



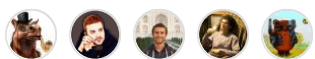
 enhorse / java-interview Public[Code](#) [Issues 9](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) master ▾

...

java-interview / io.md



Cupcake-master Correcting errors in the file io.md

 History 5 contributors 303 lines (230 sloc) | 37.3 KB

...

[Вопросы для собеседования](#)

Потоки ввода/вывода в Java

- В чём заключается разница между IO и NIO?
- Какие особенности NIO вы знаете?
- Что такое «каналы»?
- Какие существуют виды потоков ввода/вывода?
- Назовите основные классы потоков ввода/вывода.
- В каких пакетах расположены классы потоков ввода/вывода?
- Какие подклассы класса `InputStream` вы знаете, для чего они предназначены?
- Для чего используется `PushbackInputStream` ?
- Для чего используется `SequenceInputStream` ?
- Какой класс позволяет читать данные из входного байтового потока в формате примитивных типов данных?
- Какие подклассы класса `OutputStream` вы знаете, для чего они предназначены?
- Какие подклассы класса `Reader` вы знаете, для чего они предназначены?
- Какие подклассы класса `Writer` вы знаете, для чего они предназначены?

- В чем отличие класса `PrintWriter` от `PrintStream` ?
- Чем отличаются и что общего у `InputStream`, `OutputStream`, `Reader`, `Writer` ?
- Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?
- Какие классы позволяют ускорить чтение/запись за счет использования буфера?
- Какой класс предназначен для работы с элементами файловой системы?
- Какие методы класса `File` вы знаете?
- Что вы знаете об интерфейсе `FileFilter` ?
- Как выбрать все элементы определенного каталога по критерию (например, с определенным расширением)?
- Что вы знаете о `RandomAccessFile` ?
- Какие режимы доступа к файлу есть у `RandomAccessFile` ?
- Какие классы поддерживают чтение и запись потоков в компрессированном формате?
- Существует ли возможность перенаправить потоки стандартного ввода/вывода?
- Какой символ является разделителем при указании пути в файловой системе?
- Что такое «абсолютный путь» и «относительный путь»?
- Что такое «символьная ссылка»?

[к оглавлению](#)

В чём заключается разница между IO и NIO?

- **Java IO** (input-output) является потокоориентированным, а **Java NIO** (new/non-blocking io) – буфер-ориентированным. Потокоориентированный ввод/вывод подразумевает чтение/запись из потока/в поток одного или нескольких байт в единицу времени поочередно. Данная информация нигде не кэшируется. Таким образом, невозможно произвольно двигаться по потоку данных вперед или назад. В **Java NIO** данные сначала считываются в буфер, что дает больше гибкости при обработке данных.
- Потоки ввода/вывода в **Java IO** являются блокирующими. Это значит, что когда в потоке выполнения вызывается `read()` или `write()` метод любого класса из пакета `java.io.*`, происходит блокировка до тех пор, пока данные не будут считаны или записаны. Поток выполнения в данный момент не может делать ничего другого. Неблокирующий режим **Java NIO** позволяет запрашивать

считанные данные из канала (channel) и получать только то, что доступно на данный момент, или вообще ничего, если доступных данных пока нет. Вместо того, чтобы оставаться заблокированным пока данные не станут доступными для считывания, поток выполнения может заняться чем-то другим. Тоже самое справедливо и для неблокирующего вывода. Поток выполнения может запросить запись в канал некоторых данных, но не дожидаться при этом пока они не будут полностью записаны.

- В Java NIO имеются селекторы, которые позволяют одному потоку выполнения мониторить несколько каналов ввода. Т.е. существует возможность зарегистрировать несколько каналов с селектором, а потом использовать один поток выполнения для обслуживания каналов, имеющих доступные для обработки данные, или для выбора каналов, готовых для записи.

[к оглавлению](#)

Какие особенности NIO вы знаете?

- Каналы и селекторы: NIO поддерживает различные типы каналов. Канал является абстракцией объектов более низкого уровня файловой системы (например, отображенные в памяти файлы и блокировки файлов), что позволяет передавать данные с более высокой скоростью. Каналы не блокируются и поэтому Java предоставляет еще такие инструменты, как селектор, который позволяет выбрать готовый канал для передачи данных, и сокет, который является инструментом для блокировки.
- Буферы: имеет буферизация для всех классов-обёрток примитивов (кроме Boolean). Появился абстрактный класс Buffer, который предоставляет такие операции, как clear, flip, mark и т.д. Его подклассы предоставляют методы для получения и установки данных.
- Кодировки: появились кодеры и декодеры для отображения байт и символов Unicode. [к оглавлению](#)

Что такое «каналы»?

Каналы (channels) — это логические (не физические) порталы, абстракции объектов более низкого уровня файловой системы (например, отображенные в памяти файлы и блокировки файлов), через которые осуществляется ввод/вывод данных, а **буферы** являются источниками или приёмниками этих переданных данных. При организации вывода, данные, которые необходимо отправить, помещаются в буфер, который затем передается в канал. При вводе, данные из канала помещаются в заранее предоставленный буфер.

Каналы напоминают трубопроводы, по которым эффективно транспортируются данные между буферами байтов и сущностями по ту сторону каналов. Каналы – это шлюзы, которые позволяют получить доступ к сервисам ввода/вывода операционной системы с минимальными накладными расходами, а буферы – внутренние конечные точки этих шлюзов, используемые для передачи и приема данных.

[к оглавлению](#)

Какие существуют виды потоков ввода/вывода?

Назовите основные классы потоков ввода/вывода.

Разделяют два вида потоков ввода/вывода:

- **байтовые** - `java.io.InputStream` , `java.io.OutputStream` ;
- **символьные** - `java.io.Reader` , `java.io.Writer` .

[к оглавлению](#)

В каких пакетах расположены классы потоков ввода/вывода?

`java.io` , `java.nio` . Для работы с потоками компрессированных данных используются классы из пакета `java.util.zip`

[к оглавлению](#)

Какие подклассы класса `InputStream` вы знаете, для чего они предназначены?

- `InputStream` - абстрактный класс, описывающий поток ввода;
- `BufferedInputStream` - буферизованный входной поток;
- `ByteArrayInputStream` позволяет использовать буфер в памяти (массив байтов) в качестве источника данных для входного потока;
- `DataInputStream` - входной поток для байтовых данных, включающий методы для чтения стандартных типов данных Java;
- `FileInputStream` - входной поток для чтения информации из файла;
- `FilterInputStream` - абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства;
- `ObjectInputStream` - входной поток для объектов;
- `StringBufferInputStream` превращает строку (`String`) во входной поток данных `InputStream`;
- `PipedInputStream` реализует понятие входного канала;
- `PushbackInputStream` - разновидность буферизации, обеспечивающая чтение байта с последующим его возвратом в поток, позволяет «заглянуть» во входной поток и увидеть, что оттуда поступит в следующий момент, не извлекая информации.
- `SequenceInputStream` используется для слияния двух или более потоков `InputStream` в единый.

[к оглавлению](#)

Для чего используется `PushbackInputStream`?

Разновидность буферизации, обеспечивающая чтение байта с последующим его возвратом в поток. Класс `PushbackInputStream` представляет механизм «заглянуть» во входной поток и увидеть, что оттуда поступит в следующий момент, не извлекая информации.

У класса есть дополнительный метод `unread()`.

[к оглавлению](#)

Для чего используется `SequenceInputStream`?

Класс `SequenceInputStream` позволяет сливать вместе несколько экземпляров класса `InputStream`. Конструктор принимает в качестве аргумента либо пару объектов класса `InputStream`, либо интерфейс `Enumeration`.

Во время работы класс выполняет запросы на чтение из первого объекта класса `InputStream` и до конца, а затем переключается на второй. При использовании интерфейса работа продолжится по всем объектам класса `InputStream`. По достижении конца, связанный с ним поток закрывается. Закрытие потока, созданного объектом класса `SequenceInputStream`, приводит к закрытию всех открытых потоков.

[к оглавлению](#)

Какой класс позволяет читать данные из входного байтового потока в формате примитивных типов данных?

Класс `DataInputStream` представляет поток ввода и предназначен для записи данных примитивных типов, таких, как `int`, `double` и т.д. Для каждого примитивного типа определен свой метод для считывания:

- `boolean readBoolean()` : считывает из потока булевое однобайтовое значение
- `byte readByte()` : считывает из потока 1 байт
- `char readChar()` : считывает из потока значение `char`
- `double readDouble()` : считывает из потока 8-байтовое значение `double`
- `float readFloat()` : считывает из потока 4-байтовое значение `float`
- `int readInt()` : считывает из потока целочисленное значение `int`
- `long readLong()` : считывает из потока значение `long`
- `short readShort()` : считывает значение `short`
- `String readUTF()` : считывает из потока строку в кодировке UTF-8

[к оглавлению](#)

Какие подклассы класса `OutputStream` вы знаете, для чего они предназначены?

- `OutputStream` - это абстрактный класс, определяющий потоковый байтовый вывод;

- `BufferedOutputStream` - буферизированный выходной поток;
- `ByteArrayOutputStream` - все данные, посылаемые в этот поток, размещаются в предварительно созданном буфере;
- `DataOutputStream` - выходной поток байт, включающий методы для записи стандартных типов данных Java;
- `FileOutputStream` - запись данных в файл на физическом носителе;
- `FilterOutputStream` - абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства;
- `PrintStream` - выходной поток, включающий методы `print()` и `println()`;
- `ObjectOutputStream` - выходной поток для записи объектов;
- `PipedOutputStream` реализует понятие выходного канала.

[к оглавлению](#)

Какие подклассы класса `Reader` вы знаете, для чего они предназначены?

- `Reader` - абстрактный класс, описывающий символьный ввод;
- `BufferedReader` - буферизованный входной символьный поток;
- `CharArrayReader` - входной поток, который читает из символьного массива;
- `FileReader` - входной поток, читающий файл;
- `FilterReader` - абстрактный класс, предоставляющий интерфейс для классов-надстроек;
- `InputStreamReader` - входной поток, транслирующий байты в символы;
- `LineNumberReader` - входной поток, подсчитывающий строки;
- `PipedReader` - входной канал;
- `PushbackReader` - входной поток, позволяющий возвращать символы обратно в поток;
- `StringReader` - входной поток, читающий из строки.

[к оглавлению](#)

Какие подклассы класса `Writer` вы знаете, для чего они предназначены?

- `Writer` - абстрактный класс, описывающий символьный вывод;
- `BufferedWriter` - буферизованный выходной символьный поток;
- `CharArrayWriter` - выходной поток, который пишет в символьный массив;
- `FileWriter` - выходной поток, пишущий в файл;
- `FilterWriter` - абстрактный класс, предоставляющий интерфейс для классов-надстроек;
- `OutputStreamWriter` - выходной поток, транслирующий байты в символы;
- `PipedWriter` - выходной канал;
- `PrintWriter` - выходной поток символов, включающий методы `print()` и `println()`;
- `StringWriter` - выходной поток, пишущий в строку;

[к оглавлению](#)

В чем отличие класса `PrintWriter` от `PrintStream`?

Прежде всего, в классе `PrintWriter` применен усовершенствованный способ работы с символами Unicode и другой механизм буферизации вывода: в классе `PrintStream` буфер вывода сбрасывался всякий раз, когда вызывался метод `print()` или `println()`, а при использовании класса `PrintWriter` существует возможность отказаться от автоматического сброса буферов, выполняя его явным образом при помощи метода `flush()`.

Кроме того, методы класса `PrintWriter` никогда не создают исключений. Для проверки ошибок необходимо явно вызвать метод `checkError()`.

[к оглавлению](#)

Чем отличаются и что общего у `InputStream`, `OutputStream`, `Reader`, `Writer`?

- `InputStream` и его наследники - совокупность для получения байтовых данных из различных источников;
- `OutputStream` и его наследники - набор классов, определяющих потоковый байтовый вывод;
- `Reader` и его наследники определяют потоковый ввод символов Unicode;
- `Writer` и его наследники определяют потоковый вывод символов Unicode.

[к оглавлению](#)

Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?

- `OutputStreamWriter` — «мост» между классом `OutputStream` и классом `Writer`. Символы, записанные в поток, преобразовываются в байты.
- `InputStreamReader` — аналог для чтения. При помощи методов класса `Reader` читаются байты из потока `InputStream` и далее преобразуются в символы.

[к оглавлению](#)

Какие классы позволяют ускорить чтение/запись за счет использования буфера?

- `BufferedInputStream(InputStream in) / BufferedInputStream(InputStream in, int size)`,
- `BufferedOutputStream(OutputStream out) / BufferedOutputStream(OutputStream out, int size)`,
- `BufferedReader(Reader r) / BufferedReader(Reader in, int sz)`,
- `BufferedWriter(Writer out) / BufferedWriter(Writer out, int sz)`

[к оглавлению](#)

Какой класс предназначен для работы с элементами файловой системы?

`File` работает непосредственно с файлами и каталогами. Данный класс позволяет создавать новые элементы и получать информацию существующих: размер, права доступа, время и дату создания, путь к родительскому каталогу.

[к оглавлению](#)

Какие методы класса `File` вы знаете?

Наиболее используемые методы класса `File`:

- `boolean createNewFile()` : делает попытку создать новый файл;

- boolean `delete()` : делает попытку удалить каталог или файл;
- boolean `mkdir()` : делает попытку создать новый каталог;
- boolean `renameTo(File dest)` : делает попытку переименовать файл или каталог;
- boolean `exists()` : проверяет, существует ли файл или каталог;
- String `getAbsolutePath()` : возвращает абсолютный путь для пути, переданного в конструктор объекта;
- String `getName()` : возвращает краткое имя файла или каталога;
- String `getParent()` : возвращает имя родительского каталога;
- boolean `isDirectory()` : возвращает значение `true` , если по указанному пути располагается каталог;
- boolean `isFile()` : возвращает значение `true` , если по указанному пути находится файл;
- boolean `isHidden()` : возвращает значение `true` , если каталог или файл являются скрытыми;
- long `length()` : возвращает размер файла в байтах;
- long `lastModified()` : возвращает время последнего изменения файла или каталога;
- String[] `list()` : возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге;
- File[] `listFiles()` : возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге.

[к оглавлению](#)

Что вы знаете об интерфейсе `FileFilter`?

Интерфейс `FileFilter` применяется для проверки, попадает ли объект `File` под некоторое условие. Этот интерфейс содержит единственный метод `boolean accept(File pathName)` . Этот метод необходимо переопределить и реализовать. Например:

```
public boolean accept(final File file) {  
    return file.exists() && file.isDirectory();  
}
```

[к оглавлению](#)

Как выбрать все элементы определенного каталога по критерию (например, с определенным расширением)?

Метод `File.listFiles()` возвращает массив объектов `File`, содержащихся в каталоге. Метод может принимать в качестве параметра объект класса, реализующего `FileFilter`. Это позволяет включить в список только те элементы, для которых метод `accept` возвращает `true` (критерием может быть длина имени файла или его расширение).

[к оглавлению](#)

Что вы знаете о `RandomAccessFile`?

Класс `java.io.RandomAccessFile` обеспечивает чтение и запись данных в произвольном месте файла. Он не является частью иерархии `InputStream` или `OutputStream`. Это полностью отдельный класс, имеющий свои собственные (в большинстве своем *native*) методы. Объяснением этого может быть то, что `RandomAccessFile` имеет во многом отличающееся поведение по сравнению с остальными классами ввода/вывода так как позволяет, в пределах файла, перемещаться вперед и назад.

`RandomAccessFile` имеет такие специфические методы как:

- `getFilePointer()` для определения текущего местоположения в файле;
- `seek()` для перемещения на новую позицию в файле;
- `length()` для выяснения размера файла;
- `setLength()` для установки размера файла;
- `skipBytes()` для того, чтобы попытаться пропустить определённое число байт;
- `getChannel()` для работы с уникальным файловым каналом, ассоциированным с заданным файлом;
- методы для выполнения обычного и форматированного вывода из файла (`read()`, `readInt()`, `readLine()`, `readUTF()` и т.п.);
- методы для обычной или форматированной записи в файл с прямым доступом (`write()`, `writeBoolean()`, `writeByte()` и т.п.).

Так же следует отметить, что конструкторы `RandomAccessFile` требуют второй аргумент, указывающий необходимый режим доступа к файлу - только чтение (`"r"`), чтение и запись (`"rw"`) или иную их разновидность.

[к оглавлению](#)

Какие режимы доступа к файлу есть у `RandomAccessFile` ?

- `"r"` открывает файл только для чтения. Запуск любых методов записи данных приведет к выбросу исключения `IOException`.
- `"rw"` открывает файл для чтения и записи. Если файл еще не создан, то осуществляется попытка создать его.
- `"rws"` открывает файл для чтения и записи подобно `"rw"`, но требует от системы при каждом изменении содержимого файла или метаданных синхронно записывать эти изменения на физический носитель.
- `"rwd"` открывает файл для чтения и записи подобно `"rws"`, но требует от системы синхронно записывать изменения на физический носитель только при каждом изменении содержимого файла. Если изменяются метаданные, синхронная запись не требуется.

[к оглавлению](#)

Какие классы поддерживают чтение и запись потоков в компрессированном формате?

- `DeflaterOutputStream` - компрессия данных в формате deflate.
- `Deflater` - компрессия данных в формат ZLIB
- `ZipOutputStream` - ПОТОМОК `DeflaterOutputStream` для компрессии данных в формат Zip.
- `GZIPOutputStream` - ПОТОМОК `DeflaterOutputStream` для компрессии данных в формат GZIP.
- `InflaterInputStream` - декомпрессия данных в формате deflate.
- `Inflater` - декомпрессия данных в формате ZLIB
- `ZipInputStream` - ПОТОМОК `InflaterInputStream` для декомпрессии данных в формате Zip.
- `GZIPInputStream` - ПОТОМОК `InflaterInputStream` для декомпрессии данных в

формате GZIP.

[к оглавлению](#)

Существует ли возможность перенаправить потоки стандартного ввода/вывода?

Класс `System` позволяет вам перенаправлять стандартный ввод, вывод и поток вывода ошибок, используя простой вызов статического метода:

- `setIn(InputStream)` - для ввода;
- `setOut(PrintStream)` - для вывода;
- `setErr(PrintStream)` - для вывода ошибок.

[к оглавлению](#)

Какой символ является разделителем при указании пути в файловой системе?

Для различных операционных систем символ разделителя различается. Для Windows это `\`, для Linux - `/`.

В Java получить разделитель для текущей операционной системы можно через обращение к статическому полю `File.separator`.

[к оглавлению](#)

Что такое «абсолютный путь» и «относительный путь»?

Абсолютный (полный) путь — это путь, который указывает на одно и то же место в файловой системе, вне зависимости от текущей рабочей директории или других обстоятельств. Полный путь всегда начинается с корневого каталога.

Относительный путь представляет собой путь по отношению к текущему рабочему каталогу пользователя или активного приложения.

[к оглавлению](#)

Что такое «символьная ссылка»?

Символьная (символическая) ссылка (также «симлинк», Symbolic link) — специальный файл в файловой системе, в котором, вместо пользовательских данных, содержится путь к файлу, который должен быть открыт при попытке обратиться к данной ссылке (файлу). Целью ссылки может быть любой объект: например, другая ссылка, файл, каталог или даже несуществующий файл (в последнем случае, при попытке открыть его, должно выдаваться сообщение об отсутствии файла).

Символьные ссылки используются для более удобной организации структуры файлов на компьютере, так как:

- позволяют для одного файла или каталога иметь несколько имён и различных атрибутов;
- свободны от некоторых ограничений, присущих жёстким ссылкам (последние действуют только в пределах одной файловой системы (одного раздела) и не могут ссылаться на каталоги).

[к оглавлению](#)

Источники

- [Quizful](#)
- [Хабрахабр](#)
- [Освой программирование играючи](#)
- [Metanit](#)
- [javastudy.ru](#)
- [Bruce Eckel «Thinking in Java»](#)

[Вопросы для собеседования](#)