## Flink是什么

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。

## Flink相比传统的Spark Streaming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。

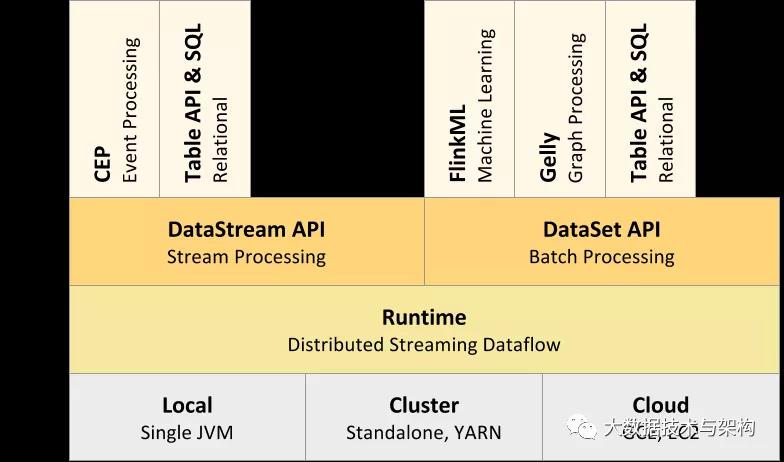
Flink 根据用户提交的代码生成 StreamGraph，经过优化生成 JobGraph，然后提交给 JobManager进行处理，JobManager 会根据 JobGraph 生成 ExecutionGraph，ExecutionGraph 是 Flink 调度最核心的数据结构，JobManager 根据 ExecutionGraph 对 Job 进行调度。

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。

## 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。



自下而上，每一层分别代表：

1. 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。
2. 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。
3. 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。
4. 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）

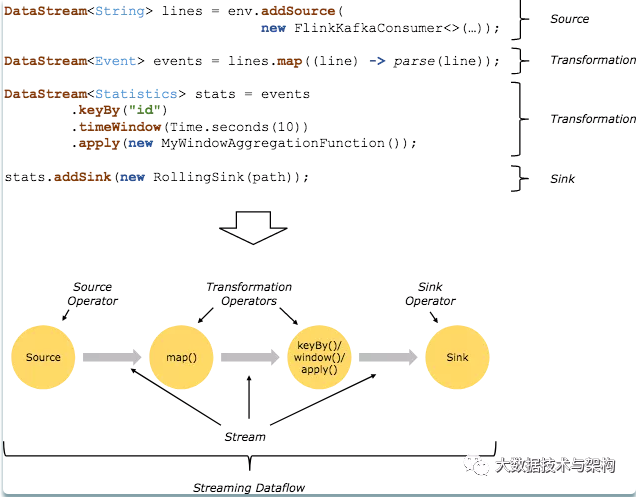
## Flink 的运行必须依赖 Hadoop组件吗？

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Hadoop 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。

## 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。

## Flink的基础编程模型了解吗？



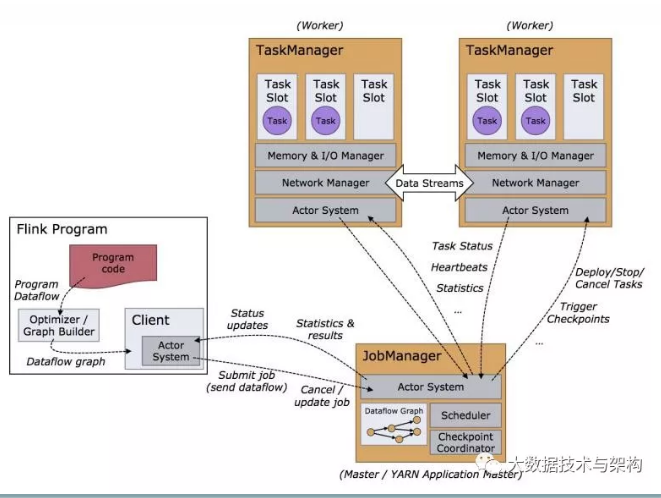
上图是来自Flink官网的运行流程图。

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。

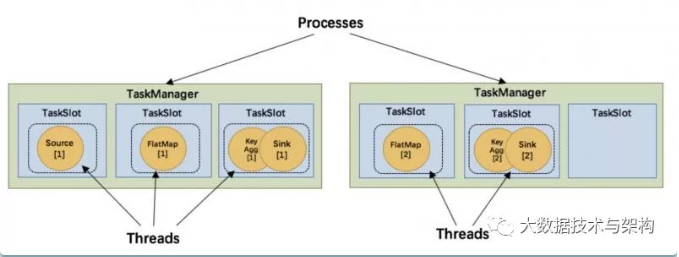
## Flink集群有哪些角色？各自有什么作用？



Flink 程序在运行时主要有 TaskManager，JobManager，Client三种角色。

1. 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。
2. 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。
3. 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。

## 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。

## 说说 Flink 的常用算子

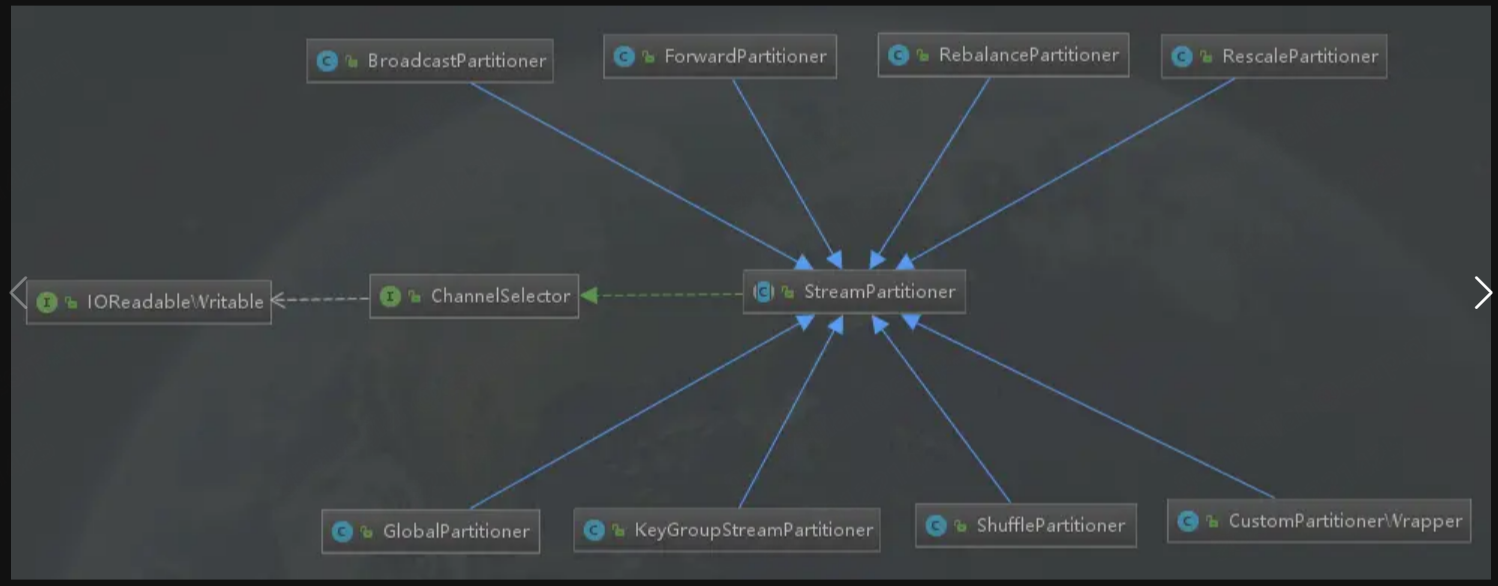
Flink 最常用的算子包括：

1. 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。
2. Filter：过滤掉指定条件的数据。
3. KeyBy：按照指定的key进行分组。
4. Reduce：用来进行结果汇总合并。
5. 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）

## 说说你知道的Flink分区策略

首先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：

1. **GlobalPartitioner** 数据会被分发到下游算子的第一个实例中进行处理。
2. **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。
3. **RebalancePartitioner** 数据会被循环发送到下游的每一个实例中进行处理。
4. **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。
5. **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。
6. **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。
7. **KeyGroupStreamPartitioner** Hash分区器。会将数据按 Key 的 Hash 值输出到下游算子实例中。
8. **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;

}

}

}

## 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)

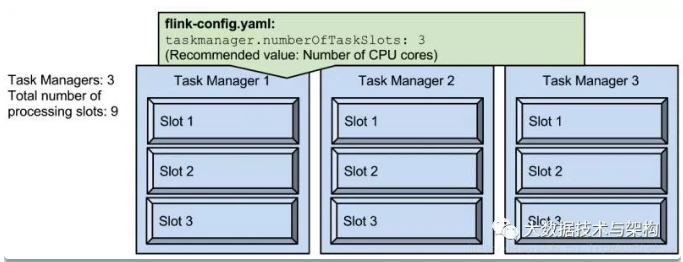
客户端层面(Client Level)

系统层面(System Level)

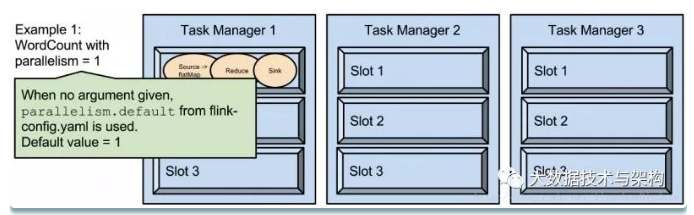
需要注意的优先级：算子层面>环境层面>客户端层面>系统层面。

## 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。

## ？？？？？？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）

## ？？？？？？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()

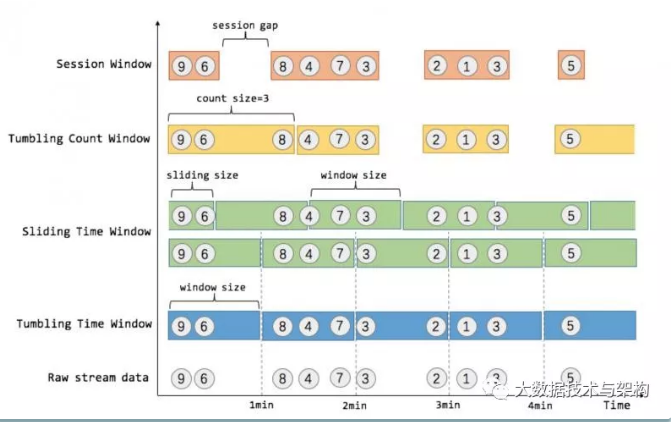
## 说说Flink中的广播变量，使用时需要注意什么？

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。

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。

## ？？？？？？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)

此处增加每个窗口具体的应用场景

## 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。

## 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。

## 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。

## 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

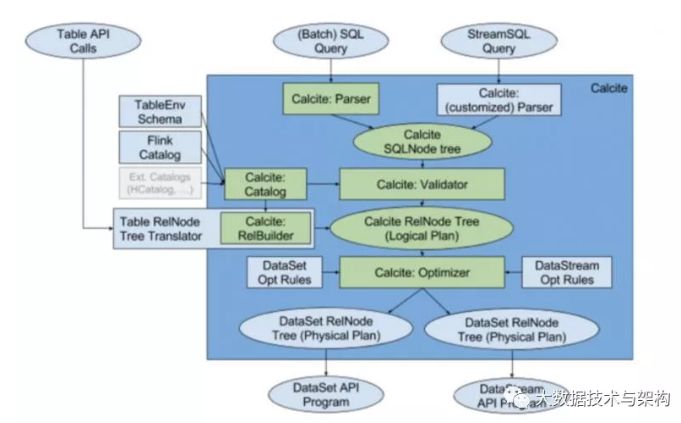
注册用户定义（标量，表或聚合）函数

将DataStream或DataSet转换为表

持有对ExecutionEnvironment或StreamExecutionEnvironment的引用

## 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：

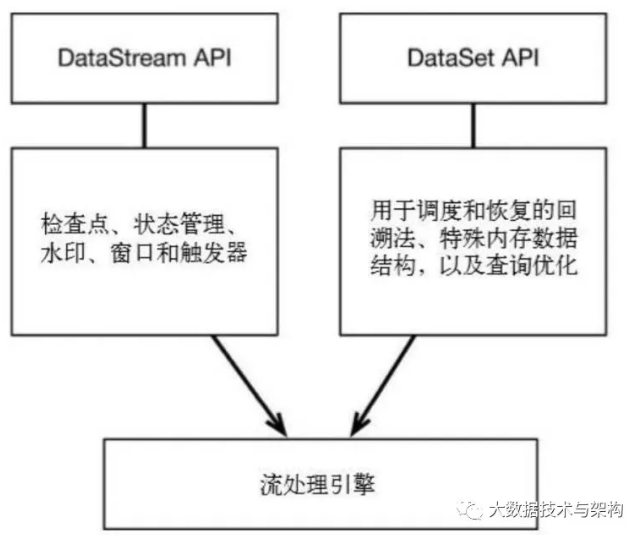
1. 用户使用对外提供Stream SQL的语法开发业务应用
2. 用calcite对StreamSQL进行语法检验，语法检验通过后，转换成calcite的逻辑树节点

3、最终形成calcite的逻辑计划

4、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

5、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

## 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。

## 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，整个数据交换是由 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，而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。

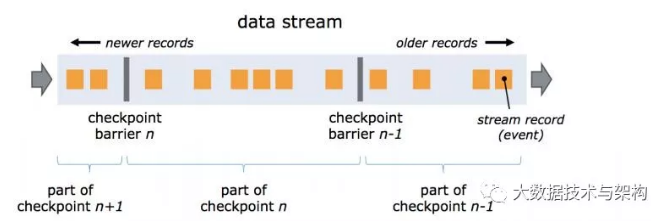
## Flink是如何做容错的？

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。

## 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。

## 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：

1. 开始事务（beginTransaction）创建一个临时文件夹，先把数据写入到这个文件夹里面
2. 预提交（preCommit）将内存中缓存的数据写入文件并关闭
3. 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
4. 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。

## 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。

## 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：

1. 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
2. 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。内存的分配支持预分配和lazy load，默认懒加载的方式。
3. 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。

## 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：

1. BasicTypeInfo: 任意Java 基本类型或 String 类型
2. BasicArrayTypeInfo: 任意Java基本类型数组或 String 数组
3. WritableTypeInfo: 任意 Hadoop Writable 接口的实现类
4. 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
5. CaseClassTypeInfo: 任意的 Scala CaseClass(包括 Scala tuples)
6. 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
7. 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。

## 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：

1. 在数据进入窗口前做预聚合
2. 重新设计窗口聚合的key

## 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：

1、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，我们可以单独处理北京和上海的数据。

2、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。

1. 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。

## 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。

## ？？？？？？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。

## 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。

## 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。

## 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：

1. 上下游的并行度一致
2. 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）
3. 上下游节点都在同一个 slot group 中（下面会解释 slot group）
4. 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）
5. 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）
6. 两个节点间数据分区方式是 forward（参考理解数据流的分区）
7. 用户没有禁用 chain