



基于flink的 电商用户行为数据分析

讲师：武晟然



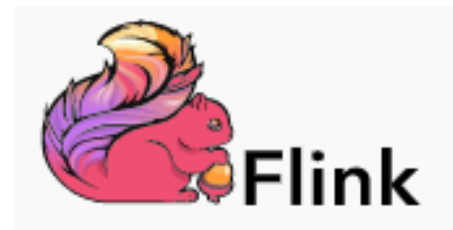
主要内容

- 批处理和流处理
- 电商用户行为分析
- 数据源解析
- 项目模块划分





批处理和流处理





批处理

- 批处理主要操作大容量静态数据集，并在计算过程完成后返回结果。

可以认为，处理的是用一个固定时间间隔分组的数据点集合。批处理模式中使用的数据集通常符合下列特征：

- 有界：批处理数据集代表数据的有限集合
- 持久：数据通常始终存储在某种类型的持久存储位置中
- 大量：批处理操作通常是处理极为海量数据集的唯一方法

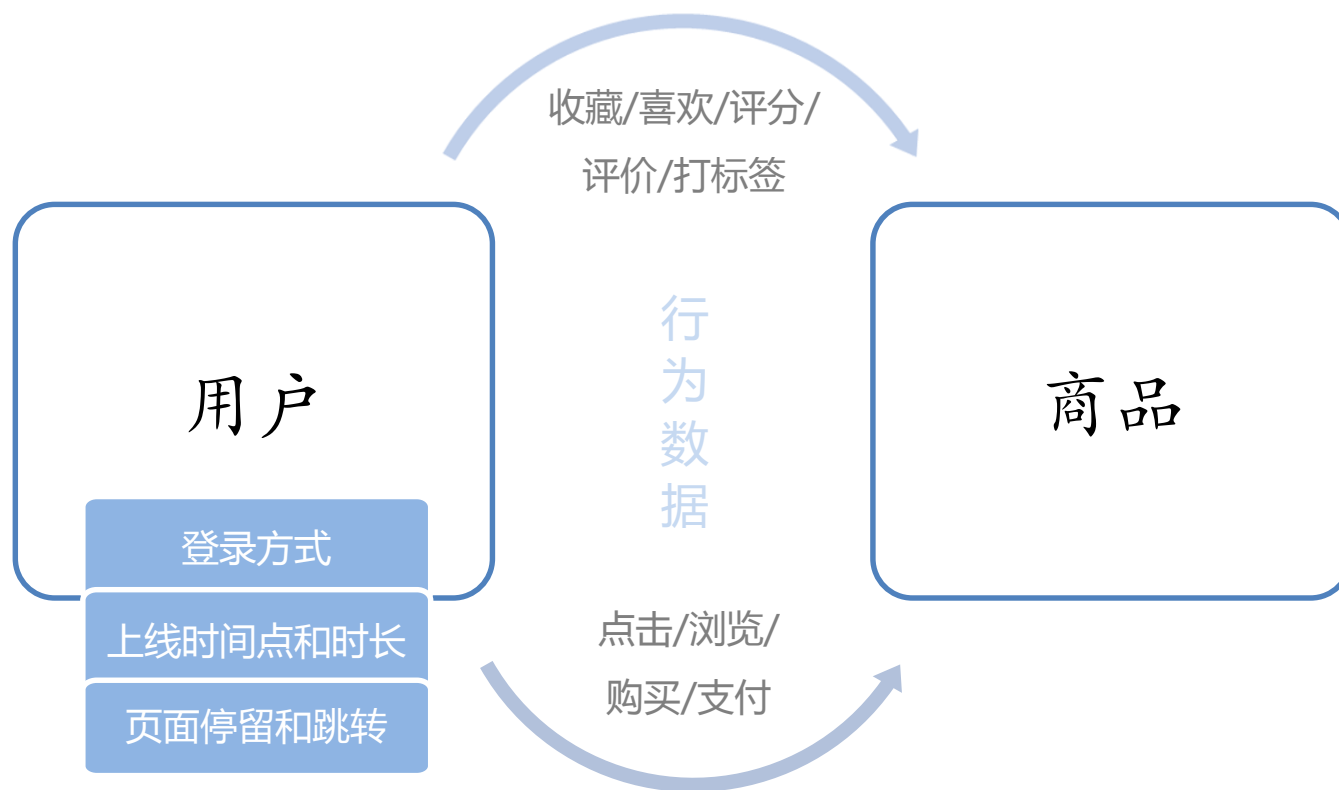


流处理

- 流处理可以对随时进入系统的数据进行计算。流处理方式无需针对整个数据集执行操作，而是对通过系统传输的每个数据项执行操作。流处理中的数据集是“无边界”的，这就产生了几点重要的影响：
 - 可以处理几乎无限量的数据，但同一时间只能处理一条数据，不同记录间只维持最少量的状态
 - 处理工作是基于事件的，除非明确停止否则没有“尽头”
 - 处理结果立刻可用，并会随着新数据的抵达继续更新。



电商用户行为分析





电商用户行为分析

- 统计分析
 - 点击、浏览
 - 热门商品、近期热门商品、分类热门商品，流量统计
- 偏好统计
 - 收藏、喜欢、评分、打标签
 - 用户画像，推荐列表（结合特征工程和机器学习算法）
- 风险控制
 - 下订单、支付、登录
 - 刷单监控，订单失效监控，恶意登录（短时间内频繁登录失败）监控



电商用户行为分析——项目模块设计

实时统计分析

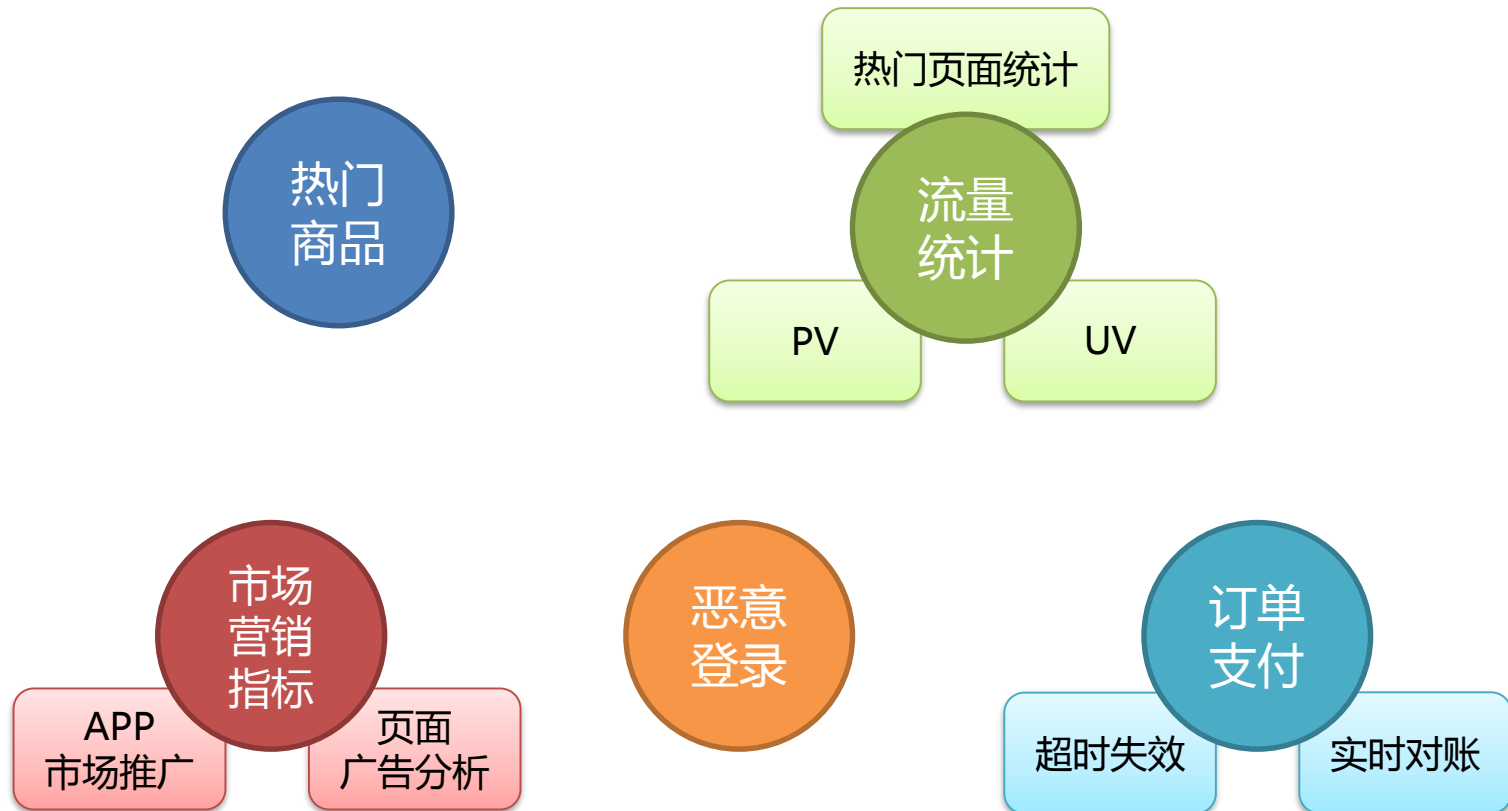
- 实时热门商品统计
- 实时热门页面流量统计
- 实时访问流量统计
- APP 市场推广统计
- 页面广告点击量统计

业务流程及风险控制

- 页面广告黑名单过滤
- 恶意登录监控
- 订单支付失效监控
- 支付实时对账



电商用户行为分析——项目模块设计





数据源解析

- 用户行为数据

UserBehavior.csv

e.g. 543462, 1715, 1464116, pv, 1511658000

- web 服务器日志

apache.log

e.g. 66.249.73.135 - - 17/05/2015:10:05:40 +0000 GET

/blog/tags/ipv6



数据源解析

- 数据结构 —— UserBehavior

字段名	数据类型	说明
userId	Long	加密后的用户ID
itemId	Long	加密后的商品ID
categoryId	Int	加密后的商品所属类别ID
behavior	String	用户行为类型，包括('pv' , 'buy' , 'cart' , 'fav')
timestamp	Long	行为发生的时间戳，单位秒



数据源解析

- 数据结构 —— ApacheLogEvent

字段名	数据类型	说明
ip	String	访问的 IP
userId	Long	访问的 user ID
eventTime	Long	访问时间
method	String	访问方法 GET/POST/PUT/DELETE
url	String	访问的 url



- 实时热门商品统计
- 实时流量统计
- 市场营销分析
- 恶意登录监控
- 订单支付实时监控





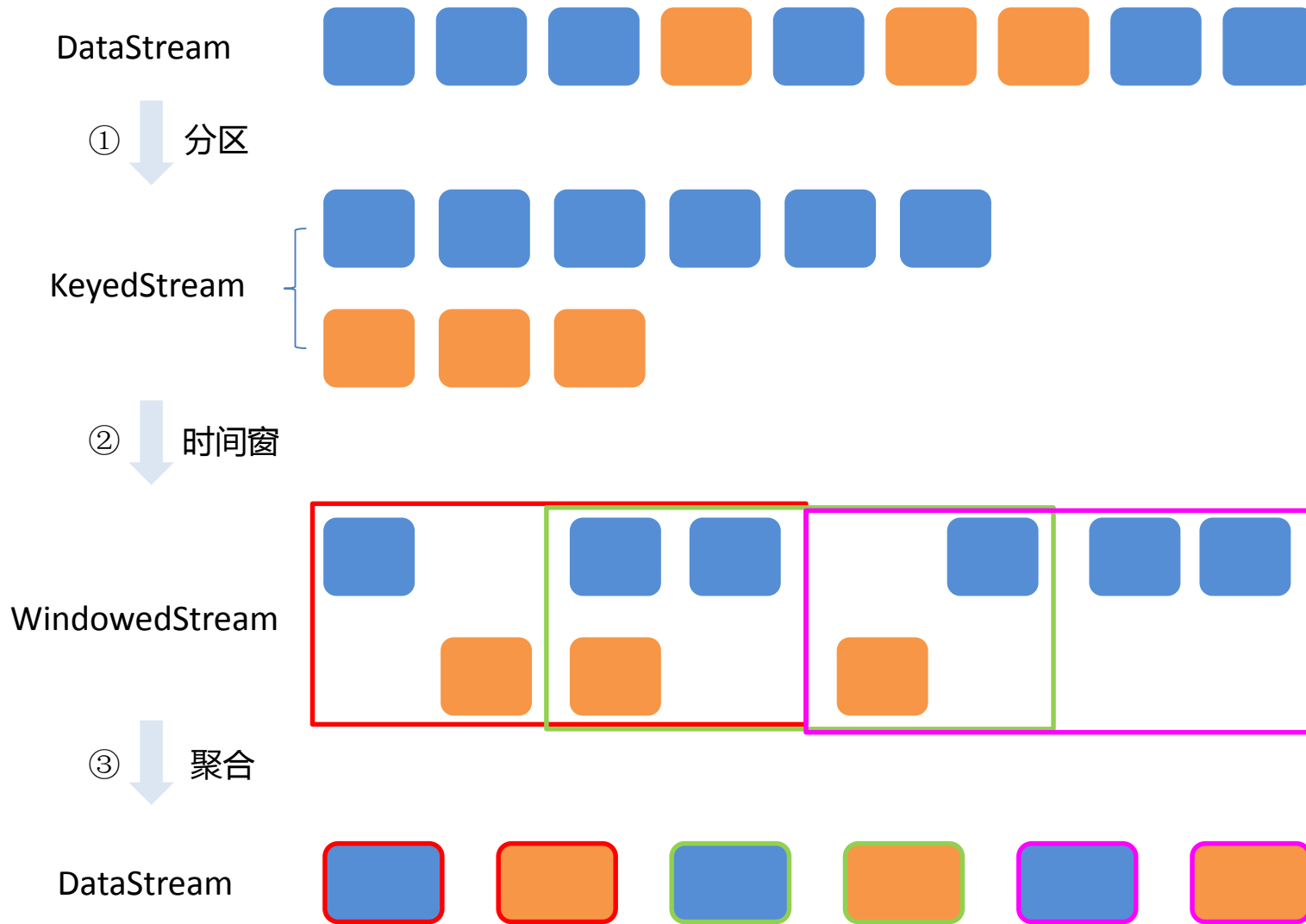
热门实时商品统计

- 基本需求

- 统计近1小时内的热门商品，每5分钟更新一次
- 热门度用浏览次数（“pv”）来衡量

- 解决思路

- 在所有用户行为数据中，过滤出浏览（“pv”）行为进行统计
- 构建滑动窗口，窗口长度为1小时，滑动距离为5分钟

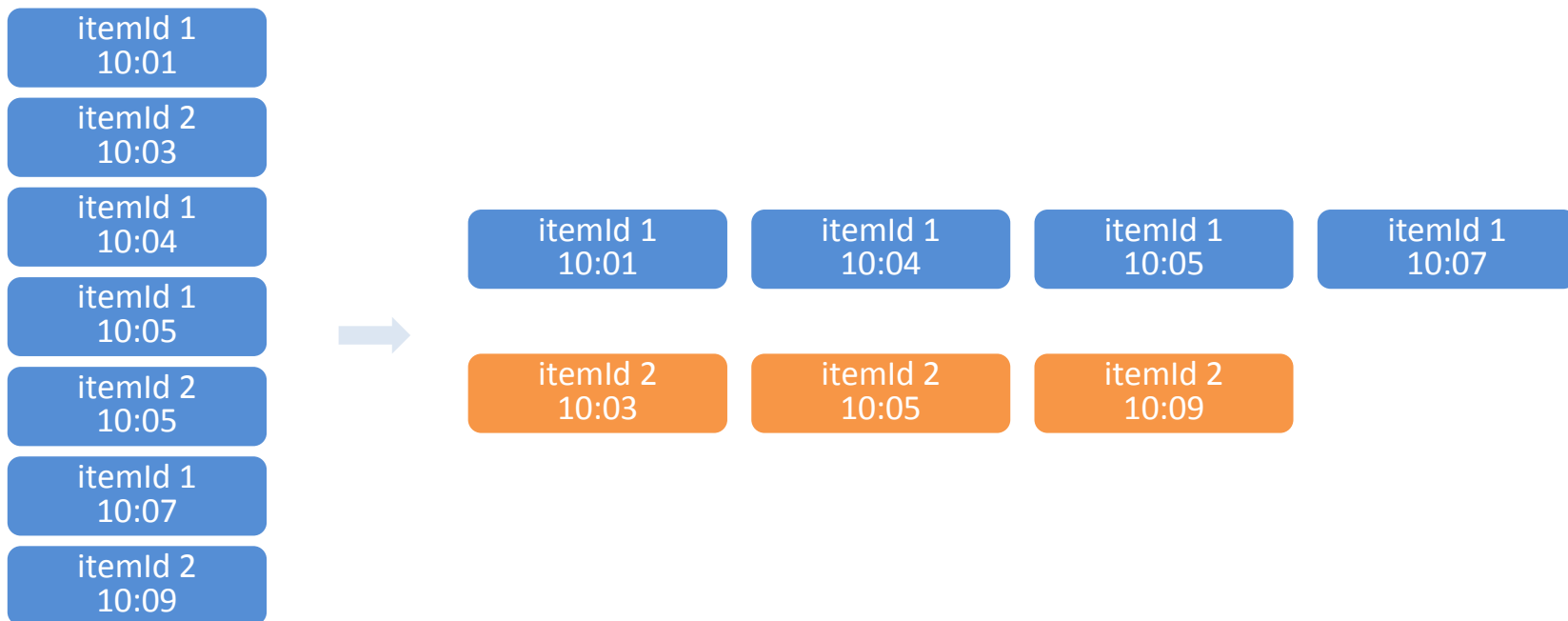




热门实时商品统计

- 按照商品Id进行分区

`.keyBy("itemId")`

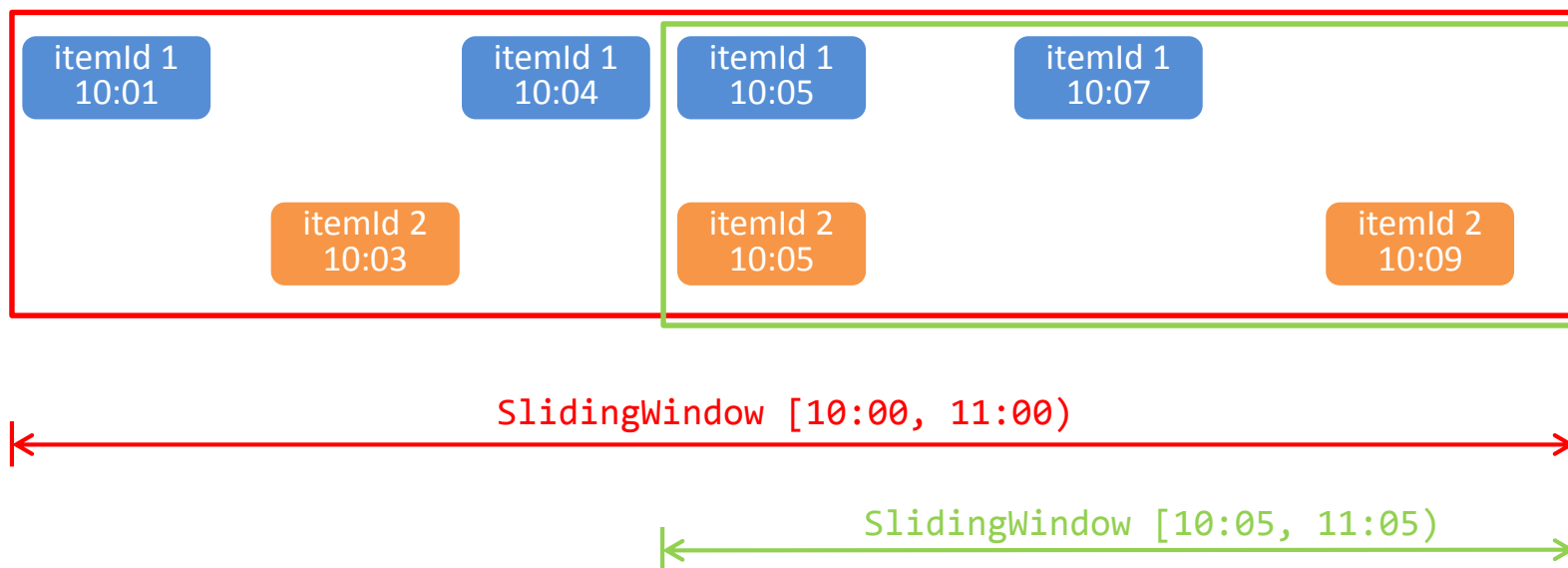




热门实时商品统计

- 设置时间窗口

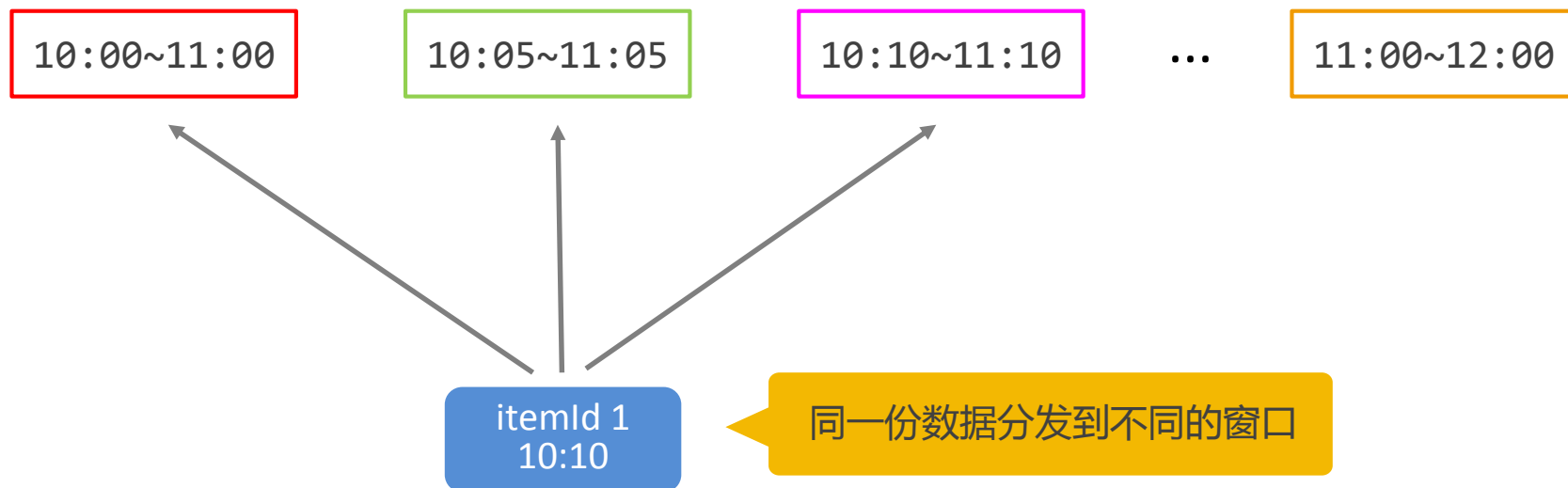
```
.timeWindow(Time.minutes(60), Time.minutes(5))
```





热门实时商品统计

- 时间窗口 (timeWindow) 区间为左闭右开
- 同一份数据会被分发到不同的窗口





热门实时商品统计

- 窗口聚合

```
.aggregate(new CountAgg(), new WindowResultFunction())
```

定义窗口聚合规则

定义输出数据结构

```
new ItemViewCount(itemId, windowEnd,  
count)
```

WindowedStream



DataStream<ItemViewCount>



热门实时商品统计

- 窗口聚合策略——每出现一条记录就加一

```
public static class ItemCountAgg implements AggregateFunction<UserBehavior, Long, Long>{  
    @Override  
    public Long createAccumulator() {  
        return 0L;  
    }  
    @Override  
    public Long add(UserBehavior value, Long accumulator) {  
        return accumulator + 1;  
    }  
    @Override  
    public Long getResult(Long accumulator) {  
        return accumulator;  
    }  
    @Override  
    public Long merge(Long a, Long b) {  
        return a + b;  
    }  
}
```

- 实现 AggregateFunction 接口
 - interface AggregateFunction<IN, ACC, OUT>



热门实时商品统计

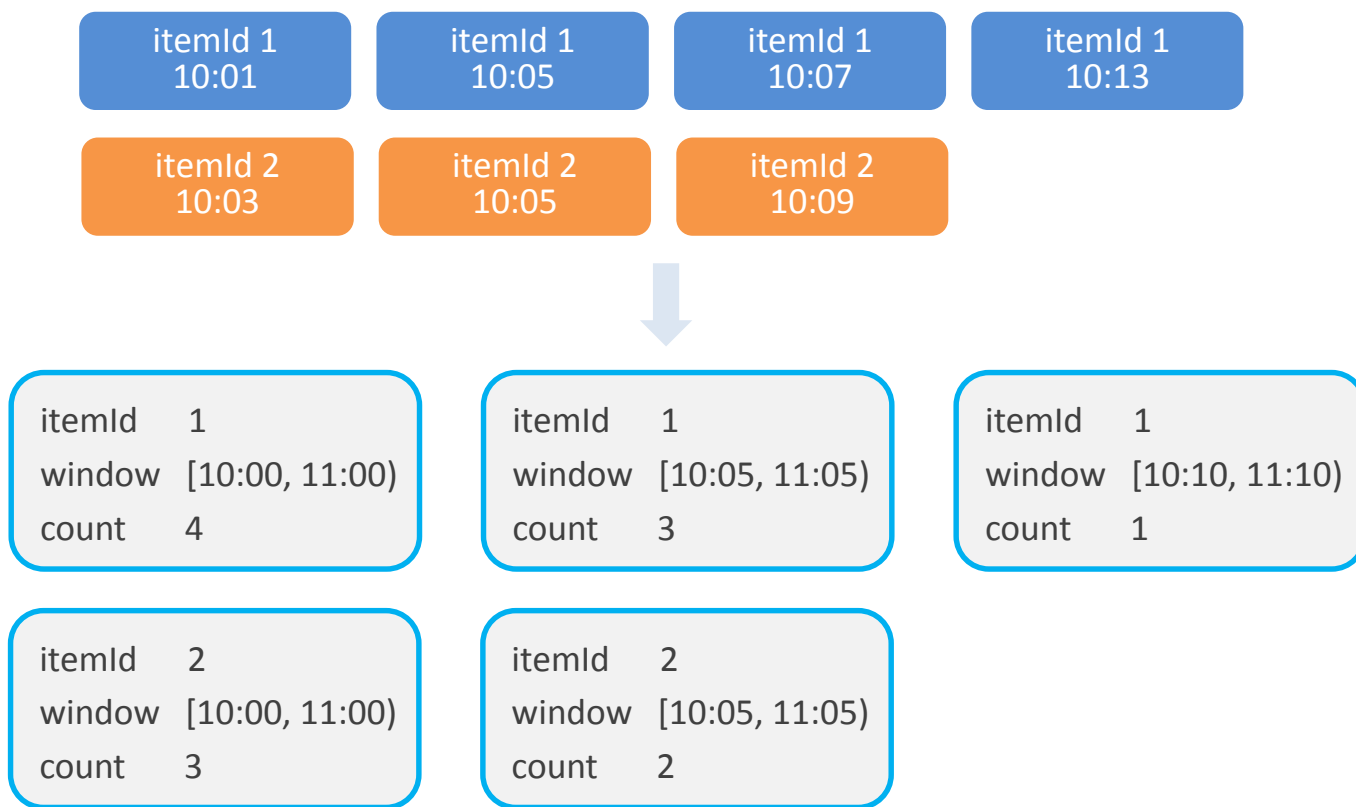
- 定义输出结构 —— ItemViewCount(itemId, windowEnd, count)
- 实现 WindowFunction 接口
 - interface WindowFunction<IN, OUT, KEY, W extends Window>
 - IN: 输入为累加器的类型, Long
 - OUT: 窗口累加以后输出的类型为 ItemViewCount(itemId: Long, windowEnd: Long, count: Long), windowEnd为窗口的结束时间, 也是窗口的唯一标识
 - KEY: Tuple泛型, 在这里是 itemId, 窗口根据itemId聚合
 - W: 聚合的窗口, w.getEnd 就能拿到窗口的结束时间

```
public void apply(Tuple tuple, TimeWindow window,
                 Iterable<Long> input, Collector<ItemViewCount> out) throws Exception {
    Long itemId = tuple.getField(pos: 0);
    Long windowEnd = window.getEnd();
    Long count = input.iterator().next();
    out.collect( new ItemViewCount(itemId, windowEnd, count) );
}
```



热门实时商品统计

- 窗口聚合示例





热门实时商品统计

- 进行统计整理 —— `keyBy("windowEnd")`

itemId 1
window [10:00, 11:00)
count 4

itemId 1
window [10:05, 11:05)
count 3

itemId 1
window [10:10, 11:10)
count 1

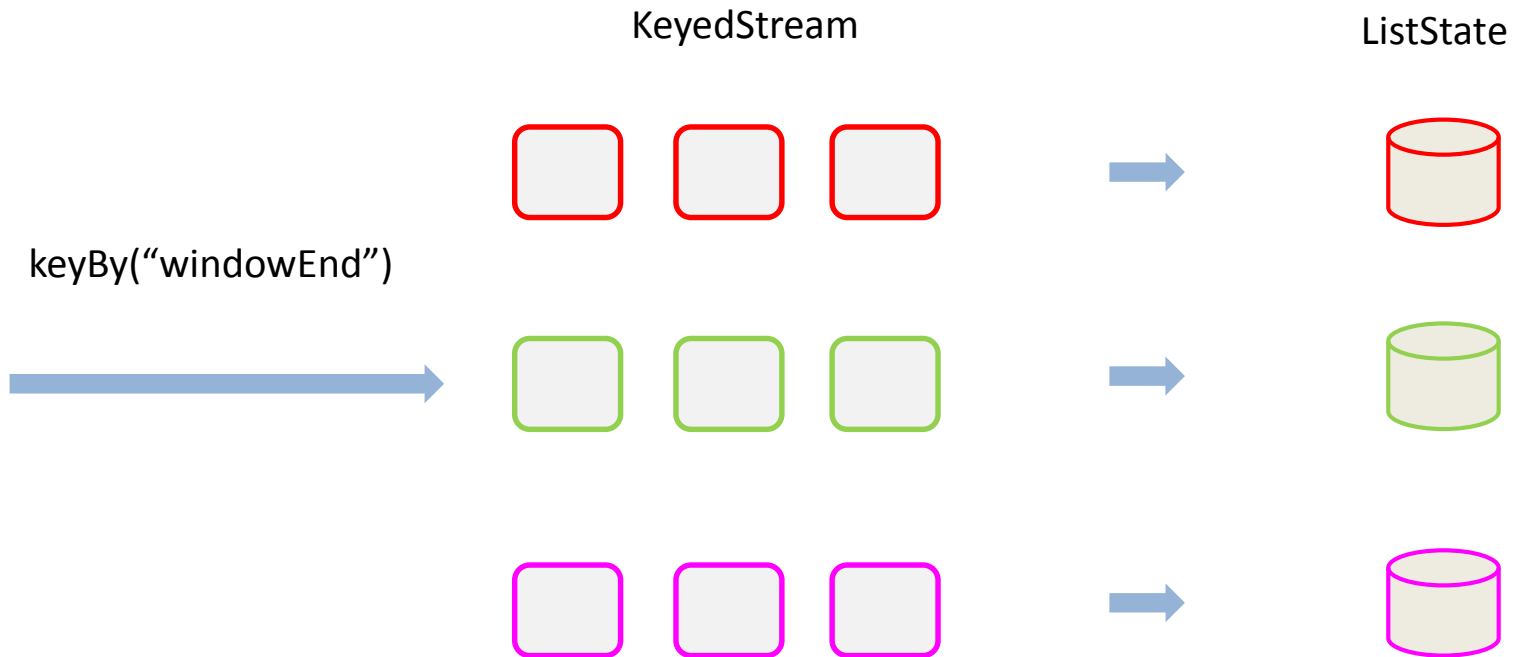
itemId 2
window [10:00, 11:00)
count 3

itemId 2
window [10:05, 11:05)
count 2



热门实时商品统计

- 状态编程





热门实时商品统计

- 最终排序输出 —— keyedProcessFunction
 - 针对有状态流的底层API
 - KeyedProcessFunction 会对分区后的每一条子流进行处理
 - 以 windowEnd 作为 key，保证分流以后每一条流的数据都在一个时间窗口内
 - 从 ListState 中读取当前流的状态，存储数据进行排序输出



热门实时商品统计

- 用 ProcessFunction 来定义 KeyedStream 的处理逻辑
- 分区之后，每个 KeyedStream 都有其自己的生命周期
 - open：初始化，在这里可以获取当前流的状态
 - processElement：处理流中每一个元素时调用
 - onTimer：定时调用，注册定时器 Timer 并触发之后的回调操作



热门实时商品统计



定时器触发时，相当于收到了大于 $\text{windowEnd} + 100$ 的 watermark，可以认为这时窗口已经收集到了所有数据，从 **ListState** 中读取进行处理



实时流量统计 —— 热门页面

- 基本需求
 - 从web服务器的日志中，统计实时的热门访问页面
 - 统计每分钟的ip访问量，取出访问量最大的5个地址，每5秒更新一次
- 解决思路
 - 将 apache 服务器日志中的时间，转换为时间戳，作为 Event Time
 - 构建滑动窗口，窗口长度为1分钟，滑动距离为5秒



实时流量统计 —— PV 和 UV

- 基本需求
 - 从埋点日志中，统计实时的 PV 和 UV
 - 统计每小时的访问量（PV），并且对用户进行去重（UV）
- 解决思路
 - 统计埋点日志中的 pv 行为，利用 Set 数据结构进行去重
 - 对于超大规模的数据，可以考虑用布隆过滤器进行去重



市场营销分析 —— APP 市场推广统计

- 基本需求
 - 从埋点日志中，统计 APP 市场推广的数据指标
 - 按照不同的推广渠道，分别统计数据
- 解决思路
 - 通过过滤日志中的用户行为，按照不同的渠道进行统计
 - 可以用 process function 处理，得到自定义的输出数据信息



市场营销分析 —— 页面广告统计

- 基本需求
 - 从埋点日志中，统计每小时页面广告的点击量，5秒刷新一次，并按照不同省份进行划分
 - 对于“刷单”式的频繁点击行为进行过滤，并将该用户加入黑名单
- 解决思路
 - 根据省份进行分组，创建长度为1小时、滑动距离为5秒的时间窗口进行统计
 - 可以用 process function 进行黑名单过滤，检测用户对同一广告的点击量，如果超过上限则将用户信息以侧输出流输出到黑名单中



恶意登录监控

- 基本需求
 - 用户在短时间内频繁登录失败，有程序恶意攻击的可能
 - 同一用户（可以是不同IP）在2秒内连续两次登录失败，需要报警
- 解决思路
 - 将用户的登录失败行为存入 ListState，设定定时器2秒后触发，查看 ListState 中有几次失败登录
 - 更加精确的检测，可以使用 CEP 库实现事件流的模式匹配



订单支付实时监控

- 基本需求

- 用户下单之后，应设置订单失效时间，以提高用户支付的意愿，并降低系统风险
- 用户下单后15分钟未支付，则输出监控信息

- 解决思路

- 利用 CEP 库进行事件流的模式匹配，并设定匹配的时间间隔
- 也可以利用状态编程，用 process function 实现处理逻辑



订单支付实时对账

- 基本需求

- 用户下单并支付后，应查询到账信息，进行实时对账
- 如果有不匹配的支付信息或者到账信息，输出提示信息

- 解决思路

- 从两条流中分别读取订单支付信息和到账信息，合并处理
- 用 connect 连接合并两条流，用 coProcessFunction 做匹配处理



Q & A