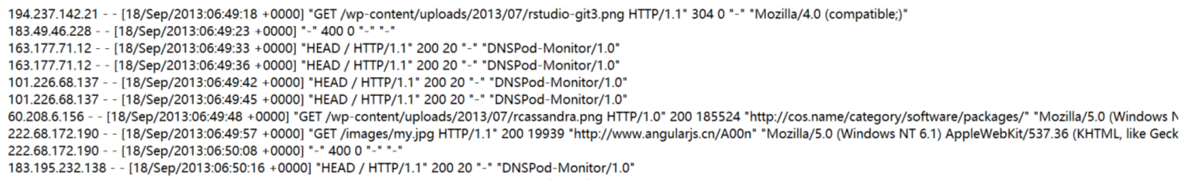
MapReduce7 数据清洗ETL

简单版日志清洗

需求

去除日志中长度小于等于11的日志



```
194.237.142.21 - - [18/Sep/2013:06:49:18 +0000] "GET /wp-content/uploads/2013/07/rstudio-git3.png HTTP/1.1" 304 0 "-" "Mozilla/4.0 (compatible;)"
183.49.46.228 - - [18/Sep/2013:06:49:23 +0000] "-" 400 0 "-" "-"
163.177.71.12 - - [18/Sep/2013:06:49:33 +0000] "HEAD / HTTP/1.1" 200 20 "-" "DNSPod-Monitor/1.0"
163.177.71.12 - - [18/Sep/2013:06:49:36 +0000] "HEAD / HTTP/1.1" 200 20 "-" "DNSPod-Monitor/1.0"
101.226.68.137 - - [18/Sep/2013:06:49:42 +0000] "HEAD / HTTP/1.1" 200 20 "-" "DNSPod-Monitor/1.0"
101.226.68.137 - - [18/Sep/2013:06:49:45 +0000] "HEAD / HTTP/1.1" 200 20 "-" "DNSPod-Monitor/1.0"
60.208.6.156 - - [18/Sep/2013:06:49:48 +0000] "GET /wp-content/uploads/2013/07/rcassandra.png HTTP/1.0" 200 185524 "http://cos.name/category/software/packages/" "Mozilla/5.0 (Windows N
222.68.172.190 - - [18/Sep/2013:06:49:57 +0000] "GET /images/my.jpg HTTP/1.1" 200 19939 "http://www.angularjs.cn/A00n" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Geck
222.68.172.190 - - [18/Sep/2013:06:50:08 +0000] "-" 400 0 "-" "-"
183.195.232.138 - - [18/Sep/2013:06:50:16 +0000] "HEAD / HTTP/1.1" 200 20 "-" "DNSPod-Monitor/1.0"
```

代码

Mapper代码

```java
public class LogMapper extends Mapper<LongWritable, Text, Text, NullWritable>{
    Text k = new Text();

    @Override protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        // 1 获取1行数据
        String line = value.toString();
        // 2 解析日志
        boolean result = parseLog(line,context);

        // 3 日志不合法退出
        if (!result) {
            return;
        }

        // 4 设置key
        k.set(line);

        // 5 写出数据
        context.write(k, NullWritable.get());
    }

    // 2 解析日志
    private boolean parseLog(String line, Context context) {
        // 1 截取
        String[] fields = line.split(" ");
        // 2 日志长度大于11的为合法
        if (fields.length > 11) {
            // 系统计数器
            context.getCounter("map", "true").increment(1);
            return true;
        }else {
            context.getCounter("map", "false").increment(1);
            return false;
        }
    }
}
```

Driver代码

```

public class LogDriver {
    public static void main(String[] args) throws Exception {
        // 输入输出路径需要根据自己电脑上实际的输入输出路径设置
        args = new String[] { "e:/input/inputlog", "e:/output1" };

        // 1 获取job信息
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        // 2 加载jar包
        job.setJarByClass(LogDriver.class);
        // 3 关联map
        job.setMapperClass(LogMapper.class);
        // 4 设置最终输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        // 设置reducetask个数为0
        job.setNumReduceTasks(0);
        // 5 设置输入和输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 6 提交
        job.waitForCompletion(true);
    }
}
```

复杂版数据清洗

需求

对Web访问日志中的个字段识别切分，去除日志中不合法的记录，根据清洗规则，输出过滤后的数据

Bean代码

```
public class LogBean {

    private String remote_addr;// 记录客户端的ip地址
    private String remote_user;// 记录客户端用户名称,忽略属性"-"
    private String time_local;// 记录访问时间与时区
    private String request;// 记录请求的url与http协议
    private String status;// 记录请求状态；成功是200
    private String body_bytes_sent;// 记录发送给客户端文件主体内容大小
    private String http_referer;// 用来记录从那个页面链接访问过来的
    private String http_user_agent;// 记录客户浏览器的相关信息
    private boolean valid = true;// 判断数据是否合法

    public String getRemote_addr() {
        return remote_addr;
    }
```

```java
    public void setRemote_addr(String remote_addr) {
        this.remote_addr = remote_addr;
    }

    public String getRemote_user() {
        return remote_user;
    }

    public void setRemote_user(String remote_user) {
        this.remote_user = remote_user;
    }

    public String getTime_local() {
        return time_local;
    }

    public void setTime_local(String time_local) {
        this.time_local = time_local;
    }

    public String getRequest() {
        return request;
    }

    public void setRequest(String request) {
        this.request = request;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getBody_bytes_sent() {
        return body_bytes_sent;
    }

    public void setBody_bytes_sent(String body_bytes_sent) {
        this.body_bytes_sent = body_bytes_sent;
    }


    public String getHttp_referer() {
        return http_referer;
    }

    public void setHttp_referer(String http_referer) {
        this.http_referer = http_referer;
    }

    public String getHttp_user_agent() {
        return http_user_agent;
    }
    public void setHttp_user_agent(String http_user_agent) {
        this.http_user_agent = http_user_agent;
    }

    public boolean isValid() {
        return valid;
    }

    public void setValid(boolean valid) {
        this.valid = valid;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(this.valid);
        sb.append("\001").append(this.remote_addr);
        sb.append("\001").append(this.remote_user);
        sb.append("\001").append(this.time_local);
        sb.append("\001").append(this.request);
        sb.append("\001").append(this.status);
        sb.append("\001").append(this.body_bytes_sent);
        sb.append("\001").append(this.http_referer);
        sb.append("\001").append(this.http_user_agent);
        return sb.toString();
    }
}
```

Mapper代码

```

public class LogMapper extends Mapper<LongWritable, Text, Text, NullWritable>{
    Text k = new Text();

    @Override protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        // 1 获取1行
        String line = value.toString(); // 2 解析日志是否合法
        LogBean bean = pressLog(line);

        if (!bean.isValid()) {
            return;
        }
        k.set(bean.toString());

        // 3 输出
        context.write(k, NullWritable.get());
    }

    // 解析日志
    private LogBean pressLog(String line) {
        LogBean logBean = new LogBean();
        // 1 截取
        String[] fields = line.split(" "); if (fields.length > 11) {
            // 2封装数据
            logBean.setRemote_addr(fields[0]);
            logBean.setRemote_user(fields[1]);
            logBean.setTime_local(fields[3].substring(1));
            logBean.setRequest(fields[6]);
            logBean.setStatus(fields[8]);
            logBean.setBody_bytes_sent(fields[9]);
            logBean.setHttp_referer(fields[10]);
            if (fields.length > 12) {
                logBean.setHttp_user_agent(fields[11] + " "+ fields[12]); }else
{
                logBean.setHttp_user_agent(fields[11]); }

            // 大于400，HTTP错误
            if (Integer.parseInt(logBean.getStatus()) >= 400) {
                logBean.setValid(false);
            }
        }else {
            logBean.setValid(false);
        }
        return logBean;
    }
}
```

Driver代码

```
public class LogDriver {
    public static void main(String[] args) throws Exception {
        // 1 获取job信息
        Configuration conf = new Configuration(); Job job =
Job.getInstance(conf);

        // 2 加载jar包
        job.setJarByClass(LogDriver.class);

        // 3 关联map
        job.setMapperClass(LogMapper.class);
        // 4 设置最终输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);

        // 5 设置输入和输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 6 提交
        job.waitForCompletion(true);
    }
}
```

MapReduce开发总结

输入数据接口

- 默认使用实现类是：TextInputFormat
- TextInputFormat的逻辑功能是：一次读一行文本，然后将该行的起始偏移量作为key，行内容作为value返回

- KeyValueTextInputFormat每一行均为一条记录，被分隔符分隔为key value。默认分隔符是"\t"
- NlineInputFormat按照指定的行数N来划分切片
- CombineTextInputFormat可以把多个小文件合并成一个切片处理，提高处理效率
- 自定义InputFormat

处理逻辑接口

用户根据业务需求实现其中三个方法：map(),setup();clearnup()

Partition分区

- 默认实现 HashPartitioner分区。逻辑是根据key的哈希值和numReduces来返回一个分区号：

  key.hash.Code() & Integer.MAX_VALUE%numReduces

- 如果业务上有需求，可以自定义分区

Comparable排序

- 当我们用自定义的对象作为key来输出的时。就必须实现WritableComparable接口，重写其中的compareTo()方法。
- 部分排序：对最终输出的每个文件进行内部排序
- 全排序：对所有数据进行排序，通常只有一个Reduce
- 二次排序：排序条件有两个

Combiner合并

Combiner合并可以提高程序执行效率，减少io传输，但是使用时必须不能影响原有的业务处理结果

Reduces分组

Mapreduce框架在记录到达Reducer之前按键对记录排序，但键所对应的值并没有被排序。一般来说，大多 数 MapReduce程序会避免让Reduce函数依赖于值的排序。但是，有时也需要通过特定的方法对键进行排序 和分组以实现对值的排序。

逻辑处理接口

Reducer用户根据业务需求实现其中三个方法：reduce(),setup(),clearnup()

输出数据接口

- 默认实现类是TextOutputFormat，逻辑功能是：将每一个KV对，向目标文本输出一行。
- 将SequenceFileOutputFormat输出作为后续Mapreduce任务的输入，这便是一种号的输出格式
- 用户还可以自定义输出格式OutputFormat