# **Hadoop**

hadoop中常问的有三块，第一：存储，问到存储，就把HDFS相关的知识点拿出来；第二：计算框架(MapReduce)；第三：资源调度框架(yarn)

1. **请说下HDFS读写流程**

HDFS写流程

1. client客户端发送上传请求，通过RPC与namenode建立通信，namenode检查该用户是否有上传权限，以及上传的文件是否在hdfs对应的目录下重名，如果这两者有任意一个不满足，则直接报错，如果两者都满足，则返回给客户端一个可以上传的信息

2）client根据文件的大小进行切分，默认128M一块，切分完成之后给namenode发送请求第一个block块上传到哪些服务器上

3）namenode收到请求之后，根据网络拓扑和机架感知以及副本机制进行文件分配，返回可用的DataNode的地址

注：Hadoop 在设计时考虑到数据的安全与高效, 数据文件默认在 HDFS 上存放三份, 存储策略为本地一份，同机架内其它某一节点上一份, 不同机架的某一节点上一份

4）客户端收到地址之后与服务器地址列表中的一个节点如A进行通信，本质上就是RPC调用，建立pipeline，A收到请求后会继续调用B，B在调用C，将整个pipeline建立完成，逐级返回client

5）client开始向A上发送第一个block（先从磁盘读取数据然后放到本地内存缓存），以packet（数据包，64kb）为单位，A收到一个packet就会发送给B，然后B发送给C，A每传完一个packet就会放入一个应答队列等待应答

6）数据被分割成一个个的packet数据包在pipeline上依次传输，在pipeline反向传输中，逐个发送ack（命令正确应答），最终由pipeline 中第一个 DataNode 节点 A 将 pipelineack 发送给 Client

7）当一个 block 传输完成之后, Client 再次请求 NameNode 上传第二个 block ，namenode重新选择三台DataNode给client

HDFS读流程

1. client向namenode发送RPC请求。请求文件block的位置

2）namenode收到请求之后会检查用户权限以及是否有这个文件，如果都符合，则会视情况返回部分或全部的block列表，对于每个block，NameNode 都会返回含有该 block 副本的 DataNode 地址； 这些返回的 DN 地址，会按照集群拓扑结构得出 DataNode 与客户端的距离，然后进行排序，排序两个规则：网络拓扑结构中距离 Client 近的排靠前；心跳机制中超时汇报的 DN 状态为 STALE，这样的排靠后

3）Client 选取排序靠前的 DataNode 来读取 block，如果客户端本身就是DataNode,那么将从本地直接获取数据(短路读取特性)

4）底层上本质是建立 Socket Stream（FSDataInputStream），重复的调用父类 DataInputStream 的 read 方法，直到这个块上的数据读取完毕

5）当读完列表的 block 后，若文件读取还没有结束，客户端会继续向NameNode 获取下一批的 block 列表

6）读取完一个 block 都会进行 checksum 验证，如果读取 DataNode 时出现错误，客户端会通知 NameNode，然后再从下一个拥有该 block 副本的DataNode 继续读

7）read 方法是并行的读取 block 信息，不是一块一块的读取；NameNode 只是返回Client请求包含块的DataNode地址，并不是返回请求块的数据

8） 最终读取来所有的 block 会合并成一个完整的最终文件

2. HDFS在读取文件的时候,如果其中一个块突然损坏了怎么办

客户端读取完DataNode上的块之后会进行checksum 验证，也就是把客户端读取到本地的块与HDFS上的原始块进行校验，如果发现校验结果不一致，客户端会通知 NameNode，然后再从下一个拥有该 block 副本的DataNode 继续读

3. HDFS在上传文件的时候,如果其中一个DataNode突然挂掉了怎么办客户端上传文件时与DataNode建立pipeline管道，管道正向是客户端向DataNode发送的数据包，管道反向是DataNode向客户端发送ack确认，也就是正确接收到数据包之后发送一个已确认接收到的应答，当DataNode突然挂掉了，客户端接收不到这个DataNode发送的ack确认

，客户端会通知 NameNode，NameNode检查该块的副本与规定的不符，NameNode会通知DataNode去复制副本，并将挂掉的DataNode作下线处理，不再让它参与文件上传与下载。

4. NameNode在启动的时候会做哪些操作

NameNode数据存储在内存和本地磁盘，本地磁盘数据存储在fsimage镜像文件和edits编辑日志文件

首次启动NameNode

1. 格式化文件系统，为了生成fsimage镜像文件

2、启动NameNode

（1）读取fsimage文件，将文件内容加载进内存

（2）等待DataNade注册与发送block report

3、启动DataNode

（1）向NameNode注册

（2）发送block report

（3）检查fsimage中记录的块的数量和block report中的块的总数是否相同

4、对文件系统进行操作（创建目录，上传文件，删除文件等）

（1）此时内存中已经有文件系统改变的信息，但是磁盘中没有文件系统改变的信息，此时会将这些改变信息写入edits文件中，edits文件中存储的是文件系统元数据改变的信息。

第二次启动NameNode

1、读取fsimage和edits文件

2、将fsimage和edits文件合并成新的fsimage文件

3、创建新的edits文件，内容为空

4、启动DataNode

5. Secondary NameNode了解吗，它的工作机制是怎样的

Secondary NameNode 是合并NameNode的edit logs到fsimage文件中；

它的具体工作机制：

（1）Secondary NameNode询问NameNode是否需要checkpoint。直接带回NameNode是否检查结果

（2）Secondary NameNode请求执行checkpoint

（3）NameNode滚动正在写的edits日志

（4）将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode

（5）Secondary NameNode加载编辑日志和镜像文件到内存，并合并

（6）生成新的镜像文件fsimage.chkpoint

（7）拷贝fsimage.chkpoint到NameNode

（8）NameNode将fsimage.chkpoint重新命名成fsimage

所以如果NameNode中的元数据丢失，是可以从Secondary NameNode恢复一部分元数据信息的，但不是全部，因为NameNode正在写的edits日志还没有拷贝到Secondary NameNode，这部分恢复不了

6. Secondary NameNode不能恢复NameNode的全部数据，那如何保证NameNode数据存储安全

这个问题就要说NameNode的高可用了，即 NameNode HA

一个NameNode有单点故障的问题，那就配置双NameNode，配置有两个关键点，一是必须要保证这两个NN的元数据信息必须要同步的，二是一个NN挂掉之后另一个要立马补上。

元数据信息同步在 HA 方案中采用的是“共享存储”。每次写文件时，需要将日志同步写入共享存储，这个步骤成功才能认定写文件成功。然后备份节点定期从共享存储同步日志，以便进行主备切换。

监控NN状态采用 zookeeper，两个NN节点的状态存放在ZK中，另外两个NN节点分别有一个进程监控程序，实施读取ZK中有NN的状态，来判断当前的NN是不是已经down机。如果standby的NN节点的ZKFC发现主节点已经挂掉，那么就会强制给原本的active NN节点发送强制关闭请求，之后将备用的NN设置为active。

如果面试官再问HA中的 共享存储 是怎么实现的知道吗？

可以进行解释下：NameNode 共享存储方案有很多，比如 Linux HA, VMware FT, QJM等，目前社区已经把由 Clouderea 公司实现的基于 QJM（Quorum Journal Manager）的方案合并到 HDFS 的 trunk 之中并且作为默认的共享存储实现

基于 QJM 的共享存储系统主要用于保存 EditLog，并不保存 FSImage 文件。FSImage 文件还是在 NameNode 的本地磁盘上。QJM 共享存储的基本思想来自于 Paxos 算法，采用多个称为 JournalNode 的节点组成的 JournalNode 集群来存储 EditLog。每个 JournalNode 保存同样的 EditLog 副本。每次 NameNode 写 EditLog 的时候，除了向本地磁盘写入 EditLog 之外，也会并行地向 JournalNode 集群之中的每一个 JournalNode 发送写请求，只要大多数 (majority) 的 JournalNode 节点返回成功就认为向 JournalNode 集群写入 EditLog 成功。如果有 2N+1 台 JournalNode，那么根据大多数的原则，最多可以容忍有 N 台 JournalNode 节点挂掉

7. 在NameNode HA中，会出现脑裂问题吗？怎么解决脑裂

假设 NameNode1 当前为 Active 状态，NameNode2 当前为 Standby 状态。如果某一时刻 NameNode1 对应的 ZKFailoverController 进程发生了“假死”现象，那么 Zookeeper 服务端会认为 NameNode1 挂掉了，根据前面的主备切换逻辑，NameNode2 会替代 NameNode1 进入 Active 状态。但是此时 NameNode1 可能仍然处于 Active 状态正常运行，这样 NameNode1 和 NameNode2 都处于 Active 状态，都可以对外提供服务。这种情况称为脑裂

脑裂对于NameNode 这类对数据一致性要求非常高的系统来说是灾难性的，数据会发生错乱且无法恢复。Zookeeper 社区对这种问题的解决方法叫做 fencing，中文翻译为隔离，也就是想办法把旧的 Active NameNode 隔离起来，使它不能正常对外提供服务。

在进行 fencing 的时候，会执行以下的操作：

首先尝试调用这个旧 Active NameNode 的 HAServiceProtocol RPC 接口的 transitionToStandby 方法，看能不能把它转换为 Standby 状态。

如果 transitionToStandby 方法调用失败，那么就执行 Hadoop 配置文件之中预定义的隔离措施，Hadoop 目前主要提供两种隔离措施，通常会选择 sshfence：

(1) sshfence：通过 SSH 登录到目标机器上，执行命令 fuser 将对应的进程杀死

(2) shellfence：执行一个用户自定义的 shell 脚本来将对应的进程隔离

8. 小文件过多会有什么危害,如何避免

Hadoop上大量HDFS元数据信息存储在NameNode内存中,因此过多的小文件必定会压垮NameNode的内存

每个元数据对象约占150byte，所以如果有1千万个小文件，每个文件占用一个block，则NameNode大约需要2G空间。如果存储1亿个文件，则NameNode需要20G空间

显而易见的解决这个问题的方法就是合并小文件,可以选择在客户端上传时执行一定的策略先合并,或者是使用Hadoop的CombineFileInputFormat<K,V>实现小文件的合并

9. 请说下HDFS的组织架构

1）Client：客户端

（1）切分文件。文件上传HDFS的时候，Client将文件切分成一个一个的Block，然后进行存储

（2）与NameNode交互，获取文件的位置信息

（3）与DataNode交互，读取或者写入数据

（4）Client提供一些命令来管理HDFS，比如启动关闭HDFS、访问HDFS目录及内容等

2）NameNode：名称节点，也称主节点，存储数据的元数据信息，不存储具体的数据

（1）管理HDFS的名称空间

（2）管理数据块（Block）映射信息

（3）配置副本策略

（4）处理客户端读写请求

3）DataNode：数据节点，也称从节点。NameNode下达命令，DataNode执行实际的操作

（1）存储实际的数据块

（2）执行数据块的读/写操作

4）Secondary NameNode：并非NameNode的热备。当NameNode挂掉的时候，它并不能马上替换NameNode并提供服务

（1）辅助NameNode，分担其工作量

（2）定期合并Fsimage和Edits，并推送给NameNode

（3）在紧急情况下，可辅助恢复NameNode

10. 请说下MR中Map Task的工作机制

简单概述：

inputFile通过split被切割为多个split文件，通过Record按行读取内容给map（自己写的处理逻辑的方法）

，数据被map处理完之后交给OutputCollect收集器，对其结果key进行分区（默认使用的hashPartitioner），然后写入buffer，每个map task 都有一个内存缓冲区（环形缓冲区），存放着map的输出结果，当缓冲区快满的时候需要将缓冲区的数据以一个临时文件的方式溢写到磁盘，当整个map task 结束后再对磁盘中这个maptask产生的所有临时文件做合并，生成最终的正式输出文件，然后等待reduce task的拉取

详细步骤：

读取数据组件 InputFormat (默认 TextInputFormat) 会通过 getSplits 方法对输入目录中的文件进行逻辑切片规划得到 block, 有多少个 block就对应启动多少个 MapTask.

将输入文件切分为 block 之后, 由 RecordReader 对象 (默认是LineRecordReader) 进行读取, 以 \n 作为分隔符, 读取一行数据, 返回 <key，value>. Key 表示每行首字符偏移值, Value 表示这一行文本内容

读取 block 返回 <key,value>, 进入用户自己继承的 Mapper 类中，执行用户重写的 map 函数, RecordReader 读取一行这里调用一次

Mapper 逻辑结束之后, 将 Mapper 的每条结果通过 context.write 进行collect数据收集. 在 collect 中, 会先对其进行分区处理，默认使用 HashPartitioner

接下来, 会将数据写入内存, 内存中这片区域叫做环形缓冲区(默认100M), 缓冲区的作用是 批量收集 Mapper 结果, 减少磁盘 IO 的影响. 我们的 Key/Value 对以及 Partition 的结果都会被写入缓冲区. 当然, 写入之前，Key 与 Value 值都会被序列化成字节数组

当环形缓冲区的数据达到溢写比列(默认0.8)，也就是80M时，溢写线程启动, 需要对这 80MB 空间内的 Key 做排序 (Sort). 排序是 MapReduce 模型默认的行为, 这里的排序也是对序列化的字节做的排序

合并溢写文件, 每次溢写会在磁盘上生成一个临时文件 (写之前判断是否有 Combiner), 如果 Mapper 的输出结果真的很大, 有多次这样的溢写发生, 磁盘上相应的就会有多个临时文件存在. 当整个数据处理结束之后开始对磁盘中的临时文件进行 Merge 合并, 因为最终的文件只有一个, 写入磁盘, 并且为这个文件提供了一个索引文件, 以记录每个reduce对应数据的偏移量

11. 请说下MR中Reduce Task的工作机制

简单描述：

Reduce 大致分为 copy、sort、reduce 三个阶段，重点在前两个阶段。copy 阶段包含一个 eventFetcher 来获取已完成的 map 列表，由 Fetcher 线程去 copy 数据，在此过程中会启动两个 merge 线程，分别为 inMemoryMerger 和 onDiskMerger，分别将内存中的数据 merge 到磁盘和将磁盘中的数据进行 merge。待数据 copy 完成之后，copy 阶段就完成了，开始进行 sort 阶段，sort 阶段主要是执行 finalMerge 操作，纯粹的 sort 阶段，完成之后就是 reduce 阶段，调用用户定义的 reduce 函数进行处理

详细步骤：

Copy阶段：简单地拉取数据。Reduce进程启动一些数据copy线程(Fetcher)，通过HTTP方式请求maptask获取属于自己的文件（map task 的分区会标识每个map task属于哪个reduce task ，默认reduce task的标识从0开始）。

Merge阶段：这里的merge如map端的merge动作，只是数组中存放的是不同map端copy来的数值。Copy过来的数据会先放入内存缓冲区中，这里的缓冲区大小要比map端的更为灵活。merge有三种形式：内存到内存；内存到磁盘；磁盘到磁盘。默认情况下第一种形式不启用。当内存中的数据量到达一定阈值，就启动内存到磁盘的merge。与map 端类似，这也是溢写的过程，这个过程中如果你设置有Combiner，也是会启用的，然后在磁盘中生成了众多的溢写文件。第二种merge方式一直在运行，直到没有map端的数据时才结束，然后启动第三种磁盘到磁盘的merge方式生成最终的文件。

合并排序：把分散的数据合并成一个大的数据后，还会再对合并后的数据排序。

对排序后的键值对调用reduce方法，键相等的键值对调用一次reduce方法，每次调用会产生零个或者多个键值对，最后把这些输出的键值对写入到HDFS文件中。

12. 请说下MR中shuffle阶段

shuffle阶段分为四个步骤：依次为：分区，排序，规约，分组，其中前三个步骤在map阶段完成，最后一个步骤在reduce阶段完成

shuffle 是 Mapreduce 的核心，它分布在 Mapreduce 的 map 阶段和 reduce 阶段。一般把从 Map 产生输出开始到 Reduce 取得数据作为输入之前的过程称作 shuffle。

Collect阶段：将 MapTask 的结果输出到默认大小为 100M 的环形缓冲区，保存的是 key/value，Partition 分区信息等。

Spill阶段：当内存中的数据量达到一定的阀值的时候，就会将数据写入本地磁盘，在将数据写入磁盘之前需要对数据进行一次排序的操作，如果配置了 combiner，还会将有相同分区号和 key 的数据进行排序。

Merge阶段：把所有溢出的临时文件进行一次合并操作，以确保一个 MapTask 最终只产生一个中间数据文件

4.\*\* Copy阶段\*\*：ReduceTask 启动 Fetcher 线程到已经完成 MapTask 的节点上复制一份属于自己的数据，这些数据默认会保存在内存的缓冲区中，当内存的缓冲区达到一定的阀值的时候，就会将数据写到磁盘之上

Merge阶段：在 ReduceTask 远程复制数据的同时，会在后台开启两个线程对内存到本地的数据文件进行合并操作

Sort阶段：在对数据进行合并的同时，会进行排序操作，由于 MapTask 阶段已经对数据进行了局部的排序，ReduceTask 只需保证 Copy 的数据的最终整体有效性即可。

Shuffle 中的缓冲区大小会影响到 mapreduce 程序的执行效率，原则上说，缓冲区越大，磁盘io的次数越少，执行速度就越快

缓冲区的大小可以通过参数调整, 参数：mapreduce.task.io.sort.mb 默认100M

13. shuffle阶段的数据压缩机制了解吗

在shuffle阶段，可以看到数据通过大量的拷贝，从map阶段输出的数据，都要通过网络拷贝，发送到reduce阶段，这一过程中，涉及到大量的网络IO，如果数据能够进行压缩，那么数据的发送量就会少得多。

hadoop当中支持的压缩算法：

gzip、bzip2、LZO、LZ4、Snappy，这几种压缩算法综合压缩和解压缩的速率，谷歌的Snappy是最优的，一般都选择Snappy压缩。谷歌出品，必属精品

14. 在写MR时，什么情况下可以使用规约

规约（combiner）是不能够影响任务的运行结果的，局部汇总，适用于求和类，不适用于求平均值，如果reduce的输入参数类型和输出参数的类型是一样的，则规约的类可以使用reduce类，只需要在驱动类中指明规约的类即可

15. yarn 集群的架构和工作原理知道多少

YARN的基本设计思想是将MapReduce V1中的JobTracker拆分为两个独立的服务：ResourceManager和ApplicationMaster。ResourceManager负责整个系统的资源管理和分配，ApplicationMaster负责单个应用程序的的管理。

ResourceManager：

RM是一个全局的资源管理器，负责整个系统的资源管理和分配，它主要由两个部分组成：调度器（Scheduler）和应用程序管理器（Application Manager）。

调度器根据容量、队列等限制条件，将系统中的资源分配给正在运行的应用程序，在保证容量、公平性和服务等级的前提下，优化集群资源利用率，让所有的资源都被充分利用应用程序管理器负责管理整个系统中的所有的应用程序，包括应用程序的提交、与调度器协商资源以启动ApplicationMaster、监控ApplicationMaster运行状态并在失败时重启它。

ApplicationMaster：

用户提交的一个应用程序会对应于一个ApplicationMaster，它的主要功能有：

a.与RM调度器协商以获得资源，资源以Container表示。

b.将得到的任务进一步分配给内部的任务。

c.与NM通信以启动/停止任务。

d.监控所有的内部任务状态，并在任务运行失败的时候重新为任务申请资源以重启任务。

nodeManager：

NodeManager是每个节点上的资源和任务管理器，一方面，它会定期地向RM汇报本节点上的资源使用情况和各个Container的运行状态；另一方面，他接收并处理来自AM的Container启动和停止请求。

container：

Container是YARN中的资源抽象，封装了各种资源。一个应用程序会分配一个Container，这个应用程序只能使用这个Container中描述的资源。

不同于MapReduceV1中槽位slot的资源封装，Container是一个动态资源的划分单位，更能充分利用资源。

16. yarn 的任务提交流程是怎样的

当jobclient向YARN提交一个应用程序后，YARN将分两个阶段运行这个应用程序：一是启动ApplicationMaster;第二个阶段是由ApplicationMaster创建应用程序，为它申请资源，监控运行直到结束。

具体步骤如下:

用户向YARN提交一个应用程序，并指定ApplicationMaster程序、启动ApplicationMaster的命令、用户程序。

RM为这个应用程序分配第一个Container，并与之对应的NM通讯，要求它在这个Container中启动应用程序ApplicationMaster。

ApplicationMaster向RM注册，然后拆分为内部各个子任务，为各个内部任务申请资源，并监控这些任务的运行，直到结束。

AM采用轮询的方式向RM申请和领取资源。

RM为AM分配资源，以Container形式返回

AM申请到资源后，便与之对应的NM通讯，要求NM启动任务。

NodeManager为任务设置好运行环境，将任务启动命令写到一个脚本中，并通过运行这个脚本启动任务

各个任务向AM汇报自己的状态和进度，以便当任务失败时可以重启任务。

应用程序完成后，ApplicationMaster向ResourceManager注销并关闭自己

17. yarn 的资源调度三种模型了解吗

在Yarn中有三种调度器可以选择：FIFO Scheduler ，Capacity Scheduler，Fair Scheduler

apache版本的hadoop默认使用的是capacity scheduler调度方式。CDH版本的默认使用的是fair scheduler调度方式

FIFO Scheduler（先来先服务）：

FIFO Scheduler把应用按提交的顺序排成一个队列，这是一个先进先出队列，在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配，以此类推。

FIFO Scheduler是最简单也是最容易理解的调度器，也不需要任何配置，但它并不适用于共享集群。大的应用可能会占用所有集群资源，这就导致其它应用被阻塞，比如有个大任务在执行，占用了全部的资源，再提交一个小任务，则此小任务会一直被阻塞。

Capacity Scheduler（能力调度器）：

对于Capacity调度器，有一个专门的队列用来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器时的时间。

Fair Scheduler（公平调度器）：

在Fair调度器中，我们不需要预先占用一定的系统资源，Fair调度器会为所有运行的job动态的调整系统资源。

比如：当第一个大job提交时，只有这一个job在运行，此时它获得了所有集群资源；当第二个小任务提交后，Fair调度器会分配一半资源给这个小任务，让这两个任务公平的共享集群资源。

需要注意的是，在Fair调度器中，从第二个任务提交到获得资源会有一定的延迟，因为它需要等待第一个任务释放占用的Container。小任务执行完成之后也会释放自己占用的资源，大任务又获得了全部的系统资源。最终的效果就是Fair调度器即得到了高的资源利用率又能保证小任务及时完成。