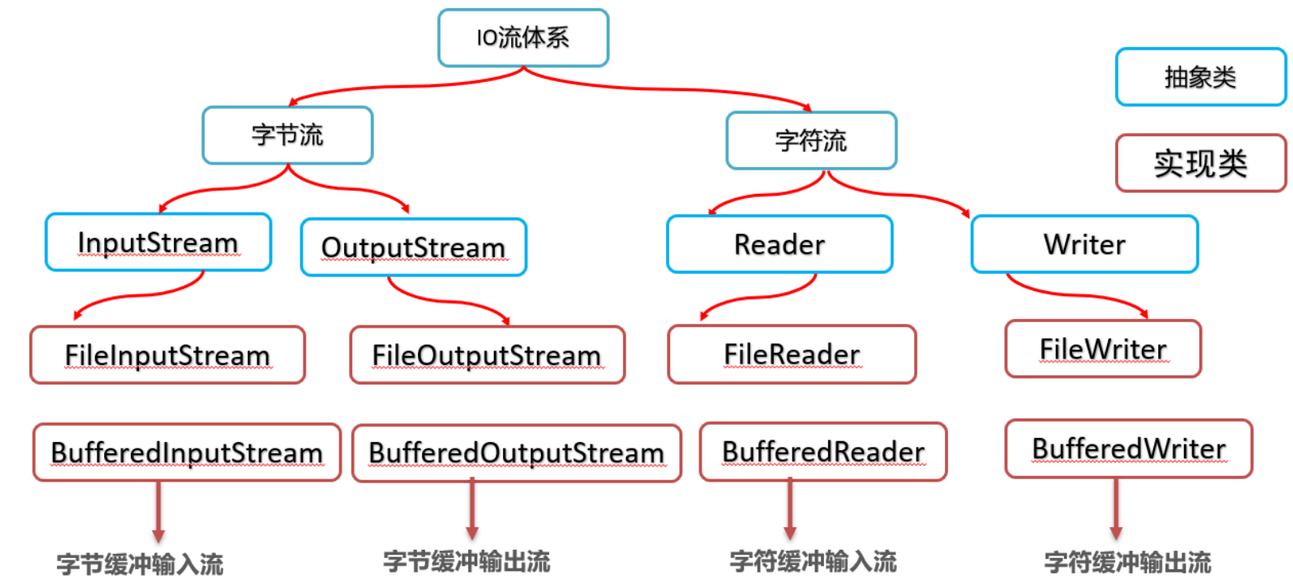
缓冲流

缓冲流也称为高效流、或者高级流。之前学习的字节流可以称为原始流。
作用：缓冲流自带缓冲区、可以提高原始字节流、字符流读写数据的性能



字节缓冲流
　字节缓冲输入流： BufferedInputStream
　字节缓冲输出流： BufferedOutputStream
字符缓冲流
　字符缓冲输入流： BufferedReader
　字符缓冲输出流： BufferedWriter

字节缓冲流

- 字节缓冲输入流：BufferedInputStream，提高字节输入流读取数据的性能，读写功能上并无变化。
- 字节缓冲输出流：BufferedOutputStream，提高字节输出流读取数据的性能，读写功能上并无变化。

构造器	说明
public BufferedInputStream(InputStream is)	可以把低级的字节输入流包装成一个高级的缓冲字节输入流管道，从而提高字节输入流读数据的性能
public BufferedOutputStream(OutputStream os)	可以把低级的字节输出流包装成一个高级的缓冲字节输出流，从而提高写数据的性能

只是提高了读写数据的性能，其读写功能没有变化

- **缓冲流拷贝案例**

```
    @Test
    public void testStream() {
        try (
                InputStream is = new FileInputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\day08、阶段项目.pptx");
                OutputStream os = new FileOutputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\2day08、阶段项目.pptx");
                InputStream bis = new BufferedInputStream(is);//自带8kb的缓冲区，可以提高原始流的性能（装饰）
                OutputStream bos = new BufferedOutputStream(os);//自带8kb的缓冲区，可以提高原始流的性能（装饰）
        ) {
            byte[] buffer = new byte[1024];
            int length;
            while ((length = bis.read(buffer)) != -1) {
                bos.write(buffer, 0, length);
            }
            System.out.println("复制完成！");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

字节缓冲流性能优化原理：
　字节缓冲输入流自带了8KB缓冲池，以后我们直接从缓冲池读取数据，所以性能较好。
　字节缓冲输出流自带了8KB缓冲池，数据就直接写入到缓冲池中去，写数据性能极高了

- **拷贝性能对比**

```
    /**
     * 一滴一滴（报废）
     */
    private static void copy4() {
        final long start = System.currentTimeMillis();
        try (
                InputStream is = new FileInputStream(old);
                OutputStream os = new FileOutputStream("D:\\Temp\\演示视频2.mp4");
```

```
        ) {
            int info;
            while ((info = is.read()) != -1) {
                os.write(info);
                os.flush();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        final long end = System.currentTimeMillis();
        System.out.println("一滴一滴耗时：" + (end - start) + "ms");
    }
```

```
    /**
     * 一桶一桶
     */
    private static void copy3() {
        final long start = System.currentTimeMillis();
        try (
                InputStream is = new FileInputStream(old);
                OutputStream os = new FileOutputStream("D:\\Temp\\演示视频3.mp4");
        ) {
            byte[] buffer = new byte[1024];
            int length;
            while ((length = is.read(buffer)) != -1) {
                os.write(buffer, 0, length);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        final long end = System.currentTimeMillis();
        System.out.println("一桶一桶耗时：" + (end - start) + "ms");
    }
```

```
    /**
     * 缓冲池一滴一滴
     */
    private static void copy2() {
        final long start = System.currentTimeMillis();

        try (
                InputStream is = new FileInputStream(old);
                OutputStream os = new FileOutputStream("D:\\Temp\\演示视频4.mp4");

                InputStream bis = new BufferedInputStream(is);
                OutputStream bos = new BufferedOutputStream(os);
        ) {
            int info;
            while ((info = bis.read()) != -1) {
                bos.write(info);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        final long end = System.currentTimeMillis();
        System.out.println("缓冲池一滴一滴耗时：" + (end - start) + "ms");
    }
```

```
    /**
     * 缓冲池一桶一桶
     */
    private static void copy1() {
        final long start = System.currentTimeMillis();

        try (
                InputStream is = new FileInputStream(old);
                OutputStream os = new FileOutputStream("D:\\Temp\\演示视频5.mp4");

                InputStream bis = new BufferedInputStream(is);
                OutputStream bos = new BufferedOutputStream(os);
        ) {
            int length;
            byte[] buffer = new byte[1024];
            while ((length = bis.read(buffer)) != -1) {
                bos.write(buffer, 0, length);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        final long end = System.currentTimeMillis();
        System.out.println("缓冲池一桶一桶耗时：" + (end - start) + "ms");
    }
```

总结：
字节缓冲输出流，结合字节数组的方式，目前来看是性能最优的组合

- 缓冲字符流实现文本拷贝排序

```
    @Test
    public void test1() {
        try (
                Reader fr = new FileReader("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\代码\\io-app2\\src\\csb.txt");

                Writer fw = new FileWriter("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\代码\\io-app2\\src\\csb2.txt");
                final BufferedReader br = new BufferedReader(fr);
                final BufferedWriter bw = new BufferedWriter(fw);
        ) {
            final List<String> lists = new ArrayList<>();
            String data;
```

```
            while ((data = br.readLine()) != null) {//只有这个字符缓冲输入流读完数据为null作为退出循环的条件
                lists.add(data);
            }
            final List<String> sortData = new ArrayList<>();
            Collections.addAll(sortData, "一", "二", "三", "四", "五", "陆", "柒", "八", "九", "十", "十一");
            Collections.sort(lists, new Comparator<String>() {
                @Override
                public int compare(String o1, String o2) {
              return sortData.indexOf(o1.substring(0, o1.indexOf("."))) -
                  sortData.indexOf(o2.substring(0, o2.indexOf(".")));
                }
            });
            for (String list : lists) {
                bw.write(list);
                bw.newLine();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    @Test
    public void testList() {
        try (
                final Reader fr = new FileReader("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\代码\\io-
app2\\src\\csb.txt");
                final Writer fw = new FileWriter("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\代码\\io-
app2\\src\\csb2.txt");
                final BufferedWriter bw = new BufferedWriter(fw);
                final BufferedReader br = new BufferedReader(fr);
        ) {
            List<String> lists = new ArrayList<>();
            List<String> sorts = new ArrayList<>();
            String line;
            while ((line = br.readLine()) != null) {
                lists.add(line);
            }
            Collections.addAll(sorts, "一", "二", "三", "四", "五", "陆", "柒", "八", "九", "十", "十一");
            Collections.sort(lists, (o1, o2) -> {
                return sorts.indexOf(o1.substring(0, o1.indexOf("."))) - sorts.indexOf(o2.substring(0,
o2.indexOf(".")));
            });
            for (String list : lists) {
                bw.write(list);
                bw.newLine();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

转换流

1问：之前我们使用字符流读取中文是否有乱码？
　没有的，因为代码编码和文件编码都是UTF-8。

2问：如果代码编码和文件编码不一致，使用字符流直接读取还能不乱码吗？
　会乱码。
　文件编码和读取的编码必须一致才不会乱码。
　字符流直接读取文本内容，必须文件和代码编码（Idea默认UTF-8）一致才不会乱码

- 粗糙解决乱码

```
    /**
     * ClassName: TransferIoDemo
     * Description: 在Idea上读取，GBK的文件；
     * 目标文件必须与字节管道接通，把以GBK编码的字节用GBK的形式解码成字符
     * 但必须确保读取到全部的字节，否则乱码
     * date:2022/3/9
     *
     * @author fgcy
     * @since JDK 1.8
     */
    @Test
    public void test1() {
        try (

                //字节流与目标文件接通
                final FileInputStream is = new FileInputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\amnb.txt");
        ) {
            //须确保读取到全部的字节
            final byte[] buffer = new byte[1024];//用字节桶装数据，一次装完，不然乱码
            int len;
            while ((len = is.read(buffer)) != -1) {
                System.out.println(new String(buffer, 0, len, "GBK"));//以GBK的解码方式获取字符
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    /**
     * ClassName: TransferIoDemo
     * Description:将Idea中的文件写到GBK文件中(操作目标文件)
     * 这里是想要将UTF-8的文件写到GBK文件中，因此目标文件必须连接字节管道，将UTF-8的字符串转为GBK类型的字节编码
     * date:2022/3/9
     *
     * @author fgcy
     * @since JDK 1.8
     */
    @Test
```

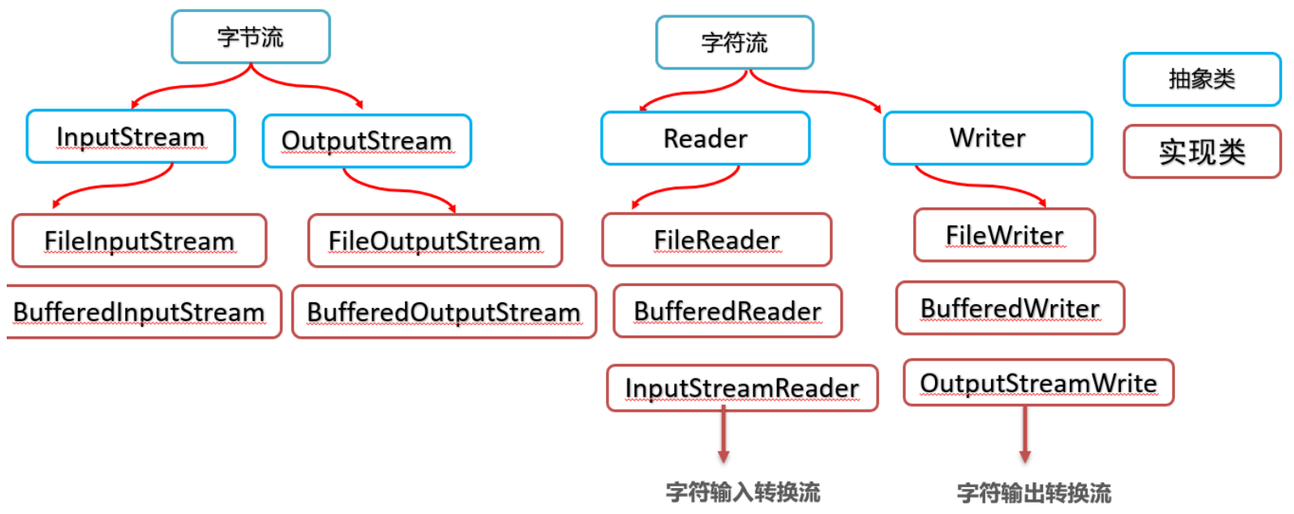
```
    public void test2() {
        try (
                //不是空项目的模式，所以是src下的
                final FileReader fr = new FileReader("src/resource/a.txt");

                //与目标文件接通的是字节流
                final FileOutputStream os = new FileOutputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流
二\\PPT\\amnb.txt", true);
        ) {
            //必须确保将字符全部装完
            final char[] buffer = new char[1024];
            int len;
            while ((len = fr.read(buffer)) != -1) {
                //将字符转为GBK编码的字节
                os.write(new String(buffer, 0, len).getBytes("GBK"));
            }

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

- 完美解决乱码问题：

使用字符输入转换流 InputStreamReader
以读取GBK的文件为例：
1.提取以GBK编码的文件的原始字节流
2.然后把字节流以指定编码（GBK）转换成字符输入流，这样字符输入流中的字符就不乱码了



字符输入转换流

- 字符输入转换流：InputStreamReader，可以把原始的字节流按照指定编码转换成字符输入流。

构造器	说明
public InputStreamReader(InputStream is)	可以把原始的字节流按照代码默认编码转换成字符输入流。几乎不用，与默认的FileReader一样。 字符流的源码就是使用字符输入转换流将字节流以默认编码转为字符
public InputStreamReader(InputStream is，String charset)	可以把原始的字节流按照指定编码转换成字符输入流，这样字符流中的字符就不乱码了(重点)

```
    @Test
    public void testRead() {
        try (
                final InputStream is = new FileInputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\ayu.txt");
                final Reader isr = new InputStreamReader(is, "GBK");//代码以GBK的方式编码，文件以Gbk编码，不会乱码
//                final Reader isr = new InputStreamReader(is);//代码以默认的方式编码（UTF-8），文件以Gbk编码，会
乱码
                final BufferedReader br = new BufferedReader(isr);
        ) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

字符输入转换流InputStreamReader作用：
可以解决字符流读取不同编码乱码的问题
public InputStreamReader(InputStream is,String charset):
可以指定编码把原始字节流转换成字符流，如此字符流中的字符不乱码。

字符输出转换流

- 字符输入转换流：OutputStreamWriter，可以把字节输出流按照指定编码转换成字符输出流。

构造器	说明
public OutputStreamWriter(OutputStream os)	可以把原始的字节输出流按照代码默认编码转换成字符输出流。几乎不用。
public OutputStreamWriter(OutputStream os，String charset)	可以把原始的字节输出流按照指定编码转换成字符输出流(重点)

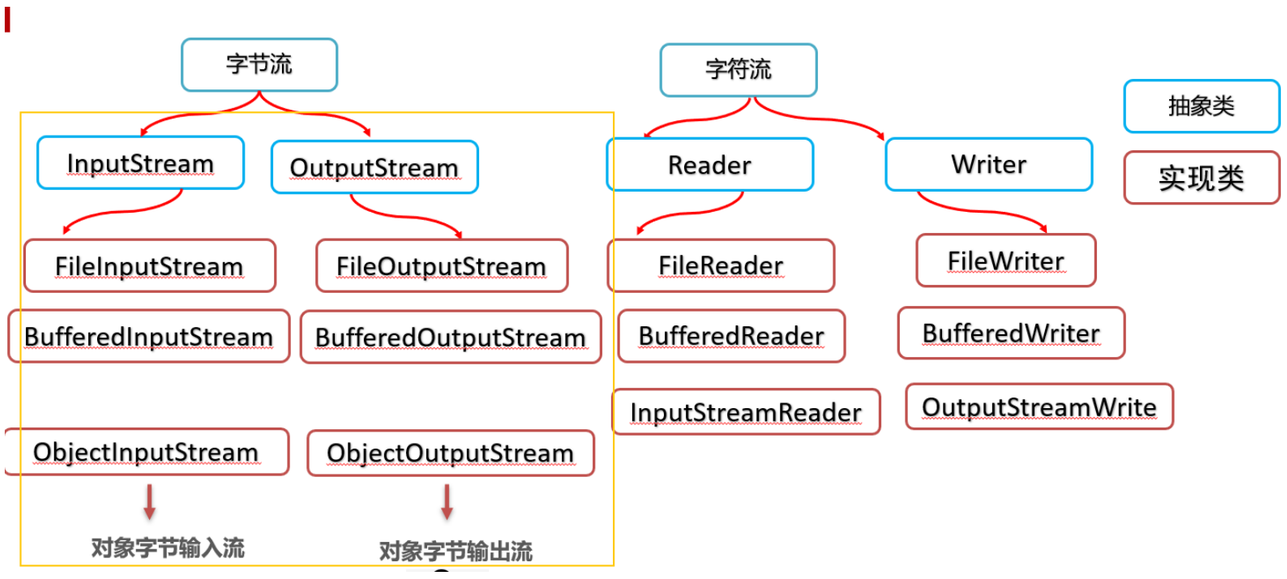
```
    @Test
    public void testRead2() {
        try (
                final OutputStream os = new FileOutputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\outputstreamwriter.txt");
                final Writer osw = new OutputStreamWriter(os, "GBK");//将字符转换成以GBK编码的字节
                final BufferedWriter bw = new BufferedWriter(osw);
        ) {
            bw.write("今明后核辐射大半年么");
            bw.newLine();
            bw.write("佛挡杀佛的速度辅导费发来了解决");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

字符输出转换流OutputStreamWriter的作用?
　public OutputStreamWriter(OutputStream os，String charset)
　可以指定编码把将字符转为与目标文件一致编码

序列化

- 定义

以内存为基准，把内存中的对象存储到磁盘文件中去，称为对象序列化
使用到的流是对象字节输出流：ObjectOutputStream



对象序列化：

- 使用到的流是对象字节输出流：ObjectOutputStream

构造器	说明
public ObjectOutputStream(OutputStream out)	把低级字节输出流包装成高级的对象字节输出流

ObjectOutputStream序列化方法

方法名称	说明
public final void writeObject(Object obj)	把对象写出去到对象序列化流的文件中去

```
    public static void main(String[] args) throws Exception {
        final Student student = new Student("张伟健", "qwe3211126", '男', 18);
        final OutputStream os = new FileOutputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\serializable.txt");
        final ObjectOutputStream oos = new ObjectOutputStream(os);
        oos.writeObject(student);
        oos.close();
    }
```

```
 ��  •sr •serializable.Student (���d�U•  •I •ageC •sexL •namet •Ljava/lang/String;L •passwordq ~ •xp
•u7t   张伟健t
qwe3211126
```

对象序列化的含义是什么?
　把对象数据存入到文件中去。
对象序列化用到了哪个流?
　对象字节输出流ObjectOutputStram
　public void writeObject(Object obj)
序列化对象的要求是怎么样的?
　对象必须实现序列化接口

对象反序列化：

- 使用到的流是对象字节输入流：ObjectInputStream
- 作用：以内存为基准，把存储到磁盘文件中去的对象数据恢复成内存中的对象，称为对象反序列化。

构造器	说明
public ObjectInputStream(InputStream out)	把低级字节输如流包装成高级的对象字节输入流

ObjectInputStream序列化方法

方法名称	说明
public Object readObject()	把存储到磁盘文件中去的对象数据恢复成内存中的对象返回

```
    @Test
    public void testSerial() throws Exception {
        final InputStream is = new FileInputStream("D:\\learn\\Idea\\basis\\src\\serializable.txt");
        final ObjectInputStream ois = new ObjectInputStream(is);
        final Student student = (Student) ois.readObject();
        System.out.println(student);
    }
```

对象反序列化的含义是什么?
　把磁盘中的对象数据恢复到内存的Java对象中。
对象反序列化用到了哪个流?
　对象字节输入流ObjectInputStram
某个字段不想序列化

```
private transient String password;//transient修饰的字段不会进行反序列化
```

```
Student{name='张伟健', password='null', sex=男, age=18}
```

```
private static final long serialVersionUID = 620462426732525268L;
```

```
当javaBean中的数据改变时或手动改变serialVersionUID，需要从新序列化更新文件中的对象，否则从文件中反序列化会失效
```

打印流

打印流可以实现方便(|)、高效(有缓冲流)的打印数据到文件中去。打印流一般是指：PrintStream（字节流），PrintWriter（字符流）两个类。
可以实现打印什么数据就是什么数据，例如打印整数97写出去就是97，打印boolean的true，写出去就是true。

PrintStream

构造器	说明
public PrintStream(OutputStream os)	打印流直接通向字节输出流管道
public PrintStream(File f)	打印流直接通向文件对象
public PrintStream(String filepath)	打印流直接通向文件路径

方法	说明
public void print(Xxx xx)	打印任意类型的数据出去

```
    @Test
    public void test() throws Exception {
//        final OutputStream os = new FileOutputStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\printStream.txt");
//        final PrintStream ps = new PrintStream(os, true, "GBK");///包装了bufferedStream所以也属于高级流,而且包装了字符输出转换流，写完一个字节数组后自动刷新，将字符通过字符输出转换流转为字节，通过底层字节流写出
        final PrintStream ps = new PrintStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\printStream.txt", "UTF-8");
        //print（）做了重载，将类型转换成同等字符串然后输出，所以才有打印什么就是什么
        ps.println("闪避");
        ps.println(111);
        ps.println(true);
        ps.println('a');
        ps.write(48);//输出字节48，以utf8的字符编码，兼容ascii所以字符是0
        ps.close();
    }
```

PrintStream和PrintWriter的区别
打印数据功能上是一模一样的，都是使用方便，性能高效（核心优势）
PrintStream继承自字节输出流OutputStream，支持写字节数据的方法。
PrintWriter继承自字符输出流Writer，支持写字符数据出去。

PrintWriter

构造器	说明
`public PrintWriter(OutputStream os)`	打印流直接通向字节输出流管道
`public PrintWriter (Writer w)`	打印流直接通向字符输出流管道
`public PrintWriter (File  f)`	打印流直接通向文件对象
`public PrintWriter (String filepath)`	打印流直接通向文件路径

方法	说明
`public void print(Xxx xx)`	打印任意类型的数据出去

- 打印流实现重定向

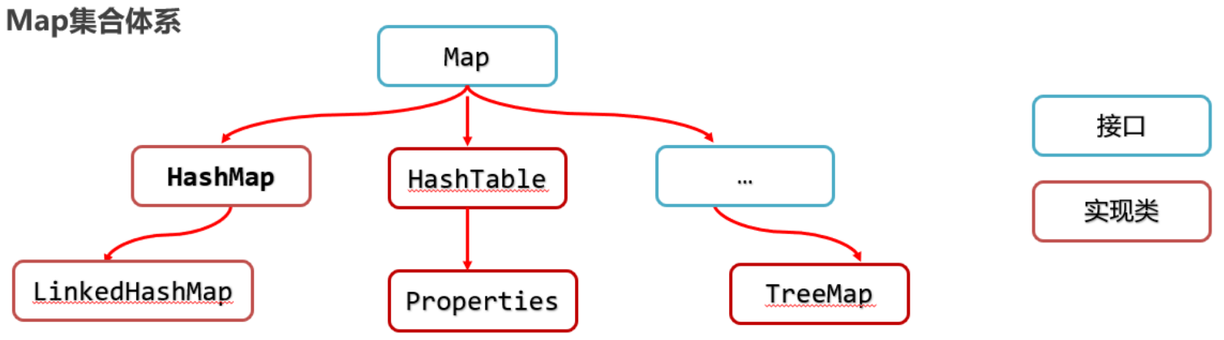
```
    @Test
    public void test2() throws Exception {
        final PrintStream ps = new PrintStream("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\relocation.txt", "UTF-8");
        System.setOut(ps);
        ps.println("儿时望星空");
        ps.println("举手若能摘");
        ps.println("而今七尺身");
        ps.println("天高不可及");
        ps.close();
    }
```

将输出的内容流向文件而不是控制台

- Properties

其实就是一个Map集合，但是我们一般不会当集合使用，因为HashMap更好用
Properties代表的是一个属性文件，可以把自己对象中的键值对信息存入到一个属性文件中去。
属性文件：后缀是.properties结尾的文件,里面的内容都是 key=value，后续做系统配置信息的

Map集合体系



Properties的API:

- Properties和IO流结合的方法：

构造器	说明
`void load(InputStream inStream)`	从输入字节流读取属性列表（键和元素对）
`void load(Reader reader)`	从输入字符流读取属性列表（键和元素对）
`void store(OutputStream out, String comments)`	将此属性列表（键和元素对）写入此 Properties表中，以适合于使用 load(InputStream)方法的格式写入输出字节流
`void store(Writer writer, String comments)`	将此属性列表（键和元素对）写入此 Properties表中，以适合使用 load(Reader)方法的格式写入输出字符流
`public Object setProperty(String key, String value)`	保存键值对（put）
`public String getProperty(String key)`	使用此属性列表中指定的键搜索属性值 (get)
`public Set<String> stringPropertyNames()`	所有键的名称的集合 (keySet())

```
    //存
    @Test
    public void test3() throws Exception {
        final Properties properties = new Properties();
        properties.setProperty("admin", "123456");
        properties.put("dlei", "56789");//使用父类的方法
        properties.setProperty("ben", "545589");
        properties.store(new FileWriter("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\user.properties"), "this is user information!");

    }
```

```
    //取
    @Test
    public void test4() throws Exception {
        final Properties properties = new Properties();
        properties.load(new FileReader("D:\\learn\\黑马程序员\\java基础\\day20、IO流二
\\PPT\\user.properties"));
        System.out.println(properties.getProperty("admin"));
        System.out.println(properties.get("dlei"));
        final Enumeration<?> enumeration = properties.propertyNames();//相当于获取keyset
        while (enumeration.hasMoreElements()) {
            System.out.println(enumeration.nextElement());
        }
    }
```

IO框架

commons-io概述

commons-io是apache开源基金组织提供的一组有关IO操作的类库，可以提高IO功能开发的效率。

commons-io工具包提供了很多有关io操作的类。有两个主要的类FileUtils, IOUtils

```
    @Test
    public void test1() throws IOException {
        //文件复制
    /*    FileUtils.copyFile(new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\day08、阶段项
目.pptx"),
                new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\day08、阶段项目2.pptx"));*/

        //文件复制到文件夹

   /*     FileUtils.copyFileToDirectory(new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\day08、
阶段项目.pptx")
                , new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\代码"),true);*/


        //复制文件夹到文件夹
     /*   FileUtils.copyDirectoryToDirectory(new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\新建文件
夹"),
                new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT"));*/


        //删除文件
//        FileUtils.delete(new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件夹\\D.txt"));

        //删除文件夹
//        FileUtils.deleteDirectory(new File("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件
夹"));
    }
```

- NIO

```
    @Test
    public void test2() throws IOException {
        //文件复制
       /* Files.copy(Paths.get("D:\\\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件夹\\D.txt"),
                Paths.get("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件夹\\D2.txt"));*/


        //删除文件
//        Files.delete(Paths.get("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件夹\\D2.txt"));


        //删除空文件夹
//        Files.delete(Paths.get("D:\\learn\\黑马程序员\\java基础\\day20、IO流二\\PPT\\新建文件夹"));


    }
```