**------------------------------行内式代码编写------------------------------**

<bottom onclick=”alert(‘hello world’)”></bottom>

onclick=”JavaScript代码”

alert(‘提示框’)

**------------------------------内嵌式代码编写------------------------------**

<script>

alert(‘你好世界’)；

</script>­­

**------------------------------外链式代码编写------------------------------**

<script scr=”/js/1.js”><script>

**-----------------------------------变量----------­­­­­­­­-------------------------**

<script>

var 变量名 = 99；

alert(变量名 \* 0.8)

</script>

------------------------------------------------

变量名命名规则：

1）首字母必须使用 字母，$，\_

2）变量名中间可以使用 字母，$，数字，\_

**---------------------------------数据类型---------------------------------**

1．数字类型：Number =============== var 变量名 = 123456 ；

2．字符串类型：String ============== var 变量名 = ‘Hello 世界’；

3．布尔值类型：Boolean ============= var 变量名 = true / false ；

4. 未定义类：Undefined ============= 不作为常用的数据类型

1. 空值：null ====================== var 变量值 = null ；

**--------------------------其他类型转换成数字类型---------------------------**

\* 查看文本类型 \*

alert(typeof 变量名)；

typeof ==================== 把数据类型作为结果返回出来

console.log(变量名)；

console.log() ============= 蓝色为数字类型，黑色为字符串类型

1．字符串(复杂)转换成数字类型

1)字符串中纯数字时，可直接转换

其他类型转换数字类型工具：Number!

var str = “123456”；

var num = Number(s)；

2)特殊的数字转换工具！

var 变量名 = parseint(转换整数)；

var 变量名 = parsefloat(转换小数)；

2．布尔值转换数字类型

只能用Number进行转换

true会转变成 1

false会转变成 0

var 变量名 = true ； var 变量名 = false；

3．undefind

转换数字类型的时候结果永远是 NaN

4．null

转换结果时 0

**--------------------------其他类型转换字符串类型---------------------------**

其他类型转换字符串类型就是给其他类型外部添加单/双引号！

其他类型转换字符串类型工具：String（数据）；

1．数字类型转换字符串类型

var num = 123456；

var str = String(num)；

2．布尔值类型转换字符串类型

var bool = true;

var str = String（bool）；

3．undefind转换字符串类型

var unde;

var str = String（unde）；

4．null转换字符串类型

------------------------------------------------

1．字符串拼接

res = “hello”+ “world”；

2．特殊的字符串转换工具：toString（）；

var a = 123456；

1. toString（）；

**--------------------------其他类型转换布尔值类型---------------------------**

其他类型转换布尔值类型工具：Boolean（数据）；

1．数字类型转换布尔值类型

0和NaN会被转换成false，其他数值会被转换成true！

2．布尔值类型转换布尔值类型

空值“”会被转换成false，其他转换成true！

3．undefind转换布尔值类型

转换成false！

4．null转换布尔值类型

转换成false！

**--------------------------赋值运算---------------------------**

= 就是赋值运算！

var a = 10；

赋值运算支持：+=，-=，\*=，/=，%=

**--------------------------比较运算/数学运算---------------------------**

等于比较：

1．== ：近似等于； (不区分数据类型)

2．=== ：绝对等于； (完全相同)

var a = 10； var b = “10”；

var res = a === b； （相同为true，不同为false）

-------------------------------------------

不等于比较：

1．!= ：近似不等于； (不区分数据类型)

2．!== ：绝对不等于； (完全相同)

var a = 10； var b = “10”；

var res = a === b； （相同为flase，不同为true）

-------------------------------------------

大小比较：

> ，<，>=，<=

var a = 10； var b = 20；

var res = a === b； （相同为true，不同为false）

-------------------------------------------

数学计算：+，-，\*，/，%

**--------------------------逻辑运算---------------------------**

与运算：&&

多个条件同时成立！ （多个条件同时成立为true）

var a = 10； var b = 20；

var res = a < 10 && b < 20；

-------------------------------------------

或运算：||

多个条件有一个成立就成立！

var a = 10； var b = 20；

var res = a < 10 || b > 20；

-------------------------------------------

非运算：!

让状态反过来！ （注：双重否定即为肯定）

var a = true；

var a = !a；

**--------------------------元运算符---------------------------**

自增：++ 自减：--

自增自减符号在前：先运算（先算右边）

自增自减符号在后：先取值（先算左边）

**--------------------------运算优先级---------------------------**

**例如：10+2\*3/4-5**

**借助文档学习优先级：**http://web.h3399.cn/Operator\_Precedence.htm

**运算基本顺序：代码从上到下执行；等式（表达式）从左到右执行!**

**--------------------------if语法---------------------------**

**if(逻辑表达式；条件){代码块}**

**if(逻辑表达式；条件){代码块}else{代码块}**

**if(逻辑表达式；条件){代码块}else(逻辑表达式；条件){代码块}**

**--------------------------switch语法---------------------------**

**switch(数据){**

**case 值1：代码块；**

**case 值2：代码块；**

**……**

**}**

**switch的条件判断具有穿透特性! 所以需要在代码后方添加break；关键字!**

**case 值1：代码块；break；**

**--------------------------三元表达式---------------------------**

**语法：? 代码1 ：代码2；**

**ture执行代码1，false执行代码2!**

**注：代码1和代码2必须一起存在!如果只执行代码1，就需要进行代码占位!推荐占位代码：“”；**

**var a = 60；**

**var res = a >=60 ? “奖励” ：“大笔都”；**

**--------------------------短路运算---------------------------**

**&&：左右两侧都为真。计算结果才是真!左右两侧有一侧为假，计算结果就是假!**

**true && console.log(“没有短路，正常执行”)；**

**false && console.log(“短路，不需要执行右侧代码”)；**

**||：左右两侧有一个为真，就为真!左右两侧都为假，计算结果才是假!**

**true || console.log(“短路，不执行代码”)；**

**false || console.log(“没有短路，正常执行代码”)；**

**直接使用数据代表布尔值进行逻辑判断：**

**0代表偶数，0转换布尔值是false；**

**1代表奇数，1转换布尔值是true；**

**var num = 9；**

**num % 2 === 0 && console.log(“偶数”)；**

**num % 2 === 0 || console.log(“奇数”)；**

**--------------------------while循环语法­­---------------------------**

**语法：while(表达式){代码块}**

**var i = 2；**

**while(i < 3){**

**i ++； （步长，每次值改变的大小）**

**console.log(“打印3次”)；**

**}**

**---------------------------do while循环­­---------------------------**

**语法: do{需要重复执行的代码块-逻辑语句是true还是false都会至少执行一次}**

**while(判定条件){执行循环之后执行的代码块-判定条件false就执行一次}**

**var num = 0；**

**do{**

**console.log(“这个代码会执行10次”)；**

**num ++；**

**}**

**while(num < 10){**

**console.log(“循环执行结束”)；**

**}**

**-----------------------------for循环­­------------------------------**

**语法: for(变量；逻辑判断；步长){代码块}**

**for(var i = 0; i < 50; i ++){**

**console.log(“循环50次”);**

**}**

-------------------------------------------

**循环流程控制语句：**

**break：在循环之中只要读到这个关键字，循环就会立即终止结束，循环中关键字后的代码会被忽略！**

**continue：在循环之中只要读到这个关键字，循环就会立即开始下一次循环，循环中关键字后的代码会被忽略！**

**----------------------------嵌套循环­­------------------------------**

**document.write(“内容”);**

**语法： for(){for(){}}**

**for(var hang = 1; hang <= 9; hang ++){**

**for(var lie = 1; lie <= hang; lie ++){**

**document.wirte(lie +“x”+ hang +“=”+ hang\*lie);**

**}**

**document.write(“<br>”);**

**}**

**------------------------------函数--------------------------------**

**语法: 声明式定义：function 函数名(形参){代码块}**

**赋值式定义：var 函数名 = function 函数名(形参){代码块}**

**调用： 函数名(实参);**

**匿名函数： function(形参){代码块}**

**function foo(a){**

**console.log(“函数形参a值为”,a);**

**}**

**foo(“hello world”);**

**实参比形参多：多余的实参被忽略！**

**形参比实参多：没有对应实参的形参为undefined！**

-------------------------------------------

**return关键字：定义函数的计算结果，同时终止函数运行！**

**function foo(a){**

**return “hello world”;**

**}**

**console.log( foo(10) );**

**--------------------------事件调用函数----------------------------**

**语法： 第一步：html结构上添加对应属性：<button onclick =“函数名()”></button>**

**第二步：在script标签中添加要被调用的函数！function 函数名(){}**

**onclick：单击元素；**

**ondblclick：双击元素；­­­­­­­­**

**onmouseover：鼠标光标移入元素；**

**onmouseout：鼠标光标移出元素；**

**<button onclick = “dianji”>鼠标点击事件</button>**

**<script>**

**function dianji(){**

**document.write(“被点击了”);**

**}**

**</script>**

**-----------------------------预解析-------------------------------**

**代码执行分为两个阶段：当有一个：var a = 10;**

1. **预解析：var a;**
2. **代码翻译：var a = 10;**

**预解析规则会把我们的变量声明拆分成两个部分：1.var a; 声明提升！**

**2.a = 10; 赋值行为！**

**-----------------------------作用域-------------------------------**

**全局作用域：不被函数大括号包裹的部分！**

**全局变量：全局作用域之中声明的变量称之为全局变量，全局变量可以在任何位置被访问到！**

**全局变量的声明周期是永久的！**

**<script> var a = 10; </script>**

-------------------------------------------

**局部作用域：被函数大括号包裹的部分就是局部作用域！**

**局部变量：局部作用域中生命的变量称之为局部变量！**

**function foo(){ var a = 10; }**

-------------------------------------------

**作用域变量访问规则：先在自己的作用域里面查找，如果有直接拿来用，如果没有就像上一级去查找，以此类推！**

**如果一致查找到全局作用域依旧没有找到要访问的变量就会报出错误(引用错误：变量 is defined)！**

**var a = 10;**

**function foo(){**

**function poo(){**

**console.log( a );**

**}**

**}**

-------------------------------------------

**函数的递归：递归的本质也是多次计算，每次计算都会按照规则进行！**

**function foo( a ){**

**if( a === 1 ){**

**return 1;**

**}**

**return foo( a – 1 );**

**}**

**foo(10);**

**------------------------------对象--------------------------------**

**对象的作用：存储多条数据！多条数据描述一个事物！ 如描述一个人！**

**存储数据的格式：key : value**

**key: 表示数据的名称；**

**value：表示数据的值；**

**每条数据用 , 隔离！**

**创建对象: 字面量创建对象：var obj = {};**

**构造函数形式创建对象：var obj = new Object();**

**---------------------------------------**

**遍历对象key值：Object.keys( obj );**

**遍历对象value值：Object.values( obj );**

**这两条对数组也适用！**

**var obj = {**

**name : “吴彦祖”,**

**gender : “男”,**

**height : 178**

**}**

**--------------------------对象增删改查----------------------------**

**增/改：取出运算：[] .**

**语法： 对象.属性名 = 属性值;**

**对象[“属性名”] = 属性值;**

-------------------------------------------

**查：**

**1.取出某一条数据的值，根据属性名找到对应的属性值！**

**语法： 对象[“属性名”]**

**console.log( obj[“a”] );**

**console.log( obj.a );**

**2.取出对象之中全部数据的值，遍历！**

**in关键字: 1.这个关键字单独使用时，用来判断属性名key是否存在于对象之中！**

**语法： “属性名/key” in 对象；**

**在for in循环之中没有特殊的函数就是固定写法,遍历！**

**for(var a in obj){**

**console.log( a , obj[a] );**

**}**

-------------------------------------------

**删：delete**

**语法： delete 对象.属性名;**

**delete 对象[“属性名”];**

**delete obj.a;**

**delete obj[“a”];**

**------------------------------数组--------------------------------**

**数组：数据的组合！ 数组一类同样的数据放在一起！**

**存储数据的格式：下标/key : value！ 不过数组key值是从0开始递增的数字！**

**索引：这个数字称之为数组的索引，索引可以代表数组之中存放的数据！**

**索引计数是从0开始！**

**创建数组： 字面量创建数组：var arr = [];**

**构造函数形式创建数组：new Array( arr );**

**length：代表数组之中的数据数量！**

**length是从1开始计数！**

**var a = obj.length;**

**--------------------------数组增删改查----------------------------**

**运算符路线：**

**增：只能使用[]来取出运算！**

**因为数组是有序的，所以只能向数组末尾新增数据！**

**arr.length正好比数组下标大1！由此可得：语法：arr[arr.length] = “hello world”;**

**改：**

**语法：arr[下标] = “内容”；**

-------------------------------------------

**查：**

**单条查询：**

**语法： arr[下标];**

**遍历查询( for in )：**

**语法： for( var a in arr ){}**

**for in循环不方便去做区间截取行为，使用for循环可以灵活的去获取下标！**

**for( var a = 1; a < arr.length; a ++){**

**console.log( a );**

**}**

**如果只是遍历数组，那么选择for in语法更简单！**

**如果需要控制获取的下标，那么选择for循环控制下标更灵活！**

-------------------------------------------

**删：**

**delete删除数组项会导致数组的连续性遭到破坏！**

**语法： delete arr[下标];**

**如：arr[100] = “不连续的数据”;**

**数组的length会被改变为101！**

**数组中间部分的内容会被默认填充为undefined！**

**不建议这种数组操作行为！！！**

**可以给length赋值！赋值后数组长度发生变化！**

**语法： arr.length = 想保留的数组长度;**

**清空数组：arr.length = 0;**

**----------------------------数组工具------------------------------**

**push: 在数组末尾添加数据！**

**语法： arr.push( 数据 );**

**返回值：添加数据后的数组长度；**

**pop: 删除末尾一条数据！**

**语法： arr.pop();**

**返回值：删除的数据;**

**unshift: 在数组前方添加数据！**

**语法： arr.unshift( 数据 );**

**返回值：添加数据后数组的长度;**

**shift: 删除开头一条数据！**

**语法： arr.shift();**

**返回值：删除的数据;**

**splice: 数组数据的截取！**

**语法： arr.splice( 起始点,替换数量,“数据” );**

**后面的参数可以有多个，写几个参数就在数组被截取的位置新增多少个！**

**把删除数据的数量写0，就可以实现像数组某个位置上添加数据！**

**返回值：截取的数据;**

**slice: 数组截取工具！(和splice不同，不会改变原数组的数据！)**

**语法： slice(起始点，终点);（下标）**

**slice如果是负数参数，表示从后面往前数几个数据！**

**slice 可以只传递一个参数，表示这个参数到结尾所有的数据都会被截取！**

**reverse: 倒序数组！**

**语法： arr.reverse();**

**返回值：倒序后的数组;**

**concat: 拼接数组！**

**语法：arr.concat(拼接的数据/数组，用 , 分隔);**

**join: 把数组拼接成字符串！**

**语法：arr.join(“参数(分隔符)”);**

**数组中的每一项数据用分隔符间隔！**

**如果不需要分隔符，参数可以传递空字符串！**

**把数组的内容直接放入到页面时：**

**document.write(); 如果传入的参数不是字符串，那么document.write会默认把这个数据转换成字符串！**

**数组转字符串arr.toString() 这个规则会保留数组的 , 字符，在渲染页面时显示多余的 , ！**

**sort: 数组排序工具！**

**语法： arr.sort();**

**只会按照数据的第一项内容进行排序！**

**数字从小到大排序语法： arr.sort(function( a,b ){ return a-b });**

**非数字类型也可使用sort排序：**

**var arr = [{ name:11111 },{ name:2222 },{ name:333 }];**

**arr.sort(function( a,b ){**

**return a.name – b.name;**

**});**

**------------------------高级类型存储方式--------------------------**

**特征：在变量之中存储的是地址，所以两个变量比对的结果是false！**

**var arr1 = arr2;**

**arr1[arr1.length] = “hello world”;**

**console.log(arr2);**

**此时arr2也添加上了数据！­­­**

**----------------------------二维数组------------------------------**

**双层嵌套循环，外层循环创建行/内部的数据，内层循环创建列！**

**需求：做一个5\*5的二维数组**

**let arr = [];**

**for(let i = 1; i <= 5; i ++){**

**let item = [];**

**for(let k = 1; k <= 5; k ++){**

**item.push((i-1)\*5+k);**

**}**

**arr.push(item);**

**}**

**---------------------------字符串创建------------------------------**

**单引号创建和双引号创建！**

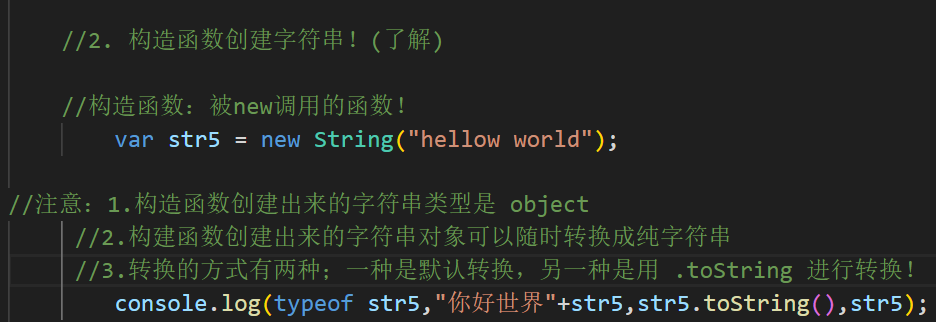
**如果数据中存在单引号时，再使用单引号创建就会报错！**

**如果字符串中同时存在单引号和双引号的情况，需要使用转义字符 / ！(在引号前用)**

var str4 = "I'm \"zhuanyi\"";

-------------------------------------------

**构造函数创建字符串（了解）**

****

**---------------------------字符串工具------------------------------**

**字符串也可使用[]语法进行取出！**

**字符串也有length属性可以查看字符串之中字符数量！**

**字符串也可以被遍历！**

**charAt：找到字符串之中的某个字符**

**语法：str.charAt(下标);**

**toLowerCase: 把字符转换成纯小写字符串！(只支持英文)**

**语法：str.toLowerCase();**

**toUpperCase: 把字符转换成纯大写字符串！(只支持英文)**

**语法：str.toUpperCase();**

**concat: 拼接字符串！(通常使用+替代concat使用)**

**语法：str.concat(字符串);**

**trim: 删除字符串前后的空白字符！**

**语法：str.trim();**

**trimStart: 删除字符串开头的空白字符！**

**语法：str.trimStart();**

**trimEnd: 删除字符串末尾的空白字符！**

**语法：str.trimEnd();**

**charCodeAt（了解）：**

**语法：str.charCodeAt(下标);**

**返回值：字符的unicode编码(10进制编码)**

**进制转换方式(10进制转换成其他进制)：数值.toString(进制);**

**-------------------------字符串截取工具----------------------------**

**这三个API都不会对原字符串进行改变！**

**substr: str.substr(起始点 , 截取几个);**

**let str = “hello world”;**

**let res = str.substr(2,7);**

**console.log(res);**

**substring: str.substring(起始点 , 结束点);**

**let str = “hello world”;**

**let res = str.substring(2,7);**

**console.log(res);**

**slice: str.slice(起始点 , 结束点);**

**let str = “hello world”;**

**let res = str.slice(2,7);**

**console.log(res);**

**----------------------字符串查找、转换工具-------------------------**

**indexOf: 正向查找字符串之中是否存在某个字符！（从左往右查找）**

**语法：str.indexOf(“需要查找的字符”);**

**返回值：查找到的字符下标，如果没找到则返回 -1！**

**lastindexOf: 反向查找字符串之中是否存在某个字符！（从右往左查找）**

**语法：str.indexOf(“需要查找的字符”);**

**返回值：查找到的字符下标，如果没找到则返回 -1！**

**传入参数的时候可以传入字符的组合,如“he”!**

**只要找到符合的字符就立刻停止，并返回当前字符的下标！**

**let str = “hello world”;**

**let res = str.indexOf(“h”);**

**let res = str.lastindexOf(“h”);**

**split: 把字符串按规则分成数组！**

**语法：str.split(“分割字符”);**

**JavaScript会根据传入的数据切分字符串，在字符串之中删除这个数据，并把字符串分割成两段，然后放入数组！**

**let str = “hello world”;**

**let res = str.split(“o”);**

**console.log(res); 结果为 =>**

-------------------------------------------

**字符串内部换行：+**

**单双引号字符串内部不支持换行！**

**let str = “‘<h1>aaa</h1>’+**

**‘<h2>bbb</h2>’+**

**‘<h3>ccc</h3>’”**

**--------------------------日期对象创建-----------------------------**

**new Date();**

**创建当前时间的日期对象：**

**语法：new Date();**

**传递字符串形式创建日期对象：**

**语法：new Date(“年,月,日 时:分:秒”);**

**传递参数形式创建日期对象：**

**语法：new Date(年,月,日,时,分,秒);**

***注意*：传参时间要比目标时间小 1 ！**

**一年有12个月，常规设置月份范围应该是0 ~ 11之间的数值！！！**

**0 代表上年的最后一月或者上个月最后一天**

**需求：创建2024/3/2 13:57:29 的时间！！！**

**let d = new Date(2024,2,2,13,57,29);**

**console.log(d);**

**如果数值过大，就会向后面得日期顺延！**

**new Date(2024,1,2,13,57,29)；**

**最终结果为：2024年2月2日**

**如果数值过小，就会像前面的数值顺延！**

**new Date(2024,-1,2,13,57,29)；**

**最终结果为：2023年12月2日**

**-----------------------------时间戳--------------------------------**

**时间戳：日期对象表述的时间到1970年1月1日之间经过的毫秒数！**

**getTime();**

**语法：日期对象.getTime();**

**let d = new Date(2024,2,2,13,57,29);**

**let t = d.getTime();**

**console.log(t);**

-------------------------------------------

**科学计数法：**

**e + n ：表示n个零！**

**e - n ：表示小数点向前移动n位！**

**--------------------------日期对象工具-----------------------------**

**get… : 读取日期对象中的信息！**

**语法：日期对象.get…();**

**getFullYear() ：获取日期对象的年份信息！**

**getMonth() ：获取日期对象的月份信息！**

**getDate() ：获取日期对象的日期信息！**

**getDay() ：获取日期对象的星期信息！**

**getHours() ：获取日期对象的小时信息！**

**getMinutes() ：获取日期对象的分钟信息！**

**getSeconds() ：获取日期对象的秒信息！**

**getMilliseconds()：获取日期对象的毫秒信息！**

**getTime() ：获取日期对象的时间戳！**

**日期对象之中的信息可以单独获取！(前提：前面已经有日期对象)**

**console.log(year,month,date,day,hour,minute,second);**

**set… ：设置日期对象中的信息！**

**语法：日期对象.set…();**

**参数：设置的日期信息！**

**返回值：新日期的时间戳！**

**set系列跟get系列的区别：set除了个体系列的星期设置getDat之外，**

**其余全部支持，只需把get改成set即可！！**

**set系列的API可以设置不规则的日期！**

**let d = new Date(2024,9,10, 10,10,10);**

**d.setDate(40);**

**let t = d.getTime();**

**-------------------------数学对象-随机数---------------------------**

**随机数：Math.random();**

**需求：随机点名功能！**

**let name\_arr = [“张三”,“李四”,“王五”];**

**let dianming = parseInt(Math.random()\*name\_arr.length);**

**console.log(dianmiong);**

**范围随机数：**

**公式：( max – min ) \* Math.random() + min**

**需求：100 ~ 200之间的随机数！**

**let a = parseInt(Math.random() \* 101 + 100);**

**--------------------------数学对象-取整----------------------------**

**ceil ：向上取整！**

**floor ：向下取整！**

**round ：四舍五入取整！**

**语法：Math.工具名称()；**

**参数：Math.工具名称(被取整的数值)！**

**返回值：按照规则计算的结果！**

**let num1 = 1.001;**

**let num2 = 1.999;**

**console.log(Math.ceil( num1 ), Math.ceil( num2 ));**

**console.log(Math.floor( num1 ), Math.floor( num2 ));**

**console.log(Math.round( num1 ), Math.round( num2 ));**

**--------------------------数学对象-计算----------------------------**

**abs:获取熟知的绝对值！**

**语法：Math.abs(数值);**

**console.log(Math.abs(-100),Math.abs(100));**

**sqrt: 开平方根！**

**语法：Math.sqrt(数值);**

**console.log(Math.abs(4));**

**pow: 乘方工具！**

**语法：Math.pow(数值，几次方);**

**如10的5次方pow(10,5) 也可写成10 \*\* 5(可能不好使)**

**console.log(Math.pow(10,5),Math.pow(10 \*\* 5));**

**min: 取最小值运算！**

**语法：Math.min(数值);**

**console.log(Math.min(1,2,3,4,5,6,7,8,9,0));**

**max: 取最大值运算！**

**语法：Math.max(数值);**

**console.log(Math.max(1,2,3,4,5,6,7,8,9,0));**

**PI: （常数）圆周率Π，3.1415926……**

**语法：Math.abs(数值);**

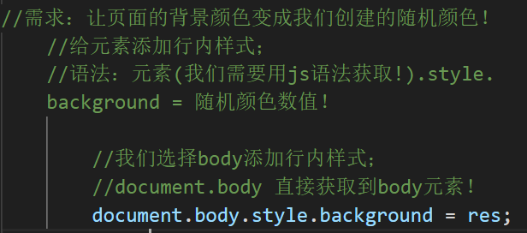
**可以用来计算圆形面积：Π \* r²**

**需求：半径为100的圆形面积！**

**console.log(Math.PI \* Math.pow(100,2));**

**-----------------------给元素添加行内样式--------------------------**

**语法：元素.style.background = 随机颜色数值；**

****

**-------------------------定时器/延时器---------------------------**

**setInterval : 定时器(重复执行)**

**语法：setInterval(重复执行的函数,执行间隔ms);**

**setTimeout : 延时器(执行一次)**

**语法：setTimeout(等待执行的函数，等待时间ms);**

**---------------------------异步机制------------------------------**

**异步机制概念：**

**异步：JavaScript特殊代码的执行顺序！**

**例子：有同学忘记带饭卡了，怎么办?如果一直等着饭卡拿来，那是不是很花时间，后面同学就不能打饭了!所以呢，我让出来，让后面的人先打饭，等我的饭卡到了，我再打饭**

**图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成**

**同步：JavaScript是一个单线程语言，上一行代码结束之后才会执行下行代码！**

**例子：按照排队的顺序，依次打饭，只有前面一个人打完饭后，才轮到后面一个人。**

**文本

描述已自动生成**

**回调函数：上面三种机制都有一个函数，因为这些函数都会在同步程序执行结束之后回头采取调用，因此我们把这些函数称之为一部顺序执行的代码！**

**不被回调函数包裹的程序都是同步程序！**

**异步机制实例：**

**console.log(a);**

**for(let i = 0; I < 10; I ++){**

**setTimeout(function(){ 延时器内部函数时异步机制顺序执行的，所以在内部函数执行的时候，循环执行结束了！**

**console.log(b);**

**}**

**}**

**console.log(c);**

**------------------------------BOM--------------------------------**

**window对象的机制：**

**全局变量都会放到window对象进行存储！**

**我们所有的全局变量访问，都会被放到window对象上进行存储！**

**window对象存储全局变量的时候是以 key:calue 格式进行存储的！**

**window.a = 10; 和 let a = 10; 得到的结果是相同的！**

**图形用户界面, 文本

描述已自动生成let a = 10;**

**window.aa = 20;**

**console.log(window);**

**window对象属性的时候是根据字母顺序A-Z进行排列的！测试时尽可能使用靠前的字母测试！**

**什么是全局变量访问：**

**我们所有所有没有对象的API访问都是全局访问：parseInt，parseFlooat，Math，Date…**

**全局访问的变量都是在window对象上获取的数据，获取的正确语法应该是window.parseInt，window.parseFloat…**

**JavaScript默认直接访问的变量添加window！**

**如果我们声明局部变量的时候没有写let/var关键字，会让我们的局部变量变成全局变量！**

**function foo(){**

**let a = 10; 局部变量！**

**}**

**function foo(){**

**a = 10; 全局变量！**

**}**

**----------------------------BOM工具------------------------------**

**alert()： 提示框！**

**语法：alert(“数据”);**

**文本

低可信度描述已自动生成**

**confirm()：选择提示框！**

**语法：confirm(“数据”);**

**返回值：确定返回true，取消返回false**

**图形用户界面, 应用程序

描述已自动生成**

**prompot()：输入提示框！**

**语法：prompot(“数据”);**

**图形用户界面, 应用程序

描述已自动生成**

**----------------------------BOM事件------------------------------**

**所有的window都可以省略！**

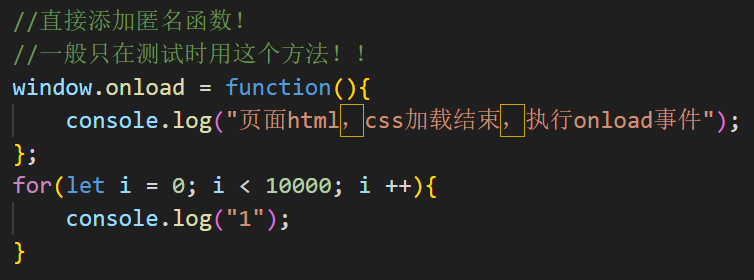
**共用语法：window.事件名 = 事件处理函数！**

**window.onload：当浏览器之中的所有结构都被加载结束之后采取执行的事件！**

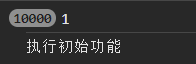
**语法：window.onload = function 函数名(){}**

**为什么有onload事件/作用：html + css 应该是最先被加载的内容，JavaScript应该在html和css加载结束之后执行！**

**文本

描述已自动生成**

**如果我们给window.onload添加了两次函数，这是只有后面的事件处理函数生效！**

****

-------------------------------------------

**window.onscroll：当页面滚动条发生滚动时触发事件！**

**语法：window.onscroll = function 函数名(){}**

**工具：**

**scrollTop：滚动条滚动后顶部隐藏长度！**

**语法：let num = document.body.scrollTop;(获取数据)**

**document.body.scrollTop = 数值;（赋值数据）**

**scrollLift：滚动条滚动后左侧隐藏长度！**

**语法：let num = document.body.scrollLift;(获取数据)**

**document.body.scrollLift = 数值;（赋值数据，可以实现回到顶部按钮功能）**

**需求：当隐藏的高度大于1000时，页面颜色变成黄色！**

**function handlerScroll(){**

**需要做兼容性：同时获取html和css的scrollTop值！哪个标签有值用哪个！**

**let scrolltop = document.body.scrollTop || document.documentElement.scrollTop;**

**if(scrolltop >= 1000){**

**document.body.style.background = “yellow”;**

**}**

**}**

**window.onscroll = handlerScroll;**

**需求：点击按钮回到顶部！**

**<button onclick=“GoBack()”>回到顶部</button>**

**function GoBack(){**

**document.body.scrollTop = 0; 表示body的scrollTop值**

**document.documentEvent.scrollTop = 0; 表示html的scrollTop值**

**}**

-------------------------------------------

**window.onresize：这个事件是在页面存储发生改变的时候触发的事件！(了解)**

**语法：window.onresize() = function 函数名();**

**window.innerWidth： 页面可视区域宽度！**

**window.innerHeight：页面可视区域高度！**

**可视区域：表示可以显示结构这部分的浏览器视口宽高！**

**需求：可视区域窗口改变时触发函数！**

**function handlerAA(){**

**console.log(“页面尺寸发生变化”)**

**console.log(window.innerWidth,window.innerHeight);**

**}**

**window.resize = handlerAA;**

-------------------------------------------

**window.scrollTo()：平滑滚动！**

**属性：**

**top：顶部距离！(number)**

**left：左侧距离！(number)**

**behavior：“smooth”动画效果！(string)**

**传参形式：window.scrollTo(x , y);**

**window.scrollTo(option);**

**需求：平滑回到顶部！**

**<button id=“btn” onclick=“dingbu()”>回到顶部</button>**

**function dingbu(){**

**window.scrollTo({**

**top:0,**

**behavior:“smooth”**

**});**

**}**

**--------------------------BOM子级对象----------------------------**

**注意：地址栏上的所有字符编码格式都是URI编码，这个编码是地址通用编码！**

**window.location：地址栏工具!**

**location.href：当前界面的地址!**

**console.log(location.href);**

**location.href = “路径字符串”；**

**相对路径：location.href = “93\_BOM子级对象.html”;**

**绝对路径：location.href = “G:\javaScript编写\93\_BOM子级对象.html”;**

**绝对路径赋值需要绝对路径前缀：**

**https://**

**file:// (不常用)**

**路径表述方式：**

**/ ：从根目录开始；**

**./ ：从当前目录开始；**

**../：从赏金目录开始；**

**location.reload()：页面刷新功能！**

**function shuaxin(){**

**location.reload();**

**}**

**window.history：操作历史记录！**

**history.back()：回退！**

**function huitui(){**

**history.back();**

**}**

**history.forward()：前进！**

**function qianjin(){**

**history.back();**

**}**

**history.go()：前进或回退几次！（正数前进，负数回退）**

**function Go(){**

**history.go( 2 );**

**}**

**function Go(){**

**history.go( -2 );**

**}**

**-------------------------页面开启关闭工具-------------------------**

**open：打开一个新的页面！**

**语法：open(“地址名”);**

**close：关闭当前页面！**

**语法：close();**

**-----------------------------本地存储-----------------------------**

**localStorage：永久存储！（不主动删除，数据不会消失）**

**添加/修改数据：**

**localStorage.setItem(“存储数据的名称”，“具体的数据”);**

**查看存储的数据：**

**localStorage.getItem(“存储数据的名称”);**

**清空数据：**

**localStorage.clear();**

**sessionStorage：临时存储！（只要关闭页面，存储的数据就消失）**

**添加/修改数据：**

**sessionStorage.setItem(“存储数据的名称”，“具体的数据”);**

**查看存储的数据：**

**sessionStorage.getItem(“存储数据的名称”);**

**清空数据：**

**sessionStorage.clear();**

**--------------------------数据驱动型页面--------------------------**

**document.getElementById(“id名称”);**

**dom名.addEventListener(“事件类型”,函数名称);**

***97\_添加事件-数据驱动型页面：***

**function shanchu( num ){**

**for(let i = 0; i < mingdan.length; i ++){**

**if( mingdan[i].id === num ){**

**break;**

**}**

**}**

**mingdan.splice(i,1);**

**MD();**

**}**

**------------------------本地存储JSON数据-------------------------**

**JSON.parse(); JSON字符串转对象/数组！**

**对象开始的格式：‘{“key”:“value”,“key”:“value”}’**

**let obj = JSON.parse(‘{“k1”:“v1”,“k2”:“v2” }’);**

**console.log(obj);**

**对象开始的格式：[{“key”:“value”,“key”:“value”}]**

**let arr = JSON.parse([{“k1”:“v1”,“k2”:“v2”}]);**

**console.log(arr);**

**JSON.stringify(); JavaScript的对象/数组转JSON字符串！**

**let obj = {k1:v1,k2:v2};**

**JSON.stringify(obj);**

**----------------------------DOM选择器----------------------------**

**基础选择器(使用麻烦，但也能实现功能)：**

**id选择器： document.getElementById(“id名称”);**

**let box = document.getElementById(“box”);**

**标签选择器：document.getElementByTagName(“标签名”);**

**把查找到的符合规则的标签的dom对象全部放到一个数组之中进行返回！**

**文本

描述已自动生成 let divs = document.getElementByTagName(“div”);**

**qs选择器(推荐)：**

**选择从上到下顺序的第一个：**

**document.querSelector(“#id名 或 .class名”);**

**let box = documentSelector(“#box”);**

**选择所有符合的dom对象：**

**document.querySelectorAll(“#id名 或 .class名”);**

**把查找到的符合规则的标签的dom对象全部放到一个数组之中进行返回！**

**let boxs = documentSelectorAll(“#box”);**

**------------------------DOM操作-行内样式-------------------------**

**dom.style.属性名 = (“属性值”)；**

**如果css属性是有 – 的，就要使用驼峰命名法！**

**background-color => backgroundColor**

**let box = document.querySelector(“.box”);**

**box.style.width = (“100px”);**

**dom.style.cssText = “行内样式”;**

**注意：使用cssText添加行内样式的时候一定要确保元素上没有其他的行内样式！**

**文本

描述已自动生成**

**----------------------------类名操作-----------------------------**

**className：类名赋值！**

**dom.className = “类名”；**

**classList：类名工具！**

**dom.classList.add(“类名”)：添加类名！**

**dom.classList.remove(“类名”)：删除类名！**

**注意：className主要用来取值，判断类名之中是否存在某些数据！classList主要用来操作类名的增删行为！**

**文本

描述已自动生成**

**--------------------------原生属性操作---------------------------**

**src：请求连接属性！**

**语法：dom.src = “字符串路径”;**

**href：用于link标签上！**

**语法：link.href = “字符串路径”;**

**document.title：更改页面标签的属性(标题)！**

**语法：document.title = “hello world”;**

**checked：用于checkbox表单工具上！**

**语法：checkbox.checked = true/false;**

**-------------------------自定义属性操作--------------------------**

**dom.setAttribute()：添加/修改自定义属性！**

**dom.getAttribute()：获取自定义属性！**

**语法：dom.setAttribute(“属性名”,“属性值”)；**

**dom.getAttribute(“属性名”)；**

**特征：元素上没有同名属性时会新增！**

**元素上有同名元素时会重新对这个属性赋值！**

**let lis = document.querySelector(“li”);**

**for(let i = 0; i < lis.length; i ++){**

**lis[i].setAttribute(“zidingyi”,i);**

**}**

**function huoqu(){**

**let res = lis[i].getAttribute(“zidingyi”);**

**console.log(res);**

**}**

**dom.removeAttribute()：删除属性！**

**语法：dom.removeAttribute(“属性名”);**

**let li = document.querySelector(“li”);**

**function tianjia(){**

**li.removeAttribute(“zidingyi”)；**

**}**

**---------------------------H5属性操作----------------------------**

**dom.dataset：data-xxx定义属性，其中xxx为我们自己定义的属性！**

**let divs = document.querySelectorAll(“div”);**

**function huoqu(){**

**console.log(divs[0].dataset.index,divs[0].dataset.src);**

**}**

**function fuzhi(){**

**divs[0].dataset.index = 123;**

**}**

**--------------------------元素内容操作---------------------------**

**innerText和innerHTML：主要针对非输入性标签操作的！（div，p，span…）**

**innerText：获取/赋值元素的内容！**

**语法：dom.innerText; 获取**

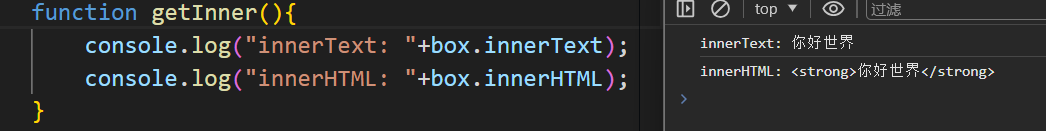
**dom.innerText = “”; 赋值**

**innerHTML：获取/赋值元素的内容！**

**语法：dom.innerHTML; 获取**

**dom.innerHTML = “”; 赋值**

**innerText和innerHTML的区别：**

**console.log打印出的结果：innerHTML连起始结尾标签一起打印！**

**文本

描述已自动生成文本

中度可信度描述已自动生成在页面中显示的结果：innerHTML起始结尾标签直接自动执行了！（em倾斜标签）**

**注意：innerHTML和innerText赋值为替换行赋值！**

**value：对输入性标签获取赋值！（input，testarea，select…）**

**语法：dom.value; (获取)**

**dom.value = “hello world”; （赋值）**

**input输入框里面的事件：**

**语法：**

**blur： 当输入框失去焦点时触发这个事件！**

**场景：用户输入结束，我们可以获取用户输入的信息来进行后续操作！**

**function aBlur(){**

**let Blur = document.getElementById(“Blur”);**

**Blur = blur.value;**

**console.log(Blur);**

**}**

**focus：当输入框触发焦点的时触发这个事件！**

**场景：用户准备输入内容，给用户显示输入提示信息！**

**function aFocus(){**

**let Focus = document.getElementById(“Focus”);**

**console.log(“请输入密码!”);**

**}**

**input：用户输入中触发这个事件！**

**场景：模糊搜索！**

**change：用户输入内容，并和之前进行比较，不同于之前的内容并且光标失去焦点才会触发事件！**

**场景：checkbox单选框，select框，input-file…！**

**--------------------------DOM样式获取---------------------------**

**offsetWidth/offsetHeight：宽高获取！**

**语法：dom.offsetWidth/offsetHeight; (获取)**

**dom.offsetWidth/offsetHeight = Number/变量; （赋值）**

**clientWidth/clientHeight：宽高获取！**

**语法：dom.clientWidth/clientHeight; (获取)**

**dom.clientWidth/clientHeight = “”; （赋值）**

**盒模型：元素分成三层：content，border，margin！**

**offset取值的时候：包含border宽度：content + padding + border ！**

**client取值的时候：不包含border宽度：content + padding ！**

**offsetLeft，offsetTop：偏移获取！**

**语法：dom.offsetLeft/offsetTop; (获取)**

**dom.offsetLeft/offsetTop = Number/变量; （赋值）**

**clientLeft,clientTop：边框宽度获取！**

**语法：dom.clientLeft/clientTop; (获取)**

**dom. clientLeft/clientTop = Number/变量; （赋值）**

**offsetParent：参照物查看！**

**语法：dom.offsetParent;**

**getComputedStyle()：获取元素的所有css样式！**

**语法：getComputedStyle(dom);**

**注意：getComputedStyle()性能消耗比较大，尽量减少使用！！！**

**文本

描述已自动生成平时编写时也应注意尽量减少调用次数：**

**--------------------------DOM节点操作---------------------------**

**包裹关系即为父子关系！**

**同级别即为兄弟即为关系！**

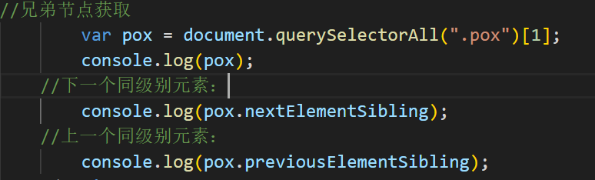
**dom.parentNode：父级节点获取！（只读）**

**dom.children：子级节点获取！（只读）**

**dom.childNodes：所有类型的子级节点获取！（只读）（了解）**

**dom.nextElementSibling：下一级兄弟节点获取！**

**dom.previousElementSibling：上一级兄弟节点获取！**

**注意：兄弟节点获取得在同一个数组之中！！！**

**----------------------------节点操作-----------------------------**

**创建节点：**

**dom.createElement()：创建dom对象！**

**语法：dom.createElemtent(“标签名”);**

**返回值：创建好的dom对象！**

**-------------------------------------------**

**插入节点：**

**dom.appendChild()：插入dom对象！（插入在dom对象的最后方！）**

**语法：dom.appendChild(插入的dom对象);**

**dom.insertBefore()：插入dom对象！（插入到谁前面！）**

**语法：dom.insertBefore(插入的dom对象,插入到谁前面);**

**-------------------------------------------**

**替换节点：**

**dom.replaceChild()：替换dom对象！**

**语法：dom.replaceChild(插入的dom对象,替换哪个dom对象);**

**删除节点：**

**dom.remove()：删除dom对象！**

**语法：dom.remove();**

**dom.cloneNode()：克隆dom对象！**

**语法：dom.cloneNode( true/false );**

**true：完全克隆！内容也一起克隆！**

**false：只会克隆最外层的结构，内层结构不克隆！**

**------------------------------事件-------------------------------**

**属性添加事件：**

**<button onclick=“foo()”>事件</button>**

**function(){ console.log(“属性添加事件！”); }**

**DOM0添加事件：**

**<button id=“btn”>事件</button>**

**let btn = document.getElementById(“btn”);**

**function foo(){ console.log(“DOM0添加事件！”); }**

**btn.obclick = foo;**

**DOM2添加事件：(安全性更高！)**

**<button id=“btn”>事件</button>**

**let btn = document.getElementById(“btn”);**

**function foo(){ console.log(“DOM2添加事件！”); }**

**btn.addEventListener(“click”,foo);**

**删除事件：**

**删除属性添加的事件：dom.removeAttribute(“onclick”);**

**删除DOM0添加的事件：dom.onclick = null;**

**删除DOM2添加的事件：dom.removeEventListener(“click”,函数名)；**

**注意：DOM2事件赋值的函数如果是匿名函数，则这个函数没法被删除！**

**--------------------------事件对象获取---------------------------**

**获取事件对象兼容语法：e = e || event;**

**在兼容之后我们就可以随意使用事件对象了！**

**----------------------------鼠标信息-----------------------------**

**鼠标位置信息：**

**offsetX,offsetY：参照物是距离鼠标最近的元素！**

**clientX,clientY：参照物是浏览器的可视区域边缘！**

**pageX,pageY：参照物是文档顶部！**

**----------------------------鼠标事件-----------------------------**

**鼠标按下事件：mousedown**

**鼠标抬起事件：mouseup**

**鼠标移动事件：mousemove**

**鼠标右键事件：contextmenu**

**鼠标移入事件：mouseover**

**鼠标移出事件：mouseout**

**鼠标移入事件：mouseenter**

**鼠标移出事件：mouseleave**

**mouseover和mouseout这套事件会在移入到元素内部元素的时候重复触发！**

**mouseenter和mouseleave这套时间只会触发一次！**

**注意：over,out和enter，leave两套事件必须配对使用，不能混用！**

**----------------------------键盘事件-----------------------------**

**键盘按下事件：onkeydown**

**键盘抬起事件：onkeyup**

**----------------------------键盘信息-----------------------------**

**1.键盘码信息：**

**keyCode：键盘码信息！**

**console.log(e.keyCode);**

**13：回车！**

**32：空格！**

**65~90：字母范围！**

**37~40：左上右下建范围！**

**2.组合键信息：**

**e.altKey，e.metaKey，e.ctrlKey，e.shiftKey**

**属性：true：表示按下组合键！**

**false：表示没有按下组合键！**

**function foo( e ){**

**e = e || event;**

**if(e.keyCode === 65 && e.ctrlKey){**

**console.log(“完成ctrl + a组合键功能！”);**

**}**

**}**

**document.addEventListener(“keydown”,foo);**

**----------------------------事件派发-----------------------------**

**dispatchEvent：模拟并触发事件！**

**语法：dom.dispatchEvent(new Event(事件类型));**

**如果只有一个事件处理函数，我们直接调用就可以，但如果有多个事件处理函数，我们最好还是使用事件派发功能来触发事件！实现事件处理函数的调用！**

**---------------------------事件流机制----------------------------**

**事件触发分成三个阶段：**

**1. 捕获阶段：事件触发信息从window => document => html => body … => 目标元素！**

**事件的触发顺序是从外向内！**

**2. 目标阶段：事件触发到达触发事件的元素上，此时目标元素上的事件会被执行！**

**3. 冒泡阶段：事件在目标阶段之后，会逐层向外传播事件出发信息，这个事件触发父级所有的元素同类别的事件！**

**.box => .container => body => html =>document => window!**

**--------------------------阻止冒泡行为---------------------------**

**stopPropagation()：阻止冒泡！**

**e.stopPropagation();**

**-------------------------浏览器默认行为--------------------------**

**默认行为概念：a标签点击之后会让页面跳转，form表单内部点击按钮会让页面跳转，文字选中变蓝色，F12开启控制台…**

**默认事件分两类：**

**1.标签内部的阻止：只能在带有路径的标签上进行阻止！**

**解决方法：在属性名之中添加 Javascript:voic(0)**

**Javascript表示执行后方的Javascript代码；void(0)是JavaScript中的一个工具，表示什么也不做！**

**2.在事件之中阻止默认行为，这个阻止行为的工具也是从事件对象之中取出的！**

**解决方法：e.preventDefault();**

**可以在任意位置调用！**

**如果仍想实现跳转功能，或者标签的原始功能，使用JavaScript来完成：location.herf = “**[**http://www.baidu.com**](http://www.baidu.com)**”**

**特殊的默认事件：**

**1.表单的提交事件：**

**可以阻止按钮触发的点击事件的默认行为！**

**文本

描述已自动生成**

**可以阻止表单的submit（提交）事件!**

**文本

描述已自动生成**

**2.右键菜单弹出默认事件：**

**文本

描述已自动生成**

**3.按键的特殊行为也可以阻止！**

**屏幕上有字

描述已自动生成**

**4.阻止标签的文字选中行为！**

**文本

描述已自动生成**

**注意：设置默认行为是为了禁止用户操作！要用 addEventListener + 匿名函数 的形式去组织默认行为！**

**因为这样阻止的默认行为用户是没有办法删除事件的！**

**----------------------------事件委托-----------------------------**

**事件委托逻辑：**

**1. 把事件添加到目标元素的父级！**

**2. 在父级事件执行的过程之中判定触发事件的事件是否符合我们的条件！**

**dom操作可以获取到的信息都可以作为判定依据！**

**class名，标签名，内容…**

**3. 判定结束之后符合条件的数据都可以作为判断依据！**

**事件源：鼠标的点击事件触发时，距离鼠标最近的元素！**

**事件源是需要在事件对象之中获取的！**

**兼容语法：e.target || e.srcElement**

**判定标签类型：nodeName;**

**语法：dom.nnodeName;**

**返回值：大写的标签名！**

**图形用户界面, 文本

描述已自动生成let list = document.queryselector(“#list”);**

**function Click(e){**

**e = e || event;**

**let orign = e.target || e. srcElement;**

**console.log(orign.nodeName);**

**}**

**list.addEventListener(“click”,Click);**

**----------------------------严格模式-----------------------------**

**use strict：开启严格模式！ 查看131\_严格模式**

**严格模式开启分成两种类型：**

**局部开启：让函数内部的代码循环严格模式！**

**全局开启：让所有代码都遵循严格模式！**

**只要这个字符串在代码前方，后面的代码就会按照严格模式进行解释！**

**严格模式特点：**

**1.同名参数不允许存在！**

**2.使用arguments关键字的时候不允许用callee属性！(了解)**

**（arguments：在函数之中给使用的关键字，这个关键字可以获取多个参数，这个关键字还可以访问当前函数本身！）**

**（arguments.callee可以代表当前函数本身，一般会应用在匿名函数之中！）**

**3.严格模式之中对this指向的严格！**

**前提：this直线window对象毫无作用**

**规定：如果this指向了window，那么this变为undefined**

**--------------------------改变this指向--------------------------**

**第一类：在函数调用时改变函数的this指向！**

**fn.call**

**语法：fn.call(this新指向，对应函数调用时的实参1，对应函数调用时的实参2)**

**function foo(a,b){**

**console.log(this,a,b);**

**}**

**foo.call(obj,10,20);**

**fn.apply**

**语法：fn.apply(this新指向，对应的实参1，对应的实参2);**

**使用apply工具调用函数的时候，可以把函数的传参形式变成数组传参！**

**function foo(a,b){**

**console.log(this,a,b);**

**}**

**foo.apply(obj,[10,20]);**

**apply工具进阶应用：**

**apply和call都可以实现函数的调用！不同的是apply可以改变函数的传参形式！**

**有了apply之后传参形式可以是数组了！**

**如果不需要给表函数的this指向，那么在第一个参数上传递false占位！**

**这两个工具的主要作用都是可以随意设置函数的this指向，在函数调用时使用！**

**bind**

**语法：fn.bind(this新指向，固定的参数1，固定的参数2);**

**返回值：返回一个新函数！！！（重点）**

**function poo(a,b,c){**

**console.log(this,a,b,c);**

**}**

**let obj = {name : “poo函数的this指向!”}**

**let bind\_poo = poo.bind(obj,“你好”);**

**bind\_poo(10,20,30);**

**注意：使用bind之后，函数调用就不再时函数本身调用了，而是使用bind返回值去调用！**

**注意：若传递实参，poo函数时可以接收到的，但是对应关系需要固定的形参之后一一对应！**

**---------------------------新数组工具----------------------------**

**forEach：数组遍历/获取数组索引！**

**语法：arr.forEach(函数);**

**无返回值！**

**函数会被调用数组项次的次数！**

**函数有三个形参会被赋值：(可以自定名字)**

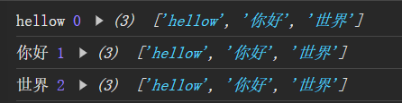
**第一个参数：item： 每条数据！**

**第二个参数：index：当前数据对应的索引！**

**第三个参数：array：正在遍历的数据！（通常不用）**

**let arr = [“hello”,“你好”,“世界”];**

**function BianLi(item,index,array){**

** console.log(item,index,array);**

**}**

**arr.forEach(BianLi);**

**map:修改工具！**

**语法：arr.map(函数);**

**返回值：我们每次执行参数函数返回值得到的所有结果数组！**

**参数之中的函数执行次数和形参也相同！**

**但是我们需要让这个函数有返回值！**

**let arr = [“hello”,“你好”,“世界”];**

**function XiuGai(item,index,array){**

**return item + “~”;**

**}**

**let res = arr.map(XiuGai);**

**console.log(res);**

**map使用场景1：把数组之中的数组拼接成符合HTML规则的数据，然后放在数组之中！**

**function PinJie(){**

**return “<li>”+ item +“</li>”;**

**}**

**let res = arr.map(PinJie);**

**document.write(res);**

**map使用场景2：统一规则进行统一的数据修改！选map肯定没错！！！**

**需求：全场八折！**

**let arr = [{price : 998},{price : 198},{price : 98}];**

**function DaZHe(item){**

**item.price \*= 0.8;**

**item.price = parseInt(item.price \* 10)/10; (保留一位小数)**

**return item;**

**}**

**let res = arr.map(DaZHe);**

**console.log(res);**

**filter：过滤数组之中符合条件的项！**

**语法：arr.filter(函数);**

**返回值：一个结果数组！**

**参数函数的返回值应该是布尔值！**

**true：表示这条数据符合条件会被记入结果之中！**

**false：表示这条数据不符合条件不会被放入结果之中！**

**let arr = [1,2,3,4,5,6,7,8,9];**

**function OuSHu(item){**

**return item % 2 === 0;**

**}**

**let res = arr.filter(OuSHu);**

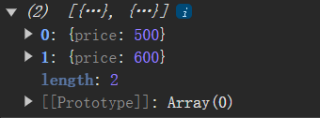
**console.log(res);**

**filter使用场景：把价格在500 ~ 1000之间的产品筛选出来！**

**let arr = [{price : 400},{price : 500},{price : 600}];**

**function SHaiXuan(item){**

**return item.price >= 500 && item.price <= 1000;**

**}**

**let res = arr,filter(SHaiXuan);**

**console.log(res);**

**some：判断（或运算）！**

**语法：arr.some(函数);**

**返回值：布尔值ture/false**

**返回布尔值规则：**

**参数函数的返回值有一个为ture，那么返回结果即为true；**

**如果所有的返回值都为false，返回结果才为false！**

**every：判断（与运算）！**

**语法：arr.every(函数);**

**返回值：布尔值ture/false**

**返回布尔值规则：**

**参数函数的返回值有一个为false，那么返回结果即为false；**

**如果所有的返回值都为true，返回结果才为true！**

**注意：some和every都有短路特征！**

**some会在返回一个true之后停止执行后面的代码！**

**every会在返回一个false之后停止执行后面的代码！**

**reduce：归并！**

**归纳合并数组之中的全部数据，归纳合并的计算规则可以自己编写！**

**语法：arr.reduce(函数);**

**arr.reduce(函数,初始值);**

**传入函数的参数：**

**prev：初始值或者前一个函数的返回值！**

**item：数组的每一项数据！**

**index：索引！**

**array：正在执行所有的总和！**

**需求：求arr之中price的价格总和！**

**let arr = [**

**{price : 100},**

**{price : 200},**

**{price : 300},**

**{price : 400},**

**{price : 500},**

**];**

**function GuiBing(prev,item){**

**return prev + item;**

**}**

**let res = arr.reduce(GuiBing , 0);**

**console.log(res);**

**-------------------------------ES6-------------------------------**

**let：是var的完美平替！**

**let声明的变量遵从的作用域规则是块级作用域！**

**函数作用域：只有函数的大括号才可以划定作用域！**

**块级作用域：只要有大括号就是作用域！**

**let作用域不允许在同一作用域声明同名变量！否则会报错！**

**const：声明的变量特征叫常量！**

**常量：在声明之后就不允许再次被赋值！否则会报错！**

**什么时候使用常量：**

**用常量存储引用类型地址！**

**路径信息也会用常量保存！**

**标记路径的唯一形式语义化的应用场景！**

**注意：常量是锦上添花的奇数，没有常量也可以，只不过有了常量更好！没有必须的场景一定要用常量！**

**---------------------------ES6解构赋值---------------------------**

**数组赋值：**

**语法：let [x,y] = [10,20];**

**解构数组要求左右两侧的结构相同：左侧也用[]去容纳变量！**

**注意：使用这种语法表示我们声明了两个变量，一个变量x，一个变量y，两个变量的作用域就是当前代码所采的作用域！**

**使用技巧：变量换位：**

**let [x,y] = [10,20];**

** [x,y] = [y,x];**

**console.log(x,y);**

**只是用于变量换位，不适用于数组换位！**

**数组解构赋值语法可以应用在函数传参过程之中！**

**function foo([ a,b ]){ console.log( a+b ) };**

**foo( [10,20] );**

**没有对应关系的数据会被忽略！**

**let [a,b] = [1,2,3,4,5];**

**没有对应数据的变量值就是undefined！**

**let [a,b,c] = [1,2];**

**---------------------------------------**

**对象赋值：**

**把对象之中的某条数据取出**

**声明变量a，把对象之中属性名为a的数据赋值给变量a！**

**let obj = { a:10 , b:20 , c:30 };**

**let { a } = obj;**

**多个变量之间的声明用 ，分隔！**

**let { a,b,c } = obj;**

**直接提取某些工具进行使用！（就不需要写前缀了）**

**let { abs,pow,PI } = Math;**

**console.log(abs(-100),pow(10,2),PI);**

**注意：这种提取并不是所有工具都适用！**

**let {reload} = location;**

**location.reload()和reload()的区别在于内部的this指向！**

**不建议这么做，非要做用：reload.call(location)；**

**解构对象中改变变量名：**

**语法：let { 对象之中的key名:新的变量名 } = 对象;**

**let obj = { a:10 , b:20 , c:30 };**

**let { a : aaa } = obj;**

**console.log(aaa);**

**---------------------------ES6箭头函数---------------------------**

**箭头函数：替代了function函数的功能！**

**箭头函数替代了函数封装部分的功能！**

**箭头函数语法更简洁，并且存在简写方案！**

**语法：() => {}**

**()：编写形参用的！**

**{}：编写代码用的！**

**=>：箭头中间不能空格！**

**let foo = (a,b) => {**

**return a + b;**

**}**

**foo(10,20);**

**注意：箭头函数之中支持this，return，关键字，不支持arguments(严格模式)关键字！**

**简写方案：**

**1.参数简写：**

**参数只有一个的时候，可以省略小括号！**

**let poo = a => {console.log(a)};**

**2.大括号简写：**

**代码只有一行的时候，可以简写大括号！**

**简写大括号之后默认会返回代码的计算结果！(默认return计算结果)**

**let poo = (a,b) => console.log( a+b );**

**箭头函数this指向**

**箭头函数的this指向在创建函数的时候就已经确定好了，跟调用者无关！**

**如果需要使用事件处理函数里面的this，这种情况就不能使用箭头函数了！**

**函数的默认值：**

**语法：( a = 默认值,b = 默认值 ) => {}**

**a = 默认值：表示如果没有和a对应的实参a就等于默认值！**

**let foo = (a = 0,b = 10) => {**

**console.log( a,b );**

**}**

**----------------------------…剩余运算----------------------------**

**函数应用：**

**在参数值中可以使用到…运算，在函数之中，…被称为剩余运算！**

**普通函数和箭头函数在函数的运用上完全相同！**

**function foo(){**

**console.log(a,b,…c);**

**}**

**foo(1,2,3,4,5,6,7,8,9);**

**…c：数组类型！无论我们传入多少实参，都会放到c形参之中！**

**…之前可以随便填写形参，…之后不能写形参！**

**剩余运算完整升级了arguments参数获取的功能！**

**数组应用：**

**在数组类型的解构赋值之中进行应用！**

**与函数参数的赋值方式完全一致！**

**let [ a,b,…c ] = [ 1,2,3,4,5,6,7,8,9 ];**

** console.log( a,b,c );**

**对象应用：**

**剩余运算叫展开运算！**

**主要用于对象的合并和扩展！**

**1. 对象合并：**

**…展开运算可以在对象之中对此应用！**

**展开运算可以随意添加自己对象之中的属性！**

**let obj1 = { a:1,b:2 };**

**let obj2 = { c:3,d:4 };**

** let obj3 = { …obj1,…obj2,e:5,f:6 };**

**补充：ES6合并对象操作之中，提供了一个对象工具也可以用于对象合并！**

**语法：Object.assign( 对象1,对象2,… );**

**返回值：返回值对象1 ！**

**let obj1 = { a:1,b:2 };**

**let obj2 = { c:3,d:4 };**

**let obj3 = Object.assign( obj1,obj2,{ e:5 } );**

** console.log( obj3 );**

**2.对象扩展：**

**let obj1 = { a:10,b:20 };**

**let obj2 = { c:30,d:40,…obj1 };**

** console.log( obj2 );**

**---------------------------模板字符串-----------------------------**

**渲染html结构的字符串的时候使用模板字符串更方便！**

**模板字符串支持换行符！**

**模板字符串在拼接变量的时候有更明确语法！**

**语法：创建字符串的时候使用反引号 ` !**

**编辑代码时可以换行！**

**let str = `hello**

**world`;**

**console.log( str );**

**拼接语法：**

**语法：${需要拼接的变量||表达式}：拼接位置！**

**let str1 = “o”;**

**let str2 = `hell${str1} worle`;**

** console.log( str2 );**

**---------------------------对象新语法-----------------------------**

**1. 属性添加的语法：**

**属性添加语法必须是变量作为属性值的情况，必须变量就是对象之中该属性名的情况！**

**使用精简语法的前提：对象上添加的属性名必须是a，属性值时变量a的值！**

**let a = 10;**

**let b = 20;**

**function foo(){ }**

**let obj = { a,b,foo };**

** console.log( obj );**

**2. 函数添加的语法：**

**let obj = {**

**foo(){**

**console.log(“hello”);**

**}，**

**poo(){};**

**}**

**console.log(obj);**

**-------------------------ES6模块化语法---------------------------**

**模块化：JavaScript规则下引入其他的script代码！**

**模块化这种需求在前端时不太需要的！**

**为什么要把函数和数据放在外部的js之中！**

**因为拆分之后的js代码在修改的时候更方便！**

**模块化的目标就是提供可以被我们使用的数据，或者提供某些实现功能的函数！**

**引入模块：**

**1. script标签必须有typr = “module”属性! 直接在script中使用报错！**

**2. 必须把页面通过服务器(协议)打开！ (右键Open with Live server选项)**

**定义模块：**

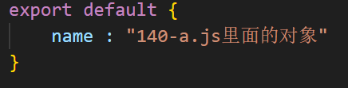
**1. export default 定义的模块！ （直接取出即可！！！）**

**对象模块定义：引用js模块的最终结果是一个对象！**

**语法：import 变量 from “js文件路径”;**

**需求：声明变量A，并把/js/140-a.js里面定义的对象赋值给A变量！**

**<script>**

**import A from “/js/140-a.js”;**

**console.log( A );**

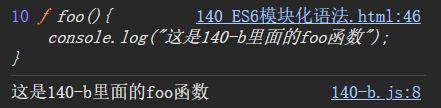
** </script>**

**2. export定义的模块！**

**解构模块的定义：分批定义要返回的模块内容！ (按需取出解构模块之中的数据！)**

**语法：import {变量1，变量2，…} from “js文件路径”;**

**import { a,foo } from “/js/140-a.js”;**

**console.log( a,foo );**

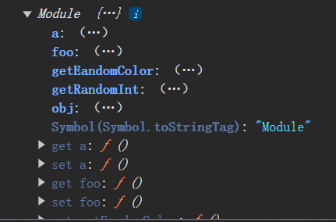
**foo();**

**import解构更名：**

**语法：原名称 as 要修改的名称！**

**import { a as aa , foo as Myfoo } from “/js/140-a.js”;**

**console.log(aa , Myfoo);**

** 可以一次性获取全部的结构对象之中的数据！**

**import \* as shuju from “/js/140-b.js”;**

**console.log( shuju );**

**--------------------------ES6新结构set---------------------------**

**set结构：很想数组，set结构之中没有相同的数据！**

**1. 创建set结构：**

**语法：let set(变量名) = new Set();**

**2. set结构之中数据数量表述：**

**语法：set(变量名).size；**

**3. set结构里存入数据：**

**语法：set.add( 数组/字符串 );**

**添加的数据放在末尾！**

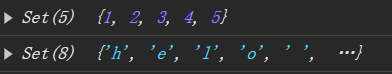
**4. set结构默认去重：**

**对象不会去重！**

**NaN自动去重！**

**let set1 = new Set([1,2,3]);**

**console.log(set1);**

** let set2 = mew Set(“hello world”);**

**console.log(set2);**

**数组去重：**

**let arr = [1,1,1,1,1,2,3,4,5,6,7];**

** arr = [… new Set( arr )];**

**console.log(arr);**

**字符串去重：**

**let str = “aaabbbcccdddeee”**

** str = [… new Set(str)].join(“”);**

**console.log(str);**

**5. set结构转换成数组类型：**

**方法一：[… set]**

**方法二：Arrary.from(set)**

**let arr = [1,2,3,4,5,6,7,8,9];**

**arr = Array.from( new Set(arr) );**

** console.log( arr );**

**6.set结构支持的工具：**

**set.add()：插入数据工具！**

**set.has()：判定当前结构之中是否存在某条数据！**

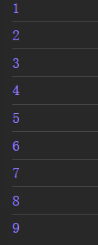
**set.delete()：删除set之中的某条数据！**

**let set = new Set( [1，2，3，4，5，111] );**

**let res = set.has( 111 );**

**console.log( res );**

**set.delete( 111 );**

** console.log( set );**

**7. set遍历：**

**for of遍历**

**for(let key of set){**

**console.log(key);**

**}**

**--------------------------ES6新结构map---------------------------**

**1. 创建map结构！**

**语法：let map = new Map();**

**2. 向map结构之中放入数据！**

**语法：map.set(“ket”,“value”);**

**3. 取出map结构之中的数据！**

**语法：map.get( key );**

**4. map中key值的数量！**

**语法：map.size；**

**5. map判定数据是否存在！**

**语法：map.has( key );**

**6. map删除数据！**

**语法：map.delete();**

**7. map清空数据！**

**语法：map.clear();**

**8. map结构遍历！**

**语法：for of遍历；**

**map可以分别遍历key或value部分：**

**let map1 = new Map();**

**map1**

**.set(“k1”,“v1”)**

**.set(“k2”,“v2”)**

**.set(“k3”,“v3”)**

**遍历key值部分：**

**let keys = map1.keys();**

**图形用户界面

中度可信度描述已自动生成 for(let key of keys){**

**console.log( key );**

**}**

**遍历value值部分：**

**let values = map1.values();**

**for(let value of values){**

**console.log( value );**

**}**

**同时遍历key值value值：**

**let entries = map1.entries();**

**for(let entry of entries){**

**console.log( entry );**

**}**

**---------------------------正则表达式----------------------------**

**字面量创建正则表达式！ （正则表达式内容不能为空！）**

**图形用户界面

描述已自动生成 语法：let reg1 = /a/;**

**构造函数形式创建正则表达式！**

**语法：let reg2 = new RegExp();**

**根据正则表达式表述的规则，在字符串字符串查找符合规则的片段，并把符合正则表达式的片段放在数组之中返回出来！**

**语法：str.match( reg );**

**多个条件使用 | 或运算来进行匹配!**

**语法：let reg = /ll|lo|ld/;**

**let str = “hello world”;**

**let res = str.match( reg );**

**console.log( res );**

**注意：已经匹配的字符会被忽略，下一个匹配不能使用！**

**重复写了多个规则用（）来进行匹配！**

**语法：let reg = /l(l|o|d)/;**

**第一个字符是l第二个字符有三种可能 l o d ;**

**let str = “hello world”;**

** let res = str = str.match(reg);**

**console.log(res);**

**修饰符：//**

**i：忽略大小写**

**g：全局多次匹配**

**文本

描述已自动生成-----------------------正则表达式-元字符-------------------------**



**需求：匹配两个字母一组！**

**let str = “helloo worrllllld”;**

**let reg = /[a-z][a-z]/g;**

** console.log( str.match( reg ) );**

**^：除了元字符[]之中的规则都可以；需要放在[]开头！**

**语法：let reg = /l[^od]/;**

**开头字符是l，第二个字符除了 o d 之外的任意字符！**

** let str = “hello world”;**

**console.log( str.match( reg ) );**

**注意：^功能慎用！使用后会让范围变得太大！**

**-：范围运算！**

**用来省略两个字符之间的字符，表述规则范围！**

**如[0123456789]：0~9之间的所有字符可以写成：[0-9]**

**[12345] 简写成 [1-5]**

**中文汉字简写：[\u4e00-\u9fa5]**

**英文字母简写：[a-z]**

**-------------------------------------**

**正则表达式字符规则简写：**

**\d：在正则中表示数字字符！**

**let str = “1h2e3l4l5o wo6r7l8d9”;**

** let reg = /\d/g;**

**console.log( str.match( reg ) );**

**\D：在正则之中的非数字！**

**let reg1 = /\D/g;**

** console.log( str.match( reg1 ) );**

**\w：数字，字母，下划线！**

**\W：除了数字，字母，下划线！**

**\s：空白字符！**

**\S：非空白字符！**

**\b：表示字符的结尾！**

**是空格，换行就视为结尾！**

**let str = “lo llo loo”;**

**let reg = /lo\b/g;**

**console.log( str.match( reg ) );**

**. ：表示任意字符！**

**不能匹配换行符!**

**注意：.规则谨慎使用！匹配范围太大，如果只是占位可以使用！但尽量不要使用！**

**-------------------------正则表达式-量词--------------------------**

**{n}： 表示出现了n次！**

**let str = “hlllelloowollrrllllldllllllllll”;**

**let reg = /l{2}/g;**

** console.log( str.match( reg ) );**

**{n,m}：表示最小重复n次，最多重复m次！**

**let str = “hlllelloowollrrllllldllllllllll”;**

**let reg = /l{2,5}/g;**

** console.log( str.match( reg ) );**

**{n,}： 表示最小出现n次，最多不限次！**

**let str = “hlllelloowollrrllllldllllllllll”;**

**let reg = /l{2,}/g;**

** console.log( str.match( reg ) );**

**-------------------------------------------**

**量词简写：**

**\*：表示最少0次，最多无限次！等同于{0,}！**

**+：表示最少1次，最多无限次！等同于{1,}！**

**?：表示最少0次，最多1次！等同于{0,1}！**

**注意：\* 需谨慎使用，范围太过广泛！**

**量词可以使用（）来决定匹配的范围！**

**let str = “abababababababab”;**

**let reg = /(ab)+/g;**

** console.log( str.match( reg ) );**

**-----------------------正则表达式-开头结尾------------------------**

**regExp.test：验证是否符合规则！**

**语法：reg.test(“参数”);**

**返回值：匹配true，不匹配false；**

**注意：使用test验证字符串的时候不要添加g修饰符！因为验证时要求字符串完整符合正则规则，所以加g修饰不是正确的语法！**

**想要进行完整验证需要在正则表达式上添加开头结尾！**

**开头：必须写在正则的开头位置！用 ^ 表示！**

**结尾：必须写在正则的结尾位置！用 $ 表示！**

**需求：输入内容必须是12345，如果验证不成功input变成红色，成功变成绿色!**

**let ipt = document.querySekector(“.ipt”);**

**function Color(){**

**let str = ipt.value;**

**let reg = /^12345$/;**

**if(str){**

**ipt.style.color = “greenyellow”**

**}else{**

**ipt.style.color = “red”**

**}**

**}**

**ipt.addElementListener(“blur”,Color);**

**-------------------------正则表达式特性---------------------------**

**replace()：根据正则规则替换掉字符串之中的某些结构！**

**语法：str.replace( regExp , 匹配到的内容要替换的字符串 );**

**返回值：替换之后的字符串！**

**let p\_container = document.queryselector(“.container”);**

**let html = p\_container.innerHTML;**

**贪婪特性：**

**正则表达式在匹配内容的时候尽可能多的匹配符合规则的内容！**

**<p>hello world</p>**

**<p>你好世界</p>**

**开始结束位置，找到开始字符后，一直向后到最后一个符合结束字符处停止查找！**

**let TanLan\_reg = /<.+>/g;**

**文本

描述已自动生成 let res1 = html.replace( TanLan\_reg,“” );**

**console.log( res1 );**

**懒惰特性：**

**正则表达式在匹配内容的时候尽可能多的匹配符合规则的内容！**

**<p>hello world</p>**

**<p>你好世界</p>**

**开始结束位置,就近找到开始结束符号后，向后继续从开始字符寻找！**

**let LanDuo\_reg = /.\*?/g;**

** let res2 = html.replace( LanDuo\_reg,“” );**

**console.log( res2 );**

**-------------------------swiper基础配置---------------------------**

**插件：就是实现了某些功能的工具，这个工具需要我们引入，使用的时候需要查看阅读插件文档！**

**swiper插件功能：实现各种轮播，滑动效果！**

**官网地址：**[**https://www.swiper.com.cn/**](https://www.swiper.com.cn/)

**swiper使用：**

**1. swiper环境搭建：**

**引入swiper css文件；**

**引入到我们自己编写的代码的上方，方便样式覆盖；**

**引入swiper js文件；**

**引入js插件也一定要放在我们业务代码之前；**

**在**[**https://www.bootcdn.cn/**](https://www.bootcdn.cn/)**获取到swiper源码！**

**2. 根据文档，结合需求调用swiper插件！**

**基础结构：swipere-wrapper + swiper-slide**

**功能配置：new Swiper(“类名”)**

**实例见157\_swiper基础配置！**

**------------------------面向对象/构造函数-------------------------**

**普通函数内部代码和构造函数内部代码差距很大！目标变成了创建对象！**

**在构造函数之中this永远指向实例对象！**

**普通函数被new调用了，那么这个函数就变成了构造函数！**

**注意：构造函数new调用创建的对象我们称之为实例对象！**

**function foo( a,b ){**

**this.a = a;**

**console.log( a );**

**this.s = function(){**

**console.log( b );**

**}**

**}**

**图形用户界面, 应用程序

描述已自动生成let aa = new foo( 100,200 );**

**aa.s();**

**-----------------------------原型链-------------------------------**

**每个对象之中都有一个默认的属性：[[prototype]]**

**这条属性就被称为原型指针！**

**这条指针会指向当前实例对象对应的原型对象！**

**普通对象(直接创建的对象)：原型指针直接指向顶级原型对象Object.prototype!**

**实例对象(构造函数创建的对象)：原型指针指向构造函数对应的原型对象！**

**原型机之就是让我们放对应数据的操作方法！**

**原型链机制：**

**let obj = {}**

**访问obj.a，查看对象之中是否存在a属性，不存在就演这原型对象向上查找！**

**如果查询到顶级原型对象一九没有找到a属性，就直接放弃查找！没有数据返回undefined!**

**点开查看原型对象的时候里面有一条属性constructor，表明了当前的原型对象属于哪个构造函数！**

**文本

描述已自动生成Object.prototype就是顶级原型！**

**-----------------------------ES6-类-------------------------------**

**ES6 class写法：**

**1. 先划定一块区域，用来编写面向对象程序，需要给这块区域命名！**

**2. 编写构造函数！**

**class Poo{**

**constructor(){**

**this.a = “hello world”;**

**}**

**foo(){**

**console.log( this.a );**

**}**

**}**

**let poo = new Poo();**

**poo.foo();**

**-------------------对象自定义属性-属性特征修改---------------------**

**Object.defineProperty()：定义一个对象的属性！包括属性的值、可枚举型、可写性、可配置性！**

**语法：Object.defineProperty(obj,prop,descriptor)**

**obj： 要定义属性的对象！**

**prop： 要定义属性的名称！**

**descriptor: 要定义的属性的描述对象！**

**descriptor对象包含以下属性：**

**value: 属性的值，默认为undefined；**

**writable： 属性是否可写，默认false；**

**enumerable： 属性是否可枚举，默认false；**

**configurable：属性是否可配置/是否可以被删除，默认false；**

**---------------------对象自定义属性-属性存取-----------------------**

**此时定义的属性并不是以存储数据为目标的，而是以触发 取值/赋值 行为为目标的属性！**

**注意：属性特征配置和属性存取配置不能同时存在！**

**在descriptor属性使用！**

**get：获取属性值的函数；**

**语法：get:function(){}；取值时就调用这个函数！**

**返回值：这个函数的返回值！**

**set：设置属性值的函数；**

**语法：set:function(){}；赋值时就调用这个函数！**

**返回值：这个函数的返回值！**

**-------------------------前后端交互AJAX---------------------------**

**前端发起工具：AJAX**

**A：异步**

**J：JavaScript**

**A：和**

**X：XML**

**前端的无刷新请求发起工具：**

**有刷新的请求：form表单！**

**缺点：**

**1.会让页面进行整体重新渲染；**

**2.前后端无法完整分离；**

**无刷新的请求：AJAX ！**

**优点：**

**1.页面性能更好，可以实现局部数据更新；**

**2.进行交互的时候让代码的语义化更好；**

**缺点：**

**1.SEO优化不好；**

**2.没有回退按钮；**

**AJAX技术：在前端都是工具(api)，我们根据需求配置工具就可以完成前后端交互！**

**配置正确的请求方式；**

**接受响应数据，编写相应的数据处理代码；**

**AJAX：xhr(原生)|fetch(原生)|axios(插件)|…**

**-------------------xhr请求发送工具(以get为例)--------------------**

**注意：在进行学习之前要开启服务端环境！！！**

**xhr：XMLHttpRequest() 的简写！**

**xhr请求三步走：**

**function sendRequset(){**

**1. 创建请求对象(一个请求一个请求对象);**

**let xhr = new HMLHttpRequest();**

**2. 配置请求信息**

**请求方式：GET\*|POST\*|PUT|PETACH|DELETE**

**请求目标：**

**服务器路径：1.协议(http://) 2.域名(localhost) 3.端口(8888)**

**业务路径：请求某个具体的功能(查看接口文档) /test/first**

**携带参数：**

**跟在业务路径后面，用 ? 分隔；格式为：key1=value1&key2=value2**

**xhr.open(“GET”,`http://localhost:8888/test/first?name=${name.value}&age=${age.value}`);**

**3. 发送请求：**

**xhr.send();**

**注意：请求发送后在浏览器的调试工具 network 查看请求的发送情况！！！**

**在控制台上的请求发送名称是请求目标最后一部分的名称！**

**4. 请求发送结束之后需要获取到服务器给的响应！**

**因为AJAX请求是异步的！所以要使用回调函数的形式来处理响应数据！**

**xhr.onload：是请求对象的事件，这个时间会在请求得到响应之后执行！**

**xhr.responseText：获取响应数据！**

**xhr.onload = function(){**

**console.log(xhr.responseText);**

**}**

**}**

**------------------------xhr响应JSON数据---------------------------**

**function sendRequset(){**

**1. 创建请求对象;**

**let xhr = new HMLHttpRequest();**

**2. 配置请求信息；**

**xhr.open(“GET”,“http://localhost:8888/test/first”);**

**3. 发送请求；**

**xhr.send();**

**接收并处理响应数据；**

**xhr.onload = function(){**

**xhr的响应数据以JSON形式转换成对象形式；**

**let data = JSON.parse(xhr.responseText);**

**console.log(data);**

**box.innerHTML = `这是服务段返回给我们的信息 <strong>${ data.message }</strong>，这是服务端给数据打的标签 <i>${ data.tips }</i>`;**

**}**

**}**

**------------------------post请求携带参数---------------------------**

**get方法在上面！**

**与gat区别：**

**post需要设置请求头：xhr.setRequestHeader(“Content-type”,“appliccation/x-www-form-urlencoded”);**

**post传输数据无大小限制，携带的参数要写在 send 中，格式与get一致！**

**三步走：**

**function sendRequset(){**

**1. 创建请求对象;**

**let xhr = new HMLHttpRequest();**

**2. 配置请求信息；**

**xhr.open(“POST”, http://localhost:8888/test/third);**

**xhr.setRequestHeader(“Content-type”,“appliccation/x-www-form-urlencoded”);**

**3. 发送请求；**

**xhr.send(`name=${name.value}&age=${age.value}`);**

**接收并响应处理数据；**

**xhr.onload = function(){**

**let data = JSON.parse(xhr.responseText);**

**console.log(data);**

**}**

**}**

**--------------------------------------**

**GET/POST排错思路：198\_get,post排错思路！**