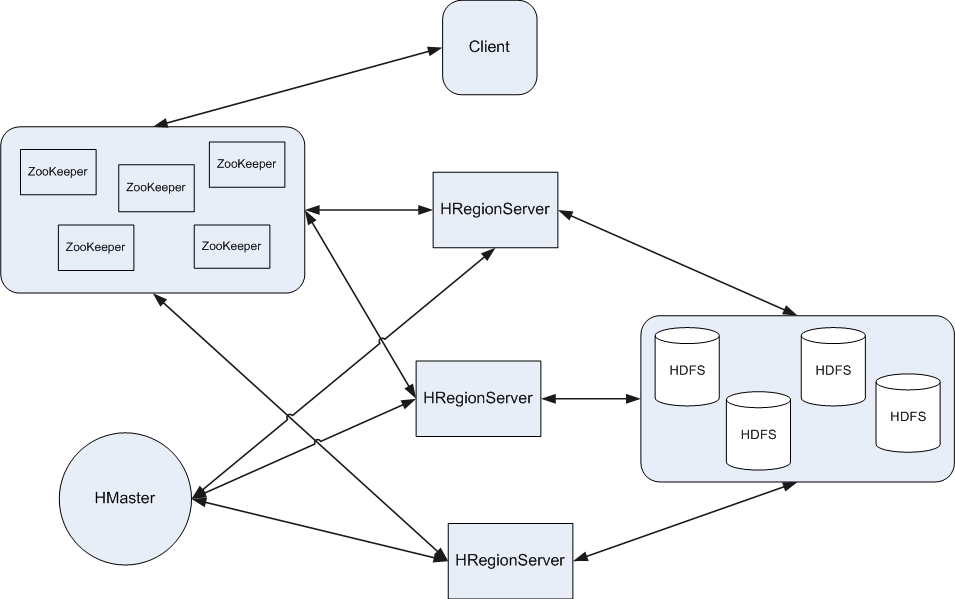
# Zookeeper

## 是什么?

ZooKeeper 顾名思义 动物园管理员，他是拿来管**[大象(Hadoop)](http://hadoop.apache.org/" \t "_blank)** 、 [**蜜蜂(Hive)**](http://hive.apache.org/) 、**[小猪(Pig)](http://pig.apache.org/" \t "_blank)**  的管理员， Apache Hbase和 Apache Solr 以及**[LinkedIn sensei](http://sna-projects.com/sensei" \t "_blank)** 等项目中都采用到了 Zookeeper。ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，ZooKeeper是以**[Fast Paxos](http://rdc.taobao.com/blog/cs/?p=261" \t "_blank)**算法为基础，实现同步服务，配置维护和命名服务等分布式应用。

Zookeeper/Hbase/Hadoop三者之间的关系，在此我把三者之间的关系画在一张图上希望能表达的清楚一些。

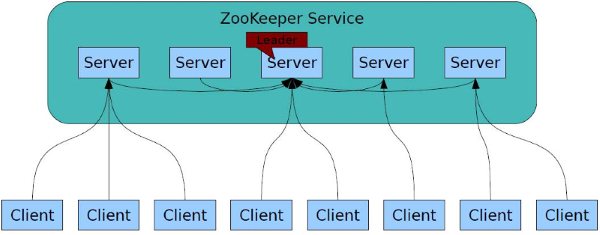


## **ZooKeeper 如何工作？**

ZooKeeper是作为分布式应用建立更高层次的同步(synchronization)、配置管理 (configuration maintenance)、群组(groups)以及名称服务(naming)。在编程上，ZooKeeper设计很简单，所使用的数据模型风格很像文件系统的目录树结构，简单来说，有点类似windows中注册表的结构，有名称，有树节点，有Key(键)/Value(值)对的关系，可以看做一个树形结构的数据库，分布在不同的机器上做名称管理。

   Zookeeper分为2个部分：服务器端和客户端，客户端只连接到整个ZooKeeper服务的某个服务器上。客户端使用并维护一个TCP连接，通过这个连接发送请求、接受响应、获取观察的事件以及发送心跳。如果这个TCP连接中断，客户端将尝试连接到另外的ZooKeeper服务器。客户端第一次连接到ZooKeeper服务时，接受这个连接的 ZooKeeper服务器会为这个客户端建立一个会话。当这个客户端连接到另外的服务器时，这个会话会被新的服务器重新建立。

 启动Zookeeper服务器集群环境后，多个Zookeeper服务器在工作前会选举出一个Leader，在接下来的工作中这个被选举出来的Leader死了，而剩下的Zookeeper服务器会知道这个Leader死掉了，在活着的Zookeeper集群中会继续选出一个Leader，选举出leader的目的是为了可以在分布式的环境中保证数据的一致性。如图所示：



另外，ZooKeeper 支持watch(观察)的概念。客户端可以在每个znode结点上设置一个观察。如果被观察服务端的znode结点有变更，那么watch就会被触发，这个watch所属的客户端将接收到一个通知包被告知结点已经发生变化。若客户端和所连接的ZooKeeper服务器断开连接时，其他客户端也会收到一个通知，也就说一个Zookeeper服务器端可以对于多个客户端，当然也可以多个Zookeeper服务器端可以对于多个客户端，如图所示：

## **Zookeeper客户端与服务端的大致结构**

### 服务端

 Zookeeper还是属于一个C/S的架构的应用服务，Zookeeper的服务器端分为2种运行模式：单台和集群多台的运行模式，通过conf/zoo.cfg中的配置判定你启用的运行模式，以及在群集模式中数据同步和心跳的频率等等。

 Zookeeper集群中的Leader和Follower之间的选举通过Paxos算法来实现的，它是一个基于消息传递的一致性算法，这里讲述了http://zh.wikipedia.org/zh-cn/Paxos算法，传说中Paxos算法是分布式一致性算法中最有效的一种算法

### 客户端

ZooKeeper的Client由三个主要模块组成：

Zookeeper

 Zookeeper是最主要的类，可以写入一个或者多个Zookeeper的服务器地址，例如："127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002" ，当你new Zookeeper( ….)的时候会有两个线程被创建：SendThread和EventThread，会在Server端创建大量的Session。

WatcherManager

在Zookeeper类中还有一个WatcherManager，用来管理Watcher的，Watcher是ZK的一大特色功能，允许多个Client对一个或多个 ZNode进行监控，当ZNode有变化时能够通知到监控这个ZNode的各个Client，管理了ZK Client绑定的所有Watcher。

ClientCnxn

 在Zookeeper类中还包含了对ClientCnxn类的调用，ClientCnxn这个类管理所有对Zookeeper服务器端的网络通讯，服务端和客户端所有交互的数据都要调用这个类，包括给ZK Server发送Request，从ZK Server接受Response，以及从ZK Server接受Watcher Event。

## **服务端运行模式**

### 服务端单机模式

zoo.cfg文件配置参数详解  
    # 这个时间是被用来做服务器之间或客户端与服务器心跳和最低会话超时时间的基数。  
    tickTime=2000  
    # 存储在内存中数据快照的目录。  
    dataDir=d:/zookeeperdata/1  
    # 服务器端开启的监听端口，用来接受客户端访问请求的端口。  
    clientPort=2181

### 服务端集群模式

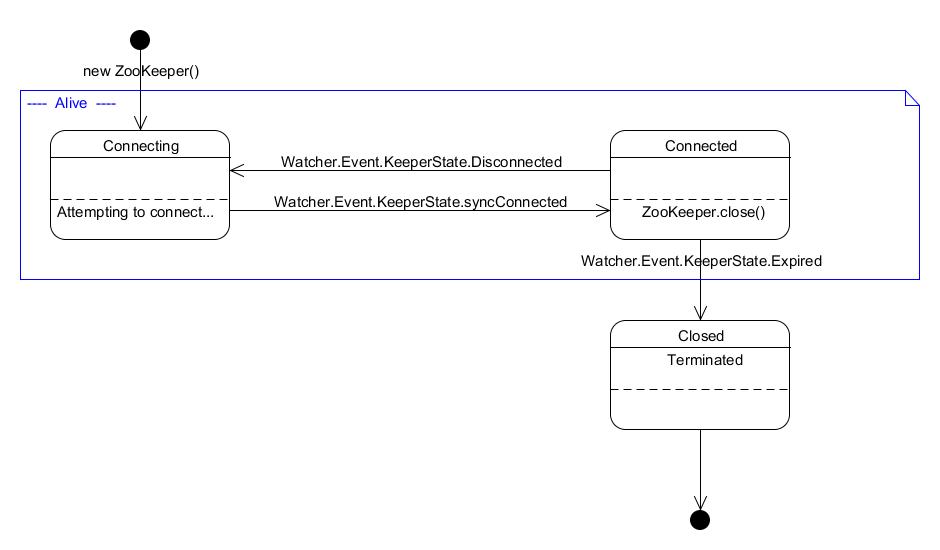
 配置参数详解  
     #Zookeeper服务器集群中连接到Leader与Follower 服务器少次心跳时间间隔数，以及最大通讯的超时时间，总时间为 5(initLimit)\*2000(tickTime)=10 秒。

 initLimit=5   
     #Leader与Follower间请求/应答时间长度,这里总时间长度就是 2(syncLimit)\*2000(tickTime)=4 秒。  
     syncLimit=2

#server是固定配置，1和2表示这个是第几号服务器，2888:3888表示服务器与集群中的 Leader 服务器的通讯端口。  
     server.1=192.168.1.1:2888:3888   
     server.2=192.168.1.2:2888:3888

另外，集群模式下还要在 dataDir 目录下创建一个myid文件，这个文件中写入的内容就是一个数字，这个数字就是和server.x中的x这个数字对应，Zookeeper 启动时会读取这个文件判定自己是谁，myid文件的编码格式是ANSI

ZooKeeper跟Chubby一样用来存放一些相互协作的信息(Coordination)，这些信息比较小一般不会超过1M，在zookeeper中是以一种hierarchical tree的形式来存放，这些具体的Key/Value信息就store在tree node中，这新信息很小一般不回城超过1M，当有事件导致node数据，例如：变更，增加，删除时，Zookeeper就会调用 triggerWatch方法，判断当前的path来是否有对应的监听者(watcher),如果有watcher，会触发其process方法，执行process方法中的业务逻辑，如图所示：



## 数据模型

Zookeeper 会维护一个具有层次关系的数据结构，它非常类似于一个标准的文件系统，如图 1 所示：



Zookeeper 这种数据结构有如下这些特点：

* 每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 Server1 这个 znode 的标识为 /NameService/Server1
* znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录
* znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据
* znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了
* znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2
* znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的，后面在典型的应用场景中会有实例介绍

## 如何使用

Zookeeper 作为一个分布式的服务框架，主要用来解决分布式集群中应用系统的一致性问题，它能提供基于类似于文件系统的目录节点树方式的数据存储，但是 Zookeeper 并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理，后面将会详细介绍 Zookeeper 能够解决的一些典型问题，这里先介绍一下，Zookeeper 的操作接口和简单使用示例

### 常用接口列表

客户端要连接 Zookeeper 服务器可以通过创建 org.apache.zookeeper. ZooKeeper 的一个实例对象，然后调用这个类提供的接口来和服务器交互。

前面说了 ZooKeeper 主要是用来维护和监控一个目录节点树中存储的数据的状态，所有我们能够操作 ZooKeeper 的也和操作目录节点树大体一样，如创建一个目录节点，给某个目录节点设置数据，获取某个目录节点的所有子目录节点，给某个目录节点设置权限和监控这个目录节点的状态变化。

这些接口如下表所示：

##### org.apache.zookeeper. ZooKeeper 方法列表

| **方法名** | **方法功能描述** |
| --- | --- |
| [String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true)[create](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#create%28java.lang.String,%20byte[],%20java.util.List,%20org.apache.zookeeper.CreateMode%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl,[CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) createMode) | 创建一个给定的目录节点 path, 并给它设置数据，[CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) 标识有四种形式的目录节点，分别是 PERSISTENT：持久化目录节点，这个目录节点存储的数据不会丢失；PERSISTENT\_SEQUENTIAL：顺序自动编号的目录节点，这种目录节点会根据当前已近存在的节点数自动加 1，然后返回给客户端已经成功创建的目录节点名；EPHEMERAL：临时目录节点，一旦创建这个节点的客户端与服务器端口也就是 session 超时，这种节点会被自动删除；EPHEMERAL\_SEQUENTIAL：临时自动编号节点 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html)[exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 判断某个 path 是否存在，并设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper 实例时指定的 watcher，[exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "exists%28java.lang.String,%20boolean%29)方法还有一个重载方法，可以指定特定的 watcher |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html)[exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20org.apache.zookeeper.Watcher%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[Watcher](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/Watcher.html) watcher) | 重载方法，这里给某个目录节点设置特定的 watcher，Watcher 在 ZooKeeper 是一个核心功能，Watcher 可以监控目录节点的数据变化以及子目录的变化，一旦这些状态发生变化，服务器就会通知所有设置在这个目录节点上的 Watcher，从而每个客户端都很快知道它所关注的目录节点的状态发生变化，而做出相应的反应 |
| void [delete](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#delete%28java.lang.String,%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, int version) | 删除 path 对应的目录节点，version 为 -1 可以匹配任何版本，也就删除了这个目录节点所有数据 |
| [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true)>[getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 获取指定 path 下的所有子目录节点，同样 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getChildren%28java.lang.String,%20boolean%29)方法也有一个重载方法可以设置特定的 watcher 监控子节点的状态 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html)[setData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#setData%28java.lang.String,%20byte[],%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, int version) | 给 path 设置数据，可以指定这个数据的版本号，如果 version 为 -1 怎可以匹配任何版本 |
| byte[] [getData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getData%28java.lang.String,%20boolean,%20org.apache.zookeeper.data.Stat%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch, [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat) | 获取这个 path 对应的目录节点存储的数据，数据的版本等信息可以通过 stat 来指定，同时还可以设置是否监控这个目录节点数据的状态 |
| void[addAuthInfo](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#addAuthInfo%28java.lang.String,%20byte[]%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) scheme, byte[] auth) | 客户端将自己的授权信息提交给服务器，服务器将根据这个授权信息验证客户端的访问权限。 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html)[setACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#setACL%28java.lang.String,%20java.util.List,%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl, int version) | 给某个目录节点重新设置访问权限，需要注意的是 Zookeeper 中的目录节点权限不具有传递性，父目录节点的权限不能传递给子目录节点。目录节点 ACL 由两部分组成：perms 和 id。 Perms 有 ALL、READ、WRITE、CREATE、DELETE、ADMIN 几种  而 id 标识了访问目录节点的身份列表，默认情况下有以下两种： ANYONE\_ID\_UNSAFE = new Id("world", "anyone") 和 AUTH\_IDS = new Id("auth", "") 分别表示任何人都可以访问和创建者拥有访问权限。 |
| [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)>[getACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getACL%28java.lang.String,%20org.apache.zookeeper.data.Stat%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat) | 获取某个目录节点的访问权限列表 |

### 基本操作

下面给出基本的操作 ZooKeeper 的示例代码，这样你就能对 ZooKeeper 有直观的认识了。下面的清单包括了创建与 ZooKeeper 服务器的连接以及最基本的数据操作：

## ZooKeeper 典型的应用场景

Zookeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式，关于 Zookeeper 的详细架构等内部细节可以阅读 Zookeeper 的源码

下面详细介绍这些典型的应用场景，也就是 Zookeeper 到底能帮我们解决那些问题？下面将给出答案。

### 统一命名服务（Name Service）

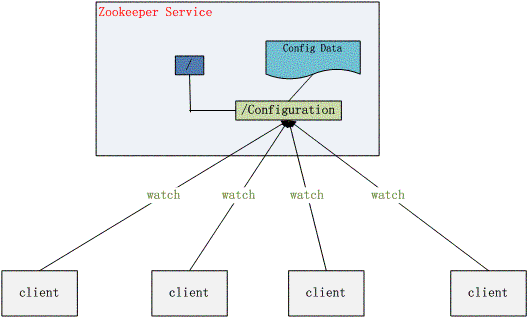
分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

### 配置管理（Configuration Management）

配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。

像这样的配置信息完全可以交给 Zookeeper 来管理，将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。



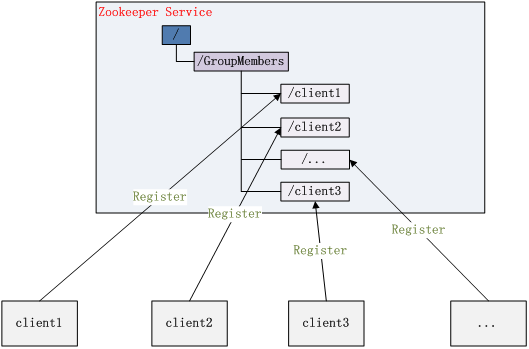
### 集群管理Group Membership

Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。

Zookeeper 不仅能够帮你维护当前的集群中机器的服务状态，而且能够帮你选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能 Leader Election。

它们的实现方式都是在 Zookeeper 上创建一个 EPHEMERAL 类型的目录节点，然后每个 Server 在它们创建目录节点的父目录节点上调用[getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) 方法并设置 watch 为 true，由于是 EPHEMERAL 目录节点，当创建它的 Server 死去，这个目录节点也随之被删除，所以 Children 将会变化，这时 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getChildren%28java.lang.String,%20boolean%29)上的 Watch 将会被调用，所以其它 Server 就知道已经有某台 Server 死去了。新增 Server 也是同样的原理。

Zookeeper 如何实现 Leader Election，也就是选出一个 Master Server。和前面的一样每台 Server 创建一个 EPHEMERAL 目录节点，不同的是它还是一个 SEQUENTIAL 目录节点，所以它是个 EPHEMERAL\_SEQUENTIAL 目录节点。之所以它是 EPHEMERAL\_SEQUENTIAL 目录节点，是因为我们可以给每台 Server 编号，我们可以选择当前是最小编号的 Server 为 Master，假如这个最小编号的 Server 死去，由于是 EPHEMERAL 节点，死去的 Server 对应的节点也被删除，所以当前的节点列表中又出现一个最小编号的节点，我们就选择这个节点为当前 Master。这样就实现了动态选择 Master，避免了传统意义上单 Master 容易出现单点故障的问题。



#### Leader Election 关键代码

void findLeader() throws InterruptedException {

byte[] leader = null;

try {

leader = zk.getData(root + "/leader", true, null);

} catch (Exception e) {

logger.error(e);

}

if (leader != null) {

following();

} else {

String newLeader = null;

try {

byte[] localhost = InetAddress.getLocalHost().getAddress();

newLeader = zk.create(root + "/leader", localhost,

ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);

} catch (Exception e) {

logger.error(e);

}

if (newLeader != null) {

leading();

} else {

mutex.wait();

}

}

}

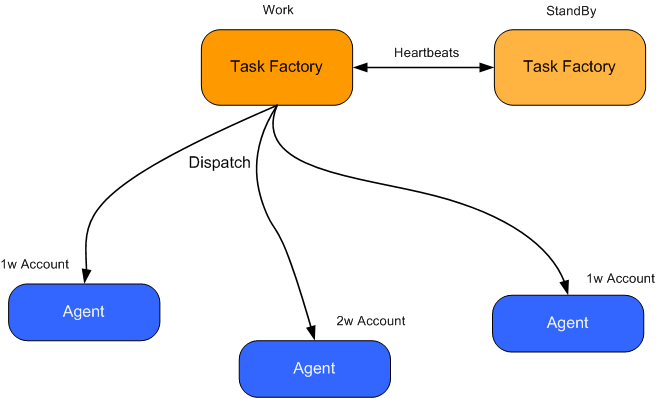
## **应用实例**

ZooKeeper有了上述的这些用途，让我们设想一下，在一个分布式系统中有这这样的一个应用：

 2个任务工厂(Task Factory)一主一从，如果从的发现主的死了以后，从的就开始工作，他的工作就是向下面很多台代理(Agent)发送指令，让每台代理(Agent)获得不同的账户进行分布式并行计算，而每台代理(Agent)中将分配很多帐号，如果其中一台代理(Agent)死掉了，那么这台死掉的代理上的账户就不会继续工作了。  
上述，出现了3个最主要的问题：

1.Task Factory 主/从一致性的问题  
    2.Task Factory 主/从心跳如何用简单+稳定 或者2者折中的方式实现。  
    3.一台代理(Agent)死掉了以后，一部分的账户就无法继续工作，需要通知所有在线的代理(Agent)重新分配一次帐号。

怕文字阐述的不够清楚，画了系统中的Task Factory和Agent的大概系统关系，如图所示：



### 解决思路

1. 任务工厂Task Factory都连接到ZooKeeper上，创建节点，设置对这个节点进行监控，监控方法例如：  
    event= new WatchedEvent(EventType.NodeDeleted, KeeperState.SyncConnected, "/TaskFactory");  
   这个方法的意思就是只要Task Factory与zookeeper断开连接后，这个节点就会被自动删除

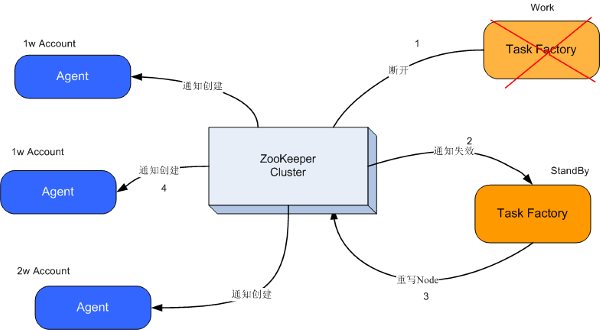
2.原来主的任务工厂断开了TCP连接，这个被创建的/TaskFactory节点就不存在了，而且另外一个连接在上面的Task Factory可以立刻收到这个事件(Event)，知道这个节点不存在了，也就是说主TaskFactory死了

3.接下来另外一个活着的TaskFactory会再次创建/TaskFactory节点，并且写入自己的ip到znode里面，作为新的标记。

4.此时Agents也会知道主的TaskFactory不工作了，为了防止系统中大量的抛出异常，他们将会先把自己手上的事情做完，然后挂起，等待收到Zookeeper上重新创建一个/TaskFactory节点，收到 EventType.NodeCreated 类型的事件将会继续工作

5.原来从的TaskFactory 将自己变成一个主TaskFactory，当系统管理员启动原来死掉的主的TaskFactory，世界又恢复平静了。

6.如果一台代理死掉，其他代理他们将会先把自己手上的事情做完，然后挂起，向TaskFactory发送请求，TaskFactory会重新分配(sharding)帐户到每个Agent上了，继续工作

上述内容，大致如图所示

## Getting Started

Start by installing ZooKeeper on a single machine or a very small cluster.

ZooKeeper is a high-performance coordination service for distributed applications. It exposes common services - such as naming, configuration management, synchronization, and group services - in a simple interface so you don't have to write them from scratch. You can use it off-the-shelf to implement consensus, group management, leader election, and presence protocols. And you can build on it for your own, specific needs.

## Deployment

This section contains information about deploying Zookeeper and covers these topics:

* [System Requirements](http://zookeeper.apache.org/doc/trunk/zookeeperAdmin.html#sc_systemReq)
* [Clustered (Multi-Server) Setup](http://zookeeper.apache.org/doc/trunk/zookeeperAdmin.html#sc_zkMulitServerSetup)
* [Single Server and Developer Setup](http://zookeeper.apache.org/doc/trunk/zookeeperAdmin.html#sc_singleAndDevSetup)

The first two sections assume you are interested in installing ZooKeeper in a production environment such as a datacenter. The final section covers situations in which you are setting up ZooKeeper on a limited basis - for evaluation, testing, or development - but not in a production environment.

### System Requirements

#### Supported Platforms

* GNU/Linux is supported as a development and production platform for both server and client.
* Sun Solaris is supported as a development and production platform for both server and client.
* FreeBSD is supported as a development and production platform for both server and client.
* Win32 is supported as a development platform only for both server and client.
* MacOSX is supported as a development platform only for both server and client.

#### Required Software

ZooKeeper runs in Java, release 1.6 or greater (JDK 6 or greater, FreeBSD support requires openjdk7). It runs as an ensemble of ZooKeeper servers. Three ZooKeeper servers is the minimum recommended size for an ensemble, and we also recommend that they run on separate machines. At Yahoo!, ZooKeeper is usually deployed on dedicated RHEL boxes, with dual-core processors, 2GB of RAM, and 80GB IDE hard drives.