Master设计

基本功能

TCP Server
绑定端口，启动监听，等待Agent连接。

信息存储
存储Agent列表
存储用户提交的Task列表。用户通过WEB提交的任务信息存储下来。

接收注册
将注册信息写入Agent列表

接收心跳信息
接收Agent发来的心跳信息

派发任务
将用户提交的任务分配到Agent端。

代码实现

构建master模块。
master.config模块

```
MASTER_URL = "tcp://0.0.0.0:9000"
```

master.cm模块，负责Tcp连接

```
from utils import getlogger

logger = getlogger(__name__, 'o:/cm.log')

class ConnectionManager:
    def handle(self, msg):
        logger.info(type(msg))
        return "{}".format(msg)

    sendmsg = handle
```

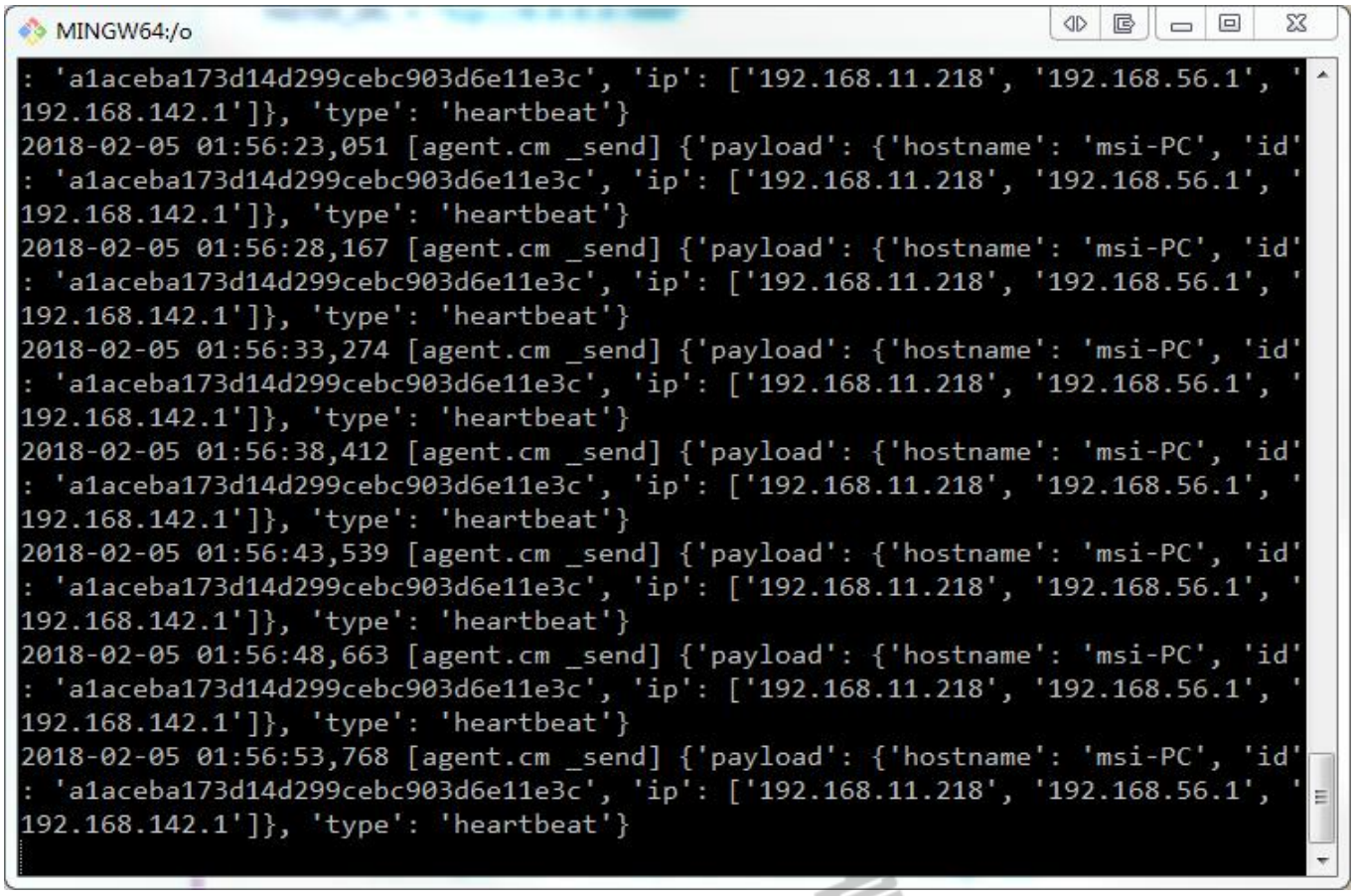
定义Master类，负责启动TCP Server。

```
import zerorpc
from .cm import ConnectionManager
from .config import MASTER_URL

class Master:
    def __init__(self):
        self.tcpserver = zerorpc.Server(ConnectionManager())
        self.tcpserver.bind(MASTER_URL)

    def start(self):
        self.tcpserver.run() # 循环阻塞的

    def shutdown(self):
        self.tcpserver.close()
```

项目根目录下构建一个appserver.py用于测试

```
from master import Master

if __name__ == '__main__':
    master = Master()
    try:
        master.start()
    except KeyboardInterrupt:
        master.shutdown()
```

查看日志，不断有数据写入，测试成功



代码实现上述的注册、心跳部分
经过观察发现，目前注册和心跳除了类型不同，其它都一样。可以这样认为第一次心跳成功，就是注册。

Master的数据设计

Master端核心需要存储2种数据：Agent端数据、用户提交的任务Task。
构造一个数据结构，存储这些信息。

```
{
  "agents":{
    "agent_id":{
      "heartbeat":"timestamp",
      "busy":false,
      "info":{
        "hostname":"",
        "ip":[]
      }
    }
  }
}
```

```
{
  "tasks": {
    "task_id": {
      "script": "base64encode",
      "targets": {
        "agent_id": {
          "state": "WAITING",
          "output": ""
        }
      },
      "state": "WAITING"
    }
  }
}
```

上面2个数据结构：

- agents里面记录了所有注册的agent
  - agent_id，字典的key，每一个agent有不同的uuid，所以这个字典就是item=uuid:{}
    - connection 给agent记录master和agent建立的连接，备用?TODO?还要不要了？
    - heartbeat 由于设计中并没有让agent端发送心跳时间，所以就在master端记录收到的时间
    - busy 如果agent上有任务在跑。就会置这个值为True
    - info 记录agent上发过来的hostname和ip列表
- tasks 记录所有任务及其target（agent）的状态
  - task_id，字典的key对应一个个task，item也是taskid:{}的结构
    - task 任务，task.json的payload信息
    - targets 目标，用来执行agent的节点，记录agent上的state和输出outpout
      - state 状态，单个agent上的执行状态
    - state 这一个task的状态，整个任务的状态，比如统计达到了agent失败上限了，这个task的state就置为失败

状态常量

'WAITING' 'RUNNING' 'SUCCEED' 'FAILED'

构建master/state.py

```
WAITING = 'WAITING'
RUNNING = 'RUNNING'
SUCCEED = 'SUCCEED'
FAILED = 'FAILED'
```

agent信息存储

构建Storage类

master/storage.py

```
import datetime

class Storage:
    def __init__(self):
        self.agents = {}
        self.tasks = {}

    def reg_hb(self, agent_id, info):
        self.agents[agent_id] = {
            'heartbeat': datetime.datetime.now(),
            'info':info,
            'busy':self.agents.get(agent_id, {}).get('busy', False)
        }
        # busy读不到置False，读的到不变。后期实现
```

为什么heartbeat不用时间戳？
可以用，但是这里时间取的是Master的时间，而且不用Master、Agent间来回传输，是Master内部数据结构Storage的数据，所以没有用时间戳。如果使用时间类型，且使用json序列化，就要注意数据类型了。zerorpc内部使用了messagepack，支持日期类型。

master/cm.py

```
from utils import getlogger
from .storage import Storage

logger = getlogger(__name__, 'o:/cm.log')

class ConnectionManager:
    def __init__(self):
        self.store = Storage()

    def handle(self, msg):
        logger.info(type(msg))
        try:
            if msg['type'] in {"heartbeat",'reg'}:
```

```
                payload = msg['payload']
                info = {'hostname':payload['hostname'], 'ip':payload['ip']}
                self.store.reg_hb(payload['id'], info)

                logger.info("{}".format(self.store.agents))
                return "ack {}".format(msg)
        except Exception as e:
            logger.error("".format(e))
            return "Bad Request."

    sendmsg = handle # zerorpc接口
```

task任务处理

用户通过WEB（HTTP）提交新的任务，任务json信息有：
1、任务脚本script，base64编码
2、超时时间timeout
3、并行度parallel
4、失败率fail_rate
5、失败次数fail_count
6、targets 是跑任务的Agent的agent_id列表，这个目前也是在用户端选择好。比如用户需要在主机名叫做WEBServer-xxx的几台主机上运行脚本。为了用户方便，可以类似ansible的分组。

在Master端收到任务信息后，需要添加2个信息：
task_id 是Master端新建任务是生成的uuid。
state 默认WAITING。

在WEB Server中最后将用户端发来的数据组成下面的字典

```
{
    'task_id': t.id,
    'script': t.script,
    'timeout': t.timeout,
    'parallel': t.parallel,
    'fail_rate': t.fail_rate,
    'fail_count': t.fail_count,
    'state': t.state,
    'targets': t.targets
}
```

将任务数据封装成任务对象，构建Task类

master/task.py

```
import uuid
from .state import *

class Task:
    def __init__(self, task_id, script, targets, timeout=0, parallel=1,
                 fail_rate=0, fail_count=-1):
        self.id = task_id
        self.script = script
        self.timeout = timeout
        self.parallel = parallel
        self.fail_rate = fail_rate
        self.fail_count = fail_count
        self.state = WAITING
        self.targets = {agent_id:{'state':WAITING, 'output':""} for agent_id in tar
gets}
        self.target_count = len(self.targets)
```

WEB Server调用Storage中方法，将任务数据存入

```
import datetime
from .task import Task

class Storage:
    def __init__(self):
        self.agents = {}
        self.tasks = {}

    def reg_hb(self, agent_id, info):
        self.agents[agent_id] = {
            'heartbeat': datetime.datetime.now(),
            'info':info,
            'busy':self.agents.get(agent_id, {}).get('busy', False)
        }
        # busy读不到置False，读的到不变

    def add_task(self, task:dict): # 从WEB Server来
        t = Task(**task)
```

```
        self.tasks[t.id] = t
        return t.id
```

任务分派*

分派方式
任务在Storage中存储，一旦有了任务，需要将任务分派到指定节点执行，交给这些节点上的Agent。
不过，目前使用zerorpc，Master是被动的接收Agent的数据并响应的。
所以，可以考虑一种Agent主动拉取机制，就是提供一个接口，让Agent访问。如果Agent处于空闲，就主动拉取任务，有任务就领走。
当Agent少的时候，**Master推送任务到Agent端**，或者**Agent端主动拉取任务**都是可以的。但是如果考虑Agent多的时候，或许Agent拉模式是更好的选择。

本次采用Agent拉取模式实现，所以Master就不需要设计调度器了。

master/storage.py

```
import datetime
from .task import Task
from .state import *

class Storage:
    def __init__(self):
        self.agents = {}
        self.tasks = {}

    def reg_hb(self, agent_id, info):
        self.agents[agent_id] = {
            'heartbeat': datetime.datetime.now(),
            'info':info,
            'busy':self.agents.get(agent_id, {}).get('busy', False)
        }
        # busy读不到置False，读的到不变

    def add_task(self, task:dict): # 从WEB Server来
        t = Task(**task)
        self.tasks[t.id] = t
        return t.id
```

```
    def iter_tasks(self, states={WAITING, RUNNING}):
        yield from (task for task in self.tasks.values() if task.state in states)

    def get_task_by_agentid(self, agent_id ,state=WAITING):
        for task in self.iter_tasks():
            if agent_id in task.targets.keys():
                t = task.targets.get(agent_id)
                if t.get('state') == state:
                    return task, t # 为节点找到一个任务就返回
```

master/cm.py

```
from utils import getlogger
from .storage import Storage
from .state import *

logger = getlogger(__name__, 'o:/cm.log')

class ConnectionManager:
    def __init__(self):
        self.store = Storage()

    def handle(self, msg):
        logger.info(type(msg))
        try:
            if msg['type'] in {"heartbeat",'reg'}:
                payload = msg['payload']
                info = {'hostname':payload['hostname'], 'ip':payload['ip']}
                self.store.reg_hb(payload['id'], info)

                logger.info("{}".format(self.store.agents))
                return "ack {}".format(msg)
        except Exception as e:
            logger.error("".format(e))
            return "Bad Request."

    sendmsg = handle # zerorpc接口

    def take_task(self, agent_id): # 闲了Agent主动来领取任务
        # 有任务则返回任务信息，否则返回None
```

```
        # {'id':task.id,'script':task.script,'timeout':task.timeout}
        # 遍历状态是RUNNING或WAITING的任务，其中targets中agent_id是自己的且状态是WAI
TING的
        info = self.store.get_task_by_agentid(agent_id)
        # 找到了，就将WAITING任务转换为RUNNING，将自己的状态置为RUNNING
        if info:
            task, target = info
            task.state = RUNNING
            target['state'] = RUNNING
            return {
                'id':task.id,
                'script':task.script,
                'timeout':task.timeout
            }
```

Agent领取任务

Agent领取任务，就是去Client去调用take_task接口。

执行流程
放在心跳循环中，不过要增加状态，这个状态直接使用master中定义的状态文件。如果在循环中，Agent的当前状态是WAITING，就可以去Master获取任务。如果没有任务，就隔一段时间尝试再次取任务。如果获取到任务，就可以将状态置为RUNNING并开启新的进程执行脚本，直到脚本执行完，把状态码和输出通过封装成result消息返回给Master。

agent/cm.py

```
import zerorpc
import threading
from .msg import Message
from utils import getlogger
from .state import *
from .executor import Executor

logger = getlogger(__name__, 'o:/cm.log')

class ConnectionManager:
    def __init__(self, master_url, message:Message):
        self.master_url = master_url
        self.message = message # 对象
```

```
        self.client = zerorpc.Client()
        self.event = threading.Event()
        self.state = WAITING
        self.executor = Executor()

    def start(self, hbinterval=5):
        try:
            self.event.clear()
            # 连接
            self.client.connect(self.master_url)

            # 注册
            self._send(self.message.reg())
            # 心跳、领任务循环
            while not self.event.wait(hbinterval):
                self._send(self.message.heartbeat())
                if self.state == WAITING:
                    self._get_task(self.message.id)
        except Exception as e:
            logger.error('Failed to connect to master. Error:{}'.format(e))
            raise e

    def shutdown(self):
        self.event.set()
        self.client.close()

    def _send(self, msg):
        try:
            ack = self.client.sendmsg(msg)
            logger.info(ack)
        except Exception as e:
            logger.error('Failed to connect to master. Error:{}'.format(e))
            self.event.set()

    def _get_task(self, agent_id):
        task = self.client.take_task(self.message.id)
        if task: # 拿回了任务
            logger.info("{}".format(task))
            self.state = RUNNING
            # 调用执行器，开启子进程
```

```
            # {'id':task.id,'script':task.script,'timeout':task.timeout}
            script = task['script'] # 注意为了简单测试，没有做base64编码，后期加上
            code, output = self.executor.run(script)

            self._send(self.message.result(task['id'], code, output))
            self.state = WAITING
```

Executor类

agent/executor.py

```
from subprocess import Popen, PIPE
from utils import getlogger

logger = getlogger(__name__, 'o:/exec.log')

class Executor:
    def run(self, script, timeout=None):
        #proc = subprocess.Popen('echo "hello magedu"', shell=True, stdout=subproce
ss.PIPE)
        logger.info('Agent start exec~~~~~~~~~~~~~~~~~~~~')
        proc = Popen(script, shell=True, stdout=PIPE)
        code = proc.wait() # 阻塞返回状态码
        output = proc.stdout.read() # 标准输出
        logger.info(code)
        logger.info(output)
        return code, output
```

agent/msg.py

```
import netifaces
import ipaddress
import os
import uuid
import socket

class Message:
    def __init__(self, myidpath):
        # 从文件中读取主机的UUID
        if os.path.exists(myidpath):
            with open(myidpath) as f:
```

```
            self.id = f.readline().strip()
    else:
        self.id = uuid.uuid4().hex
        with open(myidpath, 'w') as f:
            f.write(self.id)

def _get_addresses(self):
    """获取主机上所有接口可用的IPv4地址"""
    addresses = []

    for interface in netifaces.interfaces():
        if 2 in netifaces.ifaddresses(interface).keys():
            for ip in netifaces.ifaddresses(interface)[2]:
                # ipaddress地址验证
                #print(ip)
                ip = ipaddress.ip_address(ip['addr'])
                if ip.version != 4: # 版本
                    continue
                if ip.is_link_local: # 169.254地址
                    continue
                if ip.is_loopback: # 回环
                    continue
                if ip.is_multicast: # 多播
                    continue
                if ip.is_reserved: # 保留
                    continue

                addresses.append(str(ip))
    return addresses

def reg(self):
    """生成注册消息"""
    return {
        'type':'reg',
        'payload':{
            'id':self.id,
            'hostname':socket.gethostname(),
            'ip':self._get_addresses()
        }
    }
```

```
    def heartbeat(self):
        """生成心跳消息"""
        return {
            'type':'heartbeat',
            'payload':{
                'id':self.id,
                'hostname':socket.gethostname(),
                'ip':self._get_addresses()
            }
        }

    def result(self, task_id, code, output):
        """返回执行结果"""
        return {
            'type': 'result',
            'payload': {
                'id':task_id,
                'agent_id':self.id,
                'code':code,
                'output':output
            }
        }
```


Master接收Result消息

master/cm.py

```
class ConnectionManager:
    def __init__(self):
        self.store = Storage()

    def handle(self, msg):
        logger.info(type(msg))
        try:
            if msg['type'] in {"heartbeat",'reg'}:
                payload = msg['payload']
                info = {'hostname':payload['hostname'], 'ip':payload['ip']}
                self.store.reg_hb(payload['id'], info)
```

```
                logger.info("{}".format(self.store.agents))
                return "ack {}".format(msg)

            elif msg['type'] == 'result': # 处理result消息
                payload = msg['payload']
                agent_id = payload['agent_id']
                task_id = payload['id']
                state = SUCCEED if payload['code'] == 0 else FAILED
                output = payload['output']

                task = self.store.get_task_by_id(task_id)
                t = task.targets[agent_id]
                t.state = state
                t.output = output
                return 'ack result'

        except Exception as e:
            logger.error("".format(e))
            return "Bad Request."
```

master/storage.py

```
import datetime
from .task import Task
from .state import *

class Storage:
    def __init__(self):
        self.agents = {}
        self.tasks = {}

    def reg_hb(self, agent_id, info):
        self.agents[agent_id] = {
            'heartbeat': datetime.datetime.now(),
            'info':info,
            'busy':self.agents.get(agent_id, {}).get('busy', False)
        }
        # busy读不到置False，读的到不变

    def add_task(self, task:dict): # 从WEB Server来
```

```
        t = Task(**task)
        self.tasks[t.id] = t
        return t.id

    def iter_tasks(self, states={WAITING, RUNNING}):
        yield from (task for task in self.tasks.values() if task.state in states)

    def get_task_by_agentid(self, agent_id ,state=WAITING):
        for task in self.iter_tasks():
            if agent_id in task.targets.keys():
                t = task.targets.get(agent_id)
                if t.get('state') == state:
                    return task, t # 为节点找到一个任务就返回


    def get_task_by_id(self, task_id) -> Task:
        return self.tasks.get(task_id)
```