**1.1Linux安装zk**

1.安装jdk 至少需要jdk1.7 ,配置好环境变量

2.去官网下载 zk包 <http://zookeeper.apache.org/releases.html#download>

3.接下来就可以运行zk 了

4.为了方便操作可以配置一下环境变量 在/etc/profile 文件中加入下面几行

export ZOOKEEPER\_HOME=/usr/zookeeper

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

然后source /etc/profile 重新加载文件 就OK了

**1.2 zk使用第一步(单机模式启动zk)**

进去解压的docker文件夹的目录 在conf文件夹里面创建一个文件 内容:

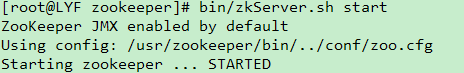
tickTime=2000 --心跳时间 服务端之间相互确认的间隔 (ms)

dataDir=/var/lib/zookeeper ---存放数据库的快照和日志

clientPort=2181 ----服务器端监听的客户端链接的端口

启动zk:

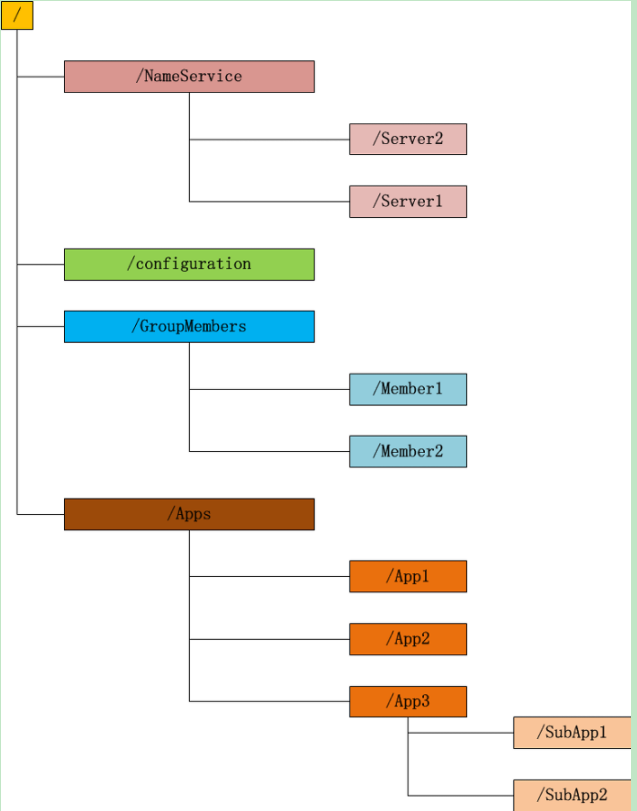
zk目录下 bin/zkServer.sh start



启动客户端连接服务端

bin/zkCli.sh -server 127.0.0.1:2181

**1.3Zookeeper 的节点Znode介绍:**



图中的每个节点称为一个znode. 每个znode由3部分组成:

1.stat：此为状态信息, 描述该znode的版本, 权限等信息.

2.data：与该znode关联的数据.

3.children：该znode下的子节点.

ZooKeeper目录树中每一个节点对应一个Znode。每个Znode维护着一个属性结构，它包含着版本号(dataVersion)，时间戳(ctime,mtime)等状态信息。ZooKeeper正是使用节点的这些特性来实现它的某些特定功能。每当Znode的数据改变时，他相应的版本号将会增加。每当客户端检索数据时，它将同时检索数据的版本号。并且如果一个客户端执行了某个节点的更新或删除操作，他也必须提供要被操作的数据版本号。如果所提供的数据版本号与实际不匹配，那么这个操作将会失败。

Znode是客户端访问ZooKeeper的主要实体，它包含以下几个特征：

（1）Watches

客户端可以在节点上设置watch(我们称之为监视器)。当节点状态发生改变时(数据的增、删、改)将会触发watch所对应的操作。当watch被触发时，ZooKeeper将会向客户端发送且仅发送一条通知，因为watch只能被触发一次。

（2）数据访问

ZooKeeper中的每个节点存储的数据要被原子性的操作。也就是说读操作将获取与节点相关的所有数据，写操作也将替换掉节点的所有数据。另外，每一个节点都拥有自己的ACL(访问控制列表)，这个列表规定了用户的权限，即限定了特定用户对目标节点可以执行的操作。

（3）节点类型

ZooKeeper中的节点有两种，分别为临时节点和永久节点。节点的类型在创建时即被确定，并且不能改变。

　　ZooKeeper的临时节点：该节点的生命周期依赖于创建它们的会话。一旦会话结束，临时节点将被自动删除，当然可以也可以手动删除。另外，需要注意是，ZooKeeper的临时节点不允许拥有子节点。

　　ZooKeeper的永久节点：该节点的生命周期不依赖于会话，并且只有在客户端显示执行删除操作的时候，他们才能被删除。

（4）顺序节点（唯一性的保证）

当创建Znode的时候，用户可以请求在ZooKeeper的路径结尾添加一个递增的计数。这个计数对于此节点的父节点来说是唯一的，它的格式为“%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。

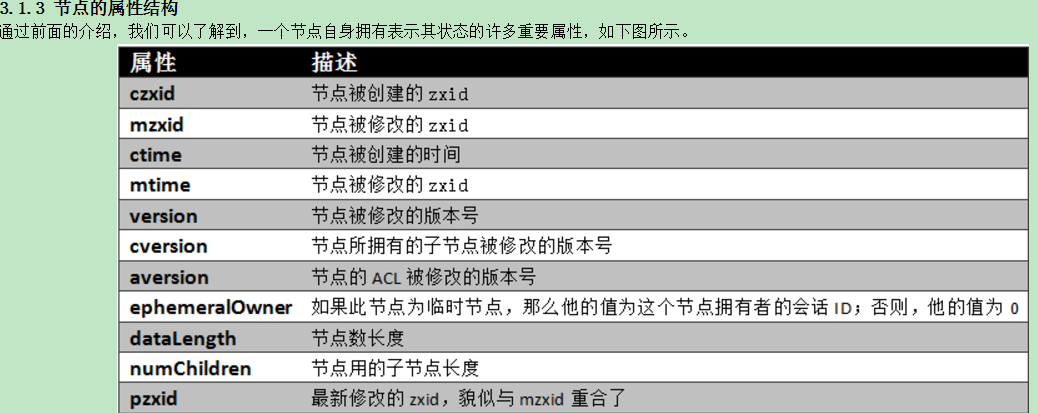
org.apache.zookeeper.CreateMode中定义了四种节点类型，分别对应：

PERSISTENT：永久节点

EPHEMERAL：临时节点

PERSISTENT\_SEQUENTIAL：永久节点、序列化

EPHEMERAL\_SEQUENTIAL：临时节点、序列化



**Zookeeper的伪集群模式搭建**

Zookeeper不但可以在单机上运行单机模式Zookeeper，而且可以在单机模拟集群模式 Zookeeper的运行，也就是将不同节点运行在同一台机器。比如，我们在实验的时候，可以先使用少量数据在集群伪分布模式下进行测试。当测试可行的时候，再将数据移植到集群模式进行真实的数据实验。这样不 但保证了它的可行性，同时大大提高了实验的效率。这种搭建方式，比较简便，成本比较低，适合测试和学习，如果你的手头机器不足，就可以在一台机器上部署了 3个server。

注意: 在一台机器上部署了3个server，需要注意的是在集群为分布式模式下我们使用的每个配置文档模拟一台机器，也就是说单台机器及上运行多个Zookeeper实例。但是，必须保证**每个配置文档的各个端口号**不能冲突，除了clientPort不同之外，dataDir也不同。另外，还要在dataDir所对应的目录中创建myid文件来指定对应的Zookeeper服务器实例。

（1）clientPort端口：如果在1台机器上部署多个server，那么每个实例都要不同的 clientPort，比如 server1是2181,server2是2182，server3是2183，

（2）dataDir和dataLogDir：dataDir和dataLogDir也需要区分下，将数据文件和日志文件分开存放，同时每个server的这两变量所对应的路径都是不同的。

（3）server.X和myid： server.X 这个数字就是对应，data/myid中的数字。在3个server的myid文件中分别写入了0，1，2，那么每个server中的zoo.cfg都配 server.0 server.2,server.3就行了。因为在同一台机器上，后面连着的2个端口，3个server都不要一样，否则端口冲突。

启动集群:

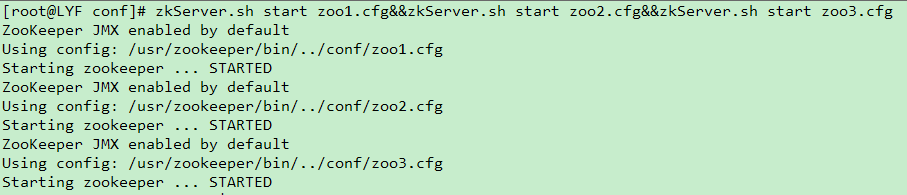
在伪集群模式下，我们只有一台机器，按时要运行三个Zookeeper实例。此时，如果在使用单机模式的启动命令是行不通的。此时，只要通过下面三条命令就能运行前面所配置的Zookeeper服务。如下所示：

zkServer.sh start zoo1.sh //分别使用三个配置启动三个实例

zkServer.sh start zoo2.sh

zkServer.sh start zoo3.sh

结果如图所示;



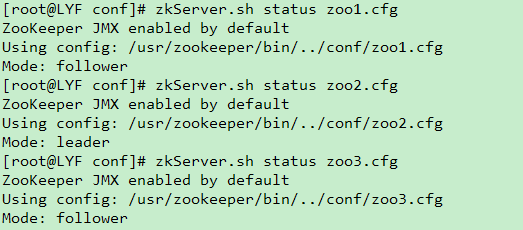
在运行完第一条指令之后，会出现一些错误异常，产生异常信息的原因是由于Zookeeper服务的每个实例都拥有全局配置信息，他们在启动的时候会随时随地的进行Leader选举操作。此时，第一个启动的Zookeeper需要和另外两个Zookeeper实例进行通信。但是，另外两个Zookeeper实例还没有启动起来，因此就产生了这样的异样信息。

此时，可以通过下面三条命令，来查询。

zkServer.sh status zoo1.cfg

zkServer.sh status zoo2.cfg

zkServer.sh status zoo3.cfg



**Zookeeper的集群模式搭建**

为了获得可靠地Zookeeper服务，用户应该在一个服务器机群上部署Zookeeper。只要机群上大多数的Zookeeper服务启动了，那么总的Zookeeper服务将是可用的。集群的配置方式，和前两种类似，同样需要进行环境变量的配置。在每台机器上conf/zoo.cf配置文件的参数设置相同

1. 在zoo.conf文件中属性 dataDir指定的目录下 创建一个 myid 文件.

Server0机器的内容为：0 -------myid中存在且仅存在一个字符 0 等等

Server1机器的内容为：1

Server2机器的内容为：2

1. 在conf目录下删除zoo\_sample.cfg文件，创建一个配置文件zoo.cfg，

# The number of milliseconds of each tick

tickTime=2000

# The number of ticks that the initial

# synchronization phase can take

initLimit=10

# The number of ticks that can pass between

# sending a request and getting an acknowledgement

syncLimit=5

# the directory where the snapshot is stored.

dataDir=/usr/local/zk/data

# the port at which the clients will connect

clientPort=2183

#the location of the log file

dataLogDir=/usr/local/zk/log

server.0=192.168.85.2:2288:3388

server.1=192.168.85.3:2288:3388

server.2=192.168.85.4:2288:3388

3.分别在三台服务器上启动zookeeper

zkServer.sh start

Zookeeper的功能特性是通过Zookeeper配置文件来进行控制管理的(zoo.cfg).这样的设计其实有其自身的原因，通过前面对Zookeeper的配置可以看出，在对Zookeeper集群进行配置的时候，它的配置文档是完全相同的。集群伪分布模式中，有少部分是不同的。这样的配置方式使得在部署Zookeeper服务的时候非常方便。如果服务器使用不同的配置文件，必须确保不同配置文件中的服务器列表相匹配。

在设置Zookeeper配置文档时候，某些参数是可选的，某些是必须的。这些必须参数就构成了Zookeeper配置文档的最低配置要求。另外，若要对Zookeeper进行更详细的配置，可以参考下面的内容。

(1)基本参数:

client：监听客户端连接的端口。

tickTime：基本事件单元，这个时间是作为Zookeeper服务器之间或客户端与服务器之间维持心跳的时间间隔，每隔tickTime时间就会发送一个心跳；最小的session过期时间为2倍tickTime

dataDir：存储内存中数据库快照的位置，如果不设置参数，更新事物的日志将被存储到默认位置。

注意: 应该谨慎的选择日志存放的位置，使用专用的日志存储设备能够大大提高系统的性能，如果将日志存储在比较繁忙的存储设备上，那么将会很大程度上影像系统性能。

(2)可选参数(高级配置)

dataLogdDir 这个操作让管理机器把事务日志写入“dataLogDir”所指定的目录中，而不是“dataDir”所指定的目录。这将允许使用一个专用的日志设备，帮助我们避免日志和快照的竞争 eg:

dataLogDir=/usr/local/zk/logs\_2

maxClientCnxns 这个操作将限制连接到Zookeeper的客户端数量，并限制并发连接的数量，通过IP来区分不同的客户端。此配置选项可以阻止某些类别的Dos攻击。将他设置为零或忽略不进行设置将会取消对并发连接的限制。eg:

maxClientCnxns=1

(3)机群配置参数

initLimit 此配置表示，允许follower(相对于Leaderer言的“客户端”)连接并同步到Leader的初始化连接时间，以tickTime为单位。当初始化连接时间超过该值，则表示连接失败

syncLimit 此配置项表示Leader与Follower之间发送消息时，请求和应答时间长度。如果follower在设置时间内不能与leader通信，那么此follower将会被丢弃。

server.A=B：C：D

A：其中 A 是一个数字，表示这个是服务器的编号；

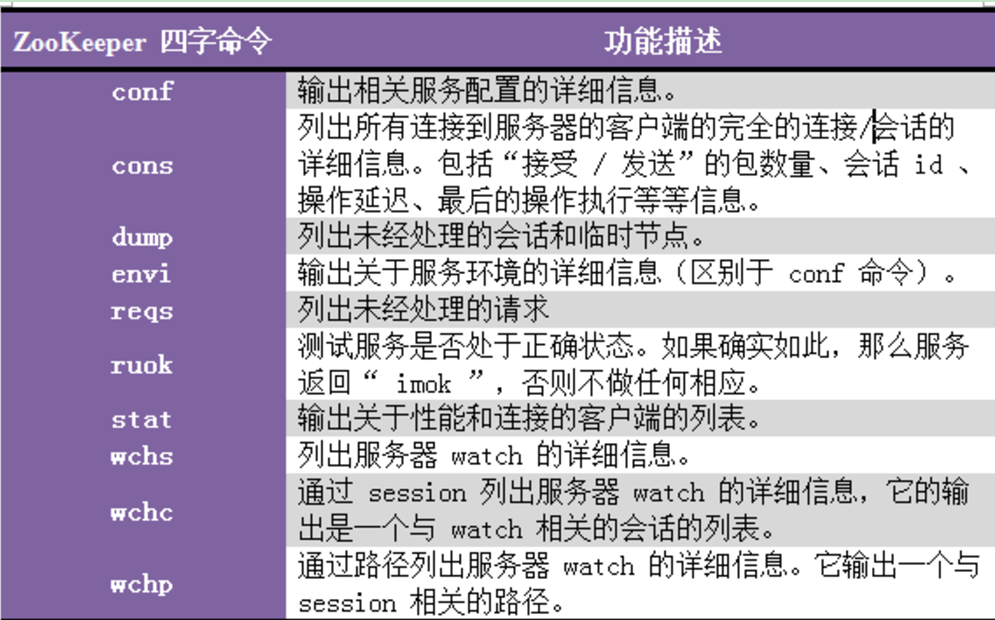
B：是这个服务器的 ip 地址；

C：Leader选举的端口；

D：Zookeeper服务器之间的通信端口。

**ZooKeeper的命令**

Zookeeper支持某些特定的四字命令字母与其的交互。他们大多数是查询命令，用来获取Zookeeper服务的当前状态及相关信息。用户在客户端可以通过telnet或nc向Zookeeper提交相应的命令。常见的命令如下:



Zookeeper的shell操作

Zookeeper命令工具

(1)在启动Zookeeper服务之后，输入以下命令，连接到Zookeeper服务：

zkCli.sh -server localhost:2183

(2)尝试使用ZooKeeper的简单命令

1使用ls命令查看当前Zookeeper中所包含的内容：

ls /

2创建一个新的Znode节点"zk"，以及和它相关字符，执行命令：create /zk myData

create /zk myData

3再次使用ls命令来查看现在Zookeeper的中所包含的内容： ls /

4可以切换到不同的主机上 ls / 查看 发现已经同步到了所有的主机,/zk节点已经被创建

5使用get命令来确认第二步中所创建的Znode是否包含我们创建的字符串，执行命令：get /zk

6接下来通过set命令来对zk所关联的字符串进行设置，执行命令：set /zk hello

7. 再次使用get命令来查看，上次修改的内容，执行命令：get /zk

8. 下面我们将刚才创建的Znode删除，执行命令：delete /zk

这样 一开始创建的/zk节点就被删除了 .

**注意:** zookeeper中的节点的路径全部为绝对路径 是不存在相对路径的

**ZooKeeper中的组成员关系**

**理解ZooKeeper的一种方法就是将其看作一个具有高可用性的文件系统。但这个文件系统中没有文件和目录，而是统一使用“节点”(node)的概念，称为znode。znode既可以作为保存数据的容器（如同文件），也可以作为保存其他znode的容器（如同目录）。所有的znode构成一个层次化的命名空间。一种自然的建立组成员列表的方式就是利用这种层次结构，创建一个以组名为节点名的znode作为父节点，然后以组成员名（服务器名）为节点名来创建作为子节点的znode。**

ZK的 java 客户端的简单监听

可以用一下两个简单的java test 类来实现对zk的理解.

Test1 类

这个类引入了

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.4.6</version>

</dependency> 依赖

package com.asiainfo.lyf;

/\*\*

\* Created by LYF on 2017/9/6.

\*/

import org.apache.zookeeper.\*;

import java.io.IOException;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.Executor;

public class Test {

// 会话超时时间，设置为与系统默认时间一致

private static final int SESSION\_TIMEOUT = 30 \* 1000;

// 创建 ZooKeeper 实例

private ZooKeeper zk;

// 创建 Watcher 实例

private Watcher wh = new Watcher() {

/\*\*

\* Watched事件

\*/

public void process(WatchedEvent event) {

System.out.println("WatchedEvent >>> " + event.toString());

}

};

// 初始化 ZooKeeper 实例

private void createZKInstance() throws IOException {

// 连接到ZK服务，多个可以用逗号分割写

zk = new ZooKeeper("192.168.85.2:2183,192.168.85.3:2183,192.168.85.4:2183", Test.SESSION\_TIMEOUT, this.wh);

}

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

System.out.println("\n1. 创建 ZooKeeper 节点 (znode ： zoo2, 数据： myData2 ，权限： OPEN\_ACL\_UNSAFE ，节点类型： Persistent");

zk.create("/zoo3", "myData2".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

System.out.println("\n2. 查看是否创建成功： ");

System.out.println(new String(zk.getData("/zoo2", this.wh, null)));// 添加Watch

// 前面一行我们添加了对/zoo2节点的监视，所以这里对/zoo2进行修改的时候，会触发Watch事件。

System.out.println("\n3. 修改节点数据 ");

zk.setData("/zoo2", "shanhy20160310".getBytes(), -1);

// 这里再次进行修改，则不会触发Watch事件，这就是我们验证ZK的一个特性“一次性触发”，也就是说设置一次监视，只会对下次操作起一次作用。

System.out.println("\n3-1. 再次修改节点数据 ");

zk.setData("/zoo2", "shanhy20160310-ABCD".getBytes(), -1);

System.out.println("\n4. 查看是否修改成功： ");

System.out.println(new String(zk.getData("/zoo2", false, null)));

System.out.println("\n5. 删除节点 ");

zk.delete("/zoo2", -1);

System.out.println("\n6. 查看节点是否被删除： ");

System.out.println(" 节点状态： [" + zk.exists("/zoo2", false) + "]");

}

private void ZKClose() throws InterruptedException {

zk.close();

}

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

Test dm = new Test();

dm.createZKInstance();

dm.ZKOperations();

dm.ZKClose();

}

}

第二个Test 类

这个类引入了mvn依赖

<dependency>

<groupId>com.github.adyliu</groupId>

<artifactId>zkclient</artifactId>

<version>2.1.1</version>

</dependency>

package com.asiainfo.lyf;

import com.github.zkclient.IZkDataListener;

import com.github.zkclient.ZkClient;

import java.io.IOException;

/\*\*

\* Created by LYF on 2017/9/7.

\*/

/\*\*

\* 从网上看到教程 引入了第三方开源 zkclient 的依赖 测试 客户端监听服务端的改变

\*/

public class Test2 {

public static void main(String args[]) {

ZkClient zkClient=new ZkClient("192.168.85.3:2183");

zkClient.subscribeDataChanges("/abc1", new IZkDataListener() { //注意 该订阅方法不像jedis的订阅一样是个阻塞方法 貌似是异步的 所以 在下方该方法后设置了一个阻塞方法 监听变化

public void handleDataChange(String s, byte[] bytes) throws Exception {

System.out.println("数据改变的提示信息"+s);

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

}

public void handleDataDeleted(String s) throws Exception {

System.out.println("数据删除的提示信息"+s);

}

});

try {

System.in.read();

} catch (IOException e) {

e.printStackTrace();

}

}

}

ZK简介

ZooKeeper本质上是一个分布式的小文件存储系统。原本是Apache Hadoop的一个组件，现在被拆分为一个Hadoop的独立子项目，在HBase（Hadoop的另外一个被拆分出来的子项目，用于分布式环境下的超大数据量的DBMS）中也用到了ZooKeeper集群。

Zookeeper是一个由多个server组成的集群

一个leader，多个follower

每个server保存一份数据副本

全局数据一致

分布式读写

更新请求转发，由leader实施

ZooKeeper运行非常快而且简单。虽然它的目标是构建更加复杂服务（例如同步）的基础，但它提供了一些保证，如下：

1.顺序一致性：来自于客户端的更新，根据发送的先后被顺序实施。

2.唯一的系统映像：尽管客户端连接到不同的服务器，但它们看到的一个唯一（一致性）的系统服务，client无论连接到哪个server，数据视图都是一致的。

3.可靠性：一旦实施了一个更新，就会一直保持那种状态，直到客户端再次更新它，同时数据更新原子性，一次数据更新要么成功，要么失败。

4.及时性：在一个确定的时间内，客户端看到的系统状态是最新的。

可靠性：具有简单、健壮、良好的性能，如果消息m被一台服务器接受，那么它将被所有的服务器接受。

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。

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。

原子性：更新只能成功或者失败，没有中间状态。

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