2017.12.12 ~ 2017.12.15

롯데정보통신

JAVA SE PROGRAMMING 고급

문성훈

목차

01. 객체지향 기본 개념

02. 객체지향 설계 원칙

03. ClassLoader

04. Java8 Lambda

05. Java Collection (Stream)

06. Java NIO

06. JAVA NIO

❖ Java NIO 개요

- NIO: New Input/Output 이라는 뜻으로 Java 4부터 java.nio 패키지 포함.
  자바 7에서 자바 IO와 자바 NIO 사이의 일관성 없는 클래스 설계를 바로 잡고, 비동기 채널 등의 네트워크 지원을 대폭 강화한 NIO.2 API가 추가

- IO와 NIO의 비교

구분	IO	NIO
입출력 방식	Stream 방식	Channel 방식
버퍼 방식	non-buffer	buffer
비동기 방식	지원 안 함	지원
블로킹 / 넌블로킹 방식	블로킹 방식	블로킹 / 넌 블로킹 모두 지원

❖ Java NIO 개요 (cont.)

- 네트워크 프로그램을 개발할 때 IO와 NIO 선택 기준

  - NIO는 불특정 다수의 클라이언트 연결 또는 멀티 파일들을 넌 블로킹이나 비동기로 처리할 수 있기 때문에 과도하게 Thread가 생성되는 것을 방지할 수 있고 Thread를 효과적으로 재사용 할 수 있음.

  - NIO는 연결 클라이언트 수가 많고,
    하나의 입출력 처리 작업이 오래 걸리지 않는 경우에 사용.

  - IO는 연결 클라이언트 수가 적고,
    전송되는 데이터가 대용량이면서
    순차적으로 처리될 필요성이 있는 경우에 사용

❖ Path interface

- Java IO의 java.io.File 클래스에 대응되는 NIO 인터페이스
- 파일의 경로는 절대경로, 상대경로 둘 다 사용가능.
- 파일명, 부모 폴더 이름 등을 알 수 있음.
- 예제 작성)

❖ Files class

- 파일과 폴더의 속성을 읽는 method제공
- 파일과 폴더의 생성 및 삭제 method 제공
- 예제 작성)

❖ WatchService interface

- 자바 7에서 처음 소개된 것으로 지정한 폴더 내부에서 파일 생성, 파일 삭제, 파일명 수정 등의 변화를 감시하는데 사용.

- 파일 변경 통지 메커니즘으로 사용.

- 예제 작성)

❖ Buffer

- NIO에서는 항상 buffer를 사용.

- IO는 기본이 non-buffer.

- Buffer는 읽고 쓰기가 가능한 메모리 배열.
  buffer를 이해하고 잘 사용할 수 있어야 NIO에서 제공하는 API를 올바르게 활용할 수 있음.

- NIO buffer는 저장되는 데이터 타입에 따라서 별도의 클래스로 제공
  (CharBuffer, IntBuffer, ByteBuffer, etc.)

❖ Buffer (cont.)

- Buffer가 사용하는 메모리의 위치에 따라서 non-direct buffer와 direct buffer로 분류.

- non-direct buffer

  - JVM이 관리하는 Heap 메모리 공간을 이용하는 버퍼.
  - JVM Heap 메모리를 사용하므로 버퍼 생성 시간이 빠름.
  - JVM의 제한된 Heap 메모리를 사용하므로 버퍼의 크기를 크게 잡을 수가 없음.
  - direct buffer를 사용하는 것보다 입출력 성능이 낮음.

❖ Buffer (cont.)

- Buffer가 사용하는 메모리의 위치에 따라서 non-direct buffer와 direct buffer로 분류.

- direct buffer

  - OS가 관리하는 메모리 공간을 이용하는 버퍼.
  - OS의 메모리를 할당 받기 위해 OS의 native-c function을 호출해야 하기 때문에 상대적으로 생성 속도가 느림. (재 사용해서 사용)
  - OS가 관리하는 메모리를 사용하므로 OS가 허용하는 범위 내에서 대용량 버퍼를 생성 가능.
  - 실제 사용에서는 non-direct buffer에 비해 효율이 좋음.

❖ Buffer (cont.)

- Buffer가 사용하는 메모리의 위치에 따라서 non-direct buffer와 direct buffer로 분류.

- 예제 작성) buffer 생성 속도 비교

❖ ByteBuffer 사용

- Channel이 데이터를 읽고 쓰는 버퍼는 모두 ByteBuffer.
- 프로그램에서 가장 많이 처리되는 데이터는 String.
- Channel을 통해 읽는 String 데이터를 복원하려면 ByteBuffer안의 데이터를 String으로 변환해야 하며 반대로 String을 Channel을 통해 쓰고 싶다면 String을 ByteBuffer로 변환해야 함.
- Channel을 통해 String을 파일이나 네트워크로 전송하려면 특정 문자셋(UTF-8, EUC-KR)으로 Encoding해서 ByteBuffer로 변환.
- 예제 작성) 문자열 〈-〉 ByteBuffer

❖ ByteBuffer 사용 (cont.)

- Buffer의 위치 속성
  - position : buffer안에서 현재 읽거나 쓰는 위치 값(index값)
  - limit : buffer안에서 읽거나 쓸 수 있는 위치의 한계값(초기값은 capacity와 같다.)
  - capacity : 버퍼의 최대 메모리 개수(index가 아님)
  - mark : reset()을 이용해서 돌아오는 위치를 지정하는 index. mark()로 지정. 단, mark값은 position값 보다 반드시 작아야 하며 그렇지 않은 경우 자동으로 제거. mark가 설정되지 않은 상태에서 reset()을 호출하면 Exception.

❖ ByteBuffer 사용 (cont.)

- Buffer의 위치 속성

  - flip() : buffer안에 내용을 다 읽어 들인 후 flip()을 호출하면 buffer안의 내용이 들어 있는 최대 index를 limit로 설정하고 position값을 0으로 세팅.(limit와 position의 위치 조절)
  - mark() : 현재 position위치에 mark값 설정
  - reset() : mark의 위치로 position 이동
  - rewind() : limit는 변하지 않고 position만 0으로 설정. 자동적으로 mark는 제거.
  - clear() : mark는 삭제, position은 0으로 limit는 capacity값으로 세팅.
    버퍼안의 데이터는 삭제되지 않는다.

❖ File 처리

- File 처리를 하기 위해서는 FileChannel 이용.
- FileChannel은 동기화 처리가 되어 있기 때문에 멀티 Thread 환경에서 안전.
- FileChannel은 static method인 open()을 호출해서 획득
- 예제 작성)

❖ File 처리 (cont.)

- File로부터 byte를 읽기 위해서는 FileChannel의 read() method 호출.
- read() method의 리턴 값은 파일에서 ByteBuffer로 읽혀진 바이트 수.
- 더 이상 읽을 바이트가 없다면 read() method는 -1을 리턴.
- 예제 작성)

❖ Asynchronous File 처리

- FileChannel의 read()와 write() method는 파일 입출력 작업 동안 blocking.

- 별도의 Thread를 생성해서 이 method를 호출해야 한다.
  만약 동시에 처리해야 할 파일 수가 많다면 Thread의 수도 증가하기 때문에 성능저하 발생.
  (ThreadPool을 이용해서 해결)

- Java NIO는 불특정 다수의 파일 및 대용량 파일의 입출력 작업을 위해서
  AsynchronousFileChannel을 별도로 제공.

❖ Asynchronous File 처리 (cont.)

- AsynchronousFileChannel은 read()와 write() method를 호출하면
  내부 ThreadPool에게 작업 처리를 요청하고 이 method들을 즉시 리턴.

- 실질적인 입출력 작업 처리는 ThreadPool의 Thread가 담당.

- Thread가 파일 입출력을 완료하게 되면 callback method가 자동 호출되어 작업 완료 후
  실행해야 할 코드가 있으면 실행

- 예제 작성)

❖ ServerSocketChannel & SocketChannel

- ServerSocketChannel과 SocketChannel을 이용한 Network 통신

- 예제 작성)

Thanks for Listening