Јава 7 (NIO.2), The asynchronous channel APIs

преведено са: http://www.ibm.com/developerworks/java/library/j-nio2-1/

Асинхрони канал представља конекцију која подржава неблокирајуће операције, попут конектовања, читања и писања и обезбеђује механизме за контролу операција након њиховог покретања.
У односу на Java 1.4 додата су четири асинхрона канала у пакет `java.nio.channels`:

```
AsynchronousSocketChannel
AsynchronousServerSocketChannel
AsynchronousFileChannel
AsynchronousDatagramChannel
```

Ове класе су по стилу сличне са NIO channel API. Поседују исту структуру метода и аргумената и већину операција доступних NIO channel класама. Главна разлика је у томе што нови канали омогућују асинхроно извршавање неких операција.

Постоје два механизма за праћење и контролу започетих асинхроних операција.

- Први је враћање објекта типа `java.util.concurrent.Future`, који моделира операцију у току и може се користити за испитивање њеног стања и добијање резултата.
- Други је прослеђивање операцији објекта нове класе, `java.nio.channels.CompletionHandler`, која дефинише handler методе који се извршавају након што се операција заврши.

Свака класа за асинхрони канал дефинише двојне методе за сваку операцију тако да се може користити произвољни од ова два механизма.

Асинхрони сокет-канали & futures:

Илустрација имплементације једноставног клијента и сервера коришћењем класа `AsynchronousServerSocketChannel` и `AsynchronousSocketChannel`.

Подешавање сервера

Асинхрони серверски сокет канал може бити отворен и везан за адресу слично као `ServerSocketChannel`:

```
AsynchronousServerSocketChannel server =
    AsynchronousServerSocketChannel.open().bind(new InetSocketAddress(1234));
```

Потом можемо рећи каналу да прихвати конекцију:

```
Future<AsynchronousSocketChannel> acceptFuture = server.accept();
```

Ово је прва разлика у односу на NIO. Позив метода `accept()` увек се завршава моментално и, за разлику од `ServerSocketChannel.accept()`, који враћа `SocketChannel`, враћа `Future<AsynchronousSocketChannel>` који се касније може користити за дохватање `AsynchronousSocketChannel`. Генерички тип `Future` објекта је резултат операције. Нпр. read и write враћају `Future<Integer>` јер операција враћа број прочитаних, односно записаних бајтова.

Користећи `Future` објекат текућа нит може блокирати чекајући резултат:

```
AsynchronousSocketChannel worker = acceptFuture.get();
```

Овде нит блокира са timeout-ом од 10 секунди:

```
AsynchronousSocketChannel worker = acceptFuture.get(10, TimeUnit.SECONDS);
```

Нит може испитати текуће стање операције и отказати је:
```
if (!acceptFuture.isDone()) {
    acceptFuture.cancel(true);
}
```

Метод `cancel()` прима `boolean` флег да укаже да ли нит која извршава `accept()` може бити прекинута. Ово је корисно побољшање, јер у претходним верзијама Јаве блокирајуће I/O операције попут ове могле су бити прекинуте једино затварањем сокета.

Подешавање клијента

Отвара се `AsynchronousSocketChannel` и конектује на сервер:
```
AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
client.connect(new InetSocketAddress("localhost", 1234)).get();
```

Коришћење byte buffer-а за читање и писање

Након што је клијент повезан на сервер, читање и писање се врши преко канала коришћењем byte buffer-а:
```
// send a message to the server
ByteBuffer message = ByteBuffer.wrap("ping".getBytes());
client.write(message).get();

// read a message from the client
worker.read(readBuffer).get(10, TimeUnit.SECONDS);
System.out.println("Message: " + new String(readBuffer.array()));
```

API нових асинхроних канала не допушта директно дохватање сокета, док је раније било могуће нпр. звати `socket()` за `SocketChannel`.
Уведена су два нова метода: `getOption()` и `setOption()`, за испитивање и постављање опција сокета у асинхроним мрежним каналима. Нпр. величина бафера за пријем података може се добити помоћу `channel.getOption(StandardSocketOption.SO_RCVBUF)`, уместо са `channel.socket().getReceiveBufferSize()`.

Completion handlers

Алтернативни механизам коришћењу `Future` објеката је регистровање callback-а за асинхрону операцију. Интерфејс `CompletionHandler` има 2 метода:

- `void completed(V result, A attachment)` извршава се ако се задатак заврши са резултатом типа V
- `void failed(Throwable e, A attachment)` извршава се ако задатак не успе због `Throwable e`.

Параметар `attachment` оба метода је објекат који се прослеђује асинхроној операцији. Може се користити за праћење која операција се завршила ако се исти completion handler објекат користи за већи број операција.

Open команде

Погледајмо на примеру класе `AsynchronousFileChannel`.
Можемо креирати нови канал прослеђивањем објекта класе `java.nio.file.Path` статичком методу `open()`:
```
AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(Paths.get("myfile"));
```

`Path` је нова класа у Јави 7. Utility метод `Paths.get(String)` се користи за креирање `Path` од `String`-репрезентације имена фајла.

Подразумевано, фајл се отвара за читање. Метод `open()` може имати додатне опције за задавање како се фајл отвара. Нпр. следећи позив отвара фајл за читање и писање, креирајући га ако је потребно и покушава да га обрише када је канал затворен или се JVM заврши:

```
fileChannel = AsynchronousFileChannel.open(Paths.get("afile"),
    StandardOpenOption.READ, StandardOpenOption.WRITE,
    StandardOpenOption.CREATE, StandardOpenOption.DELETE_ON_CLOSE);
```

Имплементирање completion handler-а

Сада желимо да пишемо у фајл, а након што се писање заврши, да се изврши нешто. Најпре конструишемо `CompletionHandler` који енкапсулира то „нешто“:

```
CompletionHandler<Integer, Object> handler =
    new CompletionHandler<Integer, Object>() {
    @Override
    public void completed(Integer result, Object attachment) {
        System.out.println(attachment + " completed with " + result + " bytes written");
    }
    @Override
    public void failed(Throwable e, Object attachment) {
        System.err.println(attachment + " failed with:");
        e.printStackTrace();
    }
};
```

Сада можемо да извршимо писање:

```
fileChannel.write(ByteBuffer.wrap(bytes), 0, "Write operation 1", handler);
```

Метод `write()` узима:

- `ByteBuffer` са садржајем који треба исписати
- апсолутну позицију у фајлу
- објекат attachment који се прослеђује completion handler методима
- completion handler

Операције морају задати апсолутну позицију у фајлу где се чита или пише. Нема смисла да фајл има интерну ознаку позиције и да се читања/писања дешавају на том месту јер операције могу бити започете пре него што се претходне заврше и није могуће гарантовати поредак којим се дешавају. Из истог разлога не постоје методи у `AsynchronousFileChannel` за испитивање и постављање позиције, који постоје у `FileChannel`.

Осим read и write метода, подржан је и асинхрони lock метод, тако да фајл може бити закључан за ексклузивни приступ без нужног блокирања у текућој нити (или коришћења `tryLock`) ако друга нит тренутно поседује катанац.

Асинхроне групе канала

Сваки конструисани асинхрони канал припада групи канала која дели thread pool. Нити из pool-а користе се за руковање завршетком започетих асинхроних I/O операција. Ово можда звучи као варање, јер можемо имплементирати већину асинхроне функционалности и сами у Јава нитима да добијемо исто понашање, а можда смо се надали да NIO.2може бити имплементиран чисто користећи асинхроне I/O могућности оперативног система због бољих перформанси. Међутим, у неким случајевима неопходно је користити Јава нити: нпр. гарантује се да ће completion-handler методи бити извршени у нитима из pool-а.

Подразумевано, канали конструисани методима `open()` припадају глобалној групи канала која се може конфигурисати коришћењем следећих системских променљивих:

- `java.nio.channels.DefaultThreadPoolthreadFactory` – дефинише `java.util.concurrent.ThreadFactory` који ће бити коришћен уместо подразумеваног
- `java.nio.channels.DefaultThreadPool.initialSize` – одређује иницијалну величину thread pool-а.

Три utility метода у `java.nio.channels.AsynchronousChannelGroup` омогућују креирање нових група канала:

- `withCachedThreadPool()`
- `withFixedThreadPool()`
- `withThreadPool()`

Ови методи узимају или дефиницију thread pool-а, дату као `java.util.concurrent.ExecutorService`, или `java.util.concurrent.ThreadFactory`. Нпр. следећи позив креира нову групу канала која има фиксирани pool од 10 нити, од којих је свака конструисана коришћењем подразумеваног thread factory класе `Executors`:

```
AsynchronousChannelGroup tenThreadGroup =
    AsynchronousChannelGroup.withFixedThreadPool(10, Executors.defaultThreadFactory());
```

Три асинхрона мрежна канала имају алтернативну верзију метода `open()` која прима задату групу канала уместо подразумеване. Нпр. следећи позив каже каналу да користи `tenThreadGroup` уместо подразумеване групе канала како би добио нити када је асинхроним операцијама то потребно:

```
AsynchronousServerSocketChannel channel =
    AsynchronousServerSocketChannel.open(tenThreadGroup);
```

Дефинисање сопствене групе канала омогућује финију контролу над нитима које се користе за сервисирање операција и такође обезбеђује механизме за завршавање нити и чекање завршетка. Пример илуструје контролисање завршетка нити групом канала:

```
// first initiate a call that won't be satisfied
channel.accept(null, completionHandler);
// once the operation has been set off, the channel group can
// be used to control the shutdown
if (!tenThreadGroup.isShutdown()) {
    // once the group is shut down no more channels can be created with it
    tenThreadGroup.shutdown();
}
```

```
if (!tenThreadGroup.isTerminated()) {
    // forcibly shutdown, the channel will be closed and the accept will abort
    tenThreadGroup.shutdownNow();
}
// the group should be able to terminate now, wait for a maximum of 10 seconds
tenThreadGroup.awaitTermination(10, TimeUnit.SECONDS);
```

`AsynchronousFileChannel` се разликује од осталих канала по томе што у циљу коришћења произвољног thread pool-а, метод `open()` узима `ExecutorService` уместо `AsynchronousChannelGroup`.