目录

[前端基础 1](#_Toc476848317)

[HTML 组件 2](#_Toc476848318)

[input 2](#_Toc476848319)

[checkbox 2](#_Toc476848320)

[Javascript 内置函数 2](#_Toc476848321)

[setTimeout 2](#_Toc476848322)

[JavaScript 语法 2](#_Toc476848323)

[变量 2](#_Toc476848324)

[const 2](#_Toc476848325)

[let 3](#_Toc476848326)

[相等性判断 4](#_Toc476848327)

[全等操作 === 4](#_Toc476848328)

[非严格相等 == 4](#_Toc476848329)

[数组 4](#_Toc476848330)

[push 4](#_Toc476848331)

[concat 5](#_Toc476848332)

[判断元素是否在数组中 5](#_Toc476848333)

[正则表达式 5](#_Toc476848334)

[RegExp 正则表达式对象 5](#_Toc476848335)

[exec方法 5](#_Toc476848336)

[闭包 6](#_Toc476848337)

[词法作用域 6](#_Toc476848338)

[闭包的组成： 6](#_Toc476848339)

[实用的闭包 7](#_Toc476848340)

[用闭包模拟私有方法(module pattern) 7](#_Toc476848341)

# 前端基础

## HTML 组件

### input

#### checkbox

设置checkbox状态：

native：document.getElementById("check1").checked=true

jquery： $('.myCheckbox').prop('checked', true);

设置与未设置checked的区别

<form action="/example/html/form\_action.asp" method="get">

<p><input type="checkbox" name="vehicle" value="Bike"> I have a bike</p>

<p><input type="checkbox" name="vehicle" value="Car" checked="checked"> I have a car</p>

<input type="submit" value="Submit">

</form>

## Javascript 内置函数

### setTimeout

setTimeout(code, millisec) 方法用于在指定的毫秒数后调用函数或计算表达式。

## JavaScript 语法

### 页面加载JS

### 变量

#### const

声明只读常量，一旦声明，常量值就不能改变

const PI = 3.141592;

PI = 3.141; //TypeError: Assignment to constant variable.

#### let

let 是es6新增的命令，用来声明变量，用法与var类似，但声明的变量只在let所在的代码块内有效。

注意：函数级作用域并不适用var,let

function fun(){

let x = 1;

console.log('x:', x);

var b = 2;

console.log('b:', b)

}

fun()

console.log(x) //error

console.log('out b:', b) //error

for 循环适用let

var arr = ["aa", "bb", "cc"];

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

console.log("i:", i);

}

console.log("out i:", i)

在for里使用let可以防止对后续变量的污染

for循环中使用var

var arr = ["aa", "bb", "cc"];

for(var i=0; i<arr.length; i++){

console.log("i:", i);

}

console.log("out i:", i)

输出：

i: 0

i: 1

i: 2

out i: 3

特性：

1. 不存在变量提升

console.log(you); //throw error

console.log(me); //undefined 变量已经存在，但没有值

var me = 'leon';

let you = 'jack';

1. 封闭作用域(temporal dead zone)

tmp = "123";

if(true){

tmp = "abc"; // throw ReferenceError:tmp is not defined

let tmp;

}

1. 不允许重复定义

if(true){

let tmp = 1;

let tmp = 2; //throw SyntaxError: Identifier 'tmp' has already been declared

}

### 相等性判断

#### 全等操作 ===

比较值及类型是否相等

#### 非严格相等 ==

比较前进行类型才转换，再比较值相等

### 数组

内置方法：

#### push

var arr = [“aa”, “bb”, “cc”];

arr.push(“cccc”);

console.log(arr);

数组添加值用push方法

note:push改写了原来对象的值

#### concat

合并两个数组

var ls = ["join\_key", "ssl\_proxy\_sni", "ssl\_proxy\_ciphers", "ssl\_proxy\_protocols"]

var arr3 = ["name"]

var arr4 = arr3.concat(ls)

console.log(arr4)

display:

[ 'name',

'join\_key',

'ssl\_proxy\_sni',

'ssl\_proxy\_ciphers',

'ssl\_proxy\_protocols' ]

note:concat()并不会改变原有的数组，而是返回一个新数组

#### 判断元素是否在数组中

var arr1 = [“aa”, “bb”, “ccc”];

arr1.indexOf(“aa”)

如果在数组中，则返回位置值，否则显示-1

#### 数组复制 slice()

slice() 返回一个新的复制的数组对象

语法：

arr.slice()

arr.slice(begin)

arr.slice(begin, end)

## 面向对象

### 分类：

#### Native Object(build-in Object)

原生对象

#### Host Object

宿主对象提供运行环境

### Object

#### methods:

Object.keys(obj)

如果传入的对象是可枚举的，则

## 正则表达式

### 修饰符

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| [i](http://www.w3school.com.cn/jsref/jsref_regexp_i.asp) | 执行对大小写不敏感的匹配。 |
| [g](http://www.w3school.com.cn/jsref/jsref_regexp_g.asp) | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

### 方括号

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符。 |
| [^abc] | 查找任何不在方括号之间的字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符。 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符。 |
| [A-z] | 查找任何从大写 A 到小写 z 的字符。 |
| [adgk] | 查找给定集合内的任何字符。 |
| [^adgk] | 查找给定集合外的任何字符。 |
| (red|blue|green) | 查找任何指定的选项。 |

### 元字符

元字符（Metacharacter）是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| . | 查找单个字符，除了换行和行结束符。 |
| \w | 查找单词字符。 |
| \W | 查找非单词字符。 |
| \d | 查找数字。 |
| \D | 查找非数字字符。 |
| \s | 查找空白字符。 |
| \S | 查找非空白字符。 |
| \b | 匹配单词边界。 |
| \B | 匹配非单词边界。 |
| \0 | 查找 NUL 字符。 |
| \n | 查找换行符。 |
| \f | 查找换页符。 |
| \r | 查找回车符。 |
| \t | 查找制表符。 |
| \v | 查找垂直制表符。 |
| \xxx | 查找以八进制数 xxx 规定的字符。 |
| \xdd | 查找以十六进制数 dd 规定的字符。 |
| \uxxxx | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

### RegExp 正则表达式对象

语法：new RegExp(*pattern*, *attributes*);

pattern 匹配模式,可以是字符串或正则表达式

attributes 有三个可选值， 就是修饰符 g i m 三个

示例：

1.用字符串匹配

var str = "www.baidu.com"

var regex = new Regex(/BAIDU/, ‘i’)

str.match(regex) // == str.match(/BAIDU/i)

2.用正则匹配

var str = "www.baidu.com"

var regex = new RegExp(/^w.\*m$/, 'i')

str.match(regex)

注：^w : 以w开头

m$: 以m 结尾

.\* : 表示任意字符串

#### exec方法

## 闭包

### 词法作用域

嵌套的函数可以访问到其外层作用域中声明的变量

function init(){

let name = "Chrome";

function display(){

console.log("name:", name);

}

display();

}

init(); //显示Chrome

实例2：

function init(){

let name = "Chrome";

function display(){

console.log("name:", name);

}

return display;

}

var myFunc = init();

myFunc();

display 函数在执行前被从外围函数中返回了。

### 闭包的组成

函数：

创建该函数的环境：在作用域中的所有局部变量

示例：

function makeAdd(x){

return function(y){

return x+y;

}

}

var add5 = makeAdd(5); //设置运行环境变量，其返回值是一个匿名函数

var add10 = makeAdd(10);

console.log(add5(2));

console.log(add10(4));

add5和add10两个闭包具有相同的函数定义，但创建了不同的变量环境。

### 实用的闭包

<script>

function makeSizer(size) {

return function() {

document.body.style.fontSize = size + 'px';

};

}

var size12 = makeSizer(12);

var size14 = makeSizer(14);

var size16 = makeSizer(16);

document.getElementById('size-12').onclick = size12;

document.getElementById('size-14').onclick = size14;

document.getElementById('size-16').onclick = size16;

</script>

### 用闭包模拟私有方法(module pattern)

var Counter = (function(){

var privateCounter = 0;

function changeBy(val) {

privateCounter += val;

}

return {

increment:function(){

changeBy(1);

},

decrement:function(){

changeBy(-1);

},

value:function(){

return privateCounter;

}

}

})();

console.log(Counter.value()); /\* logs 0 \*/

Counter.increment();

Counter.increment();

console.log(Counter.value()); /\* logs 2 \*/

Counter.decrement();

console.log(Counter.value()); /\* logs 1 \*/

console.log(Counter.privateCounter); //undefined

## Jquery

### 查找节点

<**div class="input-group"**>  
 <**select name="datacenter" class="form-control"**>  
 {% **for dc in dcs** %}  
 <**option value="**{{ **dc**.0 }}**" selected="selected"**>{{ **dc**.1 }}</**option**>  
 {% **endfor** %}  
 </**select**>  
<**span class="input-group-addon"**>  
 <**input type="checkbox" class="checkChange" aria-label="..."**>  
</**span**>  
</**div**>

$(**".checkChange"**).each(**function**(){  
 **var** selectVal = $(**this**).parent().prev(**"select"**).val()  
 **console**.log(**"select :"**, selectVal)  
});

查找以.checkChange为起点的select控件，看代码可知select位于checkbox的父类的上一个。

parent() 父节点

prev() 上一个节点

next() 下一个节点 ，select控件的下一个节点就是span