# 实验报告六

有一包含100亿条IP访问信息的文本文件，每一行是一个IPv4地址（如：192.168.1.1），要求对文件进行一遍扫描后输出所有出现过的IP地址（不能重复）。按下面三个不同的要求分别给出三个不同的实现类IPProcess1，IPProcess2，IPProcess3。需要实现的接口如下：

interface IPProcess{

void process(File in, File out) throws IOException;

}

可以添加辅助类，请添加详细的设计说明和注释来解释程序。

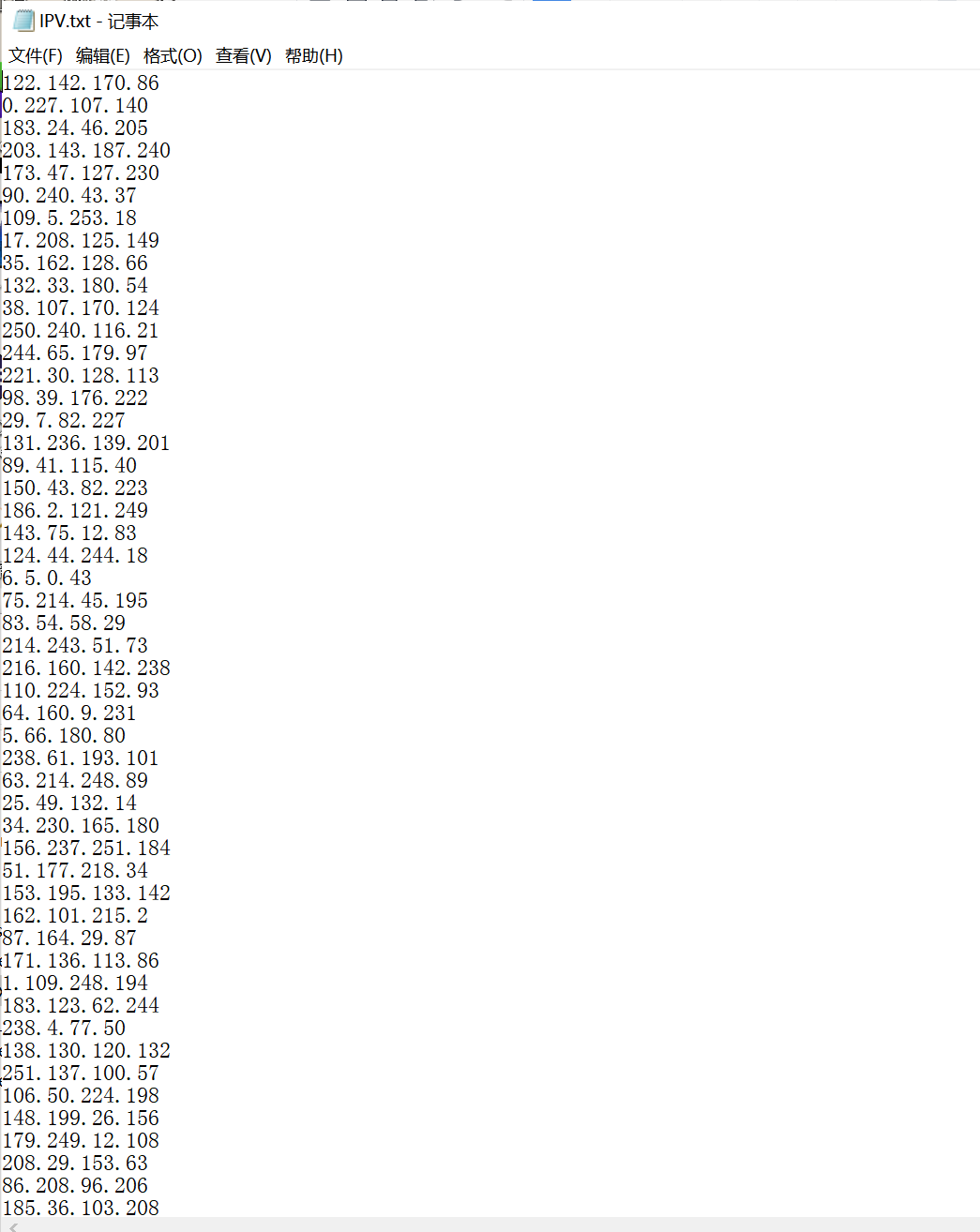
IPProcess1: 内存足够大，直接使用HashMap进行数据处理

IPProcess2: 假设可用内存为1G，在内存中实现处理过程。（提示：创建 byte数组，每一位表示一个IP地址，IPV4每位取值范围0-255，可采用位操作）

IPProcess3: 假设内存非常少，必须在文件进行操作。（提示：使用RandomAccessFile, 可以使用read()， write()，read(byte[] b) ，write(byte[] b)，seek(long pos)，getFilePointer()方法 ）

**题目解答如下：**

原IP访问信息的文本文件IPV.txt部分截图如下：



测试类实现：

**package** process;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** ceshi\_Process {

**public** **static** **void** main(String[] args) {

HashMap\_Process h = **new** HashMap\_Process();

IPProcess2 s = **new** IPProcess2();

IPProcess3 p = **new** IPProcess3();

File f1=**new** File("D:\\java\\实验六七\\process\\IPV.txt");

File f2=**new** File("D:\\java\\实验六七\\process\\new\_IPV.txt");

File f3=**new** File("D:\\java\\实验六七\\process\\process2\_IPV.txt");

File f4=**new** File("D:\\java\\实验六七\\process\\process3\_IPV.txt");

**try** {

h.process(f1, f2);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

s.process(f1, f3);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

p.process(f1, f4);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**IPProcess1:使用HashMap处理数据**

1. 题目分析

假设内存足够大，直接采用HashMap处理。

在Java中HashMap 是一个散列表，它存储的内容是键值对(key-value)映射。HashMap 实现了 Map 接口，根据键的 HashCode 值存储数据，本题在使用HashMap接口时，主要实现process接口。主要实现思路是打开文件读取内容到新建成的HashMap中，存储内容。之后应用keySet（）方法读取表中内容到keys中，逐渐读取key写入到新建立的文件中out。

1. 代码实现

**package** process;

**import** java.io.\*;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

**public** **class** HashMap\_Process **implements** IPProcess{

@Override

**public** **void** process(File in, File out) **throws** IOException {

BufferedReader bufferedReader = **new** BufferedReader(**new** FileReader(in));

Map<String,Integer> map = **new** HashMap<>();//创建新的Hash表

String str;

**while** ((str = bufferedReader.readLine()) != **null**){

//文件不为空时，向表中加入读取到得文件内容，放入到Hash表中

map.put(str, 0);

}

bufferedReader.close();

BufferedWriter bufferedWriter = **new** BufferedWriter(**new** FileWriter(out));

Set<String> keys = map.keySet();//读取Hash表中的内容

**for** (String key : keys) {//向一个空文件中写入Hash表中的内容

bufferedWriter.write(key);

bufferedWriter.newLine();

}

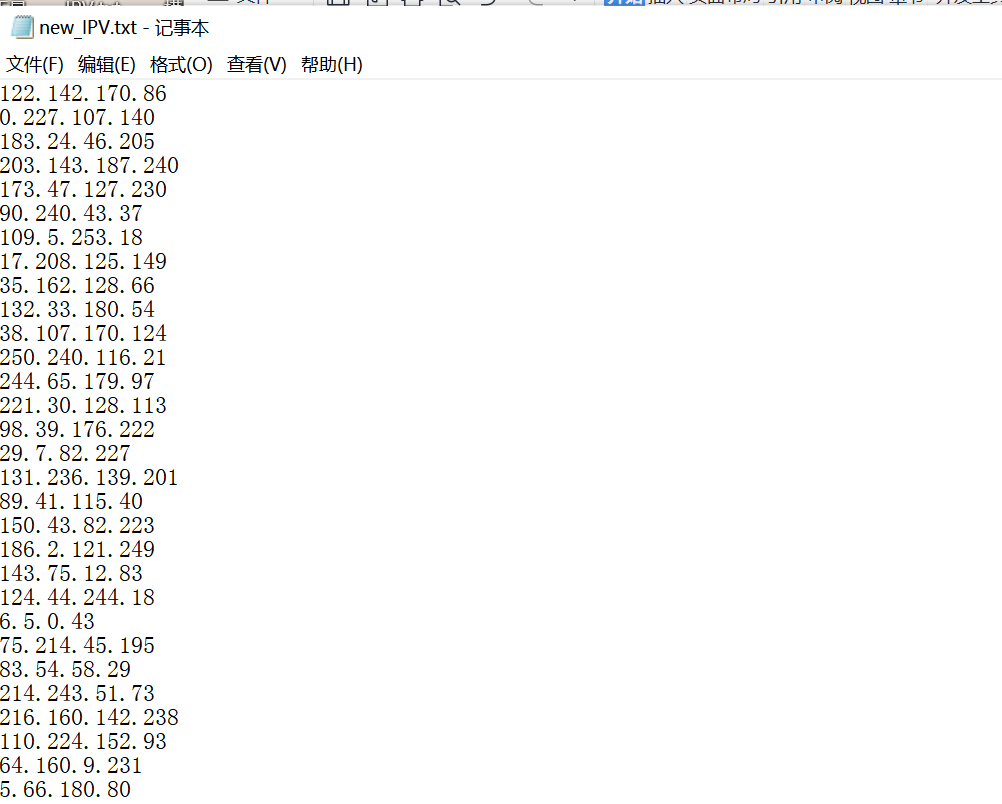
bufferedWriter.close();

}

}

1. 运行结果

将IPV.txt文本文件输出到new\_IPV.txt中，部分截图如下：



**IPProcess2：在内存中处理**

1. 题目分析

假设可用内存为1G，在内存中实现处理过程。（提示：创建 byte数组，每一位表示一个IP地址，IPV4每位取值范围0-255，可采用位操作）

用一个bit来表示一个数据。平时32位存储一个数据，我们可以换一种想法，用一个字节32位来存储0-31这32个数据。创建一个ipset用来存储ip地址。读取文件内容时，用.来分段，使用bitset处理完ip地址数据后，将其写入到新的文件当中。

1. 代码实现

**package** process;

**import** java.io.\*;

**import** java.util.BitSet;

**import** **static** java.util.BitSet.*valueOf*;

**public** **class** IPProcess2 **implements** IPProcess{

**private** BitSet ipset = **new** BitSet();

@Override

**public** **void** process(File in, File out) **throws** IOException {

BufferedReader bufferedReader = **new** BufferedReader(**new** FileReader(in));

BufferedWriter bufferedWriter = **new** BufferedWriter(**new** FileWriter(out));

String line;

**while** ((line = bufferedReader.readLine()) != **null**) {

String[] strings = line.split("\\.");

**long** ip = 0;

**for** (**int** i = 0; i < 4; i++) {

ip = (ip << 8) | Long.*parseUnsignedLong*(strings[i]);

}

//BitSet能处理的最大位数是int的最大整数，使用BitSet.valueOf()，直接将一个long类型的值转换为一个BitSet对象

BitSet ipbitset = BitSet.*valueOf*(**new** **long**[]{ip});

**if** (!ipset.intersects(ipbitset)) {

bufferedWriter.write(line);

bufferedWriter.newLine();

ipset.or(ipbitset);

}

}

bufferedWriter.close();

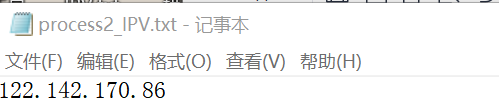
bufferedReader.close();

}

}

1. 运行结果

将IPV.txt文本文件输出到process2\_IPV.txt中，部分截图如下：



**IPProcess3：在文件中进行**

1. 题目分析

假设内存非常少，必须在文件进行操作。使用flag记录是否有重复，在while循环中，若ip地址不重复且不为空，则写入到新的文件之中。

1. 代码实现

**package** process;

**import** java.io.\*;

**import** java.util.BitSet;

**public** **class** IPProcess3 **implements** IPProcess{

@Override

**public** **void** process(File in, File out) **throws** IOException {

RandomAccessFile readIn = **new** RandomAccessFile(in, "r");

RandomAccessFile outPut = **new** RandomAccessFile(out, "rw");

String line;

**boolean** flag;//记录是否重复

readIn.seek(0);

**while** ((line = readIn.readLine())!=**null**){

flag = **false**;

outPut.seek(0);

String str;

**while** ((str = outPut.readLine())!=**null**){

**if**(line.equals(str)){

flag =**true**;

**break**;

}

}

**if**(!flag){

outPut.seek(outPut.length());

outPut.writeBytes(line + "\n");

}

}

outPut.close();

readIn.close();

}

}

1. 运行结果

