day21【字符流、 Properties】

今日内容

- 字符流
- Properties

教学目标

- [ ] 能够使用FileWriter写数据的5个方法
- [ ] 能够说出FileWriter中关闭和刷新方法的区别
- [ ] 能够使用FileWriter写数据实现换行和追加写
- [ ] 能够使用FileReader读数据一次一个字符
- [ ] 能够使用FileReader读数据一次一个字符数组
- [ ] 能够使用Properties的load方法加载文件中配置信息

第一章 字符流

当使用字节流读取文本文件时，可能会有一个小问题。就是遇到中文字符时，可能不会显示完整的字符，那是因为一个中文字符可能占用多个字节存储。所以Java提供一些字符流类，以字符为单位读写数据，专门用于处理文本文件。

1.1 字符输入流【Reader】

`java.io.Reader`抽象类是表示用于读取字符流的所有类的超类，可以读取字符信息到内存中。它定义了字符输入流的基本共性功能方法。

- `public void close()` ：关闭此流并释放与此流相关联的任何系统资源。
- `public int read()`： 从输入流读取一个字符。
- `public int read(char[] cbuf)`： 从输入流中读取一些字符，并将它们存储到字符数组 cbuf 中 。

1.2 FileReader类

`java.io.FileReader`类是读取字符文件的便利类。构造时使用系统默认的字符编码和默认字节缓冲区。

> 小贴士：
>
> 1. 字符编码：字节与字符的对应规则。Windows系统的中文编码默认是GBK编码表。
>
> idea中UTF-8
>
> 2. 字节缓冲区：一个字节数组，用来临时存储字节数据。

构造方法

- `FileReader(File file)`： 创建一个新的 FileReader ，给定要读取的File对象。
- `FileReader(String fileName)`： 创建一个新的 FileReader ，给定要读取的文件的名称。

当你创建一个流对象时，必须传入一个文件路径。类似于FileInputStream 。

- 构造举例，代码如下：

```
public class FileReaderConstructor throws IOException{
    public static void main(String[] args) {
        // 使用File对象创建流对象
        File file = new File("a.txt");
        FileReader fr = new FileReader(file);

        // 使用文件名称创建流对象
        FileReader fr = new FileReader("b.txt");
    }
}
```

读取字符数据

1. **读取字符**：`read`方法，每次可以读取一个字符的数据，提升为int类型，读取到文件末尾，返回`-1`，循环读取，代码使用演示：

```
public class FRRead {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileReader fr = new FileReader("read.txt");
        // 定义变量，保存数据
        int b ;
        // 循环读取
        while ((b = fr.read())!=-1) {
            System.out.println((char)b);
        }
        // 关闭资源
        fr.close();
    }
}
输出结果：
黑
马
程
序
员
```

> 小贴士：虽然读取了一个字符，但是会自动提升为int类型。

1. **使用字符数组读取**：`read(char[] cbuf)`，每次读取多个字符到数组中，返回读取到的有效字符个数，读取到末尾时，返回`-1`，代码使用演示：

```
public class FRRead {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileReader fr = new FileReader("read.txt");
        // 定义变量，保存有效字符个数
        int len ;
        // 定义字符数组，作为装字符数据的容器
```

```
            char[] cbuf = new char[2];
            // 循环读取
            while ((len = fr.read(cbuf))!=-1) {
                System.out.println(new String(cbuf));
            }
            // 关闭资源
            fr.close();
        }
}
输出结果：
黑马
程序
员序
```

获取有效的字符改进，代码使用演示：

```
public class FISRead {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileReader fr = new FileReader("read.txt");
        // 定义变量，保存有效字符个数
        int len ;
        // 定义字符数组，作为装字符数据的容器
        char[] cbuf = new char[2];
        // 循环读取
        while ((len = fr.read(cbuf))!=-1) {
            System.out.println(new String(cbuf,0,len));
        }
        // 关闭资源
        fr.close();
    }
}

输出结果：
黑马
程序
员
```

1.3 字符输出流【Writer】

`java.io.Writer`抽象类是表示用于写出字符流的所有类的超类，将指定的字符信息写出到目的地。它定义了字节输出流的基本共性功能方法。

- `public abstract void close()` ： 关闭此输出流并释放与此流相关联的任何系统资源。
- `public abstract void flush()` ： 刷新此输出流并强制任何缓冲的输出字符被写出。
- `public void write(int c)` ： 写出一个字符。
- `public void write(char[] cbuf)`： 将 b.length字符从指定的字符数组写出此输出流。
- `public abstract void write(char[] b, int off, int len)` ： 从指定的字符数组写出 len 字符，从偏移量 off开始输出到此输出流。
- `public void write(String str)` ： 写出一个字符串。

1.4 FileWriter类

`java.io.FileWriter`类是写出字符到文件的便利类。构造时使用系统默认的字符编码和默认字节缓冲区。

构造方法

- `FileWriter(File file)`： 创建一个新的 FileWriter，给定要读取的File对象。
- `FileWriter(String fileName)`： 创建一个新的 FileWriter，给定要读取的文件的名称。

当你创建一个流对象时，必须传入一个文件路径，类似于FileOutputStream。

- 构造举例，代码如下：

```
public class FileWriterConstructor {
    public static void main(String[] args) throws IOException {
        // 使用File对象创建流对象
        File file = new File("a.txt");
        FileWriter fw = new FileWriter(file);

        // 使用文件名称创建流对象
        FileWriter fw = new FileWriter("b.txt");
    }
}
```

基本写出数据

写出字符：`write(int b)`方法，每次可以写出一个字符数据，代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileWriter fw = new FileWriter("fw.txt");
        // 写出数据
        fw.write(97); // 写出第1个字符
        fw.write('b'); // 写出第2个字符
        fw.write('C'); // 写出第3个字符
        fw.write(30000); // 写出第4个字符，中文编码表中30000对应一个汉字。

        /*
        【注意】关闭资源时,与FileOutputStream不同。
         如果不关闭,数据只是保存到缓冲区，并未保存到文件。
        */
        // fw.close();
    }
}
```

输出结果：

```
abC田
```

> 小贴士：
>
> 1. 虽然参数为int类型四个字节，但是只会保留一个字符的信息写出。
> 2. 未调用close方法，数据只是保存到了缓冲区，并未写出到文件中。

关闭和刷新

因为内置缓冲区的原因，如果不关闭输出流，无法写出字符到文件中。但是关闭的流对象，是无法继续写出数据的。如果我们既想写出数据，又想继续使用流，就需要 `flush` 方法了。

- `flush` ：刷新缓冲区，流对象可以继续使用。
- `close` ：关闭流，释放系统资源。关闭前会刷新缓冲区。

代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileWriter fw = new FileWriter("fw.txt");
        // 写出数据，通过flush
        fw.write('刷'); // 写出第1个字符
        fw.flush();
        fw.write('新'); // 继续写出第2个字符，写出成功
        fw.flush();

        // 写出数据，通过close
        fw.write('关'); // 写出第1个字符
        fw.close();
        fw.write('闭'); // 继续写出第2个字符,【报错】java.io.IOException: Stream
closed
        fw.close();
    }
}
```

> 小贴士：即便是flush方法写出了数据，操作的最后还是要调用close方法，释放系统资源。

写出其他数据

1. **写出字符数组** ： `write(char[] cbuf)` 和 `write(char[] cbuf, int off, int len)` ，每次可以写出字符数组中的数据，用法类似FileOutputStream，代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileWriter fw = new FileWriter("fw.txt");
        // 字符串转换为字节数组
        char[] chars = "黑马程序员".toCharArray();

        // 写出字符数组
        fw.write(chars); // 黑马程序员

        // 写出从索引2开始，2个字节。索引2是'程'，两个字节，也就是'程序'。
        fw.write(b,2,2); // 程序

        // 关闭资源
        fos.close();
    }
}
```

1. **写出字符串**： `write(String str)` 和 `write(String str, int off, int len)` ，每次可以写出字符串中的数据，更为方便，代码使用演示：

```
public class FWWrite {
```

```
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileWriter fw = new FileWriter("fw.txt");
        // 字符串
        String msg = "黑马程序员";

        // 写出字符数组
        fw.write(msg); //黑马程序员

        // 写出从索引2开始，2个字节。索引2是'程'，两个字节，也就是'程序'。
        fw.write(msg,2,2);  // 程序

        // 关闭资源
        fos.close();
    }
}
```

1. **续写和换行**：操作类似于FileOutputStream。

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象，可以续写数据
        FileWriter fw = new FileWriter("fw.txt", true);
        // 写出字符串
        fw.write("黑马");
        // 写出换行
        fw.write("\r\n");
        // 写出字符串
        fw.write("程序员");
        // 关闭资源
        fw.close();
    }
}
输出结果:
黑马
程序员
```

> 小贴士：字符流，只能操作文本文件，不能操作图片，视频等非文本文件。
>
> 当我们单纯读或者写文本文件时 使用字符流 其他情况使用字节流

第二章 IO资源的处理

2.1 JDK7前处理

之前的入门练习，我们一直把异常抛出，而实际开发中并不能这样处理，建议使用`try...catch...finally`代码块，处理异常部分，代码使用演示：

```
public class HandleException1 {
    public static void main(String[] args) {
        // 声明变量
        FileWriter fw = null;
        try {
```

```
            //创建流对象
            fw = new FileWriter("fw.txt");
            // 写出数据
            fw.write("黑马程序员"); //黑马程序员
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fw != null) {
                    fw.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2.2 JDK7的处理

还可以使用JDK7优化后的`try-with-resource`语句，该语句确保了每个资源在语句结束时关闭。所谓的资源（resource）是指在程序完成后，必须关闭的对象。

格式：

```
try (创建流对象语句，如果多个,使用';'隔开) {
    // 读写数据
} catch (IOException e) {
    e.printStackTrace();
}
```

代码使用演示：

```
public class HandleException2 {
    public static void main(String[] args) {
        // 创建流对象
        try ( FileWriter fw = new FileWriter("fw.txt"); ) {
            // 写出数据
            fw.write("黑马程序员"); //黑马程序员
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

第三章 属性集

3.1 概述

`java.util.Properties` 继承于`Hashtable`，来表示一个持久的属性集。它使用键值结构存储数据，每个键及其对应值都是一个字符串。该类也被许多Java类使用，比如获取系统属性时，`System.getProperties` 方法就是返回一个 `Properties` 对象。

3.2 Properties类

构造方法

- `public Properties()` :创建一个空的属性列表。

基本的存储方法

- `public Object setProperty(String key, String value)` ： 保存一对属性。
- `public String getProperty(String key)` ：使用此属性列表中指定的键搜索属性值。
- `public Set<String> stringPropertyNames()` ：所有键的名称的集合。

```
public class ProDemo {
    public static void main(String[] args) throws FileNotFoundException {
        // 创建属性集对象
        Properties properties = new Properties();
        // 添加键值对元素
        properties.setProperty("filename", "a.txt");
        properties.setProperty("length", "209385038");
        properties.setProperty("location", "D:\\a.txt");
        // 打印属性集对象
        System.out.println(properties);
        // 通过键,获取属性值
        System.out.println(properties.getProperty("filename"));
        System.out.println(properties.getProperty("length"));
        System.out.println(properties.getProperty("location"));

        // 遍历属性集,获取所有键的集合
        Set<String> strings = properties.stringPropertyNames();
        // 打印键值对
        for (String key : strings ) {
            System.out.println(key+" -- "+properties.getProperty(key));
        }
    }
}
输出结果:
{filename=a.txt, length=209385038, location=D:\a.txt}
a.txt
209385038
D:\a.txt
filename -- a.txt
length -- 209385038
location -- D:\a.txt
```

与流相关的方法

- `public void load(InputStream inStream)`： 从字节输入流中读取键值对。

参数中使用了字节输入流，通过流对象，可以关联到某文件上，这样就能够加载文本中的数据了。文本数据格式:

```
filename=a.txt
length=209385038
location=D:\a.txt
```

加载代码演示：

```
public class ProDemo2 {
    public static void main(String[] args) throws FileNotFoundException {
        // 创建属性集对象
        Properties pro = new Properties();
        // 加载文本中信息到属性集
        pro.load(new FileInputStream("read.txt"));
        // 遍历集合并打印
        Set<String> strings = pro.stringPropertyNames();
        for (String key : strings ) {
            System.out.println(key+" -- "+pro.getProperty(key));
        }
     }
}
输出结果：
filename -- a.txt
length -- 209385038
location -- D:\a.txt
```

> 小贴士：文本中的数据，必须是键值对形式，可以使用空格、等号、冒号等符号分隔。

第四章 ResourceBundle工具类

前面我们学习了Properties工具类，它能够读取资源文件，当资源文件是以.properties结尾的文件时，我们可以使用JDK提供的另外一个工具类ResourceBundle来对文件进行读取，使得操作更加简单。

4.1. ResourceBundle类的介绍

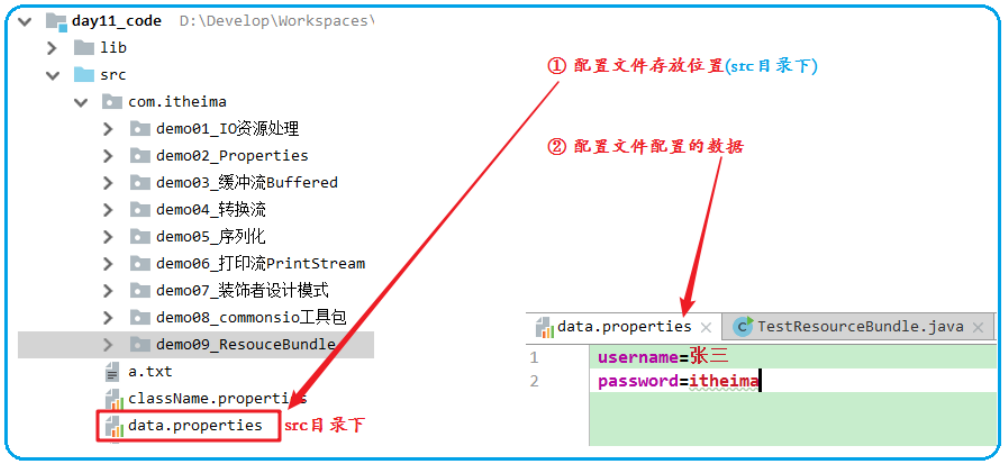
java.util.ResourceBundle它是一个抽象类，我们可以使用它的子类PropertyResourceBundle来读取以.properties结尾的配置文件。

4.2. ResourceBundle类对象的创建

在ResourceBundle类中提供了一个静态方法，用于获得它的子类对象(抽象类不能创建对象！)。

```
// 使用指定的基本名称，默认语言环境和调用者的类加载器获取资源包。
static ResourceBundle getBundle(String baseName);
```

> 注意：
>
> ①properties配置文件需要放置在类的根路径src下面
>
> ②给定参数只需要配置文件的名称，不要扩展名。



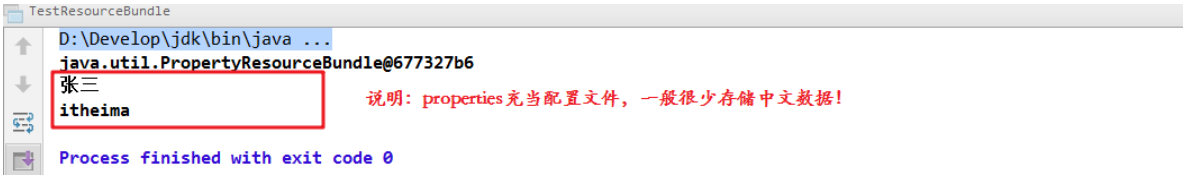

```
public class TestResourceBundle {
    public static void main(String[] args) {
        // 获得ResourceBundle类的对象(properties文件要放置在src目录下，给定参数只要文件名称，不要扩展名！)
        ResourceBundle bundle = ResourceBundle.getBundle("data");// 父类引用指向子类对象(多态)
        System.out.println(bundle);
    }
}
```

测试结果：

4.3. ResourceBundle读取配置文件操作

ResourceBundle类提供了一个getString(String key)方法用于读取配置文件中指定key的值。

```
String getString(String key) 从此资源束或其父项之一获取给定密钥的字符串。
```



代码演示(数据在上图中)：

```
![03_ResourceBundle读取配置文件效果](img/03_ResourceBundle读取配置文件效果.png)public class TestResourceBundle {
    public static void main(String[] args) {
        // 获得ResourceBundle类的对象(properties文件要放置在src目录下，给定参数只要文件名称，不要扩展名)
        ResourceBundle bundle = ResourceBundle.getBundle("data");
        System.out.println(bundle);

        // 读取指定key(username)的值
        String username = bundle.getString("username");
```

```
        System.out.println(username); // 张三

        // 读取指定key(password)的值
        String password = bundle.getString("password");
        System.out.println(password); // itheima
    }
}
```

运行结果：