 什么是结构化的并行数据流：

结构化的并行数据流是一种并行计算模型，用于处理数据流的并行操作。它是一种将数据流分解为多个阶段，并在每个阶段中进行并行处理的方法。

在结构化的并行数据流中，数据流通过一系列的数据流块（Dataflow Blocks）进行处理。每个数据流块都是一个独立的处理单元，负责执行特定的转换或操作。数据流块之间通过连接来形成一个数据处理管道，数据从一个块流向下一个块，经过一系列的转换和操作。

每个数据流块可以并行处理多个数据项，从而实现高效的并行计算。数据流块之间的连接可以通过异步方式进行，使得数据在不同的块之间以异步的方式流动，提高系统的性能和响应能力。

结构化的并行数据流具有以下特点：

1. 并行性：数据流块可以并行处理多个数据项，从而实现高效的并行计算。不同的数据流块可以同时处理不同的数据，提高处理速度和系统的吞吐量。
2. 可组合性：数据流块可以通过连接在一起形成一个完整的数据处理管道。这种可组合性使得代码更加模块化和可维护，易于理解和扩展。
3. 异步处理：数据流块可以在异步模式下执行数据处理操作。这意味着可以在处理数据时不阻塞主线程，提高应用程序的性能和响应能力。
4. 错误处理和恢复：结构化的并行数据流提供了错误处理和恢复的机制。你可以在数据流块中捕获和处理异常，以及在需要时进行错误恢复。这样可以增强系统的健壮性和可靠性。
5. 可扩展性：通过将数据流块连接在一起，可以轻松地扩展数据处理管道。你可以添加更多的数据流块来执行不同的转换和操作，以满足不同的需求。这种可扩展性使得处理大量数据或复杂数据处理逻辑变得更加容易。

Ps：如何将数据流分为多个阶段，有哪些办法：

1. 管道（Pipeline）模式：管道模式是将数据处理过程分解为多个阶段的一种常见方法。每个阶段负责执行特定的数据转换或操作，并将结果传递给下一个阶段。这种方式可以将复杂的数据处理任务分解为多个简单的阶段，使得代码更加模块化和可维护。
2. 数据流块（Dataflow Blocks）：数据流块是.NET中的一种并行计算模型，用于处理数据流。数据流块可以看作是一个独立的处理单元，负责执行特定的转换或操作。通过将多个数据流块连接在一起，可以形成一个完整的数据处理管道，将数据从一个块流向下一个块。
3. 函数式编程：函数式编程是一种将数据处理过程分解为多个函数的编程范式。每个函数负责执行特定的数据转换或操作，并将结果传递给下一个函数。函数式编程强调函数的纯粹性和不可变性，使得数据处理过程更加清晰和可组合。
4. 分布式计算：在分布式计算环境中，可以将数据流分解为多个任务，并在多个计算节点上并行处理。每个计算节点负责执行特定的数据转换或操作，并将结果传递给其他节点。这种方式可以提高数据处理的速度和扩展性。

如何创建一个结构化的并行数据流：要创建一个结构化的并行数据流，可以使用.NET中的TPL数据流（Task Parallel Library Dataflow）。TPL数据流提供了一种简单而强大的方式来处理并行数据流，能够以可组合的方式构建数据处理管道。

实现惰性共享求值状态：

下面是一个线程不安全的类：

图形用户界面, 应用程序

描述已自动生成

如果多个线程同时访问 Value 属性，且 \_value 为 null，它们都会进入 if 语句块中调用 Compute 方法进行计算。这可能导致多个线程同时执行 Compute 方法，可能会导致其他不一致的结果。传统的解决该问题的方法是双重锁定，就像下面这样：

文本, 信件

描述已自动生成

第一次检查该值时如果未被初始化，就计算该值，这样即使有多个线程同时通过了第一个if判定条件，也会被锁住，只有一个进入lock语句块，然后进行计算，剩下的线程在计算完成后被第二个if判断条件判断为不为null，这样就避免了值不一样的问题。

下面这两个例子使用惰性求值，LazyInitializer是个静态类，内部实现了双重锁定，也可以使用Lazy<T>这样就获得了线程安全且惰性求值的对象

文本

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

不关心构造方法也可以避免锁，就不存在上面的不同线程中出现对象构造不同的问题了，这个例子中没有传入外界的\_syncRoot

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：惰性求值是一种计算策略，它在需要时才进行求值。表达式不会立即求值，而是在该值被需要时才进行计算。惰性求值的主要好处是它可以节省计算资源，特别是对于大型或复杂的计算。它允许仅在需要时才计算结果，避免了不必要的计算开销。

图片包含 文本

描述已自动生成

惰性求值的对象可能包含其他属性或方法，它们可能不具备线程安全性。这些属性或方法可能会被多个线程同时访问和修改，从而引发线程安全问题。

使用BlockCollection实现并行管道：

为什么需要管道：

文本

描述已自动生成

下面是书上并行管道的例子：

处理数据的方法：

若BlockCollection中的元素都满足已经完成并且异步操作没有结束就继续循环

文本, 信件

描述已自动生成

下面是对数据进行处理，针对两个不同的构造函数提供了两种处理方式

文本

描述已自动生成

监听如果按下C键位取消过程

文本, 信件

描述已自动生成

阶段一：

文本

描述已自动生成



阶段二：

文本

低可信度描述已自动生成

阶段三

图形用户界面, 文本, 应用程序

描述已自动生成

接着的代码的解释：

文本

描述已自动生成

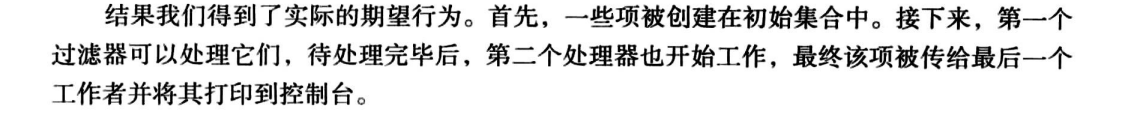
定义管道元素：

表格

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成



微软提供了一个TPL数据流库（在System.Tasks.DataFlow命名空间），可以直接使用：

文本

描述已自动生成

这样就可以使用TPL数据流创建了一个并行管道

文本

描述已自动生成

Ps：

文本, 信件

描述已自动生成

通过这个可以指定一个块中元素的最大容量，创建了一个 BufferBlock对象，并通过 DataflowBlockOptions 的 BoundedCapacity 属性将容量设置为5。

什么是Map/Reduce模式，有什么特点：

文本

描述已自动生成

使用这个例子理解;

文本

描述已自动生成

Map阶段，我们将数据集分割成多个小的数据块，并由多个并行的Map任务进行处理。每个Map任务将输入数据块中的整数映射为键值对，其中键为固定值（例如"average"），值为该整数。

Reduce阶段，中间结果根据键进行分组，并由多个并行的Reduce任务进行处理。每个Reduce任务将具有相同键的整数进行聚合和计算，最终生成平均值的中间结果。

使用PLINQ可以更方便地实现Map/Reduce模式，比如下面这个借助于Select和Average，还可以是比如ToChar等操作

文本

描述已自动生成