**big\_day04**

**Zookeeper01**

Zookeeper是一个**分布式(集群环境)**的**协调服务**框架，Zookeeper可以解决分布式环境常见的问题：集群管理、统一命名服务，信息配置管理，分布式锁等等。

zookeeper旨在在分布式应用中，提供可靠的、可扩展的、分布式的、可配置的协调机制来管理整个集群的状态。

**一**.**应用场景:**

**1.集群管理:**

掌控集群服务器的数量

掌控集群服务器的地址

监控集群服务节点的运行状态,是正常运行还是宕机

2.统一命名服务

为集群中的每台服务器命名,便于管理,要求命名服务要具备全局唯一性

3.统一的配置管理

使得集群全部或部分节点的配置信息保持一致

4.实现集群内部的主备切换,从而避免单点故障问题

5.实现数据的订阅和发布

6.分布式环境下的协调通知

7.实现分布式锁服务

在分布环境下,之前所学的同步代码块, 重入锁机制等不能满足,因为是针对一台服务器内部的多个线程而言,所以要实现分布式锁,需要用新的技术和思路来实现

8.在分布式环境下,非常容易出现死锁和活锁

死锁:由于分布式环境,要通过网路通信,而又因为网络并不总是可靠的,所以可能会造成客户端断开连接一致不释放锁,造成死锁

活锁:线程在执行任务中,由于某些条件的产生,导致任务重新执行,再次碰撞,再碰撞,一次循环往复,就形成了活锁,处于活锁的线程都不能枪占到资源,危害是活锁的线程是非阻塞的,会造成CPU空转,白白耗费CPU资源

**2.Zookeeper的由来**

Yahoo:借鉴了谷歌的一篇论文<The Chubby Lock Service For loosely coupled distributed System>.2010年zookeeper的由来于雅虎研究院

**3.zookeeper的重要性**

Hadoop

Hbase

Storm

Kafka

**4.zk指令**

bin]# sh zkServer.sh start/stop/restart

bin]# sh zkServer.shstatus---Standalone Leader Follower Observer

bin]# sh zkCli.sh 进入zk客户端

增:create 默认是创建**普通持久节点**

-e 创建临时节点，当创建此节点的客户端下线时，节点被删除。

-s 创建顺序节点，每次创建节点时，会跟上一个递增的顺序号

-e -s 创建临时顺序节点

删:delete;

递归删除:rmr

改:set

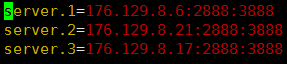
查:ls /;查看指定路径的节点信息

get /:查看指定节点的数据信息

**集群模式搭建**

配置zookeeper-3.4.7/conf/zoo.cfg

IMG_256

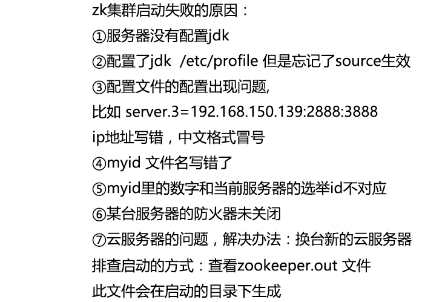


zookeeper-3.4.7/tmp/myid

1 2 3

**远程发送文件到具体虚拟机的具体位置:**

scp -r zookeeper-3.4.7 176.129.8.21:/home/software/



**zookeeper的选举流程:**

阶段一:数据恢复阶段,每台zk服务器会找自己所拥有的最大事务id

阶段二:选举阶段,每台服务器会提交自身的选举协议,协议里包含的内容:

a.最大事务id

b.选举id

c.逻辑时钟值,作用是确保每台zk服务器在同一轮选举中

d,当前服务节点的状态

1)Lonking 正在选举状态

2)Leader

3)Follower

4)Observer观察者状

选举机制//pk原则:

1.先比较最大事务id,谁大谁当Leader,因为事务id越大,意味着拥有的数据越新

2.如果最大事务id比较不出来,就比较选举id,谁大谁当leader

3.选举机制最基本的要求:满足过半性(超过半数)



zookeeper的监听

连接:KeeperState.SyncConneted

创建节点:zk.create(path, data, acl, createMode)

zk.create("/park01", "hello".getBytes(),Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL);

改变节点:zk.setData(path, data, version)

zk.setData("/park01", "hello1".getBytes(), -1);

获取节点:zk.getData(path, watch, stat)

byte[] data = zk.getData("/park01", null, null);

System.out.println(new String(data));

删除节点:zk.delete(path, version);

zk.delete("park01", -1);

获取子节点:zk.getChildren(path, watch)

zk.getChildren("/park02", null);

监听节点数据变化:zk.getData(path, watch, stat) EventType.NodeDataChanged

监听节点删除:zk.exists(path, watch) ventType.NodeDeleted

监听子节点变化:zk.getChildren(path, watch) EventType.NodeChildrenChanged

big\_day05

zookeeper02

**zookeeper知识回顾**

1.zookeeper是一个分布式(集群环境)的协调服务框架

2.zookeeper集群管理,实现思路

每台客户端服务器在启动时连接zk服务器，并注册自己的临时节点，然后通过监听机制，监听节点被删除的事件。

所以zookeeper的应用范围很广泛，只要是集群，都可以使用zookeeper。

3.zookeeper可以做统一的命名服务,可以利用zk路径唯一性的来实现。

4.zookeeper实现数据的订阅和发布,订阅方监听数据发布方节点数据产生变化事件。

5.zookeeper实现集群统一的配置信息管理

6.zookeeper实现协调服务通知

成绩排名,监听,--协调服务,分布式屏障(barrir)

7.zookeeper实现分布式锁

**通过顺序节点**的抢注,判断哪台服务器是最先抢到的,从而分配资源,此外,为了避免死锁,可以引入**临时特性**.

8.总结:

zookeeper实现各种分布式服务的手段:各类型节点+监听机制

9.zookeeper单击模式安装

复制.重命名为zoo.cfg,启动zk服务器

10.zk单机模式用于练习或测试环境,不能用于生产环境，因为单点故障问题。

11.zk集群的搭建,配置zoo.cfg:

1.data.dir 2.server.选举id=ip:原子广播端口:选举端口 3.创建myid文件. 4.启动集群

12.操作zk时,都是以znode路径来操作的,每个znode都可以储存数据,数据会在zk服务器的内存里存一份,供用户快速访问.基于这个特点.不能用zk存储海量数据,一是zk的应用就不是存储海量数据的(是做协调服务的),而是基于内存,不能存储海量数据,所以,zk存储的数据都是元数据信息,配置信息,服务器的节点数据.

13.选举机制:目的是通过一定的规则（算法）选出zk集群的leader。

pk原则: a,事务id, b,选举id c,过半性原则

14.过半性的体现:

a.选举过程的过半同意

c.zk服务集群要保证正常工作,必须要满足过半性,成为**过半存活.**

**所以,一般在工作中,zk集群数量一般都是奇数个.**

**zookeeper最重要的特点:分布式环境的数据一致性,目的是确保客户端??**

**2PC算法:**

为了解决分布式数据一致性问题，在长期的探索研究过程中，涌现出了一大批经典的一致性协议和算法，其中最著名的就是二阶段提交协议、三阶段提交协议和Paxos算法

**Two-Phase Commit二阶段提交**

是计算机网络尤其是在数据库领域内，为了使基于分布式系统架构下的所有节点在进行事务处理过程中能够保持原子性和一致性而设计的一种算法。通常，二阶段提交协议也被认为是一种一致性协议，用来保证分布式系统数据的一致性。目前，绝大部分的关系型数据库都是采用二阶段提交协议来完成分布式事务处理的，利用该协议能够非常方便地完成所有分布式事务参与者的协调，统一决定事务的提交或回滚，从而能够有效地保证分布式数据一致性，因此二阶段提交协议被广泛地应用在许多分布式系统中。

提交过程:

阶段一:提交事务请求+执行事务

1.事务询问 2.执行事务 3.各参与者向协调者反馈事务询问的响应-投票阶段

阶段二:事务提交

可能一:执行事务提交

1.发送提交请求 2.事务提交 3.反馈事务提交结果 4.完成事务

可能二:中断事务

1.发送回滚请求 2.事务回滚 3.反馈事务回滚结果 4.中断事务

优点:原理简单,实现方便

缺点:同步阻塞,单点问题,太过保守

**同步阻塞:**二阶段提交协议存在的最明显也是最大的一个问题就是同步阻塞，这会极大地限制分布式系统的性能。在二阶段提交的执行过程中，所有参与该事务操作的逻辑都处于阻塞状态，也就是说，各个参与者在等待其他参与者响应的过程中，将无法进行其他任何操作。所以性能较低

**单点问题**:在上面的讲解过程中，相信读者可以看出，协调者的角色在整个二阶段提交协议中起到了非常重要的作用。一旦协调者出现问题，那么整个二阶段提交流程将无法运转

**太过保守:**如果在协调者指示参与者进行事务提交询问的过程中，参与者出现故障而导致协调者始终无法获取到所有参与者的响应信息的话，这时协调者只能依靠其自身的超时机制来判断是否需要中断事务，这样的策略显得比较保守。换句话说，二阶段提交协议没有设计较为完善的容错机制，任意一个节点的失败都会导致整个事务的失败。

**ZAB协议**

协议是为分布式协调服务zookeeper专门设计的一种支持崩溃恢复的原子广播协议,它是一种特别为ZooKeeper设计的崩溃可恢复的原子消息广播算法。这个算法是一种类2PC算法，在2PC基础上做的改进.

ZAB协议包括两种基本的模式

**1)消息原子广播（保证数据一致性）**(略)

**2)崩溃恢复(解决2PC算法的单点问题)**(略)

崩溃恢复理解:

当leader挂掉后,会重新进行选举

清空tmp

在foller中,创建节点,事务id,递增,挂掉leader,触发新一轮的选举机制

ZAB协议算法:改进:

1.引入过半性 2.支持角色轮换

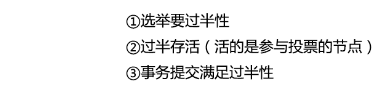
推荐书籍:

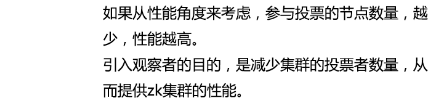
**Zookeeper观察者**

**不参与投票,只监听投票的结果,**

观察者像追随者一样转发这些请求到领导者，而他们只是简单的等待监听投票的结果。

因此他们可以故障，或者从集群断开连接，而不影响 Zookeeper服务的可用性。





观察者的使用:

1.zoo.cfg中加上peerType=observer

2.集群中每个zoo.cfg中的的server的观察者后面加上:observer

zookeeper配置详解:见笔记:(略)

zookeeper集群指令:

nc ：netcat 是一个小型的网络通信工具，可以发起TCP请求

**echo stat|nc 127.0.0.1 2181**---来查看哪个节点被选择作为follower或者leader

**echo ruok|nc 127.0.0.1 2181**---测试是否启动了该Server，若回复imok表示已经启动。

echo dump| nc 127.0.0.1 2181 ---列出未经处理的会话和临时节点

**echo conf | nc 127.0.0.1 2181**---输出相关服务配置的详细信息。

clientPort=2181

dataDir=/home/software/zookeeper-3.4.7/tmp/version-2

dataLogDir=/home/software/zookeeper-3.4.7/tmp/version-2

tickTime=2000

maxClientCnxns=60

minSessionTimeout=4000

maxSessionTimeout=40000

serverId=1

initLimit=10

syncLimit=5

electionAlg=3

electionPort=3888

quorumPort=2888

peerType=1

echo kill | nc 127.0.0.1 2181 ---关掉server

**zookeeper特性总结**

1.数据一致性:client不论连接到哪个Zookeeper，展示给它都是同一个视图，即查询的数据都是一样的。

2.原子性:要么都更新成功，要么都不更新。

3.可靠性:一旦zk服务端成功的应用了一个事务，并完成对客户端的响应，那么该事务所引起的服务端状态变更将会一直保留下来，除非有另一个事务又对其进行了改变。

4.实时性:Zookeeper保证客户端将在非常短的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，或者指定监听事件的变化信息。（前提条件是：网络状况良好）

5.顺序性: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。底层是通过递增的事务id（zxid）来实现的。

6.过半性

课外阅读:

阿里-otter(详见笔记)

阿里—Metamorphosis

阿里—Dubbo

阿里—Canal

Paxos算法