

BufferedReader, BufferedWriter

Java Collections
2 уровень, 5 лекция

ОТКРЫТА

— И это снова я.

— Привет, Элли!

— Сегодня я хочу тебе подробно рассказать про **BufferedReader** и **BufferedWriter**.

— Так ты мне уже рассказывала все про них. Ну ничего там сложного нет.

— Ок. Расскажи, как работает **BufferedReader**.

— **BufferedReader** — это как переходник в розетке с 110 к 220 вольт.

В конструктор объекта **BufferedReader** обязательно нужно передать объект **Reader**, из которого он будет читать данные. Объект **BufferedReader** читает из **Reader**'а данные большими кусками и хранит их у себя внутри в буфере. Поэтому чтение из пары **BufferedReader+Reader** быстрее, чем прямо из **Reader**.

— Верно. А **BufferedWriter**?

— Тут тоже все просто. Когда мы пишем в **FileWriter**, например, то данные сразу записываются на диск. Если мы часто пишем небольшие данные, то происходит много обращений к диску, что замедляет работу программы. А если мы используем **BufferedWriter** в качестве «переходника», то операция записи на диск ускорится. **BufferedWriter**, при записи в него, сохраняет переданные данные во внутреннем буфере, а когда буфер заполняется – пишет данные во **Writer** одним большим куском. Это гораздо быстрее.

— Гм. Все верно. А что ты забыл?

— После окончания записи у объекта **BufferedWriter** надо вызвать метод **flush()**, чтобы он записал данные из буфера во **Writer**, которые еще не записаны, т.е. буфер не заполнен до конца.

— А кроме того?

— А кроме того, пока буфер еще не записан во **Writer**, данные можно удалить и/или заменить на другие.

— Амиго! Я поражена! Да ты просто эксперт. Ладно, тогда я расскажу тебе о новых классах: **ByteArrayStream**, **PrintStream**.

Итак, **ByteArrayInputStream** и **ByteArrayOutputStream**.

Эти классы по сути чем-то похожи на **StringReader** и **StringWriter**. Только **StringReader** читал символы (**char**) из строки (**String**), а **ByteArrayInputStream** читает байты из массива байт (**ByteArray**).

StringWriter писал символы (**char**) в строку, а **ByteArrayOutputStream** пишет байты в массив байт у него внутри. При записи в **StringWriter** строка внутри него удлинялась, а при записи в **ByteArrayOutputStream** его внутренний массив байт тоже динамически расширяется.

Вот пример, который выводит в консоль полученную строку:

Чтение из объекта reader и запись в объект writer:

```
1 public static void main (String[] args) throws Exception
2 {
3     String test = "Hi!\n Mv name is Richard\n I'm a photographer\n":
```

НАЧАТЬ ОБУЧЕНИЕ

```
6      StringWriter writer = new StringWriter();
7
8      executor(reader, writer);
9
10     String result = writer.toString();
11
12     System.out.println("Результат: "+result);
13 }
14
15 public static void executor(Reader reader, Writer writer) throws Exception
16 {
17     BufferedReader br = new BufferedReader(reader);
18     String line;
19     while ((line = br.readLine()) != null) {
20         writer.write(line + '\n');
21     }
22 }
```

Вот как он будет выглядеть, если тут работать не с символами, а с байтами:

Чтение из объекта InputStream и запись в объект OutputStream:

```
1  public static void main (String[] args) throws Exception
2  {
3      String test = "Hi!\n My name is Richard\n I'm a photographer\n";
4      InputStream inputStream = new ByteArrayInputStream(test.getBytes());
5
6      ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
7
8      executor(inputStream, outputStream);
9
10     String result = new String(outputStream.toByteArray());
11     System.out.println("Результат: "+result);
12 }
13
14 public static void executor(InputStream inputStream, OutputStream outputStream) throws Exception
15 {
16     BufferedInputStream bis = new BufferedInputStream(inputStream);
17     while (bis.available() > 0)
18     {
19         int data = bis.read();
20         outputStream.write(data);
21     }
22 }
```

Тут все аналогично примеру выше. Вместо String – ByteArray. Вместо Reader – InputStream, вместо Writer – OutputStream.

Единственные еще два момента – это преобразование строки в массив байт и обратно. Как ты видишь, это делается довольно несложно:

Преобразование строки в массив байт и обратно

```
1  public static void main (String[] args) throws Exception
2  {
```

```
5
6     String result = new String(array);
7     System.out.println("Результат: "+result);
8 }
```

Чтобы получить байты, которые уже добавлены в `ByteArrayOutputStream`, надо вызвать метод **toByteArray()**.

— Ага. Аналогия с `StringReader/StringWriter` довольно сильная, особенно когда ты мне ее показала. Спасибо, Элли, действительно интересный урок.

— Куда это ты спешишь? У меня есть еще небольшой подарок – хочу рассказать тебе про класс `PrintStream`.

— `PrintStream`? В первый раз слышу о таком классе.

— Ага. Особенно, если не считать, что ты им пользуешься с первого дня, когда ты начал изучать Java. Помнишь **System.out**? так вот – **System.out** – это статическая переменная класса **System** типа... **PrintStream**! Именно оттуда растут ноги всех этих `print`, `println` и т.д.

— Ого. Как интересно. Я как-то ни разу и не задумывался. Расскажи подробнее.

— Гуд. Тогда слушай. Класс `PrintStream` был придуман для читабельного вывода информации. Он практически весь состоит из методов `print` и `println`. См. таблицу:

Методы	Методы
<code>void print(boolean b)</code>	<code>void println(boolean b)</code>
<code>void print(char c)</code>	<code>void println(char c)</code>
<code>void print(int c)</code>	<code>void println(int c)</code>
<code>void print(long c)</code>	<code>void println(long c)</code>
<code>void print(float c)</code>	<code>void println(float c)</code>
<code>void print(double c)</code>	<code>void println(double c)</code>
<code>void print(char[] c)</code>	<code>void println(char[] c)</code>
<code>void print(String c)</code>	<code>void println(String c)</code>
<code>void print(Object obj)</code>	<code>void println(Object obj)</code>
<code></code>	<code>void println()</code>
<code>PrintStream format (String format, Object ... args)</code>	
<code>PrintStream format (Locale l, String format, Object ... args)</code>	

Также есть несколько методов `format`, чтобы можно было выводить данные на основе шаблона. Пример:

Преобразование строки в массив байт и обратно

```
1     String name = "Kolan";
2     int age = 25;
3     System.out.format("My name is %s. My age is %d.", name, age);
```

My name is Kolan. My age is 25.

— Ага, помню, мы уже когда-то разбирали метод format у класса String.

— На этом все.

— Спасибо, Элли.

< Предыдущая лекция

🌌 x22

− +77 +

Комментарии (69)

популярные новые старые

JavaCoder

Введите текст комментария



Роман Кончалов Уровень 28, Россия EXPERT 24 января, 23:12 ⋮

пока буфер еще не записан во Writer, данные можно удалить и/или заменить на другие
Это как? Записать данные через другой объект, а потом закрыть ресурс?

Ответить 0



LuneFox инженер по сопровождению в BIFIT EXPERT 22 декабря 2021, 12:34 ⋮

Вот этот момент интересует:

```
1 while (bis.available() > 0)
2 {
3     int data = bis.read();
4     outputStream.write(data);
5 }
```

Что хранится в data?
Если одно интовое значение, то в чём смысл буфера, если мы перекачиваем данные по одному инту?

Второй момент:

"При записи в ByteArrayOutputStream его внутренний массив байт тоже динамически расширяется."

Внутренний массив, получается, представлен не в виде byte[]? Потому что, насколько я помню, примитивные массивы не могут менять размер после создания. Или каждый раз, когда нужно расширить массив, создаётся новый и в него переписывается всё из старого? А как же тогда производительность?

Ответить +1

Stepan Уровень 27, Москва, Россия 24 декабря 2021, 20:29 ⋮

По моему мнению:
1. BufferedInputStream.read() отдает данные из буфера, не нагружая или не дожидаясь источник потока данных, т.е. он заполняет свой буфер до выполнения BufferedInputStream.read(). При чтении из буфера, высвобожденное место занимает новыми данными из источника данных.
2. Внутренний массив ByteArrayOutputStream увеличивается при необходимости, ЕМНИП,кратно 2, а изначально 8192 байт. Да, создается новый и туда копируются данные. Производительность, как минимум, зависит от скорости данных к потребителю (запись в файл, передача по сети, иная обработка данных), размера этих данных, начального размера буфера.

Ответить 0

LuneFox инженер по сопровождению в BIFIT EXPERT 25 декабря 2021, 17:40 ⋮

То есть, BIS читает сразу полный буфер, а отдаёт нам по 1 байтику? И когда истощается, зачёрпывает ещё полный буфер?

Ответить 0

Stepan Уровень 27, Москва, Россия 26 декабря 2021, 12:10 ⋮

НАЧАТЬ ОБУЧЕНИЕ

Ответить

fedyaka

Уровень 36, Кострома, Россия

2 февраля, 11:51

Да, он берёт информацию крупными кусками, а после по запросу отдаёт её сколько нужно из буфера.

Ответить

0

Ars

Уровень 41

17 ноября 2021, 16:18

Где-то в середине статьи (а точнее в первом примере в методе public static void executor(Reader reader, Writer writer) throws Exception) забыли закрыть тег жирного шрифта.

Ответить

+1

illuminati

Уровень 31, Санкт-Петербург

3 августа 2021, 15:56

Система подачи материала мое почтение

Ответить

+7

misha_lazarev

Уровень 34, Ростов на Дону, Russian Federation

6 июня, 15:30

Согласен, особенно когда происходит разбирательство, не в диалоге сверху, а в комментариях.

Ответить

0

Михаил Ершов

Уровень 41

27 июня 2021, 21:29

Может кто-то знает в чем разница между System.out.printf и System.out.format?

1

String name = "Oleg";

2

int age = 25;

3

System.out.printf("My name is %s and i'm %d years old.\n", name, age);

4

//My name is Oleg and i'm 25 years old.

5

System.out.format("My name is %s and i'm %d years old.\n", name, age);

6

//My name is Oleg and i'm 25 years old.

Ответить

+1

Игорь

Full Stack Developer в IgorApplications

21 июля 2021, 16:32

Разницы нет, printf вызывает метод format во внутренней реализации, сделано для си программистов

Ответить

+1

Ян

Уровень 41, Лида, Беларусь

23 сентября 2021, 20:06

1

Разница есть.

2

Чтобы перенестись на новую строку в методе printf() мы должны

3

написать "%n", тогда как в format() нужно использовать "\n"

Ответить

+6

мистер т

Уровень 35, Москва

9 апреля 2021, 02:42

— А кроме того, пока буфер еще не записан во Writer, данные можно удалить и/или заменить на другие. Что имеется ввиду?

Ответить

+1

Anonymous #2489173

Уровень 35

25 марта 2021, 07:05

— PrintStream? В первый раз слышу о таком классе.

но ведь он уже был. И байт стрим был в какой-то из задач ранее. Очень сбивают такие вот приписки авторов.

Ответить

+3

Ira Tsygarova

Уровень 36, Санкт-Петербург, Россия

9 февраля 2021, 00:28

1

My name is Kolan. My age is 25.

Я одна прочитала "Колян"?

Ответить

+5

Boarder

QA Automation Engineer в ЛАНИТ

11 февраля 2021, 22:17

НАЧАТЬ ОБУЧЕНИЕ

Николай Т. Уровень 40, Рязань, Россия

28 июня 2021, 01:50 ⋮

Я пишу Kolyan :)

Ответить

−

0

+

Pig Man Главная свинья в Свинарнике

4 февраля 2021, 14:36 ⋮

"После окончания записи у объекта *BufferedWriter* надо вызвать метод *flush()*, чтобы он записал данные из буфера во *Writer*, которые еще не записаны"

Никогда не вызывал данный метод и все всегда прекрасно записывалось, но поискав в интернете нашел следующее:

Проблема возникает, когда запись буферизуемого потока прерывается (возможно, ошибкой где-то в другом месте). Соответственно, не происходит нормального закрытия потока операцией *close()*. Часть данных остается в буфере записи и не попадает в точку назначения (файл, например). В *try with resources* его вызывают автоматически перед закрытием, однако при конструкции с *finally* нужно использовать *flush()*

Ответить

−

+4

+

Aleksey Уровень 51, Пермь, Россия

21 октября 2021, 21:48 ⋮

В документации к *BufferedReader*'у написано:
void close() Closes the stream, **flushing** it first.
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/BufferedWriter.html>
Обычно *flush()* и *close()* делают друг за другом. По логике написанного выше в момент после исполнения *flush()* ДО выполнения *close()* может прерваться поток?

Ответить

−

0

+

Herr Ives Уровень 30

26 января 2021, 16:26 ⋮

PrintStream? В первый раз слышу о таком классе

Ответить

−

+9

+

↺ Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

▼

НАЧАТЬ ОБУЧЕНИЕ



"Программистами не рождаются" © 2022 JavaRush