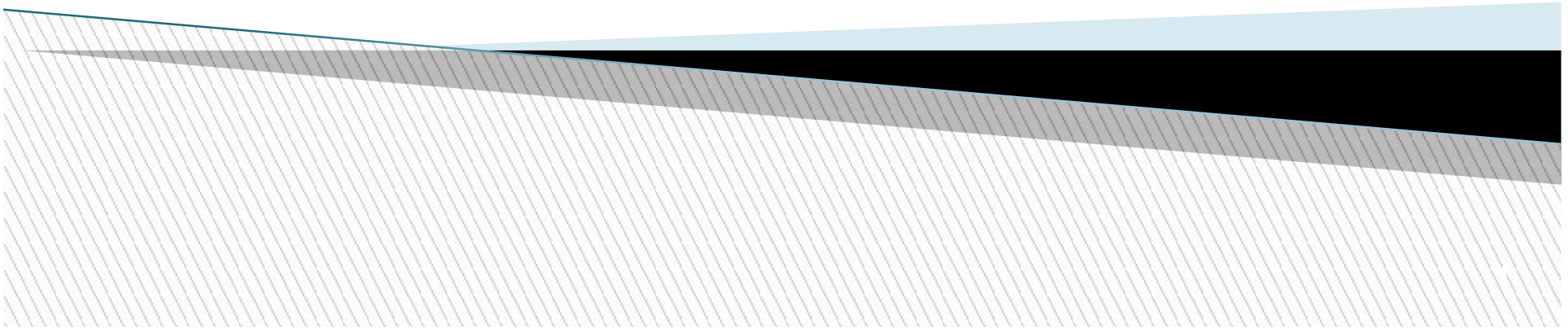
ZooKeeper原理与实践

主讲人：黄振龙

2014年11月2日



主要内容

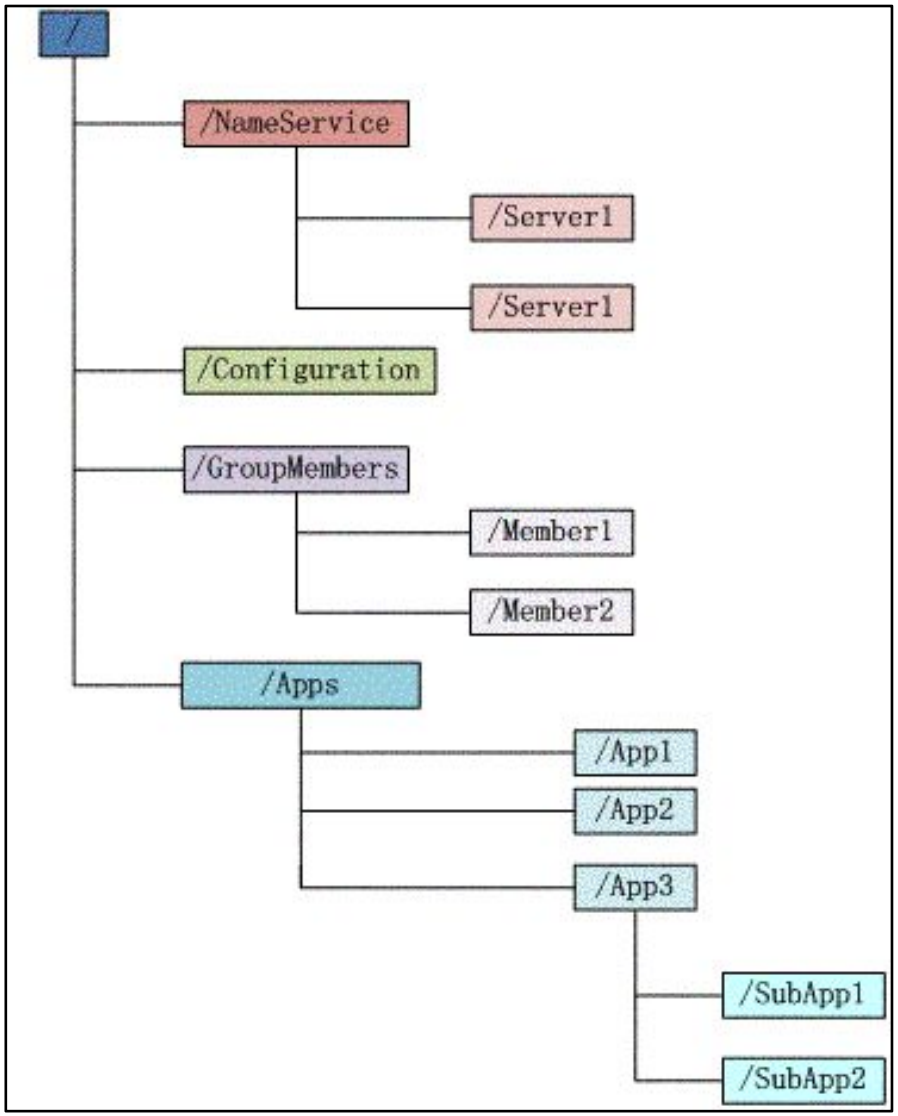
- ZooKeeper简介
- Leader选举算法
- 数据一致性协议：Zab
- ZooKeeper实践
  - Session机制
  - Watcher机制

ZooKeeper简介

- ZooKeeper是Hadoop的一个子项目，它是一个针对大型分布式系统的可靠协调系统。
- Zookeeper是Google的Chubby一个开源的实现。
- 由Yahoo捐献
- 应用：Hadoop、Hbase、Solr、Kafka...

数据模型 – 节点

- 类似Unix的数据视图
- Znode
  - data
  - stat
  - children



数据模型 – 节点类型

- 持久节点（Persisient Node）
- 临时节点（Ephemeral Node）
- 时序节点（Sequential Node）
  - Perisient Sequential Node
  - Ephemeral Sequential Node

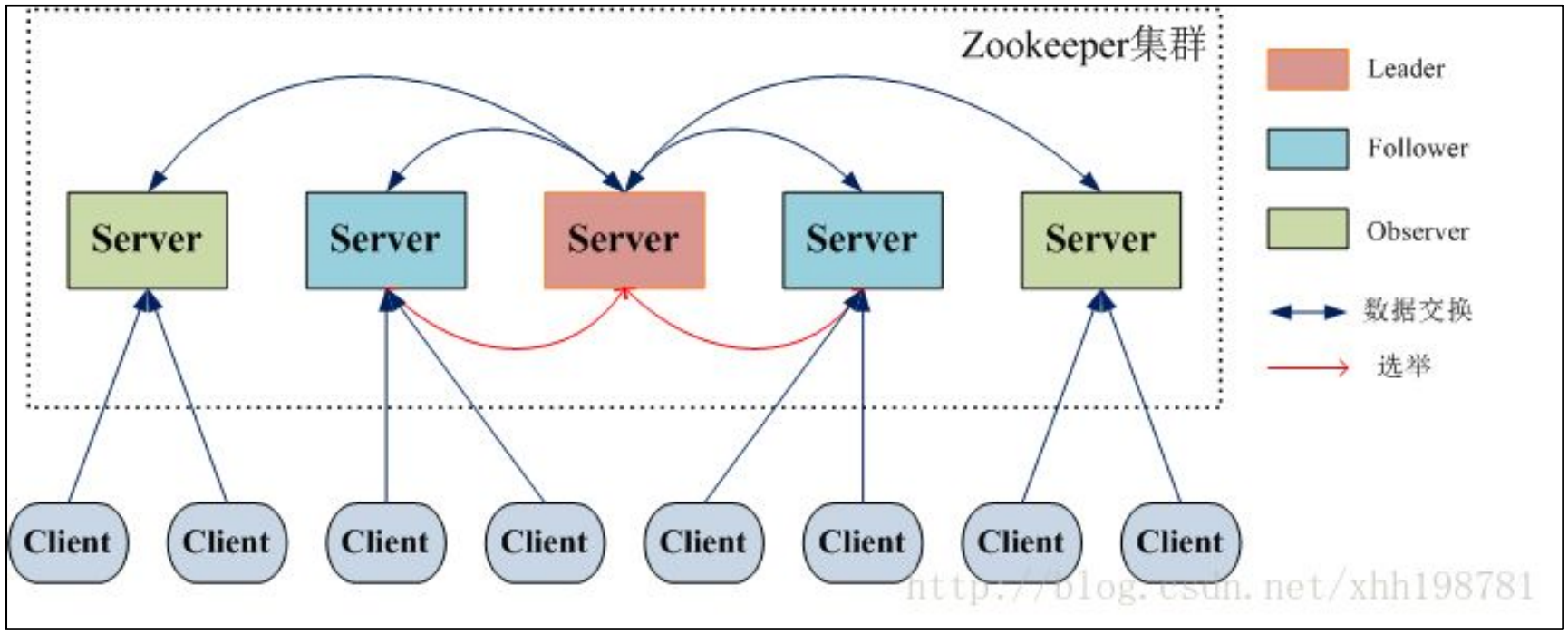
数据模型 - 节点信息

czxid	创建这个znode的zxid
mzxid	最后一次修改这个znode的zxid
ctime	该znode创建时间
mtime	该znode最后一次修改的时间
version	该znode的version，也就是该znode的修改次数。
cversion	该znode的子节点的version
aversion	该znode的ACL信息version
ephemeralOwner	如果该znode是ephemeral node，此字段就是对应client的session；否则为0。
dataLength	The length of the data field of this znode.
numChildren	子节点个数

数据模型

- Demo

集群拓扑


Zookeeper集群
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client
Leader
Follower
Observer
数据交换
选举
http://blog.csdn.net/xhh198781

Leader选举

- LeaderElection选举算法
- FastLeaderElection选举算法
- AuthFastLeaderElection选举算法

FastLeaderElection

- 数据恢复阶段
  每个ZooKeeper Server读取当前磁盘的数据获取最大的zxid。

- 发送选票
  选票包括以下4部分数据：
  - 所推举的Leader id
  - 本机的最大zxid值
  - LogicalClock
  - 本机的所处状态： LOOKING，FOLLOWING，OBSERVING，LEADING

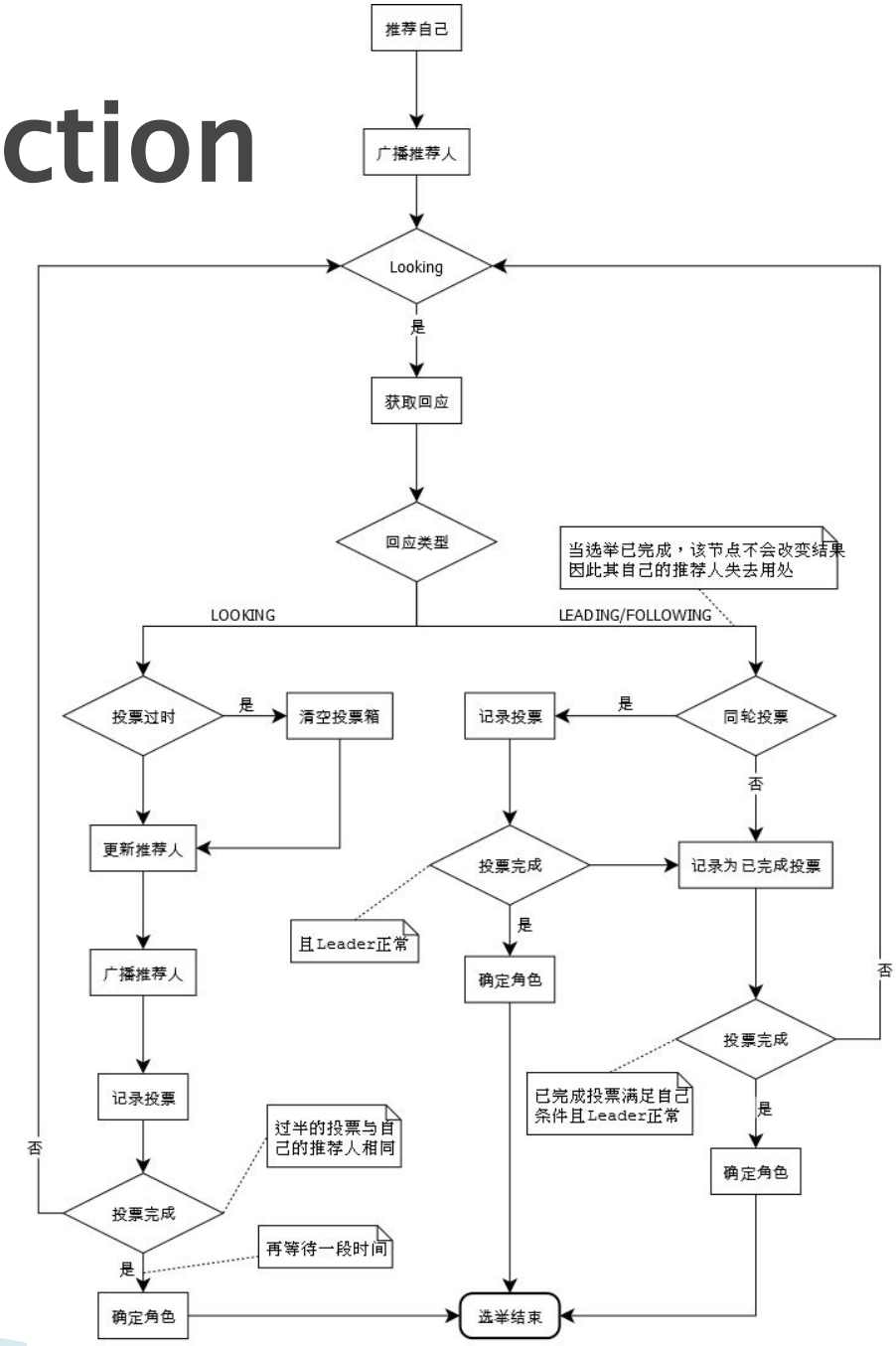
FastLeaderElection

- **处理选票 – 如果Sender的状态是LOOKING**
  - If sender.logicalclock > receiver.logicalclock:
    - receiver.logicalclock = sender.logicalclock
    - 清空recvset
    - 更新选举
  - If sender.logicalclock < receiver.logicalclock:
    - Ignore该选票
    - 发送本机选票
  - If sender.logicalclock = receiver.logicalclock:
    - 更新选票，然后广播选票

FastLeaderElection

- **处理选票 –如果Sender的状态是FOLLOWING或者LEADING**
  - 如果LogicalClock相同，将数据保存到recvset，如果Sender宣称自己是Leader，那么判断是不是半数以上的服务器都选举它，如果是设置角色并退出选举。
  - 否则，这是一条与当前LogicalClock不符合的消息，说明在另一个选举过程中已经有了选举结果，于是将该选举结果加入到OutOfElection集合中，根据OutOfElection来判断是否可以结束选举，如果可以也是保存LogicalClock，更新角色，退出选举。
- **更新Server状态（角色）**

FastLeaderElection



Leader选举示例

- 服务器1启动,此时只有它一台服务器启动了,它发出去的数据没有任何响应,所以它的选举状态一直是LOOKING状态.

- 服务器2启动,它与最开始启动的服务器1进行通信,互相交换自己的选举结果,由于两者都没有历史数据,所以id值较大的服务器2胜出,但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3),所以服务器1,2还是继续保持LOOKING状态.

- 服务器3启动,根据前面的理论分析,服务器3成为服务器1,2,3中的Leader,而与上面不同的是,此时有三台服务器选举了它,所以它成为了这次选举的Leader.

- 服务器4启动,根据前面的分析,理论上服务器4应该是服务器1,2,3,4中最大的,但是由于前面已经有半数以上的服务器选举了服务器3,所以它只能是Follower.

- 服务器5启动,同4一样,Follower.

数据一致性

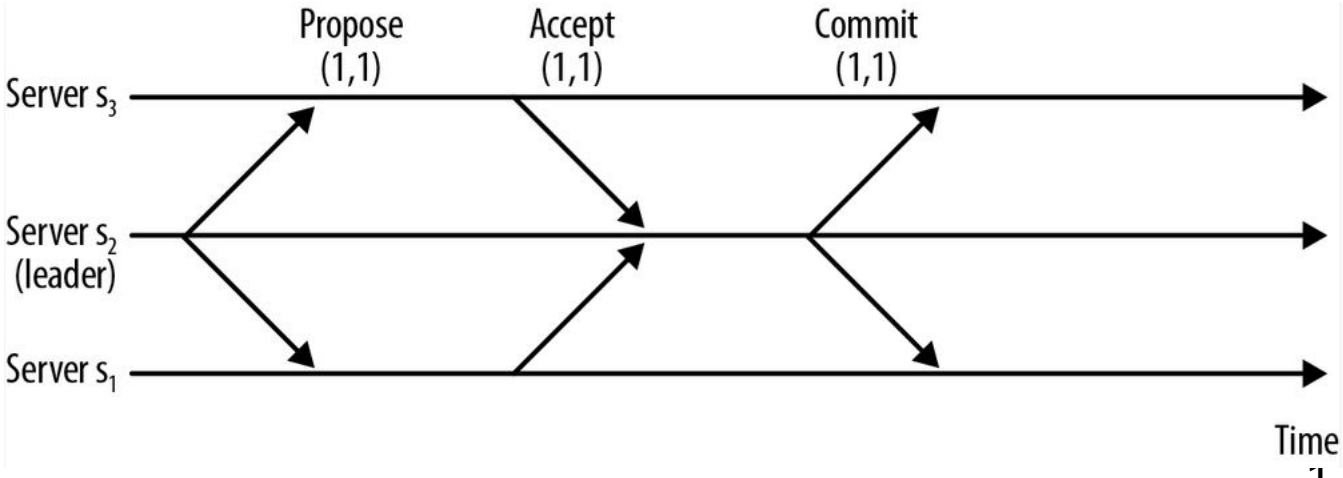
- 为了保证一致性，Zookeeper提出了两个安全属性：
  - 全序（Total Order）：如果消息A在消息B之前发送，则所有Server应该看到相同结果。
  - 因果顺序（Causal Order）：如果消息A在消息B之前发生（A导致了B），并且一起发送，则消息A始终在消息B之前被执行。
- 为了保证上述两个安全属性，Zookeeper使用了TCP协议和Leader。

Zab协议

- Zab（Zookeeper Atomic Broadcast）
- Zab协议是Paxos协议的一种变形。
- 补充：目前Paxos还没有一种分布式系统是基于标准的Paxos协议完成的。

ZAB协议

- Leader执行写操作可以简化为一个两段式提交的transaction：
  - Leader发送proposal给所有的Follower。
  - 收到proposal后，Follower回复ACK给Leader，接受Leader的proposal.
  - 当Leader收到大多数的Follower的ACK后，将commit其proposal。



Session机制

- 在ZooKeeper中，客户端和服务端建立连接后，会话随之建立，生成一个全局唯一的会话ID(Session ID)。针对session可以有保存一些关联数据（EphemeralNode）。

- 连接断开
  - ConnectionLoss
  - SessionExpired

- 实例：Alive Nodes

Watcher机制

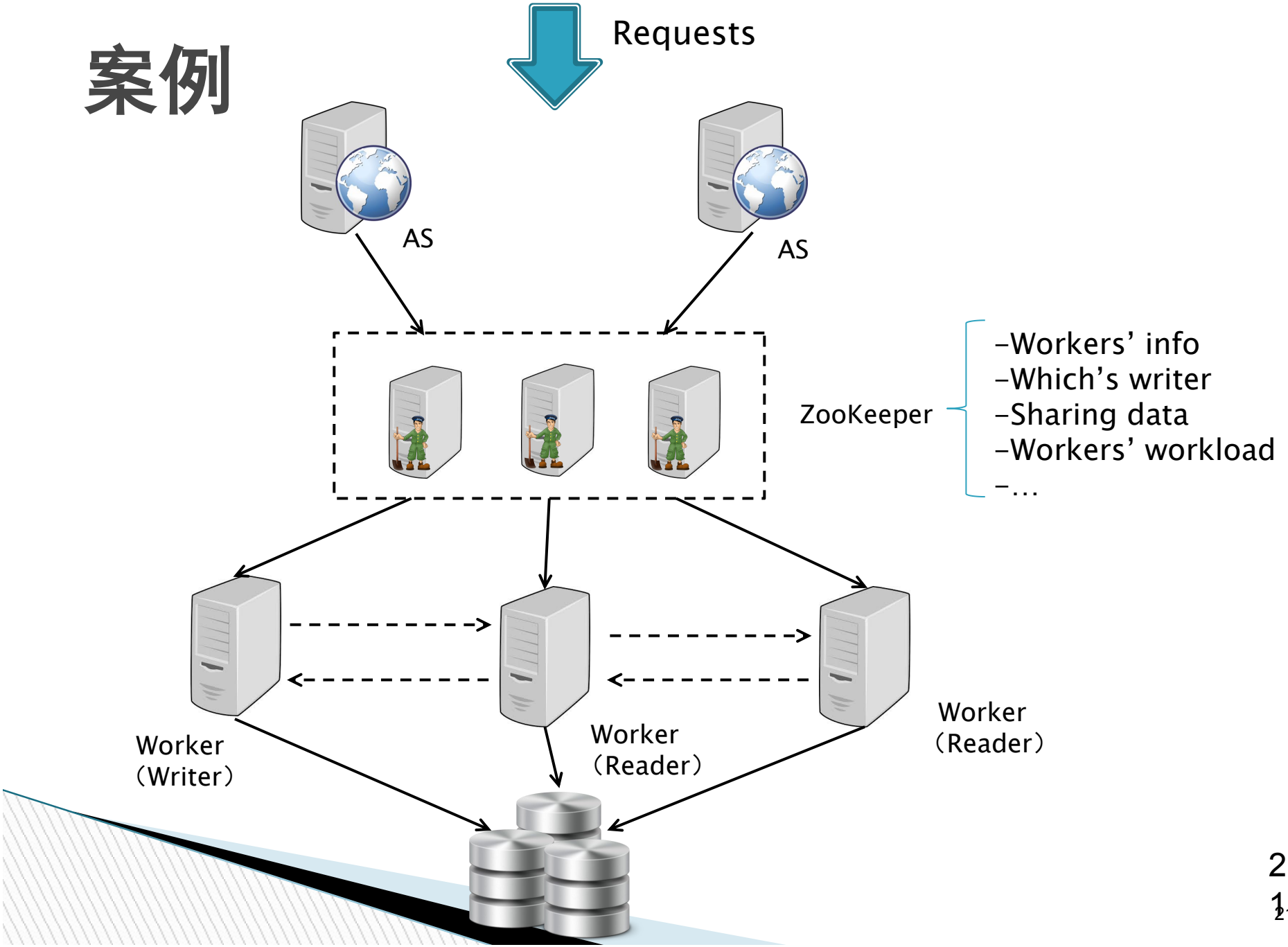
- Watcher定义：A watch event is one-time trigger, sent to the client that set the watch, which occurs when the data for which the watch was set changes.

- ZooKeeper的任何一个读操作都能设置Watch
  - getData()
  - getChildren()
  - exists()

Watcher机制

- Watcher Event包括下面两种：
  - KeeperState
    - Disconnected
    - SyncConnected
    - Expired
  - EventType
    - None
    - NodeCreated
    - NodeDeleted
    - NodeDataChanged
    - NodeChildrenChanged

案例


Requests
AS
AS
ZooKeeper
-Workers' info
-Which's writer
-Sharing data
-Workers' workload
-…
Worker
（Writer）
Worker
（Reader）
Worker
（Reader）

Q&A