1. 什么是Zookeeper ：分布式服务框架hadoop的子项目 主要用于分布式中经常用到的数据管理

本质是分布式的小文件（存储数据小）存储系统（目录树结构的），zk之所以能够进行分布式协调服务，与他自身的存储文件的目录树是息息相关的。Z 可以对这些目录中的数据进行管理（增删改）和监控，通过数据的变化来达到对 分布式中数据的管理

如：集群管理、统一命名服务、分布式配置管理、分布式消息队列、分布式锁、分布式通知协调

使用场景 ： https://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/

（1）、配置管理 ：对于分布式的项目 ， 公共的配置信息（每台服务器上的配置都配置） 假如这些信息要更换，如果手动一个个的更换的话，可能会出问题 那么可以使用zk 当一个server中的配置变化时 ，那么所有的server配置都会同时发生变化

（2）统一命名服务 ：分布式中的接口命名再项目初始化的时候在zk中没创建（保存在zk中是以目录树结构的形式保存的 (不重复) 并且该目录下的文件是一个ip地址, 当远程调用次接口时 ，实际上是找到了远程的ip地址 然后通过rpc 远程调用）

（3）集群管理 ： 集群中可能会有server挂掉，如果某个请求被分配到这个挂掉的server中就会出现问题 为了确保的服务的可用性 ， 可以在集群的nginx和server上安装zk ， server 和 nginx保持连接创建临时节点 当服务器正常运行时创建临时节点 但是服务挂掉时该节点这自动删除 （keepalived 也能实现）

（4） 分布式锁

（5） 队列管理 ：

A、当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列

创建目录，当有请求进入时就创建临时节点 ，然后判断目录中的节点数， 如果成员数未达到，则线程wait()

B、生产者、消费者 FIFO（先进先出）

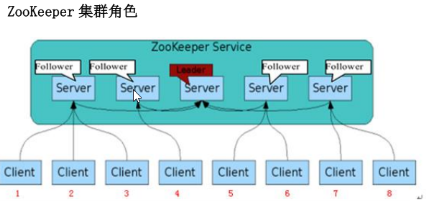
1. zk的特性
   1. 全局数据一致性：每台server上的数据都是一致的（文件夹下的数据都是一样的）
   2. 可靠性：如果消息（对目录中的数据进行增删改查）被其中一台服务器接受 ，那么将会被所有服务器接受然后一起操作
   3. 顺序性：包括全局有序 和 偏序。

假如再server1上先创建1.txt,再创建2.txt 。那么每台server创建文件夹的顺序都是一样的

* 1. 原子性：要么成功（成功的标准半数server节点成功，只要半数节点成功后面的节点会自动同步），要么失败
  2. 实时性：数据的实时

1. Zk中的集群角色（leader、follower、observer）： 分布式集群中：server承担着不同的角色， 角色承担不同的任务

如果访问量比较大的话，才会新增一个观察者的角色（提升项目的 非事物处理的能力）



Leader：处理事务性请求、调度follower 进行事物操作（对文件增删改）

（1）为了确保数据的一致性：请求分为事物性操作（对数据进行增删改） 和 非事务性操作

（2）当事物性操作发送请求给follower时，此时请求会被转发给leader，由leader进行全局的调度 。当发多个请求时并且是有顺序性的 ，leader还会对请求进行编号来确保数据的一致、顺序性

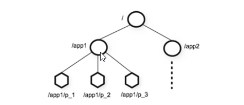
Follower：处理非事务性请求、参与集群中leader的选举，如果leader挂了，会再次选举一个

处理客户端的非事物性请求 (接受请求和选举)

Observer：处理非事务性请求、不会参与投票 （主要用于提升项目的 非事物处理的能力）

观察集群中节点的状态，并且实时同步数据。专门用于接收非事物请求，事物性请求转发给leader

1. 集群搭建 ： 由2n+1 奇数台集群（follower选举时，半数以上的投票才会产生leader） 偶数也行
2. Conf文件夹下zoo.cfg 文件 拷贝3 份
3. 配置： server.1 = ip地址:心跳端口:选举端口
4. 配置：myid文件的输出路径
5. 创建myid文件目录（创建名为myid无扩展名的文件，文件内容对应服务器的序号）
6. 再bin目录下拷贝3份zkServer.cmd 并配置zoo.cfg地址
7. Zookepper 数据模型



（1）Zk 存储数据的方式 跟操作系统存储文件的方式是一样的。 树形（节点）存储结构。 每一个存储节点 称为zNode

并且 每个节点 跟 常规的存储方式 有着不同之处

（2）Znode特点：

——每一个 节点都 兼具 文件 和 目录的 特点 （文件下还会有子目录）

——具有 原子 性操作

——大小有限制 kb 为单位

——文件的 项目引用 必须是 绝对路径 从根节点 / 开始

——序列化特性 ：如果再创建节点时指定此特性，zk会自动给此节点名字后追加一个序列号（序号是10位的数字从10000000000开始，如果有节点被创建则加1） 那么通过此特性就可以 判断 znode节点被创建的顺序

总结 每个Znode由3部分组成

Status 状态信息 ：描述该znode的版本和权限等信息

Data 数据 ：可以存储数据

Children 该节点如果关联目录的话，还可以关联子节点

1. ZNode的节点类型 ：　临时节点　　、　　永久节点

临时节点　：该临时节点 依赖他的会话（server与client的连接） ，一旦client断开与集群的连接 此节点就会被删除

注意：由于临时节点的特性 ，在该节点下不允许 创建子节点 （尤其是永久节点）

永久节点　：一旦该znode节点被创建 ，则会永久存在 除非手动命令删除

注意 ：永久节点不能修改 ， 只能删除 从新创建

1. 节点属性 （根据属性来判断 该Znode的相关信息 及 子节点相关信息 ）
   1. dataVersion数据版本号 ：每次节点进行 set操作 ，dataVersion值就会加1 。根据值来判断当前数据的新旧程度。 有效避免 数据更新时 出现先后顺序的问题
   2. cVersion 子节点版本号 ：当一个节点下的 孩子节点 发生变化 cversion的值就会加1 ，因此可以判断该节点下 的子节点 更新的 频繁程度
   3. ACLversion ：ACL 版本号 （了解） 安全认证 ，当其他客户端 访问zkserver ， 修改节点需要有认证

Zxid ：事物id ： zk 每进行一次 事务性的 操作 ，都会产生一个 全局的 唯一性的事务id（由leader 分配 id是有先后顺序的 ，因此 可以通过 事务id 来判断 事务性操作的 顺序 ） 这也是为什么事务性操作由 leader来完成

* 1. cZxid ：创建时的事务id
  2. mZxid ：修改时的事务id
  3. cTime ：节点创建的时间戳
  4. mTime ：节点修改的时间戳
  5. Ephemeralowner ： 如果该属性值 不为0 ， 则为临时节点 。 值为 临时节点的会话 sessionId

一旦客户端断开连接 ，该节点会自动删除

如果为 0 则为永久节点 。

1. 自带的zkShell 客户端 连接工具 操作 zk 集群
   1. 连接zk 集群 zkCli.sh -server ip地址 （linux） zkCli.cmd -server ip地址(windows)
      1. 如果不加 -server ip 客户端会到本机的默认2181端口 查找是否有zk集群 服务

如果加 则会到远程的服务去 连接 zk 集群服务

* + 1. 节点的增删该查
       1. Create [-s] [-e] path data acl 创建节点
          1. -s 可选 是否开启序列化节点 （默认不是序列化）
          2. -e 可选 是否开启临时节点 （默认永久节点）
          3. Path 创建节点的路径
          4. Data 数据
          5. Acl 权限控制
       2. 查询
          1. Ls path 节点下的所有子节点 （只能查看一级子节点）
          2. Get path 显示属性信息 和 保存的数据 和 子节点
          3. Ls2 path 只显示属性信息
       3. 更新 节点一旦被创建， 类型不能改变 但是数据是可以变动的 set path data version
          1. Date 修改的数据 、 version版本号 可以不指定
       4. 删除
          1. Delete path 如果删除的节点 存在子节点，不能进行删除操作 必须删除子节点
          2. Delete path rmr 强制递归删除

1. 常用命令
   1. quit 退出客户端
   2. setQuota 对节点增加限制 setquota -n | -b val path （注意：n | b 不能两个同时使用）
      1. 参数n：限制子节点的最大个数
      2. 参数b: 限制数据值的最大长度
      3. 参数vla：子节点最大个数 或 数据值的最大长度  
         注意 ： I、当对此节点设置最大子节点时 ， 节点本身也会算进去。

如： 对A 节点 的子节点设置为3 ， 此时A 节点自身算是一个节点 。 A节点下最多还能添加2 个子节点

II、如果子节点 个数超过 设置节点的 大小 ， 也不会报错 只会在日志中打一个警告

* 1. Listquota path 查询该节点是否进行 quota限制
  2. Delquota -n|-b path 删除节点的 quota限制 count的值为-1 表示大小无限制
  3. History 列出 历史操作命令
     1. 历史操作命令 会按照序号 显示出来
     2. 如果想重复执行 以上命令 redo 序号

1. Zk watcher 监听机制

Zk提供了分布式数据发布 / 订阅功能 （如我定了天气预报的业务 ，监听 监听到 那么当天气预报出来时 就会自动的吧天气预报的信息 推送给我） 分布式的服务 也有这种情况 ，为了解决这些类似的事情 zk 引入了wather机制。 zk允许 客户端向 服务端 注册一个监听 ， 监听感兴趣的事件（如：节点的创建、节点的变动、删除节点等） 一旦监听的事件发生了 ，那么设置的监听就会被触发 。 那么服务端 就会以 事件的形式 通知给客户端 来完成分布式数据发布 / 订阅功能

1. watcher 机制的实现过程
   * 1. 注册监听事件
     2. 一旦事件发生， 监听触发
     3. 服务器以事件的形式告诉客户端，客户端能够回调watcher 获取事件情况
2. watcher 机制的特点
   * 1. 一次性触发 ：同一种情况触发只实现一次 ， 如果数据再次发生变化则不会触发，监听会失效
     2. 事件的封装 ：zk 使用watcherEvent对象来封装服务端事件 ，并传递给客户端 。watcherEvent对象包含3个属性
        1. keeperStauts 通知的状态
        2. eventTpye 事件类型
        3. Path 事件路径 （当前事件发生在哪个路径下）

客户端那么客户端 可以 到watcherEvent中 根据 这三个属性 获取相应的

1. Event 异步发送 ： 事件 由 服务器 异步 发送到 客户端
2. 先注册 再触发 ：想要事件能够触发 必须先注册

1. Shell 客户端 实现 watcher机制

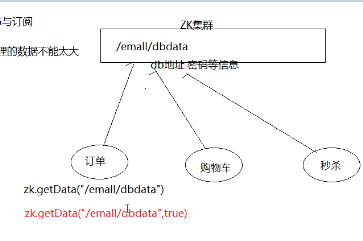
技巧 ： help 会列出所有 的命令， 其中带 watch 是跟 watcher机制有关的

* 1. 客户端1， 设置 节点变动 监听 : get path watch
  2. 当其他客户端操作使其节点数据 发生变化时
  3. 此时 客户端1 就会接收到 watcheEvent
  4. 如果再次 改变此节点数据 ， 则 不会发送通知

1. Java API 操作zk 集群

通过封装的 Zookeepe实例的对象可以连接zk服务器，并且操作znode节点

1. 、创建节点 create
2. 获取节点 getData
   1. 参数1： path
   2. 参数2：是否对此节点设置监听 ture 监听此节点 ，false 不监听此节点
   3. 参数3：状态 可以为null
3. 修改 ：setData
   1. 参数1 ：path
   2. 参数2：修改的数据
   3. 参数3：修改的版本 ， 对应当前版本 。 -1 表示版本维护交给服务器
4. Zk 的选举机制 ： 如何判断 哪台服务器做follower 哪台服务器做leader
   1. 采用fastLeaderElection采用投票数大于半数则胜
      1. 每个人都给自己投票
      2. 投票数过半 选举结束
5. zk数据发布 与 订阅模型 ： 全局的配置中心（全局管理的数据）既： 发布者将数据发布到 zk全局的配置中心 的节点上，供订阅者 动态的订阅数据 ，(多台客户端同时订阅)订阅者也可以设置监听 或更改这些数据 。实现配置信息的集中式管理 和 动态 更新



分布式的电商 ， 订单、购物车等模块不在同一个服务器 来实现 连接数据的数据的发布 和 订阅

1. 再服务启动的时候 连接zk 查询配置文件信息 zk.getData(“path”,true,null); 同时对此文件设置监听
2. 当配置文件改变时 ， 触发监听 通知客户端 ， 然后再次对文件 设置监听

这样就能完成 全局的消息发布和订阅功能

1. Zk 命名服务

在zk中，通过命名服务 客户端应用能根据指定的 名字 来获取资源 或 服务地址。 被命名的实体可以使 集群中的机器，可以是远程对象等等

dubbo通过使用zk 来作为其命名服务 。 因为zk中路径是唯一的，不存在两个一摸一样的路径， 当dubbo发布服务（ip路径或地址）的时候，此服务就会被注册到 zk中 ，形成 一对一的 关系 可以根据一方找到另一位方 从而完成服务的注册 。

1. Zk 分布式锁

类似多线程 ，多个线程操作同一资源时 可能会导致数据的不一致 。 分布式服务也会出现这种情况

锁服务分为两类 ： 保持独占 和 控制时序

持独占：只能有一个 客户端 能持有这把锁 ，当多个客户端去 获取同一个资源时 ，zk集群强制要求所有客户端先去zk上指定的目录创建节点（节点要求是一个非序列化的临时节点） 谁创建 成功 ，谁就可以获取访问数据的权限 当操作完成，客户端断开与zk的连接 节点就会被删除，其他客户端如果也需要操作这个文件的话，就监听上个文件创建的节点 ， 一旦断开连接 节点删除，其他客户端就会收到通知 然后再在这个文件下创建相同节点 ，然后就获得了 访问资源的权限

这样就 保证了 只有一个 应用能获取 一个资源的 能力

控制时序: 利用序列化的特性 。客户端创建 序列化的临时节点， 这样创建的节点 就会有一个顺序 。 那么可以根据这个序列号 进行优先级访问