입출력 관련 공부

Created	@Jan 20, 2021 3:28 PM
Property	
Property 1	
Tags	

입출력이란?

- 입출력(I/O)은 컴퓨터 내부 또는 외부의 장치와 프로그램 간의 데이터를 주고받는 것을 말하고 입출력 목표 지점은 파일, 키보드-모니터, 메모리, 네트워크 연결 등 다양하다.
- 프로그래밍을 시작하는 입장에서 가장 많이 쓰이는건 키보드로 입력(system.in, scanner), 콘솔 화면으로 데이터를 출력하는 작업(system.out.println)일 것이다.
- NIO(New input / output)가 등장하면서 기존 IO와는 달리 버퍼를 사용해 채널방식의 입출력이 가능해 졌다. 그리고 비동기 방식 지원과 넌블로킹/블로킹 모두 지원하면서 네트워크 쪽 지원이 대폭 좋아졌다.

간단한 콘솔 입출력과 덜 간단한 파일 입출력

콘솔 입출력

```
import java.io.InputStream;

public class StreamTest {
    public static void main(String[] args) throws Exception {
        InputStream in = System.in; // 자바 내장 클래스 사용

        int a;
        int b;
        int c;

        a = in.read(); // 1byte씩 3번 입력받음(단, 저장은 int형태로)
        b = in.read();
        c = in.read();

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

/* 콘솔창 */
abc (입력)
97 (출력)
98 (출력)
99 (출력)
```

- 배열을 사용해 3만큼 입력된 값 그대로 출력

```
import java.io.InputStream;
import java.io.InputStreamReader;

public class StreamTest {
    public static void main(String[] args) throws Exception {
        InputStream in = System.in;
       // 아스키 코드로 저장되는게 아닌 입력값 그대로 저장
        InputStreamReader reader = new InputStreamReader(in);
        char[] a = new char[3];// 고정된 길이(3)만큼만 읽도록 되어있다.(개선점)
        reader.read(a);

        System.out.println(a);
    }
}
```

- 엔터를 누를때까지 사용자의 입력 받기

```java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

public class StreamTest {
    public static void main(String[] args) throws Exception {
        InputStream in = System.in;
        InputStreamReader reader = new InputStreamReader(in);
        BufferedReader br = new BufferedReader(reader); // 추가

        String a = br.readLine(); // 엔터키를 누를때까지 입력한 값 전부 받기
        System.out.println(a);
    }
}

/* 핵심 */
// InputStream - byte
// InputStreamReader - character
// BufferedReader - String
```

- scanner를 사용해 입출력 진행

```java
import java.util.Scanner; // j2se(자바 스탠다드 에디션) 5.0부터 사용가능

public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(sc.next());
    }
}
```

※ 참조

```
System.err라는 것도 있는데 System.out과 동일한 역할을 한다.
다만 System.err는 오류메시지를 출력할 경우에 사용하게 되어 있다.
unix의 경우 콘솔 프로그램 실행 시 출력옵션을 지정하면 System.out으로 출력한 내용과 System.err로 출력한 내용을 별도의 파일로 저장할 수도 있다.
```

파일 입출력

- 파일 쓰기1

```java
import java.io.FileOutputStream;
import java.io.IOException;

public class FileWrite {
    public static void main(String[] args) throws IOException {
        // InputStream과 마찬가지로 byte단위로 데이터 처리하는 클래스
        // 출력시킬 파일 위치를 지정
        FileOutputStream output = new FileOutputStream("c:/out.txt");
        // fw는 byte 배열 대신 문자열을 직접 파일에 쓸수 있다. 하지만 개행 문자는 계속 써줘야한다
        // FileWriter fw = new FileWriter("c:/out.txt");
        // pw는 fw의 기능에 개행까지 적용해주는 방식
        // PrintWriter pw = new PrintWriter("c:/out.txt");
        for(int i=1; i<11; i++) {
            String data = i+" 번째 줄입니다.\r\n";
            output.write(data.getBytes()); // 출력시키기 위해 string를 byte로 변환
            // fw.write(data);
            // String data = i + " 번째 줄입니다."; pw.println(data);
        }
        output.close();
    }
}
```

- 파일에 내용 추가

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class FileWrite {
    public static void main(String[] args) throws IOException {
        PrintWriter pw = new PrintWriter("c:/out.txt");
```

```
        for(int i=1; i<11; i++) {
            String data = i+" 번째 줄입니다.";
            pw.println(data);
        }
        pw.close();

        // FileWriter의 두번째 인자가 추가모드(append)로 열것인지 설정해주는 boolean 인자다.
        PrintWriter pw2 = new PrintWriter(new FileWriter("c:/out.txt", true));
        for(int i=11; i<21; i++) {
            String data = i+" 번째 줄입니다.";
            pw2.println(data);
        }
        pw2.close();
    }
}
```

- 파일 읽기(길이를 한정)

```
import java.io.FileInputStream;
import java.io.IOException;

public class FileRead {
    public static void main(String[] args) throws IOException {
        // 1024만큼의 문자만 받아오기
        byte[] b = new byte[1024];
        // 파일에서 문자 단위로 읽는 스트림 클래스
        FileInputStream input = new FileInputStream("c:/out.txt");
        input.read(b);
        System.out.println(new String(b));
        input.close();
    }
}
```

- 파일 읽기(라인단위로 EOF까지 읽기)

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileRead {
    public static void main(String[] args) throws IOException {
        // 문자로 읽을 때 배열을 제공하여 한꺼번에 읽을 수 있도록 함(br)
        // https://velog.io/@ednadev/%EC%9E%90%EB%B0%94-%EC%9E%85%EC%B6%9C%EB%A0%A5%EA%B3%BC-%EC%8A%A4%ED%8A%B8%EB%A6%BCStream
        BufferedReader br = new BufferedReader(new FileReader("c:/out.txt"));
        while(true) {
            String line = br.readLine();
            if (line==null) break; // 라인이 없는 경우(EOF) 종료
            System.out.println(line);
        }
        br.close();
    }
}
```

그래서 스트림(Stream)이란?

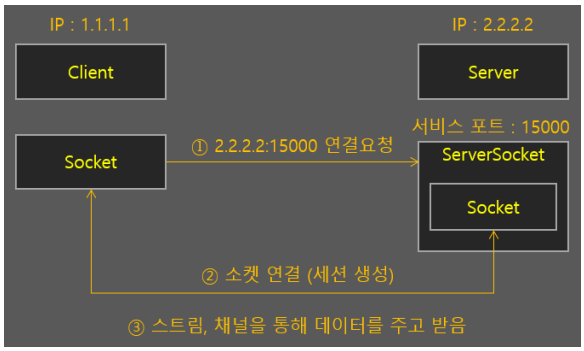
- 가장 쉽게 이해하려면 수도꼭지를 생각하면 된다. 수도꼭지를 틀면 물이 나오고 수도꼭지를 잠그면 물이 나오지 않는다. A라는 곳에서부터 B라는 곳까지 수도관이 연결되어 있고 A에서 계속 물을 보낸다면 B에서 수도꼭지를 틀때마다 물이 나오게 될 것이다. 여기서 스트림은 A수도관에서 B수도관으로 이동하는 물의 흐름이라고 할 수 있다.
- FIFO(First In First Out) 구조
- **스트림 하나당 한 방향으로만 가기에 두 방향(입력, 출력)으로 진행하려면 2가지 필요(네트워크상 데이터 교환을 위해 입출력 스트림이 따로 필요)**
- **프로그램에서 데이터 처리하는 속도와 입출력에서 수행되는 속도 차이를 줄이기 위해 사용**한다.
  - 하드디스크의 처리속도는 메모리의 전기적 처리 속도를 따라갈 수 없음
  - 따라서 스트림은 버퍼(일정량의 데이터를 저장하는 메모리 공간)를 사용하여 속도차이를 줄인다.
- 프로그래밍에서는 다음과 같은 것들을 스트림이라고 할 수 있다.
  - 파일 데이터 (파일은 그 시작과 끝이 있는 데이터의 스트림이다.)
  - HTTP 응답 데이터 (브라우저가 요청하고 서버가 응답하는 HTTP 응답 데이터도 스트림이다.)

- 키보드 입력 (사용자가 키보드로 입력하는 문자열은 스트림이다.)

- 자바에서 스트림은 기본적으로 바이트 단위로 데이터를 전송합니다.
- 하지만 **자바에서 가장 작은 타입인 char 형이 2바이트이므로, 1바이트씩 전송되는 바이트 기반 스트림으로는 원활한 처리가 힘들**기에 **바이트 기반뿐만 아니라 문자 기반의 스트림**도 존재.

그럼 네트워크에서 입출력은?

- 소켓을 통해 서버-클라이언트 간 통신을 열어주는, 즉 세션을 맺어준 뒤 스트림이나 채널을 열어서 IO작업 진행
- 서버 소켓은 소켓 생성 전 서버에서 클라이언트 응답을 기다릴 수 있도록 서비스 포트를 생성해서 대기하는 작업을 말함. 포트를 지정해 해당 서비스를 나눔



서버 소켓 생성 및 데이터 주기

```
package hs;

import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
  public static void main(String[] args) {

    // 서버 소켓을 생성해서 대기 상태로 만듦
    // 클라이언트에서 연결 요청이 오면 소켓을 생성해서 연결시킴
    // 서비스 포트를 15000으로 설정(위 그림과 같음)
    try(ServerSocket server = new ServerSocket(15000);
        Socket socket = server.accept()) {

      // 연결이 되었으므로 데이터를 전송해줄 스트림 통로를 생성함
      OutputStream out = socket.getOutputStream();

      // 속도 향상을 위해 버퍼 필터 스트림을 사용함 (선택사항)
      BufferedOutputStream bufOut = new BufferedOutputStream(out);

      // 연결되면 보낼 데이터
      String sendData = "서버 접속 완료.";

      // 버퍼에 먼저 내용을 넣음
      // 중요 : 바로 출력하는 것이 아님에 주의, 버퍼에 출력하는 것
      // byte로 출력하기 위해 변환 후 버퍼에 쓰기
      bufOut.write(sendData.getBytes());

      // 버퍼에 있는 내용을 실제로 출력함
      bufOut.flush();

    } catch (IOException e) {

      System.out.println("연결에 실패했습니다.");
    }

    // try-catch-finally를 사용했다면 socket.close()로 소켓연결을 닫아줘야함
  }
}
```

클라이언트 소켓 생성 및 서버에서 데이터 받기

```java
package hs;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;

public class Client {

  public static void main(String[] args) {

    // 서버와 소켓 연결 성공하면 데이터를 받아올 InputStream 생성
    try(Socket socket = new Socket("172.30.1.29", 15000);
        InputStream in = socket.getInputStream()) {

      // 속도 향상을 위해 버퍼 필터 스트림 사용 (선택사항)
      BufferedInputStream bufIn = new BufferedInputStream(in);

      // 받은 데이터를 저장해줄 바이트 배열
      byte[] dataFromServer = new byte[100];

      // 데이터를 읽어옴
      bufIn.read(dataFromServer);

      // 받은 데이터 출력
      System.out.println(new String(dataFromServer));

    } catch (IOException e) {
      System.out.println("서버 연결에 실패했습니다.");
    }

  }
}
```

블로킹(blocking)? 논블로킹(non-blocking)?

- **블로킹**은 네트워크 송수신을 기다리는 동안 해당 쓰레드가 대기 상태로 멈춰 있는 것을 말하고(read()메소드를 호출시 데이터 입력 때까지 wait하는 것) **논블로킹**은 입출력 작업시 쓰레드가 블로킹 되지 않도록 해 하나의 쓰레드에서 여러 입출력을 처리하도록 하는 방식을 말한다.
- 블로킹 특징
  - 채널, 스트림 구성시 송수신 받기 위해 쓰레드를 계속 중지 상태로 둬야함
  - 주고 받는 데이터가 없을 경우 쓰레드가 멈춰있어 사용자 수에 맞게 쓰레드를 늘려줘야함 → 사용자가 많은 경우 쓰레드 풀의 쓰레드도 늘려줘야 해서 오버헤드 발생할 수 있음
  - NIO에서는 스레드를 인터럽트로 빠져나올 수 있음
    (IO에서는 무조건 스트림을 닫아야지 블로킹 탈출함)
- 논블로킹 특징
  - 네트워크 소켓 채널은 NIO을 지원하지만 쓰레드가 하나인 이상 병렬 처리는 못함 → 시간이 오래 걸리는 작업은 논블로킹에서도 별도 쓰레드 생성 필요
  - 여러 채널에서 발생하는 유형별 이벤트를 돌아가면서 처리함

스트림 vs 채널

- IO는 스트림 기반이라 입출력시 따로따로 스트림을 생성해줘야 하지만 NIO는 양방향 입출력이 가능한 채널 기반이라 입출력을 따로 해주는 별도의 채널 생성은 불필요하다.
  (파일 하나에 대한 입출력은 그저 FileChannel 하나만으로 해결된다.)

개발시 IO와 NIO 선택

- NIO는 불특정 다수의 클라이언트 연결, 멀티 파일들을 논블로킹/비동기 처리할 수 있기에 효율적인 쓰레드 사용이 가능. 버퍼 사용으로 입출력 성능 향상. 단, 하나의 입출력 처리 작업이 길 경우 비추(버퍼의 크기 문제)
- IO같은 경우 연결 클라이언트 수가 적고 전송되는 데이터가 대용량에 FIFO 처리할 필요가 있을 경우에 좋다.

❓ 궁금한 것들

- 객체 스트림도 있던데 우리가 하려는 소켓 프로그래밍에 필요한가

⏭️ 다음에 다뤄보면 좋을 것들

- 쓰레드의 전반적인 구동, 실제로 쓰이는 형식 + 동기, 비동기
- NIO 구조와 작동 원리 → IO도 사용해서 둘 다 사용하는 채팅 프로그램 구현

※ 참고 자료

- https://blog.naver.com/PostView.nhn?blogId=rain483&logNo=220625042360&parentCategoryNo=&categoryNo=32&viewDate=&isShowPopularPosts=f
- https://wikidocs.net/226
- https://codevang.tistory.com/184
- https://palpit.tistory.com/640
- https://lktprogrammer.tistory.com/62
- https://ju3un.github.io/network-basic-1/