# 一、HTML

**1、常见的HTML块级元素和行内元素**

行内元素有：span a b i strong  
块级元素有：form select textarea h1-h6 table button p ol ul dl div  
行内块元素常见的有： img input td

**2、HTML5的新标签有什么了解，详述一下表单的新标签**

​HTML新元素

1. canvas

标签定义图形，比如图表和其他图像。该标签基于 JavaScript 的绘图 API （绘图元素，后面js会使用canvas   api来绘图）

canvas结合cocos2d游戏引擎可以做网页游戏开发，还可以绘制网页特效

2.新多媒体元素

（1）audio  音频播放器

（2）video   视频播放器

（3）source标签   类似src属性

属性：autoplay  自动播放   controls  控制器        loop   循环播放   muted   静音

3.新的表单元素

datalist  与input联用  定义选项列表。请与 input 元素配合使用该元素，来定义 input 可能的值。

<input type="text" list="put"/>

<datalist id="put">

<option>1</option>

<option>12</option>

<option>123</option>

<option>1234</option>

<option>12345</option>

<option>123456</option>

</datalist>

output   类似span  输出

表单新更新的属性

autucomplete：on（开启表单输入智能提示）    off（关闭智能提示）

浏览器默认存在这个属性

<form action="demo-form.php" autocomplete="on">

First name:<input type="text" name="fname"><br>

Last name: <input type="text" name="lname"><br>

E-mail: <input type="email" name="email" autocomplete="off"><br>

<input type="submit">

</form>

form   form 属性规定输入域所属的一个或多个表单。

<form action="demo-form.php" id="form1">

First name: <input type="text" name="fname"><br>

<input type="submit" value="提交">

</form>

<input type="text" name="lname" form="form1">

multiple属性

multiple 属性是一个 boolean 属性.

multiple 属性规定<input> 元素中可选择多个值。

注意: multiple 属性适用于以下类型的 <input> 标签：email 和 file:

Select images: <input type="file" name="img" multiple>

pattern属性

pattern 属性描述了一个正则表达式用于验证 <input> 元素的值。

注意:pattern 属性适用于以下类型的 <input> 标签: text, search, url, tel, email, 和 password

<form action="demo-form.php">

Country code: <input type="text" name="country\_code" pattern="[A-Za-z]{3}" title="Three letter country code">

<input type="submit">

</form>

显示了一个只能包含三个字母的文本域（不含数字及特殊字符）

5.新的语义和结构元素

article       定义页面独立的内容区域。相当于div

aside        定义页面的侧边栏内容。

bdi            允许您设置一段文本，使其脱离其父元素的文本设置。相当于span

details       用于描述文档或文档某个部分的细节

summary   标签包含 details 元素的标题

<details>

<summary>个人信息</summary>

<p>

新的语义和结构元素

article  定义页面独立的内容区域。相当于div

aside  定义页面的侧边栏内容。

bdi  允许您设置一段文本，使其脱离其父元素的文本设置。相当于span

</p>

</details>

dialog     定义对话框，比如提示框

header   定义头部区域

footer      定义底部区域

mark       突出显示

nav         定义导航的部分

meter     进度条（仅用于已知最大和最小值的度量）

<meter value="10" max="100"></meter>

progress     定义任何类型的任务的进度条

section      定义文档中的节（section、区段）

time           定义日期或时间

**3、Image标签中alt和title有什么区别**

相同之处：它们都会出现一个小浮层，显示图片相关的内容

　　不同之处如下所示

　　alt属性的特点：

　　①倘若图片加载不成功未能显示出来，就会在图片未显示的地方出现一段文字。这一作用是为了给未加载出来的图片提供信息，方便用户浏览网页，同时也方便开发人员维护网页。

　　②搜索引擎可以通过这个属性的文字描述获取图片

　　title属性的特点：

　　title属性可以用在任何元素上，当用户把鼠标移动到元素上时，就会出现title的内容，起到对图片说明的作用，其实质就是对图片的一种备注或者注释

　　通俗来讲，alt属性的实质是通过文字来代替图片的内容，而title属性的实质是对图片的描述或者注释。

# 二、CSS

**1. CSS盒子模型（IE盒子模型，标准盒子模型）**

css盒子模型 又称为框模型（Box Model），包含了元素内容（content）、内边距（padding）、边框（border）、外边距（margin）几个要素。

css盒子模型 又称为框模型（Box Model），包含了元素内容（content）、内边距（padding）、边框（border）、外边距（margin）几个要素

两个模型宽度和高度的计算（是一样的）

w3c中的盒子模型的宽:包括margin+border+padding+width;

width:margin\*2+border\*2+padding\*2+width;

height:margin\*2+border\*2+padding\*2+height;

iE中的盒子模型的width:也包括margin+border+padding+width;

上面的两个宽度相加的属性是一样的。

**2、CSS元素居中（能说几种说几种）**

​

一、水平居中

1.行内元素水平居中

（1）父元素是块级元素

直接给父元素设置text-align: center

<style>

.box{

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

text-align: center;

}

</style>

<div class="box">

<span>我是行内元素</span>

</div>

（2）父元素不是块级元素

先将父元素设置成块级元素，再给父元素设置text-align: center

<style>

.box{

display: block;

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

text-align: center;

}

</style>

<span class="box">

<span>我是行内元素</span>

</span>

2.块级元素水平居中

（1）已知块级元素的宽度和高度

需要谁居中，给其设置margin: 0 auto

<style>

.box{

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

}

.boxes{

width: 100px;

height: 50px;

border: 1px solid #e245c5;

background-color: #37ffdf;

margin: 0 auto;

}

</style>

<div class="box">

<div class="boxes"></div>

</div>

（2）未知块级元素的宽度和高度

默认子元素的宽度和高度和父元素一样，这时需要设置子元素为display: inline-block或display: inline。即将其转换成行级块/行内元素，给父元素设置text-align: center

<style>

.box{

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

text-align: center;

}

.boxes{

display: inline-block;

border: 1px solid #e245c5;

background-color: #37ffdf;

margin: 0 auto;

}

</style>

<div class="box">

<div class="boxes">我是块级元素</div>

</div>

（3）使用定位属性（父相子绝）

首先设置父元素为相对定位，子元素为绝对定位

设置子元素left：50%，让子元素的左上角水平居中

设置子元素的margin-left：-元素宽度的一半或者设置transform：translateX（-50%）

<style>

.box{

position: relative;

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

text-align: center;

}

.boxes{

width: 100px;

height: 50px;

position: absolute;

border: 1px solid #e245c5;

background-color: #37ffdf;

left: 50%;

margin-left: -50px;

}

</style>

<div class="box">

<div class="boxes"></div>

</div>

<style>

.box{

position: relative;

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

text-align: center;

}

.boxes{

width: 100px;

height: 50px;

position: absolute;

border: 1px solid #e245c5;

background-color: #37ffdf;

left: 50%;

transform: translateX(-50%);

}

</style>

<div class="box">

<div class="boxes"></div>

</div>

（4）使用flexbox布局实现（宽度确不确定都可以）

给待处理的元素的父元素添加属性display: flex; justify-content: center;

<style>

.box{

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

display: flex;

justify-content: center;

}

.boxes{

width: 100px;

height: 50px;

border: 1px solid #e245c5;

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="boxes"></div>

</div>

二、垂直居中

1.单行的行内元素垂直居中

设置单行行内元素的行高等于盒子的高即可

<style>

.box{

width: 200px;

height: 100px;

border: 1px solid blueviolet;

background-color: orangered;

}

.id{

height: 100px;

line-height: 100px;

border: 1px solid #e245c5;

background-color: #37ffdf;

}

</style>

<div class="box">

<span class="id">我是行内元素</span>

</div>

2.多行的行内元素垂直居中

给父元素设置display：table-cell和vertical-align:middle

<style>

.box{

display: table-cell;

vertical-align: middle;

width: 200px;

height: 100px;

background-color: orangered;

}

.id{

background-color: #37ffdf;

}

</style>

<div class="box">

<span class="id">我是行内元素我是行内元素我是行内元素我是行内元素</span>

</div>

3.块级元素垂直居中

(1)使用定位属性（父相子绝）

首先设置父元素为相对定位，子元素为绝对定位

设置子元素top：50%，让子元素的左上角水平居中

设置子元素的margin-top：-元素宽度的一半或者设置transform：translateY（-50%）

<style>

.box{

width: 200px;

height: 100px;

background-color: orangered;

position: relative;

}

.id{

position: absolute;

height: 50px;

top: 50%;

margin-top: -25px;

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

<style>

.box{

width: 200px;

height: 100px;

background-color: orangered;

position: relative;

}

.id{

position: absolute;

height: 50px;

top: 50%;

transform: translateY(-50%);

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

(2)使用flexbox布局实现（宽度确不确定都可以）

给待处理的元素的父元素添加属性display: flex; align-items: center;

<style>

.box{

width: 200px;

height: 100px;

display: flex;

align-items: center;

background-color: orangered;

}

.id{

height: 50px;

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

三、块级元素水平+垂直居中

1.已知宽度和高度的块级元素水平+垂直居中

方案一：

设置父元素为相对定位，

子元素设置绝对定位，top: 0;bottom: 0;left: 0;right: 0;margin: auto;​​​​​​​

<style>

.box{

position: relative;

width: 200px;

height: 100px;

background-color: orangered;

}

.id{

position: absolute;

top: 0;

bottom: 0;

left: 0;

right: 0;

width: 100px;

height: 50px;

margin: auto;

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

方案二：

设置父元素为相对定位，

子元素设置绝对定位，top: 50%;left: 50%;margin-top：-元素高度的一半；margin-left：-元素宽度的一半。

<style>

.box{

position: relative;

width: 200px;

height: 100px;

background-color: orangered;

}

.id{

position: absolute;

top: 50%;

left: 50%;

width: 100px;

height: 50px;

margin-top: -25px;

margin-left: -50px;

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

​​​​​​​

2.未知高度和宽度的块级元素水平+垂直居中

方案一：使用定位属性

设置父元素为相对定位，

子元素设置绝对定位，top: 50%;left: 50%;transform: translate(-50%,-50%);​​​​​​​

<style>

.box{

position: relative;

width: 200px;

height: 100px;

background-color: orangered;

}

.id{

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%,-50%);

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

方案二：使用flex布局实现

设置父元素为flex定位，display: flex; justify-content: center; align-items: center;

<style>

.box{

display: flex;

justify-content: center;

align-items: center;

width: 200px;

height: 100px;

background-color: orangered;

}

.id{

background-color: #37ffdf;

}

</style>

<div class="box">

<div class="id">我是块级元素</div>

</div>

**3、CSS伪类**

什么是伪类？

伪类用于定义元素的特殊状态。

例如，它可以用于：

设置鼠标悬停在元素上时的样式

为已访问和未访问链接设置不同的样式

设置元素获得焦点时的样式

/\* 未访问的链接 \*/

a:link {

color: #FF0000;

}

/\* 已访问的链接 \*/

a:visited {

color: #00FF00;

}

/\* 鼠标悬停链接 \*/

a:hover {

color: #FF00FF;

}

/\* 已选择的链接 \*/

a:active {

color: #0000FF;

}

**4、CSS定位**

position 属性规定应用于元素的定位方法的类型。

有五个不同的位置值：

static

relative

fixed

absolute

sticky

元素其实是使用 top、bottom、left 和 right 属性定位的。但是，除非首先设置了 position 属性，否则这些属性将不起作用。根据不同的 position 值，它们的工作方式也不同。

position: static;

HTML 元素默认情况下的定位方式为 static（静态）。

静态定位的元素不受 top、bottom、left 和 right 属性的影响。

position: static; 的元素不会以任何特殊方式定位；它始终根据页面的正常流进行定位：

position: relative;

position: relative; 的元素相对于其正常位置进行定位。

设置相对定位的元素的 top、right、bottom 和 left 属性将导致其偏离其正常位置进行调整。不会对其余内容进行调整来适应元素留下的任何空间。

position: fixed;

position: fixed; 的元素是相对于视口定位的，这意味着即使滚动页面，它也始终位于同一位置。 top、right、bottom 和 left 属性用于定位此元素。

固定定位的元素不会在页面中通常应放置的位置上留出空隙。

position: absolute;

position: absolute; 的元素相对于最近的定位祖先元素进行定位（而不是相对于视口定位，如 fixed）。

然而，如果绝对定位的元素没有祖先，它将使用文档主体（body），并随页面滚动一起移动。

注意：“被定位的”元素是其位置除 static 以外的任何元素。

position: sticky;

position: sticky; 的元素根据用户的滚动位置进行定位。

粘性元素根据滚动位置在相对（relative）和固定（fixed）之间切换。起先它会被相对定位，直到在视口中遇到给定的偏移位置为止 - 然后将其“粘贴”在适当的位置（比如 position:fixed）。

**5、能说一下BFC吗，BFC除了清除浮动，防止margin重叠之外，你了解它的原理吗**

BFC详解（块级格式化上下文，用于清除浮动，防止margin重叠等）

定义：块级格式化上下文，是一个独立的渲染区域，并且有一定的布局规则。

BFC区域不会与float box重叠

BFC是页面上的一个独立容器，子元素不会影响到外面

计算BFC的高度时，浮动元素也会参与计算[11] 只要元素满足下面任一条件即可触发 BFC 特性：

body 根元素

浮动元素：float 除 none 以外的值

绝对定位元素：position (absolute、fixed)

display 为 inline-block、table-cells、flex

overflow 除了 visible 以外的值 (hidden、auto、scroll)

普通流的BFC：

块元素上下结构如果属域同一个BFC，这两个元素之间的外间距会重叠

如果将另一个元素用其他元素包裹设置父元素样式 overflow: hidden;

会构建为另一个BFC，这样的两个上下元素之间的间距会叠加 不会重叠

**6、css3的动画实现举例**

**7、100px的div变成圆 有几种方法**

1. 将border-radius设置为50%

1、使用div创建一个模块，并设置其class为ijk。

2、在css标签内，通过class设置div的样式，定义它的宽度为100px，高度为100px，背景色为红色。

3、在js标签内，使用ready()方法在页面加载完成时执行function方法。

4、在function方法内，通过class获得div标签对象，使用css方法将border-radius设置为100%，便可以实现圆形。

**8、css的预处理器，能跟我讲一下你用它解决了什么问题吗，除了变量提取还有什么好用的功能**

CSS 自诞生以来，基本语法和核心机制一直没有本质上的变化，它的发展几乎全是表现力层面上的提升。最开始 CSS 在网页中的作用只是辅助性的装饰，轻便易学是最大的需求；然而如今网站的复杂度已经不可同日而语，原生 CSS 已经让开发者力不从心。

当一门语言的能力不足而用户的运行环境又不支持其它选择的时候，这门语言就会沦为 “编译目标” 语言。开发者将选择另一门更高级的语言来进行开发，然后编译到底层语言以便实际运行。

于是，在前端领域，天降大任于斯人也，CSS 预处理器应运而生。而 CSS 这门古老的语言以另一种方式 “重新适应” 了网页开发的需求。

简单梳理一下，CSS 预处理器为我们带来了几项重要的能力，由浅入深排列如下。（不用在意你用到了多少，无论深浅，都是获益。）

1.文件切分

页面越来越复杂，需要加载的 CSS 文件也越来越大，我们有必要把大文件切分开来，否则难以维护。传统的 CSS 文件切分方案基本上就是 CSS 原生的 @import 指令，或在 HTML 中加载多个 CSS 文件，这些方案通常不能满足性能要求。

CSS 预处理器扩展了 @import 指令的能力，通过编译环节将切分后的文件重新合并为一个大文件。这一方面解决了大文件不便维护的问题，另一方面也解决了一堆小文件在加载时的性能问题。

2.模块化

把文件切分的思路再向前推进一步，就是 “模块化”。一个大的 CSS 文件在合理切分之后，所产生的这些小文件的相互关系应该是一个树形结构。

树形的根结节一般称作 “入口文件”，树形的其它节点一般称作 “模块文件”。入口文件通常会依赖多个模块文件，各个模块文件也可能会依赖其它更末端的模块，从而构成整个树形。

3.选择符嵌套

选择符嵌套是文件内部的代码组织方式，它可以让一系列相关的规则呈现出层级关系。在以前，如果要达到这个目的，我们只能这样写：

4.变量

在变更出现之前，CSS 中的所有属性值都是 “幻数”。你不知道这个值是怎么来的、它的什么样的意义。有了变量之后，我们就可以给这些 “幻数” 起个名字了，便于记忆、阅读和理解。

接下来我们会发现，当某个特定的值在多处用到时，变量就是一种简单而有效的抽象方式，可以把这种重复消灭掉，让你的代码更加 DRY。

5.运算

光有变量还是不够的，我们还需要有运算。如果说变量让值有了意义，那么运算则可以让值和值建立关联。有些属性的值其实跟其它属性的值是紧密相关的，CSS 语法无法表达这层关系；而在预处理语言中，我们可以用变量和表达式来呈现这种关系。

6.函数

把常用的运算操作抽象出来，我们就得到了函数。

开发者可以自定义函数，预处理器自己也内置了大量的函数。最常用的内置函数应该就是颜色的运算函数了吧！有了它们，我们甚至都不需要打开 Photoshop 来调色，就可以得到某个颜色的同色系变种了。

7.Mixin

Mixin 是 CSS 预处理器提供的又一项实用功能。Mixin 的形态和用法跟函数十分类似——先定义，然后在需要的地方调用，在调用时可以接受参数。它与函数的不同之处在于，函数用于产生一个值，而 Mixin 的作用是产生一段 CSS 代码。

Mixin 可以产生多条 CSS 规则，也可以只产生一些 CSS 声明。

8.工程化

CSS 预处理语言无法直接运行于浏览器环境，这意味着我们编写的源码需要编译为 CSS 代码之后才能用于网页。这似乎是一个门槛，需要我们付出 “额外” 的成本。

但在目前的大环境下，大多数项目的前端开发流程已经包含了构建环节，比如选择任何一个脚本模块化方案都是需要在部署时走一道打包程序的。所以对大多数团队来说，这个门槛其实已经跨过去一大半了。

而一旦接受了这种设定，我们还可以享受到 “额外” 的福利。在给 CSS 的开发加入编译环节的同时，还可以顺道加入其它构建环节，比如代码校验、代码压缩、代码后处理等等。

**9、a页面有一个选择器，b有相同的选择器，多人合作导致样式冲突，怎么解决？（我答scoped，他说除了scoped还有什么方法？）**

1. 细化选择符

通过使用组合器（Combinator）将选择器的描述写得更加精确（参考CSS选择器 - MDN ），例如对于下述代码片段，如果想给.cellphones中的.apple增加样式，只使用.apple，势必会对.fruit中的.apple也造成影响。

1

2

3

4

5

6

<div class="cellphones">

<div class="apple"></div>

</div>

<div class="fruit">

<div class="apple"></div>

</div>

可以使用后代组合器（Descendant Combinator）或子代组合器（Child Combinator）这种更为精确的描述。描述得越精确，优先级越高，优先级更高的描述会覆盖优先级较低的描述。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

/\* 后代组合器：所有后代节点 \*/

.cellphones .apple {

border: 1px solid black;

}

/\* 更加精确的后代组合器 \*/

body .cellphones .apple {

border: 1px solid blue;

}

/\* 子代组合器：直接子节点 \*/

.cellphones > .apple {

border: 1px solid red;

}

如果给.apple按顺序加上上述全部样式，最终，边框将呈现蓝色。

详细的优先级规则参见CSS 优先级

2. 再写一次选择器名

本质上是上一种情况的特例。例如对于.apple，添加如下样式：

1

2

3

4

5

6

.cellphones > .apple.apple {

border: 1px solid purple;

}

.cellphones > .apple {

border: 1px solid red;

}

最终，边框将呈现紫色。

3. 改变CSS样式表中的顺序

对于相同类型选择器指定的样式，在CSS文件中的顺序靠后的样式会覆盖之前的样式。

例如对于下述代码中的div元素，浏览器渲染的结果会是红色的：

1

<div class="redBorder" class="blackBorder"></div>

1

2

3

4

5

6

.blackBorder {

border: 1px solid black;

}

.redBorder {

border: 1px solid red;

}

需要注意的是，尽管在HTML文件中.blackBorder出现在.redBorder后，但优先级的判断是根据他们在CSS文件中的顺序。也就是说，CSS文件中更为靠后的.redBorder才会被采用。

4. 主动提升优先级（不建议）

还有一种简单粗暴但是并不建议的办法，就是在需要使用的样式后加上关键字!important可以将样式优先级提到极高。例如：

**10、scoped的原理**

何为scoped？

在vue文件中的style标签上，有一个特殊的属性：scoped。当一个style标签拥有scoped属性时，它的CSS样式就只能作用于当前的组件，也就是说，该样式只能适用于当前组件元素。通过该属性，可以使得组件之间的样式不互相污染。如果一个项目中的所有style标签全部加上了scoped，相当于实现了样式的模块化。

scoped的实现原理

vue中的scoped属性的效果主要通过PostCSS转译实现，如下是转译前的vue代码：

复制代码

<style scoped>

.example {

color: red;

}

</style>

<template>

<div class="example">hi</div>

</template>

复制代码

转义后：

复制代码

<style>

.example[data-v-5558831a] {

color: red;

}

</style>

<template>

<div class="example" data-v-5558831a>hi</div>

</template>

复制代码

即：PostCSS给一个组件中的所有dom添加了一个独一无二的动态属性，然后，给CSS选择器额外添加一个对应的属性选择器来选择该组件中dom，这种做法使得样式只作用于含有该属性的dom——组件内部dom。

**11、鼠标悬浮按钮上，跳出下拉框，使用纯css实现**

<p>纯CSS实现鼠标移动到按钮上打开下拉菜单。</p>

　　<divclass="dropdown">

　　<buttonclass="dropbtn">下拉菜单</button>

　　<divclass="dropdown-content">

　　<ahref="#">菜单1</a>

　　<ahref="#">菜单2</a>

　　<ahref="#">菜单3</a>

　　</div>

　　</div>

　　CSS部分：

　　.dropbtn{

　　background-color:#4CAF50;

　　color:white;

　　padding:16px;

　　font-size:16px;

　　border:none;

　　cursor:pointer;

　　}

　　.dropdown{

　　position:relative;

　　display:inline-block;

　　}

　　.dropdown-content{

　　display:none;

　　position:absolute;

　　background-color:#f9f9f9;

　　width:100%;;

　　box-shadow:0px8px16px0pxrgba(0,0,0,0.2);

　　}

　　.dropdown-contenta{

　　color:black;

　　padding:12px16px;

　　text-decoration:none;

　　display:block;

　　}

　　.dropdown-contenta:hover{

　　background-color:#f1f1f1

　　}

　　.dropdown:hover.dropdown-content{

　　display:block;

　　}

　　.dropdown:hover.dropbtn{

　　background-color:#3e8e41;

　　}

**12、px、em、rem**

PX

px像素（Pixel）。相对长度单位。像素px是相对于显示器屏幕分辨率而言的。

PX特点

1. IE无法调整那些使用px作为单位的字体大小；

2. 国外的大部分网站能够调整的原因在于其使用了em或rem作为字体单位；

3. Firefox能够调整px和em，rem，但是96%以上的中国网民使用IE浏览器(或内核)。

EM

em是相对长度单位。相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸。

EM特点

REM

rem是CSS3新增的一个相对单位（root em，根em），这个单位引起了广泛关注。这个单位与em有什么区别呢？区别在于使用rem为元素设定字体大小时，仍然是相对大小，但相对的只是HTML根元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了IE8及更早版本外，所有浏览器均已支持rem。对于不支持它的浏览器，应对方法也很简单，就是多写一个绝对单位的声明。

1. em的值并不是固定的；

2. em会继承父级元素的字体大小。

**13、图片有哪些格式，区别有哪些**

JPEG格式，也叫做JPG或JPE格式，是最常用的一种文件格式，Photoshop“存储为”命令中默认的图片格式就是JPEG，大部分手机相机拍照的照片也是JPE格式。

JPEG格式的压缩技术十分先进，能够将图像压缩在很小的储存空间，不过这种压缩是有损耗的，过度压缩会降低图片的质量。JPEG格式压缩的主要是高频信息，对色彩的信息保留较好，因此特别适合应用于互联网，可减少图像的传输和加载时间。

“PNG”格式

PNG也是常见的一种图片格式，它最重要的特点是支持 alpha 通道透明度，也就是说，PNG图片支持透明背景。比如在使用Photoshop制作透明背景的圆形logo时，如果使用JPG格式，则图片背景会默认地存为白色，使用PNG格式则可以存为透明背景图片。

PNG格式图片也支持有损耗压缩，虽然PNG 提供的压缩量比JPG少，但PNG图片却比JPEG图片有更小的文档尺寸，因此现在越来越多的网络图像开始采用PNG格式。

“GIF”格式

GIF也是一种压缩的图片格式，分为动态GIF和静态GIF两种。

GIF格式的最大特点是支持动态图片，并且支持透明背景。网络上绝大部分动图、表情包都是GIF格式的，相比与动画，GIF动态图片占用的存储空间小，加载速度快，因此非常流行。

“PSD”格式

PSD格式是Photoshop默认的存储格式，适用于存储源文档和工作文件，修改起来比较方便。

PSD格式的最大特点是可以保留透明度、图层、路径、通道等PS处理信息，但是需要专业的图形处理软件才能打开。PSD格式的缺点是体积庞大，十分占用存储空间。

“TIFF”格式

TIFF格式，也叫做或TIF格式，可以支持不同颜色模式、路径、透明度、以及通道，是打印文档中最常用的格式。Photoshop支持在TIFF文件中保存图层以及其他信息，在很多方面类似于PSD格式文件。

“BMP”格式

BMP 格式是Windows操作系统中的标准图像文件格式，能够被多种Windows应用程序所支持。BMP格式包含的图像信息较丰富，几乎不进行压缩，但由此导致了它占用的存储空间很大，所以，目前BMP在单机上比较流行。

**14、CSS实现三角形**

核心：宽高设置为0，由边框来控制大小，然后边框颜色改为透明，然后更改一遍的边框颜色为自己想要的颜色

div{

width: 0;

height: 0;

border:50px solid transparent;

border-bottom-color:red;

}

注意：上面这样设计的话，div的实际占位还是个方形，如下代码实现实际占位变为三角形

div{

width: 0;

height: 0;

border-left:20px solid transparent;

border-right:20px solid transparent;

border-bottom-color:20px solid red;

}

# 三、JS

**1、JavaScript的数据类型（ES6新增的数据类型）**

五种基本数据类型/值类型：String 字符串、 Number 数值、 Boolean 布尔值、 Null 空值、Undefined 未定义

一种引用/复杂数据类型： Object 对象（数组、函数、正则表达式、Date等）

一种ES6新增数据类型： Symbol 表示独一无二的值

**2、数据隐式转换，数据类型判断**

数据类型判断

数据类型有 number string boolean null undefined function object 这些，那如何去判断数据对应的是哪种类型呢

typeof可以判断对应数据类型 写法： typeof data 或者 typeof(data) ， 一般习惯写后面这种

返回值有 number string boolean object undefined function 这些

其中对于数组和对象无法区分，可以用 instanceof 去判断

console.log([] instanceof Array) // true

console.log({} instanceof Array) // false

如果只用其中一种方法就可以判断，则需要通过 toString() 方法，对 对象原型调用 toString()方法，通过改变this指向来获得对应的类型，如下

复制代码

console.log(Object.prototype.toString.call('123')) //[object String]

console.log(Object.prototype.toString.call({})) //[object Object]

console.log(Object.prototype.toString.call([])) //[object Array]

console.log(Object.prototype.toString.call(123)) //[object Number]

console.log(Object.prototype.toString.call(true)) //[object Boolean]

console.log(Object.prototype.toString.call(null)) //[object Null]

复制代码

但是对undefined类型会报错，所以undefined类型只有在typeof方法中才不会报错

数据显示转换

显示转换number， number转换有Number() parseInt() parseFloat() 方法

复制代码

console.log(Number('123')); //123

console.log(Number(true)); //1

console.log(Number(false)); //0

console.log(Number(NaN)); //NaN

console.log(Number('abc')); //NaN

console.log(Number(undefined)); //NaN

console.log(parseInt('123abc')) //123 只为变成整形，只看数字位，直到非数字位就截断

console.log(parseInt('abc')) //NaN

console.log(parseInt(true)) //NaN

console.log(parseInt(10, 16)) //16 表示10为基底，这是16进制，转为10进制

console.log(parseFloat('100.2')) //100.2 只为转换为浮点型

console.log(parseFloat('100.2abc')) //100.2

console.log(parseFloat('100.2.3')) //100.2

复制代码

其它

String() 显示转换字符串

Boolean() 显示转换布尔值

toSting() 转换字符串 undefined和null不能用toString()

var a = 10

console.log(a.toString()) //123

console.log(a.toString(2)) //1010 表示以a为基底，转为2进制

console.log(a.toString(16)) //a 表示以a为基底，转为16进制

数据隐式转换

复制代码

// isNaN()

console.log(isNaN(123)) //false

console.log(isNaN('123')) //false

console.log(isNaN('abc')) //true 先隐式转换成Number类型

// + - / % \*

console.log(+'123') //123 number类型

console.log(false < 1) //true

console.log(2 < 1 < 3) //true 2 < 1为false flase < 3为true

console.log(2 < 3 < 1) //false 2 < 3为true true < 1为false

console.log(undefined < 0) //false

console.log(undefined > 0) //false

console.log(undefined == 0) //false

console.log(null < 0) //false

console.log(null > 0) //false

console.log(null == 0) //false

console.log(undefined == null) //true

复制代码

不发生隐式转换

// === 全等 !== 全不等

console.log('123' === 123) //false

**3、== 和 === 的区别**

简单来说： == 代表相同， ===代表严格相同, 为啥这么说呢，

这么理解： 当进行双等号比较时候： 先检查两个操作数数据类型，如果相同， 则进行===比较， 如果不同， 则愿意为你进行一次类型转换， 转换成相同类型后再进行比较， 而===比较时， 如果类型不同，直接就是false.

操作数1 == 操作数2， 操作数1 === 操作数2

比较过程：

　　双等号==：

　　（1）如果两个值类型相同，再进行三个等号(===)的比较

　　（2）如果两个值类型不同，也有可能相等，需根据以下规则进行类型转换在比较：

　　　　1）如果一个是null，一个是undefined，那么相等

　　　　2）如果一个是字符串，一个是数值，把字符串转换成数值之后再进行比较

　　三等号===:

　　（1）如果类型不同，就一定不相等

　　（2）如果两个都是数值，并且是同一个值，那么相等；如果其中至少一个是NaN，那么不相等。（判断一个值是否是NaN，只能使用isNaN( ) 来判断）

　　（3）如果两个都是字符串，每个位置的字符都一样，那么相等，否则不相等。

　　（4）如果两个值都是true，或是false，那么相等

　　（5）如果两个值都引用同一个对象或是函数，那么相等，否则不相等

　　（6）如果两个值都是null，或是undefined，那么相等

**4、ES6的了解**

<https://blog.csdn.net/liang377122210/article/details/77547584>

**5、事件委托**

（1）事件流

HTML中与javascript交互是通过事件驱动来实现的，例如鼠标点击事件onclick、页面的滚动事件onscroll等等，可以向文档或者文档中的元素添加事件侦听器来预订事件。想要知道这些事件是在什么时候进行调用的，就需要了解一下“事件流”的概念。

什么是事件流：从页面中接收事件的顺序

事件捕获阶段

处于目标阶段

事件冒泡阶段

addEventListener：addEventListener 是DOM2 级事件新增的指定事件处理程序的操作，这个方法接收3个参数：要处理的事件名、作为事件处理程序的函数和一个布尔值。最后这个布尔值参数如果是true，表示在捕获阶段调用事件处理程序；如果是false，表示在冒泡阶段调用事件处理程序。

IE只支持事件冒泡。

（2）如何让事件先冒泡后捕获

在DOM标准事件模型中，是先捕获后冒泡。但是如果要实现先冒泡后捕获的效果，对于同一个事件，监听捕获和冒泡，分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获事件。

（3）事件委托

简介：事件委托指的是，不在事件的发生地（直接dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素DOM的类型，来做出不同的响应。

举例：最经典的就是ul和li标签的事件监听，比如我们在添加事件时候，采用事件委托机制，不会在li标签上直接添加，而是在ul父元素上添加。

好处：比较适合动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制。

**6、JS的继承（多种实现方式）**

<https://editor.csdn.net/md/?articleId=112974001>

1. **什么是原型链，原型链的继承，怎么防止原型链继承创建对象全都继承父类的属性**

什么是原型链？

JavaScript是面向对象的，每个实例对象都有一个\_\_proto\_\_属性，该属性指向它的原型对象，这个实例对象的构造函数有一个原型属性prototype,与实例的proto属性指向同一个对象。当一个对象在查找一个属性的时候，自身没有就会根据\_\_proto\_\_向它的原型进行查找，如果都没有，则向它的原型的原型继续查找，直到查到Object.prototype.proto\_为null,这样也就形成了原型链。

寄生组合继承（防止原型链上属性共享）

核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

1. **作用域链**

一般情况下，变量取值到 创建 这个变量 的函数的作用域中取值。

　　但是如果在当前作用域中没有查到值，就会向上级作用域去查，直到查到全局作用域，这么一个查找过程形成的链条就叫做作用域链。

1. **高性能数组去重的方法？（隐式转换，高频考点）**

利用ES6 Set去重（ES6中最常用）

利用for嵌套for，然后splice去重（ES5中最常用）

利用indexOf去重

利用sort()

利用includes

利用hasOwnProperty

**10、数组的方法有哪些，举例，有哪些是对数组进行遍历**

**11、讲一下闭包用到什么场景，能够解决什么问题，你在项目中用到哪个地方，具体讲一下可以吗**

**12、对Promise的了解**

**13、手写一个Promise all**

**14、DOM结点树的增删改查**

**15、防抖和节流**

防抖背后的思路是指，减少函数在一定时间段内连续调用的次数，只执行一次

节流是让连续执行的函数，变为固定时间段间断地执行

**16、如何判断一个对象为空？**

1）遍历属性法

一旦该对象的属性是可遍历的即说明该对象存在属性，则返回false，否则该对象为空对象。

function judgeObj(obj){

for(var attr in obj){

return false

}

return true

}

console.log(judgeObj({}))

（2）JSON.stringify() 方法

function judgeObj2(obj){

if(JSON.stringify(obj) == '{}')

return true

else

return false

}

console.log(judgeObj2({}))

（3）ES6 Object.keys()方法

Object.keys方法是JavaScript中用于遍历对象属性的一个方法 。

它传入的参数是一个对象，返回的是一个数组，数组中包含的是该对象所有的属性名。

function judgeObj3(obj) {

if(Object.keys(obj).length == 0){

return true

} else {

return false

}

}

**17、for...in 和for...of 的区别**

for in ：ES5标准 遍历对象得到键名key（直接拿到对象的属性名，再根据属性名获取属性值）；不建议用来遍历数组（因为输出顺序不固定）

for of ：ES6标准 遍历数组（数组、类数组对象，字符串、Set、Map 以及 Generator 对象）得到键值value；不建议用来遍历对象

来个直观简要的图例吧 ~

发现for of 遍历对象报错了——for of 只能用在可迭代对象上，而obj1 不可迭代

为什么 for in 可以呢？——for in 的原理是 Object.keys()：返回给定对象所有可枚举属性的字符串数组

若要用 for of 遍历对象，要结合Object.keys()进行使用：

for (let e of Object.keys(obj1)) {

console.log(e); // 键名

console.log(obj1[e]); // 键值（使用的是对象属性访问方法）

}

回顾该部分的知识点，附上我的思维导图 note：

# 四、浏览器相关

**1、从浏览器输入一个url到显示页面经历的过程**

一次完整的HTTP请求过程 版本 1.0

浏览器对输入的网址进行DNS解析，得到对应的IP地址

根据这个IP，找到对应的服务器，发起TCP连接（TCP的三次握手）

建立TCP链接后，发起HTTP请求

服务器响应HTTP请求，浏览器得到 html 代码

浏览器解析 html 代码，并请求代码中的资源（js、css、图片等）--先得到 html 代码，才能去找这些资源

浏览器对页面进行渲染，呈现给用户

服务器关闭TCP连接（TCP的四次挥手）

HTTP请求过程--复习推演

1、url（www.baidu.com）=> ip (192.168.1.2)

2、拿到ip，与服务器建立TCP连接

3、TCP连接后，发送HTTP请求 （TCP是比http更底层的协议，ip是tcp下面一层）

4、HTTP连接后，服务器把html发给浏览器

5、浏览器解析html（js、css、图片 下载 运行）

6、浏览器渲染html

7、服务器关闭连接（TCP HTTP）

注：

（1）DNS怎么找到域名的？

DNS域名解析采用的是递归查询的方式

过程：先去找DNS缓存->缓存找不到就去找根域名服务器->根域名又会去找下一级，这样递归查找之后，找到了，给我们的web浏览器

过程（面试）：浏览器自身域名缓存区找 =》操作系统的域名缓存区找 =》hosts文件找 =》域名服务器找

（2）为什么HTTP协议要基于TCP来实现？

TCP是一个端到端的可靠的面相连接的协议，HTTP基于传输层TCP协议，不用担心数据传输的各种问题（当发生错误时，会重传）

（3）最后一步浏览器是如何对页面进行渲染的？

（4）TCP的三次握手和四次挥手

三次握手：

第一次握手：建立连接时，客户端发送syn包（syn=x）到服务器，等待服务器确认

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=y+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手

四次挥手：

第一次挥手：客户端进程发出连接释放报文（FIN报文），并且停止发送数据

第二次挥手：服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v

客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）

第三次挥手：服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1

第四次挥手：客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1。服务器只要收到了客户端发出的确认，立即进入CLOSED状态。

**2、浏览器加载过程，着重讲一下前端如何渲染到页面（浏览器运行机制）**

构建DOM树：渲染引擎解析HTML文档，将标签转换成DOM节点生成DOM树

构建渲染树：解析CSS文件，构建渲染树（Render树），使每个节点有自己的样式

布局渲染树：从根节点（body）递归调用，布局出每个节点在屏幕出现的位置

绘制渲染树：遍历渲染树，使用UI层绘制每个节点 ，呈现界面

**3、对重绘和回流的了解**

（1）概念

重绘：元素外观（如背景颜色）改变引起的浏览器行为，使元素外观重新绘制

回流（重排）：渲染树中的元素尺寸、布局、隐藏等改变，需要重新构建

（2）注意点

每个页面至少发生一次回流，即页面第一次加载的时候

重排必定引发重绘，重绘不一定引发重排

（3）补充

触发重排的条件：任何页面布局或几何属性的改变都会触发重排

页面渲染初始化（无法避免）

添加或删除可见的DOM元素

元素位置的改变，或使用动画

元素尺寸的改变（大小，外边距，边框）

浏览器窗口尺寸的变化（resize事件）

填充内容的改变（文本或图片大小改变，引起计算值宽高改变）

读取某些元素属性（offsetLeft/Top/Height/Width, clientLeft/Top/Height/Width,scrollLeft/Top/Height/Width等）

（4）扩展

重绘重排会造成耗时、浏览器卡顿，那么如何做优化呢？（尽量减少DOM操作）

1、浏览器优化：浏览器会把引起回流、重绘的操作放入一个队列，等队列中的操作到了一定数量或者时间间隔，就flush这个队列进行一个批处理。这样就可以让多次回流重绘变成一次。

2、代码优化：减少对渲染树的操作，可以合并多次DOM和样式的修改，并减少对style样式的请求。

修改元素样式的时候，直接修改样式名className（尽量一次修改元素样式，不要一会改一点。也就是把新样式放在另一个样式名中）

某些元素先设置成display: none，然后进行页面布局操作，再设置display: block（这样只会引发两次重绘重排）

使用cloneNode和repalceChild技术（引发一次重绘重排）

将需要多次重排的元素，position属性设为absoluted或fixed（元素脱离文档流，变化不会影响其他元素）

当需要创建多个节点的时候，使用DocumentFragment创建完后一次性的加入（如循环创建一个li，让循环结束后所有的li都创建完了（fragment中）再一次性加入（文档））

1. **什么是同源策略**

浏览器对不同源的脚本或者文本的访问方式进行的限制。比如源a的js不能读取或设置引入的源b的元素属性。那么先定义下什么是同源,所谓同源,就是指两个页面具有相同的协议,主机(也常说域名),端口,三个要素缺一不可。”

1. **什么是跨域，如果没有同源机制会怎样，具体说一下**

定义了一种浏览器和服务器交互的方式来确定是否允许跨域请求。

**6、跨域有几种方式（JSONP跨域的局限性）**

**7、jsonp的原理**

**8、cookie，session，sessionstorage，localstorage的区别**

Web storage（HTML5）

cookie session sessionstorage localstorage

数据生命周期 一般由服务器生成，在设置失效时间（expires）内有效（与窗口或浏览器是否关闭无关）。若在浏览器设置，默认浏览器关闭后失效 除非被清除，否则永久保存（刷新页面数据依旧存在） 仅在当前会话有效，关闭窗口或浏览器后清除 除非web应用主动删除，否则永不失效

存放数据大小 4k左右 5M 5M

与服务端通信 保存在浏览器端。每次都会携带在HTTP请求头中，保存过多数据会有性能问题 保存在服务器端，不参与服务器通信 保存在客户端，不参与服务器通信

安全性 安全性较低（cookie诈骗cookie截取） session安全性大于cookie

易用性 一般接口需要自己封装 接口可以直接使用

作用域 在浏览器所有的同源窗口中共享 不能在不同的浏览器窗口共享 不能在不同的浏览器窗口共享，在同源窗口中可以共享

使用场景

主要用于保存登录信息

1、判断用户是否登录过网站，方便下次登录实现自动登录或记住密码

2、上次登录的时间等信息

3、上次查看的页面

4、浏览计数

用于保存每个用户的专用信息，变量的值保存在服务器端，通过sessionID来区分不同用户。

1、购物车

2、用户登录信息

3、将某些数据放入session中，供同一用户的不同页面使用

4、防止用户非法登录

敏感账号一次性登录、表单，对于那种只需要在用户浏览一组页面期间保存而关闭浏览器后就可以丢弃的数据，sessionStorage会非常方便

常用于长期登录（判断用户是否登录），适合长期保存在本地的数据。

购物车信息、HTML5游戏产生的一些本地数据

优点

1、存储空间大

2、节省网络流量

3、可在本地直接获取，不需要与服务器交互

4、获取速度快

5、安全性较高

6、更多丰富易用的API接口

7、支持事件通知机制，可以将数据更新的通知发送给监听者

8、操作方法：setItem、getItem、removeItem、clear、key、length

缺点

1、大小受限

2、用户可以禁用cookie，使功能受限

3、安全性较低

4、有些状态不能保存在客户端

5、同源请求时会被携带（服务端和客户端互传，不论是否需要），加大http流量，数据过多影响性能

6、cookie数据有路径（path）的概念，可以限制cookie只属于某个路径下

**9、原生ajax的实现步骤**

**10、get和post的区别**

参数：最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数

操作数据：语义上对于GET用来获取数据，POST用来提交的数据

大小：GET提交的数据（URL长度）大小受限（与HTTP协议无关，受浏览器或web服务器影响），而POST则没有此限制

缓存方面的区别：

get请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。

post不同，post做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此get请求适合于请求缓存。

**11、常见的状态码以及意思**

状态码：状态码用于表示服务器对请求的处理结果

1xx：指示信息——表示请求已经接受，继续处理

2xx：成功——表示请求已经被成功接收、理解、接受。

3xx：重定向——要完成请求必须进行更进一步的操作

4xx：客户端错误——请求有语法错误或请求无法实现

5xx：服务器端错误——服务器未能实现合法的请求。

列举几种常见的：

200（没有问题）

302（要你去找别人）

304（要你去拿缓存）

307（要你去拿缓存）

403（有这个资源，但是没有访问权限）

404（服务器没有这个资源）

500（服务器这边有问题）

**12、浏览器的缓存机制 包括缓存位置分类类型 强缓存、协商缓存**

面试：性能提升 =》减少HTTP请求 =》HTTP缓存控制（浏览器缓存）？

回答：

http缓存能够帮助服务器提高并发性能，很多资源不需要重复请求，直接从浏览器中拿缓存

http缓存分类：强缓存、协商缓存

强缓存通过 Expires 和Cache-control 控制，协商缓存通过 Last-modify 和 Etag 控制

补充问题：

1、为什么有 expires 还要 cache-control

因为expires有服务器和浏览器时间不同步问题，expires是绝对时间，cache-control是相对时间

2、为什么有 last-modify 还要 etag

因为 last-modify 有精度问题 到秒，etag 没有精度问题，只要资源改变，etag值就会改变

详细理解：

（1）浏览器缓存机制

浏览器每次发起请求，都会先在浏览器缓存中查找该请求的结果以及缓存标识

浏览器每次拿到返回的请求结果都会将该结果和缓存标识存入浏览器缓存中

（2）浏览器缓存策略

缓存策略都是通过设置 HTTP Header 来实现的，根据是否需要向服务器重新发起HTTP请求将缓存过程分为两个部分，分别是强缓存和协商缓存。

1、浏览器加载一个页面的简单流程如下：

浏览器先根据这个资源的http头信息来判断是否命中强缓存。如果命中则直接加在缓存中的资源，并不会将请求发送到服务器。（强缓存）

如果未命中强缓存，则浏览器会将资源加载请求发送到服务器。服务器来判断浏览器本地缓存是否失效。若可以使用，则服务器并不会返回资源信息，浏览器继续从缓存加载资源。（协商缓存）

如果未命中协商缓存，则服务器会将完整的资源返回给浏览器，浏览器加载新资源，并更新缓存。（新的请求）

2、强缓存：不会向服务器发送请求，直接从缓存中读取资源

强缓存可以通过设置两种 HTTP Header 实现：Expires 和 Cache-Control

Expires（旧 http1.0）：缓存过期时间（绝对时间），用来指定资源到期的时间，是服务器端的具体的时间点（在响应http请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求）。受限于本地时间，如果修改了本地时间，可能会造成缓存失效。

Cache-Control（新 http1.1）：是一个相对时间，代表资源的有效期。

同时启用时cashe-control优先级更高

强缓存判断是否缓存的依据来自于是否超出某个时间或者某个时间段，而不关心服务器端文件是否已经更新，这可能会导致加载文件不是服务器端最新的内容，那我们如何获知服务器端内容是否已经发生了更新呢？此时我们需要用到协商缓存策略。

3、协商缓存：强制缓存未命中，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程。主要有以下两种情况：

协商缓存生效，返回304和Not Modified

协商缓存失效，返回200和请求结果

协商缓存--协商个啥呢？

浏览器问服务器，我缓存的文件有没有更新啊？

没有更新：浏览器可以用缓存 304

更新了：浏览器不能用缓存，服务器发新的给浏览器 200

协商缓存可以通过设置两种 HTTP Header 实现：Last-Modified 和 ETag （帮助浏览器跟服务器进行协商）

Last-Modified：浏览器第一次请求一个资源的时候，服务器返回的header中会加上Last-Modify，Last-modify是一个时间标识该资源的最后修改时间 。当浏览器再次请求该资源时，发送的请求头中会包含If-Modify-Since，该值为缓存之前返回的Last-Modify。服务器收到If-Modify-Since后，根据资源的最后修改时间判断是否命中缓存。如果命中缓存，则返回http304，并且不会返回资源内容，并且不会返回Last-Modify。由于对比的服务端时间，所以客户端与服务端时间差距不会导致问题。但是有时候通过最后修改时间来判断资源是否修改还是不太准确（资源变化了最后修改时间也可以一致，精度问题 到秒）。于是出现了ETag/If-None-Match。

Etag（http1.1）：服务器响应请求时，返回当前资源文件的一个唯一标识(由服务器生成)，Etag/If-None-Match返回的是一个校验码（ETag: entity tag） 。ETag可以保证每一个资源是唯一的，只要资源有变化，Etag就会重新生成。服务器根据浏览器上发送的If-None-Match值来判断是否命中缓存。

4、HTTP1.1中Etag的出现主要是为了解决几个Last-Modified比较难解决的问题：

Last-Modified标注的最后修改只能精确到秒级，如果某些文件在1秒钟以内，被修改多次的话，它将不能准确标注文件的修改时间

如果某些文件会被定期生成，当有时内容并没有任何变化，但Last-Modified却改变了，导致文件没法使用缓存

有可能存在服务器没有准确获取文件修改时间，或者与代理服务器时间不一致等情形

Etag是服务器自动生成或者由开发者生成的对应资源在服务器端的唯一标识符，能够更加准确的控制缓存。Last-Modified与ETag是可以一起使用的，服务器会优先验证ETag，一致的情况下，才会继续比对Last-Modified，最后才决定是否返回304。

5、缓存机制：强制缓存优先于协商缓存进行，若强制缓存(Expires和Cache-Control)生效则直接使用缓存，若不生效则进行协商缓存(Last-Modified / If-Modified-Since和Etag / If-None-Match)，协商缓存由服务器决定是否使用缓存，若协商缓存失效，那么代表该请求的缓存失效，返回200，重新返回资源和缓存标识，再存入浏览器缓存中；生效则返回304，继续使用缓存。

6、如果什么缓存策略都没设置，那么浏览器会怎么处理？

对于这种情况，浏览器会采用一个启发式的算法，通常会取响应头中的 Date 减去 Last-Modified 值的 10% 作为缓存时间。

（3）实际应用场景

频繁变动的资源

不常变化的资源

（4）用户行为对浏览器缓存的影响

新开窗口 有效 有效

用户操作 Expires/Cache-Control Last-Modified/Etag

地址栏回车 有效 有效

页面链接跳转 有效 有效

前进、后退 有效 有效

F5 刷新 无效 有效

Ctrl+F5 刷新 无效 无效

**13、TCP 和 UDP的区别？**

TCP是面向连接的，UDP是无连接的。TCP需要有三次握手连接，UDP不需要

TCP提供可靠的服务，保证通过TCP连接传输的数据不会丢失、不会重复、保证传输有序，但不保证按序到达。（TCP：确认与重传机制、拥塞控制、流量控制）UDP不提供可靠性服务。

TCP是面向字节流的，UDP是面向报文段的（一次发送一个报文）。

UDP传输比TCP快，TCP首部开销20字节、UDP首部开销8字节

**14、HTTP所有的请求方法类型？**

HTTP1.0 中定义的方法：

GET: 完整请求一个资源 （常用）

HEAD: 仅请求响应首部

POST：提交表单 （常用 ）

HTTP1.1 中新增的5种方法：

PUT: (webdav) 上传文件（但是浏览器不支持该方法）

DELETE：(webdav) 删除

OPTIONS：返回请求的资源所支持的方法的方法

TRACE: 追求一个资源请求中间所经过的代理（该方法不能由浏览器发出）

CONNECT

**15、什么是URL、URI、URN？**

URI Uniform Resource Identifier 统一资源标识符

URL Uniform Resource Locator 统一资源定位符

URN Uniform Resource Name 统一资源名称

URL和URN 都属于 URI，为了方便就把URL和URI暂时都通指一个东西

# 五、VUE

**1、vue生命周期，挂载**

**2、vue组件之间如何通信**

**3、Vue双向绑定的原理**

**4、VUE Router有几类，各自的原理**

**5、虚拟dom的优势和diff算法**

六、数据结构与算法

1、二叉树的遍历，已知前序遍历12473568，中序遍历47215386，请问后序遍历

74256831

2、知道的排序算法，仔细讲下快速排序和归并排序

3、反转链表的原理思路，每一步是怎么一个结果，详细讲一下

4、查找一个数组中的a元素最长连续出现次数

5、动态规划，跳台阶

6、数组和链表的区别

7、合并两个有序数组

var merge = function(nums1, m, nums2, n) {

let nums = []

let i = 0

let j = 0

while(i < m && j < n){

if(nums1[i] <= nums2[j]){

nums.push(nums1[i++])

} else {

nums.push(nums2[j++])

}

}

while(i < m){

nums.push(nums1[i++])

}

while(j < n){

nums.push(nums2[j++])

}

return nums

};

8、假设有一个大数组arr，长度很大，几百万个，循环执行pop方法，怎么样提高效率？

9、冒泡排序

10、树形结构与数组之间的转换

数组转树形结构

我们先建立这样一个对象数组

const arr = [

{

id: '1',

parent\_id: 'root',

name: 'hh'

},

{

id: '2',

parent\_id: 'root',

name: 'hh'

},

{

id: '1-1',

parent\_id: '1',

name: 'hh'

},

{

id: '1-2',

parent\_id: '1',

name: 'hh'

},

{

id: '1-1-1',

parent\_id: '1-1',

name: 'hh'

},

{

id: '1-1-2',

parent\_id: '1-1',

name: 'hh'

},

{

id: '1-2-1',

parent\_id: '1-2',

name: 'hh'

},

{

id: '2-1',

parent\_id: '2',

name: 'hh'

},

{

id: '2-2',

parent\_id: '2',

name: 'hh'

},

{

id: '2-1-1',

parent\_id: '2-1',

name: 'hh'

},

{

id: '2-2-1',

parent\_id: '2-2',

name: 'hh'

},

{

id: '2-2-1-1',

parent\_id: '2-2-1',

name: 'hh'

},

{

id: '2-2-1-2',

parent\_id: '2-2-1',

name: 'hh'

},

{

id: '2-2-1-2-1',

parent\_id: '2-2-1-2',

name: 'hh'

},

{

id: '2-3',

parent\_id: '2',

name: 'hh'

},

{

id: '2-3-1',

parent\_id: '2-3',

name: 'hh'

},

{

id: '3',

parent\_id: 'root',

name: 'hh'

},

];

这是一个扁平化的数组，我们创建一个临时对象，再做一下处理

function arrayToTree(array, id, parent\_id) {

let temp = {}; // 创建临时对象

let tree = {}; // 创建需要返回的树形对象

// 构造temp对象节点，如：

// '2-3': { id: '2-3', parent\_id: '2',... }

for (let i in array) { // 先遍历数组，将数组的每一项添加到temp对象中

temp[array[i][id]] = array[i];

}

console.log(temp);

return tree;

}

打印看看结果如何

基于此开始创建树形结构

function arrayToTree(array, id, parent\_id) {

let temp = {}; // 创建临时对象

let tree = {}; // 创建需要返回的树形对象

// 构造temp对象节点，如：

// '2-3': { id: '2-3', parent\_id: '2',... }

for (let i in array) { // 先遍历数组，将数组的每一项添加到temp对象中

temp[array[i][id]] = array[i];

}

console.log(temp);

// 遍历temp对象，将当前子节点与父节点建立连接, 并建立树形结构

for (let i in temp) {

if (temp[i][parent\_id] !== 'root') { // 判断是否是根节点下的项

if (!temp[temp[i][parent\_id].children]) {

temp[temp[i][parent\_id]].children = new Array();

}

temp[temp[i][parent\_id]].children.push(temp[i]);

} else {

tree[temp[i][id]] = temp[i];

}

}

return tree;

}

const obj = arrayToTree(arr, 'id', 'parent\_id');

console.log(obj);

打印结果

树形结构转数组

例如这样一个树形结构

const treeObj = {

id: '0',

name: '0',

children: [

{

id: '1',

name: 'cc',

children: [

{

id: '1-1',

name: 'cc',

children: [

{

id: '1-1-1',

name: 'cc',

},

{

id: '1-1-2',

name: 'cc'

},

]

},

{

id: '1-2',

name: 'cc',

children: [

{

id: '1-2-1',

name: 'cc',

},

{

id: '1-2-2',

name: 'cc'

},

]

},

]

},

{

id: '2',

name: 'cc',

children: [

{

id: '2-1',

name: 'cc',

children: [

{

id: '2-1-1',

name: 'cc',

},

{

id: '2-1-2',

name: 'cc'

},

]

},

{

id: '2-2',

name: 'cc',

children: [

{

id: '2-2-1',

name: 'cc',

children: [

{

id: '2-2-1-1',

name: 'cc',

},

{

id: '2-2-1-2',

name: 'cc'

},

]

},

{

id: '2-2-2',

name: 'cc'

},

]

},

{

id: '2-3',

name: 'cc',

children: [

{

id: '2-3-1',

name: 'cc',

},

{

id: '2-3-2',

name: 'cc'

},

]

},

]

},

{

id: '3',

name: 'cc',

children: []

}

]

};

代码实现

// 将treeObj中的所有对象，存入一个数组中，并根据 id 关联起父子关系

function treeToArray(treeObj, rootid) {

const temp = []; // 设置临时数组，用来存放队列

const out = []; // 设置输出数组，用来存放要输出的一维数组

temp.push(treeObj);

// 首先把根元素存放入out中

let pid = rootid;

const obj = deepCopy(treeObj);

obj.pid = pid;

delete obj['children'];

out.push(obj)

// 对树对象进行广度优先的遍历

while (temp.length > 0) {

const first = temp.shift();

const children = first.children;

if (children && children.length > 0) {

pid = first.id;

const len = first.children.length;

for (let i = 0; i < len; i++) {

temp.push(children[i]);

const obj = deepCopy(children[i]);

obj.pid = pid;

delete obj['children'];

out.push(obj)

}

}

}

return out

}

// 深拷贝

function deepCopy(obj) {

// 深度复制数组

if (Object.prototype.toString.call(obj) === "[object Array]") {

const object = [];

for (let i = 0; i < obj.length; i++) {

object.push(deepCopy(obj[i]))

}

return object

}

// 深度复制对象

if (Object.prototype.toString.call(obj) === "[object Object]") {

const object = {};

for (let p in obj) {

object[p] = obj[p]

}

return object

}

}

console.log(treeToArray(treeObj, 'root'))

结果打印如下

# 七、其他

**1、对懒加载的了解，如何做到懒加载？**

**2、命名法有哪些？**

**3、如何实现头像的放大缩小、剪裁、图片懒加载是怎实现的？**

**4、Webpack打包的流程？**

**5、Webpack如何提高打包效率、减少压缩体积？**

**6、你的项目是如何构建的?**

**7、页面长时间出现白屏如何排查问题？如果不能使用开发环境如何定位问题？**