# 实验一

## （一）熟悉 HDFS 操作常用的 Shell 命令

（1）

```
$ cd /usr/local/hadoop
$ ./sbin/start-dfs.sh
$ ./bin/hdfs dfs -mkdir -p /user/Hadoop
```

（2）

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -mkdir test
$ ./bin/hdfs dfs -ls .
```

（3）

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -put ~/.bashrc test
$ ./bin/hdfs dfs -ls test
```

（4）

```
$ cd /usr/local/hadoop
$./bin/hdfs dfs -get test ./
```

## （二）编程实现以下功能

（1）

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import java.io.*;

public class HDFSApi {
    /**
     * 判断路径是否存在
     */
    public static boolean test(Configuration conf, String path) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        return fs.exists(new Path(path));
    }

    /**
     * 复制文件到指定路径
     * 若路径已存在，则进行覆盖
     */
    public static void copyFromLocalFile(Configuration conf, String localFilePath, String remoteFilePath) throws
IOException {
        FileSystem fs = FileSystem.get(conf);
        Path localPath = new Path(localFilePath);
        Path remotePath = new Path(remoteFilePath);
        /* fs.copyFromLocalFile 第一个参数表示是否删除源文件，第二个参数表示是否覆盖 */
        fs.copyFromLocalFile(false, true, localPath, remotePath);
```

```
        fs.close();
    }

    /**
     * 追加文件内容
     */
    public static void appendToFile(Configuration conf, String localFilePath, String remoteFilePath) throws
IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        /* 创建一个文件读入流 */
        FileInputStream in = new FileInputStream(localFilePath);
        /* 创建一个文件输出流，输出的内容将追加到文件末尾 */
        FSDataOutputStream out = fs.append(remotePath);
        /* 读写文件内容 */
        byte[] data = new byte[1024];
        int read = -1;
        while ( (read = in.read(data)) > 0 ) {
            out.write(data, 0, read);
        }
        out.close();
        in.close();
        fs.close();
    }

    /**
     * 主函数
     */
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        conf.set("fs.default.name","hdfs://localhost:9000");
        String localFilePath = "/home/hadoop/text.txt";    // 本地路径
        String remoteFilePath = "/user/hadoop/text.txt";    // HDFS 路径
        String choice = "append";    // 若文件存在则追加到文件末尾
            // String choice = "overwrite";    // 若文件存在则覆盖
        try {
            /* 判断文件是否存在 */
            Boolean fileExists = false;
            if (HDFSApi.test(conf, remoteFilePath)) {
                fileExists = true;
                System.out.println(remoteFilePath + " 已存在.");
            } else {
                System.out.println(remoteFilePath + " 不存在.");
            }
```

```
            /* 进行处理 */
            if ( !fileExists) { // 文件不存在，则上传
                HDFSApi.copyFromLocalFile(conf, localFilePath, remoteFilePath);
                System.out.println(localFilePath + " 已上传至 " + remoteFilePath);
            } else if ( choice.equals("overwrite") ) {    // 选择覆盖
                HDFSApi.copyFromLocalFile(conf, localFilePath, remoteFilePath);
                System.out.println(localFilePath + " 已覆盖 " + remoteFilePath);
            } else if ( choice.equals("append") ) {   // 选择追加
                HDFSApi.appendToFile(conf, localFilePath, remoteFilePath);
                System.out.println(localFilePath + " 已追加至 " + remoteFilePath);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（2）

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import java.io.*;
public class HDFSApi {
    /**
     * 下载文件到本地
     * 判断本地路径是否已存在，若已存在，则自动进行重命名
     */
    public static void copyToLocal(Configuration conf, String remoteFilePath, String localFilePath) throws
IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        File f = new File(localFilePath);
        /* 如果文件名存在，自动重命名(在文件名后面加上 _0, _1 ...) */
        if (f.exists()) {
            System.out.println(localFilePath + " 已存在.");
            Integer i = 0;
            while (true) {
                f = new File(localFilePath + "_" + i.toString());
                if (!f.exists()) {
                    localFilePath = localFilePath + "_" + i.toString();
                    break;
                }
                i++;
            }
            System.out.println("将重新命名为: " + localFilePath);
        }
```

```
        // 下载文件到本地
        Path localPath = new Path(localFilePath);
        fs.copyToLocalFile(remotePath, localPath);
        fs.close();
    }

    /**
     * 主函数
     */
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        conf.set("fs.default.name","hdfs://localhost:9000");
        String localFilePath = "/home/hadoop/text.txt";    // 本地路径
        String remoteFilePath = "/user/hadoop/text.txt";    // HDFS 路径

        try {
            HDFSApi.copyToLocal(conf, remoteFilePath, localFilePath);
            System.out.println("下载完成");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（3）

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import java.io.*;

public class HDFSApi {
    /**
     * 读取文件内容
     */
    public static void cat(Configuration conf, String remoteFilePath) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        FSDataInputStream in = fs.open(remotePath);
        BufferedReader d = new BufferedReader(new InputStreamReader(in));
        String line = null;
        while ( (line = d.readLine()) != null ) {
            System.out.println(line);
        }
        d.close();
        in.close();
```

```
        fs.close();
    }

    /**
     * 主函数
     */
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        conf.set("fs.default.name","hdfs://localhost:9000");
        String remoteFilePath = "/user/hadoop/text.txt";    // HDFS 路径

        try {
            System.out.println("读取文件: " + remoteFilePath);
            HDFSApi.cat(conf, remoteFilePath);
            System.out.println("\n 读取完成");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（4）

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import java.io.*;

public class HDFSApi {
    /**
     * 判断路径是否存在
     */
    public static boolean test(Configuration conf, String path) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        return fs.exists(new Path(path));
    }

    /**
     * 创建目录
     */
    public static boolean mkdir(Configuration conf, String remoteDir) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        boolean result = fs.mkdirs(dirPath);
        fs.close();
        return result;
    }
```

```
/**
 * 创建文件
 */
public static void touchz(Configuration conf, String remoteFilePath) throws IOException {
    FileSystem fs = FileSystem.get(conf);
    Path remotePath = new Path(remoteFilePath);
    FSDataOutputStream outputStream = fs.create(remotePath);
    outputStream.close();
    fs.close();
}

/**
 * 删除文件
 */
public static boolean rm(Configuration conf, String remoteFilePath) throws IOException {
    FileSystem fs = FileSystem.get(conf);
    Path remotePath = new Path(remoteFilePath);
    boolean result = fs.delete(remotePath, false);
    fs.close();
    return result;
}
/**
 * 主函数
 */
public static void main(String[] args) {
    Configuration conf = new Configuration();
    conf.set("fs.default.name","hdfs://localhost:9000");
    String remoteFilePath = "/user/hadoop/input/text.txt";    // HDFS 路径
    String remoteDir = "/user/hadoop/input";    // HDFS 路径对应的目录

    try {
        /* 判断路径是否存在，存在则删除，否则进行创建 */
        if ( HDFSApi.test(conf, remoteFilePath) ) {
            HDFSApi.rm(conf, remoteFilePath); // 删除
            System.out.println("删除路径: " + remoteFilePath);
        } else {
            if ( !HDFSApi.test(conf, remoteDir) ) { // 若目录不存在，则进行创建
                HDFSApi.mkdir(conf, remoteDir);
                System.out.println("创建文件夹: " + remoteDir);
            }
            HDFSApi.touchz(conf, remoteFilePath);
            System.out.println("创建路径: " + remoteFilePath);
        }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（5）

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import java.io.*;

public class HDFSApi {
    /**
     * 删除文件
     */
    public static boolean rm(Configuration conf, String remoteFilePath) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        boolean result = fs.delete(remotePath, false);
        fs.close();
        return result;
    }
    /**
     * 主函数
     */
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        conf.set("fs.default.name","hdfs://localhost:9000");
        String remoteFilePath = "/user/hadoop/text.txt";    // HDFS 文件

        try {
            if ( HDFSApi.rm(conf, remoteFilePath) ) {
                System.out.println("文件删除: " + remoteFilePath);
            } else {
                System.out.println("操作失败（文件不存在或删除失败）");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# 实验二

（一）编程实现以下指定功能，并用 Hadoop 提供的 HBase Shell 命令完成相同任务：

（1）hbase> list

```
public static void listTables() throws IOException {
    init();//建立连接
    HTableDescriptor hTableDescriptors[] = admin.listTables();
    for(HTableDescriptor hTableDescriptor :hTableDescriptors){
        System.out.println("表名:"+hTableDescriptor.getNameAsString());
    }
    close();//关闭连接
}
```

（2）hbase> scan 's1'

```
//在终端打印出指定的表的所有记录数据
public static void getData(String tableName)throws  IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Scan scan = new Scan();
    ResultScanner scanner = table.getScanner(scan);
    for (Result result:scanner){
        printRecoder(result);
    }
    close();
}
//打印一条记录的详情
public  static void printRecoder(Result result)throws IOException{
    for(Cell cell:result.rawCells()){
        System.out.print("行健: "+new String(CellUtil.cloneRow(cell)));
        System.out.print("列簇: "+new String(CellUtil.cloneFamily(cell)));
        System.out.print(" 列: "+new String(CellUtil.cloneQualifier(cell)));
        System.out.print(" 值: "+new String(CellUtil.cloneValue(cell)));
        System.out.println("时间戳: "+cell.getTimestamp());
    }
}
```

（3）

```
hbase> create 's1','score'
hbase> put 's1','zhangsan','score:Math','69'
hbase> delete 's1','zhangsan','score:Math'

//向表添加数据
public static void insertRow(String tableName,String rowKey,String colFamily,String col,String val) throws
IOException {
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Put put = new Put(rowKey.getBytes());
    put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
    table.put(put);
```

```
    table.close();
    close();
}
//删除数据
public static void deleteRow(String tableName,String rowKey,String colFamily,String col) throws IOException {
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Delete delete = new Delete(rowKey.getBytes());
    //删除指定列族
    delete.addFamily(Bytes.toBytes(colFamily));
    //删除指定列
    delete.addColumn(Bytes.toBytes(colFamily),Bytes.toBytes(col));
    table.delete(delete);
    table.close();
    close();
}
```

（4）hbase> truncate 's1'

```
//清空指定的表的所有记录数据
public static void clearRows(String tableName)throws IOException{
    init();
    TableName tablename = TableName.valueOf(tableName);
    admin.disableTable(tablename);
    admin.deleteTable(tablename);
    HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
    admin.createTable(hTableDescriptor);
    close();
}
```

（5）hbase> count 's1'

```
public static void countRows(String tableName)throws IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Scan scan = new Scan();
    ResultScanner scanner = table.getScanner(scan);
    int num = 0;
    for (Result result = scanner.next();result!=null;result=scanner.next()){
        num++;
    }
    System.out.println("行数:"+ num);
    scanner.close();
    close();
}
```

（二）HBase 数据库操作

1.

hbase> create 'Student','S_No','S_Name','S_Sex','S_Age'

```
hbase>put 'Student','s001','S_No','2015001'
hbase>put 'Student','s001','S_Name','Zhangsan'
hbase>put 'Student','s001','S_Sex','male'
hbase>put 'Student','s001','S_Age','23'
hbase>put 'Student','s002','S_No','2015002'
hbase>put 'Student','s002','S_Name','Mary'
hbase>put 'Student','s002','S_Sex','female'
hbase>put 'Student','s002','S_Age','22'
hbase>put 'Student','s003','S_No','2015003'
hbase>put 'Student','s003','S_Name','Lisi'
hbase>put 'Student','s003','S_Sex','male'
hbase>put 'Student','s003','S_Age','24'

hbase> create 'Course','C_No','C_Name','C_Credit'
hbase>put 'Course','c001','C_No','123001'
hbase>put 'Course','c001','C_Name','Math'
hbase>put 'Course','c001','C_Credit','2.0'
hbase>put 'Course','c002','C_No','123002'
hbase>put 'Course','c002','C_Name','Computer'
hbase>put 'Course','c002','C_Credit','5.0'
hbase>put 'Course','c003','C_No','123003'
hbase>put 'Course','c003','C_Name','English'
hbase>put 'Course','c003','C_Credit','3.0'

hbase> create 'SC','SC_Sno','SC_Cno','SC_Score'
hbase>put 'SC','sc001','SC_Sno','2015001'
hbase>put 'SC','sc001','SC_Cno','123001'
hbase>put 'SC','sc001','SC_Score','86'
hbase>put 'SC','sc002','SC_Sno','2015001'
hbase>put 'SC','sc002','SC_Cno','123003'
hbase>put 'SC','sc002','SC_Score','69'
hbase>put 'SC','sc003','SC_Sno','2015002'
hbase>put 'SC','sc003','SC_Cno','123002'
hbase>put 'SC','sc003','SC_Score','77'
hbase>put 'SC','sc004','SC_Sno','2015002'
hbase>put 'SC','sc004','SC_Cno','123003'
hbase>put 'SC','sc004','SC_Score','99'
hbase>put 'SC','sc005','SC_Sno','2015003'
hbase>put 'SC','sc005','SC_Cno','123001'
hbase>put 'SC','sc005','SC_Score','98'
hbase>put 'SC','sc006','SC_Sno','2015003'
hbase>put 'SC','sc006','SC_Cno','123002'
hbase>put 'SC','sc006','SC_Score','95'
```

2. 请编程实现以下功能：

（1）createTable(String tableName, String[] fields)

```
public static void createTable(String tableName,String[] fields) throws IOException {
    init();
    TableName tablename = TableName.valueOf(tableName);
    if(admin.tableExists(tablename)){
        System.out.println("table is exists!");
        admin.disableTable(tablename);
        admin.deleteTable(tablename);//删除原来的表
    }
    HTableDescriptor hTableDescriptor = new HTableDescriptor(tablename);
    for(String str:fields){
        HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
        hTableDescriptor.addFamily(hColumnDescriptor);
    }
    admin.createTable(hTableDescriptor);
    close();
}
```

（2）addRecord(String tableName, String row, String[] fields, String[] values)

```
public static void addRecord(String tableName,String row,String[] fields,String[] values) throws IOException {
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    for(int i = 0;i != fields.length;i++){
        Put put = new Put(row.getBytes());
        String[] cols = fields[i].split(":");
        put.addColumn(cols[0].getBytes(), cols[1].getBytes(), values[i].getBytes());
        table.put(put);
    }
    table.close();
    close();
}
```

（3）scanColumn(String tableName, String column)

```
public static void scanColumn(String tableName,String column)throws  IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Scan scan = new Scan();
    scan.addFamily(Bytes.toBytes(column));
    ResultScanner scanner = table.getScanner(scan);
    for (Result result = scanner.next(); result != null; result = scanner.next()){
        showCell(result);
    }
    table.close();
    close();
}
```

```
//格式化输出
public static void showCell(Result result){
    Cell[] cells = result.rawCells();
    for(Cell cell:cells){
        System.out.println("RowName:"+new String(CellUtil.cloneRow(cell))+" ");
        System.out.println("Timetamp:"+cell.getTimestamp()+" ");
        System.out.println("column Family:"+new String(CellUtil.cloneFamily(cell))+" ");
        System.out.println("row Name:"+new String(CellUtil.cloneQualifier(cell))+" ");
        System.out.println("value:"+new String(CellUtil.cloneValue(cell))+" ");
    }
}
```

### （4）modifyData(String tableName, String row, String column)

```
public static void modifyData(String tableName,String row,String column,String val)throws IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Put put = new Put(row.getBytes());
    put.addColumn(column.getBytes(),null,val.getBytes());
    table.put(put);
    table.close();
    close();
}
```

### （5）deleteRow(String tableName, String row)

```
public static void deleteRow(String tableName,String row)throws IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Delete delete = new Delete(row.getBytes());
    //删除指定列族
    //delete.addFamily(Bytes.toBytes(colFamily));
    //删除指定列
    //delete.addColumn(Bytes.toBytes(colFamily),Bytes.toBytes(col));
    table.delete(delete);
    table.close();
    close();
}
```

# 实验三

## （一）编程实现合并和去重操作

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class Merge {
  //重载 map 函数，直接将输入中的 value 复制到输出数据的 key 上
  public static class Map extends Mapper<Object, Text, Text, Text> {
    private static Text text = new Text();
      public void map(Object key, Text value, Context context) throws IOException,InterruptedException {
      text = value;
      context.write(text, new Text(""));
    }
  }
  //重载 reduce 函数，直接将输入中的 key 复制到输出数据的 key 上
  public static class Reduce extends Reducer<Text, Text, Text, Text> {
      public void reduce(Text key, Iterable<Text> values, Context context ) throws
          IOException,InterruptedException {
      context.write(key, new Text(""));
    }
  }

  public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
     conf.set("fs.default.name","hdfs://localhost:9000");
    String[] otherArgs = new String[]{"input","output"}; /* 直接设置输入参数 */
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in><out>");
      System.exit(2);
      }
    Job job = Job.getInstance(conf,"Merge and duplicate removal");
    job.setJarByClass(Merge.class);
    job.setMapperClass(Map.class);
```

```
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## （二）编写程序实现对输入文件的排序

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class MergeSort {
    //map 函数读取输入中的 value，将其转化成 IntWritable 类型，最后作为输出 key
    public static class Map extends Mapper<Object, Text, IntWritable, IntWritable> {
        private static IntWritable data = new IntWritable();
            public void map(Object key, Text value, Context context) throws IOException,InterruptedException {
            String text = value.toString();
            data.set(Integer.parseInt(text));
            context.write(data, new IntWritable(1));
        }
    }
      //reduce 函数将 map 输入的 key 复制到输出的 value 上，然后根据输入的 value-list 中元素的个数决定 key 的输出次
       数,定义一个全局变量 line_num 来代表 key 的位次
        public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
        private static IntWritable line_num = new IntWritable(1);
            public void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws
                    IOException,InterruptedException {
            for(IntWritable val : values) {
                context.write(line_num, key);
                line_num = new IntWritable(line_num.get() + 1);
            }
        }
```

```
    }
        //自定义 Partition 函数，此函数根据输入数据的最大值和 MapReduce 框架中 Partition 的数量获取将输入数据按照大
小分块的边界，然后根据输入数值和边界的关系返回对应的 Partiton ID
    public static class Partition extends Partitioner<IntWritable, IntWritable> {
        public int getPartition(IntWritable key, IntWritable value, int num_Partition) {
            int Maxnumber = 65223;//int 型的最大数值
            int bound = Maxnumber/num_Partition+1;
            int keynumber = key.get();
            for (int i = 0; i<num_Partition; i++) {
                if(keynumber<bound * (i+1) && keynumber>=bound * i) {
                    return i;
                }
            }
            return -1;
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
            conf.set("fs.default.name","hdfs://localhost:9000");
        String[] otherArgs = new String[]{"input","output"};
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in><out>");
            System.exit(2);
        }
            Job job = Job.getInstance(conf,"Merge and sort");
        job.setJarByClass(MergeSort.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setPartitionerClass(Partition.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

# 实验四

## 1、pyspark 交互式编程

### （1）该系总共有多少学生?

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).map(lambda x: x[0]) //获取每行数据的第 1 列
>>> distinct_res = res.distinct()  //去重操作
>>> distinct_res.count()//取元素总个数
```

### （2）该系共开设了多少门课程?

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).map(lambda x:x[1]) //获取每行数据的第 2 列
>>> distinct_res = res.distinct()//去重操作
>>> distinct_res.count()//取元素总个数
```

### （3）Tom 同学的总成绩平均分是多少?

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).filter(lambda x:x[0]=="Tom") //筛选 Tom 同学成绩信息
>>> res.foreach(print)
>>> score = res.map(lambda x:int(x[2])) //提取 Tom 同学的每门成绩，并转换为 int 类型
>>> num = res.count() //Tom 同学选课门数
>>> sum_score = score.reduce(lambda x,y:x+y) //Tom 同学的总成绩
>>> avg = sum_score/num // 总成绩/门数=平均分
>>> print(avg)
```

### （4）求每名同学的选修的课程门数?

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).map(lambda x:(x[0],1)) //学生每门课程都对应(学生姓名,1)，学生有 n 门课程则有 n 个(学生姓名,1)
>>> each_res = res.reduceByKey(lambda x,y: x+y) //按学生姓名获取每个学生的选课总数
>>> each_res.foreach(print)
```

### （5）该系 DataBase 课程共有多少人选修;

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).filter(lambda x:x[1]=="DataBase")
>>> res.count()
```

### （6）各门课程的平均分是多少;

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
>>> res = lines.map(lambda x:x.split(",")).map(lambda x:(x[1],(int(x[2]),1))) //为每门课程的分数后面新增一列 1，表示 1 个学生选择了该课程。格式如('ComputerNetwork', (44, 1))
>>> temp = res.reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1])) //按课程名聚合课程总分和选课人数。格式如('ComputerNetwork', (7370, 142))
>>> avg = temp.map(lambda x:(x[0], round(x[1][0]/x[1][1],2)))//课程总分/选课人数 = 平均分，并利用 round(x,2)保留两位小数
>>> avg.foreach(print)
```

### （7）使用累加器计算共有多少人选了 DataBase 这门课?

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/data1.txt")
```

```
>>> res = lines.map(lambda x:x.split(",")).filter(lambda x:x[1]=="DataBase")//筛选出选了 DataBase 课程的数据
>>> accum = sc.accumulator(0) //定义一个从 0 开始的累加器 accum
>>> res.foreach(lambda x:accum.add(1))//遍历 res，每扫描一条数据，累加器加 1
>>> accum.value //输出累加器的最终值
```

## 2、编写独立应用程序实现数据去重

```
from pyspark import SparkConf, SparkContext
conf=SparkConf().setMaster('local').setAppName('remdup')
sc = SparkContext(conf=conf)
lines1 = sc.textFile("file:///usr/local/spark/mycode/rdd/A")
lines2 = sc.textFile("file:///usr/local/spark/mycode/rdd/B")
#合并两个文件的内容
lines = lines1.union(lines2)
#去重操作
distinct_lines = lines.distinct()
res = distinct_lines.sortBy(lambda x:x)
#将结果写入 result 文件中，repartition(1)的作用是让结果合并到一个文件中，不加的话会结果写入到两个文件
res.repartition(1).saveAsTextFile("file:///usr/local/spark/mycode/rdd/result")
```

## 3、编写独立应用程序实现求平均值问题

```
from pyspark import SparkConf, SparkContext
conf=SparkConf().setMaster('local').setAppName('remdup')
sc = SparkContext(conf=conf)
#加载三个文件 Algorithm.txt、Database.txt 和 Python.txt
lines1 = sc.textFile("file:///usr/local/spark/mycode/rdd/Algorithm.txt")
lines2 = sc.textFile("file:///usr/local/spark/mycode/rdd/Database.txt")
lines3 = sc.textFile("file:///usr/local/spark/mycode/rdd/Python.txt")
#合并三个文件的内容
lines = lines1.union(lines2).union(lines3)
#为每行数据新增一列 1，方便后续统计每个学生选修的课程数目。data 的数据格式为('小明', (92, 1))
data = lines.map(lambda x:x.split(" ")).map(lambda x:(x[0],(int(x[1]),1)))
#根据 key 也就是学生姓名合计每门课程的成绩，以及选修的课程数目。res 的数据格式为('小明', (269, 3))
res = data.reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1]))
#利用总成绩除以选修的课程数来计算每个学生的每门课程的平均分，并利用 round(x,2)保留两位小数
result = res.map(lambda x:(x[0],round(x[1][0]/x[1][1],2)))
#将结果写入 result 文件中，repartition(1)的作用是让结果合并到一个文件中，不加的话会结果写入到三个文件
result.repartition(1).saveAsTextFile("file:///usr/local/spark/mycode/rdd/result")
```