I/O

@M了个J
李明杰

https://github.com/CoderMJLee
http://cnblogs.com/mjios

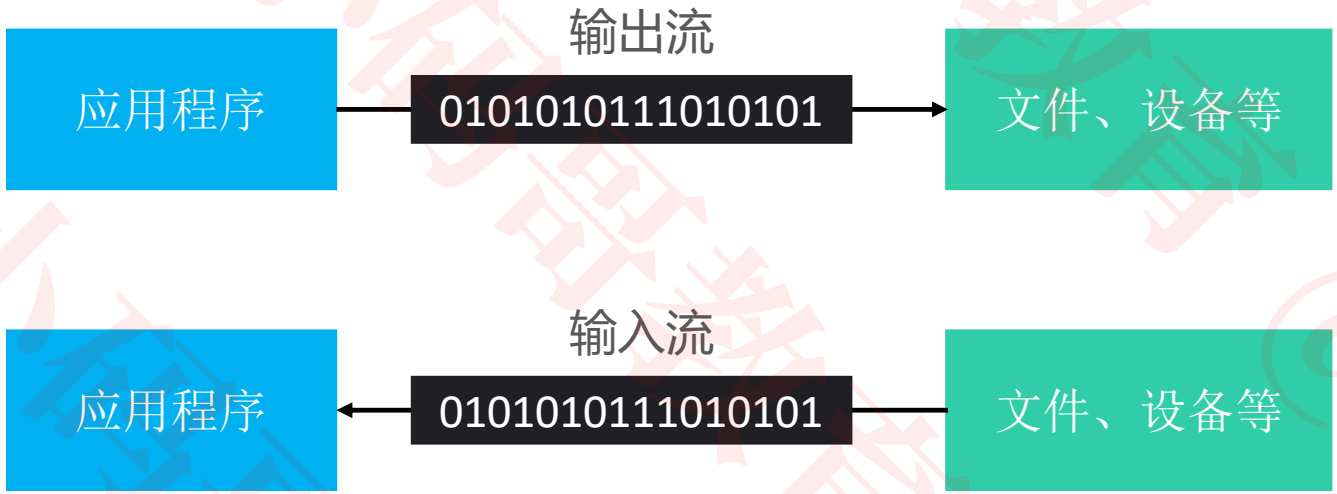
码拉松

小码哥教育 SEEMYGO

实力IT教育 www.520it.com

I/O 流

- I/O 流 全称是 Input/Output Stream，译为“输入/输出流”



小码哥教育 SEEMYGO

常用类型

- I/O 流的常用类型都在 java.io 包中

类型	输入流	输出流
字节流（Byte Streams）	InputStream	OutputStream
字符流（Character Streams）	Reader	Writer
缓冲流（Buffered Streams）	BufferedInputStream BufferedReader	BufferedOutputStream BufferedWriter
数据流（Data Streams）	DataInputStream	DataOutputStream
对象流（Object Streams）	ObjectInputStream	ObjectOutputStream

File

- 一个 File 对象就代表一个文件或目录（文件夹）

```
// file1、file2都能访问test.txt文件
File file1 = new File("F:\\Files\\Texts\\test.txt");
File file2 = new File("F:/Files/Texts/test.txt");
```

- 名字分隔符（name separator）：File.*separator*
  - 在 UNIX、Linux、Mac 系统中：正斜杠（/）
  - 在 Windows 系统中：反斜杠（\）

- 路径分隔符（path separator）：File.*pathSeparator*
  - 在 UNIX、Linux、Mac系统中：冒号（:）
  - 在 Windows 系统中：分号（;）

- 在 Windows、Mac 系统中
  - 文件名、目录名不区分大小写

- 在 UNIX、Linux 系统中
  - 文件名、目录名区分大小写

小码哥教育 SEEMYGO

File – 常用方法

```
String getName() // 获取文件或目录的名称
String getParent() // 获取父路径
File getParentFile() // 获取父文件
String getPath() // 获取路径
String getAbsolutePath() // 获取绝对路径
File getAbsoluteFile() // 获取绝对路径形式的文件
long lastModified() // 最后一次修改的时间
long length() // 文件的大小（不支持目录）
```

```
boolean isAbsolute()
boolean exists()
boolean isDirectory()
boolean isFile()
boolean isHidden()
boolean canRead()
boolean canWrite()
```

小码哥教育 SEEMYGO

File – 常用方法

```
String[] list() // 获取当前目录下所有文件、目录的名称
String[] list(FilenameFilter filter)
File[] listFiles() // 获取当前目录下所有文件、目录
File[] listFiles(FilenameFilter filter)
File[] listFiles(FileFilter filter)
```

```
boolean createNewFile() // 创建文件（不会覆盖旧文件）
boolean delete() // 删除文件或空目录（不经过回收站）
boolean mkdir() // 创建当前目录
boolean mkdirs() // 创建当前目录（包括不存在的父目录）
boolean renameTo(File dest) // 剪切到新路径
```

```
boolean setLastModified(long time)
boolean setReadOnly()
boolean setWritable(boolean writable, boolean ownerOnly)
boolean setWritable(boolean writable)
boolean setReadable(boolean readable, boolean ownerOnly)
boolean setReadable(boolean readable)
```

小码哥教育 SEEMYGO

练习 – 搜索

```
public static void search(File dir, Consumer<File> operation) {
    if (dir == null || operation == null) return;
    if (!dir.exists() || dir.isFile()) return;
    File[] subfiles = dir.listFiles();
    for (File sf : subfiles) {
        operation.accept(sf);
        if (sf.isFile()) continue;
        search(sf, operation);
    }
}
```

小码哥教育 SEEMYGO

练习 – 删除

```
public static void delete(File file) {
    if (file == null || !file.exists()) return;
    clean(file);
    file.delete();
}

public static void clean(File dir) {
    if (dir == null || !dir.exists() || dir.isFile()) return;
    File[] subfiles = dir.listFiles();
    for (File sf : subfiles) {
        delete(sf);
    }
}
```

小码哥教育 SEEMYGO

练习 – 剪切

```
private static void mkparents(File file) {
    File parent = file.getParentFile();
    if (parent == null) return;
    parent.mkdirs();
}
```

```
public static void move(File src, File dest) {
    if (src == null || dest == null) return;
    if (!src.exists() || dest.exists()) return;
    mkparents(dest);
    src.renameTo(dest);
}
```

字符集（Character Set）

- 在计算机里面
  - 一个中文汉字是一个字符
  - 一个英文字母是一个字符
  - 一个阿拉伯数字是一个字符
  - 一个标点符号是一个字符
  - ……

- 字符集（简称 Charset）：由字符组成的集合

- 常见的字符集有
  - ASCII：128个字符（包括了英文字母大小写、阿拉伯数字等）
  - ISO-8859-1：支持欧洲的部分语言文字，在有些环境也叫 Latin-1
  - GB2312：支持中文（包括了 6763 个汉字）
  - BIG5：支持繁体中文（包括了 13053 个汉字）
  - GBK：是对 GB2312、BIG5 的扩充（包括了 21003 个汉字），支持中日韩
  - GB18030：是对 GBK 的扩充（包括了 27484 个汉字）
  - Unicode：包括了世界上所有的字符

- ISO-8859-1、GB2312、BIG5、GBK、GB18030、Unicode 中都已经包括了 ASCII 中的所有字符

字符编码（Character Encoding）

- 每个字符集都有对应的字符编码，它决定了每个字符如何转成二进制存储在计算机中
- ASCII：单字节编码，编码范围是 0x00 ~ 0x7F （0 ~ 127）
- ISO-8859-1：单字节编码，编码范围是 0x00 ~ 0xFF
  - 0x00 ~ 0x7F 和 ASCII 一致，0x80 ~ 0x9F 是控制字符，0xA0 ~ 0xFF 是文字符号
- GB2312、BIG5、GBK：采用双字节表示一个汉字
- GB18030：采用单字节、双字节、四字节表示一个字符
- Unicode：有 Unicode、UTF-8、UTF-16、UTF-32 等编码，最常用的是 UTF-8 编码
  - UTF-8 采用单字节、双字节、三字节、四字节表示一个字符

小码哥教育 SEEMYGO

字符编码比较

```
String str = "MJ码哥";
str.getBytes("ASCII")          // [77, 74, 63, 63]
str.getBytes("ISO-8859-1")     // [77, 74, 63, 63]
str.getBytes("GB2312")         // [77, 74, -62, -21, -72, -25]
str.getBytes("BIG5")           // [77, 74, 63, -83, -12]
str.getBytes("GBK")            // [77, 74, -62, -21, -72, -25]
str.getBytes("GB18030")        // [77, 74, -62, -21, -72, -25]
str.getBytes("UTF-8")          // [77, 74, -25, -96, -127, -27, -109, -91]
```

- 如果 String.getBytes 方法没有传参，就使用 JVM 的默认字符编码，一般跟随 main 方法所在文件的字符编码
- 可以通过 Charset.*defaultCharset* 方法获取 JVM 的默认字符编码
  - Charset 类的全名是 java.nio.charset.Charset

小码哥教育 SEEMYGO

乱码

- 一般将【字符串】转为【二进制】的过程称为：编码（Encode）
- 一般将【二进制】转为【字符串】的过程称为：解码（Decode）

- 编码、解码时使用的字符编码必须要保持一致，否则会造成乱码

```
String str1 = "Java不难";
// 编码：[74, 97, 118, 97, -28, -72, -115, -23, -102, -66]
byte[] bytes = str1.getBytes("UTF-8");
// 解码：Java涓嶉毦
String str2 = new String(bytes, "GB18030");
```

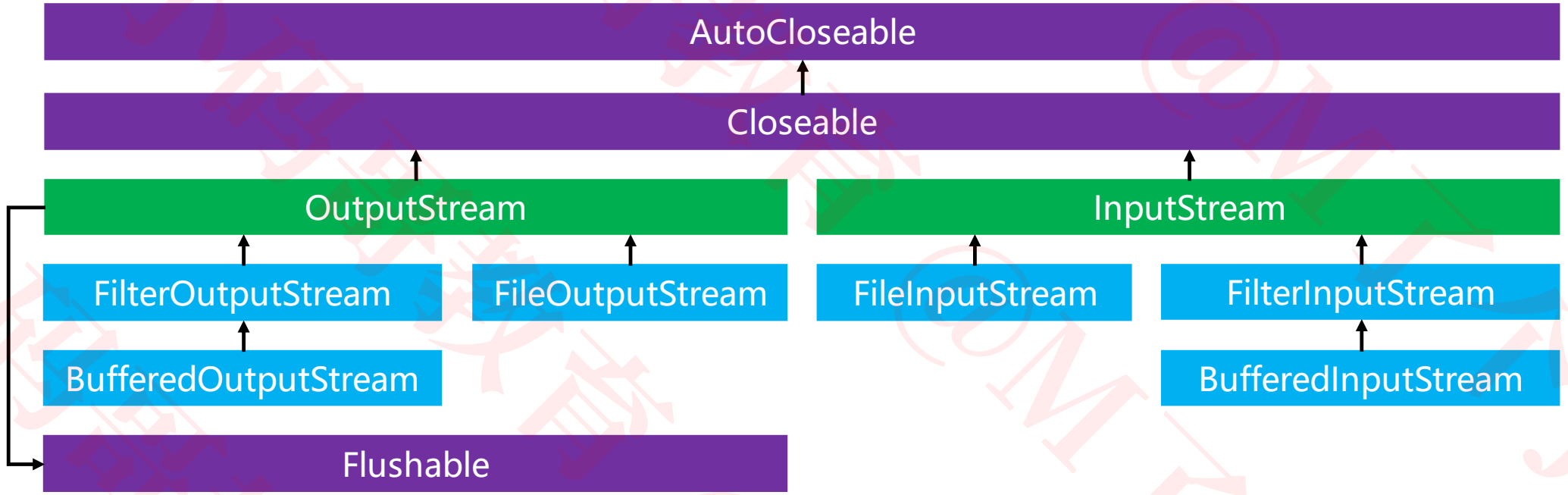
字节流（Byte Streams）

- 字节流的特点
  - 一次只读写一个字节
  - 最终都继承自 InputSteam、OutputStream

- 常用的是字节流有 FileInputStream、FileOutputStream

小码哥教育 SEEMYGO

字节流结构预览



小码哥教育 SEEMYGO

FileOutStream

```
OutputStream os = new FileOutputStream("F:/1.txt");
os.write(77); // M
os.write(74); // J
os.close();
```

```
// true表示追加内容，并非覆盖原来内容
OutputStream os = new FileOutputStream("F:/2.txt", true);
os.write("MJ码哥".getBytes());
os.close();
```

小码哥教育 SEEMYGO

FileInputStream

```
InputStream is = new FileInputStream("F:/2.txt");
// 读取第1个字节
int byte1 = is.read();
// 读取第2个字节
int byte2 = is.read();
is.close();
```

```
InputStream is = new FileInputStream("F:/2.txt");
byte[] bytes = new byte[1024];
// read返回实际读取的字节数
int len = is.read(bytes);
is.close();
```

小码哥教育 SEEMYGO

练习 – 将内存中的数据写入文件

```
public static void write(byte[] data, File file) {
    if (data == null || file == null || file.exists()) return;
    mkparents(file);
    try (
        OutputStream os = new FileOutputStream(file);
    ) {
        os.write(data);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

小码哥教育 SEEMYGO

练习 – 从文件读取数据到内存

```
public static byte[] read(File file) {
    if (file == null || !file.exists()) return null;
    if (file.isDirectory()) return null;
    try (
        InputStream is = new FileInputStream(file);
    ) {
        byte[] data = new byte[(int) file.length()];
        is.read(data);
        return data;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

小码哥教育 SEEMYGO

练习 – 复制

```
public static void copy(File src, File dest) {
    if (src == null || dest == null) return;
    if (!src.exists() || dest.exists()) return;
    if (src.isDirectory()) return;
    mkparents(dest);
    InputStream is = null;
    OutputStream os = null;
    try {
        is = new FileInputStream(src);
        os = new FileOutputStream(dest);
        byte[] data = new byte[8192];
        int len;
        while ((len = is.read(data)) != -1) { os.write(data, 0, len); }
    } catch (FileNotFoundException e) {} catch (IOException e) {}
    finally {
        if (is != null) { try { is.close(); } catch (IOException e) {} }
        if (os != null) { try { os.close(); } catch (IOException e) {} }
    }
}
```

try-with-resources 语句

- 下图就是从 Java 7 开始推出的 try-with-resources 语句（可以没有 catch、finally）

```
try (资源1; 资源2; ...) {

} catch(Exception e) {

} finally {

}
```

- 可以在 try 后面的小括号中声明一个或多个资源（resource）
  - 实现了 java.lang.AutoCloseable 接口的实例，都可以称之为是资源

- 不管 try 中的语句是正常还是意外结束
  - 最终都会自动按顺序调用每一个资源的 close 方法（close 方法的调用顺序与资源的声明顺序相反）
  - 调用完所有资源的 close 方法后，再执行 finally 中的语句

小码哥教育 SEEMYGO

try-with-resources 语句 – 示例

```
public static void copy(File src, File dest) {
    if (src == null || dest == null) return;
    if (!src.exists() || dest.exists()) return;
    if (src.isDirectory()) return;
    mkparents(dest);
    try (
        InputStream is = new FileInputStream(src);
        OutputStream os = new FileOutputStream(dest)) {
        byte[] data = new byte[8192];
        int len;
        while ((len = is.read(data)) != -1) {
            os.write(data, 0, len);
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

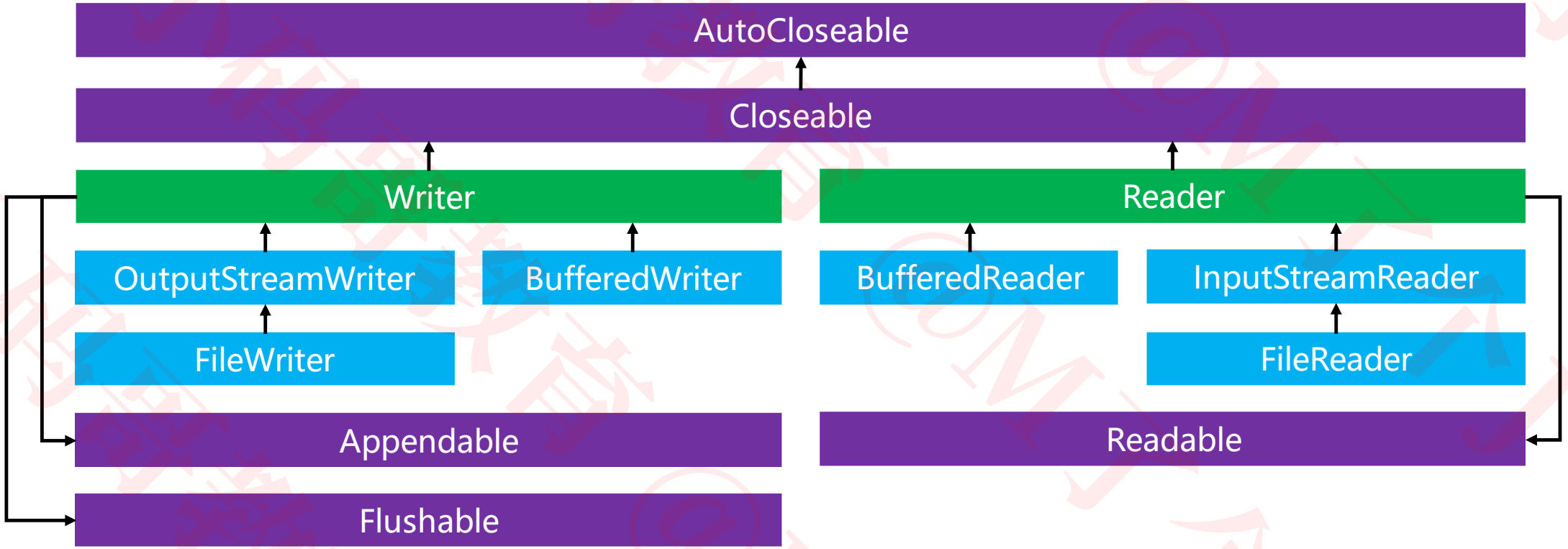
字符流（Character Streams）

- 字符流的特点
  - 一次只读写一个字符
  - 最终都继承自 Reader、Writer

- 常用的是字符流有 FileReader、FileWriter
  - 注意：这 2 个类只适合文本文件，比如 .txt、.java 等这类文件

小码哥教育 SEEMYGO

字符流结构预览



小码哥教育 SEEMYGO

FileWriter

```
Writer writer = new FileWriter("F:/2.txt");
writer.write('M');
writer.write('J');
writer.write('码');
writer.write('哥');
writer.close();
```

```
Writer writer = new FileWriter("F:/2.txt");
writer.write("MJ");
writer.write("码哥".toCharArray());
writer.close();
```

小码哥教育 SEEMYGO

FileReader

```
Reader reader = new FileReader("F:/2.txt");
// 读取第1个字符
int c1 = reader.read();
// 读取第2个字符
int c2 = reader.read();
reader.close();
```

```
Reader reader = new FileReader("F:/2.txt");
char[] chars = new char[1024];
// read方法返回实际读取的字符数
int len = reader.read(chars);
reader.close();
```

小码哥教育 SEEMYGO

练习 – 将文本文件的内容逐个字符打印出来

```
File file = new File("F:\\Files\\Texts\\mj.txt");
try (
    Reader reader = new FileReader(file)
) {
    int c;
    while ((c = reader.read()) != -1) {
        System.out.print((char) c);
        Thread.sleep(10);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

- 这个需求不太合使用字节流

缓冲流（Buffered Streams）

- 之前学习的字节流、字符流，都是无缓冲的 I/O 流，每个读写操作均由底层操作系统直接处理
  - 每个读写操作通常会触发磁盘访问，因此大量的读写操作，可能会使程序的效率大大降低

- 为了减少读写操作带来的开销，Java 实现了缓冲的 I/O 流
  - 缓冲输入流：从缓冲区读取数据，并且只有当缓冲区为空时才调用本地的输入 API（Native Input API）
  - 缓冲输出流：将数据写入缓冲区，并且只有当缓冲区已满时才调用本地的输出 API（Native Output API）

	缓冲输入流	缓冲输出流
缓冲字节流	BufferedInputStream	BufferedOutputStream
缓冲字符流	BufferedReader	BufferedWriter

- 上述表格中 4 个缓冲流的默认缓冲区大小是 8192 字节（8KB），可以通过构造方法传参设置缓冲区大小

小码哥教育 SEEMYGO

缓冲流 – 使用

- 缓冲流的常见使用方式：将无缓冲流传递给缓冲流的构造方法（将无缓冲流包装成缓冲流）
- 如果把无缓冲流比作是一个无装备的士兵，那么缓冲流就是一个有强力装备的士兵

```
File file = new File("F:/mj.txt");
InputStream is = new FileInputStream(file);
BufferedInputStream bis = new BufferedInputStream(is, 16384);
bis.close();
```

```
File file = new File("F:/6.txt");
BufferedWriter writer = new BufferedWriter(new FileWriter(file));
writer.write("111");
writer.newLine();
writer.write("222");
writer.close();
```

缓冲流 – close、flush

- 只需要执行缓冲流的 close 方法，不需要执行缓冲流内部包装的无缓冲流的 close 方法

- 调用缓冲输出流的 flush 方法，会强制调用本地的输出 API，将缓冲区的数据真正写入到文件中
- 缓冲输出流的 close 方法内部会调用一次 flush 方法

小码哥教育 SEEMYGO

练习 – 用缓冲流修改 write

```
public static void write(byte[] data, File file) {
    if (data == null || file == null || file.exists()) return;
    mkparents(file);
    try (OutputStream os = new BufferedOutputStream(new FileOutputStream(file))) {
        os.write(data);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

小码哥教育 SEEMYGO

练习 – 用缓冲流修改 read

```
public static byte[] read(File file) {
    if (file == null || !file.exists()) return null;
    if (file.isDirectory()) return null;
    try (InputStream is = new BufferedInputStream(new FileInputStream(file))) {
        byte[] data = new byte[(int) file.length()];
        is.read(data);
        return data;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

小码哥教育 SEEMYGO

练习 – 用缓冲流修改 copy

```
public static void copy(File src, File dest) {
    if (src == null || dest == null || !src.exists() || dest.exists()) return;
    if (src.isDirectory()) return;
    mkparents(dest);
    try (
        InputStream is = new BufferedInputStream(new FileInputStream(src));
        OutputStream os = new BufferedOutputStream(new FileOutputStream(dest));
    ) {
        byte[] buffer = new byte[8192];
        int len;
        while ((len = is.read(buffer)) != EOF) {
            os.write(buffer, 0, len);
        }
    } catch (FileNotFoundException e) { e.printStackTrace(); }
    catch (IOException e) { e.printStackTrace(); }
}
```

小码哥教育 SEEMYGO

练习 – 用缓冲流逐个打印字符

```
File file = new File("F:\\Files\\Texts\\mj.txt");
try (Reader reader = new BufferedReader(new FileReader(file))) {
    int c;
    while ((c = reader.read()) != -1) {
        System.out.print((char) c);
        Thread.sleep(10);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

小码哥教育 SEEMYGO

练习 – 用缓冲流逐行打印字符串

```
File file = new File("F:\\Files\\Texts\\mj.txt");
try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
        Thread.sleep(100);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

小码哥教育 SEEMYGO

练习 – 转换文本文件编码

```
try (
    BufferedReader reader = new BufferedReader(
            new InputStreamReader(new FileInputStream("F:/gbk.txt"), "GBK"));
    BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(new FileOutputStream("F:/utf-8.txt"), "UTF-8"));
) {
    char[] chars = new char[1024];
    int len;
    while ((len = reader.read(chars)) != -1) {
        writer.write(chars, 0, len);
    }
}
```

小码哥教育 SEEMYGO

练习 – “价值几百万”的 AI 代码

```
// InputStream -> InputStreamReader -> BufferedReader
BufferedReader reader = new BufferedReader(
                        new InputStreamReader(System.in));
String str;
while ((str = reader.readLine()) != null) {
    str = str.replace("你", "朕");
    str = str.replace("吗", "");
    str = str.replace("么", "");
    str = str.replace("?", "!");
    str = str.replace("？", "!");
    System.out.println("\t" + str);
}
reader.close();
```

你吃饭了么？
    朕吃饭了！
你懂Java吗
    朕懂Java
真有那么厉害？
    真有那厉害！

- System.*in* 属于标准输入流，可以从键盘接收输入
- 利用 InputStreamReader 可以实现【字节输入流】转【字符输入流】
- 同理，利用 OutputStreamWriter 可以实现【字节输出流】转【字符输出流】

Scanner

- java.util.Scanner 是一个可以使用正则表达式来解析基本类型和字符串的简单文本扫描器
  - 它默认利用空白（空格\制表符\行终止符）作为分隔符将输入分隔成多个 token

```
Scanner(InputStream source)
Scanner(Readable source)
Scanner(File source)
Scanner(String source)
```

```
Scanner s = new Scanner("jack rose kate");
while (s.hasNext()) {
    System.out.println(s.next());
}
// jack
// rose
// kate
s.close();
```

```
Scanner s = new Scanner(new File("F:/mj.txt"));
```

小码哥教育 SEEMYGO

Scanner – next

```
Scanner s = new Scanner("jack 666 888 ak47");
System.out.println(s.next()); // jack
System.out.println(s.nextInt()); // 666
System.out.println(s.nextDouble()); // 888.0
System.out.println(s.next("[a-z]{2}\\d{2}")); // ak47
s.close();
```

小码哥教育 SEEMYGO

Scanner – useDelimiter

- Scanner.useDelimiter 方法可以自定义分隔符

```
Scanner s = new Scanner("aa 1 bb 22 cc33dd");
s.useDelimiter("\\s*\\d+\\s*");
while (s.hasNext()) {
    System.out.println(s.next());
}
// aa bb cc dd
s.close();
```

```
Scanner s = new Scanner("aa11bb22cc");
s.useDelimiter("");
while (s.hasNext()) {
    System.out.println(s.next());
} // a a 1 1 b b 2 2 c c
s.close();
```

小码哥教育 SEEMYGO

Scanner – 标准输入流

```
Scanner s = new Scanner(System.in);
System.out.print("请输入第1个整数: ");
int n1 = s.nextInt();
System.out.print("请输入第2个整数: ");
int n2 = s.nextInt();
System.out.format("%d + %d = %d%n", n1, n2, n1 + n2);
s.close();
/*
请输入第1个整数: 6
请输入第2个整数: 8
6 + 8 = 14
 */
```

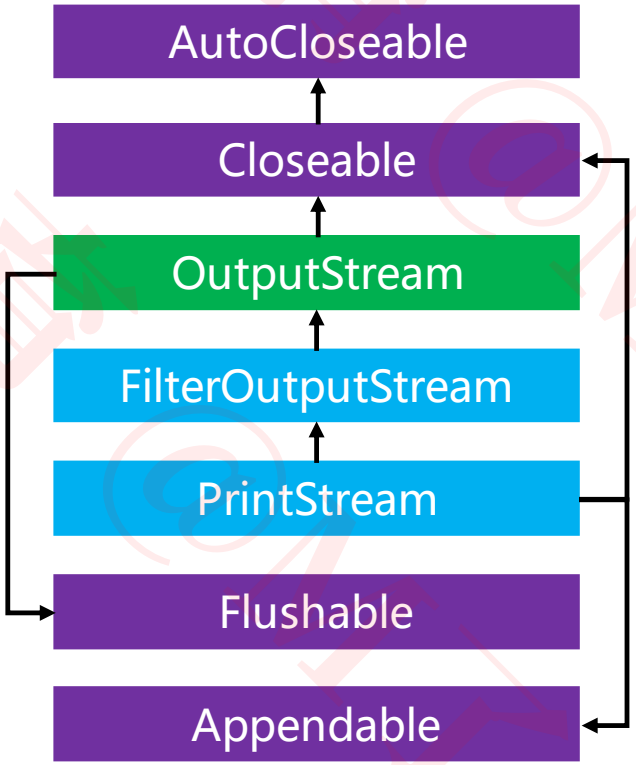
小码哥教育 SEEMYGO

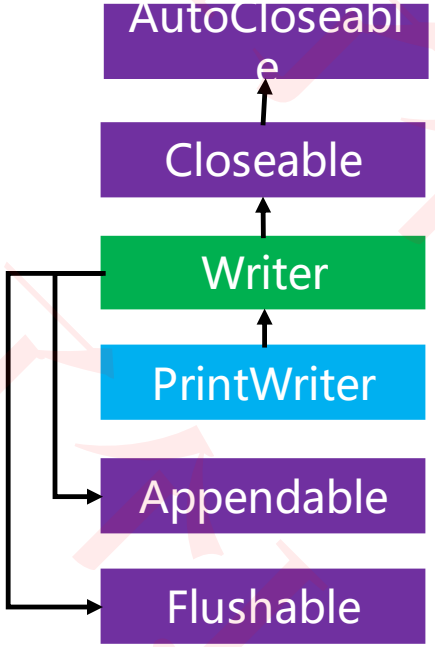
Scanner – “价值几百万”的 AI 代码

```
Scanner s = new Scanner(System.in);
while (s.hasNextLine()) {
    String str = s.nextLine();
    str = str.replace("你", "朕");
    str = str.replace("吗", "");
    str = str.replace("么", "");
    str = str.replace("?", "!");
    str = str.replace("？", "!");
    System.out.println("\t" + str);
}
s.close();
```

格式化输出

- 有 2 个类可以实现格式化输出
  - PrintStream、PrintWriter
- 它们有 3 个常用方法：print、println、format
- print、write 的区别
  - write(97) 写入的是字符 'a'
  - print(97) 写入的是字符串 "97"





PrintStream

- System.*out*、System.*err* 是 PrintStream 类型的实例
  - 属于标准输出流（Standard Ouput Stream）
  - 比如输出到屏幕、控制台（Console）

- PrintStream 是字节流，但它内部利用字符流对象来模拟字符流的许多功能

PrintWriter

- 平时若要创建格式化的输出流，一般使用 PrintWriter，它是字符流

```
String name = "Jack";
int age = 20;

PrintWriter writer = new PrintWriter("F:/1.txt");
writer.format("My name is %s, age is %d", name, age);
writer.close();
```

- 可以通过构造方法设置 PrintWriter.autoflush 为 true
  - 那么 println、printf、format 方法内部就会自动调用 flush 方法

```
PrintWriter writer = new PrintWriter(
                        new FileOutputStream(new File("F:/1.txt")), true);
```

数据流

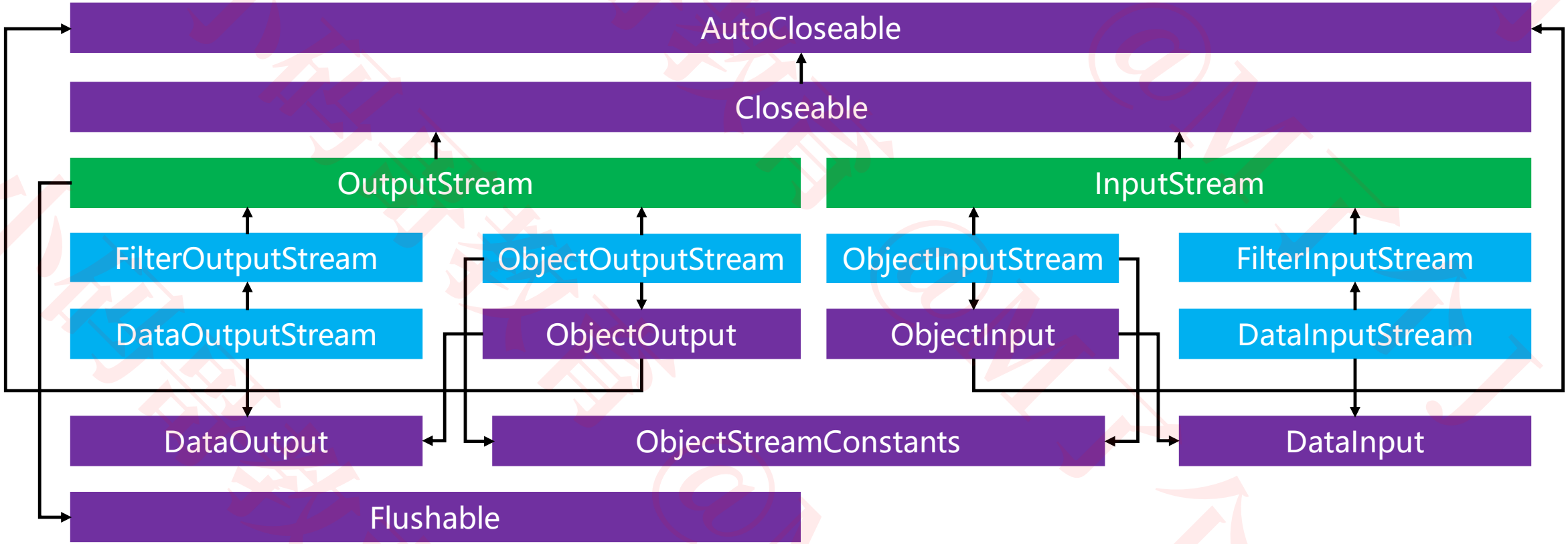
- 有 2 个数据流：DataInputStream、DataOutputStream，支持基本类型、字符串类型的 I / O 操作

```
int age = 20;    int money = 3000;    double height = 1.75;
String name = "Jack";
DataOutputStream dos = new DataOutputStream(new FileOutputStream("F:/66.txt"));
dos.writeInt(age);
dos.writeInt(money);
dos.writeDouble(height);
dos.writeUTF(name);
dos.close();
// 文件内容: 0000 0014 0000 0bb8 3ffc 0000 0000 0000 0004 4a61 636b
```

```
DataInputStream dis = new DataInputStream(new FileInputStream("F:/66.txt"));
System.out.println(dis.readInt()); // 20
System.out.println(dis.readInt()); // 3000
System.out.println(dis.readDouble()); // 1.75
System.out.println(dis.readUTF()); // Jack
dis.close();
```

小码哥教育 SEEMYGO

数据流、对象流结构预览



小码哥教育 SEEMYGO

对象流

- 有 2 个对象流：ObjectInputStream、ObjectOutputStream，支持引用类型的 I / O 操作

- 只有实现了 java.io.Serializable 接口的类才能使用对象流进行 I / O 操作
  - 否则会抛出 java.io.NotSerializableException 异常

- Serializable 是一个标记接口（Maker Interface），不要求实现任何方法

对象的序列化和反序列化

- 序列化（Serialization）
  - 将对象转换为可以存储或传输的数据
  - 利用 ObjectOutputStream 可以实现对象的序列化

- 反序列化（Deserialization ）
  - 从序列化后的数据中恢复出对象
  - 利用 ObjectInputStream 可以实现对象的反序列化

- 若将对象比作是一座冰雕
  - 序列化：将冰雕融化成水
  - 反序列化：将融化后的水恢复成冰雕

小码哥教育 SEEMYGO

Book

```
public class Book implements Serializable {
    private double price;
    private String name;
    public Book(double price, String name) {
        this.price = price;
        this.name = name;
    }
    @Override
    public String toString() {
        return "Book [price=" + price + ", name=" + name + "]";
    }
}
```

小码哥教育 SEEMYGO

Car

```
public class Car implements Serializable {
    private double price;
    private String band;
    public Car(double price, String band) {
        this.price = price;
        this.band = band;
    }
    @Override
    public String toString() {
        return "Car [price=" + price + ", band=" + band + "]";
    }
}
```

小码哥教育 SEEMYGO

Person

```
public class Person implements Serializable {
    private int age;
    private String name;
    private Car car;
    private List<Book> books = new ArrayList<>();
    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }
    public List<Book> getBooks() { return books; }
    public void setCar(Car car) { this.car = car; }
    @Override
    public String toString() {
        return "Person [age=" + age + ", name=" + name +
                ", car=" + car + ", books=" + books + "]";
    }
}
```

小码哥教育 SEEMYGO

ObjectOutputStream – 序列化

```
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("F:/p.txt"));

Person p = new Person(20, "Jack");
p.setCar(new Car(305.6, "Bently"));
p.getBooks().add(new Book(19.9, "Java"));
p.getBooks().add(new Book(38.8, "C++"));
oos.writeObject(p);

Car c = new Car(107.8, "BMW");
oos.writeObject(c);

oos.close();
```

小码哥教育 SEEMYGO

ObjectInputStream – 反序列化

```
ObjectInputStream ois = new ObjectInputStream(new FileInputStream("F:/p.txt"));
Person p = (Person) ois.readObject();
System.out.println(p);
/*
Person [
    age=20,
    name=Jack,
    car=Car [price=305.6, band=Bently],
    books=[
        Book [price=19.9, name=Java],
        Book [price=38.8, name=C++],
    ]
]
*/
Car c = (Car) ois.readObject();
System.out.println(c); // Car [price=107.8, band=BMW]
ois.close();
```

小码哥教育 SEEMYGO

transient

- transient：英[ˈtrænziənt]
- 被 transient 修饰的实例变量不会被序列化

```
public class Dog implements Serializable {
    private transient int age;
    private String name;
    public Dog(int age, String name) { this.age = age; this.name = name; }
    @Override
    public String toString() { return "Dog [age=" + age + ", name=" + name + "]"; }
}
```

```
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("F:/d.txt"));
oos.writeObject(new Dog(5, "Larry"));
oos.close();

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("F:/d.txt"));
System.out.println(ois.readObject()); // Dog [age=0, name=Larry]
ois.close();
```

小码哥教育 SEEMYGO

serialVersionUID

- 每一个可序列化类都有一个 `serialVersionUID`，相当于类的版本号
  - 默认情况下会根据类的详细信息计算出 `serialVersionUID` 的值，根据编译器实现的不同可能千差万别
  - 一旦类的信息发生修改，`serialVersionUID` 的值就会发生变化

- 如果序列化、反序列时的 `serialVersionUID` 不一致
  - 会认定为序列化、反序列时的类不兼容，会抛出 java.io.InvalidClassException 异常

- 强烈建议每一个可序列化类都自定义 `serialVersionUID`，不要使用它的默认值
  - 必须是 `static final long`
  - 建议声明为 `private`
  - 如果没有自定义 `serialVersionUID`，编译器会发出 `"serial"` 警告