1.io流-续

1.1 BufferedReader&BufferedWriter

扩展的方法： 按行读取和按行写入

```
public class Test {

    public void readFile(String path) {

        FileReader fr = null;
        BufferedReader reader = null;
        try {

            // 初始化流对象
            fr = new FileReader(path);
            reader = new BufferedReader(fr);

            String line = "";
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (fr != null) {
                    fr.close();
                }
                if (reader != null) {
                    reader.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public void writeFile(String path) {

        FileWriter fw = null;
        BufferedWriter writer = null;
        try {
            fw = new FileWriter(path, true);
            writer = new BufferedWriter(fw);

            writer.newLine(); // 换行
            writer.write("abc \n");
            // writer.newLine();
            writer.write("123");

            writer.flush();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {

                if (fw != null) {
                    fw.close();
                }
                if (writer != null) {
                    writer.close();
                }

            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }


    public static void main(String[] args) {

        Test test = new Test();
        // test.readFile("C:\\Users\\jack\\Desktop\\java10.5\\笔记\\word.txt");
        test.writeFile("C:\\Users\\jack\\Desktop\\java10.5\\笔记\\word.txt");

    }

}
```

1.2 ObjectInputStream&ObjectOutputStream

作用： 可以将java中的对象 以流的方式进行保存

1. 使用对象类需要对象 implements Serializable 接口。
2. 指定 private final static long serialVersionUID = -1;

准备一个对象

```
public class User implements Serializable {

    private String name;
    private Integer age;

    private final static long serialVersionUID = -1;


    public User() {
    }

    public User(String name, Integer age) {
        this.name = name;
        this.age = age;
    }
```

```
    public String getName() {
        return name;
    }

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

    public Integer getAge() {
        return age;
    }

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

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

基本使用

```
/**
 * 作者: jack
 * 时间: 2021-04-29 0029 08:45
 * 描述: Test
 * 1. 将对象保存到文件中
 * 2. 从文件中读取对象
 */
public class Test {

    public void saveObj2File(String path) {

        int a = 100;
        String str = "hello";
        User user = new User("admin", 20);

        FileOutputStream fo = null;
        ObjectOutputStream oo = null;
        try {

            fo = new FileOutputStream(path);
            oo = new ObjectOutputStream(fo);

            // 写入对象
            oo.writeInt(a);
            oo.writeUTF(str);
            oo.writeObject(user);
            oo.flush();

        } catch (Exception e) {
```

```
            e.printStackTrace();
        } finally {
            try {
                if (fo != null) {
                    fo.close();
                }
                if (oo != null) {
                    oo.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }

    public void readObjFromFile(String path) {

        FileInputStream fi = null;
        ObjectInputStream oi = null;
        try {

            fi = new FileInputStream(path);
            oi = new ObjectInputStream(fi);

            // PS: 读取对象流中内容的时候 需要和写入的顺序对应
            System.out.println(oi.readInt());
            System.out.println(oi.readUTF());
            System.out.println(oi.readObject());

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (fi != null) {
                    fi.close();
                }
                if (oi != null) {
                    oi.close();
                }

            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }


    public static void main(String[] args) {

        Test test = new Test();

        // test.saveObj2File("C:\\Users\\jack\\Desktop\\java10.5\\笔记
\\data.dat");

        test.readObjFromFile("C:\\Users\\jack\\Desktop\\java10.5\\笔记
\\data.dat");
```

```
    }


}
```

1.3ByteArrayInputStream&ByteArrayOutputStream

作用可以作为缓冲区来使用。

```
public class User implements Serializable {

    private String name;
    private Integer age;

    private final static long serialVersionUID = -1;


    public User() {
    }

    public User(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

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

    public Integer getAge() {
        return age;
    }

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

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

```
public class Test {
```

```
    public byte[] obj2byte() {

        int a = 100;
        String str = "hello";
        User user = new User("admin", 20);

        /**
         * Closing a <tt>ByteArrayOutputStream</tt> has no effect. The methods
in
         * this class can be called after the stream has been closed without
         * generating an <tt>IOException</tt>.
         * 字节数组流不需要程序员主动关闭
         */
        ByteArrayOutputStream bo = null;
        ObjectOutputStream oo = null;
        try {
            bo = new ByteArrayOutputStream();
            oo = new ObjectOutputStream(bo); // 通过对象流将对象写入到字节数组中

            oo.writeInt(a);
            oo.writeUTF(str);
            oo.writeObject(user);
            oo.flush();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (oo != null) {
                try {
                    oo.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return bo.toByteArray();

    }

    public void byte2obj(byte[] bytes) {

        ByteArrayInputStream bi = null;
        ObjectInputStream oi = null;
        try {

            bi = new ByteArrayInputStream(bytes);
            oi = new ObjectInputStream(bi);

            System.out.println(oi.readInt());
            System.out.println(oi.readUTF());
            System.out.println(oi.readObject());

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (oi != null) {
                try {
```

```
                    oi.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }

    public static void main(String[] args) {

        Test test = new Test();
        byte[] bytes = test.obj2byte();
        // System.out.println(Arrays.toString(bytes));
        test.byte2obj(bytes);

    }


}
```

1.4 InputStreamReader&OutputStreamWriter

作用：字节流和字符流之间的相互转换

例子： 编码处理

```
public class Test {

    public void read(String path) {

        // FileReader reader = null;
        FileInputStream inputStream = null;
        InputStreamReader reader = null;
        try {
//            reader = new FileReader(path);
//            char[] buffer = new char[1024];
//            int len = 0;
//            while ((len = reader.read(buffer)) != -1) {
//                System.out.println(new String(buffer, 0, len));
//            }
            inputStream = new FileInputStream(path);


            // 将字节流转成字符流,并指定字节流使用的编码
            reader = new InputStreamReader(inputStream, Charset.forName("GBK"));

//            byte[] buffer = new byte[1024];
//            int len;
//            while ((len = inputStream.read(buffer)) != -1) {
//                System.out.println(new String(buffer, 0, len));
//            }

            char[] buffer = new char[1024];
            int len = 0;
```

```
            while ((len = reader.read(buffer)) != -1) {
                System.out.println(new String(buffer, 0, len));
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
//            if (reader != null) {
//                try {
//                    reader.close();
//                } catch (IOException e) {
//                    e.printStackTrace();
//                }
//            }

            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }


    public static void main(String[] args) {

        new Test().read("C:\\Users\\jack\\Desktop\\java10.5\\笔记\\word.txt");

    }
}
```

1.5PrintWriter

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

        // 打印到文件
//        PrintWriter writer = new
PrintWriter("C:\\Users\\jack\\Desktop\\java10.5\\笔记\\word.txt");
//        writer.println("123");
//        writer.println("123");
//        writer.println("123");
//        writer.println("123");
//        writer.flush();


        // 打印到控制台
        PrintWriter writer = new PrintWriter(System.out);
        writer.println("hello world");
        writer.flush();
```

```
    }
}
```

1.6学习要求

1. 流的分类
2. 记住常用的流
3. 每种流的使用方法，用处。

1.7 练习

```
有文件a.txt,b.txt,c.txt ... 若干

a.txt
hello java
hello oracle

b.txt
hello oracle
hello linux

c.txt
java linux
linux shell

....
需求：
统计以上文本中，每个单词出现的次数

分析：
1. 扫描文件路径得到所有文件的绝对路径 ：List<String> allFiles;
 方法声明  scan(String path,List<String> allFiles);
2. 定义个方法读取单个文件,按行读取,将读取的内容存放到一个 List<String> lines;
 方法声明  private void readFile(String path,List<String> lines);
3. 定义一个方法 循环调用 readFile() 的到 List<String> lines;
 方法声明  List<String> readFile(List<String> allFiles);
4. 遍历 List<String> lines;  line.split(" "); 保存到 Map<String,Integer>
{word,count};
5. main 中按照顺序调用
```