概念

本页主要列出了 Storm 的主要概念和资源链接，在链接里你可以发现更多的信息。概念如下：

1. Topologies（拓扑）
2. Streams（流）
3. Spouts（数据源）
4. Bolts（数据流处理组件）
5. Stream groupings（数据流分组）
6. Reliability（可靠性）
7. Tasks（任务）
8. Workers（工作进程）

Topologies（拓扑）

Storm 的拓扑是对实时计算应用逻辑的封装，它的作用与 MapReduce 的任务（Job）很相似，区别在于 MapReduce 的一个 Job 在得到结果之后总会结束，而拓扑会一直在集群中运行，直到你手动去终止它。拓扑还可以理解成由一系列通过数据流（Stream Grouping）相互关联的 Spout 和 Bolt 组成的的拓扑结构。Spout 和 Bolt 称为拓扑的组件（Component）。我们会在后文中给出这些概念的解释。

相关资料：

- TopologyBuilder：在 Java 中使用此类构造拓扑
- 在生产环境中运行拓扑
- 本地模式：通过本文学习如何在本地模式中开发、测试拓扑

Streams（数据流）

数据流（Streams）是 Storm 中最核心的抽象概念。一个数据流指的是在分布式环境中并行创建、处理的一组元组（tuple）的无界序列。数据流可以由一种能够表述数据流中元组的域（fields）的模式来定义。在默认情况下，元组（tuple）包含有整型（Integer）数字、长整型（Long）数字、短整型（Short）数字、字节（Byte）、双精度浮点数（Double）、单精度浮点数（Float）、布尔值以及字节数组等基本类型对象。当然，你也可以通过定义可序列化的对象来实现自定义的元组类型。

在声明数据流的时候需要给数据流定义一个有效的 id。不过，由于在实际应用中使用最多的还是单一数据流的 Spout 与 Bolt，这种场景下不需要使用 id 来区分数据流，因此可以直接使用 OutputFieldsDeclarer来定义“无 id”的数据流。实际上，系统默认会给这种数据流定义一个名为“default”的 id。

相关资料：

- 元组（Tuple）：数据流由多个元组构成
- OutputFieldsDeclarer：用于声明数据流和数据流对应的模式
- 序列化（Serialization）：关于 Storm 元组的动态类型以及声明自定义序列化模型的相关内容

Spouts（数据源）

数据源（Spout）是拓扑中数据流的来源。一般 Spout 会从一个外部的数据源读取元组然后将他们发送到拓扑中。根据需求的不同，Spout 既可以定义为可靠的数据源，也可以定义为不可靠的数据源。一个可靠的 Spout 能够在它发送的元组处理失败时重新发送该元组，以确保所有的元组都能得到正确的处理；相对应的，不可靠的 Spout 就不会在元组发送之后对元组进行任何其他的处理。

一个 Spout 可以发送多个数据流。为了实现这个功能，可以先通过 OutputFieldsDeclarer 的 declareStream 方法来声明定义不同的数据流，然后在发送数据时在 SpoutOutputCollector 的 emit 方法中将数据流 id 作为参数来实现数据发送的功能。

Spout 中的关键方法是 nextTuple。顾名思义，nextTuple 要么会向拓扑中发送一个新的元组，要么会在没有可发送的元组时直接返回。需要特别注意的是，由于 Storm 是在同一个线程中调用 Spout 的所有方法，nextTuple 不能被 Spout 的任何其他功能方法所阻塞，否则会直接导致数据流的中断。

Spout 中另外两个关键方法是 ack 和 fail，他们分别用于在 Storm 检测到一个发送过的元组已经被成功处理或处理失败后的进一步处理。注意，ack 和 fail 方法仅仅对上述“可靠的” Spout 有效。

相关资料：

- IRichSpout：这是实现 Spout 的接口
- 消息的可靠性处理

Bolts（数据流处理组件）

拓扑中所有的数据处理均是由 Bolt
完成的。通过数据过滤（filtering）、函数处理（functions）、聚合（aggregations）、联结（joins）、数据库交互等功能，Bolt
几乎能够完成任何一种数据处理需求。

一个 Bolt 可以实现简单的数据流转换，而更复杂的数据流变换通常需要使用多个 Bolt
并通过多个步骤完成。例如，将一个微博数据流转换成一个趋势图像的数据流至少包含两个步骤：其中一个 Bolt
用于对每个图片的微博转发进行滚动计数，另一个或多个 Bolt 将数据流输出为“转发最多的图片”结果（相对于使用2个Bolt，如果使用3个 Bolt
你可以让这种转换具有更好的可扩展性）。

与 Spout 相同，Bolt 也可以输出多个数据流。为了实现这个功能，可以先通过 OutputFieldsDeclarer 的 declareStream
方法来声明定义不同的数据流，然后在发送数据时在 OutputCollector 的 emit 方法中将数据流 id 作为参数来实现数据发送的功能。

在定义 Bolt 的输入数据流时，你需要从其他的 Storm 组件中订阅指定的数据流。如果你需要从其他所有的组件中订阅数据流，你就必须要在定义
Bolt 时分别注册每一个组件。对于声明为默认 id
的数据流，InputDeclarer支持订阅此类数据流的语法糖。也就是说，如果需要订阅来自组件“1”的数据流，declarer.shuffleGrouping("1") 与
declarer.shuffleGrouping("1", DEFAULT_STREAM_ID) 两种声明方式是等价的。

Bolt 的关键方法是 execute 方法。execute 方法负责接收一个元组作为输入，并且使用 OutputCollector
对象发送新的元组。如果有消息可靠性保障的需求，Bolt 必须为它所处理的每个元组调用 OutputCollector 的 ack 方法，以便 Storm
能够了解元组是否处理完成（并且最终决定是否可以响应最初的 Spout
输出元组树）。一般情况下，对于每个输入元组，在处理之后可以根据需要选择不发送还是发送多个新元组，然后再响应（ack）输入元组。IBasic
Bolt 接口能够实现元组的自动应答。

在 bolts 中开启新线程做异步处理是非常好的，OutputCollector 是线程安全的，可以在任何时候被调用。

相关资料：

- IRichBolt：用于定义 Bolt 的基本接口
- IBasicBolt: 用于定义带有过滤或者其他简单的函数操作功能的 Bolt 的简便接口
- OutputCollector：Bolt 使用此类来发送数据流
- 消息的可靠性处理

Stream groupings（数据流分组）

1. 为拓扑中的每个 Bolt 的确定输入数据流是定义一个拓扑的重要环节。数据流分组定义了在 Bolt
   的不同任务（tasks）中划分数据流的方式。
2. 在 Storm 中有八种内置的数据流分组方式，而且你还可以通过CustomStreamGrouping
   接口实现自定义的数据流分组模型。这八种分组分时分别为：
3. Shuffle grouping（随机分组）：这种方式下元组会被尽可能随机地分配到 Bolt
   的不同任务（tasks）中，使得每个任务所处理元组数量能够能够保持基本一致，以确保集群的负载均衡。
4. Fields
   grouping（域分组）：这种方式下数据流根据定义的“域”来进行分组。例如，如果某个数据流是基于一个名为“user-id”的域进行分组
   的，那么所有包含相同的“user-id”的元组都会被分配到同一个任务中，这样就可以确保消息处理的一致性。
5. Partial Key
   grouping（部分关键字分组）：这种方式与域分组很相似，根据定义的域来对数据流进行分组，不同的是，当数据倾斜的时候，会在下游
   的两个 Bolt
   间进行负载均衡，它能更好的利用资源。感兴趣的读者可以参考这篇论文，其中详细解释了这种分组方式的工作原理以及它的优点。
6. All grouping（完全分组）：这种方式下数据流会被同时发送到 Bolt
   的所有任务中（也就是说同一个元组会被复制多份然后被所有的任务处理），使用这种分组方式要特别小心。
7. Global grouping（全局分组）：这种方式下所有的数据流都会被发送到 Bolt 的同一个任务中，也就是 id 最小的那个任务。
8. None grouping（非分组）：使用这种方式说明你不关心数据流如何分组。目前这种方式的结果与随机分组完全等效，不过未来 Storm
   社区可能会考虑通过非分组方式来让 Bolt 和它所订阅的 Spout 或 Bolt 在同一个线程中执行。
9. Direct
   grouping（直接分组）：这是一种特殊的分组方式。使用这种方式意味着元组的发送者可以指定下游的哪个任务可以接收这个元组。只有
   在数据流被声明为直接数据流时才能够使用直接分组方式。使用直接数据流发送元组需要使用 OutputCollector 的其中一个 emitDirect
   方法。Bolt 可以通过 TopologyContext 来获取它的下游消费者的任务 id，也可以通过跟踪 OutputCollector 的 emit
   方法（该方法会返回它所发送元组的目标任务的 id）的数据来获取任务 id。
10. Local or shuffle grouping（本地或随机分组）：如果在源组件的 worker 进程里目标 Bolt
    有一个或更多的任务线程，元组会被随机分配到那些同进程的任务中。换句话说，这与随机分组的方式具有相似的效果。

相关资料：

- TopologyBuilder：使用此类构造拓扑
- InputDeclarer：在 TopologyBuilder 中调用 setBolt 方法时会返回这个对象的实例，通过该对象就可以定义 Bolt
  的输入数据流以及数据流的分组方式

Reliability（可靠性）

Storm 可以通过拓扑来确保每个发送的元组都能得到正确处理。通过跟踪由 Spout
发出的每个元组构成的元组树可以确定元组是否已经完成处理。每个拓扑都有一个“消息延时”参数，如果 Storm
在延时时间内没有检测到元组是否处理完成，就会将该元组标记为处理失败，并会在稍后重新发送该元组。

为了充分利用 Storm 的可靠性机制，你必须在元组树创建新结点的时候以及元组处理完成的时候通知 Storm。这个过程可以在 Bolt
发送元组时通过 OutputCollector 实现：在 emit 方法中实现元组的锚定（Anchoring），同时使用 ack 方法表明你已经完成了元组的处理。

关于可靠性保障的更多内容可以参考这篇文章：消息的可靠性处理。

Tasks（任务）

在 Storm 集群中每个 Spout 和 Bolt
都由若干个任务（tasks）来执行。每个任务都与一个执行线程相对应。数据流分组可以决定如何由一组任务向另一组任务发送元组。你可以在
TopologyBuilder 的 setSpout 方法和 setBolt 方法中设置 Spout/Bolt 的并行度。

Workers（工作进程）

拓扑是在一个或多个工作进程（worker processes）中运行的。每个工作进程都是一个实际的 JVM
进程，并且执行拓扑的一个子集。例如，如果拓扑的并行度定义为300，工作进程数定义为50，那么每个工作进程就会执行6个任务（进程内部的线程）。Storm 会在所有的 worker 中分散任务，以便实现集群的负载均衡。

相关资料：

- Config.TOPOLOGY_WORKERS：这个配置项用于设置拓扑的工作进程数