# Zookeeper

## Zookeeper的简单理解

### （★★）写出你对 zookeeper 的理解

* 1. 提示：大部分分布式应用需要一个主控、协调器或控制器来管理物理分布的子
  2. 进程（如资源、任务分配等）。 目前，大部分应用需要开发私有的协调程序，缺乏一个通用的机制协调程序的反复编写浪费，且难以形成通用、伸缩性好的协调器。ZooKeeper：提供通用的分布式锁服务，用以协调分布式应用

### （★★）Zookeeper的基本概念和使用方式，ZooKeeper 被问到过其如何维护高可用（如果某个节点挂掉了它的处理机制）？

用一句话对其进行定义就是:它是一套高吞吐的分布式协调系统

如何维护高可用：应用集群利用Zookeeper来实现高可用性的原理就是将多个应用服务的入口（IP/PORT）注册到Zookeeper服务器，应用服务的调用者通过监控Zookeeper中保存的各节点状态来选择可以访问的节点。但节点宕机或不可用时，会被从可用的节点信息中移除，所以调用者可以通过监控到此节点不可用后，切换/重新连接到可用的节点上，从而实现H/A。

## Zookeeper的作用

### （★）Zookeeper作用

通过选举，保证任何时候，集群中只有一个master，Master与RegionServers启动时会向ZooKeeper注册

存贮所有Region的寻址入口

实时监控Regionserver的上线和下线信息。并实时通知给Master

存储HBase的schema和table元数据

默认情况下，HBase管理ZooKeeper实例，比如，启动或者停止ZooKeeper

Zookeeper的引入使得Master不再是单点故障

## Zookeeper的高可用

### （★★）Zookeeper实现高可用的原理

应用集群利用Zookeeper来实现高可用性的原理就是将多个应用服务的入口（IP/PORT）注册到Zookeeper服务器，应用服务的调用者通过监控Zookeeper中保存的各节点状态来选择可以访问的节点。但节点宕机或不可用时，会被从可用的节点信息中移除，所以调用者可以通过监控到此节点不可用后，切换/重新连接到可用的节点上，从而实现H/A。

## Zookeeper实现分布式锁

### （★★）Zookeeper实现分布式锁用哪个jar包

这个网上各种答案都有，我只能列举几种

① org.apache.curator.curator-recipes

https://blog.csdn.net/qiangcuo6087/article/details/79067136

② 通过实现Watch接口，实现process(WatchedEvent event)方法来实施监控

https://www.cnblogs.com/liuyang0/p/6800538.htmlZookeeper的二次开发

### （★★★）Zookeeper分布式锁实现

a、原理

ZooKeeper核心是一个精简的文件系统，它提供了一些简单的文件操作以及附加的功能 ，它的数据结构原型是一棵znode树（类似Linux的文件系统），并且它们是一些已经被构建好的块，可以用来构建大型的协作数据结构和协议 。

每个锁都需要一个路径来指定(如:/geffzhang/lock)

1.根据指定的路径, 查找zookeeper集群下的这个节点是否存在.(说明已经有锁了)

2. 如果存在, 根据查询者的一些特征数据(如ip地址/hostname), 当前的锁是不是查询者的

3. 如果不是查询者的锁, 则返回null, 说明创建锁失败

4. 如果是查询者的锁, 则把这个锁返回给查询者

5. 如果这个节点不存在, 说明当前没有锁, 那么创建一个临时节点, 并将查询者的特征信息写入这个节点的数据中, 然后返回这个锁.

据以上5部, 一个分布式的锁就可以创建了.

创建的锁有三种状态:

1. 创建失败(null), 说明该锁被其他查询者使用了.’

2. 创建成功, 但当前没有锁住(unlocked), 可以使用

3. 创建成功, 但当前已经锁住(locked)了, 不能继续加锁.

b、代码样例

public class ZooKeeperLock implements Lock, Watcher {

private Logger logger = LoggerFactory.getLogger(getClass());

private static final String SPLITSTR = "\_lock\_";

private static final int SESSION\_TIMEOUT = 60000;//等锁的毫秒数

private static final byte[] data = new byte[0];

private ZooKeeper zk = null;

private String root = "/locks";//根

private String lockName;//竞争资源的标志

private String waitNode;//等待前一个锁

private String myZnode;//当前锁

private CountDownLatch latch;//计数器

/\*\*

\* 创建分布式锁,使用前请确认config配置的zookeeper服务可用

\* @param server 127.0.0.1:2181

\* @param lockName 竞争资源标志,lockName中不能包含单词lock

\*/

public ZooKeeperLock(String server, String lockName){

this.lockName = lockName;

// 创建一个与服务器的连接

try {

zk = initZk(server);

Stat stat = zk.exists(root, false);

if(stat == null){

// 创建根节点

zk.create(root,data, ZooDefs.Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT);

}

} catch (Exception e) {

throw new LockException(e);

}

}

/\*\*

\* zookeeper节点的监视器

\*/

@Override

public void process(WatchedEvent event) {

if(this.latch != null) {

this.latch.countDown();

}

}

@Override

public void lock() {

try {

if(!tryLock()){

boolean locked = waitForLock(waitNode, SESSION\_TIMEOUT, TimeUnit.MILLISECONDS);//等待锁

if(!locked){

logger.error("can not lock...");

}

}

} catch (Exception e) {

throw new LockException(e);

}

}

public boolean tryLock() {

try {

if(lockName.contains(SPLITSTR)){

throw new LockException("lockName can not contains \\u000B");

}

//创建临时子节点

myZnode = zk.create(root + "/" + lockName + SPLITSTR, data, ZooDefs.Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);

//取出所有子节点

List<String> subNodes = zk.getChildren(root, false);

//取出所有lockName的锁

List<String> lockObjNodes = new ArrayList<>();

for (String node : subNodes) {

String \_node = node.split(SPLITSTR)[0];

if(\_node.equals(lockName)){

lockObjNodes.add(node);

}

}

Collections.sort(lockObjNodes);

if(myZnode.equals(root+"/"+lockObjNodes.get(0))){

//如果是最小的节点,则表示取得锁

return true;

}

//如果不是最小的节点，找到比自己小1的节点

String subMyZnode = myZnode.substring(myZnode.lastIndexOf("/") + 1);

waitNode = lockObjNodes.get(Collections.binarySearch(lockObjNodes, subMyZnode) - 1);

} catch (Exception e) {

throw new LockException(e);

}

return false;

}

@Override

public boolean tryLock(long time, TimeUnit unit) {

try {

return tryLock() || waitForLock(waitNode, time, unit);

} catch (Exception e) {

throw new LockException(e);

}

}

private boolean waitForLock(String lower, long waitTime, TimeUnit unit) throws InterruptedException, KeeperException {

Stat stat = zk.exists(root + "/" + lower, true);

//判断比自己小一个数的节点是否存在,如果不存在则无需等待锁,同时注册监听

if(stat != null){

this.latch = new CountDownLatch(1);

this.latch.await(waitTime, unit);

this.latch = null;

}

return true;

}

@Override

public void unlock() {

try {

zk.delete(myZnode,-1);

myZnode = null;

} catch (Exception e) {

throw new LockException(e);

}

}

private synchronized ZooKeeper initZk(String server) {

try {

if(zk == null){

zk = new ZooKeeper(server, SESSION\_TIMEOUT, this);

}

} catch (IOException e) {

throw new LockException("zk init connect fail" + e.getMessage());

}

return zk;

}

@Override

public void lockInterruptibly() throws InterruptedException {

this.lock();

}

@Override

public Condition newCondition() {

return null;

}

private class LockException extends RuntimeException {

private static final long serialVersionUID = 1L;

private LockException(String e){

super(e);

}

private LockException(Exception e){

super(e);

}

}

}

c、代码分析

      lock:根据根创建锁节点，然后获取当前已经存在锁的节点，如果第一个节点为自己创建，说明没有锁，不是自己加锁，则给自己创建节点的上一个节点加监听，线程阻塞至上一个节点释放，并通知我，或者等待超时。

      unlock:删除自己创建的节点，zookeeper会自动通知加在节点上的监听。

## 与Zookeeper类似的框架

### （★）下面与 Zookeeper 类似的框架是？ D

A Protobuf B Java C Kafka D Chubby

## Zookeeper的二次开发

### 7.Zookeeper的二次开发

///TODO