

今天来讲新的内容，今天的内容比较偏理论和概念，就是比如一些细节怎么实现，然后整个的集群的架构它是怎么运转起来的，那咋天讲的那个HDFS也是这方面，主要是看一下他那个架构是怎么样主和从之间相互关联的。然后今天讲Storm，然后Storm里面其实它又是一个全新的架构，但是这架构的话你可以看到一些之前我们在去学MapReduce或者是HDFS的一些影子，有一些很相似，但是有一些呢完全不一样，就是相当于你通过Storm的方式可以了解一下这个架构跟你之前学的架构有什么区别并且像之后我们学那个HDFS2.0，虽然这个HDFS2.0或者是Hadoop2.0虽然它这个架构跟那个Storm架构应该是没有什么关系的，但是他们之间也是有一定的交集，所以提前把Storm放这块讲一个目的也是作为一个中间的过渡这么一个状态，为什么呢？因为Storm这个集群里面用到了一个zookeeper，然后在hadoop1.0里面是没有这个zookeeper的模块的，然后在Storm里面它就有了这个zookeeper，并且在hadoop2.0中相比于hadoop1.0升级的这一块也是依赖了这个东西然后做了很多的一些事情，所以我觉得这个Storm把这个聚群架构能够了解透的话呢应该在去对之后再去学hadoop2.0还是有一些那个铺垫作用的，所以我们把这个实时计算这块提前就讲了，然后可能一开始把这个批量计算部分和实时计算部分安排了所有的整个课程体系安排在了前面，大家会觉得一开始上就学这些比较重量级的一些概念可能会压力有点大，但是你会发现从之前学的mapreduce和今天学的Storm之间你会体会一下这两者之间的一个差异和对比，然后两个不同的架构再去处理分别各自持久性领域到底有什么样的差异性，就是大家带着这个问题去学习，时刻的提醒自己我们再去学Storm的过程中你去想一想在mapreduce里面为什么不能实现，然后并且哪一块就是mapreduce可以借鉴过来，当然了在完成一个整体项目架构的过程中，通常是实时计算和批量计算是两者都会存在的，一般公司来说做项目呢，两个计算框架两个计算模式都是共存的，那相当于两者是相互依赖然后是相互补给这么一个过程，所以也带着这样的一个思考怎么样的互相之间扶持着然后把整个的项目支撑起来的，是这么一个情况。

然后在讲Storm之前，有必要把那个我们第1次上课就是这个开学典礼那次课有一个图，不知道大家还记得吗？（如下图1所示）

图1：



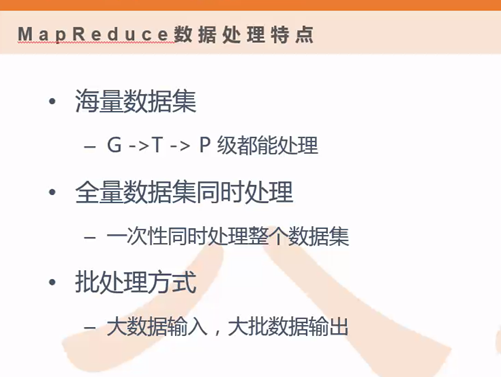
就是一个整个的一个项目架构图,那你再回来看这个图里面的时候就会发现有一部分大家已经搞的比较熟悉的了，大家熟悉的就是离线数据挖掘那一块对吧？目前大家可以说是把离线数据挖掘这块坑不能说是填满把，起码是填了一半了这是有的，那另外一半需要我们后面再继续填充，所以我们现在再保留一部分然后我们再看上面的在线数据挖掘就是Storm那一块，所以整个流程的话你会发现HDFS这块也相当于收集了一些把，然后后面还会再继续，然后Mapreduce这一块我们之前学过一些基础，基础的应该是OK的了，然后接下来就是做一些个升级和高级一点的结合业务问题去开发相应的功能，然后Spark这一块虽然没开始，但是两者也有一个相似的地方，所以我讲的时候有可能再去讲介绍业务的时候两者都会捎带着，所以这个情况对于目前我们今天的是讲在线数据挖掘（Storm）这一块知识部分，我们之前学学的是离线部分，大家要搞清楚我们之前学的这些知识是为整个的推荐项目体系里面是服务哪一块的，这个大家思路要搞清楚。

图2：



然后整个今天的内容，Storm一天是讲不完，所以今天的重点是在理论部分，然后理论部分也不一定能够讲完，大概我们今天讲Storm基础和Storm架构这两节是要讲完的（如上图2所示），然后后面的话安排到下次课去讲，大概是这么一个安排。

图3：



那回过头来和MapReduce之前我们已经学过了，然后这里面在做一个复习，那MapReduce能干什么呢？（如上图3所示）。

它能干处理海量数据，能处理大数据是不是？那具体怎么做呢？那通过之前的实践大家也应该也熟悉了些操作过程对不对？然后你会发现因为MapReduce它也是个集群模式的，然后它处理的数据的能力是和它的规模有关系，比如说咱们要用自己的笔记本不可能处理特别特别大的数据，那只能去处理个百万级的数据还是没有问题的，因为这是之前我们实践过的哈，但是你如果想处理一个上G这样一个大小的数据是有点困难的，所以对于公司来说那对MapReduce集群这块也是会耗费相应的这个时间经历去做这一块。

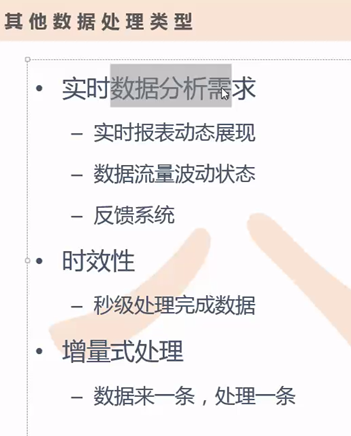
另外一个呢之前一直说MapReduce是一个批处理的这么一个计算模式，那什么是批处理的计算模式呢？就是全量数据集，也就是说比如说你要跑MapReduce的前提是你这个input的这个输入的数据一定要是给到完整的数据，那比如说我就想给你一部分数据，那好了，那MapReduce它只会跑你给的这一部分数据去最后得到一个结果，那比如说它所需要的这个输入的路径，里面这个数据它是不断的去产生的，但是你如果把这个当做一个输入数据源给MapReduce的时候呢，那不好意思MapReduce只会在启动的那一刻它确定当前有哪些的输入就是哪些输入的，然后只要它这个MapReduce一旦运行起来那你这个输入里面再有有一些个数据更新，比如说是追加的一些数据的话呢这个MapReduce是不会把它容纳进来，因为它第一步是要做一个split对不对？它要做的是一个数据分组，然后如果你要是新的数据它肯定是不会去做数据分组这么一个步骤，所以你这个要是给这个MapReduce的输入数据必须是一个一次性的给入，然后批量的方式去进行处理，那很简单的理解，这个MapReduce就是一个集成计算框架。

那大数据的输入，也就是大数据的输出。就是说它的吞吐能力非常强，你给他大量的数据，那它就会给你产生大量的数据，但是具体这个规模需要你自己的代码去控制，你的这个实际的逻辑去控制这个规模。

然后相对而言MapReduce它已经有了这么一些特点，但是它还有一些其他的特点，这里面先就不说了，那对于这上面图3所讲的几个数据的特点我们再看一下我们在去做大数据开发的时候其实也是可以胜任的了，但是你仅仅具备这些能力的话具体这些特点的话你是还完全不能够应付所有的大数据场景，那之前我们说了像这种实时进行要求比较高的场景用MapReduce来说就有点困难了。

那什么样是实时的一些场景呢？（如下图4所示）

图4：



比如说一些实时的一些数据分析需求，那举个例子肯定你们其中有人在炒股票是吧？然后呢这个大盘它随着每一秒钟或者是每几秒钟把这么一个单位或者是分钟级的单位，然后它的这个走势呢是不断的波动，实时变化的，有可能是下降的也有可能是上升，你也不可能去猜测下一秒的状态，那当然这个的一个图表其实很好理解，那相当于这样的一个显示情况可以在很多的业务场景下都需要用，比如说你的公司老板然后想知道他现在这个公司每时每刻的一个收入情况，然后他又想我要打开这个网站的话那你就不仅要告诉我之前赚了多少钱还像知道今天的比如说上一个小时或者是前几分钟这个公司到底是盈利还是亏损大概有这么一个虚视图，对于这种虚视图其实就是一些实时的一些数据处理，然后它能够实时的表达你当前的一些波动状态。

当然还有一些监控，比如说你对一个服务器的能力的一个监控，它是对于这种监控也是这种实时的，对于CPU的这种压力情况然后内存的使用率其实也有这样一个实时的一个采集和实时监控这么一个需求。

当然除了这样的一个情况还有一些反馈系统，这个反馈系统虽然是这个实时计算这个框架其实可以做，但是呢这种场景也不是特别的多，什么场景会用反馈系统呢？一般是在这个机器学习的场景下会用到，当然如果你要是大量用这个机器学习的话应该很少用Storm去做一个机器学习把。我了解到的之前腾讯是花了很大的经历去搞Storm，用Storm来做一个实时的一个模型，但现在不太清楚了。

那什么叫反馈系统呢？其实反馈系统也比较简单，我们说非反馈就是我们传统的一个系统，就是你的输入进来了，然后中间比如你的MapReduce中间这个一个集成框架处理完之后数据输出，就相当于是一个羊肉串，然后这个钳子插进来羊肉里面然后出去了，那这个时候呢反馈系统什么意思呢？就是它的输出马上反馈到的它的输入，明白这个意思把？就是它的输出它的信息之间会知道它的一些输入的一些变化，所以对于这种机器学习模型很多这种集群都需要这种不断的去迭代，什么是迭代呢？就是它会不断的循环好多轮，然后最后达到一个比较好的一个状态然后它会最后去输出这个数据或者是模型然后供限量使用，然后在这个真正得到有效的输出之前，它是不断的在迭代或自我优化的这么一个过程中，它优化是怎么优化的呢？就是通过你的输出来指导你的输入，所以呢如果把输出然后反馈到前端输入的话呢它就是一个反馈系统，可以这么理解。

然后对于时效性来说，那通常是你想它的时效性你在怎么做时效，它不可能是很完美的时效是不是？我这边有什么数据变化了然后你那边相当于完全完全同步至这是不可能的，所以现在对于一个系统来说做到秒级的这种时效性也很不错了，你想一下毕竟现在我们要实现一个功能都是通过集群的方式去实现是不是？然后集群里面有很多的机器对不对？那机器之间的数据传输它都是需要时间的，你时效性再怎么高，那你这个传输时间你这个数据处理肯定是要浪费时间的，所以你这个单位也就是秒级一般来说对于这种大公司通常的业务它的需求只要秒级就可以了，不管是做推荐也好，还是做一些个数据报表也好这都够用了。那其实呢它能够涵盖大部分的一个需求，但是Storm它虽然说是Storm这种系统它其实要比秒还要细粒度，它可以做到毫秒级的，这个大家记住。

那另外一个增量式处理，增量式处理完全是对标前面图3这个MapReduce，那MapReduce它只能是做批处理的方式，就是你的数据一次性的给入，那Storm这种实时计算模型呢就不是了，它可以是增量式处理，那什么是增量式处理呢？就是它无时无刻的在处理数据，你有新的数据过来，我就是马上去处理然后把数据处理结果进行产生，你可以把它想象成一个生产线里面的一个传送带，然后传送带上面比如说有很多的包装箱或者是一些个罐头一些牛肉等等这些产品，你可以把这些东西模拟成当作数据，然后传送带你就认为是一个实时计算框架，它能保证你这个上面的数据能够从某一个特定位置发到另外一个位置，所以暂时的把这个实时系统想象成传送带，这是比较好理解的。

图5：

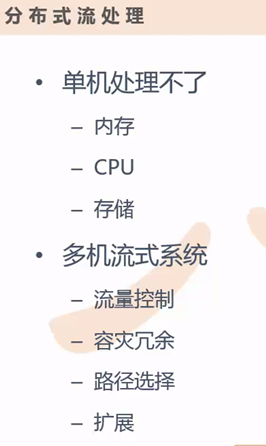


然后呢流式处理（如上图5所示）是时效性高，要对比图3的MapReduce来说它时效性是很高的，那这个好理解，来了一个数据我就处理一个数据，那这个数据根本就不用等，不像MapReduce一样，MapReduce你想跑一个任务大家都能体会，那最慢最慢是在分钟级把是不是？你像这种企业里面有很多的MapReduce，两三天都不一定能跑出数据来，所以中间这个时间呢你必须得去忍，那像有一些数据你忍不了，我这个数据来了，必须马上处理，那MapReduce这个框架是搞不定的，所以用流式处理。

然后流失处理呢是逐条处理数据，就是增量式处理，刚才讲过了。低延迟好理解，延迟低也就是说对应的时效性高，它耗的时间少。另外一个就是大家觉得这个流式处理感觉是一个很高大上的一个东西，流式流式总感觉这东西是一个很神秘的东西，其实它的本质是一个很好理解的一个概念，刚才我们举了一个传送带的这么一个概念，那你现在通过一个我们之前学过的linux的一些命令也可以表达这样的一个思想。

像图5中最下面的那个linux命令，我觉得大家实践了这么多次了，那多多少少这个linux命令大家应该都没问题了是吧？那我像图5中最下面的那个linux大家应该都能看懂，或者我们之前讲过什么？你本地调试嘛，本地调试是怎么调试的？那就是: cat input | python map.py | sort | python reduce.py > output，这条语句恐怕大家更加熟悉了对吧？那么其实图5中的那条语句也是没有什么问题。那相当于这条 cat input | python map.py | sort | python reduce.py > output的语句前面的处理的标准输出通过管道的形式传输到了下一个环节（cat input | python map.py ），然后作为下一个环节的标准输入，那这样的话相当于前后两个完全独立的环节就是被这个管道建立一些联系了是不是？那你可以把这个图5中的语句想象成生产车间，比如说input 就是相当于是把这些做一些个预处理，比如说这个鸡肉罐头的这个生产线，第一个阶段肯定是要把这个鸡给宰了（cat input ），然后第二部分呢要给它高温消毒（ python map.py ），然后第三部分就开始肢解等等，就是相当于中间有很多的环节，你可以认为这个管道跟管道之间就是一个环节，你就可以认为是生产线的一个环节，那相当于这个概念就是一个流式处理的一个概念，从前到后处理很好理解。

图6：



然后通常做分布式的一些流处理的，先不考虑分布式的情况下，那单机情况其实也是可以处理一些流的一些情况，比如说我们之前举过一个例子，就是那个当时学了一个哈希hash方法还记得吗？全流量的一个PV请求，然后呢后面有很多的服务器，通过哈希的方法去打到不同的服务器上去是不是？那如果当前一个请求新浪的这么一个用户请求，然后打到了某一个服务器，那它的请求必须要在这个服务器内部做一些响应对不对？，那在内部的话它肯定要经过很多的一些环节，首先你得解析它的请求，然后另外一个它的请求真正的一些信息做解析，解析完然后去做一些个业务处理，然后再做一些数据包装再产生，其实你可以认为这整个流程也是一个流这么一个处理，就像我们看完代码一样，比如说我们写这个C语言或者是c++或者是写java，你都是从一个main函数开始的对不对？，从main函数开始的话那就是从上往下代码逐步执行，也是从头到后的一个形式，但是单机的形式是不能够扛得住太大的压力对不对？那它通常是受限了一些硬件的一些原因，比如内存不足，CPU不够用，存储空间有限，所以我们用多个机器然后突破这一些单机硬件的一些天然的一些限制把，然后把这种限制结束掉，然后服务更多的流量是这样的一个情况。

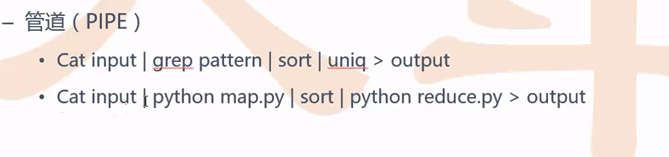
所以你想搞多机的话就不容易了，单机的情况下你很容易，搞一个脚本然后从上到下直接运行，但是你多个机的话因为每台机器，比如说每台机器都是一个独立的机器，那这个机器跟机器之间你得协调好，那机器的每一个角色你都得定位好，互相之间一个协同的一个工作，比如第一个流量控制，这个流量控制肯定是这个每一个机器它的处理能力是不一样的，你尽管每一个机器的配置是一样的，但是呢真正的实际的过程中，你每一个机器它最后能够服务的这种还是有一定的差异性的，有的机器它处理数据相对多一点，有的机器机器它处理数据相对少一点，那到时候如果来了一个新的流量，那我尽可能挑那些比较空闲的机器来对应服务，所以这是一个流量控制。

另外一个当我们这个流量分配到你某一个机器的集群节点上，但是很不幸你这个节点突然挂了，那这个请求已经给你了，但是你挂掉了那就相当于你就不能够响应了，那这个时候相当于这个流量就被浪费掉了，那既然被浪费掉了，那按正常的传统的处理方式最简单的处理方式那没办法，只能把这个请求丢掉，那如果要是这个流量跟这个用户体验或者直接跟这个钱打交道的话那你相当于你这个公司损失就大了是不是？所以这是肯定是不能够接受的，所以应该怎么样对这个流量再做一个备份，就是怎么把这个损失进行一个拟补。

然后容灾冗余这个好理解就是比如说你某台机器挂了，然后怎么样去把它迁移到其他机器上去，跟刚才我们说的很相似，还有一个就是如果这个机器这个集群规模已经扛不住当前压力了，那想给它加一些机器是怎么能够动态的扩展这是一个。

路径选择，因为像我们之前讲的那个哈希的那个例子很简单，我一个流量请求做哈希，然后分配到不同的机器上，每一个机器你给任何一个机器去请求每一个机器它的角色都是一样的，你给任何一个机器去请求的话它给你返回的新浪页面都是一模一样的，所以之前的角色没有任何差异，但是真正在一些复杂的大数据的一个处理的一个业务的情况下，每一个机器它的角色是不一样的，比方说我们刚才那个例子（如下图7所示）

图7：



就相当于那个生产线，比如这个生产线上一共有100台机器，其中有10台做第一个环节，有另外10台机器做第二个环节以此类推，相当于你会发现在这个同一个环节里每一个机器的角色是一样的，但是在不同的环节里面，这个机器角色是不一样的，起码你能够感受到这个不同机器的机器之间是有一定的依赖关系，所以把这个依赖关系角色要定位清楚，就不能够说你来了一个流量，我接下来把这个流量我不发给你第二个环节，我直接跳过你发给第三个环节这是不允许的。

那么扩展这个刚才已经说过了，就不说了哈。

图8：

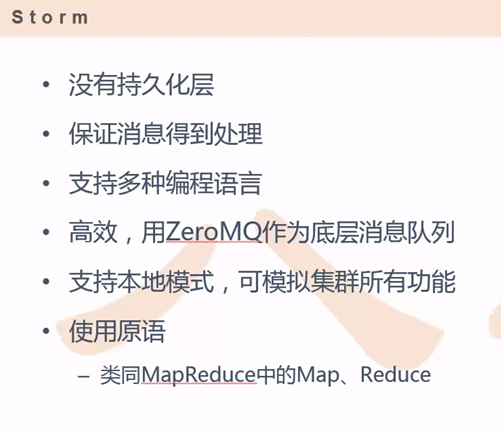


所以对于流式计算这样的一个系统，它其实有很多的实现方案，那其中Storm是一种实现方案而已，后期跟storm有一拼的就是spark streaming，这个spark streaming也是搞流式计算的，但是这两种还是有一定的差异的，如果要是这个业务需求是在分钟级别或者是这个秒级两者都可以，如果你这个业务需求就要求毫秒级别，那就只能是Storm，为什么呢？因为spark 它还是一种小步慢跑这么一个形式，就是它也不是那种真正的流式计算，它是把一小堆一小堆的数据快速的处理，它不像这个Storm一样来了一个数据就处理一个数据。

所以storm是一个开源的分布式实时计算系统，是由Twitter出品的，是一个比较成熟的系统，可以在Github上能找到开源，然后应用比较广泛。

很多公司在去搞这个实时计算框架的时候，都会去选择storm来去对接，那主要一个原因就是比较文档，并且容错性也比较高，然后实现起来也比较方便，为什么方便呢？

图9：



其实流式计算这样的框架有很多，storm只是其中的一个，那为何要选择storm呢？

因为它好用，哪里好用呢？我们看一下storm它的特点（如上图9所示）。

1. 没有持久化层，刚看到这句话的时候大家可能不太理解什么意思，所以我们拆开来看，什么叫做持久化？这个就好比我们之前讲的MapReduce从内存把数据往磁盘上转储这么一个逻辑明白了把？你可以这么理解，就是说如果你的数据在内存里面，那你这个机器挂掉了，好吧，那你这内存里面的数据全都丢失，如果你这个内存里面的数据能够定期的往磁盘上去写，那你这个机器挂了内存数据丢了没关系，你磁盘上还有是不是？那这个时候这个数据叫做持久化，把你的数据从你的内存持久化到你的磁盘上。那持久化层是什么意思呢？那就是一个存储，就是做数据存储的。
2. 那没有持久化层那就是说storm根本就不存储，所以这个东西跟MapReduce完全完全不一样的，MapReduce大家都很了解很清楚了，MapReduce它中间要经过了很多次这个写磁盘的操作是不是？从你的内存比如Map阶段就从你的内存不断的往你的磁盘上去写一些小的文件，reduce里面也有内存，然后内存也会往磁盘上去写，那最终MapReduce和reduce之间当然他们的传输也是通过远程拷贝磁盘上的文件完成这么一个partition对不对？然后这个reduce数据产生之后这个数据也是以这种文件的形式去存在HDFS的，所以这个MapReduce是非常依赖于HDFS的。

但是这个storm就不一样了，storm只要是你的数据一进来到你的数据输出和storm中间数据不会往磁盘上去存，也就是说所有的数据全部都是在内存中进行流转，那这也就是为什么

storm它的实效性高的原因，因为它是不存储到磁盘上，写磁盘是比较慢的一个过程，所以storm是不写磁盘的，完全是通过内存进行一个处理和读写。

对于storm来说没有什么TB不TB，这个它的数据像一个水管一样，你说你这个数据是多大规模的不好说，只能说你这个当前的这个数据的这个量有多大，就是成批每秒钟这个数据量有多大主要是这个样子，那你真的要是想成批这个TB，那你只能通过MapReduce，所以要是一秒钟来个TB的话那这个storm也扛不住是不是？那么这样水管就会爆了哈哈。

那保证消息得到处理是什么意思？这个很好理解，我们在举个主管跟下属来举个例子，比如说主管让这个下属去做一个工作，然后这个下属也答应做了，但是这个下属不管他做完不做完，他都不告诉你这个进度，可能这个下属可能性格比较内向不爱说话，最后他不管是自己有没有处理完，他都不会主动跟老板主动报告，所以这个老板也不知道他到底有没有完成，所以消息就是个老板让这个下属去做的这个工作或者是处理这个数据这个消息，那保证消息得到处理就是说老板要求你做的事情，你必须要及时的要反馈回来，不管你是有没有能力去做完这件事，如果你做完这件事的话，那你应该把做完的成功的这个结果要反馈给老板，如果你可能比较忙，然后又没有能力去完成这个工作你也得及时的跟老板说，这时候老板也会找另外一个人，这样的话才能继续把这个工作给别人完成。

所以相当于消息的一个确认的过程，所以有了这么一个机制就保证了你的storm在去处理数据的时候是可靠的，起码你的数据发出去，你也得知道这个数据有没有成功的处理是不是？

如果不知道的话那这相当于是一个你可以把他比喻成TCP/UDP服务是不是？UDP是不管你这个反馈，它只要发出去了，TCP呢就有三次握手。

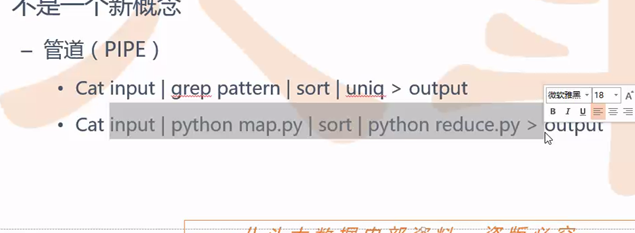
然后支持多种编程语言，这个更好理解了对不对呀？这个跟那个MapReduce，Hadoop这么一个思想一脉相承，你可以用多种语言去开发storm，那为什么呢？其实这个storm也是用java开发的，但是它的这种支持多种编程语言的方式跟MapReduce就不太一样了，MapReduce当时说它能够支持多种语言的原因是因为它做了一个包装。

那storm支持多种语言是因为它有一个thrift rpc，那这种thrift rpc协议是对语言来说是透明的，你可以用java去开发storm，你也可以用python去开发storm，通常都是这两种语言去做storm开发。

高效的话呢它是基于一个比较成熟的底层的一个消息队列把，那什么是消息队列呢？消息队列也就是一个保证你的网络传输的一个队列，连接你的客户端和服务端，就是你想把一个数据通过网络从一个节点往另外一个节点做传输，所以需要通过这么一个远程的网络的消息队列，那这种消息队列其实有很多，早期的时候是用这个ZeroMO，现在都改成netty了，这两个有什么区别呢？这个ZeroMO是通过c++开发的，然后netty是通过java开发的，这两个东西都是一个消息队列，你就认为是一个网络开发一个中间组件，然后你可以在基于这个上面去开发你的客户端开发你的服务器，然后中间这个数据具体怎么传输，它都帮你给封装好的，所以使用的时候是很方便的，因为他帮你优化的也是比较好的，如果你自己开发的话完全是没必要，重复造轮子，而且效果不一定有它的好。

有一个非常非常重要的特点，那就是支持本地模式，什么是本地模式？我们之前在讲MapReduce的时候（就比如如下图10）的语句，做一个本地的代码调试对不对？

图10：



那我们说了之前把这个代码放到本地你先跑通了你再放入集群上，就是如果你本地代码你都跑不通，那么肯定在集群上也一样跑不通，本地模式就是说你可以开发完storm的代码你在你本地先测试一下，如果你本地能够测试成功，那么集群上就有可能跑通，如果是本地都跑不通，那集群上肯定跑不通，就是在本地的方式来模拟集群上所有的功能，这是很有用的，因为去开发storm的时候，其实这个开发难度要比mapReduce要高很多，像开发mapReduce你肯定调试比较多，可能是刚刚接触然后经验不是很多经常遇到一些问题，但这些问题都不是太大的问题，但是storm的开发很容易出现问题，所以你很难是一次性把你的代码开发成功，所以通常都是要在你本地要测试很多很多遍，你才能把你的代码上传到集群里面去，所以开发难度来说呢storm要比这个MapReduce要难很多。

同样呢，我说这个难可能是比较复杂一点就是需要大家开发的时候可能心要细一点，在开发storm的很容易出错，所以心要细一点，但是为了便于大家开发，其实他已经封装了很多的一些个原语，其实就跟MapReduce的Map和Reduce很类似，就相当于在MapReduce里面你需要把Map函数和Reduce函数你写完就可以了，但是在storm里面类似，它也有像Map和Reduce这么一个函数，你把这个函数你写不写把，代码对不对把就出了这个功能，按照你这个想法，然后整体的框架你完全不需要介入，你只需要把它的那个原语那一块的函数维护好就可以了，所以开发的时候也是减少了很多的一些效率，也提高很多的效率。

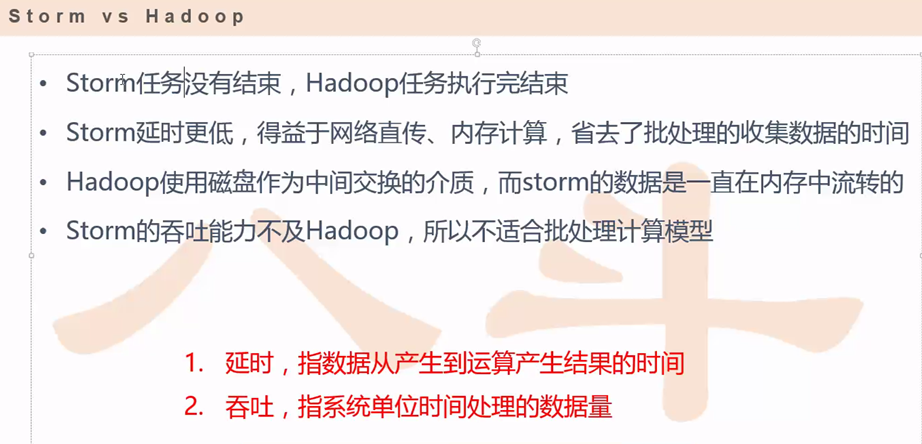
图11：



然后上图11也比较好理解对不对？同样是电梯，那一个是竖体，一个是斜体

如果对于之前的MapReduce的处理，那你可以把他比喻成竖体是吧？那你这个电梯一旦输入就是一旦人进去了一关门，好了，如果其他人想进电梯的话，那不好意思你只能等下一部电梯，然后它可以一次性的容纳很多很多人，相当于它能够比喻成你的MapReduce的吞吐能力，显示它批量的处理。而storm就不一样了，storm相当于是我这个数据源源不断的进来，然后数据源源不断的处理，然后直到产生，并且你看这个斜体它是不断的去滚动的，不像这个竖体它是一会停一会启动一会停，它这个滚梯是不断的运行的，但是有一个特点就是说hadoop就是吞吐能力比较强，而Storm呢你会发现这个并排你就是一个人两个是不是？它站不开，所以你这个数据不能这个压力给它过猛，你得像这个水流一样慢慢的喂给他，然后每一个小水流每一秒钟你给的这个数据不要太多，这样的话它才能健康的运转，那就是两者之前的区别。

图12：



那有一个特点就是说Storm这个任务只要一旦启动就没有结束，明白了把？就不像我们开发MapReduce，MapReduce启动完之后他处理完，告诉你一个输出路径然后结束，但是Storm这一块你一旦启动就没有结束，除非你手动把它的进程给杀死，除非这样。

那为什么没有结束呢？其实很好理解，就好像图11右边这个扶梯一样，那你要是这个扶梯不拔掉这个电源，它这个扶梯就不断的去滚动对不对？

然后Storm它延迟更低，这个说过了，因为这个内存计算，省去了一些批处理的那个数据落地。

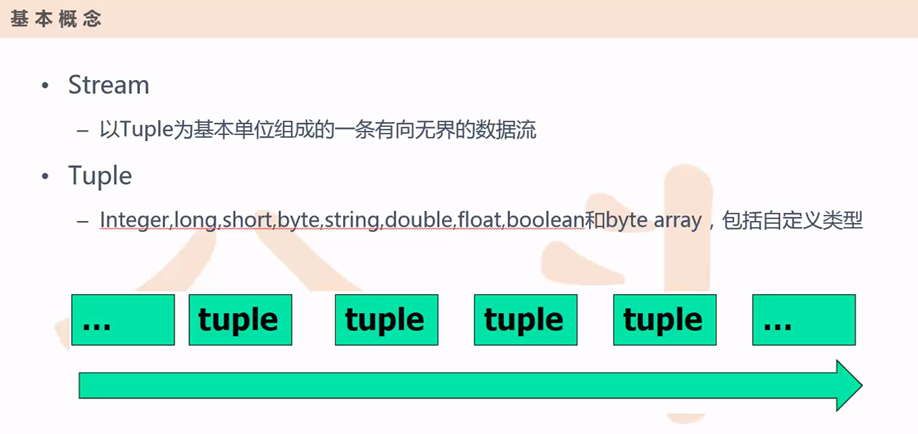
然后Hadoop因为通常使用磁盘作为一个数据交换的媒介，所以这个磁盘经常会有一些数据的交换，导致了它的实效性不高，但是吞吐能力比较好，Storm就一直在内存中进行流转。

然后Storm吞吐能力来说远远是不及hadoop的，为什么呢？因为这个跟你之前的这个HDFS大数据有关系，和它的block也有关系，这个上节课刚刚说过，这个block为什么涉及的比较大，因为它这个数据处理的时候，它是从头往后读，那只要是顺序读的话呢那他的结合着操作系统相关的优化它处理数据是非常非常快的，那所以对于大数据批处理方式这种，Storm是不适合去搞的，Storm也能搞，但是需要一定的时间，那具体的这个我会在后面结合相关的案例跟大家说。

延迟，吞吐这是两个特点，这是两个之间的对比，这个就不说了。

然后接下来我们就看一下整个Storm的基本概念（如下图13所示）

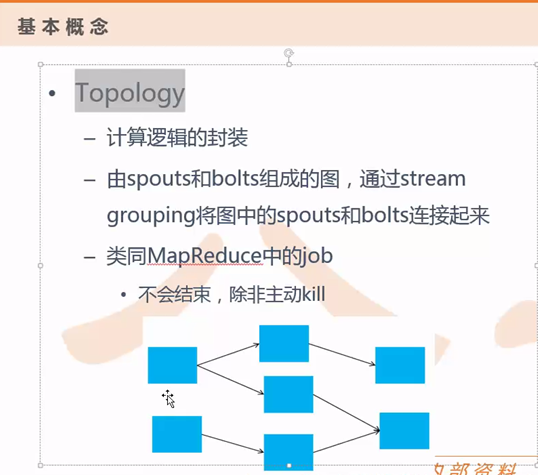
图13：



那这个Stream数据流，刚刚说这个Storm就像一个水管，你可以认为这个水管里面流躺着个水，就是数据嘛，相当于这个数据在不断的通过这种数据流的方式来去做一个数据的传输，那既然是作为数据，那肯定是有单位的，比如说数字，有这种int整型，有这种字符串型等等，那在Stream里面通通按照Tuple为单位，什么叫Tuple呢？，就是图13下面的箭头你就可以认为是生产线车间这么一个滚动带传送带，在这传送带上面有很多的数据，你把这个数据打一些包装，这些包装箱可以暂时认为比较整齐，封装的是比较规矩到位，就不会像这个数据散落一地就非常杂乱，它相当于把你的数据给封装住了，都封装成这么一个Tuple的方式，那对于Stream里面来说，Tuple是它的一个数据的基本单位，那在HDFS的数据基本单位是block对不对？所以你对应着去记。

所以Stream里面是Tuple，但是Tuple你把他能够打开，他相当于就是把一些整型，长整型，字符串型等等，把这里面的数据做一个封装，大家应该对编程比较熟悉的话呢，比如说什么结构体这个应该能理解把？其实你也可以当它当作元组，元组也是可以把很多类型的数据放在里面，所以基本上很像这个元组，可以这么理解。那么封装的这个数据可多可少，少的话就有一个，多的话就有多个，当然还可以自定义的类型。

图14



还有一个很重要的概念叫TOpology，TOpology是指什么呢？你英文翻译就是一个网络拓补，那拓补呢想一想相当于就是一个网络拓补图，就好像是图14中下面的那个图，一个网络里面包含很多个节点，这节点与节点之间是有一个相互的依赖关系和互相数据传递的一个方向性的关系对不对？那你打开之后就相当于是一个内部有很多个这样的机器或者是很多的这种单元，暂时先把它称为单元是更合理一点，数据处理单元，然后单元与单元之间是互相有着数据依赖关系的对不对？

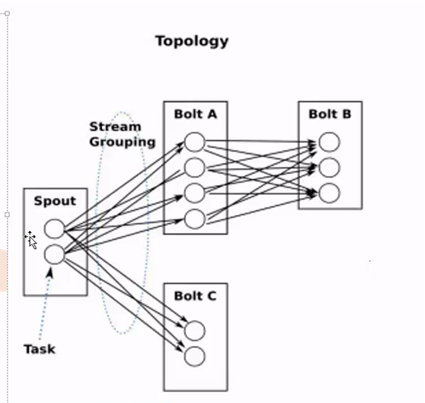
那其实这种就属于网络拓补，网络结构，所以TOpology就是描述了一个网络结构，那这个东西其实就是对应着其MapReduce里面job，比如说在去说提交一个任务再去写MapReduce的时候，我们经常说提交一个任务，什么任务怎么挂啦，任务怎么出问题了报错了，任务成功了，这个任务就是job，在这个Storm开发里面就叫拓补，其实就是叫这个TOpology，其实大家都把概念给混淆了也可以把这个当作任务关系，然后你把这个TOpology打开了，刚才也说了这里面都是计算逻辑的封装，这每一个节点都是一个计算逻辑，然后里面有两个很重要的概念，一个是spouts和bolts，这是两个原语，刚才我们说过了Storm使用原语就类似于mapreduce里面的map和reduce，那这里面的原语就是spouts和bolts这两个，那也就是说你开发的时候，你只盯准了这两个代码去开发就可以了，你把这两个spouts和bolts伺候好，你的项目就OK了。

然后什么是spouts和什么是bolts？那spouts你可以认为就是个数据的产出，其实在整个拓补里面spouts就是一个数据源，什么是数据源呢？就是在整个拓补里面的在最前面的这个节点就是spouts，你可以认为是一个水龙头，就是它控制着你这个数据的源！明白了把？

除了这个水龙头其余的后续的节点全部都是bolts。

那我们再来看一下另外一张图（如下图15所示）

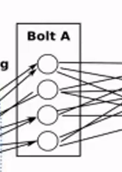
图15：



spouts就是第一个，那其余的全是bolts，spouts不止一个，你可以是一个，但是也可以是多个这个没关系，这个具体是几个这个就要看具体的设计。

这个时候我们从这个图就可以看的出来spouts和bolts之间就定位就找到了，那些节点是spouts，那些节点是bolts，你会发现还有一些细节，这个spouts和bolts，或者是bolts和其他bolts的之间它都是有连线的对不对，像这个spouts和bolts都是这个圆圈，这个框就代表着同一角色的，同一角色框着的都是这些个原语在里面，其实你假设说里面为什么是两个spouts的呢？这是可以通过参数去设置，这里面的两个几点它的角色完全是一模一样，然后因为有多个这样的一个圆圈你可以理解出他们的数据处理都是同步进行的。

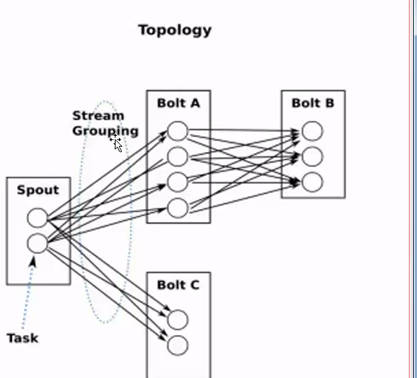
图16：



然后这个bolts里面有很多的圆圈，比如上图16有四个圆圈，相当于是四个并发，就有四个这样的一样小的节点同步帮你处理数据，如果你要是觉得你这个数据处理太慢了，那我相应增加里面的一些节点，比如增加几个spouts都可以，这需要你自己去设置参数，但是这东西不是说你设置的值越大越好这是不是的，为什么呢？你毕竟这个东西占资源，另外一个数据还在传输，相应的你增加的节点肯定还会付出一点的代价，但是太小的话也不行，太小的话相当于所有的数据都压在这个节点上了，跟之前mapreduce情况是一样的，单点的情况下你所有的数据都压在了这个reduce那肯定是比较慢,并发度不够，比较慢。

然后我们话又说回来这里面的圆圈节点就是原语，原语跟原语之前通常都是拿这个一条有巷的边来建立关系，那这个边是什么呢？很像我们mapreduce里面的partition，map的数据到reduce，数据怎么过去的？是通过拷贝过去的是不是？通过一个拷贝，就大概就是一个partition对吧？并发策略，直接把数据给抛过去了，那这里面相当于一个partition的概念，很类似，但这个partition呢，在这里面不叫partition了，叫Grouping（如下图所示17所示）

图17：



Grouping其实这个更好理解，就是分组，那其实它这个分组呢也类似于partition，像我们之前说的partition通常是按key做哈希，然后命中具体哪个分桶，但是在这个Storm里面也有这样的策略就是说这个Grouping这个分组也可以按照正常指定的数据，那个数据是作为key然后去做一个分组这么一个过程，就是它的这个策略就是跟mapreduce是一样的，但是它比mapreduce更丰富，除了这种mapreduce传统做哈希做partition这种策略之外，你的这中图17里面的线边可以去做配置，你还可以设置成随机，当然还有其他很多模式，这个后面我们会一一的讲，你暂时先把它理解成是一个数据分发的一个策略，所以这整个这个TOpology里面有很多的概念，一个概念就是spouts，一个概念就是bolts，然后原语原语之间都是有这种Grouping存在的，还有这个图17中的Task指的也是个圆圈，这个圆圈是一个原语，那跟Task有什么关系？Task在Storm里面就是一个线程，那你跑的任何一个原语，你跑的一个spouts或者是bolts来说，Task它都是一个线程，不是mapreduce里面的进程，在这里面任何跑的spouts一个bolts，它都是一个线程，就是多线程。

然后图17这个图大家第一次见到是不是？先给大家一个初步的认识，然后我接下来再对图17的图再做进一步的学习。

此时已经进入了上课中间休息阶段，请进入大数据Storm笔记2文档！！！