# 字符串

## 三单引号字符串

**# 1. 三双引号**

```groovy

def name = 'Groovy'

def template = """

    Dear Mr ${name},

    You're the winner of the lottery!

    Yours sincerly,

    Dave

"""

println template

assert template.toString().contains('Groovy')

// 执行结果为：

//     Dear Mr Groovy,

//     You're the winner of the lottery!

//     Yours sincerly,

//     Dave

```

**# 2. 三单引号字符串**

三单引号字符串是一列被三个单引号包围的字符，是普通的java.lang.String。不支持插值。

```groovy

def str = '''groovy'''

assert str.class == java.lang.String

```

三单引号字符串是`多行`的。你可以`使字符串内容跨越行边界`，不需要将字符串分割为一些片段，不需要连接，或换行转义符：

```groovy

def str = '''

one

two

three'''

//需要注意的是，这个字符串每一次换行都包含了一个换行转义符：

assert str.contains("\n") == true

assert str.size() == 14

//可以通过使用反斜杠换行符剥离该字符

def str = '''\

one\

two\

three'''

assert str.contains("\n") == false

assert str.size() == 11

```

**## 参考**

1. https://blog.csdn.net/dora\_310/article/details/52895835

2. [Groovy入门](https://blog.51cto.com/damon188/2148647?source=dra)

## 2.字符串方法

字符串方法

Groovy通过包含附加方法的方法扩展了JDK中的类：center、getAt、leftShift等等。

```groovy

//String center(Number numberOfChar)  返回一个长度为numberOfChar，其左边和右边均使用空格的填充的新字符串。

def str = "demo"

assert str.center(6) == " demo "

assert str.center(6).size() == 6

assert str.center(5) ==  "demo "    //如果numberOfChar比字符串长度大1.那么在原字符串后添加一个空格。

//String center(Number numberOfChar，String padding)  返回一个长度为numberOfChar，其左边和右边均使用padding的填充的新字符串，与上一个一致：如果第一个参数比字符串长度大1，在元字符串后添加padding

assert str.center(5,"a") == "demoa"

assert str.center(7,"ab") == "ademoab"      //先在元字符串后添加padding的前一个字符，再在原字符串前添加padding的前一个字符。然后在新字符串后添加padding的第二个字符，以此类推。

//int compareToIgnoreCase(String str) 按字典大小比较两个字符串，忽略大小写，返回他们的顺序差值

def str = "a"

assert str.compareToIgnoreCase("a") == 0    //相同返回0

assert str.compareToIgnoreCase("A") == 0    //忽略大小写

assert str.compareToIgnoreCase("c") == -2   //返回差值

//Boolean equalsIgnoreCase(String str) 判断两个字符串是否相等，忽略大小写

//String getAt(int index)   字符串的下标运算符

assert "abcdefg".getAt(2) == "c"

assert "abcdefg"[2] == "c"

assert "abcdefg".getAt(1..2) == "bc"    //String getAt(Range range)

assert "abcdefg"[1..2] == "bc"

//Int indexOf(String str) 返回给定子字符串在当前字符串中首次出现的索引值

assert "abcdefg".indexOf("b") == 1

assert "abcd".indexOf("g") == -1    //如果原字符串中不存在给定子字符串就返回-1

//StringBuffer leftShift(Object value)  重载<< 使两个字符串相加，返回一个新的字符串

assert "ab".leftShift("cd").class ==  java.lang.StringBuffer

assert "ab".leftShift("cd").toString() == "abcd"

//int length() / int size()   返回字符串的长度

assert "abcd".length() == 4

assert "abcd".size() == 4

//String concat(String str) 在字符串后添加str字符串

assert "ab".concat("12") == "ab12"

//Boolean endsWith(String suffix) 测试字符串是否以给定的后缀结尾

assert "demo1".endsWith("1") == true

//String minus(Object value) 删除字符串中value部分

assert "abcd".minus("bc") == "ad"

//String next()  按字典顺序增加给定字符串末尾字符顺序

assert "abcd".next() == "abce"

assert "abc1".next() == "abc2"

//String previous() 按字典顺序增加给定字符串末尾字符顺序

assert "abcd".previous() == "abcc"

assert "abc1".previous() == "abc0"

//String padLeft(Number numberOfCharacters) 与center()用法类似，在元字符串左边填充空格字符

assert "abcd".padLeft(5) == " abcd"

//String padLeft(Number numberOfCharacters，String padding) 与center()用法类似，在元字符串左边填充padding字符

assert "abcd".padLeft(5,"12") == "1abcd"

//String padRight(Number numberOfCharacters) 与padLeft()用法类似，在元字符串左边填充空格字符

assert "abcd".padRight(5) == "abcd "

//String padRight(Number numberOfCharacters，String padding) 与padLeft()用法类似，在元字符串左边填充padding字符

assert "abcd".padRight(5,"12") == "abcd1"

//String plus(Object valus) 字符串相加

assert "abcd".plus("123") == "abcd123"

//String reverse() 创建当前字符串的逆序字符串

assert "abcd".reverse() == "dcba"

//String substring(int beginIndex) 返回一个当前字符串的指定索引开始的子字符串

assert "abcd".substring(1) == "bcd"

//String substring(int beginIndex,int endIndex) 返回一个当前字符串的指定索引开始的子字符串

assert "abcd".substring(1,2) == "bc"

//Character toCharacter()

//Double toDouble()

//Float toFloat()

//Integer toInteger()

//Long toLong() 字符串类型转换

//List toList() 将指定的字符串转换成一个由单个字符组成的字符串列表

assert "abcd".toList() == ["a","b","c","d"]

//String toUpperCase() 将当前字符串对象的所有字符转换为大写

assert "abcd".toUpperCase() == "ABCD"

//String toLowerCase() 将当前字符串对象的所有字符转换为小写

assert "ABCD".toUpperCase() == "abcd"

//List tokenize()  使用空格作为字符串的分隔符

//List tokenize(String token) 使用指定的token参数作为字符串的分隔符

//String[] split(String regex) 使用与给定的正则表达式相匹配的子字符串将字符串分隔为多个字符串

// Boolean matches(String regex) 测试字符串是否匹配给定子字符串

```

**## 参考:**

1. https://blog.csdn.net/dora\_310/article/details/52895835

## 3.tokenize() vs split()

**## 1.split()返回string[]， tokenize()返回list**

**## 2.tokenize()忽略空字符串**

String testString = 'hello brother'

assert testString.split() instanceof String[]

assert ['hello','brother']==testString.split() //split with no arguments

assert['he','','o brother']==testString.split('l')

assert testString.tokenize() instanceof List

assert ['hello','brother']==testString.tokenize() //tokenize with no arguments

assert ['he','o brother']==testString.tokenize('l')

**## 3.tokenize()使用字符串内的所有字符**

String  testString1='hello world'

assert ['hel',' world']==testString1.split('lo')

assert ['he',' w','r','d']==testString1.tokenize('lo')

**## 4.split()可以使用正则表达式**

String testString2='hello world 123 herload'

assert['hello world ',' herload']==testString2.split(/\d{3}/)

**## 参考**

1. https://blog.csdn.net/dora\_310/article/details/52895835

## 4.双引号字符串

双引号字符串

双引号字符串是一列被双引号包围的字符，如果没有插值表达式，双引号字符串就是普通的java.lang.String。如果有插值存在则是groovy.lang.GString实例：

任何Groovy表达式可以在所有字符文本进行插值，除了单引号和三单引号字符串。插值是使用占位符上的字符串计算值替换占位符的操作。占位符表达式是被包围，或前缀为包围，或前缀为的表达式。当GString被传递给一个带有一个String参数的方法时，占位符的表达式被计算值，并通过调用表达式的toString()方法以字符串形式表示。

这里是一个占位符引用局部变量的字符串：

```java

def name = 'Groovy'

def greeting = "Hello ${name}"

assert greeting.toString() == 'Hello Groovy'

//而且任何Groovy表达式是合法的，正如我们在示例中使用算数表达式所见一样:

def sum = "The sum of 2 and 3 equals ${2 + 3}"

assert sum.toString() == 'The sum of 2 and 3 equals 5'

//除了${}占位符以外，也可以使用$作为表达式前缀：

def person = [name: 'Guillaume', age: 36]

assert "$person.name is $person.age years old" == 'Guillaume is 36 years old'

//空${}占位符等于null

assert "demo${}" == "demonull"

//如果在GString中你需要转义$或${}占位符，使它们不出现插值，那么你只需要使用反斜杠字符转义美元符号：

assert '${name}' == "\${name}"

```

当占位符包含一个箭头时：${->}。表达式实际上是一个闭包表达式：

name = "zhangsan"

assert "${->name}" == "zhangsan"

**## 参考：**

1. https://blog.csdn.net/dora\_310/article/details/52895835

## 5.GString和String的hascode不同

GString和String的hascode不同

虽然插值字符串可以代替普通Java字符串，字符串相同的两者还是有区别：它们的hashCode是不同的。普通Java字符串是不可变的，而一个GString依赖于插入的值，它的String是可变的。即使有相同的字符串结果，GString和String也没有相同的hashCode。

def key = "a"

def m = ["${key}": "letter ${key}"]  //map被一个初始化键值对创建，其键值是GString

assert m["a"] == null   //当我们尝试使用String键值获取值时，我们并没获取对应值，因为String和GString有不同的hashCode

**## 参考：**

1. https://blog.csdn.net/dora\_310/article/details/52895835

## 6.延迟加载

延迟加载

嵌入的闭包表达式仅仅允许携带0或者1个参数。

闭包作为表达式比普通插值表达式有一个很有用的特性：延迟加载。

 def b = 'old'

 def value1 = "${b}"

 def value2 = "${->b}"

 assert value1 == "old"

 assert value2 == "old"

 b = "new"

 assert value1 == "old"    //使用纯插值表达式，结果在GString创建时结合

 assert value2 == "new"    //但使用闭包表达式，GString被强转为Sring时，闭包被调用，并产生包含新数值的更新字符串。

**## 参考：**

1. https://blog.csdn.net/dora\_310/article/details/52895835

## 7.大小写

.toLowerCase()

.toUpperCase()

## 8.字符串截取substring

http://findsrc.com/groovy/detail/8582

字符串截取需要经常使用到，在groovy中我们可以通过多种方式截取字符串。

```groovy

// 1、使用substring方法，同java中的字符串截取

def log = "Exception on saving user with username:johntheripper"

def username = log.substring(log.lastIndexOf(":") + 1, log.length())

println username // johntheripper

def usernameWithoutEndIndex = log.substring(log.lastIndexOf(":") + 1)

println usernameWithoutEndIndex // johntheripper

def git = '/jtp/gateway.git'

   def username = git.substring(git.lastIndexOf("/") + 1, git.lastIndexOf("."))

   println(username) //gateway

// 2、使用subSequence方法，也是Java中的原生方法

def log = "Exception on saving user with username:johntheripper"

def username = log.subSequence(log.lastIndexOf(":") + 1, log.length())

println username // johntheripper

// 3、groovy风格的字符串截取

接下来我们可以了解到groovy风格的字符串截取。

def text1 = "My last character will be removed soon"

println text1[0..-2] // My last character will be removed soo

def text2 = "My first word will be removed soon";

println text2[3..-1] // first word will be removed soon

def text3 = "noos em daer lliw uoy ,tneitap eB"

println text3[-1..0] // Be patient, you will read me soon

可以看到，groovy风格的字符串截取既简单又容易理解，通过[]号方式定义截取方式。比如0..5代表着索引号从0到5的字符组成的字符串。索引号可以是负数，如果结束索引号为负数则表明从尾部开始截取。如果起始索引号为负数，则表明是反向截取，截取的字符串也是反序的。

// 4、使用getAt方法截取字符串

def text1 = "crazy fox jumps over lazy dog"

println text1.getAt(0..(text1.length() - 5)) // crazy fox jumps over lazy

def text2 = "keep calm and carry on"

println text2.getAt(-1..5) // no yrrac dna mlac

getAt同[]截取是一致的，也同样支持负数。

// 5、字符串减法操作

def text1 = "Sorry, I need to separate from you"

println text1 - " you" // Sorry, I need to separate from

def text2 = "Minus string usage"

println text2.minus(" usage") // Minus string

可以使用减法操作从字符串中去掉指定字符串。

在groovy中字符串的截取是非常方便的，你既可以像使用java那样去使用也可以使用groovy风格的语法截图字符串。

```

## 09.string类型转换成int

（转载）http://www.cnblogs.com/killbug/archive/2012/04/20/2459175.html

在Groovy语言中，系统就有API来帮助我们直接做这件事情，相当的简单。如，如果我们需要判断一个字符串是否为int型，则只需要写如下的代码：

```groovy

def num = '23'

println num.isInteger()

```

运行结果为：

true

如果是如下的代码：

      def num1 = '2.3'

      println num1.isInteger()

则运行结果为：

false

类似的方法还有：isDouble()、isFloat()、isLong()、isNumber()、isBigDecimal()、isBigInteger()等。

判断完了以后，如果该字符串是我们所需要的数字类型，则我们接下来需要将它们转化成相应的数字类型。

例如，我们如果需要把一个字符串转化为整型数字的话，就需要些如下的代码：

```groovy

int num = Integer.parseInt("12");

// 对于这样的代码，Groovy语言也进行了进一步的简化：

def num = '23'

println num.toInteger()

```

类似的方法还有：toBigDecimal()、toBigInteger()、toBoolean()、toDouble()、toFloat()等等。

// -----------------------------------------------------------------------

Interger.parseInt()和String.valueOf()将字符串类型转化为int类型，将int类型转化为字符串类型。

## Groovy Tip 1 ==、equals及其他 | 学步园

https://www.xuebuyuan.com/2088913.html

我们说基于Groovy语言的Java项目开发是一个敏捷的开发，除了我们使用Grails作为Web开发平台，Groovy语言本身就提供了很多敏捷的地方。

我们知道，在Java语言中，判断两个字符串的值是否相等，我们使用equals方法，而“==”作为判断两个对象引用是否相同的符号。事实证明，我们使用equals方法多过使用“==”符号，特别是在字符串的使用中。

在实际编码的过程中，形如：

**if**(abc1.equals(abc2))

     {

**……**

     }

这样的if语言特别多，但是我们每一次都得拼写equals方法，而不是简单的使用“==”符号。虽然Eclipse等IDE能够帮我们把equals方法拼写完整，但使用equals方法毕竟耽误了我们的时间，远远没有使用“==”符号敏捷。

Groovy语言考虑到了我们的这个实际的需求，果断的使用“==”符号代替了equals方法，来提供Groovy语言编程的敏捷性。

**def** abc1 = 'abc'

**def** abc2 = "abc"

**println** abc1 == abc2

打印的结果为true。

这样，我们就不必再**if**(abc1.equals(abc2))了，而可以直接使用：

**if**(abc1==abc2)

    {

**……**

    }

是不是方便了很多？

甚至判断Collection两个对象的内容是否相同也可以使用“==”符号，如：

**def** list1 = [1]

**def** list2 = [1]

**println** list1 == list2

你可以把上面的语句运行看看，结果为true。

两个Map对象的内容是否相同也可以使用“==”符号判断，如：

**def** map1 = ['name':'tom']

**def** map2 = ['name':'tom']

**println** map1 == map2

结果为true。但是你要运行下面的语句：

**def** map1 = ['name':'tom']

**def** map3 = ['name':'alice']

**println** map1 == map3

结果却是false了，原因很简单：map1和map2的key为“name”时，value却不同。

你可能要问，既然“==”符号是判断两个对象的内容是否相同，那么用什么来判断两个对象的引用是否相同呢？

其实很简单，在Groovy语言中，我们使用is方法来判断两个对象的引用是否相同。如：

**def** list1 = [1]

**def** list2 = [1]

**println** list1.is(list2)

你可能想象得到，上面的语句运行结果为false。

但是，如果是下面的语句：

**def** list1 = [1]

**def** list2 = [1]

     list2 = list1

**println** list1.is(list2)

聪明的你一定知道，运行结果为true。

但是你看看下面的语句：

**def** abc1 = 'abc'

**def** abc2 = "abc"

**println** abc1.is(abc2)

猜猜它的运行结果是什么？

是的，结果为true。

Stop!你刚刚不是说了，判断两个对象的引用是否相同使用“==”符号啊，现在abc1和abc2是两个引用啊。

不错，abc1和abc2是两个引用，但是是两个相同的引用，都指向同一个对象“abc”。在声明**def** abc1 = 'abc'时，系统会生成一个“abc”对象，同时将引用abc1指向该对象；但是，在声明**def** abc2 = "abc"时，系统不再生成新的“abc”对象，而是直接将引用abc2指向上面已经生成了的“abc”对象。所以abc1和abc2指向同一个对象。

如果你还没有想清楚，请结合下面的例子再想一想：

     String a1 = **new** String('a')

     String a2 = "a"

**println** a1.is(a2)

是的，结果为false。

对于new String（），系统永远为它生成新的对象。