Zookeeper

第一章 简介及核心概念

简介

Zookeeper 是一个开源的分布式协调服务，目前由 Apache 进行维护。Zookeeper 可以用于实现分布式系统中常见的发布/订阅、负载均衡、命令服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等功能。它具有以下特性：

- **顺序一致性**：从一个客户端发起的事务请求，最终都会严格按照其发起顺序被应用到 Zookeeper 中；
- **原子性**：所有事务请求的处理结果在整个集群中所有机器上都是一致的；不存在部分机器应用了该事务，而另一部分没有应用的情况；
- **单一视图**：所有客户端看到的服务端数据模型都是一致的；
- **可靠性**：一旦服务端成功应用了一个事务，则其引起的改变会一直保留，直到被另外一个事务所更改；
- **实时性**：一旦一个事务被成功应用后，Zookeeper 可以保证客户端立即可以读取到这个事务变更后的最新状态的数据。

设计目标

Zookeeper 致力于为那些高吞吐的大型分布式系统提供一个高性能、高可用、且具有严格顺序访问控制能力的分布式协调服务。它具有以下四个目标：

2.1 目标一：简单的数据模型

Zookeeper 通过树形结构来存储数据，它由一系列被称为 ZNode 的数据节点组成，类似于常见的文件系统。不过和常见的文件系统不同，Zookeeper 将数据全量存储在内存中，以此来实现高吞吐，减少访问延迟。

2.2 目标二：构建集群

可以由一组 Zookeeper 服务构成 Zookeeper 集群，集群中每台机器都会单独在内存中维护自身的状态，并且每台机器之间都保持着通讯，只要集群中有半数机器能够正常工作，那么整个集群就可以正常提供服务。

2.3 目标三：顺序访问

对于来自客户端的每个更新请求，Zookeeper 都会分配一个全局唯一的递增 ID，这个 ID 反映了所有事务请求的先后顺序。

2.4 目标四：高性能高可用

ZooKeeper 将数据存全量储在内存中以保持高性能，并通过服务集群来实现高可用，由于 Zookeeper 的所有更新和删除都是基于事务的，所以其在读多写少的应用场景中有着很高的性能表现。

核心概念

3.1 集群角色

Zookeeper 集群中的机器分为以下三种角色：

- **Leader** ： 为客户端提供读写服务，并维护集群状态，它是由集群选举所产生的；
- **Follower** ： 为客户端提供读写服务，并定期向 Leader 汇报自己的节点状态。同时也参与写操作“过半写成功”的策略和 Leader 的选举；
- **Observer** ： 为客户端提供读写服务，并定期向 Leader 汇报自己的节点状态，但不参与写操作“过半写成功”的策略和 Leader 的选举，因此 Observer 可以在不影响写性能的情况下提升集群的读性能。

3.2 会话

Zookeeper 客户端通过 TCP 长连接连接到服务集群，会话 (Session) 从第一次连接开始就已经建立，之后通过心跳检测机制来保持有效的会话状态。通过这个连接，客户端可以发送请求并接收响应，同时也可以接收到 Watch 事件的通知。

关于会话中另外一个核心的概念是 sessionTimeOut(会话超时时间)，当由于网络故障或者客户端主动断开等原因，导致连接断开，此时只要在会话超时时间之内重新建立连接，则之前创建的会话依然有效。

3.3 数据节点

Zookeeper 数据模型是由一系列基本数据单元 `Znode` (数据节点) 组成的节点树，其中根节点为 `/` 。每个节点上都会保存自己的数据和节点信息。Zookeeper 中节点可以分为两大类：

- **持久节点** ： 节点一旦创建，除非被主动删除，否则一直存在；
- **临时节点** ： 一旦创建该节点的客户端会话失效，则所有该客户端创建的临时节点都会被删除。

临时节点和持久节点都可以添加一个特殊的属性： `SEQUENTIAL` ，代表该节点是否具有递增属性。如果指定该属性，那么在这个节点创建时，Zookeeper 会自动在其节点名称后面追加一个由父节点维护的递增数字。

3.4 节点信息

每个 ZNode 节点在存储数据的同时，都会维护一个叫做 `Stat` 的数据结构，里面存储了关于该节点的全部状态信息。如下：

状态属性	说明
czxid	数据节点创建时的事务 ID
ctime	数据节点创建时的时间
mzxid	数据节点最后一次更新时的事务 ID
mtime	数据节点最后一次更新时的时间
pzxid	数据节点的子节点最后一次被修改时的事务 ID
cversion	子节点的更改次数
version	节点数据的更改次数
aversion	节点的 ACL 的更改次数
ephemeralOwner	如果节点是临时节点，则表示创建该节点的会话的 SessionID；如果节点是持久节点，则该属性值为 0
dataLength	数据内容的长度
numChildren	数据节点当前的子节点个数

3.5 Watcher

Zookeeper 中一个常用的功能是 Watcher(事件监听器)，它允许用户在指定节点上针对感兴趣的事件注册监听，当事件发生时，监听器会被触发，并将事件信息推送到客户端。该机制是 Zookeeper 实现分布式协调服务的重要特性。

3.6 ACL

Zookeeper 采用 ACL(Access Control Lists) 策略来进行权限控制，类似于 UNIX 文件系统的权限控制。它定义了如下五种权限：

- **CREATE**：允许创建子节点；
- **READ**：允许从节点获取数据并列出其子节点；
- **WRITE**：允许为节点设置数据；
- **DELETE**：允许删除子节点；
- **ADMIN**：允许为节点设置权限。

ZAB协议

4.1 ZAB协议与数据一致性

ZAB 协议是 Zookeeper 专门设计的一种支持崩溃恢复的原子广播协议。通过该协议，Zookeepe 基于主从模式的系统架构来保持集群中各个副本之间数据的一致性。具体如下：

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，并采用原子广播协议将数据状态的变更以事务 Proposal 的形式广播到所有的副本进程上去。如下图：

具体流程如下：

所有的事务请求必须由唯一的 Leader 服务来处理，Leader 服务将事务请求转换为事务 Proposal，并将该 Proposal 分发给集群中所有的 Follower 服务。如果有半数的 Follower 服务进行了正确的反馈，那么 Leader 就会再次向所有的 Follower 发出 Commit 消息，要求将前一个 Proposal 进行提交。

4.2 ZAB协议的内容

ZAB 协议包括两种基本的模式，分别是崩溃恢复和消息广播：

1. 崩溃恢复

当整个服务框架在启动过程中，或者当 Leader 服务器出现异常时，ZAB 协议就会进入恢复模式，通过过半选举机制产生新的 Leader，之后其他机器将从新的 Leader 上同步状态，当有过半机器完成状态同步后，就退出恢复模式，进入消息广播模式。

2. 消息广播

ZAB 协议的消息广播过程使用的是原子广播协议。在整个消息的广播过程中，Leader 服务器会每个事物请求生成对应的 Proposal，并为其分配一个全局唯一的递增的事务 ID(ZXID)，之后再对其进行广播。具体过程如下：

Leader 服务会为每一个 Follower 服务器分配一个单独的队列，然后将事务 Proposal 依次放入队列中，并根据 FIFO(先进先出) 的策略进行消息发送。Follower 服务在接收到 Proposal 后，会将其以事务日志的形式写入本地磁盘中，并在写入成功后反馈给 Leader 一个 Ack 响应。当 Leader 接收到超过半数 Follower 的 Ack 响应后，就会广播一个 Commit 消息给所有的 Follower 以通知其进行事务提交，之后 Leader 自身也会完成对事务的提交。而每一个 Follower 则在接收到 Commit 消息后，完成事务的提交。

典型应用场景

5.1数据的发布/订阅

数据的发布/订阅系统，通常也用作配置中心。在分布式系统中，你可能有成千上万个服务节点，如果想要对所有服务的某项配置进行更改，由于数据节点过多，你不可逐台进行修改，而应该在设计时采用统一的配置中心。之后发布者只需要将新的配置发送到配置中心，所有服务节点即可自动下载并进行更新，从而实现配置的集中管理和动态更新。

Zookeeper 通过 Watcher 机制可以实现数据的发布和订阅。分布式系统的所有的服务节点可以对某个 ZNode 注册监听，之后只需要将新的配置写入该 ZNode，所有服务节点都会收到该事件。

5.2 命名服务

在分布式系统中，通常需要一个全局唯一的名字，如生成全局唯一的订单号等，Zookeeper 可以通过顺序节点的特性来生成全局唯一 ID，从而可以对分布式系统提供命名服务。

5.3 Master选举

分布式系统一个重要的模式就是主从模式 (Master/Salves)，Zookeeper 可以用于该模式下的 Matser 选举。可以让所有服务节点去竞争性地创建同一个 ZNode，由于 Zookeeper 不能有路径相同的 ZNode，必然只有一个服务节点能够创建成功，这样该服务节点就可以成为 Master 节点。

5.4 分布式锁

可以通过 Zookeeper 的临时节点和 Watcher 机制来实现分布式锁，这里以排它锁为例进行说明：

分布式系统的所有服务节点可以竞争性地去创建同一个临时 ZNode，由于 Zookeeper 不能有路径相同的 ZNode，必然只有一个服务节点能够创建成功，此时可以认为该节点获得了锁。其他没有获得锁的服务节点通过在该 ZNode 上注册监听，从而当锁释放时再去竞争获得锁。锁的释放情况有以下两种：

- 当正常执行完业务逻辑后，客户端主动将临时 ZNode 删除，此时锁被释放；
- 当获得锁的客户端发生宕机时，临时 ZNode 会被自动删除，此时认为锁已经释放。

当锁被释放后，其他服务节点则再次去竞争性地进行创建，但每次都只有一个服务节点能够获取到锁，这就是排他锁。

5.5 集群管理

Zookeeper 还能解决大多数分布式系统中的问题：

- 如可以通过创建临时节点来建立心跳检测机制。如果分布式系统的某个服务节点宕机了，则其持有的会话会超时，此时该临时节点会被删除，相应的监听事件就会被触发。
- 分布式系统的每个服务节点还可以将自己的节点状态写入临时节点，从而完成状态报告或节点工作进度汇报。
- 通过数据的订阅和发布功能，Zookeeper 还能对分布式系统进行模块的解耦和任务的调度。
- 通过监听机制，还能对分布式系统的服务节点进行动态上下线，从而实现服务的动态扩容。

第二章 常用Shell命令

节点增删改查

1.1 启动服务和连接服务

```
# 启动服务
bin/zkServer.sh start

#连接服务 不指定服务地址则默认连接到localhost:2181
zkCli.sh -server hadoop001:2181
```

1.2 help命令

使用 `help` 可以查看所有命令及格式。

```
ZooKeeper -server host:port -client-configuration properties-file cmd args
          addWatch [-m mode] path # optional mode is one of [PERSISTENT, PERSISTENT_RECURSIVE] - default is PERSISTENT_RECURSIVE
          addauth scheme auth
          close
          config [-c] [-w] [-s]
          connect host:port
          create [-s] [-e] [-c] [-t ttl] path [data] [acl]
          delete [-v version] path
          deleteall path [-b batch size]
          delquota [-n|-b] path
          get [-s] [-w] path
          getAcl [-s] path
```

```
            getAllChildrenNumber path
            getEphemerals path
            history
            listquota path
            ls [-s] [-w] [-R] path
            printwatches on|off
            quit
            reconfig [-s] [-v version] [[-file path] | [-members
serverID=host:port1:port2;port3[,...]*]] | [-add serverId=host:port1:port2;port3[,...]]* [-remove
serverId[,...]*]
            redo cmdno
            removewatches path [-c|-d|-a] [-l]
            set [-s] [-v version] path data
            setAcl [-s] [-v version] [-R] path acl
            setquota -n|-b val path
            stat [-w] path
            sync path
            version
```

1.3 查看节点列表

查看节点列表有 `ls path` 和 `ls -s path` 两个命令，后者是前者的增强，不仅可以查看指定路径下的所有节点，还可以查看当前节点的信息。

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zk: localhost:2181(CONNECTED) 1] ls -s /
[a0000000001, b0000000002, c0000000003, hadoop, zookeeper]
cZxid = 0x0
ctime = Thu Jan 01 08:00:00 CST 1970
mZxid = 0x0
mtime = Thu Jan 01 08:00:00 CST 1970
pZxid = 0x300000011
cversion = 9
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 0
numChildren = 7
```

1.4 新增节点

```
create [-s] [-e] [-c] [-t ttl] path [data] [acl]    #其中-s 为有序节点，-e 临时节点
```

创建节点并写入数据：

```
create /hadoop 123456
```

创建有序节点，此时创建的节点名为指定节点名 + 自增序号：

```
[zk: localhost:2181(CONNECTED) 23] create -s /a   "aaa"
Created /a0000000022
[zk: localhost:2181(CONNECTED) 24] create -s /b   "bbb"
Created /b0000000023
[zk: localhost:2181(CONNECTED) 25] create -s /c   "ccc"
Created /c0000000024
```

创建临时节点，临时节点会在会话过期后被删除：

```
[zk: localhost:2181(CONNECTED) 26] create -e /tmp   "tmp"
Created /tmp
```

1.5 查看节点

1. 获取节点数据

```
# 格式
# get [-s] [-w] path
[zk: localhost:2181(CONNECTED) 24] get -s -w /hadoop
123456
cZxid = 0x200000002
ctime = Sat Nov 12 17:08:09 CST 2020
mZxid = 0x200000002
mtime = Sat Nov 12 17:08:09 CST 2020
pZxid = 0x200000002
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0

[zk: localhost:2181(CONNECTED) 25] get -s -w /tmp
tmp
cZxid = 0x200000006
ctime = Sat Nov 12 17:10:45 CST 2021
mZxid = 0x200000006
mtime = Sat Nov 12 17:10:45 CST 2021
pZxid = 0x200000006
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x1000251b9bb0000
dataLength = 3
numChildren = 0
```

节点各个属性如下表。其中一个重要的概念是 Zxid(ZooKeeper Transaction Id)，ZooKeeper 节点的每一次更改都具有唯一的 Zxid，如果 Zxid1 小于 Zxid2，则 Zxid1 的更改发生在 Zxid2 更改之前。

状态属性	说明
cZxid	数据节点创建时的事务 ID
ctime	数据节点创建时的时间
mZxid	数据节点最后一次更新时的事务 ID
mtime	数据节点最后一次更新时的时间
pZxid	数据节点的子节点最后一次被修改时的事务 ID
cversion	子节点的更改次数
dataVersion	节点数据的更改次数
aclVersion	节点的 ACL 的更改次数
ephemeralOwner	如果节点是临时节点，则表示创建该节点的会话的 SessionID；如果节点是持久节点，则该属性值为 0
dataLength	数据内容的长度
numChildren	数据节点当前的子节点个数

2. 查看节点状态

可以使用 `stat` 命令查看节点状态，它的返回值和 `get` 命令类似，但不会返回节点数据。

```
[zk: localhost:2181(CONNECTED) 32] stat /hadoop
cZxid = 0x14b
ctime = Fri May 24 17:03:06 CST 2019
mZxid = 0x14b
mtime = Fri May 24 17:03:06 CST 2019
pZxid = 0x14b
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

1.6 更新节点

更新节点的命令是 `set`，可以直接进行修改，如下：

```
[zk: localhost:2181(CONNECTED) 33] set /hadoop 345
cZxid = 0x14b
ctime = Fri May 24 17:03:06 CST 2019
mZxid = 0x14c
mtime = Fri May 24 17:13:05 CST 2019
pZxid = 0x14b
cversion = 0
dataVersion = 1   # 注意更改后此时版本号为 1，默认创建时为 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 3
numChildren = 0
```

也可以基于版本号进行更改，此时类似于乐观锁机制，当你传入的数据版本号 (dataVersion) 和当前节点的数据版本号不符合时，zookeeper 会拒绝本次修改：

```
[zk: localhost:2181(CONNECTED) 34] set  -v 0 /hadoop 678
version No is not valid : /hadoop  #无效的版本号
```

1.7 删除节点

删除节点的语法如下：

```
delete path -v [version]
```

和更新节点数据一样，也可以传入版本号，当你传入的数据版本号 (dataVersion) 和当前节点的数据版本号不符合时，zookeeper 不会执行删除操作。

```
[zk: localhost:2181(CONNECTED) 35] delete /hadoop -v 0
version No is not valid : /hadoop
[zk: localhost:2181(CONNECTED) 36] delete /hadoop
```

要想删除某个节点及其所有后代节点，可以使用递归删除，命令为 `rmr path`。

监听器

2.1 get [-w] path

使用 `get [-w] path` 注册的监听器能够在节点内容发生改变的时候，向客户端发出通知。需要注意的是 zookeeper 的触发器是一次性的 (One-time trigger)，即触发一次后就会立即失效。

```
[zk: localhost:2181(CONNECTED) 4] get -w /hadoop
[zk: localhost:2181(CONNECTED) 5] set /hadoop 456

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/hadoop
```

2.2 stat [-w] path

使用 `stat [-w] path` 注册的监听器能够在节点状态发生改变的时候，向客户端发出通知。

```
[zk: localhost:2181(CONNECTED) 7] stat -w /hadoop
[zk: localhost:2181(CONNECTED) 8] set /hadoop 112233
WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/hadoop
```

2.3 ls [-s] [-w] [-R] path

使用 `ls [-s] [-w] -R path` 注册的监听器能够监听该节点下所有**子节点**的增加和删除操作。

```
[zk: localhost:2181(CONNECTED) 9] ls -w -R /hadoop
[]
[zk: localhost:2181(CONNECTED) 10] create  /hadoop/yarn "aaa"
WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/hadoop
```

zookeeper 四字命令

命令	功能描述
conf	打印有关服务配置的详细信息。
cons	列出连接到此服务器的所有客户端的完整连接/会话详细信息。包括有关接收/发送的数据包数量、会话 ID、操作延迟、上次执行的操作等信息...
crst	重置所有连接的连接/会话统计信息。
dump	列出未完成的会话和临时节点。
envi	打印有关服务环境的详细信息
ruok	测试服务器是否在非错误状态下运行。如果服务器正在运行，它将以 imok 响应。否则它根本不会响应。“imok”响应并不一定表示服务器已加入仲裁，只是服务器进程处于活动状态并绑定到指定的客户端端口。使用“stat”获取有关状态wrt quorum和客户端连接信息的详细信息。
srst	重置服务器统计信息。
srvr	列出服务器的完整详细信息。
stat	列出服务器和连接客户端的简要详细信息。
wchs	列出有关服务器监视的简要信息。
wchc	按会话列出有关服务器监视的详细信息。这将输出具有关联手表（路径）的会话（连接）列表。请注意，根据观察次数，此操作可能会很昂贵（即影响服务器性能），请谨慎使用。
dirs	以字节为单位显示快照和日志文件的总大小
wchp	按路径列出有关服务器监视的详细信息。这将输出具有关联会话的路径（znode）列表。请注意，根据观察次数，此操作可能会很昂贵（即影响服务器性能），请谨慎使用。
mntr	输出可用于监控集群健康状况的变量列表。
isro	测试服务器是否以只读模式运行。如果处于只读模式，服务器将响应“ro”，如果不是只读模式，则响应“rw”。
hash	返回与 zxid 关联的树摘要的最新历史记录。
gtmk	以十进制格式的 64 位有符号长值形式获取当前跟踪掩码。有关stmk可能值的说明，请参见。
stmk	设置当前跟踪掩码。跟踪掩码是 64 位，其中每一位启用或禁用服务器上特定类别的跟踪日志记录。Log4J 必须首先配置为启用TRACE级别才能查看跟踪日志消息。跟踪掩码的位对应于以下跟踪记录类别。

> 更多四字命令可以参阅官方文档： https://zookeeper.apache.org/doc/current/zookeeperAdmin.html

使用前需要使用 `sudo yum install nc` 安装 nc 命令，使用示例如下：

```
[root@hadoop001 bin]# echo stat | nc localhost 2181
```

```
Zookeeper version: 3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03,
built on 06/29/2018 04:05 GMT
Clients:
 /0:0:0:0:0:0:0:1:50584[1](queued=0,recved=371,sent=371)
 /0:0:0:0:0:0:0:1:50656[0](queued=0,recved=1,sent=0)
Latency min/avg/max: 0/0/19
Received: 372
Sent: 371
Connections: 2
Outstanding: 0
Zxid: 0x150
Mode: standalone
Node count: 167
```

```
[hadoop@node02 ~]$ echo stat | nc localhost 2181
stat is not executed because it is not in the whitelist.
```

- 若如上提示 指令不在白名单中，则需要修改 `$ZK_HOME/conf/zoo.cfg`

```
vim $ZK_HOME/conf/zoo.cfg

# 将需要的命令添加到白名单中
4lw.commands.whitelist=stat, ruok, conf, isro

# 将所有命令添加到白名单中
4lw.commands.whitelist=*
```

- node01修改后将文件同步至其他节点

```
cd $ZK_HOME/conf
scp zoo.cfg node02:$PWD
scp zoo.cfg node03:$PWD
```

- 重启zookeeper

第三章 JavaAPI

```
package zkClient;
/**
 * Project:  BigDataCode
 * Create date:  2023/5/15
 * Created by fujiahao
 */

import org.apache.zookeeper.*;
import org.apache.zookeeper.data.Stat;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.util.List;

/**
 * zookeeper-java-api
```

```
 */
public class zkClient {

	// master:2181、slave1:2181、slave2:2181三个逗号之间不能有空格
	private String connectString = "master:2181,slave1:2181,slave2:2181";
	private int sessionTimeout = 2000;
	// ctrl + alt + f：全局变量zkClient
	private ZooKeeper zkClient;

	// 连接Zookeeper集群
	@Before
	public void init() throws IOException {

		zkClient = new ZooKeeper(connectString, sessionTimeout, new Watcher() {
			@Override
			public void process(WatchedEvent watchedEvent) {
				System.out.println("========================");
				List<String> children = null;
				try {
					children = zkClient.getChildren("/", true);
					for (String child : children) {
						System.out.println(child);
					}
					System.out.println("========================");
				} catch (KeeperException e) {
					e.printStackTrace();
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		});
	}

	// 创建节点
	@Test
	public void create() throws KeeperException, InterruptedException {
		String nodeCreated = zkClient.create("/lwPigKing", "lwPigKing.avi".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
	}

	// 监听节点
	@Test
	public void getChildren() throws KeeperException, InterruptedException {
		List<String> children = zkClient.getChildren("/", true);
		for (String child : children) {
			System.out.println(child);
		}
		// 延时
		Thread.sleep(Long.MAX_VALUE);
	}

	// 判断节点是否存在
	@Test
	public void exist() throws KeeperException, InterruptedException {
		Stat stat = zkClient.exists("/lwPigKing", false);
		System.out.println(stat == null? "not exist" : "exist");
	}
}
```

```
75
```