|  |  |
| --- | --- |
| 产品名称Product name | 密级Confidentiality level |
| 概要设计文档 | *内部* |
| 产品版本Product version | Total pages 共页 |
|  |

***概要设计文档***

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by  拟制 |  | Date  日期 | 2016-03-28 |
| Reviewed by  审核 |  | Date  日期 |  |
| Approved by  批准 |  | Date  日期 |  |



ZHICloud Technologies Co., Ltd.

致云科技有限公司

All rights reserved

版权所有 侵权必究

（REP01T01 V0.1 / for internal use only）

（REP01T01 V0.1 / 仅供内部使用）

[1. 文档介绍 3](#_Toc32388)

[1.1目的 3](#_Toc22330)

[1.2范围 3](#_Toc27879)

[1.3术语 3](#_Toc1827)

[2. 总体设计 4](#_Toc13974)

[2.1 模块功能 4](#_Toc15867)

[资源池管理 4](#_Toc30212)

[云主机管理 4](#_Toc29933)

[vpc网络管理 4](#_Toc3601)

[物理架构管理 5](#_Toc8252)

[服务器管理 5](#_Toc13159)

[镜像管理 5](#_Toc13708)

[状态管理 5](#_Toc29216)

[2.2 工作原理 5](#_Toc20877)

[2.3 三方库使用 6](#_Toc6455)

[3. 软件架构 7](#_Toc24599)

[3.1 模块结构描述 7](#_Toc714)

[3.2 功能流程图 7](#_Toc493)

[4. 内部实现 8](#_Toc10048)

[4.1 数据结构定义 8](#_Toc1471)

[云主机状态定义 8](#_Toc6863)

[机房信息 8](#_Toc13864)

[机房状态 8](#_Toc20466)

[机架信息 9](#_Toc3261)

[机架状态 9](#_Toc3794)

[服务器信息 10](#_Toc6756)

[服务器状态 10](#_Toc20653)

[云主机状态 11](#_Toc14217)

[云主机配置信息 11](#_Toc26557)

[云主机磁盘文件 14](#_Toc25334)

[存储设备 14](#_Toc24380)

[云主机forward转发器 14](#_Toc11341)

[磁盘镜像服务 17](#_Toc7266)

[光盘镜像 17](#_Toc12154)

[VPC网络 18](#_Toc20639)

[端口资源池 20](#_Toc4249)

[端口资源 22](#_Toc3678)

[4.2主要功能类定义 25](#_Toc9250)

[状态管理类 25](#_Toc7437)

[地址资源管理 26](#_Toc3416)

[计算资源池管理 32](#_Toc20035)

[配置管理 43](#_Toc11598)

[转发器管理 53](#_Toc25913)

[磁盘镜像管理 62](#_Toc20227)

[光盘镜像管理 66](#_Toc27382)

[监控管理 70](#_Toc8541)

[vpc网络管理 73](#_Toc23765)

[端口资源池管理 85](#_Toc22036)

[物理架构管理 92](#_Toc20565)

[状态管理 104](#_Toc13507)

[5. 接口定义 112](#_Toc15956)

[6. 参数配置 113](#_Toc11198)

[6.1 参数配置 113](#_Toc31005)

[6.2 启停方式 113](#_Toc29788)

# 文档介绍

## 1.1目的

介绍cs基本结构、业务流程、功能接口。

## 1.2范围

平台开发人员用于快速了解control\_server基本工作流程、软件架构、代码模块协作关系。

## 1.3术语

计算资源池——各个nc节点CPU、网络、存储资源集合

地址资源池——独享公网ip集合

端口资源池——将公网ip对应1~65535端口号映射给云主机，所有映射的公网ip端口集合

VPC网络——专有虚拟网络

# 总体设计

## 2.1 模块功能

### 资源池管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 计算资源池管理 | 创建|删除|修改|查询 |
| 地址资源池管理 | 创建|删除|修改|查询 |
| 端口资源池管理 | 创建|删除|修改|查询 |

### 云主机管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 云主机操作 | 创建|关机|开机（从光盘镜像）|重启（从光盘镜像）|强制关机|重置 |
| spice密码修改 |  |
| cpu优先级配置 |  |
| 云主机磁盘iops配置 |  |
| 备份 | 创建|恢复|查询 |
| 数据盘管理 | 添加|移除 |
| 光驱设备管理 | 添加|移除 |
| 共享ip端口绑定 | 绑定|解绑 |
| 云主机信息查询 |  |
| 转发器管理 | 添加|删除|修改 |

### vpc网络管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 创建 |  |
| 删除 |  |
| 添加云主机 |  |
| 打开vpc网络 |  |
| 关闭vpc网络 |  |

### 物理架构管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 机房管理 | 创建|删除|查询|编辑|监控 |
| 机架管理 | 创建|删除|查询|编辑|监控 |
| 服务器管理 | 创建|删除|查询|编辑|监控 |

### 服务器管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 存储管理 | 添加|删除|查询|禁用 |
| 网络聚合设备管理 | 添加|删除|查询 |
| 网卡设备管理 | 查询|删除|添加 |

### 镜像管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 光盘镜像管理 | 上传（nas、本地）|删除|查询 |
| 磁盘镜像管理 | 上传（本地）|创建|删除|查询 |

### 状态管理

|  |  |
| --- | --- |
| 功能 | 备注 |
| 云主机状态监控 | 查询 |
| 服务器状态监控 | 查询 |

## 2.2 工作原理

control\_server 通过NodeService加入工作域，和相关模块建立必要连接。NodeService初始化时，构造自身异步事物处理类ControlTransManager。NodeService服务模块运行时接收异步事物，通过ControlTransManager调用事物处理类处理响应的事物请求。

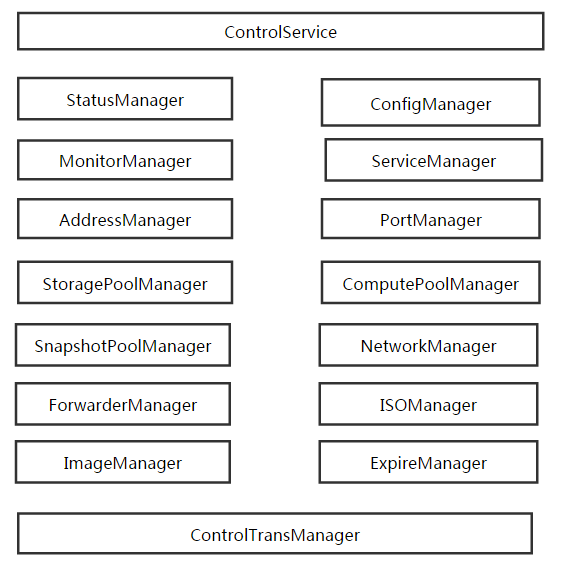
ControlTransManager初始化时注册事物回调处理类，事物处理类内部设置状态机跳转规则，ControlTransManager根据跳转规则调用调处理类内部相应处理函数。

## 2.3 三方库使用

无。

# 软件架构

## 3.1 模块结构描述

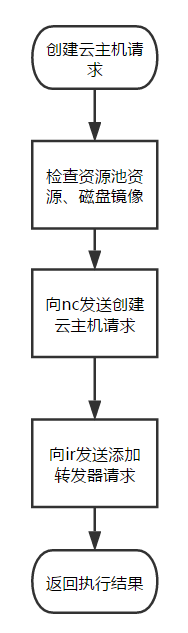


ControlService是平台模块服务，负责管理所有模块业务类。StatusManager管理状态监控；

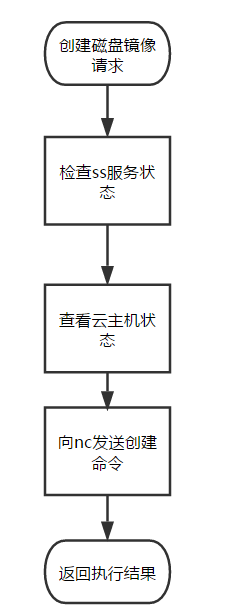
ConfigManager管理配置文件；MonitorManager管理云主机监控；ServiceManager管理服务模块；AddressManager管理地址资源；PortManager管理端口资源；StoragePoolManager管理云储存；ComputePoolManager管理计算资源；SnapshotManager管理快照；NetworkManager管理vpc云主机；ForwarderManager管理转发器；ISOManager管理光盘镜像；ImageManager管理磁盘镜像；ExpreManager管理平台模块定时监控任务；controlTransManager管理异步事物。

## 3.2 功能流程图

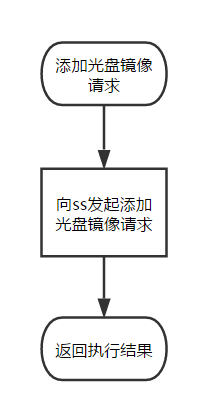
### 创建云主机



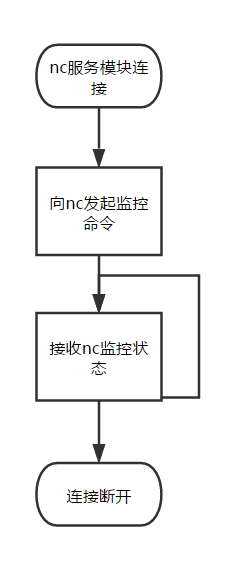
### 创建磁盘镜像



### 添加光盘镜像



### 状态监控



# 内部实现

## 4.1 数据结构定义

### 云主机状态定义

class StatusEnum(object):

running = 0

warning = 1

error = 2

stop = 3

### 机房信息

class ServerRoomInfo(object):

def \_\_init\_\_(self):

self.domain = ""

self.uuid = ""

self.name = ""

self.display\_name = ""

self.description = ""

### 机房状态

class ServerRoomStatus(object):

def \_\_init\_\_(self):

self.name = ""

self.uuid = ""

##[stop, warning, error, running]

self.server = [0, 0, 0, 0]

self.cpu\_count = 0

self.cpu\_usage = 0.0

##[available, total]

self.memory = [0, 0]

self.memory\_usage = 0.0

##[available, total]

self.disk\_volume = [0,0]

self.disk\_usage = 0.0

##read\_count, read\_bytes, write\_count, write\_bytes, io\_error

self.disk\_io = [0, 0, 0, 0, 0]

##receive\_bytes, receive\_packets, receive\_error, receive\_drop,

##send\_bytes, send\_packets, send\_error, send\_drop

self.network\_io = [0, 0, 0, 0, 0, 0, 0, 0]

##read, write, receive, send

self.speed = [0, 0, 0, 0]

self.status = StatusEnum.running

##format "YYYY-MM-DD HH:MI:SS"

self.timestamp = ""

### 机架信息

class ServerRackInfo(object):

def \_\_init\_\_(self):

self.server\_room = ""

self.uuid = ""

self.name = ""

### 机架状态

class ServerRackStatus(object):

def \_\_init\_\_(self):

self.name = ""

self.uuid = ""

##[stop, warning, error, running]

self.server = [0, 0, 0, 0]

self.cpu\_count = 0

self.cpu\_usage = 0.0

##[available, total]

self.memory = [0, 0]

self.memory\_usage = 0.0

##[available, total]

self.disk\_volume = [0,0]

self.disk\_usage = 0.0

##read\_count, read\_bytes, write\_count, write\_bytes, io\_error

self.disk\_io = [0, 0, 0, 0, 0]

##receive\_bytes, receive\_packets, receive\_error, receive\_drop,

##send\_bytes, send\_packets, send\_error, send\_drop

self.network\_io = [0, 0, 0, 0, 0, 0, 0, 0]

##read, write, receive, send

self.speed = [0, 0, 0, 0]

self.status = StatusEnum.running

##format "YYYY-MM-DD HH:MI:SS"

self.timestamp = ""

### 服务器信息

class ServerInfo(object):

def \_\_init\_\_(self):

self.rack = ""

self.uuid = ""

self.name = ""

self.ip = ""

### 服务器状态

class ServerStatus(object):

def \_\_init\_\_(self):

self.uuid = ""

self.cpu\_count = 0

self.cpu\_usage = 0.0

##[available, total]

self.memory = [0, 0]

self.memory\_usage = 0.0

##[available, total]

self.disk\_volume = [0,0]

self.disk\_usage = 0.0

##read\_count, read\_bytes, write\_count, write\_bytes, io\_error

self.disk\_io = [0, 0, 0, 0, 0]

##receive\_bytes, receive\_packets, receive\_error, receive\_drop,

##send\_bytes, send\_packets, send\_error, send\_drop

self.network\_io = [0, 0, 0, 0, 0, 0, 0, 0]

##read, write, receive, send

self.speed = [0, 0, 0, 0]

##format "YYYY-MM-DD HH:MI:SS"

self.timestamp = ""

self.status = StatusEnum.running

self.ip = ""

### 云主机状态

class HostStatusEnum(object):

status\_running = 0

status\_warning = 1

status\_error = 2

status\_stop = 3

class HostStatus(object):

def \_\_init\_\_(self):

self.cpu\_count = 0

self.cpu\_usage = 0.0

##[available, total]

self.memory = [0, 0]

self.memory\_usage = 0.0

##[available, total]

self.disk\_volume = [0,0]

self.disk\_usage = 0.0

##read\_count, read\_bytes, write\_count, write\_bytes, io\_error

self.disk\_io = [0, 0, 0, 0, 0]

##receive\_bytes, receive\_packets, receive\_error, receive\_drop,

##send\_bytes, send\_packets, send\_error, send\_drop

self.network\_io = [0, 0, 0, 0, 0, 0, 0, 0]

##read, write, receive, send

self.speed = [0, 0, 0, 0]

self.status = HostStatusEnum.status\_stop

##format "YYYY-MM-DD HH:MI:SS"

self.timestamp = ""

##server uuid

self.server = ""

self.uuid = ""

##[target\_ip, public\_ip]

self.ip = ["",""]

### 云主机配置信息

class PortProtocolEnum(object):

all\_protocol = 0

tcp\_protocol = 1

udp\_protocol = 2

class NetworkTypeEnum(object):

private = 0

mono = 1

share = 2

class DiskTypeEnum(object):

local = 0

cloud = 1

nas = 2

ip\_san = 3

class EnableLocalBackupEnum(object):

disabled = 0

enabled = 1

class EnableUsbExtEnum(object):

disabled = 0

enabled = 1

class BackupHostModeEnum(object):

fully = 0

partial = 1

class VideoTypeEnum(object):

mjpeg = 0

h264 = 1

h265 = 2

class HostPort(object):

def \_\_init\_\_(self):

self.server\_port = 0

self.host\_port = 0

self.public\_ip = ""

self.public\_port = 0

self.protocol = PortProtocolEnum.all\_protocol

class HostInfo(object):

def \_\_init\_\_(self):

self.container = ""

self.uuid = ""

self.name = ""

self.cpu\_count = 0

self.memory = 0

self.auto\_start = False

self.data\_disk\_count = 0

self.disk\_volume = [0, 0]

##list of HostPort

self.port = []

self.user = ""

self.group = ""

self.display = ""

self.authentication = ""

self.network = ""

self.inbound\_bandwidth = 0

self.outbound\_bandwidth = 0

self.max\_iops = 0

self.cpu\_priority = 0

self.server\_ip = ""

self.public\_ip = ""

##display port in server

self.server\_port = 0

##display port in public

self.public\_port = 0

## output port range, content type of int

self.output\_port\_range = []

##compute pool id

self.pool = ""

##forwarder uuid

self.forwarder = ""

self.network\_type = NetworkTypeEnum.private

##address pool id

self.network\_source = ""

self.disk\_type = DiskTypeEnum.local

##storage pool id

self.disk\_source = ""

# self.vpc\_network = ""

self.vpc\_ip = ""

self.enable\_local\_backup = EnableLocalBackupEnum.disabled

self.enable\_usb\_ext = EnableUsbExtEnum.disabled

self.thin\_provisioning = ThinProvisioningModeEnum.disabled

self.backing\_image = BackingImageModeEnum.disabled

self.video\_type = VideoTypeEnum.mjpeg

### 云主机磁盘文件

class DiskImageFileTypeEnum(object):

raw = 0

qcow2 = 1

class DiskImage(object):

def \_\_init\_\_(self):

self.name = ""

self.uuid = ""

self.enabled = True

self.size = 0

self.tags = []

self.description = ""

self.group = ""

self.user = ""

self.path = "" # nas path

self.disk\_type = ComputeStorageTypeEnum.local

self.file\_type = DiskImageFileTypeEnum.raw

# #storage service

self.container = ""

### 存储设备

class Device(object):

def \_\_init\_\_(self):

self.storage\_pool = ""

self.uuid = ""

self.name = ""

self.status = 0

self.disk\_volume = []

self.level = 0

self.identity = ""

self.security = 0

self.crypt = 0

self.page = 0

### 云主机forward转发器

class ForwarderTypeEnum(object):

mono = 0

share = 1

domain = 2

vpc = 3

class ForwarderPort(object):

def \_\_init\_\_(self):

self.protocol = 0

self.server\_port = 0

self.host\_port = 0

self.public\_ip = ""

self.public\_port = 0

def toJSONString(self):

def computeSignature(self):

class HostForwarder(object):

def \_\_init\_\_(self):

self.logger = logging.getLogger("host\_forwarder")

self.uuid = ""

self.type = ForwarderTypeEnum.mono

self.host\_id = ""

self.host\_name = ""

self.public\_ip = []

self.public\_monitor = 0

self.server\_ip = ""

self.server\_monitor = 0

self.vpc\_ip = ""

self.vpc\_range = ""

self.output\_port\_range = [] # [min\_port, max\_port]

##list of ForwarderPort

self.port = []

self.enable = True

self.crc = 0

def copy(self):

def computeSignature(self):

def computePortSignature(self):

### 磁盘镜像服务

class ImageService:

def \_\_init\_\_(self):

self.service = ""

self.type = 0

self.ip = ""

self.port = 0

### 光盘镜像

class ISOImage(object):

def \_\_init\_\_(self):

self.name = ""

self.uuid = ""

self.enabled = True

self.size = 0

self.description = ""

self.ip = ""

self.port = 0

self.group = ""

self.user = ""

self.path = "" # nas path

self.disk\_type = ComputeStorageTypeEnum.local

# #storage service name

self.container = ""

### VPC网络

class NetworkStatus(object):

disabled = 0

enabled = 1

class NetworkInfo(object):

def \_\_init\_\_(self):

self.logger = logging.getLogger("network\_info")

self.\_lock = threading.RLock()

self.uuid = ""

self.name = ""

self.size = 0

self.description = ""

self.pool = ""

self.network\_address = ""

self.netmask = 0

self.broadcast\_address = ""

self.status = NetworkStatus.disabled

# containing host UUIDs

self.hosts = set()

# containing public ips

self.public\_ips = set()

# containing allocated (inner vpc)ips

self.allocated\_ips = set()

# key: "protocol:public\_ip:public\_port", value: "host\_uuid:host\_port"

self.bound\_ports = {}

def copy(self):

def allocateIp(self, count):

def deallocateIp(self, ip):

def containsPublicIp(self, ip):

### 端口资源池

class PortPool(object):

def \_\_init\_\_(self):

self.logger = logging.getLogger("PortPool")

self.name = ""

self.uuid = ""

self.enable = True

##key = ip, value = PorlResource

self.resource = {}

self.modified = True

def setModified(self, modified):

def isModified(self):

def statistic(self):

def isEmpty(self):

def addResource(self, ip\_list):

def removeResource(self, ip\_list):

def getAllResource(self):

def allocate(self, count):

def allocatePort(self, ip, count):

def deallocate(self, ip, port\_list):

def setAllocated(self, ip, port\_list):

def isAllAllocated(self, ip, port\_list):

def setUnallocated(self, ip, port\_list):

### 端口资源

class PortResource(object):

default\_begin\_port = 1024

default\_end\_port = 65535

def \_\_init\_\_(self):

self.logger = logging.getLogger("PortResource")

self.ip = ""

self.enable = True

self.begin\_port = PortResource.default\_begin\_port

self.end\_port = PortResource.default\_end\_port

self.count = self.end\_port - self.begin\_port

self.last\_offset = 0

##allocated port

self.allocated = set()

self.modified = True

def setModified(self, modified):

def isModified(self):

def allocatedCount(self):

def getIP(self):

def getAllocated(self):

def statistic(self):

def isAvailable(self, count):

def isEmpty(self):

def allocate(self, count):

def deallocate(self, port\_list):

def setAllocated(self, port\_list):

def isAllAllocated(self, port\_list):

def setUnallocated(self, port\_list):

## 4.2主要功能类定义

### 状态管理类

class StatusManager(object):

max\_timeout = 24

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

##key = room id, value = server room status

self.server\_room\_status = {}

##key = room id, value = counter

self.server\_room\_counter = {}

##key = rack id, value = server rack status

self.server\_rack\_status = {}

##key = rack id, value = counter

self.server\_rack\_counter = {}

##key = uuid, value = server status

self.server\_status = {}

##key = uuid, value = counter

self.server\_counter = {}

##key = uuid, value = host status

self.host\_status = {}

##key = uuid, value = counter

self.host\_counter = {}

##key = uuid, value = compute pool status

self.compute\_pool\_status = {}

##key = uuid, value = counter

self.compute\_pool\_counter = {}

self.system\_status = None

self.lock = threading.RLock()

### 地址资源管理

class AddressManager(object):

def \_\_init\_\_(self, resource\_path, logger\_name):

self.logger = logging.getLogger(logger\_name)

self.root\_path = os.path.join(resource\_path, "address")

if not os.path.exists(self.root\_path):

os.mkdir(self.root\_path)

##key = uuid, value = AddressPool

self.address\_pool = {}

##lazy save

self.modified = False

self.lock = threading.RLock()

def load(self):

def save(self):

def syncPoolList(self):

def syncPoolInfo(self):

def syncResource(self):

def queryAllPool(self):

def createPool(self, config):

def deletePool(self, uuid):

def containsPool(self, uuid):

def getPool(self, uuid):

def queryResource(self, pool\_id):

def statisticStatus(self):

def allocate(self, pool\_id):

def deallocate(self, pool\_id, ip):

### 计算资源池管理

class ComputePoolManager(object):

def \_\_init\_\_(self, resource\_path, logger\_name):

self.logger = logging.getLogger(logger\_name)

self.pool\_path = os.path.join(resource\_path, "compute")

if not os.path.exists(self.pool\_path):

os.mkdir(self.pool\_path)

# #key = uuid, value = compute Pool

self.compute\_pool = {}

# #uuid for default pool

self.default\_pool = ""

self.lock = threading.RLock()

def load(self):

def getDefaultPoolID(self):

def getDefaultPool(self):

def loadAllPool(self):

def savePoolList(self):

def savePoolInfo(self, pool\_id):

def deleteResourceFile(self, pool\_id, resource\_name):

def deletePoolPath(self, pool\_id):

def queryAllPool(self):

def createPool(self, config):

def modifyPool(self, uuid, config):

def deletePool(self, uuid):

def containsPool(self, pool\_id):

def getPool(self, uuid):

def addResource(self, pool\_id, resource\_list):

def removeResource(self, pool\_id, name\_list):

def searchResource(self, name):

def containsResource(self, pool\_id, name):

def getResource(self, pool\_id, name):

def containNetwork(self, network):

### 配置管理

class ConfigManager(object):

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

##key = uuid, value = server room info

self.server\_rooms = {}

##key = uuid, value = server rack info

self.server\_racks = {}

##key = uuid, value = server info

self.servers = {}

##key = uuid, value = HostInfo

self.hosts = {}

##key = room id, value = list of rack id

self.room\_members = {}

##key = rack id, value = list of server id

self.rack\_members = {}

##key = service name, value = list of host id

self.host\_in\_service = {}

##key = session id, value = host info

self.host\_in\_creating = {}

self.lock = threading.RLock()

def loadServerRooms(self, room\_list):

def loadServerRacks(self, room\_id, rack\_list):

def loadServers(self, rack\_id, server\_list):

def containsServerRoom(self, room\_id):

def containsServerRack(self, rack\_id):

def containsServer(self, server\_id):

def queryAllServerRooms(self):

def queryServerRacks(self, room\_id):

def getServerRack(self, rack\_id):

def queryServers(self, rack\_id):

def getServer(self, server\_id):

def addServerRoom(self, info):

def removeServerRoom(self, uuid):

def modifyServerRoom(self, uuid, info):

def addServerRack(self, info):

def removeServerRack(self, rack\_id):

def addServer(self, info):

def removeServer(self, server\_id):

def modifyServer(self, uuid, info):

def loadHosts(self, service\_name, host\_list):

def addHost(self, info):

def modifyHost(self, uuid, host\_dict):

def removeHost(self, uuid):

def updateHost(self, info):

def getHost(self, uuid):

def containsHost(self, uuid):

def queryHosts(self, service\_name):

def addCreatingHost(self, session\_id, host\_info):

def removeCreatingHost(self, session\_id):

def getAllCreatingHost(self):

### 转发器管理

class ForwarderManager(object):

def \_\_init\_\_(self, resource\_path, logger\_name):

self.logger = logging.getLogger(logger\_name)

self.config\_file = os.path.join(resource\_path, "forwarder.ini")

##key = uuid, value = HostForwarder

self.forwarder = {}

self.modified = False

self.lock = threading.RLock()

self.crc = 0

def save(self):

def setSaveFlag(self):

def load(self):

def create(self, hostForwarder):

def modify(self, uuid, config):

def modifyByDict(self, uuid, forwarder\_dict):

def put(self, hostForwarder):

def isInvalid(self, uuid):

def delete(self, uuid):

def updateTotalCRC(self):

def getCRC(self):

def contains(self, uuid):

def get(self, uuid, copy=False):

def query(self, forwarder\_type):

def getAllCRC(self):

def statistic(self):

def enable(self, uuid):

def disable(self, uuid):

### 磁盘镜像管理

class ImageManager(object):

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

# #key = uuid, value = disk image

self.images = {}

# #key = image name, value = uuid

self.image\_names = {}

# #key = service name, value = list of image uuid

self.image\_containers = {}

self.lock = threading.RLock()

def loadImages(self, container\_name, image\_list):

def containsImage(self, uuid):

def containsImageName(self, name):

def getImage(self, uuid):

def getAllImages(self):

def statisticStatus(self):

def addImage(self, image):

def removeImage(self, uuid):

def modifyImage(self, uuid, new\_image):

def removeAllImageInContainer(self, container\_name):

def updateImage(self, image):

### 光盘镜像管理

class ISOManager(object):

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

# #key = uuid, value = iso image

self.images = {}

# #key = image name, value = uuid

self.image\_names = {}

# #key = service name, value = list of image uuid

self.image\_containers = {}

self.lock = threading.RLock()

def loadImages(self, container\_name, image\_list):

def containsImage(self, uuid):

def containsImageName(self, name):

def getImage(self, uuid):

def getAllImages(self):

def statisticStatus(self):

def addImage(self, image):

def removeImage(self, uuid):

def modifyImage(self, uuid, new\_image):

def removeAllImageInContainer(self, container\_name):

def updateImage(self, image):

### 监控管理

class Monitor(object):

level = 0

listener\_list = []

global\_listener = []

target\_map = {}

def \_\_init\_\_(self, level):

self.level = level

##list of all task

self.listener\_list = []

##list of listen all

self.global\_listener = []

##key = target id, value = list of task id

self.target\_map = {}

class MonitorTask(object):

class MonitorManager(LoggerHelper):

def addTask(self, task):

def getTask(self, task\_id):

def removeTask(self, task\_id):

def getAllMonitor(self):

def checkTimeout(self):

def processHeartBeat(self, task\_id):

def isNodeMonitored(self, node\_name):

def getNodeMonitor(self, node\_name):

def isDomainMonitored(self, domain\_name):

def getDomainMonitor(self, domain\_name):

### vpc网络管理

class NetworkManager(object):

def \_\_init\_\_(self, resource\_path, logger\_name):

self.logger = logging.getLogger(logger\_name)

self.pool\_path = os.path.join(resource\_path, "network")

if not os.path.exists(self.pool\_path):

os.mkdir(self.pool\_path)

# key = uuid, value = NetworkInfo

self.network\_info = {}

self.\_min\_ip = socket\_util.convertAddressToInt("10.0.0.0");

self.\_max\_ip = socket\_util.convertAddressToInt("10.255.255.255");

# key = int\_begin\_ip, value = (int\_begin\_ip, int\_end\_ip, netmask, str\_begin\_ip, str\_end\_ip)

self.network\_resource = OrderedDict()

self.is\_sorted = True

self.\_lock = threading.RLock()

def load(self):

def loadAllNetwork(self):

def containsNetwork(self, uuid):

def getNetwork(self, network\_id):

def getAllNetworks(self):

def createNetwork(self, networkInfo):

def deleteNetwork(self, network\_id):

def sortResource(self):

def \_generateNetworkItem(self, netmask):

def deleteNetworkResource(self, str\_ip, netmask):

def putNetwork(self, networkInfo):

def saveNetworkList(self):

def deleteNetworkPath(self, network\_id):

def saveNetworkInfo(self, network\_id):

def startNetwork(self, network\_id):

def attachAddress(self, network\_id, ips, save=True):

def detachAddress(self, network\_id, ips, save=True):

def allocateIp(self, network\_id, count, save=True):

def deallocateIp(self, network\_id, ip, save=True):

def containAddressPool(self, address\_pool):

### 端口资源池管理

class PortManager(object):

def \_\_init\_\_(self, resource\_path, logger\_name):

self.logger = logging.getLogger(logger\_name)

self.root\_path = os.path.join(resource\_path, "port")

if not os.path.exists(self.root\_path):

os.mkdir(self.root\_path)

##key = uuid, value = Port Pool

self.port\_pool = {}

##lazy save

self.modified = False

self.lock = threading.RLock()

def load(self):

def save(self):

def syncPoolList(self):

def syncPoolInfo(self):

def syncResource(self):

def queryAllPool(self):

def createPool(self, config):

def deletePool(self, uuid):

def containsPool(self, uuid):

def getPool(self, uuid):

def queryResource(self, pool\_id):

def statisticStatus(self):

def allocate(self, pool\_id, count):

def allocatePort(self, pool\_id, ip, count):

def deallocate(self, pool\_id, ip, port\_lost):

### 物理架构管理

class ServiceManager(object):

"""

store domain node register info/status

not thread safe

"""

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

# #type group list

# #key = server type, value = {group:list of service name}

self.server\_groups = {}

# #key = service name, value = service

self.service\_map = {}

# #key = server uuid, value = [service name]

self.service\_in\_server = {}

# #key = service name, value = [whisper service]

self.whispers = {}

# #key = service name, value = {ip, port}

self.image\_services = {}

self.lock = threading.RLock()

def activeService(self, service):

def deactiveService(self, name):

def containsService(self, service\_name):

def getService(self, service\_name):

def loadService(self, service\_type, service\_list):

def queryService(self, service\_type, service\_group):

def updateService(self, name, disk\_type):

def queryServicesInServer(self, server\_id):

def getAllServiceType(self):

def queryServiceGroup(self, service\_type):

def updateWhisper(self, service\_name, whisper\_list):

def removeWhisper(self, service\_name):

def getAllWhisper(self):

def containsWhisper(self, service\_name):

def getWhisper(self, service\_name):

def updateImageService(self, image\_service):

def getAllImageService(self):

def statisticStatus(self):

### 服务管理

class ServiceManager(object):

"""

store domain node register info/status

not thread safe

"""

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

# #type group list

# #key = server type, value = {group:list of service name}

self.server\_groups = {}

# #key = service name, value = service

self.service\_map = {}

# #key = server uuid, value = [service name]

self.service\_in\_server = {}

# #key = service name, value = [whisper service]

self.whispers = {}

# #key = service name, value = {ip, port}

self.image\_services = {}

self.lock = threading.RLock()

def activeService(self, service):

def deactiveService(self, name):

def containsService(self, service\_name):

def getService(self, service\_name):

def loadService(self, service\_type, service\_list):

def queryService(self, service\_type, service\_group):

def updateService(self, name, disk\_type):

def queryServicesInServer(self, server\_id):

def getAllServiceType(self):

def queryServiceGroup(self, service\_type):

def updateWhisper(self, service\_name, whisper\_list):

def removeWhisper(self, service\_name):

def getAllWhisper(self):

def containsWhisper(self, service\_name):

def getWhisper(self, service\_name):

def updateImageService(self, image\_service):

def getAllImageService(self):

def statisticStatus(self):

### 状态管理

class StatusManager(object):

max\_timeout = 24

def \_\_init\_\_(self, logger\_name):

self.logger = logging.getLogger(logger\_name)

##key = room id, value = server room status

self.server\_room\_status = {}

##key = room id, value = counter

self.server\_room\_counter = {}

##key = rack id, value = server rack status

self.server\_rack\_status = {}

##key = rack id, value = counter

self.server\_rack\_counter = {}

##key = uuid, value = server status

self.server\_status = {}

##key = uuid, value = counter

self.server\_counter = {}

##key = uuid, value = host status

self.host\_status = {}

##key = uuid, value = counter

self.host\_counter = {}

##key = uuid, value = compute pool status

self.compute\_pool\_status = {}

##key = uuid, value = counter

self.compute\_pool\_counter = {}

self.system\_status = None

self.lock = threading.RLock()

def checkTimeout(self):

def updateServerStatus(self, uuid, status):

def updateHostStatus(self, status\_list):

def containsServerStatus(self, uuid):

def containsHostStatus(self, uuid):

def getServerStatus(self, uuid):

def removeServerStatus(self, uuid):

def getHostStatus(self, uuid):

def addHostStatus(self, status):

def removeHostStatus(self, host\_id):

def changeHostStatus(self, host\_id, status):

def updateComputePoolStatus(self, status\_list):

def containsComputePoolStatus(self, uuid):

def getComputePoolStatus(self, uuid):

def getAllComputePoolStatus(self):

def removeComputePoolStatus(self, uuid):

def updateServerRoomStatus(self, status\_list):

def containsServerRoomStatus(self, uuid):

def getServerRoomStatus(self, uuid):

def getAllServerRoomStatus(self):

def updateServerRackStatus(self, status\_list):

def containsServerRackStatus(self, uuid):

def getServerRackStatus(self, uuid):

def getAllServerRackStatus(self):

def updateSystemStatus(self, status):

def containsSystemStatus(self):

def getSystemStatus(self):

def statisticHostStatus(self):

def getAllServerStatus(self):

def getAllHostStatus(self):

# 接口定义

详见git上node\_client接口文档:https://github.com/zhicloud/doc.git

目录位置：interface/Control Server消息接口.xmind

# 参数配置

## 6.1 参数配置

平台服务模块启动之前需要配置/var/zhicloud/config/control\_server/node.conf，配置工作域、组播地址、组播端口。此外不需要做其他配置，程序自动完成配置文件维护。

node.conf配置如下：

[DEFAULT]

domain=zhicloud

node=control\_server\_000c299d5ddd

ip=

group\_ip=224.4.4.6

group\_port=5666

## 6.2 启停方式