Java IO - 常见类使用

主要介绍Java IO常见类的使用，包括：磁盘操作，字节操作，字符操作，对象操作和网络操作。

IO常见类的使用

Java 的 I/O 大概可以分成以下几类:

- 磁盘操作: File
- 字节操作: InputStream 和 OutputStream
- 字符操作: Reader 和 Writer
- 对象操作: Serializable
- 网络操作: Socket

*File*相关

File 类可以用于表示文件和目录的信息，但是它不表示文件的内容。

递归地列出一个目录下所有文件:

```
public static void listAllFiles(File dir) {
    if (dir == null || !dir.exists()) {
        return;
    }
    if (dir.isFile()) {
        System.out.println(dir.getName());
        return;
    }
    for (File file : dir.listFiles()) {
        listAllFiles(file);
    }
}
```

字节流相关

```
public static void copyFile(String src, String dist) throws IOException {

    FileInputStream in = new FileInputStream(src);
    FileOutputStream out = new FileOutputStream(dist);
    byte[] buffer = new byte[20 * 1024];

    // read() 最多读取 buffer.length 个字节
```

```
    // 返回的是实际读取的个数
    // 返回 -1 的时候表示读到 eof，即文件尾
    while (in.read(buffer, 0, buffer.length) != -1) {
        out.write(buffer);
    }

    in.close();
    out.close();
}
```

实现逐行输出文本文件的内容

```
public static void readFileContent(String filePath) throws IOException {

    FileReader fileReader = new FileReader(filePath);
    BufferedReader bufferedReader = new BufferedReader(fileReader);

    String line;
    while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
    }

    // 装饰者模式使得 BufferedReader 组合了一个 Reader 对象
    // 在调用 BufferedReader 的 close() 方法时会去调用 Reader 的 close() 方法
    // 因此只要一个 close() 调用即可
    bufferedReader.close();
}
```

序列化 & *Serializable* & *transient*

序列化就是将一个对象转换成字节序列，方便存储和传输。

- 序列化: ObjectOutputStream.writeObject()
- 反序列化: ObjectInputStream.readObject()

不会对静态变量进行序列化，因为序列化只是保存对象的状态，静态变量属于类的状态。

Serializable

序列化的类需要实现 Serializable 接口，它只是一个标准，没有任何方法需要实现，但是如果不去实现它的话而进行序列化，会抛出异常。

```
public static void main(String[] args) throws IOException, ClassNotFoundException {
    A a1 = new A(123, "abc");
    String objectFile = "file/a1";
    ObjectOutputStream objectOutputStream = new ObjectOutputStream(new
FileOutputStream(objectFile));
    objectOutputStream.writeObject(a1);
    objectOutputStream.close();

    ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream(objectFile));
    A a2 = (A) objectInputStream.readObject();
```

```
    objectInputStream.close();
    System.out.println(a2);
}

private static class A implements Serializable {
    private int x;
    private String y;

    A(int x, String y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return "x = " + x + "  " + "y = " + y;
    }
}
```

transient

transient 关键字可以使一些属性不会被序列化。

ArrayList 中存储数据的数组 elementData 是用 transient 修饰的，因为这个数组是动态扩展的，并不是所有的空间都被使用，因此就不需要所有的内容都被序列化。通过重写序列化和反序列化方法，使得可以只序列化数组中有内容的那部分数据。

```
private transient Object[] elementData;
```

Java 中的网络支持:

- InetAddress: 用于表示网络上的硬件资源，即 IP 地址；
- URL: 统一资源定位符；
- Sockets: 使用 TCP 协议实现网络通信；
- Datagram: 使用 UDP 协议实现网络通信。

InetAddress

没有公有的构造函数，只能通过静态方法来创建实例。

```
InetAddress.getByName(String host);
InetAddress.getByAddress(byte[] address);
```

URL

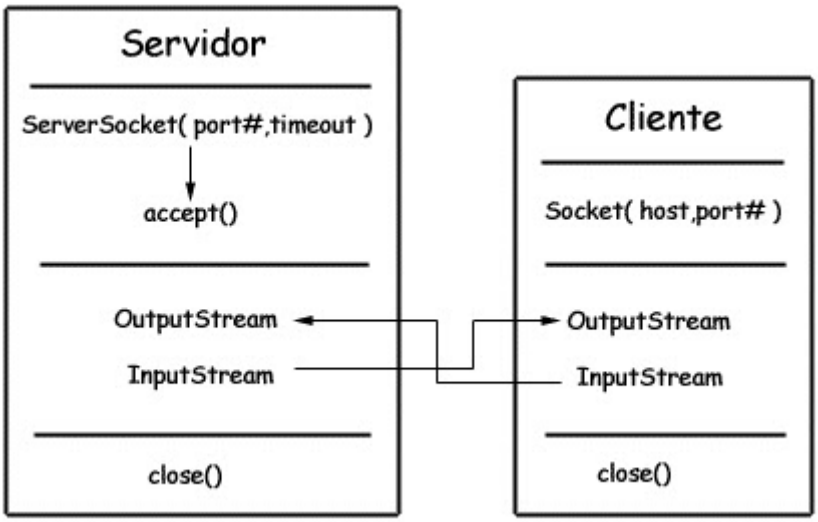
可以直接从 URL 中读取字节流数据。

```
public static void main(String[] args) throws IOException {

    URL url = new URL("http://www.baidu.com");

    /* 字节流 */
    InputStream is = url.openStream();
```

```
    /* 字符流 */
    InputStreamReader isr = new InputStreamReader(is, "utf-8");

    /* 提供缓存功能 */
    BufferedReader br = new BufferedReader(isr);

    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }

    br.close();
}
```

Sockets

- ServerSocket: 服务器端类
- Socket: 客户端类
- 服务器和客户端通过 InputStream 和 OutputStream 进行输入输出。



Datagram

- DatagramSocket: 通信类
- DatagramPacket: 数据包类

常见问题

- Java 字节读取流的read方法返回int的原因

https://blog.csdn.net/congwiny/article/details/18922847