Пост

Лента постов Все группы Мои группы

Alexander Klimenko

13 уровень

Харьков

1 мая 2018 20:19 12999 15

Экранирование символов в Java

Пост из группы Random

995227 участников

Вы в группе

Что-то пошло не так! Данная статья писалась как выполнение тестового задания на должность в команде JavaRush. И писалась как полноценная лекция. За счет этого гарантирую вам качество и количество полезных знаний скопившихся в этом посте. Помимо практической и теоретической информации, в статье присутствуют интересные факты, о которых вы могли даже не догадываться!

Hello World!

Экранирование символов — это очень интересное и необходимое техническое решение. Необходимость в экранировании символов сыграло важную роль в истории всей индустрии программирования.

В этой статье мы поговорим о том, что такое экранирование символов, почему появилась потребность их экранировать, и как экранирование символов реализовано в Java. В статье будут приведены примеры и интересные факты, связанные с темой экранирования символов. Приятного чтения!

Вся информация в компьютерной системе представлена в виде текста, который на более низком уровне представлен байтами. Когда мы пишем письмо или сообщение, мы набираем текст, который будет понятен для человека. Когда же мы пишем код в IDE, мы набираем текст, который сможет разобрать компилятор. В java текст можно представить в виде типа **String**, для обозначения данных которого используются управляющие символы - парные кавычки.

```
1  String str = "Hello World!";
```

С текстом “Hello World!” никаких проблем не возникает, но что если этот же текст необходимо выделить прямой речью? Воспользовавшись правилами грамматики становится ясно, что текст “Hello World!”, помимо управляющих символов от типа String, требуется поместить в кавычки прямой речи.

```
1  String str = "Java said, "Hello World!"";
```

Такой вариант будет нерабочим, т.к. компилятор попросту не поймет в какой же момент заканчивается инициализация переменной str.

Для решения этой и подобных ей проблем было придумано **экранировать символы**, то есть менять **управляющие символы** на так называемые управляющие последовательности, известные также, как **escape-последовательности**. Ниже приведен список действующих escape-последовательностей java для использования в строках.

\t — Символ табуляции (в java - эквивалент четырех пробелов);
\b — Символ возврата в тексте на один шаг назад или удаление одного символа в строке (backspace);
\n — Символ перехода на новую строку;
\r — Символ возврата каретки;
\f — Прогон страницы к началу следующей страницы;

\’ — Символ одинарной кавычки;

\” — Символ двойной кавычки;

\\ — Символ обратной косой черты (\).

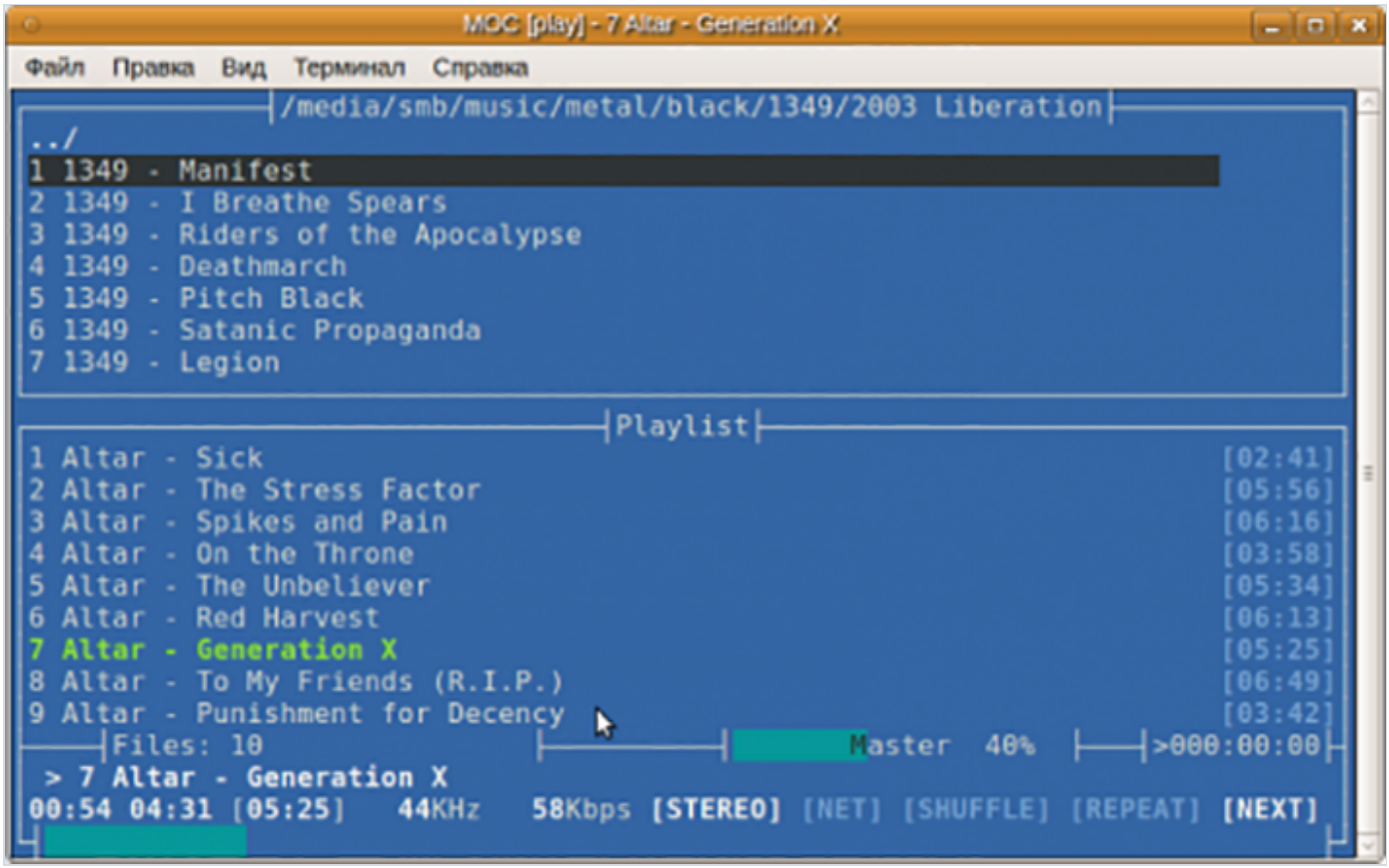
Теперь давайте выделим прямую речь в нашей фразе так, чтобы компилятор смог без проблем разобрать написанное.

```
1  String str = "Java said, \"Hello World!\"";
```

Таким образом, написанный текст понятен и компилятору и человеку, если содержимое переменной str вывести на экран.

Мы разобрались с тем, что такое экранирование символов и для чего оно нужно. И даже экранировали символ двойной кавычки! Приступим к разбору оставшихся escape-последовательностей.

Символ табуляции в строке обозначается escape-последовательностью **\t** и является аналогом четырех пробелов. Однако, если длина строки, состоящая из четырех пробелов будет равна длине четырех символов, то длина строки с символом табуляции будет равна одному. Символ табуляции часто используется для построения таблиц или [псевдографических](#) элементов интерфейса, т.к. это удобнее записи четырех пробелов. Ниже пример псевдографического интерфейса.



Среди всех escape-последовательностей символ **\b** пожалуй самый интересный, ведь он позволяет нам удалить последний символ в строке вывода, подобно, если бы мы стирали его нажатием клавиши **backspace**.

```
System.out.print("2 + 2 = 5"); // На экране отображается 2 + 2 = 5
System.out.print("\b");// На экране отображается 2 + 2 =
System.out.print("4");// На экране отображается 2 + 2 = 4
```

Символы **\n** и **\r** имеют общую историю - рассмотрим их вместе. С символом переноса строки **\n** вы могли встречаться ранее. Например, если метод *println()* выводит информацию так, что следующий вывод будет с новой строки, то метод *print()* не выполняет переноса строки после вывода, но если добавить в конец вывода символ **\n**, то перенос строки будет выполнен.

```
System.out.print("Следующий вывод будет с новой строки\n");
System.out.println("Следующий вывод будет с новой строки");
```

Символ возврата каретки **\r** позволяет нам вернуть курсор к началу строки вывода и отображать новую информацию так, как будто ранее в этой строке ничего не было.

```
System.out.print("Текст который необходимо переписать.");//На экране о
System.out.print('\r');//На экране пусто
System.out.print("Новый текст.");//На экране отображается "Новый текст
```

На самом деле возврат каретки берет свое начало еще со времен, когда текст печатали на печатных машинках. Чтобы выполнить перенос строки, необходимо было передвинуть каретку и опустить рычажок (части механизма печатной машинки), после чего будет выполнен перенос строки. Если же рычажок не опустить, то можно было продолжать печатать в той же строке. Что мы и наблюдаем, выводя символ \r. В связи с этим, когда программист хотел выполнить перенос строки, он, по привычке, в конце вывода выполнял последовательность из символов **\r\n**. Когда эра печатных машинок подошла к концу, появилось поколение программистов, которые все еще использовали эту последовательность, хотя сами за печатной машинкой никогда не работали. Они часто забывали в каком порядке необходимо было выполнить данную последовательность — **\r\n** или **\n\r**. Тогда им на помощь пришло проверочное слово ***return***, где наглядно виден порядок вывода этих символов. Однако позже при разработке программного обеспечения на первые версии Windows, после MS-DOS, программисты вынуждены были использовать последовательность **\r\n**. Сейчас же об этом можно не беспокоиться и для переноса строки использовать только символ **\n**.

Вернемся еще раз в прошлое, примерно в 80-е годы. Именно тогда символ прогона страницы **\f** к началу следующей страницы имел популярность. В то время были большие линейные принтеры, для работы с которыми необходимо было писать программный код, содержащий что и как принтер должен напечатать. И для обозначения, что текст необходимо начать печатать с новой страницы использовался символ **\f**. В наше же время этот символ давно утратил свою актуальность, и навряд ли вы с ним когда-либо столкнетесь. Размеры линейного принтера весьма внушительны.

С символами \’ и \\ все точно также как и с экранированием двойной кавычки, пример был в начале статьи. Экранировать одинарную кавычку придется, например, для инициализации типа char одинарной кавычкой.

```
1  char ch = '\'';
```

Экранировать символ обратной косой черты необходимо для указания, что последующий символ не будет являтся частью escape-последовательности.

```
1  System.out.println("\\n - escape-последовательность переноса строки");
2  // Вывод: \n - escape-последовательность переноса строки
```

На практике же экранировать обратный слеш чаще приходится при работе с путями:

```
1  System.out.println("It's Java string: \"C:\\Program Files\\Java\\jdk1.
2  // Вывод: It's Java string: "C:\Program Files\Java\jdk1.7.0\bin"
```

Я подчеркнул, что данные escape-последовательности употребляются в строках (строковых литералах), т.к. остальная их часть используется для описания регулярных выражений класса **Pattern** и не относится к теме данной статьи. Здесь можно ознакомиться со списком всех escape-последовательностей класса **Pattern**. Однако, стоит отметить, что регулярные выражения в том виде, в котором они есть сейчас, невозможно представить без использования escape-последовательностей не только в java, но и в других популярных языках программирования, например, PHP.

В java экранирование символов используется и в форматировании строк. Например, задавая формат строки для отображения символа процента, необходимо продублировать символ процента - **%%**, иначе получим ошибку, а IDE будет предлагать дописать процент.

```
1  System.out.printf("Процент жирности молока : %d%%", 10);
2  // Процент жирности молока : 10%
```

На этом статья подходит концу. Надеюсь, вы узнали много нового об экранировании символов, и о том, как применять это на практике. Экранирование символов присуще многим языкам программирования. В java, как и в других си-подобных языках данная технология реализована почти одинаково. Поэтому, полученные вами знания из этой статьи вполне могут пригодится не только в java. Спасибо за внимание и удачи в обучении!
