



Zookeeper分布式系统开发实战 第6课

【声明】 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

- **Dataguru (炼数成金) 是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。**
- **关于逆向收费式网络的详情，请看我们的培训网站 <http://edu.dataguru.cn>**

- 创建zookeeper连接对象时，如何选择哪个服务器进行连接？
 - 客户端的connectstring : localhost:2181,localhost:2182,localhost:2183
 - 通过类org.apache.zookeeper.client. StaticHostProvider维护地址列表
 - 通过解析connectstring后，进行随机排序，形成最终的地址列表
 - 每次从形成的地址列表中选择第一个地址进行连接，如果连接不上再选择第二个地址
 - 当如果当前节点是列表的最后一个节点，则再重新选择第一个节点，相当于一个环
 - 通过随机排序，每个zk的客户端就会随机的去连接zk服务器，分布相对均匀
 - 举例：connectstring为：192.168.1.2:2181, 192.168.1.3:2181, 192.168.1.4:2181
 - 随机打乱后的顺序为：192.168.1.3:2181, 192.168.1.2:2181, 192.168.1.4:2181
 - 则第一次连接时选择1.3这个节点，如果连接不上，则重新选择1.2，然后是1.4；如果1.4还是连接不上则开始连接第一个节点1.3

第五讲 客户端连接

初始化连接地址

```
public StaticHostProvider(Collection<InetSocketAddress> serverAddresses)
    throws UnknownHostException {
    for (InetSocketAddress address : serverAddresses) {
        InetAddress ia = address.getAddress();
        InetAddress resolvedAddresses[] = InetAddress.getAllByName((ia!=null) ? ia.getHostAddress() : address.getHostName());
        for (InetAddress resolvedAddress : resolvedAddresses) {
            // If hostName is null but the address is not, we can tell that
            // the hostName is an literal IP address. Then we can set the host string as the
            // safely to avoid reverse DNS lookup.
            // As far as i know, the only way to check if the hostName is null is use toString()
            // Both the two implementations of InetAddress are final class, so we can trust the
            // the toString() method.
            if (resolvedAddress.toString().startsWith("/")
                && resolvedAddress.getAddress() != null) {
                this.serverAddresses.add(
                    new InetSocketAddress(InetAddress.getByAddress(
                        address.getHostAddress(),
                        resolvedAddress.getAddress()),
                        address.getPort()));
            } else {
                this.serverAddresses.add(new InetSocketAddress(resolvedAddress.getHostAddress(),
                    address.getPort()));
            }
        }
    }

    if (this.serverAddresses.isEmpty()) {
        throw new IllegalArgumentException(
            "A HostProvider may not be empty!");
    }
    Collections.shuffle(this.serverAddresses);
}
```

随机排序

选择一个地址:

```
public InetAddress next(long spinDelay) {
    ++currentIndex;
    //如果已经是最后一个节点，则从第一个节点开始
    if (currentIndex == serverAddresses.size()) {
        currentIndex = 0; //currentIndex表示遍历到的地址数值下表
    }
    //当地址列表只有一个地址时，再次获取之前先sleep一定时间再返回，这算是一个重试间隔时间
    if (currentIndex == lastIndex && spinDelay > 0) {
        try {
            Thread.sleep(spinDelay);
        } catch (InterruptedException e) {
            LOG.warn("Unexpected exception", e);
        }
    } else if (lastIndex == -1) {
        // We don't want to sleep on the first ever connect attempt.
        lastIndex = 0; //lastIndex表示当前用到的连接地址的数值下标
    }

    return serverAddresses.get(currentIndex);
}
```

返回下一个节点的地址

■ 什么是会话

- 代表客户端与服务器端的一个zk连接
- 底层通信是通过tcp协议进行连接通信
- Zookeeper会在服务器端创建一个会话对象来维护这个连接的属性
- 当网络出现断网的抖动现象的时候，并不代表会话一定断开，细节后面讲超时时会再详细讲
- 会话对象的实现是SessionImpl，包括以下四个属性
 - sessionId：唯一标识一个会话，具备全局唯一性
 - Timeout：会话超时时间，创建客户端zookeeper对象时传入，服务器会根据最小会话时间和最大会话时间的规定来明确此值具体是什么
 - Ticktime：下次会话超时的时间，与“分桶策略”有关，后面会讲
 - isClosing：标记一个会话是否已经被关闭，当服务器端检测到有会话失效时，就会把此会话标记为已关闭

■ 会话状态

- CONNECTING
- CONNECTED
- RECONNECTING
- RECONNECTED
- CLOSE
 - 服务器端判断会话超时后，在进行会话清除之前，会把会话的状态置为CLOSE

■ sessionTracker

- 服务器端通过此类来管理会话，包括会话的创建 管理和清除工作
- 通过三个数据结构从三个维度来管理会话
 - sessionsById属性:用于根据sessionID来查找session
 - sessionsWithTimeout属性：通过sessionID来查找此session的失效时间是什么时候
 - sessionSets属性：通过某个时间查询都有哪些会话在这个时间点会失效

```
public class SessionTrackerImpl extends Thread implements SessionTracker {  
    private static final Logger LOG = LoggerFactory.getLogger(SessionTrackerImpl.class);  
  
    HashMap<Long, SessionImpl> sessionsById = new HashMap<Long, SessionImpl>();  
  
    HashMap<Long, SessionSet> sessionSets = new HashMap<Long, SessionSet>();  
  
    ConcurrentHashMap<Long, Integer> sessionsWithTimeout;
```

如何检查会话是否失效

如何高效的检测和清除失效会话

■ 分桶策略

- 把所有的会话按照时间维度进行分类管理，即同一个时间点失效的会话都在一起管理，即上面的属性

`ConcurrentHashMap<Long, Integer> sessionsWithTimeout`

- 会话失效时间计算

- 约定：把所有的时间按照某个单位进行等份（默认为服务器的ticktime配置）切割，此单位称呼为ExpirationInterval
- 公式：某次超时时间 = $((currentTime + sessiontimeout) / ExpirationInterval + 1) \times ExpirationInterval$

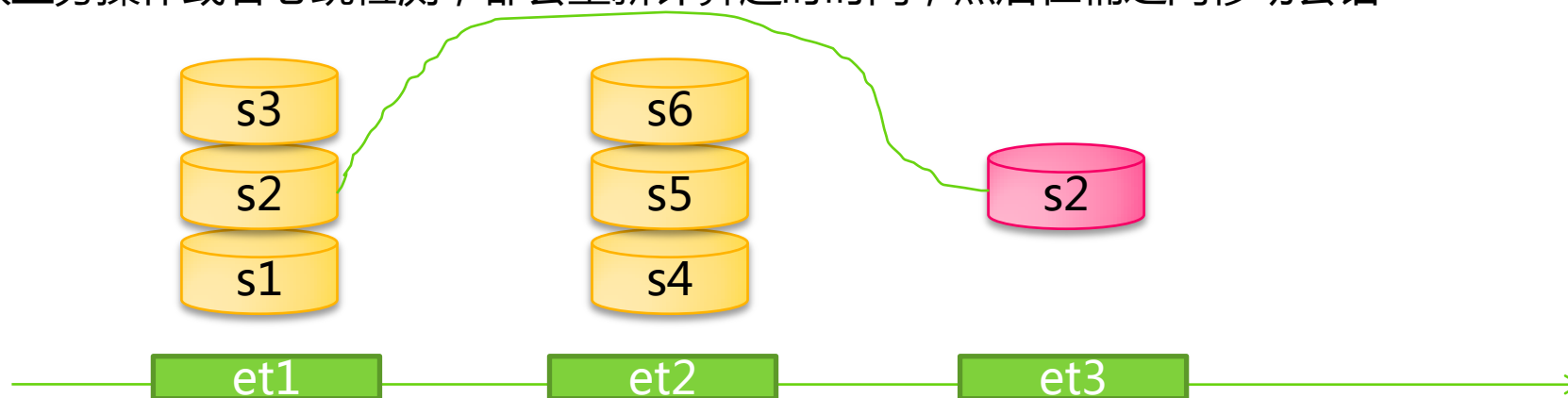
- 举例说明：

- 由于服务器的ticktime的默认值是2000ms，则ExpirationInterval=2000ms
- 第一次创建会话时，currenttime=1370907000000
- 创建会话时，客户端传入的超时时间是15000ms
- 则，此会话超时时间为 $((1370907000000 + 15000) / 2000 + 1) \times 2000 = 1370907016000$

- 当某个会话由于有操作而导致超时时间变化，则会把会话从上一个桶移动到下一个桶

■ 会话激活

- 当此会话下一直有操作，则会话就不会失效
- 影响会话超时时间的因素
 - 心跳检测，及PING命令
 - 当客户端发现在 $\text{sessionTimeout}/3$ 时间范围内还没有任何操作命令产生，即会发送一个PING心跳请求
 - 正常业务操作，比如get 或者set
- 每次业务操作或者心跳检测，都会重新计算超时时间，然后在桶之间移动会话



■ 会话超时检测

- 由sessionTracker中的一个线程负责检查session是否失效
- 线程检查周期也是ExpirationInterval的倍数
- 当某次检查时，如果在此次的分桶（即前面的ExpirationInterval）之前还有会话，就说明这些会话都超时了，因为会话如果有业务操作或者靠心跳，会不断的从交小的分桶迁移到大的分桶
- 举例：系统启动时的时间是100001，此时ExpirationInterval=2000ms，则桶的刻度为 $100001/2000=50$ 下一次的检查时间为 $(100001/2000+1) \times 2000=102000$

■ 会话清理流程

- 修改会话状态为close
 - 由于清理过程需要一定时间，为了避免清理期间会话状态出现变化
- 向所有的集群节点发起会话关闭请求
- 收集跟被清理的会话相关的临时节点
- 向集群节点发出删除临时节点的事务请求
- 集群中的所有节点执行删除临时节点事务
- 从sessionTracker的列表中移除会话
- 关闭会话的网络连接，具体类是NIOServerCnxnFactory

■ 会话重连

- 当客户端与服务器端的网络断开后，客户端会不断的重新连接，当连接上后会话的状态是以下两种情况
 - CONNECTED //服务器端会话依然存在
 - EXPIRED 即服务器端的会话已经被关闭清除了
- 注意：网络断开并不代表会话超时

■ 三个会话异常

- CONNECTION_LOSS
- SESSION_EXPIRED
- SESSION_MOVE



■ CONNECTION_LOSS

- 网络闪断导致或者是客户端服务器出现问题导致
- 出现此问题，客户端会重新查找地址进行连接（connectstring），直到连接上
- 当做某个操作过程（比如setData）中出现了CONNECTION_LOSS现象，则客户端会接收到None-Disconnected通知（设置了默认watcher情况下），同时会抛出异常
org.apache.zookeeper KeeperException\$ConnectionLossException
- 当重新连接上后，客户端会收到事件通知（None-SyncConnected）//在设置了默认watcher时

■ SESSION_EXPIRED

- 通常发生在CONNECTION_LOSS期间，因为没有网络连接，就不能有操作和心跳进行，会话就会超时
- 由于重新连接时间较长，导致服务器端关闭了会话，并清除会话，此时会话相关联的watcher等数据都会丢失，watcher失效
- 出现这种情况，客户端需要重新创建zookeeper对象，并且恢复数据（比如注册watcher）
- 会收到异常SessionExpiredException

第五讲 会话—会话管理

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    WatcherExample we = new WatcherExample();  
    WatcherRegister wr = new WatcherRegister("192.168.1.103:2181",we);  
    try {  
        wr.testWatcherdisabled("/node2");  
        Thread.sleep(30000);  
        System.out.println("-----");  
        wr.getData();  
        Thread.sleep(30000);  
        System.out.println("-----111111----");  
        wr.getData();  
    } catch (KeeperException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    } catch (InterruptedException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
    try {  
        Thread.sleep(300000000);  
    } catch (InterruptedException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

- 1.测试watcher是否正常
- 2.Sleep期间断开网络连接
- 3.断开网络连接期间获取数据
- 4.重新连接网络
- 5.再次获取数据

第五讲 会话—会话管理

<terminated> WatcherRegister [Java Application] D:\Program Files\Java\jdk1.7.0_67\bin\javaw.exe (2015年12月20日 下午5:22:44)

```
watcher=com.zk.example.watcher.WatcherExample
path=null
eventType=None
SyncConnected
watcher=com.zk.example.watcher.WatcherExample1
path=/node2
eventType=NodeDataChanged
watcher=com.zk.example.watcher.WatcherExample1
path=null
eventType=None
watcher=com.zk.example.watcher.WatcherExample
path=null
eventType=None|
Disconnected
```

1.连接上后的输出

2.测试watcher，在其它命令窗口修改数据节点

3.端口网络连接

```
org.apache.zookeeperKeeperException$ConnectionLossException: KeeperErrorCode = ConnectionLoss for /node
    at org.apache.zookeeper.KeeperException.create(KeeperException.java:99)
    at org.apache.zookeeper.KeeperException.create(KeeperException.java:51)
    at org.apache.zookeeper.ZooKeeper.getData(ZooKeeper.java:1155)
    at com.zk.example.watcher.WatcherRegister.getData(WatcherRegister.java:28)
    at com.zk.example.watcher.WatcherRegister.main(WatcherRegister.java:38)
```

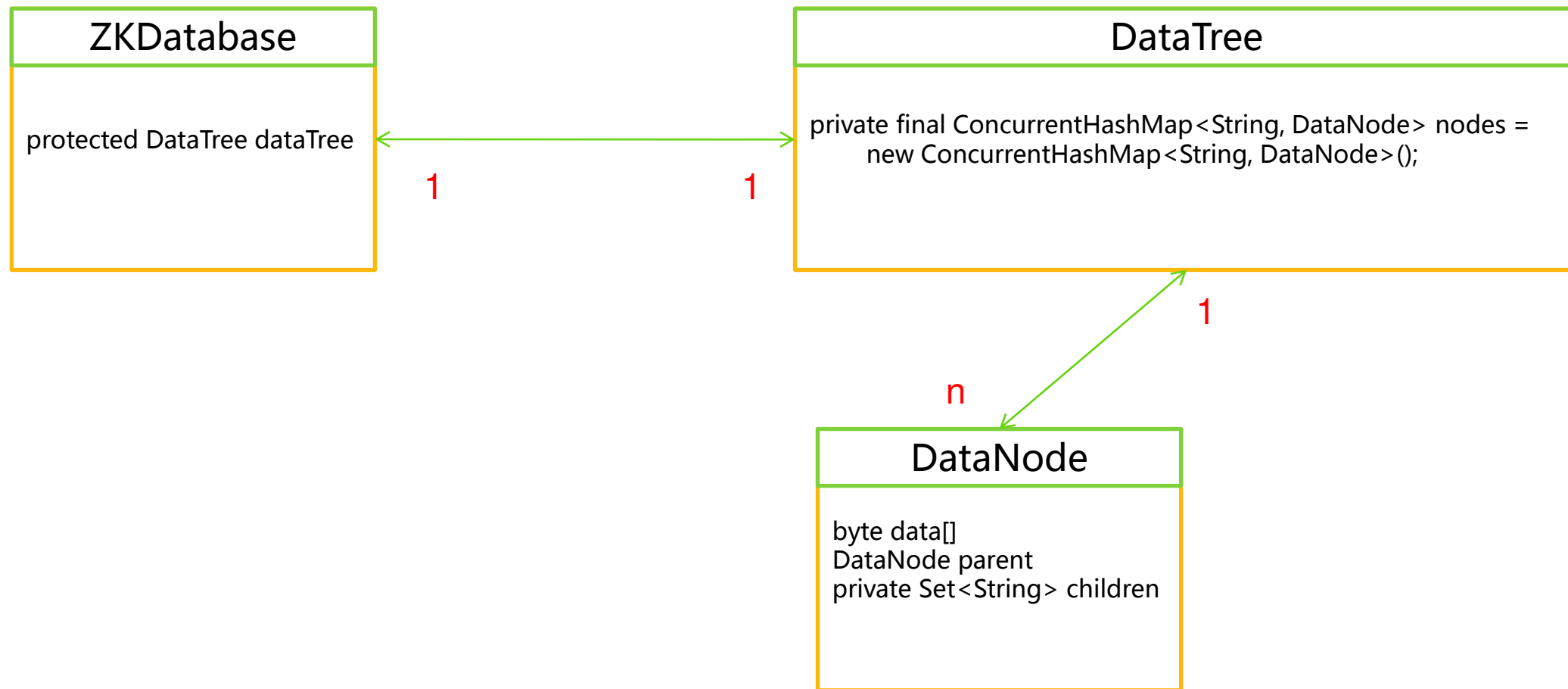
4.调用获取数据接口

```
watcher=com.zk.example.watcher.WatcherExample1
path=null
eventType=None
watcher=com.zk.example.watcher.WatcherExample
path=null
eventType=None
Expired
```

5.经过一段时间，重新打开网络连接后

■ SESSION_MOVED

- 出现CONNECTION_LOSS时，客户端尝试重新连接下个节点（connectstring）
- 例如：客户端刚开始连接的是s1，由于网络中断，客户端尝试连接s2，连接成功后，s2延续了会话，即会话从S1迁移到S2
- 当出现以下业务场景时，服务器端会抛出SessionMovedException异常，由于客户端的连接已经发生了变化（从s1-->s2）所以，客户端收不到异常
 - 有三台服务器s1 s2 s3
 - 开始时，客户端连接s1，此时客户端发出一个修改数据的请求r1
 - 在修改数据的请求到达s1之前，客户端重新连接上了s2服务器，此时出现了会话转移
 - 连接s2后，客户端又发起一次数据修改请求r2
 - r1被s1服务器处理，r2被s2处理（比r1被处理要早）
 - 这样对于客户端来说，请求被处理两次，并且r2被r1的处理结果覆盖
 - 服务器端通过检查会话的所有者来判断此次会话请求是否合法，不合法就抛出moved异常



■ ZKDatabase

- 负责管理zk的所有会话 datatree存储和事务日志
- 会定时向磁盘快照内存数据
- 当节点启动后，会通过磁盘上的事务日志和快照文件恢复完整的内存数据

■ DataTree

- 整个zk的数据就靠DataTree维护，包括数据，目录，权限
- 数据的领域模型，不包括对外的连接

■ DataNode

- 树形结构中的每个节点
- 引用到父节点
- 也引用到子节点列表

第五讲 数据与存储—事务日志

■ 日志文件

- 存储于datalog或者datalogDir配置的目录
- 对应目录下的version-2代表的是日志格式版本号
- 日志文件命名
 - 文件大小都是64m
 - 后缀都是16进制格式数字，逐渐增大，其本质是本日志文件的第一条zxid

log.200000001	2015/11/30 22:00	200000001 文件	65,537 KB
log.400000001	2015/12/1 23:24	400000001 文件	65,537 KB
log.500000001	2015/12/3 19:13	500000001 文件	65,537 KB
log.600000001	2015/12/4 18:03	600000001 文件	65,537 KB
log.700000001	2015/12/6 16:05	700000001 文件	65,537 KB
log.800000001	2015/12/8 14:45	800000001 文件	65,537 KB
log.900000001	2015/12/9 12:12	900000001 文件	65,537 KB
log.a00000001	2015/12/11 11:46	A00000001 文件	65,537 KB
log.b00000001	2015/12/20 17:23	B00000001 文件	65,537 KB

第五讲 数据与存储—事务日志

■ 日志格式

- Zk提供了工具类org.apache.zookeeper.server.LogFormatter解析日志的内容
- 第一行是日志格式信息

- ZooKeeper Transactional Log File with dbid 0 txnlog format version 2

```
ZooKeeper Transactional Log File with dbid 0 txnlog format version 2
15-12-8 上午10时31分20秒 session 0x15172ac80e90000 cxid 0x0 zxid 0x800000001 closeSession null
15-12-8 上午10时34分00秒 session 0x1517f6d52040000 cxid 0x0 zxid 0x800000002 createSession 30000

15-12-8 上午10时39分47秒 session 0x1517f6d52040001 cxid 0x0 zxid 0x800000003 createSession 10000

15-12-8 上午10时40分00秒 session 0x1517f6d52040001 cxid 0x0 zxid 0x800000004 closeSession null
15-12-8 上午10时40分13秒 session 0x1517f6d52040000 cxid 0x2 zxid 0x800000005 create
'/node2,#6e6f64653264617461,v{s{31,s{'world','anyone'}}},F,4

15-12-8 上午10时40分25秒 session 0x1517f6d52040002 cxid 0x0 zxid 0x800000006 createSession 10000

15-12-8 上午10时41分15秒 session 0x1517f6d52040000 cxid 0x3 zxid 0x800000007 setData
'/node2,#6e6f6465326461746131,1

15-12-8 上午10时41分19秒 session 0x1517f6d52040000 cxid 0x4 zxid 0x800000008 setData
'/node2,#6e6f6465326461746131,2
```

第6行从作到右代表事务操作时间、客户端会话id、cxid、zxid、操作类型、节点路径、节点数据内容、acl信息、是否临时节点、父节点的子节点版本号
F代表持久 T代表临时











■ 日志写入

- Zk通过类org.apache.zookeeper.server.persistence. FileTxnLog实现对事务日志的管理
 - 通过append方法来添加事务日志
- 写入过程
 - 确定是否有事务日志文件可写，当第一次创建事务日志文件或者上一个事务日志文件写满后都会关闭这个文件流
 - 确定事务日志是否需要扩容，当文件剩余空间不足4KB时，把文件新增64MB（新增一个日志文件），用“0”填充空余的空间
 - 事务序列化
 - 生成Checksum
 - 写入事务日志文件流
 - 事务日志刷入磁盘，本质是调用系统的fsync接口

第五讲 数据与存储—事务日志

■ 数据快照

- Zk某个时刻的完整数据
- 快照文件的后缀为服务器最新的zxid
- 通过工具类SnapshotFormatter可以查看快照文件的文件内容

 snapshot.60000002c	2015/12/5 23:04	60000002C 文件	2 KB
 snapshot.90000000c	2015/12/10 18:26	90000000C 文件	3 KB
 snapshot.100000003	2015/11/30 21:56	100000003 文件	1 KB
 snapshot.200000002	2015/11/30 22:00	200000002 文件	1 KB
 snapshot.300000001	2015/12/1 20:45	300000001 文件	1 KB
 snapshot.400000011	2015/12/3 15:10	400000011 文件	1 KB
 snapshot.500000010	2015/12/4 9:33	500000010 文件	1 KB
 snapshot.700000042	2015/12/8 10:30	700000042 文件	3 KB
 snapshot.800000036	2015/12/9 11:51	800000036 文件	3 KB
 snapshot.a00000004	2015/12/20 11:24	A00000004 文件	3 KB

■ 快照流程

- 确定是否需要进行数据快照
 - snapCount 默认为100000，表示达到这个数量的事务日志才开始进行快照
 - 为了避免集群节点同时进行快照，按照如下工时触发快照
$$\logCount > (\text{snapCount}/2 + \text{randRoll})$$
//randRoll是为1---snapCount/2之间的随机数
- 切换事务日志文件
 - 创建新的事务日志文件
- 创建数据快照异步线程
- 获取全量数据和会话信息
- 生成快照数据文件
- 把数据刷入快照文件

Thanks

FAQ时间