## Javascript语言

由浏览器编译运行

1. javascript存在形式，

javascirpt写在html的script标签，页面效果的script标签写到body内部最下面

2，js是弱类型语言

ECMAscript是javascript的标准，是一套规范

简单实现：

<body>

<script>

document.write("hello world");

</script>

</body>

单独的文件

Common.js:

console.log("apports");

html引用：

<script src="common.js"></script>

Alert浏览器对话框

alert(string)

### js变量

var a=123表示创建一个局部变量

a=123表示创建全局变量

#### var，let，const

var定义的变量，可以提前

**注释：//和/\*\*/**

### Js数据类型

数字，字符串，布尔值

数组，对象

数字类型包含浮点，NaN非数字

浮点和整数转换

parseInt(float)

parseFloat(int)

isNaN(num)判断是否为空

isFinite(num)无穷大

num.toString() 数字变成字符串

var i = 10; # 只能定义变量

var s = i.toString();

num.fixed(num) 精确几位后变成字符串

#### 消息框

<script type="text/javascript">

alert("apports");

var ret = confirm("apports"); //有判断

console.log(ret);

ret = prompt("apports"); //有输入框，返回值是输入框的内容

console.log(ret);

</script>

#### 字符串：

方法：

a=”alex”

Length获取长度

chatAt(num)获取字符串的字符 == a[num]

a.Bold() "<b>alex</b>"

a.big() "<big>alex</big>"

a.anchor(string url) "<a name=url>alex</a>"

a.concat(b) 字符串相加 +

a.trim()去空格 py---strip()

a.endswitch(string) 判断结尾字符串，startswitch(string)判断开头字符串

a.sub/a.sup上标和下标

a.italics()

"<i>alex</i>"

a.split(seq, n) seq按什么分割，n分割后拿多少个

子串：

subStr (str\_start,langth) 切片

subString(str\_start,str\_end)

Indexof(string substr,int start)查找

Includes(string) 是否包含

tolowercase/touppercase大小写切换

Search/match/replace正则表达式

Match()

["alex", index: 1, input: " alexbutton "]

split也可以匹配正则

"a>b<c>d>e".split(/[a-z]/)

(6) ["", ">", "<", ">", ">", ""]

Str.padStart(10, “”) 如果不满10就在前面填充10个字符

Str.padEnd(10, “”)

#### url编码和解码

url编码%xx%xx

encodeURL(string)->string

url解码

decodeURL(string)->string

decodeURIComponent("%E4%BD%A0%E5%A5%BD%EF%BC%8C%E4%B8%96%E7%95%8C%EF%BC%81")

"你好，世界！"

encodeURIComponent("你好，世界！")

"%E4%BD%A0%E5%A5%BD%EF%BC%8C%E4%B8%96%E7%95%8C%EF%BC%81"

encodeURI方法不会对下列字符编码  ASCII字母、数字、~!@#$&\*()=:/,;?+'

encodeURIComponent方法会对下列字符编码 ASCII字母、数字、~!\*()'

应用场景：当把url当作get参数传到后台时，必须用这个

encodeURIComponent("/etc/init.d/p")

"%2Fetc%2Finit.d%2Fp"

include，startswith，endswith

'hello'.repeat(2)

模板字符串

箭头函数

var f = v => v;

f("ddd")

#### 布尔

==

!=

=== 值和类型相同

!===

||

&&

#### 数组

a=[1,2,...]

a=[]

尾部添加删除：

Push()/pop()

Unshift()/shift()

concat([Array,...])->Array

splite(index,deletes,adds) 数组的某个位置，要删除多少，然后添加什么元素

删除某个元素

Arr.splite(2, 1)

删除索引从2开始的2个元素

Arr(splite(2,2))

删除索引从2开始的的元素

Arr.splite(2)

从arr中删除从索引为2开始的后面3个元素，最后在索引为2的位置插入”ass”

Arr.splite(2,3, “ass”)

obj.join(string)

slice(start,end,step)

切片

filter和map参考python的

数组追加

Arr.foreach((item)=>{})

Arr.map((item)=>{}) 对arr的每一项进行修改

Arr.Filter((item)=>{}) 过滤的每一项如果False则过滤

Arr.some((item, i)=>{return true}) //循环获取值和索引，直到返回，默认循环完后返回false

Array.prototype.push.apply(a1, a2)

a1

#### 对象

for(var k in dict) 遍历的是键

a={"k1":"v1","k2":[1,2,3,4]}

Object {k1: "v1", k2: Array[4]}

a["k1"]

"v1"

a["k2"]

[1, 2, 3, 4]

a["k2"][2]

3  
console.log(a.k2);

两种提取方式，一种像取数组一样的方式，因为字典是对象，所以也能以对象方式取

javascript中一切可以点的都可以用下标的方式表示：如  
location是一个类或者对象，下面有href

location.href就得到当前地址，也可以location["href"]

所以javascript默认支持反射

#### json序列号

对象序列化成字符串

JSON.stringify(obj) -> string

反序列化成对象

JSON.parse(string)->obj

escape()

unescape()

console.log(message) 控制台输出

typeof(obj)->string 输出类型

eval(string) 字符串转换成javascript语句，执行javascript多行代码

#### 获取时间

Var d=new Date();

Fri Oct 14 2016 23:55:04 GMT+0800 (中国标准时间)

~new Date是反转时间

d.getxxx()获取具体元素

d.setxxx()设置时间

### javascript语句

#### 条件语句：

If(条件){

}else if(条件){

}else{

}

Switch(obj){

Case value:

...;

Case value:

...;

Default

...;

}

#### 循环

for(var ;;){

}

for(var item in obj){

/\*item拿到的是索引\*/

}

for(var item of Iterable){

/\*拿到的是值，Iterable只能是数组或类数组\*/

}

while(表达式){

}

do{

}while(表达式);

Break; continue;

#### 异常处理：

try{

} catch (e) {

} finally {

}

主动抛出异常throw

throw new Error("fdsfdsf");

### javascript函数

普通函数，匿名函数，自执行函数(闭包)

js参数---默认参数

function foo(a,b) {  
 var a=10; //设置参数a的默认值  
 var b=20;  
 return a+b;  
}  
console.log(foo());

js参数存放位置：argument，所以可以传任何值，argument.length是函数传入的参数数量

js任何一个函数外面都 套着一个对象，传入任何数量的值会先存放在函数外面的对象ActiveObject，真正属于函数的参数是函数括号中指定的，函数的 可以调用函数外对象的变量。

变量没有声明和变量声明称undefined有区别，函数在

函数调用：  
func(…)

func.call(window,…)

#### Js随机数

Math.random()返回的是0-1的随机数

Math.round()四舍五入

#### arguments

arguments给函数传递的变量存放在这里，无论形参指定多少，所有实参都在这里，可以实现动态传参，如：

(function sum(){for(var i in arguments){console.log(i)}})(1,2,3,4)

1

2

3

4

undefined

arguments.callee返回当前的函数对象

剩余参数

**function** *OrderDict3*(num, ...args){  
 console.log(args);  
 console.log(arguments)  
}

…args表示除了一般的形参，剩余的形参存放的位置

默认参数

**function** *OrderDict3*(num = 20){  
  
 console.log(num)  
 console.log(arguments[0])  
}  
*OrderDict3*()

默认参数不能给arguments，arguments只存储传入的参数

let x1 = (a, b) => {console.log(a, b)}

x1(1, 2)

自执行函数：

自己定义的函数只能自己执行

b = (function f1() { return 10; })()

console.log(b);

var g=((x)=>{console.log(x); })(100);

g(10)

### Javascript作用域

Javascript无块级作用域

Javascript采用函数作用域

Javascript作用域链 (由内向外找)

Javascript作用域在执行之前已经创建

声明提前

在预编译时函数内部变量就已经声明，都是undefined，调用时再赋值

#### this

function.call

function.apply

this->window

this->obj

#### 闭包

闭包一个函数和被这个函数创建的函数的作用域的组合

利用闭包可以隐藏局部变量

(function(){var a=10})()

### 面向对象

javascript创建对象就是给this赋值，new就是给this赋值，直接调用函数就是执行函数，new的时候函数内部创建了一个新的this，一边执行函数一边给this赋值。返回给变量的是this。

给this赋值方法，是this的方法。每次创建对象都会创建一次。

给Person.prototype对象创建方法属性，通过this对象都可以调用Person.prototype属性，也就是Person.prototype只能给通过Person创建的对象调用。

Person的对象\_\_proto\_\_指向Person.prototype

Object.prototype-->Person.prototype-->

function foo(name,age) {

this.name = name;

this.age = age;

this.print=function() {

console.log(this.name + this.age);

}

}

s1 = new foo("alex", 12);

s1.print();

foo充当构造函数

this指向对象

创建对象用new

new一下之后会先执行foo一遍。

对象内部直接用this定义的方法每次创建对象都会定义一次，浪费空间，如果有一个可以共享的方法，每次调用就都去一个地方拿方法就可以了。

操作对象的方法，都是在Objects类。属于静态方法

大致有：

Objects.keys(obj) 拿到这个的所有key

Objects.values(obj) 拿到所有的value

一个function就是对象，可以function.xx

普通函数内的this和函数外的this都是window，事件函数的this是选中的document，对象的this是对象本身

#### 原形

定义一个类的两种方式：

// 原型

var Person = function(name){

this.name = name;

}

Person.prototype.sayHello = function(){

console.log("Hello I,m " + this.getName());

}

Person.prototype.getName = function(){

return this.name;

}

// 类

class Person{

constructor(name){

this.name = name;

}

sayHello() {

console.log("Hello I,m " + this.getName())

}

getName(){

return this.name

}

}

调用

var p01 = new Person("aaa")

p01.sayHello()

原型方法和变量方法

javascript有显式原型和隐式原型，每个函数创建后，都有一个名为prototype的属性，该属性指向函数的原型对象

**function Person(){this}**

**> Person.prototype**

**var p01 = new Person()**

javascript中任何对象都有prototype属性，大多数浏览器都是通过\_\_proto\_\_来访问，

Person是p01的构造方法，prototype是p01的原型对象，this就是p01

constructor

p01.constructor === Person True

p01.\_\_proto\_\_ === prototype True

Object.create(object) 用现有的对象创建新对象

var p02 = Object.create(p01)

object2基于object1创建

p02的原型对象是p01

如果现在要用p02对象调用show函数

p02---p01---Person.prototype 就是一条原型链

p02.show() 首先会去p02中找，没有去原型链的上一级p01中找，然后去Person.prototype找，还找不到就去Object中找，直到找到就调用。

为什么prototype找不到去Object找？

因为Person.prototype是由Object创建的，也就是Object.prototype 是Person.prototype的上一级原型

每一个对象的constructor指向创建该对象的构造函数

#### 类

**生成器**

在方法名前加\*就是一个生成器

**静态方法**

类内部static修饰的方法

**super**

调用父类的方法

super() 调用父类构造函数

super.xxx

类继承

class Teacher extends Person{

sayHello() {

console.log("Hello I,m " + this.getName())

}

}

var p01 = new Teacher("alex")

#### getter和setter

class Person{

constructor(name){

this.name = name;

this.\_age = 18;

}

get age(){

return this.\_age;

}

set age(value){

this.\_age = value;

}

#### 代理器

用来拦截一部分操作：

var obj = new Proxy({

count: 1,

getCount(){

return this.count

}

},{

get: function (target, key, receiver) {

console.log(`getting ${key}!`);

return Reflect.get(target, key, receiver);

},

set: function (target, key, value, receiver) {

console.log(`setting ${key}!`)

return Reflect.set(target, key, value,receiver);

}

})

get会捕获任何读操作，传入的参数：目标对象，属性名，proxy对象本身

set捕获任何写操作，传入的参数：目标对象，属性名，属性值，proxy对象本身

apply捕获函数调用时，传入参数：目标对象，目标对象的上下文对象，参数列表

has捕获hasProperty操作，判断对象是否具有某些属性，hasProperty也可以用in关键字来代替，传入参数：目标对象，属性名

construct捕获new操作，传入参数：目标对象，构造函数参数列表

deleteProperty捕获delete操作，传入对象：目标对象，属性名

defineProperty捕获定义新属性时的操作，传入对象：目标对象，属性名，描述

getOwnPropertyDescriptor捕获Object.getOwnPropertyDescriptor()时的操作，返回一个属性的描述对象。传入参数：target，key

getPropertyOf捕获对象原型的读取操作，

拦截对象：

Object.prototype.\_\_proto\_\_

Object.prototype.isPrototypeOf

Object.getPrototypeOf

Reflect.getPrototypeOf

instanceof

传入参数：目标对象

setPropertyOf 捕获对象原型的写入操作，主要拦截setPrototypeOf

ownKeys捕获对象自身属性的读取操作，具体拦截：

Object.getOwnPropertyName

Object.getOwnPropertySymbols

Object.keys

传入参数：目标对象

isExtensible捕获Object.isExtensible操作，传入target

preventExtensions捕获Object.preventExtensions，传入target

#### Reflect反射

Reflect.has(Object, 'assign')

Reflect.apply(target, thisArg, args)

Reflect.construct(target, args)

Reflect.get(target, name, receiver)

Reflect.set(target, name, value, receiver)

Reflect.defineProperty(target, name, desc)

Reflect.deleteProperty(target, name)

Reflect.has(target, name)

Reflect.ownKeys(target)

Reflect.isExtensible(target)

Reflect.preventExtensions(target)

Reflect.getOwnPropertyDescriptor(target, name)

Reflect.getPrototypeOf(target)

Reflect.setPrototypeOf(target, prototype)

#### call和apply

var foo = {

value: 65535

}

function bar(name, age){

console.log(name)

console.log(age)

console.log(this.value)

}

bar("darlin", 18)

bar.apply(foo, ["darlin", 18])

bar.call(foo, "darlin", 18)

### 模块

require导入：

var assert\_obj = require("assert")

console.log(assert\_obj)

{ [Function: ok]

**fail: [Function: fail],**

**AssertionError: [Function: AssertionError],**

**ok: [Circular],**

**equal: [Function: equal],**

**notEqual: [Function: notEqual],**

**deepEqual: [Function: deepEqual],**

**deepStrictEqual: [Function: deepStrictEqual],**

**notDeepEqual: [Function: notDeepEqual],**

**notDeepStrictEqual: [Function: notDeepStrictEqual],**

**strictEqual: [Function: strictEqual],**

**notStrictEqual: [Function: notStrictEqual],**

**throws: [Function: throws],**

**doesNotThrow: [Function: doesNotThrow],**

**ifError: [Function: ifError] }**

require导入结果是一个对象

自定义模块导入：

constant.js自定义模块

module.exports = {

test: function (name) {

console.log(name)

}

}

test1.js导入

var mod\_obj = require("./constant")

mod\_obj.test("张三")

#### module.exports和exports

module.exports赋值后给require变量用的，exports点出来的属性是给require变量的属性

### Js事件

**$**(**".js-status-filter"**)[0].**onclick**=*get\_filter*(**this**);

直接传普通函数会在页面加载的时候执行，应该传匿名函数，在匿名 函数中再调用 普通函数。匿名函数的this就是这个元素。

Onmousedown和onmouseup

Onmouseout和onmouseover

Onmousewheel 鼠标滚轮

滚轮事件：wheelDelta/wheelDeltaX/wheelDeltaY +上滚/-下滚

Onmouseout是当鼠标移出标签时执行，onmouseover是当鼠标移入标签时执行

Onload

Onsubmit提交表单时执行

Onerror当发生错误的时候执行

Onclick

onFocus获取焦点,onblur失去焦点

onscroll滚动事件，只有当滚轴移动时触发

onchange 内容被更改

oninput

事件=javascript语句

事件返回结果：false 该元素不触发其他事件

<div onclick="this.innerHTML='hello!'">fdsfdsgds</div>

用this参数可传递给事件当前标签的节点

<div id="menu" class="menu" onclick="show(this)"> </div>

窗口事件：

Window.xxx

window.onload=function(){

document.write("ooo");

}

Html中第一个空格必须用那个&nbsp

Onkeypress事件用来接收键盘字母或数字键的代码

Onkeydown事件用来接收任何键盘输入的代码，keyCode用来识别键盘输入的代码

键盘码：左：37，上：38，右：39，下：40

window.onkeydown = function (e) {

e = e.keyCode;

document.write(e);

switch (e) {

case 37: //left

document.write("left");

break;

case 38: //top

document.write("top");

break;

case 39: //right

document.write("right");

break;

case 40: //bottom

document.write("bottom");

break;

}

}

Event参数由浏览器提供

#### 键盘代码

|  |  |
| --- | --- |
| **键盘按键** | **对应的数码** |
| backspace | 8 |
| tab | 9 |
| enter | 13 |
| shift | 16 |
| ctrl | 17 |
| alt | 18 |
| pause/break | 19 |
| caps lock | 20 |
| escape | 27 |
| page up | 33 |
| Space | 32 |
| page down | 34 |
| end | 35 |
| home | 36 |
| arrow left | 37 |
| arrow up | 38 |
| arrow right | 39 |
| arrow down | 40 |
| insert | 45 |
| delete | 46 |
| 0 | 48 |
| 1 | 49 |
| 2 | 50 |
| 3 | 51 |
| 4 | 52 |
| 5 | 53 |
| 6 | 54 |
| 7 | 55 |
| 8 | 56 |
| 9 | 57 |
| a | 65 |
| b | 66 |
| c | 67 |
| d | 68 |
| e | 69 |
| f | 70 |
| g | 71 |
| h | 72 |
| i | 73 |
| j | 74 |
| k | 75 |
| l | 76 |
| m | 77 |
| n | 78 |
| o | 79 |
| p | 80 |
| q | 81 |
| r | 82 |
| s | 83 |
| t | 84 |
| u | 85 |
| v | 86 |
| w | 87 |
| x | 88 |
| y | 89 |
| z | 90 |
| left window key | 91 |
| right window key | 92 |
| select key | 93 |
| numpad 0 | 96 |
| numpad 1 | 97 |
| numpad 2 | 98 |
| numpad 3 | 99 |
| numpad 4 | 100 |
| numpad 5 | 101 |
| numpad 6 | 102 |
| numpad 7 | 103 |
| numpad 8 | 104 |
| numpad 9 | 105 |
| multiply | 106 |
| add | 107 |
| subtract | 109 |
| decimal point | 110 |
| divide | 111 |
| f1 | 112 |
| f2 | 113 |
| f3 | 114 |
| f4 | 115 |
| f5 | 116 |
| f6 | 117 |
| f7 | 118 |
| f8 | 119 |
| f9 | 120 |
| f10 | 121 |
| f1 |  |

#### 键盘事件

onkeydown/onkeyup 所有的键按下/弹起

onkeypress 非控制键

键盘事件返回属性：

key:"g"

keyCode:71

#### Js默认事件

默认事件：a默认事件是跳转，form默认事件是跳转。a自定义先执行

自定义事件函数内return false阻止默认事件执行，true允许默认事件执行

<form>  
 <input type="text" value="aaa" />  
 <input type="submit" onclick="**return** func1(**this**);"/>  
</form>  
<script>  
 **function** func1(ths) {  
 **return true**;  
 }  
</script>

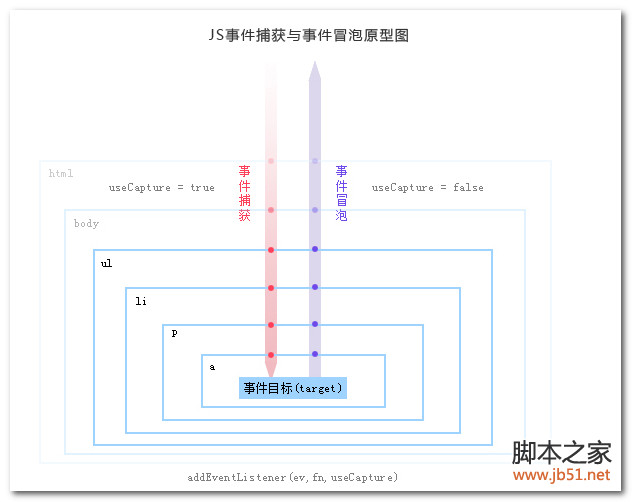
#### addEventListen(event,function,useCapture)添加事件句柄

removeEventListen为删除事件句柄

参数event：指定事件名，用字符串，click事件，不能使用on前缀

Function：事件触发后执行的函数

UseCapture：是否在捕获或者冒泡阶段执行，布尔值，true从外到内—捕获，false从内到外—冒泡



* 使用此函数添加鼠标滚轮事件：

document.addEventListener('mousewheel',function (ev) {  
 console.log(ev);  
},false);

此事件只能在chrome里执行

DOMMouseScroll为火狐的滚轮事件

#### 移动端事件：

Touchstart->touchmove->...->touchmove->touchend

addEventListener来添加事件

#### addEventListener

#### 窗口事件

onbeforeunload 是正要去服务器读 取新的页面时调用，此时还没开始读取；而onunload则已经从服务器上读到了需要加载的新的页面，在即将替换掉当前页面时调用。

### Js正则表达式

正規表現は２種類の方法で作成できます

/<正则>/ 不加引号

Search，match，replace

Replace是一个内置的字符串转换函数，如果像取到一些样式表有px结尾的，可以用replace替换成整数

snake[i].style.left.replace("px", "")

'fgd123'.replace(/[0-9]+/,"k")->replace\_res

#### 替换模式

/.../

/.../g 全局匹配

/.../i 不区分大小写

/.../m 多行匹配

'fgd123'.replace(/[0-9]/g,"k")

"fgdkkk"

$&当前匹配的内容

test测试是否匹配,返回bool

var a=/^\d$/

a.test("12fsdf45")

exec所有匹配到，并组成数组，如果没有返回null

var a=/([0-9]+)([a-z]\*)([0-9]+)/

undefined

a.exec("12fsdf45")

["12fsdf45", "12", "fsdf", "45"]

//

后项引用用$

**location.search**

"?meaning=1&status=2" //去掉?

**location.search.replace(/\?(.\*)/,"$1")**

"meaning=1&status=2"

//正则表达式在js中的类型是对象，是RegExp的对象。正则表达式对象的方法：  
regex.test(str) 是否能匹配到

regex.

### 严格模式

一般使用'use strict'就能让代码按照严格模式解释，类和模块内部默认是严格模式。严格模式：

1，不存在变量提升

### Dom----html文档对象模型

通过javascript可以重构html文档，DOM可以被javascript读取，改变html,xhtml,xml文档，在DOM中整个html文档就是一个节点，每个成分是子节点，包括标签，属性，内容，注释

任何节点之间都存在关系，查找节点的方法：通过标签名，id名，类名，name名等

#### Dom节点访问

1. 得到节点的id
2. 查看id节点的标签吗
3. 查看id节点的属性组的其中一个属性的属性名和属性值
4. 查看id节点的内容

#### 获取节点列表

geteElementsById(id)/ getElementsByTagName(标签)/getElementsByClassName(类)

tag1 = document.getElementById("bar"); //得到节点元素

[object HTMLDivElement] //表示得到div标签的节点

间接查找节点：

parentNode/parentElement

childNodes/children 所有子节点/所有子标签

firstchild/firstElementChild 第一个子节点/第一个子标签

lastchild/lastElementChild 最后一个子节点/最后一个子标签

同级关系：

nextSibling/nextElementSibling 上一个节点或标签

previousSibling/previousElementSibling 下一个节点或标签

获取节点的标签，内容，属性等

document.write(tag1.tagName+"<br />"); //得到节点元素的标签

document.write(tag1.innerHTML + "<br />"); //得到节点元素的内容

innerHTML和innerText：

innerHTML获取内容，包括html标签，解析成html

innerText获取所有内容，只是内容

value获取值

<div id="kr">b<i>aa</i>

</div>

<script>

var bId = document.getElementById("kr");

document.write(bId.innerText+"<br />");

document.write(bId.innerHTML);

</script>

#### 类相关操作：

className获取所有class名，返回字符串

classlist获取所有class名，返回数组----remove删除class名，add添加class名

i1HD.classList.add("c4");  
i1HD.classList.remove("c1");  
document.write(i1HD.classList);

2表示类型为属性节点

对于属性来说，属性有属性节点和属性名和属性值

属性节点名==属性名 nodename==name

tag1.getAttributeNode("class") + "<br />" //获取该元素的class属性的节点

tag1.getAttributeNode("class").nodeName + "<br />" //获取属性名

tag1.getAttributeNode("class").value + "<br />" //获取属性值

tag1.attributes 获取属性组

tag1.attributes.length 获取属性组的属性数量

tag1.attributes.item(0) 获取属性组的第一个属性节点

获取所有属性的属性名和属性值

for (var i = 0; i < tag1.attributes.length; i++) {

document.write(tag1.attributes.item(i).name + ":" +tag1.attributes.item(i).value + "<br />");

}

Nodename节点名称

Nodevalue节点值

Nodetype节点类型

document添加节点：

createElement(标签)->elementNode

createAttribe(属性名)->attribeNode

parentNode.appendChild(childNode)

parentNode.removeChild(childNode)

node.parentNode该标签的父级标签

insertBefore()

insertAfter()

先要有节点，才能访问，script访问节点的代码一般写在html元素布局完后

设置属性

Setattribe(属性名，属性值)

tag1.setAttribute("class", "bar"); //修改节点

tag1.setAttribute("style", "background-color:red"); //添加节点

设置具体的css样式

tag1.style.color = "blue";

类操作

ClassName

classList remove和add添加和删除类

#### 创建新元素

先组合一个标签，然后添加到某个节点下面

var para=document.createElement("p");

var node = document.createTextNode("this is a new parameter");

para.appendChild(node);

tag1.appendChild(para);

insertBefore(插入的节点,已存在的节点)表示插入指定节点的前面

#### 删除节点

para.parentNode.removeChild(para);

javascript中任何修改html的行为都会在浏览器的html代码中显示

dom浏览器操作：

window对象---窗口相关

open(url,target,feature)

feature是窗口的特性

function f1() {

myWind = window.open("","", "height=200,width=400");

myWind.document.write("hello world");

}

Height/width

Left/top

可批量打开

按毫秒周期循环调用函数

setInterval(string handle,int time)->number

clearinterval(number)->number

指定毫秒后调用函数

setTimeout(string handle,int time)->number

clearTimeout(number)->number

### js异步

传统异步解决方案：回掉函数+事件

setTimeout(function () {

console.log("这里是最里面");

},1000);

console.log("这里是外面");

这个执行结果是：

这里是外面

这里是最里面

看上去是setTimeout又分配了一个线程，但是浏览器只分配给js一个线程，js利用这一个线程实现的异步。

setTimeout(function () {

setTimeout(function () {

console.log("这里是最里面"); # 2秒后执行

},1000);

console.log("这里是里面"); # 1秒后执行

},1000);

console.log("这里是外面"); #立刻执行

**var *num***=*setInterval*(**function** () {  
  
 ***console***.log(**"这里是里面"**,***count***++);  
 **if**(***count*** == 2){  
 *clearInterval*(***num***); #num的作用域在外面  
 }  
}, 1000);  
  
***console***.log(**"这里是外面"**);

----执行结果

这里是外面

这里是里面 0

这里是里面 1

**var *num***=*setInterval*(**function** () {  
  
 ***console***.log(**"这里是里面"**,***count***++);  
  
}, 1000);  
**if**(***count*** == 2){  
 *clearInterval*(***num***);  
 }

-----像这样，它是不会停下来的，因为那个时候count才只有0，这个clearInterval就永远不会执行

浏览器是事件驱动的，浏览器中很多行为是异步的，如鼠标事件，键盘事件，定时器事件，这些会放入专门的事件队列中，当一个异步事件发生后，进入事件队列，进行事件的循环，当一边在执行onclick事件，又出来了window.resize事件，就会排队。如setTimeout，会把setTimeout放到事件队列中，等待当前线程执行完毕再执行setTimeout

ajax请求也是异步的。

取消ajax异步：

$.ajax({  
 **async**:**false**,  
});

### js缓存

navigator返回浏览器的一些值：

|  |  |
| --- | --- |
| appName | 返回浏览器名 |
| Appversion | 返回那啥 |
|  |  |
|  |  |
|  |  |
|  |  |

ChildNodes子节点列表

item.childNodes[1].style.display = "block";

a ：javascript：void(0)

滚动高度：

获取窗口的滚动条—body的高度

document.body.scrollTop

document.body.scrollLeft

文档body的滚动条高宽

表单提交：

<form id="form">  
 <input type="text" />  
  
</form>  
<input type="submit" onclick="subForm()" />

**function** subForm() {  
 **var** tag=document.getElementById("form");  
 tag.submit();  
}

submit就能指定表单的id提交

### window

打开窗口

    window.open ("page.html", "newwindow", "height=100, width=400, top=0, left=0, toolbar=no, menubar=no, scrollbars=no, resizable=no,location=no, status=no")

参数：

参数 | 取值范围 | 说明

alwaysLowered | yes/no | 指定窗口隐藏在所有窗口之后   
alwaysRaised | yes/no | 指定窗口悬浮在所有窗口之上   
depended | yes/no | 是否和父窗口同时关闭   
directories | yes/no | Nav2和3的目录栏是否可见   
height | pixel value | 窗口高度   
hotkeys | yes/no | 在没菜单栏的窗口中设安全退出热键   
innerHeight | pixel value | 窗口中文档的像素高度   
innerWidth | pixel value | 窗口中文档的像素宽度   
location | yes/no | 位置栏是否可见   
menubar | yes/no | 菜单栏是否可见   
outerHeight | pixel value | 设定窗口(包括装饰边框)的像素高度   
outerWidth | pixel value | 设定窗口(包括装饰边框)的像素宽度   
resizable | yes/no | 窗口大小是否可调整   
screenX | pixel value | 窗口距屏幕左边界的像素长度   
screenY | pixel value | 窗口距屏幕上边界的像素长度   
scrollbars | yes/no | 窗口是否可有滚动栏   
titlebar | yes/no | 窗口题目栏是否可见   
toolbar | yes/no | 窗口工具栏是否可见   
Width | pixel value | 窗口的像素宽度   
z-look | yes/no | 窗口被激活后是否浮在其它窗口之上

关闭窗口

**window**.close();

### navigator

获取当前浏览器的类型

navigator.userAgent

--------

ie11: "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko" 正则匹配：Trident.\*rv

chrome/firefox: "mozilla/5.0 (windows nt 6.1; win64; x64) applewebkit/537.36 (khtml, like gecko) chrome/58.0.3029.110 safari/537.36"

正则匹配: applewebkit

ie用document.documentMode查看ie的版本

navigator.cookieEnabled判断是否开启cookie

chrome禁用cookie: 设置->隐私设置->内容设置->阻止网站任何数据

navigator.onLine 是否脱机

### location

location.href查看当前url，location.href=xxx转跳

location.reload() 刷新页面

location.host 域名加端口

location.origin 完整url

location.pathname 路径名

location.replace(xxx) url尾部字符串追加

location.search ?后面的内容

js生成url字典：

**var** url\_dict={  
 **"path\_info"**:**null**,  
 **"params\_dict"**:{}  
};

### history

history.back() 后退一步

history.forward() 前进一步

history.go(x) 去往第几步，-为后退，0为刷新页面

### document

document.cookie 返回当前页面cookie

document.title 返回当前文档的title

### dom宽高和位置

对象的宽高和位置

domobj.style.height=xxx+”px”;

domobj.style.width=xxx+”px”;

domobj.style.left=xxx+”px”;

domobj.style.top=xxx+”px”

浏览器内宽高（window）:

innerWidth/innerHeight

浏览器外宽高（window）

outerWidth/outerHeight

浏览器窗口的位置（window）

screenTop/screenLeft

浏览器滚动条偏移位置

pageXOffset/pageYOffset==scrollX/ScrollY

screen.width/screen.height

对象内部宽高:

obj.clientWidth/obj.clientHeight

边框的位置，对于relative父节点的位置

obj.clientLeft/obj.clientTop

整个对象宽高

obj.offsetWidth/obj.offsetHeight

整个对象相对文档位置，对于relative父节点的位置

obj.offsetTop/obj.offsetLeft

对象内部滚动条

obj.scrollTop/scrollLeft/scrollWidth/scrollHeight

event事件：

clientX/clientY/x/y 鼠标到浏览器窗口的位置

offsetX/offsetY相对于对象的位置

screenX/screenY 鼠标到屏幕的位置

pageX/pageY 鼠标到文档的位置

key:

shiftKey/ ctrlKey/ altKey 是否按住shift键

type事件类型

srcElement

altKey:false

bubbles:true

button:0

buttons:0

cancelBubble:false

cancelable:true

clientX:219

clientY:209

composed:true

ctrlKey:false

currentTarget:null

defaultPrevented:false

detail:1

eventPhase:0

fromElement:null

isTrusted:true

layerX:199

layerY:199

metaKey:false

movementX:0

movementY:0

offsetX:199

offsetY:199

pageX:219

pageY:209

path:Array(6)

relatedTarget:null

returnValue:true

screenX:219

screenY:275

shiftKey:false

sourceCapabilities:InputDeviceCapabilities

srcElement:div.js-box

target:div.js-box

timeStamp:46041.865

toElement:div.js-box

type:"click"

view:Window

which:1

x:219

y:209

\_\_proto\_\_:MouseEvent

### js上传文件

如果用<input type="file" />上传文件

document.files[0]得到一个文件对象，而value得到的是本地文件路径

File {name: "0516\_2.png", lastModified: 1494910340194, lastModifiedDate: Tue May 16 2017 12:52:20 GMT+0800 (中国标准时间), webkitRelativePath: "", size: 837165…}

包含一个头部和一个主体

头部信息有name文件名, lastModified最后上传的时间戳，size文件大小，type文件类型

js和form自动提交的区别，form是**自动整合**form内所有name和value，遇到multype时收集文件内容，js需要手动进行收集。

js本地缓存：

localStorage是HTML5的新特性，cookie只能4K，localStorage有5M

localStorage.setItem('testAddItem', 'New item in localStorage');

localStorage.getItem('testAddItem');

localStorage.removeIterm('testAddItem')

localStorage.clear()

### js类库：

#### RegExp 正则表达式

#### Set集合

s1 = new Set()

**res:** Set(0) {}

s2 = new Set([2,3,4])

res:Set(3) {2, 3, 4}

s1.add(3)

s1.delete(3)

s1.has(4) //是否有项

res:true

s1.keys()

SetIterator {12, 15, 1}

s1.values()

SetIterator {12, 15, 1}

#### Map字典

d1 = new Map()

d1.set("name","alex")

d1.has("name")

d1.get("name")

d1.delete("name")

d1.size

#### Array字符串

#### Date时间

## nodejs

### assert模块

断言：如果不符合true，就会报出异常：EssertionError

assert.deepEqual(obj1,obj2,message) 匹配两个可迭代对象(对象或数组)

assert.deepStrictEqual(obj1,obj2, message) 严格匹配，比deepEqual多了类型判断

assert.notDeepEqual(obj1, obj2, message)

assert.equal(obj1, obj2, message)

### buffer模块

const buffer = Buffer.from('helloworld', "ascii");

console.log(buffer)

比较两个缓冲区

Buffer.compare(buf1, buf2)

缓冲区拼接：

Buffer.concat([buf1, buf2], totalLength) totalLength可截取合并后的一部分

buf1.buffer 指向buffer底层的ArrayBuffer

ArrayBuffer { byteLength: 8192 }

buf1.fill(xx) 一个字节的内容重复填充到buf1，直到填充满

bu1.include("ixxx") 是否包含

buf1.slice(start, end) 切片

buf1.toJSON()

{ type: 'Buffer',

data: [ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ] }

buf1.toString(encoding)

编码类型：hex，ascii，utf8

### child\_process

exec是child\_process下的一个属性

exec(cmd, callback) cmd传入命令，backend能拿到结果

const {exec} = require("child\_process");

exec("ls -l", (err, stdout, stderr) => {

if(err){

console.error(err)

}

console.log(stdout)

})

const {exec} = require("child\_process")

const ls = exec("ls -l")

ls.stdout.on('data', (data) => {

console.log(`true====${data}`);

})

ls.stderr.on('data', function(data){

console.log(`error===${data}`);

})

ls.on('exit', function(code){

console.log(`=====${code}`)

})

child\_process.execFileSync

path:

path.dirname

path.basename

path.extname

path.join(...args) == path.join("/", "a", "b", "c") "/a/b/c"

path.format({

dir:"xxx",

base: "xxx"

})

path.parse

path.parse("/home/a/b/c.docx")

{ root: '/',

dir: '/home/a/b',

base: 'c.docx',

ext: '.docx',

name: 'c' }

指定操作系统

path.posix.xxx

path.win32.xxx

### 加密模块

> crypto.createHmac('sha256', "afdc").digest("hex")

'08f22afcbd64bafc10854aeeb5e390358e41995850517f5e07f6a4eeaa60fd49'

### fs文件操作

fs.stat(file, callback)

fs.readFile(file, encoding, callback) callback===(err, data) => {}

fs.writeFile(file, data, callback)

fs.appendFile(file, data, callback)

fs.copyFile(src,dec, callback)

fs.mkdir

所有文件操作都是事件，javascript一切事件都是异步，如果要同步，每个方法加sync，如：

fs.readFileSync

所有同步操作都没有callback，直接操作就行，返回值就是结果

文件流操作：

var fs = require("fs")

var rs = fs.createReadStream("/etc/passwd", "utf8")

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

console.log(chunk)

})

### http模块

agent和server

## jquery

是一个强大的javascript库

Jquery官方网站：<http://jquery.com/download/#Download_jQuery>

Jquery谷歌：

<head>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js">

</script>

</head>

微软：

<head>

<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.8.0.js">

</script>

</head>

### 查找：选择器和筛选器

$("#i1").addClass("c1");

详见css选择器

选择后筛选

#id id选择器

Element 标签选择器

.class 类选择器

\*

逗号和层级选择器

$(选择器) 返回的是jQuery对象

$("#i1").addClass("hide")

[<div id=​"i1" class=​"hide">​</div>​]

$("#i1").removeClass("hide")

[<div id=​"i1" class>​</div>​]

<div>DIV1</div>

<div>DIV2</div>

<span>SPAN</span>

<div id="myDiv">id="myDiv"</div>

<div id="notMe"><p>id="notMe"</p></div>

查找标签为div的元素：

$("div")

查找id为myDiv的元素

$(“#myDiv”)

$()转换为jQuery

$()[0]转换成dom

基本筛选器：

:first/:last

:even/:odd 奇偶

属性选择器

$("ul li[content='ThePointItem']")

[<li content=​"ThePointItem">​list item 3​</li>​]

[attribute]/[attribute=”value”]

[attribute!=”value”]

在$外面再筛选：

$("form input[type='button']").first();

Find()查找该元素下的指定选择器的元素

Next()/nextAll()同级的下一个标签/同级以下的所有标签

NextUntil(选择器)从下几个当中直到有选择器选到的为止

$("select").find("[selected='selected']").nextUntil(":even");

[<option>​2​</option>​]

Parent()该标签父级

Parents()该标签父级以及父级的所有同级标签

$("select").find("[selected='selected']").parent();

[<select>​…​</select>​]

$("select").find("[selected='selected']").parents();

[<select>​…​</select>​, <form>​…​</form>​, <body>​…​</body>​, <html lang=​"en">​<head>​…​</head>​<body>​…​</body>​</html>​]

Siblings()同级标签的除了该标签的所有标签

其它：

如果一堆radio中要找选中的标签

$(“input:checked”)

像类似:checked,:selected,:disabled,:enabled

#### 筛选器：

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数 |  |
| first() |  | 查找第一个 |
| last() |  |  |
| eq(num) |  | 指定索引值 |
| Filter() | String选择器  jqObject对象  一个dom元素  Function | 从筛选的结果中再次筛选匹配的 |
| hasClass()->boolean | String---class类名 |  |
|  |  |  |
| Children(expr) |  | 所有子标签 |
| Find(expr) | String选择器 | 查找符合条件的子标签 |
| Next(expr) |  | 同级的下一个标签 |
| NextAll(expr) |  | 同级的下面所有标签 |
| Prev(expr) |  | 同级的上一个标签 |
| PrevAll(expr) |  | 同级上面的所有标签 |
| Parent(expr) |  | 父级标签 |
| Parents(expr) |  | 父级标签以及父级标签的同级所有标签 |
| Siblings(expr) |  | 同级的其它所有标签 |

### 操作：css，属性，文本

#### 属性操作：

removeAttr(name) 删除指定属性名的属性

attr(name,value/dict)添加属性,dict可用字典添加多个属性

$("ul li:first").attr({style:"background-color:red",class:"bk1"});

Prop()专门用来做checked/selected的打勾

$("table tr input[type='checkbox']").prop("checked",true);

addClass(cname)添加类名

removeClass(name)删除类名

toggleClass(name) 如果存在，删除，如果不存在，添加

#### 文本操作：

Text(string)->string添加文本，可返回当前文本

Html(string)->string 添加html元素，可返回html元素

$("ul li:first").text($("ul li:first").text()+"dddd");

[<li>​list item 1dddd​</li>​]

Val(string)->string 添加值，可返回值

#### Css操作：

Css(name,value)

#### 位置/尺寸：

scrollTop()/scrollLeft()设置滚动条位置

height(string/int)返回高度整数，可设置高度

jquery的width/height不算浏览器滚动条

$("#d1").height(50);

$("#d1").height("50px");

Width()返回宽度

#### 元素的遍历

each() 参数可以是一个函数，函数里面this表示当前循环的每一个元素

$("table tr input[type='checkbox']").each(function () {  
 if($(this).prop("checked")){  
 $(this).prop("checked",false);  
 }else{  
 $(this).prop("checked",true);  
 }  
});

Length查看有多少元素，这个也可以用来遍历

#### 添加删除元素

append和appendTo

父元素.append(子元素)

子元素.appendTo(父元素)

//在指定内部尾部添加

$.fn.append(content/fn)

$.fn.appendTo(content/fn)

preppend(Element)/prependTo //在指定内部开头添加

after(Element)/insertafter() //在指定下面添加

before(Element)/insertbefore() //在指定上面添加

wrap(Element) //在指定外部套一层

remove() //删除指定的

empty() //清空元素内部

clone() //克隆一份

$(".menu .item").append($(".menu .item").clone());

Each(function(i){

//i指代元素的下标

})

### jquery循环

$.fn.each(backcall)

$("li").each(**function** (i,j) {  
 /\*  
 \* i循环的编号  
 \* j==this  
 \* return false 跳出循环break  
 \* return true 跳到下一层循环continue\*/  
 **return true**;  
 console.log(i,j,**this**);  
})

$.each(循环列表，循环函数)

$.each([1,2,3],**function** (k,v) {  
 console.log(k,v);  
})

$.fn.size() 选择器选中了多少 $.fn.length

$.fn.selector() 选择器

### 定义事件----优化

#### 1，使用jquery绑定事件

Click(function)

$(".menu .title").click(function () {  
 showMenu(this); //只要写一次所有标签都会调用  
});  
  
function showMenu(ths) {  
 $(ths).next().removeClass("hide");  
 console.log($(ths).parent().siblings().find(".body").addClass("hide"));  
}

#### Bind()绑定事件，事件名用字符串表示

$(".menu .title").bind("click",function () {  
 showMenu(this);  
});

on和bind功能一样

unbind/off

更改事件---- off("click")----on

$(".js-submit-blog").click()如果内部没有function，为触发点击事件，有function为等待触发

#### 2，当文档加载完毕时自动执行事件

当网页图片很多的时候，网页不能一下子显示图片，但是文档树已经形成。执行事件可以不用等到图片加载完，只要文档树一成立就可执行

$(**function** () {  
 //文档树加载完毕时执行  
 $(".menu .title").bind("click",**function** () {  
 showMenu(**this**);  
 });  
});

jQuery(callback)的callback是当文档加载完后执行的函数

$(document).ready 该

$(document).ready(  
 **function** () {  
 $("li").click(**function** () {  
 console.log(**this**);  
 })  
 }  
);

#### 3，delegate延迟绑定

当现有的所有内容都绑定了事件后，我要新建一个标签，让它也绑定和以上内容一样的事件。但是直接新建标签没法让它绑定事件

Delegete()都是当事件发生的时候绑定，加载的时候不绑定

$(**function** () {  
 $("ul").delegate("li","click",**function** () {  
 alert($(**this**).text());  
 });  
});

### Jquery动画

#### 隐藏和显示

$("input[type='text']").hide();

$("input[type='text']").show();

Hide(speed)

show(speed)

toggle(speed)-----一次隐藏，一次显示

speed华丽的隐藏和显示---可用slow/fast/毫秒

#### 淡入和淡出

fadeIn/fadeout/fadetoggle

fadeTo---透明度到多少

$("input[type='text']").fadeTo("slow",0.2);

表示逐渐淡出到20%的透明度

#### 滑动

slideUp/slideDown/slideToggle

$("#cute").slideToggle("slow");

#### 动画

**function** func4() {  
 //$("#cute").animate({width:'250px'},"slow");  
 //$("#cute").animate({width:'+=250px'},"slow");//使用相对值  
 //此动画会分两步完成  
 $("#cute").animate({width:'250px'},"slow");  
 $("#cute").animate({height:'250px'},"slow");  
 //$("#cute").animate({width:'250px'},"slow");  
}

$({f:12})定义一个对象，f=12，animate可以让这个值自增

animate语法二：

$({value: 15}).animate({value: 50},{  
 duration:1000,  
 easing:"swing",  
 step: function() {  
 $("#haha").text(Math.floor(this.value));  
 },  
 complete:function () {  
 $("#haha").text(stop);  
 }  
});

duration速度

easing

step:每一步执行一个函数

complete:执行完毕后执行一个函数

#### stop停止动画

$("#cute").stop();

#### Callback

以上动画的callback函数在动画执行完后执行

$("#cute").animate({height:'250px'},"slow",**null**,**function** () {  
 alert("动画加载完毕");  
});

### Ajax

#### 原生的ajax

XMLHttpResponse()用来给服务器发请求，是一个对象

几个方法

Open(method,url,async,user,password)，定义如何发送

Send(data)，发送data内容

setRequestHeader(head,value)，添加请求头部的一个键值对

属性：

Onreadystatechange 监听更改发送的状态

readyState 当前状态的状态码：发送接收完毕为4

responseText 服务器返回的字符串

发送实例：

var xhr=new XMLHttpRequest();  
xhr.onreadystatechange=function () {  
 if(xhr.readyState==4){  
 console.log(xhr.responseText)  
 }  
};  
xhr.open("GET","/ajax\_xhr/?p=123");

xhr.send();

post

var xhr=new XMLHttpRequest();  
xhr.onreadystatechange=function () {  
 if(xhr.readyState==4){  
 console.log(xhr.responseText)  
 }  
};  
xhr.open("POST","/ajax\_xhr/?p=123");

xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded; charset=UTF-8");

xhr.send("a=123;b=456");

//转换成obj

设置头文件  
xhr.setRequestHeader必须在open后

关于chrome的显示的发送类型：

Query String Parameters url携带的参数

form data post发送的时候携带参数

request payload post发送时携带的参数

form data和request payload的区别：

当request的content-type为multipart/form-data时，为request payload传输 为请求载荷

当request的content-type为x-www-form-urlencoded，为form data传输 为表单数据

默认就是multipart/form-data-----request payload

发送formdata

var xhr=new XMLHttpRequest();  
xhr.onreadystatechange=function () {  
 if(xhr.readyState==4){  
 console.log(xhr.responseText)  
 }  
};  
xhr.open("POST","/ajax\_xhr/?p=123");  
var form=new FormData();  
form.append("user","alex");  
form.append("pwd","123");  
xhr.send(form);

原生ajax发送文件

需要拿到文件标签的对象：document.getElementById(“user”).files[0]

jquery的ajax发送文件：

**var** fd = **new** FormData();  
{# console.log(this.files);#}  
 fd.append("file\_obj", **this**.files[0]);  
 $.ajax({  
 url: location.href,  
 type: "POST",  
 data: fd,  
 processData : **false**,  
 contentType : **false**,  
{# dataType: "JSON",#}  
 beforeSend:**function** () {  
 $("#exist-func .js-update-excel").text("正在发送" + "<" + $(btn\_temp).val() + ">");  
 },  
 success: **function** () {  
 $("#exist-func .js-update-excel").text("发送完毕");  
 }

[关于ajax异步](#_js单线程异步)

#### Jquery的ajax

$.ajax({

url:"/web/ajax/",

type:"POST",

data:{"username":$("#username").val(),"pwd":$("#pwd").val()},

success:function (data) {

if(data=="1"){

location.href="http://www.baidu.com/";

}else{

alert("用户名密码错误");

}

}

});

url：发送给谁

type：发送的类型

data:具体数据

success:提交成功后处理服务器返回的数据，并执行相应的脚本

error:失败后执行的函数

服务器返回给success用

return HttpResponse(user)

如果要传多个参数，可用json.

python转成json：json.dumps(str)，javascript转成字典：JSON.parse(0)

对于上述，jquery有一个快捷操作

在ajax中再加一个参数：

dataType:”json”,

jquery的另外两种发送ajax的方法：

$.get(url,function)

$.post(url,function)

Jquery循环—each()

serialize()

Jquery默认发送post的ajax时，加了请求头，用来转换成对象。如果用FormData后已经是对象就不用再设置请求头，在ajax中设置

processData:false //不用设置文件转换

contentType:false //不用设置请求头

header:xxx //请求头

$.each([1,2,3,4],**function** (k,v) {  
 console.log(k);  
})

K为索引，v为值s

一个return不会停止循环(continue)，只有return false才停止循环(break)

**快速收集form表单数据：**

全局csrf

全局Ajax中添加请求头X-CSRFToken，用于跨过CSRF验证  
 \*/  
$.ajaxSetup({  
 beforeSend: **function** (xhr, settings) {  
 **if** (!csrfSafeMethod(settings.type) && !**this**.crossDomain) {  
 xhr.setRequestHeader("X-CSRFToken", $.cookie('csrftoken'));  
 }  
 }  
});

#### 基于Iframe发送伪ajax

详见[iframe](file:///C:\Users\nextgo\AppData\Roaming\Microsoft\Word\html.docx)

### 表单验证

前端验证只能优化，当禁用js时前端验证也就失效了，所以后端也要做验证

jquery实现form表单ajax提交：

### jQuery自定义函数

$.extend() jQuery扩展函数,不可传选择器

$.fn.extend() 可以传选择器，this指的是fn选择器

$.extend({  
 'showName':**function** (name) {  
 console.log(name);  
 }  
});  
$.showName("aaa");

$.fn.extend({  
 showname:**function** (k) {  
 console.log(**this**,k);  
 }  
});  
$("ul").showname("ddd");

### 前端插件

Fontawsome

Bootstrap

Easyui

Jqueryui

Bxslider

jqueryLazyLoad

## 画布

<canvas width="320" height="240" class="myCanvas"></canvas>

<script>

var my\_canvas = document.querySelector(".myCanvas");

</script>

.width/.height