

1.StringBuffer

String的问题，一旦创建，值无法修改，如需要频繁修改字符串的花，会造成空间浪费。

为什么？

String的底层是一个char数组，数组的特点是一旦创建，长度无法修改

如何解决？

使用StringBuffer来解决，StringBuffer和String类似，都是用来创建字符串的，底层也是用数组来存储数据的，但是它的数组是可以扩容的，并且数组长度默认为16

当我们调用有参构造来创建一个StringBuffer对象的时候，数组长度就不是16了，而是根据当前的对象的值来决定，值的长度+16

得出结论：当我们使用StringBuffer来创建字符串的时候，默认会预留16个空的数组长度，让我们去执行追加操作，而不用创建新的数据，在原数组的基础上

进行修改，而不需要重写创建数组

StringBuffer创用方法

方法	描述
public StringBuffer()	创建一个空的StringBuffer，数组长度16
public StringBuffer(String str)	创建一个值为str的String Buffer，数组长度str.length+16
public int length()	返回 StringBuffer 的长度
public char charAt(int index)	返回字符串中指定位置的字符
public StringBuffer append(String str)	追加字符
public StringBuffer delete(int start,int end)	删除指定区间的数据
public StringBuffer deleteCharAt(int index)	删除指定下标的数据
public StringBuffer replace(int start,int end,String str)	将指定区间内的值替换成 str
public String substring(int start)	截取字符串从指定位置开始到结束
public String substring(int start,int end)	截取字符串从指定位置开始到指定位置结束
public StringBuffer insert(int offset,String str)	向指定位置插入 str
public int indexOf(String str)	从头开始查找指定位置的字符
public int indexOf(String str,int index)	从指定位置开始查找指定字符的位置
public StringBuffer reverse()	进行反转
public String toString()	将 StringBuffer 转为 String

```

public class Test {
    public static void main(String[] args) {
        StringBuffer stringBuffer = new StringBuffer();
        System.out.println("StringBuffer"+stringBuffer);
        System.out.println("StringBuffer的长度"+stringBuffer.length());
        StringBuffer stringBuffer1 = new StringBuffer("Hello World");
        System.out.println("StringBuffer"+stringBuffer1);
        System.out.println("下标为2的字符"+stringBuffer1.charAt(2));
        stringBuffer=stringBuffer.append("java");
        System.out.println("append之后的StringBuffer: "+stringBuffer);
        stringBuffer1=stringBuffer1.delete(3,6);
        System.out.println("delete之后的StringBuffer1"+stringBuffer1);
        stringBuffer1=stringBuffer1.deleteCharAt(3);
        System.out.println("deleteCharAt之后的StringBuffer: "+stringBuffer1);
        stringBuffer1=stringBuffer1.replace(2,3,"haha");
        System.out.println("replace之后的StringBuffer"+stringBuffer1);
        String substring=stringBuffer1.substring(2);
        System.out.println("substring之后的string: "+substring);
        substring=stringBuffer1.substring(2,4);
        System.out.println("substring之后的String"+substring);
        stringBuffer1=stringBuffer1.insert(6,"six");
        System.out.println("insert之后的String"+stringBuffer1);
        System.out.println("e的下标是"+stringBuffer1.indexOf("e"));
        stringBuffer1=stringBuffer1.reverse();
    }
}

```

```
System.out.println("reverse之后的StringBuffer"+stringBuffer1);

String string=stringBuffer1.toString();
System.out.println("StringBuffer对应的String: "+string);
```

```
StringBuffer
StringBuffer的长度0
StringBufferHello World
下标为2的字符l
append之后的StringBuffer: java
delete之后的StringBuffer1HelWorld
deleteCharAt之后的StringBuffer: Helorld
replace之后的StringBufferHehahaorld
substring之后的string: hahaorld
substring之后的Stringha
insert之后的StringHehahasixorld
e的下标是1
reverse之后的StringBufferdlroxisahaheH
StringBuffer对应的String: dlroxisahaheH
|
```

2.日期类

Date Calendar

2.1Date

```
public class Test {
    public static void main(String[] args) throws InterruptedException {
        Date date = new Date();
        System.out.println(date);
    }
}
```

可以通过SimpleDateFormat类对Date进行格式化处理，转换成希望的格式，SimpleDateFormat提供了模板标记

标记	描述
y	年, yyyy表示4位数的年份信息
M	月, MM表示2位数的月份信息
m	分钟, mm表示2位数的分钟信息
d	天, dd表示2位数的天信息
H	小时, HH表示2位数的24小时制下的小时信息
h	小时, hh表示2位数的12小时制下的小时信息
s	秒, ss表示2位数的秒信息
S	毫秒, SSS表示3位数的毫秒信息

```

public class Test {
    public static void main(String[] args) throws InterruptedException,
        ParseException {

        Date date = new Date();
        //    date 转String
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String format1 = format.format(date);
        System.out.println(format1);

        //    String转Date
        String str = "2021-12-25 21:21:21";
        Date parse = format.parse(str);
        System.out.println(parse);
    }
}

```

```

Test (2) x
D:\Java\jdk1.8.0_281\bin\java.exe ...
2021-12-25 21:28:54
Sat Dec 25 21:21:21 CST 2021

```

2.2Calendar

常用的静态常量

常量	描述
public static final int YEAR	年
public static final int MONTH	月
public static final int DAY_OF_MONTH	天, 以月为单位, 当天是该月中的第几天
public static final int DAY_OF_YEAR	天, 以年为单位, 当天是该年中的第几天
public static final int HOUR_OF_DAY	小时
public static final int MINUTE	分钟
public static final int SECOND	秒
public static final int MILLISECOND	毫秒

常用方法

方法	描述
public static Calendar getInstance()	获取Calendar实例化对象
public void set(int field,int value)	给静态常量赋值
public int get(int field)	取出静态常量
public final Date getTime()	获取Calendar对应的Date对象

```

public class Test {
    public static void main(String[] args) {
        // 计算2021-8-6所在的周是2021年的第几周
        Calendar calendar = Calendar.getInstance();
        // 输入存入calendar
        calendar.set(Calendar.YEAR,2021);
        // 1月为0
        calendar.set(Calendar.MONTH,7);
        calendar.set(Calendar.DAY_OF_MONTH,6);
        int i = calendar.get(Calendar.WEEK_OF_YEAR);
        System.out.println("2021-8-6所在的周是2021年的第" + i + "周");
        // 计算2021-8-6之后的178天的日期
        int sum=calendar.get(Calendar.DAY_OF_YEAR)+ 178;
        calendar.set(Calendar.DAY_OF_YEAR,sum);
        Date time = calendar.getTime();
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
        String format1 = simpleDateFormat.format(time);
        System.out.println(format1);

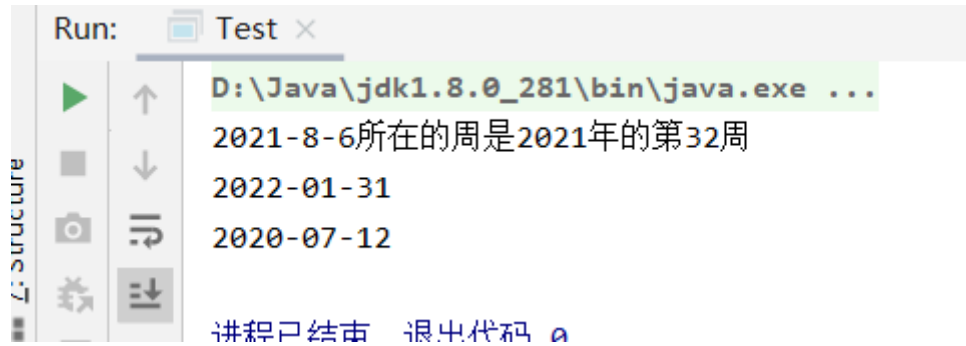
        // 计算2021-8-6之前的178天的日期
        calendar.set(Calendar.YEAR,2021);
        calendar.set(Calendar.MONTH,8);
        calendar.set(Calendar.DAY_OF_YEAR,6);
        sum= calendar.get(Calendar.DAY_OF_YEAR)-178;
        calendar.set(Calendar.DAY_OF_YEAR,sum);
        time=calendar.getTime();
        String simpleDateFormat2 = new String("yyyy-MM-dd");
        simpleDateFormat2=simpleDateFormat.format(time);
    }
}

```

```

        System.out.println(simpleDateFormat2);
    }
}

```



3.IO流

Java IO流使用Java完成输入Input、输出 Output的功能，输入是指将文件以数据流的形式读取到Java程序中，输出是指通过Java程序将数据流写入到文件中。

3.1File类

File类是Java 用来描述文件的一个类，每一个File对象都对应一个文件，常用方法如下所示。

方法	描述
public File(String pathname)	根据路径创建对象
public String getName()	获取文件名
public String getParent()	获取文件所在目录
public File getParentFile()	获取文件所在路径对应的File对象
public String getPath()	获取文件路径
public boolean exists()	判断文件是否存在
public boolean isDirectory()	判断对象是否为目录
public boolean isFile()	判断对象是否为文件
public long length()	获取文件大小
public boolean createNewFile()	根据当前对象创建新文件
public boolean delete()	删除对象
public boolean mkdir()	根据当前对象创建新目录
public boolean renameTo(File file)	为已存在的对象重命名

```

public class Test {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\DELL\\Desktop\\hello.txt");
        System.out.println(file.toString());
    }
}

```

```

        System.out.println(file.getName());
        System.out.println(file.length());
        File parentFile = file.getParentFile();
        System.out.println(parentFile.isDirectory());
        System.out.println(parentFile.isFile());
        File file1 = new File("C:\\Users\\DELL\\Desktop\\hello2.txt");
        boolean newFile=file1.createNewFile();
        System.out.println(newFile);
        File file2 = new File("C:\\Users\\DELL\\Desktop\\hello3.txt");
        System.out.println(file1.renameTo(file2));
        System.out.println(file1.delete());

    }
}

```

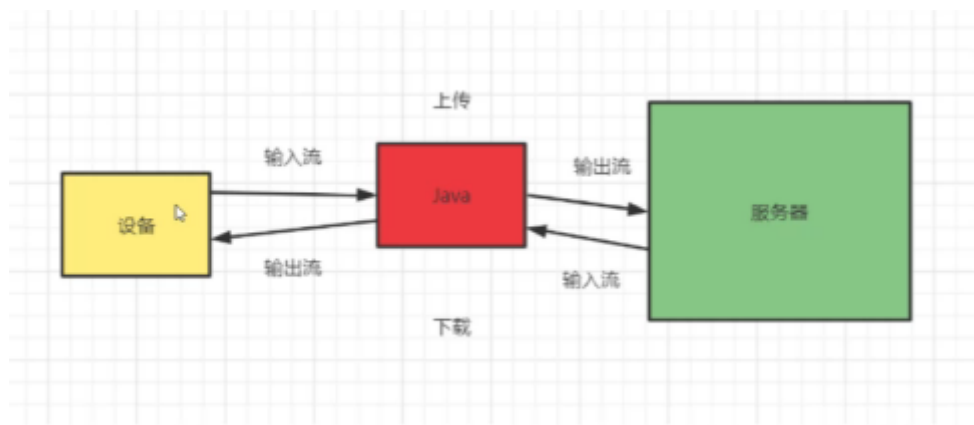
Io流就是通过数据流对文件进行操作

Input 输入流

Output 输出流

1.上传：将本地文件上传到服务器

2.下载：将服务器的文件下载到本地



流是一种先进先出的方式传输数据的序列，java中的流有很多不同的分类

- 按照方向分：可以分为输入流，输出流
- 按照单位分：可以分为字节流和字符流，字节流是指每次处理的数据是以字节为单位的，字符流是指每次处理的数据是以字符为单位的
- 按照功能来分：可以分为节点流和处理流

3.1字节流

字节流分为字节输入流，字节输出流

输入字节流 InputStream

输出字节流 OutputStream

都是抽象类

InputStream常用方法

方法	描述
public int read()	以字节为单位读取数据
public int read(byte b[])	将数据存入数组，返回数组长度
public int read(byte[],int off,int len)	将数据存入数组指定区间，返回存入数据长度
public int available()	返回当前数据流中未读取的数据个数
public void close()	关闭数据流

不能直接实例化InputStream，实例化它的子类（非抽象） FileInputStream

```
public class Test {
    public static void main(String[] args) {
        InputStream inputStream = null;
        try {
            inputStream = new
FileInputStream("C:\\Users\\DELL\\Desktop\\test.txt");
            int temp=0;
            while((temp=inputStream.read())!=-1){

                System.out.println(temp);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e){
            e.printStackTrace();
        }
        finally{
            inputStream.close();
        }
    }
}
```

英文一个字符（字母）就是一个字节

汉字，一个字符（汉字）对应3个字节，utf-8编码

```
package com.ishang.IO.InputStream;

import java.io.*;

public class Test {
    public static void main(String[] args) {

        try {
            InputStream inputStream = new
FileInputStream("C:\\Users\\DELL\\Desktop\\test.txt");
            byte[] bytes = new byte[10];
            int read = inputStream.read(bytes,2,6);
            System.out.println(read);
        }
    }
}
```

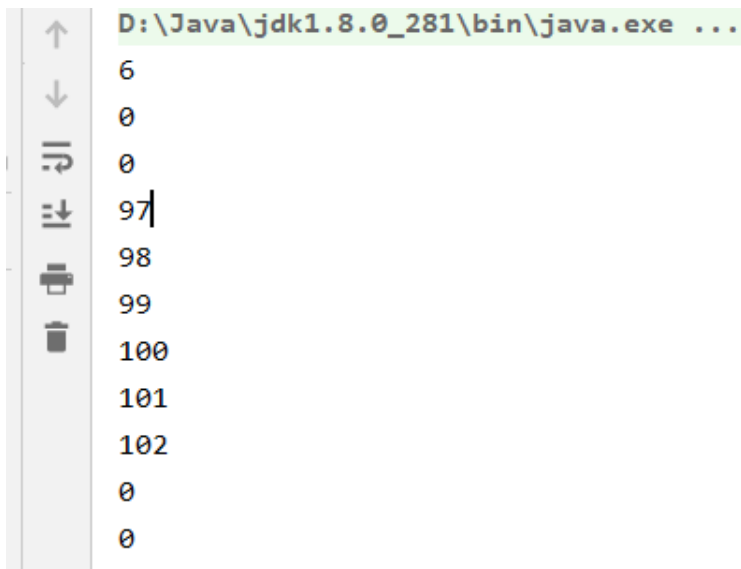


```

        for (byte aByte : bytes) {
            System.out.println(aByte);
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```



```

D:\Java\jdk1.8.0_281\bin\java.exe ...
6
0
0
97
98
99
100
101
102
0
0

```

输出字节流 OutputStream

方法	描述
public void write(int b)	以字节为单位写数据
public void write (byte b[])	将byte数组中的数据写出
public void write (byte b[],int off,int len)	将byte数组中的指定区间写出
public void flush()	强制将缓冲区的数据同步到输出流
public void close()	关闭数据流

```

package com.ishang.IO.OutputStream;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class Test {
    public static void main(String[] args) {
        OutputStream outputStream=null;
        try {
            outputStream = new
FileOutputStream("C:\\Users\\DELL\\Desktop\\test2.txt");
            outputStream.write(99);

```

```

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                outputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

当文件不存在的时候，IO流会先创建文件，再进行写入操作

```

public class Test {
    public static void main(String[] args) {
        OutputStream outputStream=null;
        try {
            outputStream = new
FileOutputStream("C:\\Users\\DELL\\Desktop\\test2.txt");
            byte[] bytes ={99,98,91,101,102};
            outputStream.write(bytes,2,3);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                outputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

图片复制:

```

package com.ishang.IO;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class Test3 {
    public static void main(String[] args) {
        //    读取原图片
        FileInputStream fileInputStream = null;
        FileOutputStream fileOutputStream = null;
        try {
            fileInputStream= new
FileInputStream("C:\\Users\\DELL\\Desktop\\123.png");

```

```

        fileOutputStream = new
FileOutputStream("C:\\Users\\DELL\\Desktop\\abc.png");
        int temp=0;
        while ((temp=fileInputStream.read())!=-1){
            System.out.println(temp);
            fileOutputStream.write(temp);
        }

    } catch (IOException e) {
        e.printStackTrace();
    }finally {
        try {
            fileInputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
}

```

数组方式复制图片

```

package com.ishang.IO;

import java.io.*;

public class Test3 {
    public static void main(String[] args) {
        // 读取原图片
        FileInputStream fileInputStream = null;
        FileOutputStream fileOutputStream = null;
        File file = new File("C:\\Users\\DELL\\Desktop\\123.png");
        try {
            fileInputStream= new FileInputStream(file);
            fileOutputStream = new
FileOutputStream("C:\\Users\\DELL\\Desktop\\7bc.png");
            byte[] bytes = new byte[(int) file.length()];
            fileInputStream.read(bytes);
            fileOutputStream.write(bytes);
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                fileInputStream.close();
                fileOutputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

3.2字符流

字节流：InputStream、OutputStream

字符流：输入字符流Reader、输出字符流Writer

3.2.1 Reader

方法	描述
public int read()	以字符为单位读取数据
public int read (char c[])	将数据存入char数组中，返回数据长度
public int read (char c[],int off,int len)	将数据存入char数组的指定区间，返回数据长度
public void close()	关闭数据流
public long transferTo(Writer out)	将数据直接读入字符输出流

英文一个字节就是一个字符

汉字3个字节等于一个字符

```
public class Test {
    public static void main(String[] args) {
        try {
            Reader reader = new FileReader("C:\\Users\\DELL\\Desktop\\a.txt");
            int temp=0;
            while((temp=reader.read())!=-1){
                System.out.println(reader);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        try {
            Reader reader = new FileReader("C:\\Users\\DELL\\Desktop\\a.txt");
            char[] chars = new char[20];
            int read = reader.read(chars,2,6);
            for (char achar : chars){
                System.out.println(achar);
            }
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

```
package com.ishang.IO.Read;
```

```
import java.io.*;
```

```

import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        Reader reader= null;
        try {
            reader = new FileReader("C:\\Users\\DELL\\Desktop\\a.txt");
            char[] chars = new char[20];
            int read = reader.read(chars,2,6);
            for (char achar : chars){
                System.out.println(achar);
            }
        }catch ( Exception e){
            e.printStackTrace();
        }finally {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

3.2.2Writer

方法	描述
public void write(int c)	以字符为单位写数据
public void write (char c[])	将char数组中的数据写出
public void write (char c[],int off,int len)	将插入数组中指定区间的数据写出
public void write (String str)	将String类型的数据写出
public void write (String str, int off,int len)	将String指定区间的数据写出
public void close()	关闭数据流
public void flush()	将缓冲区数据同步到输出流中

```

public class Test2 {
    public static void main(String[] args) throws IOException {
        Writer writer=null;
        try {
            writer = new FileWriter("C:\\Users\\DELL\\Desktop\\b.txt");
            char[] chars = {'你','好','啊'};
            writer.write(chars,1,2);
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            writer.close();
        }
    }
}

```

```

public class Test2 {
    public static void main(String[] args) throws IOException {
        Writer writer=null;
        try {
            writer = new FileWriter("C:\\Users\\DELL\\Desktop\\b.txt");
            String str="你好世界";
            writer.write(str,1,2);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            writer.close();
        }
    }
}

```

节点流处理流

字符流和字节流都是节点流，节点流是可以直接连接到文件的数据流。

处理流基于节点流，不能直接连接到文件，只能连接到节点流，进行再次处理的流

3.3处理流

输入处理流：InputStreamReader

输出处理流：OutputStreamWriter

```

public class Test {
    public static void main(String[] args) {
        InputStream inputStream = null;
        OutputStream outputStream = null;
        InputStreamReader inputStreamReader = null;
        OutputStreamWriter outputStreamWriter = null;

        try {
            inputStream = new
FileInputStream("C:\\Users\\DELL\\Desktop\\aaa.txt");
            inputStreamReader = new InputStreamReader(inputStream);
            outputStream = new
FileOutputStream("C:\\Users\\DELL\\Desktop\\bbb.txt");
            outputStreamWriter = new OutputStreamWriter(outputStream);
            int temp=0;
            while((temp=inputStreamReader.read())!=-1){
                outputStreamWriter.write(temp);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                outputStream.flush();
                outputStreamWriter.flush();
                inputStream.close();
                outputStream.close();
                inputStreamReader.close();
                outputStreamWriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

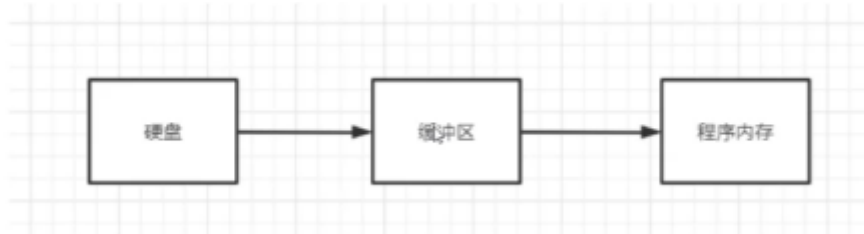
    }

    }

}

```

3.4 缓冲流



缓冲流自带缓冲区，可以一次性从硬盘读取部分数据存入缓冲区，再写入到程序内存中，提高效率，减少程序对硬盘的访问

- 字节缓冲流：
 - 字节输入缓冲流 `BufferedInputStream`
 - 字节输出缓冲流 `BufferedOutputStream`
- 字符缓冲流
 - 字符输入缓冲流 `BufferedReader`
 - 字符输出缓冲流 `BufferedWriter`

3.4.1 `BufferedInputStream`

```

package com.ishang.IO.BufferedInputStream;

import jdk.internal.util.xml.impl.Input;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

public class Test {
    public static void main(String[] args) {
        InputStream inputStream = null;
        BufferedInputStream bufferedInputStream=null;
        try {
            inputStream = new
FileInputStream("C:\\Users\\DELL\\Desktop\\aaa.txt");
            bufferedInputStream = new BufferedInputStream(inputStream);
            int temp=0;
            while((temp = bufferedInputStream.read())!=-1){
                System.out.println(temp);
            }
            bufferedInputStream.read();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {

```

```

        inputStream.close();
        bufferedInputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}

}

```

3.4.2 BufferedReader

```

package com.ishang.IO.BufferedInputStream;

import jdk.internal.util.xml.impl.Input;

import java.io.*;

public class Test {
    public static void main(String[] args) {
        Reader reader = null;
        BufferedReader bufferedReader = null;
        try {
            reader = new FileReader("C:\\Users\\DELL\\Desktop\\aaa.txt");
            bufferedReader = new BufferedReader(reader);
            String string = "";
            while((string=bufferedReader.readLine())!=null){
                System.out.println(string);
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                reader.close();
                bufferedReader.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

3.4.3BufferedOutputStream

```

package com.ishang.IO.BufferedWriter;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

```



```

import java.io.Writer;

public class Tset {
    public static void main(String[] args) {
        Writer writer = null;
        BufferedWriter bufferedWriter = null;
        try {
            writer = new FileWriter("C:\\Users\\DELL\\Desktop\\bbb.txt");
            bufferedWriter = new BufferedWriter(writer);
            String string = "你好世界";
            bufferedWriter.write(string);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                bufferedWriter.flush();
                writer.close();
                bufferedWriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

3.4.4 序列化和反序列化

序列化是指将Java内存中的对象存储到硬盘文件中

反序列化是指将硬盘的文件中存储的数据还原到Java内存中

User 实现Serializable接口

```

package com.ishang.IO.Serializable;

import java.io.Serializable;

public class User implements Serializable {
    private String name;
    private Integer id;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
}

```

```
}  
}
```

2.实现序列化操作

```
package com.ishang.IO.Serilaziabale;  
  
import java.io.*;  
  
public class Test {  
    public static void main(String[] args) {  
        User user = new User();  
        user.setId(1);  
        user.setName("张三");  
        OutputStream outputStream = null;  
        // 专门写对象的输出流  
        ObjectOutputStream objectOutputStream = null;  
        try {  
            outputStream = new  
FileOutputStream("C:\\Users\\DELL\\Desktop\\test2.txt");  
            objectOutputStream = new ObjectOutputStream(outputStream);  
            objectOutputStream.writeObject(user);  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {  
            try {  
                outputStream.flush();  
                objectOutputStream.flush();  
                outputStream.close();  
                objectOutputStream.close();  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

反序列化

```
package com.ishang.IO.Serilaziabale;  
  
import java.io.*;  
  
public class UnSerializable {  
    public static void main(String[] args) {  
        InputStream inputStream = null;  
        ObjectInputStream objectInputStream = null;  
        try {  
            inputStream = new  
FileInputStream("C:\\Users\\DELL\\Desktop\\test2.txt");  
            objectInputStream = new ObjectInputStream(inputStream);  
            Object object = objectInputStream.readObject();  
            System.out.println(object);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
    }finally {  
        try {  
            inputStream.close();  
            objectInputStream.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}  
}
```