IO流

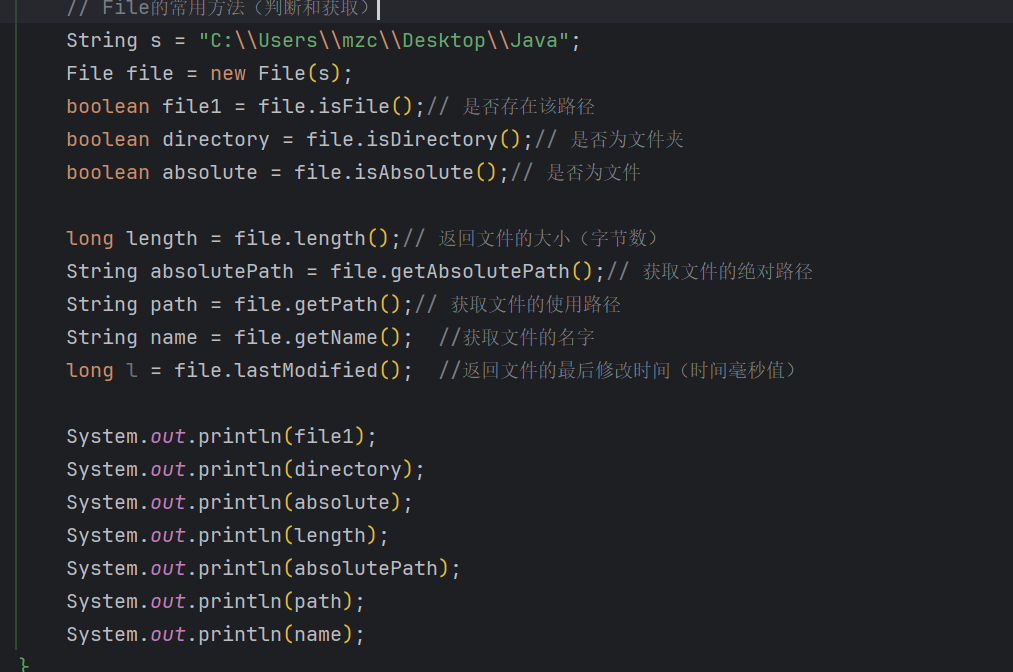
1. File（路径）
2. 作用：创建数据保存的路径
3. File路径：路径可以存在也可以不存在
4. File构造方法：（1）用字符串表示路径，变成File对象

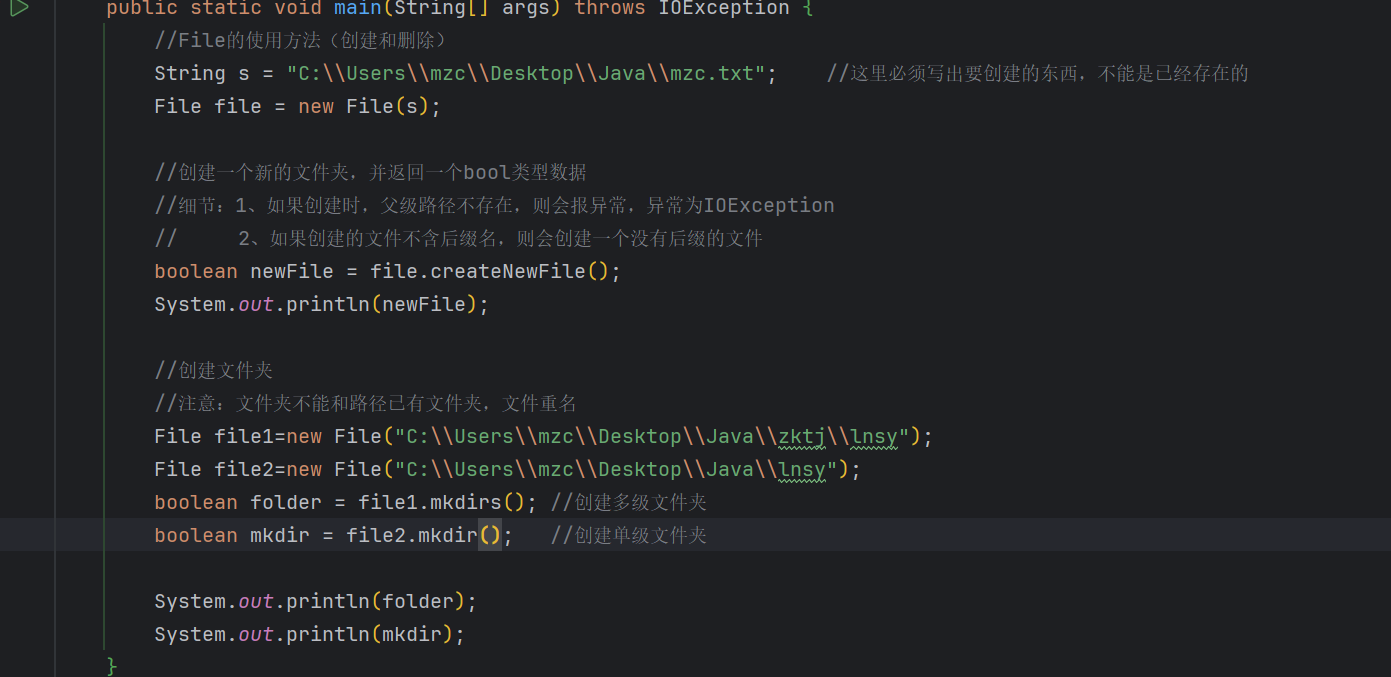
（2）父集和子集拼接：在创建新的File对象，参数列表加父、子集

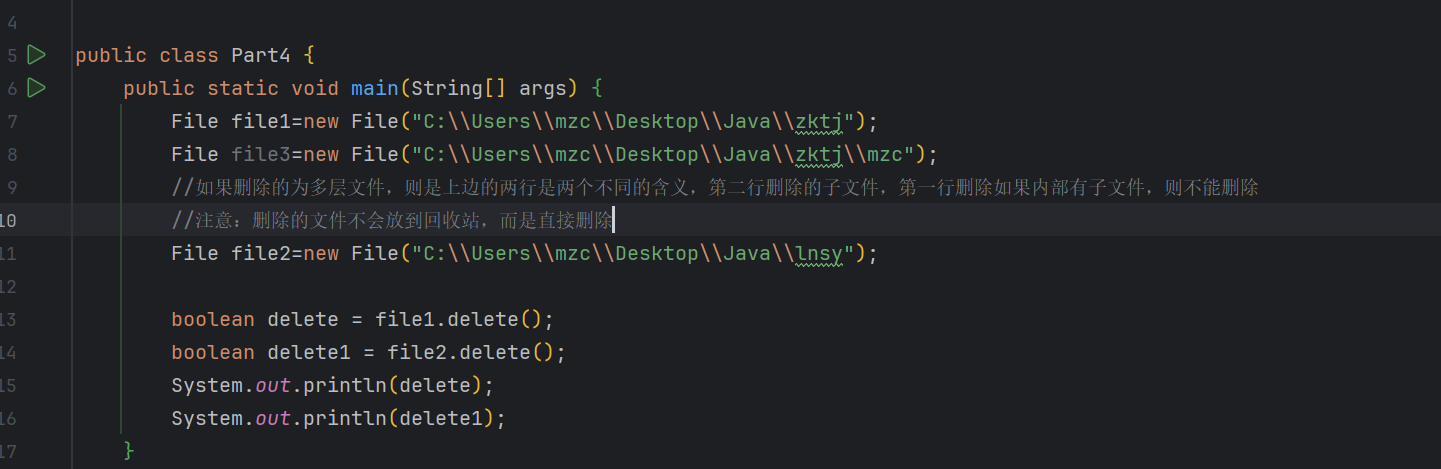
（3）字符串和File对象拼接：参数列表分别加入这两个

4、路径的分类：绝对路径：带盘符，如C：

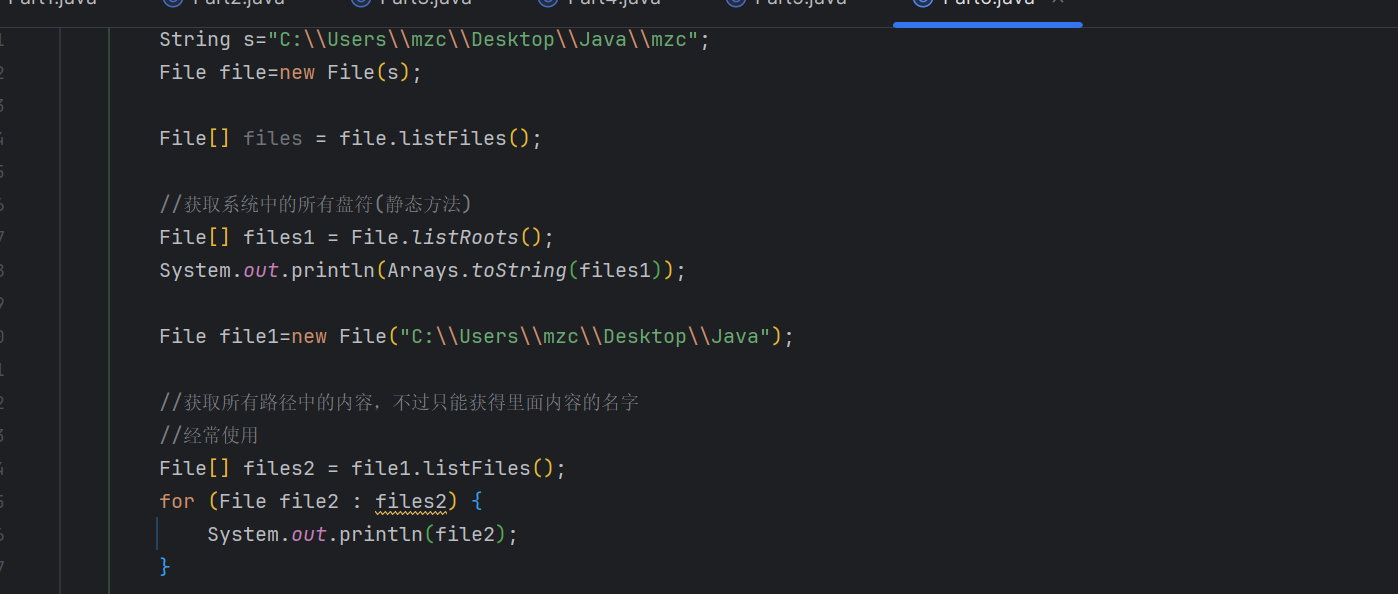
相对路径：不带盘符，一般会在当前项目下找

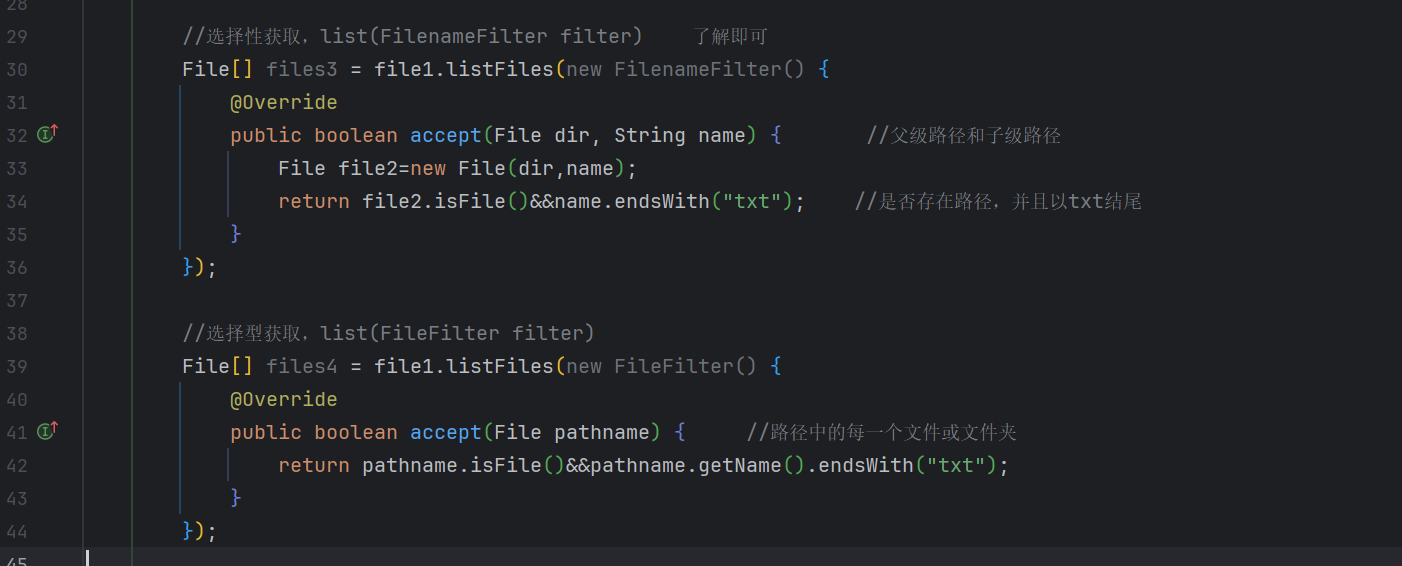
1. 常见方法：











1. 创建文件和文件夹的标准流程
2. public static void main(String[] args) {  
    //创建文件和文件夹的标准流程  
    File folder=new File("C:\\Users\\mzc\\Desktop\\Java\\mzc");  
    if(!folder.exists()){ //判断文件夹是否存在  
    boolean mkdir = folder.mkdir();  
    if(!mkdir){  
    System.*out*.println("创建文件夹失败");  
    return;  
    }  
    System.*out*.println("文件夹创建成功");  
    }else{  
    System.*out*.println("文件夹已存在");  
    }  
     
    File myFile=new File("C:\\Users\\mzc\\Desktop\\Java\\mzc\\a.txt");  
    if(!myFile.exists()){ //判断文件是否存在  
    try {  
    boolean newFile = myFile.createNewFile();  
    if(!newFile){  
    System.*out*.println("文件创建失败");  
    }else{  
    System.*out*.println("文件创建成功");  
    }  
    } catch (IOException e) {  
    throw new RuntimeException(e);  
    }  
    }else{  
    System.*out*.println("文件已存在");  
    }  
     
   }
3. IO流
4. 概述：存储和读取数据的解决方案
5. IO的分类：（1）流的方向：输入流、输出流

（2）操作的文件类型：字节流和字符流

字节流：所有类型文件

字符流：纯文件（记事本打开能看懂的）

1. 作用：用于读写数据（本地或网络）
2. IO流的体系：字节流和字符流
3. 字节流：字节输入（InputStream）、字节输出（OutputStream）

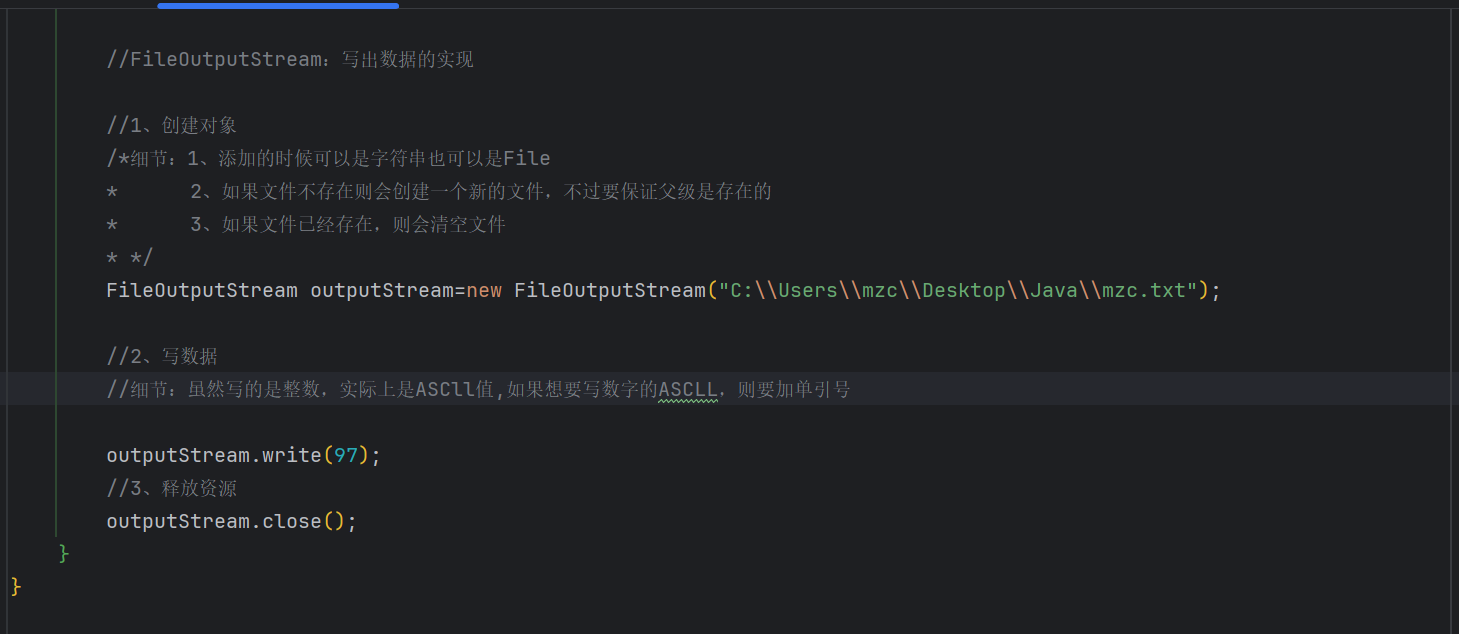
* 作用：**字节流**：以字节为单位处理数据。常用于处理二进制数据（如图像、音频、视频、以及其他非文本文件）。

1. 字符流: 字符输入（Read）、字符输出（Writer）

注意：这些方法都为抽象类不能够直接创建

1. 方法的子类：在词前加上表示作用的前缀
2. 实现：

OutputSteam：（1）



（2）写出数据的3种方式

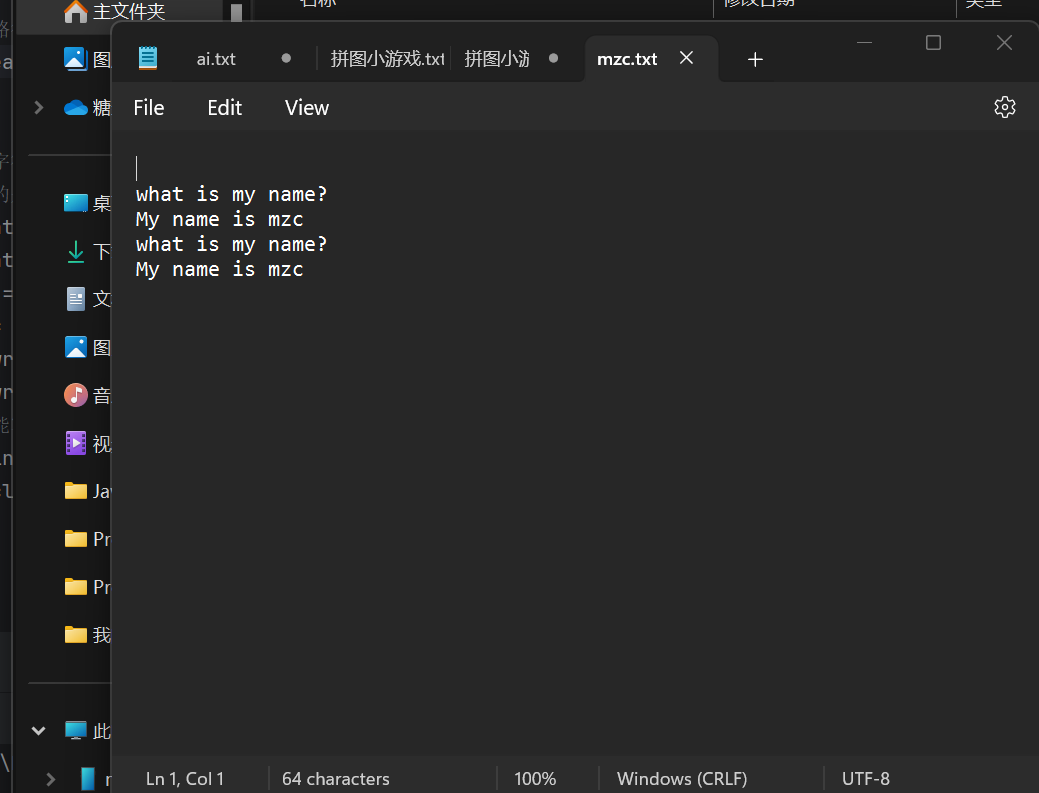
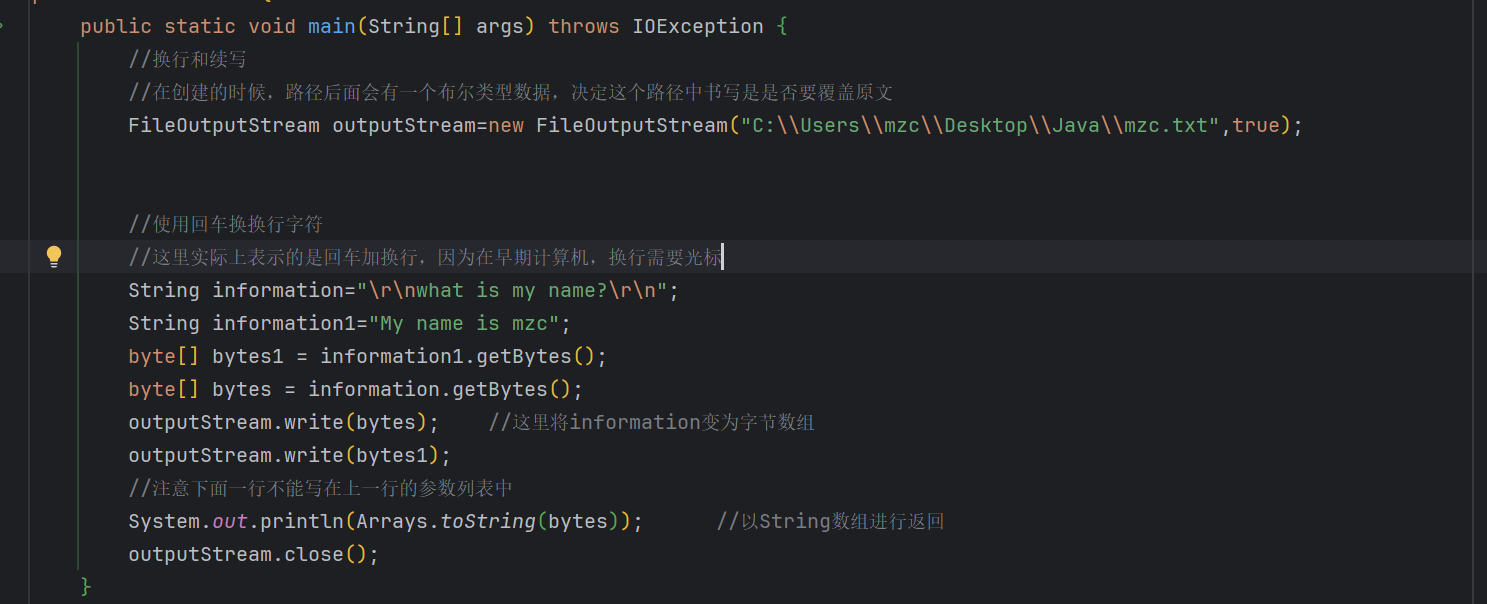
Void write(int b)

Void write(byte[] b) 先写一个字符数组，然后再添加进去

Void Write（byte[] b, int off, int len）依次写数组的一部分数据

（3）换行和续写

换行：使用换行字符，\r\n

续写：在创建的时候，参数列表中路径后面加上布尔类型数据，True为可以续写，False为不可以续写

1. FileInputStream

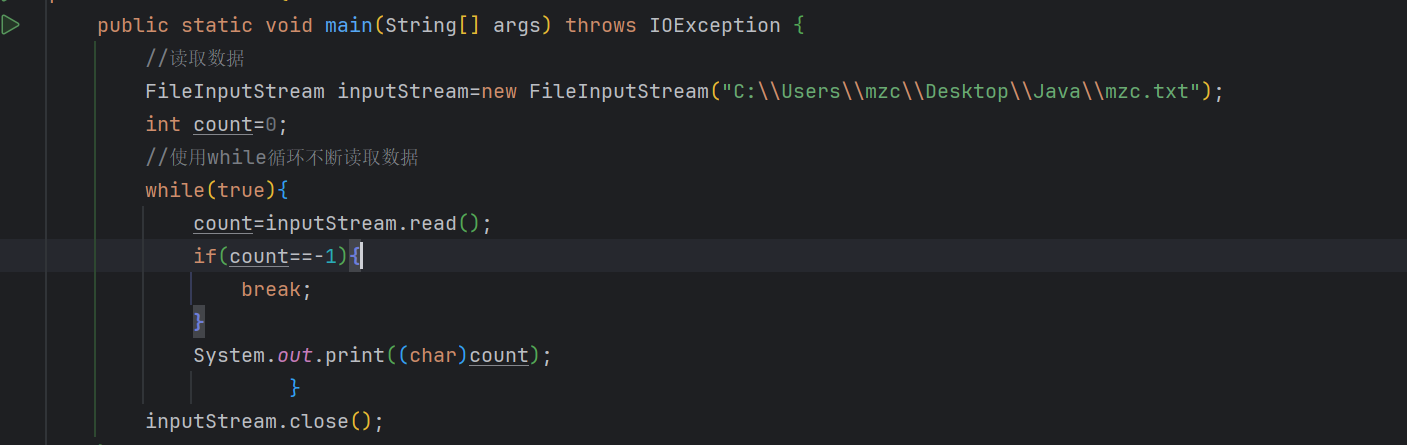
作用：字节输入流，可以把本地的文件读取出来

语法：创建一个FileInputStream对象，在参数列表中，写明要读取文件的位置。

注意：这里不能是不存在的文件，因为在读取的时候，本身就没有数据可以读取，在创建一个新的文件没有意义。

而输出流是可以的，因为数据在程序中，即使文件不存在，也可以创建一个新的文件，将数据存入

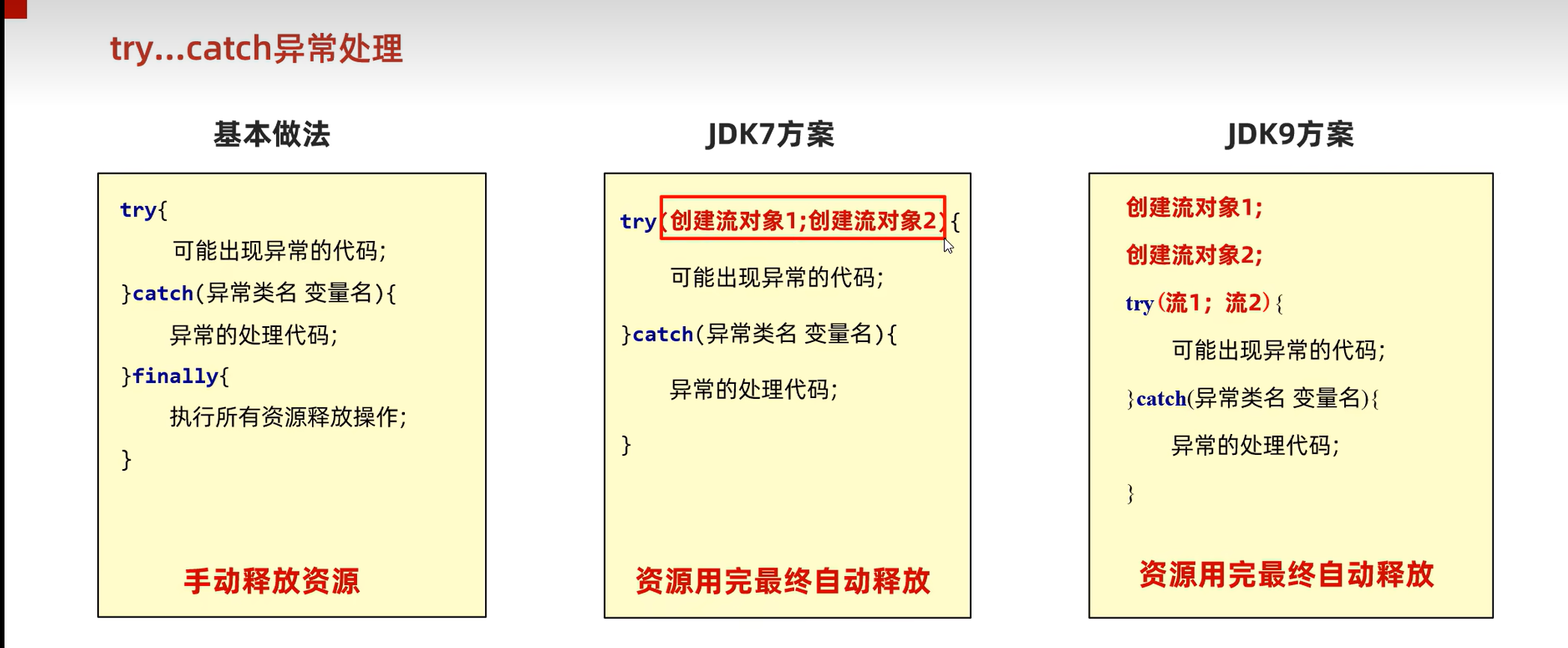
1. 文件的拷贝：

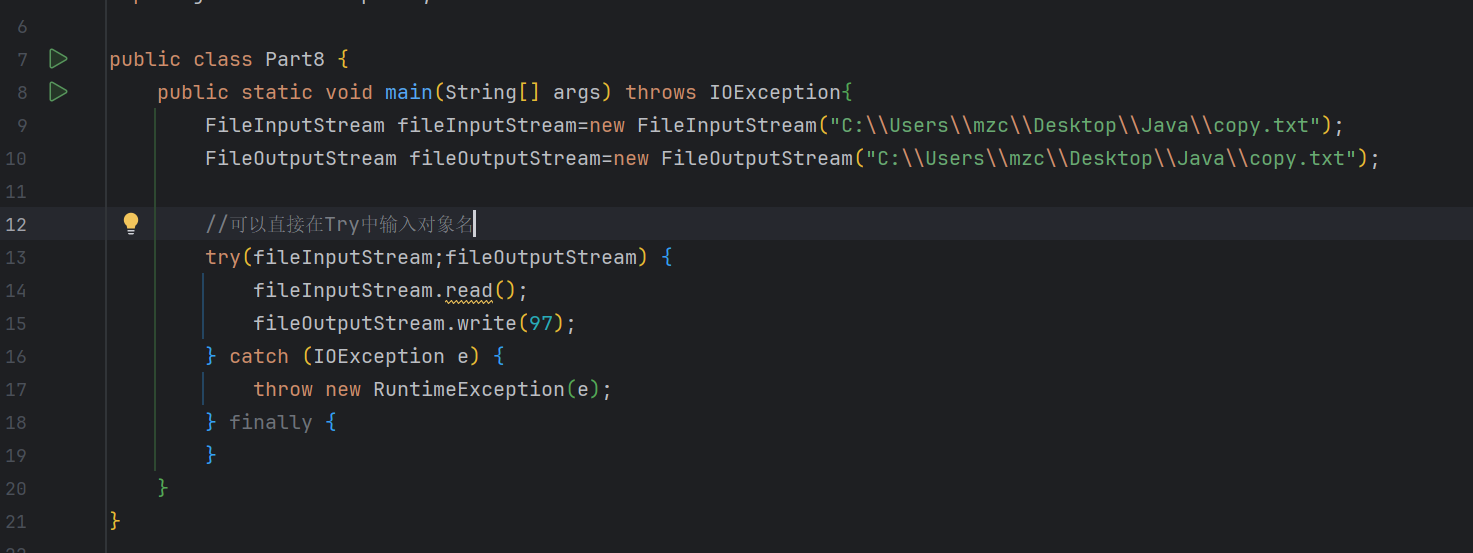
（1）策略：边读边写

（2）弊端与解决方案：弊端：由于这里采取的是一个字节一个字节的去读取，运行的速度非常的慢

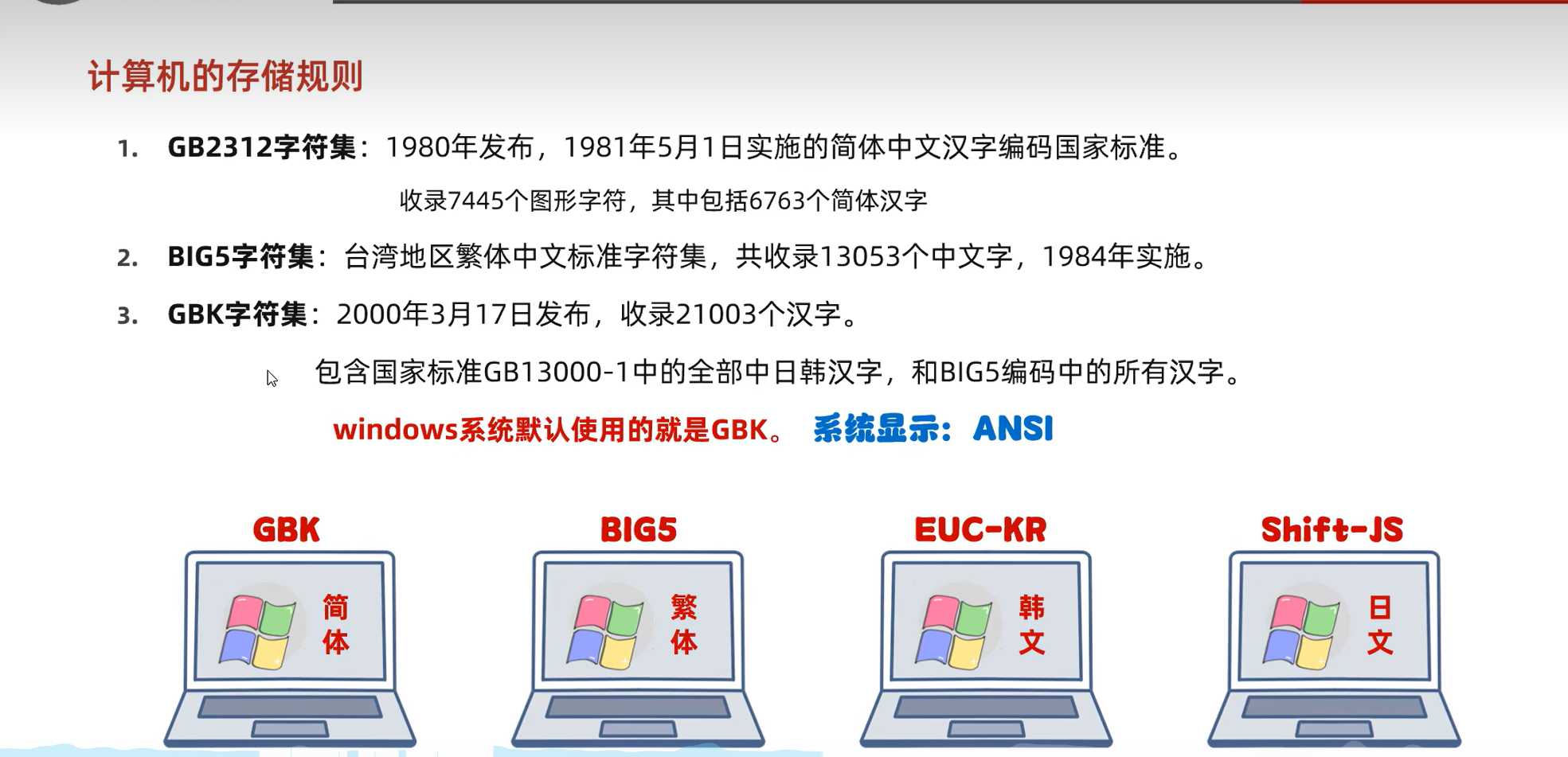
解决方案：



1. IO流中不同JDK版本捕获异常的方式
2. 异常处理的完整版是在Try catch的后面加上finally，里面书写的代码一定会被执行
3. IO流中处理异常的方法：（1）基本方法：通常创建对象放在Try catch的外面，并赋值为空，然后再在里面书写内容，最后在finally中书写close方法。
4. （2）JDK7：在JDK7之后，添加了新的功能，AutoClose方法，先接入接口AutoClose，然后再在Try的后面书写参数列表，将要释放的对象在此处进行创建，写进去。
5. （3）JDK9：对于AutoClose进行了新的优化，不用再参数列表中创建对象，在外面创建对象，然后将对象名写在参数列表中
6. 



1. 字符集
2. 原理：在计算集中采取的是二进制，有0和1组成，叫做bit，而每8个构成一个字节，在输入英文字母的时候，每个字母只占一个字节，因此对于刚一开始是够用的，但是随着使用计算机的国家越来越多，此时，存储其它语言，就显得捉襟见肘，因此引入了字符集
3. 字符集的分类



存储英文：（1）输入字母，一个字母占一个字节，如果字母的不够占满一个字节，则会在前面进行补0操作

（2）输入汉字：大小：占两个字节，其中分为高位字节，低位字节。高位字节必须以1开头，存储到硬盘中不需要进行任何的变动，一般在转化为十进制的数字后，高位字节一定为负数，低位字节，则都有可能

存储规则：输入汉字，查找GBK，编码成二进制。

获得汉字：解码，查找GBK，变为汉字

1. Unicode字符集（万国码）
2. 演变：从UTF-16到UTF-32再到UTF-8，不同点：16代表的是使用最多的bit位为16，

32则是直接固定使用4个字节，8是动态使用1-4个字节

1. 中、英：在Unicode中，中文占3个字节，英文（ASCll）占1个字节
2. 注意：UTF-8实质并不是一个字符集，而是一种编码方式
3. 出现乱码的原因：（1）未能读取完整的数据，例如：汉字

如果使用字节流进行读取的时候，因为一次只能读取一个字节，因此无法读取完整的汉字

注意：即使一次读取多个字节，也会因为字节流读取数据不考虑字符编码问题，只是单纯的读取字节

（2）编码和读码使用的规则不一致

（3）扩展：字节流读取的时候会出现乱码，在进行拷贝的时候则不会

原因：在拷贝的过程中，是一个字节一个字节的拷贝并不会出现字节丢失的情况，如果在目标文件中，它的读码方式和原来的文件的编码方式是一样的，则可以进行正常的读取。

1. 编码和解码方式不同的解决方法：



1. 字符流
2. 作用：上边已经解决了因为编码和解码方式不同而出现的问题，而字符流就是为了解决不能够完全读取数据的问题
3. 底层原理：（1）字符流就是将字节流加上编码方式，解决了字节流不进行编码的问题。

（2）具体原理：其底层也是一个一个字节的进行读取，并将其转化为十进制；当读取到汉字或其他非ASCLL值的数据时，会一次读取多个字节，同样也转化为十进制。并将十进制数进行返回，因此解码的时候只需要将其强转为char类型的数据就可以了



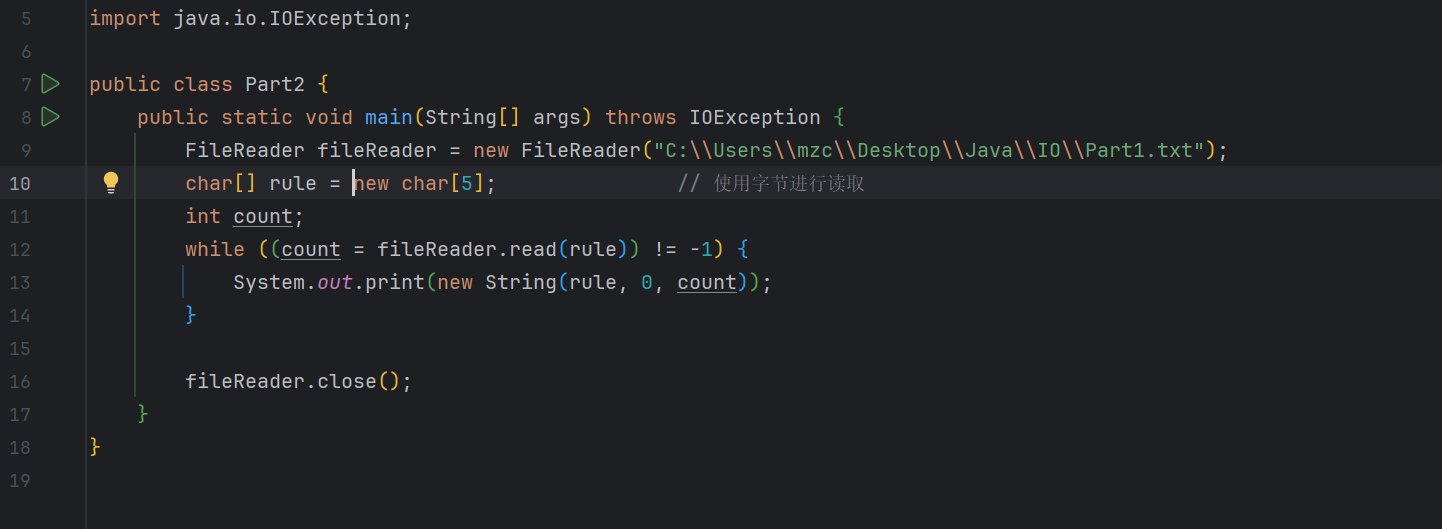
1. 子类：reader（输入流），writer（输出流）
2. **字符流**：以字符为单位处理数据。适用于处理文本数据。
3. FileReader：（1）



（2）代码演示（空参）：



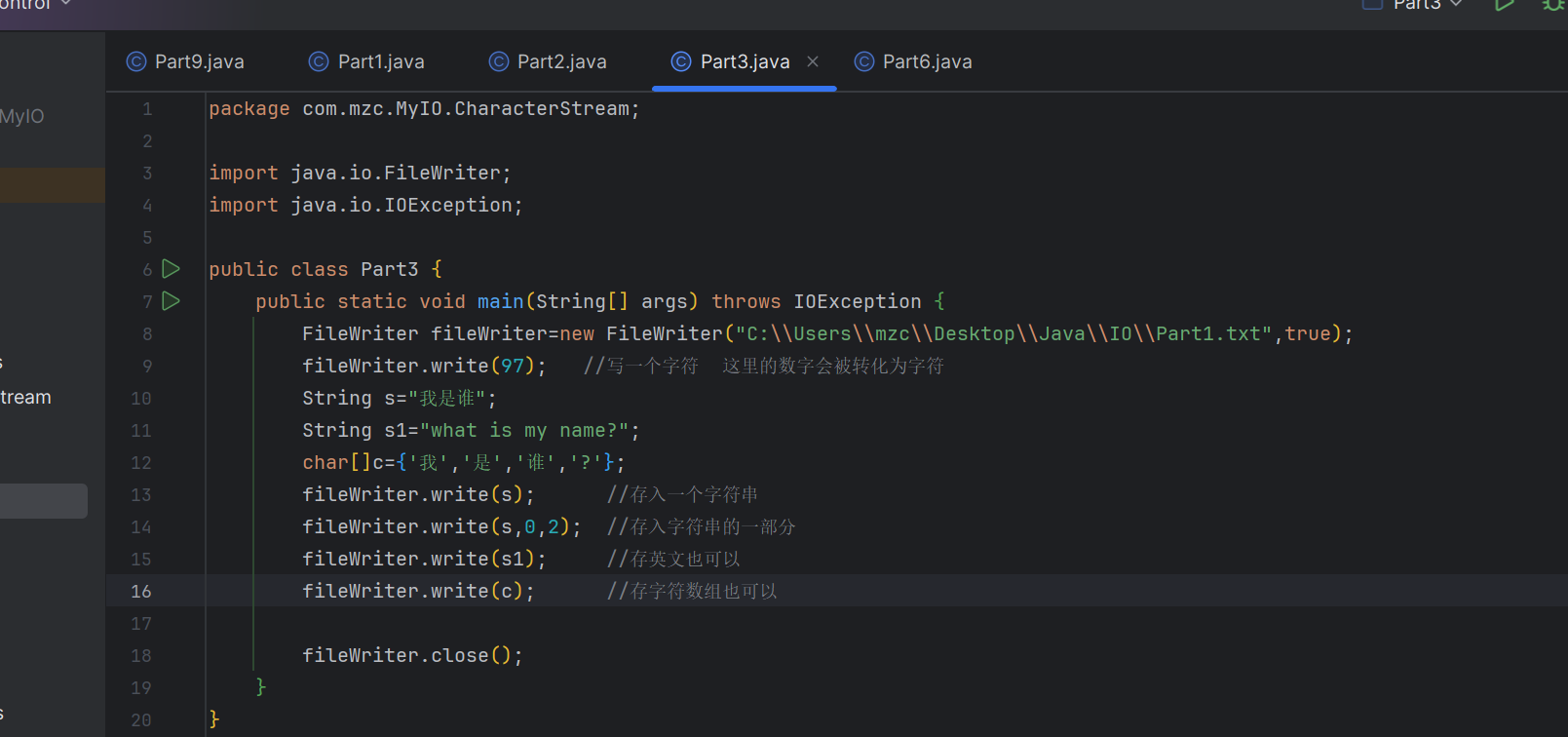
（3）有参构造：原理：read(chars) chars为字符数组，将读取数据，解码，强转结合在了一起



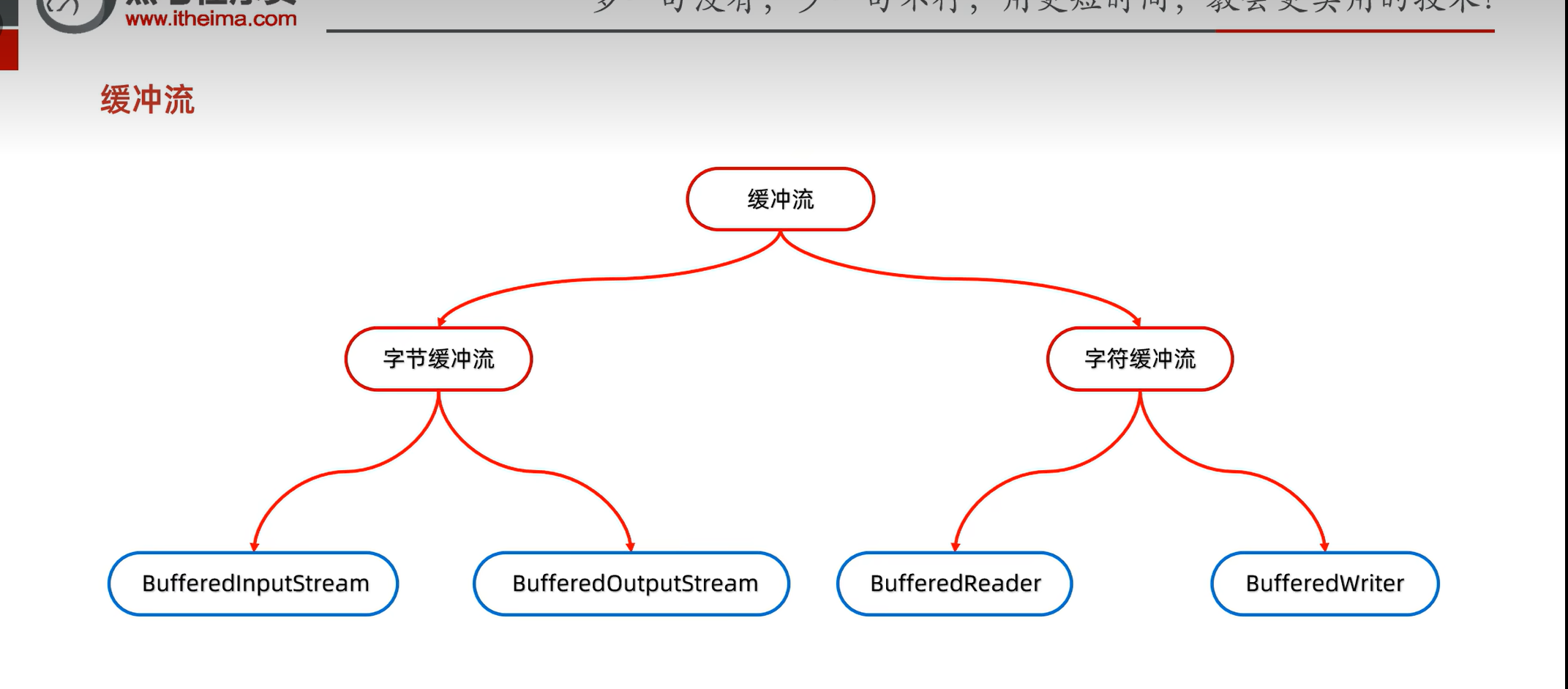
1. FileWriter构造方法

（1）成员方法：一个字符、一个字符串或一部分、一个字符数组或其中一部分

（2）代码演示：



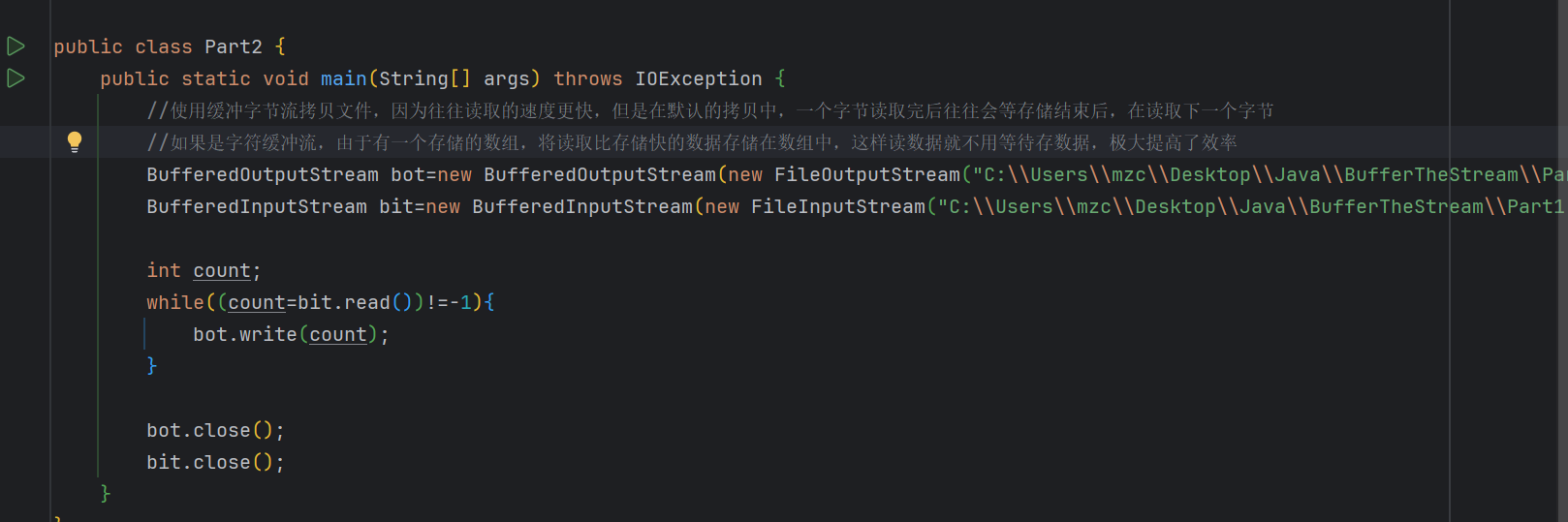
1. 字符缓冲流
   1. 字节缓冲流：（1）一次读写一个字节



（2）原理：在字节流的基础上，自带长度为8192的缓冲区提高性能

（3）代码演示：使用缓冲流拷贝文件

一个字节进行拷贝

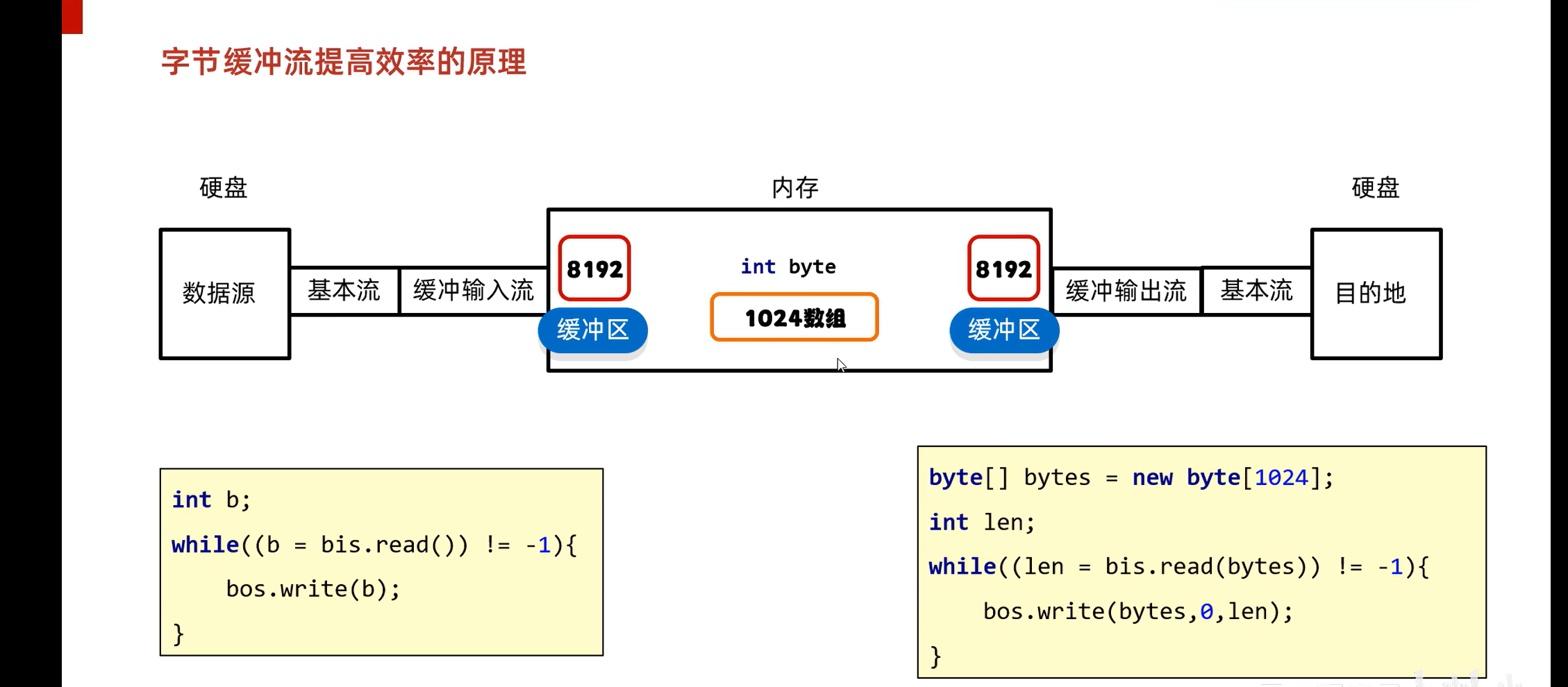


多个字节进行拷贝



是写更高读取速度的原因：

读和写分别创建了两个大小为8192的数组，而数组是放在内存中的，而内存的执行效率非常的高，每次读的数据会先存储到读的数组中，当数组满的时候，会一个字节一个字节的传输给写的数组，当写的数组满的时候会传给写的；而多个字节进行读取，就是在数组之间进行传递的时候，再创建一个数组，在读写数组之间进行交换的时候，交给这个数组，提高了交换的速度。



* 1. 字符缓冲流：
     1. 字符缓冲流的特有方法：readLine（） 依次读取一行数据

newline（）跨平台的换行，即Windows是\r\n，Linu是\n，这个可以实现在哪个平台都可以