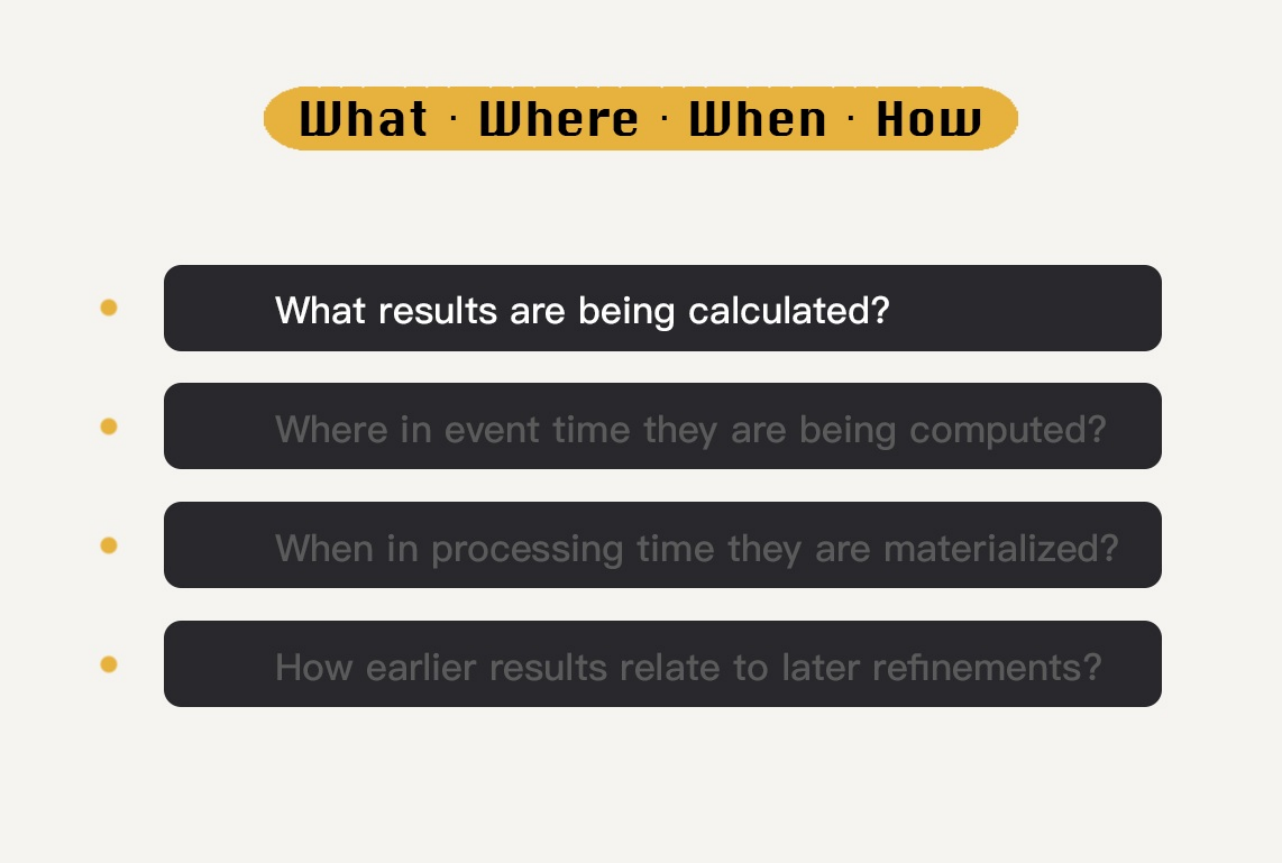
26-Pipeline：Beam如何抽象多步骤的数据流水线？

你好，我是蔡元楠。

今天我要与你分享的主题是“Pipeline：Beam如何抽象多步骤的数据流水线”。

在上两讲中，我们一起学习了Beam是如何抽象封装数据，以及如何抽象对于数据集的转换操作的。在掌握了这两个基本概念后，我们就可以很好地回答Beam编程模型里的4个维度What、Where、When、How中的第一个问题——What了。也就是，我们要做什么计算？想得到什么样的结果？



这个时候你可能已经跃跃欲试，开始想用PCollection和Transform解决我们平常经常会使用到的批处理任务了。没有问题，那我们就先抛开Where、When和How这三个问题，由简至繁地讲起。

现在假设我们的数据处理逻辑只需要处理有边界数据集，在这个情况下，让我们一起来看看Beam是如何运行一套批处理任务的。

数据流水线

在Beam的世界里，所有的数据处理逻辑都会被抽象成**数据流水线（Pipeline）**来运行。那么什么是数据流水线呢？

Beam的数据流水线是对于数据处理逻辑的一个封装，它包括了从**读取数据集**，**将数据集转换成想要的结果**和**输出结果数据集**这样的一整套流程。

所以，如果我们想要跑自己的数据处理逻辑，就必须在程序中创建一个Beam数据流水线出来，比较常见的做法是在main()函数中直接创建。

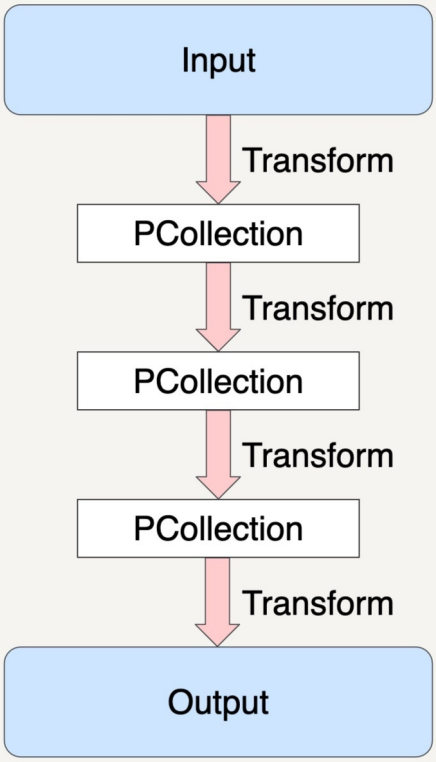
Java

```
PipelineOptions options = PipelineOptionsFactory.create();
Pipeline p = Pipeline.create(options);
```

在创建Beam数据流水线的同时，我们必须给这个流水线定义一个**选项**（Options）。这个选项会告诉Beam，用户的Pipeline应该如何运行。例如，是在本地的内存上运行，还是在Apache Flink上运行？关于具体Beam选项的解释，我会在第30讲中展开讲解。

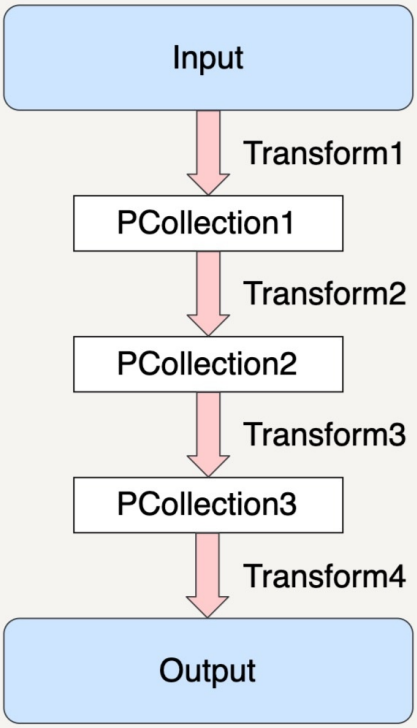
Beam数据流水线的应用

有了数据流水线这个抽象概念之后，我们就可以将PCollection和Transform应用在这个流水线里面了。



上图就是一个Beam的数据流水线，整个数据流水线包括了从读取数据，到经过了N个Transform之后输出数据的整个过程。

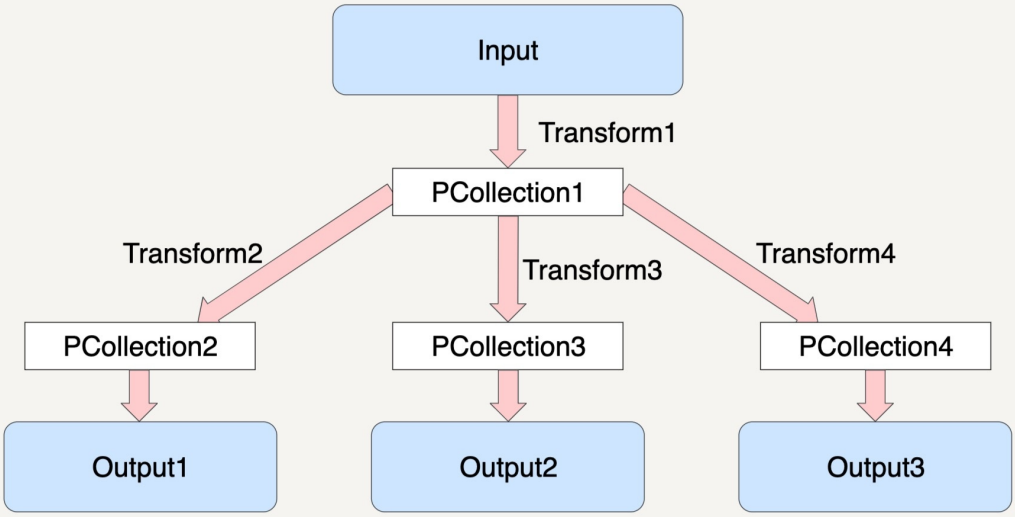
在第24讲中我们学习过PCollection的不可变性。也就是说，一个PCollection一经生成，我们就不能够再增加或者删除它里面的元素了。所以，在Beam的数据流水线中，每次PCollection经过一个Transform之后，流水线都会新创建一个PCollection出来。而这个新的PCollection又将成为下一个Transform的新输入。



在上图的示例中，Beam数据流水线在经过Transform1读取了输入数据集之后，会创建出一个新的PCollection1，而经过了Transform2之后，数据流水线又会创建出新的PCollection2出来，同时PCollection1不会有任何改变。也就是说，在上面的例子中，除去最终的输出结果，数据流水线一共创建了3个不同的PCollection出来。

这种特性可以让我们在编写数据处理逻辑的时候，对同一个PCollection应用多种不同的Transfrom。

例如下图所示，对于PCollection1，我们可以使三个不同的Transform应用在它之上，从而再产生出三个不同的PCollection2、PCollection3和PCollection4出来。



Beam数据流水线的处理模型

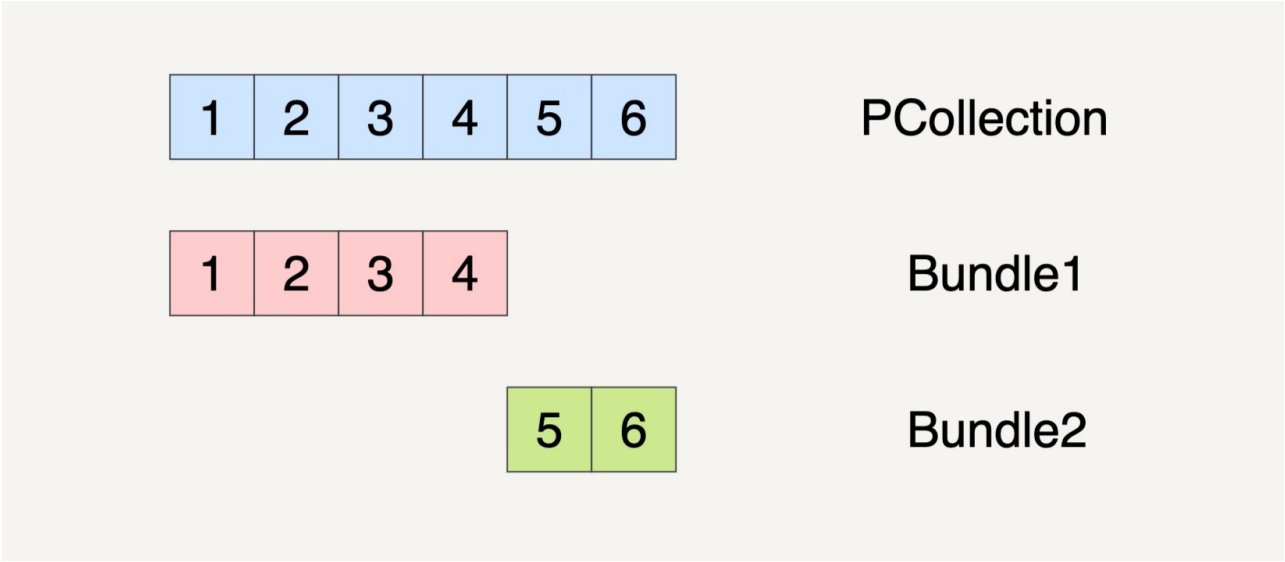
在了解完Beam数据流水线高度抽象的概念后，紧接着，我想和你介绍一下Beam数据流水线的处理模型，也就是数据流水线在运行起来之后，会发生些什么，它是如何处理我们定义好的PCollection和Transform的。

Beam数据流水线的底层思想其实还是动用了MapReduce的原理，在分布式环境下，整个数据流水线会启动N个Workers来同时处理PCollection。而在具体处理某一个特定Transform的时候，数据流水线会将这个Transform的输入数据集PCollection里面的元素分割成不同的Bundle，将这些Bundle分发给不同的Worker来处理。

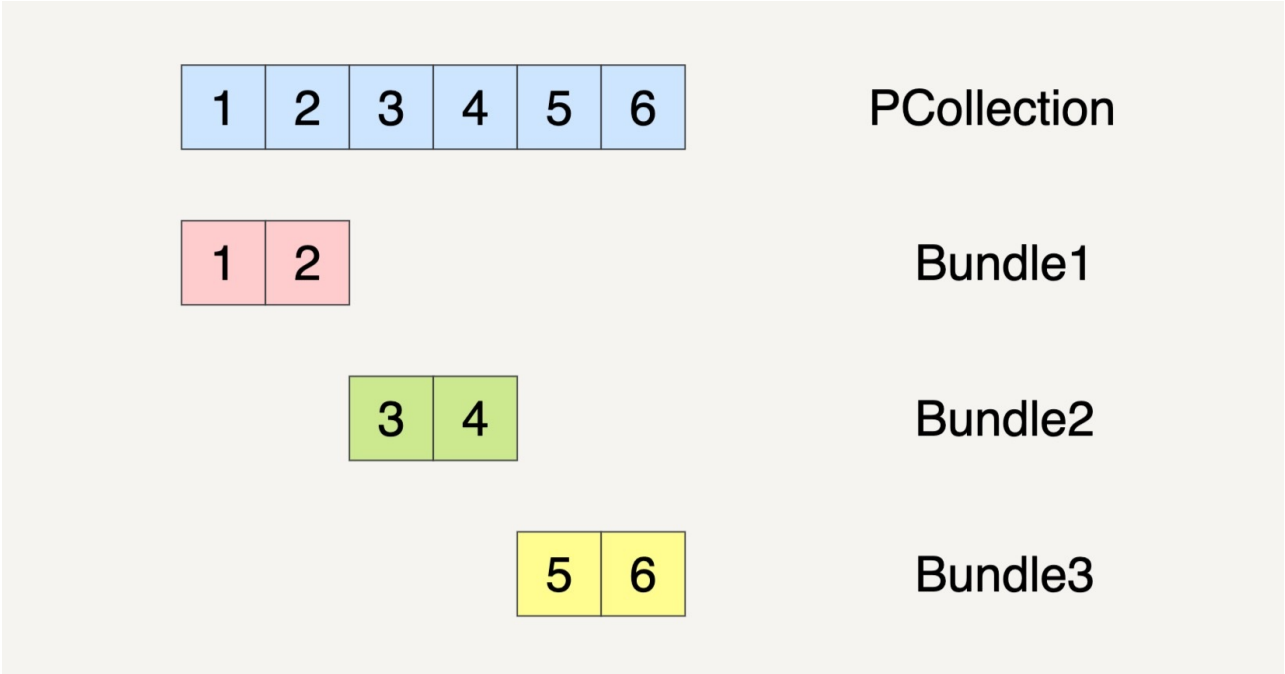
Beam数据流水线具体会分配多少个Worker，以及将一个PCollection分割成多少个Bundle都是随机的。但Beam数据流水线会尽可能地让整个处理流程达到**完美并行**（Embarrassingly Parallel）。

我想举个几个例子让你更好地来理解这个概念。

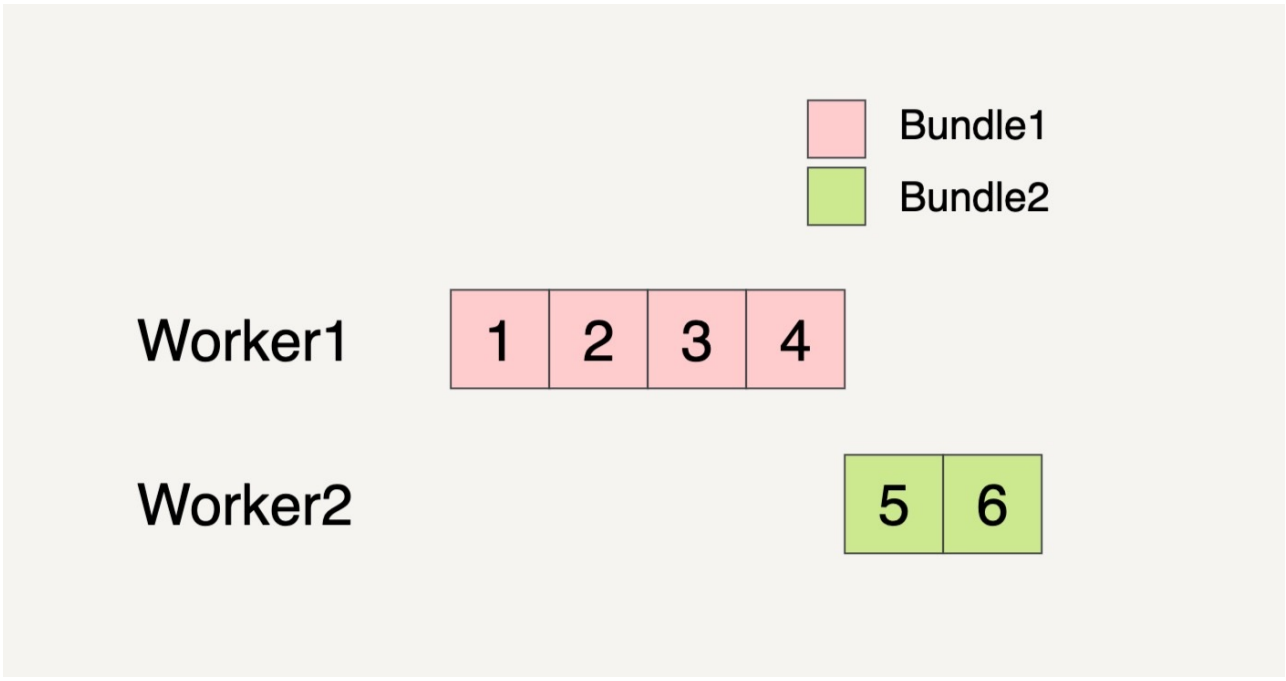
假设在数据流水线的一个Transform里面，它的输入数据集PCollection是1、2、3、4、5、6这个6个元素。数据流水线可能会将这个PCollection按下图的方式将它分割成两个Bundles。



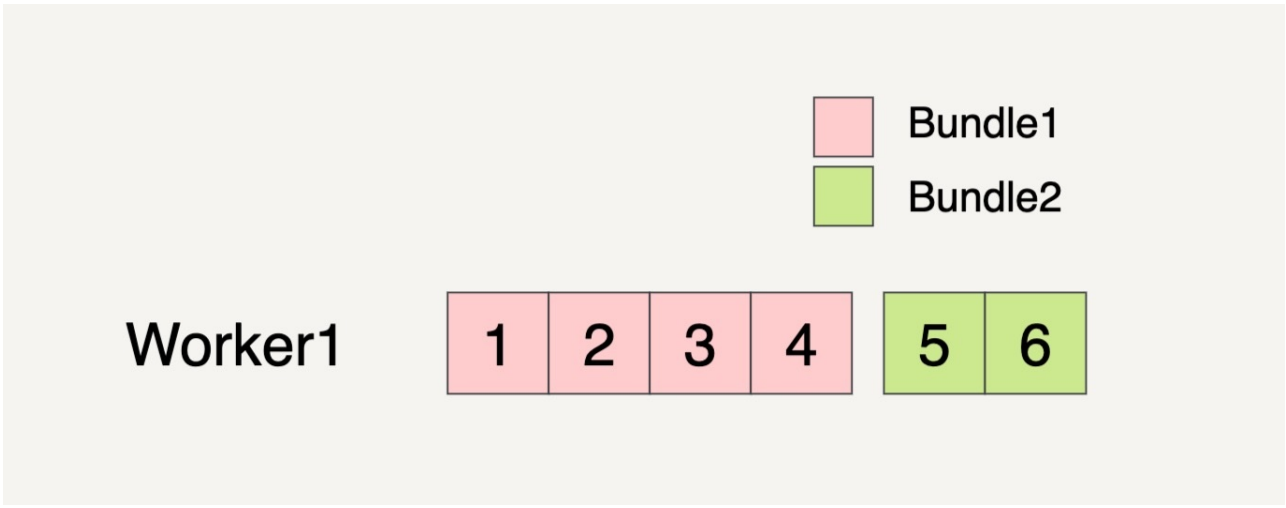
当然，PCollection也有可能会被分割成三个Bundles。



那数据流水线会启用多少个Worker来处理这些Bundle呢？这也是任意的。还是以刚刚的PCollection输入数据集作为例子，如果PCollection被分割成了两个Bundles，数据流水线有可能会分配两个Worker来处理这两个Bundles。



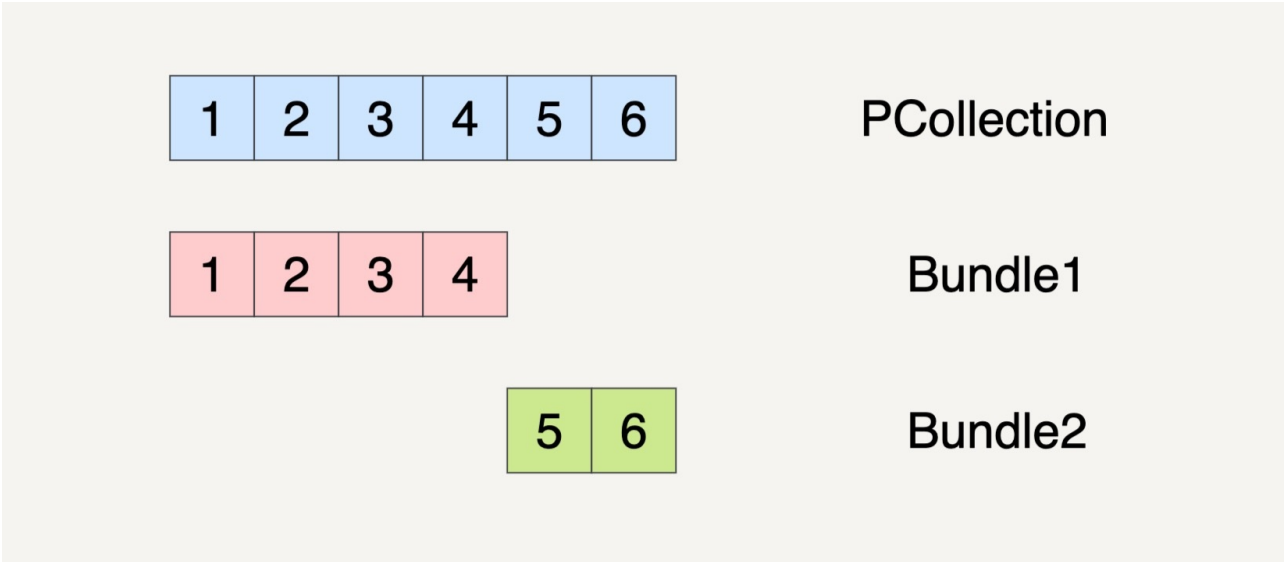
甚至有可能只分配一个Worker来处理这两个Bundles。



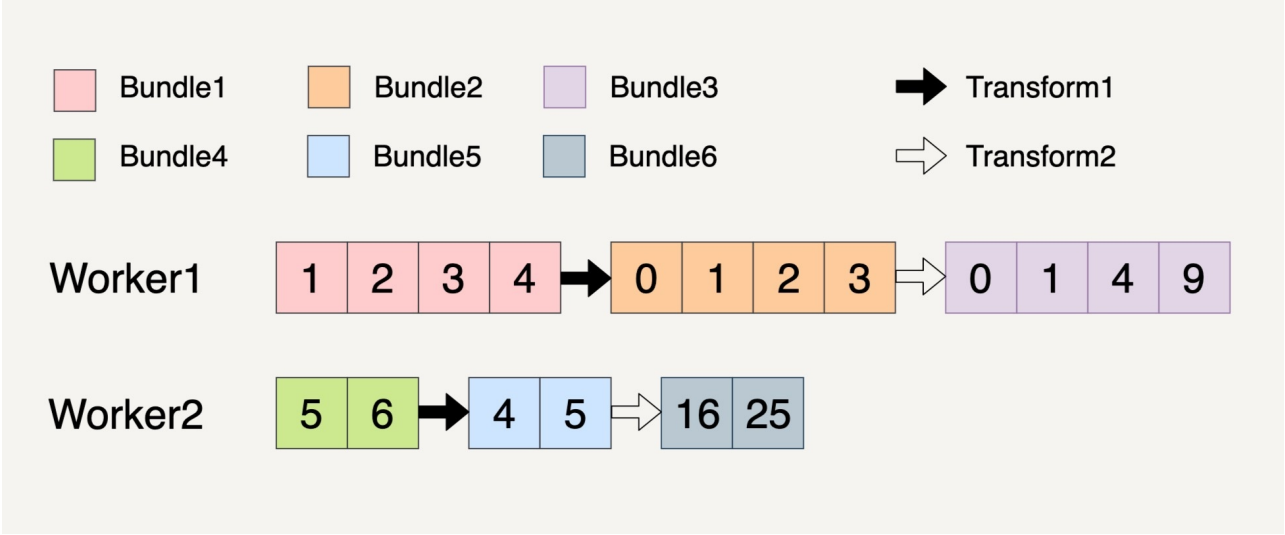
在多步骤的Transforms中，一个Bundle通过一个Transform产生出来的结果会作为下一个Transform的输入。

之前刚刚讲过，在Beam数据流水线中，抽象出来的PCollection经过一个Transform之后，流水线都会新创建一个PCollection出来。同样的，Beam在真正运行的时候，每一个Bundle在一个Worker机器里经过Transform逻辑后，也会产生出来一个新的Bundle，它们也是具有不可变性的。像这种具有关联性的Bundle，必须在同一个Worker上面处理。

我现在来举例说明一下上面的概念。现在假设输入数据集如下图所示，它被分成了两个Bundles。



我们现在需要做两个Transforms。第一个Transform会将元素的数值减一；第二个Transform会对元素的数值求平方。整个过程被分配到了两个Workers上完成。



过程就如上图所示，总共产生了6个不可变的Bundle出来，从Bundle1到Bundle3的整个过程都必须放在Worker1上完成，因为它们都具有关联性。同样的，从Bundle4到Bundle6的整个过程也都必须放在Worker2上完成。

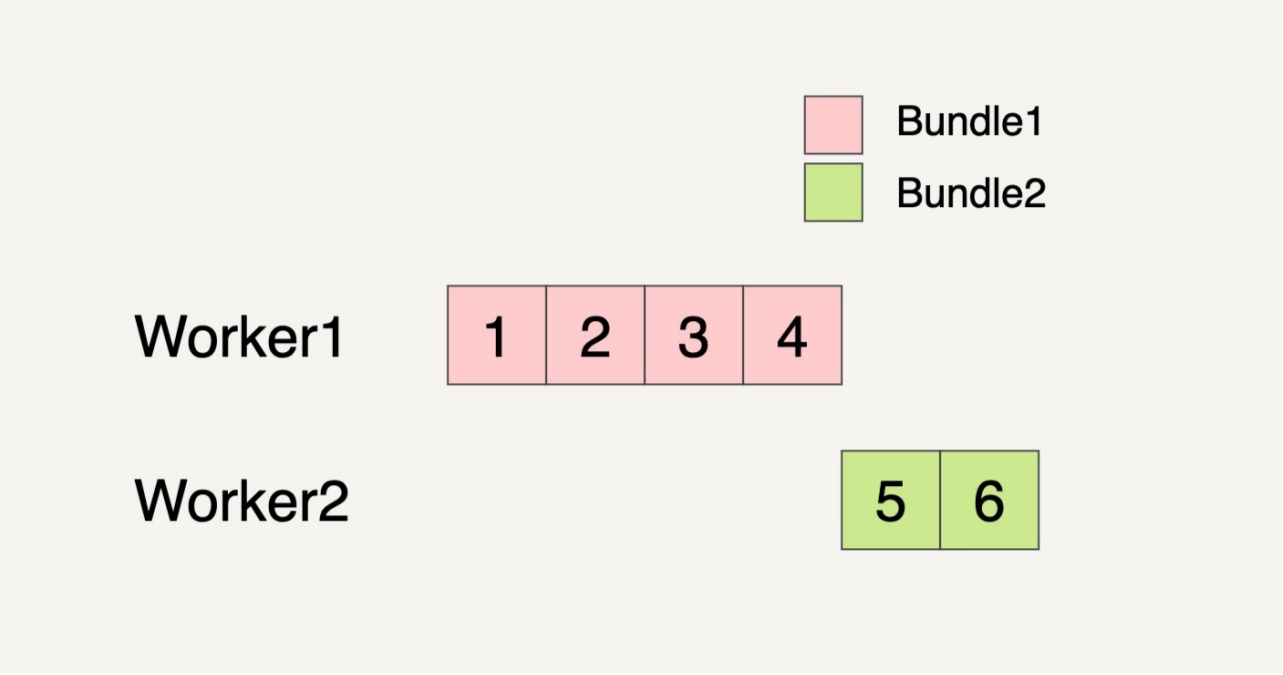
Beam数据流水线的错误处理

在学习完Beam数据流水线底层的处理模型之后，你可能会有个疑问：既然Bundle都是放在分布式环境下处理的，要是其中一个步骤出错了，那数据流水线会做什么样的处理？接下来我会给你讲解一下Beam数据流水线的错误处理机制。

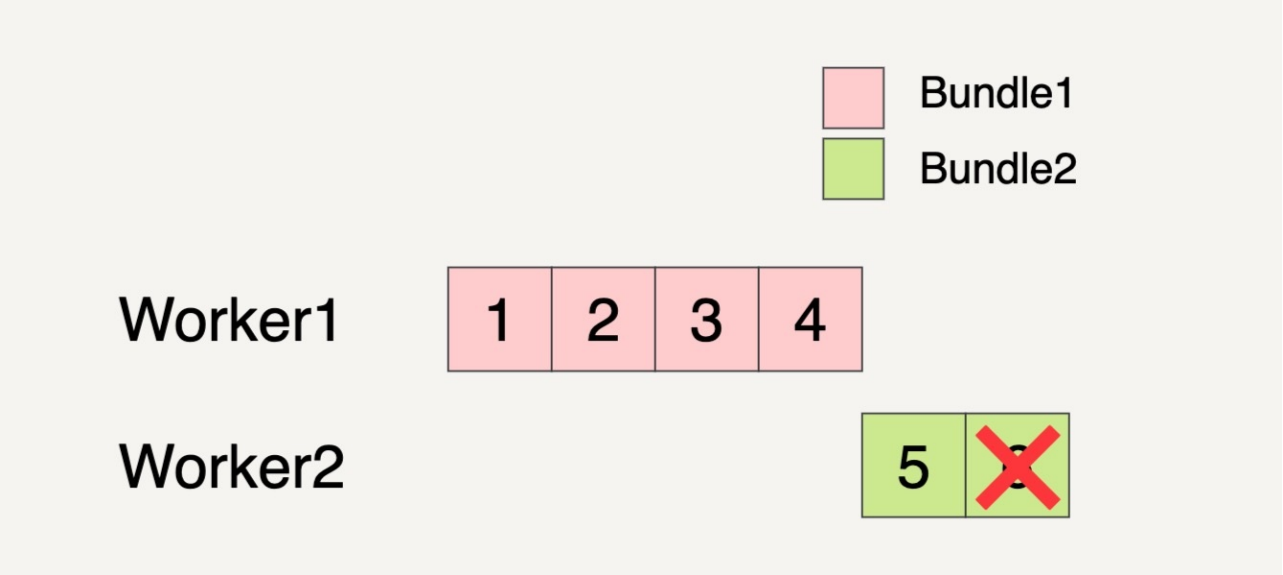
单个Transform上的错误处理

我们还是以单个Transform开始讲解。在一个Transform里面，如果某一个Bundle里面的元素因为任意原因导致处理失败了，则这整个Bundle里的元素都必须重新处理。

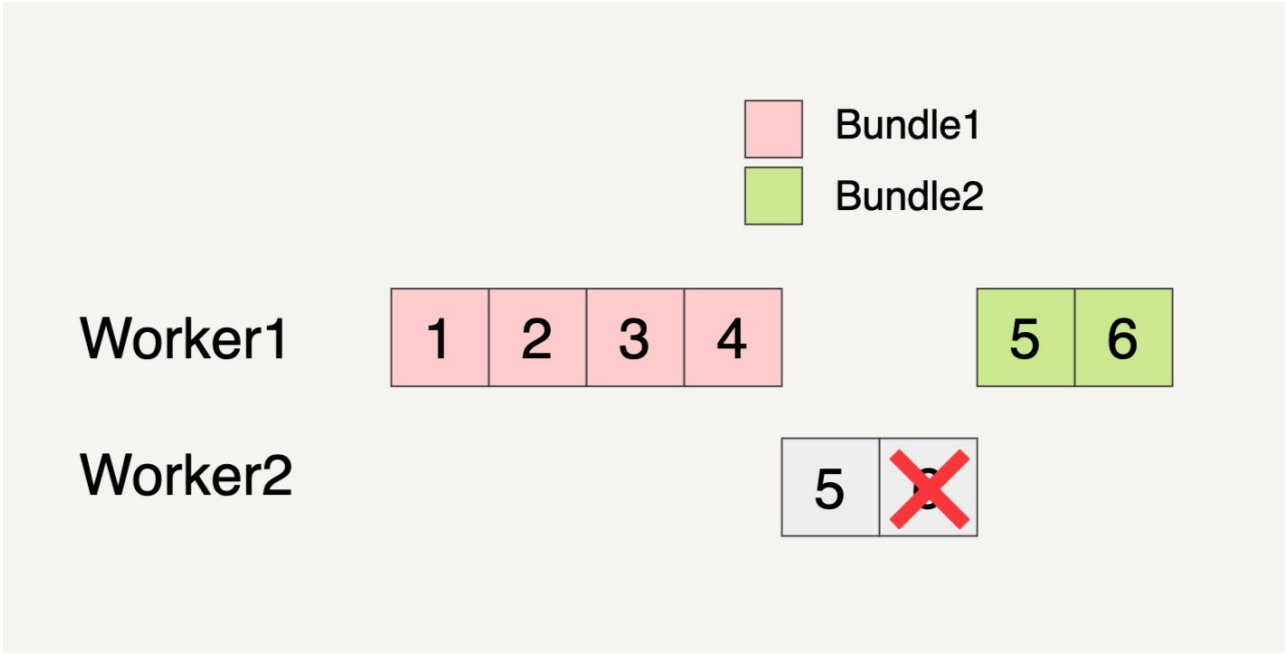
还是假设输入数据集如下图所示，被分成了两个Bundles。



Beam数据流水线分配了两个Worker来处理这两个Bundles。我们看到下图中，在Worker2处理Bundle2的时候，最后一个元素6处理失败了。



这个时候，即便Bundle2的元素5已经完成了处理，但是因为同一个Bundle里面的元素处理失败，所以整个Bundle2都必须拿来重新处理。



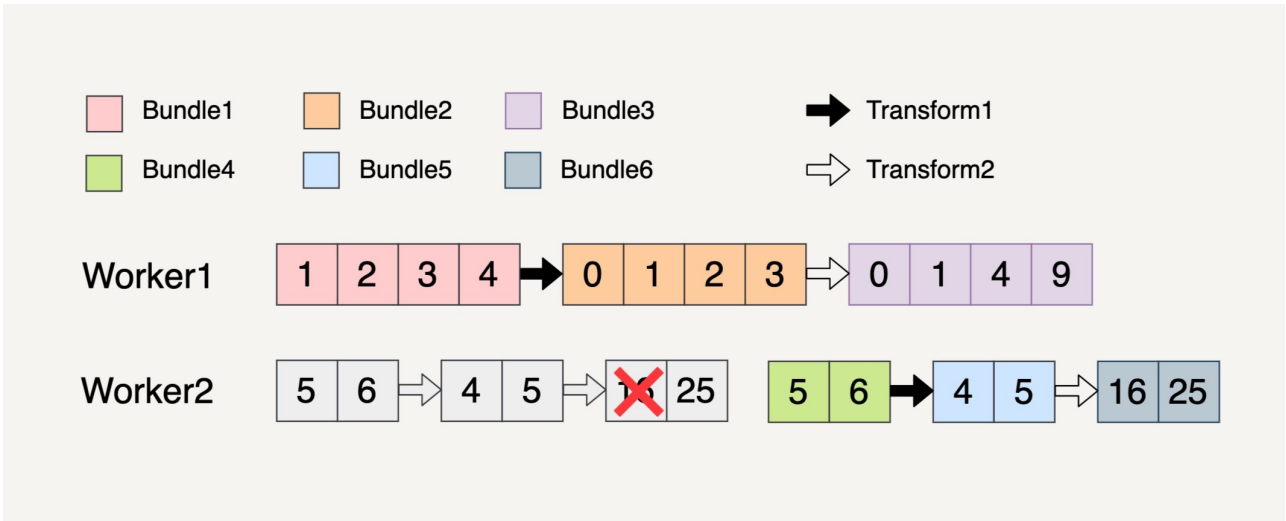
重新处理的Bundle也不一定要在原来的Worker里面被处理，有可能会被转移到另外的Worker里面处理。如上图所示，需要重新被处理的Bundle2就被转移到Worker1上面处理了。

多步骤Transform上的错误处理

学习完单个Transform上的错误处理机制，我们再来看看在多步骤的Transform上发生错误时是如何处理的。

在多步骤的Transform上，如果处理的一个Bundle元素发生错误了，则这个元素所在的整个Bundle以及与这个Bundle有关联的所有Bundle都必须重新处理。

我们还是用上面的多步骤Transform来讲解这个例子。



你可以看到，在Worker2中，处理Transform2逻辑的时候生成Bundle6里面的第一个元素失败了。因为Bundle4、Bundle5和Bundle6都是相关联的，所以这三个Bundle都会被重新处理。

小结

今天我们一起学习了Beam里对于数据处理逻辑的高度抽象数据流水线，以及它的底层处理模型。数据流水线是构建数据处理的基础，掌握了它，我们就可以根据自身的应用需求，构建出一套数据流水线来处理数

据。

思考题

你能根据自己的理解重述一下在Beam的数据流水线中，当处理的元素发生错误时流水线的错误处理机制吗？

欢迎你把答案写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

精选留言：

- cricket1981 2019-06-21 06:17:57
  bundle随机分配会不会产生数据倾斜？完美并行背后的机制是？beam应该也有类似spark的persist方法缓存转换中间结果，防止出错恢复链太长吧？