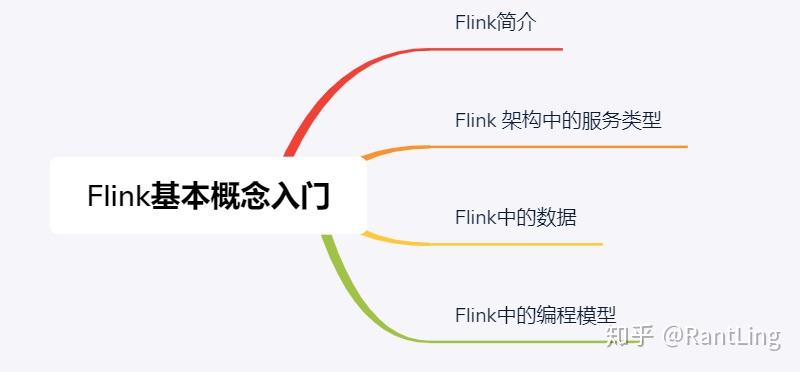
**Flink(一)-基本概念**

这篇文章主要按照以下思路，简单的交流一下Flink的基本概念和用途。自知资历尚浅，见闻有限，如有纰漏还望指正！

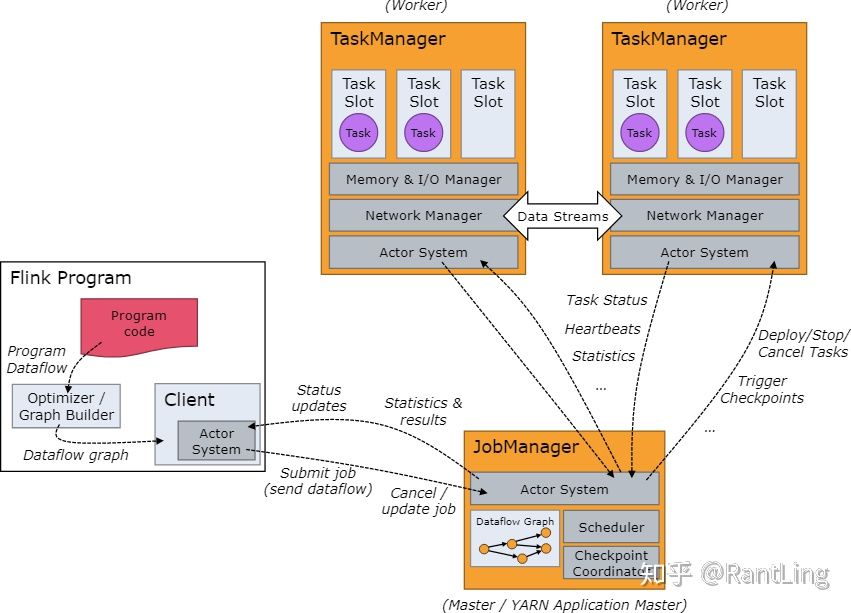


**1. Flink 简介**

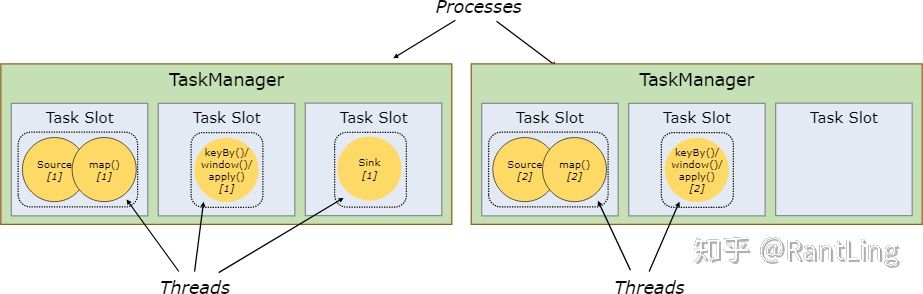
在当前的互联网用户，设备，服务等激增的时代下，其产生的数据量已不可同日而语了。各种业务场景都会有着大量的数据产生，如何对这些数据进行有效地处理是很多企业需要考虑的问题。以往我们所熟知的Map Reduce，Storm，Spark等框架可能在某些场景下已经没法完全地满足用户的需求，或者是实现需求所付出的代价，无论是代码量或者架构的复杂程度可能都没法满足预期的需求。新场景的出现催产出新的技术，Flink即为实时流的处理提供了新的选择。Apache Flink就是近些年来在社区中比较活跃的分布式处理框架，加上阿里在中国的推广，相信它在未来的竞争中会更具优势。 Flink的产生背景不过多介绍，感兴趣的可以Google一下。Flink相对简单的编程模型加上其高吞吐、低延迟、高性能以及支持exactly-once语义的特性，让它在工业生产中较为出众。相信正如很多博客资料等写的那样"Flink将会成为企业内部主流的数据处理框架，最终成为下一代大数据处理标准。"

**2. Flink 架构中的服务类型**

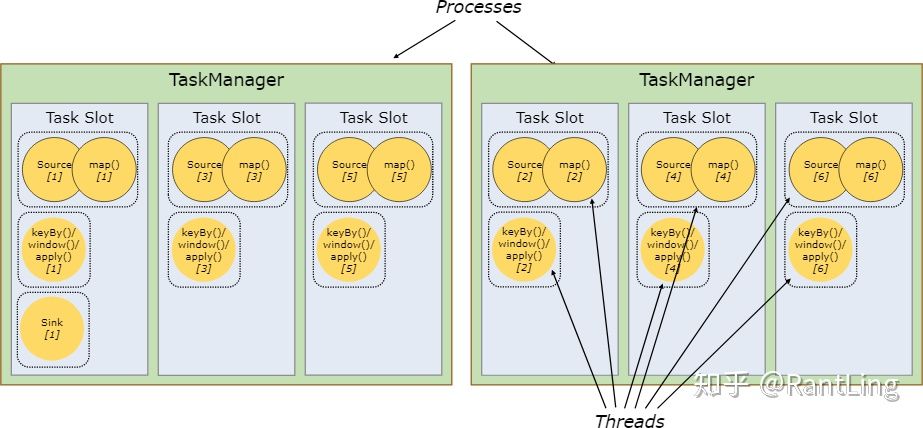
下面是从Flink官网截取的一张架构图：



在Flink运行时涉及到的进程主要有以下两个：  ***JobManager****：主要负责调度task，协调checkpoint已经错误恢复等。当客户端将打包好的任务提交到JobManager之后，JobManager就会根据注册的TaskManager资源信息将任务分配给有资源的TaskManager，然后启动运行任务。TaskManger从JobManager获取task信息，然后使用slot资源运行task；***TaskManager**：执行数据流的task，一个task通过设置并行度，可能会有多个subtask。 每个TaskManager都是作为一个独立的JVM进程运行的。他主要负责在独立的线程执行的operator。其中能执行多少个operator取决于每个taskManager指定的slots数量。Task slot是Flink中最小的资源单位。假如一个taskManager有3个slot，他就会给每个slot分配1/3的内存资源，目前slot不会对cpu进行隔离。同一个taskManager中的slot会共享网络资源和心跳信息。  
当然在Flink中并不是一个slot只可以执行一个task，在某些情况下，一个slot中也可能执行多个task，如下：



一般情况下，flink都是默认允许共用slot的，即便不是相同的task，只要都是来同一个job即可。共享slot的好处有以下两点：  
1. 当Job的最高并行度正好和flink集群的slot数量相等时，则不需要计算总的task数量。例如，最高并行度是6时，则只需要6个slot，各个subtask都可以共享这6个slot； 2. 共享slot可以优化资源管理。如下图，非资源密集型subtask source/map在不共享slot时会占用6个slot，而在共享的情况下，可以保证其他的资源密集型subtask也能使用这6个slot，保证了资源分配。



**3. Flink中的数据**

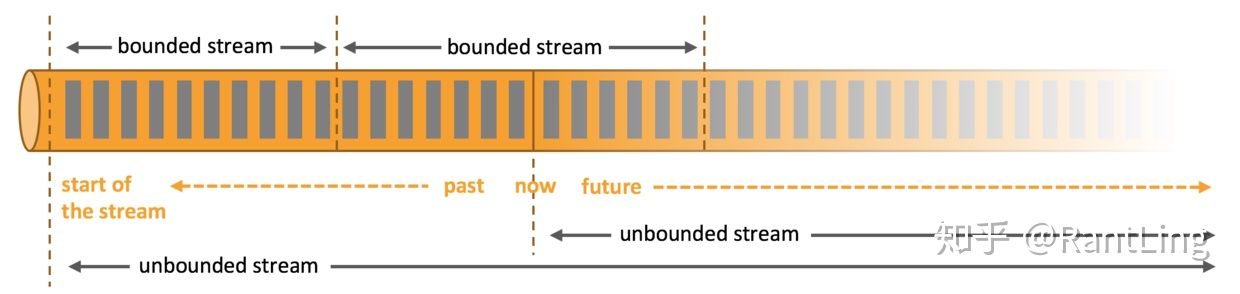
Flink中的数据主要分为两类：**有界数据流(Bounded streams)**和**无界数据流(Unbounded streams)**。

**3.1 无界数据流**

顾名思义，**有界数据流**就是指有始无终的数据，数据一旦开始生成就会持续不断的产生新的数据，即数据没有时间边界。无界数据流需要持续不断地处理。

**3.2 有界数据流**

相对而言，**有界数据流**就是指输入的数据有始有终。例如数据可能是一分钟或者一天的交易数据等等。处理这种有界数据流的方式也被称之为**批处理**：

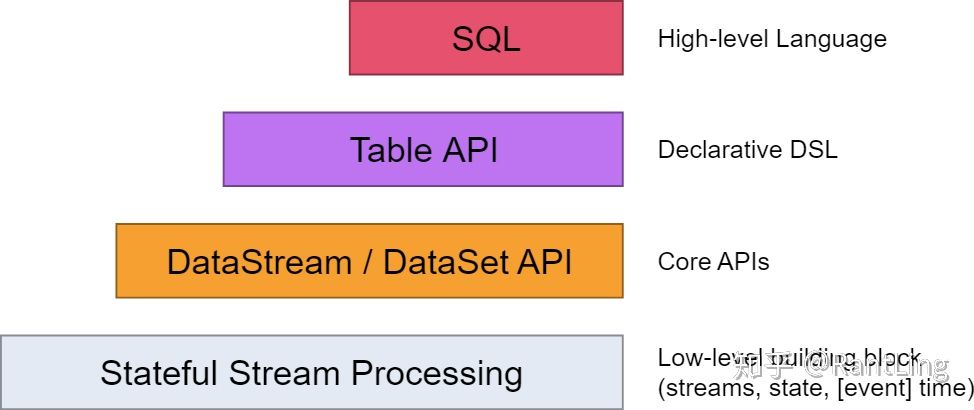


需要注意的是，我们一般所说的**数据流**是指数据集，而**流数据**则是指数据流中的数据。

**4. Flink中的编程模型**

**4.1 编程模型**

在Flink，编程模型的抽象层级主要分为以下4种，越往下抽象度越低，编程越复杂，灵活度越高。



这里先不一一介绍，后续会做详细说明。这4层中，一般用于开发的是第三层，即DataStrem/DataSetAPI。用户可以使用DataStream API处理无界数据流，使用DataSet API处理有界数据流。同时这两个API都提供了各种各样的接口来处理数据。例如常见的map、filter、flatMap等等，而且支持python，scala，java等编程语言，后面的demo主要以scala为主。

**4.2 程序结构**

与其他的分布式处理引擎类似，Flink也遵循着一定的程序架构。下面以常见的WordCount为例：

**val** env **=** **ExecutionEnvironment.**getExecutionEnvironment

*// get input data*

**val** text **=** env**.**readTextFile**(**"/path/to/file"**)**

**val** counts **=** text**.**flatMap **{** **\_.**toLowerCase**.**split**(**"\\W+"**)** filter **{** **\_.**nonEmpty **}** **}**

**.**map **{** **(\_,** 1**)** **}**

**.**groupBy**(**0**)**

**.**sum**(**1**)**

counts**.**writeAsCsv**(**outputPath**,** "\n"**,** " "**)**

下面我们分解一下这个程序。  
第一步，我们需要获取一个ExecutionEnvironment(如果是实时数据流的话我们需要创建一个StreamExecutionEnvironment)。这个对象可以设置执行的一些参数以及添加数据源。所以在程序的main方法中我们都要通过类似下面的语句获取到这个对象：

**val** env **=** **ExecutionEnvironment.**getExecutionEnvironment

第二步，我们需要为这个应用添加数据源。这个程序中是通过读取文本文件的方式获取数据。在实际开发中我们的数据源可能有很多中，例如kafka，ES等等，Flink官方也提供了很多的connector以减少我们的开发时间。一般都是都通addSource方法添加的，这里是从文本读入，所以调用了readTextFile方法。当然我们也可以通过实现接口来自定义source。

**val** text **=** env**.**readTextFile**(**"/path/to/file"**)**

第三步，我们需要定义一系列的operator来对数据进行处理。我们可以调用Flink API中已经提供的算子，也可以通过实现不同的Function来实现自己的算子，这个我们会在后面讨论。这里我们只需要了解一般的程序结构即可。

**val** counts **=** text**.**flatMap **{** **\_.**toLowerCase**.**split**(**"\\W+"**)** filter **{** **\_.**nonEmpty **}** **}**

**.**map **{** **(\_,** 1**)** **}**

**.**groupBy**(**0**)**

**.**sum**(**1**)**

上面的就是先对输入的数据进行分割，然后转换成（word，count）这样的Tuple，接着通过第一个字段进行分组，最后sum第二个字段进行聚合。  
第四步，数据处理完成之后，我们还要为它指定数据的存储。我们可以从外部系统导入数据，亦可以将处理完的数据导入到外部系统，这个过程称为Sink。同Connector类似，Flink官方提供了很多的Sink供用户使用，用户也可以通过实现接口自定义Sink。

counts**.**writeAsCsv**(**outputPath**,** "\n"**,** " "**)**

**小结：**

以上，通过简单的介绍来了解Flink中的一些基本概念及编程方式。后面会对每个细节进行更为详尽地分析。Flink(一)-基本概念