

Javascript

- 亂碼

- ```
<input type = 'button' value = 'OK' onclick = 'alert('OK')'>
```

- 脚本

- ```
<script>
    // alert
    alert('NO')
</script>
```

- 语句

语句

- alert
- console.log
- prompt
- document.write

```
// 语句
alert("OK");
// 打印hello world
console.log("hello world");
// 提交
prompt('请输入姓名');
// 打印body
document.write('<h1>Hello World</h1>')
```

语句语义

javascript语句语义

- 单行语句//
- 多行语句/* */
- 多行语句 ; 语句

语句语义

```
// 语句
var item = 45;
let item = 45;
const item = 45;
```

5语句语义

1. number 语句

2. string 틀어놓기
3. boolean 틀어놓기 true/false
4. undefined 틀어놓기 undefined
5. null 틀어놓기 null
6. object 틀어놓기
 - 틀어놓기 object
 - 틀어놓기 function

00typeof 틀어놓기

```
// 틀어놓기
var num = 20;
var string = '00';
var unData;
// 틀어놓기
var operson = {
  name:'itcast',
  age:12
};
// 틀어놓기
alert(typeof(num));    //number
alert(typeof(string)); //string
alert(typeof(unData)); //undefined
alert(typeof(unData)); //object

// 틀어놓기 틀어놓기
console.log(operson.name);
console.log(operson.age);
```

00TemplateString

- 틀어놓기` \${}` 틀어놓기

```
let name = 'whs';
let content = `00000${name}`;
console.log(content) //00000whs
```

00function 틀어놓기

```
// 틀어놓기
function fnAlert(){
  alert('hello!');
};

function add(num1,num2){
  var irs = num1 + num2;
  return irs
};
```

```
// 誤り
fnAlert();
var result = add(3,4);
alert(result); // 誤りresult
```

比較演算子

`x = 5`

値	式	結果
<code>==</code>	<code>x == 8</code>	<code>false</code>
<code>====</code>	<code>x === 5</code> ; <code>x === "5"</code>	<code>true</code> ; <code>false</code>

```
var score = 79;
if(score < 60){
    alert('誤り');
}else{
    alert('正解');
}
```

値	式
<code>&&</code>	<code>□</code>
<code> </code>	<code>□</code>
!	<code>□</code>

DOM操作

式	結果
<code>getElementsByClassName()</code>	要素の配列
<code>getElementById()</code>	要素のID
<code>getElementsByTagName()</code>	要素のタグ名
<code>getElementByName()</code>	要素のname属性

```
// クラスによる検索
let class = document.getElementsByClassName('my_class');
// IDによる検索
let id = document.getElementById('my_id');
// タグ名による検索
let tagname = document.getElementsByTagName('div');
// name属性による検索
let name = document.getElementsByName('my_name');
```

- document.getElementById() id html

 - onload

 - onclick

```
// id btn
var oBth = document.getElementById('btn');
alert(oBth);
// function fnload()
function fnload(){
    var oBth = document.getElementById('btn');
    alert(oBth);
}
// onload
window.onload = fnload;
// 
window.onload = function(){
    var oBth = document.getElementById('btn');
    alert(oBth);
}

// onclick
<input type = 'button' value = '按钮' id = 'bin' onclick = 'fnload()'>
```

CSS

方法	说明
querySelector()	单个元素
querySelectorAll()	所有元素

```
// 单个元素
let div = document.querySelector('div');
// class
let cla = document.querySelector('.class');
// id
let id = document.querySelector('#id');
// 所有元素
let divs = document.querySelectorAll('div');
```

事件处理程序

事件处理程序

```
// 事件处理程序
var obth = document.getElementById('bin');
// 事件处理程序
```

```
obtn.value = 'user_name';
// ...
obtn.style.background = 'red';

<input type = 'button' value = '登入' id = 'bin'>
```

JavaScript

1. html문서js문서를통해html을 className='클래스명'으로
2. 'style'문서를통해html을 fontSize='크기'로

innerHTML와innertext

- **innerHTML**은 html문서를통해 html을
- **innertext**은 html문서를통해 html을

```
window.onload = function(){
    // ...
    var odiv = document.getElementById('mydiv');
    // ...
    alert(odiv.innerHTML);
    // ...
    odiv.innerHTML = '登入';
}

<div id = 'mydiv'>登入</div>
```

```
// ...
let content = document.querySelector('.class').innertext
content = '登入'

<div class = 'class'> 登入 </div>
```

Array

Array의생성방법

- 1. new Array()

```
// ...
var alist = new Array(1,2,3);

// ...
var alist2 = [1,2,3,'asd'];

// ...
var alist3 = [[1,2,3],[4,5,6]];
```

- 2. [1,2,3]

```
var alist = [1,2,3,4];
alert(alist.length); // 4
```

- 4

```
var alist = [1,2,3,4];
alert(alist[0]); // 1
```

- 1

```
var alist = [1,2,3,4];
alist.push(5); // 4
alert(alist); // 12345
alist.pop(); // 4
alert(alist) // 1234
```

- 4

- .splice(start,num,element1,...elementN)
- start
- num
- elementN

start num elementN start

```
// 4
var array = [10,20,30,40];
array.splice(0,1,'itcast');
console.log(array); // itcast,20,30,40
```

4

+"4"

```
var inum1 = 10;
var inum2 = 11.1;
var str = 'abc';

result = inum1 + inum2;
alert(result); // 21.1

result = inum2 + str; // 11.1abc
alert(result); // 11.1abc
```

4

4+"4"

- js

- func
- delay
- param1

◎	◎
setTimeout(func[,delay,param1,param2])	延时调用
setInterval(func[,delay,param1,param2])	间隔调用

- 停止
 - clearTimeout(timeoutID)
 - clearInterval(intervalID)
 - **timeoutID**

```
// 延时调用
function fnshow(name){
    alert(name);
    clearTimeout(Id);
}

var Id = setTimeout(fnshow,2000,'itcast');
```

jQuery

JavaScript的子集JavaScript

jQuery

```
// 引入jQuery
<script src = 'jQuery'> </script>
```

jQuery

jQuery的ready事件

事件

```
// 事件
$(document).ready(function(){
    ...
});

// 事件
$(function(){
    ...
});
```

jQuery

jQuery 介绍

1. 选择器
2. 事件
3. id选择器
4. 类选择器
5. 伪选择器

```
$('#myid')           // id#myid
$('.myclass')         // class.myclass
$('li')               // li
$('#ul1 li span')   // id#ul1 li span
$('input[name=first]') // name:first input
```

jQuery对象 \$

```
$(function(){
  var $myp = $("p");
  alert($myp.length); // p的个数
});

$(function(){
  var $myobject = $("div p");
  alert($myobject.length); // div中p的个数
});

$(function(){
  var $myclass = $('.myclass');
  alert($myclass.length); // class.myclass的个数
});
```

方法

- **has(选择器)** 检查元素是否包含子元素
- **eq(索引)** 选择元素的第n个子元素

```
$(function(){
  // 选择器
  var $myobject = $("div");
  // 方法
  $myobject.css({"height":"100px"});
  // has()
  $myobject.has("#bin").css({"background":"red"});
  // eq()
  $myobject.eq(0).css({"background":"red"}); // 第1个div
});

<div> <input type = 'button' value = 'OK'> </div>
<div> <input type = 'button' value = 'OK'_> </div>
```

jQuery

jQuery API Reference

方法

- `$("#box").prev();` 之前 id 为 box 的元素
- `$("#box").prevAll();` 所有之前 id 为 box 的元素
- `$("#box").next();` 之后 id 为 box 的元素
- `$("#box").nextAll();` 所有之后 id 为 box 的元素
- `$("#box").parent();` 父级 id 为 box 的元素
- `$("#box").children();` 子级 id 为 box 的元素
- `$("#box").siblings();` 兄弟 id 为 box 的元素
- `$("#box").find('.myclass');` 所有 id 为 box 且 class 为 myclass 的元素

```
// 之前
var $mydiv = $('#div');
// 所有之前
alert($mydiv.prev());
// 之后
alert($mydiv.prevAll());
// 所有之后
alert($mydiv.next());
// 所有之后
alert($mydiv.nextAll());
// 父级
alert($mydiv.parent());
// 子级
alert($mydiv.siblings());
// 兄弟
alert($mydiv.children());
// class 为 myclass 的
alert($mydiv.find('.myclass'));
```

方法

- **html**
◦ `.html()` 替换所有子节点的 HTML 内容
◦ `.append()` 追加内容

```
// 之前
var $mydiv = $('#div');
// 所有
var $result = $mydiv.html();
alert($result);
// 所有
$mydiv.html('测试');
// 追加
$mydiv.append('测试');
```

jQuery

- height\weight\font-size\等
 - CSS 属性
- type\value\id\等
 - prop 属性

```
$function(){
    // 读取属性
    var $my = $("p");
    // 读取CSS
    var $style = $my.css('font-size');
    // 读取HTML
    $my.css({"background": 'red'});
    // 读取DOM
    var $myt = $('#btn');
    // 读取属性
    var $result = $myt.prop('type');
    alert($result);
    // 读取DOM
    $myt.prop({'value': 'name'});
    // 读取DOM
    $myt.val('name');
}

<p>itcast</p>
<input type = 'button' value = '提交' id = 'btn'>
```

jQuery事件

事件

- click()事件
- blur()事件
- focus()事件
- mouseover()事件
- mouseout()事件
- ready()事件

```
// ready事件
$function(){
    // 读取属性
    var $mytext = $('#text1');
    var $mybutton = $('#btn1');
    // click事件
    $mybutton.click(function(){
        alert('successfully')
    });
    // focus事件
    $mytext.focus(function(){
```

```

    // this이 mytext
    $(this).css({"background":"red"});
});
// blur() 이벤트
$mytext.blur(function(){
    $(this).css({"background":"white"});
});
// mouseover() 이벤트
$mydiv = $('div');
$mydiv.mouseover(function(){
    $(this).css({"background":"red"});
});
// mouseout() 이벤트
$mydiv.mouseout(function(){
    $(this).css({"background":"white"});
});
})
<div>
    <input type = 'text' id = 'text1'>
    <input type = 'button' value = '확인' id = 'btn1'>
</div>

```

이벤트

- \$(document).ready(function(){}) 이벤트
- \$(document).on('이벤트명', '선택자', function(){});
- \$(선택자).on('이벤트명', function(){});
- delegate의 경우
 - delegate(childSelector,event,function)
 - childSelector: 자식 선택자
 - event: 이벤트
 - function: 처리할 함수

```

$(function(){
    var $myul = $('ul')
    $myul.delegate('li','click',function(){
        $(this).css({"background":"red"})
    });
});

```

javascript 객체

기본 객체

- javascript object 기본 객체

- var person = new object();

 // 초기화
 person.name = 'tom';
 person.age = '25';

```
// 例題
person.sayname = function(){
    alert(this.name);
}

// 実行確認
alert(pered.name);
pered.sayname()
```

- オブジェクトの実行

- ```
var person = {
 name: 'tom';
 age: '26';
 sayname:function(){
 alert(this.name);
 }
}

// 実行確認
alert(person.name);
pered.sayname();
```

## Json

データ構造(オブジェクト)を文字列で表現するための規格

- オブジェクト
  - JSONはjsonと呼ばれる規格で、key:valueで構成される
- オブジェクトJSON

- ```
{
    "name": "tom",
    "age": 18
}
```

- オブジェクトJSON

- リスト構造で表現する
- ```
[{"name": "Tom", "age": 18}, {"name": "Jerry", "age": 23}]
```

- JSONとJavaScript
  - `JSON.parse()`

- ```

var sjson = '{\n    "name": "tom",\n    "age": 18\n};\nvar json = '[{"name": "小明", "age": 18}, {"name": "小红", "age": 23}]';

// 将字符串转为JavaScript对象\nvar operson = JSON.parse(sjson);

// 将对象转为字符串\nvar list = JSON.stringify();

```

ajax

- 通过JavaScript向HTTP服务器发送请求并接收响应
- 通过AJAX向HTTP服务器发送请求并接收响应
- ajax**
- 通过`$.ajax()`

Axios

通过Axios向HTTP服务器发送请求并接收响应

通过Axios向js发送请求

```

// 通过Axios向HTTP服务器发送请求并接收响应
axios({
  url: '',
  method: 'GET',
}).then((result) =>{
  console.log('成功');
}).catch((err) =>{
  console.log(err);
})

// 通过Axios向js发送请求
// 通过axios.方法(url[, , data[, config]])
axios.get('').then((result) =>{
  console.log('成功');
}).catch((err) =>{
  console.log(err);
})

```

async/await

- 通过async和await向HTTP服务器发送请求并接收响应
- 通过Promise的rejected和resolved向HTTP服务器发送请求并接收响应

□□

```
async search(){
  let result = await axios.get('');
  this.searchbox = result.data.data;
}
```

Vue□□□□

□□	□□□□
beforeCreate	□□□
created	□□□
beforeMount	□□□
mounted	□□□□
beforeUpdate	□□□□□
updated	□□□□□
beforeUnmount	□□□□□
unmounted	□□□□□