

# Программирование на языке Java.

## Работа со строками

Картузов А.В.

В этой главе обсуждаются средства языка Java для работы со строками. В языках C и C++ отсутствует встроенная поддержка такого объекта, как строка. В них при необходимости передается адрес последовательности байтов, содержимое которых трактуется как символы до тех пор, пока не будет встречен нулевой байт, отмечающий конец строки. В пакет `java.lang` встроен класс, инкапсулирующий структуру данных, соответствующую строке. Этот класс, называемый `String`, не что иное, как объектное представление неизменяемого символьного массива. В этом классе есть методы, которые позволяют сравнивать строки, осуществлять в них поиск и извлекать определенные символы и подстроки. Класс `StringBuffer` используется тогда, когда строку после создания требуется изменять.

### Замечание:

И `String`, и `StringBuffer` объявлены `final`, что означает, что ни от одного из этих классов нельзя производить подклассы. Это было сделано для того, чтобы можно было применить некоторые виды оптимизации позволяющие увеличить производительность при выполнении операций обработки строк.

## 1. Конструкторы

Как и в случае любого другого класса, вы можете создавать объекты типа `String` с помощью оператора `new`. Для создания пустой строки используется конструктор без параметров:

```
String s = new String();
```

Приведенный ниже фрагмент кода создает объект `s` типа `String` инициализируя его строкой из трех символов, переданных конструктору в качестве параметра в

СИМВОЛЬНОМ МАССИВЕ.

```
char chars[] = { 'a', 'b', 'c' };  
String s = new String(chars);  
System.out.println(s);
```

Этот фрагмент кода выводит строку <abc>. Итак, у этого конструктора—3 параметра:

```
String(char chars[], int начальныйИндекс, int числоСимволов);
```

Используем такой способ инициализации в нашем очередном примере:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };  
String s = new String(chars, 2, 3);  
System.out.println(s);
```

Этот фрагмент выведет: **cde**.

## 2. Специальный синтаксис для работы со строками

В Java включено несколько приятных синтаксических дополнений, цель которых—помочь программистам в выполнении операций со строками. В числе таких операций создание объектов типа String слияние нескольких строк и преобразование других типов данных в символьное представление.

## 3. Создание строк

Java включает в себя стандартное сокращение для этой операции—запись в виде литерала, в которой содержимое строки заключается в пару двойных кавычек. Приводимый ниже фрагмент кода эквивалентен одному из предыдущих, в котором строка инициализировалась массивом типа char.

```
String s = "abc";  
System.out.println(s);
```

Один из общих методов, используемых с объектами String—метод `length`, возвращающий число символов в строке. Очередной фрагмент выводит число 3, поскольку в используемой в нем строке—3 символа.

```
String s = "abc";  
System.out.println(s.length);
```

В Java интересно то, что для каждой строки-литерала создается свой представитель класса `String`, так что вы можете вызывать методы этого класса непосредственно со строками-литералами, а не только со ссылочными переменными. Очередной пример также выводит число 3.

```
System.out.println("abc".Length());
```

## 4. Слияние строк

Строку `String s = <He is > + age + " years old."`; в которой с помощью оператора `+` три строки объединяются в одну, прочесть и понять безусловно легче, чем ее эквивалент, записанный с явными вызовами тех самых методов, которые неявно были использованы в первом примере:

```
String s = new StringBuffer("He is ").append(age);  
s.append(" years old.").toString();
```

По определению каждый объект класса `String` не может изменяться. Нельзя ни вставить новые символы в уже существующую строку, ни поменять в ней одни символы на другие. И добавить одну строку в конец другой тоже нельзя. Поэтому транслятор Java преобразует операции, выглядящие, как модификация объектов `String`, в операции с родственным классом `StringBuffer`.

### Замечание:

Все это может показаться вам необоснованно сложным. А почему нельзя обойтись одним классом `String`, позволив ему вести себя примерно так же, как `StringBuffer`? Все дело в производительности. Тот факт, что объекты типа `String` в Java неизменны, позволяет транслятору применять к операциям с ними различные способы оптимизации.

## 5. Последовательность выполнения операторов

Давайте еще раз обратимся к нашему последнему примеру:

```
String s = "He is " + age + " years old.";
```

В том случае, когда `age`—не `String`, а переменная, скажем, типа `int`, в этой строке кода заключено еще больше магии транслятора. Целое значение переменной `int` передается совмещенному методу `append` класса `StringBuffer`, который преобразует его в текстовый вид и добавляет в конец содержащейся в объекте строки. Вам нужно быть внимательным при совместном использовании целых выражений и слияния строк, в противном случае результат может получиться совсем не тот, который вы ждали. Взгляните на следующую строку:

```
String s = "four: " + 2 + 2;
```

Быть может, вы надеетесь, что в `s` будет записана строка `<four: 4>`? Не угадали—с вами сыграла злую шутку последовательность выполнения операторов. Так что в результате получается `"four: 22"`.

Для того, чтобы первым выполнилось сложение целых чисел, нужно использовать скобки :

```
String s = "four: " + (2 + 2);
```

## 6. Преобразование строк

В каждом классе `String` есть метод `toString`—либо своя собственная реализация, либо вариант по умолчанию, наследуемый от класса `Object`. Класс в нашем очередном примере замещает наследуемый метод `toString` своим собственным, что позволяет ему выводить значения переменных объекта.

```
class Point {
    int x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "Point[" + x + ", " + y + "];"
    }
}
```

```
class toStringDemo {
    public static void main(String args[]) {
        Point p = new Point(10, 20);
        System.out.println("p = " + p);
    }
}
```

Ниже приведен результат, полученный при запуске этого примера.

```
C:\> Java toStringDemo
p = Point[10, 20]
```

## 7. Извлечение символов

Для того, чтобы извлечь одиночный символ из строки, вы можете сослаться непосредственно на индекс символа в строке с помощью метода `charAt`. Если вы хотите в один прием извлечь несколько символов, можете воспользоваться методом `getChars`. В приведенном ниже фрагменте показано, как следует извлекать массив символов из объекта типа `String`.

```
class getCharsDemo {
    public static void main(String args[]) {
        String s = "This is a demo of the getChars method.";
        int start = 10;
        int end = 14;
        char buf[] = new char[end-start];
        s.getChars(start, end, buf, 0);
        System.out.println(buf);
    }
}
```

Обратите внимание—метод `getChars` не включает в выходной буфер символ с индексом `end`. Это хорошо видно из вывода нашего примера—выводимая строка состоит из 4 символов.

```
C:\> java getCharsDemo
demo
```

Для удобства работы в `String` есть еще одна функция—`toCharArray`, которая

возвращает в выходном массиве типа `char` всю строку. Альтернативная форма того же самого механизма позволяет записать содержимое строки в массив типа `byte`, при этом значения старших байтов в 16-битных символах отбрасываются. Соответствующий метод называется `getBytes`, и его параметры имеют тот же смысл, что и параметры `getChars`, но с единственной разницей—в качестве третьего параметра надо использовать массив типа `byte`.

## 8. Сравнение

Если вы хотите узнать, одинаковы ли две строки, вам следует воспользоваться методом `equals` класса `String`. Альтернативная форма этого метода называется `equalsIgnoreCase`, при ее использовании различие регистров букв в сравнении не учитывается. Ниже приведен пример, иллюстрирующий использование обоих методов:

```
class equalDemo {
    public static void main(String args[]) {
        String s1="Hello";
        String s2="Hello";
        String s3="Good-bye";
        String s4="HELLO";
        System.out.println(s1+"equals"+s2+"->"+s1.equals(s2));
        System.out.println(s1+"equals"+s3+"->"+s1.equals(s3));
        System.out.println(s1+"equals"+s4+"->"+s1.equals(s4));
        System.out.println(s1+"equalsIgnoreCase"+s4+"->"+
            s1.equalsIgnoreCase(s4));
    }
}
```

Результат запуска этого примера :

```
C:\> java equalDemo
Hello equals Hello -> true
Hello equals Good-bye -> false
Hello equals HELLO -> false
Hello equalsIgnoreCase HELLO -> true
```

В классе `String` реализована группа сервисных методов, являющихся специализированными версиями метода `equals`. Метод `regionMatches` используется для сравнения подстроки в исходной строке с подстрокой в строке-параметре. Метод

`startsWith` проверяет, начинается ли данная подстрока фрагментом, переданным методу в качестве параметра. Метод `endsWith` проверяет совпадает ли с параметром конец строки.

## 9. Равенство

Метод `equals` и оператор `==` выполняют две совершенно различных проверки. Если метод `equal` сравнивает символы внутри строк, то оператор `==` сравнивает две переменные-ссылки на объекты и проверяет, указывают ли они на разные объекты или на один и тот же. В очередном нашем примере это хорошо видно—содержимое двух строк одинаково, но, тем не менее, это—различные объекты, так что `equals` и `==` дают разные результаты.

```
class EqualsNotEqualTo {
    public static void main(String args[]) {
        String s1="Hello";
        String s2=new String(s1);
        System.out.println(s1+"equals"+s2+"->"+s1.equals(s2));
        System.out.println(s1+"=="+"s2+", ->"+(s1==s2));
    }
}
```

Вот результат запуска этого примера:

```
C:\> java EqualsNotEqualTo
Hello equals Hello -> true
Hello == Hello -> false
```

## 10. Упорядочение

Зачастую бывает недостаточно просто знать, являются ли две строки идентичными. Для приложений, в которых требуется сортировка, нужно знать, какая из двух строк меньше другой. Для ответа на этот вопрос нужно воспользоваться методом `compareTo` класса `String`. Если целое значение, возвращенное методом, отрицательно, то строка, с которой был вызван метод, меньше строки-параметра, если положительно—больше. Если же метод `compareTo` вернул значение 0, строки идентичны. Ниже приведена

программа, в которой выполняется пузырьковая сортировка массива строк, а для сравнения строк используется метод `compareTo`. Эта программа выдает отсортированный в алфавитном порядке список строк.

```
class SortString {
static String arr[] = {"Now", "is", "the", "time", "for", "all",
                      "good", "men", "to", "come", "to", "the",
                      "aid", "of", "their", "country" };
public static void main(String args[]) {
for (int j = 0; j < arr.length; j++) {
    for (int i = j + 1; i < arr.length; i++) {
        if (arr[i].compareTo(arr[j]) < 0) {
            String t = arr[j];
            arr[j] = arr[i];
            arr[i] = t;
        }
    }
    System.out.println(arr[j]);
}
} }
```

## 11. `indexOf` и `lastIndexOf`

В класс `String` включена поддержка поиска определенного символа или подстроки, для этого в нем имеются два метода—`indexOf` и `lastIndexOf`. Каждый из этих методов возвращает индекс того символа, который вы хотели найти, либо индекс начала искомой подстроки. В любом случае, если поиск оказался неудачным методы возвращают значение `-1`. В очередном примере показано, как пользоваться различными вариантами этих методов поиска.

```
class indexOfDemo {
public static void main(String args[]) {
String s = "Now is the time for all good men " +
          "to come to the aid of their country " +
          "and pay their due taxes.";
System.out.println(s);
System.out.println("indexOf(t) = " + s.indexOf('f'));
System.out.println("lastIndexOf(t) = " + s.lastIndexOf('f'));
}
```



```
System.out.println("indexOf(the) = " + s.indexOf("the"));
System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));
System.out.println("indexOf(t, 10) = " + s.indexOf('f' , 10));
System.out.println("lastIndexOf(t, 50) = " + s.lastIndexOf('f' , 50));
System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));
System.out.println("lastIndexOf(the, 50) = " + s.lastIndexOf("the", 50));
} }
```

Ниже приведен результат работы этой программы. Обратите внимание на то, что индексы в строках начинаются с нуля.

```
C:> java indexOfDemo
Now is the time for all good men to come to the aid of their country
and pay their due taxes.
indexOf(t) = 7
lastIndexOf(t) = 87
indexOf(the) = 7
lastIndexOf(the) = 77
indexOf(t, 10) = 11
lastIndexOf(t, 50) = 44
indexOf(the, 10) = 44
lastIndexOf(the, 50) = 44
```

## 12. Модификация строк при копировании

Поскольку объекты класса `String` нельзя изменять, всякий раз, когда вам захочется модифицировать строку, придется либо копировать ее в объект типа `StringBuffer`, либо использовать один из описываемых ниже методов класса `String`, которые создают новую копию строки, внося в нее ваши изменения.

### 12.1. `substring`

Вы можете извлечь подстроку из объекта `String`, используя метод `sub-string`. Этот метод создает новую копию символов из того диапазона индексов оригинальной строки, который вы указали при вызове. Можно указать только индекс первого символа нужной подстроки—тогда будут скопированы все символы, начиная с указанного и до конца строки. Также можно указать и начальный, и конечный индексы—при этом в новую строку будут скопированы все символы, начиная с

первого указанного, и до (но не включая его) символа, заданного конечным индексом.

```
"Hello World".substring(6) -> "World"  
"Hello World".substring(3,8) -> "lo Wo"
```

## 12.2. concat

Слияние, или конкатенация строк выполняется с помощью метода `concat`. Этот метод создает новый объект `String`, копируя в него содержимое исходной строки и добавляя в ее конец строку, указанную в параметре метода.

```
"Hello".concat(" World") -> "Hello World"
```

## 12.3. replace

Методу `replace` в качестве параметров задаются два символа. Все символы, совпадающие с первым, заменяются в новой копии строки на второй символ.

```
"Hello".replace('l' , 'w') -> "Hewwo"
```

## 12.4. toLowerCase и toUpperCase

Эта пара методов преобразует все символы исходной строки в нижний и верхний регистр, соответственно.

```
"Hello".toLowerCase() -> "hello"  
"Hello".toUpperCase() -> "HELLO"
```

## 12.5. trim

И, наконец, метод `trim` убирает из исходной строки все ведущие и замыкающие пробелы.

```
"Hello World   ".trim() -> "Hello World"
```

## 13. valueOf

Если вы имеете дело с каким-либо типом данных и хотите вывести значение этого типа в удобочитаемом виде, сначала придется преобразовать это значение в текстовую строку. Для этого существует метод `valueOf`. Такой статический метод определен для любого существующего в Java типа данных (все эти методы совмещены, то есть используют одно и то же имя). Благодаря этому не составляет труда преобразовать в строку значение любого типа.

## 14. StringBuffer

`StringBuffer`—близнец класса `String`, предоставляющий многое из того, что обычно требуется при работе со строками. Объекты класса `String` представляют собой строки фиксированной длины, которые нельзя изменять. Объекты типа `StringBuffer` представляют собой последовательности символов, которые могут расширяться и модифицироваться. Java активно использует оба класса, но многие программисты предпочитают работать только с объектами типа `String`, используя оператор `+`. При этом Java выполняет всю необходимую работу со `StringBuffer` за сценой.

### 14.1. Конструкторы

Объект `StringBuffer` можно создать без параметров, при этом в нем будет зарезервировано место для размещения 16 символов без возможности изменения длины строки. Вы также можете передать конструктору целое число, для того чтобы явно задать требуемый размер буфера. И, наконец, вы можете передать конструктору строку, при этом она будет скопирована в объект и дополнительно к этому в нем будет зарезервировано место еще для 16 символов. Текущую длину `StringBuffer` можно определить, вызвав метод `length`, а для определения всего места, зарезервированного под строку в объекте `StringBuffer` нужно воспользоваться методом `capacity`. Ниже приведен пример, поясняющий это:

```
class StringBufferDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer = " + sb);
        System.out.println("length = " + sb.length());
        System.out.println("capacity = " + sb.capacity());
    }
}
```

```
    }  
}
```

Вот вывод этой программы, из которого видно, что в объекте `StringBuffer` для манипуляций со строкой зарезервировано дополнительное место.

```
C:\> java StringBufferDemo  
buffer = Hello  
length = 5  
capacity = 21
```

## 14.2. `ensureCapacity`

Если вы после создания объекта `StringBuffer` захотите зарезервировать в нем место для определенного количества символов, вы можете для установки размера буфера воспользоваться методом `ensureCapacity`. Это бывает полезно, когда вы заранее знаете, что вам придется добавлять к буферу много небольших строк.

## 14.3. `setLength`

Если вам вдруг понадобится в явном виде установить длину строки в буфере, воспользуйтесь методом `setLength`. Если вы зададите значение, большее чем длина содержащейся в объекте строки, этот метод заполнит конец новой, расширенной строки символами с кодом нуль. В приводимой чуть дальше программе `setCharDemo` метод `setLength` используется для укорачивания буфера.

## 14.4. `charAt` и `setCharAt`

Одиночный символ может быть извлечен из объекта `StringBuffer` с помощью метода `charAt`. Другой метод `setCharAt` позволяет записать в заданную позицию строки нужный символ. Использование обоих этих методов проиллюстрировано в примере:

```
class setCharAtDemo {  
    public static void main(String args[]) {  
        StringBuffer sb = new StringBuffer("Hello");  
        System.out.println("buffer before = " + sb);  
        System.out.println("charAt(1) before = " + sb.charAt(1));  
    }  
}
```

```
        sb.setCharAt(1, 'i');
        sb.setLength(2);
        System.out.println("buffer after = " + sb);
        System.out.println("charAt(1) after = " + sb.charAt(1));
    }
}
```

Вот вывод, полученный при запуске этой программы.

```
C:\> java setCharAtDemo
buffer before = Hello
charAt(1) before = e
buffer after = Hi
charAt(1) after = i
```

## 14.5. append

Метод `append` класса `StringBuffer` обычно вызывается неявно при использовании оператора `+` в выражениях со строками. Для каждого параметра вызывается метод `String.valueOf` и его результат добавляется к текущему объекту `StringBuffer`. К тому же при каждом вызове метод `append` возвращает ссылку на объект `StringBuffer`, с которым он был вызван. Это позволяет выстраивать в цепочку последовательные вызовы метода, как это показано в очередном примере.

```
class appendDemo {
    public static void main(String args[]) {
        String s;
        int a = 42;
        StringBuffer sb = new StringBuffer(40);
        s = sb.append("a = ").append(a).append("!").toString();
        System.out.println(s);
    }
}
```

Вот вывод этого примера:

```
C:\> Java appendDemo
a = 42!
```

## 14.6. insert

Метод `insert` идентичен методу `append` в том смысле, что для каждого возможного типа данных существует своя совмещенная версия этого метода. Правда, в отличие от `append`, он не добавляет символы, возвращаемые методом `String.valueOf`, в конец объекта `StringBuffer`, а вставляет их в определенное место в буфере, задаваемое первым его параметром. В очередном нашем примере строка "there" вставляется между "hello" и "world!".

```
class insertDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("hello world !");
        sb.insert(6,"there ");
        System.out.println(sb);
    }
}
```

При запуске эта программа выводит следующую строку:

```
C:\> java insertDemo
hello there world!
```

## 15. Без строк не обойдешься

Почти любой аспект программирования в Java на каком либо этапе подразумевает использование классов `String` и `StringBuffer`. Они понадобятся и при отладке, и при работе с текстом, и при указании имен файлов и адресов URL в качестве параметров методам. Каждый второй байт большинства строк в Java—нулевой (Unicode пока используется редко). То, что строки в Java требуют вдвое больше памяти, чем обычные ASCII, не очень пугает, пока вам для эффективной работы с текстом в редакторах и других подобных приложениях не придется напрямую работать с огромным массивом типа `char`.