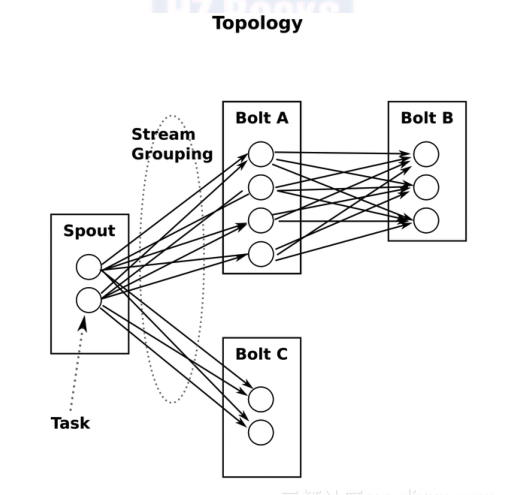
Topology拓扑

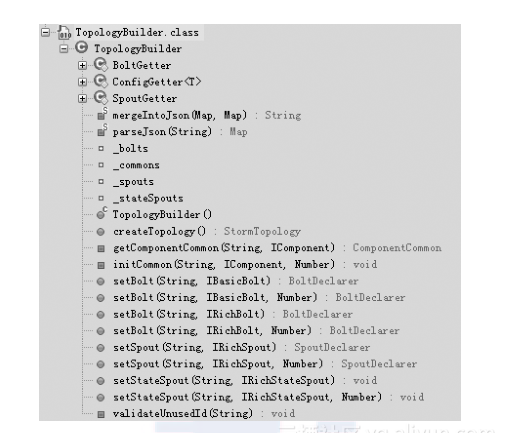
Hadoop上运行的是MapReduce作业，而在Storm上运行的是拓扑Topology，这两者之间是非常不同的。一个关键的区别是：一个MapReduce作业最终会结束，而一个Topology拓扑会永远运行（除非手动杀掉）。

从字面上解释Topology，就是网络拓扑，是指用传输介质互连各种设备的物理布局，是构成网络的成员间特定的物理的（即真实的），或者逻辑的（即虚拟的）排列方式。拓扑是一种不考虑物体的大小、形状等物理属性，而只使用点或者线描述多个物体实际位置与关系的抽象表示方法。拓扑不关心事物的细节，也不在乎相互的比例关系，只是以图的形式表示一定范围内多个物体之间的相互关系。从Storm角度考虑，它不是网络拓扑，但是又类似于网络拓扑的结构，所以取名Topology。

那么Storm的Topology指的是类似于网络拓扑图的一种虚拟结构。Storm的拓扑Topology类似于MapReduce任务，一个关键的区别是MapReduce任务运行一段时间后最终会完成，而Storm拓扑一直运行（直到杀掉它）。一个拓扑是由Spout和Bolt组成的图，Spout和Bolt之间通过流分组连接起来。下图形象地描述了Topology中的Spout和Bolt之间的关系。



通过对上图的理解可以看出，Topology是由Spout、Bolt、数据载体Tuple等构成的一定规则的网络拓扑图。Storm提供了TopologyBuilder类来创建Topology。打个比方，TopologyBuilder是Topology的骨架，Spout、Bolt是Topology的肉和血液。TopologyBuilder类的主要方法如下图所示。



TopologyBuilder实际上是封装了Topology的Thrift接口，也就是说Topology实际上是通过Thrift定义的一个结构，TopologyBuilder将这个对象建立起来，然后Nimbus实际上运行一个Thrift服务器，用于接收用户提交的结构。由于采用Thrift实现，所以用户可以用其他语言建立Topology，这样旧提供了比较方便的多语言操作支持。

Topology实例

下面从一个简单的例子开始介绍Topology的构建和定义，通过此案例能够基本理解Storm，并且能够构建一个简单的Topology。本实例使用Topology来统计一个句子中单词出现的频率。下面详细介绍如何设计和运行Topology，以及一些注意事项。

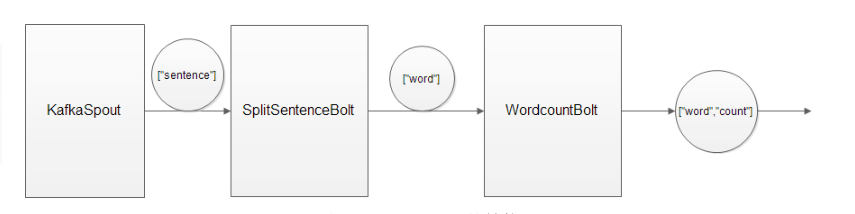
1. 设计Topology结构

在编写代码之前，首先要设计Topology。在理清数据处理逻辑之后，创建Topology就非常简单了。统计单词词频的Topology的大致结构如下图所示。可以将Topology分成3个部分：

一是数据源KafkaSpout，负责发送语句；

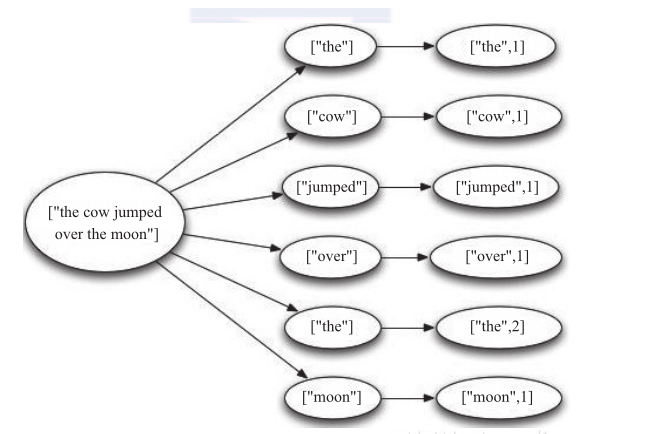
二是数据处理者SplitSentenceBolt，负责切分语句；

三是数据再处理者WordCountBolt，负责累加单词的频率。



1. 设计数据流

设计的Topology是从KafkaSpout中读取句子，并把句子划分成单词，然后汇总每个单词出现的次数，一个Bolt负责获取句子划分成单词，一个Bolt分别对应计算每一个单词出现的次数，然后Tuple在Spout和Bolt之间传递，如下图所示。



1. 代码实现
2. 构建Maven环境

为了开发Topology，需要把Storm相关的JAR包添加到CLASSPATH中，要么手动添加所有相关的JAR包，要么使用Maven来管理所有的依赖。Storm的JAR包发布在Clojars（一个Maven库），如果使用Maven，需要把下面的配置代码添加在项目的pom.xml中。

<repository>

<id>clojars.org</id>

<url>http:// clojars.org/repo</url>

</repository>

<dependency>

<groupId>storm</groupId>

<artifactId>storm</artifactId>

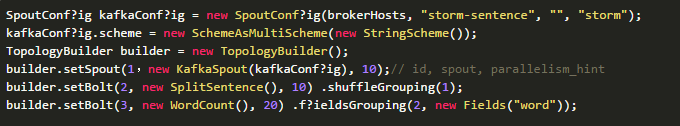
<version>0.8.2</version>

<scope>test</scope>

</dependency>

1. 定义Topology

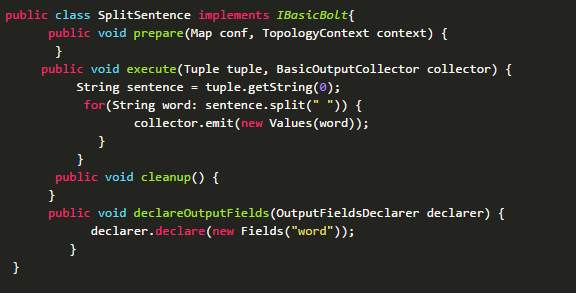
定义Topology的内部逻辑，代码如下：

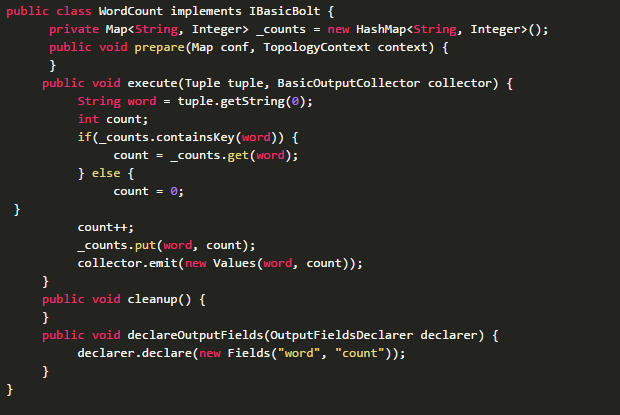


声明的Topology的Spout是从Kafka中读取句子，Spout用setSpout方法插入一个独特的ID到Topology中。Topology中的每个节点必须给予一个ID，ID是由其他Bolt用于订阅该节点的输出流，KafkaSpout在Topology中的ID为1.

setBolt用于在Topology中插入Bolt。在Topology中定义的第一个Bolt是切割句子的Bolt，该Bolt（即SplitSentence）将句子流转成单词流；setBolt的最后一个参数是Bolt的并行量，因为SplitSentence是10个并发，所以在Storm集群中有10个线程并行执行。当Topology遇到性能瓶颈时，可以通过增加Bolt并行数量来解决。setBolt方法返回一个对象，用来定义Bolt的输入。例如，SplitSentence约定使用组件ID为1的输出流，1是指已经定义的KafkaSpout。SplitSentence会消耗KafkaSpout发出的每一个元组。

SplitSentence的关键方法是execute，它将句子拆分成单词，并发出每个单词作为新的元组。另一个重要的方法是declareOutputFields，其中声明了Bolt输出元组的架构，这个方法声明它发出一个域为”word”的元组。SplitSentence对句子中的每个单词发射一个新的Tuple，WordCount在内存中维护每个单词出现次数的映射，WordCount每收到一个单词，都会更新内存中的统计状态。





1. Topology运行

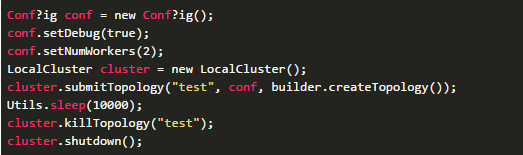
Topology运行有两种模式：本地模式和分布式模式。这两种模式的接口区别很大，使用场景也不相同。另外，下面还将介绍Topology的运行流程、方法调用过程以及并行度等。

1. Topology运行模式

Topology的运行模式可以分为本地模式和分布式模式，模式可以在配置文件中和代码中设置。

1. 本地模式

Storm用一个进程中的线程来模拟所有的Spout和Bolt。本地模式对开发和测试来说比较有用。storm-starter中的Topology是以本地模式运行的，可以看到Topology中的每一个组件发射的消息。示例代码如下：



首先，这段代码通过定义一个LocalCluster对象来定义一个进程内的集群。提交Topology给这个虚拟的集群和提交Topology给分布式集群相同。通过调用submitTopology方法来提交Topology，共有3个参数：要运行的Topology的名称、一个配置对象，以及要运行的Topology本身。

Topology是以名称来唯一区别的，可以用这名称来杀掉该Topology，而且必须显式地杀掉，否则它会一直运行。

conf对象可以配置内容很多，下面两个是最常见的：

* TOPOLOGY\_WORKERS（setNumWorkers）

定义希望集群分配多少个工作进程来执行这个Topology。Topology中的每个组件都需要线程来执行。每个组件到底用多少个线程是通过setBolt和setSpout来指定的。这些线程都运行在工作进程中。每一个工作进程包含一些节点的一些工作线程。例如，指定300个线程，60个进程，那么每个工作进程中要执行6个线程，而这6个线程可能属于不同的组件（Spout或Bolt）。可以调整每个组件的并行度以及这些线程所在的进程数量来调整Topology的性能。

* TOPOLOGY\_DEBUG（setDebug）

当它设置为true时，Storm会记录下每个组件发射的每条消息。这在本地环境调试Topology时很有用，但是在生产环境中如果这么做，则会影响性能。

1. 分布式模式

Storm由若干节点组成。提交Topology给Nimbus时，也会提交Topology代码。Nimbus负责分发代码和给Topology分配工作进程。如果一个工作进程挂掉了，Nimbus节点会将其重新分配到其他节点。分布式模式提交拓扑的代码如下：

https://images2017.cnblogs.com/blog/855959/201709/855959-20170914212541797-47826171.png

在Storm代码编写完成之后，需要打包成JAR包放到Nimbus中运行。在打包时，不需要把依赖的JAR都打进去，否则运行时会出现重复的配置文件错误导致Topology无法运行，因为在Topology运行之前，会加载本地的storm.yaml配置文件。

在Nimbus运行的命令如下：

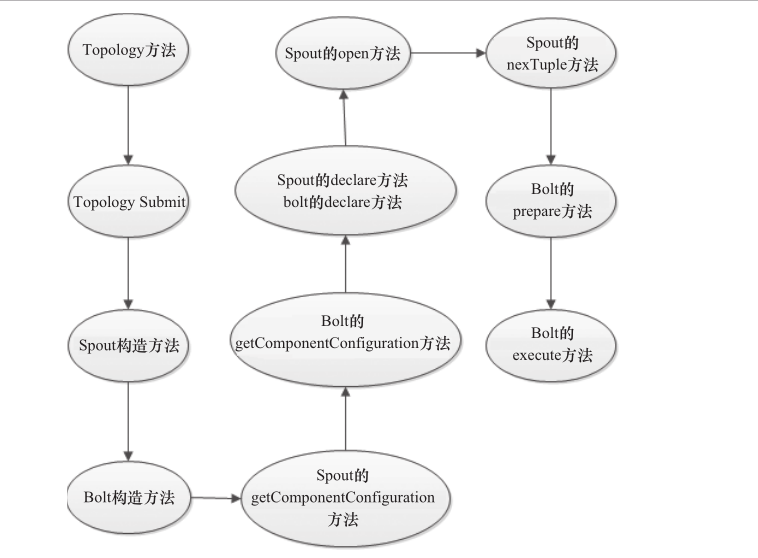
https://images2017.cnblogs.com/blog/855959/201709/855959-20170914212608953-1409555444.png

1. Topology运行流程

在Topology的运行流程中，有几点需要特别说明。

1. 提交Topology后，Storm会把代码先存放到Nimbus节点的inbox目录下；之后，把当前Storm运行的配置生成一个stormconf.ser文件放到Nimbus节点的stormdist目录中，此目录中同时还有序列化之后的Topology代码文件。
2. 在设定Topology关联的Spout和Bolt时，可以同时设置当前Spout和Bolt的Executor和Task数量。在默认情况下，一个Topology的Task总和与Executor的总和一致。之后，系统根据Worker的数量，尽量将这些Task平均分配到不同的Worker上执行。Worker在哪个Supervisor节点上运行是由Storm本身决定的。
3. 在任务分配好之后，Nimbus节点将任务的信息提交到ZooKeeper集群，同时在ZooKeeper集群中有Workerbeats，这里存储了当前Topology所有Worker进程的心跳信息。
4. Supervisor节点不断轮询ZooKeeper集群，在ZooKeeper的assignments中保存了所有Topology的任务分配信息、代码存储目录、任务之间的关联关系等，Supervisor通过轮询此节点的内容来领取自己的任务，启动Worker进程运行。
5. 一个Topology运行之后，不断通过Spout来发送流，通过Bolt来不断处理接收到的流，流是无界的。最后一步会不间断地执行，除非手动结束该Topology。
6. Topology的方法调用流程

Topology中的流处理时，调用方法的过程如下图所示。



Topolog方法调用的过程有如下一些要点：

1. 每个组件（Spout或者Bolt）的构造方法和declareOutputFields方法都只被调用一次。
2. open方法和prepare方法被调用多次。在入口函数中设定的setSpout或者setBolt中的并行度参数是指Executor的数量，是负责运行组件中的Task的线程数量，此数量是多少，上述两个方法就会被调用多少次，在每个Executor运行时调用一次。
3. nextTuple方法和execute方法是一直运行的，nextTuple方法不断发射Tuple，Bolt的execute不断接收Tuple进行处理。只有这样不断地运行，才会产生无界的Tuple流，体现实时性。这类似于Java线程的run方法。
4. 提交一个Topology之后，Storm创建Spout/Bolt实例并进行序列化。之后，将序列化的组件发送给所有任务所在的节点（即Supervisor节点），在每一个任务上反序列化组件。
5. Spout和Bolt之间、Bolt和Bolt之间的通信，通过ZeroMQ的消息队列实现。
6. ack和fail方法。在一个Tuple成功处理之后，需要调用ack方法来标记成功，否则调用fail方法标记失败，重新处理该Tuple。
7. Topology并行度

在Topology的执行单元中，有几个和并行度相关的概念。

1. Worker

每个Worker都属于一个特定的Topology，每个Supervisor节点的Worker可以有多个，每个Worker使用一个单独的端口，Worker对Topology中的每个组件运行一个或者多个Executor线程来提供Task的执行服务。

1. Executor

Executor是产生于Worker进程内部的线程，会执行同一个组件的一个或者多个Task。

1. Task

实际的数据处理由Task完成。在Topology的生命周期中，每个组件的Task数量不会变化，而Executor的数量却不一定。Executor数量小于等于Task的数量，在默认情况下，二者是相等的。

在运行一个Topology时，可以根据具体的情况来设置不同数量的Worker、Task、Executor，设置的位置也可以在多个地方。

1. Worker设置

可以设置yaml中的topology.workers属性。在代码中通过Config的setNumWorkers方法设定。

1. Executor设置

通过Topology的入口类中的setBolt、setSpout方法的最后一个参数指定，如果不指定，则使用默认值1。

1. Task设置

在默认情况下，和executor数量一致。在代码中通过TopologyBuilder的setNumTasks方法设定具体某个组件的Task数量。

1. 终止Topology

在Nimbus启动的节点上，使用下面的命令来终止一个Topology的运行。

storm kill topologyName

执行kill之后，通过UI界面查看Topology状态，其先变成KILLED状态，清理完本地目录和ZooKeeper集群中与当前Topology相关的信息之后，此Topology将彻底消失。

1. Topology跟踪

提交Topology后，可以在Storm UI界面查看整个Topology运行的过程。