1. ZooKeeper概述

专业解释：分布式协调服务组件

通俗的解释：劝架者，仲裁机构

作用：如果有多个角色出现了分歧，没法达成一致，ZooKeeper 帮我们达成一致

看官网吧：https://zookeeper.apache.org/

> What is ZooKeeper?
>
> **Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination**
>
> ZooKeeper is a centralized service for **maintaining configuration information**, **naming**, **providing distributed synchronization**, and **providing group services**. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them ,which make them brittle in the presence of

> change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed

ZooKeeper 是一个分布式的，开放源码的**分布式应用程序协调服务**，是 Google 的 Chubby 一个开源的实现。它提供了简单原始的功能，分布式应用可以基于它实现更高级的服务，比如**分布式同步，配置管理，集群管理，命名管理，队列管理。**它被设计为易于编程，使用文件系统目录树作为数据模型。服务端跑在 Java 上，提供 Java 和 C 的客户端 API。

官网地址：http://ZooKeeper.apache.org/
官网快速开始地址：http://zookeeper.apache.org/doc/current/zookeeperStarted.html
官网API地址：http://ZooKeeper.apache.org/doc/r3.4.10/api/index.html

ZooKeeper在分布式领域，能够帮助解决很多很多的分布式难题，但是底层却只是依赖于两个主要的组件：

```
znode文件/数据存储系统：为分布式应用存储少量关键的核心状态数据
watch监听系统：如果某个客户端对这个znode系统中的某个数据感兴趣（如果发生了什么变化，我想知道），zookeeepr可以提供一个监听服务，客户端去注册针对某个数据的监听，如果这个数据真的发生了变化。zookeeper会立即推送消息给客户端。
```

```
数据：data  zk的专业术语：znode
zookeeper服务器
client客户端
```

另外其实还有一大模块：就是 ACL 系统，但是对于我们现在了解 ZooKeeper 的核心功能，用处不大。暂不详解。

2. ZooKeeper的znode存储系统解密

文件系统！ （树形结构，文件夹，文件）

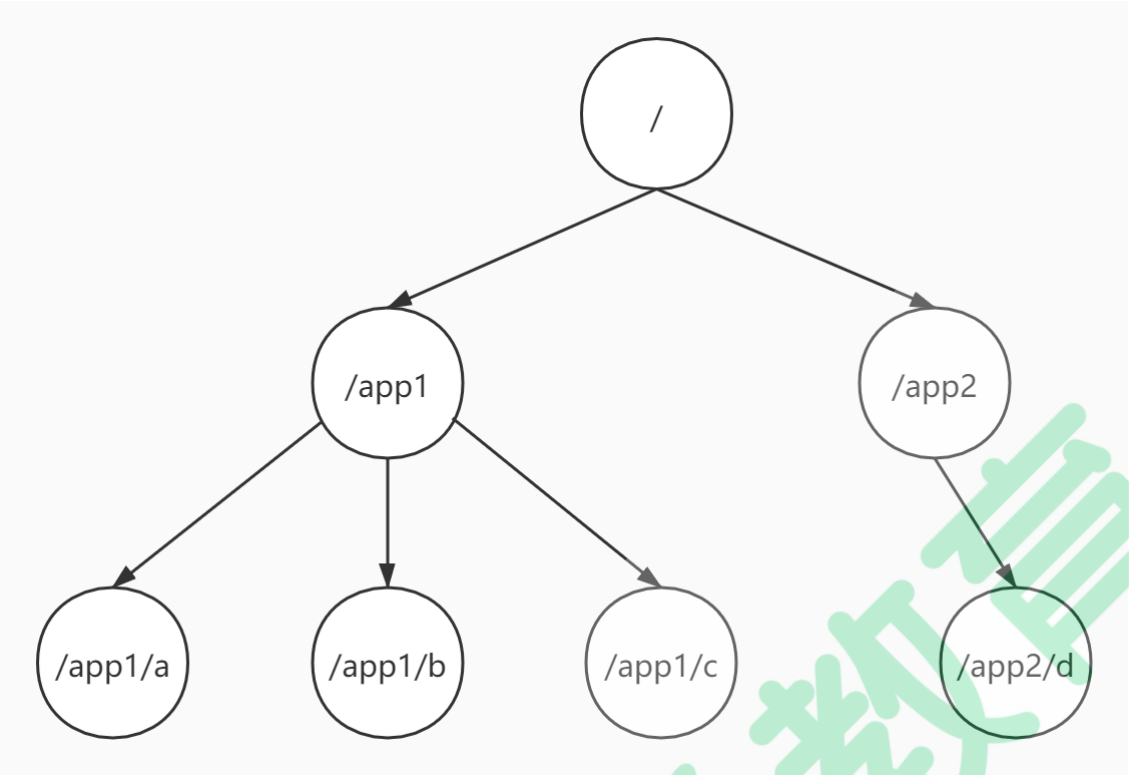
```
树形结构：文件系统的元数据
文件夹：区分管理不同种类型的文件，不能！挂载子节点（文件夹，文件）
文件：真是存储数据的载体，不能挂载子节点
```

zookeeper的文件系统：

```
树形结构
没有了文件夹和文件的区别，就叫做 znode（既可以认为是文件，也可以认为是文件夹，也可以认为不是文件，也可以认为不是文件夹）：这个znode既能挂载子节点，也能存储数据
```

ZooKeeper 的命名空间就是 ZooKeeper 应用的文件系统，它和 Linux 的文件系统很像，也是树状，这样就可以确定每个路径都是唯一的，对于命名空间的操作必须都是绝对路径操作。与Linux文件系统不同的是，Linux 文件系统有目录和文件的区别，而 ZooKeeper 统一叫做 znode，一个 znode 节点下可以包含子 znode，同时该节点也可以存储数据。

所以总结说来，**znode 即是文件夹又是文件的概念，但是在 ZooKeeper 这里面就不叫文件也不叫文件夹，叫 znode，每个 znode 有唯一的路径标识，既能存储数据，也能创建子 znode。但是 znode 只适合存储非常小量的数据，不能超过1M，最好小于1K。**



关于 znode 的分类：

- 按照生命周期可以分为：
  - 短暂（ephemeral）（断开连接自己删除）
  - 持久（persistent）（断开连接不删除，默认情况）
- 按照是否自带序列编号可以分为：
  - SEQUENTIAL（带自增序列编号，由父节点维护）
  - 非SEQUENTIAL（不带自增序列编号，默认情况）

详细解释：

节点类型	详解
PERSISTENT	持久化 znode 节点，一旦创建这个 znode 节点，存储的数据不会主动消失，除非是客户端主动 delete
PERSISTENT_SEQUENTIAL	自动增加自增顺序编号的 znode 节点，比如 ClientA 去 zookeeper service 上建立一个 znode 名字叫做 /zk/conf，指定了这种类型的节点后zk会创建 /zk/conf0000000000，ClientB 再去创建就是创建 /zk/conf0000000001，ClientC 是创建 /zk/conf0000000002，以后任意 Client 来创建这个 znode 都会得到一个比当前 zookeeper 命名空间最大 znod e编号 +1 的 znode，也就说任意一个 Client 去创建 znode 都是保证得到的 znode 编号是递增的，而且是唯一的 znode 节点
EPHEMERAL	临时 znode 节点，Client 连接到 zk service 的时候会建立一个 session，之后用这个 zk 连接实例在该 session 期间创建该类型的 znode，一旦 Client 关闭了 zookeeper 的连接，服务器就会清除session，然后这个 session 建立的 znode 节点都会从命名空间消失。总结就是，这个类型的 znode 的生命周期是和 Client 建立的连接一样的。比如 ClientA 创建了一个 EPHEMERAL 的 /zk/conf 的 znode 节点，一旦 ClientA 的 zookeeper 连接关闭，这个 znode 节点就会消失。整个zookeeper service命名空间里就会删除这个znode节点
EPHEMERAL_SEQUENTIAL	临时自动编号节点，znode 节点编号会自动增加，但是会随 session 消失而消失

注意点：

- 创建 znode 时设置顺序标识，znode 名称后会附加一个值，顺序号是一个单调递增的计数器，由父节点维护
- 在分布式系统中，顺序号可以被用于为所有的事件进行全局排序，这样客户端可以通过顺序号推断事件的顺序
- EPHEMERAL 类型的节点不能有子节点，所以只能是叶子结点
- 客户端可以在 znode 上设置监听器

3. ZooKeeper的监听机制解密

3.1. 概述

客户端注册监听它关心的目录节点，当目录节点发生变化（**数据改变、节点删除、子目录节点增加删除**）时，ZooKeeper 会通知客户端。监听机制保证 ZooKeeper 保存的任何的数据的任何改变都能快速的响应到监听了该节点的应用程序。

监听器的工作机制，其实是在客户端会专门创建一个监听线程，在本机的一个端口上等待 ZooKeeper 集群发送过来事件。

1、那些客户端
2、那些znode节点
3、什么变化

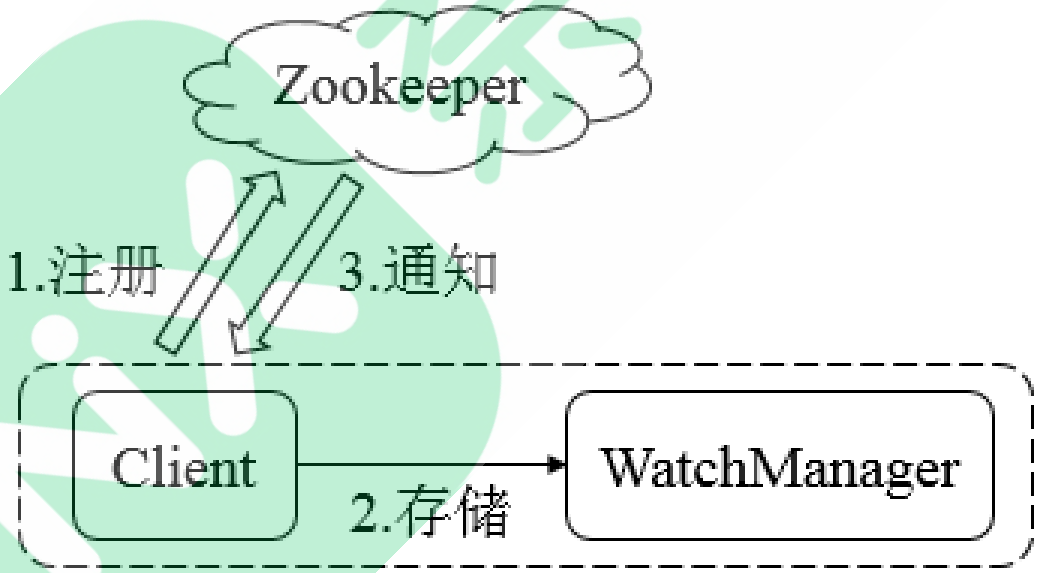
1、有三种注册监听的方式
2、有五种触发监听的方式
3、总共有四种类型的事件

创建Watch的API	触发事件				
	Create		Delete		setData
	znode	child	znode	child	znode
exists()	NodeCreated		NodeDeleted		NodeDataChanged
getData()			NodeDeleted		NodeDataChanged
getChildren()		NodeChildrenChanged	NodeDeleted	NodeChildrenChanged	NodeDataChanged

NodeCreated	创建节点时触发该事件
NodeChildrenChanged	当前节点下创建或者删除子节点触发该事件，修改子节点的数据不触发该事件
NodeDeleted	节点删除时触发该事件
NodeDataChanged	当前节点的数据被修改的时候触发该事件

注意：注册的监听在事件响应之后就失效了。那么怎么做到连续监听？请思考回答。

监听工作原理：ZooKeeper 的 Watcher 机制主要包括客户端线程、客户端 WatcherManager、ZooKeeper 服务器三部分。客户端在向 ZooKeeper 服务器注册的同时，会将 Watcher 对象存储在客户端的 WatcherManager 当中。当 ZooKeeper 服务器触发 Watcher 事件后，会向客户端发送通知（WatchedEvent），客户端线程从 WatcherManager 中取出对应的 Watcher 对象来执行回调逻辑。



文字描述：

1、如果某个客户端A对zookeeper系统中的某个节点/a1/b1的数据变化便感兴趣，需要注册一个监听，zookeeper.getData(Watch)
2、客户端会生成一个WatchManager服务，专门用来管理各种Watch对象
3、等待响应
4、如果zookeeper系统真的检测到你A感兴趣的znode节点/a1/b1发生了数据变化。那么zookeeper就会返回一个消息：WatchedEvent，包含了三个信息：Path,EventType,connect
5、客户端就会根据这个 WatchedEvent 的信息，来决定回调那个 Watch 对象中的 process方法
void process(WatchedEvent event)
6、在进行开发的时候，都会提前把对应的业务逻辑都编写在 process 方法中。

3.2. 总结

zookeeper只为我们提供了两个功能：

```
1、znode系统，存储一些关键数据，类似于unix文件系统
2、watch监听机制，可以让客户端有能力实时感知zookeeper代为保管的数据的变化，从而进行相应处理。
```

4. ZooKeeper应用场景解密

4.1. 发布/订阅

应用服务器集群可能存在两个问题： 1、因为集群中有很多机器，当某个通用的配置发生变化后，怎么自动让所有服务器的配置同一生效？ 2、当集群中某个节点宕机，如何让集群中的其他节点知道？ 为了解决这两个问题，zk引入了watcher机制来实现发布/订阅功能，能够让多个订阅者同时监听某一个主题对象，当这个主题对象自身状态发生变化时，会通知所有订阅者。

数据发布/订阅即所谓的配置中心：发布者将数据发布到zk的一个或一些列节点上，订阅者进行数据订阅，可以即时得到数据的变化通知。

发布/订阅有2种设计模式，推Push & 拉Pull。在推模中，服务端将所有数据更新发给订阅的客户端，而拉是由客户端主动发起请求获取最新数据。通常采用轮寻。

zk采用推拉结合，客户端向服务端注册自己需要关注的节点，一旦该节点数据发生变更，服务器像客户端发送Watcher事件通知，收到消息主动向服务端获取最新数据。**这种模式主要用于配置信息获取同步。**

A有一条消息，要让B知道：A主动告诉B， B不断的来询问有没有新的消息：如果有A就会告诉他

4.2. 命名服务

zookeeper 系统中的每个 znode 都有一个绝对唯一的路径！所以只要你创建成功了一个znode节点，也就意味着，你命名了一个全局唯一的 名称！

命名服务是分布式系统中较为常见的一类场景，分布式系统中，被命名的实体通常可以是集群中的机器、提供的服务地址或远程对象等，通过命名服务，客户端可以根据指定名字来获取资源的实体、服务地址和提供者的信息。ZooKeeper也可帮助应用系统通过资源引用的方式来实现对资源的定位和使用，广义上的命名服务的资源定位都不是真正意义上的实体资源，在分布式环境中，上层应用仅仅需要一个全局唯一的名字。**ZooKeeper可以实现一套分布式全局唯一ID的分配机制。**

由于zk可以创建顺序节点，保证了同一节点下子节点是唯一的，所以直接按照存放文件的方法，设置节点，比如一个路径下不可能存在两个相同的文件名，这种定义创建节点，就是全局唯一ID

4.3. 配置管理

程序总是需要配置的，如果程序分散部署在多台机器上，要逐个改变配置就变得困难。现在把这些配置全部放到ZooKeeper上去，保存在 ZooKeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 ZooKeeper 的通知，然后从 ZooKeeper 获取新的配置信息应用到系统中就好。

我是老师
你们是学员

```
方式1：我直接每个人挨个儿发送通知
方式2：公告栏：我把消息发布在公告栏。你们谁感兴趣就去获取！

我是客户端，发送消息到zookeeper
你们也是客户端，对我发送的消息是感兴趣的，你们会去zookeeper先打个招呼。
如果有我发送的消息到zookeeper，那么zookeeper就会告诉你们
```

```
客户端发送消息到zk一次即可！
集群中的所有的服务器，都监听。都能立即受到关于这个消息的通知
```

4.4. 集群管理

所谓集群管理无在乎两点：**是否有机器退出和加入、选举master。**

对于第一点，所有机器约定在父目录 GroupMembers 下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与 ZooKeeper 的连接断开，其所创建的代表该节点存活状态的临时目录节点被删除，所有其他机器都将收到通知：某个兄弟目录被删除，于是，所有人都知道：有兄弟节点挂掉了。新机器加入也是类似，所有机器收到通知：新兄弟目录加入，又多了个新兄弟节点。

对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为 master 就好。当然，这只是其中的一种策略而已，选举策略完全可以由管理员自己制定。在分布式环境中，相同的业务应用分布在不同的机器上，有些业务逻辑（例如一些耗时的计算，网络I/O处理），往往只需要让整个集群中的某一台机器进行执行，其余机器可以共享这个结果，这样可以大大减少重复劳动，提高性能。

利用ZooKeeper的强一致性，能够保证在分布式高并发情况下节点创建的全局唯一性，即：同时有多个客户端请求创建 /currentMaster 节点，最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很轻易的在分布式环境中进行集群选取了。（其实只要实现数据唯一性就可以做到选举，关系型数据库也可以，但是性能不好，设计也复杂）

4.5. 分布式锁

有了 ZooKeeper 的一致性文件系统，锁的问题变得容易。

锁服务可以分为两三类： 独占锁，共享锁，时序锁

```
一个是写锁，对写加锁，保持独占，或者叫做排它锁，独占锁
一个是读锁，对读加锁，可共享访问，释放锁之后才可进行事务操作，也叫共享锁
一个是控制时序，叫时序锁
```

```
编写java代码的时候：synchronized    lock   单进程多线程的环境
复杂到分布式环境中呢？  上海的A 和 广州的B 也做同一件事。分布式锁。
```

对于第一类，我们将 ZooKeeper 上的一个znode看作是一把锁，通过 createznode() 的方式来实现。所有客户端都去创建 /distribute_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除掉自己创建的 /distribute_lock 节点就释放出锁。

对于第二类，/distribute_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选 Master 一样，编号最小的获得锁，用完删除，依次有序

4.6. 队列管理

两种类型的队列：

```
1、同步队列/分布式屏障：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。
2、先进先出队列：队列按照 FIFO 方式进行入队和出队操作。
```

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。
第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。

4.7. 负载均衡

ZooKeeper 实现负载均衡本质上是利用zk的配置管理功能，实现负载均衡的步骤：

1. 服务提供者把自己的域名及IP端口映射注册到zk中。
2. 服务消费者通过域名从zk中获取到对应的IP及端口，这里的IP及端口可能有多个，只是获取其中一个。
3. 当服务提供者宕机时，对应的域名与IP的对应就会减少一个映射。
4. 阿里的dubbo服务框架就是基于zk实现服务路由和负载。

5. ZooKeeper Shell 使用

常用命令详解：

命令	作用
ls / ls /ZooKeeper	查看znode子节点列表
create /zk "myData"	创建znode节点
get /zk get /zk/node1	获取znode数据
set /zk "myData1"	设置znode数据
ls /zk watch	就对一个节点的**子节点变化事件**注册了监听
get /zk watch	就对一个节点的**数据内容变化事件**注册了监听
create -e /zk "myData"	创建临时znode节点
create -s /zk "myData"	创建顺序znode节点
create -e -s /zk "myData"	创建临时的顺序znode节点
delete /zk	只能删除没有子znode的znode
rmr /zk	不管里头有多少znode，统统删除
status /zk	查看/zk节点的状态信息

6. ZooKeeper API 使用

详见代码实现

7. 基于ZooKeeper实现服务发布订阅

详见代码实现

8. 基于ZooKeeper实现服务器动态上下线感知

详见代码实现！

9. 基于ZooKeeper实现分布式锁

详见代码实现！

10. 基于ZooKeeper实现分布式选举算法

详见代码实现!

11. 基于ZooKeeper实现同步队列

详见代码实现!

12. 基于典型常用应用场景的实现思路详解

核心要点:

1、利用 ZooKeeper 提供的 数据存储系统的 znode 节点来存储状态信息

2、ZooKeeper 可以把客户端感兴趣的哪个节点发生什么事件的信息推送给过来。让客户端立即感知状态变化

3、ZooKeeper提供了四种事件类型:

```
NodeCreated, NodeDeleted, NodeDataChanged, NodeChildrenChanged
```

4、ZooKeeper 提供了三种添加监听的方式

```
zookeeper.exists()
zookeeper.getData()
zookeeper.getChildren()
```

5、ZooKeeper 提供了对应的触发监听的方法:改变 znode 的状态, zookeeper就会立即通知给 对应的注册了监听的客户端

```
zookeeper.create()
zookeeper.delete()
zookeeper.setData()
```

6、核心:记住一张图!

创建Watch的API	触发事件				
	Create		Delete		setData
	znode	child	znode	child	znode
exists()	NodeCreated		NodeDeleted		NodeDataChanged
getData()			NodeDeleted		NodeDataChanged
getChildren()		NodeChildrenChanged	NodeDeleted	NodeChildrenChanged	NodeDataChanged

NodeCreated	创建节点时触发该事件
NodeChildrenChanged	当前节点下创建或者删除子节点触发该事件，修改子节点的数据不触发该事件
NodeDeleted	节点删除时触发该事件
NodeDataChanged	当前节点的数据被修改的时候触发该事件