正课:

1. \*\*\*String

替换

切割

2. \*\*\*RegExp

3. Math

4. \*\*\*Date

1. \*\*\*String

替换: 将找到的敏感词，替换为规定的新内容

如何: 2种:

1. 简单替换: 将所有敏感词，都替换为统一的新内容

str=str.replace(/正则/ig,"新值");

强调: 1. 必须加g, 替换所有

2. 必须用=保存回去，否则无法获得替换结果

2. 高级替换: 根据本次找到的敏感词不同，动态选择不同的新内容替换。

str=str.replace(/正则/ig,function(kw){

kw //自动获得本次找到的关键词内容

return //根据kw的不同选择替换不同的值

})

其实: 比较器函数，也可用匿名函数:

arr.sort(function(a,b){return a-b;})

衍生: 删除关键词: 其实就是将关键词替换为""

切割: 将字符串按指定字符切割为多段子字符串

如何:

1. 简单切割: 按一个固定的字符/字符串切割

var subs=str.split("切割符")

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

将来如果希望对字符串，使用数组API时，都要先打散。

2. 复杂切割: 切割符不是固定的，但有一定规律

var subs=str.split(/正则表达式/i);

2. 正则表达式对象RegExp:

什么是: 封装一条正则表达式，并提供用正则表达式执行查找和验证操作的API

何时: 2种:

1. 用正则表达式验证字符串格式

2. 即查找内容，又查找位置时

如何:

创建: 2种:

1. 创建一个固定不变的正则表达式:

var reg=/正则表达式/ig;

字符冲突: 如果正则中出现/，要转为\/

2. 如果需要动态生成正则表达式:

var reg=new RegExp("正则"[,"ig"]);

字符冲突: 如果正则中出现引号，转为\"

正则中正常的\ 都要转为\\

API:

1. 验证: var bool=reg.test(str);

用reg验证str是否符合规则的要求

返回bool类型

问题: 正则默认只要找到部分匹配就返回true

解决: 要求从头到尾必须完整匹配

今后，凡是验证，必须同时前加^后加$

2. 查找: 即找位置，又找内容

var arr=reg.exec(str);

在str中找下一个满足reg要求的关键词的内容和位置

返回值: 本次找到的关键词的内容和位置

数组: 0号房间保存本次找到的关键词内容

index的房间保存本次找到的关键词位置

如果找不到，返回null

强调: 反复调用，会自动向后查找下一个！

3. Math:

什么是: 专门封装数学计算的常量和API的对象

何时: 今后只要进行数学计算时，都可用Math提供的数学API

如何: 不能new，所有API都直接用.调用!

包括:

取整:

上取整: 只要超过，就取下一个整数

Math.ceil(num)

下取整: 舍弃小数部分

Math.floor(num)

vs parseInt(str): 1. parseInt可去掉结尾非数字字符

floor不行！

2. parseInt要求传入字符串

floor要求数字

四舍五入取整: Math.round(num)

vs .toFixed(d): 1. round只能取整

toFixed 可按任意小数位数四舍五入

2. round返回number，可直接用于计算

toFixed 返回string，不能直接用于+计算

自定义round:

乘方和开平方:

乘方: Math.pow(底数,幂); 比如: Math.pow(2,3)=8

开平方: Math.sqrt(num)

最大值和最小值:

Math.max(值1,值2,...)

Math.min(值1,值2,...)

问题: max和min不支持查找数组中的最大值/最小值

解决: Math.max.apply(null,arr)

apply可打散数组类型的参数为单个值，再传递给调用国的函数

随机数:

1. Math.random(): 单独使用返回0~1之间的一个小数

2. 在min~max之间取随机整数:

parseInt(Math.random()\*(max-min+1)+min)

3. 如果从0~max之间取随机整数:

parseInt(Math.random()\*(max +1))

4. \*\*\*Date

什么是: 封装一个时间点, 提供操作日期和时间的API

何时: 今后只要存储或操作日期和时间

如何:

创建: 4种:

1. 创建日期对象，并自动获得客户端当前系统时间:

var now=new Date();

2. 创建日期对象，并自定义时间点:

var date=new Date("yyyy/MM/dd hh:mm:ss");

new Date(yyyy,MM-1,dd,hh,mm,ss);

3. 复制一个日期对象:

为什么: js中日期计算会直接修改原日期对象，计算前的时间无法保留

何时: 只要希望同时保留计算前后的开始和结束时间时，都要先复制日期的副本，再用副本计算新日期。

如何: var date2=new Date(date1);

4. 将毫秒数转化为日期对象:

为什么: 在跨国系统中，都是用毫秒数代表一个时间点

其实，日期对象中，保存的也是毫秒数:

代表1970年1月1日0点至今的毫秒数

毫秒数不受时区的影响

问题: 毫秒数无法直接看懂和使用

解决: 获得毫秒数后，都要用new Date转化为当地时区对应的时间显示。

如何: var date=new Date(ms);

API: 三句话:

1. 单位: FullYear Month Date Day 没有s结尾

Hours Minutes Seconds Milliseconds 都有s结尾

2. 每个单位上都有一对儿get和set函数

其中:get 负责获得该单位的数值

date.getDate() 获得日期

date.getDay() 获得星期

date.getHours() 获得几点

set 负责修改该单位的数值

date.setDate(10) 改为10号

date.setHours(19) 改为19点

特例: Day，没有set，不可修改！

3. 取值范围:

Month: 0~11 比现实中都小1

Date: 1~31

Day: 0~6

Hours: 0~23

Minutes/Seconds: 0~59

日期计算: 2种:

1. 两日期对象可直接相减: 结果是毫秒差

2. 对指定单位的值做加减: 3步:

1. 取值: var n=date.getXXX();

2. 计算: n+/-值

3. 放回去: date.setXXX(n);

福利: set可自动调整进制

其实可简化为一步: date.setXXX(date.getXXX()+/-n)

问题: setXXX直接修改原日期对象

无法保留计算前的原日期

解决: 如果希望同时保留计算前后的新旧时间，都要先复制副本，再用副本计算。

日期输出:

1. date.toString() 转化为当地时间的完整格式

2. date.toLocaleString() 转为当地时间的简化版格式

3. date.toLocaleDateString()转为当地时间的简化版格式，仅保留日期部分

4. date.toLocaleTimeString()转为当地时间的简化版格式，仅保留时间部分

5. date.toGMTString() 转为0时区标准时间