

# Web编程基础

## HTML基础

告诉浏览器显示什么内容

类型：

(1) 双标签：<标签 属性>内容</标签>

```
1 <p style="font-size: 29px;">Hello, world!</p>
```

(2) 单标签：<标签 属性/>

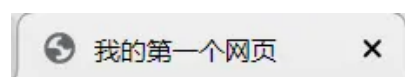
```
1 <meta charset="utf-8"/>
```

## 简单实例

```
1 <!DOCTYPE html>
2 <html lang="zh-cn">
3   <head>
4     <meta charset="utf-8">
5     <title>我的第一个网页</title>
6   </head>
7   <body>
8     <h1>你好，世界！</h1>
9     <p style="font-size: 29px;">Hello, world!</p>
10  </body>
11 </html>
```

你好，世界！

Hello, world!



<!DOCTYPE html>文件类型，尽管不加通常不会有什么问题，但养成良好的编程习惯尽量加上  
<head>会放一些文件的元信息（如标题、编码方式、搜索引擎显示的标题、简介等）。其中  
<title>就是标题，这个会在浏览器的标签页标题上显示  
<body>显示内容的主体，包括<h1>和<p>两个标签

## 问题

有时候标签的属性很多，如果存在大量相同属性的标签需要每次都列出所有属性，这样后面修改和维护起来也很麻烦。解决这个问题就引入了css

```
1 <p
2   title="this is a sentence"
3   style="
4     color: rgb(226, 114, 114);
5     font-size: 25px;
6     text-align: center;
7     text-shadow: 2px 2px #888;
8   "
9 >Sentence 1</p>
```

## Sentence 1

## CSS基础

告诉浏览器以什么样式显示内容，是对相同的标签属性的一个封装。

原来的写法：

```
1 <p
2   title="this is a sentence"
3   style="
4     color: rgb(226, 114, 114);
5     font-size: 25px;
6     text-align: center;
7     text-shadow: 2px 2px #888;
8   "
9 >Sentence 1</p>
```

对所有p标签增加css样式

```
1 <style>
2   p {
3     color: color: rgb(226, 114, 114);
4     font-size: 25px;
5     text-align: center;
6     text-shadow: 2px 2px #888;
7   }
8 </style>
9 <p title="this is a sentence">Sentence 1</p>
```

对所有class="sentence"的标签增加css样式

```
1 <style>
2   .sentence {
3     color: color: rgb(226, 114, 114);
4     font-size: 25px;
5     text-align: center;
6     text-shadow: 2px 2px #888;
7   }
```

```
8 </style>
9 <p class="sentence" title="this is a sentence">Sentence 1</p>
```

对所有id="sentence"的标签增加css样式

```
1 <style>
2   #sentence {
3     color: color: rgb(226, 114, 114);
4     font-size: 25px;
5     text-align: center;
6     text-shadow: 2px 2px #888;
7   }
8 </style>
9 <p id="sentence" title="this is a sentence">Sentence 1</p>
```

(1) 还有一些层次选择、特殊的选择器 (:hover、:fullscreen等)，有需要可以自行查阅

(2) css还可以写成单独的.css文件，然后在html文件中使用<style>标签引用这个文件，大家也可以去自行查阅

## JavaScript基础

### 基本内容

(1) JavaScript是一种解释型编程语言，与之相对的是编译型编程语言。

解释型的优点：跨平台性能好，编写一套代码可以在任何平台上运行

解释型的缺点：由于没有经过编译，运行速度会慢一些

在网络应用场景，通常更加看重跨平台，而运行效率相对而言不是很重要

(2) JavaScript代码可以直接嵌入HTML

(3) 基本功能

a. 读写HTML元素

b. 对浏览器事件（鼠标点击、悬停事件；键盘事件等）做出响应

c. 在nodejs出现之后，JavaScript有了更多的功能，但本课程主要涉及前两点

### 基本语法

(1) 变量声明：let, var, const

声明时不需指定类型（并不是说JavaScript没有类型）

let和var都是声明一个变量，但是两者有一些区别。建议都使用let。const声明常量

8种基本数据类型：Number、String、Boolean、Object、Null、Undefined、Symbol、BigInt。后两个本课程基本不会涉及。类型Null和Undefined分别都只有一个值（对应的小写形式null和undefined），都代表空值，但约定俗成地，null为定义了但是是空值，undefined为未定义（事实上任何声明了但是没有赋值的变量的值都是undefined）

(2) 多说一句Object

数据的格式经常是这种形式。例如我们的课程：

```
1 {
2   "className": "数据可视化",
3   start: "2023/09/27 18:40",
4   end: "2023/09/27 21:30",
5   location: "二教515",
6 }
```

可以看出，Object由一对对key-value对构成。key是一个字符串（可以不加引号），value可以是任何数据类型

Object的衍生类：Array、Set、Function、……

(3) 运算操作类似于Python

+, -, \*, /, %, \*\*, ... ...。乘方\*\*是ES7标准中引入的，IE浏览器可能会不支持

(4) 基本语法和C++类似

if条件语句、switch语句、for、while、do-while循环等

## 特性

主要会提一些后面可视化编程中可能碰到的特性，以及一些可能踩坑的地方。

(1) Object是引用类型

示例1：

```
1 let a = { x: 1 };
2 let b = a;
3 b.x = 2;
4 console.log(a.x); // 2
```

示例2：

```
1 let a = { x: 1 };
2 let b = { x: 1 };
3 console.log(a === b); // false
```

(2) ==和===

大体上，==可以理解成差不多相等，===可以理解成严格相等。建议尽量使用===

```
1 console.log(1 == '1'); // true
2 console.log(1 === '1'); // false
3 console.log(undefined == null); // true
4 console.log(undefined === null); // false
```

(3) Object操作的灵活性

```
1 let obj = {};
2 console.log(obj.x); // undefined
3 obj.x = 1;
4 obj['y'] = [1, 2, 3];
5 obj['z'] = function(d) {
6   return d * 2;
7 };
8 delete obj.x;
9 delete obj['y'];
```

(4) Function的多种形式

a. 传统定义

```
1 function f(args) {}
```

#### b. 匿名函数

```
1 let g = function(args) {};  
2 let h = (args) => {};
```

两者区别（大致了解即可，可能会遇到类似的问题）：this 指向不同。function有自己的this，箭头函数使用外部函数的this

匿名函数的作用：

```
1 let array = [1, 2, 3, 4, 5];  
2  
3 // 以下的结果都是[2, 4, 6, 8, 10]  
4 array.map(function(d) {  
5     return d * 2;  
6 });  
7 array.map((d) => {  
8     return d * 2;  
9 });  
10 array.map(d => d * 2);
```

#### (5) 链式写法

数组的filter方法返回的还是一个数组，可以紧接着调用其他方法。这个写法在d3.js中很常见

```
1 let array = [1, 2, 3, 4, 5];  
2 array  
3   .filter(d => d > 2)  
4   .map(d => d * 2)  
5   .reverse() // 最终结果为[10, 8, 6]
```

#### (6) 异步和同步

例1：输出1。因为setTimeout后不会等待，而是继续执行。

```
1 let a = 1;  
2 setTimeout(() => {  
3     a = 2;  
4 }, 1000);  
5 console.log(a);
```

例2：一个网络请求的例子。由于请求有延迟，请求结果得到后会执行相应的函数，但是在等待请求结果的期间会继续执行之后的代码（如果有的话）。最终先输出”Other works”，后输出”Network response”。

```
1 $.get(  
2     "vis.pku.edu.cn/course/datavis_f23",  
3     {},  
4     function(response) {  
5         console.log("Network response");  
6     }  
7 );
```

```
8 console.log("Other works");
```

(7) 加分号vs.不加分号

存在争议，具体看个人习惯就好。

(8) 一些奇怪的特性

```
1 (! + [] + [] + ! []).length  
2 // 9
```