小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->JSCORE->day02->点开每个问题方块，就可看到问题的详细解答

当老师在回答其它同学的问题时，你可以打开问题清单浏览曾经回答过的问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day02

2部分: 1. 课上重点知识和案例的回放！2. 重点扩展笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频: 小程序->在线->JSCORE->day02

四个"扩展"开头的视频

所有扩展视频绝\*\*\*不要求当天必须看完\*\*\*！而且和第二天课程也没有关系！

只要在\*\*\*两个月后\*\*\*鄙视面试前\*\*\*看完就行\*\*\*！

3. 单词列表: 小程序->单词->JSCORE->day02

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

—— 去ftp WEB2003或WEB2004班 下载师兄师姐的笔记跟着看！——极其详细

2. 拼写错误！自己找！一旦让我发现是拼写错误，自己还懒得找，我可以帮你找。但是，如果我找到，你需要手抄20遍，给我拍照！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻微信私信问我！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻微信私信问我！

3. 个人项目中哪个功能不会做！立刻微信私信问我！

4. 刷笔试题时，遇到不会的笔试题！立刻微信私信问我！

笔记和源代码下载: (内网和外网甚至今后工作了，都可随时下载！没有网络限制的)

<http://code.tarena.com.cn/WEBCode/>

用户名: tarenacode

密码: code\_2015

下载源代码: 右键->链接另存为！

正课:

1. String类型提供的正则相关的函数

2. RegExp对象

一. String类型提供的正则相关函数: 3件事:

对应小程序视频: 小程序->在线->JSCORE->day02 0. 查找敏感词

1. 查找敏感词: 4种情况:

(1). 查找一个固定的敏感词出现的位置

a. var 敏感词的下标位置=字符串.indexOf("要找的敏感词")

位置的 敏感词

b. 意为: 在字符串中，查找一个指定的敏感词出现的下标位置。

c. 返回值: 2种:

1). 如果找到敏感词，会返回敏感词中第一个字在整个字符串中的下标位置。(js中所有下标默认从0开始！)

2). 如果没找到，会返回-1 （信任）

d. 示例: 检查用户输入的消息内容中，是否包含敏感词我草:

1\_indexOf.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <!--第三阶段所有案例，<script>都要放在<body>结尾！-->    <script>      //回顾indexOf——不用做      var str="那天，我去了她家。我说：我草！你家真大！";      //尝试查找字符串str中敏感词"我草"出现的位置:      var i=str.indexOf("我草");      console.log(i);      //尝试查找“海王”      var i=str.indexOf("海王");      console.log(i);      //从这里开始写:      //请用户输入一条消息      var input=prompt("请输入消息内容");      //查找消息中是否包含敏感词"我草"      var i=input.indexOf("我草");      //如果找到，返回下标      //如果没找到，返回-1      //如果找到敏感词      if(i!=-1){        //就提示，包含敏感词，禁止发送        //网页    写        //(向)当前网页中写一条HTML内容        document.write(`<h3 style="color:red">包含敏感词,禁止发送</h3>`);//复习第一阶段"模板字符串"      }else{//否则没找到敏感词        //就提示，"然哥说: 消息内容"        document.write(`<h3 style="color:green">然哥说：${input}</h3>`)//复习第一阶段"模板字符串"      }    </script>  </body>  </html>  运行结果：    **包含敏感词,禁止发送** 然哥说：我去！你家真大！ |

e. 问题：indexOf只能查找一种固定的敏感词，一旦用户换字，就防不住了！

(2). 用正则表达式查找多种敏感词出现的位置:

js的地盘←|正则地盘 |→ js的地盘

a. var 下标位置=字符串.search(/正则表达式/i);

查找

b. 意为: 在字符串中查找第一个和正则表达式匹配的敏感词的下标位置

c. 返回值: 2种:

1). 如果找到，返回敏感词第一个字的下标位置

2). 如果没找到，返回-1

d. 问题: 正则表达式默认区分大小写的！

e. 解决: 在正则第二个/后加后缀i, 意为ignore(忽略)，表示忽略大小写！

f. 示例: 使用正则表达式查找多种我草

2\_search.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <!--第三阶段所有案例，<script>都要放在<body>结尾！-->    <script>      //从这里开始写:      //请用户输入一条消息      var input=prompt("请输入消息内容");      //查找消息中是否包含敏感词"我草"      // var i=input.indexOf("我草");      var i=input.search(/([我卧]|wo)\s\*([艹草槽]|cao)/i);//忽略大小写      //如果找到，返回下标      //如果没找到，返回-1      //如果找到敏感词      if(i!=-1){        //就提示，包含敏感词，禁止发送        //网页    写        //(向)当前网页中写一条HTML内容        document.write(`<h3 style="color:red">包含敏感词,禁止发送</h3>`);//复习第一阶段"模板字符串"      }else{//否则没找到敏感词        //就提示，"然哥说: 消息内容"        document.write(`<h3 style="color:green">然哥说：${input}</h3>`)//复习第一阶段"模板字符串"      }    </script>  </body>  </html>  运行结果：   包含敏感词,禁止发送然哥说：我去！你家真大！ |

g.问题: 无法获得敏感词的内容

(3). 获得敏感词的内容: 2种:

a. 只获得第一个敏感词的内容和位置:

1). var 数组=字符串.match(/正则表达式/i); //不要加g

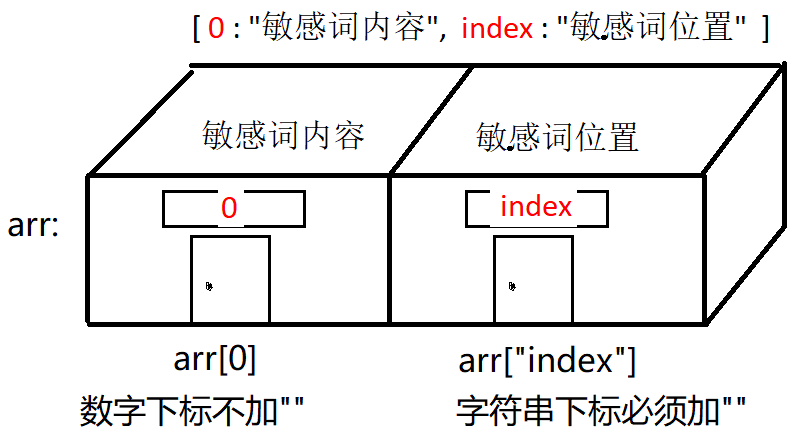
匹配

2). 意为: 在字符串中查找第一个符合正则表达式要求的敏感词的内容和位置。

3). 返回值:

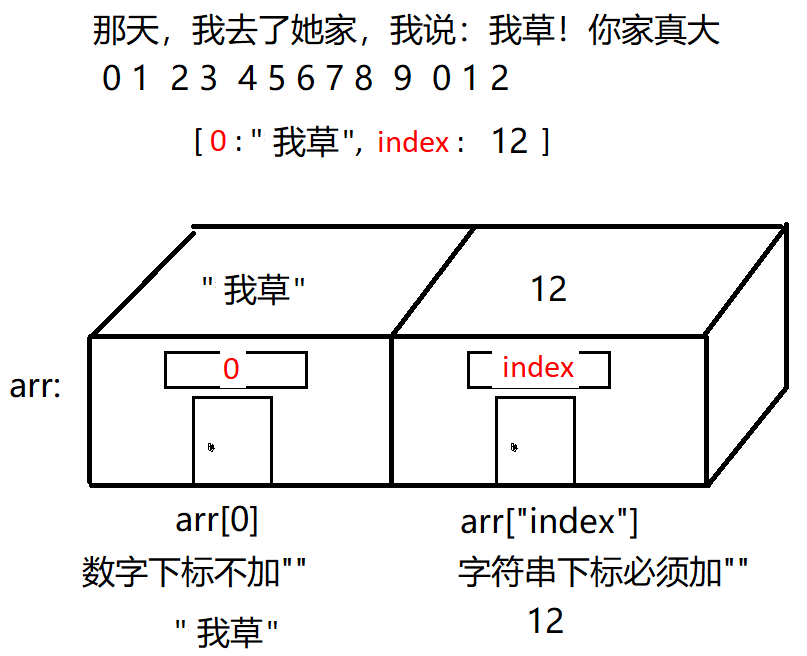
i. 如果找到敏感词，就返回一个数组，其中既包含敏感词的内容，又包含敏感词的位置

数组: [ 0 : "敏感词内容", index : "敏感词位置" ]



ii. 如果找不到，返回null

4). 示例: 既查找敏感词内容，又查找敏感词位置:



3\_match.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <!--第三阶段所有案例，<script>都要放在<body>结尾！-->    <script>      //从这里开始写:      //请用户输入一条消息      var input=prompt("请输入消息内容");      //查找消息中是否包含敏感词"我草"      // var i=input.indexOf("我草");      // var i=input.search(/([我卧]|wo)\s\*([艹草槽]|cao)/i);//忽略大小写      var arr=input.match(/([我卧]|wo)\s\*([艹草槽]|cao)/i);//不要加g！      console.log(arr);      //如果找到，返回数组      //如果没找到，返回null      //如果找到敏感词      if(arr!=null){        //就提示，包含敏感词，禁止发送        //标准:        // document.write(`<h3 style="color:red">在位置${arr["index"]}，发现敏感词${arr["0"]}，禁止发送</h3>`);//复习第一阶段"模板字符串"        //简写:        document.write(`<h3 style="color:red">在位置${arr.index}，发现敏感词${arr[0]}，禁止发送</h3>`);//复习第一阶段"模板字符串"      }else{//否则没找到敏感词        //就提示，"然哥说: 消息内容"        document.write(`<h3 style="color:green">然哥说：${input}</h3>`)//复习第一阶段"模板字符串"      }//11:40上课    </script>  </body>  </html>  运行结果：   在位置12，发现敏感词我草，禁止发送 |

5). 问题: 正则表达式默认只查找第一个符合要求的敏感词，不会查找所有

b. 查找所有敏感词的内容:

1). var 数组=字符串.match(/正则表达式/ig) //global全部

2). 意为: 在字符串中查找所有符合正则表达式要求的敏感词的内容。

3). 返回值: 2种:

i. 如果找到，返回一个数组，其中包含所有符合要求的敏感词内容。

强调: 不包含位置

ii. 如果没找到，返回null

4). 示例: 查找一句话中所有以小字开头人名:

4\_match\_g.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var str="老师: 请用小红 我的 朋友造句。小亮: 小红是我的朋友。小然:朋友！小红是我的！";      //找出这句话中所有以小字开头的人名      var arr=str.match(/小[\u4e00-\u9fa5]/ig);      console.log(arr);    </script>  </body>  </html>  运行结果:   1. *(5) ["小红", "小亮", "小红", "小然", "小红"]*    1. 0: "小红"    2. 1: "小亮"    3. 2: "小红"    4. 3: "小然"    5. 4: "小红"    6. length: 5 |

(4). 既查找每个敏感词的内容，又能够查找每个敏感词的位置:

reg.exec() (未完待续...)

|  |
| --- |
| 补: 查找相关的函数的返回值规律:  1. 如果原函数返回的是下标位置，则如果找不到，都返回-1  2. 如果原函数返回的是一个数组或一个对象，则如果找不到，返回null |

|  |
| --- |
| 补: 数组本质:  1. 索引数组: 下标全为数字的数组  比如: arr=["亮亮","然然","东东"]  自动分配 0 1 2  结果: [  0: "亮亮",  1: "然然",  2: "东东"  ]  2. 关联数组: 下标全为自定义字符串名称的数组  比如: ran=[]  ran["数学"]=89;  ran["语文"]=69;  ran["英语"]=91;  结果: [  数学: 89,  语文: 69,  英语: 91  ]  3. 本质: js中底层其实只有一种数组，都是关联数组。  索引数组: [  0: "亮亮",  1: "然然",  2: "东东"  ]  底层存储时，下标都是字符串类型:  [  "0": "亮亮",  "1": "然然",  "2": "东东"  ]  关联数组: [  数学: 89,  语文: 69,  英语: 91  ]  底层存储时，下标也都是字符串类型:  [  "数学": 89,  "语文": 69,  "英语": 91  ]  4. 访问数组中的元素: 2种:  (1). 标准写法: 数组名["下标名"]  比如: arr["0"] -> "亮亮" ran["数学"] -> 89  (2). 简写:  a. 如果下标是数字内容，可简写为arr[0]  说明，数字下标~~不能简写为.0~~，会和小数中的小数点冲突！  b. 如果下标是自定义字符串内容，可简写为ran.数学 |

2. 替换敏感词: 2种:

对应小程序视频：小程序->在线->JSCORE->day02 3. 替换...

(1). 简单替换: 将所有敏感词都替换为统一的新值

a. 变量=字符串.replace(/正则表达式/ig, "新值");

b. 意为: 将字符串中所有符合正则表达式要求的敏感词头替换为统一的新值。

c. 强调: 因为字符串是不可变类型，所以，所有字符串家的函数都无权修改原字符串。只能返回一个新的字符串。原字符串保持不变。所以，今后所有字符串家的函数，都要用变量接住执行后返回的新字符串。

d. 示例: 替换字符串中所有小字开头的名字

5\_replace.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var str="老师: 请用小红 我的 朋友造句。小亮: 小红是我的朋友。小然:朋友！小红是我的！";      //将所有以小字开头的人名替换为\*\*      str=str.replace(/小[\u4e00-\u9fa5]/g,"\*\*");      console.log(str);    </script>  </body>  </html>  运行结果：  老师: 请用\*\* 我的 朋友造句。\*\*: \*\*是我的朋友。\*\*:朋友！\*\*是我的！ |

(2). 高级替换: 根据每次找到的敏感词不同，动态选择比替换为不同的新值。

a. 变量=字符串.replace(/正则表达式/ig, function(形参){

return 根据本次找到的敏感词不同，动态选择的新值

})

b. 意为: 将字符串中所有符合正则要求的敏感词，根据每个敏感词的不同，动态替换为不同的新值。

c. 原理:

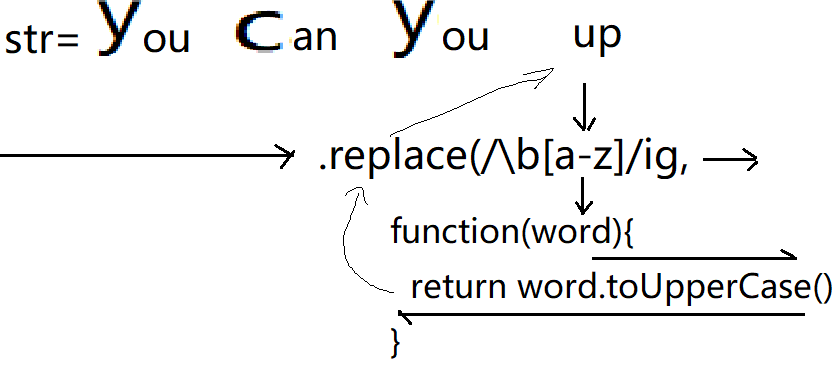
1). replace用正则表达式去查找字符串中每个敏感词

2). 每找到一个敏感词,replace就自动在内调用一次回调函数

3). 每次调用回调函数时，replace都会自动将本次找到的敏感词传给回调函数的形参。（信任）

4). 在回调函数内部，可以根据本次收到的敏感词，动态生成要替换的新值，并返回给replace

5). replace接到回调函数返回的新值，自动替换到原字符串中本次找到的敏感词位置。



d. 示例: 将每个单词首字母转为大写:

6\_replace2.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //高频鄙视题: 将每个单词首字母转为大写:      var str="you can you up";      str=str.replace(        /\b[a-z]/ig,        //回调函数: 我们自己定义的，但是我们自己不调用，而是交给另一个函数，去自动反复调用。        function(word){          console.log(`自动调用了一次回调函数，形参word自动收到${word}，返回了${word.toUpperCase()}`);          return word.toUpperCase();        }      )      console.log(str); //You Can You Up    </script>  </body>  </html>  运行结果:  自动调用了一次回调函数，形参word自动收到y，返回了Y  自动调用了一次回调函数，形参word自动收到c，返回了C  自动调用了一次回调函数，形参word自动收到y，返回了Y  自动调用了一次回调函数，形参word自动收到u，返回了U  You Can You Up |

(3). 衍生操作: 删除敏感词，其实就是替换为""。

a. 字符串=字符串.replace(/正则表达式/ig, "");

b. 示例: 去掉字符串开头和结尾的空字符

7\_trim.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //去掉字符串开头和结尾的空字符      //定义3个函数      //1. 只去掉开头的空字符      function trimLeft(str){        return str.replace(/^\s+/,"");      }      //2. 只去掉结尾的空字符      function trimRight(str){        return str.replace(/\s+$/,"");      }      //3. 同时去掉开头和结尾的空字符      function trim(str){        return str.replace(/^\s+|\s+$/g,"");      }      //测试:      var str="     zhang     dong     ";      console.log(trimLeft(str));      //"zhang     dong     "      console.log(trimRight(str));      //"     zhang     dong"      console.log(trim(str));      //"zhang     dong"      //ES5、6      // var str="    zhang    dong    ";      // console.log(str.trimLeft());      // console.log(str.trimRight());      // console.log(str.trim())    </script>  </body>  </html>  运行结果:  zhang dong  zhang dong  zhang dong |

3. 切割字符串: 2种

对应小程序视频: 小程序->在线->JSCORE->day02 4. 切割str.split() ...

(1). 简单切割: 将一个字符串，按固定的切割符，切割为多段子字符串

a. var 数组=字符串.split("切割符");

b. 意为: 将字符串按照固定的一种切割符，切割为多段子字符串。

c. 返回值:

1). 返回切割后多段子字符串组成的数组

2). 切割后的结果数组中，是不包含切割符的！

(2). 复杂切割: 切割符不是固定的，但是也可以找到规律。

a. var 数组=字符串.split(/正则表达式/) //不用加g

b. 意为: 将字符串按所有符合正则要求的切割符，切割为多段子字符串。

c. 强调: 即使不加g，也能找到所有切割符。

(3). 衍生: 打散字符串: 将字符串打散为字符组成的数组

a. var 数组=字符串.split("")

b. 意为: 将字符串按每个字符之间的缝隙，切割为字符组成的数组。

c. 返回值: 由字符组成的数组

(4). 示例: 切割字符串，翻转一个字符串

8\_split.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var str="you can you up";      //将英文的句子，变成单词的数组，按空格切割      var arr=str.split(" ");      console.log(arr);      var str="you      can        you up";      //将英文的句子，变成单词的数组，按空格切割      var arr=str.split(/\s+/);      console.log(arr);      var arr=[1,2,3,4,5];      arr.reverse();      console.log(arr);      //高频鄙视题: 翻转一个字符串:      var str="helloworld";      //将str打散为字符数组      var arr=str.split("");      console.log(arr);      //将arr翻转      arr.reverse();      console.log(arr);      //将arr无缝拼接为一个字符串      //复习第一阶段数组函数join:用任意自定义连接符，将数组中每个元素值拼接为一个字符串。      str=arr.join("");//无缝拼接      console.log(str); //"dlrowolleh"    </script>  </body>  </html>  运行结果:  (4) ["you", "can", "you", "up"]  (4) ["you", "can", "you", "up"]  (5) [5, 4, 3, 2, 1]  (10) ["h", "e", "l", "l", "o", "w", "o", "r", "l", "d"]  (10) ["d", "l", "r", "o", "w", "o", "l", "l", "e", "h"]  dlrowolleh |

二. RegExp对象:

对应小程序视频: 小程序->在线->JSCORE->day02 5. new RegExp() 动态生成...

1. 什么是RegExp对象: 专门保存一条正则表达式，并提供用正则表达式执行查找和验证的功能 的对象

2. 为什么: 正则表达式的语法，是独立于任何语言之外的独特的语法。js语言默认不认识正则。就需要一个专门的对象，将正则翻译为js认识的东西，才能执行！

3. 何时: 今后，只要在js中使用正则表达式，都要先创建正则表达式对象。

4. 如何创建正则表达式对象: 2种:

(1). 标准:

a. var 变量=new RegExp("正则","ig")

b. 强调: 用new RegExp，则""中不再需要/ /

c. 坑: 因为js的字符串中也有\x开头的特殊符号，表示转义字符。比如: \n 表示换行， \t表示Tab键。和正则中的字符集简写冲突了。比如: \d \s等。

d. 解决: 今后如果使用new RegExp()，而刚好正则中包含\d或\s、\w等，都要变成\\d、\\s、\\w

(2). 简写:

a. var 变量=/正则/ig;

(3). 问题: / /之中只支持写死的固定不变的正则表达式。不支持使用变量或js表达式动态生成正则表达式。

(4). 解决: 今后，只要需要根据变量或数组动态生成正则表达式，则都必须用new RegExp()。不能用/ /

(5). 原因: new RegExp()需要的是一个字符串类型的正则表达式，而我们有很多方法，拼接出任意我们想要的字符串，再传给new RegExp()

(6). 示例: 根据数组内容，动态生成正则表达式：

9\_RegExp.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //将所有的敏感词都放在一个数组中集中管理      var arr=["明月","青天","紫烟"];      //希望生成: 明月|青天|紫烟      var str=arr.join("|");      console.log(str);      //希望数组中有什么敏感词，程序就要防住什么敏感词      //其实: 将数组的内容，动态生成为正则表达式      var input=prompt("请用户输入消息内容");      //用正则查找input中是否包含符合要求的敏感词      //错误:      //         js地盘← |正则|→js地盘      // var i=input.search(/str/);      //正确:      var i=input.search(new RegExp(str));      //                           变量不能加""      //如果找到敏感词      if(i!=-1){        //就提示，包含敏感词，禁止发送        document.write(`<h3 style="color:red">包含敏感词,禁止发送</h3>`);      }else{//否则没找到敏感词        //就提示，"然哥说: 消息内容"        document.write(`<h3 style="color:green">然哥说：${input}</h3>`)      }      </script>  </body>  </html>  运行结果:   发现敏感词！禁止发送!  发现敏感词！禁止发送!   发现敏感词！禁止发送! |

5. 验证字符串格式:

对应小程序视频列表：小程序->在线->JSCORE->day02 6. 验证字符串格式

(1). var 验证结果=正则表达式对象.test(要验证的字符串)

验证

(2). 意为: 用正则表达式，验证字符串的格式是否符合正则的要求

(3). 返回值: 如果字符串的格式符合正则的要求，则返回true; 如果字符串的格式不符合正则的要求，就返回false。

(4). 坑: 默认，正则表达式只要在字符串中找到部分内容匹配，就会返回true！但是，通常验证时，都要求必须从头到尾完整匹配才行！

(5). 解决: 今后，只要验证，都必须同时前加^，后加$，表示从头到尾必须完整匹配。

(6). 示例: 验证手机号格式是否正确:

10\_test.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //请用户输入手机号      var phone=prompt("请输入手机号");      //用正则验证手机号的格式是否正确      var result=/^1[3-9]\d{9}$/.test(phone);      if(result==true){        document.write(`<h3 style="color:green">手机号格式正确</h3>`)      }else{        document.write(`<h3 style="color:red">手机号格式不正确</h3>`)      }    </script>  </body>  </html>  运行结果:   手机号格式正确   手机号格式不正确 |

总结:

1. 只要验证字符串格式或查找、屏蔽敏感词时都要用正则

(1). 最简单的正则: 一个敏感词的原文

(2). 某一位字符上可能有多种备选字时用: [备选字列表]

(3). 如果[]中部分字符是连续的，可用: [x-x]

a. 一位小写字母: [a-z]

b. 一位大写字母: [A-Z]

c. 一位字母(大小写都行): [A-Za-z]

d. 一位字母或数字都行: [0-9A-Za-z]

e. 一位汉字: [\u4e00-\u9fa5]

(4). 预定义字符集:

a. \d 一位数字

b. \w 一位数字、字母或\_

c. \s 空格、tab、换行等空字符

d. . 任意字符

(5). 如果规定一个字符集或子规则反复出现的次数时就用量词:

a. 有明确数量边界的量词:

1). {n} =n 必须n个，不能多也不能少

2). {n,m} n个<= <=m个

3). {n,} n个<= 多了不限

b. 没有明确数量边界的量词:

1). \* 0个<= 可有可无，多了不限

2). ? 0个或1个 可有可无，最多一个

3). + 1个<= 至少一个，多个不限

(6). 两个规则中选其一匹配即可: 规则1|规则2

(7).希望将多个子规则分为一组先联合匹配，再和分组外的其他规则联合匹配:

(多个子规则)

(8). 匹配特殊位置: 3个

a. 字符串的开头位置: ^

b. 字符串的结尾位置: $

c. 英文句子中的单词的左右边界: \b

2. String家提供的正则相关函数: 3件事

(1). 查找敏感词: 4种情况

a. 查找一个固定的敏感词出现的位置:   
var i=str.indexOf("敏感词")

//如果找不到，返回-1

b. 用正则查找多种敏感词出现的位置:   
var i=str.search(/正则/i)

//如果找不到，返回-1

c. 查找敏感词的内容:

1). 查找第一个敏感词的内容和位置:   
 var arr=str.match(/正则/i)

//arr: [ 0:"敏感词内容", index:敏感词位置 ]

//如果找不到返回null

2). 查找所有敏感词的内容，不关心位置:   
 var arr=str.match(/正则/ig)

//arr: [ 敏感词1, 敏感词2, ... ]

//如果找不到返回null

d. 查找每个敏感词的内容和位置: reg.exec

|  |
| --- |
| 补: js中所有数组底层本质都是关联数组(下标都为字符串)  1. 访问数组中元素值的标注写法:  arr["下标"]  2. 简写:  a. 如果下标为自定义字符串名称，可简写为:  arr.自定义名称的下标  b. 如果下标为数字内容的字符串，可简写为:  arr[数字下标] |

|  |
| --- |
| 总结: 查找方法的返回值规律  1. 如果原函数返回的是下标位置i，如果找不到，都返回-1  2. 如果原函数返回的是一个数组arr或一个对象obj，如果找不到，都返回null  3. （未完...待续...） |

(2). 替换敏感词: 2种

a. 简单替换:   
变量=str.replace(/正则/ig, "新值")

b. 高级替换:   
变量=str.replace(/正则/ig, function(形参){

return 根据本次敏感词动态生成一个新值

})

c. 删除敏感词:   
变量=str.replace(/正则/ig, "")

(3). 切割字符串:

a. 简单切割:   
var arr=str.split("切割符")

b. 复杂切割:   
var arr=str.split(/正则/i)

c. 打散字符串为字符数组:   
var arr=str.split("")

2. RegExp对象:

(1). 创建正则表达式对象:

a. 如果正则是固定的:   
var reg=/正则/ig

b. 如果正则需要动态生成:   
var reg=new RegExp("正则",ig)

(2). 验证字符串格式:   
var bool=reg.test(str)   
reg必须同时前加^后加$

(3). 既查找每个关键词的内容又查找每个关键词的位置: (待续)

今日对应小程序视频列表:

小程序->在线->JSCORE->day02 0. 查找敏感词 查找一个固定的...

1. 查找一个敏感词的内容和位置match...

3. 替换 str.replace() 简单替换...

4. 切割str.split() ...

5. new RegExp()

6. 验证字符串格式 reg.test()

作业:

1. 复习小程序今日问题清单(小程序->首页->JSCORE->day02)

2. 看小程序视频复习和预习声明提前：

小程序->在线->JSCORE->day03 2. 声明提前

3. 声明提前完整笔试题

3. 看小程序视频复习简单作用域和按值传递

小程序->在线->JSCORE->day03 6. 简单 作用域 按值传递

4. 看小程序视频预习RegExp的exec函数:

小程序->在线->JSCORE->day02 5. new RegExp() 动态生成正则表达式

小程序->在线->JSCORE->day03

1.1. 验证字符串格式

1.2. 查找敏感词 既查找每个敏感词的内容...