edwardsbean的专栏

记录一些东西

Flume数据传输事务分析

标签： flume flume事务 flume源码

2014-09-30 14:59 4270人阅读 评论(2) 收藏 举报

分类：

Flume（8）

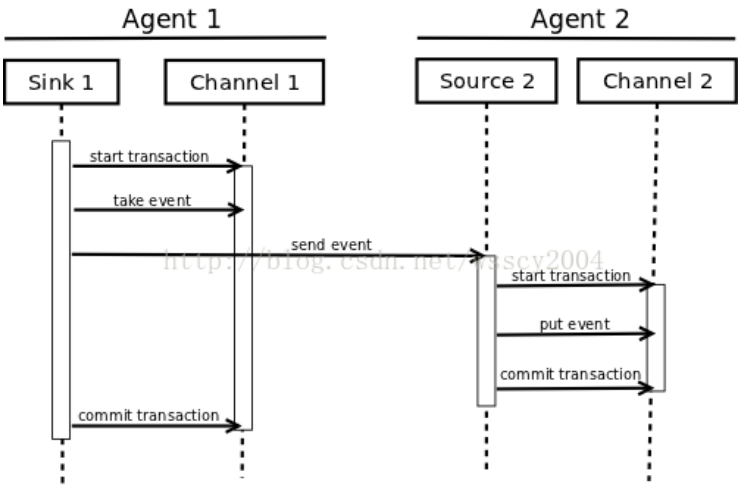
目录(?) [+]

Flume数据传输事务分析

本文基于ThriftSource,MemoryChannel,HdfsSink三个组件，对Flume数据传输的事务进行分析，如果使用的是其他组件，Flume事务具体的处理方式将会不同。一般情况下，用MemoryChannel就好了，我们公司用的就是这个，FileChannel速度慢，虽然提供日志级别的数据恢复，但是一般情况下，不断电MemoryChannel是不会丢数据的。

Flume提供事物操作，保证用户的数据的可靠性，主要体现在：

- 数据在传输到下个节点时(通常是批量数据)，如果接收节点出现异常，比如网络异常，则回滚这一批数据。因此有可能导致数据重发
- 同个节点内，Source写入数据到Channel,数据在一个批次内的数据出现异常，则不写入到Channel。已接收到的部分数据直接抛弃，靠上一个节点重发数据。



编程模型

Flume在对Channel进行Put和Take操作的时候，必须要用事物包住,比如：

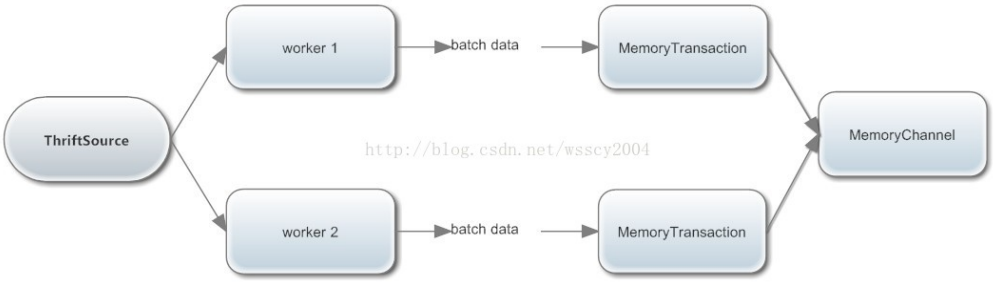
```
Channel ch = new MemoryChannel();
Transaction txn = ch.getTransaction();
//事物开始
txn.begin();
try {

  Event eventToStage = EventBuilder.withBody("Hello Flume!",
                       Charset.forName("UTF-8"));
  //往临时缓冲区Put数据
  ch.put(eventToStage);
  //或者ch.take()

  //将这些数据提交到channel中
  txn.commit();
} catch (Throwable t) {
  txn.rollback();


  if (t instanceof Error) {
    throw (Error)t;
  }
} finally {
  txn.close();
}
```

Put事务流程

Put事务可以分为以下阶段：

- doPut:将批数据先写入临时缓冲区putList
- doCommit:检查channel内存队列是否足够合并。
- doRollback:channel内存队列空间不足，抛弃数据

我们从Source数据接收到写入Channel这个过程对Put事物进行分析。



ThriftSource会spawn多个Worker线程(ThriftSourceHandler)去处理数据，Worker处理数据的接口，我们只看batch批量处理这个接口：

```
    @Override
    public Status appendBatch(List<ThriftFlumeEvent> events) throws TException {

      List<Event> flumeEvents = Lists.newArrayList();
      for(ThriftFlumeEvent event : events) {
        flumeEvents.add(EventBuilder.withBody(event.getBody(),
          event.getHeaders()));
      }

      //ChannelProcessor,在Source初始化的时候传进来.将数据写入对应的Channel
      getChannelProcessor().processEventBatch(flumeEvents);
      ...
```

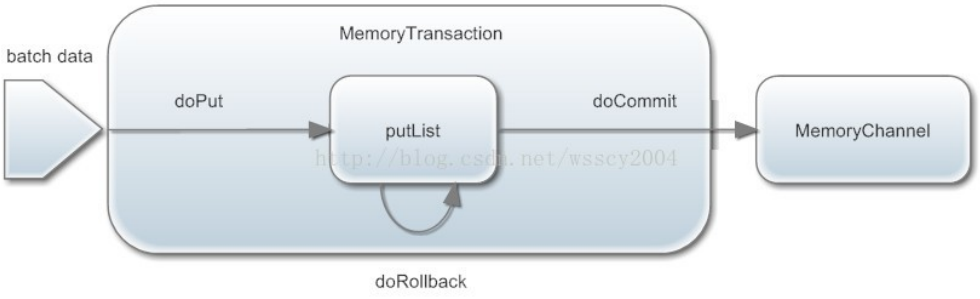
```
      return Status.OK;
    }
```

事务逻辑都在processEventBatch这个方法里：

```
public void processEventBatch(List<Event> events) {
    ...
    //预处理每行数据，有人用来做ETL嘛
    events = interceptorChain.intercept(events);
    ...
    //分类数据，划分不同的channel集合对应的数据

    // Process required channels
    Transaction tx = reqChannel.getTransaction();
    ...
        //事务开始，tx即MemoryTransaction类实例
        tx.begin();
        List<Event> batch = reqChannelQueue.get(reqChannel);
        for (Event event : batch) {
          // 这个put操作实际调用的是transaction.doPut
          reqChannel.put(event);
        }
        //提交，将数据写入Channel的队列中
        tx.commit();
      } catch (Throwable t) {
        //回滚
        tx.rollback();
        ...
      }
    }
    ...
  }
```

每个Worker线程都拥有一个Transaction实例，保存在Channel(BasicChannelSemantics)里的ThreadLocal变量currentTransaction.

那么，事务到底做了什么？



实际上，Transaction实例包含两个双向阻塞队列LinkedBlockingDeque(感觉没必要用双向队列，每个线程写自己的putList，又不是多个线程？),分别为：

- putList
- takeList

对于Put事物操作，当然是只用到putList了。putList就是一个临时的缓冲区，数据会先put到putList,最后由commit方法会检查channel是否有足够的缓冲区，有则合并到channel的队列。

channel.put -> transaction.doPut：

```
    protected void doPut(Event event) throws InterruptedException {
      //计算数据字节大小
```

```
      int eventByteSize = (int)Math.ceil(estimateEventSize(event)/byteCapacitySlotSize)
;
      //写入临时缓冲区putList
      if (!putList.offer(event)) {
        throw new ChannelException(
          "Put queue for MemoryTransaction of capacity " +
            putList.size() + " full, consider committing more frequently, " +
            "increasing capacity or increasing thread count");
      }
      putByteCounter += eventByteSize;
    }
```

transaction.commit：

```
@Override
    protected void doCommit() throws InterruptedException {
      //检查channel的队列剩余大小是否足够
      ...

      int puts = putList.size();
      ...
      synchronized(queueLock) {
        if(puts > 0 ) {
          while(!putList.isEmpty()) {
            //写入到channel的队列
            if(!queue.offer(putList.removeFirst())) {
              throw new RuntimeException("Queue add failed, this shouldn't be able to h
appen");
            }
          }
        }
        //清除临时队列
        putList.clear();
        ...
      }
      ...
    }
```

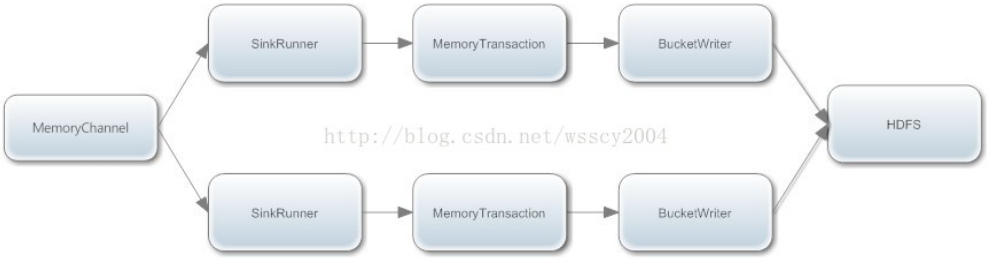
如果在事务期间出现异常，比如channel剩余空间不足，则rollback:

```
@Override
    protected void doRollback() {
    ...
        //抛弃数据，没合并到channel的内存队列
        putList.clear();
      ...
    }
```

Take事务

Take事务分为以下阶段：

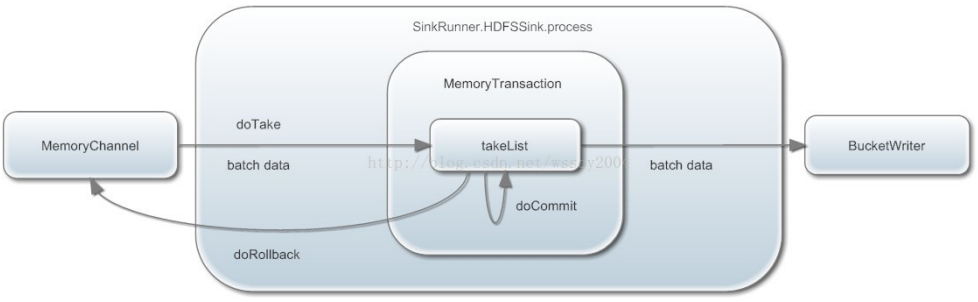
- doTake:先将数据取到临时缓冲区takeList
- 将数据发送到下一个节点
- doCommit:如果数据全部发送成功，则清除临时缓冲区takeList
- doRollback:数据发送过程中如果出现异常，rollback将临时缓冲区takeList中的数据归还给channel内存队列。



Sink其实是由SinkRunner线程调用Sink.process方法来了处理数据的。我们从HdfsEventSink的process方法说起，Sink类都有个process方法，用来处理传输数据的逻辑。：

```
public Status process() throws EventDeliveryException {
    ...
    Transaction transaction = channel.getTransaction();
    ...
    //事务开始
    transaction.begin();
    ...
      for (txnEventCount = 0; txnEventCount < batchSize; txnEventCount++) {
        //take数据到临时缓冲区,实际调用的是transaction.doTake
        Event event = channel.take();
        if (event == null) {
          break;
        }
        ...
      //写数据到HDFS
      bucketWriter.append(event);
      ...
      // flush all pending buckets before committing the transaction
      for (BucketWriter bucketWriter : writers) {
        bucketWriter.flush();
      }
      //commit
      transaction.commit();
      ...
    } catch (IOException eIO) {
      transaction.rollback();
      ...
    } finally {
      transaction.close();
    }
  }
```

大致流程图：



接着看看channel.take，作用是将数据放到临时缓冲区,实际调用的是transaction.doTake:

```
protected Event doTake() throws InterruptedException {
      ...
```

```
      //从channel内存队列取数据
      synchronized(queueLock) {
        event = queue.poll();
      }
      ...
      //将数据放到临时缓冲区
      takeList.put(event);
      ...
      return event;
    }
```

接着，HDFS写线程bucketWriter将take到的数据写到HDFS,如果批数据都写完了，则要commit了：

```
protected void doCommit() throws InterruptedException {
    ...
    takeList.clear();
    ...
}
```

很简单，其实就是清空takeList而已。如果bucketWriter在写数据到HDFS的时候出现异常，则要rollback:

```
protected void doRollback() {
      int takes = takeList.size();
      //检查内存队列空间大小，是否足够takeList写回去
      synchronized(queueLock) {
        Preconditions.checkState(queue.remainingCapacity() >= takeList.size(), "Not eno
ugh space in memory channel " +
            "queue to rollback takes. This should never happen, please report");
        while(!takeList.isEmpty()) {
          queue.addFirst(takeList.removeLast());
        }
        ...
      }
      ...
    }
```


上一篇 Actor生命周期理解
下一篇 Java Metrics

我的同类文章

Flume（8）

• HdfsSink原理解析	2014-03-26 阅读 564	• Flume-ng启动过程分析	2014-03-26 阅读 1876
• Flume-ng出现HDFS IO err…	2014-03-26 阅读 2832	• Flume-ng的HdfsSink出现L…	2014-03-17 阅读 2446
• Flume C# Thrift客户端	2014-02-26 阅读 1242	• Flume-ng ThriftSource原理…	2014-02-26 阅读 2326
• 修改Flume Log4j Appender	2014-01-25 阅读 2326	• 使用Flume Log4j Appende…	2014-01-25 阅读 6175

猜你在找

Flume数据采集系统
HDFS精讲
linux设备驱动之USB数据传输分析
蓝牙基带数据传输机理分析

Hadoop学习从零到一系列课程（2）---HDFS和YARN精
数据结构基础系列(3)：栈和队列
大数据编程语言：Java基础
Linux I2C驱动分析二----I2C板级设备扫描和数据
网络中数据传输过程的分析
linux设备驱动之USB数据传输分析

查看评论

2楼 DreamOfHeaven 2015-09-28 14:43发表

施总好←←

1楼 mango_song 2015-04-14 06:14发表


不错，赞一个

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap