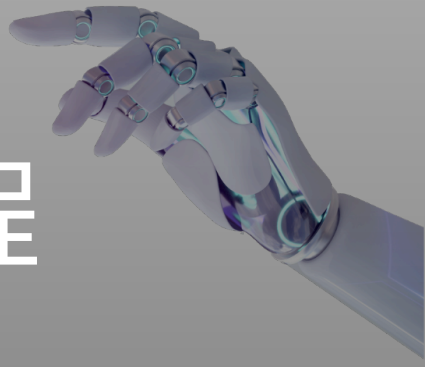


Web 前端编程

Web Front-end Programming

尹庆 Yin Qing
qyin@cuit.edu.cn

JavaScript

什么是JavaScript

- JavaScript：是一种轻量级、解释型、面向对象的**编程语言**

- 网页开发：使网页具有更强的交互性和动态性
  - 网页特效
  - 表单验证（判断用户输入内容的合法性）
  - 数据交互 （获取后台数据，渲染到前端）
- 后端开发：使用node.js在服务端运行，实现服务端应用开发

什么是JavaScript

- JavaScript的组成
  - ECMAScript：规定了 JS 基础语法，如变量声明、循环语句等
  - Web APIs：
    - DOM：操作**文档**，如对页面元素进行移动、添加、删除操作等
    - BOM：操作**浏览器**，如页面弹窗、检测窗口宽度、存储数据到浏览器等

1 HTML 使用 JavaScript 的方法

- 内部导入：在结束标签 `</body>` 前添加 JS 代码

```
<body>
  <!-- 页面内容相关的 HTML 代码 -->
  <script>
    console.log('Hello World!');
  </script>
</body>
```

💡 `<script>` 放在 HTML文件底部附近加载原因：浏览器会顺序加载HTML文件内代码， 而JavaScript只能修改已经被浏览器加载的HTML元素，所以放在所有内容元素之后

1 HTML 使用 JavaScript 的方法

- 外部导入（推荐）：使用 `<script>` 元素的 `src` 属性指定需要导入的 JS 文件
  - 建议使用 `相对路径` ，如 `src="scripts/example.js"`

```
<script src="example.js" defer></script>
```

- ⚠ `script` 元素用于外部导入时，元素中间部分写的JS代码会被忽略
- 💡 `<head>` 中导入 JS 文件需要配合使用 `defer` 属性
- 💡defer 属性将阻塞 DOMContentLoaded 事件触发，直到完成加载后才执行

2 基础语法

- 声明变量

```javascript
// JavaScript
let x = 10;
const y = 20;
```

```c
// C
int x = 10;
const int = 20;
```

- 声明函数

```javascript
// JavaScript
function add(a, b) {
  return a + b;
}
```

```c
// C
int add(int a, int b) {
  return a + b;
}
```

> 💡 JavaScript 的结束符 ； 不是必须的（C语言是必须的）

2 基础语法

- 创建数组

```javascript
// JavaScript
let arr = [1, 2, 3];
```

```c
// C
int arr[3] = {1, 2, 3};
```

- 条件语句

```javascript
// JavaScript
if (x > 10) { ... } else { ... }
```

```c
// C
if (x > 10) { ... } else { ... }
```

- 循环语句

```javascript
// JavaScript
for (let i = 0; i < 5; i++) { ... }
```

```c
// C
for (int i = 0; i < 5; i++) { ... }
```

2 基础语法

调试 Debug

- 方法1: "print"大法
  - 使用 `console.log()` 方法在调试窗口上打印变量值

```
a = 5;
b = 6;
c = a + b;
console.log(c);
```

- 方法2: 断点调试

```
F12 > 菜单栏 sources > 点击左侧栏选择文件 > 点击文件中行号处设置断点
```

3 对象

🤔 思考：保存网站用户信息（如姓名、年龄等）用一种基本数据类型保存方便吗？

```
let arr = ['ikun', 2.5, 'rap']
let str = 'ikun 2.5 rap'
```

- 不方便，很难区分数据含义
- 需要一种新的数据类型，能够更好的保存数据

3 对象

- 对象 (object)：JavaScript 中的一种**数据类型**
  - 可以理解为一种**无序的数据集合**
  - 用来描述某个事物，如人（姓名、年龄等）
  - 相比于用多个变量零散保存关于一个事物的信息，对象更集中、统一

```
// 语法
let 对象名 = {}
```

```
let teacher = {
  uname: '任课老师',
  age: 18,
  gender: '男'
}
```

3 对象

- 对象由**属性**和**方法**组成
  - 属性（变量）：信息或特征(名词)，如名字、年龄等
  - 方法（函数）：功能或行为(动词)，如打电话、~~启动原神~~

```
let 对象名 = {
  属性: 属性值,
  方法名: 函数
}
```

- 属性 = 对象内定义的**变量**

3 对象

- 对象本质是无序的数据集合，操作包括 **增、删、改、查、调用(函数)**
  - 查：查询对象属性，`对象名.属性` 或 `对象名['属性']`
  - 改：对象属性重新赋值，`对象名.属性 = 新属性值`
  - 增：为对象添加新的数据，`对象名.新属性名 = 新属性值`
  - 删：删除对象中的属性，`delete 对象名.属性名`
  - 调用：调用对象中的函数，`对象名.方法名()`

```
// 增
teacher.hobby = 'basketball'
// 删
delete teacher.uname
// 改
teacher.age = 28
```

```
// 查
console.log(teacher.age)
```

3 对象

🤔 以下代码打印的结果分别是多少?

```
let age = 18
b = age
b = 28
console.log(age)

let obj_1 = {
  age: 18
}
obj_2 = obj_1
obj_2.age = 28
console.log(obj_1.age)
```

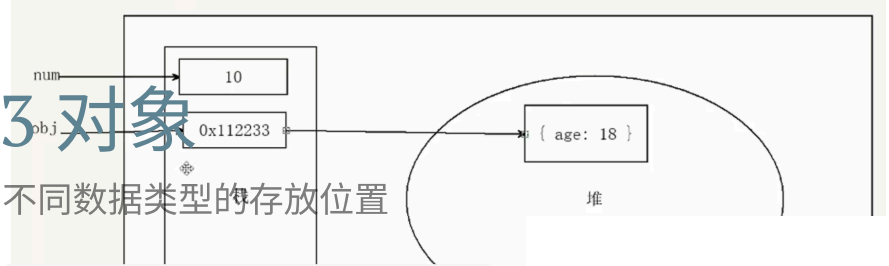
3 对象

不同数据类型的存放位置

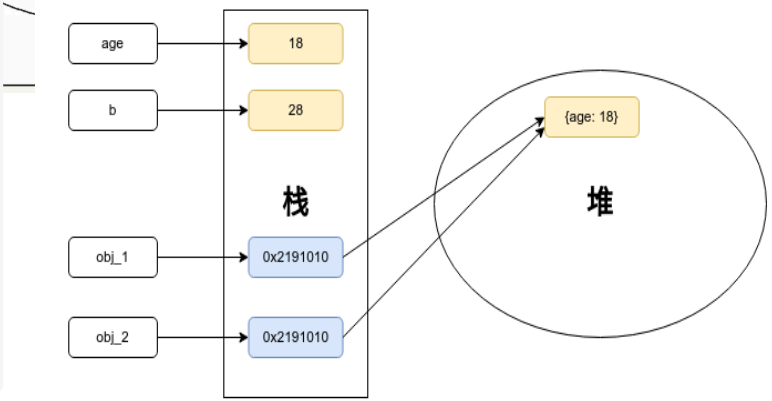
- 值类型：基本数据类型（如number, boolean），变量值存放在**栈**
- 引用数据类型：复杂数据类型（如Object, Array），变量值存放在**堆**
  - 栈：存放地址，用于在堆里面找值
  - 堆：存放变量值

3 对象

不同数据类型的存放位置




```
1   let age = 18
2   b = age
3   b = 28
4   console.log(age)
5
6   let obj_1 = {
7     age: 18
8   }
9   obj_2 = obj_1
10  obj_2.age = 28
11  console.log(obj_1.age)
```



Web APIs

操作 DOM、BOM

什么是 Web APIs

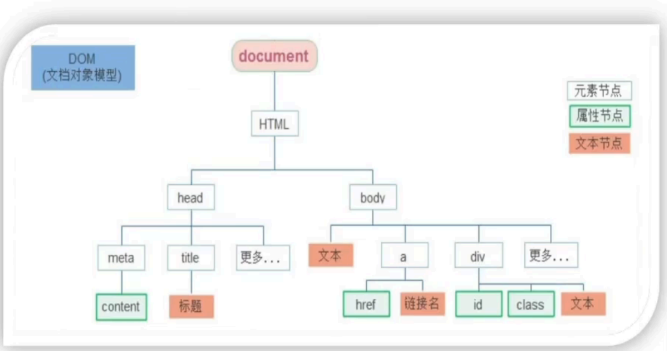
- 作用：使用 JavaScript 操作 HTML 和浏览器
- 分类：
  - DOM：文档对象模型
  - BOM：浏览器对象模型

1 什么是 DOM

- DOM (Document Object Model, 文档对象模型)：用于操作**网页内容**
  - 如对页面元素进行移动、添加、删除操作等

- DOM 树：将 HTML 文档转化为树状结构，该树状结构称之为文档树（DOM 树 🌳）
  - 作用：直观体现 HTML 元素与元素之间的关系，便于修改指定元素

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>标题</title>
</head>
<body>
  文本
  <a href="">链接名</a>
  <div id="" class="">文本</div>
</body>
</html>
```



1 什么是 DOM

- DOM 对象：浏览器根据 HTML元素 生成的 **JavaScript 对象**
  - 所有 HTML 元素属性都可以在 DOM 对象内找到
  - 修改 DOM 对象的属性会自动映射到 HTML 元素上

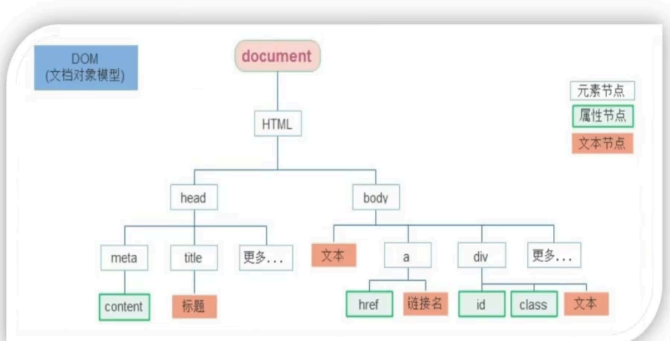
```
<body>
  <div>123</div>
  <script>
    const div = document.querySelector('div')
    console.dir(div) // 打印 DOM 对象
  </script>
</body>
```

💡 DOM 核心思想：把网页内容当作**对象**来处理

1 什么是 DOM

- `document` 对象：DOM 中最大的对象
  - 提供的属性和方法都是用来访问和操作网页内容，如 `document.write()`
  - 网页所有内容都在 `document` 对象里

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>标题</title>
</head>
<body>
  文本
  <a href="">链接名</a>
  <div id="" class="">文本</div>
</body>
</html>
```



2 获取 DOM 对象

- 根据 CSS 选择器来获取 DOM 元素（重点，主流方法）
  - `document.querySelector('CSS选择器')`：通过CSSS选择器语法 选取匹配到的 **第一个** HTML 元素对应的 JS 对象(HTMLElement)
  - `document.querySelectorAll('CSS选择器')`：选择匹配到的多个元素

```
<div class="box">abc</div>
<div>123</div>
<script>
  const box = document.querySelector('.box')
  const boxs = document.querySelectorAll('div')
  console.dir(box)
  console.dir(boxs)
</script>
```

2 获取 DOM 对象

- 其他获取 DOM 元素的方法（了解）

```
// 根据 `HTML 元素标签名` 获取 DOM 元素（对象集合）
document.getElementsByTagName('div')
// 根据 `类名` 获取 DOM 元素（对象集合）
document.getElementsByClassName('myclass')
// 根据 `ID` 获取 DOM 元素（第一个匹配）
document.getElementById('myid')
```

3 操作元素内容

- 修改 HTML 元素的文本内容
  - `对象名.innerText` 属性
  - `对象名.innerHTML` 属性

3 操作元素内容

- `innerText` 属性：用于修改/添加**纯**文本内容到对应的 HTML 元素

> 💡⚠ 添加内容显示为纯文本，不解析添加内容中的 HTML 元素标签

```
1  const box= document.querySelector('.box')
2  console.log(box.innerText)
3  // strong 元素标签不会被解析生效
4  box.innerText = '<strong>更改内容</strong>'
```

3 操作元素内容

- `innerHTML` 属性：用于修改/添加文本内容到对应的 HTML 元素

> 💡⚠ 会解析添加内容中的 HTML 元素标签

```
const box= document.querySelector('.box')
console.log(box.innerText)
// strong 元素标签能被解析生效
box.innerText = '<strong>更改内容</strong>'
```

> ✅ 对比
>
> - `innerText` 属性，只识别文本，不解析 HTML 元素标签
> - `innertHTML` 属性，能识别文本，还能解析 HTML 元素标签

4 操作元素属性

- 操作元素常用属性
- 如通过src更换图片

```
对象名.属性 = 属性值
```

```
// 1. 获取元素
const pic = Document.querySelector('img')
// 2. 操作元素
pic.src = './images/example.jpg'
pic.title = '示例图片'
```

5 操作 CSS 样式

- 通过 `style` 属性操作 CSS
- 通过类名 `className` 操作 CSS
- 通过 `classList` 操作 CSS

5 操作 CSS 样式

- 通过 `style` 属性操作 CSS

```
对象名.style.样式属性 = 值
```

例：

```
// 1. 获取元素
const box = Document.querySelector('.box')
// 2. 修改样式属性
box.style.width = '200px' // ⚠️ 注意加引号和单位
box.style.backgroundColor = 'teal' // 原CSS属性有连接符的用小驼峰命名法代替
```

5 操作 CSS 样式

🐱 课堂练习

1. 设置/修改 HTML 元素样式属性通过 DOM 对象的哪个属性设置?

```
box.???.height = 200
```

2. 上述代码赋值有什么错误?
3. 修改 HTML 元素的左内边距 `padding-left`，对应的 DOM 元素属性是什么?

5 操作 CSS 样式

- 通过 `style` 属性操作 CSS 的缺点：修改样式较多时，书写不太方便

```
box.style.width = '200px'
box.style.height = '200px'
box.style.color = 'red'
box.style.backgroundColor = 'teal'
```

> 💡 改进思路：通过赋值带有 CSS 样式的类名实现样式修改

5 操作 CSS 样式

- `className`：给元素设置一个 HTML 类名
  - 用于一个元素需要设置较多的样式时，配合 CSS **类选择器**设置样式
  - 新类名会**完全覆盖**掉原来的 HTML 类名

语法

```
// myclass 是一个 CSS 选择了的类名
元素名.className = 'myclass'
```

> 💡⚠注意是 `className` 不是 `class`

5 操作 CSS 样式

- `classList`（推荐使用）：给元素添加或删除一个 HTML 类名

语法

```
// 追加一个 HTML 类名
元素名.classList.add('类名')
// 删除一个 HTML 类名
元素名.classList.remove('类名')
// 切换一个 HTML 类名，有该类名就删除，没有就加上（类似开关灯）
元素名.classList.toggle('类名')
```

🤔 className 和 classList 设置类名最大的区别是什么?

5 操作 CSS 样式

🐱 课堂练习

- 自动轮播效果
  - 
  -
  -
  -
  -

> 💡自动切换配合 JS 内置定时器函数 `setInterval(函数名，间隔时间ms)`

6 事件监听

- 事件（event）：用户或浏览器与页面交互时发生的行为或动作

事件	描述
onClick	点击事件
onMouseOver	鼠标经过
onMouseOut	鼠标移出
onChange	文本内容改变事件
onSelect	文本框选中

onFocus	光标聚集
onBlur	移开光标

6 事件监听

- 事件监听：JS 程序检测是否有事件发生，如果有则调用预设的函数进行响应
  - 事件 = 函数触发的条件

语法：

```
对象名.addEventListener('事件类型', 要执行的函数)
```

> 💡⚠ 不推荐以下两种添加事件监听的方法

```
<!-- HTML 文件中， 通过 HTML 属性添加事件 -->
<button onClick="click_event()">点击按钮</button>
```

```
// JavaScript 文件中， 通过DOM属性： 对象名.on事件属性
button.onclick = function () { alert('Button clicked!'); };
```

6 事件监听

- 事件类型要添加引号，如 'click'

- 函数不会根据代码编写顺序执行，而是等到事件发生后才执行
- 事件每触发一次，函数就会执行一次

示例

```
<button>按钮</button>
<!-- JS 代码 -->
<script>
  const btn = Document.querySelector('buttion')
  btn.addEventListener('click', function () {
    alert('按钮被点击! ') })
</script>
```

6 事件监听

🐱 课堂案例

- 实现广告关闭效果