Класс StringUtils из Apache Commons

JSP & Servlets
20 уровень, 2 лекция

ОТКРЫТА

Знакомство со StringUtils

StringUtils — наиболее используемый класс Apache Commons. Он содержит различные утилиты и методы, которые помогают разработчикам избегать написания шаблонного или просто громоздкого кода для базовых операций.

Многие методы в классе `StringUtils` имеют свои эквиваленты в **java.lang.String**, но, в отличие от методов **java.lang.String**, являются безопасными для работы с null. Это означает, что исключение **NullPointerException** не выбрасывается в самый неожиданный момент.

Apache Commons содержит ряд методов, и мы рассмотрим некоторые из наиболее часто используемых.

Список методов StringUtils:

`isEmpty()`	Проверяет, пустая ли строка
`equals()`	Сравнивает строки
`compare()`	Сравнивает строки
`indexOf()`	Поиск подстроки в строке
`lastIndexOf()`	Поиск подстроки в строке
`contains()`	Проверяет вхождение подстроки в строку
`containsIgnoreCase()`	Проверяет вхождение подстроки в строку, игнорируя регистр
`containsAny()`	Проверяет вхождение подстроки в любом месте строки
`containsNone()`	Проверяет, нет ли вхождения подстроки в любом месте строки
`containsOnly()`	Проверяет вхождение подстроки в строку
`substring()`	Получение подстроки
`split()`	Разбиваем строку на подстроки
`join()`	Объединяем подстроки
`remove()`	Удаляем подстроку
`replace()`	Заменить подстроку
`countMatches()`	Считаем количество совпадений

StringUtils.isEmpty() и StringUtils.isBlank()

Оба метода используются для проверки того, содержит ли строка какой-либо текст. Они возвращают true, если строка действительно пуста. Кроме того, `isBlank()` также вернет *true*, если строка содержит только пробелы.

У них также есть свои обратные методы: `isNotEmpty()` и `isNotBlank()`.

Давай посмотрим, как ты можешь использовать `isEmpty()` вместе с его аналогом `java.lang.String.isEmpty()`, а также `isBlank()`:

```
String nullValue = null;
String emptyValue = "";
String blankValue = "\n \t   \n";

if(StringUtils.isEmpty(emptyValue)) {
   System.out.println("emptyValue is emptyValue");
}

if(StringUtils.isBlank(blankValue)) {
   System.out.println("blankValue is blankValue");
}

if(!nullValue.isEmpty()) {
   System.out.println("nullString isn't null");
}
```

Здесь три переменных типа String. Одна указывает на ***null***, вторая не является ***null***, но не имеет содержимого (пустая строка), а третья не пустая, но при печати выдаст пустой результат.

Выполнение этого кода приводит к:

```
emptyValue is emptyValue
blankValue is blankValue
Exception in thread "main" java.lang.NullPointerException
```

Метод `isEmpty()`, встроенный в **java.lang.String**, небезопасный для ***null***. Ты легко получишь **NullPointerException**, если попытаешься проверить, пуста ли она, так как вызовешь метод по ссылке ***null***. Нужно будет заранее проверить, является ли ссылка нулевой:

```
String nullValue = null;
String emptyValue = "";
String blankValue = "\n \t   \n";

if(StringUtils.isEmpty(emptyValue)) {
   System.out.println("emptyValue is emptyValue");
}

if(StringUtils.isBlank(blankValue)) {
   System.out.println("blankValue is blankValue");
}

if(nullValue != null && !nullValue.isEmpty()) {
   System.out.println("nullString isn't null");
}
```

Теперь это приводит к:

```
emptyValue is emptyValue
blankValue is blankValue
```

И если мы протестируем эти методы на `nullString` :

```
String nullValue = null;

if(StringUtils.isEmpty(nullValue)) {
   System.out.println("nullValue is emptyValue");
}

if(StringUtils.isBlank(nullValue)) {
   System.out.println("nullValue is blankValue");
}
```

То получим:

```
nullValue is emptyValue
nullValue is blankValue
```

Методы `StringUtils` безопасны для ***null*** и дают ожидаемый результат, даже если в них передали ***null***.

StringUtils.equals()

Этот метод сравнивает две строки и возвращает *true*, если они идентичны или обе ссылки указывают на ***null***, но имей ввиду, что этот метод чувствителен к регистру.

Давай посмотрим, как он работает:

```
System.out.println(StringUtils.equals(null, null));
System.out.println(StringUtils.equals(null, "какая-то информация"));
System.out.println(StringUtils.equals("какая-то информация", null));
System.out.println(StringUtils.equals("какая-то информация",  "какая-то информация"));
System.out.println(StringUtils.equals("какая-то дополнительная информация", "какая-то информация"));
```

Результат:

```
true
false
false
true
false
```

Для сравнения метода `equals()` из `StringUtils` с `java.lang.String.equals()`:

```
String nullValue = null;

System.out.println(StringUtils.equals(nullValue, null));
System.out.println(StringUtils.equals(nullValue, "какая-то информация"));

System.out.println(nullValue.equals(null));
System.out.println(nullValue.equals("какая-то информация"));
```

Это опять привело тебя к:

```
true
false
Exception in thread "main" java.lang.NullPointerException
```

Опять же, вызов метода для ссылки ***null*** приводит к исключению **NullPointerException**, и нужно будет заранее проверить, является ли ссылочная переменная ***null***, перед ее использованием.

StringUtils.compare()

Объявление этого метода выглядит следующим образом:

```
public static int compare(final String str1, final String str2)
```

Этот метод сравнивает две строки лексикографически, как это делает метод `java.lang.String.compareTo()`, возвращая:

- 0, если str1 равно str2 (или оба равны null)
- Значение меньше, чем 0, если str1 меньше, чем str2
- Значение, большее, чем 0, если str1 больше, чем str2

Лексикографический порядок – это порядок по словарю. Давай посмотрим, как мы можем использовать это в нашей программе:

```
System.out.println(StringUtils.compare(null, null));
System.out.println(StringUtils.compare(null , "javaRush"));
System.out.println(StringUtils.compare("javaRush", null));
System.out.println(StringUtils.compare("javaRush", "JAVARUSH"));
System.out.println(StringUtils.compare("javaRush", "javaRush"));
```

Получаем:

```
0
-1
1
32
0
```

Примечание: значение ***null*** считается меньшим, чем значение, отличное от ***null***. Два значения ***null*** считаются равными.

Проверяем, содержит ли строка другую подстроку

Для этого в `StringUtils` есть 5 методов:

- `contains()`
- `containsIgnoreCase()`
- `containsAny()`
- `containsNone()`
- `containsOnly()`

Метод `contains()` возвращает *true* или *false* в зависимости от того, содержится ли последовательность поиска в другой последовательности или нет.

Если в такой метод передать ***null***, то он вернет *false*. Если передать не ***null***, то метод просто вызовет `java.lang.String.indexOf(String str)` у передаваемого объекта.

Примеры:

```
String value = "JavaRush is cool";

System.out.println(StringUtils.contains(null, "a"));
System.out.println(StringUtils.contains(value, "JavaRush"));
System.out.println(StringUtils.contains(value, "C++"));
System.out.println(StringUtils.contains(value, "javarush"));
```

Метод чувствителен к регистру, поэтому последний вызов также вернет *false*:

```
false
true
false
false
```

Метод `containsAny()` возвращает *true*, если строка, переданная первым аргументом, содержит хотя бы одну из подстрок, переданную 2-N аргументами.

Пример:

```
String value = "JavaRush is cool";
System.out.println(StringUtils.containsAny(value, "cool", "c00l", "bro", "hello"));
```

Выведет на экран:

```
true
```

Этот метод также чувствителен к регистру.

Метод containsNone()

Когда нужно проверить, что определенная строка не содержит ничего из списка, можно воспользоваться методом `containsNone()`. Первым параметром в него передается строка, а следующие параметры – это строки, которых не должно быть в целевой стоке.

Пример:

```
String s = "JavaRush is cool";
System.out.println(StringUtils.containsNone(s, 'g', 'a'));
```

Вывод в консоль:

```
false
```

Работа с подстроками

Работа с подстроками похожа на работу методов класса `String`:

```
substring(String str, int start)
substring (String str, int start, int end)
```

Эти методы возвращают подстроку из строки `str`. Строка задается двумя индексами: **start** и **end**. И как принято в Java, последний символ диапазона – **end-1**. В чем же преимущество этих методов?

Если передать в такой метод ***null***, он просто вернет ***null***, а не кинет исключение. Эти методы поддерживают отрицательные значение индексов. При этом строка рассматривается как замкнутая петля. После последнего символа идет первый и т.д.

Давай посмотрим, как мы можем его использовать:

```
System.out.println(StringUtils.substring("lets java", 2, 6));
System.out.println(StringUtils.substring("lets java", -8));
System.out.println(StringUtils.substring(null, 3));
```

Выполнение приведенного выше кода дает нам:

```
ts j
ets java
```

```
null
```

StringUtils.split()

Метод, который позволяет разбить строку на подстроки, используя специальный символ-разделитель. Если такой есть в целевой строке, то метод вернет массив подстрок. Если символа нет – вернется пустой массив. Ну а если в метод передать ***null***, он вернет ***null***. Давай рассмотрим этот код и работу метода:

```
String myData = "Address, City, State, Zip, Phone, Email, Password";

System.out.println(Arrays.toString(StringUtils.split(myData, ',')));
System.out.println(Arrays.toString(StringUtils.split(null, '.')));
System.out.println(Arrays.toString(StringUtils.split("", '.')));
```

Результат:

```
[Address,  City,  State,  Zip,  Phone,  Email,  Password]
null
[]
```

StringUtils.join()

Метод `join()` позволяет склеить массив строк в одну строку. При этом в него можно передать специальный символ-разделитель, которые будет добавлен между подстроками в результирующей строке. А если в метод передать ***null***, то он вернет ***null***.

Этот метод представляет собой прямую противоположность методу `split()`. Давай рассмотрим этот простой пример:

```
String myData = "Address, City, State, Zip, Phone, Email, Password";

String[] myString =  StringUtils.split(myData, ',');
System.out.println(StringUtils.join(myString, '-'));
```

Выполнение приведенного выше кода дает нам:

```
Address- City- State- Zip- Phone- Email- Password
```

StringUtils.replace()

Ищет строку внутри строки, находит ее, если она существует, и заменяет все ее вхождения новой строкой.

Объявление этого метода выглядит следующим образом:

```
public static String replace(final String text, final String searchString, final String replacement)
```

Если строка поиска не найдена в тексте, то ничего не произойдет и текст останется прежним. Следуя той же логике, если текст равен ***null***, этот метод возвращает ***null***. Если ты ищешь ***null***-строку или заменяешь подстроку на ***null***, то метод вернет оригинальную строку.

Давай попробуем этот метод:

```
String value = "JavaRush is the best";
System.out.println(StringUtils.replace(value, "best", "cool"));
```

Результат:

```
JavaRush is the cool
```

- Guide to Apache Commons' StringUtils Class in Java

- Руководство по классу StringUtils Apache Commons в Java

‹ Предыдущая лекция

Следующая лекция ›

+11

Комментарии популярные новые старые

JavaCoder

Введите текст комментария

У ЭТОЙ СТРАНИЦЫ ЕЩЕ НЕТ НИ ОДНОГО КОММЕНТАРИЯ

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

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store