**字符串的扩展**

**目录：**

**1、模板字符串**

**2、标签模板**

**3、模板字符串的限制**

**4、字符串新增方法**

**本章介绍 ES6 对字符串的改造和增强，下一章介绍字符串对象的新增方法。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**模板字符串**

**传统的 JavaScript 语言，输出模板通常是这样写的（下面使用了 jQuery 的方法）。**

**$('#result').append(**

**'There are <b>' + basket.count + '</b> ' +**

**'items in your basket, ' +**

**'<em>' + basket.onSale +**

**'</em> are on sale!'**

**);**

**上面这种写法相当繁琐不方便，ES6 引入了模板字符串解决这个问题。**

**$('#result').append(`**

**There are <b>${basket.count}</b> items**

**in your basket, <em>${basket.onSale}</em>**

**are on sale!**

**`);**

**模板字符串（template string）是增强版的字符串，用反引号（`）标识。它可以当作普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。**

**// 普通字符串**

**`In JavaScript '\n' is a line-feed.`**

**// 多行字符串**

**`In JavaScript this is**

**not legal.`**

**console.log(`string text line 1**

**string text line 2`);**

**// 字符串中嵌入变量**

**let name = "Bob", time = "today";**

**`Hello ${name}, how are you ${time}?`**

**上面代码中的模板字符串，都是用反引号表示。如果在模板字符串中需要使用反引号，则前面要用反斜杠转义。**

**let greeting = `\`Yo\` World!`;**

**如果使用模板字符串表示多行字符串，所有的空格和缩进都会被保留在输出之中。**

**$('#list').html(`**

**<ul>**

**<li>first</li>**

**<li>second</li>**

**</ul>**

**`);**

**上面代码中，所有模板字符串的空格和换行，都是被保留的，比如<ul>标签前面会有一个换行。如果你不想要这个换行，可以使用trim方法消除它。**

**$('#list').html(`**

**<ul>**

**<li>first</li>**

**<li>second</li>**

**</ul>**

**`.trim());**

**模板字符串中嵌入变量，需要将变量名写在${}之中。**

**function authorize(user, action) {**

**if (!user.hasPrivilege(action)) {**

**throw new Error(**

**// 传统写法为**

**// 'User '**

**// + user.name**

**// + ' is not authorized to do '**

**// + action**

**// + '.'**

**`User ${user.name} is not authorized to do ${action}.`);**

**}**

**}**

**大括号内部可以放入任意的 JavaScript 表达式，可以进行运算，以及引用对象属性。**

**let x = 1;**

**let y = 2;**

**`${x} + ${y} = ${x + y}`**

**// "1 + 2 = 3"**

**`${x} + ${y \* 2} = ${x + y \* 2}`**

**// "1 + 4 = 5"**

**let obj = {x: 1, y: 2};**

**`${obj.x + obj.y}`**

**// "3"**

**模板字符串之中还能调用函数。**

**function fn() {**

**return "Hello World";**

**}**

**`foo ${fn()} bar`**

**// foo Hello World bar**

**如果大括号中的值不是字符串，将按照一般的规则转为字符串。比如，大括号中是一个对象，将默认调用对象的toString方法。**

**如果模板字符串中的变量没有声明，将报错。**

**// 变量place没有声明**

**let msg = `Hello, ${place}`;**

**// 报错**

**由于模板字符串的大括号内部，就是执行 JavaScript 代码，因此如果大括号内部是一个字符串，将会原样输出。**

**`Hello ${'World'}`**

**// "Hello World"**

**模板字符串甚至还能嵌套。**

**const tmpl = addrs => `**

**<table>**

**${addrs.map(addr => `**

**<tr><td>${addr.first}</td></tr>**

**<tr><td>${addr.last}</td></tr>**

**`).join('')}**

**</table>**

**`;**

**上面代码中，模板字符串的变量之中，又嵌入了另一个模板字符串，使用方法如下。**

**const data = [**

**{ first: '<Jane>', last: 'Bond' },**

**{ first: 'Lars', last: '<Croft>' },**

**];**

**console.log(tmpl(data));**

**输出：**

**//<table>**

**//**

**// <tr><td><Jane></td></tr>**

**// <tr><td>Bond</td></tr>**

**//**

**// <tr><td>Lars</td></tr>**

**// <tr><td><Croft></td></tr>**

**</table>**

**如果需要引用模板字符串本身，在需要时执行，可以写成函数。**

**let func = (name) => `Hello ${name}!`;**

**func('Jack') // "Hello Jack!"**

**上面代码中，模板字符串写成了一个函数的返回值。执行这个函数，就相当于执行这个模板字符串了。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**标签模板**

**模板字符串的功能，不仅仅是上面这些。它可以紧跟在一个函数名后面，该函数将被调用来处理这个模板字符串。这被称为“标签模板”功能（tagged template）。**

**alert`hello`**

**// 等同于**

**alert(['hello'])**

**标签模板其实不是模板，而是函数调用的一种特殊形式。“标签”指的就是函数，紧跟在后面的模板字符串就是它的参数。**

**但是，如果模板字符里面有变量，就不是简单的调用了，而是会将模板字符串先处理成多个参数，再调用函数。**

**let a = 5;**

**let b = 10;**

**tag`Hello ${ a + b } world ${ a \* b }`;**

**// 等同于**

**tag(['Hello ', ' world ', ''], 15, 50);**

**上面代码中，模板字符串前面有一个标识名tag，它是一个函数。整个表达式的返回值，就是tag函数处理模板字符串后的返回值。**

**函数tag依次会接收到多个参数。**

**function tag(stringArr, value1, value2){**

**// ...**

**}**

**// 等同于**

**function tag(stringArr, ...values){**

**// ...**

**}**

**tag函数的第一个参数是一个数组，该数组的成员是模板字符串中那些没有变量替换的部分，也就是说，变量替换只发生在数组的第一个成员与第二个成员之间、第二个成员与第三个成员之间，以此类推。**

**tag函数的其他参数，都是模板字符串各个变量被替换后的值。由于本例中，模板字符串含有两个变量，因此tag会接受到value1和value2两个参数。**

**tag函数所有参数的实际值如下。**

* **第一个参数：['Hello ', ' world ', '']**
* **第二个参数: 15**
* **第三个参数：50**

**也就是说，tag函数实际上以下面的形式调用。**

**tag(['Hello ', ' world ', ''], 15, 50)**

**我们可以按照需要编写tag函数的代码。下面是tag函数的一种写法，以及运行结果。**

**let a = 5;**

**let b = 10;**

**function tag(s, v1, v2) {**

**console.log(s[0]);**

**console.log(s[1]);**

**console.log(s[2]);**

**console.log(v1);**

**console.log(v2);**

**return "OK";**

**}**

**tag`Hello ${ a + b } world ${ a \* b}`;**

**// "Hello "**

**// " world "**

**// ""**

**// 15**

**// 50**

**// "OK"**

**下面是一个更复杂的例子。**

**let total = 30;**

**let msg = passthru`The total is ${total} (${total\*1.05} with tax)`;**

**function passthru(literals) {**

**let result = '';**

**let i = 0;**

**while (i < literals.length) {**

**result += literals[i++];**

**if (i < arguments.length) {**

**result += arguments[i];**

**}**

**}**

**return result;**

**}**

**msg // "The total is 30 (31.5 with tax)"**

**上面这个例子展示了，如何将各个参数按照原来的位置拼合回去。**

**passthru函数采用 rest 参数的写法如下。**

**function passthru(literals, ...values) {**

**let output = "";**

**let index;**

**for (index = 0; index < values.length; index++) {**

**output += literals[index] + values[index];**

**}**

**output += literals[index]**

**return output;**

**}**

**“标签模板”的一个重要应用，就是过滤 HTML 字符串，防止用户输入恶意内容。**

**let message =**

**SaferHTML`<p>${sender} has sent you a message.</p>`;**

**function SaferHTML(templateData) {**

**let s = templateData[0];**

**for (let i = 1; i < arguments.length; i++) {**

**let arg = String(arguments[i]);**

**// Escape special characters in the substitution.**

**s += arg.replace(/&/g, "&amp;")**

**.replace(/</g, "&lt;")**

**.replace(/>/g, "&gt;");**

**// Don't escape special characters in the template.**

**s += templateData[i];**

**}**

**return s;**

**}**

**上面代码中，sender变量往往是用户提供的，经过SaferHTML函数处理，里面的特殊字符都会被转义。**

**let sender = '<script>alert("abc")</script>'; // 恶意代码**

**let message = SaferHTML`<p>${sender} has sent you a message.</p>`;**

**message**

**// <p>&lt;script&gt;alert("abc")&lt;/script&gt; has sent you a message.</p>**

**标签模板的另一个应用，就是多语言转换（国际化处理）。**

**i18n`Welcome to ${siteName}, you are visitor number ${visitorNumber}!`**

**// "欢迎访问xxx，您是第xxxx位访问者！"**

**模板字符串本身并不能取代 Mustache 之类的模板库，因为没有条件判断和循环处理功能，但是通过标签函数，你可以自己添加这些功能。**

**// 下面的hashTemplate函数**

**// 是一个自定义的模板处理函数**

**let libraryHtml = hashTemplate`**

**<ul>**

**#for book in ${myBooks}**

**<li><i>#{book.title}</i> by #{book.author}</li>**

**#end**

**</ul>**

**`;**

**除此之外，你甚至可以使用标签模板，在 JavaScript 语言之中嵌入其他语言。**

**jsx`**

**<div>**

**<input**

**ref='input'**

**onChange='${this.handleChange}'**

**defaultValue='${this.state.value}' />**

**${this.state.value}**

**</div>**

**`**

**上面的代码通过jsx函数，将一个 DOM 字符串转为 React 对象。你可以在 GitHub 找到jsx函数的具体实现。**

**下面则是一个假想的例子，通过java函数，在 JavaScript 代码之中运行 Java 代码。**

**java`**

**class HelloWorldApp {**

**public static void main(String[] args) {**

**System.out.println("Hello World!"); // Display the string.**

**}**

**}**

**`**

**HelloWorldApp.main();**

**模板处理函数的第一个参数（模板字符串数组），还有一个raw属性。**

**console.log`123`**

**// ["123", raw: Array[1]]**

**上面代码中，console.log接受的参数，实际上是一个数组。该数组有一个raw属性，保存的是转义后的原字符串。**

**请看下面的例子。**

**tag`First line\nSecond line`**

**function tag(strings) {**

**console.log(strings.raw[0]);**

**// strings.raw[0] 为 "First line\\nSecond line"**

**// 打印输出 "First line\nSecond line"**

**}**

**上面代码中，tag函数的第一个参数strings，有一个raw属性，也指向一个数组。该数组的成员与strings数组完全一致。比如，strings数组是["First line\nSecond line"]，那么strings.raw数组就是["First line\\nSecond line"]。两者唯一的区别，就是字符串里面的斜杠都被转义了。比如，strings.raw 数组会将\n视为\\和n两个字符，而不是换行符。这是为了方便取得转义之前的原始模板而设计的。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**模板字符串的限制**

**前面提到标签模板里面，可以内嵌其他语言。但是，模板字符串默认会将字符串转义，导致无法嵌入其他语言。**

**举例来说，标签模板里面可以嵌入 LaTEX 语言。**

**function latex(strings) {**

**// ...**

**}**

**let document = latex`**

**\newcommand{\fun}{\textbf{Fun!}} // 正常工作**

**\newcommand{\unicode}{\textbf{Unicode!}} // 报错**

**\newcommand{\xerxes}{\textbf{King!}} // 报错**

**Breve over the h goes \u{h}ere // 报错**

**`**

**上面代码中，变量document内嵌的模板字符串，对于 LaTEX 语言来说完全是合法的，但是 JavaScript 引擎会报错。原因就在于字符串的转义。**

**模板字符串会将\u00FF和\u{42}当作 Unicode 字符进行转义，所以\unicode解析时报错；而\x56会被当作十六进制字符串转义，所以\xerxes会报错。也就是说，\u和\x在 LaTEX 里面有特殊含义，但是 JavaScript 将它们转义了。**

**为了解决这个问题，ES2018 放松了对标签模板里面的字符串转义的限制。如果遇到不合法的字符串转义，就返回undefined，而不是报错，并且从raw属性上面可以得到原始字符串。**

**function tag(strs) {**

**strs[0] === undefined**

**strs.raw[0] === "\\unicode and \\u{55}";**

**}**

**tag`\unicode and \u{55}`**

**上面代码中，模板字符串原本是应该报错的，但是由于放松了对字符串转义的限制，所以不报错了，JavaScript 引擎将第一个字符设置为undefined，但是raw属性依然可以得到原始字符串，因此tag函数还是可以对原字符串进行处理。**

**注意，这种对字符串转义的放松，只在标签模板解析字符串时生效，不是标签模板的场合，依然会报错。**

**let bad = `bad escape sequence: \unicode`; // 报错**

**字符串新增方法**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实例方法：includes(), startsWith(), endsWith()**

**传统上，JavaScript 只有indexOf方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6 又提供了三种新方法。**

* **includes()：返回布尔值，表示是否找到了参数字符串。**
* **startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。**
* **endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部。**

**let s = 'Hello world!';**

**s.startsWith('Hello') // true**

**s.endsWith('!') // true**

**s.includes('o') // true**

**这三个方法都支持第二个参数，表示开始搜索的位置。**

**let s = 'Hello world!';**

**s.startsWith('world', 6) // true**

**s.endsWith('Hello', 5) // true**

**s.includes('Hello', 6) // false**

**上面代码表示，使用第二个参数n时，endsWith的行为与其他两个方法有所不同。它针对前n个字符，而其他两个方法针对从第n个位置直到字符串结束。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实例方法：repeat()**

**repeat方法返回一个新字符串，表示将原字符串重复n次。**

**'x'.repeat(3) // "xxx"**

**'hello'.repeat(2) // "hellohello"**

**'na'.repeat(0) // ""**

**参数如果是小数，会被取整。**

**'na'.repeat(2.9) // "nana"**

**如果repeat的参数是负数或者Infinity，会报错。**

**'na'.repeat(Infinity)**

**// RangeError**

**'na'.repeat(-1)**

**// RangeError**

**但是，如果参数是 0 到-1 之间的小数，则等同于 0，这是因为会先进行取整运算。0 到-1 之间的小数，取整以后等于-0，repeat视同为 0。**

**'na'.repeat(-0.9) // ""**

**参数NaN等同于 0。**

**'na'.repeat(NaN) // ""**

**如果repeat的参数是字符串，则会先转换成数字。**

**'na'.repeat('na') // ""**

**'na'.repeat('3') // "nanana"**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**实例方法：padStart()，padEnd()**

**ES2017 引入了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。**

**'x'.padStart(5, 'ab') // 'ababx'**

**'x'.padStart(4, 'ab') // 'abax'**

**'x'.padEnd(5, 'ab') // 'xabab'**

**'x'.padEnd(4, 'ab') // 'xaba'**

**上面代码中，padStart()和padEnd()一共接受两个参数，第一个参数是字符串补全生效的最大长度，第二个参数是用来补全的字符串。**

**如果原字符串的长度，等于或大于最大长度，则字符串补全不生效，返回原字符串。**

**'xxx'.padStart(2, 'ab') // 'xxx'**

**'xxx'.padEnd(2, 'ab') // 'xxx'**

**如果用来补全的字符串与原字符串，两者的长度之和超过了最大长度，则会截去超出位数的补全字符串。**

**'abc'.padStart(10, '0123456789')**

**// '0123456abc'**

**如果省略第二个参数，默认使用空格补全长度。**

**'x'.padStart(4) // ' x'**

**'x'.padEnd(4) // 'x '**

**padStart()的常见用途是为数值补全指定位数。下面代码生成 10 位的数值字符串。**

**'1'.padStart(10, '0') // "0000000001"**

**'12'.padStart(10, '0') // "0000000012"**

**'123456'.padStart(10, '0') // "0000123456"**

**另一个用途是提示字符串格式。**

**'12'.padStart(10, 'YYYY-MM-DD') // "YYYY-MM-12"**

**'09-12'.padStart(10, 'YYYY-MM-DD') // "YYYY-09-12"**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**实例方法：trimStart()，trimEnd()**

**ES2019 对字符串实例新增了trimStart()和trimEnd()这两个方法。它们的行为与trim()一致，trimStart()消除字符串头部的空格，trimEnd()消除尾部的空格。它们返回的都是新字符串，不会修改原始字符串。**

**const s = ' abc ';**

**s.trim() // "abc"**

**s.trimStart() // "abc "**

**s.trimEnd() // " abc"**

**上面代码中，trimStart()只消除头部的空格，保留尾部的空格。trimEnd()也是类似行为。**

**除了空格键，这两个方法对字符串头部（或尾部）的 tab 键、换行符等不可见的空白符号也有效。**

**浏览器还部署了额外的两个方法，trimLeft()是trimStart()的别名，trimRight()是trimEnd()的别名。**

**讲解：**

**ES5：字符串拼接**

**let name = "吴东坡"**

**let age = 18**

**let str = "我的名字叫"+name+"年龄"+age**

**console.log(str)**

**ES6：字符串模板的写法：**

**let name = "吴东坡"**

**let age = 18**

**let str = `我的名字叫${name}年龄是${age}`**

**console.log(str)**

**大括号内部可以放入任意的 JavaScript 表达式，可以进行运算，以及引用对象属性。**

**let x = 1;**

**let y = 2;**

**let ret=`${x} + ${y} = ${x + y}`**

**console.log(ret)**

**let ret1=`${x} + ${y \* 2} = ${x + y \* 2}`**

**console.log(ret1)**

**let obj = {x: 1, y: 2};**

**let ret2=`${obj.x + obj.y}`**

**console.log(ret2)**

**模板字符串之中还能调用函数。**

**function fn() {**

**return "Hello World";**

**}**

**let ret=`foo ${fn()} bar`**

**console.log(ret)**

**ES6：字符串模板的实例写法：**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title></title>**

**<script type="text/javascript">**

**window.onload=function(){**

**let data=[**

**{title:"增速转正！习近平领航中国经济行稳致远",visit:1203,url:"http://www.baidu.com"},**

**{title:"上合秘书长：中国如何创下世界减贫记录？", visit:543,url:"http://www.hao123.com"},**

**{title:"美国专家:美国经济或将二次探底 都是天气惹的祸", visit:344,url:"http://www.sohu.com"},**

**{title:"双赢就是中国赢两次？美司法部长还真用了这句话", visit:4234,url:"http://www.sina.com"}**

**]**

**let contentDiv=document.querySelector(".content")**

**let ul=document.createElement("ul")**

**contentDiv.appendChild(ul)**

**for(let i=0;i<data.length;i++){**

**let li=document.createElement("li")**

**li.innerHTML=`${data[i].title}<span>阅读：${data[i].visit }</span><a href="${data[i].url}">详情</a>`**

**ul.appendChild(li)**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<div class="content">**

**</div>**

**</body>**

**</html>**

**标签模板**

**alert`我跳出来了`**

**console.log`我飞出来了`**

**const func=(m)=>`${m}`;**

**let ret =func(5)**

**console.log(ret)**

**ES6新增的查找字符串功能：**

**首先讲ES5怎么查找字符串的：**

**let str="this is a es5"**

**if(str.indexOf("this")!=-1){**

**console.log("找到了")**

**}else{**

**console.log("没找到")**

**}**

**ES6查找字符串的写法：新增的includes() 方法：返回的true或者false**

**let str="this is a es5"**

**ret = str.includes("this")**

**console.log(ret)**

**判断浏览器：**

**if(navigator.userAgent.includes("Chrome")){**

**alert("谷歌浏览器")**

**}else if(navigator.userAgent.includes("Firefox")){**

**alert("火狐浏览器")**

**}else{**

**alert("其他浏览器")**

**}**

**判断字符串以什么开头的方法：startsWith()**

**let str1="http://www.sohu.com"**

**let str2="https://www.sohu.com"**

**if(str1.startsWith("http")){**

**location.href="http://www.sohu.com"**

**}else{**

**location.href="** **https://www.bootcss.com "**

**}**

**模拟用户输入：**

**let arr=["http://www.sohu.com","https://www.bootcss.com"];**

**for (let i = 0; i < arr.length; i++) {**

**let item=arr[i];**

**if(item.startsWith("http")){**

**location.href="http://www.sohu.com";**

**break;**

**}else{**

**location.href="https://www.bootcss.com";**

**break;**

**}**

**}**

**判断字符串以什么结尾的方法：endsWith()**

**let str1="http://www.sohu.com"**

**let str2="http://www.bwlc.net"**

**if(str1.endsWith("com")){**

**location.href="http://www.sohu.com"**

**}else{**

**location.href="http://www.bwlc.net"**

**}**

**字符串的重复的次数方法：repeat(n)**

**let str="聚焦"**

**ret = str.repeat(3)**

**console.log(ret)**

**字符串填充方法：padStart(length，newStr)前面填充和padEnd(length，newStr)后面填充**

**案例讲解：**

**let str="聚焦"**

**let ret = str.padStart(4,"廊坊")**

**alert(ret)**

**let ret1 = ret.padEnd(6,"教育")**

**alert(ret1)**

**模板字符可以嵌套：**

**const data = [**

**{ name: '张三', sex: '男',age:20 },**

**{ name: '小花', sex: '女',age:18 },**

**{ name: '赵彤', sex: '男',age:28 },**

**];**

**const tmpl = arr => `**

**<table>**

**<tr>**

**<th>姓名</th>**

**<th>性别</th>**

**<th>年龄</th>**

**</tr>**

**${arr.map(obj => `**

**<tr>**

**<td>${obj.name}</td>**

**<td>${obj.sex}</td>**

**<td>${obj.age}</td>**

**</tr>**

**`).join('')}**

**</table>**

**`;**

**$("#list").html(tmpl(data));**

**trim()方法、trimLeft()方法、trimRight()方法 去空白**

**let str=" str "**

**console.log(str.trim())**

**console.log(str.trimLeft())**

**console.log(str.trimRight())**