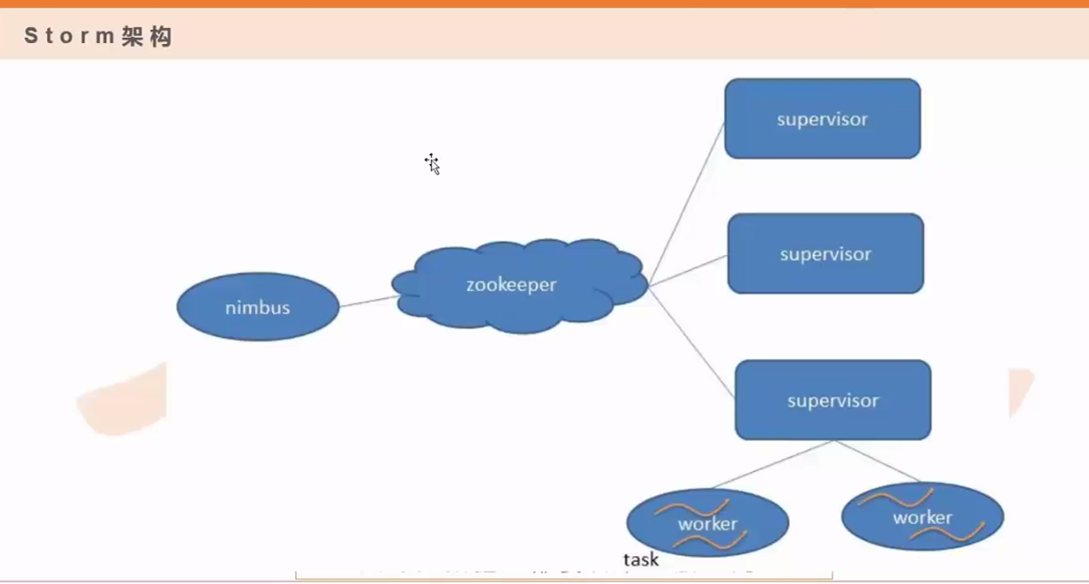
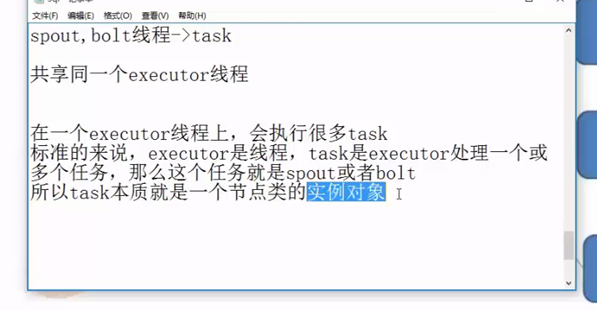
图1：



如上图1所示中的Storm架构我们说过了啊，有主然后有从，中间靠着zookeeper进行一些联系和调度，然后主呢主要的工作就是用一个资源分配和任务的调度，然后另外这个从是干嘛的呢？从就是主要来接收nimbus它分配的任务，然后通过这个任务之后，supervisor在它的内部去启动和停止属于自己管理的进程，这个进程在整个的Storm架构里面叫做worker，然后nimbus和supervisor分别代表不同的机器，是一个节点。

另外一个很重要的概念就是这个task就像两根在worker里面的两根线一样，这个task是什么意思呢？task你可以暂时认为是一个线程，目前大家先记住就是我们上一次课说的那个你开发时候有两个很重要的原语，一个是spout还有一个是bolts对吧？你可以认为这个spout或者这个bolts这两个角色都是一个线程，然后这个线程就是个task，暂时先这么认为，在storm在它最细的粒度就是一个task，另外一个由于在storm架构里面是一个多线程的这么一个场景，所以同一个spout和bolts会共享同一个物理线程，所以这块有点绕。我这边大概写一下好吧？（如下图2所示）



就是说这个spout和bolts线程，这个线程叫做task，然后因为这个task刚才说过它可能是一个线程，其实很多个task其实它是共享同一个executor线程，就是在这个Storm里面每一个spout和bolts在整个集群里面执行，说到这就是说每一个spout和bolts都会被当作很多task在整个集群里面执行，然后每一个executor是对应一个线程。

在一个executor线程上，会执行很多task，所以task和这个executor执行的关系也是这样的一个关系，这个task是隶属于executor，所以如果标准的来说executor是真真实实的线程，但是这个task这个线程里面处理的一个或者多个的任务，task是executor处理一个或多个任务，那么这个任务就是这个spout或者是bolts，大概是这么一个关系。

所以task本质就是一个节点类的实例化对象，相当于这个task你可以把它当作一个任务一样，这个task是一个任务，这个任务是由这个executor线程来执行的，然后这个executor线程是可以执行很多个任务，然后这个task本质其实就是一个实例化对象，是这么一个关系，并且这个executor它是可以处理一个节点，但是这个节点是可能有多个实例对象,也就是说这个executor和你这个task可能是一个一对多的关系，然后你可以通过一些配置来调整你的并发度，比如说在你后面看的话还有一个set\_num\_task，如果是提前有这个学员预习的话，就是这个set\_num\_task是来配置一个executor同时处理多少个task。

所以默认情况下如果你不设置这个set\_num\_task函数的话，这个函数set\_num\_task写的不一定对，不过大概是差不多，如果你要是不设置这个set\_num\_task函数的话默认就是一个executor执行一个task明白了把？

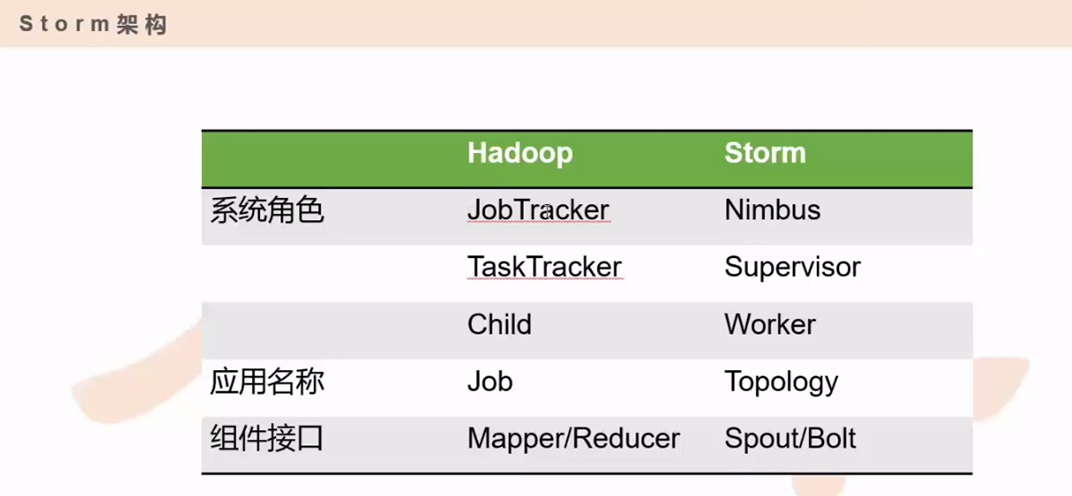
Work是进程，work里面有多个executor。

Work是进程，然后executor是线程，task是这个executor里面执行的一个实例对象，这个executor你可以执行一个实例对象也可以执行多个实例对象，然后具体你可以通过如果执行多个实例对象的话，你就需要去写代码然后通过一个set\_num\_task函数去指定配置，task本质上说它其实不是线程。

然后我们再看一下整个的Storm里面的架构（如下图3所示）

因为我们之前已经有了这个mapreduce还有hadoop一些相关的经验了，mapreduce里面的架构进行一个对应学习的话你会发现Storm的架构和mapreduce架构很相似，只不过是那个命名不一样。

图3：



那在这个hadoop里面或者是在这个mapreduce里面就是这个主叫做JobTracker是吧？对应的storm里面叫做Nimbus，然后对应的是TaskTracker和Superviosr，然后进程在这个hadoop和mapreduce里面叫做Child，然后在Storm里面的进程叫Worker，在mapreduce里面如果你提交一个任务叫做提交一个Job，那么在Storm里面默认提交的是一个Topology提交一个网络是吧？然后组件接口就是你之前开发的时候，是面对的Mapper和Reducer两个函数去做开发对不对？那么在Storm里面类似，但是就不叫Mapper和Reducer了，就分别叫做Spout和Bolt。

所以在去开发Storm的时候其实跟有些思想跟开发mapreduce很相似，也就是把个别的一些函数填好就可以，就是把你自己的那个逻辑往相应的函数里面去做一个像填空一样，把这个空给填上并且能够符合你们的逻辑这样一个情况。

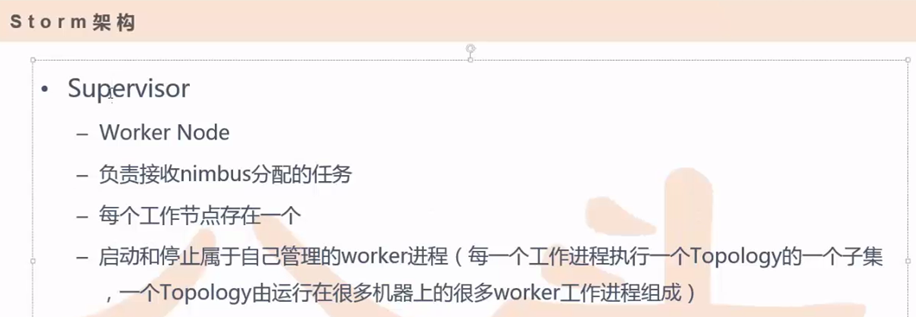
那么我们继续往下，那既然说到Storm架构那然后我们先要从主来开始说（如下图4所示）

图4：



然后这个主这个角色其实大家已经知道了，主要是负责资源分配和任务的调度，类似于hadoop里面的jobTracker，负责你的代码分发，这个应该大家很好理解了，对应的主就是从(如下图5所示)

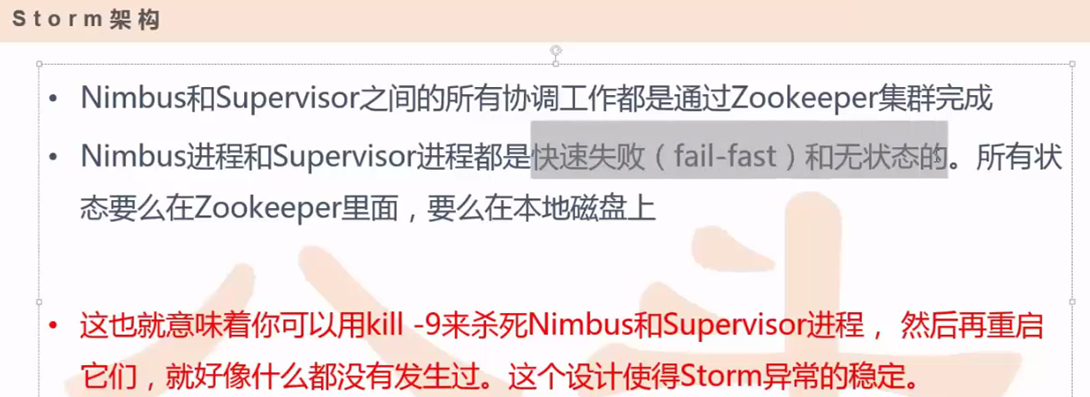
图5：



从Supervisor主要是来接收主给的分配的任务，然后Supervisor去执行，每一个工作节点其实就存在了一个Supervisor或者是一个Nimbus，然后启动停止自己管理的worker进程，也就是说这个进程因为在这个节点上去运行的，那一个节点上可能会运行很多个进程对不对？，那具体这个Supervisor拿到了这个主分配的任务之后，让哪个进程来执行，这是由Supervisor来决定的。

然后呢所以呢根据我们刚才的那个上图1我们已经看的出来，这个主Nimbus和从Supervisor所有的协调工作都是通过Zookeeper这个集群来完成的。如下图6所示

图6：



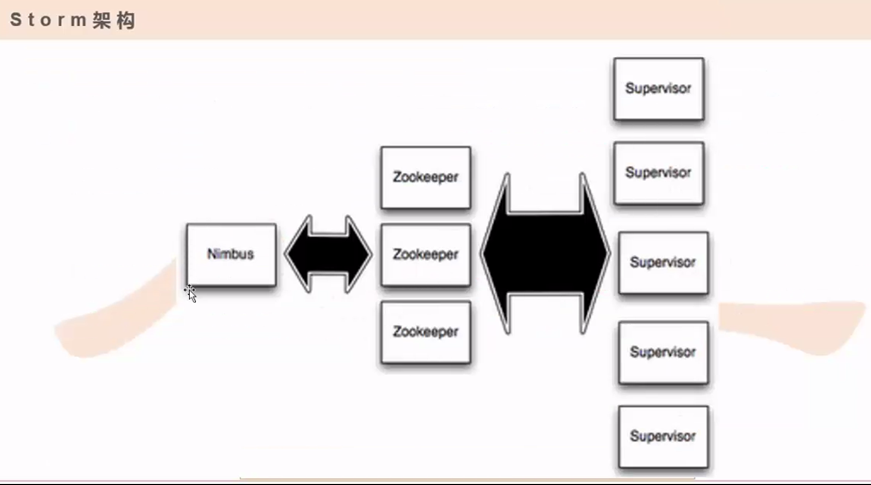
还是有一个很重要的特点就是大家要知道一下，什么叫快速失败和无状态，也就是说在我们mapreduce里面的如果你的JobTracker挂了，那相当于集成框架就挂了，这个大家没有问题是吧？但是呢在你的Storm集群里面如果你的Nimbus挂了，你只要是之前已经提交过这些任务这个Topology是不受影响的，但是它会影响什么呢？会影响你未来要提交新的任务，因为你没有主了，没有主了你就没有办法去分发你的代码去分发你的任务，所以没有主的问题主要就是影响你之后提交新的任务，这样的一个情况。

另外一个不能够提交新的一个任务这是一个，另外一个就是Supervisor如果出现一个问题是没有办法调度的，如果出现一个主真要挂了怎办？那就是把这个主重启一下，为什么呢？为什么重启以后相当于是什么事情都没有发生过一样，因为所有的数据都存在了zookeeper上，我们咋天刚学了zookeeper，因为zookeeper里面它创建的节点都可以存储少量的数据是吧？

这个数据不管你是主的数据还是从的数据以及worker的数据都是在zookeeper里面通过这种类似于目录数的结构进行维护。

另外一个如果要是说你的Supervisor挂了会不会受影响呢？其实Supervisor挂了也是不受影响的，然后就是把它重启以后当什么事情都没有发生过。那这个下图7刚才我们已经说过了

图7：



主要是三层，主，zookeeper，然后从Supervisor。

那么我们然后看一下这个Worker，Worker是干嘛的呢？Worker它是个进程处理组件逻辑的进程，就是你有什么任务是由这个Worker这个进程来帮你处理的，一个Topology它肯定是有多个进程的，但是这么多个进程肯定不能全部跑到一个机器上去，很多个从节点都会有这么所有人员这么一个Worker属于秘书的这个Topology的子集，如果你提交了一个任务，这个任务是有很多个worker是帮你同步的去运转，而且这个worker具体分布到哪一个位置，它是属于那种散落在不同的从节点上的，然后这个worker进程里面是有很多的这种Executor，就是很多个这些线程，那这里面举了一个例子（如下图8所示）

图8：

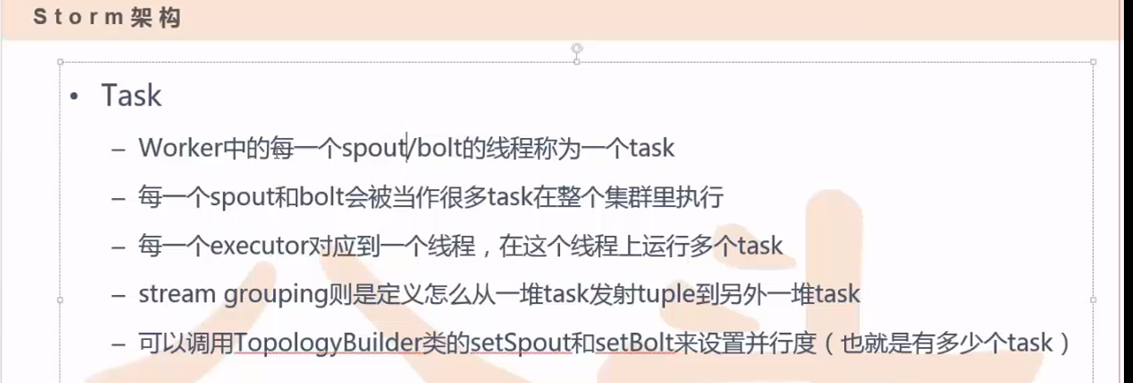


对于Storm来说有一个很重要的概念就是一个并行度的概念，因为有了并行度，这个并行度大家可以简单的理解为就是我们在执行mapreduce里面的时候，你的map的一个并发度或者你的reduce里面的一个并发度，就是你对这个map的进程越多或者reduce进程越多，那么你每一个进程处理的数据相对来说就是变少了对吧？然后使得到你整个的任务并发起来之后你的任务就可以快速的完成，就相当于是你这个任务能够一个是性能方面一个是快速的处理方面都可以得到一个有效的维护起来。

那在这个storm里面它是有一个并行度，这个并行度是可以通过配置去配置的，假设说我配了一个并行度是300的一个Topology来说，如果我们用了50个进程来执行任务，那么每一个进程就会平均一下，这个300除以50的是每一个进程里面就会有6个task，那么storm会尽量均匀去把这些工作分配给相关的worker上。

然后这个task，刚才说了一个task相当于是一个spout或者是一个bolt（如下图9所示）

图9：



然后worker中的每一个spout和bolt线程都称之为一个task，所以这块这个task如果你不考虑executor的存在的话你可以认为是task就是个线程，但是本质task就是一个对象，所以你把task你把它当作一个线程去理解也是没有什么太大的问题。

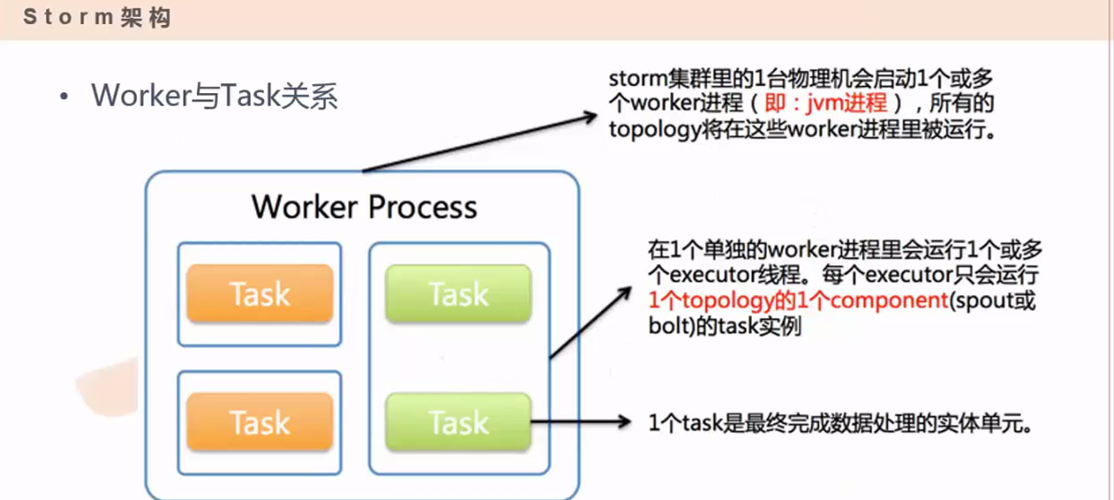
然后如果真的要去细分的话，那就是一个executor里面默认情况下这个executor和这个task是一对一的关系，但是你这个executor里面还是可以运行很多个task，这里面有个需要说明的地方，如果一个executor里面存了很多个task，那这个时候想问大家，那它这里面它这个task是同时运行的吗？答案：不是的哈！！大家请记住如果你一个executor里面有多个task的话，那这些task并不是并行执行的，它只会在这个task里面选择一个来执行，而且它选择哪一个呢？这个都没有关系的，它相当于是一个轮转的这么一个，比如说这个一个executor里面有10个task，那想做到这种雨露精湛，我第一个task我执行了一个消息，那么我第二个task就开始一直轮转,大概是这么一个过程。

然后之前我们说了一个分发机制是很重要的一个Steam grouping，这个grouping其实定义了怎么从一堆的task 发送tuple到另外一个task的过程，这个策略我们上次课已经说过了，这个应该好理解。

那么接下来关于代码这一块怎么去设你的SetSpout和怎么设你的setBolt的并发度，这个实践部分再去理解。

所以接下来我们看一个很重要的一张图（如下图10所示）

图10：



上图10里面的图很重要，这个图直接就描述了你worker和task之间的关系，就像刚才我们说的那个task和worker之间的关系大家有可能不理解的地方，你可以通过图10里面的图再进一步的加深印象好吧？那你从这个图10里面的图其实你很容易的发现，我外面的那个最大的框叫做一个进程空间，在一个进程里面有三个小框，那每一个小框就叫做一个executor，每一个大框叫做一个worker，这个小框里面有橙色的有蓝色的，不管是橙色的还是蓝色的上面都标了Task，那也就是说有可能一个executor里面就有一个Task，也可以存在另外一种情况，一个executor里面可以存在多个task，其中这个task是最终完成数据处理的实体单元，这里面可能需要在更大家再多说一些，就是一个executor里面处理多个task，其实这种多个task有什么用？刚才我已经说了尽管有多少个task，我只会执行一个，那为什么这个接口还允许开放让我设多个task这么一个原因？

那有一种可以理解的地方就是说我设了多个task之后，这个方便我之后的扩容这是一点，为什么说方便扩容呢？首先要知道我们这个topology这个代码的一旦提交到你的nimbus之后，然后你的task的数量就随之而定，就是你的任务只要你一旦提交了，你这个task数目就已经固定下来了，而且这个数量一旦定了之后，你这个数字就不能变了，为什么这个数字怎么定的呢？就是在你的任务里面写好的，通过一个配置去写死的，所以一旦这个数字确定了之后，这个task数目是不变了，甚至就是你重启了这个任务，它也不会变，除非你把代码改了，然后重新提交，你这个task的个数才会变。

所以你想改这个task是比较相当于是静态把，比较不是那么特别的灵活，那这个时候我设了多个task是有什么好处呢？好处就在这里面，如果我设置了多个task，ok,可能会我设置多了，然后导致了很多个task是提前被我浪费掉了对不对？，如果我们设了四个executor，那默认的情况下就是每一个executor处理一个task，那这个时候全局来看你就会发现就是比如四个executor，那么默认就是四个task对不对？那么一个executor中处理一个task就OK了，如果我们现在发现这个并发不够了怎办？那处理的速度已经跟不上了，如果我们想把这个并发度调高一些，那么我们调高到8个，但是你这个task的数量仍然是四个也是没有任何的意义对不对？所以应该怎么办呢？就是我看看能不能把这个task提前多设置一些，如果是这样调整并发度的时候可以能够在这方面做一些个扩展，然后一个spout或者一个bolts其实他们都叫做一个Conpoent，如果是涉及到storm开发的时候呢，首先你要去创建一个配置的一个对象（Config conf=new config）,你可以通过这个config对象，能够得到你当前这个集群默认的一些配置（如下图11所示）

图11：



比如说这个集群里面默认配置什么呢？就是你一个spout一秒钟最多能发出多少个消息，比如说类似于这么一个配置，或者说一个消息从头到尾一个超时时间，如果是这个超过了某一个超时时间，这个消息没有被成功处理的话，它就默认被当作一个失败的消息直接给扔掉了。

那这是另外一种，其实类似于这样的参数，那系统其实都是有默认的配置来管理的，你创建了一个Cpmfig配置一个对象，其实对象的内部就已经把这些默认的参数就已经继承过来了，在默认的配置基础之上你可以去修改个别的一些配置，比如说设置Worker数目，你可以通过这样的一个方式你可以设当前我们运行这个任务需要几个进程来完成。

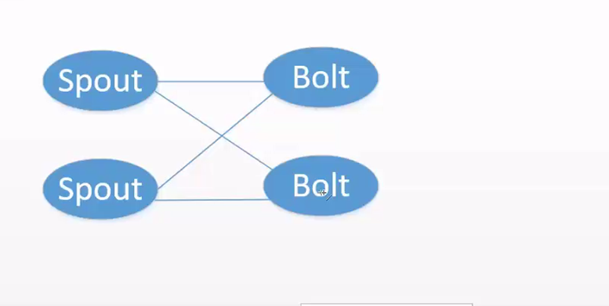
我们之前说这个Storm，你想提交一个任务，这个任务其实就是一个拓补，那什么是括补呢？就是一个网络是吧？那这个网络怎么来描述呢？，那么我们看一下在如下图12中在一个相对比较完整的网络里面，有这么几个原语，一个是Spout一个是Bolt，然后紧接着我在我的这个Spout是相对于整个的网络的一个输入点，它相当于是整个的消息的发送源。

图12：



那Spout它作为一个原语来说它是可以设置并发度的，这个并发度就在这个后面的参数是2，那这个时候我们可以通过画图的方式去理解（如下图13所示）

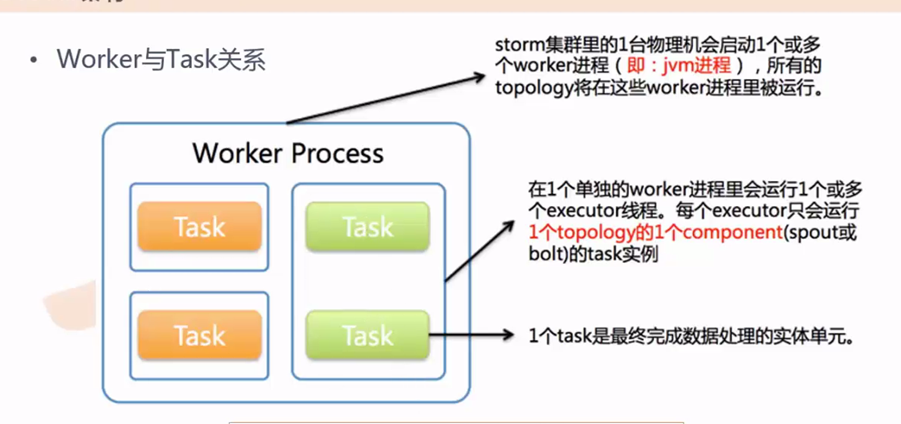
图13：



因为图12中设置了两个并发度，就相当于是有两个Spout线程它可以帮我来同时做事情，就相当于你的mapreduce里面，你一共可以启动多少个map进程来帮你去同时处理数据一样，那提高了并发度的好处就是加快了你的数据的处理速度对不对？，那图12中这个Bolt它的并发度也设置了2个，这个Bolt和你的spout怎么相连呢？是通过后面有一个Grouping()，这Grouping是通过shuffleGrouping的方式，shuffleGrouping的策略大家应该没有问题把？就是通过随机的方式，所以图13前面的那个spout是怎么和后面的Bolt相连呢？完全是随机相连的。

那另外图12中设置了NumTask（4），这个4就是像我刚才图10里面描述的一种特殊情况，一个executor里面存了两个task（重复如下图14所示）

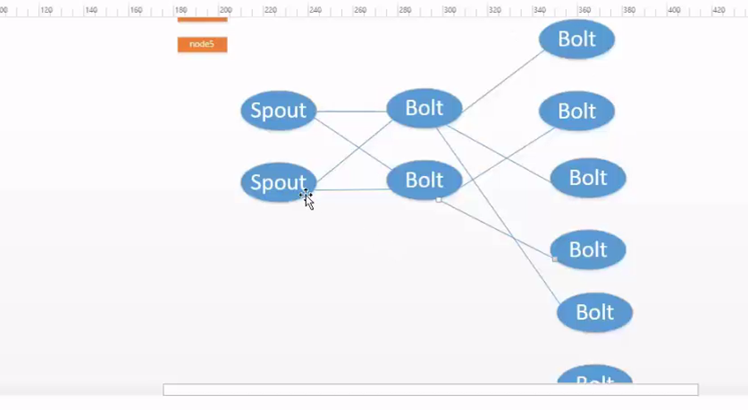
图14：



那你看，我一共针对这个Bolt这个角色的我设了四个task，但是它的并发度仅有两个，那四除以2等于2，那相当于每一个Bolt的上面其实它里面有两个task，相当于你真正执行这个Bolt，其实它是一个executor，那它这个executor内部其实有两个task这样好理解了把？

然后我接下来看图12中的第三个Bolt，那这个Bolt的并发度是6个（如下图15所示）

图15：



然后也是通过shuffleGrouping去随机相连，所以你会发现通过这三行代码就可以完完整整的把一张网络图给你画出来了对不对？

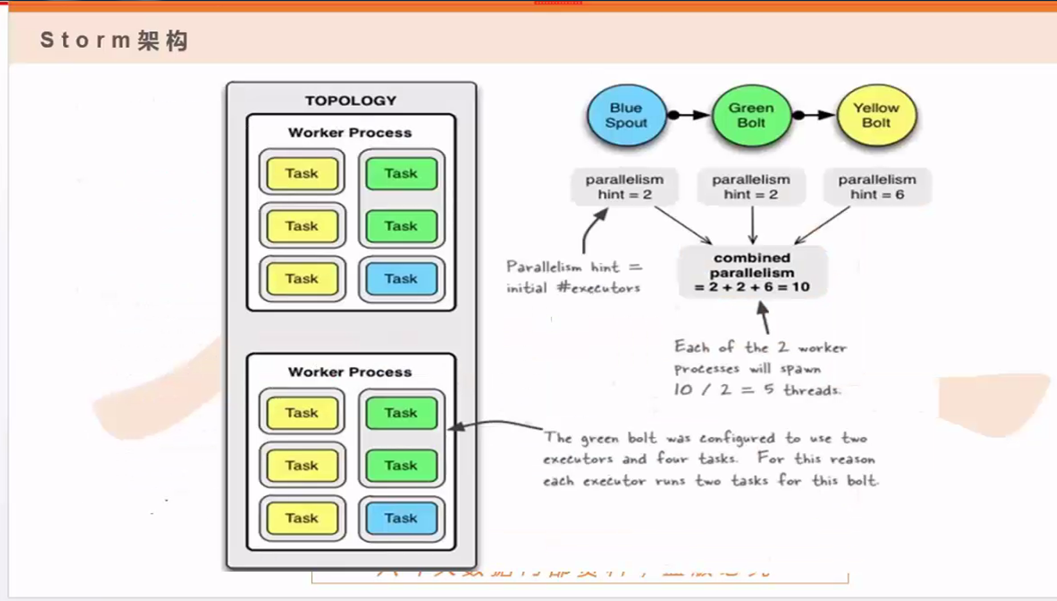
然后你再看一下，这全局并发度多少个？那就是2+2+6=10个，一共10个线程同时启动，可以这么理解（如下图16所示）

图16：



那为了跟进一步的让大家能够了解这样一个过程，就有个图（如下图17所示）

图17：



然后配合着下图18中的程序来进一步的描述

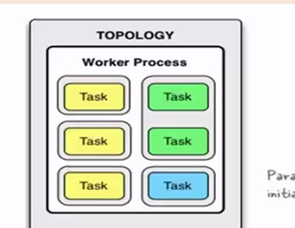
图18：



Spout并发度是不是有两个？它的名字叫BlueSpout（蓝色），然后第二层的Bolt叫做GreenBolt（绿色），然后第三层的Bolt叫做YellowBole（黄色），所以你会发现从17的图中就是一个蓝色和绿色和黄色这三个，然后因为这个BlueSpout的并发度是2，然后GreenBolt的并发度也是2，YellowBole的并发度就是6，总共加起来就是10个并发度，然后因为我前面图18中有两个进程是吧？这样的话每个进程就承担了5个线程，那我对应一个TOPOLOGY来说也有两个进程，然后这进程空间有两个，图17中的图里面上面是一个，下面是一个就是两个，然后其实中有5个线程。

那么我们随便找个进程来看，上面的那个（如下图19所示）

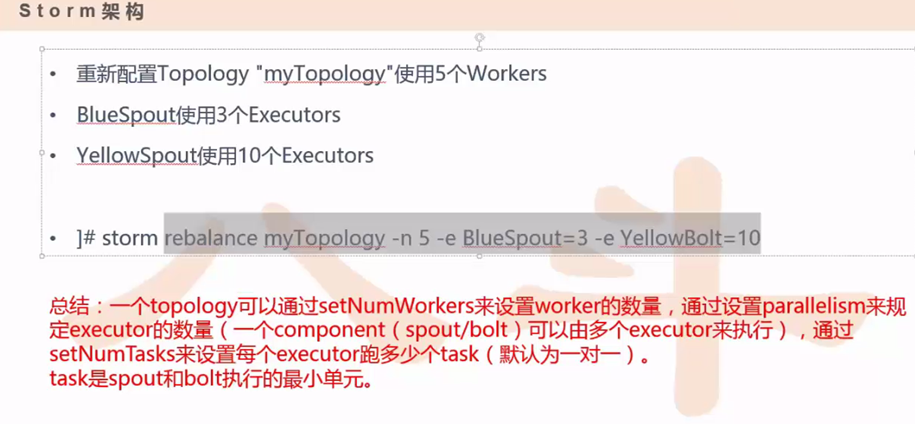
图19：



其中绿色的部分，因为在图18中它多设了一个setNumTask，所以在这个executor里面同时存在多个task的情况，通过这样的一个图其实就相当于是把刚才画的（如上图15所示）的这个并发度还有这些每一个角色的相互关系就能够表达的很清楚了。

那我们再继续，刚才说过了，我们说设了多个task，由于我们之前可能设置的并发度可能比较小，然后导致了很多的task是浪费的状态，所以为了我们便于我们以后的扩容，可以存在多种类似于这样的情况，一个executor里面有多个task的这样的情况，是为了方便扩容，既然你的task数目是不能变得，那么我们就可以通过调整你的并发度来去调整，你可以通过如下图20的图中命令，然后随时可以动态调整你的并行度（如下图20所示）

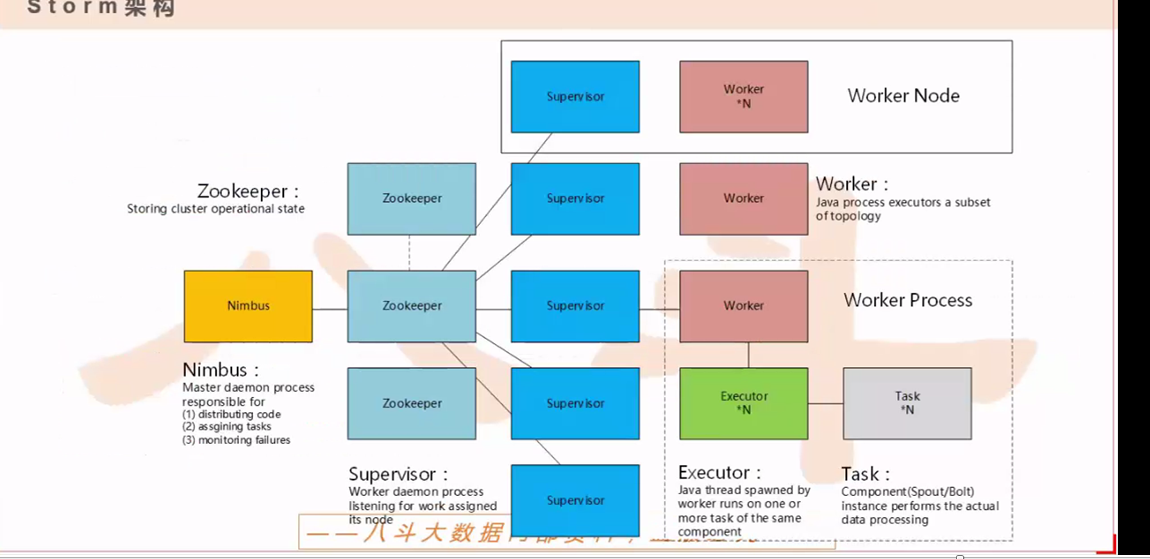
图20：



你像图20中里面，-n是什么意思呢？你刚才不是说有两个进程吗？ 我可以通过这样的命令把那个图18中的conf.setNumWorkers(2)改成conf.setNumWorkers(5)，通过-n的形式把整个的进程数目由2改成5，然后如果你觉得中间有哪些个并行度呢？你觉得太小的话就可以调大一些，怎么调呢？也可以通过这样的一个方式，前面把你的Spout名字写上来然后等于一个数字然后-e相当于是可以同时执行多个，可以通过这样的方式去临时调整你的并发度。

好了，再回过头来去看一个架构（如下图21所示）

图20：



这个架构里面有很多的角色，这个大家已经分的很清楚了，那么我们再看一下这个Nimbus其实它主要工作就已经列出来了，但是它是主控然后提交任务并且做一些任务的分配然后做一些监控，这是Nimbus。

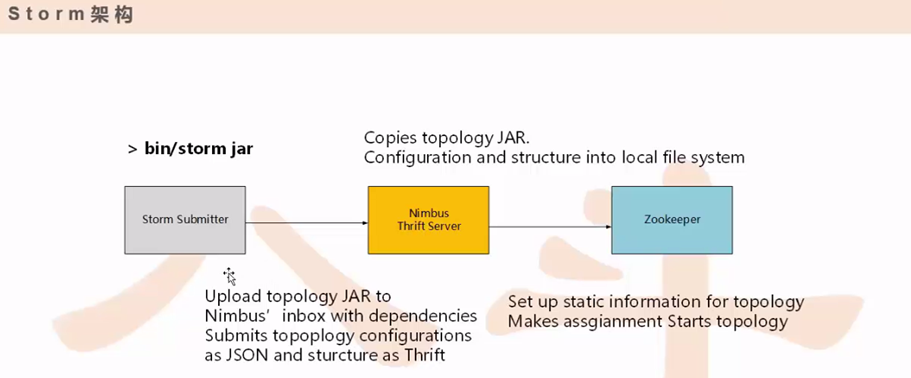
然后Zookeeper主要是用来做一个协调，然后把一些数据进行一些存放。

这个Supervisor这个从主要是用来接收这个主分配的任务，然后管理属于自己的一个worker进程。

然后worker进程在这个worker内部，其实刚才我们说了很多都是围绕着这个worker内部的一些线程它们之间的关系是怎么进行展开的，然后从这个图里面你会发现一个worker的里面可以对应多个executor，然后一个executor默认情况下是对应一个task。所以这个理解起来应该没有什么太大的问题。

然后我们再进一步的看一下（如下图21所示）

图21：

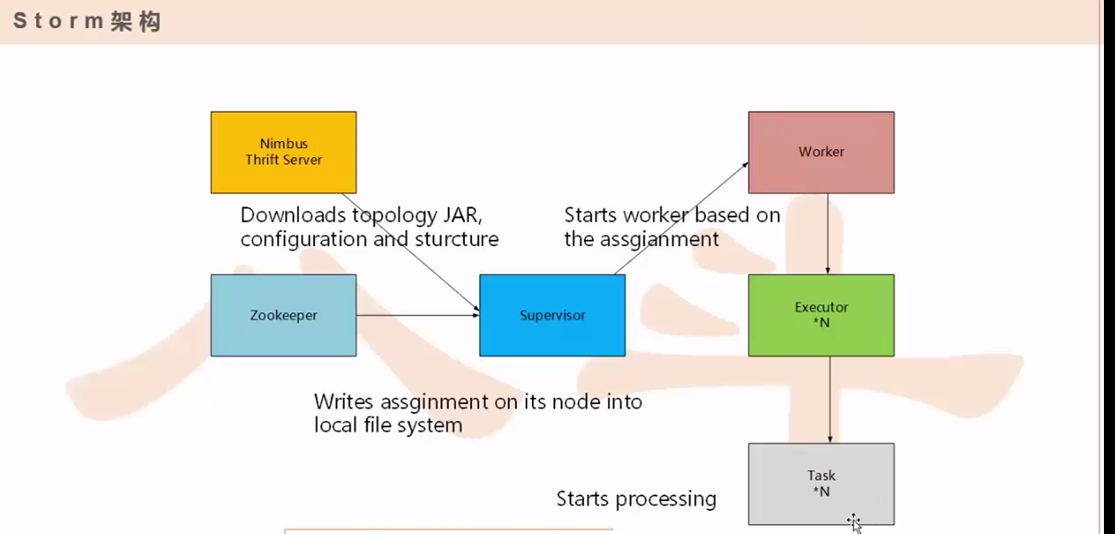


就是当一开始你开发是通过你的主去提交你的任务，不管你是在这个主上开发啊还是从别的信息进行开发，首先你想提交任务，你必须要跟你的Storm这个nimbus进行联系，那怎么联系呢？就是你提交的这个客户端叫做一个Storm Submitter，Storm任务的提交，就是在你的这个机器上你需要去执行一个提交命令，这个提交命令其实很简单就是一个storm.jar然后后面跟着你的类。

紧接着你想提交任务，你需要跟你这个nimbus这个Server，这个Server叫Thrift Server，为什么叫Thrift Server呢？之前我们说过Storm 不仅仅是虽然它的源码是通过java开发的，但是你可以通过java去开发，也可以通过其他语言去开发，比如说是python，就可以开发一个Storm，那不管是什么样的语言开发最后你是提交了一个任务上来，然后由这个nimbus这个 Server来去接管，然后当这个nimbus Server得到了你的任务之后，好了这个时候该通知你的Zookeeper，然后把你的任务通知Zookeeper，然后因为之前我们上节课说过了Zookeeper是可以帮你去存一些信息并且可以代劳帮你把一些任务分配出去。

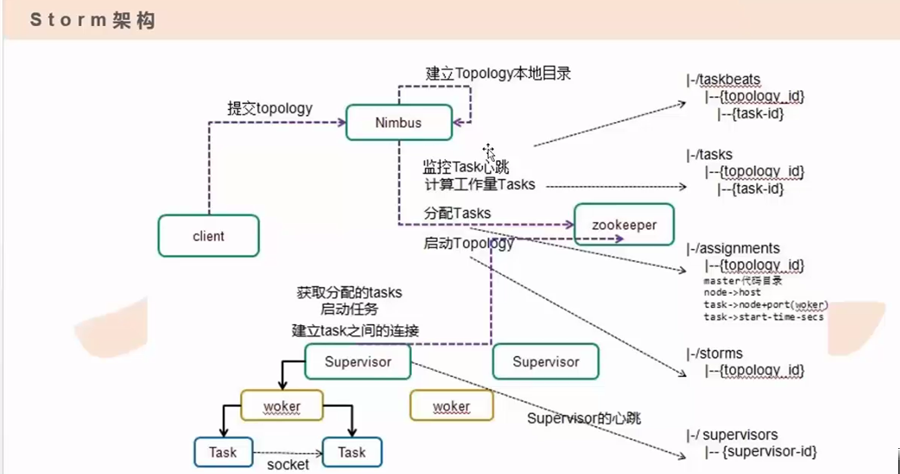
然后接下来这个Zookeeper，因为它上面已经有了一些新的一些任务了，然后接下来这个Supervisot是可以通过这样的Zookeeper的一些任务它的一些列表的状态变化，它是可以得到一些感应的，一旦感应到了有新的任务到来，然后Supervisot就开始去取任务，那去哪里取任务呢？因为Zookeeper里面是存数据的，它是可以存一些Nimbus的一些地址和你的路径，然后这个时候Supervisot可以通过Zookeeper反馈的相应的提示去从Nimbus上面把一些你的代码和配置和结构进行下载下来，下载完以后开始在它的内部去启动相应的进程，去把任务分发下去，分发完了之后，那每一个worker根据你之前这个任务里面设置的并发度以及设置这个task的数目就开始启动相应数目的executor的个数以及task的数目，就是这么一个过程（如下图22所示）

图22：



然后这个图23这个图相对来说，看起来还比较复杂一点，其实这个图跟刚才讲的那个图20中的图的描述都是差不多的（如下图23）

图23：



然后在这个图23这个图里面，相当于是给大家体现了一个 概念就是说因为我们咋天学了这个Zookeeper，这个Zookeeper是一个类似于目录数的方式来管理不同的节点对吧？那可以从图23这个图里面右侧其实就可以当作一个Zookeeper的目录数，如果你打开一个结构的描述，那具体怎么看这个结构呢？大家也应该知道方法是吧？可以通过Zookeeper自带的一个client就可以把这个目录结构树进行打开然后进行分析

好了接下来休息中，请看Storm进阶笔记2！！