|  |  |
| --- | --- |
| 文档编号 | IAAS-DDS 002 |
| 文档版本 | 1.0.4 |
| 拟制人 | 高翔 |
| 日期 | 2016.07.25 |

**公有云管理控制台**

**（网络1.2）**

**概要设计说明书**

变更说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **日期** | **版本** | **变更位置** | **变更说明** | **作者** |
| 2016.07.06 | 1.0.0 |  | 新建 | 高翔 |
| 2016.07.21 | 1.0.1 |  | 重做 | 高翔 |
| 2016.07.24 | 1.0.2 | 1. 用例图区分用户和管理员； 2. 私有网络增加绑定和解绑路由的场景逻辑； 3. 针对含有计费业务的模块，后付费添加对计费模块消息发送的的场景逻辑； 4. E-R图将类名变成表名； 5. 模块类图修改； 6. 业务层代码逻辑当中添加各个方法的返回值和参数类的设计； 7. 文档结尾添加各种场景问题； 8. 遗留问题：vpn底层状态还未获取全。。。 | 修改第二版 | 高翔 |
| 2016.07.25 | 1.0.3 | 2用例图重新构建，不同角色体现在同一张用例图中；  4云资源order表统一加入pay\_type字段，用作备查；  4.4.3加入对VPN状态字段vpn\_status的说明；  5.1.4所有与付费相关的云业务统一加入价格接口，计划根据前端用户更改配置事件触发调用；  6清理场景问题，仅遗留三个； | 修改第三版 | 高翔 |
| 2016.07.25 | 1.0.4 | 1概述重新定义；  5.1.4.1.2私有网络业务层中改写“变更私有网络状态”接口和“限制或恢复私有网络带宽”接口；  5.1.4.2.2负载均衡器业务层改写“变更负载均衡器状态”接口；  5.1.4.4.2VPN业务层中改写“变更VPN状态”接口； | 修改云资源欠费和续费的相关接口 | 高翔 |
| 2016.07.27 | 1.0.5 | 3流程图当中添加所有底层接口调用失败的处理，流程图当中添加回写资源id的步骤；  4表结构的调整：  5.1.4业务层伪代码逻辑； |  |  |
| 2016.08.01 | 1.0.6 | 5.1.4.3子网控制器加入获取不同类型子网列表的接口； |  |  |

目　　录

[变更说明 2](#_Toc15144)

[目　　录 3](#_Toc4472)

[1 概述 4](#_Toc18941)

[2 模块改造（仅涉及改造部分用例） 5](#_Toc30596)

[2.1私有网络改造用例图 5](#_Toc3422)

[2.2负载均衡改造用例图 5](#_Toc17775)

[2.3VPN改造用例图 6](#_Toc29072)

[3 改造相关的业务分析 7](#_Toc27807)

[3.1 私有网络 7](#_Toc18379)

[3.1.1 购买私有网络网 7](#_Toc12593)

[3.1.2 创建私有网络网（包年包月） 8](#_Toc9329)

[3.1.3 变更私有网络配置（带宽） 9](#_Toc7116)

[3.1.4 编辑私有网络配置（包年包月带宽） 10](#_Toc25620)

[3.1.5 续费私有网络 12](#_Toc23327)

[3.1.6 激活续费后的私有网络 13](#_Toc13703)

[3.1.7 删除私有网络 14](#_Toc6689)

[3.1.8 创建子网 15](#_Toc14732)

[3.2 端口映射 16](#_Toc14739)

[3.2.1端口映射查询列表流程 16](#_Toc3293)

[3.2.2端口映射创建流程 17](#_Toc3666)

[3.2.3端口映射编辑流程 17](#_Toc30632)

[3.2.4端口映射删除流程 18](#_Toc4987)

[3.3 负载均衡 18](#_Toc5552)

[3.3.1购买负载均衡 19](#_Toc13940)

[3.3.2创建负载均衡（包年包月） 19](#_Toc9395)

[3.3.3变更负载均衡配置（最大连接数） 21](#_Toc21575)

[3.3.4编辑负载均衡配置（包年包月最大连接数） 22](#_Toc21190)

[3.3.5续费负载均衡 24](#_Toc31494)

[3.3.6激活续费后的负载均衡 25](#_Toc29890)

[3.3.7删除负载均衡 26](#_Toc22278)

[3.4 VPN 27](#_Toc3329)

[3.4.1 VPN查询列表流程 27](#_Toc13105)

[3.4.2 VPN购买流程 28](#_Toc3307)

[3.4.3 VPN创建流程（包年包月） 29](#_Toc27851)

[3.4.4 VPN编辑配置流程 31](#_Toc20531)

[3.4.5 VPN删除流程 32](#_Toc32335)

[3.4.6 VPN续费流程 33](#_Toc1714)

[3.4.7 VPN激活续费的流程 34](#_Toc8790)

[4 改造数据库分析 35](#_Toc10804)

[4.1私有网络数据库 36](#_Toc22187)

[4.1.1调整cloud\_network表 36](#_Toc18372)

[4.1.1新增cloudorder\_network表 36](#_Toc2288)

[4.1.1调整cloud\_subnetwork表 37](#_Toc24626)

[4.2新增cloud\_portmapping表 38](#_Toc16775)

[4.3 负载均衡 38](#_Toc488)

[4.3.1调整cloud\_ldpool表 38](#_Toc1687)

[4.3.2新增cloudorder\_ldpool表 39](#_Toc3657)

[4.4 VPN 40](#_Toc16484)

[4.4.1 E-R图 40](#_Toc28466)

[4.4.2新增 cloud\_vpncon表 40](#_Toc29019)

[4.4.3新增 cloud\_vpnservice表 42](#_Toc25939)

[4.4.4新增 cloud\_ikepolicy表 42](#_Toc26493)

[4.4.5新增 cloud\_ipsecpolicy表 43](#_Toc7801)

[4.4.6新增 cloudorder\_vpn 表 43](#_Toc31240)

[5 后台设计 45](#_Toc31490)

[5.1 类设计 46](#_Toc15765)

[5.1.1 模块类图 46](#_Toc25612)

[5.1.2 Model设计 46](#_Toc10757)

[5.1.3 模块类说明 52](#_Toc24620)

[5.1.4 业务层代码逻辑改动 53](#_Toc18674)

[6相关设计问题的整理 77](#_Toc17057)

# 概述

网络1.2改造需求的主要原因是：

1. 为了满足客户对于多网卡的业务需求，并规避多网卡绑定公网IP导致路由无法正确分发数据的错误，而将原本平台下的子网分成了受管子网和自管子网；
2. 端口映射和VPN功能模块引入；
3. 兼容用户中心1.0的云资源订单付费的需求；

在这些改变之下，使得相关模块必须也做出一系列的改动调整。

具体涉及到的模块有：私有网络VPC模块，负载均衡模块，弹性公网IP模块以及此次改动新加入的VPN模块。

# 模块改造（仅涉及改造部分用例）

## 2.1私有网络改造用例图



## 2.2负载均衡改造用例图



## 2.3VPN改造用例图



# 改造相关的业务分析

## 私有网络

### **购买私有网络网**



### **创建私有网络网**



### **变更私有网络配置（带宽）**



### **编辑私有网络配置（包年包月带宽）**



### **续费私有网络**



### **激活续费后的私有网络**



### **删除私有网络**



### **创建子网**

创建子网的流程图改动：



## 端口映射

## 3.2.1端口映射查询列表流程



### 3.2.2端口映射创建流程



### 3.2.3端口映射编辑流程



### 3.2.4端口映射删除流程



## 负载均衡

### 3.3.1购买负载均衡



### 3.3.2创建负载均衡



### 3.3.3变更负载均衡配置（最大连接数）



### 3.3.4编辑负载均衡配置（包年包月最大连接数）



### 3.3.5续费负载均衡



### 3.3.6激活续费后的负载均衡



### 3.3.7删除负载均衡



## VPN

### 3.4.1 VPN查询列表流程



### 3.4.2 VPN购买流程



### 3.4.3 VPN创建流程（包年包月）



### 3.4.4 VPN编辑配置流程



### 3.4.5 VPN删除流程



### 3.4.6 VPN续费流程



### 3.4.7 VPN激活续费的流程



# 改造数据库分析

## 4.1私有网络数据库

### 4.1.1调整cloud\_network表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_network** | | | | | |
| **功能说明** | | | **存储网络数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | net\_id | | 网络id | VC(100) |  | N | PK |
| 2 | net\_name | | 网络名称 | VC(100) |  | N |  |
| 3 | create\_name | | 创建人 | VC(100) | 登录用户 |  |  |
| 4 | create\_time | | 创建时间 | DT |  | N |  |
| 5 | prj\_id | | 项目id | VC(100) |  | N | FK |
| 6 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 7 | net\_status | | 状态 | VC(20) |  | N |  |
| 8 | admin\_stateup | | 管理员状态 | C(1) | 1代表 是，0代表 否 |  |  |
| 9 | is\_shared | | 共享 | C(1) | 1代表 是，0代表 否 |  |  |
| 10 | router\_external | | 是否外网 | C(1) | 1代表 是，0代表 否 |  |  |
| 11 | pay\_type | | 计费模式 | C(1) | 1代表包年包月，  2代表按需按量 | N |  |
| 12 | charge\_state | | 计费状态 | C(1) | 0代表正常，  1代表余额不足，  2代表已到期，  3代表停服务 | N |  |
| 13 | end\_time | | 到期时间 | DT | 针对包年包月  默认null |  |  |
| 14 | reserve1 | | 保留字段1 | VC(100) |  |  |  |
| 15 | reserve2 | | 保留字段2 | VC(100) |  |  |  |
| 16 | reserve3 | | 保留字段3 | VC(100) |  |  |  |
| 17 | reserve4 | | 保留字段4 | VC(100) |  |  |  |
| 18 | reserve5 | | 保留字段5 | VC(100) |  |  |  |
| **索引名称** | | | | **Unique** | **索引字段** | | |
| Idx\_networkprj | | | |  | prj\_id | | |

### 4.1.2新增cloudorder\_network表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloudorder\_network** | | | | | |
| **功能说明** | | | **存储网络订单数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | ordernetwork\_id | | id | VC(32) |  | N | PK |
| 2 | order\_no | | 订单编号 | CHAR(18) |  | N | FK |
| 3 | order\_type | | 订单类型 | CHAR(1) | 0代表新购，  1代表续费，  2代表升级。 |  |  |
| 4 | buy\_cycle | | 购买周期 | INT | 代表月数 |  |  |
| 5 | pay\_type | | 计费模式 | C(1) | 1代表包年包月，  2代表按需按量 | N |  |
| 6 | price | | 总价 | NUMERIC(16,3) |  |  |  |
| 7 | net\_id | | 网络id | VC(100) |  | N | FK |
| 8 | net\_name | | 网络名称 | VC(100) |  | N |  |
| 9 | rate | | 路由带宽 | INT(9) |  | N |  |
| 10 | create\_name | | 创建人 | VC(100) | 登录用户 |  |  |
| 11 | create\_time | | 创建时间 | DT |  | N |  |
| 12 | prj\_id | | 项目id | VC(100) |  | N | FK |
| 13 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 14 | customer\_id | | 客户id | VC(32) |  | N |  |

### 4.1.3调整cloud\_route表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_route** | | | | | |
| **功能说明** | | | **存储虚拟化层路由模块信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | route\_id | | 路由id | VC(100) |  | N | PK |
| 2 | route \_name | | 路由名称 | VC(100) |  |  |  |
| 3 | create\_name | | 创建人 | VC(100) |  |  |  |
| 4 | prj\_id | | 项目id | VC(100) |  | N | FK |
| 5 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 6 | create\_time | | 创建时间 | DT |  |  |  |
| 7 | route\_status | | 路由状态 | VC(50) |  |  |  |
| 8 | net\_id | | 外网网关的id | VC(32) | 设置的网关id（外网id） |  |  |
| 9 | rate | | 带宽宽度 | N(9) | 带宽限速本地DB中1代表底层对应字段的125000 |  |  |
| 10 | qos\_id | | 带宽id | VC(100) |  |  |  |
| 11 | reserve1 | | 保留字段1 | VC(100) |  |  |  |
| 12 | reserve2 | | 保留字段2 | VC(100) |  |  |  |
| 13 | reserve3 | | 保留字段3 | VC(100) |  |  |  |
| 14 | default\_queue\_id | | QOS默认的queue的id | VC(100) |  |  |  |
| 15 | filter\_queue\_id | | 新创建的queue队列id | VC(100) |  |  |  |
| 16 | network\_id | | 所属私有网络 | VC(100) |  |  |  |
| 17 | gateway\_ip | | 网关ip | VC(100) |  |  |  |
| 18 | rate\_old | | 限服前的带宽 | N(9) | 限制服务前的带宽 |  |  |
| **索引名称** | | | | **Unique** | **索引字段** | | |
| cloud\_routeprj | | | |  | prj\_id | | |

### 4.1.4调整cloud\_subnetwork表

新增子网类型subnet\_type字段char（1）用于区分子网资源是受管子网or自管子网：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_subnetwork** | | | | | |
| **功能说明** | | | **存储子网数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | subnet\_id | | 子网id | VC(100) |  | N | PK |
| 2 | subnet\_name | | 子网名称 | VC(100) |  | N |  |
| 3 | create\_name | | 创建人 | VC(50) | 登录用户 |  |  |
| 4 | create\_time | | 创建时间 | DT |  | N |  |
| 5 | prj\_id | | 项目id | VC(100) |  | N | FK |
| 6 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 7 | net\_id | | 网络id | VC(100) |  | N | FK |
| 8 | route\_id | | 路由id | VC(100) |  |  | FK |
| 9 | ip\_version | | Ip版本 | C(1) | 4或6 | N |  |
| 10 | cidr | | 网络地址 | VC(100) |  | N |  |
| 11 | gateway\_ip | | 网关ip | VC(100) |  | N |  |
| 12 | is\_shared | | 共享 | C(1) | 1代表 是，0代表 否 |  |  |
| 13 | pooldata | | 网络地址池 | VC(100) |  | N |  |
| 14 | is\_forbiddengw | | 是否禁用网关 | C(1) | 1代表 是，0代表 否 |  |  |
| 15 | reserve1 | | 保留字段 | VC(32) |  |  |  |
| 16 | reserve2 | | 保留字段2 | VC(100) |  |  |  |
| 17 | reserve3 | | 保留字段3 | VC(100) |  |  |  |
| 18 | reserve4 | | 保留字段4 | VC(100) |  |  |  |
| 19 | reserve5 | | 保留字段5 | VC(100) |  |  |  |
| 20 | subnet\_type | | 子网类型 | C(1) | 1代表受管子网，  0代表自管子网，  默认是1受管子网 | N |  |
| **索引名称** | | | | **Unique** | **索引字段** | | |
| Cloud\_subnetworkprj | | | |  | prj\_id | | |
|  | | | |  |  | | |

## 4.2新增cloud\_portmapping表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_portmapping** | | | | | |
| **功能说明** | | | **存储端口映射数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | pm\_id | | id | VC(100) |  | N | PK |
| 2 | protocol | | 协议 | VC(10) | TCP or UDP |  |  |
| 3 | resource\_ip | | 源IP | VC(50) | 路由网关IP | N |  |
| 4 | resource\_port | | 源端口 | VC(10) | 路由端口 |  |  |
| 5 | destiny\_id | | 目的资源id | VC(32) | 云主机ID |  |  |
| 6 | destiny\_ip | | 目的IP | VC(50) | 云主机IP | N |  |
| 7 | destiny\_port | | 目的端口 | VC(10) | 云主机端口 |  |  |
| 8 | net\_id | | 所属网络id | VC(32) | 所属私有网络id | N | FK |

## 4.3 负载均衡

### 4.3.1调整cloud\_ldpool表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_ldpool** | | | | | |
| **功能说明** | | | **存储虚拟化层负载均衡资源池模块信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | pool\_id | | 资源池id | VC(100) |  | N | PK |
| 2 | pool\_name | | 资源池名称 | VC(100) |  |  |  |
| 3 | prj\_id | | 项目Id | VC(100) |  | N | FK |
| 4 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 5 | pool\_description | | 资源池描述 | VC(255) |  |  |  |
| 6 | pool\_provider | | 资源池提供者 | VC(255) |  |  |  |
| 7 | subnet\_id | | 子网id | VC(32) |  | N | FK |
| 8 | pool\_protocol | | 协议类型 | VC(32) | HTTP/HTTPS/TCP三种 | N |  |
| 9 | lb\_method | | 负载均衡方式 | VC(32) | ROUND\_ROBIN/  LEAST\_CONNECTIONS/ SOURCE\_IP | N |  |
| 10 | pool\_status | | 资源池状态 | VC(16) |  | N |  |
| 11 | admin\_stateup | | 资源池是否是管理员 | C(1) | 1：管理员；  0：非管理员 | N |  |
| 12 | create\_name | | 创建人 | VC(100) |  |  |  |
| 13 | create\_time | | 创建时间 | DT |  |  |  |
| 14 | pay\_type | | 付款方式 | C(1) | 1：包年包月；  2：按需按量； |  |  |
| 15 | end\_time | | 到期时间 | DT |  |  |  |
| 16 | charge\_state | | 付费状态 | C(1) | 0：正常；  1：余额不足；  2：已到期；  3：停服务； |  |  |
| 17 | reserve1 | | 保留字段1 | VC(100) |  |  |  |
| 18 | reserve2 | | 保留字段2 | VC(100) |  |  |  |
| 19 | reserve3 | | 保留字段3 | VC(100) |  |  |  |
| 20 | reserve4 | | 保留字段4 | VC(100) |  |  |  |
| **索引名称** | | | | **Unique** | **索引字段** | | |
| cloud\_ldpoolprj | | | |  | prj\_id | | |

### 4.3.2新增cloudorder\_ldpool表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloudorder\_ldpool** | | | | | |
| **功能说明** | | | **存储虚拟化层负载均衡资源池订单信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | orderpool\_id | | Id | VC(32) |  | N | PK |
| 2 | order\_no | | 订单编号 | VC(18) |  | N | FK |
| 3 | order\_type | | 订单类型 | CHAR(1) | 0：新购；  1：续费；  2：升级； |  |  |
| 4 | buy\_cycle | | 购买周期 | INT | 代表月数 |  |  |
| 5 | pay\_type | | 付款方式 | C(1) | 1：包年包月；  2：按需按量； |  |  |
| 6 | price | | 总价 | NUMERIC(16,3) |  |  |  |
| 7 | pool\_id | | 资源池id | VC(100) |  | N |  |
| 8 | pool\_name | | 资源池名称 | VC(100) |  |  |  |
| 9 | connection\_limit | | 最大连接数 | Decimal(11,0) |  | N |  |
| 9 | prj\_id | | 项目Id | VC(100) |  | N | FK |
| 10 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 11 | pool\_description | | 资源池描述 | VC(255) |  |  |  |
| 12 | pool\_provider | | 资源池提供者 | VC(255) |  |  |  |
| 13 | subnet\_id | | 子网id | VC(100) |  | N | FK |
| 14 | pool\_protocol | | 协议类型 | VC(32) | HTTP/HTTPS/TCP三种 | N |  |
| 15 | lb\_method | | 负载均衡方式 | VC(32) | ROUND\_ROBIN/  LEAST\_CONNECTIONS/ SOURCE\_IP | N |  |
| 16 | create\_name | | 创建人 | VC(100) |  |  |  |
| 17 | create\_time | | 创建时间 | DT |  |  |  |
| 18 | customer\_id | | 客户id | VC(32) |  | N |  |

## 4.4 VPN

### 4.4.1 E-R图



### 4.4.2新增 cloud\_vpnconn表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_****vpnconn** | | | | | |
| **功能说明** | | | **存储****vpn管道数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | vpn\_id | | id | VC(100) |  | N | PK |
| 2 | vpn\_name | | VPN名字 | VC(50) |  |  |  |
| 3 | vpn\_status | | 状态 | VC(50) | PENDING\_CREATE：  处于创建状态或创建成功状态；  ACTIVE：  该VPN有ipsec连接建立成功；  DOWN：  该VPN处于down状态，相关的服务没有生效； | N |  |
| 4 | peer\_address | | 对端网址 | VC(100) |  |  |  |
| 5 | peer\_id | | 对端网络id | VC(50) |  |  |  |
| 6 | peer\_cidrs | | 对端网段 | VC(2000) | 例如“192.168.2.0/24,192.168.3.0/24”最多2000字符 |  |  |
| 7 | local\_cidrs | | 本端网段 | VC(50) |  |  |  |
| 9 | mtu | | 最大传输单元 | INT | 取值范围[68,999999999] |  |  |
| 10 | dpd\_action | | 失效处理 | C(1) | 可选：  0：保留；  1：清除；  2：禁用；  3：重启；  默认保留。 | N |  |
| 11 | dpd\_interval | | 检测间隔（秒） | INT | 默认30，  小于等于999999999 | N |  |
| 12 | dpd\_timeout | | 超时（秒） | INT | 默认1500  小于等于999999999 | N |  |
| 14 | psk\_key | | 预共享密钥 | VC(80) | 最大80字符，  不限制字符类型 |  |  |
| 15 | initiator | | 发起状态 | C(1) | 可选  0：只应答；  1：双向；  默认双向 | N |  |
| 16 | vpnservice\_id | | 关联VPNService | VC(100) |  | N | FK |
| 17 | ipsec\_id | | 关联IPSecPolicy | VC(100) |  | N | FK |
| 18 | ike\_id | | 关联IKEPolicy | VC(100) |  | N | FK |
| 19 | create\_time | | 创建时间 | DT |  |  |  |
| 20 | pay\_type | | 付款方式 | C(1) | 1：包年包月；  2：按需按量； |  |  |
| 20 | end\_time | | 到期时间 | DT |  |  |  |
| 21 | charge\_state | | 付费状态 | C(1) | 0：正常；  1：余额不足；  2：已到期；  3：停服务； | N |  |

### 4.4.3新增 cloud\_vpnservice表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloud\_vpnservice** | | | | | |
| **功能说明** | | | **存储vpnservice数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | vpnservice\_id | | id | VC(100) |  | N | PK |
| 2 | prj\_id | | 项目id | VC(100) |  | N | FK |
| 3 | vpnservice\_name | | VPN名称 | VC(50) |  |  |  |
| 4 | subnet\_id | | 本端子网id | VC(32) |  |  |  |
| 5 | route\_id | | 对端路由id | VC(200) | IP地址、e-mail、键ID或者FQDN.支持输入多段？ |  |  |

### 4.4.4新增 cloud\_ikepolicy表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloudike\_policy** | | | | | |
| **功能说明** | | | **存储IKEPolicy策略数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | ike\_id | | id | VC(100) |  | N | PK |
| 2 | encryption\_algorithm | | 加密算法 | VC(10) | 可选：  3DES，  AES-128，  AES-192，  AES-256。  默认3DES | N |  |
| 3 | version | | 版本 | VC(10) | 可选：  V1，  V2。  默认V1 | N |  |
| 4 | auth\_algorithm | | 认证算法 | VC(10) | Sha1 | N |  |
| 5 | nogotiation\_mode | | 协商模式 | VC(10) | main | N |  |
| 6 | lifetime\_value | | 生存周期（秒） | INT | 默认86400，  [60,999999999] | N |  |
| 7 | dh\_algorithm | | DH算法 | VC(10) | 可选：  GROUP2，  GROUP5，  GROUP14。  默认GROUP2 | N |  |

### 4.4.5新增 cloud\_ipsecpolicy表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloudIpsec\_policy** | | | | | |
| **功能说明** | | | **存储IPSec策略数据信息** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | ipsec\_id | | id | VC(100) |  | N | PK |
| 2 | encryption\_algorithm | | 加密算法 | VC(10) | 可选：  3DES,  AES-128,  AES-192,  AES-256.  默认3DES | N |  |
| 3 | transform\_protocol | | 传输协议 | VC(10) | 可选：  ESP,  AH.  默认ESP | N |  |
| 4 | auth\_algorithm | | 认证算法 | VC(10) | sha1 | N |  |
| 5 | encapsulation\_mode | | 封装模式 | VC(10) | 隧道模式 | N |  |
| 6 | lifetime\_value | | 生存周期 | INT | 默认3600，[60,999999999] | N |  |
| 7 | dh\_algorithm | | DH算法 | VC(10) | 可选：  GROUP2,  GROUP5,  GROUP14.  默认GROUP2 | N |  |

### 4.4.6新增 cloudorder\_vpn 表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **数据表名称** | | **cloudorder\_vpn** | | | | | |
| **功能说明** | | | **存储vpn订单数据表** | | | | |
| **序号** | **字段名称** | | **中文名称** | **类型** | **说明** | **非空** | **业务规则** |
| 1 | ordervpn\_id | | id | VC(32) |  | N | PK |
| 2 | order\_no | | 订单编号 | VC(18) |  | N | FK |
| 3 | order\_type | | 订单类型 | CHAR(1) | 0代表新购，  1代表续费，  2代表升级。 | N |  |
| 4 | buy\_cycle | | 购买周期 | INT | 月数 |  |  |
| 5 | pay\_type | | 计费模式 | C(1) | 1代表包年包月，  2代表按需按量 | N |  |
| 6 | price | | 总价 | NUMERIC(16,3) |  |  |  |
| 7 | vpn\_id | | Vpnid | VC(100) |  | N | FK |
| 8 | vpnservice\_id | |  | VC(100) |  | N | FK |
| 9 | ike\_id | |  | VC(100) |  | N | FK |
| 10 | ipsec\_id | |  | VC(100) |  | N | FK |
| 8 | prj\_id | | 所属项目id | VC(100) |  | N | FK |
| 9 | dc\_id | | 数据中心id | VC(100) |  | N | FK |
| 10 | vpn\_name | | 名称 | VC(50) |  |  |  |
| 10 | Subnet\_id | | 本端子网 | VC(32) |  |  |  |
| 11 | local\_cidrs | | 本端网段 | VC(50) |  |  |  |
| 12 | Router\_id | | 关联路由 | VC(200) | IP地址、email、键ID或者FQDN。支持输入多段 |  |  |
| 13 | psk\_key | | 预共享密钥 | VC(80) | 最大80字符  不限制字符输入类型 |  |  |
| 14 | peer\_address | | 对端网址 | VC(100) |  |  |  |
| 15 | peer\_id | | 对端网络id | VC(32) |  |  |  |
| 16 | peer\_cidrs | | 对端网段 | VC(2000) | 例如“192.168.2.0/24,192.168.3.0/24”最多2000字符 |  |  |
| 17 | ike\_encryption | | 加密算法 | VC(10) | 可选：  3DES，  AES-128，  AES-192，  AES-256。  默认3DES |  |  |
| 18 | ike\_version | | 版本 | VC(10) | 可选：  V1，  V2。  默认V1 |  |  |
| 19 | ike\_auth | | 认证算法 | VC(10) | Sha1 |  |  |
| 20 | ike\_nogotiation | | 协商模式 | VC(10) | main |  |  |
| 21 | ike\_lifetime | | 生存周期（秒） | INT | 默认86400，[60,999999999] |  |  |
| 22 | ike\_dh | | DH算法 | VC(100) | 可选：  GROUP2，  GROUP5，  GROUP14.  默认GROUP2 |  |  |
| 23 | IPsec\_encryption | | 加密算法 | VC(10) | 可选：  3DES,  AES-128,  AES-192,  AES-256.  默认3DES | N |  |
| 24 | IPsec\_protocol | | 传输协议 | VC(10) | 可选：  ESP，  AH  默认ESP。 | N |  |
| 25 | IPsec\_auth | | 认证算法 | VC(10) | Sha1 | N |  |
| 26 | IPsec\_encapsulation | | 封装模式 | VC(10) | 隧道模式 | N |  |
| 27 | IPsec\_lifetime | | 生命周期（秒） | INT | 默认3600，[60,999999999] | N |  |
| 28 | IPsec\_dh | | DH算法 | VC(10) | 可选：  GROUP2，  GROUP5，  GROUP14.  默认GROUP2 |  |  |
| 29 | con\_mtu | | 最大传输单元 | INT | 取值范围[68,999999999] | N |  |
| 30 | dpd\_action | | 失效处理 | C(1) | 可选：  0保留，  1清除，  2禁用，  3重启。  默认保留 | N |  |
| 31 | dpd\_interval | | 检测间隔（秒） | INT | 默认30，  小于等于999999999 | N |  |
| 32 | dpd\_timeout | | 超时（秒） | INT | 默认1500，  小于等于999999999 | N |  |
| 33 | initiator | | 发起状态 | C(1) | 可选：  0只应答，  1双向。  默认双向 | N |  |
| 34 | create\_time | | vpn创建时间 | DT |  |  |  |

# 后台设计

## 类设计

### 模块类图

#### 端口映射模块类图：



#### VPN模块类图：



### Model设计

#### 5.1.2.1 私有网络 Model设计

BaseCloudNetWork {

添加属性：

**Private** String payType; //付款方式：1预付费，2后付费

**Private** String endTime; //到期时间

**Private** String chargeState; //付费状态：0正常，1余额不足，2已欠费

}

BaseCloudOrderNetWork {

**Private** String orderNetWorkId; //id

**Private** String orderNo; //订单编号

**Private** String orderType; //订单类型

**Private int** buyCycle; //购买周期

**Private** bigdecimal price; //价格

**Private** String netId; //网络id

**Private** String netName; //网络名称

**Private int** rate; //路由带宽

**Private** String createName; //创建人

**Private** String createTime; //创建时间

**Private** String prjId; //项目id

**Private** String dcId; //数据中心id

**Private** String customerId; //客户id

}

BaseCloudSubNetWork {

添加属性：

**Private int** subnetType; //子网类型：1受管子网，0自管子网

}

BaseCloudRoute {

**private** String routeId;

**private** String routeName;

**private** String createName;

**private** Date createTime;

**private** String dcId; //数据中心id

**private** String prjId; //所在的项目ID

**private** String routeStatus;

**private** String netId;

**private** **int** rate=0;

**private** String qosId;

**private** String defaultQueueId;

**private** String filterQueueId;

**private** String netWorkId;

**private** String gatewayIp;

添加属性：

**private int** rateOld=0; //超期限制服务前的带宽

}

#### 5.1.2.2 端口映射 Model设计

PortMapping {

**Private** String id; //id

**Private** String protocol; //协议

**Private** String resourceIp; //源IP

**Private** String resourcePort; //源端口

**Private** String destinyId; //目的ID

**Private** String destinyIp; //目的IP

**Private** String destinyPort; //目的端口

**Private** String netId; //所属私有网络iD

}

#### 5.1.2.3 负载均衡 Model设计

BaseCloudLdPool {

......

添加属性：

Private String payType; //付款方式：1预付费，2后付费

Private Date endTime; //到期时间

Private String resourceState; //付费状态：0正常，1余额不足，2已欠费

}

BaseCloudOrderLdPool {

Private String orderPoolId; //id

Private String orderNo; //订单编号

Private String orderType; //订单类型

Private int buyCycle; //购买周期

Private bigdecimal price; //价格

Private String poolId; //资源池

Private String poolName; //资源池名字

Private Long connectionLimit; //最大连接数

Private String prjId; //项目id

Private String dcId; //数据中心id

Private String poolDescription; //资源池描述

Private String poolProvider; //资源池提供者

Private String subnetId; //子网id

Private String poolProtocol; //协议类型

Private String lbMethod; //负载均衡方式

Private String createName; //创建人

Private Date createTime; //创建时间

Private String customerId; //客户id

}

#### 5.1.2.4 VPN Model设计

##### 5.1.2.4.1 BaseCloudIPSecPolicy

BaseCloudIPSecPolicy {

Private String ipSecId; //IPSec策略的id

Private String encryption; //加密算法

Private String transform; //传输协议

Private String authAlgorithm; //认证算法

Private String encapsulation; //封装模式

Private Long lifetimeV; //生命周期（秒）

Private String DHAlgorithm; //DH算法

}

##### 5.1.2.4.2 BaseCloudIKEPolicy

BaseCloudIKEPolicy {

Private String ikeId; //IKE策略的Id

Private String encryption; //加密算法

Private String version; //版本

Private String authAlgorithm; //认证算法

Private String nogotiation; //协商模式

Private Long lifetimeV; //生存周期（秒）

Private String DHAlgorithm; //DH算法

}

##### 5.1.2.4.3 BaseCloudVPN

BaseCloudVPN {

Private String vpnId; //VPN的Id

Private String prjId; //项目id

Private String name; //VPN名称

Private String subnetId; //本端子网id

Private String routeId; //关联路由id

}

##### 5.1.2.4.4 BaseCloudVPNConnection

BaseCloudVPNConnection {

Private String vpnConnId; //VPNConnection的Id

Private String name; //VPNConnection的名称

Private String status; //VPN的状态

Private String peerAddress; //对端网址

Private String peerId; //对端网络id

Private String peerCidrs; //对端网段

Private String localCidrs; //本端网段

Private String mtu; //最大传输单元

Private String dpdAction; //失效处理

Private Long dpdInterval; //检测间隔（秒）

Private Long dpdTimeout; //超时（秒）

Private String pskKey; //预共享密钥

Private String initiator; //发起状态

Private String vpnServiceId; //关联VPNService

Private String ipSecPolicyId; //关联IPSecPolicy

Private String ikePolicyId; //关联IKEPolicy

Private String createTime; //创建时间

Private String payType; //付款方式

Private String endTime; //到期时间

Private String chargeState; //付费状态

}

##### 5.1.2.4.5 BaseCloudOrderVPN

BaseCloudOrderVPN {

Private String orderVPNId; //id

Private String orderNo; //订单编号

Private String orderType; //订单类型：0新购，1续费，2升级

Private int buyCycle; //购买周期（月数）

Private bigdecimal price; //价格

Private String prjId; //所属项目id

Private String dcId; //数据中心id

Private String vpnName; //vpn名称

Private String subnetId; //本端子网id

Private String localCidrs; //本端网段

Private String routerId; //关联路由

Private String pskKey; //预共享密钥

Private String peerAddress; //对端网址

Private String peerId; //对端网络id

Private String peerCidrs; //对端网段

Private String ikeEncryption; //ike加密算法

Private String ikeVersion; //版本V1V2，默认V1

Private String ikeAuth; //认证算法

Private String ikeNogotiation; //协商模式

Private String ikeLifetime; //生存周期（秒）

Private String ikeDh; //dh算法

Private String ipSecEncryption; //加密算法

Private String ipSecProtocol; //传输协议

Private String ipSecAuth; //认证算法

Private String ipSecEncapsulation;//封装模式

Private String ipSecLifeTime; //生存周期（秒）

Private String ipSecDh; //dh算法

Private Long conMtu; //最大传输单元

Private String dpdAction; //失效处理

Private Long dpdInterval; //检测间隔

Private Long dpdTimeout; //超时（秒）

Private String vpnConnId;

Private String vpnServiceId;

Private String ikeId;

Private String ipSecId;

Private String initiator; //发起状态

Private String createTime; //创建时间

Private String payType; //付费方式：1预付费2后付费

}

##### 5.1.2.4.6 VPNModel（页面交互model）

VPNModel {

Private String id; //id

Private String vpnName; //vpn名称

Private String prjId; //所属项目id

Private String vpnStatus; //vpn状态

Private String subnetId; //本端子网id

Private String localCidrs; //本端网段

Private String routerId; //关联路由

Private String pskKey; //预共享密钥

Private String peerAddress; //对端网址

Private String peerId; //对端网络id

Private String peerCidrs; //对端网段

Private String ikeEncryption; //ike加密算法

Private String ikeVersion; //版本V1V2，默认V1

Private String ikeAuth; //认证算法

Private String ikeNogotiation; //协商模式

Private Long ikeLifetime; //生存周期（秒）

Private String ikeDh; //dh算法

Private String ipSecEncryption; //加密算法

Private String ipSecProtocol; //传输协议

Private String ipSecAuth; //认证算法

Private String ipSecEncapsulation;//封装模式

Private String ipSecLifeTime; //生存周期（秒）

Private String ipSecDh; //dh算法

Private Long conMtu; //最大传输单元

Private String dpdAction; //失效处理

Private Long dpdInterval; //检测间隔

Private Long dpdTimeout; //超时（秒）

Private String initiator; //发起状态

Private String createTime; //创建时间

Private String vpnServiceId;

Private String vpnIKEPolicyId;

Private String vpnIpSecPolicyId;

Private String chargeState; //付费状态

Private String payType; //付费方式：1预付费2后付费

}

### 模块类说明

|  |  |  |  |
| --- | --- | --- | --- |
| 类名 | 包名 | 说明 | 描述 |
| NetWorkController | com.eayun.virtualization.controller | 私有网络控制层 |  |
| NetWorkService | com.eayun.virtualization.service | 私有网络业务接口 |  |
| NetWorkServiceImpl | com.eayun.virtualization.service.impl | 私有网络业务实现 |  |
| SubNetWorkService | com.eayun.virtualization.service | 子网业务接口 |  |
| SubNetWorkServiceImpl | com.eayun.virtualization.service.impl | 子网业务实现 |  |
| PoolController | com.eayun.virtualization.controller | 负载均衡器控制层 |  |
| PoolService | com.eayun.virtualization.service | 负载均衡器业务层接口 |  |
| PoolServiceImpl | com.eayun.virtualization.service.impl | 负载均衡器业务层实现 |  |
| PortMappingController | com.eayun.virtualization.controller | 端口映射控制层 |  |
| PortMappingService | com.eayun.virtualization.service | 端口映射业务接口 |  |
| PortMappingServiceImpl | com.eayun.virtualization.service.impl | 端口映射业务实现 | 实现PortMappingService |
| PortMappingDao | com.eayun.virtualization.dao | 端口映射持久化层接口 |  |
| VPNController | com.eayun.virtualization.controller | VPN控制层 |  |
| VPNService | com.eayun.virtualization.serivce | VPN业务接口 |  |
| VPNServiceImpl | com.eayun.virtualization.service.impl | VPN业务接口实现 |  |
| VPNDao | Com.eayun.virtualization.dao | VPN 持久化层接口 |  |
| VPNJob | Com.eayun.virtualization.job | VPN计划任务 |  |

### 业务层代码逻辑改动

#### 5.1.4.1私有网络

##### 5.1.4.1.1私有网络控制器

NetWorkController (/cloud/network)

1. buyNetWork(/buy)

|  |  |
| --- | --- |
| 说明 | 购买私有网络 |
| URL | /cloud/network/buy.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderNetWork 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

1. changeNetWork(/change)

|  |  |
| --- | --- |
| 说明 | 变更配置私有网络 |
| URL | /cloud/network/change.do |
| 方式 | POST |
| Post数据 | {  ...//CloudOrderNetWork 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

1. getPriceOfNetWork(/getprice)

|  |  |
| --- | --- |
| 说明 | 生成私有网络界面价格 |
| URL | /cloud/network/getprice.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderNetWork 类  } |
| 返回Json | 成功：  {  "code":"000000"，  "resultData":"120"  }  失败：  {"code":"010120"} |

1. renewNetWork(/renew)

|  |  |
| --- | --- |
| 说明 | 续费私有网络 |
| URL | /cloud/network/renew.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderNetWork 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

Public class NetWorkController extends BaseController {

购买私有网络接口：

Public String buyNetWork(

HttpServletRequest request,

@RequestBody CloudOrderNetWork cloudOrderNetWork

){

调用私有网络业务层中的购买私有网络接口；

}

变更配置私有网络接口：

Public String changeNetWork(

HttpServletRequest request,

@RequestBody CloudNetWork cloudNetWork

){

调用私有网络业务层中的修改配置购买私有网络接口；

}

续费私有网络接口：

Public String renewNetWork(

HttpServletRequest request,

@RequestBody CloudOrderNetWork cloudOrderNetWork

){

调用私有网络业务层中的续费私有网路接口；

}

生成界面价格接口：

Public String getPriceOfNetWork(

HttpServletRequest request,

@RequestBody CloudOrderNetWork cloudOrderNetWork

){

调用私有网络业务层中生成界面价格接口；

}

}

##### 5.1.4.1.2私有网络业务层

**public** **class** NetWorkServiceImpl **implements** NetWorkService {

购买私有网络接口：

**public** **boolean** buyNetWork(CloudOrderNetWork cloudOrderNetWork){

/\*

\* 1、调用创建订单接口；

{

Order order = new Order();

order.setOrderType("0");

//0新购1续费2升级

order.setProdName(cloudOrderNetWork.getNetName());

order.setDcId(cloudOrderNetWork.getDcId());

order.setPayType(cloudOrderNetWork.getPayType());

order.setBuyCycle(cloudOrderNetWork.getBuyCycle());

order.setResourceType("3");

//0云主机、1云硬盘、2云硬盘快照、3私有网络VPC、4负载均衡、5弹性公网IP、6对象存储、7VPN

...

}

Order reorder = OrdersService.createOrder(order);

\* 2、判断付款方式;

**if**("1".equals(cloudOrderNetWork.getPayType()))

then

cloudOrderNetWork.setOrderId(reorder.getOrderNo());

存表cloudorder\_network;

**else**

调用创建私有网络接口；

CloudNetWork cloudNetWork = **this**.addNetWork(cloudNetWork);

**if**(cloudNetWork != null)

then

向消息队列发送资源新购消息;

rabbitTemplate.send(Key,Body);

Key:

BILL\_RESOURCE\_PERCHASE

Body:

{

"bill\_factors" : {

"rate": "100"

//cloudOrderNetWork.getRate()

}

"charge\_from" : "1468823819",

//cloudOrderNetWork.getCreateTime()

"order\_number" : "N201607181434",

//cloudOrderNetWork.getOrderId()

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

//cloudNetWork.getNetId()

"datacenter\_id" : "1511251732240",

//cloudOrderNetWork.getDCId()

"resource\_type" : "VPC",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

//cloudOrderNetWork.getCustomerId()

}

\*/

}

创建私有网络接口：

**public** CloudNetWork addNetWork(CloudOrderNetWork cloudOrderNetWork){

/\*

\* 1、调用底层接口创建私有网络资源；

data =

{

"network" :{

"name" :cloudOrderNetWork.getNetName(),

"admin\_state\_up":"true"

}

}

NetWork network = openStackNetWorkService.create(

cloudOrderNetWork.getDcId(),

cloudOrderNetWork.getPrjId(),

data);

\* 2、判断底层创建成功;

**if**(network != null)

then

存库cloud\_network;

\* 将底层生成的network.getNetId()更新入库 cloudorder\_network；

\* 调用创建路由接口;

**boolean** createRouteFlag =

**this.**addRoute(cloudRoute);

\* 判断路由是否创建成功;

**if**(createRouteFlag)

then

\* 调用订单完成接口;

OrdersService.completeOrder(

cloudOrderNetWork.getOrderNo(),

cloudOrderNetWork.getNetName(),

description, "4");

//0待审核1待支付2资源创建中3资源创建失败-已取消4已完成5已取消

**return** cloudNetWork;

**else**

\* 调用私有网络删除接口;

**boolean** delNetWorkFlag = **this.**delNetWorkByNetId(cloudOrderNetWork);

判断私有网络是否删除成功;

**if**(!delNetWorkFlag)

then

**return null**;

\* 调用订单完成接口;

OrdersService.completeOrder(

cloudOrderNetWork.getOrderNo(),

cloudOrderNetWork.getNetName(),

description, "3");

\* 调用创建失败的消息接口;

**String** content = "'您的订单"

+ cloudOrderNetWork.getOrderNo()

+ "-私有网络VPC-"

+ "1".equals(cloudOrderNetWork.getPayType()) ?

"包年包月" : "按需按量"

+ "，因资源创建失败，已于"

+ cloudOrderNetWork.getCreateTime()

+ "被取消，系统稍后会将费用返还到您的账户中，请耐心等待！若有需要，您也可以到订单列表重新下单，给您带来不便，敬请谅解！";

MessageCenterService.addResourFailMessage(content,cloudOrderNetWork.getCustomerId());

**return null**;

\*/

}

创建路由接口：

/\*\*

\*

\* **@param** cloudRoute

\* **@return true/false**

\*/

**private** **boolean** addRoute(CloudRoute cloudRoute){

/\*

\* 1、调用底层接口创建路由；

resultData =

{

"router":{

"name" :cloudRoute.getRouteName()

}

}

Route route = openstackService.create(

cloudRoute.getDcId(),

cloudRoute.getPrjId(),

resultData);

\* 2、判断是否创建成功;

**if**(route != null)

then

本地入库cloud\_route;

**return** true;

**else**

**return** false;

\*/

}

变更配置私有网络接口：

**public** **boolean** changeNetWork(CloudOrderNetWork cloudOrderNetWork){

/\*

\* 1、调用创建订单接口；

Order order = OrdersService.createOrder(order);

\* 2、

**if**(cloudOrderNetWork.getPayType().equals("1"))

then

cloudOrderNetWork.setOrderNo(order.getOrderNo());

存表cloudorder\_network;

**else**

调用修改私有网络配置接口;

**this**.updateNetWork(cloudOrderNetWork);

向消息队列发送资源变配消息;

rabbitTemplate.send(Key,Body);

Key:

BILL\_RESOURCE\_UPGRADE

Body:

{

"bill\_factors" : {

"rate": "100"

}

"charge\_from" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "VPC",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

\*/

}

修改配置私有网络接口：

**public** CloudNetWork updateNetWork(CloudOrderNetWork cloudOrderNetWork) {

/\*

\* 1、调用底层接口编辑私有网络资源;

Network network =

OpenStackNetWorkService.update(

cloudOrderNetWork.getDcId(),

cloudOrderNetWork.getPrjId(),

data,

cloudOrderNetWork.getNetId());

\* 2、判断底层编辑是否成功;

**if**(network != null)

then

更新cloud\_network表;

\* 调用编辑路由接口;

**this**.updateRoute(

**String** routeId,

cloudOrderNetWork.getRate());

\* 调用订单完成接口;

OrdersService.completeOrder(

cloudNetWork.getOrderNo(),

cloudNetWork.getNetName(),

description,"4");

**return** cloudNetWork;

**else**

调用订单完成接口;

OrdersService.completeOrder(

cloudNetWork.getOrderNo(),

cloudNetWork.getNetName(),

description,"3");

调用创建失败的消息接口;

MessageCenterService.addResourceFailMessage(content,cloudOrderNetWork.getCustomerId());

\*/

}

编辑路由接口：

/\*\*

\*

\* **@param** routeId 路由id

\* **@param** rate 配置带宽

\* **@return**

\*/

**private** **void** updateRoute(String routeId,**int** rate){

/\*

\* 1、通过routeId获取实体类baseRoute;

\* 2、判断rate是否和当前路由的带宽相等;

if(rate == baseRoute.getRate())

then

**return**;

\* 3、判断路由是否关联qos;

if(!StringUtils.isEmpty(baseRoute.getQosId()))

then

调用qos的编辑接口更改带宽带宽;

eayunQosService.updateQos(baseRoute,rate);

更新cloud\_route库;

\*/

}

续费私有网络接口：

**public** **boolean** renewNetWork(CloudOrderNetWork cloudOrderNetWork){

/\*

\* 1、调用订单接口创建订单;

拼装Order order，不再赘述;

OrdersService.createOrder(order);

\* 2、存库cloudorder\_network;

\*/

}

改变私有网络计费状态的接口：

/\*\*

\* **@param** resourceId 资源id

\* **@param** chargeState 资源需要改变成的计费状态

\* **@param** endTime 新的到期时间 如果是后付费或不需要设定，传null

\* **@param** isRestrict 是否限制服务

\* **@param** isResumable 是否恢复服务

\*/

**public** **void** modifyStateForNetWork(

String resourceId,

String chargeState,

Date endTime,

**boolean** isRestrict,

**boolean** isResumable){

/\*

\* 1、将resourceId对应资源model cloudNetWork的计费状态改变为chargeState;

cloudNetWork.setChargeState(chargeState);

\* 2、if(endTime != null)

\* then 将对应资源的到期时间改变为endTime;

cloudNetWork.setEndTime(endTime);

\* 3、cloudNetWork入库cloud\_network

\* 4、if(isRestrict)

\* then

调用限制带宽接口限制带宽为1MB;

restrictNetWorkRate(resourceId);

向消息队列发送限制资源服务消息;

Key:

BILL\_RESOURCE\_RESTRICT

Body:

{

"timestamp" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "VPC",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

\* else

\* if(isResumable)

\* then

调用恢复带宽接口恢复带宽配置;

resumeNetWorkRate(resourceId);

向消息队列发送恢复资源服务消息;

Key:

BILL\_RESOURCE\_RECOVER

Body:

{

"timestamp" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "VPC",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

\*/

}

限制带宽接口：

/\*\*

\*

\* **@param** netId 私有网络id

\*/

**private** restrictNetWorkRate(String netId){

/\*

\* 1、获取与参数netId相匹配的实体类cloudNetWork

\* from cloud\_network where net\_id = netId

\* 2、获取与cloudNetWork.getRouteId()相匹配的实体类cloudRoute

\* from cloud\_route where route\_id = cloudNetWork.getRouteId()

\* 3、更改新旧的带宽属性值;

\* cloudRoute.setRateOld(cloudRoute.getRate());

\* cloudRoute.setRate("1");

\* 4、调用编辑路由配置接口;

\* updateRoute(cloudRoute.getRouteId(),int cloudRoute.getRate());

\*/

}

恢复带宽接口：

/\*\*

\*

\* **@param** netId 私有网络id

\*/

**private** resumeNetWorkRate(String netId){

/\*

\* 1、获取与参数netId相匹配的实体类cloudNetWork

\* from cloud\_network where net\_id = netId

\* 2、获取与cloudNetWork.getRouteId()相匹配的实体类cloudRoute

\* from cloud\_route where route\_id = cloudNetWork.getRouteId()

\* 3、

\* cloudRoute.setRate(cloudRoute.getRateOld());

\* 4、调用编辑路由配置接口;

\* updateRoute(cloudRoute.getRouteId(),int cloudRoute.getRate());

\*/

}

删除私有网络接口：

**public** **boolean** delNetWorkByNetId(CloudNetWork cloudNetWork) {

/\*

\* 1、调用底层删除接口删除私有网络；（接口会级删子网）

**boolean** bool = openStackNetWorkService.delete(cloudNetWork.getDcId(), cloudNetWork.getPrjId(), cloudNetWork.getNetId());

\* 2、判断底层是否删除成功;

**if**(bool == **true**)

then

\* 删库cloud\_network对应私有网络数据;

\* 删库cloud\_subnetwork对应子网数据;

通过cloudNetWork.getNetId()检索cloud\_route获取对应的路由实体类baseCloudRoute;

**if**(baseCloudRoute != null)

then

\* 调用底层删除接口删除私有网络对应的路由;

**boolean** delRouteFlag = openstackService.delete(cloudNetWork.getDcId(),**null**,baseCloudRoute.getRouteId());

\* **if**(delRouteFlag == **true**)

then

删库cloud\_route对应的路由数据；

\* 判断付款方式是否为后付费;

if("2".equals(cloudNetWork.getPayType()))

\* then

向消息队列发送资源删除消息;

Key:

BILL\_RESOURCE\_DELETE

Body:

{

"timestamp" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "VPC",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

**return** true;

**else**

调用底层资源删除失败的消息接口;

MessageCenterService.deleteResourceFailMessage(String content);

**else**

调用底层资源删除失败的消息接口;

MessageCenterService.delecteResourFailMessage(String content);

**return** false;

\*/

}

界面生成价格接口：

Public bigdecimal getPriceOfNetWork(BaseCloudOrderNetWork){

调用生成价格的接口获取价格信息并返回；

}

}

##### 5.1.4.1.3子网控制器（增加绑定和解绑接口）

SubNetWorkController (/cloud/subnetwork)

1、bindRoute(/bind)

|  |  |
| --- | --- |
| 说明 | 子网绑定路由 |
| URL | /cloud/subnetwork/bind.do |
| 方式 | POST |
| Post数据 | {  “dcId”：””,  “routeId”：””,  “subnetId”：””  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {  "code":"010120"  “resultData”：reason  } |

2、unbindRoute(/unbind)

|  |  |
| --- | --- |
| 说明 | 子网解绑路由 |
| URL | /cloud/subnetwork/unbind.do |
| 方式 | POST |
| Post数据 | {  “dcId”：””,  “routeId”：””,  “subnetId”：””  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {  "code":"010120"  “resultData”：reason  } |

3、getSubNetListByType(/getsubnetlist)

|  |  |
| --- | --- |
| 说明 | 获取某私有网络下自管或受管子网列表 |
| URL | /cloud/subnetwork/getsubnetlist.do |
| 方式 | POST |
| Post数据 | {  “netId”：””, //私有网络id  “subnetType”：”” //1受管子网，2自管子网  } |
| 返回Json | 成功：  {  "code":"000000"  "resultData": List<CloudSubnetWork>  }  失败：  {  "code":"010120"  } |

**public class** SubNetWorkController Extends BaseController {

绑定路由接口：

**public** String bindRoute(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

调用业务层绑定路由接口；

}

解绑路由接口：

**public** String unbindRoute(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

调用业务层解绑路由接口；

}

获取私有网络下自管或受管子网列表的接口：

**Public** String getSubnetListByType(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

获取受管子网或自管子网列表接口：

}

}

##### 5.1.4.1.4子网业务层

**public** **class** SubNetWorkServiceImpl **implements** SubNetWorkService {

添加子网接口：

**public** CloudSubNetWork addSubNetWork(CloudSubNetWork cloudSubNetWork) {

/\*

\* 1、调用底层接口对子网所在项目添加metering\_label\_rule标签;

\* 2、调用底层接口创建子网;

SubNetwork subNetWork

=openstackSubNetWorkService.create(

cloudSubNetWork.getDcId(), cloudSubNetWork.getPrjId(), data);

\* 3、判断创建成功;

if(subNetWork != null)

then

存库cloud\_subnetwork;

\* 4、判断子网类型

if(cloudSubNetWork.getSubnetType().equals("1"))

\* then

获取子网所在私有网络关联的路由;

CloudRoute cloudRoute = **this**.findRouteByNetId(cloudSubNetWork.getNetId());

判断所在私有网络关联的路由是否空

if(cloudRoute!=null

&&StringUtils.empty(cloudRoute.getRouteId()))

\* then

调用子网绑定路由私有接口;

**this**.connectSubnet(

cloudSubNetWork.getDcId(), cloudRoute.getRouteId(), cloudSubNetWork.getSubnetId());

\*/

}

/\*\*

\* 绑定路由私有接口：

\* **@param** dcId 数据中心id

\* **@param** routeId 路由id

\* **@param** subnetworkId 子网id

\* **@return**

\*/

**private** BaseCloudSubNetWork connectSubnet(

String dcId,

String routeId,

String subnetworkId){

/\*

\* 1、调用底层子网绑定路由接口;

\* 2、绑定成功后，if(路由的qosId不为空)

\* then 调用eayunQosService.createQos(BasecloudRoute cloudRoute)接口创建QOS;

\* 3、subNetwork.setRouteId(routeId)，更新入库cloud\_subnetwork;

\* 4、路由入库cloud\_route;

\*/

}

子网绑定路由开放接口：

/\*\*

\*

\* **@param** dcId 数据中心id

\* **@param** routeId 路由id

\* **@param** subnetId 子网id

\* **@return**

\*/

**public** String bindRoute(

String dcId,

String routeId,

String subnetId){

/\*

\* 调用connectSubnet(dcId,routeId,subnetId)接口绑定路由;

\*/

}

子网解绑路由开发接口：

/\*\*

\*

\* **@param** dcId 数据中心id

\* **@param** routeId 路由id

\* **@param** subnetId 子网id

\* **@return**

\*/

**Public** String unbindRoute(

String dcId,

String routeId,

String subnetId){

/\*

\* 调用detachSubnet(dcId,routeId,subnetId)接口解绑路由;

\*/

}

获取受管子网或自管子网列表接口：

/\*\*

\*

\* **@param** netId 私有网络id

\* **@param** subnetType 子网类型 1是受管子网；2是自管子网；

\* **@return List<CloudSubNetWork>**

\*/

**public** List<CloudSubNetWork> getSubNetListByType(

String netId，

String subnetType

){

通过子网类型获取子网列表；

}

}

##### 5.1.4.1.4私有网络计划任务

**public class** VPCRebbitMQ {

/\*

\* 监听order.pay.vpc消息队列的变化；

\* order.pay.vpc:{“orderNo”:””,”operateType”:””}

\* 由获取数据中的orderNo查找cloudorder\_network表中对应的配置数据；

\* 根据数据中的operateType来调用service层addNetWork接口创建网络资源或updateNetWork接口编辑网络资源；

\*/

}

#### 5.1.4.2负载均衡

##### 5.1.4.2.1负载均衡器控制器

PoolController(/cloud/loadbalance/pool)

1、buyBalancer(/buy)

|  |  |
| --- | --- |
| 说明 | 购买负载均衡器 |
| URL | /cloud/loadbalance/pool/buy.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderLdPool 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

2、changeBalancer(/change)

|  |  |
| --- | --- |
| 说明 | 变更配置负载均衡器 |
| URL | /cloud/loadbalance/pool/change.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderLdPool 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

3、getPriceOfLdPool(/getprice)

|  |  |
| --- | --- |
| 说明 | 生成负载均衡器界面价格 |
| URL | /cloud/loadbalance/pool/getprice.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderLdPool 类  } |
| 返回Json | 成功：  {  "code":"000000"，  "resultData":"120"  }  失败：  {"code":"010120"} |

1. renewLdPool(/renew)

|  |  |
| --- | --- |
| 说明 | 续费负载均衡器 |
| URL | /cloud/loadbalance/pool/renew.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderLdPool 类  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

Public class PoolController extends BaseController {

购买负载均衡器接口：

Public String buyBalancer(

HttpServletRequest request,

@RequestBody CloudOrderLdPool pool

){

调用负载均衡器业务层的购买负载均衡器接口；

}

变更配置负载均衡器接口：

Public String changeBalancer(

HttpServletRequest request,

@RequestBody CloudOrderLdPool pool

){

调用负载均衡器业务层的修改配置购买负载均衡器接口；

}

续费负载均衡器接口：

Public String renewBalancer(

HttpServletRequest request,

@RequestBody CloudOrderLdPool pool

){

调用负载均衡器业务层的续费负载均衡器接口；

}

生成界面价格接口：

Public String getPriceOfLdPool(

HttpServletRequest request,

@RequestBody CloudOrderLdPool pool

){

调用负载均衡器业务层中生成界面价格接口；

}

}

##### 5.1.4.2.2负载均衡器业务层

**public** **class** PoolServiceImpl **implements** PoolService {

购买负载均衡器接口：

**public** CloudLdPool buyBalancer(

CloudOrderLdPool pool,

SessionUserInfo sesionUser

){

/\*

\* 1、调用创建订单接口;

拼装Order参数类;

Order order = OrdersService.createOrder(sysOrder);

\* 2、判断付费方式;

**if**("1".equals(pool.getPayType()))

then

pool.setOrderId(order.getOrderNo());

存表cloudorder\_ldpool;

**else**

调用创建负载均衡器接口；

CLoudLdPool cloudLdPool = createBalancer(pool,sesionUser);

判断创建是否成功;

**if**(cloudLdPool != null)

then

向消息队列发送资源新购消息;

rabbitTemplate.send(Key,Body);

Key:

BILL\_RESOURCE\_PERCHASE

Body:

{

"bill\_factors" : {

"connection\_limit": "100"

//pool.getConnectionLimit()

}

"charge\_from" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "LB",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

\*/

}

创建负载均衡器接口：

**public** CloudLdPool createBalancer(

CloudOrderLdPool pool,

SessionUserInfo sessionUser

) **throws** AppException {

/\*

\* 1、调用底层接口创建负载均衡器资源;

Pool result = openStackService.create(

pool.getDcId(),

pool.getPrjId(),

data);

\* 2、判断是否创建成功

**if**(result != **null**)

then

入库cloud\_ldpool;

\* 将底层生成的pool\_id更新入库cloudorder\_ldpool;

\* 调用vip创建接口

BaseCloudLdVip vip = VipService.addVip(CloudLdVip vip);

判断vip是否创建成功;

**if**(vip != null)

then

\* 调用订单完成接口;

completeOrder(

pool.getOrderNo(),

pool.getPoolName,

description, "4");

**return** pool;

**else**

调用负载均衡删除接口;

**boolean** delLBFlag = deleteBalancer(pool);

判断删除是否成功;

**if**(!delLBFlag)

then

**return null**;

**else**

调用订单完成接口;

completeOrder(

pool.getOrderNo(),

pool.getPoolName,

description, "3");

调用创建失败的消息接口;

String content = "'您的订单"

+ pool.getOrderNo() +

"-负载均衡器LB-"

+ pool.getPayType().equals("1") ?

"包年包月" : "按需按量" +

"，因资源创建失败，已于"

+ pool.getCreateTime() +

"被取消，系统稍后会将费用返还到您的账户中，请耐心等待！若有需要，您也可以到订单列表重新下单，给您带来不便，敬请谅解！"

MessageCenterService.addResourFailMessage(content,pool.getCustomerId());

**return null**;

\*/

}

变更负载均衡配置接口：

**public** CloudLdPool changeBalancer(CloudOrderLdPool pool){

/\*

\* 1、调用创建订单接口;

Order order = createOrder(Order order);

\* 2、判断付费方式;

**if**("1".equals(pool.getPayType()))

then

pool.setOrderNo(order.getOrderNo());

存表cloudorder\_ldpool;

**else**

调用修改负载均衡器配置接口；

updateBalancer(pool);

向消息队列发送资源变配消息；

Key:

BILL\_RESOURCE\_UPGRADE

Body:

{

"bill\_factors" : {

"connection\_limit": "100"

}

"charge\_from" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "LB",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

\*/

}

修改负载均衡器配置接口：

**public** CloudLdPool updateBalancer(CloudLdPool pool) **throws** AppException {

/\*

\* 1、调用底层接口编辑负载均衡器资源;

Pool result = openStackService.update(

pool.getDcId(),

pool.getPrjId(),

data,

pool.getPoolId());

\* 2、判断底层编辑是否成功

**if**(result != **null**)

then

更新cloud\_ldpool表;

\* **if**(!StringUtils.isEmpty(pool.getVipId()))

then

调用vip编辑接口;

BaseCloudLdVip cloudLdvip = VipService.modifyVip(CloudLdVip vip);

调用订单完成接口;

completeOrder(

pool.getOrderNo(),

pool.getPoolName(),

description, "4");

**else**

调用订单完成接口；

completeOrder(

pool.getOrderNo(),

pool.getPoolName(),

description, "3");

调用创建失败的消息接口;

MessageCenterService.addResourceFailMessage(content,pool.getCustomerId());

\*/

}

改变负载均衡计费状态的接口：

/\*\*

\* **@param** resourceId 资源id

\* **@param** chargeState 资源需要改变成的计费状态

\* **@param** endTime 新的到期时间 如果是后付费或不需要设定，传null

\* **@param** isLimit 是否限制服务

\* **@param** isResumable 是否恢复服务

\*/

**public** **void** modifyStateForLdPool(

String resourceId,

String chargeState,

Date endTime,

**boolean** isRestrict,

**boolean** isResumable){

/\*

\* 1、将resourceId对应资源model cloudLdPool cloudLdPool.setChargeState(chargeState);

\* 2、if(endTime != null)

\* then cloudLdPool.setEndTime(endTime);

\* 3、cloudLdPool入库cloud\_ldpool;

\* 4、if(isRestrict == true)

\* then

调用解绑公网IP接口；

CloudFloatIpService.unbundingResource(cloudLdPool.getFloatIp());

\*/

}

删除负载均衡器接口：

**public** **boolean** deleteBalancer(CloudLdPool pool) {

/\*

\* 1、if(pool已经绑定了弹性公网ip)

\* then 向上抛AppException

\* 2、判断负载均衡器是否关联vip

\* **if**(!StringUtils.isEmpty(pool.getVipId()))

\* then

调用vip删除接口;

boolean vipFlag =VipService.deleteVip(CloudLdVip vip);

\* **if**(vipFlag || StringUtils.empty(pool.getVipId()))

\* then

调用底层接口删除负载均衡器资源;

**boolean** poolFlag = openStackService.delete(

pool.getDcId(), pool.getPrjId(), pool.getPoolId());

\* 判断删除成功

**if**(poolFlag)

then

删cloud\_ldpool表中对应的数据;

\* 级联删除负载均衡器成员和健康检查; deleteMemberAndMonitor(pool);

判断是否后付费;

\* **if**("2".equals(pool.getPayType()))

\* then

向消息队列发送资源删除消息;

Key:

BILL\_RESOURCE\_DELETE

Body:

{

"timestamp" : "1468823819",

"order\_number" : "N201607181434",

"resource\_id" : "0124e64b-aa69-4ea7-93ba-b05cede047db",

"datacenter\_id" : "1511251732240",

"resource\_type" : "LB",

"customer\_id" : "40289090523f758c01523fa1c0f5000d"

}

**return** true;

**else**

调用底层资源删除失败的消息接口;

MessageCenterService.delecteResourFailMessage(String content);

**else**

调用底层资源删除失败的消息接口;

MessageCenterService.delecteResourFailMessage(String content);

\*/

}

界面生成价格接口：

**public** String getPriceOfLdPool(BaseCloudOrderLdPool pool){

调用生成价格的接口获取价格信息并返回；

getPriceByFactor();

}

}

##### 5.1.4.2.3负载均衡器计划任务

Public class LdPoolRebbitMQ{

Public String monitorMQ(){

/\*

\* 监听order.pay.balancer消息队列的变化

\* order.pay.balancer:{“orderNo”:””,”operateType”:””}

\* 获取其中的orderNo来查cloudorder\_ldpool获取其中的配置参数

\* 根据operateType来调用不同的接口；

switch(operateType)

case ‘0’:

调用createBalancer接口创建负载均衡器资源；

case ‘2’:

调用updateBalancer接口修改负载均衡器资源配置；

}

}

#### 5.1.4.3端口映射

##### 5.1.4.3.1端口映射控制器

PortMappingController(/cloud/portmapping)

1. getPortMappingList(/getlist)

|  |  |
| --- | --- |
| 说明 | 获取端口映射列表 |
| URL | /cloud/portmapping/getlist.do |
| 方式 | POST |
| Post数据 | {  "pageNumber": "1",  "pageSize": "20",  "params": {  netId： “4028b881540d697301540d6ae6680000”  }  } |
| 返回Json | {  "pageSize": 10,  "totalCount": 78,  "result": [  {  ...//参见PortMapping 类  id: ”4028b881540d697301540d6ae6680000”,  protocol: ”TCP”,  resourceIp: ”192.168.1.1”,  resourcePort: ”8080”,  destinyId: “”,  destinyIp: ”10.10.10.1”,  destinyPort: ”90”,  netId: “4028b881540d697301540d6ae6680000”  },  ...  ],  "currentPageNo": 1,  "totalPageCount": 8  } |
|  |  |

1. createPortMapping(/create)

|  |  |
| --- | --- |
| 说明 | 创建端口映射 |
| URL | /cloud/portmapping/create.do |
| 方式 | POST |
| Post数据 | {  ...//PortMapping 类  "protocol": "TCP",  "resourceIp": "192.168.1.1",  "resourcePort": "8080",  “destinyId”: “”,  "destinyIp": "10.10.10.1",  "destinyPort": "90",  “netId”: “”  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

1. updatePortMapping(/update)

|  |  |
| --- | --- |
| 说明 | 修改端口映射 |
| URL | /cloud/portmapping/update.do |
| 方式 | POST |
| Post数据 | {  ...//参见PortMapping 类  "id": "4028b881540d697301540d6ae6680000",  "protocol": "TCP",  "resourceIp": "192.168.1.1",  "resourcePort": "8080",  “destinyId”: “”,  "destinyIp": "10.10.10.1",  "destinyPort": "90"  } |
| 返回Json | 成功：  {"code":"000000"}  失败：  {"code":"010120"} |

1. deletePortMapping(/delete)

|  |  |
| --- | --- |
| 说明 | 删除端口映射 |
| URL | /cloud/portmapping/delete.do |
| 方式 | POST |
| Post数据 | {  "id": "4028b881540d697301540d6ae6680000"  } |
| 返回Json | 成功：  {"code":"000000",  "status":true}  失败：  {"code":"010120"} |

@RequestMapping(“/cloud/portmapping”)

public class PortMappingController extends BaseController {

@RequestMapping(“/getlist”)

Public String getPortMappingList(

HttpServletRequest request,

Page page,

@RequestBody ParamsMap<String, String> params

) throws AppException{

调用service层的getPortMappingList接口；

}；

@RequestMapping(“/create”)

Public String createPortMapping(

HttpServletRequest request,

@RequestBody PortMapping portMapping

){

调用service层的createPortMappinp接口；

}；

@RequestMapping(“/update”)

Public String updatePortMapping(

HttpServletRequest request,

@RequestBody PortMapping portMapping

){

调用service层的updatePortMapping接口；

}；

@RequestMapping(“/delete”)

Public String deletePortMapping(

HttpServletRequest request,

@RequestBody Map<String, String> map

){

调用service层的deletePortMapping接口；

}；

}

##### 5.1.4.3.2端口映射业务层

public class PortMappingServiceImpl implements PortMappingService {

查询接口：

Public Page getPortMappingList(String netId) throws AppException {

1、调用dao层查询接口获取数据表中对应的数据；

}；

创建接口：

Public PortMapping createPortMappinp(PortMapping portMapping){

调用openstack创建接口创建底层端口映射资源；

调用dao层创建接口创建数据表中对应的数据；

}；

编辑接口：

Public PortMapping updatePortMapping(PortMapping portMapping){

调用openstack编辑接口编辑底层端口映射资源；

调用dao层编辑接口编辑数据表中对应的数据；

}；

删除接口：

Public boolean deletePortMapping(String pmId){  
 1、调用openstack删除接口删除底层端口映射资源；

2、调用dao层删除接口删除数据表中对应的数据；

}；

}

#### VPN业务模块

##### 5.1.4.1 VPN控制器

VPNController(/cloud/vpn)

1、getVPNList(/getlist)

|  |  |
| --- | --- |
| 说明 | 查询VPN列表 |
| URL | /cloud/vpn/getlist.do |
| 方式 | POST |
| Post数据 | {  "pageNumber": "1",  "pageSize": "20",  "params": {  "vpnName": "A001"  }  } |
| 返回Json | {  "pageSize": 10,  "totalCount": 78,  "result": [  {  "id": "aef71fb3-a195-4a01-8ec0-dfb4d93cbd63",  "vpnName": "A001",  "local\_network": "",  "local\_subnetwork": "",  "peer\_network": "",  "peer\_subnetwork": "",  "create\_time": ""  },  ...  ],  "currentPageNo": 1,  "totalPageCount": 8  } |

2、showVPNDetail(/showdetail)

|  |  |
| --- | --- |
| 说明 | 展示VPN详情 |
| URL | /cloud/vpn/showdetail.do |
| 方式 | POST |
| Post数据 | {  "id": "0eb4c7a2-8f89-4399-9be2-a5fa017918f4"  } |
| 返回Json | 成功：  {  "code": "000000",  "resultData": {  "VPN": VPN,  "IKEPolicy": IKEPolicy,  "IPSecPolicy": IPSecPolicy,  "VPNConnection": VPNConnection  }    }  失败：  {"code":"010120"} |

3、buyVPN(/buy)

|  |  |
| --- | --- |
| 说明 | 购买VPN |
| URL | /cloud/vpn/buy.do |
| 方式 | POST |
| Post数据 | {  "orderVPN": orderVPN,  } |
| 返回Json | 成功：  {  "code": "000000",  "resultData": true/false(生成订单成功或者失败)  }  失败：  {"code":"010120"} |

4、changeVPN(/change)

|  |  |
| --- | --- |
| 说明 | 修改VPN |
| URL | /cloud/vpn/change.do |
| 方式 | POST |
| Post数据 | {  "VPNModel": vpnModel  } |
| 返回Json | 成功：  {  "code":"000000"  "resultData":"true/false"  }  失败：  {"code":"010120"} |

5、deleteVPN(/delete)

|  |  |
| --- | --- |
| 说明 | 删除VPN |
| URL | /cloud/vpn/delete.do |
| 方式 | POST |
| Post数据 | {  "id": "4028b881540d697301540d6ae6680000"  } |
| 返回Json | 成功：  {  "code":"000000"  "resultData":"true/false"  }  失败：  {"code":"010120"} |

6、checkVPNName(/checkname)

|  |  |
| --- | --- |
| 说明 | 检验重名 |
| URL | /cloud/vpn/checkname.do |
| 方式 | POST |
| Post数据 | {  "id": "111111111",  "name": vpnName  } |
| 返回Json | 成功（有重名）：  {"code":"000000"，"resultData": "true"}  成功（无重名）  {"code":"000000"，"resultData": "false"}  失败：  {"code":"010120"} |

7、renewVpn(/renew)

|  |  |
| --- | --- |
| 说明 | VPN 续费 |
| URL | /cloud/vpn/renew.do |
| 方式 | POST |
| Post数据 | {  ..//参见BaseCloudOrderVPN类  } |
| 返回Json | {  "code": "000000",  "resultData": true  } |
| {"code":"010120"}//失败 |

8、getPriceOfVPN(/getprice)

|  |  |
| --- | --- |
| 说明 | 生成VPN界面价格 |
| URL | /cloud/vpn/pool/getprice.do |
| 方式 | POST |
| Post数据 | {  ...//参见CloudOrderVPN 类  } |
| 返回Json | 成功：  {  "code":"000000"，  "resultData":"120"  }  失败：  {"code":"010120"} |

@RequestMapping(“/cloud/vpn”)

public class VPNController extends BaseController {

1. 查询VPN接口；

Public String getVPNList(

HttpServletRequest request,

Page page,

@RequestBody ParamsMap paramsMap

){

调用service层getVPNList接口;

}

1. 详情VPN接口；

Public String showVPNDetail(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

调用service层showVPNDetail接口;

}

1. 购买VPN接口；

Public String buyVpn(

HttpServletRequest request,

@RequestBody CloudOrderVPN orderVPN

){

调用service层buyVpn接口;

}

1. 变更VPN配置接口；

Public String changeVpn(

HttpServletRequest request,

@RequestBody CloudOrderVPN orderVPN

){

调用service层changeVpn接口;

}

1. 删除VPN接口；

Public String deleteVpn(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

调用service层deleteVPN接口;

}

1. 校验重名接口；

Public String checkVPNName(

HttpServletRequest request,

@RequestBody Map<String ,String> map

){

调用service层checkVPNName接口;

}

1. 续费VPN接口：

Public String renewVPN(

HttpServletRequest request,

@RequestBody CloudOrderVPN orderVPN

){

调动service层renewVPN接口;

}

1. 生成VPN界面价格接口：

Public String getPriceOfVPN(

HttpServletRequest request,

@RequestBody CloudOrderVPN orderVPN

){

调用service层getPriceOfVPN接口;

}

}

##### 5.1.4.2 VPN业务层

public class VPNServiceImpl implements VPNService {

查询接口：

Public Page getVPNList(String vpnName){

调用dao层查询接口获取数据表中对应的数据；

}

详情接口：

Public VPNModel getVPNDetail(String vpnId){

调用dao层获取对应vpnId的数据详情；

(GET /v2.0/vpn/vpnservices/{service\_id})Show details for a specified VPN service.

}

购买接口：

**public boolean** buyVpn(BaseCloudOrderVPN orderVPN){

/\*

\* 1、调用订单接口生成新购VPN订单;

Order order = OrdersService.createOrder(order);

\* 2、判断付费方式;

if(orderVPN.getPayType().equals(“1”))

then

\* orderVPN.setOrderNo(order.getOrderNo());

存入cloudorder\_vpn表中;

else

调用创建vpn资源接口创建；

createVPNResource(orderVPN);

向消息队列发送资源新购消息;

rabbitTemplate.send(Key,Body);

Key:

BILL\_RESOURCE\_PERCHASE

Body:

{

"bill\_factors" :{}

//无配置计费因子

"resource\_type" : "VPN"

。。。其他参看私有网络和负载均衡

}

\*/

}

创建VPNService接口

**private** BaseCloudVPn createVPNService(BaseCloudOrderVPN vpnOrder){

/\*

\* 1、调用底层创建VPNservice接口;

BaseCloudVPN result = openStackVPNService.create(

String vpnOrder.getPrjId(),

String vpnOrder.getDcId(),

data);

\* 2、创建是否成功;

if(result != null)

then

存入cloud\_vpnservice表中;

vpnOrder.setVpnServiceId(result.getVpnId());

更新入cloudorder\_vpn表中;

return result;

else

return null;

\*/

}

编辑VPNService接口

**private** BaseCloudVPN updateVPNService(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层编辑VPNservice接口;

BaseCloudVPN result = openStackVPNService.update(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

data,

vpnOrder.getVPNServiceId());

\* 2、更新cloud\_vpnservice表中;

\*/

}

删除VPNService接口

**private boolean** deleteVPNService(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层删除VPNservice接口;

boolean bool = openStackVPNService.delete(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

vpnOrder.getVpnServiceId());

\* 2、判断是否删除成功;

if(bool)

then

删除cloud\_vpnservice对应的数据;

return true;

else

return false;

\*/

}

创建IKEPolicy接口

**private** BaseCloudIKEPolicy createIKEPolicy(BaseCloudOrderVPN vpnOrder){

/\*

\* 1、调用底层创建IKEPolicy接口创建;

BaseCloudIKEPolicy result = openStackIKEPolicyService.create(

String vpnOrder.getPrjId(),

String vpnOrder.getDcId(),

data);

\* 2、创建是否成功;

if(result != null)

then

存入cloud\_ikepolicy表中;

vpnOrder.setIkePolicyId(result.getIkePolicyId());

更新入cloudorder\_vpn表中;

return result;

else

return null;

\*/

}

编辑IKEPolicy接口

**private** BaseCloudIKEPolicy updateIKEPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层编辑VPNservice接口;

BaseCloudVPN result = openStackIKEPolicy.update(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

data,

vpnOrder.getIkeId());

\* 2、判断编辑是否成功;

if(result != null)

then

更新cloud\_ikepolicy表中;

return result;

else

return null;

\*/

}

删除IKEPolicy接口

**private boolean** deleteIKEPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层删除IKEPolicy接口;

boolean bool = openStackVPNService.delete(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

vpnOrder.getIkeId());

\* 2、判断是否删除成功;

if(bool)

then

删除cloud\_ikepolicy对应的数据;

return true;

else

return false;

\*/

}

创建IPSecPolicy接口

**private** BaseCloudIPSecPolicy createIPSecPolicy(BaseCloudOrderVPN vpnOrder){

/\*

\* 1、调用底层创建IKEPolicy接口创建;

BaseCloudIPSecPolicy result = openStackIPSecPolicyService.create(

String vpnOrder.getPrjId(),

String vpnOrder.getDcId(),

data);

\* 2、创建是否成功;

if(result != null)

then

存入cloud\_ipsecpolicy表中;

vpnOrder.setIpSecPolicyId(result.getIpSecPolicyId());

更新入cloudorder\_vpn表中;

return result;

else

return null;

\*/

}

编辑IPSecPolicy接口

**private** BaseCloudIPSecPolicy updateIPSecPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层编辑IPSecPolicy接口;

BaseCloudVPN result = openStackIPSecPolicy.update(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

data,

vpnOrder.getIpSecId());

\* 2、判断编辑是否成功;

if(result != null)

then

更新cloud\_ipsecpolicy表中;

return result;

else

return null;

\*/

}

删除IPSecPolicy接口

**private boolean** deleteIKEPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层删除IKEPolicy接口;

boolean bool = openStackVPNService.delete(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

vpnOrder.getIpSecId());

\* 2、判断是否删除成功;

if(bool)

then

删除cloud\_ipsecpolicy对应的数据;

return true;

else

return false;

\*/

}

创建vpnConn接口

**private** BaseCloudVPNConnection createVPNConn(BaseCloudOrderVPN vpnOrder){

/\*

\* 1、调用底层创建IKEPolicy接口创建;

BaseCloudIKEPolicy result = openStackIKEPolicyService.create(

String vpnOrder.getPrjId(),

String vpnOrder.getDcId(),

data);

\* 2、创建是否成功;

if(result != null)

then

存入cloud\_ikepolicy表中;

vpnOrder.setVPNId(result.getId());

return result;

else

return null;

\*/

}

编辑vpnConn接口

**private** BaseCloudIPSecPolicy updateIKEPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层编辑vpnConn接口;

BaseCloudVPN result = openStackvpnConnection.update(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

data,

vpnOrder.getVpnConnId());

\* 2、更新cloud\_vpnconn表中;

\*/

}

删除IPSecPolicy接口

**private boolean** deleteIKEPolicy(BaseCLoudOrderVPN vpnOrder){

/\*

\* 1、调用底层删除IKEPolicy接口;

boolean bool = openStackVPNService.delete(

vpnOrder.getPrjId(),

vpnOrder.getDcId(),

vpnOrder.getIpSecId());

\* 2、判断是否删除成功;

if(bool)

then

删除cloud\_ipsecpolicy对应的数据;

return true;

else

return false;

\*/

}

创建VPN资源接口：

**public** BaseCloudOrderVPN createVPNResource(BaseCloudOrderVPN vpnOrder){

调用openstack创建接口创建底层VPN资源；  
  (POST/v2.0/vpn/vpnservices)creates a VPN service.

({

“vpnservice” : {

“subnet\_id” : vpn.subnetId,

“router\_id” : vpn.routeId,

“name” : vpn.name,

“admin\_state\_up” : “true”

}

})

(POST /v2.0/vpn/ikepolicies)creates a IKE policy.

{

“ikepolicy” : {

“phase1\_negotiation\_mode” : ikePolicy.nogotiation,

“auth\_algorithm” : ikePolicy.authAlgorithm,

“encryption\_algorithm” : ikePolicy.encryption,

“pfs” : “group5”,

“lifetime” : {

“units” : “seconds”,

“value” : ikePolicy.lifetimeV,

},

“ike\_version” : ikePolicy.version,

“name” : ikePolicy

}

(POST /v2.0/vpn/ipsecpolicies)creates a IPSec policy.

{

“ipsecpolicy” : {

“name” : “”,

“transform\_protocol” : ipSecPolicy.transform,

“auth\_algorithm” : ipSecPolicy.authAlgorithm,

“encapsulation\_mode” : ipSecPolicy.encapsulation,

“encryption\_algorithm” : ipSecPolicy.encryption,

“pfs” : “group5”,

“lifetime” : {

“units” : “seconds”,

“value” : ipSecPolicy.lifetimeV

}

}

}

(POST /v2.0/vpn/ipsecsiteconnections)creates a IPSec connection.

{

“ipsec\_site\_connection” : {

“psk” : “secret”,

“initiator” : vpnConnection.initiator,

“ipsecpolicy\_id” : vpnConnection.ipSecPolicyId,

“admin\_state\_up” : “true”,

“peer\_cidrs” : vpnConnection.peerCidrs,

“mtu” : vpnConnection.mtu,

“ikepolicy\_id” : vpnConnection.ikePolicyId,

“dpd” : {

“action” : vpnConnection.dpdAction,

“interval” : vpnConnection.dpdInterval,

“timeout” : vpnConnection.dpdTimeout

},

“vpnservice\_id” : vpnConnection.vpnServiceId,

“peer\_address” : vpnConnection.peerAddress,

“peer\_id” : vpnConnection.peerId,

“name” : vpnConnection.name

}

}

/\*

\* 调用创建vpn基础配置接口

BaseCloudVPN vpnService = createVPNService(vpnOrder);

**if**(vpnService!=null)

then

调用创建ikepolicy接口

BaseCloudIKEPolicy ike =

createIKEPolicy(vpnOrder);

**if**(ike!=null)

then

调用创建ipsec接口；

BaseCloudIPSecPolicy ipsec =

createIPSecPolicy(vpnOrder);

**if**(ipsec!=null)

then

调用创建vpn管道数据接口;

BaseCloudVPNConnection vpnConn =

createVPNConn(vpnOrder);

**if**(vpnConn!=null)

then

调用订单完成接口;

OrdersService.completeOrder(

vpnOrder.getOrderNo(),

vpnOrder.getVPNName(),

description, "4"

);

**return** vpnOrder;

调用删除ipsec接口；

deleteIPSecService(vpnOrder);

调用删除ikepolic接口；

deleteIkePolicy(vpnOrder);

调用删除vpnService接口；

deleteVPNService(vpnOrder);

调用底层删除失败的消息接口发送资源创建失败的消息;

MessageCenterService.addResourFailMessage(content,vpnOrder.getCustomerId());

**return null**;

\*/

}

变更VPN配置接口：

**public boolean** changeVPN(BaseCloudOrderVPN vpnOrder){

调用openstack编辑接口编辑底层VPN资源；

调用dao层编辑接口编辑数据表中对应的数据；

调用订单完成接口；

/\*

\* 调用vpnservice编辑接口;

BaseCloudVPN vpnservice = updateVPNService(vpnOrder);

if(vpnservice!=null)

then

调用vpnconn编辑接口；

BaseCloudVPNConn vpnConn = updateConn(vpnOrder);

if(vpnConn!=null)

Then

调用订单完成接口；

OrdersService.completeOrder(

vpnOrder.getOrderNo(),

vpnOrder.getVPNName(),

description, "4");

return true;

调用编辑资源失败的消息接口向客户发送消息;

MessageCenterService.addResourFailMessage(content,vpnOrder.getCustomerId());

return false;

\*/

}

续费VPN接口：

**public boolean** renewVpn(BaseCloudOrderVPN orderVPN){

1. 调用订单接口生成续费VPN订单;

Order order = OrdersService.createOrder(order);

2、orderVPN.setOrderNo(order.getOrderNo());

3、orderVPN插入cloudorder\_vpn中;

}

改变VPN计费状态接口：

/\*\*

\* **@param** resourceId 资源id

\* **@param** chargeState 资源需要改变成的计费状态

\* **@param** endTime 新的到期时间 如果是后付费或不需要设定，传null

\* **@param** isLimit 是否限制服务

\* **@param** isResumable 是否恢复服务

\*/

**public** **void** modifyStateForVPN(

String resourceId,

String chargeState,

Date endTime,

**boolean** isLimit,

**boolean** isResumable){

/\*

\* 1、将resourceId对应的vpn资源的计费状态改变为chargeState；

vpn.setChargeState(chargeState);

\* 2、if(endTime != null)

\* then 将对应资源的到期时间改变为endTime;

vpn.setEndTime(endTime);

\* 3、入库cloud\_vpnconn

\*/

}

删除接口：

**public** String deleteVPN(VPNModel vpnModel){

/\*

\* 1、调用删除vpnService接口;

boolean vpnFlag = deleteVPNService(vpnOrder);

if(vpnFlag)

then

调用删除IKEPolicy接口;

boolean ikeFlag = deleteIKEPolicy(vpnOrder);

if(ikeFlag)

Then

调用删除IPSECPolicy接口;

boolean ipsecFlag = deleteIPSecPolicy(vpnOrder);

if(ipsecFlag)

then

boolean vpnconn = deleteVPNConn(vpnOrder);

if("2".equals(vpnOrder.getPayType()))

then

调用消息接口向计费模块发送删除资源的message

return true;

调用底层删除资源失败的接口向运维人员发送消息;

return false;

\*/

}

校验重名接口：

**public boolean** checkVPNName(String vpnName){

1、检验数据库中是否存在重名数据；

}

获取VPN界面价格接口：

**public** String getPriceOfVPN(CloudOrderVPN orderVPN){

调用生成价格的接口获取价格信息并返回；

}

}

##### 5.1.5.3 VPN计划任务

Public class VPNRabbitMQ () {

/\*

\* 监听order.pay.vpn消息队列的变化

order.pay.vpn:{“orderNo”:””,”operateType”:””}

\* 通过获取的orderNo查cloudorder\_vpn来获取创建vpn的资源配置；

\* 通过获取的operateType来决定是否调用createVPN接口创建VPN资源：

if(“0”.equals(operateType))

then

调用createVPN接口创建vpn资源；

\*/

1. 监听RESOURCE\_VPN\_CREATE消息队列，由获取数据中的参数JSON来调用service层的createVPN接口创建VPN资源；

}

# 6相关设计问题的整理

1. 用户手动删除资源和创建资源时底层报错，而回滚删除已经创建的云资源是否都向相同的人员发送消息通知？ 解答：手动删除不需要！



