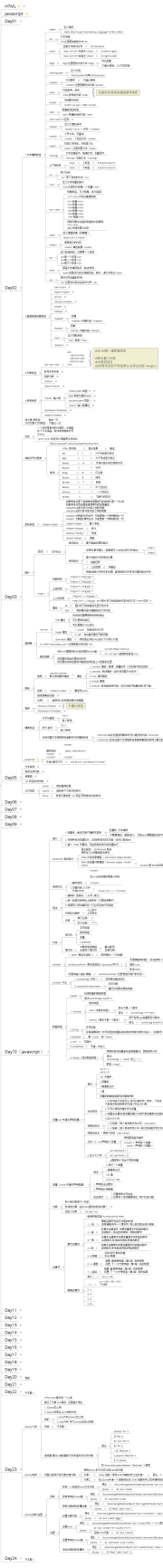
柠檬学院1909全栈工程师



1. 一阶段

1.1. HTML

HTML 指的是超文本标记语言: HyperText Markup Language（超出文本本身，不仅限于文本

）

HTML 不是一种编程语言，而是一种标记语言

标记语言是一套标记标签 (markup tag)

HTML 使用标记标签来描述网页

HTML 文档包含了HTML 标签及文本内容

HTML文档也叫做 web 页面

1.1.1. HTML 实例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
```

<h1>我的第一个标题</h1>

** 我的第一个段落。 **

</body>
</html>

1.1.2. 实例解析

<!DOCTYPE html> 声明为 **HTML5** 文档
<html> 元素是 **HTML** 页面的根元素
<head> 元素包含了文档的元（**meta**）数据，如 **<meta charset="utf-8">**
定义网页编码格式为 **utf-8**。
<title> 元素描述了文档的标题
<body> 元素包含了可见的页面内容
<h1> 元素定义一个大标题
** ** 元素定义一个段落
注：在浏览器的页面上使用键盘上的 **F12**
按键开启调试模式，就可以看到组成标签。

1.2. Javascript

为什么学习 JavaScript?

JavaScript web 开发人员必须学习的 3 门语言中的一门:

HTML 定义了网页的内容
CSS 描述了网页的布局
JavaScript 网页的行为

1.3. Day01

1.3.1. 学习方式？

1.坚持看笔记

2.坚持做作业

3.坚持看手册

4.坚持做预习

5.坚持下去

1.3.2. 前端是什么？

•前端开发工程师是一个很新的职业，在国内乃至国际上真正开始受到重视的时间是从2005年开始的，是指Web前端开发工程师的简称。

Web前端开发是从美工演变而来的，名称上有很明显的时代特征。在互联网的演化进程中，Web

1.0时代，网站的主要内容都是静态的，用户使用网站的行为也以浏览为主。如2005年以后，互联网进入Web

2.0时代，各种类似桌面软件的Web应用大量涌现，网站的前端由此发生了翻天覆地的变化网页不再只是承载单一的文字和图片，各种富媒体让网页的内容更加生动，网页上软件化的交互形式为用户提供了更好的使用体验，这些都是基于前端技术实现的。目前web前端工程师的年薪待遇平均在10万以上，高级HTML前端工程师年薪达30—

50万，很多企业对于与web前端相关的技术职位更是求贤若渴。

•前端工程师，也叫Web前端开发工程师。他是随着web发展，细分出来的行业。Web前端开发技术主要包括三个要素：HTML、CSS和JavaScript!HTML甚至不是一门语言，仅仅是简单的标记语言!CSS只是无类型的样式修饰语言。当然可以勉强算作弱类型语言。Javascript的基础部分相对来说不难，入手还算快。

•前端开发的入门门槛很低，与服务器端语言先慢后快的学习曲线相比，前端开发的学习曲线是先快后慢。也正因为如此，前端开发领域有很多自学成“才”的同行，但大多数人都停留在会用的阶段，因为后面的学习曲线越来越陡峭，每前进一步都很难。人们常说：不想当裁缝的司机，不是个好厨师。如果单纯只是学习前端编程语言、而不懂后端编程语言(PHP、ASP.NET，JSP、Python)，也不能算作是优秀的前端工程师。在成为一个优秀的前端工程师的道路上，充满了汗水和辛劳。

1.3.3. 前端开发软件环境安装？

 # 代码编辑器

1. hbuilder

https://www.dcloud.io/hbuilderx.html

2. vscode

https://code.visualstudio.com/

3. sublimetext

http://www.sublimetext.com/

4. atom

https://atom.io

5. WebStorm

https://www.jetbrains.com/

1.4. Day02

1.4.1. 1.文本辅助标签

<abbr>

定义缩写

<abbr title="Hyper Text Markup Language">HTML</abbr>

<b>

文本加粗

<b>这里是加粗的文本**<b>**

<bdo>

控制文本排列方向

dir (direction)

<bdo dir="ltr">左到右**</bdo>**

ltr (left to right)

<bod dir="rtl">右到左**</bdo>**

rtl (right to left)

<big>

<big>这里是放大的文本**</big>**

可以嵌套

不建议使用，CSS不好控制

<blockquote>

定义引用

<blockquote>引用</blockquote>

** **

文本居中

不建议使用

** 这里是居中的内容 **

<cite>

内容参考，斜体

<cite>参考的内容</cite>

这里的内容用来描述参考信息(内容参考，斜体, <cite>参考的内容</cite>)

<code>

代码展示标签

<code> var app = 1000; </code>

<del>

带删除线的标签

<del> 带删除线的内容 </del>

<dfn><i>区别

<dfn>

专业术语展示标签，外观斜体

<dfn> 宕机 **</dfn>**

****

强调内容，斜体效果

**** 强调的内容 ****

****

外观斜体文本

**** 这里是斜体的文本 ****

1、 把文本定义为强调的内容

****
标签告诉浏览器把其中的文本表示为强调的内容。对于所有浏览器来说，这意味着要把这段文字用斜体来显示。

尽管现在 ****
标签修饰的内容都是用斜体字来显示，但这些内容也具有更广泛的含义，将来的某一天，浏览器也可能会使用其他的特殊效果来显示强调的文本。如果你只想使用斜体字来显示文本的话，请使用 ****
标签。除此之外，文档中还可以包括用来改变文本显示的级联样式定义。

2、显示斜体文本效果

**** 标签和基于内容的样式标签 ****

类似。它告诉浏览器将包含其中的文本以斜体字（italic）或者倾斜（oblique）字体显示。如果这种斜体字对该浏览器不可用的话，可以使用高亮、反白或加下划线等样式。

3、<dfn>定义一个定义项目

<dfn> 标签可标记那些对特殊术语或短语的定义。

现在流行的浏览器通常用斜体来显示 <dfn> 中的文本。将来，<dfn> 还可能有助于创建文档的索引或术语表。

与其他许多基于内容的样式和物理样式标签一样，<dfn> 标签尽量少用为妙。

也就是说它们要实现的目的不同，但都用同样的表现方式，就是斜体(<dfn>, , <i>)

<samp>

定义计算机样本

<samp> var q1 = "你好"; </samp>

<small>

小号文本，可叠加

<small> 小号的文本 </small>

万能文本标签，没有语义化

**** 这里的是文本 ****

****

文本加粗显示，强调文本，加重语气

**** 加粗文本 ****

上下标标签

<sup>

上标签

5²m

<sub>

下标签

5₂m

<u>

带下划线

<u> 带下划线的文本 **</u>**

<var>

定义文本变量的部分

<var>这里的内容是一个变量**</var>**

<h1>-<h6>

标题标签，主次标题，依次递减

<h1>-<h3>不可以随便使用

<h1>标题</h1>

<h2>标题</h2>

<h3>标题</h3>

<h4>标题</h4>

<h5>标题</h5>

<h6>标题</h6>

搜索引擎会去查找标题的内容展现

<h1>-<h3>

会以关键字展示出来

<kbd>

定义键盘按键（快捷键）

<kbd> Ctrl + C </kbd>

<mark>

高亮显示的内容

<mark> 黄色背景 **</mark>**

** **

定义段落标签，这里是一个段落

** **第一个段落** **

** **第二个段落** **

** **第三个段落** **

<pre>

保留文本编写格式，贴边使用

<pre>这里的内容会保留空格，换行，源文本格式</pre>

<q>

带双引号包裹的引用

<q> 这里的内容会包含双引号 </q>

1.4.2. 2.普通排版容器标签

 这是一个快容器 没有语义化

 这是一个行内容器 没有语义化

<p><p>

<p> 段落标签，块标签 <p>

<article></article>

定义文章

<article>文章内容</article>

<aside>

定义页面之外的内容，比如侧边栏

<aside>菜单</aside>

<dialog>

对话框

默认有定位居中，隐藏的

需要手动设置显示display

<dialog> 一个对话框 </dialog>

<header>

页眉

<header> 页眉内容 </header>

<footer>

页脚

<footer> 页脚内容 </footer>

<nav>

定义导航标签

<nav> 百度 </nav>

<section>

文档中的区段

<section>区段标签<section>

1.4.3. 3.列表标签

<dl> <dt> <dd>

<dl>

<dt>XXX</dt>

<dd>XXX</dd>

<dt>XXX</dt>

</dl>

dl dt dd是一套配套标签

dl表示整个列表

dt列表的项目

dd列表项目的子项目(默认会有左边距 margin)(<dl>

<dt>XXX</dt>

<dd>XXX</dd>

<dt>XXX</dt>

</dl>)

有序无序列表

有序列表

项目1

项目2

项目3

无序列表

项目1

项目2

项目3

在列表里可互相嵌套多个列表

它们都需要配合li标签来显示li表示列表的子项目(有序列表
,

无序列表
)

目录列表

<dir>

目录列表

不建议使用

<dir>

目录列表1

目录列表2

目录列表3

</dir>

比无序列表多了些 margin (<dir>

目录列表

不建议使用)

<menu>

<menu>

定义命令的列表或菜单

<menu-item>

定义用户可以从弹出菜单调用的命令/菜单项目

```
<menu>
        <menuitem>1</menuitem>
        <menuitem>2</menuitem>
        <menuitem>3</menuitem>
</menu>
```

1.4.4. 4.表单标签

<form></form>

表单标记

成套使用

一个

<form>

是一套表单

```
  <!-- 单选功能 -->
    <form>
      <label for="sex1">男</label>
      <input checked value="男" type="radio" name="性别" id="sex1">
      <label for="sex2">女</label>
      <input value="女" type="radio" name="性别" id="sex2">
      <button type="submit">单选功能提交按钮</button>
    </form>
```

<input> 输入框

<input type="类型" />

1. text 文本输入框

2. password 密码类型

3. file 文件上传

4. radio 单选按钮

必须指定一个name属性且一致

checked属性 在radio情况下 默认选中

5. checkbox 多选按钮

必须指定一个name属性且一致

必须指定一个value属性（代表输入框的值）

6. email 邮箱

手机浏览器里 完全支持 会弹出邮箱键盘 带@符号

7. date 日期

支持性不是很好 尽量不用

8. number 数字类型

手机浏览器里 完全支持 会弹出数字键盘 有小数点

9. range 范围输入

支持性不是很好 尽量不用

10. color 颜色输入

选择输入颜色 大部分浏览器不支持 尽量不用

11. button 按钮

设置一个按钮　定义类型　必须设置 value　否则没有内容

<input type="button" value=" 普通按钮">

12. submit 提交按钮

设置一个提交按钮　定义类型　必须设置 value　否则没有内容

<input type="submit" value="提交按钮" />

13. reset 重置按钮

设置一个重置按钮　定义类型　必须设置 value　否则没有内容

input type="reset" value="重置" />

label 表单元素的label

1. for 用于关联 表单元素的id属性 当点击它的时候
会自动给表单元素添加焦点（focus）状态

```
<!-- 单选按钮 -->
  <form>
    <label for="sex1">男</label>
    <input checked value="男" type="radio" name="性别" id="sex1">
    <label for="sex2">女</label>
    <input value="女" type="radio" name="性别" id="sex2">
    <button type="submit">单选功能提交按钮</button>
  </form>
```

<button type="类型">

<button type="button">普通按钮</button>

<button type="submit">提交按钮</button>

<button type="reset">重置表单</button>

<input> 输入框属性

1. id 每一个标签都具有的属性 给标签编号

2. checked 在radio的情况下 默认选中

3. name

a.表单的字段名称 提交表单时 浏览器采集数据的字段名称

b.只有设置了 name 属性的表单元素才能在提交表单时传递它们的值

4. value 属性 代表输入框的值/input 元素的值

<textarea></textarea>

文本域

多行文本输入框

cols="30”

占多少列

rows="10"

占多少行

<textarea name="res" id="re" cols="30" rows="10">< , textarea >

<select></select>

下拉列表

单地区选择

```
<label for="city">地区：</label>
  <select name="city" id="city">

  <option value="北京" selected>北京
  </option>

  <option value="上海" >上海</option>
</select>
```

分组地区选择

```
<div>
    <label for="city-group">分组地区</label>
    <select name="citys" id="city-group">
      <optgroup label="北京">
        <option value="东城区">东城区</option>
        <option value="西城区">西城区</option>
        <option value="海淀区">海淀区</option>
      </optgroup>

      <optgroup label="重庆">
        <option value="东城区">江北区</option>
        <option value="西城区">渝中区</option>
        <option value="海淀区">渝北区</option>
      </optgroup>
    </select>
</div>
```

selected 是默认选中一个元素

下拉列表是由select 标签+ option组 合在一起的

1.4.5. 块元素/块标签　　独占一行

行内元素/行内标签　不独占一行

1.5. Day03

1.5.1. 空格

一个空格是单词的分隔符，会保留

多个不会保留，除非使用特殊符号

** **

<pre></pre> 标签可以保留原文本格式

1.5.2. 特殊符号对照表

https://www.jb51.net/onlineread/htmlchar.htm

常用

HTML 原代码	显示结果	描述
<	<	小于号或显示标记

>

>

大于号或显示标记

&

&

可用于显示其它特殊字符

"

"

引号

®

®

已注册

©

©

版权

™

™

商标

** **

半个空白位

** **

一个空白位

** **

不断行的空白

1.5.3. 表格标签

<tabel></tabel>

如果表格设置了宽度表格里面的列的宽度只是一个比例

如果表格没有设置宽度按照列的宽度叠加

cellpadding单元格与内容之间的间距

cellspacing单元格与单元格之间的间距

rowspan 行跨越 纵向合并 只能是前一行跨域到后一行

colspan 列跨越 横向合并 只能是前一列跨域到后一列

<tabel></tabel>

整个表格

<thead></thead>

表头

<tbody></tbody>

表体

<tfoot></tfoot>

表脚

1.5.4. 图片

路径

文件协议

绝对路径

基于磁盘位置的路径

相对路径

如果在服务器上（就是基于 http协议的文件地址）

http://

https://

基于当前文件的相对位置

./ 当前目录

../ 上级目录

可叠加

根据当前文件所在的位置 查找目标文件所在位置的路径关系

<img />

当前目录

****

****

上级目录

****

****

上上级目录

****

****

属性

alt

图片找不到或者丢失显示的文字

title

鼠标悬浮显示编辑好的文本内容

1.5.5. 超链接

<a>

href 属性

告诉浏览器要跳转的目标地址

可以是相对路径

可以是基于http协议

target 属性

1. _blank　在新选项卡打开

2. self　在当前页面打开新页面

download 属性

网页是必须在http协议下才可以下载

这里是显示的内容

锚点链接

#在html里面被称为(路径里的)hash值

test

<h1 id="test">被跳转的标签

浏览器在地址栏里发现它时
浏览器会自动匹配到#值指定的标签上id匹配的位置

1.5.6. 媒体

音频

<audio></audio>
默认使用脚本播放

属性

1. controls 播放、暂停、音量控件（交给用户自己控制）

2. autoplay 自动播放（部分浏览器不会生效）

3. loop 循环播放

4. muted 静音

5. preload 预加载音频内容（在还没有开始播放前 预下载）

视频

<video></video>

属性同audio

音视频兼容处理

<source />

音频

<source src="./hello.mp3" />

<source src="./hello.wav" />

<source src="./hello.ogg" />

视频

<source src="./hello.mpg" />

注明：

音频可以放视频但是没有画面

1.5.7. 框架

<iframe></iframe>

属性

1. src

嵌入网页的地址

嵌入本地网页地址

width

宽

2. height

高

<iframe width="300" height="300" src="http://www.baidu.com"></iframe>

<frameset></frameset>

配合 <frame></frame> 标签使用

src

嵌入网页的地址

cols="number,number,number"

网页的百分比列

cols指定每一个页面占多少列每一列的宽度用百分比逗号隔开

不建议使用(<iframe></iframe>)

1.5.8. 辅助标签

水平分割线

<hr />

很少使用

折行/换行

**
**

很少使用

当浏览器不支持框架或者脚本时的辅助标签

<noscript>当浏览器禁用脚本时可以看见的内容</noscript>

<noframes>当浏览器不支持框架或者框架集的时候可以看见的内容</noframes>

1.5.9. Javascript

HTML 定义了网页的内容
CSS 描述了网页的布局
JavaScript 网页的行为

编写标签
<script>
</script>

<script>

alert(" Hello World ")

</script>

外链js脚本文件

<script src="./js/index.js"></script>

1.6. Day05

1.6.1. 文本装饰

text-decoration

1. underline 下划线

2. line-through 横穿线

3. overline 上划线

4. blink 指定文字的装饰是闪烁的（兼容性不多）

5. none 无装饰

1.6.2. 有序无序列表

列表样式可以直接使用在 **ul** 和 **ol** 上
都有同等效果

设置在 **ul** 上所有的 **li** 都应用这个效果
单个 **li** 只会应用指定的元素

list-style-type

设置列表样式

1. disc 实心圆

2. circle 空心圆

3. square 是实心方块

4. decimal 阿拉伯数字

5. lower- roman 小写罗马数字**(CSS1)**

6. upper-roman 大写罗马数字**(CSS1)**

7. lower-alpha 小写英文字母**(CSS1)**

8. upper-alpha 大写英文字母**(CSS1)**

9. none 不使用项目符号**(CSS1)**

10. armenian 传统的亚美尼亚数字**(CSS2)**

11. cjk-ideographic 浅白的表意数字**(CSS2)**

12. cjk-ideographic 浅白的表意数字**(CSS2)**

13. georgian 传统的乔治数字**(CSS2)**

14. lower-greek 基本的希腊小写字母**(CSS2)**

15. hebrew 传统的希伯莱数字**(CSS2)**

16. hiragana 日文平假名字符**(CSS2)**

17. hiragana-iroha 日文平假名序号**(CSS2)**

18. katakana 日文片假名字符**(CSS2)**

19. katakana-iroha 文片假名序号**(CSS2)**

20. lower-latin 小写拉丁字母**(CSS2)**

21. upper-latin 大写拉丁字母**(CSS2)**

list-style-position

1. outside 排序的符号在内容外面

2. inside 排序的符号包含在内容里面

list-style-image

使用图片的列表样式

样式和内容不同颜色写两套样式
样式文本各一套

1.6.3. 背景图

background-image: url("./imgr/1.jpg")

url() 指定图片的地址

url 里的引号加了有代码提示
不加没有
两者用途没区别

background-repeat: ;

1. repeat 平铺

2. no-repeat 不平铺

3. repeat-x 允许横向平铺

4. repeat-y 纵向平铺

backgrou-position: ;

background-position-x

1. left 左对齐

2. center 居中对齐

3. right 右对齐

4. length 数值+px

正值向右

负值向左

background-position-y

1. top 上对齐

2. center 居中对齐

3. bottom 下对齐

4. length 数值+px

正值向下

负值像上

background-attachment: ;

1. fixed 固定

2. scroll 跟着滚动

background-size: ;

默认：auto

1. cover 完全缩放到容器，可能会超出容器

2. contain 等比例缩放

宽度或者高度达到条件停止

3. 百分比

1.6.4. <a> 标签私有伪类

使用冒号隔开

只用在 <a> 标签

a:visited

设置访问过的样式

a:link

设置未访问的样式

1.6.5. 公共伪类

:hover

鼠标悬停效果

:active

鼠标按下不放开的样式

:focus

表单元素或者 <a> 标签 获取焦点时的样式

1.7. Day06

1.7.1. 伪类补充

:nth-child

选择指定的元素

添加样式

odd 奇数

even 偶数

number 数字

1.7.2. 伪元素

必须加 **content**

::before

在元素之前加入内容
图片，文本等等

```
.box::before {
 content: " 之前 ";
}
```

```
图片 ::befor {
  content: "  url(" ./img/jpg ")  ";
}
```

::after

在元素之后加入内容
图片，文本等等

```
.box::after {
 content: " 之后 ";
}
```

1.7.3. display

text-align 可以控制 inline 和 inline-block元素 对齐

如果想要使用行高来控制**inline - block**的垂直对齐

子元素必须单独设置行高 否则会继承父元素

1. inline 行内元素

不占整行

有盒模型

只是不可以设置内容区域的宽高

2. block 块元素

占一行

可以设置内容区域的宽高

3. inline-block 行快

不占一行

可以设置内容区域的宽高

默认会有一个间距（要去掉它必须在父元素使用**font-size**来设置）

4. flex 弹性布局

5. table 表格

不常用

6. none 无/隐藏

1.7.4. 属性选择器

联合选择器

复合选择器

div.aa.cc { background-color: red; }

div[class] { background:blue; }

div[class="aa bb"] { background:blue; }

必须是 div 标签 且具有class样式属性值为 aa 和 bb

div[class~=" aa "]

必须是div标签 且里面包含了用空格隔开的aa属性值即可满足条件

即可满足条件

div[class|="ccc"]

必须是div标签 class样式里包含了以 ccc- 开头的属性值 即可满足条件

div[class^="ee"]

只要class样式里包含 ee 开头的都可以被选中

div[class$="end"]

只要class样式里包含 end 结束的都可以被选中

div[class*="c"]

只要class样式里包含 c 的都可以被选中

1.7.5. css样式的优先级

1. 默认样式（浏览器自带的）

2. 通配选择符

3. 标签选择符

4. class选择符

5. id选择符

6. !important

7. 通配选择符 + !important

8. 标签选择符 + !important

9. calss选择符 + !important

10. id选择符 + !important

1.7.6. 盒模型

除了 **head** 以外的每个元素都有一个盒模型

盒模型 元素的组成部分（元素的大小是如何计算的）

1. 内容区域（content）

固定元素的宽高

min-width

最小宽度

min-height

最小高度

max-width

最大宽度

max-height

最大高度

2. 内部区域（padding）

元素变大了 内容区域未变

3. 边框区域（border）

元素变大

4. 元素区域（margin）

不会改变元素的大小 会改变当前元素和其它元素之间的距离

盒模型设置方向

margin

padding

border

width-

style-

color-

顺时针设置(margin, padding, border)

上右下左

margin塌陷

子元素的margin-top会影响到父元素top

1. 添加border 可以解决

2. overflow: hidden

溢出隐藏

inherit 继承父元素

1.8. Day07

1.8.1. 浮动塌陷

伪元素在之后加入 **clear**

1.8.2. 溢出处理

overflow: hidden;

1. hidden 隐藏

2. visible 显示

3. auto 根据内容

横向超出 横向显示滚动条

纵向超出 纵向显示滚动条

4. scroll 滚动条 始终显示滚动条

overflow-x: ;

横向溢出处理

overflow-y: ;

纵向溢出处理

1.8.3. 隐藏显示元素

display: none;

隐藏之后 从文档流种去掉

只是不显示（浏览器跳过渲染了）

visibility

1. visible 在文档流中显示

2. hidden 在文档流总隐藏 保留位置

浏览器没有完全跳过（把位置渲染出来了）

1.8.4. 定位

1. position 静态

静态 没有定位的元素 默认使用的是**static** 没有层

2. relative 相对定位

（当前文档流所在位置相对位置 以子级所在位置 为中心移动）

所占的文档流位置固定不变（ 显示的位置 可以移动）（保留占位）

3. fixed 固定定位

是根据浏览器的四周可视区域来进行计算的 **(**完全脱离文档流**)**

如果不是设置 **top left right bottom**
定位默认从他本身的文档流位置开始显示（不占位）

4. absolute 绝对定位

如果不是设置 **top left right bottom**

定位默认从他本身的文档流位置开始显示

默认计算是按照浏览器可视区域来计算的

计算需要一个参照物（默认是浏览器）

绝对定位 的参照物 可以是除了 **static**定位 以外的所有定位的 祖先元素

任意一个祖先元素上有定位 （除了**static**定位以外的所有定位） 都可以被 **absolute**定位 当作参照物

默认参照（向上查找） 第一个有定位的祖先

定位的子属性

1. top

2. right

3. bottom

4. left

如果上下存在取上

如果左右存在取左

根据坐标轴

1.8.5. :first-child 选择第一个元素

1.8.6. :last-child 选择最后一个元素

1.8.7. z-index（层）

z-index: ;

取值范围 -999 0 999

数值越大 层级越高

定位层级

1. 定位元素比不定位的元素层级高

2. 同级后写定位的元素比先写的高

3. 没有定位的元素 始终都比不过定位元素（absolute fixed relative）

1.8.8. flex（弹性布局）

1. 在ie10 一下的浏览器不能使用

2. flex 多使用在移动端上（9.0）

本身也是一个块元素
控制子元素如何排列

flex-direction（定义主轴的方向）

1. row

   控制子元素 横向排列（主轴是横向）

2. column

   控制子元素 纵向排列（主轴是纵向）

3. row-reverse

3. column-revers

镜像(3. row-reverse, 3. column-revers)

控制主侧轴的排列方式

1. flex-start

沿着主轴开始的位置排列

2. flex-end

沿着主轴结束的位置排列

3. space-between

用剩余的位置 均匀分配到每个元素之间

项目位于各行之间留有空白的容器内

4. space-around

项目位于各行之前、之间、之后都留有空白的容器内

5. align-items

flex-start 侧轴开始位置显示

center 侧轴水平位置显示

flex-end 侧轴结束位置显示

1.9. Day08

1.9.1. flex

1. flex-grow: n;

定义扩展比列（把主轴剩余的距离 按比列分配） 弹性扩展

2. flex-shrink: n;

定义压缩比列（当所有的子元素 加起来超过了）

3. flex-baseise

align-self: ;

控制当前元素的位置

1.9.2. 尺寸

1. PC

比列

4：3

16：9

1080 / 9 * 16 = 1920

分辨率

1280 * 720

1366 * 768

1200+ PC

1200- pad

768- mobile

2. mobile

手机分辨率

1334 * 750（手机分辨率不一定是浏览器的分辨率）

4.7 英寸的手机 **1920*1080**的 **1334*750**

手机浏览器的分辨率是根据 一英寸转换成的像素

1.9.3. meta属性

initial-scale=1.0, initial-scale=1.0,

maximum-scale=1.0, 最大缩放比例

user-scalable=0 是否允许双指缩放

1.10. Day09

1.10.1. 圆角

border-radius

值

1. 百分率
2. **PX**

方向

左上角

左下角

右上角

右下角

1.10.2. 阴影

box-shadow: 0px 0px 0px 0px red;

box-shadow: 水平 垂直 模糊度 大小 颜色 扩散方向（**inset outset**）

水平 正数时 向右移动 负数像左

垂直 正数时 向下移动 负数向上移动

1.10.3. 渐变

线性渐变

background-image: liner-gradient(to bottom,red,blue)

liner-gradient（**direction || angle**）

1. direction： to（上右下左）

2. angle deg（角度单位）

径向渐变

background-image（）

1

1.10.4. 过渡

动 **=>** 变化（位置的移动 颜色变化）

时间 **1s 1ms** （秒**/s** 毫秒**ms**）

过渡 自动计算**a-b**之间的变化 配合时间 形成动画

transition 过度

transition-property: ;

设置哪一个属性变化过度属性

支持多个属性（all）

transition-duration: ;

持续时间

transition-delay: ;

等待时间

transition-timing-functio: ;

运动曲线

liner

ease

transition 简写

除了时间

第一个是持续时间

第二个是等待时间

transition: linear all 5s 2s;

1.10.5. 转换

transform: ;

以自身为中心向四周处理

1. translate()

平移

（X,Y）

2. rotate()

旋转 俯视（2d 情况下只能看到俯视的效果 Z轴）

3. skew()

扭曲

4. scale()

缩放

（数值，比列 没有单位）

放大镜效果

1.10.6. 3d转换

创建一个三维空间在父级上设置

transform-style: preserve-3d;

1.10.7. 透视

透视距离

perspective: 100px;

1.10.8. 动画

啊

1.11. Day10（Javascript）

1.11.1. 简介

1. 轻量级、直译式客户端脚本语言

轻量级 文本编写

不需要编译，直接运行， 列如Java 需要编译成字节码文件

2. 可被所有浏览器执行，实现网页的动态功能（也可以做后台）

3. 插入 HTML 页面后，可由所有的现代浏览器执行

1.11.2. Javascript 组成部分

语法规范： ECMAScript 标准

用来定义js的基础语法规范

DOM 文档对象模型 （Document Object Model）

BOM 浏览器对象模型（browser object model）

widow

window 是 BOM的顶层对象

由基础语言创建出来的对象(DOM 文档对象模型 （Document Object Model）,

BOM 浏览器对象模型（browser object model）

widow)

1.11.3. 使用方式

1. 脚本结构

<script>

在script标签里的就是JS代码

</script>

2. 页面内嵌 **JS** 文件

外部**JS**文本

<script src=" ./... "></script>

1.11.4. 语法

1. 编写时 结束符 **;**（分号 英文）

2. 每一给语句结束加上结束符（不是轻质要求）

3. 如果两个语句编写在一行必须添加分号结束

代码执行顺序

从上到下

从左到右

注释

单行注释： **//**

多行注释： **/** ** */**

1.11.5. 内存

栈内存

字符类型

数字类型

变量

undefined

堆内存

对象类型的数据

会被存入堆内存

解决复用

互相引用

1.11.6. window

1. alert("弹出对话框");

网页弹出一个对话框

window.confirm("是否选择加入Javascript 学习！ ")

页面弹窗询问框 （系统自带）两个按钮 确定取消

确定 true

取消 false

页面待输入框的 弹窗

window.prompt("这里是提示用户的内容")

1.11.7. console 方法

1. console.log("你好")

在控制台输出日志

2. console.dir(document)

打印对象

遍历出对象的原始信息

1.11.8. 数据类型

typeof

检测变量的数据类型

语法**console.log(typeof 1)**

1. Number

数字类型

NaN（非数字类型）

表示不是一个数字

语法

```
console.log( 1 === Nan )
```

isNanN（是否不是一个数字）

用于检测**iyge**变量是否为数字

语法

```
consol.log( isNaN(1) )
```

2. String

字符类型

在做减乘除时 字符类型的变量会自动隐式转换为数字 除去 **+**（拼接符）

3. Boolean

真（**true**）**/** 假（**false**）

4. Null

空指针

5. Undefined

无值，未定义

6. Object（混合数据类型）

使用花括号包裹起来且是键值对，使用逗号分开

语法:

```
console.log( {  name: "张三"; } )
```

数组 **Array()**

1.11.9. 变量(var 关键字声明变量)

语法

```
var a = 1;
var b = 2;
```

var 关键字

a 变量名

= 赋值表达式

1 值

变量的数据类型跟存的值相关联

命名规范

1. 以字母或下划线开头 后可以跟字母、数字、下划线

不能有空格或特殊字符(除了和$以外)等

2. 不可以使用关键字作为变量

3. 变量命名最好把变量的意义与其代表的意思对应起来 以免发现错误

4. 区分大小写

驼峰命名法

1. 大驼峰（每个单词首字母大写）UserName

2. 小驼峰（首个单词字母小写 其它单词首字母大写）userName

烤串命名法 （使用下划线 user_name）

允许一个 var 声明多个变量

声明是用逗号隔开

```
va test1 = "声明多个变量1",
    test2 = "声明多个变量2";
```

js 区分大小写

```
var userName = 1;

var username = 2；
```

上面是两个完全不同的变量

1.11.10. 常量（const 关键字声明常量）

语法

a 表示一个常量

= 赋值表达式

100 值

const a = 100;

1. 声明后无法更改

2. 声明后必须赋值

命名规范

1. 尽量使用大写命名
2. 如果多个单词需要突出 用下划线分割

1.11.11. 对象

两个新对象是不一样的

标准类对象（由ECMA提供的标准对象）

1. String 字符对象

2. Date 日期对象

3. Array 数组对象

4. Math 数字对象

5. RegExp 正则类对象

6. Object 对象类

自定义对象

```
var obj1 = {};
```

1.11.12. 运算符

算术运算符

强制转换类型 **Number(strNumber)**

1. + 加

算数运算符在进行处理的时候
如果遇到其中一个是字符 那么他们就会进行拼接

2. - 减

在算术运算减中 如果变量是字符类型的数字
会被隐式（系统自动转换）转换成数字

3. * 乘

在算术运算乘中如果变量是字符类型的数字
会被隐式**(**系统自动转换**)**转换成数字

4. / 除

在算术运算乘中如果变量是字符类型的数字
会被隐式**(**系统自动转换**)**转换成数字

5. % 取模

没有 除尽 剩余的值
求余**/**取模

6. ++ 递增

前置 直接使用递（增**/**减）后的结果
后置 下一次才使用递（增**/**减）后的结果

7. -- 递减

前置 直接使用递（增/减）后的结果

后置 下一次才使用递（增/减）后的结果

赋值运算符

等于 =

var a = 100;

1. +=

a += 100; (100 + 100)

下列同

2. -=

3. *=

4. /=

5. %=

1.12. Day11

1.12.1. 比较运算符

都会产生一个 Boolean 类型的结果

true

false

1. >

大于

2. <

小于

3. >=

大于等于

4. <=

小于等于

5. ==

等于

两个 == 号比较字面量不比较类型

6. ===

全等于

除了判断字面量 还会判断类型是否相等

7. !=

不等于

8. !==

不全等于

原理同 ===

1.12.2. 逻辑运算符

&& 与

判断两遍条件表达式都成立返回 **true**，反之**false**

|| 或

只要两边的表达式有一个条件成立就成立 为true

! 非

取反 console.log(!0)

加括号可以提升优先级比较

强制转换为Boolean为 false的数据有哪些

1. 数字 0

2. null

3. undefined

4. 空字符串

1.12.3. 三目运算符

语法 （ 条件？ 结果1 : 结果2 ）

实例

```
var cc = 100 == 99.99 ? "ok" : "no";

console.log(cc)  //返回 no
```

1.12.4. 流程控制

程序代码的执行顺序，在任何一种语言中，程序控制是必须的。它能使整个程序减少混乱。使之顺利按照其一定的方式执行。 Javascript
常用的控制流程有3种基本结果

1. 顺序结构

正常情况下，程序种的语句是按照书写顺序执行的，这种结构称为顺序结构。前面的程序都是顺序结构的 程序

注： 有异步的话就不会按照顺序执行

2. 选择结构（选择语句/判断语句）

主要用判断语句来实现根据一定的条件决定进一步执行那些语句。Javascript 提供了两种类型的选择结构语句，也称为判断语句

1. if 语句

2. switch 语句

一般来说

if 语句 用于从两条执行路线中选择其一执行
switch 语句 用于从多条执行路线种选择其一执行

3. 循环结构

循环结构的执行步骤

1、声明循环变量；

2、判断循环条件

3、执行循环体操作

4、更新循环变量

1.12.5. 选择语句

页面弹窗询问框 （系统自带）两个按钮 确定取消

window.confirm("是否选择加入Javascript 学习！ ")

确定 true

取消 false

页面待输入框的 弹窗

window.prompt("这里是提示用户的内容")

if 语句

语句1

```
if () {
  //条件成立执行这里的代码块
}
```

语句2

```
if (条件) {
  //条件成立执行这里的代码块
  //代码块
} else {
  //反之执行这里的代码块
  //代码块
}
```

switch 语句

```
switch (指定一个值) {
```

```
case 1:
   //条件成立执行这里的代码块
break; //跳出语句

case 2:
    //条件成立执行这里的代码块
 break;//跳出语句

 default : //默认输出
     //以上条件都未成立执行的代码块

}
```

1.12.6. 循环结构

对于实际情况，需要根据条件或计数器来重复执行多行程序代码的能力，利用循环语句来实现

Javascript种有两种主要类型的循环语句

for循环 while循环

for循环

流程 语句1 => 语句2（ 判断条件是否成立 ） => 语句3 （ 修改变量使条件能停止 ）

练习：99 乘法表

```
for ( 初始化 （语句1）; 表达式（语句2）; 递增/递减（语句3）) {

  //循环执行的代码块，条件成立后结束
```

```
}
```

for in 循环

语法

```
for( var key in obj ) {

 //用于循环对象
 //同样可以循环数组 但是效率比 for 循环效率低

 cosole.log( key + " : " ,obj[key] );

}
```

for of 循环

直接遍历（循环）出数组的值 不含下标
不可循环对象

语法

```
for(index of users) {

  console.log( index );

}
```

while & do-while

while

```
//初始值写到这里
while( 条件 ) {

  //递增写内部
}
```

do-while

```
//至少会执行一次

do {

  //递增
} while ( i < users.length );
```

break-continue

break;

语法

```
for (var i = 0; i < 10; i++) {
  console.log( i );
  if ( i == 5 ) {
   //break; 执行时 整个循环停止循环
    break;
  }
}
```

跳出本层循环，继续执行循环后面的语句。

如果循环有多层嵌套，则**break**只能跳出一层

continue;

语法

```
for (var i = 0; i < 10; i++) {
  console.log( i );
  if ( i == 5 ) {
   //continue; 执行时 整个循环停止循环
    continue;
  }
}
```

取偶数

```
   for ( var i = 0; i < 101; i++) {

            if( i % 2 === 1) {
               continue;
            }
            sum += i;
            console.log(sum);
         }

   var sum = 0;

         for ( var i = 0; i < 101; i++) {
            // i %= 2;
            // console.log(i,i % 2);
            if ( i % 2 === 0) {
               console.log(i + "是偶数")
               sum += i;
               console.log("0 - 100的偶数和是" + sum)
```

```
        }
    }
```

跳过本次循环剩余的代码，继续执行下一次循环，

① 对于for循环，contiune之后执行的语句，是循环变量更新语句i++；

② 对于while、do-while循环，contiune之后执行的语句，是循环条件判断；

因为，使用这两个循环时，必须将contiune必须放到i++之后使用，否则contiune将跳过i++

导致死循环；

1.12.7. document.write("在页面上输入内容")

```
document.write("在页面上输入内容")

document.writeln("在页面上输入内容后添加一个空格")
```

1.12.8. 对象

对象 （键值对集合 周逗号隔开的键值对）

```
var obj {

  username: "zhangsan",
  password: "123456"

}
```

使用点访问对象

```
console.log( obj.username );
```

使用下边访问 下边（方括号）[string]

console.log(obj["username"]);

当对象的 key 为数字时 必须使用下标来访问 obj["1"]

1.12.9. 数组

使用方括号来定义

数组 对象的扩展

数组也是对象 key是数组在初始化的同时自动生成

arr.1 点后面跟数组会被js解析为小数 所以不能访问

console.log(arr[0]);

语法

var arr = [];

数组里的内容可以添加任意内容

1.12.10. 函数基础

解决复用的问题

定义函数 关键字（function）

语法

```
function functionName() {
}
```

1. function 关键字

2. functionName 函数名称

3. () 函数参数

4. {} 函数体

使用函数 call 调用

functionName();

name 以定义好的函数名称

语法

```
function foot() {

 alert( " 您调用了函数foot " );

}

foot();
```

单参数的函数

```
function sum( x, y ) {
    // x 参数x (形参) 形式上的参数 等待传参
    // 形参可以接受多个 使用逗号分隔
    // y 参数y

    // 如果x是一个函数 我们可以直接调用它（回调函数）
    // 因为js是弱类型语言 尽量在使用参数前 先判断参数的类型去使用
    if( typeof x === "function" ) {
        x( "这里可以给传递进来的函数 传递实参" )
```

```
  }
}


// sum(1) 1为实参 在调用函数时 可以传递实参
// 实参可以同时传递多个 使用逗号分隔
// 只要是js支持的数据类型都可以传递到函数里
// sum(1);

//传递函数
sum(x);

function x() {
  alert(1);
}


//在调用未 传参的时候 返回值是 undefined
```

1.13. Day12

1.13.1. 函数

function

定义函数

```
function a() {}
```

使用函数

```
a()
```

函数的参数

定义形参（形式上的参数 值不固定）

定义函数时 所有的参数都是形参

语法

function b(test) {}

弱类型的语言 形参是需要判断类型的

if(typeof test !== "string") {}

如果在使用函数时 不传递实参

相当于 **test**变量没有被赋值（**undefined**）

强弱类型的区别

强类型

编写时（静态类型判断）

弱类型

运行时（动态类型判断）

定义实参

在使用函数时 可以给函数传递实际的参数

传递的类型不限制

参数的个数也不限制

b(1,2,3,4)

接口文档 约定

（单独 js 文件）

```
/**
 * 测试函数
 * @param ( string ) a 描述a是用来干嘛的
 * @param ( string } b
 */
```

写函数最好写上约定

return 任意值;

在函数里使用 可以用来给函数调用之后返回结果

让函数输出结果

1.13.2. 作用域

（避免命名冲突）

单函数

函数内部可以访问函数外部的成员（变量）

从函数外部 不能访问到函数内部的变量

在 window 下面的变量被称为全局变量

在所有函数体外（不在任意一个函数体内编写的变量）编写的变量会被默认托管到window对象下

不使用 var 关键字声明的 变量

在局部和全局都好不到的变量

如果直接赋值 会默认托管在window对象下

```
function etst() {
  a1 = 3;
} //不建议使用
```

函数内部的变量是函数私有的（局部变量）

函数嵌套

函数的变量 可以给子函数或者子孙函数使用

子孙函数可以使用祖先函数的变量

变量提升

函数内外相同变量名且使用 var 关键字声明的值
会使用函数体内的变量（就近原则）
不会影响到外部的变量

函数解析流程

函数内外相同变量名且使用 var 关键字声明的值
会使用函数体内的变量（就近原则）

函数的新的上下文里 就会先把 变量创建好

函数在解析的时 会创建一个新的上下文

创建函数的过程是在解析的阶段就创建好了

不是在执行阶段才会去创建函数

1.13.3. 匿名函数

没有名称的函数

```
var a = function() {}
```

属于变量 不会预加载

立即执行函数

（闭包/闭环 环境）

语法

```
;(function(a){
  console.log(a)
}("哈哈"));

;(function(a){
  console.log(a)
})("哈哈");
```

加分号的原因 是为了 强制结束之前的代码（使他不报错）

加括号的原因 匿名函数不存在变量会报错 我们可以括号的计算出结果

匿名函数调用的括号写在内外都可以

1. 立马就执行（自己执行自己）

2. 不命名 不占用命名空间（为了防止后面写函数的人 把我的函数覆盖）

函数体内函数和变量是私有的（匿名函数空间）

给环境外使用（创建接口**window**）

语法

```
<script>

;(function(a){
```

```
function test() {

  console.log( " 闭包里的函数 " );

}
  // 这里把函数赋值在window（全局）下
  window.test = test;

  window.aaa = 1024;

}());

window.test();

console.log(aaa);

</script>
```

1.13.4. 参数数量（argumens）

语法

```
function sum() {

console.log(arguments)
arguments.length
arguments[0]

}
sum();
```

argumens(关键词) 每个函数独有

捕获函数里面参数的个数 和参数的实参

arguments 是一个数组对象

用来收集当前函数的实际参数

arguments 是关键词 不能作为变量名称 和形参名

1.13.5. 对象

语法1

```
var obj = {};
```

效率高（推介使用）

```
var obj1 = {
 name: "XX",
 age: 20,

};
```

语法2

```
var obj1 = new  Object{};
```

利用object 构造函数创建的对象

1.13.6. 类（分类）生产对象

模板

工厂

抽象

抽象的作用是让人 好去理解代码

1. 抽

抽取一个现实物品的特点 然后用代码给描述出来

2. 象

形成对象

目的 让人和计算机之间 好理解

ES5 语法

```
;function Man(name, age, city) {
        this.name = name;
        this.age = age;
        this.city = city;
      };

      var Man_1 = new Man("张三", 20, "贵州");
      console.log(Man_1);
```

ES6 语法

```
;(function() {
        // ES6语法
        class Person{
          // 构造函数
          // 添加生产出来的对象的特征
          constructor(name,sex) {
            this.name = name;
            this.sex = sex;
```

```
            this.city = "中国";
        }
    }
    var p1 = new Person("李四", "男");
    var p2 = new Person("王五", "男");
    console.log("ES6语法", p1, p2)
})();
```

1.14. Day13

1.14.1. 对象

对象的结构是键值对（key: value,）结构

键可以是数字 也可以是字符

值可以是js的数据类型的任意类型

函数类型

```
var obj1 = {

// es5的语法
 run: function() {};

// es6的语法 同上函数
 run() {};

 };
```

如果对象的 key 键 value 值 相等的话 可以简写

```
var name = "王五";
function Fun() {};

var obj = {
 // name: name,
 name,  // 这里是简写
 // test: test,
 test,  // 这里是简写
};
console.log( obj );
```

语法

1. **var obj1 = { };**
2. **var obj1 = new Object{ };**

对象在堆内存，指针没有指向同一个堆的时候两者是不相等的，反之亦然

访问方法

一

使用下标给对象设置属性

```
obj[ " name " ] = " 张三 " ;
```

使用下标获取对象的属性

```
console.log(obj[ " name " ]);
```

二

使用点的方式给对象设置属性

obj.sex = " 男 ";

使用点获取对象的属性

console.log(obj.sex);

console.log(obj[" sex "]);

this 指针 指某一个对象

用在不同地方 就指不同的对象

在函数里使用**this**

函数属于那个对象 **this**就指向那个对象

谁的方法就 **this** 指向谁

语法

```
var obj = {
        name: "张三",
        run() {
          console.log(this.name + "正在跑步");
          // 这里调用了obj下的 say() 方法
          this.say();
        },
        child: {
          name: "小张三",
          buy() {
            console.log(this.name + "去买东西");
          },
        },
        say() {
```

```
            console.log("喊口号");
            // 这里调用了child里的 buy() 方法
            this.child.buy();
        },
    }

    obj.run();
```

1.14.2. 类（类型）

工厂（模具）

命名方式 大驼峰

1. 类相当于是一个对象加工器

2. 类加功出来的对象 属于当前这个类的 类型

两种版本的语法

es5

在类中的this 表示 即将生产出来的对象

普通函数的this指向是window

把函数当成类使用 必须使用 new 关键字

（new 生产出来的对象 类的实例）实例

实例如果想要访问对象的原型链方法 需要使用 __proto__ 来访问

```
console.log(obj.__proto__)

function person(sex, name) {
```

```
    this.name = name;
    this.sex = sex;
    console.log(this);
  }

  new person( "男", "张三" );
```

es6

```
  class Person {

    //构造函数用来接收调用时的参数
    constructor(name, age) {

    this.name = name;
    this.age = age;
  }
  run() {
    console.log(this.name + "正在跑步");
  }
  }

  var p1 = new Person("张三", "50");
  console.log(p1);
```

instanceof 关键字

可以用来判断某个对象是否是当前这个类型

```
console.log( obj === Object )
```

原型链（prototype）[e56去除了原型链]

给类生产出来的对象添加扩展

通过 prototype 给类添加扩展

利用原型链给对象添加更多的扩展

语法

```
class Person {

  //构造函数用来接收调用时的参数
  constructor(name, age) {

  this.name = name;
  this.age = age;
}
}

Person.prototype.run() {
  console.log( " 正在跑步 " );
}

var p1 = new Person("小李子", "男");
// 每一个生产出来的对象都会拥有run方法
```

作用

1. 在使用某一个类 生产对象时 你觉得对象里的方法或者成员不够使用

2. 可以使用原型链给这个类扩展方法和属性

3. 生产出来的对象就会都有你扩展的方法

4. 原型链方法中的this 指向的就是 即将创建的对象

ES6语法

```
class Person {
  constructor(name, sex) {
    this.name = name;
    this.sex = sex;
  }
  run() {
    console.log(this.name + "正在跑步");
  };
}

class PersonPlus extends Person {
  constructor(name, sex, age) {
    // 如果当前类继承自某个类 必须调用父类的构造函数
    // super 相当于Person的constructor（构造函数）
    // 如果有继承 super必须第一个使用
    super(name, sex);
    // 扩展的属性
    this.age = age;
  }
  // 扩展的方法
  say() {
    console.log("");
  };
}

var p = new PersonPlus("小李子", "男", 20);
p.run();
```

p.say();

console.log(p)

1.14.3. 定时器

seTimeout（延时执行）

seTimeout(function() {

consol.log("延时两秒执行");

}, 2000)

只执行一次

异步执行栈

遇到异步 放入到异步执行栈 等待执行
（跳过这里进入下一个代码片段）

seTinterval(定时执行)

子主题 1

停止方法

1. 存变量

2. clearInterval （ setimeout 同)

1.14.4. Math

1. Math.fllor()

子主题 2

1.15. Day14

1.15.1. 字符

indexOf()

查找文本首次出校的下标

找到 返回下标

未找到 返回 -1

1. 取字符

使用下标的方式（new）

length

charAt()

```
var  name = "abc";
console.log( name.charAt(0) );
```

2. 转换大写

toUpperCase()

toLocaleUpperrCase()

当前地区对应大写字母

3. 转换小写

toLowerCase()

toLocaleLowerCase()

当前地区对应的小写字母

4. 截取

substr()

(from, length)

a. from 开始下标

b. length 取多长

substring()

(start, end)

不含最后一位数值

a. 开始下标

b. 结束下表

5. replace()

找到需要替换的内容，替换成空或者指定的字符

6.分割

var str2 = "useaname=zhangsan&password=123456";

split() //把字符转换成数组

1.15.2. Number

字面量和类型

var num = 123.1;

var num1 = 1;

var num2 = new Number("1");

console.log(num == nun2); //字面量等于 true

console.log(num === nun2); //字面量等于 类型不等于 false

1. toString([radix])

转换成字符串

[radix] 转换成 （2 - 36）进制

2. toFiexd(length)

取小数点后的位数（长度）

parseInt((num))

转换整数

parseFloat(numn)

转换浮点数

1.15.3. Object

静态部分

1. Object.create(proto);

用指定的原型对象 创建一个新的对象

2. Pbject.assign(o1,o2...//可以合并多个);

用只当对象 o1 合并 o2 （o2不变） o1 生成新的合并后的对象

a. target 目标对象

b. other 其它对象

Object.keys();

1.15.4. 拦截器

子主题 1

1.15.5. Array

语法

1. var arr1 = [];

2. var arr2 = new Array();

3. var arr3 = Array();

instanceof 判断是否是数组类型

consol.log(arr1 instanceof Array)

Array 静态方法

1. Array.isArray() 返回bool

属性

1. length 长度 数组里有多少个成员

1.16. Day15

1.16.1. 数组

二维数组

```
var arr1 = [

[
1,
2,
3
],
[

1,
2,
3

]

];
```

多维数组

```
var arr1 = [

[

["",""],
["",""],
[]

],
[

1,
2,
```

3

]

];

数组里可以包含任意类型的成员

例子

var array = [null, " function() {} ", null];

没有下标越界这一说法 如果下标不存在 返回 **undefined**

数组常用的方法

var arr = ["张三","李四"];

1. push(... member)

向后追加 （会改变原始数组）

返回新的长度

语法

arr.push("小明","小芳")

2. unshift(... member)

向前追加（会改变原始数组）

返回新的长度

arr.unshift("小明","小芳")

3. pop()

删除最后一个成员（并返回最后一个）（会改变原始数组）

语法

```
arr.pop( "小明","小芳" )
```

4. shift()

删除一个成员（并返回第一个）（会改变原始数组）

5. concat(... member[])

连接两个或更多的数组，并返回结果

不会改变原始数组 返回连接后的新数组

语法

```
arr.concat( arr1, arr2 )
```

6. join(separator)

通过分隔符把数组组合成一个字符串

1. separator 分隔符 可选 默认为逗号

arr.join( )

6. reverse( )

反转/颠倒顺序

reverse()：颠倒数组中元素的顺序

语法

```
arr.reverse();
```

7. sort()

对数组的元素进行行排序

语法

```
arr.sort()

arr.sort( function(a,b) ) {

 //使用 a b两个数组做比较
 // return a - b;  升序
 // return b - a;  降序

}
```

默认排序方式 先数字 后 字母 拼音首字母

字符串排序

```
var arrStr = [ "你好", "他好", "大家好", ];
```

排序语法

```
arrStr.sort( function(a,b) {

  return b.localeCompare(a, "zh-Han")
  return b.localeCompare(a, "zh-Han")
} );
```

8. slice(start?: number, end: number)

截取数组中的成员 {返回新数组 } 截取不包含结束下标

1. start 开始下标

如果不设置 start开始 默认从开始截取到结束

2. end 结束下标

9. splice(start: number, deleteCount?: number)

截取并从元素数组删除指定元素 返回截取的数组 **{影响原数组}**

1. start 开始截取删除的下标位置

2. end 是可以选的 不设置（默认为数组的长度）会包含后面的所有成员

3. deleteCount 代表的是数量 是可以选的
不设置（默认为数组的长度）会包含后面的所有成员

10. indexOf(value: string)

在数组里查找对应**value**的下标 能找到返回下标 反之返回**-1**

数组的扩展方法

var arr = [];

1. arr.forEach(callback);

无返回值
用于循环
遍历（循环）数组

arr.forEach(function(value, index, arr) {});

1. value 每次循环的成员

2. index 成员的下标

3. arr 原始数组

2. arr.find(callback);

有返回值

需要做判断

如果 callback 返回值为 true 则返回true的成员 就停止循环

作用 从数组困里遍历查找某个成员

找不到返回 **undefined**

3. arr.findIndex(callback);

返回下标

4. arr.map(callback);

根据原始数组生成一个新数组

5. arr.filter(callback);

从数组中去过滤

1.16.2. Date

语法

会比较字面量

1. var d1 = new Date();

new 可以在里面传入时间

var d1 = new Date(" 2017-08-10 07:00:00 ");

2. var d2 = Date();

获取属性

1. 获取年

getFullYear()

本地的

getUTCFullYear()

世界的

2. 获取月 （0-11）

如果要获取准确的月份 需要+1

getMonth()

3. 获取星期几（0-6）0 代表星期天

getDay()

5. 时

getHours()

6. 分

getMinutes()

7. 秒

getSeconds()

8. 毫秒

getMilliseconds()

9. 时间戳

js 浮点精度问题 所以时间戳也 X 1000

设置属性

var d4 = new Date();

1. 设置年

setFullYear

语法

```
d4.setFullYear( d4.getFullYear() + 1 );
```

2. 设置月

setMonth()

语法

```
d4.setMonth( d4.setMonth() + 1 )
```

3. 设置日

setDate()

语法

```
d4.setDate( d4.setDate() + 1 )
```

4. 设置时

5. 设置分

6. 设置秒

7. 设置毫秒

1.16.3. 正则表达式

RegExp

正则范围

1. [0-9]

匹配数字

2. [a-z]

匹配小写字母 a-z 小写

3. [A-Z]

匹配大写字母 A-Z 大写

4. [A-z]

匹配大小写字母

5. [abc]

自定义内容 你好

6. (a|b|c)

自定义内容 或者 只要字符中有自定义的内容都能匹配

[^a] 不包括

[^a-z] 不包括小写字母

[^A-z] 不包括所有大小写字母

元字符

具有功能的字符 待斜杠开始 在正则里表示转义字符

1. \w

查找单词字符。包括下划线 [A-z _ 0-9]

2. \W

查找非单词字符。和 \w 相反 [^A-z _ 0-9]

4. \d

查找数字字符。 相当于[0-9]

5. \D

查找非数字字符。和\d相反 [^0-9]

6. \s

查找空白字符。 space

7. \S

查找非空格。 和\s相反

8. \b

匹配单词边界

9. \B

匹配非单词边界

10. \n

匹配换行符

量词

^

开始

语法

^b 文本的内容必须以b字母开始才能被匹配到

^[a-z] 文本的内容必须以小写字母开始才能被匹配到

^[A-z] 文本的内容必须是字母开始才能被匹配到

^ \w 文本只要是字母开始都能匹配到

+

匹配任何包含至少一个 n 的字符串

b+ 一个或者多个b就能匹配

^b+ 一个或者多个b开头就能匹配

*****

匹配任何二包含零个或者多个 n 的字符串

?

匹配任何包含零个或1个的字符串

{x}

必须包含指定的数量 符号

语法

x{2} 必须包含2个x

[A-z]{3} 必须包含3个字母

[A-z]{3-5} 必须包含3-5个字母

$

结束

(com|cc|cn\org)$ 以其中任意一个单词结束

[0-9_x]$ 必须以数字或者x结尾

?=n

匹配跟随的

^http(?=:) 匹配 http 后面跟的是 冒号

?!n

匹配没有跟随

^http(?!:) 匹配 http 后面跟的 不是冒号

子主题 2

1.16.4. dom

onclick=""

单击事件

ondblclick=""

双击事件

1.17. Day16

1.17.1. 正则

表达式后面 加上 **flag i**　　表示正则表达式不区分大小写

i

不区分大小写

g

全局匹配

1.17.2. DOM = document object model（文档对象模型）

顶层对象 **document**

里面包含的是所有的**html**标签对象

element 元素

document.getElementById(stringID);

使用 **id** 获取元素

document.getElementsByClassName(stringClassName);

使用 **class** 获取元素

document.getElementsByName(stringName);

使用 **name** 属性的值 获取元素

document.getElementsByTagName(stringTagName);

使用 标签名称 获取元素

documen.querySelector( );

根据CSS 选择器查找 找一个 找到就返回

语句

documen.querySelector(" #userName ");

documen.querySelectorAll( );

使用 CSS选择器查找所有的

语法

documen.querySelectorAll(" #userName ");

在找到的元素内部找

aadocumen.querySelectorAll(" #userName .b ");

1.17.3. 修改标签的样式

1.17.4. 获取表单内容

子主题 1

1.17.5. 标签子内容

document 由节点组成（1. text 文本节点 2.element 元素节点）

所有成对的标签 容器（可以嵌套的元素）

innerHtml

给标签里替换元素节点（包括 标签节点和文本节点）

innerText

给标签里替换文本节点（node）=（也被称作文本元素）

createElement( );

创建元素

创建一个 li 标签

只是创建 没有添加到文档中

document.createElement(" li ");

每个元素对象在文档中（documen中）都是唯一的

标签元素的（ classList ）对象用于管理标签的class属性

classList(" a ");

element.appendchild( );

向后追加

添加元素

element。insertBefore( );

element.children

所有的子元素 不包括文本节点

element.childNodes

所有的子元素 包括文本节点

element.firstChild

子元素的第一个 包含文本节点 （有可能是标签节点
也有可能是文本节点）

element.listElementChild

子元素的第一个 不包含文本节点

element.lastChild

子元素的最后一个 包含文本节点 （有可能是标签节点
也有可能是文本节点）

element.lastElementChild

子元素的最后一个 不包含文本节点

当前元素的父元素

element.parentNode

element.parentElement

element.remove

element.replaceChild( );

1.18. Day17

1.18.1. 实例化一个错误

js 自带的一个类 创建错误

new Error( );

new TypeError("类型错误");

1.18.2. throw 关键字 用于抛出错误（抛出异常）

throw new Error(" 显示的错误内容 ");

1.19. Day18

1.19.1. 兄弟节点

在事件的处理函数中 如果传递this 表示当前元素

nextSibling 下一个兄弟元素 包含文本节点

nextElementSibling 下一个兄弟元素 不包含文本节点

previousSibling 上一个兄弟元素 包含文本节点

previousElementSibling 上一个兄弟元素 不包含文本节点

1.19.2. 父节点

parentNode 获取当前元素的父节点

offsetParent 找定位的父元素 如果没有定位 默认返回 body

向上

1.19.3. 事件

onload

window的onload事件 当窗口加载完毕之后触发

window.document 也加载完毕了

调用时机实在window加载完毕之后才触发的

1.19.4. 鼠标事件

单击

```
var box = document.getElementById( " box " );
box.style.background = " black ";
```

box.onclick = function() {

this.style.background = " #000 ";

}

必须是鼠标左键单击

双击

var box = document.getElementById(" box ");

box.ondblclick = function() {

this.style.background = " #000 ";

}

必须是鼠标左键双击

box.onmousedown

鼠标按下

box.onmousedown = function() {

console.log(" 鼠标按下了 ");

}

鼠标在当前元素上 并且按下就会触发

鼠标正常的 左右键和滚轮都可以触发

box.onmouseenter

鼠标进入

```
box.onmouseenter = function() {

    console.log( " 鼠标进入了 " );

}
```

只要鼠标进入到元素的范围之内就会触发

box.onmouseleave

鼠标离开

```
box.onmouseleave = function() {

    console.log( " 鼠标离开了 " );

}
```

鼠标从当前元素离开触发

box.onmousemove

鼠标移动

```
box.onmousemove = function() {

    console.log( " 鼠标正在移动 " );

}
```

鼠标在当前元素上的移动

box.onmouseout

鼠标离开

```
box.onmouseout = function() {

    console.log( " 鼠标离开了 " );

}
```

和 box.onmouseleave 一样 只是在同一个元素上out比leave事件先触发

box.onmouseover

鼠标停留

```
box.onmouseout = function() {

    console.log( " 鼠标正在移动 " );

}
```

鼠标停留在当前元素上触发 只执行一次

over 和 enter的 效果一样

over 比 enter 先触发

box.onmouseup

鼠标抬起

```
box.onmouseup = function() {

    console.log( " 鼠标弹起 " );

}
```

鼠标在当前元素上弹起时就会触发

event

是一个全局对象 指此时正在触发的事件

box.onmousedown = function() {

console.log(event)

console.log(event.button)

}

1.19.5. 利用鼠标事件拖拽元素

// 如果event为 undefined 就使用ev变量 兼容老浏览器

var event = event || ev;

event.button

0 鼠标左键

1 鼠标滚轮

3 鼠标右键

潘顿是否鼠标左键语法

if(event.button == 0) {

}

event.buttons 判断是否有多个鼠标触发

1.19.6. 元素的 offset

oBox.offsetLeft

元素的左偏移值

oBox.offsetTop

元素的上偏移值

oBox.offsetWidth

元素的宽度

oBox.offsetHeight

元素的高度

1.19.7. var elemenX = oBox.offsetLeft;

获取元素原本在页面中的**x**位置

1.19.8. var elemenY = oBox.offsetTop;

获取元素原本在页面中的**y**位置

1.19.9. 操作标签的属性

子主题 1

1.19.10. 表单事件

1. onfocus

当前元素获取到 焦点（光标）触发

2. obblur

当前元素失去光标 焦点 触发

3. onchange

当输入框里的内容发生改变 并且失去焦点时触发

4. oninput

当输入框的内容输入新的内容时触发

概要(var elemenX = oBox.offsetLeft;, var elemenY = oBox.offsetTop;, 操作标签的属性, 表单事件)

1.20. Day19

1.20.1. 键盘事件

1. event.ctrlKey 如果为true 说明

2. event.altKey

3. event.shiftKey

4. event.metaKey

5. event.code 键盘的按键名称

6. event.Key 键盘按键的值

1.20.2. 事件委托

子主题 1

1.20.3. 事件冒泡

1.21. Day20

1.21.1. 他把

1.22. Day21

1.22.1. window

返回窗口的文档显示区的宽高度

window.innerHeight

window.innerWidth

返回窗口外面的宽高度
浏览器当前的窗口大小

其中Y轴会减掉任务栏的高度
隐藏掉才会显示真是的高度

window.outHeight

window.outWidth

client 包含padding 不含border

宽高

window.clientHeight

window.clientWidth

offsetHeight

1.23. Day22

1.23.1. 子主题 1

1.24. Day23

1.24.1. jQuery介绍

HTML dom操作的一个js库

简化了元素dom操作 还是基于原生

1. jQuery怎么用

2. jQuery和原生api之间的关系

目的

1. Java方向 jQuery怎么用

2. web方向 学习jQuery的设计思路

安装

子主题 1

选择器 通过cs选择器的方式来查找对应的元素

语法

```
jQuery(" div ");

$(" #ss ");

$(" .box .btn ");

$(" div ");

1. $(" #banner ")

2. jQuery(" #banner ")

$ === JQuery
```

和 **document.queryselector(" #bannaer ")** 相同

1.24.2. jQuery使用

页面头部找不到元素处理方案

使用**jQuery**的方法在头部**head**找元素

方案一

ready 函数（等待**DOM**加载完毕之后处理）

语法

```
$().ready(function() {
    console.log(1);
})
```

方案二

给 **jQuery**传递一个函数他会在 **DOM** 加载完毕之后来调用你的函数

语法

```
jQuery(function () {})
```

1.24.3. jQuery获取/设置

获取

获取一个标签的**html**文本内容

原生

```
document.getElementsByClassName(" box ").innerText
```

jQuery

$(" .box ").text()

获取表单的**value**值

原生

document.getElementsByClassName("username")[0].value

jQuery

$(" .username ").val()

获取元素的标签属性值

原生

documen.getElementById(" test ").getAttribute("test");

jQuery

$(" #test ").attr("test")

设置

设置元素内容

原生

document.getElementsByClassName(" box ").innerText = " abcd "

jQuery

$(" .box ").text(" fggg ")

设置**html**

原生

document.getElementsByClassName(" box ").innerHTML =

"<h1>原生方法</h1>"

jQuery

$(" .box ").html("<h1> jQuery </h1>")

获取html元素

$(" .box ").html()

设置表单的value值

原生

document.getElementsByClassName("username")[0].value = 111123

jQuery

$(" .username ").val(112233)

添加元素的标签属性

原生

documen.getElementById(" test ").setAttribute("index", 1);

jQuery

$(" #test ").attr("test", 2)

1.25. Day24

1.25.1. 子主题 1

2. 二阶段

2.1. Day01

2.1.1. Java语言概述

1.1 什么是Java语言

Java语言是SUN（Stanford University Network）公司

在 1995年 推出的一门高级的面向对象的编程语言

编程语言：计算机语言，人们可以使用编程语言对计算机发送指令，完成人们提出的任务

Java语言的创始人：James Gosling

1.2 Java语言的发展史

1995年 SUN公司 发布Java1.0

2004年 SUN公司 发布Java1.5(出现了很多的新特性 泛型 注解 枚举...)

2009年 Oracle公司 收购 SUN

2014年发布 Java1.8（出现了很多的新特性：lambda，stream API，新日期对象）

2019年发布 Java13

1.3 Java语言能做什么

应用在互联网程序开发中：电商平台，P2P平台，在线教育平台，社交平台 ...

传统企业级项目（ERP，电信，移动）

后台大数据的数据挖掘

嵌入式

1.4 Java语言特点（重点）

开源（ **open source** ）：很多大厂商（**Oracle IBM Google**）的拥护

跨平台：编写一次代码，任意 **OS** 运行。 **write once，run anywhere**

简单：针对**C++**对比，**Java**去除了指针运算，垃圾回收

多线程：一项任务（进程）

解释编译：编译（**Compile**）又要结束（**interpret**）

2.1.2. JDK、JRE、JVM三者关系

2.1 什么是**JDK**

Java

2.2 什么是**JRE**

JRE = JVM + class library + user interface + deployment technology

2.3 什么是**JVM**

2.4三者关系

JDK > JRE > JVM

层层包含

2.5 JVM深入了解

运行所有**Java**程序的一个假象计算机。是**Java**程序的运行环境

我们编写的所有**Java**代码都必须在**JVM**之上

跨平台

任何软件的运行都必须在操作系统上

Java编写的软件可以运行在任何操作系统之上，但是在这些操作系统之上必须先要安装 JVM

2.1.3. Java语言发展历史

2.1.4. Java语言特点

2.1.5. Java程序的运行流程

Java源文件（.java）

Java编译器

字节码文件（.class）

类加载器（）

2.1.6. 开发环境搭建

4.1 安装JDK

4.1.1 下载

官网下载（需要注册用户）

4.1.2 安装

安装路径不能有空格

公共 JRE 和 私有 JRE

和JDK平级的JRE是公共JRE

在JDK文件夹内的JRE是私有JRE

区别

私有有 客户端的 JVM 和 服务区版的 JVM

子主题 3

4.2 配置环境变量（重要）

4.2.1 配置PATH环境变量

作用：告诉操作系统在某某路径下找到对应的命令来执行

C:\Program Files\Java\jdk-12.0.2\bin（%JAVA_HOME%\bin;）

每一个环境变量使用 分号 " ; " 隔开

4.2.2 配置JAVA_HOME环境变量

安装JDK后的宿主目录

C:\Program Files\Java\jdk-12.0.2;

如果先配置了JAVA_HOME，那么可以使用 %JAVA_HOME% 指代那串路径

4.2.3 配置CLASSPATH环境变量

程序的运行需要一些其它的类的支撑，必须告知这些类的路径才可以使用

2.1.7. 使用开发工具(Eclipse、IDEA)