**Java — Потоки ввода/вывода и работа с файлами и каталогами**

Пакет java.io содержит почти каждый класс, который может потребоваться Вам для совершения ввода и вывода в Java. Все данные потоки представлены потоком ввода и адресом вывода. Поток в пакете java.io осуществляет поддержку различных данных, таких как примитивы, объекты, локализованные символы и т.д.

Отличительной чертой многих языков программирования является работа с файлами и потоками. В Java основной функционал работы с потоками сосредоточен в классах из пакета **java.io**.

Ключевым понятием здесь является понятие **потока**. Хотя понятие "поток" в программировании довольно перегружено и может обозначать множество различных концепций. В данном случае применительно к работе с файлами и вводом-выводом мы будем говорить о потоке (stream), как об абстракции, которая используется для чтения или записи информации (файлов, сокетов, текста консоли и т.д.).

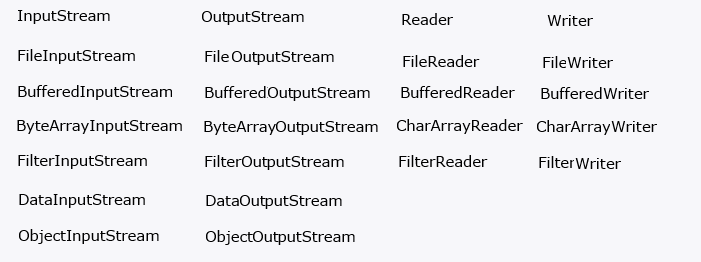
Поток связан с реальным физическим устройством с помощью системы ввода-вывода Java. У нас может быть определен поток, который связан с файлом и через который мы можем вести чтение или запись файла. Это также может быть поток, связанный с сетевым сокетом, с помощью которого можно получить или отправить данные в сети. Все эти задачи: чтение и запись различных файлов, обмен информацией по сети, ввод-ввывод в консоли мы будем решать в Java с помощью потоков.

Объект, из которого можно считать данные, называется **потоком ввода**, а объект, в который можно записывать данные, -**потоком вывода**. Например, если надо считать содержание файла, то применяется поток ввода, а если надо записать в файл - то поток вывода.

В основе всех классов, управляющих потоками байтов, находятся два абстрактных класса: **InputStream** (представляющий потоки ввода) и **OutputStream** (представляющий потоки вывода)

Но поскольку работать с байтами не очень удобно, то для работы с потоками символов были добавлены абстрактные классы **Reader** (для чтения потоков символов) и **Writer** (для записи потоков символов).

Все остальные классы, работающие с потоками, являются наследниками этих абстрактных классов. Основные классы потоков:



|  |  |  |  |
| --- | --- | --- | --- |
| SequenceInputStream |  |  | PrintRriter |
| PipedInputStream | PipedOutputStream | - рекомендуют использовать очереди | | |
| ZipInputStream | ZipOutputStream |  |  |
| RandomAccessFile(“r”),(“rw”) | |  |  |

Классы PipedInputStream и PipedOutputStream характеризуются тем, что их объекты всегда используются в паре – к одному объекту PipedInputStream привязывается (подключается) один объект PipedOutputStream. Они могут быть полезны, если в программе необходимо организовать обмен данными между модулями (например, между *потоками*  выполнения). Эти классы применяются следующим образом: создается по объекту PipedInputStream и PipedOutputStream, после чего они могут быть соединены между собой. Один объект PipedOutputStream может быть соединен с ровно одним объектом PipedInputStream, и наоборот. Затем в объект PipedOutputStream записываются данные, после чего они могут быть считаны именно в подключенном объекте PipedInputStream. Такое соединение можно обеспечить либо вызовом метода connect() с передачей соответствующего объекта PipedI/OStream (будем так кратко обозначать пару классов, в данном случае PipedInputStream и PipedOutputStream ), либо передать этот объект еще при вызове конструктора.

Класс SequenceInputStream объединяет *поток данных* из других двух и более входных *потоков*. Данные будут вычитываться последовательно – сначала все данные из первого *потока* в списке, затем из второго, и так далее. Конец *потока* SequenceInputStream будет достигнут только тогда, когда будет достигнут конец *потока*, последнего в списке.

**Потоки байтов**

**Класс InputStream**

Класс InputStream является базовым для всех классов, управляющих байтовыми потоками ввода. Рассмотрим его основные методы:

* int available(): возвращает количество байтов, доступных для чтения в потоке
* void close(): закрывает поток
* int read(): возвращает целочисленное представление следующего байта в потоке. Когда в потоке не останется доступных для чтения байтов, данный метод возвратит число -1
* int read(byte[] buffer): считывает байты из потока в массив buffer. После чтения возвращает число считанных байтов. Если ни одного байта не было считано, то возвращается число -1
* int read(byte[] buffer, int offset, int length): считывает некоторое количество байтов, равное length, из потока в массив buffer. При этом считанные байты помещаются в массиве, начиная со смещения offset, то есть с элемента buffer[offset]. Метод возвращает число успешно прочитанных байтов.
* long skip(long number): пропускает в потоке при чтении некоторое количество байт, которое равно number

**Класс OutputStream**

Класс OutputStream является базовым классом для всех классов, которые работают с бинарными потоками записи. Свою функциональность он реализует через следующие методы:

* void close(): закрывает поток
* void flush(): очищает буфер вывода, записывая все его содержимое
* void write(int b): записывает в выходной поток один байт, который представлен целочисленным параметром b
* void write(byte[] buffer): записывает в выходной поток массив байтов buffer.
* void write(byte[] buffer, int offset, int length): записывает в выходной поток некоторое число байтов, равное length, из массива buffer, начиная со смещения offset, то есть с элемента buffer[offset].

**Абстрактные классы Reader и Writer**

Абстрактный класс Reader предоставляет функционал для чтения текстовой информации. Рассмотрим его основные методы:

* absract void close(): закрывает поток ввода
* int read(): возвращает целочисленное представление следующего символа в потоке. Если таких символов нет, и достигнут конец файла, то возвращается число -1
* int read(char[] buffer): считывает в массив buffer из потока символы, количество которых равно длине массива buffer. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
* int read(CharBuffer buffer): считывает в объект CharBuffer из потока символы. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
* absract int read(char[] buffer, int offset, int count): считывает в массив buffer, начиная со смещения offset, из потока символы, количество которых равно count
* long skip(long count): пропускает количество символов, равное count. Возвращает число успешно пропущенных символов

Класс Writer определяет функционал для всех символьных потоков вывода. Его основные методы:

* Writer append(char c): добавляет в конец выходного потока символ c. Возвращает объект Writer
* Writer append(CharSequence chars): добавляет в конец выходного потока набор символов chars. Возвращает объект Writer
* abstract void close(): закрывает поток
* abstract void flush(): очищает буферы потока
* void write(int c): записывает в поток один символ, который имеет целочисленное представление
* void write(char[] buffer): записывает в поток массив символов
* absract void write(char[] buffer, int off, int len) : записывает в поток только несколько символов из массива buffer. Причем количество символов равно len, а отбор символов из массива начинается с индекса off
* void write(String str): записывает в поток строку
* void write(String str, int off, int len): записывает в поток из строки некоторое количество символов, которое равно len, причем отбор символов из строки начинается с индекса off

Функционал, описанный классами Reader и Writer, наследуется непосредственно классами символьных потоков, в частности классами FileReader и FileWriter соответственно, предназначенными для работы с текстовыми файлами.

Теперь рассмотрим конкретные классы потоков.

**19.1. Потоки**

**Потоки в Java** определяются в качестве последовательности данных. Существует два типа потоков:

* **InPutStream** – поток ввода используется для считывания данных с источника.
* **OutPutStream** – поток вывода используется для записи данных по месту назначения.

Java предоставляет сильную, но гибкую поддержку в отношении ввода/вывода, связанных с файлами и сетями, однако в данном руководстве рассмотрены лишь базовые функции, связанные с потоками и вводом/выводом. Рассмотрим далее по порядку наиболее распространенные примеры.

**Байтовый поток**

Потоки байтов в Java используются для осуществления ввода и вывода 8-битных байтов. Не смотря на множество классов, связанных с потоками байтов, наиболее распространено использование следующих классов: FileInputStream и FileOutputStream. Ниже рассмотрен пример, иллюстрирующий использование данных двух классов для копирования из одного файла в другой.

**Примечание по примеру:** чтобы скопировать файл, необходимо в папке проекта создать файл file.txt с любым или пустым содержимым.

**Пример**

import java.io.\*;

public class FileCopy {

public static void main(String args[]) throws IOException {

FileInputStream fileIn = null;

FileOutputStream fileOut = null;

try {

fileIn = new FileInputStream("file.txt");

fileOut = new FileOutputStream("copied\_file.txt");

int a;

// Копирование содержимого файла file.txt

while ((a = fileIn.read()) != -1) {

fileOut.write(a); // Чтение содержимого файла file.txt и запись в файл copied\_file.txt

}

}finally {

if (fileIn != null) {

fileIn.close();

}

if (fileOut != null) {

fileOut.close();

}

}

}

}

Теперь рассмотрим файл file.txt со следующим содержимым:

Содержимое файла file.txt

В качестве следующего шага необходимо скомпилировать java-программу и выполнить ее, что позволит создать файл copied\_file.txt с тем же содержимым, что имеется в file.txt. Таким образом, разместим обозначенный код в файле FileCopy.java и выполним следующее действие:

$javac FileCopy.java

$java FileCopy

**Символьные потоки**

Потоки байтов в Java позволяют произвести ввод и вывод 8-битных байтов, в то время как потоки символов используются для ввода и вывода 16-битного юникода. Не смотря на множество классов, связанных с потоками символов, наиболее распространено использование следующих классов: FileReader и FileWriter. Не смотря на тот факт, что внутренний FileReader использует FileInputStream, и FileWriter использует FileOutputStream, основное различие состоит в том, что FileReader производит считывание двух байтов в конкретный момент времени, в то время как FileWriter производит запись двух байтов за то же время.

Мы можем переформулировать представленный выше пример, в котором два данных класса используются для копирования файла ввода (с символами юникода) в файл вывода.

**Примечание по примеру:** чтобы скопировать файл, необходимо в папке проекта создать файл file.txt с любым или пустым содержимым.

**Пример**

import java.io.\*;

public class FileCopy {

public static void main(String args[]) throws IOException {

FileReader fileIn = null;

FileWriter fileOut = null;

try {

fileIn = new FileReader("file.txt");

fileOut = new FileWriter("copied\_file.txt");

int a;

while ((a = fileIn.read()) != -1) {

fileOut.write(a);

}

} finally {

if (fileIn != null) {

fileIn.close();

}

if (fileOut != null) {

fileOut.close();

}

}

}

}

Или например, запишем в файл строку (современное написание, без закрывающего ***finally***):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import java.io.\*;    public class Program {        public static void main(String[] args) {            String text = "Hello world!"; // строка для записи          try(FileOutputStream fos=new FileOutputStream("[C://SomeDir//notes.txt](file:///C:\SomeDir\notes.txt)"))          {              // перевод строки в байты              byte[] buffer = text.getBytes();                fos.write(buffer, 0, buffer.length);          }          catch(IOException ex){                System.out.println(ex.getMessage());          }          System.out.println("The file has been written");      }  } |

**Стандартные потоки**

Все языки программирования обеспечивают поддержку стандартного ввода/вывода, где программа пользователя может произвести ввод посредством клавиатуры и осуществить вывод на экран компьютера. Если вы знакомы с языками программирования C либо C++, вам должны быть известны три стандартных устройства STDIN, STDOUT и STDERR. Аналогичным образом, Java предоставляет следующие три стандартных потока:

* Стандартный ввод – используется для перевода данных в программу пользователя, клавиатура обычно используется в качестве стандартного потока ввода, представленного в виде System.in.
* Стандартный вывод – производится для вывода данных, полученных в программе пользователя, и обычно экран компьютера используется в качестве стандартного потока вывода, представленного в виде System.out.
* Стандартная ошибка – используется для вывода данных об ошибке, полученной в программе пользователя, чаще всего экран компьютера служит в качестве стандартного потока сообщений об ошибках, представленного в виде System.err.

Ниже представлена простая программа, которая создает InputStreamReader для чтения стандартного потока ввода, до введения пользователем "q":

**Пример**

import java.io.\*;

public class ReadConsole {

public static void main(String args[]) throws IOException {

InputStreamReader inStRe = null;

try {

inStRe = new InputStreamReader(System.in);

System.out.println("Введите символы, символ 'q' для выхода.");

char a;

do {

a = (char) inStRe.read();

System.out.print(a);

} while(a != 'q');

}finally {

if (inStRe != null) {

inStRe.close();

}

}

}

}

Разместим представленный выше код в файле ReadConsole.java и попробуем скомпилировать и выполнить его согласно тому, как это представлено в следующей программе. Данная программа продолжает чтение и вывод одного и того же символа до нажатия 'q':

$javac ReadConsole.java

$java ReadConsole

Введите символы, 'q' для выхода.

proglang.su

proglang.su

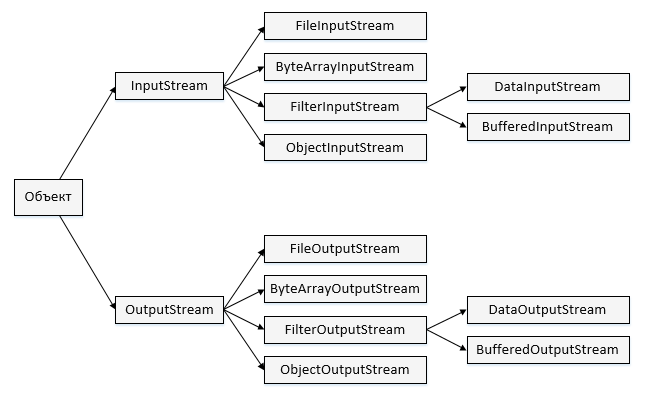
q

q

**Чтение и запись файла**

Как было указано выше, поток представляет собой последовательность данных. InputStream используется для считывания данных с источника, OutputStream служит для записи данных по месту назначения.

Ниже представлена иерархия классов для управления потоками Ввода и Вывода.



В данном уроке нам предстоит рассмотреть два важных потока: FileInputStream и FileOutputStream.

**Поток FileInputStream – чтение из файла**

**Поток FileInputStream** – это поток, который используется в Java для чтения данных из файла. Объекты могут быть созданы при использовании ключевого слова new, доступны несколько типов конструкторов.

Представленный конструктор использует имя файла в качестве потока с целью создания объекта входного потока для считывания файла:

InputStream a = new FileInputStream("D:/myprogramm/java/test");

Представленный ниже конструктор использует объектный файл с целью создания объекта входного потока для чтения файла. Сперва мы создаем объектный файл при использовании метода File() следующим образом:

File a = new File("D:/myprogramm/java/test");

InputStream a = new FileInputStream(a);

Теперь, получив объект InputStream, следует ознакомиться со следующим перечнем вспомогательных методов, которые могут быть использованы для считывания потока либо выполнения иных операций в потоке.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public void close() throws IOException{}** Данный метод в Java закрывает выходной файловый поток. Освобождает какие-либо системные ресурсы, связанные с файлом. Выдает IOException. |
| 2 | **protected void finalize()throws IOException {}** Данный метод выполняет очистку соединения с файлом. Позволяет удостовериться в вызове закрытого метода данного выходного файлового потока при отсутствии каких-либо ссылок на данный поток. Выдает IOException. |
| 3 | **public int read(int r)throws IOException{}** Данный метод осуществляет в Java считывание заданных байтов данных из InputStream. Возврат данных типа int. Возврат следующего байта данных, в конце файла будет произведен возврат к -1. |
| 4 | **public int read(byte[] r) throws IOException{}** Данный метод производит считывание байтов r.length из входного потока в массив. Возврат общего числа считанных байтов. В конце файла будет произведен возврат к -1. |
| 5 | **public int available() throws IOException{}** Выдает число байтов, которые могут быть считаны из входного файлового потока. Возврат данных типа int. |

Существуют также другие доступные входные потоки, более детальные сведения о которых представлены по следующим ссылкам:

* [ByteArrayInputStream](http://proglang.su/java/io-and-files-and-directories-bytearrayinputstream)
* [DataInputStream](http://proglang.su/java/io-and-files-and-directories-datainputstream)

**Поток FileOutputStream – создание и запись файла**

**Поток FileOutputStream** – это поток, который используется в Java для создания файла и последующей записи в него. Поток создаст файл в случае его отсутствия перед его открытием для вывода.

Далее представлены два конструктора, которые могут быть задействованы при создании объекта FileOutputStream.

Представленный конструктор использует имя файла в качестве строки с целью создания объекта входного потока для записи файла в Java:

OutputStream a = new FileOutputStream("D:/myprogramm/java/test")

Представленный ниже конструктор использует объектный файл с целью создания объекта выходного потока для записи файла. Сперва мы создаем объектный файл при использовании метода File() следующим образом:

File a = new File("D:/myprogramm/java/test");

OutputStream a = new FileOutputStream(a);

Теперь, получив объект OutputStream, следует ознакомиться со следующим перечнем вспомогательных методов, которые могут быть использованы для записи потока либо выполнения иных операций в потоке.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public void close() throws IOException{}** Данный метод в Java закрывает выходной файловый поток. Освобождает какие-либо системные ресурсы, связанные с файлом. Выдает IOException. |
| 2 | **protected void finalize()throws IOException {}** Данный метод выполняет очистку соединения с файлом. Позволяет удостовериться в вызове закрытого метода данного выходного файлового потока при отсутствии каких-либо ссылок на данный поток. Выдает IOException. |
| 3 | **public void write(int w)throws IOException{}** Данный метод осуществляет запись заданного байта в выходной поток. |
| 4 | **public void write(byte[] w)** Запись байтов w.length из указанного массива байтов в OutputStream. |

Существуют также другие доступные выходные потоки, более детальные сведения о которых представлены по следующим ссылкам:

* [ByteArrayOutputStream](http://proglang.su/java/io-and-files-and-directories-bytearrayoutputstream)
* [DataOutputStream](http://proglang.su/java/io-and-files-and-directories-dataoutputstream)

**Пример**

В следующем примере представлены InputStream и OutputStream – потоки для чтения, создания и записи файла:

import java.io.\*;

public class File {

public static void main(String args[]) {

try {

char c[] = {'a','b','c'};

OutputStream output = new FileOutputStream("file.txt"); // Создание текстового файла

for(int i = 0; i < c.length; i++) {

output.write(c[i]); // Запись каждого символа в текстовый файл

}

output.close();

InputStream input = new FileInputStream("file.txt");

int size = input.available();

for(int j = 0; j < size; j++) {

System.out.print((char)input.read() + " "); // Чтение текстового файла посимвольно

}

input.close();

}catch(IOException e) {

System.out.print("Exception");

}

}

}

Представленный выше java-код создаст файл file.txt и пропишет заданные символы в формате char. То же самое будет выводиться на экран стандартного вывода.

**Навигация по файловой системе и вводу/выводу**

Существует ряд других классов, которые нам предстоит рассмотреть с целью ознакомления с основами навигации в файловой системе и вводу/выводу.

* [File Class](http://proglang.su/java/io-and-files-and-directories-file-class)
* [FileReader Class](http://proglang.su/java/io-and-files-and-directories-filereader-class)
* [FileWriter Class](http://proglang.su/java/io-and-files-and-directories-filewriter-class)

**19.2.1 Java — Класс ByteArrayInputStream: конструкторы и методы**

**Класс ByteArrayInputStream** позволяет использовать буфер в памяти как InputStream. Входным источником является массив байтов.

**Конструкторы**

Класс ByteArrayInputStream предоставляет следующие конструкторы.

|  |  |
| --- | --- |
| **№** | **Конструктор и описание** |
| 1 | **ByteArrayInputStream(byte [] a)** Конструктор принимает байтовый массив как параметр. |
| 2 | **ByteArrayInputStream(byte [] a, int off, int len)** Конструктор принимает массив байтов и два целочисленных значения, где **off** – первый байт для чтения, а **len** – количество байтов, которые нужно cчитать. |

**Методы**

Если используйте объект ByteArrayInputStream, то у Вас есть под рукой вспомогательные методы, которые можно использовать для чтения потока или для выполнения других операций над потоком.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public int read()** Метод считывает следующий байт данных из InputStream. Возвращает int как байт данных. Если это конец файла, он возвращает -1. |
| 2 | **public int read(byte[] r, int off, int len)** Метод считывает до **len** количество байт, начиная с **off** входного потока в массива. Возвращает общее количество прочитанных байтов. Если это конец файла, будет возвращен -1. |
| 3 | **public int available()** Получение количества байтов, которые могут быть прочитаны из этого потока ввода файла. Возвращает int, который получает количество прочитанных байтов. |
| 4 | **public void mark(int read)** Устанавливает текущую отмеченную позицию в потоке. Параметр возвращает максимальный предел байтов, который может быть прочитан до того, как помеченная позиция станет недействительной. |
| 5 | **public long skip(long n)** Пропускает 'n' количество байтов из потока. Возвращает фактическое количество пропущенных байтов. |

**19.2.2 Java — Конструкторы и методы класса ByteArrayOutputStream**

Поток класса ByteArrayOutputStream создает буфер в памяти, и все данные, отправленные в поток, хранятся в буфере.

**Конструкторы**

Ниже приведен список конструкторов, предоставляемых классом ByteArrayOutputStream.

|  |  |
| --- | --- |
| **№** | **Конструктор и описание** |
| 1 | **ByteArrayOutputStream()** Конструктор создает ByteArrayOutputStream с буфером в 32 байт. |
| 2 | **ByteArrayOutputStream(int a)** Конструктор создает ByteArrayOutputStream с буфером заданного размера. |

**Методы**

Если Вы используйте объект ByteArrayOutputStream, тогда у Вас всегда есть список вспомогательных методов, которые можно использовать для записи потока или для выполнения других операций над потоком.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public void reset()** Метод сбрасывает количество действительных байт в выходном потоке байтового массива до нуля, поэтому все накопленное на выходе будет сброшено. |
| 2 | **public byte[] toByteArray()** Метод создает недавно выделенный массив байтов. Его размер будет текущим размером выходного потока, и содержимое буфера будет скопировано в него. Возвращает текущее содержимое выходного потока в виде байтового массива. |
| 3 | **public String toString()** Преобразует содержимое буфера в строку. Перевод будет выполняться в соответствии с кодировкой установленной по умолчанию. Возвращает строку, переведенную из содержимого буфера. |
| 4 | **public void write(int w)** Запись указанного массива в выходной поток. |
| 5 | **public void write(byte []b, int of, int len)** Запись len количества байтов, начиная смещение с of. |
| 6 | **public void writeTo(OutputStream outSt)** Запись всего содержимого потока в указанный аргумент потока. |

**Пример**

Ниже приведен пример демонстрации классов ByteArrayInputStream и ByteArrayOutputStream.

import java.io.\*;

public class TestByteStream {

public static void main(String args[])throws IOException {

ByteArrayOutputStream outputByte = new ByteArrayOutputStream(12);

while(outputByte.size()!= 5) {

outputByte.write("hello".getBytes());

}

byte a [] = outputByte.toByteArray();

System.out.println("Вывод содержимого:");

for(int i = 0 ; i < a.length; i++) {

// Вывод символов

System.out.print((char)a[i] + " ");

}

System.out.println();

int b;

ByteArrayInputStream inputByte = new ByteArrayInputStream(a);

System.out.println("Преобразование символов в верхний регистр:" );

for(int j = 0 ; j < 1; j++) {

while(( b = inputByte.read())!= -1) {

System.out.println(Character.toUpperCase((char)b));

}

inputByte.reset();

}

}

}

Получим следующий результат:

Вывод содержимого:

h e l l o

Преобразование символов в верхний регистр:

H

E

L

L

O

**19.3.1 Java — Класс DataInputStream и его методы**

**DataInputStream класс-обертка** используется в контексте DataOutputStream и может использоваться для чтения примитивных типов данных и дает дополнительные методы.

Ниже конструктор для создания InputStream:

InputStream in = DataInputStream(InputStream in);

**Методы**

Если используйте объект DataInputStream, то у Вас есть под рукой вспомогательные методы, которые можно использовать для чтения потока или для выполнения других операций над потоком.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public final int read(byte[] r, int off, int len)throws IOException** Чтение до len байтов данных из входного потока в массив байтов. Возвращает общее количество байтов, считанных в буфер, в противном случае -1, если это конец файла. |
| 2 | **public final int read(byte [] b)throws IOException** Считывает некоторые байты из входного потока и сохраняет в массив байтов. Возвращает общее количество байтов, считанных в буфер, в противном случае -1, если это конец файла. |
| 3 | **(1) public final Boolean readBooolean()throws IOException** **(2) public final byte readByte()throws IOException** **(3) public final short readShort()throws IOException** **(4) public final Int readInt()throws IOException** Эти методы будут считывать байты из содержимого InputStream. Возвращает следующие два байта InputStream в качестве конкретного примитивного типа. |
| 4 | **public String readLine() throws IOException** Считывает следующую строку текста из входного потока. Метод считывает последовательно байты, преобразуя каждый байт отдельно в символ, пока он не встретит ограничитель строки или конец файла; прочитанные символы затем возвращаются как строка. |

**19.3.2 Java — Класс DataOutputStream и его методы**

Поток **DataOutputStream класс-обертка** позволяет записывать примитивные данные в исходный код и дает дополнительные методы.

Ниже конструктор для создания DataOutputStream:

DataOutputStream out = DataOutputStream(OutputStream out);

**Методы**

Если используйте объект DataOutputStream, то у Вас есть под рукой вспомогательные методы, которые можно использовать для записи потока или для выполнения других операций над потоком.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public final void write(byte[] w, int off, int len)throws IOException** Запись len байт из указанного байтового массива, начиная с off, в основной поток. |
| 2 | **public final int write(byte [] b)throws IOException** Запись текущего количества байтов, записанных в этот выходной поток данных (DataOutputStream). Возвращает общее количество байтов, записанных в буфер. |
| 3 | **(1) public final void writeBooolean()throws IOException** **(2) public final void writeByte()throws IOException** **(3) public final void writeShort()throws IOException** **(4) public final void writeInt()throws IOException** Эти методы будут записывать данные определенного примитивного типа в выходной поток в виде байтов. |
| 4 | **Public void flush()throws IOException** Сбрасывает поток вывода данных. |
| 5 | **public final void writeBytes(String s) throws IOException** Записывает строку в основной выходной поток в виде последовательности байтов. Каждый символ в строке записывается последовательно, отбрасывая свои восемь бит. |

**Пример**

Ниже приведен пример демонстрации DataInputStream и DataOutputStream. В этом примере происходит запись и чтение с указанного файла test.txt.

import java.io.\*;

public class TestByteStream {

public static void main(String args[])throws IOException {

// Запись строки в файл в кодировке UTF-8

DataOutputStream dataOutput = new DataOutputStream(new FileOutputStream("E:\\test.txt"));

dataOutput.writeUTF("proglang.su");

// Чтение данных из того же файла

DataInputStream dataInput = new DataInputStream(new FileInputStream("E:\\test.txt"));

while(dataInput.available() > 0) {

String a = dataInput.readUTF();

System.out.print(a);

}

}

}

Получим следующий результат:

proglang.su

**19.4 Java — Буферизованные потоки BufferedInputStream и BufferedOutputStream**

Служат для “оборачивания” других потоков с целью оптимизации операций ввода-вывода. Эти потоки добавляют к стандартным специальный буфер в памяти, с помощью которого повышается производительность при чтении и записи потоков.

**Класс BufferedInputStream**

Класс BufferedInputStream накапливает вводимые данные в специальном буфере без постоянного обращения к устройству ввода. Класс BufferedInputStream определяет два конструктора:

BufferedInputStream(InputStream inputStream)

BufferedInputStream(InputStream inputStream, int bufSize)

Первый параметр - это поток ввода, с которого данные будут считываться в буфер. Второй параметр - размер буфера.

Например, буферизируем считывание данных из потока ByteArrayInputStream:

import java.io.\*;

public class Program {

    public static void main(String[] args) {

         String text = "Hello world!";

         byte[] buffer = text.getBytes();

        ByteArrayInputStream in = new ByteArrayInputStream(buffer);

        try(BufferedInputStream bis = new BufferedInputStream(in)){ //достаточно странный пример, т.к. как написано выше поток ByteArrayInputStream и так уже буферизирован. Хотя ниже написано, что это еще больше оптимизирует производительность (х.з.)

            int c;

            while((c=bis.read())!=-1){

                System.out.print((char)c);

            }

        }

        catch(Exception e) { System.out.println(e.getMessage()); }

        System.out.println();

    }

}

Класс BufferedInputStream в конструкторе принимает объект InputStream. В данном случае таким объектом является экземпляр класса ByteArrayInputStream.

Как и все потоки ввода BufferedInputStream обладает методом read(), который считывает данные. И здесь мы считываем с помощью метода read каждый байт из массива buffer.

Фактические все то же самое можно было сделать и с помощью одного ByteArrayInputStream, не прибегая к буферизированному потоку. Класс BufferedInputStream просто оптимизирует производительность при работе с потоком ByteArrayInputStream. Естественно вместо ByteArrayInputStream может использоваться любой другой класс, который унаследован от InputStream.

**Класс BufferedOutputStream**

Класс BufferedOutputStream аналогично создает буфер для потоков вывода. Этот буфер накапливает выводимые байты без постоянного обращения к устройству. И когда буфер заполнен, производится запись данных.

BufferedOutputStream определяет два конструктора:

BufferedOutputStream(OutputStream outputStream)

BufferedOutputStream(OutputStream outputStream, int bufSize)

Первый параметр - это поток вывода, который унаследован от OutputStream, а второй параметр - размер буфера.

Рассмотрим на примере записи в файл:

import java.io.\*;

public class Program {

    public static void main(String[] args) {

        String text = "Hello world!"; // строка для записи

        try(FileOutputStream out=new FileOutputStream("notes.txt");

                BufferedOutputStream bos = new BufferedOutputStream(out))

        {

            byte[] buffer = text.getBytes(); // перевод строки в байты

            bos.write(buffer, 0, buffer.length);

        }

        catch(IOException ex) { System.out.println(ex.getMessage()); }

    }

}

Класс BufferedOutputStream в конструкторе принимает в качестве параметра объект OutputStream - в данном случае это файловый поток вывода FileOutputStream. И также производится запись в файл. Опять же BufferedOutputStream не добавляет много новой функциональности, он просто оптимизирует действие потока вывода.

**19.5. Java — Класс File, его конструкторы и методы**

В Java класс File представляет файлы и пути каталогов в абстрактной манере. Этот класс используется для создания, поиска, удаления файлов и каталогов и т.д.

**Конструкторы**

File-объект представляет собой фактический файл или каталог на диске. Ниже приведен список конструкторов для создания file-объекта.

|  |  |
| --- | --- |
| **№** | **Конструктор и описание** |
| 1 | **File(File parent, String child)** Конструктор создает новый экземпляр файла из родительского абстрактного пути и строки дочернего пути. |
| 2 | **File(String pathname)** Конструктор создает новый экземпляр файла, преобразовывая указанную строку pathname в абстрактный путь. |
| 3 | **File(String parent, String child)** Конструктор создает новый экземпляр файла из строки родительского пути и строки дочернего пути. |
| 4 | **File(URI uri)** Конструктор создает новый экземпляр файла, преобразовывая данный файл: URI в абстрактный путь. |

**Методы**

Если Вы используйте file-объекты, то у Вас есть под рукой список вспомогательных методов, которые можно использовать для управления файлами.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public String getName()** Возвращает имя файла или каталога, по указанному абстрактному имени пути. |
| 2 | **public String getParent()** Возвращает строковый путь родителя абстрактного пути, или null, если путь не указывает родительский каталог. |
| 3 | **public File getParentFile()** Возвращает абстрактный путь родителя абстрактного пути, или null, если путь не указывает родительский каталог. |
| 4 | **public String getPath()** Преобразует абстрактный путь в строку. |
| 5 | **public boolean isAbsolute()** Провяет, является ли абстрактный путь абсолютным. Возвращает true, если абстрактный путь является абсолютным, иначе false. |
| 6 | **public String getAbsolutePath()** Возвращает строкой абсолютный путь. |
| 7 | **public boolean canRead()** Проверяет, может ли приложение прочитать файл, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда файл, указанный в абстрактном пути, существует и может быть прочитан приложением; в противном случае false. |
| 8 | **public boolean canWrite()** Проверяет, может ли приложение изменять файл, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда файловая система фактически содержит файл, по указанному абстрактному имени пути, и приложению разрешено записывать в файл; в противном случае false. |
| 9 | **public boolean exists()** Проверяет, существует ли файл или каталог, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда существует файл или каталог, по указанному абстрактному имени пути; в противном случае false. |
| 10 | **public boolean isDirectory()** Проверяет, является ли файл, по указанному абстрактному имени пути, каталогом. Возвращает true тогда и только тогда, когда файл, обозначенный этим абстрактным именем, существует и является каталогом; в противном случае false. |
| 11 | **public boolean isFile()** Проверяет, является ли файл, по указанному абстрактному имени пути, нормальным файлом. Файл является нормальным, если он не является каталогом и, кроме того, удовлетворяет другим системным критериям. Любой файл без каталога, созданный приложением Java, гарантированно является нормальным файлом. Возвращает true тогда и только тогда, когда файл, обозначенный этим абстрактным пустым именем, существует и является нормальным файлом; в противном случае false. |
| 12 | **public long lastModified()** Возвращает время последнего изменения файла, по указанному абстрактному имени пути. Возвращает длинное значение, представляющее время последнего изменения файла, измеренное в миллисекундах с эпохи (00:00:00 GMT, 1 января 1970 г.) или 0L, если файл не существует или возникает ошибка ввода-вывода. |
| 13 | **public long length()** Возвращает длину файла, обозначенного этим абстрактным именем пути. Возвращаемое значение не указано, если путь указывает на каталог. |
| 14 | **public boolean createNewFile() throws IOException** Атомарно создает новый пустой файл, названный этим абстрактным именем пути, тогда и только тогда, когда файл с этим именем еще не существует. Возвращает true, если названный файл не существует и был успешно создан; false, если именованный файл уже существует. |
| 15 | **public boolean delete()** Удаляет файл или каталог, по указанному абстрактному имени пути. Если это имя пути обозначает каталог, каталог должен быть пустым, чтобы его можно было удалить. Возвращает true тогда и только тогда, когда файл или каталог успешно удалены; в противном случае false. |
| 16 | **public void deleteOnExit()** Просит, чтобы файл или каталог, обозначенные данным абстрактным пустым именем, были удалены при завершении работы виртуальной машины. |
| 17 | **public String[] list()** Возвращает массив строк, называющий файлы и каталоги в каталоге, обозначаемом этим абстрактным именем пути. |
| 18 | **public String[] list(FilenameFilter filter)** Возвращает массив строк, называющий файлы и каталоги в каталоге, обозначаемом этим абстрактным пустым именем пути, которое удовлетворяет указанному фильтру. |
| 19 | **public File[] listFiles()** Возвращает массив абстрактных путей, обозначающих файлы в каталоге, обозначаемом этим абстрактным именем пути. |
| 20 | **public File[] listFiles(FileFilter filter)** Возвращает массив абстрактных путей, обозначающих файлы и каталоги в каталоге, обозначаемом этим абстрактным пустым именем пути, которое удовлетворяет указанному фильтру. |
| 21 | **public boolean mkdir()** Создает каталог с именем этого абстрактного пути. Возвращает true тогда и только тогда, когда каталог был создан, в противном случае false. |
| 22 | **public boolean mkdirs()** Создает каталог с именем этого абстрактного пути, включая любые необходимые, но несуществующие родительские каталоги. Возвращает true тогда и только тогда, когда каталог был создан вместе со всеми необходимыми родительскими каталогами, в противном случае false. |
| 23 | **public boolean renameTo(File dest)** Переименовывает файл, обозначенный этим абстрактным пустым именем. Возвращает true тогда и только тогда, когда переименование выполнено успешно, в противном случае false. |
| 24 | **public boolean setLastModified(long time)** Устанавливает последней модификацией время файла или каталога с именем этого абстрактного пути. Возвращает true тогда и только тогда, когда операция выполнена успешно, в противном случае false. |
| 25 | **public boolean setReadOnly()** Помечает файл или каталог с именем этого абстрактного пути, чтобы разрешить только операции чтения. Возвращает true тогда и только тогда, когда операция выполнена успешно, в противном случае false. |
| 26 | **public static File createTempFile(String prefix, String suffix, File directory) throws IOException** Создает новый пустой файл в указанном каталоге, используя строковые prefix и suffix, чтобы сгенерировать его имя. Возвращает абстрактный путь, обозначающий вновь созданный пустой файл. |
| 27 | **public static File createTempFile(String prefix, String suffix) throws IOException** Создает пустой файл в каталоге временных файлов по умолчанию, используя данные prefix и suffix, чтобы сгенерировать его имя. Вызов этого метода эквивалентен вызову createTempFile(prefix, suffix, null). Возвращает абстрактное имя пути, обозначающее вновь созданный пустой файл. |
| 28 | **public int compareTo(File pathname)** Сравнивает лексикографически два абстрактных пути. Возвращает ноль, если аргумент равен этому абстрактному пути, значение меньше нуля, если это абстрактное имя пути лексикографически меньше аргумента или значение больше нуля, если этот абстрактный путь лексикографически больше аргумента. |
| 29 | **public int compareTo(Object o)** Сравнивает этот абстрактный путь с другим объектом. Возвращает ноль, если аргумент равен этому абстрактному пути, значение меньше нуля, если это абстрактное имя пути лексикографически меньше аргумента или значение больше нуля, если этот абстрактный путь лексикографически больше аргумента. |
| 30 | **public boolean equals(Object obj)** Проверяет этот абстрактный путь на равенство с данным объектом. Возвращает true тогда и только тогда, когда аргумент не является нулевым и представляет собой абстрактный путь, который обозначает тот же файл или каталог, что и этот абстрактный путь. |
| 31 | **public String toString()** Возвращает строковый путь этого абстрактного пути. Это просто строка, возвращаемая методом getPath(). |

**Каталоги в Java**

В Java каталог представлен Файлом, который может содержать список других файлов и каталогов. Используя объект File, вы можете создать каталог, прокрутить список файлов, представленных в каталоге. Для получения более детальных сведений, ознакомьтесь с перечнем всех методов, которые могут быть вызваны из объекта File, будучи связанными с каталогами.

**Создание каталогов**

Существуют два служебных метода File, которые могут быть использованы для создания каталогов:

* Метод mkdir() позволяет создать папку в Java, возвращая значение true при успехе операции, и false в случае сбоя. Сбой свидетельствует о том, что путь указанный в объекте File уже существует, либо что каталог не может быть создан в связи с тем, что полный путь еще не существует.
* Метод mkdirs() создает каталог и все вышестоящие каталоги.

В следующем примере представлено создание папки "/java/proglang/newdir":

**Пример**

import java.io.File;

public class CreateDirectory {

public static void main(String args[]) {

String nameDir = "/java/proglang/newdir";

File a = new File(nameDir);

// Создание на диске папки и всех вышестоящих каталогов

a.mkdirs();

}

}

Скомпилируйте и выполните следующий код для создания каталога "/java/proglang/newdir".

Примечание ? Java автоматически формирует разделители пути в UNIX и Windows с учетом соглашений. При использовании косой черты (/) при работе с Java в системе Windows, производится корректное разрешение пути.

**Список файлов в папке**

Метод list(), представленный объектом File, может быть использован для предоставления перечня всех файлов и каталогов, имеющихся в заданной папке, в следующем виде:

**Пример**

import java.io.File;

public class ReadDirectory {

public static void main(String[] args) {

File pathDir = null;

String[] pathsFilesAndDir;

try {

// Создание нового объекта file

pathDir = new File("/NetBeans 8.2/Projects/ReadDirectory/ReadDirectory/"); // Обязательно должен существовать указанный каталог на диске, иначе программа выдаст ошибку

// Массив файлов и папок

pathsFilesAndDir = pathDir.list();

for(String path:pathsFilesAndDir) {

// Вывод списка файлов и каталогов

System.out.println(path);

}

}catch(Exception e) {

// Если произошла ошибка

e.printStackTrace();

}

}

}

Вследствие этого будет получен следующий результат, основанный на каталогах и файлах, доступных в вашем каталоге /NetBeans 8.2/Projects/ReadDirectory/ReadDirectory/:

build

build.xml

manifest.mf

nbproject

src

ReadDirectory

**19.6.1 Java — Класс FileReader, его конструкторы и методы, пример записи и чтения файла**

Класс FileReader наследуется от класса InputStreamReader. FileReader используется для чтения потоков символов.

**Конструкторы**

Он имеет несколько конструкторов для создания нужных объектов. Ниже приведен список конструкторов, предоставляемых классом FileReader.

|  |  |
| --- | --- |
| **№** | **Конструктор и описание** |
| 1 | **FileReader(File file)** Конструктор создает новый FileReader с учетом файла, который требуется прочитать. |
| 2 | **FileReader(FileDescriptor fd)** Конструктор создает новый FileReader, с учетом FileDescriptor для чтения. |
| 3 | **FileReader(String fileName)** Конструктор создает новый FileReader, учитывая имя файла для чтения. |

**Методы**

Когда у Вас есть объект FileReader, тогда есть список вспомогательных методов, которые можно использовать для управления файлами.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public int read() throws IOException** Читает один символ. Возвращает int, который представляет прочитанный символ. |
| 2 | **public int read(char [] c, int offset, int len)** Читает символы в массив. Возвращает количество прочитанных символов. |

**19.6.2 Java — Класс FileWriter, его конструкторы и методы, пример записи и чтения файла**

Класс FileWriter наследуется от класса OutputStreamWriter. Класс используется для записи потоков символов.

**Конструкторы**

Класс FileWriter имеет несколько конструкторов для создания требуемых объектов. Ниже приведен список.

|  |  |
| --- | --- |
| **№** | **Конструктор и описание** |
| 1 | **FileWriter(File file)** Конструктор создает объект FileWriter для объекта File. |
| 2 | **FileWriter(File file, boolean append)** Конструктор создает объект FileWriter, заданный объектом File с логическим значением, указывающим, следует ли добавлять записанные данные. |
| 3 | **FileWriter(FileDescriptor fd)** Конструктор создает объект FileWriter, связанный с данным файловым дескриптором. |
| 4 | **FileWriter(String fileName)** Конструктор создает объект FileWriter, учитывая имя файла. |
| 5 | **FileWriter(String fileName, boolean append)** Конструктор создает объект FileWriter с именем файла с логическим значением, указывающим, следует ли добавлять записанные данные. |

**Методы**

Когда у вас есть объект FileWriter, тогда есть список вспомогательных методов, которые можно использовать для управления файлами.

|  |  |
| --- | --- |
| **№** | **Метод и описание** |
| 1 | **public void write(int c) throws IOException** Записывает один символ. |
| 2 | **public void write(char [] c, int offset, int len)** Записывает часть массива символов, начиная с offset, длиной len. |
| 2 | **public void write(String s, int offset, int len)** Напишите часть строки, начиная со смещения и длины len. |

**Пример в Java чтения и записи в текстовый файл**

Ниже приведен пример демонстрации класса (FileReader class) в Java - запись и чтение из файла:

import java.io.\*;

public class Test {

public static void main(String args[])throws IOException {

File file = new File("Example.txt");

// Создание файла

file.createNewFile();

// Создание объекта FileWriter

FileWriter writer = new FileWriter(file);

// Запись содержимого в файл

writer.write("Это простой пример,\n в котором мы осуществляем\n с помощью языка Java\n запись в файл\n и чтение из файла\n");

writer.flush();

writer.close();

// Создание объекта FileReader

FileReader fr = new FileReader(file);

char [] a = new char[200]; // Количество символов, которое будем считывать

fr.read(a); // Чтение содержимого в массив

for(char c : a)

System.out.print(c); // Вывод символов один за другими

fr.close();

}

}

Получим следующий результат:

Это простой пример,

в котором мы осуществляем

с помощью языка Java

запись в файл

и чтение из файла

Содержимое созданного текстового файла Example.txt:

