

MENU



JAVA知识点 I/O流框架简要笔记

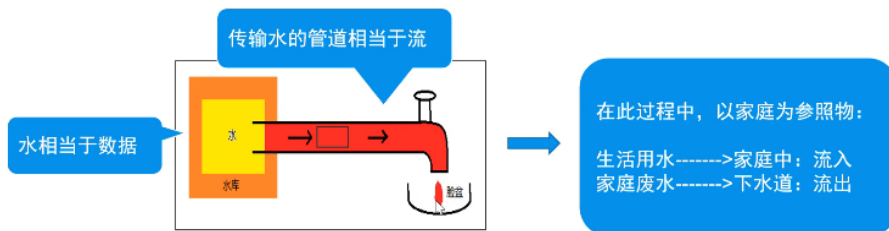
🕒 发表于 2020-09-14 23:13 📖 阅读次数: 2779 💬 评论次数: 0

JAVA JAVA

I/O 框架

🔗 流的概念

内存与存储设备之间传输数据的通道



🔗 流的分类

🔗 按方向【重点】

- 输入流：将<存储设备>中的内容读到<内存>中
- 输出流：将<内存>中的内容写到<存储设备>中

🔗 按单位

- 字节流：以字节为单位，可以读写所有数据
- 字符流：以字符为单位，只能读写文本数据



按功能

- 节点流：具有实际传输数据的读写功能
- 过滤流：在节点流的基础之上增强功能

字节流

字节流的父类（抽象类）

```
//InputStream 字节输入流
public int read(){}
public int read(byte[] b){}
public int read(byte[] b, int off, int len){}

// OutputStream 字节输出流
public void write(int n){}
public void write(byte[] b){}
public void write(byte[] b, int off, int len){}
```

文件字节流

文件输入流

```
psvm(String[] args) throws Exception{
    // 1 创建FileInputStream 并指定文件路径
    FileInputStream fis = new FileInputStream("d:\\abc.txt");
    // 2 读取文件
    // fis.read();
    // 2.1单字节读取
    int data = 0;
    while((data = fis.read()) != -1){
        sout((char)data);
    }
    // 2.2 一次读取多个字节
    byte[] buf = new byte[3]; // 大小为3的缓存区
    int count = fis.read(buf); // 一次读3个
    sout(new String(buf));
    sout(count);
    int count2 = fis.read(buf); // 再读3个
    sout(new String(buf));
    sout(count2);

    // 上述优化后
    int count = 0;
    while((count = fis.read(buf)) != -1){
```



```
sout(new String(buf, 0, count));
}

// 3 关闭
fis.close();
}
```

文件输出流

```
psvm(String[] args) throws Exception{
    // 1 创建文件字节输出流
    FileOutputStream fos = new FileOutputStream("路径", true);
    // 2 写入文件
    fos.write(97);
    fos.write('a');
    // String string = "hello world";
    fos.write(string.getBytes());
    // 3 关闭
    fos.close();
}
```

🔗 图片复制案例

```
// 1 创建流
// 1.1 文件字节输入流
FileInputStream fis = new FileInputStream("路径");
// 1.2 文件字节输出流
FileOutputStream fos = new FileOutputStream("路径");
// 2 边读边写
byte[] buf = new byte[1024];
int count = 0;
while((count = fis.read(buf)) != -1){
    fos.write(buf, 0, count);
}
// 3 关闭
fis.close();
fos.close();
```

🔗 字节缓冲流

缓冲流: BufferedInputStream/ BufferedOutputStream

- 提高IO效率, 减少访问磁盘次数



- 数据存储在缓冲区中，flush是将缓冲区的内容写入文件中，也可以直接close

```
// 使用字节缓冲流 读取 文件
psvm(String[] args) throws Exception{
    // 1 创建BufferedInputStream
    FileInputStream fis = new FileInputStream("路径");
    BufferedInputStream bis = new BufferedInputStream(fis);
    // 2 读取
    int data = 0;
    while((data = bis.read()) != -1){
        sout((char)data);
    }
    // 用自己创建的缓冲流
    byte[] buf = new byte[1024];
    int count = 0;
    while((count = bis.read(buf)) != -1){
        sout(new String(buf, 0, count));
    }

    // 3 关闭
    bis.close();
}

// 使用字节缓冲流 写入 文件
psvm(String[] args) throws Exception{
    // 1 创建BufferedInputStream
    FileOutputStream fos = new FileOutputStream("路径");
    BufferedOutputStream bis = new BufferedOutputStream(fos);
    // 2 写入文件
    for(int i = 0; i < 10; i++){
        bos.write("hello".getBytes()); // 写入8k缓冲区
        bos.flush(); // 刷新到硬盘
    }
    // 3 关闭
    bos.close();
}
```

对象流

ObjectOutputStream / ObjectInputStream

- 增强了缓冲区功能
- 增强了读写8种基本数据类型和字符串的功能



- 增强了读写对象的功能
 - `readObject()` 从流中读取一个对象
 - `writeObject(Object obj)` 向流中写入一个对象

使用流传输对象的过程称为序列化、反序列化

序列化与反序列化

序列化

```
// 使用ObjectOutputStream实现序列化
psvm(String[] args){
    // 1. 创建对象流
    FileOutputStream fos = new FileOutputStream("d:\\
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    // 2. 序列化（写入操作）
    Student zhangsan = new Student("zs", 20);
    oos.writeObject(zhangsan);
    // 3. 关闭
    oos.close();
    sout("序列化完毕");
}
```

反序列化

```
// 使用ObjectInputStream实现反序列化（读取重构对象）
psvm(String[] args){
    // 1. 创建对象流
    FileInputStream fis = new FileInputStream("d:\\stu.bin");
    ObjectInputStream ois = new ObjectInputStream(fis);
    // 2. 读取文件（反序列化）
    Student s = (Student)ois.readObject();
    // 3. 关闭
    ois.close();
    sout("执行完毕");
    sout(s.toString());
}
```

注意事项

1. 某个类要想序列化必须实现Serializable接口
2. 序列化类中对象属性要求实现Serializable接口
3. 序列化版本号ID，保证序列化的类和反序列化的类是同一个类



4. 使用transient修饰属性，这个属性就不能序列化
5. 静态属性不能序列化
6. 序列化多个对象，可以借助集合来实现

Ø 编码方式

UTF-8... 不赘述

Ø 字符流

```
// 传统字节流读取
psvm(String[] args){
    // 1. 创建FileInputStream对象
    FileInputStream fis = new FileInputStream("路径");
    // 2. 读取
    int data = 0;
    while((data = fis.read()) != -1){
        sout((char)data);
    }
    // 3. 关闭
    fis.close();
}
```

Ø 字符流的父类（抽象类）

Reader 字符输入流

- `public int read(){}`
- `public int read(char[] c){}`
- `public int read(char[] b, int off, int len){}`

Writer 字符输出流

- `public void write(int n){}`
- `public void write(String str){}`
- `public void write(char[] c){}`

```
// 1. 创建FileReader 文件字符输入流
FileReader fr = new FileReader("..");
// 2. 读取
// 2.1 单个字符读取
int data = 0;
```



```
while((data = fr.read()) != -1){
    sout((char)data); // 读取一个字符
}
char[] buf = new char[2]; // 字符缓冲区读取
int count = 0;
while((count = fr.read(buf) != -1)){
    sout(new String(buf, 0, count));
}
// 3. 关闭
fr.close();

// 1. 创建FileWriter对象
FileWriter fw = new FileWriter("..");
// 2. 写入
for(int i = 0; i < 10; i++){
    fw.write("写入的内容");
    fw.flush();
}
// 3. 关闭
fw.close();
sout("执行完毕");
```

🔪 (案例)使用上述内容进行文本文件复制

不能复制图片或二进制文件，使用字节流可以复制任意文件

```
psvm(String[] args) throws Exception{
    // 1. 创建
    FileReader fr = new FileReader("...");
    FileWriter fw = new FileWriter("...");
    // 2. 读写
    int data = 0;
    while((data = fr.read()) != -1){
        fw.write(data);
        fw.flush();
    }
    // 3. 关闭
    fw.close();
    fr.close();
}
```

🔪 字符缓冲流

BufferedReader / BufferedWriter



高效读写、支持输入换行符、可一次写一行读一行

```
psvm(String[] args) throws Exception{
    // 创建缓冲流
    FileReader fr = new FileReader("..");
    BufferedReader br = new BufferedReader(fr);
    // 读取
    // 1. 第一种方式
    char[] buf = new char[1024];
    int count = 0;
    while((count = br.read(buf)) != -1){
        sout(new String(buf, 0, count));
    }
    // 2. 第二种方式 一行一行读取
    String line = null;
    while((line = br.readLine()) != null){
        sout(line);
    }

    // 关闭
    br.close();
}

psvm(String[] args){
    // 1. 创建BufferedWriter对象
    FileWriter fw = new FileWriter("..");
    BufferedWriter bw = new BufferedWriter(fw);
    // 2. 写入
    for(int i = 0; i < 10; i++){
        bw.write("写入的内容");
        bw.newLine(); // 写入一个换行符
        bw.flush();
    }
    // 3. 关闭
    bw.close(); // 此时会自动关闭fw
}
```

7.4 PrintWriter

封装了 `print()` / `println()` 方法 支持写入后换行

支持数据原样打印

```
psvm(String[] args){
    // 1 创建打印流
```




```
PrintWriter pw = new PrintWriter("..");
// 2 打印
pw.println(12);
pw.println(true);
pw.println(3.14);
pw.println('a');
// 3 关闭
pw.close();
}
```

转换流

桥转换流 `InputStreamReader` / `OutputStreamWriter`

可将字节流转换为字符流

可设置字符的编码方式

```
psvm(String[] args) throws Exception{
    // 1 创建InputStreamReader对象
    FileInputStream fis = new FileInputStream("..");
    InputStreamReader isr = new InputStreamReader(fis, "utf-8");
    // 2 读取文件
    int data = 0;
    while((data = isr.read()) != -1){
        sout((char)data);
    }
    // 3 关闭
    isr.close();
}
```

```
psvm(String[] args) throws Exception{
    // 1 创建OutputStreamWriter对象
    FileOutputStream fos = new FileOutputStream("..");
    OutputStreamWriter osw = new OutputStreamWriter(fos, "utf-8")
    // 2 写入
    for(int i = 0; i < 10; i ++){
        osw.write("写入内容");
        osw.flush();
    }
    // 3 关闭
    osw.close();
}
```

File类



概念：代表物理盘符中的一个文件或者文件夹

```

/*
File类的使用
1. 分隔符
2. 文件操作
3. 文件夹操作
*/
public class Demo{
    psvm(String[] args){
        separator();
    }
    // 1. 分隔符
    public static void separator(){
        sout("路径分隔符" + File.pathSeparator);
        sout("名称分隔符" + File.separator);
    }
    // 2. 文件操作
    public static void fileOpen(){
        // 1. 创建文件
        if(!file.exists()){ // 是否存在
            File file = new File("...");
            boolean b = file.createNewFile();
        }

        // 2. 删除文件
        // 2.1 直接删除
        file.delete(); // 成功true
        // 2.2 使用jvm退出时删除
        file.deleteOnExit();

        // 3. 获取文件信息
        sout("获取绝对路径" + file.getAbsolutePath());
        sout("获取路径" + file.getPath());
    }
}

```

递归遍历文件夹

```

psvm(String[] args){
    listDir(new File("d:\\myfiles"));
}
public static void listDir(File dir){
    File[] files = dir.listFiles();
    sout(dir.getAbsolutePath());
    if(files != null && files.length > 0){
        for(File file : files){
            if(file.isDirectory()){

```



MENU

```
        listDir(file); // 递归
    }else {
        sout(file.getAbsolutePath());
    }
}
}
```

递归删除文件夹

```
public static void deleteDir(File dir){
    File[] files = dir.listFiles();
    if(files != null && files.length > 0){
        for(File file : files){
            if(file.isDirectory()){
                deleteDir(file); // 递归
            }else{
                // 删除文件
                sout(file.getAbsolutePath() + "删除" + file.delete());
            }
        }
    }
}
```

__EOF__



本文作者: Coder Zjz's blog

本文链接:

<https://www.cnblogs.com/coderzjz/p/13670088.html>

关于博主: 评论和私信会在第一时间回复。或者直接私信我。

版权声明: 本博客所有文章除特别声明外, 均采用 BY-NC-SA 许可协议。转载请注明出处!

声援博主: 如果您觉得文章对您有帮助, 可以点击文章右下角

【推荐】一下。您的鼓励是博主的最大动力!

分类: java

标签: java

好文要顶

关注我

收藏该文



13

0



MENU



CoderZjz

关注 - 3

粉丝 - 69

+加关注

« 上一篇: [JAVA集合类简要笔记 - 内部类 包装类 Object类 String类 BigDecimal类 system类](#)» 下一篇: [VMware虚拟机ubuntu下安装VMware Tools步骤](#)

posted @ 2020-09-14 23:13 CoderZjz 阅读(2779) 评论(0) 编辑 收藏 举报

登录后才能查看或发表评论 · 立即 [登录](#) 或者 [逛逛](#) 博客园首页

编辑推荐：

- 带团队后的日常思考 (五)
- 聊聊我在微软外服的工作经历及一些个人见解
- 死磕 NIO — Reactor 模式就一定意味着高性能吗？
- 消息队列那么多，为什么建议深入了解下RabbitMQ？
- 技术管理进阶——管人还是管事？

最新新闻：

- 集成3400 条commit ! PyTorch 1.10 正式版发布，能帮你选batch size的框架 (2021-10-26 20:00)
- 超越FB！特斯拉市值破万亿美元，马斯克身价近3000亿美元稳坐全球首富 (2021-10-26 18:45)
- Google将联网电视业务视为首要任务 死磕亚马逊和Roku (2021-10-26 18:30)
- 人造肌肉机械臂太逼真引热议，举7kg哑铃“肌肉”清晰可见 (2021-10-26 18:17)
- 育碧：游戏出现性能/崩溃问题 或是后台软件冲突导致 (2021-10-26 17:17)
- » 更多新闻...

「 凡是过往  皆为序章 」This blog has running : 568 d 14 h 17 m 17 s ) / ♡友情链接：[Github](#) / [CSDN博客](#) / [申请坑位](#) / [申请坑位](#) / [申请坑位](#)

Copyright © 2021 CoderZjz Powered by .NET 6 on Kubernetes

Theme version: [v1.3.0](#) / Loading theme version: [v1.3.0](#)