|  |  |
| --- | --- |
| **问题** | **答案** |
| Flink有哪几种部署模式，每种模式特点 | **Local模式**：  适用于本地开发和测试环境，占用的资源较少，部署简单 ，只需要部署[JDK](https://www.baidu.com/s?wd=JDK&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)和flink即可达到功能开发和测试的目的。只需要一台主机即可。  **Standalone模式**  分布式部署集群， 自带完整的服务，资源管理和任务监控是Flink自己监控，这个模式也是其他模式的基础  **Flink on Yarn模式**  分布式部署集群，资源和任务监控交给yarn管理，目前仅支持粗粒度资源分配方式，包含cluster和client运行模式，cluster适合生产,job client运行在集群子节点，具有容错功能，client适合调试，job client运行在客户端 |
| Flink技术栈有哪些组件，每个组件都有什么功能 | Flink 分别提供了面向流处理的接口（DataStream API）和面向批处理的接口（DataSet API），支持的拓展库涉及机器学习（FlinkML）、复杂事件处理（CEP），以及图计算（Gelly），还有分别针对流处理和批处理的 Table API&SQL |
| Flink都有哪些特点 | * Streaming-first、流处理引擎。 * Fault-tolerant，容错，可靠性，checkpoint。 * Scalable，可扩展性，1000节点以上。 * Performance，性能，高吞吐量， 低延迟。 * 低延时：提供ms级时延的处理能力。 * Exactly Once：提供异步快照机制，保证所有数据真正处理一次。 * HA：JobManager支持主备模式，保证无单点故障。 * 水平扩展能力：TaskManager支持手动水平扩展。 |
| Flink系统架构 | Flink 集群启动后，首先会启动一个 JobManger 和多个的 TaskManager。用户的代码会由JobClient 提交给 JobManager，JobManager 再把来自不同用户的任务发给 不同的TaskManager 去执行，每个TaskManager管理着多个task，task是执行计算的最小结构， TaskManager 将心跳和统计信息汇报给 JobManager。TaskManager 之间以流的形式进行数据的传输。上述除了task外的三者均为独立的 JVM 进程。  JobClient：需求提出方，负责提交应用，构造流图，客户端不是运行时和程序执行的一部分,但用于向JobManage发送数据流。客户端可以断开,或者保持联系接收Job进度报告。  JobManager负责接收 flink 的作业，调度 task，收集 job 的状态、管理 TaskManagers，协调检查点，失败恢复等等。集群中至少需要有一个Job Managers，作为leader，其他备用。  TaskManager是flink中资源管理的基本组件，负责实际计算，一个应用会拆给多个TaskManager来进行计算，是所有执行任务的基本容器，它提供了内存管理、IO管理、通信管理等一系列功能  TaskSlot：任务槽，类似于Yarn当中的Container，用于资源的封装。但是在Flink中，taskSlot只负责封装内存的资源，不包含CPU的资源。每一个TaskManager中会默认包含3个TaskSlot，所以每一个TaskManager中最多能并发执行的任务是可控的，最多3个。TaskSlot有独占的内存资源，在一个TaskManager中可以运行不同的任务。 |
| Flink有哪几种时间模型 | * EventTime是数据被生产出来的时间，可以是比如传感器发出信号的时间等（此时数据还没有被传输给flink）。 * IngestionTime是数据进入flink的时间，也就是从Source进入flink流的时间（此时数据刚刚被传给flink） * ProcessingTime是针对当前算子的系统时间，是指该数据已经进入某个operator时，operator所在系统的当前时间 |
| Flink作业提交流程，client和 cluster 有什么区别，各有什么作用 | 两者主要区别就是JobClient的运行位置，在客户端模式下，JobClient运行在提交作业的客户端机器上负责与集群进行资源申请调度等工作。而集群模式下JobClient运行在集群中的某一个节点上负责资源申请以及调度。 |
| Flink 有哪几种窗口，分别特点是什么 | Tumbing Windows：滚动窗口，窗口之间时间点不重叠。它是按照固定的时间，或固定的事件个数划分的，分别可以叫做滚动时间窗口和滚动事件窗口。  Sliding Windows：滑动窗口，窗口之间时间点存在重叠。对于某些应用，它们需要的时间是不间断的，需要平滑的进行窗口聚合。例如，可以每30s记算一次最近1分钟用户所购买的商品数量的总数，这个就是时间滑动窗口；或者每10个客户点击购买，然后就计算一下最近100个客户购买的商品的总和，这个就是事件滑动窗口。  Session Windows：会话窗口，经过一段设置时间无数据认为窗口完成。 |
| 描述Flink Checkpoint机制（分布式异步快照算法） | 分布式异步快照算法是一种轻量级的快照技术，能以低成本备份 DAG（有向无环图）或 DCG（有向有环图）计算作业的状态，这使得计算作业可以频繁进行快照并且不会对性能产生明显影响。分布式异步快照算法的核心思想是通过屏障消息（barrier）来标记触发快照的时间点和对应的数据，从而将数据流和快照时间解耦以实现异步快照操作，同时也大大降低了对管道数据的依赖（对 DAG 类作业甚至完全不依赖），减小了随之而来的快照大小。  **流程**   1. CheckpointCoordinator周期性的向该流应用，所有的source算子发送barrier。 2. Source算子接收到一个barrier后，便暂停处理数据，将当前的状态制作成快照，并保存到指定的持久化存储中，最后它再向CheckpointCoordinator报告自己的快照制作情况。同时向自身下游所有算子广播该barrier。然后恢复该算子的数据处理工作。 3. 下游的算子接收到barrier后，也会暂停自的数据处理过程，同2过程 4. CheckpointCoordinator会确认它接收到的报告，如果收到本周期的所有算子的快照就认为快照制作成功，否则失败。 |
| 描述Flink State管理 | 计算任务的结果不仅仅依赖于输入，还依赖于它的当前状态，其实大多数的计算都是有状态的计算。  **Flink按照数据的划分和扩张方式,有两种基础的状态：**  KeyedStream上的状态，这个状态是跟特定的key绑定的，对KeyedStream流上的每一个key，可能都对应一个state。  Operator State：Operator State跟一个特定operator的一个并发实例绑定，整个operator只对应一个state。相比较而言，在一个operator上，可能会有很多个key，从而对应多个keyed state。  **提供了三种状态存储后端：**  Filesystem State Backend：数据保存在taskmanager的内存中，执行checkpoint的时候，会把state的快照数据保存到配置的文件系统中，可以使用hdfs等分布式文件系统  RocksDB State Backend：跟上面的都略有不同，它会在本地文件系统中维护状态，state会直接写入本地rocksdb中。同时RocksDB需要配置一个远端的filesystem。  MemoryStateBackend：state数据保存在java堆内存中，执行checkpoint的时候，会把state的快照数据保存到jobmanager的内存中 |
| Flink怎么做到Extract Once | 在发生故障时，Flink 通过从检查点加载应用程序状态来恢复，并从恢复的读取位点继续处理，就好像什么事情都没发生一样。Flink的状态存储在Flink的内部,这样做的好处就是不再依赖外部系统,降低了对外部系统的依赖,在Flink的内部,通过自身的进程去访问状态变量.同时会定期的做checkpoint持久化,把checkpoint存储在一个分布式的持久化系统中,如果发生故障,就会从最近的一次checkpoint中将整个流的状态进行恢复. |
| Flink支持的数据类型 | 基础类型：所有java的基础类型  数组：基础类型构成的数组以及Object[]  复合类型:Flink Java Tuple,Scala case class ,Row,POJO  辅助类型：Option，Either,Lists,Maps  泛型和其他类:Kryo提供序列化支持 |
| Flink有哪些常用的数据交换方式 | Forward：直接转发  Broadcast：广播  Key-based：按key分发  Random：随机分发 |
| Flink作业执行流程 | 在JobClient端生成流图，然后生成Job图，提交到JobManager，生成执行任务图，最终分发到各TaskManager  StreamGraph：根据用户通过Stream API编写的代码生成的最初的图，用来表示程序的拓扑结构。   JobGraph：StreamGraph经过算子连接等优化后生成的图，它是提交给JobManager的数据结构。   ExecutionGraph：JobManager根据JobGraph生成的分布式执行图，是调度层最核心的数据结构。 |
| 什么是flink的 Operator Chains | Flink为了更高效地分布式执行，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。 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   用户没有禁用 chain |
| Flink中Watermark功能及生成 | **功能**  watermark是一种衡量Event Time进展的机制，它是数据本身的一个隐藏属性。通常基于Event Time的数据，Watermarks随着时间不断变化，一旦Watermarks大于了某个window的end\_time，就会触发此window的计算，Watermarks就是用来触发window计算的  **Watermark生成**  生成时间：  可以在接收到source的数据后，立刻生成watermark；  也可以在source后，应用简单的map或者filter操作，然后再生成watermark  怎样生成：  生成watermark的方式主要有2大类  With Periodic Watermarks：定期分配时间戳并生成watermarks  With Punctuated Watermarks：带有标记的watermark |
| Flink的调度机制 | **逻辑调度**  **JobClient**将用户提交的Flink程序组装一个JobGraph， 以JobGraph的形式提交。一个JobGraph是一个Flink Dataflow，它由多个JobVertex组成的DAG。其中，一个JobGraph包含了一个Flink程序的如下信息：JobID、Job名称、配置信息、一组JobVertex等  JobManager接收到Client提交的JobGraph形式的Flink Job，JobManager会将一个JobGraph转换映射为一个ExecutionGraph    **物理调度**  Flink 通过任务槽（Task slot）定义执行资源，每个 TaskManager 都有一或多个任务槽，每个任务槽都可以运行一个并行任务流，一个 pipeline 包括多个连续的任务，例如一个 MapFunction 的第n个并行实例与一个 ReduceFunction 的第n个并行实例的连续任务。Flink 通常会并发执行连续的任务，对于流式程序来说，任何情况都如此执行；而对于批处理程序，多数情况也如此执行。 |
| Flink内存模型 | **堆内存结构：**   **Network Buffers:**一定数量的32KB大小的缓存，主要用于数据的网络传输。在 TaskManager 启动的时候就会分配。默认数量是 2048 个   **Memory Manager Pool:** 这是一个由 MemoryManager 管理的，由众多MemorySegment组成的超大集合。Flink 中的算法（如 sort/shuffle/join）会向这个内存池申请 MemorySegment，将序列化后的数据存于其中，使用完后释放回内存池。默认情况下，池子占了堆内存的 70% 的大小。    **Remaining (Free) Heap:** 这部分的内存是留给用户代码以及 TaskManager 的数据结构使用的，可以把这里看成的新生代  **好处：**  **减少GC压力**：因为所有常驻型数据都以二进制的形式存在 Flink 的MemoryManager中，这些MemorySegment一直呆在老年代而不会被GC回收。其他的数据对象基本上是由用户代码生成的短生命周期对象，这部分对象可以被 Minor GC 快速回收。  **避免了OOM：**有的运行时数据结构和算法只能通过内存池申请内存，保证了其使用的内存大小是固定的，不会因为运行时数据结构和算法而发生OOM。在内存吃紧的情况下，算法（sort/join等）会高效地将一大批内存块写到磁盘，之后再读回来  **节省内存空间**：Java对象在存储上有很多额外的消耗，如果只存储实际数据的二进制内容，就可以避免这部分消耗  **高效的二进制操作 & 缓存友好的计算**：二进制数据以定义好的格式存储，可以高效地比较与操作 |
| Flink DataSteam流编程步骤 | 1、 获取一个execution environment 2、 拉取或者创建一个初始数据集 3、 指定数据集的转换操作 4、 指定计算结果保存在哪 5、 触发程序执行 |
| Flink DataStream API 常用算子 | Map：dataStream.map { x => x \* 2 }  FlatMap：dataStream.flatMap { str => str.split(" ") }  Filter：dataStream.filter { \_ != 0 }  KeyBy：dataStream.keyBy("someKey")  Reduce：keyedStream.reduce { \_ + \_ }  Min/Max/Sum：keyedStream.sum(0)  Window：dataStream.keyBy(0).window(TumblingEventTimeWindows.of(Time.seconds(5))); |
| Flink并行度怎么设置 | 操作算子层面：.sum(1).setParallelism(5)  执行环境层面：env.setParallelism(3)  客户端层面：./bin/flink run -p 10 ../examples/\*WordCount-java\*.jar  系统层面：parallelism.default，属性在conf/flink-conf.yaml文件中设置  资源层面：设置[slots](https://www.baidu.com/s?wd=slots&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd) |
| Flink性能优化 | **开发优化：**   1. 使用flink tuples 2. 复用flink对象 3. 使用注解功能 4. 选择合适的join类型 5. 优化数据结构 6. 合理设计分区，可以优化task的切分   **配置调优：**   1. **资源调优**：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 2. **Checkpoint调优**：设置合适的Checkpoint间隔时间 3. **状态后端调优**：选择合适的状态后端存储 4. **GC优化** ：设置合适的GC参数 5. **JobManager/TaskManger内存：提高内存，以提高其运行效率** 6. **开启Job最大允许延迟** 7. **缓存区超时调优**：由于task在执行过程中存在数据通过网络进行交换，数据在不同服务器之间传递的缓冲区超时时间可以通过setBufferTimeout进行设置   **数据倾斜优化：**   1. 重新设计key，以更小力度key使得task大小更加合理 2. 修改并行度   3． 调用rebalance操作，使数据更加均匀 |