### mysql 创建表

[hadoop@h201 ~]$ mysql -h 192.168.8.201 -u sq -p

mysql> use flink;

mysql> create table t2(id int,name varchar(50));  
啊

和source差不多，这里就不沾代码了，主要区别就是。run换成了invoke,这个方法在每条数据进来的时候都会调用，把写下游系统的逻辑写到这里面就行了。

### 开发 通过SinkFunction：

**package** com.mysql;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.java.tuple.Tuple2;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.datastream.DataStreamSource;  
**import** org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.streaming.api.functions.sink.SinkFunction;  
  
**import** java.sql.\*;  
  
  
**public class** mm {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 DataStream<String> s1 = env.socketTextStream(**"192.168.8.66"**, 9999);  
 DataStream<Tuple2<Integer, String>> m1 = s1.map(**new** MapFunction<String, Tuple2<Integer, String>>() {  
 @Override  
 **public** Tuple2<Integer, String> map(String s) **throws** Exception {  
 String[] words = s.split(**" "**);  
 **return new** Tuple2<Integer, String>(Integer.*parseInt*(words[0]), words[1]);  
 }  
 });  
 m1.addSink(**new** ss2());  
 env.execute(**"aaa"**);  
 }  
 **public static class** ss2 **implements** SinkFunction<Tuple2<Integer,String>>{  
 **private static final** String ***URL***=**"jdbc:mysql://192.168.8.201:3306/flink"**;  
 **private static final** String ***USER***=**"sq"**;  
 **private static final** String ***PASSWORD***=**"sq123"**;  
 @Override  
 **public void** invoke(Tuple2<Integer,String> value) **throws** Exception {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 Connection conn = DriverManager.*getConnection*(***URL***, ***USER***, ***PASSWORD***);  
 PreparedStatement pstmt = conn.prepareStatement(**"INSERT INTO t2(ID,NAME ) VALUES (?,?);"**);  
 pstmt.setInt(1,value.**f0**);  
 pstmt.setString(2,value.**f1**);  
 pstmt.executeUpdate();  
 **if** (pstmt != **null**) {  
 pstmt.close();  
 }  
 **if** (conn != **null**) {  
 conn.close();  
 }  
  
 }  
 }  
}

这样实现有个问题，每一条数据，都要打开mysql连接，再关闭，比较耗时，这个可以使用flink中比较好的Rich方式来实现

=======================

### 开发 通过 RichSinkFunction

**package** com.mysql;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.java.tuple.Tuple2;  
**import** org.apache.flink.configuration.Configuration;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.streaming.api.functions.sink.RichSinkFunction;  
  
**import** java.sql.Connection;  
**import** java.sql.DriverManager;  
**import** java.sql.PreparedStatement;  
  
**public class** mm2 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 DataStream<String> s1 = env.socketTextStream(**"192.168.8.66"**, 9999);  
 DataStream<Tuple2<Integer, String>> m1 = s1.map(**new** MapFunction<String, Tuple2<Integer, String>>() {  
 @Override  
 **public** Tuple2<Integer, String> map(String s) **throws** Exception {  
 String[] words = s.split(**" "**);  
 **return new** Tuple2<Integer, String>(Integer.*parseInt*(words[0]), words[1]);  
 }  
 });  
 m1.addSink(**new** ss5());  
 env.execute(**"aasa"**);  
 }  
 **public static class** ss5 **extends** RichSinkFunction<Tuple2<Integer,String>>{  
 **private static final** String ***URL***=**"jdbc:mysql://192.168.8.201:3306/flink"**;  
 **private static final** String ***USER***=**"sq"**;  
 **private static final** String ***PASSWORD***=**"sq123"**;  
 **private** Connection **conn** = **null**;  
 **private** PreparedStatement **pstmt** = **null**;  
 @Override  
 **public void** invoke(Tuple2<Integer,String> value) **throws** Exception {  
  
 **pstmt** = **conn**.prepareStatement(**"INSERT INTO t2(ID,NAME ) VALUES (?,?);"**);  
 **pstmt**.setInt(1,value.**f0**);  
 **pstmt**.setString(2,value.**f1**);  
 **pstmt**.executeUpdate();  
 }  
 @Override  
 **public void** open(Configuration parameters) **throws** Exception{  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 **conn** = DriverManager.*getConnection*(***URL***, ***USER***, ***PASSWORD***);  
 }  
 @Override  
 **public void** close() **throws** Exception{  
 **if** (**pstmt** != **null**) {  
 **pstmt**.close();  
 }  
 **if** (**conn** != **null**) {  
 **conn**.close();  
 }  
 }  
 }  
}

Rich方式的优点在于，有个open和close方法，在初始化的时候建立一次连接，之后一直使用这个连接即可，缩短建立和关闭连接的时间，也可以使用连接池实现，这里只是提供这样一种思路。