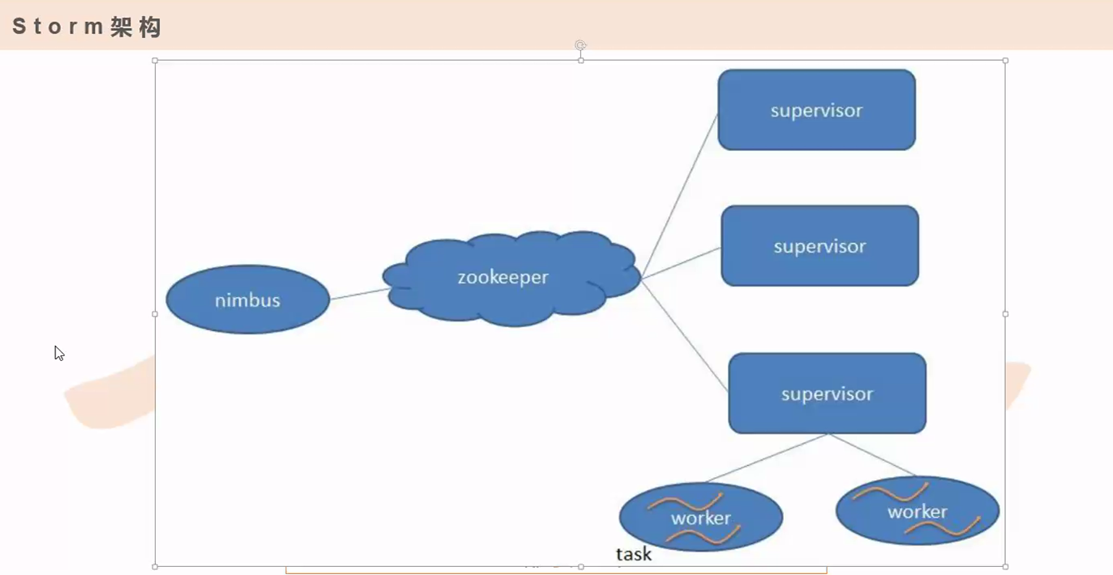
图1：



上次课我们讲了Storm基础的，接下来我们继续我们的内容，接下来的内容我们开始讲Storm架构。

图2：



那这个就是Storm的一个架构，那在这个架构里面大家应该能看得清是吧？在这个架构里面也有很多的机器，节点，集群是吧？然后在这个集群里面有很多的角色，一般在这个集群里面有主有从是吧？但是这个Storm这个架构我们之前学的这些架构不太一样是吧？我们之前学的HDFS还是MapReduce里面，中间就是主和从。那在这个Storm这个架构里面做了一些变化，你会发现中间加了一个zookeeper，这zookeeper它左手牵了一个nimbus，右手牵了一堆的从节点是吧？，那这样感觉什么意思呢？相当于是这个主和从之间不直接建立关系了，完全是通过一个像zookeeper这么一个中介，然后互相响应一些信息，那其实zookeeper就是这样一个角色，它就是来协调整个集群的一些有序运转的这么一个角色，它是一个分布式锁的服务，那也就是说整个集群，你这个集群是否能够健康是否能够运转zookeeper这个功能是很大的，然后因为我们之前开始讲完了MapReduce讲完了HDFS，现在又讲Storm，如果再往下去讲的话，那接下来的内容都是和我们的架构相关的，那只要是讲架构部分，那zookeeper这个地方一定要明白，得有这方面的知识铺垫，否则的话你没有zookeeper的一些知识铺垫的话你再去了解这个架构，会有些困难，所以今天我们改下策略。

我们先讲zookeeper，所以今天的课程是讲zookeeper，然后讲完zookeeper之后，我们回过来再继续Storm这么一个内容。这是为了能让大家更好的理解。

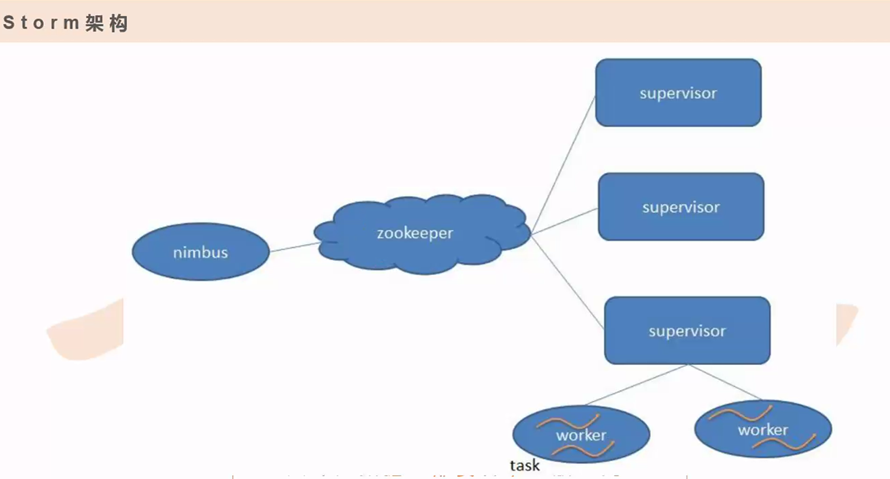
图2：



那这次我们主要围绕两个点，一个是zookeeper概述和zookeeper的应用场景，然后这个zookeeper是一个分布式的锁服务，它对于一个集群的稳定是起到一个非常非常关键的作用，因为它可以协调不同节点之间的工作，就是每个节点之间能够协同工作。

然后接下来我们通过概述和应用场景这两部分我们能够深入了解一下zookeeper的内在东西，但是你得带着个问题，什么问题呢？就是为什么zookeeper是可以有效的协调不同机器之间的工作，就像(如下图3所示)，为什么zookeeper是可以在整个Storm集群中起到了一个协调的这么一个角色。

图3：



大家带着这个问题去学zookeeper，好了那我们看一下（如下图4所示）

图4：



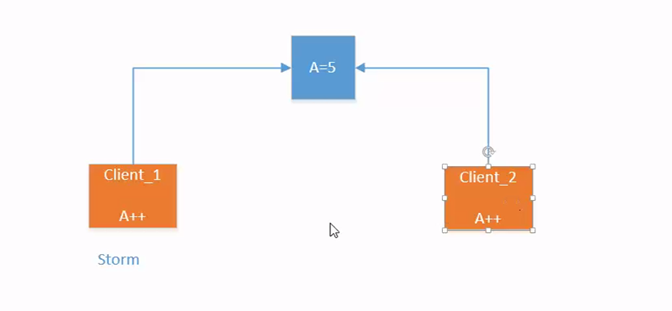
这块比较简单，我们快速的过一下。

这个单机系统就是什么是分布式锁服务呢？就是假如说我们在传统的开发过程中，那我们通常用单机去开发对吧？那我们用单机开发的时候涉及到锁是在什么地方的？一般都是在多线程的开发，多线程开发然后就会涉及到一些锁，搞一些什么内存锁啊，什么互斥锁，然后读写锁等等的这些锁。

那为什么可以通过这样的一个锁服务能够让你在单机的环节下不同的线程之间能够有效的运转呢？是因为这个锁它可以能够做一个全局唯一的这么一个标准，然后所有的线程，因为每条线程你可以暂时认为它是相互独立的，如果每个线程都各自为证，谁也不听谁的，然后各自去运行自己的任务运行自己的代码段，这样的话代码段跟代码段之间它是有重合的话就是有这种数据交互的直接关系或者有公共区域读写的情况下，那这样很容易会造成一个混乱的状态。

那有一个例子，这个例子好像我不知道之前有没有讲过，这个例子就是这样的，比如说我这有一台服务器，在这服务器里面就多线程，比如说这两个黄色是一个线程（如下图5所示）

图5：



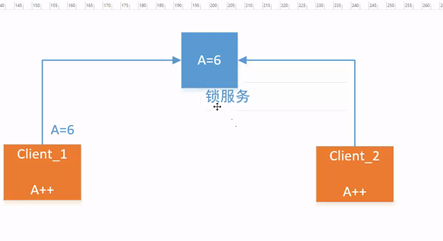
然后这个蓝色区域是一个公共数据的一个内存的一个变量，A=5，然后这两个黄色的客户端都想对这个蓝色里面的数据进行请求，那请求就包括读和写对不对？那它们的请求包括两个部分，我先想把这个数据读进来，读完之后在我的线程的内部对你这个蓝色里面的变量做一个A++，然后把这个A++再写回去，那么客户端2一样的目的。

那如果是我们模拟一下，假设说Client1先读到了这个蓝色内存区域里面的变量，然后Client1里面不是有A++吗？那么结果就是A=6，然后把这个最终得到的结果写回去给蓝色内存区域中，那么这个时候蓝色内存区域里面就变成A=6了对吧？，那这个时候Client2也开始想做一个A++是吧？它把这个蓝色内存区域的A=6读下来然后自己再做一个A++，最终结果变成了A=7是吧？然后把这个最终的结果A=7又写回给蓝色内存区域里面，然后蓝色内存区域就变成了A=7，这感觉是一点毛病都没有对不对？但是如果要是说中间顺序发生混乱的话会有问题，比如说蓝色区域A=5，客户端1先把这个A=5从这个蓝色内存区域把这个变量读进来，然后做一个A++得到A=6，但是这个时候A=6这个变量还没有来得及往这个蓝色内存区域上返回数据，这个时候客户端2去请求蓝色内存区域的数据，把A=5读进来到客户端2中，这个时候客户端2也对他做了一次加法操作，然后客户端1在它读完之后就往蓝色内存区域里面去写，这个时候蓝色区域就由A=5变成了A=6，而客户端2因为他也做完了加法，再返回数据给蓝色区域部分，这个时候在蓝色区域部分A=6不变，因为整个全局里面A++一共是加了几次？加了两次是吧？

但是按理来说这个时候蓝色区域部分已经是变成了A=7，其实A还是=6，说明这个数据是错的，那为什么是错误的呢？就是因为这个客户端1和客户端2之间没有一个顺序关系，就是没有一个对公共区域的一个锁，什么叫做锁呢？

你可以认为就是一个保安，比如说这里面只能有一个人进去，好了这个人进去以后，然后外面有一个保安就不能让外人进了，如果是第二个人想进的时候，保安说这个屋里面有人，等着屋里这个人办完事之后你才能进对不对？如果这个人客户端1这个A++做好加法之后，然后出来了，这个时候客户端2才能往里走对不对？所以相当于这个保安就相当于是一个锁服务，这个大家理解了吗？（如下图6所示）

图6：



这个例子应该比较简单的例子，大家应该好理解把？

好了你会发现刚才这个锁服务其实在你的单机系统里面去实现的时候，它的成本其实很低的，为什么呢？因为我刚才举得这个例子是一个多线程的情况，多线程的情况是在一个进程内部对一个公共资源进行竞争对吧？那在一个竞争空间它的内存地址是一致的，所以它对公共的内存的区域它的访问，就是普通的线程对同样的一个进程空间它都是可以操作的对吧？

另外一个单机出问题概率低，这个就不解释了好吧，那这是我们传统的开发模式。

那现在因为要处理大数据，通常是通过一个集群的方式，通过多台机器搭建的一套分布式的一个服务然后去做一个事情，类似这种事情可以在单机系统里面可以做，但是现在由于数据规模大了，比如说你在单机的系统里面做A++，你想在分布式环境里面你也想做一个A++，虽然都是A++这么一个功能，于是由于你的环境你的客观条件的一个情况，那在分布式的环境下你想做A++就很不容易了对不对。

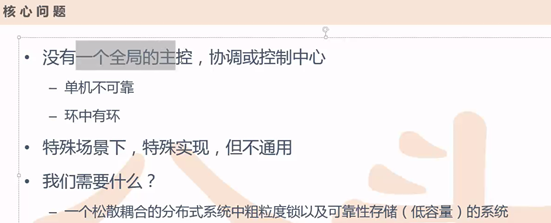
那目的是什么呢？目的就是分布式系统也想拥有一个类似这么一个锁服务，在单机系统里面你搞一个锁是很容易的，在内存里面就写一个变量或者是写几个代码搞定了,你在分布式的场景下比如说这几个机器都是一个独立的机器节点，这机器与机器之间完全是独立的然后隔离的对吧？那彼此之间别说内存区域了，你连彼此之间的内存条你都看不见，那就别说内存区域能不能访问就是这样的一个情况。

所以相当于我能不能把这种每一个都独立的机器节点然后都能够像单机一样能够有效把它维护起来，我想分布式里面也需要这么一个锁服务，但是你想搞这个锁服务就不容易了，你需克服很多的问题，有哪些问题呢？有一致性的问题，什么是一致性的问题？ 这个之前我们讲过强一致性和弱一致性是吧？我们在讲那个HDFS里面讲过是吧？然后还有容灾容错，比如说你有一台机器挂掉了怎办？你得解决。

然后执行顺序，刚才我上面讲的这个例子其实就是一个执行顺序的一个没有协调好导致的一个问题。

然后事务性的问题也是跟一致性和执行顺序都有关系，所以为了防止分布式的系统中多个进程之间互相干扰就我们需要有一种协调的技术对这些进程进行调度，然后这个分布式的协调技术的核心就是来实现一个分布式的一个锁服务，把这个锁服务变成一个分布式锁。

那如果要是没有一个分布式锁服务的话，其实就有一些问题了，这个问题其实很严重（如下图7所示）

图7：  


就是没有一个全局的主控，然后没有一个全局的协调控制中心导致了每一个机器之间互相就没有那个顺序然后去提供服务。

然后如果有这个分布式锁服务，那我们想要一个怎样的一个锁服务，比如我们想要一个叫松散耦合的分布式系统中粗粒度锁以及可靠性存储低容量的系统，感觉这句话很长而且信息量很大对吧？并且有点绕口。其实呢我们需要就是一个分布式锁服务，但这个分布式锁服务需要具备一下特点，那有哪些特点呢？都在（松散耦合的分布式系统中粗粒度锁以及可靠性存储低容量的系统）这句话里面包含了。

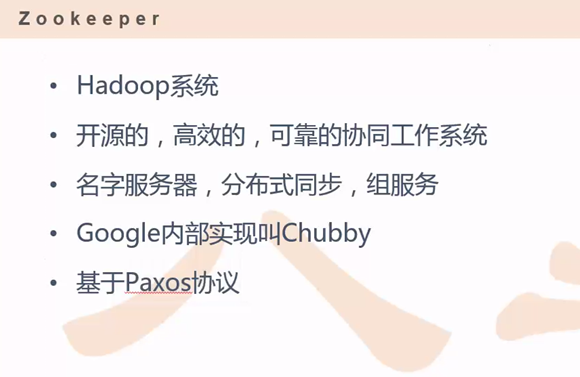
首先我们一个一个字去拆分去理解，这个锁服务第一个特点就是说松散耦合，什么是松散耦合呢？就是我对这个硬件的依赖性不强，你随便给我搞几个配置不用特别高的机器就可以了，就不一定非要你搞一个什么特别强，内存又特别大，然后硬盘又特别大，CPU又好几个核，那这种机器的配置就是特别特别高级的机器，完全没必要，不需要特别高的机器，那这个相当于是对这个硬件不是特别强的依赖，就跟你的松散耦合硬件没有太多的保定关系可以这么理解。

分布式系统这个更容易理解，就是这个分布式锁服务是在你的分布式里面去服务的，然后粗粒度锁那就是细粒度的锁对不对？那刚才在这个进程空间里面如果我们创建这个锁，那这个锁属于那种线程级的锁，那这种锁是力度很细的，那但是在那个zookeeper里面就需要一个锁服务就可以了，那这个力度是可以放粗一点，不一定需要像进程空间那么一个搞一个锁来控制，可以搞那种只要能保证我每台机器之间能顺利的协调工作就可以了。

然后可靠性存储是什么意思呢？就是说我这个分布式锁服务它不仅能够帮整个集群提供一个锁的这么一个服务，能够协调节点之间有效运转，并且我还要求要有一定的存储功能，我可以能不能把它当作一个数据库类似这么一个数据库，能不能把它当作一个数据库帮我在存储一些数据这么一个系统，所以这个分布式锁服务系统就有这么几个特点，就是跟你的硬件耦合没那么高，然后并且是服务分布式系统中，然后粗粒度锁呢这个锁起码能够给我协调不通机器之间的一个有序就可以了，不需要特别的精细到然后内存的某一个变量上，这个是不需要的，只需要告诉我一个信号,说当前这个机器之间能够有效能够协调工作就可以了。然后可靠性存储就是我可以在这个系统上可以存一些数据。

然后zookeeper就是这样的一个系统，那zookeeper其实它属于hadoop生态里面的一个重要成员，然后它是开源的，高效的，可靠的（如下图8所示）

图8：



然后名字服务器，什么是名字服务器呢？就是zookeeper这个系统一会大家学了之后就会发现这个系统可以给大家取名字,比如说你这个有几台机器，然后你也不确定给这几台机器取什么名字好，但是你有个要求，什么要求呢？就是你要求每一台机器名字不重复明白了把？

你每台机器的这个名字是不重复的，就好像是hostName，你要是有两个机器hostName一样的话是不希望看到的，所以要求你每一台机器你给配的这个名字是不要重复的，所以这个取名字的话能不能有一个中介来帮我取这个名字，那zookeeper是可以承担这么一个功能的，是可以帮我给一批机器统一取名的明白这意思把？具体怎么取名字呢？这个我们后面会说。

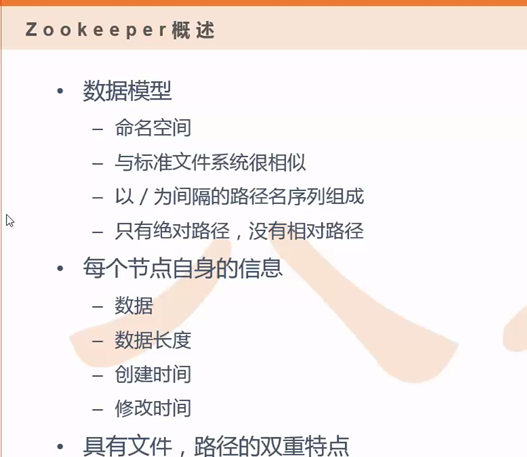
然后分布式同步，然后组服务这都相当于是比如说我用5台机器搭建了一个zookeeper集群，然后你去访问任何一个zookeeper节点你得到的信息都是一致的，然后在Google内部叫Chubby，但是这个东西是没有开源出来的，是谷歌自己用的，后来雅虎就模仿了这么一个服务，然后搞出了一个zookeeper，并且zookeeper作为一种开源程序。

然后通过Paxos协议能够保证你在这个集群里面你访问任何一台机器你得到的结果都是一样的明白把？

然后对这个zookeeper这一块它的数据模型就比较容易理解，那数据模型类似于我们在linux环境下的一个文件系统的目录，比如这个目录里面有文件然后有目录，然后目录里面又套着很多目录和文件对不对？相当于是每一个目录与目录之间它上下是有一个层级关系或者是有一个父子之间的关系，比如说你去访问一个linux目录，这个目录里面有很多的层级，相当于是上面的和这个下面的是有这种层级关系的，并且同一层之间你可以允许有很多的节点。

然后你想访问某一个节点的话你必须以这种反斜杠做间隔去访问（如下图9所示）

图9：

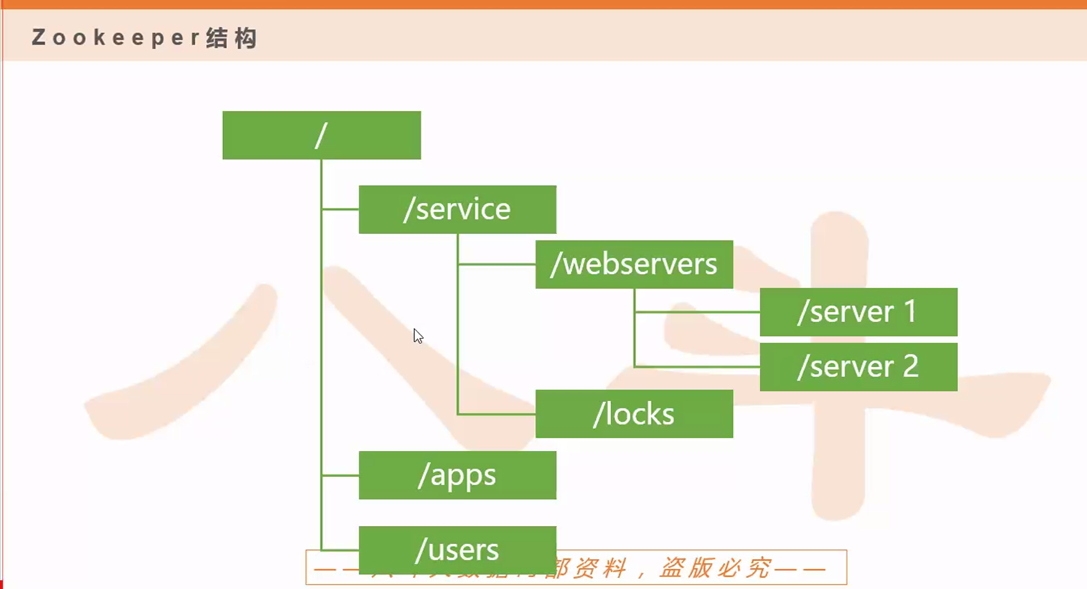


这个跟我们在linux环境下去访问某一个文件的目录是一样的，比如说我们去访问linux环境下去访问某一个文件，就相当于一串路径，然后以反斜杠为间隔，然后通常我们在linux的环境下你去访问一个目录的话你可以用cd命令这个大家没问题，比如cd 到某一个目录下。

然后在linux环境下你想去你的父级目录你就是cd ..对不对？你再去它的上一级再上一级目录就是cd ../cd ...，但是在这个zookeeper你想访问某一个节点的话是不允许这样的，你只能通过一个绝对路径去访问某一个节点，并且在zookeeper这么一个节点里面，它可以存储一些数据，并且他自然而然就给你配一些属性，这个属性就关于你数据的长度，创建时间，修改时间等等，然后你这个节点具有文件属性又有路径的一个特点。

具有文件的特点是什么意思呢？是它可以存数据，路径的特点就是你可以通过一个绝对路径能够访问到指定的节点，那如果通过一个很形象的图来看一下（如下图10所示）

图10：



这个就很像我们linux非常普遍的一个文件目录数的这么一个结构对不对？

比如图中/server1你把这个可以认为是个目录也好是个普通文件都可以，那你想访问这个/server1文件，那你想访问这个文件的话怎么访问呢？你必须前面带一个斜杠然后进入/server然后再进入/webserver然后找到/server1，必须要通过这种绝对路径从你的根目录开始逐条的去访问。跟那个HDFS有点类似，所以zookeeper去访问的路径的时候跟文件系统很相似，你对zookeeper操作就像访问一个在我们linux环境下或者是在HDFS上操作一个文件类似，其实zookeeper里面它是没有文件目录这么一个说法的，但这个每一个根下面就像这些绿色的四方块都叫做节点，在zookeeper里面所有的文件都是节点，只不是节点下面有多个节点。

那说道节点的话，这这节点就有一定的属性了（如下图11所示）

图11：

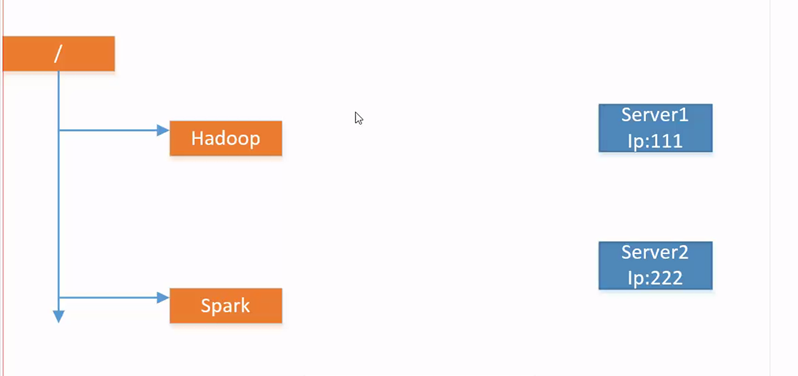


每一个节点它都是有属性的，那有几种节点形式呢？，其实是有四种，为什么有四个呢？当然上面图11中才三个节点形式，首先看一下他的类型。

第一个节点Persistent Nodes永久性节点，就是在linux环境下你去建一个文件然后里面写一些字符串一些信息等等，你再保存退出，只要是你这个系统不崩溃不被格式化，只要是你的电脑还可以用你创建这个文件就永远不会丢失明白了把？你创建的文件就放在那里，只要你不去删它。永久性节点就是这样的，如果你不去删除它，那么你创建的这个节点就会一直存在着。

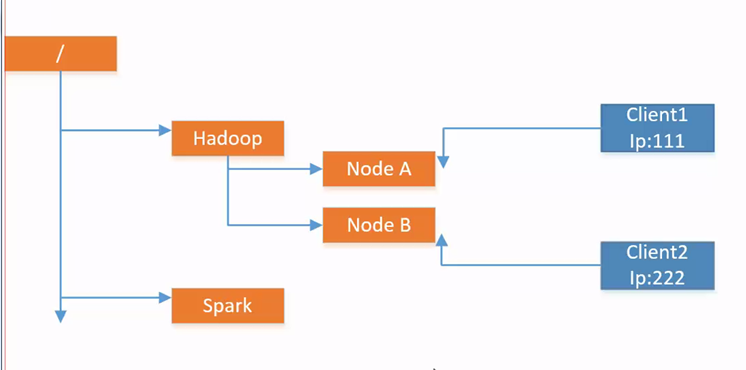
还有一个临时性节点Ephemeral Nodes，什么是临时性节点呢？就是你创建的这个节点，可能还没等到你删，它就已经可能消失了，那这种节点有什么用呢？这个类型的节点是非常非常有用的，举个例子：（如下图12所示）

图12



一开始没有NodeA和NodeB这两个节点，我通过这个Server1开始创建节点，于是NodeA这个节点是由Server1来创建的（如下图13所示）还有NodeB是由Server2来创建的。

图13：



Server1和Server2都有两台独立的ip地址，Server1和Server2它们一旦创建了一个NodeA和NodeB，这个NodeA一直发送给自己的心跳给创建自己的那台服务器，包括NodeB也是如此。那么假设Server1挂掉了，那么这台机器消失，那么NodeA这个节点也跟着消失，如果Server2消失，那么NodeB也消失明白了把？

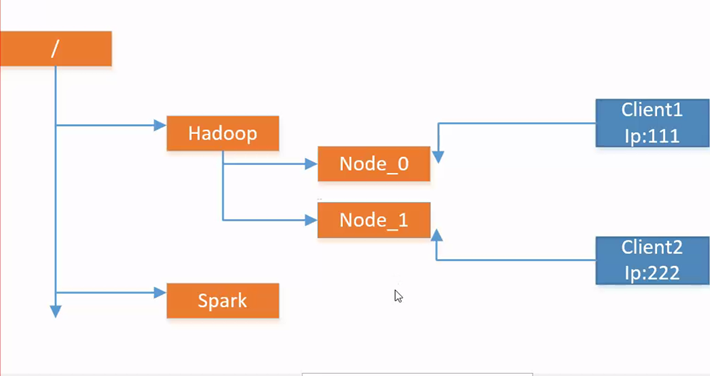
就是说这个NodeA和NodeB这两个节点的消失并不是我想让它消失的，而是它的外部环境发生了变化导致了它们两个节点被删除，并不是我要人为的要求删除它明白了把？所以这类的节点就是一个临时节点。然后你创建这个临时节点client，这个client其实就是一个Client1和Client2，这两个Client机器就会与它保持一定的连接，在这连接的过程中一直有效，一旦这个机器有问题或者是连接失败，Zookeeper会自动把这个节点删除明白了把？

这里面注意一个问题，假设说我Client2消失了，这个NodeB消失了,那这个时候我Node2这个节点我又恢复了，那问一下大家这个NodeB恢复吗？答案是：不恢复的。那么如果你想让他恢复咋办，没办法你只能Client2再自己再去创建一个，我只能监控你的出问题，但是你又活过来了，我是监控不到的，就是这个意思。

还有一个很有意思的节点就是顺序节点，什么是顺序节点呢？这个顺序节点就刚才我们讲到那个名字服务器就有关系了，取名字。那我怎么保证这个名字之间不重复？比如说这个像以前那个年代，一家都有很多的小孩，为了让这个孩子之间这个名字不重复并且又好记，那就是老大老二老三等等，相当于这个数字就一直不断的累加对不对？那这样每个人的外号或者一个名字都是不可能重复的对不对？那你这个后面的数字是不断的累加而且又是唯一的。

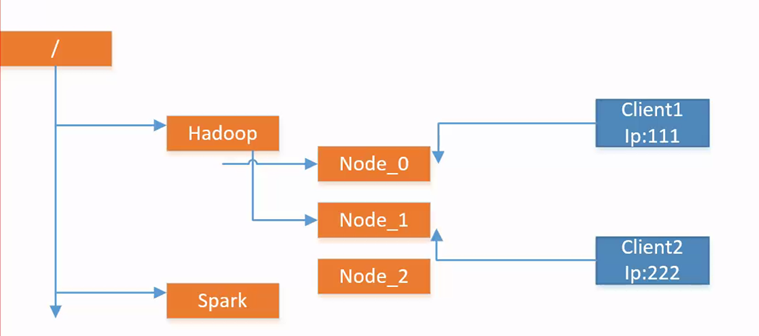
所以你可以认为这个顺序节点也是这么一个机制把，也就是说当你这个Client1创建了一个NodeA，比如说NodeA就是一个Node，然后系统会自动帮你在后面追加一个0，如果你这个另外的一个机器，你还在这个hadoop目录下面再去创建一个节点的话那你就直接追一个1（如下图14所示）

图14：



如果你再往后面添加一个节点的话，那就把这个节点后面改成了2（如下图所示15）

图15：



就不断不断的去累加，这样的话就保证了你每一个节点的名字不重复，那你的名字不重复跟我这个取名字有什么关系是不是？你只是把你的节点名字取好了，对我服务器有什么关系？

假如说我想给这个服务器Client1取名，这个Client1搞了半天我给它取了个名字，但是这个时候你是可以通过Client1去写一些代码，你可以把这个名字读回来，然后你就知道了我已经在Node\_0这个地方领了一个名字过来然后我自己把这个名字配上去明白这个过程把？？相当于这个Node\_0地方已经给我起好了名字，但是你得过来取是不是？就好像以前照相片似得是吧？你当场不能取到，你只能隔两天才能来照相馆，把你的相片取走，就类似这么一个操作，这个就是顺序节点。

然后刚才说过了在整个的Zookeeper节点类型里面一共有四种方式，而为什么只介绍三种？

这里面有一个大家一定要注意的点，这个其实很重要，第一个就是这个永久性节点(Persistent Nodes)这是一种节点，第二种节点临时节点（Ephemeral Nodes）是一种节点，顺序节点（Sequence Nodes）不允许单独存在明白了把？，顺序节点只能由这个永久性节点(PersistentNodes) 或者是临时节点（Ephemeral Nodes）绑定，这个时候就有四个了对不对？所以为什么在Zookeeper里面有四种类型呢？那就是因为这个原因。

好了那这个节点的事就已经讲完了，然后我们后面再围绕这个节点去做进一步的分析。

那么监控机制（watch）是什么意思呢？（如下图16所示）

图16：



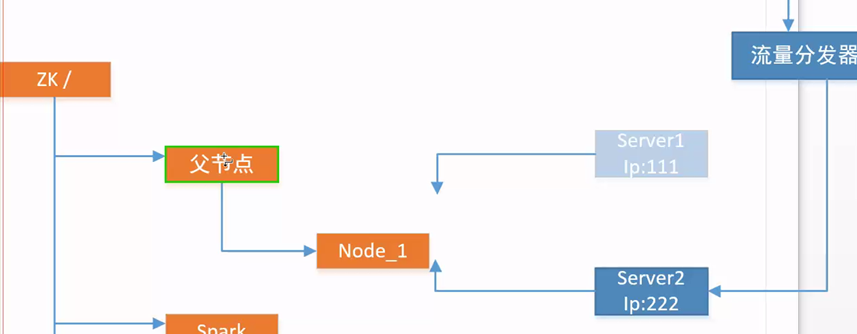
刚才我们举得这个例子就很形象了，假设说这个Zookeeper这个是有监控功能的，它监控功能是监控什么呢？是可以监控某一台机器是否生效，这个功能就很有用了，假设说比如在这个集群里面有主节点和从节点对不对？那主挂了之后，如果你这个环境里面仍然还有一个备份的主的话，这个时候你就可以把备份的主启动起来，那问题是前提是你能监控到主挂了这么一个事件是不是？

那这个由谁来监控呢？由Zookeeper来监控，那Zookeeper怎么监控？假设说就想监控你这个机器健康还是不健康，好了因为你这个机器可以创建临时节点，那我就通过这个临时节点一直监控着你这个机器，它为什么能监控着你的机器是因为之前有心跳，刚才说过了。

那这个时候你这个机器出现了异常，好了我这个节点是不是就是消失了？那我这个节点消失了之后，那我这个Zookeeper是可以能够感知到我的节点消失了对不对？它能感知到这么一个事件发生，那谁能感知这个事件发生呢？是由刚才这个消失的这个节点的父节点它能够感知到。

再举个例子：（如下图17所示）

图17：



比如说这Server1和Server2这两台机器是新浪的两台机器，然后你作为你普通的一个网名去访问这个新浪页面，你可以访问Server1也可以访问Server2机器都可以，这两个机器提供的服务都一样，如果说Server1这台机器出现了异常，那么Node1这个节点就出现了问题消失了就挂了，但是对于流量分发器来说它是不知道的，那如果你有网名继续发流量的话，那比如说按照这百分之50去流量做分发，仍然有百分之50流量往这个已经出现问题的这个机器上仍然去发请求大流量对不对？因为这个机器不能够提供服务了，所以你发给他这个请求流量都有问题，就是这个网名得不到结果，那也就是说直接影响到你的线上一半服务了对不对？

那想怎么样解决这个问题呢？很简单，我流量分发器我就不忘你这分发了，我把所有流量都分发给仍然健康存活的那些机器对不对？那我流量分发器怎么知道哪台机器挂掉了？我怎么知道可以不往你这发流量了是不是？这是谁来告诉呢？因为不是这个出问题的节点已经没了嘛是吧？那它的这个父节点是可以知道这个事件发生，这个父节点能够知道它下面的子节点消失了，然后它能监控到这么一个情况，于是它可以把这个监控直接告诉这个流量分发器，说我这里面有一个Node0这个子节点出问题没有了，说明你流量分发器下流的某一台机器出问题了，具体哪台出问题的话呢？这个的话可以去查到的！那怎么查到呢？这个很容易查，先暂时不说怎么查到的。

于是这个流量分发器就已经定位到Server1出问题，于是流量分发器就不给它打入流量请求了，就把这个坏的Server1取消掉，那么这个流程大家听明白了吗？（父节点发现自己有坏的机器是主动上报的）。

好了，那这个流量分发器怎么知道这个Server1挂掉了呢？刚才说过了，这个Zookeeper有一个非常重要的一个特点，节点是可以存数据的明白了把？就是说当Server1往Node0这个节点上创建了一个数据，那其实我这个Server1就可以往Node0这个节点上去写数据对不对？那这个写数据，我完全就可以把我的这个Server1机器的IP地址或者是这个host一些网络的一些能访问到我的地方，我可以写到Node0这个里面来对不对？那也就是说你请求的这里面每一个节点这个节点都存了一个数据，这个数据里面就包含它与哪台机器比如具体哪台IP地址建立关系是不是？那这个时候你在这个节点里面，比如说大家都很健康的状态下其实你流量分发器完全是可以根据你的父节点把他下面所有的子节点遍历一遍，把这里面的数据全都取一遍，这个时候其实流量分发器就已经知道了它的映射关系，如果你这个节点挂了没关系，我这流量分发器有这个名单，我一查你哪个节点挂了我就知道你哪个IP地址出问题了对不对？大概是这么一个逻辑,这个逻辑还比较清晰把？

所以这个逻辑里面就有这么几个特点，监控功能很重要，还有一个临时节点这个特点，还有这个临时节点怎么与你外部的机器进行建立一个关联以及能够完成整个的一个监控这个意思。

那既然说到监控，刚才我举得这个例子这个监控还是比较单一，我说的这个监控仅仅是一个消失监控明白了把？当某一个节点消失的时候我才给你发一个监控对不对？除此之外还有getData()，就是说你这个节点里面的数据发生变化了你可以发出一个监控明白这意思把？

getChildren()就是刚才这个意思，当我父节点向我管辖范围内这里面的子节点名单发生了变化，我就可以发出一个监控信息好吧？然后还有一个exists() ，这个是一个退出，比如说你监控一个节点，这个节点突然消失了，你也发出一个监控，但是这个exists()这个节点就不一定是临时节点了，它可以是一个永久性节点也都可以，其实这几个都适合于永久性节点，然后临时节点也都适合，就是这么一个情况。

所以对于监控要记住，监控来说是有这三类情况那就是：1.数据修改，2.子节点变化以及3.节点退出这三个类，然后怎么用呢？这个用法大家可能就要注意一下，比方说当我这个机器出了问题了，这个节点没有了，那么这个机器我又恢复了，那么刚才这个节点还恢复吗？其实就不恢复了，也就是说你一次性监控，只要这个监控被触发了，你就需要重新设置，就是说当你这个节点如果重新恢复的话，那你需要重新把这个节点创建起来。

另外一个就是从节点角度，如果这个节点消失了，我父节点就主动的上报一次给流量分发器报警，它上报一次之后，那么父节点跟流量分发器的一条通信的路就段了，如果仍然有一个节点挂了，那么这个父节点就不会再去往流量分发器去通知了，有这样的隐含大家要注意。怎么办呢？（如图18所示）

图18：



就是说你每一次发送监控触发的时候，你这个流量分发器都必须重置这个监控，这监控谁设置的？是你流量分发器对你的父节点去设置的，就是说当有一次监控触发之后你需要重新设置这个监控。

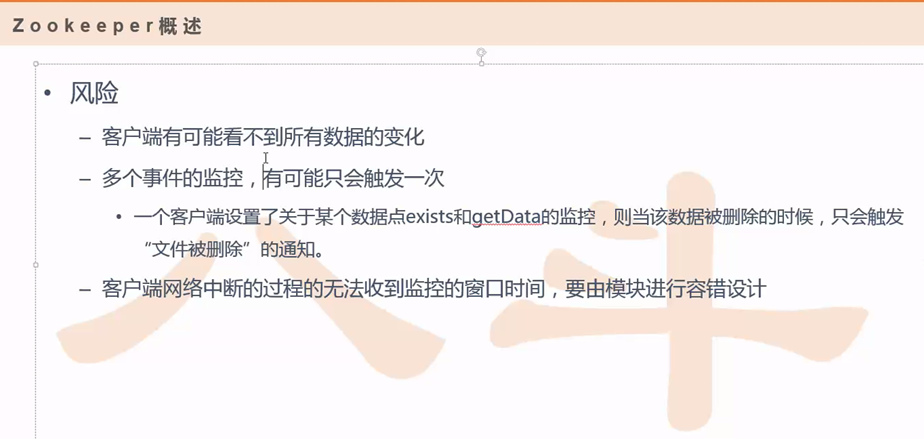
还有一个保证先收到事件，再收到数据修改的信息，什么意思呢？就是你这个监控(getData())它会告诉你是因为什么出现的问题，然后为了保证你这个逻辑，就是你必须先知道你收到的是什么样的报警或者是收到什么样的类型事件然后你才能去了解到底是哪一个节点出了问题明白把？你得先知道有这么一个事情出现了然后再去更进一步的去分析为什么，就不要收到了一个监控，你还不知道监控是什么具体是几点退出了还是这个子节点发生变化了，你还没有考虑清楚你就开始去调研这个是什么问题，这个的话不太适合的。

还有一个如果一个创建节点的话呢，它也会触发你的父节点的监控点，什么意思呢这个？就假设说我父节点里面有一个节点，我这个时候又创建了一个，这个时候父节点下面有两个子节点了对不对？，那这个时候它的那个子节点列表就发生变化了是不是？,那这个列表发生变化了，这父节点的监控就同时被触发了明白了把？

并且呢这个创建的新节点变化了，就因为我从无到有，如果假设有一台服务器它就监控着我这个节点，但是我这个节点不存在，你就一直监控着我，如果我这个节点一直存在的话，那其实我同步的会给你发一个存在的一个信息，一个是exists()信息报警，然后同时你的父节点也会发出一个报警，相当于这个节点里头有一些报警的话其实你父节点也会被影响，大概是这么个理解。Create和delete都会有这样的情况。

然后风险是什么意思呢？（如下图19所示）

图19：



风险这个大家看一下就可以了。就是有的时候你的客户端是看不到所有的数据变化，就因为你这个监控毕竟是通过网络传输的，然后有可能你这个监控被触发了但是由于网络的原因，你是看不到数据的变化，就是说你还接不到这样的一个触发事件是吧？所以有可能会有这种的隐含，由于网络的原因可能导致了你这个信息接不到。

还有一个就是多个事件监控有可能只触发一次，就是刚才说过假设我这父节点里面我有一个子节点我删除了，那这个监控的话，那因为我删除的是会发一个退出的一个警告对不对？，但是同时会导致我这个父节点向下面这个节点的数目会发生变化，所以父节点也会发送一次警告，但是由于这个监控因为有两次，但是你真正去通知的话你只能通知一次只能触发一次。

比如说关于某个数据它既绑定了exists节点又geiData这个监控，那当这个数据被删除的时候只能触发被删除的通知。

然后这个是那个网络中断跟刚才说的差不多，由于网络的问题有可能你收不到这样的一个监控，所以这就是风险，风险就是说你这个系统可能会存在这样的问题，但是这种问题通常来说是概率出现不是很大，但是它可能会存在这样的问题。

然后你对Zookeeper去访问的时候，就是说在Zookeeper里面因为它特别像一个文件目录数，比如说在这个linux环境下你对某一些目录或者某一些文件去访问的时候都是有权限的对不对？这个大家没问题啊(如下图20）

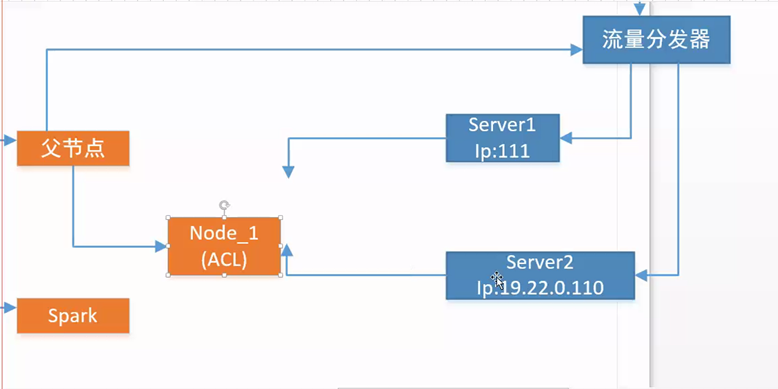
图20：



其实在做Zookeeper你对某一个节点，它也是有着这么一个权限的，那在这里面的权限就叫访问控制，这个访问控制叫做ACL，那访问控制的话比如举个简单的例子：如上图21里面有一个ip，然后这个IP后面才是地址的网段，然后第三列的信息(READ)是读，那这个很好理解，也就是说表示IP地址以19.22开头的机器有对这个节点的读权限明白了把？

那这个权限是直接设置到了某一个节点上，比如说我对这个Node1节点（如下图21所示）

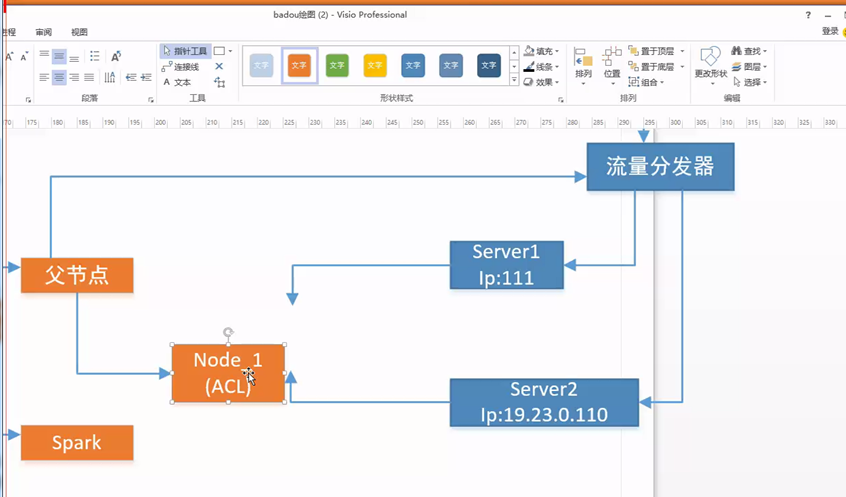
图21：



我在Node1下面做一个ACL，那只有当我这个Server2是ip为19.22.0.110，那么这个Server2就有权对这个Node1进行访问并且我可以读你的信息是不是？

假设说我这个网段变了，变成23（如下图22所示），我这个机器就不允许访问Node1这个节点了明白了把？

图22：



好了，这个例子是个比较简单的例子，这个是以IP地址去做一个访问控制，其实类似的方法有很多种，我们先来看这个读(READ)，类似与读的话它其实还有创建，有读，有写，有删除，有管理（如下图23所示）

图23：



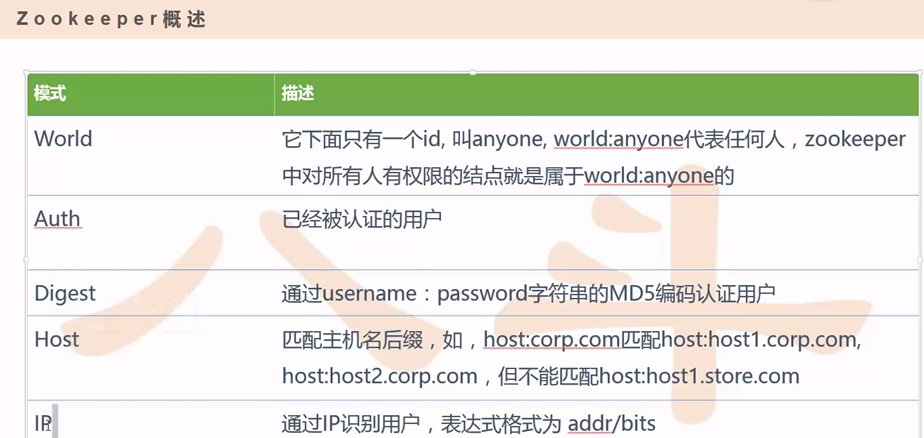
逐渐恶化ADMIN是什么意思呢？就是可以设置节点权限的权限，相当于是把它的权限算是比较大的了比较高的了，如果说我们没有读权限，但是我有ADMIN权限怎样？我可以把你的节点直接改掉，相当于是一个超级管理员一样，所以这五个权限是可以配置到整个模式的第三个信息里面（如下图24所示）

图24：



那在这个三元组这个是什么意思呢，就是第一个信息表示你的机制，第二个表示你的用户，第三个表示你的权限，三个区域的不同的角色是不一样的，那类似于机制这种呢，IP只是其中的一种，还有其他（如下图25所示）

图25：



刚刚我举得例子仅仅是IP的这种情况，那我们看一下World，World是什么意思呢？World就是anyone，那么anyone是什么意思呢？就是任何人的意思，就是说你任何一个用户或者任何一个机器你对我某一个节点是有某些权限的是吧？就是World的下边只有anyone这么一个，这个anyone就是代表任何人。那具体是任何有读的权限也好还是写权限也好还是怎样的就需要套这样的一个式子是不是？（如下图26所示）

图26：



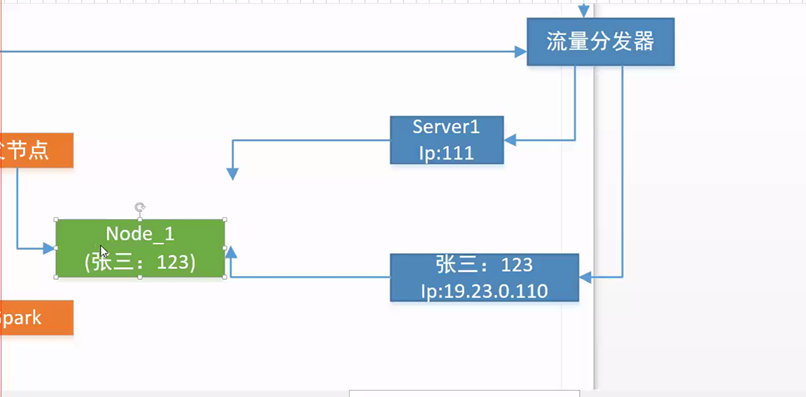
这里面的IP就可以写World的是不是？，中间就可以写anyone，最后就是你可以让所有的人对我这个节点是有什么样的权限是不是？

然后机制这块你也可以改为Auth，其实Auth这个东西官方是说已被认证的用户，然后其实看了一下这个Auth其实你如果通过Auth去创建一个权限的话，那创建是失败的，但是如果你想创建一个Auth权限，它也是可以通过用用户名和密码的方式去控制，但是这种用户名密码，就是你在创建节点的时候就已经给它了一个权限的设定是吧？

比如说这个权限名就是张三，然后密码是123

首先这个张三这个用户名和密码前提是已经在这个Node1这个节点上配置了，已经赋给这个权限了（如下图27所示）

图27：



如果这个带ip的这个张三还真的来了，并且密码也都对，这个机器就可以访问了，明白这意思把？

但是你会发现Digest也是可以通过用户名和密码的方式去访问，但是Digest和Auth有什么区别呢？但是这个其实大概看上去没什么太大区别，就是说这个Auth除了通过用户名和密码这一种方式之外，它还可以通过其他方式，有一个机制不知道怎么写了，就是那个：Kerberos，就是Kerberos,通过这样一个可以被系统已经认定的用户，所以通过Auth的方式它有两种认证方式，一个是类似于通过Digest模式用户名密码，一种可以通过Kerberos然后也去提前给你配置好一个用户。

那么主要的是Digest用的比较多，通过用户名和密码。

Host的和IP类似，IP呢是刚才我们其实已经体会到了，IP是可以通过IP地址进行识别。

Host的这个很好理解，你可以通过一个HostName去识别。

其实你把这几个模式你都可以套入到（如下图28所示）的上面来就可以了

图28：



然后后面那些就可以参照不同机制或者模式去修改，比如你的Host的就是你的HostName，你的IP的话就写图29里面的网段，然后后面的是读权限和写权限和那些什么创建，删除等等权限。。

好了此时进入中途下课，接下来进入Zookeeper笔记2！！！