# 1》zookeeper

## 1.1》什么是zookeeper

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，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进行下一步合理操作。最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户

参照博客：https://www.cnblogs.com/felixzh/p/5869212.html

## 1.2》zookeeper能做什么

* 命名服务
* 配置管理
* 集群管理
* 分布式锁
* 队列管理

## 1.3》命名服务

在zookeeper的文件系统里创建一个目录，即有唯一的path。在我们使用tborg无法确定上游程序的部署机器时即可与下游程序约定好path，通过path即能互相探索发现。

## 1.4》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

|  |
| --- |
|  |

## 1.5》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：它上船了。

新机器加入也是类似，所有机器收到通知：新兄弟目录加入，highcount又有了，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。

|  |
| --- |
|  |

## 1.6》Zookeeper设计目的

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，如果消息被一台服务器接受，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。

## 1.7》Zookeeper工作原理

Zookeeper 的核心是原子广播，这个机制保证了各个Server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数Server完成了和 leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和Server具有相同的系统状态。   
  
 为了保证事务的顺序一致性，zookeeper采用了递增的事务id号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了zxid。实现中zxid是一个64位的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch，标识当前属于那个leader的统治时期。低32位用于递增计数。

## 1.8》zookeeper的四种节点类型znode

* PERSISTENT-持久化目录节点 客户端与zookeeper断开连接后，该节点依旧存在
* PERSISTENT\_SEQUENTIAL-持久化顺序编号目录节点 客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号
* EPHEMERAL-临时目录节点 客户端与zookeeper断开连接后，该节点被删除
* EPHEMERAL\_SEQUENTIAL-临时顺序编号目录节点 客户端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进行顺序编号

# 2》zookeeper安装

## 下载与配置

》http://mirrors.hust.edu.cn/apache/zookeeper 下载一个版本

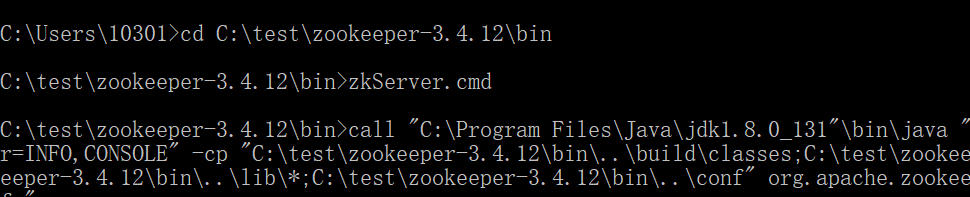
》解压

》找到目录conf 下创建 zoo.cfg 文件，默认就是加载这个文件,然后修改些东西

|  |
| --- |
| #zoo.cfg 的内容  # 心跳检查的时间 2秒  tickTime=2000  # 初始化时 连接到服务器端的间隔次数，总时间10\*2=20秒  initLimit=10  # ZK Leader 和follower 之间通讯的次数，总时间5\*2=10秒  syncLimit=5  # 存储内存中数据库快照的位置，如果不设置参数，更新事务日志将被存储到默认位置。  dataDir=C:\\test\\tmp\\zookeeper  # 错误日志的存放位置  dataLogDir=C:\\test\\log\\zookeeper  # ZK 服务器端的监听端口  clientPort=2181 |

## 启动

》进入到bin目录，启动,执行zkServer.cmd 就启动成功了



# 3》客户端添加节点代码

|  |
| --- |
| **public static final** String ***ROOT*** = **"/root"**; *//根节点* **public static void** main( String[] args ) **throws** Exception {  *//1.创建一个与服务器的连接* ZooKeeper zooKeeper = **new** ZooKeeper(**"localhost:2181"**, 3000, **new** Watcher() {  *//监控所有被触发的事件* @Override  **public void** process(WatchedEvent event) {  System.***out***.println(**"状态State:"** + event.getState());  System.***out***.println(**"类型Type:"** + event.getType());  System.***out***.println(**"Wrapper:"** +event.getWrapper());  System.***out***.println(**"路径Path:"** + event.getPath());  }  });   *//2.创建节点  //2.1 创建根节点,不控制权限，这里需要用持久化节点，不然下面的节点创建容易出错  //zooKeeper.create("/root","/root".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);   //2.2 子节点,持久化顺序编号目录节点  // zooKeeper.create("/root/node1","/root/node1".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT\_SEQUENTIAL);  // zooKeeper.create("/root/node2","/root/node2".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT\_SEQUENTIAL);   //2.3 子节点 临时目录节点 客户端与zookeeper断开连接后，该节点被删除  //zooKeeper.create("/root/node3", "/root/node3".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);   //2.4 子节点 session 过期自动删除，也会加数字的后缀  //zooKeeper.create("/root/node4", "/root/node4".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL\_SEQUENTIAL);  //3.查看节点* List<String> ktvs = zooKeeper.getChildren(***ROOT***, **true**);  System.***out***.println(**"所有子节点:"** + Arrays.*toString*(ktvs.toArray()));   *//4.删除子节点* **for** (String nodeId : ktvs){  zooKeeper.delete(***ROOT*** + **"/"** + nodeId,-1);  }   *//删除根节点* zooKeeper.delete(***ROOT***,-1);  zooKeeper.close(); }  依赖 |
| <**dependency**>  <**groupId**>org.apache.zookeeper</**groupId**>  <**artifactId**>zookeeper</**artifactId**>  <**version**>3.4.6</**version**> </**dependency**>  <**dependency**>  <**groupId**>log4j</**groupId**>  <**artifactId**>log4j</**artifactId**>  <**version**>${log4j.version}</**version**> </**dependency**> <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-api</**artifactId**>  <**version**>${slf4j.version}</**version**> </**dependency**> <**dependency**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  <**version**>${slf4j.version}</**version**> </**dependency**>  <**log4j.version**>1.2.17</**log4j.version**> <**slf4j.version**>1.7.12</**slf4j.version**> |