ZooKeeper 基础知识

目录

CAP 理论

1、**一致性（Consistency）（C）**：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

2、**可用性（Availability）（A）**：在集群中一部分节点故障后，在一定时间内，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

3、**分区容错性（Partition tolerance）（P）**：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在 C 和 A 之间做出选择。

ACID:

Automaticy 原子性
Consistency 一致性
Isolation 隔离性
Durability 持久性

CAP:

Consistency 最终一致性

Availability 可用性

Partition tolerance 分区容错性

BASE:

Basically Available 基本可用

Soft state 软状态

Eventually consistent 最终一致性

分布式事务：

2PC：Two-Phase Commit

阶段一：提交事务请求

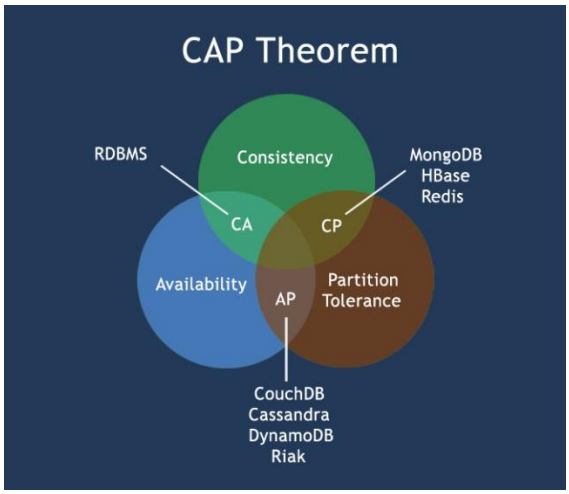
阶段二：执行事务请求

3PC：Three-Phase Commit

阶段一：CanCommit

阶段二：PreCommit

阶段三：DoCommit



定理：**任何分布式存储系统只可同时满足二点，没法三者兼顾**。

忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍

原因总的来说就是：数据存在的节点越多，分区容错性（P）越高，但要复制更新的数据就越多，一致性（C）就越难保证。为了保证一致性，更新所有节点数据所需要的时间就越长，可用性（A）就会降低

MySQL：满足 CA

ZooKeeper：满足 CP

分布式系统的 CAP 理论：http://www.hollischuang.com/archives/666

1、引言

Hadoop 集群当中 N 多的配置信息如何做到全局一致并且单点修改迅速响应到整个集群？

--- 配置管理

Hadoop 集群中的 namonode 和 resourcemanager 的单点故障怎么解决？
--- 集群的主节点的单点故障

ZooKeeper 的学习要点：

- ZooKeeper 是什么？What
- ZooKeeper 干什么用？Where
- ZooKeeper 怎么用？How
- ZooKeeper 底层实现原理是什么？Why

2、ZooKeeper 是什么

What is ZooKeeper？

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them ,which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现。它提供了简单原始的功能，分布式应用可以基于它实现更高级的服务，比如分布式同步，配置管理，集群管理，命名管理，队列管理。它被设计为易于编程，使用文件系统目录树作为数据模型。服务端跑在 java 上，提供 java 和 C 的客户端 API

众所周知，协调服务非常容易出错，但是却很难恢复正常，例如，协调服务很容易处于竞态以至于出现死锁。我们设计 ZooKeeper 的目的是为了减轻分布式应用程序所承担的协调任务

ZooKeeper 是集群的管理者，监视着集群中各节点的状态，根据节点提交的反馈进行下一步合理的操作。最终，将简单易用的接口和功能稳定，性能高效的系统提供给用户。

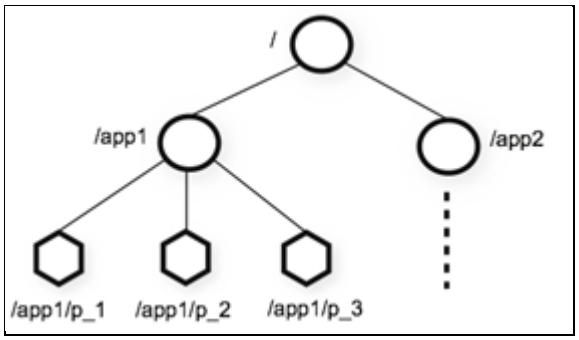
官网地址：http://ZooKeeper.apache.org/
官网快速开始地址：http://ZooKeeper.apache.org/doc/trunk/ZooKeeperStarted.html
官网 API 地址：http://ZooKeeper.apache.org/doc/r3.4.10/api/index.html

3、ZooKeeper 提供了什么

3.1、文件系统

ZooKeeper 的命名空间就是 ZooKeeper 应用的文件系统，它和 linux 的文件系统很像，也是树状，这样就可以确定每个路径都是唯一的，对于命名空间的操作必须都是绝对路径操作。与 linux 文件系统不同的是，linux 文件系统有目录和文件的区别，而 ZooKeeper 统一叫做 znode，一个 znode 节点可以包含子 znode，同时也可以包含数据。

所以总结说来，znode 即是文件夹又是文件的概念，所以在 ZooKeeper 这里面就不叫文件文件也不叫文件夹，叫 znode，每个 znode 有唯一的路径标识，既能存储数据，也能创建子 znode。但是 znode 只适合存储非常小量的数据，不能超过 1M，最好小于 1K。



下面是关于 Znode 的介绍（非常重要）：

1、Znode 有两种类型：

短暂（ephemeral）（断开连接自己删除）

持久（persistent）（断开连接不删除）

2、Znode 有四种形式的目录节点（默认是 persistent ）

PERSISTENT	持久化 znode 节点，一旦创建这个 znode 点存储的数据不会主动消失，除非是客户端主动的 delete
PERSISTENT_SEQUENTIAL	自动增加顺序编号的 znode 节点，比如 ClientA 去 zk service 上建立一个 znode 名字叫做/zk/conf，指定了这种类型的节点后 zk 会创建/zk/conf0000000000，ClientB 再去创建就是创建/zk/conf0000000001，ClientC 是创建/zk/conf0000000002，以后任意 Client 来创建这个 znode 都会得到一个比当前 zk 命名空间最大 znode 编号+1 的 znode，也就说任意一个 Client 去创建 znode 都是保证得到的 znode 是递增的，而且是唯一的
EPHEMERAL	临时 znode 节点，Client 连接到 zk service 的时候会建立一个 session，之后用这个 zk 连接实例创建该类型的 znode，一旦 Client 关闭了 zk 的连接，服务器就会清除 session，然后这个 session 建立的 znode 节点都会从命名空间消失。总结就是，这个类型的 znode 的生命周期是和 Client 建立的连接一样的。比如 ClientA 创建了一个 EPHEMERAL 的/zk/conf0000000011 的 znode 节点，

	一旦 ClientA 的 zk 连接关闭，这个 znode 节点就会消失。整个 zk service 命名空间里就会删除这个 znode 节点
EPHEMERAL_SEQUENTIAL	临时自动编号节点，znode 节点编号会自动增加，但是会随 session 消失而消失

3、创建 znode 时设置顺序标识，znode 名称后会附加一个值，顺序号是一个单调递增的计数器，由父节点维护

4、在分布式系统中，顺序号可以被用于为所有的事件进行全局排序，这样客户端可以通过顺序号推断事件的顺序

5、EPHEMERAL 类型的节点不能有子节点

6、客户端可以在 znode 上设置监听器

3.2、监听机制

客户端注册监听它关心的目录节点，当目录节点发生变化（**数据改变**、**节点删除**、**子目录节点增加删除**）时，ZooKeeper 会通知客户端。监听机制保证 ZooKeeper 保存的任何的数据的任何改变都能快速的响应到监听了该节点的应用程序

监听器的工作机制，其实是在客户端会专门创建一个监听线程，在本机的一个端口上等待 zk 集群发送过来事件

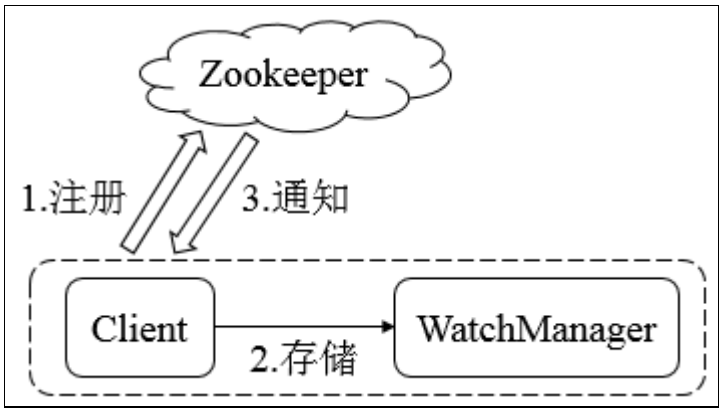
		触发事件				
		create		delete		setData
		znode	child	znode	child	znode
创建Watch的API	exist	NodeCreated		NodeDeleted		NodeDataChanged
	getData			NodeDeleted		NodeDataChanged
	getChildren		NodeChildrenChanged	NodeDeleted	NodeChildrenChanged	

注意：监听只生效一次
So，怎么做到循环监听？

具体请看 ZooKeeper Shell 和 API 应用时的使用

3.3、监听工作原理

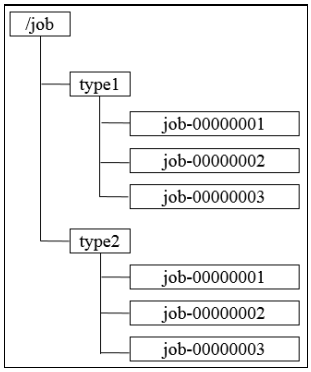
ZooKeeper 的 Watcher 机制主要包括**客户端线程**、**客户端 WatcherManager**、**Zookeeper 服务器**三部分。客户端在向 ZooKeeper 服务器注册的同时，会将 Watcher 对象存储在客户端的 WatcherManager 当中。当 ZooKeeper 服务器触发 Watcher 事件后，会向客户端发送通知，客户端线程从 WatcherManager 中取出对应的 Watcher 对象来执行回调逻辑



3.4、ZooKeeper 典型应用场景

3.4.1、命名服务

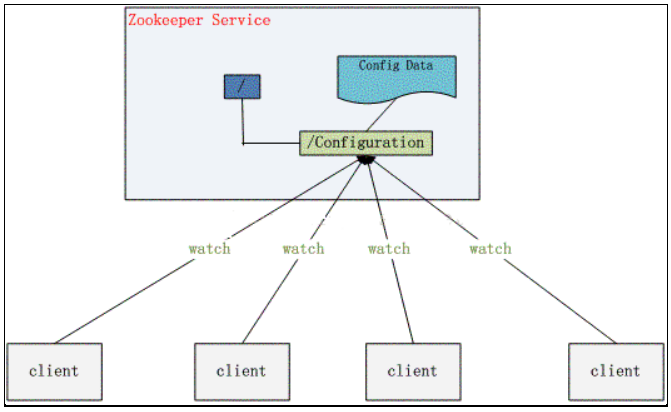
命名服务是分布式系统中较为常见的一类场景，分布式系统中，被命名的实体通常可以是集群中的机器、提供的服务地址或远程对象等，通过命名服务，客户端可以根据指定名字来获取资源的实体、服务地址和提供者的信息。Zookeeper 也可帮助应用系统通过资源引用的方式来实现对资源的定位和使用，广义上的命名服务的资源定位都不是真正意义上的实体资源，在分布式环境中，上层应用仅仅需要一个全局唯一的名字。Zookeeper 可以实现一套分布式全局唯一 ID 的分配机制。



3.4.2、配置管理

程序总是需要配置的，如果程序分散部署在多台机器上，要逐个改变配置就变得困难。现在把这些配置全部放到 ZooKeeper 上去，保存在 ZooKeeper 的某个目录节点中，然后所有相

关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 ZooKeeper 的通知，然后从 ZooKeeper 获取新的配置信息应用到系统中就好

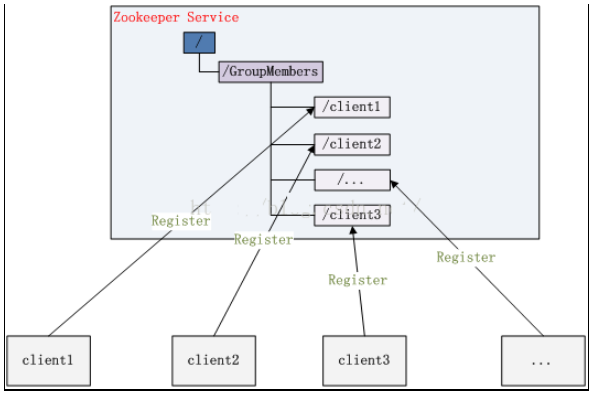


3.4.3、集群管理

所谓集群管理无在乎两点：**是否有机器退出和加入、选举 master**。

对于第一点，所有机器约定在父目录 GroupMembers 下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与 ZooKeeper 的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除，于是，所有人都知道：有兄弟挂了。新机器加入也是类似，所有机器收到通知：新兄弟目录加入，又多了个新兄弟。

对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为 master 就好。当然，这只是其中的一种策略而已，选举策略完全可以由管理员自己制定。



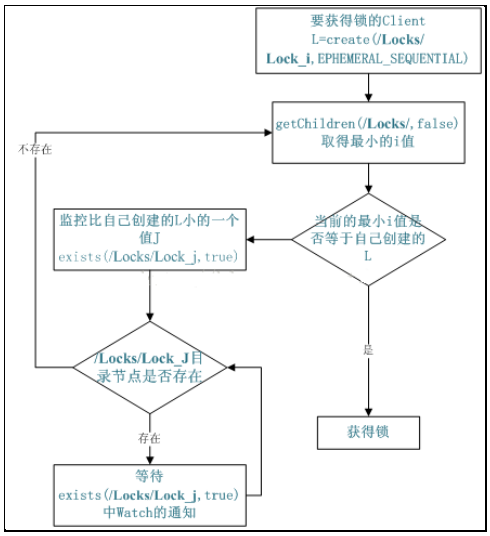
3.4.4、分布式锁

有了 ZooKeeper 的一致性文件系统，锁的问题变得容易。
锁服务可以分为两三类
一个是写锁，对写加锁，保持独占，或者叫做排它锁，独占锁
一个是读锁，对读加锁，可共享访问，释放锁之后才可进行事务操作，也叫共享锁

一个是控制时序，叫时序锁

对于第一类，我们将 ZooKeeper 上的一个 znode 看作是一把锁，通过 createznode 的方式来实现。所有客户端都去创建 /distribute_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除掉自己创建的 distribute_lock 节点就释放出锁

对于第二类， /distribute_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选 master 一样，编号最小的获得锁，用完删除，依次有序

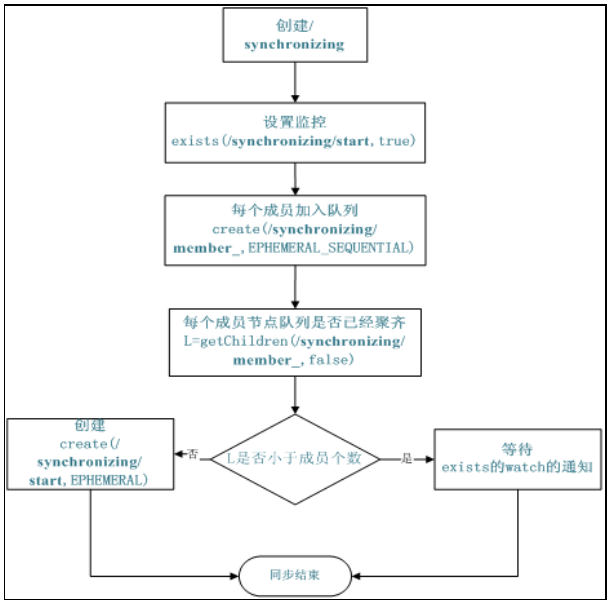


3.4.5、队列管理

两种类型的队列：
1、**同步队列**：当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。
2、**先进先出队列**：队列按照 FIFO 方式进行入队和出队操作。

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。
第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。

同步队列的流程图：



4、ZooKeeper 特点/设计目的

ZooKeeper 作为一个集群提供数据一致的协调服务，自然，最好的方式就是在整个集群中的各服务节点进行数据的复制和同步。

数据复制的好处：
1、容错：一个节点出错，不至于让整个集群无法提供服务、、
2、扩展性：通过增加服务器节点能提高 ZooKeeper 系统的负载能力，把负载分布到多个节点上
3、高性能：客户端可访问本地 ZooKeeper 节点或者访问就近的节点，依次提高用户的访问速度

从客户端读写访问的透明度来看，数据复制集群系统分下面两种：
1、写主：对数据修改的请求提交给主节点，对读数据请求没有限制，任何节点都能响应。

1、 **最终一致性**：client 不论连接到哪个 Server，展示给它都是同一个视图，这是 ZooKeeper 最重要的性能。

2、 **可靠性**：具有简单、健壮、良好的性能，如果消息 m 被到一台服务器接受，那么它将被所有的服务器接受。

3、 **实时性**：ZooKeeper 保证客户端将在一个时间间隔范围内获得服务器的更新信息，或者服务器失效的信息。但由于网络延时等原因，ZooKeeper 不能保证两个客户端能同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用 sync()接口。

4、 **等待无关（wait-free）**：慢的或者失效的 client 不得干预快速的 client 的请求，使得每个 client 都能有效的等待。

5、 **原子性**：更新只能成功或者失败，没有中间状态。

6、 **顺序性**：包括全局有序和偏序两种：全局有序是指如果在一台服务器上消息 a 在消息 b 前发布，则在所有 Server 上消息 a 都将在消息 b 前被发布；偏序是指如果一个消息 b 在消息 a 后被同一个发送者发布，a 必将排在 b 前面。

5、ZooKeeper 集群搭建

5.1、ZooKeeper 软件安装须知

鉴于 ZooKeeper 本身的特点，服务器集群的节点数推荐设置为奇数台。我这里我规划为三台，为别为 hadoop01,hadoop02,hadoop03

注意：牢记我说的 linux 上安装大数据领域技术软件的安装四大步骤

5.2、具体安装

5.2.1、上网找 ZooKeeper 的软件安装，并下载下来

下载地址：http://mirrors.hust.edu.cn/apache/ZooKeeper/
版本号：ZooKeeper-3.4.7.tar.gz

5.2.2、解压安装到自己的目录

tar -zxvf ZooKeeper-3.4.7.tar.gz -C apps/

```
[root@hadoop01 zookeeper-3.4.7]# ll
total 1580
drwxr-xr-x.  2 1000 1000    4096 Nov 10  2015 bin
-rw-rw-r--.  1 1000 1000   83235 Nov 10  2015 build.xml
-rw-rw-r--.  1 1000 1000   87716 Nov 10  2015 CHANGES.txt
drwxr-xr-x.  2 1000 1000    4096 Nov 10  2015 conf
drwxr-xr-x. 10 1000 1000    4096 Nov 10  2015 contrib
drwxr-xr-x.  2 1000 1000    4096 Nov 10  2015 dist-maven
drwxr-xr-x.  6 1000 1000    4096 Nov 10  2015 docs
-rw-rw-r--.  1 1000 1000    1953 Nov 10  2015 ivysettings.xml
-rw-rw-r--.  1 1000 1000    3375 Nov 10  2015 ivy.xml
drwxr-xr-x.  4 1000 1000    4096 Nov 10  2015 lib
-rw-rw-r--.  1 1000 1000   11938 Nov 10  2015 LICENSE.txt
-rw-rw-r--.  1 1000 1000     171 Nov 10  2015 NOTICE.txt
-rw-rw-r--.  1 1000 1000    1770 Nov 10  2015 README_packaging.txt
-rw-rw-r--.  1 1000 1000    1585 Nov 10  2015 README.txt
drwxr-xr-x.  5 1000 1000    4096 Nov 10  2015 recipes
drwxr-xr-x.  8 1000 1000    4096 Nov 10  2015 src
-rw-rw-r--.  1 1000 1000 1360000 Nov 10  2015 zookeeper-3.4.7.jar
-rw-rw-r--.  1 1000 1000     819 Nov 10  2015 zookeeper-3.4.7.jar.asc
-rw-rw-r--.  1 1000 1000      33 Nov 10  2015 zookeeper-3.4.7.jar.md5
-rw-rw-r--.  1 1000 1000      41 Nov 10  2015 zookeeper-3.4.7.jar.sha1
```

ZooKeeper 运行最重要的四个东西

5.2.3、修改配置文件

cd conf/

mv zoo_sample.cfg zoo.cfg

```
[root@hadoop01 conf]# ll
total 12
-rw-rw-r--. 1 1000 1000  535 Nov 10  2015 configuration.xsl
-rw-rw-r--. 1 1000 1000 2161 Nov 10  2015 log4j.properties
-rw-rw-r--. 1 1000 1000  922 Nov 10  2015 zoo_sample.cfg
[root@hadoop01 conf]# mv zoo_sample.cfg zoo.cfg
[root@hadoop01 conf]# ll
total 12
-rw-rw-r--. 1 1000 1000  535 Nov 10  2015 configuration.xsl
-rw-rw-r--. 1 1000 1000 2161 Nov 10  2015 log4j.properties
-rw-rw-r--. 1 1000 1000  922 Nov 10  2015 zoo.cfg
[root@hadoop01 conf]#
```

vi zoo.cfg

修改配置项：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/apps/zkdata
clientPort=2181

server.1=hadoop01:2888:3888
server.2=hadoop02:2888:3888
server.3=hadoop03:2888:3888
```

保存退出

```
补充：假如要配置 observer，那么请把 zoo.cfg 改成如下配置：
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/apps/zkdata
dataLogDir=/root/apps/zklog
clientPort=2181

server.1=hadoop01:2888:3888
server.2=hadoop02:2888:3888
server.3=hadoop03:2888:3888
server.4=hadoop04:2888:3888:observer   // 如果有第四台机器，并且想配置 observer
```

配置参数解析：

```
tickTime
```

基本事件单元，以毫秒为单位。它用来控制心跳和超时，默认情况下最小的会话超时时间为两倍的 tickTime。

initLimit
此配置表示，允许 follower （相对于 leader 而言的“客户端”）连接并同步到 leader 的初始化连接时间，它以 tickTime 的倍数来表示。当超过设置倍数的 tickTime 时间，则连接失败。

syncLimit
此配置表示，leader 与 follower 之间发送消息，请求和应答时间长度。如果 follower 在设置的时间内不能与 leader 进行通信，那么此 follower 将被丢弃。

dataDir
存储内存中数据库快照的位置
注意：如果需要保留日志信息，那么可以考虑配置 dataLogDir 的位置，这个位置就是日志的存储目录。通常情况下是分开存储的。并且应该谨慎地选择日志存放的位置，使用专用的日志存储设备能够大大地提高系统的性能，如果将日志存储在比较繁忙的存储设备上，那么将会在很大程度上影响系统的性能。

clientPort
监听客户端连接的端口，默认是 2181，最好不要修改

最后再增加 ZooKeeper 的服务器列表信息，格式为：
server.id=主机名:心跳端口:选举端口
例子：server.1=hadoop01:2888:3888
其中 id 虽然可以随便写，但是有两点要求，第一不能重复，第二范围是 1-255，并且对应服务器列表上还得存在对应的 id 文件，具体看下面操作

然后分发至其他服务器：
[hadoop @hadoop01 zkdata]# scp -r ZooKeeper-3.4.7/ hadoop@hadoop02:$PWD
[hadoop @hadoop01 zkdata]# scp -r ZooKeeper-3.4.7/ hadoop @hadoop03:$PWD

然后是最重要的步骤，一定不能忘了。
去你的各个 ZooKeeper 服务器节点，新建目录 dataDir=/home/hadoop/apps/zkdata，这个目录就是你在 zoo.cfg 中配置的 dataDir 的目录，建好之后，在里面新建一个文件，文件名叫 myid,里面存放的内容就是服务器的 id,就是 server.1=hadoop01:2888:3888 当中的 id,就是 1，那么对应的每个服务器节点都应该做类似的操作
拿服务器 hadoop01 举例：
[hadoop @hadoop01 zkdata]# mkdir /home/hadoop/apps/zkdata
[hadoop @hadoop01 zkdata]# echo 1 > myid

当以上所有步骤都完成时，意味着我们 ZooKeeper 的配置文件相关的修改都做完了。

5.2.4、启动软件，并验证安装是否成功

先配置环境变量
vi ~/.bashrc

增加两行：
export ZOOKEEPER_HOME=/home/hadoop/apps/ZooKeeper-3.4.7
export PATH=$PATH:$ZOOKEEPER_HOME/bin

保存退出，执行 source ~/.bashrc

启动命令：**zkServer.sh start**
注意：虽然我们在配置文件中写明了服务器的列表信息，但是，我们还是需要去每一台服务器去启动，不是一键启动集群模式

然后检查在每台服务器之间是不是都启动了有 QuorumPeerMain 进程，并检查每台服务器的角色，使用命令：
jps 检查 QuorumPeerMain 进程
zkServer.sh status 查看服务器角色（leader or follower）

6、ZooKeeper 集群使用

6.1、ZooKeeper 集群 cli 使用

首先，我们可以是用命令 bin/zkCli.sh 进入 ZooKeeper 的命令行客户端，这种是直接连接本机的 ZooKeeper 服务器，还有一种方式，可以连接其他的 ZooKeeper 服务器，只需要我们在命令后面接一个参数-server 就可以了。例如：zkCli.sh -server hadoop01:2181

进入命令行之后，键入 help 可以查看简易的命令帮助文档，如下图

```
[zk: localhost:2181(CONNECTED) 2] help
ZooKeeper -server host:port cmd args
        connect host:port
        get path [watch]
        ls path [watch]
        set path data [version]
        rmr path
        delquota [-n|-b] path
        quit
        printwatches on|off
        create [-s] [-e] path data acl
        stat path [watch]
        close
        ls2 path [watch]
        history
        listquota path
        setAcl path acl
        getAcl path
        sync path
        redo cmdno
        addauth scheme auth
        delete path [version]
        setquota -n|-b val path
```

查看 znode 子节点内容 ls / ls /ZooKeeper
创建 znode 节点 create /zk "myData"
获取 znode 数据 get /ZooKeeper get /ZooKeeper/node1
设置 znode 数据 set /zk "myData1"
监听 znode 事件 ls /ZooKeeper watch ## 就对一个节点的**子节点变化事件**注册了监听 get /ZooKeeper watch ## 就对一个节点的**数据内容变化事件**注册了监听
创建临时 znode 节点 create -e /zk "myData"
创建顺序 znode 节点 create -s /zk "myData"
删除 znode 节点 delete /zk ## 只能删除没有子 znode 的 znode rmr /zk ## 不管里头有多少 znode，统统删除

znode 数据信息字段解释：

mydata 节点数据 **cZxid = 0x400000093** 节点创建的时候的 zxid The zxid of the change that caused this znode to be created. **ctime = Fri Dec 02 16:41:50 PST 2016** 节点创建的时间

The time in milliseconds from epoch when this znode was created.

mZxid = 0x400000093 节点修改的时候的 zxid，与子节点的修改无关
The zxid of the change that last modified this znode.

mtime = Fri Dec 02 16:41:50 PST 2016 节点的修改的时间
The time in milliseconds from epoch when this znode was last modified.

pZxid = 0x400000093 和子节点的创建/删除对应的 zxid，和修改无关，和孙子节点无关
The zxid of the change that last modified children of this znode.

cversion = 0 子节点的更新次数
The number of changes to the children of this znode.

dataVersion = 0 节点数据的更新次数
The number of changes to the data of this znode.

aclVersion = 0 节点（ACL）的更新次数
The number of changes to the ACL of this znode.

ephemeralOwner = 0x0 如果该节点为 ephemeral 节点, ephemeralOwner 值表示与该节点绑定的 session id. 如果该节点不是 ephemeral 节点, ephemeralOwner 值为 0
The session id of the owner of this znode if the znode is an ephemeral node. If it is not an ephemeral node, it will be zero.

dataLength = 6 节点数据的字节数
The length of the data field of this znode.

numChildren = 0 子节点个数，不包含孙子节点
The number of children of this znode.

ZooKeeper 四字命令：
使用格式：
[root@hadoop02 ~]# echo conf|nc hadoop02 2181
注意：需要安装 nc，不然不能使用

conf	输出相关服务配置的详细信息
cons	列出所有连接到服务器的客户端的完全的连接/会话的详细信息。包括“接受/发送”的包数量、会话 id、操作延迟、最后的操作执行等等信息
dump	列出未经处理的会话和临时节点
envi	输出关于服务环境的详细信息（区别于 conf 命令）
reqs	列出未经处理的请求
ruok	测试服务是否处于正确状态。如果确实如此，那么服务返回“imok ”，否则不做任何相应

stat	输出关于性能和连接的客户端的列表
wchs	列出服务器 watch 的详细信息
wchc	通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表
wchp	通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径

6.2、ZooKeeper 集群 Java API 使用

create(path, data, flags): 创建一个 znode, path 是其路径，data 是存储在该 ZNode 上的数据，flags 常用的有: PERSISTEN, PERSISTENT_SEQUENTAIL, EPHEMERAL, EPHEMERAL_SEQUENTAIL
delete(path, version): 删除一个 ZNode，可以通过 version 删除指定的版本，如果 version 是-1 的话，表示删除所有的版本
exists(path, watch): 判断指定 ZNode 是否存在，并设置是否 Watch 这个 ZNode。这里如果要设置 Watcher 的话，Watcher 是在创建 ZooKeeper 实例时 指定的，如果要设置特定的 Watcher 的话，可以调用另一个重载版本的 exists(path, watcher)。以下几个带 watch 参数的 API 也都类似
getData(path, watch): 读取指定 ZNode 上的数据，并设置是否 watch 这个 ZNode
setData(path, watch): 更新指定 ZNode 的数据，并设置是否 Watch 这个 ZNode
getChildren(path, watch): 获取指定 ZNode 的所有子 ZNode 的名字，并设置是否 Watch 这个 ZNode
sync(path): 把所有在 sync 之前的更新操作都进行同步，达到每个请求都在半数以上的 ZooKeeper Server 上生效。path 参数目前没有用
setAcl(path, acl): 设置指定 ZNode 的 Acl 信息
getAcl(path): 获取指定 ZNode 的 Acl 信息

具体见代码

```
package com.ghgj.zk;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import org.apache.ZooKeeper.CreateMode;
import org.apache.ZooKeeper.ZooDefs.Ids;
import org.apache.ZooKeeper.ZooKeeper;
import org.apache.ZooKeeper.data.ACL;
import org.apache.ZooKeeper.data.Stat;

public class ZooKeeperDemo {

    // 客户端去请求链接的时候的服务器链接地址信息
```

```
    private static String connectString = "hadoop02:2181,hadoop03:2181,hadoop04:2181";

    // 客户端去请求链接的超时时长
    private static int sessionTimeout = 4000;

    // 节点名称，统一命名
    private static String znode = "/zk/huangbo1";

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

        // 拿 ZooKeeper 链接
        ZooKeeper zk = new ZooKeeper(connectString, sessionTimeout, null);

        // 创建 znode
        String createdNode = zk.create(znode, "huangbo".getBytes(), Ids.OPEN_ACL_UNSAFE,
CreateMode.PERSISTENT);
        System.out.println(createdNode+"节点创建成功");

        // 查看节点
        byte[] data = zk.getData(znode, null, null);
        System.out.println(new String(data));

        // 修改节点数据
        Stat setData = zk.setData(znode, "huangbo-xifu".getBytes(), -1);
        System.out.println(new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").format(new
Date(setData.getMtime())));
        byte[] data1 = zk.getData(znode, null, null);
        System.out.println(new String(data1));

        // 获取 znode 的权限信息
        List<ACL> aclList = zk.getACL(znode, null);
        for(ACL acl: aclList){
            System.out.println(acl.getPerms());
        }

        // 判断节点是否存在
        Stat exists = zk.exists(znode, null);
        System.out.println(null != exists?true:false);

        // 获取子节点
        List<String> children = zk.getChildren("/", null);
        for(String child: children){
            System.out.println(child);
        }
```

```
        // 删除节点
        zk.delete(znode, -1);
        Stat exists1 = zk.exists(znode, null);
        System.out.println(null == exists1?"删除成功":"删除失败");

        zk.close();
    }
}
```