Python

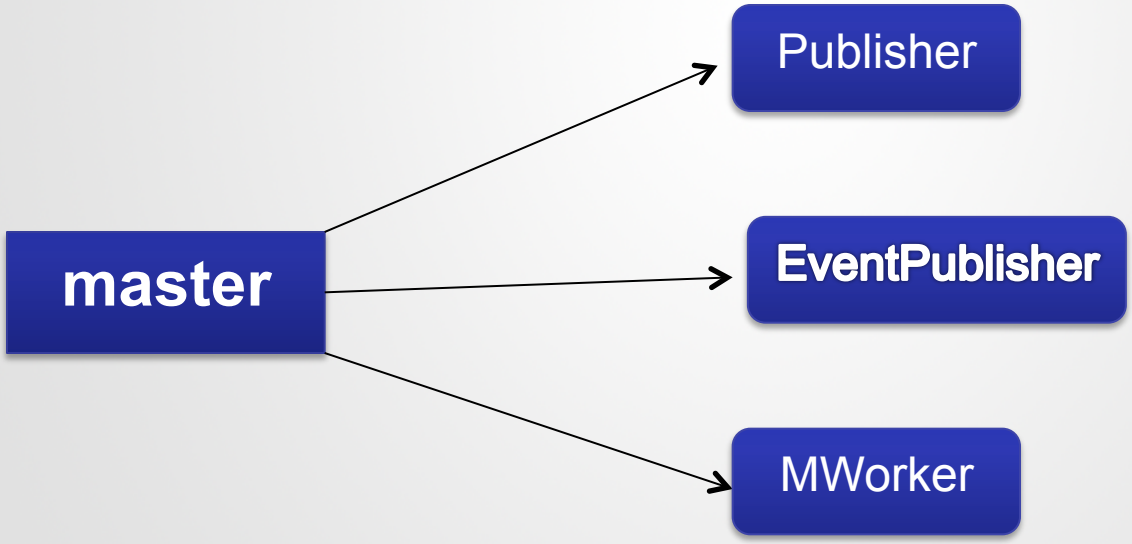
saltstack 初探

QQ群：365534424

目录

工作流

Saltstack Master的基本构成



工作流

Pulisher工作流

工作流

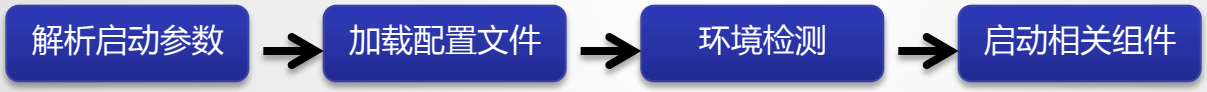
EventPulisher工作流

工作流

Mworker&reqsever工作流

启动

Saltstack 启动流程



启动

Saltstack 启动流程

入口函数

/etc/init.d/salt-master

```
def salt_master():
    '''
    Start the salt master.
    '''
    import salt.cli.daemons
    master = salt.cli.daemons.Master()
    master.start()
```

Saltstack 启动流程

入口函数

salt/cli/daemons.py

```
class Master(parsers.MasterOptionParser):
    '''
    Creates a master server
    '''
    def prepare(self):
        '''
        Run the preparation sequence required to start a salt master server.
        If sub-classed, don't **ever** forget to run:
            super(YourSubClass, self).prepare()
        '''
        self.parse_args()
```

启动

Saltstack 启动流程

入口函数

salt/cli/daemons.py

```
def start(self):
    '''
    Start the actual master.
    If sub-classed, don't **ever** forget to run:
        super(YourSubClass, self).start()
    NOTE: Run any required code before calling `super()`.
    '''
    self.prepare()
    if check_user(self.config['user']):
        logger.info('The salt master is starting up')
        self.master.start()
```

Saltstack 启动流程

入口函数

salt/cli/daemons.py

```
if self.config['transport'].lower() == 'zeromq':
        if not verify_socket(self.config['interface'],
                             self.config['publish_port'],
                             self.config['ret_port']):
            self.exit(4, 'The ports are not available to bind\n')
        self.config['interface'] = ip_bracket(self.config['interface'])
        migrations.migrate_paths(self.config)

        # Late import so logging works correctly
        import salt.master
        self.master = salt.master.Master(self.config)
    else:
        # Add a udp port check here
        import salt.daemons.flo
        self.master = salt.daemons.flo.IofloMaster(self.config)
    self.daemonize_if_required()
    self.set_pidfile()
```

Saltstack 启动流程

入口函数

salt/master.py

```
class Master(SMaster):
    '''
    The salt master server
    '''
    def __init__(self, opts):
        '''
        Create a salt master server instance
        :param dict: The salt options
        '''
        # Warn if ZMQ < 3.2
        try:
```

Saltstack 启动流程

入口函数

salt/master.py

```
def start(self):
    self._pre_flight()

        enable_sigusr1_handler()
        enable_sigusr2_handler()

        self.__set_max_open_files()
        log.info('Creating master process manager')
        process_manager = salt.utils.process.ProcessManager()
        log.info('Creating master maintenance process')
        process_manager.add_process(Maintenance, args=(self.opts,))
        log.info('Creating master publisher process')
        process_manager.add_process(Publisher, args=(self.opts,))
        log.info('Creating master event publisher process')
        process_manager.add_process(salt.utils.event.EventPublisher, args=(self.opts,))
        salt.engines.start_engines(self.opts, process_manager)
```

Saltstack指令下发

入口函数

/bin/salt

```
def salt_main():
    '''
    Publish commands to the salt system from the command line on the
    master.
    '''
    import salt.cli.salt
    if '' in sys.path:
        sys.path.remove('')
    client = None
    try:
        client = salt.cli.salt.SaltCMD()
        client.run()
    except KeyboardInterrupt as err:
        trace = traceback.format_exc()
        try:
            hardcrash = client.options.hard_crash
        except (AttributeError, KeyError):
            hardcrash = False
```

Saltstack指令下发

入口函数

salt/cli/salt.py

```
try:
    # We don't need to bail on config file permission errors
    # if the CLI
    # process is run with the -a flag
    skip_perm_errors = self.options.eauth != ''

    local = salt.client.get_local_client(
        self.get_config_file_path(),
        skip_perm_errors=skip_perm_errors)
except SaltClientError as exc:
```

Saltstack指令下发

入口函数

salt/client/__init__.py

```
def get_local_client(
        c_path=os.path.join(syspaths.CONFIG_DIR, 'master'),
        mopts=None,
        skip_perm_errors=False):


    import salt.config
        opts = salt.config.client_config(c_path)
    if opts['transport'] == 'raet':
        import salt.client.raet
        return salt.client.raet.LocalClient(mopts=opts)
    elif opts['transport'] == 'zeromq':
        return LocalClient(mopts=opts,
skip_perm_errors=skip_perm_errors)
```

Saltstack指令下发

入口函数

salt/client/__init__.py

```
class LocalClient(object):
    '''
    The interface used by the :command:`salt` CLI tool on the Salt Master
    ``LocalClient`` is used to send a command to Salt minions to execute
    :ref:`execution modules <all-salt.modules>` and return the results to the
    Salt Master.
    Importing and using ``LocalClient`` must be done on the same machine as the
    Salt Master and it must be done using the same user that the Salt Master is
    running as. (Unless :conf_master:`external_auth` is configured and
    authentication credentials are included in the execution).
    .. code-block:: python
        import salt.client
        local = salt.client.LocalClient()
        local.cmd('*', 'test.fib', [10])
```

Saltstack指令下发

入口函数

salt/client/__init__.py

```
def cmd(
        self,
        tgt,
        fun,
        arg=(),
        timeout=None,
        expr_form='glob',
        ret='',
        jid='',
        kwarg=None,
        **kwargs):
    '''
    Synchronously execute a command on targeted minions
    The cmd method will execute and wait for the timeout period for all
    minions to reply, then it will return all minion data at once.
    .. code-block:: python
```

Saltstack指令下发

入口函数

salt/client/__init__.py

```
local.cmd('*', 'test.arg', ['arg1', 'arg2'], kwarg={'foo': 'bar'})
    Compound commands can be used for multiple executions in a single
    publish. Function names and function arguments are provided in separate
    lists but the index values must correlate and an empty list must be
    used if no arguments are required.
    .. code-block:: python
        >>> local.cmd('*', [
                'grains.items',
                'sys.doc',
                'cmd.run',
            ],
            [
                [],
                [],
                ['uptime'],
            ])
```

Saltstack指令下发

入口函数

salt/client/__init__.py

```
arg = salt.utils.args.condition_input(arg, kwarg)
pub_data = self.run_job(tgt,
                        fun,
                        arg,
                        expr_form,
                        ret,
                        timeout,
                        jid,
                        **kwargs)
```

Saltstack指令下发

入口函数

run_job

Asynchronously send a command to connected minions

Prep the job directory and publish a command to any targeted minions.

:return: A dictionary of (validated) ``pub_data`` or an empty dictionary on failure. The ``pub_data`` contains the job ID and a list of all minions that are expected to return data.

.. code-block:: python

```
>>> local.run_job('*', 'test.sleep', [300])
{'jid': '20131219215650131543', 'minions': ['jerry']}
```

Saltstack指令下发

入口函数

run_job

```
pub_data = self.pub(
        tgt,
        fun,
        arg,
        expr_form,
        ret,
        jid=jid,
        timeout=self._get_timeout(timeout),
        **kwargs)

    return self._check_pub_data(pub_data)
```

Saltstack指令下发

入口函数

pub

```
Take the required arguments and publish the given command.
    Arguments:
        tgt:
            The tgt is a regex or a glob used to match up the ids on
            the minions. Salt works by always publishing every command
            to all of the minions and then the minions determine if
            the command is for them based on the tgt value.
        fun:
            The function name to be called on the remote host(s), this
            must be a string in the format "<modulename>.<function name>"
        arg:
            The arg option needs to be a tuple of arguments to pass
            to the calling function, if left blank
    Returns:
        jid:
            A string, as returned by the publisher, which is the job
            id, this will inform the client where to get the job results
        minions:
            A set, the targets that the tgt passed should match.
```

Saltstack指令下发

入口函数

pub

```
payload_kwargs = self._prep_pub(
                tgt,
                fun,
                arg,
                expr_form,
                ret,
                jid,
                timeout,
                **kwargs)

        master_uri = 'tcp://' + salt.utils.ip_bracket(self.opts['interface']) + \
                ':' + str(self.opts['ret_port'])
        channel = salt.transport.Channel.factory(self.opts,
                                                 crypt='clear',
                                                 master_uri=master_uri)
```

Saltstack指令下发

入口函数

/salt/transport/__init__.py

```
class Channel(object):
    '''
    Factory class to create communication-channels for different transport
    '''
    @staticmethod
    def factory(opts, **kwargs):
        # Default to ZeroMQ for now
        ttype = 'zeromq'

        # determine the ttype
        if 'transport' in opts:
            ttype = opts['transport']
        elif 'transport' in opts.get('pillar', {}).get('master', {}):
            ttype = opts['pillar']['master']['transport']

        # switch on available ttypes
        if ttype == 'zeromq':
            return ZeroMQChannel(opts, **kwargs)
```

Saltstack指令下发

入口函数

/salt/transport/__init__.py

```
def send(self, load, tries=3, timeout=60):
      if self.crypt == 'clear':  # for sign-in requests
         return self._uncrypted_transfer(load, tries, timeout)

def _uncrypted_transfer(self, load, tries=3, timeout=60):
      return self.sreq.send(self.crypt, load, tries, timeout)

@property
  def sreq(self):
      # When using threading, like on Windows, don't cache.
      # The following block prevents thread leaks.
      if not self.opts.get('multiprocessing'):
         return salt.payload.SREQ(self.master_uri, opts=self.opts)
```

Saltstack指令下发

入口函数

/salt/payload.py/SREQ

```
payload = {'enc': enc}
        payload['load'] = load
        pkg = self.serial.dumps(payload)
        self.socket.send(pkg)
        self.poller.register(self.socket, zmq.POLLIN)
        tried = 0
        while True:
            polled = self.poller.poll(timeout * 1000)
            tried += 1
            if polled:
                break
            if tries > 1:
                log.info('SaltReqTimeoutError: after {0} seconds. (Try {1} of {2})'.format(
                  timeout, tried, tries))
            if tried >= tries:
                self.clear_socket()
                raise SaltReqTimeoutError(
                    'SaltReqTimeoutError: after {0} seconds, ran {1} tries'.format(timeout * tried, tried)
                )
        return self.serial.loads(self.socket.recv())
```

Saltstack指令下发

入口函数

/salt/payload.py/Serial

```
class Serial(object):
    '''
    Create a serialization object, this object manages all message
    serialization in Salt
    '''
    def __init__(self, opts):
        if isinstance(opts, dict):
            self.serial = opts.get('serial', 'msgpack')
        elif isinstance(opts, str):
            self.serial = opts
        else:
            self.serial = 'msgpack'
```

Web ui 开发

入口函数

salt/netapi/__init__.py

```
class NetapiClient(object):
    '''
    Provide a uniform method of accessing the various client interfaces in Salt
    in the form of low-data data structures. For example:
    >>> client = NetapiClient(__opts__)
    >>> lowstate = {'client': 'local', 'tgt': '*', 'fun': 'test.ping', 'arg': ''}
    >>> client.run(lowstate)
```

Web ui 开发

入口函数

salt/netapi/__init__.py

```
class NetapiClient(object):
    '''
    Provide a uniform method of accessing the various client interfaces in Salt
    in the form of low-data data structures. For example:
    >>> client = NetapiClient(__opts__)
    >>> lowstate = {'client': 'local', 'tgt': '*', 'fun': 'test.ping', 'arg': ''}
    >>> client.run(lowstate)
```

Web ui 开发

入口函数

salt/netapi/__init__.py

```
def local(self, *args, **kwargs):
    '''
    Run :ref:`execution modules <all-salt.modules>` synchronously
    See :py:meth:`salt.client.LocalClient.cmd` for all available
    parameters.
    Sends a command from the master to the targeted minions. This is the
    same interface that Salt's own CLI uses. Note the ``arg`` and ``kwarg``
    parameters are sent down to the minion(s) and the given function,
    ``fun``, is called with those parameters.
    :return: Returns the result from the execution module
    '''
    local = salt.client.get_local_client(mopts=self.opts)
    return local.cmd(*args, **kwargs)
```

End