Day01笔记

|  |
| --- |
| HTML笔记：  1.前端环境搭建：安装VSCode，安装开发插件  2.HTML与CSS：HTML和CSS之间的关系？  HTML是⽹⻚内容的载体。内容就是⽹⻚制作者放在⻚⾯上想要让⽤户  浏览的信息，可以包含⽂字、图⽚、视频等。  CSS样式是表现。就像⽹⻚的外⾐。⽐如，标题字体、颜⾊变化，或  为标题加⼊背景图⽚、边框等。所有这些⽤来改变内容外观的东⻄称  之为表现。  <!--⽂档头信息-->  <!--html5标准⽹⻚声明-->  <!--不加这⼀⾏，就表示⻚⾯采⽤浏览器本身的解析标准，这样会造成⻚⾯在不同  的浏览器（IE、⽕狐等）可能出现不同的显示效果。-->  <!--h5的⽂档声明，声明当前的⽹⻚是按照HTML5标准编写的编写⽹⻚时⼀定要将  h5的⽂档声明写在⽹⻚的最上边。  如果不写⽂档声明，则会导致有些浏览器会进⼊⼀个怪异模式，进⼊怪异模式以  后，浏览器解析⻚⾯会导致⻚⾯⽆法正常显示，所以为了避免进⼊该模式，⼀定要写  ⽂档声明  -->  <!DOCTYPEhtml>  <!--属性lang是单词language的缩写，意思是语⾔，”en”代表英语，”zh-CN”  代表中⽂-->  <!--  html根标签，⼀个⻚⾯中有且只有⼀个根标签，⽹⻚中的所有内容都应该写在  html根标签中  -->  <!--只有两个⼦标签head:浏览器设置body：可视区展示-->  <htmllang="en">  <!--head标签，该标签中的内容，不会在⽹⻚中直接显示，它⽤来帮助浏览器解析  ⻚⾯的-->  <head>  <!--  meta标签⽤来设置⽹⻚的⼀些元数据，⽐如⽹⻚的字符集，关键字、简介  meta是⼀个⾃结束标签，编写⼀个⾃结束标签时，可以在开始标签中添加  ⼀个/  -->  <!--  字符编码，浏览器会根据字符编码进⾏解析  常⻅的字符编码有：gb2312、gbk、unicode、utf-8。  -->  <metacharset="UTF-8">  <!--  viewport设备的屏幕  width=device-widthwidth属性控制设备的宽度。假设您的⽹站将被  带有不同屏幕分辨率的设备浏览，那么将它设置为device-width可以  确保它能正确呈现在不同设备上。  initial-scale=1.0确保⽹⻚加载时，以1:1的⽐例呈现，不会有  任何的缩放。  -->  <metaname="viewport"content="width=device-width,initialscale=1.0">  <!--浏览器选项卡上的名字-->  <title>HelloHTML</title>  </head>  <!--  body标签⽤来设置⽹⻚的主体内容，⽹⻚中所有可⻅的内容，都应该在body中编  写  -->  <body>  <!--在body⾥的内容都是显示在浏览器的视图区的-->  <!--  在这个结构中，可以来编写HTML的注释注释中的内容，不会在⻚⾯中显示，  但是可以在源码中查看我们可以通过编写注释来对代码进⾏描述，从⽽帮助  其他的开发⼈员⼯作,⼀定要养成良好的编写注释的习惯，但是注释⼀定要简  单明了  -->  </body>  </html>  日期2024.6.25 |

Day02笔记

|  |
| --- |
| 1.功能元素简介  1.1列表标签  列表标签的作用是为一组数据添加列表语义，让搜索引擎和浏览器知道这是一组相关的数据。列表标签分为三类：  -无序列表（unorderedlist，最多）  -有序列表（orderedlist，最少）  -定义列表（definitionlist）  1.1.1无序列表  无序列表用于为一组数据添加列表语义，且这些数据没有先后顺序。例如，中国的城市列表。无序列表的格式如下：  <ul>  <li>条目内容</li>  </ul>  无序列表的应用场景包括新闻列表、商品列表和导航条。  1.1.2有序列表  有序列表用于为一组数据添加列表语义，且这些数据有先后顺序。例如，排行榜。格式如下：  <ol>  <li>条目内容</li>  </ol>  有序列表的样式可以通过type属性设置，如a、A、i、I等。  1.1.3定义列表  定义列表用于为一组数据添加列表语义，使用dt标签定义列表标题，dd标签添加描述信息。格式如下：  <dl>  <dt>标题</dt>  <dd>描述</dd>  </dl>  定义列表的应用场景包括网站尾部信息和图文混排。  1.2表格标签  表格标签用于为一组数据添加表格语义，是一种清晰的数据展示形式。表格标签的基本格式如下：  <table>  <tr>  <td>姓名</td>  <td>年龄</td>  <td>身高</td>  </tr>  <tr>  <td>张三</td>  <td>19</td>  <td>1.78</td>  </tr>  </table>  表格的完整结构包括标题、表头、主体和页尾信息。表格的应用场景多用于数据展示。  1.3form表单  表单用于收集用户信息，包含多种表单元素，如文本输入框、密码输入框、单选框、多选框、提交按钮等。表单的基本格式如下：  <formaction="服务器地址">  <inputtype="text"name="account"placeholder="请输入用户名">  <inputtype="password"name="password"placeholder="请输入密码">  <inputtype="submit"value="提交">  </form>  表单元素有多种类型，input标签的type属性决定其功能和外观，如text、password、radio、checkbox、submit等。  注意点  -列表标签中的ul和li是一个整体，通常一起使用，不单独出现。  -表格标签中的table、tr和td是一个组合标签，通常一起出现。  -表单元素需要写在form标签内，form标签需要指定action属性来提交数据。  其它用法和注意事项  -列表、表格和表单标签的样式和外观可以通过CSS进行控制，而不是依赖标签自带的属性。  -表格的宽度和高度可以通过CSS进行手动指定，align和valign属性用于水平和垂直对齐。  -表单中的输入框可以设置默认值，单选框和多选框可以通过checked属性设置默认选中状态。  -提交按钮需要与form标签的action属性配合使用，提交数据到指定的服务器地址。  以上内容总结了HTML中的功能元素，包括列表、表格和表单的基本用法和注意事项。  日期2024.6.26 |

Day03笔记

|  |
| --- |
| 什么是CSS  CSS(CascadingStyleSheets)层叠样式表，是一种用于修饰文档（可以是标记语言HTML，也可以是XML或者SVN）的语言。它能使文档以更优雅的形式呈现给用户。  HTML与CSS的关系：  HTML：网页内容的载体，包括文字、图片、视频等信息。  CSS：表现网页的外观，例如标题字体、颜色、背景图片、边框等。  CSS语法  CSS声明（CSSdeclarations）：  CSS的核心功能是为特定的属性设定特定的值，属性与值通过“:”分隔。  CSS声明块（CSSdeclarationblocks）：  将多个CSS声明写在一起，每个声明通过“;”分隔，用“{}”将多个CSS声明括起来，构成一个CSS声明块。  CSS规则集（CSSrulesets）：  选择器（Selector）：HTML元素的名称，用于选择需要添加样式的元素。  声明（Declaration）：指定添加样式元素的属性，例如color:red;。  属性（Properties）：改变HTML元素样式的途径。  属性的值（Propertyvalue）：从指定属性的众多外观中选择一个值。  CSS可读性  空白（Whitespace）：实际空格、制表符和新行，可以添加空白使样式表更加可读。  注释（Comments）：用/\*注释内容\*/表示。  速记写法（Shorthand）：如font、background、padding、border、margin等属性可以速记写法，节省时间，使代码整洁。  CSS的引入方式  内联样式（行内样式）：  优点：优先级较高。  缺点：样式与结构冗余，不方便维护。  内部样式表：  优点：样式与结构分离。  缺点：样式的复用率较低。  外部样式表：  优点：样式与结构分离，解耦；样式的复用率高，共用。  使用方式：通过link标签引入外部CSS文件或使用@import方式。  CSS选择器  标签选择器：  根据指定的标签名称找到所有该名称的标签并设置属性。  ID选择器：  根据指定的ID名称找到对应的标签并设置属性，ID名称在同一个页面中不能重复。  类选择器：  根据指定的类名称找到对应的标签并设置属性，类名可以重复使用。  日期2024.6.27 |

Day04笔记

|  |
| --- |
| 后代选择器：  找到指定标签的所有特定的后代标签并设置属性，通过空格分隔。  <style>  div p {  color: red; /\* 设置div中的所有p标签的文字颜色为红色 \*/  }  </style>  <div>  <p>这是一个段落。</p>  <span>  <p>这是另一个段落。</p>  </span>  </div>  子元素选择器：  找到指定标签中所有特定的直接子元素并设置属性，通过>分隔。  <style>  div > p {  color: blue; /\* 设置div中的直接子元素p标签的文字颜色为蓝色 \*/  }  </style>  <div>  <p>这是一个段落。</p>  <span>  <p>这是另一个段落。</p>  </span>  </div>  交集选择器：  给所有选择器选中的标签中相交的那部分标签设置属性。  <style>  .class1.class2 {  font-weight: bold; /\* 设置同时具有class1和class2类的元素的字体加粗 \*/  }  </style>  <p class="class1 class2">这是一个段落。</p>  <p class="class1">这是另一个段落。</p>  <p class="class2">这是再一个段落。</p>  并集选择器：  给所有选择器选中的标签设置属性，通过,分隔。  <style>  h1, h2, p {  margin-bottom: 20px; /\* 设置h1、h2和p标签的下边距为20px \*/  }  </style>  <h1>这是标题1</h1>  <h2>这是标题2</h2>  <p>这是一个段落。</p>  兄弟选择器：  相邻兄弟选择器：给指定选择器后面紧跟的那个选择器选中的标签设置属性，通过+分隔。  <style>  h1 + p {  color: green; /\* 设置紧跟在h1后的p标签的文字颜色为绿色 \*/  }  </style>  <h1>这是标题</h1>  <p>这是一个段落。</p>  <p>这是另一个段落。</p>  通用兄弟选择器：给指定选择器后面的所有选择器选中的所有标签设置属性，通过~分隔。  <style>  h1 ~ p {  color: orange; /\* 设置h1之后的所有p标签的文字颜色为橙色 \*/  }  </style>  <h1>这是标题</h1>  <p>这是一个段落。</p>  <p>这是另一个段落。</p>  伪类选择器：  伪类以:开头，用于选中特定状态的元素。  <style>  a:hover {  color: red; /\* 设置鼠标悬停在a标签上的文字颜色为红色 \*/  }  p:first-child {  font-style: italic; /\* 设置第一个子元素p标签的字体为斜体 \*/  }  p:nth-child(2) {  font-weight: bold; /\* 设置第二个子元素p标签的字体加粗 \*/  }  </style>  <a href="#">这是一个链接</a>  <p>这是第一个段落。</p>  <p>这是第二个段落。</p>  <p>这是第三个段落。</p>  日期2024.6.28 |

第二周

Day05笔记

|  |
| --- |
| CSS文本属性  1.颜色属性：使用color属性来修改文本颜色。格式为color:值;，常见的取值是英文单词表示的颜色，也可以用RGB或十六进制代码表示。  -示例：color:red;  -示例：color:rgb(255,,);  -示例：color:#FF000;  2.字体样式属性：  a)font-style用于设置斜体文本。常见取值有normal（正常）和italic（斜体）。  -示例：font-style:italic;  b)font-weight用于设置字体粗细程度。常见取值有normal（正常），bold（粗体），bolder（更粗），lighter（细线）。  -示例：font-weight:bold;  3.字号属性：  使用font-size属性指定文字大小。单位为像素(px)。  -示例：font-size:16px;  4.字体族属性：  使用font-family指定特殊字体，浏览器会选择可访问到的字体来显示文本。  -示例："Arial","Helvetica",sans-serif;  5.文本修饰：  a)text-decoration用于设置或取消文本修饰效果，如下划线、删除线等。  取值有underline(下划线)，line-through(删除线)，overline(上划线)，none(无修饰)等。  b)text-transform用于改变字母大小写形式，如全大写、全小写等。  6.文本对齐方式：  使用text-align属性指定文字排列方式。可选值有left、right和center。  7.文本缩进：  使用text-indent属性控制段落首行缩进的距离。单位通常使用em进行度量。  8.字间距和行高：  可以通过line-height来设置行高，通过letter-spacing来调整字符间距。  9.文本阴影：  使用text-shadow属性来设置文本阴影效果。格式为text-shadow:h-shadowv-shadowblurcolor;，可以设置水平位置、垂直位置、模糊程度和颜色。  其他样式包括列表样式和一些杂项样式：  10.列表样式：  可以使用list-style-type属性来设置列表项标志的类型，如圆点、方框等。  -示例：list-style-type:circle;  11.显示方式切换：  使用display属性可以切换元素的显示方式，常见取值有inline（内联显示）、block（块级显示）和inline-block（行内块级显示）。  12.其他样式：  还有一些其他常用的CSS文本属性，包括visibility(可见性)、opacity(透明度)、overflow(溢出处理)和cursor(鼠标指针样式)等。  以上是CSS中常用的文本属性。了解这些属性可以帮助我们更好地控制文本在页面中的展示效果。  日期2024.7.1 |

Day06笔记

|  |
| --- |
| CSS盒子模型和背景属性  1.盒子模型： -CSS盒子模型是用来描述HTML元素的布局方式的，包括外边距（margin）、边框（border）、内边距（padding）和内容区域（content）。 -通过设置宽度（width）和高度（height），可以控制内容区域的尺寸。 -外边距用于控制元素与其他元素之间的距离，可以设置上下左右四个方向的外边距值。 -边框属性用于定义元素周围的线条样式、宽度和颜色。 -内边距属性用于在元素内容周围创建空间。  2.背景属性： -背景颜色：使用background-color属性来设置标签的背景颜色，可以使用具体单词、RGB或十六进制值表示。 -背景图像：使用background-image属性来添加背景图片，并通过指定图片路径或URL来引入图像资源。 -平铺方式：通过background-repeat属性来控制背景图像在盒子中如何平铺。可选值有repeat、no-repeat、repeat-x和repeat-y等选项。 -定位方式：通过background-position属性来设置背景图像在盒子中的位置。可以使用关键字或具体像素值进行定位。  3.盒模型注意事项： -外部盒子之间的垂直方向上的外边距会发生合并现象，取两者中最大的外边距值作为最终结果。水平方向上不会发生合并现象。 -可以利用margin:auto;的方式将内部盒子在外部容器中水平居中对齐。  4.背景图像与插入图像的区别： -背景图像仅用于装饰，不会占据元素的位置，而插入图像会占据位置。 -背景图像具有定位属性，可以方便地控制图像的位置和平铺方式，而插入图像没有定位属性，控制位置不太便捷。 -插入图像在语义上比背景图像更强，所以如果希望图片被搜索引擎收录推荐使用插入图片。  日期2024.7.2 |

Day07笔记

|  |
| --- |
| CSS布局 1.网页的布局方式是指浏览器如何对网页中的元素进行排版。 2.标准流（文档流/普通流）是网页布局中最基本的排版方式，处在网页的最底层。所有创建的元素默认都在标准流中。 3.块级元素在标准流中会独占一行，并且从上到下排列。默认情况下，块级元素的宽度是父元素的100%，高度被内容撑开。 4.内联元素在标准流中只占据自身大小，在一行上从左到右排列。如果一行放不下所有内联元素，则换到下一行继续从左到右排列。内联元素的宽度和高度都被内容撑开。 5.浏览器默认使用标准流进行排版。 6.CSS将元素分为三类：块级元素、行内元素和行内块级元素。 7.标准流有两种排版方式：垂直排版和水平排版。如果一个块级元素，则会垂直排版；如果一个行内元素或行内块级元素，则会水平排版。 8.通过使用float属性可以使块级元素浮动，以实现水平排列。可选值有none（默认值）、left和right。当设置了某个值后，该元素就脱离了标准流，其下方的元素除非清除浮动，否则会受到影响位置 9.浮动可以实现两栏布局或三栏布局等复杂布局效果 10.当为父容器设置了浮动以后，父容器的高度会发生塌陷。解决这个问题的方法有以下几种： -设置父容器的高度写死，但这样会导致父容器无法自动适应子元素的高度，不推荐使用。 -在父容器最后添加一个空白的div，并对其清除浮动。这样可以通过空白div来撑开父容器的高度，并且没有副作用。 -使用伪类:after可以向元素最后添加一个空白块元素，并对其清除浮动。这样做和添加一个div原理相同，但不会在页面中添加多余的结构。 11.BFC（块级格式化上下文）是CSS中一种独立布局环境，在BFC内部进行布局时与外部不互相影响。可以通过以下方式创建BFC： -根元素 -设置float属性为非none -position属性为absolute或fixed -display属性为inline-block、table-cell、table-caption、flex或inline-flex -overflow属性不为visible 12.BFC具有以下特点和作用： -解决浮动元素导致父元素高度塌陷问题。 -提供了一个隔离独立环境，内部元素布局与外部互不影响。 -可以防止浮动元素重叠。 13.两列布局可以通过给固定栏设置固定宽度，给自适应栏开启BFC来实现。 14.三列布局可以通过给两边栏设置固定宽度，中间栏设置margin来实现。 CSS布局涉及到标准流、浮动流和BFC等概念。学习如何利用标准流和float属性进行基本排版，并了解如何使用BFC解决浮动元素导致的高度塌陷问题。掌握这些布局技巧可以帮助我们实现各种复杂的网页布局。  日期2024.7.3 |

Day08笔记

|  |
| --- |
| 1.定位流分类  1.1静态定位（StaticPositioning）  定义:所有元素的默认定位方式，将一个元素定位在默认文档流中。  使用方法:position:static;  1.2相对定位（RelativePositioning）  定义:相对定位是相对于元素在标准流中的原始位置来移动。  使用方法:position:relative;，使用top、right、bottom、left来控制位置。  **注意点:**  不脱离标准流，会继续占用标准流中的空间。  同一方向上只能使用一个定位属性。  影响标准流的布局，如margin和padding属性。  应用场景:用于元素微调，配合绝对定位使用。  1.3绝对定位（AbsolutePositioning）  定义:绝对定位是相对于最近的定位流祖先元素来定位，没有则以body为参考点。  使用方法:position:absolute;  参考点:  默认情况下，以body为参考点。  如果有定位流祖先元素，以最近的定位流祖先元素为参考点。  注意点:  脱离标准流，不占用标准流中的空间。  不区分块级元素、行内元素、行内块级元素。  忽略祖先元素的padding。  子绝父相:子元素绝对定位，父元素相对定位。  水平居中:设置left:50%;然后margin-left:-元素宽度的一半px;。  日期2024.7.4 |

Day09笔记

|  |
| --- |
| 1.4固定定位（FixedPositioning）  定义:固定定位使元素相对于浏览器窗口固定，不随滚动条滚动。  使用方法:position:fixed;  注意点:  脱离标准流，不占用标准流中的空间。  不区分行内、块级、行内块级元素。  1.5粘滞定位（StickyPositioning）  定义:粘滞定位结合了相对定位和固定定位的功能，适用于特殊场景。  使用方法:position:sticky;，同时设置top、bottom、left、right中的一个。  表现:  元素在跨越特定阈值前为相对定位，之后为固定定位。  保留元素在文档流中的位置。  使用条件:  父元素不能设置overflow:hidden或overflow:auto。  必须指定top、bottom、left、right中的一个。  父元素的高度不能低于粘滞元素的高度。  1.6z-index  定义:用于控制定位流元素的覆盖关系。  默认值:0  覆盖规则:  定位流元素覆盖标准流元素。  后编写的定位流元素覆盖前面的。  设置了z-index属性的元素，值较大的显示在上面。  注意点:  如果两个元素的父元素都没有设置z-index属性，比较z-index值。  如果父元素设置了z-index属性，子元素的z-index属性失效，比较父元素的z-index值。  小结  静态定位是默认的定位方式，元素在文档流中按照自然顺序排列。  相对定位使元素相对于自身原位置移动，但不脱离文档流。  绝对定位使元素脱离文档流，相对于最近的定位流祖先元素定位。  固定定位使元素固定在浏览器窗口中，不随滚动条滚动。  粘滞定位结合相对和固定定位，适用于特殊场景。  z-index控制定位流元素的层叠顺序。  日期2024.7.5 |

第三周

Day10笔记

|  |
| --- |
| 一、CSS 过渡概述  CSS 过渡提供了一种在更改 CSS 属性时控制动画速度的方法。它可以让属性变化成为一个持续一段时间的过程，而不是立即生效的。  二、实现步骤  要实现 CSS 过渡效果，需要规定以下两项内容：  指定要添加效果的 CSS 属性。  指定效果的持续时间。  三、过渡三要素  必须有属性发生变化。  必须告诉系统哪个属性需要执行过渡效果。  必须告诉系统过渡效果持续时长。  四、过渡触发方式  鼠标悬停触发  用户单击元素并按住鼠标时触发  获得焦点时触发  @media 符合媒体查询条件时触发  点击事件 用户点击元素时触发  日期2024.7.8 |

Day11笔记

|  |
| --- |
| 五、CSS 过渡属性  transition-property: 规定应用过渡的 CSS 属性的名称。  none: 没有属性会获得过渡效果。  all: 所有属性都将获得过渡效果。  property: 定义应用过渡效果的 CSS 属性名称列表，列表以逗号分隔。  transition-duration: 定义过渡效果花费的时间。默认是 0。  time: 规定完成过渡效果需要花费的时间（以秒或毫秒计）。  transition-timing-function: 规定过渡效果的时间曲线。默认是 "ease"。  linear: 以相同速度开始至结束的过渡效果。  ease: 慢速开始，然后变快，然后慢速结束的过渡效果。  ease-in: 以慢速开始的过渡效果。  ease-out: 以慢速结束的过渡效果。  ease-in-out: 以慢速开始和结束的过渡效果。  cubic-bezier(n,n,n,n): 在 cubic-bezier 函数中定义自己的值。  transition-delay: 规定过渡效果何时开始。默认是 0。  time: 指定秒或毫秒数之前要等待切换效果开始。  六、注意点  当多个属性需要同时执行过渡效果时用逗号隔开即可：  transition-property: width, background-color;  transition-duration: 5s, 5s;  transition 是 transition-property， transition-duration，transition-timing-function，transition-delay 的速写形式。  七、过渡连写格式  div {  transition: <property> <duration> <timing-function> <delay>;  }  示例：  /\* 单个属性 \*/  transition: background-color 5s linear 0s;  /\* 多个属性 \*/  transition: width 5s, background-color 5s, height 5s;  八、连写注意点  和分开写一样，如果想给多个属性添加过渡效果也是用逗号隔开即可。  连写时可以省略后面的两个参数，因为只要编写了前面的两个参数就已经满足了过渡的三要素。  如果多个属性的运动速度、延迟时间、持续时间都一样，可以简写为：  transition: all 0s;  日期2024.7.9 |

Day12笔记

|  |
| --- |
| CSS 动画  动画的定义  动画模块 (Animations) 是 CSS3 的一个模块，使用 keyframes 定义如何随着时间改变 CSS 的属性值。  动画由两部分组成：  CSS 动画的配置  一系列的关键帧（描述动画的开始、过程、结束状态）  过渡和动画的异同  不同点  过渡必须人为触发才会执行。  动画不需要人为触发就可以执行。  相同点  过渡和动画都是用来给元素添加动画的。  过渡和动画都需要满足三要素才会有动画效果。  动画属性  animation-name：指定要绑定到选择器的关键帧的名称。  animation-duration：动画指定需要多少秒或毫秒完成。  animation-timing-function：设置动画将如何完成一个周期。  linear：动画从头到尾的速度相同。  ease：默认，动画以低速开始，然后加快，在结束前变慢。  ease-in：动画以低速开始。  ease-out：动画以低速结束。  ease-in-out：动画以低速开始和结束。  cubic-bezier(n,n,n,n)：自定义速度曲线。  animation-delay：设置动画在启动前的延迟间隔。  animation-iteration-count：定义动画的播放次数。  animation-direction：指定是否轮流反向播放动画。  normal：默认，执行完一次后回到起点继续执行下一次。  alternate：往返动画，执行完一次后往回执行下一次。  reverse：反向执行。  animation-fill-mode：规定当动画不播放时要应用到元素的样式。  none：不做任何改变。  forwards：结束状态保持动画最后一帧的样式。  backwards：等待状态显示动画第一帧的样式。  both：等待状态显示动画第一帧的样式，结束状态保持动画最后一帧的样式。  animation-play-state：告诉系统当前动画是否需要暂停。  running：执行动画。  paused：暂停动画。  动画模块连写格式  格式：animation: 动画名称 animation-duration animation-timing-function animation-delay animation-iteration-count animation-direction;  简写：animation: 动画名称 动画时长;  日期2024.7.10 |

Day13笔记

|  |
| --- |
| 2D 转换模块  转换属性  transform 属性向元素应用从 2D 或 3D 转换，可以旋转、缩放、移动或倾斜元素。  常用转换类型  旋转 (rotate)  语法：transform: rotate(45deg);  其中 deg 是单位，表示旋转的角度。  平移 (translate)  语法：transform: translate(100px, 0px);  第一个参数：水平方向。  第二个参数：垂直方向。  缩放 (scale)  语法：transform: scale(1.5); 或 transform: scale(0.5, 0.5);  第一个参数：水平方向。  第二个参数：垂直方向。  注意：取值为 1 表示不变，大于 1 表示放大，小于 1 表示缩小。  综合转换连写格式  语法：transform: rotate(45deg) translate(100px, 0px) scale(1.5, 1.5);  注意：多个转换用空格隔开，2D 的转换模块会修改元素的坐标系，旋转后再平移不是水平平移，是按照旋转后的角度进行平移。  形变中心点  transform-origin  第一个参数：水平方向。  第二个参数：垂直方向。  取值有三种形式：具体像素、百分比、特殊关键字（如 center）。  旋转轴向  默认所有元素围绕 Z 轴旋转：  transform: rotateZ(45deg);  transform: rotateX(45deg);  transform: rotateY(45deg);  透视 (perspective)  定义 3D 元素距视图的距离，以像素计。  允许改变 3D 元素查看的视图，子元素会获得透视效果，而不是元素本身。  透视属性 必须添加到需要呈现近大远小效果的元素的父元素上。  日期2024.7.11 |

Day14笔记

|  |
| --- |
| HTML与CSS考试  http://218.4.196.178:8899/sms/  HTML与CSS总复习  日期2024.7.12 |

第四周  
Day15笔记

|  |
| --- |
| 静态页面项目——网易严选页面的重构  1.完成导航栏架构的编写  2.完成导航栏样式的编写  /\* 清除原生自带的样式 \*/  \*{  margin: 0;  padding: 0;  list-style: none;  }  /\* 根节点的样式 \*/  body{  width: 100%;  }  /\* 主容器的样式 \*/  .wyContainer{  height: 2000px;  /\* 最小宽度 \*/  min-width: 1150px;  /\* background-color: antiquewhite; \*/  }  /\* 顶部区域的样式 \*/  .header{  width: 100%;  height: 176px;  }  /\* 顶部导航栏区域的样式 \*/  .header .headerNav{  width: 100%;  height: 36px;  background-color: #333333;  }  /\* 顶部导航栏的样式 \*/  .headerNav .nav{  width: 1090px;  height: 36px;  margin: 0 auto;  display: flex;  justify-content: flex-end;  background-color: #333333;  }  /\* 每一个导航子选项的样式 \*/  .nav>div{  line-height: 36px;  cursor: pointer;  font-size: 12px;  color: #CCCCCC;  }  /\* 第四个 div 设置样式 \*/  .nav>div:nth-child(4){  color: #CC9756;  position: relative;  }  /\* 企业采购下拉列表的样式 \*/  .nav>div:nth-child(4) .selectMenu{  width: 83px;  height: 100px;  background-color: white;  position: absolute;  left: 10px;  display: none;  /\* 盒子阴影 \*/  box-shadow: 0px 0px 2px 2px gray;  }  /\* 鼠标移入企业采购时 弹出下拉列表 \*/  .nav>div:nth-child(4):hover .selectMenu{  display: block;  }  /\* 列表项的样式 \*/  .selectMenu>div{  width: 70%;  line-height: 33px;  margin: 0 auto;  text-align: center;  font-size: 14px;  color: #999999;  }  .selectMenu>div:nth-child(n+3){  border-top: 1px solid gainsboro;  }  /\* 鼠标移入时改变蚊子的颜色 \*/  .nav>div:hover{  color: white;  }  日期2024.7.15 |

Day16笔记

|  |
| --- |
| 静态页面项目——网易严选页面的重构  1.iconfont的使用  2.完成lcon、搜索栏、购物车的编写  /\* 每一个导航子选项前的边框样式 \*/  .nav>div:nth-child(n+2)::before{  content: '';  display: inline-block;  height: 16px;  border-left: 1px solid #5C5C5C;  vertical-align: middle;  margin: 0 10px;  }  /\* :has()里面是选择子级的选择器，通过这个子级的选择器来选择前面的选择器的父级 \*/  .nav>div:nth-child(4):has(.selectMenu:hover){  color: orange;  }  /\* 字体图标的样式 \*/  .icon-xiangxiajiantou{  font-size: 8px;  }  .icon-shouji-copy{  font-size: 10px;  color: white;  }  .icon-xiangxiasanjiao{  color: white;  position: absolute;  top: -20px;  left: 50%;  margin-left: -8px;  }  /\* 搜索区域的样式 \*/  .searchArea{  width: 1090px;  height: 110px;  margin: 0 auto;  display: flex;  }  /\* logo 区域的样式 \*/  .searchArea .logo{  width: 20%;  height: 100%;  display: flex;  align-items: center;  }  /\* logo 的样式 \*/  .logo img{  width: 100%;  }  /\* 搜索框的样式 \*/  .searchArea .sInput{  width: 60%;  height: 100%;  display: flex;  justify-content: center;  align-items: center;  }  .sInput .input{  width: 530px;  height: 38px;  display: flex;  }  /\* 左侧输入框的样式 \*/  .input input{  width: 80%;  height: 38px;  box-sizing: border-box;  border: 1px solid #CC9756;  border-top-left-radius: 20px;  border-bottom-left-radius: 20px;  outline: none;  /\* 背景图片属性设置 icon \*/  background-image: url('./static/search.png');  background-repeat: no-repeat;  background-size: 16px 16px;  background-position: left 10px center;  padding-left: 38px;  }  日期2024.7.16 |

Day17笔记

|  |
| --- |
| 静态页面项目——网易严选页面的重构  1.完成页面轮播图  2.完成页面整体架构的编写  /\* 商品展示区域 \*/  .proArea1 .proShow{  width: 100%;  height: 365px;  display: flex;  justify-content: space-between;  }  /\* 商品卡片的整体样式 \*/  .proShow>div{  width: 265px;  height: 365px;  }  /\* 商品卡片背景图片样式 \*/  .proShow>div .proImg{  width: 265px;  height: 265px;  background-color: antiquewhite;  }  /\* 第一个卡片的背景图片 \*/  .p1 .proImg{  background-image: url('./static/1.png');  background-size: 265px 265px;  }  /\* p1 hover时 改变背景图片 \*/  .p1:hover .proImg{  background-image: url('https://yanxuan-item.nosdn.127.net/da59d01f74679ddef05480b4cdeff6ac.jpg?type=webp&quality=95&thumbnail=265x265&imageView');  background-size: 265px 265px;  }  /\* p1hover 时 样式的设置 \*/  .p1:hover{  box-shadow: 0px 0px 3px 3px gainsboro;  background-color: #F4F0EA;  }  /\* 商品描述的样式 \*/  .proShow>div .proDesc{  width: 70%;  margin: 0 auto;  text-align: center;  font-size: 14px;  font-weight: bold;  margin-top: 10px;  }  /\* 商品价格的样式 \*/  .proShow>div .proPrice{  width: 40%;  margin: 0 auto;  text-align: center;  color: #D4282D;  font-size: 15px;  font-weight: lighter;  margin-top: 10px;  }  /\* 原价的样式 \*/  .zkPrice{  color: gray;  text-decoration: line-through;  }  日期2024.7.17 |

Day18笔记

|  |
| --- |
| JavaScript  1.初识JavaScript  1.1什么是JavaScript  JavaScript(JS)是一种具有函数优先的轻量级、解释型或即时编译型的编程语言。虽然它因作为开发网页的脚本语言而出名，但它也被用于很多非浏览器环境中，例如Node.js、ApacheCouchDB和AdobeAcrobat。JavaScript是一种基于原型编程、多范式的动态脚本语言，并且支持面向对象、命令式和声明式（如函数式编程）风格。  1.2JavaScript的组成部分  ECMAScript：描述了该语言的语法和基本对象。  文档对象模型（DOM）：描述处理网页内容的方法和接口。  浏览器对象模型（BOM）：描述与浏览器进行交互的方法和接口。  1.3JavaScript的历史  -1994年，网景公司（Netscape）发布了Navigator浏览器0.9版，这是历史上第一个比较成熟的网络浏览器。  -1995年，网景公司决定开发一种网页脚本语言，使浏览器可以与网页互动。  -1995年，BrendanEich设计了JavaScript，最初命名为Mocha，后改名为LiveScript，再改名为JavaScript。  -1996年，微软发布了支持JScript的IE3。  -1997年，第一个ECMAScript标准发布。  2.JavaScript的特点  1.\*\*解释型语言\*\*：被内置于浏览器或Node.js平台中的JS解析器解析执行，无需编译。  2.\*\*弱类型语言\*\*：变量的数据类型取决于值的数据类型。  3.\*\*顺序执行\*\*：从上到下顺序解析执行。  4.\*\*跨平台\*\*：既可以作为前端脚本语言，也可以作为后端语言。  3.JavaScript的组成  1.ECMAScript：JS的标准，类似于核心Java，制定了基础的语法。  2.DOM：文档对象模型，JS操作HTML的API。  3.BOM：浏览器对象模型，JS操作浏览器的API。  4.在网页中使用JavaScript  4.1内部JavaScript  在<head>标签中添加<script>元素，然后将JS代码填写进来即可。  <!DOCTYPEhtml>  <htmllang="en">  <head>  <metacharset="UTF-8">  <title>Document</title>  <scripttype="text/javascript">  alert('HelloJavaScript');  </script>  </head>  <body>  </body>  </html>  4.2外部JavaScript  单独新建一个后缀名为.js的JS文件，在HTML文件中使用<script>标签的src属性将JS文件导入进来。  <!DOCTYPEhtml>  <htmllang="en">  <head>  <metacharset="UTF-8">  <title>Document</title>  <scripttype="text/javascript"src="hello.js"></script>  </head>  <body>  </body>  </html>  5.编辑器与运行环境  -编辑器：vi、vim、vscode、sublime。  -运行环境：浏览器控制台、Node.js。  6.JavaScript关键字与保留字  6.1关键字  在JS中有特殊功能的保留字，如break、case、catch、continue、debugger、default、delete、do、else、finally、for、function、if、in、instanceof、new、return、switch、this、throw、try、typeof、var、void、while、with。6.2保留字  将来可能成为关键字的保留字，如abstract、boolean、byte、char、class、const、debugger、double、enum、export、extends、final、float、goto、implements、import、int、interface、long、native、package、private、protected、public、short、static、super、synchronized、throws、transient、volatile。  7.注释  单行注释：//aaa  多行注释：/\*aaa\*/  日期2024.7.18 |

Day19笔记

|  |
| --- |
| undefined继承null，所以undefined == null结果为true，但是null表示空对象，undefined表示未定义；null与undefined⽤途不同，null可以⽤来表示⼀个空对象但是没有必要把⼀个变量的值显式设置为undefined  深拷⻉与浅拷⻉主要针对于引⽤数据类型参数说的，浅拷⻉表示仅拷⻉引⽤地址，深拷⻉ 表示对于对象的克隆。  实现深拷⻉的⽅法 通过json对象实现深拷⻉（JSON.stringify，JSON.parse） Object.assign()拷⻉ lodash函数库实现深拷⻉ 递归的⽅式实现深拷⻉  使⽤typeof判断数据类型  返回该变量名所指向数据的类型 语法：  typeof 变量名 返回值如下：  "undefined"未定义  "boolean"布尔类型  "string" 字符串  "number"数值  "object" 对象或者null或者数组  "function" 函数  日期2024.7.19 |

第五周

Day20笔记

|  |
| --- |
| **1. 操作符**  1.1 算术运算符   * **加法 (+)**: 数字相加，字符串拼接。   var num = 1 + 2; // 3  var str = "Hello" + " " + "World"; // "Hello World"   * **减法 (-)**: 数字相减，可以将字符串转换成数字后相减。   var num = 4 - 3; // 1  var num = "8" - "4"; // 4   * **乘法 (\*)**: 数字相乘，可以将字符串转换成数字后相乘。   var num = 4 \* 3; // 12   * **除法 (/)**: 数字相除，可以将字符串转换成数字后相除。   var num = 6 / 3; // 2   * **取余 (%)**: 取余数操作。   var num = 7 % 3; // 1  1.2 一元运算符   * **正号 (+)**: 将操作数转换成数字。   var num = +"123"; // 123   * **负号 (-)**: 将操作数转换成负数。   var num = -"123"; // -123   * **逻辑取反 (!)**: 将布尔值取反。   var bool = !true; // false   * **递增 (++)**: 变量自增1。   var num = 1;  num++; // 2   * **递减 (--)**: 变量自减1。   var num = 1;  num--; // 0   * **删除 (delete)**: 删除对象或数组的属性。   var obj = { name: 'John', age: 30 };  delete obj.age; // { name: 'John' }   * **类型判断 (typeof)**: 返回操作数的类型。   var type = typeof 123; // "number"  **2. 赋值运算符**   * **赋值 (=)**: 赋值操作。   var x = 5;   * **加等 (+=)**: 相加后赋值。   var x = 5;  x += 2; // 7   * **减等 (-=)**: 相减后赋值。   var x = 5;  x -= 2; // 3   * **乘等 (\*=)**: 相乘后赋值。   var x = 5;  x \*= 2; // 10   * **除等 (/=)**: 相除后赋值。   var x = 5;  x /= 2; // 2.5   * **取余等 (%=)**: 取余后赋值。   var x = 5;  x %= 2; // 1  **3. 比较运算符**   * **等于 (==)**: 仅值相等即可。   var result = (5 == "5"); // true   * **全等 (===)**: 值和类型都要相等。   var result = (5 === "5"); // false   * **不等于 (!=)**: 值不等即可。   var result = (5 != "5"); // false   * **不全等于 (!==)**: 值或类型不等。   var result = (5 !== "5"); // true   * **大于 (>)**: 大于。   var result = (5 > 3); // true   * **小于 (<)**: 小于。   var result = (5 < 3); // false   * **大于等于 (>=)**: 大于或等于。   var result = (5 >= 3); // true   * **小于等于 (<=)**: 小于或等于。   var result = (5 <= 3); // false  **4. 隐式转换**   * **ToPrimitive**: 将值转换为原始值。 * **ToNumber**: 将值转换为数字。 * **ToString**: 将值转换为字符串。   **隐式转换规则**   1. 字符串与数字相加时，数字会被转换为字符串。   var result = 1 + "2"; // "12"   1. 字符串与数字相减时，字符串会被转换为数字。   var result = "5" - 3; // 2  **5. 流程控制语句**   1. **if 语句**: 条件判断。   if (condition) {  // TODO  }   1. **switch 语句**: 多条件判断。   switch (expression) {  case value1:  // TODO  break;  case value2:  // TODO  break;  default:  // TODO  }   1. **for 循环**: 计数器控制的循环。   for (var i = 0; i < 5; i++) {  // TODO  }   1. **while 循环**: 条件控制的循环。   j  while (condition) {  // TODO  }   1. **do...while 循环**: 至少执行一次的循环。   do {  // TODO  } while (condition);  **使用注意**   * 尽量使用严格比较运算符（===），避免隐式转换带来的困惑。 * 注意类型转换的规则和顺序，避免意外的结果。   日期2024.7.22 |

Day21笔记

|  |
| --- |
| **1. JavaScript 对象基础**   * **对象定义**: JavaScript中的对象是无序属性的集合，其属性可以包含基本值、对象或函数。对象是由键值对（key   ）组成，值可以是数据或函数。   * **现实中的对象类比**: 每个人都是一个对象，具有属性（如身高和体重）和方法（如走路和跑步）。每个人的属性和方法虽相似，但具体值和执行时间各不相同。 * **对象创建**:   + **字面量模式**: 使用 {} 作为对象的边界，属性之间用 , 隔开，属性名和属性值用 : 隔开。   + **构造函数模式**: 使用 Object 或自定义构造函数来初始化对象。   var person = {name:"张三", height:1.71, gender: 'male'};  var obj = new Object();  obj.name = "terry";  obj.age = 12;  obj.sayName = function() {  console.log("my name is", this.name);  };  **2. 对象的访问**   * **属性访问**: 有点访问（.）和中括号访问（[]）两种方式。   console.log(obj.name);  console.log(obj['name']);   * **方法的访问**: 需要按照函数调用的方式使用对象的方法。   obj.sayName(); // 调用对象的方法  **3. 遍历对象的属性**   * 使用 for..in 循环遍历对象的属性。   for (var key in obj) {  console.log(key + "--" + obj[key]);  }  **4. 新增和删除对象中的属性**   * 新增属性可以直接赋值。   obj.newProperty = "newValue";   * 删除属性使用 delete 关键字。   delete obj.propertyName;  **5. 对象的强制类型转换**   * **转换为 Boolean**:   var obj = { name: "briup", age: 12 };  console.log(Boolean(obj)); // true   * **转换为 String**: 调用 toString 方法。   console.log(obj.toString());   * **转换为 Number**: 调用 valueOf 方法。   var obj = {  valueOf: function() { return 10; }  };  console.log(Number(obj)); // 10  日期2024.7.23 |

Day22笔记

|  |
| --- |
| **开题报告、开题PPT，任务书，选题审批表**  **1. 检测属性**   * **in 运算符**: 检测属性是否存在于对象中，无论是自有属性还是继承属性。   console.log('name' in obj); // true   * **hasOwnProperty**: 检测属性是否为对象的自有属性。   console.log(obj.hasOwnProperty('name')); // true   * **propertyIsEnumerable**: 检测属性是否为对象的可枚举自有属性。   console.log(obj.propertyIsEnumerable('name')); // true  **2. 对象原型属性及方法**   * **Object 构造函数的原型属性和方法**: 包括 constructor、hasOwnProperty、propertyIsEnumerable、valueOf、toLocaleString、toString 和 isPrototypeOf。   var obj = new Object();  console.log(obj.toString());  console.log(Object.prototype);  **3. 深入理解对象-定义属性**   * **数据属性**: 包含 [[Configurable]]、[[Enumerable]]、[[Writable]] 和 [[Value]] 特性。   Object.defineProperty(obj, 'name', {  configurable: true,  enumerable: true,  writable: true,  value: 'terry'  });   * **访问器属性**: 包含 [[Configurable]]、[[Enumerable]]、[[Get]] 和 [[Set]] 特性。   Object.defineProperty(obj, 'fullName', {  get: function() { return this.firstName + ' ' + this.lastName; },  set: function(value) {  var parts = value.split(' ');  this.firstName = parts[0];  this.lastName = parts[1];  }  });  日期2024.7.24 |

Day23笔记

|  |
| --- |
| **函数概述与声明**  **1. 函数介绍**   * **定义**：函数是封装一系列代码以完成特定任务的结构。 * **用途**：   + 功能封装，提高代码复用率   + 构建对象的模板（构造函数） * **函数类型**：每个函数都是 Function 类型的实例，具有属性和方法。   **2. 函数声明与表达式**   * **函数声明**：   function 函数名(形参列表) {  // 函数体  }  示例：  function add(a, b) {  var result = a + b;  return result;  }  var total = add(1, 2); // total = 3   * **函数表达式**：   var 函数名 = function(形参列表) {  // 函数体  }  示例：  var add = function(a, b) {  var result = a + b;  return result;  }  **3. 函数提升**   * 函数声明会被提升到代码的最前面，可以在声明前调用。   foo(); // 可以调用  function foo() {  console.log("Hello World");  }   * 函数表达式不会提升，必须在赋值后调用。   **4. 函数内部属性**   * **arguments**：类数组对象，包含传入函数的所有参数。   + arguments.callee：指向拥有该 arguments 对象的函数。   function add(a, b) {  console.log(arguments[0], arguments[1]);  return a + b;  }  add(10, 20); // 输出：10 20  **5. this 关键字**   * **用法**：   + 方法中：this 指向方法所属的对象。   + 函数中：默认指向全局对象（浏览器中为 window，Node.js 中为 global）。   + 事件中：指向接收事件的元素。   + 显式绑定：使用 call 和 apply 切换 this 的上下文。   var person = {  firstName: "张",  lastName: "建安",  fullName: function() {  return this.firstName + " " + this.lastName;  }  };  console.log(person.fullName()); // 输出  **1. IIFE（立即调用函数表达式）**   * **定义**：声明函数的同时立即调用该函数。   (function() {  console.log("IIFE called");  })();   * **用途**：   + 只执行一次的初始化代码   + 创建局部作用域，防止变量污染全局作用域   **2. 作用域**   * **全局作用域**：函数外声明的变量，全局可访问。 * **局部作用域**：函数内部声明的变量，仅在函数内可访问。   **3. 作用域链**   * **定义**：嵌套函数在当前作用域找不到变量时，会沿着作用域链向上查找直至全局作用域。   var a = 100;  function outer() {  var b = 200;  function inner() {  var c = 300;  console.log(a); // 100  console.log(b); // 200  console.log(c); // 300  }  inner();  }  outer();  **4. 闭包**   * **定义**：函数能够记住并访问其词法作用域，即使函数在其词法作用域之外执行。   function outer() {  var a = 100;  return function inner() {  console.log(a); // 100  }  }  var fn = outer();  fn();  **5. 经典面试题 - IIFE 和作用域**   * **IIFE 用于循环中的变量隔离**：   for (var i = 0; i < 5; i++) {  (function(j) {  console.log(j); // 输出：0, 1, 2, 3, 4  })(i);  }   * **解释**：IIFE 创建了一个新的作用域，使得每次循环的 i 值作为参数传递给 IIFE，从而避免变量提升导致的错误。   日期2024.7.25 |

Day24笔记

|  |
| --- |
| 什么是闭包？  简单讲，闭包就是指有权访问另⼀个函数作⽤域中的变量的函数。  MDN 上⾯这么说：闭包是⼀种特殊的对象。它由两部分构成：函数，以及创建该函数的环境。环境由闭包创建时在作⽤域中的任何局部变量组 成。  闭包的⽣成有三个必要条件 函数嵌套函数 内部函数引⽤了外部函数中的数据（属性、函数） 参数和变量不会被回收 这样就形成了⼀个不会销毁的函数空间 产⽣⼀个闭包  创建闭包最常⻅⽅式，就是在⼀个函数内部创建另⼀个函数。下⾯例⼦中 的 closure 就是⼀个闭包：  function func() {  var a = 1, b = 2;  function closure() {  return a + b;  }  return closure;  }  console.log(func()());  闭包的作⽤域链包含着它⾃⼰的作⽤域，以及包含它的函数的作⽤域和全局作⽤域。  在Javascript语⾔中，只有函数内部的⼦函数才能读取局部变量，因此可以把闭包简单理解成"定义在⼀个函数内部的函数"。所以，在本质上，闭包就是将函数内部和函数外部连接起来的⼀座桥梁。  闭包的⽤途，闭包可以⽤在许多地⽅。它的最⼤⽤处有两个，⼀个是前⾯提到的可以读 取函数内部的变量，另⼀个就是让这些变量的值始终保持在内存中。  function f1() {  var n = 999;  nAdd = function () {  n += 1  }  function f2() {  console.log(n);  }  return f2;  }  var result = f1();  result(); // 999  nAdd();  result(); // 1000  =使⽤闭包的注意点  1）由于闭包会使得函数中的变量都被保存在内存中，内存消耗很⼤，所以 不能滥⽤闭包，否则会造成⽹⻚的性能问题，在IE中可能导致内存泄露， 这是IE的BUG。解决⽅法是，在退出函数之前，将不使⽤的局部变量全部 删除。  2）闭包会在⽗函数外部，改变⽗函数内部变量的值。所以，如果你把⽗函 数当作对象（object）使⽤，把闭包当作它的公⽤⽅法（Public Method），把内部变量当作它的私有属性（private value），这时⼀定要 ⼩⼼，不要随便改变⽗函数内部变量的值。多个⼦函数的scope都是同时指 向⽗级，是完全共享的。因此当⽗级的变量对象被修改时，所有⼦函数都 受到影响。  **字⾯量创建数组**  由⼀对包括元素的⽅括号"[]"表示，元素之间以逗号","隔开  var names = ["terry","larry","tom"]  var name = "name"  // 可以接收任何类型的数据当做数组的元素  var arr = [12,name,true,"larry",{},function(){},[],null];  // 通过数组下标获取对应的数据  console.log(arr,arr[2]);  // 获取数组的⻓度  console.log(arr.length);//arr.length  构造函数创建数组  通过Array构造函数来创建数组  var names = new Array(); // 等价于 var names = [];  // 如果⼀个参数，并且是number类型的整数，则代表的是数组的⻓度。如果是  number类型的⼩数，则报错。如果是其他类型，则当做数组元素放进去。  //var arr = new Array(length); 创建⼀个⻓度为length的数组  var names = new Array(3);  // 创建⼀个包含3个元素的数组  var arr = [undefined,undefined,undefined];  //var ages = new Array(2.4);//Error: Invalid array length  var ages = new Array("2.4");  var names = new Array('terry') //创建⼀个包含1个元素的数组，该元素的值为'terry'  // 两个参数或者多个参数,当做数组元素放进去  var names = new Array('terry','robin')//创建⼀个数组，数组中的元素使⽤实参初始化  通过索引访问数组，数组的索引从0开始，数组的索引超过数组⻓度会访问  到undefined值⽽不会报错。数组的⻓度通过length属性获取  a) [index] 直接访问,索引可以超过索引范围，只不过访问的值为undefined  b) length-1=Max(index)  c) length+N 或 length-N 开辟新的内存空间 或 数组元素的删除  日期2024.7.26 |

Day25笔记

|  |
| --- |
| **数组 API**  **1. 数组的创建与访问**  var array1 = new Array(10);  var array2 = [1, 2, 3];  console.log(array2[0]); // 1  console.log(array1.length); // 10  通过数组字面量和构造函数创建数组是JavaScript中的基础操作。特别是在处理动态数据时，`length` 属性的灵活性非常有用，可以控制数组的大小。但是要注意越界访问会返回 `undefined`，这在调试时可能导致意料之外的错误。  **2. 数组方法示例**  var stack = [];  stack.push(1);  stack.push(2);  console.log(stack.pop()); // 2  `push` 和 `pop` 方法模拟了栈的行为，在需要后进先出（LIFO）数据结构时非常有用。这些方法非常直观且易于使用。  var queue = [];  queue.unshift(1);  queue.unshift(2);  console.log(queue.shift()); // 2  `shift` 和 `unshift` 方法适用于队列操作，模拟了先进先出（FIFO）的数据结构。值得注意的是，这些方法在大数组上可能不如 `push` 和 `pop` 高效，因为它们涉及数组的重排。  var arr = [1, 2, 3, 4, 5];  arr.splice(2, 1, 'a', 'b');  console.log(arr); // [1, 2, 'a', 'b', 4, 5]  `splice` 是非常强大的数组方法，可以在数组中添加、删除或替换元素。灵活性很高，但也容易导致代码复杂性增加，因此在使用时应注意维护代码的可读性。  **正则表达式**  **1. 创建和匹配正则表达式**  var regex = /ab+c/;  console.log(regex.test('abc')); // true  var regex = new RegExp('ab+c');  console.log(regex.test('abc')); // true  正则表达式是字符串处理的强大工具。无论是字面量方式还是构造函数方式，正则表达式都能让我们高效地进行模式匹配。理解和掌握正则表达式语法对于处理复杂字符串匹配场景非常重要。  2. 常用方法  var regex = /a(bc)d/;  var result = regex.exec('abcd');  console.log(result); // ['abcd', 'bc']  `exec` 方法返回匹配结果的数组，并包含捕获组的信息，非常适合需要详细匹配结果的情况。需要注意的是，当使用全局标志 `g` 时，`exec` 会在多次调用之间记录匹配位置。  var regex = /hello/;  console.log(regex.test('hello world')); // true  `test` 方法用于检测字符串是否匹配正则表达式，简单直观。它返回布尔值，是快速检查字符串内容的首选方法，适合需要快速验证的场景。  **3. 边界符**  匹配字符串的开始和结束  var regexStart = /^abc/;  var regexEnd = /abc$/;  console.log(regexStart.test('abcde')); // true  console.log(regexEnd.test('deabc')); // true  边界符 `^` 和 `$` 非常适合匹配字符串的开头和结尾，尤其在验证格式和模式时，如验证特定前缀或后缀。使用边界符可以提高正则匹配的准确性，但需要小心使用，以确保匹配的灵活性。  **总结**  通过对数组 API 和正则表达式的学习，我们可以看到它们在处理数据和字符串操作时的强大之处。数组 API 提供了丰富的操作方法，让我们能够灵活地操控数据集合；正则表达式则是处理字符串匹配和提取的利器。了解这些 API 的边界情况，能够帮助我们在编写代码时避免常见错误，并写出更加健壮的程序。  日期2024.8.19 |

Day26笔记

|  |
| --- |
| JavaScript中的原型与对象  一、JavaScript 原型与原型链  1.1 函数  function add(a, b) {  return a + b;  }  function Person(name, age) {  this.name = name;  this.age = age;  }  var p1 = new Person('张三', 18);  函数不仅是基本的操作单元，它还可以作为构造函数来创建对象。理解函数的双重角色（普通函数和构造函数）对于掌握原型链至关重要。`new` 操作符在创建对象时，背后会涉及到原型链的构建。  1.2 属性  \_\_proto\_\_: 对象的私有属性，指向其原型对象。  prototype: 函数独有属性，指向实例对象的原型对象。  constructor: 对象的构造函数属性，指向创建该对象的函数。  console.log(p1.\_\_proto\_\_ === Person.prototype); // true  console.log(p1.constructor); // Person  这些属性揭示了对象与其原型对象之间的关系。`\_\_proto\_\_` 和 `prototype` 在继承和属性查找过程中起着核心作用，而 `constructor` 则指明了对象的来源。  1.2.1 prototype  Person.prototype.sayHi = function() {  console.log(this.name, '说hi');  };  p1.sayHi(); // 张三 说hi  p2.sayHi(); // 李四 说hi  `prototype` 是实现对象共享属性和方法的关键。将方法定义在 `prototype` 上可以让所有实例共享该方法，避免了重复定义。但需要注意的是，原型方法的共享性意味着修改时需要考虑所有实例的影响。  二、深入理解对象  2.1 对象的创建  function createStudent(name, age) {  var stu = new Object();  stu.name = name;  stu.age = age;  stu.sayHi = function() {  console.log(this.name + '说Hi');  };  return stu;  }  var stu1 = createStudent('张三', 18);  ```  手动创建对象虽然简单，但在大量创建时效率低且容易出错。构造函数或工厂模式封装了对象的创建逻辑，可以大大提高代码的复用性和维护性。  2.2 instanceof 操作符  console.log(stu1 instanceof Student); // true  `instanceof` 检查对象是否是某个构造函数的实例，其本质是通过原型链判断构造函数的 `prototype` 是否出现在对象的原型链上。这个操作符在类型检查和多态性实现中非常重要。  2.3 组合模式  Student.prototype.sayHi = function() {  console.log(this.name + '说Hi');  };  var stu1 = new Student('张三', 18);  var stu2 = new Student('李四', 20);  console.log(stu1.sayHi === stu2.sayHi); // true  将方法定义在原型上是提高性能和代码可维护性的常见策略，避免了每个实例重复创建方法。但要注意的是，所有实例共享同一个方法，如果需要特定实例有不同的行为，需要对方法进行适当扩展或覆盖。  2.4 in 操作符  console.log('name' in stu1); // true  `in` 操作符不仅检查对象自身是否有某个属性，还会沿着原型链查找。这在调试和验证对象属性时非常有用，但也需要注意避免意外地访问到继承自原型的属性。  总结：  在 JavaScript 中，原型链和对象的关系决定了属性和方法的查找规则。理解 `\_\_proto\_\_`、`prototype` 和 `constructor` 等属性，以及如何使用 `instanceof`、`in` 等操作符来操控对象，能够帮助我们更深入地掌握 JavaScript 的面向对象编程思想。这些知识点不仅在原生 JS 中重要，在使用如 Vue 这样依赖原型机制的框架时也尤为关键。  日期2024.8.20 |

Day27笔记

|  |
| --- |
| JavaScript事件与继承  一、JavaScript事件  JavaScript事件是浏览器和用户之间互动的核心机制。事件由三部分组成：事件源、事件类型、事件处理程序，这三者共同构成了事件的完整过程。  1.事件的三要素  事件源：触发事件的HTML元素，如按钮或文本框。  事件类型：事件的类型，如`click`（点击）、`mouseover`（鼠标悬停）等。  事件处理程序：响应事件的JavaScript函数。  2.事件流  事件捕获：事件从最不具体的元素向最具体的元素传播。类似将点击事件从`document`传递到目标元素。  事件冒泡：事件从最具体的元素向外层元素传播，直到`document`为止。它是事件的默认传播方式。  3.事件绑定示例  <button id="btn">点我啊</button>  <script>  var btn = document.getElementById("btn");  btn.onclick = function() {  alert("按钮被点击了！");  }  </script>  这段代码展示了如何为一个按钮绑定点击事件。当按钮被点击时，会弹出一个提示框。绑定事件的方式非常直观，直接通过`onclick`属性将函数赋值给事件处理程序。  4.事件冒泡的应用  在复杂的网页中，利用事件冒泡机制，可以在父元素上统一管理子元素的事件处理，而不需要为每个子元素单独添加事件。这不仅减少了代码的冗余，还提高了性能。  二、JavaScript继承  JavaScript中的继承是面向对象编程中的一个重要概念。主要通过\*\*原型链\*\*实现。  1.原型链  概念：每个构造函数都有一个原型对象，这个原型对象又指向另一个构造函数的原型对象，形成了一条链。实例对象可以通过这条链继承属性和方法。  示例：  function Animal() {  this.type = "animal";  }  Animal.prototype.getType = function() {  console.log(this.type);  }  function Dog() {  this.name = "dog";  }  Dog.prototype = new Animal();  var myDog = new Dog();  myDog.getType(); // 输出 "animal"  `Dog`继承了`Animal`的属性和方法。当调用`myDog.getType()`时，会输出`Animal`中的`type`属性。  2.组合继承  概念：结合原型链和构造函数两者的优点，使用原型链继承方法，使用构造函数继承属性。  function Animal(name) {  this.name = name;  }  Animal.prototype.sayName = function() {  console.log(this.name);  }  function Dog(name, age) {  Animal.call(this, name);  this.age = age;  }  Dog.prototype = new Animal();  Dog.prototype.sayAge = function() {  console.log(this.age);  }  var myDog = new Dog("Buddy", 5);  myDog.sayName(); // 输出 "Buddy"  myDog.sayAge(); // 输出 "5"  `Dog`不仅继承了`Animal`的属性`name`，还添加了自己的属性`age`和方法`sayAge`。这种继承方式能够最大化代码复用，并确保每个实例都拥有独立的属性。  3.继承的注意事项  当使用原型链继承时，需要注意所有实例共享同一个原型对象。这意味着一个实例的改动会影响其他实例。因此，在构造函数中定义实例属性，而在原型中定义共享方法是更合理的选择。  日期2024.8.21 |

Day28笔记

|  |
| --- |
| BOM  1. 什么是BOM  BOM（浏览器对象模型）是用于控制浏览器窗口及其各项功能的JavaScript API集合。它包含了一系列对象，能够访问和操作浏览器的功能，如窗口的大小、位置、导航历史、用户屏幕信息以及网络请求等。  alert("Hello, World!"); // 弹出提示框  var width = window.innerWidth; // 获取窗口的内部宽度  var location = window.location.href; // 获取当前页面的URL  BOM的核心是`window`对象，它在浏览器中既是全局对象，又是浏览器窗口的接口。通过BOM，我们可以实现许多与浏览器交互的操作，如弹出窗口、页面重定向、获取屏幕分辨率等。BOM虽然功能强大，但由于其各浏览器实现存在差异性，使用时需注意兼容性问题。  2. 常见的BOM对象与方法  2.1 window 对象  `window`对象是BOM的核心，表示浏览器窗口或框架。它提供了浏览器窗口相关的属性和方法。  console.log(window.name); // 获取窗口的名称  window.open('https://www.example.com', '\_blank'); // 在新窗口打开链接  通过`window`对象，我们可以获取和操作浏览器窗口的属性，如名称、大小、位置等。`window`对象还与ECMAScript中的全局对象紧密关联，因此全局变量和函数也是`window`对象的属性和方法。  2.2 navigator 对象  `navigator`对象包含了浏览器的信息，如应用程序的名称、版本号、用户代理等。  console.log(navigator.userAgent); // 输出浏览器的用户代理字符串  `navigator`对象让我们能够检测浏览器的信息，对于实现不同浏览器的兼容性处理非常有用。特别是在处理用户代理（user agent）时，可以根据不同浏览器的特性加载不同的脚本或样式。  2.3 location 对象  `location`对象提供了当前文档的位置信息（URL），并允许重定向页面。  console.log(location.href); // 输出当前页面的URL  location.href = "https://www.newpage.com"; // 重定向到新的URL  `location`对象是处理页面导航的核心对象，通过它可以轻松获取当前URL信息并执行页面跳转操作。它在实现动态导航和单页应用程序（SPA）的路由管理时尤为重要。  2.4 screen 对象  `screen`对象包含有关用户屏幕的信息，如分辨率、颜色深度等。  console.log(screen.width); // 获取屏幕的宽度  console.log(screen.height); // 获取屏幕的高度  `screen`对象主要用于处理与用户设备屏幕相关的操作，例如根据屏幕分辨率调整页面布局。虽然它的使用场景较为有限，但在响应式设计中非常实用。  2.5 history 对象  `history`对象用于操作浏览器的会话历史，可以实现页面的前进、后退功能。  history.back(); // 返回前一个页面  history.forward(); // 前进到下一个页面  `history`对象的前进、后退功能使得我们能够更好地控制用户的浏览体验，特别是在单页应用中，通过`pushState`和`replaceState`等方法还能实现浏览器地址栏的动态更新而不刷新页面。  3. 兼容性与BOM的注意事项  虽然BOM提供了强大的功能，但由于各浏览器厂商在实现上存在差异，部分属性和方法在不同浏览器中的支持情况并不完全一致。  在使用BOM时，应始终考虑到兼容性问题。可以通过检测特性是否存在来确保代码的健壮性，例如使用`if`语句检测`window.screen`对象是否存在。了解并应对不同浏览器之间的差异，对于编写跨平台兼容性好的代码至关重要。  通过这次学习，我对BOM的功能和应用有了更深入的理解。BOM在JavaScript编程中提供了非常重要的功能，使得我们能够更好地与浏览器交互。在实际开发中，合理使用BOM能够极大提升用户体验，但也需要注意浏览器兼容性问题，并在必要时进行相应的处理。  日期2024.8.22 |

Day29笔记

|  |
| --- |
| AJAX学习笔记  1. 什么是AJAX？  AJAX（Asynchronous JavaScript And XML）是一种用于创建交互式网页应用的开发技术，旨在通过异步数据传输（HTTP请求）来增强网页与服务器之间的数据交换。其特点是能够在不重新加载整个页面的情况下，对页面的部分内容进行更新。  AJAX技术是现代Web开发中的核心部分。它的异步特性使得用户能够在不感知页面刷新的情况下进行操作，提高了用户体验。尽管名称中包含XML，但实际应用中JSON更为普遍，因为它更轻量且易于解析。  2. HTTP请求与响应  HTTP（超文本传输协议）是Web浏览器与服务器之间通信的基础。AJAX通过JavaScript对HTTP的请求和响应进行操作，从而实现数据的动态加载。  3. XMLHttpRequest对象  `XMLHttpRequest`对象是实现AJAX功能的关键，它用于在后台与服务器交换数据。常用的方法包括：  `open(method, url, async)`：设置请求类型（如GET或POST）、目标URL以及是否异步。  `setRequestHeader(header, value)`：设置HTTP请求头。  `send(data)`：发送请求数据。  var request = new XMLHttpRequest();  request.open('GET', 'https://api.example.com/data', true);  request.send();  request.onreadystatechange = function() {  if (request.readyState === 4 && request.status === 200) {  console.log(request.responseText);  }  };  `XMLHttpRequest`提供了一个简洁的接口用于与服务器进行通信，尤其是在处理异步请求时，它避免了页面的阻塞。然而，随着Fetch API的引入，`XMLHttpRequest`的使用逐渐减少，但它仍然是理解AJAX的基础。  4. AJAX的封装  为了简化AJAX的使用，通常会对其进行封装。  var myAjax = {  get: function(url, params, success, error) {  var httpRequest = new XMLHttpRequest();  httpRequest.open('GET', url + "?" + new URLSearchParams(params).toString(), true);  httpRequest.send();  httpRequest.onreadystatechange = function() {  if (httpRequest.readyState === 4 && httpRequest.status === 200) {  success(JSON.parse(httpRequest.responseText));  } else if (httpRequest.readyState === 4) {  error(httpRequest.responseText);  }  };  }  };  封装AJAX有助于提高代码的可复用性和可维护性。通过封装，我们可以统一处理请求的细节，减少冗余代码，同时也可以更容易地应对跨浏览器的兼容性问题。  5. 实战操作：GET与POST请求  GET请求  var request = new XMLHttpRequest();  request.open('GET', 'https://api.example.com/data?param=value', true);  request.send();  POST请求  var request = new XMLHttpRequest();  request.open('POST', 'https://api.example.com/data', true);  request.setRequestHeader('Content-type', 'application/json;charset=UTF-8');  request.send(JSON.stringify({key: 'value'}));  GET请求适用于获取数据且不会改变服务器状态的操作，POST则更适合提交数据。理解两者的区别并合理选择，可以提高请求的效率与安全性。  总结  AJAX通过异步数据传输大大提升了Web应用的交互性，理解并熟练掌握其基本原理和常用操作是前端开发的重要技能。通过对`XMLHttpRequest`的深入理解和灵活应用，可以实现更加高效和友好的用户体验。  日期2024.8.23 |