



InputStream/OutputStream

Java Core (/quests/QUEST_JAVA_CORE)
8 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=8), 4 лекция (/quests/lectures/questcore.level08.lecture04)

ОТКРЫТА

— Привет, Амиго! Сегодня мы снова будем заниматься разбором работы InputStream и OutputStream. На самом деле, то первое объяснение было немного упрощенным. Это не интерфейсы, а абстрактные классы, и они даже имеют по паре реализованных методов. Давай посмотрим, какие методы у них есть:

Методы класса InputStream

Что метод делает

```
1 int read(byte[] buff);
```

— метод сразу читает блок байт в **буфер** (**массив байт**), пока **буфер** не заполнится или не закончатся байты там, откуда он их читает. Метод возвращает количество реально прочитанных байт (оно может быть меньше длины массива)

```
1 int read();
```

— метод читает один байт и возвращает его как результат. Результат расширяется до int, для красоты. Если все байты уже прочитаны, метод вернет «-1».

```
1 int available();
```

— метод возвращает количество непрочитанных (доступных) байт.

```
1 void close();
```

— метод «закрывает» поток — вызывается после окончания работы с потоком. Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д. Из потока больше нельзя читать данные.

— Т.е. мы можем читать не только по одному байту, а и целыми блоками?

— Да.

— А записывать целыми блоками тоже можно?

— Да, вот смотри:

Методы OutputStream

Что метод делает

```
1 void write(int c);
```

— метод записывает один байт информации. Тип `int` сужается до `byte`, лишняя часть просто отбрасывается.

```
1 void write(byte[] buff);
```

— метод записывает блок байт.

```
1 void write(byte[] buff, int from, int count);
```

— метод записывает часть блока байт. Используется в случаях, когда есть вероятность, что блок данных был заполнен не целиком

```
1 void flush();
```

— если есть данные, которые хранятся где-то внутри и еще не записаны, то они записываются.

```
1 void close();
```

— метод «закрывает» поток — вызывается после окончания работы с потоком.

Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д. В поток больше нельзя писать данные, `flush` при этом вызывается автоматически.

— А как будет выглядеть код копирования файла, если мы будем читать не по одному байту, а целыми блоками?

— Гм. Примерно так:

Копируем файл на диске

```
1 public static void main(String[] args) throws Exception
2 {
3     //Создаем поток-чтения-байт-из-файла
4     FileInputStream inputStream = new FileInputStream("c:/data.txt");
5     // Создаем поток-записи-байт-в-файл
6     FileOutputStream outputStream = new FileOutputStream("c:/result.txt");
7
8     byte[] buffer = new byte[1000];
9     while (inputStream.available() > 0) //пока есть еще непрочитанные байты
10    {
11        // прочитать очередной блок байт в переменную buffer и реальное количество в count
12        int count = inputStream.read(buffer);
13        outputStream.write(buffer, 0, count); //записать блок(часть блока) во второй поток
14    }
15
16    inputStream.close(); //закрываем оба потока. Они больше не нужны.
17    outputStream.close();
18 }
```

— С буфером все понятно, а что это за переменная `count`?

— Когда мы читаем самый последний блок данных в файле, может оказаться, что байт осталось не 1000, а, скажем, 328. Тогда и при записи нужно указать, что записать не весь блок, а только его первые 328 байт.

Метод `read` при чтении последнего блока вернет значение равное количеству реально прочитанных байт. Для всех чтений – 1000, а для последнего блока – 328.

Поэтому при записи блока мы указываем, что нужно записать не все байты из буфера, а байты с 0 по 328 (т.е. значение, хранимое в переменной `count`).

— Теперь понятно, как все это работает. Спасибо, Элли.

[< \(/quests/lectures/questcore.level08.lecture03\)](/quests/lectures/questcore.level08.lecture03)

[×18 > \(/quests/lectures/questcore.level08.lecture05\)](/quests/lectures/questcore.level08.lecture05)

Комментарии (26)

популярные

новые

старые

Никита

rotarru 20 уровень, Минск

8 февраля, 06:12

[Побайтовая работа с файлами](#)

Кто ещё не почитал статью по ссылке, почитайте. И ответьте, кали ласка, на мой вопрос. Почему

```
1 int read(byte[] buff);
```

работает в тысячу раз быстрее, чем

```
1 int read();
```

Это как-то связано с тем, что чтение передаётся через нативный метод сишной библиотеке? Где возникает узкое место, когда читаем побайтово? Может за каждым байтом сишная библиотека каждый раз обращается к винчестеру отдельно, потому что она не знает, что read() в джаве стоит у нас в цикле?

И ещё. BufferedInputStream улучшает ситуацию с read() по-байтово не намного. По времени в два раза только на моём замере. Тогда зачем этот буфер нужен, если можно прочитать сразу массив байтов в тысячу раз быстрее.

Ответить

0

xeturg88 25 уровень, Нижний Новгород

12 февраля, 18:32

В статье же написано

"Память ОЗУ (DRAM) где обычно выполняется программа и хранятся переменные имеет высокую скорость доступа, но небольшой размер.

Память на жестком/flash диске (HDD или Flash-накопители) где обычно хранятся файлы, наоборот имеет низкую скорость доступа, но большой размер.

Так что когда мы побайтно читаем 1Gb файл (то есть миллиард раз обращаемся к HDD) — мы тратим много времени на работу с низкоскоростным устройством"

Т.е. пока жесткий диск найдет нужный байт, пока его считает, пока поместит в регистры. Очень много накладных расходов. Это все равно, что ходить за продуктами для салата в магазин за три километра, покупать в нем один продукт и идти домой. Затем снова в магазин и так пока не купишь все продукты. Быстрее прийти и купить все сразу.

Ответить

+4

MaKaRi4 28 уровень, Киев

6 января, 14:15

Разъясните пожалуйста эту строчку, кто знает как она работает:

```
byte[] buffer = new byte[in.available()];
```

Ответить

0

Дмитрий Дедков 26 уровень

6 января, 14:36

создаётся массив байт с размером, равным количеству доступных байт в in (что такое in, я ещё не знаю до конца) (и потом, видимо, все эти байты полностью заполняют массив при записи в него)

Ответить

0

MaKaRi4 28 уровень, Киев

7 января, 12:51

in это название потока, например:

```
FileInputStream in = new FileOutputStream();
```

"создаётся массив байт с размером, равным количеству доступных байт в in" - это то что я хотел узнать, спасибо!

Ответить

+1

Артур Пряхников 25 уровень, Одесса

16 декабря 2017, 15:04

"Метод read при чтении последнего блока вернет значение равное количеству реально прочитанных байт. Для всех чтений — 1000, а для последнего блока — 328."

Не совсем правильно. Метод не будет возвращать 1000 байт при каждом чтении. Количество байт почти везде будет разным.

Кому интересно - выводите на экран количество байтов, которое передается за один раз.

Ответить

+1

Макс Куркудюк 28 уровень, Киев

7 марта, 15:05

Це через те, що файл містить " ".

Ответить

0

Кирилл Кириллов 32 уровень, Екатеринбург

29 августа 2017, 07:26

Кто то писал что метод read() читает сразу два байта, тут пишут что один байт. Как на самом деле? Если это всего один байт то зачем все таки приводить к int, тут пишут "для красоты" что то меня не устраивает такой ответ. А если это все же два байта то как из двух байтов получается число

Ответить

0

Vikentsi 22 уровень, Минск

18 сентября 2017, 15:11

Если верить [документации](#), то read() читает один байт.
Приводить в int не приходится так как сам метод read() имеет возвращающей тип int (это вы можете увидеть в документации или в IDEA наставив курсор на метод и нажав Ctrl+B окажитесь прям в родительском классе в этом методе)
Слово для красоты не совсем тут приемлемо я с вами согласен.
Так как часто мы будем встречать файлы с числом байт выше 256 а следовательно в метод write(byte[] buff, int from, int count), который требует int будем давать неверное значение.

Ответить

+1

Anton Stezhkin 19 уровень

12 декабря 2017, 15:48

"Так как часто мы будем встречать файлы с числом байт выше 256" - так .read() возвращает ЗНАЧЕНИЕ прочитанного байта. А оно от 0 до 255. Так что всё описанное в примере срабатывает.

Ответить

0

Kirill 21 уровень, Санкт-Петербург

17 декабря 2017, 10:52

byte от -128 до 127
вы хотите применить byte к char со значением от 0 до 255?

Ответить

0

Anton Stezhkin 19 уровень

24 декабря 2017, 12:55

да ладно! всё прекрасно конвертится.

Ответить

0

Anton Stezhkin 19 уровень

1 января, 23:38

+128 при записи в массив и -128 при чтении инфы. :)

Ответить

0

Anton Stezhkin 19 уровень

1 января, 23:39

или маску если сильно умным быть.
<http://info.javarush.ru/Joysi/2016/02/18/%D0%9F%D0%BE%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D0%B0%D1%8F-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D1%84%D0%B0%D0%B9%D0%BB%D0%B0%D0%BC%D0%B8.html>

Ответить

+6

Kirill 21 уровень, Санкт-Петербург

5 января, 21:17

спасибо за ссылку, полезная статья!

Ответить

0

Александр Орлов 18 уровень, Орёл

пятница, 23:22

Огромное спасибо за статью. Крайне интересно.

Ответить

0

Юрий 35 уровень, Минск

27 августа 2017, 19:52

Разделение файла - если считывать имена файлов в потоках, а не отдельно в виде строковых значений - не проходит валидацию, хотя с консоли считываются именно три имени файлов.

Ответить

0

Vra 40 уровень

22 августа 2017, 23:19

void write(byte b[], int off, int len) записывает в поток часть массива len байтов, начиная с элемента b[off].

Ответить

0

kto_to 35 уровень

24 июня 2017, 17:10

"Метод read при чтении последнего блока вернет значение равное КОЛИЧЕСТВУ реально прочитанных байт.", а массив как известно начинается с нуля. Разве не до count-1 надо идти? Если кол-во записанных байтов два (например такие: 24 и 87), то count = 2(т.е. кол-ву), а buffer[0]=24 и buffer[1] = 87. Не?

Ответить

+1

Сергей Зимовец 31 уровень

1 июля 2017, 16:07

если не ошибаюсь в метод последним значение передается не индекс а кол-во, то есть читай с бвффера . начиная с 0 индекса 2 байта!!!!

Буферы, на которые выводится в консоль.

Ответить

0

Николай Ковчуга 28 уровень, Минск

14 июля 2017, 10:39

нет, все верно написано. Там количество, а не конечный индекс.

Ответить

0

Джонни Мнемоник 24 уровень

20 сентября 2017, 13:03

Будьте спокойны в методе read() все отнимут за вас.

Ответить

+3

Стасик Томилов 19 уровень

20 марта 2017, 20:41

Не совсем понятен следующий момент:

```
byte[] buffer = new byte[1000];
while (inputStream.available() > 0) //пока есть еще непрочитанные байты
{
    // прочитать очередной блок байт в переменную buffer и реальное количество в count
    int count = inputStream.read(buffer);
    outputStream.write(buffer, 0, count); //записать блок(часть блока) во второй поток
}
```

Как метод, не возвращая нам массив производит запись в переменную buffer? она ведь не статическая.

Ответить

0

Петр 22 уровень

17 апреля 2017, 12:49

Сделай в коде System.out.println(buffer); и увидишь, что массив, это ссылка "[B@1540e19d". То есть в функцию массив (даже примитивных чисел), в отличии от примитивного типа, уходит по ссылке, а не копируется!

То есть примитивные типы, если ничего не путаю, хранятся в СТЕКЕ, а всё остальное в КУЧЕ. Всё, что получается из КУЧИ, приходит ссылкой на то, где в куче это лежит, а из СТЕКА возвращается значение.

Ответить

+4

victor 30 уровень, Минск

18 апреля 2017, 00:35

я сразу тоже не очень понял, как заполняется buffer, потом дошло, что это сам метод другой. inputStream.read(buffer) будучи записан с параметром, производит запись в указанный массив до упора. а далее работаем с полученным блоком.

Ответить

0

Максим Караваев 40 уровень, Санкт-Петербург

14 сентября 2017, 15:41

Объекты передаются в методы по ссылке, а не по значению, значит при изменении объекта в методе объект меняется везде. И после метода read(buffer) мы имеем заполненный массив.

Ответить

0

javarush.ru/ [G+ \(https://plus.google.com/114772402300089087607/\)](https://plus.google.com/114772402300089087607/) [Twitter \(https://twitter.com/javarush_ru\)](https://twitter.com/javarush_ru) [Facebook \(https://www.facebook.com/javarush.ru\)](https://www.facebook.com/javarush.ru)



Программистами не рождаются
© 2018