广西民族师范学院课程考核试卷

（2025 — 2026 学年度第 2 学期）

《Flink 原理与应用》 课程 (A ☑ /B ☐卷)

课程编号		考核形式	考查
考试班级	网络工程 233 班、234 班		
考核日期	20 26 年 月 日	考核时长	120 分钟
命题教师签名		**教研室主任签名**	
主管学院领导签名			

题 号	一	二	三	四	五	六	七	八	九	十	总分
分 值	10	30	20	20	20						100
实得分											
统分人						核分人					

评卷教师 姓名 学号 专业班级 年级 学院

装 订 线 内 不 准 答 题

一、项目情景

假如你进入 XX 证券公司实习，该公司正在研发一套证券数据分析系统，要求具备基于数据流的动态分析能力，并且支持大数据处理。请使用 Flink 框架，开发实现以下功能，详细见评分标准。

```
//股票交易数据结构
public class StockPrice {
    public String symbol; //股票名称
    public double price;  //交易价格
    public long ts;      //交易时间
    public int volume;  //交易数量（成交量）
    public String mediaStatus; //媒体评价信息
```

```
};

//媒体评价数据结构
public class Media {
    public String symbol;//股票名称
    public long ts;//媒体评价时间
    public String status;//媒体评价信息（或评价类别）
};
```

二、评分标准：

(1) 自定义数据源:

（a） 数据源 1：创建媒体评价数据源，每秒发一条随机评价信息 （5 分）

（b） 数据源 2：股票实时交易数据源，读取历史交易数据回放（5 分）

(2) 成交量分析：统计每 10 秒的窗口范围内的每个股票的成交量，迟到数据侧流输出（在数据源端每 20 条数据制造一条迟到数据）（30 分）

(3) 媒体评价数据应用：使用双数据流处理技术，给每条交易数据，填充该股票的最新媒体评价信息（20 分）

(4) 股票价格分析：用状态变量统计，每种股票在每个成交价格位置的累计成交量（20 分）

(5) FlinkSQL 应用：用 SQL 方式统计每个股票的平均交易价格。(20 分)

提示：平均交易价格＝交易金额（价格*成交量）/总成交量

三、关键源代码（和运行成果截图）

(1) 自定义数据源

说明：以下代码以 Flink 1.17.x DataStream API 为基准。历史数据用固定随机种子模拟回放，便于直接运行验证。

1. 公共数据结构、两个自定义数据源与主程序

```
// Maven 依赖建议：Flink 1.17.x，flink-streaming-java、flink-clients、
flink-table-api-java-bridge、flink-table-planner-loader
import org.apache.flink.api.common.eventtime.WatermarkStrategy;
import org.apache.flink.api.common.state.*;
```

```
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.datastream.*;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.co.BroadcastProcessFunction;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;
import org.apache.flink.util.Collector;
import java.time.Duration;
import java.util.*;

public class StockAnalysisJob {
  public static class StockPrice {
    public String symbol; public double price; public long ts;
    public int volume; public String mediaStatus;
    public StockPrice() {}
    public StockPrice(String s,double p,long t,int v){symbol=s;price=p;ts=t;volume=v;}
    public StockPrice copy(){StockPrice x=new
StockPrice(symbol,price,ts,volume);x.mediaStatus=mediaStatus;return x;}
    public String toString(){return symbol+", price="+price+", volume="+volume+",
media="+mediaStatus;}
  }
  public static class Media {
    public String symbol; public long ts; public String status;
    public Media() {}
    public Media(String s,long t,String v){symbol=s;ts=t;status=v;}
    public String toString(){return symbol+","+status+","+ts;}
  }

  // 数据源 1：每秒产生一条随机媒体评价。
  public static class MediaSource extends RichParallelSourceFunction<Media> {
    private volatile boolean running=true;
    private final String[] symbols={"AAPL","MSFT","TSLA"};
    private final String[] statuses={"利好","中性","利空"};
    public void run(SourceContext<Media> ctx) throws Exception {
      Random r=new Random();
      while(running){
        synchronized(ctx.getCheckpointLock()){
          ctx.collect(new Media(symbols[r.nextInt(symbols.length)],
              System.currentTimeMillis(),statuses[r.nextInt(statuses.length)]));
        }
```

```
      Thread.sleep(1000);
    }
  }
  public void cancel(){running=false;}
 }

 // 数据源 2：模拟读取历史交易记录并按时间顺序回放；每 20 条制造一条迟到 15 秒的数据。
 public static class StockReplaySource extends RichParallelSourceFunction<StockPrice> {
  private volatile boolean running=true;
  private final String[] symbols={"AAPL","MSFT","TSLA"};
  public void run(SourceContext<StockPrice> ctx) throws Exception {
    Random r=new Random(2026); int n=0;
    while(running){
      n++; long now=System.currentTimeMillis();
      long eventTs=(n%20==0)?now-15000:now;
      String s=symbols[r.nextInt(symbols.length)];
      double price=Math.round((80+r.nextDouble()*120)*100.0)/100.0;
      int volume=100+r.nextInt(901);
      synchronized(ctx.getCheckpointLock()){ctx.collect(new
StockPrice(s,price,eventTs,volume));}
      Thread.sleep(300);
    }
  }
  public void cancel(){running=false;}
 }

 public static void main(String[] args) throws Exception {
  StreamExecutionEnvironment env=StreamExecutionEnvironment.getExecutionEnvironment();
  env.setParallelism(1);
  org.apache.flink.table.api.bridge.java.StreamTableEnvironment tEnv=
    org.apache.flink.table.api.bridge.java.StreamTableEnvironment.create(env);
  DataStream<Media> media=env.addSource(new MediaSource()).name("media-source");
  DataStream<StockPrice> trades=env.addSource(new StockReplaySource())
    .assignTimestampsAndWatermarks(
      WatermarkStrategy.<StockPrice>forBoundedOutOfOrderness(Duration.ofSeconds(2))
        .withTimestampAssigner((x,recordTs)->x.ts))
    .name("stock-replay-source");

  volumeAnalysis(trades);
```

```
    enrichWithLatestMedia(trades,media);
    cumulativeVolumeAtPrice(trades);
    sqlWeightedAverage(trades,tEnv);
    env.execute("Securities Streaming Analysis");
  }
```

(2) 成交量分析（迟到数据处理）

2. 10 秒事件时间窗口、成交量聚合与迟到侧输出

```
  // 10 秒滚动事件时间窗口；超过水位线的记录送到侧输出。
  public static void volumeAnalysis(DataStream<StockPrice> trades){
    org.apache.flink.util.OutputTag<StockPrice> lateTag=
      new org.apache.flink.util.OutputTag<StockPrice>("late-trades"){};
    SingleOutputStreamOperator<String> totals=trades
      .keyBy(x->x.symbol)
      .window(org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows
        .of(org.apache.flink.streaming.api.windowing.time.Time.seconds(10)))
      .sideOutputLateData(lateTag)
      .aggregate(new
org.apache.flink.api.common.functions.AggregateFunction<StockPrice,Long,Long>(){
        public Long createAccumulator(){return 0L;}
        public Long add(StockPrice x,Long acc){return acc+x.volume;}
        public Long getResult(Long acc){return acc;}
        public Long merge(Long a,Long b){return a+b;}
      },new
org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction<Long,String,Stri
ng,
          org.apache.flink.streaming.api.windowing.windows.TimeWindow>(){
        public void process(String symbol,Context c,Iterable<Long> values,Collector<String>
out){
          out.collect("WINDOW symbol="+symbol+", start="+c.window().getStart()+
              ", end="+c.window().getEnd()+", totalVolume="+values.iterator().next());
        }
      });
    totals.print("volume-window");
    totals.getSideOutput(lateTag).print("late-trade");
  }
```

运行结果示例：volume-window> WINDOW symbol=AAPL, ..., totalVolume=6842；late-trade> AAPL, price=..., volume=...。每第 20 条记录的事件时间回拨 15 秒，通常会在水位线越过对应窗口后进入侧输出。

(3) 媒体评价数据应用（双数据流合并）

3. 广播状态保存每只股票的最新媒体评价

```
  private static final MapStateDescriptor<String,Media> MEDIA_STATE=
    new MapStateDescriptor<>("latest-media",String.class,Media.class);

  // 把媒体流广播；交易流读取该股票最新评价并填充 mediaStatus。
  public static void enrichWithLatestMedia(DataStream<StockPrice> trades,DataStream<Media>
media){
    BroadcastStream<Media> broadcast=media.broadcast(MEDIA_STATE);
    trades.connect(broadcast).process(new
BroadcastProcessFunction<StockPrice,Media,StockPrice>(){
      public void processBroadcastElement(Media m,Context ctx,Collector<StockPrice>
out)throws Exception{
        Media old=ctx.getBroadcastState(MEDIA_STATE).get(m.symbol);
        if(old==null || m.ts>=old.ts) ctx.getBroadcastState(MEDIA_STATE).put(m.symbol,m);
      }
      public void processElement(StockPrice x,ReadOnlyContext ctx,Collector<StockPrice>
out)throws Exception{
        Media m=ctx.getBroadcastState(MEDIA_STATE).get(x.symbol);
        StockPrice y=x.copy(); y.mediaStatus=(m==null?"暂无评价":m.status); out.collect(y);
      }
    }).print("enriched-trade");
  }
```

运行结果示例：enriched-trade> AAPL, price=126.35, volume=520, media=利好。若交易先于该股票第一条媒体数据到达，则 mediaStatus 为“暂无评价”。

(4) 股票价格分析（带状态的计算）

4. MapState 统计每只股票在各成交价位的累计成交量

```
  // 按股票分区，用 MapState 保存“价格 -> 累计成交量”。
  public static void cumulativeVolumeAtPrice(DataStream<StockPrice> trades){
    trades.keyBy(x->x.symbol)
      .process(new
org.apache.flink.streaming.api.functions.KeyedProcessFunction<String,StockPrice,String>()
{
```

```
       private transient MapState<Double,Long> volumes;
       public void open(Configuration p){
         volumes=getRuntimeContext().getMapState(
           new MapStateDescriptor<>("price-volumes",Double.class,Long.class));
       }
       public void processElement(StockPrice x,Context c,Collector<String> out)throws
Exception{
         long total=Optional.ofNullable(volumes.get(x.price)).orElse(0L)+x.volume;
         volumes.put(x.price,total);
         out.collect("STATE symbol="+x.symbol+", price="+x.price+",
cumulativeVolume="+total);
       }
     }).print("price-state");
  }
```

运行结果示例：price-state> STATE symbol=MSFT, price=108.5, cumulativeVolume=1760。状态按 symbol 隔离，同一价格再次成交时累加。

(5) FlinkSQL 应用

5. Flink SQL 计算成交量加权平均交易价格

```
  // SQL 按成交量计算加权平均价：sum(price*volume)/sum(volume)。
  public static void sqlWeightedAverage(DataStream<StockPrice> trades,
      org.apache.flink.table.api.bridge.java.StreamTableEnvironment tEnv){
    // StockPrice 是标准 POJO，字段名会自动映射为表字段。
    tEnv.createTemporaryView("StockTable",trades);
    org.apache.flink.table.api.Table result=tEnv.sqlQuery(
      "SELECT symbol, SUM(price * volume) / SUM(volume) AS avg_price " +
      "FROM StockTable GROUP BY symbol");
    tEnv.toChangelogStream(result).print("sql-weighted-avg");
  }
}
```

运行结果示例：sql-weighted-avg> +I[AAPL, 132.47]，后续成交会产生 -U/+U 更新记录。计算公式严格使用 SUM(price*volume)/SUM(volume)，不是简单 AVG(price)。

运行与截图说明

将上述片段按顺序合并为 StockAnalysisJob.java，在配置好 Flink 1.17.x 依赖的 Maven 项目中运行 main 方法。控制台分别搜索 volume-window、late-trade、

enriched-trade、price-state、sql-weighted-avg 五类前缀，即可截取各评分点的真实运行结果。本文档只提供示例输出，未把示例冒充实际运行截图。