JavaScript基础：http://www.w3school.com.cn/jsref/index.asp

> JavaScript是运行在浏览器端的脚本语言，JavaScript主要解决的是前端与用户交互的问题
>
> 是一种动态性、弱类型的语言；
>
> 他的解释器就在我们的浏览器中，是浏览器的一部分
>
> 这门语言对大小写敏感，并会忽略多余的空格，可以使用\进行代码换行，注释使用//或/**/

- 主要由三部分组成：
  - ECMAScript：语言的语法和基本对象
  - 文档对象模型Dom(Document Object Model)：处理网页内容的方法和接口
  - 浏览器对象模型Bom(BrowserObjectModel)：与浏览器进行交互的方法和接口
- 前端三大部：
  - HTML：页面的内容、结构
  - CSS：页面的表现形式、部分动画
  - JavaScript：页面的行为、交互、功能

JavaScript引入

1. 行间事件：为某一个具体的元素标签赋予js内容
   - ```
     <input type="button" value="按钮" onclick="alert('点我');">
     ```
2. 嵌入引入：在文档页面通过Script标签嵌入
   - ```
     <head>
         <title></title>
         <script type="text/javascript">
             alert("ok!");
         </script>
     </head>
     ```
3. 外部引入：定义单独js文件，通过script标签进行引入
   - ```
     <script type="text/javascript" src="js/main.js"></script>
     ```

- > alert函数用来展示一个提示框

变量定义

- ```
  var x = 1
  var y = "2"
  var z = 2
  ```

- 定义变量需要使用关键字：var
  - 同时定义多个变量可以使用 , 隔开
- > **注意**：javascript变量均为对象，每当声明一个变量，就相当于创建了一个对象
- 命名规则：
  1. 区分大小写
  2. 首字符为字母、数字、下划线_、或美元符号$
  3. 其他字符可以为字母、数字、下划线、美元符号
- 调试程序的方法：
  - alert：弹框
  - console.log()：浏览器控制台
  - document.title()：页面标题

基本数据类型

- **Number**：数字类型，可以带小数点，也可以不带
  - ```
    var a = 1;
    var b = 1.5;
    ```
- **String**：字符串类型，可以使用单引号或双引号
  - ```
    var a = "abc";
    var b = "aaaa" + 1
    ```
- **Boolean**：布尔类型，只能是true|false
  - ```
    var a = true;
    ```
- **undefined**：未定义类型
  - ```
    var a;
    ```
- **null**：空对象类型
  - ```
    var a = null;
    ```
- 查看变量数据类型：
  - ```
    var x = "abc";
    alert(typeof x)
    ```
- 匈牙利命名规则：
  - 对象 o (Object)：oPerson
  - 数组 a (Array)：aUsers
  - 字符串 s (String)：sAccount
  - 整数 i (Integer)：iScore
  - 布尔值 b (Boolean)：bIsLogin

- 浮点数 f (Float)：fPrice
- 函数 f (Function)：fEats
- 正则 re (RegExp)：reIDCard

类型转换

- 转换为字符串：**toString**，支持**Boolean**、**Number**、**String**三种主要类型
  - ```
    var x = 1;
    var y = "abc";
    var z = true;
    alert(x.toString()) // "1"
    alert(y.toString()) // "abc"
    alert(z.toString()) // "true"
    ```
- 转换为数字：**parseInt**、**parseFloat**，将只含有数字的字符串变为整形或浮点型，其他类型返回NaN()
  - ```
    var x = "123"
    var y = "123.01"
    var z = "123aa"
    alert(parseInt(x)) //123
    alert(parseFloat(x)) //123
    alert(parseInt(y)) //123
    alert(parseFloat(y)) //123.01
    alert(parseInt(z)) //123
    alert(parseFloat(z)) //123
    ```
- > **注意**：parseFloat转换的包含浮点数的字符串应该是十进制
  >
  > 八进制或十六进制，该方法会忽略前导0，八进制数字020会被解析为20，十六进制数字0xFF，会返回Nan，因为x符号不是有效字符。

强制类型转换

- **Boolean**()：当要转换的值是至少有一个字符的字符串；
  - > 非 0 数字或对象时，Boolean() 函数将返回 true。
    >
    > 如果该值是空字符串、数字 0、undefined 或 null，它将返回 false。
  - ```
    alert(Boolean(0)) // false
    alert(Boolean(1)) // true
    alert(Boolean("1")) // true
    alert(Boolean("1a")) // true
    ```
- **Number**()：换与 parseInt() 和 parseFloat() 方法的处理方式相似，只是它转换的是整个值，而不是部分值。
    ```

```
alert(Number(false)) // 0
alert(Number(true)) // 1
alert(Number(undefined)) // NaN
alert(Number(null)) // 0
alert(Number("1.2")) // 1.2
alert(Number("12")) // 12
alert(Number("1.2.3")) // NaN
alert(Number(new object())) // NaN
alert(Number(50)) // 50
```

- **String()**：可把任何值转换成字符串
  - > **注意**：强制转换成字符串和调用 toString() 方法的唯一不同之处在于，对 null 和 undefined 值强制类型转换可以生成字符串而不引发错误

复合类型

- **Array**：数组，索引从0开始，
  - ```
    var people = ['张三','李四','王五'];
    var people = new Array('张三','李四','王五');
    var people = new Array();
    people[0] = "张三"
    people[1] = "李四"
    people[2] = "王五"
    ```
- **Object**：对象，就像是字典，定义时key值不需要设置类型
  - ```
    var person = {
        name: "张三",
        age: 18,
        sex: "male",
    };
    /*对象有两种访问方式：*/
    person["name"]
    person.name
    ```
  - ```
    var person = new Object();
    person.name = "张三";
    person.age = 17;
    ```

函数

- 函数语法：包裹在花括号中的代码块，前面使用了关键词 function
  - ```
    <button onclick="func()">点击这里</button>
    ```

```
function func(arg1,arg2,...) {
    alert("函数被执行")
    // 执行代码
    return 1; // return是可选的，并且可以不写返回值，单纯只做函数终止
}
// 函数名 func
// 参数 arg1,arg2,...
// 返回值 return 1
func() // 函数执行
```

- 变量作用域：
  - 局部变量
    - 在 JavaScript 函数内部声明的变量（使用 var）是**局部**变量，
      只能在函数内部访问它
      该变量的作用域是局部的
      生命周期：局部变量会在函数运行以后被删除 (生命期从它们被声明的时间开始)
  - 全局变量
    - 在函数外声明的变量是**全局**变量
      网页上的所有脚本和函数都能访问它
      生命周期：全局变量会在页面关闭后被删除 (生命期从它们被声明的时间开始)
      局部变量如果希望变为全局变量
      可以使用windows.var = 的形式赋予给当前窗口
      { var x = 1;
      window.x = x; };

- ```
  function func(x,y){
      return x + y
  }
  var res = func(1,2)
  alert(res)
  ```

- JavaScript函数解析过程：
  1. 预编译：function函数提前，并将var定义的变量声明提前，先暂时赋值为undefined
  2. 执行

```
func() // 弹出提示
alert(iNum) // undefined
alert(abc) // 出错
function func() {
    alert("这个是函数")
}
var iNum = 1
```
```

匿名函数

- 函数可以没有名字，比如直接为某些事件赋值：
- ```
  window.onload = function(){
      var sDate = new Date()
      console.log(sDate)
  }
  ```

封闭函数

- 封闭函数常用来创建一个开始就执行而不用命名的函数
  - ```
    (function(){
        alert("你好");
    })();
    ```
- 也可以在函数定义前加上"~"和"!"等符号来定义匿名函数
  - ```
    !function(){
        alert("你好");
    }();
    ```
- 封闭函数可以创建一个独立的空间，在封闭函数内定义的变量不会影响外部同名的函数和变量，可以避免命名冲突。
  - ```
    var x = 1;
    !function(){
        var x = "这是同名变量";
        alert(x);
    }
    alert(x);
    ```
  - 当页面上引入多个js文件时，用这种办法比较安全。

运算

算术运算符：如y=5

运算符	描述	示例	结果
+	加	x=y+2	x=7
-	减	x=y-2	x=3
*	乘	x=y*2	x=10
/	除	x=y/2	x=2.5
%	取余	x=y%2	x=1
++	累加	x=++y	x=6
--	递减	x=--y	x=4

赋值运算符：如x=10，y=5

运算符	例子	等价于	结果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

- > **注意：数字与字符串相加，结果将成为字符串**

比较运算符：如x=5

运算符	描述	示例
==	等于	x==8 为 false
===	全等（值和类型）	x===5 为 true；x==="5" 为 false
!=	不等于	x!=8 为 true
>	大于	x>8 为 false
<	小于	x<8 为 true
>=	大于或等于	x>=8 为 false
<=	小于或等于	x<=8 为 true

- 比较运算符常在条件语句中进行使用：
  - ```
    var name = "张三";
        if (name=="张三") {
            document.write("这个人是张三")
        }
    ```

逻辑运算符

运算符	描述	示例
&&	and	(x < 10 && y > 1) 为 true
||	or	(x==5 || y==5) 为 false
!	not	!(x==y) 为 true

条件运算符

- ```
  var NumCheck = 0;
  var Dis = (NumCheck==0) ? "是数字0":"不是数字0";
  ```
- 如果变量NumCheck是0，则Dis的值为："是数字0"；反之为："不是数字0"

条件语句

- 条件语句：

```
var iNum = 0;
if (iNum==0){
    ...;
}else if (iNum==1) {
    条件细分...;
}else{
    不满足以上条件均到这里;
}
```

- switch语句：
  - ```
    var day = new Date().getDay();
    // 星期日:0  范围:0~6
    switch(day){
        case 0:
            alert("今天是星期二");
            break;
        case 1:
            ...
            break;
        ...
    }
    ```
  - > **工作原理**：首先设置表达式 n（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 *break* 来阻止代码自动地向下一个 case 运行

---

for循环语句

- **语法**：
  - ```
    for(var i = 0; i < len; i++){
        ...
    }
    // for(起点数据; 判断条件; 数据递增或递减){}
    ```
  - ```
    var i = 0;
    for( ;i < 10; i++){
        ...
    }
    // 如果循环起始值已经被设置，可以在开头省略
    ```
    ```

```
for(var i = 0; ; i++){
    ...
    if (i==5){
        ...
        break; //终止循环
    }
}
// 当没有第二个语句时，必须在循环内提供break，否则循环则无法停下来，可能令浏览器崩溃
```

```
for(var i = 0; i < 10; ){
    console.log(i);
    i += 2;
}
// 如果没有提供第三个语句，可以在for循环中进行编写数值的变化
```

- for/in语句循环遍历对象的属性

```
for (x in object){
    console.log(x);
}
// 字符串：x 取下标
// 数组：x 取下标
// 对象：x 取key
```

```
var x = "abcdef"  // 0,1,2,3,4,5
var y = [1,2,3,4,"5"] // 0,1,2,3,4
var z = { //  name,age,gender
    name:"张三",
    age:16,
    gender:"male",
}
for (obj in z){
    console.log(obj);
}
```

---

while循环语句

- 语法：

```
while (条件){
    执行代码;
}
```

```
var x = "abcdef";
var i = 0;
while (x[i]){
    console.log(x[i]);
    i++;
}
// 下表超出范围时不会报错，返回undefined
```

- do/while循环：
  - > do/while 循环是 while 循环的变体
    >
    > 该循环首先会执行一次循环代码块，然后检查循环条件是否为真
    >
    > 然后如果条件为真的话，就会重复这个循环
  - ```
    do{
        循环执行代码
    }while (条件);
    ```
  - ```
    var i = 3;
    do{
        console.log(i)
        i--;
    }while (i > 5);
    // do/while循环至少会执行一次
    ```

获取页面元素

- 通过页面元素ID值进行获取：document.getElementById('')
  - 获取到的是一个HTML对象，可以赋值给一个变量
- > **注意**：获取对应元素时，首先要确定页面已经生成所需元素，
  >
  > 通常我们将javascript代码写到页面最下面;
  >
  > 或通过使用windows.onload()事件判断是否已经生成页面。
- ```
  <body>
      这是一段待获取的文字
      <script>
          function func(){
              var sP = document.getElementById('p');
              console.log(sP);
          }
          func()
      </script>
  </body>
  <!-- 获取到的内容：<p id="p">这是一段待获取的文字</p> -->
  ```
  ```

```
<body>
    <p id="p">这是一段待获取的文字</p>
    <script>
        window.onload = function(){
            var sP = document.getElementById('p');
            console.log(sP);
        }
    </script>
</body>
<!-- 获取到的内容: <p id="p">这是一段待获取的文字</p> -->
```

操作页面元素

- 可以通过id方式获取到对应页面内的元素，就可以对元素的属性进行操作，包括对属性的读和写
- 读取元素属性：元素.属性
    - ```
      这是一段待获取的文字
      <script>
          var oP = document.getElementById('aaa');
          console.log(oP)
          console.log(oP.id);
          console.log(oP.style);
          console.log(oP.style.color);
      </script>
      ```
- 修改元素属性：元素.属性 = xxx
    - ```
      <p id="aaa" style="color: red;">这是一段待获取的文字</p>
      <button onclick="blue_font()">按钮</button>

      <script>
          function blue_font(){
              var oP = document.getElementById('aaa');
              oP.style.color = "blue";
              // 修改字体样式属性中的字体颜色为蓝色
          }
      </script>
      ```
    - ```
      这是一段待获取的文字
      <button id="color_button">按钮</button>

      <script>
          color_button.onclick = function(){
              var oP = document.getElementById('aaa');
              oP.style.color = "blue";
              // 修改字体样式属性中的字体颜色为蓝色
          }
      </script>
      ```
    - 也可以获取到对应按钮元素后在绑定函数到它：
      ```

```
<p id="aaa" style="color: red;">这是一段待获取的文字</p>
<button id="Button">按钮</button>
<script>
    var oButton = document.getElementById('Button');
    oButton.onclick = function(){
        var oP = document.getElementById('aaa');
        oP.style.color = "blue";
        // 修改字体样式属性中的字体颜色为蓝色
    }
</script>
```

- 读取或写入标签包裹的内容(读取或修改标签文本内容)：**innerHTML**

```
<a id="a" href="https://www.baidu.com">百度</a>
<button onclick="urlChange()">变搜狗</button>
<script>
    function urlChange(){
        var oA = document.getElementById('a');
        oA.href = "https://www.sougou.com";
        console.log(oA.innerHTML); // 获取标签文本内容
        oA.innerHTML = "搜狗"; //修改标签文本内容
    }
</script>
```

JS事件及属性

- 常见事件：

```
- 用户点击鼠标
- 网页已加载
- 图像已加载
- 鼠标移动某个元素上
- 输入字段被改变时
- 提交表单时
- 用户触发某些按键时
```

- **onclick**事件：用户点击鼠标

```
<p onclick="TextChange(this)">这是文本</p>
<!-- this 代表当前所处的元素 -->

<script>
    function TextChange(id){
        id.innerHTML = "文本修改"//可以直接通过传来的参数进行页面元素的读取及修改
    }
</script>
```

```
<p id="p">这是文本</p>

<script>
    var oP = document.getElementById("p");
    oP.onclick = function(){
        oP.innerHTML = "文本修改"//可以直接通过传来的参数进行页面元素的读取及修改
    }
</script>
```

- **onmouseover**事件：鼠标移入
- **onmouseout**事件：鼠标移出
    - ```
      请把鼠标移动过来
      <script>
          var oP = document.getElementById("aaa");
          oP.onmouseover = function(){
              oP.style.color = "green";//可以直接通过传来的参数进行页面元素的读取及修改
          }
          oP.onmouseout = function(){
              oP.style.color = "red";
          }
      </script>
      ```

JS高级

字符串及操作方法

1. 字符串合并：+
2. 数字字符串变整数：**parseInt**()
3. 数字字符串变浮点数：**parseFloat**()
4. 字符串按分隔符切分：**split**("*")
    - ```
      var x = "a*b*c*d"
      alert(x.split("*")) // a,b,c,d
      console.log(x.split("*")) // ["a", "b", "c", "d"]
      ```
5. 查找字符串是否含有某字符，找到返回索引，找不到返回-1：String.**indexOf**()
    - ```
      var x = "abcdefag"
      var res = x.indexOf("z")
      alert(res)
      ```
6. 截取字符串：String.**substring**(start, end)，不包含end索引位置数据
    - ```
      var x = "abcdefag"
      alert(x.substring(2)) // cdefag
      alert(x.substring(2,4)) // cd
      alert(x.substring()) // abcdefag
      ```

7. 字符串反转：通过结合数组的**reverse()**函数

```
var x = "abcd";
console.log(x.split("").reverse().join("")) //dcba
```

数组及操作方法

1. 定义数组的方法：

```
var aList = new Array(1,2,3);
var aList = new Array();
aList[0] = "a";
aList[1] = "b";
var aList = [1,2,3,4,"a"];
```

2. 获取数组的长度：Array.**length()**

```
var aList = new Array(1,2,3);
console.log(aList.length) // 3
```

3. 将数组成员通过指定拼接符合并成一个字符串：Array.**join("*")**

```
var aList = [1,2,3,4,5]
console.log(aList.join("*")) // 1*2*3*4*5
```

4. 向数组的最后增加或删除成员：Array.**pop()**、Array.**push()**

```
var aList = [1,2,3,4,5]
var opa = aList.pop() // opa: 5
console.log(opa) // 5
console.log(aList) // [1, 2, 3, 4]
aList.push("a")
console.log(aList) // [1, 2, 3, 4, "a"]
```

5. 将数组反转：Array.**reverse()**

```
var aList = [1,2,3,4,5];
aList.reverse();
console.log(aList); //[5, 4, 3, 2, 1]
```

6. 返回数组中元素第一次出现的索引值：Array.**indexOf(chr)**

```
var aList = [1,2,3,4,5];
console.log(aList.indexOf(3)) // 2
```

7. 在数组中增加或删除成员，并返回被删除的：Array.**splice(index, howmany, items...)**

> 从index位置开始，给定的hwomany个数的值，并用后面的items替换这些被删除的值

```
var aList = [1,2,"a",4,5]
aList.splice(2,1,"b","c")
console.log(aList) // [1, 2, "b", "c", 4, 5]
```

多维数组

- > 数组的成员包含数组
- ```
  var aList = [1,2,3,["a","b"]]
  console.log(aList[-1][0]) // 出错 undefined
  console.log(aList[3][0]) // a
  ```

定时器

- 作用：
  - > 定时调用函数
    >
    > 制作动画

反复执行定时器

- **setInterval(code, millisec)**：反复执行的定时器
  - **code**: 必须参数，要调用的函数或要执行的代码串
  - **millisec**: 必须参数，执行code任务所需要的事件间隔，以毫秒计
- **clearInterval(setInterval_obj)**：关闭反复执行的定时器
  - ```
    <!--跑马灯效果-->
    <h3 id="h3">abcdefg</h3>
    <button id="start_button">开始</button>
    <button id="stop_button">停止</button>

    <script>
        start_button.onclick = function(){ // 开启定时事件
            var sT = setInterval(loop,1000);
            window.sT = sT; // 声明此sT定时事件为全局变量
        }
        stop_button.onclick = function(){ // 关闭定时事件
            clearInterval(sT)
        }
        function loop(){
            var Opstr = document.getElementById('h3');
            Opstr.innerHTML = Opstr.innerHTML.substring(1) + Opstr.innerHTML[0]
            console.log(Opstr.innerHTML)
        }
    </script>
    ```

等待执行定时器

- **setTimeout**(**code**, **millisec**)：定义只执行一次的等待定时器
  - **code**: 必须参数，要调用的函数或要执行的代码串
  - **millisec**: 必须参数，执行code任务所需要的事件间隔，以毫秒计
- **clearTimeout**(setTimeout_obj)：关闭只执行一次的等待计时器
  - ```
    <h3 id="h3">我是一个内容</h3>
    <button id="start_button">让上面的内容消失</button>
    <script type="text/javascript">
        start_button.onclick = function(){
            var st = setTimeout(clear,1000)
            window.st = st;
        }
        function clear(){
            var oH3 = document.getElementById('h3');
            oH3.innerHTML = "";
        }
    </script>
    ```
    ```