TRANSWARP
DATA HUB

Transwarp Data Hub Version 4.3.3
StreamSQL使用手册

星环信息科技（上海）有限公司

Transwarp Data Hub v4.3.3

StreamSQL使用手册

版本：1.0v

发布日期： 2016-03-26

版本号：T001433-10-010

免责声明

本说明书依据现有信息制作，其内容如有更改，恕不另行通知。星环信息科技（上海）有限公司在编写该说明书的时候已尽最大努力保证期内容准确可靠，但星环信息科技（上海）有限公司不对本说明书中的遗漏、不准确或印刷错误导致的损失和损害承担责任。具体产品使用请以实际使用为准。

注释：Hadoop® 和 SPARK® 是Apache™ 软件基金会在美国和其他国家的商标或注册的商标。

修订历史记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本T001433-10-010（2016-03）第一次发布。

文档版本T001430-10-010（2016-01）第一次发布。

目录

插图清单

1. Inceptor StreamSQL简介

在4.3版本之前，Transwarp Data Hub（TDH）上的流处理应用必须用Java开发，这将流处理应用的开发门槛设置得非常高。从TDH 4.3开始，全新的StreamSQL让用户通过命令行、JDBC或者ODBC等常用的SQL交互方式，与Inceptor交互，用SQL创建流、管理流并对流上的数据进行查询，大大降低了流应用开发的门槛。所以，星环科技推荐多数用户采用StreamSQL开发流应用。StreamSQL可以应用于大量常见业务场景，包括ETL工具，规则报警工具等。

1.1. StreamSQL的优势

相对于采用编程的方式开发流应用，采用StreamSQL具有以下优势：

- **极高的易用性**

  以往的流处理平台有较高的入门门槛，比如Spark Streaming，Storm，必须对框架以及流处理本身，甚至是底层技术比较熟悉的情况下，才能写出高效的流处理程序。这大大地限制了流处理的推广和应用。而使用StreamSQL，用户只需要有编写普通SQL的经验，就可以写出高效、安全、稳定的流处理应用。

- **性能提升**

  在一些条件下，采用StreamSQL的方式甚至比编程方式获得更高的性能提升。这是因为StreamSQL做了一些特殊优化，在编程模式下无法轻易实现。比如，多条针对同一个输入流的SQL只需读取一份数据；增加迭代框架，使得原本无法利用Spark API进行优化的迭代计算效率大大提升。

- **产品化程度高**

  通过编程的方式来实现流处理的另一个问题是产品化程度非常低。由于编程有较高的自由度，出现问题的可能性很大；而又由于编程的方式将流处理平台和用户程序绑定在一起，用户没办法很好地区分是自己代码的问题还是平台的问题，导致无法及时地分析出错原因。SQL作为一个通用的接口将大大地提高产品化程度。

- **迁移成本低**

  用户原有的业务逻辑很多是通过SQL实现的，如果通过编程的方式迁移到流上，迁移成本非常高，还不能保证迁移后的逻辑是否正确。而一旦采用StreamSQL的方式，用户只需要修改少量SQL，迁移成本几乎接近零。

1.2. StreamSQL的技术指标

StreamSQL有以下技术指标：

- **99%的ANSI SQL 2003的支持率**

  为了降低应用迁移成本，StreamSQL使用了完整的SQL编译器,支持ANSI SQL 92和SQL 99标准,并且支持ANSI SQL 2003 OLAP核心扩展,可以满足绝大部分现有业务对SQL的要求,方便应用平 滑迁移。

- **强大的优化器提升性能**

  除了更好的SQL语义分析层以外，StreamSQL包含强大的优化器保证SQL在引擎上有最佳的性能。包含3级优化器:首先是基于规则的优化器,应用静态优化规则并生成一个逻辑执行计划,其次是基于成本的优化器,通过衡量多个不同执行计划的CPU,IO和网络成本,来选择一个更合理的计划并生成物理执行计划;最后是代码生成器,对一些比较核心的执行逻辑生成更高效的执行代码或者Java Byte Code,从而保证SQL业务在分布式平台上有最佳性能。

- **支持按数据字段时间切分滑动窗口和跳动窗口**

  常见的流框架一般只针对系统时间做窗口切分，但这往往达不到实际应用的要求。因此，StreamSQL实现了复杂内部逻辑，使得用户通过简单的SQL语法就能指定特定数据字段作为窗口的依据。用户可以通过与SQL 2003兼容的Window语法来创建滑动窗口或者跳动窗口，并在窗口上进行各种复杂的聚合操作。

- **支持多种输出方式**

  包括Hyperbase，Hive以及Holodesk等。

- **支持企业数据总线**

  从一个流读入数据，再将其输入另一个流，比如一个常见的企业数据总线以Kafka为存储中间件。

- **支持运行时隔离**

  保证多用户安全、稳定运行各自StreamSQL。原来流处理应用要么物理上完全隔离，但无法共享硬件资源也不方便管理；要么部署在同一个集群，但可能导致资源分配不均，甚至有误操作影响其它用户的可能。而StreamSQL抽离出运行时隔离概念，能保证用户启动、查看和停止流处理应用的操作只能针对当前所在隔离层进行。

- **StreamSQL与ANSI SQL 2003无缝兼容**

  比如，用户可以使用 `CREATE STREAM` 创建流；使用 `DESCRIBE STREAM` 获取流的信息；使用 `ALTER STREAM` 修改已经定义的流；使用 `DROP STREAM` 删除流；使用 `SHOW STREAMS` 查看已经创建的流。

2. 快速入门

在开始正式介绍StreamSQL之前，我们先演示一个简单的StreamSQL例子。在这个例子中，我们将使用Kafka的console工具生成一些简单的数据，并让StreamSQL来处理。这个演示包含三个部分：

1. 建一个Kafka数据源；
2. 在Inceptor中建一个stream并触发stream job；
3. 在Inceptor中处理stream从Kafka数据源接收的数据。

我们下面演示的是Kafka和Zookeeper在 **非安全模式** 下的操作——Kafka和Zookeeper没有开启Kerberos。如果您要在安全模式下操作，可以参考第 A.2.6 节 “Kafka权限管理”。

2.1. 建Kafka数据源

1. 登陆您集群中的一台安装了Kafka的节点。进入/usr/lib/kafka/bin目录，该目录下有建Kafka数据源的所需要的一些脚本。
2. 建一个Kafka Topic

   执行下面指令，运行/usr/lib/kafka/bin目录下的kafka-create-topic.sh脚本：

   ```
   ./kafka-create-topic.sh --partition 3 --topic demo --broker tw-node127:9092 --
   zookeeper 172.16.1.128:2181
   creation succeeded!
   ```

   该topic名称为demo，使用172.16.1.128上的Zookeeper，分3个partition。
3. 查看Kafka Topic

   执行下面指令，运行/usr/lib/kafka/bin目录下的kafka-list-topic.sh脚本：

   ```
   ./kafka-list-topic.sh --zookeeper 172.16.1.128:2181
   topic: demo partition: 0 leader: 5 replicas: 5 isr: 5
   topic: demo partition: 1 leader: 6 replicas: 6 isr: 6
   topic: demo partition: 2 leader: 4 replicas: 4 isr: 4
   ```

   我们可以看到刚才建的名为demo的topic和一些相关信息，这些信息的意义会在之后的介绍中解释。
4. 建Kafka producer并发布消息

   执行下面指令，运行/usr/lib/kafka/bin目录下的kafka-console-producer.sh脚本：

   ```
   ./kafka-console-producer.sh --broker-list 172.16.1.128:9092 --topic demo
   ```

这里我们指定了使用172.16.1.128节点为Kafka broker（详细介绍见后文），并且指定了producer发布消息的topic为demo。现在，我们可以在命令行中输入一些消息，这些消息都将被发布给demo：

```
hello
world
```

5. 到此，我们已经建好了一个Kafka数据源，并发布了一些消息。先不要停止上面producer的进程，让它保持运行，您可以继续在命令行中输入消息。现在您可以打开另一个窗口登陆集群，进入Inceptor，建一个stream并触发stream job的开始。

2.2. 建stream及触发streamjob

1. 登陆您集群中的任意一个节点，连接到Inceptor。这里，我们连接一个LDAP认证的Inceptor Server 2。

```
beeline -u "jdbc:hive2://localhost:10000/default" -n hive -p 123
```

2. 建一个stream

```
0: jdbc:hive2://localhost:10000/default> CREATE STREAM demo_stream(id
 INT, letter STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
 TBLPROPERTIES("topic"="demo","kafka.zookeeper"="172.16.1.128:2181");
```

这里，我们建了一个名为demo_stream的stream，它使用Kafka为源，接收发布给名为demo的topic的消息，将接收的消息按“,”分隔为两列：id（类型为INT）和letter（类型为STRING）。

3. 查看stream

```
0: jdbc:hive2://localhost:10000/default> SHOW STREAMS;
+--------------+
|   tab_name   |
+--------------+
| demo_stream  |
+--------------+
```

我们可以看到刚刚建好的demo_stream。

4. 创建并触发一个stream job

   a. 建一张新表demo_table，它的列格式需要和demo_stream对应列的格式相同：

```
0: jdbc:hive2://localhost:10000/default> CREATE TABLE demo_table(id INT,
 letter STRING);
```

   b. 向demo_table插入demo_stream中的数据，这个操作会触发stream job的执行：

```
0: jdbc:hive2://localhost:10000/default> INSERT INTO demo_table SELECT * FROM
 demo_stream;
```

```
+------+
| _c0  |
+------+
+------+
```

5. 列出正在运行的stream job

   执行下面指令：

```
LIST streamjobs;
```

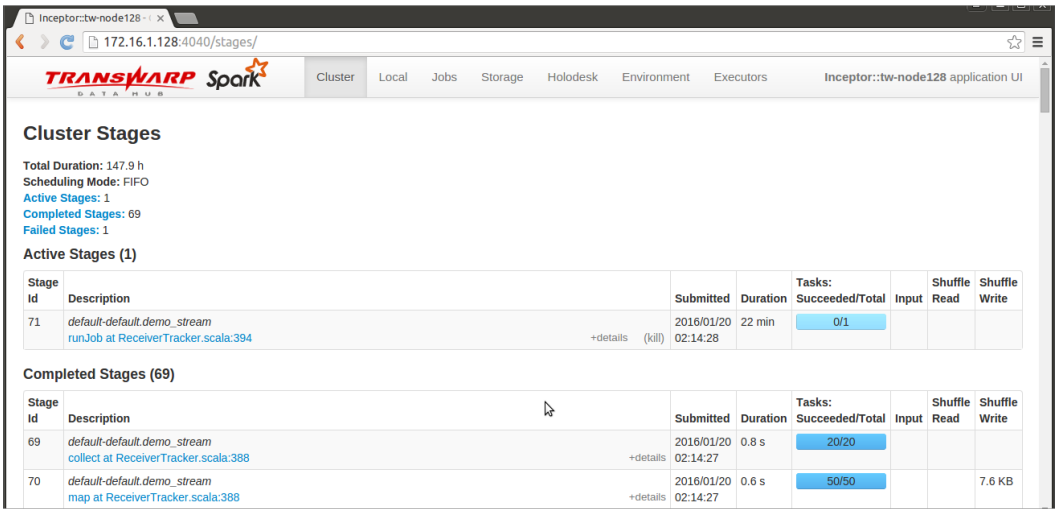
   我们可以看到下面输出：

```
0: jdbc:hive2://localhost:10000/default> LIST streamjobs;
+----------------------------------+-----------------------------------------------+----------+
|             streamid             |                      sql                      |  status  |
+----------------------------------+-----------------------------------------------+----------+
| 29008ed34b9e45bca784362948b88a85 | INSERT INTO demo_table SELECT * FROM demo_stream | running  |
+----------------------------------+-----------------------------------------------+----------+
```

   输出中包含streamid，触发streamjob的sql和status。

6. 在Inceptor管理界面查看stream job运行状态

   打开浏览器，访问http://<inceptor_server_ip>:4040，可以在Inceptor的管理界面看到当前正在运行的stream job：



   可以看到在“Active Stages”下有正在运行的demo_stream。

7. 此时demo_stream已经开始接收发布到之前创建的demo的消息。
   需要注意的是，demo_stream对发布到demo的消息的接收是从streamid=29008ed34b9e45bca784362948b88a85的streamjob触发开始的，也就是说从执行INSERT开始的，在执行INSERT之前发布到demo的消息都不会被demo_stream接收。所以，目前demo_table中没有任何记录：

```
0: jdbc:hive2://localhost:10000/default> SELECT * FROM demo_table;
+-----+---------+
| id  | letter  |
+-----+---------+
```

```
+-----+---------+
```

2.3. 接收并处理Kafka传来的数据

1. 切换到正在运行向demo发布数据的Kafka producer的页面：

```
./kafka-console-producer.sh --broker-list 172.16.1.128:9092 --topic demo
hello
world
```

由于“hello”，“world”都是在streamid=29008ed34b9e45bca784362948b88a85的stream job触发前发布的，这两条消息都不会被demo_stream接收。

2. 在命令行中输入一些数据。由于我们已经规定了demo_stream接收消息的类型是由“,”分隔的两列文本，我们需要输入这个类型的消息，以便demo_stream处理。输入：

```
1,a
1,b
1,a
1,a
```

3. 切换回到Inceptor命令行的窗口，现在查看demo_table中的数据，我们可以看到demo_table中有我们刚才发布的四条消息：

```
0: jdbc:hive2://localhost:10000/default> SELECT * FROM demo_table;
+-----+---------+
| id  | letter  |
+-----+---------+
| 1   | a       |
| 1   | b       |
| 1   | a       |
| 1   | a       |
+-----+---------+
```

4. 现在我们可以在demo_table上进行一些查询：

```
0: jdbc:hive2://localhost:10000/default> SELECT COUNT(*) FROM demo_table GROUP BY
 letter;
+------+
| _c0  |
+------+
| 1    |
| 3    |
+------+
```

5. 演示到此结束，下面我们详细介绍StreamSQL所需的基础知识和操作指南。

3. StreamSQL基础

StreamSQL有三个核心的概念：**stream**、**streamjob** 和 **application**。概括地说，stream是数据流，streamjob是对一个或多个stream进行计算并将结果写进一张表的任务，application是一个或多个streamjob的集合。下面我们详细介绍这些概念。

3.1. Stream

Stream分为两种：**input stream** 和 **derived stream**。直接用于接收数据源传来的数据称为input stream；对已有stream进行变形得到的stream称为derived stream。

3.1.1. Input Stream

Input Stream用下面的语句创建：

```
CREATE STREAM <stream_name> (<column> <data_type>, <column> <data_type>, ...) ❶
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<delimiter>' ❷
TBLPROPERTIES (<source_information>) ❸
```

❶ 用SQL处理stream要求stream有表的schema，所以input stream的创建需要定义列名和列数据类型。

❷ <delimiter>给定消息中的文本分隔符，用来将消息分隔成列。

❸ <source_information>给出数据源的信息，用于配置消息的获取方式。

StreamSQL中，一个stream被分割成一系列 **小数据块**（**batch**） 来处理。Batch在获取数据时生成：在一段单位时间内获取的数据都放进一个batch中，这段单位时间称为 **batch interval**，它的长度（**batch duration**）可以在application中设置。

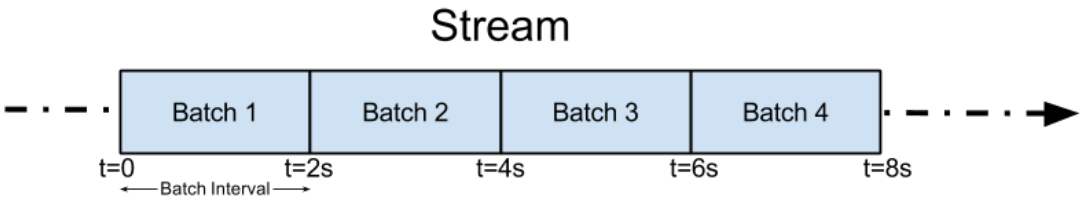


图 3.1. Stream由Batch组成

3.1.2. Derived Stream

Derived stream由CVAS（CREATE VIEW AS SELECT）语句对存在的stream变形（transform）得来。前面我们提到，stream是一系列batch，那么stream的变形实际上是对batch计算得到新batch的过程。Stream的变形分两种：单batch变形和多batch变形。单batch变形是对stream中单个 batch进行计算得到新batch的过程：

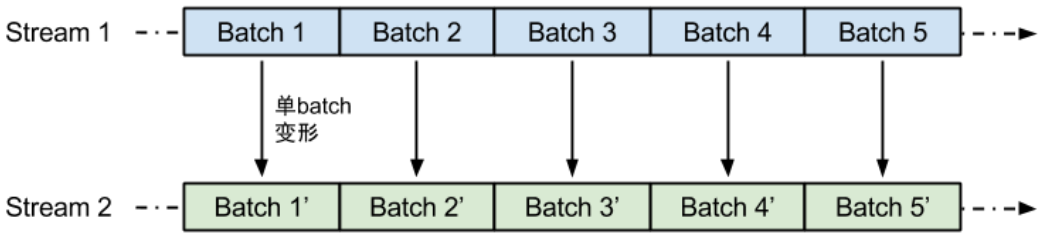


图 3.2. 单Batch变形

与之相对的多batch变形则是对一个 **时间窗口（window）** 内的多个batch计算得到新batch的过程，所以多batch变形常常又叫窗口变形，通过窗口变形得到的derived stream又称 **window stream** 。对一个stream进行窗口变形需要两个重要的参数：**length** 和 **slide**。 Length是窗口的持续时间，slide则是两个相邻窗口之间的间隔时间。Length和slide的长度都必须是batch duration的倍数。下图的窗口变形中，length是batch duration的3倍，slide是batch duration的2倍。

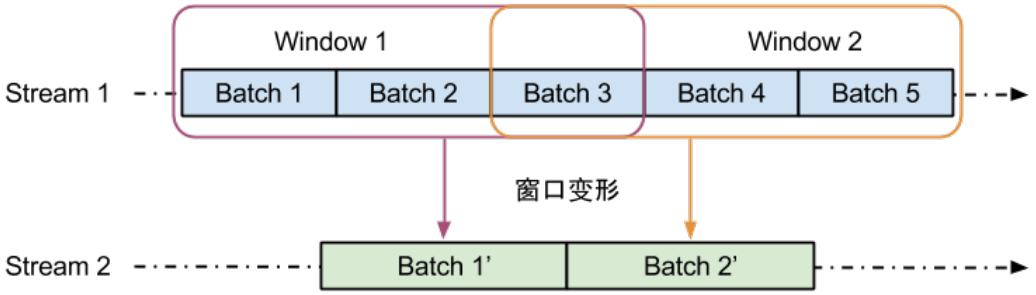


图 3.3. 窗口变形

3.2. Streamjob

StreamSQL中的stream是静态的——它们仅仅描述了如何对数据源传来的数据进行接收和变形的 **计划**，但并不执行这些计划。要让StreamSQL执行计划，需要通过将stream插入一张表来触发一个streamjob。一个streamjob启动时，StreamSQL会为每一个input stream启动一组称为 **receiver** 的任务来接收数据，接收来的数据经过一系列derived stream的变形最终被插入一张表，供用户查询：

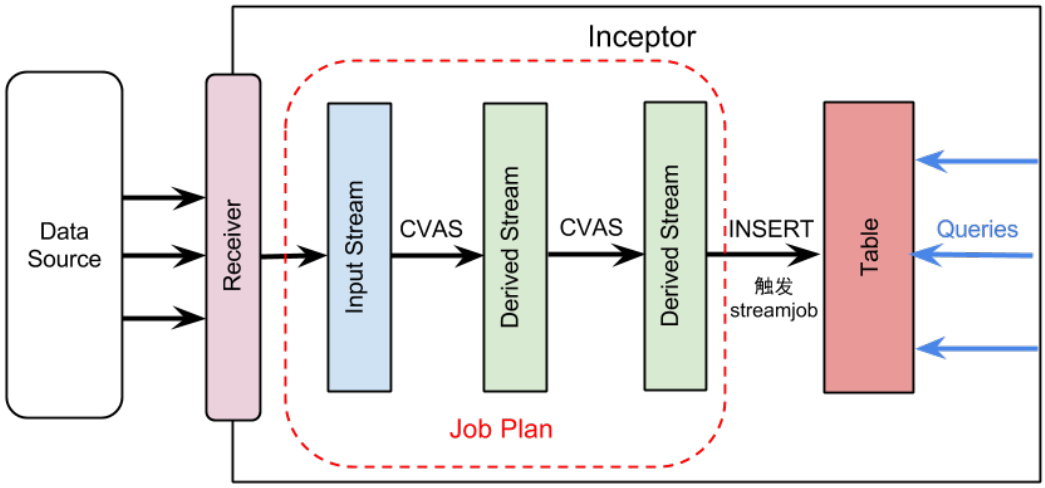


图 3.4. Streamjob

3.3. Application

Application是一组业务逻辑相关的streamjob的集合。合理地使用application划分streamjob可以实现资源的共享和隔离：

- 资源共享：application内使用同一个input stream的streamjob共享一组receiver。

  例如图 3.5 “Application实现的资源共享和隔离”中，Application 1中的Streamjob 2和Streamjob 3都使用Input Stream 2，它们共享Application 1中为Input Stream 2启动的Receiver。

- 资源隔离：不同application中的streamjob若使用同一个input stream，则每个application都为这个input stream启动一组receiver。

  例如图 3.5 “Application实现的资源共享和隔离”中，Application 1和Application 2中都有Streamjob使用Input Stream 2，Application 1和Application 2各为Input Stream 2启动一组Receiver。

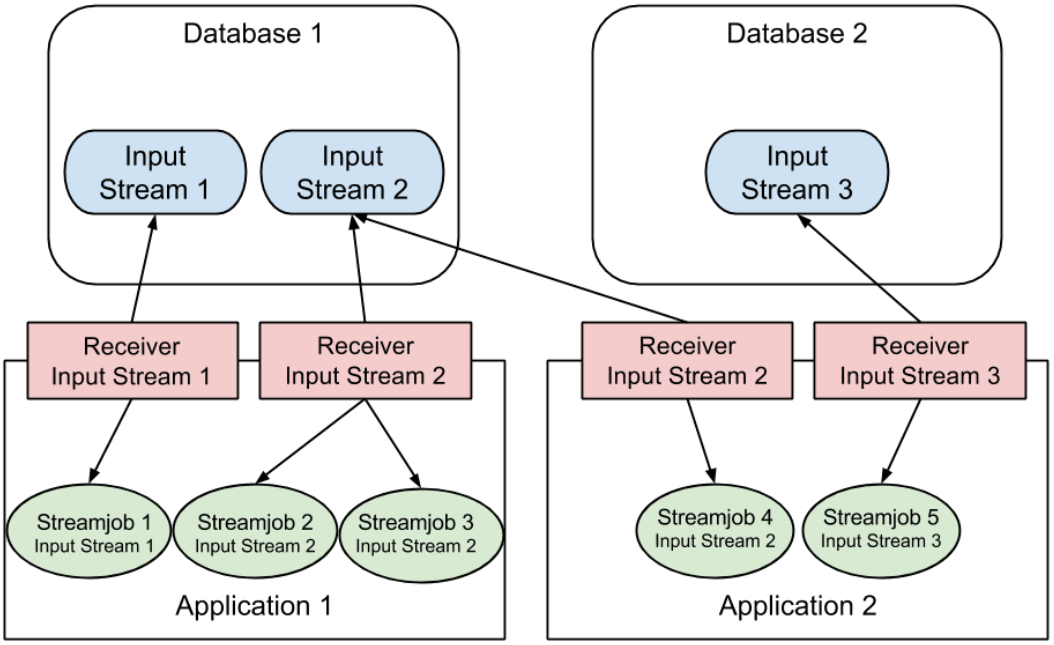


图 3.5. Application实现的资源共享和隔离

4. 将Apache Kafka作为数据源

StreamSQL支持Apache Kafka作为数据源，如果您需要了解更多关于Apache Kafka的概念和操作，请参考附录：Kafka基础。Apache Kafka作为StreamSQL数据源时，发布给Kafka topic的消息通过input stream进入StreamSQL。一个input stream只能接收一个topic中的消息，多个input stream可以指向同一个topic。

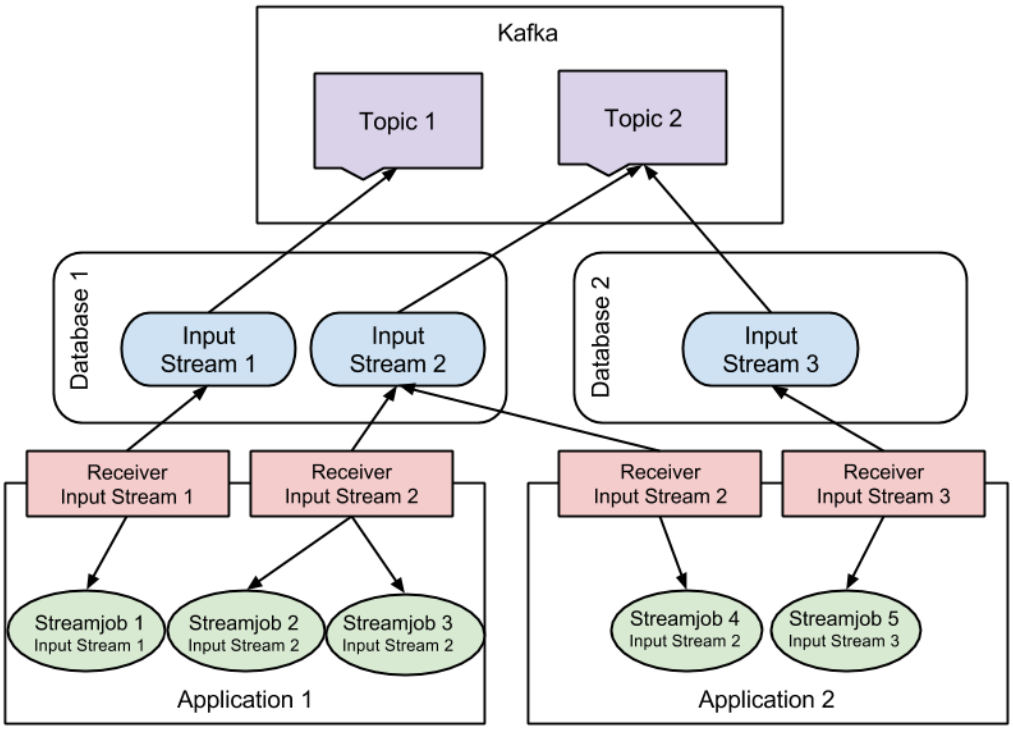


图 4.1. Apache Kafka as Source

创建input stream时，需要在 TBLPROPERTIES(<source_info>) 子句中定义数据源的信息。当使用Kafka作为数据源时，<source_info> 中需要包含下面信息：

```
"topic"="<topic_name>", ❶
"kafka.zookeeper"="<ip:port>", ❷
["kafka.broker.list"="<ip:port>"] ❸
```

❶ <topic_name>指定input stream指向的Kafka topic。

❷ <ip:port>指定TDH集群中的一个Zookeeper节点的ip和端口号，除非另外设置，Zookeeper的端口号为2181。

❸ 可选项。<ip:port>指定集群中的任意一个Kafka broker的ip和端口号，除非另外设置，Kafka broker的端口号为9092。如果提供这一选项，使用这个stream的streamjob启动时会为topic中的每一个partition启动一个receiver；不提供则是为topic中所有的partition启动一个receiver。

5. Stream管理

Stream管理的语法包括：

- CREATE STREAM：创建一个input stream
- CREATE VIEW：创建一个derived stream
- SHOW：列出所有的stream
- DESCRIBE：描述stream
- ALTER STREAM：修改input stream
- DROP STREAM：删除input stream
- DROP VIEW：删除derived stream

5.1. CREATE STREAM

CREATE STREAM用于建一个input stream。

语法：创建以Kafka为源的input stream

```
CREATE STREAM <stream_name>(<column> <data_type>, <column> <data_type>, ...)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<delimiter>' ❶
TBLPROPERTIES("topic"="<topic_name>", ❷
              "kafka.zookeeper"="<ip:port>", ❸
              ["kafka.broker.list"="<ip:port>"] ❹
              ["<key>"="<value>"], ❺
              ...);
```

❶ <delimiter>指定数据中的分隔符。

❷ <topic_name>指定input stream指向的Kafka topic。

❸ <ip:port>指定TDH集群中的一个Zookeeper节点的ip和端口号，除非另外设置，Zookeeper的端口号为2181。

❹ 可选项。<ip:port>指定集群中的任意一个Kafka broker的ip和端口号，除非另外设置，Kafka broker的端口号为9092。如果提供这一选项，使用这个stream的streamjob启动时会为topic中的每一个partition启动一个receiver；不提供则是为topic中所有的partition启动一个receiver。

❺ TBLPROPERTIES中还可以包含任意自定义的键值对用于存放用户自定义的流属性。

例1

```
CREATE STREAM employee_stream (id INT, name STRING) ROW FORMAT
 DELIMITED FIELDS TERMINATED BY ',' TBLPROPERTIES ("topic"="employees",
 "kafka.zookeeper"="172.16.1.128:2181");
```

例2

```
CREATE STREAM student_stream (id INT, name STRING) ROW FORMAT
 DELIMITED FIELDS TERMINATED BY ',' TBLPROPERTIES ("topic"="employees",
 "kafka.zookeeper"="172.16.1.128:2181", "kafka.broker.list"="172.16.1.127:9092");
```

5.2. CREATE VIEW

使用现有的stream和table可以创建derived stream。在StreamSQL中，derived stream使用CREATE VIEW AS SELECT创建。

语法

```
CREATE VIEW <view_name> AS SELECT <select_statement>;
```

举例

```
CREATE VIEW employee_name_view AS SELECT name FROM employee_stream;
```

语法：创建window stream

```
CREATE VIEW <view_name> AS SELECT <select_statement>
STREAMWINDOW <window_name> AS (length '<window_duration>' second
 slide '<sliding_duration>' second); ❶
```

❶ <window_duration>和<sliding_duration>应该都是batch interval的整数倍（见“length和slide”）。

举例

```
CREATE VIEW employee_window_view AS SELECT * FROM employee_stream STREAMWINDOW w1 AS
 (length '12' second slide '4' second);
```

5.3. SHOW

语法：显示所有input stream

```
SHOW STREAMS;
```

语法：显示所有的tables

```
SHOW TABLES;
```

因为derived stream由CREATE VIEW创建，而view需要用SHOW TABLES查看，所以SHOW TABLES可以查看到所有的derived stream，但也会列出其他表。

5.4. DESCRIBE

语法

```
DESCRIBE|DESC <stream_name>|<view_name> [EXTENDED|FORMATTED] ❶
```

❶ DESCRIBE可以简写为DESC；[EXTENDED|FORMATTED]为可选项，选择它们可以输出比单独DESCRIBE更多的信息。

5.5. ALTER STREAM

ALTER STREAM用于修改input stream。

语法： 重命名input stream

```
ALTER STREAM <old_name> RENAME TO <new_name>;
```

语法： 修改input stream属性

```
ALTER STREAM <stream_name> SET TBLPROPERTIES ("key"="value");
```

使用该指令可以修改任意流属性，包括指向的topic、使用的Zookeeper节点、broker list以及其它自定义的流属性。

举例： 修改input stream指向的topic

```
ALTER STREAM demo_stream SET TBLPROPERTIES ("topic"="demo1");
```

5.6. DROP STREAM

删除一个input stream（也就是以CREATE STREAM创建的stream）。

语法

```
DROP STREAM <stream_name>;
```

5.7. DROP VIEW

删除一个derived stream（也就是以CREATE VIEW创建的stream）。

语法

```
DROP VIEW <view_name>;
```

6. Streamjob管理

Streamjob管理的语法包括：

- 用INSERT触发streamjob
- LIST：列出streamjob
- STOP：停止streamjob

6.1. INSERT

将一个stream插入一个table的操作会触发一个streamjob的执行。

语法

```
INSERT INTO <table_name> SELECT * FROM <stream_name>; ❶
```

❶ <table_name>指定的table需要和<stream_name>指定的stream有完全相同的schema。

举例

```
INSERT INTO employee_table SELECT * FROM employee_stream;
```

6.2. LIST

语法：列出当前所有的streamjob：

```
LIST STREAMJOBS;
```

这条指令会列出当前所有的streamjob的id（每个streamjob都有一个独特的id）、触发它们的INSERT语句以及它们的状态，例如：

```
0: jdbc:hive2://localhost:10000/default> LIST streamjobs;
+----------------------------------+-------------------------------------------------------------+----------+
|             streamid             |                             sql                             |  status  |
+----------------------------------+-------------------------------------------------------------+----------+
| 1398b372722c4fb8b46f00aa5d2b914d | INSERT INTO employee_table SELECT * FROM employee_stream    | running  |
| 995ba940d5304758956ba83e5b6f2b30 | insert into student_table select * from student_stream      | running  |
+----------------------------------+-------------------------------------------------------------+----------+
```

语法：列出指定id的streamjob

```
LIST STREAMJOB <streamjob_id>;
```

这条指令会列出指定id的streamjob的Start Time（起始时间）、Duration（运行时间）和Batches Processed（处理的batch数量）。

举例

```
0: jdbc:hive2://localhost:10000/default> LIST streamjob 995ba940d5304758956ba83e5b6f2b30;
+--------------------+-----------------------------------+
|      property      |               value               |
+--------------------+-----------------------------------+
| Stream Id          |  995ba940d5304758956ba83e5b6f2b30 |
| Start Time         |  2016-01-27 02:54:03.793          |
| Duration           |  260 s                            |
| Batches Processed  |  0                                |
+--------------------+-----------------------------------+
```

6.3. STOP

语法：停止指定id的streamjob

```
STOP STREAMJOB <streamjob_id>;
```

语法：停止所有的streamjob

```
STOP STREAMJOB ALL;
```

7. Application管理

Application是一组由业务逻辑组织的相关streamjob的集合，使用不同的application可以实现资源隔离（例如不同application使用同一个stream时，各个application会使用各自的receiver去接收这个stream中的消息），同一个application内部则可以实现资源共享。在StreamSQL基础中有更多关于application概念的介绍，本章我们将重点介绍application的管理。StreamSQL中application的管理操作包括：

1. USE APPLICATION：使用一个application
2. Application内参数设置

7.1. USE APPLICATION

语法

```
USE APPLICATION <app_name>;
```

使用指定的application。执行该指令后的所有stream job都划分在<app_name>指定的application中运行。如果不指定application，则默认在default application中运行streamjob。

举例

```
USE APPLICATION myapp;
```

7.2. Application内参数设置

7.2.1. Batch Duration

stream.batch.duration.ms参数用于以毫秒（ms）为单位设置batch duration，默认值为2000。该参数只有在启动新的streamjob开始前设置才会这个streamjob生效，streamjob开始运行后修改无效。

设置方法

```
set stream.batch.duration.ms = <integer>;
```

举例

```
set stream.batch.duration.ms= 4000;
```

7.2.2. Receiver个数

stream.number.receivers参数用于设置stream读取数据时使用的receiver数量。创建stream时可以用kafka.broker.list设置receiver数量，stream.number.receivers的设置覆盖kafka.broker.list的设置。该参数只有在启动新的streamjob开始前设置才会这个streamjob生效，streamjob开始运行后修改无效。

设置方法

```
set stream.number.receivers = <integer>;
```

举例

```
set stream.number.receivers = 2;
```

附录 A. Kafka基础

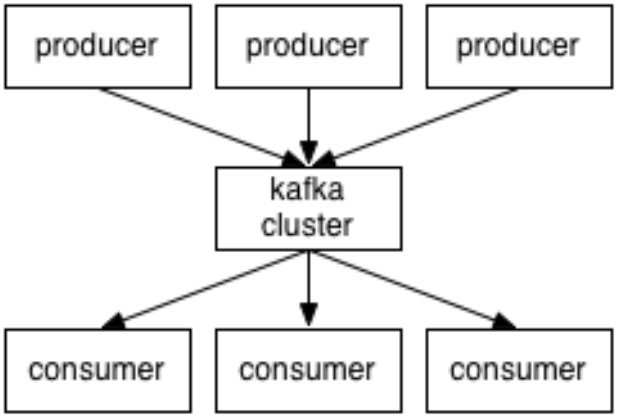
目前，Apache Kafka是StreamSQL唯一支持的数据源。本章将介绍Kafka的基础概念和基本操作，让您在学习使用StreamSQL时可以自己生成一些数据用于StreamSQL的使用。

A.1. 基础概念

Kafka是一个开源的分布式消息系统，Kafka中的核心概念有：

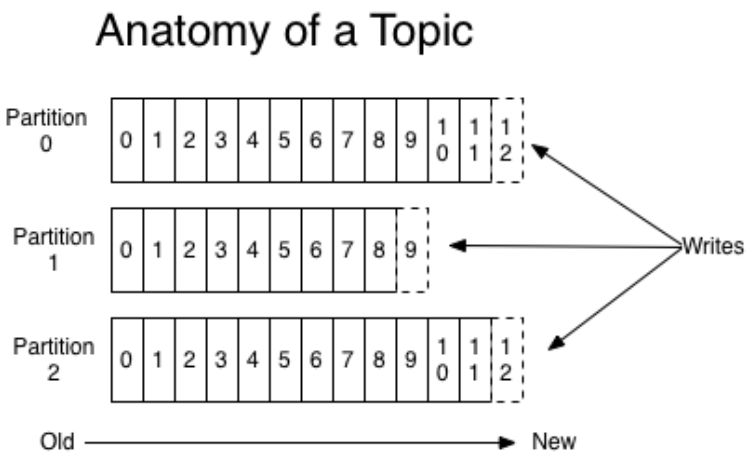
- Kafka用 **topic** 对消息（message）归类。例如，在网页活动跟踪中，每个活动种类（包括网页浏览、搜索、点击等）的消息都可以发布到一个各自的topic中。
- 向Topic发布消息的进程称为 **producer**。
- 向Topic订阅消息并处理已发布消息的进程称为 **consumer**。
- Kafka运行在一台或多台机器组成的集群上，Kafka集群中的节点称为 **broker**。

Kafka的工作流程大致如下：producer向topic发布消息，Kafka集群对发布的消息进行处理后传给consumer。



A.1.1. Topic和Producer

发布给一个topic的消息可以在Kafka集群中分割成多个 **partition**，每个parititon都是一个log，log中的消息按发布的顺序排成一个消息列 （**message sequence**），Kafka不断地向log的末尾写入新的消息。每个partition中的消息都有一个在partition中唯一的ID，称为 **offset**，也就是相对log开头的偏移量，Consumer在读取消息的时候会记住当前消息的offset。一个有三个partition的topic内部结构如下所示：



一个topic中的每个partition都可以分给Kafka集群中的一个broker来处理（一个broker可以处理多个partitions），这样的设计让Kafka可以方便地横向扩展，同时也提高了并行度。Kafka可以对每个partition进行备份（replicate），备份数量（replication factor）可以设置，将备份分散到集群中不同的broker上来实现HA。对每个partition来说都有一个作为 **leader** （领导者）的broker和零到多个 **follower** （跟随者）。Leader负责所有对partition的读写请求，follower则被动地复制leader的操作。如果leader失效，follower之一会自动成为新的leader。

Producer可以依据一些规则来选择将消息分发给topic中的哪个partition，常见的规则有轮流分发、按哈希值分发等等。

A.1.2. Consumer和Consumer Group

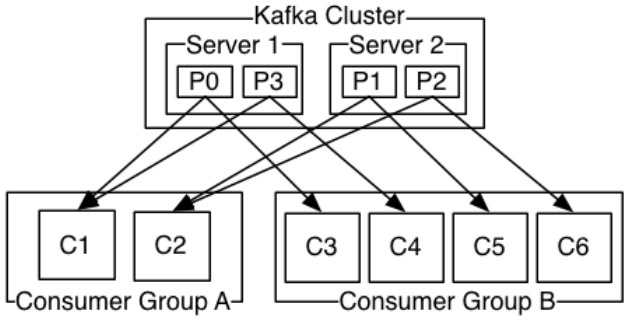
传统的消息分发模式一般有两种：queueing（队列）模式和publish-subscribe（发布订阅）模式。在queueing模式中，消息列中的消息会依次被consumer读取。在publish-subscribe模式中，每条消息会广播（broadcast）给所有的consumer，各个consumer根据预先定义好的方式订阅所有被广播的消息的一部分。Kafka使用 **Consumer Group** 实现了这两种模式的结合。

在Kafka中，一个或多个consumer组成consumer group。Consumer group表现为“逻辑的订阅者（logical subscriber）”——所有topic收到的消息都会广播给所有的consumer group，而每个consumer group选择订阅一个或多个topic中的消息，也就是说consumer group外部的消息分发模式是publish-subscribe模式。而同一个consumer group内部的consumer则遵循queueing模式。这个混合的消息分发模式有两个极限情况：

1. 所有consumer都在同一个consumer group中：这种情况是纯粹的queueing模式。
2. 每个consumer都在不同的consumer group中：这种情况是纯粹的publish-subscribe模式。

当同一个消息列中的消息被多个consumer读取时，会出现消息到达consumer的顺序和在消息列中的顺序不同的情况，所以在Kafka中，一个topic中的每个partition都只能被订阅它的各个consumer group中的 **一个** consumer读取（但是一个consumer可以读多个partition），以保证在各consumer group内，各partition的消息按发布的顺序被读取。因此，一个consumer group中的consumer个数不能多于它订阅的topic中的partition个数。在下图中，一个两节点的Kafka集群在分发四个partition的消息，每个节点负责两个partition的分发。Kafka集群将

消息分发给两个consumer group：Consumer Group A中有两个consumer，Consumer Group B中有四个consumer。



注意，Kafka只保持partition中的消息顺序不变，并不保证topic中的消息顺序不变。要保证topic中消息顺序不变，可以让topic只有一个partition，但是这样会牺牲并行度。

A.1.3. Replication和Failover

前面我们提到，Kafka对消息以partition为单位进行备份，一个备份称为一个replica。一个partition的replica总数称为replication factor，replication factor按topic设置。一个partition的所有replica中会有一个leader和零到若干个follower，所有的读写请求都由leader处理，follower像consumer一样接收leader上的消息，写进自己的log中，保持自己的log和leader的log一致（offset一致、消息一致、消息顺序一致）。

在Kafka中，一个replica和leader同步的标准有两个：

1. 和Zookeeper的session没有终止（通过Zookeeper的心跳机制）。
2. 如果这个replica是一个follower，它相对leader的延迟不能“太多”（可容忍的延迟可以设置）。

一个replica如果满足这两个标准，它则被称为一个 **in sync replica (ISR)**。Leader会记录ISR的列表，并将这个列表存在Zookeeper上，如果有replica失效或者延迟过多，则会被leader移出ISR列表。

只有当所有的ISR都接收到了一条消息并将这条消息写进自己的log中，这条消息才算被提交（committed），而broker只会将被提交的消息发给consumer，这就保证了只要有一个ISR还在工作，提交过的消息就不会丢失。当leader失效，ISR中的一个replica将被选举为新的leader。

对Kafka中一些概念的简单介绍到此结束。您可以通过开源社区的资料了解更多关于Kafka的细节。

A.2. Kafka相关脚本的简单使用

在生产中，通常需要使用Java来写producer和consumer用于发布和读取消息，以实现生产中的任务需求。在Transwarp Data Hub中，我们提供了一系列脚本让您可以通过命令行对Kafka进行一些简单的使用。

我们将以一个三节点的Kafka集群为例介绍如何使用我们提供的脚本进行下面操作：

1. 创建Topic
2. 查看Topic
3. 向Producer发送消息
4. 用Consumer接收消息
5. 进行kafka权限管理

一个Kafka集群中的各个broker各自用一个整数来互相区分，这个整数被称为broker id。集群上的broker id的查看方法为：

1. 运行/usr/lib/kafka/bin目录下的zookeeper-shell.sh脚本，进入Zookeeper Shell：

```
./zookeeper-shell.sh <zookeeper:port>
```

注意这里要提供一个Zookeeper节点以及端口号，默认端口号为2181。

2. 进入Zookeeper Shell后执行：

```
ls /brokers/ids
```

您会看到一列整数，这些整数就是您Kafka集群中broker的broker id。

我们的示例集群中的broker id为4, 5, 6。

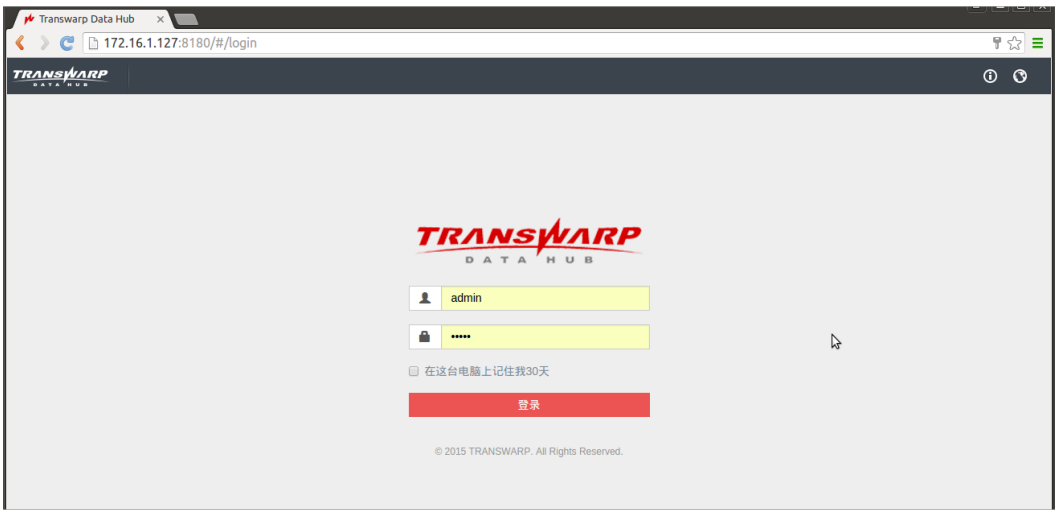
3. 执行 `quit` 退出Zookeeper Shell。

本节所使用脚本在您TDH集群上任意一个节点的/usr/lib/kafka/bin目录下。

A.2.1. 启动Zookeeper和Kafka服务

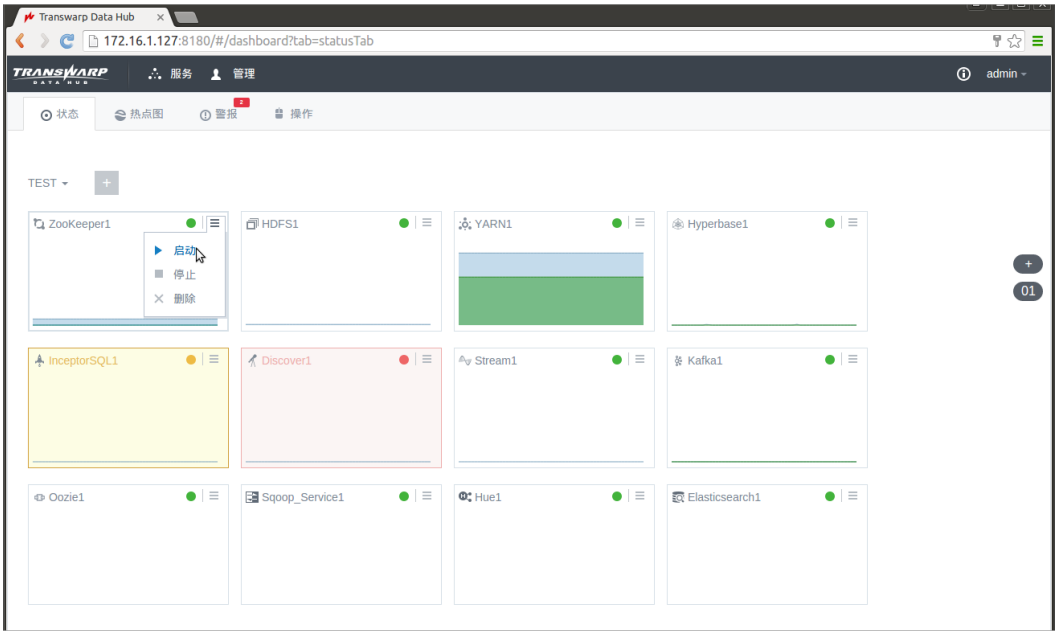
在开始使用Kafka前，您需要保证您集群上的Kafka服务正在运行。由于Kafka的运行依赖于Zookeeper，运行Kafka前要先开始Zookeeper服务。您可以通过Transwarp Data Hub的管理界面Transwarp Manager来查看和启动Kafka和Zookeeper服务：

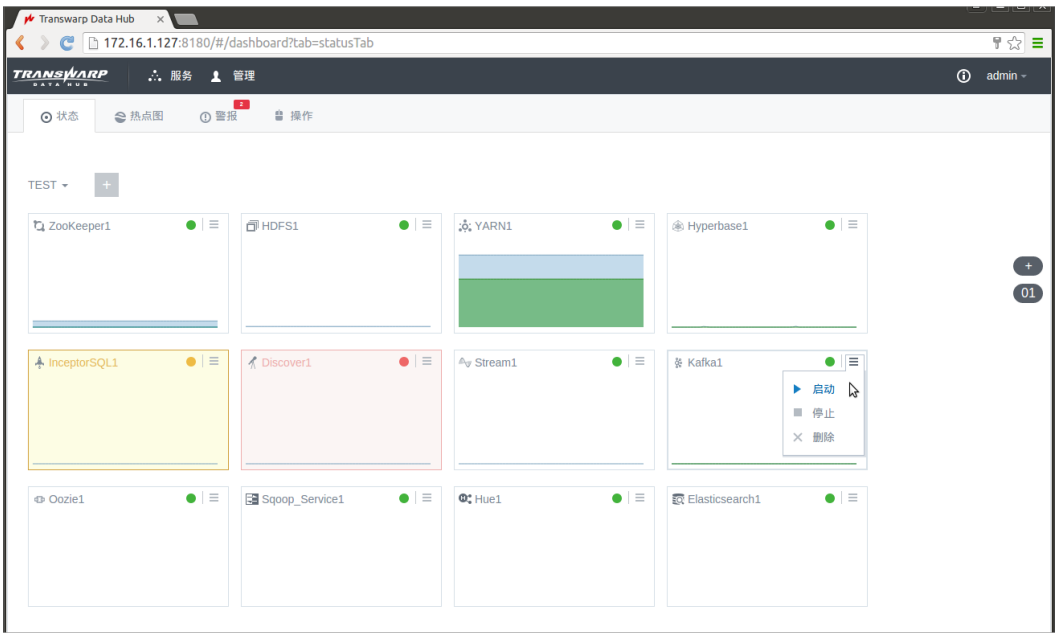
1. 打开一个网页浏览器访问http://<manager_node_ip>:8180。这里<manager_node_ip>处您需要提供您TDH集群的管理节点的IP：



在这个页面输入管理员用户名和密码登陆。

2. 在Transwarp Manager的首页您能看到集群中的各个服务的状态。如果服务显示为绿色，则说明服务在正常运行。将鼠标移到服务的右上角可以选择启动、停止或者删除服务：





3. 在确保Kafka和Zookeeper服务都在正常运行后，您可以命令行登陆到集群中任意一台节点上，进入/usr/lib/kafka/bin目录，这个目录中有将要使用的脚本。

A.2.2. 创建Topic

用于创建topic的脚本为kafka-create-topic.sh。直接运行脚本可以查看参数以及它们的说明。

必选参数

- `--broker`：指定使用的Kafka broker

  用法

  ```
  --broker <host:port>
  ```

- `--topic`：指定topic名称。

  用法

  ```
  --topic <topic_name>
  ```

- `--zookeeper`：指定使用的Zookeeper节点。

  用法

  ```
  --zookeeper <host:port>
  ```

  host处可以提供这个topic使用的Zookeeper节点的IP或hostname；port处提供zookeeper的端口号，默认值为2181。您可以提供多个Zookeeper节点来防止单个Zookeeper节点失效带来的问题。

可选参数

- `--principal` 和 `--keytab`：（在 **安全模式下必须使用**）：指定请求Kafka服务的用户的Kerberos principal和keytab路径。

  用法

  ```
  --principal <principal> --keytab <keytab_path>
  ```

- `--partition`：指定partition个数，默认值为1。

  用法

  ```
  --partition <Integer>
  ```

- `--replica`：设置该topic中各个partition的备份数（replication factor）。

  用法

  ```
  --replica <Integer>
  ```

- `--replica-assignment-list`：手动设置分配各broker处理的replica。

  用法

  ```
  --replica-assignment-list
    <broker_id_for_part1_replica1 :
    broker_id_for_part1_replica2,
    broker_id_for_part2_replica1 :
    broker_id_for_part2_replica2, ...> ❶
  ```

  ❶ 这里依次输入处理各个partition下各replica的broker id。处理同一个partition的不同replica的broker id用冒号（“:”）分隔；处理不同partition的broker id用逗号（“,”）分隔。

  举例：

  2个partition，每个partition各2个replica。第一个partition的两个replica分别分配给broker id为4和5的broker；第二个partition的两个replica分别分配给broker id为5和6的broker。

  ```
  --replica-assignment-list 4:5,5:6
  ```

使用示例

例1：

```
# ./kafka-create-topic.sh --topic demo1 --broker tw-node127:9092  --zookeeper tw-node127
```

例2：

```
# ./kafka-create-topic.sh --topic demo2 --broker tw-node127:9092 --zookeeper tw-
node127 tw-node128 --partition 2 --replica 2 --replica-assignment-list 4:5,5:6
```

例3：安全模式下的操作

```
# ./kafka-create-topic.sh --zookeeper tw-node127,tw-node128 --topic demo4 --broker
 tw-node127:9092 --principal alice --keytab /tmp/alice.keytab
```

注意，要执行此操作，alice用户必须有全局C权限，权限管理细节请见第 A.2.6 节 “Kafka权限管理”。

A.2.3. 查看Topic

用于查看topic的脚本为kafka-list-topic.sh。直接运行脚本可以查看参数以及它们的说明。

必选参数

- `--zookeeper`：指定使用的Zookeeper节点。

  用法

  ```
  --zookeeper <host:port>
  ```

  host处提供这个topic使用的Zookeeper节点的IP或hostname；port处提供Zookeeper的端口号，默认值为2181。您可以提供多个Zookeeper节点来防止单个Zookeeper节点失效带来的问题。

可选参数

- `--topic`：指定显示的topic名称，如果不指定则显示所有的topic。

  用法

  ```
  --topic <topic_name>
  ```

- `--unavailable-partitions`：如果加上这个参数，则只显示leader失效的partition。
- `--under-replicated-partitions`：如果加上这个参数，则只显示replica个数少于设置的个数的partition。

使用示例

例1

```
# ./kafka-list-topic.sh --zookeeper tw-node127
topic: demo1 partition: 0 leader: 6 replicas: 6     isr: 6
topic: demo2 partition: 0 leader: 4 replicas: 4,5 isr: 4,5
topic: demo2 partition: 1 leader: 5 replicas: 5,6 isr: 5,6
```

例2

```
# ./kafka-list-topic.sh --zookeeper tw-node127 --topic demo1
topic: demo1 partition: 0 leader: 6 replicas: 6     isr: 6
```

例3

假如集群中broker.id=4的broker失效：

```
# ./kafka-list-topic.sh --zookeeper tw-node127 --unavailable-partitions
topic: demo partition: 2 leader: -1 replicas: 4 isr:

# ./kafka-list-topic.sh --zookeeper tw-node127 --under-replicated-partitions
topic: demo     partition: 2 leader: -1 replicas: 4 isr:
topic: demo2 partition: 0 leader: 5 replicas: 4,5 isr: 5
```

A.2.4. 用Producer发送消息

用producer发送消息所使用的脚本为kafka-console-producer.sh。直接运行脚本可以查看参数以及它们的说明。

必选参数

- `--broker-list`：提供至少一个集群中正在运行的broker。

  用法

  ```
  --broker-list <host1:port, host2:port, ...>
  ```

  host处提供broker的HOSTNAME或者IP；port默认为9092。
- `--topic`：producer发布消息的目标topic。

  用法

  ```
  --topic <topic>
  ```

可选参数

- `--principal` 和 `--keytab`：（在 **安全模式下必须使用**）：指定请求Kafka服务的用户的Kerberos principal和keytab路径。

  用法

  ```
  --principal <principal> --keytab <keytab_path>
  ```

使用示例

执行下面指令启动向demo1发送消息的producer，并发送两条消息Hello和World。

```
./kafka-console-producer.sh --broker-list tw-node127:9092 --topic demo1
Hello
World
```

在安全模式下进行相同操作：

```
./kafka-console-producer.sh --broker-list tw-node127:9092 --topic demo1 --principal
 alice --keytab /tmp/alice.keytab
Hello
World
```

注意，要执行此操作，alice必须对demo1有W权限，权限管理细节请见第 A.2.6 节 “Kafka权限管理”。

A.2.5. 用Consumer接收消息

用于启动consumer接收消息的脚本为kafka-console-consumer.sh。直接运行脚本可以查看参数以及它们的说明。

必选参数

- `--topic`：源topic的名称。

  用法

  ```
  --topic <topic>
  ```

- `--zookeeper`：使用的Zookeeper节点和端口号。

  用法

  ```
  --zookeeper <zookeeper:port>
  ```

  Zookeeper的默认端口号为2181。

可选参数

- `--principal` 和 `--keytab`：（在 **安全模式下必须使用**）：指定请求Kafka服务的用户的Kerberos principal和keytab路径。

  用法

  ```
  --principal <principal> --keytab <keytab_path>
  ```

使用示例

让上一节中的producer保持运行，打开另外一个窗口登陆集群，运行kafka-console-consumer.sh脚本：

```
# ./kafka-console-consumer.sh --topic demo1 --zookeeper tw-node127:2181
```

在安全模式下进行相同操作：

```
# ./kafka-console-consumer.sh --topic demo1 --zookeeper tw-node127 --principal alice
 --keytab /tmp/alice.keytab
```

注意，要执行此操作，alice必须对demo1有R权限，权限管理细节请见第 A.2.6 节 “Kafka权限管理”。

在运行着的producer中输入一些信息，这些信息会显示在consumer正在运行的窗口中。

A.2.6. Kafka权限管理

安全模式下，用户执行Kafka操作需要有对应操作的权限。一个用户在Kafka中有下面四种权限：

- R(ead)：从Topic读取数据
- W(rite)：向Topic生产数据
- C(reate)：创建Topic
- D(elete)：删除Topic

只有kafka用户可以进行权限管理操作，包括赋予权限、收回权限和查看权限。下面将提到的授权操作脚本都在/usr/lib/kafka/bin下。

授予权限

语法：

```
./kafka-grant-permission.sh --permission <privileges> [--topic <topic>] --user
 <user_name> --zookeeper <zookeeper_hosts>  --principal <kafka_principal> --keytab
 <kafka_keytab_path>
```

说明

- `<privileges>` 处填授予的权限，也就是 `RWCD` 中的一个或多个。
- `--topic` 为可选项，用于指定权限生效的Topic，如果不指定，则代表授予全局权限。
- `<user_name>` 出填被授予权限的用户。
- `<zookeeper_hosts>` 处填使用的ZooKeeper节点的hostname。
- `<kafka_principal>` 和 `<kafka_keytab_path>` 处分别填当前节点上kafka用户的principal和keytab的路径。

举例

授予全局权限

```
./kafka-grant-permission.sh --permission C --user alice --zookeeper tw-node128 --
principal kafka/tw-node128@TDH --keytab /etc/kafka1/kafka.keytab
```

授予Topic权限

```
./kafka-grant-permission.sh --permission RWD --topic demo3 --user alice --zookeeper
 tw-node128 --principal kafka/tw-node128@TDH --keytab /etc/kafka1/kafka.keytab
```

查看权限

语法

```
./kafka-show-permission.sh --user <user_name> [--topic <topic>] --zookeeper
 <zookeeper_hosts>  --principal <kafka_principal> --keytab <kafka_keytab_path>
```

- `--topic` 为可选项，用于指定查看权限的Topic，如果不指定，则代表查看所有Topic以及全局权限。
- `<user_name>` 出填被授予权限的用户。
- `<zookeeper_hosts>` 处填使用的ZooKeeper节点的hostname。
- `<kafka_principal>` 和 `<kafka_keytab_path>` 处分别填当前节点上kafka用户的principal和keytab的路径。

举例

查看所有Topic以及全局权限

```
./kafka-show-permission.sh --user alice --zookeeper tw-node128 --principal kafka/tw-
node128@TDH --keytab /etc/kafka1/kafka.keytab
```

查看指定Topic的权限

```
./kafka-show-permission.sh --user alice --topic demo3 --zookeeper tw-node128 --
principal kafka/tw-node128@TDH --keytab /etc/kafka1/kafka.keytab
```

收回权限

语法

```
./kafka-revoke-permission.sh --permission <privileges> [--topic <topic>] --user
 <user_name> --zookeeper <zookeeper_hosts> --principal <kafka_principal> --keytab
 <kafka_keytab_path>
```

说明

- `<privileges>` 处填收回的权限，也就是 `RWCD` 中的一个或多个。
- `--topic` 为可选项，用于指定收回权限的Topic，如果不指定，则代表收回全局权限。
- `<user_name>` 出填被收回权限的用户。

- <zookeeper_hosts> 处填使用的ZooKeeper节点的hostname。
- <kafka_principal> 和 <kafka_keytab_path> 处分别填当前节点上kafka用户的principal和keytab的路径。

举例：

收回指定Topic上的权限

```
./kafka-revoke-permission.sh --permission W --topic demo3 --user alice --zookeeper
 tw-node128 --principal kafka/tw-node128@TDH --keytab /etc/kafka1/kafka.keytab
```

收回全局权限

```
./kafka-revoke-permission.sh --permission C --user alice --zookeeper tw-node128 --
principal kafka/tw-node128@TDH --keytab /etc/kafka1/kafka.keytab
```

客户服务

技术支持

感谢你使用星环信息科技（上海）有限公司的产品和服务。如您在产品使用或服务中有任何技术问题，可以通过以下途径找到我们的技术人员给予解答。

Email: support@transwarp.io

技术支持热线电话：4008 079 976

技术支持QQ专线：3221723229, 3344341586

官方网址：www.transwarp.io

意见反馈

如果你在系统安装，配置和使用中发现任何产品问题，可以通过以下方式反馈：

Email: support@transwarp.io

感谢你的支持和反馈，我们一直在努力！

关于我们：

星环信息科技（上海）有限公司是一家大数据领域的高科技公司，致力于大数据基础软件的研发。星环科技目前掌握的企业级Hadoop和Spark核心技术在国内独树一帜，其产品Transwarp Data Hub (TDH)的整体架构及功能特性堪比硅谷同行，在业界居于领先水平，性能大幅领先Apache Hadoop，可处理从GB到PB级别的数据。星环科技的核心开发团队参与部署了国内最早的Hadoop集群，并在中国的电信、金融、交通、政府等领域的落地应用拥有丰富经验，是中国大数据核心技术企业化应用的开拓者和实践者。星环科技同时提供存储、分析和挖掘大数据的高效数据平台 和服务，立志成为国内外领先的大数据核心技术厂商。

行业地位：

来自知名外企的创业团队，成功完成近千万美元的A轮融资，经验丰富的企业级Hadoop发行版开发团队，国内最多落地案例。

核心技术：

高性能、完善的SQL on Hadoop、R语言的并行化支持，为企业数据分析与挖掘提供优秀选择。

应用案例：

已成功部署多个关键行业领域，包括电信、电力、智能交通、工商管理、税务、金融、广电、电商、物流等。

地址：上海市徐汇区桂平路481号18幢3层301室（漕河泾新兴技术开发区）
邮编：200233
电话：4008-079-976
网址：www.transwarp.io