window下面的collector，forwarder，dogstatsd，jmxfetch都在agent.py里面

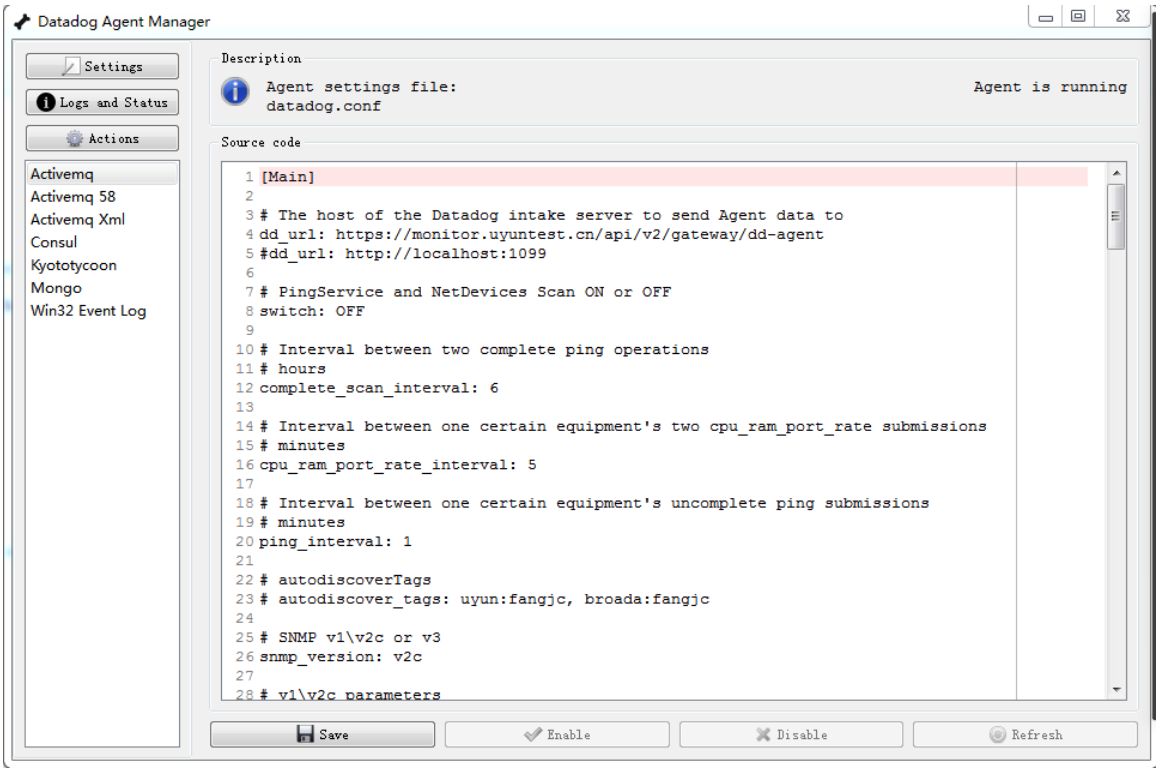
win32/gui.py：开启管理页面

依赖：guidata，pywin32，pyQt，SIP

依赖装完后，

```
python agent.py install
 python agent.py start
 python gui.py
```

```
ddagent.exe                   804 Services
ddagent.exe                  1756 Services
ddagent.exe                  9332 Services
```



坑：

1. 不管是win上还是Linux上，要确保讲之前安装的ddagent卸载干净，不然跑的还是之前安装的dd-agent。

2. agent.py 命令行传进的系统参数，win32和Linux上不一样的，

```
win32: python agent.py  install|update|remove|start|stop|restart|stop|debug
       linux: python agent.py  start|stop|restart|status|foreground|info|check|configcheck|jmx|flare
```

3. win下面第一次要跑 python agent.py install的，这步一直没发现，以至于一直报gui线程同步的错误，阻断了很久。

4.
依赖的安装最好用官方提供一键安装后打包好的，这样依赖版本完全吻合，而且省去自己去找的痛苦，比如win32下面，依赖都打包位于Datadog\Datadog Agent\files\library.zip里面

dd-agent扩展监测weblogic能力

- 安装weblogic
- 监测weblogic

安装weblogic

环境：ubuntu14

JAVA：JDK1.8

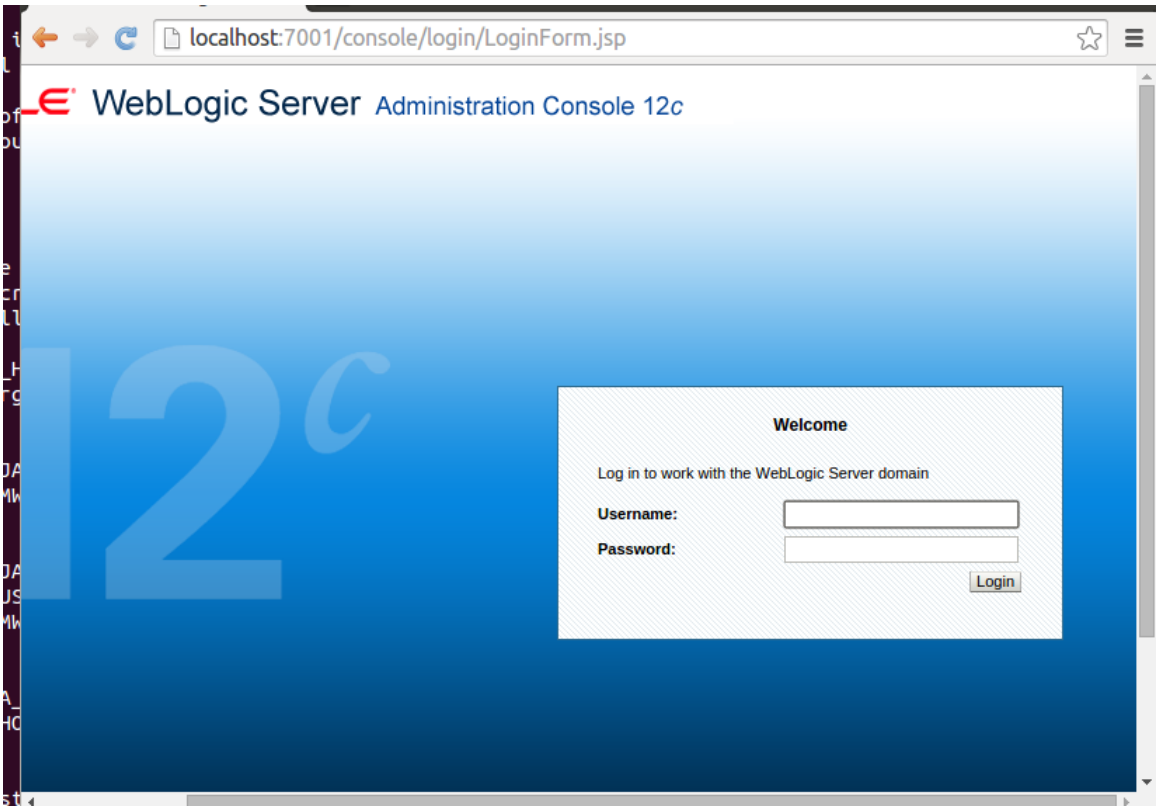
download_url：http://www.oracle.com/technetwork/cn/middleware/ias/downloads/wls-main-091116-zhs.htm

过程：

- 先去oracle 下载 weblogic安装文件l，得到wls1213_dev.zip
- unzip 解压wls1213_dev.zip
- cd wls1213
- export MW_HOME=/home/sunc/Download/wls12130
- export JAVA_HOME=/usr/jdk/jdk1.8.0_101
- ./config.sh
- 一路next...
- 当控制台上显示<Server state changed to RESUMING.>时。你就能够从浏览器中訪问 http://localhost:7001/console 了

```
<Sep 27, 2016 5:20:16 PM CST> <Notice> <Log Management> <BEA-170027> <The server has successfully establishe
<Sep 27, 2016 5:20:17 PM CST> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to ADMIN.>
<Sep 27, 2016 5:20:17 PM CST> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RESUMING.>
<Sep 27, 2016 5:20:17 PM CST> <Notice> <Server> <BEA-002613> <Channel "Default[1]" is now listening on 192.
<Sep 27, 2016 5:20:17 PM CST> <Notice> <Server> <BEA-002613> <Channel "Default[4]" is now listening on 127.
<Sep 27, 2016 5:20:17 PM CST> <Notice> <Server> <BEA-002613> <Channel "Default[3]" is now listening on 0:0:
<Sep 27, 2016 5:20:17 PM CST> <Error> <Server> <BEA-002606> <The server is unable to create a server socket
ncorrect or another process is using port 7001: java.net.BindException: Cannot assign requested address>
<Sep 27, 2016 5:20:17 PM CST> <Notice> <Server> <BEA-002613> <Channel "Default[2]" is now listening on fe80
<Sep 27, 2016 5:20:17 PM CST> <Notice> <WebLogicServer> <BEA-000331> <Started the WebLogic Server Administra

<Sep 27, 2016 5:20:17 PM CST> <Notice> <WebLogicServer> <BEA-000360> <The server started in RUNNING mode.>
<Sep 27, 2016 5:20:17 PM CST> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING.>
```



安装成功

监测weblogic

doing

Linux环境下agent采集详情

agent采集指标汇总

1.Collector (agent.py) -- Collector 会采集当前运行机器的集成环境，抓取系统性能指标，如内存和 CPU 数据。

2.agent.py Agent类中有两个私有变量，self._checksd，加载checks.扩展采集功能和self.collector，基础采集模块。

3.checks.collector.py Collector类的run方法采集定义的指标，调用Agentpayload类的emit方法将数据发送给Forwarder

基础监测

checks.collector.py Collector类私有变量里定义了基础采集的指标：

主要分为4个大类，System checks, Old-system checks, Custom metrics checks, Resource checks

collector.py

```
class Collector(object):
    """
    The collector is responsible for collecting data from each check and
    passing it along to the emitters, who send it to their final destination.
    """
    def __init__(self, agentConfig, emitters, systemStats, hostname):
        ......
        # Unix System Checks
        self._unix_system_checks = {
            'io': u.IO(log),
            'load': u.Load(log),
            'memory': u.Memory(log),
            'processes': u.Processes(log),
            'cpu': u.Cpu(log),
            'system': u.System(log)
        }
        # Old-style metric checks
        self._ganglia = Ganglia(log)
        self._dogstream = Dogstreams.init(log, self.agentConfig)
        self._ddforwarder = DdForwarder(log, self.agentConfig)
        # Agent performance metrics check
        self._agent_metrics = None
        self._metrics_checks = []
        # Custom metric checks
        for module_spec in [s.strip() for s in self.agentConfig.get('custom_checks', '').split(',')]:
            if len(module_spec) == 0:
                continue
            try:
                self._metrics_checks.append(modules.load(module_spec, 'Check')(log))
                log.info("Registered custom check %s" % module_spec)
                log.warning("Old format custom checks are deprecated. They should be moved to the checks.d
interface as old custom checks will be removed in a next version")
            except Exception:
                log.exception('Unable to load custom check module %s' % module_spec)
        # Resource Checks
        self._resources_checks = [
            ResProcesses(log, self.agentConfig)
        ]
```

System checks

checks.system.win32.py win32系统指标采集

checks.system.unix.py unix系统指标采集

unix.py 中分别定义了6个类，IO Load Memory Processes CPU
System，这6个类中分别都有一个check方法来获取相关指标，在collector的run方法调用unix.py中6个check方法来获取系统指标。

> unix.py

1. Io：io采集返回filtered_io，这是一个字典，filtered_io[device] = stats，分别是设备和对应的io速率
2. Load：负载返回字典类似于
 { 'system.load.1': float(load[0]),
   'system.load.5': float(load[1]),
   'system.load.15': float(load[2]),
   'system.load.norm.1': float(load[0])/cores,
   'system.load.norm.5': float(load[1])/cores,
   'system.load.norm.15': float(load[2])/cores,}
3. Memory: 返回字典memData
 { memData['physTotal'] = int(meminfo.get('MemTotal', 0)) / 1024
   memData['physFree'] = int(meminfo.get('MemFree', 0)) / 1024
   memData['physBuffers'] = int(meminfo.get('Buffers', 0)) / 1024
   memData['physCached'] = int(meminfo.get('Cached', 0)) / 1024
   memData['physShared'] = int(meminfo.get('Shmem', 0)) / 1024
   memData['physSlab'] = int(meminfo.get('Slab', 0)) / 1024
   memData['physPageTables'] = int(meminfo.get('PageTables', 0)) / 1024
   memData['physUsed'] = memData['physTotal'] - memData['physFree']}
4. Processes: 返回字典
  {'processes':   processes,
   'apiKey':   agentConfig['api_key'],
   'host':   get_hostname(agentConfig)}
5. Cpu： 返回字典cpu信息 %usr %nice %sys %iowait %irq %soft %steal %guest %idle
6. System： 返回字典system.uptime

Old-system checks

> Old-system checks

1. GangliaData: ganglia.py Ganglia类check方法
2. dogstreamData: datadog.py Dogstream类check方法
3. ddforwarderData: datadog.py DdForwarder类check方法

Custom metrics checks

> custom metrics checks

调用modules.py
的load方法，传入参数custom_checks，这部分和扩展采集相关，custom_checks是旧版的扩展模块脚本：配置键值对

Resource checks

> Resource checks

调用resources.processes.py的Processes的check方法

扩展监测

1. 按照kb 如何扩展datadog agent采集能力中的方法，可以打开扩展监测内容，checks.d和conf.d里面分别包括了扩展采集的脚本以及配置文件。
2. agent.py 会 import config.py中的load_check_directory方法，该方法调用check_yaml会传入默认路径下的yaml配置文件。
3. 返回一个包含了所有yaml文件中键值对配置信息的实例对象，由键值对构成，类似一个层次化结构，最上层有两个键值对，分别是'init_config'和'instances'键值对，对应的值也是键值对。此对象将用于初始化监测实例。

checks.d 里面包含了扩展监测的性能的监测脚本：

checks.d

```
root@ubuntu:/etc/dd-agent/conf.d# ls /opt/datadog-agent/agent/checks.d/
activemq_xml.py   directory.py      go_expvar.py      kafka_consumer.py     mesos.py
oracle_metrics_execute_temp.xml  riak.py          teamcity.py          yarn.py
agent_metrics.py  disk.py           gunicorn.py       kubernetes.py         mesos_slave.py  pgbouncer.py
snmp.py          test_cx_oracle.py   zk.py
apache.py         dns_check.py      haproxy.py        kyototycoon.py        mongo.py        php_fpm.py
sqlserver.py     test_ping.py
btrfs.py          docker_daemon.py  hdfs_datanode.py  lighttpd.py           mysql.py        postfix.py
ssh_check.py     tokumx.py
cacti.py          docker.py         hdfs_namenode.py  linux_proc_extras.py  nagios.py       postgres.py
statsd.py        varnish.py
ceph.py           elastic.py        hdfs.py           mapreduce.py          network.py      process.py
supervisord.py   vsphere.py
consul.py         etcd.py           http_check.py     marathon.py           nginx.py        rabbitmq.py
system_core.py   win32_event_log.py
couchbase.py      fluentd.py        iis.py            mcache.py             ntp.py          redisdb.py
system_swap.py   windows_service.py
couch.py          gearmand.py       jenkins.py        mesos_master.py       openstack.py    riakcs.py
tcp_check.py     wmi_check.py
```

conf.d
里面包含这些扩展指标的配置信息，根据上面load_check_directory方法的原理，使用扩展采集功能，需要将相应的yaml配置文件后缀exmaple删除。

conf.d

```
root@ubuntu:/etc/dd-agent/conf.d# ls
activemq_58.yaml.example     dns_check.yaml.example       http_check.yaml.example
mesos_slave.yaml.example  rabbitmq.yaml.example     test_collector.yaml.example
activemq_xml.yaml.example    docker_daemon.yaml.example  iis.yaml.example                mesos.yaml.example
redisdb.yaml.example       test_cx_oracle.yaml.example
activemq.yaml.example        docker.yaml.example          jenkins.yaml.example            mongo.yaml.example
riakcs.yaml.example        test_ping.yaml.example
agent_metrics.yaml.default   elastic.yaml.example         jmx.yaml.example                mysql.yaml.example
riak.yaml.example          tokumx.yaml.example
apache.yaml.example          etcd.yaml.example            kafka_consumer.yaml.example     nagios.yaml.example
snmp.yaml.example          tomcat.yaml.example
btrfs.yaml.example           fluentd.yaml.example         kafka.yaml.example              network.yaml.default
solr.yaml.example          varnish.yaml.example
cacti.yaml.example           gearmand.yaml.example        kubernetes.yaml.example         nginx.yaml.example
sqlserver.yaml.example     vsphere.yaml.example
cassandra.yaml.example       go_expvar.yaml.example       kyototycoon.yaml.example        ntp.yaml.default
ssh_check.yaml.example     win32_event_log.yaml.example
ceph.yaml.example            go-metro.yaml.example        lighttpd.yaml.example           openstack.yaml.example
statsd.yaml.example        windows_service.yaml.example
consul.yaml.example          gunicorn.yaml.example        linux_proc_extras.yaml.example
pgbouncer.yaml.example     supervisord.yaml.example   wmi_check.yaml.example
couchbase.yaml.example       haproxy.yaml.example         mapreduce.yaml.example          php_fpm.yaml.example
system_core.yaml.example   yarn.yaml.example
couch.yaml.example           hdfs_datanode.yaml.example   marathon.yaml.example           postfix.yaml.example
system_swap.yaml.example   zk.yaml.example
directory.yaml.example       hdfs_namenode.yaml.example   mcache.yaml.example
postgres.yaml.example      tcp_check.yaml.example
disk.yaml.default            hdfs.yaml.example            mesos_master.yaml.example       process.yaml.example
teamcity.yaml.example
```

agent采集频率

collector读取config.py里面get_config方法获取agent的基本参数，采集频率可以在/opt/datadog/config.py里面设置。

参数是 DEFAULT_CHECK_FREQUENCY 默认的采集频率是15秒一次

config.py

```
# CONSTANTS
AGENT_VERSION = "5.7.4"
DATADOG_CONF = "datadog.conf"
UNIX_CONFIG_PATH = '/etc/dd-agent'
MAC_CONFIG_PATH = '/opt/datadog-agent/etc'
DEFAULT_CHECK_FREQUENCY = 15                # seconds 采集频率
LOGGING_MAX_BYTES = 5 * 1024 * 1024         # 日志文件每达到5M就归档一次

.....

def get_config(parse_args=True, cfg_path=None, options=None):
    if parse_args:
        options, _ = get_parsed_args()
    # General config
    agentConfig = {
        'check_freq': DEFAULT_CHECK_FREQUENCY,
        'dogstatsd_port': 8125,
        'dogstatsd_target': 'http://localhost:17123',
        'graphite_listen_port': None,
        'hostname': None,
        'listen_port': None,
        'tags': None,
        'use_ec2_instance_id': False,  # DEPRECATED
        'version': get_version(),
        'watchdog': True,
        'additional_checksd': '/etc/dd-agent/checks.d/',
        'bind_host': get_default_bind_host(),
        'statsd_metric_namespace': None,
        'utf8_decoding': False
    }
```

根据collector日志文件也可以得到结果，默认的采集频率是15秒一次

```
2016-08-24 19:59:31 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 20.6, metrics count: 39, events count: 0
2016-08-24 19:59:46 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 66.5, metrics count: 39, events count: 0
2016-08-24 20:00:01 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 21.6, metrics count: 39, events count: 0
2016-08-24 20:00:16 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 44.6, metrics count: 39, events count: 0
2016-08-24 20:00:31 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 24.1, metrics count: 39, events count: 0
2016-08-24 20:00:46 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 36.1, metrics count: 39, events count: 0
2016-08-24 20:01:01 PDT | INFO | dd.collector | checks.agent_metrics(agent_metrics.py:148) | CPU consumed (%) is high: 55.4, metrics count: 39, events count: 0
```

agent通信网络流量监测

方法：agent对外流量是Forwarder向datadog产生，因此可以监控forwarder进程在一段时间内的流量情况。（当然监听端口也可以）

工具：nethogs

时间： 10min

过程：

1. sudo apt-get install nethogs # 安装
2. date # 记录下开始时间
3. nethogs # 运行
4. 10分钟后再看

```
NetHogs version 0.8.0

  PID USER     PROGRAM                                                                DEV        SENT      RECEIVED
11029 dd-agent python                                                                 eth0       379.323     540.248 KB
```

结论：10分钟内得到forwarder接收和发送的流量分别是，540.2kb和379.3kb，接收和发送的速率大概0.9kb/s，0.63kb/s

---

说明：貌似上诉方法计算forwarder接收流量的时候不是很准确，因为包含了forwarder从collector和dogstatsd发送过来的流量，属于内部流量，因此换用监控端口的方式

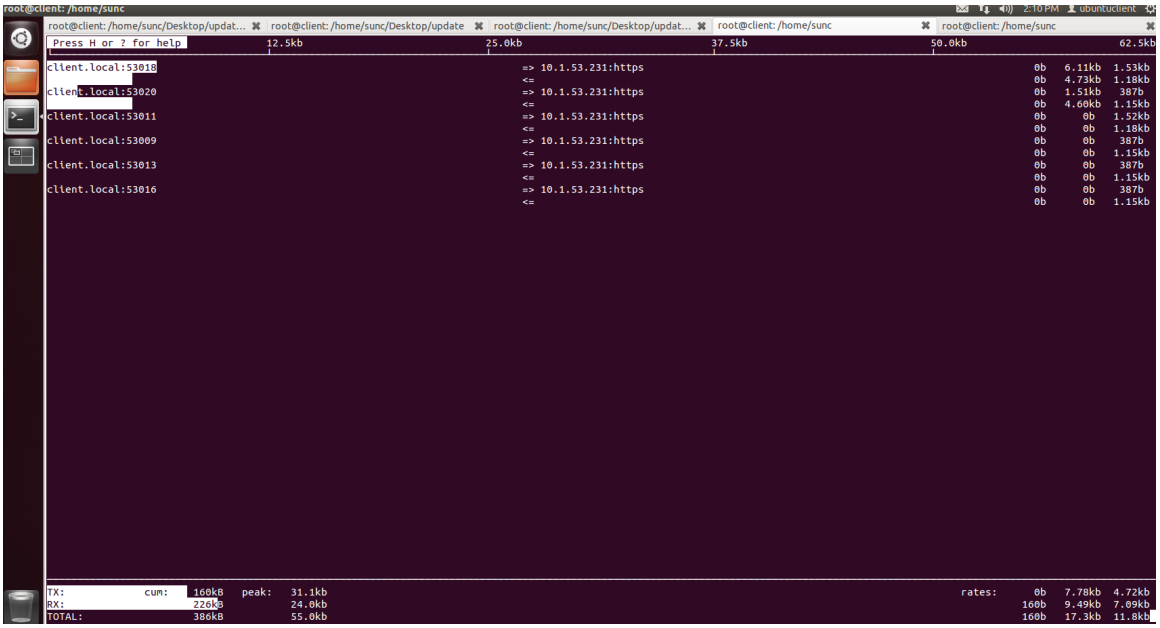
方法：forwarder和server间通信采用HTTPS，默认端口是443

工具：iftop，具体操作可以参考 http://my.oschina.net/lionel45/blog/261234

时间：300秒

过程：

1. 安装 sudo apt-get install iftop
2. 计时4分钟
3. iftop -P (显示端口)
4. 进入iftop界面，按l之后输入ip=10.1.53.231(次ip为与之通信的server IP，端口显示https)
5. 240秒后，记录TX和RX分别是发送和接收总流量分别是160kb和226kb



6. 得到平均发送流量和接收流量是 0.67kb/s和0.94kb/s

Forwarder与Server之间流量

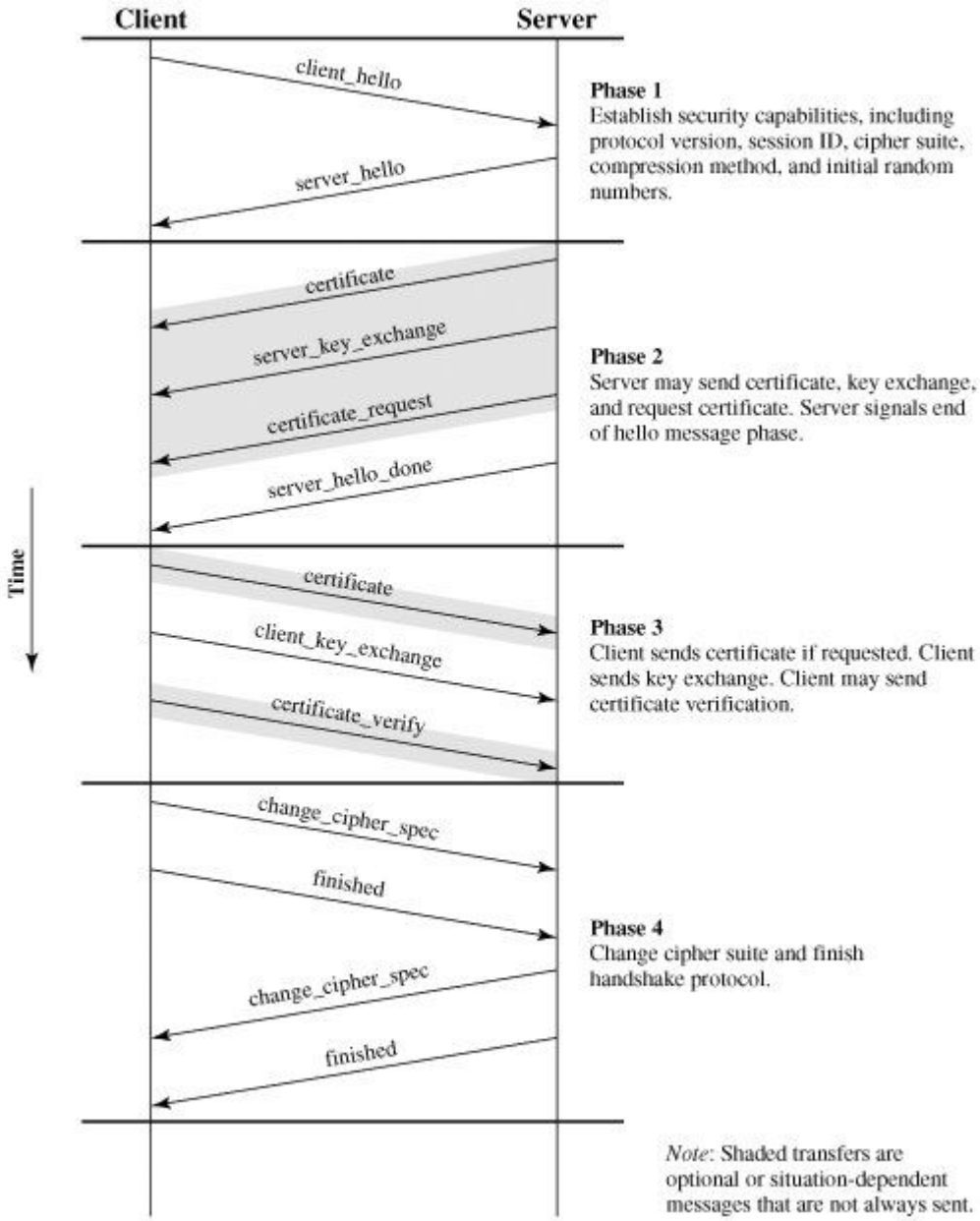
基于上面的统计，我们观察到一个现象，就是forwarder发送出去的流量比接受到的流量还要小，这是怎么回事呢，按道理说发出去的是数据，接收到的是响应，应该发出大于接受的才对。

这个源资源forwarder与server之间的通信是HTTPS协议。

HTTPS简单来说是加密后的HTTP协议，数据采用SSL加密，具有较复杂的加密，认证机制，因此建立连接通信过程比HTTP复杂很多，因此传输相同大小的数据包，总的通信流量要比HTTP来的大

HTTPS 连接

连接建立详细过程可以参考这篇博客 http://www.cnblogs.com/hrhguanli/p/3804146.html



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.195.141	192.168.195.2	DNS	79	Standard query A monitor.uyuntest.cn
2	0.000853	192.168.195.2	192.168.195.141	DNS	95	Standard query response A 10.1.53.231
3	0.001237	192.168.195.141	10.1.53.231	TCP	74	51174 > https [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=896982 TSecr=0 WS=16
4	0.001919	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.001962	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=1 Ack=1 Win=14600 Len=0
6	0.003319	192.168.195.141	10.1.53.231	TLSv1.2	571	Client Hello
7	0.003482	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=1 Ack=518 Win=64240 Len=0
8	0.007459	10.1.53.231	192.168.195.141	TLSv1.2	1514	Server Hello
9	0.007474	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=518 Ack=1461 Win=17520 Len=0
10	0.007521	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
11	0.007528	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=518 Ack=2921 Win=20440 Len=0
12	0.007560	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
13	0.007565	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=518 Ack=4381 Win=23360 Len=0
14	0.007599	10.1.53.231	192.168.195.141	TLSv1.2	692	Certificate, Server Key Exchange, Server Hello Done
15	0.007604	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=518 Ack=5019 Win=26280 Len=0
16	0.009581	192.168.195.141	10.1.53.231	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
17	0.009783	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5019 Ack=644 Win=64240 Len=0
18	0.011334	10.1.53.231	192.168.195.141	TLSv1.2	312	Encrypted Handshake Message, Change Cipher Spec, Encrypted Handshake Message
19	0.011868	192.168.195.141	10.1.53.231	TCP	7354	[TCP segment of a reassembled PDU]
20	0.011960	192.168.195.141	10.1.53.231	TLSv1.2	483	Application Data
21	0.012092	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=2104 Win=64240 Len=0
22	0.012100	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=3564 Win=64240 Len=0
23	0.012311	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=5024 Win=64240 Len=0
24	0.012323	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=6484 Win=64240 Len=0
25	0.012326	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=7944 Win=64240 Len=0
26	0.012329	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=5277 Ack=8373 Win=64240 Len=0
27	0.017033	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
28	0.017050	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
29	0.017057	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=8373 Ack=8197 Win=35040 Len=0
30	0.017099	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
31	0.017104	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
32	0.017107	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=8373 Ack=11117 Win=40880 Len=0
33	0.017138	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
34	0.017143	10.1.53.231	192.168.195.141	TCP	1514	[TCP segment of a reassembled PDU]
35	0.017147	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=8373 Ack=14037 Win=46720 Len=0
36	0.017177	10.1.53.231	192.168.195.141	TLSv1.2	278	Application Data
37	0.017941	192.168.195.141	10.1.53.231	TCP	54	51174 > https [FIN, ACK] Seq=8373 Ack=14261 Win=49640 Len=0
38	0.019232	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [ACK] Seq=14261 Ack=8374 Win=64239 Len=0
39	0.019252	10.1.53.231	192.168.195.141	TCP	60	https > 51174 [FIN, PSH, ACK] Seq=14261 Ack=8374 Win=64239 Len=0
40	0.019259	192.168.195.141	10.1.53.231	TCP	54	51174 > https [ACK] Seq=8374 Ack=14262 Win=49640 Len=0

上面左图是HTTPS client和server建立连接的过程，右图是wireshark抓取到的https数据包的一次通信过程，可以对应起来看。

作出统计后得到，client与server一次连接并通信，client发出9281长度的数据，server发出15568长度的数据。就符合了上面我们用iftop流量监测工具得到的forwarder发出的流量要小于接收数据的流量的现象。

Python

使用fabric进程远程部署安装

- fabric
    - 简介：
    - api：
    - 管理SSH密码、用户、端口：
- 部署实现
    - 需求
    - 配置和代码
    - 执行过程
    - 说明
- 附件

fabric

简介：

Fabric是一个python的远程部署工具，最大的特点是不需要ssh登录机器，在本地即可为远程机器安装。

Fabric的所有操作都是基于SSH执行的，它会提示输入口令，非常安全。最好的办法是在指定的部署服务器上用证书配置无密码的ssh连接。

默认当命令执行失败时，Fabric会停止执行后续命令。如果想忽略执行失败继续运行，用with settings(warn_only=True):执行命令，这样Fabric只会打出警告信息而不会中断执行。

文档地址： http://www.phperz.com/article/15/1005/161116.html

api：

- local()方法：在本地执行命令
- run()方法：远程执行命令
- put()方法：可以把本地文件上传到远程，当需要在远程指定当前目录时，只需用with cd('dir'):即可。

管理SSH密码、用户、端口：

Fabric提供了环境变量的字典env，包含了hosts字典项，可以定义需要连接的机器。

```
env.hosts = ['host']

env.password = 'pwd'
```

多台机器的时候可以用

```
env.hosts = ['host1', 'host2']

env.passwords = {
    'host1': "pwd1",
    'host2': "pwd2",
}
```

部署实现

需求

- 有70台linux主机，都是基于CentOS和RedHat的操作系统，需要批量安装agent。
- 每台主机的安装指令都是一致的，即：HOSTNAME="主机名称" DD_API_KEY=86830b460bd64cce990def8a357c38fd TAGS="uyun:monitor" REPERTORY_URL="https://monitor.uyuntest.cn" bash -c "$(curl -L https://monitor.uyuntest.cn/downloads/agent/install_agent.sh)"
- 其中HOSTNAME，DD_API_KEY，TAGS，以及REPERTORY_URL，需要配置方自己填写的。HOSTNAME和TAGS每台机器是不同的，DD_API_KEY和REPERTORY_URL暂时每台机器都是相同的。
- 比如在主机A为70台主机安装agent，通过fabric方式，必须保证主机A与另外70台机器的ssh都能打通。
- 70台主机的 user需为root用户，或者在root群组当中，因为安装agent需要最高权限的。

配置和代码

配置：

配置以yaml文件的形式存储，安装时只需填写host.yaml文件里面的host_list即可。

descripton选项选填，hostname是安装后一边盘显示的hostname，os是仪表盘上面的tag显示。

```yaml
hosts_list:
  - desc: test fedora64
    ip: 192.168.195.132
    user: root
    password: vobile!@#
    port: 22
    hostname: fedora64
    os: fedora
  - desc: machine1
    ip: 10.10.120.132
    user: monitor
    password: monitor
    port: 22
    hostname: machine1
    os: Linux CentOS release 6.6 (Final)
。。。。。。。
。。。。。。。
```

代码：

1. 首先创建host列表，password字典，fabric会从中读取host以及user与password远程ssh到机器上；
2. 创建ip_hostname和ip_os两个字典，方便从配置文件中根据ip找到相应的hostname以及os；
3. 解析host.yaml拿到对应的值，组装成host列表，password，ip_hostname，ip_os字典。并且根据env设定角色和密码；
4. test_install_agent函数执行的即是安装agent，首先根据`/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|grep -v inet6|awk '{print $2}'|tr -d 'addr:'` 拿到机器的ip地址，这边可能会出现有外网ip和内网ip一起拿到的情况，因此 ip = ip[:15].strip("\r\n") 确保拿的是外网ip；
5. 根据ip从ip_hostname和ip_os两个字典两个字典中拿到hostname和os，组装成安装指令；
6. 执行安装指令。

   注1：为了使得env.passwords生效，host的格式必须是user@ip:port，端口号必须显示的写出来。

   注2：url和key需要根据用户自己的需求填写。

   注3：没有处理执行失败的情况，基本和手动安装是一样的，运行程序的时候，执行结果都会输出到控制台，如果执行失败，手动安装也需要去看看原因在哪。

code

```
# !/opt/python/bin/python
# -*- coding:utf-8 -*-
# notice: to enable env.passwords, host must be user@ip:port, port must be written explicitly

from fabric.api import *
from fabric.context_managers import *
import yaml

hosts = []
passwords = {}
ip_hostname = {}
ip_os = {}

host_list = yaml.load(file('host.yaml', 'r'))["hosts_list"]
for host_detail in host_list:
    host = "%s@%s:%s" %(host_detail["user"], host_detail["ip"], host_detail["port"])
    password = host_detail["password"]
    hosts.append(host)
    passwords[host] = password
    ip_hostname[host_detail["ip"]] = host_detail["hostname"]
    ip_os[host_detail["ip"]] = host_detail["os"]

env.passwords = passwords
env.roledefs = {'install_agent': hosts}

@roles('install_agent')
def test_install_agent():
    try:
        ip = run("/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|grep -v inet6|awk '{print $2}'|tr -d 'addr:'")
        ip = ip[:15].strip("\r\n")
        hostname = ip_hostname[ip]
        os = ip_os[ip]
        key = "86830b460bd64cce990def8a357c38fd"
        url = "https://monitor.uyuntest.cn"
        install_agent_cmd = 'HOSTNAME="%s" DD_API_KEY=%s TAGS="%s:monitor" REPERTORY_URL="%s" bash
-c "$(curl -L https://monitor.uyuntest.cn/downloads/agent/install_agent.sh)"' %(hostname, key, os, url)
        with settings(warn_only=True):
            run(install_agent_cmd)
    except Exception,e:
        print e
```

执行过程

1. 将所有文件上传到某文件夹
2. 解压绿色python到/opt: tar zxf fabric.tar.gz -C /opt/
3. cd 到代码所在的目录下，填写好host.yaml文件
4. host.yaml里面需填写一下hostname和os，hostname就是安装agent的hostname，os是TAGS这个参数，api_key暂时在代码中写死。
5. 执行脚本: /opt/fabric/bin/python /opt/fabric/bin/fab -f test_install.py test_install_agent

说明

1. 其中HOSTNAME，DD_API_KEY，TAGS(上面的例子os是需要显示在TAGS中的，如果需求有变，可能需要改下)，以及REPERTORY_URL，需要配置方自己填写的。HOSTNAME和TAGS每台机器是不同的，DD_API_KEY和REPERTORY_URL暂时每台机器都是相同的，写死在代码中。
2. 远程安装，通过fabric方式，必须保证主机A与另外70台机器的ssh都能打通。
3. 需要安装的linux主机的user需为root用户，或者在root群组当中，因为安装agent需要最高权限的。

附件

fabric.tar.gz：打包好的绿色版fabric环境，解压即可在CentOS7即以上使用

code.tar.gz：解压后配置host.yaml即可使用。

读书笔记

读书笔记 2016/08

《大数据时代》

进度：80%

读后感：

- 大数据带来的商业变革：

  大数据在商业方面的应用，我们只看到了非常小的一部分。

  a. 商家用折扣吸引消费者，消费者成为其会员，从订阅号中了解最新一季的服装产品。

  b.在超市使用支付宝，微信自动关注起了超市的公众号，会有更多商品信息退给给消费者。

  c.关注一个婚纱摄影的博主，微博推荐给我们蜜月旅行的博主，我们就能接收到蜜月相关的广告信息。

  生活上大数据用来挖掘客户各类需求

  a.挖掘客户贷款需求，定位消费信贷人群，关联信息对之进行评级，给出不同信用等级客户的授信额度。

  b.iphone可能就是随身的一个间谍，会手机手机位置，运行信息等等传回apple公司。

  c.亚马逊记录客户购买记录，个性化推荐，facebook跟踪用户兴趣，进行广告植入。
- 大数据带来的思维变革：

  a.大数据思维是一种意识，认为数据公开就能解决许多问题。

  b.数据量的增加造成结果不确定性，可能造成错误，然而数据量越大，错误的概率就越小，大数据用概率说话。

  c.跟上大数据的时代，我们需要变革三个思维方式：“不是随机样本，而是全体数据”，“不是精确性，而是混杂性”，“不是因果关系，而是相关关系”。
-
- 大数据带来的隐忧：

  a.信息的安全性：我们暴露在数据分析之下，亚马逊分析我们的购物习惯，facebook挖掘我们的人脉关系，google监视我们的上网习惯，如何保护我们自己的隐私？

  b.大数据给我们带来的便利的同时，随着而来的也有很多隐患。

  c.大数据会提前预测我们的行为，随着技术的进步，预测的准确率会越来越高，会不会出现一个情况，大数据能够提前预知你今天一天将要干什么，准确率90%以上，随着数据量的增大和技术的进步这是完全有可能的。

算法

事件台时间对齐

描述：

事件台对齐算法介绍

代码：

time

```
# -*- coding:utf-8 -*-

import random
import datetime as dt

class TimeAligner(object):
    def __init__(self, beg, end, g):
        self.beg = beg
        self.end = end
        self.g = g

    def do_align(self):
        delta = self.end-self.beg
        interval = (delta.seconds + delta.days * 86400 )/ self.g
        beg_seconds = self.beg.hour*3600+self.beg.minute*60+self.beg.second
        aligned_beg_seconds = beg_seconds if beg_seconds % interval == 0 else
((beg_seconds/interval)+1)*interval
        aligned_beg_date = self.beg - dt.timedelta(seconds=beg_seconds) +
dt.timedelta(seconds=aligned_beg_seconds)
        aligned_end_date = aligned_beg_date + dt.timedelta(seconds=delta.seconds + delta.days * 86400)
        print "after align: %s-->%s, %s->%s"%(self.beg.strftime('%Y-%m-%d
%H:%M:%S'),aligned_beg_date.strftime('%Y-%m-%d %H:%M:%S'), self.end.strftime('%Y-%m-%d %H:%M:%S')
,aligned_end_date.strftime('%Y-%m-%d %H:%M:%S'))
        return aligned_beg_date, aligned_end_date

class TestCase(object):
    def __init__(self, time_aligner):
        self.TimeAligner = time_aligner
        self.end = dt.datetime.now()
        self.beg = ""

    def init_time_aligner(self, beg, end, g):
        self.TimeAligner.beg = beg
        self.TimeAligner.end = end
        self.TimeAligner.g = g

    def test_30mins(self, granularity = 30):
        self.beg = self.end - dt.timedelta(seconds=1800)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

    def test_1hour(self, granularity = 30):
        self.beg = self.end - dt.timedelta(hours=1)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

    def test_6hours(self, granularity = 36):
        self.beg = self.end - dt.timedelta(hours=6)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

    def test_12hours(self, granularity = 24):
        self.beg = self.end - dt.timedelta(hours=12)
        self.init_time_aligner(self.beg, self.end, granularity)
```

```
        self.TimeAligner.do_align()

    def test_1day(self, granularity = 24):
        self.beg = self.end - dt.timedelta(days=1)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()
    def test_3days(self, granularity = 36):
        self.beg = self.end - dt.timedelta(days=3)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

    def test_7days(self, granularity = 28):
        self.beg = self.end - dt.timedelta(days=7)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

    def test_30days(self, granularity = 30):
        self.beg = self.end - dt.timedelta(days=30)
        self.init_time_aligner(self.beg, self.end, granularity)
        self.TimeAligner.do_align()

if __name__=="__main__":
    t = TimeAligner(None,None,None)
 testcase= TestCase(t)
    testcase.test_30mins()
    testcase.test_1hour()
    testcase.test_6hours()
    testcase.test_12hours()
    testcase.test_1day()
```

```
testcase.test_3days()
testcase.test_7days()
testcase.test_30days()
```

结果：

时间间隔	粒度	开始时间	结束时间	对齐后开始时间	对齐后结束时间	
30mins	30	2016-11-30 16:40:45	2016-11-30 17:10:45	2016-11-30 16:41:00	2016-11-30 17:11:00	
1hour	30	2016-11-30 16:10:45	2016-11-30 17:10:45	2016-11-30 16:12:00	2016-11-30 17:12:00	
6hours	36	2016-11-30 11:10:45	2016-11-30 17:10:45	2016-11-30 11:20:00	2016-11-30 17:20:00	
12hours	24	2016-11-30 05:10:45	2016-11-30 17:10:45	2016-11-30 05:30:00	2016-11-30 17:30:00	
1day	24	2016-11-29 17:10:45	2016-11-30 17:10:45	2016-11-29 18:00:00	2016-11-30 17:30:00	
3days	36	2016-11-27 17:10:45	2016-11-30 17:10:45	2016-11-27 18:00:00	2016-11-30 18:00:00	
7days	28	2016-11-23 17:10:45	2016-11-30 17:10:45	2016-11-23 18:00:00	2016-11-30 18:00:00	
30days	30	2016-10-31 17:10:45	2016-11-30 17:10:45	2016-11-01 00:00:00	2016-11-30 18:00:00	

杂项

agent执行shell扩展脚本插件

- 需求
- 设计
    - yaml配置文件
    - 用户自定义脚本
    - 说明
- 实现

需求

基于客户的需要，对我们之前给出的采集机器指标并且上报到monitor的服务端的方式： shell监控机器本地指标并上传

过于繁琐，用户需要写脚本将数据封装成json格式，并且post给server，因此提出了需要扩展shell脚本作为agent插件的方式：自定义指标在agent代理服务器上实施方案

用户的角度是简单，比如用户手中有若干个已经写好的监测机器指标的脚本，如果能让这些指标传送给monitor的server并且在仪表盘中可以展现，我们接触agent插件的方法来实现。

设计

yaml配置文件

yaml配置文件instances里面有三项需要填：

- metric_name: 用户自定义shell脚本的metrc
- value: 用户自定义脚本的执行语句
- tags: 用户自定义脚本的tags，yaml配置文件中式默认的tags，用户自定义脚本中也可以输出tags。

```yaml
init_config:
instances:
  - metric_name: system.service.state
    value: cat /opt/datadog-agent/ext/service_monitor.sh | wc -l
    tags:
     - service: datadog-agent
  - metric_name: app.log
    value: bash /opt/datadog-agent/ext/log_monitor.sh /var/log/ anaconda.log "INFO DEBUG"
    tags:
     - file: anaconda.log
```

用户自定义脚本

用户自定义了一个监控某个文件里面关键词keyword出现次数的脚本。

执行命令是：bash /opt/datadog-agent/ext/log_monitor.sh /var/log/ anaconda.log "INFO DEBUG"

- 脚本名称：log_monitor.sh
- 脚本路径：/opt/datadog-agent/ext/
- 文件路径： /var/log/
- 文件名：anaconda.log
- 关键词列表： "INFO DEBUG"

这个脚本就是检测 /var/log/anaconda.log这个日志文件下面INFO和DEBUG关键词出现的个数

```log
#!/bin/bash

for keyword in $3
do
    value=`cat $1$2 | grep $keyword | wc -l`
    echo $value
done
```

说明

1. yaml配置文件中，一个metric对应一个脚本，value是脚本执行语句；
2. 用户自定义脚本放在/opt/datadog-agent/ext/目录下，作为agent的扩展；
3. 用户自定义脚本的标准输出为 echo $value $tags_list，其他tags_list是可选项。

实现

作为agent的一个检测插件，代码在/opt/datadog-agent/agent/checks.d/shell.py中。

执行流程：

1. 拿到配置文件yaml里面的各项值，包括metric，tags和执行语句；
2. 执行对应的shell脚本，通过subprocess模块拿到标准输出；
3. 调用self.gauge讲组装成的数据结构上报出去。

shell.py

```
import subprocess
from checks import AgentCheck

class Shell(AgentCheck):
    def __init__(self, name, init_config, agentConfig, instances=None):
        AgentCheck.__init__(self, name, init_config, agentConfig, instances)

    def get_config(self,instance):
        metric_name = instance.get("metric_name",'')
        cmd = instance.get("value",'')
        tags = instance.get("tags",'')
        return metric_name, cmd, tags

    def check(self,instance):
        try:
            metric_name, cmd, yaml_tags = self.get_config(instance)
            shell_return = subprocess.Popen(str(cmd), shell = True, stdout = subprocess.PIPE)
            res = (line.strip("\n").split(" ") for line in shell_return.stdout.readlines())
            for r in res:
                value = r[0]
                yaml_tags = str(yaml_tags).replace("}","").replace("{","")
                tags = (yaml_tags if len(r) == 1 else [res[1]])
                self.gauge(metric_name, float(value), tags=tags)
        except IOError,e:
            self.log.error("IO error on %s" %e)
        except Exception,e:
            self.log.error("uncaught typeerror on %s" %e)
```

Open-falcon agent

- 简介
    - 官方性能
    - 数据流
    - 数据结构
    - 重要组件
        - agent
        - transfer
        - judge
- Open-falcon agent
    - 安装部署（官方）
    - 采集
    - 自动更新
- 与datadog的对比
    - 支持版本
    - 数据转发
    - 插件扩展
    - 进程管理
    - 自动升级
- 参考文献

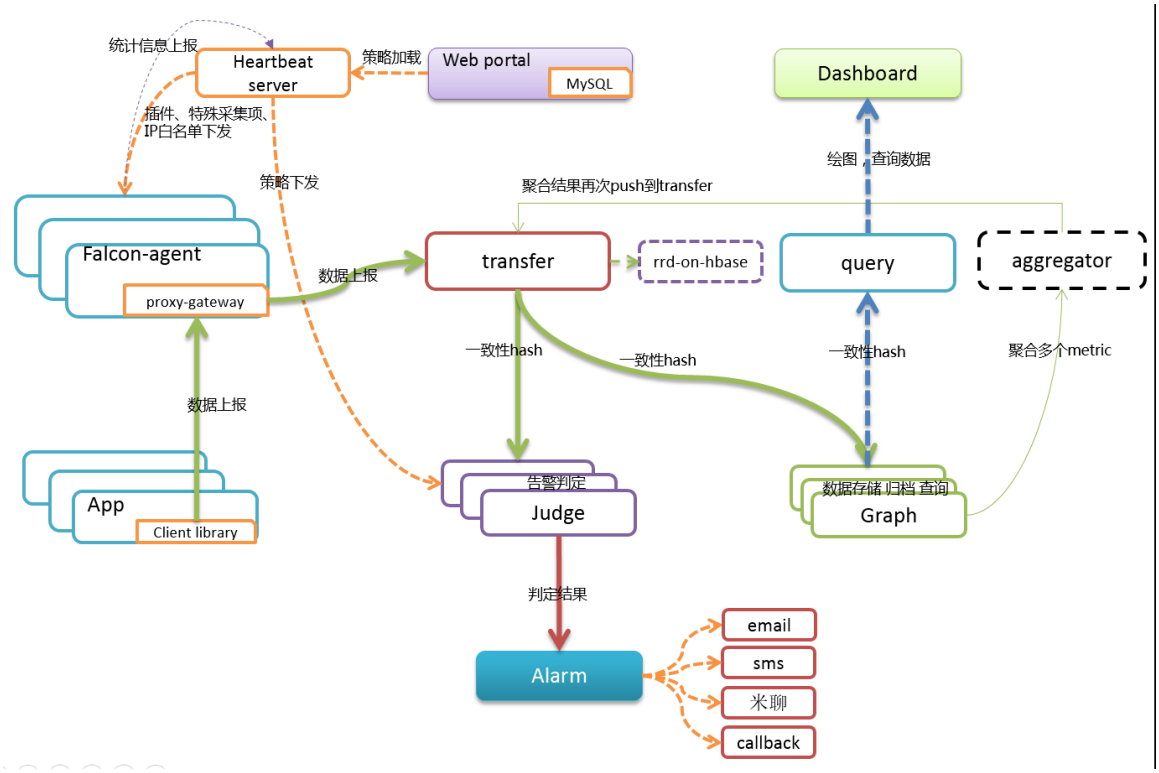
简介

监控系统是运维的核心环节，事情发现并报警和事后定位处理是关键的两个环节。open-falcon是小米公司开发的一套企业级的高性能，开放性的监控系统。

官方性能

下面是官方给出的open-falcon系统的性能：

- 强大灵活的数据采集：自动发现，支持falcon-agent、snmp、支持用户主动push、用户自定义插件支持、opentsdb data model like（timestamp、endpoint、metric、key-value tags）
- 水平扩展能力：支持每个周期上亿次的数据采集、告警判定、历史数据存储和查询
- 高效率的告警策略管理：高效的portal、支持策略模板、模板继承和覆盖、多种告警方式、支持callback调用
- 人性化的告警设置：最大告警次数、告警级别、告警恢复通知、告警暂停、不同时段不同阈值、支持维护周期
- 高效率的graph组件：单机支撑200万metric的上报、归档、存储（周期为1分钟）
- 高效的历史数据query组件：采用rrdtool的数据归档策略，秒级返回上百个metric一年的历史数据
- dashboard：多维度的数据展示，用户自定义Screen
- 高可用：整个系统无核心单点，易运维，易部署，可水平扩展
- 开发语言： 整个系统的后端，全部golang编写，portal和dashboard使用python编写。



数据流

根据上图看到，绿色实线代表整个open-falcon主要工作过程的数据流向。

APP：这边的APP应该是用户自定义的插件或者用户主动push的数据给agent；

agent：agent是golang开发的deamon程序，采集机器的各项指标（类比我们所用的datadog-agent），上报给transfer；

transfer：transfer是转发数据的组件（类比我们所用的forwarder），它接收agent上报的数据，按照哈希规则进行数据分片、讲处理完的数据转发给judge和graph等组件；

judge：judge接收transfer发送来的数据，judge判断数据是否异常，如果异常，则将相应的异常事件写入redis，由alarm组件读取redis中的数据处理；judge只负责判断，不负责处理；

Graph：存储绘图数据的组件，Graph接收transfer发送过来的数据，同时处理query组件的查询请求、返回绘图数据；对数据进行存储，归档和查询；

query：。query组件接收查询请求，根据一致性哈希算法去相应的graph实例查询不同metric的数据，然后汇总拿到的数据，最后统一返回给用户；

dashboard：报表展现组件（类比仪表盘），用户查询数据。

数据结构

采用和opentsdb相同的数据格式

data

```
{
    metric: load.1min,
    endpoint: open-falcon-host,
    tags: srv=falcon,idc=aws-sgp,group=az1,
    value: 1.5,
    timestamp: `date +%s`,
    counterType: GAUGE,
    step: 60
}
```

重要组件

agent

golang开发，采集基本指标包括cpu，磁盘，IO，内存，进程指标等等。当然不仅仅这些，官方提供了200多个基本指标。

机器安装了open-falcon agent之后就开始自动采集数据，主动上报，server端不需要做配置。基本是和dd-agent相同的。

默认采集间隔为60秒，异于dd-agent默认的15秒。

transfer

接收agent上报过来的数据，根据一致性hash算法对数据进行分片，主要是监控系统数据量比较大，一台机器显然是搞不定的，所以必须进行数据分片，judge或者graph只需要处理一个分片的数据就可以了。

transfer的数据来源：

- agent采集到上报上来的数据；
- agent执行用户自定义插件上报上来的数据；
- 线上的业务系统中每个RPC接口的qps、latency都会主动采集并上报的数据。

transfer转发数据到三个组件：

- judge
- graph
- opentsDB

judge

用户在web portal组件上配置好相应的报警策略和级别，judge定期和heartbeat server通信获取最新的报警策略。

transfer转发到judge的每条数据，都会触发相关策略的判定，来决定是否满足报警条件，如果满足条件，则会发送给alarm，alarm各种形式进行报警。

Open-falcon agent

安装部署（官方）

1. 安装agent之前需要安装好open-falcon相关的依赖：redis，mysql；
2. 初始化mysql表结构；
3. 安装go语言环境；

4. 下拉代码；
5. 安装agent。

```
install
```

```
yum install -y redis
yum install -y mysql-server

export HOME=/home/work
export WORKSPACE=$HOME/open-falcon
mkdir -p $WORKSPACE
cd $WORKSPACE
git clone https://github.com/open-falcon/scripts.git
cd ./scripts/
mysql -h localhost -u root -p < db_schema/graph-db-schema.sql
mysql -h localhost -u root -p < db_schema/dashboard-db-schema.sql
mysql -h localhost -u root -p < db_schema/portal-db-schema.sql
mysql -h localhost -u root -p < db_schema/links-db-schema.sql
mysql -h localhost -u root -p < db_schema/uic-db-schema.sql

cd ~
wget http://dinp.qiniudn.com/go1.4.1.linux-amd64.tar.gz
tar zxf go1.4.1.linux-amd64.tar.gz
mkdir -p workspace/src
echo "" >> .bashrc
echo 'export GOROOT=$HOME/go' >> .bashrc
echo 'export GOPATH=$HOME/workspace' >> .bashrc
echo 'export PATH=$GOROOT/bin:$GOPATH/bin:$PATH' >> .bashrc
echo "" >> .bashrc
source .bashrc

cd $GOPATH/src
mkdir github.com
cd github.com
git clone --recursive https://github.com/open-falcon/of-release.git

cd $GOPATH/src/github.com/open-falcon/agent
go get ./...
./control build
./control pack
```

进程管理

```
pro
```

```
./control start 启动进程
./control stop 停止进程
./control restart 重启进程
./control status 查看进程状态
./control tail 用tail -f的方式查看var/app.log
```

采集

agent只要部署到机器上，并且配置好了heartbeat和transfer就自动采集数据了。

采集：agent采集机器监控指标，比如cpu.idle、load.1min、disk.io.util等等，每隔60秒发送给Transfer。

进程管理：采用control脚本进行控制agent进程

用户扩展插件：

- 编写脚本采集到数据之后打印到stdout，agent会截获并push给server。数据格式是json；
- 脚本编写好了放到git仓库中，把git仓库地址配置上，enabled设置为true就是该插件可运行状态，http端口1988用户agent挨个curl一下http://ip:1988/plugin/update ，agent主动git pull这个插件并且运行。
- 对插件扩展脚本没有要求，只需要在机器上有运行环境。

自动更新

机器数量少的时候我们可以采用ssh，fabric等方式对agent进行升级或者回滚，但如果机器数量十分庞大，这一方式就不可行了。因此，open-falcon开发了一个工具名叫agent-updater的工具。

agent-updater组件包含：

- ops-updater：默认在机器安装agent的时候也会安装ops-updater，默认从不升级，它是管理agent升级的agent。
- ops-meta：服务端，和ops-updater定期通信，告知ops-updater是否该升级，回滚，改动agent。

升级方式：

1. ops-meta作为服务端保存了每个ops-updater的配置信息；
2. ops-updater与ops-meta的通信周期默认是5min，ops-updater会汇报自己管理的各个agent的状态、版本号，ops-meta告知updater是否需要升级；
3. 升级的时候ops-updater只需要去ops-meta指定的位置上拉下来安装包或者升级包即可。

与datadog的对比

支持版本

open-falcon agent：

- 只支持64位的linux，mac和windows均不支持

datadog-agent：

- 支持64位，32位 linux，mac，windows

数据转发

open-falcon agent：

- open-falcon 的transfer执行了转发功能，它不隶属于agent，作为独立的模块，可以集群部署，小米就部署了20台机器作为transfer；
- transfer与agent是一对多的关系，一例transfer对应多例agent；
- transfer对agent上报上来有着初步的处理，剔除格式不合法的数据；
- transfer对上报上来的数据使用一致性hash算法进行分片，转发给不同的graph和judge。

datadog-agent：

- dd是用forwarder进程进行数据转发，forwarder进程属于agent，是agent的一个进程；
- forwarder与collector是一对一的关系；
- 并且功能比较单一，只有转发，缓存等简单的功能。

异同：

- open-falcon讲transfer转发功能从agent独立出来，dd的forwarder集成到agent内部。

插件扩展

open-falcon agent：

- 编写脚本采集到数据之后打印到stdout，agent会截获并push给server。数据格式是json；
- 脚本编写好了放到git仓库中，把git仓库地址配置上，enabled设置为true就是该插件可运行状态，http端口1988用户agent挨个curl一下http://ip:1988/plugin/update ，agent主动git pull这个插件并且运行。
- 对插件扩展脚本没有要求，只需要在机器上有运行环境。

datadog-agent：

- dd的插件扩展非常简单，只需要用户在/opt/datadog-agent/agent/checks.d/中添加插件监测代码(继承AgentCheck类)，并且/etc/dd-agent/conf.d/中写完配置即可采集；
- dd的collector会自动扫描/etc/dd-agent/conf.d/目录下面可用的插件配置文件，并且去执行/opt/datadog-agent/agent/checks.d/里

面相应的配置脚本，然后上报；
- 扩展采集脚本必须由python编写。

异同：

- 扩展都比较容易，dd扩展限定了扩展插件脚本必须是python，而open-falcon没有；
- dd更加具有管理性，脚本和配置均在自己工程下面，open-falcon需要拉去git仓库，万一git挂了或者网络通信不好怎么办？

进程管理

open-falcon agent：

- 由于open-falcon和dd不一样的是，它的agent只负责采集指标并且上报，因此没有像dd那样的父进程作为监控；
- open-falcon agent是由golang写的，control脚本负责agent的start|stop|restart|status等等，类比于dd-agent的datadog-agent脚本；

datadog-agent：

- dd进程管理作为独立父进程，linux下面有supervisord控制agent的collector，forwarder等进程；windows下面由父进程watchdog控制collector，forwarder等子进程；

自动升级

open-falcon agent：

- 根据上面的介绍，open-falcon自动更新，采取的是在机器上部署一个独立的ops-updater；
- ops-updater可以当做一个独立的进程，它不会被更新，只有一个作用就是和ops-meta服务端通信，来更新机器的agent。

datadog-agent：

- 将自动更新集成进agent，作为独立进程，定期与服务端通信，决定agent是否更新。

异同：

- 两者在于一个集成进agent与否的问题。

参考文献

文档：http://book.open-falcon.org/

git地址：https://github.com/open-falcon

博客：https://segmentfault.com/a/1190000006047609

http://ulricqin.com/project/ops-updater/

http://www.th7.cn/system/lin/201603/155278.shtml

http://www.111cn.net/sys/linux/88539.htm

shell监控机器本地指标并上传

- 1.需求
    - 监控任务：
    - 部署方式：
- 2.实现
    - service监控
    - 日志监控
    - 文件监控
    - crontab监控
    - post数据
- 3.测试

1.需求

监控任务：

- service监控：获取机器所有service name，以及service的运行状态。
- 日志监控：根据脚本传入的关键字，如：ERROR，INFO等，统计某一日志中出错日志行数。
- 文件监控：根据脚本传入的文件路径，统计文件夹下文件的个数。

部署方式：

- 在某台机器上部署监控脚本，以crontab的方式定时运行，监控，获取指标并向server上报。
- 通过curl向服务器post数据。

2.实现

service监控

执行过程：

1. 无系统命令行参数，根据systemctl list-units --type=service命令，并且根据状态，得到所有running状态的service，返回码为1；
2. cd 进/usr/lib/systemd/system，得到所有service；
3. 遍历usr/lib/systemd/system所有service，判断是否在第一步中，如果不在，则service是非running状态，返回码为0；
4. 将service_name和service_code封装成datapoints，curl发送给服务器。

代码：

monitor_service

```
function monitor_service_redhat(){
    echo "-------monitor service redhat-------"
    declare -a datapoints_array
    declare -a running_service
    sname_list=`systemctl list-units --type=service | awk '{print $1}'`
    sname_list_all=`ls -lh  /usr/lib/systemd/system | grep ^-r | awk '{print $9}'`
    for sname in $sname_list
    do
        sstatus=`systemctl list-units --type=service | grep $sname | awk '{print $4}'`
        if [ "$sstatus"x = 'running'x ]; then
            generate_datapoint $HOST_ID $HOSTNAME 'system.service.state' 1
'["host:'$HOSTNAME'","service:'$sname'"]' 'gauge'
            datapoints_array=("${datapoints_array[@]}" $datapoint)
            running_service=("${running_service[@]}" $sname)
        fi
    done
    for sname in $sname_list_all
    do
        if echo "${running_service[@]}" | grep -w $sname &>/dev/null; then
            continue
        else
            generate_datapoint $HOST_ID $HOSTNAME 'system.service.state' 0
'["host:'$HOSTNAME'","service:'$sname'"]' 'gauge'
            datapoints_array=("${datapoints_array[@]}" $datapoint)
        fi
    done
    generate_json "${datapoints_array[*]}"
    post_json "$datapoints"
}
```

日志监控

运行方式：

execute method

```
bash monitor_log.sh  /var/log/anaconda/ anaconda.log "INFO DEBUG"
 #后面第一个参数代表文件路径，第二个参数是文件名，第三个参数是关键字列表，注意要加上""
```

执行过程：

1. 命令行第三个参数是关键字列表，根据传入的关键字列表，做一个循环；
2. grep并且wc -l一下关键字出现的行数，封装成datapoints，tag是host和file
3. curl发送给服务器

代码：

monitor_log

```
declare -a datapoints_array
for keyword in $3
do
    num=`cat $1$2 | grep $keyword | wc -l`
    generate_datapoint $HOST_ID $HOSTNAME 'app.log.'$keyword $num '["host:'$HOSTNAME'","file:$1$2"]'
'gauge'
    datapoints_array=("${datapoints_array[@]}" $datapoint)
done
generate_json "${datapoints_array[*]}"
post_json "$datapoints"
echo $datapoints
```

文件监控

运行方式：

execute method

```
bash monitor_file.sh /var/log/  all
 #后面第一个参数代表路径， 第二个参数可选，如果不加第二个参数，默认统计文件夹下面
```

执行过程：

1. 第一个参数是路径，首先进入此目录；
2. 如果传递的第二个参数是空，默认统计当前文件下面的文件和目录数目，如果第二个参数是all，则统计当前文件夹下面以及子文件下面所有的文件和目录数；
3. 得到的file_num作为参数封装进datapoint字符串里，tag是host和file；
4. curl 发送给服务器。

代码：

monitor_file

```
declare -a datapoints_array
cd $1
if [ "$2"x = ""x ];then
    file_num=`ls -lh |wc -l`
elif [ "$2"x = "all"x ];then
    file_num=`ls -lR|grep "^-"|wc -l`
fi
generate_datapoint $HOST_ID $HOSTNAME "system.file.number" $file_num '["host:'$HOSTNAME'","file:'$1'"]'
'gauge'
datapoints_array[0]=$datapoint
generate_json "${datapoints_array[*]}"
post_json "$datapoints"
```

crontab监控

执行过程:

1. 根据crontab -l 循环拿到里面每一个计划任务；
2. cat /var/log/cron 根据cron的日志获取该计划任务跑了多少次；
3. curl发送给服务器。

代码：

monitor_crontab

```
line_num=`crontab -l | wc -l`
for((i=1;i<=$line_num;i++));
do
    CMD=`crontab -l | awk '{print $7}' | sed -n "$i"p`
    CMD1=`echo $CMD`
    count=`eval "cat /var/log/cron | grep '$CMD1' | wc -l"`
    echo $CMD1 $count

    generate_datapoint $HOST_ID $HOSTNAME 'system.crontab.count' $count
'["host:'$HOSTNAME'","task_name:'$CMD1'"]' 'gauge'
    datapoints_array=("${datapoints_array[@]}" $datapoint)
done
```

post数据

执行过程：

1. 单个指标通过generate_datapoint函数封装成datapoint字符串；
2. 声明一个datapoints_array数据，将第一步封装成的datapoint字符串添加到数组中去，数组的元素就是一个个上报的指标；
3. 讲datapoints_array作为参数传递给generate_json函数，组成可以post的json字符串；
4. curl json字符串至服务器。

代码：

post_data

```
#!/bin/bash

HOST_ID='c0ba787fc33d517d8cf51aa5000b8c59'
HOSTNAME='localhost.localdomain'
server_base_url="http://10.1.53.101/monitor"
apikey="e10adc3949ba59abbe56e057f2gg88dd"

function generate_datapoint(){
    current=`date "+%Y-%m-%d %H:%M:%S"`
    timeStamp=`date -d "$current" +%s`

datapoint="{\"hostId\":\"$1\",\"host\":\"$2\",\"metric\":\"$3\",\"timestamp\":\"$timeStamp\",\"value\":$4,\"tags\":
$5,\"type\":\"$6\"}"
}

function generate_json(){
    datapoints="["
    local datapoints_array=($1)
    datapoints_count=${#datapoints_array[*]}
    for ((i=0; i<$datapoints_count; i++));
    do
        datapoint=${datapoints_array[$i]}
        datapoints="$datapoints$datapoint,"
    done
    result_count=`echo ${#datapoints}`
    temp=`expr $result_count - 1`
    datapoints=`echo ${datapoints%,*}`
    datapoints="$datapoints]"
}

function post_json(){
    curl -X POST "$server_base_url/openapi/v2/datapoints?api_key=$apikey" -H Content-type:application/json
--data $1
}

echo "-------"monitor log-------""
declare -a datapoints_array
generate_json "${datapoints_array[*]}"
post_json "$datapoints"
echo $datapoints
```

3.测试

建了三个仪表盘测了一下，好像没出现大问题。。