事务Topology的实现概述如下。 □ 事务类型的Spout节点实际上是一个子Topology,它包含一个协调Spout节点（Coordinator ) 以及一些消息发送Bolt节点（Emitter )。 □ 协调Spout节点的并行度为1, 消息发送Bolt节点的并行度则可根据需要来设定。 □ 协调Spout节点并不发送实际的数据，而是将事务尝试发送到消息将Bolt节点中。事务尝 试（ TransactionalAttempt ) 包含一个Biglnteger类型的事务号以及长整型类型的尝试号。 当事务被重传时，事务号是相同的，但是尝试号不同。 □ 协调Spout节点与消息发送Bolt节点之间采用全局分组方式进行消息传输，也就意味着从 协调Spout中发送的一条事务尝试消息都会被所有的消息发送Bolt节点接收。每个消息发 送Bolt节点会根据收到的事务尝试消息来发送与该事务对应的消息集合。消息发送Bolt节 点之间则需要对该事务所对应的数据进行协作，即每个节点只负责事务的一部分数据。 □ 当所有的消息发送节点都成功处理了该事务在该节点上所对应的消息后（通过Storm的 Ack框架），协调Spout节点认为该事务已经被成功处理，协调Spout节点将会产生并发送下 一个事务尝试消息。 □ 协调Spout节点中含有两个系统输出流：事务消息流（Batch流）和事务提交流（Commit流） 协调Spout节点会向事务消息流发送事务尝试消息，向事务提交流发送事务的提交消息。 □ 事务的处理实际上被分成以下两个阶段。 ■ 事务处理阶段：协调Spout节点向消息发送节点发送事务尝试消息，消息发送Bolt节点

244 第 15 章 事 务 Topology的 实 现

发送该事务所对应的消息集合，然后Storm系统开始处理这些消息。该阶段属于事务的 处理阶段。 ■ 事务提交阶段：通过系统的Ack框架，当系统中的消息均被成功处理后，协调Spout节 点将收到其发送的事务尝试消息的Ack, 这表明事务处理阶段已经结束。 在事务类型的Topology中，可以定义事务提交Bolt ( CommitBolt ), 这种类型的Bolt节点可保 证其处理的事务会按照顺序被提交。事务的提交操作可能是将事务的处理结果存储起来。 在Storm中，若消息丢失或者超时，一个事务可能会被重做，此时将导致消息重复。但由于提 交节点可以保证事务是按照顺序提交的，此处更利于去重操作。例如，若每个事务对应的数据在重 传时相同，则在提交节点看到相同的事务序号时，可认为该事务是重传的事务，进而将其忽略掉。 当事务的处理阶段完成后，协调Spout节点会检查该事务是否为下一个要提交的事务，若是 则将该事务的事务序号发送到事务提交流。所有的事务提交Bolt节点都会接收该流的消息，并会 在收到事务提交消息后，对该事务进行提交。可以看出，当事务处理阶段结束后，并不一定会立 即进人事务提交阶段，它需要等待之前的事务都已经被成功提交后，方可进人事务提交阶段。此 处，保证了事务按顺序提交。 通过系统的Ack框架，在收到事务提交消息的Ack之后，该事务被认为已成功处理。 在事务提交节点，将调用节点的finishBatch方法完成事务的提交。其他的处理节点（BatchBolt ) 同样含有finishBatch回调方法，但它却表示在该节点上所有属于同一事务的消息都已经被处理。 本 章将在CoordinatedBolt类的实现中详细讨论Storm是如何保证finishBatch方法被正确调用的。 在事务处理阶段，属于一个事务的消息以及所有衍生出来的消息均以协调Spout节点发送的 事务尝试消息为根。基于Storm的Ack框架，当所有的消息均被成功处理之后，协调Spout节点将 收到一条事务尝试消息的Ack。若消息处理失败或者超时，协调Spout节点则会收到失败消息，事 务类型的Topology此时会对该事务进行重传处理。这样，事务类型的Topology便可以保证消息不 丢失。实际上，事务尝试消息会存储于ZooKeeper中，所以即便Topology异常停止运行，仍可保 证在其重新启动时事务能被正确重传。 根据其中Spout类型的不同，事务Topology可被进一步分为基本事务Topology和模糊事务 Topology ( Opaque TransactionalTopology

ZOOKEEPER\_ROOT的值为/transactional;id为Spout节点名字，该名字是在仓1j建Topology时指 定的；subroot为user或者coordinator: 例如，若Spout的名字为TestSpout，则兀数据的路径 为/transactional/TestSpout/coordinator。每次提交同样的Topology时，Storm会产生不同的 Topologyld以作区分，但Spout节点名字是固定的。于是重新提交的Topology会继续访问上 一个Topology在ZooKeeper中存储的元数据，也就实现了从上一个Topology结束点继续执行 的目标。这也是事务Topology可以实现可靠消息传输的原因之一

CoordinatedBolt是非常关键的一个类，它用于协调系统中的Bolt节点。对于事务Tbpology在 —个Bolt节点中，若能获知属于某一个事务的消息是否已经全部收到，这将是非常有帮助的。 CoordinatedBolt会记录自身发送至目标节点的消息数目，当属于一个事务的消息发送结束 后，它通过直接分组的方式向协调流发送一条消息，通知目标节点需要从该CoordinatedBolt接收 多少条消息。 事务Topology中Bolt节点都利用CoordinatedBolt进行包装，它期待从上游节点收到这些协调 消息以判断属于一个事务的消息是否收全，并在收全一个事务的消息后，向它的下游节点发出其 需要接收的消息数目的通知。 CoordinatedBolt节点收到并处理完属于一个事务的全部消息后，将调用finishBatch方法c finishBatch方法是事务Topology中非常重要的方法，在事务提交节点中，它用于存储事务处理 的结果。当它被调用时，就表示属于一个事务的消息都已经被成功处理了

storm.trident.planner.SpoutNode@45592af7[

spout=com.gantian.tridentTopology.MySpout@36359723

txId=txId

type=BATCH

nodeId=55d4bdbe-0082-4c6e-9e60-700ba878276e

name=<null>

allOutputFields=[tx, val]

streamId=s1

parallelismHint=<null>

stateInfo=<null>

creationIndex=1

]

storm.trident.planner.ProcessorNode@53eba4b8[

committer=false

processor=storm.trident.planner.processor.EachProcessor@4b88ca8e

selfOutFields=[jiou]

nodeId=229598cf-4b70-4016-8f26-c52e49a5425d

name=<null>

allOutputFields=[tx, val, jiou]

streamId=s2

parallelismHint=<null>

stateInfo=<null>

creationIndex=2

]storm.trident.planner.ProcessorNode@53eba4b8[

committer=false

processor=storm.trident.planner.processor.EachProcessor@4b88ca8e

selfOutFields=[jiou]

nodeId=229598cf-4b70-4016-8f26-c52e49a5425d

name=<null>

allOutputFields=[tx, val, jiou]

streamId=s2

parallelismHint=<null>

stateInfo=<null>

creationIndex=2

]

Function:

用户自定义的Function

Function:

CombinerAggregatorInitImpl

storm.trident.planner.ProcessorNode@7c46c9c3[

committer=false

processor=storm.trident.planner.processor.EachProcessor@7197b07f

selfOutFields=[count]

nodeId=0838631b-a18a-43f6-9f2e-e697c09f708d

name=<null>

allOutputFields=[tx, val, jiou, count]

streamId=s3

parallelismHint=<null>

stateInfo=<null>

creationIndex=3

]

essor=storm.trident.planner.processor.EachProcessor@4b88ca8e

selfOutFields=[jiou]

nodeId=229598cf-4b70-4016-8f26-c52e49a5425d

name=<null>

allOutputFields=[tx, val, jiou]

streamId=s2

parallelismHint=<null>

stateInfo=<null>

creationIndex=2

]

storm.trident.planner.ProcessorNode@30a7653e[

committer=false

processor=storm.trident.planner.processor.AggregateProcessor@7cff3f1d

selfOutFields=[jiou, count]

nodeId=a64aa99a-03e0-4f44-9366-5311499d88c8

name=<null>

allOutputFields=[jiou, count]

streamId=s4

parallelismHint=<null>

stateInfo=<null>

creationIndex=4

]

Aggregator:

ChainedAggregatorImpl

里面有个Aggregator[] \_aggs

存了CombinerAggregatorCombineImpl

而CombinerAggregatorCombineImpl里面有个Aggregator是用户自定义的

通过层层包装

storm.trident.planner.PartitionNode@33a8c9c9[

nodeId=e5e23293-843f-4cbb-ab96-da05a0b37b00

name=<null>

allOutputFields=[jiou, count]

streamId=s4

parallelismHint=<null>

stateInfo=<null>

creationIndex=5

],

storm.trident.planner.ProcessorNode@382dc417[

committer=false

processor=storm.trident.planner.processor.AggregateProcessor@3daf03d8

selfOutFields=[jiou, count]

nodeId=351d4ad5-857a-4386-93fe-f1b2f5db12db

name=<null>

allOutputFields=[jiou, count]

streamId=s5

parallelismHint=<null>

stateInfo=<null>

creationIndex=6

]

Aggregator:

GroupedAggregator

包装ChainedAggregatorImpl

里面数组装的CombinerAggregatorCombineImpl

storm.trident.planner.ProcessorNode@2415e4c7[

committer=true

processor=storm.trident.planner.processor.PartitionPersistProcessor@72ce812e

selfOutFields=[jiou, count]

nodeId=6f8b98aa-f113-49d4-9f6a-eb1f095ba648

name=<null>

allOutputFields=[jiou, count]

streamId=s6

parallelismHint=<null>

stateInfo=storm.trident.planner.NodeStateInfo@521441d5

creationIndex=7

]

StateUpdater:

MapCombinerAggStateUpdater

里面有个CombinerAggregator来进行combine操作得到

每个finishBatch的每个分组上会调用该combine操作将当前值累加到数据库的值上