stonezhu Lv2
2019年06月11日 阅读 86

ShuffleManager 原理

在 Spark 的源码中，负责 shuffle 过程的执行、计算、处理的组件主要是 ShuffleManager。

在 Spark 1.2 以前，默认的 shuffle 计算引擎是 HashShuffleManager。该 ShuffleMananger 有一个非常严重的弊端，就是会产生大量的磁盘文件，进而有大量的磁盘 IO 操作，比较影响性能。

因此在 Spark 1.2 之后，默认的 ShuffleManager 改成了 SortShuffleManager。SortShuffleManager 相对来说，有了一定的改进。主要就在于，每个 Task 在 Shuffle Write 操作时，虽然也会产生较大的磁盘文件，但最后会将所有的临时文件合并 (merge) 成一个磁盘文件，因此每个 Task 就只有一个磁盘文件。在下一个 Stage 的 Shuffle Read Task 拉取自己数据的时候，只要根据索引拉取每个磁盘文件中的部分数据即可。

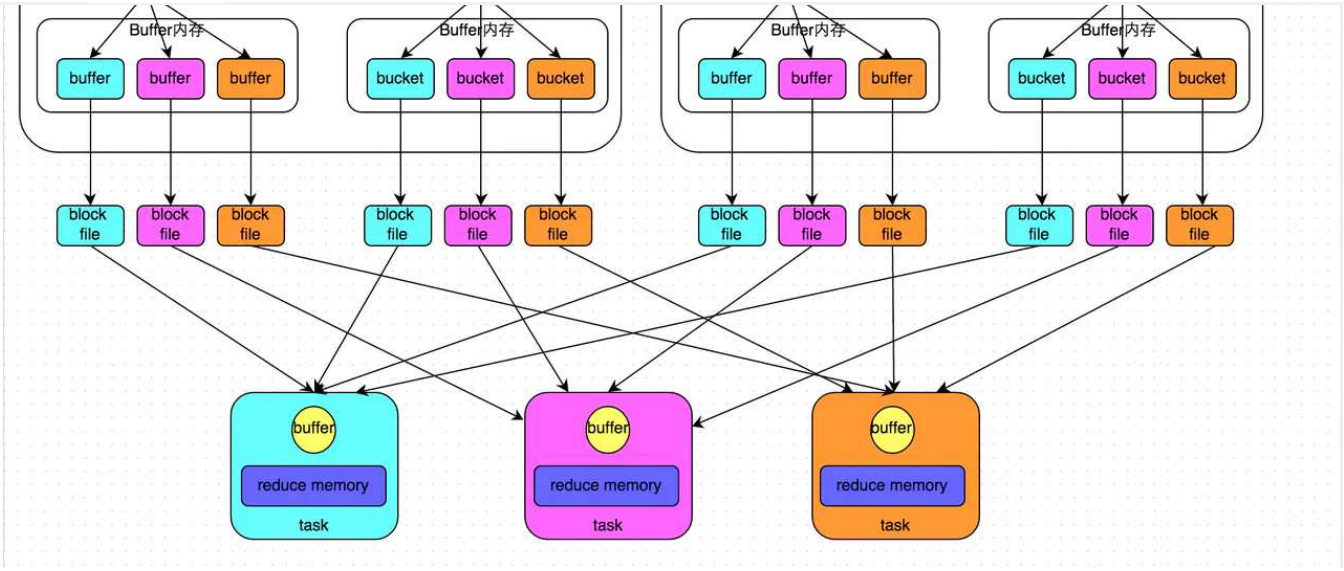
一，HashShuffleManager 运行原理

普通模式下，在 Shuffle Write 阶段，每个 Task 将数据按照 Key 进行 Hash 计算，然后按照计算结果，将相同的 Key 对应的数据写入内存缓冲区，当内存缓冲区写满之后会直接溢写到磁盘文件。这里需要写多少个磁盘文件，和下一个 stage 的 Shuffle Read Task 的数量一致。

然后，Shuffle Read 阶段的每个 Task 会拉取 Shuffle Write 阶段所有相同 Key 的文件，一遍拉取一遍聚合。每个 Shuffle Read 阶段的 Task 都有自己的缓冲区，每次只能拉取与缓冲区大小一致的数据，然后通过内存中的 Map 进行聚合等操作，聚合完一批再取下一批数据。

比如，当前 Stage 有 5 个 Executor，每个 Executor 分配一个 cpu core，有 50 个 task，每个 Executor 执行 10 个 task；下一个 stage 有100 个 task。那么在 Shuffle Write 阶段每个 task 要创建 100 个磁盘文件，每个 Executor 进程要创建 1000 个文件，一共要创建 1000 * 5 = 5000 个磁盘文件，数量很多。

具体执行原理图如下图所示：

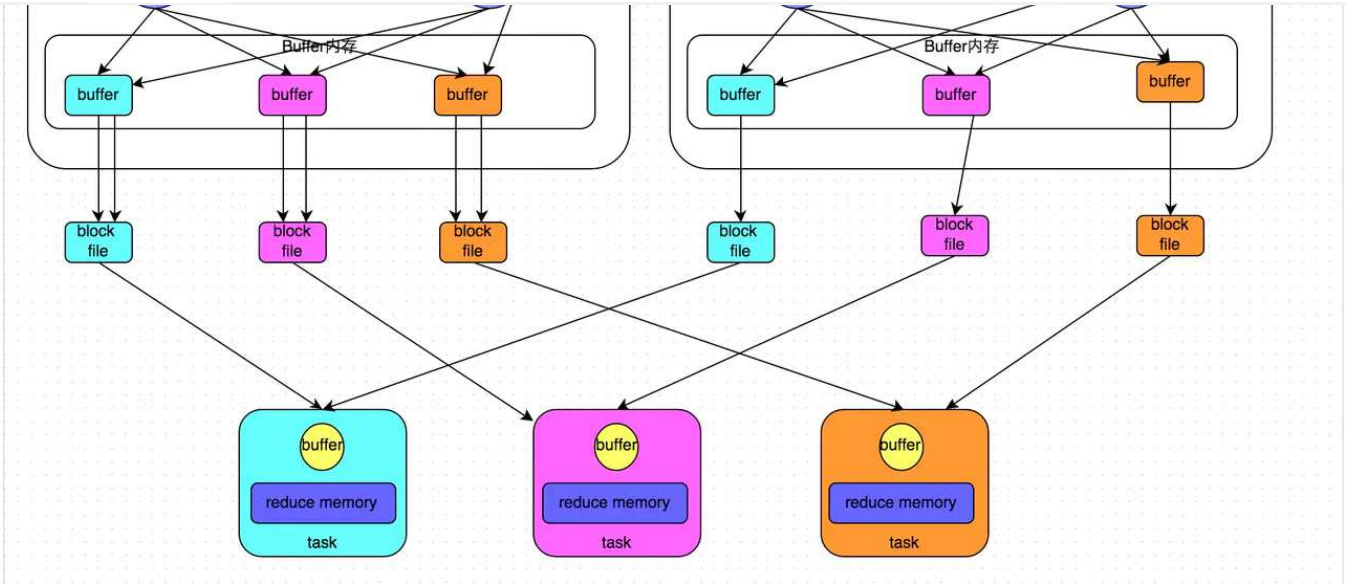


针对 HashShuffleManager 我们可以设置一个参数： `spark.shuffle.consolidateFiles` 。这个参数的值默认是 fasle，如果设置成 true 之后就会开启优化机制。

当开启这个参数之后，在 Shuffle Write 阶段写文件的时候会复用文件，每个 task 不会为 Shuffle Read 阶段的 task 都创建一份文件。此时会出现一个 shuffleFileGroup 的概念，每个 shuffleFileGroup 会对应一批磁盘文件，磁盘文件的数量和 Shuffle Read 阶段的 task 数量一致。每个 Executor 上有多少个 cpu core 就会并行执行几个 task，每个 task 会创建一个 shuffleFileGroup，然后后续并行执行的 task 会复用前面生成的这个 shuffleFileGroup。

比如，当前 stage 有 5 个 Executor，每个 Executor 分配 3 个 cpu core，一共有 50 个 task，每个 Executor 执行 10 个 task，Shuffle Read 阶段有 100 个 task。那么此时，每个 Executor 进程会创建 3 * 100 个文件，一共会创建 5 * 3 * 100 个文件。

具体原理如图示：



二，SortShuffleManager 运行原理

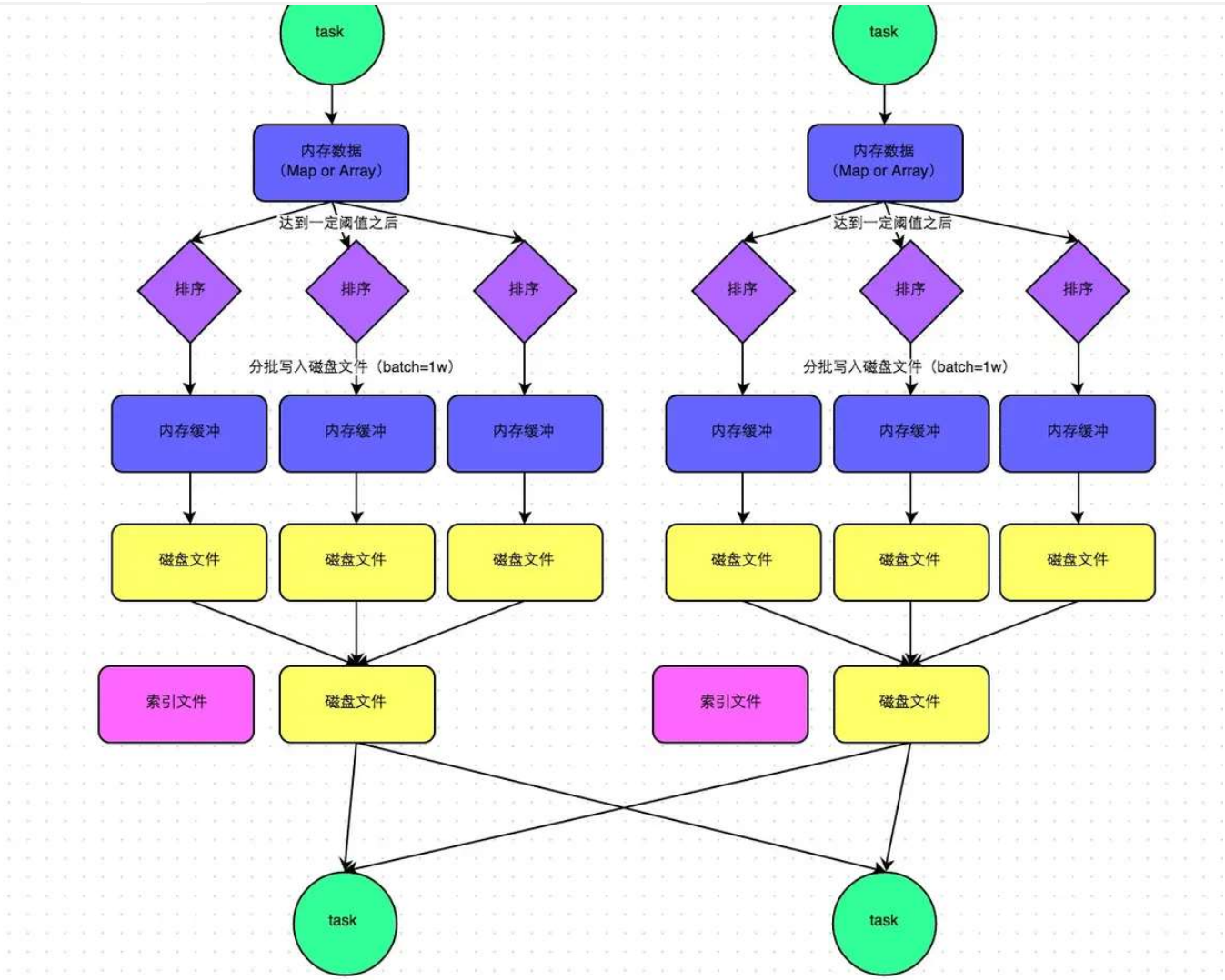
SortShuffleManager 运行机制有两种，一种是普通运行机制，另一种是 bypass 运行机制。当 shuffle read task 的数量小于等于 `spark.shuffle.sort.bypassMergeThreshold` 参数值时 (默认是 200)，就会启用 bypass 机制。

1，普通机制

在该模式下，Shuffle Write 阶段会将数据写入一个内存的数据结构中，此时根据不同的算子会有不同的数据结构。比如是 reduceByKey 这种聚合类的 shuffle 算子，会选用 Map 数据结构，一遍用 Map 进行聚合（HashShuffleManager 聚合操作是放在 Shuffle Read 阶段），一遍写入内存；如果是 join 相关的普通 shuffle 算子的话，会用 Array 数据结构，直接写入内存。当内存达到临界阈值之后，会将内存中的数据进行排序，然后分批次写入磁盘 (默认每批次有 1W 条数据)，在写入磁盘的时候不会像 HashShuffleManager 那样直接写入磁盘，这里会先写入内存缓冲流，当缓冲流满溢之后一次性写入磁盘。

此时也会生成大批量的文件，最后会将之前所有的临时磁盘文件进行合并，这就是 merge 过程 (就是将所有的临时磁盘文件中的数据读取出来，然后依次写入最终的文件中)。每个 task 最终会生成一份磁盘文件和一份索引文件，索引文件中标示了下游每个 task 的数据在文件中的 start offset 和 end offset。

比如，当前 stage 有 5 个 Executor，每个 Executor 分配 1 个 cpu core，共有 50 个 task，每个 Executor 执行 10 个 task；下一个 stage 有 100 个 task。那么每个 Executor 创建 10 个磁盘文件，一共有 50 个磁盘文件。



2，bypass 机制

触发该机制的条件：

1，shuffle reduce 端的 task 数量小于 `spark.shuffle.sort.bypassMergeThreshold` 参数值的时候；

2，不是聚合类的shuffle算子（比如reduceByKey）；

该机制下，当前 stage 的每个 task 会将数据的 key 进行 hash，然后将相同 hash 的 key 锁对应的数据写入到同一个内存缓冲区，缓冲写满后会溢写到磁盘文件，这里和 HashShuffleManager一致。

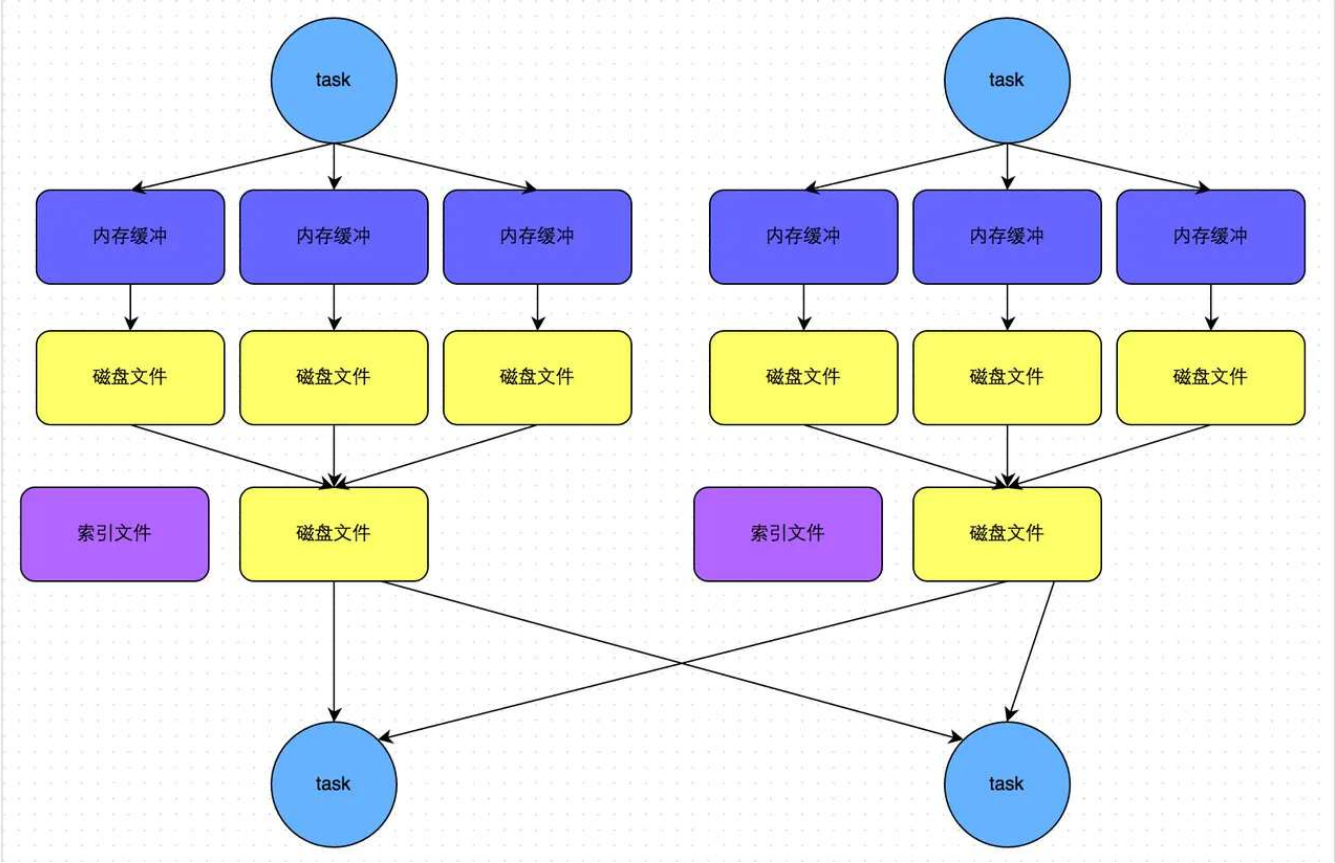
然后会进入 merge 阶段，将所有的磁盘文件合并成一个磁盘文件，并创建一个索引文件。

相比较于普通机制，这里有两个地方不同：

2，该模式下在写入磁盘之前不会排序；

3，磁盘写机制不同。

具体如图示：



三，shuffle 相关的参数

spark.shuffle.file.buffer

- 默认值：32k
- 参数说明：该参数用于设置 shuffle write task 的 BufferedOutputStream 的 buffer 缓冲大小。将数据写到磁盘文件之前，会先写入 buffer 缓冲中，待缓冲写满之后，才会溢写到磁盘。
- 调优建议：如果作业可用的内存资源较为充足的话，可以适当增加这个参数的大小（比如 64k），从而减少 shuffle write 过程中溢写磁盘文件的次数，也就可以减少磁盘 IO 次数，进而提升性能。在实践中发现，合理调节该参数，性能会有 1%~5% 的提升。

spark.reducer.maxSizeInFlight

- 默认值：48m

- 调优建议：如果作业可用的内存资源较为充足的话，可以适当增加这个参数的大小（比如96m），从而减少拉取数据的次数，也就可以减少网络传输的次数，进而提升性能。在实践中发现，合理调节该参数，性能会有 1%~5% 的提升。

spark.shuffle.io.maxRetries

- 默认值：3
- 参数说明：shuffle read task 从 shuffle write task 所在节点拉取属于自己的数据时，如果因为网络异常导致拉取失败，是会自动进行重试的。该参数就代表了可以重试的最大次数。如果在指定次数之内拉取还是没有成功，就可能会导致作业执行失败。
- 调优建议：对于那些包含了特别耗时的 shuffle 操作的作业，建议增加重试最大次数（比如 60 次），以避免由于 JVM 的 full gc 或者网络不稳定等因素导致的数据拉取失败。在实践中发现，对于针对超大数据量（数十亿~上百亿）的 shuffle 过程，调节该参数可以大幅度提升稳定性。

spark.shuffle.io.retryWait

- 默认值：5s
- 参数说明：具体解释同上，该参数代表了每次重试拉取数据的等待间隔，默认是 5s。
- 调优建议：建议加大间隔时长（比如 60s），以增加 shuffle 操作的稳定性。

spark.shuffle.memoryFraction

- 默认值：0.2
- 参数说明：该参数代表了 Executor 内存中，分配给 shuffle read task 进行聚合操作的内存比例，默认是 20%。
- 调优建议：在资源参数调优中讲解过这个参数。如果内存充足，而且很少使用持久化操作，建议调高这个比例，给 shuffle read 的聚合操作更多内存，以避免由于内存不足导致聚合过程中频繁读写磁盘。在实践中发现，合理调节该参数可以将性能提升 10% 左右。

spark.shuffle.manager

- 默认值：sort
- 参数说明：该参数用于设置 ShuffleManager 的类型。Spark 1.5 以后，有三个可选项：hash、sort 和 tungsten-sort。HashShuffleManager 是 Spark 1.2 以前的默认选项，但是 Spark 1.2 以及之后的版本默认都是 SortShuffleManager 了。tungsten-sort 与 sort 类似，但是使用了 tungsten 计划中的堆外内存管理机制，内存使用效率更高。
- 调优建议：由于 SortShuffleManager 默认会对数据进行排序，因此如果你的业务逻辑中需要该排序机制的话，则使用默认的 SortShuffleManager 就可以；而如果你的业务逻辑不需要对数据进行排序，那么建议参考后面的几个参数调优，通过 bypass 机制或优化的

spark.shuffle.sort.bypassMergeThreshold

- 默认值：200
- 参数说明：当 ShuffleManager 为 SortShuffleManager 时，如果 shuffle read task 的数量小于这个阈值（默认是200），则 shuffle write 过程中不会进行排序操作，而是直接按照未经优化的 HashShuffleManager 的方式去写数据，但是最后会将每个task产生的所有临时磁盘文件都合并成一个文件，并会创建单独的索引文件。
- 调优建议：当你使用 SortShuffleManager 时，如果的确不需要排序操作，那么建议将这个参数调大一些，大于 shuffle read task 的数量。那么此时就会自动启用 bypass 机制，map-side 就不会进行排序了，减少了排序的性能开销。但是这种方式下，依然会产生大量的磁盘文件，因此 shuffle write 性能有待提高。

spark.shuffle.consolidateFiles

- 默认值：false
- 参数说明：如果使用 HashShuffleManager，该参数有效。如果设置为 true，那么就会开启 consolidate 机制，会大幅度合并 shuffle write 的输出文件，对于 shuffle read task 数量特别多的情况下，这种方法可以极大地减少磁盘 IO 开销，提升性能。
- 调优建议：如果的确不需要 SortShuffleManager 的排序机制，那么除了使用 bypass 机制，还可以尝试将 spark.shffle.manager 参数手动指定为 hash，使用 HashShuffleManager，同时开启 consolidate 机制。在实践中尝试过，发现其性能比开启了 bypass 机制的 SortShuffleManager 要高出 10%~30%。

相关推荐

专栏 · 南宫饱虎 · 15小时前 · Spark

一篇文章搞清spark任务如何执行

1

专栏 · 说出你的愿望吧 · 19天前 · Spark

Spark Streaming 的容错机制

28 2

专栏 · 说出你的愿望吧 · 28天前 · Spark

一文带你理清Spark Core调优的方方面面

48 4

专栏 · 说出你的愿望吧 · 1月前 · Spark

从零开始认识 Spark

47 10

专栏 · 说出你的愿望吧 · 1月前 · Spark

Spark的Shuffle总结分析

37 12

专栏 · 说出你的愿望吧 · 1月前 · Spark

一文带你过完Spark RDD的基础概念

45 8

专栏 · 说出你的愿望吧 · 1月前 · Spark

关于Spark基础的一些小问题补充

26 1

荐 · 专栏 · wlysola · 1年前 · 面试 / 后端

金九银十铁12，目前腾讯、美团等五家大厂都收到意向offer | 掘金技术征文

597 55

专栏 · OPPO互联网技术 · 18天前 · Spark

剖析Spark数据分区之Spark streaming & TiSpark

荐 · 专栏 · 字节跳动技术团队 · 4月前 · 架构 / Spark

字节跳动在Spark SQL上的核心优化实践 | 字节跳动技术沙龙

12 2

专栏 · 胡七筒 · 1年前 · JavaScript / iOS

程序猿生存指南-24 加班狂魔

72 40

专栏 · Hiway · 2月前 · Spark

Spark RPC模块源码学习

5

专栏 · OPPO互联网技术 · 3月前 · Spark / 人工智能

Spark ML的特征处理实战

6 1

专栏 · 萧洒的身影 · 9月前 · Spark

用 Spark 处理复杂数据类型（Struct、Array、Map、JSON字符串等）

专栏 · chouheiwa · 1年前 · 命令行 / C++

一道值得思考的iOS面试题

69 36

专栏 · 姜文奇 · 11月前 · Spark

在Docker上一键部署你的Spark计算平台

11 6

专栏 · 美图数据技术团队 · 1年前 · Spark / Hadoop

Hello Spark! | Spark，从入门到精通

97 5

专栏 · OPPO互联网技术 · 3月前 · Spark

剖析Spark数据分区之Spark RDD分区

2 3

实验楼 · 2年前 · Spark