
ORACLE®

ORACLE®

Операции ввода-вывода в Java

Аркадий Сучилин
arkadiy.sutchilin@oracle.com

Устройства ввода-вывода

Замечание: из рассмотрения в силу своей специфики исключаются графические устройства.

- Физические устройства.
- Иерархия логических устройств.

Одному физическому устройству могут соответствовать несколько логических устройств и наоборот.

В Java мы имеем дело только с логическими устройствами самого верхнего уровня.

Ввод-вывод на физическом уровне

Операции ввода вывода предполагают (помимо простой пересылки данных) управление соответствующими устройствами и временную синхронизацию.

Операция ввода в общем случае предполагает:

- Инициирование операции ввода
- Ожидание готовности данных (завершения операции устройством)
- Считывание данных из устройства

Операция вывода в общем случае предполагает

- Инициирование операции вывода
- Запись данных в устройство
- Ожидание завершения операции устройством

В обоих случаях ожидание завершения операции может быть релизовано как:

- Циклический опрос состояния устройства (polling)
- Обработка аппаратных сигналов прерывания от устройства (interrupt processing)
- Комбинация вышеперечисленных методов

Запись / чтение данных в / из устройства может осуществляться

- Процессором
- Самим устройством в режиме прямого доступа к памяти (DMA - Direct Memory Access)

Ввод-вывод на логическом уровне

Байтовый (двоичный) или символьный?

- Байтовый: обмен данными с устройством осуществляется в "сыром" двоичном виде. Возможно преобразование данных (например, сжатие), но оно всегда происходит без потерь и искажений.
- Символный: данные перед записью в устройство преобразуются в символьный (человекочитаемый) формат, а при считывании из устройства - в "сырой" двоичный формат. При этом возможны искажения данных, связанные с несоответствием возможностей "сырого" и символьного форматов.

Потоковый или блочный?

- Потоковый - данные записываются в устройство и считываются из устройства побайтно/посимвольно. Операция ввода-вывода рассматривается как операция передачи последовательности данных.
- Блочный - данные записываются и считываются блоками (как правило, фиксированной максимальной длины).

Синхронный (блокирующий) или асинхронный?

- Синхронный (блокирующий) - все этапы операции ввода-вывода реализуются в рамках одного метода. Поток (thread), инициировавщий операцию ввода-вывода, блокируется до ее завершения.
- Асинхронный (неблокирующий) - инициация операции ввода-вывода выделена в отдельный метод. Поток (thread), инициировавший операцию ввода вывода продолжает выполняться параллельно с нею.

Базовые средства ввода-вывода в Java

Реализуются классами, входящими в состав пакета `java.io`

Основу пакета составляют классы:

- `File` - абстракция файла. Фактически описывает путь к файлу / каталогу. Реализует ряд базовых операций с файлами / каталогами (создание, удаление, чтение / изменение атрибутов и т.п.).
- `InputStream` / `OutputStream` и их наследники - реализуют синхронный потоковый байтовый ввод-вывод.

- `Reader` / `Writer` - "надстройки" над `InputStream` / `OutputStream`, реализующие символьный ввод-вывод. Фактически осуществляют преобразование между байтовым и символным вводом-выводом.
- `RandomAccessFile` - предоставляет возможность выборочного чтения / записи частей файла. В грубом приближении может рассматриваться как массив байтов во внешнем хранилище.

Класс File

Предоставляет методы общего назначения, например:

- проверить существование файла `boolean exists()`
- получить родительский каталог
  `String getParent(), File getParentFile()`
- определить тип файла
  `boolean isFile(), boolean isDirectory()`
- создание файла / каталога
  `createNewFile(), mkdir()`
- удаление файла `delete()`
- оглавление каталога
  `String [] list(), File [] listFiles()`

Основные методы InputStream

Наиболее часто используемые методы:

- количество доступных байтов `int available()`
- чтение одного байта `int read()` Результат `-1` означает конец файла / потока
- чтение серии байтов в массив `int read(byte[])`
- пропуск указанного числа байтов `skip(long)`
- закрытие потока `void close()`

Наследники InputStream

- буферизованный поток данных - `BufferedInputStream(InputStream)`
- для чтения сериализованного объекта - `ObjectInputStream(InputStream)`
- работа с примитивными типами данных - `DataInputStream(InputStream)`
- поток с возможностью "возврата" прочитанных данных - `PushbackInputStream(InputStream)`
- поток, распаковывающий данные в формате ZIP - `ZipInputStream`
- файловый поток данных - `FileInputStream(File)`

Основные методы OutputStream

- запись байта данных в поток `write(byte b)`
- запись массива данных в поток `write(byte b[])`
- запись всех хранящихся во внутреннем буфере данных `flush()`
- закрытие потока `close()`

Наследники OutputStream

- буферизованный поток данных - `BufferedOutputStream(OutputStream)`
- для записи сериализованного объекта - `ObjectOutputStream(OutputStream)`
- работа с примитивными типами данных - `DataOutputStream(OutputStream)`
- поток для записи символов, а не байтов - `PrintStream(OutputStream)`
- поток, упаковывающий данные в формате ZIP - `ZipOutputStream`
- файловый поток данных - `FileOutputStream(File)`

Классы семейства Reader / Writer

Надстройка над потоками, обеспечивающая переход от байтового ввода-вывода к символьному.

Соответствуют потокам:

- InputStream — Reader, InputStreamReader
- OutputStream — Writer, OutputStreamWriter
- FileInputStream — FileReader, InputStreamReader
- FileOutputStream — FileWriter, OutputStreamWriter

. . .

Пример: копирование файла

```
File inFile = new File("infile.dat");
File outFile = new File("outfile.dat");
FileInputStream inStream = new
   FileInputStream(inFile);
FileOutputStream outStream = new
    FileOutputStream(outFile);
int c;
while ((c = inStream.read()) != -1)
           { outStream.write(c); }
```

Каналы ввода-вывода в Java

- реализуют байтовый блочный ввод-вывод
- могут использовать буферы данных доступные драйверам устройств (и самим устройствам)
- обеспечивают единообразие файловых и сетевых операций ввода-вывода
- каналы, являющиеся наследниками `SelectableChannel` могут работать в неблокирующем режиме
- `FileChannel` также позволяет отображать регион файла на область памяти (Memory Mapped File)
- за счет более естественного взаимодействия с устройствами ввода-вывода, в общем случае, имеют лучшую производительность, чем потоки

ORACLE

Каналы - более эффективная альтернатива потокам.

Основные классы "каналов" (channel), входят в состав пакета `java.nio.channels`

- `DatagramChannel` - прием-передача датаграмм
- `FileChannel` - файловый ввод-вывод
- `SocketChannel` - прием-передача данных через установленное сетевое соединение
- `ServerSocketChannel` - "псевдоканал" - обеспечивает установление сетевого соединения по запросу клиентов (фабрика `SocketChannel`)

Начиная с JDK версии 1.4 рассмотренные ранее классы потоков `FileInputStream`, `FileOutputStream`, а так же `RandomAccessFile` реализованы на базе каналов и содержат метод

```
FileChannel getChannel();
```

В свою очередь, канал всегда можно "обернуть" в поток и даже в `Reader` или `Writer`, используя статические методы утилитного класса `Channels`. Например:

```
static InputStream
        Channels.newInputStream(
                ReadableByteChannel ch)
```

Для хранения блоков данных используются наследники класса `Buffer` - как правило, это `ByteBuffer`. Это контейнеры для данных одного примитивного типа (`char`, `int`, `float`, `double`, а чаще всего, `byte`).

Буферы бывают *direct* (память буфера непосредственно доступна и используются драйвером / устройством) или *non-direct* (память буфера искусственно синхронизируется с областью памяти доступной драйверу / устройству).

Особый тип буфера - `MappedByteBuffer`, представляющий собой фрагмент файла, отображенный на память. Всегда только *direct*. Используется вкупе с `FileChannel`.

Пример: копирование файла v.2

```
File inFile = new File("infile.dat");
File outFile = new File("outfile.dat");
FileChannel inChannel = new
  FileInputStream(inFile).getChannel();
FileChannel outChannel = new
  FileOutputStream(outFile).getChannel();
ByteBuffer buffer =
  ByteBuffer.allocateDirect(1024);
int read;
while ((read = fci.read(buffer)) != -1) {
    buffer.flip();
    fco.write(buffer);
    buffer.clear();  }
```

Пример: копирование файла v.3

```
File inFile = new File("infile.dat");
File outFile = new File("outfile.dat");
FileChannel inChannel = new
  FileInputStream(inFile).getChannel();
FileChannel outChannel = new
  FileOutputStream(outFile).getChannel();

inChannel.transferTo(0, inChannel.size(),
  outChannel);
```

JDK 7: Асинхронный ввод-вывод

Представлен несколькими новыми классами из пакета `java.nio.channels`

- `AsynchronousFileChannel` - асинчхронный аналог `FileChannel`
- `AsynchronousServerSocketChannel` - асинхронный аналог `ServerSocketChannel`
- `AsynchronousSocketChannel` – асинхронный аналог `SocketChannel`

Для работы с блоками данных и синхронные, и асинхронные каналы используют одни и те же классы - наследники `Buffer`

Для реализации асинхронности используется один из двух механизмов:

- Будущий результат операции
- Обработчик события завершения операции

Будущий результат операции

```
Future<Integer> ftr = channel.read(buffer)
```

Асинхронный метод инициирует операцию, но не дожидается ее завершения. Вместо результата метод возвращает объект, который позволит узнать этот результат по завершении операции - `Future<T>`

Методы класса `Future<T>`

- `T get()` - ожидает завершения операции неограниченно долгое время, по завершении возвращает результат
- `T get(long timeout, TimeUnit unit)` - ожидает завершения операции неограниченно долгое время, по завершении возвращает результат или бросает `TimeoutException`
- `boolean cancel(boolean mayInterruptIfRunning)` - отменяет запрос на выполнение операции
- `boolean isCancelled(), boolean isDone()` - в комментариях не нуждаются

Как превратить асинхронную операцию в синхронную ?

```
Future<Integer> ftr = channel.read(buffer);
```

В синхронном (блокирующем) варианте:

```
int result = channel.read(buffer).get();
```

То же самое в "гарантированно независающем" варианте (максимальное время ожидания - 10 сек)

```
int result = channel.read(buffer).get(
                    10, TimeUnit.SECONDS);
```

ORACLE

Обработчик события завершения операции

Интерфейс `CompletionHandler<V, A>`
V - тип результата операции,
A - тип присоединенного объекта (attachment)

Предполагает реализацию двух методов обработки событий:

- `void completed(V result, A attachment)` - вызывается в случае успешного завершения
- `void failed(Throwable exc, A attachment)` - вызывается в случае ошибки

Использование обработчика события завершения операции

```
CompletionHandler<Integer, FileChannel>
   handler = ...

FileChannel attachmt = channel1;
channel1.read(buffer1, attachmt, handler);

attachmt = channel2;
channel2.read(buffer2, attachmt, handler);
```

Тип результата определяется типом операции.
Тип присоединенного объекта определяется потребностями обработчика.

Использование присоединенного объекта позволяет обойтись одним экземпляром `CompletionHandler` для нескольких однотипных операций.

Порядок завершения операций может не совпадать с порядком, в котором они были инициированы.

`CompletionHandler` не позволяет отменить / прервать инициированную операцию.

Совместное использование `Future` и `CompletionHandler` в одной и той же операции не поддерживается.

JDK 7: пакет java.nio.file

Реализует широкий спектр сопутствующих операций:

- Современный аналог `java.io.File` - класс `Path` представляет путь к файлу в структурированном виде
- Типовые операции над файлами и директориями (копирование, перемещение и т.д.) - утилитный класс `Files`.
- Обход дерева файлов / директориев и выполнение операций над каждым файлом / директорием - интерфейс `FileVisitor` и его реализации.

- Мониторинг изменений файлов и директориев - класс `WatchService`.
- Итератор по содержимому директория с возможностью фильтрации - интерфейс `DirectoryStream` и реализующие его классы.
- Абстракция логического диска (партиции) - класс `FileStore`.
- Абстракция логической файловой системы, соответствующей определенной URI-схеме - например, "file", "ftp" и т.п. - класс `FileSystem`.
- Работа с атрибутами файлов и списками управления доступом (ACL - Access Control List) - пакет `java.nio.file.attribute`

ВОПРОСЫ


ORACLE®