实验4 部署ZooKeeper集群和实战ZooKeeper

本实验的知识地图如图4-1所示（表示重点表示难点）。

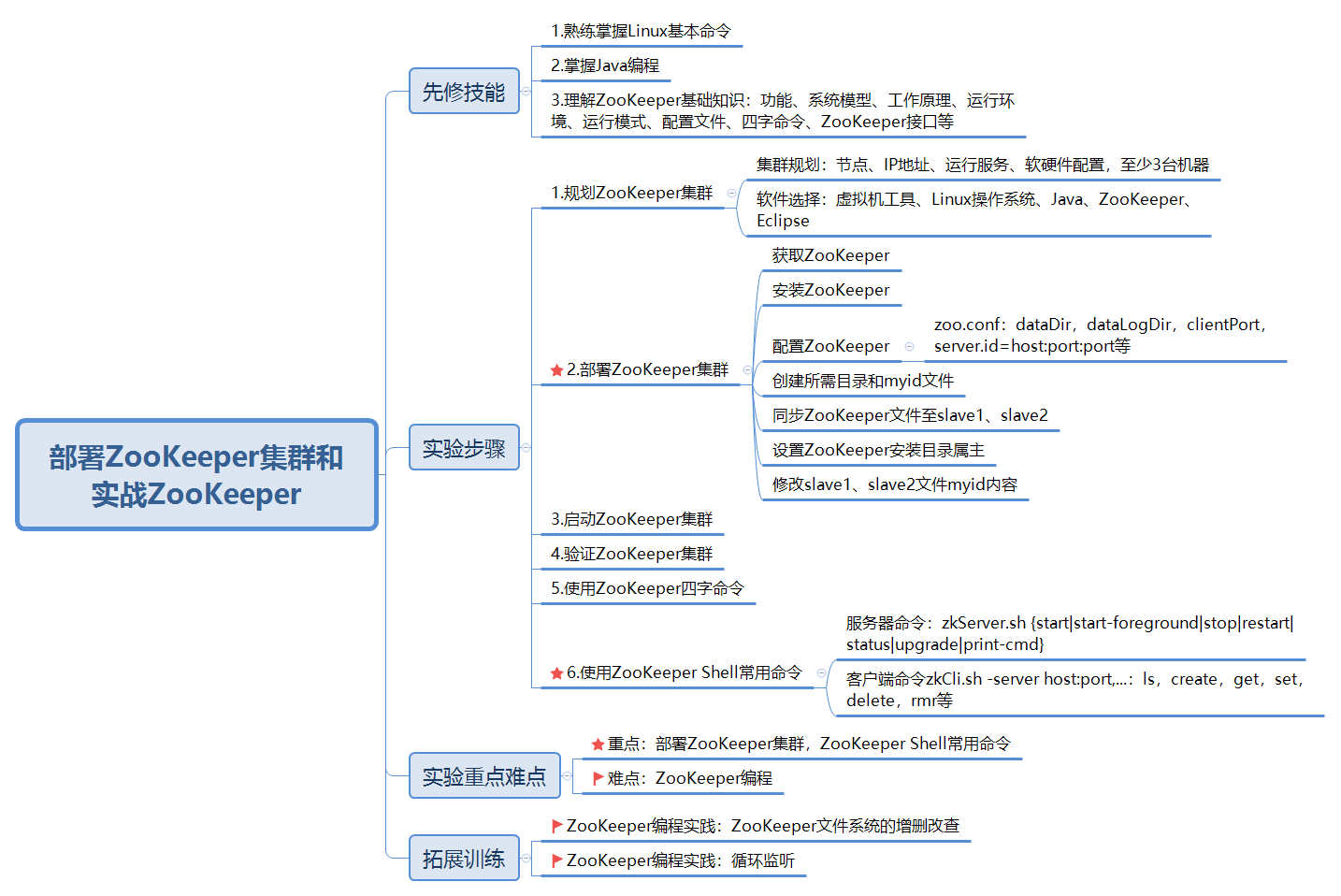


图4-1 实验4部署ZooKeeper集群和实战ZooKeeper知识地图

一、实验目的

1. 理解ZooKeeper的系统模型，包括数据模型、版本机制、Watcher监听机制、ACL权限控制机制。

2. 理解ZooKeeper的工作原理，包括集群架构、Leader选举机制。

3. 熟练掌握ZooKeeper集群的部署和运行。

4. 掌握ZooKeeper四字命令的使用。

5. 熟练掌握ZooKeeper Shell常用命令的使用。

6. 了解ZooKeeper Java API，能看懂简单的ZooKeeper编程。

二、实验环境

本实验所需的软件环境包括Linux集群（至少3台机器）、Oracle JDK 1.6+、ZooKeeper安装包、Eclipse。

三、实验内容

1. 规划ZooKeeper集群。

2. 部署ZooKeeper集群。

3. 启动ZooKeeper集群。

4. 验证ZooKeeper集群。

5. 使用ZooKeeper四字命令。

6. 使用ZooKeeper Shell常用命令。

7. 关闭ZooKeeper集群。

四、实验原理

Apache ZooKeeper是一个分布式的、开放源码的分布式应用程序协调框架，是Google Chubby的开源实现，它为大型分布式系统中的各种协调问题提供了一个解决方案，主要用于解决分布式应用中经常遇到的一些数据管理问题，如配置管理、命名服务、分布式同步、集群管理等。

（一）ZooKeeper系统模型

1. ZooKeeper数据模型

ZooKeeper采用类似标准文件系统的数据模型，其节点构成了一个具有层次关系的树状结构，如图4-2所示。其中每个节点被称为数据节点ZNode，ZNode是ZooKeeper中数据的最小单元，每个节点上可以存储数据，同时也可以挂载子节点，因此构成了一个层次化的命名空间。

/

/app1

/app2

/app1/p\_1

/app1/p\_2

/app1/p\_3

图4-2 ZooKeeper数据模型

ZNode通过路径引用，如同Unix中的文件路径。路径必须是绝对的，因此它们必须由斜杠“/”来开头。

在ZooKeeper中，每个数据节点都是有生命周期的，其生命周期的长短取决于数据节点的节点类型。ZNode类型在创建时即被确定，并且不能改变。节点可以分为持久节点（PERSISTENT）、临时节点（EPHEMERAL）和顺序节点（SEQUENTIAL）三大类型。在节点创建过程中，通过组合使用，可以生成以下四种组合型节点类型：

（1）持久节点PERSISTENT：持久节点是ZooKeeper中最常见的一种节点类型。所谓持久节点，是指此类节点的生命周期不依赖于会话，自节点被创建就会一直存在于ZooKeeper服务器上，并且只有在客户端显式执行删除操作时，它们才能被删除。

（2）持久顺序节点PERSISTENT\_SEQUENTIAL：持久顺序节点的基本特性与持久节点相同，额外特性表现在顺序性上。在ZooKeeper中，每个父节点都会为它的第一级子节点建立一个文件，用于记录每个子节点创建的先后顺序。基于这个顺序特性，在创建子节点的时候，可以设置这个标记，那么在创建节点过程中，ZooKeeper会自动为给定节点名加上一个数字后缀，作为一个新的、完整的节点名。不过ZooKeeper会给此类节点名称进行顺序编号，自动在给定节点名后加上一个数字后缀。这个数字后缀的上限是整型的最大值，其格式为“%10d”（10位数字，没有数值的数位用0补充，例如“0000000001”），当计数值大于232-1时，计数器将溢出。

（3）临时节点EPHEMERAL：与持久节点不同的是，临时节点的生命周期依赖于创建它的会话，也就是说，如果客户端会话失效，临时节点将被自动删除，当然也可以手动删除。注意，这里提到的是客户端会话失效，而非TCP连接断开。另外，ZooKeeper规定临时节点不允许拥有子节点。

（4）临时顺序节点EPHEMERAL\_SEQUENTIAL：临时顺序节点的基本特性和临时节点也是一致的，同样是在临时节点的基础上，添加了顺序的特性。

一个ZNode除了存储数据内容，还存储了许多表示其自身状态的重要信息，其状态信息使用Stat对象存放。关于数据节点的Stat对象所有状态属性的说明如表4-1所示。

表4-1 节点Stat对象状态属性说明

|  |  |
| --- | --- |
| 状态属性 | 说明 |
| czxid | 数据节点创建时的事务ID |
| mzxid | 数据节点最后一次更新时的事务ID |
| ctime | 数据节点创建时的时间 |
| mtime | 数据节点最后一次更新时的时间 |
| version | 数据节点数据内容的版本号 |
| cversion | 数据节点子节点的版本号 |
| aversion | 数据节点的ACL版本号 |
| ephemeralOwner | 如果节点是临时节点，则表示创建该节点的会话的SessionID；如果节点是持久节点，则该属性值为0 |
| dataLength | 数据内容的长度 |
| numChildren | 数据节点当前的子节点个数 |
| pzxid | 数据节点的子节点列表最后一次被修改（子节点列表的变更而非子节点内容的变更）时的事务ID |

2. ZNode版本

ZooKeeper中为数据节点引入了版本的概念，每个数据节点都具有三种类型的版本信息，对数据节点的任何更新操作都会引起版本号的编号。这三种类型的版本信息前文已介绍到，分别为：

（1）version：当前数据节点数据内容的版本号。

（2）cversion：当前数据节点子节点的版本号。

（3）aversion：当前数据节点的ACL版本号。

ZooKeeper中的版本概念和传统意义上的软件版本有很大区别，它表示的是对数据节点的数据内容、子节点列表、或是节点ACL信息的修改次数。以version这种版本类型为例，在一个数据节点“/xijing”被创建完毕之后，节点的version是0，表示“当前节点自从创建之后，被更新过0次”，如果现在对该节点的数据内容进行更新操作，那么随后，version的值就会变成1。同时需要注意的是，在上文中提到的关于version的说明，其表示的是数据节点数据内容的变更次数，强调的是变更次数，因此即使前后两次变更并没有使得数据内容的值发生变化，version的值依然会变更。

ZooKeeper中之所以引入版本概念，是因为版本可以保证分布式数据的原子性操作，version属性可以用来实现乐观锁机制中的“写入校验”，在写入校验阶段，事务会检查数据在读取阶段后是否有其他事务对数据进行过更新，以确保数据更新的一致性。

3. Watcher监听机制

在ZooKeeper中，引入了Watcher机制来实现分布式的通知功能。ZooKeeper允许客户端向服务端注册一个Watcher监听，当服务端的一些指定事件触发了这个Watcher，那么就会向指定客户端发送一个事件通知来实现分布式的通知功能。

ZooKeeper的Watcher机制主要包括客户端线程、客户端WatcherManager和ZooKeeper服务器三部分。在工作流程上，简单地讲，客户端在向ZooKeeper服务器注册的同时，会将Watcher对象存储在客户端的WatcherManager当中。当ZooKeeper服务器触发Watcher事件后，会向客户端发送通知，客户端线程从WatcherManager中取出对应的Watcher对象来执行回调逻辑。整个Watcher注册与通知过程如图4-3所示。

ZooKeeper

Client

WatcherManager

①注册

②存储

③通知

图4-3 ZooKeeper Watcher机制概述

在ZooKeeper中，接口类Watcher用于表示一个标准的事件处理器，其定义了事件通知相关的逻辑，包含KeeperState和EventType两个枚举类，分别代表了通知状态和事件类型，同时定义了事件的回调方法：process(WatchedEvent event)。同一事件类型在不同的通知状态中代表的含义有所不同，表4-2列举了常见的通知状态和事件类型。

表4-2 Watcher通知状态与事件类型一览表

|  |  |  |  |
| --- | --- | --- | --- |
| 通知状态 | 事件类型 | 触发条件 | 说明 |
| SyncConnected | None | 客户端与服务器成功建立会话 | 此时客户端和服务器处于连接状态 |
| NodeCreated | Watcher监听的对应数据节点被创建 |
| NodeDeleted | Watcher监听的对应数据节点被删除 |
| NodeDataChanged | Watcher监听的对应数据节点的数据内容发生变更 |
| NodeChildrenChanged | Watcher监听的对应数据节点的子节点列表发生变更 |
| Disconnected | None | 客户端与ZooKeeper服务器断开连接 | 此时客户端和服务器处于断开连接状态 |
| Expired | None | 会话超时 | 此时可客户端会话失效，通常同时也会收到SessionExpiredException |
| AuthFailed | None | 通常有两种情况：  （1）使用错误的scheme进行权限检查  （2）SASL权限检查失败 | 通常同时也会收到AuthFailedException |

其中，NodeDataChanged事件的触发条件是数据内容的变化变更，此处所说的变更包括节点的数据内容和数据版本号version的变化，因此，即使使用相同的数据内容来更新，还是会触发这个事件，因为对于ZooKeeper来说，无论数据内容是否变更，一旦有客户端调用了数据更新的接口且更新成功，就会更新version值。

NodeChildrenChanged事件会在数据节点的子节点列表发生变更的时候被触发，这里说的子节点变化特指子节点个数和组成情况的变更，即新增子节点或删除子节点，而子节点数据内容的变化时不会触发这个事件的。

对于AuthFailed事件，需要注意的是，它的触发条件并不是简单因为当前客户端会话没有权限，而是授权失败。

4. ACL权限控制机制

ZooKeeper提供了一套完善ACL（Access Control List）权限控制机制来保障数据的安全。

ACL，即访问控制列表，是一种相对来说比较新颖且更细粒度的权限管理方式，可以针对任意用户和组进行细粒度的权限控制。目前绝大部分Unix系统都已经只支持ACL，Linux也从2.6版本的内核开始支持这ACL权限控制方式。ZooKeeper的ACL权限控制和Unix/Linux操作系统中的ACL有一些区别，读者可以从三个方面来理解ACL机制，分别是：权限模式（Scheme）、授权对象（ID）和权限（Permission），通常使用“scheme:id:permission”来标识一个有效的ACL信息。

权限模式（Scheme）用来确定权限验证过程中使用的检验策略，授权对象（ID）指的是权限赋予的用户或一个指定实体，例如IP地址或机器等。在不同的授权模式下，授权对象是不同的，表4-3中列出了各个权限模式和授权对象之间的对应关系。

表4-3 权限模式和授权对象之间对应关系

|  |  |
| --- | --- |
| 权限模式 | 授权对象 |
| IP | 通常是一个IP地址或IP段，例如“ip:192.168.18.130”或“ip:192.168.18.1/24” |
| Digest | 自定义，通常是“username:BASE64(SHA-1(username:password))” |
| World | 只有一个ID“anyone” |
| Super | 与Digest模式一致 |

权限（Permission）就是指那些通过权限检查后可以被允许执行的操作。在ZooKeeper中，所有对数据的操作权限分为以下五大类：

（1）CREATE（C）：数据节点的创建权限，允许授权对象在该数据节点下创建子节点。

（2）DELETE（D）：子节点的删除权限，允许授权对象删除该数据接地点的子节点。

（3）READ（R）：数据节点的读取权限，允许授权对象访问该数据节点并读取其数据内容或子节点列表等。

（4）WRITE（W）：数据节点的更新权限，允许授权对象对该数据节点进行更新操作。

（5）ADMIN（A）：数据节点的管理权限，允许授权对象对数据节点进行ACL相关的设置操作。

（二）ZooKeeper工作原理

1. ZooKeeper集群架构

ZooKeeper运行模式有两种：单机模式（Standalone Mode）、集群模式（Replicated Mode）。其中，单机模式主要用于评估、开发和测试，而在实际的生产环境中均采用集群模式。

ZooKeeper单机模式部署时，只在一台机器上安装ZooKeeper，该ZooKeeper提供一切协调服务；ZooKeeper单机模式运行时该ZooKeeper就是Leader。

ZooKeeper集群模式部署时，在多台机器上安装ZooKeeper，ZooKeeper采用对等结构，无Master、Slave之分，统一都是QuorumPeerMain进程；ZooKeeper集群模式运行时采取选举方式选择Leader，采用原子广播协议ZAB完成，此协议是对Paxos算法的修改与实现，获得n/2+1票数的服务器成为Leader，其余节点是Follower。当发生客户端读写操作时，读操作可在所有节点上实现，但写操作必须经Leader同意后方可执行。若有新加入的服务器，则该服务器发起一次选举，如果该服务器获得n/2+1个票数，则此服务器将成为整个ZooKeeper集群的Leader，当Leader发生故障时，剩下的Follower将重新进行新一轮Leader选举。ZooKeeper集群模式架构如图4-4所示。

ZooKeeper集群

Client

Client

Client

Client

Client

Client

Client

Client

Server

（Observer）

Server

（Follower）

Server

（Leader）

Server

（Follower）

Server

（Observer）

数据交换

选举

图4-4 ZooKeeper集群模式架构

Leader选举时要求“可用节点数量>总节点数量/2”，即ZooKeeper集群中的存活节点必须过半。因此，在节点数量是奇数的情况下，ZooKeeper集群总能对外提供服务（即使损失了一部分节点）；如果节点数量是偶数，会存在ZooKeeper集群不能用的可能性。在生产环境中，如果ZooKeeper集群不能提供服务，那将是致命的，所以ZooKeeper集群的节点数一般采用奇数。

每个Server在工作过程中有四种状态：

（1）LOOKING：竞选状态。

（2）FOLLOWING：随从状态，同步Leader状态，参与投票。

（3）OBSERVING：观察状态，同步Leader状态，不参与投票。

（4）LEADING：领导者状态。

关于ZooKeeper集群中的各个角色的介绍如下。

（1）Leader：Leader服务器是整个ZooKeeper集群工作机制中的核心，其主要工作包括以下两个方面：

* 事务请求的唯一调度和处理者，保证集群事务处理的顺序性。
* 集群内部各服务器的调度者。

（2）Follower：Follower服务器是ZooKeeper集群状态的跟随者，其主要工作包括以下三个方面：

* 处理客户端非事务请求，转发事务请求给Leader服务器。
* 参与事务请求Proposal的投票。
* 参与Leader选举投票。

（3）Observer：Observer是ZooKeeper自3.3.0版本开始引入的一个全新的服务器角色。从字面意思看，该服务器充当了一个观察者的角色——其观察ZooKeeper集群的最新状态变化并将这些状态变更同步过来。Observer服务器在工作原理上和Follower基本是一致的，对于非事务请求，都可以进行独立的处理，而对于事务请求，则会转发给Leader服务器进行处理。和Follower唯一的区别在于，Observer不参与任何形式的投票，包括事务请求Proposal的投票和Leader的选举投票。简单地说，Observer服务器只提供非事务服务，通常用于在不影响集群事务处理能力的前提下提升集群的非事务处理能力。

2. Leader选举机制

Leader选举机制是ZooKeeper中最重要的技术之一，也是保证分布式数据一致性的关键所在。

1）选举机制中的术语

（1）SID：服务器ID。SID是一个数字，用来唯一标识一台ZooKeeper集群中机器，每台机器不能重复，和myid值一致。

（2）ZXID：事务ID。ZXID用来唯一标识一次服务器状态的变更。在某一个时刻，集群中的每台服务器的ZXID值不一定全都一致，这和ZooKeeper服务器对于客户端“更新请求”的处理逻辑有关。

（3）Vote：投票。Leader选举必须通过投票来实现。当集群中的机器发现自己无法检测到Leader机器时，就会开始尝试进行投票。

（4）Quorum：过半机器数。这是整个Leader选举算法中最重要的一个术语，我们可以把它理解为一个量词，指的是ZooKeeper集群中过半的机器数，如果集群中总的机器数是n的话，quorum计算公式为：quorum = n/2 + 1。

（5）服务器状态：服务器状态有4种，分别是LOOKING竞选状态，FOLLOWING随从状态，OBSERVING观察状态，LEADING领导者状态。

2）Leader选举概述

读者需要注意的一点是，ZooKeeper集群至少拥有2台机器，这里，我们以3台机器组成的服务器集群为例介绍，假设3台机器的myid依次为1、2、3，称它们依次为Server1、Server 2和Server3，那么Server1的SID为1，Server2的SID为2，Server3的SID为3。

（1）服务器启动时期的Leader选举

在服务器集群初始化阶段，当只有服务器Server1启动时，它是无法完成Leader选举的，当第二台服务器Server2也启动后，此时这两台机器已经能够进行互相通信，每台机器都试图找到一个Leader，于是便进入了Leader选举流程。

① 每个Server会发出一个投票。由于是初始状态，因此Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票包含的最基本的元素包括：所推举的服务器的SID和ZXID，以（SID，ZXID）形式表示。因为是初始化阶段，因此无论是Server1还是Server2，都会投给自己，即Server1的投票为（1，0），Server2的投票为（2，0），然后各自将这个投票发给集群中其他所有机器。

② 接收来自各个服务器的投票。每个服务器都会接收来自其他服务器的投票。集群中的每个服务器在接收到投票后，首先会判断该投票的有效性，包括检查是否是本轮投票、是否来自LOOKING状态的服务器。

③ 处理投票。在接收到来自其他服务器的投票后，针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下：

* 优先检查ZXID。ZXID比较大的服务器优先作为Leader。
* 如果ZXID相同，那么就比较SID。SID比较大的服务器作为Leader服务器。

根据以上规则，对于Server1，它自己的投票是（1，0），而接收到的投票为（2，0）。首先对比两者的ZXID，因为都是0，所以无法决定谁是Leader；接下来会对比两者的SID，很显然，Server1发现接收到的投票中的SID是2，大于自己，于是就会更新自己的投票为（2，0），然后重新将投票发出去。而对于Server2来说，不需要更新自己的投票信息，只是再一次向集群中所有机器发出上一次投票信息即可。

④ 统计投票。每次投票后，服务器都会统计所有投票，判断是否已经有过半的机器收到相同的投票信息。对于Server1和Server2服务器来说，都统计出集群中已经有2台机器接受了（2，0）这个投票信息，对于由3台机器构成的集群，2台即达到了“过半”要求（≥n/2+1）。那么，当Server1和Server2都收到相同的投票信息（2，0）时，即认为已经选出了Leader。

⑤ 改变服务器状态。一旦确定了Leader，每个服务器都会更新自己的状态：如果是Follower，那么就变更为FOLLOWING；如果是Leader，那么久变更为LEADING。

（2）服务器运行期间的Leader选举

在ZooKeeper集群正常运行过程中，一旦选出一个Leader，那么所有服务器的集群角色一般不会再发生变化——也就是说，Leader服务器将一直作为集群的Leader，即使集群中有非Leader宕机或是有新机器加入集群，也不会影响Leader。但是一旦Leader宕机，那么整个集群将暂时无法对外服务，而是进入新一轮的Leader选举。服务器运行期间的Leader选举和启动时期的Leader选举基本过程是一致的。

假设当前正在运行的ZooKeeper集群有Server1、Server2、Server3三台机器组成，当前Leader是Server2，假设某一瞬间，Leader宕机，这个时候便开始了新一轮Leader选举，具体过程如下所示。

① 变更服务器状态。当Leader宕机后，余下的非Observer服务器就会将自己的服务器状态变更为LOOKING，然后开始进入Leader选举流程。

② 每个Server都会发出一个投票。在这个过程中，需要生成投票信息（SID，ZXID）。因为是运行期间，因此每个服务器上的ZXID可能不同，假定Server1的ZXID为123，而Server3的ZXID为122。在第一轮投票中，Server1和Server3都会投自己，即分别产生投票（1，123）和（3，122），然后各自将投票发给集群中所有机器。

③ 接收来自各个服务器的投票。

④ 处理投票。对于投票的处理，和上面提到的服务器启动期间的处理规则是一致的。在这个例子中，由于Server1的ZXID值大于Server3的ZXID值，因此，Server1会成为Leader。

⑤ 统计投票。

⑥ 改变服务器状态。

3）Leader选举算法

在ZooKeeper中，提供了三种Leader选举算法，分别是LeaderElection、UDP版本的FastLeaderElection和TCP版本的FastLeaderElection，可以通过在配置文件zoo.cfg中使用electionAlg属性来指定，分别使用数字0-3来表示，各个数字所代表的Leader选举算法如表4-4所示。

表4-4 Leader选举算法

|  |  |  |
| --- | --- | --- |
| electionAlg属性值 | 对应的Leader选举算法 | 备注 |
| 0 | LeaderElection，这是一种纯UDP实现的Leader选举算法 | 自3.4.0版本起，废弃了0、1、2这三种Leader选举算法 |
| 1 | UDP版本的FastLeaderElection，并且是非授权模式 |
| 2 | UDP版本的FastLeaderElection，但使用授权模式 |
| 3 | TCP版本的FastLeaderElection |  |

如表4-4所示，由于自ZooKeeper 3.4.0版本起，废弃了0、1、2这三种Leader选举算法，例如ZooKeeper 3.4.13采用的Leader选举算法是TCP版本的FastLeaderElection。

（三）部署ZooKeeper

1. 运行环境

对于大部分Java开源产品而言，在部署与运行之前，总是需要搭建一个合适的环境，通常包括操作系统和Java环境两方面。同样，ZooKeeper部署与运行所需要的系统环境，同样包括操作系统和Java环境两部分。

1）操作系统

ZooKeeper支持不同平台，在当前绝大多数主流的操作系统上都能够运行，例如GNU/Linux、Sun Solaris、FreeBSD、Windows、Mac OS X等。需要注意的是，ZooKeeper官方文档中特别强调，不建议在Mac OS X系统上部署生成环境的ZooKeeper服务器。本书采用的操作系统为Linux发行版CentOS 7。

2）Java环境

ZooKeeper使用Java语言编写，因此它的运行环境需要Java环境的支持，对于ZooKeeper 3.4.13，需要Java 1.6及以上版本的支持。

2. 运行模式

ZooKeeper有两种运行模式：单机模式和集群模式。单机模式是只在一台机器上安装ZooKeeper，主要用于开发测试，而集群模式则是在多台机器上安装ZooKeeper，实际的生产环境中均采用集群模式。无论哪种部署方式，创建ZooKeeper的配置文件zoo.cfg都是至关重要的。单机模式和集群模式部署的步骤基本一致，只是在zoo.cfg文件的配置上有些差异。

读者需要注意的是，假设你拥有一台比较好的机器（CPU核数大于10，内存大于等于8GB），如果作为单机模式进行部署，资源明显有点浪费，如果按照集群模式进行部署，需要借助硬件上的虚拟化技术，把一台物理机器转换成几台虚拟机，这样操作成本太高。幸运的是，和其他分布式系统如Hadoop一样，ZooKeeper也允许在一台机器上完成一个伪集群的搭建。所谓伪集群，是指集群所有的机器都在一台机器上，但是还是以集群的特性来对外提供服务。这种模式和集群模式非常类似，只是把zoo.cfg文件中配置项“server.id=host:port:port”略做修改。

3. 配置文件

ZooKeeper启动时，默认读取$ZOOKEEPER\_HOME/conf/zoo.cfg文件，zoo.cfg文件需要配置ZooKeeper的运行参数。ZooKeeper部分配置参数及其含义如表4-5所示，注意，这里仅列举了部分参数，关于完整的配置参数介绍请参见官方文档https://zookeeper.apache.org/doc/r3.4.13/zookeeperAdmin.html#sc\_configuration。

表4-5 zoo.cfg部分配置参数

|  |  |  |
| --- | --- | --- |
| 参数名 | | 说明 |
| 基本配置 | clientPort | 用于配置当前服务器对外的服务端口，客户端会通过该端口和ZooKeeper服务器创建连接，一般设置为2181 |
| dataDir | 用于配置ZooKeeper服务器存储ZooKeeper数据快照文件的目录，同时用于存放集群的myid文件 |
| tickTime | 用于配置ZooKeeper中最小时间单元（单位：毫秒）。ZooKeeper所有时间均以这个时间单元的整数倍配置，例如，Session的最小超时时间是2\*tickTime |
| 高级配置 | dataLogDir | 用于配置ZooKeeper服务器存储ZooKeeper事务日志文件的目录。默认情况下，ZooKeeper会将事务日志文件和数据快照文件存储在同一个目录即dataDir中。应尽量给事务日志的输出配置一个单独的磁盘或者挂载点，这将允许使用一个专用日志设备，帮助避免事务日志和数据快照之间的竞争 |
| maxClientCnxns | 用于配置单个客户端与单台服务器之间的最大并发连接数，根据IP来区分。默认值为60，如果设置为0，表示没有任何限制 |
| minSessionTimeout | 用于配置服务端对客户端会话超时时间的最小值，默认值为2\*tickTime |
| maxSessionTimeout | 用于配置服务端对客户端会话超时时间的最大值，默认值为20\*tickTime |
| 集群选项 | initLimit | 用于配置Leader服务器等待Follower启动，并完成数据同步的时间，以tickTime的倍数来表示，当超过设置倍数的tickTime时间，则连接失败 |
| syncLimit | 用于配置Leader服务器和Follower之间进行心跳检测的最大延迟时间，如果超过此时间Leader还没有收到响应，那么Leader就会认为该Follower已经脱离了和自己的同步 |
| server.id=host:port:port | 用于配置组成ZooKeeper集群的机器列表。集群中每台机器都需要感知到整个集群是由哪几台机器组成的，表示不同ZooKeeper服务器的自身标识。“id”被称为Server ID，用来标识该机器在集群中的机器序号，与每台服务器myid文件中的数字相对应；“host”代表服务器的IP地址；第一个端口“host”用于指定Follower服务器与Leader进行运行时通信和数据同步时所使用的端口；第二个端口“port”代表进行Leader选举时服务器相互通信的端口。myid文件应创建于服务器的dataDir目录下，这个文件的内容只有一行且是一个数字，对应于每台机器的Server ID数字，比如，服务器“1”应该在myid文件中写入“1”，该id必须在集群环境下服务器标识中是唯一的，且大小在1～255之间 |

例如，单机模式的zoo.cfg文件示例内容如下。

tickTime=2000

dataDir=/var/lib/zookeeper

clientPort=2181

集群模式的zoo.cfg文件示例内容如下。

tickTime=2000

dataDir=/var/lib/zookeeper/

clientPort=2181

initLimit=5

syncLimit=2

server.1=zoo1:2888:3888

server.2=zoo2:2888:3888

server.3=zoo3:2888:3888

伪集群模式的zoo.cfg文件示例内容如下。

dataDir=/var/lib/zookeeper/

clientPort=2181

initLimit=5

syncLimit=2

server.1=zoo1:2888:3888

server.2=zoo2:2889:3889

server.3=zoo3:2890:3890

（四）ZooKeeper四字命令

ZooKeeper中有一系列的命令可以查看服务器的运行状态，它们的长度通常都是4个英文字母，因此又被称之为“四字命令”。ZooKeeper四字命令及功能如表4-6所示。

表4-6 ZooKeeper四字命令

|  |  |
| --- | --- |
| 命令 | 功能描述 |
| conf | 用于输出ZooKeeper服务器运行时使用的基本配置信息，包括clientPort、dataDir和tickTime等 |
| cons | 用于输出当前这台服务器上所有客户端连接的详细信息，包括每个客户端的客户端IP、会话ID和最后一次与服务器交互的操作类型等 |
| crst | 功能性命令，用于重置所有的客户端连接统计信息 |
| dump | 用于输出当前集群的所有会话信息，包括这些会话的会话ID，以及每个会话创建的临时节点等信息 |
| envi | 用于输出ZooKeeper所在服务器运行时的环境信息，包括os.version、java.version和user.home等 |
| mntr | 用于输出比stat命令更为详尽的服务器统计信息，包括请求处理的延迟情况、服务器内存数据库大小和集群的数据同步情况 |
| ruok | 用于输出当前ZooKeeper服务器是否正在运行。该命令的名字非常有趣，其谐音正好是“Are you ok”。执行该命令后，如果当前ZooKeeper服务器正在运行，那么返“imok”，否则没有任何响应输出 |
| stat | 用于获取ZooKeeper服务器的运行时状态信息，包括基本的ZooKeeper版本、打包信息、运行时角色、集群数据节点个数等信息 |
| srvr | 和stat命令的功能一致，唯一的区别是srvr不会将客户端的连接情况输出，仅仅输出服务器的自身信息 |
| srst | 功能性命令，用于重置所有服务器的统计信息 |
| wchc | 用于输出当前服务器上管理的Watcher的详细信息，以会话为单位进行归组，同时列出被该会话注册了Watcher的节点路径 |
| wchp | 和wchc命令非常类似，也是用于输出当前服务器上管理的Watcher的详细信息，不同点在于wchp命令的输出信息以节点路径为单位进行归组 |
| wchs | 用于输出当前服务器上管理的Watcher的概要信息 |

需要注意的是，ruok命令的输出仅仅只能表明当前服务器是否正在运行，准确地讲，只能说明2181端口打开着，同时四字命令执行流程正常，但是不能代表ZooKeeper服务器是否运行正常。在很多时候，如果当前服务器无法正常处理客户端的读写请求，甚至已经无法和集群中的其他机器进行通信，ruok命令依然返回“imok”。

ZooKeeper四字命令的使用很简单，通常有两种方式。第一种是通过Telnet方式，使用Telnet客户端登录ZooKeeper对外服务端口，然后直接输入四字命令即可，此方式需要在机器上安装Telnet。例如，telnet方式使用ZooKeeper四字命令“conf”命令效果如图4-5所示。

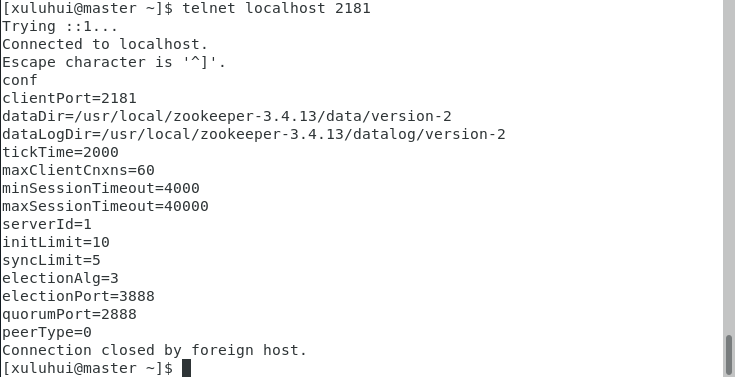


图4-5 telnet方式使用ZooKeeper四字命令

第二种则是使用nc方式，命令语法如下所示：

echo {command} | nc {host} 2181

nc方式使用ZooKeeper四字命令“conf”命令效果如图4-6所示。

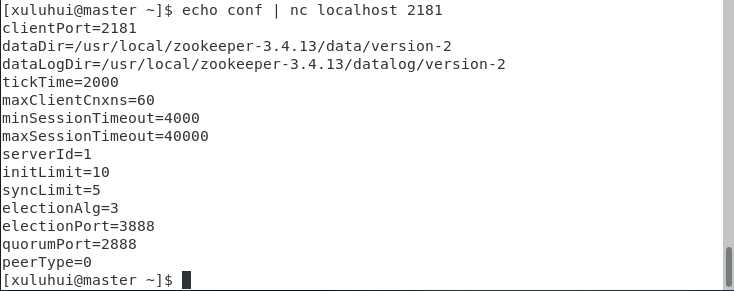


图4-6 nc方式使用ZooKeeper四字命令

（五）ZooKeeper Shell

1. 服务器命令行工具zkServer.sh

zkServer.sh用于启动、查看、关闭ZooKeeper集群等，可以使用“zkServer.sh -help”查看其帮助，其具体用法如图4-7所示。

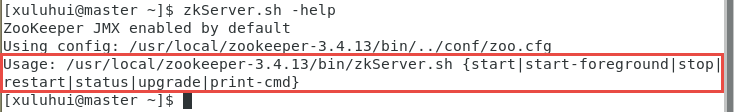


图4-7 ZooKeeper Shell服务器命令用法

常用选项功能如下：

（1）start：启动ZooKeeper服务。

（2）stop：停止ZooKeeper服务。

（3）restart：重启ZooKeeper服务。

（4）status：查看ZooKeeper状态。

2. 客户端命令行工具zkCli.sh

zkCli.sh用于对ZooKeeper文件系统中数据节点进行新建、查看或删除等操作，进入客户端命令行的方法有如下几种：

（1）连接本地ZooKeeper服务器

使用命令“zkCli.sh”即可连接到本地ZooKeeper服务器，命令中没有显式指定ZooKeeper服务器地址，那么默认是本地ZooKeeper服务器。使用效果如下所示。

[xuluhui@master ~]$ zkCli.sh

[zk: localhost:2181(CONNECTED) 0]

（2）连接指定ZooKeeper服务器

若希望连接到指定的ZooKeeper服务器，可以通过如下命令实现：

zkCli.sh -server host:port

其中参数“ip”表示提供ZooKeeper服务的节点IP或主机名，参数“port”是4.4.3节介绍的客户端连接当前ZooKeeper服务器的端口号，一般设置为2181。例如，连接到slave1节点的ZooKeeper服务器通过如下命令实现：

[xuluhui@master ~]$ zkCli.sh -server slave1:2181

[zk: slave1:2181(CONNECTED) 0]

再如，如下命令并不是连接了两个节点，而是按照顺序连接一个，当第一个连接无法获取时，就连接第二个。

[xuluhui@master ~]$ zkCli.sh -server slave1:2181,slave2:2181

[zk: slave1:2181,slave2:2181(CONNECTED) 0]

读者可以通过客户端命令行工具zkCli.sh命令“help”来查看可以进行的所有操作，如图4-8所示。

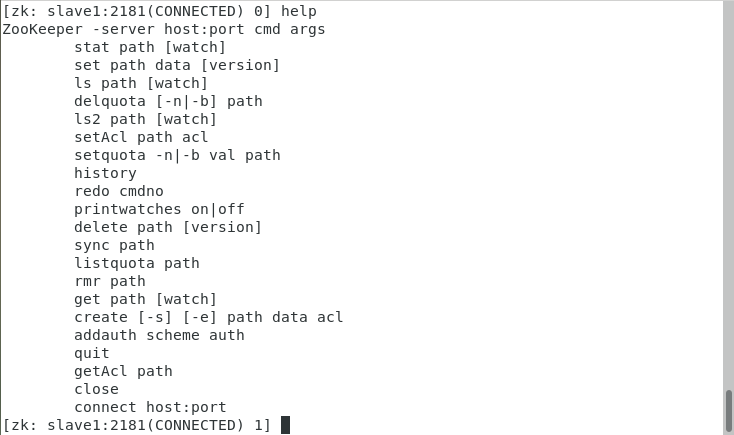


图4-8 ZooKeeper Shell客户端命令用法

图4-7所展示的客户端命令中，create用于新建节点，set用于设置节点数据，get用于获取节点数据，delete只能删除一个节点，rmr可以级联删除，close用于关闭当前session，quit用于退出客户端命令行。关于几个常用命令的使用方法如表4-7所示，由于命令众多，此处不再一一讲解，读者可以自行查阅资料并进行实践。

表4-7 ZooKeeper Shell客户端部分命令使用说明

|  |  |  |
| --- | --- | --- |
| 命令 | 语法 | 功能 |
| ls | ls path [watch] | 列出ZooKeeper指定节点下的所有子节点。这个命令仅能看到指定节点下第一级的所有子节点，其中，参数path用于指定数据节点的节点路径 |
| create | create [-s] [-e] path data acl | 创建ZooKeeper数据节点。其中，参数-s和-e用于指定节点特性，-s为顺序节点，-e为临时节点。默认情况下，即不添加-s或-e参数的，创建的是持久节点；参数path指定节点路径；参数data指定节点数据内容；参数acl用来进行权限控制，默认情况下，不做任何权限控制 |
| get | get path [watch] | 获取ZooKeeper指定节点的数据内容和属性信息 |
| set | set path data [version] | 更新ZooKeeper指定节点的数据内容。其中，参数data就是要更新的新内容；参数version用于指定本次更新操作时基于数据节点的哪一个数据版本进行的 |
| delete | delete path [version] | 删除ZooKeeper上指定数据节点。其中，参数version  的作用与set命令中version参数一致 |

（六）ZooKeeper Java API

ZooKeeper作为一个分布式服务框架，主要用来解决分布式数据一致性问题，提供了简单的分布式原语，并且对多种编程语言提供了API。下面我们重点介绍ZooKeeper的Java客户端API使用方式。

ZooKeeper Java API面向开发工程师，包含org.apache.zookeeper、org.apache.zookeeper.data、org.apache.zookeeper.server、org.apache.zookeeper.server.quorum、org.apache.zookeeper.server.upgrade等包，其中org.apache.zookeeper包含ZooKeeper类，它是编程时最常用的类文件。本小节仅讲述几个常用的ZooKeeper Java API，完整的ZooKeeper Java API请参考官方参考指南http://zookeeper.apache.org/doc/r3.4.13/api/index.html。其首页如图4-9所示。

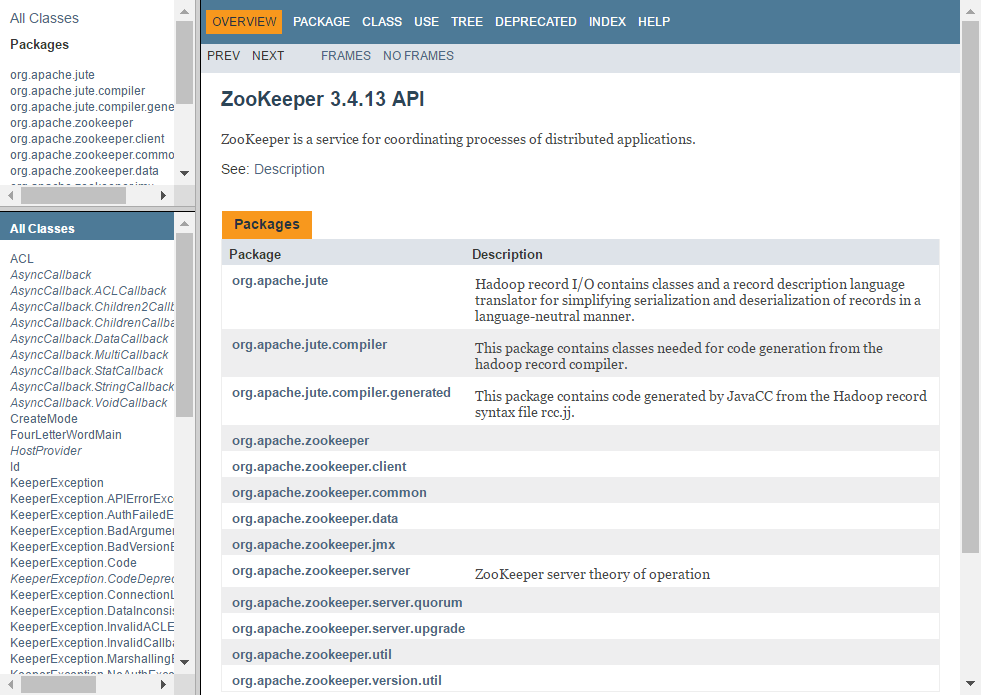


图4-9 ZooKeeper 3.4.13 API官方参考指南首页

ZooKeeper API中关于ZNode有9个基本操作，如表4-8所示。

表4-8 ZNode的9个基本操作

|  |  |
| --- | --- |
| 操作 | 描述 |
| create | 创建ZNode（父ZNode必须存在） |
| delete | 删除ZNode（无子节点） |
| exists | 测试ZNode是否存在，并获取它的元数据 |
| getACL | 为ZNode获取ACL |
| setACL | 为ZNode设置ACL |
| getChildren | 获取ZNode所有子节点的列表 |
| getData | 获取ZNode相关数据 |
| setData | 设置ZNode相关数据 |
| sync | 使客户端的ZNode视图与ZooKeeper服务器你同步 |

更新ZooKeeper操作是有限制的。delete或setData必须明确待更新的ZNode版本号，可以调用exists找到，如果版本号不匹配，更新将会失败。

更新ZooKeeper操作是非阻塞式的。因此客户端如果失去了一个更新（由于另一个进程在同时更新这个ZNode），它可以在不阻塞其他进程执行的情况下，选择重新尝试或进行其他操作。

五、实验步骤

（一）规划ZooKeeper集群

1. ZooKeeper集群规划

部署ZooKeeper集群时，最低需要3台机器。本实验拟将ZooKeeper集群运行在Linux上，将使用3台安装有Linux操作系统的机器，主机名分别为master、slave1、slave2。具体ZooKeeper集群规划表如表4-9所示。

表4-9 ZooKeeper集群部署规划表

|  |  |  |  |
| --- | --- | --- | --- |
| 主机名 | IP地址 | 运行服务 | 软硬件配置 |
| master | 192.168.18.130 | QuorumPeerMain | 内存：4G  CPU：1个2核  硬盘：20G  操作系统：CentOS 7.6.1810  Java：Oracle JDK 8u191  ZooKeeper：ZooKeeper 3.4.13  Eclipse：Eclipse IDE 2018-09 for Java Developers |
| slave1 | 192.168.18.131 | QuorumPeerMain | 内存：1G  CPU：1个1核  硬盘：20G  操作系统：CentOS 7.6.1810  Java：Oracle JDK 8u191  ZooKeeper：ZooKeeper 3.4.13 |
| slave2 | 192.168.18.132 | QuorumPeerMain | 内存：1G  CPU：1个1核  硬盘：20G  操作系统：CentOS 7.6.1810  Java：Oracle JDK 8u191  ZooKeeper：ZooKeeper 3.4.13 |

2. 软件选择

本实验中所使用各种软件的名称、版本、发布日期及下载地址如表4-10所示。

表4-10 本实验使用的软件名称、版本、发布日期及下载地址

|  |  |  |  |
| --- | --- | --- | --- |
| 软件名称 | 软件版本 | 发布日期 | 下载地址 |
| VMware Workstation Pro | VMware Workstation 14.5.7 Pro for Windows | 2017年6月22日 | https://www.vmware.com/products/workstation-pro.html |
| CentOS | CentOS 7.6.1810 | 2018年11月26日 | https://www.centos.org/download/ |
| Java | Oracle JDK 8u191 | 2018年10月16日 | http://www.oracle.com/technetwork/java/javase/downloads/index.html |
| ZooKeeper | ZooKeeper 3.4.13 | 2018年7月15日 | http://zookeeper.apache.org/releases.html |
| Eclipse | Eclipse IDE 2018-09 for Java Developers | 2018年9月 | https://www.eclipse.org/downloads/packages/ |

注意，本书采用的是ZooKeeper版本是3.4.13，3个节点的机器名分别为master、slave1、slave2，IP地址依次为192.168.18.130、192.168.18.131、192.168.18.132，后续内容均在表4-9规划基础上完成，请读者务必确认自己的ZooKeeper版本、机器名等信息。

由于“部署全分布模式Hadoop集群”实验中已完成VMware Workstation Pro、CentOS、Java的安装，故本实验直接从部署ZooKeeper集群开始讲述。

（二）部署ZooKeeper集群

本书采用的ZooKeeper版本是3.4.13，因此本章的讲解都是针对这个版本进行的。尽管如此，由于ZooKeeper各个版本在部署和运行方式上的变化不大，因此本章的大部分内容都适用于ZooKeeper其他版本。

1. 初始软硬件环境准备

（1）准备3台机器，安装操作系统，编者使用CentOS Linux 7。

（2）对集群内每一台机器，配置静态IP、修改机器名、添加集群级别域名映射、关闭防火墙。

（3）对集群内每一台机器，安装和配置Java，要求Java 1.6或更高版本，编者使用Oracle JDK 8u191。

（4）可以选择性地安装和配置Linux集群中各节点间的SSH免密登录。关于安装和配置SSH免密登录并不是部署ZooKeeper集群必须的，这样做仅是为了操作方便。

以上步骤已在实验项目1中详细介绍，具体操作过程请读者参见教材，此处不再赘述。

2. 获取ZooKeeper

ZooKeeper官方下载地址为https://zookeeper.apache.org/releases.html，建议读者下载stable目录下的当前稳定版本，本实验选用的ZooKeeper版本是2018年7月15日发布的稳定版ZooKeeper 3.4.13，其安装包文件zookeeper-3.4.13.tar.gz例如存放在master机器的/home/xuluhui/Downloads中。

3. 安装ZooKeeper

切换到root，在master机器上解压zookeeper-3.4.13.tar.gz到安装目录如/usr/local下，依次使用的命令如下所示，使用命令及运行效果如图4-10所示。

su root

cd /usr/local

tar -zxvf /home/xuluhui/Downloads/zookeeper-3.4.13.tar.gz

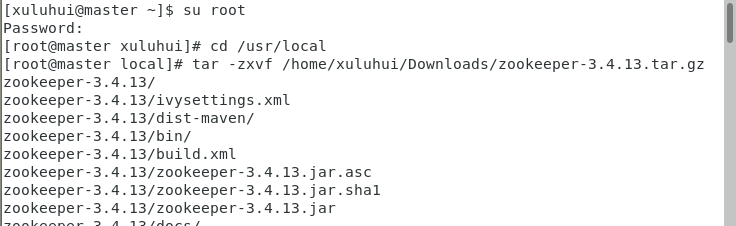


图4-10 安装zookeeper-3.4.13.tar.gz

4. 配置ZooKeeper

1）复制模板配置文件zoo\_sample.cfg为zoo.cfg

在master机器上使用命令“cp”将ZooKeeper示例配置文件zoo\_sample.cfg复制并重命名为zoo.cfg。使用如下命令实现，假设当前目录为“/usr/local/zookeeper-3.4.13”。使用命令及运行效果如图4-11所示。

cp conf/zoo\_sample.cfg conf/zoo.cfg

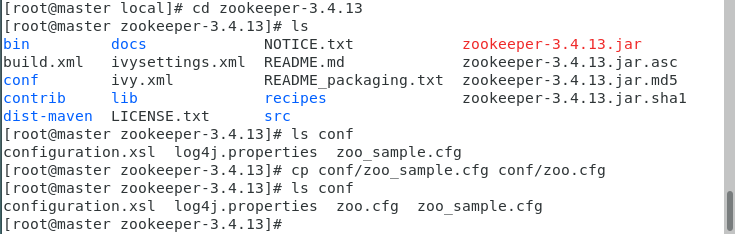


图4-11 复制并更名配置文件

2）修改配置文件zoo.cfg

读者可以发现，模板中已配置好tickTime、initLimit、syncLimit、dataDir、clientPort等配置项，此处，编者仅在master机器上修改配置参数dataDir和添加配置参数dataLogDir。由于机器重启后，系统会自动清空/tmp目录下文件，所以将存放数据快照的目录更改为某固定目录，将原始的“dataDir=/tmp/zookeeper”修改为“/usr/local/zookeeper-3.4.13/data”；另外，添加事务日志存放路径dataLogDir，设置为“/usr/local/zookeeper-3.4.13/datalog”。修改后的配置文件zoo.cfg内容如图4-12所示。

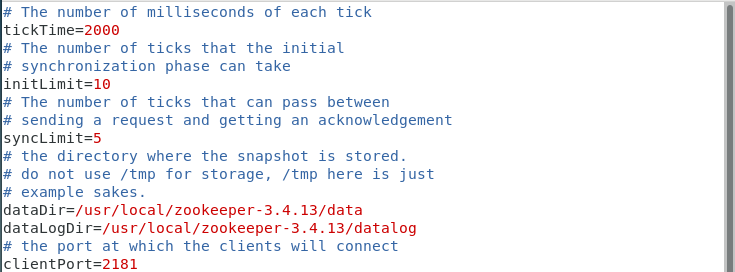


图4-12 修改配置文件zoo.cfg

其次，在master机器上配置ZooKeeper集群地址，在配置文件zoo.cfg最后补充几行内容，如下所示。

server.1=master:2888:3888

server.2=slave1:2888:3888

server.3=slave2:2888:3888

5. 创建所需目录和新建myid文件

在上步修改配置文件zoo.cfg中，将存放数据快照和事务日志的目录设置为目录data和datalog，因此需要在master机器上创建这两个目录，使用如下命令实现，假设当前目录为以上步骤操作后的所在目录“/usr/local/zookeeper-3.4.13”。

mkdir data

mkdir datalog

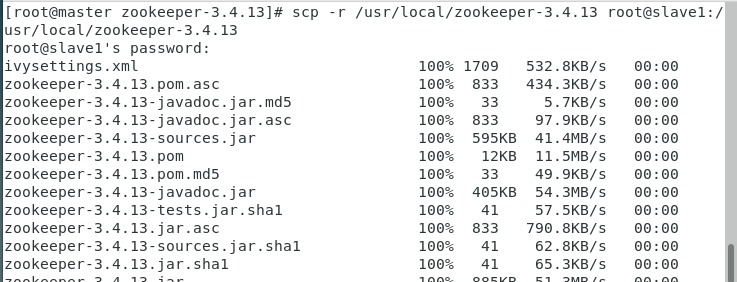
然后，在数据快照目录下新建文件myid并填写ID。在master机器配置项dataDir指定目录下创建文件“myid”，例如在dataDir目录“/usr/local/zookeeper3.4.13/data”下使用命令“vim”新建文件myid，并将其内容设置为“1”。之所以为“1”，是由于配置文件zoo.cfg中“server.id=host:port:port”配置项master机器对应的“id”为“1”。

6. 同步ZooKeeper文件至slave1、slave2

使用scp命令将master机器中目录“zookeeper-3.4.13”及下属子目录和文件统一拷贝至slave1和slave2上，依次使用的命令如下所示，使用命令及运行效果如图4-13所示。

scp -r /usr/local/zookeeper-3.4.13 root@slave1:/usr/local/zookeeper-3.4.13

scp -r /usr/local/zookeeper-3.4.13 root@slave2:/usr/local/zookeeper-3.4.13



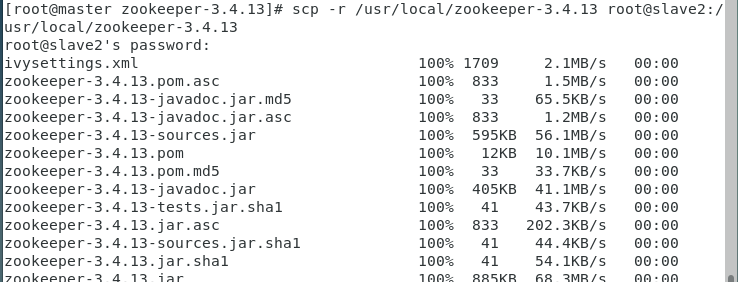


图4-13 同步ZooKeeper目录到集群其它机器

7. 设置$ZOOKEEPER\_HOME目录属主

为了在普通用户下使用ZooKeeper集群，依次将3台机器master、slave1、slave2上的$ZOOKEEPER\_HOME目录属主设置为Linux普通用户例如xuluhui，使用以下命令完成。

chown -R xuluhui /usr/local/zookeeper-3.4.13

8. 修改slave1、slave2文件myid内容

配置文件conf/zoo.cfg中“server.id=host:port:port”配置项中id与哪台主机对应，myid文件中的内容就是什么数字。本例中，3台机器按master、slave1、slave2对应的“id”依次为“1、2、3”，因此将slave1机器上文件myid的内容修改为“2”，将slave2机器上文件myid的内容修改为“3”。

至此，Linux集群中3台机器的ZooKeeper均已安装和配置完毕。

9. 在系统配置文件目录/etc/profile.d下新建zookeeper.sh

在ZooKeeper集群的所有机器上执行以下操作。

首先，切换到root用户，使用“vim /etc/profile.d/zookeeper.sh”命令在/etc/profile.d文件夹下新建文件zookeeper.sh，添加如下内容。

export ZOOKEEPER\_HOME=/usr/local/zookeeper-3.4.13

export PATH=$ZOOKEEPER\_HOME/bin:$PATH

其次，重启机器，使之生效。

此步骤可省略，之所以将$ZOOKEEPER\_HOME/bin目录加入到系统环境变量PATH中，是因为当输入启动和管理ZooKeeper集群命令时，无需再切换到$ZOOKEEPER\_HOME/bin目录，否则会出现错误信息“bash: \*\*\*\*: command not found...”。

例如master机器，新建配置文件zookeeper.sh前，直接启动ZooKeeper集群时会出现错误信息“bash: zkServer.sh: command not found...”。而新建配置文件zookeeper.sh且重启机器后，可以直接启动ZooKeeper集群而无需进入到$ZOOKEEPER\_HOME/bin目录下启动ZooKeeper，$ZOOKEEPER\_HOME/bin加入系统环境变量PATH前后服务端命令运行效果如图4-14所示，这样使用起来会更加方便。

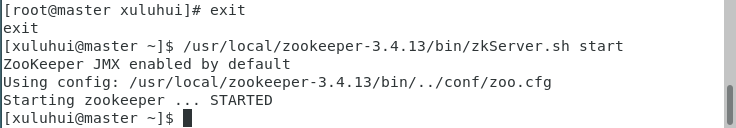


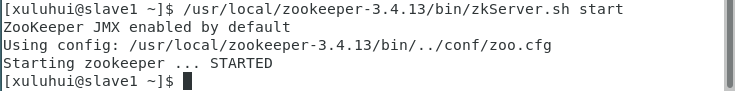


图4-14 $ZOOKEEPER\_HOME/bin加入系统环境变量PATH前后服务端命令使用区别

（三）启动ZooKeeper集群

在ZooKeeper集群的每个节点上，在普通用户xuluhui下使用命令“zkServer.sh start”来启动ZooKeeper，使用的命令及运行效果如图4-15所示，从图中可以看出，3个节点均显示“Starting zookeeper … STARTED”信息。





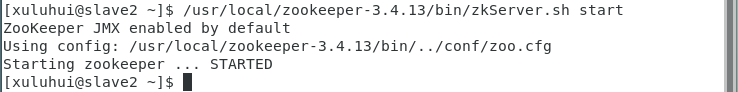
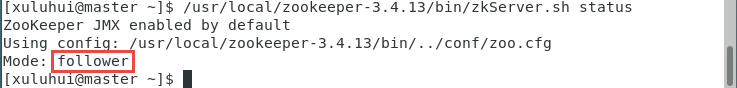


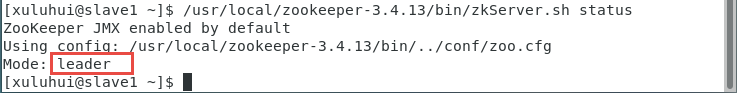
图4-15 启动ZooKeeper集群

（四）验证ZooKeeper集群

启动后可查看zookeeper.out的日志。由于ZooKeeper集群启动的时候，每个节点都试图去连接集群中的其它节点，故存在启动时后边的节点还没启动，所以会出现异常的日志，这是正常的。启动选出一个Leader后就稳定了。

查看ZooKeeper是否部署成功的第1种方法是：在各个节点上通过“zkServer.sh status”命令查看状态，包括集群中各个节点的角色，使用命令及运行效果如图4-16所示，从图4-16中可以看出，slave1是Leader。





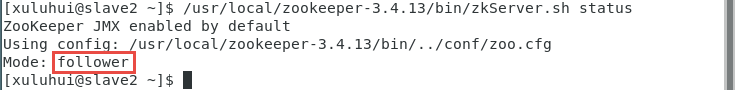


图4-16 通过zkServer.sh status查看ZooKeeper集群启动状态

查看ZooKeeper是否部署成功的第2种方法是：在各个节点上通过“jps”命令查看进程服务，若部署成功的话，可在各个节点上看到QuorumPeerMain进程，运行效果如图4-17所示。







图4-17 通过jps查看进程服务

（五）使用ZooKeeper四字命令

1. conf

conf命令用于输出ZooKeeper服务器运行时使用的基本配置信息，包括clientPort、dataDir和tickTime等。使用conf命令查看master节点上ZooKeeper服务器基本配置信息的效果如图4-18所示。

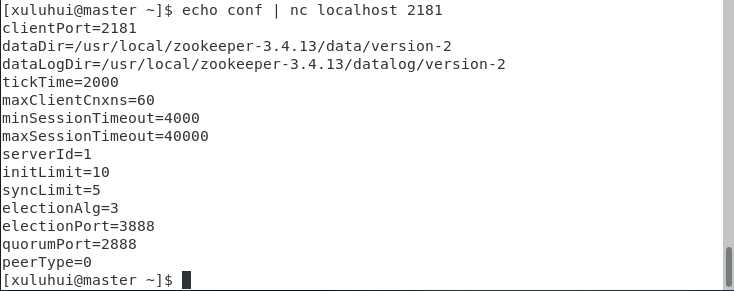


图4-18 查看master节点上ZooKeeper服务器运行时使用的基本配置信息

2. stat

stat命令用于获取ZooKeeper服务器的运行时状态信息，包括基本的ZooKeeper版本、打包信息、运行时角色、集群数据节点个数等信息。使用stat命令查看slave1节点上ZooKeeper服务器的运行时状态信息的效果如图4-19所示。

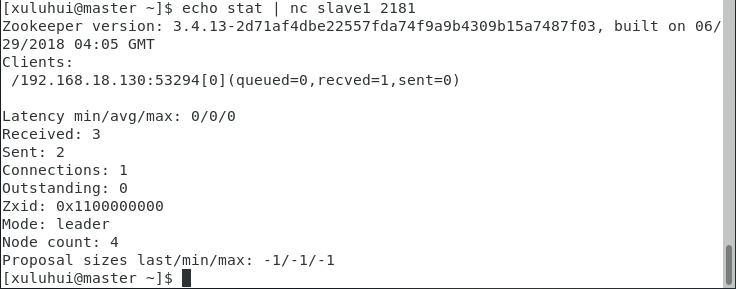


图4-19 查看slave1节点上ZooKeeper服务器运行时状态信息

3.envi

envi命令用于输出ZooKeeper所在服务器运行时的环境信息，包括os.version、java.version和user.home等。使用envi命令查看slave1节点上ZooKeeper服务器运行时环境信息的效果如图4-20所示。

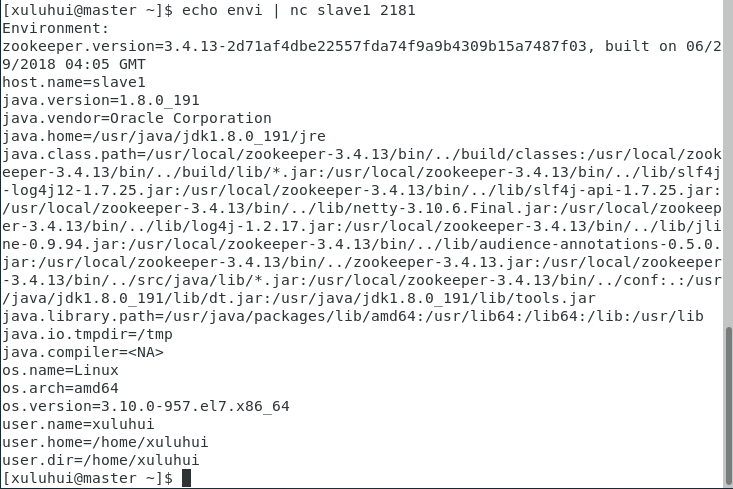


图4-20 查看slave1节点上ZooKeeper服务器运行时环境信息

（六）使用ZooKeeper Shell常用命令

1. 使用ZooKeeper Shell服务器命令

zkServer.sh用于启动、查看、关闭ZooKeeper集群等，例如，ZooKeeper服务器的启动、查看帮助、查看状态所使用的命令及运行效果如图4-21所示。

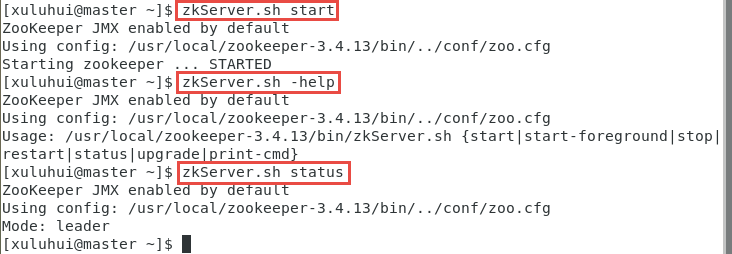


图4-21 ZooKeeper服务器的启动、查看帮助、查看状态

2. 使用ZooKeeper Shell客户端命令

【案例4-1】使用zkCli.sh实现对ZooKeeper文件系统中数据节点进行新建、查看或删除等操作。

（1）连接slave1节点ZooKeeper服务器，进入ZooKeeper命令行，使用的命令及运行效果如图4-22所示。

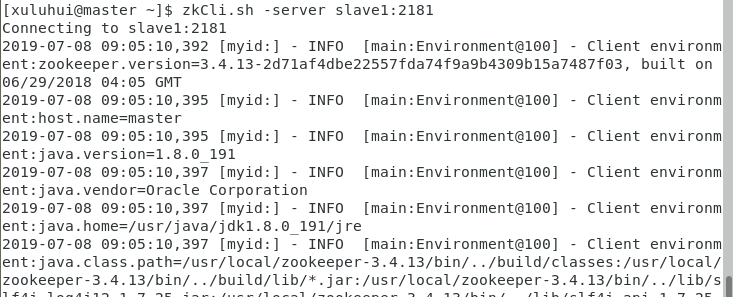


图4-22 进入ZooKeeper客户端命令行

连接成功之后，系统会输出该ZooKeeper服务器的相关环境及配置信息，并在屏幕输出“Welcome to ZooKeeper！”等信息，如图4-23所示。

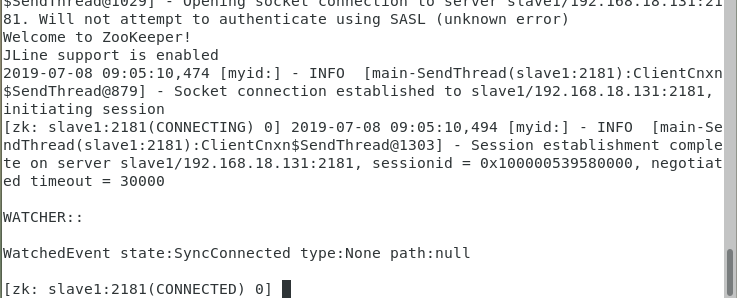


图4-23 ZooKeeper的相关环境及配置信息

（2）使用“help”命令查看所有支持操作，运行效果如图4-24所示。

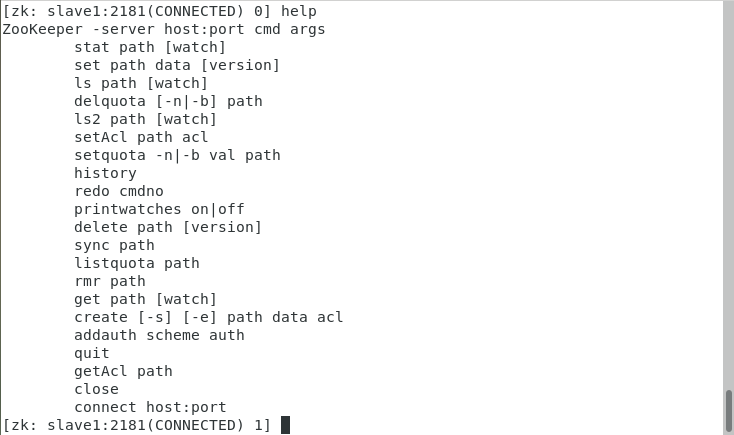


图4-24 查看ZooKeeper客户端命令行帮助

（3）使用ZooKeeper Shell命令进行下列一系列简单操作。

① 使用命令ls查看根节点下的所有子节点，使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 0] ls /

[zookeeper]

[zk: slave1:2181(CONNECTED) 1]

使用命令create在根目录“/”下创建ZNode“xijing”及相关数据内容，默认创建持久节点，使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 1] create /xijing "it's a persistent node"

Created /xijing

[zk: slave1:2181(CONNECTED) 2]

使用命令get查看ZNode“/xijing”数据内容及节点信息，使用命令及运行效果如下所示，这里，各个属性信息后均人工添加了注释。

[zk: slave1:2181(CONNECTED) 2] get /xijing

it's a persistent node # 数据节点的数据内容

cZxid = 0x100000002 # 数据节点创建时的事务ID

ctime = Wed Jul 17 02:25:54 EDT 2019 # 数据节点创建时的时间

mZxid = 0x100000002 # 数据节点最后一次更新时的事务ID

mtime = Wed Jul 17 02:25:54 EDT 2019 # 数据节点最后一次更新时的时间

pZxid = 0x100000002 # 数据节点的子节点列表最后一次被修改（子节点列表的变更而非子节点内容的变更）时的事务ID

cversion = 0 # 子节点的版本号

dataVersion = 0 # 数据节点的版本号

aclVersion = 0 # 数据节点的ACL版本号

ephemeralOwner = 0x0 # 如果节点是临时节点，则表示创建该节点的会话的SessionID；如果节点是持久节点，则该属性值为0

dataLength = 22 # 数据内容的长度

numChildren = 0 # 数据节点当前的子节点个数

[zk: slave1:2181(CONNECTED) 3]

从上面的输出信息中我们可以看到，第一行是节点/xijing的数据内容个，其他几行则是创建该节点的事务ID（cZxid）、最后一次更新该节点的事务ID（mZxid）、最后查一次更新该节点的时间（mtime）、数据节点的版本号（dataVersion）等属性信息。

使用命令set对ZNode“/xijing”数据内容进行更新，使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 3] set /xijing "xijing is a persistent node"

cZxid = 0x100000002

ctime = Wed Jul 17 02:25:54 EDT 2019

mZxid = 0x100000003

mtime = Wed Jul 17 02:29:23 EDT 2019

pZxid = 0x100000002

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 27

numChildren = 0

[zk: slave1:2181(CONNECTED) 4]

执行完以上命令后，节点“/xijing”的数据内容被更新为“xijing is a persistent node”。从上面的输出信息中我们可以看到，数据节点最后一次更新时的事务ID（mZxid）发生了变化，最后查一次更新该节点的时间（mtime）发生了变化，数据节点的版本号（dataVersion）由原来的“0”变化为当前的“1”，数据内容的长度（dataLength）由原来的“22”变化为当前的“27”。

使用命令create在根目录“/”下依次创建持久顺序节点“xijing”、临时节点“xijingTmp”、临时顺序节点“xijingTmp”及各自相关数据内容，并查看这三个节点信息。依次使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 4] create -s /xijing "it's a persistent sequential node"

Created /xijing0000000001

[zk: slave1:2181(CONNECTED) 5] get /xijing0000000001

it's a persistent sequential node

cZxid = 0x100000004

ctime = Wed Jul 17 02:34:14 EDT 2019

mZxid = 0x100000004

mtime = Wed Jul 17 02:34:14 EDT 2019

pZxid = 0x100000004

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 33

numChildren = 0

[zk: slave1:2181(CONNECTED) 6]

执行完以上命令后，就在根目录“/”下依次创建了持久顺序节点“xijing”，从上面输出的信息可以看到，ZooKeeper自动在持久顺序节点“xijing”名字后添加了数字后缀“0000000001”，该持久顺序节点的完整名字为“/xijing0000000001”。

[zk: slave1:2181(CONNECTED) 6] create -e /xijingTmp "it's a ephemeral node"

Created /xijingTmp

[zk: slave1:2181(CONNECTED) 7] get /xijingTmp

it's a ephemeral node

cZxid = 0x100000005

ctime = Wed Jul 17 02:34:59 EDT 2019

mZxid = 0x100000005

mtime = Wed Jul 17 02:34:59 EDT 2019

pZxid = 0x100000005

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x200075948aa0000

dataLength = 21

numChildren = 0

[zk: slave1:2181(CONNECTED) 8]

执行完以上命令后，就在根目录“/”下依次创建了临时节点“xijingTmp”，从上面输出的信息可以看到，ephemeralOwner不再是之前持久节点的值“0x0”，因为该节点是临时节点，此状态信息表示创建该节点的会话的SessionID。

[zk: slave1:2181(CONNECTED) 8] create -e -s /xijingTmp "it's a ephemeral sequential node"

Created /xijingTmp0000000003

[zk: slave1:2181(CONNECTED) 9] get /xijingTmp0000000003

it's a ephemeral sequential node

cZxid = 0x100000006

ctime = Wed Jul 17 02:35:27 EDT 2019

mZxid = 0x100000006

mtime = Wed Jul 17 02:35:27 EDT 2019

pZxid = 0x100000006

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x200075948aa0000

dataLength = 32

numChildren = 0

[zk: slave1:2181(CONNECTED) 10]

执行完以上命令后，就在根目录“/”下依次创建了临时顺序节点“xijingTmp”，从上面输出的信息可以看到，ZooKeeper自动在临时顺序节点“xijingTmp”名字后添加了数字后缀“0000000003”，该临时顺序节点的完整名字为“/xijingTmp0000000003”。

使用命令ls再次查看当前根目录下包含的数据节点。使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 10] ls /

[xijing0000000001, xijingTmp, xijingTmp0000000003, zookeeper, xijing]

[zk: slave1:2181(CONNECTED) 11]

使用命令close关闭本次连接回话Session，再使用命令connect host:port重新打开一个连接。使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 11] close

2019-07-17 02:50:32,598 [myid:] - INFO [main:ZooKeeper@693] - Session: 0x200075948aa0000 closed

2019-07-17 02:50:32,601 [myid:] - INFO [main-EventThread:ClientCnxn$EventThread@522] - EventThread shut down for session: 0x200075948aa0000

[zk: slave1:2181(CLOSED) 12] connect slave1:2181

2019-07-17 02:51:16,873 [myid:] - INFO [main:ZooKeeper@442] - Initiating client connection, connectString=slave1:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@6996db8

2019-07-17 02:51:16,876 [myid:] - INFO [main-SendThread(slave1:2181):ClientCnxn$SendThread@1029] - Opening socket connection to server slave1/192.168.18.131:2181. Will not attempt to authenticate using SASL (unknown error)

[zk: slave1:2181(CONNECTING) 13] 2019-07-17 02:51:16,877 [myid:] - INFO [main-SendThread(slave1:2181):ClientCnxn$SendThread@879] - Socket connection established to slave1/192.168.18.131:2181, initiating session

2019-07-17 02:51:16,886 [myid:] - INFO [main-SendThread(slave1:2181):ClientCnxn$SendThread@1303] - Session establishment complete on server slave1/192.168.18.131:2181, sessionid = 0x200075948aa0001, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: slave1:2181(CONNECTED) 13]

再次使用命令ls查看当前根目录下包含的数据节点。使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 13] ls /

[xijing0000000001, zookeeper, xijing]

[zk: slave1:2181(CONNECTED) 14]

从上面输出的信息可以看到，临时节点“xijingTmp”和“/xijingTmp0000000003”均已随着上次会话的关闭而自动删除了。

删除持久节点“/xijing”和持久顺序节点“/xijing0000000001”，并查看根目录下包含的数据节点。依次使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 14] delete /xijing

[zk: slave1:2181(CONNECTED) 15] delete /xijing0000000001

[zk: slave1:2181(CONNECTED) 16] ls /

[zookeeper]

[zk: slave1:2181(CONNECTED) 17]

退出与slave1节点的ZooKeeper服务器连接。使用命令及运行效果如下所示。

[zk: slave1:2181(CONNECTED) 17] quit

Quitting...

2019-07-17 02:56:34,848 [myid:] - INFO [main:ZooKeeper@693] - Session: 0x200075948aa0001 closed

2019-07-17 02:56:34,848 [myid:] - INFO [main-EventThread:ClientCnxn$EventThread@522] - EventThread shut down for session: 0x200075948aa0001

[xuluhui@master ~]$

（七）关闭ZooKeeper集群

在ZooKeeper集群的每个节点上，在普通用户xuluhui下使用命令“zkServer.sh stop”来关闭ZooKeeper服务。若Linux集群各机器节点间已配置好SSH免密登录，也可以仅在master一台机器上输入一系列命令以关闭整个ZooKeeper集群，依次使用的命令及运行效果如图4-25所示，从图4-25中可以看出，3个节点均显示“Stoppping zookeeper … STOPPED”信息。

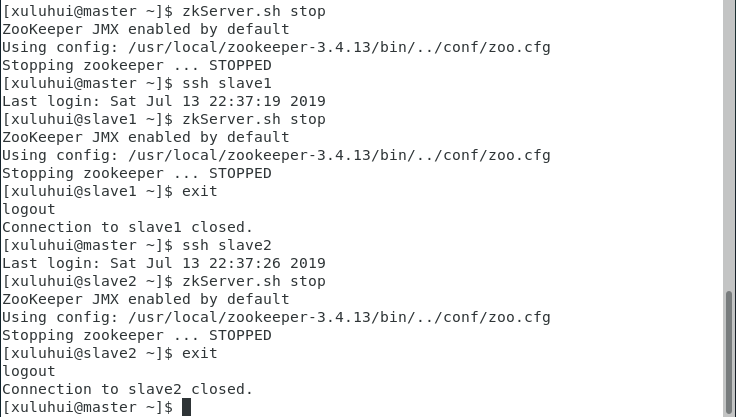


图4-25 在master一台机器上输入一系列命令关闭整个ZooKeeper集群

六、实验报告要求

实验报告以电子版和打印版双重形式提交。

实验报告主要内容包括实验名称、实验类型、实验地点、学时、实验环境、实验原理、实验步骤、实验结果、总结与思考等。实验报告格式如表1-9所示。

七、拓展训练

本节中，编者将通过一些简单实例来介绍如何使用ZooKeeper API编写应用程序，若要深入学习ZooKeeper编程，读者可以访问ZooKeeper官方网站提供的完整ZooKeeper API文档（http://zookeeper.apache.org/doc/r3.4.13/api/index.html）。

为了提高程序编写和调试效率，本书采用Eclipse工具编写Java程序，采用版本为适用于64位Linux操作系统的Eclipse IDE 2018-09 for Java Developers，关于如何在Linux下安装Eclipse，具体过程如下所示。

（一）搭建ZooKeeper开发环境Eclipse

在ZooKeeper集群中某一节点上搭建ZooKeeper开发环境Eclipse，具体过程请读者参考本书2.3.4节，此处不再赘述。

（二）ZooKeeper编程实践：ZooKeeper文件系统的增删改查

【案例4-2】使用ZooKeeper Java API实现对ZooKeeper文件系统的增删改查。本案例的具体实现步骤如下所示。

1. 在Eclipse中创建项目Java Project

进入/usr/local/eclipse中通过可视化桌面打开Eclipse IDE，默认的工作空间为“/home/xlh/eclipse-workspace”。

选择菜单『File』→『New』→『Java Project』，创建Java项目“ZooKeeperExample”，如图4-26所示。本书中关于ZooKeeper编程实例均存放在此项目下。

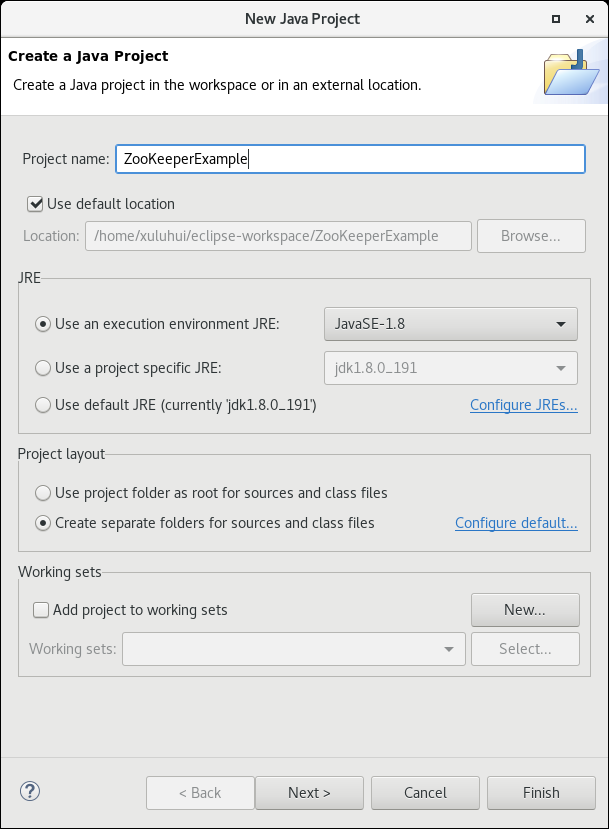


图4-26 创建Java项目“ZooKeeperExample”

2. 添加所需JAR包

为了编写关于ZooKeeper文件系统增删改查的Java应用程序，需要向Java工程中添加JAR包，这些JAR包中包含了可以访问ZooKeeper的Java API，这些JAR包都位于Linux系统的$ZOOKEEPER\_HOME目录下，对于本书而言，就是在/usr/local/zookeeper-3.4.13目录下。读者可以按以下步骤添加该应用程序编写时所需的JAR包。

（1）右键单击Java项目“ZooKeeperExample”，从弹出的菜单中选择『Build Path』→『Configure Build Path…』，如图4-27所示。

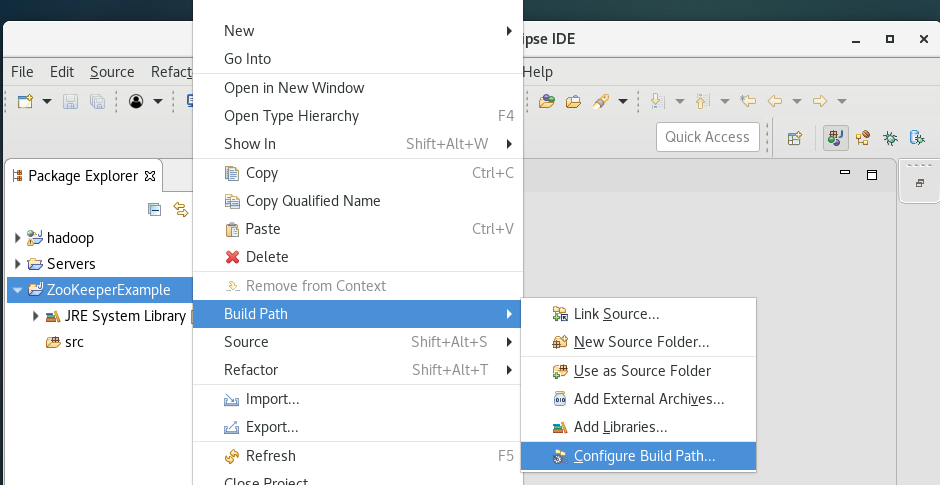


图4-27 进入“ZooKeeperExample”项目“Java Build Path”

（2）进入窗口【Properties for ZooKeeperExample】，可以看到添加JAR包的主界面，如图4-28所示。

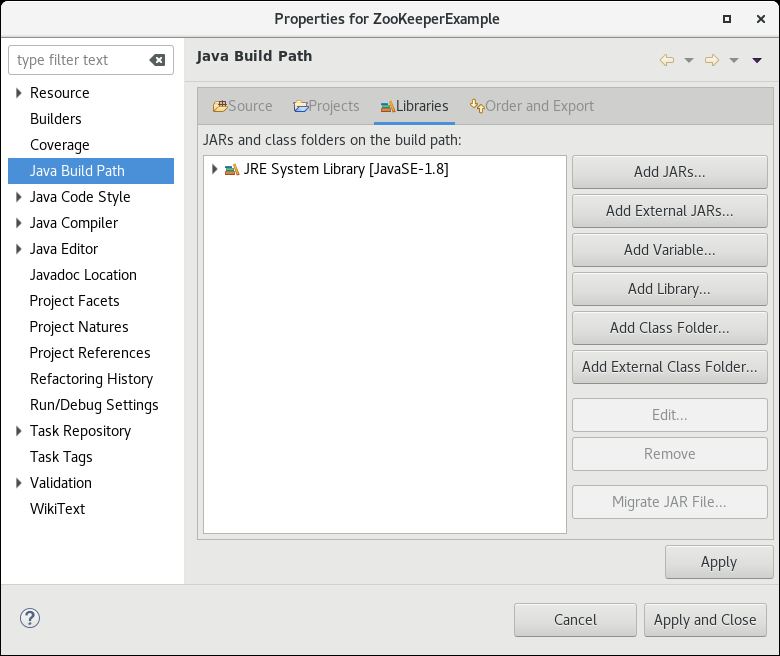


图4-28 添加JAR包主界面

（3）单击图中的按钮Add External JARS，依次添加/usr/local/zookeeper-3.4.13目录下zookeeper-3.4.13.jar和/usr/local/zookeeper-3.4.13/lib目录下slf4j-api-1.7.25.jar。其中添加JAR包zookeeper-3.4.13.jar的过程如图4-29所示，找到此JAR包后选中并单击右上角的OK按钮，这样就成功把zookeeper-3.4.13.jar增加到了当前Java项目中。添加slf4j-apii-1.7.25.jar的过程同此，不再赘述。

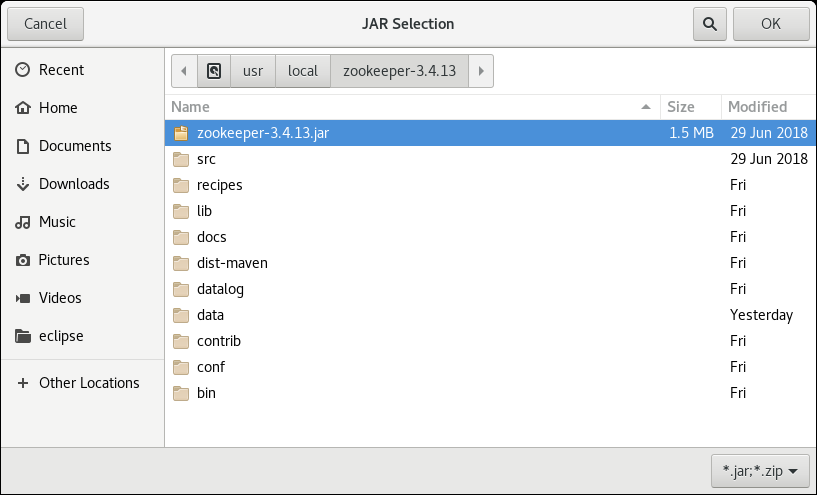


图4-29 添加zookeeper-3.4.13.jar到Java项目中

（4）完成JAR包添加后的界面如图4-30所示，单击按钮Apply and Close。

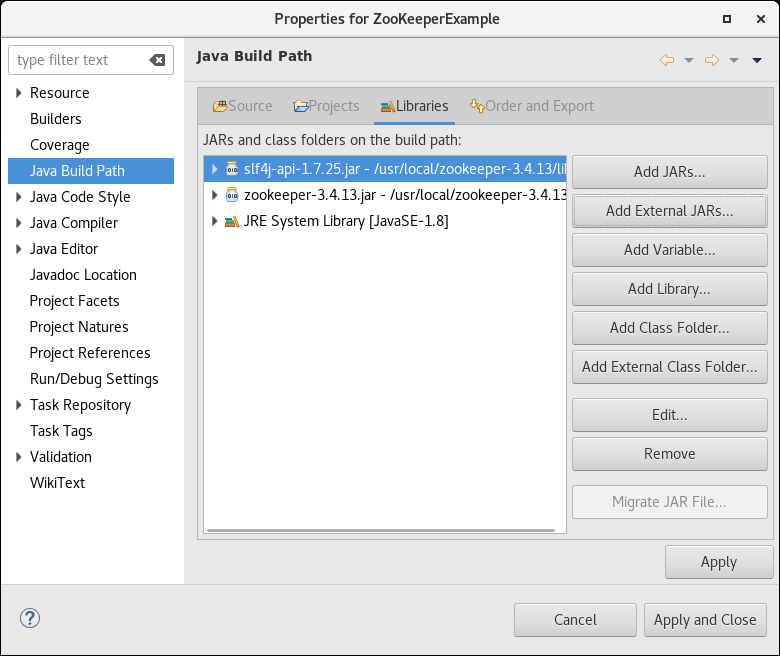


图4-30 完成JAR包添加后的界面

（5）自动返回到Eclipse界面，如图4-31所示，从图中可以看到，项目“ZooKeeperExample”目录树下多了“Referenced Libraries”，内部有以上步骤添加进来的两个JAR包。

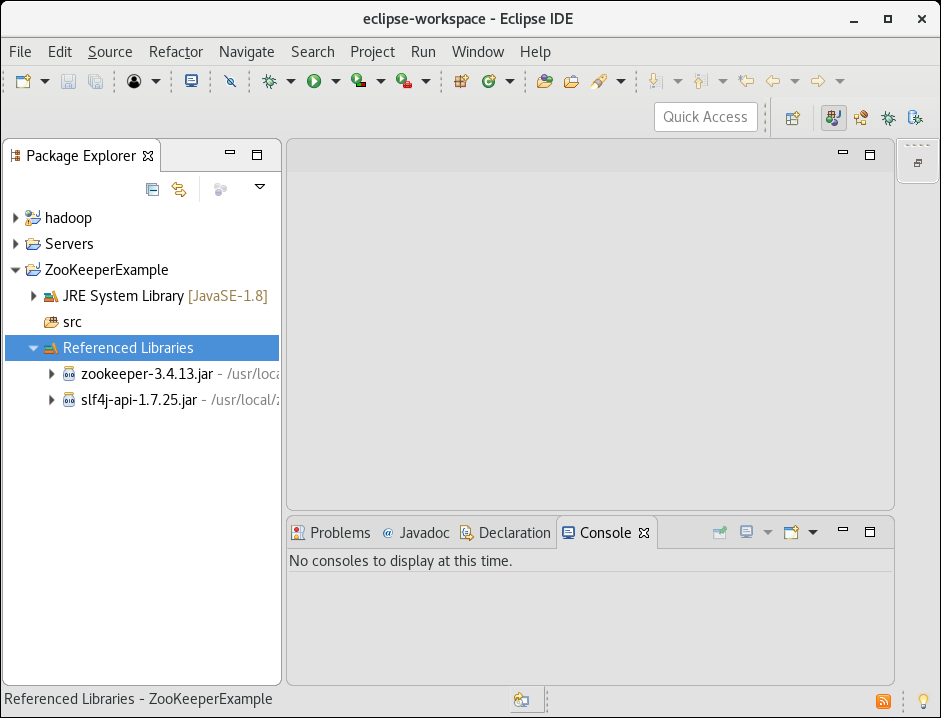


图4-31 添加JAR包后“ZooKeeperExample”项目目录树变化

3. 新建包Package

（1）右键单击Java项目“ZooKeeperExample”，从弹出的菜单中选择『New』→『Package』，如图4-32所示。

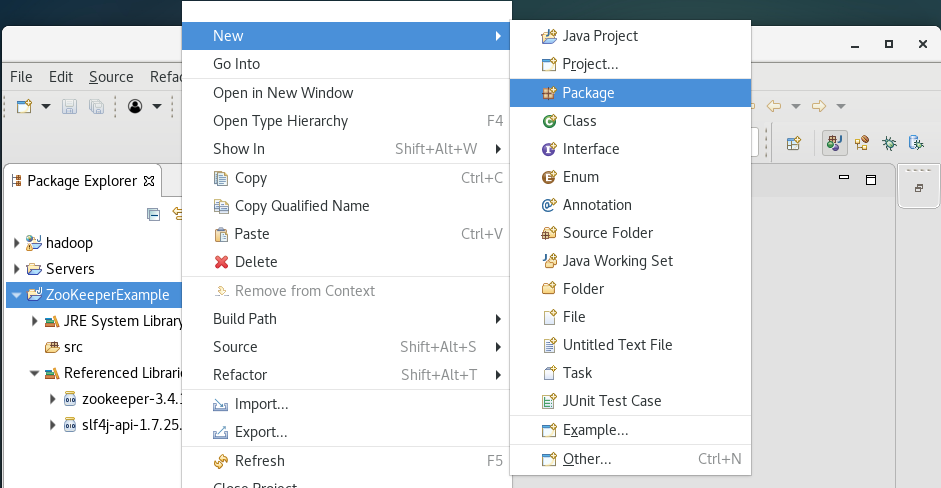


图4-32 进入“ZooKeeperExample”项目新建包窗口

（2）进入窗口【New Java Package】，输入新建包的名字，例如“com.xijing.zookeeper”，如图4-33所示，完成后单击Finish按钮。

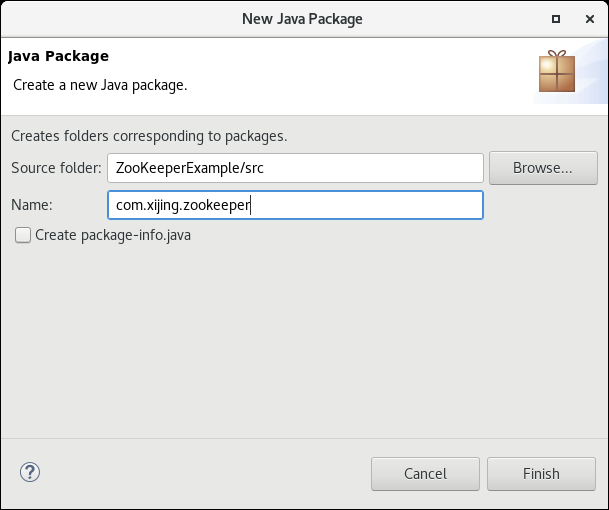


图4-33 新建包“com.xijing.zookeeper”

4. 编写Java程序

下面编写一个Java应用程序，借助ZooKeeper API，实现对ZooKeeper节点ZNode的创建、修改、查看、删除功能。

（1）右键单击Java项目“ZooKeeperExample”中目录“src”下的包“com.xijing.zookeeper”，从弹出的菜单中选择『New』→『Class』，如图4-34所示。

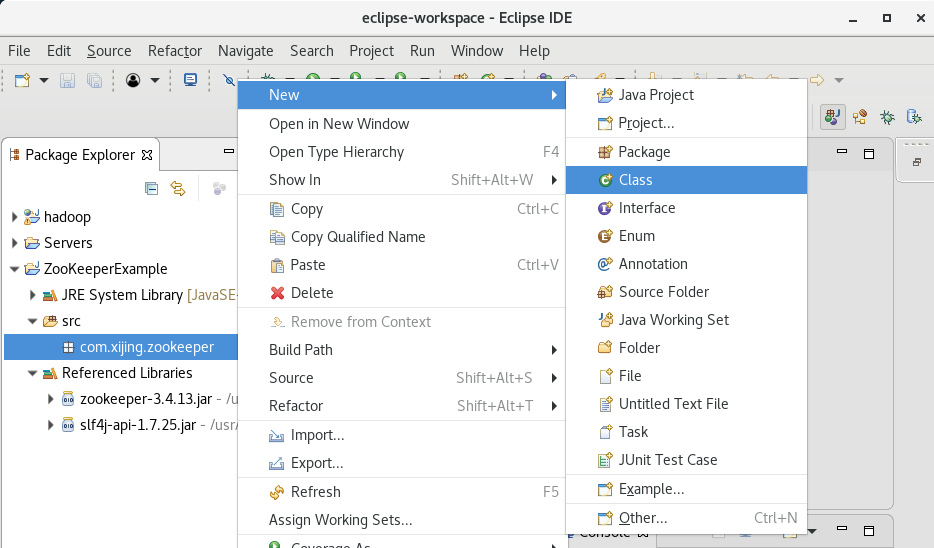


图4-34 进入“com.xijing.zookeeper”包的新建类窗口

（2）进入窗口【New Java Class】。可以看出，由于上步在包“com.xijing.zookeeper”下新建类，故此处不需要选择该类所属包；输入新建类的名字，例如“ZnodeCDRW”，之所以这样命名，是本程序实现的是对ZNode节点的增删改查，C、D、R、W分别对应ACL访问控制列表中节点的操作权限CREATE、READ、WRITE、DELETE，建议读者命名时也要做到见名知意；读者还可以选择是否创建main函数。本实验中新建类“ZnodeCDRW”的具体输入和选择如图4-35所示。完成后单击Finish按钮。

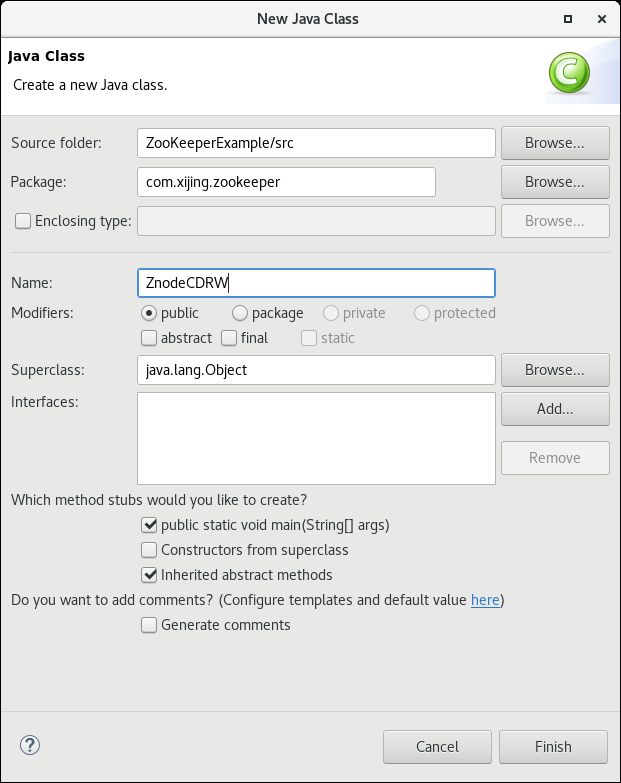


图4-35 新建类“ZnodeCDRW”

（3）自动返回到Eclipse界面，可以看到，Eclipse自动创建一个名为ZnodeCDRW.java的源代码文件，包、类、main方法已出现在代码中，如图4-36所示。

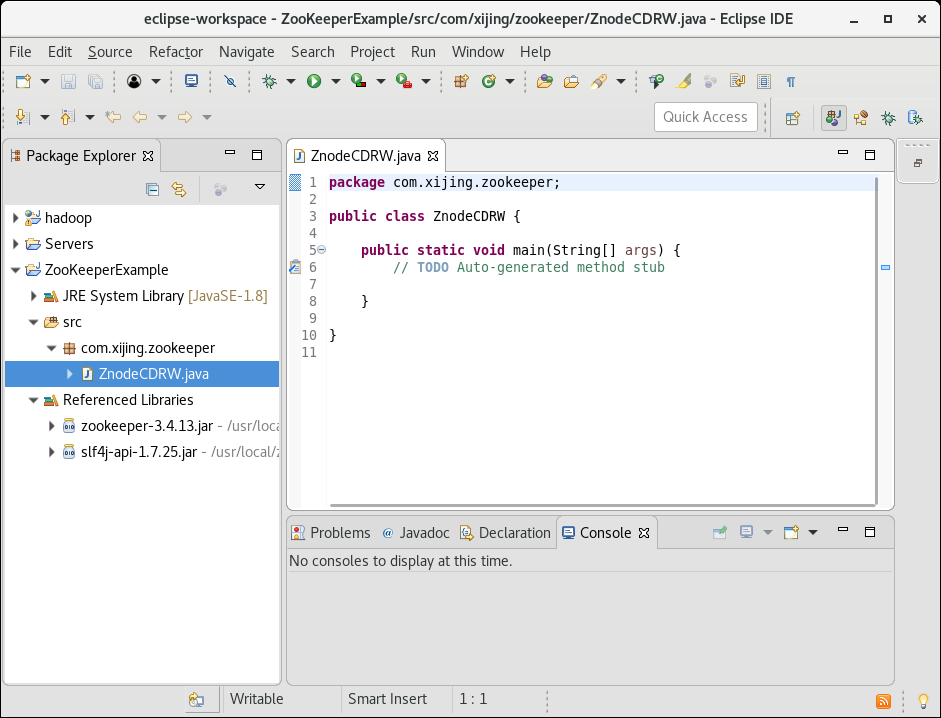


图4-36 新建类“ZnodeCDRW”后的Eclipse界面

（4）为实现程序功能，在该文件中添加代码，该程序完整代码如下所示。

package com.xijing.zookeeper;

import java.io.IOException;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.Watcher;

import org.apache.zookeeper.ZooDefs.Ids;

import org.apache.zookeeper.ZooKeeper;

public class ZnodeCDRW {

// 会话超时时间，设置为与系统默认时间一致

private static final int SESSION\_TIMEOUT=30000;

// 创建 ZooKeeper 实例

ZooKeeper zk;

// 创建 Watcher 实例

Watcher wh=new Watcher(){

public void process(org.apache.zookeeper.WatchedEvent event) {

System.out.println(event.toString());

}

};

// 初始化 ZooKeeper 实例

private void createZKInstance() throws IOException {

zk=new ZooKeeper("localhost:2181",ZnodeCDRW .SESSION\_TIMEOUT,this.wh);

}

private void ZKOperations() throws IOException,InterruptedException,KeeperException {

System.out.println("(1)创建ZooKeeper节点(znode:xijing,数据:xijing colleage,权限:OPEN\_ACL\_UNSAFE,节点类型:persistent");

zk.create("/xijing","xijing colleage".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

System.out.println("(2)查看是否创建成功： ");

System.out.println(new String(zk.getData("/xijing",false,null)));

System.out.println("(3)修改节点数据 ");

zk.setData("/xijing", "xijing university".getBytes(), -1);

System.out.println("(4)查看是否修改成功： ");

System.out.println(new String(zk.getData("/xijing", false, null)));

System.out.println("(5)删除节点 ");

zk.delete("/xijing", -1);

System.out.println("(6)查看节点是否被删除： ");

System.out.println(" 节点状态： ["+zk.exists("/xijing", false)+"]");

}

private void ZKClose() throws InterruptedException {

zk.close();

}

public static void main(String[] args) throws IOException,InterruptedException,KeeperException {

ZnodeCDRW dm=new ZnodeCDRW();

dm.createZKInstance();

dm.ZKOperations();

dm.ZKClose();

}

}

此类包含两个主要的ZooKeeper函数，分别为createZKInstance()和ZKOperations()。其中createZKInstance()函数负责对ZooKeeper实例zk进行初始化。这里使用的ZooKeeper类构造函数是“ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)”对其进行初始化。因此，需要提供初始化所需的参数：连接字符串信息、会话超时时间、一个watcher实例。代码“Watcher wh=new Watcher(){…}”是程序所构造的一个Watcher实例，它能够输出所发生的事件。

ZKOperations()函数定义了对节点的一系列操作，包括：创建ZooKeeper节点create、查看节点getData、修改节点数据setData、查看修改后节点数据、删除节点delete、查看节点是否存在exists。另外，需要注意的是，在创建节点的时候，需要提供节点的名称、数据、权限以及节点类型。此外，使用exists函数时，如果节点不存在将返回一个null值。

5. 编译运行程序

单击Eclipse工具栏中的Run按钮，运行ZnodeCDRW，执行结果如图4-37所示。

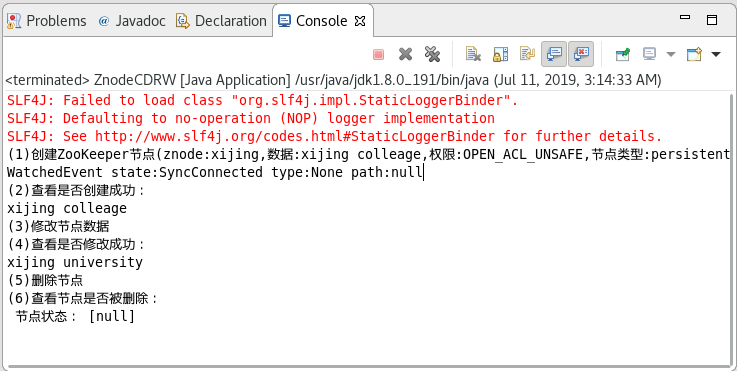


图4-37 ZnodeCDRW运行结果

（三）ZooKeeper编程实践：循环监听

【案例4-3】使用ZooKeeper Java API实现对节点的循环监听。

ZooKeeper可以对某个节点进行监听，监听的事件有以下4种：

（1）NodeCreated：节点创建。

（2）NodeDeleted：节点删除。

（3）NodeDataChanged：节点数据变化。

（4）NodeChildrenChanged：子节点变化，包括子节点删除、创建和数据变化。

但每个事件只能触发一次，之后监听就不再生效，本ZooKeeper编程案例将介绍怎么实现ZooKeeper对节点的循环监听，即永久生效。本案例将实现客户端发起对节点的事务操作以NodeChildrenChanged事件为例，服务端监听到对应的事件后进行相应的操作。关于循环监听的ZooKeeper编程代码依然存放在之前创建的Java项目“ZooKeeperExample”下。本案例的具体实现步骤如下所示。

1. 添加所需JAR包

关于ZooKeeper循环监听的应用程序需要的JAR包包括：/usr/local/zookeeper-3.4.13目录下zookeeper-3.4.13.jar、/usr/local/zookeeper-3.4.13/lib目录下slf4j-api-1.7.25.jar。由于4.4.2节中已详细介绍如何添加所需JAR包，此处不再赘述。

2. 编写Java程序

为了方便管理，本ZooKeeper循环监听应用程序也隶属于包“com.xijing.zookeeper”，为了实现循环监听，需要编写客户端程序和服务端程序。其中客户端程序RepeatWatcherClient.java发起对节点的事务操作，本实例以NodeChildrenChanged事件为例；服务端程序RepeatWatcherServerjava监听到对应的事件后进行相应的操作。

客户端程序RepeatWatcherClient.java的完整代码如下所示。

package com.xijing.zookeeper;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.ZooDefs.Ids;

import org.apache.zookeeper.ZooKeeper;

public class RepeatWatcherClient{

private static final String CONNECT\_STRING = "master:2181,slave1:2181,slave2:2181";

private static final int SESSION\_TIMEOUT = 5000;

private static final String PARENT = "/xijing";

private static final String CHILD = "xuluhui";

public static void main(String[] args) throws Exception {

ZooKeeper zk = new ZooKeeper(CONNECT\_STRING, SESSION\_TIMEOUT, null);

// 客户端创建了一个子节点，会触发NodeChildrenChanged事件

String path = zk.create(PARENT + "/" + CHILD, CHILD.getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT\_SEQUENTIAL);

System.out.println(path);

zk.close();

}

}

服务端程序RepeatWatcherServer.java的完整代码如下所示。

package com.xijing.zookeeper;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

import org.apache.zookeeper.Watcher.Event.EventType;

import org.apache.zookeeper.Watcher.Event.KeeperState;

import org.apache.zookeeper.ZooKeeper;

/\*\*

\* @Description: 循环监听某节点

\*\*/

public class RepeatWatcherServer {

private static ZooKeeper zk;

private static final String CONNECT\_STRING = "master:2181,slave1:2181,slave2:2181";

private static final int SESSION\_TIMEOUT = 5000;

private static final String PARENT = "/xijing";

public static void main(String[] args) throws Exception {

zk = new ZooKeeper(CONNECT\_STRING, SESSION\_TIMEOUT, new Watcher() {

@Override

public void process(WatchedEvent event) {

String path = event.getPath();

EventType type = event.getType();

KeeperState state = event.getState();

System.out.println(path + "\t" + type + "\t" + state);

// 循环监听

try {

zk.getChildren(PARENT, true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

// 添加监听

zk.getChildren(PARENT, true);

// 模拟服务器一直运行

Thread.sleep(Long.MAX\_VALUE);

}

}

3. 编译运行程序

（1）运行RepeatWatcherServer.java，Server控制台输出：

null None SyncConnected

这是获取连接时的事件，每次获得连接都会触发

（2）第一次运行RepeatWatcherClient.java，Client控制台输出：

/xijing/xuluhui0000000000

因为RepeatWatcherClient创建的是PERSISTENT\_SEQUENTIAL类型的节点，所以会自动递增编号。

此时，读者可以看到，Server控制台的内容会实时更新，当前内容为：

null None SyncConnected

/xijing NodeChildrenChanged SyncConnected

（3）第二次运行RepeatWatcherClient..java，Client控制台输出：

/xijing/xuluhui0000000001

读者可以继续看到，Server控制台的内容此时实时更新为：

null None SyncConnected

/xijing NodeChildrenChanged SyncConnected

/xijing NodeChildrenChanged SyncConnected

至此，就成功实现了对某个节点的循环监听！

ZooKeeper编程之循环监听应用程序说明：

（1）监听到对应的事件触发后，本程序只是做了打印，具体的行为应该根据业务逻辑来设计。

（2）监听某个节点的子节点变化（NodeChildrenChangeds）事件，首先该节点应该存在，本例中监听/xijing节点的子节点变化，那么/xijing节点应该提前创建好。

思考与练习题

1. 给ZooKeeper节点配置的id为1-255之间的一个数字，那么当zookeeper集群的数量超过255怎么办？

2. 假设有两个线程，两个线程要同时到MySQL中更新一条数据，对数据库中的数据进行累加更新。由于在分布式环境下，这两个线程可能存在于不同机器上的不同JVM进程中，所以这两个线程的关系就是跨主机和跨进程的，使用Java中的synchronized锁是无法实现的。思考如何使用ZooKeeper实现分布式锁？

参考文献

[1] 倪超. 从Paxos到ZooKeeper：分布式一致性原理与实践[M]. 北京:电子工业出版社,2015.

[2] Apache ZooKeeper[EB/OL]. https://zookeeper.apache.org/releases.html

[3] GitHub-Apache ZooKeeper[EB/OL]. https://github.com/apache/zookeeper.

[4] Apache ZooKeeper. Apache ZooKeeper Download[EB/OL]. https://zookeeper.apache.org/releases.html.

[5] Apache Software Foundation. Apache ZooKeeper 3.4.13官方参考指南[EB/OL]. [2018-12-15]. https://zookeeper.apache.org/doc/r3.4.13/.

[6] Apache Software Foundation. ZooKeeper 3.4.13 API[EB/OL]. [2018-12-15]. https://zookeeper.apache.org/doc/r3.4.13/api/index.html.