图1：

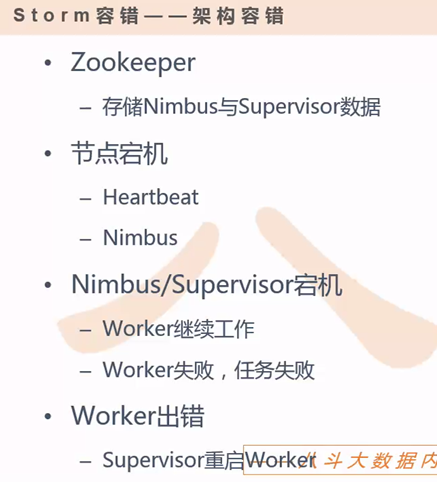


然后我们进入第三个环节，第三个环节就是Storm容错，我们看一下容错，容错的话我们之前已经大概了提到了一些，相当于这里面做一些总结性的说明。

那首先容错最重要的工程就是Zookeeper，这大家都没有什么疑问是吧？那个Zookeeper为什么它可以能够帮助你这个架构可以进行容错呢？就是因为你这个主的nimbushe和从Supervisor是都可以在Zookeeper留下一些约定好的一些信息，比如你这个Supervisor启动的时候，它就会主动的在Zookeeper上注册一些相应的信息，然后这个时候Nimbus就可以发现这个信息变化了，然后再去进一步的查询发现了有一个新的Supervisor上来了上线了是吧？那Supervisor就可以在这个Zookeeper上留下一些心跳。

那接下来Nimbus就可以通过这些心跳来对Supervisor进行管理和检测然后并且来检查一下是否有坏的节点（如下图2所示）然后另外Supervisor因为它可以存储少量的信息，它可以存Nimbus的信息也可以存Supervisor的一些信息，所以由于这个特点全局就有一个无状态的一个特性叫fail-fast，就是快速失败。

图2：



不管你是这个Nimbus也好还是Supervisor也好，挂了之后不会影响之前的一些任务这个是一个容错的一个地方

另外一个就是这个Worker其实它也可以通过Zookeeper把自己的一些状态情况也可以通过一些心跳的形式汇报给Nimbus，然后Supervisor进程可以通过Zookeeper把自己的状态也可以通过一些心跳的形式回报给Nimbus，可以大家记住可以通过心跳的形式大致有两种，一种就是worker来汇报，第二种Supervisor来汇报，那么worker汇报什么呢？就汇报他的孩子，他的孩子是谁？就叫executor，那Supervisor来汇报什么呢？就汇报自己的状态，那这个是架构容错。

那么接下来呢还有一个数据容错，那数据容错主要是有两个点来保证的，第一个很简单通过一个超时时间(timeout)，那也就是说当有一个消息在指定的时间内没有被完全处理（如下图3所示），有可能由于网络慢，比如说从你的spout生产出来到最后一个阶段的Bolt，中间可能经历过了很多的一些节点的一些传递，导致了你从前到后你肯定会花时间的。

图3：



那总会有些消息在这个运输的过程中,它这个时间就会被消耗没了，这个消耗到了一定的程度比如它已经花了很长时间仍然没有被处理，也不是指这个消息是不能被处理，也不是说你的这个消息失败了，那只要它超过了这个时间，我就强制的认为它是一个失败的，我就强制的把它干掉，为什么呢？这样的话能够保证你的整个的一个集群的一个性能，从这个节点考虑，如果你觉得大家都觉得自己虽然这个延迟有点大，但是大家都觉得自己是一个仍然可以被执行的这么一个状态，那好吧，老的消息出不去，新的消息就进不来，这样的话就会恶果就会慢慢的体现出来，然后导致你的集群就会出现一些阻塞的这么一个状态。

另外一个很重要的机制（Ack机制），其实Ack是什么东西呢？Ack本质是一个或者多个task，默认的情况下是一个，多个是什么情况呢？你可以通过配置去设置，就像设我们task的个数一样，但是这种task是一种特殊的task，而且非常轻量级，什么叫轻量呢？就是它的数据处理速度是非常块，然后在这个task所需要做的事情不是很多，不需要有太多的计算逻辑在里面，然后它的处理的一些逻辑然后处理的一些方法都是比较简单的，不会占用特别多的时间消耗。

那么它的工作主要是做什么呢？主要的工作是反馈信息和透传这两个工作，那我们接下来看一下这个Ack怎么可以保证你的数据的容错，那么我们先达成一个一致达成一个共识，就是说当你有一个消息出去之后，那你这个消息一旦被成功执行的话，那你就要返回你的ack，就是不管你这个消息是在你的spout阶段还是在你中间经历了很多bolt阶段，那只要是每一个节点都执行成功的话，那你相当于是每一个节点都能够成功的反馈你的ack，那说明你这个消息就是一个成功的任务消息。

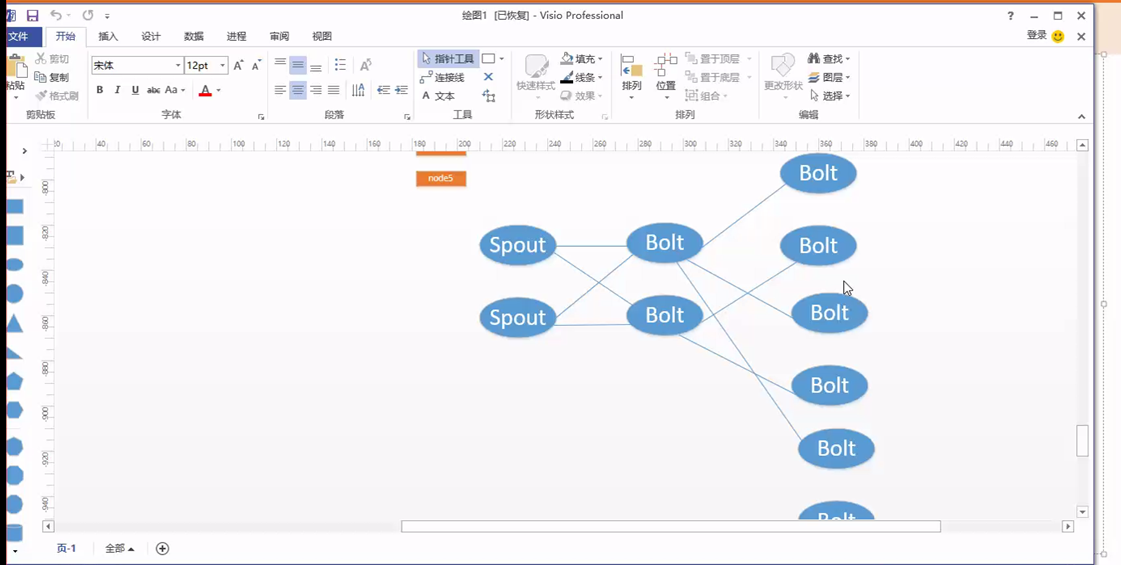
那么我们看一下有什么样的方法可以能够做到这样一个事情呢？（如下图4所示）

图4：



一个叫tuple树这么一个结构，tuple也容易理解把，那我们看一下刚才我们上个笔记上画的一张图（如下图5所示），我们可以通过那么几个简短的代码然后就可以画出这么一个网络拓补图是吧？

图5：



其实你可以想象的出来，如果一个消息它能够被成功处理的话，它相当于是在图5里面也是通过一些图的一些路径然后被处理的，比如说有一个消息从一个Spout出来了，然后接下来这个消息被传到下面的Bolt上去了，然后这个Bolt处理完了之后，这个消息可能被随机的一个Bolt来处理，所以你这个消息中间就经历了三个阶段是吧？如果这三个阶段都能够被成功处理的话，相当于你这个消息就成功了。

如果你其中有一个阶段你ack丢失了，比如一个Bolt消失了，那这个消息 就没有被成功处理。

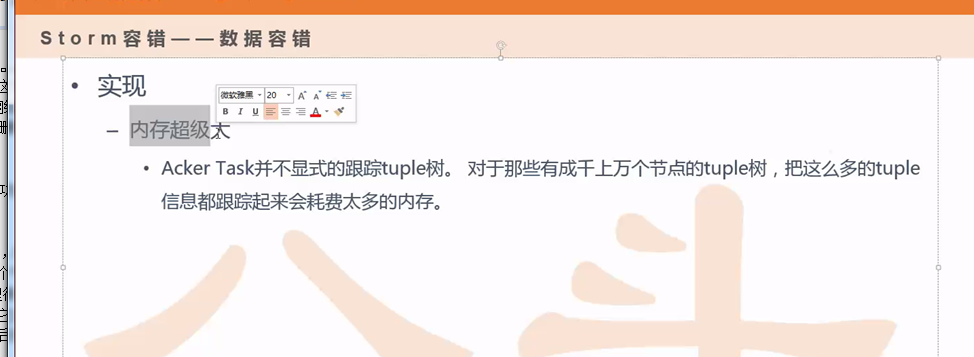
Spout处理完会通知ack，Bolt处理完也会通知ack，最后一个Bolt处理完也会通知ack，如果这所有的ack都 处理完之后，这个ack就会回调你的spout，就是告诉你这个消息被成功处理了，大概是这么一个过程。那也就是说你这个消息从头到尾处理相当于是一个树形结构是吧？

那如果你把所有的树，比如你图4中的A处理完了数据你可以下游是谁你是可以知道的对吧？那比如说下游它衍生了两个一个是B一个是C，那这个时候A已经处理完了是吧？那如果C也处理完了就把这两个处理完之后的节点从这个网络拓补上去删除，那直到把这个所有的树都遍历完之后，那也就是说把这颗树的所有节点都删除掉的话就相当于你这个消息就成功被执行完了。

如果成功执行完了之后，相当于你这个消息就是一个成功的，然后你就可以需要反馈回调你的spout通知你这个消息是成功的对吧？

但是这里面有一个问题，什么问题呢？就是说性能问题，比如说在你这个网络里面，你这个网络节点这么多，你一个消息就对应着很多节点，每一个节点都需要在你内存里面维护起来对吧？那你当前这个系统是可以通过并发很多同时处理很多很多的消息，那每个消息都要维护这么一棵树的话你会发现你在内存是吃不消的，所以它这种情况占用的内存是非常非常大的（如下图6所示），所以这种方法是不太适合的，然后在线上也不是这么实现的

图6：



那它是怎么实现的呢？它是用了非常非常巧妙的方法来去解决这么一个问题，用什么方法呢？就是借助了很简单的异或原理。

我们简单的讲下什么叫异或：

A^B

A=100010

B=000110

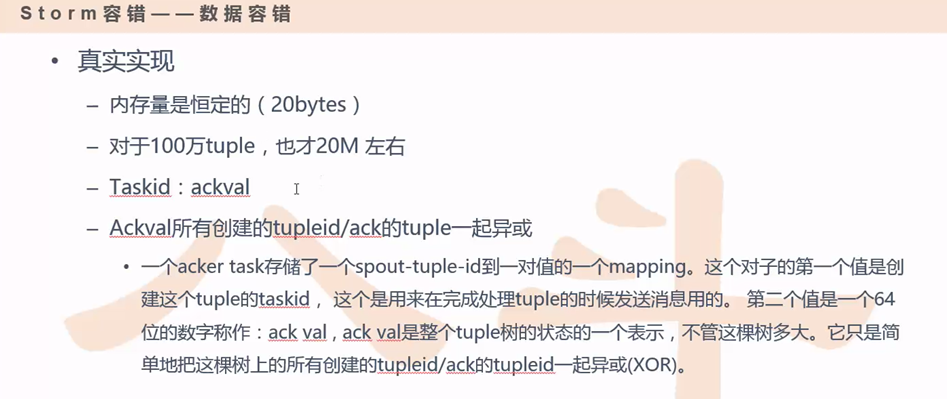
那这个时候A^B=100100

相同位数不一致我就认为是1，如果相同位数一致就用0，那么A^B的结果等于100100

异或这个逻辑能看懂吗？

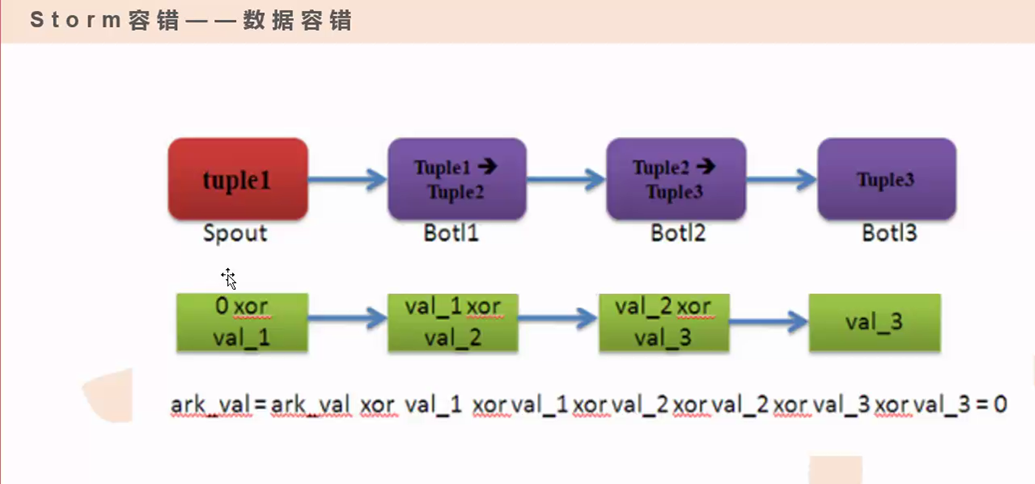
如果我们做这样的一个事情，我们让A异或A等于多少？那相当于每一位都相同是吧？那么结果就是0嘛，那也就是说如果成对出现的信息异或一下那就是0对吧？那借助这么一个特性然后就运用到了这下图7中来了

图7：



然后这里面就是初始的时候就用了一个20个字节，然后来去维护这么一个数据，那相当于用这20个字节来代替了刚才的整颗树，那怎么来通过这20个字节来完成这么一个情况，来完成这么一个消息的校验呢？很容易，比如说我们看（如下图8）

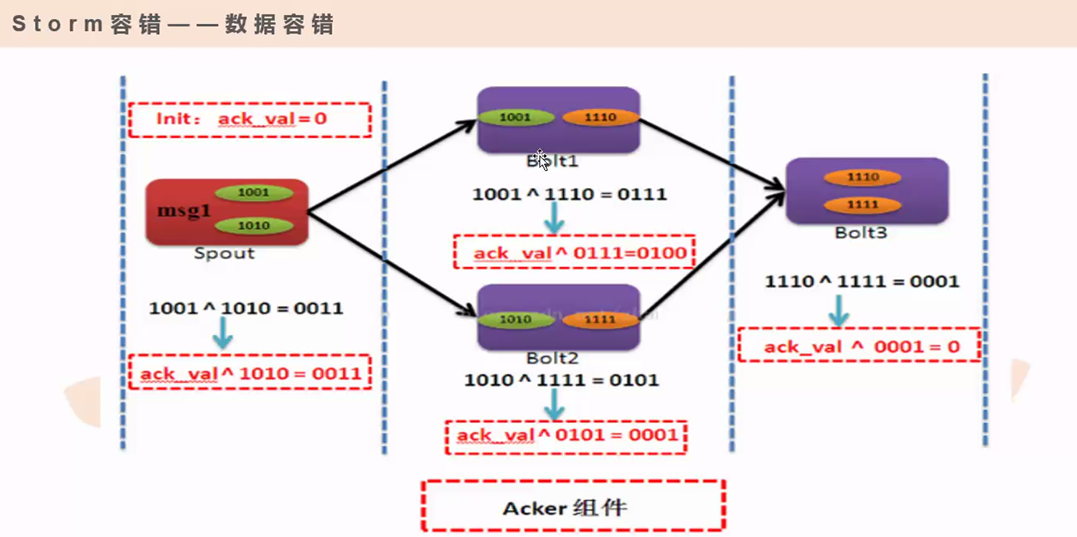
图8：



那我们由一个spout的发出信息通过了三个Botl来处理，那我们看一下如果我们初始化有一个0，这个0作为一个value，那如果一个task发出去消息的时候它就会帮你生成一个id号，那这个value1就代表这个你这个tuple1，你从spout生成的那个tuple，只要你生成了一个tuple它就会有一个tuple的ID，那么这个value1就是你的tuple的ID，你输出和输入都会做一下异或，你输出的时候做一下异或，输入的时候也要进行异或，那这个时候你就可以发现一个很有意思的现象，你的输出和你的输入是成对出现的，只要是中间你有任何一个节点都不失败的话那里面的异或最后的结果肯定是个0，这是一个比较简单的流程。

那么相当于一些复杂的流程其实也同样的一个逻辑，就是说当我一个消息发出去了，但是这个时候比较特殊，我Spout是向两个不同的Bolt去发消息（如下图9所示）

图9：

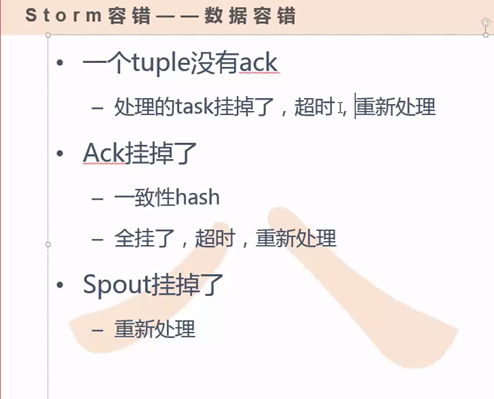


那这个时候也是你的输入和你的输出是成对出现的，不论是从上面的这条线走还是往下面的这条线走，你的输出你的输入都是成对出现的，那也是全局可以保证你如果你这个消息被成功处理的话也是可以保证你全局的一个异或的那个value值也是成对出现的，所以不管是你上面这个图8比较简单的这种方式还是图9比较复杂的网络结构，那你最后异或出来的值都是这个0，所以它可以通过这种异或得到这个结果就可以保证你的消息能够是否被处理的一个结果。

好了，那有人说那到底是谁错了呢？那这个谁错了有意义吗？如果错了怎办？那么就是重来对不对？

好了这块相当于是一个总结（如下图10）

图10：

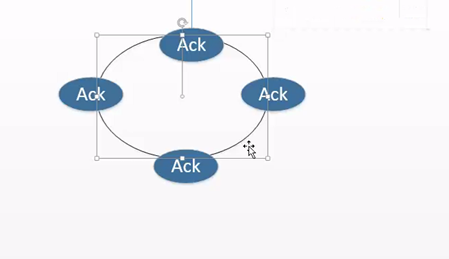


那其中一个tuple没有被ack，那这里面异或呢肯定会把消息丢掉，一旦丢掉的话，最后你异或出来的结果肯定不是0了，那么这就是错了，那么超时也会当作一个错误消息来处理。

还有一个很严重的问题就是你的Ack 刚才我们说过了你这Ack是一个特殊的task，你这个既然是一个task它就是个数字对不对？默认的情况下就给你分配一个task，那既然你这task就类似于一个线程是不是？那既然是个线程它就有可能会失败，有可能会挂掉对吧？那一旦你这个ack挂掉的话，那相当于尽管你这个消息可以从头到尾可以成功的执行，但是你到最后你因为这个没有ack机制保证了,所以了它会把所有消息当做一个失败处理。

那么这个时候怎办呢？那么就可以找预备了，那么这个预备就可以设置多个ack，这时候你就可以通过这样的方式。（如下图11所示）

图11：



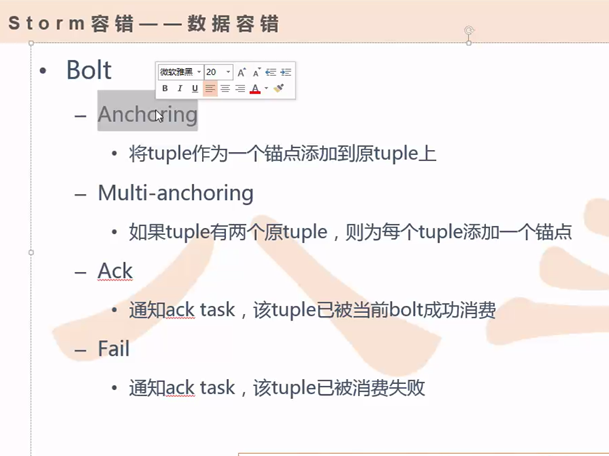
假设说设四个，你发现这个图11中的图像我们之前讲过的一致性hash，如果其中有一个ack挂掉的话，那你就可以找另外一个ack来帮他做这样的一个工作 ，其实在内部实现的时候，这个ack也是通过这样的形式来实现的。

那如果全挂的话，超时，就要重新处理。

那么如果Spout挂掉怎办？那也没办法那只能重新处理。

然后这后面有一个Bolt，如下图12：

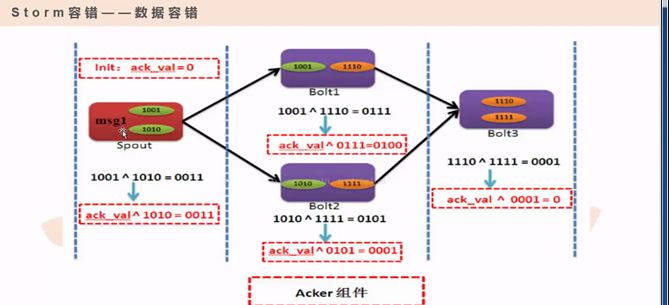
图12：



什么叫Anchcring锚点？什么叫锚点呢？锚点就相当于是它只有产生了一个新的tuple消息的时候才会有一次Anchcring，你认为这个Anchcring相当于就是一个产生新tuple消息的一个情况的描述，但有一个新的tuple产生了就需要把这个锚点添加到原来的这个tuple上，相当于刚才我们讲的树一样，如果有新的消息就在后面进行不断的延伸，延伸一棵树。

你可以有一个锚点，也可以有多个锚点，什么情况会有多个锚点呢？就相当于这个下图13一样

图13：



那我发出去这个消息我可以和你外面的tuple是一对一，我也可以spout同时发出去两个消息是不是？如果发出两个消息的话，就相当于是两个新的消息然后同时锚到你前面那个tuple消息上去，然后这个Bolt你是可以写一些逻辑处理的一些代码，但是你这个代码里面需要有一些异常逻辑判断，当比如说你这个前面发了消息，这个消息里面存的非法字符，那这个消息是不允许被处理的，那这个时候你的Bolt一旦发现这么一个情况存在那就开始通知说你这个是错误消息，直接通知你的那个acktask，说你这个tuple不能被消费，因为它是错误的，直接fail掉，如果你这个消息是可以的，然后并且当前这个tuple这个流程里面也成功的从头到尾处理完了，那这个时候tuple就可以通过写代码的方式然后去告诉ack通知一下说你这个消息由我这个环节是可以成功处理的。

好了那接下来关于整个Storm理论部分就先讲那么多，接下来就开始涉及到了具体的一些实现了。

那我们先看一个很简单的代码（如下图14所示）

图14



这个代码很简单，我们最一开始要解决的就是怎么把Topology进行描述是吧？怎么把代码把这个网络结构描述起来，那你需要描述这个结构的前提你就需要一个对象，这个对象就需要依赖一个TopologyBuilder()这么一个类然后创建一个对象，这个对象可以在这上面去设置你的一些spout设置一些bolt对吧？然后spout里面你可以设置一些你的Spout的名字，前面的第一个字段就代表着你这个原组的名字，这个名字用来干嘛的呢？就相当于给你这个角色的组件起一个名字。

你看这个Bolt它也有这么一个名字，那我怎么能够保证我这个spout和Bolt能够相连呢？就需要把这个名字一致起来，我给spout起了个words这个名字，那我重新建了一个Bolt，那我想和这个Bolt关联怎么关联呢？那我就把你的这个words通过Grouping()的方式进行连接，相当于通过这个名字上下两层就建立了一定的关系。

并且你每一个组件执行的功能是需要开发一些个逻辑的对吧？那这个逻辑就是写一些个类，这个类的实现就需要我们去完成，然后第三个字段已经讲过了就是一个并发度。

那么接下来第三层的bolt想跟第二层的bolt建立一定的关系，那么相当于就把第二层的名字直接通过这个通过Grouping()的方式进行连接，你会发现我第二层的bolt和第三层的bolt其实名字不一样，但是我执行的代码是一样的，那么我们要看我们想要实现什么样的功能，这个功能其实就很简单首先看这个spout这个功能（如下图15）

图15：

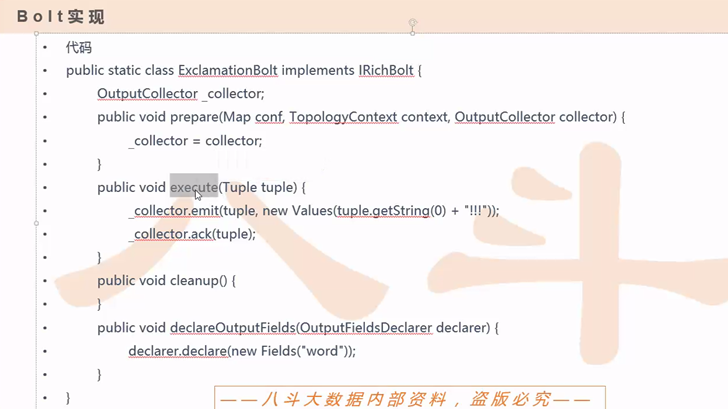


其实你要实现spout最主要的一个函数就是一个nextTuple()这么一个函数，你把这个方法写好了就可以了，当然了还需要写一些其他的方法，有一些字段的一些发布或者是初始化的一些字段，那些个地方的开发就不是特别的多，主要的呢你必须要开发的就是一个nextTuple()这个是核心，那么你读这个代码其实也读的懂，其实一进来的时候先睡一下，先睡100毫秒，为什么要睡100毫秒呢？因为要控制流量，如果我不让它睡，那我一直让他去发信息，因为你spout是干嘛的呢？你spout相当于是水龙头这么一个开关，你这个水龙头开的大水就大，开的小水就小对不对？，如果你把这个水龙头开关直接拔掉，那你这个水直接就以最大的流量冲破了你整个的一个水管对吧？所以需要做个睡眠，其实你可以通过配置的方式把这个睡眠时间动态的调一下，其实是可以实现的。

好了接下来看一下下面有一个数组，这数组里面相当于是一个慈包，这个慈包里面有一些个单词，一共五个单词，然后从这个数组里面随机抽取一个，Random一下，随机抽取一个word的单词，有一个emit，emit是什么意思呢？就是发送的意思，就是把你的消息直接从这个水龙头里面直接传出去，那这个功能很简单。

那么我们再看一下这个bolt的实现功能（如下图16所示）

图16：



Bolt实现核心的逻辑，就是实现在什么地方呢？实现了图16中的execute这个函数，这个函数也是一定要去开发的，那在这里面开发的比较简单，都开发了什么内容呢？那么这个execute（）括号里面的Tuple是你的函数输入，也就是你图15中的spout输出就是那个emit你发出去这个单词通过values进行封装，那么它就是一个Tuple，那么在图16中的execute（）接到了你的Tuple的信息，然后对这个信息进行一个解析对不对？，解析完了之后后面追加了一个三个小叹号，就相当于在你原来图15单词这个背后加了三个小叹号。

由于我在图15中有两层Bolt，这两层Bolt的方法用的是一样的，所以最后的一个结果符合预期的结果是这个样子，那我spout发射单词，然后在每一个Bolt后面都要追加一个三个叹号（！！！），三个节点连成一条线，我spout把消息发送给了首个Bolt，然后这个Bolt就发射给了另外一个Bolt。那也就是说我比如发射元组是一个bob和这个john，然后我第一个这个bolt将发射你的bob后面加三个叹号，然后我第二个bolt也就是第三层发出去的是bob再追加三个叹号相当于总共是6个叹号是不是？所以这个代码逻辑大家能看懂吗？如下图17所示

图17：



这个相当于是最简单的一个流程把

然后这下两张图里面的话都是一些关于你去实现代码的一些个说明把，大家看一眼就可以了（如下图18和图19所示）

图18：

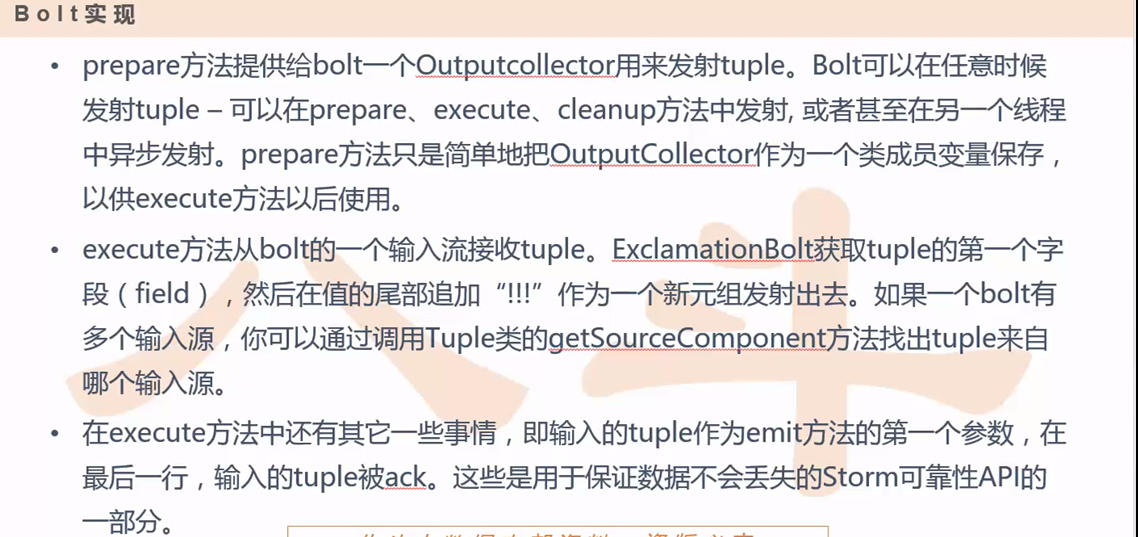
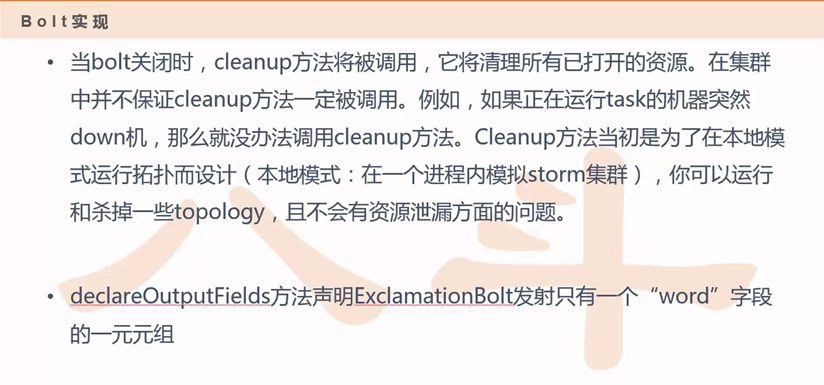


图19：



好了然后这边说一下本地模式（如下图20所示）

图20：



什么是本地模式呢？之前我们在上次笔记所讲的storm有一个非常好的一个特点，它特点是有一个本地模式，是用来做本地调试用的，如果你本地代码都跑不过，那么你这个代码在集群上一样也是跑不了的，如果你本地能跑的通，有可能在集群上是跑的通的是这样的一个情况，所以建议大家在去开发的时候首先先要在你的本地先调试一下。

那么本地调试也很简单，需要通过图20里面的LocalCluster 这么一个代码来完成，然后它产生了一个对象(cluster)，通过这个对象去提交你的任务(submitTopology)就可以了

然后setDebug（true）是什么意思呢？相当于就是把一些更多的信息然后输出，就相当于它系统里面有一个日志的一个级别，然后把你的Debug消息打开的话相当于有一些本来是可以屏蔽的消息就是更多的能够给你打出来，然后让你去分析你的问题。

然后setNumWorker（2）就是控制你的进程的个数。

然后我们来看一下常用配置（如下图21所示）

图21：



虽然这个配置只列了四个，其实真正运行一个任务的时候这个配置还很多的，大部分的话其实你是不需要配置的。

首先第一个Config.TOPOLOGY\_WORKERS:这个是一个配置，那么这个配置其实跟setNumWorker（2）是一样的，只不过是如果说你不写setNumWorker（2）这句话，那么Config.TOPOLOGY\_WORKERS就会从全局默认这个数字来去继承一个，就是你可以在代码里面可以不用去指定你的进程的个数，那么它本来这个环境里面是可以帮你配置一个的。

还有一个TOPOLOGY\_ACKERS，这大家能看懂是什么意思把？可以设置ACK的个数是吧？

还有一个TOPOLOGY\_MAX\_SPOUT\_PENDING，这个很重要就是最大（MAX）的spout，PENDING，这个PENDING是什么意思呢？就类似于像停留一样暂停一样，处于这种还没有被完全处理处于这种等待的这么一个状态，什么意思呢？当你这个Storm管子里面最多最多停留这么多个MAX\_SPOUT\_PENDING消息，如果你没有这个参数控制的话,它是不能够保证你这个管道，有可能你这个管道可能会被消息阻塞，然后开始爆掉,它相当于也是可以通过这样的方式来控制你的消息的一个个数，就保证了你系统的性能。

另外一个就是我们刚才说过了可以通过一个超时时间TOPOLOGY\_MESSAGE\_TIMEOUT\_SECS，这个超时时间就是当一个消息从你的spout出来到最后，就是它还没有完成的时候，它已经延迟时间已经超出这些个个数的话，那么就认为这个消息就失败了。

所以可以通过TOPOLOGY\_MAX\_SPOUT\_PENDING和TOPOLOGY\_MESSAGE\_TIMEOUT\_SECS这两个参数能够控制你整个管道的一个消息的一个拥塞程度把，就是尽可能的保证你的Storm这个集群内部稳定。

然后接下来就看一下这个环境怎么搭建了，这个Storm环境搭建还是相对来说也不是特别复杂，主要是面对着要修改的文件就这么一个。

那么接下来如何搭建Storm在这里就不演示了哈！！