# 数据源构造

堆积到 RocketMQ 的消息类型分别是交易消息和付款消息，一条交易消息可能对应多条付款消息。这两种消息堆积在三个 Topic 里面，付款消息堆积在付款 Topic，来自天猫的订单消息堆积在天猫订单 Topic 里，来自淘宝的订单消息堆积在淘宝订单 Topic 里。

为了标识流式数据的结束，我们会在消息生产结束时向三个 Topic 发送一条特殊消息，该消息的消息体为一个为 0 的 short。详情见我们提供的 Demo。

# 数据源模型

- **数据格式：订单消息**

  天猫订单 Topic 和淘宝订单 Topic，都是这样的消息结构

| 名称 | 类型 | 含义 |
| --- | --- | --- |
| **orderId** | Long | 订单 ID，全局唯一 |
| **buyerId** | String | 买家 ID |
| **productId** | String | 商品 ID |
| **salerId** | String | 卖家 ID，天猫和淘宝上的卖家 |
| **createTime** | Long | 订单的创建时间戳(ms)， 13 位数 |
| **totalPrice** | Double | 订单的总价格 |

订单消息结构定义只要保证各个字段的类型及顺序和上表保持一致。选手从 RocketMQ 消息数据时，就可以利用 kryo 反序列出订单消息。订单消息结构可以像下面一样定义：

```
public class OrderModel implements Serializable {
    public long orderId;  //订单 ID， 唯一性
    public String buyerId;  //买家 ID
    public String productId;  //商品 ID
    public String salerId;  //卖家 ID
    public long createTime;  //订单创建时间，毫秒级时间戳，13 位数
    public double totalPrice;  //订单价格
}
```

- **数据格式：付款消息**

| 名称 | 类型 | 含义 |
| --- | --- | --- |
| **orderId** | Long | 该条支付消息属于的订单 ID 标识 |
| **payAmount** | Double | 支付金额，<=订单的总价格 |
| **paySource** | Short | 支付金额来源：0，支付宝；1，红包或代金券；2，银联；3，其它 |
| **payPlatform** | Short | 支付平台：0，PC；1，无线 |
| **createTime** | Long | 付款记录创建时间戳(ms), 13 位数 |

```
public class PayModel implements Serializable {
    public long orderId; //交易订单 ID，  和 OrderModel  订单一一对应
    public double payAmount; //每次支付的费用
    /**
     * Money 来源
     * 0,支付宝
     * 1,红包或代金券
     * 2,银联
     * 3,其他
     */
    public short paySource;
    /**
     *  支付平台，  默认就两种支付平台
     * 0，pC
     * 1，无线
     */
    public short payPlatform;
    public long createTime; //付款记录创建时间，毫秒级时间戳，13 位数,我们试题要求中的交易时间指的就是该付款时间
}
```

**备注：**

一次交易对应一条订单消息，但一条订单消息可能对应多条付款消息。比如一条交易消息是：

OrderModel:orderId=14,buyerId='buyer17',productId='pro00',salerId='tb34',createTime=1234567890981, totalPrice=41

对应两条付款消息：

PayModel_1: orderId=14, payAmount=20, paySource=1, payPlatform=0, createTime =

1234567890201

PayModel_2: orderId=14, payAmount=21, paySource=0, payPlatform=1, createTime = 1234567890301

但我们会保证多次付款消息的费用之和恰好等于交易订单对应的总价。

同时赛题的统计值都是基于付款时间来计算的。

# 数据源导出

消息经过 kryo 序列化事先导入到 RocketMq 中，选手只要按照上述定义的数据格式，从 Mq 中拉取消息，kryo 反序列出自己定义的对象。具体可以参考下图所示：

```
public class OrderModel implements Serializable {
    public long orderId;
    public String buyerId;
    public String productId;
    public String salerId;
    public long createTime;
    public double totalPrice;
}
public static void main(byte [] bytes) throws Exception {
    Kryo kryo = new Kryo();
    Input input = new Input(bytes);
    input.close();
    OrderModel ret = kryo.readObject(input, OrderModel.class);
    System.out.println(ret);
}
```

**备注**：JStorm 本身依赖了 Kryo，所以选手代码不需要额外依赖 Kryo

# 数据导入

Tair 存储的是 key-value 的结构，我们约定选手存入 tair 的数据格式 key 字符串格式，value 是 number 类型。key 统一以“固定前缀_teamcode_整分时间戳”方式命名的字符串，整分时间戳就是整分时刻对应的时间戳，可以表示该一分钟，例如 2015/11/11 08:11:00 分钟对应的时间戳为 1447200660（10 位数），可表示 2015/11/11 08:11:00 ~ 2015/11/11 08:12:00（不包含该时刻）这一分钟。

| 类型 | key | vaule |
| --- | --- | --- |
| 淘宝每分钟的交易 | platformTaobao_teamcode_整分时间戳 | Number 类型 |
| 天猫每分钟的交易 | platformTmall_teamcode_整分时间戳 | Number 类型 |

| | | |
|---|---|---|
| **整分时刻无线和 PC 历史交易比** | ratio_teamcode_ 整分时间戳 | Number 类型，保留两位小数 |

# 比赛环境

- JDK1.7；JStorm 2.1.1；Tair 2.3.5；RocketMQ 3.2.6；
- 拓扑运行时间不超过 20 分钟，否则我们会对拓扑进行强杀，所以要求选手自己保重计算数据及时写入 Tair；
- 每个拓扑的总计算资源：6 核/12G 内存；
- 拓扑运行的最大 worker 数量不超过 4 个，每个 worker 的默认内存大小是 3G，默认的运行 CPU 核是 1 个；
- 集群用容器中隔离，可以保证选手拓扑间运行独立不相互影响；

# 选手代码规则约定

- 选手提交的代码 pom 打包插件要是 maven-assembly-plugin，且编译打包的地址就放在默认地址上，即生成的默认 target 目录下；
- 选手提交拓扑到 jstorm 集群，拓扑的运行路口类路径一定要是：com.alibaba.middleware.race.jstorm.RaceTopology；

具体都可以参考我们提供的 demo.

# 集群配置信息

#Tair
读写 Tair 所需的 config 主地址；
读写 Tair 所需的 config 备用地址；
读写 Tair 所需的 groupName 名称；　//唯一
读写 Tair 所需的 nameSpace;　　//唯一
#RocketMq
读写 RocketMQ 所需的各个 Topic 名称；
读写 RocketMQ 所需的消费组；　　//唯一
#Jstorm
每个队伍提交拓扑的名称；　　//唯一
队伍的 ID；　　//唯一

备注：这些配置信息，我们都会在正式提交代码前公布。选手务必按照这些规则配置信息，否则跑出来的成绩将被视为无效。

# 代码提交流程（具体的流程我们会在正式提交代码前补充）

- 选手在 https://code.aliyun.com 注册账号，将自己的项目托管到：https://code.aliyun.com；
- 再将天池账号 tianchi_bigdata 加入到这个项目，进行授权，权限设置为 master，方便天池去拉取代码编译打包；

# Demo

https://code.aliyun.com/MiddlewareRace/PreliminaryDemo

选手可以参考我们提供的 demo，这个 demo 提供:

1) 模拟数据的生成；
2) 消息模型的定义；
3) 还有一些规范的备注说明