2024-STA321

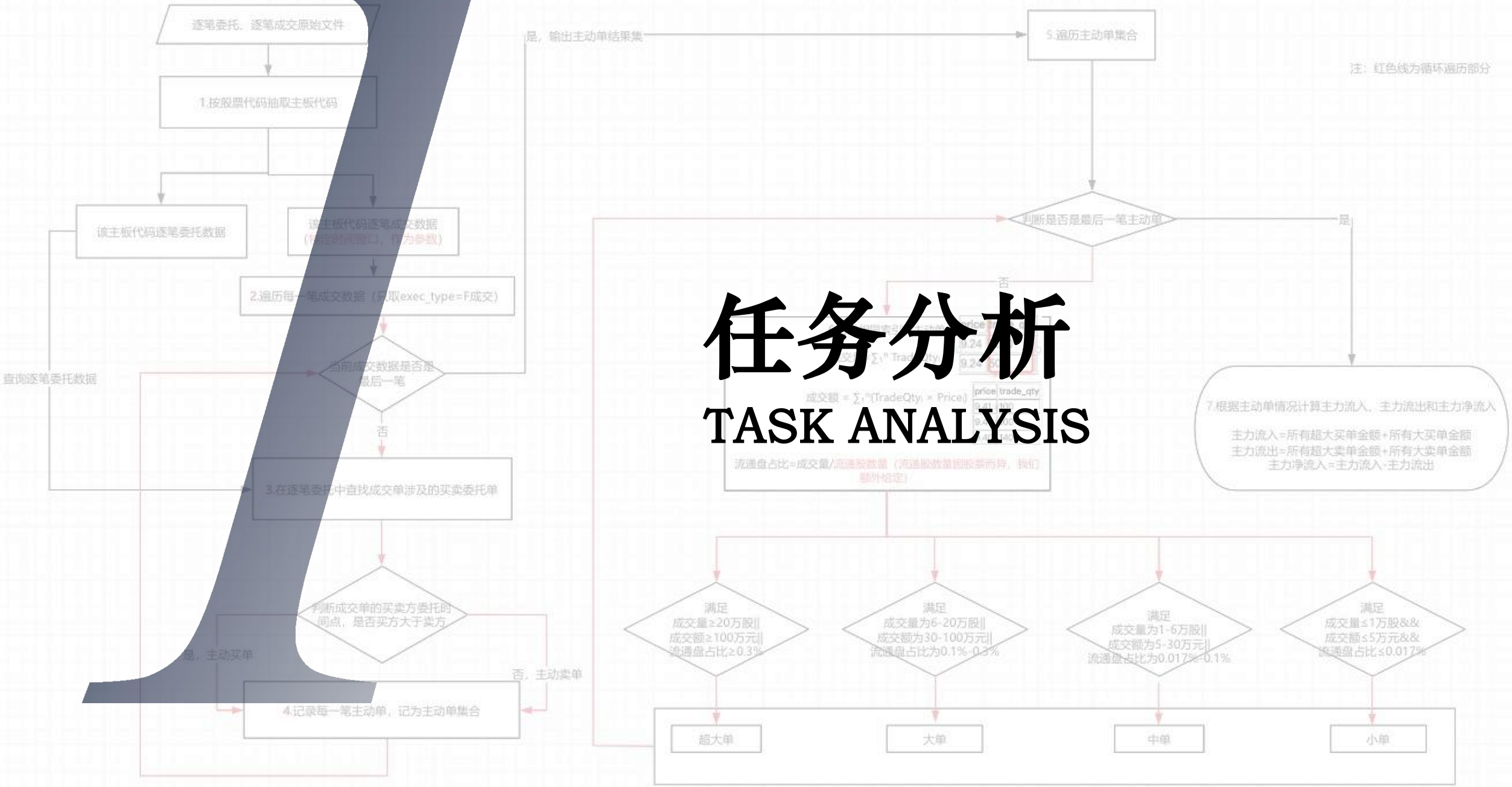
分布式存储与并行计算课程项目展示

Presenters: 徐锦焜
Team Member: 谢子成，徐锦焜
Date: 2024.12.24

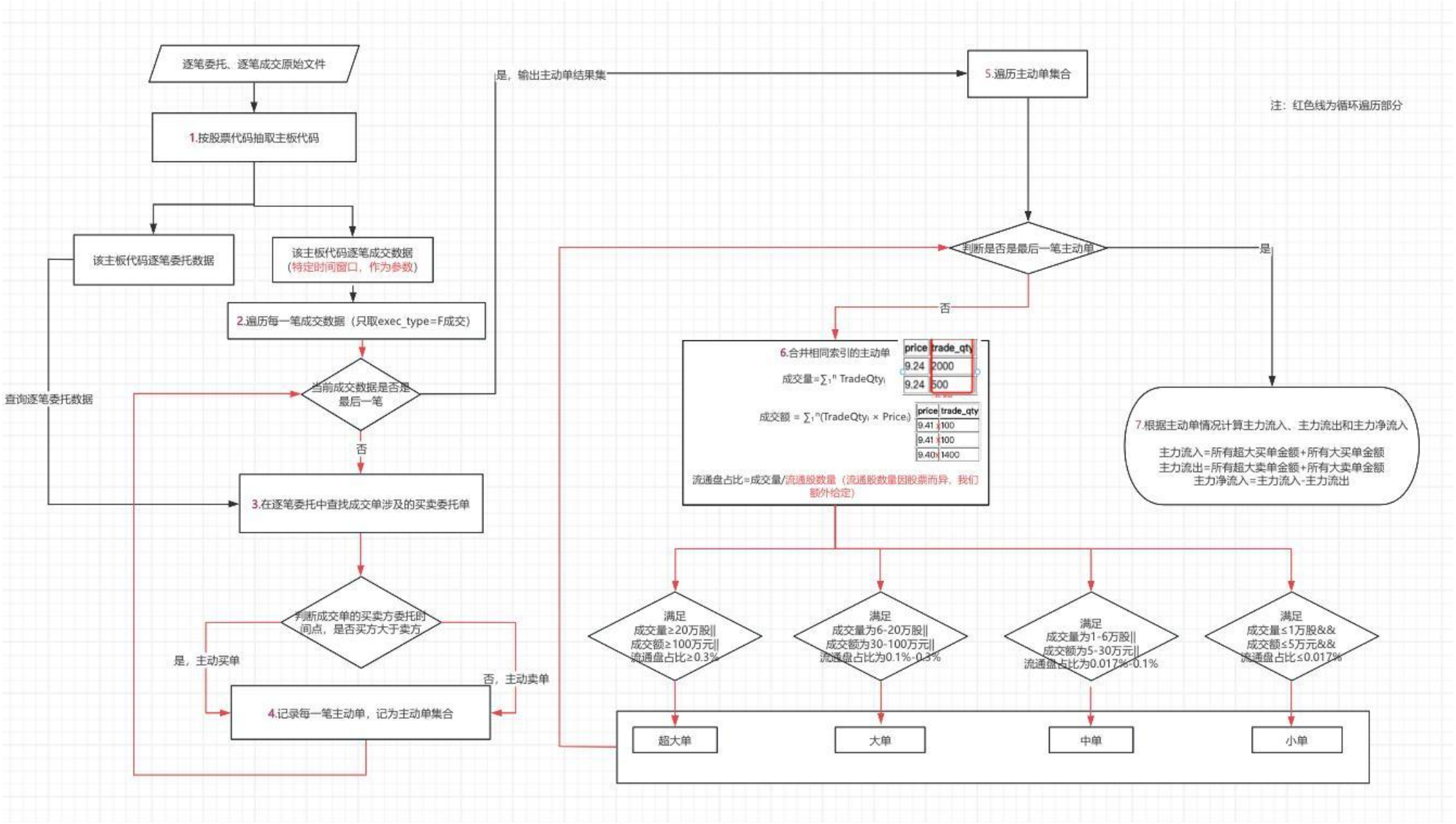
1

任务分析

TASK ANALYSIS



1 任务分析


逐笔委托、逐笔成交原始文件
1.按股票代码抽取主板代码
该主板代码逐笔委托数据
该主板代码逐笔成交数据
（特定时间窗口，作为参数）
2.遍历每一笔成交数据（只取exec_type=F成交）
当前成交数据是否是最后一笔
查询逐笔委托数据
否
3.在逐笔委托中查找成交单涉及的买卖委托单
判断成交单的买卖方委托时间点，是否买方大于卖方
是，主动买单
否，主动卖单
4.记录每一笔主动单，记为主动单集合
是，输出主动单结果集
5.遍历主动单集合
注：红色线为循环遍历部分
判断是否是最后一笔主动单
是
否
6.合并相同索引的主动单
price trade_qty
9.24 2000
9.24 500
成交量=∑ TradeQty
成交额 = ∑(TradeQty × Price)
price trade_qty
9.41 100
9.41 100
9.40 1400
流通盘占比=成交量/流通股数量（流通股数量因股票而异，我们额外给定）
7.根据主动单情况计算主力流入、主力流出和主力净流入
主力流入=所有超大买单金额+所有大买单金额
主力流出=所有超大卖单金额+所有大卖单金额
主力净流入=主力流入-主力流出
满足
成交量≥20万股||
成交额≥100万元||
流通盘占比≥0.3%
满足
成交量为6-20万股||
成交额为30-100万元||
流通盘占比为0.1%-0.3%
满足
成交量为1-6万股||
成交额为5-30万元||
流通盘占比为0.017%-0.1%
满足
成交量≤1万股&&
成交额≤5万元&&
流通盘占比≤0.017%
超大单
大单
中单
小单

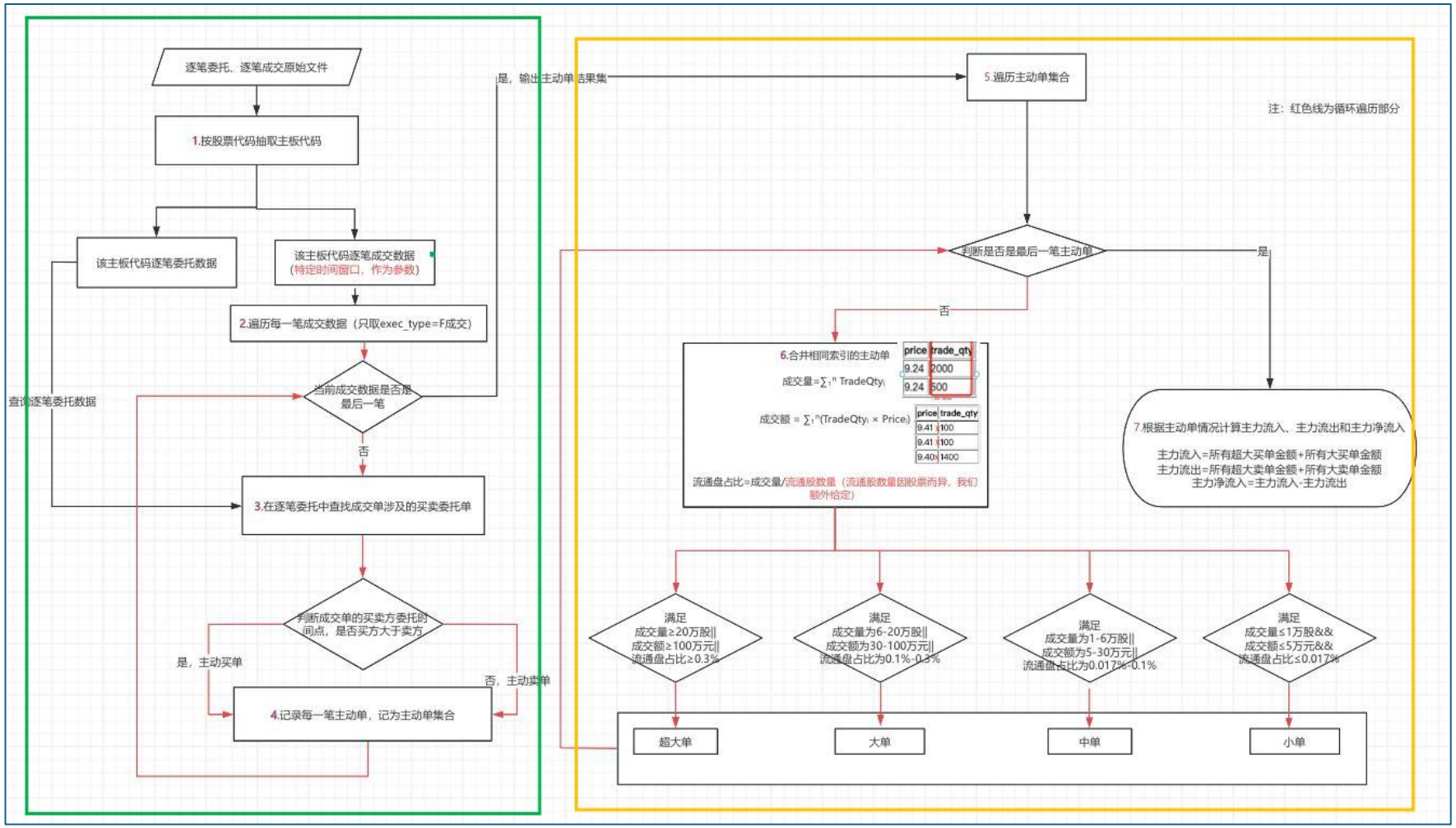
1 任务分析

最初思路：

将任务分为两部分——

1.根据需求筛选出所需主动单（Multi-Inputs）

2.根据单号进行计算并输出结果

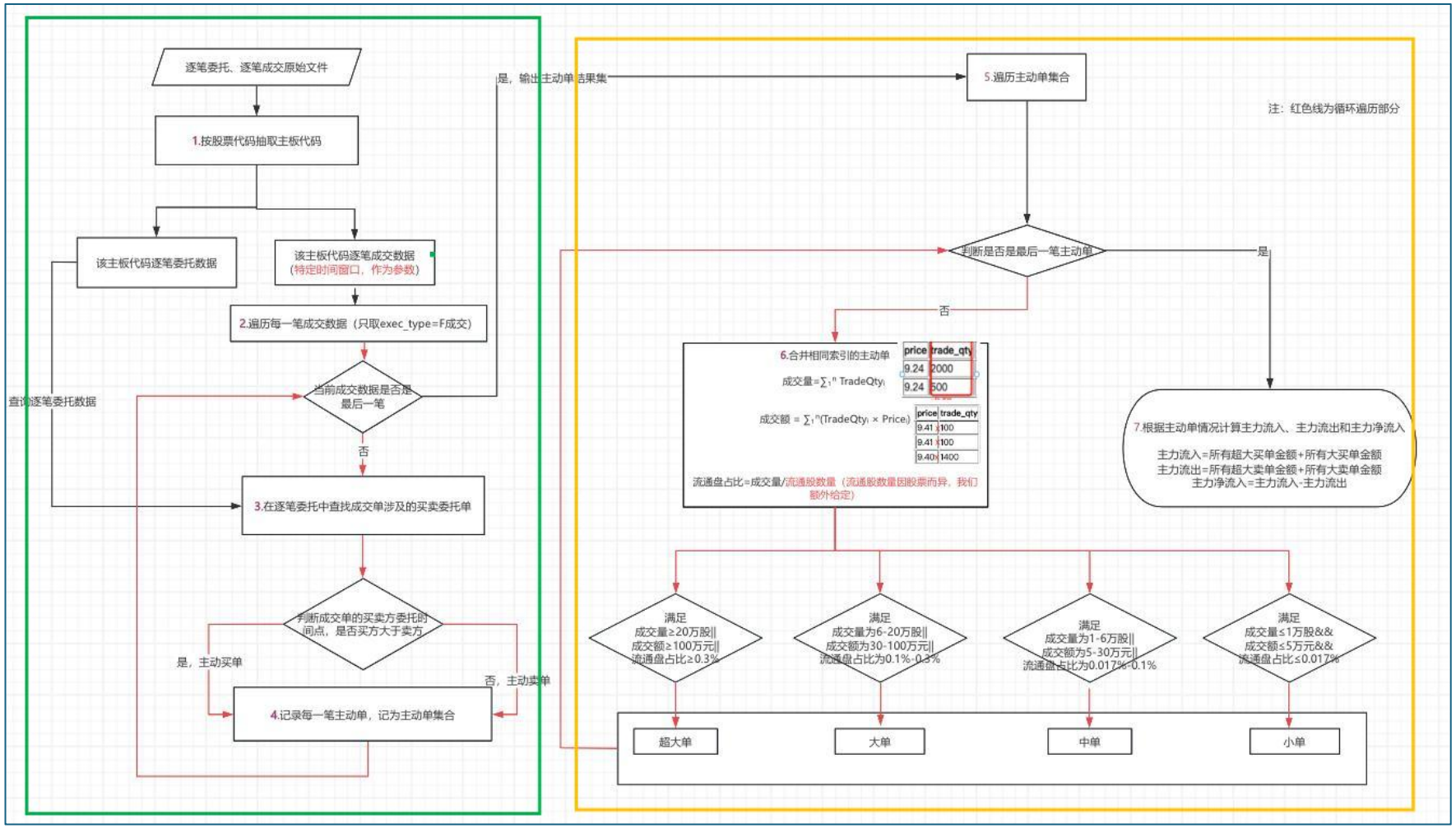


1 任务分析

关键信息：
订单索引随时间单调递增！

优化方案：
1.只使用逐笔成交单中的数据，减少Mapper数量，使计算耗时大大缩短

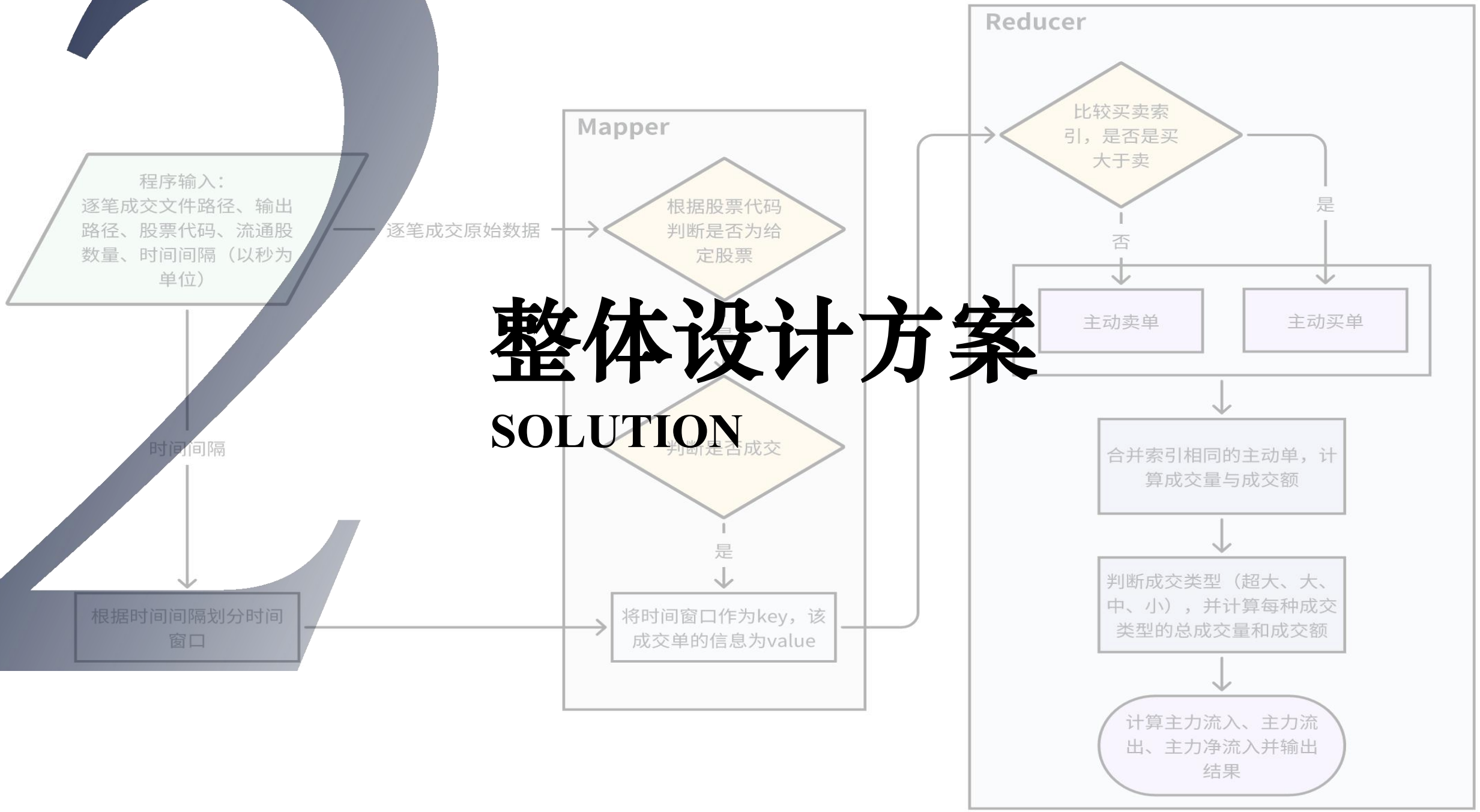
2.使用 indexOf 和 substring 手动提取字段代替split方法。这样可以避免创建多个中间数组对象，提高性能。



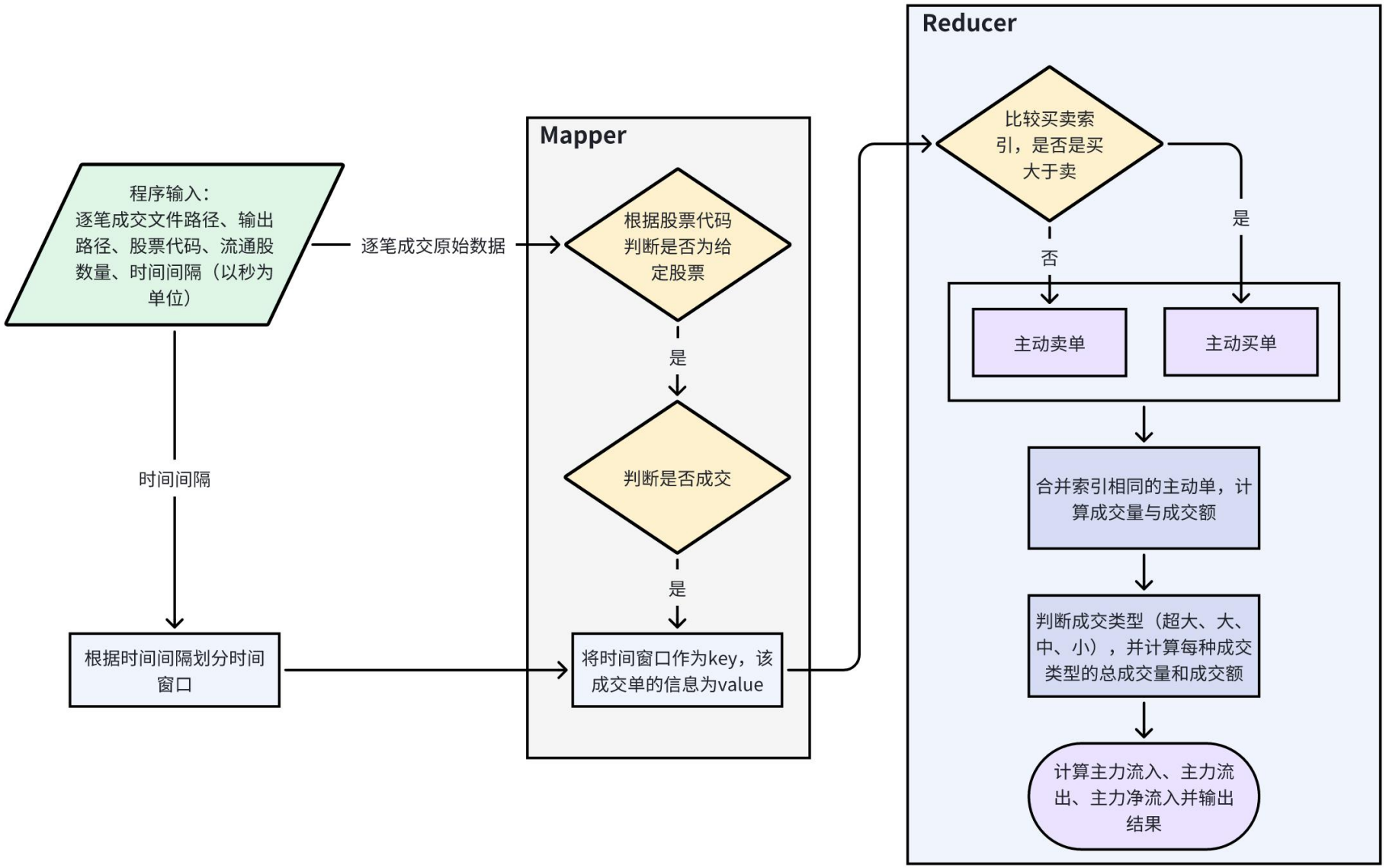
2

整体设计方案

SOLUTION



2 整体设计方案


Reducer
比较买卖索引，是否是买大于卖
是
否
主动卖单
主动买单
合并索引相同的主动单，计算成交量与成交额
判断成交类型（超大、大、中、小），并计算每种成交类型的总成交量和成交额
计算主力流入、主力流出、主力净流入并输出结果
Mapper
根据股票代码判断是否为给定股票
是
判断是否成交
是
将时间窗口作为key，该成交单的信息为value
程序输入：逐笔成交文件路径、输出路径、股票代码、流通股数量、时间间隔（以秒为单位）
逐笔成交原始数据
时间间隔
根据时间间隔划分时间窗口

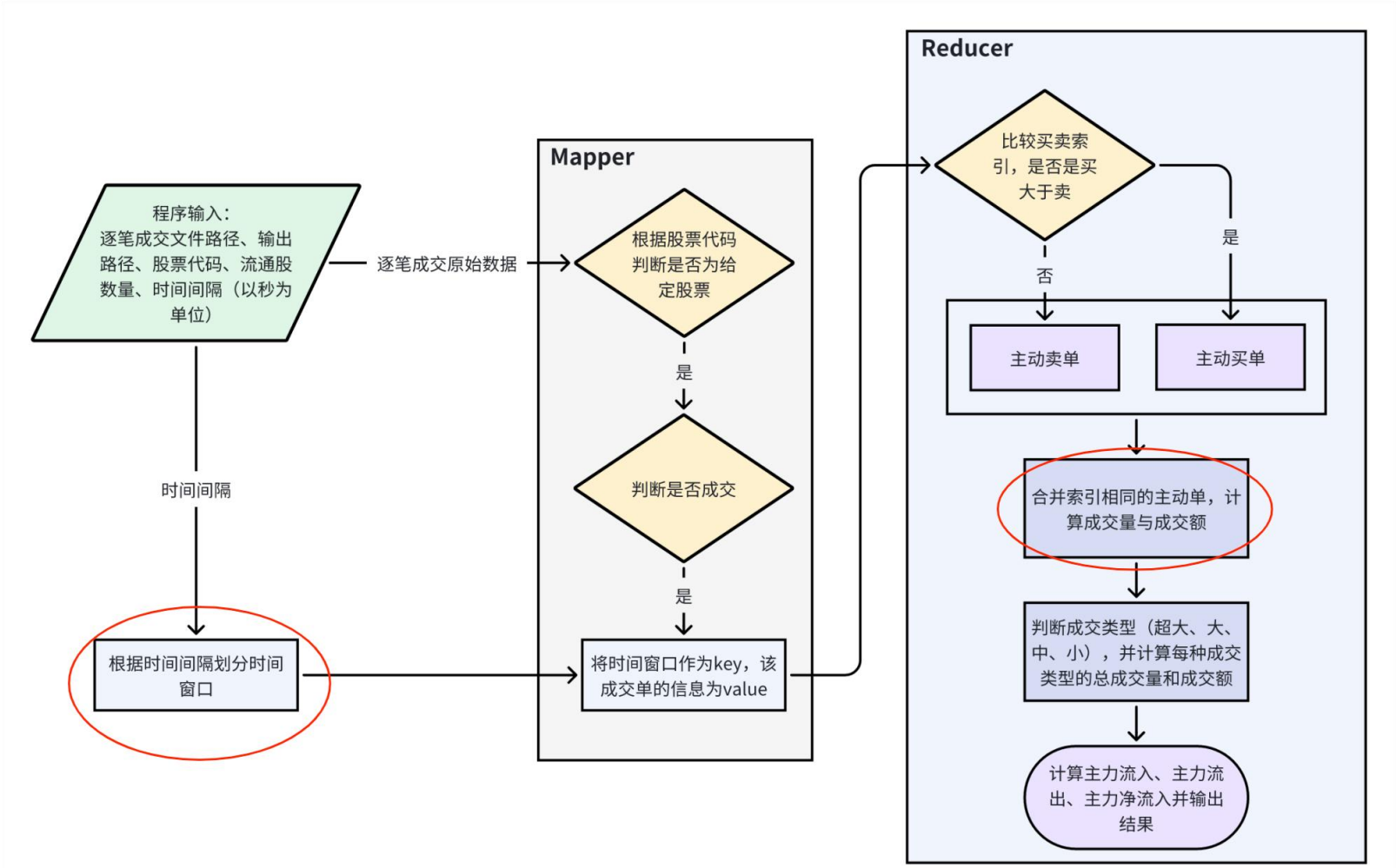
具体功能实现

SPECIFIC FUNCTION IMPLEMENTATION

```java
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class TimeWindowUtils {  4 个用法  Edward Xu +1
    public static List<String> generateTimeWindows(String intervalStr) throws Exception{  2 个用法  Edward Xu +1
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyyMMddHHmmssSSS"); //定义时间格式
        List<String> timeWindows = new ArrayList<>(); //初始化时间窗口列表
        int interval = Integer.parseInt(intervalStr) * 1000; //转换为毫秒

        String morningStart2 = "20190102093000000"; //定义上午开始时间
        String morningEnd2 = "20190102113000000"; //定义上午结束时间

        generateTimeWindowsForPeriod(sdf, morningStart2, morningEnd2, interval, timeWindows); //生成上午时间窗口

        String afternoonStart1 = "20190102130000000"; //定义下午开始时间
        String afternoonEnd1 = "20190102150000000"; //定义下午结束时间

        generateTimeWindowsForPeriod(sdf,afternoonStart1,afternoonEnd1,interval,timeWindows); //生成下午时间窗口

        return timeWindows; //返回时间窗口列表
    }

    private static void generateTimeWindowsForPeriod(SimpleDateFormat sdf, String start, String end, int interval, List<String> timeWindows) throws Exception{
        Date startTime = sdf.parse(start); //将字符串转换为时间
        Date endTime = sdf.parse(end); //将字符串转换为时间

        while(startTime.before(endTime)){ //严格小于
            Date nextTime = new Date(startTime.getTime() + interval); //计算下一个时间
            if(nextTime.after(endTime)){ //严格大于
                timeWindows.add(sdf.format(startTime)+"-"+sdf.format(endTime)); //将时间窗口添加到列表
                break;
            }
            timeWindows.add(sdf.format(startTime)+"-"+sdf.format(nextTime)); //将时间窗口添加到列表
            startTime = nextTime; //更新开始时间
        }
    }
```

3 具体功能实现

- 3.1 时间窗口分割
- 3.2 数据聚合计算



3.1 时间窗口分割

创建TimeWindowUtils类，

其中包含两个方法：

```
public class TimeWindowUtils {  4 个用法  Edward Xu +1
    public static List<String> generateTimeWindows(String intervalStr) throws Exception
```

```
private static void generateTimeWindowsForPeriod(SimpleDateFormat sdf, String start, String end, int interval, List<St
```

Reducer的输入：（key = timeWindow，订单信息）

我们需要做的工作——

对key相同的数据：

1.判断主力流入/流出

2.计算成交量与成交额

3.分类别输出结果

3.2 数据聚合计算

创建AggData类：

```
static class AggData {  4 个用法   Zicheng2333 +1
    int type; // 1:买,0:卖  2 个用法
    long totalQty; // 总成交量  13 个用法
    double totalAmount; // 总成交额  16 个用法
    AggData(int type) { this.type = type; }
} // 定义AggData类
```

使用Hash Map保存同一时间窗口下的AggData：

```
AggData ad = aggMap.getOrDefault(activeApplSeqNum, new AggData(type)); // 获取或初始化AggDat
ad.totalQty += tradeQty; // 累加tradeQty
ad.totalAmount += tradeQty * price; // 累加tradeQty * price
aggMap.put(activeApplSeqNum, ad); // 更新aggMap
```

在计算完一个key下的所有AggData后清空aggMap以便对下一个key中的数据进行操作：

```
// 先清理之前窗口的累计（
resetAggregation();
```

4

可视化展示

VISUALIZATION

感谢聆听！

THANK YOU !