**Javascript的笔记（只有函数才有作用域）**

**Day01**

1. **js语法(程序默认都是自上而下逐行执行的)**

（一）js书写

（1）script[src 引入js文件的路径][type=“text/javascript”文件类型 可省略]

\* **在拥有src属性的script标签写js代码，无效**；

\* head>script ,body>script

（2）a[href=“javascript:js 代码 ;”]

（二）js注释

// 单行注释

/\* 多行注释，不要嵌套多行注释 \*/

（三）声明变量及赋值

1.声明变量,通过关键字var

var cup;

2.给变量赋值,通过 **=** ,将右边的值赋给左边的变量

cup = "orange";

3. 同时声明变量及赋值

var fruit = "lemon";

4. 同时声明多个变量,用逗号隔开

1、var a,b,c;

a = 10;

b = 20;

c = 30;

2、var a = 10,b = 20,c = 30;

**【\*\*\*\*声明提前：在正式执行脚本之前，将所有var声明的变量，提前预读到（当前作用域的）顶部，集中声明创建然后再开始执行程序。\*\*\*赋值留在本地。】**

（四）变量名的命名规则：

\* 不能以数字开头,包含字母、数字、\_、$

\* 不能使用关键字——关键字：ES标准中已经使用的，有特殊意义的词



\* 严格区分大小写

\* 建议：

(1) 见名知意

(2) 驼峰命名 myUserName

**二、数据类型（值）**

（一）基本数据类型

1. 数字类型number

\* 普通数字

\* NaN：代表非数字

\* 与任何数进行运算结果都是NaN

\* NaN不等于，不大于，不小于任何值，甚至不等于自己

\* isNaN()判断是不是非数字，是数字就返回false，其他值都返回true

2. 字符串类型 string

\* 有 **引号（单双引号都行）** 的值都是字符串类型

\* 字符串中还存在引号？

\* 将外层引号替换成另外一种

\* 通过转义字符 \

3. 布尔类型 boolean

\* 只有两个值：true 、false

（二）特殊数据类型

1、null是空对象，只有唯一的值null。

\* 注意代码的执行顺序：程序默认都是自上而下逐行执行的；

\* 注意报错信息：Cannot set property '\*\*\*' of null。不能设置属性\*\*\*给空对象

2、undefined 代表的是声明变量未赋值，

\* 注意区分报错信息：\*\* is not defined。代表的是变量未声明。

1. **引用数据类型（重点）**

两种：Object、Array数组

**\* 基本数据类型传值时，传递的是值**

**\* 引用数据类型传值时，传递的是地址**

**三、输出**

1、alert(具体的值||变量(不要加引号)) 弹窗；

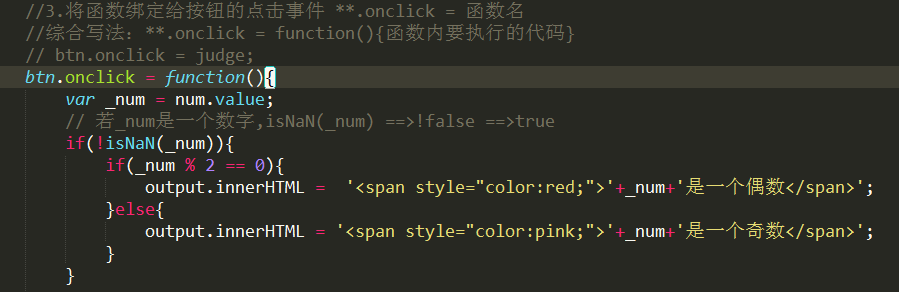
2、document.write(具体的值||变量(不要加引号)) 往body里面追加内容；

3、console.log(具体的值||变量(不要加引号)) 打印到控制台console.log，一般用于**代码调试；**

4、\*\*.innerHTML=“ ”(在什么里面，必须是双标签，才能用这个)；给\*\*（HTML双标记）元素添加内容，内容可以**包含标签**；

1. 关于onclick点击事件的写法，谁需要就给谁添加；

写法如下：



**四、运算**

（一）算术运算 + - \* / %取余

备注： + 号两侧只要有一侧为字符串，就代表字符串拼接

备注： 隐式转换: 若运算无法进行下去的时候，会尝试将数据类型进行隐式转换后，再运算

备注：字符串**中间拼接变量**：口诀：引号引号（最外层），加号加号；

（二）赋值运算 -=，\*=，/=，%=、++、--

（1）= 将右边的值赋值给左边的变量（不是等于的意思！！！）

（2）+= 将右边的值追加给左边的变量，eg:a+=2==>表示a=a+2

（3）一元运算

++ 对变量进行加1运算（**都会对变量进行加1运算，只是返回的值会有所不一样而已**）

1、++变量（++a）：先对变量进行加1运算，再将变量的新值返回出去

2、变量++（a++）：先将变量的值返回出去，再对变量进行加1运算

-- 对变量进行减1运算**（原理：同上的++）**

（三）关系运算（返回值是布尔值（Boolean），可以进行隐式转换）

（1）符号：>、<、>=、<=、==

（2）== 判断左右两边的值是否相等，类型不同先隐式转换成数字再比较。

（3）!= 判断左右两边的值是否不相等。隐式转换后不相等才返回true。

（4）=== （**判断值和数据类型是否都一致**）全等于，只有当**值及数据的类型都一致**才返回true。

（5）!== 不全等于，只有当值及数据的类型都是不一致的才返回true。

（6）关系运算符的比较规则：

A、数字和数字比较，直接比较大小；

B、数字和字符串比较，字符串转换为数字后再比较大小；

C、 字符串和字符串比较，进行字符串的ASCII码值比较；字符0-9==>ASCII码48-57 （四）逻辑运算

（1）&& 与运算。运算两边返回值都为true，才返回true

1、与的短路逻辑运算（重要的运算）：**短路逻辑，简化代码**

\* &&运算左边返回值为false，不再执行右边的代码

\* &&运算左边返回值为true，右边代码直接返回对应的值

（2） || 或运算。运算两边返回值都为false，才返回false

1、或的短路逻辑运算（重要的运算）：

\* || 运算左边返回值为true，不再执行右边的代码

\* || 运算左边返回值为false，右边代码直接返回对应的值

（3）! 非运算，取反。

1. 运算符的优先级问题

( ) > 一元运算（++、--、!） > 算术运算符（+、-、\*、/） > 关系运算符（先比较大于小于，在比较等于） > 逻辑运算（&&>||） > 赋值运算（=） > ,

**五、数据类型的转换**

（一）直接转换

Number() 转换成数字类型

\* 字符串->数字： 若可以转换成数字，返回值就是数字。**若不能转成数字就是NaN**

\* 布尔值->数字： true->1,false->0

\* “”空字符串转为0

String() 转换成字符串类型

\* 直接加引号

Boolean() 转换成布尔类型的值

\* 除了0、NaN、""、null、undefined转成false，其他都转成true。

（二）隐式转换

若运算无法进行下去的时候，会尝试将数据类型进行隐式转换后，再运算。

\* 若隐式转换后依旧无法运算，会返回NaN

1. **常用的数学方法：**

（1）parseInt():取整；

从第一个字符开始依次读取每个数字只要碰上第一个非数字字符就停止，自动跳过开头的空字符

\*\*\*不认识小数点

（2）parseFloat():取浮点数；

用法同parseInt\*\*认识第一个小数点；何时使用：将字符串转为小数时

（3）Math.round() 四舍五入

（4）Math.random() 获取0-1的随机数，包含0，不包含1 ，简单写法[0,1)

（5）数字.toFixed(n位小数)，给该数字保留多少位小数；

**Day02**

1. **进制**

二进制：0-1

八进制：0-7

十进制：0-9

十六进制：0-9 a-f

进制之间的转换：

\* 将十进制转成其他进制：数字.toString(n进制)

\* 将多进制转成十进制：parseInt(“n进制的字符串”,n进制)

1. **判断数据类型 typeof()——注意typeof()判断出来的数据类型，都是带引号的，都是字符串**

typeof(123); //"number"

typeof(NaN); //"number"

typeof(""); //"string"

typeof("AFDF"); //"string"

typeof(true); //"boolean"

typeof(false); //"boolean"

typeof(null); //"object" ——null是一个空对象

typeof(undefined); //"undefined"

typeof(typeof(123)); //"string"

1. **选择判断语句，分支语句**

（一）if条件语句

1. 单分支 if（条件）{满足条件执行的代码段;}；
2. 双分支 if（条件）{满足条件执行的代码段;}else{不满足条件执行的代码段;}；
3. 三目运算：三元运算，条件运算（一般该运算用于双分支的情况）；

什么是：根据不同的条件，执行不同的操作/返回不同的值

语法：（如果语句比较简单，建议用三目运算代替if...else...）

条件1?值1|操作1：//如果满足条件1，就返回值1或执行操作1

条件2?值2|操作2：//否则，如果满足条件2，就返回值2或执行操作2

...?...：默认值/操作；//否则，（前边条件都不满足）//执行默认操作，或返回默认值

总结：多个条件，多件事，多选一执行。

备注：每个条件和操作或值为一组，组和组之间用冒号连接（每个（条件+输出）用冒号连接）

1. 多分支 if（条件1）{满足条件1执行的代码段;}else if（条件2）{满足条件2执行的代码段;}else{不满足条件执行的代码段;}；

**注意：对于能写清楚条件的，一定要将所有的条件写清楚，以免出错！**

1. switch case语句

何时使用：当条件都是\*全等\*比较时，才可用switch结构；

switch(变量){

case 值1 :

当满足值1时，执行这里的代码;

break;

case 值2 :

当满足值2时，执行这里的代码;

break;

...

default :

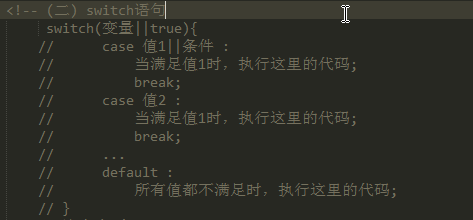
所有值都不满足时，执行这里的代码;

}

注意事项：

\* switch判断值相等时是利用**全等于（===）**，值及类型都必须一致

\* 每个case语句执行代码完毕后，记得break。

**switch case的另外一种用法：**

**Day03**

1. **循环语句(while，do...while，for）**

（一）\*\*\*循环结构（三种循环结构）：while、do while、for——（循环嵌套）

1、\*\*\*循环结构：在给定条件成立时，让程序\*反复\*执行同一段代码

\*\*\*循环三要素：

a、循环条件：让循环\*反复\*执行的条件

比如：圈数<3

b、循环变量：循环条件中用于比较的变量

比如：圈数

从几开始，（到几结束），每次变化几，循环变量总要向着不满足循环条件的趋势不断的变化

c、循环体：循环内，反复执行的代码段

2、三种循环结构：while、do while 、for

a、while：先判断循环条件，在执行循环体

何时使用：循环变量的变化，没有规律时

语法：var 循环变量=初始值;

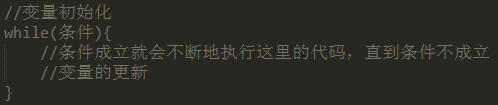
while（循环条件）{

//当...的时候

循环体；

迭代修改循环变量；

}



b、do...while：先执行循环体，再判断循环条件

何时使用：即使不满足循环条件，也希望至少执行一次时。

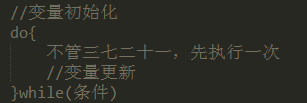
语法：var 循环变量=初始值；

do{

循环体；

迭代修改循环变量；

}while(条件);



**备注：while vs do while的区别：就看第一次条件是否满足、第一次条件满足，两者完全等效，第一次不满足条件，while一次都不执行、do while至少可以执行一次；**

c、for循环：for—while完全等效

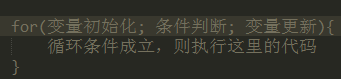
while循环语法：var 循环变量=初始值；

while(循环条件){

循环体;

迭代修改循环变量；

}



for循环语法：

for(var 循环变量=初始值;循环条件;迭代循环变量（变量更新）){

循环体；//满足循环条件执行的代码段

}

**【for(;;)等价于while(true)——>死循环】**

for的特殊用法：

1、声明部分：可同时声明并初始化多个变量，用逗号分隔

2、迭代修改循环变量部分：可同时执行多个小的操作，用逗号分隔

3、for(;;)取代while(true),实行死循环效果\*\*\*\*【for(;;){}==while(true){}】

d、循环语句中的关键字：**break（退出当前整个循环），continue（跳过本次循环，继续下次）**

1、break：//退出当前整个循环

\* 只能在循环语句中使用

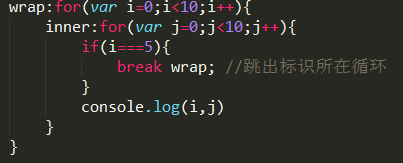
\* 循环体中位于break后的语句不会被执行

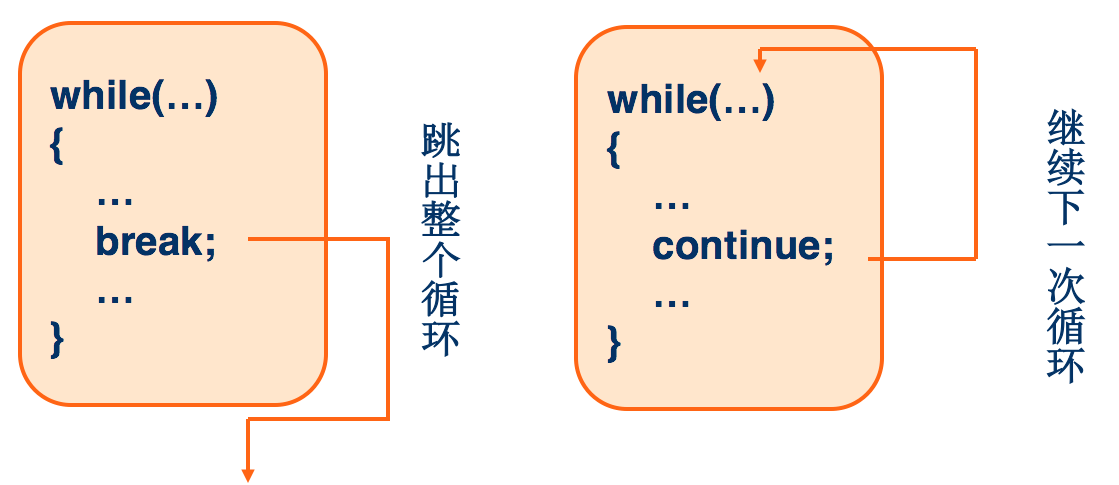
\* **在多层循环嵌套中，一个break语句只向外跳一层**

2、continue：//跳过本次循环，继续下一次循环

\* 只能在循环语句中使用，

\* 结束本次循环（**即跳过循环体中下面尚未执行的代码**），接着执行下次循环。

3、break和continue后如果带标识，则跳出标识所在循环



e、循环嵌套

**Day04**

1. **函数的定义及好处**

1、函数：封装一项专门任务的步骤清单的代码段，起一个名字，程序中提供的一项服务的步骤说明；

2、何时使用函数？当一项任务需要反复执行，但又不希望重复编写时，为了实现代码重用；

3、使用函数的好处:

\* 函数可以重复某一部分代码（通过函数名调用）

\* 使程序变得更简短而清晰

\* 有利于程序维护

1. **如何声明函数**

（1）function 关键字声明

function 函数名（形参）{调用函数时执行的代码段;}

（2）赋值式

var 函数名=**[function(形参){调用函数时执行的代码段;}]——匿名函数**

1. 构造函数

new Function();——这个在面向对象阶段用的比较多

**备注：函数在声明时，不执行！也不读取内部的代码！函数调用时，才读取内部的代码，逐行执行！**

1. 匿名函数：没有名字的函数就叫匿名函数

Eg：（function（形参）{}）（实参）

1. **调用函数**
2. 手动执行：函数名（）;
3. 事件驱动
4. [onclick=“函数名()”]
5. ele.onclick=函数名———**谨记：这里的函数名后不能加括号**
6. ele.onclick=function(){}
7. **函数的分类**
8. 自定义函数，声明函数
9. 匿名函数 function(){}
10. 内置函数

**五、声明提前**

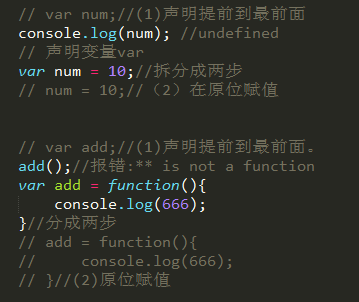
1. **在js代码执行之前，会将所有的声明提升到当前作用域的最顶端，值保留在原地；**
2. 按照代码的执行顺序，代码都是自上而下顺序执行的；

\* **备注：js代码一旦报错，下面的代码都将终止执行了**；

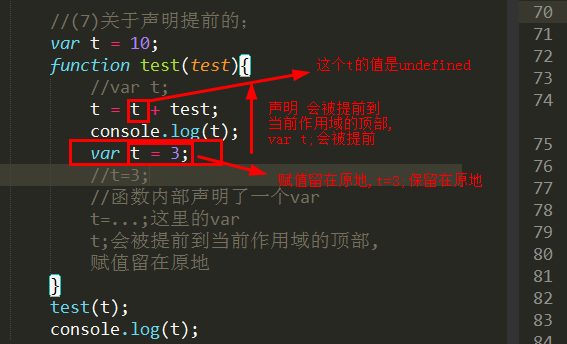
\* 若使用赋值式（var声明函数）声明函数，在函数赋值之前使用函数，会出现报错，\*\*is not a function。因为声明提前还没有赋值，所以值为undefined，undefined不是函数。

\* 若使用关键字function声明函数，能在函数声明代码的前面使用；

声明提前如图所示：



**【声明提前经典点例子：】**



**六、作用域（全局作用域和局部作用域）**

（1）概念：某个变量的使用范围，分成全局作用域及局部作用域；

（2）全局作用域以及局部作用域

1、全局变量：全局作用域下声明的变量（函数外声明的变量）；

2、局部变量：局部作用域下声明的变量（函数内声明的变量）；

（3）变量的访问规则（**就近原则**）

1、在函数内查找变量a，若自己内部已经声明该变量，那么使用函数内的该变量

2、若在函数内查找不到声明，往函数外一层查找。若找到则使用，找不到继续往外找

3、直到找到全局作用域，找到则使用，还找不到则报错，a is not defined

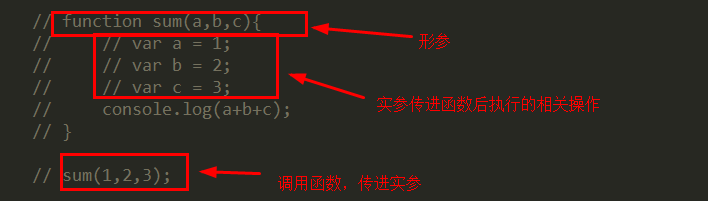
（4）**作用域链**

当函数访问变量时，根据就近原则从内到外查询变量，这个路径称为作用域链。

**七、函数的参数**

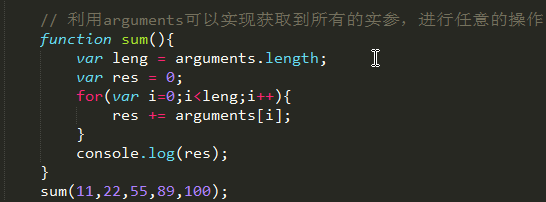
1. 形参：在函数声明时的参数（变量）；
2. 实参：在函数执行时执行的参数（值）；

\* 实参和形参的个数可以不一致。如图所示：



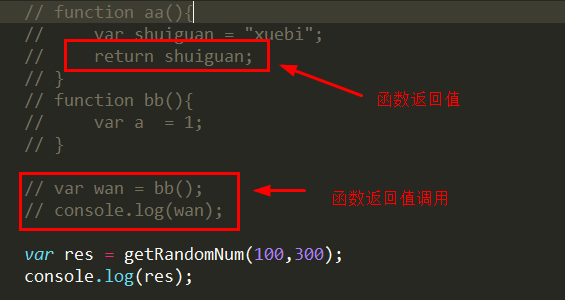
1. arguments 函数内部隐藏的对象，是一个类数组对象，包含着所有实参的信息；

如图：



**八、函数返回值return（返回）**

1. return将函数内部的值返回到函数外，函数外若想使用，必须要接收该返回值；
2. return若不书写或者直接写return，函数的返回值为**undefined**；
3. **return 还有退出当前函数，返回上一层函数的意思，当执行了return，函数后面的代码不再执行；**



（3）调用者何时如何获得返回值：

1、一个有返回值的函数调用，可以当做一个值使用

2、调用时，可使用变量保存住函数调用的返回值

**九、this**

**（1）指的是调用该函数的当前对象**

**1、函数手动执行时，this为window；普通函数的执行：this为window**

**2、事件驱动函数时，this为事件绑定的对象**

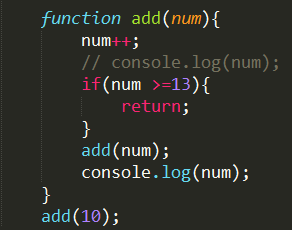
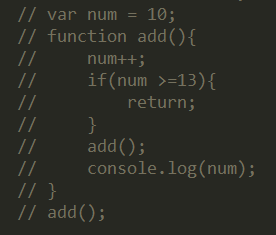
**3、对象执行自己本身的方法，this为当前对象**

**十、递归函数——难点，函数自己调用自己**

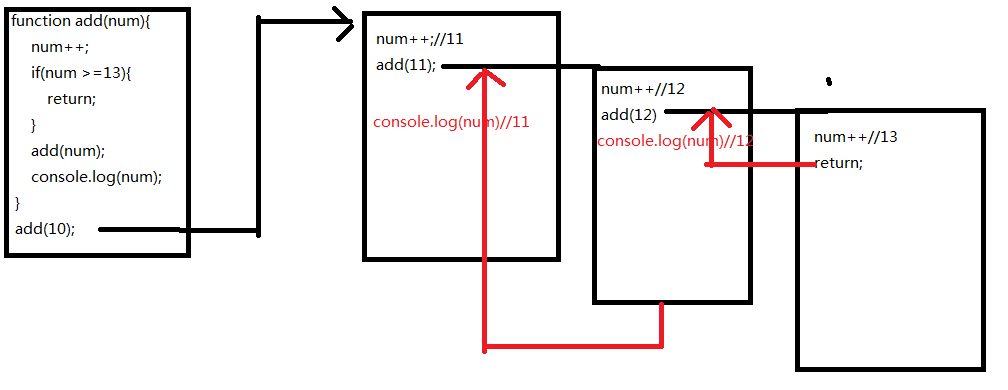
（1）自己调用自己，记得设置临界点，return退出函数。

\* 很容易进入死循环，报错 Maximum call stack size exceeded

Eg：如下图所示：



**详解：**



**十一、回调函数（将函数作为参数传递，重点）**

（1）函数a作为另外一个函数b的参数传递，在b内部执行函数a；

**Day05**

1. **数组：用于存储多个值 eg：arr=[值1,值2,值3]——数组的格式;**
2. **数组的概念：**一系列数据的集合，**每一项可以保存任何类型的数据**，称为数组的元素，每个元素之间用逗号隔开；

\*\*\*程序=数据结构+算法

算法：程序执行的步骤

数据结构：数据在内存中的存储结构

好的数据结构，可以极大提高程序的执行效率。

相关的一组数据，都要使用数组集中管理。

1. 如何使用数组：创建 取值 赋值
2. 创建：2种
3. **字面量创建 var arr=[1,2,3]，创建空数组 arr=[];——这种创建最常用，推荐**
4. 构造函数创建 new Array()——**这种用的少，一般不推荐使用**

\* 声明数组的同时赋值 new Array(1,2,3)

\* 声明数组同时定义数组的长度 new Array(10)

**（二）数组的操作**

1. 通过索引号操作数组arr里面的每个值。arr[索引]，索引（idx）取值从0开始到arr.length-1结束——**若是arr[索引]没有值，则返回undefined**
2. 通过length属性获取数组arr的长度：arr.length
3. 数组的遍历，利用for循环——for(var i=0;i<arr.length;i++)
4. **\*\*\*数组的方法（增删改）——关于数组的API，用.——这种操作会改变原数组**
5. 增删改，改变原数组。
6. .push（item）——往数组尾部添加一个或多个元素；
7. .unshift（item）——往数组开头添加一个或多个元素；
8. .pop（）——删除数组的最后一个元素
9. .shift（）——删除数组的第一个元素

**备注：增加方法（.push()，.unshift()）的返回值是数组增加元素后的长度；**

**删除方法（.pop()，.shift()）的返回值是被删除的那个元素；**

**.splice方法的返回值：被删除的元素组成的数组，可能是空数组**

1. .splice（startidx，deleteNum，...items）

\* 参数：

\* startidx开始的索引

\* deleteNum 删除数组元素的个数

\* items 数组中的元素

\* 作用：

\* 删除元素 splice（idx，delNum）;

\* 添加元素 splice（idx，0，item）；

\* 修改元素 splice（idx，1，item）；



1. **数组的其他方法**
2. slice(start[,end])——截取数组（原数组不变）

\* 返回值为裁切数组中的某一部分组成的数组

\* 从strat开始到end结束（不包括end所对应的元素）

\* end省略，代表一直截取到数组的最后一项

\* 支持负数

**备注：可应用于复制一个数组为新数组，通过slice(0);**

1. 将数组拼接成字符串，用join(separator)

\* 返回值为拼接后的字符串，其中包含了连接到一起的数组的所有元素

\* 其中separator为分隔符，默认为**逗号**；

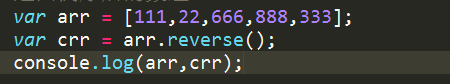
1. 数组合并——得到一个新的数组——**不改变原数组**

\* 用concat（数组1[,数组2]）方法——返回一个新数组，这个新数组是由调用这个方法的数组和参数组成

\* 参数可以是多个

1. reverse()——数组倒序——**会改变原数组**

\* 只将数组中的元素颠倒顺序，返回逆序后的数组

\* eg：

1. **数组的排序（数组的sort()方法）——改变原数组**
2. arr.sort() 直接修改原数组——所有的数据必须经过排序后再给到客户

问题：默认sort方法，只能将所有元素转为字符串再排序，默认用字符的ascii码进行比较？

结果：无法对数字类型排序

解决：自定义比较规则：

1. 先定义比较器函数对象

比较器函数：专门封装任意两数比较逻辑的函数

参数必须是2个：a，b

函数体：比较两个数的大小

返回值：如果a>b，必须返回>0的数

如果a<b，必须返回<0的数

若果a=b，必须返回=0的数

数字比较器：

升序：function compare(a,b){return a-b};

降序：function compare(a,b){return b-a};

始终a是当前，b是下一个，所以参数a，b顺序不能变更

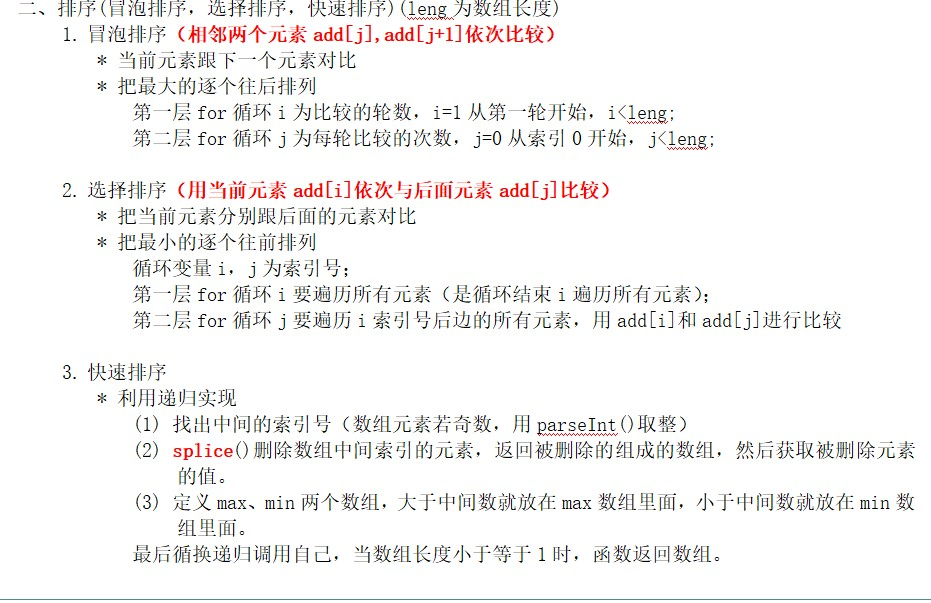
2、将比较器函数对象，传入sort方法内：

\*\*\*\*js中函数也是一个对象！—函数就能传来传去，并且函数可以随意的调用

\*\*\*\*函数对象可以被当做参数传递！

\*\*\*\*传递到哪儿，就可以在哪儿调用！

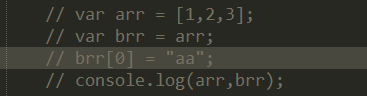
arr.sort(compare);



**Day06**

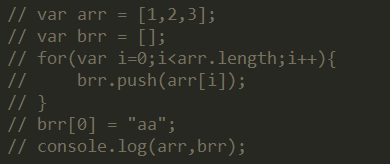
1. **数组的传递及复制**
2. 传递 var brr =arr;

\* 数组是引用类型的对象，上面的表示是将arr在堆中的地址传给了brr，所以一旦brr的值进行修改，arr也会受影响

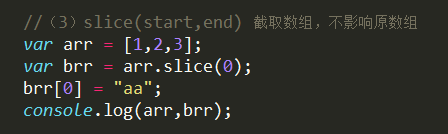
这两个console.log的值是一样的

1. 复制

1、利用for循环拷贝里面的每个元素，如下图所示：

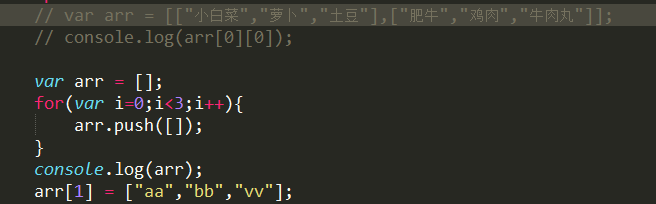


1. 利用数组的slice(0)方法，给方法不会改变原数组，如下图所示：

****

1. **多维数组：数组套数组[[],[],...]**

如下图：操作方法同数组一样



1. **对象（对象的很多操作和数组是一样的）——谨记数组的操作方法**
2. 对象的定义
3. 字面量 var obj={}——**这种创建对象的方式比较常用**

2、构造函数 var obj=new Object();——这种不常用

（二）对象的组成：键值对

{键：值，}===>{属性名：属性，}

1. 对对象的操作：通过键获取值（类似于数组的操作，利用索引号获取某个数组的元素）

对象.具体的键(属性名)

对象[“具体的键（属性名）”]====>obj[变量key]

\* 删除某个键：delete obj.属性名;

1. 遍历对象 for...in

for(var key in 对象){

//key代表对象的每一个键

}



1. **重点（es5中新增数组的方法）**
2. 遍历方法——不改变原数组

**（备注：以下提到的fn（表示的是函数）有三个参数，**

**\* item 数组中的每一项**

**\* idx 索引**

**\* arr 当前数组）**

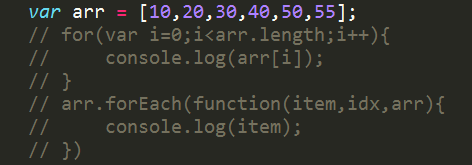
1. forEach（fn）——与for循环没有太大差别，比for循环更加方便

如下：arr.forEach(function(item){

console.log(item)

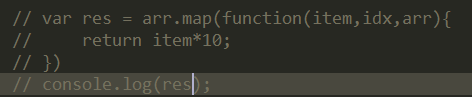
});——重点注意：**forEach（）方法没有返回值的**

Eg：如下图所示



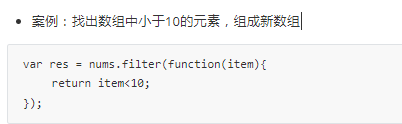
1. map（fn）方法——返回一个数组（数组的长度等同于原数组的长度），返回的数组里面的元素是什么取决于在fn中返回的值

**该方法：有返回值，不同的参数，返回的是不同的内容**



**小结：forEach（fn）和map（fn）这两种方法类似，只是返回值的问题，forEach（fn）方法没有返回值；map（fn）有返回值——返回值可以直接用一个变量接收，并当做一个值来使用**

1. filter（fn） 返回一个数组，存放执行fn后返回true的数组元素。利用这个方法可以对数组元素进行过滤筛选



（4）some(fn) 判断数组中是否有一个元素满足return的条件，有即最终返回true——类似于逻辑 运算中的或运算，**返回值是boolean类型的值；**

（5）every(fn) 判断数组中是元素是否满足return的条件，有一个不满足条件，最终返回false——类似于逻辑运算中的与（&&）运算，**返回值是boolean类型的值；**

（二）归并方法——reduce（fn，initval）——**返回的是最后的值**

\* fn的参数

\* prev 上一次函数的返回值

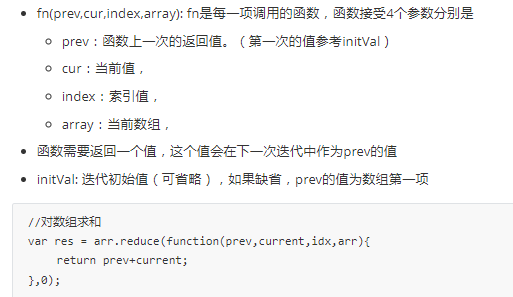
\* 若reduce的第二个参数initval有值，则prev第一次的值为initval

\* 若reduce的第二个参数initval没有值，则prev第一次的情况会默认以索引为0对应的元素为prev的值。

\* item当前数

\* 当前索引

\* 数组



1. 静态方法

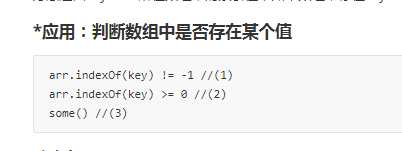
Array.isArray()判断是否为数组，若为数组则返回true，否则返回false

console.log(Array.isArray([1,2,3]));

1. 数组的索引方法
2. indexOf（keyword[,startidx]）——从前往后找
3. lastIndexOf（keyword[,startidx]）——从后往前找

返回keyword所在数组中的索引值，如果数组不存在keyword，则返回-1；

**判断数组中是否存在某个值的方法：**



**Day08**

1. **字符串**
2. 字符串的定义

字符串就是一串字符，由双（单）引号括起来。

1. 创建字符串
2. 字面量（推荐）

var str=“知识就是财富”;

1. 构造函数（不推荐使用）

Var str2=new String(“知识就是财富”);

1. 操作字符串——**操作方法类似于数组的操作方法**

通过索引idx获取字符：

\* str[idx] ==>这个是es5的方法,,ie9才使用

\* str.charAt(idx)

获取字符串的长度，str.length

1. 字符串**的**方法——重点需要掌握的——重点，**字符串的方法都不会改变原字符串的**
2. indexOf / lastIndexOf（字符[,startIdx]）从开头 / 尾部向后找字符串 keyword（每次只能知道一次）第一次出现的位置，如果没有就返回-1；

**小结：**indexOf（）和lastIndexOf（）是**数组**和**字符串**的方法，若是其他的数据类型用了这个方法，会出现报错##.\*\*is not a function

1. split（切割符）根据分割字符，把字符串拆分成数组，最后返回的结果是数组
2. 字符串的替换方法

replace（str|regExp(正则)，替换的字符），原字符串不变，返回被替换的字符串

\* 字符串的替换只能执行一次，不能够进行全局匹配

\* 如果需要全局匹配，则应是用正则表达式

1. 字符串的截取方法
2. slice（startIdx[,endIdx]）（同数组的裁切方法一样）返回被裁切后的字符串

\* 不包括endIdx所在字符

\* endIdx省略，代表截取到最后一个

**\* 支持负数**

1. substring（startIdx[,endIdx]）——方法同slice方法一样

\* 不支持负数下标

1. substr（startIdx[,length]）返回裁切后的字符串

\* length省略，代表截取到最后一个

\* startIdx支持负数

（5）字符串大小写转换

\* toLowerCase()：转换成小写

\* toUpperCase()：转换成大写

（6）ASCII码和字符集

\* 字符串.charCodeAt（3）//获取下标为3的字符的编码

\* String.fromCharCode（97）//编码转换成字符

1. **trim() 裁切掉字符串的前后空格——这个是es5的方法，ie9及以上**
2. 初始正则表达式
3. 正则表达式的创建
4. 字面量

var reg=/字符/；

1. 构造函数——这种创建比较常用

var reg=new RegExp（“字符”，“gi”）；

var reg=new RegExp（变量，“gi”）；

\* 正则表达式的参数

g 全局匹配

i 不区分大小写

**Day09**

1. **Math对象方法——一个保存数学公式和信息的对象，一般用于执行数学任务**
2. 属性：Math.PI=3.1415926

（一）Math的方法——Math.

\* round（3.6）四舍五入取整

\* ceil（11.3）12向上取整

\* floor（11.8）11向下取整

\* random()返回0-1之间的随机数（不包括1）

\* max（num1，num2）返回较大的数，这里的数可以是多个，不单单是两个数的比较

\* min（num1，num2）返回较小的数，这里的数可以是多个，不单单是两个数的比较

\* abs（num）取绝对值

\* pow（x，y）x的y次方

\* sqrt（num）num开平方根

（二）相关的数学方法

\* Math.sin（弧度)

\* Math.cos（弧度)

\* Math.tan（弧度)

\* 角度与弧度的转换

弧度=角度\*Math.PI/180

1. **日期的相关的方法**
2. 创建日期对象

1、创建当前时间的日期和时间

var d = new Date();//得到的是代码执行时的时间（本地时间）

2、创建指定日期的时间和日期

var d = new Date("2015/08/22");

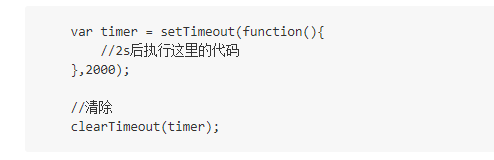
var d = new Date(56521211021);//参数为距1970-1-1零时的毫秒数

1. 日期对象**的**获取方法——**获取方法需要用新的变量来接收，不会改变原来的时间**
2. getFullYear()——获取年份
3. getMonth()——获取月份，得到0-11
4. getDate()——获取日期
5. getDay()——获取星期几，得到0-6，星期天到星期六
6. getHours()——获取时
7. getMinutes()——获取分
8. getSeconds()——获取秒
9. getTime()——获取某个日期自1970年1月1日0时以来的毫秒数
10. **定时器（比较常用）—**—**这个在动画中特别常用**
11. setInterval(fn,time)——每隔time毫秒执行一次函数，间隔执行

\* time——指的是毫秒数

1. setTimeout(fn,time)——延迟time毫秒执行函数，只执行一次
2. clearInterval(变量名)——清空变量名对应的定时器
3. clearTimeout(变量名)——清空变量名对应的定时器

Eg：



（五）日期对象**的设置**方法——**设置方法不需要用新的变量接收，直接改变原来的时间**

1、setFullYear()——设置年份

2、setMonth()——设置月份，得到0-11代表1-12月

3、setDate()——设置日期

4、setHours()——设置时

5、setMinutes()——设置分

6、setSeconds()——设置秒

7、setTime()——修改某个日期自1970年1月1日0时以来的毫秒数

（六）es5获取毫秒数（静态方法）

1、Date.parse("2020-01-01 00:00:00") 获取特定日期距离1970年的毫秒数

2、Date.now() 获取当前日期距离1970年的毫秒数

（七）日期的处理

1、toLocaleDateString(); 以特定地区格式显示年、月、日

2、toUTCString(); 转换成UTC时间

**Day10**

1. **BOM**

**（一）BOM的概念**

BOM是Browser Object Model（浏览器对象模型）的缩写，提供与浏览器窗口进行交互的对象。JavaScript语法的标准化组织是ECMA，DOM的标准化组织是W3C，而BOM缺乏标准。这也是各种浏览器不兼容的根源所在

1. **Window**
2. 基本概念

\* 全局变量都是window对象的属性，使用时可以省略window对象。

\* 若函数内没有通过var声明的变量，会被当做全局变量。

\* 把变量 尽量声明在函数体内部

\* 避免全局污染，全局变量起名字不要起name，top

\* 避免命名冲突

\* 全局变量无法用delete删除，但是对象的键可以用delete obj.key删除

\* window对象本身存在的属性及事件，调用的时候尽量不要省略window的调用。以及不要对window本身存在的属性赋值！

\* 一般window的方法可以省略window.调用。例如window.alert()可以省略window.

1. window**的**属性及事件
2. window.innerWidth/window.innerHeight——表示浏览器窗口“可视区域”的尺寸

\* window.outerWidth/window.outerHeight——表示浏览器窗口尺寸

1. 与滚动相关的属性

\* scrollX / scrollY——获取浏览器窗口滚动过的距离

\* scrollTo（x，y）——设置浏览器滚动到的位置

\* scrollBy（x，y）——设置相对于当前滚动位置滚动的距离

1. window的事件

\* window.onscroll——浏览器滚动条滚动时触发的函数

\* window.onload——等待页面加载完毕后触发

\* window.onresize——屏幕尺寸改变时触发

1. 系统弹窗
2. alert()——弹窗
3. prompt（msg，默认值）——输入框，返回值为消息或者是null
4. confirm(msg)——确认框，返回值为布尔值
5. open(url)——打开一个新窗口并返回新window对象
6. close()——关闭本窗口
7. print()——调出打印对话框
8. 属性对象
9. **history(重要)：**历史对象，包含窗口的浏览历史，可以实现后退

属性length：返回浏览器历史列表中的URL数量。

\* back()加载history列表中的下一个URL。

\* forward()加载history列表中的下一个URL。

\* go()加载history列表中的某个具体页面，支持负数。



1. **location(重要)：保存文档的相关信息**

属性：这个是重中之重

\* hash设置或返回从井号（#）开始URL（锚）==>哈希值。

\* href 设置或返回完整的URL。——**主要用于A页面设置链接**

\* search 设置或返回从问号（？）开始的URL（查询部分）——主要用于B页面**接收链接**

**Eg：商品页和详情页相互之间的传递**

**A页面用来设置链接，B页面用来接收返回的链接，并将其用对象的方式接收保存起来，以便后期的调用**

encodeURI（）；转码

decodeURI（）；解码

**报错信息：**



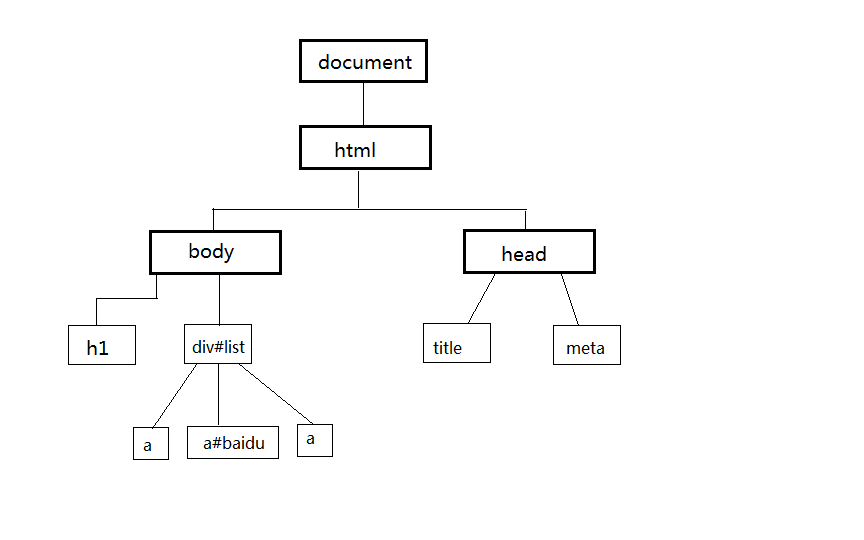
**Day10**

1. **DOM基本概念**

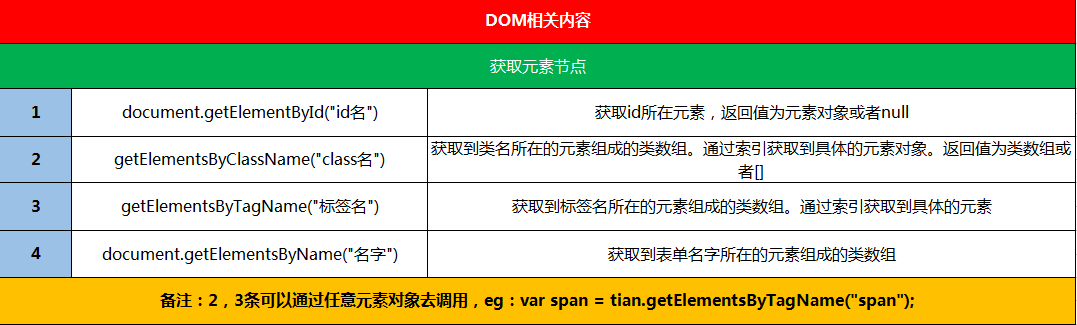
DOM是Document Object Model（文档对象模型）的缩写，它是W3C国际组织的一套Web标准。是针对HTML和XML文档的一个API，它定义了访问HTML文档对象的一套属性、方法和事件。

**节点是对象**

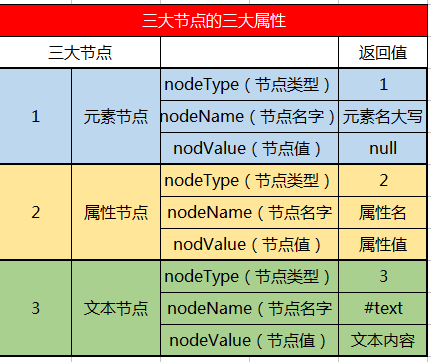
**DOM树如图所示：**



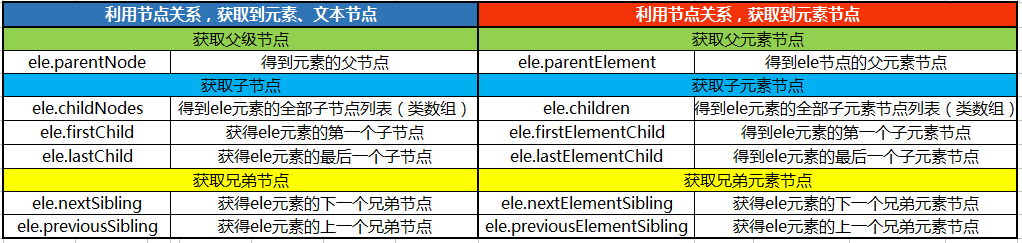
1. **获取元素节点——节点本质上也是一个对象**

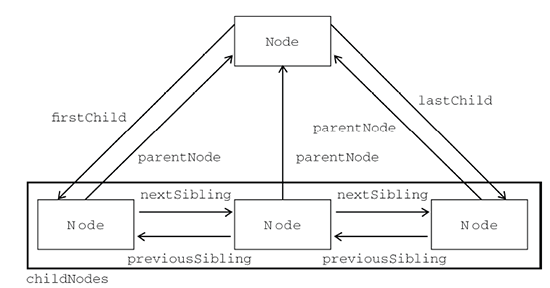
**Eg：相关内容如下——document.body获取body；document.documentElement获取html**

**三、三大节点的三大属性**



**四、利用节点关系，获取元素、文本节点**



****

1. **节点的增删改查**
2. 节点的创建
3. document.createElement(“标签名”)——创建一个元素节点
4. document.createTextNode(“文本”)——了解，创建一个文本节点
5. document.createAttribute(“属性名”)——了解，创建一个属性节点
6. 节点的插入
7. 父节点.appendChild(子节点)——往父元素节点插入最后一个子节点
8. 父节点.insertBefore(子节点，页面已经存在的)——在指定的子节点node前插入新的子节点new。
9. 父节点.setAttributeNode(属性节点)——在指定元素中插入一个属性节点（了解）
10. 节点的删除、复制
11. 父节点.removeChild(ele)——父节点删除子节点，若要删除子节点的话，套路让我的父节点把我删了：ele.parentElement.removeChild(ele)
12. 当前节点.cloneNode(boolean)——复制节点，为true是深复制。返回值为复制后的新节点**（需要用一个新的变量来接收一下）**

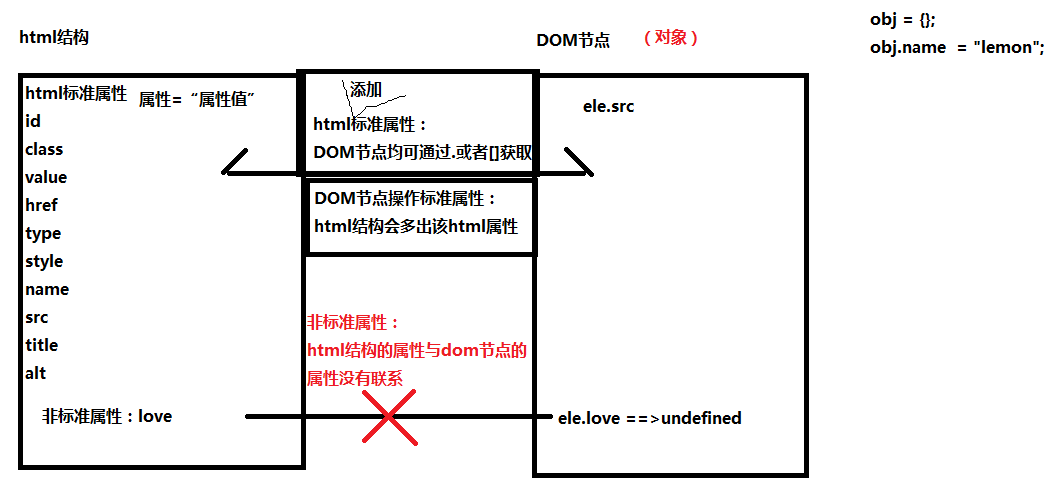
**重点：cloneNode（boolean）——深复制克隆了元素里面的所有的内容，但是不会复制元素的行为**

1. 判断：父节点.hasChildNodes()——判断当前节点是否拥有子节点，返回布尔值，括号中如果有参数的话，表示判断是否有这个子节点
2. **节点的属性及方法**

**\* 节点是对象，**给对象添加属性与给html元素添加属性不是一个意思

\* html标准属性会相互影响，但是 不是标准的属性不会互相影响

如下图所示：



1. 节点（对象）**的**属性
2. tagName ——获取元素的标签名
3. className——设置 / 获取元素的class属性——**获取到元素的所有的类名，所以如果想选中某个元素的话，该元素的class类名必须全部写全，否则选不中该元素**
4. id——设置 / 获取元素的id属性
5. name——设置 / 获取元素name属性 —— 一般用于表单中
6. style——设置 / 获取元素的内联样式
7. innerHTML——设置 / 获取元素的内容（包含html代码）
8. innerText——设置 / 获取位于元素标签内的文本
9. 节点**的**方法
10. setAttribute（“html属性”，“属性值”）——给html元素添加属性
11. getAttribute（“html属性”）——拿到html属性对应的属性值
12. ele.removeAttribute（attr）——删除html属性
13. ele.hasAttribute（attr）——判断是否存在html属性
14. **节点的盒模型相关的属性**
15. offsetWhite 当前元素的宽度（border+padding+content）
16. offsetHeight 当前元素的高度（border+padding+content）
17. offsetLeft 当前元素离<定位父级>元素左边的距离
18. offsetTop 当前元素离<定位父级>元素顶部的距离

\* 若不存在定位父级，则相对于html的距离，offsetLeft,offsetTop表示的是相对于已定位父级的距离

1. **获取css样式（非内联样式） ..的样式**
2. style 只能操作内联样式
3. window.getComputedStyle(ele)——得到包含ele所有样式**的对象**
4. ele.currentStyle 得到包含ele所有样式**的对象**（IE8-）

注意：以上得到的都是对象，对象的调用需要用 . 调用，或者obj[“属性名”]

\* currentStyle 不能获得总属性

**Day12**

**Event 事件**

**一、概念**

事件是可以被javaScript侦测到的行为。网页中的每个元素都可以产生某些可以触发javaScript函数的事件。事件是一瞬间触发的。

1. **常用事件**
2. 鼠标事件

onclick 当用户点击某个对象时调用的事件。

ondblclick 当用户双击某个对象时调用的事件。

onmousedown 鼠标按钮被按下。

onmouseup 鼠标按键被松开。

onmouseover 鼠标移动到某元素之上。

onmouseout 鼠标从某元素移开。

onmousemove 鼠标被移动时触发。

onmouseenter 鼠标移入。这个事件不冒泡

onmouseleave 鼠标移出。这个事件不冒泡

oncontextmenu 鼠标右键菜单展开时触发。

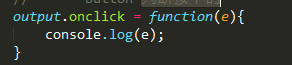
\* onclick=onmousedown+onmouseup

\* ondblclick=onclick\*2

执行顺序：onmouseover==>onmouseenter; onmouseout==>onmouseleave

1. 键盘事件
2. onkeydown——键盘按下触发
3. onkeyup——键盘弹起触发
4. onkeypress——键盘的**字符键**按下触发——只有字符键按下时才会触发相关的事件
5. **event对象**
6. 作用：监听事件执行过程中的状态，用来保存当前事件的信息

Eg:



ele.事件 = function(e){

e = e || window.event;**//获取event对象的兼容写法**。IE8-：window.event

}

**上面的属性存在兼容性的问题：兼容ie浏览器**

**e=e || window.event;**

（二）event对象**的属性**：

1、**鼠标相关的**

**\* 鼠标按键**

\* button 返回当事件被触发时，哪个鼠标按键被点击。

\* 标准浏览器：012左滚右

\* **光标位置**

clientX / clientY 光标相对于浏览器可视区域的左上角位置，也就是浏览器坐标。

screenX / screenY 光标指针相对于电脑屏幕左上角的水平/垂直坐标。

pageX / pageY 鼠标相对于文档左上角的位置。

\* 包括滚动条滚动的距离，即：e.clientX+window.scrollX

\* IE8-不支持

offsetX,offsetY: **光标相对于事件源对象左上角的相对偏移量。**

**\* 事件源对象：触发事件的对象——**

2、**与键盘有关的属性**

（1）which

\* \_对于keypress事件，该属性声明了被敲击的键生成的Unicode 字符码（ascii码）——这个用的比较少，用的比较多的是下面的那个——了解

\* \_对于keydown和keyup事件，它指定了被敲击的键的虚拟键盘码。

（2）ctrlKey 当事件被触发时，CTRL键是否被按下。返回布尔类型的值

（3）altKey——同上

（4）shiftKey——同上

**兼容写法：e.which || e.keyCode**

**四、事件冒泡**

**（一）概念**

在一个对象上触发某类事件（如onclick事件），那么事件就会沿着DOM树向这个对象的父级传播，从里到外，直至事件到达了最顶层（document/window）。

（1）不是所有的事件都能冒泡。

​ 以下事件不冒泡：blur、focus、load、unload…。

​ 【onmouseover与onmouseenter的区别】

（2）冒泡到最顶层的目标不同。大部分浏览器到window对象，IE8-到document对象

（3）阻止事件冒泡

**\* event对象.stopPropagation（）**

**标准：event.stopPropagation();**

**IE8-：event.cancelBubble = true;**

**// 阻止事件冒泡兼容写法：**

**if(e.stopPropagation){**

**e.stopPropagation();**

**}else{**

**e.cancelBubble = true;**

**}**

**阻止事件冒泡兼容写法：e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;**

1. **事件委托**

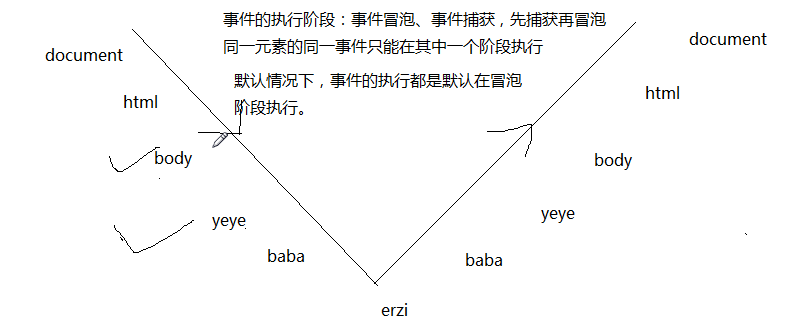
利用事件冒泡原理，把本来绑定为某个元素的事件委托给它的父级（已存在页面元素）处理。

\* 备注：影响页面性能的三大操作

1. 频繁操作dom节点
2. 过多的事件绑定
3. 页面请求过多

\* 事件源对象：获取到触发事件的元素（event对象.target）——触发该事件的对象

**备注：事件的执行阶段：事件冒泡、事件捕获，先捕获再冒泡，同一元素的同一事件只能在其中一个阶段执行，默认情况下，事件的执行都是默认在冒泡阶段执行。**



**五、阻止浏览器的默认行为**

浏览器默认行为：

\* 链接跳转

\* 表单提交

\* 右键菜单...

\* 文本的选择

标准：event.preventDefault();

IE8-：event.returnValue=false;

**兼容写法：e.preventDefault ? e.preventDefault() : e.returnValue = false;**

**六、事件的绑定方式**

1、html属性[onclick=“函数名(实参)”] 不推荐使用，维护不方便，调用数据困难；

2、“on”+type：

\* ele.onclick=函数名；

\* ele.onclick=function(){}

\* 只能在冒泡阶段执行，只能给同一个元素绑定一个相同事件

**3、事件监听器——重点**

**（当给同一个元素绑定一个事件但是需要执行多个方法，后面的方法会将前一个方法给覆盖掉，若想解决这个问题就需要使用到事件监听器）**

1、标准浏览器：元素.addEventListener(事件名（type）,事件处理函数（fn）,是否捕获（默认false，为冒泡；若为true为捕获阶段）)

target.addEventListener("click", fn, false);

type：事件类型

fn：事件处理函数

\* IE8-：元素.attachEvent(on+事件名,事件处理函数)

target.attachEvent("onclick",fun);

2、作用如下：

（1）可以绑定多个处理函数在一个对象上, 执行顺序按照绑定的顺序来(标准)；

（2）不同元素事件执行顺序跟html结构有关；

（3）相同元素事件执行顺序跟绑定先后有关；

（4）可以绑定多个函数在一个对象上, 执行顺序按照绑定的反序（ie8-）；

**封装：绑定事件，兼容浏览器**

function bind(ele,type,handler,isCapture){

// 优先使用事件监听器

if(ele.addEventListerner){

// 标准浏览器

ele.addEventListerner(type,handler,isCapture);

}else if(ele.attachEvent){

// IE8-

ele.attachEvent('on' + type,handler);

}else{

// DOM节点绑定方式

ele['on' + type] = handler

}

}

（二）取消绑定事件

1、ele[“on”+type]=null；

\* ele.onclick=null

2、事件监听器移除

标准：ele.removeEventListener(type,fn, true)传入的参数fn要跟添加时一样(同一个函数)，否则不能移除事件

IE8-：ele.detachEvent('on'+type,fun)，传入的参数fun要跟添加时一样，否则不能移除事件

**Day13**

**cookie相关的内容**

**一、概念：**

cookie是客户端与服务器端进行通讯使用的一个能够在浏览器本地化存储的技术。

chrome不支持本地文件的cookie读写

**二、cookie的基本设置及获取**

1、设置（存储）：

document.cookie=“cname=cval”

\* 每次只能设置一条cookie

2、获取：document.cookie

\* 一次性获取到所有的cookie，获取到的cookie是一串字符

**三、cookie的组成**

1、设置（存储）：

document.cookie=“cname=cval[； expirse=日期对象的字符串]”

\* expirse 设置期限（cookie保存的期限）；

\* 默认情况下，为临时日期

\* d.UTCString() 具体的日期

**\* path 设置cookie的存储路径**

\* 默认路径为当前文件所在的文件夹

\* / 根目录

2、获取：document.cookie

\* 一次性获取到从当前文件夹往上找到根目录的所有cookie

3、删除：利用将cookie设置成过期的时间

4、JSON字符串 **格式** ‘{“name”：“lemon”，“price”：98.88}’；

\* cookie中存储的必须是字符串，其他的不能存储

\* 转换

\* 对象/数组--> json字符串：JSON.stringify()

\* json字符串-->对象/数组：JSON.parse()

JSON字符串 格式

格式：

\* 属性名和字符串必须使用双引号

\* 不能有注释

\* 不能存在多余的逗号

属性值必须为一下类型

\* String

\* Number

\* Boolean

\* Object

\* Array

\* Null

**注意：空字符串不能通过JSON.parse（）转成对象，会报错：Unexpected end of JSON input（该内容不是JSON字符串）；**

**Day14：正则表达式——必须能看懂他人写的正则表达式**

1. **创建正则表达式**
2. 字面量 var reg=/字符串/gi；
3. 构造函数 var reg=new RegExp（“字符串”，“gi”）；

\* var reg=new RegExp（变量，“gi”）；

\* g 全局匹配

\* i 不区分大小写

1. **字符串的方法：（字符串的方法一般也适合正则）**
2. indexOf(keyword[，stratIdx])——这个方法不支持正则

返回第一次匹配字符在字符串中的索引，匹配不到返回-1；**不支持正则**

1. search（keyword）返回第一次匹配字符在字符串中的索引，**支持正则表达式，**匹配不到返回-1
2. match（字符串||正则） 默认匹配字符串，返回一个数组

0：所匹配的字符 ； index：匹配第一个字符所在的索引

input：对字符串的引用

\* 全局匹配（g），返回一个所有满足匹配条件的字符串组成的数组

\* 如果匹配不到则返回null

4、replace（“需要替换的内容”，替换的内容）——替换字符串，不会改变原来的字符串—支持正则

1. **正则匹配规则：**
2. 字符类

|  |  |  |  |
| --- | --- | --- | --- |
| . | | 匹配换行外的任意字符 | |
| \d | 数字0-9 | \D | 非数字 |
| \w | 包含字母、数字、下划线 | \W | 非数字字母和下划线字符 |
| \s | 空格 | \S | 空格以外的字符 |
| \b | 匹配数字或单词边界（空格、汉字） | \B | 匹配非单词边界 |

1. 特殊符号

{} 表示数量，默认贪婪模式（会尽可能多的去匹配结果）

非贪婪模式（加？号）：str.match（/a{1,}?/g）——“a”，匹配尽量少的内容；

\* {a} 表示匹配a个

\* {a，} 表示匹配a个及以上

\* {a，b} 表示匹配a到b个

+：{1，}

\*：{0，}

？:{0，1}

[] 或者，表示满足中括号内其中一个条件即可

- 表示区间

^ 表示非

锚点定位：

^：表示以什么开头

$：表示以什么结尾

|：或者

(): 表示分组（n是以最左边括号出现的顺序排列）

$1: 表示第一个分组

$n: 表示第n个分组（不能写在正则表达式里）

\n: 在正则分组后面使用，表示对第n个分组的引用(一定要写在正则表达式里)

**备注：若是正则表达式中含有特殊的字符的话，必须使用转义字符“\”转移一下**

**四、正则表达式的属性和方法**

测试正则表达式用test方法,返回布尔值——**判断某个字符串是否符合正则表达式的规则，true符合，false不符合**

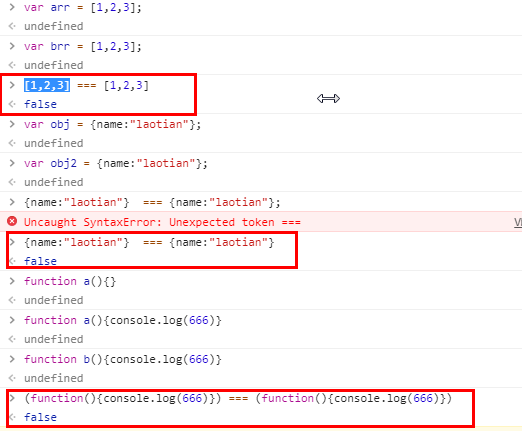
格式：正则表达式.test(字符串)

用<正则表达式>测试<字符串>是否匹配,返回true/false

测试正则表达式exec方法

/xx/.exec(字符串)

**重点：创建引用数据类型，保存的是地址；**



**Day15——ES5**

1. **支持ES5的浏览器**

\* Opera 11.60+

\* Internet Explorer 9+（IE9不支持严格模式）

\* Firefox 4+

\* Safari 5.1+

\* Chrome 13

1. **加载事件**

（一）DOM文档加载的步骤为（页面加载顺序及事件如下：）

1. 解析HTML结构。
2. 加载外部脚本和样式表文件。
3. 解析并执行脚本代码。
4. DOM树构建完成。//DOMContentLoaded
5. 加载图片等外部文件。
6. 页面加载完毕。//window.onload

（二）事件

1、onreadystatechange 准备阶段改变时触发函数

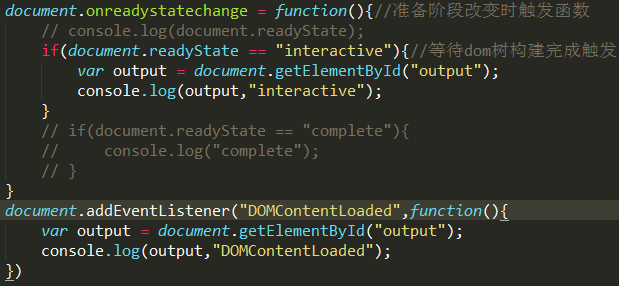
\* document.readyState 准备阶段

\* interactive 等待dom树构建完成时触发

\* complete 等待页面加载完毕时触发

2、document.addEventListener（“DOMContentLoaded”，function(){}）——DOM树构建完成——这种方式使用的比较多，一般的程序都是这么写，等到所有页面都加载完成后再执行程序，可能会出现延迟等问题

**代码案例如下：**

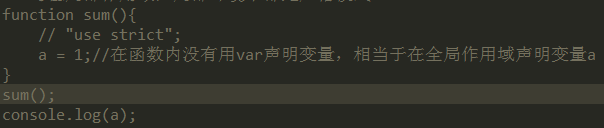


**三、严格模式（ie9及以下不支持）**

“use strict”; 写在当前作用域的最顶端

\* 写在全局作用域，全局环境下都是严格模式

\* 写在局部作用域，局部环境下都是严格模式



**四、获取元素节点**

1、document.querySelector（“css选择器”）获取匹配选择器的第一个元素节点，返回DOM节点

2、document.querySelectorAll（）获取匹配选择器的所有元素，返回数组

**以上两种方法支持forEach（）方法；**

**五、Function方法**

\* bind(指定对象)

**用于将当前函数和指定对象绑定（改变函数内this指向为指定对象），返回一个新的函数**

Eg：相关的案例如下：



**六、获取class列表属性**

classList 返回的是**类数组**，包含所有的类名，js的方法

\* length：class类名的个数

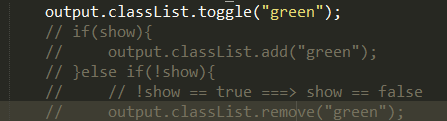
\* add()：添加class方法

\* remove()：删除class方法

\* toggle()：切换class方法

\* contains()：判断时候含有某个类名，返回的是布尔值

写法为：ele.classList.方法

****

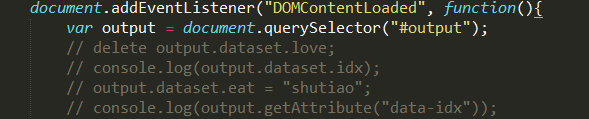
**七、自定义属性**

1、符合W3C标准自定义属性：html属性都是以“data-\*”呈现的

2、dataset：存放所有data自定义属性对象。

\* 获取 ele.dataset.age；//获取data-age的属性对象

\* 设置 ele.dataset.age=“”//设置data-age的属性值



**Day16（ES6）**

1. **声明变量**
2. let：声明变量
3. 没有声明提前的概念
4. 不能在同一作用域重复声明同一个变量
5. 块级作用域{}

**报错消息：Identifier 'a' has already been declared 变量a已经被声明**

1. const：声明常量
2. 变量声明不会提前
3. const不允许相同作用域内多次声明同一变量
4. 块级作用域{}
5. **声明后无法修改值**

**报错消息：Assignment to constant variable不能修改常量**

**备注：const常用于引用第三方库的声明**

1. **解构（其实就是赋值）**

（一）概念：声明变量时，从数组和对象中提取值，对变量进行赋值，这被叫做“解构”。

（二）数组解构（**[]=[]两两对应**）

1、基本解构：[变量名，变量名]=数组

Eg：var [a,b,c]=[10,20,30];

2、...表示获取剩余参数

Eg：var [a,...b]=[10,20,30];//a=10,b=[20,30]

3、解构失败

（1）变量的长度大于数组的长度，出现解构失败，得到undefined

（2）解构右侧不是数组，会出现报错。

4、指定默认值：当a解构失败的话，拿到默认值

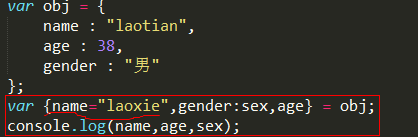
Eg：var arr=[10,30]；

var [a=0,b=0,c=0]=arr;

1. 对象解构（**{}={}两两对应**）
2. 基本解构

\* 要求变量名与对象的属性名一致，这里的变量名可以更换前后的顺序

1. 解构失败：变量名与属性名不一致，会出现解构失败
2. 如果变量名与属性名不相同，则必须写成以下格式才能取到值：{属性名：变量名}



1. 指定默认值：当变量解构失败的话，拿到默认值

Eg：var {a=10} = {};//当a解构失败的话，拿到默认值a=10

\* 不仅解构时可以传递默认值，基本数据类型作为形参时也可以传递默认值，也可以通过“...变量名”获取剩余的参数

1. **字符串扩展**
2. 字符串方法
3. includes 判断是否包含某个字符，返回布尔值

Eg：console.log(“abc”.includes(“a”));

1. startsWith / endsWith 是否以某一字符或字符串开头 / 结尾，返回布尔类型的值

Eg：console.log(“abc”.startsWith(“a”));

console.log(“abc”.endsWith(“c”));

1. repeat（n）得到字符串重复n次后的结果，n可以是小数，但不能为负数

Eg：console.log("I love U ❤ \n".repeat(1.5));

1. **字符串模板（重点需要掌握的内容）**

使用反引号` `表示，使用 ${变量} 往字符串中插入变量

Eg：console.log(`好，我的名字叫${username},接下来是我的自我介绍：${introduce()}` );

备注：以上方式更加的美观和直观，模板字符串中所有的空格、换行、缩进，都会原样输出在生成的字符串中。

1. **函数扩展**
2. **箭头函数（重点需要掌握下）**

\* 格式：标识符=>表达式

\* 省略了 function、return关键字、圆括号、花括号

1. 函数内只有一句return代码：省略了function、return、{}

Eg：var sum = function(){return 666;}

**简写**：var sum = () => 666;

1. 没有参数或存在多个参数，写法如下

Eg：var sum = function(a,b){return a+b;}

**简写**：var sum = (a,b) => a+b;

1. 只有一个参数，还可以省略();

Eg：var sum = function(a){return a+1;}

**简写**：var sum = a => a+1;

1. 函数内有多句代码，不要省略{}，return

\* 箭头函数=>后面紧跟着的 {} 中的内容是代码块

Eg：var sum = function(a,b){

var c = a \* b +1;

return c+1;

}

简写： var sum = (a,b) => {

console.log(this);

var c = a \* b +1;

return c+1;

}

1. **箭头函数自身没有this值，箭头函数内的this值继承自外围的作用域。**
2. 生成器函数 Generators

\* function 和函数名之间有一个 \* 号===>function\* 函数名（）{}

执行生成器函数会返回一个suspended的对象

\* yield：暂停代码执行

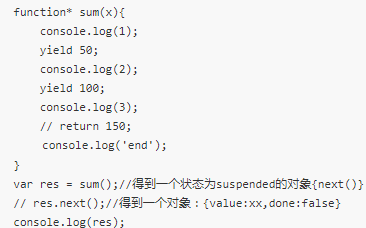
\* return：终止函数

\* next()：执行后得到一个对象，有两个属性如下：（next（）是suspended的方法）

\* value：暂停是返回的值（yield）

\* done：表示函数是否执行完毕

Eg：生成器函数例子如下：



**五、set集合**

（一）概念：Set集合，类似于数组，但是成员的值都是唯一的，可自动去重。

\* 去重的前提是两个值恒等于（===）。

\* 利用set集合去重数组

**\*\*\* Array.from(set集合)，将set集合或者类数组转成数组（重点）；**

（二）set的方法：

\* add(value)——添加某个值，返回Set结构本身

\* delete(value)——删除某个值，返回一个布尔值，表示删除是否成功。

\* has(value)——返回一个布尔值，表示Set集合中是否存在该值。

\* clear()——清除所有成员，没有返回值。

1. 遍历set集合

\* forEach()

\* for(var item of set){item代表集合中的每一项值}

**六、对象的扩展（重点）**

（一）对象的合并 Object.assign（obj1，obj2）将多个对象合并成obj1，并返回obj1，后面的对象会覆盖前面的对象。

（二）对象的传递与复制（**对象的浅复制与深复制是重点内容**）

1、var obj2=obj1；对象是引用类型，在堆中传递的是地址，所以一旦改变其中一个对象的值，另外一个也会受到影响

2、浅复制（对于引用类型，只拷贝地址，无法进行完全的复制）

（1）for...in——遍历对象中的键，传递给另外一个对象

（2）var new=Object.assign({},obj1);

3、深复制：var new=JSON.parse（JSON.stringify(obj1)）;——利用JSON字符串进行深复制

\* 将引用数据类型转成基本数据类型进行传递

\* 在转成引用数据类型

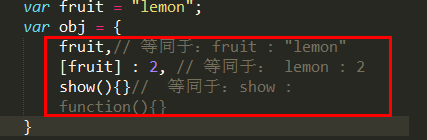
（三）对象的简写

1、属性名直接写成变量名：变量名作为属性名

2、属性名为[变量名]，代表变量值为属性名

3、方法简写：方法名(){}

Eg：对象的简写如下



1. map集合：它能让所有类型的数据作为键

常用的方法：

\* 设置set（key，value）

\* 获取get（key）

\* has（key）判断是否含有某个键，返回布尔类型的值

\* delete（key）判断某个键是否被删除，返回布尔类型的值

\* clear（）清空对象中的键值对

遍历的方法：

\* keys（）——获取所有键（类数组），可以用Array.from（）转成数组

\* values（）——获取所有值（类数组），可以用Array.from（）转成数组

\* entries（）——获取所有键值对（类数组），可以用Array.from（）转成数组

\* for（var item of map）{item存放map集合的每一项键值对，item为一个数组，item[0]代表键，item[1]代表值}

\* 配合解构，获取for（var [key,val] of map）{key代表map集合的每个键，val代表每个值}

1. **Symbol（了解）**

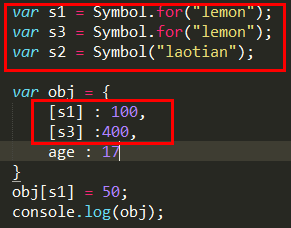
表示独一无二的值，一旦创建后就不可更改

\* 接收一个字符串作为参数，表示对Symbol实例的描述，主要是为了标识和区分

\* Symbol不可以与其他的值进行运算，会报错

\* Symbol.for（）第一次就是创建symbol，会先查找当前Symbol是否存在——这个主要是为了防止命名重复

Eg：



1. **for...of方法——这个比较好用，必须牢记**

\* 这是最简洁、最直接的遍历数组元素的语法

\* for...of跟for...in（用在对象中）的区别很明显，就是直接取值，而不再取下标了

\* 与forEach（）不同的是，它可以正确响应break、continue和return语句

备注：**只要有[迭代器Symbol(Symbol.iterator) ]就可以用for...of遍历——这个在控制台中就能看到了**

\* Array

\* DOM

\* Set/Map集合

\* String

\* 不支持普通对象

**Day17动画效果（主要运用的就是定时器，setInterval，setTimeout）**

一、动画

利用定时器，改变对象的属性，从而改变样式，产生动画效果

（一）匀速动画

1、初始化速度变量

2、开启定时器

（1）获取当前值

（2）改变当前值：当前值+速度

（3）将改变后的值赋值给元素的样式

（4）当改变后的值大于等于目标值，清除定时器，同时将改变后的值改成目标值。这一步应该在赋值给样式之前

（二）加速动画（顾名思义就是速度会越来越快，在匀速动画的基础上，多了个速度的变化）

1、初始化速度变量

2、开启定时器

（1）获取当前值

（2）改变当前值：当前值+速度

（3）速度不断增加

（4）将改变后的值复制给元素的样式

（5）当改变后的值大于等于目标值，清除定时器，同时将改变后的值改成目标值。这一步应该在赋值给样式之前。

（三）减速动画（速度在不断的减小的动画，有可能会变成赋值）

1、初始化速度变量

2、开启定时器

（1）获取当前值

（2）改变当前值：当前值+速度

（3）速度不断减小

（4）将改变后的值赋值给元素的样式

（5）当改变后的值大于等于目标值，清除定时器，同时将改变后的值改成目标值。这一步应该在赋值给样式之前

（6）当速度小于0时，清除定时器。写在速度自减下面

（四）抛物线运动（可以认为是水平方向上速度不变，主要的速度变化发生在垂直方向上）

1、初始化速度变量（水平、垂直）

2、开启定时器

（1）获取当前值(水平、垂直)

（2）改变当前值：当前值+速度

（3）速度不断减小（垂直方向的速度）

（4）将改变后的值赋值给元素的样式

（5）当改变后的值大于等于目标值，清除定时器，同时将改变后的值改成目标值。这一步应该在赋值给样式之前

（五）匀速动画(圆周运动)

1、初始化速度变量（角度）

2、开启定时器

（1）获取当前值

（2）改变当前值：当前值+速度

（3）将改变后的值赋值给元素的样式

（4）当改变后的值大于等于目标值，清除定时器，同时将改变后的值改成目标值。这一步应该在赋值给样式之前

**(六) 缓冲动画（当快到达目的地的时候，速度逐渐变慢）——重点，速度的变化是没有规律可循的**

关键：动态计算速度（目标值-当前值有关）

1、开启定时器

（1）获取当前值

（2）获取当前速度(目标值-当前值).

\* 当速度大于0时，Math.ceil()

\* 当速度小于0时，Math.floor()

（3）改变当前值：当前值+速度

（4）将改变后的值赋值给元素的样式

（5）当改变后的值等于目标值，清除定时器

\* 若事件里面开启定时器，记得开启定时器先清除定时器

**轮播图的制作方法：**

1. **先创建图片，制作好最简易的轮播的图样式（图片动画轮播），这里需要注意的点是，当滑到最后一张图片的时候，需要变回第一张，同时left的值需要变成0；**
2. **创建下方的按钮，对应的图片有多少张，则对应的按钮有多少个，已进入页面，第一个按钮显示高亮；**
3. **当移入banner图时出现，出现banner图暂停，移出banner图时，图片继续滚动；**
4. **按钮效果，当点击对应的按钮时，跳转到对应的图片。同时对应的按钮出现高亮；**

**Day20——PHP**

1. **php的基本语法**
2. 基本结构 <?php ?>
3. 内部注释同js一样，//——单行注释，/\* \*/——多行注释
4. 变量

- 以 $ 符号开始，后面跟着变量的名称（$称为标识符，不属于变量组成部分）

- 只能包含字母、数字字符以及下划线，不能以数字开头（不能包含空格）

- 区分大小写

- 没有声明变量的命令，也没有声明提前的概念

- 定义常量并赋值：define（“MY\_NAME”,“laoxie”）；

1. **php作用域也分为全局作用域和局部作用域：**

- 全局变量：在函数外部定义的变量，可以在任意位置访问（需要手动定义为全局变量）

- 局部变量：函数内部声明的变量，仅能在函数内部访问

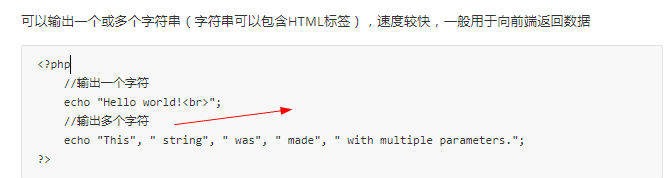
\* 手动定义为全局变量：

\* global关键字：global $变量名

\* $GLOBALS 超级全局变量，数组：$GLOBALS[“变量名”]

1. 输出语句

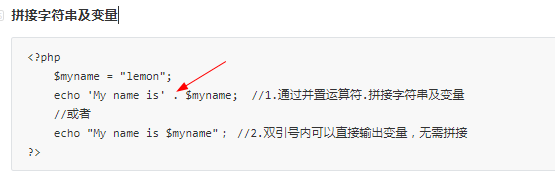
- echo 可以输出一个或多个字符串（字符串可以包含HTML标签），速度较快，一般用于向前端返回数据（备注：一个页面只出现一个echo，如果出现的多了，就会变成字符串拼接了）



- **print\_r（）**——打印关于变量的信息，适用于数组、对象的打印，一般用于测试

- var\_dump（）——判断一个变量的类型与长度，并输出变量数据类型和数值，一般用于测试

（6）拼接字符串。



1. **数据类型**
2. 字符串 string

- strlen（）——获取字符串的长度，得到的**字符的字节数**

- mb\_strlen（）——获取字符串长度

- strpos（）——查找某个字符在字符串中的索引，如果未找到匹配，则返回false，若是找到的话，返回的是对应的字节数，一个汉字大概对应的是3个字节数

1. Array——**php的数组类型，这个是比较常用的**
2. 创建数组 array（）
3. 数值数组 array（1，“aa”，3）
4. 关联数组（类似于js对象）

\* => 代替：

（3）多维数组：一般都是外层为数值数组，内层为关联数组（类似于js的数组对象）

2、操作数组

- 数组[索引/“键”]

3、常用方法：

- count（）——获取数组的长度

- in\_array（）——判断某个值是否存在数组中

- array\_slice（）——从数组中取出一段

- array\_rand（）——随机获取数组的索引值

4、数组的遍历

- for（）——一般用于遍历数值数组

- foreach（）——一般用于遍历关联数组，也可以用于遍历数值数组

\* foreach（$arr as $item）——遍历数值数组，$item代表数组的每一项

\* foreach（$arr2 as $key=>$val）遍历关联数组，$key代表数组每一项的键，$val代表数组每一项的值

5、数组排序（这个对于前端来说用的比较少）

- sort() 对数组进行升序排列

- rsort() 对数组进行降序排列

- asort() 根据关联数组的值，对数组进行升序排列

- ksort() 根据关联数组的键，对数组进行升序排列

- arsort() 根据关联数组的值，对数组进行降序排列

- krsort() 根据关联数组的键，对数组进行降序排列

**二、ajax**

\*\*\* 关键点在于XMLHttpRequest异步**请求对象——谨记，这里创建的是一个对象**

1. ajax基本请求步骤：**关键在于创建异步请求对象XMLHttprequest**
2. 创建异步请求对象 （目的，是为了后面与服务器进行连接，从而获取到服务器的参数）

var xhr=new XMLHttpRequest（）；

1. **建立与服务器的连接，设置请求参数** open（type，url，async）

Eg：xhr.open("get", "http://localhost/api/ajaxtest", true);

**（备注：这一步是为了方便前端的数据和后端的数据进行更进一步的联系）**

- type 请求类型：get post

**get&post区别：**

\* get——得到；post——发送

\* get请求的数据通过url传递给服务器，post请求的数据保存在request body发送（send（））；

\* get发送的数据有长度的限制，post没有限制

\* get发送的数据暴露在url中，所以安全性比post请求低

\* get请求：路径?属性名=属性值&属性名=属性值

- url 请求路径（get请求的数据必须接在url后面）——传输你想要的数据给后端，从而做分析判断

Eg：xhr.open("get", "http://localhost/api/getdata.php?type=get&qty=10", true);

\* 相对路径：必须在同一个根目录下

\* 绝对路径：同源策略

同域（同源策略）：协议，域名，端口三者一致

\* 报错： No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is therefore not allowed access. 说明跨域了

\* async 是否异步，默认为true（异步）

\* 同步：按步骤顺序执行，前面的代码执行完后，后面的代码才会执行，做完前一件事情，才能下一件事情（排队）

\* 异步：与其他操作同时执行，也叫并发（图片加载，ajax请求，定时器）；

1. 发送请求 xhr.send（）

- post：send（**data（这里传递参数）**）——post请求，**参数为你想要传递的数据**

- get：send（null）——get请求，为null

1. 等待响应数据解析完毕，处理服务器返回数据

- 判断准备阶段xhr.readyState==4，一般是等待解析完毕后，才开始处理返回的数据的

- 判断http响应状态 xhr.status==200 || xhr.status==304

\* 响应的HTTP 状态。

200（OK）：服务器成功返回了页面

304（Not Modified）：数据与服务器相同，不需要重服务器请求（直接使用缓存的数据）

400（Bad Request）：语法错误导致服务器不识别

401（Unauthorized）：请求需要用户认证

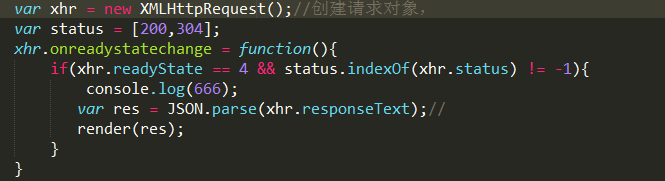
404（Not found）：请求地址不存在

500（Internal Server Error）：服务器出错或无响应

503（ServiceUnavailable）：由于服务器过载或维护导致无法完

- 获取服务器返回值 xhr.responseText

**处理服务器返回的数据，如下代码，一般放在创建异步请求对象的后面**



1. **php本地数据操作**
2. 获取前端数据

- $\_GET：获取前端用get方式传递过来的数据（url地址后面的数据也能被获取）

- $\_POST：获取前端用get方式传递过来的数据

- isset()：判断某个值是否被设置，若不存在返回boolean

1. 文件的读取与写入

fopen（path，mode）：打开文件

- 使用fopen函数打开文件时，你首先需要明确打开文件的模式：

1. 将数据写入文件，亦或者读写文件
2. 考虑如果文件中已经存在相关数据，你是覆盖原有文件中的数据呢，还是仅仅将新数据添加至文件末尾

- 文件模式：

\* r：以只读方式打开文件，从文件头开始读

\* r+：以读写方式打开文件，写入时以追加的方式写入文件

\* w：以写入方式打开文件，从文件头开始写。文件不存在则尝试创建，若存在，则先删除文件中的内容

\* w+：以读写方式打开文件，从文件头开始读写。文件不存在则尝试创建，若存在，则先删除文件中的内容

\* a：以写入方式打开，从文件末尾开始追加写。如果文件不存在则尝试创建

\* a+：以读写方式打开，从文件末尾开始追加写入或者读。如果文件不存在则尝试创建。

**- fread(file,length)：读取内容**

**fwrite(file,json字符串)：写入内容**

**fclose(file)：关闭文件,避免资源占用**

**filesize(path)：读取文件字符长度**

**读取写入文件的步骤如下：**

**先找到文件——读取整个文件的长度——关闭文件，这里已经得到了文件了——接着，以写入的方式打开文件——开始写入文件（这里的话原来的文件的内容已经发生了改变）——关闭文件夹，减少资源的占用**

**Eg：** 1、以读取模式打开文件

$path = './data/weibo.json'

$myfile = fopen($path, 'r');

2、读取文件内容

$content = fread($myfile, filesize($path));

3、关闭文件，减少资源占用

fclose($myfile);

- **echo返回数据**

\* json\_encode(); 把数组转成字符串

php5.4+ 使用JSON\_UNESCAPED\_UNICODE防止中文转义

\* json\_decode(json,assoc); 把字符串转成数组/对象

json：待解码的 json string 格式的字符串

assoc：默认false,返回嵌套对象的多维数组，通过arr->key获取对象键对应的值。当该参数为 true 时，将返回嵌套关联数组的多维数组，通过arr[key]获取关联数组键对应的值。

1. **ajax跨域解决方案**

**JSONP（主要集中在<script src=”地址”></script>）**

\* JSONP 是JSON with padding（填充式JSON 或参数式JSON）的简写。

\* JSONP是一种可以绕过浏览器的安全限制，从不同的域请求数据的方法。

\* **JSONP请求不是ajax请求，是利用script标签能加载其他域名的js文件的原理，来实现跨域数据的请求**

\* JSONP的局限性：

- 只能为get请求

- 接口必须有回调函数的执行

1. **CORS**

CORS是一个W3C标准，全称是”跨域资源共享”（Cross-origin resource sharing），**它允许浏览器向跨源服务器发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。**

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，**IE浏览器不能低于IE10**。服务器需要设置响应头，允许该域名访问。

1.Access-Control-Allow-Origin

header('Access-Control-Allow-Origin: \*');

该字段是必须的。需要在后端响应头添加词字段，值要么是一个\*，表示接受任意域名的请求，要么指定一个域名http://localhost。

2.Access-Control-Allow-Methods

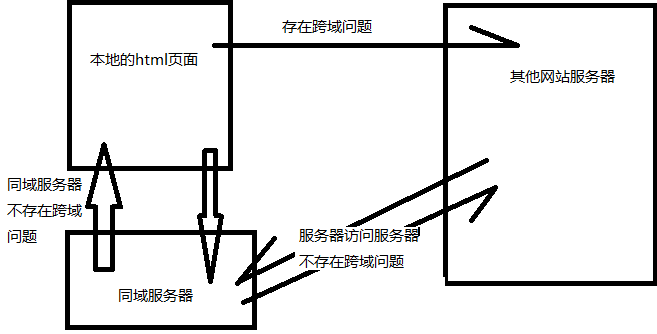
3.Access-Control-Allow-Headers

header('Access-Control-Allow-Methods:POST');

header('Access-Control-Allow-Headers:x-requested-with,content-type');

1. **服务器代理**

原理：后端不存在跨域的问题，所以可以利用后端间接获取其他网站的数据；



**（一）ajax跨域请求之服务端代理（爬虫）**

原理：获取页面所有内容，并利用正则匹配所需内容

\* file\_get\_contents($url) //获取网站内容

\* preg\_match\_all($reg,$str,$res)

\* preg\_match($reg,$str,$res) //$str代表被匹配的内容，$res值为数组[整个匹配的结果，分组匹配的结果]

\* $content = iconv(原字符编码,新字符编码,$content);//修改$content字符编码

总结：ajax跨域解决方案

1. JSONP

原理：利用script标签能加载其他域名的js文件的原理，来实现跨域数据的请求

局限性：只能为get请求；接口必须有回调函数的执行

1. CORS

原理：允许浏览器向跨源服务器发出XMLHttpRequest请求，从而克服了ajax只能同源使用的限制。

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。服务器需要设置响应头，允许该域名访问。

1. 服务器代理：后端不存在跨域问题，所以可以利用后端间接获取其他网站的数据。
2. **Promise（用来传递异步操作的数据）**

概念：

Promise是一个构造函数，所谓的Promise对象，就是通过new Promise（）实例化得到的对象，用来传递异步操作的数据。它代表了某个未来才知道结果的事件（通常是一个异步操作），并且这个事件提供统一的API，可供进一步处理。

1. Promise 传递异步操作的数据

var p=new Promise（function（resolve,reject）{

resolve（）//执行第一个参数resolve（），相当于执行p.then（）里面的函数

reject（）//执行第二个参数reject（），相当于执行p.catch（）里面的函数

}）——利用的是回调函数的原理：回调函数

eg：Promise相关的内容如下：



**静态方法**

**Promise.all([p1,p2,p3...])**

\* 将多个Promise实例，包装成一个新的Promise实例

\* 所有参数中的promise状态都为resolved时，新的promise状态才为resolved

\* 只要p1、p2、p3..之中有一个被rejected，新的promise的状态就变成rejected   
 Promise.all([p1,p2,p3]).then(function(res){  
 console.log(res);  
 })

**Promise.race([p1,p2,p3...]) // 竞速，完成一个即可**

1. **try...catch**

尝试执行代码，如果有错误则执行catch捕获错误(不报错)

try{  
 console.log(666)  
 //先尝试执行这里的代码  
 show();  
 //无报错，则忽略catch  
 //如果报错，则执行catch，并传递错误信息  
 }catch(error){  
 console.log(error)  
 }  
 console.log('end');===>最终的结果

**Day21——MySQL**

**一、数据库**

（一）概念

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库

（二）数据库分类

1、热门数据库 Oracle、SQLServer、MySQL、MongoDB、SQLite、access、DB2

2、结构化的数据：我们把一段复杂的信息，进行了结构化处理，瞬间显得清爽多了。计算机处理起来也相当方便。 因此，这样的数据，我们就称之为结构化数据

3、关系型数据库：我们把用户账号的信息，和消费记录信息分离开来，通过用户编号进行了关联 既把不同的数据进行了分离，使得查询数据更加方便。同时又保证了它们关系的正确性。



这就是传统的关系型数据库。 其中 <用户编号> 我们称之为外键

（三）表(table)

一个数据库通常包含一个或多个表，一个数据表有一个唯一名称，并有行和列组成。

表是数据库中一个重要的组成部分, 数据库只是一个框架，数据表才是其实质内容。

（四）MySQL数据库

特点：

\* MySql是目前最流行的关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司

\* MySQL是开源的，免费。

\* MySQL支持标准的SQL数据语句

\* MySQL可以允许于多个系统上，并且支持多种语言。这些编程语言包括C、C++、Python、Java、 Perl、PHP、Eiffel、NodeJS、Ruby和Tcl等。

\* MySQL对PHP有很好的支持，PHP是目前最流行的Web开发语言。

\* MySQL支持大型数据库，支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB， 64位系统支持最大的表文件为8TB

（五）命令窗口数据库操作

1、连接数据库  
 \* 格式：mysql -h主机地址 -u用户名 -p用户密码

2、显示所有数据库

\* 格式：show databases;

3、创建数据库

\* 格式：create database <数据库名>;

4、使用数据库

\* 格式：use <数据库名>;

5、显示当前数据库所有表

\* 格式：show tables;

6、创建数据表

\* 格式：create table <表名> (<字段名1> <类型1> [,..<字段名n> <类型n>]);

create table MyGuests (

id int(6) unsigned auto\_increment primary key,

firstname varchar(30) not null,

lastname varchar(30) not null,

email varchar(50),

reg\_date timestamp

)

\* 以上创建一个名为 “MyGuests”的表，包含5个列：“id”,“firstname”,“lastname”, “email”和“reg\_date”，参数如下：

·int (整形)

·float (浮点型)

·char (固定长度字符串)

·varchar (可变长度字符串)

·blob (二进制)

·text (字符串)

·timestamp (时间戳)

\* 列的其他属性

·not null (每一行都含有值，不能为空，null值是不允许的)

·default value (设置默认值)

·unsigned (使用无符号数值类型，0及正数)

·auto\_increment (设置 MySQL 字段的值在新增记录时每次自动增长 1)

·primary key (设置数据表中每条记录的唯一标识。 通常列的 PRIMARY KEY 设置 为ID数值，与auto\_increment一起使用)

7、删除表

\* 格式：drop table <表名>;

drop table MyGuests;

drop table if exists MyGuests

8、查询表结构：

\* desc MyGuests

9、修改表名

\* rename table MyClass to YouClass;

(10) 增加字段

\* alter table MyGuests add sku\_id bigint(20) unsigned DEFAULT NULL COMMENT '商品 销售码';

(11) 复制表结构

\* create table table1 like table;

**二、Navicat操作数据库——这个是重点中的重点**

\* 数据库创建

\* 表创建

\* 增删查改（crud）

\* 数据导入

· json

· excel

（一）数据操作——增删改查，**增-insert，删-delete，查-select，改-update**

1、插入数据——insert插入数据  
 格式：insert into <表名> (<字段名1>[,..<字段名n > ]) values (值1 [,(值n)]);

例： insert into results (username,gender,team) values ("谢文东","男",5)

备注：字段与值需要一一对应

2、删除表数据——delete删除数据  
 格式：delete from 表名 where 表达式

例： delete from results where id=5 //删除results表中id为5的数据

delete from results //删除results表中所有数据

3、查询表中的数据——select查询数据

格式：select <字段1, 字段2, …> from < 表名 > where < 表达式 >

例： select \* from results //查看表results中的所有数据，用 \* 表示所有的内容

select \* from results order by id limit 0,4;//查看表results前4行数据

备注：select一般配合where使用，以查询更精确更复杂的数据

4、修改表中的数据——update修改数据  
 格式：update 表名 set 字段=新值,… where 条件;

例：update results set username='黄晓明' where id=2;

（二）条件控制语句

where语句

\* select \* from results where id=1；//查询id为1的所有数据

相关条件控制符——**便于精确处理数据**

\* = 例：select \* from results where username = '范冰冰'

\* > 例：select \* from results where team > 4

\* < 例：select \* from results where team < 4

\* <> (不等于) 例：select \* from results where team <> 5

\* in(1,2,3...) 例：select \* from results where id(字段) in(1,2)

\* between a and b 例：select \* from results where team between 3 and 8

\* and 例：select \* from results where team=5 and username='谢文东'

\* or 例：select \* from results where team=5 or username='岳云鹏'

\* not 例：select \* from results where not username='范冰冰'

\* like %匹配任意字符； \_匹配一个字符(可以是汉字)

例：select \* from results where username like '张%'

\* limit id,qty：数量控制

例：select \* from results limit 1,4

\* is null (空值检测)

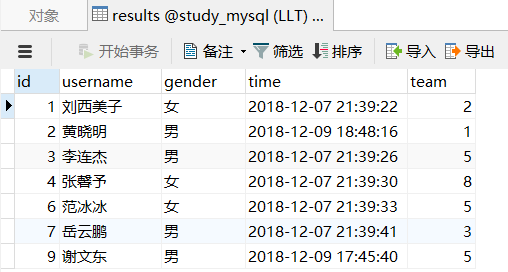
例：select \* from results where team is null

\* order by

\* asc 升序(默认)

\* desc 降序

例：select \* from results order by team asc



（三）PHP操作数据库

1、连接数据库

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = 'user';

$conn = new mysqli($servername, $username, $password, $dbname); // 创建连接

// 检测连接

if ($conn->connect\_error) {

die("连接失败: " . $conn->connect\_error);

}

$conn->set\_charset('utf8'); //查询前设置编码，防止输出乱码

echo "连接成功";

2、书写语句，执行语句

\* 执行语句的代码：$res = $conn->query()

\* 若执行的是 **增删改** 语句,$res的值为布尔值(insert, update, delete)

\* 若执行的是 **查询** 语句，得到查询结果集(对象)(select)

\* num\_rows 结果集中的数量，用于判断是否查询到结果

\* fetch\_all(MYSQLI\_ASSOC) 得到所有结果

\* fetch\_assoc() 得到第一个结果

\* fetch\_row() 得到第一个结果,只能拿到值

\* 若是查询语句，记得释放查询结果集，避免资源浪费

\* $res->close();

3、关闭数据库

\* $conn->close();

备注：调用完数据库的内容后，最后记得需要关闭数据库

（四）导入数据

1、先在数据库建立好表，表的数据类型及编码都先定义好。再从其他文件导入进来

2、转储及运行数据库

3、先创建同名的数据库，右键运行数据库，重新进入界面才会看到

select \*from (select \*from main\_goods where main\_goods.goods\_id in (select goods from user\_goods where user\_name = "xiaogui" ))a

left join (select goods,qty from user\_goods where user\_name = "xiaogui" )b

on a.goods\_id =b.goods

main\_goods 是商品表

user\_goods购物车表

**Day22——面向对象**

1. **创建对象及使用**
2. 创建对象
3. 对对象进行描述

\* 有什么（属性）

\* 能做什么（方法）

1. 指挥对象做事情
2. **对象的创建方式**
3. 字面量{}

Eg：var student = {id:10,name:'小明',age:18}

一般只用于创建单一对象

1. new Object（）

一般用于创建单一对象

Eg：var student = new Object()

student.id = 10;

student.name = '王铁锤';

student.age = 18;

\* 工厂模式：将参数传入函数内，函数内生成对象，给对象设置属性及方法（具体值来自传入的参数），最后将对象通过return返回出去。

\* 缺点：没有解决对象识别的问题（对象的类型都是Object）。——**无法知道这个对象是谁创建的**。

1. **自定义构造函数创建对象（重点）**

\* 自定义构造函数与普通函数定义没有区别，唯一不同的是执行方式不同

\* 函数名（）==>普通函数，若没有return，则返回值为undefined

\* new 函数名（）==>构造函数

执行new构造函数时的四大步骤：

\* 1、在构造函数内声明了一个对象O；

\* 2、将构造函数的this指向上面声明的对象O；

\* 3、给this绑定属性及方法，相当于给对象O绑定同样的属性及方法；

\* 4、将this对象返回出去。

\* 优点：

\* 解决了重复创建多个同一类型对象的问题

\* 解决了对象的识别问题

\* 不成文的约定：构造函数定义时，首字母大写

4、this函数的当前对象：（**记住只有方法才能调用到this**）

（1）函数自执行：this指的是执行函数的对象

\* 一般省略了对象调用的函数，this指的是window

\* 对象.函数（），this指的是前面的对象

（2）事件驱动函数：this指的是函数绑定的元素对象

（3）new 构造函数（）的**this指的就是当前实例对象**

1. **对象的组成部分**
2. 构造函数 function

函数名（）{通过this定义属性及方法}

1. 实例对象

**\* new 构造函数（） 产生的对象称为实例对象**

\* 实例对象能拷贝构造函数的所有属性及方法

\* 实例对象可以使用其原型对象的所有方法

1. **原型对象：创造实例对象的那个对象就称为原型对象（构造函数.prototype）**

\* 原型对象都拥有一个constructor属性（构造器），指向构造函数

\* 构造函数通过prototype属性指向原型对象

\* **实例对象**通过\_\_proto\_\_指向原型对象

**原型对象.constructor——>构造函数**

**构造函数.prototype——>原型对象**

**实例对象.\_\_proto\_\_——>原型对象**

1. **实际运用**
2. 解决方案：构造函数+原型对象

- 使用构造函数添加私有属性

- 使用原型对象添加共享方法

**备注：属性一般都是私有的，但是方法的话可以是公用的，一般大家都可能用到相同的方法。**

1. 优点：

- 实例对象都有自己的独有属性

- 实例对象共享了原型中的方法，最大限度的节省了内存

- 支持向构造函数传递参数

1. **判断原型对象和实例对象的关系**
2. constructor：一般用于判断该实例对象是否由某一构造函数生成

实例对象.constructor==构造函数 // 返回布尔类型的值

1. instanceof：检测某个对象是不是某一构造函数的实例，适用于原型链中的所有实例

实例对象 instanceof 构造函数 //返回布尔类型的值

1. isPrototypeOf：判断当前对象是否为实例对象的原型对象

原型对象.isPrototypeOf（实例对象） //返回布尔类型的值

**Day23——原型和继承**

1. **属性特性：ES5对象扩展（了解）**
2. 值属性的属性特性
3. configurable

\* 可配置性，控制着其描述的属性的修改，表示能否修改属性特性

1. enumerable

\* 可枚举性，表示能否通过for-in遍历得到属性

1. writable

\* 可写性，表示能否修改属性的值

1. value

\* 数据属性，表示属性的值。默认值为undefined

1. 与属性特性相关的方法
2. 设置属性特性：

\* 通过 . 或者[]给对象添加的属性，拥有的属性特性默认值都为true（除了value为具体的值）

\* Object.definePrototype（obj，prototype，descriptor）给对象的某个属性设置属性特性

\* 拥有的属性特性默认值为false（除了value为具体的值）

\* Object.defineProperties（object，descriptors）给对象的所有属性设置属性特性

1. **重置原型对象（理论上比较常用）**
2. 重置原型对象，可以一次性给原型对象添加多个方法，但切断了与原来原型对象的联系

Eg：

function Popover(){}——构造函数

Popover.prototype = {

show:function(){},

hide:function(){}

}

//- 注意覆盖问题（系统的原型对象不建议重写）

//- 注意识别问题（不可枚举性）

Object.defineProperty(Popover.prototype,"constructor",{value:"Popover",configurable:true})；

1. **原型链**

1、实例对象与Object原型对象之间的链条称为原型链

1. **原型模式的访问机制（原型搜索机制）**

1、读取实例对象的属性时，先从实例对象本身开始搜索。如果在实例中找到了这个属性，则返回该属性的值；

2、如果没有找到，则继续搜索实例的原型对象，如果在原型对象中找到了这个属性，则返回该属性的值

3、如果还是没找到，则向原型对象的原型对象查找，依此类推，直到Object的原型对象（最顶层对象）；

4、如果再Object的原型对象中还搜索不到，则抛出错误；

实例对象属性——>其原型对象——>原型对象的原型对象——...>Object的原型对象

1. **继承**

（一）概念：继承是面向对象中一个非常重要的特征。指的是：子类继承父类的属性和方法。

\* 我们可以通过继承的方式，在父类的属性和方法基础上，让子类也拥有这些属性和方法，并可以扩展。

（二）继承的好处：

\* 子类拥有父类所有的属性和方法（代码复用）；

\* 子类可以扩展自己的属性和方法（更灵活）；

\* 子类可以重写父类的方法。

（三）继承方式

**（备注：bind（）改变函数中的this的指向，返回新的函数；call（）改变函数中this的指向，同时执行该函数）**

**组合式继承**

1. 原型链继承：子类的原型对象指向父类的实例对象

\* 缺点：无法直接在实例化子类的同时，给父类的属性赋值（无法操作父类构造函数中的属性）；

1. 借用构造函数：将父类的构造函数拿到子类的构造函数中，改变this的执行，同时执行该函数，记得传参。——**相当于把父类的实例属性复制一份给子类实例**

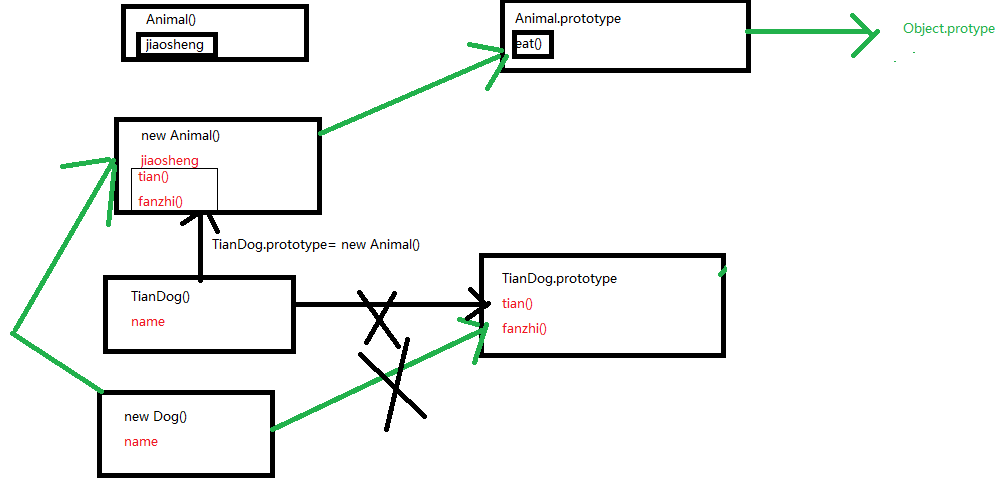
\* 解决了子类拷贝父类的所有属性的问题；

\* 缺点：父类存在多余的属性

**组合式继承：原型链继承方法+借用构造函数拷贝属性；如下图所示：**

**缺点：父类存在多余的属性**

缺点：



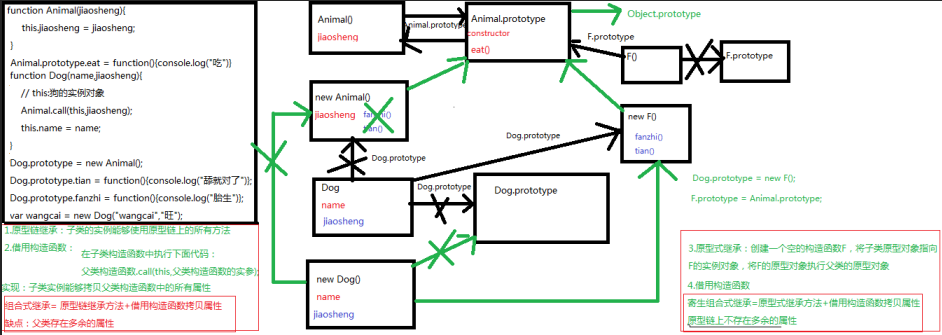
**寄生组合式继承**

1. 原型式继承：创建一个空的构造函数F，将子类原型对象指向F的实例对象，将F的原型对象指向父类的原型对象
2. 借用构造函数

**寄生组合式继承：原型式继承方法+借用构造函数拷贝属性；**

**原型链上不存在多余的属性**

**如下图所示：**



（四）ES6继承的方法

1、class定义类

\* 写在类里面的方法实际是给Person.prototype添加方法

\* constructor方法是类的默认方法，相当于在构造函数内生成属性

2、extends继承

\* 子类继承了父类，在子类构造函数中必须调用super方法。

\* 子类的constructor方法没有调用super之前，不能使用this关键字，否则报错，而放在super方法之后就是正确的。

3、静态方法

\* 如果在一个方法前，加上static关键字，这就称为“静态方法”

\* 静态方法方法不会被实例继承，而是直接通过类来调用Person.getInfo()

\* 父类的静态方法，可以被子类继承Man.getInfo()

Eg：  
class Person{  
   constructor(name,age,gender){//这里写构造函数的属性  
       this.name = name;  
       this.age = age;  
       this.gender = gender;  
   }  
   // 此处的方法写入原型对象中，此处写原型对象的方法  
   say(){console.log(大家好,我叫${this.name})}；  
   static eat(){console.log('我是吃货');}  
}  
let lemon = new Person('lemon',17,'女');  
// 继承方法  
class Man extends Person{  
   constructor(name,age,gender){  
       // 继承属性  
       super(name,age,gender);  
       this.hobby = ['吃','喝'];  
   }  
}  
let laoxie = new Man('laoxie',48,"男");

（五）call和apply和bind方法

\* bind（）——改变this的指向，返回新的函数

\* call（）——改变this的指向，并且执行函数

\* apply（）——改变this的指向，并且执行函数，函数的参数变为了数组

实际应用：

1. 当自己的方法没有的时候借用别人的方法；

\* 借用其他构造函数的原型对象中定义的方法；

\* 自己本身存在某个方法，但想使用原型链上其他原型对象的某个方法；

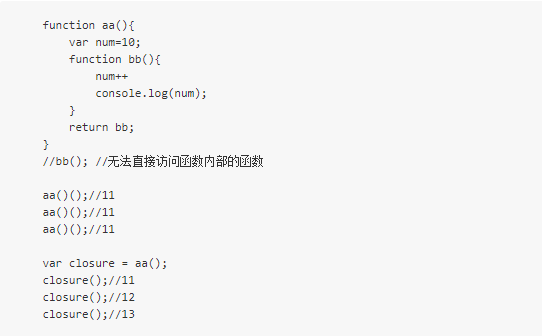
1. 利用apply方法，可以将参数转成数组传入；
2. **闭包**
3. 概念：

闭包是这样一种机制：**函数嵌套函数，内部函数可以引用外部函数的参数和变量**，参数和变量不会被垃圾回收机制所回收。

以上涉及到几个概念：

1. 函数嵌套函数
2. 作用域（全局变量和局部变量）——变量的访问规则（就近原则）
3. 垃圾回收机制——js内部不断扫描内存，并清理无引用对象（自动完成）;

eg：代码如下：



2、闭包的好处

（1）可以让一个变量长期驻扎在内存当中不被释放

（2）避免全局变量的污染，和全局变量不同，闭包中的变量无法被外部使用

（3）私有成员的存在，无法被外部调用，只可以自己内部使用

结论：

\* 闭包是指有权访问另一函数作用域中的变量的函数

\* 闭包，可以访问函数内部的局部变量，并让其长期驻留内存

\* 由于闭包会携带包含它的作用域（运行环境），因此会比其他函数占用更多的内存，过度使用闭包可能会造成性能问题。

**Git pull... Before pushing again——这个是个报错的信息，表明先要git pull一下才行**

**Day24-j Query**

1. **jQuery的下载与安装**

1、官网下载  
 \* <http://jquery.com/download/>

2、CDN

\* <https://code.jquery.com/jquery-3.0.0.js>

\* <https://code.jquery.com/jquery-3.0.0.min.js>

\* <https://www.bootcdn.cn>——这个比较常用

**二、jQuery实现原理**

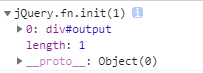
\* jQuery构造函数

\* jQuery实例对象

eg：jQuery（“#box”）;//得到jquery对象（实例）

**jQuery对象只能用jQuery的方法——这个是重点**

1、jQuery实例属性如下：



\* length：返回jQuery对象中匹配元素的个数

\* jQuery：当前jquery类库版本号（一般用于判断是否是jQuery对象）

\* 下标（索引）：DOM节点

1. jQuery的别名：$
2. 延迟代码执行：jQuery（document）.ready（fn）——**这个表示DOM树渲染完成时执行**

\* 此方法传入一个匿名函数；

\* 页面DOM树渲染完成时执行；

\* 简写方式：jQuery（function（）{}）；

\* 安全使用$：jQuery（function（$）{}）；

1. **编写jQuery代码只需两步——1、选择元素；2、操作元素**
2. 获取jQuery对象 jQuery（选择器 | dom节点，上下文对象）==>$（选择器 | dom节点，上下文对象）

\* 上下文对象可以是css选择器、jQuery对象

\* **jQuery对象只能用jQuery的方法**

\* jQuery对象得到的是类数组

\* length 若length为0，说明获取不到该jQuery对象

\* jquery 得到对应的版本号，用于判断是否为jquery对象

\* [索引] 获取到dom节点（对象）

1. dom节点与jQuery对象的转换

\* jQuery对象转成dom节点

\* [索引]—只要运用索引就行了

\* dom节点转成jquery对象

\* jQuery（dom节点）

**三、jQuery的选择器**（选择页面中的元素，得到jQuery实例对象）

1、ID选择器 $(“#save”)

2、类选择器 $(“.class”)

3、标签选择器 $(“div”)

4、复合选择器 $(“div,span,p.myClass”)

5、属性选择器 $(‘[id=box]’)

\* $(‘li[data-index]’)：获取有data-index属性的所有元素

\* $(‘li[data-index!=10]’)：data-index属性不等于10的元素，css目前未支持

\* $(‘li[data-index^=10]’)：data-index属性以10开头的元素

\* $(‘li[data-index$=10]’)：data-index属性以10结尾的元素

\* $(‘li[data-index\*=10]’)：data-index属性包含10的元素

6、表单选择器$(‘:input’)

\* :radio //匹配所有单选按钮

\* :checkbox //匹配所有复选按钮

\* :selected //获取已选择的option元素

\* :checked //匹配所有被选中的元素(复选框、单选框等，select中的option)

\* :submit //匹配所有提交按钮

\* :reset //匹配所有重置按钮

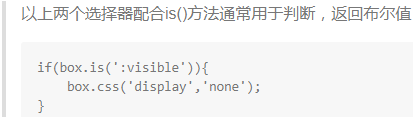
\* :button //匹配所有按钮

\* :text //匹配所有的单行文本框

\* :password //匹配所有密码框

7、可见性  
 \* :hidden //匹配所有不可见元素(display:none)，或者type为hidden的元素（这里指的是input type=hidden）  
 \* :visible //匹配所有可见元素

Eg：



8、基本筛选（利用选择器得到的结果不一定是我们想要的最终结果，有时需要进一步筛选）

\* :odd/:even,:gt(n)（这个表示从左往右）/:lt(n)（这个表示从右往左） //筛选范围（索引支持负数）

\* :contains(奥巴马) //筛选出包含“奥巴马”这三个字的元素

**四、jQuery对象的常用操作方法**

1、jQuery对象与原生对象的转换

\* jquery转原生DOM节点：

\* get（0）/[0]获取集合中第一个DOM节点

\* get（）不传参得到集合中所有的DOM节点

\* 原生转jQuery：$（dom）

2、判断是否为jQuery对象

\* var box = $('#box');

if(box.jquery){

}

3、判断一个jQuery对象是否存在（是否能获取到元素）

\* length 若是length为0，则表示没有获取到元素

\* 转成原生对象再判断，若是jQuery对象存在，这能转成原生对象

**五、jQuery的筛选方法（expr | obj | ele——表示 “选择器 | jQuery对象 | DOM节点”**

1、筛选的方法——这个是方法，需要用 . 来调用

\* first() / last()：获取集合中第一个/最后一个元素

\* eq(index | -index)：获取第N个元素,n支持负数（表示从后面查找）

\* filter(expr | obj | ele | fn)：缩小匹配范围，将匹配到的jQuery对象进行过滤，得到过滤后的jQuery对象——**表示选中filter括号中的，同级**

\* map(fn)：将一组元素转换成其他数组（不论是否是元素数组）

\* slice(start,[end])：选取一个从start到end(不包括end)匹配的子集

**\* has(expr | ele)：保留包含特定后代的元素，去掉那些不含有指定后代的元素。**

\* not(expr | ele | fn)：缩小匹配范围，将被匹配到的jQuery对象进行过滤，除去过滤后的jQuery对象——表示选中除了not括号中的内容

2、查找的方法（利用当前元素去查找其他元素），传的参数可以是

（1）查找子元素

\* find(expr | obj | ele)：查找后代元素

\* children([expr])：取得匹配元素的所有子元素。(原生js:children)

（2）查找父级元素

\* parent([expr])：获取父元素

\* parents([expr])：取得所有父级元素

\* closest(expr | obj | ele)：从元素本身开始，逐级向上级元素匹配，并返回最先匹配的元素

\* offsetParent()：返回第一个有定位属性(absolute,relative,fixed)\* 的父元素,如果没有定位父级，则返回html元素

（3）查找兄弟元素，[ ]——表示这个可以省略

\* next([expr])：返回下一个同辈元素 ==> nextElementSibling

\* prev([expr])：获取前一个同辈元素 ==> previousElementSibling

\* nextAll([expr])：获取当前元素之后所有的同辈元素

\* prevAll([expr])：获取当前元素之前所有的同辈元素

\* siblings([expr])：获取当前元素的所有兄弟元素（除自身以外的所有兄弟元素 = \* prevAll + nextAll）

1. **DOM节点操作——增删改**
2. 创建jq对象

$(“<div/>”)；

$(“<div>生成一个div</div>”)

1. 元素添加
2. 往父元素内部追加

\* append（content | obj | ele | fn）：给父元素添加最后一个子元素

\* prepend：给父元素添加第一个子元素

\* appendTo（父元素） prependTo（父元素）

1. 外部插入内容（添加为兄弟元素）

\* after：在元素后面插入内容

\* before：在元素前面插入内容

\* insetAfter（兄弟元素） insertBefore（兄弟元素）

3、元素删除

（1）remove（）：删除元素；

（2）empty（）：清空内容；

**4、元素复制**

\* clone（[Event[,deepEvent]]）;

\* Event：（true或false）是否复制元素的行为，默认为false；

\* deepEvent：（true或false）是否复制子元素的行为，默认为Event的值

备注：clone（）与 父元素.cloneNode（）的区别

\* 父元素.cloneNode（），当括号中的值为true时，表示深复制克隆元素里面的所有内容，但是不会复制元素的行为，clone（）当里面的条件为true时，表示会复制元素的行为——重点知道下；

**七、jQuery动画** duration——表示动画时间

一、基本动画效果

1、显示与隐藏：show（）/ hide（）

\* show（duration）：**显示** 通过改变元素的高度、宽度和不透明度，直至内容完全可见；

\* hide（duration）：**隐藏** 通过改变元素的高度、宽度和不透明度、直到这三个属性值为0；

2、滑动（通过改变高度）

\* slideDown（duration，fn）

\* 显示元素

\* 不断改变高度，直到样式内设定的值

\* slideUp（duration，fn）

\* 不断改变高度，直到0

\* 隐藏元素

\* slideToggle（duration，fn）

\* 当元素隐藏时调用slideDown（），当元素显示时调用slideUp（）

3、淡入淡出（通过改变不透明度）

\* fadeIn（duration，fn）

\* 显示元素

\* 不断改变透明度直到1

\* fadeOut（duration，fn）

\* 不断改变透明度直到0

\* 隐藏元素

\* fadeToggle（duration，fn）

\* 同上slideToggle（duration，fn）

\* fadeTo（duration，opacity，fn）

\* 不断改变透明度opacity，直到设定的值，并在动画完成后可选地触发一个回调函数

二、自定义动画

1、animate（params，[speed]-动画执行的时间，[fn]）

Eg：$("#block").animate({

width: "90%",

height: "100%",

fontSize: "10em",

borderWidth: 10

}, 1000 );

2、:animated

\* 获取正在执行动画的元素，一般与is（）方法配合使用，用于判断元素是否处于动画状态

三、动画队列

1、一个元素上的动画：

\* 当animate中存在多个属性时，动画同时发生

\* 当同一个元素链式调用animate时，动画是按顺序发生（队列）

2、不同元素上的动画：

\* 默认情况下，动画同时发生

\* **回调函数内的动画等到当前动画执行完毕后才接着执行**

3、stop（[clearQueue]，[jumpToEnd]）

\* 不加参数：停止当前元素所有《正在运行》的动画。——这个常用

\* clearQueue:值为true时，清除队列

\* jumpToEnd:值为true时，跳到当前动画的最后一帧

4、delay(duration)  
 \* 设置一个延时来推迟执行队列中之后的动画。

\* duration:延迟的时间

**八、事件**

一、常用事件方法

1、鼠标事件

\* click([[data],fn]) //点击时触发 click = mousedown + mouseup

\* dblclick([[data],fn]) //双击事件 dblclick = 2\*click

\* mousedown([[data],fn])

\* mouseup([[data],fn])

\* mousemove([[data],fn])

\* mouseout([[data],fn])

\* mouseover([[data],fn])

\* mouseenter([[data],fn]) //事件不会冒泡

\* mouseleave([[data],fn]) ) //事件不会冒泡

2、键盘事件

\* keydown([[data],fn]) //键盘按下时触发

\* keypress([[data],fn])//字符按键

\* keyup([[data],fn]) //键盘弹起时触发

3、表单事件

\* blur([[data],fn]) //失去焦点时触发

\* focus([[data],fn]) //获得焦点

\* change([[data],fn]) //值改变并失去焦点时触发

\* submit([[data],fn])

4、其他事件

\* resize([[data],fn]) //元素大小改变时触发

\* scroll([[data],fn]) //滚动时触发

二、事件绑定与移除

1、事件绑定 on（type，[选择器selector]，fn）

\* selector： 把本来绑定给selector的事件委托给他的父级**（这里的fn的this指向被执行的selector）**

\* 可以一次性绑定多个事件，事件之间以空格隔开

\* 事件命名空间，自定义事件（对事件加以细分）

格式：事件类型 . 自定义名字



\* 支持自定义事件的绑定

$(ele).on('laowang',function(){})  
 触发自定义事件：$(ele).trigger(‘laowang’);

2、off：清除绑定事件

\* off(‘click’)；//清除当前元素的点击事件

\* off()；//清除当前元素所有事件

\* off(‘click mouseover’)；//一次性清除多个事件，事件之间以空格隔开

\* off(‘click.output’)；//清除命名空间事件

三、事件的其他方法（了解）

1、hover(enter[,leave])

\* enter:鼠标移入时执行

\* leave:鼠标移出时执行

备注：hover方法内部使用mouseenter + mouseleave来实现效果

2、trigger(type)：手动触发事件（即使事件没有发生，也能执行事件处理函数）,默认有浏览器默认行为和冒泡。

3、triggerHandler(type)：这个方法会触发指定的事件类型上所有绑定的处理函数。但不会执行浏览器默认行为，也不会产生事件冒泡

4、阻止浏览器默认行为  
 \* event.preventDefault();

5、阻止事件传播（阻止事件冒泡）  
 \* event.stopPropagation();

6、两者一起阻止：  
 \* return false;

**九、盒模型属性**

1、offset（）：获取匹配元素相当于根元素的偏移量

\* 返回一个对象，包含当前元素的top、left值

2、position（）：获取匹配元素相对（有定位属性）父元素的偏移量，如果没有定位父级，则相对于根元素（html），返回一个对象，包含当前元素的top，left值。——**这个方法不会获取margin的值**

3、width（val）=width；//取值 / 赋值，当传入v时，相当于css（‘width’，val）；

4、innerWidth（）=width+padding；<==>clientWidth

5、outerWidth（）=width+padding+border；<==>offsetWidth

6、outerWidth（true）=width+padding+border+margin；

**备注：盒模型的高度同上**

**十、jQuery的ajax方法**

1、$.ajax（settings）

\* type：请求类型，默认GET

\* url：数据请求地址（API地址）

\* data：发送到服务器的数据对象，格式：{Key：value}。

\* success：请求成功时回调函数。

\* dataType：设定返回数据的格式，json，jsonp，text（默认），html，xml，script

\* async：是否为异步请求，默认为true

2、$.get(url,[data],[fn],[dataType]) // type:’get’

3、$.post(url,[data],[fn],[dataType]) // type:’post’

4、$.getJSON(url,[data],[fn]) // type:’get’, dataType:’json’

5、$.getScript(url,[callback]) // type:’get’, dataType:’script’

6、load(url,[data],[callback]) 载入远程 HTML 文件代码并插入页面中。

**十一、插件**

1、了解插件

（1）什么是插件

\* 插件(Plugin)也称为jQuery的扩展。以jQuery核心代码为基础编写的符合一定规范的应用程序。通过js文件的方式引用。

（2）插件分类

\* UI类、表单及验证类、输入类、特效类、Ajax类、滑动类、图形图像类、导航类、综合工具类、动画类等等

1. 常用插件：

\* jqueryUI //官方插件，功能多且全面，官网：[www.jqueryui.com](http://www.jqueryui.com)

\* jquery.validation //表单验证插件,官网：<https://jqueryvalidation.org/>

\* jquery.easyUI //是一组基于jQuery的UI插件集合,Demo：<http://www.jeasyui.net/>

\* bootstrap //是最受欢迎的 HTML、CSS 和 JS 框架，用于开发响应式布局、移动设备优先的 WEB 项目，官网：<http://v3.bootcss.com/>

\* highcharts

<https://www.hcharts.cn>

1. **常用jQuery原型对象的方法——写在jQuery原型对象中的方法，通过jQuery实例调用**
2. css（attr[，val]）：获取 / 改变元素style属性（内联样式）

\* 取值：css（attr），css（[“class”，“text-align”]）<==>getComputedStyle[attr]

\* 赋值：css（attr，val），css（{attr：val}）；

1. val（v）：获取 / 设置匹配表单元素的值（等同于原生js中的value值）

\* 取值：

input.val（）

\* 赋值：

Input.val（v）

\* v：字符串

\* v：数组（一般用于单选 / 复选框的勾选）

\* v：函数function（idx，val）{return 值} //函数内部一定要有返回值

1. html（）：（等同于原生js中的innerHTML）

\* 取值div.html（）：取得第一个匹配元素的html内容

\* 赋值 div.html（）：设置匹配元素的内容

1. text（）：取得所有匹配元素的文本内容
2. addClass（）/ removeClass（）：添加 / 删除类，支持多个类同时添加或删除

\* toggleClass（）：如果存在（不存在）就删除（添加）类。

\* hasClass（“con”）：判断当前元素是否包含con这个类，返回布尔值（不支持多个类进行判断）

1. eq（n）：获取第N个jquery对象（元素），n支持负数（表示从后面查找）
2. index（）：获取当前元素在同辈元素中的索引值

\* $(this).index（）；

1. 显示 / 隐藏

\* show（）：显示元素

\* hide（）：隐藏元素

\* 带参数：同时改变width，height，opacity的动画

1. is（expr | obj | ele | fn）：根据选择器、DOM元素或jQuery对象来检测匹配元素集合，其中如果有一个元素符合这个给定的选择器表达式就返回true。如果没有元素符合，或者表达式无效，都返回false
2. attr（name[,val]）：设置 / 获取html标签属性
3. prop（attr[,val]）：设置 / 获取DOM节点属性（一般修改布尔类型属性）

\* 获取：获取在匹配的元素集合中的第一个元素的属性值。

\* 赋值： 给集合中所有元素属性赋值

\* val为函数

Eg：$(':checkbox').prop('checked',function(idx,oldVal){

return !oldVal;

}

1. each（function（idx，ele）{}）//用于遍历jQuery对象

\* return true；//跳过当前循环，进入下一循环（等效原生js中的continue）；

\* return false；//退出整个each循环（等效原生js中的break）；

**jQuery大部分方法的共性：**

1. 无参数时为取值，带参数时为赋值
2. 取值：取得第一个匹配元素的值
3. 赋值：设置所有匹配元素的值
4. 隐式迭代（隐式遍历）：看不见的遍历，大部分的jquery方法都支持隐式迭代
5. 设置方法：返回值为this，当前的实例对象；
6. 获取方法：返回值为获取到的值；
7. **常用jQuery静态方法**
8. $.each（arr | obj，callback）：通用遍历方法，用于遍历对象和数组

\* callback（idx，item）

1. $.map（arr | obj，callback）：根据现有数组生成一个新的数组，新数组的元素为callback内return的值

\* callback（item，idx）

1. $.type（n）：检测参数n的数据类型
2. $.makeArray（obj）：将类数组对象转换为数组。
3. $.parseJSON（json）：接收一个JSON字符串，返回解析后的对象或数组。类似于原生js中的JSON。parse（）
4. $.inArray（value，array，[fromIndex]）：确定value在数组array中的位置，从0开始计数（如果没有找到则返回-1），一般用于判断数组中是否包含某一个字符。
5. serialize（）/serializeArray（）：只能在form表单中使用，并且表单元素必须有name属性。

**Day25-sass**

1. **sass**
2. sass 语言解析器，提供嵌套、变量
3. gulp 自动化构建工具（想要实现对应的功能，就下载对应的包就行了）

\* sass->css（gulp-sass需要安装这个包才能实现sass转成css）

\* html压缩（gulp-htmlmin）

1.node环境-npm包管理工具-gulp

测试命令：

node -v

npm -v

2.下载gulp：

（1）为了任意位置都能执行gulp任务，gulp得全局安装 命令：npm install -g gulp

\* 全局包都在这个目录 C:\Users\Administrator\AppData\Roaming\npm;

（2）在当前项目根目录，需要再次下载gulp，目的为了引入gulp.js

\* shift+鼠标右键-在此处打开命令行

\* 命令：npm install gulp：重新下载gulp，局部安装gulp

3.为了实现sass->css

下载gulp-sass（想要实现sass->css功能的话，还需要在项目根目录下载gulp-sass）

在当前的项目根目录，下载gulp-sass，为了引入js

\* 命令：npm install gulp-sass

4、创建gulpfile.js文件——在项目根目录下创建  
 **在项目根目录下创建一个名为 gulpfile.js 的文件（重要）** 说明：gulpfile.js是gulp项目的配置文件，是位于项目根目录的普通js文件，内容如下：

Eg:

var gulp = require('gulp');

gulp.task('default', function() {

// 将你的默认的任务代码放在这

});

5、运行 gulp：  
在命令行执行以下命名，如果不写任务名称，则自动运行default任务（如果有）

gulp <任务名称>