如果一个事务通过了有效性检查，提交者将使用写集来更新世界状态。在更新阶段，对于写集中的每个键，相同键的值都设置为在写集中指定的值，进一步地，这个世界状态的键的版本会被改变，以反映最新的版本。

3. 模拟和验证案例

本节通过一个示例场景帮助理解语义。对于本例的目的，在世界状态中，键 k 的存在是由元组(k、ver、val)表示的，其中，ver 是键 k 的最新版本，它的值由 val 表示。

现在，考虑一组 5 个事务：T1、T2、T3、T4 和 T5，它们都在同一个快照上模拟世界状态。下面的代码片段显示了对事务进行模拟的世界状态的快照，以及由这些事务执行的读取和写入活动的顺序。

```
World state: (k1,1,v1), (k2,1,v2), (k3,1,v3), (k4,1,v4), (k5,1,v5)
T1 -> Write(k1, v1'), Write(k2, v2')
T2 -> Read(k1), Write(k3, v3')
T3 -> Write(k2, v2'')
T4 -> Write(k2, v2'''), read(k2)
T5 -> Write(k6, v6'), read(k5)
```

现在，假设这些事务是在 T1 的序列中排序的。T5 可以包含在一个区块或不同的区块中：

- T1 通过验证，因为它不执行任何读取操作。此外，世界状态中的键 k1 和 k2 的元组被更新为(k1，2，v1')和(k2，2，v2')。
- T2 失败了，因为它读取了一个键 k1，它被之前的事务修改为 T1。
- T3 通过验证，因为它不执行读操作。进一步的，在这个世界状态下的键的元组被更新到(k2，3，v2")。
- T4 失败了，因为它读取了一个键 k2，它被之前的事务 T1 修改过。
- T5 通过验证，因为它读取了一个键 k5，它没有被前面的任何事务修改过。

注意：目前还不支持具有多个读写集的事务。

6.4 Kafka集群配置

有了前面 5 个章节的部署经验，再加上本章前三节的概述理论，将会容易理解本章的重点 Kafka 集群部署。

在进行本次方案之前，首先需要对 Kafka 集群的拓扑有些简单的了解。搭建 Kafka 集群的最小单位组成如下：

- 三个 Zookeeper 节点集群。
- 四个 Kafka 节点集群。
- 三个 Orderer 排序服务节点集群。
- 其他 Peer 节点。

以上集群至少需要 10 个服务节点提供集群服务，其余节点用于背书验证、提交及数据同步。

Kafka 是一个分布式消息系统，由 LinkedIn 使用 scala 编写，用作 LinkedIn 的活动流（Activity Stream）和运营数据处理管道（Pipeline）的基础。具有高水平扩展和高吞吐量。

在 Fabric 网络中，数据是由 Peer 节点提交到 Orderer 排序服务，而 Orderer 相对于 Kafka 来说相当于上游模块，且 Orderer 还兼具提供了对数据进行排序及生成符合配置规范及要求的区块。而使用上游模块的数据计算、统计、分析，这个时候就可以使用类似于 Kafka 这样的分布式消息系统来协助业务流程。

有人说 Kafka 是一种共识模式，也就是说平等信任，所有的 HyperLedger Fabric 网络加盟方都是可信方，因为消息总是均匀地分布在各处。但具体生产使用的时候是依赖于背书来做到确权，相对而言，Kafka 应该只能是一种启动 Fabric 网络的模式或类型。

Zookeeper 是一种在分布式系统中被广泛用来作为分布式状态管理、分布式协调管理、分布式配置管理和分布式锁服务的集群。Kafka 增加和减少服务器都会在 Zookeeper 节点上触发相应的事件，Kafka 系统会捕获这些事件，进行新一轮的负载均衡，客户端也会捕获这些事件来进行新一轮的处理。

Orderer 排序服务是 Fabric 网络事务流中的最重要的环节，也是所有请求的终点，它并不会立刻对请求给予回馈，一是因为生成区块的条件所限，二是因为依托下游集群的消息处理需要等待结果。Kafka 集群拓扑图如图 6-8 所示。

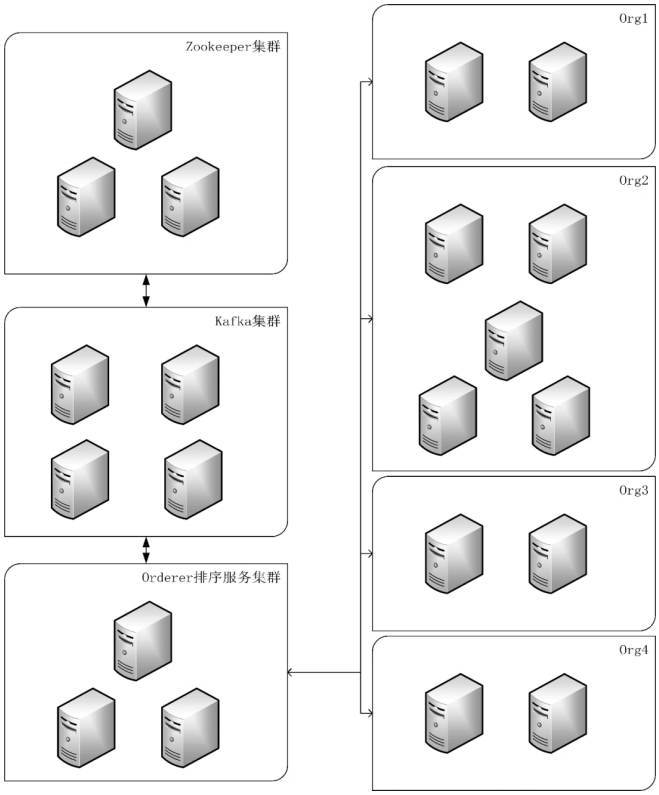

图6-8　Kafka集群拓扑图

本书根据实际需求，启用了如下 12 台服务器，见表 6-1。

表 6-1　集群配置表

名　　称	IP	Hostname	组织机构
Zk1	172.31.159.137	zookeeper1	
Zk2	172.31.159.135	zookeeper2	
Zk3	172.31.159.136	zookeeper3	
Kafka1	172.31.159.133	kafka1	
Kafka2	172.31.159.132	kafka2	
Kafka3	172.31.159.134	kafka3	
Kafka4	172.31.159.131	kafka4	
Orderer0	172.31.159.130	orderer0.example.com	
Orderer1	172.31.143.22	orderer1.example.com	
Orderer2	172.31.143.23	orderer2.example.com	
peer0	172.31.159.129	peer0.org1.example.com	Org1
peerX	172.31.143.21	x.x.example.com	Org2

如同前述拓扑图及 Kafka 集群网络最小单元所述，所启用的服务器为最小配置单元。如果考虑到高可用性，可以自行学习参考 K8s 管理 Docker 的方案。

12 台服务器中 peer0 服务器是 peer0org1 节点服务器，而 peerX 则是任意可选节点服务器，用于测试各种节点的可能性。

在这些服务器当中，每一台都会安装 Docker、Docker-Compose 环境，而 Orderer 排序服务器及 Peer 节点服务器会额外地安装 Go 语言及 Fabric 环境。

当需要生成必要的配置文件时，任选其中一台部署有 Fabric 环境的服务器即可，本书选择 Orderer0 排序服务器作为配置文件生成服务器，即 172.31.159.130 服务器。

注意：*在没有特殊说明的情况下，所有的安装有 Fabric 环境的服务器部署方案均与第 5 章保持一致。*

6.4.1　crypto-config.yaml配置

前面章节中的 crypto-config.yaml 等没有被贴出来的配置文件都是采用 Fabric1.0 版本源码 e2e_cli 案例中的，从 Kafka 集群配置开始，将要准备编写属于自己的 crypto-config.yaml 配置文件。

还是先从源码开始，如下所示：

```
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer0
      - Hostname: orderer1
      - Hostname: orderer2

PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2
    Users:
      Count: 1

  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 2
```

```
    Users:
      Count: 1
    Specs:
      - Hostname: foo
        CommonName: foo27.org2.example.com
      - Hostname: bar
      - Hostname: baz


  - Name: Org3
    Domain: org3.example.com
    Template:
      Count: 2
    Users:
      Count: 1

  - Name: Org4
    Domain: org4.example.com
    Template:
      Count: 2
    Users:
      Count: 1

  - Name: Org5
    Domain: org5.example.com
    Template:
      Count: 2
    Users:
      Count: 1
```

在上述配置源码中，Org1 即组织 1 的配置，与第 5 章的一样，Org3、Org4 和 Org5 是新增的组织配置信息，结构与 Org1 一致。与拓扑图相比，这里多出了一个 Org5，但普通 Peer 节点的多少并不影响集群部署的整体方案。

其中，Org2 加入了一些小的变化，采用了一种混合编排组织配置的方案，既有模板配置也有自定义配置。模板配置内容与 Org1、Org3、Org4 及 Org5 一样。自定义配置主要是新增了三个节点且名称分别是 foo、bar 和 baz，而 foo 又自定义了其生成的具体节点名称后缀，姑且理解为编号。

将编写完成的配置文件上传至 172.31.159.130 服务器/home/docker/github.com/hyperledger/fabric/aberic 目录下，并执行如下命令生成节点所需配置文件：

```
./bin/cryptogen generate --config=./crypto-config.yaml
```

根据第 4.1 节所述，该命令执行完成后会在/home/docker/github.com/hyperledger/fabric/aberic/crypto-config/peerOrganizations 目录下看到所有的组织配置文件，由于在编写 org2 的时候做了一些自定义，可以进入/home/docker/github.com/hyperledger/fabric/aberic/crypto-config/peerOrganizations/org2.example.com/peers 目录下查看自定义节点的目录信息，如图 6-9 所示。

图6-9　org2节点信息

在图中，除了默认模板生成的节点配置文件外，还有在 crypto-config.yaml 自定义的三个节点配置文件，说明自定义配置文件属性无误。且继续观察目录结构和内容可以发现，org1、org3、org4 和 org5 的结构是一致的，都是通过模板配置生成的。

6.4.2　configtx配置

通过第 4、5 两章的内容，知道了 configtx.yaml 配置文件在上下文中需要与 crypto-config.yaml 配置文件相匹配，同时该配置用于生成创世区块及设定 Fabric 网络启动类型。

由于本次采用的是 Kafka 集群部署，所以在本次文件配置中的启动类型应该为“kafka”。此外，还需要在 Addresses 中将 Orderer 可用排序服务即集群排序服务器的地址补全，在 Kafka 的 Brokers 中可填写非全量 Kafka 集群所用服务器 IP 或域名。

在组织配置中，需要将新增的三个组织配置编写入内，且所设置的锚节点必须是在生成范围中的某一节点。

具体的配置源码如下：

```
Profiles:

    TwoOrgsOrdererGenesis:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
        Consortiums:
            SampleConsortium:
```