场景一：**统一命名服务**。有一组服务器向客户端提供某种服务（例如：我前面做的分布式网站的服务端，就是由四台服务器组成的 集群，向前端集群提供服务），我们希望客户端每次请求服务端都可以找到服务端集群中某一台服务器，这样服务端就可以向客户端提供客户端所需的服务。对于这 种场景，我们的程序中一定有一份这组服务器的列表，每次客户端请求时候，都是从这份列表里读取这份服务器列表。那么这分列表显然不能存储在一台单节点的服 务器上，否则这个节点挂掉了，整个集群都会发生故障，我们希望这份列表时高可用的。高可用的解决方案是：这份列表是分布式存储的，它是由存储这份列表的服 务器共同管理的，如果存储列表里的某台服务器坏掉了，其他服务器马上可以替代坏掉的服务器，并且可以把坏掉的服务器从列表里删除掉，让故障服务器退出整个 集群的运行，而这一切的操作又不会由故障的服务器来操作，而是集群里正常的服务器来完成。这是一种主动的分布式数据结构，能够在外部情况发生变化时候主动 修改数据项状态的数据机构。Zookeeper框架提供了这种服务。这种服务名字就是：统一命名服务，它和javaEE里的JNDI服务很像,也和网址对应ip的DNS服务器类似。

场景二：**分布式锁服务**。当分布式系统操作数据，例如：读取数据、分析数据、最后修改数据。在分布式系统里这 些操作可能会分散到集群里不同的节点上，那么这时候就存在数据操作过程中一致性的问题，如果不一致，我们将会得到一个错误的运算结果，在单一进程的程序 里，一致性的问题很好解决，但是到了分布式系统就比较困难，因为分布式系统里不同服务器的运算都是在独立的进程里，运算的中间结果和过程还要通过网络进行 传递，那么想做到数据操作一致性要困难的多。Zookeeper提供了一个锁服务解决了这样的问题，能让我们在做分布式数据运算时候，保证数据操作的一致 性。

场景三：**配置管理**。在分布式系统里，我们会把一个服务应用分别部署到n台服务器上，这些服务器的配置文件是 相同的（例如：我设计的分布式网站框架里，服务端就有4台服务器，4台服务器上的程序都是一样，配置文件都是一样），如果配置文件的配置选项发生变化，那 么我们就得一个个去改这些配置文件，如果我们需要改的服务器比较少，这些操作还不是太麻烦，如果我们分布式的服务器特别多，比如某些大型互联网公司的 hadoop集群有数千台服务器，那么更改配置选项就是一件麻烦而且危险的事情。这时候zookeeper就可以派上用场了，我们可以把 zookeeper当成一个高可用的配置存储器，把这样的事

情交给zookeeper进行管理，我们将集群的配置文件拷贝到zookeeper的文件系统的某个节点上，然后用zookeeper监控所有分布式系统里配置文件的状态，一旦发现有配置文件发生了变化，每台服务器都会收到zookeeper的通 知，让每台服务器同步zookeeper里的配置文件，zookeeper服务也会保证同步操作原子性，确保每个服务器的配置文件都能被正确的更新。

场景四：**为分布式系统提供故障修复的功能(领导者选举)**。集群管理是很困难的，在分布式系统里加入了zookeeper服务，能让我们很容易的对集群进行管理。集群管理最麻烦的事情就是节点故障管理，zookeeper可以让集群选出一个健康的节点作为 master，master节点会知道当前集群的每台服务器的运行状况，一旦某个节点发生故障，master会把这个情况通知给集群其他服务器，从而重新分配不同节点的计算任务。Zookeeper不仅可以发现故障，也会对有故障的服务器进行甄别，看故障服务器是什么样的故障，如果该故障可以修复，zookeeper可以自动修复或者告诉系统管理员错误的原因让管理员迅速定位问题，修复节点的故障。大家也许还会有个疑问，master故障了，那 怎么办了？ zookeeper也考虑到了这点，zookeeper内部有一个“选举领导者的算法”，master可以动态选择，当master故障时 候，zookeeper能马上选出新的master对集群进行管理。

- 1. 启动ZK服务:　　sh bin/zkServer.sh start
- 2. 查看ZK服务状态: sh bin/zkServer.sh status
- 3. 停止ZK服务:　　sh bin/zkServer.sh stop
- 4. 重启ZK服务:　　sh bin/zkServer.sh restart
- 5　启动ZK客户端　　sh zkCli.sh -server 127.0.0.1:2181
-
- 6 显示根目录下、文件： ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容
- 7. 显示根目录下、文件： ls2 / 查看当前节点数据并能看到更新次数等数据
- 8. 创建文件,并设置内容: create /zk "test" 创建一个znode节点“ zk ”以及与它关联的字符串
- 9. 获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串
- 10. 修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置
- 11. 删除文件： delete /zk 将刚才创建的 znode 删除
- 12. 退出客户端： quit
- 13. 帮助命令： help

ZooKeeper 常用四字命令：

ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用

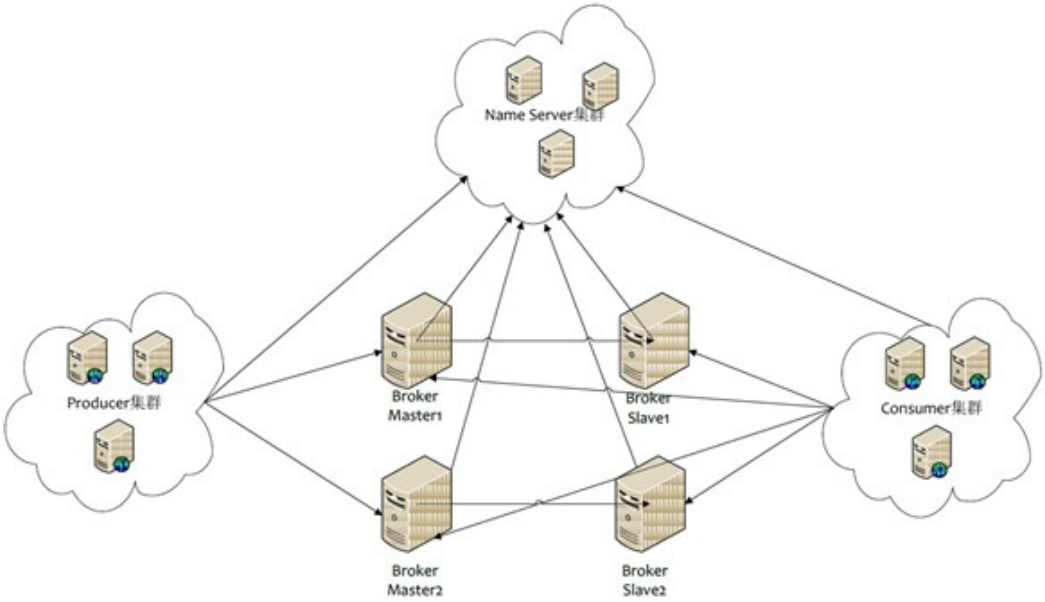
来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令

- 1. 可以通过命令：echo stat|nc 127.0.0.1 2181 来查看哪个节点被选择作为follower或者leader
- 2. 使用echo ruok|nc 127.0.0.1 2181 测试是否启动了该Server，若回复imok表示已经启动。
- 3. echo dump| nc 127.0.0.1 2181 ,列出未经处理的会话和临时节点。
- 4. echo kill | nc 127.0.0.1 2181 ,关掉server
- 5. echo conf | nc 127.0.0.1 2181 ,输出相关服务配置的详细信息。
- 6. echo cons | nc 127.0.0.1 2181 ,列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。
- 7. echo envi |nc 127.0.0.1 2181 ,输出关于服务环境的详细信息（区别于 conf 命令）。
- 8. echo reqs | nc 127.0.0.1 2181 ,列出未经处理的请求。
- 9. echo wchs | nc 127.0.0.1 2181 ,列出服务器 watch 的详细信息。
- 10. echo wchc | nc 127.0.0.1 2181 ,通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表。
- 11. echo wchp | nc 127.0.0.1 2181 ,通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径。

https://yq.aliyun.com/articles/2892

metaq架构&消息的收发

metaq的整体架构如下图所示，主要包括Broker集群（metaq的服务端），client集群（发布者集群和订阅者集群），nameServer集群。



Broker分为master和slave。每个Broker与nameserver集群(zookeeper服务器)中的所有节点建立长连接，定时注册topic信息到所有的nameServer。Producer与nameServer集群(zookeeper服务器)中的一个节点（随机）建立长连接，定期从nameServer(zookeeper服务器) 取topic路由信息，并向提供topic服务的master broker建立长连接，且定时向master发送心跳。Producer发布消息是发布到master，在由master同步到所有broker。Consumer与nameServer集群(zookeeper服务器)中的一个节点建立长连接，定期从nameServer(zookeeper服务器)取topic的路由信息，并向提供topic服务的master、slave broker建立长连接，并定时向master、slave发送心跳。Comsumer既可以从slave订阅消息，也可以从master订阅消息。

保证消息的可靠性

一个消息从发送端应用，到消费端应用，中间有三个过程需要保证消息的可靠性。

1.发送端发消息

消息生产者发送消息后返回SendResult，如果isSuccess返回为true,则表示消息已经确认发送到服务器并被服务器接收存储。整个发送过程是一个同步的过程。保证消息送达服务器并返回结果。

只有当消息中间件及时明确的返回成功，才能确认消息可靠到达消息中间件。

2.消息中间件把消息存储起来

metaq服务器收到消息后首先把消息存放在磁盘文件中，确保持久存储，写入成功之后返回应答给发布者。因此，可以确认每条发送结果为成功的消息服务器都是写入磁盘的。

内存中内容属于非持久数据，会在断电之后丢失。

3.消费端消费消息

消费者是一条接着一条地顺序消费消息，只有在成功消费一条消息后才会接着消费下一条。

如果在消费某条消息失败（如异常），则会尝试重试消费这条消 息（默认最大5次），超过最大次数后仍然无法消费，则将消息存储在消费者的本地磁盘，由后台线程继续做重试。而主线程继续往后走，消费后续的消息。。由此来保证消息的可靠消费。

用ZooKeeper的应用程序：

- Apache Hadoop 依靠 ZooKeeper 来实现 Hadoop HDFS NameNode 的自动故障转移，以及 YARN ResourceManager 的高可用性。
- Apache HBase 是构建于 Hadoop 之上的分布式数据库，它使用 ZooKeeper 来实现区域服务器的主选举（master election）、租赁管理以及区域服务器之间的其他通信。
- Apache Accumulo 是构建于 Apache ZooKeeper（和 Apache Hadoop）之上的另一个排序分布式键/值存储。
- Apache Solr 使用 ZooKeeper 实现领导者选举和集中式配置。
- Apache Mesos 是一个集群管理器，提供了分布式应用程序之间高效的资源隔离和共享。Mesos 使用 ZooKeeper 实现了容错的、复制的主选举。
- Neo4j 是一个分布式图形数据库，它使用 ZooKeeper 写入主选择和读取从协调（read slave coordination）。
- Cloudera Search 使用 ZooKeeper（通过 Apache Solr）集成了搜索功能与 Apache Hadoop，以实现集中式配置管理。
- Dubbo
- MetaQ
- ==========