TURING 图灵原创

从源代码的角度深入剖析Storm设计与实现

微软搜索技术部门高级研发工程师实战经验

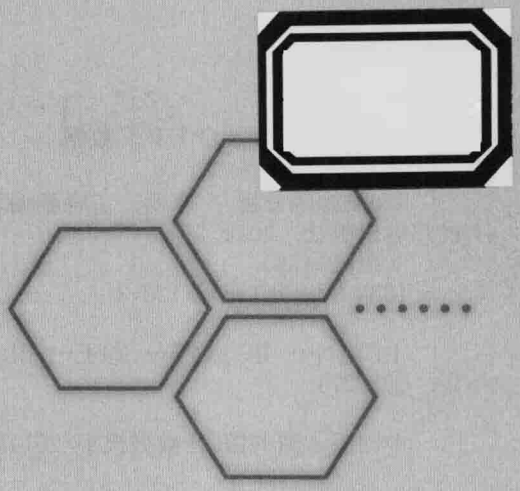
学习大牛如何实现和高效利用“实时的Hadoop”

Storm 源码分析

李明 王晓鹏 ◎ 编著

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵原创



Storm 源码分析

李明 王晓鹏 ◎ 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

Storm源码分析 / 李明，王晓鹏编著. -- 北京 : 人民邮电出版社，2014.11
（图灵原创）
ISBN 978-7-115-37126-3

Ⅰ. ①S… Ⅱ. ①李… ②王… Ⅲ. ①数据处理软件 Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2014)第221005号

内 容 提 要

本书从源代码的角度详细分析了Storm的设计与实现，共分为三个部分，第一部分介绍了Storm的基本原理以及Storm集群系统的搭建方法，第二部分深入剖析了Storm的底层架构，如Nimbus、Supervisor、Worker以及Task，第三部分系统讨论了Storm如何实现可靠的消息传输，如Transaction Topology以及Trident。

本书适用于程序员、架构师以及计算机专业的学生。

◆ 编　　著　李　明　王晓鹏
责任编辑　王军花
执行编辑　李鸿鹏
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：30.25
字数：715千字　　2014年11月第1版
印数：1-3 000册　　2014年11月北京第1次印刷

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

专家推荐

流计算是目前计算机领域非常热门的技术，Storm 平台的出现大大推进了该项技术的发展，并被很多包括微软在内的大公司采用。《Storm 源代码分析》从源代码角度深入浅出地分析了 Storm 的设计及实现，一方面可以使读者更好地了解并用好 Storm 技术，另一方面可以让读者学习如何设计大规模分布式系统，相信读者一定会受益匪浅。

——于伟，微软资深开发总监

在当今互联网众多领域中，大数据和云计算无疑是两个最火的主题，而当中尤其以大数据的实时流处理为很多开发者都感兴趣的。作者在书中对 Storm 进行了详尽的介绍，按部就班，化繁为简，让读者能一步一景地学懂 Storm 的箇中细节，实在是 Storm 入门者的必备良药。

在我十多载微软职业生涯当中（美国总部和中国），遇到的技术大牛多如繁星，但李明和晓鹏给我的印象尤为深刻。记得当初我们组建广告 BI 团队的时候就先立下了采用开源技术这个指导思想，对传统的微软人来说，这是开创先河之举。李明和晓鹏是最早加入的，从第一天开始，我就感觉到他们对技术的热情与执着。在短短的一个月里，他们不仅理解了 Storm 的精髓和关键，还实现了一个 BI pipeline 的雏形，让我们能展现出实时流处理大数据的力量。在往后的日子里，每天的工作量都非常庞大，但更加令我惊讶的是，他们居然写了一本关于 Storm 的书。要知道学懂一门技术对于开发者来说不难，但要著书立说却要经过一定程度的沉淀和思考，并非旦夕之功。我拜读了他们的初稿后，发觉书中对 Storm 的理解精辟透彻，对 Storm 的运用和各处细节也都阐述入微。尤其是对 Storm 的入门初学者来说，是一本不可多得的好书。在后续的日子里，当我们构建大数据平台时，他们已经转岗到别的团队，但我们在遇到技术难点的时候，还会借助他们对 Storm 的深刻了解来解决问题。这是一本应该放在桌面、随时可以翻阅的参考书。赞！

——章英基，前微软资深开发总监，现阿里巴巴资深总监

大数据处理是当前计算机科技的热点，而流式实时大数据处理更是这皇冠上璀璨的明珠。实时流数据处理在搜索引擎、社交网络、电商网站、广告平台等领域有着相当广泛的应用。Storm 是极其高效、灵活、高扩展的流式数据处理平台，它被 Twitter、Taobao、Yahoo！和 Groupon 等公司采用。

本书由微软公司互联网工程院经验丰富的一线程序员操刀编写，包含很多实战经验和使用心得，很好地结合了代码分析和应用实例。本书对于进行流式数据处理的研究、Storm 的深入理解

以及实际应用都有很好的参考价值。我有幸与本书作者李明、王晓鹏共同开发基于 Storm 的应用，他们深厚的程序功力、不懈的钻研精神令我深深地叹服，这在本书中也得到了很好的体现。我相信读者一定会受益匪浅。

——王明雨，微软资深开发工程师

本书是国内为数不多的讲述大数据流计算并进行源码分析的一本实战书。它出自技术精湛并且具有丰富实战经验的的微软工程师之手，我相信绝不会让你失望！作为解决大数据 5 个“V”之一的“Velocity”问题，Storm 是最流行的实时计算框架，它被认为是 Hadoop 批处理计算的补充，它比 Map/Reduce 更加灵活，而且性能、可靠性和可扩展性出众，所以在 Twitter 等互联网公司被广泛应用到大数据实时和准实时处理的生产环境。

在微软公司担任数据平台产品经理期间，我有幸和李明、王晓鹏等同事合作，一起将 Storm 应用于微软搜索中心的广告、监控、安全和预测等应用场景。在工作期间，这本书对我帮助很大，即便对于像我这样在分布式领域工作 12 年的老手来讲，这本书仍然让我受益良多。无论你是大数据领域、分布式系统的从业人员，还是开源系统的爱好者、开发者或互联网从业人员，我认为这本书都值得仔细研读。如果你想了解流计算的设计原理，想洞悉 Storm 的设计精髓，或揣摩用 Clojure，抑或是探求如何用 Storm 来解决大数据的准实时需求场景，这本书都会对你大有裨益。

——贺军，微软资深项目经理

前　言

Storm是一个分布式的、可靠的实时计算系统。与Hadoop的批处理不同，Storm采用流式的消息处理方法，它使得消息可以得到快速的处理，可以用于实时性要求较高的系统，例如广告点击的在线统计等。Storm弥补了Hadoop在实时处理方面的缺陷，目前被各大互联网公司广泛使用并日益流行。

本书作为第一本深入介绍Storm的图书，从源代码的角度详细剖析了Storm的设计与实现。本书适合各类型的计算机工作者，初学者可以通过本书来学习如何实现一个可靠的、高容错性的、实时的分布式处理平台。而对于Storm用户来讲，本书不仅可以帮助他们更深入地了解这套系统的工作原理，还可以帮助他们正确地使用该平台，也有利于实现对Storm的二次开发。鉴于Storm是基于Clojure和Java开发的，所以需要读者对这两种语言有一定的了解。

本书主要分析阐述了Storm的底层架构，例如Nimbus、Supervisor、Worker、Executor以及Task，并对Storm如何实现可靠的消息传输进行了系统讨论，例如事务Topology以及Trident。

本书对Storm的最新源代码进行了系统而详尽的分析，相信读者在阅读过程中一定会获益匪浅。

致谢

诚挚感谢人民邮电出版社和图灵公司为我提供创作的平台。感谢本书的编辑王军花、张霞等，你们的专业态度和细致工作提升了本书的品质。

诚挚感谢妻子包黎云，没有她的支持，我无法完成本书。最后，将该书献给我即将出世的孩子。

——李明

诚挚感谢同事贺军、王明雨以及童杰，感谢你们在这本书的编写过程中给予我们的无私帮助。特别感谢我的妻子白鸽，感谢你对我的理解、包容和支持！

——晓鹏

目　录

第 1 章　总体架构与代码结构 …… 1

1.1 Storm 的总体结构 …… 1

1.2 Storm 的元数据 …… 3

1.2.1 元数据介绍 …… 3

1.2.2 Storm 怎么使用这些元数据 …… 4

1.3 Storm 的代码结构 …… 7

1.3.1 Clojure 代码 …… 7

1.3.2 Java 代码 …… 8

1.3.3 Trident 代码 …… 9

1.3.4 其他代码 …… 10

第 2 章　搭建 Storm 集群 …… 11

2.1 搭建单机 Storm 集群 …… 11

2.2 搭建多机 Storm 集群 …… 14

2.2.1 设置环境 …… 14

2.2.2 启动 Storm 集群 …… 15

2.2.3 提交 Topology …… 15

2.3 WordCountTopology 介绍 …… 15

2.3.1 RandomSentenceSpout …… 15

2.3.2 SplitSentence …… 16

2.3.3 WordCount …… 17

2.3.4 WordCountTopology 构建 …… 17

第 3 章　Storm 编程基础 …… 19

3.1 Fields 定义 …… 19

3.2 Tuple 接口 …… 20

3.3 常用声明接口 …… 21

3.3.1 配置声明接口 …… 22

3.3.2 输入声明接口 …… 23

3.3.3 输出字段声明接口 …… 24

3.3.4 组件声明接口 …… 25

3.4 Spout 输出收集器 …… 25

3.4.1 ISpoutOutputCollector 和 SpoutOutputCollector …… 25

3.4.2 Executor 中 ISpoutOutputCollector 的实现 … 27

3.5 Bolt 输出收集器 …… 28

3.5.1 IOutputCollector 和 OutputCollector …… 28

3.5.2 IBasicOutputCollector 和 BasicOutputCollector …… 31

3.5.3 BatchOutputCollector 和 BatchOutputCollectorImpl …… 32

3.5.4 Executor 中的 IOutputCollector 实现 …… 34

3.6 组件接口 …… 35

3.7 Spout 接口 …… 35

3.7.1 ISpout …… 36

3.7.2 IRichSpout …… 38

3.8 Bolt 接口 …… 38

3.8.1 IBolt …… 38

3.8.2 IRichBolt …… 40

3.8.3 IBasicBolt …… 40

3.8.4 IBatchBolt …… 42

3.8.5 小结 …… 45

3.9 Storm 数据结构 …… 46

3.9.1 GlobalStreamId …… 46

3.9.2 消息分组方式 …… 46

3.9.3 StreamInfo …… 47

3.9.4 ShellComponent …… 47

3.9.5 ComponentObject …… 47

3.9.6 ComponentCommon …… 47

3.9.7 SpoutSpec ····48
3.9.8 Bolt ····48
3.9.9 StormTopology ····49
3.9.10 TopologySummary ····49
3.9.11 SupervisorSummary ····49
3.9.12 ClusterSummary ····50
3.9.13 BoltStats ····50
3.9.14 SpoutStats ····50
3.9.15 统计信息 ····50
3.9.16 DRPC ····51
3.10 基本 Topology 构建器 ····52
3.10.1 TopologyBuilder ····52
3.10.2 ConfigGetter ····55
3.10.3 SpoutGetter 和 BoltGetter ····55
3.10.4 一个简单例子 ····56
3.11 异常处理 ····57
第 4 章 基础函数和工具类 ····58
4.1 计时器 ····58
4.1.1 mk-timer ····58
4.1.2 check-active! ····60
4.1.3 schedule ····60
4.1.4 schedule-recurring ····60
4.1.5 cancel-timer ····61
4.2 async-loop ····61
4.3 event-manager ····62
4.4 even-sampler ····63
4.5 ZooKeeper 工具类 ····64
4.5.1 mk-client ····64
4.5.2 create-node ····65
4.5.3 get-data ····65
4.5.4 进程内启动 ZooKeeper ····66
4.6 LocalState ····66
4.7 ClusterState ····68
4.8 StormClusterState ····69
第 5 章 通信机制 ····71
5.1 进程间通信 ····71
5.1.1 进程间通信协议 ····71
5.1.2 LocalCluster 模式实现 ····72
5.1.3 分布式模式实现 ····73
5.1.4 协议使用 ····75
5.2 进程内通信 ····77
5.2.1 Disruptor Queue 的使用 ····77
5.2.2 DisruptorQueue 的 Clojure 处理器 ····80
第 6 章 Nimbus ····81
6.1 Nimbus 服务接口定义 ····81
6.2 Nimbus 相关的数据结构 ····83
6.2.1 Java 数据结构 ····83
6.2.2 Clojure 数据结构 ····84
6.3 Nimbus 中的线程介绍 ····86
6.3.1 mk-assignments ····87
6.3.2 do-cleanup ····89
6.3.3 clean-inbox ····90
6.4 Topology 状态转移 ····90
6.4.1 transition-name! ····90
6.4.2 transition! ····91
6.4.3 state-transitions ····92
6.5 启动 Nimbus 服务 ····96
6.5.1 launch-server! ····96
6.5.2 service-handler ····97
6.6 关闭 Nimbus 服务 ····99
6.7 主要服务方法 ····99
6.7.1 submitTopology ····99
6.7.2 kill、rebalance、activate、deactivate 方法 ····101
6.7.3 文件上传与下载 ····102
6.7.4 获取 UI 所需的信息 ····104
6.7.5 获取 Topology ····106
6.7.6 获取 Storm 配置项 ····107
6.8 主要辅助方法 ····107
6.8.1 system-topology! ····107
6.8.2 normalize-topology ····112
6.8.3 compute-new-topology->executor->node+port ····114
6.8.4 compute-executors ····117
第 7 章 Scheduler ····119
7.1 IScheduler 接口 ····119
7.2 EvenScheduler ····120

7.2.1 schedule-topolpgies-evenly ····· 120
7.2.2 schedule-topology ····· 121
7.2.3 get-alive-assigned-node+port->executors ····· 122
7.2.4 sort-slots ····· 123
7.3 DefaultScheduler ····· 124
7.3.1 default-schedule ····· 124
7.3.2 slots-can-reassign ····· 126
7.3.3 bad-slots ····· 126
7.4 IsolationScheduler ····· 127
7.5 调度示例 ····· 131
7.5.1 EvenScheduler 和 DefaultScheduler ····· 131
7.5.2 IsolationScheduler ····· 134

第 8 章 Scheduler ····· 137

8.1 与 Supervisor 相关的数据结构 ····· 137
8.1.1 standalone-supervisor ····· 137
8.1.2 Supervisor 的数据 ····· 138
8.1.3 本地存储数据 ····· 139
8.2 Supervisor 中的线程 ····· 140
8.2.1 计时器线程 ····· 140
8.2.2 同步 Nimbus 任务的线程 ····· 140
8.2.3 管理 Worker 进程的线程 ····· 143
8.3 启动 Supervisor ····· 145
8.4 关闭 Supervisor ····· 147
8.5 重要方法介绍 ····· 147
8.5.1 launch-worker ····· 147
8.5.2 read-allocated-workers ····· 150
8.5.3 wait-for-worker-launch ····· 151
8.5.4 shutdown-worker ····· 152
8.5.5 download-storm-code ····· 152

第 9 章 Worker ····· 155

9.1 Worker 中的数据 ····· 155
9.2 Worker 中的计时器 ····· 157
9.2.1 Worker 的心跳 ····· 157
9.2.2 Executor 的心跳 ····· 158
9.2.3 Worker 中对 ZMQ 连接的维护 ····· 159
9.2.4 从 ZooKeeper 获取 Topology 的活跃情况 ····· 161
9.2.5 小结 ····· 162
9.3 创建 Worker ····· 163
9.4 关闭 Worker ····· 164
9.5 重要辅助方法介绍 ····· 165
9.5.1 Worker 中的接收函数 ····· 166
9.5.2 Worker 中的发送函数 ····· 167
9.5.3 获取属于 Worker 的 Executor ····· 169
9.5.4 创建 Executor 的接收消息队列和查找表 ····· 169
9.5.5 下载 Topology 的配置项以及代码 ····· 170
9.6 小结 ····· 171

第 10 章 Executor ····· 172

10.1 Executor 的数据 ····· 172
10.2 Executor 的输入和输出 ····· 174
10.2.1 Executor 的输入及处理 ····· 174
10.2.2 Executor 的输出及发送 ····· 175
10.3 Spout 类型的 Executor ····· 176
10.3.1 准备消息循环的数据 ····· 176
10.3.2 Spout 输入处理函数 ····· 178
10.3.3 Spout 消息发送函数 ····· 180
10.3.4 Spout 对象的初始化 ····· 181
10.3.5 消息循环 ····· 182
10.4 Bolt 类型的 Executor ····· 184
10.4.1 准备消息循环的数据 ····· 184
10.4.2 Bolt 输入处理函数 ····· 184
10.4.3 Bolt 的消息发送函数 ····· 185
10.4.4 Bolt 对象的初始化 ····· 185
10.4.5 消息循环 ····· 186
10.5 创建 Executor ····· 187
10.6 辅助函数介绍 ····· 188
10.6.1 组件的 Grouper 函数 ····· 188
10.6.2 带流量控制的错误报告方法 ····· 193
10.6.3 触发系统 Ticks ····· 194
10.7 小结 ····· 196

第 11 章 Task ……198
11.1 Task 的上下文对象 ……198
11.1.1 TopologyContext ……198
11.1.2 GeneralTopologyContext ……199
11.1.3 WorkerTopologyContext ……200
11.1.4 TopologyContext ……201
11.2 创建 Task 数据 ……202
11.3 mk-tasks-fn 函数 ……204
11.4 send-unanchored ……205
11.5 创建 Task ……206
11.6 Storm 中传输的消息以及序列化 ……206
第 12 章 Storm 的 Ack 框架 ……208
12.1 Acker Bolt 的实现分析 ……209
12.2 启动消息跟踪 ……211
12.3 消息跟踪 ……212
12.4 Ack 机制的例子 ……214
第 13 章 系统运行统计 ……216
13.1 基础数据结构以及更新算法 ……216
13.1.1 滑动窗口的数据结构 ……216
13.1.2 滑动窗口的回调函数 ……220
13.1.3 滑动窗口集合的类型 ……221
13.2 Storm 中的统计信息 ……222
13.2.1 Stats 中定义的统计类别 ……222
13.2.2 运行统计的更新 ……223
13.2.3 运行统计的更新时间点 ……223
13.2.4 获取统计数据 ……228
13.3 运行统计的 Thrift 结构 ……229
第 14 章 系统运行统计的另一种实现 ……231
14.1 内置统计信息的计算 ……231
14.1.1 MultiCountMetric ……232
14.1.2 MultiReducedMetric ……233
14.2 内置统计类型 ……234
14.2.1 Spout 类型的内置统计 ……235
14.2.2 Bolt 类型的内置统计 ……235
14.3 统计触发消息 ……235
14.3.1 注册统计信息 ……236
14.3.2 触发消息的产生与发送 ……237
14.3.3 处理统计触发消息 ……238
14.4 运行统计收集节点 ……239
14.5 SystemBolt ……241
第 15 章 事务 Topology 的实现 ……243
15.1 事务 Topology 的实现概述 ……243
15.1.1 事务 Topology 的类型 ……244
15.1.2 事务 Topology 的类关系 ……245
15.2 ITransactionalSpout 接口 ……246
15.3 协调 Spout 节点的执行器 ……248
15.3.1 ZooKeeper 客户端工具 ……248
15.3.2 协调 Spout 的执行器 ……255
15.3.3 消息发送 Bolt 的执行器 ……261
15.4 CoordinatedBolt 的实现分析 ……264
15.4.1 TrackingInfo ……264
15.4.2 CoordinatedOutput-Collector ……265
15.4.3 CoordinatedBolt 中的消息类型 ……267
15.4.4 成员变量以及主要方法分析 ……267
15.5 分区的事务类型 ……271
15.5.1 分区的事务 Spout 接口 ……271
15.5.2 分区的事务 Spout 的执行器 ……273
15.6 分区的模糊事务 Spout ……277
15.6.1 分区的模糊事务 Spout 的接口 ……277
15.6.2 模糊的事务 Spout 执行器 ……278
15.7 事务 Topology 的构建器 ……281
15.7.1 构建器的构造函数及成员变量 ……281
15.7.2 设置 Bolt 对象 ……283
15.7.3 构建 Topology ……284
15.7.4 输入流声明器 ……286
第 16 章 事务 Topology 示例 ……288
16.1 例子代码 ……288
16.1.1 分区的事务 Spout ……288
16.1.2 局部计数 Bolt 的实现 ……291

16.1.3 全局计数 Bolt 的实现 ····· 292
16.2 构建 Topology ····· 293
16.3 事务处理示例 ····· 295
第 17 章 Trident 的 Spout 节点 ····· 298
17.1 ITridentSpout 接口 ····· 298
17.1.1 BatchCoordinator 接口 ····· 299
17.1.2 TridentSpoutCoordinator ····· 300
17.1.3 MasterBatchCoordinator ····· 301
17.1.4 消息发送节点接口 ····· 306
17.1.5 消息发送接口的执行器 ····· 306
17.2 适配 IRichSpout 接口 ····· 307
17.3 适配 IBatchSpout 接口 ····· 311
17.4 Trident 中分区的 Spout 类型 ····· 311
17.4.1 分区 Spout 接口 ····· 311
17.4.2 分区 Spout 的执行器 ····· 313
17.5 模糊事务类型的 Spout 节点 ····· 316
17.5.1 模糊事务类型的 Spout 接口 ····· 317
17.5.2 模糊事务类型 Spout 的执行器 ····· 317
17.6 构建 Spout 节点 ····· 320
17.6.1 TridentTopology 的 newStream 调用 ····· 320
17.6.2 TridentTopology 中 newDRPCStream 调用 ····· 321
第 18 章 Trident 的存储 ····· 322
18.1 存储的基本接口 ····· 322
18.2 MapState 接口的实现 ····· 323
18.2.1 非事务类型的存储 ····· 324
18.2.2 事务类型的存储 ····· 325
18.2.3 模糊事务类型存储 ····· 327
18.3 值的序列化方法 ····· 329
18.4 数据更新接口 ····· 330
18.4.1 CombinerValueUpdater ····· 330
18.4.2 ReducerValueUpdater ····· 331
18.5 存储更新接口 ····· 331
18.5.1 ReducerAggStateUpdater ····· 332
18.5.2 MapReducerAggStateUpdater ····· 332
18.5.3 BaseStateUpdater ····· 334
18.6 创建存储对象 ····· 334
第 19 章 Trident 消息 ····· 336
19.1 ValuePointer ····· 336
19.2 Factory 接口及其实现 ····· 337
19.2.1 ProjectionFactory ····· 338
19.2.2 FreshOutputFactory ····· 339
19.2.3 OperationOutputFactory ····· 339
19.2.4 RootFactory ····· 341
19.3 消息工厂的例子 ····· 342
19.4 TridentTupleView ····· 342
19.5 ComboList ····· 343
第 20 章 Trident 操作与处理节点 ····· 346
20.1 操作的基本接口 ····· 346
20.2 Aggregator 实现 ····· 347
20.2.1 GroupedAggregator ····· 348
20.2.2 ChainedAggregatorImpl ····· 350
20.2.3 SingleEmitAggregator ····· 353
20.3 用户接口及其实现 ····· 355
20.3.1 ReducerAggregator 接口及其实现 ····· 355
20.3.2 CombinerAggregator 接口及其实现 ····· 356
20.4 所有处理节点的上下文 ····· 357
20.4.1 单个处理节点的上下文 ····· 358
20.4.2 操作执行的上下文 ····· 359
20.5 Trident 的输出收集器 ····· 359
20.5.1 FreshCollector ····· 359
20.5.2 CaptureCollector ····· 360
20.5.3 GroupCollector ····· 360
20.5.4 AppendCollector ····· 361
20.5.5 AddIdCollector ····· 361
20.6 Trident 的处理节点 ····· 362
20.6.1 TridentProcessor 接口 ····· 363
20.6.2 PartitionPersistProcessor ····· 363
20.6.3 StateQueryProcessor ····· 365
20.7 聚集器的执行 ····· 367

第 21 章　Trident 流的基本操作……370
21.1　流的成员变量和基础方法……370
21.1.1　流的成员变量……370
21.1.2　流节点名字……370
21.1.3　流的映射检查……372
21.1.4　添加节点……372
21.2　流映射操作……373
21.3　流的分组操作……374
21.4　流的逐行操作……374
21.5　流的分区操作……374
21.6　流的单聚集器聚集操作……376
21.7　流的多聚集器聚集操作……377
21.7.1　ChainedAggregatorDeclarer……377
21.7.2　分区上的局部聚集操作……379
21.7.3　全局聚集操作……379
21.7.4　含有多个聚集器的 partitionAggregate 操作……381
21.8　流的聚集操作……382
21.9　流的分区写入操作……383
21.10　查询操作……384
21.11　流的全局写入操作……384
21.12　流的操作与有向图构建……384
21.13　分组流……385
21.13.1　成员变量……385
21.13.2　逐行操作……385
21.13.3　分组流的分区聚集操作……386
21.13.4　查询操作……386
21.13.5　聚集操作……386
21.13.6　写入操作……387
21.14　利用流操作来构建 Topology 的例子……388
第 22 章　Trident 中流的交互操作……392
22.1　基本接口……392
22.2　JoinerMultiReducer……393
22.2.1　成员变量及构造函数……393
22.2.2　execute 方法……395
22.2.3　complete 方法……397
22.3　GroupedMultiReducerExecutor……397
22.4　MultiReducerProcessor……399
22.5　连接操作……401
22.6　流合并操作……403
第 23 章　Trident 中的 Bolt 节点……404
23.1　SubTopologyBolt……404
23.1.1　输入准备……404
23.1.2　成员变量……405
23.1.3　主要方法……406
23.2　Trident 中的 Bolt 执行器……409
23.2.1　ITridentBatchBolt 接口……410
23.2.2　TrackedBatch……410
23.2.3　定制的输出收集器……412
23.2.4　消息类型……414
23.2.5　数据成员分析……414
23.2.6　主要成员方法分析……416
第 24 章　Trident 的执行优化……420
24.1　节点类型……420
24.1.1　基本节点类型……420
24.1.2　Spout 节点……422
24.1.3　处理节点……422
24.1.4　分区节点……423
24.2　执行优化算法……426
24.2.1　节点组……426
24.2.2　节点组的合并算法……427
24.2.3　处理节点组中的分区节点……431
24.2.4　节点组以不同的方式收听相同流……431
24.2.5　执行优化后的节点组……434
24.2.6　计算节点组的并行度……434
第 25 章　Trident 与 DRPC……437
25.1　DRPC 服务器……438
25.1.1　DRPC 服务器的成员变量……438
52.1.2　DRPC 用户接口及其实现……439
25.1.3　DRPC Topology 端接口及其实现……440
25.1.4　启动 DRPC 服务器……441
25.2　DRPC 的客户端……442
25.3　DRPC 中 Spout 节点……443
25.4　DRPC Spout 的执行器……446

25.5 completeDRPC 操作 ····· 449
25.6 返回 DRPC 结果 ····· 451
第 26 章 Trident 的 Topology 构建器 ····· 453
26.1 基本工具函数 ····· 453
26.1.1 committerBatches ····· 453
26.1.2 fleshOutStreamBatchIds ····· 453
26.1.3 getOutputStreamBatchGroups ····· 454
26.2 TridentTopologyBuilder ····· 455
26.2.1 成员变量 ····· 455
26.2.2 设置 Spout 节点 ····· 456
26.2.3 设置 Bolt 节点 ····· 458
26.3 一个例子 ····· 460
第 27 章 多语言 ····· 462
27.1 ShellProcess ····· 462
27.2 ShellBolt ····· 464
27.2.1 成员变量 ····· 464
27.2.2 读写线程 ····· 465
27.3 ShellSpout ····· 467
第 28 章 Storm 中的配置项 ····· 469

第1章 总体架构与代码结构

Storm是由BackType开发的一个实时、分布式的计算平台，后来Twitter收购了BackType并将其源代码开放。在GitHub上（https://github.com/nathanmarz/storm），我们可以获得Storm的最新源代码及相关文档。

1.1 Storm的总体结构

Storm中会涉及的术语包括Stream、Spout、Bolt、Worker、Executor、Task、Stream Grouping和Topology，现简要介绍如下。

- Stream是被处理的数据。
- Spout是数据源。
- Bolt封装了数据处理逻辑。
- Worker是工作进程。一个工作进程中可以含有一个或多个Executor线程。
- Executor是运行Spout或Bolt处理逻辑的线程。
- Task是Storm中的最小处理单元。一个Executor中可以包含一个或多个Task，消息的分发都是从一个Task到另一个Task进行的。
- Stream Grouping定义了消息分发策略，定义了Bolt节点以何种方式接收数据。消息可以随机分配（Shuffle Grouping，随机分组），或者根据字段值分配（Fields Grouping，字段分组），或者广播（All Grouping，全部分组），或者总是发给同一个Task（Global Grouping，全局分组），也可以不关心数据是如何分组的（None Grouping，无分组），或者由自定义逻辑来决定，即由消息发送者决定应该由消息接收者组件的哪个Task来处理该消息（Direct Grouping，直接分组）。
- Topology是由消息分组方式连接起来的Spout和Bolt节点网络，它定义了运算处理的拓扑结构，处理的是不断流动的消息。除非杀掉Topology，否则它将永远运行下去。

Storm的基本结构如图1-1所示。

Storm集群中存在两种类型的节点：运行Nimbus服务的主节点和运行Supervisor服务的工作节点。Storm集群由一个主节点和多个工作节点组成。主节点上运行一个名为“Nimbus”的守护进程，用于分配代码、布置任务及检测故障。每个工作节点则运行一个名为“Supervisor”的守护

进程，用于监听工作、开始并终止工作进程。

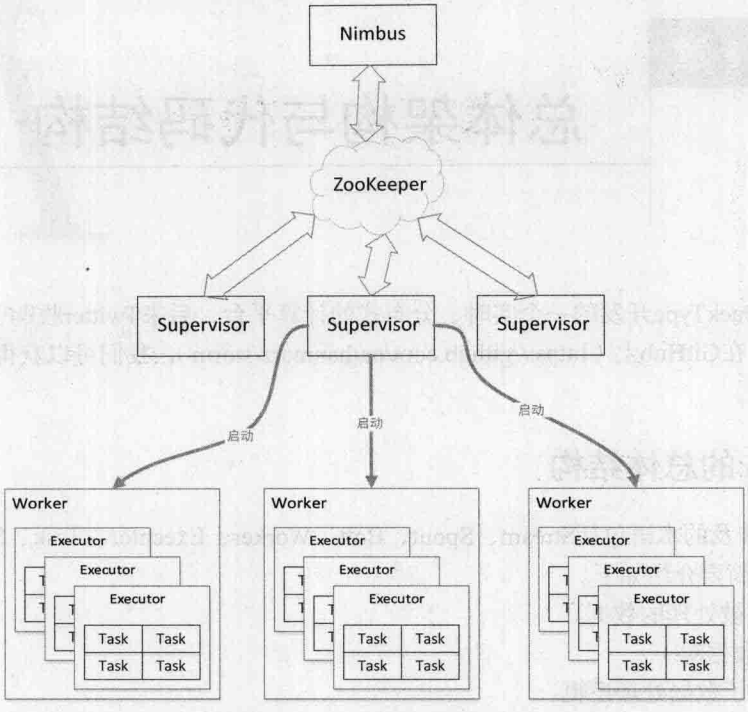


图1-1　Storm的基本结构

Nimbus和Supervisor都能快速失败并恢复，而且它们是无状态的，其元数据存储在ZooKeeper中，这使得系统具有很高的容错性。Nimbus与Supervisor之间的协调工作是通过ZooKeeper来完成的，它是Apache下面的开源项目，用于分布式系统的同步等，详情可参考http://zookeeper.apacheorg/。

Worker由Supervisor负责启动，一个Worker中可以有多个Executor线程，每个Executor中又可包含一个或多个Task。Task为Storm中的最小处理单元，它是Topology组件诸多并行度中的一个。每个Executor都会启动一个消息循环线程，用以接收、处理和发送消息。当Executor收到属于其下某一Task的消息后，就会调用该Task对应的处理逻辑对消息进行处理。

在逻辑上，Storm中消息的来源节点被称为Spout，消息的处理节点被称为Bolt。在系统中，可以存在多个Spout及Bolt，且每个Spout或Bolt都可设置不同的并行度，示例如图1-2所示。

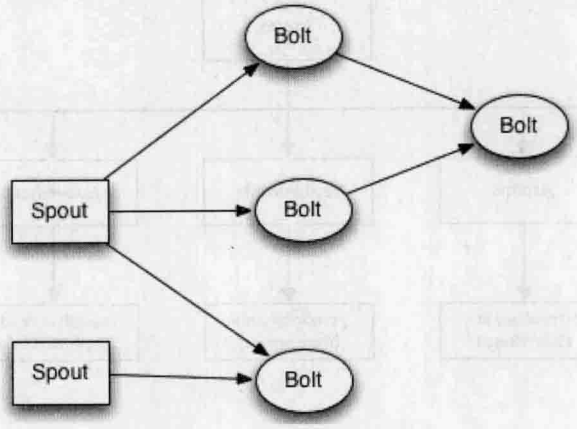


图1-2 示例图

1.2 Storm 的元数据

Storm采用ZooKeeper来存储Nimbus、Supervisor、Worker以及Executor之间共享的元数据，这些模块在重启之后，可以通过对应的元数据进行恢复。因此Storm的模块是无状态的，这是保证其可靠性及可扩展性的基础。了解元数据以及Storm如何使用这些元数据，有助于我们更好地理解Storm的设计。

1.2.1 元数据介绍

Storm在ZooKeeper中存储数据的目录结构如图1-3所示，这是一个根路径为/storm的树，树中的每一个节点代表ZooKeeper中的一个节点（znode），每一个叶子节点是Storm真正存储数据的地方。在图1-3中，从根节点到叶子节点的全路径代表了该数据在ZooKeeper中的存储路径，该路径可被用来写入或获取数据。

下面分别介绍ZooKeeper中每项数据的具体含义。

- /storm/workerbeats/<topology-id>/node-port：它存储由node和port指定的Worker的运行状态和一些统计信息，主要包括storm-id（也即topology-id）、当前Worker上所有Executor的统计信息（如发送的消息数目、接收的消息数目等）、当前Worker的启动时间以及最后一次更新这些信息的时间。在一个topology-id下面，可能有多个node-port节点。它的内容在运行过程中会被更新。
- /storm/storms/<topology-id>：它存储Topology本身的信息，包括它的名字、启动时间、运行状态、要使用的Worker数目以及每个组件的并行度设置。它的内容在运行过程中是不变的。

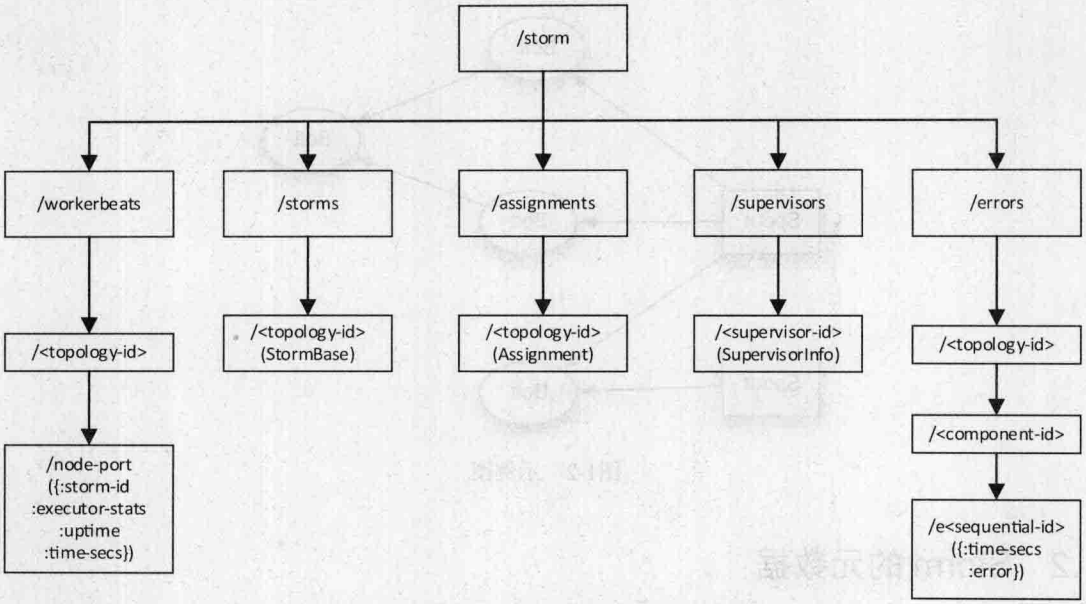


图1-3　Storm在ZooKeeper中存储的数据

- /storm/assignments/<topology-id>：它存储了Nimbus为每个Topology分配的任务信息，包括该Topology在Nimbus机器本地的存储目录、被分配到的Supervisor机器到主机名的映射关系、每个Executor运行在哪个Worker上以及每个Executor的启动时间。该节点的数据在运行过程中会被更新。
- /storm/supervisors/<supervisor-id>：它存储Supervisor机器本身的运行统计信息，主要包括最近一次更新时间、主机名、supervisor-id、已经使用的端口列表、所有的端口列表以及运行时间。该节点的数据在运行过程中也会被更新。
- /storm/errors/<topology-id>/<component-id>/e<sequential-id>：它存储运行过程中每个组件上发生的错误信息。<sequential-id>是一个递增的序列号，每一个组件最多只会保留最近的10条错误信息。它的内容在运行过程中是不变的（但是有可能被删除）。

1.2.2　Storm怎么使用这些元数据

了解了存储在ZooKeeper中的数据，我们自然想知道Storm是如何使用这些元数据的。例如，这些数据何时被写入、更新或删除，这些数据都是由哪种类型的节点（Nimbus、Supervisor、Worker或者Executor）来维护的。接下来，我们就简单介绍一下这些关系，希望读者能对Storm的整体设计实现有更深一层的认识。带上这些知识，能让你的Storm源码之路变得更加轻松愉快。

首先来看一下总体交互图，如图1-4所示。