Luxoft
Компания
рейтинг 106,62

Профиль | 144 Блог | 0 Вакансии | 1,5k Подписчики

14 апреля в 21:35

Разработка → Шпаргалка Java программиста 7.1 Типовые задачи: Оптимальный путь преобразования InputStream в строку

Разработка веб-сайтов*, Программирование*, Java*, Блог компании Luxoft

У меня есть хобби: я собираю различные решения типовых задач в Java, которые нахожу в инете, и пытаюсь выбрать наиболее оптимальное по размеру/производительности/элегантности. В первую очередь по производительности. Давайте рассмотрим такую типовую задач, которые часто встречаются в программировании на Java как "преобразование InputStream в строку" и разные варианты её решения.

Посмотрим какие ограничения есть у каждого (требования подключения определенной библиотеки/определенной версии, корректная работа с unicode и т.д.). Английскую версию этой статьи можно найти в моем ответе на stackoverflow. Тесты в моем проекте на github.

Общее оглавление 'Шпаргалок'

Преобразование InputStream в строку (String)

Очень часто встречающая задача, давайте рассмотрим какими способами можно это сделать (их будет 11):

1. Используя **IOUtils.toString** из библиотеки `Apache Commons`. Один из самых коротких однострочников.

```
String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
```

2. Используя **CharStreams** из библиотеки `guava`. Тоже довольно короткий код.

```
try(InputStreamReader reader = new InputStreamReader(inputStream, Charsets.UTF_8)) {
    String result = CharStreams.toString(reader);
}
```

3. Используя `Scanner` (**JDK**). Решение короткое, хитрое, с помощью чистого JDK, но это скорее хак, который вынесет мозг тем кто о таком фокусе не знает.

```
try(Scanner s = new Scanner(inputStream).useDelimiter("\\A")) {
    String result = s.hasNext() ? s.next() : "";
}
```

4. Используя **Stream Api** с помощью `Java 8`. **Предупреждение**: Оно заменяет разные переносы строки (такие как `\r\n`) на `\n`, иногда это может быть критично.

```
try(BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
    String result = br.lines().collect(Collectors.joining("\n"));
}
```

5. Используя **parallel Stream Api** (`Java 8`). **Предупреждение**: Как и 4 решение, оно заменяет разные переносы строки (такие как `\r\n`) на `\n`.

```
try(BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
    String result = br.lines().parallel().collect(Collectors.joining("\n"));
}
```

6. Используя **InputStreamReader** и **StringBuilder** из обычного `JDK`

```
final int bufferSize = 1024;
final char[] buffer = new char[bufferSize];
final StringBuilder out = new StringBuilder();
try(Reader in = new InputStreamReader(inputStream, "UTF-8")) {
    for (; ; ) {
        int rsz = in.read(buffer, 0, buffer.length);
        if (rsz < 0)
             break;
        out.append(buffer, 0, rsz);
    }
    return out.toString();
}
```

7. Используя **StringWriter** и **IOUtils.copy** из `Apache Commons`

```
try(StringWriter writer = new StringWriter()) {
    IOUtils.copy(inputStream, writer, "UTF-8");
    return writer.toString();
}
```

8. Используя **ByteArrayOutputStream** и **inputStream.read** из `JDK`

```
try(ByteArrayOutputStream result = new ByteArrayOutputStream()) {
    byte[] buffer = new byte[1024];
    int length;
    while ((length = inputStream.read(buffer)) != -1) {
        result.write(buffer, 0, length);
    }
```

```
    return result.toString("UTF-8");
}
```

9. Используя **BufferedReader** из `JDK`. **Предупреждение**: Это решение заменяет разные переносы строк (такие как `\n\r`) на `line.separator` system property (например, в Windows на "\r\n").

```
String newLine = System.getProperty("line.separator");
try(BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream))) {
      StringBuilder result = new StringBuilder();
      String line; boolean flag = false;
      while ((line = reader.readLine()) != null) {
          result.append(flag? newLine: "").append(line);
          flag = true;
      }
     return result.toString();
}
```

10. Используя **BufferedInputStream** и **ByteArrayOutputStream** из `JDK`

```
try(BufferedInputStream bis = new BufferedInputStream(inputStream); ByteArrayOutputStream buf = new ByteArrayOutputStream()) {
        int result = bis.read();
        while(result != -1) {
            buf.write((byte) result);
            result = bis.read();
        }
        return buf.toString();
}
```

11. Используя **inputStream.read()** и **StringBuilder** (`JDK`). **Предупреждение**: Это решение не работает с Unicode, например с русским текстом

```
int ch;
StringBuilder sb = new StringBuilder();
while((ch = inputStream.read()) != -1)
    sb.append((char)ch);
reset();
return sb.toString();
```

Итак о использовании:

1. Решения `4`, `5` и `9` преобразую разные переносы строки в одну.
2. Решения `11` не работает с Unicode текстом
3. Решение `1`, `7` требует использование библиотеки **Apache Commons**, `2` требует библиотеку **Guava**, `4` и `5` требуют Java 8 и выше,

Замеры производительности

Предупреждение: замеры производительности всегда сильно зависят от системы, условий замера и т.п. Я измерял на двух разных компьютерах, один Windows 8.1, Intel i7-4790 CPU 3.60GHz*2, 16Gb, второй — Linux Mint 17.2, Celeron Dual-Core T3500 2.10Ghz*2, 6Gb, однако не могу гарантировать что результаты являются абсолютной истиной, вы всегда можете повторить тесты (test1 и test2) на вашей системе.

Замеры производительности для небольших строк (длина = 175), тесты можно найти на github (режим = среднее время выполнения (AverageTime), система = Linux Mint 17.2, Celeron Dual-Core T3500 2.10Ghz*2, 6Gb, чем значение ниже тем лучше, 1,343 — наилучшее):

```
              Benchmark                        Mode  Cnt   Score   Error  Units
8. ByteArrayOutputStream and read (JDK)        avgt   10   1,343 ± 0,028  us/op
6. InputStreamReader and StringBuilder (JDK)   avgt   10   6,980 ± 0,404  us/op
```

```
10.BufferedInputStream, ByteArrayOutputStream  avgt   10   7,437 ± 0,735  us/op
11.InputStream.read() and StringBuilder (JDK)  avgt   10   8,977 ± 0,328  us/op
7. StringWriter and IOUtils.copy (Apache)      avgt   10  10,613 ± 0,599  us/op
1. IOUtils.toString (Apache Utils)             avgt   10  10,605 ± 0,527  us/op
3. Scanner (JDK)                               avgt   10  12,083 ± 0,293  us/op
2. CharStreams (guava)                         avgt   10  12,999 ± 0,514  us/op
4. Stream Api (Java 8)                         avgt   10  15,811 ± 0,605  us/op
9. BufferedReader (JDK)                        avgt   10  16,038 ± 0,711  us/op
5. parallel Stream Api (Java 8)                avgt   10  21,544 ± 0,583  us/op
```

Замеры производительности для больших строк (длина = 50100), тесты можно найти на github (режим = среднее время выполнения (AverageTime), система = Linux Mint 17.2, Celeron Dual-Core T3500 2.10Ghz*2, 6Gb, чем значение ниже тем лучше, 200,715 — наилучшее):

```
             Benchmark                         Mode  Cnt   Score        Error  Units
8. ByteArrayOutputStream and read (JDK)        avgt   10   200,715 ±   18,103  us/op
1. IOUtils.toString (Apache Utils)             avgt   10   300,019 ±    8,751  us/op
6. InputStreamReader and StringBuilder (JDK)   avgt   10   347,616 ±  130,348  us/op
7. StringWriter and IOUtils.copy (Apache)      avgt   10   352,791 ±  105,337  us/op
2. CharStreams (guava)                         avgt   10   420,137 ±   59,877  us/op
9. BufferedReader (JDK)                        avgt   10   632,028 ±   17,002  us/op
5. parallel Stream Api (Java 8)                avgt   10   662,999 ±   46,199  us/op
4. Stream Api (Java 8)                         avgt   10   701,269 ±   82,296  us/op
10.BufferedInputStream, ByteArrayOutputStream  avgt   10   740,837 ±    5,613  us/op
3. Scanner (JDK)                               avgt   10   751,417 ±   62,026  us/op
11.InputStream.read() and StringBuilder (JDK)  avgt   10  2919,350 ± 1101,942  us/op
```

График зависимости среднего времени от длины строки, система Windows 8.1, Intel i7-4790 CPU 3.60GHz 3.60GHz, 16Gb:

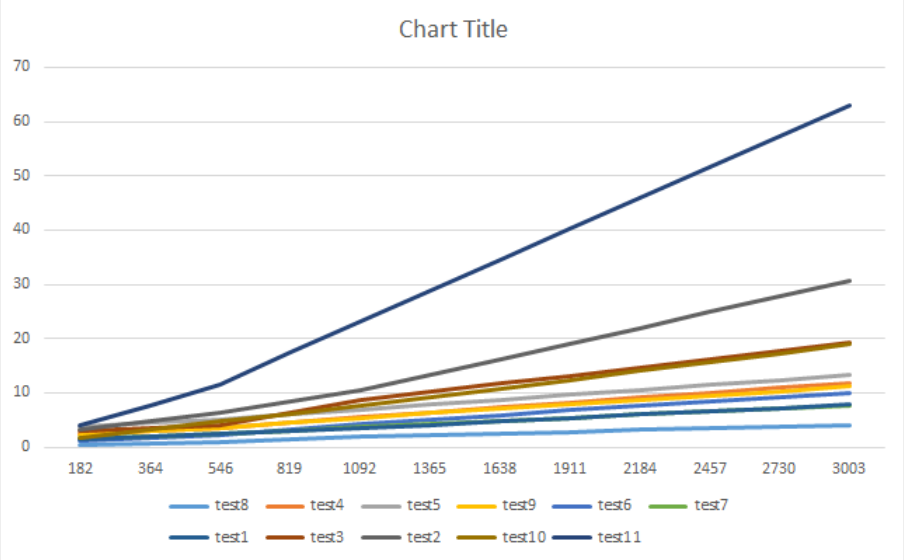

Таблица зависимости среднего времени от длины строки, система Windows 8.1, Intel i7-4790 CPU 3.60GHz 3.60GHz, 16Gb:

длинна	182	546	1092	3276	9828	29484	58968
test8	0.38	0.938	1.868	4.448	13.412	36.459	72.708
test4	2.362	3.609	5.573	12.769	40.74	81.415	159.864
test5	3.881	5.075	6.904	14.123	50.258	129.937	166.162
test9	2.237	3.493	5.422	11.977	45.98	89.336	177.39
test6	1.261	2.12	4.38	10.698	31.821	86.106	186.636
test7	1.601	2.391	3.646	8.367	38.196	110.221	211.016
test1	1.529	2.381	3.527	8.411	40.551	105.16	212.573
test3	3.035	3.934	8.606	20.858	61.571	118.744	235.428
test2	3.136	6.238	10.508	33.48	43.532	118.044	239.481
test10	1.593	4.736	7.527	20.557	59.856	162.907	323.147
test11	3.913	11.506	23.26	68.644	207.591	600.444	1211.545

Выводы

1. Самым быстрым решением во всех случаях и всех системах оказался 8 тест: Используя **ByteArrayOutputStream** и **inputStream.read** из JDK

```
ByteArrayOutputStream result = new ByteArrayOutputStream();
byte[] buffer = new byte[1024];
int length;
while ((length = inputStream.read(buffer)) != -1) {
    result.write(buffer, 0, length);
}
return result.toString("UTF-8");
```

2. Коротким и весьма быстрым решением будет использование IOUtils.toString из Apache Commons

```
String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
```

3. Stream Api из Java 8 показывает среднее время, а использование параллельных стримов имеет смысл только при довольно большой строки, иначе он работает очень долго (что в общем-то было ожидаемо)

4. Решение 11 лучше не использовать в принципе, так как он работает медленнее всех и не работает с Unicode,

P.S.

1. Английскую версию этой статьи можно найти в моем ответе на stackoverflow. Тесты в моем проекте на github. Если эта статья вам понравилась и вы поставите плюс на stackoverflow буду вам благодарен.
2. Буду очень благодарен за любые замечания, исправления, указания на ошибки или другие способы преобразования InputStream в строку
3. Так же советую посмотреть мой opensource проект useful-java-links — возможно, наиболее полная коллекция полезных Java библиотек, фреймворков и русскоязычного обучающего видео. Так же есть аналогичная английская версия этого проекта и начинаю opensource подпроект Hello world по подготовке коллекции простых примеров для разных Java библиотек в одном maven проекте (буду благодарен за любую помощь).

Общее оглавление 'Шпаргалок'

java, inputStream, java8, stream api, string, comparison

+21 29,4k 354

Автор: @vedenin1980

рейтинг
Luxoft 106,62
Сайт Twitter

Похожие публикации

+29 Шпаргалка Java-программиста 5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
96,8k 1205 26

+25 Шпаргалка Java программиста 4. Java Stream API
116k 727 17

+57 Шпаргалка Java программиста 3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
105k 1126 39

Комментарии (27)

speakingfish 14 апреля 2016 в 22:02 +2

А где в ваших примерах вызывается close()?

vedenin1980 14 апреля 2016 в 23:07 +1

Закрытие, открытие и создания стримов сознательно вынесено в методы производительность, которых не замеряется, чтобы они не влияли на замеры.

speakingfish 14 апреля 2016 в 23:25 0

Это замечательно, можете убрать это в тестах производительности, но наверное не надо публиковать это как «короткий однострочный» но некорректный и вызывающий утечки памяти код?

vedenin1980 14 апреля 2016 в 23:32 *(комментарий был изменён)* +5

Стоп, стрим в этом блоке кода не открывается и не создается, значит он не обязан в этом блоке кода и закрываться. Стрим может переиспользоваться в дальнейшем используя mark и т.п. функции и закрытие стрима, которые там сам не создавал, не всегда корректное поведение.

speakingfish 14 апреля 2016 в 23:38 −3

Не совсем так — у вас создаются и теряются ссылки на BufferedReader и InputStreamReader.

vedenin1980 14 апреля 2016 в 23:50 +4

Так метод close() у BufferedReader и InputStreamReader лишь обертка над вызовом close() стрима, сами Reader'ы просто собираются сборщиком как только станут недоступны. Как раз вызов close у Reader'ов, если мы не планируем закрыть стрим вызовет ошибку.

speakingfish 15 апреля 2016 в 00:15 *(комментарий был изменён)* −1

del

speakingfish 15 апреля 2016 в 00:23 +1

Смотрим:

```
public class InputStreamReader extends Reader {

    private final StreamDecoder sd;

    public void close() throws IOException {
        sd.close();
    }
}
```

Всё ok. Смотрим дальше:

```
public class StreamDecoder extends Reader
{
    public void close() throws IOException {
        synchronized (lock) {
            if (!isOpen)
                return;
            implClose();
            isOpen = false;
        }
    }

    void implClose() throws IOException {
```

```
        if (ch != null)
        ch.close();
        else
        in.close();
    }

}
```

Ой, что это за «ch»?

А вот что:

```
    private ReadableByteChannel ch;

    StreamDecoder(InputStream in, Object lock, CharsetDecoder dec) {
        super(lock);
        this.cs = dec.charset();
        this.decoder = dec;

        // This path disabled until direct buffers are faster
        if (false && in instanceof FileInputStream) {
        ch = getChannel((FileInputStream)in);
        if (ch != null)
            bb = ByteBuffer.allocateDirect(DEFAULT_BYTE_BUFFER_SIZE);
        }
        if (ch == null) {
        this.in = in;
        this.ch = null;
        bb = ByteBuffer.allocate(DEFAULT_BYTE_BUFFER_SIZE);
        }
        bb.flip();                      // So that bb is initially empty
    }
```

Так что выбирайте — вы или будете рассматривать частные случаи, полагаться на конкретную реализацию, которая может меняться или всё-таки изначально сделаете правильно?

vedenin1980 15 апреля 2016 в 02:09 +3

Ok, не хочу спорить, переписал все примеры на try-resources, раз вы считаете что это кого-то может ввести в заблуждение.

speakingfish 15 апреля 2016 в 11:38 0

Вот теперь вижу корректный код, который можно предлагать как учебный материал.

dougrinch 15 апреля 2016 в 15:35 *(комментарий был изменён)* +2

На самом деле, похоже, в общем случае (библиотечный метод *read(InputStream):String*) эта задача корректно вообще не решается. @vedenin1980 правильно написал что исходный стрим не мы открывали, не нам и закрывать. С другой же стороны, открытые нами ридеры закрыть все-таки необходимо. Только вот в подавляющем большинстве случаев они внутри закроют исходный стрим.

Так что только какой-нибудь *readAndClose(InputStream)* или всевозможные *readFile(String/File)*.

speakingfish 15 апреля 2016 в 18:46 0

Речь шла о создании цепочек обёрток и утере ссылок на них вместо корректного закрытия. Т.к. в начальном варианте этой статьи не было ни одного try-with-resources.

terryP 15 апреля 2016 в 19:02 *(комментарий был изменён)* +3

Ну, выше речь о том же, смотрите, если вы пишете код:

```
public String readStream(InputStream inputStream) {
    try(BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
         return br.lines().collect(Collectors.joining("\n"));
     }
}
```

То это не совсем корректный код, так как программист не ожидает что функция readStream закроет стрим, который он его передал. И это:

1. Нарушение функционального подхода,
2. Нарушение контракта,
3. Нарушение принципе одна функция — одно действия,
4. В целом, не очевидно из названия
5. Не всегда нужно закрывать стрим, иногда просто получить строку не закрывая, как в первом примере с Apache

С другой стороны, код:

```
public String readStream(InputStream inputStream) {
    BufferedReader br = new BufferedReader(new InputStreamReader(inputStream));
    return br.lines().collect(Collectors.joining("\n"));
}
```

Очевидно имеет тоже проблему в том что некрасиво не закрывать BufferedReader и InputStreamReader, даже при том что реальных проблем и утечек памяти это не вызовет.

Следовательно, само по себе использование BufferedReader и InputStreamReader в данном случае не очень корректно, так как приходится выбирать или не закрывать их, полагаясь на текущую реализацию, получая варнинги и чувство костылей.
Ила закрывать и получать проблемы с логичностью и понятностью данного метода.

У вас есть идея как сделать код выше лучше? Использовать BufferedReader и InputStreamReader без закрытия стрима, который передали методу?

 **dougrinch** 15 апреля 2016 в 19:46 +2

Вот.

```
public static String readStream(InputStream is) throws IOException {
    class InputStreamWrapper extends InputStream {

        private final InputStream is;

        public InputStreamWrapper(InputStream is) {
            this.is = is;
        }

        @Override
        public void close() throws IOException {
        }

        @Override
        public int read() throws IOException {
            return is.read();
        }

        @Override
        public int read(byte[] b) throws IOException {
            return is.read(b);
        }

        @Override
        public int read(byte[] b, int off, int len) throws IOException {
            return is.read(b, off, len);
        }

        @Override
        public long skip(long n) throws IOException {
            return is.skip(n);
        }

        @Override
        public int available() throws IOException {
            return is.available();
        }

        @Override
        public void mark(int readlimit) {
            is.mark(readlimit);
        }

        @Override
        public void reset() throws IOException {
```

```
                is.reset();
            }

            @Override
            public boolean markSupported() {
                return is.markSupported();
            }
        }

        try (InputStreamWrapper isw = new InputStreamWrapper(is)) {
            try (InputStreamReader isr = new InputStreamReader(isw)) {
                try (BufferedReader br = new BufferedReader(isr)) {
                    return br.lines().collect(joining("\n"));
                }
            }
        }
    }
```

Kolyuchkin 17 апреля 2016 в 11:13 +2

Мне показалось или действительно Ваш InputStreamWrapper очень уж похож на java.io.FilterInputStream? Не проще ли его расширить и переопределить пару нужных методов?

dougrinch 17 апреля 2016 в 14:44 −1

Да там половина java.io такая. Но вот конкретно про FileInputStream вам таки показалось. Он все-таки сильно отличается. С другой стороны, если мы хотим меньше кода, то можно переопределить close у уже используемого нами BufferedReader. Вот там все будет отлично. Только вот это будет, в отличии от моего примера, опора на текущую реализацию.

Kolyuchkin 17 апреля 2016 в 19:31 +1

Посмотрите внимательнее, пожалуйста, на мой комментарий. Я писал не про FileInputStream, а про FilterInputStream.

dougrinch 17 апреля 2016 в 20:42 0

Ваша правда, был невнимателен. FilterInputStream идеально подходит.

speakingfish 15 апреля 2016 в 18:38 0

Минусующим поясняю: в начальном варианте этой статьи не было ни одного try-with-resources

vba 15 апреля 2016 в 12:24 *(комментарий был изменён)* 0

Скажите а там в 5-ом решении, случайно, порядок срок не будет отличаться от изначального из за параллельной обработки?

vedenin1980 15 апреля 2016 в 12:34 +1

Нет, проблем там вроде бы быть не должно. Collector'ы «знают» как соединять результаты, полученные в параллельных стримах. Можете посмотреть эту мою статью пункт 3.10 (он в самом конце), там как раз описывается создания кастомного Collectors.joining

vba 15 апреля 2016 в 12:44 *(комментарий был изменён)* 0

Спасибо, я пока особо Stream.API не пользовался, больше все на вражеском Linq, так вот там после объявления параллельной обработки нужно его пнуть что бы он порядок восстановил.

darkslave 15 апреля 2016 в 12:52 0

надо заметить, что в пп. 3, 4, 5, 9, 10, 11 не указана явно кодировка.
по умолчанию, она определяется из настроек окружения и не всегда utf-8.

Chvalov 17 апреля 2016 в 13:17 0

Какая разница в использовании: Charset.forName(«UTF-8») и Charsets.UTF_8?

vedenin1980 17 апреля 2016 в 13:20 +2

Никакой, если посмотреть исходный код Charsets.UTF_8 определен как Charset.forName(«UTF-8»), просто константу использовать несколько более правильно, чем тестовое поле, где можно случайно описаться или ошибиться с кодировкой (вроде того что написать русскую С вместо английской) и т.п.

Nikiti4 17 апреля 2016 в 15:02 0

В 9-м примере не проще и быстрее будет если оставим тело цикла так

```
result.append("").append(line);
```

а потом:

```
result = result.trim();
```

?

vedenin1980 17 апреля 2016 в 20:35 *(комментарий был изменён)* +1

Вы имели в виду в первой строчке:

```
    result.append(newLine).append(line);
```

?
В принципе, и такой вариант возможен проблема только в том что удаляться все пробелы и знаки переноса в начале и в конце текста, иногда это может вызывать проблемы. Насчет простоты не могу судить, но быстрее работать не стало (trim() довольно тяжелая операция, а if не очень), добавил в тесты, вот результаты:

Для больших строк:

```
bufferedReaderReadLine           121.193 ±  2.160  us/op
bufferedReaderReadLine2          138.037 ±  4.962  us/op
```

Для маленьких строк:

```
bufferedReaderReadLine           2.200 ± 0.912  us/op
bufferedReaderReadLine2          2.384 ± 1.229  us/op
```

Разница невелика, но все-таки ваш вариант получился чуть-чуть медленнее.

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+242 Как Skype уязвимости чинил
23,3k 99 134

+20 Улучшение производительности PHP 7
2,4k 25 3

+77 Здравствуй, дорогой Мегафон
9k 15 79

+10 Прототип RFC HTTP-кодов состояния для ошибок разработчиков (диапазон 7XX)
1,5k 4 6

+12 Компания Google представила набор тестов Wycheproof
1,3k 7 0

Интересные публикации

Гейзенбаг: Версия 1.0 0

Компания Google представила набор тестов Wycheproof 0

Улучшение производительности PHP 7 3

Защищенный Dell 4

Преобразование формы представления данных при помощи Excel+PowerQuery 0