7. 文件与IO

7.1. File类

- File类：表示文件和目录路径名的抽象表示形式
- File类可以实现文件的创建、删除、重命名、得到路径、创建时间等等，是唯一与文件本身有关的操作类
- Files类的操作方法：

public static final String separator	表示路径分隔符“\”
public File(String pathname)	构造File类实例，要传入路径
public boolean createNewFile()	创建新文件
public boolean delete()	删除文件
public boolean isDirectory()	判断给定的路径是否是文件夹
public boolean isFile()	判断给定的路径是否是文件
public String[] list()	列出文件夹中的文件
publci File[] listFiles()	列出文件夹中的所有文件
public boolean mkdir()	创建新的文件夹
public boolean renameTo(File dest)	为文件重命名
public long length()	返回文件大小
String getPath()	路径名字符串

- 示例：

```java
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;

/**
 * @author xiao儿
 * @date 2019年8月28日 上午10:15:42
 * @description File类基本使用
 */
public class FileDemo {
    public static void main(String[] args) {
        // File 类表示一个文件或目录
        File file = new File("./src/day07_文件与IO/file/file.txt");
        // File file = new File("." + File.separator + "test" +
File.separator +
        // "file.txt");
        if (!file.exists()) { // 判断文件是否存在
            try {
                file.createNewFile();// 创建文件
                System.out.println("文件创建成功");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        // 判断是否为文件夹
        System.out.println("是否为文件夹: " + file.isDirectory());

        // 创建文件夹
        File file2 = new File("./src/day07_文件与IO/file/my");
        boolean b = file2.mkdir();
        System.out.println("my文件夹创建成功: " + b);

        // 创建文件
        File file3 = new File("./src/day07_文件与IO/file/my/my.txt");
        if (!file3.exists()) {
            try {
                file3.createNewFile();
                System.out.println("my.txt文件创建成功");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        // 删除文件
        boolean b2 = file3.delete();
        System.out.println("my.txt文件是否删除: " + b2);

        // 删除文件夹时需要里面为空
        boolean b3 = file2.delete();
        System.out.println("my文件夹是否删除成功: " + b3);

        // 列出文件夹中的文件: 列出当前目录下的文件名
```

```
        File file4 = new File("./src");
        String[] lists = file4.list();
        System.out.println(Arrays.toString(lists));

        // 列出文件夹中所有的文件: 列出当前目录下的所有文件, 以 File 对象返回
        File[] listFiles = file4.listFiles();
        System.out.println(Arrays.toString(listFiles));
        for (File file5 : listFiles) {
            System.out.println("name=" + file5.getName() + "; length=" + file5.length() + "; path=" + file5.getPath()
                    + "; absolutePath=" + file5.getAbsolutePath() + "; hidden=" + file5.isHidden() + "; read="
                    + file5.canRead() + "; lastModified="
                    + new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss").format(new Date(file5.lastModified())));
        }

        // 为文件重命名
        boolean b4 = file.renameTo(new File("./src/day07_文件与IO/file/my.txt"));
        System.out.println("file.txt重命名是否成功: " + b4);
        System.out.println(file.getName());

        // 返回文件大小
        System.out.println("file.txt文件大小为: " + file.length());

        // 返回路径名字符串
        String pathname = file.getPath();
        System.out.println("file.txt的文件路径为: " + pathname);

        // 过滤器
//      File[] files = file4.listFiles(new FileFilter() {
//          @Override
//          public boolean accept(File pathname) {
//              return pathname.getName().endsWith(".md");
//          }
//      });
        File[] files = file4.listFiles((name) -> name.getName().endsWith(".md"));
        System.out.println(Arrays.toString(files));
    }
}
```

- 查找指定目录下的文件:

```
import java.io.File;

/**
 * @author xiao儿
 * @date 2019年8月28日 下午5:49:10
 * @description 在指定的目录下查找文件
 */
public class FindFile {
    public static void main(String[] args) {
        findFile(new File("./src"), ".java");
    }

```

```
    // 查找文件的方法
    private static void findFile(File target, String ext) {
        if (target == null)
            return;
        // 如果文件是目录
        if (target.isDirectory()) {
            File[] files = target.listFiles();
            if (files != null) {
                for (File file : files) {
                    findFile(file, ext);// 递归调用
                }
            }
        } else {
            // 此处表示 File 是一个文件
            String name = target.getName().toLowerCase();
            if (name.endsWith(ext)) {
                System.out.println(target.getAbsolutePath());
            }
        }
    }
}
```

7.2. IO流

- IO流：输入输出流（Input/Output）
- 流是一组有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象。即数据在两设备间的传输称为流
- 流的本质是数据传输，根据数据传输特性将流抽象为各种类，方便更直观的进行数据操作
- IO流的分类：
  - 根据处理数据类型的不同分为：字符流和字节流
  - 根据数据流向不同分为：输入流和输出流

7.3. 字节流

- 字节输出流：

```
OutputStream类定义
public abstract class OutputStream extends Object implements Closeable, Flushable
// 此抽象类是表示输出字节流的所有类的超类。输出流接收输出字节并将这些字节发送到InputStream类某个接收器要向文件中输出，使用FileOutputStream类
```

- 字节输入流：

```
InputStream类定义
public abstract class InputStream extends Object implements Closeable
// 此抽象类是表示字节输入流的所有类的超类。FiLeInputStream从文件系统中的某个文件中获得输入字节
```

- 示例：

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * @author xiao儿
 * @date 2019年8月28日 下午7:24:16
 * @description 字节输入输出流
 * 输入流：超类 InputStream，对文件的输入流使用子类：FileInputSteam
 * 输出流：超类 OutputStream，对文件的输出流使用子类：FileOutputStream
 *
 * 输入输出字节流：每次只会操作一个字节（从文件读取或写入）
 * 字节操作流，默认每次执行写入操作会直接把数据写入文件
 */
public class ByteStreamDemo {
    public static void main(String[] args) {
        out();
        in();
    }

    private static void out() {
        // 0.确定目标文件
        File file = new File("./src/day07_文件与IO/bytestream/bytestream.txt");
        // 1.构建文件输出流对象
        try {
            OutputStream outputStream = new FileOutputStream(file, true);// append 为 true表示追加内容
            // 2.输出的内容
            String info = "小河流水哗啦啦\r\n";
            // String line = System.getProperty("line.separator");// 获取换行符
            // 3.把内容写入到文件
            outputStream.write(info.getBytes());
            // 4.关闭流
            outputStream.close();
            System.out.println("write success.");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void in() {
        // 0.确定目标文件
        File file = new File("./src/day07_文件与IO/bytestream/bytestream.txt");
        // 1.创建一个输入流对象
        try {
            InputStream inputStream = new FileInputStream(file);
```

```
            byte[] bytes = new byte[1024];
            // 如果上次读取完的剩余字符不足下次的长度，后面的剩余空间将由上次的内容继续填充
            // byte[] bytes = new byte[10];
            StringBuilder builder = new StringBuilder();
            int len = -1;
            // 把数据读入到数组中，并返回读取的字节数，当不等于-1时，表示读取到数据，等于-1时表示已经读完
            while ((len = inputStream.read(bytes)) != -1) {
                // 根据读取到的字节数组，再转换为字符串内容，添加到 StringBuilder 中
                builder.append(new String(bytes));
            }
            // 输出内容
            System.out.println(builder);
            // 关闭输入流
            inputStream.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 注意：
  - 在使用字节流读取文件时需要注意：如果存储的字节数组长度小于读取的文件长度，要使用String(bytes, offset, length)这个构造方法，将byte数组转换为字符串，否则会出现多读几个字节的情况

7.4. 字符流

- Writer

```
// 写入字符流的抽象类。子类必须实现的方法仅有write(char[], int, int)、flush()和close()。但是，多数子类将重写此处定义的一些方法，一提供更高的效率和/或其他功能

// 与OutputStream一样，对文件的操作使用，FileWriter类完成
```

- Reader

```
// 用于读取字符流的抽象类。子类必须实现的方法仅有read(char[], int, int)和close()。但是，多数子类将重写此处定义的一些方法，以提供更高的效率和/或功能

// 使用FiLeReader类进行实例化操作
```

- 示例：

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

```java
import java.io.Reader;
import java.io.Writer;

/**
 * @author xiao儿
 * @date 2019年8月28日 下午9:41:01
 * @description 字符输入输出流
 *
 * 输入流：Reader，对文件的操作使用子类：FiLeReader
 * 输出流：Writer，对文件的操作使用子类：FileWriter
 *
 * 每次操作的单位是一个字符
 * 文件字符操作类会自带缓存，默认大小为1024字节，在缓存满后，或手动刷新缓存，或关闭流时会把数据写入文件
 */
public class CharStreamDemo {
    public static void main(String[] args) {
        out();
        in();
    }

    private static void out() {
        File file = new File("./src/day07_文件与IO/charstream/charstream.txt");
        try {
            Writer writer = new FileWriter(file, true);
            writer.write("小河流水哗啦啦");
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void in() {
        File file = new File("./src/day07_文件与IO/charstream/charstream.txt");
        try {
            Reader reader = new FileReader(file);
            char[] cs = new char[1];
            int len = -1;
            StringBuilder builder = new StringBuilder();
            while ((len = reader.read(cs)) != -1) {
                builder.append(new String(cs, 0, len));
            }
            reader.close();
            System.out.println(builder);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 文件的复制：

```java
import java.io.File;
```

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * @author xiao儿
 * @date 2019年8月29日 下午6:55:12
 * @description 文件的复制
 */
public class CopyFileDemo {
    public static void main(String[] args) {
        System.out.println("start copy...");
        copy("./src/day07_文件与IO/charstream/charstream.txt",
"./src/day07_文件与IO/copyfile/charstream.txt");
        System.out.println("copy success!");
    }

    private static void copy(String src, String target) {
        File srcFile = new File(src);
        File targetFile = new File(target);
        InputStream input = null;
        OutputStream output = null;
        try {
            input = new FileInputStream(srcFile);
            output = new FileOutputStream(targetFile);
            byte[] bytes = new byte[1024];
            int len = -1;
            while ((len = input.read(bytes)) != -1) {
                output.write(bytes, 0, len);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (input != null) {
                    input.close();
                }
                if (output != null) {
                    output.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

7.5. 字节流和字符流的区别和联系

- 如何选择使用字节流还是字符流?

```
// 一般操作非文本文件时，使用字节流
// 操作文本文件时，建议使用字符流
```

- 字符流的内部实现还是字节流

7.6. 字节字符转换流

- 转换流，可以将一个字节流转换为字符流，也可以将一个字符流转换为字节流

```
// 可以将输出的字符流转换为字节流的输出形式
OutputStreamWriter
// 将输入的字节流转换为字符流输入形式
InputStreamReader
```

- 示例：

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Reader;
import java.io.Writer;
import java.nio.charset.Charset;

/**
 * @author xiao儿
 * @date 2019年8月29日 下午7:01:30
 * @description 字节字符转换流
 *
 * OutputStreamWriter: 可以将输出的字符流转换为字节流的输出形式
 * InputStreamReader: 将输入的字节流转换为字符流输入形式
 */
public class ByteCharacterStream {
    public static void main(String[] args) throws FileNotFoundException
{
        InputStream input = new FileInputStream("./src/day07_文件与
IO/bytecharacterstream/bytecharacterstream.txt");
        read(input);
        OutputStream output = new FileOutputStream("./src/day07_文件与
IO/bytecharacterstream/bytecharacterstream.txt", true);
        write(output);
    }

    private static void read(InputStream input) {
        Reader reader = new InputStreamReader(input,
Charset.forName("UTF-8"));
        char[] chars = new char[1024];
        int len = -1;
        try {
```

```
            while ((len = reader.read(chars)) != -1) {
                System.out.println(new String(chars, 0, len));
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void write(OutputStream output) {
        Writer writer = new OutputStreamWriter(output,
Charset.forName("UTF-8"));
        try {
            writer.write("开开心心来玩耍");
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

7.7. 缓冲流

- 对文件或其他目标频繁的读写操作，效率低，性能差。使用缓冲流的好处是能够高效的读写信息，原理是将数据先缓冲起来，然后一起写入或者读取出来

```
BufferedInputStream：为另一个输入流添加一些功能，在创建BufferedInputStream时，会创建一个内部缓冲区数组，用于缓冲数据

BufferedOutputStream：通过设置这种输出流，应用程序就可以将各个字节写入底层输出流中，而不必针对每次字节写入调用底层系统

BufferedReader：从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取

BufferedWriter：将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入
```

- 字节缓冲流：

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * @author xiao儿
 * @date 2019年8月31日 下午8:09:18
 * @description ByteBufferStreamDemo
```

```java
 * 缓存的目的:
 * 解决在写入文件操作时，频繁的操作文件所带来的性能降低的问题
 * BufferedOutputStream 内部默认的缓存大小是8KB，每次写入时先存储到缓存中的byte数组中，
 * 当数组存满，会把数组中的数据写入文件，并且缓存下标归零
 */
public class ByteBufferStreamDemo {
    public static void main(String[] args) {
        byteWriter();
        byteReader();
        byteReader2();
    }

    private static void byteWriter() {
        File file = new File("./src/day07_文件与IO/bufferstream/bufferstream.txt");
        try {
            OutputStream output = new FileOutputStream(file);
            // 构造一个字节缓冲流
            BufferedOutputStream buffer = new BufferedOutputStream(output);
            String info = "小河流水哗啦啦";
            buffer.write(info.getBytes());
            buffer.close();
            // output.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void byteReader() {
        File file = new File("./src/day07_文件与IO/bufferstream/bufferstream.txt");
        try {
            InputStream input = new FileInputStream(file);
            BufferedInputStream buffer = new BufferedInputStream(input);
            byte[] bytes = new byte[1024];
            int len = -1;
            while ((len = buffer.read(bytes)) != -1) {
                System.out.println(new String(bytes, 0, len));
            }
            buffer.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void byteReader2() {
        File file = new File("./src/day07_文件与IO/bufferstream/bufferstream.txt");
        try (BufferedInputStream buffer = new BufferedInputStream(new FileInputStream(file))) {
            byte[] bytes = new byte[1024];
```

```
            int len = -1;
            while ((len = buffer.read(bytes)) != -1) {
                System.out.println(new String(bytes, 0, len));
            }
            buffer.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 字符缓冲流：

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;

/**
 * @author xiao儿
 * @date 2019年8月31日 下午9:24:46
 * @description CharacterBufferStreamDemo
 *
 * 字符流:
 * 1.加入字符缓存流，增强读取功能（readLine）
 * 2.更高效的读取数据
 * FileReader: 内部使用 InputStreamReader (sun.nio.cs.StreamDecoder)，解码过程: byte->char，默认缓存大小为8KB
 * BufferedReader: 默认缓存大小是8KB，但可以手动指定缓存大小，把数据直接读取到缓存中，减少每次转换过程，效率更高
 * BufferedWriter: 同上
 */
public class CharacterBufferDemo {
    private static final String pathname_eclipse = "./src/day07_文件与IO/bufferstream/bufferstream.txt";
    private static final String pathname_idea = "./Java入门/src/day07_文件与IO/bufferstream/bufferstream.txt";
    private File[] files;

    public static void main(String[] args) {
        charWriter();
        charReader();
    }

    private static void charWriter() {
        File file = new File(pathname_idea);
        try {
            Writer writer = new FileWriter(file);
            BufferedWriter buffer = new BufferedWriter(writer);
            String info = ", 村花到我家。";
```

```
            buffer.write(info);
            buffer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void charReader() {
        File file = new File(pathname_idea);
        try {
            Reader reader = new FileReader(file);
            // 为字符流提供缓存，已达到高效读取的目的
            BufferedReader buffer = new BufferedReader(reader);
            char[] chars = new char[1024];
            int len = -1;
            if ((len = buffer.read(chars)) != -1) {
                System.out.println(new String(chars, 0, len));
            }
            buffer.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

7.8. 打印流

- 打印流的主要功能是用于输出，在整个IO包中打印流分为两种类型：

```
// 字节打印流
PrintStream
// 字符打印流
PrintWriter
```

- 字节打印流：

```
import java.io.*;

/**
 * @author xiao儿
 * @date 2019/9/1 9:41
 * @Description PrintStreamDemo
 *
 * 字节打印流：在字节打印时，可以增强输出功能
 */
public class PrintStreamDemo {
    private static final String pathname_eclipse = "./src/day07_文件与IO/print/printStream.txt";
    private static final String pathname_idea = "./Java入门/src/day07_文件与IO/print/printStream.txt";

    public static void main(String[] args) {
        printStream();
    }
```

```
17
18      private static void printStream() {
19          File file = new File(pathname_idea);
20          try {
21              OutputStream out = new FileOutputStream(file);
22              // 缓存
23              BufferedOutputStream bos = new BufferedOutputStream(out);
24              // 增强打印功能
25              PrintStream ps = new PrintStream(bos);
26              ps.println("小河流水甜甜");
27              ps.close();
28          } catch (FileNotFoundException e) {
29              e.printStackTrace();
30          }
31      }
32  }
```

- 字符打印流：

```
1   import java.io.*;
2
3   /**
4    * @author xiao儿
5    * @date 2019/9/1 9:52
6    * @Description PrintWriterDemo
7    *
8    * 字符打印流：在打印字符时，可以增强打印功能
9    */
10  public class PrintWriterDemo {
11      private static final String pathname_eclipse = "./src/day07_文件与
    IO/print/printWriter.txt";
12      private static final String pathname_idea = "./Java入门/src/day07_文
    件与IO/print/printWriter.txt";
13
14      public static void main(String[] args) {
15          printWriter();
16      }
17
18      private static void printWriter() {
19          File file = new File(pathname_idea);
20          try {
21              Writer writer = new FileWriter(file);
22              BufferedWriter bw = new BufferedWriter(writer);
23              PrintWriter pw = new PrintWriter(bw);
24              pw.println("小哥你好");
25              pw.close();
26          } catch (IOException e) {
27              e.printStackTrace();
28          }
29      }
30  }
```

7.9. 对象流

- 对象流的两个类：

- ObjectOutputStream将Java对象的基本数据类型和图形写入OutputStream
- ObjectInputStream对以前使用ObjectOutputStream写入的基本数据和对象进行反序列化

```java
// Dog
import java.io.Serializable;

/**
 * @author xiao儿
 * @date 2019/9/1 10:14
 * @Description Dog
 */

// 如果一个类创建的对象需要被序列化，那么该类必须实现 Serializable 接口
// Serializable 和 Cloneable 一样是一个标记接口，没有任何定义，为了告诉JVM该类对象可以被序列化

// 什么时候对象需要被序列化?
// 1.把对象保存到文件中（存储到物理介质）
// 2.对象需要在网络上进行传输

// 如果对象没有实现 Serializable 接口，会报错误:java.io.NotSerializableException
public class Dog implements Serializable {
    private String name;
    private int age;
    private String sex;

    public Dog() {
    }

    public Dog(String name, int age, String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSex() {
```

```
49          return sex;
50      }
51
52      public void setSex(String sex) {
53          this.sex = sex;
54      }
55
56      @Override
57      public String toString() {
58          return "Dog{" +
59                  "name='" + name + '\'' +
60                  ", age=" + age +
61                  ", sex='" + sex + '\'' +
62                  '}';
63      }
64  }
65
66  // ObjectStreamDemo
67  import java.io.*;
68
69  /**
70   * @author xiao儿
71   * @date 2019/9/1 10:13
72   * @Description ObjectStreamDemo
73   */
74  public class ObjectStreamDemo {
75      private static final String pathname_eclipse = "./src/day07_文
    件与IO/objectstream/dog.obj";
76      private static final String pathname_idea = "./Java入
    门/src/day07_文件与IO/objectstream/dog.obj";
77
78      public static void main(String[] args) {
79          writeObject();
80          readObject();
81      }
82
83      /**
84       * 对象序列化
85       * 把对象写入文件: 实际写入的是类名、属性名、属性类型、属性的值等
86       */
87      private static void writeObject() {
88          Dog dog = new Dog("旺旺", 2, "母");
89          File file = new File(pathname_idea);
90          try {
91              OutputStream output = new FileOutputStream(file);
92              ObjectOutputStream oos = new
    ObjectOutputStream(output);
93              oos.writeObject(dog);
94              oos.close();
95          } catch (FileNotFoundException e) {
96              e.printStackTrace();
97          } catch (IOException e) {
98              e.printStackTrace();
99          }
100     }
101
102     /**
103      * 反序列化
```

```
     * 从文件中把对象的内容读取出来，还原成一个对象
     */
    private static void readObject() {
        File file = new File(pathname_idea);
        try {
            InputStream input = new FileInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(input);
            Dog dog = (Dog) ois.readObject();
            ois.close();
            System.out.println(dog.toString());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

- 序列化一组对象：
  - 序列化一组对象可采用：对象数组的形式，因为对象数组可以向Object进行转型操作

```
import java.io.*;
import java.util.Arrays;

/**
 * @author xiao儿
 * @date 2019/9/1 11:11
 * @Description ObjectsStreamDemo
 */
public class ObjectsStreamDemo {
    private static final String pathname_eclipse = "./src/day07_文件与IO/objectstream/dog.obj";
    private static final String pathname_idea = "./Java入门/src/day07_文件与IO/objectstream/dog.obj";

    public static void main(String[] args) {
        writeObjects();
        readObjects();
    }

    private static void writeObjects() {
        Dog dog = new Dog("旺旺", 2, "母");
        Dog dog2 = new Dog("万万", 3, "公");
        Dog dog3 = new Dog("旺财", 1, "母");
        Dog[] dogs = {dog, dog2, dog3};
        File file = new File(pathname_idea);
        try {
            OutputStream output = new FileOutputStream(file);
            ObjectOutputStream oos = new ObjectOutputStream(output);
            oos.writeObject(dogs);
            oos.close();
```

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

    private static void readObjects() {
        File file = new File(pathname_idea);
        try {
            InputStream input = new FileInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(input);
            Dog[] dogs = (Dog[]) ois.readObject();
            System.out.println(Arrays.toString(dogs));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

- transient关键字：
  - 如果用transient声明一个实例变量，当对象存储时，它的值不需要维持

```
// 在对象序列化时被忽略
private transient int id;
```

7.10. 字节数组流

7.11. 数据流

7.12. 字符串流、管道流、合并流

7.13. RandomAccessFile

7.14. Properties文件操作

7.15. 文件压缩与解压缩

7.16. 装饰者模式

7.17. 常见字符编码

7.18. New IO