АВТОР

Pavlo Plynko

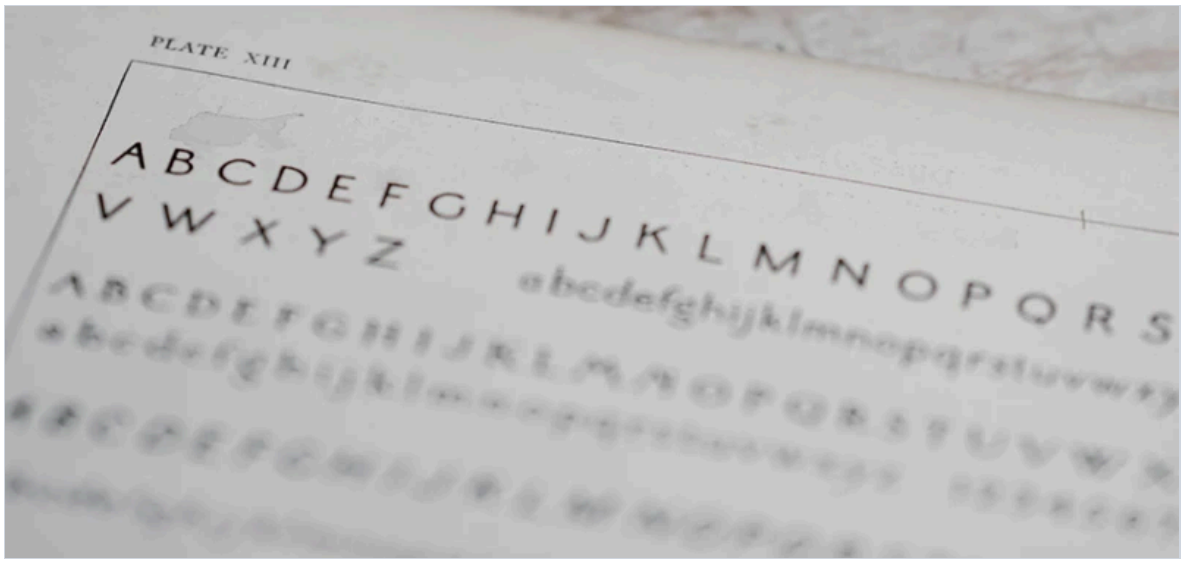
Java-разработчик в **CodeGym**

17 сентября 2019 829185 20

Знакомство со String, StringBuffer и StringBuilder в Java

Для работы с текстовыми данными в Java есть три класса: **String**, **StringBuffer** и **StringBuilder**. С первым каждый разработчик сталкивается еще в самом начале изучения языка. А что насчет оставшихся двух? Какие у них есть различия, и когда лучше использовать тот или иной класс? В общем-то, разница между ними небольшая, но лучше во всем разобраться на практике :)

- Класс String в Java
- Класс StringBuffer в Java
- Класс StringBuilder в Java



Класс String

Этот класс представляет собой последовательность символов. Все определенные в программ строковые литералы, вроде "This is String" — это экземпляры класса String.

У String есть две фундаментальные особенности:

- это immutable (неизменный) класс
- это final класс

Общем, у класса String не может быть наследников (`final`) и экземпляры класса нельзя изменить после создания (`immutable`).

Это дает классу String несколько важных преимуществ:

1. Благодаря неизменности, хэшкод экземпляра класса String кэшируется. Его не нужно вычислять каждый раз, потому что значения полей объекта никогда не изменятся после его создания. Это дает высокую производительность при использовании данного класса в качестве ключа для `HashMap`.
2. Класс String можно использовать в многопоточной среде без дополнительной синхронизации.
3. Еще одна особенность класса String — для него перегружен оператор "`+`" в Java. Поэтому конкатенация (сложение) строк выполняется довольно просто:

```
public static void main(String[] args) {
    String command = "Follow" + " " + "the" + " " + "white" + " " + "rabbit";
    System.out.println(command); // Follow the white rabbit
}
```

Под капотом конкатенация строк выполняется классом StringBuilder либо StringBuffer (на усмотрение компилятора) и методом `append` (об этих классах поговорим чуть позже).

Если мы будем складывать экземпляры класса String с экземплярами других классов, последние будут приводиться к строковому представлению:

```
public static void main(String[] args) {
    Boolean b = Boolean.TRUE;
    String result = "b is " + b;
    System.out.println(result); //b is true
}
```

Это еще одно интересное свойство класса String: объекты любых классов можно привести к строковому представлению, используя метод `toString()`, определенный в классе `Object` и наследуемый всеми остальными классами.

Часто метод toString() у объекта вызывается неявно. Например когда мы выводим что-то на экран или складываем String с объектом другого класса.

У класса String есть еще одна особенность. Все строковые литералы, определенные в Java коде, вроде "asdf", на этапе компиляции кэшируются и добавляются в так называемый пул строк.

Если мы запустим следующий код:

```
String a = "Wake up, Neo";
String b = "Wake up, Neo";

System.out.println(a == b);
```

Мы увидим в консоли *true*, потому что переменные `a` и `b` в действительности будут ссылаться на один и тот же экземпляр класса String, добавленный в пул строк на этапе компиляции. То есть, разные экземпляры класса с одинаковым значением не создаются, и память экономится.

Недостатки:

Нетрудно догадаться, что класс String нужен в первую очередь для работы со строками. Но в некоторых случаях перечисленные выше особенности класса String могут превратиться из достоинств в недостатки.

После создания строк в коде Java-программы с ними часто совершается множество операций:

- перевод строк в разные регистры;
- извлечение подстрок;
- конкатенация;

- и т.д.

Давайте посмотрим на этот код:

```
public static void main(String[] args) {

    String s = " Wake up, Neo! ";
    s = s.toUpperCase();
    s = s.trim();

    System.out.println("\"" + s + "\"");
}
```

С первого взгляда кажется, что мы всего-то перевели строку "Wake up, Neo!" в верхний регистр, удалили из данной строки лишние пробелы и обернули в кавычки.

На самом деле, в силу неизменности класса String, в результате каждой операции создаются новые экземпляры строк, а старые отбрасываются, порождая большое количество мусора.

Как же избежать нерационального использования памяти?

Класс StringBuffer

Чтобы справиться с созданием временного мусора из-за модификаций объекта String, можно использовать класс StringBuffer.

Это `mutable` класс, т.е. изменяемый. Объект класса StringBuffer может содержать в себе определенный набор символов, длину и значение которого можно изменить через вызов определенных методов.

Посмотрим, как работает данный класс.

Для создания нового объекта используется один из его конструкторов, например:

- StringBuffer() — создаст пустой (не содержащий символов) объект
- StringBuffer(String str) — создаст объект на основе переменной str (содержащий все символы str в той же последовательности)

Практика:

```
StringBuffer sb = new StringBuffer();
StringBuffer sb2 = new StringBuffer("Not empty");
```

Конкатенация строк через StringBuffer в Java выполняется с помощью метода `append`.

Вообще метод `append` в классе StringBuffer перегружен таким образом, что может принимать в себя практически любой тип данных:

```
public static void main(String[] args) {
    StringBuffer sb = new StringBuffer();

    sb.append(new Integer(2));
    sb.append("; ");
    sb.append(false);
    sb.append("; ");
    sb.append(Arrays.asList(1,2,3));
    sb.append("; ");

    System.out.println(sb); // 2; false; [1, 2, 3];
}
```

Метод `append` возвращает объект, на котором был вызван (как и многие другие методы), что позволяет вызывать его “цепочкой”.

Пример выше можно написать так:

```
public static void main(String[] args) {
    StringBuffer sb = new StringBuffer();

    sb.append(new Integer(2))
            .append("; ")
            .append(false)
            .append("; ")
            .append(Arrays.asList(1,2,3))
            .append("; ");

    System.out.println(sb); // 2; false; [1, 2, 3];
}
```

Для работы со строками у класса StringBuffer есть ряд методов. Перечислим основные:

- `delete(int start, int end)` — удаляет подстроку символов начиная с позиции `start`, заканчивая `end`
- `deleteCharAt(int index)` — удаляет символ в позиции `index`
- `insert(int offset, String str)` — вставляет строку `str` в позицию `offset`. Метод `insert` также перегружен и может принимать различные аргументы
- `replace(int start, int end, String str)` — заменит все символы начиная с позиции `start` до позиции `end` на `str`
- `reverse()` — меняет порядок всех символов на противоположный
  `substring(int start)` — вернет подстроку, начиная с позиции `start`
- `substring(int start, int end)` — вернет подстроку, начиная с позиции `start` до позиции `end`

Полный список методов и конструкторов есть в официальной документации.

Как работают указанные выше методы? Посмотрим на практике:

```
public static void main(String[] args) {
     String numbers = "0123456789";

     StringBuffer sb = new StringBuffer(numbers);

     System.out.println(sb.substring(3)); // 3456789
     System.out.println(sb.substring(4, 8)); // 4567
     System.out.println(sb.replace(3, 5, "ABCDE")); // 012ABCDE56789

     sb = new StringBuffer(numbers);
     System.out.println(sb.reverse()); // 9876543210
     sb.reverse(); // Вернем изначальный порядок

     sb = new StringBuffer(numbers);
     System.out.println(sb.delete(5, 9)); // 012349
     System.out.println(sb.deleteCharAt(1)); // 02349
     System.out.println(sb.insert(1, "One")); // 0One2349
    }
```

Преимущества:

1. Как уже сказано, StringBuffer — изменяемый класс, поэтому при работе с ним не возникает такого же количества мусора в памяти, как со String. Поэтому если над строками проводится много модификаций, лучше использовать `StringBuffer`.
2. StringBuffer — потокобезопасный класс. Его методы синхронизированы, а экземпляры могут быть использованы несколькими потоками одновременно.

Недостатки:

С одной стороны, потокобезопасность — преимущество класса, а другой — недостаток. Синхронизированные методы работают медленнее не сихнронизированных.

И здесь в игру вступает StringBuilder. Давайте разберемся, что это за класс Java — StringBuilder, какие методы есть и в чем его особенности.

Класс StringBuilder

StringBuilder в Java — класс, представляющий последовательность символов. Он очень похож на StringBuffer во всем, кроме потокобезопасности.

StringBuilder предоставляет API, аналогичный API StringBuffer'а.

Продемонстрируем это на уже знакомом примере, заменив объявление переменных со StringBufer'а на StringBuilder:

```
public static void main(String[] args) {
    String numbers = "0123456789";

    StringBuilder sb = new StringBuilder(numbers);

    System.out.println(sb.substring(3)); //3456789
    System.out.println(sb.substring(4, 8)); //4567
    System.out.println(sb.replace(3, 5, "ABCDE")); //012ABCDE56789

    sb = new StringBuilder(numbers);
    System.out.println(sb.reverse()); //9876543210
    sb.reverse(); // Вернем изначальный порядок

    sb = new StringBuilder(numbers);
    System.out.println(sb.delete(5, 9)); //012349
    System.out.println(sb.deleteCharAt(1)); //02349
    System.out.println(sb.insert(1, "One")); //0One2349
}
```

Разница лишь в том, что StringBuffer потокобезопасен, и все его методы синхронизированы, а StringBuilder — нет. Это единственная особенность.

StringBuilder в Java работает быстрее StringBuffer'а благодаря несинхронизированности методов.

Поэтому в большинстве случаев, кроме многопоточной среды, лучше использовать для программы на Java StringBuilder. Резюмируем все в сравнительной таблице трех классов:

String vs StringBuffer vs StringBuilder

	String	StringBuffer	StringBuilder
Изменяемость	`Immutable` (нет)	`mutable` (да)	`mutable` (да)
Расширяемость	`final` (нет)	`final` (нет)	`final` (нет)
Потокобезопасность	Да, за счет неизменяемости	Да, за счет синхронизации	Нет
Когда использовать	При работе со строками, которые редко будут модифицироваться	При работе со строками, которые часто будут модифицироваться в многопоточной среде	При работе со строками, которые часто будут модифицироваться, в однопоточной среде