**ZooKeeper 典型的应用场景**

Zookeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式，关于 Zookeeper 的详细架构等内部细节可以阅读 Zookeeper 的源码

下面详细介绍这些典型的应用场景，也就是 Zookeeper 到底能帮我们解决那些问题？

**统一命名服务（Name Service）**

分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

**配置管理（Configuration Management）**

配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。

像这样的配置信息完全可以交给 Zookeeper 来管理，将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。

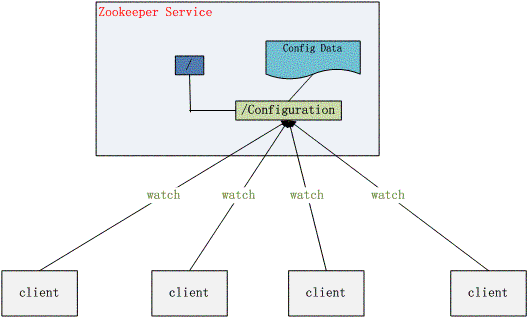


图 1. 配置管理结构图

**集群管理（Group Membership）**

Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。

Zookeeper 不仅能够帮你维护当前的集群中机器的服务状态，而且能够帮你选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能 Leader Election。

它们的实现方式都是在 Zookeeper 上创建一个 EPHEMERAL 类型的目录节点，然后每个 Server 在它们创建目录节点的父目录节点上调用 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) 方法并设置 watch 为 true，由于是 EPHEMERAL 目录节点，当创建它的 Server 死去，这个目录节点也随之被删除，所以 Children 将会变化，这时 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)上的 Watch 将会被调用，所以其它 Server 就知道已经有某台 Server 死去了。新增 Server 也是同样的原理。

Zookeeper 如何实现 Leader Election，也就是选出一个 Master Server。和前面的一样每台 Server 创建一个 EPHEMERAL 目录节点，不同的是它还是一个 SEQUENTIAL 目录节点，所以它是个 EPHEMERAL\_SEQUENTIAL 目录节点。之所以它是 EPHEMERAL\_SEQUENTIAL 目录节点，是因为我们可以给每台 Server 编号，我们可以选择当前是最小编号的 Server 为 Master，假如这个最小编号的 Server 死去，由于是 EPHEMERAL 节点，死去的 Server 对应的节点也被删除，所以当前的节点列表中又出现一个最小编号的节点，我们就选择这个节点为当前 Master。这样就实现了动态选择 Master，避免了传统意义上单 Master 容易出现单点故障的问题。

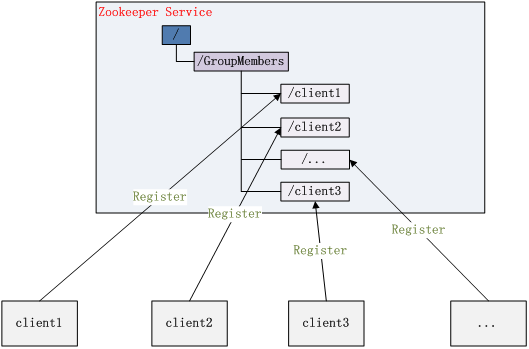


图 2. 集群管

这部分的示例代码如下，

[复制代码](javascript:void(0);)

void findLeader() throws InterruptedException {

byte[] leader = null;

try {

leader = zk.getData(root + "/leader", true, null);

} catch (Exception e) {

logger.error(e);

}

if (leader != null) {

following();

} else {

String newLeader = null;

try {

byte[] localhost = InetAddress.getLocalHost().getAddress();

newLeader = zk.create(root + "/leader", localhost,

ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);

} catch (Exception e) {

logger.error(e);

}

if (newLeader != null) {

leading();

} else {

mutex.wait();

}

}

}

[复制代码](javascript:void(0);)

共享锁（Locks）

共享锁在同一个进程中很容易实现，但是在跨进程或者在不同 Server 之间就不好实现了。Zookeeper 却很容易实现这个功能，实现方式也是需要获得锁的 Server 创建一个 EPHEMERAL\_SEQUENTIAL 目录节点，然后调用 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)方法获取当前的目录节点列表中最小的目录节点是不是就是自己创建的目录节点，如果正是自己创建的，那么它就获得了这个锁，如果不是那么它就调用 [exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) 方法并监控 Zookeeper 上目录节点列表的变化，一直到自己创建的节点是列表中最小编号的目录节点，从而获得锁，释放锁很简单，只要删除前面它自己所创建的目录节点就行了。

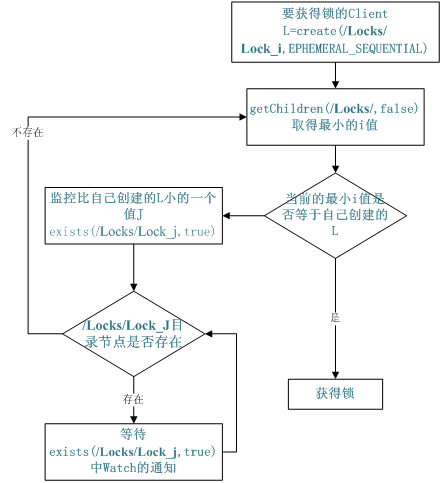


图 3. Zookeeper 实现 Locks 的流程图

同步锁的实现代码如下，同步锁的关键代码：

[复制代码](javascript:void(0);)

void getLock() throws KeeperException, InterruptedException{

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

String[] nodes = list.toArray(new String[list.size()]);

Arrays.sort(nodes);

if(myZnode.equals(root+"/"+nodes[0])){

doAction();

}

else{

waitForLock(nodes[0]);

}

}

void waitForLock(String lower) throws InterruptedException, KeeperException {

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

if(stat != null){

mutex.wait();

}

else{

getLock();

}

}

[复制代码](javascript:void(0);)

**队列管理**

Zookeeper 可以处理两种类型的队列：

1. 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，这种是同步队列。
2. 队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型。

同步队列用 Zookeeper 实现的实现思路如下：

创建一个父目录 /synchronizing，每个成员都监控标志（Set Watch）位目录 /synchronizing/start 是否存在，然后每个成员都加入这个队列，加入队列的方式就是创建 /synchronizing/member\_i 的临时目录节点，然后每个成员获取 / synchronizing 目录的所有目录节点，也就是 member\_i。判断 i 的值是否已经是成员的个数，如果小于成员个数等待 /synchronizing/start 的出现，如果已经相等就创建 /synchronizing/start。

用下面的流程图更容易理解：

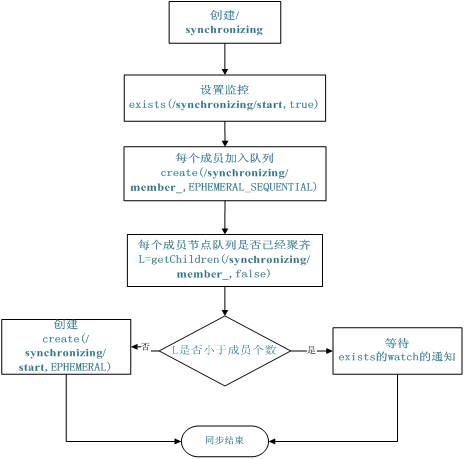


图 4. 同步队列流程图

同步队列的关键代码如下：

[复制代码](javascript:void(0);)

void addQueue() throws KeeperException, InterruptedException{

zk.exists(root + "/start",true);

zk.create(root + "/" + name, new byte[0], Ids.OPEN\_ACL\_UNSAFE,

CreateMode.EPHEMERAL\_SEQUENTIAL);

synchronized (mutex) {

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

if (list.size() < size) {

mutex.wait();

} else {

zk.create(root + "/start", new byte[0], Ids.OPEN\_ACL\_UNSAFE,

CreateMode.PERSISTENT);

}

}

}

[复制代码](javascript:void(0);)

当队列没满是进入 wait()，然后会一直等待 Watch 的通知，Watch 的代码如下：

[复制代码](javascript:void(0);)

public void process(WatchedEvent event) {

if(event.getPath().equals(root + "/start") &&

event.getType() == Event.EventType.NodeCreated){

System.out.println("得到通知");

super.process(event);

doAction();

}

}

[复制代码](javascript:void(0);)

FIFO 队列用 Zookeeper 实现思路如下：

实现的思路也非常简单，就是在特定的目录下创建 SEQUENTIAL 类型的子目录 /queue\_i，这样就能保证所有成员加入队列时都是有编号的，出队列时通过 getChildren( ) 方法可以返回当前所有的队列中的元素，然后消费其中最小的一个，这样就能保证 FIFO。

下面是生产者和消费者这种队列形式的示例代码，

生产者代码：

[复制代码](javascript:void(0);)

boolean produce(int i) throws KeeperException, InterruptedException{

ByteBuffer b = ByteBuffer.allocate(4);

byte[] value;

b.putInt(i);

value = b.array();

zk.create(root + "/element", value, ZooDefs.Ids.OPEN\_ACL\_UNSAFE,

CreateMode.PERSISTENT\_SEQUENTIAL);

return true;

}

[复制代码](javascript:void(0);)

消费者代码：

[复制代码](javascript:void(0);)

int consume() throws KeeperException, InterruptedException{

int retvalue = -1;

Stat stat = null;

while (true) {

synchronized (mutex) {

List<String> list = zk.getChildren(root, true);

if (list.size() == 0) {

mutex.wait();

} else {

Integer min = new Integer(list.get(0).substring(7));

for(String s : list){

Integer tempValue = new Integer(s.substring(7));

if(tempValue < min) min = tempValue;

}

byte[] b = zk.getData(root + "/element" + min,false, stat);

zk.delete(root + "/element" + min, 0);

ByteBuffer buffer = ByteBuffer.wrap(b);

retvalue = buffer.getInt();

return retvalue;

}

}

}

}

[复制代码](javascript:void(0);)

总结

Zookeeper 是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，如它管理 Hadoop 集群中的 NameNode，还有 Hbase 中 Master Election、Server 之间状态同步等。

本文介绍的 Zookeeper 的基本知识，以及介绍了几个典型的应用场景。这些都是 Zookeeper 的基本功能，最重要的是 Zoopkeeper 提供了一套很好的分布式集群管理的机制，就是它这种基于层次型的目录树的数据结构，并对树中的节点进行有效管理，从而可以设计出多种多样的分布式的数据管理模型，而不仅仅局限于上面提到的几个常用应用场景。