## Java NIO



### New I/O

Input/Output (I/O) là một trong những phần căn bản mà bất cứ một người lập trình máy tính nào cũng phải một lần sử dụng khi phát triển một phần mềm. Hiện thực quá trình I/O lúc nào cũng là một thử thách lớn, nhất là tốc độ thực hiện vì tốc độ của I/O phụ thuộc vào nhiều yếu tố: nền tảng hệ điều hành, tốc độ thiết bị lưu trữ,…

Trước version 1.4, Java cung cấp một gói thư viện IO cho người sử dụng là java.io, nhưng nó không đáp ứng được yêu cầu ngày càng cao của người sử dụng. Do đó, Sun Microsystem đã nghiên cứu và cung cấp gói thư viện IO mới, New IO – java.nio, nhằm mục đích cung cấp cho người sử dụng một thư viện có hiệu suất hoạt động cao, portable và scalable.

Java NIO cung cấp các chức năng mở rộng sau:

Buffer đối với primitive types.

* Bộ encoders/decoders cho character-sets.
* Hỗ trợ regular expression
* Cung cấp một abstraction mới, Channel
* Interface mới hỗ trợ locking và memory mapping
* Multiplexed, non-blocking I/O

### Input/Output với hiệu suất sao

Với các version trước Java SDK 1.4, lập trình viên chỉ có một lựa chọn duy nhất cho I/O là sử dụng stream. Có 2 dạng stream cơ bản phục vụ cho tất cả các tác vụ IO của Java tại thời điểm đó: *byte stream* và *character stream*. Java IO cung cấp API để người sử dụng đọc dữ liệu từ stream vào array hay ghi dữ liệu từ array vào stream.

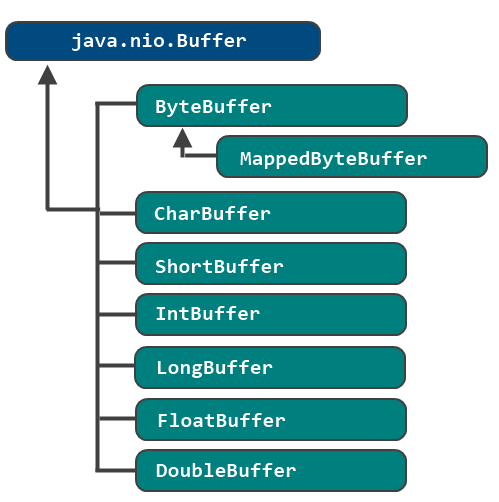
Điểm yếu của Java IO là sử dụng cơ chế blocking, nghĩa là stream sẽ bị block cho tới khi native IO thực hiện xong. Phương thức read() sẽ không return cho tới khi tất cả dữ liệu được đọc vào stream và write() chỉ return khi tất cả dữ liệu được ghi vào stream .Cơ chế này gây ra hiện tượng thắt cổ chai khi thực hiện I/O với hiệu năng cao.

Ngoài việc sử dụng cơ chế blocking, Java IO để lập trình viên phải sử dụng tới raw bytes cũng là một bất tiện trong việc thực hiện IO. Để tăng tốc IO, lập trình viên thường sử dụng một primitive array và truyền vào dữ liệu trực tiếp từ stream. Người sử dụng phải tự đọc raw bytes và kiểm soát EOF (end-of-file). Để tránh việc sử dụng buffering trực tiếp như vậy, người sử dụng được cung cấp một loạt các class implement sẵn buffering như BufferInputStream, BufferReader, BufferWriter,..và sử dụng Decorater Pattern, kết nối một stream vào các lớp buffer này để thực hiện IO. Việc sử dụng này gây bất tiện và rắc rối.

Với những bất lợi đó, từ version 1.4, Java SDK cung cấp gói java.nio với những chức năng và kiến trúc mới nhằm khắc phục các thiếu sót và gói java.io gặp phải. Java NIO cung cấp API tiện lợi và dễ sử dụng hơn, cung cấp các concept mới: Buffer và Channel.

### Buffer

Một Buffer đơn giản chỉ là một container mà bên trong là một chuỗi có giới hạn các primitive type của Java, đơn giản mà nói một Buffer chỉ là một wrapper của một array primitive type. Java NIO cung cấp mỗi một primitive type một loại buffer.



Lớp Buffer là một abstract class cung cấp hầu hết các thao tác mặc định mà một Buffer cần có. Buffer có 7 subclass, tương ứng với 7 primitive type của Java: ByteBuffer, CharacterBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer, DoubleBuffer.

Một Buffer có 3 property chính quan trọng: *capacity*, *limit* và *position*. Các property này được set lúc khởi tạo buffer và trong lúc sử dụng.

Buffer được khởi tạo bởi 2 cách: sử dụng phương thức allocate hay allocateDirect, vốn nhận một tham số là một integer, ấn định *capacity* của Buffer. Điểm khác biện giữa 2 phương thức này là allocateDirect sẽ khởi tạo một buffer nằm bên ngoài heap của JVM, giúp bỏ bới thao tác copy dữ liệu giữa JVM buffer và OS buffer, kết quả là tăng tốc đáng kể tác vụ I/O. Tuy nhiên, việc này cũng có 1 tradeoff, đó là quy trình khởi tạo buffer sẽ tốn nhiều thời gian và công sức hơn. Tuy nhiên yếu điểm này cũng có thể giảm nếu sử dụng một vài practice đúng cách để tối ưu hóa.

// Khởi tạo một buffer có capacity là 5000 bytes

ByteBuffer heapBuffer = ByteBuffer.*allocate*(5000);

// Khởi tạo một DIRECT buffer có capacity là 5000 bytes;

ByteBuffer directBuffer = ByteBuffer.*allocateDirect*(5000);

// Sử dụng buffer

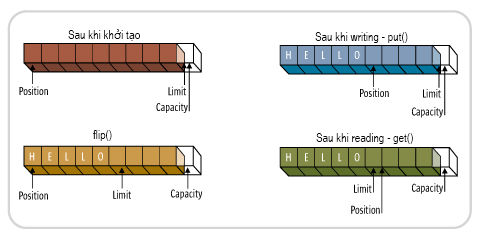
String text = "Hello";

**byte**[] textInBytes = text.getBytes();

heapBuffer.put(textInBytes);

Buffer cung cấp một số phương thức để sử dụng đơn giản:

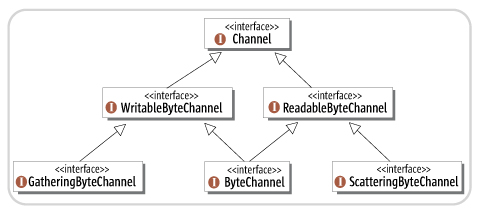
* Phương thức flip: chuẩn bị buffer cho một chuỗi thao tác channel-write hay relative get, phương thức này sẽ set giá trị của limit về position hiện tại và position trở về 0.
* Phương thức get: lấy dữ liệu từ buffer để sử dụng. Có 2 loại: *relative get* lấy dữ liệu từ position hiện tại, và *bulk get*: lấy một lượng dữ liệu từ position hiện tại vào một mảng array.
* Phương thức put: bỏ dữ liệu vào buffer. Cũng có 2 loại giống như get.
* Phương thức rewind: chuẩn bị buffer để thực hiện phương thức get sau khi sử dụng phương thức put. Phương thức này giữ nguyên limit và chuyển position về 0.
* Phương thức clear: chuẩn bị buffer để thực hiện các thao tác channel-read hay relative put. Dời position về 0 và limit tới capacity.
* Phương thức reset: reset buffer. Dời position về 0, limit tới capacity, và bỏ giá trị của mark.



### Channels

Channel là một khái niệm mới được đưa vào Java NIO. Một channel thể hiện một kết nối được mở tới một thực thể có thể thực hiện một hay nhiều tác vụ I/O như thiết bị phần cứng, file, network socket, hay program component. Thí dụ với FileChannel, SocketChannel thể hiện một kênh nối tới một tập tin hay một socket.

Điểm khác biệt lớn nhất giữa stream và channel là với stream, việc thực hiện thao tác read hay write phải tách biệt nhau nhưng với channel thì có thể thực hiện đồng thời cả 2 việc. Điểm khác biệt nữa là trong khi stream hoạt động dựa trên raw bytes thì channel dựa trên buffer.



Interface Channel và các subinterface được định nghĩa trong gói java.nio.channels. Ba interface: ByteChannel, ReadableByteChannel, WritableByteChannel có thể đoán được chức năng của nó từ tên gọi. ScatteringByteChannel và GatheringByteChannel cung cấp khả năng đọc và ghi bytes từ nhiều buffer khác nhau chỉ bằng một lệnh thực thi.

### FileChannel và Memory Mapping

FileChannel tất nhiên là channel thường được sử dụng nhất. Lập trình viên có thể sử dụng FileChannel để đọc, ghi, map, thay đổi nội dung file. Mapping file vớ FileChannel được nâng cấp lên một mức hoàn toàn mới.

Với Java NIO, hoàn toàn có thể map một phần hay toàn bộ file trực tiếp vào bộ nhớ. Bất cứ thay đổi nào xày ra với dữ liệu được map sẽ được cập nhất tới file thật sự. Chức năng này thực sự là một sự bổ sung thiết yếu bởi trước đây để thực hiện được, lập trình viên phải sử dụng native code.

Thực hiện map file chỉ đơn giản như sau:

File largeFile = **new** File("VeryBigLargeFile.dat");

RandomAccessFile raf = **new** RandomAccessFile(largeFile, "rw");

FileChannel fileChannel = raf.getChannel();

MappedByteBuffer mappedByteBuffer = fileChannel.map(MapMode.READ\_WRITE, 0, largeFile.length());

Phương thức map của FileChannel sẽ trả về một MappedByteBuffer, được thiết kế đặc biệt cho chức năng mapping. Ưu điểm chính của việc map file là nội dung của file sẽ được map và sử dụng như là virtual memory của OS. Nếu một phần nào đó của file bị thay đổi, chỉ có phần đó được load vào bộ nhớ vật lý. Với cơ chế này, thích hợp cho việc làm việc với file có dung lượng lớn (có thể là gigabyte).

### Character-sets

Trong Java, một character-set là một hệ thống mapping giữa Unicode character và bytes. Trước đây với Java IO, việc sử dụng character-set vẫn còn hạn chế và thiếu sót với character stream.

Java NIO cung cấp một bộ công cụ (encoder/decoder) để xác định character-set và cung cấp giải thuật encode/decode cho các lớp mapping.

### Regular Expressions

Việc tích hợp Regular Expression vào Java NIO là một bước cải tiến tuyệt vời. Trước đây việc tìm kiếm nội dung trong một file text phải trải qua nhiều bước và gặp phải rất nhiều khó khăn. Với việc cung cấp khả năng search bằng Regular Expression trực tiếp với Channel, moi việc trở nên dễ dạng hơn rất nhiều.