正课:

1. RegExp

2. Math

3. Date

1. RegExp

什么是: 专门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的API的对象

何时: 只要用正则表达式执行验证和查找时，都必须先创建正则表达式对象，来保存正则表达式

如何:

创建: 2种:

1. 用/创建: var reg=/正则/ig。

何时: 如果正则表达式是固定不变的，不需要动态生成。

问题: //中间不支持js语句

2. 用new创建: var reg=new RegExp("正则","ig");

何时: 如果正则表达式需要动态拼接而成，就只能用new！

强调: 字符冲突:

//内，不能再出现/，如果出现必须改为\/

""内，不能出现\，如果出现必须用\\

比如: "\d{6}" 应改为 "\\d{6}"

API: 2个:

1. 查找: 即查找每个关键词的内容，又能获得每个关键词的位置！

var arr=reg.exec(str)

在str中查找下一个符合reg要求的关键词的内容和位置，保存在数组中返回。

返回值: 包含本次找到的关键词的内容和位置的数组:{ 0: "本次找到的关键词", index: 关键词位置 }

如果没找到，返回null

2. 验证: var bool=reg.test(str)

用reg验证str的格式是否符合规则

问题: 默认，正则表达式只要部分匹配关键词，就返回true。

解决: 要求从头到尾必须完整匹配

强调: 今后，凡是验证，必须同时前加^后加$

2. Math:

什么是: 专门保存数学计算的常量并提供数学计算的API

何时: 只要进行数学计算时

如何:

创建: 不用创建！更不能new！

API: 所有API都用大写Math直接调用！

取整:

1. 上取整: 只要超过，就取下一个整数

Math.ceil(num);

2. 下取整: 只要超过，就去掉小数部分

Math.floor(num);

vs parseInt(str)——首选

1. parseInt可将字符串自动转为数字

2. parseInt可去掉结尾的非数字字符

3. 四舍五入取整: 超过部分够5进1，不够，舍弃。

Math.round(num);

vs n.toFixed(d)

1. 小数位数:

Math.round(num) 无法指定小数位数，只能取整

n.toFixed(d) 可指定小数位数，按任意小数位数四舍五入

2. 返回值:

Math.round(num) 返回number类型可直接参与任何算术计算

n.toFixed(d) 返回字符串，不能直接参数加法计算。

自定义round函数:

乘方和开平方:

1. 乘方: Math.pow(底数,幂)

2. 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: 不支持数组类型参数!

解决: Math.max(...arr)

... : 拆, 将数组拆散为单个元素，再传入函数

强调: 只能用在函数传参时

随机数:

基本用法: 0~1

Math.random()

在任意min~max之间取一个随机整数:

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

简写: 在0~max之间取一个随机整数:

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

3. Date:

什么是: 保存一个时间值

提供操作时间和日期的API

何时: 今后只要在程序中存储和操作日期或时间都必须用Date

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();

何时: 只要获得当前时间，都必须用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");

3. 将一个毫秒数转化为日期对象:

问题: 用文字表示时间

1. 受时区和时差的影响，产生偏差。

2. 不便于左加减计算

解决: 日期对象中实际存储的是1970年1月1日至今的毫秒数——整数

如何获得: var ms=date.getTime();

优点:

1. 毫秒数是时间段，全球统一，不受时区的影响。

2. 便于算术计算

总结: 今后存储日期和时间必须用毫秒数，不能用文字.

何时: 今后只要将ms转化为日期对象时，都用new Date();

4. 基于现有日期，复制一个一模一样的新日期对象:

为什么: 因为日期计算直接修改原日期对象，计算前的日期无法保留

何时: 如果用户希望同时保留计算前后得开始和结束时间时，都要先复制日期对象副本，再用副本计算。

如何: var date2=new Date(date1);

API: 3件事:

1. 单位: 8个:

FullYear Month Date Day

Hours Minutes Seconds Milliseconds

2. 每个单位都有一对儿get/set函数:

其中:

date.getXXX() 负责获得日期中指定单位的数值

比如: date.getHours();

date.getDate();

date.getFullYear();

date.setXXX(n) 负责修改日期中指定单位的数值

比如: 修改日期中的时间为25号

date.setDate(25)

特例: Day没有set函数

3. 取值/返回值的范围:

Month: 0~11 都比现实中小1

取值时，+1修正为现实的月份

修改时，-1修正为计算机中的月份

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: 结果为毫秒差

2. 对日期的任意一个单位做加减:

3步:

1. 取值: var d=tick.getDate();

2. 计算: d-=30;

3. 放回去: tick.setDate(d);

简写: tick.setDate(tick.getDate()-30);

总结: date.setXXX(date.getXXX()+/-n)

问题: date.setXXX()直接修改原日期对象

无法保留计算前的原日期对象

解决: 如果用户希望同时保留计算前后的开始和结束时间时，必须先复制一个日期对象的副本。再用副本计算目标时间。

日期格式化:

date.toString() 转为当地时间的完整格式

date.toLocaleString() 转为当地时间的简版格式

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

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

date.toGMTString() 转为0时区标准时间