# 概述

有状态的函数和运算符在处理单个元素/事件的过程中存储数据，从而使状态成为任何类型的更精细操作的关键构建块。

包括：

- 当应用程序搜索某些事件模式时，状态将存储到目前为止遇到的事件序列。

- 在每分钟/小时/天汇总事件时，状态将保留待处理的汇总。

- 在数据点流上训练机器学习模型时，状态保持模型参数的当前版本。

- 当需要管理历史数据时，该状态允许有效访问过去发生的事件。

Flink需要了解状态，以便使用检查点使状态容错并允许流应用程序的保存点。

状态还允许重新缩放Flink应用程序，这意味着Flink负责在并行实例之间重新分配状态。

Flink 的可查询状态功能允许在运行时从Flink外部访问状态。

使用状态时，阅读有关Flink的状态后端的信息也可能会很有用。Flink提供了不同的状态后端，用于指定状态的存储方式和位置。状态可以位于Java的堆或堆外。根据状态后端，Flink还可以管理应用程序的状态，这意味着Flink可以处理内存管理（如有必要，可以落到磁盘上）以允许应用程序保持很大的状态。可以配置状态后端，而无需更改应用程序逻辑。

# 工作状态

Flink中有两种基本状态：Keyed State和Operator State。

1.键控状态

键控状态始终与键相关，并且只能在KeyedStream中使用函数和运算符。

可以将\*\*键控状态\*\*视为已分区或分片的操作状态，每个键仅具有一个状态分区。每个键状态在逻辑上都绑定到<parallel-operator-instance，key>的唯一组合，并且由于每个键都完全属于键操作符的一个并行实例，因此我们可以简单地将其视为<operator，key >。

Keyed State被进一步组织成所谓的Key Groups。密钥组是Flink可以重新分配密钥状态的原子单位；与定义的最大并行度完全相同的键组数。在执行期间，键控运算符的每个并行实例都使用一个或多个键组的键。

2.操作状态

使用运算符状态（或非键状态），每个运算符状态都绑定到一个并行运算符实例。在kafka连接器使用Flink运营状况的一个很好激励的例子。Kafka使用者的每个并行实例都维护一个主题分区和偏移量的映射作为其操作状态。

当更改并行性时，操作状态接口支持在并行操作实例之间重新分配状态。可以有不同的方案来进行此重新分配。

# 原始状态和托管状态

键控状态和操作状态以两种形式存在：托管状态和原始状态。

托管状态用Flink运行时控制的数据结构表示，例如内部哈希表或RocksDB。例如“ ValueState”，“ ListState”等。Flink的运行时对状态进行编码，并将其写入检查点。

原始状态是操作保留其自己的数据结构的状态。被检查点时，它们仅将字节序列写入检查点。Flink对状态的数据结构一无所知，只看到原始字节。

所有数据流功能都可以使用托管状态，但是原始状态接口只能在实现运算符时使用。建议使用托管状态（而不是原始状态），因为有了托管状态，Flink可以在更改并行性时自动重新分配状态，并且还可以进行更好的内存管理。

注意：如果管理状态需要自定义序列化逻辑，请参阅相应的指南，以确保将来的兼容性。Flink的默认序列化器不需要特殊处理。