# toggle

# 容器

length 表示一个容器, 'length' 表示字符串, length <- 'length' 表示将字符串放入容器。

# 排序

1. 两两比较排序（冒泡排序）
2. 一个个选择排（选择排序）
3. 起扑克牌法（插入排序）
4. 强迫症收扑克牌法（计数排序）

缺点：

* 排序速度比快排还快；
* 负数和小数不能排序；

1. 快排
2. 归并排序
3. 堆排序
4. 桶排序

将数字分桶，例如1~10一个桶11~20一个桶，每个桶里再排序。

1. 基数排序

先排个位数，后排十位数，再拍百位数。。。。。，先出先进。

# js中的.style只能取内联样式

# 数据结构

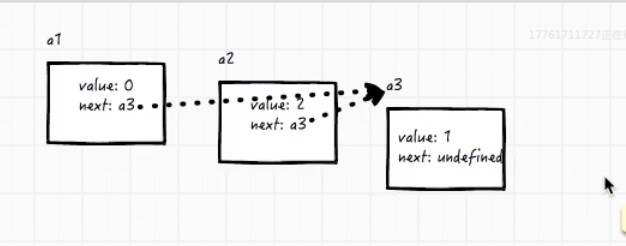
1. 哈希表（Hash Table）
   1. 计数排序中的桶（复杂度 O(n+max)，比快排还快
   2. [桶排序](http://bubkoo.com/2014/01/15/sort-algorithm/bucket-sort/) 与计数排序的区别
   3. [基数排序](http://bubkoo.com/2014/01/15/sort-algorithm/radix-sort/) 与计数排序的区别
2. 队列（Queue）
3. 先进先出
4. 可以用数组实现
5. 举例：排队

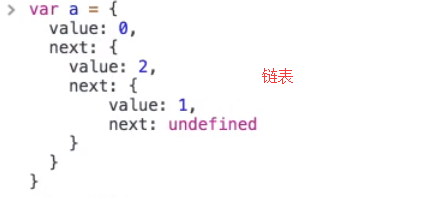
数组.push(“徐涛”)进桶； 数组.shift(“徐涛”)出桶（先进来的先出桶）；

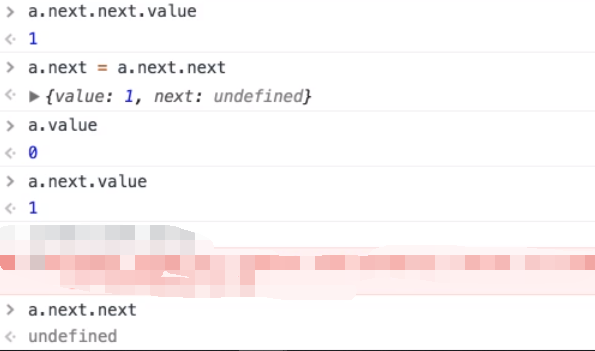
1. 栈（Stack）
2. 先进后出
3. 可以用数组实现
4. 举例：盗梦空间

数组.push(“徐涛”)进栈； 数组.pop(“徐涛”)出栈（从最后开始出栈）；

1. 链表（Linked List）
2. 数组无法**直接**删除中间的一项，链表可以
3. 用哈希（JS里面用对象表示哈希）实现链表
4. head、node 概念







1. 树（tree）
2. 举例：层级结构、DOM
3. 概念：层数、深度、节点个数
4. [二叉树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%A0%91)
5. 满二叉树
6. 完全二叉树
7. 完全二叉树和满二叉树可以用数组实现
8. 其他树可以用哈希（对象）实现
9. 操作：增删改查
10. 堆排序用到了 tree（和快排差不多）
11. 其他：[B树](https://zh.wikipedia.org/wiki/B%E6%A0%91)、[红黑树](https://zh.wikipedia.org/wiki/%E7%BA%A2%E9%BB%91%E6%A0%91)、[AVL树](https://zh.wikipedia.org/wiki/AVL%E6%A0%91)

# 标识符

1. 第一个字符，可以是任意 Unicode 字母（包括英文字母和其他语言的字母），以及美元符号（$）和下划线（\_）。
2. 第二个字符及后面的字符，除了 Unicode 字母、美元符号和下划线，还可以用数字0-9。

不合法标识符：

1a // 第一个字符不能是数字

23 // 同上

\*\*\* // 标识符不能包含星号

a+b // 标识符不能包含加号

-d // 标识符不能包含减号或连词线

JavaScript有一些保留字，不能用作标识符：arguments、break、case、catch、class、const、continue、debugger、default、delete、do、else、enum、eval、export、extends、false、finally、for、function、if、implements、import、in、instanceof、interface、let、new、null、package、private、protected、public、return、static、super、switch、this、throw、true、try、typeof、var、void、while、with、yield。

# undefined

undefined是一个 JavaScript 关键字，表示“无定义”，如果只是声明变量而没有赋值，则该变量的值；

var a;

a // undefined

# null

null：表示空值，即此处的值为空。

# 条件语句

if结构和switch结构，完成条件判断，即只有满足预设的条件，才会执行相应的语句。

## if语句

if (m === 0) {

// ...

} else if (m === 1) {

// ...

} else if (m === 2) {

// ...

} else {

// ...

}

## switch结构

var x = 1;

switch (x) {

case 1:

console.log('x 等于1');

break;

case 2:

console.log('x 等于2');

break;

default:

console.log('x 等于其他值');

}

## 三元运算符 ?:

(条件) ? 表达式1 : 表达式2

上面代码中，如果“条件”为true，则返回“表达式1”的值，否则返回“表达式2”的值。

var even = (n % 2 === 0) ? true : false;

上面代码中，如果n可以被2整除，则even等于true，否则等于false；

# 循环语句

## while 循环

var i = 0;

while (i < 100) {

console.log('i 当前为：' + i);

i = i + 1;

}

上面的代码将循环100次，直到i等于100为止。

## for 循环

var x = 3;

for (var i = 0; i < x; i++) {

console.log(i);

}

// 0

// 1

// 2

上面代码中，初始化表达式是var i = 0，即初始化一个变量i；测试表达式是i < x，即只要i小于x，就会执行循环；递增表达式是i++，即每次循环结束后，i增大1。

注意：for (var n = 1; n < 10; n = n + 1) {

console.log(n)//1~9

}

console.log(n)//10

## do…while 循环

do...while循环与while循环类似，唯一的区别就是先运行一次循环体，然后判断循环条件。

do

语句

while (条件);

// 或者

do {

语句

} while (条件);

不管条件是否为真，do...while循环至少运行一次，这是这种结构最大的特点。另外，while语句后面的分号注意不要省略。

下面是一个例子。

var x = 3;

var i = 0;

do {

console.log(i);

i++;

} while(i < x);

## break 语句和 continue 语句

break语句和continue语句都具有跳转作用，可以让代码不按既有的顺序执行。

break语句用于跳出代码块或循环。

var i = 0;

while(i < 100) {

console.log('i 当前为：' + i);

i++;

if (i === 10) break;

}

上面代码只会执行10次循环，一旦i等于10，就会跳出循环。

# 确定值是什么类型（typeof）

JavaScript 有三种方法，可以确定一个值到底是什么类型。

typeof运算符

instanceof运算符

Object.prototype.toString方法

typeof运算符:数值、字符串、布尔值分别返回number、string、boolean;undefined返回undefined; 对象返回object; 空数组（[]）的类型也是object

# 布尔值

空数组（[]）和空对象（{}）对应的布尔值，都是true。

六个值被转为false，其他值都视为true。

* undefined
* null
* false
* 0
* NaN
* ""或''（空字符串）

# 数值

1. 整数和浮点数

JavaScript 内部，所有数字都是以64位浮点数形式储存，即使整数也是如此。所以，1与1.0是相同的，是同一个数。

JavaScript 提供的有效数字最长为53个二进制位。大于2的53次方的数值，都无法保持精度。

Math.pow(2, 53)

// 9007199254740992

Math.pow(2, 53) + 1

// 9007199254740992

1. 数值范围

如果一个数大于等于2的1024次方，那么就会发生“正向溢出”，即 JavaScript 无法表示这么大的数，这时就会返回Infinity。

Math.pow(2, 1024) // Infinity

如果一个数小于等于2的-1075次方（指数部分最小值-1023，再加上小数部分的52位），那么就会发生为“负向溢出”，即 JavaScript 无法表示这么小的数，这时会直接返回0。

Math.pow(2, -1075) // 0

1. 数值的进制

JavaScript 内部会自动将八进制、十六进制、二进制转为十进制；

有前导0的数值会被视为八进制，但是如果前导0后面有数字8和9，则该数值被视为十进制。

1. NaN

NaN是 JavaScript 的特殊值，表示“非数字”（Not a Number），主要出现在将字符串解析成数字出错的场合。

需要注意的是，NaN不是独立的数据类型，而是一个特殊数值，它的数据类型依然属于Number，使用typeof运算符可以看得很清楚。

typeof NaN // 'number'

NaN不等于任何值，包括它本身。

NaN === NaN // false

NaN与任何数（包括它自己）的运算，得到的都是NaN。

1. Infinity

Infinity表示“无穷”，用来表示两种场景。一种是一个正的数值太大，或一个负的数值太小，无法表示；另一种是非0数值除以0，得到Infinity；一种是非0数值除以-0，得到-Infinity。

Infinity大于一切数值（除了NaN），-Infinity小于一切数值（除了NaN），结果都是false。

# 字符串

1. \

如果长字符串必须分成多行，可以在每一行的尾部使用反斜杠。

1. 转义

反斜杠（\）在字符串内有特殊含义，用来表示一些特殊字符，所以又称为转义符。

需要用反斜杠转义的特殊字符，主要有下面这些。

\0 ：null（\u0000）

\b ：后退键（\u0008）

\f ：换页符（\u000C）

\n ：换行符（\u000A）

\r ：回车键（\u000D）

\t ：制表符（\u0009）

\v ：垂直制表符（\u000B）

\' ：单引号（\u0027）

\" ：双引号（\u0022）

\\ ：反斜杠（\u005C）

上面这些字符前面加上反斜杠，都表示特殊含义。

console.log('1\n2')

// 1

// 2

上面代码中，\n表示换行，输出的时候就分成了两行。

反斜杠还有三种特殊用法。

（1）\HHH

反斜杠后面紧跟三个八进制数（000到377），代表一个字符。HHH对应该字符的 Unicode 码点，比如\251表示版权符号。显然，这种方法只能输出256种字符。

（2）\xHH

\x后面紧跟两个十六进制数（00到FF），代表一个字符。HH对应该字符的 Unicode 码点，比如\xA9表示版权符号。这种方法也只能输出256种字符。

（3）\uXXXX

\u后面紧跟四个十六进制数（0000到FFFF），代表一个字符。XXXX对应该字符的 Unicode 码点，比如\u00A9表示版权符号。

下面是这三种字符特殊写法的例子。

'\251' // "©"

'\xA9' // "©"

'\u00A9' // "©"

'\172' === 'z' // true

'\x7A' === 'z' // true

'\u007A' === 'z' // true

如果在非特殊字符前面使用反斜杠，则反斜杠会被省略。

'\a'

// "a"

上面代码中，a是一个正常字符，前面加反斜杠没有特殊含义，反斜杠会被自动省略。

如果字符串的正常内容之中，需要包含反斜杠，则反斜杠前面需要再加一个反斜杠，用来对自身转义。

"Prev \\ Next"

// "Prev \ Next"

1. 字符串与数组

字符串可以被视为字符数组，因此可以使用数组的方括号运算符，用来返回某个位置的字符（位置编号从0开始）。如果方括号中的数字超过字符串的长度，或者方括号中根本不是数字，则返回undefined。

无法改变字符串之中的单个字符

var s = 'hello';

s[0] // "h"

s[1] // "e"

s[4] // "o"

// 直接对字符串使用方括号运算符

'hello'[1] // "e"

# 对象

1. 键名

对象的所有键名都是字符串

// 不报错

var obj = {

//键名'1p': 'Hello World',//属性

'h w': 'Hello World',

//键名 'p': function (x) { //函数

return 2 \* x;

}

};

};

1. 属性的读取

读取对象的属性，有两种方法，一种是使用点运算符，还有一种是使用方括号运算符。

如果使用方括号运算符，键名必须放在引号里面，否则会被当作变量处理

注意：数值键名不能使用点运算符（因为会被当成小数点），只能使用方括号运算符。

1. 属性的赋值

同属性读取；

var obj = { p: 1 };

// 等价于

var obj = {};

obj.p = 1;

1. 属性的查看

查看一个对象本身的所有属性，可以使用Object.keys方法。

1. 属性的删除：delete 命令

delete obj.p // true

注意，删除一个不存在的属性，delete不报错，而且返回true。

1. 属性的遍历：for...in 循环

var obj = {a: 1, b: 2, c: 3};

for (var i in obj) {

console.log('键名：', i);

console.log('键值：', obj[i]);

}

// 键名： a

// 键值： 1

// 键名： b

// 键值： 2

// 键名： c

// 键值： 3

1. with语句（批量给对象已有键名赋值）

with (对象) {

语句;

}

它的作用是操作同一个对象的多个属性时，提供一些书写的方便。

// 例一

var obj = {

p1: 1,

p2: 2,

};

with (obj) {

p1 = 4;

p2 = 5;

}

// 等同于

obj.p1 = 4;

obj.p2 = 5;

注意：

如果with区块内部有变量的赋值操作，必须是当前对象已经存在的属性，否则会创造一个当前作用域的全局变量。

var obj = {};

with (obj) {

p1 = 4;

p2 = 5;

}

obj.p1 // undefined

p1 // 4

# 函数

如果是通过变量赋值定义的函数，那么name属性返回变量名（函数未命名）。

var f2 = function () {};

f2.name // "f2"

但是，上面这种情况，只有在变量的值是一个匿名函数时才是如此。如果变量的值是一个具名函数，那么name属性返回function关键字之后的那个函数名。

var f3 = function myName() {};

f3.name // 'myName'

1. 参数

函数运行的时候，有时需要提供外部数据，不同的外部数据会得到不同的结果，这种外部数据就叫参数。

function square(x) {

return x \* x;

}

square(2) // 4

square(3) // 9

上式的x就是square函数的参数。每次运行的时候，需要提供这个值，否则得不到结果。

1. 传递方式（注意）

函数参数如果是原始类型的值（数值、字符串、布尔值），传递方式是传值传递（passes by value）。这意味着，在函数体内修改参数值，不会影响到函数外部。

var p = 2;

function f(p) {

p = 3;

}

f(p);

p // 2

如果函数参数是复合类型的值（数组、对象、其他函数），传递方式是传址传递（pass by reference）。也就是说，传入函数的原始值的地址，因此在函数内部修改参数，将会影响到原始值。

var obj = { p: 1 };

function f(o) {

o.p = 2;

}

f(obj);

obj.p // 2

上面代码中，传入函数f的是参数对象obj的地址。因此，在函数内部修改obj的属性p，会影响到原始值。

注意，如果函数内部修改的，不是参数对象的某个属性，而是替换掉整个参数，这时不会影响到原始值。

var obj = [1, 2, 3];

function f(o) {

o = [2, 3, 4];

}

f(obj);

obj // [1, 2, 3]

1. 同名参数

函数f有两个参数，且参数名都是a。取值的时候，以后面的a为准，即使后面的a没有值或被省略，也是以其为准。

function f(a, a) {

console.log(a);

}

f(1) // undefined

调用函数f的时候，没有提供第二个参数，a的取值就变成了undefined。这时，如果要获得第一个a的值，可以使用arguments对象。

function f(a, a) {

console.log(arguments[0]);

}

f(1) // 1

1. arguments 对象

arguments对象包含了函数运行时的所有参数，arguments[0]就是第一个参数，arguments[1]就是第二个参数，以此类推。这个对象只有在函数体内部，才可以使用。

arguments对象可以在运行时修改。

var f = function(a, b) {

arguments[0] = 3;

arguments[1] = 2;

return a + b;

}

f(1, 1) // 5

arguments.length;可以判断函数调用时到底带几个参数。

1. 闭包

只有函数内部的子函数才能读取内部变量，因此可以把闭包简单理解成“定义在一个函数内部的函数”。

闭包的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量始终保持在内存中，即闭包可以使得它诞生环境一直存在。请看下面的例子，闭包使得内部变量记住上一次调用时的运算结果。

# 事件

1. eval 命令

eval命令接受一个字符串作为参数，并将这个字符串当作语句执行。

通常情况下，eval最常见的场合是解析 JSON 数据的字符串，不过正确的做法应该是使用原生的JSON.parse方法

var a = 1;

eval('a = 2');

a // 2

1. parseInt方法用于将字符串转为整数。

如果parseInt的参数不是字符串，则会先转为字符串再转换。遇到有不能转换的字符选择跳过。

如果字符串的第一个字符不能转化为数字（后面跟着数字的正负号除外），返回NaN

parseInt方法还可以接受第二个参数（2到36之间），表示被解析的值的进制，返回该值对应的十进制数。默认情况下，parseInt的第二个参数为10，即默认是十进制转十进制。

parseInt('1000', 2) // 8

parseInt('1000', 6) // 216

parseInt('1000', 8) // 512

1. parseFloat方法用于将一个字符串转为浮点数。

使用注意点同上parseInt

1. isNaN方法可以用来判断一个值是否为NaN。

isNaN只对数值有效，如果传入其他值，会被先转成数值

isNaN('Hello') // true

// 相当于

isNaN(Number('Hello')) // true

1. isFinite方法返回一个布尔值，表示某个值是否为正常的数值。

除了Infinity、-Infinity、NaN和undefined这几个值会返回false，isFinite对于其他的数值都会返回true

1. typeof 判断操作数的类型。
2. className(覆盖之前的class)

*let cName* = *elementNodeReference*.className;

*elementNodeReference*.className = *cName*;

* *cName*是一个字符串变量,表示当前元素的class属性的值,可以是由空格分隔的多个class属性值.

1. classList（添加、删除、）

**elementClasses**.classList.add('active')

add( String [, String] )

添加指定的类值。如果这些类已经存在于元素的属性中，那么它们将被忽略。

remove( String [,String] )

删除指定的类值。

item ( Number )

按集合中的索引返回类值。

toggle ( String [, force] )

当只有一个参数时：切换 class value; 即如果类存在，则删除它并返回false，如果不存在，则添加它并返回true。

当存在第二个参数时：如果第二个参数的计算结果为true，则添加指定的类值，如果计算结果为false，则删除它

contains( String )

检查元素的类属性中是否存在指定的类值。

replace( oldClass, newClass )

用一个新类替换已有类。

1. toString()

函数的toString方法返回一个字符串，内容是函数的源码。函数内部的注释也可以返回。

function f() {

a();

b();

c();

// 这是一个

// 多行注释

}

f.toString()

//返回结果：

// function f() {

// a();

// b();

// c();

// 这是一个

// 多行注释

// }

1. onerror （图像加载过程中发生错误）

onerror事件会在文档或图像加载过程中发生错误时被触发。

1. previousSibling（同层前一个节点）

previousSibling 属性返回同一树层级中指定节点的前一个节点。

1. onkeypress（在键盘上）按键时触发。
2. localStorage.setItem()（本地持久缓存）

localStorage.setItem('zzz', JSON.stringify(hash))

1. JSON.stringify()（JavaScript值(对象或者数组)转换为一个 JSON字符串显示）

JSON.stringify(hash)

1. JSON.parse()（解析字符串）

JSON.parse(localStorage.getItem(name) || 'null')

1. 获取窗口大小

window.innerHeight（innerWidth）：表示可见区域的高度（宽度），如果底部（右侧）有滚动条，会把底部（右侧）滚动条的高度（宽度）计算进去，此方法适用于标准浏览器及IE9+。

document.documentElement.clientHeight（clientWidth）：表示可见区域的高度（宽度），不会把滚动条的高度宽度计算进去，此方法适用于标准浏览器及低版本IE标准模式。

document.documentElement.offsetHeight（offsetWidth）：首先offsetWidth获得的值与clientWidth一样，而offsetHeight获得的是整个文档的高度。

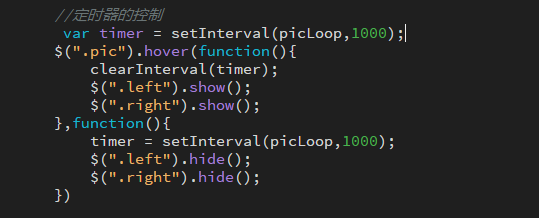
注意：

个人认为上面的documentElement不代表body。当一个element有border时，element.offsetHeight包含border的值，而element.clientHeight不包含border的宽度。

1. 监听窗口大小  
   window.οnresize
2. onmouserdown鼠标按下、onmousermove移动、onmouserup松开（手机不适用）
3. ontouchstart鼠标按下、ontouchmove鼠标移动、ontouchend鼠标松开（适用手机）
4. 计时器setInterval()clearInterval()方法

setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。

由 setInterval() 返回的 ID 值可用作 clearInterval() 方法的参数。

1. Match方法

match() 方法可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。

该方法类似 indexOf() 和 lastIndexOf()，但是它返回指定的值，而不是字符串的位置。

stringObject.match(searchvalue)

stringObject.match(regexp)

|  |  |
| --- | --- |
| **参数** | **描述** |
| searchvalue | 必需。规定要检索的字符串值。 |
| regexp | 必需。规定要匹配的模式的 RegExp 对象。如果该参数不是 RegExp 对象，则需要首先把它传递给 RegExp 构造函数，将其转换为 RegExp 对象。 |

1. isNaN函数

isNaN() 函数用于检查其参数是否是非数字值。

isNaN(x)

|  |  |
| --- | --- |
| **参数** | **描述** |
| x | 必需。要检测的值。 |

如果 x 是特殊的非数字值 NaN（或者能被转换为这样的值），返回的值就是 true。如果 x 是其他值,则返回 false。

1. prompt() 方法（提示用户进行输入的对话框）

prompt() 方法用于显示可提示用户进行输入的对话框。

<html>

<head>

<script type="text/javascript">

function disp\_prompt()

{

var name=prompt("Please enter your name","")

if (name!=null && name!="")

{

document.write("Hello " + name + "!")

}

}

</script>

</head>

<body>

<input type="button" onclick="disp\_prompt()"

value="Display a prompt box" />

</body>

</html>

1. in操作符（判断键名是否属于对象，包含继承属性）

var mycar = {make: "Honda", model: "Accord", year: 1998};

//注意，属性名必须是字符串形式，因为make不是一个变量

if ( "make" in mycar ){

document.write('true');

}

else{

document.write('false'); // 显示true

}

1. hasOwnProperty方法，是否为对象自身的属性，而非继承。

in运算符的一个问题是，它不能识别哪些属性是对象自身的，哪些属性是继承的。就像上面代码中，对象obj本身并没有toString属性，但是in运算符会返回true，因为这个属性是继承的。

这时，可以使用对象的hasOwnProperty方法判断一下，是否为对象自身的属性。

var obj = {};

if ('toString' in obj) {

console.log(obj.hasOwnProperty('toString')) // false

}

# 更改盒子内容

1. Write方法

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

**提示：**您只能在 HTML 输出中使用 document.write。如果您在文档加载后使用该方法，会覆盖整个文档。

1. innerText属性

innerText：页面获取内容的时候，会把标签过滤掉，会对标签进行转义。

1. Value属性

val()不同于text()和html()，val()用于input节点内容的存取。

1. innerHTML属性

设置或返回表格行的开始和结束标签之间的 HTML，会对标签进解析。

<html>

<head>

<script type="text/javascript">

function getInnerHTML()

{

alert(document.getElementById("tr1").innerHTML);

}

</script>

</head>

<body>

<table border="1">

<tr id="tr1">

<th>Firstname</th>

<th>Lastname</th>

</tr>

<tr id="tr2">

<td>Peter</td>

<td>Griffin</td>

</tr>

</table>

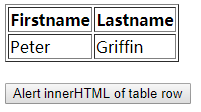
<br />

<input type="button" onclick="getInnerHTML()"

value="Alert innerHTML of table row" />

</body>

</html>



显示如下：

点击后

1. Jquery中使用text方法

设置或者获取所选元素的文本内容；

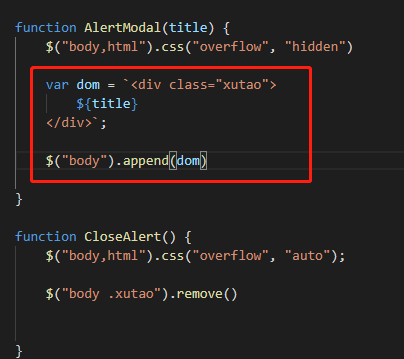
1. Jquery中使用html方法

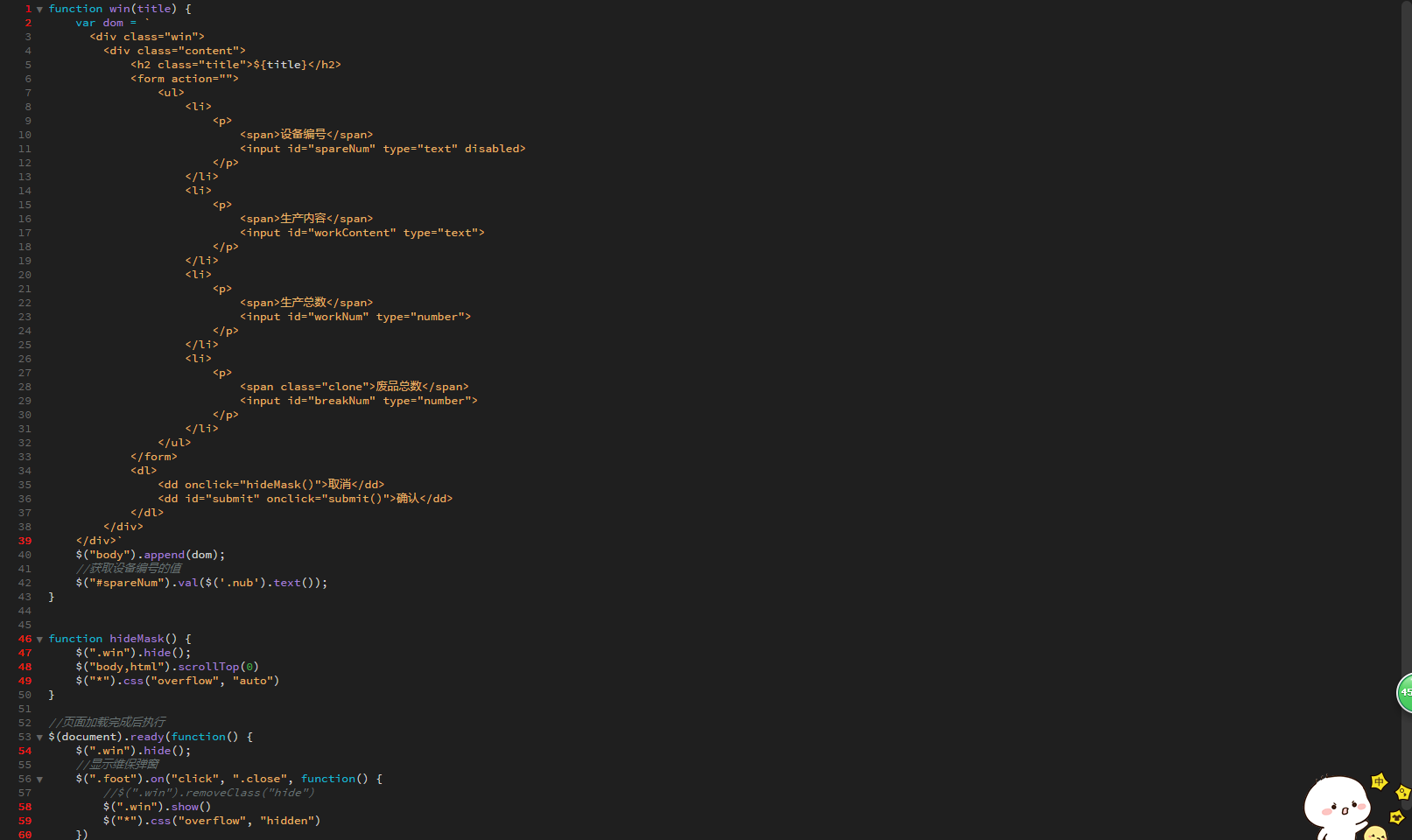
html();设置或者获取所选元素的内容（包括html标记）；

1. Jquery中使用val方法

val();设置或者获取表单字段的值（前提是表单设置了value属性）；

1. jq拼接





1. jq中attr更改div属性

$("img").attr("width","180");

1. jq中css更改css属性

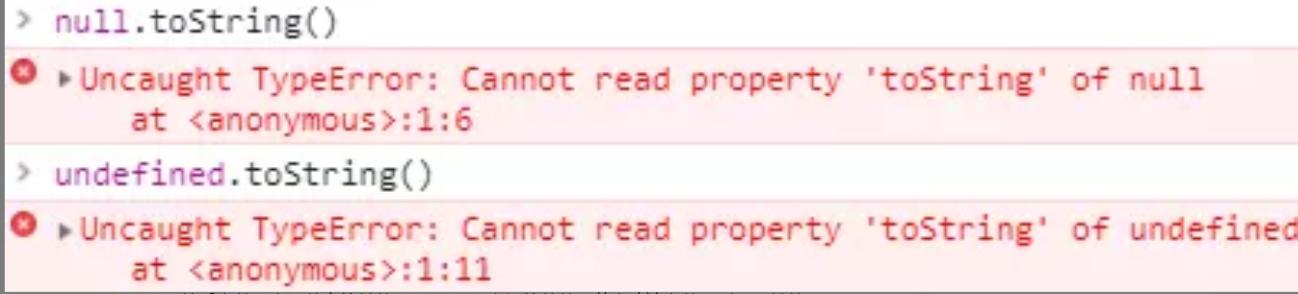
$("p").css("color","red");

# 转成字符串

1. toString()

可以将除null和undefined以外所有的的数据都转换为字符串

* null和undefined无法调用toString方法



* 数字调用toString

a. 接受参数：表示数字的基数，是 2 ~ 36 之间的整数。若省略该参数，则使用基数 10。

(15).toString() // "15"

(15).toString(2) // "1111"

(15).toString(8) // "17"

(15).toString(16) // "f"

b.

// 1.07 连续乘以七个 1000

var a = 1.07 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000;

// 七个1000一共21位数字

a.toString(); // "1.07e21"

* 布尔值调用toString

true.toString() // "true"

false.toString() // "false"

* Date调用toString

Date.toString() // "function Date() { [native code] }"

(new Date).toString() // "Wed May 15 2019 13:33:28 GMT+0800 (中国标准时间)"

* Error调用toString

Error.toString() // "function Error() { [native code] }"

(new Error('bad request')).toString() // "Error: bad request"

* 其他复杂类型调用toString

({}).toString() // "[object Object]"

({ a:2, b:function(){} }).toString() // "[object Object]"

([]).toString() // ""

([1,'a',true]).toString() // "1,a,true"

[1,undefined,function(){},4].toString() // "1,,function(){},4"

(function a(){console.log(111)}).toString() // "function a(){console.log(111)}"

1. String()

可以将null和undefined转换为字符串，但是没法转进制字符串，其他和toString相同

String(null) // "null"

String(undefined) // "undefined"

String(11) // "11"

String(1.07 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000 \* 1000) // "1.07e+21"

String(true) // "true"

String(false) // "false"

String(Date) // "function Date() { [native code] }"

String(new Date) // "Wed May 15 2019 15:30:10 GMT+0800 (中国标准时间)"

String({}) // "[object Object]"

String([]) // ""

String([1, true, 'a']) // "1,true,a"

String(Error) // "function Error() { [native code] }"

String(new Error('bad request')) // "Error: bad request"

String(function a(){console.log(111)}) // "function a(){console.log(111)}"

1. JSON.stringify()

将 JSON 对象序列化为字符串时也用到了 toString，对大多数简单值来说，JSON 字符串化和 toString() 的效果基本相同，只不过序列化的结果总是字符串。**JSON.stringify()在对象中遇到 undefined 、 function 和 symbol 时会自动将其忽略，在数组中则会返回 null 。**  
以下是不同

JSON.stringify(null) // "null"

JSON.stringify(undefined) // undefined

typeof JSON.stringify(undefined) // "undefined"

JSON.stringify(Date) // undefined

typeof JSON.stringify(Date) // "undefined"

JSON.stringify(new Date) // ""2019-05-15T07:41:58.810Z""

JSON.stringify({}) // "{}"

JSON.stringify(Error) // undefined

JSON.stringify(new Error('bad request')) // "{}"

JSON.stringify(function a(){console.log(111)}) // undefined

typeof JSON.stringify(function a(){console.log(111)}) // "undefined"

JSON.stringify([1,undefined,function(){},4]); // "[1,null,null,4]"

JSON.stringify({ a:2, b:function(){} }); // "{"a":2}"

我们可以向 JSON.stringify() 传递一个可选参数 replacer，它可以是数组或者函数，用来指定对象序列化过程中哪些属性应该被处理，哪些应该被排除。

* 字符串数组，其中包含序列化要处理的对象的属性名称，除此之外其他的属性则被忽略。
* 函数，它会对对象本身调用一次，然后对对象中的每个属性各调用一次，每次传递两个参数，键和值。如果要忽略某个键就返回 undefined ，否则返回指定的值。

var a = {

b: 42,

c: "42",

d: [1,2,3]

};

JSON.stringify( a, ["b","c"] ); // "{"b":42,"c":"42"}"

JSON.stringify( a, function(k,v){

console.log(k, v)

if (k !== "c") return v;

} );

// {b: 42, c: "42", d: Array(3)}

// b 42

// c 42

// d [1, 2, 3]

// 0 1

// 1 2

// 2 3

// "{"b":42,"d":[1,2,3]}"

通过上面的例子我们看到，当replacer为函数时它是递归调用的。