Flume它是一个消息采集系统，什么是消息采集呢？消息就是说你的数据源也就是你的消息源，在这个用户他会通过一系列行为他会留下大量的行为数据或者是行为消息，那这些消息都是更接近于更原始的最原生没有任何过滤的一些有价值的信息提取啊，相当于是整个的一个记录序列里面，它既有价值信息又有参杂的一些过渡修饰的一些结构，需要被过滤的一些消息，那这个时候你需要把这些大量的消息从数据源开始进行一个收集，因为用户在去留下它们的日志行为的时候，其实这些行为都会被留在了或者被散落在了各个不同的服务器的一个角落，那相当于这些服务器也就是散落在不同机房不同地域的一些各个的数据节点上或者一个服务器节点上，那这个时候它这个数据是非常一个散落的状态，这个时候就需要一个服务，把这些散落的这些原始日志进行一个统一的一个收集，然后供后续的整个的流程比如什么数据过滤，数据入库，数据挖掘等等这些后续我们有待于去进一步操作的事情，所以第一步首先想办法怎么把这个最原始的数据先对接过来，那这个时候就需要用类似于Flume这样的一个消息采集系统

到目前为止我们在之前的学习过程中已经完成了很多的一些有重要的一些组件，那不同的组件其实有各自的特点然后每一个组件都适合不同的场景，那其实我们在之前不管学hive也好学hbase也好或者是学hdfs，那你会发现这么多个组件其实在整个的这个架构里面它们都处于一个完整项目的一个中下游，中下游就是相当于说是消费者这么一个状态，那起码你的有生产者，生产者你不可能消息一开始就生产到了HDFS上对吧？然后这个时候你需要通过一个中间介然后把最原始的消息采集过来然后再去传到后面不管是HDFS上还是hbase这些等等的存储上面去，这个时候相当于我们之前学的不管是HDFS，hive，hbase都是属于下游的一个角色，而且我们还学了一些集成框架，这框架有mapreduce，storm以及我们后面要学的spark都是来解决数据计算的一些问题，然后hive,hbase主要是解决一个数据存储和结构化的问题，所以这个时候我们既然已经学了这么多处于一个从消费者的状态这么角色一些个组件的话，我们就要想我们这些数据的源头是哪里对不对？

那很多时候我们做项目的时候这数据已经给你好现成的了，你就直接去做处理就可以了，你不需要关心这个数据源在哪里，但是你从一个完整的一个项目的宏观角度去观察，你必须要知道这个数据它的来源是在哪里是不是?所以为了保证整个的项目的完整性，保证你对整个的一个数据流的一个打通一个鼻环的一个认识，就是你的数据采集这一块也是有必要掌握的对吧？所以数据采集这一块我们通常就用这个flume方式进行一个消息采集

那说到消息采集，就得有一个数据源，然后这个数据源有了之后还需要一个数据采集，这个数据采集就是我们今天要注重介绍的地方，好了你有数据采集之后那你接下来就需要一个相当于是把这个原始信息拿到了是吧？那么就需要把这个数据做一个缓存，先把这个消息进行存储起来，然后存储起来之后因为你这个消息会存在着大量的无效的一些信息，你需要做一些有效字段或者有效结构化一些提取，这时候就涉及到了数据过滤环节。

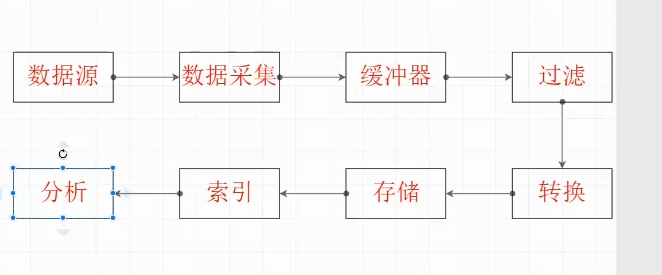
跟着这样的一个思路，从数据源开始通过一个服务把数据采集下来，采集的数据需要通过某一个存储或者是某一个缓存把它暂时的存储起来，起码这个数据先落地，落地到具体哪个位置的话这个先不用考虑，起码先把这个数据拿下来，那下来这个数据因为比较原始所以你需要对这个数据进行过滤，因为这数据存在大量的无效的数据。

然后接下来过滤完之后你需要做一些个转换工作，比如说你这个数据是从一个非结构化的数据变成一个结构化的数据是吧？你怎么把这个数据从你的文件系统里面怎么样转换到一个像数据表格那样，字段然后记录行列之间非常清晰分布这么一个状态，这样有便于后续的一些分析，那转换后的数据就很明显了，需要把它进行一些存储，比如说把它存到HDFS上或者存在你的hbase上等，好了你把这个数据存储完之后那接下来需要做一些检索工作，就需要用于一些检索用，那检索相信你这个数据存储的话你是肯定是要后面来用的对吧？那怎么让下游用的更方便或者是更快捷？这里面就肯定涉及到了一些键索引，那你比如说像之前搞过mysql的人，然后为了让你的数据检索的更快那它自身会支持一些个索引。

那在整个的大数据里面比如说这个HDFS，比如存储在hbase上那这个时候估计大家就会有疑问了，这个索引怎么建呢？那这个时候其实跟我们后面要学的数据挖掘部分怎么去做一个数据分析还是有一定的关联的。Hbase这块它也是支持一定的索引的，所以索引这一块后面还能接触的到。

那不管怎么说就目的就是能够快速的检索你的数据，那检索到数据之后就开始做一些个数据分析，然后分析后的数据就是把你有用的信息怎么去挖掘出来，然后把挖掘这最后的数据进行一些服务，大概是这么几个环节（如下图1所示）

图1：



那整个的一个数据的一个走向那从上游慢慢从水游一样游到了下游，那你会发现数据采集这一块是非常非常贴近于最源头了是吧？那所以今天我们就要了解数据采集这块，那有很多人不太理解为什么你今天非要讲flume，为什么不讲kafka，那有基础的同学那在上图1中里面应该是处于哪个模块？Kafka是一个消息缓冲器对吧？那么相当于kafka是相当于上图1中的缓存器，那么我们学完数据采集再去学kafka，像这样的话你会发现会整个的思路来说会比较更顺畅一些是吧？我们先学上游再去学下游这么一个思路，那么这个过滤是需要做什么呢？这个过滤结合我们之前的内容来说，我们这个过滤如果要做你想怎么做？你可以用mapreduce用storm等去实现，我们之后会有一个案例，怎么把flume和kafka和Storm关联起来对不对？相当于整个链路就打通了。那转换这一块就好说了，不就是转成hive或者是hbase。

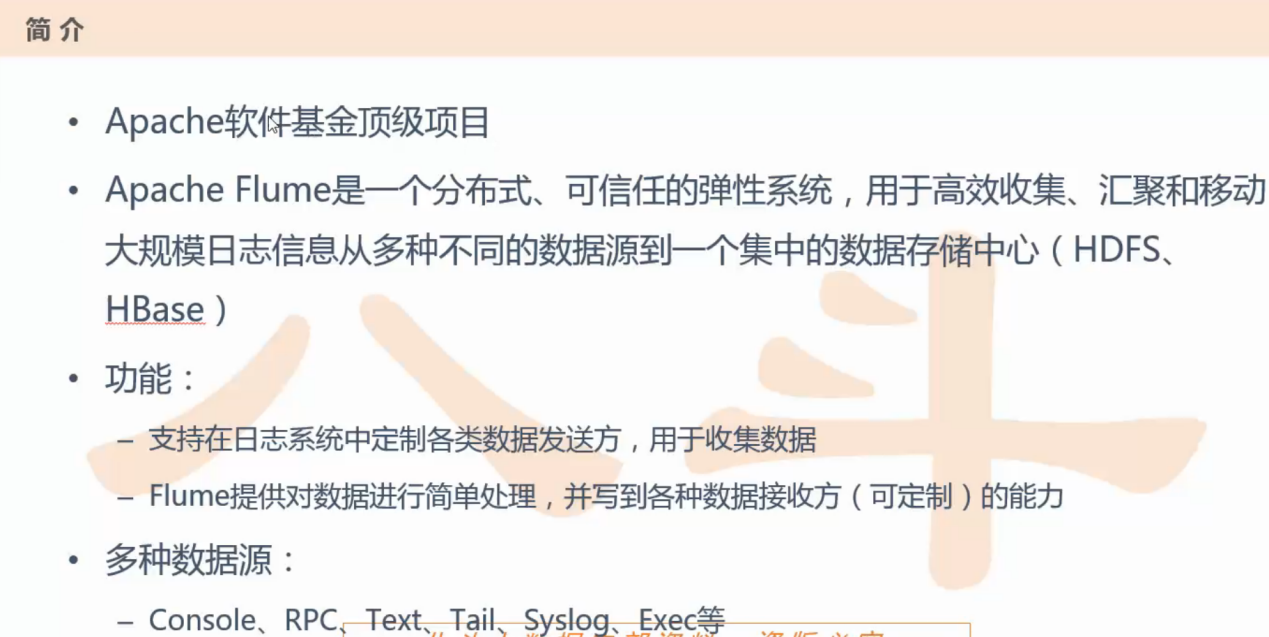
好了我们正式开始讲flume,那么flume也分别分为三个部分，分别是理论简介和核心知识包含我们后面的实践（如下图2所示）

图2：



那我们开始进入简介了，我们快速的过一遍，因为这个flume的理论是比较简单的（如下图3所示）

图3：



那么这里就没有什么好说了flume是apache的一个顶级项目，它是一个什么日志收集系统，它是一个分布式的，然后收集的过程是可信任的，什么是可信任呢？主要是指它有一个事务性，就是保证它的消息能够有效的处理和传递，然后它有收集数据的功能，就是从各类的数据源进行收集数据，那这个数据源有很多很多种，我们大概列出几种把，就是你的数据源可以从哪里来呢？第一个就是从你的服务器日志里面(server log)，大家有做这个后端的同学可能会知道像webserver这一块，如果有用户请求过来了，它后面的log是不断的滚动的，就像我们去搭建一个服务的时候一样，比如说你搭一个storm服务这些服务，那它不是有一个log日志吗？然后随着你的这个集群的状态不断的更新，它的这个日志是不断的滚动是不是？那既然不断滚动的话，我们通常去检查日志那粗暴一点就是vim，但是不建议大家这么做，因为你的日志文件可能会很大，如果vim打开这个非常大的文件，是非常慢的。那么咋办呢？那么通常就是用tail命令，比如我看最近的一万行的日志就是:  
tail -n 10000 a.log | grep fatal/warning，然后grep 出看看 有没有fatal/warning这样的一些错误，

那还有就是通过远程调用的方式，比如通过一个http接口，通过url的方式来请求你后端的一个服务，那你后端来接收这么一个请求开始对它做一个请求的应答，那对应你一个消息采集系统来说其实通过http url的方式它也可以携带一些有价值的信息传递到后面来对不对？还是有一种也是通过网络的方式，那这种网络的方式可能通过netcat，就相当于指定一个ip或指定一个端口号，然后生产者往这个ip以及端口号上面去不断的发送消息，消费者是从这个ip和端口号上面是不断的去取消息是吧？一个生产一个消费，那这也是通过网络的方式来进行的。其实还有很多比如文件系统，比如一个目录下面，然后这个目录下面一旦有数据变化，那它马上会同步这样的消息，会让你的消息采集系统能够第一时间获得它的一些感应。这图3中也列了几种数据源的方式，有文本的，有远程的，有终端方式去发送消息，所以这一点大家要了解首先就是说flume它去做消息采集的时候它支持的场景还是很丰富的。

好了对于整个从一个简单的一个视图大概去了解整个的flume的话，那flume其实在整个的消息采集系统里面是在这么一个位置（如下图4所示）

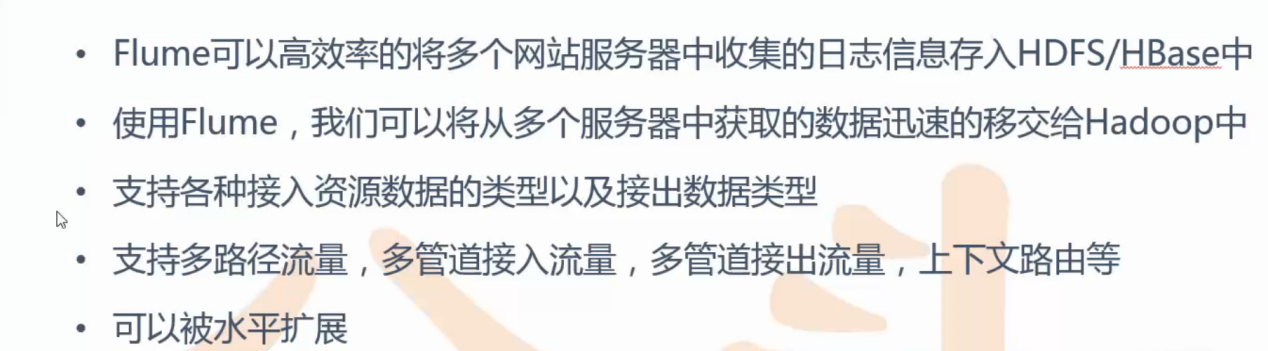
图4：



它属于承上启下这么一个角色，那承上就是左边的这个就是各个服务的产生的一些数据对吧？那下面就是flume采集一些消息你需要把这个消息进行存储，那存储什么地方呢？就是存储在HDFS或者是hbase上或者是存到你的本地的文件系统都可以，所以它的左边的接口流入的支持还是比较多种多样的，可以从远程的方式去支持可以从本地的方式去支持，然后你数据的输出也可以往远程的服务数据库上去打，你也可以往本地去下载数据或者是保存数据，主要flume相当于就是一个吸管一样，那么有什么办法通过这个吸管把这个左边的数据不断的吸收过来给HDFS或者是hbase上，而且你后面的分析，数据挖掘都是在你的统一的这套存储系统上去做的，主要是想办法怎么去让日志消息收集过来，就是这么一个作用。

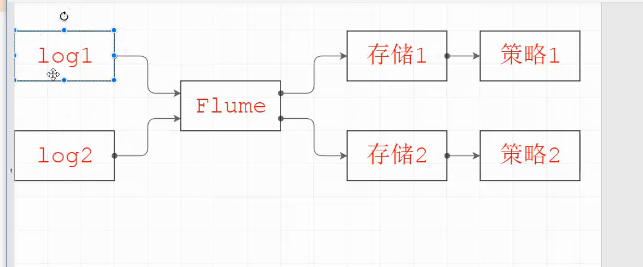
然后这里面列了关于flume的一些特点（如下图5所示）

图5：



其实flume做这个事情它是很具体很直接，所以理解起来很容易，它就是把不同的服务器上的消息存入到HDFS或者是hbase里面去，然后它支持各种的一些接入，这个怎么接入呢？这个刚才说过了，通过很多的方式可以去接入这样的消息然后并且接出的消息类型也很丰富，而且它还支持多路径的流量，这个的意思就是比如说多个log服务器和一个flume和两个存储系统，好了那我这个日志就可以通过flume集群，其实这个flume只画了一个模块，其实这个flume你可以搭建一个集群的方式通过多个不同的机器来维护整套的消息采集这么一个系统，因为这个消息量还是很大的，你只要通过一台机器的话，那通常是搞不定，所以flume还是通过一系列集群来去并发的收集信息。

然后flume的这个数据也可以进行一个多路输出，意思就是你可以把一个消息可以选择性的去存储到1还是2中，比如说你这个存储1，你这个存储1后面走的挖掘策略就不一样了。这存储2就是另外一套挖掘策略方案（如下图6所示）

图6：  


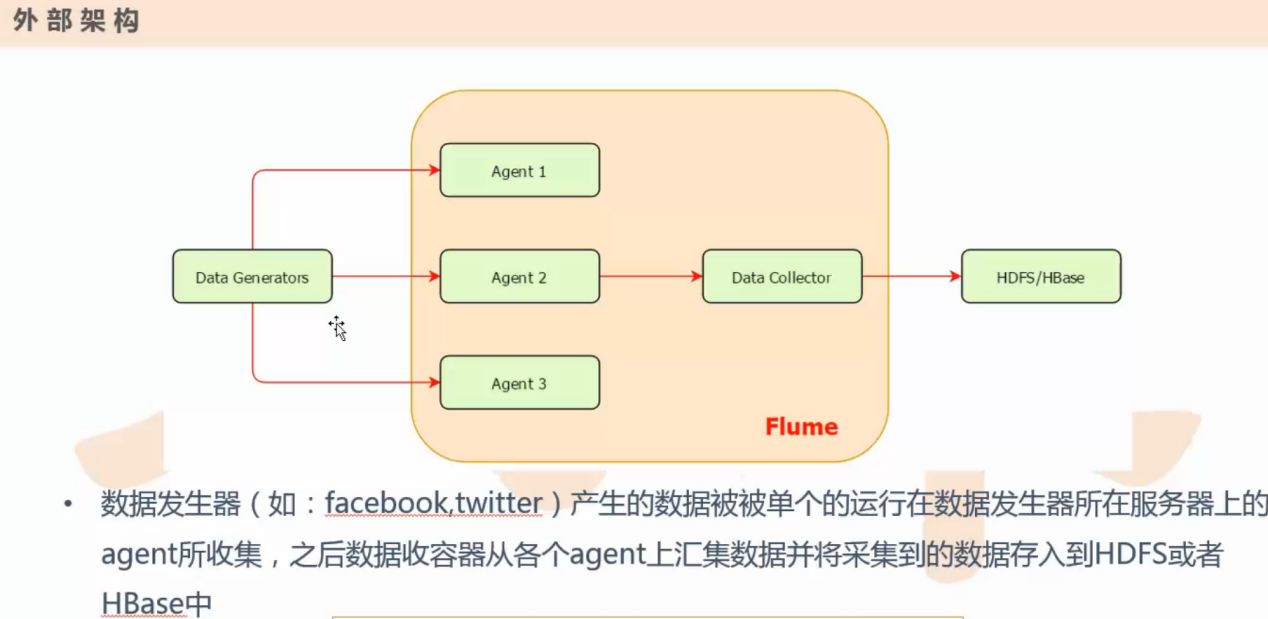
而且这两个日志可能不是完全一样的，比如说这个日志1是来收集展现，这个日志2是来收集点击的，那这个策略1可能就是对展现日志进行一个处理，这个策略2对点击日志进行一个处理，那所以这个时候我们希望展现日志可以通过这么一个管道的方式能够有效的流入存储1里面去，我就不想这log1里面的日志不想流到存储2里面去，那么相当于是flume一旦发现你这个源是来自日志1，那我就可以自动把你这个数据直接放到存储1里面去，并不是说把你的数据能够复制，把同样的一个日志只要你这个下游都是对接到了flume上，那下游所有的节点都会收到同样的一份消息，那flume可以这样有区别的进行对待处理，这是一个它可以指定有效路径的方式，这个方式叫做复用机制。

还有一个就是复制机制，就是说我不管你这个flume前面这个数据源是什么，只要是你来了一个数据源，只要是我这个存储是属于flume下游，那我所有的存储器我都会接收到同样的一份数据，比如说你这个log1的一条数据进来了，那比如说后面有两个存储，那相当于把你一份的数据我复制成两份，每一份节点我都发一份，这是一个复制机制。

然后多管道接入流量这块也可以体现出这个问题，就相当于这两个日志有两个管道，那这两个log日志是来自不同的一个日志源级别的，这时候flume就可以通过不同的管道去对一些不同的日志源，然后多管道接出，剩下一个上下文路由，路由刚大概说过了，然后水平扩展这一块因为这个flume通常用的时候也是通过一个集群的方式去用，这个flume你可以进行一些个扩展，比如说你一些节点就是在数据采集过程中不够用了，你可以往上面加一些个节点或者资源然后共同的支撑共同的并发，相当于是可扩展性比较强把。

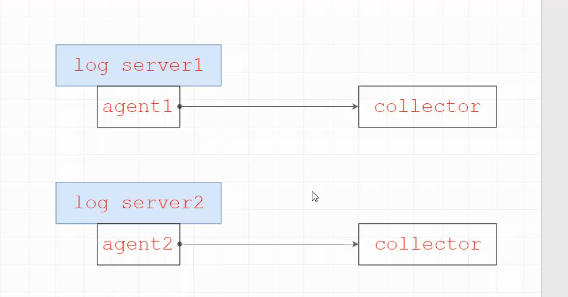
那解下来我们进入到flume一些知识的细节，让我们对flume有一个更深入的了解，所以我们看一下整个flume的核心知识点（如下图7所示）

图7：



那我们先从外部的整个框架入手，那最左边是一个消息的一个发生器(Data Generators)，什么是消息发生器呢？就是一个日志服务器，比如说就是你们公司里面那个用户接收请求的它的一些请求信息，然后一些收集的这些服务器，就是一些webserver，那相当于日志发生器就是它已经开始陆续的产生消息了，那后面有一个橙色的一个大框对吧？，这里面就是一个整个的flume，然后flume就是从这个发生器(Data Generators)日志来进行采集，采集之后又得到了后面的一个HDFS或者是hbase存储里面去。

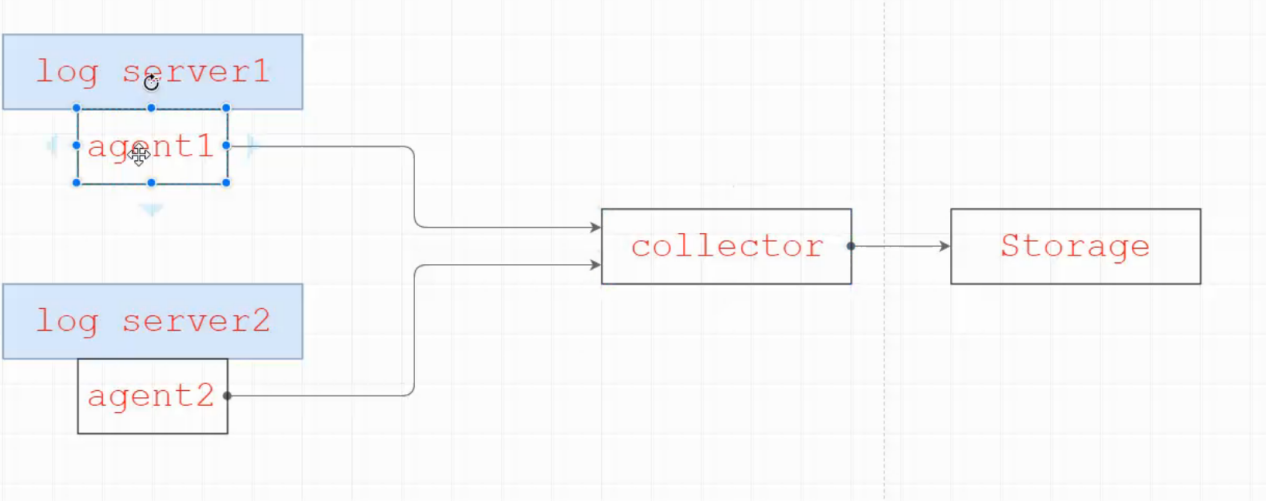
好了这个时候把这个整个的flume这个黑盒的面纱解开你就可以看到这个flume里面就大概长这样的一个样子，里面分成了一个Agent和一个Data Collector，Agent的意思就是一个代理模块，它是用来对消息进行接收和汇集。比如有两个log server1和log server2，那么这个Agent通常是部署到跟你同一个server同一台机器上，你这个og server1是用来不断的产生的日志消息，然后你一旦产生这个消息由这个Agent1这个消息来从你这个server上直接发送出去，那把消息发送给谁呢？就发送给一个叫collector（如下图8所示）

图8：  


所以你从这图8里面就可以看出我虽然这个flume里面包含了Agent和这个collector，其实通常来说Agent和这个collector是分班部署到不同的节点上的，就是结耦。

通常来说就是你的Agent会很多，因为你这每一个log server都有一个Agent，那你这个server通常会很多对不对？所以Agent也会很多。那这个时候你后面不是像上图8画的一样，一个Agent和一个collector一一对应并不是这样的，通常来说就是一个Agent可能会对应多个collector，就相当于是你前面有多个Agent的消息统一的被你的collector进行一个收集，一般来说server和Agent是一比一的，好了那Agent把消息发给collector，因为这两个属于不同的机器，这个时候collector会去把真正收集到的信息再去做一个存储（Storage），因为这个存储就HDFS或者是hbase，所以这一块就不需要大家开发了，那你就需要把collector怎么能够通过一个配置的形式把前面消息能够直接发送到指定的相应配置的目标路径上去就可以了，所以通过这个地方大家先了解一下你的Agent和你的collector的定位是怎样的（如下图9所示）

图9：



Agent就相当于冲到最前线的，这个collector就相当于是后方基地，然后不断去接收前线的一些消息，然后它在把这个消息怎么再往存储上再去做处理，collector也是可以多个的。

接下来再看一下我们在去讲storm的时候，storm也类似于这么一个流程是不是？从头往后一个数据流进行一个传输是不是？然后再storm里面它的数据流也是有一定的单位的形式做传输对吧？那这个storm的单位是什么呢？还记得把？就是type对吧？，那在hdfs上数据的单位是一个block对吧？在flume里面数据单位是Event，是一个事件。假如说在整个的flume里面它内部流转了这些消息都是一个事件，所以flume是用这个Event对象来做一个消息传递的格式（如下图10）

图10：

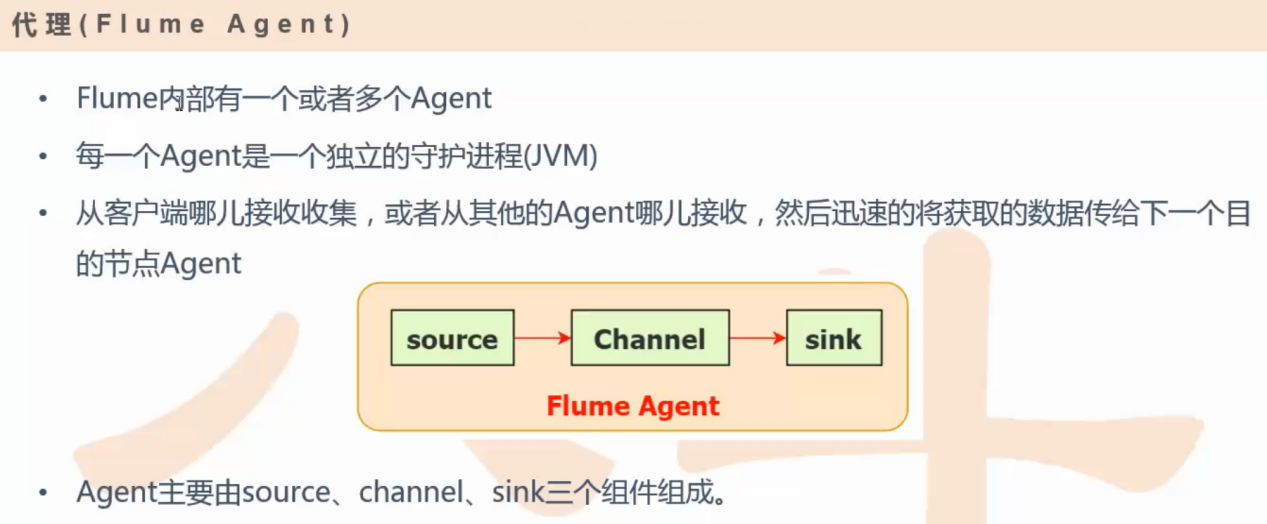


然后它属于内部数据传输的一个最基本单元，那你把这个事件已经打开，打开它有两个部分组成，那第一个部分就是一个Header，第二个部分就是一个Byte Payload，就是你的头和身体对吧？通常这个数据这个头部你可以有也可以没有这个可以选择的，不一定说这个header就是一定要存在的好吧？那如果说这个header要存在的时候可以理解为一个key，body你把它想象成一个value，如果你把数据有一个key有一个value，大家很容易想到的是在mapreduce里面有一个partition对吧？partition就是用来做分发消息的，也就是说这个Event有两个部分，Header和Byte Payload，这个Header是可有可无，如果是它没有这个Header只有byte Payload的时候，那么byte Payload其实就是存的是数据，那这个时候数据就开始往后流向进行传输是不是？这是一个最直接的流程，但是有的时候你需要对这个数据做一些个路由，最后一些个分发，就是说有的消息我想分发到A节点上但是我不想分发到B节点上，有的消息我想分发到B节点但是我不想分发到A节点上，那对于这种有特殊需求的情况，这个时候你就需要用到header，你必须要分配给它一个key，然后这个时候它做分发key的时候就根据你这个header里面的信息去做一个数据的一个路由，有点类似于分桶，所以相当于把这块跟我们之前学的partition结合起来可能理解起来更容易一些。

上图10中写的header是key/value形式的，这个其实跟我们说的key和value不属于同一个层次，就是这个header如果你要是有这个信息，这个信息就大概长（k:v，k1:v1）这样的样子，然后你去做一个partition的时候你就根据这个key可以value去做一个分发这么一个情况，所以这个时候大家就记住一点我的header就是为了做分发用的，Byte Payload就是存实际的数据内容用的。

好了这个时候就讲了一个比较重点的东西，就是一个代理（如下图11所示）

图11：

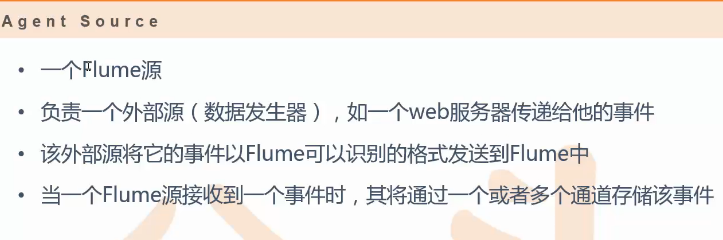


这个Agent刚刚我们讲过了，这个flume可以拥有多个Agent，当然也可以拥有一个，然后每一个Agent就是一个进程，这个进程就相当于是在你的服务器上一直运行着然后一直监控着你的这个消息，一直监控着你这个日志的产生，一旦你有日志发生变化了，那这个进程就会把你的消息进行一个数据的收集然后往下游不断的传输。

如果说你把这个Agent再进行打开，再进行把你的内部细节暴露出来，Agent就可以暴露出三个部分，这三个部分就是source和Channel和sink这三个，那么这三个模块有什么用呢？source就是真正对接你数据源进行输入，而Channel就是一个管道，sink就是一个输出，就是你的source把消息接收过来，然后消息会存到你的管道里面会做一个缓存，缓存我们之前学过一些对吧？这个缓存存储可以是文件形式的，相当于就是在你本地一个落地到磁盘上的那个文件对不对？还有一个就是在你内存分配出一个区域，就是这个数据在你内存中扭转的，不落地这是通过一个memory的形式，所以你的数据输入是可以存在你的文件里面也可以存到你的memory里面，那存到memory里面会更快一点，但是有一个问题就是一旦你这个agent出现了问题那你这个消息因为你存在memory里面，在消息可能会存在丢失的风险。但是为了保证你的消息的可用性可靠性通常建议把消息直接存在你的文件里面，但是这是存到你的文件里还是存到你的内存里面这是你要通过一个配置去配置的。

然后输入就是对接各个数据源，输出就是对接各种存储，所以相当于是每一个组件都是各司其职，然后彼此之间能够协同工作，然后让消息能够有效的在内部进行一个扭转。如果要是在Agent里面我们在对各个组件在做一个更深入的了解，那我们接下来看一下source（如下图12所示）

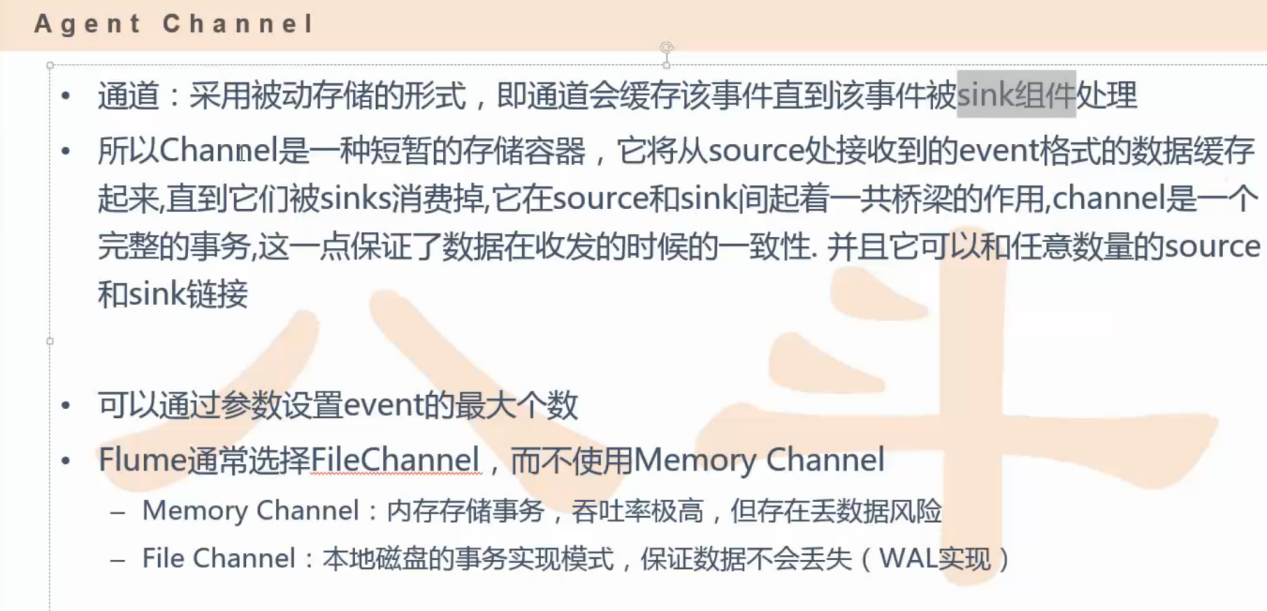
图12：



source是一个整个的flume的源，它是最贴近于你的消息源的那么一个模块对吧？。它相当于就是一个数据源的外部采集，然后把它外部源数据接收过来然后变化成flume可以识别的格式。这个格式就是一个事件（Event），然后在从这个flume开始内部进行流转。

然后Channel就是一个通道，刚才我们一直说缓存，你可以把它理解成缓存就好，那它都是采用一个被动的存储形式，也就是说这个通道它会缓存这样的一个数据，然后直到这个事件被sink组件来处理（如下图13所示）

图13：



就是这个数据先放到这，然后通过sink来异步的方式来去抓取，所以这个Channel是一个很短暂的一个数据存储器

这时候大家会有一个疑问了，你是一个存储器，那如果要是采集的这个消息量非常大，那一旦超过了你这个缓存的限制，那相当于你的内存就爆掉了对不对？这时候会导致一些节点的风险对吧？会不会有一个数据的累计，然后不断的去膨胀这么一个风险。那这个其实处理和之前学的storm有点类似，它是可以通过配置去配置的，那配置的话就是控制流量，就是控制你这个source从外界接收这个数据每一次接收需要接收多少个事件，它是有一个流量控制的，如果你前面流量放的很足那肯定会对这个存储内存会有一定的压力，那一旦有压力你可以减少这样的一个采集量就可以能够进行一个减缓，这是可以通过一个配置来进行配置。

另外一个就是消息传到存储这块来之后，那它需要sinks来去对它进行一个消费，所以它这个Channel在这个source和sinks之间搭建了一个桥梁作用。那刚才我们说过了这个Channel它既然是一个存储，那你这个数据可以存到你的FileChannel里面，然后memoryChannel都是把数据存到内存，吞吐力高，效率高，但是容易存在丢数据的风险，那么FileChannel就是需要把你的数据落地了是不是？一旦你这个机器挂了，数据也不会丢失。

然后sinks相当于就是在整个的Agent里面，sinks就是一个消费者，怎么把这个消息消费掉，那消息是在你的Channel里面，sinks会将你的消息或者你的事件从Channel里面进行，然后并且把你的事件开始往外输出，输出到外部的存储上面去（如下图14所示）

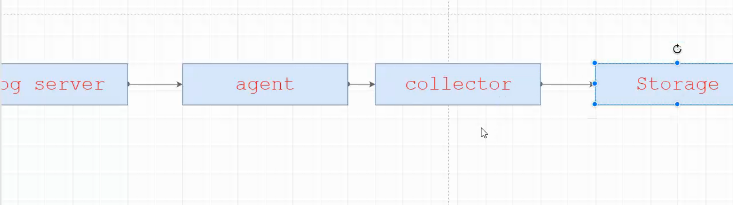
图14：



然后一旦这个消息被sink消费掉之后，这个消息就会从这个Channel里面就会移除，它有点类似于队列的形式，那最后你这个数据是以哪种存储的形式落地了呢？是由sink来决定的，也就是你需要通过配置来控制sink你最终数据是怎么样的方式输出，你的数据是可以存在你的HDFS上或者是hbase上，它没有一个默认的一个输出，这个需要你通过一个很简单的配置你可以来控制这数据是怎么样输出来的，另外一个其实在整个的flume集群里面它是可以允许有类似多个flume把，然后进行一个彼此之间的一个关联，这个像flume我们刚才打开过，它主要里面是一个Agent是吧？然后你可以把它当中一个玩具一样，然后进行一个彼此之间的拼装，然后你可以把这个集群做的规模很复杂或者一个很简单都可以，所以它在整个的集群或者是消息采集过程中它的这种集群搭建还是很灵活的。

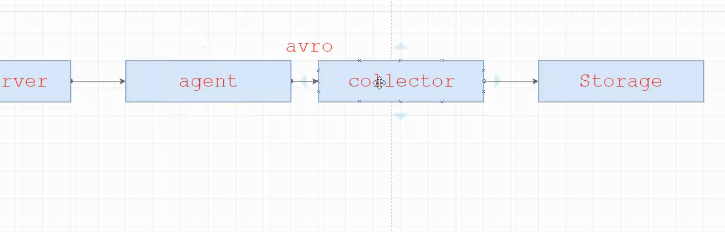
比如这个Agent如果你在本地搭建的时候，你这个Agent是可以直接存储到你的存储上的，这是可以的，但是有的时候你的Agent和你这个server是部署到了同一台机器上，那你这个机器就是为了来存储日志的，那你在给这个机器再去开放往这个Storage这个机器上去写的这个权限就不太合适，所以就需要把Agent数据再转到一个统一的一个中心，然后这个机器就可以进行一个对外的写服务（如下图15所示）

图15：



这个是一种形式，另外一个collector它得把这个分散的数据进行收集，所以通常用的时候就是配了一个agent和一个collector，但是从字面上来看这两个感觉差距很大，其实agent和collector你会发现配置的时候基本上是一样的，只不过是数据源不一样，agent的数据源是来自于你的外边真实的外部数据，你collector来的数据是来自于你的agent，就相当于你来自你的flume组件，其实这个agent和collector本质是一样的，只不过是数据源来源不一样，只不过是为了区分他们的角色，如果说遇到了这种内部组件之间的一个对接就相当于这两个flume之间的一个对接的话，那这个数据通过传输的方式的话就需要通过一个avro的方式进行对接（如下图16）

图16：



你collector要对接的话必须要通过这样的方式去对接，你这个agent这个就很丰富了，对接着这种数据源的格式，数据源的类型就很丰富了。

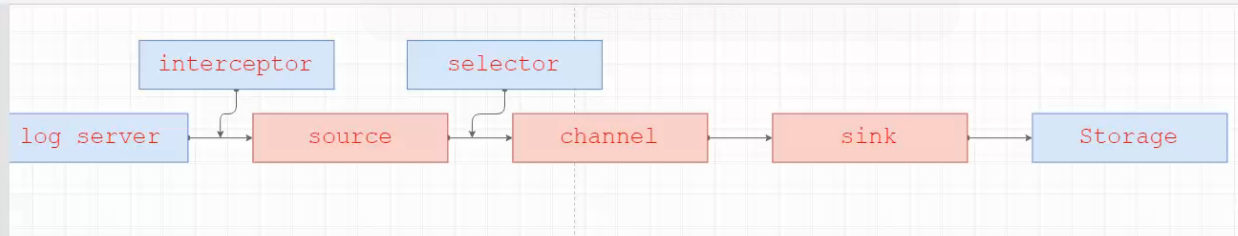
刚才我们已经说过了它一个agent内部分为最基本的三个组件是source和channcl和sink，那么这三个组件都是缺一不少的，但是它还有两种组件是可以选项，就是说你可以用可以不用，根据你的业务需求，如果你的业务需求确实是涉及到了这方面的一个要求那你这个组件就应该用，那么source和channcl和sink这三个是必须要有的，还是有两个可选的组件分别是interceptor和一个selector，interceptor是拦截器（如下图17所示）

图17：

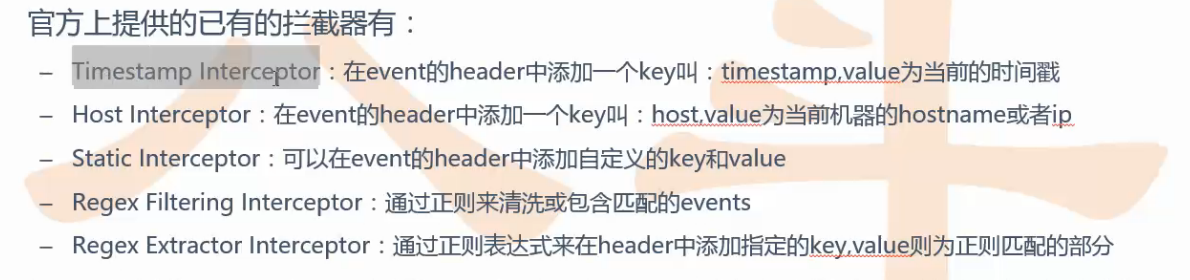


这个拦截器是在什么位置呢？就是说这个interceptor是在你的数据源和你的source之间的一个环节，那这个环节相当于就是可以对你的数据源提前会做一个过滤，然后这个selector是有点类似于路由选择，就是消息已经在这了，我开始对这个消息进行一个存储，你这个消息不是要存到channel上吗？如果你后面如果有多个channel的话，那这个消息我应该是存到哪个channel上，或者是所有的channel都应该存储这样的消息，这个时候就要配置一个selector，所以selector这块就是像我们最一开始说的复制与复用（如下图18所示）

图18：



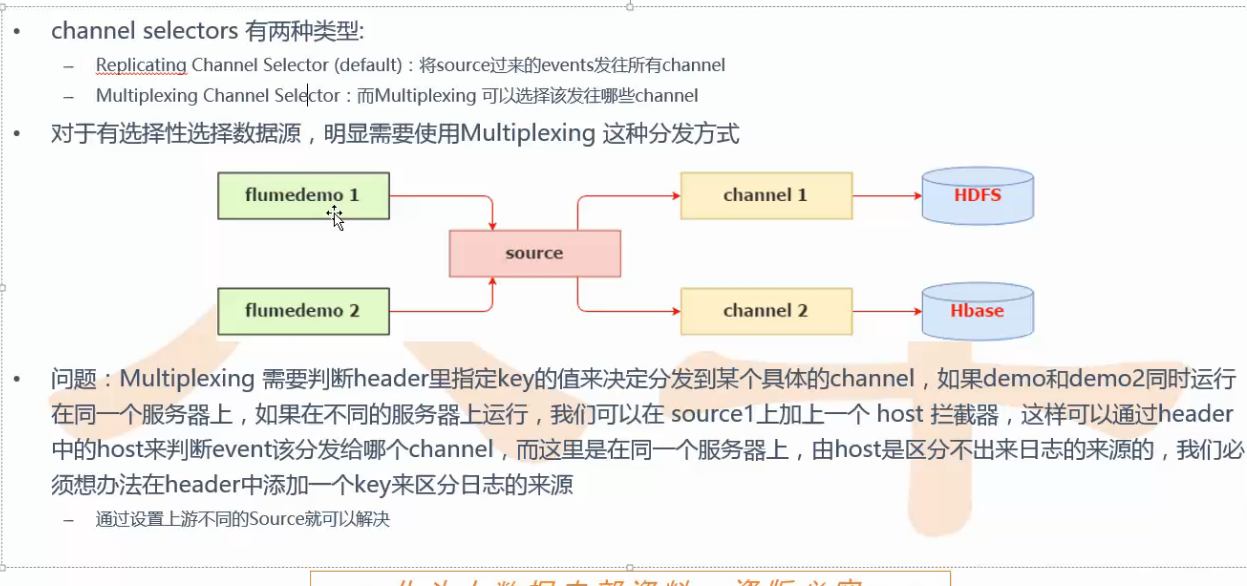
我们继续看拦截器，拦截器主要是对这个event进行一个过滤或者是自定义一些处理逻辑的实现，它主要是在你这个日志与source之间的，然后对这个日志进行一个拦截，就相当于提前制定哪些日志可以往后传，哪些数据可以直接被丢掉，然后它除了拦截之后它还可以对你的日志数据重新做一些个包装，那主要的提供的一些拦截器就有这么几个（如下图19所示）

图19：  


第一个是时间戳拦截器，相当于是它会在你的header里面添加一个key，然后这个key就是一个时间戳，然后第二个host就是在你的header里面添加一个host，那么第三个这个静态的是通过一个自定义的，你想定义什么一个key和value都可以，然后第四个就是一个正则匹配正则过滤或者是第五个正则选择，正则过滤就是说我这个消息里面，如果出现了一些非法字段或者是这个消息里面不满足，如果命中我这个正则的话我就会把这个过滤掉。还有第五个就是一种一旦我这个消息能够匹配到正则的话，那我就对这个消息进行一个采集然后把其他的过滤掉，然后并且你看有这么多拦截器，其实这些拦截器可以直接互相组合，就是不仅仅通过一个拦截，可以通过多个进行一个拦截器的拼装，通过这个chain的方式进行一个组合起来，然后对于组合之后的话，你可以对它进行一个前后的一个顺序依次的处理。

然后我们再看一下这个selector，这个selector这块也是容易理解的，刚才我们说过这个selector它有两个事情，一个是复制和复用对吧？那这复制就是分别对外两个配置，一个配置就是这个Replicating，还有一个复用Multiplexing，复制刚刚讲过了，就是一个消息能够被复制多份，复用就是一个消息可以选择性的去选择（如下图20）

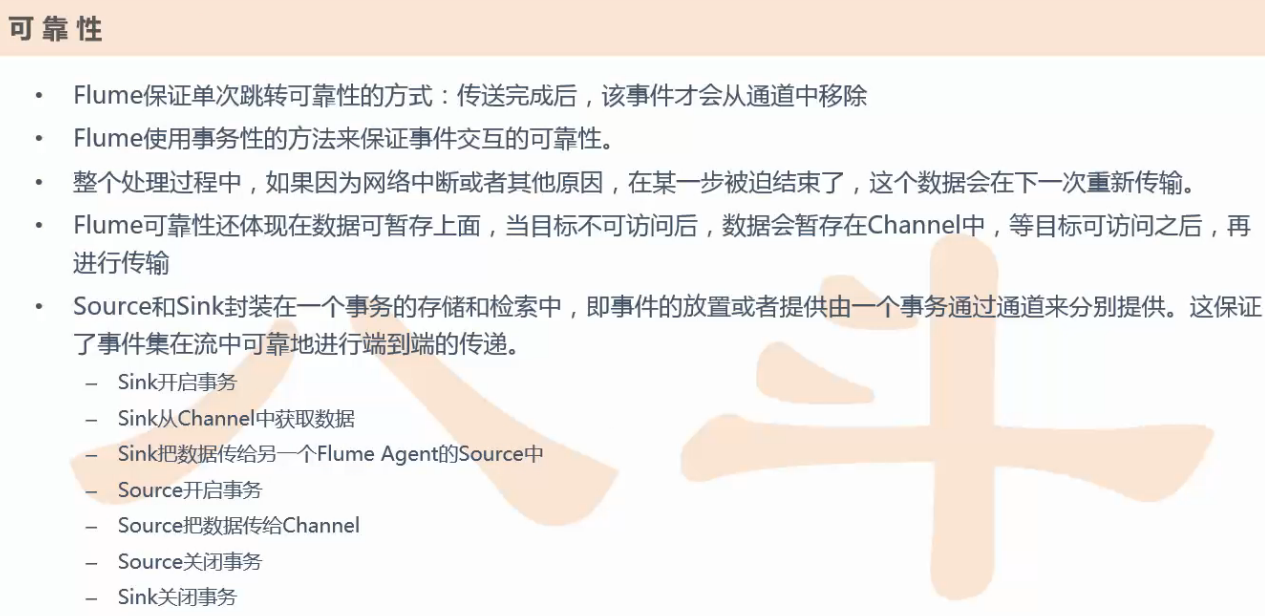
图20：



这个source前面有两个不同类型的消息，那这个一个类型的消息你可以选择后面的一个channel，如果只选择某一个channel去做传递消息的话，你可以选择复用的方式，如果这一个消息可以被复制多份，就像一个广播的形式发送消息的话，广播是什么意思呢？广播就是一个消息，被复制出多份然后下游每一个节点都同时接收同样的消息是不是？，那这样的情况就可以用复制的形式去用。

然后接下来就看一下从整体的角度来看可靠性，那为什么说这个flume它的可靠性还是比较OK的呢？那从这么几点来看（如下图21）

图21：

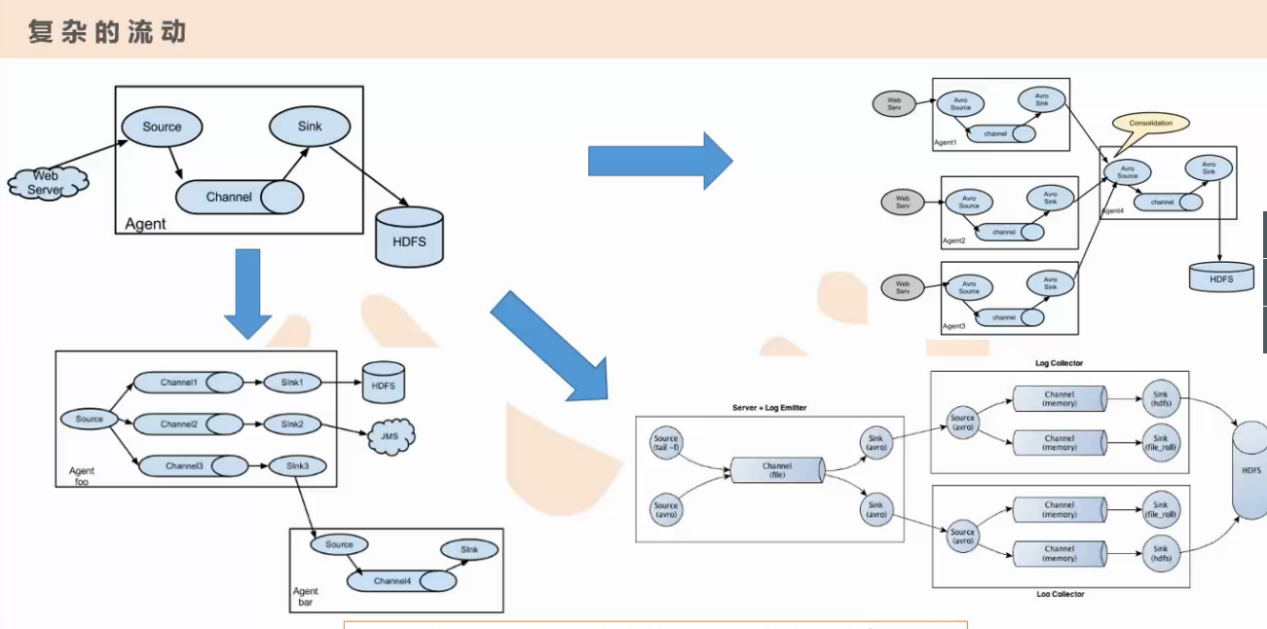


首先就是说它有一个事务性，这个事务性什么意思呢？就是我们刚才已经提过了，它主要是和channel类型有关，刚刚我们说了channel类型有两个一个是file一个是memory对吧？为了保证消息不丢失，为了可靠就是你可以选择file对不对？通过file channel的方式去传输，然后另外一个就是说当我这个消息在传输的过程中，当传输到了下一个节点上，那如果要是说接收的这个节点出现了一些异常，比如说一些网络异常，那由此就会就可以导致你的数据就需要重发的，然后另外在同一个节点内，如果是source写入的数据，把这个数据已经写入到这个channel里面去，那这个数据在，比如说它写这个数据它也是成批成批写的，那同时在这批之类，它整体的数据出现了一些异常的话，那这个时候就所有的数据一旦有一个数据出现了异常，那同一批的其他数据都不会写入到channel里面去，那已经接收到的部分这批数据就直接被抛弃，然后这个时候靠上一个节点重新再发送一些数据，重新再补充一些数据进来，那这里面就涉及到了事务性，那flume是使用了一个事务性的方式来保证了传输event或者传输整个事件在整个过程中的可靠性，就是说在你sink必须在你的event传入到channel之后或者是已经这个event传输到下一个channel里面或者是你这个消息已经到了外部的数据目的地之后，就相当于你这个数据已经可以认为是被下游已经完整的接收到了，就是你的数据已经在下游非常可靠的落地了，这个时候你的event才能从你的channel中进行移除掉的，所以这样的机制才能保证你的event无论是在一个agent里面还是多个agent里面之间的流转都是可以保证可靠的，并且由于这样的一个事务保证，你整个event就可以被成功的存入起来。

然后这是一个整个消息在传输过程中比如说一个端到端传递的一个步骤，就从你的前面的sink怎么把你的数据传输到下游的另一个agent里面去这么一个过程。

好了这个flume大概我们之前也说了，可以支持一个很强的扩展性是不是？（如下图22所示）

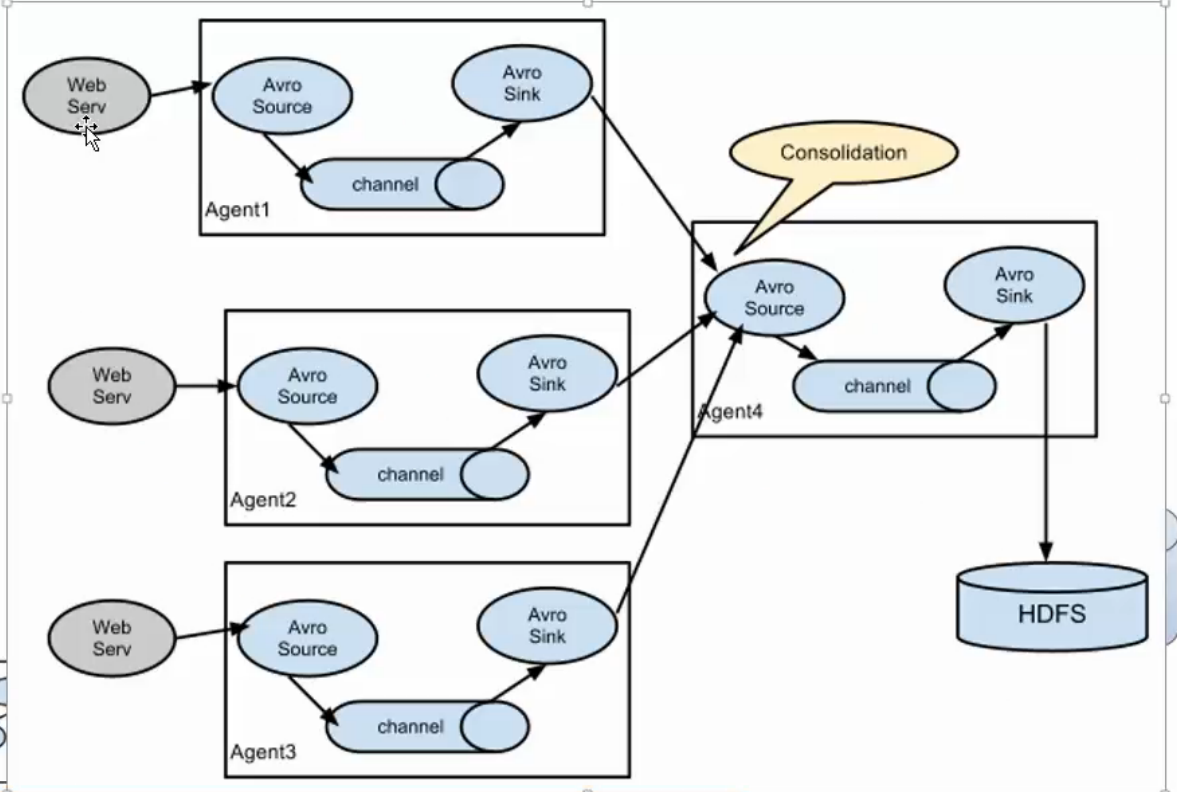
图22：



你可以把它想象成一个乐高玩具一样，然后进行一个有效的拼装，比如说这是一个其中一个组件，然后把这个组件进行一个前后的关联，你可以把这个组件之间并行多套，或者你可以把这个组件进行上下游进行一个关联，就是上游的sink要对接到下游的source上面，也可以多个sink同时消息汇入到下游的同一个source里面去，并且你可以做更复杂的一些搭建，这些都可以通过简单的配置就可以去完成。

复杂流动的目的就是说这根据你的业务场景的一个复杂程度了，那你每一套agent可能下游就是面对着业务处理的流程是不一样，这个是完全是可以根据你的进行一些选择，比如如下图23所示

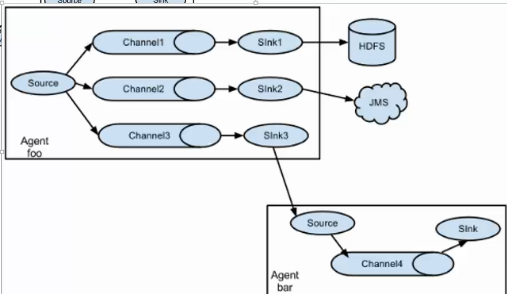
图23：



每一个webserver上都部署一个agent对不对？那你这有三个webserver就对着三个agent对不对？那你在这个实际的架构过程中你的日志服务器是有很多节点的，那你不可能每一个节点，如果是没有这个Consolldation的话，你每一个节点都去往这个HDFS直接去写的话，这个就不太合适，因为你这个日志服务器它仅仅是用来做日志收集做的，而你把它权限在放开到再去写HDFS，它相当于是那个角色定位有些混乱，另外一个就是说你这个数据源并发同时去写的话对HDFS操作也不是一个很好的设计，所以最好把这里面数据都汇总到同一台或者是数目比较少的那几种，只要是你那个压力能扛得住的一些Consolldation进行一个集中处理，然后Consolldation再去对接到后台的一些存储服务，这样会前面和后面部分相当于它们面对的角色是不一样的。

还有一个就比较典型了（如下图24所示）

图24



这个sink和这个source的一个关联。然后我再看一下这个（如下图25所示）

图25：  


这个上图25其实有点像我们之前讲的路由选择，你看这个左边的source可能通过不同的方式去对接的，然后把这个不同的消息通过一个channel然后通过多个sink，每一个sink都对街到后面不同的应该是Consolldation,然后每一个Consolldation进行各自的一个日志收集，那你这里面就跟你的业务相关了，可能是你这个Consolldation之间收集的数据是一致的也有可能是不一致的对不对？，然后后面对着不同的sink，这不同的sink就往后端存储的时候你可以写到HDFS上也可以写到hbase上，因为你这个数据是可以，比如说这两个channel就对接着不同的存储是不是？那你上面的这个channel是往HDFS上去写，下面这个channel是往本地文件去写，但是你可以再搞一个channel然后往Hbase上去写都可以很随意。