准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->JSCORE->day02

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->JSCORE->day02

如果因为某些原因错过直播讲解，不用着急！直接看小程序中视频的回看即可。

如果tmooc不能用了，立刻看小程序视频，小程序视频比tmooc要全，甚至有扩展的新知识。

如果学有余力，将来有能力拿更高薪资，小程序中所有视频必须都要看。但是不必现在都看完。只要在笔试面试前都看完即可！

问题: 个人版小程序有功能限制：禁止在线播放视频

解决: 长按视频连接地址，完整复制视频连接地址和验证码到浏览器打开。通过百度网盘下载后观看。

3. 今日课程中包含的英语单词的翻译：小程序->单词->JSCORE->day02

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例，至少写三遍:

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩20天）

1. 上课时那句话没听懂，立刻问！

2. 笔记中那句话看不懂，案例中那句话看不懂，立刻问

3. 小程序视频中那句话看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

正课:

1. String提供的正则相关的函数

2. RegExp对象

一. String提供的正则相关的函数: 3件事:

1. 查找关键词: 4种情况

(1). 查找一个固定的关键词出现的位置:

var i=str.indexOf("关键词",fromi)

问题: 只能查找一种固定的关键词，只要关键词一变，就查不出来了。

(2). 用正则模糊查找多种关键词的位置:

var i=str.search(/正则/i)

问题: 只能返回关键词的位置，无法返回关键词的内容！

(3). 用正则模糊查找多种关键词的内容:

a. var arr=str.match(/正则/i)

b. 意为: 在str中查找第一个符合正则表达式要求的关键词的内容和位置

c. 返回值:

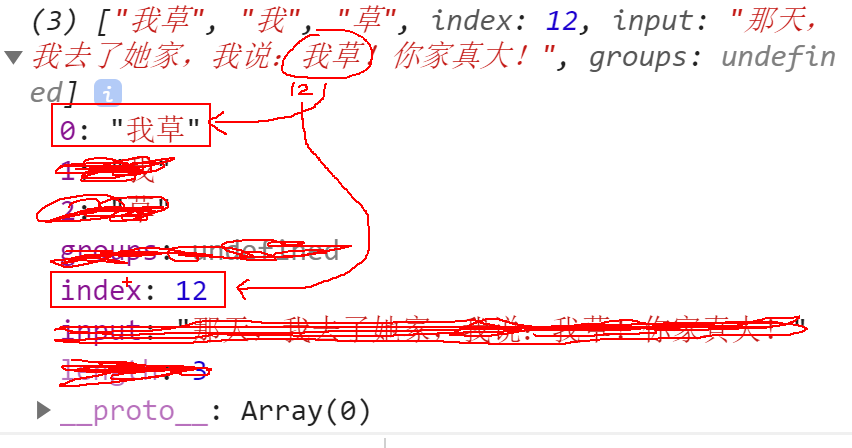
1). 如果match找到关键词，就返回一个数组:

arr : [ "0": "关键词内容", "index":"关键词的下标位置i" ]

如果想获得本次找到的敏感词的内容: 标准: arr["0"] 可简写为 arr[0]

如果想获得本次找到的敏感词的位置: 标准: arr["index"] 可简写为 arr.index

0和index禁止改名！因为match()是内置的函数，返回的结果格式也是固定的！不是人为可以修改的！所以我们只能遵照match()的要求使用！而无法自定义返回值的结构。



2). 如果match没找到关键词，返回null —— 造成错误最多的根源！

强调: 将来如果一个函数有可能返回null时，在使用这个函数的返回值之前，都必须先判断返回值是不是null，再决定如何使用返回值！必须为null的情况提供备选方案，保证程序在任何情况下都不会报错！

d. 示例: 既查找敏感词的内容，又查找敏感词的位置

|  |
| --- |
| <script>  //请用户输入一条消息内容  var msg=prompt("请输入消息内容");  // console.log(`用户输入了: ${msg}`);  //在消息内容中查找符合正则表达式要求的敏感词的位置  var arr=msg.match(/([我卧]|wo)\s\*([艹槽草操]|cao)/i);  //console.log(arr);  //arr : ["0":"我草", ..., "index":12]  //如果找到敏感词（找到敏感词的位置i不是-1）  if(arr!=null){  //就在网页上显示"在位置i发现敏感词"  //向网页中 写 (`一条html代码`)  // document.write(`<h1 style="color:red">在位置${arr["index"]}发现敏感词${arr["0"]}，禁止发送</h1>`);  document.write(`<h1 style="color:red">在位置${arr.index}发现敏感词${arr[0]}，禁止发送</h1>`);  }else{//否则如果没找到敏感词  //就在网页上显示"然哥说：消息内容"  document.write(`<h1 style="color:green">然哥说:${msg}</h1>`)  }  </script>  运行效果： |

e. 问题: 默认js中的正则表达式只要找到第一个符合要求的敏感词就下班了！不再继续往后找！

f. 解决: 在正则表达式第二个/后加g，意为global，全部

g. 新问题: match()一旦加上g之后，虽然可以返回所有关键词的内容，但是无法返回每个关键词的位置了！

h. 示例: 查找一个字符串中所有关键词的内容

|  |
| --- |
| <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！";  //用正则表达式找出str中\*所有\*以小字开头的人名  var arr=str.match(/小[\u4e00-\u9fa5]/g);  console.log(arr);  </script>  运行效果： |

(4). 既查找每个敏感词的内容，又查找每个敏感词的位置:

String家的函数，无法实现如此复杂的查找功能。

只能依靠reg.exec()

|  |
| --- |
| 总结:函数返回值有什么规律:  1. 如果原函数返回下标位置i，则如果找不到返回: -1  2. 如果原函数返回一个数组或一个对象，则如果找不到返回: null |

2. 替换关键词: 2种:

(1). 简单替换: 将所有找到的敏感词都替换为统一的新值

a. str=str.replace(/正则/ig, "新值");

b. 意为: 将str字符串中所有符合正则要求的关键词都替换为统一的新值！

c. 强调: 要向获得替换后的新字符串，必须用=接住replace的返回值新字符串才行

d. 示例: 将字符串中所有关键词都替换为统一的新值

|  |
| --- |
| <script>  //如果找到关键词  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！";  //如果找不到关键词:  // var str="you can you up,no can no bibi";  //用正则表达式找出str中\*所有\*以小字开头的人名  var arr = str.match(/小[\u4e00-\u9fa5]/g);  //arr=null  console.log(arr);  //希望将所有以小字开头的人名都替换为\*\*  str = str.replace(/小[\u4e00-\u9fa5]/g, "\*\*");  //不报错，而是返回原字符串，什么也不干！  console.log(str);  //如果match找到了敏感词  if (arr != null) {  //才输出共替换了几处？  console.log(`共替换了${arr.length}处`);  // null.length 报错: 无法读取null的length属性  } else {//否则如果match没找到敏感词  //就输出共替换0处  console.log(`共替换了0处`)  }  </script>  运行结果： |

(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. str=str.replace(/正则/ig, function(keyword){

return 根据keyword收到的关键词不同，动态返回不同的新值替换到字符串中

})

b. 意为: 将str中每个符合正则要求的敏感词都交给一个回调函数中去判断应该替换为何种新值。

c. 原理:

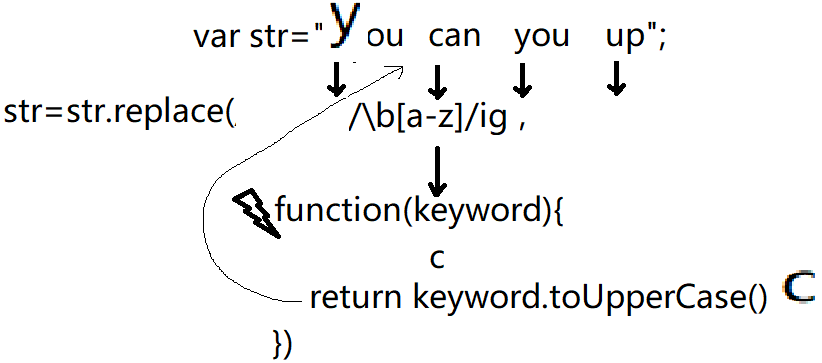
1). replace会用正则查找str中每个关键词

2). replace每找到一个关键词，就自动调用一次回调函数，并自动将本次找到的关键词传递给回调函数的keyword形参变量。

3). 回调函数内部，根据每次传入的关键词不同，动态选择不同的新值返回给replace

4). replace会将回调函数返回的新值，替换进字符串中本次找到的关键词位置。

5). 最后replace会将替换后的新字符串返回，并保存在=左边的变量中



d. 示例: 替换一句话中每个单词首字母为大写

|  |
| --- |
| <script>  //鄙视题: 将每个单词首字母改为大写  var str="you can you up";  //每找到一个敏感词就自动调用一次回调函数  str=str.replace(/\b[a-z]/ig,function(keyword){  //打桩: 输出回调函数调用次数、关键变量的值、函数的返回值  console.log(`自动调用了一次回调函数,keyword收到${keyword},返回新值:${keyword.toUpperCase()}`);  //keyword依次接到: y, c, y, u  return keyword.toUpperCase()  //return 依次返回: Y, C, Y, U  })//replace会将回调函数返回的值替换到字符串中关键词的位置上。  console.log(str);  </script>  运行效果:  自动调用了一次回调函数,keyword收到y,返回新值:Y  自动调用了一次回调函数,keyword收到c,返回新值:C  自动调用了一次回调函数,keyword收到y,返回新值:Y  自动调用了一次回调函数,keyword收到u,返回新值:U  You Can You Up |

(3). 衍生操作：删除关键词，将关键词替换为""，就算删除了

a. str=str.replace(/正则/ig, "")

b. 示例: 去掉字符串开头和结尾的空字符

|  |
| --- |
| <script>  //鄙视: 定义函数去掉字符串开头的和结尾的空字符  // 修剪 字符串开头的空字符  function trimLeft(str){  return str.replace(/^\s+/,"");  }  // 修剪 字符串结尾的空字符  function trimRight(str){  return str.replace(/\s+$/,"");  }  // 修剪 字符串开头和结尾的空字符  function trim(str){  return str.replace(/^\s+|\s+$/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/ES6: 内置了,不用自定义可直接使用！  console.log(str.trimLeft());//"zhang dong "  console.log(str.trimRight());//" zhang dong"  console.log(str.trim());//"zhang dong"  </script>  运行结果:  zhang dong  zhang dong  zhang dong  zhang dong  zhang dong  zhang dong |

3. 切割字符串: 将一个字符串，切割为多段子字符串，2种:

(1). 简单切割: 切割符是固定不变的

a. var arr=str.split("切割符")

b. 意为: 将字符串str，按指定的切割符，切割成多段子字符串，保存在一个数组中返回

c. 强调: 切割后的结果中，不包含切割符的！

(2). 复杂切割: 切割符不是固定不变的，但是可以找到规律

a. var arr=str.split(/正则/i)

b. 意为: 将字符串str，按正则匹配到的切割符，切割成多段子字符串，保存在一个数组中返回

c. 强调:

1). 切割后的结果中，不包含切割符的！

2). 不用加g，split也能找到所有符合条件的切割符！

(3). 固定套路: 打散一个字符串为字符数组:

var arr=str.split("")

(4). 示例: 按规定切割字符串

|  |
| --- |
| <script>  var email="zhangdong@tedu.cn";  // 用户名 | 域名  //希望获得电子邮件中的用户名和域名  //按@切割电子邮件  var arr=email.split("@");  console.log(arr);  var uname=arr[0];  var domain=arr[1];  var str="you can you up";  //获得一个英文句子中单词的列表  //按空格切割字符串str  var words=str.split(" ");  console.log(words);  var str="you can you up";  //按一个以上的连续空格来切割  var words=str.split(/\s+/);  console.log(words);  //回顾:  var arr=[1,2,3];  arr.reverse();  console.log(arr);  //鄙视: 翻转一个字符串:  var str="hello";  //str.reverse();//报错: str.reverse不是一个函数  //先把str打散为字符数组  var arr=str.split("");  console.log(arr);  //调用数组家的函数翻转数组  arr.reverse();  //要将翻转后的字符数组再拼接回字符串  //str=arr.toString();//错误: toString()只能用逗号连接每个字  str=arr.join("");//无缝拼接  console.log(str);//olleh  </script>  运行结果：  ["you", "can", "you", "up"]  ["you", "can", "you", "up"]  [3, 2, 1]  ["h", "e", "l", "l", "o"]  olleh |

二. RegExp对象:

1. 什么是: 专门在内存中保存一条正则表达式，并提供用正则表达式执行验证和查找功能的函数 的对象

2. 为什么: 因为正则表达式是一种独立于编程语言之外的全新的语法，js语言不认识正则表达式，就需要一种专门的对象能将正则表达式翻译为js能理解的方式。让js语言也能使用正则表达式了。

3. 何时: 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种方式:

(1). 标准写法: var reg=new RegExp("正则表达式", "ig")

a. 强调: ""中不要再写//了！

b. 何时: 少数情况下，如果正则表达式需要动态拼接生成，就被迫选择new RegExp()方式。

c. 因为: js中字符串可以随意拼接！

d. 示例: 根据数组内容，动态生成正则表达式

|  |
| --- |
| <script>  //假设每次都要从服务器端动态获取了一个新的包含关键词的数组  //希望，数组中有什么关键词，咱们就要防住什么关键词  var arr=["青天","明月","紫烟"];//服务器上关键词列表可能发生变化  console.log(arr.join("|"));  //目的: 也不需要修改下方的js程序，就可屏蔽新的关键词列表。    //请用户输入一条消息  var msg=prompt("请输入消息内容");  //定义正则表达式防住arr数组中包含的关键词  //var reg=/arr.join("|")/; 错误://之间不认识js语句  var reg=new RegExp(arr.join("|"));  // "明月|青天|紫烟"  // new RegExp( "正则" )  console.log(reg);  //用正则表达式查找msg中是否包含关键词  var i=msg.search(reg);  //如果msg中包含关键词  if(i!=-1){  //则在网页中显示: 包含关键词禁止发送  document.write(`<h1 style="color:red">发现敏感词，禁止发送！</h1>`)  }else{//否则如果msg中不包含关键词  //则在网页中显示: 消息内容  document.write(`<h1 style="color:green">然哥说: ${msg}</h1>`)  }  </script>  运行效果: |

(2). 简写: var reg=/正则/ig

a. 何时: 绝大多数情况下，如果正则表达式是固定不变的，比如:手机号规则、电子邮件规则等，首选//创建。

b. 因为: //中是正则表达式的地盘，不认识js，不支持动态拼接字符串的语句。

5. RegExp提供的函数: 2个

(1). 验证字符串格式是否符合正则表达式要求:

a. var bool=reg.test(str)

b. 意为: 用这规则表达式reg去检查字符串str是否符合正则表达式格式要求

c. 何时: 今后只要网页中格式的验证，都用reg.test()

比如: 验证手机号格式、身份证号格式、电子邮件格式等

d. 返回值:

1). 如果字符串str的格式符合正则表达式reg的要求，则返回true

2). 如果字符串str的格式不符合正则表达式reg的要求，则返回false

e. 问题: 默认只要str中包含符合reg要求的部分内容，则test就返回true

比如: var reg=/\d{6}/

reg.test("abc123456") 返回true

reg.test("123456abc") 返回true

f. 解决: 今后凡是验证格式的正则表达式必须同时前加^后加$

表示，从头到尾必须和规则完全匹配才行！

比如: var reg=/^\d{6}$/ 意为从头到尾中间只能有6位数字！

reg.test("abc123456") 返回false

reg.test("123456abc") 返回false

g. 示例: 验证密码格式:

|  |
| --- |
| <script>  //请用户输入一个密码:  var pwd=prompt("请输入密码");  //定义一个正则表达式描述密码的格式: 6位数字  var reg=/^\d{6}$/; //从头到尾只能是6位数字  //用reg检查用户输入的密码pwd是否符合格式要求:  var bool=reg.test(pwd);  //如果检查通过  if(bool==true){  //就在页面输出: 密码可用  document.write(`<h1 style="color:green">密码可用</h1>`)  }else{//否则如果检查未通过  //就在页面输出: 密码格式不正确  document.write(`<h1 style="color:red">密码格式不正确</h1>`)  }  </script>  运行结果: |

(2). 既查找每个敏感词的内容，又查找每个敏感词的位置:

a. var arr=reg.exec(str);

b. 意为: 在str中查找下一个符合reg要求的关键词的内容和位置

c. 返回值: 每次查找的返回值和match不加g的情况完全一样

1). 如果找到关键词，就返回一个数组

arr : [ "0":"本次找到的关键词", ... , "index":"本次找到的关键词的下标位置" ]

2). 如果找不到关键词，就返回null

d. reg.exec() vs str.match()

1). str.match()不加g时，永远只能返回第一个关键词的内容和位置。无法自动找下一个关键词

2). reg.exec()在加g时，每找到一个关键词，再次调用reg.exec()时，可以自动继续找下一个关键词！直到返回null，说明找完了.

e. 固定套路: 循环+reg.exec()可找出一句话中所有关键词的内容和位置

//不管有没有关键词，都要先查找一次！才知道有没有关键词，有没有必要继续

var reg=/正则/g;

do{

var arr=reg.exec(str);

if(arr!=null){//如果本次找到了关键词，才有必要输出关键词的内容和位置

//console.log(arr);

console.log(`在位置${arr.index}发现敏感词${arr[0]}`)

}//否则如果没找到关键词，就什么也不输出。

}while(arr!=null);//如果arr!=null，说明本次找到关键词了，才有必要继续找下一个

f. 强调: 想让reg.exec()找所有，必须加g

g. 示例: 查找一句话中所有关键词的内容和位置

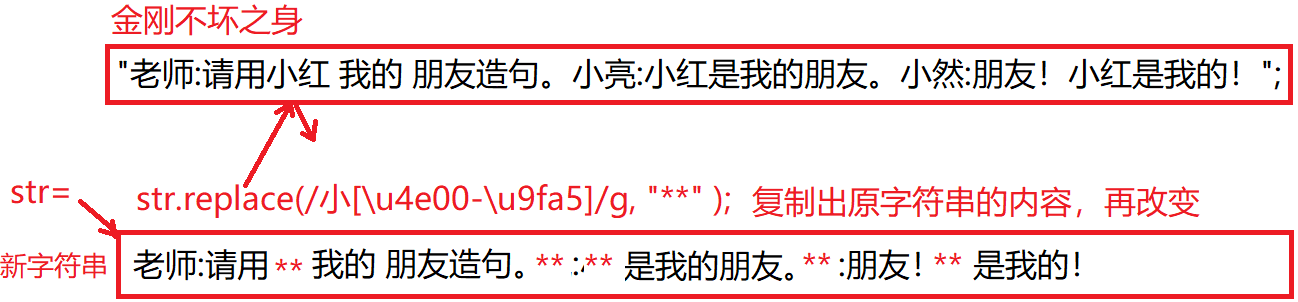
|  |
| --- |
| <script>  var str = "老师:请用小红 我的 朋友造句。小亮:小红是我的朋友。小然:朋友！小红是我的！";  //定义正则描述小字开头的人名    //用正则表达式找出str中\*所有\*以小字开头的人名和位置  var reg=/小[\u4e00-\u9fa5]/;  //用match()做: 无法实现找所有的要求  var arr = str.match(reg);  console.log(arr);//返回第一个人名和位置  var arr = str.match(reg);  console.log(arr);//只能返回第一个人名和位置  var reg=/小[\u4e00-\u9fa5]/g;  //do while  //不管有没有关键词，都要先查找一次！才知道有没有关键词，有没有必要继续  do{  var arr=reg.exec(str);  if(arr!=null){//如果本次找到了关键词，才有必要输出关键词的内容和位置  //console.log(arr);  console.log(`在位置${arr.index}发现敏感词${arr[0]}`)  }//否则如果没找到关键词，就什么也不输出。  }while(arr!=null);//如果arr!=null，说明本次找到关键词了，才有必要继续找下一个  </script>  运行效果:  ["小红", index: 5...]  ["小红", index: 5...]  在位置5发现敏感词小红  在位置16发现敏感词小亮  在位置19发现敏感词小红  在位置27发现敏感词小然  在位置33发现敏感词小红 |

补: 字符串是不可变类型:

1. 字符串一旦创建，内容不可改变！只能整体替换！

2. 所以字符串家的函数，都无权直接修改原字符串，只能返回加工后的新字符串。原字符串始终保持不变！

3. 所以要想获得字符串家函数返回的处理结果，必须用=接住返回值！否则，处理结果没人要！就自动扔了！



补: 数组本质: js中根本就没有索引数组。js中一切数组底层都是关联数组。所有数组元素的下标都是字符串类型！

1. 其实，访问所有数组元素的标准写法，都可写成: 数组名["下标名"]

比如: var arr=[1,2,3]; arr["0"] 访问数组中"0"位置的元素

再比如: var ym=[]; ym["math"]=89; ym["math"]访问数组中"math"位置的元素

2. 简写: 自动翻译

1). 自定义名称的下标，可简写为: 数组名.下标名

比如: ym["math"] 可简写为 ym.math 会被自动翻译回 ym["math"]

2). 数字下标，~~不能简写为.下标名~~，因为和小数的写法冲突。应该只去掉""，保留[]。数组名[下标]

比如: arr.1 —— 错误, 和小数点冲突

arr[1] —— 正确

总结:

1. 正则表达式:

(1). 最简单的规则就是要找的关键词正文

(2). 如果规则中一位字符上有多种备选字时: [备选字]

(3). 如果[]中部分备选字符是连续的: [x-x]

a. [a-z] 一位小写字母

b. [A-Z] 一位大写字母

c. [A-Za-z] 一位字母，不区分大小写

d. [0-9A-Za-z] 一位字母或数字，不区分大小写

e. [\u4e00-\u9fa5] 一位汉字

(4). 预定义字符集: 针对4种最常用[]的情况提供的更简化写法

a. \d 一位数字 [0-9]

b. \w 一位字母数字或\_都行 [0-9A-Za-z\_]

c. \s 一位空字符 空格、Tab、换行等

d. 通配符 .

(5). 如果规定一个字符集出现的次数，可以用量词。

a. 有明确数量边界的量词

1). {n} 必须n个，不能多也不能少

2). {n,m} 至少n个，最多m个

n<= <=m

3). {n,} 至少n个，多了不限

n<=

b. 没有明确数量边界的量词

1). \* 可有可无，多了不限 >=0

2). ? 可有可无，最多一个 0或1

3). + 至少一个，多了不限 >=1

(6). 如果两个规则，二选一匹配即可: 规则1 | 规则2

(7). 如果希望多个子规则作为一个整体先匹配，再和其它规则联合匹配: (子规则)

(8). 如果想匹配特殊位置的关键词:

a. ^规则 匹配紧挨着开头的一个关键词

b. 规则$ 匹配紧挨着结尾的一个关键词

c. \b规则 匹配紧挨着单词边界之后的一个关键词

d. 规则\b 匹配紧挨着单词边界之前的一个关键词

2. String正则相关的函数: 3件事

(1). 查找关键词: 4种情况

a. 查找一个固定的关键词出现的位置:

var i=str.indexOf("关键词", fromi)

b. 用正则模糊查找多种敏感词的位置:

var i=str.search(/正则/i)

c. 查找关键词的内容:

1). 如果只查找一个关键词的内容和位置：

var arr=str.match(/正则/i)

2). 如果查找所有关键词的内容，不关心位置:

var arr=str.match(/正则/ig)

d. 既查找每个关键词的内容，又查找每个关键词的位置:

reg.exec()

(2). 替换关键词: 2种情况:

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

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

return 根据keyword不同，动态选择不同的新值返回并替换到字符串

})

c. 删除关键词: str=str.replace(/正则/ig, "")

(3). 切割字符串: 2种情况:

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. 固定套路: var arr=str.split("")//将字符串打散为字符数组

3. RegExp对象:

(1). 创建:

a. var reg=new RegExp("正则","ig")

b. var reg=/正则/ig

(2). 验证:

var reg=/^正则$/;

var bool=reg.test(str)

(3). 既查找每个敏感词内容，又查找每个敏感词位置:

var reg=/正则/g

do{

var arr=reg.exec(str);

...

}while(arr!=null);

今日相关视频列表: 小程序->首页->JSCORE->day02

1. 3. 4. 5. 8. 10

作业:

1. 继续看小程序视频学习回调函数原理:

小程序->在线->JSCORE->DAY02 3. 回调函数 callback 亮哥 带饭

2. 复习今日问题清单(小程序->首页->JSCORE->day02)，预习明日问题清单(小程序->首页->JSCORE->day03)

3. 看小程序视频复习第一阶段: 函数、声明提前、作用域和按值传递

小程序->在线->JSCORE->day03

1. 回顾 定义函数...

2. 声明提前

6. 简单 作用域 按值传递

4. 复习第一阶段: 三目运算、短路逻辑