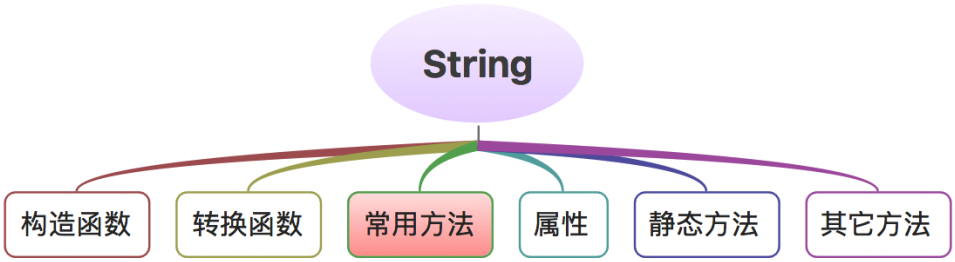
string

> 本文将重点介绍javaScript语言中字符串，操作字符串的常见方法以及具体的代码实现等。



字符串简单介绍

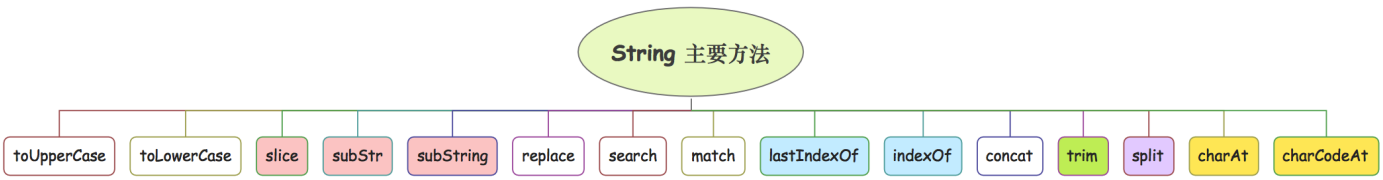
> 关于字符串(类型)，其实在我的另一篇文章中 JavaScript语言基础 已经有过下面这段简单介绍。

定义 由〇个或多个16位Unicode字符组成的字符序列。
表示 字符串可以由双引号或单引号表示。
操作 可以通过length属性来获取字符串的长度，且多个字符串之间可以通过 + 来进行拼接。
注意 JavaScript中的字符串是 不可变的 ，即改变某个变量保存的字符串需先销毁然后再重新填充。

```
var str1 = "Hi ~";
var str2 = "Wendingding!";
var str3 = str1 + " " +str2;     /*字符串的拼接 */
console.log(str3);               /*输出结果: Hi ~ Wendingding!*/
console.log(str3.length);        /*输出结果: 17 */
```

其实，掌握了上面这些知识点就已经对JavaScript语言中的字符串有了一个七七八八的概念了，需要注意的是，本文的核心是在此基础上讲清楚 字符串操作(大小写转换、搜索、拼接、查找等)相关的知识点 ，因为这涉及到一大堆的方法(函数)，因此要解释清楚它们这将会是庞大艰难的任务，下面我将先列出JavaScript语言中String相关的所有操作然后再分门别类的进行介绍。

字符串核心方法



字符串的拼接

> JavaScript提供了 `concat()` 方法来进行字符串的拼接。

语法 `string.concat(value,...)`
说明 `concat()` 方法可以接受N(一个或多个)个待连接的字符串，它的作用是将每个参数都转换为字符串(若不是字符串则内部会自动转换)，并将它们按顺序追加到当前字符串的末尾并返回最后的结果。
备注 `concat()` 方法更简单的写法是直接使用符号 `+` 来实现。

```
/*01 第一种拼接的方式 使用+*/
/*02 第二种拼接的方式 使用concat方法*/
var str1 = "Hello ";
var str2 = "wendingding";
console.log(str1 + str2 + "!");       //Hello wendingding !
console.log(str1.concat(str2 + "?")); //Hello wendingding?
/*备注：需要注意concat方法并不会修改拼接的字符串本身而是把结果作为返回值*/
console.log(str1);                    //Hello
console.log(str2);                    //Hello

console.log(str1.concat("world", "!"," Nice to", "meet u"));
//Hello world ! Nice tomeet u  演示接收多个参数的情况
```

字符串的大小写转换

> JavaScript提供了专门处理大小写字符转换的方法,它们分别是 `toLowerCase()` 、 `toUpperCase()` 、 `toLocaleUpperCase()` 以及 `toLocaleLowerCase()` 四个方法，其中后面的两个方法在进行大小写转换的时候需要考虑当前本地化语言环境的大小写映射，大多数情况下它们和前两个方法保持一致。

toLowerCase( ) 和 toLocaleLowerCase( )	将字符串转换为小写
toUpperCase( ) 和 toLocaleUpperCase( )	将字符串转换为大写

```
var str = "Hi! Nice to meet u";
console.log(str.toLowerCase());       //hi! nice to meet u
console.log(str.toLocaleLowerCase()); //hi! nice to meet u
console.log(str.toUpperCase());       //HI! NICE TO MEET U
console.log(str.toLocaleUpperCase()); //HI! NICE TO MEET U
```

字符串的子串搜索

在开发中有时候我们需要检查某个字符串中是否存在指定的字符或者是子字符串，这时候就需要用到JavaScript提供的子串搜索方法，它们是 `indexOf()` 和 `lastIndexOf()` 方法。这两个方法的核心区分在于 `indexOf()` 方法从前往后搜索而 `lastIndexOf()` 方法从后往前搜索。

语法 `string.indexOf(substring,[start])` `string.lastIndexOf(substring,[start])`
参数 `substring` 表示要搜索的字符(子字符串)。

参数　start 是可选的整数值用来指定开始搜索的位置，合法值为 0 ~ string.length-1。

说明　indexOf()方法 默认从第一个字符开始搜索， lastIndexOf()方法 则默认从最后一个字符开始。

结果　若没有找到指定字符(子串)就返回-1，否则就返回找到的子串中第一个字符的位置。

备注　在字符串中第一个字符的位置为 0，最后一个字符的位置为string.length -1 。

```
var str = "Hello ! Hello wendingding.";
var sub1 = "Hello";
var sub2 = "ding";

console.log(str.indexOf(sub1));     //0
console.log(str.lastIndexOf(sub1)); //8
/*indexOf 前-->后   lastIndexOf 后-->前*/
console.log(str.indexOf(sub2));     //17
console.log(str.lastIndexOf(sub2)); //21

/*测试第二个参数[start]的使用情况*/
console.log(str.indexOf(sub1,2));   //8 查找的范围缩小为"llo ! Hello wendingding."
console.log("123456".lastIndexOf("3",1));  //-1 查找范围为"12"
console.log("123456".lastIndexOf("3",4)); //2 查找范围为"12345"
```

字符串的子串切割

JavaScript字符串提供了 split() 方法来将一个字符串切割为数组。

语法　string.split(delimiter,limit)

参数　delimiter 是用来切割(分割)的字符串或正则表达式， limit 用于指定数组长度(默认不限制)。

```
var str1 = "苹果,香蕉,橙子,榴莲,水蜜桃";
var str2 = "javaScript ios java go python perl";


/*01-以逗号来分隔切割字符串为数组*/
console.log(str1.split(","));      //["苹果", "香蕉", "橙子", "榴莲", "水蜜桃"]

/*02-以特殊字符来切割*/
console.log("a|b|c|d".split("|")); //["a","b","c","d"]
console.log("1:2:3:4".split(":")); //["1","2","3","4"]

/*03-对比参数是字符串和正则表达式的情况*/
console.log(str2.split(" "));     //["javaScript","ios","java","go","python","perl"]
console.log(str2.split(/\s+/));

/*04-以空字符串分隔切割字符串为数组，每个字符都切割*/
console.log(str1.split(""));
//["苹", "果", ",", "香", "蕉", ",", "橙", "子", ",", "榴", "莲", ",", "水", "蜜", "桃"]
/*说明：这种方式常用来把完整的单词切割成字符*/
console.log("hello".split(""));   //["h","e","l","l","o"]

/*05-limit参数的使用*/
console.log("hello".split("",3)); //["h","e","l"]
console.log(str1.split(",",3));   //["苹果", "香蕉", "橙子"]

/*06-匹配的分隔符在开头或者结尾*/
console.log("hello".split("he")); //["","llo"];
```

```
console.log("hello".split("lo"));//["hel",""];
console.log("helohe".split("he"));//["","lo",""];
```

说明 split() 方法常用来处理高度结构化的字符串，在开发中经常使用。在使用split()来切割字符串的时候，得到的数组中每个元素都是字符串类型(得到的一定是字符串数组)，且指定的分隔符不可能出现在数组元素中。如果分隔符匹配给定字符串的开头或者是结尾内容，那么返回数组的第一个元素和最后一个元素将是空字符串(参考上文代码的06)。

获取字符串的指定字符(编码)

JavaScript语言提供了 charAt() 和 charCodeAt() 方法来获取字符串中的指定字符或字符编码。

语法 string.charAt(n) string.charCodeAt(n)
作用 charAt() 用于获取字符串中的第 n个字 ， charCodeAt() 方法获取的是第 n个字符的编码 。
说明 字符串的第一个字符的索引编号为0，长度为 string.length - 。
备注 JavaScript中并没有字符数据类型，所以返回的 某个字符 其实是一个长度为1的字符串。

```
var str = "hello world";
/*charAt方法返回的是指定位置(索引)的字符*/
console.log(str[0]);                    //"h" 也可以直接使用索引下标访问
console.log(str.charAt(0));             //"h"
console.log(str.charAt(1));             //"e"
console.log(str.charAt(str.length -1)); //"d"

/*注意：如果charAt的参数不在 0 ~ length-1范围内，则返回空字符串*/
console.log(str.charAt(-1));            //""
console.log(str.charAt(str.length));    //""

/*charCodeAt方法返回的是指定位置(索引)字符对应的字符编码*/
console.log(str.charCodeAt(0));         //104
/*注意：如果charCodeAt的参数不在 0 ~ length-1范围内，则返回NaN*/
console.log(str.charCodeAt(-1));        //NaN
console.log(str.charCodeAt(str.length));//NaN
```

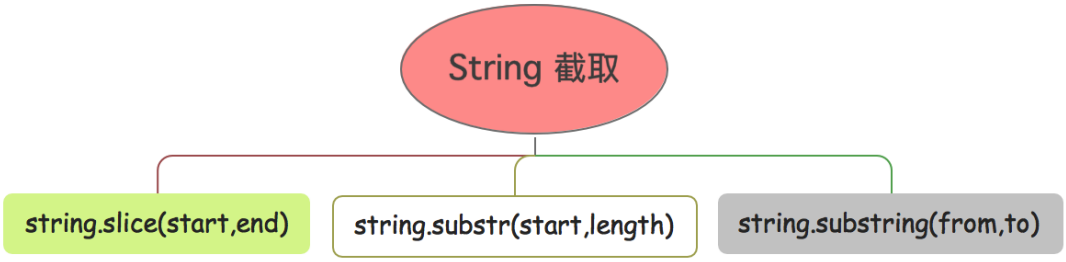
编码 charCodeAt() 方法返回的字符编码指的是 Unicode编码 ， 为16位整数，取值在0~65535之间。

> 有时候我们需要根据Unicode编码来创建字符串，因此JavaScript提供了和 charCodeAt() 相反的方法 fromCharCode() ，该方法定义在String构造函数身上是一个静态方法。

```
/*语法：String.fromCharCode(c1,c2,c3,...)*/
var result = String.fromCharCode(104,105,32,44,32,88,105,97,32,33);
console.log(result);                    //hi , Xia !
```

字符串的截取

字符串截取是编程中经常用到的操作，JavaScript提供了 slice() 、 substr() 和 substring() 三个方法来实现字符串的截取操作，这三个方法在使用的时候 很像但是又不太一样。




```
/*截取字符串的方法比较*/
var test = "hi 文顶顶";

/*01-如果只传递一个参数，表现一致*/
console.log(test.slice(3));     //文顶顶 默认截取 3 ~ test.length-1的字符串
console.log(test.substr(3));    //文顶顶
console.log(test.substring(3)); //文顶顶

/*02-传递第二个参数
* slice(start,end)     (开始截取的位置，结束截取的位置[不包含])
* substr(start,length) (开始截取的位置，截取的长度)
* substring(from,to)   (开始位置，结束位置)
* */

console.log(test.slice(3,4));     //文
console.log(test.substr(3,4));    //文顶顶
console.log(test.substring(3,4)); //文
```

注意 我们在给这些截取字符串的方法传递参数的时候，是可以传递负值的，如果传递的值是负数那么这三个方法将表现出巨大的差异，在使用的时候要注意区分。

```
var test = "hi 文顶顶";
/*01-参数为负数的情况*/
console.log(test.slice(-2));              //"顶顶" 倒数第三个(索引值为 -2 + test.length-1)
console.log(test.slice(-2 + test.length));//"顶顶"

console.log(test.substr(-2));             //"顶顶"

/*substring方法的参数不能为负数，如果是负数那么默认转换为0*/
console.log(test.substring(-2));          //"hi 文顶顶"

/*02-更复杂(两个参数)的情况*/
/*slice方法中第二个参数为负数，那么将 + string.length*/
console.log(test.slice(3,-1));           //"文顶"  等价于slice(3,5)
/*substr方法中第二个参数为负数，那么转换为0*/
console.log(test.substr(3,-1));          //""     等价于substr(3,0)
/*substring方法中第二个参数为负数，会先转换为0，然后比较两个参数的大小再调整*/
console.log(test.substring(3,-1));       //"hi "  等价于test.substring(0,3)
console.log(test.substring(3,1));        //"i "   等价于test.substring(1,3)
```

备注 `substring()` 方法会将两个参数中较小的数值作为开始位置，将较大的数值作为结束位置，因此像上面示例代码中的 `test.substring(3,1)` 这行代码，相当于调用了 `test.substring(1,3)` 。

> `substring()` 方法的参数不接受负值，如果传递了负值那么总是会被转换为0。 `substr()` 方法已经不再在ECMAScript的内容，已经被弃用。

清除字符串前后空格

在早期的时候，JavaScript中并没有专门用来清除字符串前后N各空格的方法可以使用。但是，在开发中又确实有这样的需求，譬如我们在获取用户的表单输入后往往需要先做清空格处理和校验之后才提交给服务器端，以前在处理这种开发场景的时候可能需要自己来封装一个专门的方法或者直接使用jQuery框架中的 `$.trim()` 方法。庆幸的是在ES5中，JavaScript为我们提供了这样一个trim方法，它的使用方式非常简单直接调用即可，作用是 清除字符串前后的1个或多个空格。

```
var testStr = " abc def  ";
console.log(testStr);          //" abc def  "
console.log(testStr.trim());  //"abc def"
```

字符串其它说明

在JavaScript中字符串除了上述介绍的这些方法外，其实还有一些内容。譬如 `String()函数` 用来把数字、布尔值等数据转换为字符串， `localeCompare()` 方法用于比较两个字符串的顺序， `toString()` 和 `valueOf()` 方法对于字符串而言很少使用，`search()` 、 `match()` 以及 `replace()` 方法用来进行字符串的模式匹配，因为它们或多或少的涉及到正则表达式的知识点所以此文不再额外扩展。

当然，字符串知识模块还有一些内容没有讲解，譬如构造函数以及String对象类型甚至ES6新增的字符串模板等，我并没有把它们纳入到本文的范围中，大家可以自行查阅相关资料和扩展。

`string` 是JavaScript中的一种原始(简单|基本)数据类型，字符串中的length属性指定该字符串中字符的个数(长度)，上文中介绍了字符串操作的诸多常用方法。需要重申的是：JavaScript中的字符串是不可变的，文中介绍的所有方法都不允许改变当前字符串的的内容，它们做的只是返回一个全新的字符串，并没有修改原始字符串。