Flink面试，看这篇就足够了

概述

2019 年是大数据实时计算领域最不平凡的一年，2019 年 1 月阿里巴巴 Blink （内部的 Flink 分支版本）开源，大数据领域一夜间从 Spark 独步天下走向了两强争霸的时代。 Flink 因为其天然的流式计算特性以及强大的处理性能成为炙手可热的大数据处理框架。

时至今日，Flink 已经发展到 1.9 版本，在大数据开发领域，面试中对于 Flink 的考察已经是大数据开发求职者必须面对的，本文结合自己作为面试官过程中的经验详细总结了近 50 个关于 Flink 的面试考察点。

在本文中，分为以下几个部分：

第一部分： Flink 中的核心概念和基础篇，包含了 Flink 的整体介绍、核心概念、算子等考察点。

第二部分： Flink 进阶篇，包含了 Flink 中的数据传输、容错机制、序列化、数据热点、反压等实际生产环境中遇到的问题等考察点。

第三部分： Flink 源码篇，包含了 Flink 的核心代码实现、Job 提交流程、数据交换、分布式快照机制、Flink SQL 的原理等考察点。

第一部分：Flink 中的核心概念和基础考察

一、 简单介绍一下 Flink

Flink 是一个框架和分布式处理引擎，用于对无界和有界数据流进行有状态计算。 并且 Flink 提供了数据分布、容错机制以及资源管理等核心功能。

Flink提供了诸多高抽象层的API以便用户编写分布式任务：

DataSet API， 对静态数据进行批处理操作，将静态数据抽象成分布式的数据集，用户可以方便地使用Flink提供的各种操作符对分布式数据集进行处理，支持Java、Scala和Python。

DataStream API，对数据流进行流处理操作，将流式的数据抽象成分布式的数据流，用户可以方便地对分布式数据流进行各种操作，支持Java和Scala。

Table API，对结构化数据进行查询操作，将结构化数据抽象成关系表，并通过类SQL的DSL对关系表进行各种查询操作，支持Java和Scala。

此外，Flink 还针对特定的应用领域提供了领域库，例如： Flink ML，Flink 的机器学习库，提供了机器学习Pipelines API并实现了多种机器学习算法。 Gelly，Flink 的图计算库，提供了图计算的相关API及多种图计算算法实现。

根据官网的介绍，Flink 的特性包含：

支持高吞吐、低延迟、高性能的流处理
支持带有事件时间的窗口 （Window） 操作
支持有状态计算的 Exactly-once 语义
支持高度灵活的窗口 （Window） 操作，支持基于 time、count、session 以及 data-driven 的窗口操作
支持具有 Backpressure 功能的持续流模型
支持基于轻量级分布式快照（Snapshot）实现的容错
一个运行时同时支持 Batch on Streaming 处理和 Streaming 处理
Flink 在 JVM 内部实现了自己的内存管理
支持迭代计算
支持程序自动优化：避免特定情况下 Shuffle、排序等昂贵操作，中间结果有必要进行缓存

二、 Flink 相比传统的 Spark Streaming 有什么区别?

这个问题是一个非常宏观的问题，因为两个框架的不同点非常之多。 但是在面试时有非常重要的一点一定要回答出来： Flink 是标准的实时处理引擎，基于事件驱动。而 Spark Streaming 是微批（Micro-Batch）的模型。

下面我们就分几个方面介绍两个框架的主要区别：

1. 架构模型

Spark Streaming 在运行时的主要角色包括： Master、Worker、Driver、Executor，Flink 在运行时主要包含： Jobmanager、Taskmanager和Slot。

2. 任务调度

Spark Streaming 连续不断的生成微小的数据批次，构建有向无环图DAG，Spark Streaming 会依次创建 DStreamGraph、JobGenerator、JobScheduler。

Flink 根据用户提交的代码生成 StreamGraph，经过优化生成 JobGraph，然后提交给 JobManager进行处理，JobManager 会根据 JobGraph 生成 ExecutionGraph，ExecutionGraph 是 Flink 调度最核心的数据结构，JobManager 根据 ExecutionGraph 对 Job 进行调度。

3. 时间机制

Spark Streaming 支持的时间机制有限，只支持处理时间。 Flink 支持了流处理程序在时间上的三个定义： 处理时间、事件时间、注入时间。 同时也支持 watermark 机制来处理滞后数据。

4. 容错机制

对于 Spark Streaming 任务，我们可以设置 checkpoint，然后假如发生故障并重启，我们可以从上次 checkpoint 之处恢复，但是这个行为只能使得数据不丢失，可能会重复处理，不能做到恰好一次处理语义。

Flink 则使用两阶段提交协议来解决这个问题。

三、 Flink 的组件栈有哪些?

根据 Flink 官网描述，Flink 是一个分层架构的系统，每一层所包含的组件都提供了特定的抽象，用来服务于上层组件。

图片来源于： https://flink.apache.org

自下而上，每一层分别代表： Deploy 层： 该层主要涉及了Flink的部署模式，在上图中我们可以看出，Flink 支持包括local、Standalone、Cluster、Cloud等多种部署模式。 Runtime 层： Runtime层提供了支持 Flink 计算的核心实现，比如： 支持分布式 Stream 处理、JobGraph到ExecutionGraph的映射、调度等等，为上层API层提供基础服务。 API层： API 层主要实现了面向流（Stream）处理和批（Batch）处理API，其中面向流处理对应DataStream API，面向批处理对应DataSet API，后续版本，Flink有计划将DataStream和DataSet API进行统一。 Libraries层： 该层称为Flink应用框架层，根据API层的划分，在API层之上构建的满足特定应用的实现计算框架，也分别对应于面向流处理和面向批处理两类。 面向流处理支持： CEP（复杂事件处理）、基于SQL-like的操作（基于Table的关系操作）； 面向批处理支持： FlinkML（机器学习库）、Gelly（图处理）。

四、Flink 的运行必须依赖 Hadoop组件吗?

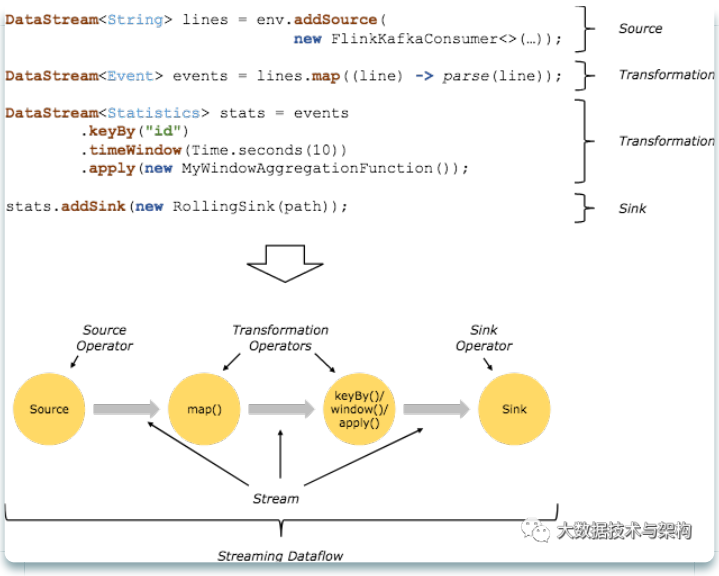
Flink可以完全独立于Hadoop，在不依赖Hadoop组件下运行。 但是做为大数据的基础设施，Hadoop体系是任何大数据框架都绕不过去的。 Flink可以集成众多Hadooop 组件，例如Yarn、Hbase、HDFS等等。 例如，Flink可以和Yarn集成做资源调度，也可以读写HDFS，或者利用HDFS做检查点。

五、你们的Flink集群规模多大?

大家注意，这个问题看起来是问你实际应用中的Flink集群规模，其实还隐藏着另一个问题： Flink可以支持多少节点的集群规模?

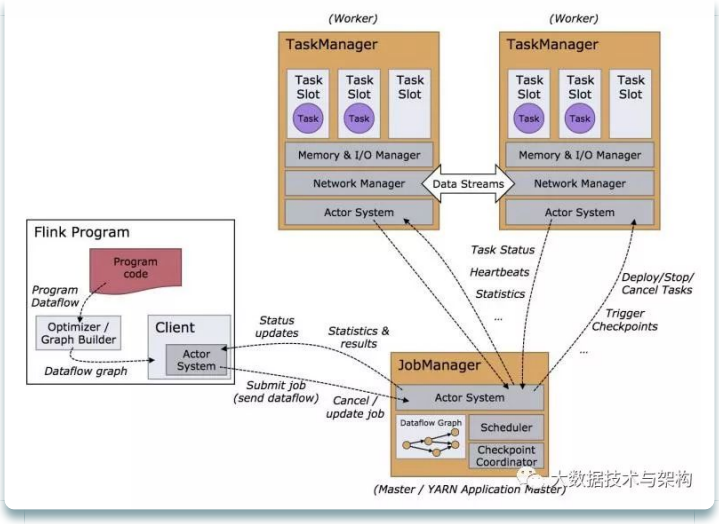
在回答这个问题时候，可以将自己生产环节中的集群规模、节点、内存情况说明，同时说明部署模式（一般是Flink on Yarn），除此之外，用户也可以同时在小集群（少于5个节点）和拥有 TB 级别状态的上千个节点上运行 Flink 任务。

六、Flink的基础编程模型了解吗?



上图是来自Flink官网的运行流程图。 通过上图我们可以得知，Flink 程序的基本构建是数据输入来自一个 Source，Source 代表数据的输入端，经过 Transformation 进行转换，然后在一个或者多个Sink接收器中结束。 数据流（stream）就是一组永远不会停止的数据记录流，而转换（transformation）是将一个或多个流作为输入，并生成一个或多个输出流的操作。 执行时，Flink程序映射到 streaming dataflows，由流（streams）和转换操作（transformation operators）组成。

七、Flink集群有哪些角色？各自有什么作用?

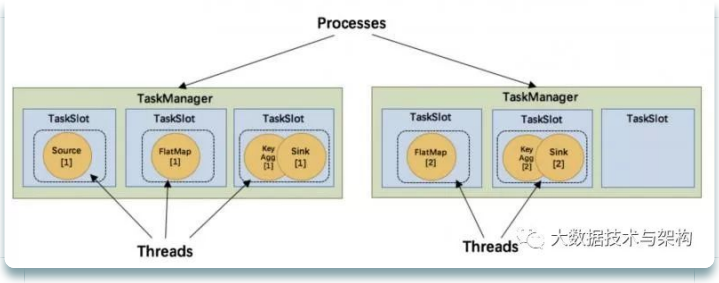


Flink 程序在运行时主要有 TaskManager，JobManager，Client三种角色。 其中JobManager扮演着集群中的管理者Master的角色，它是整个集群的协调者，负责接收Flink Job，协调检查点，Failover 故障恢复等，同时管理Flink集群中从节点TaskManager。

TaskManager是实际负责执行计算的Worker，在其上执行Flink Job的一组Task，每个TaskManager负责管理其所在节点上的资源信息，如内存、磁盘、网络，在启动的时候将资源的状态向JobManager汇报。

Client是Flink程序提交的客户端，当用户提交一个Flink程序时，会首先创建一个Client，该Client首先会对用户提交的Flink程序进行预处理，并提交到Flink集群中处理，所以Client需要从用户提交的Flink程序配置中获取JobManager的地址，并建立到JobManager的连接，将Flink Job提交给JobManager。

八、说说 Flink 资源管理中 Task Slot 的概念



在Flink架构角色中我们提到，TaskManager是实际负责执行计算的Worker，TaskManager 是一个 JVM 进程，并会以独立的线程来执行一个task或多个subtask。 为了控制一个 TaskManager 能接受多少个 task，Flink 提出了 Task Slot 的概念。

简单的说，TaskManager会将自己节点上管理的资源分为不同的Slot： 固定大小的资源子集。 这样就避免了不同Job的Task互相竞争内存资源，但是需要主要的是，Slot只会做内存的隔离。 没有做CPU的隔离。

九、说说 Flink 的常用算子?

Flink 最常用的常用算子包括： Map： DataStream → DataStream，输入一个参数产生一个参数，map的功能是对输入的参数进行转换操作。 Filter： 过滤掉指定条件的数据。 KeyBy： 按照指定的key进行分组。 Reduce： 用来进行结果汇总合并。 Window： 窗口函数，根据某些特性将每个key的数据进行分组（例如： 在5s内到达的数据）

十、说说你知道的Flink分区策略?

什么要搞懂什么是分区策略。 分区策略是用来决定数据如何发送至下游。 目前 Flink 支持了8中分区策略的实现。

上图是整个Flink实现的分区策略继承图：

GlobalPartitioner 数据会被分发到下游算子的第一个实例中进行处理。

ShufflePartitioner 数据会被随机分发到下游算子的每一个实例中进行处理。

RebalancePartitioner 数据会被循环发送到下游的每一个实例中进行处理。

RescalePartitioner 这种分区器会根据上下游算子的并行度，循环的方式输出到下游算子的每个实例。 这里有点难以理解，假设上游并行度为2，编号为A和B。 下游并行度为4，编号为1，2，3，4。 那么A则把数据循环发送给1和2，B则把数据循环发送给3和4。 假设上游并行度为4，编号为A，B，C，D。 下游并行度为2，编号为1，2。 那么A和B则把数据发送给1，C和D则把数据发送给2。

BroadcastPartitioner 广播分区会将上游数据输出到下游算子的每个实例中。 适合于大数据集和小数据集做Jion的场景。

ForwardPartitioner ForwardPartitioner 用于将记录输出到下游本地的算子实例。 它要求上下游算子并行度一样。 简单的说，ForwardPartitioner用来做数据的控制台打印。

KeyGroupStreamPartitioner Hash分区器。 会将数据按 Key 的 Hash 值输出到下游算子实例中。

CustomPartitionerWrapper 用户自定义分区器。 需要用户自己实现Partitioner接口，来定义自己的分区逻辑。 例如：

```
  static classCustomPartitionerimplementsPartitioner<String> {
@Override
publicintpartition(String key, int numPartitions) {
switch (key){
case "1":
return 1;
case "2":
return 2;
case "3":
return 3;
default:
return 4;
}
}
}
```

十一、Flink的并行度了解吗？Flink的并行度设置是怎样的？

Flink中的任务被分为多个并行任务来执行，其中每个并行的实例处理一部分数据。 这些并行实例的数量被称为并行度。

我们在实际生产环境中可以从四个不同层面设置并行度：

操作算子层面(Operator Level)

执行环境层面(Execution Environment Level)

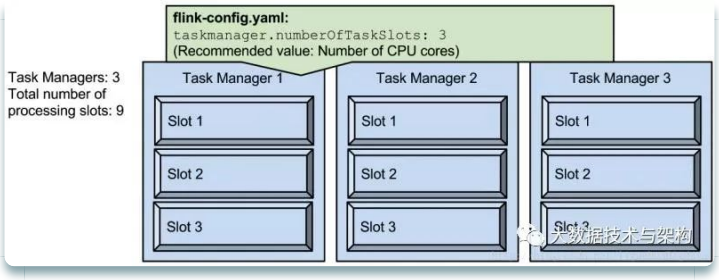
客户端层面(Client Level)

系统层面(System Level)

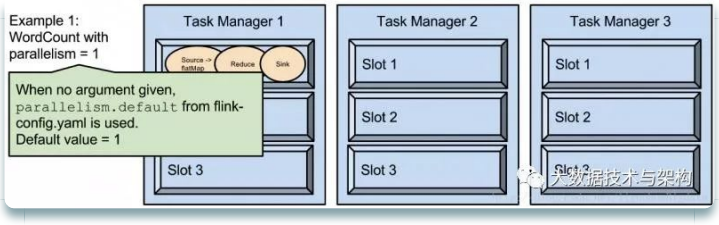
需要注意的优先级： 算子层面>环境层面>客户端层面>系统层面。

十二、Flink的Slot和parallelism有什么区别?

官网上十分经典的图：



slot是指taskmanager的并发执行能力，假设我们将 taskmanager.numberOfTaskSlots 配置为3 那么每一个 taskmanager 中分配3个 TaskSlot, 3个 taskmanager 一共有9个TaskSlot。



parallelism是指taskmanager实际使用的并发能力。 假设我们把 parallelism.default 设置为1，那么9个 TaskSlot 只能用1个，有8个空闲。

十三、Flink有没有重启策略？说说有哪几种?

Flink 实现了多种重启策略。

固定延迟重启策略（Fixed Delay Restart Strategy）

故障率重启策略（Failure Rate Restart Strategy）

没有重启策略（No Restart Strategy）

Fallback重启策略（Fallback Restart Strategy）

十四、用过Flink中的分布式缓存吗？如何使用?

Flink实现的分布式缓存和Hadoop有异曲同工之妙。 目的是在本地读取文件，并把他放在 taskmanager 节点中，防止task重复拉取。

```
    val env = ExecutionEnvironment.getExecutionEnvironment

// register a file from HDFS
env.registerCachedFile("hdfs:///path/to/your/file", "hdfsFile")

// register a local executable file (script, executable, ...)
env.registerCachedFile("file:///path/to/exec/file", "localExecFile", true)

// define your program and execute
...
val input: DataSet[String] = ...
val result: DataSet[Integer] = input.map(new MyMapper())
...
env.execute()
```

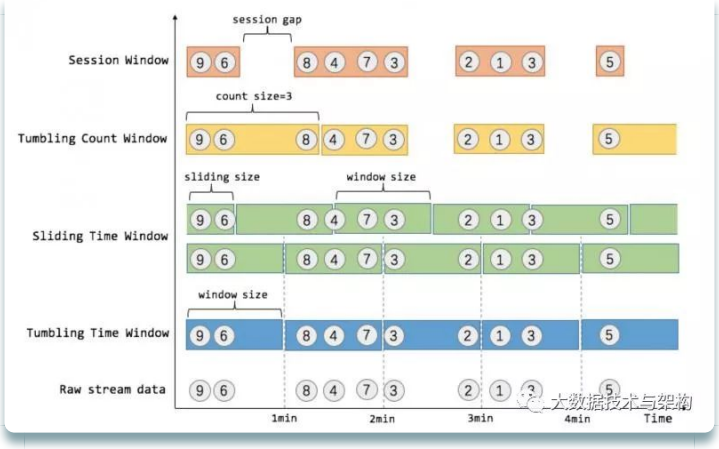
十五、说说Flink中的广播变量，使用时需要注意什么?

我们知道Flink是并行的，计算过程可能不在一个 Slot 中进行，那么有一种情况即： 当我们需要访问同一份数据。 那么Flink中的广播变量就是为了解决这种情况。

我们可以把广播变量理解为是一个公共的共享变量，我们可以把一个dataset 数据集广播出去，然后不同的task在节点上都能够获取到，这个数据在每个节点上只会存在一份。

十六、说说Flink中的窗口?

来一张官网经典的图：



Flink 支持两种划分窗口的方式，按照time和count。 如果根据时间划分窗口，那么它就是一个time-window 如果根据数据划分窗口，那么它就是一个count-window。

flink支持窗口的两个重要属性（size和interval）

如果size=interval,那么就会形成tumbling-window(无重叠数据) 如果size>interval,那么就会形成sliding-window(有重叠数据) 如果size< interval, 那么这种窗口将会丢失数据。 比如每5秒钟，统计过去3秒的通过路口汽车的数据，将会漏掉2秒钟的数据。

通过组合可以得出四种基本窗口：

time-tumbling-window 无重叠数据的时间窗口，设置方式举例：timeWindow(Time.seconds(5))

time-sliding-window 有重叠数据的时间窗口，设置方式举例：timeWindow(Time.seconds(5), Time.seconds(3))

count-tumbling-window无重叠数据的数量窗口，设置方式举例：countWindow(5)

count-sliding-window 有重叠数据的数量窗口，设置方式举例：countWindow(5,3)

十七、说说Flink中的状态存储?

Flink在做计算的过程中经常需要存储中间状态，来避免数据丢失和状态恢复。 选择的状态存储策略不同，会影响状态持久化如何和 checkpoint 交互。

Flink提供了三种状态存储方式： MemoryStateBackend、FsStateBackend、RocksDBStateBackend。

十八、Flink 中的时间有哪几类

Flink 中的时间和其他流式计算系统的时间一样分为三类： 事件时间，摄入时间，处理时间三种。

如果以 EventTime 为基准来定义时间窗口将形成EventTimeWindow,要求消息本身就应该携带EventTime。 如果以 IngesingtTime 为基准来定义时间窗口将形成 IngestingTimeWindow,以 source 的systemTime为准。 如果以 ProcessingTime 基准来定义时间窗口将形成 ProcessingTimeWindow,以 operator 的systemTime 为准。

十九、Flink 中水印是什么概念，起到什么作用?

Watermark 是 Apache Flink 为了处理 EventTime 窗口计算提出的一种机制, 本质上是一种时间戳。 一般来讲Watermark经常和Window一起被用来处理乱序事件。

二十、Flink Table & SQL 熟悉吗？TableEnvironment这个类有什么作用

TableEnvironment是Table API和SQL集成的核心概念。

这个类主要用来：

在内部catalog中注册表

注册外部catalog

执行SQL查询

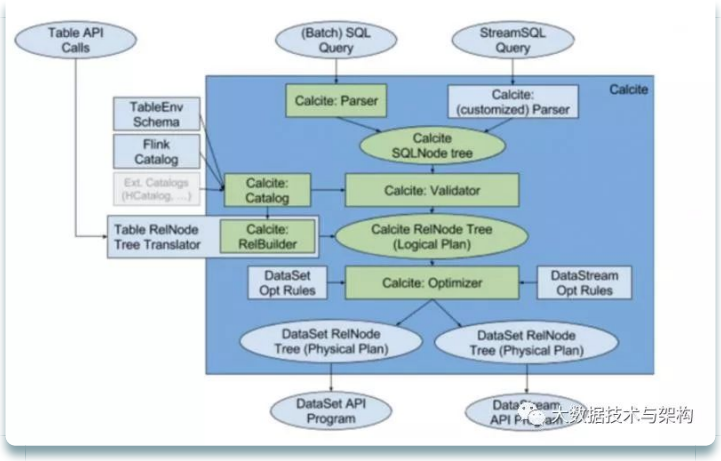
注册用户定义（标量，表或聚合）函数

将DataStream或DataSet转换为表

持有对ExecutionEnvironment或StreamExecutionEnvironment的引用

二十、Flink SQL的实现原理是什么？是如何实现 SQL 解析的呢?

首先大家要知道 Flink 的SQL解析是基于Apache Calcite这个开源框架。



基于此，一次完整的SQL解析过程如下：

用户使用对外提供Stream SQL的语法开发业务应用

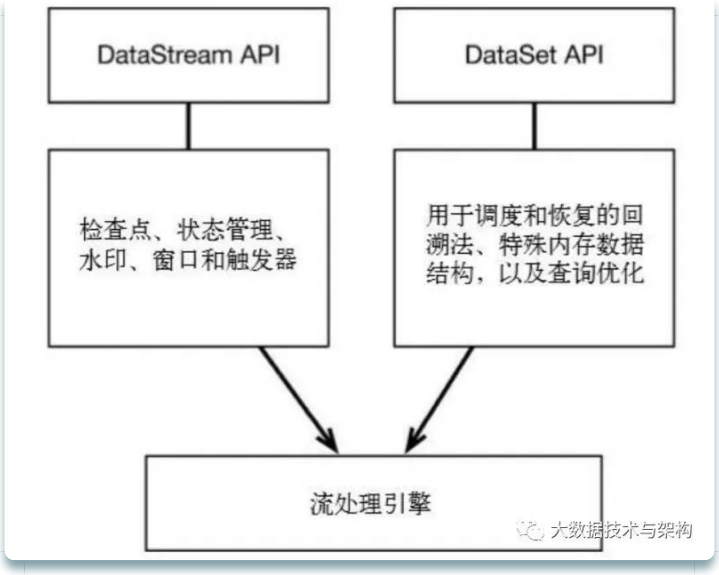
用calcite对StreamSQL进行语法检验，语法检验通过后，转换成calcite的逻辑树节点；最终形成calcite的逻辑计划

采用Flink自定义的优化规则和calcite火山模型、启发式模型共同对逻辑树进行优化，生成最优的Flink物理计划

对物理计划采用janino codegen生成代码，生成用低阶API DataStream 描述的流应用，提交到Flink平台执行

第二部分：Flink 面试进阶篇

一、Flink是如何支持批流一体的?



本道面试题考察的其实就是一句话： Flink的开发者认为批处理是流处理的一种特殊情况。 批处理是有限的流处理。 Flink 使用一个引擎支持了DataSet API 和 DataStream API。

二、Flink是如何做到高效的数据交换的?

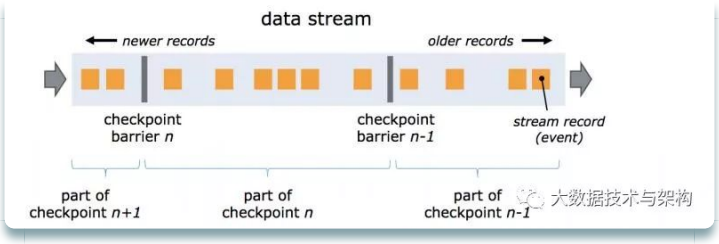
在一个Flink Job中，数据需要在不同的task中进行交换，整个数据交换是有 TaskManager 负责的，TaskManager 的网络组件首先从缓冲buffer中收集records，然后再发送。 Records 并不是一个一个被发送的，二是积累一个批次再发送，batch 技术可以更加高效的利用网络资源。

三、Flink是如何做容错的?

Flink 实现容错主要靠强大的CheckPoint机制和State机制。 Checkpoint 负责定时制作分布式快照、对程序中的状态进行备份； State 用来存储计算过程中的中间状态。

四、Flink 分布式快照的原理是什么?

Flink的分布式快照是根据Chandy-Lamport算法量身定做的。 简单来说就是持续创建分布式数据流及其状态的一致快照。



核心思想是在 input source 端插入 barrier，控制 barrier 的同步来实现 snapshot 的备份和 exactly-once 语义。

五、Flink 是如何保证Exactly-once语义的?

Flink通过实现两阶段提交和状态保存来实现端到端的一致性语义。 分为以下几个步骤：

开始事务（beginTransaction）创建一个临时文件夹，来写把数据写入到这个文件夹里面

预提交（preCommit）将内存中缓存的数据写入文件并关闭

正式提交（commit）将之前写完的临时文件放入目标目录下。这代表着最终的数据会有一些延迟

丢弃（abort）丢弃临时文件

若失败发生在预提交成功后，正式提交前。 可以根据状态来提交预提交的数据，也可删除预提交的数据。

六、Flink 的 kafka 连接器有什么特别的地方?

Flink源码中有一个独立的connector模块，所有的其他connector都依赖于此模块，Flink 在1.9版本发布的全新kafka连接器，摒弃了之前连接不同版本的kafka集群需要依赖不同版本的connector这种做法，只需要依赖一个connector即可。

七、说说 Flink的内存管理是如何做的?

Flink 并不是将大量对象存在堆上，而是将对象都序列化到一个预分配的内存块上。 此外，Flink大量的使用了堆外内存。 如果需要处理的数据超出了内存限制，则会将部分数据存储到硬盘上。 Flink 为了直接操作二进制数据实现了自己的序列化框架。

理论上Flink的内存管理分为三部分：

Network Buffers：这个是在TaskManager启动的时候分配的，这是一组用于缓存网络数据的内存，每个块是32K，默认分配2048个，可以通过“taskmanager.network.numberOfBuffers”修改

Memory Manage pool：大量的Memory Segment块，用于运行时的算法（Sort/Join/Shuffle等），这部分启动的时候就会分配。下面这段代码，根据配置文件中的各种参数来计算内存的分配方法。（heap or off-heap，这个放到下节谈），内存的分配支持预分配和lazy load，默认懒加载的方式。

User Code，这部分是除了Memory Manager之外的内存用于User code和TaskManager本身的数据结构。

八、说说 Flink的序列化如何做的?

Java本身自带的序列化和反序列化的功能，但是辅助信息占用空间比较大，在序列化对象时记录了过多的类信息。

Apache Flink摒弃了Java原生的序列化方法，以独特的方式处理数据类型和序列化，包含自己的类型描述符，泛型类型提取和类型序列化框架。

TypeInformation 是所有类型描述符的基类。 它揭示了该类型的一些基本属性，并且可以生成序列化器。 TypeInformation 支持以下几种类型：

BasicTypeInfo: 任意Java 基本类型或 String 类型

BasicArrayTypeInfo: 任意Java基本类型数组或 String 数组

WritableTypeInfo: 任意 Hadoop Writable 接口的实现类

TupleTypeInfo: 任意的 Flink Tuple 类型(支持Tuple1 to Tuple25)。Flink tuples 是固定长度固定类型的 Java Tuple实现

CaseClassTypeInfo: 任意的 Scala CaseClass(包括 Scala tuples)

PojoTypeInfo: 任意的 POJO (Java or Scala)，例如，Java对象的所有成员变量，要么是 public 修饰符定义，要么有 getter/setter 方法

GenericTypeInfo: 任意无法匹配之前几种类型的类

针对前六种类型数据集，Flink皆可以自动生成对应的TypeSerializer，能非常高效地对数据集进行序列化和反序列化。

九、 Flink中的Window出现了数据倾斜，你有什么解决办法?

window产生数据倾斜指的是数据在不同的窗口内堆积的数据量相差过多。 本质上产生这种情况的原因是数据源头发送的数据量速度不同导致的。 出现这种情况一般通过两种方式来解决:

在数据进入窗口前做预聚合

重新设计窗口聚合的key

十、 Flink中在使用聚合函数 GroupBy、Distinct、KeyBy 等函数时出现数据热点该如何解决?

数据倾斜和数据热点是所有大数据框架绕不过去的问题。 处理这类问题主要从3个方面入手:

在业务上规避这类问题

例如一个假设订单场景，北京和上海两个城市订单量增长几十倍，其余城市的数据量不变。 这时候我们在进行聚合的时候，北京和上海就会出现数据堆积，我们可以单独数据北京和上海的数据。

Key的设计上

把热key进行拆分，比如上个例子中的北京和上海，可以把北京和上海按照地区进行拆分聚合。

参数设置

Flink 1.9.0 SQL(Blink Planner) 性能优化中一项重要的改进就是升级了微批模型，即 MiniBatch。 原理是缓存一定的数据后再触发处理，以减少对State的访问，从而提升吞吐和减少数据的输出量。

十一、Flink任务延迟高，想解决这个问题，你会如何入手?

在Flink的后台任务管理中，我们可以看到Flink的哪个算子和task出现了反压。 最主要的手段是资源调优和算子调优。 资源调优即是对作业中的Operator的并发数（parallelism）、CPU（core）、堆内存（heap_memory）等参数进行调优。 作业参数调优包括： 并行度的设置，State的设置，checkpoint的设置。

十二、Flink是如何处理反压的?

Flink 内部是基于 producer-consumer 模型来进行消息传递的，Flink的反压设计也是基于这个模型。 Flink 使用了高效有界的分布式阻塞队列，就像 Java 通用的阻塞队列（BlockingQueue）一样。 下游消费者消费变慢，上游就会受到阻塞。

十三、Flink的反压和Strom有哪些不同?

Storm 是通过监控 Bolt 中的接收队列负载情况，如果超过高水位值就会将反压信息写到 Zookeeper ，Zookeeper 上的 watch 会通知该拓扑的所有 Worker 都进入反压状态，最后 Spout 停止发送 tuple。

Flink中的反压使用了高效有界的分布式阻塞队列，下游消费变慢会导致发送端阻塞。

二者最大的区别是Flink是逐级反压，而Storm是直接从源头降速。

十四、 Operator Chains（算子链）这个概念你了解吗?

为了更高效地分布式执行，Flink会尽可能地将operator的subtask链接（chain）在一起形成task。 每个task在一个线程中执行。 将operators链接成task是非常有效的优化： 它能减少线程之间的切换，减少消息的序列化/反序列化，减少数据在缓冲区的交换，减少了延迟的同时提高整体的吞吐量。 这就是我们所说的算子链。

十五、 Flink什么情况下才会把Operator chain在一起形成算子链?

两个operator chain在一起的的条件：

上下游的并行度一致

下游节点的入度为1 （也就是说下游节点没有来自其他节点的输入）

上下游节点都在同一个 slot group 中（下面会解释 slot group）

下游节点的 chain 策略为 ALWAYS（可以与上下游链接，map、flatmap、filter等默认是ALWAYS）

上游节点的 chain 策略为 ALWAYS 或 HEAD（只能与下游链接，不能与上游链接，Source默认是HEAD）

两个节点间数据分区方式是 forward（参考理解数据流的分区）

用户没有禁用 chain

十六、 说说Flink1.9的新特性?

支持hive读写，支持UDF

Flink SQL TopN和GroupBy等优化

Checkpoint跟savepoint针对实际业务场景做了优化

Flink state查询

十七、消费kafka数据的时候，如何处理脏数据?

可以在处理前加一个fliter算子，将不符合规则的数据过滤出去。

第三部分：Flink 面试源码篇

一、Flink Job的提交流程 用户提交的Flink Job会被转化成一个DAG任务运行，分别是： StreamGraph、JobGraph、ExecutionGraph，Flink中JobManager与TaskManager，JobManager与Client的交互是基于Akka工具包的，是通过消息驱动。 整个Flink Job的提交还包含着ActorSystem的创建，JobManager的启动，TaskManager的启动和注册。

二、Flink所谓"三层图"结构是哪几个"图"?

一个Flink任务的DAG生成计算图大致经历以下三个过程：

StreamGraph 最接近代码所表达的逻辑层面的计算拓扑结构，按照用户代码的执行顺序向StreamExecutionEnvironment添加StreamTransformation构成流式图。

JobGraph 从StreamGraph生成，将可以串联合并的节点进行合并，设置节点之间的边，安排资源共享slot槽位和放置相关联的节点，上传任务所需的文件，设置检查点配置等。相当于经过部分初始化和优化处理的任务图。

ExecutionGraph 由JobGraph转换而来，包含了任务具体执行所需的内容，是最贴近底层实现的执行图。

三、JobManger在集群中扮演了什么角色?

JobManager 负责整个 Flink 集群任务的调度以及资源的管理，从客户端中获取提交的应用，然后根据集群中 TaskManager 上 TaskSlot 的使用情况，为提交的应用分配相应的 TaskSlot 资源并命令 TaskManager 启动从客户端中获取的应用。

JobManager 相当于整个集群的 Master 节点，且整个集群有且只有一个活跃的 JobManager ，负责整个集群的任务管理和资源管理。

JobManager 和 TaskManager 之间通过 Actor System 进行通信，获取任务执行的情况并通过 Actor System 将应用的任务执行情况发送给客户端。

同时在任务执行的过程中，Flink JobManager 会触发 Checkpoint 操作，每个 TaskManager 节点 收到 Checkpoint 触发指令后，完成 Checkpoint 操作，所有的 Checkpoint 协调过程都是在 Fink JobManager 中完成。

当任务完成后，Flink 会将任务执行的信息反馈给客户端，并且释放掉 TaskManager 中的资源以供下一次提交任务使用。

四、JobManger在集群启动过程中起到什么作用?

JobManager的职责主要是接收Flink作业，调度Task，收集作业状态和管理TaskManager。 它包含一个Actor，并且做如下操作：

RegisterTaskManager: 它由想要注册到JobManager的TaskManager发送。注册成功会通过AcknowledgeRegistration消息进行Ack。

SubmitJob: 由提交作业到系统的Client发送。提交的信息是JobGraph形式的作业描述信息。

CancelJob: 请求取消指定id的作业。成功会返回CancellationSuccess，否则返回CancellationFailure。

UpdateTaskExecutionState: 由TaskManager发送，用来更新执行节点(ExecutionVertex)的状态。成功则返回true，否则返回false。

RequestNextInputSplit: TaskManager上的Task请求下一个输入split，成功则返回NextInputSplit，否则返回null。

JobStatusChanged：它意味着作业的状态(RUNNING, CANCELING, FINISHED,等)发生变化。这个消息由ExecutionGraph发送。

五、TaskManager在集群中扮演了什么角色?

TaskManager 相当于整个集群的 Slave 节点，负责具体的任务执行和对应任务在每个节点上的资源申请和管理。

客户端通过将编写好的 Flink 应用编译打包，提交到 JobManager，然后 JobManager 会根据已注册在 JobManager 中 TaskManager 的资源情况，将任务分配给有资源的 TaskManager节点，然后启动并运行任务。

TaskManager 从 JobManager 接收需要部署的任务，然后使用 Slot 资源启动 Task，建立数据接入的网络连接，接收数据并开始数据处理。 同时 TaskManager 之间的数据交互都是通过数据流的方式进行的。

可以看出，Flink 的任务运行其实是采用多线程的方式，这和 MapReduce 多 JVM 进行的方式有很大的区别，Flink 能够极大提高 CPU 使用效率，在多个任务和 Task 之间通过 TaskSlot 方式共享系统资源，每个 TaskManager 中通过管理多个 TaskSlot 资源池进行对资源进行有效管理。

六、TaskManager在集群启动过程中起到什么作用?

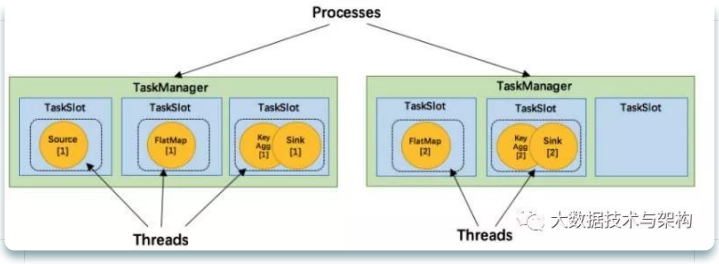
TaskManager的启动流程较为简单： 启动类： org.apache.flink.runtime.taskmanager.TaskManager 核心启动方法 ： selectNetworkInterfaceAndRunTaskManager 启动后直接向JobManager注册自己，注册完成后，进行部分模块的初始化。

七、Flink 计算资源的调度是如何实现的?

TaskManager中最细粒度的资源是Task slot，代表了一个固定大小的资源子集，每个TaskManager会将其所占有的资源平分给它的slot。

通过调整 task slot 的数量，用户可以定义task之间是如何相互隔离的。 每个 TaskManager 有一个slot，也就意味着每个task运行在独立的 JVM 中。 每个 TaskManager 有多个slot的话，也就是说多个task运行在同一个JVM中。

而在同一个JVM进程中的task，可以共享TCP连接（基于多路复用）和心跳消息，可以减少数据的网络传输，也能共享一些数据结构，一定程度上减少了每个task的消耗。 每个slot可以接受单个task，也可以接受多个连续task组成的pipeline，如下图所示，FlatMap函数占用一个taskslot，而key Agg函数和sink函数共用一个taskslot：



八、简述Flink的数据抽象及数据交换过程?

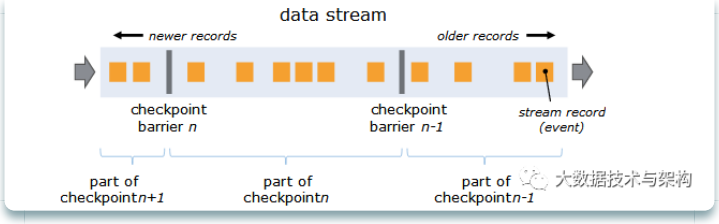
Flink 为了避免JVM的固有缺陷例如java对象存储密度低，FGC影响吞吐和响应等，实现了自主管理内存。 MemorySegment就是Flink的内存抽象。 默认情况下，一个MemorySegment可以被看做是一个32kb大的内存块的抽象。 这块内存既可以是JVM里的一个byte[]，也可以是堆外内存（DirectByteBuffer）。

在MemorySegment这个抽象之上，Flink在数据从operator内的数据对象在向TaskManager上转移，预备被发给下个节点的过程中，使用的抽象或者说内存对象是Buffer。

对接从Java对象转为Buffer的中间对象是另一个抽象StreamRecord。

九、Flink 中的分布式快照机制是如何实现的?

Flink的容错机制的核心部分是制作分布式数据流和操作算子状态的一致性快照。 这些快照充当一致性checkpoint，系统可以在发生故障时回滚。 Flink用于制作这些快照的机制在“分布式数据流的轻量级异步快照”中进行了描述。 它受到分布式快照的标准Chandy–Lamport算法的启发，专门针对Flink的执行模型而定制。



barriers在数据流源处被注入并行数据流中。 快照n的barriers被插入的位置（我们称之为Sn）是快照所包含的数据在数据源中最大位置。 例如，在Apache Kafka中，此位置将是分区中最后一条记录的偏移量。 将该位置Sn报告给checkpoint协调器（Flink的JobManager）。

然后barriers向下游流动。 当一个中间操作算子从其所有输入流中收到快照n的barriers时，它会为快照n发出barriers进入其所有输出流中。 一旦sink操作算子（流式DAG的末端）从其所有输入流接收到barriers n，它就向checkpoint协调器确认快照n完成。 在所有sink确认快照后，意味快照着已完成。

一旦完成快照n，job将永远不再向数据源请求Sn之前的记录，因为此时这些记录（及其后续记录）将已经通过整个数据流拓扑，也即是已经被处理结束。

十、简单说说FlinkSQL的是如何实现的?

Flink 将 SQL 校验、SQL 解析以及 SQL 优化交给了Apache Calcite。 Calcite 在其他很多开源项目里也都应用到了，譬如 Apache Hive, Apache Drill, Apache Kylin, Cascading。 Calcite 在新的架构中处于核心的地位，如下图所示。

构建抽象语法树的事情交给了 Calcite 去做。 SQL query 会经过 Calcite 解析器转变成 SQL 节点树，通过验证后构建成 Calcite 的抽象语法树（也就是图中的 Logical Plan）。 另一边，Table API 上的调用会构建成 Table API 的抽象语法树，并通过 Calcite 提供的 RelBuilder 转变成 Calcite 的抽象语法树。 然后依次被转换成逻辑执行计划和物理执行计划。

在提交任务后会分发到各个 TaskManager 中运行，在运行时会使用 Janino 编译器编译代码后运行。