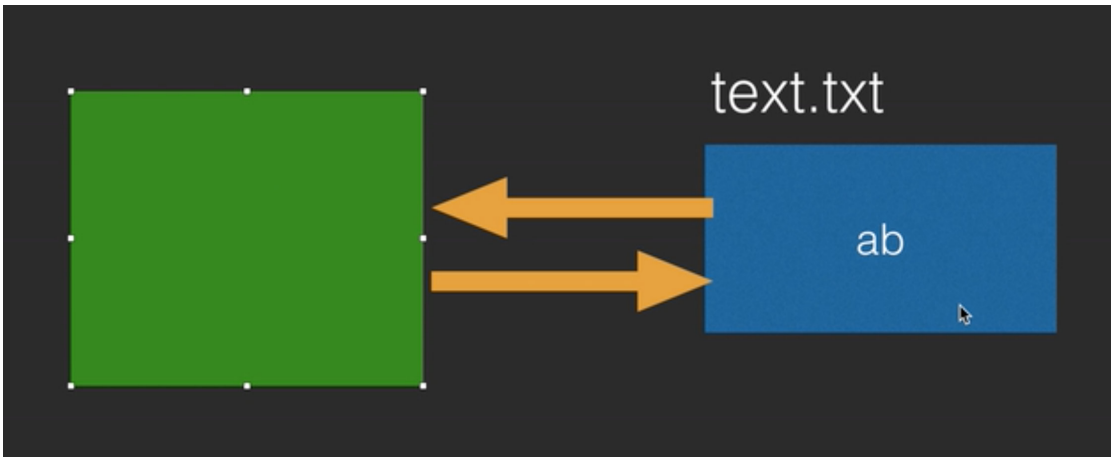
# HenCoder Plus 第 19 课 讲义

# Java I/O 和 Okio

## I/O

- I/O 是什么?
  - 程序内部和外部进行数据交互的过程，就叫输入输出。
    - 程序内部是谁？内存
    - 程序外部是谁?
      - 一般来说是两类：本地文件和网络。
      - 也有别的情况，比如你和别的程序做交互，和你交互的程序也属于外部，但一般来说，就是文件和网络这么两种。
    - 从文件里或者从网络上读数据到内存里，就叫输入；从内存里写到文件里或者发送到网络上，就叫输出
  - Java I/O 作用只有一个：和外界做数据交互
- 用法
  - 使用流，例如 FileInputStream / FileOutputStream

    
    
  - 可以用 Reader 和 Writer 来对字符进行读写
  - 流的外面还可以套别的流，层层嵌套都可以
  - BufferedXXXX 可以给流加上缓冲。对于输入流，是每次多读一些放在内存里面，下次再去数据就不用再和外部做交互（即不必做 IO 操作）；对于输出流，是把数据先在内存里面攒一下，攒够一波了再往外部去写。

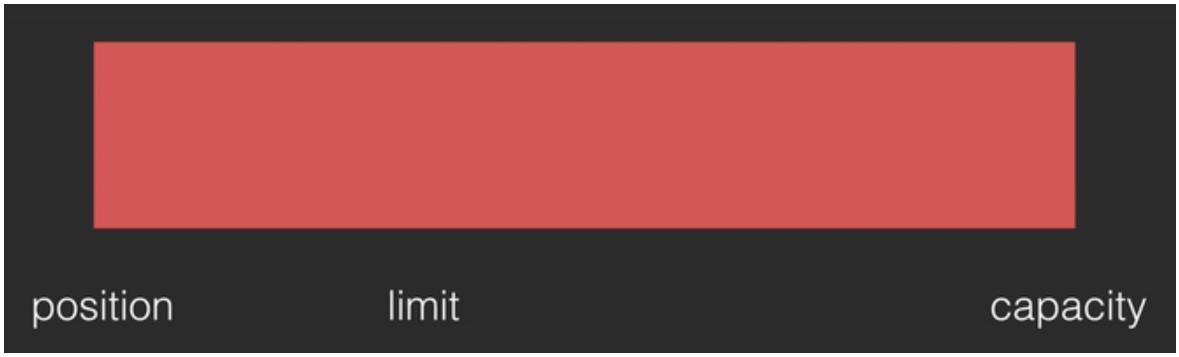
    通过缓存的方式减少和和外部的交互，从而可以提高效率
  - 文件的关闭：close()
  - 需要用到的写过的数据，flush() 一下可以保证数据真正写到外部去（读数据没有这样的担忧）

- 这个就是 Java 的 I/O，它的原理就是内存和外界的交互
  - Java I/O 涉及的类非常多，但你用到哪个再去关注它就行了，不要背类的继承关系图
- 使用 Socket 和 ServerSocket 进行网络交互

```
try {
    ServerSocket serverSocket = new ServerSocket( port: 8080);
    Socket socket = serverSocket.accept();
    BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
    String data;
    while ((data = reader.readLine()) != null) {
        writer.write(data);
        writer.write(System.lineSeparator());
        writer.flush();
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

# NIO

- NIO 和 IO 的区别有几点：
  1. 传统 IO 用的是插管道的方式，用的是 Stream；NIO 用的也是插管道的方式，用的是 Channel。
     - NIO 的 Channel 是双向的
  2. NIO 也用到 buffer
     - 它的 Buffer 可以被操作
     - 它强制使用 Buffer
     - 它的 buffer 不好用
  3. NIO 有非阻塞式的支持
     - 只是支持非阻塞式，而不是全是非阻塞式。默认是阻塞式的
     - 而且就算是非阻塞式，也只是网络交互支持，文件交互是不支持的
- 使用：
  - NIO 的 Buffer 模型：




  - 用 NIO 来读文件的写法：
    - 使用 fine.getChannel() 获取到 Channel
    - 然后创建一个 Buffer
    - 再用 channel.read(buffer)，把文件内容读进去
    - 读完以后，用 flip() 翻页
    - 开始使用 Buffer
    - 使用完之后记得 clear() 一下

```
try (RandomAccessFile file = new RandomAccessFile( name: "./26/new_text.txt",  mode: "r");
     FileChannel channel = file.getChannel()) {
    ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
    channel.read(byteBuffer);
    byteBuffer.flip();
    System.out.println(Charset.defaultCharset().decode(byteBuffer));
    byteBuffer.clear();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

## Okio

特点：

- 它也是基于插管的，而且是单向的，输入源叫 Source，输出目标叫 Sink
- 支持 Buffer
    - 向 NIO 一样，可以对 Buffer 进行操作
    - 但不强制使用 Buffer

用法：

- 输入：

```
try (BufferedSource source = Okio.buffer(Okio.source(new File( pathname: "./26/text.txt")))) {
    System.out.println(source.readUtf8Line());
} catch (IOException e) {
    e.printStackTrace();
}
```

- 输出进去之后，还可以把它当做输入源来取数据：

```
Buffer buffer = new Buffer();
ObjectOutputStream objectOutputStream = new ObjectOutputStream(buffer.outputStream());
objectOutputStream.writeUTF( str: "abab");
objectOutputStream.writeBoolean( val: true);
objectOutputStream.writeChar( val: 'b');
objectOutputStream.flush();
ObjectInputStream objectInputStream = new ObjectInputStream(buffer.inputStream());
System.out.println(objectInputStream.readUTF());
System.out.println(objectInputStream.readBoolean());
```

## 问题和建议?

课上技术相关的问题，都可以在学员群里和大家讨论，我一旦有时间也都会来解答。如果我没来就 @我一下吧！

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。

## 更多内容：

- 网站：https://hencoder.com
- 微信公众号：HenCoder