ZooKeeper与分布式锁

技术架构部　　吴晋

目录

01 ZooKeeper简介

02 ZooKeeper的基础概念

03 ZooKeeper的使用

04 ZooKeeper技术解析

05 ZooKeeper应用场景

01

章节 PART

ZooKeeper简介

Chapter One

ZooKeeper是什么？

ZooKeeper 是由 Hadoop的子项目发展而来，为分布式应用提供高效且可靠的协调服务。被广泛应用于配置维护、命名服务、集群管理等方面，在大多数场景中，它被作为一个协调分布式环境中各子系统之间共享状态数据的基础组件。

为什么使用ZooKeeper？

集群的维护比单机模式复杂很多，需要花很多工作来处理资源竞态和状态同步的问题。例如客户端程序实时感知服务机状态、应用与应用之间的公共资源的互斥访问、同一配置在多台机器上的同步，而ZooKeeper将这些复杂的问题包装成了简单易用、性能高效、功能稳定的接口提供给用户，提高用户的开发效率、减低开发难度。

ZooKeeper特性

原子性

任何事务的结果在整个集群中所有机器上的应用情况是一致的，要么都应用，要么都不应用。

顺序一致性

从同一个客户端发起的请求，会严格的按照其发起的顺序被应用到ZooKeeper中。

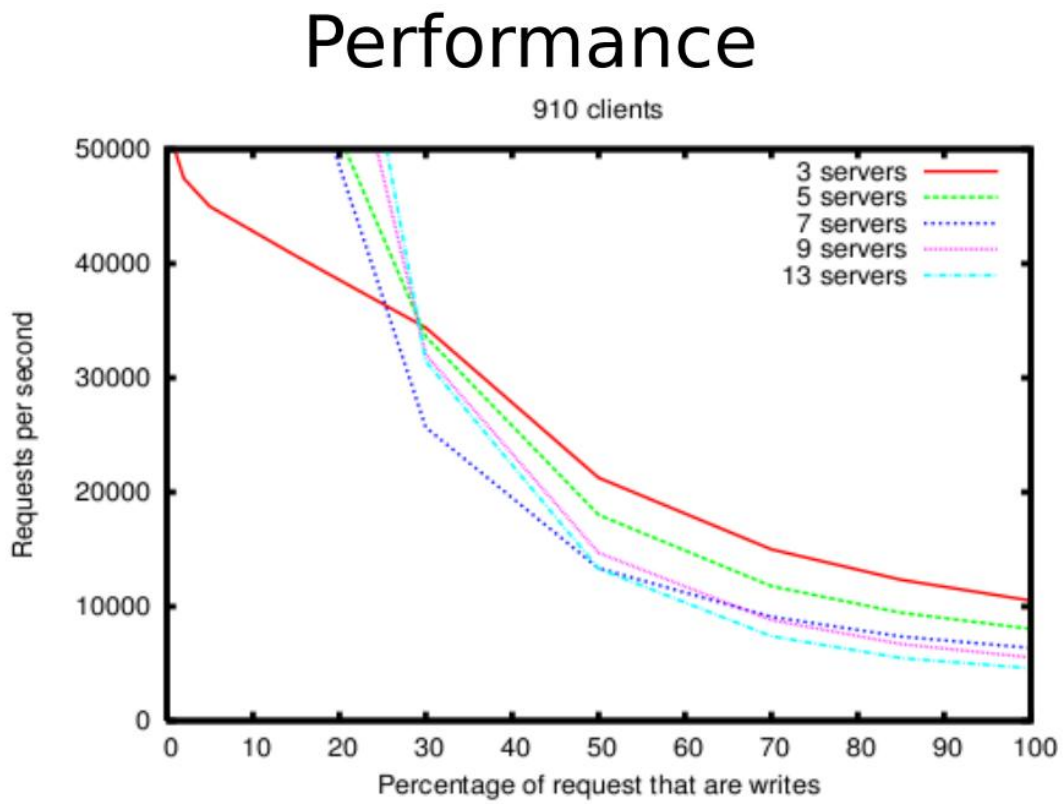
单一视图

客户端连接集群中任一台服务器，看到的服务端数据模型都是一致的。

可靠性

一旦服务器应用了某个事务，那么这个事务引起的服务端状态变更就会一直保存下去。

ZooKeeper的性能



集群服务器数	1%写操作	100%写操作
13	265115	4592
9	195178	5550
7	147810	6371
5	75308	8048
3	49827	10519

02

章节 PART

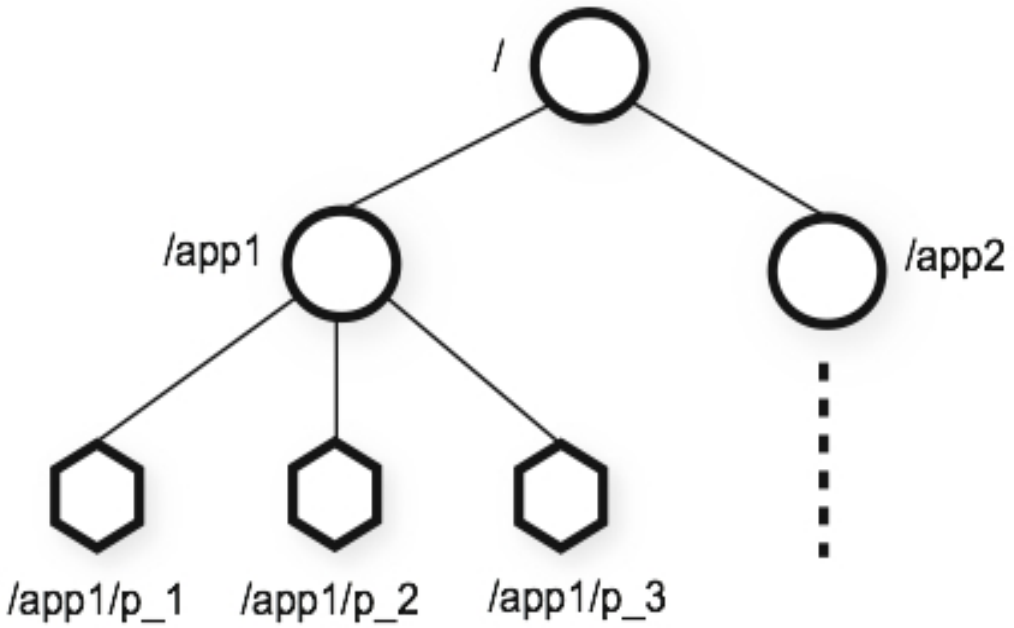
ZooKeeper的基础概念

Chapter Two

ZooKeeper的基础概念

1. ZooKeeper的数据模型
2. ZNode（数据节点）
3. ZooKeeper集群架构
4. Session（会话）
5. Watcher（事件监听器）

ZooKeeper的数据模型



- 树形分层结构
- 树上每个节点被称为ZNode
- 每个ZNode拥有自己唯一的路径，用斜线分割，类似于文件系统
- ZNode中保存有数据（byte[]）和属性（stat)
- 数据大小不超过1MB

数据节点（ZNode）

在ZooKeeper当中，节点类型可以分为持久节点、临时节点，以及顺序节点，组合使用可以产生以下四种节点类型：

- 持久节点（PERSISTENT）
- 持久顺序节点（PERSISTENT_SEQUENTIAL）
- 临时节点（EPHEMERAL）
- 临时顺序节点（EPHEMERAL_SEQUENTIAL）

持久节点会一直存在于ZooKeeper当中，除非用户主动删除它。

临时节点的生命周期与会话绑定，临时节点不能有子节点，只能作为叶子节点。

顺序节点在创建时会在末尾添加一个自增的数字后缀，该数字由其父节点维护。

数据节点（ZNode）

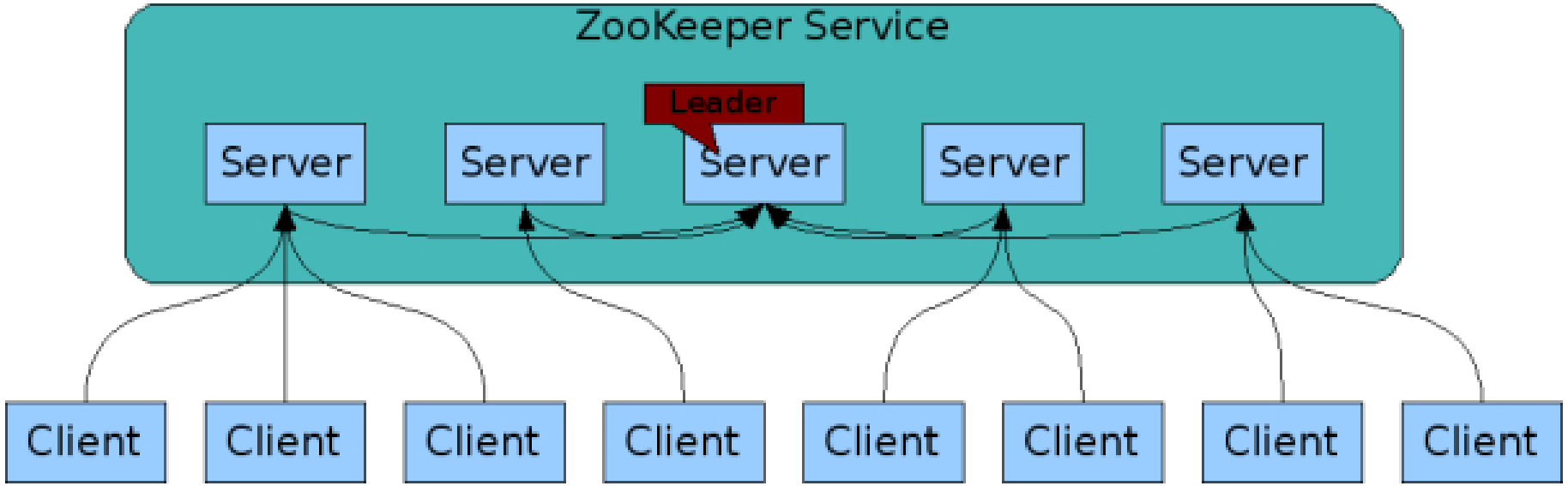
```
[zk: localhost:2181(CONNECTED) 16] create /test/node "test node"
Created /test/node
[zk: localhost:2181(CONNECTED) 17] get /test/node
test node
cZxid = 0x500000016
ctime = Fri Apr 19 17:34:02 CST 2019
mZxid = 0x500000016
mtime = Fri Apr 19 17:34:02 CST 2019
pZxid = 0x500000016
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 9
numChildren = 0
[zk: localhost:2181(CONNECTED) 18]
```

ZNode属性

cZxid	创建节点时的事务ID
ctime	创建节点时的时间
mZxid	最后修改节点时的事务ID
mtime	最后修改节点时的时间
pZxid	表示该节点的子节点列表最后一次修改的事务ID，添加子节点或删除子节点就会影响子节点列表，但是修改子节点的数据内容则不影响该ID
cversion	子节点版本号，子节点每次修改版本号加1
dataVersion	数据版本号，数据每次修改该版本号加1
aclVersion	权限版本号，权限每次修改该版本号加1
ephemeralOwner	临时节点拥有者id，如果是持久节点，该值为0
dataLength	该节点的数据长度
numChildren	该节点拥有子节点的数量

ZooKeeper集群架构

- 集群采用ZAB协议(多数派协议)进行通信以保证一致性，所以最少需要3个节点，推荐奇数节点
- 客户端随机连接集群中任何一台server
- 集群根据内部算法选出Leader，写操作会被转发给Leader进行，并向Learner广播。



集群角色

- Leader

  事务请求的唯一调度和处理者，保证集群事务处理的顺序性。

  集群内部各服务器调度者。

- Follower

  处理非事务请求，转发事务请求给Leader

  参与事务请求投票

  参与Leader选举投票

- Observer

  只提供非事务服务，不参与任何投票

Session

会话是ZooKeeper最重要的概念之一，客户端与服务端之间的任何交互都与会话息息相关，其中包括临时节点的生命周期、客户端请求的顺序执行以及Watcher通知机制等。

ZooKeeper服务器启动是会根据SID和时间戳生成全局唯一的SessionID，被称为基准SessionID，以后针对每个客户端在“基准SessionID”的基础上逐个递增产生。

Watcher（事件监听器）

ZooKeeper允许在ZNode上注册一些Watcher，并且在一些特定事件触发的时候，ZooKeeper服务器会调用Watcher并通知到感兴趣的客户端上。

- 非事务操作可以添加Watcher
- 事务操作会触发Watcher

Watcher是一次性的触发器，触发后会被删除，需要按需重复添加。

Watcher的触发条件

触发条件	KeeperState	EventType	说明
客户端与服务端成功建立连接	SyncConnected（3）	None（-1）	此时客户端和服务器处于连接状态
Watcher监听的对应数据节点被创建		NodeCreated（1）	
Watcher监听的对应数据节点被删除		NodeDeleted（2）	
Watcher监听的对应数据节点的数据内容发生变更		NodeDataChanged（3）	
Wather监听的对应数据节点的子节点列表发生变更		NodeChildChanged（4）	
客户端与ZooKeeper服务器断开连接	Disconnected（0）	None（-1）	此时客户端和服务器处于断开连接状态
会话超时	Expired(-112)	None（-1）	此时客户端会话失效
通常有两种情况，1：使用错误的schema进行权限检查 2：SASL权限检查失败	AuthFailed(4)	None（-1）	通常同时也会收到AuthFailedException异常

Watcher（事件监听器）

ZooKeeper允许在ZNode上注册一些Watcher，并且在一些特定事件触发的时候，ZooKeeper服务器会调用Watcher并通知到感兴趣的客户端上。

- 非事务操作可以添加Watcher
- 事务操作会触发Watcher

Watcher是一次性的触发器，触发后会被删除，需要按需重复添加。

03

章节 PART

ZooKeeper的使用

Chapter Three

ZooKeeper的使用

- 下载地址：

  https://zookeeper.apache.org/releases.html#download

- 启动模式：
  - 单机模式
  - 集群模式

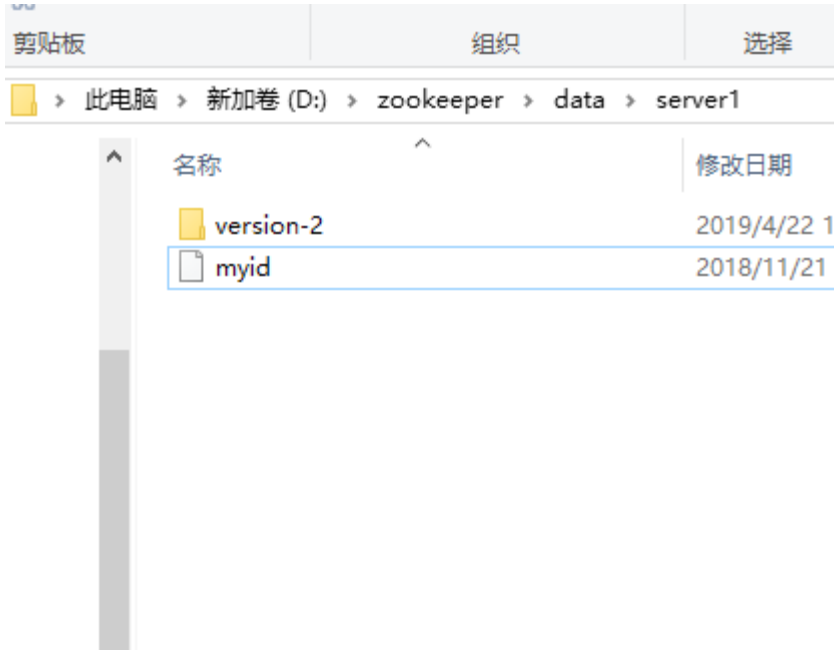
ZooKeeper配置

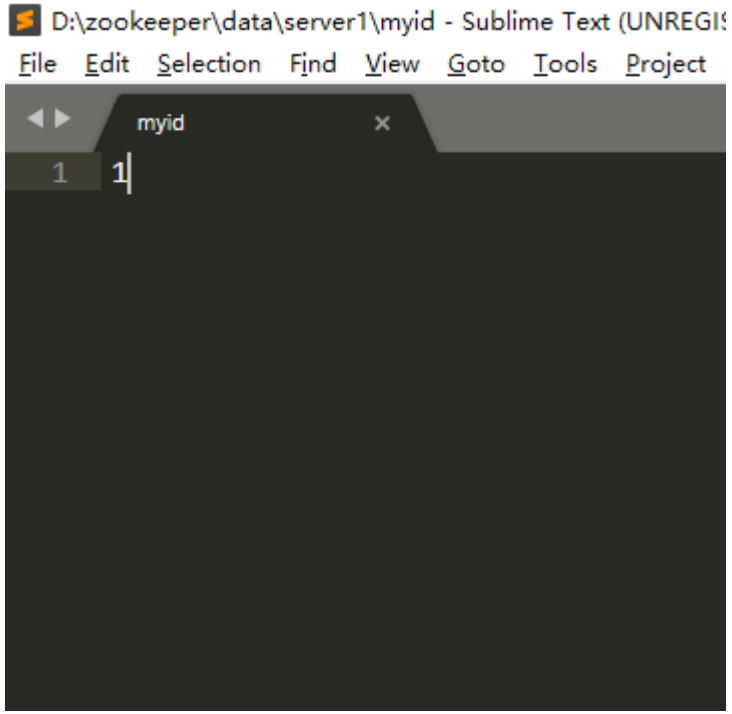
```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=D:\\zookeeper\\data\\server1
#
# the port at which the clients will connect
clientPort=2181

server.1=127.0.0.1:2883:3883
server.2=127.0.0.1:2881:3881
server.3=127.0.0.1:2882:3882
```

ZooKeeper配置

创建dataDir，并且在dataDir中创建一个myid文件，内容为服务器编号，称为SID





ZooKeeper启动和连接

启动：

```
${ZK_HOME}/bin/zkServer.sh start
```

命令行连接：

```
${ZK_HOME}/bin/zkCli.sh -server host:port
```

客户端常见命令

功能	命令
查看节点列表	ls path
创建节点	create [-e] [-s] path data acl
获取节点	get path
修改节点数据	set path data [version]
删除节点	delete path [version]

Java API使用

- 创建会话
- 创建节点
- 获取数据
- 删除节点
- 更新数据
- 检测节点是否存在

创建会话

- ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)
- ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly)

创建会话

参数名	说明
connectString	指ZooKeeper服务器列表，由英文状态逗号host:port字符串组成，每一个都代表一台ZooKeeper机器，例如，192.168.1.1:2181,192.168.1.2:2181,192.168.1.3:2181，这样就为客户端指定了三台服务器地址。另外，也可以在connectString中设置客户端连接上ZooKeeper后的根目录，方法是在host:port字符串之后添加上这个根目录，例如：192.168.1.1:2181,192.168.1.2:2181,192.168.1.3:2181/zk-book，这样就指定了该客户端连接上ZooKeeper服务器之后，所有对ZooKeeper的操作，都会基于这个根目录。例如，客户端对/foo/bar的操作，都会指向节点/zk-book/foo/bar——这个目录也叫Chroot，即客户端隔离命名空间。
sessionTimeout	指会话的超时时间，是一个以“毫秒”为单位的整型值。在ZooKeeper中有会话的概念，在一个会话周期内，ZooKeeper客户端和服务端之间会通过心跳检测机制来维持会话的有效性，一旦在sessionTimeout时间内没有进行有效的心跳检测，会话就会失效。
watcher	ZooKeeper允许客户端在构造方法中传入一个接口Watcher的实现类对象来作为默认的Watcher事件通知处理器。当然，该参数可以设置为null以表明不需要设置默认的Watcher处理器。
canBeReadOnly	这是一个boolean类型的参数，用于标识当前会话是否支持“read-only”模式。默认情况下，在ZooKeeper集群中，一个机器如果和集群中过半以上机器失去了网络连接，那么这个机器将不再处理客户端请求（包括读写请求）。但是在某些使用场景下，当ZooKeeper服务器发生此类故障的时候，我们还是希望ZooKeeper服务器能够提供读服务（当然写服务肯定无法提供）——这就是ZooKeeper的“read-only”模式。
sessionId和sessionPasswd	分别代表会话ID和会话秘钥。这两个参数能够唯一确定一个会话，同时客户端使用这两个参数实现客户端会话复用，从而达到恢复会话的效果，具体使用方法是，第一次连接上ZooKeeper服务器时，通过调用ZooKeeper对象实现的一下两个接口，即可获得当前会话的ID和秘钥：long getSessionId(); byte[] getSessionPasswd(); 获取到这两个参数值后，就可以在下次创建ZooKeeper对象实例的时候传入构造方法了

创建节点

- String create(final String path,byte data[],List<ACL> acl,CreateMode createMode)
- void create(final String path,byte date[],List<ACL> acl,CreateMode createMode,StringCallback cb,Object ctx)

创建节点

参数名	说明
path	需要创建的数据节点的节点路径，录入，/zk-book/foo
data[]	一个字节数组，是节点创建后的初始内容
acl	节点的ACL策略
createMode	节点类型
ctx	用于传递一个对象，可以在回调方法执行的时候使用，通常是一个上下文(Context)信息
cb	注册一个异步回调函数。开发人员需要实现StringCallBack接口，主要是对下面这个方法的重写：void processResult(int rc,String path,Object ctx,String name); 当服务端节点创建完毕后，ZooKeeper客户端就会自动调用这个方法，这样就可以处理相关的业务逻辑了

获取数据

byte[] getData(String path, boolean watch, Stat stat)

void getData(final String path, Watcher watcher, DataCallback cb, Object ctx)

List<String> getChildren(String path, boolean watch)

void getChildren(String path, boolean watch, ChildrenCallback cb, Object ctx)

List<String> getChildren(String path, Watcher watcher)

void getChildren(String path, Watcher watcher, ChildrenCallback cb, Object ctx)

获取数据

参数名	说明
path	需要创建的数据节点的节点路径，录入，/zk-book/foo
watcher	注册的watcher，一旦在本次节点获取之后，子节点列表发生变更的话，那么就会像客户端发送通知，该参数允许传入null
watch	表明是否需要注册一个watcher。 在上面<创建会话>中我们提到过一个默认watcher的概念，这里我们使用该默认watcher了。 如果参数为true，那么ZooKeeper客户端会自动使用上文中默认的watcher； 如果参数为false，那就表明不需要注册watcher。
ctx	用于传递一个对象，可以在回调方法执行的时候使用，通常是一个上下文(Context)信息
cb	注册一个异步回调函数。开发人员需要实现StringCallBack接口，主要是对下面这个方法的重写：void processResult(int rc,String path,Object ctx,String name); 当服务端节点创建完毕后，ZooKeeper客户端就会自动调用这个方法，这样就可以处理相关的业务逻辑了

04

章节 PART

ZooKeeper技术解析

Chapter Four

ZooKeeper技术解析

1. ZAB协议
2. Leader选举
3. 一次事务请求的过程
4. 数据和存储
5. 服务器初始化
6. 数据同步

ZAB协议（ZooKeeper Atomic Broadcast）

ZAB协议是专门为ZooKeeper设计的一种崩溃可恢复的原子广播协议。

基于此协议，ZooKeeper实现了一种主备模式架构来保持集群中个副本之间的数据一致性，使用多数派决议进行选举Leader。具体的，ZooKeeper使用一个单一的主进程来接收并处理客户端的所有事务请求，并采用ZAB协议，将服务器数据的状态变更以事务提案（Proposal）的形式广播到集群中所有的服务器中去。

ZAB协议（ZooKeeper Atomic Broadcast）

ZAB协议分为崩溃恢复和消息广播两种基本模式。

当整个服务集群刚刚启动，或者是当Leader服务器出现网络中断、崩溃退出、重启等异常情况时，ZAB协议就会进入恢复模式并选举产生新的Leader服务器。然后在Leader与过半的服务器同步数据之后结束崩溃恢复模式，开始消息广播模式。

新服务器加入一个已有Leader的集群中时，会直接进行消息同步，结束后加入到消息广播流程中。

Leader选举

Leader选举是保证分布式数据一致性的关键所在。当Zookeeper集群中的一台服务器出现以下两种情况之一时，需要进入Leader选举。

(1) 服务器初始化启动。

(2) 服务器运行期间无法和Leader保持连接。

选举中的概念

SID：dataDir路径下myid中的数字，代表了这台服务器的ID。

ZXID：即事务ID，代表某台服务器最新事务的ZXID。

逻辑时钟：用来确认多个投票是否属于同一轮选举周期。

服务器状态：该服务器所处状态，包括LOOKING、LEADING、FOLLOWING、OBSERVERING四种。

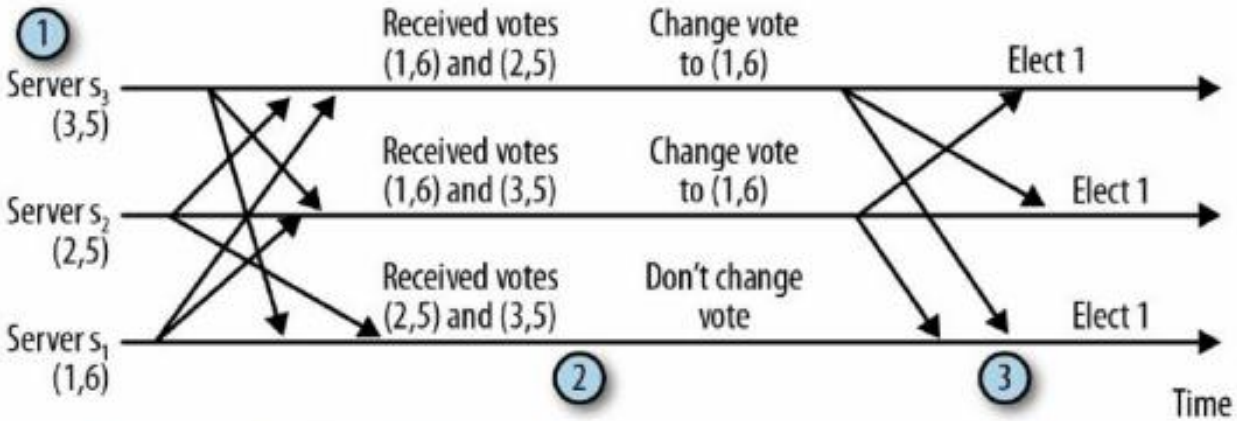
选票PK：对比自己投的票和其他服务器投的票。

选举步骤

当服务器处于LOOKING状态时，就会向集群中的其他节点发送信息，这个信息叫做投票，投票的基本元素包括SID和ZXID两项，例如（1,6）代表推举SID为1，ZXID为6的服务器为Leader。

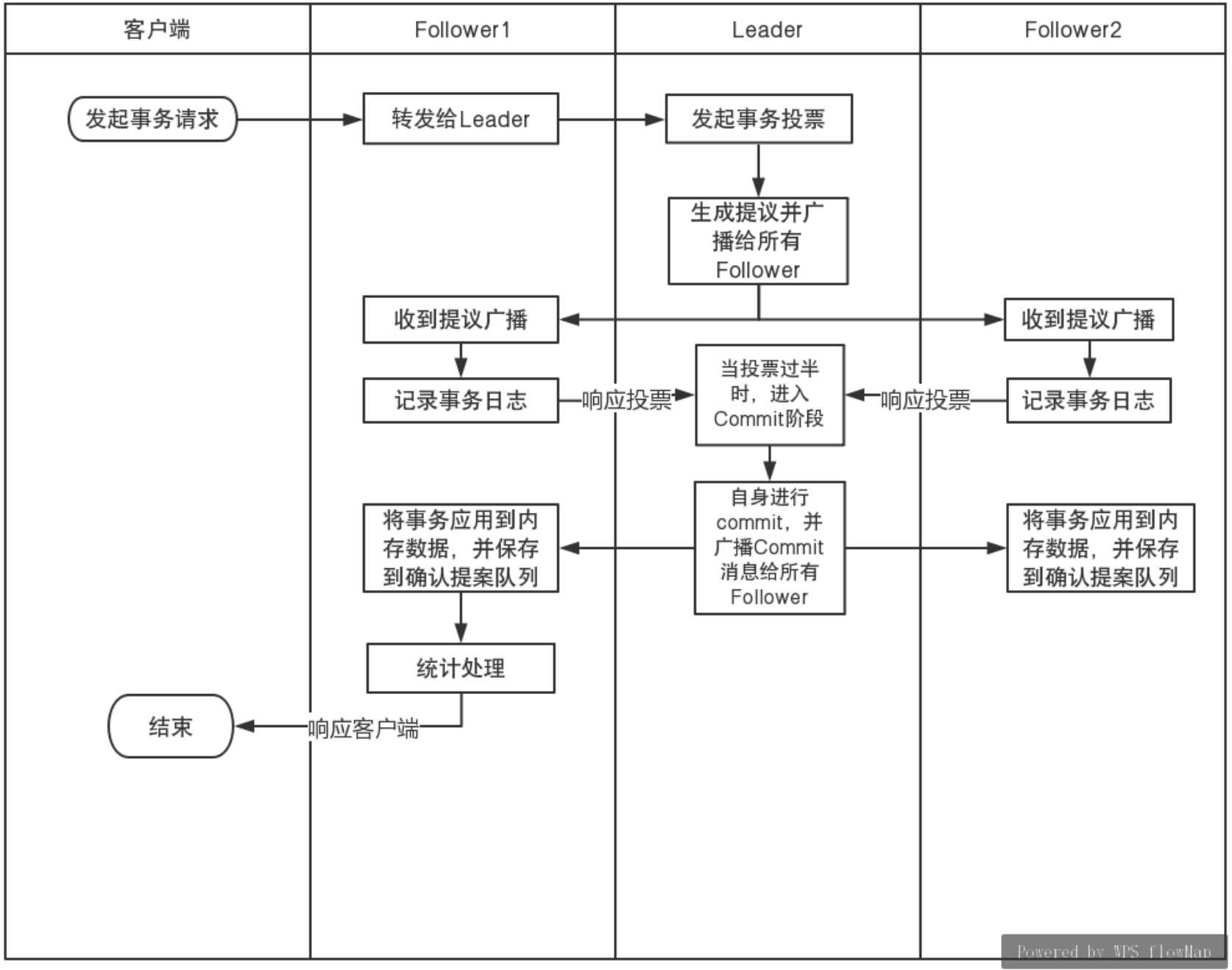
- 发出自己的投票：第一次投票时，还无法检测到集群中其他机器的状态，因此每台服务器都推举自己为Leader
- 接收其他服务器的投票：接收到投票首先验证有效性，然后用自己的投票和收到的投票进行选票PK，PK的规则很简单，首先选择ZXID最大的那个，如果ZXID相同，则选择SID最大的那个，再将更新后的投票发出去。
- 统计投票：每次投票后，服务器会统计所以投票，如果有过半的机器接收到相同的投票，则认为已经选出了Leader
- 改变服务器的状态，如果是Leader，就更新成Leading，如果是Follower就更新成Following。

Leader选举



1. 服务器S_1的投票值为（1,6），服务器S_2的投票值为（2,5），服务器S_3的投票值为（3,5）。
2. 服务器S_2和服务器S_3将会改变其投票（vote）值为（1,6）并发送新的通知消息。
3. 所有服务器从仲裁处收到一样的投票（vote）值，选举服务器S_1为群首。

事务请求



数据和存储

一、内存数据

ZooKeeper所有的数据都存储在内存中，像一个内存数据库一样，存储了整棵树的内容，包括所有节点路径、节点数据及节点属性。

数据和存储

二、事务日志

事务日志类似于MySQL的binlog和Redis的AOF文件，将所有通过的事务都以一定的格式存储在预分配的日志文件中，日志文件的存储在配置文件中的dataLogDir对应位置。

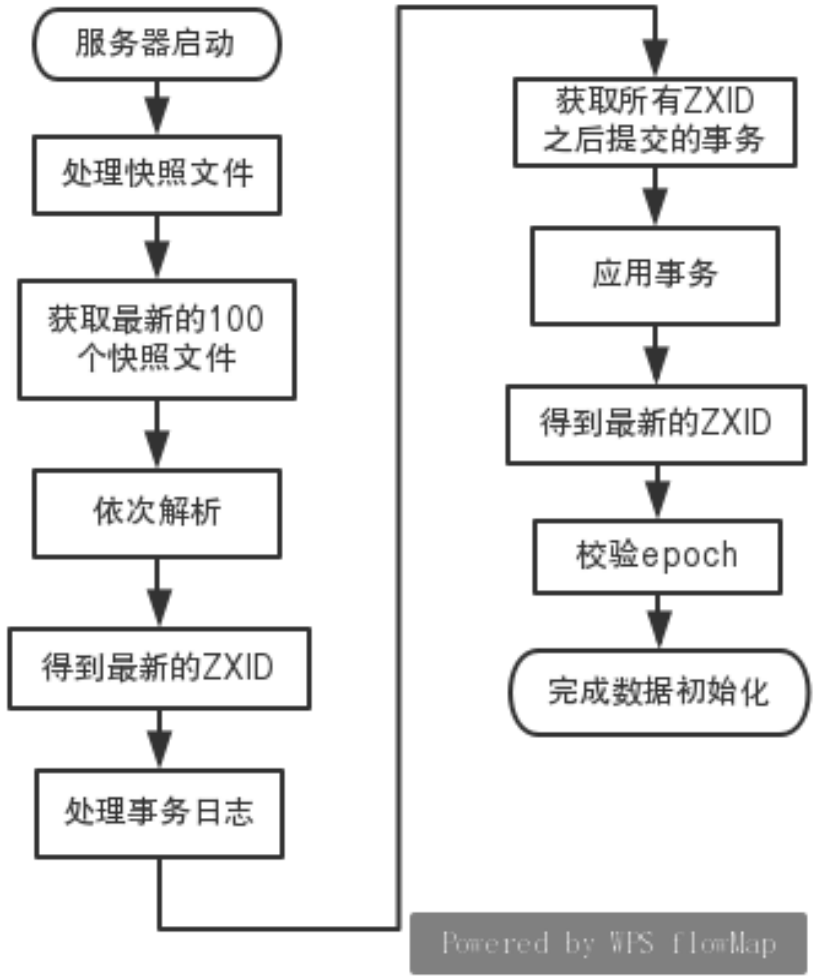
```
java -classpath "D:\zookeeper-3.4.12\lib\slf4j-api-1.7.25.jar;D:\zookeeper-3.4.12\zookeeper-3.4.12.jar" org.apache.zookeeper.server.LogFormatter .\log.100000001
```

数据和存储

三、数据快照

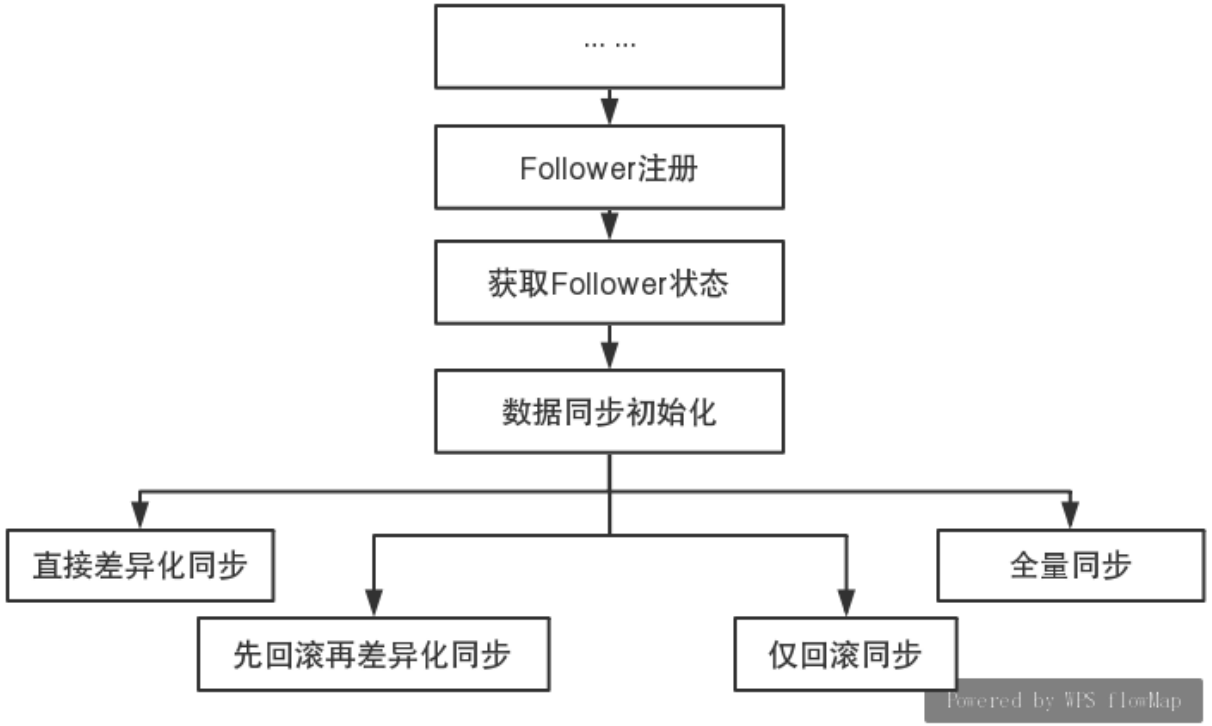
顾名思义，数据快照记录了ZooKeeper服务器某个时刻的全量内存数据内容，并将其写入指定的磁盘文件中。

数据初始化



数据同步

当集群服务器完成数据初始化后会立即进行Leader选举，在Follower向Leader注册完毕就会进入数据同步环节。数据同步过程就是Leader服务器将那些没在Follower服务器上提交的日志请求同步给Follower服务器。



数据同步

在服务器初始化过程中，会把从事务日志中提取出来的事务维护在一个内存队列中（下面称为commitedLog队列）。

当Leader获取Follwer状态时，会完成对以下三个ZXID值的初始化。

- peerLastZxid：该Follower服务器最后处理的ZXID
- minCommittedLog：Leader服务器commitedLog队列中最小的ZXID
- maxCommittedLog：Leader服务器commitedLog队列中最大的ZXID

数据同步

- 直接差异化同步

  peerLastZxid介于minCommittedLog和maxCommittedLog之间

- 先回滚再差异化同步

  peerLastZxid的epoch小于Leader，且Leader的CommittedLog中不包含peerLastZxid

- 仅回滚同步

  peerLastZxid大于maxCommittedLog

- 全量同步

  peerLastZxid小于minCommittedLog

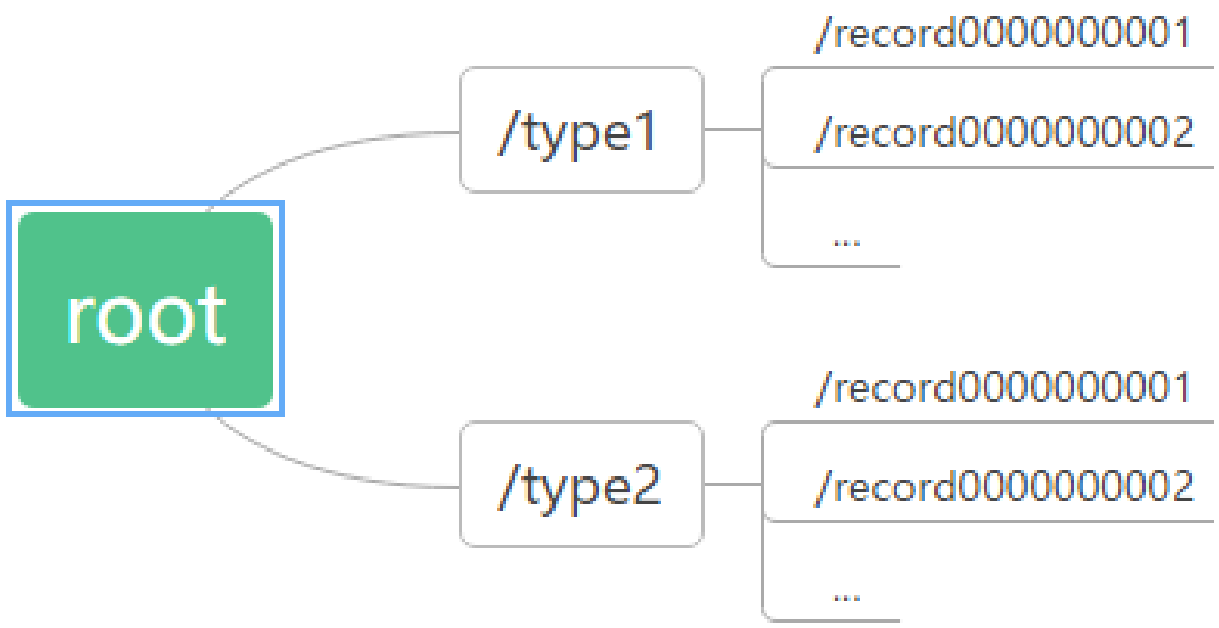
05

章节 PART

ZooKeeper应用场景

Chapter Five

全局唯一ID（命名服务）



分布式锁——排它锁

排它锁，又称写锁或独占锁，是一种基本的锁类型。若事务T对数据对象A加上排它锁，则只允许T读取和修改A，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁。

分布式锁——排它锁

1. 定义锁：通过ZooKeeper上的一个临时子节点来表示一个锁，例如/exclusive_lock/lock节点就可以当成一个锁。

2. 获取锁：当需要获取锁时，所有客户端都会试图通过调用create()接口，在/exlusive_lock节点下创建临时子节点/exclusive_lock/lock。由于在ZK会保证在所有的客户端之中，最终只有一个客户端能够创建成功，那么就可以认为该客户端获取了锁。

其他没有获取到锁的客户端，就需要在/exclusive_lock节点注册一个Watcher用于监听子节点变化，以便实时获取到锁节点的变更情况。

分布式锁——排它锁

3. 释放锁

因为锁节点是一个临时节点，所以有以下两种情况可能释放锁。

- 当前拥有锁的客户端发生宕机，ZK上的临时节点被删除。
- 正常执行完业务逻辑之后，客户端主动删除自己创建的临时节点。

不论在那种情况下删除了锁节点，ZooKeeper都会通知所有在/exlusive_lock节点注册Watcher的客户端。这些客户端在接收到通知后，会再次发起对锁的争夺，即重复“获取锁”的流程。

分布式锁——共享锁

共享锁，又称读锁。若事务T对数据对象A加上共享锁，则其它事务只能再对A加共享锁，而不能加排它锁，直到T释放A上的共享锁。

分布式锁——共享锁

1. 定义锁：和排它锁一样，同样是通过ZooKeeper上的数据节点来表示一个锁，是一个类似于“/shared_lock/[hostname]-请求类型-序号”的临时顺序节点。例如/shared_lock/192.168.1.1-R-0000000001，这样的节点代表了一个共享锁。

2. 获取锁：当需要获取共享锁时，所有客户端会在/shared_lock这个节点下面创建一个临时顺序节点，如果是读请求，那么就创建例如/shared_lock/192.168.1.1-R-0000000001的节点，如果是写请求，那么就创建例如/shared_lock/192.168.1.1-W-0000000001的节点。

分布式锁——共享锁

3. 判断读写顺序：根据共享锁的定义，不同的事务可以同时对同一个数据对象进行读操作，但是更新操作必须在当前没有任何事务进行读写操作的情况下进行。

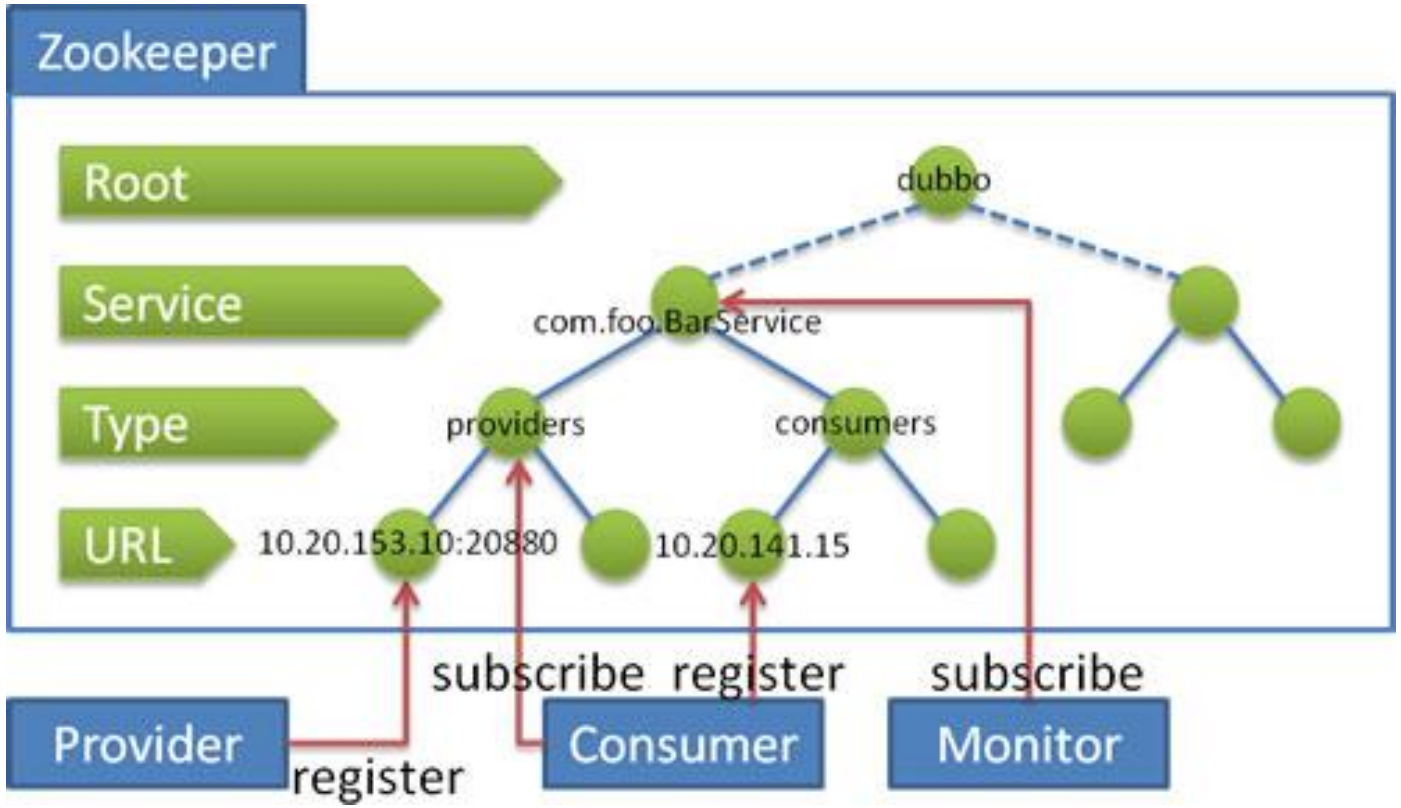
基于这个原则，我们来看看如何通过ZooKeeper的节点来确定读写顺序，大致可以分为一下几个步骤。

1. 创建节点后，获取/shared_lock下所有子节点。
2. 确认自己在节点列表中的位置。
3. 对于读请求：如果没有比自己序号小的节点，或是所有比自己序号小的节点都是读请求，那么表示自己已经成功获取到了共享锁，可以开始执行业务逻辑。如果比自己序号小的子节点中有写请求，那么就需要进入等待。同时向比自己序号小的最后一个写请求节点注册Watcher监听。对于写请求：如果自己不是序号最小的子节点，那么就需要进入等待。同时向比自己序号小的最后一个节点注册Watcher监听。
4. 等待Watcher通知，重复步骤1-3。

ZooKeeper的常见应用场景

- 数据发布订阅：对统一配置信息等数据可以通过在Zookeeper创建一个数据节点并让客户端进行监听，主要利用了Zookeeper的Watcher监听特性
- 负载均衡：创建一个节点，负载应用把自己的服务地址写到此节点下，如果此应用挂掉，则此子节点消失
- 命名服务：利用Zookeeper创建顺序无重复子节点的特性
- 分布式协调/通知：不同客户端都对Zookeeper上的同一个数据节点进行watcher注册，监听数据节点的变化，当发生变化所有订阅的客户端接收到通知并进行处理
- 集群管理：利用了watcher监听与临时节点在会话失效自动清除的特性。同时，各服务器可以讲运行状态信息写入到临时节点中进而有助于Leader收集负载信息
- Master选举：所有客户端创建同一个path的数据节点，只有一个能成功，即为Master
- 分布式锁：创建临时节点，谁成功即获得锁。另外，根据创建时不同的类型-序号，根据一定的规则可以模拟出共享锁、读写锁等
- 分布式队列：每个客户端在指定节点下创建临时节点，然后获取该指定节点下的所有子节点并判断自己是否是序号最小的节点，如果是则可以进行处理，如果不是则进入等待并监听比自己序号小的最后一个节点，待接到watcher通知后，重复检查。

ZooKeeper在Dubbo中的应用



谢谢观看

THANK YOU

友信普惠
友信金服旗下品牌