Java프로그래밍

10강. java.nio 패키지의 활용 (교재 9장)

컴퓨터과학과 김희천 교수

한국방송통신대학교

오늘의 학습목차

1. java.nio 패키지
2. 버퍼
3. FileChannel 클래스
4. WatchService 인터페이스

1. java.nio 패키지

1) java.nio 패키지

- NIO는 'New Input Output의 약자'
  - ✓ 기존 java.io 패키지를 개선한 새로운 입출력 패키지
- JDK 7부터는 파일 I/O를 개선한 NIO2도 등장
  - ✓ java.nio와 그것의 서브 패키지 형태
    - java.nio.file , java.nio.channels, java.nio.charset 등
  - ✓ File 클래스보다 다양한 기능을 제공하는 Path
  - ✓ Files의 static 메소드를 통한 파일/디렉터리의 조작, 파일의 읽기/쓰기
  - ✓ 입력과 출력이 모두 가능한 FileChannel 클래스
    - 버퍼링 기능, 멀티스레드에 안전
  - ✓ 비동기식 입출력을 위한 AsynchronousFileChannel 클래스
    - non-blocking 방식 파일 입출력

2) Path 인터페이스

- java.nio.file 패키지에 존재하며 java.io.File 클래스를 대신함
- 파일시스템에 존재하는 파일이나 디렉터리에 해당하는 경로를 표현함
  - ✓ 절대 경로 또는 상대 경로로 표현됨
- 경로의 생성, 경로의 비교, 경로 정보 추출, 경로 요소 조작 기능 등을 제공
- java.nio.file.Files 클래스의 static 메소드를 이용해 Path 객체에 대한 다양한 실제 조작(읽기, 쓰기, 복사, 이동 등)이 가능함

Path 객체의 생성 방법

- java.nio.file.Paths.get(" C:\\tmp\\foo ")
  - ✓ 파일이나 디렉터리 경로(절대 또는 상대 경로)를 명시해야 함

3) Path 인터페이스의 메소드

◆ 메소드

- ✓ int compareTo(Path other)
- ✓ Path getFileName( )
- ✓ FileSystem getFileSystem( )
- ✓ Path getName(int index)
- ✓ int getNameCount( )
- ✓ Path getParent( )
- ✓ Path getRoot( )
- ✓ Iterator<Path> iterator( )
- ✓ File toFile( )
- ✓ String toString( )

4) Path 인터페이스의 사용 예

```
import java.util.*;
import java.nio.file.*;

public class PathTest {
   public static void main(String args[ ]) {
      try {
         Path path = Paths.get("C:\\windows\\system32\\drivers\\etc\\hosts");
         System.out.println("파일 이름 : " + path.getFileName( ));
         System.out.println("상위 폴더 : " + path.getParent( ).getFileName( ));
         System.out.println("경로 길이 : " + path.getNameCount( ));

         System.out.print("현재 경로 : ");
         for (int i = 0; i < path.getNameCount( ); i++)
              System.out.print(path.getName(i) + "\\");

         Iterator<Path> it = path.iterator( );
         System.out.print("\n현재 경로 : ");
         while (it.hasNext( ))
             System.out.print(it.next( ).getFileName( ) + "\\");
           } catch (Exception e) {
         System.out.println(e);
    }
   }
}
```

```
파일 이름 : hosts
상위 폴더 : etc
경로 길이 : 5
현재 경로 : windows₩system32₩drivers₩etc₩hosts₩
현재 경로 : windows₩system32₩drivers₩etc₩hosts₩
```

5) FileSystem과 FileStore 클래스

- FileSystem의 메소드
  - ✓ FileSystem은 파일 시스템에 대한 인터페이스를 제공
    - 하나 이상의 파일 스토어로 구성됨
    - FileSystems.getDefault( )은 기본 파일 시스템을 리턴함
  - ✓ Iterable <FileStore> getFileStores( )
  - ✓ WatchService newWatchService( )
- FileStore의 메소드
  - ✓ FileStore는 파티션(또는 볼륨)을 표현함
  - ✓ String name( ), String type( )
  - ✓ long getTotalSpace( ),
  - ✓ long getUnallocatedSpace( ), long getUsableSpace( )

6) FileSystem과 FileStore 클래스의 사용 예

```
import java.nio.file.*;

public class FileSystemTest {
   public static void main(String args[]) throws Exception {
       FileSystem fs = FileSystems.getDefault( );
       for (FileStore store : fs.getFileStores( )) {
           System.out.println("드라이브 이름 : " + store.name( ));
           System.out.println("파일시스템 타입 : " + store.type( ));
           long total = store.getTotalSpace( );
           long free = store.getUnallocatedSpace( );
           System.out.println("전체 공간 : " + total + " bytes");
           System.out.println("사용 중인 공간 : " + (total - free) + "
bytes");
           System.out.println("사용 가능한 공간 : " + free + " bytes");
           System.out.println( );
       }
   }
}
```

```
드라이브 이름 : C드라이브
파일시스템 타입 : NTFS
전체 공간 : 2000290836480 bytes
사용 중인 공간 : 97686507520 bytes
사용 가능한 공간 : 1902604328960 bytes
```

7) Files 클래스

- 파일 조작 기능을 제공하는 static 메소드를 제공함
  - ✓ 메소드는 Path 객체를 인자로 가지고 작업함
  - ✓ 파일의 읽기와 쓰기
    - byte[ ] readAllBytes(Path), Path write(Path, byte[ ])
  - ✓ 파일이나 디렉터리의 검사/생성/삭제/복사/이동/속성관리
    - boolean isDirectory(Path), boolean isRegularFile(Path)
    - Path createFile(Path), void delete(Path)
    - Path copy(Path, Path), Path move(Path, Path)
    - long size(Path), UserPrincipal getOwner(Path)

8) Files 클래스의 사용 예

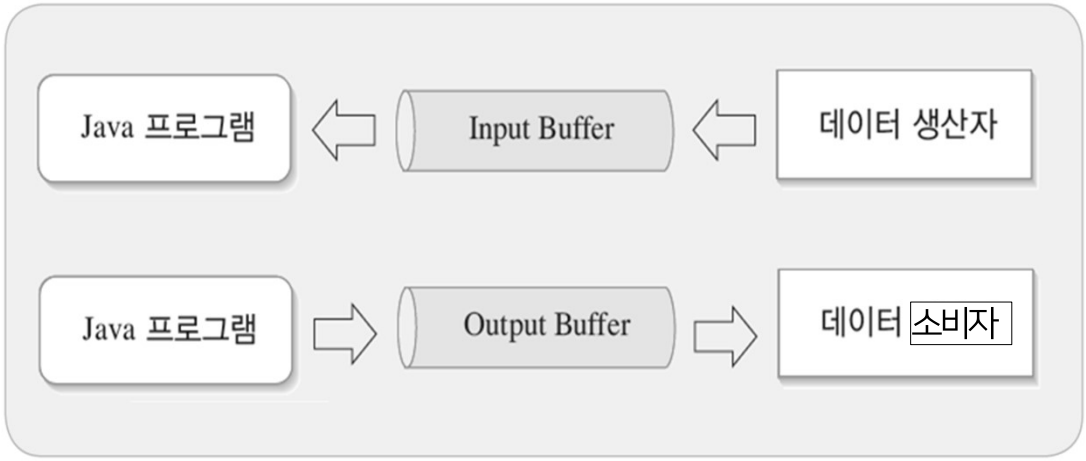
```
import java.nio.file.*;

public class FilesTest2 {
   public static void main(String args[ ]) throws Exception {
        Path path = Paths.get("C:\\Java");

        DirectoryStream<Path> ds = Files.newDirectoryStream(path);
        for (Path p : ds) {
           if (Files.isDirectory(p)) {
                System.out.println("[디렉터리] " + p.getFileName( ));
           } else {
                System.out.print("[파일] " + p.getFileName( ));
                System.out.println(" (" + Files.size(p) + ")");
           }
        }
   }

}
```

```
[디렉터리] Example
[파일] FileInputStreamTest.java (434)
[파일] FilesTest1.java (800)
[디렉터리] temp
[파일] winhlp.exe (9728)
[파일] winhlp32.exe (9728)
```

2. 버퍼

1) 버퍼

- 데이터 생산자와 프로그램(입력), 프로그램과 데이터 소비자(출력) 간 속도 차로 인해 지연이 발생할 수 있음
- 버퍼를 사용하면 지연 현상을 방지할 수 있음
  - ✓ 프로그램은 버퍼로부터 데이터를 읽음(입력)
  - ✓ 프로그램은 버퍼로 데이터를 출력함(출력)



2) Buffer 클래스

- 버퍼는 기본형 값을 저장하는 데이터 보관소
  - ✓ 채널 입출력에 사용되며 버퍼 단위로 입출력할 수 있음
- java.nio 패키지에 존재하며, Buffer는 추상 클래스
  - ✓ 자식 클래스에서 구현해야 할 공통의 메소드를 선언
- 실제 사용을 위해 boolean을 제외한 모든 기본형에 대해 서브 클래스가 존재함
  - ✓ ByteBuffer, CharBuffer , DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

3) 버퍼의 생성

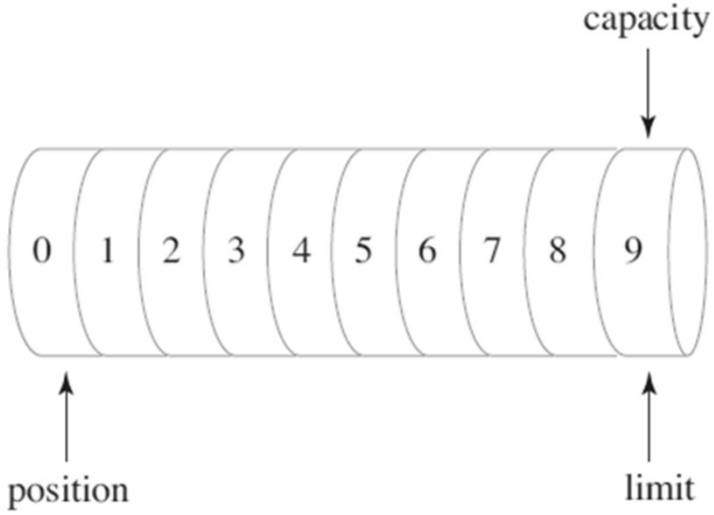
- 버퍼의 생성
  - ✓ Buffer buffer = ByteBuffer.allocate(1024*1024);
  - ✓ byte[ ] barray = new byte[100];
    Buffer bbuffer = ByteBuffer.wrap(barray);

버퍼의 속성

(0 <= mark <= position <= limit <= capacity)

- capacity: 버퍼의 크기(데이터의 개수)로 생성될 때 정해짐
- position: 읽기나 쓰기가 적용되는 위치 (position <= limit)
- limit: 읽거나 쓸 수 없는 최초 위치 (limit <= capacity)
- mark: reset( ) 되었을 때 position이 가리킬 위치

4) Buffer 클래스의 메소드



◆ 메소드

- ✓ Buffer mark( )
  - mark를 position의 값으로 설정
- ✓ Buffer reset( )
  - position을 mark의 값으로 설정
- ✓ Buffer rewind( )
  - position을 0으로 바꾸고 mark를 삭제.
  - 처음부터 다시 읽기를 준비하는 것
- ✓ Buffer flip( )
  - limit를 position 값으로 설정, position은 0으로 변경
  - 버퍼에 쓰기를 끝내고, 버퍼 읽기를 준비하는 것
- ✓ Buffer clear( )
  - 버퍼를 초기 상태로 돌림. 새로운 쓰기를 준비하는 것

5) 버퍼 읽기와 쓰기

- Buffer의 서브클래스에서 제공
  - ✓ ByteBuffer, CharBuffer, DoubleBuffer, …
- 상대적 읽기/쓰기 메소드(ByteBuffer에서)
  - ✓ 현재 position에서 읽기 또는 쓰기를 수행하며, 읽거나 쓴 요소 만큼 position 값이 증가함
  - ✓ byte get(), ByteBuffer get(byte[ ])
  - ✓ ByteBuffer put(byte), ByteBuffer put(byte[ ])
- 절대적 읽기/쓰기 메소드
  - ✓ position 값에 영향을 주지 않음
  - ✓ byte get(int index)
  - ✓ ByteBuffer put(int index, byte b)

3. FileChannel 클래스

3. FileChannel 클래스

1) FileChannel 클래스

- java.io 패키지의 파일 관련 입출력 스트림을 대체
  - ✓ java.nio.channels 패키지에 존재
  - ✓ 파일에 대한 읽기와 쓰기를 모두 제공
  - ✓ 멀티 스레드 환경에서도 안전하게 사용할 수 있음
- 읽기와 쓰기 메소드
  - ✓ int read(ByteBuffer dst), int write(ByteBuffer src)

FileChannel 객체의 생성 방법

- FileChannel.open(Path path, OpenOption … options)
  - ✓ 옵션은 StandardOpenOption.READ 등
- FileInputStream 이나 RandomAccessFile 객체에서 getChannel( )

2) FileChannel 클래스로 파일 만들기

```
import … …
public class FileChannelWriteTest{
   public static void main(String args[ ]) throws IOException{
      String[ ] data = {"안녕하세요, 여러분",… …, "모든 방법에 대해 공부해봅시다."};

      Path path = Paths.get("c:\\Java\\temp\\file.txt");
      Files.createDirectories(path.getParent( ));

      FileChannel fileChannel = FileChannel.open(path,
            StandardOpenOption.CREATE, StandardOpenOption.WRITE);

      Charset charset = Charset.defaultCharset( );
      ByteBuffer buffer;
      int byteCount = 0;
      for(int i = 0;i < data.length;i++) {
         buffer = charset.encode(data[i]);
         byteCount = fileChannel.write(buffer);
      }


      fileChannel.close( );
   }
}
```

안녕하세요, 여러분Java 프로그래밍 언어의 세계로 오신 것을 환영합니다. JDK를 설치하는 방법에서부터Java 프로그램을 compile하고 실행시키는 모든 방법에 대해 공부해봅시다.

3) FileChannel 클래스로 파일 읽기

```
import … …
public class FileChannelReadTest {
   public static void main(String args[ ]) throws IOException {
       Path path = Paths.get("c:\\java\\temp\\file.txt");
       FileChannel fileChannel = FileChannel.open(path,
                      StandardOpenOption.READ);
       ByteBuffer buffer = ByteBuffer.allocate(1024 * 1024);
       Charset charset = Charset.defaultCharset( );

       StringBuffer sb = new StringBuffer( );
       int byteCount;
       while ((byteCount = fileChannel.read(buffer)) >= 0) {
              buffer.flip( );
              sb.append(charset.decode(buffer));
              buffer.clear( );
       }
       System.out.println(sb);
       fileChannel.close;
   }
}
```

안녕하세요, 여러분Java 프로그래밍 언어의 세계로 오신 것을 환영합니다. JDK를 설치하는 방법에서부터Java 프로그램을 compile하고 실행시키는 모든 방법에 대해 공부해봅시다.

4. WatchService 인터페이스

4. WatchService 인터페이스

1) WatchService 인터페이스

- WatchService
  - ✓ 어떤 대상에 대해 변화나 이벤트가 생기는 것을 감시(watch)
  - ✓ 디렉터리의 변화를 감지
    - 디렉터리 내의 파일 또는 서브 디렉터리의 생성, 삭제, 수정
  - ✓ java.nio.file 패키지에 존재

- 감시자의 생성
  - ✓ 먼저 WatchService 객체를 생성함
    - WatchService ws =
      FileSystems.getDefault( ).newWatchService( );

2) 감시 절차(1)

◆ 감시 서비스를 구현하는 절차

✓ 감시 대상 디렉터리를 WatchService에 등록

- Path path = Paths.get("c:\\java\\temp");
- 알림을 받고자 하는 이벤트를 명시
- path.register(ws, StandardWatchEventKinds.ENTRY_CREATE,
  StandardWatchEventKinds.ENTRY_DELETE,
  StandardWatchEventKinds.ENTRY_MODIFY);

✓ WatchService는 take( ) 메소드를 호출하여 감시함

- 무한 루프 안에서, 이벤트가 발생할 때 까지 기다림
- While(true) {
- WatchKey key = ws.take( );
- … …

2) 감시 절차(2)

- ✓ 이벤트가 발생하면, take( )가 리턴하는 WatchKey 객체를 이용하여 이벤트를 처리
- ✓ WatchKey는 감시 대상 객체의 상태 정보를 가짐
  - pollEvents( )를 호출하여 WatchEvent 객체를 얻고 어떤 변화가 생겼는지 알 수 있음

```
  for (WatchEvent<?> event : key.pollEvents( )) {
       WatchEvent.Kind k = event.kind( ); //이벤트 종류
       Path p = (Path)event.context( ); //파일 이름
       … …
  }
  boolean valid = key.reset(); //계속 감시하기 위해
  if (!valid) break;
}
```

3) WatchService 예제

```
public class WatchServiceTest {
  public static void main(String args[]) {
        try {
                WatchService ws;
                ws = FileSystems.getDefault().newWatchService();
                Path path = Paths.get("c:\\java\\temp");
                path.register(ws, StandardWatchEventKinds.ENTRY_CREATE,
StandardWatchEventKinds.ENTRY_DELETE, StandardWatchEventKinds.ENTRY_MODIFY);
                while (true) {
                        WatchKey key = ws.take();
                        for (WatchEvent<?> event : key.pollEvents()) {
                                WatchEvent.Kind k = event.kind();
                                Path p = (Path) event.context();
                                if (k == StandardWatchEventKinds.ENTRY_CREATE)
{
                                        System.out.println("File " +
p.getFileName() + " is created.");
… …
                        }
                        boolean valid = key.reset();
                        if (!valid)  break;
                }
… …
```

```
File 새 텍스트 문서.txt is deleted.
File 새 텍스트.txt is created.
```

Java프로그래밍

다음시간안내

11강. 컬렉션(교재 10장)