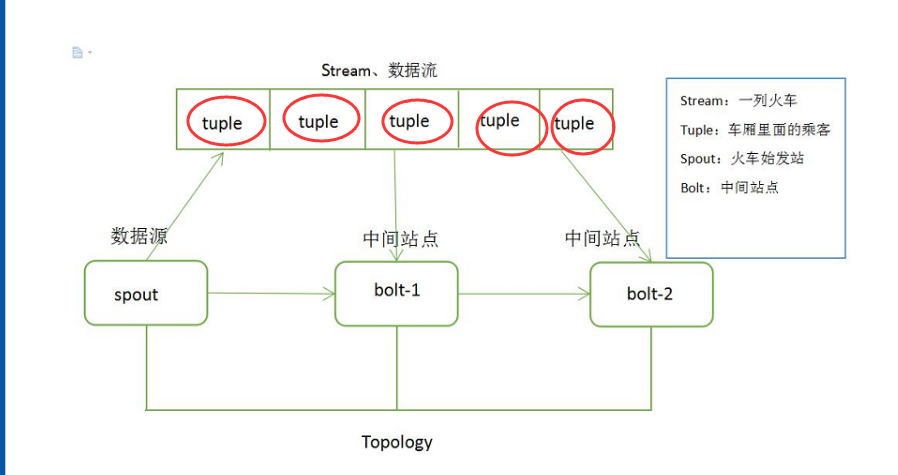
Tuple是Storm的主要数据结构，并且是Storm中使用的最基本单元、数据模型和元组。



Tuple描述

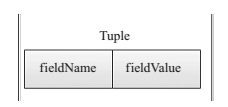
Tuple就是一个值列表，Tuple中的值可以是任何类型的，动态类型的Tuple的fields可以不用声明；默认情况下，Storm中的Tuple支持私有类型、字符串、字节数组等作为它的字段值，如果使用其他类型，就需要序列化该类型。

Tuple的字段默认类型有：integer、float、double、long、short、string、byte、binary( byte[] )。

Tuple元组，是消息传递的基本单元，是一个命名的值列表，元组中的字段可以是任何类型的对象。Storm使用元组作为其数据模型，元组支持所有的基本类型、字符串合字节数组作为字段值，只要实现类型的序列化接口就可以使用该类型的对象。

元组本来应该是一个key-vaue的Map，但是由于各个组件间传递的元组的字段名称已经事先定义好，所以只要按序把元组填入各个value即可，所以元组是一个value的List。

Tuple是Storm采用的数据表示模型，所有的数据都以Tuple的形式在各个组件之间流动。Tuple是一组字段列表，每个字段由一个字段名和字段值组成，每个Tuple类似于数据库中的一行记录。在默认的情况下，Tuple的字段类型可以是integer、long、short、byte、string、double、float、boolean和byte array。当然，你也可以通过实现序列化器自定义类型。



Tuple可以理解成键值对。例如，创建一个Bolt要发送两个字段（命名为double和triple），其中键就是定义在declareOutputFields方法中的Fields对象，值就是在emit方法中发送的Values对象。

以下是一个简单例子

public class DoubleAndTripleBolt extends BaseRichBolt {

OutputCollectorBase \_collector;

@Override

public void prepare(Map conf, TopologyContext context, OutputCollectorBase collector) {

\_collector = collector;

}

@Override

public void execute(Tuple input) {

int val = input.getInteger(0);

\_collector.emit(input, new Values(val\*2, val\*3));

\_collector.ack(input);

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("double", "triple"));

}

}

此外，在使用的Storm Java包中，backtype.storm.tuple主要有以下几个类:

Fileds.class

MessageId.class

Tuple.class

TupleImpl.class

Values.class

列出以上内容是为了更好地理解Tuple，这样能够从本质上理解Tuple，在使用时更加得心应手。

Tuple的生命周期

了解一个Tuple的生命周期就需要查看源码，如下的Java代码展示了Spout（消息源）接口发出Tuple（消息）的整个过程。

public interface ISpout extends Serializable {

void open(Map conf, TopologyContext context, SpoutOutputCollector collector);

void nextTuple();

void ack(Object msgId);

void fail(Object msgId);

void close();

}

首先，Storm调用Spout（消息源）的nextTuple方法类获取下一个Tuple，Spout通过Open方法的参数提供的SpoutOoutputCollector将新Tuple发射到其中一个输出消息流。注意：发射Tuple时，Spout提供一个message-id，通过这个ID来追踪该Tuple。

接下来，Storm跟踪该Tuple的树形结构是否成功创建，并根据messageid调用Spout中的ack函数，以确认Tuple是否被完全处理。如果Tuple超时，则调用Spout的fail方法。

由此看出，同一个Tuple不管是acked，还是failed都是由创建它的Spout发出并维护的，所以，即使Spout在集群环境中同时执行很多的任务，该Tuple也不会被其他任务调用或生成acked或failed状态。总之，Storm会利用内部的Acker机制保证每个Tuple被可靠地处理。最后，在任务完成后，Spout调用Close方法结束Tuple的使命。

比如

