Javascript学习记录

<https://www.runoob.com/js/js-tutorial.html>

<https://www.liaoxuefeng.com/wiki/1022910821149312>

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference>

1. 快速入门
   1. 代码书写

法一：

将JavaScript代码放在<head>的<script>中，将直接被浏览器执行

<html>

<head>

<script>

alert('Hello, world');

</script>

</head>

<body></body>

</html>

法二：

把JavaScript代码放到一个单独的.js文件，然后在HTML中通过<script src="..."></script>引入这个文件，这样，/static/js/abc.js就会被浏览器执行

<html>

<head>

<script src="/static/js/abc.js"></script>

</head>

<body>

...

</body>

</html>

把JavaScript代码放入一个单独的.js文件中更利于维护代码，并且多个页面可以各自引用同一份.js文件。

可以在同一个页面中引入多个.js文件，还可以在页面中多次编写<script> js代码... </script>，浏览器按照顺序依次执行。

* 1. 代码调试

使用chrome浏览器，点击菜单“查看(View)”-“开发者(Developer)”-“开发者工具(Developer Tools)”，快捷键F12

在Elements可以查看页面源代码，在Console（控制台）可以输入js指令。





* 1. 基本语法

每个语句以;结束，一行可以包含多个语句，语句块使用{…}分隔

使用//注释单行，使用/\*...\*/注释多行语句

JavaScript是大小写敏感的

1. 数据类型
   1. Number

JavaScript不区分整数和浮点数，统一用Number表示，以下都是合法的Number类型：

123; // 整数123

0.456; // 浮点数0.456

1.2345e3; // 科学计数法表示1.2345x1000，等同于1234.5

-99; // 负数

NaN; // NaN表示Not a Number，当无法计算结果时用NaN表示

Infinity; // Infinity表示无限大，当数值超过Number所能表示的最大值时，就表示为Infinity

Number可以直接做四则运算+,-,\*,/,%(求余)

* 1. 字符串

字符串是以单引号’或双引号”括起来的任意文本

转义字符\，\n换行，\t制表符，\\表示\，\x##可以表示ASCII码

多行字符串`…`

字符串连接+

模板字符串`…${varname}…`用变量替换${}的内容

var name = '小明'; var age = 20;

console.log(`你好, ${name}, 你今年${age}岁了!`);

常用属性/函数

|  |  |  |
| --- | --- | --- |
| length | 字符串长度属性 | var s = 'Hello, world!'  s.length; // 13 |
| [] | 字符串索引 | s[12]; // '!'  s[13]; // undefined |
| toUpperCase() | 全部变为大写 | s.toUpperCase() // "HELLO, WORLD!" |
| toLowerCase() | 全部变为小写 | s.toLowerCase() // "hello, world!" |
| indexOf() | 搜索指定字符串出现的位置 | s.indexOf(“world”) //7 |
| substring() | 返回指定索引区间的子串 | s.substring(0, 5); // 从0到5（不包括5），返回'hello'  s.substring(7); // 从索引7开始到结束，返回'world' |

* 1. 布尔值

有true、false两种值，布尔运算：&&与运算，||或运算，!非运算

比较运算符

当我们对Number做比较时，可以通过比较运算符得到一个布尔值，<,>,<=,>=,==

JavaScript允许对任意数据类型做比较，注意：

1）==和===

false == 0; // true

false === 0; // false

==比较，它会自动转换数据类型再比较，很多时候，会得到非常诡异的结果；

===比较，它不会自动转换数据类型，如果数据类型不一致，返回false，如果一致，再比较。

由于JavaScript这个设计缺陷，不要使用==比较，**始终坚持使用**===**比较**。

2）NaN与所有其他值都不相等，包括它自己：

唯一能判断NaN的方法是通过isNaN()函数：

isNaN(NaN); // true

3）浮点数的相等比较

要比较两个浮点数是否相等，**只能计算它们之差的绝对值，看是否小于某个阈值**Math.abs<阈值：

Math.abs(1 / 3 - (1 - 2 / 3)) < 0.0000001; // true

* 1. null和undefined

null表示一个“空”的值，辨析：0是一个数值，''表示长度为0的字符串

undefined表示值未定义。区分两者的意义不大。**大多数情况下，我们都应该用null。undefined仅仅在判断函数参数是否传递的情况下有用。**

* 1. 数组

<https://www.runoob.com/jsref/jsref-obj-array.html>

数组是一组按顺序排列的集合，集合的每个值称为元素。JavaScript的数组可以包括任意数据类型。

1）数组的创建

法一：数组用[]表示，元素之间用,分隔。

[1, 2, 3.14, 'Hello', null, true];

法二：通过Array()函数实现

new Array(1, 2, 3); // 创建了数组[1, 2, 3]

然而，出于代码的可读性考虑，强烈**建议直接使用[]**。

2）数组元素的访问

数组的元素可以通过索引来访问。索引的起始值为0：

var arr = [1, 2, 3.14, 'Hello', null, true];

arr[0]; // 返回索引为0的元素，即1

arr[5]; // 返回索引为5的元素，即true

arr[6]; // 索引超出了范围，返回undefined

3）常用属性/方法

|  |  |  |
| --- | --- | --- |
| length | 数组长度 | arr.length |
| [] | 数组索引 | arr[0] |
| indexOf() | 搜索指定元素位置 | arr.indexOf(10) |
| slice() | 截取数组的部分元素 | arr.slice(0,3) //从索引0开始到3，不包括3  arr.slice() //复制arr |
| push() | 向数组末尾添加若干元素  返回数组长度 | arr.push(1,2) |
| pop() | 删除数组的最后一个元素  返回删除的元素 | arr.pop() |
| unshift() | 向数组头部添加若干元素  返回数组长度 | arr.unshift(‘a’,’b’) |
| shift() | 删除数组的第一个元素  返回删除的元素 | arr.shift() |
| sort() | 对数组排序，默认顺序 | arr.sort() |
| reverse() | 反转数组 | arr.reverse() |
| splice() | 从指定索引开始删除，添加元素  返回删除的元素 | var arr=[1,2,3,4,5];  arr.splice(2, 3, 'a', 'b'); // 返回[3,4,5]  arr; // [1,2,’a’,’b’] |
| concat() | 连接两个数组，返回新的数组 | arr.concat(1.2.[3,4]); //在arr后面连接1,2,3,4 |
| join() | 把数组元素用指定字符连起来  返回字符串 | arr.join(‘-‘) |

4）多维数组

如果数组的某个元素又是一个Array，则可以形成多维数组

练习：1）如何通过索引取到500这个值 2）打印数组元素

//1

'use strict';

var arr = [[1, 2, 3], [400, 500, 600], '-'];

//2

'use strict';

var arr = ['小明', '小红', '大军', '阿黄'];

arr.sort();

console.log(`欢迎${arr[0]},${arr[1]},${arr[2]}和${arr[3]}同学`);

//console.log(`欢迎${arr.sort().slice(0,-1)}和${arr.sort().slice(-1)}同学!`);

* 1. 对象

JavaScript的对象是一组由键-值组成的无序集合，例如：

var person = {

name: 'Bob',

age: 20,

tags: ['js', 'web', 'mobile'],

};

对象的键都是字符串类型，值可以是任意数据类型。

其中每个键又称为对象的**属性**，要获取一个对象的属性，用对象变量.属性名：

person.name; // 'Bob'

判断一个对象是否包含某个属性

|  |  |  |
| --- | --- | --- |
| in | 返回true/false，对于变量继承的属性也会判为真 | ‘name’ in person; // true  ‘toString’ in person; // true |
| hasOwnProperty() | 判断属性是否是变量自身拥有的而不是继承的 | person.hasOwnProperty(‘name’); //true  person. hasOwnProperty(‘toString’); //false |

* 1. 变量

变量名规范：大小写英文、数字、$和\_的组合，且不能用数字开头，不能是JavaScript的关键字

申明变量用var语句，只能申明一次

变量赋值用=，JavaScript是动态语言，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，可以是不同类型的变量

var a; // 申明了变量a，此时a的值为undefined

var $b = 1; // 申明了变量$b，同时给$b赋值，此时$b的值为1

var s\_007 = '007'; // s\_007是一个字符串

var Answer = true; // Answer是一个布尔值true

var t = null; // t的值是null

要显示变量的内容，可以用console.log(x)

strict模式

JavaScript中，如果一个变量没有通过var申明就被使用，那么该变量就自动被申明为全局变量：

strict模式下，强制通过var申明变量，未使用var申明变量就使用的，将导致运行错误。

启用strict模式的方法是在JavaScript代码的第一行写上：

'use strict';

这是一个字符串，不支持strict模式的浏览器将报错

1. 条件和循环
   1. 条件判断

if () {

…

} elseif () {

…

} else {

…

}

* 1. For循环

for (i=1; i<=100; i++) {

…

}

* 1. For in

循环对象的属性，可以用hasOwnProperty()过滤继承的属性

var o = {

name: 'Jack',

age: 20,

city: 'Beijing'

};

for (var key in o) {

if (o.hasOwnProperty(key)) {

console.log(key); // 'name', 'age', 'city'

}

}

数组Array也是对象，索引是属性，可以通过索引循环访问

var a = ['A', 'B', 'C'];

for (var i in a) {

console.log(i); // '0', '1', '2'

console.log(a[i]); // 'A', 'B', 'C'

}

* 1. While

while () {…}

do {…} while ()

1. Iterable：Map & Set

JavaScript的对象有个小问题，就是键必须是字符串。但实际上Number或者其他数据类型作为键也是非常合理的。

* 1. Map

Map是一组key-value键值对结构，具有极快的查找速度。对一个key多次赋不同的value会覆盖之前的值

var m = new Map(); // 空Map

通过set(key,value)方法添加键值

m.set('Adam', 67); // 添加新的key-value

通过has(key)方法检验key是否存在，通过get(key)方法访问相应的值

m.has('Adam'); // 是否存在key 'Adam': true

m.get('Adam'); // 67

通过delete(key)方法删除元素

m.delete('Adam'); // 删除key 'Adam'

* 1. Set

Set和Map类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在Set中，没有重复的key，重复的元素会自动被过滤。

创建Set，需要提供一个Array作为输入，或者直接创建一个空Set：

var s1 = new Set(); // 空Set

var s2 = new Set([1, 2, 3]); // 含1, 2, 3

通过add(key)方法可以添加元素到Set中，可以重复添加，但不会有效果：

s.add(4);

通过delete(key)方法可以删除元素：

s.delete(3);

* 1. Iterable

ES6标准引入了新的iterable类型，Array、Map和Set都属于iterable类型。在for … in循环中，它遍历的是对象的属性，所以如果给Array对象手动增加属性，会导致属性也被遍历。以通过新的for ... of循环来遍历，解决这个问题

var a = ['A', 'B', 'C'];

var s = new Set(['A', 'B', 'C']);

var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);

for (var x of a) { // 遍历Array

console.log(x);

}

for (var x of s) { // 遍历Set

console.log(x);

}

for (var x of m) { // 遍历Map

console.log(x[0] + '=' + x[1]);

}

此外，iterable还内置了forEach()方法

Array

var a = ['A', 'B', 'C'];

a.forEach(function (element, index, array) {

// element: 指向当前元素的值

// index: 指向当前索引

// array: 指向Array对象本身

console.log(element + ', index = ' + index);

});

Set

var s = new Set(['A', 'B', 'C']);

s.forEach(function (element, sameElement, set) {

console.log(element);

});

Map

var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);

m.forEach(function (value, key, map) {

console.log(value);

});

1. 函数
   1. 函数基础

函数定义

function 函数名(参数) {函数体}

return语句返回结果，没有return返回undefined

var 函数名=function(参数) {函数体};

注意第二种定义最后要加分号’;’

函数调用

函数名(输入参数)，按顺序输入参数，参数比定义的多或少都不会报错，但会因为参数为undefined不能完成功能，可以在函数中进行参数检查

function abs(x) {

if (typeof x !== 'number') {throw 'Not a number';}

…

}

arguments

只在函数内部起作用，并且永远指向当前函数的调用者传入的所有参数，即使函数没有定义形参也能获取实参，常用于判断传入参数

function foo(x) {

console.log('x = ' + arguments[0]); // 10

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

console.log('arg ' + i + ' = ' + arguments[i]); // 10, 20, 30

}

}

foo(10, 20, 30);

rest参数

指向除了函数内已定义参数的其他输入参数

function sum(...rest) {

s=0;

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

if (typeof rest[i]==='number'){

s+=rest[i];

}

}

return s;

}

// 测试:

var i, args = [];

for (i=1; i<=100; i++) {

args.push(i);

}

if (sum() !== 0) {

console.log('测试失败: sum() = ' + sum());

} else if (sum(1) !== 1) {

console.log('测试失败: sum(1) = ' + sum(1));

} else if (sum(2, 3) !== 5) {

console.log('测试失败: sum(2, 3) = ' + sum(2, 3));

} else if (sum.**apply**(null, args) !== 5050) {

console.log('测试失败: sum(1, 2, 3, ..., 100) = ' + sum.apply(null, args));

} else {

console.log('测试通过!');

}

return

由于JavaScript引擎会自动在句末添加分号，可能会导致语句无法执行，正确的写法是把return {写在一行

function foo() {

return { // 这里不会自动加分号，因为{表示语句尚未结束

name: 'foo'

};

}

* 1. 变量作用域与结构赋值

作用域

函数内部声明的变量作用域为整个函数

嵌套的函数内部可以访问外部函数的变量

JavaScript编译时会先扫描函数体把所有声明的变量提升到函数顶部，所以通常首先声明所有变量

function foo() {

**var**

x = 1, // x初始化为1

y = x + 1, // y初始化为2

z, i; // z和i为undefined

// 其他语句:

...

}

不在函数内部声明的变量就是全局变量，默认绑定到全局对象window

名字空间

全局变量会绑定到window上，不同的js文件如果使用了相同的全局变量，或者定义了相同名字的顶层函数，都会造成命名冲突

减少冲突的一个方法是把自己的所有变量和函数全部绑定到一个全局变量中。例如：

// 唯一的全局变量MYAPP:

var MYAPP = {};

// 其他变量:

MYAPP.name = 'myapp';

MYAPP.version = 1.0;

// 其他函数:

MYAPP.foo = function () {

return 'foo';

};

局部作用域

用关键字let替代var声明变量，可以使for循环等语句中定义的变量具有块级（函数）作用域

for (**let** i=0; i<100; i++) {...}

常量

const定义常量，具有块级作用域

解构赋值

直接对多个变量同时赋值

var [x, y, z] = ['hello', 'JavaScript', 'ES6'];

let [x, [y, z]] = ['hello', ['JavaScript', 'ES6']];

let [, , z] = ['hello', 'JavaScript', 'ES6'];

从一个对象中取出若干属性，对象中嵌套的属性也可以

var person = {

name: '小明',

age: 20,

gender: 'male',

passport: 'G-12345678',

school: 'No.4 middle school'

address: {

city: 'Beijing',

street: 'No.1 Road',

zipcode: '100001'

}

};

var {name, age, passport} = person;

var {gender, address: {city, zip}} = person;

如果要使用的变量名和属性名不一致，可以用下面的语法获取

let {name, passport:id} = person;

name; // '小明'

id; // 'G-12345678'，passport不是变量

解构赋值还可以使用默认值

var {name, single=true} = person;

如果变量已经被声明了，使用{}批量赋值会报错，可以用小括号括起来

({x, y} = { name: '小明', x: 100, y: 200});

实例

交换两个变量的值：[x,y]=[y,x]

快速获取当前页面的域名和路径：var {hostname:domain, pathname:path} = location;

如果函数把对象作为参数，可以直接把对象属性绑定到变量上，如输入Date对象

function buildDate({year, month, day, hour=0, minute=0, second=0}){...}

* 1. 方法

绑定到对象上的函数称为方法，在一个方法内部，this是一个特殊变量，它始终指向当前对象

用var that = this;，

var xiaoming = {

name: '小明',

birth: 1990,

age: function () {

var that = this; // 在方法内部一开始就捕获this

function getAgeFromBirth() {

var y = new Date().getFullYear();

return y - that.birth; // 用that而不是this

}

return getAgeFromBirth();

}

};

xiaoming.age(); // 25

apply

要指定函数的this指向哪个对象，可以用函数本身的apply方法，它接收两个参数，第一个参数就是需要绑定的this变量，第二个参数是Array，表示函数本身的参数。

function getAge() {

var y = new Date().getFullYear();

return y - this.birth;

}

var xiaoming = {

name: '小明',

birth: 1990,

age: getAge

};

xiaoming.age(); // 25

getAge.apply(xiaoming, []); // 25, this指向xiaoming, 参数为空

call

与apply()类似，区别是：

apply()把参数打包成Array再传入；

call()把参数按顺序传入。

比如调用Math.max(3, 5, 4)，分别用apply()和call()实现如下：

Math.max.apply(null, [3, 5, 4]); // 5

Math.max.call(null, 3, 5, 4); // 5

对普通函数调用，我们通常把this绑定为null。

修饰器

利用apply()，我们还可以动态改变函数的行为。JavaScript的所有对象都是动态的，即使内置的函数，我们也可以重新指向新的函数。

var count = 0;

var oldParseInt = parseInt; // 保存原函数

window.parseInt = function () {

count += 1;

return oldParseInt.apply(null, arguments); // 调用原函数

};

1. 标准对象

number、string、boolean、function和undefined是单独的对象类型。

特别注意null、Array、{}、[]的类型是object

typeof 123; // 'number'

* 1. 包装对象

number、boolean和string都有包装对象，用new创建，产生的包装对象类型为object，所以不要这样用！

可以的用法：

parseInt()或parseFloat()来转换任意类型到number；

用String()来转换任意类型（除了null和undefined）到string，或者直接调用某个对象的toString()方法，注意number直接调用会报错；

123..toString(); // '123', 注意是两个点！

(123).toString(); // '123'

通常不必把任意类型转换为boolean再判断，因为可以直接写if (myVar) {...}；

typeof操作符可以判断出number、boolean、string、function和undefined；

判断Array要使用Array.isArray(arr)；

判断null请使用myVar === null；

判断某个全局变量是否存在用typeof window.myVar === 'undefined'；

函数内部判断某个变量是否存在用typeof myVar === 'undefined'。

* 1. RegExp正则表达式

创建正则表达式，可以使用/…/，也可以调用RegExp()，第二种由于转义字符\要写成\\

var re1 = /ABC\-001/;

var re2 = new RegExp('ABC\\-001');

正则语法

详细的参考：<https://www.runoob.com/regexp/regexp-syntax.html>

常用函数/关键字

|  |  |  |
| --- | --- | --- |
| test | 判断匹配 | re.test('010-12345'); // true |
| split | 切分字符串 | 'a,b;; c d'.split(/[\s\,\;]+/); // ['a', 'b', 'c', 'd'] |
| exec | 用括号定义分组，返回Array[匹配的整个串，子串…] | var re = /^(\d{3})-(\d{3,8})$/;  re.exec('010-12345'); // ['010-12345', '010', '12345'] |
| g | 全局匹配 | var r1 = /test/g; // 等价于:  var r2 = new RegExp('test', 'g'); |
| i | 忽略大小写 |  |
| m | 执行多次匹配 |  |

1. 面向对象编程
   1. 创建对象

构造函数

法一：

定义构造函数之后使用关键字new调用，共享的函数存放在构造函数对引发的原型prototype上

function Student(name) {

this.name = name;

}

Student.prototype.hello = function () {

alert('Hello, ' + this.name + '!');

};

var xiaoming = new Student('小明');

xiaoming.name; // '小明'

xiaoming.hello(); // Hello, 小明!

法二：推荐！

重构代码，编写createStudent()函数，在内部封装所有的new操作。

优点：1）不需要new来调用，2）参数非常灵活

function Student(props) {

this.name = props.name || '匿名'; // 默认值为'匿名'

this.grade = props.grade || 1; // 默认值为1

}

Student.prototype.hello = function () {

alert('Hello, ' + this.name + '!');

};

function createStudent(props) {

return new Student(props || {})

}

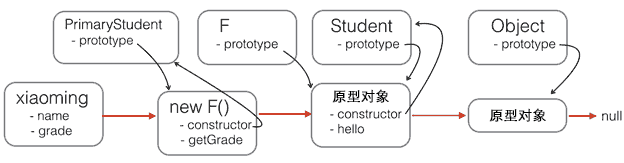
var xiaoming = createStudent({

name: '小明'

});

* 1. 原型继承

定义空函数F来桥接



把继承这个动作用一个inherits()函数封装起来，代码复用

function inherits(Child, Parent) {

var F = function () {};

F.prototype = Parent.prototype;

Child.prototype = new F();

Child.prototype.constructor = Child;

}

实现原型继承，内部用call()调用希望“继承”的构造函数，并绑定this

function Student(props) {

this.name = props.name || 'Unnamed';

}

Student.prototype.hello = function () {

alert('Hello, ' + this.name + '!');

}

function PrimaryStudent(props) {

Student.call(this, props);

this.grade = props.grade || 1;

}

// 实现原型继承链:

inherits(PrimaryStudent, Student);

// 绑定其他方法到PrimaryStudent原型:

PrimaryStudent.prototype.getGrade = function () {

return this.grade;

};

* 1. Class继承

比上一节更简单的写法

创建类：

class Student {

constructor(name) {

this.name = name;

}

hello() {

alert('Hello, ' + this.name + '!');

}

}

创建实例：

var xiaoming = new Student('小明');

继承extends

class PrimaryStudent **extends** Student {

constructor(name, grade) {

super(name); // 记得用super调用父类的构造方法!

this.grade = grade;

}

myGrade() {

alert('I am at grade ' + this.grade);

}

}

1. 浏览器
   1. 浏览器对象

一些实例：<https://www.runoob.com/js/js-ex-browser.html>

window

window对象不但充当全局作用域，而且表示浏览器窗口。

有innerWidth和innerHeight属性，可以获取浏览器窗口的内部宽度和高度。

内部宽高是指除去菜单栏、工具栏、边框等占位元素后，用于显示网页的净宽高。

兼容不同浏览器的宽高属性获取：

var w=window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

var h=window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

navigator

navigator对象表示浏览器的信息，最常用的属性包括：

navigator.appName：浏览器名称；

navigator.appVersion：浏览器版本；

navigator.language：浏览器设置的语言；

navigator.platform：操作系统类型；

navigator.userAgent：浏览器设定的User-Agent字符串。

screen

screen对象表示屏幕的信息，常用的属性有：

screen.width：屏幕宽度，以像素为单位；

screen.height：屏幕高度，以像素为单位；

screen.colorDepth：返回颜色位数，如8、16、24。

location

location对象表示当前页面的URL信息。

location.href; //获取完整URL

location.protocol; // 'http'

location.host; // 'www.example.com'

location.port; // '8080'

location.pathname; // '/path/index.html'

location.search; // '?a=1&b=2'

location.hash; // 'TOP'

location.assign() 加载一个新页面

location.reload() 重新加载当前页面

document

document对象表示当前页面，是整个DOM树的根节点。

document.title属性是从HTML文档中的<title>xxx</title>读取的，但是可以动态改变：

document.title = '努力学习JavaScript!';

document.getElementById()，按ID获得一个DOM节点

document.getElementsByTagName()，按Tag名称获得一组DOM节点

document.cookie，读取到当前页面的Cookie，服务器端在设置Cookie时，使用httpOnly，使Cookie不被JavaScript读取

history

history对象保存了浏览器的历史记录，JavaScript可以调用

history.back() 浏览器的“后退”

history.forward () 浏览器的”前进”

不推荐再使用

* 1. 操作DOM

浏览器解析HTML文档后产生DOM树，对DOM节点的操作有：

更新：更新该DOM节点的内容，相当于更新了该DOM节点表示的HTML的内容；

遍历：遍历该DOM节点下的子节点，以便进行进一步操作；

添加：在该DOM节点下新增一个子节点，相当于动态增加了一个HTML节点；

删除：删除该DOM节点的内容以及它包含的所有子节点。

获取DOM节点

法一：

|  |  |
| --- | --- |
| getElementById() | 按ID获得节点 |
| getElementsByTagName() | 按Tag获得节点 |
| getElementsByClassName() | 按ClassName获得节点 |
| children | 获得所有子节点 |
| firstElementChild | 获得第一个子节点 |
| lastElementChild | 获得最后一个子节点 |

实例：

// 先定位ID为'test-table'的节点，再返回其内部所有tr节点：

var trs = document.getElementById('test-table').getElementsByTagName('tr');

// 先定位ID为'test-div'的节点，再返回其内部所有class包含red的节点：

var reds = document.getElementById('test-div').getElementsByClassName('red');

// 获取节点test下的所有直属子节点:

var cs = test.children;

// 获取节点test下第一个、最后一个子节点：

var first = test.firstElementChild;

var last = test.lastElementChild;

法二：

|  |  |
| --- | --- |
| querySelector () | 按ID获得节点 |
| querySelectorAll (‘表达式’) | 获得符合条件的节点 |

实例：

// 通过querySelector获取ID为q1的节点：

var q1 = document.querySelector('#q1');

// 通过querySelectorAll获取q1节点内的符合条件的所有节点：

var ps = q1.querySelectorAll('div.highlighted > p');

// 通过querySelectorAll获取class包含c-red的节点下的的所有子节点：

var arr = document.querySelectorAll('.c-red')[1].children; //返回多个节点时

更新DOM节点

修改节点文本

法一：修改innerHTML属性

不光能修改文本，也可以直接编辑HTML代码，注意，是否需要写入HTML。如果写入的字符串是通过网络拿到的，要注意对字符编码来避免XSS攻击。

// 获取<p id="p-id">...</p>

var p = document.getElementById('p-id');

// 设置文本为abc:

p.innerHTML = 'ABC';

// <p id="p-id">ABC</p>

// 设置HTML:

p.innerHTML = 'ABC <span style="color:red">RED</span> XYZ';

// <p>...</p>的内部结构已修改

法二：修改innerText或textContent属性

两者的区别在于读取属性时，innerText不返回隐藏元素的文本，而textContent返回所有文本。

// 获取<p id="p-id">...</p>

var p = document.getElementById('p-id');

// 设置文本:

p.innerText = '<script>alert("Hi")</script>';

// HTML被自动编码，无法设置一个<script>节点:

// <p id="p-id">&lt;script&gt;alert("Hi")&lt;/script&gt;</p>

修改CSS

DOM节点的style属性对应所有的CSS，可以直接获取或设置。

注意CSS属性要写成驼峰式，例如font-size写成fontSize

// 获取<p id="p-id">...</p>

var p = document.getElementById('p-id');

// 设置CSS:

p.style.color = '#ff0000';

p.style.fontSize = '20px';

p.style.paddingTop = '2em';

插入DOM节点

如果节点为空，可以直接修改innerHTML属性

如果节点非空，有两种方法

**法一**：appendChild，添加一个子节点到最后

把已经存在的节点插入时，会删除原节点，插入到新位置

动态添加新节点：document.createElement()

var

list = document.getElementById('list'),

haskell = document.createElement('p'); //创建节点，Tag为p

haskell.id = 'haskell'; //设置节点ID为haskhell

haskell.innerText = 'Haskell'; //修改节点文本内容为haskhell

list.appendChild(haskell); //插入到list节点的子节点末尾

具体用途：在head节点末尾添加style节点，可以直接动态给文档添加CSS定义

var d = document.createElement('style');

d.setAttribute('type', 'text/css');

d.innerHTML = 'p { color: red }';

document.getElementsByTagName('head')[0].appendChild(d);

**法二**：insertBefore，添加子节点到指定位置

parentElement.insertBefore(newElement, referenceElement);子节点会插入到referenceElement之前。

var

list = document.getElementById('list'),

ref = document.getElementById('python'),

haskell = document.createElement('p');

haskell.id = 'haskell';

haskell.innerText = 'Haskell';

list.insertBefore(haskell, ref);

参考子节点的获取：遍历，定位到第i个子节点，然后插入

var

i, c,

list = document.getElementById('list');

for (i = 0; i < list.children.length; i++) {

c = list.children[i]; // 拿到第i个子节点

}

练习：对节点按文本内容重排序

HTML结构：

<!-- HTML结构 -->

<ol id="test-list">

<li class="lang">Scheme</li>

<li class="lang">JavaScript</li>

<li class="lang">Python</li>

<li class="lang">Ruby</li>

<li class="lang">Haskell</li>

</ol>

\*排序：

var arr\_list=Array.from(document.getElementById('test-list').children)

//Array.from,通过给定的对象创建一个数组

var arr\_lang=Array.from(document.getElementsByClassName("lang"),item=>item.innerText)

//=>箭头语法和映射函数更改元素的值

arr\_lang.sort();

//对文本数组排序

arr\_list.map((x,i)=>{x.innerText=arr\_lang[i]})

//map方法，通过指定函数处理数组的每个元素，并返回处理后的数组。

//Array.from可以用扩展运算符[...]代替

删除DOM节点

获得该节点本身以及它的父节点parentElement，然后，调用父节点的removeChild把自己删掉

// 拿到待删除节点:

var self = document.getElementById('to-be-removed');

// 拿到父节点:

var parent = self.parentElement;

// 删除:

var removed = parent.removeChild(self);

removed === self; // true

删除多个子节点时注意，每执行一次removeChild，父节点的children属性都会发生变化

练习\*批量删除

HTML结构

<ul id="test-list">

<li>JavaScript</li>

<li>Swift</li>

<li>HTML</li>

<li>ANSI C</li>

<li>CSS</li>

<li>DirectX</li>

</ul>

删除到只剩三个节点

var arr = ['JavaScript','HTML','CSS'];

var parent = document.getElementById('test-list');

for (var i of parent.children) { //遍历所有子节点

if (!arr.includes(i.innerText)) {

parent.removeChild(i); //删除值不包含在arr内的子节点

}

}

* 1. 操作表单

表单也是一种DOM树

表单输入控件<input>节点

|  |  |  |
| --- | --- | --- |
| 文本框 | <input type="text"> | 用于输入文本 |
| 口令框 | <input type="password"> | 用于输入口令 |
| 单选框 | <input type="radio"> | 用于选择一项 |
| 复选框 | <input type="checkbox"> | 用于选择多项 |
| 下拉框 | <select> | 用于选择一项 |
| 隐藏文本 | <input type="hidden"> | 用户不可见，单表单提交时会发到服务器 |

获取值和设置值

对于text password hidden select，直接调用value属性

// <input type="text" id="email">

var input = document.getElementById('email');

input.value; // '用户输入的值'

对于radio和checkbox，value属性是HTML预设值。

需要获取的值是该选项有没有被勾选，可以用checked属性为true和false判断

// <label><input type="radio" name="weekday" id="monday" value="1"> Monday</label>

// <label><input type="radio" name="weekday" id="tuesday" value="2"> Tuesday</label>

var mon = document.getElementById('monday');

var tue = document.getElementById('tuesday');

mon.value; // '1'

tue.value; // '2'

mon.checked; // true或者false

tue.checked; // true或者false

HTML5控件

date、datetime、datetime-local、color，都使用input标签

<input type="date" value="2015-07-01"> //格式是YYYY-MM-DD

<input type="datetime-local" value="2015-07-01T02:03:04">

<input type="color" value="#ff0000">

提交表单

法一：通过<form>元素的submit()方法

实例：响应按钮点击事件

<!-- HTML -->

<form id="test-form">

<input type="text" name="test">

<button type="button" onclick="doSubmitForm()">Submit</button>

</form>

<script>

function doSubmitForm() {

var form = document.getElementById('test-form');

// 可以在此修改form的input...

// 提交form:

form.submit();

}

</script>

法二：响应<form>本身的onsubmit事件

<!-- HTML -->

<form id="test-form" onsubmit="return checkForm()">

<input type="text" name="test">

<button type="submit">Submit</button>

</form>

<script>

function checkForm() {

var form = document.getElementById('test-form');

// 可以在此修改form的input...

// 继续下一步:

return true;

}

</script>

一般使用法二，因为法一对表单的修改仅在点击事件时发生

隐藏文本应用

很多登录表单希望用户输入用户名和口令，但出于安全考虑，提交表单时不传输明文口令，而是口令的MD5。

<!-- HTML -->

<form id="login-form" method="post" onsubmit="return checkForm()">

<input type="text" id="username" name="username">

<input type="password" id="input-password">

<input type="hidden" id="md5-password" name="password"> //有name属性的数据才会被提交

<button type="submit">Submit</button>

</form>

<script>

function checkForm() {

var input\_pwd = document.getElementById('input-password');

var md5\_pwd = document.getElementById('md5-password');

// 把用户输入的明文变为MD5:

md5\_pwd.value = toMD5(input\_pwd.value);

// 继续下一步:

return true;

}

</script>

练习

判断表单输入是否符合要求：用户名必须是3-10位英文字母或数字；口令必须是6-20位；两次输入口令必须一致。

<!-- HTML结构 -->

<form id="test-register" action="#" target="\_blank" onsubmit="return checkRegisterForm()">

<p id="test-error" style="color:red"></p>

<p>

用户名: <input type="text" id="username" name="username">

</p>

<p>

口令: <input type="password" id="password" name="password">

</p>

<p>

重复口令: <input type="password" id="password-2">

</p>

<p>

<button type="submit">提交</button> <button type="reset">重置</button>

</p>

</form>

判断，符合要求才能成功提交

'use strict';

var checkRegisterForm = function () {

let username = document.getElementById("username");

let pwd\_1 = document.getElementById("password");

let pwd\_2 = document.getElementById("password-2");

const reUsername = /^[0-9A-Za-z]{3,10}$/; //正则表达式

const rePwd = /^[0-9A-Za-z]{6,20}$/;

if(!reUsername.exec(username.value) || !rePwd.exec(pwd\_1.value) || pwd\_1.value !== pwd\_2.value){

alert("please recheck your password or username!")

return false;

}

* 1. 操作文件

在HTML表单中，可以上传文件的唯一控件就是<input type="file">。

注意：当一个表单包含<input type="file">时，表单的enctype必须指定为multipart/form-data，method必须指定为post，浏览器才能正确编码并以multipart/form-data格式发送表单的数据。

出于安全考虑，浏览器只允许用户点击<input type="file">来选择本地文件，用JavaScript对<input type="file">的value赋值是没有任何效果的。当用户选择了上传某个文件后，JavaScript也无法获得该文件的真实路径

通过后缀检测无效格式：

var f = document.getElementById('test-file-upload');

var filename = f.value; // 'C:\fakepath\test.png'

if (!filename || !(filename.endsWith('.jpg') || filename.endsWith('.png') || filename.endsWith('.gif'))) {

alert('Can only upload image file.');

return false;

}

* 1. AJAX执行异步网络请求

<https://www.runoob.com/ajax/ajax-tutorial.html>

AJAX 不是新的编程语言，而是一种使用现有标准的新方法。

AJAX 最大的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。

在现代浏览器上写AJAX主要依靠XMLHttpRequest对象：

'use strict';

function success(text) {

var textarea = document.getElementById('test-response-text');

textarea.value = text;

}

function fail(code) {

var textarea = document.getElementById('test-response-text');

textarea.value = 'Error code: ' + code;

}

var request = new XMLHttpRequest(); // 新建XMLHttpRequest对象

request.onreadystatechange = function () { // 状态发生变化时，函数被回调

if (request.readyState === 4) { // 成功完成

// 判断响应结果:

if (request.status === 200) {

// 成功，通过responseText拿到响应的文本:

return success(request.responseText);

} else {

// 失败，根据响应码判断失败原因:

return fail(request.status);

}

} else {

// HTTP请求还在继续...

}

}

// 发送请求:

request.open('GET', '/api/categories');

request.send();

alert('请求已发送，请等待响应...');

* 1. Promise

[https://es6.ruanyifeng.com/#docs/promise](https://es6.ruanyifeng.com/%23docs/promise)

Promise().then().then....catch() 多任务串行执行.

情景化记忆:在一个任务链中,比如我要向上级部门一层层的往上提交申请,if(某种条件)承诺帮你resolve解决问题,else承诺reject你的请求. 他给出的resolve问题的办法只是个空头Promise,then到总经理那实现具体承诺,如果总经理还是给一个空头承诺(返回Promise实例),还得then到董事长那里.... 任一一步做出的是reject的承诺,还有什么好说的,被拒绝了,后面的就不会再往上走了呀. 准备catch 拒绝通知吧blablabla

Promise.all([p1,p2,...]) 多任务并行执行

都要成功才进入then,返回结果数组.

Promise.race([p1,p2,...]) 多任务赛跑.

then()和catch(),谁先调用算谁的,其它任务中断.

* 1. Canvas

<https://www.runoob.com/html/html5-canvas.html>

<https://www.runoob.com/w3cnote/html5-canvas-intro.html>

1. jQuery

<https://www.runoob.com/jquery/jquery-tutorial.html>

基本功能：

* 消除浏览器差异：你不需要自己写冗长的代码来针对不同的浏览器来绑定事件，编写AJAX等代码；
* 简洁的操作DOM的方法：写$('#test')肯定比document.getElementById('test')来得简洁；
* 轻松实现动画、修改CSS等各种操作。

使用方法：

在<head>中引入jQuery

<html>

<head>

<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>

...

</head>

<body>

...

</body>

</html>

jQuery把所有功能全部封装在一个全局变量jQuery中，$符号是变量jQuery的别名。$本质上就是一个函数，但是函数也是对象，于是$除了可以直接调用外，也可以有很多其他属性。

如果想把$符号用作其他用途，可以使用jQuery.noConflict()，使$变为undefined类型。原理是jQuery在占用$之前，先在内部保存了原来的$,调用jQuery.noConflict()时会把原来保存的变量还原。

* 1. 选择器

<https://www.runoob.com/jquery/jquery-ref-selectors.html>

快速定位到一个或多个DOM节点。比上文提到的调用document方法写起来简洁

1.基本选择器

$("#id") //ID选择器

$("div") //元素选择器

$(".classname") //类选择器

$(".classname.classname1") //类选择器,与

$(".classname,.classname1,#id1") //组合选择器，或

2.层次选择器

$("#id>.classname ") //子元素选择器

$("#id .classname ") //后代元素选择器

$("#id + .classname ") //紧邻下一个元素选择器

$("#id ~ .classname ") //兄弟元素选择器

3.过滤选择器(重点)

$("li:first") //第一个li

$("li:last") //最后一个li

$("li:even") //挑选下标为偶数的li

$("li:odd") //挑选下标为奇数的li

$("li:eq(4)") //下标等于 4 的li(第五个 li 元素)

$("li:gt(2)") //下标大于 2 的li

$("li:lt(2)") //下标小于 2 的li

$("li:not(#runoob)") //挑选除 id="runoob" 以外的所有li

3.2内容过滤选择器

$("div:contains('Runob')") // 包含 Runob文本的元素

$("td:empty") //不包含子元素或者文本的空元素

$("div:has(selector)") //含有选择器所匹配的元素

$("td:parent") //含有子元素或者文本的元素

3.3可见性过滤选择器

$("li:hidden") //匹配所有不可见元素，或type为hidden的元素

$("li:visible") //匹配所有可见元素

3.4属性过滤选择器

$("div[id]") //所有含有 id 属性的 div 元素

$("div[id='123']") // id属性值为123的div 元素

$("div[id!='123']") // id属性值不等于123的div 元素

$("div[id^='qq']") // id属性值以qq开头的div 元素

$("div[id$='zz']") // id属性值以zz结尾的div 元素

$("div[id\*='bb']") // id属性值包含bb的div 元素

$("input[id][name$='man']") //多属性选过滤，同时满足两个属性的条件的元素

3.5状态过滤选择器

$("input:enabled") // 匹配可用的 input

$("input:disabled") // 匹配不可用的 input

$("input:checked") // 匹配选中的 input

$("option:selected") // 匹配选中的 option

4.表单选择器

$(":input") //匹配所有 input, textarea, select 和 button 元素

$(":text") //所有的单行文本框，$(":text") 等价于$("[type=text]")，推荐使用$("input:text")效率更高，下同

$(":password") //所有密码框

$(":radio") //所有单选按钮

$(":checkbox") //所有复选框

$(":submit") //所有提交按钮

$(":reset") //所有重置按钮

$(":button") //所有button按钮

$(":file") //所有文件域

5. 查找

find()查找：

var dy = ul.find('.dy'); // 按类查找

var swf = ul.find('#swift'); // 按id查找

var hsk = ul.find('[name=haskell]'); // 按条件过滤查找

parent()从当前节点向上查找：

var parent = swf.parent(); // 获得Swift的上层节点

var a = swf.parent('.red'); // 获得Swift的上层节点，同时传入过滤条件。如果不符合条件，返回空jQuery对象

next()和prev()方法查找位于同一层级的节点：

swift.next(); // 下一个节点

swift.next('[name=haskell]'); // 下一个节点，且传入过滤条件

swift.prev(); // 前一个节点

swift.prev('.dy'); // 前一个节点，且传入过滤条件

6. 过滤

filter()方法，可以过滤掉不符合选择器条件的节点：

var a = langs.filter('.dy'); // 过滤出langs下类为dy的节点

或者传入函数，要特别注意函数内部的this被绑定为DOM对象，不是jQuery对象：

langs.filter(function () {

return this.innerHTML.indexOf('S') === 0; // 返回S开头的节点

});

map()方法，把一个jQuery对象包含的若干DOM节点转化为其他对象：

var arr = langs.map(function () {

return this.innerHTML;

}).get(); // 过滤出节点，并返回每个节点的文本对象

first()、last()和slice()方法，对于包含多个DOM节点的jQuery对象，返回一个新的jQuery对象，把不需要的DOM节点去掉：

var js = langs.first(); // 取第一个子节点，相当于$('ul.lang li:first-child')

var haskell = langs.last(); // 取最后一个子节点, 相当于$('ul.lang li:last-child')

var sub = langs.slice(2, 4); // 索引2到4，不包括4, 参数和数组的slice()方法一致

练习

<form id="test-form" action="#0" onsubmit="return false;">

<p><label>Name: <input name="name"></label></p>

<p><label>Email: <input name="email"></label></p>

<p><label>Password: <input name="password" type="password"></label></p>

<p>Gender: <label><input name="gender" type="radio" value="m" checked> Male</label> <label><input name="gender" type="radio" value="f"> Female</label></p>

<p><label>City: <select name="city">

<option value="BJ" selected>Beijing</option>

<option value="SH">Shanghai</option>

<option value="CD">Chengdu</option>

<option value="XM">Xiamen</option>

</select></label></p>

<p><button type="submit">Submit</button></p>

</form>

输入值后，用jQuery获取表单的JSON字符串，key和value分别对应每个输入的name和相应的value，例如：{"name":"Michael","email":...}

'use strict';

var json = null;

let formData = {}; //创建空对象

$('#test-form label [name]').map(function () { //map

formData[this.name] = this.value; //this指向过滤出的DOM对象，构造成formData的键值对

});

json = JSON.stringify(formData);

* 1. 操作DOM

<https://www.runoob.com/jquery/jquery-ref-html.html>

修改文本

jQuery对象的text()和html()方法分别获取节点的文本和原始HTML文本。无参数调用text()是获取文本，传入参数就变成设置文本，HTML同理。一个jQuery对象可以包含0个或任意个DOM对象，它的方法实际上会作用在对应的每个DOM节点上。

修改CSS

jQuery对象的css('name', 'value')方法，可以修改CSS

$('#test-css li.dy>span').css('background-color', '#ffd351').css('color', 'red'); //修改背景色和字色

addClass()方法可以修改class属性

var div = $('#test-div');

div.hasClass('highlight'); // false， class是否包含highlight

div.addClass('highlight'); // 添加highlight这个class

div.removeClass('highlight'); // 删除highlight这个class

练习：两种方法使文本高亮

//css方法

$('#test-highlight-css li.js>span').css('background-color', '#ffd351').css('color', 'red');

//addClass方法

$("#test-highlight-css li.js").addClass('highlight');

显示和隐藏DOM

调用jQuery对象的show() hide()方法

var a = $('a[target=\_blank]');

a.hide(); // 隐藏

a.show(); // 显示

获取DOM信息

// 浏览器可视窗口大小:

$(window).width(); // 800

$(window).height(); // 600

// HTML文档大小:

$(document).width(); // 800

$(document).height(); // 3500

// 某个div的大小:

var div = $('#test-div');

div.width(); // 600

div.height(); // 300

div.width(400); // 设置CSS属性

div.height('200px'); // 设置CSS属性

attr()和removeAttr()方法用于操作DOM节点的属性：

// <div id="test-div" name="Test" start="1">...</div>

var div = $('#test-div');

div.attr('data'); // undefined, 属性不存在

div.attr('name'); // 'Test'

div.attr('name', 'Hello'); // div的name属性变为'Hello'

div.removeAttr('name'); // 删除name属性

div.attr('name'); // undefined

prop()方法应该用于检索属性值，和attr()类似，检索HTML属性时还是使用attr()

处理属性checked时有所不同：

var radio = $('#test-radio');

radio.attr('checked'); // 'checked'

radio.prop('checked'); // true

prop()返回值更合理一些。不过，用is()方法判断更好：

radio.is(':checked'); // true

selected属性同理，处理时最好用is(':selected')。

操作表单

对于表单元素，jQuery对象统一提供val()方法获取和设置对应的value属性：

无参数调用时获取文本，传入参数时设置文本

修改DOM结构

append()方法

传入HTML片段，DOM对象，jQuery对象和函数对象。

append()把DOM添加到最后，prepend()则把DOM添加到最前。

使用该方法添加节点时，如果节点已存在，则会删除源节点，效果上就是移动了节点

获取jQury对象

var ul = $('#test-div>ul');

//传入HTML片段：

ul.append('<li><span>Haskell</span></li>');

// 创建DOM对象:

var ps = document.createElement('li');

ps.innerHTML = '<span>Pascal</span>';

// 添加DOM对象:

ul.append(ps);

// 添加jQuery对象:

ul.append($('#scheme'));

// 添加函数对象:

ul.append(function (index, html) {

return '<li><span>Language - ' + index + '</span></li>';

});

after()或者before()方法

对同级节点，把新节点插入到指定位置

var js = $('#test-div>ul>li:first-child');

js.after('<li><span>Lua</span></li>');

删除节点

要删除DOM节点，拿到jQuery对象后直接调用remove()方法就可以了。如果jQuery对象包含若干DOM节点，实际上可以一次删除多个DOM节点：

var li = $('#test-div>ul>li');

li.remove(); // 所有<li>全被删除

练习 排序

<!-- HTML结构 -->

<div id="test-div">

<ul>

<li><span>JavaScript</span></li>

<li><span>Python</span></li>

<li><span>Swift</span></li>

</ul>

</div>

除了列出的3种语言外，请再添加Pascal、Lua和Ruby，然后按字母顺序排序节点：

法一：

//加入三种语言

var ul = $("#test-div>ul");

ul.append("<li><span>Pascal</span></li>");

ul.append("<li><span>Lua</span></li>");

ul.append("<li><span>Ruby</span></li>");

//过滤出语言节点

var li = ul.find("li");

li.sort((x, y) => {

return $(x).text() > $(y).text() ? 1 : -1;

});

ul.append(li);

三目运算符 A?B:C 如果满足A，执行B，否则执行C

说明：sort(function(a,b){return a-b}) 表示升序排序

Sort方法中，返回值为-1时表示a<b，返回值为1时表示a>b，上面的写法功能为按文本字母的顺序排序

详见文档：<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/sort>

法二：

//提取文本数组，加入三种语言，并排序

li=Array.from($("#test-div>ul>li>span"))

val=li.map(x=>x.innerHTML)

val.push("Pascal")

val.push("Lua")

val.push("Ruby")

val.sort()

//删除原节点

$("#test-div>ul>li").remove()

//加入新节点

val.map(x=>$("#test-div>ul").append(` <li><span>${x}</span></li>`)) //记一下

* 1. 事件

<https://www.runoob.com/jquery/jquery-ref-events.html>

jQuery能够绑定的事件主要包括：

**鼠标事件**

* click: 鼠标单击时触发；
* dblclick：鼠标双击时触发；
* mouseenter：鼠标进入时触发；
* mouseleave：鼠标移出时触发；
* mousemove：鼠标在DOM内部移动时触发；
* hover：鼠标进入和退出时触发两个函数，相当于mouseenter加上mouseleave。

**键盘事件**

键盘事件仅作用在当前焦点的DOM上，通常是<input>和<textarea>。

* keydown：键盘按下时触发；
* keyup：键盘松开时触发；
* keypress：按一次键后触发。

**其他事件**

* focus：当DOM获得焦点时触发；
* blur：当DOM失去焦点时触发；
* change：当<input>、<select>或<textarea>的内容改变时触发；
* submit：当<form>提交时触发；
* ready：当页面被载入并且DOM树完成初始化后触发。

Ready事件

在DOM完成初始化后触发，且只触发一次，所以非常适合用来写其他的初始化代码。

<html>

<head>

<script>

$(document).on('ready', function () {

$('#testForm).on('submit', function () {

alert('submit!');

});

});

</script>

</head>

<body>

<form id="testForm">

...

</form>

</body>

简化一：

$(document).ready(function () {

// on('submit', function)也可以简化:

$('#testForm).submit(function () {

alert('submit!');

});

});

简化二：最常用 记住！

$(function () {

// init...

});

事件参数

有些事件，如mousemove和keypress，我们需要获取鼠标位置和按键的值，否则监听这些事件就没什么意义了。所有事件都会传入Event对象作为参数，可以从Event对象上获取到更多的信息：

获取鼠标坐标参数

$(function () {

$('#testMouseMoveDiv').mousemove(function (e) {

$('#testMouseMoveSpan').text('pageX = ' + e.pageX + ', pageY = ' + e.pageY);

});

});

取消绑定

一个已被绑定的事件可以解除绑定，通过off('click', function)实现：

function hello() {

alert('hello!');

}

a.click(hello); // 绑定事件

// 10秒钟后解除绑定:

setTimeout(function () {

a.off('click', hello);

}, 10000);

off('click') 取消所有click事件绑定

off() 取消所有事件绑定

事件触发条件

用户输入或操作时，会触发change事件，而用js代码改动不会触发change事件。

监听change事件：

var input = $('#test-input');

input.change(function () {

console.log('changed...');

});

如果希望用代码触发change：

var input = $('#test-input');

input.val('change it!');

input.change(); // 触发change事件

input.change()相当于input.trigger('change')，它是trigger()方法的简写。

练习

[https://www.liaoxuefeng.com/wiki/1022910821149312/1025427850131520#0](https://www.liaoxuefeng.com/wiki/1022910821149312/1025427850131520" \l "0)

//根据全选/全不选控制 语言选择框

selectAll.click(function () { //这里使用change事件会导致其他选框不能修改

langs.prop('checked', $(this).is(':checked')); //将全选框的状态同步到langs

});

//全选标签修改

selectAll.change(function () {

if ($(this).is(':checked')) {

deselectAllLabel.show();

selectAllLabel.hide();

} else {

deselectAllLabel.hide();

selectAllLabel.show();

}

});

//反转 全选标签修改

invertSelect.click(function () {

langs.map(function () {

$(this).prop('checked', !$(this).is(':checked'));

});

langs.change();

});

//判断全选中

langs.change(function () {

if (langs.get().every(i=>i.checked)) {

selectAll.prop('checked', true);

} else {

selectAll.prop('checked', false);

}

selectAll.change();

});

* 1. 动画

<https://www.runoob.com/jquery/jquery-ref-effects.html>

show/ hide/ toggle 从左上角展开/收缩

slideUp/ slideDown/ slideToggle 垂直方向展开收缩

fadeIn/ fadeOut/ fadeToggle 淡入淡出

animate 自定义动画

delay 实现暂停，做串行动画

练习 删除时添加淡出效果

'use strict';

function deleteFirstTR() {

var tr = $('#test-table>tbody>tr:visible').first();

tr.fadeOut(1000,() => tr.remove());

}

deleteFirstTR();

* 1. AJAX

<https://www.runoob.com/jquery/jquery-ajax-intro.html>

<https://www.w3school.com.cn/jquery/jquery_ajax_intro.asp>

AJAX = 异步 JavaScript 和 XML（Asynchronous JavaScript and XML）。

简短地说，在不重载整个网页的情况下，AJAX 通过后台加载数据，并在网页上进行显示。

jQuery load() 方法

jQuery load()方法从服务器加载数据，并把返回的数据放入被选元素中。

语法：

$(selector).load(URL,data,callback);

实例

<!DOCTYPE html>

<html>

<head>

<script src="/jquery/jquery-1.11.1.min.js"></script>

<script>

$(document).ready(function(){

$("button").click(function(){

$("#div1").load("/example/jquery/demo\_test.txt",function(responseTxt,statusTxt,xhr){

if(statusTxt=="success")

alert("外部内容加载成功！");

if(statusTxt=="error")

alert("Error: "+xhr.status+": "+xhr.statusText);

});

});

});

</script>

</head>

<body>

<div id="div1"><h2>使用 jQuery AJAX 来改变文本</h2></div>

<button>获得外部内容</button>

</body>

</html>

jQuery get() 和 post() 方法

用于通过 HTTP GET 或 POST 请求从服务器请求数据。

GET - 从指定的资源请求数据 POST - 向指定的资源提交要处理的数据

get() 方法通过 HTTP GET 请求从服务器上请求数据。

语法：$.get(URL,callback);

//从服务器上的一个文件中取回数据：

$("button").click(function(){

$.get("demo\_test.asp",function(data,status){

alert("Data: " + data + "\nStatus: " + status);

});

});

post() 方法通过 HTTP POST 请求从服务器上请求数据。

语法：$.post(URL,data,callback);

//连同请求一起发送数据：

$("button").click(function(){

$.post("demo\_test\_post.asp",

{

name:"Donald Duck",

city:"Duckburg"

},

function(data,status){

alert("Data: " + data + "\nStatus: " + status);

});

});

GET 基本上用于从服务器获得（取回）数据。注释：GET 方法可能返回缓存数据。

POST 也可用于从服务器获取数据。不过，POST 方法不会缓存数据，并且常用于连同请求一起发送数据。

* 1. 扩展

编写jQuery插件

要点：

* 给$.fn绑定函数，实现插件的代码逻辑；
* 插件函数最后要return this;以支持链式调用；
* 插件函数要有默认值，绑定在$.fn.<pluginName>.defaults上；
* 用户在调用时可传入设定值覆盖默认值。var opts = $.extend({}, $.fn. <pluginName>.defaults, options);

实例一：

编写扩展highlight1()，实现高亮

$.fn.highlight1 = function () {

// this已绑定为当前jQuery对象:

this.css('backgroundColor', '#fffceb').css('color', '#d85030');

return this; // jQuery对象支持链式操作，扩展方法也要能继续链式下去

}

函数内部的this在调用时被绑定为jQuery对象，所以函数内部代码可以正常调用所有jQuery对象的方法。

调用highlight1：

$('#test-highlight1 span').highlight1();

实例二：

highlight2()，实现支持传参的扩展，用户可输入高亮颜色

我们可以给方法加个参数，让用户自己把参数用对象传进去。于是我们有了第二个版本的highlight2()：

$.fn.highlight2 = function (options) {

// 要考虑到各种情况:

// options为undefined

// options只有部分key

// 简单的&&和||短路操作符，总能得到一个有效的值。

var bgcolor = options && options.backgroundColor || '#fffceb';

var color = options && options.color || '#d85030';

this.css('backgroundColor', bgcolor).css('color', color);

return this;

}

调用highlight2：参数为键值对形式

$('#test-highlight2 span').highlight2({

backgroundColor: '#00a8e6',

color: '#ffffff'

});

实例三：

highlight()，允许用户自己设置**缺省值**

$.fn.highlight = function (options) {

// 合并默认值和用户设定值:

var opts = $.extend({}, $.fn.highlight.defaults, options);

this.css('backgroundColor', opts.backgroundColor).css('color', opts.color);

return this;

}

// 设定默认值:

$.fn.highlight.defaults = {

color: '#d85030',

backgroundColor: '#fff8de'

}

设置缺省值并调用：

$.fn.highlight.defaults.color = '#659f13';

$.fn.highlight.defaults.backgroundColor = '#f2fae3';

$('#test-highlight p:first-child span').highlight();

$('#test-highlight p:last-child span').highlight({

color: '#dd1144'

});

针对特定元素的扩展

借助filter()方法过滤来实现针对特定元素的扩展。

实例：

<!-- HTML结构 -->

<div id="test-external">

<p>如何学习<a href="http://jquery.com">jQuery</a>？</p>

<p>首先，你要学习<a href="/wiki/1022910821149312">JavaScript</a>，并了解基本的<a href="https://developer.mozilla.org/en-US/docs/Web/HTML">HTML</a>。</p>

</div>

要给所有指向外链的超链接加上跳转提示

$.fn.external = function () {

// return返回的each()返回结果，支持链式调用:

return this.filter('a').each(function () {

// 注意: each()内部的回调函数的this绑定为DOM本身!

var a = $(this);

var url = a.attr('href');

if (url && (url.indexOf('http://')===0 || url.indexOf('https://')===0)) {

a.attr('href', '#0')

.removeAttr('target')

.append(' <i class="uk-icon-external-link"></i>')

.click(function () {

if(confirm('你确定要前往' + url + '？')) {

window.open(url);

}

});

}

});

}

用户调用

$('#test-external a').external();

1. 错误处理
   1. 错误处理方法

try ... catch ... finally

执行try部分的代码，一旦发生错误，转跳到catch，无论有没有错误最后都要执行finally

try {

... //异常的抛出

} catch (e) {

... //异常的捕获和处理

} finally {

... //结束处理

}

//只有try…catch，没有finally：

try {

...

} catch (e) {

...

}

只有try ... finally，没有catch：

try {

...

} finally {

...

}

throw

throw实现程序主动抛出一个错误，让执行流程直接跳转到catch块。异常可以是 JavaScript 字符串、数字、逻辑值或对象。

实例：

var r, n, s;

try {

s = prompt('请输入一个数字');

n = parseInt(s); //判断是否为数字

if (isNaN(n)) {

throw new Error('输入错误');

}

// 计算平方:

r = n \* n;

console.log(n + ' \* ' + n + ' = ' + r);

} catch (e) {

console.log('出错了：' + e);

* 1. 错误传播

如果在一个函数内部发生了错误，它自身没有捕获，错误就会被抛到**外层调用函数**，如果外层函数也没有捕获，该错误会一直沿着**函数调用链向上抛出**，直到被JavaScript引擎捕获，代码终止执行。

所以，不必在每一个函数内部捕获错误，只需要在合适的地方来个统一捕获。

* 1. 异步错误处理

原则：在哪里可能出错就在哪里捕获错误。

JS很多函数是高容错性的，多层调用时，某一层抛出异常，异常逐层向外抛出，如果异常被当中的某一层所“包融”了，则这一层之外就捕捉不到异常了。

实例：

$btn.click(function () {

try {

var

x = parseFloat($('#x').val()),

y = parseFloat($('#y').val()),

r;

if (isNaN(x) || isNaN(y)) {

throw new Error('输入有误');

}

r = x + y;

alert('计算结果：' + r);

} catch (e) {

alert('输入有误！');

}

});

1. 第三方库underscore

<https://underscorejs.net/>

underscore在加载时会把自身绑定到唯一的全局变量 \_ 上

* 1. Collections

<http://underscorejs.org/#collections>

map

map\_.map(list, iteratee, [context])

通过对 list 里的每个元素调用转换函数(iteratee迭代器)生成一个与之相对应的数组。iteratee传递三个参数：value，然后是迭代 index，最后一个是引用指向整个list。

当作用于Object时，传入的函数为function (value, key)，第一个参数接收value，第二个参数接收key

\_.map([1, 2, 3], function(num){ return num \* 3; });

=> [3, 6, 9]

\_.map({one: 1, two: 2, three: 3}, function(num, key){ return num \* 3; });

=> [3, 6, 9]

\_.map([[1, 2], [3, 4]], \_.first);

=> [1, 3]

1. Node.js

<http://nodejs.cn/api/>

<https://www.runoob.com/nodejs/nodejs-tutorial.html>

* 1. 安装和环境配置

安装

<https://nodejs.org/en/>

安装Node.js的同时也会安装好nmp

windows环境下，在命令提示符输入指令

node -v， 查看版本

node 进入Node.js的交互环境，在交互环境下，可以输入任意JavaScript语句

nmp -v 查看nmp版本

nmp nmp命令提示

要退出Node.js环境，连按两次Ctrl+C。

VS Code环境

<https://code.visualstudio.com/>

VS Code以文件夹作为工程目录（Workspace Dir），所有的JavaScript文件都存放在该目录下。此外，VS Code在工程目录下还需要一个.vscode的配置目录，里面存放里VS Code需要的配置文件。

运行和调试时VS Code可以快速创建launch.json文件

* 1. 模块封装

要在模块中对外输出变量，用：

module.exports = variable;

输出的变量可以是任意对象、函数、数组等等。

要引入其他模块输出的对象，用：

var foo = require('other\_module');

引入的对象具体是什么，取决于引入模块输出的对象。

注意！引用模块要写相对地址，写模块名，例如’./hello’

* 1. fs 文件系统模块

<http://nodejs.cn/api/fs.html>

引用模块fs

var fs = require('fs');

读写文件，提供同步和异步两种方法

由于Node环境执行的JavaScript代码是服务器端代码，所以，绝大部分需要在服务器运行期反复执行业务逻辑的代码，必须使用异步代码

服务器启动时如果需要读取配置文件，或者结束时需要写入到状态文件时，可以使用同步代码，因为这些代码只在启动和结束时执行一次，不影响服务器正常运行时的异步执行。

异步读文件

读取文本文件

fs.readFile('sample.txt', 'utf-8', function (err, data) {

if (err) {

console.log(err);

} else {

console.log(data);

}

});

读取图片文件

fs.readFile('sample.png', function (err, data) {

if (err) {

console.log(err);

} else {

console.log(data);

console.log(data.length + ' bytes');

}

});

同步读文件

读取文本文件

var data = fs.readFileSync('sample.txt', 'utf-8');

console.log(data);

//捕获错误

try {

var data = fs.readFileSync('sample.txt', 'utf-8');

console.log(data);

} catch (err) {

// 出错了

}

异步写文件

var data = 'Hello, Node.js';

fs.writeFile('output.txt', data, function (err) {

if (err) {

console.log(err);

} else {

console.log('ok.');

}

});

同步写文件

var data = 'Hello, Node.js';

fs.writeFileSync('output.txt', data);

* 1. stream流

<http://nodejs.cn/api/stream.html>

在Node.js中，流也是一个对象，我们只需要响应流的事件就可以了：

data事件表示流的数据已经可以读取了

end事件表示这个流已经到末尾了，没有数据可以读取了

error事件表示出错了。

所有可以读取数据的流都继承自stream.Readable，所有可以写入的流都继承自stream.Writable。

从文件流读取文本内容：

\*要注意，data事件可能会有多次，每次传递的chunk是流的一部分数据。

'use strict';

var fs = require('fs');

// 打开一个流:

var rs = fs.createReadStream('sample.txt', 'utf-8');

rs.on('data', function (chunk) {

console.log('DATA:')

console.log(chunk);

});

rs.on('end', function () {

console.log('END');

});

rs.on('error', function (err) {

console.log('ERROR: ' + err);

});

以流的形式写入文件：

\*不断调用write()方法，最后以end()结束

'use strict';

var fs = require('fs');

var ws1 = fs.createWriteStream('output1.txt', 'utf-8');

ws1.write('使用Stream写入文本数据...\n');

ws1.write('END.');

ws1.end();

var ws2 = fs.createWriteStream('output2.txt');

ws2.write(new Buffer('使用Stream写入二进制数据...\n', 'utf-8'));

ws2.write(new Buffer('END.', 'utf-8'));

ws2.end();

pipe 复制文件

一个Readable流和一个Writable流串起来后，所有的数据自动从Readable流进入Writable流，这种操作叫pipe。

用pipe()把一个文件流和另一个文件流串起来，这样源文件的所有数据就自动写入到目标文件里了，所以，这实际上是一个复制文件的程序：

'use strict';

var fs = require('fs');

var rs = fs.createReadStream('sample.txt');

var ws = fs.createWriteStream('copied.txt');

rs.pipe(ws);

默认情况下，当Readable流的数据读取完毕，end事件触发后，将自动关闭Writable流。如果我们不希望自动关闭Writable流，需要传入参数：

readable.pipe(writable, { end: false });

* 1. HTTP

HTTP服务器

'use strict';

// 导入http模块:

var http = require('http');

// 创建http server，并传入回调函数:

var server = http.createServer(function (request, response) {

// 回调函数接收request和response对象,

// 获得HTTP请求的method和url:

console.log(request.method + ': ' + request.url);

// 将HTTP响应200写入response, 同时设置Content-Type: text/html:

response.writeHead(200, {'Content-Type': 'text/html'});

// 将HTTP响应的HTML内容写入response:

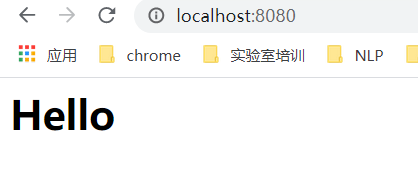
response.end('<h1>Hello</h1>');

});

// 让服务器监听8080端口:

server.listen(8080);

console.log('Server is running at http://127.0.0.1:8080/');

文件服务器

解析request.url中的路径，然后在本地找到对应的文件，把文件内容发送出去就实现文件服务器。

解析URL

url模块，通过parse()将一个字符串解析为一个Url对象：

var url = require('url');

console.log(url.parse('http://user:pass@host.com:8080/path/to/file?query=string#hash'));

处理本地文件目录

path模块，它可以方便地构造目录：

var path = require('path');

// 解析当前目录:

var workDir = path.resolve('.'); // '/Users/michael'

// 组合完整的文件路径:当前目录+'pub'+'index.html':

var filePath = path.join(workDir, 'pub', 'index.html');

// '/Users/michael/pub/index.html'

实现文件服务器file\_server.js：

'use strict';

var

fs = require('fs'),

url = require('url'),

path = require('path'),

http = require('http');

// 从命令行参数获取root目录，默认是当前目录:

var root = path.resolve(process.argv[2] || '.');

console.log('Static root dir: ' + root);

// 创建服务器:

var server = http.createServer(function (request, response) {

// 获得URL的path，类似 '/css/bootstrap.css':

var pathname = url.parse(request.url).pathname;

// 获得对应的本地文件路径，类似 '/srv/www/css/bootstrap.css':

var filepath = path.join(root, pathname);

// 获取文件状态:

fs.stat(filepath, function (err, stats) {

if (!err && stats.isFile()) {

// 没有出错并且文件存在:

console.log('200 ' + request.url);

// 发送200响应:

response.writeHead(200);

// 将文件流导向response:

fs.createReadStream(filepath).pipe(response);

} else {

// 出错了或者文件不存在:

console.log('404 ' + request.url);

// 发送404响应:

response.writeHead(404);

response.end('404 Not Found');

}

});

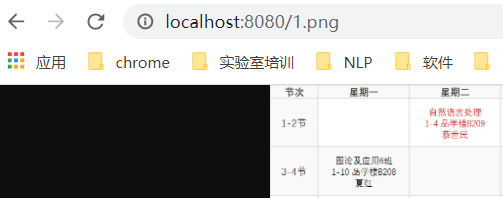
});

server.listen(8080);

console.log('Server is running at http://127.0.0.1:8080/');

在命令行运行node file\_server.js /path/to/dir，把/path/to/dir改成你本地的一个有效的目录

在浏览器中输入<http://localhost:8080/filename>，测试能否正确返回文件



* 1. Crypto 加密和哈希

<https://www.liaoxuefeng.com/wiki/1022910821149312/1023025778520640>

<http://nodejs.cn/api/crypto.html>

1. Web开发

Web框架：[Express](http://expressjs.com/" \t "_blank)，[Sails.js](http://sailsjs.org/" \t "_blank)，[koa](http://koajs.com/" \t "_blank)，[Meteor](https://www.meteor.com/" \t "_blank)，[DerbyJS](http://derbyjs.com/" \t "_blank)，[Total.js](https://www.totaljs.com/" \t "_blank)，[restify](http://restify.com/" \t "_blank)……

ORM框架：[Sequelize](http://www.sequelizejs.com/" \t "_blank)，[ORM2](http://dresende.github.io/node-orm2/" \t "_blank)，[Bookshelf.js](http://bookshelfjs.org/" \t "_blank)，[Objection.js](http://vincit.github.io/objection.js/" \t "_blank)……

模版引擎：[Jade](http://jade-lang.com/" \t "_blank)，[EJS](http://ejs.co/" \t "_blank)，[Swig](https://github.com/paularmstrong/swig" \t "_blank)，[Nunjucks](http://mozilla.github.io/nunjucks/" \t "_blank)，[doT.js](http://olado.github.io/doT/" \t "_blank)……

测试框架：[Mocha](http://mochajs.org/)，[Expresso](http://visionmedia.github.io/expresso/" \t "_blank)，[Unit.js](http://unitjs.com/" \t "_blank)，[Karma](http://karma-runner.github.io/" \t "_blank)……

构建工具：[Grunt](http://gruntjs.com/" \t "_blank)，[Gulp](http://gulpjs.com/" \t "_blank)，[Webpack](http://webpack.github.io/" \t "_blank)……

* 1. Koa

文档：<https://koa.bootcss.com/>

前后端分离模式下 web框架 主要处理后端代码

安装

命令行输入 npm install -g koa-generator

<https://www.cnblogs.com/ts119/p/12061023.html>

使用

<https://www.jianshu.com/p/9838ee0501e4>

引用

const Koa = require('koa');

依赖

"koa": "2.0.0"

引用依赖写好以后在该目录下运行npm install

// 导入koa，和koa 1.x不同，在koa2中，我们导入的是一个class，因此用大写的Koa表示:

const Koa = require('koa');

// 创建一个Koa对象表示web app本身:

const app = new Koa();

// 对于任何请求，app将调用该异步函数处理请求：

app.use(async (ctx, next) => {

    await next();

    ctx.response.type = 'text/html';

    ctx.response.body = '<h1>Hello, koa2!</h1>';

});

// 在端口3000监听:

app.listen(3000);

console.log('app started at port 3000...');

加入package.json

{

"name": "dir\_name",

"version": "1.0.0",

"description": "xxxxxxxxxx",

"main": "main\_name.js",

"scripts": {

"start": "node main\_name.js"

},

"keywords": [

"koa",

"async"

],

"author": "xxx",

"license": "Apache-2.0",

"repository": {

"type": "git",

"url": "https://github.com/xxxxxx.git"

},

"dependencies": {

"koa": "2.0.0"

}

}

在当前目录执行：npm install，安装依赖

命令行运行js文件：node 文件名

koa-router

处理URL映射

<https://www.jianshu.com/p/f169c342b4d5>

文档：<https://www.npmjs.com/package/koa-router>

安装npm i @koa/router

引用

const router = require('koa-router')();

依赖

"koa-router": "10.0.0"

koa-bodyparser

解析原始request请求，然后，把解析后的参数，绑定到ctx.request.body中

文档：<https://www.npmjs.com/package/koa-bodyparser>

安装npm install --save koa-bodyparser

引用

const bodyParser = require('koa-bodyparser');

依赖

"koa-bodyparser": "4.3.0"