漫步云中网络

在生产环境中，云中的网络通常被划分为公共网络、管理网络和服务网络。本文首先通过三个小试验向您介绍了如何通过 TAP/TUN、NAT、Linux Bridge、VLAN 等技术实现云中网络的一般原理。有了这些基础，相信您会对接下来介绍的一个具体的 OpenStack 云的示例网络配置倍感亲切。同理，这些基础也将助您在其他云中网络中轻松漫步。

张华，IBM 高级软件工程师热衷于技术钻研，拥有丰富的搜索引擎、应用服务器、互联网、云计算领域的行业经验，精通 Java、JavaEE、Linux、Network 等技术，目前正从事 OpenStack 相关的工作。

龚永生，IBM 资深软件工程师，热衷于开源软件，具有多年的 Linux，Java 和 JavaEE 经验。目前是 OpenStack 项目的积极贡献者。

2012 年 9 月 25 日

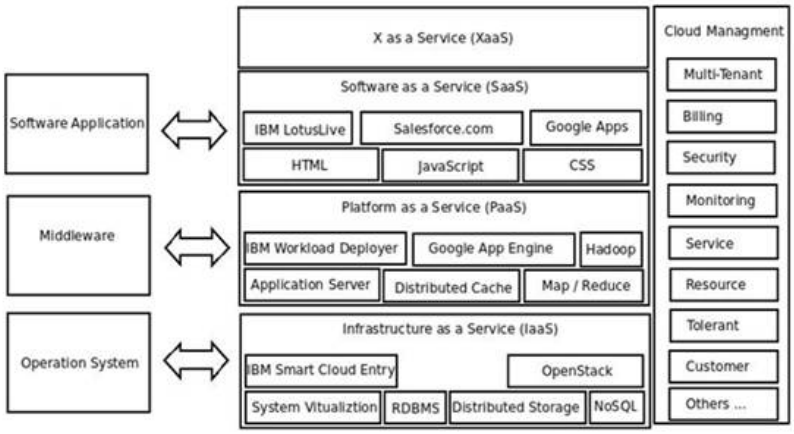
阅读本文前，最好能先了解以下的知识：

- 了解 OpenStack 将有助于对本文的理解。本文讲解的是 Linux 虚拟网络中的一般原理方法，虽不仅限于应用在 OpenStack 之中，但本文的实验是以 OpenStack 为基础的。OpenStack 是一个开源的 IaaS 云，您可以从 devstack 脚本 (http://devstack.org/) 开始熟悉它。
- 了解 QEMU 也将有助于对本文的理解。QEMU 是一种支持多种 CPU 的机器模拟器，本文采用 QEMU 来创建虚拟机验证本文中的试验。

什么是云？

什么是云？我的理解是，为多租户提供各层次上的服务（如操作系统层、中间件层、应用软件层等）的可动态水平扩展的服务器集群称之为云。所以云具有大规模、高可扩展性、按需服务、自动化、节能环保、高可靠性等特点。下图 1 从软件堆栈视角勾画了云的架构：

图 **1.** 云的架构



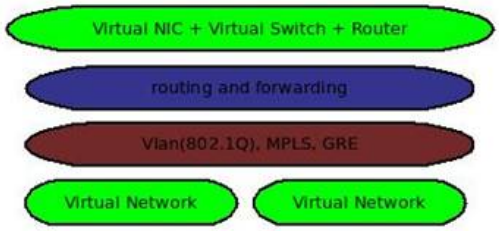
- **IaaS, Infrastructure as a Service**，基础设施即服务：您可以简单理解为将可伸缩的操作系统（虚机或实机）实例作为基础设施服务卖给多租户，然后按需计算费用。当然，将操作系统作为基础设施服务只是 IaaS 中的一种，且是最主要的一种，我怕大家概念混淆所以就只重点提了这种。实际上，只要是基础设施提供服务了从概念上讲都应该叫 IaaS，比如说关系型数据库，如果是集群部署的话，它也是基础设施提供服务了，也应该叫 IaaS。这类产品如 IBM 的 Smart Cloud Entry，如开源的 OpenStack。
- **PaaS, Platform as a Service,** 平台即服务：您可以简单理解为将可伸缩的中间件资源作为平台服务卖给多租户，然后按需计算费用。举个例子，如果 SaaS 应用程序的并发瞬间加大的话，PaaS 可以自动实时地启动一个由 IaaS 提供的操作系统实例，然后自动在它上面部署中间件应用服务器（如 IBM 的 WebSphere），最后再部署一套该 SaaS 应用实例，并自动将它们纳入到负载均衡体系之中，从而实现平台服务的自动伸缩，这就是 PaaS。这类产品如 IBM 的 IWD，如 Google 的 App Engine。
- **SaaS, Software as a Service,** 软件即服务：您可以简单理解为可伸缩的分布式软件作为软件服务为用户提供某种在线服务，如视频服务，地图服务等。
- **XaaS, X as a Server,** 一切即服务：只要是给多租户按需提供服务都可以叫 XaaS，像在 OpenStack 中，将网络部分代码单独抽出来组成 Quantum 工程，就可以叫网络即服务（NaaS, Network as a Service)；像使用 xCat 自动部署裸机可以叫裸机即服务（MaaS, Bare-metal as a Service)。

什么是云中网络？

在传统的数据中心中，每个网口对应唯一一个物理机；有了云，一台物理网卡可能会承载多个虚拟网卡。物理网卡与虚拟网卡之间的关系无外乎就是下列三种情况：

1. 一对一，一个物理网卡对应对一个虚拟网卡，是下面一对多情况的一种特例
2. 一对多，一个物理网卡对应多个虚拟网卡，是本文要介绍的情况
3. 多对一，多个物理网对应一个虚拟网卡，即我们常说的 Bonding，用作负载均衡

图 **2.** 虚拟网络的主要内容



上图 2 显示了虚拟网络的主要内容：

1. 目前，对网络的虚拟化主要集中在第 2 层和第 3 层
2. 在 Linux 中，第 2 层通常使用 TAP 设备来实现虚拟网卡，使用 Linux Bridge 来实现虚拟交换机
3. 在 Linux 中，第 3 层通常是基于 Iptable 的 NAT，路由及转发
4. 对于网络隔离，可以采用传统的基于 802.1Q 协议的 VLAN 技术，但这受限于 VLAN ID 大小范围的限制，并且需要手动地在各物理交换机上配置 VLAN；也可以采用虚拟交换机软件，如 Openvswitch，它可以自动创建 GRE 隧道来避免手动去为物理交换机配置 VLAN。

下面将结合一个生产环境中的网络实例来讲解如何实现一个虚拟网络。

云中网络实验

在生产环境中，按通用做法一般将云中网络划分为三大部分，公共网络、管理&存储网络、服务网络。

公共网络：用于云向外部租户提供 API 调用或者访问

管理网络：用于云中各物理机之间的通信

存储网络：用于 iSCSI 服务端与客户端之间的流量，一般与管理网络同

服务网络：虚机内部使用的网络

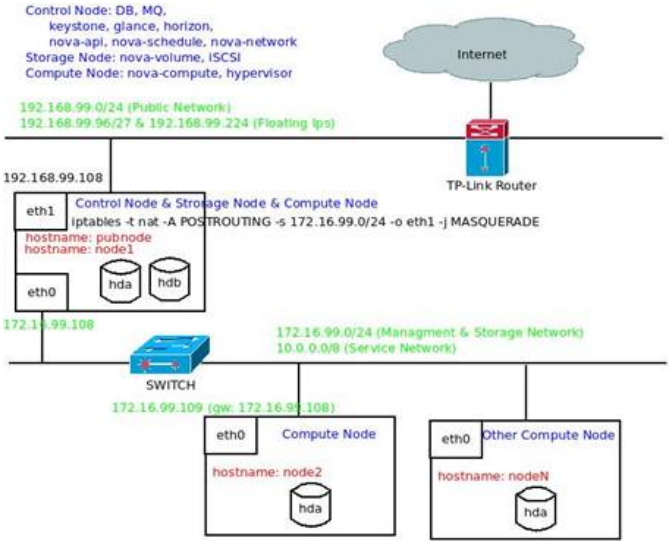
为了将上述网络的实现原理讲清楚，我们选择了两台物理机做实验，并将采用 NAT、Linux Bridge、VLAN 技术分步实现一个典型的 OpenStack 云的网络拓扑。当然，这种网络的原理是通用的，并不仅限于 OpenStack 云。

1. 台式机 (node1)， 双有线网卡， 将作为控制节点、存储节点及一个计算节点
2. 笔记本 (node2)，一有线网卡，将作为一个计算节点
3. 路由器，家中 ADSL 宽带出口
4. 交换机 , 用于连接各物理机

值得一提的是，如果采用了 VLAN 技术进行网络隔离，且想要两台物理机上的虚机能够互访的话，交换机必须是支持 VLAN 的，且需要手动将交换机相应的端口配置成 Trunk 模式。因为我没有支持 VLAN 的物理交换机，在本实验中，我是采用直连线直接连接两台实验机器的。

下图 3 显示了实验网络拓扑：

图 **3. OpenStack** 实验网络拓扑



公共网络：192.168.99.0/24 网段，外网用户通过公共网络上提供的服务来访问云。注意：在实际的生产环境中，公共网络一般采用外网 IP，因为我没有外网 IP，所以用 192.168.99.0 网段模拟。将台式机的一有线网卡 eth1 与 TP-Link 路由器相连即可。

管理＆存储网络：172.16.99.0/24 网段，管理网络用于 OpenStack 各组件以及 DB、MQ 之间进行通信；存储网络用于存储节点和需要使用外部存储的计算节点之间的通信。将台式机的另一有线网卡 eth0 和笔记本电脑的有线网卡 eth0 连接到交换机即可。

服务网络：10.0.0.0/8 网段，用于虚机内部。

两个节点的基本网络配置如下：

清单 **1. node1** 的基本网络配置

```
root@node1:/home/hua# cat /etc/network/interfaces
auto lo
iface lo inet loopback
auto eth1
iface eth1 inet dhcp
up iptables -t nat -A POSTROUTING -s 172.16.99.0/24 -o eth1 -j MASQUERADE
auto eth0
iface eth0 inet static
address 172.16.99.108
netmask 255.255.255.0
network 172.16.99.0
broadcast 172.16.99.255
```

清单 **2. node2** 的基本网络配置

```
cat /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
DEVICE=eth0
HWADDR=00:21:86:94:63:27
ONBOOT=yes
BOOTPROTO=static
USERCTL=yes
PEERDNS=yes
IPV6INIT=no
NM_CONTROLLED=yes
TYPE=Ethernet
NETMASK=255.255.255.0
IPADDR=172.16.99.109
NETWORK=172.16.99.0
GATEWAY=172.16.99.108
DNS1=202.106.195.68
DNS2=202.106.46.151
```

我虽然只用了两台物理机来模拟实际生产环境的部署模型， 但文中的这种部署结构是典型的，如果是大规模部署的话，只需要将控制节点上的每一个进程（如 DB、MQ、

glance、keystone、nova-api、nova-schedule、nova-network 等）分布部署在每一台物理机即可。想要进一步的 HA 的话，可以：

- 将 DB 配置成集群模式
- 将 MQ 配置成集群模式
- 采用 multi-host 模式，将 nova-network 同时安装在计算节点 (nova-compute) 上
- 将 nova-api、nova-schedule 这些无状态的服务也同时部署在计算节点上，再加上负载均衡器分发负载
- 采用多网卡做 Bonding

NAT

node2 可以通过 NAT 方式访问外网，数据流向如下：

1. node2 中需设置网关指向 node1 的 eth0，例：

```
GATEWAY=172.16.99.108
```

2. 在 node1 中打开 ipv4 转发功能，这样，node1 会相当于一台路由器，在 eth0 收到来自 node2 的数据之后，会将数据包转发到其他网卡 eth1，

```
sysctl -w net.ipv4.ip_forward=1
```

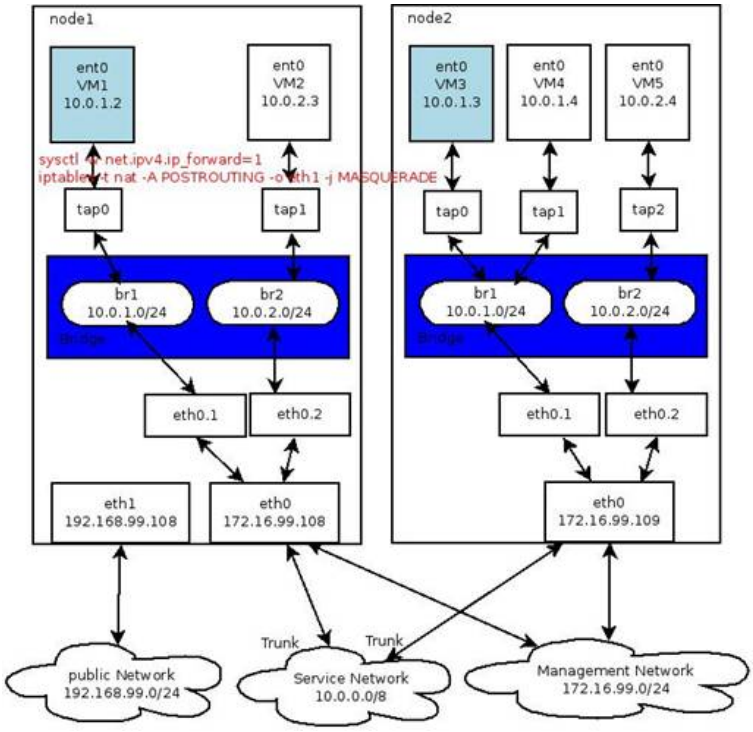
3. 在 node1 上设置 NAT 规则，这样，从 node2（172.16.99.0/24 网段）发出的数据包看来起就像从 node1 的 eth1 发出的一样：

```
iptables -t nat -A POSTROUTING -s 172.16.99.0/24 -o eth1 -j MASQUERADE
```

Linux Bridge

网桥 (Bridge) 工作在二层，了解链路层协议，按帧转发数据。就是交我们常说的交换机，所以连接到网桥的设备处于同一网段。

图 **4.** 网桥示例



上图 4 显示了 node1 网桥中的 VM1 与 node2 网桥中的 VM2 是如何通信的。在 openstack 中，这是典型的 multi-host 模式，即每一个计算节点均部署了网络服务来提供网关服务。Linux Bridge 充当了交换机的功能，而将 sysctl -w net.ipv4.ip_forward 设置为 1 也相当于 node1 同时充当了一个路由器（路由器的实质就是一个具有多个网卡的机器，因为它的多网卡同时具有这些不同网段的 IP 地址，所以它能将一个网络的流量路由转发到另一个网络）。

网桥，交换机，是用来连接两个 LAN 的。是根据 MAC 与端口的映射进行转发的，而在虚机的网卡都是知道的，若从转发数据库知道目的 MAC 地址，以太网帧就只会正确的网桥端口传输，否则，就会扩散到网桥设备的所有端口。

因为网桥工作在第二层，所以 eth0.1, tap0, tap1 这些网卡均不需要设置 IP（因为对于上层路由器来说，它们是同一个子网，只需要将 IP 设置在 br1 上即可）。同时，对 Linux 而言，网桥是虚拟设备，因此，除非将一个或多个真实设备绑定到网桥设备上，否则它就无法接收或传输任何东西。所以需要将一个真实设备（如 eth0）或者真实设备的 vlan 接口（如 eth0.1) 加入到网桥。对于前一种情况，将 eth0 加入到网桥之后，eth0 便不再具有 IP，这时候它与 tap0 这些虚拟网卡均通过 br1 处于 10.0.1.0/24 网络，当然我们也可以为网桥 br1 设置一个别名让它也具有 172.16.99.0/24 网管网段的 IP 地址。）

下面，我们来实现这个示例网桥，在 node1 与 node2 上分别执行下述脚本（对重要命令的描述请参见注释）：

清单 **3. node1** 与 **node2** 的 **Linux Bridge** 配置脚本

```
#!/bin/sh
TAP=tap0
BRIDGE=br1
IFACE=eth0
MANAGE_IP=172.16.99.108
SERVICE_IP=10.0.1.1
GATEWAY=10.0.1.1
BROADCAST=10.0.1.255
# 设置物理网卡为混杂模式
ifdown $IFACE
ifconfig $IFACE 0.0.0.0 promisc up
# 创建网桥，并物理网卡加入网桥，同时设置网桥的 IP 为服务网络网段
brctl addbr $BRIDGE
brctl addif $BRIDGE $IFACE
brctl stp $BRIDGE on
ifconfig $BRIDGE $SERVICE_IP netmask 255.255.255.0 broadcast $BROADCAST
route add default gw $GATEWAY
# 在网桥上设置多 IP，让它同时具有管理网段的 IP
ifconfig ${BRIDGE}:0 $MANAGE_IP netmask 255.255.255.0 broadcast 172.16.99.255
```

注意，上述黑体的一句在 node2 中需要作相应修改，其余不变，如下：

```
MANAGE_IP=172.16.99.109
```

VLAN

图 4 同样适用于 VLAN 网络，下面我们来实现它。在 node1 与 node2 上分别执行下述脚本：

清单 **4. node1** 与 **node2** 的 **VLAN** 配置脚本

```
MAC=c8:3a:35:d7:86:da
 IP=10.0.1.1/24
 ip link add link eth1 name eth1.1 type vlan id 1
 ip link set eth1.1 up
 brctl addbr br1
 brctl setfd br1 0
 brctl stp br1 on
 ip link set br1 address $MAC
```

```
ip link set br1 up
brctl addif br1 eth1.1
ip addr add $IP dev br1
```

注意，上述黑体的一句在 node2 中需要作相应修改，其余不变，如下：

```
MAC= c8:3a:35:d7:86:db
```

测试

我们采用 QEMU 创建虚拟机来进行测试，其中网络部分的配置为：

```
   <interface type='bridge'>
      <mac address='52:54:00:00:01:89'/>
      <source bridge='br1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
   </interface>
```

我们使用一个现成的镜像，下载地址：

```
curl http://wiki.qemu.org/download/linux-0.2.img.bz2 -o /bak/kvmimages/linux-0.2.img
```

在 node1 与 node2 上分别用下列配置定义两个虚机，注意，下面打粗体的部分（<mac address='52:54:00:00:01:89'/>）在两个节点中请设置不一样的值。

cat /etc/libvirt/qemu/test.xml

清单 **5. node1** 与 **node2** 的虚机定义文件

```
<domain type='qemu'>
<name>VM1</name>
<uuid></uuid>
<memory>393216</memory>
<currentMemory>393216</currentMemory>
<vcpu>1</vcpu>
<os>
<type arch='i686' machine='pc-1.0'>hvm</type>
<boot dev='hd'/>
</os>
<features>
<acpi/>
</features>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
<emulator>/usr/bin/qemu-system-i386</emulator>
<disk type='block' device='disk'>
<driver name='qemu' type='raw'/>
<source dev='/bak/kvmimages/linux-0.2.img'/>
<target dev='hda' bus='ide'/>
<address type='drive' controller='0' bus='0' unit='0'/>
</disk>
<controller type='ide' index='0'>
<address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
</controller>
<interface type='bridge'>
<mac address='52:54:00:00:01:89'/>
<source bridge='br1'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
<input type='tablet' bus='usb'/>
<input type='mouse' bus='ps2'/>
<graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'>
<listen type='address' address='127.0.0.1'/>
</graphics>
<video>
<model type='cirrus' vram='9216' heads='1'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</video>
<memballoon model='virtio'>
<address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</memballoon>
</devices>
</domain>
```

然后用上述配置创建虚机（virsh define /etc/libvirt/qemu/test.xml ），接着启动虚机（ virsh start test ），最后设置虚机的 IP 和默认网关， 如下：

VM1：

```
ifconfig eth0 10.0.1.2 netmask 255.255.255.0 broadcast 10.0.1.255
route add default gw 10.0.1.1
```

VM3：

```
ifconfig eth0 10.0.1.3 netmask 255.255.255.0 broadcast 10.0.1.255
route add default gw 10.0.1.1
```

这时候在一虚机上 ping 另一虚机，如果能够 ping 通，成功。若想要 ping 外网的话，还需在 /etc/resolv.conf 文件中添加域名，如下图 5 所示：

图 **5.** 验证实验是否成功

OpenStack 云中网络拓扑配置示例

在掌握了上述基本原理之后，应该不难理解下面 OpenStack 云中的网络拓扑。

在 OpenStack 中，目前存在着 nova-network 与 quantum 两种网络组件。nova-network 仅支持下列三种网络拓扑：

- FlatManager, 不支持 VLAN，也不支持 DHCP 的扁平网络
- FlatDHCPManager，不支持 VLAN，但支持 DHCP 的扁平网络
- VlanManager，支持 VLAN，也支持 DHCP 的 VLAN 隔离网络

如今 nova-network 的代码已经全部挪到了 Quantum 工程中，但在网络拓扑方面，二者的原理是一致的，所以下面只给出一个典型的 nova-network 的网络配置，有了上面的基础，现在看这段配置是否会感到很亲切呢？

清单 **6. /etc/nova/nova.conf** 中的网络配置示例

```
##### nova-network #####
network_manager=nova.network.manager.VlanManager
public_interface=eth1
vlan_interface=eth0
network_host=node1
fixed_range=10.0.0.0/8
network_size=1024
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
force_dhcp_release=True
fixed_ip_disassociate_timeout=30
my_ip=172.16.99.108
routing_source_ip=192.168.99.108
```

相关参数说明如下：

- network_manager，目前支持 VlanManager、FlatManager、FlatDHCPManager 三种拓扑
- public_interface, 接外网的物理网卡，floating ip 功能需要用到它
- valn_interface, 用于划分 VLAN 的物理网卡
- fixed_range, 服务网络，即虚机内部所用的网络地址
- my_ip，管理网络，用于安装 Openstack 组件的物理机之间的通信。例如：本实验中的控制节点同时具有外网网络地址 192.168.99.108 与管理网络地址

172.16.99.108，另一计算节点的管理网络地址为 172.16.99.109，所以 my_ip 应该设置为 172.16.99.108

- routing_source_ip, NAT 映射后的公共网络 IP，设置了此参数，会自动执行 NAT 命令：

```
iptables -t nat -A POSTROUTING -s 172.16.99.0/24 -o eth1 -j SNAT --to 192.168.99.108
```

显然，如果没有区分公共网络与管理网络，即它们处于同一网段的话，并不需要配置 my_ip 及 routing_source_ip 两个参数。

结论

云中网络一般被划分为公共网络、管理网络 & 存储网络与服务网络三大类。虚拟网络拓扑一般有 NAT、Bridge、VLAN 三种情形。我们手工一步一步地通过 NAT、Bridge、VLAN 三个试验简单实现了一个上述典型的云中网络。原理都是相通的，您再看 OpenStack 云中网络或才其他云的网络时都会倍感亲切。

参考资料

学习

- 参考 Devstack 官网，您可以从 devstack 脚本快速上手 Openstack。
- 参考 Openstack 官网，您可以获得更多关于 Openstack 的知识。
- 参考 Quantum Wiki, 您可以获得关于 Linux Bridge 网络的一般原理。
- “使用 OpenStack 实现云计算和存储”（developerWorks，2012 年 9 月）：Infrastructure as a Service (IaaS) 云平台种类繁多，例如像 Nebula 和 Eucalyptus 这样为人熟知的解决方案。而此领域的一个新来者已展示了其不俗的增长，不仅包括用户数量的增长，还包括支持公司的数量的大量增长。在本文中，我们将了解这个开源平台 OpenStack，发现它是否真的是一种开源云操作系统。
- developerWorks 云计算站点 提供了有关云计算的更新资源，包括
  - 云计算 简介。
  - 更新的 技术文章和教程，以及网络广播，让您的开发变得轻松，专家研讨会和录制会议 帮助您成为高效的云开发人员。
  - 连接转为云计算设计的 IBM 产品下载和信息。
  - 关于 社区最新话题 的活动聚合。
- 加入云计算讨论组，了解和讨论云计算的最新技术、解决方案、趋势等内容。

讨论

- 加入 developerWorks 中文社区。查看开发人员推动的博客、论坛、组和维基，并与其他 developerWorks 用户交流。