Map接口：

Map集合是无序的，每条数据都包含两个信息：Key-Value；

Key：唯一不重复（equals比较为true视为重复）

Value：是可以重复的值

Map接口和Collection是没有交际的，Map接口是独立的接口

Put(K,V)：添加元素

1. 当添加的元素k-v在集合不存时，为添加元素
2. 当添加的元素k-v在集合中存在时，则会根据k值替换与之相等的v
3. 返回的值，为map集合中与key相等的，value值，没有为null

Get(K): equals比较为true视为相同的key该方法根据给定的K获取对应的v值，返回值为对应泛型的v值，如果不存在返回null

Remove(K)：根据给定的K，来删除这一行数据，返回值为被删除的K所对应的v值，不存在返回null

重写equals方法时应当重写hashCode(),重写规则如下：

1. 若两个对象equals方法比较为true，那么hashCode()返回的值必须相同
2. 反之不是必须的，但是这种情况要避免：即两个对象equals比较为false，但是hashCode()返回值一样，这种情况要避免，如果hashCode()返回值一样，则会影响HashMap的性能
3. 注意：hashCode()在当前内容没有发生改变的前提下，多次调用应当是返回相同的值，如果内容修改后，则应当返回不同的值

加载因子，0.75: 平衡检索和空间利用率，没有初始化散列组大小时默认值为16，可以根据数据大小发生变化，保证加载因子为0.75

Map集合遍历：

1. 遍历所有的Key
   1. Set<K> keyset()：可以获取Map集合中所有的Key值并将其存入一个Set集合中返回，遍历该Set集合即可
2. 遍历所有的Key及Value
   1. Set<Entry<K,V>> entrySet(): 讲每一组K-v存入一个Entry<k,v>实例中，并返回Set集合，然后遍历该Set集合即可
3. 遍历所有的Value
   1. Collect<T> Values(): 返回的值时对应的value泛型的值，存入Collection<T>集合，然后迭代器遍历即可

File：文件和目录路径的抽象表现形式，只能获取文件的周边信息

1. 不能对文件内容进行读写操作
2. 想要操作一个文件，需要现有一个文件的对象
3. 跨平台：一次编译到处运行，处处改错，文件分割符号，各个平台表现形式不一样
   1. Windows：\
   2. Linux：/
   3. 为了解决这种跨平台引起的非逻辑性的一场，引入以常量表示文件的分隔符：
   4. Separator：这个常量可以在各个平台上，根据各个平台的属性，自动使用文件分隔符的表现形式
   5. 文件分隔符：separaotr（常量），--忽略平台的差异性
   6. 属于文件类中的常量
   7. 语法：File.separator调用
4. 路径：
   1. 绝对路径：全路径
   2. 相对路径：有一个参照物，参照物是当前的项目，相对项目名
   3. ./ 代表是当前目录
   4. ../ 代表的是当前目录的上一级目录（父级目录）
5. 获取文件的周边信息的方法
   1. getName()获取文件或者目录的名字，返回String字符串
   2. 删除目录，只能删除空目录

RandomAccessFile:

File类不能操作文件的内容，如果想操作文件的内容，需要用一个RandomAccessFile类

此类的实例支持随机访问文件，对文件进行读取和写入操作

此类基于指针的操作，总是在当前位置读写子接

两种模式：“rw”: 可读可写，”r“：只读

两个构造方法：

RandomAccessFile(String path, String mode)

RandomAccessFile(File file, String mode)

该类方法的读和写都是参照虚拟内存而言的

虚拟内存（程序）🡨读入—物理内存（文件）

虚拟内存（程序）--写入🡪物理内存（文件）

Write():写入一个字节，写入低八位-void—写的是当前指针指向的字节，写完后指针后移到下一位

Read(): 读取一个字节，读取低八位-int—读的是当前指针指向的字节—返回值为读回得字节数，存入得是int类型得低八位上：

--一次读一个字节，读完后指针自动后移到下一个字节处，如果返回值为-1表示没有数据可读

底层是封装了两个流—输入和输出

RandomAccessFile向一个文件中写入一个字节

1. 如果写入得字节数大于原文件中的字节数，则进行全部覆盖。
2. 如果写入的字节数小于原文件中的字节数，则进行写入字节数的等量覆盖

文件的复制粘贴

1. 一个一个字节的复制
2. 使用一个字节的数组

写入字符串，

字符变成字节—写入

将当前的字符串按照系统默认的字符集编码转换成对应的字节，返回值为对应的字节数组

重载的方法：

String的getBytes(String charset)：将当前的字符串按照指定的字符集编码转换成对应的字节返回值为对应的字节数组

字符集编码：256\*256 = 65536 256

1. gbk：英文一个字节表示一个字符，中文两个字节表示一个字符
2. utf-8：万国码，英文一个字节表示一个字符，中文是三个字节表示一个字符（少数的两字节，四个字节）
3. gb2312：繁体字
4. Unicode：一个字符一个字节—只服务于英文

RandomAccessFile：这个类是基于指针的操作

getFilePointer():获取指针的当前位置（返回值为long类型）在该位置发生读取或写入操作。

Seek(Long l)：调整指针的当前位置，设置到此文件开头的测量位置的偏移量，在该位置发生读取或者写入操作，无返回值，可以任意位置偏移（前后都可以偏移），偏移量的是从0开始

表示指针调整字节为2的位置

SkipByte(int n) 尝试跳过输入的n个字节，以丢弃跳过的字节、只能往后跳，不能往前跳，使用频率较低

输入流和输出流 – IS与OS

IO流：

1. InputStream：抽象类，所有字节输入流的父类
2. OutputStream：抽象类，所有字节输出流的父类
3. 输入和输出是以内存（虚拟内存）为主导，读取外界数据使用read()，写出程序中的数据到外界使用输出write()
4. 节点流（低级流）：可以从一个特定的地方读取/写入数据
5. 处理流（高级流）：对一个已经存在的流进行封装，它不会单独存在，是建立在低级流的基础之上，高级流是可以嵌套使用

第一组实现流： 低级流

1. FileInputStream：文件输入流
   1. Read() read(byte[] buf)
2. FileOutPutStream：文件输出流
   1. Write() write(byte[] buf)
      1. Wirte（byte[] buf, 0 ,length）
3. New FileOutputStream(String name, Boolean append)
   1. True – 追加
   2. False – 不追加
   3. 若是写false，等同于new FileOutputStream内容不追加，当前文本的内容都会被清除，那后在重新写入，这一点与RandomAccessFile不同

第二组实现流：缓冲字节流 高级流

1. BufferedInputStream：缓冲字节输入流—read()
2. BufferedOutputStream：缓冲字节输出流—write()
3. 这两个流内部各自维护着一个缓冲区（自带的缓冲区）
4. 自带缓冲区的输出流，只有在缓冲区存满的时候才会将缓冲区的数据自动写出，如果没有存满，则不写出，
   1. 两种方式将未满的缓冲区数据写出：
      1. 关流
      2. 调用flush()方法，属于强制性操作（会降低读写效率，为了保证信息的及时性和有效性，需要调用该方法）
   2. 关流的时候只需要关闭高级流即可，低级流会随着高级流的关闭而关闭，
   3. 自带缓冲区和外加的缓冲区（字节数组）的区别：
   4. 在相同字节长度情况下，FileInputStream/FileOutputStream加换成字节数组，要比BufferedInputStream/BufferedOutputStream自带的缓冲区数据的读取和写入要快一些，因为少了一个“换车”的环节

第三组实现流： 对象流 高级流

1. 可以直接保存一个对象
   1. 将对象转成字节
   2. 长久保存在硬盘中
   3. 将一个特定的数据结构转成一组字节的过程—序列化
   4. 将数据写入到硬盘中长久保存的过程—持久化
   5. 存：先将对象转成字节—保存 序列化
   6. 取：将字节转成特定的数据结构—取出 反序列化
2. 序列化和反序列一般用于
   1. 传输
   2. 保存
3. ObjectInputStream:对象输入流  
   readObject():读取一个对象
4. ObjectOutputStream:对象输出流

witeObject()：写入一个对象

1. 如果一个类的对象想要实现序列化，必须先实现可序列化的接口，这时该类的对象，才有资格进行序列化
2. 序列化的接口：serializable
3. readObject():方法读取的是一个对象，返回值为Object类型，需要进行强制类型转换，才能进行对应类型中的属性和成员方法的调用
4. 该类在序列化之后，反序列化之前，对该类的属性/方法进行修改会出现报错，为了避免这种报错出现，需要在序列化之前增加一个serialVersionUID（版本号）可以避免这种出错
5. transient：修饰的成员变量在序列化时，可以忽略其属性值，在反序列化后，不保留该属性的值，但是会该属性的存在，且有对应的默认值