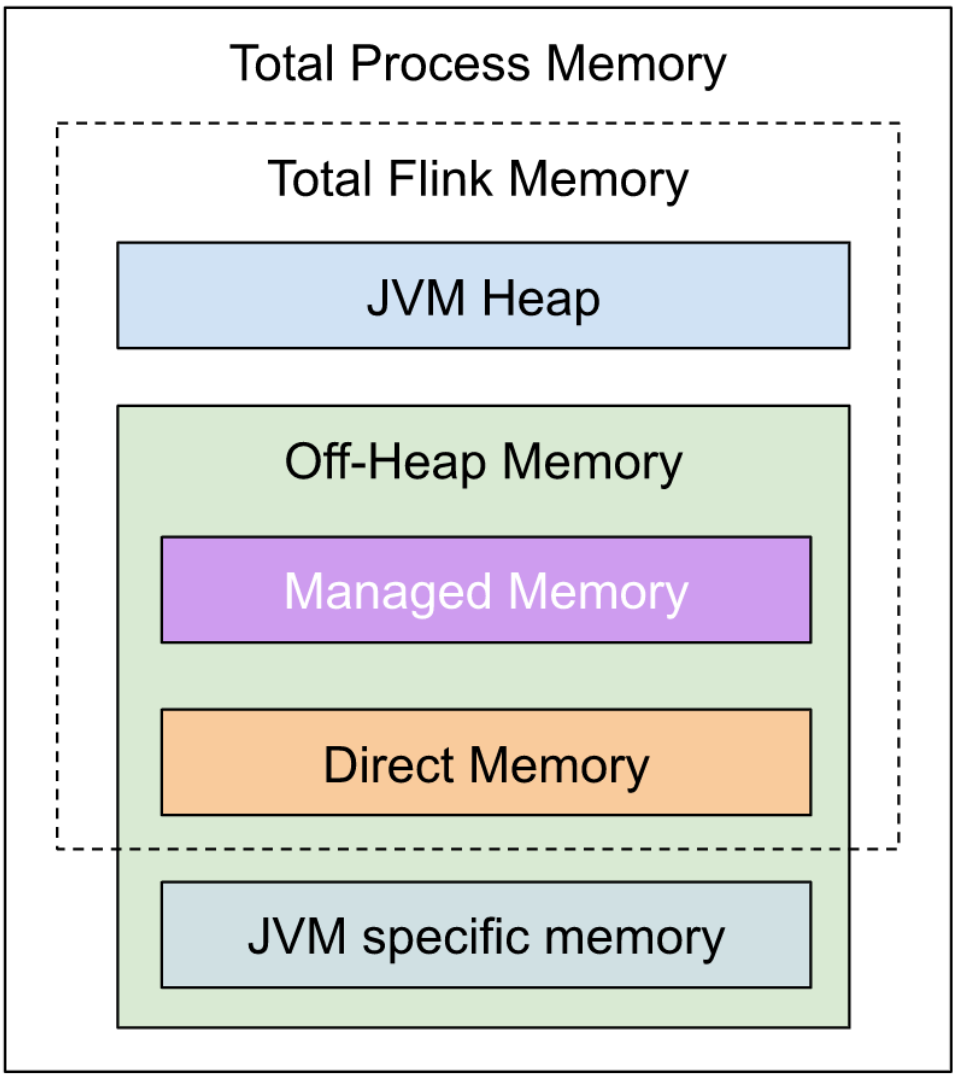
Apache Flink 1.10 对 TaskManager 的内存模型和 Flink 应用程序的配置选项进行了重大变更。这些最近引入的更改做到了对内存消耗提供了严格的控制，使得 Flink 在各种部署环境（例如 Kubernetes，Yarn，Mesos）更具有适应能力，

在本文中，我们将介绍 Flink 1.10 中的内存模型、如何设置和管理 Flink 应用程序的内存消耗以及社区在最新的 Apache Flink Release 版本中的变化。

Flink 内存模型的介绍

对 Apache Flink 的内存模型有清晰的了解，可以使您更有效地管理各种情况下的资源使用情况。 下图描述了 Flink 中的主要内存组件：



TaskManager 进程是一个 JVM 进程，从较高的角度来看，它的内存由 JVM Heap 和 Off-Heap 组成。这些类型的内存由 Flink 直接使用，或由 JVM 用于其特定目的（比如元空间 metaspace）。

Flink 中有两个主要的内存使用者：

- 用户代码中的作业 task 算子

- Flink 框架本身的内部数据结构、网络缓冲区 (Network Buffers)等

请注意，用户代码可以直接访问所有的内存类型：JVM 堆、Direct 和 Native 内存。因此，Flink 不能真正控制其分配和使用。但是，有两种供作业 Task 使用并由 Flink 严格控制的 Off-Heap 内存，它们分别是：

- Managed Memory (Off-Heap)
- 网络缓冲区 (Network Buffers)

网络缓冲区 (Network Buffers) 是 JVM Direct 内存的一部分，分配在算子和算子之间用于进行用户数据的交换。

怎么去配置 Flink 的内存

在最新 Flink 1.10 版本中，为了提供更好的用户体验，框架提供了内存组件的高级和细粒度调优。在 TaskManager 中设置内存基本上有三种选择。

前两个（也是最简单的）选择是需要你配置以下两个选项之一，以供 TaskManager 的 JVM 进程使用的总内存：

- Total Process Memory：Flink Java 应用程序（包括用户代码）和 JVM 运行整个进程所消耗的总内存。
- Total Flink Memory：仅 Flink Java 应用程序消耗的内存，包括用户代码，但不包括 JVM 为其运行而分配的内存。

如果是以 standalone 模式部署，则建议配置 Total Flink Memory，在这种情况下，显式声明为 Flink 分配多少内存是一种常见的做法，而外部 JVM 开销却很少。

对于在容器化环境（例如 Kubernetes，Yarn 或 Mesos）中部署 Flink 的情况，建议配置 Total Process Memory，因为它表示所请求容器的总内存大小，容器化环境通常严格执行此内存限制。

其余的内存组件将根据其默认值或其他已配置的参数自动进行调整。Flink 还会检查整体一致性。你可以在相应的文档中找到有关不同内存组件的更多信息。 此外，你可以使用 FLIP-49 的配置电子表格尝试不同的配置选项，并根据你的情况检查相应的结果。

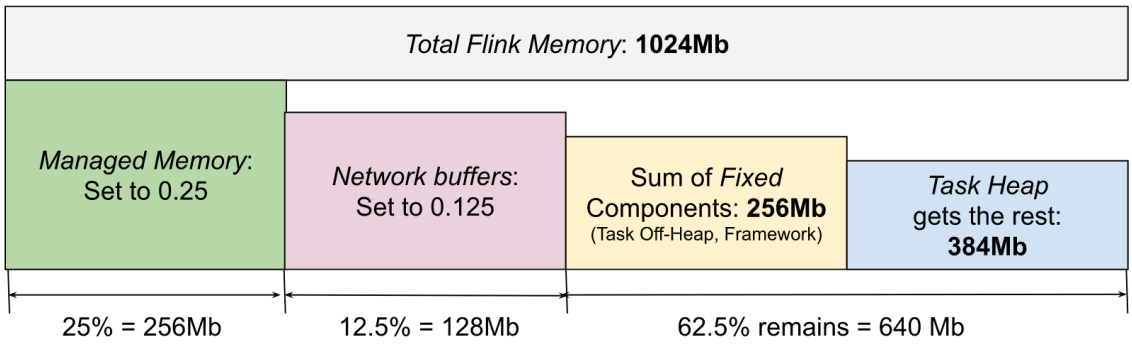
如果要从 1.10 之前的 Flink 版本进行迁移，我们建议你遵循 Flink 文档的迁移指南中的步骤。

其他组件

在配置 Flink 的内存时，可以使用相应选项的值固定不同内存组件的大小，也可以使用多个选项进行调整。下面我们提供有关内存设置的更多信息。

按比例细分 Total Flink Memory

此方法允许按比例细分 Total Flink Memory，其中 Managed Memory（如果未明确设置）和网络缓冲区可以占用一部分。然后，将剩余的内存分配给 Task Heap（如果未明确设置）和其他固定的 JVM Heap 和 Off-Heap 组件。下图是这种设置的示例：



Flink: Example of Memory Setup

请注意：

Flink 会校验分配的 Network Memory 大小在其最小值和最大值之间，否则 Flink 的启动会失败，最大值和最小值的限制具有默认值，这些默认值是可以被相应的配置选项覆盖。

通常，Flink 将配置的占比分数视为提示。在某些情况下，真正分配的值可能与占比分数不匹配。例如，如果将 Total Flink Memory 和 Task Heap 配置为固定值，则 Managed Memory 将获得一定比例的内存，而 Network Memory 将获得可能与该比例不完全匹配的剩余内存。

控制容器内存限制的更多提示

堆内存和 direct 内存的使用是由 JVM 管理的。在 Apache Flink 或其用户应用程序中，还有许多其他 native 内存消耗的可能来源，它们不是由 Flink 或 JVM 管理的。通常很难控制它们的限制大小，这会使调试潜在的内存泄漏变得复杂。

如果 Flink 的进程以不受管理的方式分配了过多的内存，则在容器化环境中通常可能导致 TaskManager 容器会被杀死。在这种情况下，可能很难理解哪种类型的内存消耗已超过其限制。 Flink 1.10 引入了一些特定的调整选项，以清楚地表示这些组件。 尽管 Flink 不能始终严格执行严格的限制和界限，但此处的想法是明确计划内存使用情况。 下面我们提供一些示例，说明内存设置如何防止容器超出其内存限制：

- **RocksDB 状态不能太大**：RocksDB 状态后端的内存消耗是在 Managed Memory 中解决的。RocksDB 默认情况下遵守其限制（仅自 Flink 1.10 起）。你可以增加 Managed Memory 的大小以提高 RocksDB 的性能，也可以减小 Managed Memory 的大小以节省资源。
- **用户代码或其依赖项会消耗大量的 off-heap 内存**：调整 Task Off-Heap 选项可以为用户代码或其任何依赖项分配额外的 direct 或 native 内存。Flink 无法控制 native 分配，但它设置了 JVM Direct 内存分配的限制。Direct 内存限制由 JVM 强制执行。
- **JVM metaspace 需要额外的内存**：如果遇到 `OutOfMemoryError：Metaspace` ，Flink 提供了一个增加其限制的选项，并且 JVM 将确保不超过该限制。

- **JVM 需要更多内部内存**：无法直接控制某些类型的 JVM 进程分配，但是 Flink 提供了 JVM 开销选项。这些选项允许声明额外的内存量，这些内存是为这些分配所预期的，并且未被其他选项覆盖。

结论

最新的 Flink 版本（Flink 1.10）对 Flink 的内存配置进行了一些重大更改，从而可以比以前更好地管理应用程序内存和调试 Flink。未来 JobManager 的内存模型也会采取类似的更改，可以参考 FLIP-116，因此请继续关注即将发布的新版本中新增的功能。如果你对社区有任何建议或问题，我们建议你注册 Apache Flink 邮件列表并参与其中的讨论。

> 博客英文地址：https://flink.apache.org/news/2020/04/21/memory-management-improvements-flink-1.10.html

> 作者: Andrey Zagrebin

> 本文翻译作者：zhisheng

> 翻译后首发地址：http://www.54tianzhisheng.cn/2020/05/16/flink-taskmanager-memory-model/