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！

那些自己发现不了问题，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->day01、day02

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day01、day02

2部分: 1. 课上重点知识和案例的回放！2. 重点高频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

扩展: 鄙视题1.1 使用字符串切割方式解析url

扩展: 鄙视题1.2 使用正则表达式方式解析url

扩展: 鄙视题1.3 使用正则表达式0宽断言方式接续url

扩展: 鄙视题1.4 解析url的各个组成部分 源代码

3. 单词列表: 小程序->单词->JSCORE->day01、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！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

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的函数: 3件事

对应小程序视频: 小程序->在线->JSCORE->day01 5.查找敏感词查找一个固定的敏感词的位置

1. 查找敏感词：用正则查找一句话中包含的敏感词，共4种情况:

**(1).** 只查找一个固定的敏感词出现的位置:

a. var i=str.**indexOf**("敏感词",fromi)

位置的敏感词

b. 意为: 在str中查找**第一个**"敏感词"出现的**下标**位置

c. 返回值:

1). 如果找到，返回敏感词第一个字在字符串中的下标位置

2). 如果没找到，返回**-1**

d. 问题: **不支持正则**！只能找一种固定的敏感词！

e. 示例: 用indexOf查找敏感词:

ftp上的.html文件，要右键->链接另存为下载

1\_indexOf.html

|  |
| --- |
| <!--按住ctrl+滚轮，缩放字体-->  <!--输入!按回车，可生成网页基本结构-->  <!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>    <!--第三阶段必须都用右键 live server运行！-->    <!--网页中如果写js，都要写在script标签中-->    <script>      //请用户输入一条消息内容      var msg=prompt("请输入消息内容");      //查找消息中是否包含敏感词我草      var i=msg.indexOf("我草");      //如果找到敏感词，返回敏感词位置      //如果没找到敏感词，返回-1      //如果包含敏感词      if(i!=-1){        //就在网页中提示红字: 包含敏感词，禁止发送        //``和${}不会的同学去第一阶段复习模板字符串  //  (向) 页面(中)写  (一条HTML代码片段)        document.write(`<h1 style="color:red">包含敏感词，禁止发送</h1>`)      }else{//否则如果不包含敏感词        //就在网页中提示绿字：然哥说: 消息内容        document.write(`<h1 style="color:green">然哥说: ${msg}</h1>`)      }    </script>  </body>  </html>  运行结果： |

**(2)**. 用正则查找多种敏感词出现的位置:

a. var i=str.**search**(**/**正则**/i**)

查找

b. 意为: 在字符串str中查找第一个符合正则要求的敏感词出现的位置

c. 返回值: 同indexOf完全一样！

1). 如果找到，返回敏感词第一个字在字符串中的下标位置

2). 如果没找到，返回**-1**

d. 问题: 虽然在猫头鹰里有个选项可以忽略大小写，但是正则表达式天生是区分大小写的！

e. 解决: 在第二个/后加后缀 **i** 即可，意为ignore **忽略大小写**

f. 问题: 只能查找敏感词的位置，无法获得敏感词的内容

g. 示例: 使用search代替indexOf查找敏感词

ftp上的.html文件，要右键->链接另存为下载

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>    <script>      //请用户输入一条消息内容      var msg=prompt("请输入消息内容");      //查找消息中是否包含符合正则要求的敏感词      var i=msg.search(/([我卧]|wo)\s\*([艹槽操草]|cao)/i);      //如果找到敏感词，返回敏感词位置      //如果没找到敏感词，返回-1      //如果包含敏感词      if(i!=-1){        //就在网页中提示红字: 包含敏感词，禁止发送        //``和${}不会的同学去第一阶段复习模板字符串  //  (向) 页面(中)写  (一条HTML代码片段)        document.write(`<h1 style="color:red">包含敏感词，禁止发送</h1>`)      }else{//否则如果不包含敏感词        //就在网页中提示绿字：然哥说: 消息内容        document.write(`<h1 style="color:green">然哥说: ${msg}</h1>`)      }    </script>  </body>  </html>  运行结果: 同上一个示例 |

**(3)**. 查找敏感词的内容: 2种:

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...

a. 只查找**第一个**敏感词的内容和位置：

1). var arr=str.**match**(/正则/i)

匹配

2). 意为: 在字符串str中查找第一个符合正则要求的敏感词的内容和位置

3). 返回值: 2种:

i. 如果找到，返回一个**数组**，2个房间(元素):

**arr: [ 0:"敏感词内容", "index":敏感词下标位置 ]**

如果想获得本次找到的关键词的内容: 标准: arr["0"] 简写:arr[0]

如果想获得本次找到的关键词的下标位置: 标准: arr["index"] 简写arr.index

说明: **0和index是不能改名**，因为数组是match函数返回给我们的，不是我们自己创建的！

ii. 如果没找到，返回**null**

4). 问题: 如果一句话中包含多个敏感词，match默认只能找第一个敏感词，无法获得所有敏感词

5). 示例: 使用match查找敏感词的内容和位置:

4\_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>    <script>      //请用户输入一条消息内容      var msg=prompt("请输入消息内容");      //查找消息中第一个敏感词的内容和位置      var arr=msg.match(/([我卧]|wo)\s\*([艹槽操草]|cao)/i);      //如果找到敏感词，返回数组      //如果没找到敏感词，返回null      //如果包含敏感词      if(arr!=null){        //就在网页中提示红字: 在位置?发现敏感词?，禁止发送        //``和${}不会的同学去第一阶段复习模板字符串  //  (向) 页面(中)写  (一条HTML代码片段)        document.write(`<h1 style="color:red">在位置 ${arr.index} 发现敏感词"${arr[0]}"，禁止发送</h1>`)      }else{//否则如果不包含敏感词        //就在网页中提示绿字：然哥说: 消息内容        document.write(`<h1 style="color:green">然哥说: ${msg}</h1>`)      }    </script>  </body>  </html>  运行结果: |

b. 查找所有敏感词的内容:

1). var arr=str.match(/正则/ig)

2). **g: global，查找全部敏感词**

3). 返回值:

i. 如果找到敏感词，则返回一个数组，只包含所有敏感词的内容，不再包含每个敏感词的位置。

ii. 如果找不到敏感词，返回null

4). 问题: 不加g时，只能找一个，无法找所有；加g时，能找所有，但是无法获得位置。

5). 示例: 使用match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！";      //希望查找出str中所有以小字开头的人名      //因为汉字没有大小写之分，所以不用加i。      var arr=str.match(/小[\u4e00-\u9fa5]/g);      console.log(arr);    </script>  </body>  </html>  运行结果: |

(4). 既查找每个关键词的内容，又查找每个关键词的位置:

String类型没有提供这种函数  
 RegExp类型提供了一个**reg.exec()**函数

|  |
| --- |
| 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**，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，如果找不到，都返回**null**  强调: 如果一个函数有可能返回null！一定先判断是不是null，再使用。并且为null的情况提供备选方案！  3. （未完...待续...） |

对应小程序视频: 小程序->在线->JSCORE->day02 3. 替换 str.**replace()** 简单替换...

2. 替换敏感词：将找到的敏感词替换为新内容，2种:

(1). 简单替换: 将所有敏感词都替换为统一的新值

a. **var 变量=str.replace(/正则/ig, "新值")**

**替换**

b. 意为: 替换字符串str中所有符合正则要求的敏感词为统一的新值！

c. 强调: 因为所有字符串类型都是不可变类型，所以，replace无权修改原字符串，而是返回替换后的新字符串。原字符串保持不变。所以，必须用变量接住返回值，才能获得替换后的新字符串。

d. 示例: 使用replace替换所有人名为\*\*

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！";      // var str="you can you up!no can no bibi!";      //先用match在替换之前找一次      var arr=str.match(/小[\u4e00-\u9fa5]/g);      //然后查找结果的元素个数，也就是将来要替换的个数！      console.log(arr);//null      //希望替换str中所有以小字开头的人名为\*\*      str=str.replace(/小[\u4e00-\u9fa5]/g, "\*\*");      console.log(str);      //还想显示共替换了多少处！      //基础一般:      //如果match找到了敏感词，才输出arr.length      if(arr!=null){        console.log(`共替换了${arr.length}处`);      }else{//否则如果match没找到敏感词，就直接输出0处代替        console.log(`共替换了0处`);      }      //学有余力: //复习第一阶段三目运算      //console.log(`共替换了${arr!=null?arr.length:0}处`)    </script>  </body>  </html>  运行结果:  ["小红", "小亮", "小红", "小然", "小红"]  老师:请用\*\* 我的 朋友造句。\*\*:\*\*是我的朋友。\*\*:朋友！\*\*是我的！  共替换了5处 |

(2). 高级替换: 根据每次找到的敏感词不同，动态替换成不同的新值

a. var 变量=str.replace(/正则/ig, function(形参){

return 根据本次"形参"获得的敏感词内容，动态生成一个新值

})

b. 原理: 回调函数的学习，可以靠打桩来判断执行次数，传入参数和返回值。

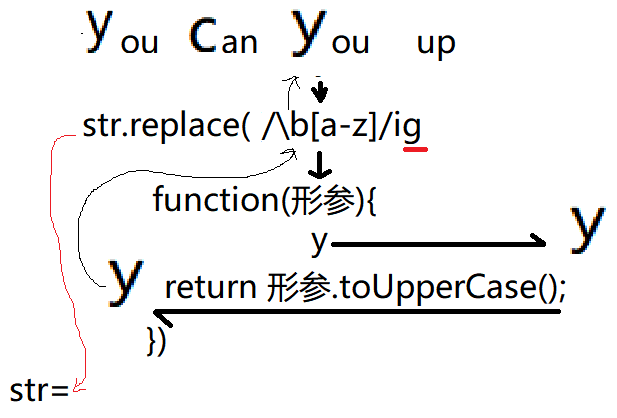
1). replace会用正则表达式查找str中所有符合正则要求的敏感词

2). 每找到一个敏感词就自动调用一次回调函数。——信任

3). 每次调用回调函数时，都自动将本次找到的一个敏感词内容传给形参变量。——信任

4). 回调函数中根据本次找到的敏感词内容，动态生成一个新值，返回出来

5). replace函数获得回调函数返回的新值，并自动替换到字符串中本次找到的敏感词位置



c. 示例: 鄙视题: 将字符串中每个单词首字母改为大写:

5\_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>    <script>      //鄙视题: 将字符串中每个单词首字母转为大写      var str="you can you up";      //                              敏感词:y    c   y   u      //                                     ↓      str=str.replace(/\b[a-z]/ig, function(形参){        //打桩可以让回调函数执行过程现原形！        console.log(`调用了一次回调函数,形参=${形参},返回${形参.toUpperCase()}`);        //形参: 4种可能: y c y u        return 形参.toUpperCase();        //返回: 也4种可能Y C Y U      })      console.log(str);//"You Can You Up";    </script>  </body>  </html>  运行结果:  调用了一次回调函数,形参=y,返回Y  调用了一次回调函数,形参=c,返回C  调用了一次回调函数,形参=y,返回Y  调用了一次回调函数,形参=u,返回U  You Can You Up |

(3). 衍生操作: **删除敏感词: 其实就是将敏感词替换为空字符串**

**变量=str.replace(/正则/ig, "")**

示例: 鄙视题，定义三个函数，分别去掉字符串开头的空字符，去掉结尾的空字符，以及同时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>      //鄙视题三个函数:      //修剪      //trimLeft() 可去掉字符串开头的空字符      //                可以改名      // function trimLeft(字符串){      //   var str2=字符串.replace(/^\s+/,"");      //   return str2;      // }      //trimRight() 可去掉字符串结尾的空字符      // function trimRight(字符串){      //   var str2=字符串.replace(/\s+$/,"");      //   return str2;      // }      //trim() 可去掉字符串开头和结尾的空字符      // function trim(字符串){      //   //因为/^\s+|\s+$/在字符串中匹配到两组敏感词，一组是开头的空字符，一组是结尾的空字符。如果想把两组空字符都替换，必须加g。      //   return 字符串.replace(/^\s+|\s+$/g,"")      // }      //测试:      var str="   hello   world    ";      // console.log(trimLeft(str));//"hello   world    "      // console.log(trimRight(str));//"   hello   world"      // console.log(trim(str));//"hello   world"        //其实ES5和ES6标准中，已经新增了现成的trim系列函数，不用自己定义。      console.log(str.trimLeft());      console.log(str.trimRight());      console.log(str.trim());    </script>  </body>  </html>  运行结果：  hello world  hello world  hello world |

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

3. 切割字符串: 按指定的切割符，把一个完整的字符串，切割为多段子字符串

(1). 简单切割: 切割符是固定的

a. **var arr=str.split("切割符")**

切割

b. 按照字符串str中指定的切割符，将字符串切割为多段子字符串

c. 返回值: 包含切割后的多段子字符串的**数组**

d. 强调: 切割后的结果中，不包含切割符的！

(2). 复杂切割: 切割符不是固定的，但是却能找到规律！

a. var arr=str.split(/正则/);

b. 意为: 按str中所有符合正则要求的切割符将字符串切割为多段子字符串

c. 强调: split中不用加g，也能自动找到所有符合条件的切割符！

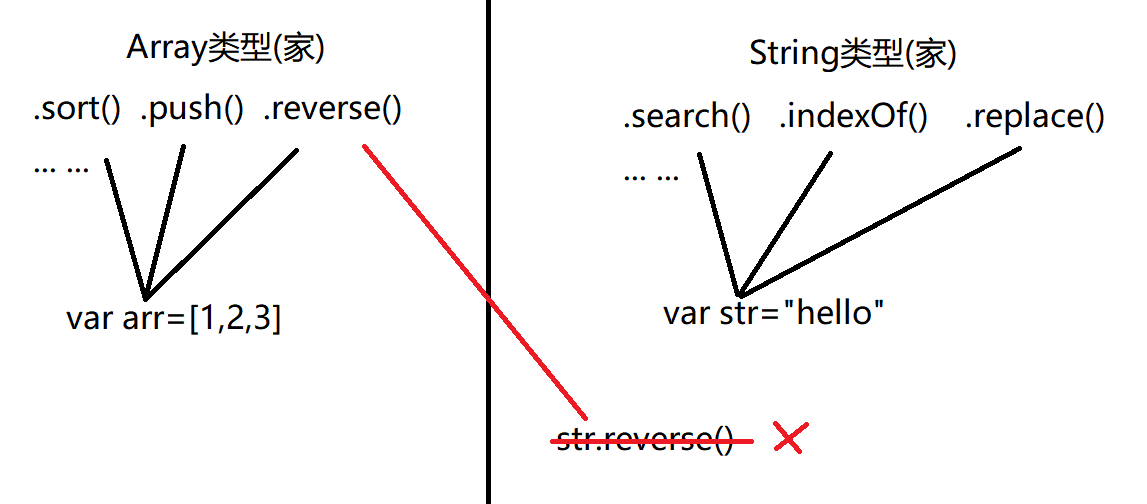
(3). 示例: 分别使用固定切割符和正则表达式切割字符串:

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>      //想把电子邮件切割为用户名和域名两部分      var email="zhangdong@tedu.cn";      //                  |      //      ["zhangdong","tedu.cn"]      //            0          1      //按@将字符串email一分为二      var arr=email.split("@");      console.log(arr);      var uname=arr[0];      var domain=arr[1];      console.log(uname, domain);      //想把一个英文的句子切割为单词数组      var str="you can you up";      //按空格切割str      var arr=str.split(" ");      console.log(arr);      //如果每个单词之间的空格个数不确定有几个？      var str="you       can you   up";      //按一个或多个空格切割str      var arr=str.split(/\s+/);      console.log(arr);    </script>  </body>  </html>  运行结果：  (2) ["zhangdong", "tedu.cn"]  zhangdong tedu.cn  (4) ["you", "can", "you", "up"]  (4) ["you", "can", "you", "up"] |

(4). 固定套路:

a. 问题: 虽然数组和字符串都有下标，但是因为字符串和数组不是同一类型，所以字符串用不了数组家的函数！



b. 解决: 先将字符串打散为一个一个字符组成的字符数组！然后就可以调用数组家的函数。只不过，加工完成后，一定再拼接回字符串！

c. 如何将字符串打散为字符数组: 用空字符串""切割

var arr=str.split("");

d. 示例: 翻转字符串

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>      //数组家现成的reverse函数      var arr=[1,2,3,4,5];      arr.reverse();          //翻转      console.log(arr);        //鄙视题: 翻转字符串：      var str="helloworld";      //       0123456789.length=10      //console.log(str[1]);//e      //console.log(str.length);//10      //数组家不是有reverse()函数吗?      //错误:      //str.reverse();//报错：str.reverse is not a function      //正确:      var arr=str.split("");//1. 打散为字符数组      console.log(arr);      arr.reverse();//2. 翻转字符数组的内容      console.log(arr);      //错误: join()什么都不写，默认用,连接每个字符      //str=arr.join();      //正确:      str=arr.join("");//3. 将翻转后的字符数组无缝拼接回字符串      console.log(str);//"dlrowolleh"    </script>  </body>  </html>  运行结果:  (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 |

|  |
| --- |
| 补第一阶段: **字符串**其实是一种**不可变**类型！  1. 不可变类型: 一旦创建值不可改变，只能整体替换！  直接后果: 所有字符串家的函数都无权直接修改原字符串。只能创建新字符串保存加工后的结果，并返回新字符串。原字符串始终保持不变！  强调: 所有字符串家的函数执行后的结果，必须用变量接住，才能保存下来！  2. 可变类型: 创建后，还可改变其内容。比如: 数组  比如: 数组排序arr.sort(), 向数组中追加一个新元素arr.push(xxx), 都**直接修改原数组**，无需用变量=接住 |

|  |  |
| --- | --- |
| 补第一阶段数组本质:  1. 索引数组: 下标都是数字的数组，比如: var arr=[1,2,3];  结果: [  0: 1,  1: 2,  2: 3  ]  2. 关联数组: 下标都是自定义名称，比如:  var ym=[];  ym["数学"]=89,  ym["语文"]=61,  ym["英语"]=91  结果: [  "数学": 89,  "语文": 61,  "英语": 91  ]  3. 其实: js中所有数组以及对象底层都是关联数组！本质上是没有索引数组！  (1). 数组和对象底层，其实都是名值对儿的集合  (2). 每个数组的下标和每对象属性的属性名，都是字符串类型  (3). 无论访问数组的元素，还是访问对象的属性，标准写法都是["下标名称"]  (4). 简写:  a. 如果下标名称**不是数字**，可简写为"**.下标名称**"  b. 如果下标**是数字**，只能简写为"**[下标]**"，不能用.  因为".数字"和程序中的小数点冲突了！  4. 示例: 揭露数组和对象底层的原理和下标访问方式  3\_arr\_obj.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];      var ym=[];      ym["数学"]=89;      ym["语文"]=61;      ym["英语"]=91;      var lilei={        姓名:"Li Lei",        年龄:11      };      console.log(arr);      console.log(ym);      console.log(lilei);      //访问每个数组和对象中的一个成员，用标准写法      console.log(arr["0"]);      console.log(ym["数学"]);      console.log(lilei["姓名"]);      //用简写      console.log(arr[0]);      console.log(ym.数学);      console.log(lilei.姓名);    </script>  </body>  </html>  运行结果： | |
|  |

中午作业: 必看: 小程序->在线->JSCORE->day02 2. 回调函数callback 亮哥带饭

总结: 以下内容根本不用背！

1. 正则表达式:

(1). 最简单的正则: 一个关键词原文

(2). 如果某一位上字符有多种备选字时: [字符集]

(3). 如果[]中部分字符是连续的: [x-x]

(4). 常用的字符集简写:

[0-9] [a-z] [A-Z]

[A-Za-z] [0-9A-Za-z]

[\u4e00-\u9fa5]

(5). 预定义字符集:

\d \w=[0-9A-Za-z\_] \s .

(6). 量词:

a. {n} 必须n个,

{n,m} n个<= <=m个,

{n,} n个<= 多了不限

b. \* >=0个 ,

? 0个或1个 ,

+ >=1个

(7). 两个规则二选一匹配: |

(8). 将多个子规格分为一组先匹配，再和其它规则联合匹配: ()

(9). 位置: ^开头, $结尾, \b单词边界

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",从哪个位置开始找)

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

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)

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

d. 查找每个敏感词的内容和位置:

(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("")

|  |
| --- |
| 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，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，如果找不到，都返回null  3. （未完...待续...） |

|  |
| --- |
| 数组本质: js中所有数组以及对象底层都是关联数组！本质上是没有索引数组！  (1). 数组和对象底层，其实都是名值对儿的集合  (2). 每个数组的下标和每对象属性的属性名，都是字符串类型  (3). 无论访问数组的元素，还是访问对象的属性，标准写法都是["下标名称"]  (4). 简写:  a. 如果下标名称不是数字，可简写为".下标名称"  b. 如果下标是数字，只能简写为"[下标]" |

今日对应小程序视频列表:

小程序->在线->JSCORE->day01 5. 查找敏感词 查找一个固定的敏感词的位置...

小程序->在线->JSCORE->day02 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() ...

作业:

1. 复习小程序今日问题清单(小程序->首页->JSCORE->day01、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

**0宽断言**

ab(?=[A-Z])——匹配ab后面的一个字母是大写的字符串ab

ab(?![A-Z])——匹配后面不是大写字母的字符串ab

**千分符** 29,999,999.00

\d(?=\d{3}$)——匹配到结尾之前间隔一组3位数的一个数字

\d(?=\d{3}\d{3}$)——匹配到结尾之前间隔两组3位数的一个数字

规律：\d(?=(\d{3})+$)——匹配到结尾之前间隔n组三位数的一个数字

考虑结尾可能有小数部分：

\d(?=(\d{3})+(\.|$))

结合JS：

$1表示（\d）匹配到的字符，，$1是replace固定的

str=str.replace(/(\d)(?=(\d{3})+(\.|$)/g,”$1,”);

**密码错误：**

至少包含一位大写：^(?![0-9a-z]+$)

至少包含一个数字：^(?![A-Za-z]+$)

字母数字的组合，不能有特殊符号：^[9-0A-Za-z]+$

总结：^(?![9-0a-z]+$)(?![A-Za-z]+$)[9-0A-Za-z]+$