Zookeeper是什么？

1. 是一个针对大型分布式系统的可靠协调系统
2. 提供的功能包括：配置维护、名字服务、分布式同步、组服务。
3. 目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户；
4. Zookeeper已经成为Hadoop生态系统中重要的组件。

特点：

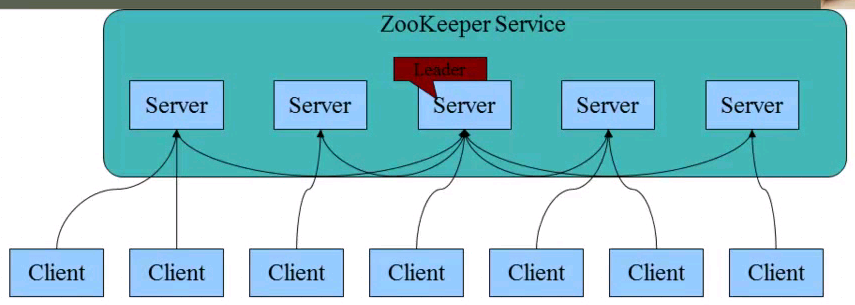
1. 最终一致性：为客户端展示同一视图，这是Zookeeper最重要的性能。
2. 可靠性：如果消息被送到一台服务器接收，那么它将被所有的服务器接受。
3. 实时性：Zookeeper不能保证两个客户端能同时得到刚更新的数据，如果需要更新数据，应该在读数据之前调用sync()接口。
4. 等待无关（wait-free）:慢的或者失效的client不干预快速的client请求。
5. 原子性：更新只能成功或者失败，没有中间状态。
6. 顺序性：所有Server，统一消息发布顺序一致。

使用：

1. HDFS
2. YARN
3. Storm
4. Hbase
5. Flume
6. Dubbo（阿里）：RPC框架
7. Metaq(阿里)：消息队列

架构：

1. 每个Server在内存中存储了一份数据
2. Zookeeper启动时，将从实例中选举一个leader（Paxos）
3. Leader负责处理数据更新操作（Zab协议）
4. 一个更新操作成功，当仅当大多数Server在内存中成功修改数据。



各种角色:

1. Leader:领导者负责进行投票的发起和决议，更新系统状态。
2. 学习者：
   1. 跟随者：Follower用于接收客户端请求并向客户端返回结果，在选主过程中参与投票。
   2. 观察者：Observer可以接收客户端连接，将写请求转发给leader节点，但Observer不参与投票过程，只同步leader的状态。Observer的目的只是为了扩展系统，提高读取速度。
3. 客户端：请求发起方。

3.3版本新增Observer角色：

1、 Zookeeper需要保证高可用和强一致性

2、 为了支持更多的客户端，需要增加更多的Server；

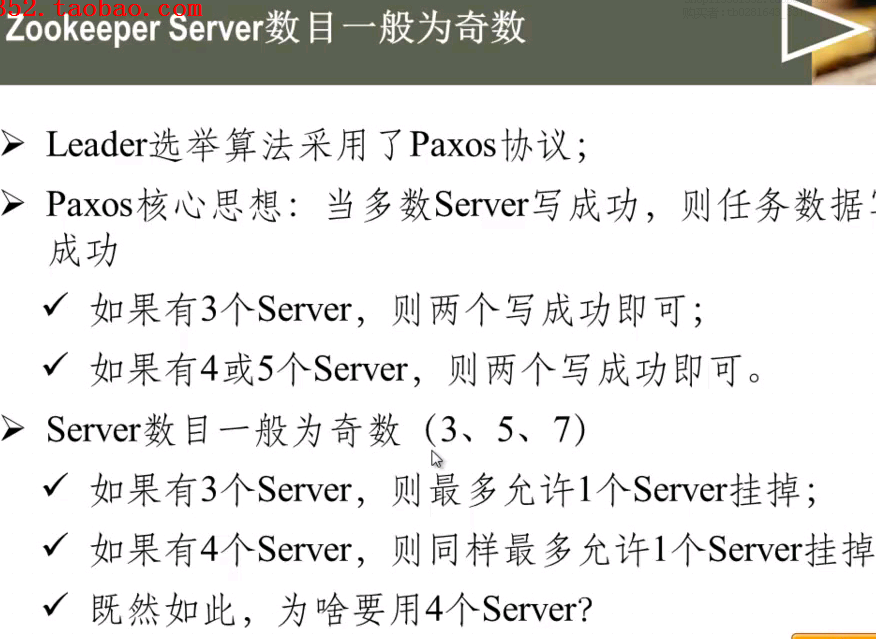
3、 Server增多，投票阶段延迟增大，影响性能；

4、 权衡伸缩性和高吞吐率，引入Observer

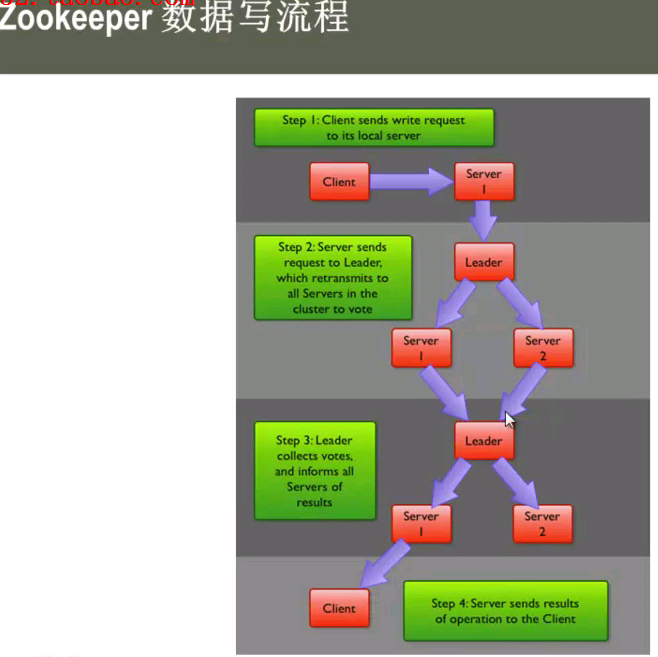
1、Observer不参与投票；

2、Observer接收客户端的连接，并将写请求转发给leader节点；

3、加入更多Observer节点，提高伸缩性，同时不影响吞吐率。

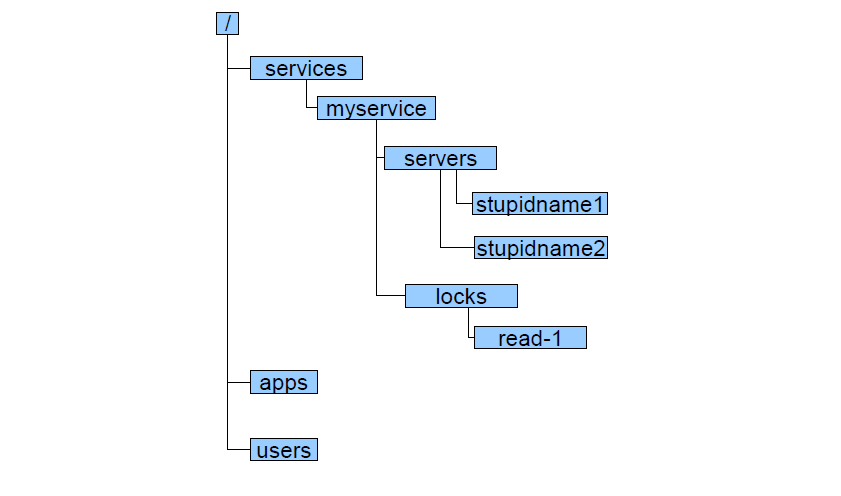


Zookeeper写数据：

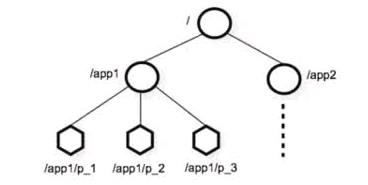
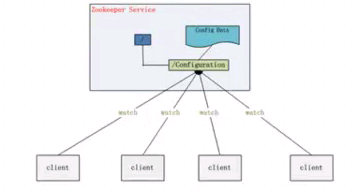


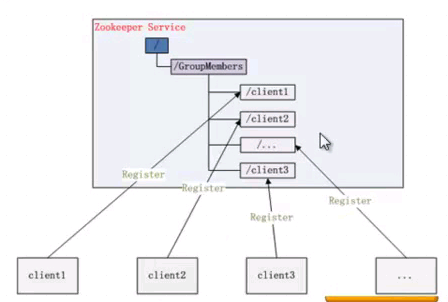
Zookeeper数据模型：

1. 层次化的数据模型；
2. 每个节点在Zookeeper中叫做znode，并且其有一个唯一的路径标识；
3. 节点Znode可以包含数据和子节点（EPHEMERAL类型的节点不可能有子节点）；
4. Znode中的数据可以有多个版本，比如某个路径下存储多个数据版本，那么查询这个路径下的数据需要带上版本；
5. 客户端应用可以在节点上设置监视器（Watcher）；
6. 节点不支持部分读写，而是一次性完整读写。
7. Znode有两种类型，短暂的（ephemeral）和持久的（persistent）；
8. Znode的类型在创建时确定并且之后不能再修改；
9. 短暂的Znode的客户端回话结束时，Zookeeper会将短暂Znode删除，短暂不可以有子节点；
10. 持久Znode不依赖于客户端会话，只有当客户端明确要删除持久Znode时才会被删除；
11. Znode有四种形式的目录节点，PERSISTENT、PERSISTENT\_SEQUENTIAL、EPHEMERAL、EPHEMERAL\_SEQUENTIAL。



统一命名服务：

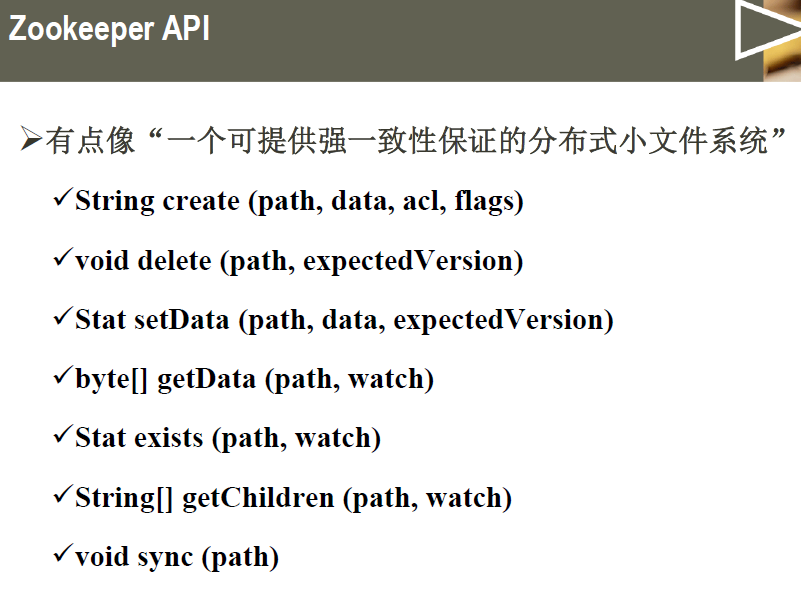
1. 分布式环境下，经常需要对应用/服务进行统一命名，便于识别不同服务；
   1. 类似于域名与IP之间的对应关系，域名容易记住；
   2. 通过名称来获取资源或者服务的地址，提供者等信息；
2. 按照层次结构组织服务/应用名称
   1. 可将服务名称以及地址写到Zookeeper上，客户端通过Zookeeper获取可用服务列表；
   2. 
3. 配置管理：
   1. 分布式环境下，配置文件管理和同步是一个常见问题；
      1. 一个集群中，所有节点的配置文件信息是一致的，比如hadoop；
      2. 对配置文件修改后，希望能够快速同步到各个节点上；
   2. 配置管理可提交Zookeeper实现；
      1. 可将配置信息写入Zookeeper的一个Znode上；
      2. 各个节点监听这个Znode
      3. 一旦Znode中的数据被修改，Zookeeper将通知各个节点
      4. 
4. 集群管理
   1. 分布式环境中，实时掌握每个节点的状态是必要的；
      1. 可根据节点实时状态做出一些调整；
   2. 可交由Zookeeper实现：
      1. 可将节点信息写入Zookeeper的一个Znode上；
      2. 监听这个Znode可获取它的实时状态变化
   3. 典型应用
      1. Hbase中的Master状态监听与选举。



1. 分布式通知/协调
   1. 分布式环境中，经常存在一个服务需要知道它所管理的自服务的状态；
      1. NN需要知道哥哥DN的状态
      2. JobTracker需要知道各个TaskTracker的状态。
   2. 心跳检测机制可通过Zookeeper实现；
   3. 信息推送可由Zookeeper实现（发布/订阅模式）
2. 分布式锁
   1. Zookeeper是强一致性的：
      1. 多个客户端同时在Zookeeper上创建相同的Znode，只有一个创建成功。（同时写的话，只有一个能得到锁）
   2. 实现所的独占性
      1. 多个客户端同时在Zookeeper上创建相同的Znode，创建成功的那个客户端得到锁，其他客户端等待。
   3. 控制锁的时序
      1. 各个客户端在某个Znode下创建临时Znode（类型为CreateMode.EPHEMERAL\_SEQUENTIAL），这样，该znode可掌握全局访问时序。（对Znode编码）
3. 分布式队列：
   1. 两种队列
      1. 当一个队列的成员都聚齐时，这个队列才可用，否则一致等到所有成员到达，这种事同步队列。
      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式（通过分布式锁实现）
   2. 同步队列
      1. 一个job由多个task组成，只有所有任务完成后，job才运行完成。
      2. 可为job创建一个/job目录，然后在该目录下，为每个完成的task创建一个临时Znode，一旦临时节点数目到达task总数，则job运行完成。







Zookeeper Watcher：

1. Watcher在Zookeeper是一个核心功能
   1. 可以监控目录节点的数据变化以及子目录的变化；
   2. 一旦状态发生变化，服务器就会通知所有设置在这个目录节点上的watcher
2. 基本特点
   1. 一次设置对应一次处罚。
   2. 异步触发。
   3. 顺序触发
3. 可以设置观察的操作：exists getChildren getData
4. 可以触发观察的操作：create delete setData

