# Hadoop应用开发实战案例 第3周

# 法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

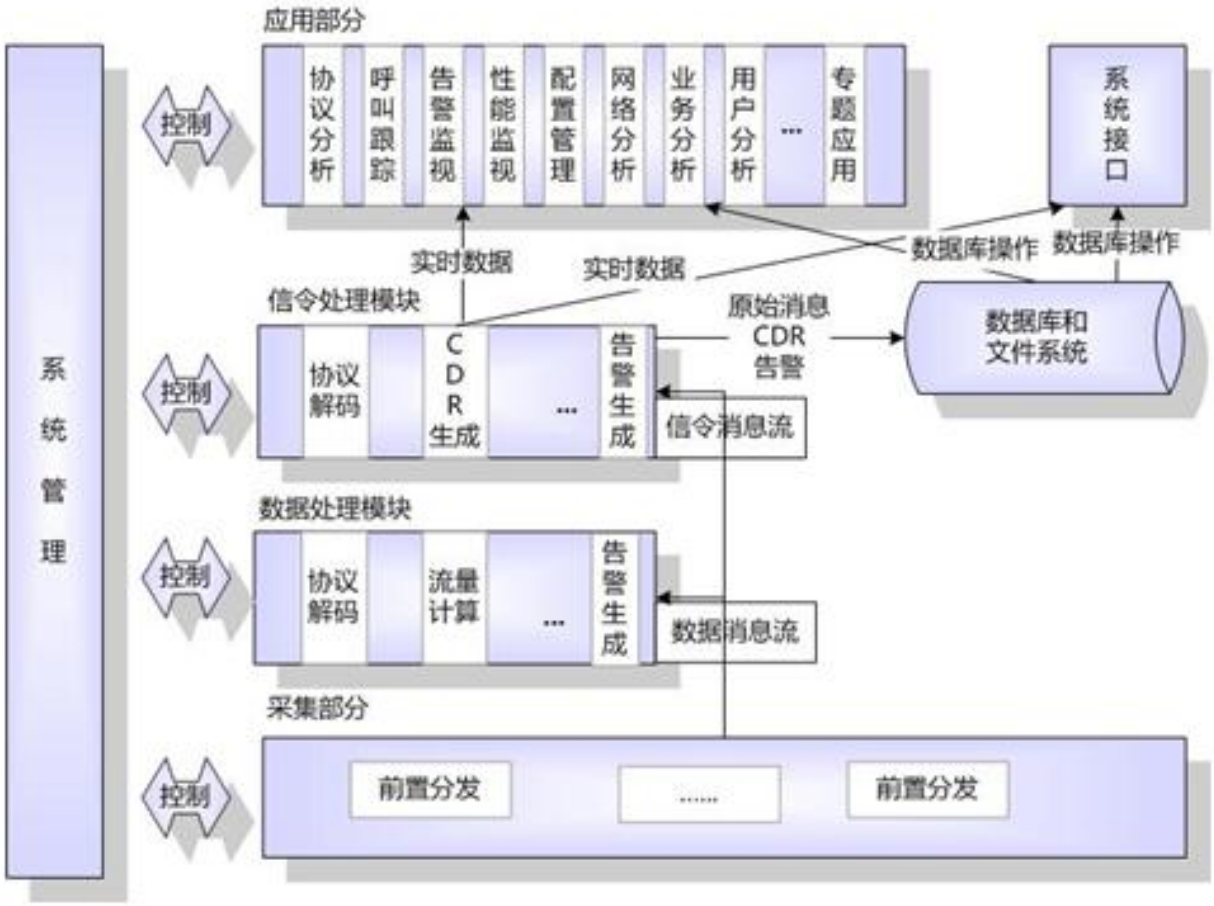
# 运营商那些事儿

- 运营商数据分析的现状
- 运营商感兴趣的分析主题
- 运营商对Hadoop的看法



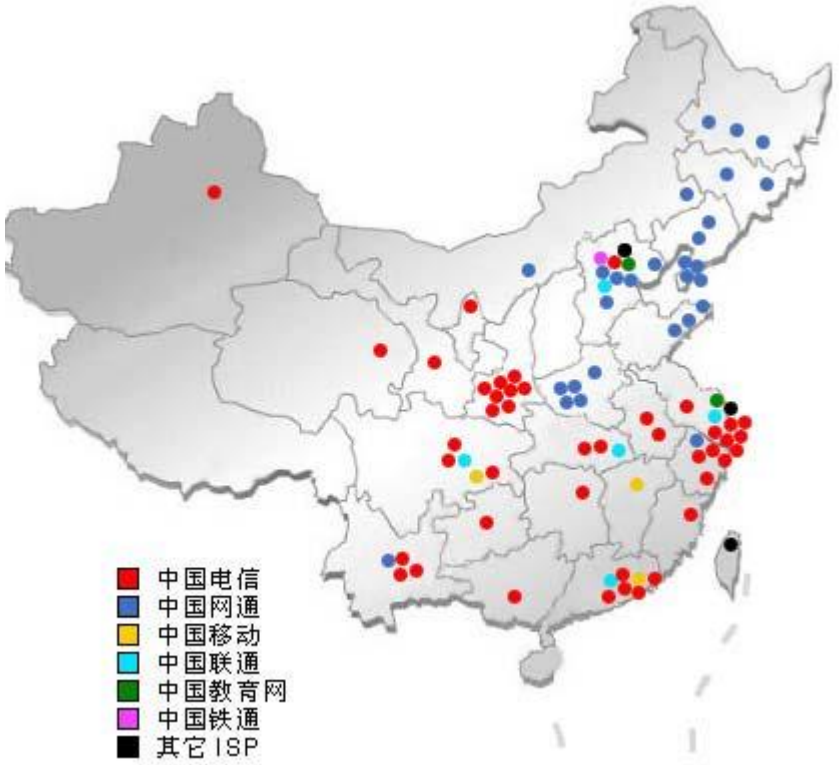



# 典型架构


应用部分
协议分析
呼叫跟踪
告警监视
性能监视
配置管理
网络分析
业务分析
用户分析
...
专题应用
系统接口
控制
实时数据
实时数据
数据库操作
数据库操作
信令处理模块
协议解码
CDR生成
...
告警生成
信令消息流
原始消息
CDR
告警
数据库和文件系统
数据处理模块
协议解码
流量计算
...
告警生成
数据消息流
采集部分
前置分发
.....
前置分发
系统管理

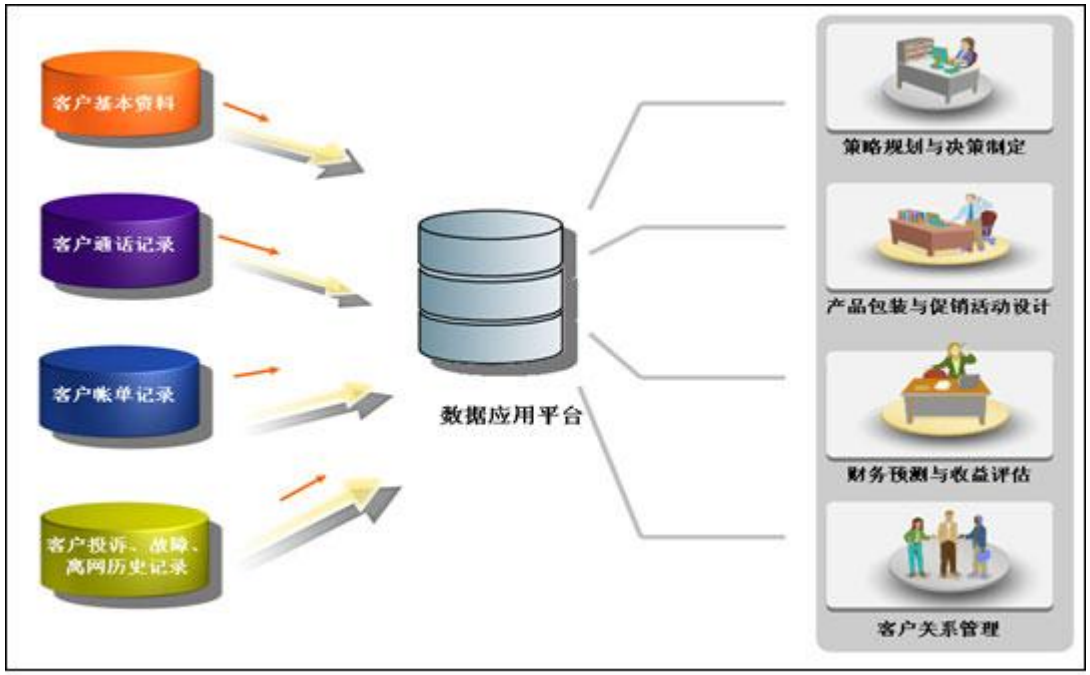
# 做过的项目

- 全国性项目，19省市节点
- 省级移动公司数据分析
- 地区级公司数据分析
- 产品型项目
- 即席型项目
- 网分，网监，经分，外包



# 分析主题

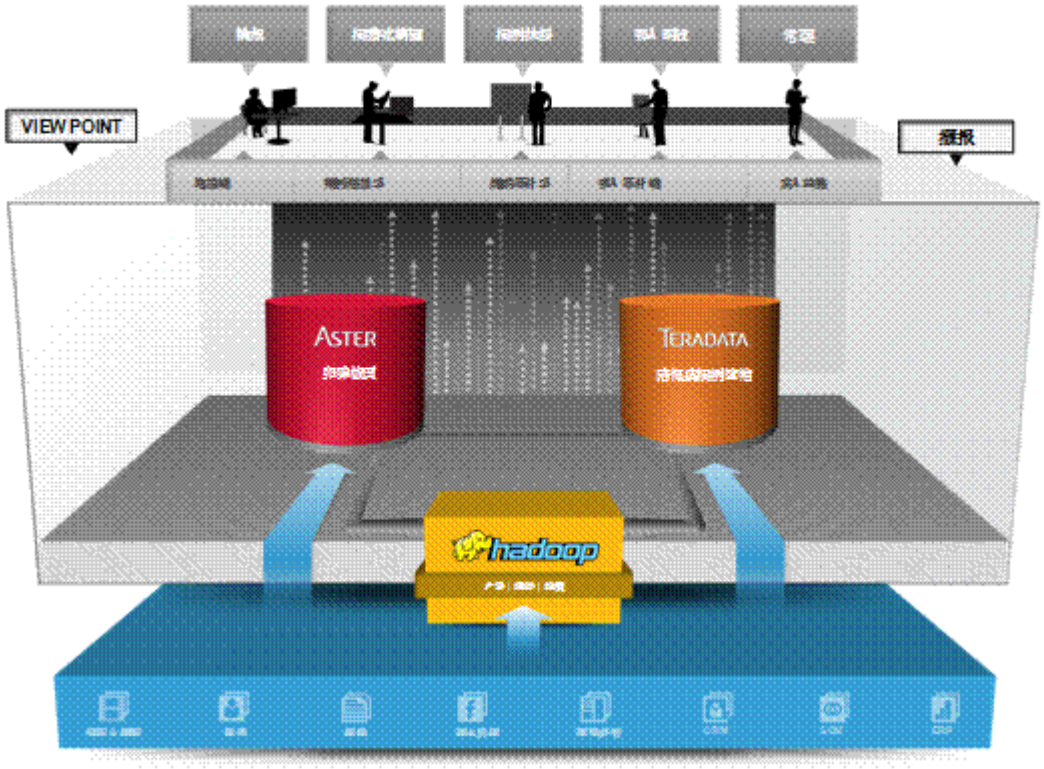
- 回拨分析
- 养卡分析
- 标签系统
- 流失预警
- 位置应用
- 目前很多分析问题不少，但正在改善中
- 极端需要有创意能产生极大经济效益的分析主题




客户流失与挽留的建设架构图

# 对Hadoop的看法

- 喜欢高档的厂商产品
- 抗拒开源产品
- 正在改变



# 什么是基站数据

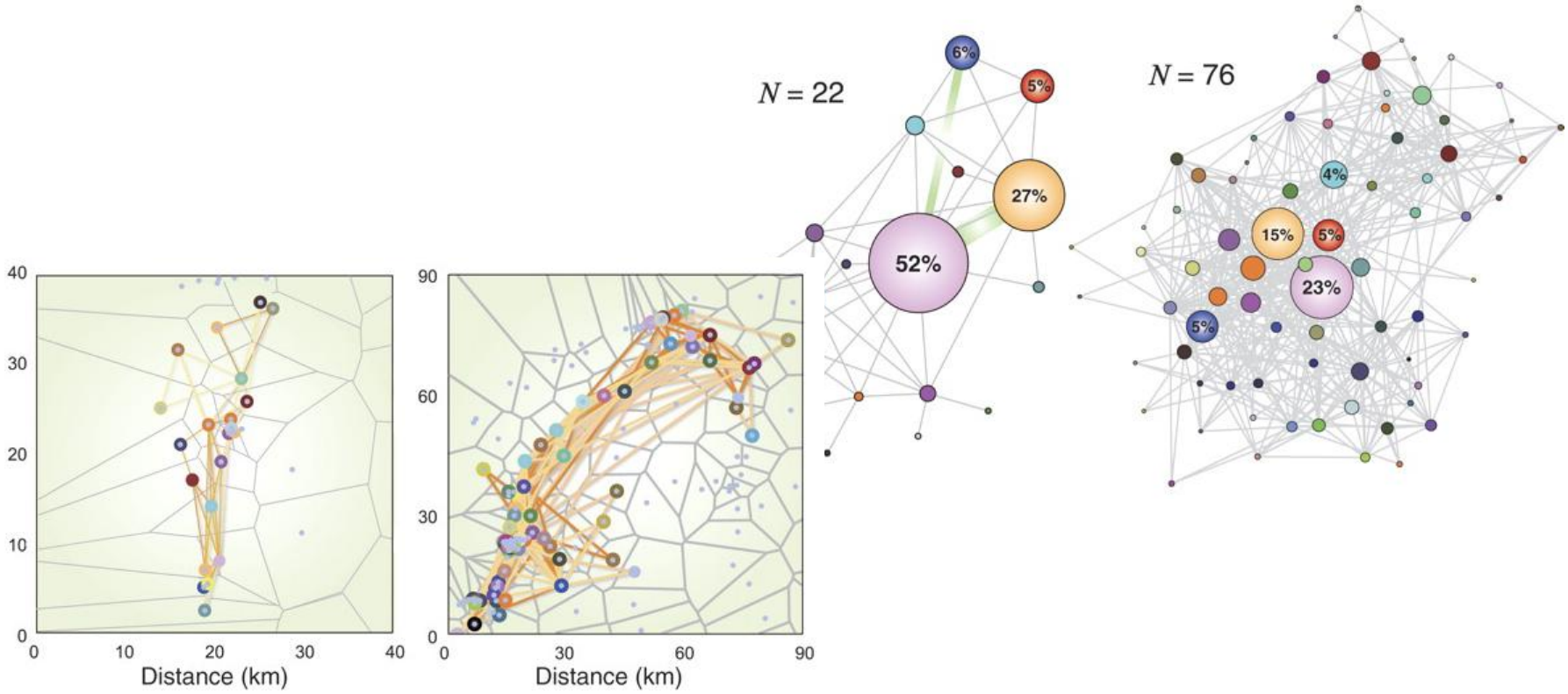
# 通信基站产生的数据



| | |
|---|---|
| 呼叫数据 | 每一次呼叫产生一条呼叫记录 |
| | 区分主叫/被叫, 记录对方号码 |
| 短信数据 | 每一条短信产生一条短信记录 |
| | 区分发送/接收, 记录对方号码 |
| 位置更新/开关机 | 每次开机和关机都产生一条记录 |
| | 手机将会周期性和基站进行通信 |
| 上网记录 | 区分3G/2G |
| | 终端/IP/URL |
| 其它 | 位置切换 |
| | 漫游数据 |

# 记录每位用户的移动轨迹






*Barabasi. 2010*

# 应用前景



## 通过用户移动轨迹识别客户身份

- 在没有客户身份数据的情况下
- 认定上课在学校附近出现, 而且上课时间通话很少的用户作为疑似学生

## 分析轨迹流量

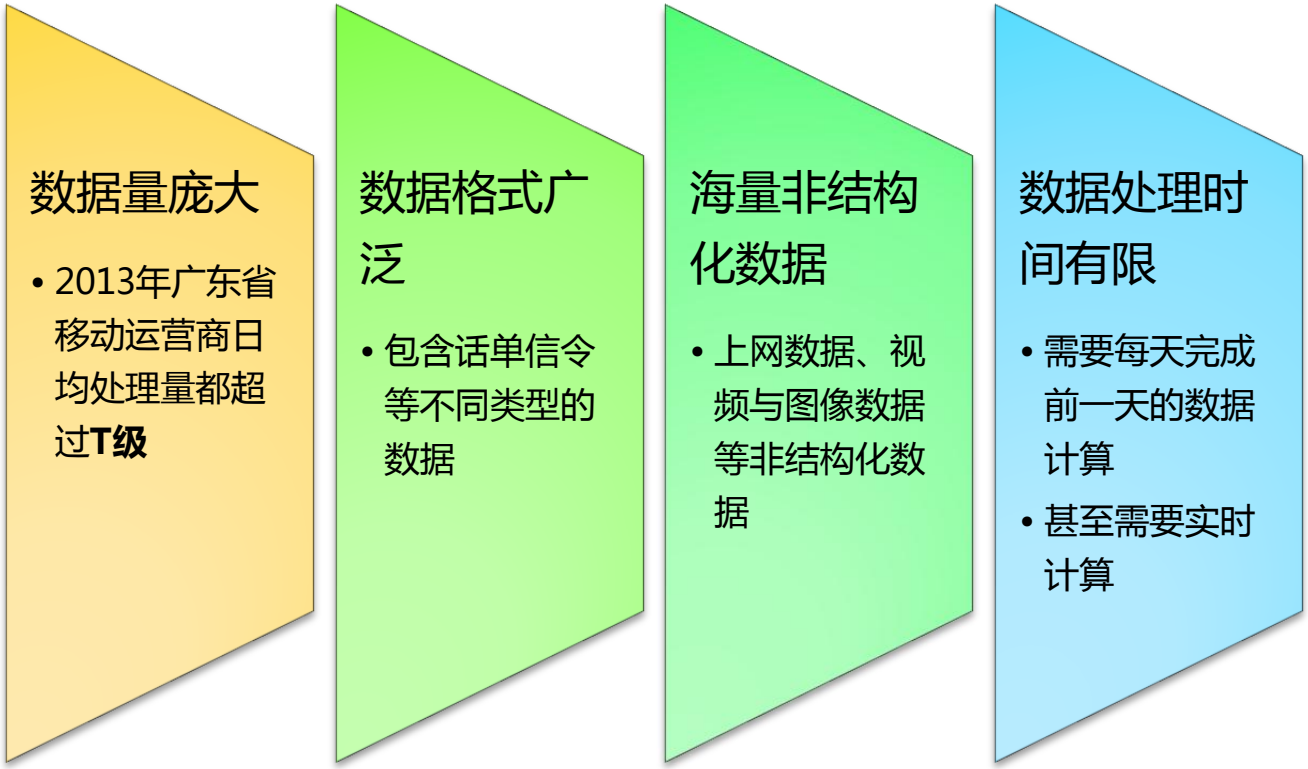
- 分析目标地点的用户量和用户类型, 为户外广告选址和促销提供决策支持

## 决策支撑

- 分析土地利用情况，为政府的土地分配和使用作决策支撑

# 数据处理的挑战




数据量庞大
• 2013年广东省移动运营商日均处理量都超过T级
数据格式广泛
• 包含话单信令等不同类型的数据
海量非结构化数据
• 上网数据、视频与图像数据等非结构化数据
数据处理时间有限
• 需要每天完成前一天的数据计算
• 甚至需要实时计算

# 数据解决思路



架设Hadoop分布式计算平台, 搭建数据存储体系

对数据进行初步的分析和统计

只保留用户的活动位置数据

如有必要导入结构化数据库进行分析


Source: http://www.zdnet.com/uk/mobile-big-data-cloud-m2m-7000015484/

# 案例需求



**500万客户**

- 每天接收20G的数据

**数据分两种类型**

- 上网数据
- 位置数据

**只存储每个用户在不同时段停留最长的三个位置**

- 记录（用户ID, 时间段, 地点）

**把一天分割成不同的时段**

- 区分凌晨, 上午通勤时间, 工作时间, 下午回家时间和晚上
- 可以根据需求更改时间段的划分

# 问题1

- 为什么要选择“3”作为保留基站位置数目？
  1. 领导说选 3
  2. 分析用户停留最长的前 1/2/3… 个基站的数据量的和占数据总量的平均比例
  3. 因为本选项是第三个

# 解决方案


eclipse
软件开发
•Eclipse 测试程序
数据导出
•Sqoop
cloudera
Ask Bigger Question
数据传入
•T+1, 隔日接收前日数据
ORACLE
软件平台
•Cloudera 的 CDH3u6

# 数据格式



位置数据

| IMSI | IMEI | UPDATETYPE | LOC | TIME |
|---|---|---|---|---|
| … | | | | |
| A | 001 | 0 | X基站 | 2013-09-12 09:00:00 |
| A | 001 | 2 | Y基站 | 2013-09-12 09:45:00 |
| … | | | | |

两种数据最大的差别在于文件名

数据文件名以POS开头

数据文件名以NET开头

上网数据

| IMSI | IMEI | LOC | TIME | URL |
|---|---|---|---|---|
| … | | | | |
| A | 001 | X基站 | 2013-09-12 09:15:00 | www.baidu.com |
| A | 001 | Y基站 | 2013-09-12 09:30:00 | www.google.com |
| … | | | | |

# 计算规则



认为用户在任何时间的停留位置都取决于之前一次位置更新的基站位置

时间间隔超过超过60分钟的判定为关机

| IMSI | IMEI | UPDATETYPE | LOC | TIME |
|---|---|---|---|---|
| … | | | | |
| A | 001 | 0 | X基站 | 2013-09-12 09:00:00 |
| A | 001 | 2 | Y基站 | 2013-09-12 09:45:00 |
| … | | | | |

| IMSI | IMEI | LOC | TIME | URL |
|---|---|---|---|---|
| … | | | | |
| A | 001 | X基站 | 2013-09-12 09:15:00 | www.baidu.com |
| A | 001 | Y基站 | 2013-09-12 09:30:00 | www.google.com |
| … | | | | |

用户A在X基站停留了30分钟

# 算法流程 - Mapper

输入数据

| IMSI | IMEI | UPDATETYPE | LOC | TIME |
|---|---|---|---|---|
| … | | | | |
| A | 001 | 0 | X基站 | 2013-09-12 09:00:00 |
| A | 001 | 2 | Y基站 | 2013-09-12 09:45:00 |
| … | | | | |

| IMSI | IMEI | LOC | TIME | URL |
|---|---|---|---|---|
| … | | | | |
| A | 001 | X基站 | 2013-09-12 09:15:00 | www.baidu.com |
| A | 001 | Y基站 | 2013-09-12 09:30:00 | www.google.com |
| … | | | | |

根据文件名提取字段

| IMSI | LOC | TIME |
|---|---|---|
| … | | |
| A | X基站 | 2013-09-12 09:00:00 |
| A | Y基站 | 2013-09-12 09:45:00 |
| … | | |

| IMSI | LOC | TIME |
|---|---|---|
| … | | |
| A | X基站 | 2013-09-12 09:15:00 |
| A | Y基站 | 2013-09-12 09:30:00 |
| … | | |

计算时间所属时间段
把日期转换为UNIX格式

| IMSI | LOC | TimeFlag | TIME |
|---|---|---|---|
| … | | | |
| A | X基站 | 09-17 | 1386579600 |
| A | Y基站 | 09-17 | 1386582300 |
| A | X基站 | 09-17 | 1386580500 |
| A | Y基站 | 09-17 | 1386581400 |
| … | | | |

# 算法流程 - Mapper

| IMSI | TimeFlag |
|---|---|
| ... | |
| A | 09-17 |
| A | 09-17 |
| A | 09-17 |
| A | 09-17 |
| ... | |

| LOC | TIME |
|---|---|
| | |
| X基站 | 1386579600 |
| Y基站 | 1386582300 |
| X基站 | 1386580500 |
| Y基站 | 1386581400 |
| | |

Map

输出

| IMSI | LOC | TIME |
|---|---|---|
| ... | | |
| A | X基站 | 2013-09-12 09:15:00 |
| A | Y基站 | 2013-09-12 09:45:00 |
| ... | | |

| IMSI | LOC | TIME |
|---|---|---|
| ... | | |
| A | X基站 | 2013-09-12 11:15:00 |
| A | Y基站 | 2013-09-12 09:30:00 |
| ... | | |

以IMSI和TimeFlag作为Key
以LOC和TIME作为VALUE

| IMSI | LOC | TimeFlag | TIME |
|---|---|---|---|
| ... | | | |
| A | X基站 | 09-17 | 1386579600 |
| A | Y基站 | 09-17 | 1386582300 |
| A | X基站 | 09-17 | 1386580500 |
| A | Y基站 | 09-17 | 1386581400 |
| ... | | | |

1. 计算时间所属时间段
2. 把日期转换为UNIX格式

# 算法流程 - Reducer

| IMSI | TimeFlag |
|---|---|
| … | |
| A | 09-17 |
| A | 09-17 |
| A | 09-17 |
| A | 09-17 |
| … | |

| LOC | TIME |
|---|---|
| | |
| X基站 | 1386579600 |
| Y基站 | 1386582300 |
| X基站 | 1386580500 |
| Y基站 | 1386581400 |
| | |

具有相同Key的数据发送到同一Reducer

| IMSI | TimeFlag |
|---|---|
| … | |
| 其他 | 其他 |
| … | |

| LOC | TIME |
|---|---|
| | |
| …… | …… |
| | |

| IMSI | TimeFlag |
|---|---|
| … | |
| A | 09-17 |
| … | |

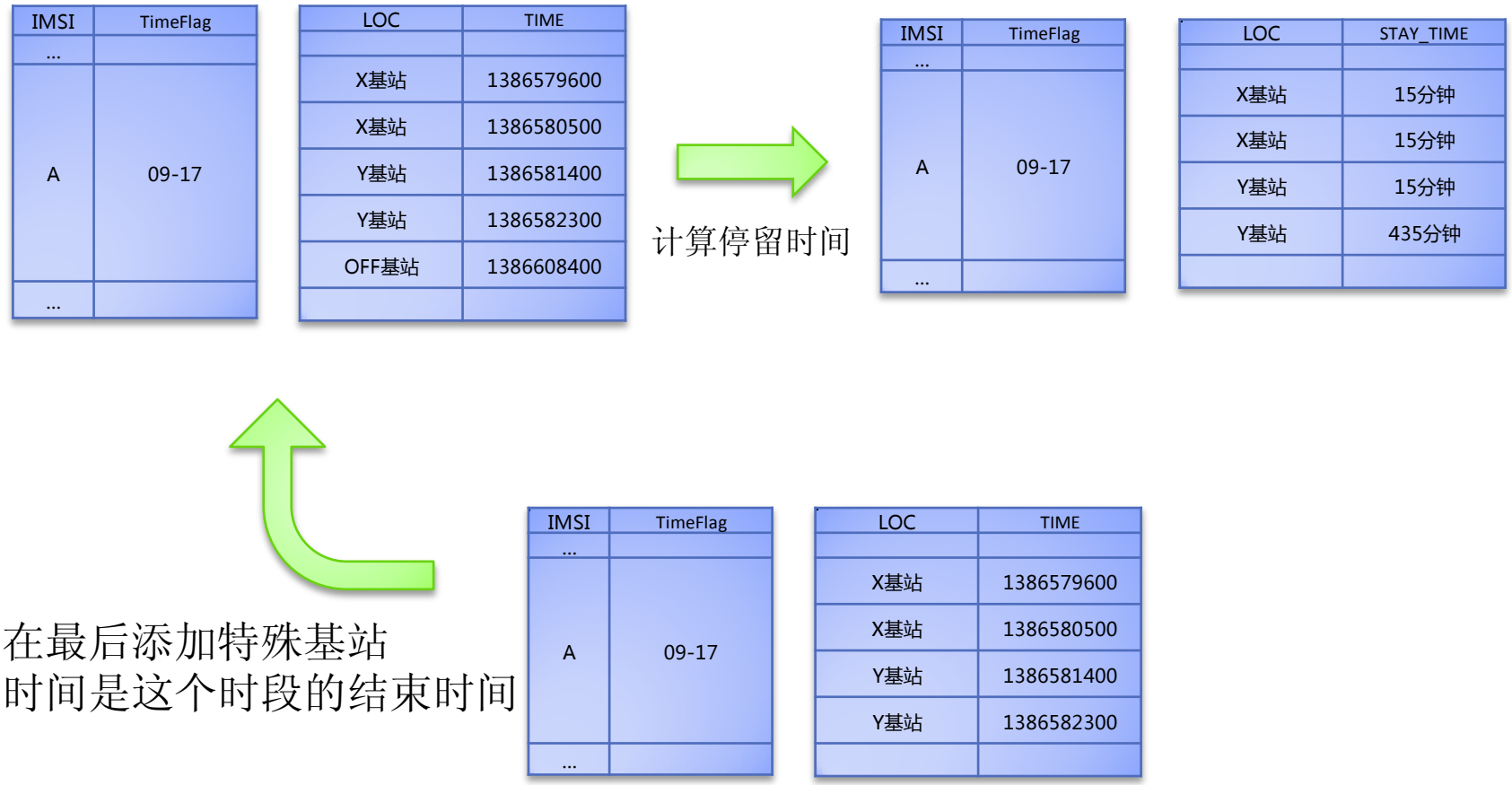
| LOC | TIME |
|---|---|
| | |
| X基站 | 1386579600 |
| Y基站 | 1386582300 |
| X基站 | 1386580500 |
| Y基站 | 1386581400 |
| | |

| IMSI | TimeFlag |
|---|---|
| … | |
| A | 09-17 |
| … | |

| LOC | TIME |
|---|---|
| | |
| X基站 | 1386579600 |
| X基站 | 1386580500 |
| Y基站 | 1386581400 |
| Y基站 | 1386582300 |
| | |

对时间排序

# 算法流程 - Reducer



# 算法流程 - Reducer

| IMSI | TimeFlag |
| --- | --- |
| ... | |
| A | 09-17 |
| ... | |

| LOC | TIME |
| --- | --- |
| | |
| X基站 | 1386579600 |
| X基站 | 1386580500 |
| Y基站 | 1386581400 |
| Y基站 | 1386582300 |
| OFF基站 | 1386608400 |
| | |

计算停留时间

| IMSI | TimeFlag |
| --- | --- |
| ... | |
| A | 09-17 |
| ... | |

| LOC | STAY_TIME |
| --- | --- |
| | |
| X基站 | 15分钟 |
| X基站 | 15分钟 |
| Y基站 | 15分钟 |
| Y基站 | 435分钟 |
| | |

删除大于60分钟的
汇总统计各个基站

| IMSI | TimeFlag |
| --- | --- |
| ... | |
| A | 09-17 |
| ... | |

| LOC | STAY_TIME |
| --- | --- |
| | |
| X基站 | 30分钟 |
| Y基站 | 15分钟 |
| | |

输出数据

# 程序概览

程序分四个主体部分

```
/**
 * 汇总基站数据表
 * 计算每个用户在不同的时间段不同的基站停留的时长
 * 输入参数 < input path > < output path > < date > < timepoint >
 * 参数示例： “/base /output 2012-09-12 09-17-24"
 * 意味着以“/base”为输入，"/output"为输出，指定计算2012年09月12日的数据，并分为00-07，07-17，17-24三个时段
 * 输出格式 “IMSI|CGI|TIMFLAG|STAY_TIME”
 */
public class BaseStationDataPreprocess extends Configured implements Tool
{
    /**
     * 计数器
     */
    enum Counter {
    }

    /**
     * 读取一行数据
     * 以“IMSI+时间段”作为 KEY 发射出去
     */
    public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    }

    /**
     * 统计同一个IMSI在同一时间段
     * 在不同CGI停留的时长
     */
    public static class Reduce extends Reducer<Text, Text, NullWritable, Text> {
    }

    public int run(String[] args) throws Exception {
    }

    public static void main(String[] args) throws Exception {
    }
```

Map部分

Reduce部分

任务提交，参数传递

程序执行

# Mapper – 概览

Mapper重写两个函数

```
public class BaseStationDataPreprocess extends Configured implements Tool
{
     * 计数器
    enum Counter

    /**
     * 读取一行数据
     * 以“IMSI+时间段”作为 KEY 发射出去
     */
    public static class Map extends Mapper<LongWritable, Text, Text, Text>
    {
        String date;
        String [] timepoint;
        boolean dataSource;

         * 初始化
        public void setup ( Context context ) throws IOException

         * MAP任务
        public void map ( LongWritable key, Text value, Context context ) throws
    }
```

Setup函数, 每一个Mapper开始的时候执行一次

Map函数, 对每一行输入数据执行一次

# 问题2

- Setup函数和Map函数的主要差别是什么？
  1. 功能上的不同，Setup函数是用来传递参数的，而Map函数是用来执行任务的
  2. 效率上的不同，一个Mapper中Setup只会执行一次，Map函数每处理一行都回执行一次
  3. 很明显名字就不一样嘛

# Mapper – Setup函数

Setup函数用于初始化参数

```
public static class Map extends Mapper<LongWritable, Text, Text, Text>
{
    String date;
    String [] timepoint;
    boolean dataSource;

     * 初始化
    public void setup ( Context context ) throws IOException
    {
        this.date = context.getConfiguration().get("date");
        this.timepoint = context.getConfiguration().get("timepoint").split("-");

        //提取文件名
        FileSplit fs = (FileSplit)context.getInputSplit();
        String fileName = fs.getPath().getName();
        if( fileName.startsWith("POS") )
            dataSource = true;
        else if ( fileName.startsWith("NET") )
            dataSource = false;
        else
            throw new IOException("File Name should starts with POS or NET");
    }
```

Setup函数, 每一个Mapper开始的时候执行一次

提取参数

从Context获取文件名
区分数据来源和字段

# Mapper – Map函数

Map函数对每一行输入数据进行处理

```
/**
 * MAP任务
 * 读取基站数据
 * 找出数据所对应时间段
 * 以IMSI和时间段作为 KEY
 * CGI和时间作为 VALUE
 */
public void map ( LongWritable key, Text value, Context context ) throws IOException
{
    String line = value.toString();
    TableLine tableLine = new TableLine();

    //读取行
    try
    {
        tableLine.set(line, this.dataSource, this.date, this.timepoint );
    }
    catch ( LineException e )
    {
        if(e.getFlag()==-1)
            context.getCounter(Counter.OUTOFTIMESKIP).increment(1);
        else
            context.getCounter(Counter.TIMESKIP).increment(1);
        return;
    }
    catch (Exception e)
```

读取一行数据

自定义 TableLine 类提取字段

# Mapper – TableLine类

TableLine类能够提取一行数据的字段并进行转换

```
/**
 * 读取一行数据
 * 提取所要字段
 */
public class TableLine
{
    private String imsi, position, time, timeFlag;
    private Date day;
    private SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    /**
     * 初始化并检查该行的合法性
     */
    public void set ( String line, boolean source, String date, String [] timepoint ) th

    /**
     * 输出KEY
     */
    public Text outKey()

    /**
     * 输出VALUE
     */
    public Text outValue()
}
```

提取字段

设置输出Key

设置输出Value

# Mapper – TableLine类

Set函数利用输入的数据初始化变量

```
public void set ( String line, boolean source, String date, String [] timepoint )
{
    String [] lineSplit = line.split("\t");
    if( source )
    {
        this.imsi = lineSplit[0];
        this.position = lineSplit[3];
        this.time = lineSplit[4];
    }
    else
    {
        this.imsi = lineSplit[0];
        this.position = lineSplit[2];
        this.time = lineSplit[3];
    }
```

根据数据源提取字段

# Mapper – TableLine类

对字段做合法性检查

```
//检查日期合法性
if ( ! this.time.startsWith(date) )          //年月日必须与date一致
    throw new LineException("", -1);

try
{
    this.day = this.formatter.parse(this.time);
}
catch ( ParseException e )
{
    throw new LineException("", 0);
}
```

对日期的合法性做检查

把不同情况的异常分类统计

定义异常类
捕获异常情况

```
/**
 * 定义异常类
 */
class LineException extends Exception
{
    private static final long serialVersionUID = 8245008693589452584L;
    int flag;
    public LineException(String msg, int flag)
    {
        super(msg);
        this.flag = flag;
    }
    public int getFlag()
    {
        return flag;
    }
}
```

# Mapper – TableLine类

根据时间字段计算所属的时间段

```
//计算所属时间段
int i = 0, n = timepoint.length;
int hour = Integer.valueOf( this.time.split(" ")[1].split(":")[0] );
while ( i < n && Integer.valueOf( timepoint[i] ) <= hour )
    i++;
if ( i < n )
{
    if ( i == 0 )
        this.timeFlag = ( "00-" + timepoint[i] );
    else
        this.timeFlag = ( timepoint[i-1] + "-" + timepoint[i] );
}
else                                        //Hour大于最大的时间点
    throw new LineException("", -1);
```

# Mapper – TableLine类

设置输出的KEY和VALUE

```
 * 输出KEY
public Text outKey()
{
    return new Text ( this.imsi + "|" + this.timeFlag );
}

 * 输出VALUE
public Text outValue()
{
    long t = ( day.getTime() / 1000L );                    //用时间的偏移量作为输出时间
    return new Text ( this.position + "|" + String.valueOf(t) );
}
```

把时间转换成UNIX格式

# Mapper – Map函数

把异常情况记录到计数器中

```
public void map ( LongWritable key, Text value, Context context ) throws I
{
    String line = value.toString();
    TableLine tableLine = new TableLine();

    //读取行
    try
    {
        tableLine.set(line, this.dataSource, this.date, this.timepoint );
    }
    catch ( LineException e )
    {
        if(e.getFlag()==-1)
            context.getCounter(Counter.OUTOFTIMESKIP).increment(1);
        else
            context.getCounter(Counter.TIMESKIP).increment(1);
        return;
    }
    catch (Exception e)
    {
        context.getCounter(Counter.LINESKIP).increment(1);
        return;
    }

    context.write( tableLine.outKey(), tableLine.outValue() );
}
```

对各种情况修改计数器

# Mapper – Map函数

把异常情况记录到计数器中

```
public void map ( LongWritable key, Text value, Context context ) throws I
{
    String line = value.toString();
    TableLine tableLine = new TableLine();

    //读取行
    try
    {
        tableLine.set(line, this.dataSource, this.date, this.timepoint );
    }
    catch ( LineException e )
    {
        if(e.getFlag()==-1)
            context.getCounter(Counter.OUTOFTIMESKIP).increment(1);
        else
            context.getCounter(Counter.TIMESKIP).increment(1);
        return;
    }
    catch (Exception e)
    {
        context.getCounter(Counter.LINESKIP).increment(1);
        return;
    }

    context.write( tableLine.outKey(), tableLine.outValue() );
}
```

对各种情况修改计数器

# Reducer – 概览

对具有相同IMSI以及时间段的数据进行汇总处理

```
/**
 * 统计同一个IMSI在同一时间段
 * 在不同CGI停留的时长
 */
public static class Reduce extends Reducer<Text, Text, NullWritable, Text>
{
    private String date;
    private SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

     * 初始化
    public void setup ( Context context )

    public void reduce ( Text key, Iterable<Text> values, Context context ) throws IOExc

    /**
     * 获得位置停留信息
     * @param timeToCGI
     * @return
     */
    private HashMap<String, Float> getStayTime(TreeMap<Long, String> uploads)

}
```

Setup函数, 每一个Mapper开始的时候执行一次

Map函数, 对每一行输入数据执行一次

自定义函数, 用于汇总停留时间

# Reducer – Reduce函数

提取具有相同KEY的数据并对其按时间排序

```
public void reduce ( Text key, Iterable<Text> values, Context context ) throws IOException, Interrupt
{
    String imsi = key.toString().split("\\|")[0];
    String timeFlag = key.toString().split("\\|")[1];

    //用一个TreeMap记录时间
    TreeMap<Long, String> uploads = new TreeMap<Long, String>();
    String valueString;

    for ( Text value : values )
    {
        valueString = value.toString();
        try
        {
            uploads.put( Long.valueOf( valueString.split("\\|")[1] ), valueString.split("\\|")[0] );
        }
        catch ( NumberFormatException e )
        {
            context.getCounter(Counter.TIMESKIP).increment(1);
            continue;
        }
    }
```

使用 TreeMap 可以
让数据按照时间排序

时间

地点

# Reducer – Reduce函数

计算停留时间并输出数据

```
try
{
    //在最后添加“OFF”位置
    Date tmp = this.formatter.parse( this.date + " " + timeFlag.split("-")[1] + ":00:00" );
    uploads.put ( ( tmp.getTime() / 1000L ), "OFF");

    //汇总数据
    HashMap<String, Float> locs = getStayTime(uploads);

    //输出
    for( Entry<String, Float> entry : locs.entrySet() )
    {
        StringBuilder builder = new StringBuilder();
        builder.append(imsi).append("|");
        builder.append(entry.getKey()).append("|");
        builder.append(timeFlag).append("|");
        builder.append(entry.getValue());

        context.write( NullWritable.get(), new Text(builder.toString()) );
    }
}
```

调用自定义函数
统计停留时间

由于输出数据
要用'|'作为分隔符
把输出的KEY
设为 NullWritable

# Reducer – StayTime函数

统计停留时间

```
 * 获得位置停留信息
private HashMap<String, Float> getStayTime(TreeMap<Long, String> uploads)
{
    Entry<Long, String> upload, nextUpload;
    HashMap<String, Float> locs = new HashMap<String, Float>();
    //初始化
    Iterator<Entry<Long, String>> it = uploads.entrySet().iterator();
    upload = it.next();
    //计算
    while( it.hasNext() )
    {
        nextUpload = it.next();
        float diff = (float) (nextUpload.getKey()-upload.getKey()) / 60.0f;
        if( diff <= 60.0 )                                    //时间间隔过大则代表关机
        {
            if( locs.containsKey( upload.getValue() ) )
                locs.put( upload.getValue(), locs.get(upload.getValue())+diff );
            else
                locs.put( upload.getValue(), diff );
        }
        upload = nextUpload;
    }
    return locs;
}
```

# 程序执行 – Main函数

Main函数检查参数数量是否正确并决定是否执行

检查参数个数是否正确

```
public static void main(String[] args) throws Exception
{
    if ( args.length != 4 )
    {
        System.err.println("");
        System.err.println("Usage: BaseStationDataPreprocess < input path > < output path > < date > < timepoint >");
        System.err.println("Example: BaseStationDataPreprocess /user/james/Base /user/james/Output 2012-09-12 07-09-17-24");
        System.err.println("Warning: Timepoints should be begined with a 0+ two digit number and the last timepoint should be 24");
        System.err.println("Counter:");
        System.err.println("\t"+"TIMESKIP"+"\t"+"Lines which contain wrong date format");
        System.err.println("\t"+"OUTOFTIMESKIP"+"\t"+"Lines which contain times that out of range");
        System.err.println("\t"+"LINESKIP"+"\t"+"Lines which are invalid");
        System.err.println("\t"+"USERSKIP"+"\t"+"Users in some time are invalid");
        System.exit(-1);
    }

    //运行任务
    int res = ToolRunner.run(new Configuration(), new BaseStationDataPreprocess(), args);

    System.exit(res);
}
```

调用Run函数执行任务

# 程序执行 – Run函数

Main函数检查参数数量是否正确并决定是否执行

```
@Override
public int run(String[] args) throws Exception {
    Configuration conf = getConf();

    conf.set("date", args[2]);
    conf.set("timepoint", args[3]);

    Job job = new Job(conf, "BaseStationDataPreprocess");
    job.setJarByClass(BaseStationDataPreprocess.class);

    FileInputFormat.addInputPath( job, new Path(args[0]) );          //输入路径
    FileOutputFormat.setOutputPath( job, new Path(args[1]) );        //输出路径

    job.setMapperClass( Map.class );                                 //调用上面Map类作为Map任务代码
    job.setReducerClass ( Reduce.class );                            //调用上面Reduce类作为Reduce任务代码
    job.setOutputFormatClass( TextOutputFormat.class );
    job.setOutputKeyClass( Text.class );
    job.setOutputValueClass( Text.class );

    job.waitForCompletion(true);

    return job.isSuccessful() ? 0 : 1;
}
```

传递参数给任务

任务执行命令

# 上传数据

- 上传数据命令：

Hadoop fs –put 本地文件路径 集群文件路径

```
james@Cluster-0:~/Documents/Data$ cat NETWORK
0000000001      4600000000000000        0000-0001       2013-09-12 09:45:00     www.google.com
0000000001      4600000000000000        0000-0002       2013-09-12 10:40:00     scholar.google.com
0000000001      4600000000000000        0000-0001       0000-09-00 10:40:00     xzvf
0000000001      4600000000000000        0000-0001       0000-09-00 10:40:00     xzvf
james@Cluster-0:~/Documents/Data$ cat POSITION
0000000001      4600000000000000        0       0000-0001       2013-09-12 09:15:00
0000000001      4600000000000000        2       0000-0002       2013-09-12 09:30:00
0000000001      4600000000000000        0       0000-0001       2013-09-12 10:00:00
0000000001      4600000000000000        1       0000-0002       2013-09-12 11:45:00
james@Cluster-0:~/Documents/Data$ hadoop fs -put ~/Documents/Data/ /user/james/Base
```

NETWORK文件
有两行数据日期有误

# 程序执行

- 程序运行命令：

hadoop jar ~/BaseStationDataPreprocess.jar \

完整的类名 \

完整的输入路径 输出路径 \

其他参数

```
james@Cluster-0:~/Documents/Data$ cat POSITION
0000000001      4600000000000000        0       0000-0001       2013-09-12 09:15:00
0000000001      4600000000000000        2       0000-0002       2013-09-12 09:30:00
0000000001      4600000000000000        0       0000-0001       2013-09-12 10:00:00
0000000001      4600000000000000        1       0000-0002       2013-09-12 11:45:00
james@Cluster-0:~/Documents/Data$ hadoop fs -put ~/Documents/Data/ /user/james/Base
james@Cluster-0:~/Documents/Data$ hadoop jar ~/BaseStationDataPreprocess.jar \
> cn.dataguru.hadoop.BaseStationDataPreprocess \
> Base Output \
> 2013-09-12 07-09-17-24
13/11/28 00:07:25 INFO input.FileInputFormat: Total input paths to process : 2
13/11/28 00:07:25 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/11/28 00:07:25 WARN snappy.LoadSnappy: Snappy native library not loaded
```

# 程序结果

```
Job Counters
  Launched reduce tasks=1
  SLOTS_MILLIS_MAPS=18354
  Total time spent by all reduces waiting after reserving slots (ms)=0
  Total time spent by all maps waiting after reserving slots (ms)=0
  Launched map tasks=2
  Data-local map tasks=2
  SLOTS_MILLIS_REDUCES=9972
cn.dataguru.hadoop.BaseStationDataPreprocess$Counter
  OUTOFTIMESKIP=2
FileSystemCounters
  FILE_BYTES_READ=246
  HDFS_BYTES_READ=737
  FILE_BYTES_WRITTEN=178486
  HDFS_BYTES_WRITTEN=64
Map-Reduce Framework
  Map input records=8
  Reduce shuffle bytes=252
  Spilled Records=12
  Map output bytes=228
  CPU time spent (ms)=3170
  Total committed heap usage (bytes)=247275520
  Combine input records=0
  SPLIT_RAW_BYTES=221
  Reduce input records=6
  Reduce input groups=1
  Combine output records=0
  Physical memory (bytes) snapshot=317296640
  Reduce output records=2
  Virtual memory (bytes) snapshot=1128271872
  Map output records=6
```

日期错误数据被记录到计数器中

# 程序结果

- 查看数据：

Hadoop fs –cat 目标文件路径 | more

```
james@Cluster-0:~/Documents/Data$ hadoop fs -cat /user/james/Output/part-r-00000 | more
0000000001|0000-0002|09-17|15.0
0000000001|0000-0001|09-17|70.0
```

# 炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

# Thanks

## FAQ时间