[Edit](http://maxiang.info/#/?provider=evernote&guid=01e61863-812a-4c5a-ae9f-15bb565bc94a&notebook=linux)

# salt 从入门到精通

linux slat

* [salt 从入门到精通](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#salt-从入门到精通)
  + [基础介绍](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#基础介绍)
  + [证书管理](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#证书管理)
    - [salt 远程执行](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#salt-远程执行)
      * [目标](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#目标)
    - [管理对象 target](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#管理对象-target)
    - [Grains](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#grains)
    - [数据管理中心 Pillar](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#数据管理中心-pillar)
    - [针对管理对象操作](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#针对管理对象操作)
    - [配置管理](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#配置管理)
    - [执行结果处理](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#执行结果处理)
  + [salt 状态模块](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#salt-状态模块)
  + [salt实践](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#salt实践)
    - [分发文件](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#分发文件)
  + [jinja2](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#jinja2)
* [将nginx加到systemd服务中](https://app.yinxiang.com/note/01e61863-812a-4c5a-ae9f-15bb565bc94a?usn=7336#将nginx加到systemd服务中)

## 基础介绍

[salt官司网](http://docs.saltstack.com/)

salt 组件有 master; minions; excution modules; states; grains; pillar; top file; runner; ssh 等 详情请参见

<https://docs.saltstack.com/en/getstarted/overview.html>

salt有三种运行模式

* local 本地模式 （一台主机也可以使用）
* master/Minion 主从模式
* salt SSH #可用于批量安装salt-minion   
  有三大功能
* 运程执行
* 配置管理（状态管理）
* 云管理

安装salt

yum install salt-master salt-minion -y

chkconfig salt-master on

chkconfig salt-minion on

配置minion的配置文件

vim /etc/salt/minion

16 master: 10.0.0.111

72 id: #fqdn 完整的主机名

启动 /etc/init.d/salt-minion start

saltstack是一种基于c/s架构的服务模式，server端叫作master,client端叫做minion. 他们都以守护进程模式运行，一直监听配置文件里面定义的ret\_port(接受minion请求)和publish\_port(发布消息)

salt会依赖一些模块

* msgpack-python, saltstack 消息交换库
* YAML saltstack配置解析定义语法
* Jinja2 saltstack 配置模板
* MarkupSafe python unicode 转换库
* apache-libcloud saltstack对去架构编排库
* requests HTTP python 库
* ZeroMQ saltstack消息系统
* pyzqm zeromq python库
* pycrypto python 密码库
* m2crypto openssl python包装库

## 证书管理

要使用salt便必须使用认证。

cd /etc/salt/pki/minion

minion.pem minion.pub

安装minion后会把公钥发给master,

cd /etc/salt/pki/master

里面有master的公钥私钥。

环境准备

* master 端 主机名**master\_101** ip 172.16.1.101
* minion 端 主机名**client\_102** ip 172.16.1.102
* 在每一台机子都做hosts解析   
  证书管理是在master端 由salt-key管理

安装完minion端后需要在 配置文件/etc/salt/minion中修改参数才可以接收key

$ sed -i 's/#master: salt/master: 172.16.1.101/g' /etc/salt/minion

$ salt-key -A -y

The following keys are going to be accepted:

Unaccepted Keys:

client\_102

master\_101

Key **for** minion client\_102 accepted.

Key **for** minion master\_101 accepted.

$ salt-key -L

Accepted Keys:

client\_102

master\_101

Denied Keys:

Unaccepted Keys:

Rejected Keys:

$ salt '\*' test.ping

client\_102:

True

master\_101:

True

### salt 远程执行

目标(target)

模块(module)

返回(returnners)

salt '\*' test.**ping** # salt target module.method key（test模块里的ping方法）

salt '\*' cmd.**run** 'uptime'

#### 目标

目标可以通过正则来匹配minion id. 或者用grains, pillar subnet/ip address, compound matching, node groups 来匹配

vim /etc/salt/**master**

**file\_roots**:

base:

- /srv/salt # 把文件放到这，

默认必须是一个base环境，

$ cd /srv/salt

$ cat apache.sls

apache-install: #起个名字

pkg.installed: #yum安装

-names:

- httpd

- httpd-devel

apache-service:

service.running:

- name: httpd

- enable: True # 开启自启动

- reload: True #支持重载

$ salt '\*' salte.sls apache

如果有多个状态文件，那么就需要一个入口文件 top.sls 放到base下，其它的状态文件可以放到做生意位置

$ cat top.sls

base:

'[linux-node2.example.com](http://linux-node2.example.com/)':

- apache

$ salt '\*' **state**.highstate

### 管理对象 target

saltstack系统中我们的管理对象叫作target,在master上我们可以采用不同的target 去管理不同的minion

在target options下可以分很多种匹配方式

[root@master\_101 ~]# salt -L ‘master\*,client\*’ test.ping

No minions matched the target. No command was sent, no jid was assigned.

ERROR: No return received

$ salt -E 'master\*' test.ping # 正则匹配

master\_101:

True

$ salt -L 'master\_101,client\_102' test.ping # 列表

master\_101:

True

client\_102:

True

### Grains

可用于做信息查询

Grains 是saltstack组件之一，记录saltstack Minion 的一些静态信息的组件，(CPU, 内存, 磁盘, 网络, 等) 可以通过grains.items查看某台minion的所有Grain信息，minion的grains信息会在minions启动时汇报给master,在实际应用环境中我们需要根据自己的业务需求去算定义grains

$ salt 'minion\*' grains.ls # 查看所的grains的key

$ salt 'minion\*' grains.items #查看所有item的key与值

[root@master-node1 salt]# salt 'master\*' grains.item fqdn

[master-node1.gsandow.com](http://master-node1.gsandow.com/):

----------

fqdn:

[master-node1.gsandow.com](http://master-node1.gsandow.com/)

[root@master-node1 salt]# salt 'master\*' grains.get fqdn

[master-node1.gsandow.com](http://master-node1.gsandow.com/):

[master-node1.gsandow.com](http://master-node1.gsandow.com/)

[root@linux-node1 /srv/salt/\_grains]# cat my\_grains.py

#!/usr/bin/env python

#-\*- coding: utf-8 -\*-

**def** **my\_grains**():

# 初始化一个grains字典

grains = {}

grains['iaas'] = 'openstack'

grains['edu'] = 'sandow'

**return** grains

[root@linux-node1 /srv/salt/\_grains]# salt '\*' saltutil.sync\_grains

[linux-node1.example.com](http://linux-node1.example.com/):

- grains.my\_grains

[linux-node2.example.com](http://linux-node2.example.com/):

- grains.my\_grains

grains 优先级

系统自带

grains 文件写的

master grains

可以用于使用grains做机器的匹配

[root@master-node1 salt]# salt 'minion\*' grains.get os

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

CentOS

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

CentOS

[root@master-node1 salt]# salt 'master\*' grains.get os

master-[node1.gsandow.com](http://node1.gsandow.com/):

CentOS

[root@master-node1 salt]# salt -G os:CentOS cmd.run 'w'

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

19:36:41 up 3:25, 2 users, load average: 0.00, 0.00, 0.00

**USER** **TTY** FROM LOGIN@ IDLE JCPU PCPU WHAT

root tty1 - 13:21 6:15m 0.01s 0.01s -bash

root pts/1 10.0.0.1 18:46 23:43 0.02s 0.02s -bash

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

19:36:41 up 3:26, 2 users, load average: 0.00, 0.00, 0.00

**USER** **TTY** FROM LOGIN@ IDLE JCPU PCPU WHAT

root tty1 - 13:20 6:15m 0.01s 0.01s -bash

root pts/1 10.0.0.1 18:46 15:47 0.02s 0.02s -bash

master-[node1.gsandow.com](http://node1.gsandow.com/):

19:36:41 up 3:31, 2 users, load average: 0.01, 0.00, 0.00

**USER** **TTY** FROM LOGIN@ IDLE JCPU PCPU WHAT

root tty1 - 19:04 31:30 0.29s 0.27s vim /etc/ssh/ss

root pts/1 10.0.0.1 18:45 1.00s 0.60s 0.45s /usr/bin/python

在minion配置文件中配置grains 或者使用写到 /etc/salt/grains

grains:

roles:

- webserver

- memcache

[root@master-node1 salt]# salt '\*' grains.item roles

[master-node1.gsandow.com](http://master-node1.gsandow.com/):

----------

roles:

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

----------

roles:

- webserver

- memcache

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

----------

roles:

$ cat /etc/salt/grains

web: nginx

$ salt '\*' grains.item web

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

----------

web:

nginx

[master-node1.gsandow.com](http://master-node1.gsandow.com/):

----------

web:

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

----------

web:

也可以在top file里

$ cat top.sls

base:

'web:nginx':

- match: grain

- apache

查看grains函数

[root@master\_101 ~]# salt -E 'client\*' sys.list\_functions grains

client\_102:

- grains.append

- grains.delval

- grains.filter\_by

- grains.get

- grains.get\_or\_**set**\_**hash**

- grains.has\_value

- grains.item

- grains.items

- grains.ls

- grains.remove

- grains.setval

- grains.setvals

为grains增加一个item并查看

[root@master\_101 minion.d]# salt -E 'master\*' grains.append saltbook 'verycool'

master\_101:

----------

saltbook:

- verycool

[root@master\_101 minion.d]# salt -E 'master\*' grains.item saltbook

master\_101:

----------

saltbook:

- verycool

[root@master\_101 minion.d]# salt -E 'master\*' grains.setvals "{'salt': 'good', 'book':'cool'}"

master\_101:

----------

book:

cool

salt:

good

### 数据管理中心 Pillar

Pillar 也是salt组件之一，叫数据管理中心，会经常配合states在大规模配置管理工作中使用它，pillar在saltstack中主要的作用就是存储和定义配置管理中需要的一些数据，比如软件版本号，用户名，密码等信息，它的定义存储格式跟grains类似，

pillar 用来定位主机。 修改完pillar时需要刷新

[root@master-node1 pillar]# salt '\*' saltutil.refresh\_pillar

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

True

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

True

master-[node1.gsandow.com](http://node1.gsandow.com/):

True

[root@master-node1 pillar]# salt -I 'apache:httpd' test.ping

master-[node1.gsandow.com](http://node1.gsandow.com/):

True

[minion-node1.gsandow.com](http://minion-node1.gsandow.com/):

True

[minion-node2.gsandow.com](http://minion-node2.gsandow.com/):

True

给minion指定他想要的数据，查看pillar salt '\*' pillar.items

首先要把master里552行 改成True . **pillar\_opts: True** 不过一般是不用于开启的。

master 配置文件中指定了pillar的文件位置放置地方，

#pillar\_roots:

# base:

# - /srv/pillar

所以我们可以在这个文件下面新建几个文件

[root@master-node1 pillar]# cat apache.sls

{% **if** grains['os'] == 'CentOS' %}

apache: httpd

{% **elif** grains['os'] == 'Debian' %}

apache: apache2

{% endif%}

[root@master\_101 pillar]# cat packages.sls

zabbix:

package-name: zabbix

version: 2.2.4

[root@master\_101 pillar]# cat services.sls

zabbix:

port: 10050

user: admin

[root@master\_101 pillar]# cat top.sls

base: #指定环境

'\*': # target

- packages # 引用package.sls 或者packages/init.sls

- services # 引用services.sls

- apache

# 然后便可以使用命令来查看pillar啦

[root@master\_101 pillar]# salt '\*' pillar.item zabbix

master\_101:

----------

zabbix:

----------

package-name:

zabbix

port:

10050

user:

admin

version:

2.2.4

grains 与 pillar的区别

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **存储位置** | **数据类型** | **数据采集更新方式** | **应用** |
| Grains | Minion端 | 静态数据 | Minion启动时收集，也可以使用saltutil.sync\_grans进行刷新 | 存储Mnion基本数据，比如用于匹配Minion,自身数据可以用来做资产管理等。 |
| Pillar | Master端 | 动态数据 | 在master端定义，指定给对应的minion，可以使用saltutil.refresh\_pillar刷新 | 存储master指定的数据，只有指定的Minion可以看到。用于敏感数据保存 |

### 针对管理对象操作

**module** 是我们日常使用saltstack最多的一个组件，是用于管理对象操作的，这也是saltstack通过push的方式管理的入口，比如管理日常简单的执行命令，查看包安装情况，查看服务运行情况等都是通过 **module**来实现的

查看 **cmd module**的所有function

[root@master\_101 pillar]# salt 'master\*' sys.list\_functions cmd

master\_101:

- cmd.exec\_code

- cmd.exec\_code\_all

- cmd.has\_**exec**

- cmd.retcode

- cmd.run

- cmd.run\_all

- cmd.run\_chroot

- cmd.run\_stderr

- cmd.run\_stdout

- cmd.script

- cmd.script\_retcode

- cmd.shell

- cmd.shells

- cmd.tty

- cmd.which

- cmd.which\_bin

### 配置管理

**States**是saltstack中的配置语言，在日常进行配置管理时需要编写大量的states文件，比如我们需要安装一个包，然后管理一个配置文件，最后保证某个服务正常运行。需要我们编写一些states sls文件（描述状态配置的文件）去描述和实现我们的功能。

查看states列表可以使用命令salt 'Minion' sys.list\_state\_modules

查看指定states(git)的所有functions salt 'Minion' sys.list\_state\_functions git

查看指定function的用法salt 'Minion' sys.state\_doc git.config

### 执行结果处理

return组件可以理解为saltstack系统对执行minion返回后的数据进行存储或者返回给其他程序，

## salt 状态模块

状态可以多次执行，

先看两个例子

# cat apache.sls

apache-install: #id声明

pkg.installed:

- name:

- httpd

- httpd-devel

apache-service:

service.running:

- name: httpd

- enable: True

- reload: True

# cat top.lsl

id 声明可以不写，但写上是为了方便以后管理及引用，易理解。

/etc/http/conf/http.conf; # 文件名

file.managed: # 模块.方法

- source: salt://apache/http.conf

- user: root

- group: root

- mode: 644

- template: jinja

- defaults:

custom\_var: "default value"

other\_var: 123

{% if grains['os'] == 'Ubuntu' %}

- context:

custom\_var: 'override'

{% endif %}

文件： 描述应该存在于那个服务器上，

file.managed

file 模块 managed为方法

整个状态需要有一个入口，这个入口便是top file ，

cat top.sls

base:

'web:nginx':

- match: grain

- apache

top file放在那里需要在配置文件里审明,top file 放在base目录下面

vim /etc/salt/master

file\_roots:

base:

- /srv/salt/base

test:

- /srv/salt/test

prod:

- /srv/salt/prod

写一个文件管理

cd /srv/salt/base/

cat dns.sls

/etc/resolv.conf:

file.managed:

- source: <salt://files/resolv.conf>

- user: root

- group: root

- mode: 644

- template: jinja

- defaults:

DNS\_SERVER: 10.0.

vim files/resolv.conf

nameserver {{ DNS\_SERVER }}

source 下面的目录就是指定原文件目录。要把resolve.conf放在目录下

然后执行状态文件

salt ‘\*’ state.sls dns

第二种方法：加到高级状态，只要有对应关系都要执行

$ vim /srv/salt/base/top.sls

base:

'\*' :

- dns

$ vim files/reslove.conf

# {{ grains ['fqdn\_ip4']}}

nameserver {{ DNS\_SERVER}}

$ salt '\*' state.highstate

架构布署

1。 系统初始化，

2 功能模块

3 业务模块

jinjia棋版

python的模版引擎，

jinja 模版来写keepalived的优写级

{% if grains['fqdn']=='[lb-node1.unixhot.com](http://lb-node1.unixhot.com/)' %}

- ROUTEID: HAPROXY\_MASTER

- STATEID: MASTER

- PRIORITYID: 101

{% elif grains['fqdn']=='[lb-node2.unixhot.com](http://lb-node2.unixhot.com/)' %}

- ROUTEID: HAPROXY\_BACKUP

- STATEID: BACKUP

- PRIORITYID: 100

{% endif %}

## salt实践

### 分发文件

1. salt-cp 批量分发文件   
   salt-cp 语法格式为

salt-cp '\*' [ **options** ] **SOURCE** DEST

salt-cp -E '.\*' [ **options** ] **SOURCE** DEST

salt-cp -G 'os:Arch.\*' [ **options** ] **SOURCE** DEST

salt-cp -G 'roles:nginx' conf/nginx /etc/init.d/

这样一条命令便可以把nginx文件分发到所有 roles中有nginx的服务器上

1. 通过状态模块分发文件   
   在正式环境下不使用 \* 在执行状态的时候在后面加上 test=True

[root@test02 salt]# pwd

/srv/salt

[root@test02 salt]# ll conf/

total 20

-rwxr-xr-x 1 root root 1712 May 11 17:45 nginx

-rw-r--r-- 1 root root 2037 May 11 17:26 nginx.conf

-rw-r--r-- 1 root root 10361 May 10 10:56 zabbix\_agentd.conf

[root@test02 salt]# cat nginx\_conf.sls

/usr/local/nginx/conf/nginx.conf:

file.managed:

- name: /usr/local/nginx/conf/nginx.conf

- source: salt://conf/nginx.conf

- user: root

- group: root

- mode: 644

[root@test02 salt]# salt -G 'roles:nginx' state.sls nginx\_conf

如果分发文件里面的配置有一些是动态的，那么就需要使用 jinja 模版

[root@test02 salt]# cat zabbix\_agent.sls

/etc/zabbix/zabbix\_agentd.conf:

file.managed:

- name: /etc/zabbix/zabbix\_agentd.conf

- source: salt://conf/zabbix\_agentd.conf

- user: root

- group: root

- mode: 644

- template: jinja

- defaults:

hostname: grains['fqdn']

[root@test02 salt]# grep '^[a-Z]' conf/zabbix\_agentd.conf

PidFile=/var/run/zabbix/zabbix\_agentd.pid

LogFile=/var/log/zabbix/zabbix\_agentd.log

LogFileSize=0

Server=10.51.48.249

ServerActive=10.51.48.249:10051

Hostname= {{ grains['fqdn'] }}

Include=/etc/zabbix/zabbix\_agentd.d/

[root@test02 salt]# salt -G 'roles:nginx' state.sls zabbix\_agent

文件管理之为文件增加一些东西

salt ‘\*’ file.append /etc/hosts ‘this is add line’

salt

zeromq 消息队列

状态不能回滚

apache-install: #id 声明（在高级状态下id必须唯一）， name 声明

1 安装软件包 pkg

2 修改配置文件 file

3 启动服务 service

网络 系统 数据库 去计算 自动化 架构 安全

一个id声明下面，状态模块不能重复使用

common\_packges:

pkg.installed:

- pkgs:

- unzip

- dos2unix

- salt-minion: 2015.8.5-1.el6

lamp-pkg:

pkg.installed:

- pkgs:

- httpd

- php

- marriadh-server

- php-mysql

- php-cli

- php-mbstring

/etc/http/conf/http.conf:

file.managed:

- source: <salt://apache/http.conf>

- user: root

- group: root

- mode: 644

- template:jinja

- defaults:

php-config:

file.managed:

- name: /etc/php.ini

- source: <salt://files/php.ini>

- user: root

- group: root

- mode: 644

php-config:

file.managed:

- name: /etc/my.cnf

- source: <salt://files/my.cnf>

- user: root

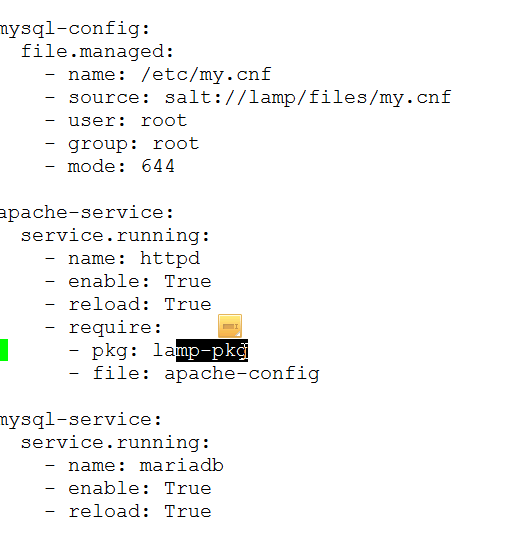
- group: root

- mode: 644

依赖关系

状态间关系：

1 我依赖谁 require



2 我被谁依赖 require\_in

mysql-config:

file.managed:

- name:/etc/my.cnf

- require\_in:

- service: mysql-service

3 我监控谁 watch

监控某一个状态，如果状态变化 就做一些事

apache-servie:

* watch:
  + file: apache-config

如果apache.conf 这个id的状态发生变化，变 reload，如果不加reload: True, 那么就restart

4 被谁监控 watch\_in

5 我引用谁 include

6 我扩展谁

如何编写sls技巧：

1 按状态分类 如果单独使用，清晰

2 按服务分类，可以被其他的sls include.

## jinja2

{{% … %}}

{{[ … ]}} 变量

3 步走：

1 告诉file模块， 你要使用jinja

template: jinja

2 你要列出参数列表

* defaultes:   
  PORT: 88  
    
  模板里面支持 salt grains pillar 进行赋值   
  salt 运程执行模块 {{ salt[‘network.hw\_addr’](‘eth0’) }}   
  {{ pillar[‘apache’]}}

1 系统初始化

2 功能模块： 设置单独的目录 haproxy nginx php mysql memcached

3 业务模块： 根据业务类型划分，例如web 服务，论坛，bbs

干活： 1 salt环境配置

开发，测试，（功能测试环境，性能测试环境） 预生产，生产

base 基础环境

init目录： 环境初始化 1 dns配置， 2 history记录时间， 3 记录命令操作 4 内核参数优化 5 安装yum仓库 6 安装zabbix-agent

socket 网络套接字

五元组

源地址，源端口， 目的地址，目的端口， 协议

file.manage

backup:minion

会把minion文件备份到下面目录下

/var/cache/salt/minion/file\_baup/

继续学习状态间关系

1. unless
2. onlyif

把变量写到最上面，然后在下面引用

{% set keepalived\_tar= ‘keepalived-1.2.17.tar.gz’ %}

name: {{ keepalived\_tar }}

session

对于php程序来说，只要在../php/lib/php.ini文件里面配置即可session.save\_handler = files【session存储形式—files或者memcache】；session.save\_path = “[tcp://IP:Port](tcp://IP:Port/)”【session的存储路径】，他的session默认是保存在/tmp目录下，但是我们将他们都挂载到后端的memcache内存缓存服务器中。

例如：

session.save\_handler = memcache

session.save\_path = “tcp://127.0.0.1:11211” 或者”[tcp://host1:port1?persistent=1&weight=2](tcp://host1:port1/?persistent=1&weight=2),[tcp://host2:port2](tcp://host2:port2/)”

对于java程序来说，一般的session共享也是有一个memcache服务，比较多的情况下使用程序来实现session共享，通过一个变量将session记下来，然后存到memcache里面，需要的时候在取。

另外一般比较大的java程序一般都会使用cookie，将session存到客户端，这样情况下就不需要共享，只要我本地设置了cookie就可以了

# 将nginx加到systemd服务中

cat > /usr/lib/systemd/system/nginx.service << ‘EOF’

[Unit]

Description=nginx - high performance web server

Documentation=<http://nginx.org/en/docs/>

After=network.target remote-fs.target nss-lookup.target

[Service]

Type=forking

PIDFile=/run/nginx.pid

ExecStartPre=/usr/sbin/nginx -t -c /etc/nginx/nginx.conf

ExecStart=/usr/sbin/nginx -c /etc/nginx/nginx.conf

ExecReload=/bin/kill -s HUP /var/folders/mq/4lvpwvdj5y353n3h1fv3lv3m0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/p1686MAINPID

PrivateTmp=true

[Install]

WantedBy=multi-user.target

EOF

master\_job\_cache: mysql

mysql.host: ‘192.168.56.11’

mysql.user: ‘salt’

mysql.pass: ‘salt@pw’

mysql.db: ‘salt’

mysql.port: 3306

systemctl restart salt-master

<https://www.unixhot.com/docs/saltstack/ref/modules/all/salt.modules.saltutil.html#module-salt.modules.saltutil>

1 修改minion配置文件

file-client ; local

file\_roots:

piller\_roots

2 编写SLS

3 salt-call –local sate.highstate

<https://www.unixhot.com/docs/saltstack/ref/netapi/all/salt.netapi.rest_cherrypy.html>

SALT-API

1 https 证书

2 配置证书

3

useradd -M -s /sbin/nologin saltapi

passwd saltapi

[root@linux-node1 /etc/pki/tls/certs]# make testcert

cd /etc/pki/tls/private

openssl rsa -in localhost.key -out salt\_nopass.key

systemctl start salt-api

pip install CherryPy==3.2.6

[root@linux-node1 ~]# vim /etc/salt/master

default\_include: master.d/\*.conf

rest\_cherrypy:

host: 192.168.56.11

port: 8000

ssl\_crt: /etc/pki/tls/certs/localhost.crt

ssl\_key: /etc/pki/tls/private/salt\_nopass.key

external\_auth:

pam:

saltapi:

- .\*

- ‘@wheel’

- ‘@runner’

[root@linux-node1 ~]# curl -k <https://192.168.56.11:8000/login> \

-H ‘Accept: application/x-yaml’ \

-d username=’saltapi’ \

-d password=’saltapi’ \

-d eauth=’pam’

return:

* eauth: pam   
  expire: 1464663850.123221   
  perms:
  + .\*
  + ‘@wheel’
  + ‘@runner’   
    start: 1464620650.123221   
    token: 785db9bc5e79dee828bfb1649bc49c59900e0ebf   
    user: saltapi

curl -k <https://192.168.56.11:8000/minions/linux-node1.oldboyedu.com> \

-H ‘Accept: application/x-yaml’ \

-H ‘X-Auth-Token: 785db9bc5e79dee828bfb1649bc49c59900e0ebf’ \

curl -k <https://192.168.56.11:8000/> \

-H ‘Accept: application/x-yaml’ \

-H ‘X-Auth-Token: 785db9bc5e79dee828bfb1649bc49c59900e0ebf’ \

-d client=’runner’ \

-d fun=’manage.status’

curl -k <https://192.168.56.11:8000/> \

-H ‘Accept: application/x-yaml’ \

-H ‘X-Auth-Token: 785db9bc5e79dee828bfb1649bc49c59900e0ebf’ \

-d client=’local’ \

-d tgt=’\*’ \

-d fun=’test.ping’

<https://github.com/binbin91/oms>