WEB前端

介绍

HTML：网页的骨架没有任何的样式

CSS：给骨架添加各种样式变得好看

JS：控制网页的动态效果

前端框架：BOOTSTRAP、JQuery、Vue

提前给你封装好了很多操作你只需要按照固定的语法调用即可

HTTP

介绍

HTTP报文格式由请求报文和响应报文两种，都是由三部分组成：起始行（Start Line）、头部（Headers）和正文（Body）。以下是这两种报文的详细格式。

四大特性

1.基于请求响应

2.基于TCP/IP作用于应用层之上的协议

3.无状态，不保存用户的信息

eg：一个人来了一千次你都记不住每次都当他如初见

由于HTTP协议是无状态的所以后续出现了一些专门用来记录用户状态的技术

cookie、session、token...

4.无/短链接

请求来一次我响应一次之后我们两个就没有任何链接和关系了

长链接：双方建立连接之后默认不断开websocket（后面讲项目的时候会讲）

1. HTTP请求报文（Request Message）

请求报文格式：

请求行

请求头部

空行

请求正文（可选）

各部分说明：

* **请求行（Request Line）**：描述客户端发出的请求，包括请求方法、请求目标和HTTP协议版本。
  + 例子：GET /index.html HTTP/1.1
  + 组成：
    - **请求方法**（Method）：如GET、POST、PUT、DELETE等。
    - **请求目标**（Request Target）：即URL路径，如/index.html。
    - **HTTP协议版本**（HTTP Version）：如HTTP/1.1。
* **请求头部（Headers）**：包含关于客户端、请求及响应的附加信息。每个头部字段由一个字段名和字段值组成。
  + 例子：

Host: www.example.com

User-Agent: Mozilla/5.0

Accept-Language: en-US,en;q=0.5

* + 头部字段（Headers）有很多种，常见的有：
    - Host：指定请求的主机。
    - User-Agent：指定发出请求的浏览器类型。
    - Accept：告诉服务器客户端可以处理的媒体类型。
    - Content-Type：请求体的媒体类型。
    - Authorization：包含身份验证信息。
* **空行（Empty Line）**：请求头部和正文之间的空行（一个换行符）。
* **请求正文（Request Body）**：可选，通常在POST、PUT请求中会有数据，传输客户端的数据。
  + 例子：表单数据或JSON数据。

2. HTTP响应报文（Response Message）

响应报文格式：

状态行

响应头部

空行

响应正文（可选）

各部分说明：

* **状态行（Status Line）**：描述服务器处理请求的结果。
  + 例子：HTTP/1.1 200 OK
  + 组成：
    - **HTTP协议版本**（HTTP Version）：如HTTP/1.1。
    - **状态码**（Status Code）：如200（成功）、404（未找到）、500（服务器错误）等。
    - **状态消息**（Status Message）：状态码的描述，如OK、Not Found等。
* **响应头部（Headers）**：包含关于响应及返回内容的附加信息。
  + 例子：

Content-Type: text/html; charset=UTF-8

Content-Length: 3052

Server: Apache

* + 常见的响应头部：
    - Content-Type：响应体的媒体类型。
    - Content-Length：响应体的长度。
    - Server：服务器软件信息。
    - Set-Cookie：设置cookie。
* **空行（Empty Line）**：响应头部和正文之间的空行（一个换行符）。
* **响应正文（Response Body）**：包含请求的实际数据，通常是HTML文档、图片、JSON数据等。
  + 例子：返回的HTML代码或JSON数据。

示例：HTTP请求与响应

请求示例：

GET /index.html HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0

Accept-Language: en-US,en;q=0.5

响应示例：

HTTP/1.1 200 OK

Content-Type: text/html; charset=UTF-8

Content-Length: 3052

Server: Apache

<html>

<head><title>Example</title></head>

<body>

<h1>Welcome to Example</h1>

</body>

</html>

响应状态码

用一串简单的数字来表示一些复杂的状态或者描述性信息404：请求资源不存在

1XX：服务端已经成功接收到了你的数据正在处理，你可以继续提交额外的数据

2XX：服务端成功响应了你想要的数据（2000K请求成功）

3XX：重定向（当你在访问一个需要登陆之后才能看的页面你会发现会自动跳转到登陆页面）

4XX：请求错误

404：请求资源不存在

403：当前请求不合法或者不符合访问资源的条件

5xx：服务器内部错误（500serverintern）

请求方式

1.get请求

朝服务端要数据

eg：输入网址获取对应的内容

2.post请求

朝服务端提交数据

eg：用户登陆输入用户名和密码之后提交到服务端后端做身份校验

HTML

介绍

HTML（HyperText Markup Language，超文本标记语言）是一种用于创建网页和网上信息的结构的标记语言。它由一系列的元素组成，这些元素告诉浏览器如何展示内容。HTML文档由HTML元素组成，HTML元素是由标签包围的文本组成。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类别** | **标签** | **描述** | **属性示例** | **备注** |
| 文档结构 | <!DOCTYPE  html> | 声明文档类型和HTML版本。 | 无 | 必须放在HTML文档的第一行。 |
| <html> | 包围整个HTML文档。 | lang="en" | 定义文档的语言。 |
| <head> | 包含文档的元数据，如 <title> 和 <link> 标签。 | 无 | 包含不显示在页面上的元信息。 |
| <body> | 包含可见的页面内容。 | bgcolor="yellow" background="image.jpg" | 设置页面背景颜色或图片。 |
| 文本内容 | <h1> 到  <h6> | 定义标题。 | align="center" style="color: blue;" | <h1> 是最大的标题，<h6> 是最小的标题。 |
| <p> | 定义段落。 | align="justify" class="text" | 段落自动换行。 |
| <strong> 或 <b> | 定义粗体文本。 | style="font-weight: bold;" | <strong> 表示重要文本，<b> 仅表示粗体样式。 |
| <em> 或  <i> | 定义斜体文本。 | style="font-style: italic;" | <em> 表示强调文本，<i> 仅表示斜体样式。 |
| 链接和图片 | <a> | 定义超链接。 | href="url" target="\_blank" title="description" | href 指定链接地址，target 指定打开方式，title 提供额外信息。 |
| <img> | 定义图像。 | src="image.png" alt="description" width="100" height="100" | src 指定图像路径，alt 提供图像无法显示时的替代文本。 |
| 列表 | <ul> | 定义无序列表。 | type="circle/square/disc" class="list" | type 改变列表标记的类型。 |
| <ol> | 定义有序列表。 | type="1/A/a/I/i" start="5" | type 改变数字的类型，start 指定开始的数字。 |
| <li> | 定义列表项。 | value="5" class="item" | value 指定有序列表的起始值。 |
| 表格 | <table> | 定义表格。 | border="1" width="100%" cellpadding="5" | border 设置边框宽度，cellpadding 设置单元格内边距。 |
| <tr> | 定义表格行。 | bgcolor="grey" align="center" | 设置行的背景颜色和对齐方式。 |
| <th> | 定义表头单元格。 | scope="col" rowspan="2" colspan="3" | scope 定义单元格是表头还是跨行/列，rowspan 和 colspan 控制合并行或列。 |
| <td> | 定义表格单元格。 | rowspan="2" colspan="3" | 控制单元格合并行或列。 |
| 表单 | <form> | 定义表单。 | action="submit.php" method="get/post" | action 指定表单数据发送地址，method 指定提交方式。 |
| <input> | 定义输入控件。 | type="text/password/radio/checkbox" name="field" value="default" | type 指定输入类型，name 和 value 指定字段名和默认值。 |
| <button> | 定义按钮。 | type="submit/reset" name="btn" | type 指定按钮类型。 |
| 语义化标签 | <header> | 定义文档或部分的页眉。 | 无 | 用于引入页面或部分的头部内容。 |
| <footer> | 定义文档或部分的页脚。 | 无 | 用于引入页面或部分的底部内容。 |
| <section> | 定义文档中的一个区段。 | 无 | 表示文档中的一个独立区段。 |
| <article> | 定义独立的内容区块。 | 无 | 用于独立的内容，如博客文章或新闻故事。 |
| <aside> | 定义内容之外的侧边栏。 | 无 | 表示与页面主要内容稍微相关的部分。 |
| 分组和结构 | <div> | 用于对HTML文档中的块级元素进行分组。 | class="container" id="main" | 块级元素，常用于CSS布局。 |
| <span> | 用于对HTML文档中的内联元素进行分组。 | class="text" id="title" | 内联元素，用于不影响页面布局的文本分组。 |
| CSS和JavaScript | <style> | 在HTML中定义CSS样式。 | type="text/css" media="screen" | 用于内部样式表。 |
| <script> | 嵌入或引用JavaScript代码。 | src="script.js" type="text/javascript" | src 用于引用外部JavaScript文件，type 指定脚本类型。 |

注释

#注释：注释是代码之母

<!--单行注释-->

<!--

多行注释1

多行注释2

多行注释3

-->

由于HTML代码非常的杂乱无章并且很多，所以我们习惯性的用注释来划定区域方便后续的查找

<！--导航条开始-->

导航条所有的html代码

<!--导航条结束-->

<!--左侧菜单栏开始-->

左侧菜单栏的HTM1代码

<!--左侧菜单栏结束-->

标签的分类

分为双标签和单标签

#1块级标签：独占一行

h1~h6、p、div标签

1. 块儿级标签可以修改长宽行内标签不可以修改了也不会变化

2. 块儿级标签内部可以嵌套任意的块儿级标签和行内标签

但是p标签虽然是块儿级标签但是它只能嵌套行内标签不能嵌套块儿级标签

如果你套了问题也不大因为浏览器会自动帮你解开（浏览器是直接面向用户的不会轻易的报错哪怕有报错用户也基本感觉不出来）

总结：

只要是块儿级标签都可以嵌套任意的块儿级标签和行内标签

但是p标签只能嵌套行内标签（HTML书写规范）

#2行内标签：自身文本多大就占多大

i、u、s、b、span标签

行内标签不能嵌套块儿级标签可以嵌套行内标签

总结：大的可以套小的，小的不能套大的。

标签具有的两个重要内容

1.id值

类似于标签的身份证号在同一个html页面上id值不能重复

2.class值

该值有点类似于面向对象里面的继承一个标签可以继承多个class值

标签既可以有默认的书写也可以有自定义的书写

<p id="d1" class="c1" username="" jasonpassword="123"></p>

文档结构

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <title>Document</title>  
</head>  
<body>  
  
</body>  
</html>

<!DOCTYPE html>: 声明文档类型和HTML版本，不包含属性，必须放在HTML文档的第一行。

<html>: 包围整个HTML文档，可以包含lang属性来定义文档的语言。

<head>: 包含文档的元数据，如<title>和<link>标签，不包含属性。

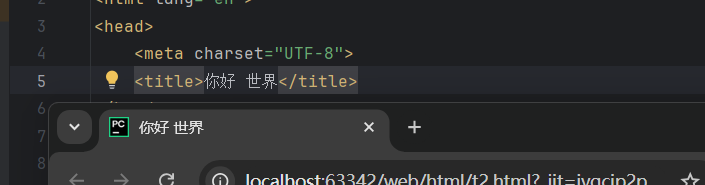
<head>内常用标签：

* <title>：定义文档的标题，显示在浏览器的标题栏或页面标签上。

<head>

<title>页面标题</title>

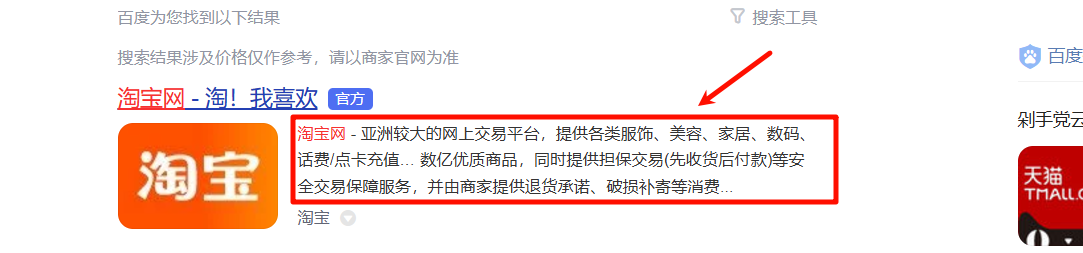
</head>



* <meta>：提供关于HTML文档的元数据。比如字符集声明、页面描述、关键词等。

<meta charset="UTF-8">

<meta name="description" content="页面描述">



<meta name="keywords" content="关键词1,关键词2"> #给搜索引擎的搜索关键词

<meta http-equiv="refresh"content="2;URL=https://www.oldboyedu.com"> #2秒后跳转https://www.oldboyedu.com

* <link>：链接外部资源，通常用于链接样式表。

<link rel="stylesheet" href="styles.css"> #引入外部css文件

* <script>：在文档中嵌入或引用脚本。

<script src="script.js"></script> #引用外部文件

<script>

js代码

</script>

* <style>：在文档内部包含样式信息（CSS）。

<head>

<style>

/\* CSS规则 \*/

</style>

</head>

* <base>：指定了文档中所有相对URL的基准URL。

<base href="http://www.example.com/">

<body>: 包含可见的页面内容，可以包含如bgcolor和background属性来设置背景颜色或图片。

body内常用标签

基本标签

<h1>我是h1</h1>标题标签1～6级标题

<b>加粗</b>

<i>斜体</i>

<u>下划线</u>

<s>删除线</s>

<p>段落</p>

<br>换行

<hr>水平分割线

特殊字符

&nbsp; 空格

&gt; 大于号 >

&lt; 小于号 <

&amp; &

&yen; ￥

&copy; ©

&reg; ®

常用标签

div块儿级标签

span行内标签

上述的两个标签是在构造页面初期最常使用的页面的布局，一般先用div和span占位之后再去调整样式，尤其是div使用非常的频繁。div你可以把它看成是一块区域，也就意味着用div来提前规定所有的区域，之后往该区域内部填写内容即可，而普通的文本先用span标签。

img标签

#图片标签

<img src="” alt=””>

src

1.图片的路径可以是本地的也可以是网上的

2.ur1自动朝该ur1发送get请求获取数据

alt="这是我的前女友”

当图片加载不出来的时候给图片的描述性信息

title="新垣结衣"

当鼠标悬浮到图片上之后自动展示的提示信息

height="800px”

width=""

高度和宽度当你只修改一个的时候另外一个参数会等比例缩放

如果你修改了两个参数并且没有考虑比例的问题那么图片就会失真

a标签

#链接标签

<a href=""></a>

当a标签指定的网址从来没有被点击过那么a标签的字体颜色是蓝色

如果点击过了就会是紫色（浏览器给你记忆了）

href

1.放url，用户点击就会跳转到该ur1页面

2.放其他标签的id值点击即可跳转到对应的标签位置

target

默认a标签是在当前页面完成跳转target=“\_self“

你也可以修改为新建页面跳转target=“\_blank“

#a标签的锚点功能

点击一个文本标题页面自动跳转到标题对应的内容区域

<a href=""id="d1">顶部</a>

<div style="height: 1000px;background-color: red"></div>

<a href=""id="d2">中间</a>

<div style="height:1000px;background-color: greenyellow"></div>

<a href="#d1">底部</a>

<ahref="#d2">回到中间</a>

列表标签

·无序列表(较多）

<ul>

<1i>第-项</1i>

<1i>第二项</1i>

<1i>第二项</1i>

<1i>第二项</1i>

</ul>

虽然u1标签很丑但是在页面布局的时候只要是排版一致的几行数据基本上用的都是u1标签

·有序列表(了解)

<ol type="1"start="5">

<1i>111</1i>

<1i>222</1i>

<1i>333</1i>

</o1>

1AIa...参考博客了机即可

·标题列表(了解)

<d1>

<dt>标题1</dt>

<dd>内容1</dd>

<dt>标题2</dt>

<dd>内容2</dd>

<dt>标题3</dt>

<dd>内容3</dd>

</dl>

表格标签

jason 123 read

egon123 dbj

tank123 hecha

<table>

<thead> #表头（字段信息）

<tr> #一个个tr就表示一行

<th>username</th> #加粗文本

<td>username</td> #正常文本

</tr>

</thead>

<tbody>

<tr> #表单（数据信息）

<td>jason</td>

<td>123</td>

<td>read</td>

</tr>

</tbody>

</table>

<table border="1">加外边宽

<td colspan="2">egon</td>水平方向占多行

<td rowspan="2">DBJ</td>垂直方向占多行

#原生的表格标签很丑但是后续我们一般都是使用框架封装好的很好看

表单标签

能够获取前端用户数据（用户输入的、用户选择、用户上传..·）基于网络发送给后端服务器

<form action=""></form>

#写一个注册功能

<form action="">

<input type="text">  
<button>点我</button>  
<textarea></textarea>

</form>在该form标签内部书写的获取用户的数据都会被form标签提交到后端

action：控制数据提交的后端路径（给哪个服务端提交数据）

1.什么都不写默认就是朝当前页面所在的ur1提交数据

2.写全路径：https：//www.baidu.com朝百度服务端提交

3.只写路径后缀action='/index/

自动识别出当前服务端的ip和port拼接到前面host:port/index/

<label for="d1">点击文本也能选中input输入框

for=id，id代表要指定的那个输入框的id值

<label for="d1">第一种直接讲input框写在label内

username:<input type="text" id="d1">

</label>

<label for="d2">password:</label>

<input type="text"id="d2">

ps:input不跟labe1关联也没有问题

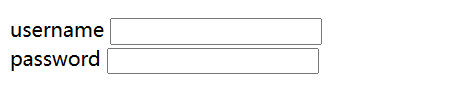
label和input都是行内标签

input标签，一般套在label标签里

type属性

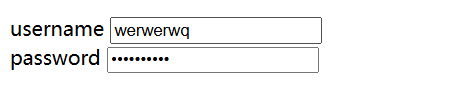
text：普通文本 文本输入框

username <input type="text" id="1">



password：密文 文本输入框

password <input type="password" id="2">



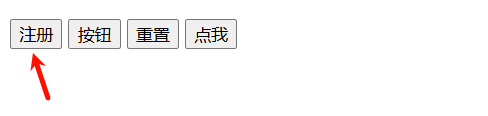
date:日期

生日 <input type="date" id="3">



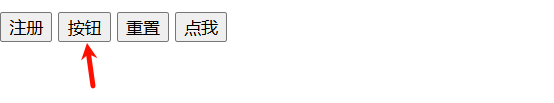
submit：用来触发form表单提交数据的动作

<input type="submit" value="注册"> value指定按钮中间显示的内容



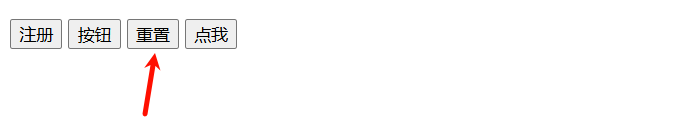
button：就是一个普普通通的按钮本身没有任何的功能但是它是最有用的，学完js之后可以给它自定义各种功能

<input type="button" value="按钮">



reset：重置内容

<input type="reset" value="重置">



radio：单选

默认选中男hecked='checked’

<input type="radio" name="gender" checked='checked'>男 # checked代表默认选择男

当标签的属性名和属性值一样的时候可以简写

<input type="radio" name="gender" checked >女 #也可以这样写checked



checkbox：多选

<input type="checkbox">read  
<input type="checkbox">read  
<input type="checkbox">read



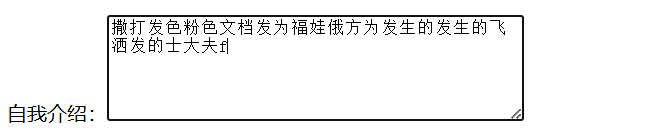
file：获取文件也可以一次性获取多个

file:<input type="file" multiple>



textarea标签，输入大段文本

自我介绍：<textarea name="" id="" cols="30" rows="10"></textarea>



button标签，点击后会提交数据

<button>点我</button>



select标签，默认单选，multiple多选

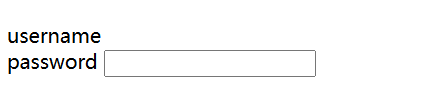
<select name="" id="" multiple> # multiple可多选  
 <option value="">北京</option>  
 <option value="" id="001">上海</option>  
 <option value="" selected>湖南</option>  
</select>



其它属性

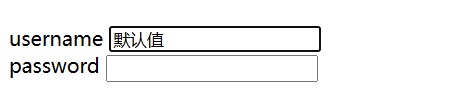
hidden：隐藏输入框

username <input type="text" id="1" name="username" formmethod="post" hidden="hidden">



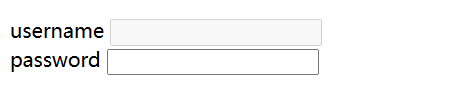
readonly：只读，不能输入内容

username <input type="text" id="1" name="username" value="默认值" readonly>



disabled：使输入框失效

username <input type="text" id="1" name="username" formmethod="post" disabled="disabled">



name和value：以kv值提交数据给服务器

例如：

代码

<label for="1">用户名

    <input type="username" value="123" id="1" name="username"/>

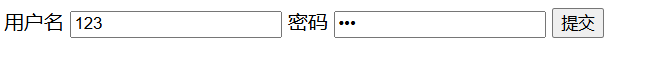
</label>

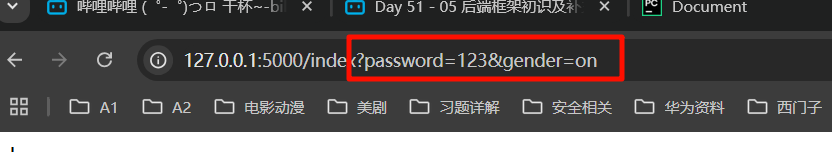
<label for="2">密码

    <input type="password" id="2" name="password"/>

</label>

输入框





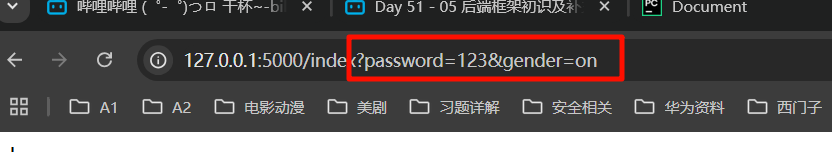
能触发form表单提交数据的按钮

1、<inputtype="submit”value="注册">

2、<button>点我</button>

form表单属性以及提交数据

提交数据说明



在输入标签中要有name属性，和value属性

提交的时候就是这样的数据：password=123&gender=0n

例如：

username <input type="text" id="1" name="username" value="默认值">

文本框不需要value，如果有value值，就会传入value值

非文本框需要value

form表单默认提交数据的方式是get请求数据是直接放在ur1后面的

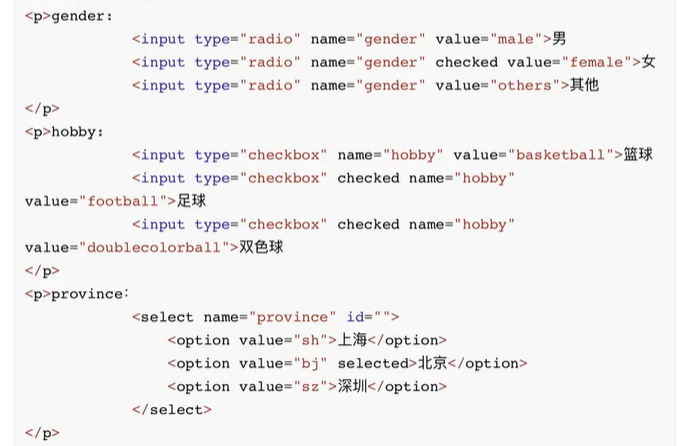
<http://127.0.0.1:5000/index/?username=sdadasdsda&gender=on>

你可以通过method指定提交方式

<form action=<http://127.0.0.1:50oo/index/> method="post"> #数据在请求体里面，如果是get方法数据在查询参数里面

<http://127.0.0.1:5000/index>

针对用户选择的标签用户不需要输入内容但是你需要提前给这些标签添加内容value值



medtod和enctype属性

form表单提交文件需要注意

1.method必须是post

2.enctype="multipart/form-data" #支持提交文件

enctype类似于数据提交的编码格式

默认是urlencoded只能够提交普通的文本数据

formdata就可以支持提交文件数据

<formaction=<http://127.0.0.1:5000/index/> method="post" enctype="multipart/form-data">

CSS

CSS（层叠样式表，Cascading Style Sheets）是一种用于描述HTML或XML文档的表现形式的语言。CSS可以用来控制网页的布局、颜色、字体以及其他视觉和版式效果

注释

#注释

/\*单行注释\*/

/\*

多行注释1

多行注释2

多行注释3

\*/

通常我们在写cSs样式的时候也会用注释来划定样式区域（因为HTML代码多所以对呀的cSs代码也会很多）

/\*这是博客园首页的css样式文件\*/

…

/\*顶部导航条样式\*/

…

/\*左侧菜单栏样式\*/

…

/\*右侧菜单栏样式\*/

…

语法

选择器{

属性1：值1；

属性2：值2；

属性3：值3；

属性4：值4；

}

三种引入方式

* 1. style标签内部直接书写（为了教学演示方便我们用第一种）

<style>

h1 {color:burlywood;}

</style>

* 2. link标签引l入外部css文件（最正规的方式解耦合）

<link rel="stylesheet" href="mycss.css">

* 3. 行内式（一般不用）

<h1 style="color：green">老板好要上课吗?</h1>

选择器

1. ID选择器：根据元素的id属性选择元素，id前加’#’。

#header {

font-size: 24px;

}

1. 类选择器：根据元素的class属性选择元素，只要包含就能选中，类名前面加‘.’。

.highlight {

background-color: yellow;

}

1. 标签选择器：根据元素的类型（标签名）选择元素。

p {

color: red;

}

1. 通用选择器：选择所有元素，‘\*’。

\* {  
 color: yellow;  
}

1. 后代选择器：选择作为某元素所有后代的元素，中间加空格。

div p { #选择所有在<div>元素内部的<p>标签

margin: 10px;

}

1. 子选择器：只选择作为某元素直接儿子元素的元素，孙子不选。

ul > li { #<ul>元素内部的<li>标签

list-style-type: none;

}

1. 毗邻选择器：选择紧接在另一元素后的元素，下面第一个同级元素。

h1 + p { #选择紧跟在<h1>元素后的<p>标签

color: blue;

}

1. 弟弟选择器：选择匹配选择器的元素的所有的元素，从h1往下选择同级p。

h1 ~ p { #选择所有<h1>元素之后的所有同级<p>标签

color: gray;

}

1. 属性选择器：根据元素的属性及其值选择元素，。

input[type="text"] { #选择所有type=text的<input>元素

border: 1px solid black;

}

1. 组合选择器：可以组合以上选择器来更精确地选择元素,。

* 并选择

div.highlight#header { #选择ID为header，并且具有class="highlight"的<div>元素

border: 2px solid blue;

}

* 或选择

div,.high,#header{ #选择所有div标签，class=high，id=header的元素

border: 2px solid blue;

}

* 嵌套选择

div .high #header{ #选择div标签内部，所有的class=high标签内部，id=header的元素

border: 2px solid blue;

}

1. 伪类选择器：根据元素的特殊状态选择元素。

a:link{ #没点击之前的a标签状态  
 color: red;  
}  
a:hover{ #鼠标悬浮的状态  
 color: yellow;  
}  
a:active{ #点击中的状态  
 color: aqua;  
}  
a:visited{ #点完后的状态  
 color: darkgray;  
}

input：focus{ /\*input框获取焦点（鼠标点了input框）\*/

background-color: red;

}

1. 伪元素选择器：根据元素的特殊部分选择元素。

p:first-letter{ #作用段的第一个字  
 font-size: 48px;  
 color: orange;  
}  
p:before{ #作用于段之前，不可选中  
 color: red;  
 content: 'abc';  
}  
p:after{ #作用于段之后，不可选中  
 color: greenyellow;  
 content: 'xyz';  
}

选择器优先级

1.选择器相同书写顺序不同

就近原则：谁离标签更近就听谁的

2.选择器不同。。。

行内>id选择器>类选择器>标签选择器

精确度越高优先级越高

高度（y轴）和宽度（x轴）

span标签没有height和width

p{  
 background: red;  
 height: 10px; #高度  
 width: 800px; #宽度  
}  
span{  
 background: green ;  
 height: 100px;  
 width: 100px;  
}

字体属性

P{

/\*font-family：“ArialBlack"，"微软雅黑"，"..."； 字体第一个不生效，就用后面的写多个备用\*！\*/

/\*font-size:24px; 字体大小\*！\*/

/\*font-weight:inherit; bolder（粗） lighter（细） 100~900（数组表示粗细） inherit继承父元素的粗细值\*！\*/

/\*color：red； ！\*直接写颜色英文\*！\*/

/\*color：#ee762e; ！\*颜色编号\*！\*/

/\*color：rgb（128，23，45）； ！\*三基色数字范围0-255\*1\*/

/\*color：rgba（23，128，91，0.9）； ！\*第四个参数是颜色的透明度范围是0-1\*！\*/

/\*当你想要一些颜色的时候可以利用现成的工具

pycharm提供的取色器

qq或者微信截图功能

文字属性

text-align对齐：center（居中），right（右对齐），left（左对齐）justify（两端对齐）

p{  
 text-align: center;  
}

text-decoration装饰：underline（下划线），overline（上划线），line-through（删除线），none（清除格式）

p{  
 text-decoration: underline;  
}

a{

text-decoration: none; #删除a标签自带的下划线

}

背景

/\*background-color:red;\*/

/\*背景图片\*/

/\*background-image：urL（"222.png")；！\*默认要全部铺满\*！\*/

/\*background-repeat:no-repeat;!\*不铺\*!\*/

/\*background-repeat:no-repeat;！\*不铺\*！\*/

/\*background-repeat:repeat-x; x方向重复

/\*background-repeat:repeat-y; y方向重复

/\*background-position:center center；！\*居中\*！\*/

/\*如果出现了多个属性名前缀是一样的情况一般情况下都可以简写只写前缀\*/

background:red url（"222.png") no-repeat center center;

background-attachment:fixed; 吸附

边框

border-width:5px; #边框宽度

border-style: solid; #实线边框

border-color:green; #边框颜色

border-left-width: 5px; #左边框宽度

border-left-color:red; #左边框颜色

border-left-style: dotted; #左边框点虚线

简写

border:3px solid red;/\*三者位置可以随意写\*/

画圆

border-radius:50%; /\*直接写50%即可长宽一样就是圆，不一样就是椭圆\*/

display属性

dispLay：bLock； ！\*将标签设置成块儿级标签的特点\*！\*/

display: inline; ！\*将标签设置为行内标签的特点\*！\*

display:inline-block; /\*标签即可以在一行显示又可以设置长宽\*/

display:none; 隐藏标签，位置也不存在

visibiLity：hidden； 单纯的隐藏位置还在

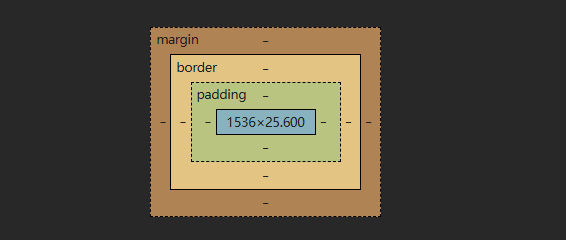
盒子模型

快递盒与快递盒之间的距离（标签与标签之间的距离margin外边距）

盒子的厚度（标签的边框border）

盒子里面的物体到盒子的距离（内容到边框的距离padding内边距）

物体的大小（内容content）



margin用法

2个相邻标签的margin距离只取最大的

margin-left:0; 左

margin-top:0； 上

margin-right:0; 右

margin-bottom:o; 底

margin：0；/\*上下左右全是0

/\*margin：10px 20px；！\*第一个上下 第二个左右\*！\*/

/\*margin：10px 20px 30px；！\*第一个上 第二个左右 第三个下\*！\*/

/\*margin:10px 20px 30px 40px;！\*上右下左\*1\*/

margin:0 auto; #上0，左右居中

padding用法和margin一样

padding-left:10px;

浮动

CSS中的浮动（Float）是一种布局技术，它允许你将元素向左或向右移动，直到它的外边缘触及包含框或另一个浮动元素的边缘。浮动元素会脱离正常的文档流，这意味着它不会被后面的内容所覆盖，而后面的内容会围绕浮动元素进行布局。

1. **脱离文档流**：浮动元素将不在文档流中占据位置，也就是说，它原本所在的位置会被后面的内容填补。
2. **环绕文本**：当一个元素浮动后，其周围的内联元素（如文本）会环绕在浮动元素的周围。
3. **容器塌陷**：如果一个容器（如<div>）内的所有子元素都浮动了，那么这个容器可能看起来“塌陷”了，因为它的高度会根据浮动元素的高度进行调整。
4. **清除浮动**：可以通过设置clear属性为left、right或both来阻止元素后面的元素浮动到它的旁边。
5. **浮动方向**：元素可以设置float: left;或float: right;来向左或向右浮动。

float：left；

float：right；

在现代的布局技术中，浮动已经不是首选的布局方式了，取而代之的是Flexbox和Grid等更加强大和灵活的布局系统。不过，浮动仍然是CSS布局中一个重要的概念。

浮动会造成父标签塌陷

解决办法：

1. 自己加一个div设置高度
2. 利用clear属性

#d4{

clear:left;/\*该标签的左边（地面和空中）不能有浮动的元素\*/

1. 通用的解决浮动带来的影响方法

在写htm1页面之前先提前写好处理浮动带来的影响的css代码

.clearfix:after{

content:“”；

display:block;

clear:both;

}

之后只要标签出现了塌陷的问题就给该塌陷的标签加一个clearfix属性即可

上述的解决方式是通用的到哪都一样并且名字就叫clearfix

然后引用：<div class="clearfix">ppp</div>

溢出



/\*overflow: visible;！\*默认就是可见溢出还是展示\*！\*

/\*overflow:hidden;！\*溢出部分直接隐藏\*！\*/

/\*overflow: scroll;！\*设置成左右上下滚动条的形式\*！\*/

/\*overflow: auto; \*设置成上下左右滚动条的形式\*！\*/\*/

定位

·静态

所有的标签默认都是静态的static，无法改变位置

·相定位（了解）

相对于标签原来的位置做移动relative，用了该属性就可以移动位置

position: relative;

left: 30px; 距离左边30px  
top: 30px; 距离顶部30px

·绝对定位（常用)

相对于已经定位过的父标签做移动absolute

position: absolute;

·固定定位(常用)

相对于浏览器窗口固定在某个位置fixed

position: fixed;

bottom: 30px; 距离顶部30px

验证是否脱离文档流

不脱离文档流（原来的位置不许其它元素占用）

1.相对定位

脱离文档流（原来的位置允许元素占用）

1.浮动

2.绝对定位

3.固定定位

z-index模态框

eg:百度登陆页面其实是三层结构

1.最底部是正常内容（z=0）最远的

2.黑色的透明区（z=99）中间层

3.白色的注册区域（z=100））离用户最近

z-index:55;







透明度opacity

#它不单单可以修改颜色的透明度还同时修改字体的透明度

rgba只能影响颜色

而opacity可以修改颜色和字体

opacity: 50%;

JavaScript

介绍

JavaScript是一种高级的、解释型的编程语言。它是一种多范式语言，支持面向对象、命令式、声明式和函数式编程风格。JavaScript最初由Netscape公司的Brendan Eich在1995年开发，目的是为网页添加动态功能和交互性。

存在大量的JavaScript框架和库，如React、Angular、Vue.js等，它们提供了构建复杂应用的工具和抽象。

注释

//单行注释

/\*

多行注释1

多行注释2

多行注释3

\*/

引入方式和语法

#两种引入方式

1.script标签内部直接书写js代码

2.script标签src属性引l入外部js代码

#js语法结构

js是以分号作为语句的结束

但是如果你不写分号，问题也不大也能够正常执行但是它就相当于没有结束符

变量

定义

在js中首次定义一个变量名的时候需要用关键字声明

1.关键字var

var name='jason'

2.es6推出的新语法

let name='jason'

如果你的编辑器支持的版本是5.1那么无法使用1et

如果是6.o则向下兼容varlet

var与1et的区别

var在for循环里面定义也会影响到全局

let在局部定义只会在局部生效

命名规范

变量只能是：数字 字母 下划线 $

命名规范

常量声明const

const PI 3.14

PI的值不可改变

数据类型

声明变量后，也能指向其它的数据类型

js/python是一门拥有动态类型

name='jason'

name =123

name =[1,2,3,4]

#name可以指向任意的数据类型

#但是有一些语言中，变量名之内指向一种后续不能更改

数值类型number

js里面没有整数浮点数之分，只有number一种数值类型

var a = 1；

typeof a；//查看a的数据类型

number

NaN特殊数值类型

特殊的NaN：数值类型表示的意思是“不是一个数字”NOTANUMBER

typeof NaN

'number'

类型转换

parseInt()

parseFloat()

parseInt('12312312')

12312312

parseFloat(\*11.11\*)

11.11

parseInt('11.11\*)

11

parseInt('123sdasdajs2312dasd')

123

parseInt('asdasdad123sdasdajs2312dasd')

NaN

字符串类型string

单引号‘’

var a = ‘阿斯蒂芬撒’

双引号“”

var a = “式打法是”

模板字符串esc下面的按键``，可以写多行

var a = `

sadfsadfas

sadfsadf

sadffs

`

也能格式化输出

a = `sdfsaf ${name},sfdsf ${score}` //${变量名}会替换成变量的内容

字符串拼接+

a = 'sadfasd' + 'asdf'

'sadfasdasdf'

1. 查找/搜索

* 变量名[索引号]

var a = ‘qwe’

console.log(a[0])

q

* **indexOf()**: 返回指定值在字符串中首次出现的索引，未找到返回 -1。

let str = "Hello World";

console.log(str.indexOf("o")); // 4

* **lastIndexOf()**: 返回指定值在字符串中最后一次出现的索引，未找到返回 -1。

console.log(str.lastIndexOf("o")); // 7

* **includes()**: 判断字符串是否包含指定值，返回 true 或 false。

console.log(str.includes("World")); // true

* **startsWith()**: 判断字符串是否以指定值开头，返回 true 或 false。

console.log(str.startsWith("Hello")); // true

* **endsWith()**: 判断字符串是否以指定值结尾，返回 true 或 false。

console.log(str.endsWith("World")); // true

2. 截取/提取

* **slice()**: 提取字符串的一部分，返回新字符串。

console.log(str.slice(0, 5)); // "Hello"

* **substring()**: 类似于 slice()，但不支持负数索引。

console.log(str.substring(6, 11)); // "World"

* **substr()**: 从指定位置开始提取指定长度的字符串（已弃用，推荐使用 slice）。

console.log(str.substr(6, 5)); // "World"

3. 替换/修改

* **replace()**: 替换字符串中的指定值（仅替换第一个匹配项）。

console.log(str.replace("World", "JS")); // "Hello JS"

* **replaceAll()**: 替换字符串中所有匹配的指定值。

let str2 = "Hello Hello";

console.log(str2.replaceAll("Hello", "Hi")); // "Hi Hi"

* **toLowerCase()**: 将字符串转换为小写。

console.log(str.toLowerCase()); // "hello world"

* **toUpperCase()**: 将字符串转换为大写。

console.log(str.toUpperCase()); // "HELLO WORLD"

* **trim()**: 去除字符串两端的空白字符。

let str3 = " Hello ";

console.log(str3.trim()); // "Hello"

* **trimStart() / trimEnd()**: 分别去除字符串开头或结尾的空白字符。

console.log(str3.trimStart()); // "Hello "

console.log(str3.trimEnd()); // " Hello"

4. 分割/连接

* **split()**: 将字符串按指定分隔符拆分为数组。

console.log(str.split(" ")); // ["Hello", "World"]

* **concat()**: 连接两个或多个字符串，返回新字符串。

console.log("Hello".concat(" ", "World")); // "Hello World"

5. 判断/测试

* **charAt()**: 返回指定索引位置的字符。

console.log(str.charAt(1)); // "e"

* **charCodeAt()**: 返回指定索引位置字符的 Unicode 编码。

console.log(str.charCodeAt(1)); // 101

* **match()**: 返回字符串中匹配正则表达式的结果。

console.log(str.match(/o/g)); // ["o", "o"]

* **search()**: 返回匹配正则表达式的第一个索引，未找到返回 -1。

console.log(str.search(/World/)); // 6

6. 其他

* **repeat()**: 将字符串重复指定次数，返回新字符串。

console.log("Hi".repeat(3)); // "HiHiHi"

* **padStart()**: 在字符串开头填充指定字符，直到达到指定长度。

console.log("5".padStart(3, "0")); // "005"

* **padEnd()**: 在字符串结尾填充指定字符，直到达到指定长度。

console.log("5".padEnd(3, "0")); // "500"

* **localeCompare()**: 比较两个字符串的排序顺序，返回一个数字。

console.log("a".localeCompare("b")); // -1

7. 模板字符串（ES6）

* **模板字符串**: 使用反引号（`）包裹字符串，支持多行文本和嵌入表达式。

let name = "World";

console.log(`Hello, ${name}!`); // "Hello, World!"

这些方法涵盖了字符串的常见操作，可以根据实际需求灵活使用。

布尔值

js中布尔值全小写

true fale

#布尔值是false的有哪些

0、null、undefined、NaN

null和undefined的区别

nul1

表示值为空一般都是指定或者清空一个变量时使用

name ='jason

name = nul1

undefined

表示声明了一个变量但是没有做初始化操作（没有给值）

函数没有指定返回值的时候返回的也是undefined

数组object

定义 var a = [1,2,3];

typeof(a)

'object'

数组类型为'object'，js中的数组类似python的列表。

变量名[索引号]

var a = [1,2,3]

console.log(a[0])

1

遍历

var a = [1,2,3];

for(let i in a){

console.log(a[i])

}

VM1003:3 1

VM1003:3 2

VM1003:3 3

1. 添加/删除元素

* **push()**: 在数组末尾添加一个或多个元素，返回新数组的长度。
* **pop()**: 删除并返回数组的最后一个元素。
* **unshift()**: 在数组开头添加一个或多个元素，返回新数组的长度。
* **shift()**: 删除并返回数组的第一个元素。
* **splice()**: 从指定位置删除或添加元素，返回被删除的元素数组。

let arr = [1, 2, 3];

arr.push(4); // [1, 2, 3, 4]

arr.pop(); // [1, 2, 3]

arr.unshift(0); // [0, 1, 2, 3]

arr.shift(); // [1, 2, 3]

arr.splice(1, 1, 'a'); // [1, 'a', 3] (删除索引1的元素，并插入'a')

2. 遍历数组

* **forEach()**: 对数组中的每个元素执行一次提供的函数。
* **map()**: 创建一个新数组，其结果是该数组中的每个元素调用一次提供的函数后的返回值。
* **filter()**: 创建一个新数组，包含通过所提供函数测试的所有元素。
* **reduce()**: 对数组中的每个元素执行一个提供的 reducer 函数，将其结果汇总为单个返回值。
* **reduceRight()**: 与 reduce() 类似，但从右到左执行。

let arr = [1, 2, 3];

arr.forEach(item => console.log(item)); // 依次输出: 1, 2, 3

let doubled = arr.map(item => item \* 2); // [2, 4, 6]

let evens = arr.filter(item => item % 2 === 0); // [2]

let sum = arr.reduce((acc, item) => acc + item, 0); // 6

3. 查找/筛选

* **find()**: 返回数组中满足提供的测试函数的第一个元素的值，否则返回 undefined。
* **findIndex()**: 返回数组中满足提供的测试函数的第一个元素的索引，否则返回 -1。
* **indexOf()**: 返回数组中指定元素第一次出现的索引，如果不存在则返回 -1。
* **lastIndexOf()**: 返回数组中指定元素最后一次出现的索引，如果不存在则返回 -1。
* **includes()**: 判断数组是否包含某个值，返回 true 或 false。

let arr = [1, 2, 3];

arr.forEach(item => console.log(item)); // 依次输出: 1, 2, 3

let doubled = arr.map(item => item \* 2); // [2, 4, 6]

let evens = arr.filter(item => item % 2 === 0); // [2]

let sum = arr.reduce((acc, item) => acc + item, 0); // 6

4. 排序/反转

* **sort()**: 对数组元素进行排序，默认按字符串Unicode码点排序。可以传入比较函数进行自定义排序。
* **reverse()**: 反转数组中元素的顺序。

let arr = [3, 1, 4, 2];

arr.sort((a, b) => a - b); // [1, 2, 3, 4]

arr.reverse(); // [4, 3, 2, 1]

5. 拼接/切片

* **concat()**: 合并两个或多个数组，返回一个新数组。
* **slice()**: 返回一个从开始到结束（不包括结束）选择的数组部分的新数组。
* **join()**: 将数组中的所有元素连接成一个字符串，可以指定分隔符。

let arr1 = [1, 2];

let arr2 = [3, 4];

let combined = arr1.concat(arr2); // [1, 2, 3, 4]

let sliced = combined.slice(1, 3); // [2, 3]

let joined = combined.join('-'); // "1-2-3-4"

6. 判断/测试

* **every()**: 测试数组中的所有元素是否都通过了提供的测试函数，返回 true 或 false。
* **some()**: 测试数组中是否至少有一个元素通过了提供的测试函数，返回 true 或 false。

let arr = [1, 2, 3];

let allPositive = arr.every(item => item > 0); // true

let someEven = arr.some(item => item % 2 === 0); // true

7. 转换

* **toString()**: 将数组转换为字符串，元素之间用逗号分隔。
* **toLocaleString()**: 将数组转换为本地化字符串，元素之间用逗号分隔。
* **Array.from()**: 从类数组对象或可迭代对象创建一个新的数组实例。
* **Array.of()**: 创建一个具有可变数量参数的新数组实例，不考虑参数的数量或类型。

let arr = [1, 2, 3];

let str = arr.toString(); // "1,2,3"

let localeStr = arr.toLocaleString(); // "1,2,3"

let newArr = Array.from('123', Number); // [1, 2, 3]

let arrOf = Array.of(1, 2, 3); // [1, 2, 3]

8. 其他

* **fill()**: 用一个固定值填充数组中从起始索引到终止索引内的全部元素。
* **copyWithin()**: 浅复制数组的一部分到同一数组中的另一个位置，并返回它，而不修改其大小。
* **flat()**: 将嵌套的数组“拉平”，变成一维数组。可以指定拉平的层数。
* **flatMap()**: 首先使用映射函数映射每个元素，然后将结果压缩成一个新数组。

let arr = [1, 2, 3];

arr.fill(0, 1, 3); // [1, 0, 0]

arr.copyWithin(0, 1, 2); // [0, 0, 0] (将索引1到2的元素复制到索引0)

let nested = [1, [2, [3]]];

let flat = nested.flat(2); // [1, 2, 3]

let flatMapped = arr.flatMap(item => [item, item \* 2]); // [1, 2, 0, 0, 0, 0]

自定义对象object

创建自定义对象

var a={‘name‘：’cj‘，’age‘：18}

var a = new object（） //创建一个空对象

对象赋值遍历；

var c = {}

undefined

c.name='cj' //可以用.的形式赋值

'cj'

c.age=18

18

for(let i in c){

console.log(i, c[i])

}

VM1301:2 name cj

VM1301:2 age 18

取值

c.name

'cj'

date对象

let c1 = new Date //获取当前时间

undefined

c1

Tue Dec 31 2024 01:18:01 GMT+0800 (中国标准时间)

c1.toLocaleString() //转换格式

'2024/12/31 01:18:01'

也可以自定义

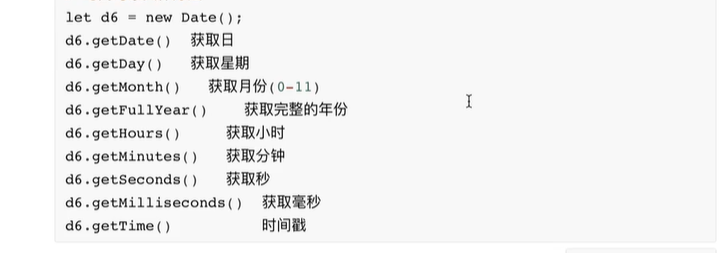
let c2 = new Date('2000/11/11 11:11:11')

undefined

c2.toLocaleString()

'2000/11/11 11:11:11'

时间对象方法



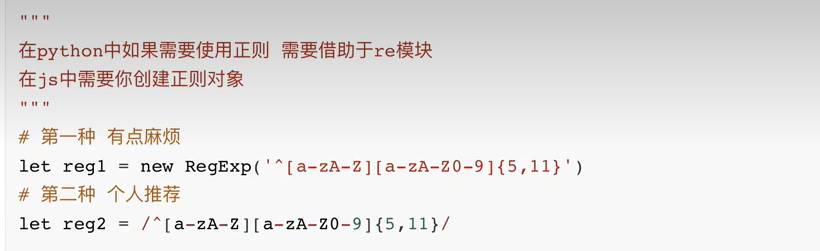
json对象

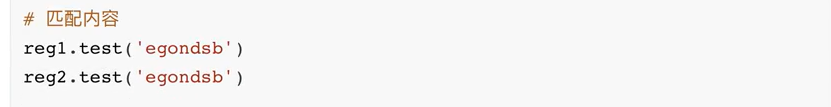
序列化：把该对象变为json格式的字符串

反序列化：json格式的字符串反变为原来的对象



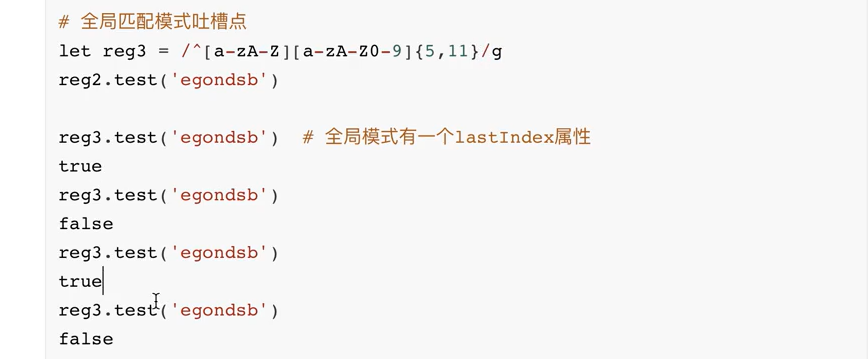
RegExp对象

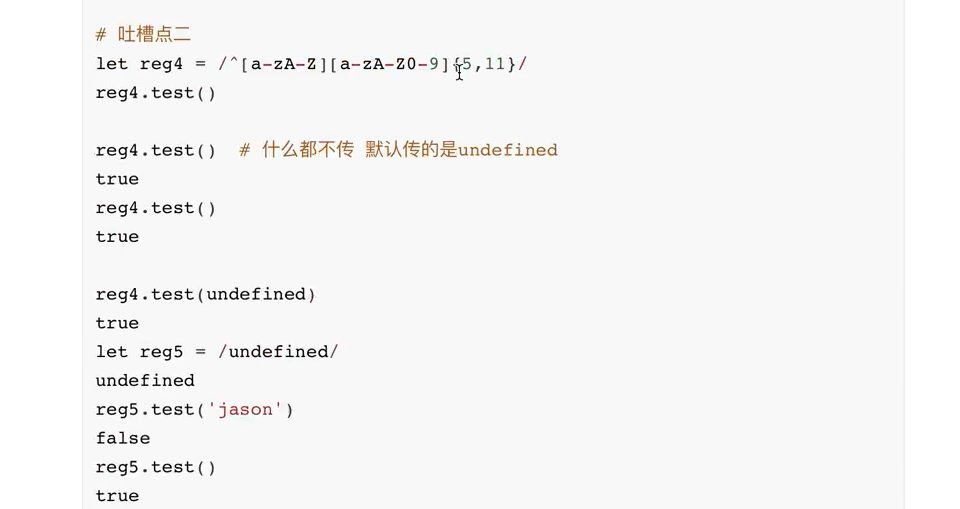












运算符

算术运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 示例 |
| + | 加法 | 1 + 2 → 3 |
| - | 减法 | 5 - 3 → 2 |
| \* | 乘法 | 2 \* 3 → 6 |
| / | 除法 | 6 / 2 → 3 |
| % | 取余（取模） | 7 % 3 → 1 |
| \*\* | 指数（幂） | 2 \*\* 3 → 8 |

赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 示例 |
| = | 赋值 | let a = 5 |
| += | 加后赋值 | a += 2 → 7 |
| -= | 减后赋值 | a -= 3 → 4 |
| \*= | 乘后赋值 | a \*= 2 → 8 |
| /= | 除后赋值 | a /= 4 → 2 |
| %= | 取余后赋值 | a %= 3 → 2 |

比较运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 示例 |
| == | 等于（值相等） | 1 == '1' → true |
| === | 严格等于（值和类型都相等） | 1 === '1' → false |
| != | 不等于（值不相等） | 1 != '2' → true |
| !== | 严格不等于（值或类型不相等） | 1 !== '1' → true |
| > | 大于 | 3 > 2 → true |
| < | 小于 | 3 < 2 → false |
| >= | 大于等于 | 3 >= 3 → true |
| <= | 小于等于 | 3 <= 2 → false |

逻辑运算符

&& || ！ 与或非

与shell命令相似

5 && ‘5’

‘5’

!5 && 4

false

1 || 5

1

0 || 5

5

三元运算符

? :三元运算符let result = (5 > 3) ? 'Yes' : 'No'; → 'Yes'

If和switch流程控制

if（a>0）{

程序

}

else{

程序

}

if(a>0){

程序

}

else if（a>-5）{

程序

}

else{

程序

}

switch num{

case 数字：

程序；

程序；

break：

case 数字：

程序；

程序；

break：

case 数字：

程序；

程序；

break：

default：

程序；

程序；

}

for循环

for（let i=0; i<10; i++）{

程序；

}

while循环

while（a<10）{

程序；

}

函数

function 函数名（形参1，形参2，形参3...）{函数体代码}

#无参函数

function funcl（）{

console.log('hello world')

func1()#调用加括调用跟python是一样的

#有参函数

function func2（a,b){

console.log(a,b)

}

func2(1,2)

1，2

func2（1,2,3,4,5,6,7,8,9)）#多了没关系只要对应的数据

VM3610:212

undefined

func2(1)#少了也没关系

VM3610:21undefined

#关键字arguments

function func2（a,b）{

console.log(arguments) #能够获取到函数接受到的所有的参数

console.log(a,b)

}

function func2（a,b){

if（arguments.length<2）{

console.1og（传少了）

}else if （arguments.length>2){

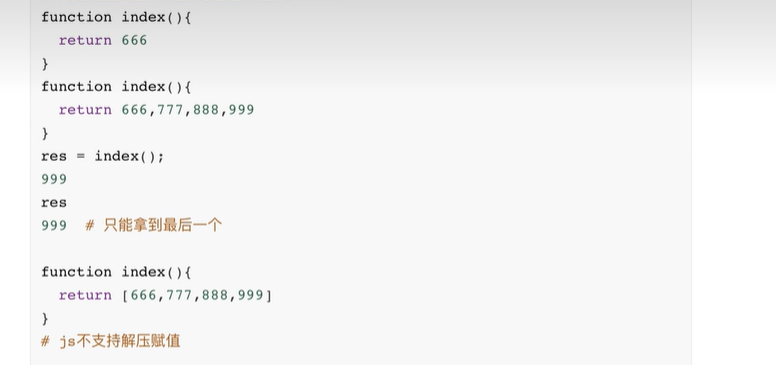
console.log（‘传多了）

}else{

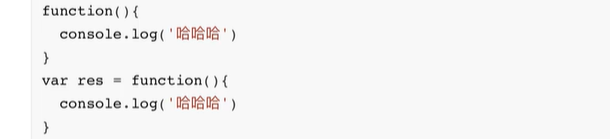
console.log（‘正常执行'）}

}

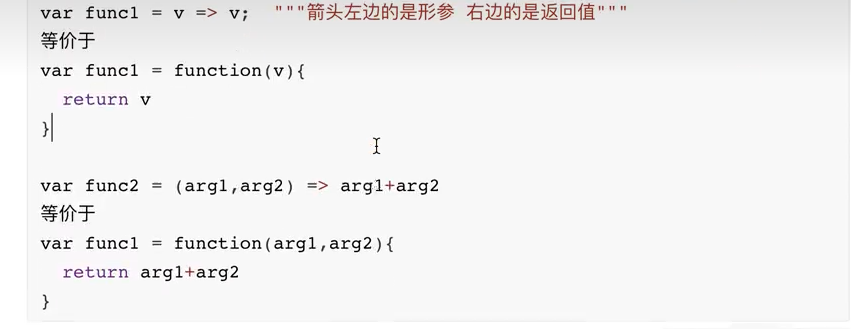
#函数的返回值使用的也是关键字return



匿名函数，没有名字



箭头函数



全局变量与局部变量

用法与python相同

BOM

介绍

在JavaScript中，BOM指的是浏览器对象模型（Browser Object Model）。BOM提供了一组对象，允许JavaScript与浏览器进行交互。

1. window 对象

window 是 BOM 的核心对象，代表浏览器窗口。

**属性**

* window.innerWidth / window.innerHeight：获取视口的宽度和高度。

console.log(window.innerWidth, window.innerHeight); // 视口宽度和高度

* window.outerWidth / window.outerHeight：获取浏览器窗口的宽度和高度。

console.log(window.outerWidth, window.outerHeight); // 浏览器窗口宽度和高度

**方法**

* window.open() / window.close()：打开或关闭新窗口。

window.open("https://www.example.com"， ‘’，‘height=400px,width=400px,top=40

0px,1eft=400px’); // 打开新窗口

window.close(); // 关闭当前窗口

* window.alert() / window.confirm() / window.prompt()：显示弹窗。

window.alert("Hello!"); // 显示警告框

let isConfirmed = window.confirm("Are you sure?"); // 显示确认框

let userInput = window.prompt("Enter your name:"); // 显示输入框

* window.setTimeout() / window.setInterval()：设置定时器。

setTimeout(() => console.log("Timeout!"), 1000); // 1秒后执行

setInterval(() => console.log("Interval!"), 1000); // 每隔1秒执行

2. navigator 对象

navigator 提供有关浏览器的信息。

**属性**

* navigator.userAgent：返回浏览器的用户代理字符串。

console.log(navigator.userAgent); // 浏览器信息

* navigator.platform：返回操作系统平台。

console.log(navigator.platform); // 操作系统

* navigator.language：返回浏览器的首选语言。

console.log(navigator.language); // 浏览器语言

**方法**

* navigator.geolocation.getCurrentPosition()：获取用户的地理位置。

navigator.geolocation.getCurrentPosition((position) => {

console.log(position.coords.latitude, position.coords.longitude);

});

3. location 对象

location 用于操作当前页面的 URL。

**属性**

* location.href：获取或设置完整的 URL。

console.log(location.href); // 当前页面 URL

location.href = "https://www.example.com"; // 跳转到新页面

* location.search：获取查询字符串。

console.log(location.search); // 查询字符串（如 "?name=John"）

* location.hash：获取 URL 的哈希部分。

console.log(location.hash); // 哈希部分（如 "#section1"）

**方法**

* location.reload()：重新加载页面。

location.reload(); // 刷新页面

* location.replace()：替换当前页面。

location.replace("https://www.example.com"); // 替换当前页面

4. history 对象

history 用于操作浏览器的历史记录。

**属性**

* history.length：获取历史记录中的页面数量。

console.log(history.length); // 历史记录数量

**方法**

* history.back()：返回上一页。

history.back(); // 返回上一页

* history.forward()：前进到下一页。

history.forward(); // 前进到下一页

* history.go(n)：跳转到历史记录中的第 n 个页面。

history.go(-2); // 返回前两页

5. screen 对象

screen 提供有关用户屏幕的信息。

**属性**

* screen.width / screen.height：获取屏幕的宽度和高度。

console.log(screen.width, screen.height); // 屏幕宽度和高度

* screen.availWidth / screen.availHeight：获取屏幕的可用宽度和高度。

console.log(screen.availWidth, screen.availHeight); // 可用宽度和高度

* screen.colorDepth：获取屏幕的颜色深度。

console.log(screen.colorDepth); // 颜色深度

6. document 对象

document 是 BOM 的一部分，也是 DOM 的核心对象。

**属性**

* document.title：获取或设置页面标题。

console.log(document.title); // 页面标题

document.title = "New Title"; // 设置页面标题

* document.cookie：操作页面的 cookie。

document.cookie = "name=John"; // 设置 cookie

console.log(document.cookie); // 获取 cookie

**方法**

* document.write()：向页面写入内容。

document.write("<h1>Hello!</h1>"); // 向页面写入内容

7. localStorage 和 sessionStorage

用于在浏览器中存储数据。

**方法**

* localStorage.setItem() / localStorage.getItem()：存储和获取数据（永久存储）。

localStorage.setItem("name", "John"); // 存储数据

console.log(localStorage.getItem("name")); // 获取数据

* sessionStorage.setItem() / sessionStorage.getItem()：存储和获取数据（会话级别存储）。

sessionStorage.setItem("name", "John"); // 存储数据

console.log(sessionStorage.getItem("name")); // 获取数据

**总结**

BOM 提供了丰富的对象和方法，用于操作浏览器窗口、获取浏览器信息、管理 URL 和历史记录等。通过结合简单示例，可以更好地理解和使用这些功能。

DOM

介绍

在 JavaScript 中，DOM（Document Object Model，文档对象模型）是一种编程接口，它将 HTML 和 XML 文档呈现为树状结构，允许你通过 JavaScript 对文档的结构、样式和内容进行动态访问和修改。

1. 查询获取元素

用于选择文档中的元素。

**方法**

* document.getElementById()：通过 id 获取元素。

let element = document.getElementById("myId"); //获取的是单个元素对象

* document.getElementsByClassName()：通过 class 获取元素集合。

let elements = document.getElementsByClassName("myClass"); //获取的是元素对象数组，取数据许加索引，例如elements[0]

* document.getElementsByTagName()：通过标签名获取元素集合。

let elements = document.getElementsByTagName("div");

* document.querySelector()：通过 CSS 选择器获取第一个匹配的元素。

let element = document.querySelector(".myClass");

* document.querySelectorAll()：通过 CSS 选择器获取所有匹配的元素集合。

let elements = document.querySelectorAll("div.myClass");

2. 修改元素内容

用于修改元素的文本或 HTML 内容。

**属性**

* element.textContent：获取或设置元素的文本内容。

element.textContent = "Hello, World!";

* element.innerHTML：获取或设置元素的 HTML 内容。

element.innerHTML = "<strong>Hello</strong>, World!";

3. 修改获得元素属性

用于操作元素的属性。

**方法**

* element.getAttribute()：获取元素的属性值。

let value = element.getAttribute("href");

如果是<input type="text">类似的输入标签，可用.value来获取实时的输入内容。

如果是<input type="file" multiple>文件输入标签，可用.files[0]来获取文件对象。

* element.setAttribute()：设置元素的属性值,没有则创建。

element.setAttribute("href", "https://www.example.com");

* element.removeAttribute()：移除元素的属性。

element.removeAttribute("href");

**属性**

* element.id：获取或设置元素的 id。

element.id = "newId";

* element.className：获取或设置元素的 class。

element.className = "newClass";

* element.classList：操作元素的 class 列表。

element.classList.add("newClass"); // 添加 class

element.classList.remove("oldClass"); // 移除 class

element.classList.toggle("active"); // 切换 class，有该属性就删除，无就添加

4. 修改元素样式

用于操作元素的样式。

**属性**

* element.style：直接修改元素的样式。

element.style.color = "red";

element.style.backgroundColor = "yellow";

* window.getComputedStyle()：获取元素的计算样式。

let styles = window.getComputedStyle(element);

console.log(styles.color);

5. 创建和添加元素

用于动态创建和添加元素。

**方法**

* document.createElement()：创建新元素。

let newElement = document.createElement("div");

* element.appendChild()：将元素添加到父元素的末尾。

parentElement.appendChild(newElement); //只能是元素对象，不能是列表元素对象，列表元素对象这样添加parentElement[0].appendChild(newElement);这只是临时加入，刷新就不存在了。

* element.insertBefore()：将元素插入到指定元素之前。

parentElement.insertBefore(newElement, existingElement);

* element.cloneNode()：克隆元素。

let clonedElement = element.cloneNode(true); // 深度克隆

6. 删除和替换元素

用于删除或替换元素。

**方法**

* element.remove()：删除元素。

element.remove();

* parentElement.removeChild()：从父元素中删除子元素。

parentElement.removeChild(childElement);

* parentElement.replaceChild()：替换子元素。

parentElement.replaceChild(newElement, oldElement);

7. 事件操作

用于绑定和触发事件。

绑定事件的2种触发方式





事件列表

|  |  |
| --- | --- |
| **事件** | **说明** |
| onclick | 当用户点击某个对象时调用的事件句柄。 |
| ondblclick | 当用户双击某个对象时调用的事件句柄。 |
| onfocus | 元素获得焦点。 |
| onblur | 元素失去焦点。 |
| onchange | 域的内容被改变。 |
| onkeydown | 某个键盘按键被按下。 |
| onkeypress | 某个键盘按键被按下并松开。 |
| onkeyup | 某个键盘按键被松开。 |
| onload | 一张页面或一幅图像完成加载。 |
| onmousedown | 鼠标按钮被按下。 |
| onmousemove | 鼠标被移动。 |
| onmouseout | 鼠标从某元素移开。 |
| onmouseover | 鼠标移到某元素之上。 |
| onselect | 在文本框中的文本被选中时发生。 |
| onsubmit | 确认按钮被点击，使用的对象是form。 |

**方法**

* element.addEventListener()：绑定事件监听器。

element.addEventListener("click", () => {

console.log("Clicked!");

});

* element.removeEventListener()：移除事件监听器。

element.removeEventListener("click", handler);

* element.dispatchEvent()：触发事件。

let event = new Event("click");

element.dispatchEvent(event);

8. 遍历 DOM

用于遍历元素树。

**属性**

* element.parentElement：获取父元素。

let parent = element.parentElement;

* element.children：获取子元素集合。

let children = element.children;

* element.firstElementChild / element.lastElementChild：获取第一个或最后一个子元素。

let firstChild = element.firstElementChild;

let lastChild = element.lastElementChild;

* element.previousElementSibling / element.nextElementSibling：获取前一个或后一个兄弟元素。

let previousSibling = element.previousElementSibling;

let nextSibling = element.nextElementSibling;

9. 表单操作

用于操作表单元素。

**属性**

* element.value：获取或设置表单元素的值。

let inputValue = inputElement.value;

inputElement.value = "New Value";

* element.checked：获取或设置复选框或单选按钮的状态。

let isChecked = checkboxElement.checked;

checkboxElement.checked = true;

**方法**

* element.focus()：使表单元素获得焦点。

inputElement.focus();

* element.blur()：使表单元素失去焦点。

inputElement.blur();

10. 操作类名和样式

用于动态修改元素的类名和样式。

**方法**

* element.classList.add()：添加类名。

element.classList.add("newClass");

* element.classList.remove()：移除类名。

element.classList.remove("oldClass");

* element.classList.toggle()：切换类名。

element.classList.toggle("active");

**总结**

DOM 操作是前端开发的核心技能之一，涵盖了获取元素、修改内容、操作属性、处理事件、动态创建和删除元素等功能。通过熟练掌握这些操作，可以实现丰富的交互效果和动态页面更新。