Java 課程

第12章：輸入輸出

大綱

- 檔案
- 位元串流
- 字元串流

File類別

- 不同的作業系統對於檔案系統路徑的設定各有差別
- Windows

```
"C:\\Workspace\\CH14\\"
```

- Linux

```
"/home/justin/workspace/ch14"
```

File類別

- File實例用作一個檔案或目錄的抽象表示

```
File file = new File(args[0]);
if(file.isFile()) { // 是否為檔案
    System.out.println(args[0] + " 檔案");
    System.out.print(
          file.canRead() ? "可讀 " : "不可讀 ");
    System.out.print(
          file.canWrite() ? "可寫 " : "不可寫 ");
    System.out.println(
          file.length() + "位元組");
}
```

File類別

```
else {
    // 列出所有的檔案及目錄
    File[] files = file.listFiles();
    ArrayList<File> fileList =
                            new ArrayList<File>();
    for(int i = 0; i < files.length; i++) {
        // 先列出目錄
        if(files[i].isDirectory()) { //是否為目錄
            // 取得路徑名
            System.out.println("[" +
                    files[i].getPath() + "]");
        }
        else {
            // 檔案先存入fileList，待會再列出
            fileList.add(files[i]);
        }
    }
```

File類別

```
        // 列出檔案
        for(File f: fileList) {
            System.out.println(f.toString());
        }
        System.out.println();
    }
```

RandomAccessFile類別

```
File file = new File(args[0]);
// 建立RandomAccessFile實例並以讀寫模式開啟檔案
RandomAccessFile randomAccessFile =
        new RandomAccessFile(file, "rw");

for(int i = 0; i < students.length; i++) {
  // 使用對應的write方法寫入資料
  randomAccessFile.writeChars(students[i].getName());
  randomAccessFile.writeInt(students[i].getScore());
}
```

RandomAccessFile類別

```
// 使用seek()方法操作存取位置
randomAccessFile.seek((num-1) * Student.size());
Student student = new Student();
// 使用對應的read方法讀出資料
student.setName(readName(randomAccessFile));
student.setScore(randomAccessFile.readInt());
System.out.println("姓名：" + student.getName());
System.out.println("分數：" + student.getScore());

// 設定關閉檔案
randomAccessFile.close();
```

RandomAccessFile類別

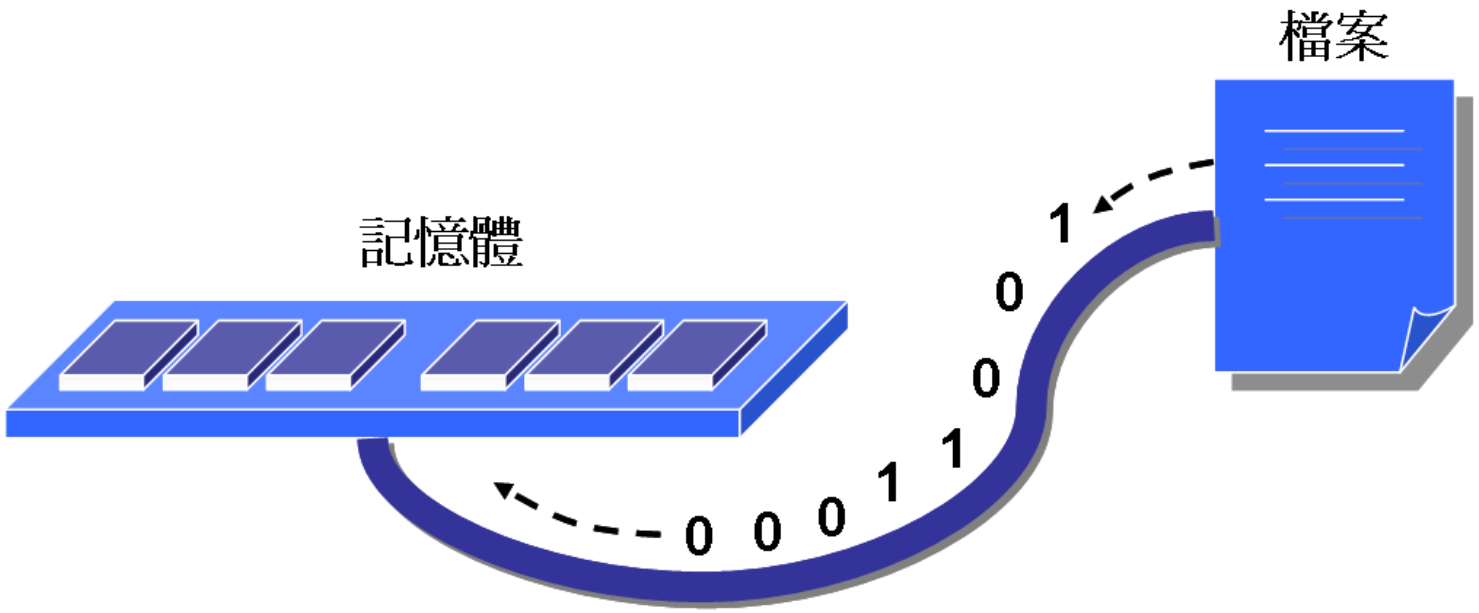
```
private static String readName(RandomAccessFile randomAccessfile)
                                throws IOException {
    char[] name = new char[15];

    for(int i = 0; i < name.length; i++)
        name[i] = randomAccessfile.readChar();
    // 將空字元取代為空白字元並傳回
    return new String(name).replace('\0', ' ');
}
```

RandomAccessFile類別

- 讀寫檔案時幾個必要的流程
  - 開啟檔案並指定讀寫方式
  - 使用對應的寫入方法
  - 使用對應的讀出方法
  - 關閉檔案

InputStream、OutputStream

- 資料流動抽象化為一個串流（Stream）



InputStream、OutputStream

- InputStream是所有表示位元輸入串流的類別之父類別
  - System中的標準輸入串流in物件就是一個InputStream類型的實例
- OutputStream是所有表示位元輸出串流的類別之父類別
  - System中的標準輸出串流物件out其類型是java.io.PrintStream，OutputStream的子類別

InputStream、OutputStream

- 很少直接操作InputStream或OutputStream上的方法，這些方法比較低階
- 通常會操作它們的子類別

```
try {
    System.out.print("輸入字元: ");
    System.out.println("輸入字元十進位表示: " +
                                System.in.read());
}
catch(IOException e) {
    e.printStackTrace();
}
```

FileInputStream、FileOutputStream

- 建立FileInputStream或FileOutputStream的實例時，必須指定檔案位置及檔案名稱，實例被建立時檔案的串流就會開啟
- 不使用串流時，您必須關閉檔案串流，以釋放與串流相依的系統資源

```
FileInputStream fileInputStream =
    new FileInputStream(new File(args[0]));
FileOutputStream fileOutputStream =
    new FileOutputStream(new File(args[1]));
…
fileInputStream.close();
fileOutputStream.close();
```

FileInputStream、FileOutputStream

```
while(true) {
    if(fileInputStream.available() < 1024) {
        // 剩餘的資料比1024位元組少
        // 一位元一位元讀出再寫入目的檔案
        int remain = -1;
        while((remain = fileInputStream.read())
                                    != -1) {
            fileOutputStream.write(remain);
        }
        break;
    }
    else {
        // 從來源檔案讀取資料至緩衝區
        fileInputStream.read(buffer);
        // 將陣列資料寫入目的檔案
        fileOutputStream.write(buffer);
    }
}
```

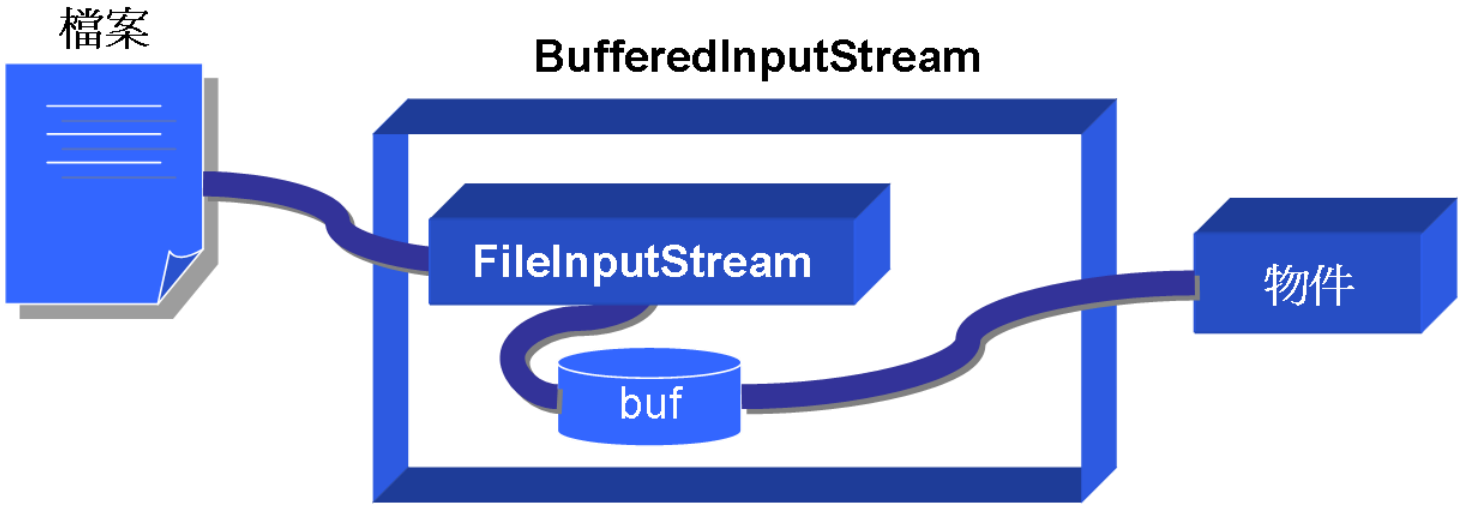
FileInputStream、FileOutputStream

- 以附加的模式來寫入檔案

```
FileOutputStream fileOutputStream =
                    new FileOutputStream(args[1], true);
```

BufferedInputStream、BufferedOutputStream

- BufferedInputStream的資料成員buf是個位元陣列，預設為2048位元組
- BufferedOutputStream的資料成員buf是個位元陣列，預設為512個位元組



BufferedInputStream、BufferedOutputStream

```
BufferedInputStream bufferedInputStream =
    new BufferedInputStream(
            new FileInputStream(srcFile));
BufferedOutputStream bufferedOutputStream =
    new BufferedOutputStream(
            new FileOutputStream(desFile));

System.out.println("複製檔案：" +
                   srcFile.length() + "位元組");
while(bufferedInputStream.read(data) != -1) {
    bufferedOutputStream.write(data);
}

// 將緩衝區中的資料全部寫出
bufferedOutputStream.flush();

// 關閉串流
bufferedInputStream.close();
bufferedOutputStream.close();
```

BufferedInputStream、BufferedOutputStream

- BufferedInputStream、BufferedOutputStream並沒有改變InputStream或 OutputStream的行為
- 只是在操作對應的方法之前，動態的為它們加上一些是緩衝區功能

DataInputStream、DataOutputStream

- 提供一些對Java基本資料型態寫入的方法

```
DataOutputStream dataOutputStream =
     new DataOutputStream(
               new FileOutputStream(args[0]));

for(Member member : members) {
    // 寫入UTF字串
    dataOutputStream.writeUTF(member.getName());
    // 寫入int資料
    dataOutputStream.writeInt(member.getAge());
}
// 出清所有資料至目的地
dataOutputStream.flush();
// 關閉串流
dataOutputStream.close();
```

DataInputStream、DataOutputStream

```
DataInputStream dataInputStream =
    new DataInputStream(
            new FileInputStream(args[0]));
// 讀出資料並還原為物件
for(int i = 0; i < members.length; i++) {
    // 讀出UTF字串
    String name = dataInputStream.readUTF();
    // 讀出int資料
    int score = dataInputStream.readInt();
    members[i] = new Member(name, score);
}
// 關閉串流
dataInputStream.close();
```

ObjectInputStream、ObjectOutputStream

- 要直接儲存物件，定義該物件的類別必須實作java.io.Serializable介面

```
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    …
}
```

- serialVersionUID代表了可序列化物件版本
- 從檔案讀回物件時兩個物件的serialVersionUID不相同的話，就會丟出java.io.InvalidClassException

ObjectInputStream、ObjectOutputStream

- 在寫入物件時，您要使用writeObject()方法
- 讀出物件時則使用readObject()方法，被讀出的物件都是以Object的型態傳回

ObjectInputStream、ObjectOutputStream

```
public static void writeObjectsToFile(
                        Object[] objs, String filename) {
    File file = new File(filename);

    try {
        ObjectOutputStream objOutputStream =
            new ObjectOutputStream(
                new FileOutputStream(file));
        for(Object obj : objs) {
            // 將物件寫入檔案
            objOutputStream.writeObject(obj);
        }
        // 關閉串流
        objOutputStream.close();
    }
    catch(IOException e) {
        e.printStackTrace();
    }
}
```

ObjectInputStream、ObjectOutputStream

```
FileInputStream fileInputStream =
    new FileInputStream(file);
ObjectInputStream objInputStream =
    new ObjectInputStream(fileInputStream);

while(fileInputStream.available() > 0) {
    list.add((User) objInputStream.readObject());
}
objInputStream.close();
```

ObjectInputStream、ObjectOutputStream

```
// 附加模式
ObjectOutputStream objOutputStream =
    new ObjectOutputStream(
        new FileOutputStream(file, true)) {
          // 如果要附加物件至檔案後
          // 必須重新定義這個方法
          protected void writeStreamHeader()
                              throws IOException {}
    };

for(Object obj : objs) {
     // 將物件寫入檔案
     objOutputStream.writeObject(obj);
}
objOutputStream.close();
```

SequenceInputStream

- 可以看作是數個 InputStream物件的組合
- 當一個InputStream物件的內容讀取完畢後，它就會取出下一個InputStream物件，直到所有的 InputStream物件都讀取完畢

SequenceInputStream

```
// 建立SequenceInputStream
// 並使用BufferedInputStream
BufferedInputStream bufInputStream =
    new BufferedInputStream(
            new SequenceInputStream(enumation),
            8192);
BufferedOutputStream bufOutputStream =
            new BufferedOutputStream(
               new FileOutputStream(filename), 8192);
byte[] data = new byte[1];
// 讀取所有檔案資料並寫入目的地檔案
while(bufInputStream.read(data) != -1)
    bufOutputStream.write(data);
bufInputStream.close();
bufOutputStream.flush();
bufOutputStream.close();
```

PrintStream

- 使用 java.io.PrintStream可以自動為您進行字元轉換的動作
- 預設會使用作業系統的編碼來處理對應的字元轉換動作

```
PrintStream printStream = new PrintStream(
        new FileOutputStream(
                new File("test.txt")));
printStream.println(1);

printStream.close();
```

ByteArrayInputStream、ByteArrayOutputStream

- ByteArrayInputStream可以將一個陣列當作串流輸入的來源
- ByteArrayOutputStream則可以將一個位元陣列當作串流輸出的目的地

PushbackInputStream

- 擁有一個PushBack緩衝區
- 從PushbackInputStream讀出資料後，只要PushBack緩衝區沒有滿，就可以使用unread()將資料推回串流的前端

Reader、Writer

- 在處理串流資料時，會根據系統預設的字元編碼來進行字元轉換
- Reader、Writer是抽象類別，在進行文字檔案的字元讀寫時真正會使用其子類別
- 可以直接在建構Reader的實例時，自行指定讀取時的編碼

```
InputStreamReader reader =
        new InputStreamReader(byteArrayStream, "Big5");
```

InputStreamReader、OutputStreamWriter

- 要對InputStream、OutputStream進行字元處理，可以使用InputStreamReader、OutputStreamWriter為加上字元處理的功能

```
FileInputStream fileInputStream =
    new FileInputStream(args[0]);
// 為FileInputStream加上字元處理功能
InputStreamReader inputStreamReader =
    new InputStreamReader(fileInputStream);
FileOutputStream fileOutputStream =
    new FileOutputStream("backup_" + args[0]);
// 為FileOutputStream加上字元處理功能
OutputStreamWriter outputStreamWriter =
    new OutputStreamWriter(fileOutputStream);
```

InputStreamReader、OutputStreamWriter

```
int ch = 0;
// 以字元方式顯示檔案內容
while((ch = inputStreamReader.read()) != -1) {
    System.out.print((char) ch);
    outputStreamWriter.write(ch);
}
System.out.println();
inputStreamReader.close();
outputStreamWriter.close();
```

- 可以自行指定字元編碼

```
InputStreamReader inputStreamReader =
        new InputStreamReader(fileInputStream, "Big5");
```

- 想要存取的是一個文字檔案，可直接使用java.io.FileReader、java.io.FileWriter類別

```
FileReader fileReader =
    new FileReader(args[0]);
FileWriter fileWriter =
    new FileWriter(args[0] + ".txt");

int in = 0;
char[] wlnChar = {'\r', '\n'};
while((in = fileReader.read()) != -1) {
    if(in == '\n') {
        // 寫入"\r\n"
        fileWriter.write(wlnChar);
    }
    else
        fileWriter.write(in); }
fileReader.close();
fileWriter.close();
```

BufferedReader、BufferedWriter

- System.in是個位元串流，為了轉換為字元串流，可使用InputStreamReader為其進行字元轉換，然後再使用BufferedReader為其增加緩衝功能

```
BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
```

BufferedReader、BufferedWriter

```
// 緩衝FileWriter字元輸出串流
BufferedWriter bufWriter =
    new BufferedWriter(new FileWriter(args[0]));

String input = null;
// 每讀一行進行一次寫入動作
while(!(input =
          bufReader.readLine()).equals("quit")) {
    bufWriter.write(keyin);
    // newLine()方法寫入與作業系統相依的換行字元
    bufWriter.newLine();
}
```

PrintWriter

- 除了接受OutputStream實例作為引數之外，PrintWriter還可以接受Writer物件作為輸出的對象

CharArrayReader、CharArrayWriter

- 可以將字元陣列當作字元資料輸出或輸入的來源

PushbackReader

- 擁有一個PushBack緩衝區，只不過PushbackReader所處理的是字元
- 只要PushBack緩衝區沒有滿，就可以使用unread()將資料回推回串流的前端