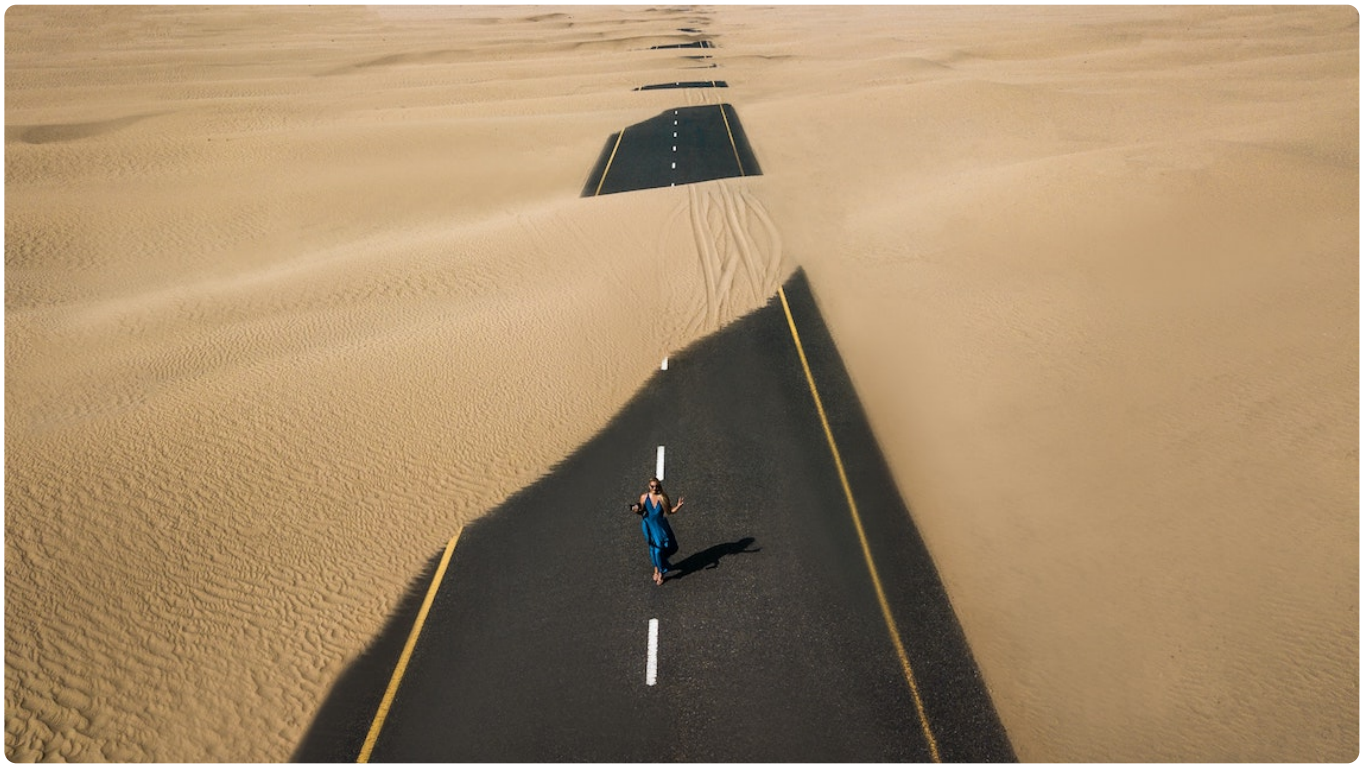
15 | 内存视角（一）：如何最大化内存的使用效率？

2021-04-16 吴磊

Spark性能调优实战 进入课程 >



讲述：吴磊
时长 18:23 大小 16.85M

你好，我是吴磊。

上一讲我们说，想要提升 CPU 利用率，最重要的就是合理分配执行内存，但是，执行内存只是 Spark 内存分区的一部分。因此，想要合理分配执行内存，我们必须先从整体上合理划分好 Spark 所有的内存区域。

可在实际开发应用的时候，身边有不少同学向我抱怨：“Spark 划分不同内存区域的原理我都知道，但我还是不知道不同内存区域的大小该怎么设置，纠结来、纠结去。最后，所有跟内存有关的配置项，我还是保留了默认值。”

这种不能把原理和实践结合起来的情况很常见，所以今天这一讲，我就从熟悉的 Label Encoding 实例出发，**一步步带你去分析不同情况下，不同内存区域的调整办法，**帮你归纳

出最大化内存利用率的常规步骤。这样，你在调整内存的时候，就能结合应用的需要，做到有章可循、有的放矢。

从一个实例开始

我们先来回顾一下第 5 讲中讲过的 Label Encoding。在 Label Encoding 的业务场景中，我们需要对用户兴趣特征做 Encoding。依据模板中兴趣字符串及其索引位置，我们的任务是把千亿条样本中的用户兴趣转换为对应的索引值。模板文件的内容示例如下所示。

复制代码

```
//模板文件
//用户兴趣
体育-篮球-NBA-湖人
军事-武器-步枪-AK47
```

实现的代码如下所示，注意啦，这里的代码是第 5 讲中优化后的版本。

复制代码

```
/**
输入参数：模板文件路径，用户兴趣字符串
返回值：用户兴趣字符串对应的索引值
*/
//函数定义
val findIndex: (String) => (String) => Int = {
(filePath) =>
val source = Source.fromFile(filePath, "UTF-8")
val lines = source.getLines().toArray
source.close()
val searchMap = lines.zip(0 until lines.size).toMap
(interest) => searchMap.getOrElse(interest, -1)
}
val partFunc = findIndex(filePath)

//Dataset中的函数调用
partFunc("体育-篮球-NBA-湖人")
```

下面，咱们先一起回顾一下代码实现思路，再来分析它目前存在的性能隐患，最后去探讨优化它的方法。

首先，findIndex 函数的主体逻辑比较简单，就是读取模板文件和构建 Map 映射，以及查找用户兴趣并返回索引。不过，findIndex 函数被定义成了高阶函数。这样一来，当以模板文件为实参调用这个高阶函数的时候，我们会得到一个内置了 Map 查找字典的标量函数 partFunc，最后在千亿样本上调用 partFunc 完成数据转换。**利用高阶函数，我们就避免了让 Executor 中的每一个 Task 去读取模板文件，以及从头构建 Map 字典这种执行低效的做法。**

在运行时，这个函数在 Driver 端会被封装到一个又一个的 Task 中去，随后 Driver 把这些 Task 分发到 Executor，Executor 接收到任务之后，交由线程池去执行（调度系统的内容可以回顾第 5 讲）。这个时候，每个 Task 就像是一架架小飞机，携带着代码“乘客”和数据“行李”，从 Driver 飞往 Executor。Task 小飞机在 Executor 机场着陆之后，代码“乘客”乘坐出租车或是机场大巴，去往 JVM stack；数据“行李”则由专人堆放在 JVM Heap，也就是我们常说的堆内内存。

回顾 Label encoding 中的 findIndex 函数不难发现，其中大部分都是代码“乘客”，唯一的数据“行李”是名为 searchMap 的 Map 字典。像这样用户自定义的数据结构，消耗的内存区域就是堆内内存的 User Memory（Spark 对内存区域的划分内容可以回顾一下第 7 讲）。

User Memory 性能隐患

回顾到这里，你觉得 findIndex 函数有没有性能隐患呢？你可以先自己思考一下，有了答案之后再来看我下面的分析。

答案当然是“有”。首先，每架小飞机都携带这么一份数据“大件行李”，自然需要消耗更多的“燃油”，这里的“燃油”指的**是 Task 分发过程中带来的网络开销**。其次，因为每架小飞机着陆之后，都会在 Executor 的“旅客行李专区”User Memory 寄存上这份同样的数据“行李”，所以，**User Memory 需要确保有足够的空间可以寄存所有旅客的行李，也就是大量的重复数据**。

那么，User Memory 到底需要准备出多大的内存空间才行呢？我们不妨来算一算。这样的计算并不难，只需要用飞机架次乘以行李大小就可以了。

用户自定义的数据结构往往是用于辅助函数完成计算任务的，所以函数执行完毕之后，它携带的数据结构的生命周期也就告一段落。**因此，在 Task 的数量统计上，我们不必在意**

一个 Executor 总共需要处理多少个 Task，只需要关注它在同一时间可以并行处理的 Task 数量，也就是 Executor 的线程池大小即可。

我们说过，Executor 线程池大小由 spark.executor.cores 和 spark.task.cpus 这两个参数的商（spark.executor.cores/spark.task.cpus）决定，我们暂且把这个商记作 #threads。

接下来是估算数据“行李”大小，由于 searchMap 并不是分布式数据集，因此我们不必采用先 Cache，再提取 Spark 执行计划统计信息的方式。对于这样的 Java 数据结构，我们完全可以在 REPL 中，通过 Java 的常规方法估算数据存储大小，估算得到的 searchMap 大小记为 #size。

好啦！现在，我们可以算出，User Memory 至少需要提供 #threads * #size 这么大的内存空间，才能支持分布式任务完成计算。但是，对于 User Memory 内存区域来说，使用 #threads * #size 的空间去重复存储同样的数据，本身就是降低了内存的利用率。那么，我们该怎么省掉 #threads * #size 的内存消耗呢?

性能调优

学习过广播变量之后，想必你头脑中已经有了思路。没错，咱们可以尝试使用广播变量，来对示例中的代码进行优化。

仔细观察 findIndex 函数，我们不难发现，函数的核心计算逻辑有两点。一是读取模板文件、创建 Map 映射字典；二是以给定字符串对字典进行查找，并返回查找结果。显然，千亿样本转换的核心需求是其中的第二个环节。既然如此，我们完全可以把创建好的 Map 字典封装成广播变量，然后分发到各个 Executors 中去。

有了广播变量的帮忙，凡是发往同一个 Executor 的 Task 小飞机，都无需亲自携带数据“行李”，这些大件行李会由“联邦广播快递公司”派货机专门发往各个 Executors，Driver 和每个 Executors 之间，都有一班这样的货运专线。思路说完了，优化后的代码如下所示。

复制代码

```
1 /**
2 广播变量实现方式
3 */
```

```
//定义广播变量
val source = Source.fromFile(filePath, "UTF-8")
val lines = source.getLines().toArray
source.close()
val searchMap = lines.zip(0 until lines.size).toMap
val bcSearchMap = sparkSession.sparkContext.broadcast(searchMap)

//在Dataset中访问广播变量
bcSearchMap.value.getOrElse("体育-篮球-NBA-湖人", -1)

```

上面代码的实现思路很简单：第一步还是读取模板文件、创建 Map 字典；第二步，把 Map 字典封装为广播变量。这样一来，在对千亿样本进行转换时，我们直接通过 bcSearchMap.value 读取广播变量内容，然后，通过调用 Map 字典的 getOrElse 方法来获取用户兴趣对应的索引值。

相比最开始的第一种实现方式，第二种实现方式的代码改动还是比较小的，那这一版代码对内存的消耗情况有什么改进呢？

我们发现，Task 小飞机的代码“乘客”换人了！**小飞机之前需要携带函数 findIndex，现在则换成了一位“匿名的乘客”**：一个读取广播变量并调用其 getOrElse 方法的匿名函数。由于这位“匿名的乘客”将大件行李托运给了“联邦广播快递公司”的专用货机，因此，Task 小飞机着陆后，没有任何“行李”需要寄存到 User Memory。换句话说，优化后的版本不会对 User Memory 内存区域进行占用，所以第一种实现方式中 #threads * #size 的内存消耗就可以省掉了。

Storage Memory 规划

这样一来，原来的内存消耗转嫁到了广播变量身上。但是，广播变量也会消耗内存，这会不会带来新的性能隐患呢？那我们就来看看，广播变量消耗的具体是哪块内存区域。

回顾存储系统那一讲，我们说过，Spark 存储系统主要有 3 个服务对象，分别是 Shuffle 中间文件、RDD 缓存和广播变量。它们都由 Executor 上的 BlockManager 进行管理，对于数据在内存和磁盘中的存储，BlockManager 利用 MemoryStore 和 DiskStore 进行抽象和封装。

那么，广播变量所携带的数据内容会物化到 MemoryStore 中去，以 Executor 为粒度为所有 Task 提供唯一的一份数据拷贝。MemoryStore 产生的内存占用会被记入到 Storage

Memory 的账上。**因此，广播变量消耗的就是 Storage Memory 内存区域。**

接下来，我们再来盘算一下，第二种实现方式究竟需要耗费多少内存空间。由于广播变量的分发和存储以 Executor 为粒度，因此每个 Executor 消耗的内存空间，就是 searchMap 一份数据拷贝的大小。searchMap 的大小我们刚刚计算过就是 #size。

明确了 Storage Memory 内存区域的具体消耗之后，我们自然可以根据公式：（spark.executor.memory – 300MB）* spark.memory.fraction * spark.memory.storageFraction 去有针对性地调节相关的内存配置项。

内存规划两步走

现在，咱们在两份不同的代码实现下，分别定量分析了不同内存区域的消耗与占用。对于这些消耗做到心中有数，我们自然就能够相应地去调整相关的配置项参数。基于这样的思路，想要最大化内存利用率，我们需要遵循两个步骤：

预估内存占用

调整内存配置项

我们以堆内内存为例，来讲一讲内存规划的两步走具体该如何操作。我们都知道，堆内内存划分为 Reserved Memory、User Memory、Storage Memory 和 Execution Memory 这 4 个区域。预留内存固定为 300MB，不用理会，其他 3 个区域需要你去规划。

预估内存占用

首先，我们来说内存占用的预估，主要分为三步。

第一步，计算 User Memory 的内存消耗。我们先汇总应用中包含的自定义数据结构，并估算这些对象的总大小 #size，然后**用 #size 乘以 Executor 的线程池大小，即可得到 User Memory 区域的内存消耗 #User。**

第二步，计算 Storage Memory 的内存消耗。我们先汇总应用中涉及的广播变量和分布式数据集缓存，分别估算这两类对象的总大小，分别记为 #bc、#cache。另外，我们把集群

中的 Executors 总数记作 #E。这样，**每个 Executor 中 Storage Memory 区域的内存消耗的公式就是：#Storage = #bc + #cache / #E。**

第三步，计算执行内存的消耗。学习上一讲，我们知道执行内存的消耗与多个因素有关。第一个因素是 Executor 线程池大小 #threads，第二个因素是数据分片大小，而数据分片大小取决于数据集尺寸 #dataset 和并行度 #N。因此，**每个 Executor 中执行内存的消耗的计算公式为：#Execution = #threads * #dataset / #N。**

调整内存配置项

得到这 3 个内存区域的预估大小 #User、#Storage、#Execution 之后，调整相关的内存配置项就是一件水到渠成的事情（由公式（spark.executor.memory – 300MB）* spark.memory.fraction * spark.memory.storageFraction）可知），这里我们也可以分为 3 步。

首先，根据定义，**spark.memory.fraction 可以由公式（#Storage + #Execution）/（#User + #Storage + #Execution）计算得到。**

同理，**spark.memory.storageFraction 的数值应该参考（#Storage）/（#Storage + #Execution）。**

最后，对于 Executor 堆内内存总大小 spark.executor.memory 的设置，我们自然要参考 4 个内存区域的总消耗，也就是 **300MB + #User + #Storage + #Execution。不过，我们要注意，利用这个公式计算的前提是，不同内存区域的占比与不同类型的数据消耗一致。**

总的来说，在内存规划的两步走中，第一步预估不同区域的内存占比尤为关键，因为第二步中参数的调整完全取决于第一步的预估结果。如果你按照这两个步骤去设置相关的内存配置项，相信你的应用在运行时就能够充分利用不同的内存区域，避免出现因参数设置不当而导致的内存浪费现象，从而在整体上提升内存利用率。

小结

合理划分 Spark 所有的内存区域，是同时提升 CPU 与内存利用率的基础。因此，掌握内存规划很重要，在今天这一讲，我们把内存规划归纳为两步走。

第一步是预估内存占用。

求出 User Memory 区域的内存消耗，公式为：#User=#size 乘以 Executor 线程池的大小。

求出每个 Executor 中 Storage Memory 区域的内存消耗，公式为：#Storage = #bc + #cache / #E。

求出执行内存区域的内存消耗，公式为：#Execution = #threads * #dataset / #N。

第二步是调整内存配置项：根据公式得到的 3 个内存区域的预估大小 #User、#Storage、#Execution，去调整（spark.executor.memory – 300MB）* spark.memory.fraction * spark.memory.storageFraction 公式中涉及的所有配置项。

spark.memory.fraction 可以由公式（#Storage + #Execution）/（#User + #Storage + #Execution）计算得到。

spark.memory.storageFraction 的数值应该参考（#Storage）/（#Storage + #Execution）。

spark.executor.memory 的设置，可以通过公式 300MB + #User + #Storage + #Execution 得到。

这里，我还想多说几句，**内存规划两步走终归只是手段，它最终要达到的效果和目的，是确保不同内存区域的占比与不同类型的数据消耗保持一致，从而实现内存利用率的最大化。**

每日一练

1. 你知道估算 Java 对象存储大小的方法有哪些吗？不同的方法又有哪些优、劣势呢？
2. 对于内存规划的第一步来说，要精确地预估运行时每一个区域的内存消耗，很费时、费力，调优的成本很高。如果我们想省略掉第一步的精确计算，你知道有哪些方法能够粗略、快速地预估不同内存区域的消耗占比吗？

期待在留言区看到你的思考和答案，我们下一讲见！

提建议

上一篇 14 | CPU视角：如何高效地利用CPU?

下一篇 16 | 内存视角（二）：如何有效避免Cache滥用?

精选留言 (8)

写留言

胡浩
2021-04-16

老师，为何UserMemory中自定义数据结构不能像bc那样在StorageMemory中只存一份?

展开

作者回复: 好问题，我们换个角度讨论这个问题。你的问题其实是：为什么广播变量可以复用，但是User Memory中的数据却不行。

想要复用数据，前提是有“一个人”或是“一个地方”，明确记录了需要复用的这份数据，存储在什么地方，只有这样，后面的任务才能复用这份数据，这样的信息，又叫元数据、元信息。

这就好比，你去宜家买家具，你是需要根据家具的Id，去仓库中自提货物的。家具Id上明确标记了，你需要的货物，存在哪个货架、哪个层、哪个位置。如果没有这样的元信息，偌大的宜家家居超市，你不可能找到你需要的东西。

广播变量这个“货物”的具体地址，BlockManager会帮忙记录，所以“后来的人”（Task），如果需要访问广播变量，它能从BlockManager那里迅速地知道广播变量存储在哪里（哪个executor的什么位置），可以迅速地访问数据。

但是，User Memory也好，Storage、Execution memory也罢，他们本质上就是JVM heap。JVM heap虽然也会记录对象引用对应的存储地址，但是，它没有办法区分，两份数据，在内容上，是不是一样的。实际上，JVM也没有这样的义务。

因此，同一个Task当中两份完全一样的数据，到了JVM那里，它并不知道：“OK，这两份数据一样，后来的人只要访问已有的数据就OK了”。对他来说，这是两份不同的数据，即便内容都一样，他照样会单独花费存储空间来存储。根本原因在于，他并没有Spark的运行时上下文， 不能像BlockManager那样维护全局的元数据。

7

巧克力黑
2021-04-19

REPL 中，通过 Java 的常规方法估算数据存储大小
老师，这个过程具体是怎么做呢。

展开

作者回复: 有不少类库可以“拿来即用”，比如Java类库：Instrumentation，或者第三方库，比如RamUsageEstimator，可以多搜一搜，这样的工具还是蛮多的~

4

zxk
2021-04-18

老师，这边想请教两个问题。
问题一：User Memory、Execution Memory、Storage Memory 是属于 Spark 自身对内存区域的划分，但 Spark 的 executor 实际上又是一个 JVM，假如我把 User Memory 设置的非常小，又自定义了一个很大的数据结构，此时 User Memory 不够用了，而 Exec

ution Memory、Storage Memory 还有很大的空闲，那么这时候会不会 OOM？如果...

展开

作者回复: 都是非常好的问题。
问题一：你说的没错，不管是哪片内存区域，实际上都是JVM Heap的一部分。回答你的问题：如果User Memory空间不足，但是Spark Memory（Storage + Execution）空闲，会不会OOM？对于这种情况，即使你自定的数据结构其实超过了User Memory，但实际上，并不会立即报OOM，因为Spark对于堆内内存的预估，没有那么精确。这里有些Tricky，对于Spark划定的各种“线”，也就是通过配置项设置的不同区域的百分比，它类似于一种“软限制”，也就是建议你遵守它划出的一道道“线”，但如果你没有遵守，强行“跨线”，只要其他区域还有空间，你真的抢占了，Spark也不会立即阻止你。这就好比“中国式”过马路，红绿灯是一种“软限制”，即使是红灯，只要没有机动车通行，咱们照样可以凑一堆人过马路，但如果突然窜出来一辆车，把咱撞了，那就是咱们理亏，因为我们是“闯红灯”的那一方。Spark的“线”也类似，是一种“软限制”。但是，“出来混迟早是要还的”，你的自定义数据结构，占了本该属于Spark Memory的地盘，那么Spark在执行任务或是缓存的时候，很有可能就跟着倒霉，最后（加重加粗）“看上去”是执行任务或是缓存任务OOM，但本质上，是因为你的自定义数据结构，提前占了人家的地方。这种时候，你会很难debug。这也就是为什么Spark要求开发者要遵循各个配置项的设置，不要“越线”。

问题二：还是User Memory，那个map就是自定义数据结构，它会消耗User Memory。你的那种用法，本质上就是闭包。

4

Fendora范东_
2021-04-16

还有个疑问，想请教下磊哥
文中说的数据集大小是内存中的数据集吧
文件落盘后数据集大小可以很方便查看，那内存中数据集大小怎么看呢

展开

作者回复: 对，没错，文中说的数据集大小是内存中的数据集。

这块咱们有过介绍哈~ 最精确的办法：

```
val df: DataFrame = _
df.cache.count

val plan = df.queryExecution.logical
val estimated: BigInt = spark
```

```
.sessionState
.executePlan(plan)
.optimizedPlan
.stats
.sizeInBytes
```

就是先Cache，再查看物理执行计划，可以获得数据集在内存中存储大小。

4

Geek_d794f8
2021-04-18

老师，我有一个疑惑，对于有多个stage的任务，每个stage的内存预估的情况可能不一样。那样就无法给一个比较适合所有stage得内存配置？

展开

作者回复: 好问题，确实，并发度、并行度、执行内存，三者“三足鼎立”。并发度和执行内存，都是一开始就设置好了，在整个作业的执行过程中，都是一样的，不会改变。但是，并行度不是，并行度可以随着作业的进展，随时调整。

所以，在不同的Stages内，为了维持3者的平衡，我们可以通过调整并行度，来维持“三足鼎立”的平衡。那么问题来，在不同的Stages，咋调整并行度呢？

有两种办法：
1. spark.sql.shuffle.partitions
通过spark.conf.set(spark.sql.shuffle.partitions, XXX)来控制每个Stages的并行度大小。这么做有个缺点，就是当你Stages比较多得时候，这个配置项频繁地改来改去，应用的维护成本会很高。

2. AQE的自动分区合并，也就是让Spark自动帮你算合适的并行度。但是这个需要相关的配置项，比如你期望的每个分片大小，等等。配置项的具体细节可以参考“配置项”的第二讲哈~

2

辰
2021-04-16

老师，结合这一节内容和之前的，自定义数据结构其实和hive中的表关系不大，不涉及自定义，那么这时候是不是就可以把存储自定义这部分的内存匀出来给到统一内存身上

展开

作者回复: 对，如果没有自定义的数据结构，可以把spark.memory.fraction调高，多给Spark的执行任务和缓存任务分配些内存空间。

毕竟，没有自定义结构，User memory留着也是浪费。

4 1

aof
2021-05-04

老师讲的预估内存占用非常细，但就像老师给出的第二题中说的那样，如果Spark应用程序中的计算逻辑很多，这样预估自然是很精确，但是会花费大量时间，成本巨大！分享一下自己平时粗略估算内存占用的方法（如有不对，还望老师纠正）：
1、Storage Memory估算：将要缓存到内存的RDD/Dataset/Dataframe或广播变量进行cache，然后在Spark WEBUI的Storage标签页下直接查看所有的内存占用，大致就对应...
展开

Fendora范东_
2021-04-16

1.C/C++党，阅读修改spark源码无障碍，虽然有些scala高级语法叫不上名，不影响理解源码。就是不懂java。。😅
2.我觉得应该根据使用场景直接预估。比如如果仅仅使用sql，那么fraction和storageFraction尽量调小就行;如果是使用DataSet API较多，省略保留内存情况下，#user,#storage,#execution各占1/3，然后根据代码中自定义数据结构和cache方法在代码总行数中出现...
展开

作者回复: 哈哈，Scala玩得那么溜，修改源码无障碍，咋会不懂Java、JVM，你咋做到的？教教我，哈哈~

第二个不太对哈，首先和API没什么关系，不论是SQL、DF还是DS，都需要按照比例划分不同的内存区域。

再者，不能按照代码行数去估算内存消耗，这个不科学，主要是两者之间没有必然联系，用统计的话说，两者不是正相关的。

不过，3个区域各自占1/3，但是不失为一个不错的初始值，不过这也仅仅是个开始，only a start point。还是要结合作业对于内存的消耗占比，相应地调整不同区域的空间大小。

提示：可以通过Spark UI的一些指标，来粗略地估计“数据内存放大倍数”，然后再去预估不同

空间的消耗占比。（好吧，我又开始挖坑了，Spark UI后面一定找机会直播一把，不过不妨先熟悉、了解下Spark UI，先有个整体的认知）