一、API概述

znode节点可能含有数据，也可能没有。如果一个znode节点包含任何数据，那么数据存储为字节数组（byte array）。

create/path data

创建一个名为/path的znode节点，并包含数据data。

delete/path

删除名为/path的znode。 如果设置为true，我们就

可以通过我们创建ZooKeeper句柄时所设置的Watcher对象得到事件，同时另一个版本的方法提供了以Watcher对象为入参，通过这个传入的对象来接收变更的事件

exists/path

检查是否存在名为/path的节点。

setData/path data

设置名为/path的znode的数据为data。

getData/path

返回名为/path节点的数据信息。

getChildren/path

返回所有/path节点的所有子节点列表。

需要注意的是，ZooKeeper并不允许局部写入或读取znode节点的数据。当设置一个znode节点的数据或读取时，znode节点的内容会被整个替换或全部读取进来。

* 1. znode的不同类型

当新建znode时，还需要指定该节点的类型（mode），不同的类型决定了znode节点的行为方式。

持久节点和临时节点

znode节点可以是持久（persistent）节点，还可以是临时

（ephemeral）节点。持久的znode，如/path，只能通过调用delete来进行删除。临时的znode与之相反，当创建该节点的客户端崩溃或关闭了与ZooKeeper的连接时，这个节点就会被删除。

持久znode是一种非常有用的znode，可以通过持久类型的znode为应用保存一些数据，即使znode的创建者不再属于应用系统时，数据也可以保存下来而不丢失.

临时znode传达了应用某些方面的信息，仅当创建者的会话有效时这些信息必须有效保存。临时节点不允许有子节点。

有序节点：一个znode还可以设置为有序（sequential）节点。一个有序znode节点被分配唯一个单调递增的整数。当创建有序节点时，一个序号会被追加到路径之后。

总之，znode一共有4种类型：持久的（persistent）、临时的（ephemeral）、持久有序的（persistent\_sequential）和临时有序的（ephemeral\_sequential）。

版本

每一个znode都有一个版本号，它随着每次数据变化而自增。两个API操作可以有条件地执行：setData和delete。这两个调用以版本号作为转入参数，只有当转入参数的版本号与服务器上的版本号一致时调用才会成功。

ZooKeeper架构

独立模式：

有一个单独的服务器，ZooKeeper状态无法复制。

总裁模式 ：

在仲裁模式下，ZooKeeper复制集群中的所有服务器的数据树。但如果让一个客户端等待每个服务器完成数据保存后再继续，延迟问题将无法接受。在公共管理领域，法定人数是指进行一项投票所需的立法者的最小数量。而在ZooKeeper中，则是指为了使ZooKeeper工作必须有效运行的服务器的最小数量。这个数字也是服务器告知客户端安全保存数据前，需要保存客户端数据的服务器的最小个数。例如，我们一共有5个ZooKeeper服务器，但法定人数为3个，这样，只要任何3个服务器保存了数据，客户端就可以继续，而其他两个服务器最终也将捕获到数据，并保存数据。

使用ZooKeeper:

Bin文件目录：以.sh结尾的脚本运行在UNIX平台，以.cmd结尾的脚本则用于Windows。

Conf目录：保存配置文件

Lib目录：包含了java的JAR文件，它们是运行ZooKeeper需要的第三方文件。

连接：

connectString

包含主机名和ZooKeeper服务器的端口。我们之前通过zkCli连接ZooKeeper服务时，已经列出过这些服务器。

sessionTimeout

以毫秒为单位，表示ZooKeeper等待客户端通信的最长时间，之后会声明会话已死亡。

watcher

用于接收会话事件的一个对象，这个对象需要我们自己创建。因为Wacher定义为接口，所以我们需要自己实现一个类，然后初始化这个类的实例并传入ZooKeeper的构造函数中。客户端使用Watcher接口来监控与ZooKeeper之间会话的健康情况。与ZooKeeper服务器之间建立或失去连接时就会产生事件。它们同样还能用于监控ZooKeeper数据的变化。最终，如果与ZooKeeper的会话过期，也会通过Watcher接口传递事件来

通知客户端的应用。(实现Watch)

NodeCreated

通过exists调用设置一个监视点。

NodeDeleted

通过exists或getData调用设置监视点。

NodeDataChanged

通过exists或getData调用设置监视点。

NodeChildrenChanged

通过getChildren调用设置监视点。