



下载APP



## 07 | 内存管理基础：Spark如何高效利用有限的内存空间？

2021-03-29 吴磊

Spark性能调优实战

[进入课程 >](#)**讲述：吴磊**

时长 20:08 大小 18.44M



你好，我是吴磊。

对于 Spark 这样的内存计算引擎来说，内存的管理与利用至关重要。业务应用只有充分利用内存，才能让执行性能达到最优。

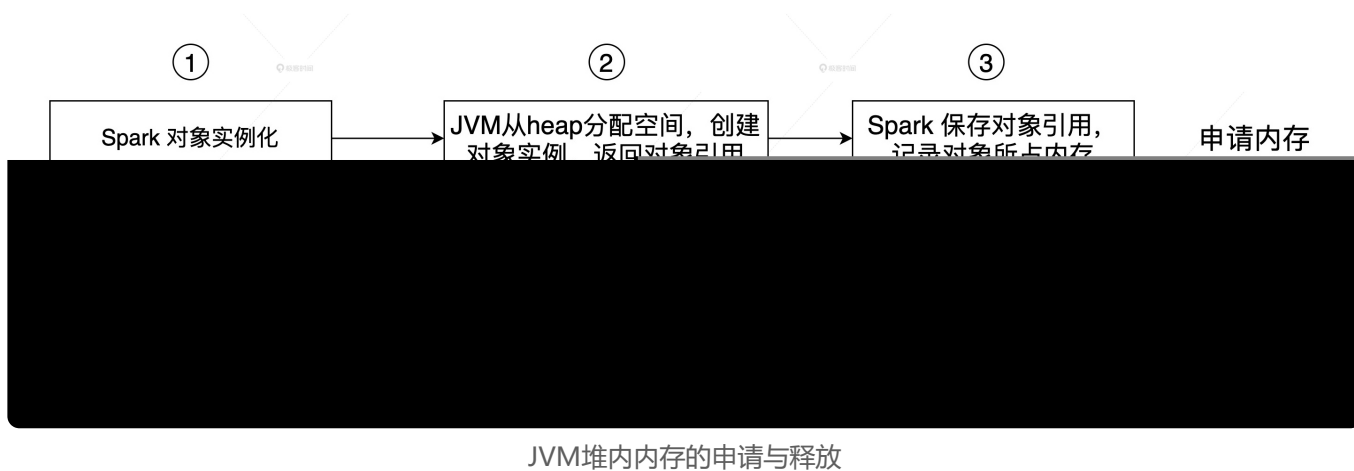
那么，你知道 Spark 是如何使用内存的吗？不同的内存区域之间的关系是什么，它们又是如何划分的？今天这一讲，我就结合一个有趣的小故事，来和你深入探讨一下 Spark 内存管理的基础知识。



### 内存的管理模式

在管理方式上，Spark 会区分**堆内内存**（On-heap Memory）和**堆外内存**（Off-heap Memory）。这里的“堆”指的是 JVM Heap，因此堆内内存实际上就是 Executor JVM 的堆内存；堆外内存指的是通过 Java Unsafe API，像 C++ 那样直接从操作系统中申请和释放内存空间。

**其中，堆内内存的申请与释放统一由 JVM 代劳。**比如说，Spark 需要内存来实例化对象，JVM 负责从堆内分配空间并创建对象，然后把对象的引用返回，最后由 Spark 保存引用，同时记录内存消耗。反过来也是一样，Spark 申请删除对象会同时记录可用内存，JVM 负责把这样的对象标记为“待删除”，然后再通过垃圾回收（Garbage Collection，GC）机制将对象清除并真正释放内存。



在这样的管理模式下，Spark 对内存的释放是有延迟的，因此，当 Spark 尝试估算当前可用内存时，很有可能会高估堆内的可用内存空间。

**堆外内存则不同，Spark 通过调用 Unsafe 的 allocateMemory 和 freeMemory 方法直接在操作系统内存中申请、释放内存空间**，这听上去是不是和 C++ 管理内存的方式很像呢？这样的内存管理方式自然不再需要垃圾回收机制，也就免去了它带来的频繁扫描和回收引入的性能开销。更重要的是，空间的申请与释放可以精确计算，因此 Spark 对堆外可用内存的估算会更精确，对内存的利用率也更有把握。

为了帮助你更轻松地了解这个过程，我来给你讲一个小故事。

## 地主招租（上）：土地划分

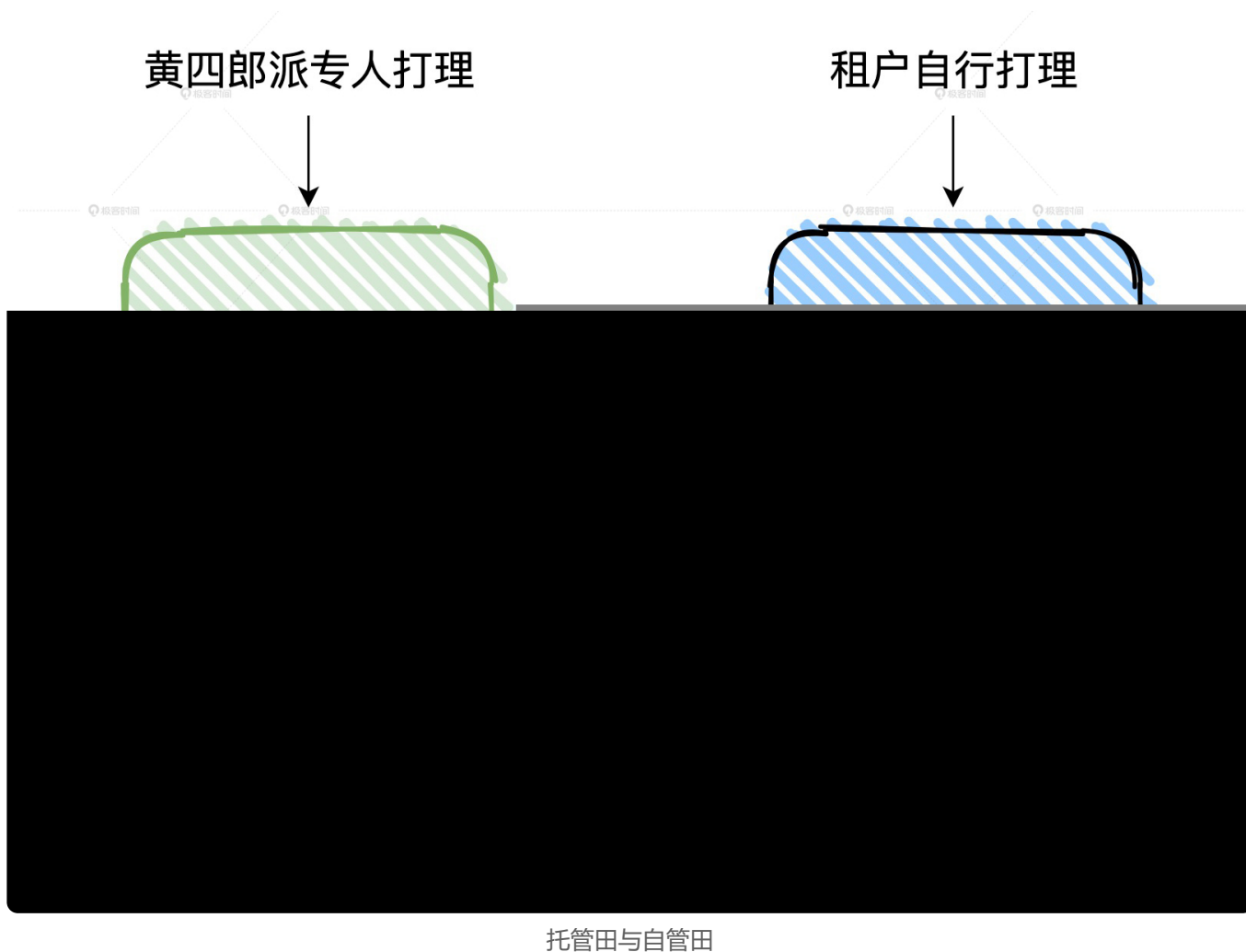
很久以前，燕山脚下有一个小村庄，村里有个地主，名叫黄四郎，四郎家有良田千顷，方圆数百里都是他的田地。黄四郎养尊处优，自然不会亲自下地种田，不过这么多田地也不

能就这么荒着。于是，他想了个办法，既不用亲自动手又能日进斗金：收租子！

黄四郎虽然好吃懒做，但在管理上还是相当有一套的，他把田地划分为两块，一块叫“托管田”，另一块叫“自管田”。

我们知道，庄稼**丰收之后，田地需要翻土、整平、晾晒**，来年才能种下一茬庄稼。那么，托管田指的就是丰收之后，由黄四郎派专人帮你搞定翻土、整平这些琐事，不用你操心。相应的，自管田的意思就是庄稼你自己种，秋收之后的田地也得你自己收拾。

**毫无疑问，对租户来说托管田更省心一些，自管田更麻烦。**当然了，相比自管田，托管田的租金自然更高。



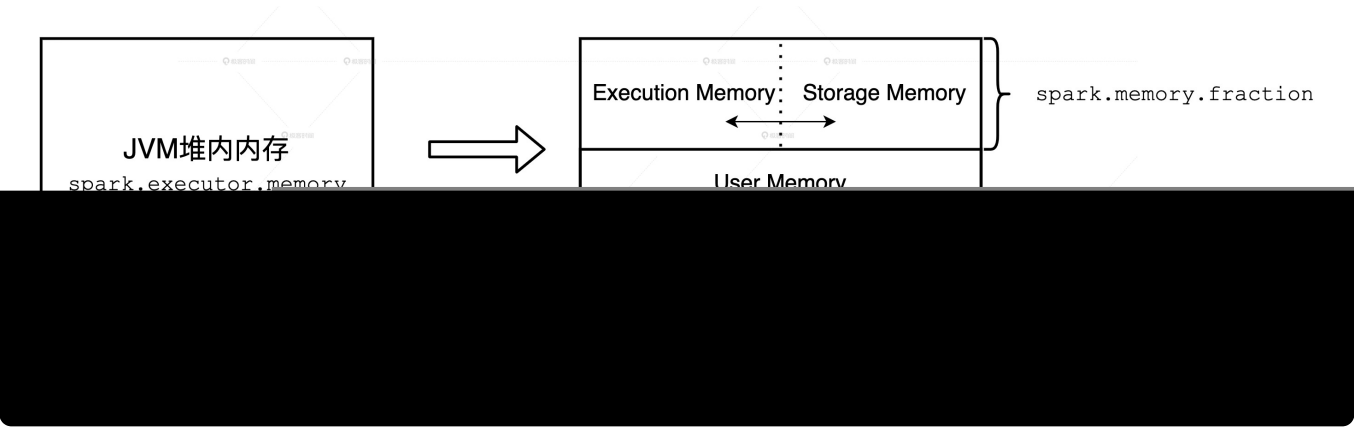
那么，这个故事中黄四郎的托管田就是内存管理中的堆内内存，自管田类比的则是堆外内存，田地的翻土、整平这些操作实际上就是 JVM 中的 GC。这样类比起来是不是更好理解了呢？

## 内存区域的划分

故事先讲到这儿，让我们暂时先回到 Spark 的内存管理上。现在，我们知道了 Spark 内存管理有堆内和堆外两种模式，那 Spark 又是怎么划分内存区域的呢？

我们先来说说堆外内存。Spark 把堆外内存划分为两块区域：一块用于执行分布式任务，如 Shuffle、Sort 和 Aggregate 等操作，这部分内存叫做 **Execution Memory**；一块用于缓存 RDD 和广播变量等数据，它被称为 **Storage Memory**。

堆内内存的划分方式和堆外差不多，Spark 也会划分出用于执行和缓存的两份内存空间。不仅如此，Spark 在堆内还会划分出一片叫做 **User Memory** 的内存空间，它用于存储开发者自定义数据结构。



不同内存区域的划分

除此之外，Spark 在堆内还会预留出一小部分内存空间，叫做 **Reserved Memory**，它被用来存储各种 Spark 内部对象，例如存储系统中的 BlockManager、DiskBlockManager 等等。

对于性能调优来说，我们在前三块内存的利用率上有比较大的发挥空间，因为业务应用主要消耗的就是它们，也即 Execution memory、Storage memory 和 User memory。而预留内存我们却动不得，因为这块内存仅服务于 Spark 内部对象，业务应用不会染指。

好了，不同内存区域的划分与计算，我也把它们总结到了下面的表格中，方便你随时查阅。

内存区域划分	堆内内存	堆外内存 (spark.memory.offHeap.enabled)
内存空间划分		offHeap

执行与缓存内存

在所有的内存区域中，最重要的无疑是缓存内存和执行内存，而内存计算的两层含义也就是数据集缓存和 Stage 内的流水线计算，对应的就是 Storage Memory 和 Execution Memory。

在 Spark 1.6 版本之前，Execution Memory 和 Storage Memory 内存区域的空间划分是静态的，一旦空间划分完毕，不同内存区域的用途就固定了。也就是说，即便你没有缓存任何 RDD 或是广播变量，Storage Memory 区域的空闲内存也不能用来执行 Shuffle 中的映射、排序或聚合等操作，因此宝贵的内存资源就被这么白白地浪费掉了。

考虑到静态内存划分潜在的空间浪费，在 1.6 版本之后，Spark 推出了统一内存管理模式。**统一内存管理指的是 Execution Memory 和 Storage Memory 之间可以相互转化**，尽管两个区域由配置项 spark.memory.storageFraction 划定了初始大小，但在运行时，结合任务负载的实际情况，Storage Memory 区域可能被用于任务执行（如 Shuffle），Execution Memory 区域也有可能存储 RDD 缓存。

但是，我们都知道，执行任务相比缓存任务，在内存抢占上有着更高的优先级。那你有没有想过这是为什么呢？接下来，就让我们带着“打破砂锅问到底”的精神，去探索其中更深层次的原因。

首先，执行任务主要分为两类：**一类是 Shuffle Map 阶段的数据转换、映射、排序、聚合、归并等操作；另一类是 Shuffle Reduce 阶段的数据排序和聚合操作。它们所涉及的**



## 数据结构，都需要消耗执行内存。

我们可以先假设，执行任务与缓存任务在内存抢占上遵循“公正、公平和公开”的三原则。也就是说，不论谁抢占了對方的内存，当对方有需要时都会立即释放。比如说，刚开始双方的预设比例是五五开，但因为缓存任务在应用中比较靠后的位置，所以执行任务先占据了 80% 的内存空间，当缓存任务追赶上来之后，执行任务就需要释放 30% 的内存空间还给缓存任务。

这种情况下会发生什么？假设集群范围内总共有 80 个 CPU，也就是集群在任意时刻的并行计算能力是 80 个分布式任务。在抢占了 80% 内存的情况下，80 个 CPU 可以充分利用，每个 CPU 的计算负载都是比较饱满的，计算完一个任务，再去计算下一个任务。

但是，由于有 30% 的内存要归还给缓存任务，这意味着有 30 个并行的执行任务没有内存可用。也就是说会有 30 个 CPU 一直处在 I/O wait 的状态，没法干活！宝贵的 CPU 计算资源就这么白白地浪费掉了，简直是暴殄天物。

因此，相比于缓存任务，执行任务的抢占优先级一定要更高。说了这么多，我们为什么要弄清楚其中的原因呢？我认为，只有弄清楚抢占优先级的背后逻辑，我们才能理解为什么要同时调节 CPU 和内存的相关配置，也才有可能做到不同硬件资源之间的协同与平衡，这也是我们进行性能调优要达到的最终效果。

不过，即使执行任务的抢占优先级更高，但它们在抢占内存的时候一定也要遵循某些规则。那么，这些规则具体是什么呢？下面，咱们就接着以地主招租的故事为例，来说说 Execution memory 和 Storage memory 之间有哪些有趣的规则。

### 地主招租（下）：租地协议

黄四郎招租的告示贴出去没多久，村子里就有两个年富力强的小伙子来租种田地。一个叫黄小乙，是黄四郎的远房亲戚，前不久来投奔黄四郎。另一个叫张麻子，虽是八辈贫农，小日子过得也算是蒸蒸日上。张麻子打算把田地租过来种些小麦、玉米这样的庄稼。黄小乙就不这么想，这小子挺有商业头脑，他把田地租过来准备种棉花、咖啡这类经济作物。

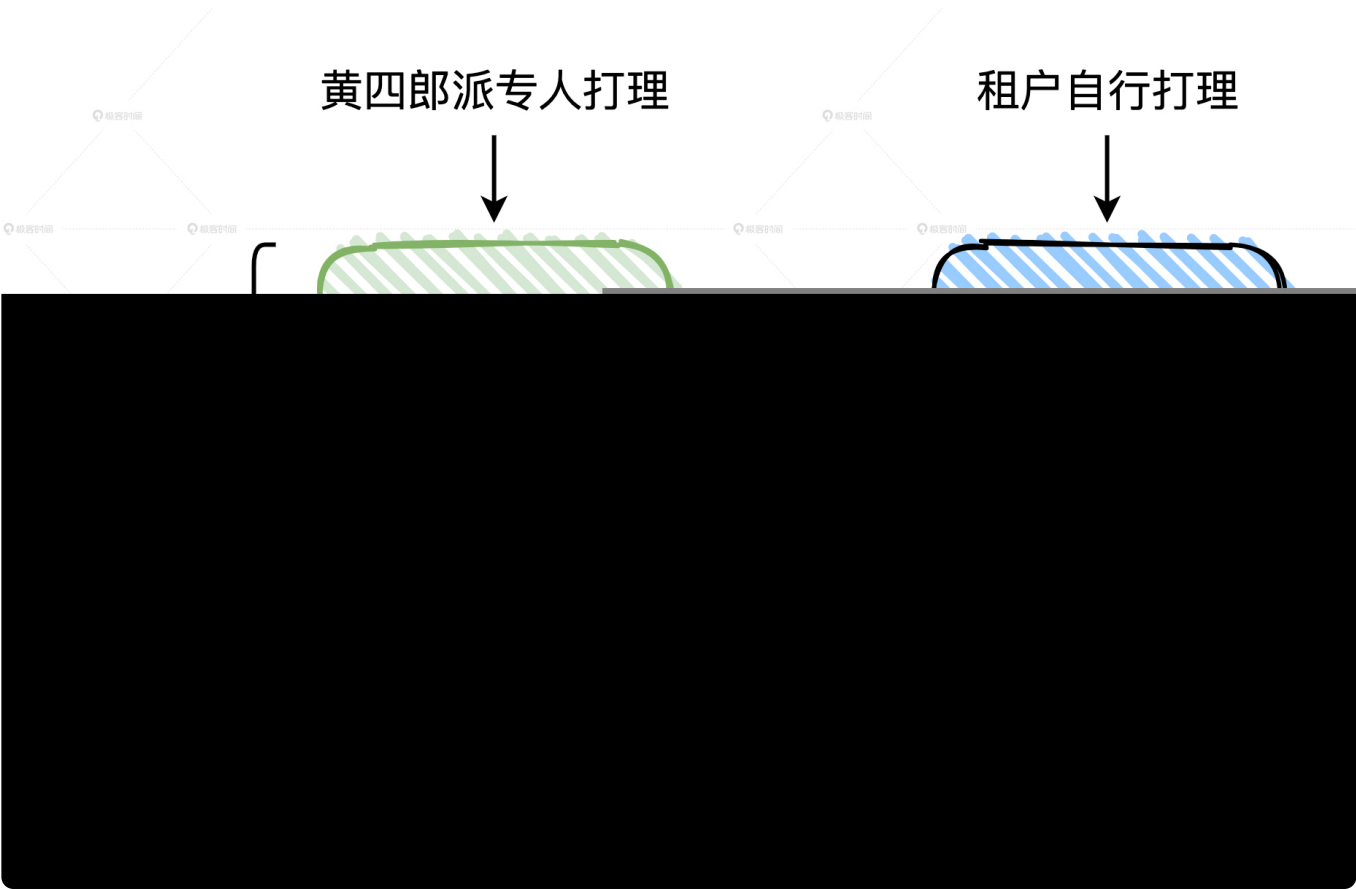
两个人摩拳擦掌都想干出一番事业，恨不得把黄四郎的地全都包圆！地不愁租，黄四郎自然是满心欢喜，但烦恼也接踵而至：“既要照顾小乙这孩子，又不能打击麻子的积极性，得想个万全之策”。

于是，他眼珠一转，计上心来：“按理说呢，咱们丈量土地之后，应该在你们中间划一道实线，好区分田地的归属权。不过呢，毕竟麻子你是本村的，小乙远道而来，远来即是客嘛！咱们对小乙还是得多少照顾着点”。张麻子心生不悦：“怎么照顾？”

黄四郎接着说：“很简单，把实线改为虚线，多劳者多得。原本呢，你们应该在分界线划定的那片田地里各自劳作。不过呢，你们二人的进度各不相同嘛，所以，勤奋的人，自己的田地种满了之后，可以跨过分界线，去占用对方还在空着的田地。”

黄小乙不解地问：“四舅，这不是比谁种得快吗？也没对我特殊照顾啊！”张麻子眉间也拧了个疙瘩：“如果种得慢的人后来居上，想要把被占的田地收回去，到时候该怎么办呢？”

黄四郎得意道：“刚才说了，咱们多多照顾小乙。所以如果麻子勤快、干活也快，先占了小乙的地，种上了小麦、玉米，小乙后来居上，想要收回自己的地，那么没说的，麻子得把多占的地让出来。不管庄稼熟没熟，麻子都得把地铲平，还给人家小乙种棉花、咖啡”。



黄小乙与张麻子的占地协议

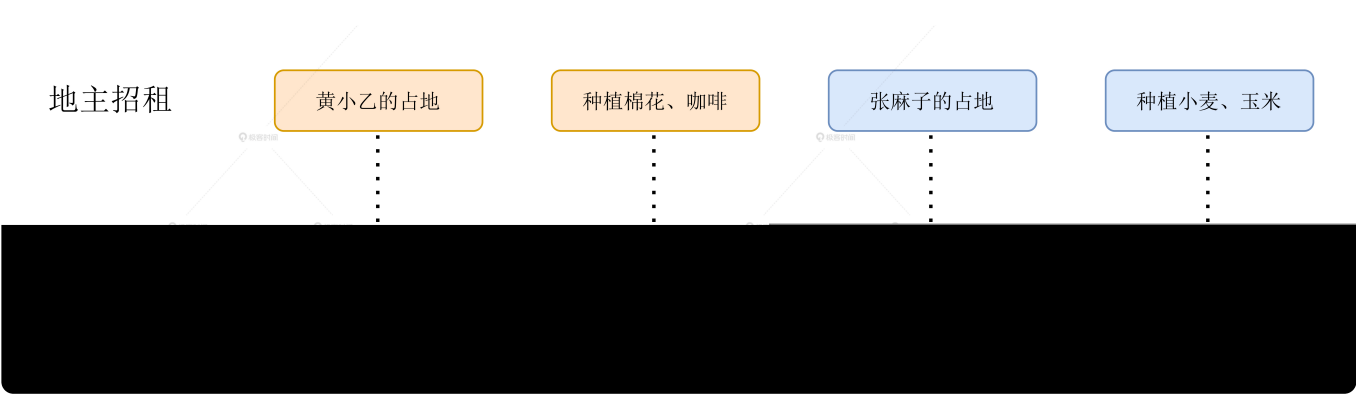
黄四郎偷眼看了看两人的反应，继续说：“反过来，如果小乙更勤快，先占了麻子的地，麻子后来居上，想要收回，这个时候，咱们就得多照顾照顾小乙。小乙有权继续占用麻子的地，直到地上种的棉花、咖啡都丰收了，再把多占的地让出来。你们二位看怎么样？”

黄小乙听了大喜。张麻子虽然心里不爽，但也清楚黄四郎和黄小乙之间的亲戚关系，也不好再多说什么，心想：“反正我勤快些，先把地种满也就是了”。于是，三方击掌为誓，就此达成协议。

好啦，地主招租的故事到这里就讲完了。不难发现，黄小乙的地类比的是 Execution Memory，张麻子的地其实就是 Storage Memory。他们之间的协议其实就是 Execution Memory 和 Storage Memory 之间的抢占规则，一共可以总结为 3 条：

- 如果对方的内存空间有空闲，双方就都可以抢占；
- 对于 RDD 缓存任务抢占的执行内存，当执行任务有内存需要时，RDD 缓存任务必须立即归还抢占的内存，涉及的 RDD 缓存数据要么落盘、要么清除；
- 对于分布式计算任务抢占的 Storage Memory 内存空间，即便 RDD 缓存任务有收回内存的需要，也要等到任务执行完毕才能释放。

同时，我也把这个例子中的关键内容和 Spark 之间的对应关系总结在了下面，希望能帮助你加深印象。




地主招租与Spark的类比关系

## 从代码看内存消耗

说完了理论，接下来，咱们再从实战出发，用一个小例子来直观地感受一下，应用中代码的不同部分都消耗了哪些内存区域。



示例代码很简单，目的是读取 words.csv 文件，然后对其中指定的单词进行统计计数。

 复制代码

```
1 val dict: List[String] = List("spark", "scala")
2 val words: RDD[String] = sparkContext.textFile("~/words.csv")
3 val keywords: RDD[String] = words.filter(word => dict.contains(word))
4 keywords.cache
5 keywords.count
6 keywords.map((_, 1)).reduceByKey(_ + _).collect
```

整个代码片段包含 6 行代码，咱们从上到下逐一分析。

首先，第一行定义了 dict 字典，这个字典在 Driver 端生成，它在后续的 RDD 调用中会随着任务一起分发到 Executor 端。第二行读取 words.csv 文件并生成 RDD words。**第三行很关键，用 dict 字典对 words 进行过滤，此时 dict 已分发到 Executor 端，Executor 将其存储在堆内存中，用于对 words 数据分片中的字符串进行过滤。Dict 字典属于开发者自定义数据结构，因此，Executor 将其存储在 User Memory 区域。**

接着，第四行和第五行用 cache 和 count 对 keywords RDD 进行缓存，以备后续频繁访问，分布式数据集的缓存占用的正是 Storage Memory 内存区域。在最后一行代码中，我们在 keywords 上调用 reduceByKey 对单词分别计数。我们知道，reduceByKey 算子会引入 Shuffle，而 Shuffle 过程中所涉及的内部数据结构，如映射、排序、聚合等操作所仰仗的 Buffer、Array 和 HashMap，都会消耗 Execution Memory 区域中的内存。

不同代码与其消耗的内存区域，我都整理到了下面的表格中，方便你查看。

代码行数	代码片段	内存区域
1	val dict: List[String] = List("spark", "scala")	Driver Memory
2	val words: RDD[String] = sparkContext.textFile("~/words.csv")	Storage Memory
3	val keywords: RDD[String] = words.filter(word => dict.contains(word))	User Memory
4	keywords.cache	Storage Memory
5	keywords.count	Storage Memory
6	keywords.map((_, 1)).reduceByKey(_ + _).collect	Execution Memory

## 小结

深入理解内存管理的机制，有助于我们充分利用应用的内存，提升其执行性能。今天，我们重点学习了内存管理的基础知识。

**首先是内存的管理方式。**Spark 区分堆内内存和堆外内存：对于堆外内存来说，Spark 通过调用 Java Unsafe 的 `allocateMemory` 和 `freeMemory` 方法，直接在操作系统内存中申请、释放内存空间，管理成本较高；对于堆内内存来说，无需 Spark 亲自操刀而是由 JVM 代理。但频繁的 JVM GC 对执行性能来说是一大隐患。另外，Spark 对堆内内存占用的预估往往不够精确，高估可用内存往往会为 OOM 埋下隐患。

**其次是统一内存管理，以及 Execution Memory 和 Storage Memory 之间的抢占规则。**它们就像黄四郎招租故事中黄小乙和张麻子的田地，抢占规则就像他们之间的占地协议，主要可以分为 3 条：

如果对方的内存空间有空闲，那么双方都可以抢占；

对 RDD 缓存任务抢占的执行内存，当执行任务有内存需要时，RDD 缓存任务必须立即归还抢占的内存，其中涉及的 RDD 缓存数据要么落盘、要么清除；

对分布式计算任务抢占的 Storage Memory 内存空间，即便 RDD 缓存任务有收回内存的需要，也要等到任务执行完毕才能释放。

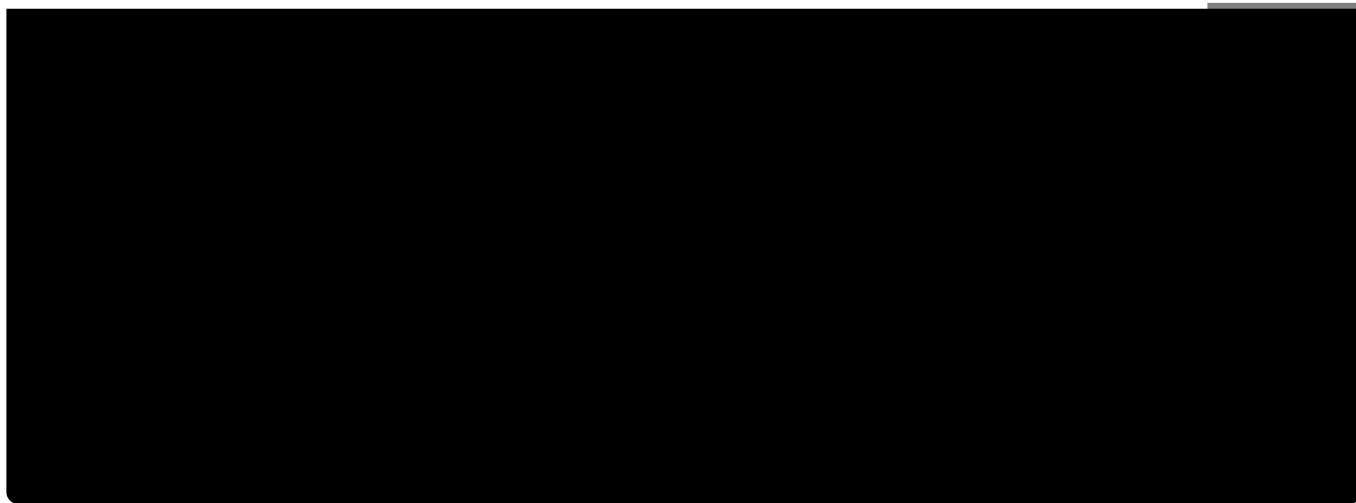
**最后是不同代码对不同内存区域的消耗。**内存区域分为 Reserved Memory、User Memory、Execution Memory 和 Storage Memory。其中，Reserved Memory 用于存储 Spark 内部对象，User Memory 用于存储用户自定义的数据结构，Execution Memory 用于分布式任务执行，而 Storage Memory 则用来容纳 RDD 缓存和广播变量。

好了，这些就是内存管理的基础知识。当然了，与内存相关的话题还有很多，比如内存溢出、RDD 缓存、内存利用率，以及执行内存的并行计算等等。在性能调优篇，我还会继续从内存视角出发，去和你探讨这些话题。

## 每日一练

1. 你知道启用 off-heap 之后，Spark 有哪些计算环节可以利用到堆外内存？你能列举出一些例子吗？

2. 相比堆内内存，为什么在堆外内存中，Spark 对于内存占用量的预估更准确？
3. 结合我在下面给定的配置参数，你能分别计算不同内存区域（Reserved、User、Execution、Storage）的具体大小吗？



期待在留言区看到你的思考和答案，我们下一讲见！

提建议

12.12 大促

# 每日一课 VIP 年卡

10分钟，解决你的技术难题

¥159/年 ¥365/年

每日一课  
VIP 年卡

仅3天，【点击】图片，立即抢购 >>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 存储系统：空间换时间，还是时间换空间？

下一篇 08 | 应用开发三原则：如何拓展自己的开发边界？

## 精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。