Java加强

zcx

一、File

- 是java.io包下的类，File类的对象，用于当前操作系统的文件(可以是文件、或文件夹)
- File类只能对文件本身进行操作，不能读写文件里面存储的数据
- **创建File对象**

方法	作用
File(String pathname)	将给定的路径名字符串转换为抽象路径名来创建新的File实例
File(String parent, String child)	将给定父路径名和子路径名字符串转换为新的File实例
File(File parent, String child)	将给定父抽象路径名和子路径名字符串转换为新的File实例

```
File f1 = new File("E:\java-workplace\file_practice");
File f2 = new File("E:\java-workplace\file_practice","demo1_practice.txt");
File f3 = new File(f1,"demo1_practice.txt");
```

f1是文件夹，文件夹名为file_practice，f2和f3都是在文件夹file_practice下创建的文件demo1_practice.txt

- **File类提供的方法**

方法	作用
public String getAbsolutePath()	返回此抽象路径名的绝对路径名形式
public String getPath()	返回此抽象路径名的路径名部分
public String getName()	返回此抽象路径名表示的文件名
public long length()	返回此抽象路径名表示的文件的长度
public boolean exists()	测试此抽象路径名表示的文件或目录是否存在
public boolean isDirectory()	测试此抽象路径名表示的文件是否为目录
public boolean isFile()	测试此抽象路径名表示的文件是否为标准文件
public long lastModified()	返回此抽象路径名表示的文件最后一次被修改的时间

- ```
  System.out.println(f1.getName());
  System.out.println(f2.length());
  System.out.println(f3.length());
  System.out.println(f1.isDirectory());
  System.out.println(f2.isFile());
  ```
- 可以先代表不存在的文件，到时候再创建
方法	作用
public boolean createNewFile()	创建此抽象路径名指定的文件
public boolean mkdir()	创建此抽象路径名指定的目录
public boolean mkdirs()	创建此抽象路径名指定的目录，包括任何必需但不存在的父目录
public boolean delete()	删除此抽象路径名表示的文件或目录
- delete方法只能默认删除文件和空白文件夹，删除后的文件不会进入回收站
- ```
  File f4 = new File("E:\java-workplace\file_practice\demo2_practice.txt");
  System.out.println(f4.createNewFile());
  File f5 = new File("E:\java-workplace\file_practice\demo3_practice");
  System.out.println(f5.mkdir());
  File f6 = new File("E:\java-workplace\file_practice\demo4_practice\demo5_practice");
  System.out.println(f6.mkdirs());
  System.out.println(f6.delete());
  ```
  f4创建文件成功，f5创建文件夹成功，f6创建多级文件夹成功，f6删除成功
- **File提供的遍历文件夹的方法**
方法	作用
public String[] list()	返回此抽象路径名表示的目录中的文件和目录的名称
public File[] listFiles()	返回此抽象路径名表示的目录中的文件和目录的File对象
- ```
  File[] files = f1.listFiles();
  for (File file : files) {
  System.out.println(file);
  } 返回路径，且只能返回一级文件或目录
  String[] list = f1.list();
  for (String s : list) {
  System.out.println(s);
  ```
  ```

```
} 返回文件名，且只能返回一级文件或目录
输出：
E:\java-workplace\file_practice\demo1_practice.txt
E:\java-workplace\file_practice\demo2_practice.txt
E:\java-workplace\file_practice\demo3_practice
E:\java-workplace\file_practice\demo4_practice
demo1_practice.txt
demo2_practice.txt
demo3_practice
demo4_practice
```

- **使用listFiles方法遍历注意事项**
  - 当主调时文件，或者路径不存在时，返回null
  - 当主调时空文件夹时，返回一个长度为0的数组
  - 当主调时一个有内容的文件夹，将里面所有的一级文件好文件夹的路径放在File数组中返回
  - 当主调时一个文件夹，且里面有隐藏文件时，返回的数组中包含隐藏文件
  - 当主调是一个文件夹，但是没有权限访问该文件夹时，返回null

二、方法递归

- 是一种算法，方法调用自身的形式称为方法递归
- 直接递归：自己调用自己
- ```
  public static void main(String[] args) {
  //认识递归
  print(5);
  }
  public static void print(int n)
  {
  if(n > 0)
  {
  System.out.println(n);
  print(n - 1); 递归调用自身
  }
  }
  ```
- 间接递归：自己调用其他方法，其他方法又调用自己
- ```
  public static void main(String[] args) {
  print2(5); 调用print2方法
  }
  public static void print(int n)
  {
  print2( n); 调用回print2方法
  ```

```
}
//间接递归
public static void print2(int n)
{
if(n > 0)
{
System.out.println(n);
print(n - 1); 调用print方法
}
}
```

- **递归算法和执行流程**
  - 三要素;
    - 递归的公式:例如：f(n)=f(n-1)*n
    - 递归的终结点：例如：f(1)
    - 递归的方向必须走向终结点
  - C++如何，这个如何
- **文件搜索**
  - 从某个盘中，搜索某个文件名，找到后直接输出其位置。这种需求就可以用递归去做

三、字符集

- **标准ASCII字符集**：标准ASCII使用一个字节存储一个字符，首位是0，故可存128个字符
- **GBK字符集**：包含了2万多个汉字等字符，一个中文字符编码成2个字节的形式存储，也兼容了ASCII字符集。GBK规定了汉字的第一个字节的第一位必须是1，故把汉字和ASCII字符区分开了
- **Unicode字符集**：统一码，万国码。这是国际组织制定的，可以容纳世界上的所有文字、符号的字符集
- **UTF-8字符集**：是Unicode字符集的一种编码方案，采取可变长编码方案，共分为4个长度区：1个字节、2个字节、3个字节、4个字节
  - 英文字符、数字只占1个字节(兼容标准ASCII码)，汉字字符占用3个字节
  - 用前缀码区分长度区(加粗的)
字符	编码
0-127(1字节区)	**0**xxxxxxx
128-2047(2字节区)	**110**xxxxx **10**xxxxxx
2048-4095(3字节区)	**1110**xxxx **10**xxxxxx **10**xxxxxx
4096-65535(4字节区)	**11110**xxx **10**xxxxxx **10**xxxxxx **10**xxxxxx
- **注意**：
  - 字符编码时使用的字符集，和解码时使用的字符集必须一致，否则乱码

- 英文、数字一般不会乱码，因为很多字符集都兼容了ASCII编码

- **字符集的编码、解码操作**
  - 对字符的编码
方法	作用
byte[] getBytes()	使用默认字符集将字符串编码为字节数组
byte[] getBytes(String charsetName)	使用指定的字符集将字符串编码为字节数组

  - 对字符的解码
方法	作用
String(byte[])	使用默认字符集将字节数组解码为字符串
String(byte[], String charsetName)	使用指定的字符集将字节数组解码为字符串

  - ```
    String name = "啦啦啦啦啦";
    byte[] by=name.getBytes(); 使用默认字符集UTF-8将字符串编码为字节数组
    System.out.println(by.length); 5个汉字，一个汉字3字节，输出15
    byte[] by2=name.getBytes("GBK"); 使用GBK字符集将字符串编码为字节数组
    System.out.println(by2.length); 5个汉字，一个汉字2字节，输出10
    String name2 = new String(by); 使用默认字符集UTF-8将字节数组解码为字符串
    System.out.println(name2); 输出啦啦啦啦啦
    String name3 = new String(by2,"GBK"); 使用GBK字符集将字节数组解码为字符串
    System.out.println(name3);
    ```

四、IO流

- **IO流**：输入输出流，用于处理数据源，把数据从源读入到程序，把程序中的数据输出到目的地
- **分类**
  - 按照流方向分为输入流和输出流
  - 按照流内容分为**字节流**(适合操作所有类型的文件，如音频、视频、图片、文本文件)和**字符流**(只适合操作纯文本文件)
- **IO流的体系：字节输入流(InputStream)、字节输出流(OutputStream)、字符输入流(Reader)、字符输出流(Writer)。都是抽象类**
- **IO流的实现类：文件字节输入流FileInputStream、文件字节输出流FileOutputStream、文件字符输入流FileReader、文件字符输出流FileWriter**
- **字节流**
  - **文件字节输入流**：FileInputStream：以内存为基准，把磁盘文件中的数据以字节的形式读入到内存中
    ```

方法	作用
int read()	读取一个字节，返回字节值，如果已读到文件末尾，则返回-1
int read(byte[] b)	批量读取字节，返回实际读取的字节数，如果已读到文件末尾，则返回-1
FileInputStream(String path)	创建一个字节输入流对象，用于从指定路径的文件中读取数据
FileInputStream(File file)	创建一个字节输入流对象，用于从指定文件对象中读取数据

- ```
  public static void main(String[] args) throws IOException {
  InputStream is = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt"); 创建字节输入流对象
  int b; 为了看文件是否读取完成
  while ((b = is.read()) != -1) { 循环读取字节，这是一个一个读取
  System.out.print((char)b);
  }打印字符
  byte[] bs = new byte[1024]; 创建字节数组，为了批量读取字节
  int len; 为了看文件是否读取完成
  while ((len = is.read(bs)) != -1) { 循环读取字节到bs字节数组中，每次读取1024字节，因为bs的长度为1024
  System.out.print(new String(bs,0,len));
  把字节数组转换成字符串，并打印，把bs字节数组中索引从0->len的字符转成字符串
  }
  }字节流批量读取性能得到提升，但是会出现，在某个数组长度，和读取的文件的中文长度不匹配的情况，导致乱码
  可以通过设置数组长度与文件的全部字节长度一致，一次性全部读完，来避免乱码，只适合小文件，或者设置为1024
  ```

- **文件字节输出流**： FileOutputStream：把内存数据以字节形式写入磁盘文件

方法	作用
FileOutputStream(String path)	创建一个字节输出流对象，用于向指定路径的文件中写入数据
FileOutputStream(File file)	创建一个字节输出流对象，用于向指定文件对象中写入数据
  ```

方法	作用
FileOutputStream(String path, boolean append)	创建一个字节输出流对象， 用于向指定路径的文件中写入数据， 如果append为true，则追加写入
FileOutputStream(File file, boolean append)	创建一个字节输出流对象， 用于向指定文件对象中写入数据， 如果append为true，则追加写入
void write(int b)	把指定的字节写入文件
void write(byte[] b)	把字节数组写入文件
void write(byte[] b, int off, int len)	把字节数组的指定部分写入文件
void close()	关闭流，释放资源

- ```
  OutputStream f1 = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice.txt"); 创建字节输出流对象，每次都是覆盖
  f1.write(97); 写入一个字节，这是一个字符a
  f1.write("\r\n".getBytes()); 这是换行符
  f1.write(98);
  f1.write('v'); 写入一个字节，这是一个字符v
  byte[] bt = {97,98,99,100,101}; 创建字节数组
  f1.write(bt); 写入字节数组
  f1.write(bt,1,3); 写入字节数组的指定部分
  byte[] bt2 = "hello world".getBytes(); 创建字节数组，把一个字符串转换成字节数组写入
  f1.write(bt2);
  f1.close(); 关闭流(管道)，释放资源
  //OutputStream f1 = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice.txt",true); 创建字节输出流对象，每次都是追加，因为加了一个参数true
  ```

- **文件复制**

- ```
  public static void main(String[] args) throws IOException {
  //字节流完成文件的复制，源文件是：E:\java-workplace\file_practice\demo1_practice.txt
  //复制到的目标位置为：E:\java-workplace\file_practice\demo1_practice_copy.txt
  FileInputStream f1 = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt");
  FileOutputStream f2 = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice_copy.txt");
  int b;
  ```

```
while((b = f1.read()) != -1)
{
f2.write(b);
}
f1.close();
f2.close();
}
```

- **字节流非常适合做文件的复制操作，任何文件的底层都是字节，只要复制后的文件格式一致就没问题**
- **资源释放问题：**
- **try-catch-finally**，无论try中的程序是否正常执行，最后都一定会执行finally区，除非JVM终止。作用：一般用于在程序执行完成后进行资源的释放操作(专业级做法)

```
public static void main(String[] args){
FileInputStream f1 = null;
FileOutputStream f2 = null;
try {
f1 = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt");
f2 = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice_copy.txt");
int b;
while((b = f1.read()) != -1)
{
f2.write(b);
}
} catch (IOException e) {
throw new RuntimeException(e);
} finally { try-catch-finally,为了释放资源和不报错，导致finally套娃了try-catch，代码很臃肿
try {
if(f1!=null)f1.close();
if(f2!=null)f2.close();
} catch (IOException e) {
throw new RuntimeException(e);
}
}
```

- **try-with-resources：**

```
public static void main(String[] args){
==try( FileInputStream f1 = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt");
FileOutputStream f2 = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice_copy.txt");
```

```
) try-with-resources，try中小括号内声明的资源，会在try执行完成后自动释放
{
int b;
while((b = f1.read()) != -1)
{
f2.write(b);
}
} catch (IOException e) {
throw new RuntimeException(e);
}==**
}
```

- **字符流**
  - **文件字符输入流**：FileReader：把文件内容以字符形式读入到程序中
方法	作用
FileReader(String path)	创建一个字符输入流对象，用于从指定路径的文件中读取数据
FileReader(File file)	创建一个字符输入流对象，用于从指定文件对象中读取数据
int read()	读取一个字符
int read(char[] cbuf)	读取字符数组
  - ```
    public static void main(String[] args) throws IOException {
    //文件字符输入流
    FileReader fr = new FileReader("E:\java-workplace\file_practice\demo1_practice.txt");
    int b;
    while((b = fr.read()) != -1)
    {
    System.out.print((char)b);
    }
    fr.close();
    FileReader fr2 = new FileReader("E:\java-workplace\file_practice\demo1_practice.txt");
    char[] c = new char[1024];
    int len;
    while((len = fr2.read(c)) != -1)
    {
    System.out.print(new String(c,0,len));
    }
    fr2.close();
    }
    ```
    ```

- **文件字符输出流**：FileWriter：把数据写入文件中

方法	作用
FileWriter(String path)	创建一个字符输出流对象， 用于向指定路径的文件中写入数据
FileWriter(File file)	创建一个字符输出流对象， 用于向指定文件对象中写入数据
FileWriter(String path, boolean append)	创建一个字符输出流对象， 用于向指定路径的文件中写入数据， 可以指定追加还是覆盖
FileWriter(File file, boolean append)	创建一个字符输出流对象， 用于向指定文件对象中写入数据，可以指定追加还是覆盖
void write(int c)	写一个字符
void write(char[] cbuf)	写一个字符数组
void write(String str)	写一个字符串
void write(String str, int off, int len)	写一个字符串的指定部分
void write(char[] cbuf, int off, int len)	写一个字符数组的指定部分

```
public static void main(String[] args)
{ //文件字符输出流
try(FileWriter fw = new FileWriter("E:\java-workplace\file_practice\demo1_practice.txt"))
{
fw.write("hello world");
fw.write("hello world");
//字符数组
char[] chs = {'a','b','c'};
fw.write(chs);
fw.write(chs,0,2); 写数组的一部分
//写字符串的一部分
fw.write("hello world",0,5);
} catch (IOException e) {
throw new RuntimeException(e);
}
}在try括号内创建的，所以自动关闭流了
```

- **文件输出流写出数据后，必须刷新流或者关闭流，写出去的数据才能生效，这是因为用了缓冲区书写，数据不会先写入文件，而是先写入缓冲区，需要刷新才会把缓冲区的数据同步到文件中**
方法	作用
void flush()	刷新缓冲区，将数据写出去
void close()	关闭流，刷新缓冲区，将数据写出去，并释放资源

- **缓冲流**：为了提高字符流，字节流的效率。有：BufferedReader、BufferedWriter、BufferedInputStream、BufferedOutputStream
  - **缓冲字节输入流**：BufferedInputStream：字节输入流，作用：提高字节输入流的效率。因为自带了8 KB的缓冲区。
方法	作用
BufferedInputStream(InputStream in)	创建一个带缓冲的字节输入流对象，作用：提高字节输入流的效率

  - ```
    public static void main(String[] args) throws IOException {
    //缓冲字节输入流
    FileInputStream fis = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt");
    BufferedInputStream bis = new BufferedInputStream(fis);
    int b;
    while((b = bis.read()) != -1)
    {
    System.out.print((char)b);
    }
    bis.close();
    }
    ```

  - **缓冲字节输出流**：BufferedOutputStream：字节输出流，作用：提高字节输出流的效率。因为自带了8 KB的缓冲区。
方法	作用
BufferedOutputStream(OutputStream out)	创建一个带缓冲的字节输出流对象，作用：提高字节输出流的效率

  - ```
    //缓冲字节输出流
    FileOutputStream fos = new FileOutputStream("E:\java-workplace\file_practice\demo1_practice.txt");
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    ```

```
bos.write("hello world".getBytes());
bos.close();
```

- **缓冲字符输入流**：BufferedReader：字符输入流，作用：提高字符输入流的效率。因为自带了8 K的字符缓冲区。

方法	作用
BufferedReader(Reader in)	创建一个带缓冲的字符输入流对象，作用： 提高字符输入流的效率
String readLine()	读取一行数据

```
//缓冲字符输入流
FileReader fr = new FileReader("E:\java-workplace\file_practice\demo1_practice.txt");
BufferedReader br = new BufferedReader(fr);
String line;
while((line = br.readLine()) != null)
{
System.out.println(line);
}
br.close();
```

- **缓冲字符输出流**：BufferedWriter：字符输出流，作用：提高字符输出流的效率。因为自带了8 K的字符缓冲区。

方法	作用
BufferedWriter(Writer out)	创建一个带缓冲的字符输出流对象，作用： 提高字符输出流的效率
void newLine()	换行，具体换行符由系统决定

```
//缓冲字符输出流
FileWriter fw = new FileWriter("E:\java-workplace\file_practice\demo1_practice.txt");
BufferedWriter bw = new BufferedWriter(fw);
bw.write("hello world");
bw.newLine();
bw.write("hello world");
bw.newLine();
bw.close();
```

- **性能分析**：建议用缓冲字节输入输出流，结合字节数组的方式，提高字节流读写数据的性能。但不绝对。用低级字节流把桶加大也可以其实
- **字符输入转换流**：InputStreamReader：字节输入转换流，作用：把字节输入流转换为字符输入流。当你的代码是UTF-8，而你的读取的文件是GBK时，会出现乱码。所以解决不同编码时，文

本内容乱码的问题

- 先获取文件的原始字节流，再将其按真实的字符集编码转换成字符输入流，就不乱码了
-

方法	作用
InputStreamReader(InputStream in, String charsetName)	把字节流转换成想要的字符流

-

```
public static void main(String[] args)
{
//字符输入转换流，构造方法
try(
FileInputStream isr = new FileInputStream("E:\java-workplace\file_practice\demo1_practice.txt");
InputStreamReader isr1 = new InputStreamReader(isr,"GBK");
){
int b;
while((b = isr.read()) != -1)
{
System.out.print((char)b);
}
}catch (IOException e){
e.printStackTrace();
System.out.println("文件不存在");
}
}
```

- **打印流**：PrintStream(继承自字节输出流)/PrintWriter(继承自字符输出流)
  - 作用：可以实现更方便，更高效的打印数据出去，能实现打印啥出去就是啥出去
  -

方法	作用
PrintStream(OutputStream out)	创建一个打印流对象，作用：实现更方便，更 efficient 的打印数据
PrintStream(String filePath)	创建一个打印流对象，作用：实现更方便，更 efficient 的打印数据
PrintStream(File file)	创建一个打印流对象，作用：实现更方便，更 efficient 的打印数据
void println(XXX xx)	打印指定数据类型的数据
void write(int/byte[]/byte[] 一部分)	支持写字节数据

```
//打印流
try(PrintWriter pw = new PrintWriter("E:\java-workplace\file_practice\demo1_practice.txt"))
{
pw.println("hello world");
pw.println("hello world");
pw.println(true);
pw.println(10);
pw.println(10.5);
pw.println('a');
} catch (FileNotFoundException e) {
throw new RuntimeException(e);
} 打印自动换行，打印10就是10，而不是别的字符，PrintWriter和PrintStream都是打印数据的，是一样的，只不过一个是字节流输出，一个是字符流输出，但是我们用.println()打印数据，所以两者没有区别
hello world
hello world
true
10
10.5
a
要是追加的话，需要用到低级流，添加true，打印流因为用到了缓冲流，所以不能追加
PrintWriter pw = new PrintWriter(new FileOutputStream("E:\java-workplace\file_practice\demo1_practice.txt", true)) 需要这种写法
```

- **特殊数据流**：DataInputStream(特殊字节输入流)/DataOutputStream(特殊字节输出流)
  - 输出的时候是整型97，到文件中去还是整型97，而不是字符串"97"，读入的时候读的就是整型97

方法	作用
DataInputStream(InputStream in)	创建一个特殊字节输入流对象，作用：把字节输入流转换为特殊字节输入流对象
DataOutputStream(OutputStream out)	创建一个特殊字节输出流对象，作用：把字节输出流转换为特殊字节输出流对象
int readInt()	读取一个int类型的数据
void writeInt(int i)	写入一个int类型数据
String readUTF()	读取一个字符串
void writeUTF(String s)	写入一个字符串

方法	作用
long readLong()	读取一个long类型数据
void writeLong(long l)	写入一个long类型数据
double readDouble()	读取一个double类型数据
void writeDouble(double d)	写入一个double类型数据
boolean readBoolean()	读取一个boolean类型数据
void writeBoolean(boolean b)	写入一个boolean类型数据
char readChar()	读取一个char类型数据
void writeChar(char c)	写入一个char类型数据
byte readByte()	读取一个byte类型数据
void writeByte(byte b)	写入一个byte类型数据
void write(byte[] b)	写入字节数据
void write(byte[] b, int off, int len)	写入字节数据一部分

```
public static void main(String[] args) throws FileNotFoundException {
//特殊数据流，特殊字节输入流构造器
FileInputStream fis = new FileInputStream("E:\java-
workplace\file_practice\demo1_practice.txt");
DataInputStream dis = new DataInputStream(fis);
try(dis){
System.out.println(dis.readInt());
System.out.println(dis.readBoolean());
System.out.println(dis.readChar());
} catch (Exception e){
e.printStackTrace();
}
//特殊字符输出流
FileOutputStream fos = new FileOutputStream("E:\java-
workplace\file_practice\demo1_practice.txt");
DataOutputStream dos = new DataOutputStream(fos);
try(dos){
dos.writeInt(100);
dos.writeBoolean(true);
dos.writeChar('a');
} catch (Exception e){
```

```
e.printStackTrace();
}
}特殊数据输出流是为了给别人输数据用的，让别人读的，不是给看的，自己看也是乱码
```

- **IO框架**
  - 框架：是一个预先写好的代码库或一组工具，旨在简化和加速开发过程
  - 框架的形式：一般是把类、接口等编译成class形式，再压缩成一个.jar包，然后发行出去
  - IO框架是指封装了java提供的对文件、数据进行操作的代码，对外提供了更简单的方式对文件进行操作，对数据进行读写等
  - 导入commons-io-2.21.0.jar包
    - 在项目中创建一个文件夹：lib
    - 将commons-io-2.21.0.jar包复制放入lib文件夹中(去搜找commons-io下载)
    - 在jar文件点右键，选择Add as Library -> 点击OK
    - 在类中导包使用,import org.apache.commons.io.FileUtils;
  -

方法	作用
FileUtils.copyFile(File srcFile, File destFile)	复制文件
FileUtils.copyDirectory(File srcDir, File destDir)	复制文件夹
FileUtils.deleteQuietly(File file)	删除文件
FileUtils.readFileToString(File file,String encoding)	读取文件
FileUtils.writeStringToFile(File file, String data,String charsetName,boolean append)	写入文件

  - ```
    public static void main(String[] args)
    {//Commom-io框架的使用
    try {
    FileUtils.copyFile(new java.io.File("E:\java-workplace\file_practice\demo1_practice.txt"),new java.io.File("E:\java-workplace\file_practice\demo1_practice_copy.txt"));
    System.out.println("复制成功！");
    } catch (Exception e) {
    e.printStackTrace();
    }
    }
    ```
    ```