实训 1 熟悉常用的 Linux 操作和 Hadoop 操作

1 实训目的

1. 为后续上机实训做准备，熟悉常用的 Linux 操作。

2 实训平台

操作系统：Linux
Hadoop 版本：2.7.1

3 实训内容和要求

cd 命令：切换目录
（1）切换到目录 /usr/local
```
cd /usr/local
```
（2）去到目前的上层目录
```
cd ..
```
（3）回到自己的主文件夹
```
cd ~
```

ls 命令：查看文件与目录
（4）查看目录/usr 下所有的文件
```
cd /usr
ls -al
```

mkdir 命令：新建新目录
（5）进入/tmp 目录，创建一个名为 a 的目录,并查看有多少目录存在
```
cd /tmp
mkdir a
ls -al
```
（6）创建目录 a1/a2/a3/a4
```
mkdir -p a1/a2/a3/a4
```

rmdir 命令：删除空的目录
（7）将上例创建的目录 a（/tmp 下面）删除
```
rmdir a
```
（8）删除目录 a1/a2/a3/a4，查看有多少目录存在
```
rmdir -p a1/a2/a3/a4
ls -al
```

cp 命令：复制文件或目录
（9）将主文件夹下的.bashrc 复制到/usr 下，命名为 bashrc1
```
sudo cp ~/.bashrc /usr/bashrc1
```
（10）在/tmp 下新建目录 test，再复制这个目录内容到/usr
```
cd /tmp
mkdir test
sudo cp -r /tmp/test /usr
```

mv 命令：移动文件与目录，或更名
（11）将上例文件 bashrc1 移动到目录/usr/test
```
sudo mv /usr/bashrc1 /usr/test
```
（12）将上例 test 目录重命名为 test2
```
sudo mv /usr/test /usr/test2
```

rm 命令：移除文件或目录
（13）将上例复制的 bashrc1 文件删除
sudo rm /usr/test2/bashrc1
（14）将上例的 test2 目录删除
sudo rm -r /usr/test2

cat 命令：查看文件内容
（15）查看主文件夹下的.bashrc 文件内容
cat ~/.bashrc

tac 命令：反向列示
（16）反向查看主文件夹下.bashrc 文件内容
tac ~/.bashrc

more 命令：一页一页翻动查看
（17）翻页查看主文件夹下.bashrc 文件内容
more ~/.bashrc

head 命令：取出前面几行
（18）查看主文件夹下.bashrc 文件内容前 20 行
head -n 20 ~/.bashrc
（19）查看主文件夹下.bashrc 文件内容，后面 50 行不显示，只显示前面几行
head -n -50 ~/.bashrc

tail 命令：取出后面几行
（20）查看主文件夹下.bashrc 文件内容最后 20 行
tail -n 20 ~/.bashrc
（21） 查看主文件夹下.bashrc 文件内容，只列出 50 行以后的数据
tail -n +50 ~/.bashrc

touch 命令：修改文件时间或创建新文件
（22）在/tmp 下创建一个空文件 hello 并查看时间
cd /tmp
touch hello
ls -l hello
（23）修改 hello 文件，将日期调整为 5 天前
touch -d “5 days ago” hello

chown 命令：修改文件所有者权限
（24）将 hello 文件所有者改为 root 帐号，并查看属性
sudo chown root /tmp/hello
ls -l /tmp/hello

find 命令：文件查找
（25）找出主文件夹下文件名为.bashrc 的文件
find ~ -name .bashrc

tar 命令：压缩命令
（26）在/目录下新建文件夹 test,然后在/目录下打包成 test.tar.gz
sudo mkdir /test
sudo tar -zcv -f /test.tar.gz test
（27）解压缩到/tmp 目录
sudo tar -zxv -f /test.tar.gz -C /tmp

grep 命令：查找字符串
（28）从～/.bashrc 文件中查找字符串'examples'
grep -n 'examples' ~/.bashrc

（29）配置 Java 环境变量，在～/.bashrc 中设置
gedit ~/.bashrc
将语句　export JAVA_HOME=JDK 安装路径　加入文件第一行并保存
source ~/.bashrc

（30）查看 JAVA_HOME 变量的值
echo $JAVA_HOME

HDFS 相关命令
（31）启动 hadoop，在 HDFS 中创建用户目录（现在已经在 hadoop 目录/usr/local/hadoop）
./bin/hdfs dfs -mkdir -p /user/hadoop
（32）接着在此用户目录下创建 text 文件夹，并查看文件列表
./bin/hdfs dfs -mkdir test
./bin/hdfs dfs -ls .
（33）将～/.bashrc 文件上传到 HDFS 的 test 文件夹，并查看 test
./bin/hdfs dfs -put ~/.bashrc test
./bin/hdfs dfs -ls test
（34）将 HDFS 文件夹 test 拷到本机
./bin/hdfs dfs -get test ./test

4 实训报告

《大数据技术原理与应用》课程机房上机实训报告				
题目：		姓名		日期
实训环境：				
实训内容与完成情况：				
出现的问题：				
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：				

实训 2 熟悉常用的 HDFS 操作

1 实训目的

1. 理解 HDFS 在 Hadoop 体系结构中的角色；
2. 熟练使用 HDFS 操作常用的 Shell 命令；
3. 熟悉 HDFS 操作常用的 Java API。

2 实训平台

操作系统：Linux
Hadoop 版本：2.6.0 或以上版本
JDK 版本：1.6 或以上版本
Java IDE：Eclipse

3 实训内容和要求

1. 编程实现以下指定功能，并利用 Hadoop 提供的 Shell 命令完成相同任务：

提示：
a.i.1) 部分 Shell 命令的参数路径只能是本地路径或者 HDFS 路径。
a.i.2) 若 Shell 命令的参数既可以是本地路径，也可以是 HDFS 路径时，务必注意区分。为保证操作正确，可指定路径前缀 hdfs:/// 或者 file:///
a.i.3) 注意区分相对路径与绝对路径
a.i.4) 具体命令的说明可参考教材或 http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

（1） 向 HDFS 中上传任意文本文件，如果指定的文件在 HDFS 中已经存在，由用户指定是追加到原有文件末尾还是覆盖原有的文件；

Shell 命令:
检查文件是否存在: hdfs dfs -test -e text.txt(执行完这一句不会输出结果，需要继续输入命令 " echo $?")
追加命令: hdfs dfs -appendToFile local.txt text.txt
覆盖命令 1: hdfs dfs -copyFromLocal -f local.txt text.txt
覆盖命令 2: hdfs dfs -cp -f file:///home/hadoop/local.txt text.txt

也可以使用如下命令实现：
（如下代码可视为一行代码，在终端中输入第一行代码后，直到输入 fi 才会真正执行）：

```
if $(hdfs dfs -test -e text.txt);
then $(hdfs dfs -appendToFile local.txt text.txt);
else $(hdfs dfs -copyFromLocal -f local.txt text.txt);
fi
```

Java 代码：

```
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） 从 HDFS 中下载指定文件，如果本地文件与要下载的文件名称相同，则自动对下载的文件重命名；

Shell 命令：

```
if $(hdfs dfs -test -e file:///home/hadoop/text.txt);
then $(hdfs dfs -copyToLocal text.txt ./text2.txt);
else $(hdfs dfs -copyToLocal text.txt ./text.txt);
fi
```

Java 代码：

```
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;
                    }
                }
                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） 将 HDFS 中指定文件的内容输出到终端中；

Shell 命令：

```
hdfs dfs -cat text.txt
```

Java 代码：

```
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） 显示 HDFS 中指定的文件的读写权限、大小、创建时间、路径等信息；

Shell 命令：

```
hdfs dfs -ls -h text.txt
```

Java 代码：

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

public class HDFSApi {
    /**
     * 显示指定文件的信息
     */
    public static void ls(Configuration conf, String remoteFilePath) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        FileStatus[] fileStatuses = fs.listStatus(remotePath);
        for (FileStatus s : fileStatuses) {
            System.out.println("路径: " + s.getPath().toString());
            System.out.println("权限: " + s.getPermission().toString());
            System.out.println("大小: " + s.getLen());
            /* 返回的是时间戳,转化为时间日期格式 */
            Long timeStamp = s.getModificationTime();
            SimpleDateFormat format =  new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String date = format.format(timeStamp);
```

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

（5） 给定 HDFS 中某一个目录，输出该目录下的所有文件的读写权限、大小、创建时间、路径等信息，如果该文件是目录，则递归输出该目录下所有文件相关信息；

Shell 命令：

```
hdfs dfs -ls -R -h /user/hadoop
```

Java 代码：

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

public class HDFSApi {
    /**
     * 显示指定文件夹下所有文件的信息（递归）
     */
    public static void lsDir(Configuration conf, String remoteDir) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        /* 递归获取目录下的所有文件 */
        RemoteIterator<LocatedFileStatus> remoteIterator = fs.listFiles(dirPath, true);
        /* 输出每个文件的信息 */
        while (remoteIterator.hasNext()) {
            FileStatus s = remoteIterator.next();
            System.out.println("路径: " + s.getPath().toString());
            System.out.println("权限: " + s.getPermission().toString());
            System.out.println("大小: " + s.getLen());
            /* 返回的是时间戳,转化为时间日期格式 */
            Long timeStamp = s.getModificationTime();
```

```
                SimpleDateFormat format =  new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                String date = format.format(timeStamp);
                System.out.println("时间: " + date);
                System.out.println();
            }
            fs.close();
        }

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

            try {
                System.out.println("(递归)读取目录下所有文件的信息: " + remoteDir);
                HDFSApi.lsDir(conf, remoteDir);
                System.out.println("读取完成");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

（6） 提供一个 HDFS 内的文件的路径，对该文件进行创建和删除操作。如果文件所在目录不存在，则自动创建目录；

Shell 命令：

```
if $(hdfs dfs -test -d dir1/dir2);
then $(hdfs dfs -touchz dir1/dir2/filename);
else $(hdfs dfs -mkdir -p dir1/dir2 && hdfs dfs -touchz dir1/dir2/filename);
fi
```

删除文件：hdfs dfs -rm dir1/dir2/filename

Java 代码：

```
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();
            }
        }
}
```

（7） 提供一个 HDFS 的目录的路径，对该目录进行创建和删除操作。创建目录时，如果目录文件所在目录不存在则自动创建相应目录；删除目录时，由用户指定当该目录不为空时是否还删除该目录；

Shell 命令：

创建目录：hdfs dfs -mkdir -p dir1/dir2
删除目录（如果目录非空则会提示 not empty，执行删除）：hdfs dfs -rmdir dir1/dir2
强制删除目录：hdfs dfs -rm -R dir1/dir2

Java 代码：

```
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));
    }

    /**
     * 判断目录是否为空
     * true: 空，false: 非空
     */
    public static boolean isDirEmpty(Configuration conf, String remoteDir) throws IOException
{
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        RemoteIterator<LocatedFileStatus> remoteIterator = fs.listFiles(dirPath, true);
        return !remoteIterator.hasNext();
    }

    /**
     * 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;
    }
```

```
    /**
     * 删除目录
     */
    public static boolean rmDir(Configuration conf, String remoteDir) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        /* 第二个参数表示是否递归删除所有文件 */
        boolean result = fs.delete(dirPath, true);
        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 remoteDir = "/user/hadoop/input";    // HDFS 目录
        Boolean forceDelete = false;  // 是否强制删除

        try {
            /* 判断目录是否存在，不存在则创建，存在则删除 */
            if ( !HDFSApi.test(conf, remoteDir) ) {
                HDFSApi.mkdir(conf, remoteDir); // 创建目录
                System.out.println("创建目录: " + remoteDir);
            } else {
                if ( HDFSApi.isDirEmpty(conf, remoteDir) || forceDelete ) { // 目录为空或强
制删除
                    HDFSApi.rmDir(conf, remoteDir);
                    System.out.println("删除目录: " + remoteDir);
                } else  { // 目录不为空
                    System.out.println("目录不为空，不删除: " + remoteDir);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（8） 向 HDFS 中指定的文件追加内容，由用户指定内容追加到原有文件的开头或结尾；

Shell 命令：

追加到文件末尾：hdfs dfs -appendToFile local.txt text.txt
追加到文件开头：
（由于没有直接的命令可以操作，方法之一是先移动到本地进行操作，再进行上传覆盖）：
hdfs dfs -get text.txt
cat text.txt >> local.txt
hdfs dfs -copyFromLocal -f text.txt text.txt

Java 代码：

```
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));
    }

    /**
     * 追加文本内容
     */
        public static void appendContentToFile(Configuration conf, String content, String remoteFilePath) throws IOException {
        FileSystem fs = FileSystem.get(conf);
        Path remotePath = new Path(remoteFilePath);
        /* 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);
        out.write(content.getBytes());
        out.close();
        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();
    }

    /**
     * 移动文件到本地
```

```
     * 移动后，删除源文件
     */
      public static void moveToLocalFile(Configuration conf, String remoteFilePath, String localFilePath) throws IOException {
          FileSystem fs = FileSystem.get(conf);
          Path remotePath = new Path(remoteFilePath);
          Path localPath = new Path(localFilePath);
          fs.moveToLocalFile(remotePath, localPath);
    }

    /**
     * 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();
    }

    /**
     * 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
          String content = "新追加的内容\n";
          String choice = "after";          //追加到文件末尾
//          String choice = "before";    // 追加到文件开头

          try {
               /* 判断文件是否存在 */
               if ( !HDFSApi.test(conf, remoteFilePath) ) {
                    System.out.println("文件不存在: " + remoteFilePath);
               } else {
                    if ( choice.equals("after") ) { // 追加在文件末尾
                         HDFSApi.appendContentToFile(conf, content, remoteFilePath);
                         System.out.println("已追加内容到文件末尾" + remoteFilePath);
                    } else if ( choice.equals("before") ) { // 追加到文件开头
                         /* 没有相应的 api 可以直接操作，因此先把文件移动到本地，创建一个新的 HDFS，再按顺序追加内容 */
                         String localTmpPath = "/user/hadoop/tmp.txt";
                         HDFSApi.moveToLocalFile(conf, remoteFilePath, localTmpPath);  // 移动到本地
                         HDFSApi.touchz(conf, remoteFilePath);    // 创建一个新文件
                         HDFSApi.appendContentToFile(conf, content, remoteFilePath);   // 先写入新内容
                         HDFSApi.appendToFile(conf, localTmpPath, remoteFilePath);    // 再写入原来内容
                         System.out.println("已追加内容到文件开头: " + remoteFilePath);
```

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

（9） 删除 HDFS 中指定的文件；

Shell 命令：

```
hdfs dfs -rm text.txt
```

Java 命令：

```
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();
        }
    }
}
```

（10） 删除 HDFS 中指定的目录，由用户指定目录中如果存在文件时是否删除目录；

Shell 命令：

```
删除目录（如果目录非空则会提示 not empty，
```

执行删除）：hdfs dfs -rmdir dir1/dir2
强制删除目录：hdfs dfs -rm -R dir1/dir2

Java 代码：

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

public class HDFSApi {
    /**
     * 判断目录是否为空
     * true: 空，false: 非空
     */
    public static boolean isDirEmpty(Configuration conf, String remoteDir) throws IOException
{
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        RemoteIterator<LocatedFileStatus> remoteIterator = fs.listFiles(dirPath, true);
        return !remoteIterator.hasNext();
    }

    /**
     * 删除目录
     */
    public static boolean rmDir(Configuration conf, String remoteDir, boolean recursive) throws
IOException {
        FileSystem fs = FileSystem.get(conf);
        Path dirPath = new Path(remoteDir);
        /* 第二个参数表示是否递归删除所有文件 */
        boolean result = fs.delete(dirPath, recursive);
        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 remoteDir = "/user/hadoop/input";    // HDFS 目录
        Boolean forceDelete = false;  // 是否强制删除

        try {
            if ( !HDFSApi.isDirEmpty(conf, remoteDir) && !forceDelete ) {
                System.out.println("目录不为空，不删除");
            } else {
                if ( HDFSApi.rmDir(conf, remoteDir, forceDelete) ) {
                    System.out.println("目录已删除: " + remoteDir);
                } else {
                    System.out.println("操作失败");
                }
            }
```

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

（11） 在 HDFS 中，将文件从源路径移动到目的路径。

Shell 命令：

```
hdfs dfs -mv text.txt text2.txt
```

Java 代码：

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

public class HDFSApi {
    /**
     * 移动文件
     */
        public static boolean mv(Configuration conf, String remoteFilePath, String remoteToFilePath) throws IOException {
            FileSystem fs = FileSystem.get(conf);
            Path srcPath = new Path(remoteFilePath);
            Path dstPath = new Path(remoteToFilePath);
            boolean result = fs.rename(srcPath, dstPath);
            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 = "hdfs:///user/hadoop/text.txt";    // 源文件 HDFS 路径
        String remoteToFilePath = "hdfs:///user/hadoop/new.txt";    // 目的 HDFS 路径

        try {
            if ( HDFSApi.mv(conf, remoteFilePath, remoteToFilePath) ) {
                System.out.println(" 将 文 件    " + remoteFilePath + "  移 动 到    " + remoteToFilePath);
            } else {
                    System.out.println("操作失败(源文件不存在或移动失败)");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2. 编程实现一个类“MyFSDataInputStream”，该类继承“org.apache.hadoop.fs.FSDataInputStream”，要求如下：实现按行读取 HDFS 中指定文件的方法“readLine()”，如果读到文件末尾，则返回空，否则返回文件一行的文本。

Java 代码：

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

public class MyFSDataInputStream extends FSDataInputStream {
    public MyFSDataInputStream(InputStream in) {
        super(in);
    }

    /**
     * 实现按行读取
     * 每次读入一个字符，遇到"\n"结束，返回一行内容
     */
    public static String readline(BufferedReader br) throws IOException {
        char[] data = new char[1024];
        int read = -1;
        int off = 0; // 循环执行时，br 每次会从上一次读取结束的位置继续读取，因此该函数里，off 每次都从 0 开始
        while ( (read = br.read(data, off, 1)) != -1 ) {
            if (String.valueOf(data[off]).equals("\n") ) {
                off += 1;
                break;
            }
            off += 1;
        }

        if (off > 0) {
            return String.valueOf(data);
        } else {
            return null;
        }
    }

    /**
     * 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 br = new BufferedReader(new InputStreamReader(in));
        String line = null;
        while ( (line = MyFSDataInputStream.readline(br)) != null ) {
            System.out.println(line);
        }
        br.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 {
                MyFSDataInputStream.cat(conf, remoteFilePath);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

3. 查看 Java 帮助手册或其它资料，用“java.net.URL”和“org.apache.hadoop.fs.FsURLStreamHandlerFactory”编程完成输出 HDFS 中指定文件的文本到终端中。

Java 代码：

```
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.IOUtils;
import java.io.*;
import java.net.URL;

public class HDFSApi {
    static{
        URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());
    }

    /**
     * 主函数
     */
    public static void main(String[] args) throws Exception {
        String remoteFilePath = "hdfs:///user/hadoop/text.txt";    // HDFS 文件
        InputStream in = null;
        try{
            /* 通过 URL 对象打开数据流，从中读取数据 */
            in = new URL(remoteFilePath).openStream();
            IOUtils.copyBytes(in,System.out,4096,false);
        } finally{
            IOUtils.closeStream(in);
        }
    }
}
```

4 实训报告

《大数据技术原理与应用》课程机房上机实训报告				
题目：		姓名		日期

实训环境：
实训内容与完成情况：
出现的问题：
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：

实训 3 熟悉常用的 HBase 操作

1 实训目的

1. 理解 HBase 在 Hadoop 体系结构中的角色；
2. 熟练使用 HBase 操作常用的 Shell 命令；
3. 熟悉 HBase 操作常用的 Java API。

2 实训平台

操作系统：Linux
Hadoop 版本：2.6.0 或以上版本
HBase 版本：1.1.2 或以上版本
JDK 版本：1.6 或以上版本
Java IDE：Eclipse

3 实训内容和要求

1. 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：（完整可执行代码见 代码/QuestionOne.java）

（1） 列出 HBase 所有的表的相关信息，例如表名；

Shell:

```
list
```

编程：

```
//(1)列出 HBase 所有的表的相关信息，例如表名、创建时间等
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） 在终端打印出指定的表的所有记录数据；

Shell:

```
scan 's1'
```

编程：

```
//(2)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） 向已经创建好的表添加和删除指定的列族或列；

此题请先在 Shell 中创建 s1 作为示例表: create 's1','score'

a) 在 s1 表,添加数据：

Shell:

put 's1','zhangsan','score:Math','69'

编程：

```
//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();
}
insertRow("s1",'zhangsan','score','Math','69')
```

b) 在 s1 表，删除指定的列：

Shell:

delete 's1','zhangsan','score:Math'

编程：

```
//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();
}
deleteRow("s1",'zhangsan','score','Math')
```

（4） 清空指定的表的所有记录数据；

Shell:

truncate 's1'

编程:

```
//(4)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） 统计表的行数。

Shell:

count 's1'

编程：

```
//(5)统计表的行数
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();
    }
```

2. 现有以下关系型数据库中的表和数据，要求将其转换为适合于 HBase 存储的表并插入数据：

学生表（Student）

学号（S_No）	姓名（S_Name）	性别（S_Sex）	年龄（S_Age）
2015001	Zhangsan	male	23
2015002	Mary	female	22
2015003	Lisi	male	24

课程表（Course）

课程号（C_No）	课程名（C_Name）	学分（C_Credit）
123001	Math	2.0
123002	Computer	5.0
123003	English	3.0

选课表（SC）

学号（SC_Sno）	课程号（SC_Cno）	成绩（SC_Score）
2015001	123001	86
2015001	123003	69
2015002	123002	77
2015002	123003	99
2015003	123001	98
2015003	123002	95

i. 学生 Student 表

主键的列名是随机分配的，因此无需创建主键列

创建表：create 'Student','S_No','S_Name','S_Sex','S_Age'

插入数据:

	插入数据 shell 命令
第一行数据	put 'Student','s001','S_No','2015001' put 'Student','s001','S_Name','Zhangsan' put 'Student','s001','S_Sex','male' put 'Student','s001','S_Age','23'
第二行数据	put 'Student','s002','S_No','2015002' put 'Student','s002','S_Name','Mary' put 'Student','s002','S_Sex','female' put 'Student','s002','S_Age','22'
第三行数据	put 'Student','s003','S_No','2015003' put 'Student','s003','S_Name','Lisi' put 'Student','s003','S_Sex','male' put 'Student','s003','S_Age','24'

ii. 课程 Course 表

创建表：create 'Course','C_No','C_Name','C_Credit'

插入数据：

	插入数据 shell 命令
第一行数据	put 'Course','c001','C_No','123001' put 'Course','c001','C_Name','Math'

	put 'Course','c001','C_Credit','2.0'
第二行数据	put 'Course','c002','C_No','123002' put 'Course','c002','C_Name','Computer' put 'Course','c002','C_Credit','5.0'
第三行数据	put 'Course','c003','C_No','123003' put 'Course','c003','C_Name','English' put 'Course','c003','C_Credit','3.0'

iii. 选课表

创建表：create 'SC','SC_Sno','SC_Cno','SC_Score'

插入数据：

	插入数据 shell 命令
第一行数据	put 'SC','sc001','SC_Sno','2015001' put 'SC','sc001','SC_Cno','123001' put 'SC','sc001','SC_Score','86'
第二行数据	put 'SC','sc002','SC_Sno','2015001' put 'SC','sc002','SC_Cno','123003' put 'SC','sc002','SC_Score','69'
第三行数据	put 'SC','sc003','SC_Sno','2015002' put 'SC','sc003','SC_Cno','123002' put 'SC','sc003','SC_Score','77'
第四行数据	put 'SC','sc004','SC_Sno','2015002' put 'SC','sc004','SC_Cno','123003' put 'SC','sc004','SC_Score','99'
第五行数据	put 'SC','sc005','SC_Sno','2015003' put 'SC','sc005','SC_Cno','123001' put 'SC','sc005','SC_Score','98'
第六行数据	put 'SC','sc006','SC_Sno','2015003' put 'SC','sc006','SC_Cno','123002' put 'SC','sc006','SC_Score','95'

同时，请编程完成以下指定功能：（完整可执行代码见 代码/QuestionTwo.java）

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

创建表，参数 tableName 为表的名称，字符串数组 fields 为存储记录各个域名称的数组。要求当 HBase 已经存在名为 tableName 的表的时候，先删除原有的表，然后再创建新的表。

```
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)

向表 tableName、行 row（用 S_Name 表示）和字符串数组 files 指定的单元格中添加对应的数据 values。其中 fields 中每个元素如果对应的列族下还有相应的列限定符的话，用“columnFamily:column”表示。例如，同时向“Math”、“Computer Science”、“English”三列添

加成绩时，字符串数组 fields 为{“Score:Math”,”Score；Computer Science”,”Score:English”}，数组 values 存储这三门课的成绩。

```
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)

浏览表 tableName 某一列的数据，如果某一行记录中该列数据不存在，则返回 null。要求当参数 column 为某一列族名称时，如果底下有若干个列限定符，则要列出每个列限定符代表的列的数据；当参数 column 为某一列具体名称（例如“Score:Math”）时，只需要列出该列的数据。

```
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)

修改表 tableName，行 row（可以用学生姓名 S_Name 表示），列 column 指定的单元格的数据。（在 SHELL 中修改数据和添加数据都是用 put）

```
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)

删除表 tableName 中 row 指定的行的记录。

```
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();
}
```

3. 利用 HBase 和 MapReduce 完成如下任务：

假设 HBase 有 2 张表，表的逻辑视图及部分数据如下所示：

表 逻辑视图及部分数据

书名（bookName）	价格（price）
Database System Concept	30$
Thinking in Java	60$
Data Mining	25$

要求：从 HBase 读出上述两张表的数据，对“price”的排序，并将结果存储到 HBase 中。(MapReduce 是后面的章节，考虑到不使用 MapReduce,Hbase 只能对 Rowkey 进行排序，此题的不用 Mapreduce 的方法只有把价格 price 作为第二张表的 RowKey)

操作方法如下：

```
create 'book','bookName'
put 'book','val_60$','bookName:','Thinking in Java'
put 'book','val_20$','bookName:','Database System Concept'
put 'book','val_30$','bookName:','Data Mining'
scan #此处查询所有的数据，就会把按照 rowKey 自动排序
```

4 实训报告

《大数据技术原理与应用》课程机房上机实训报告				
题目：		姓名		日期
实训环境：				
实训内容与完成情况：				
出现的问题：				
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：				

实训 4 NoSQL 和关系数据库的操作比较

1 实训目的

1. 理解四种数据库(MySQL,HBase,Redis,MongoDB)的概念以及不同点;
2. 熟练使用四种数据库操作常用的 Shell 命令；
3. 熟悉四种数据库操作常用的 Java API。

2 实训平台

操作系统：Linux
Hadoop 版本：2.6.0 或以上版本
MySQL 版本:5.6 或以上版本
HBase 版本：1.1.2 或以上版本
Redis 版本: 3.0.6 或以上版本
MongoDB 版本: 3.2.6 或以上版本
JDK 版本：1.7 或以上版本
Java IDE：Eclipse

3 实训内容和要求

3.1 MySQL 数据库操作

Student 学生表

Name	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88

1. 根据上面给出的表格,利用 MySQL5.6 设计出 student 学生表格;

解答: 创建上述表格的 sql 语句为:

```
create table student(
    name varchar(30) not null,
    English tinyint unsigned not null,
    Math tinyint unsigned not null,
    Computer tinyint unsigned not null
    );
```

插入两条记录 sql 语句为:

```
insert into student values("zhangsan",69,86,77);
insert into student values("lisi",55,100,88);
```

a) 设计完后,用 select 语句输出所有的相关信息,并给出截图;

解答: sql 语句: select * from student;

```
mysql> select * from student;
+----------+---------+------+----------+
| name     | English | Math | Computer |
+----------+---------+------+----------+
| zhangsan |      69 |   86 |       77 |
| lisi     |      55 |  100 |       88 |
+----------+---------+------+----------+
2 rows in set (0.00 sec)
```

b) 查询 zhangsan 的 Computer 成绩,并给出截图;

解答:sql 语句为:select name , Computer from student where name = "zhangsan";

```
mysql> select name , Computer from student where name = "zhangsan";
+----------+----------+
| name     | Computer |
+----------+----------+
| zhangsan |       77 |
+----------+----------+
1 row in set (0.02 sec)
```

c) 修改 lisi 的 Math 成绩,改为 95.给出截图.

解答: sql 语句为: update student set Math=95 where name="lisi";

```
mysql> update student set Math=95 where name="lisi";
Query OK, 0 rows affected (0.05 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> select name,Math from student where name = "lisi";
+------+------+
| name | Math |
+------+------+
| lisi |   95 |
+------+------+
1 row in set (0.00 sec)
```

2. 根据上面已经设计出的 student 表,用 MySQL 的 JAVA 客户端编程;

附: jdbc 下载地址:http://downloads.mysql.com/archives/c-j/

a) 添加数据:English:45 Math:89 Computer:100

scofield	45	89	100

```
import java.sql.*;
public class mysql_test {

    /**
     * @param args
     */
    //JDBC DRIVER and DB
    static final String  DRIVER="com.mysql.jdbc.Driver";
    static final String DB="jdbc:mysql://localhost/test";
    //Database auth
    static final String USER="root";
    static final String PASSWD="root";

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Connection conn=null;
        Statement stmt=null;
        try {
            //加载驱动程序
            Class.forName(DRIVER);
            System.out.println("Connecting to a selected
database...");
            //打开一个连接
            conn=DriverManager.getConnection(DB, USER, PASSWD);
            //执行一个查询
            stmt=conn.createStatement();
            String sql="insert into student
values('scofield',45,89,100)";
            stmt.executeUpdate(sql);
            System.out.println("Inserting records into the
table successfully!");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally
        {
            if(stmt!=null)
                try {
                    stmt.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            if(conn!=null)
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
    }
    }
```

Eclipse 控制台输出如下:

```
Connecting to a selected database...
Inserting records into the table successfully!
```

插入数据之后,MySQL 客户度查询结果如下:

```
mysql> select * from student;
+----------+---------+------+----------+
| name     | English | Math | Computer |
+----------+---------+------+----------+
| zhangsan |      69 |   86 |       77 |
| lisi     |      55 |   95 |       88 |
| scofield |      45 |   89 |      100 |
+----------+---------+------+----------+
3 rows in set (0.00 sec)
```

b) 获取 scofield 的 English 成绩信息

```
import java.sql.*;
public class mysql_qurty {

 /**
  * @param args
  */
 //JDBC DRIVER and DB
 static final String  DRIVER="com.mysql.jdbc.Driver";
 static final String DB="jdbc:mysql://localhost/test";
 //Database auth
 static final String USER="root";
 static final String PASSWD="root";

 public static void main(String[] args) {
     // TODO Auto-generated method stub
     Connection conn=null;
     Statement stmt=null;
     ResultSet rs=null;
     try {
```

```
            //加载驱动程序
            Class.forName(DRIVER);
            System.out.println("Connecting to a selected
database...");
            //打开一个连接
            conn=DriverManager.getConnection(DB, USER, PASSWD);
            //执行一个查询
            stmt=conn.createStatement();
            String sql="select name,English from student where
name='scofield' ";
            //获得结果集
            rs=stmt.executeQuery(sql);
            System.out.println("name"+"\t\t"+"English");
            while(rs.next())
            {
                System.out.print(rs.getString(1)+"\t\t");
                System.out.println(rs.getInt(2));
            }
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally
        {
            if(rs!=null)
                try {
                    rs.close();
                } catch (SQLException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
            if(stmt!=null)
                try {
                    stmt.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            if(conn!=null)
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
    }
   }
```

Eclipse 控制台输出如下:

```
Connecting to a selected database...
name        English
scofield    45
```

3.2 HBase 数据库操作

Student 学生表

	score

name	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88

1. 根据上面给出的表格，用 Hbase Shell 模式设计 student 学生表格。

解答：创建 Student 表格的命令为：create 'student','score'

插入上面表格数据命令为：

put 'student','zhangsan','score:English','69'
put 'student','zhangsan','score:Math','86'
put 'student','zhangsan','score:Computer','77'
put 'student','lisi','score:English','55'
put 'student','lisi','score:Math','100'
put 'student','lisi','score:Computer','88'

上述命令执行结果为：

```
hbase(main):018:0> create 'student','score'
0 row(s) in 2.2800 seconds

=> Hbase::Table - student
hbase(main):019:0> put 'student','zhangsan','score:English','69'
0 row(s) in 0.0240 seconds

hbase(main):020:0> put 'student','zhangsan','score:Math','86'
0 row(s) in 0.0070 seconds

hbase(main):021:0> put 'student','zhangsan','score:Computer','77'
0 row(s) in 0.0060 seconds

hbase(main):022:0> put 'student','lisi','score:English','55'
0 row(s) in 0.0100 seconds

hbase(main):023:0> put 'student','lisi','score:Math','100'
0 row(s) in 0.0080 seconds

hbase(main):024:0> put 'student','lisi','score:Computer','88'
0 row(s) in 0.0080 seconds
```

a) 设计完后，用 scan 指令浏览表的相关信息，给出截图。

解答：命令为： scan 'student'

```
hbase(main):001:0> scan 'student'
ROW                    COLUMN+CELL
 lisi                  column=score:Computer, timestamp=1462605149677, value=88
 lisi                  column=score:English, timestamp=1462605127827, value=55
 lisi                  column=score:Math, timestamp=1462605138004, value=100
 zhangsan              column=score:Computer, timestamp=1462605105787, value=77
 zhangsan              column=score:English, timestamp=1462605086516, value=69
 zhangsan              column=score:Math, timestamp=1462605096683, value=86
2 row(s) in 0.3410 seconds
```

b) 查询 zhangsan 的 Computer 成绩,给出截图。

解答：命令为：get 'student','zhangsan','score:Computer'

```
hbase(main):002:0> get 'student','zhangsan','score:Computer'
COLUMN                 CELL
 score:Computer        timestamp=1462605105787, value=77
1 row(s) in 0.0610 seconds
```

c) 修改 lisi 的 Math 成绩，改为 95,给出截图。

解答：命令为 put 'student','lisi','score:Math','95'

```
hbase(main):003:0> put 'student','lisi','score:Math','95'
0 row(s) in 0.1020 seconds

hbase(main):004:0> get 'student','lisi','score:Math'
COLUMN                          CELL
 score:Math                     timestamp=1462605841339, value=95
1 row(s) in 0.0090 seconds
```

2. 根据上面已经设计出的 student，用 Hbase API 编程。
   a) 添加数据：English:45 Math:89 Computer:100

scofield	45	89	100

附：HBase 不需要另外下载驱动，可直接使用 hbase/lib 下的 jar 包编程

```
    import java.io.IOException;
  import org.apache.hadoop.conf.Configuration;
  import org.apache.hadoop.hbase.HBaseConfiguration;
  import org.apache.hadoop.hbase.TableName;
  import org.apache.hadoop.hbase.client.Admin;
  import org.apache.hadoop.hbase.client.Connection;
  import org.apache.hadoop.hbase.client.ConnectionFactory;
  import org.apache.hadoop.hbase.client.Put;
  import org.apache.hadoop.hbase.client.Table;


  public class hbase_insert {

    /**
     * @param args
     */
      public static Configuration configuration;
        public static Connection connection;
        public static Admin admin;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
         configuration  = HBaseConfiguration.create();

configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
            try{
                connection =
ConnectionFactory.createConnection(configuration);
                admin = connection.getAdmin();
            }catch (IOException e){
                e.printStackTrace();
            }
            try {
            insertRow("student","scofield","score","English","45");
            insertRow("student","scofield","score","Math","89");

insertRow("student","scofield","score","Computer","100");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
            close();
    }
     public static void insertRow(String tableName,String
rowKey,String colFamily,
              String col,String val) throws IOException {
            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();
        }
       public static void close(){
                try{
                    if(admin != null){
                        admin.close();
                    }
                    if(null != connection){
                        connection.close();
                    }
                }catch (IOException e){
                    e.printStackTrace();
                }
            }
   }
```

可以用 scan 输出数据库数据检验是否插入成功：

```
hbase(main):005:0> scan 'student'
ROW                     COLUMN+CELL
 lisi                   column=score:Computer, timestamp=1462605149677, value=88
 lisi                   column=score:English, timestamp=1462605127827, value=55
 lisi                   column=score:Math, timestamp=1462605841339, value=95
 scofield               column=score:Computer, timestamp=1462607153886, value=100
 scofield               column=score:English, timestamp=1462607153858, value=45
 scofield               column=score:Math, timestamp=1462607153883, value=89
 zhangsan               column=score:Computer, timestamp=1462605105787, value=77
 zhangsan               column=score:English, timestamp=1462605086516, value=69
 zhangsan               column=score:Math, timestamp=1462605096683, value=86
3 row(s) in 0.0390 seconds
```

b) 获取 scofield 的 English 成绩信息

```
  import java.io.IOException;
  import org.apache.hadoop.conf.Configuration;
  import org.apache.hadoop.hbase.Cell;
  import org.apache.hadoop.hbase.CellUtil;
  import org.apache.hadoop.hbase.HBaseConfiguration;
  import org.apache.hadoop.hbase.TableName;
  import org.apache.hadoop.hbase.client.Admin;
  import org.apache.hadoop.hbase.client.Connection;
  import org.apache.hadoop.hbase.client.ConnectionFactory;
  import org.apache.hadoop.hbase.client.Get;
  import org.apache.hadoop.hbase.client.Put;
  import org.apache.hadoop.hbase.client.Result;
  import org.apache.hadoop.hbase.client.Table;


  public class hbase_query {

    /**
     * @param args
     */
      public static Configuration configuration;
        public static Connection connection;
        public static Admin admin;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
         configuration  = HBaseConfiguration.create();

configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
```

```
            try{
                connection =
ConnectionFactory.createConnection(configuration);
                admin = connection.getAdmin();
            }catch (IOException e){
                e.printStackTrace();
            }
            try {
            getData("student","scofield","score","English");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
            close();
    }
      public static void getData(String tableName,String
rowKey,String colFamily,
             String col)throws  IOException{
            Table table =
connection.getTable(TableName.valueOf(tableName));
            Get get = new Get(rowKey.getBytes());
            get.addColumn(colFamily.getBytes(),col.getBytes());
            Result result = table.get(get);
            showCell(result);
            table.close();
        }
      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))+" ");
            }
        }
      public static void close(){
              try{
                  if(admin != null){
                      admin.close();
                  }
                  if(null != connection){
                      connection.close();
                  }
              }catch (IOException e){
                  e.printStackTrace();
              }
          }
  }
```

Eclipse 控制台输出如下：

```
RowName:scofield
Timetamp:1462607153858
column Family:score
row Name:English
value:45
```

3.3 Redis 数据库操作

Student 键值对:

zhangsan: {
English: 69
Math: 86
Computer: 77
}
lisi: {
English: 55
Math: 100
Computer: 88
}

1. 根据上面给出的键值对，用 Redis 的哈希结构设计出上述表格;(键值可以用 student.zhangsan,student.lisi 来表示两个键值属于同一个表格)

解答:插入上述键值对命令为:

```
127.0.0.1:6379> hset student.zhangsan English 69
(integer) 1
127.0.0.1:6379> hset student.zhangsan Math 86
(integer) 1
127.0.0.1:6379> hset student.zhangsan Computer 77
(integer) 1

127.0.0.1:6379> hset student.lisi English 55
(integer) 1
127.0.0.1:6379> hset student.lisi Math 100
(integer) 1
127.0.0.1:6379> hset student.lisi Computer 88
(integer) 1
```

a) 设计完之后,用 hgetall 命令分别输出 zhangsan 和 lisi 的成绩信息,并截图;

解答:查询 zhangsan 和 lisi 成绩信息命令为:

hgetall student.zhangsan

```
127.0.0.1:6379> hgetall student.zhangsan
1) "English"
2) "69"
3) "Math"
4) "86"
5) "Computer"
6) "77"
```

hgetall student.lisi

```
127.0.0.1:6379> hgetall student.lisi
1) "English"
2) "55"
3) "Math"
4) "100"
5) "Computer"
6) "88"
```

b) 用 hget 命令查询 zhangsan 的 Computer 成绩,给出截图。

解答:命令为: hget student.zhangsan Computer

```
127.0.0.1:6379> hget student.zhangsan Computer
"77"
```

c) 修改 lisi 的 Math 成绩，改为 95,给出截图。

解答:命令为:hset student.lisi Math 95

```
127.0.0.1:6379> hset student.lisi Math 95
(integer) 0
```

2. 根据上面已经设计出的 student 表格,用 Redis 的 JAVA 客户端编程(jedis)

附:jedis 下载地址:https://github.com/xetorthio/jedis

a) 添加数据：English:45 Math:89 Computer:100

```
scofield: {
        English: 45
        Math: 89
        Computer: 100
}
```

代码如下:

```
import java.util.Map;
import redis.clients.jedis.Jedis;

public class jedis_test {

   /**
    * @param args
    */
   public static void main(String[] args) {
       // TODO Auto-generated method stub
       Jedis jedis = new Jedis("localhost");
       jedis.hset("student.scofield", "English","45");
       jedis.hset("student.scofield", "Math","89");
       jedis.hset("student.scofield", "Computer","100");
       Map<String,String>  value =
jedis.hgetAll("student.scofield");
       for(Map.Entry<String, String>
entry:value.entrySet())
       {
           System.out.println(entry.getKey()
+":"+entry.getValue());
       }
   }
}
```

Eclipse 截图如下, 说明插入成功

```
Computer:100
Math:89
English:45
```

b) 获取 scofield 的 English 成绩信息

代码如下:

```
import java.util.Map;
import redis.clients.jedis.Jedis;

public class jedis_query {

 /**
```

```
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Jedis jedis = new Jedis("localhost");
        String value=jedis.hget("student.scofield", "English");
        System.out.println("scofield's English score is:
"+value);
    }
}
```

Eclipse 控制台输出查询到的信息:

```
scofield's English score is:   45
```

3.4 MongoDB 数据库操作

Student 文档如下:

```
{
        “name”: “zhangsan”,
        “score”: {
            “English”: 69,
            “Math”: 86,
            “Computer”: 77
        }
}
{
        “name”: “lisi”,
        “score”: {
            “English”: 55,
            “Math”: 100,
            “Computer”: 88
        }
}
```

1. 根据上面给出的文档,用 Mongo shell 设计出 student 集合.

解答:首先切换到 student 集合: use student

其次定义包含上述两个文档的数组:

```
var stus=[
{"name":"zhangsan","scores":{"English":69,"Math":86,"Computer":77}},
{"name":"lisi","score":{"English":55,"Math":100,"Computer":88}} ]
```

最后调用 db.student.insert(stus)插入数据库;

执行结果如下:

```
> use student
switched to db student
> var stus=[
... {"name":"zhangsan","scores":{"English":69,"Math":86,"Computer":77}},
... {"name":"lisi","score":{"English":55,"Math":100,"Computer":88}}
... ]
> db.student.insert(stus)
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

a) 设计完后,用 find()方法输出两个学生的信息,给出截图;

解答:命令为: db.student.find().pretty()

```
> db.student.find().pretty()
{
        "_id" : ObjectId("572daa49ac079cb68a23068e"),
        "name" : "zhangsan",
        "scores" : {
                "English" : 69,
                "Math" : 86,
                "Computer" : 77
        }
}
{
        "_id" : ObjectId("572daa49ac079cb68a23068f"),
        "name" : "lisi",
        "score" : {
                "English" : 55,
                "Math" : 100,
                "Computer" : 88
        }
}
```

b) 用 find 函数查询 zhangsan 的所有成绩(只显示 score 列),给出截图。

解答:命令为 db.student.find({"name":"zhangsan"},{"_id":0,"name":0})

```
> db.student.find({"name":"zhangsan"},{"_id":0,"name":0})
{ "scores" : { "English" : 69, "Math" : 86, "Computer" : 77 } }
```

c) 修改 lisi 的 Math 成绩，改为 95,给出截图。

解答:命令为: db.student.update({"name":"lisi"}, {"$set":{"score.Math":95}})

```
> db.student.update({"name":"lisi"}, {"$set":{"score.Math":95}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find({"name":"lisi"},{"_id":0})
{ "name" : "lisi", "score" : { "English" : 55, "Math" : 95, "Computer" : 88 } }
```

2. 根据上面已经设计出的 student 集合,用 MongoDB 的 JAVA 客户端编程

附:MongoDB JAVA 驱动下载地址为:

http://central.maven.org/maven2/org/mongodb/mongo-java-driver/3.2.2/mongo-java-driver-3.2.2.jar

a) 添加数据:English:45 Math:89 Computer:100

{

“name”: “scofield”,
“score”: {
“English”: 45,
“Math”: 89,
“Computer”: 100
}
}

代码如下:

```
import java.util.ArrayList;
import java.util.List;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class mongo_insert {

 /**
  * @param args
  */
 public static void main(String[] args) {
     // TODO Auto-generated method stub
     //实例化一个mongo客户端
     MongoClient  mongoClient=new
MongoClient("localhost",27017);
     //实例化一个mongo数据库
     MongoDatabase mongoDatabase =
mongoClient.getDatabase("student");
     //获取数据库中某个集合
     MongoCollection<Document> collection =
mongoDatabase.getCollection("student");
     //实例化一个文档,内嵌一个子文档
     Document document=new Document("name","scofield").
            append("score", new Document("English",45).
                    append("Math", 89).
                    append("Computer", 100));
     List<Document> documents = new ArrayList<Document>();
         documents.add(document);
         //将文档插入集合中
         collection.insertMany(documents);
         System.out.println("文档插入成功");
 }
```

通过以下截图,可以检测数据已经正确插入 MongoDB 数据库中

```
> db.student.find().pretty()
{
        "_id" : ObjectId("572daa49ac079cb68a23068e"),
        "name" : "zhangsan",
        "scores" : {
                "English" : 69,
                "Math" : 86,
                "Computer" : 77
        }
}
{
        "_id" : ObjectId("572daa49ac079cb68a23068f"),
        "name" : "lisi",
        "score" : {
                "English" : 55,
                "Math" : 95,
                "Computer" : 88
        }
}
{
        "_id" : ObjectId("572db5296381924c1aacc9e4"),
        "name" : "scofield",
        "score" : {
                "English" : 45,
                "Math" : 89,
                "Computer" : 100
        }
}
```

b) 获取 scofield 的所有成绩成绩信息(只显示 score 列)

代码如下;

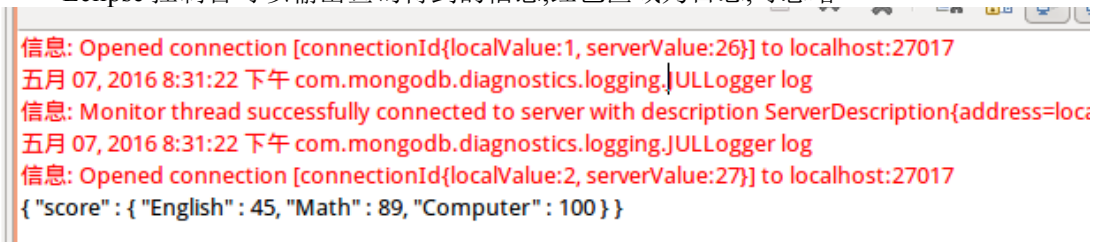
```
import java.util.ArrayList;
import java.util.List;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import static com.mongodb.client.model.Filters.eq;
public class mongo_query {

 /**
  * @param args
  */
 public static void main(String[] args) {
     // TODO Auto-generated method stub
     //实例化一个mongo客户端
     MongoClient  mongoClient=new
MongoClient("localhost",27017);
     //实例化一个mongo数据库
     MongoDatabase mongoDatabase =
mongoClient.getDatabase("student");
     //获取数据库中某个集合
     MongoCollection<Document> collection =
```

```
mongoDatabase.getCollection("student");
      //进行数据查找,查询条件为 name=scofield, 对获取的结果集只显示 score
这个域
      MongoCursor<Document>  cursor=collection.find( new
Document("name","scofield")).
              projection(new Document("score",1).append("_id",
0)).iterator();
      while(cursor.hasNext())
          System.out.println(cursor.next().toJson());
  }
  }
```

Eclipse 控制台可以输出查询得到的信息,红色区域为日志,可忽略



4 实训报告

《大数据技术原理与应用》课程机房上机实训报告				
题目：		姓名		日期
实训环境：				
实训内容与完成情况：				
出现的问题：				
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：				

实训 5 MapReduce 编程初级实践

1. 实训目的

1. 通过实训掌握基本的 MapReduce 编程方法；

2. 掌握用 MapReduce 解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

2. 实训平台

已经配置完成的 Hadoop 伪分布式环境。

Ubuntu 下 Hadoop 伪分布式环境配置：http://dblab.xmu.edu.cn/blog/install-hadoop-in-centos/

Ubuntu 下使用 Eclipse 编译运行 MapReduce 程序示例：
http://dblab.xmu.edu.cn/blog/hadoop-build-project-using-eclipse/

本实训运行环境为

Ubuntu14.04

Hadoop2.7.1

3. 实训内容和要求

1. 编程实现文件合并和去重操作

对于两个输入文件，即文件 A 和文件 B，请编写 MapReduce 程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件 C。下面是输入文件和输出文件的一个样例供参考。

输入文件 A 的样例如下：

```
20150101    x
20150102    y
20150103    x
20150104    y
20150105    z
20150106    x
```

输入文件 B 的样例如下：

```
20150101    y
20150102    y
20150103    x
20150104    z
20150105    y
```

根据输入文件 A 和 B 合并得到的输出文件 C 的样例如下：

```
20150101    x
20150101    y
20150102    y
20150103    x
20150104    y
20150104    z
```

```
20150105    y
20150105    z
20150106    x
```

答案：

```
package com.Merge;

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 {

    /**
     * @param args
     * 对A,B两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件C
     */
    //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{

        // TODO Auto-generated method stub
        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);
    }

}
```

2. 编写程序实现对输入文件的排序

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。下面是输入文件和输出文件的一个样例供参考。

输入文件 1 的样例如下：

```
33
37
12
40
```

输入文件 2 的样例如下：

```
4
16
39
5
```

输入文件 3 的样例如下：

```
1
45
25
```

根据输入文件 1、2 和 3 得到的输出文件如下：

```
1 1
2 4
3 5
4 12
```

```
5 16
6 25
7 33
8 37
9 39
10 40
11 45
```

答案：

```
package com.MergeSort;

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 {

    /**
     * @param args
     * 输入多个文件，每个文件中的每行内容均为一个整数
     * 输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数
     */
    //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{
        // TODO Auto-generated method stub
        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 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);

    }

}
```

3. 对给定的表格进行信息挖掘

下面给出一个 child-parent 的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。

输入文件内容如下：

```
child parent
Steven Lucy
Steven Jack
Jone Lucy
Jone Jack
Lucy Mary
Lucy Frank
Jack Alice
Jack Jesse
David Alice
David Jesse
Philip David
Philip Alma
Mark David
Mark Alma
```

输出文件内容如下：

```
grandchild  grandparent
Mark Jesse
Mark Alice
Philip      Jesse
Philip      Alice
Jone  Jesse
Jone  Alice
Steven      Jesse
Steven      Alice
Steven      Frank
Steven      Mary
Jone  Frank
Jone  Mary
```

答案：

```
package com.simple_data_mining;

import java.io.IOException;
import java.util.*;

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 simple_data_mining {
    public static int time = 0;

    /**
     * @param args
     * 输入一个 child-parent 的表格
     * 输出一个体现 grandchild-grandparent 关系的表格
     */
    //Map 将输入文件按照空格分割成 child 和 parent，然后正序输出一次作为右表，反序输出一次作为左表，需要注意的是在输出的 value 中必须加上左右表区别标志
    public static class Map extends Mapper<Object, Text, Text, Text>{
        public void map(Object key, Text value, Context context) throws IOException,InterruptedException{
            String child_name = new String();
            String parent_name = new String();
            String relation_type = new String();
            String line = value.toString();
            int i = 0;
            while(line.charAt(i) != ' '){
                i++;
            }
            String[] values = {line.substring(0,i),line.substring(i+1)};
            if(values[0].compareTo("child") != 0){
                child_name = values[0];
                parent_name = values[1];
                relation_type = "1";//左右表区分标志
                context.write(new Text(values[1]), new Text(relation_type+"+"+child_name+"+"+parent_name));
                //左表
                relation_type = "2";
                context.write(new Text(values[0]), new Text(relation_type+"+"+child_name+"+"+parent_name));
                //右表
            }
```

```
        }
    }

    public static class Reduce extends Reducer<Text, Text, Text, Text>{
        public void reduce(Text key, Iterable<Text> values,Context context)
throws IOException,InterruptedException{
            if(time == 0){   //输出表头
                context.write(new Text("grand_child"), new
Text("grand_parent"));
                time++;
            }
            int grand_child_num = 0;
            String grand_child[] = new String[10];
            int grand_parent_num = 0;
            String grand_parent[]= new String[10];
            Iterator ite = values.iterator();
            while(ite.hasNext()){
                String record = ite.next().toString();
                int len = record.length();
                int i = 2;
                if(len == 0) continue;
                char relation_type = record.charAt(0);
                String child_name = new String();
                String parent_name = new String();
                //获取 value-list 中 value 的 child

                while(record.charAt(i) != '+'){
                    child_name = child_name + record.charAt(i);
                    i++;
                }
                i=i+1;
                //获取 value-list 中 value 的 parent
                while(i<len){
                    parent_name = parent_name+record.charAt(i);
                    i++;
                }
                //左表，取出 child 放入 grand_child
                if(relation_type == '1'){
                    grand_child[grand_child_num] = child_name;
                    grand_child_num++;
                }
                else{//右表，取出 parent 放入 grand_parent
                    grand_parent[grand_parent_num] = parent_name;
                    grand_parent_num++;
                }
            }

            if(grand_parent_num != 0 && grand_child_num != 0 ){
                for(int m = 0;m<grand_child_num;m++){
                    for(int n=0;n<grand_parent_num;n++){
```

```
                        context.write(new Text(grand_child[m]), new
Text(grand_parent[n]));
                        //输出结果
                    }
                }
            }
        }
    }
    public static void main(String[] args) throws Exception{
        // TODO Auto-generated method stub
        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,"Single table join ");
        job.setJarByClass(simple_data_mining.class);
        job.setMapperClass(Map.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);

    }

}
```

4. 实训报告

《大数据技术原理与应用》 实训报告				
题目：		姓名		日期
实训环境：				
解决问题的思路：				
实训内容与完成情况：				
出现的问题：				
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：				