ddd一、HTML

1. 标签语义化：

不同的标签代表不同的结构，便于开发者阅读和编写出更易理解的代码，有利于团队开发和维护，在没有CSS的情况下，页面也能呈现出较容易分辨的结构，提高用户体验，有利于SEO和解析。

1. 常用标签的默认样式

块级元素：html,body,div,ol,ul,p,h1,h2,h3,h4,h5,h6,address,blockquote,form,dd,dl,dt,fieldset,frame,frameset,noframes,center,dir,hr,menu,pre {display:block};

列表元素：

li{display:list-item,list-style:disc}

ol{list-style-type:decimal}

ol ul,ul ol,ul ul,ol ol{margin-top:0;margin-bottom:0}

ol,ul{margin-left:40px}

预格式文本：

i,cite,em,var,address{font-style:italic}

big{font-size:1.17em}

small,sub,sup{font-size:.83em}

sub/sup{vertical-align:sub/super}

u,ins{text-decoration:underline}

标题

h1{font-size:2em;margin:.67em 0}

h2{font-size:1.5em;margin:.75em 0}

h3{font-size:1.17em;margin:.83em 0}

h4,p,blockquote,ul,fieldset,form,ol,dl,dir,menu{margin:1.12em 0}

h5{font-size:.83em;margin:1.5em 0}

h6{font-size:.75em;margin:1.67em 0}

h1,h2,h3,h4,h5,h6,b,strong{font-weight:bolder}

表格

table{display:table}

tr{display:table-row}

thead{display:table-header-group}

tbody{display:table-row-group}

tfoot{display:table-footer-group}

col{display:table-column}

其他

head{display:none}

body{margin:8px;line-height:1.12em}

button,textarea,input,object,select{display:inline-block}

blockquote{margin-left:40px;margin-right:40px;}

pre,tt,code,kbd,samp{font-family:monospace}

pre{write-space:pre}

hr{border:1px inset}

center{text-align:center}

abbr,acronym{font-variant:small-caps;letter-spacing:0.1em}

注：剩余的以后补充

1. 常用标签属性

<https://www.cnblogs.com/yt4561761/p/6538928.html>

1. 浏览器兼容

https://www.cnblogs.com/simba-lkj/p/6137494.html

1. HTML5离线缓存机制
2. 当浏览器访问一个包含manifest特性的文档时，如果应用缓存不存在，浏览器会加载文档，然后获取所有在清单文件中列出的文件，生成应用缓存的第一个版本。
3. 对该文档的后续访问会使浏览器直接从应用缓存中加载文档或其他在清单文件中列出的资源。此外，浏览器还会向window.applicationCache对象发送一个checking事件，在遵循合适的HTTP缓存规则前提下，获取清单文件。
4. 如果当前缓存的清单副本是最新的，浏览器将向applicationCache对象发送一个noupdate事件，到此，更新过程结束。（注：如果在服务器修改了任何缓存资源，同时也应该修改清单文件，这样浏览器才能知道它需要重新获取资源）。
5. 如果清单文件已经改变，文件中列出的所有文件包括通过调用applicationCache.add()方法添加到缓存中的那些文件都会被获取并放到一个临时缓存中，遵循适当的HTTP缓存规则。对于每个加入到临时缓存中的文件，浏览器会向applicationCache对象发送一个progress事件。如果出现任何错误，浏览器会发送一个error事件，并暂停更新。
6. 一旦所有文件都获取成功，它们会自动移送到真正的离线缓存中，并向applicationCache对象发送一个cached事件。由于文档已经被从缓存加载到浏览器中，所以更新后的文档不会重新渲染，直到页面重新加载。

详见http://www.ituring.com.cn/article/213932

1. CSS

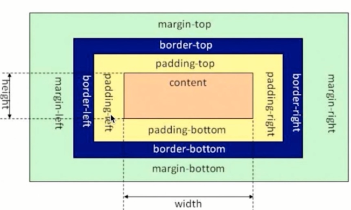
1.BFC(Block Formatting Context)：

BFC即块级格式化上下文：是一个独立的布局环境，其中的元素布局不受外界影响；内部的box(css布局的基本单位)会在垂直的方向一个接一个的放置；box垂直方向的距离由margin决定，属于同一个BFC的两个相邻box的margin会发生重叠；每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)，即使存在浮动也是如此；BFC的区域不会与浮动区域的box重叠；浮动元素参与BFC高度的计算。

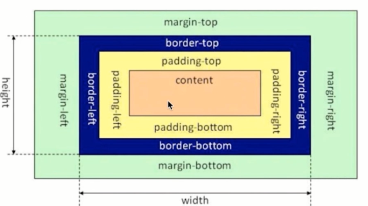
BFC的创建:

1. float属性不为none；
2. position为absolute或fixed
3. display为inline-block,table-cell,table-caption,flex,inline-flex;
4. Overflow不为visible
5. 根元素
6. 盒模型

标准盒模型(box-sizing:content-box)



IE盒模型(box-sizing:border-box)



1. css选择器
2. 标签选择器
3. 类选择器
4. ID选择器
5. 通配符
6. 组合选择器
7. 子元素选择器（>）
8. 群组选择器
9. 继承选择器
10. 伪类选择器
11. 兄弟选择器
12. 属性选择器

优先级：!important>行内样式>ID选择器>类选择器>标签选择器>通配符>继承选择器>浏览器默认属性

Css继承：https://www.cnblogs.com/super86/p/4121824.html

At规则：<https://blog.csdn.net/scorpio_h/article/details/86799790#>

1. css伪类、伪元素

伪类：用于向某些选择器设置特殊效果(弥补常规选择器的不足)

详见http://www.w3school.com.cn/css/css\_pseudo\_classes.asp和 <http://www.imooc.com/wenda/detail/460416>

伪元素：用于向某些选择器设置特殊的效果(创建一个有内容的虚拟容器，如清浮动，同时只能使用一个伪元素)

详见：<http://www.w3school.com.cn/css/css_pseudo_elements.asp>

1. HTML文档流

元素排版过程中，元素会自动从左往右，从上往下的流式排列。

定位：<http://www.w3school.com.cn/css/css_positioning.asp>

1. 雪碧图实现原理

将小图标和背景图像合并到一张图片上，利用CSS的背景定位来显示需要的图片部分

详见：<https://www.cnblogs.com/xuzewen/p/9310007.html>

1. 水平垂直居中方案

水平：

1. 父元素{text-align:center}，子元素{display:inline-block}
2. 子元素{display:table;margin:0 auto}
3. 父元素{position:relative}，子元素{display:absolute;left:50%;transform:translateX(-50%)}
4. ①父元素{display:flex;justify-content:center}

②父元素{display:flex}，子元素{margin:0 auto}

垂直：

1. 父元素{display:table-cell;vertical-align:middle}
2. 父元素{position:relative}，子元素{position:absolute;top:50%;transform:translateY(-50%)}
3. 父元素{display:flex;align-items:center}

详见：<https://segmentfault.com/a/1190000016389031>

1. sass和less

Sass(scss)：是一种动态语言，属于缩排语法

详见：<https://www.sass.hk/guide/>

Less：是一种动态语言...

详见：https://less.bootcss.com/

Sass和less的异同

不同：less基于JavaScript，是在客户端处理的，变量用”@”

Sass基于Ruby，是在服务器端处理的”$”

相同：

1.混入(Mixins)——class中的class；

2.参数混入——可以传递参数的class，就像函数一样；  
 3.嵌套规则——Class中嵌套class，从而减少重复的代码；

4.运算——CSS中用上数学；  
 5.颜色功能——可以编辑颜色；  
 6.名字空间(namespace)——分组样式，从而可以被调用；  
 7.作用域——局部修改样式；  
 8.JavaScript 赋值——在CSS中使用JavaScript表达式赋值。

详见：<https://www.cnblogs.com/hope666/p/6791790.html>

1. css模块化，按需加载，防止css阻塞渲染

Css模块化具体看团队

按需加载：详见Mixins

防止css阻塞渲染：浏览器大概渲染流程：浏览器自上而下解析目标HTMl文件=>HTML解析器将HTML结构转换为基础的DOM tree=>CSS解析器将CSS解析为CSSOM=>CSSOM和DOM合并构成渲染树=>计算渲染树中每个节点的信息绘制页面；因此当CSSOM没有构建完成时，页面不会渲染，即css阻塞渲染。

解决方法：精简CSS，尽早提供，利用媒体查询和类型

1. css动画

## @keyframes和animation

渐变、移动、旋转、缩放

详见：<https://www.runoob.com/css3/css3-tutorial.html>

1. css浏览器兼容写法及不同API在不同浏览器下的兼容情况

加前缀，剩下的以后总结补充

1. 响应式布局方案
2. 媒体查询

使用@media媒体查询可以针对不同的媒体类型和不同的屏幕大小定义不同的样式，从而达到自适应的效果

缺点：多套样式代码较繁琐

1. 百分比

通过百分比单位使浏览器中组件的宽和高随着浏览器的变化而变化

1. Rem

rem单位是相对于根元素html的font-size来决定大小的，根据视图容器的大小动态的改变font-size。

1. Vw/vh

100vw为视窗的宽度，100vh为视窗的高度

(opera浏览器不兼容)

剩下的待补充

13.CSS常用效果

详见：<https://github.com/Bpimp/draft.git>

1. JavaScript
2. 数据类型

基本数据类型：null、string、number、undefined、boolen。

引用数据类型：对象数据类型（数组、正则、对象）、函数数据类型

注：ES6新增symbol

Symbol：表示独一无二的值，在为对象添加新的属性时，可以防止属性名的冲突。

Symbol实现：https://segmentfault.com/a/1190000015262174#articleHeader4

1. 数据结构

常用数据结构：Array、Object

不常用数据结构：Set、WeakSet、Map、WeakMap

结构型数据：JSON

特殊结构：Function

详见：<https://developer.mozilla.org/zh-CN/>

1. JavaScript中的变量在内存中的具体存储形式

基本数据类型(值类型)保存在栈内存，引用数据类型(引用类型)实际上是一个指针，这个指针保存在栈中，但是这个指针指向的对象则保存在堆内存中；访问时，基本数据类型访问的是它们实际保存的值。应用数据类型则是按引用访问，先从栈中读取内存地址，然后找到保存在堆内存中的值；复制时，基本数据类型会在栈中创建一个新值，然后把值复制到为新变量分配的空间中，引用类型的变量则是复制存储在栈中的指针，将指针复制到栈中为新变量分配的空间中，而这个指针副本和原指针指向的是同一个堆内存中的对象，两个变量实际上将引用同一个对象，因此改变其中一个将影响到另外一个；检测基本数据类型为typeof，引用数据类型为instanceof。

1. 基本数据类型对应的内置对象

字符串：String();

数组：Array();

数字：Number();

布尔值：Boolean();

...

1. null和undefined区别

Null：Null类型，代表”空值”，代表一个空对象指针，是一个特殊的对象值。

Undefined：当一个变量被声明但未初始化时，就是undefined，JavaScript独有。

1. JavaScript数据类型的判断方法
2. typeof()无法区分引用数据类型
3. Instanceof()无法区分原始数据类型
4. Constructor
5. Object.prototype.toString.call()
6. Jquery.type()

详见:https://www.cnblogs.com/dushao/p/5999563.html

1. JavaScript隐式类型转换

详见：<https://www.cnblogs.com/chenmeng0818/p/5954215.html>

1. JavaScript数字相关

Js的数字类型采用IEEE 754标准的64-bits的双精度存储，当数字到了计算机的最底层都会转换成二进制，但二进制无法准确表示这种包含小数点的数据，其本质是将浮点数转换成了用二进制表示的最接近的近似值，并且js也没有相应的封装类来处理浮点数运算。

精度丢失解决方法：整数不超过Math.pow(2,53)(最大数字)。注：最大安全数字为Math.pow(2,53)-1，个人理解。

小数先乘以倍数转换为整数，运算后在除以相同的倍数

1. **原型和原型链**

Js原型模式：js有一个根对象(Object.prototype)，通过拷贝这个根对象创建新的对象。详见：JavaScript设计模式与开发实践

Js原型规则：

1. 所有引用类型(数组、对象、函数)都具有对象特性，即可自由扩展属性
2. 所有的引用类型(数组、对象、函数)，都有一个\_proto\_属性(隐式原型)，属性值是一个普通的对象
3. 所有的函数都具有一个prototype(显示原型)，属性值也是一个普通对象
4. 所有的引用类型，其隐式原型指向其构造函数的显示原型：(obj.\_proto\_===Object.prototype)
5. 当试图得到一个对象的某个属性时，如果这个对象本身没有这个属性，那么会去它的\_proto\_(即它的构造函数的prototype)中去寻找
6. instanceof的底层实现原理

语法：Object instanceof constructor

Instanceof运行的过程中遍历左边对象的原型链，如果找到右边对象的prototype返回true，否则返回false。

手写实现instanceof：

function instanceOf(*left*,*right*) {  
 let rightproto=*right*.prototype;  
 *left*=*left*.\_\_proto\_\_;  
 while(true){  
 if(*left*===null){  
 return false;  
 }else if(*left*===rightproto){  
 return true;  
 }  
 *left*=*left*.\_\_proto\_\_;  
 }  
}

1. 继承的实现方法

详见：http://www.ruanyifeng.com/blog/2011/06/designing\_ideas\_of\_inheritance\_mechanism\_in\_javascript.html

1. 原型链继承：将父类的实例作为子类的原型

function Basepro(){  
 this.property=true;  
}  
Basepro.prototype.getValue=function(){  
 return this.property;  
};  
function Newpro(){  
}  
Newpro.prototype=new Basepro();  
let *instance*=new Newpro();  
*console*.log(*instance*.getValue());

优点：简单，易于实现；父类新增原型方法或属性，子类都能访问到

缺点：来自原型对象的所有属性被所有实例共享，导致对一个实例的修改会影响到其它实例；创造子类实例时，无法向父类构造函数传参；等等

1. 构造继承：借用call或apply函数调用父类的构造函数

function Basepro(){  
 this.name ="Tom";  
}  
function Newpro(){  
 Basepro.call(this)  
}  
let *rat* = new Newpro();  
*rat*.name = "Jerry";  
let *cat* = new Newpro();  
*console*.log(*cat*.name);

优点：解决了子类实例共享父类引用类型属性的问题；创建子类实例时可以向父类传递参数；可以实现多继承

缺点：实例不是父类的实例，只是子类的实例，只能继承父类的实例属性和方法，不能继承原型属性和方法；方法都在构造函数中定义无法实现函数复用。

1. 组合继承：将原型链继承和构造继承相结合

function Basepro(*name*){  
 this.name=*name*;  
 this.type ="cat";  
}  
Basepro.prototype.getName=function () {  
 *console*.log(1,this.name);  
};  
function Newpro(*name*){  
 Basepro.call(this,*name*)  
}  
Newpro.prototype=new Basepro();  
Newpro.prototype.constructor=Newpro;  
  
let *rat* = new Newpro("Jerry");  
*rat*.type="rat";  
let *cat* = new Newpro("Tom");  
*console*.log(*rat*.type,*cat*.type);  
*rat*.getName();  
*cat*.getName();

优点：结合了一和二两种方法的优点，解决了不足之处

缺点：调用了两次父类的构造函数

1. 拷贝继承：将父类的所有属性拷贝进子类中