# 中间件实现技术实验报告

姓名：孙一铭

学号：2017104105

专业：软件工程

学院：信息学院

报告课题：模拟实时统计阿里双十一交易数据

目录

[一、 背景介绍](#_Toc20696_WPSOffice_Level1) 1

[1. 项目名称](#_Toc20064_WPSOffice_Level2) 1

[2. 问题描述](#_Toc25517_WPSOffice_Level2) 1

[3. 实现预期结果](#_Toc19487_WPSOffice_Level2) [1](#_Toc19487_WPSOffice_Level2)

[二、源代码分析](#_Toc20064_WPSOffice_Level1) 1

1. 技术分析 2

[2. OrderSystem](#_Toc29168_WPSOffice_Level2) 2

[A. OrderSystem](#_Toc20064_WPSOffice_Level3) 3

[B. OrderSystemImpl](#_Toc20064_WPSOffice_Level3) 3

[C. OrderSystemImpDemo 4](#_Toc20064_WPSOffice_Level3)

[D. Tester](#_Toc20064_WPSOffice_Level3) 5

[E. Util](#_Toc20064_WPSOffice_Level3) 5

[F. Table](#_Toc20064_WPSOffice_Level3) 6

[G. Store](#_Toc20064_WPSOffice_Level3) 6

[H. Model](#_Toc20064_WPSOffice_Level3) 7

[三、 实现结果](#_Toc20696_WPSOffice_Level1) 7

[1. 预期结果](#_Toc20064_WPSOffice_Level2) 7

[2. 图表分析](#_Toc20064_WPSOffice_Level2) 7

[3. 总结](#_Toc20064_WPSOffice_Level2) 7

1. **背景介绍**
2. 项目名称

模拟实时统计阿里双十一交易数据

1. 问题描述

我们将脱敏后的双11交易数据存储在RocketMQ中，实现者利用JStorm从RocketMQ拉取数据，按照要求实时计算出结果，并将结果写入Tair。我们约定用整分时刻对应的时间戳来标识这一分钟，比如2015/11/11 08:11:00分钟对应的整分时间戳为1447200660，那么我们就用这个时间戳来表示这一分钟，即2015/11/11 08:11:00~2015/11/11 08:12::00(不包含这一时刻)。  
我们需要计算的统计值如下：  
1）分平台（淘宝、天猫）每分钟的交易金额，即计算每一分钟内的交易值。比如计算2015/11/11 11:11:00这一分钟的淘宝交易金额，就是统计从11:11:00分开始到11:12:00（不包含该时刻）这一分钟内的淘宝的交易总金额；  
2）每整分时刻无线和PC端总交易金额比值；注意这里统计的是整分时刻对应的总交易比值，要将过去发生的所有交易都计算进去。比如计算2015/11/11 11:11:00整分时刻的交易比值，那么需要统计从交易开始时间到2015/11/11 11:12:00（不包含该时刻）这一区间内所有的交易值，来计算无线和PC端的比值；  
  最终存储到Tair 中的数据是key-value 的结构，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，例如2015/11/11 08:11:00分钟对应的时间戳为447200660(注意是10位数)，即该时刻的整分时间戳。

1. 实现预期结果

(1)淘宝每分钟的交易金额的key是“****platformTaobao\_teamcode\_整分时间戳****”， 值都是number类型；例如存入Tair的分平台淘宝实时交易金额格式如下： {"platformTaobao\_1234auu\_1447200660": 100}

(2)天猫每分钟的交易金额的key是“****platformTmall\_ teamcode \_整分时间戳****”， 值都是number类型；例如存入Tair的分平台天猫实时交易金额格式如下： {"platformTmall\_1234auu\_1447200660": 200}

(3) 每整分时刻无线和PC端总交易金额比值的key为“****ratio\_ teamcode\_整分时间 戳****”, 值是一个保留两位小数的number类型。例如存入Tair的无线和PC端实时交易 金额比值格式如下：{"ratio\_1234auu\_1447200660": 2.12}

**二、源代码分析**

1. 技术分析

(1) RocketMQ-3.2.6环境配置

### 下载[release包](https://github.com/alibaba/RocketMQ/releases/download/v3.2.6/alibaba-rocketmq-3.2.6.tar.gz" \t "_blank) 下载[源码](https://github.com/alibaba/RocketMQ" \t "_blank)

· cd /alibaba-rocketmq/bin (第１步中解压出的文件夹)

· 启动namesrv   
· 部署Broker(单Master模式)

1. Tair-2.3.4

Tair是一个高性能，分布式，可扩展，高可靠的key/value结构存储系统！

Tair专为小文件优化，并提供简单易用的接口。

1. JStorm-2.1.1环境配置

**1. 安装zeromq**

**2. 安装jzmq**

**3. 安装zookeeper**

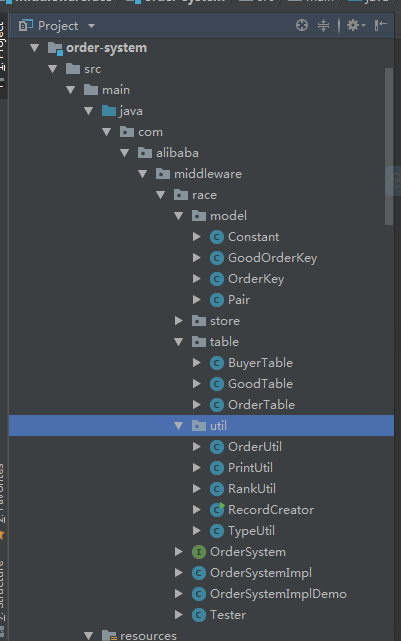
**4. 安装jstorm**

**5. 全部完成后使用jps命令应该能够看到如下进程**

admin@ubuntu:~$ jps 9840 NimbusServer 8051 Bootstrap 9573 QuorumPeerMain 10086 Supervisor 8924 Jps

2. OrderSystem

以下为提供交易订单系统接口，提供实现代码机制，具体分析如下：



1. **OrderSystem 接口类**

其中包含construct（）方法，此方法用来实现，测试程序调用此接口构建交易订单记录查询系统。

定义了一个内部接口KeyValue ：用于访问记录中的Key-Value接口

定义了一个Result内部接口：用于访问单条订单记录的接口

1. **OrderSystemImpl** ：OrderSystem的实现类，交易订单系统接口

public void construct(Collection<String> orderFiles,Collection<String> buyerFiles,Collection<String> goodFiles Collection<String> storeFolders) throws IOException, InterruptedException{

orderTable = new OrderTable(orderFiles,storeFolders);

orderTable.buildIndex();

goodTable = new GoodTable(goodFiles, storeFolders);

goodTable.buildIndex();

buyerTable = new BuyerTable(buyerFiles, storeFolders);

buyerTable.buildIndex();

}

测试程序调用此接口构建交易订单记录查询系统，其中各个参数含义orderFiles：订单文件列表；buyerFiles：买家文件列表；goodFiles：商品文件列表；storeFolders：存储文件夹的目录，保证每个目录是有效的，每个目录在不同的磁盘上。

内部类的实现：此内部类实现了K——V键值的访问。

public static class BaseKeyValue implements KeyValue{

String key; //定义key键

String value;//定义值

public BaseKeyValue(String key,String value){ //内部类的构造方法

this.key = key;

this.value = value;

}

@Override //对方法的重写

public String key(){

return this.key;

}

@Override

public String valueAsString(){

return this.value;

}

@Override

public long valueAsLong() throws TypeException{

try {

return Long.valueOf(this.value);

}catch (Exception e){

/\*

System.out.println("ParseLone Fail:" + this.key +"-" + this.value);

try {

return Double.valueOf(this.value).longValue();

}catch (Exception ee){

throw new TypeException();

}\*/

throw new TypeException();

}

}

@Override

public double valueAsDouble() throws TypeException{

try {

return Double.valueOf(this.value);

}catch (Exception e){

throw new TypeException();

}

}

@Override

public boolean valueAsBoolean() throws TypeException{

try {

return Boolean.valueOf(this.value);

}catch (Exception e){

throw new TypeException();

}

}

@Override

public String toString(){

return key+":" + value;

}

}

1. **OrderSystemImpDemo** 交易订单系统接口实现实例类。

List<File> \_orderFiles; //存储订单的文件集合

List<File> \_buyerFiles; //存储买家的文件集合

List<File> \_goodFiles; //存储商品的文件集合

List<RandomAccessFile> \_orderAccessList; //定义随机访问订单列表

List<RandomAccessFile> \_buyerAccessList; //定义随机访问买家列表

List<RandomAccessFile> \_goodAccessList; //定义随机访问商品列表

//此方法实现父类OrderSystem 功能

public void construct(Collection<String> orderFiles,Collection<String> buyerFiles, Collection<String> goodFiles, Collection<String> storeFolders)；

//测试创建索引文件

RandomAccessFile randomAccessFile = new RandomAccessFile(storeFolders.iterator().next()+"/orderIndex","rw");

//读取数据建立索引

//读取order建立索引

orderIdMap = new HashMap<Long, Map<String, String>>(1024\* 512);

orderByBuyerMap = new HashMap<String, List<Long>>(1024 \* 8);

orderByGoodMap = new HashMap<String, List<Long>>(1024 \* 8);

//读取buyer列表

buyerIdMap = new HashMap<String, Map<String, String>>(1024 \* 8);

//读取good列表

goodIdMap = new HashMap<String, Map<String, String>>(1024 \* 8);

//查询订单号为orderid的指定字段 orderId:订单号;keys:待查询的字段

public Result queryOrder(long orderId, Collection<String> keys){

return getByOrderId(orderId,keys);

//查询某位买家createtime字段从[startTime, endTime) 时间范围内发生的所有订单的所有信息

public Iterator<Result> queryOrdersByBuyer(long startTime, long endTime,

String buyerid);

//查询某位卖家某件商品所有订单的某些字段

public Iterator<Result> queryOrdersBySaler(String salerid, String goodid,

Collection<String> keys);

//对某件商品的某个字段求和，只允许对long和double类型的KV求和 如果字段中既有long又有double，则使用double如果求和的key中包含非long/double类型字段，则返回null 如果查询订单中的所有商品均不包含该字段，则返回null

public KeyValue sumOrdersByGood(String goodid, String key);

1. **Tester** 交易订单系统接口实现测试类。

//测试方法

public static void test(OrderSystem orderSystem)；

// 根据 br读取流数据查询商品信息。通过读取销售id和商品id，成功返回true，否则为false

public static boolean parseSalerGood(BufferedReader br,OrderSystem orderSystem)；

//根据订单id，查询订单，成功返回true，否则返回false

public static boolean parseOrder(BufferedReader br,OrderSystem orderSystem)；

//根据时间范围查询买家信息，标志位开始时间和结束时间

public static boolean parseBuyerTsRange(BufferedReader br,OrderSystem orderSystem)；

//计算商品数量总和

public static boolean parseGoodSum(BufferedReader br,OrderSystem orderSystem)；

1. **Util**文件夹

此文件下下包含OrderUtil、PrintUtil、RankUtil、RecordCreator、TypeUtil。以下对每个类的功能做简要分析。

OrderUtil：解析订单的工具类

PrintUtil：打印输出工具类

RankUtil：比较、根据索引查找孩子节点、根据key和range查找返回结果

RecordCreator：创建对象、读写消息流的类

TypeUtil： 变量类型工具类

1. **Table**文件夹

BuyerTable：构建买家表，实现记录买家信息，建立索引、实现构建买家任务多线程操作。

public BuyerTable(Collection<String> buyerFiles, Collection<String> originStoreFolders)；

public class BuildTask implements Runnable{}

public void buildIndex()throws IOException,InterruptedException；

1. **Store**

BTree：

为简化实现,定有以下约定

\* 约定1:key与data的长度固定

\* 约定2:每个node只用一个page保存

\* 约定3:内节点全部加载到内存中,叶节点只保存指针,具体的内容存在叶节点map中,并实行淘汰策略

\* 约定4:内节点中的关键字不重复,叶节点中的关键字可以少量重复

//节点分裂模式

public enum SplitMode{

LEFT,MID,RIGHT;

}

//内存中的

private AtomicLong leafIndex = new AtomicLong(System.currentTimeMillis());

public PageMgr pageMgr; //磁盘页管理

public BNode root; //根节点

//B树节点,尤其是叶子节点,大部分都放在磁盘中, 这里保存映射关系,如果不停地刷新磁盘,则效率会比较低

public Map<Long,BNode> leafCache = new LinkedHashMap<Long, BNode>()；

//默认建的树直接存储在内存中

public BTree(){}

BTreeV2：与BTree第一版相比, 主要在于写模式的缓冲。

DirectBuffMgr：堆外内存的受gc影响,释放不可控,这里自己动手管理

FlushHelper：多线程实现任务刷新

LruCache：基于LinkedHashMap实现一个简单的lrucache

OriginStore：存储原始内容,以换行符分割

Page：

内存和磁盘交换的基本单位

\* 约定1: 所有内容采取覆盖制,前面4个字节,表示该page写入的内容的长度

\* 约定2: 调用方自己控制内容的长度,如果这里发现内容超过page的范围,则自动截断,读和写均截断

PageMgr：

page的管理器

\* 理论上来说,可以设计成跨文件的,但是这里简化成单个文件;

\* 约定1: 单个对象管理的page都在一个文件中

\* 约定2: 第一个page,用来记录文件的page分配状况,如第一个page在哪个位置,总共分配了多少个page,分配的位置点,删除的page等

\* 约定3: 按顺序依次申请page,但在逻辑上应该理解为page链表,而不是page数组

SimpleBTree：

简单的B树,约定如下:

\* 1 每颗B树可以完整加载到内存中

\* 2 每棵B树只能构建一次,中途插入的成本很高

\* 3 先按插入顺序存储到文件中,待插入结束,从文件中读出所有数据在内存中排序后再插入

StoreConfig：存储配置工具

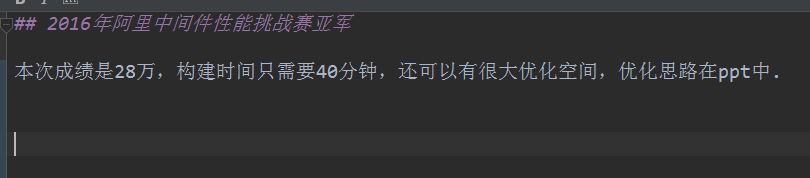
1. **Model** 模板类

Constant： 常量字段的定义，使用静态方法

GoodOrderKey： 商品、订单关联键值字段构建添加

三、实验结果

1.预期结果



1. 图表分析结果



因为随机读取比较特殊，每次读取和上一次的位置间隔较大，所以根据IOPS特性，在随机读取阶段尽可能的读取最少的数据。此处我们选取了1K，如果在预知数据大小的情况下尽可能的小，减少IO操作次数。

1. 总结

对Java的使用有更深的认识，在多线程方面，同步的影响，合理使用Atomic包，消除同步。写代码时注意内存使用，合理的复用对象，不用的对象及时清除。了解常用的类源码，知道底层实现，根据场景进行合理的选择。