## ZooKeeper 提供的通用服务如下

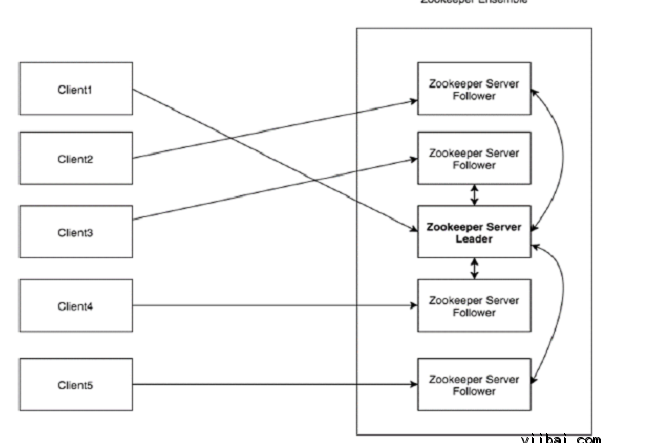
* 命名服务 − 确定在一个集群中的节点的名字。它类似于DNS，只不是过节点。
* 配置管理 − 系统最近加入节点和向上最新配置信息。
* 集群管理 − 加入/节点的群集和节点状态实时离开。
* 节点领导者选举 − 选举一个节点作为领导者协调的目的。
* 锁定和同步服务 − 锁定数据，同时修改它。这种机制可以帮助自动故障恢复，同时连接其它的分布式应用程序。如Apache HBase。
* 高可靠的数据注册表 − 一个或几个节点的可用性的数据向下。

## ZooKeeper的优点

* 简单的分布式协调过程
* 同步 − 互斥和服务器进程之间的合作。这个过程有助于Apache HBase 的配置管理。
* 有序消息
* 序列化− 根据特定的规则进行编码数据。确保应用程序不断地运行。这种方法可以用来在MapReduce的协调队列以执行正在运行的线程。
* 可靠性
* 原子性 − 数据传输成功或完全失败，但没有事务处理部分。

## ZooKeeper的体系结构

看看下面的图。它描绘ZooKeeper 的“客户端 - 服务器架构”。



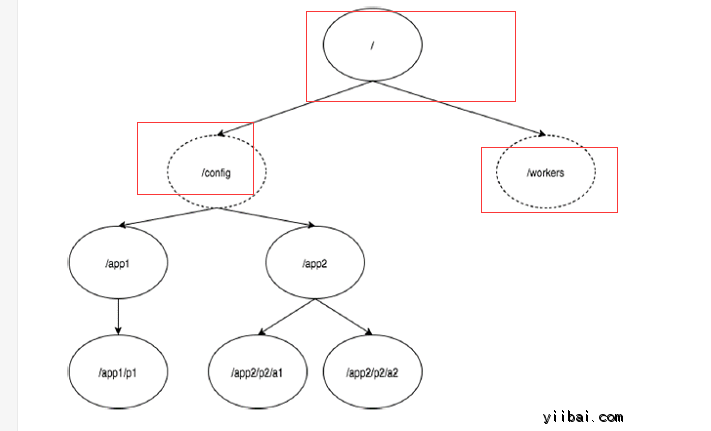
ZooKeeper 架构的一部分组件如下表中所解释。

|  |  |
| --- | --- |
| **部分** | **描述** |
| Client | 客户端，在我们的分布式应用集群的一个节点，从服务器获取信息。对于一个特定的时间间隔，每个客户端将消息发送到服务器，以让服务器都知道客户机是活的。  同样，服务器会发送一个确认当客户端连接。如果没有从所连接的服务器的响应，客户端自动重定向消息到另一个服务器 |
| Server | 服务器，ZooKeeper集成的一个节点，提供所有的服务提供给客户。给出应答客户，告知该服务器还活着 |
| 合组 | ZooKeeper 服务器组。节点所需要形成的合奏的最小数目为3 |
| Leader | 它执行自动恢复，如果任何连接的节点的故障的服务器节点。领导者服务启动 |
| Follower | 遵循领导指示服务器节点 |

## 分层命名空间

下图显示了用于内存中表示 ZooKeeper 文件系统的树形结构。 ZooKeeper节点被称为znode。每个znode由一个名称识别，并通过路径（/）序列隔开。

* 在图中，首先有一个根znode，它由“/”分隔。在根下，有两个逻辑命名空间 config 和 workers。
* 在config命名空间用于集中配置管理以及 workers 命名空间用于命名。
* 在 config 命名空间下，每个znode可以存储高达 1MB 的数据。这类似于UNIX文件系统，不同的是父 znode 也可以存储数据。这种结构的主要目的是存储同步数据以及描述znode的元数据。这种结构被称为 ZooKeeper数据模型



在 ZooKeeper 数据模型中每个 znode 维护一个 stat 结构。 一个统计（stat ）只是提供了一个 znode 元数据。 它由版本号，动作控制列表（ACL），时间戳和数据长度组成。

* **版本号** − 每个znode都有一个版本号，这意味着每个相关的时间使用节点改变数据，其相应的版本号也将增加。使用版本号是重要的，在多个 zookeeper 的客户端正在努力通过相同znode执行操作。
* **动作控制列表（ACL）** −ACL是基本的身份验证机制，用于访问znode。它管理所有的znode读写操作。
* **时间戳**− 时间戳表示过去时间，从znode创建和修改起算。它通常以毫秒表示。ZooKeeper 确定每次从“事务ID”（zxid）更改znodes。Zxid是独特的，为每个事务处理维持时间，使您可以轻松地识别从一个请求到另一个请求经过的时间。
* **数据长度** − 存储在 znode 数据的合计量是数据长度。可以存储的最大数据容量为1MB。

## Znodes 类型

Znodes 被归类为持久性，顺序和短暂。

* **持久性znode** − 持久性 znode 处于活动状态，即使客户端，它创造了特定的 znode。默认情况下，所有的 znodes 是持久的，除非另有说明。
* **短暂znode** − 短暂znodes活跃，直到客户端还活着。当客户端被从 ZooKeeper 集合断开连接，然后znodes自动删除。由于这个原因，只有短暂znodes不允许再有一个子。如果短周期znode被删除，那么下一个合适的节点，将填补其位置。短暂znodes 发挥在领导选举中起重要作用。
* **连续znode**− 连续znodes可以是持久或短暂的。当一个新的znode作为连续znode创建的，则 ZooKeeper 通过将10位的序列号为原始名称设置znode的路径。例如，如果使用路径 /myapp 来创建一个znode作为连续znode，ZooKeeper将改变路径 /myapp0000000001并设置一个序列号为0000000002。如果两个连续znodes同时被创建，ZooKeeper从来不使用相同数量在每个znode上。连续znodes在锁定和同步中起到重要作用。

## 会话

会话对于 ZooKeeper 操作是非常重要的。请求在会话 FIFO 顺序执行。当一个客户端连接到服务器，会话将建立一个会话ID并分配给客户端。

客户端在特定的时间间隔发送心跳来保持会话有效。如果 ZooKeeper 从客户端接收检测信号超过在服务的开始指定的期间（会话超时），它认为该客户死亡。

会话超时通常以毫秒表示。当一个会话因任何原因而结束，该会话期间短暂创造了的 znodes 会被删除。

## 监视

监视是一个简单的机制，在ZooKeeper集合通知下以获取客户有关的变化。 客户端可以设置监视，同时读取特定znode。监视发送通知给注册的客户机对任何znode（在其上的客户端寄存器）的变化。

节点改变时znode或子znode变化相关联的数据也会被修改。监视只被触发一次。如果客户想要再次通知，则必须通过另一次读操作来完成。当一个连接会话已过期，客户端会从服务器断开，并在相关的监视也将被删除。

# Zookeeper工作流

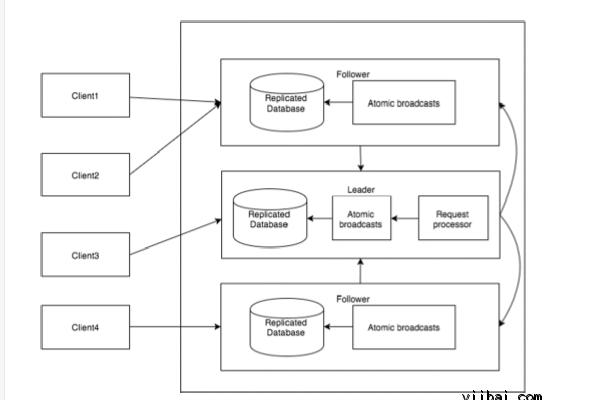
当ZooKeeper集合启动时，它会等待客户端连接。客户端将连接到ZooKeeper的集合的其中一个节点。它可能是一个领导者或跟随者节点。当客户机连接时，该节点分配会话ID给特定的客户端，并发送一个确认消息给客户端。如果客户端没有得到确认，它会尝试连接ZooKeeper集合的另一个节点。当连接到一个节点后，客户端将以规则的间隔发送心跳到节点，以确保连接不会丢失。

* 如果客户想要读取特定的znode，它发送一个读请求使用znode路径的节点，所述节点从其自己的数据库中获取它返回所请求的znode。出于这个原因，读取在动物园管理员集合中速度非常快。
* 如果客户希望将数据存储在ZooKeeper 集合，它发送znode路径和数据到服务器。连接的服务器将请求转发到领导者，那么领导者将重新发出书面请求到所有的追随者。如果只有一个数节点成功响应，接着写请求将成功及一个成功的返回代码将被发送到客户端。否则，写请求将失败。严格大部分节点被称为定额。

## ZooKeeper集合的节点

让我们来分析ZooKeeper集合不同数量的节点的作用。

* 如果我们有一个节点，那么当该节点出现故障时ZooKeeper集合失败。它有利于“单一失败教程”，它不建议用在生产环境中。
* 如果我们有两个节点，一个节点出现故障，我们也没有“多数”，因为二分之一并不是一个大多数。
* 如果我们有三个节点及其一个节点发生故障，我们有大多数，因此它是最低要求。它强制 ZooKeeper 集合在实际生产环境中至少有三个节点。
* 如果我们有四个节点及其有当两个节点失败，它类似于有三个节点。额外的节点没有任何作用，因此，最好是单数增加节点，例如，3, 5, 7.
* 我们知道，写处理它比在 ZooKeeper 集合读过程是昂贵的，由于所有的节点需要写相同的数据在其数据库中。因此，最好是具有节点（3，5或7）比具有大量节点的一个平衡的环境的数量少。
* 下图描述了ZooKeeper 的工作流程以及在随后的表说明了其不同的组件



|  |  |
| --- | --- |
| 组件 | 描述 |
| 写入 | 写过程是由领导节点处理。领导者转发写请求到所有znodes及其等待来自znodes应答。如果一半的znodes的回复，那么写入过程就完成了。 |
| 读取 | 读取在内部由特定连接znode进行的，所以没有必要与集群交互。 |
| 复制数据库 | 它是用来将数据存储在zookeeper。每个znode都有自己的数据库及其每个znode 在一致性的作用下，每次有相同的数据。 |
| 领导者（节点） | 领导者是由Znode负责处理写请求。 |
| 追随者（节点） | 追随者收到来自客户端的写请求，并将其转发到领导znode。 |
| 请求处理器 | 目前仅在领导节点。它从跟随节点的请求支配写入。 |
| 原子广播 | 负责从领导节点到从节点广播更改。 |

# Zookeeper领导人选举

让我们来分析一下一个领导节点在ZooKeeper集合的选举。考虑集群中有N多的节点。领导人选举的过程如下 −

* 所有节点创建一个顺序，znode具有相同路径，/app/leader\_election/guid\_。
* ZooKeeper 的集合将追加的10位序列号的路径，创造了 znode 将会是 /app/leader\_election/guid\_0000000001, /app/leader\_election/guid\_0000000002, ...等。
* 对于给定的实例，它在znode创建最小数量的节点成为领导者以及所有其他节点的追随者。
* 每一个追随者节点监控下一个最小号的znode。

        例如，节点这将创建znode /app/leader\_election/guid\_0000000008 将监控znode/app/leader\_election/guid\_0000000007   
        及其该节点创建znode /app/leader\_election/guid\_0000000007 将监控znode /app/leader\_election/guid\_0000000006.

* 如果领导停机，接着其对应的znode/app/leader\_electionN被删除。
* 跟随节点接下来将通过观察者得到关领导去除的通知。
* 跟随节点接下来会检查是否有其他znodes用最小数量。 如果没有，接着它将承担领导者的角色。否则，它会找到哪些用最小数创造了znode作为领导者的节点。
* 同样，其他所有跟随节点选举创造了znode用最小数作为领导者的节点。

领导人选举时，它从头开始做一个复杂的过程。但ZooKeeper服务，使得它非常简单。让我们在接下来的章节介绍 ZooKeeper 安装和开发。