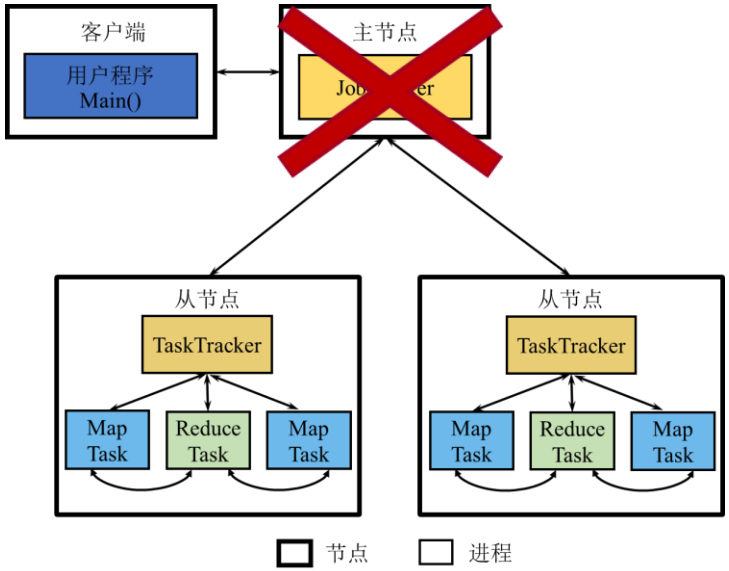
第六章　协调服务系统ZooKeeper

徐　辰
cxu@dase.ecnu.edu.cn

華東師範大學

回顾：JobTracker故障

- 对于MapReduce 1.0的架构，JobTracker故障意味着所有作业需要重新执行
- MapReduce 1.0没有处理JobTracker故障的机制，因而成为单点瓶颈



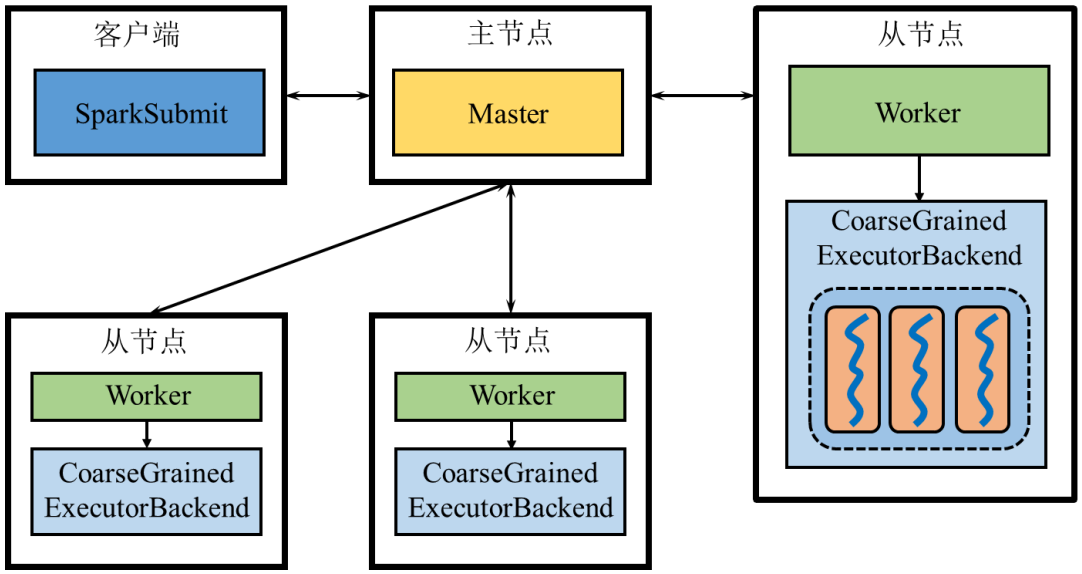
回顾：Resource Manager故障

- 如果Resource Manager发生故障，那么它在进行故障恢复时需要从某一持久化存储系统中恢复状态信息，所有应用将会重新执行

- 我们可以部署多个Resource Manager并通过ZooKeeper进行协调，从而保证Resource Manager的高可用性

回顾：故障类型

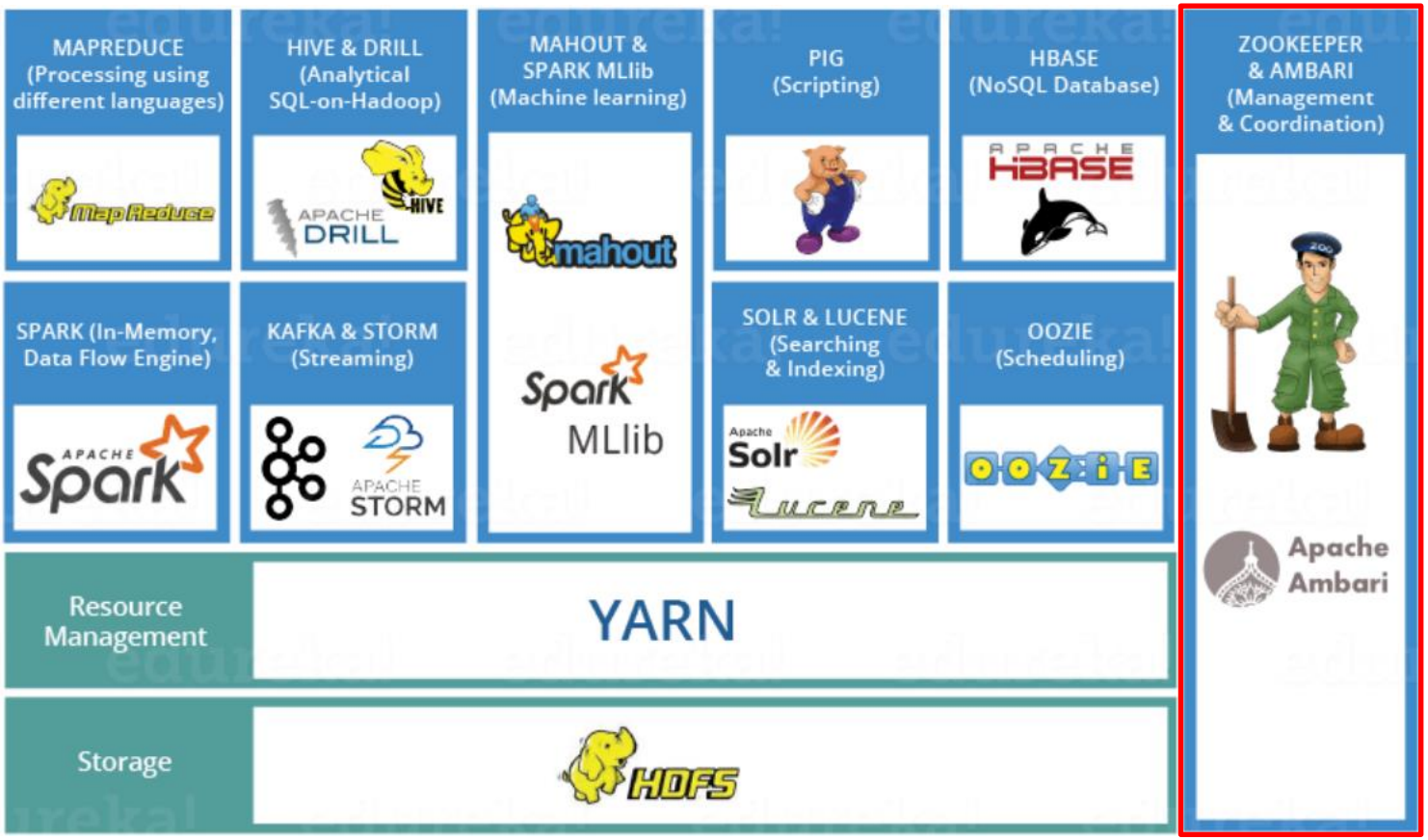
- Master故障：ZooKeeper配置多个Master
- Worker故障
- Executor故障
- Driver故障：重启



ZooKeeper简介

- 轻量级的分布式系统
  - 并不用于存储大量数据，而是用于存储元数据或配置信息等
- 用于解决分布式应用中通用的协作问题
  - 命名服务
  - 集群管理
  - 配置更新
  - 同步控制

ZooKeeper的广泛使用



大纲

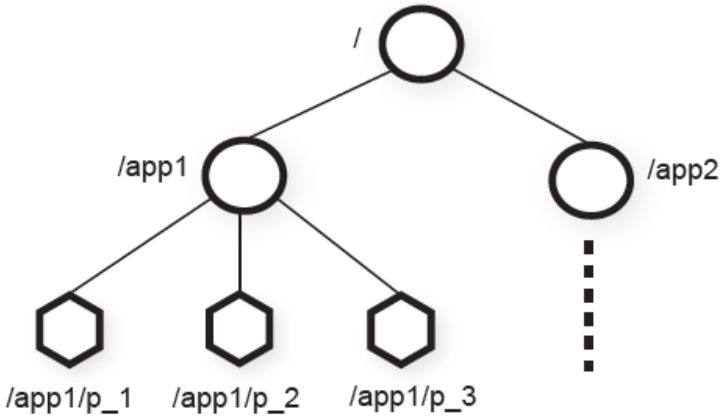
- 设计思想
  - 数据模型
  - 操作原语
- 体系架构
- 工作原理
- 容错机制
- 典型示例

数据模型

- 树：类似文件系统的层次数据结构

- Znode：树中的节点，均用于保存信息
  - /app1　　文件夹
  - /app1/p_1
  - /app1/p_2

文件



Znode

- Znode的持久性
  - 持久 (Persist)：一旦创建了这个Znode，除非主动进行Znode的移除操作，否则这个Znode将一直存在于Zookeeper系统中
  - 临时 (Ephemeral)：Znode生命周期和客户端会话（Session）绑定，一旦会话失效，那么这个客户端创建的所有临时Znode都会被移除
- Znode的顺序性
  - Sequential属性：创建该Znode时会自动在Znode名字后面追加上一个整型数字（由该Znode的父Znode维护的自增数字）

Znode

- 是否会自动删除
  - 持久Znode：用户需要显式的创建、删除
  - 临时Znode：用户创建后，可以显式的删除，也可以在Session结束后，由ZooKeeper服务器自动删除
- 是否带有顺序号
  - 如果创建的时候指定Sequential属性，该Znode的名字后面会自动追加一个自增的整型数字（顺序号）

Znode的四种类型

- **PERSISTENT**(持久Znode)：客户端断开连接，该Znode依旧存在
- **PERSISTENT_SEQUENTIAL**(持久顺序编号Znode)：客户端断开连接后，该Znode仍存在；Zookeeper给该Znode的名称进行顺序编号
- **EPHEMERAL**(临时Znode)：客户端与Zookeeper断开连接后，该Znode被删除
- **EPHEMERAL_SEQUENTIAL**(临时顺序编号Znode)： 客户端断开连接后，该Znode被删除；Zookeeper给该Znode的名称进行顺序编号

大纲

- 设计思想
  - 数据模型
  - 操作原语
- 体系架构
- 工作原理
- 容错机制
- 典型示例

Client API

- create(path, data, flags)
- delete(path, version)
- exist(path, watch)
- getData(path, watch)
- setData(path, data, version)
- getChildren(path, watch)
- sync(path)

常用API含义

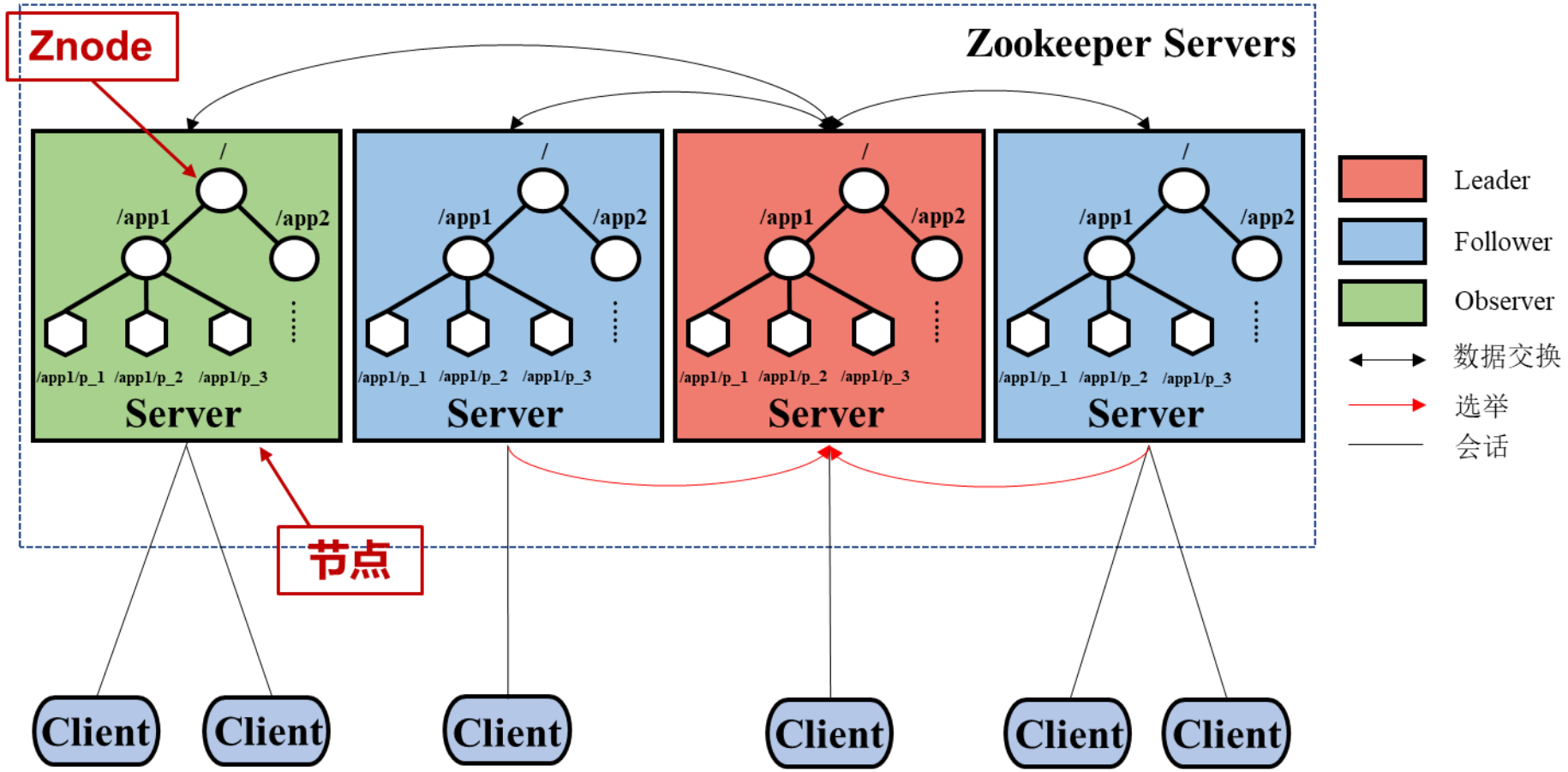
- create: 在树中某一位置添加一个Znode
- delete: 删除某一Znode
- exists: 判断某一位置是否存在Znode
- get data: 从某一Znode读取数据
- set data: 向某一Znode写入数据
- get children: 查找某一Znode的子节点
- sync: 用于等待数据同步

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例

架构图

- 与MapReduce、Spark架构图有很大区别

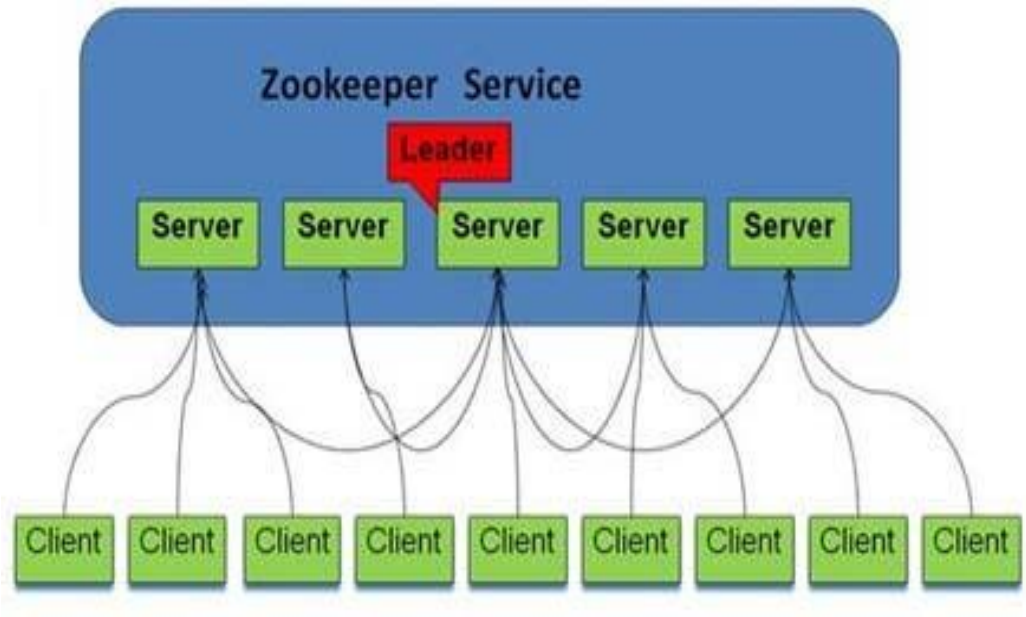


服务器

- 服务器(Server)：每台服务器都维护一份树形结构数据的备份
  - 领导者(Leader)：参与选主过程的服务器通过一定的算法选定一个节点作为领导者，领导者节点可以为客户端直接提供读、写服务。
  - 追随者(Follower)：追随者仅直接提供读服务，客户端发给追随者的写操作要将转发给领导者
  - 观察者(Observer)：与追随者类似，区别在于观察者不参与选举领导者的过程。观察者的角色是可选的，服务器中可以没有观察者节点

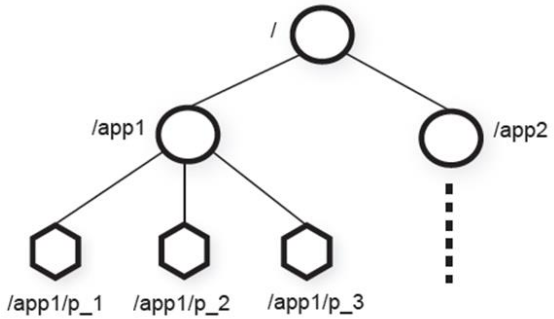
客户端

- 客户端通过执行前述API来操纵ZooKeeper中维护的数据
- 问题：谁是客户端?



会话机制

- 会话(Session)
  - 服务器与客户端之间的连接
  - 心跳和超时机制
- 客户端
  - 客户端在某个Znode上设置一个Watcher，跟踪该Znode上的变化
- 服务器:
  - 一旦该Znode发生变化（包括存储的数据、名称改变、子Znode增加等），服务器会通知设置Watcher的客户端



大纲

- 设计思想
- 体系架构
- 工作原理
  - 领导者选举
  - 读写请求流程
- 容错机制
- 典型示例

为什么需要选领导者？

- ZooKeeper从某种意义上说是一个轻量级的数据存储系统，维护了数据的多个副本
- 由于写操作会改变数据的内容，所以必须保证每个节点都执行相同的操作序列，从而达到副本之间的一致性。
- ZooKeeper中需要有领导者节点来保证各节点执行相同的写操作序列。如果没有领导者节点，那么难以保证各节点副本之间的一致性。

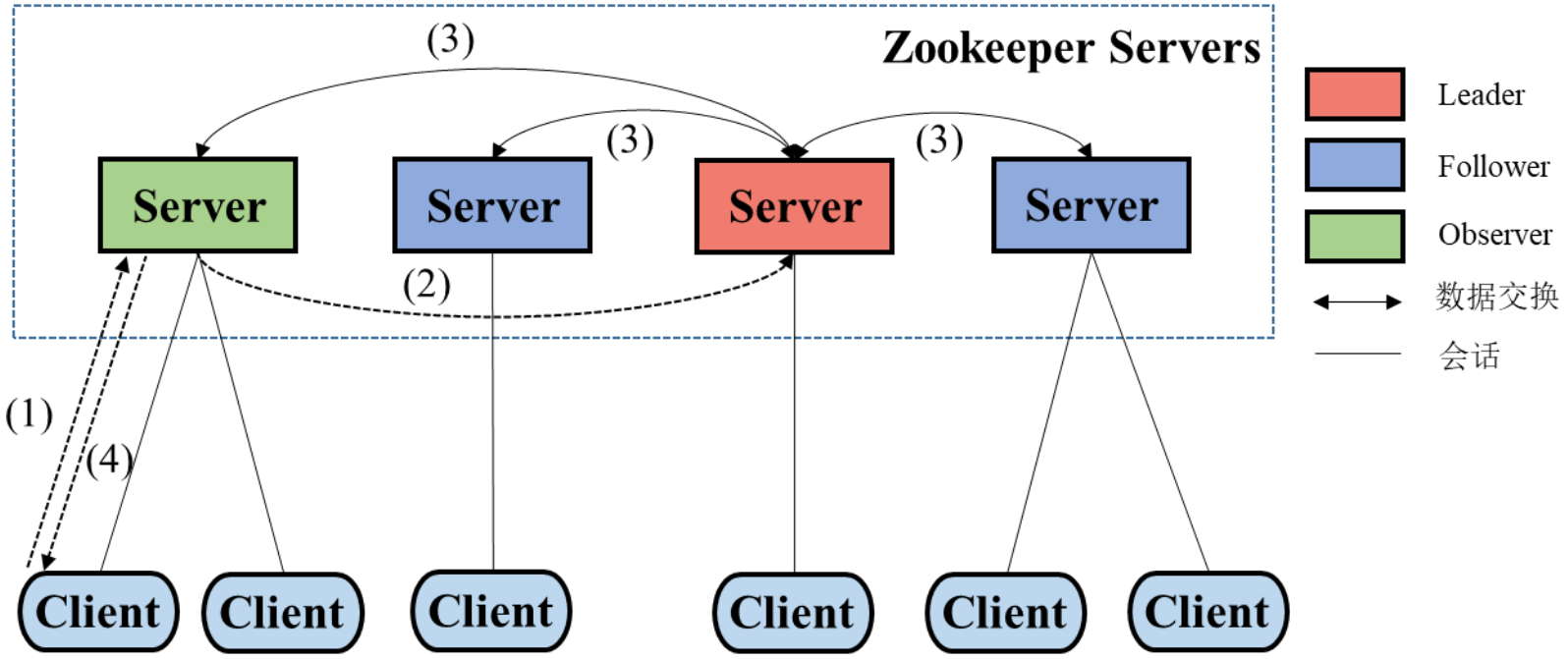
如何进行领导者选举？

- ZooKeeper工作过程中最重要的问题是如何在服务器之间确定领导者，确定了领导者之后才可以进行读写操作
- 分布式一致性协议是用来解决分布式系统如何就某个提议达成一致的问题
  - 经典的实现分布式一致性协议的算法有Paxos等。这些算法的基本思想都是由某些节点发出提议，再由其它节点进行投票表决，最终所有节点达成一致
  - 本课程不深入讨论^_^

大纲

- 设计思想
- 体系架构
- 工作原理
  - 领导者选举
  - 读写请求流程
- 容错机制
- 典型示例

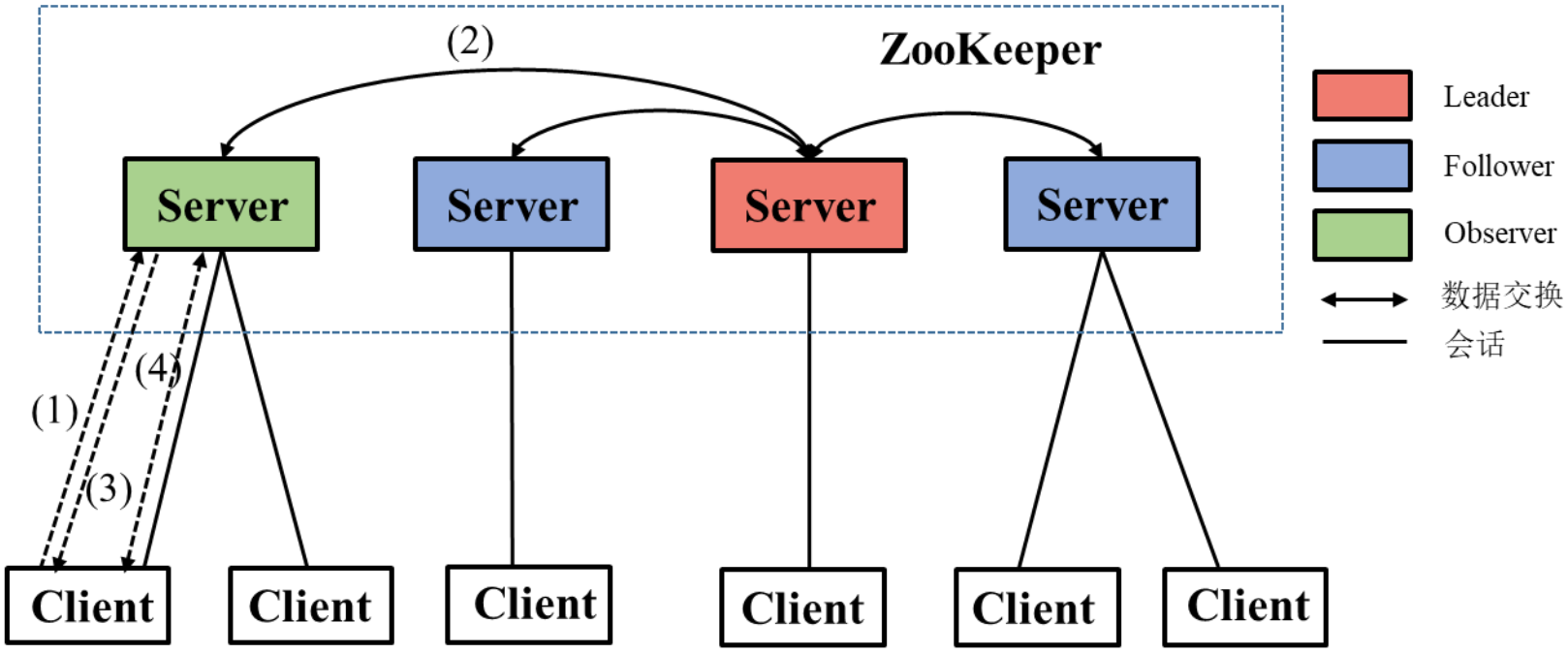
写请求流程


Zookeeper Servers
(3)
(3)
(3)
Server
Server
Server
Server
(2)
(1)
(4)
Client
Client
Client
Client
Client
Client
Leader
Follower
Observer
数据交换
会话

写请求流程

1. 客户端与某一服务器建立连接发起写请求
2. 若服务器是追随者或观察者，则将接收到写请求转发给领导者，否则领导者直接处理
3. 数据同步
   - 理论上，领导者要将写请求发给所有服务器，直到所有服务器都成功执行了写操作，该写请求才算是完成
   - 事实上，保证半数以上的节点写操作成功就可以认为写请求完成。这一过程实际上是追随者、观察者与领导者之间进行数据同步的过程，该过程需要使用分布式一致性协议
4. 数据返回

读请求流程


(2)
ZooKeeper
Server
Server
Server
Server
(1)
(4)
(3)
Client
Client
Client
Client
Client
Client
Leader
Follower
Observer
数据交换
会话

读请求流程

1. 客户端与某一服务器建立连接并发起sync()请求
   - 当客户端发起读数据请求时，无论是领导者、追随者还是观察者都可以响应请求。
   - 如果是追随者或观察者响应请求，那么直接返回给客户端的数据可能不是最新的。
2. 当如果该服务器是追随者或观察者，那么与领导者进行数据同步
   - 数据同步使之后客户端发起的读请求获得最新数据。
3. 服务器向客户端返回同步成功的信息
4. 客户端从服务器读取数据

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例

故障类型

- 领导者节点故障：ZooKeeper需要重新进行领导者选举

- 追随者或观察者节点故障：该节点无法对外提供服务，但其它节点依然可以正常提供服务，所以不影响ZooKeeper的服务
  - 如果追随者或者观察者节点发生故障后进行重启，那么它们将可从领导者节点或其它节点进行数据恢复

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例
  - 命名服务
  - 集群管理
  - 配置更新
  - 同步控制

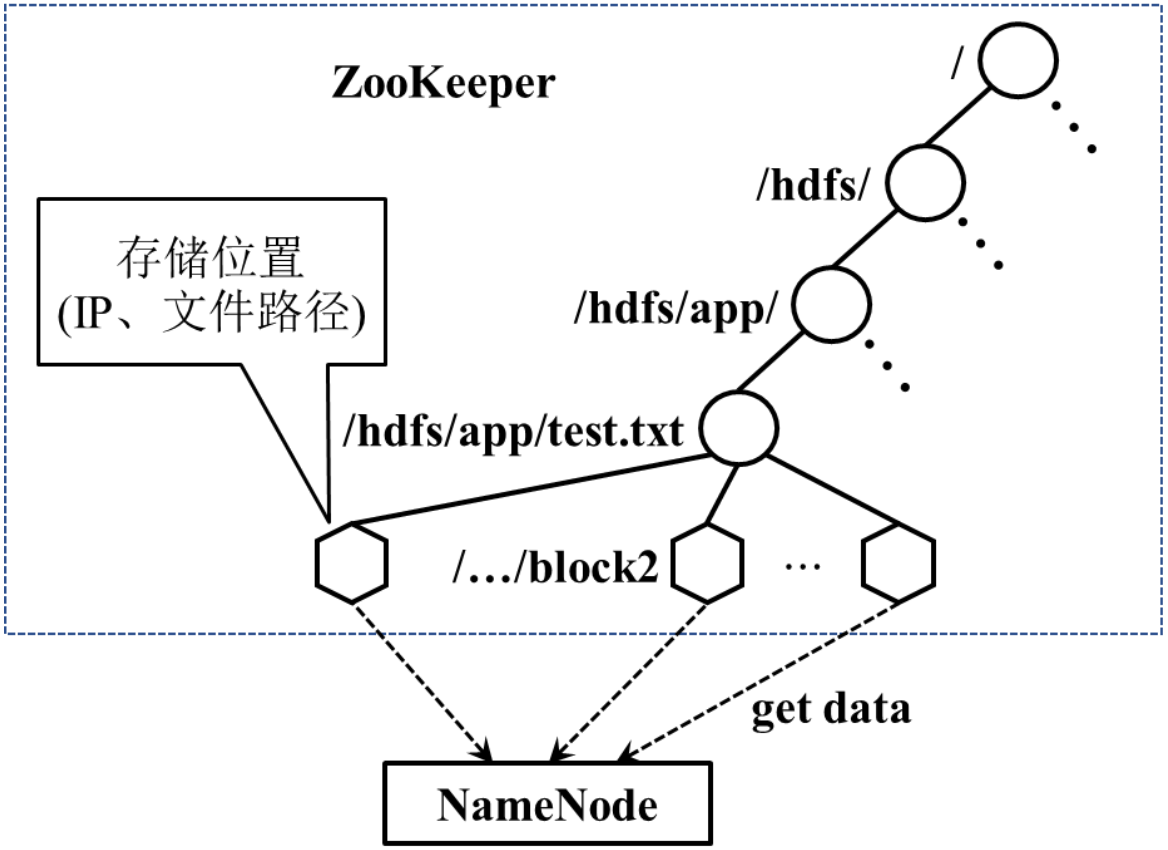
命名服务

- 统一命名服务：树形的名称结构
  - 域名解析：某一域名的IP地址是多少？
  - HDFS目录与文件：某一文件存储在哪里？

- 命名服务已经是Zookeeper内置的功能，只要调用Zookeeper的API 就能实现
  - 调用create接口就可以创建一个Znode
  - 利用Znode存储信息，用于实现相应功能

命名服务

- 例子：若HDFS使用ZooKeeper维护命名空间，文件hdfs:///app/test.txt的存取



大纲

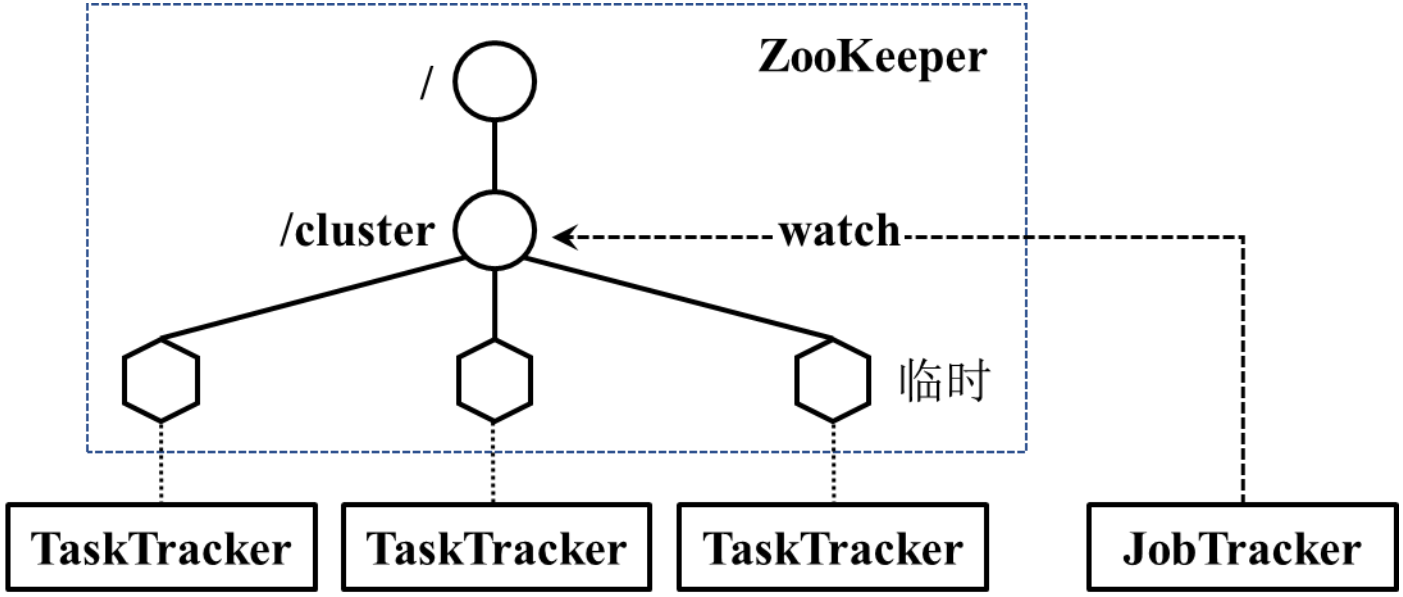
- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例
  - 命名服务
  - 集群管理
  - 配置更新
  - 同步控制

集群管理

- 在分布式环境下节点故障时有发生，因而存在如下需求
  - 状态监控：主节点需要实时监控集群中的从节点的状态（存活还是宕机）是否发生变化
  - 选主：为了支持分布式计算系统的高可用，需要配置多个主节点
    - 选主依赖于复杂的一致性协议（如Paxos等），工程实现复杂
    - Zookeeper提供了便捷的选主方式

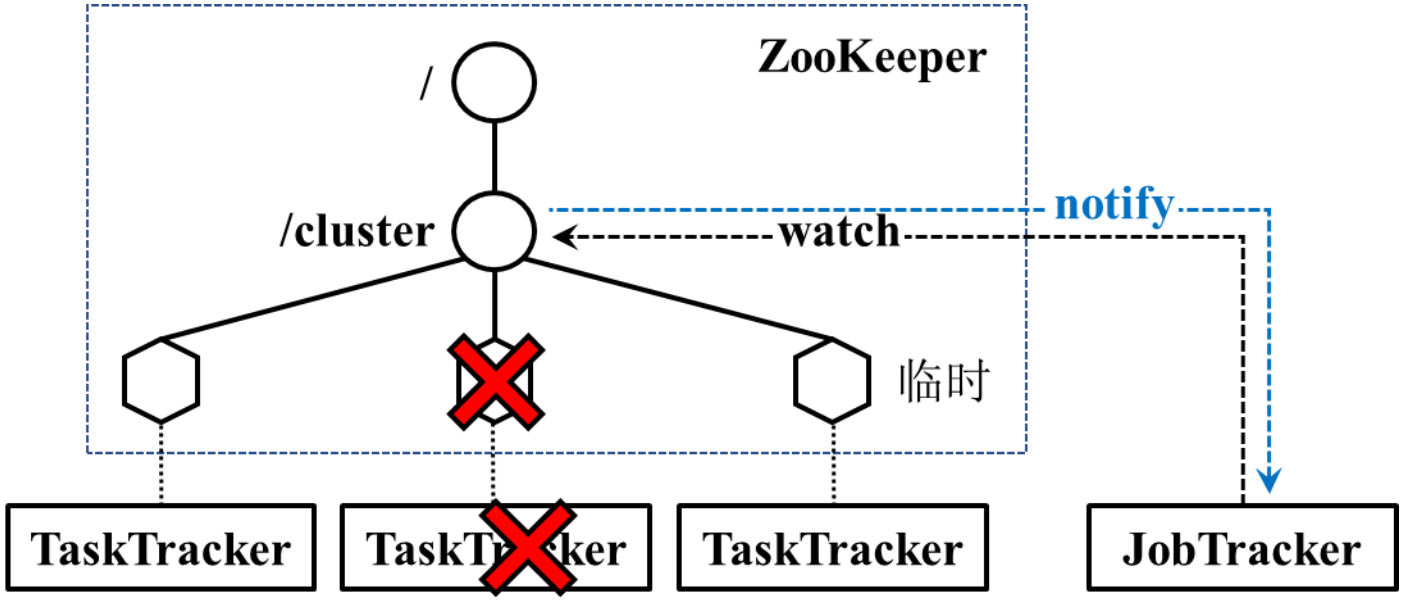
状态监控

- 以MapReduce架构为例的状态监控步骤
  - JobTracker创建一个Znode（/cluster）并监听
  - 每个TaskTracker启动后在/cluster下创建一个临时Znode



状态监控

- 以MapReduce架构为例的状态监控步骤
  - TaskTracker创建的临时Znode增加或减少（即TaskTracker增加或减少）时，ZooKeeper会通知JobTracker



选主

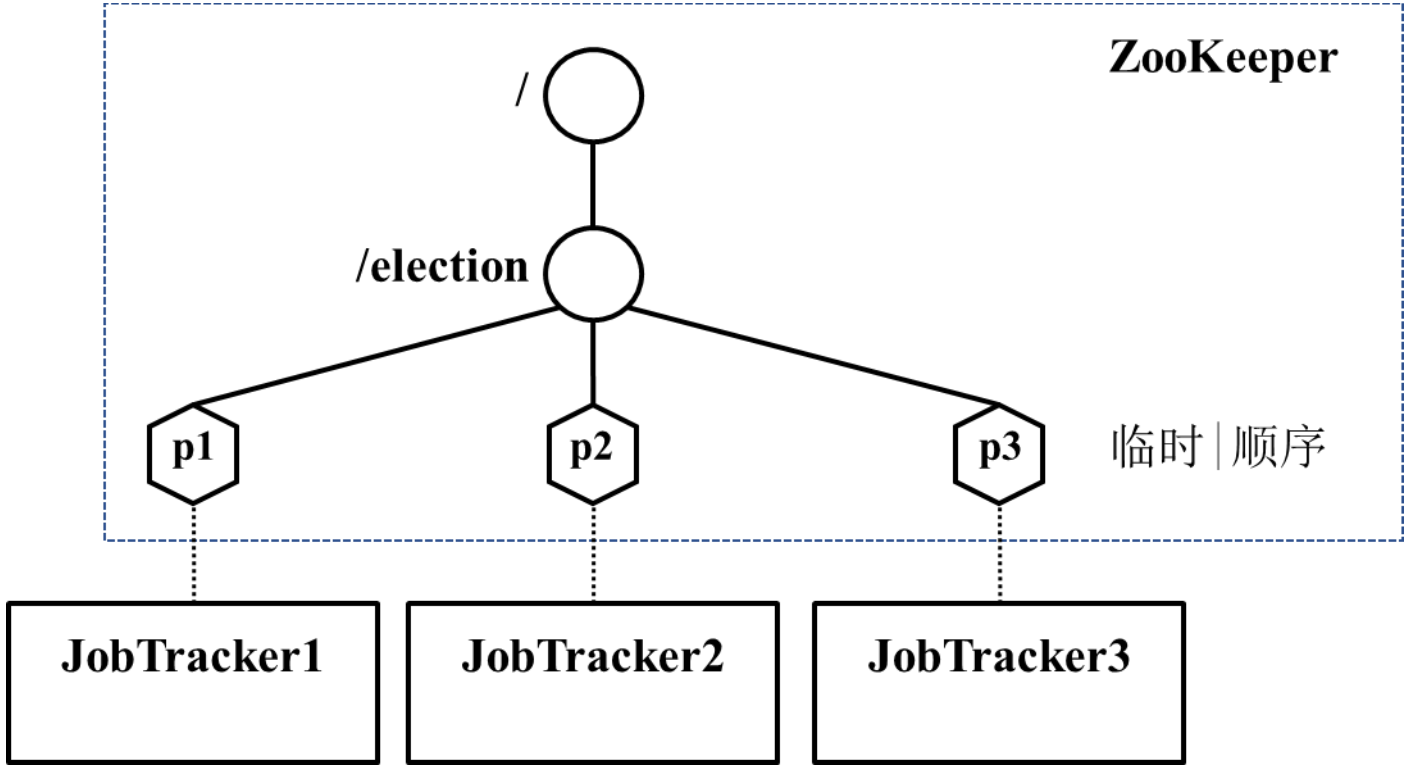
- MapReduce 1.0中存在JobTracker单点故障问题，需要配置多个JobTracker实现高可用

  注意与ZooKeeper本身Leader选举的区别

  - 何时需要选主?
  - 如何从多个JobTracker中进行选主?
- 两种情况
  - 初次选主：系统启动时在多个JobTracker中选择一个作为主节点
  - 重新选主：当主节点宕机时选取新的JobTracker作为主节点

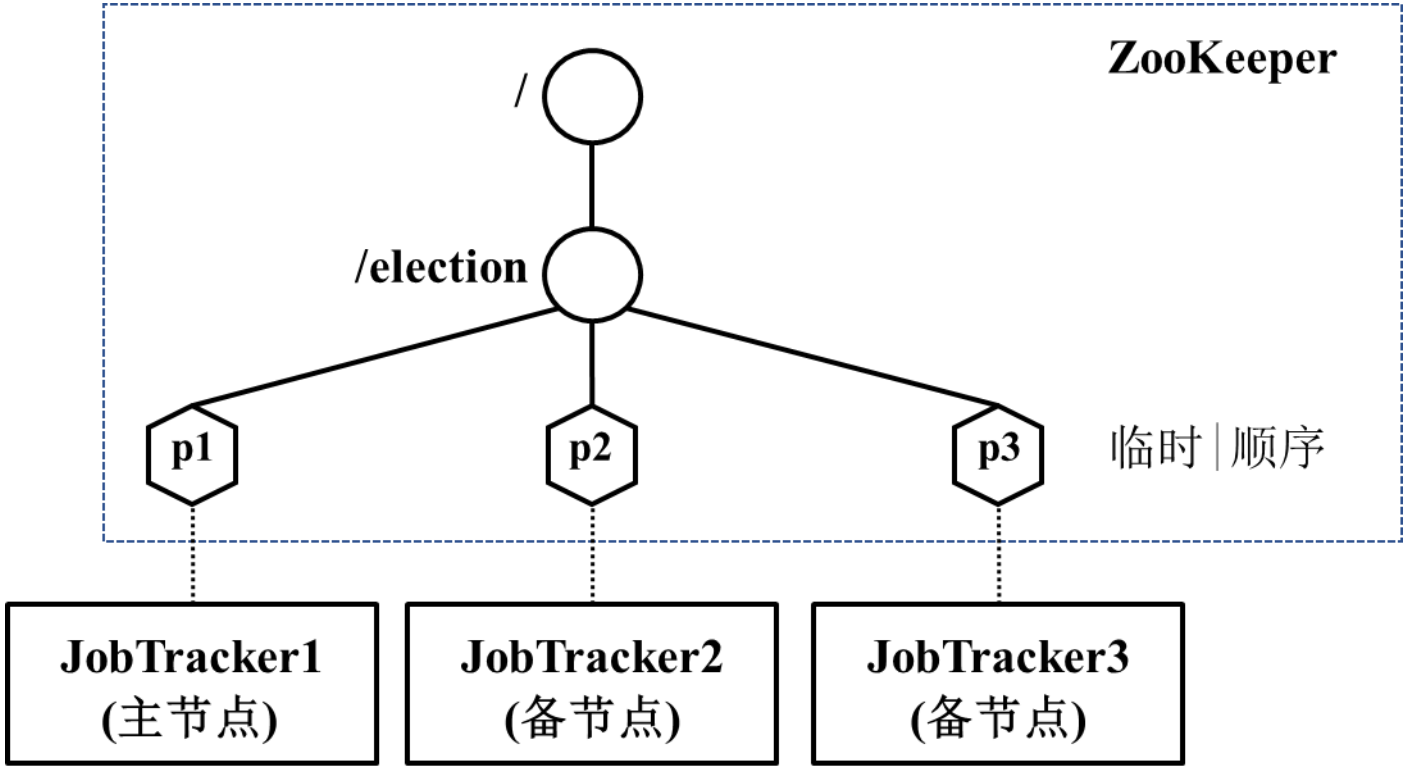
初次选主

- 初始时每个JobTracker均创建名为/election/p的临时Znode并添加顺序属性



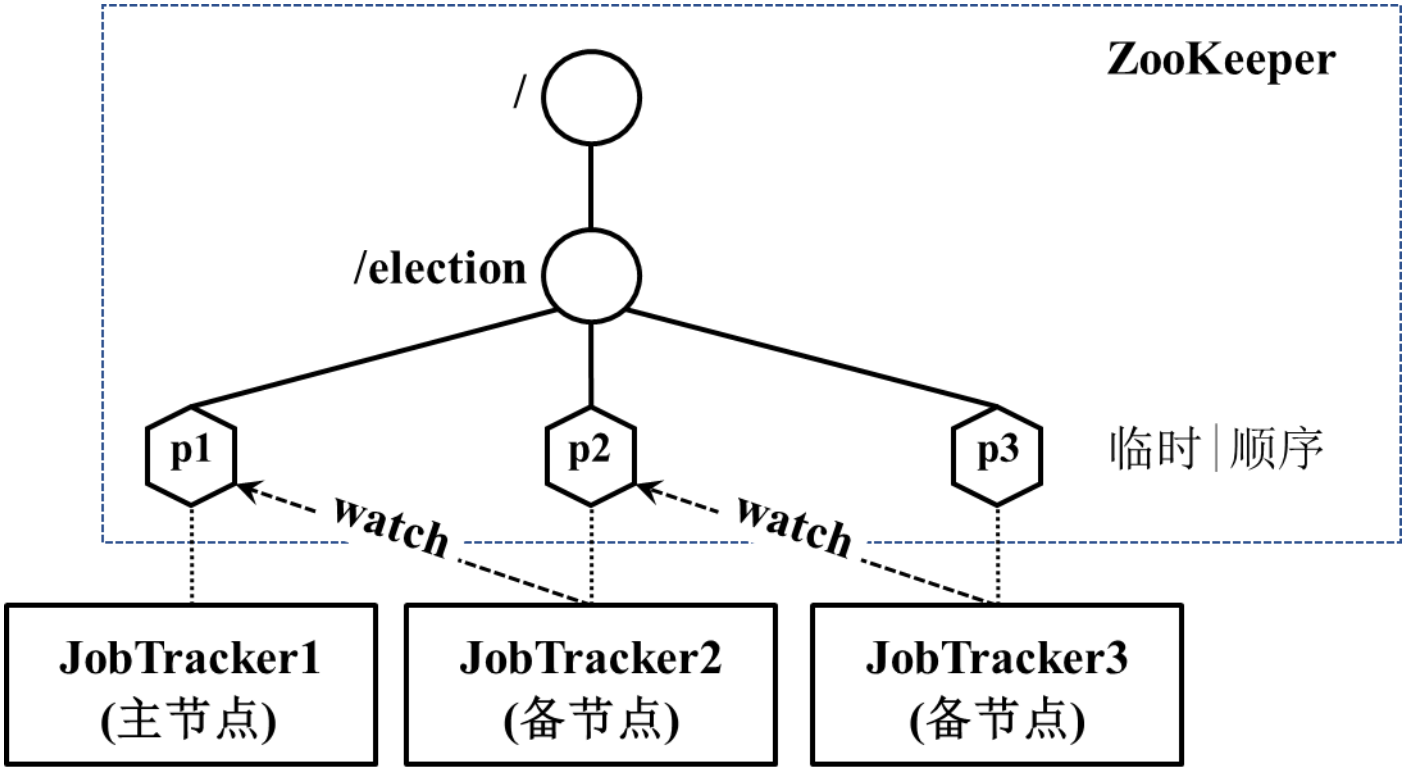
初次选主

- 选主即为选取/election下编号最小的Znode



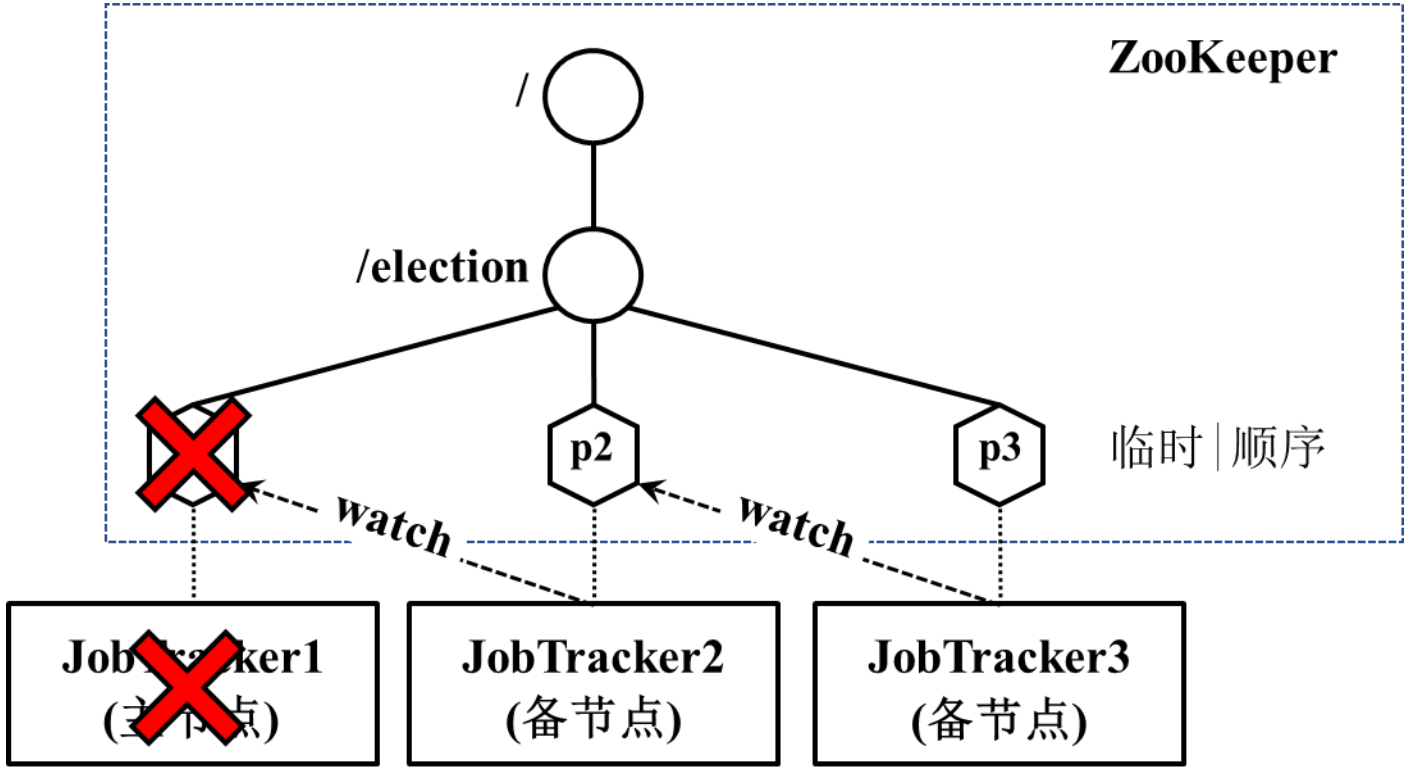
初次选主

- 选主过程中未当选主节点的JobTracker监听前一位JobTracker的Znode



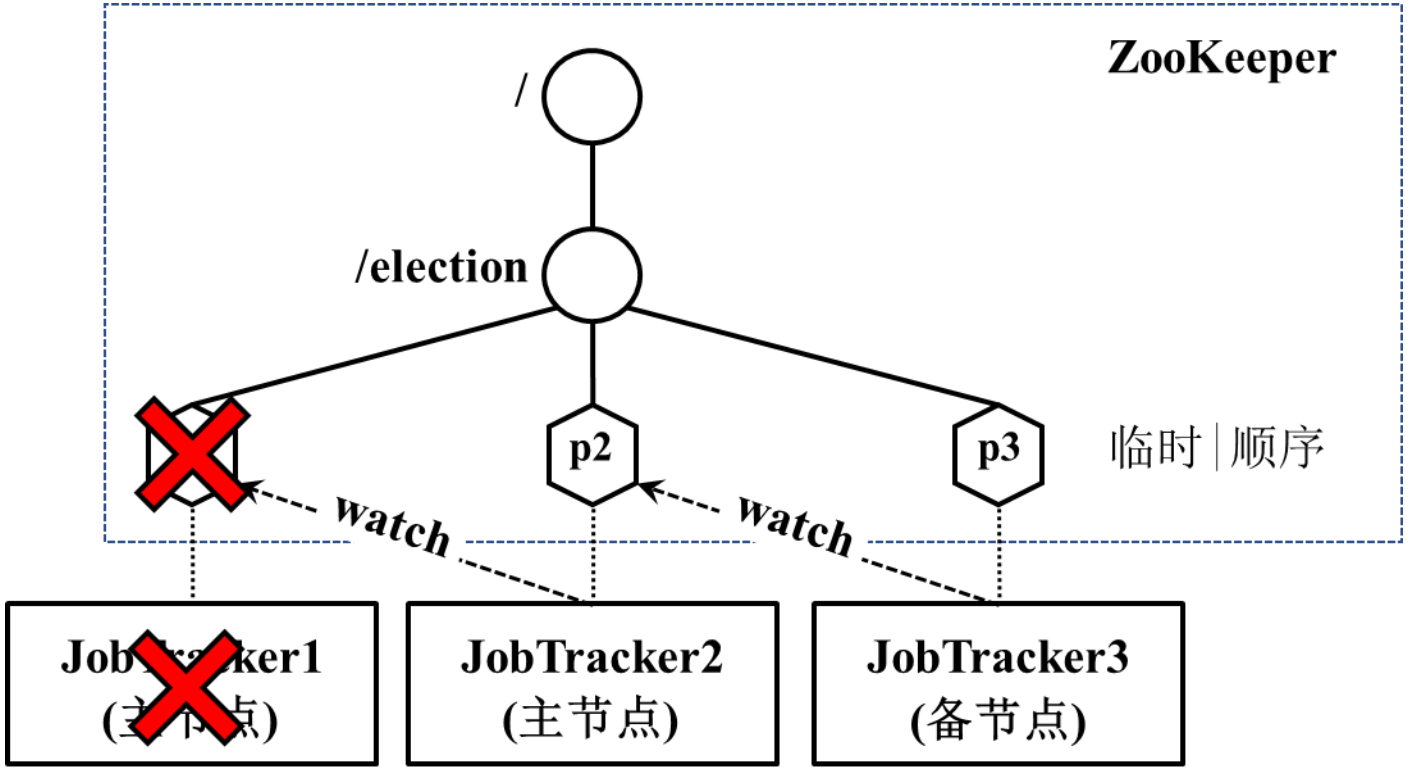
重新选主

- 当主节点宕机后，排在其后一位的JobTracker监听到这一变化



重新选主

- 此时，该JobTracker监听自己的编号是否是最小的，若是则当选主节点。



大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例
  - 命名服务
  - 集群管理
  - 配置更新
  - 同步控制

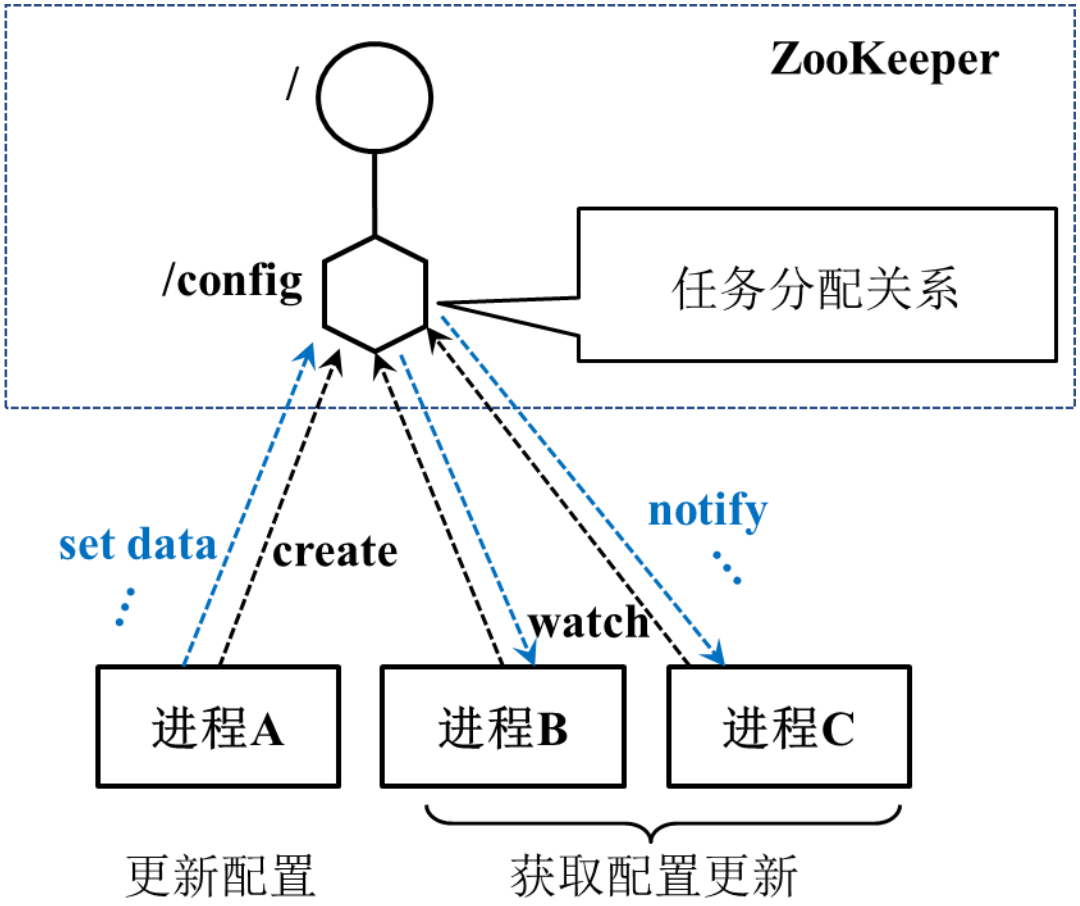
配置更新

- 场景：分布式系统运行过程中往往需要动态地修改配置

- 问题简化
  - 假设分布式系统中有A、B、C三个进程
    - 进程A负责创建配置文件
    - 进程B、C读取该配置文件并据此进行相应操作
  - 在这些进程运行的过程中，进程A根据需要修改该配置文件，进程B、C则根据修改后的配置进行动态地调整

简化示例

- 动态改变任务分配关系



一般方法

- 配置信息存放在 ZooKeeper 某个Znode，所有需要获取配置更新的进程监听该Znode

- 一旦配置信息发生变化，每个监听的进程收到通知，然后从ZooKeeper获取新的配置信息并应用

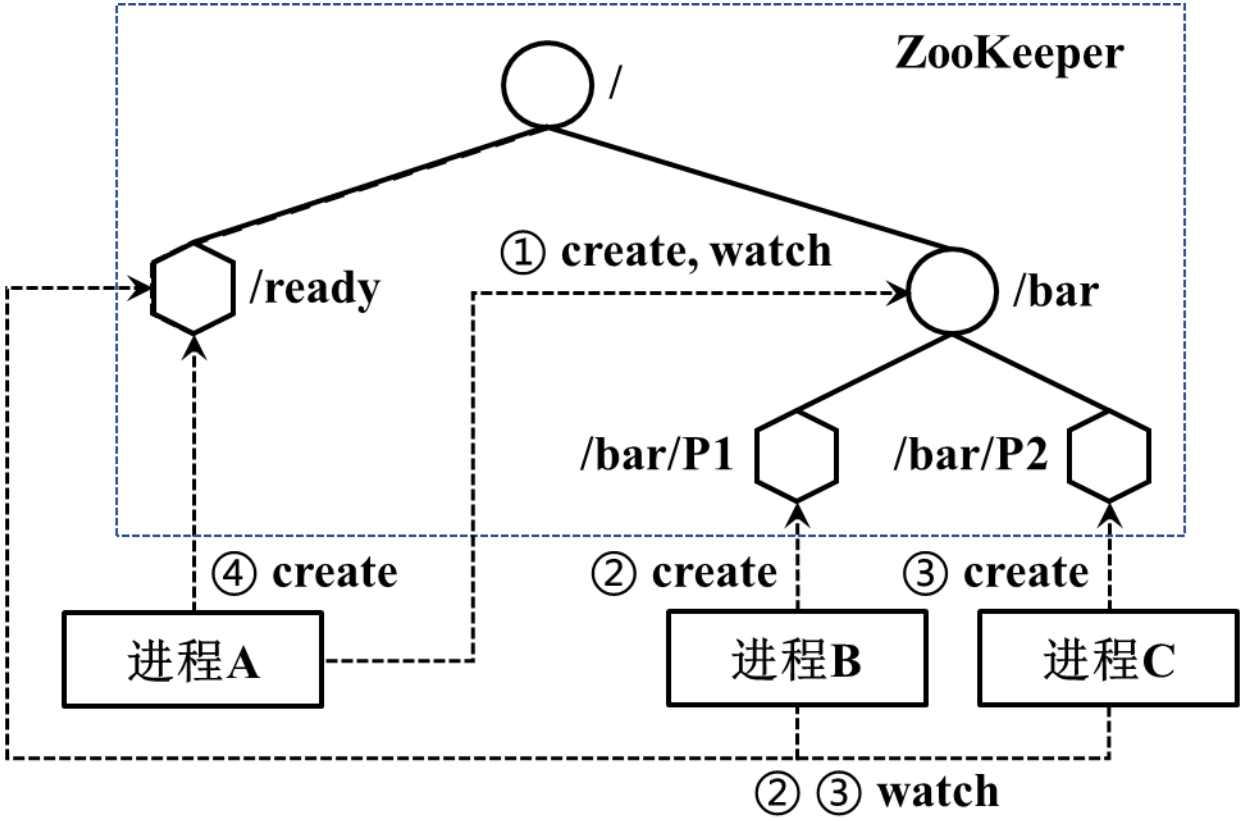
大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例
  - 命名服务
  - 集群管理
  - 配置更新
  - 同步控制

同步控制

- 场景：BSP模型要求在每一轮迭代计算的开始和结束时同步所有参与的进程
- “双屏障”机制
  - 进入屏障时，需要足够多的进程准备好
  - 当所有进程执行完任务时，才可离开屏障
- 问题简化：假设需要由三个进程A、B、C来协作完成一项作业
  - 进程A作为负责协调的主进程
  - 进程B、C负责执行分配的任务

进入屏障

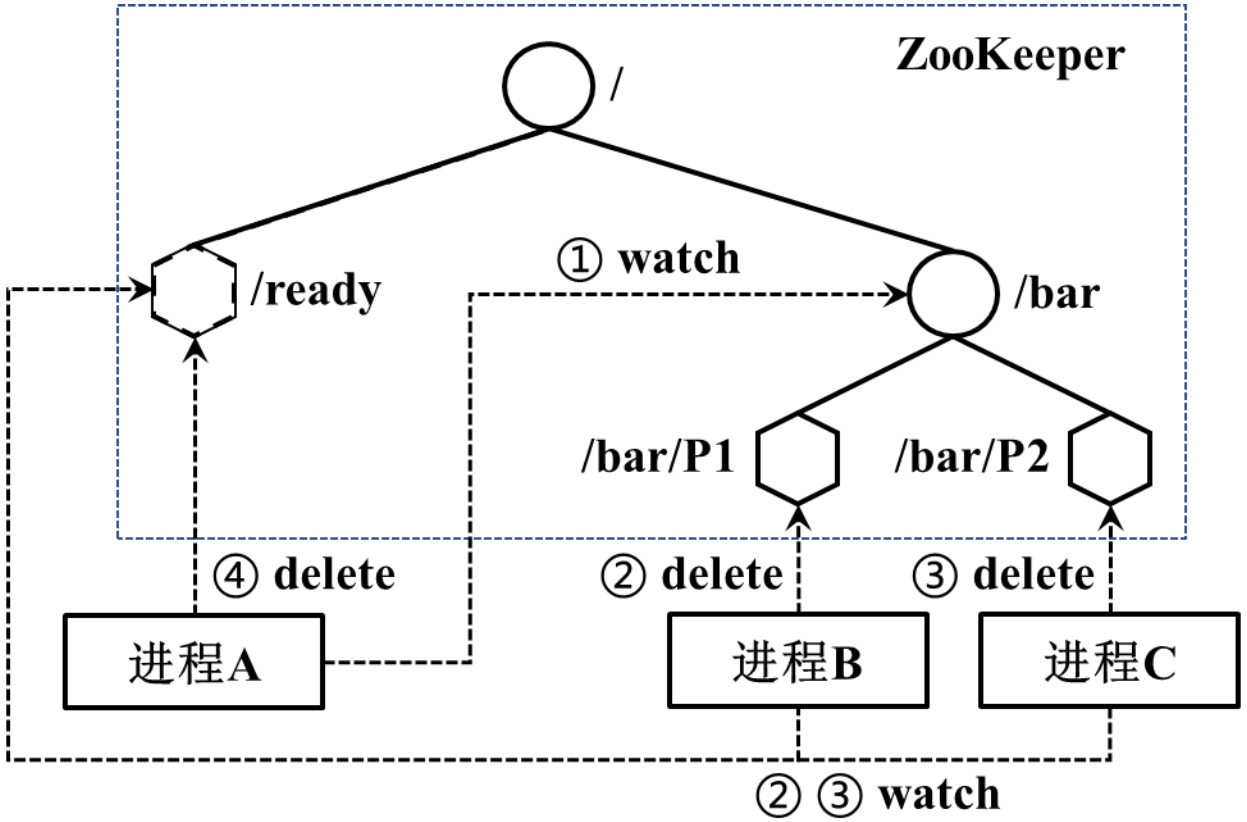


在进入屏障阶段，进程A创建并监听名为/bar的Znode

进程B、C则在/bar下创建一个Znode来表示自己已准备好，并通过Watch机制监听是否存在名为/ready的Znode

进程A通过创建/ready来通知其他进程开始执行任务

离开屏障



进程B、C在完成任务后删除/bar/P1来向进程A表明自己已准备离开屏障，并监听/ready来等待进程A发出的离开屏障的通知

当进程A判断/bar下没有Znode时，意味着所有进程已完成任务，于是进程A删除/ready来通知它们离开屏障

一般方法

- 双屏障机制实现方法
  - 进入屏障：协调进程创建名为/bar的Znode。每个工作进程都到/bar下注册，并监听是否存在/ready Znode。/bar下注册的进程数达到要求时，协调进程创建/ready，通知工作进程开始执行任务。
  - 离开屏障：每个工作进程完成任务后删除其在/bar下注册的Znode。/bar下无Znode时，协调进程删除/ready，通知工作进程离开屏障。

课后阅读

- 论文
  - Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2010). ZooKeeper : Wait-free coordination for Internet-scale systems. In USENIX Annual Technology Conference (pp. 1–14).

本章小结

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 典型示例