Java

String / Regular expressions

Лекция #4 С

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

Класс Formatter

Класс Formatter

Базовой частью поддержки создания форматированного вывода в языке Java служит класс Formatter, включенный в пакет **java.util**.

Он обеспечивает преобразования формата (format conversions) позволяющие выводить числа, строки и время и даты практически в любом формате.

В классе Formatter объявлен метод format(), который преобразует переданные в него параметры в строку заданного формата и сохраняет в объекте типа Formatter.

Класс Formatter. Спецификаторы формата

%a	Шестнадцатеричное значение с плавающей точкой
%b	Логическое (булево) значение аргумента
%c	Символьное представление аргумента
%d	Десятичное целое значение аргумента
%h	Хэш-код аргумента
%e	Экспоненциальное представление аргумента
%f	Десятичное значение с плавающей точкой
%g	Выбирает более короткое представление из двух: %e или %f
%o	Восьмеричное целое значение аргумента
%n	Вставка символа новой строки
%s	Строковое представление аргумента
%t	Время и дата
%x	Шестнадцатеричное целое значение аргумента
%%	Вставка знака %

Класс Formatter. Флаги формата

-	Выравнивание влево
#	Изменяет формат преобразования
0	Выводит значение, дополненное нулями вместо пробелов. Применим только к числам.
Пробел	Положительные числа предваряются пробелом
+	Положительные числа предваряются знаком +. Применим только к числам.
,	Числовые значения включают разделители групп. Применим только к числам.
(	Отрицательные числовые значения заключаются в скобки. Применим только к числам.

Класс Formatter. Примеры

```
// %x   Шестнадцатеричное целое значение аргумента
//%n   Вставка символа новой строки
System.out.printf("%x%n", 16);
```

10

```
// %e   Экспоненциальное представление аргумента
System.out.printf("%.2e%n", 16.0);
```

1,60e+01

```
// %f   Десятичное значение с плавающей точкой
//(   Отрицательные числовые значения заключаются в скобки.
System.out.printf("%(5.1f%n", -16.0);
```

(16,0)

Класс Formatter. Примеры. Выравнивание по левому краю

```
System.out.printf("%-12s%-8s%-6s%n","Фамилия","Класс","Возраст");
System.out.printf("%-12s%-8s%-6s%n","Коржиков","Иван","13");
System.out.printf("%-12s%-8s%-6s%n","Хорошкина","Мария","12");
```

```
Фамилия     Класс   Возраст
Коржиков    Иван    13
Хорошкина   Мария   12
```

Класс Formatter

%[argument_index$][flags][width][.precision]conversion

% — специальный символ, обозначающий начало инструкций форматирования.

[argument_index$] — целое десятичное число, указывающее позицию аргумента в списке аргументов. Ссылка на первый аргумент "1$", ссылка на второй аргумент "2$" и т.д. Не является обязательной частью инструкции. Если позиция не задана, то аргументы должны находиться в том же порядке, что и ссылки на них в строке форматирования.

[flags] — специальные символы для форматирования. Например, флаг "+" означает, что числовое значение должно включать знак +, флаг "-" означает выравнивание результата по левому краю, флаг «,» устанавливает разделитель тысяч у целых чисел. Не является обязательной частью инструкции.

[width] — положительное целое десятичное число, которое определяет минимальное количество символов, которые будут выведены. Не является обязательной частью инструкции.

[.precision] — не отрицательное целое десятичное число с точкой перед ним. Обычно используется для ограничения количества символов. Специфика поведения зависит от вида преобразования. Не является обязательной частью инструкции.

conversion — это символ, указывающий, как аргумент должен быть отформатирован. Например **d** для целых чисел, **s** для строк, **f** для чисел с плавающей точкой. Является обязательной частью инструкции.

Класс Formatter. Примеры форматирования

```
System.out.printf("%1$+020.10f", Math.PI);
```

```
+00000003,1415926536
```

```
System.out.printf("%2$.2e%1$4d%n", 123, Math.PI);

System.out.printf("%.2e%4d%n",Math.PI, 123);
```

```
3,14e+00  123
3,14e+00  123
```

Класс Formatter

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Main {

    public static void main(String[] args) {

        Date now = new Date();

        DateFormat formatter =
            new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");

        String s = formatter.format(now);

  System.out.println(s);

        System.out.println(now);
    }
}
```

```
28.02.2020 13:00:22

Fri Feb 28 13:00:22 GMT+03:00 2020
```

Регулярные выражения

https://docs.oracle.com/javase/tutorial/essential/regex/

Регулярные выражения в Java

Регулярное выражение – это шаблон для поиска строки в тексте.

В Java исходным представлением этого шаблона всегда является строка, то есть объект класса String.

Не каждая строка компилируется в регулярное выражение, а только та, что соответствует синтаксису регулярных выражений.

```
// шаблон строки из трех цифровых символов;
String regex="\\d{3}";
```

```
// шаблон строки из трех нецифровых символов в количестве от 2 до 5;
String regex="\\D{2,5}";
```

Регулярные выражения в Java. java.util.regex

Пакет java.util.regex исходно состоит из следующих трех классов:

Pattern Class – объект класса Pattern представляет скомпилированное представление регулярного выражения.
Для создания шаблона необходимо вызвать один из методов **compile()**, который далее произведет возврат объекта класса Pattern. Регулярное выражение в данных методах принимается как первый аргумент.

Matcher Class – объект класса Matcher представляет механизм, который интерпретирует шаблон, а также производит операции сопоставления с вводимой строкой. Объект класса Matcher может быть получен путем вызова метода **matcher()** на объекте класса Pattern.

PatternSyntaxException – объект класса PatternSyntaxException представляет непроверяемое исключение, которое обозначает синтаксическую ошибку в шаблоне регулярного выражения.

Регулярные выражения в Java. Pattern

Класс Java Pattern (java.util.regex.Pattern) является основной точкой доступа к API Java регулярных выражений.

Java Pattern можно использовать двумя способами:

- Метод **Pattern.matches**() нужен для быстрой проверки соответствия текста заданному регулярному выражению.

- Так же можно скомпилировать экземпляр Pattern, используя **Pattern.compile**(). Его можно использовать несколько раз для сопоставления регулярного выражения с несколькими текстами.

Регулярные выражения в Java. Matcher

Класс Java Matcher (**java.util.regex.Matcher**) создан для поиска некоторого множества вхождений регулярного выражения в одном тексте и поиска по одному шаблону в разных текстах.

```
1. Pattern p = Pattern.compile (RegExpression);
2. Matcher m = p.matcher (InputString);
3. System.out.println (m.find());
```

Регулярные выражения в Java. Matcher. Методы

- boolean **matches**(): вернет значение true при совпадении строки с шаблоном.
- boolean **find**(): вернет значение true при обнаружении подстроки, совпадающей с шаблоном, и перейдет к ней.
- int **start**(): начальный индекс найденного совпадения.
- int **end**(): вернет конечный индекс найденного совпадения.
- String **replaceAll(String** str): вернет значение измененной строки подстрокой str.

Примеры

Регулярные выражения в Java. 1. Пример D{4}

```
// Четыре нецифровых символа подряд
Pattern p = Pattern.compile ("\\D{4}");

Matcher m = p.matcher ("ABCD");
System.out.println (m.find());
```

→ true

ABC

ABCDE

→ true

A123

A1B2C3D

→ false

Регулярные выражения в Java. 2. Пример D{2,}

```
// Больше, или равно двух нецифровых символов подряд
    Pattern p = Pattern.compile("\\D{2,}");
    Matcher m = p.matcher("AB");
    System.out.println(m.find());
```

true

```
// Пробел – нецифровой символ
```

A B C3

A1 B2

true

123A4B

d1a2b3c

false

Регулярные выражения в Java. 3. Пример d{2,4}

```
// от 2 до 4 цифровых символов
        Pattern p = Pattern.compile("\\d{2,4}");

        Matcher m = p.matcher("A 12 B");

        System.out.println(m.find());
```

true

Регулярные выражения в Java. 4. Пример S\\d{2,4}\\s

```
// S - непробельный символ
// s - пробельный символ
        Pattern p = Pattern.compile("\\S\\d{2,4}\\s");

        Matcher m = p.matcher("-123 A");

        System.out.println(m.find());
```

→ **true**

" 123A"

0123A

→ **false**

Регулярные выражения в Java. 5. Пример w{3}

```
// w - буквенно-цифровой символ, или знак подчёркивания
        Pattern p = Pattern.compile("\\w{3}");

        Matcher m = p.matcher("_A_1_");

        System.out.println(m.find());
```

true

"1 A 3"

"!!!"

false

Регулярные выражения в Java. 6. Пример W+

```
// W - любой символ, кроме буквенного, цифрового или знака
// подчёркивания
// + один или более раз

    Pattern p = Pattern.compile("\\W+");

    Matcher m = p.matcher("#&");

    System.out.println(m.find());
```

true

"1_A"

"AB"

false

Регулярные выражения в Java. 7. Пример [^xyz]

```
// [^xyz]  - любой, кроме перечисленных

    Pattern p = Pattern.compile("\\[^xyz]");

    Matcher m = p.matcher("a12");

    System.out.println(m.find());
```

→ **true**

1abx

y_AB

→ **false**

Регулярные выражения в Java. 8. Пример [a-d[1-5]]

```
// [a-d[1-5]]  - объединение символов

        Pattern p = Pattern.compile("\\[a-d[1-5]]");

        Matcher m = p.matcher("xyz789");

        System.out.println(m.find());
```

→ **false**

1a

d*@

→ **true**

Регулярные выражения в Java. 9. Пример ^ABC с начала

```
Pattern p = Pattern.compile("^ABC");

Matcher m1 = p.matcher("ABC123");

System.out.println(m1.find());
```

true

1ABC23

123ABC

false

Регулярные выражения в Java. 10. Пример ABC$ с конца

```
Pattern p = Pattern.compile("ABC$");

Matcher m1 = p.matcher("123ABC");

System.out.println(m1.find());
```

true

1ABC23

ABC123

false

Регулярные выражения в Java. 11. Пример A|B или

```
Pattern p = Pattern.compile("A|B");

Matcher m1 = p.matcher("BC");

System.out.println(m1.find());
```

→ **true**

123C → **false**

Методы

Регулярные выражения в Java. replaceFirst и replaceAll

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

    private static String REGEX = "Собака";
    private static String INPUT = "Собака говорит мяу.";
    private static String REPLACE = "Кот";

    public static void main(String[] args) {
        Pattern p = Pattern.compile(REGEX);

        // получение matcher объекта
        Matcher m = p.matcher(INPUT);
        INPUT = m.replaceAll(REPLACE);
        System.out.println(INPUT);
    }}
```

```
Кот говорит мяу.
```

http://proglang.su/java/regular-expressions

Регулярные выражения в Java. Группы сбора

- Группы сбора представляют способ обращения с несколькими символами как с одной единицей.
- Они создаются путем размещения символов, которые предстоит сгруппировать, в серии круглых скобок.
- К примеру, регулярное выражение (dog) составляет отдельную группу, содержащую буквы "d", "o", и "g".

Регулярные выражения в Java. Группы сбора

Группы сбора нумеруются посредством определения числа открывающих круглых скобок слева направо.

Так, в выражении ((A)(B(C))) присутствуют четыре подобные группы:

1. ((A)(B(C)))
2. (A)
3. (B(C))
4. (C)

Регулярные выражения в Java. Группы сбора

Для определения числа групп, представленных в выражении, вызвать метод **groupCount** на объекте класса matcher в Java.

Метод **groupCount** извлекает число типа int, отображающее количество групп сбора, представленных в сопоставляемом шаблоне.

Также имеется специальная группа, группа 0, которая во всех случаях представляет выражение в полном виде. Данная группа не включается в сумму, представленную методом groupCount.

Регулярные выражения в Java. Группы сбора

```
результат: Крещение Руси произошло в 988 году! Не так ли?
результат: Крещение Руси произошло в 98
результат: 8
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

        public static void main( String args[] ) {
          // Строка для сканирования, чтобы найти шаблон
          String str = "Крещение Руси произошло в 988 году! Не так ли?";
          String pattern = "(.*)(\\d+)(.*)";

          // Создание Pattern объекта
          Pattern r = Pattern.compile(pattern);

          // Создание matcher объекта
          Matcher m = r.matcher(str);
          if (m.find( )) {
             System.out.println("результат:  " + m.group(0));
             System.out.println(" результат : " + m.group(1));
             System.out.println(" результат : " + m.group(2));
          }else {
             System.out.println("НЕ СОВПАДАЕТ");
          }
       }
}
```

http://proglang.su/java/regular-expressions

Регулярные выражения в Java. Количество совпадений

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches {

    private static final String REGEX = "\\bcat\\b";
    private static final String INPUT = "cat cat cat cattie cat";

    public static void main(String args[]) {
        Pattern p = Pattern.compile(REGEX);
        Matcher m = p.matcher(INPUT); // get a matcher object
        int count = 0;

        while (m.find()) {
            count++;
            System.out.println("Match number " + count);
            System.out.println("start(): " + m.start());
            System.out.println("end(): " + m.end());
        }
    }
}
```

Регулярные выражения в Java. Количество совпадений

```
private static final String REGEX = "\\bcat\\b";
private static final String INPUT = "cat cat cat cattie cat";
```

```
Match number 1
start(): 0
end(): 3

Match number 2
start(): 4
end(): 7

Match number 3
start(): 8
end(): 11

Match number 4
start(): 19
end(): 22
```

Регулярные выражения в Java. Методы matches и lookingAt

```
Текущее регулярное выражение: Pro
Текущие входные данные: ProgLang
lookingAt(): true
matches(): false
```

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

    private static final String REGEX = "Pro";
    private static final String INPUT = "ProgLang";
    private static Pattern pattern;
    private static Matcher matcher;

    public static void main( String args[] ) {
        pattern = Pattern.compile(REGEX);
        matcher = pattern.matcher(INPUT);

        System.out.println("Текущее регулярное выражение: " + REGEX);
        System.out.println("Текущие входные данные: " + INPUT);

        System.out.println("lookingAt(): " + matcher.lookingAt());
        System.out.println("matches(): " + matcher.matches());
    }
}
```

Оба метода matches и lookingAt направлены на попытку поиска соответствия вводимой последовательности с шаблоном. Разница, однако, заключается в том, что для метода matches требуется вся вводимая последовательность, в то время как lookingAt этого не требует.

Регулярные выражения. appendReplacement и appendTail

-ProgLang-ProgLang-ProgLang-

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

    private static String REGEX = "а*д";
    private static String INPUT = "ааdProgLangaaaдProgLangадProgLangд";
    private static String REPLACE = "-";
    public static void main(String[] args) {

        Pattern p = Pattern.compile(REGEX);

        // получение matcher объекта
        Matcher m = p.matcher(INPUT);
        StringBuffer sb = new StringBuffer();
        while(m.find()) {
            m.appendReplacement(sb, REPLACE);
        }
        m.appendTail(sb);
        System.out.println(sb.toString());
    }
}
```

http://proglang.su/java/regular-expressions

PatternSyntaxException

Методы класса PatternSyntaxException

PatternSyntaxException представляет непроверяемое исключение, которое отображает синтаксическую ошибку в шаблоне регулярного выражения.

public String getDescription() - представляет описание ошибки.

public int getIndex() - представляет индекс ошибки.

PatternSyntaxException

public String getPattern() - представляет шаблон регулярного выражения, содержащего ошибку.

public String getMessage() - производит возврат многострочной строки, содержащей описание синтаксической ошибки и ее индекс, ошибочный образец регулярного выражения, а также визуальную индикацию индекса ошибки в шаблоне.

Метасимволы

Метасимволы для поиска символьных классов

Метасимвол	Назначение
\d	цифровой символ
\D	нецифровой символ
\s	символ пробела
\S	непробельный символ
\w	буквенно-цифровой символ или знак подчёркивания
\W	любой символ, кроме буквенного, цифрового или знака подчёркивания
.	любой символ

Метасимволы для поиска символов редактирования текста

Метасимвол	Назначение
\t	символ табуляции
\n	символ новой строки
\r	символ возврата каретки
\f	переход на новую страницу
\u 0085	символ следующей строки
\u 2028	символ разделения строк
\u 2029	символ разделения абзацев

Метасимволы для группировки символов

Метасимвол	Назначение
[абв]	любой из перечисленных (а,б, или в)
[^абв]	любой, кроме перечисленных (не а,б, в)
[a-zA-Z]	слияние диапазонов (латинские символы от a до z без учета регистра)
[a-d[m-p]]	объединение символов (от a до d и от m до p)
[a-z&&[def]]	пересечение символов (символы d,e,f)
[a-z&&[^bc]]	вычитание символов (символы a, d-z)

Метасимволы для обозначения количества символов

Метасимвол / квантификатор	Назначение
?	один или отсутствует
*	ноль или более раз
+	один или более раз
{n}	n раз
{n,}	n раз и более
{n,m}	не менее n раз и не более m раз

Квантификатор всегда следует после символа или группы символов

String regex="**\\d{3,}**";

Режимы квантификаторов

жадный и ленивый

Жадный режим +

В жадном режиме (по умолчанию) квантификатор повторяется столько раз, сколько это возможно.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {

        String text = "Егор Алла Александр";
        Pattern pattern = Pattern.compile("A.+a");
        Matcher matcher = pattern.matcher(text);
        if (matcher.find()) {
            System.out.println(text.substring(matcher.start(), matcher.end()));
        }    }}
```

Алла Алекса

Ленивый режим ?

«Ленивый» режим противоположен «жадному». Он означает: «повторять квантификатор наименьшее количество раз».

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {

        String text = "Егор Алла Александр";
        Pattern pattern = Pattern.compile("A.+?a");
        Matcher matcher = pattern.matcher(text);
        if (matcher.find()) {
            System.out.println(text.substring(matcher.start(), matcher.end()));
        }   }}
```

Алла

Метасимволы для поиска совпадений границ строк

Метасимвол	Назначение
^	начало строки
$	конец строки
\b	граница слова
\B	не граница слова
\A	начало ввода
\G	конец предыдущего совпадения
\Z	конец ввода
\z	конец ввода

Создание регулярных выражений в Java

Чтобы создать регулярное выражение Java, нужно:

1. написать его в виде строки с учётом синтаксиса регулярных выражений;
2. скомпилировать эту строку в регулярное выражение;

Работа с регулярными выражениями в любой Java-программе начинается с создания объекта класса **Pattern**.

```
// подключение библиотеки regex.Pattern
import java.util.regex.Pattern;
…
// поиск между x и z включительно
Pattern pattern1 = Pattern.compile ("[x-z]+");

// без учета регистра
Pattern pattern1 = Pattern.compile ("[x-z]+", Pattern.CASE_INSENSITIVE);
```

Спасибо за внимание!