二

08 集群模式：服务器如何从初始化到对外提供服务?

通过上个课时的学习，我们知道了 ZooKeeper 在单机模式下从启动运行到对外提供服务的整个过程。而在日常工作中，无论是出于性能上的优势还是可靠性的考虑，单机模式都无法满足要求。因此，ZooKeeper 也采用集群的方式运行。本课时我们就来学习一下 ZooKeeper 集群模式下，启动过程的底层实现。

什么是集群模式?

为了解决单机模式下性能的瓶颈问题，以及出于对系统可靠性的高要求，集群模式的系统架构方式被业界普遍采用。那什么是集群模式呢？集群模式可以简单理解为将单机系统复制成几份，部署在不同主机或网络节点上，最终构成了一个由多台计算机组成的系统“集群”。而组成集群中的每个服务器叫作集群中的网络节点。

到现在我们对集群的组织架构形式有了大概的了解，那么你可能会产生一个问题：我们应该如何使用集群？当客户端发送一个请求到集群服务器的时候，究竟是哪个机器为我们提供服务呢？为了解决这个问题，我们先介绍一个概念名词“调度者”。调度者的工作职责就是在集群收到客户端请求后，根据当前集群中机器的使用情况，决定将此次客户端请求交给哪一台服务器或网络节点进行处理，例如我们都很熟悉的负载均衡服务器就是一种调度者的实现方式。

ZooKeeper 集群模式的特点

通过上面的介绍，我们知道了集群是由网络中不同机器组成的一个系统，集群工作是通过集群中调度者服务器来协同工作的。那么现在我们来看一下 ZooKeeper 中的集群模式，在 ZooKeeper 集群模式中，相比上面说到的集群架构方式，在 ZooKeeper 集群中将服务器分成 Leader 、Follow 、Observer 三种角色服务器，在集群运行期间这三种服务器所负责的工作各不相同：

- Leader 角色服务器负责管理集群中其他的服务器，是集群中工作的分配和调度者。
- Follow 服务器的主要工作是选举出 Leader 服务器，在发生 Leader 服务器选举的时

候，系统会从 Follow 服务器之间根据多数投票原则，选举出一个 Follow 服务器作为新的 Leader 服务器。

- Observer 服务器则主要负责处理来自客户端的获取数据等请求，并不参与 Leader 服务器的选举操作，也不会作为候选者被选举为 Leader 服务器。

接下来我们看看在 ZooKeeper 中集群是如何架构和实现的。

底层实现原理

到目前为止我们对 ZooKeeper 中集群相关的知识有了大体的了解，接下来我们就深入到 ZooKeeper 的底层，看看在服务端，集群模式是如何启动到对外提供服务的。

在上一课时中，我们已经对 ZooKeeper 单机版服务的启动过程做了详细的介绍。而集群中的启动过程和单机版的启动过程有很多地方是一样的。所以本次我们只对 ZooKeeper 集群中的特有实现方式做重点介绍。

程序启动

首先，在 ZooKeeper 服务启动后，系统会调用入口 QuorumPeerMain 类中的 main 函数。在 main 函数中的 initializeAndRun 方法中根据 zoo.cfg 配置文件，判断服务启动方式是集群模式还是单机模式。在函数中首先根据 arg 参数和 config.isDistributed() 来判断，如果配置参数中配置了相关的配置项，并且已经指定了集群模式运行，那么在服务启动的时候就会跳转到 runFromConfig 函数完成之后的集群模式的初始化工作。

```
protected void initializeAndRun(String[] args){

  ...

  if (args.length == 1 && config.isDistributed()) {

            runFromConfig(config);

        } else {


            ZooKeeperServerMain.main(args);

        }

}
```

QuorumPeer 类

在 ZooKeeper 服务的集群模式启动过程中，一个最主要的核心类是 QuorumPeer 类。我们可以将每个 QuorumPeer 类的实例看作集群中的一台服务器。在 ZooKeeper 集群模式的运行中，一个 QuorumPeer 类的实例通常具有 3 种状态，分别是参与 Leader 节点的选举、作为 Follow 节点同步 Leader 节点的数据，以及作为 Leader 节点管理集群中的 Follow 节点。

介绍完 QuorumPeer 类后，下面我们看一下在 ZooKeeper 服务的启动过程中，针对 QuorumPeer 类都做了哪些工作。如下面的代码所示，在一个 ZooKeeper 服务的启动过程中，首先调用 runFromConfig 函数将服务运行过程中需要的核心工具类注册到 QuorumPeer 实例中去。

这些核心工具就是我们在上一节课单机版服务的启动中介绍的诸如 FileTxnSnapLog 数据持久化类、ServerCnxnFactory 类 NIO 工厂方法等。这之后还需要配置服务器地址列表、Leader 选举算法、会话超时时间等参数到 QuorumPeer 实例中。

```
public void runFromConfig(QuorumPeerConfig config){
  ServerCnxnFactory cnxnFactory = null;
  ServerCnxnFactory secureCnxnFactory = null;
  ...
  quorumPeer = getQuorumPeer()
  quorumPeer.setElectionType(config.getElectionAlg());
  quorumPeer.setCnxnFactory(cnxnFactory);
  ...
}
```

与开篇中提到的一般构建集群的方式不同，ZooKeeper 将集群中的机器分为 Leader 、Follow 、Obervser 三种角色，每种角色服务器在集群中起到的作用都各不相同。比如 Leader 角色服务器主要负责处理客户端发送的数据变更等事务性请求操作，并管理协调集群中的 Follow 角色服务器。而 Follow 服务器则主要处理客户端的获取数据等非事务性请求操作。Observer 角色服务器的功能和 Follow 服务器相似，唯一的不同就是不参与 Leader 头节点服务器的选举工作。

在 ZooKeeper 中的这三种角色服务器，在服务启动过程中也有各自的不同，下面我们就以 Leader 角色服务器的启动和 Follow 服务器服务的启动过程来看一下各自的底层实现原理。

Leader 服务器启动过程

在 ZooKeeper 集群中，Leader 服务器负责管理集群中其他角色服务器，以及处理客户端的数据变更请求。因此，在整个 ZooKeeper 服务器中，Leader 服务器非常重要。所以在整个 ZooKeeper 集群启动过程中，首先要先选举出集群中的 Leader 服务器。

在 ZooKeeper 集群选举 Leader 节点的过程中，首先会根据服务器自身的服务器 ID（SID）、最新的 ZXID、和当前的服务器 epoch （currentEpoch）这三个参数来生成一个选举标准。之后，ZooKeeper 服务会根据 zoo.cfg 配置文件中的参数，选择参数文件中规定的 Leader 选举算法，进行 Leader 头节点的选举操作。而在 ZooKeeper 中提供了三种 Leader 选举算法，分别是 LeaderElection 、AuthFastLeaderElection、FastLeaderElection。在我们日常开发过程中，可以通过在 zoo.cfg 配置文件中使用 electionAlg 参数属性来制定具体要使用的算法类型。具体的 Leader 选举算法我们会在之后的章节中展开讲解。

这里我们只需要知道，在 ZooKeeper 集群模式下服务启动后。首先会创建用来选举 Leader 节点的工具类 QuorumCnxManager 。下面这段代码给出了 QuorumCnxManager 在创建实例的时候首先要实例化 Listener 对象用于监听 Leader 选举端口。

```
package org.apache.zookeeper.server.quorum;

public class QuorumCnxManager {

...

  public QuorumCnxManager(QuorumPeer self) {

    String cnxToValue = System.getProperty("zookeeper.cnxTimeout")

    listener = new Listener();

    listener.setName("QuorumPeerListener");

  }

 ...

}
```

而在 ZooKeeper 中，Leader 选举的大概过程，总体说来就是在集群中的所有机器中直接进行一次选举投票，选举出一个最适合的机器作为 Leader 节点。而具体的评价标准就是我们上面提到的三种选举算法。而从 3.4.0 版本开始，ZooKeeper 只支持 FastLeaderElection 这一种选举算法。同时没有被选举为 Leader 节点的机器则作为 Follow 或 Observer 节点机器存在。

Follow 服务器启动过程

现在，我们已经选举出 ZooKeeper 集群模式下的 Leader 节点机器了。我们再看一下 Follow 节点机器在 ZooKeeper 集群模式下服务器的启动过程。

在服务器的启动过程中，Follow 机器的主要工作就是和 Leader 节点进行数据同步和交互。当 Leader 机器启动成功后，Follow 节点的机器会收到来自 Leader 节点的启动通知。而该通知则是通过 LearnerCnxAcceptor 类来实现的。该类就相当于一个接收器。专门用来接收来自集群中 Leader 节点的通知信息。下面这段代码中 LearnerCnxAcceptor 类首先初始化要监听的 Leader 服务器地址和设置收到监听的处理执行方法等操作 。

```
class LearnerCnxAcceptor extends ZooKeeperCriticalThread {

    private volatile boolean stop = false;

    public LearnerCnxAcceptor() {

        super("LearnerCnxAcceptor-" + ss.getLocalSocketAddress(), zk

                .getZooKeeperServerListener());

    }

 }
```

在接收到来自 Leader 服务器的通知后，Follow 服务器会创建一个 LearnerHandler 类的实例，用来处理与 Leader 服务器的数据同步等操作。

```
package org.apache.zookeeper.server.quorum;

public class LearnerHandler extends ZooKeeperThread {

  protected final Socket sock;

  final Leader leader;

  ...

}
```

在完成数据同步后，一个 ZooKeeper 服务的集群模式下启动的关键步骤就完成了，整个服务就处于运行状态，可以对外提供服务了。

小结

本课时我们主要学习了 ZooKeeper 集群模式的启动过程和底层实现。与一般的集群架构不同，ZooKeeper 集群模式把其中的机器分成 Leader 、Follow 、Obsever 三种角色。当作

为 Leader 节点的机器失效时，系统会根据配置文件中的选举算法产生新的节点。这种方式避免了一般集群模式中产生的单点失效等问题。

现在，我们思考这样一个问题。在我们日常使用 ZooKeeper 集群服务器的时候，集群中的机器个数应该如何选择？

答案是最好使用奇数原则，最小的集群配置应该是三个服务器或者节点。而如果采用偶数，在 Leader 节点选举投票的过程中就不满足大多数原则，这时就产生“脑裂”这个问题。请你多注意。

上一页 下一页