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). 只查找一个固定的敏感词出现的位置:

a. var 下标位置=字符串.indexOf("要找的敏感词")

位置 的 要找的敏感词

b. 意为: 在字符串中，查找"敏感词"第一次出现的位置。

c. 返回值:

1). 如果找到，返回下标

2). 如果每找到，返回-1

d. 示例: 使用indexOf查找敏感词我草:

1\_indexOf.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>  </head>  <body>    <!--所有js代码必须放在body结尾-->    <script>      //1. 弹出一个输入框，请用户输入消息内容      var input=prompt("请输入消息内容");      //2. 用indexOf查找用户输入的内容中是否包含敏感词"我草"      var i=input.indexOf("我草");      //如果找到敏感词，就返回我草的下标位置      //如果没找到敏感词才返回-1      //3. 如果用户输入的内容中找到了敏感词"我草"      if(i!=-1){        //就在网页上显示红字:包含敏感词，禁止发送  //(向)  网页(中)写("一条HTML片段代码")        document.write(          //复习第一阶段: 模板字符串          `<h1 style="color:red">包含敏感词，禁止发送</h1>`        )      }else{//4. 否则如果用户输入的内容中没找到敏感词，        //就在网页上显示绿字:然哥说: xxx        document.write(          //复习第一阶段: 模板字符串          `<h1 style="color:green">然哥说：${input}</h1>`        )      }    </script>  </body>  </html>  运行结果： |

e. 问题: 只能防住一种敏感词，无法防住多种不同的敏感词！

(2). 用正则表达式模糊查找多种敏感词的位置:

JS语言的地盘 正则的地盘

a. var 下标位置=字符串.search(/正则表达式/i)

b. 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位置

c. 返回值: 同indexOf

1). 如果找到敏感词，返回敏感词的下标位置

2). 如果没找到敏感词，返回-1

d. 问题: 正则表达式默认区分大小写的！如何让正则忽略大小写？(既能匹配大写，又能匹配小写)

e. 解决: 只要在正则表达式第二个/后加后缀i即可！表示ignore——忽略大小写的意思。

f. 示例: 使用search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">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>  </head>  <body>    <!--所有js代码必须放在body结尾-->    <script>      //1. 弹出一个输入框，请用户输入消息内容      var input=prompt("请输入消息内容");      //2. 用search查找用户输入的内容中是否包含敏感词"我草"      var i=input.search(/([我卧]|wo)\s\*([草艹槽操]|cao)/i);      //如果找到敏感词，就返回我草的下标位置      //如果没找到敏感词才返回-1      //3. 如果用户输入的内容中找到了敏感词"我草"      if(i!=-1){        //就在网页上显示红字:包含敏感词，禁止发送  //(向)  网页(中)写("一条HTML片段代码")        document.write(          //复习第一阶段: 模板字符串          `<h1 style="color:red">包含敏感词，禁止发送</h1>`        )      }else{//4. 否则如果用户输入的内容中没找到敏感词，        //就在网页上显示绿字:然哥说: xxx        document.write(          //复习第一阶段: 模板字符串          `<h1 style="color:green">然哥说：${input}</h1>`        )      }    </script>  </body>  </html>  运行结果: |

g. 问题: 只能返回下标位置，无法返回敏感词的内容

(3). 查找多种敏感词的内容:

对应小程序视频: 小程序->在线->JSCORE->day02 1. 查找一个敏感词的内容和位置match...

a. 只查找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)

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下标位置

3). 返回值:

i. 如果找到敏感词，就会返回一个数组:

数组:[

0:"敏感词的内容",

index: 敏感词的下标位置

]

ii. 如果没找到，返回null

4). 示例: 使用match查找敏感词的内容和位置：

3\_match.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>  </head>  <body>    <!--所有js代码必须放在body结尾-->    <script>      //1. 弹出一个输入框，请用户输入消息内容      var input=prompt("请输入消息内容");      //2. 用match查找用户输入的内容中是否包含敏感词"我草"      var arr=input.match(/([我卧]|wo)\s\*([草艹槽操]|cao)/i);      //假设input:      //那天，我去了她家，我说：我草！你家真大！      //0 1 2 3 4 5 6 78 9 0 1 2      //                       ↑      //arr:[ 0:"我草", index:12  ]      console.log(arr);      //如果找到敏感词，就返回数组      //如果没找到敏感词才返回null      //3. 如果用户输入的内容中找到了敏感词"我草"      if(arr!=null){        //就在网页上显示红字:包含敏感词，禁止发送  //(向)  网页(中)写("一条HTML片段代码")        document.write(          //复习第一阶段: 模板字符串          //标准:          // `<h1 style="color:red">在位置${arr["index"]},发现敏感词${arr["0"]},禁止发送</h1>`          //简写:          `<h1 style="color:red">在位置${arr.index},发现敏感词${arr[0]},禁止发送</h1>`        )      }else{//4. 否则如果用户输入的内容中没找到敏感词，        //就在网页上显示绿字:然哥说: xxx        document.write(          //复习第一阶段: 模板字符串          `<h1 style="color:green">然哥说：${input}</h1>`        )      }    </script>  </body>  </html>  运行结果: |

5). 问题: match默认只能找到第一个敏感词。如果一句话中包含多个敏感词，就找不到了！

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)

g - global 全部 ——

不加g，正则很懒，只找到一个敏感词就下班了！

加g，等于告诉正则必须找所有敏感词，找不完不许下班！

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。

3). 返回值:

i. 如果找到敏感词，就返回所有敏感词的内容组成的数组

ii. 如果没找到，返回null

4). 示例: 使用match+g查找所有敏感词的内容:

5\_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！";      //即想获得内容，又想获得位置      //var arr=str.match(/小[\u4e00-\u9fa5]/)      //想查找所有敏感词      var arr=str.match(/小[\u4e00-\u9fa5]/g)      console.log(arr);    </script>  </body>  </html>  运行结果: |

5). 问题: 只能获得所有敏感词的内容，而无法获得每个敏感词的位置了！

(4). 即查找每个敏感词的内容，又能查找每个敏感词的位置: reg.exec()

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: 将所有找到的敏感词都替换为一个统一的新值。

a. var 变量=字符串.replace(/正则表达式/ig, "新值")

b. 意为: 替换字符串中所有符合正则表达式要求的敏感词为一个统一的新值。

c. 强调: 因为字符串是不可变类型，所以，replace无权直接修改原字符串，只能返回新字符串。所以，将来使用replace时，都要用变量接住返回的处理后的新字符串副本。

d. 示例: 将字符串中所有以小字开头的人名都替换为\*\*

6\_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！";        //想将字符串中所有小字开头的人名替换为\*\*      str=str.replace(/小[\u4e00-\u9fa5]/g,"\*\*")      console.log(str);      var arr=[1,2,3];      arr.push(4);      console.log(arr);//1,2,3,4    </script>  </body>  </html>  运行结果: |

(2). 高级替换: 将不同的敏感词替换为不同的新值

进

↓

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

出← return 新值

})

b. 意为: 将字符串中每个符合正则表达式要求的敏感词都经过回调函数的处理，转化为新值，替换回字符串中。

c. 原理:

1). replace会用正则表达式去字符串中查找所有符合要求的敏感词

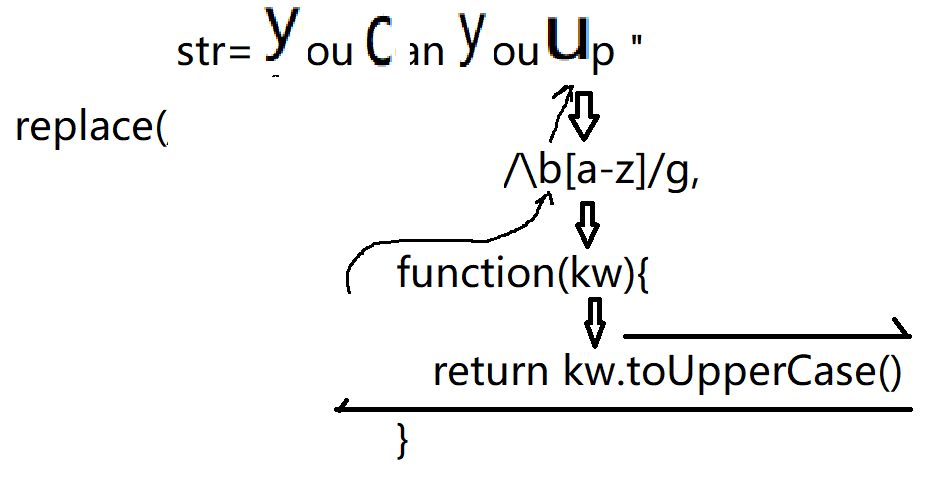
2). 每找到一个敏感词就会自动调用一次回调函数。

3). 每次调用回调函数时都会自动传入本次找到的敏感词的内容。

4). 在回调函数内部，对敏感词进行加工，获得新值，返回给replace函数。

5). replace得到回调函数返回的新值后，自动替换到字符串中本次找到的敏感词的位置！

6). 当所有敏感词替换完，replace返回替换后新字符串的副本。



d. 示例: 将每个单词首字母转大写

7\_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:      //将每个单词首字母改为大写字母:      var str="you can you up";      //       ↓   ↓   ↓   ↓      str=str.replace(        /\b[a-z]/g,        //       y c y u        //       ↓ ↓ ↓ ↓        function(keyword){          console.log(`自动调用了一次回调函数,keyword接住了${keyword},回调函数返回了${keyword.toUpperCase()}`);          return keyword.toUpperCase();          //            转为 大  写        }      )      console.log(str);//You Can You Up    </script>  </body>  </html>  运行结果: |

e. 对回调函数理解不深的同学可以看一下:

小程序->在线->JSCORE->day02 2.回调函数 callback 亮哥带饭

(3). 衍生操作: 删除敏感词: 其实就是把敏感词替换为""

a. var 变量=字符串.replace(/正则表达式/,"");

b. 示例: 去掉字符串开头结尾的空字符:

8\_replace\_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>      //高频鄙视题:      //定义三个函数，分别可以去掉字符串开头，结尾，以及同时去掉开头结尾的空字符。      //想实现去掉一个字符串开头(左边)的空字符      // function trimLeft(str){      //   //其实就是把原字符串中开头的一组空字符替换为""      //   return str.replace(/^\s+/,"");      //   //\*，连没有空格的情况也会匹配一次，但是，如果没有空格，是没必要执行替换操作的！      //   //+，只有有空格时，才匹配，才有必要执行替换操作      // }      // //想实现去掉一个字符串结尾(右边)的空字符      // function trimRight(str){      //   return str.replace(/\s+$/,"");      // }      // //想实现同时去掉开头和结尾的空字符      // function trim(str){      //   return str.replace(/^\s+|\s+$/g,"")      // }      // var str="   zhang    dong    ";      // console.log(trimLeft(str));      // //"zhang    dong    "      // console.log(trimRight(str));      // //"   zhang    dong"      // console.log(trim(str));      // //"zhang    dong"      //其实: ES5~ES6等新标准中已经内置了三个trim函数！      //功能一样      //但是，必须按调用原生函数的格式调用:      //字符串.函数名()      var str="   zhang    dong    ";      console.log(str.trimLeft());      //"zhang    dong    "      console.log(str.trimRight());      //"   zhang    dong"      console.log(str.trim());      //"zhang    dong"    </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: 切割符是固定不变的

a. var 数组=字符串.split("固定切割符")

b. 意为: 将字符串按指定的切割符，切割为多段子字符串，保存在数组中

c. 强调: 在切割后的结果中，不再包含切割符

d. 示例: 切割电子邮件为用户名和域名，切割英文句子为单词数组

9\_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 email="zhangdong@tedu.cn";      //想获得电子邮件中的用户名和域名      //所以，要按@切割      var arr=email.split("@");      console.log(arr);      //arr:["zhangdong","tedu.cn"]      //          0          1      console.log(`用户名：${arr[0]}`);      console.log(`域名: ${arr[1]}`);      var str="no zuo no die";      //希望将一条英文的句子切割为单词的数组      //所以，按空格切割      var arr=str.split(" ");      console.log(arr);    </script>  </body>  </html>  运行结果:  (2) ["zhangdong", "tedu.cn"]  0: "zhangdong"  1: "tedu.cn"  length: 2  \_\_proto\_\_: Array(0)  用户名：zhangdong  域名: tedu.cn  (4) ["no", "zuo", "no", "die"] |

(2). 复杂切割: 切割符不是固定不变的，但是可以找到规律

a. var 数组=字符串.split(/正则/i);

b. 意为: 将字符串按所有符合正则要求的敏感词作为切割符，切割为多段子字符串。

c. 强调: 虽然split中不加g，但是自动可以找所有敏感词！

d. 示例: 将一个英文句子切割为单词列表（单词之间不确定有几个空格）

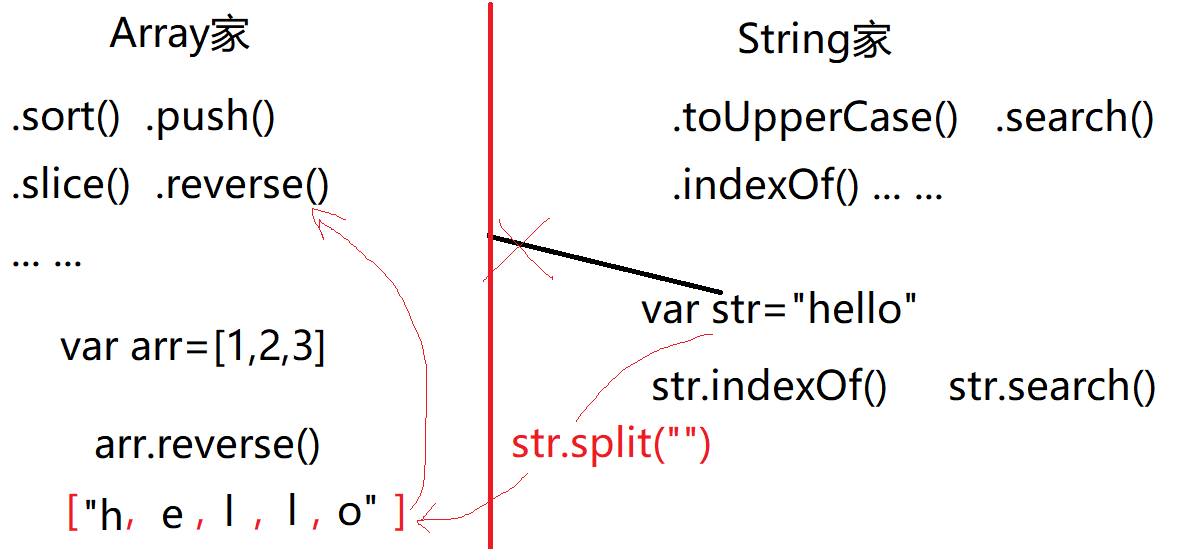
10\_split2.html

|  |
| --- |
| <!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="no  zuo     no die";      //也想切割出单词列表:      var arr=str.split(/\s+/);      console.log(arr);    </script>  </body>  </html>  运行结果:  ["no", "zuo", "no", "die"] |

(3). 衍生: 打散字符串为字符数组:

a. 问题: 字符串无法使用数组家一些好的函数！

b. 解决: 先将字符串打散为字符数组，就变成数组家孩子，就可以随意使用数组家函数了



c. 如何: 用split按字与字之间的缝隙切割字符串:

var 数组=字符串.split("");

d. 示例: 翻转一个字符串:

11\_split3.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 arr=[1,2,3,4,5];      arr.reverse();      console.log(arr);      //高频鄙视题：      //颠倒一个字符串:      var str="hello";      //先将字符串打散为字符数组:      var arr=str.split("");      console.log(arr);      //翻转数组中的字母      arr.reverse();      console.log(arr);      //将数组中的字符无缝拼接为一个字符串      str=arr.join("");      //复习数组函数:join和toString      console.log(str);//olleh    </script>  </body>  </html>  运行结果:  [5, 4, 3, 2, 1]  (5) ["h", "e", "l", "l", "o"]  (5) ["o", "l", "l", "e", "h"]  olleh |

|  |
| --- |
| 补: 字符串是不可变类型:  1. 什么是不可变类型: 一旦创建后，内容或值不能改变！  2. 因为字符串是不可变类型，所以，所有字符串家提供的函数，都无权直接修改原字符串。只能创建一个新的修改后的字符串副本返回！所以所有的字符串函数，要想获得结果，必须用变量接住新的副本，才能的到修改后的新值！  3. 数组就是可变类型: 可以随时向原数组中加入新值，而不会产生副本。  4. 所以，数组家绝大多数函数，不需要使用变量接住返回值，就可直接修改原数组的！ |

|  |  |
| --- | --- |
| 补: 数组的本质:  1. 索引数组: 下标都是数字的数组  var arr=[1,2,3];  内存中:  arr:[  下标: 元素值  0: 1,  1: 2,  2: 3  ]  //想输出arr中1位置的元素值: arr[1]  2. 关联数组: 下标都是自定义字符串的数组  var ym=[];  ym["math"]=89;  ym["chs"]=59;  ym["eng"]=91;  内存中: [  math: 89,  chs: 59,  eng: 91  ]  //想输出ym的语文成绩: ym["chs"]  3. 其实在js底层，只有一种数组——关联数组:  (1). 无论是索引数组的数字下标，还是关联数组的自定义字符串下标，一旦进入内存，都变成了字符串下标!  (2). 所以，无论访问索引数组还是访问关联数组中的元素，标准写法都是: 数组名["下标名"]  (3). 因为每次都写[""]，会很繁琐，所以js语法中提供了简写：  a. 如果下标名称为自定义的字符串下标，则:  数组名.下标名  b. 如果下标是数字，则不能用.简写（.数字和js中小数写法冲突了！）。只能用[]简写:  数组名[下标数字]  4. 总结:  (1). 今后无论访问索引数组还是关联数组，都可用["下标名"]方式——标准写法！  (2). 自定义字符串下标可用.简写  (3). 数字下标只能用[]简写  5. 示例: 分别用标准写法和简写来访问不同数组的元素  4\_arr.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>      //创建索引数组:      var arr=[1,2,3];      //       0 1 2      console.log(arr);      //标准:      console.log(arr["1"]);//2      //简写:      console.log(arr[1]);//2      //创建关联数组:      var ym=[];      ym["math"]=89;      ym["chs"]=59;      ym["eng"]=91;      console.log(ym);      //标准:      console.log(ym["chs"]);      //简写:      console.log(ym.chs);    </script>  </body>  </html>  运行结果： | |

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...