

厦门大学林子雨编著《大数据技术原理与应用》教材配套实验

五个实验已经收录到林子雨编著《大数据基础编程、实验和案例教程》教材官网：

<http://dblab.xmu.edu.cn/post/bigdatappractice/>

<http://dblab.xmu.edu.cn/post/bigdata/>

附录 A 大数据课程实验答案

本部分内容为本教程相关章节实验题目的答案,仅供参考,包含了以下五个实验的答案:

- 实验一: 熟悉常用的 Linux 操作和 Hadoop 操作
- 实验二: 熟悉常用的 HDFS 操作
- 实验三: 熟悉常用的 HBase 操作
- 实验四: NoSQL 和关系数据库的操作比较
- 实验五: MapReduce 初级编程实践

为了方便读者直接使用答案中的代码,本教程官网“下载专区”的“实验答案”目录下,提供了附录 A 内容的电子书。

A.1 实验一: 熟悉常用的 Linux 操作和 Hadoop 操作

A.1.1 实验目的

Hadoop 运行在 Linux 系统上,因此,需要学习实践一些常用的 Linux 命令。本实验旨在熟悉常用的 Linux 操作和 Hadoop 操作,为顺利开展后续其他实验奠定基础。

A.1.2 实验平台

- 操作系统: Linux (建议 Ubuntu16.04);
- Hadoop 版本: 2.7.1。

A.1.3 实验步骤

(一) 熟悉常用的 Linux 操作

- cd 命令: 切换目录

(1) 切换到目录 “/usr/local”

```
$ cd /usr/local
```

(2) 切换到当前目录的上一级目录

```
$ cd ..
```

(3) 切换到当前登录 Linux 系统的用户的自己的主文件夹

```
$ cd ~
```

● ls 命令：查看文件与目录

(4) 查看目录 “/usr” 下的所有文件和目录

```
$ cd /usr  
$ ls -al
```

● mkdir 命令：新建目录

(5) 进入 “/tmp” 目录，创建一个名为 “a” 的目录，并查看 “/tmp” 目录下已经存在哪些目录

```
$ cd /tmp  
$ mkdir a  
$ ls -al
```

(6) 进入 “/tmp” 目录，创建目录 “a1/a2/a3/a4”

```
$ cd /tmp  
$ mkdir -p a1/a2/a3/a4
```

● rmdir 命令：删除空的目录

(7) 将上面创建的目录 a（在 “/tmp” 目录下面）删除

```
$ cd /tmp  
$ rmdir a
```

(8) 删除上面创建的目录 “a1/a2/a3/a4”（在 “/tmp” 目录下面），然后查看 “/tmp” 目录下面存在哪些目录

```
$ cd /tmp  
$ rmdir -p a1/a2/a3/a4  
$ ls -al
```

● cp 命令：复制文件或目录

(9) 将当前用户的主文件夹下的文件 .bashrc 复制到目录 “/usr” 下，并重命名为 bashrc1

```
$ sudo cp ~/.bashrc /usr/bashrc1
```

(10) 在目录 “/tmp” 下新建目录 test，再把这个目录复制到 “/usr” 目录下

```
$ 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 “/usr” 目录下的文件 `bashrc1` 移动到 “/usr/test” 目录下

```
$ sudo mv /usr/bashrc1 /usr/test
```

(12) 将 “/usr” 目录下的 `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 “/usr/test2” 目录下的 `bashrc1` 文件删除

```
$ sudo rm /usr/test2/bashrc1
```

(14) 将 “/usr” 目录下的 `test2` 目录删除

```
$ sudo rm -r /usr/test2
```

- cat 命令：查看文件内容

(15) 查看当前用户主文件夹下的 `.bashrc` 文件内容

```
$ cat ~/.bashrc
```

- tac 命令：反向查看文件内容

(16) 反向查看当前用户主文件夹下的 `.bashrc` 文件的内容

```
$ tac ~/.bashrc
```

- more 命令：一页一页翻动查看

(17) 翻页查看当前用户主文件夹下的 `.bashrc` 文件的内容

```
$ more ~/.bashrc
```

- head 命令：取出前面几行

(18) 查看当前用户主文件夹下 `.bashrc` 文件内容前 20 行

```
$ head -n 20 ~/.bashrc
```

(19) 查看当前用户主文件夹下 `.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) 查看当前用户主文件夹下 `.bashrc` 文件内容最后 20 行

```
$ tail -n 20 ~/.bashrc
```

(21) 查看当前用户主文件夹下 `.bashrc` 文件内容，并且只列出 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” 目录下创建一个空文件 `hello`，并查看文件时间

```
$ cd /tmp
```

```
$ touch hello
$ ls -l hello
```

(23) 修改 hello 文件，将文件时间整为 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) 在根目录 “/” 下新建文件夹 test，然后在根目录 “/” 下打包成 test.tar.gz

```
$ sudo mkdir /test
$ sudo tar -zcv -f /test.tar.gz test
```

(27) 把上面的 test.tar.gz 压缩包，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
```

- 配置环境变量

(29) 请在 “~/.bashrc” 中设置，配置 Java 环境变量

首先，使用 vim 编辑器打开文件 “~/.bashrc”，命令如下：

```
$ vim ~/.bashrc
```

然后，在该文件的最上面加入一行如下形式的语句：

```
export JAVA_HOME=JDK 安装路径
```

最后，执行如下命令使得环境变量配置生效：

```
$ source ~/.bashrc
```

(30) 查看 JAVA_HOME 变量的值

```
$ echo $JAVA_HOME
```

(二) 熟悉常用的 Hadoop 操作

(31) 使用 hadoop 用户登录 Linux 系统，启动 Hadoop（Hadoop 的安装目录为 “/usr/local/hadoop”），为 hadoop 用户在 HDFS 中创建用户目录 “/user/hadoop”；

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

(32) 接着在 HDFS 的目录 “/user/hadoop” 下，创建 test 文件夹，并查看文件列表

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

(33) 将 Linux 系统本地的 “~/.bashrc” 文件上传到 HDFS 的 test 文件夹中，并查看 test

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

(34) 将 HDFS 文件夹 test 复制到 Linux 系统本地文件系统的 “/usr/local/hadoop” 目录下。

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

A.1.4 实验报告

实验报告				
题目：		姓名		日期
实验环境：				
实验内容与完成情况：				
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）：				

A.2 实验二：熟悉常用的 HDFS 操作

A.2.1 实验目的

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

A.2.2 实验平台

- 操作系统：Linux（建议 Ubuntu16.04）；
- Hadoop 版本：2.7.1；
- JDK 版本：1.7 或以上版本；
- Java IDE：Eclipse。

A.2.3 实验步骤

（一）编程实现以下功能，并利用 Hadoop 提供的 Shell 命令完成相同任务：

（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，则由用户来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，可以使用如下命令：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -test -e text.txt
```

执行完上述命令不会输出结果，需要继续输入命令查看结果：

```
$ echo $?
```

如果结果显示文件已经存在，则用户可以选择追加到原来文件末尾或者覆盖原来文件，具体命令如下：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -appendToFile local.txt text.txt #追加到原文件末尾
$ ./bin/hdfs dfs -copyFromLocal -f local.txt text.txt #覆盖原来文件，第一种命令形式
$ ./bin/hdfs dfs -cp -f file:///home/hadoop/local.txt text.txt #覆盖原来文件，第二种命令形式
```

实际上，也可以不用上述方式，而是采用如下命令来实现：

```
$ 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
```

上述代码可视为一行代码，在终端中输入第一行代码后，代码不会立即被执行，可以继续输入第 2 行代码和第 3 行代码，直到输入 fi 以后，上述代码才会真正执行。另外，上述代码中，直接使用了 hdfs 命令，而没有给出命令的路径，因为，这里假设已经配置了 PATH 环境变量，把 hdfs 命令的路径 “/usr/local/hadoop/bin” 写入了 PATH 环境变量中。

- 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 */
        FSDDataOutputStream 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

```

$ cd /usr/local/hadoop
$ ./bin/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> remoteliterator = fs.listFiles(dirPath, true);
        /* 输出每个文件的信息 */
        while (remoteliterator.hasNext()) {
            FileStatus s = remoteliterator.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
$ hdfs dfs -rm dir1/dir2/filename    #删除文件

```

● 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

创建目录的命令如下：

```
$ hdfs dfs -mkdir -p dir1/dir2
```

删除目录的命令如下：

```
$ hdfs dfs -rmdir dir1/dir2
```

上述命令执行以后，如果目录非空，则会提示 not empty，删除操作不会执行。如果要强制删除目录，可以使用如下命令：

```
$ 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> remoteliterator = fs.listFiles(dirPath, true);
        return !remoteliterator.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

追加到原文件末尾的命令如下:

```
$ hdfs dfs -appendToFile local.txt text.txt
```

追加到原文件的开头, 在 HDFS 中不存在与这种操作对应的命令, 因此, 无法使用一条命令来完成。可以先移动到本地进行操作, 再进行上传覆盖, 具体命令如下:

```

$ 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";
        // 移动到本地
        HDFSApi.moveToLocalFile(conf, remoteFilePath, localTmpPath);
        // 创建一个新文件
        HDFSApi.touchz(conf, remoteFilePath);
        // 先写入新内容
        HDFSApi.appendContentToFile(conf, content, remoteFilePath);
        // 再写入原来内容
        HDFSApi.appendToFile(conf, localTmpPath, remoteFilePath);
        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 代码

```

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) 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();
        }
    }
}

```

（二）编程实现一个类“MyFSDatInputStream”，该类继承“org.apache.hadoop.fs.FSDatInputStream”，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。

```

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

public class MyFSDatInputStream extends FSDatInputStream {
    public MyFSDatInputStream(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();
        }
    }
}

```

（三）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。

```

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);
        }
    }
}

```

10.2.4 实验报告

实验报告				
题目：		姓名		日期
实验环境：				

实验内容与完成情况:
出现的问题:
解决方案(列出遇到的问题 and 解决办法, 列出没有解决的问题):

10.3 实验三：熟悉常用的 HBase 操作

A.3.1 实验目的

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

A.3.2 实验平台

- 操作系统: Linux;
- Hadoop 版本: 2.7.1 或以上版本;
- HBase 版本: 1.1.2 或以上版本;
- JDK 版本: 1.7 或以上版本;
- Java IDE: Eclipse。

A.3.3 实验步骤

本部分实验的完整代码 QuestionOne.java, 可以到本书官网的“下载专区”的“实验答案”目录下载。

(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) 列出 HBase 所有的表的相关信息, 例如表名;

- Shell 命令

```
hbase> list
```

- Java 代码

```
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
}
```

需要注意的是，上述代码需要放入到完整的程序代码中才能顺利执行。

(2) 在终端打印出指定的表的所有记录数据；

● Shell 命令

```
hbase> scan 's1'
```

● Java 代码

```
//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){
        printRecorder(result);
    }
    close();
}

//打印一条记录的详情
public static void printRecorder(Result result)throws IOException{
    for(Cell cell:result.rawCells()){
        System.out.print("行键: "+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；

● Shell 命令

请先在 Shell 中创建表 s1，作为示例表，命令如下：

```
hbase> create 's1','score'
```

然后，可以在 s1 中添加数据，命令如下：

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

之后，可以执行如下命令删除指定的列：

```
hbase> delete 's1','zhangsan','score:Math'
```

● Java 代码

```
//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", 'zhangsang', 'score', 'Math', '69')

//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", 'zhangsang', 'score', 'Math')

```

(4) 清空指定的表的所有记录数据;

● Shell 命令

```
hbase> truncate 's1'
```

● Java 代码

```

//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 命令

```
hbase> count 's1'
```

● Java 代码

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

(二) HBase 数据库操作

1. 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
2015003	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 Science	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

(a) 学生 Student 表

创建表的 HBase Shell 命令语句如下:

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

插入数据的 HBase Shell 命令如下:

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

(b) 课程 Course 表

创建表的 HBase Shell 命令语句如下:

```
hbase> create 'Course','C_No','C_Name','C_Credit'
```

插入数据的 HBase Shell 命令如下:

	插入数据的 HBase Shell 命令
第一行数据	<pre>put 'Course','c001','C_No','123001' put 'Course','c001','C_Name','Math' put 'Course','c001','C_Credit','2.0'</pre>
第二行数据	<pre>put 'Course','c002','C_No','123002' put 'Course','c002','C_Name','Computer' put 'Course','c002','C_Credit','5.0'</pre>
第三行数据	<pre>put 'Course','c003','C_No','123003' put 'Course','c003','C_Name','English' put 'Course','c003','C_Credit','3.0'</pre>

(c) 选课表

创建表的 HBase Shell 命令语句如下:

```
hbase> create 'SC','SC_Sno','SC_Cno','SC_Score'
```

插入数据的 HBase Shell 命令如下:

	插入数据的 HBase Shell 命令
第一行数据	<pre>put 'SC','sc001','SC_Sno','2015001' put 'SC','sc001','SC_Cno','123001'</pre>

	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'

2. 请编程实现以下功能:

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

创建表，参数 `tableName` 为表的名称，字符串数组 `fields` 为存储记录各个字段名称的数组。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。

与本小题对应的参考代码如下：

```
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();
}
```

上述代码需要放入完整的程序代码中才可以顺利执行，可以到本书官网的“下载专区”的“实验答案”目录下载完整的程序代码 `QuestionTwo.java`。

(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 `fields` 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。

本小题的参考代码如下：

```
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();
}
```

上述代码需要放入完整的程序代码中才可以顺利执行。

(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。

```

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();
}

```

A.3.4 实验报告

实验报告				
题目：		姓名		日期
实验环境：				
实验内容与完成情况：				

出现的问题:
解决方案（列出遇到的问题 and 解决办法，列出没有解决的问题）：

A.4 实验四：NoSQL 和关系数据库的操作比较

A.4.1 实验目的

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

A.4.2 实验平台

- 操作系统：Linux（建议 Ubuntu16.04）；
- Hadoop 版本：2.7.1；
- MySQL 版本：5.7.15；
- HBase 版本：1.1.2；
- Redis 版本：3.0.6；
- MongoDB 版本：3.2.6；
- JDK 版本：1.7 或以上版本；
- Java IDE：Eclipse；

A.4.3 实验步骤

（一）MySQL 数据库操作

学生表 Student

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

1. 根据上面给出的 Student 表，在 MySQL 数据库中完成如下操作：

（1）在 MySQL 中创建 Student 表，并录入数据；

创建 Student 表的 SQL 语句如下：

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

向 Student 表中插入两条记录的 SQL 语句如下：

```
insert into student values("zhangsan",69,86,77);
```

```
insert into student values("lisi",55,100,88);
```

(2) 用 SQL 语句输出 Student 表中的所有记录；
输出 Student 表中的所有记录的 SQL 语句如下：

```
select * from student;
```

上述 SQL 语句执行后的结果截图如下：

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

(3) 查询 zhangsan 的 Computer 成绩；
查询 zhangsan 的 Computer 成绩的 SQL 语句如下：

```
select name , Computer from student where name = "zhangsan";
```

上述语句执行后的结果截图如下：

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

(4) 修改 lisi 的 Math 成绩，改为 95。
修改 lisi 的 Math 成绩的 SQL 语句如下：

```
update student set Math=95 where name="lisi";
```

上述 SQL 语句执行结果截图如下：

```
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，使用 MySQL 的 JAVA 客户端编程实现以下操作：

(1) 向 Student 表中添加如下所示的一条记录：

scofield	45	89	100
----------	----	----	-----

向 Student 表添加上述记录的 Java 代码如下：

```
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();
        }
    }
}
}

```

(2) 获取 scofield 的 English 成绩信息

获取 scofield 的 English 成绩信息的 Java 代码如下：

```

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();
            }
    }
}
}
}

```

(二) HBase 数据库操作

学生表 Student

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

1. 根据上面给出的学生表 **Student** 的信息，执行如下操作：

(1) 用 Hbase Shell 命令创建学生表 Student

创建 Student 表的命令如下：

```
create 'student','score'
```

向 Student 表中插入上面表格数据的命令如下：

```
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'
```

上述命令执行结果截图如下：

```
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
```

(2) 用 scan 指令浏览 Student 表的相关信息

用 scan 指令浏览 Student 表相关信息的命令如下：

```
scan 'student'
```

上述命令执行结果截图如下：

```
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
```

(3) 查询 zhangsan 的 Computer 成绩

查询 zhangsan 的 Computer 成绩的命令如下：

```
get 'student','zhangsan','score:Computer'
```

上述命令执行结果截图如下：

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

(4) 修改 lisi 的 Math 成绩, 改为 95

修改 lisi 的 Math 成绩的命令如下:

```
put 'student','lisi','score:Math','95'
```

上述命令的执行结果截图如下:

```
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 表, 用 HBase API 编程实现以下操作:

(1) 添加数据: English:45 Math:89 Computer:100

scofield	45	89	100
----------	----	----	-----

实现添加数据的 Java 代码如下:

```
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();
        }
    }
}

```

执行完上述代码以后，可以用 `scan` 命令输出数据库数据，以检验是否插入成功，执行结果截图如下：


```
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
```

(2) 获取 scofield 的 English 成绩信息

Java 代码如下：

```
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();
    }
}
}

```

可以在 Eclipse 中执行上述代码，会在控制台中输出如下信息：

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

(三) Redis 数据库操作

Student 键值对如下:

```
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, 完成如下操作:

(1) 用 Redis 的哈希结构设计出学生表 Student (键值可以用 student.zhangsan 和 student.lisi 来表示两个键值属于同一个表);

插入上述键值对的命令如下:

```
hset student.zhangsan English 69
hset student.zhangsan Math 86
hset student.zhangsan Computer 77
hset student.lisi English 55
hset student.lisi Math 100
hset student.lisi Computer 88
```

(2) 用 hgetall 命令分别输出 zhangsan 和 lisi 的成绩信息;

查询 zhangsan 成绩信息的命令如下:

```
hgetall student.zhangsan
```

执行该命令的结果截图如下:

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

查询 lisi 成绩信息的命令如下:

```
hgetall student.lisi
```

执行该命令的结果截图如下:

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

(3) 用 hget 命令查询 zhangsan 的 Computer 成绩;
查询 zhangsan 的 Computer 成绩的命令如下:

```
hget student.zhangsan Computer
```

执行该命令的结果截图如下:

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

(4) 修改 lisi 的 Math 成绩, 改为 95。
修改 lisi 的 Math 成绩的命令如下:

```
hset student.lisi Math 95
```

执行该命令的结果截图如下:

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

2. 根据上面已经设计出的学生表 Student, 用 Redis 的 JAVA 客户端编程(jedis), 实现如下操作:

(1) 添加数据: English:45 Math:89 Computer:100

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

完成添加数据操作的 Java 代码如下:

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

public class jedis_test {

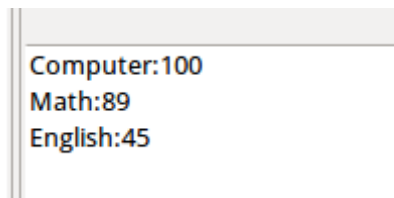
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Jedis jedis = new Jedis("localhost");
        jedis.hset("student.scofield", "English", "45");
        jedis.hset("student.scofield", "Math", "89");
        jedis.hset("student.scofield", "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());
        }
    }
}

```

在 Eclipse 中执行上述代码后，在 Eclipse 控制台输出的信息截图如下：



```

Computer:100
Math:89
English:45

```

（2）获取 scofield 的 English 成绩信息

获取 scofield 的 English 成绩信息的 Java 代码如下：

```

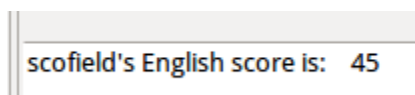
import java.util.Map;
import redis.clients.jedis.Jedis;

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);
    }
}

```

在 Eclipse 中执行上述代码后，在 Eclipse 控制台输出的信息截图如下：



```

scofield's English score is: 45

```

（四）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，完成如下操作：

(1) 用 MongoDB Shell 设计出 student 集合；

首先，切换到 student 集合，命令如下：

```
use student
```

其次，定义包含上述两个文档的数组，命令如下：

```

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

```

最后，调用如下命令插入数据库：

```
db.student.insert(stus)
```

上述命令及其执行结果的截图如下：

```

> 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" : [ ]
})

```

(2) 用 find()方法输出两个学生的信息；

用 find()方法输出两个学生信息的命令如下：

```
db.student.find().pretty()
```

上述命令及其执行结果的截图如下：

```
> 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
  }
}
```

(3) 用 find 函数查询 zhangsan 的所有成绩(只显示 score 列);

用 find 函数查询 zhangsan 的所有成绩的命令如下:

```
db.student.find({"name":"zhangsan"},{"_id":0,"name":0})
```

上述命令及其执行结果的截图如下:

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

(4) 修改 lisi 的 Math 成绩, 改为 95。

修改 lisi 的 Math 成绩的命令如下:

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

上述命令及其执行结果的截图如下:

```
> 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, 实现如下操作:

(1) 添加数据: English:45 Math:89 Computer:100

与上述数据对应的文档形式如下:

```
{
  "name": "scofield",
  "score": {
    "English": 45,
    "Math": 89,
    "Computer": 100
  }
}
```

```
}
```

实现上述添加数据操作的 Java 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");
    }
}
```

可以使用 find()方法验证数据是否已经成功插入到 MongoDB 数据库中,具体命令及其执行结果截图如下：


```
> 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
  }
}
```

(2) 获取 scofield 的所有成绩成绩信息(只显示 score 列)

Java 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());
}
}
```

A.4.4 实验报告

实验报告				
题目：		姓名		日期
实验环境：				
实验内容与完成情况：				
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）：				

A.5 实验五：MapReduce 初级编程实践

A.5.1 实验目的

- 通过实验掌握基本的 MapReduce 编程方法；
- 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。

A.5.2 实验平台

- 操作系统：Linux（建议 Ubuntu16.04）
- Hadoop 版本：2.7.1

A.5.3 实验步骤

（一）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：

20170101	x
20170102	y
20170103	x
20170104	y
20170105	z
20170106	x

输入文件 B 的样例如下：

20170101	y
20170102	y
20170103	x
20170104	z
20170105	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：

20170101	x
20170101	y
20170102	y
20170103	x
20170104	y
20170104	z
20170105	y
20170105	z
20170106	x

实现上述操作的 Java 代码如下：

```
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);
    }
}

```

(二) 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

实现上述操作的 Java 代码如下：

```
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);
}
}
    
```

（三）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
Steven	Alice
Steven	Jesse
Jone	Alice
Jone	Jesse
Steven	Mary
Steven	Frank
Jone	Mary
Jone	Frank
Philip	Alice
Philip	Jesse
Mark	Alice

Mark

Jesse

实现上述操作的 Java 代码如下：

```
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 == 'l'){
                    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]));
            }
        }
    }
}

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);
}
}

```

A.5.4 实验报告

实验报告				
题目：		姓名		日期
实验环境：				
解决问题的思路：				
实验内容与完成情况：				
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）：				