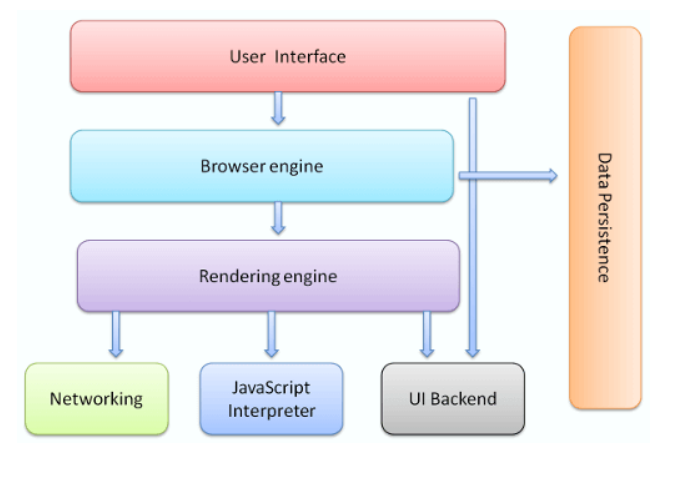
# 前端知识文档

## 1.浏览器介绍

浏览器（browser application）是专门用来访问和浏览万维网页面的客户端软件，也是现代计算机系统中应用最为广泛的软件之一，其重要性不言而喻。前端工程师作为负责程序页面显示的工程师，需要直接与浏览器打交道。本文将详细介绍浏览器的工作原理。

浏览器的组成如下图所示



主要组件包括：

1. 用户界面 － 包括地址栏、后退/前进按钮、书签目录等，也就是所看到的除了用来显示所请求页面的主窗口之外的其他部分

2. 浏览器引擎 － 用来查询及操作渲染引擎的接口

3. 渲染引擎 － 用来显示请求的内容，例如，如果请求内容为html，它负责解析html及css，并将解析后的结果显示出来。

4. 网络 － 用来完成网络调用，例如http请求，它具有平台无关的接口，可以在不同平台上工作。

5. UI后端 － 用来绘制类似组合选择框及对话框等基本组件，具有不特定于某个平台的通用接口，底层使用操作系统的用户接口。

6. JS解释器 － 用来解释执行JS代码。

7. 数据存储 － 属于持久层，浏览器需要在硬盘中保存类似cookie的各种数据，HTML5定义了web database技术，这是一种轻量级完整的客户端存储技术

### 1.1内核

浏览器内核分成两部分：渲染引擎和js引擎，由于js引擎越来越独立，内核就倾向于只指渲染引擎，负责请求网络页面资源加以解析排版并呈现给用户

默认情况下，渲染引擎可以显示html、xml文档及图片，它也可以借助插件显示其他类型数据，例如使用PDF阅读器插件，可以显示PDF格式

【渲染引擎】

firefox使用gecko引擎

IE使用Trident引擎，2015年微软推出自己新的浏览器，原名叫斯巴达，后改名edge，使用edge引擎

opera最早使用Presto引擎，后来弃用

chrome\safari\opera使用webkit引擎，13年chrome和opera开始使用Blink引擎

UC使用U3引擎

QQ浏览器和微信内核使用X5引擎，16年开始使用Blink引擎

【js引擎】

老版本IE使用Jscript引擎，IE9之后使用Chakra引擎，edge浏览器仍然使用Chakra引擎

firefox使用monkey系列引擎

safari使用的SquirrelFish系列引擎

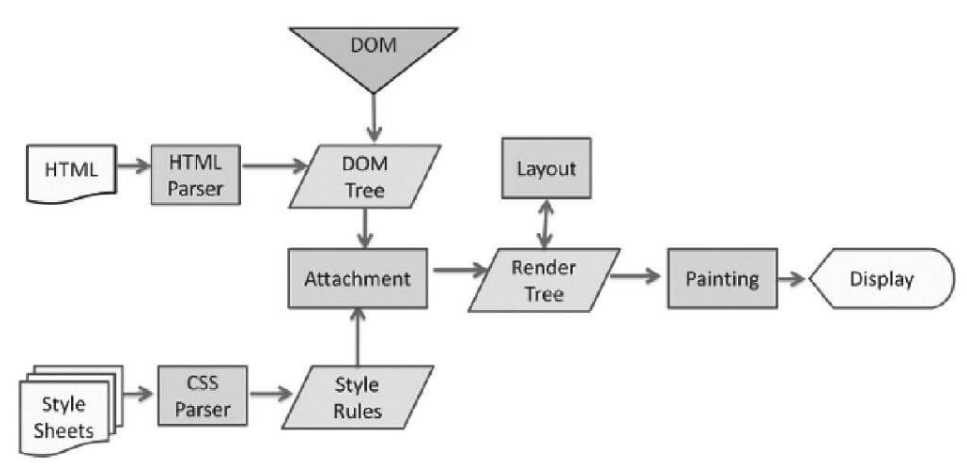
Opera使用Carakan引擎

chrome使用V8引擎。nodeJs其实就是封装了V8引擎

### 1.2渲染流程

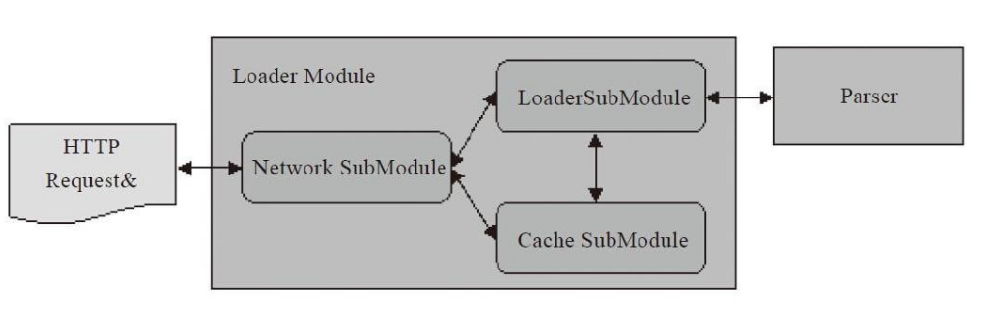
从资源的下载到最终的页面展现，渲染流程可简单地理解成一个线性串联的变换过程的组合，原始输入为URL地址，最终输出为页面Bitmap，中间依次经过了Loader、Parser、Layout和Paint模块

渲染引擎的核心流程如下所示



【Loader】

Loader模块负责处理所有的HTTP请求以及网络资源的缓存，相当于是从URL输入到Page Resource输出的变换过程。HTML页面中通常有外链的JS/CSS/Image资源，为了不阻塞后续解析过程，一般会有两个IO管道同时存在，一个负责主页面下载，一个负责各种外链资源的下载



注意：虽然大部分情况下不同资源可以并发下载异步解析（如图片资源可以在主页面解析显示完成后再被显示），但JS脚本可能会要求改变页面，因此有时保持执行顺序和下载管道后续处理的阻塞是不可避免的

Parser】

1、解析HTML

Parser模块主要负责解析HTML页面，完成从HTML文本到HTML语法树再到文档对象树（Document Object Model Tree，DOM Tree）的映射过程

HTML语法树生成是一个典型的语法解析过程，可以分成两个子过程：词法解析和语法解析

词法解析按照词法规则（如正则表达式）将HTML文本分割成大量的标记（token），并去除其中无关的字符如空格。语法解析按照语法规则（如上下文无关文法）匹配Token序列生成语法树，通常有自上而下和自下而上两种匹配方式

浏览器内核中对HTML页面真正的内部表示并不是语法树，而是W3C组织规范的文档对象模型(Document Object Model，DOM)。DOM也是树形结构，以Document对象为根。DOM节点基本和HTML语法树节点一一对应，因此在语法解析过程中，通常直接生成最终的DOM树

2、解析CSS

页面中所有的CSS由样式表CSSStyleSheet集合构成，而CSSStyleSheet是一系列CSSRule的集合，每一条CSSRule则由选择器CSSStyleSelector部分和声明CSSStyleDeclaration部分构成，而CSSStyleDeclaration是CSS属性和值的Key-Value集合

CSS解析完毕后会进行CSSRule的匹配过程，即寻找满足每条CSS规则Selector部分的HTML元素，然后将其Declaration部分应用于该元素。实际的规则匹配过程会考虑到默认和继承的CSS属性、匹配的效率及规则的优先级等因素

3、解析Javascript

JavaScript一般由单独的脚本引擎解析执行，它的作用通常是动态地改变DOM树（比如为DOM节点添加事件响应处理函数），即根据时间（timer）或事件（event）映射一棵DOM树到另一棵DOM树。

简单来说，经过了Parser模块的处理，内核把页面文本转换成了一棵节点带CSS Style、会响应自定义事件的Styled DOM树

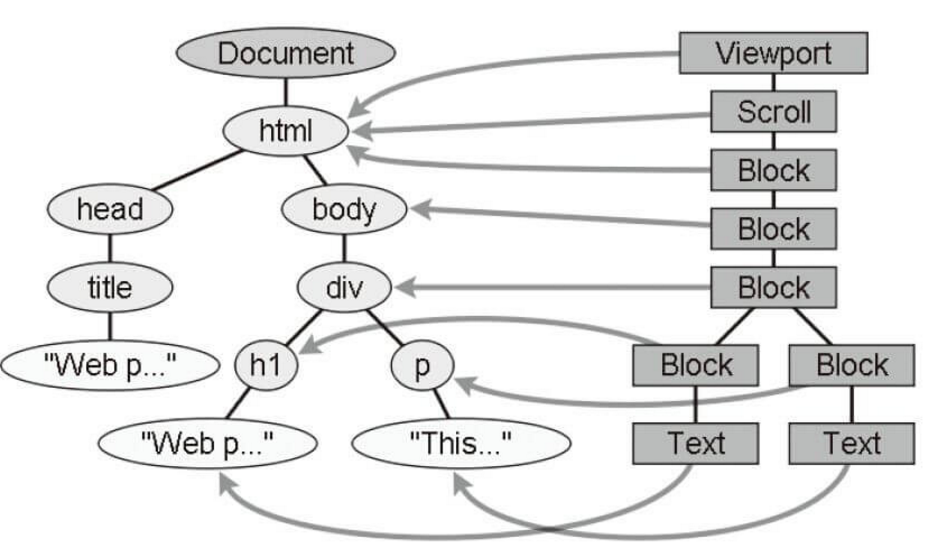
layout】

Layout过程就是排版，它包含两大过程

1、创建渲染树

布局树（或者叫做渲染树、Render Tree）和DOM树大体能一一对应，两者在内核中同时存在但作用不同。DOM树是HTML文档的对象表示，同时也作为JavaScript操纵HTML的对象接口。Render树是DOM树的排版表示，用以计算可视DOM节点的布局信息（如宽、高、坐标）和后续阶段的绘制显示

注意：并非所有DOM节点都可视，也就是并非所有DOM树节点都会对应生成一个Render树节点。例如head标签（HTMLHeadElement节点）不表示任何排版区域，因而没有对应的Render节点。同时，DOM树可视节点的CSS Style就是其对应Render树节点的Style



2、计算布局

布局就是安排和计算页面中每个元素大小位置等几何信息的过程。HTML采用流式布局模型，基本的原则是页面元素在顺序遍历过程中依次按从左至右、从上至下的排列方式确定各自的位置区域

一个HTML元素对应一个以CSS盒子模型描述的方块区域，HTML元素分成两个基本类型，Inline和Block。Inline元素不会换行，按从左到右来布局。Block元素的出现意味着需要从上至下换到下一行来布局。除了这种基本的顺序按照元素的Inline和Block来进行流式布局之外，还有特殊指定的一些布局方式，如Absolute/Fixed/Relative三种定位布局以及Float浮动布局

简单情况下，布局可以顺序遍历一次Render树完成，但也有需要迭代的情况。当祖先元素的大小位置依赖于后代元素或者互相依赖时，一次遍历就无法完成布局，如Table元素的宽高未明确指定而其下某一子元素Tr指定其高度为父Table高度的30%的情况

经过了Layout阶段的处理，把带Style的DOM树变换成包含布局信息和绘制信息的Render树，接下来的显示工作就交由Paint模块进行操作了

【Paint】

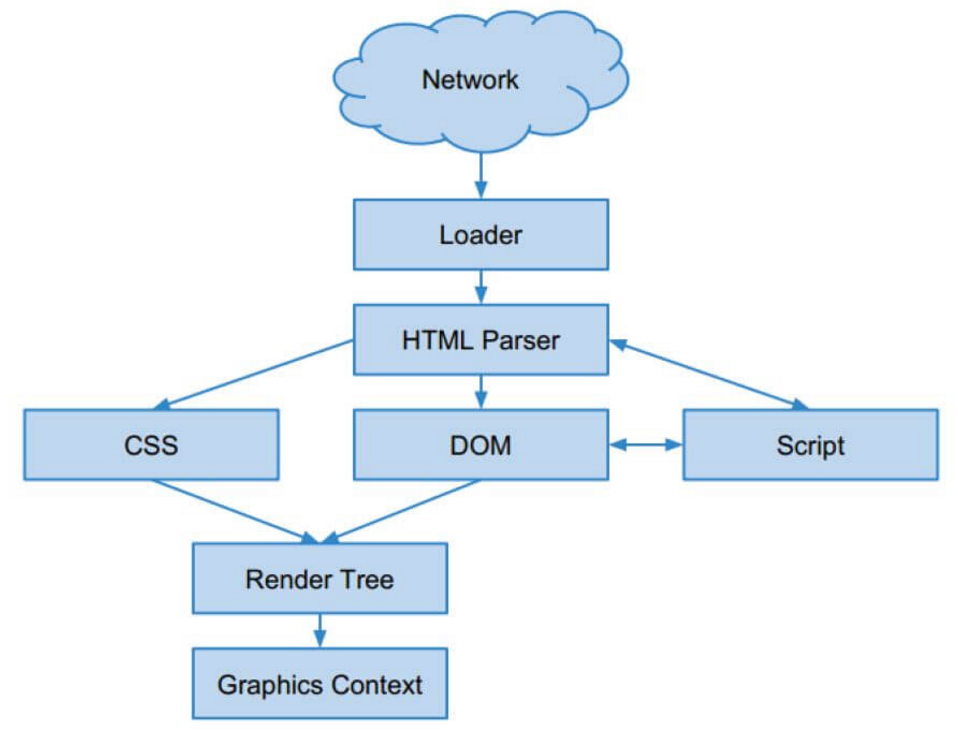
Paint模块负责将Render树映射成可视的图形，它会遍历Render树调用每个Render节点的绘制方法将其内容显示在一块画布或者位图上，并最终呈现在浏览器应用窗口中成为用户看到的实际页面。每个节点对应的大小位置等信息都已经由Layout阶段计算好了，节点的内容取决于对应的HTML元素，或是文本，或是图片，或是UI控件

通常情况下，布局和绘制是相当耗时的操作。如果DOM树每次略有改动都要重新布局和绘制一次，效率会相当低下。因此，一般浏览内核都会实现一种增量布局和增量绘制的方式。当一个DOM树节点（或者它的子节点）内容或者样式发生变化时，内核会确定其影响范围，在布局阶段会标记出受该节点布局影响的其他节点（比如可能是子节点），在绘制阶段则会标记出一个Dirty区域并通知系统重绘

按照HTML相关规范，页面元素的CSS属性也规定了其绘制顺序，如根据不同Layer必须按顺序绘制，否则覆盖叠加效果会出现错误，如元素的边框轮廓和内容背景的绘制次序也有规定

使用浏览器上网时，首先会在地址栏输入一个网址，浏览器会依据网址向服务器发送资源请求，服务器解析请求，并将相关数据资源传送回给浏览器，这些数据资源包括Page的描述文档、图片、JavaScript脚本、CSS等。此后，浏览器引擎会对数据进行解码、解析、排版、绘制等操作，最终呈现出完整的页面。Loader是浏览器的排头兵，负责资源加载的工作

Loader在浏览器中承上启下，一方面它作为网络模块的客户，通过网络模块来加载资源；另一方面它为Parser模块加载页面的内容，控制着浏览器后续的解析以及绘制过程



Loader有两条资源加载路径：主资源加载路径和派生资源加载路径。这两类资源的加载过程颇有不同，比如对资源加载失败的处理，主资源下载失败会有报错提示，而派生资源如图片下载失败，往往只显示一个占位

在地址栏输入新地址或者在已经打开的页面中点击链接，都会触发主资源的加载流程，随着主资源在HTTP协议的传输下分段到达，浏览器的Parser模块解析主资源的内容，生成派生资源对应的DOM结构，然后根据需求触发派生资源的加载流程。主资源的加载是立刻发起的，而派生资源则可能会为了优化网络，在队列中等待

主资源和派生资源的加载还有一个区别，在Android 4.2版本中主资源是没有缓存的，而派生资源是有缓存机制的。这里的缓存指的是Memory Cache，用于保存原始数据（比如CSS、JS等），以及解码过的数据，通过Memory Cache可以节省网络请求和图片解码的时间

浏览器在加载主资源后，主资源会被解码，然后进行解析，生成DOM（文档对象模型）树。在解析过程中，如果遇到<img的起始标签，会创建相应的image元素HTMLImageElement，接着依据img标签的内容设置HTMLImageElement的属性。在设置src属性时，会触发图片资源加载，发起加载资源请求

### 1.3缓存

缓存在浏览器中也得到了广泛的应用，对提高用户体验起到了重要作用。在浏览器中，主要存在三种类型的缓存：Page Cache、Memory Cache、Disk Cache。这三类Cache的容量都是可以配置的，比如限制Memory Cache最大不超过30MB，Page Cache缓存的页面数量不超过5个等

Page Cache：是将浏览的页面状态临时保存在缓存中，以加速页面返回等操作

Memory Cache：浏览器内部的缓存机制，对于相同url的资源直接从缓存中获取，不需重新下载

Disk Cache：资源加载缓存和服务器进行交互，服务器端可以通过HTTP头信息设置网页要不要缓存。

网页解析

可以将浏览器整体看作一个网页处理模块，这个模块的输入是网络上接收到的字节流形式的网页内容。输出是三棵树型逻辑结构：DOM树、Render树及RenderLayer树

浏览器的解析过程就是将字节流形式的网页内容构建成DOM树、Render树及RenderLayer树的过程

浏览器的解析对象是网页内容，网页内容包括以下三个部分：

1、HTML文档：超文本标记语言，制作Web页面的标准语言

2、CSS样式表（Cascading Style Sheet）：级联样式表，用来控制网页样式，并允许样式信息与网页内容相分离的一种标记性语言

3、JavaScript脚本：JavaScript是一种无类型的解释型脚本语言。常用于为网页添加动态功能

HTML文档决定了DOM树及Render树的结构。CSS样式表决定了Render树上节点的排版布局方式。JavaScript代码可以操作DOM树，改变DOM树的结构，也可以用来给页面添加更丰富的动态功能

HTML文档被解析生成DOM树，由DOM节点创建Render树节点时，会触发CSS匹配过程，CSS匹配的结果是RenderStyle实例，这个实例由Render节点持有，保存了Render节点的排版布局信息。CSS的解析过程即是CSS语法在浏览器的内部表示过程，解析的结果是得到一系列的CSS规则。CSS的匹配过程主要依据CSS选择器的不同优先级进行，高优先级选择器优先适用。根据网页上定义的JavaScript脚本的不同属性，JavaScript脚本的下载和执行时机会有所不同。JavaScript脚本的执行是由渲染引擎转交给JS引擎执行的。下面分别看一下HTML、CSS、JavaScript的具体解析和执行

## 2前端发展简史

### 2.1起源

#### 2.1.1 1990 HTML

1990 年，Tim 以超文本语言 HTML 为基础在 NeXT 电脑上发明了最原始的 Web 浏览器。

1991 年，Tim 作为布道者在 Internet 上广泛推广 Web 的理念，与此同时，美国国家超算应用中心（National Center for Supercomputer Applications）对此表现出了浓厚的兴趣，并开发了名为 Mosaic 的浏览器，于 1993 年 4 月进行了发布。

#### 2.1.2 1994 年 5 月，第一届万维网大会在日内瓦召开。

1994.7 HTML 2.0 规范发布

1994 年 9 月，因特网工程任务组（Internet Engineering Task Force）设立了 HTML 工作组。

1994 年 11 月，Mosaic 浏览器的开发人员创建了网景公司（Netscape Communications Corp.），并发布了 Mosaic Netscape 1.0 beta 浏览器，后改名为 Navigator。

1994 万维网联盟（World Wide Web Consortium）成立，简称 W3C

1994 年底，由 Tim 牵头的万维网联盟（World Wide Web Consortium）成立，这标志着万维网的正式诞生。

此时的网页以 HTML 为主，是纯静态的网页，网页是“只读”的，信息流只能通过服务器到客户端单向流通，由此世界进入了 Web 1.0 时代。

#### 2.1.3 1995 网景推出 JavaScript

1995 年，网景工程师 Brendan Eich 花了10天时间设计了 JavaScript 语言。起初这种脚本语言叫做 Mocha，后改名 LiveScript，后来为了借助 Java 语言创造良好的营销效果最终改名为 JavaScript。网景公司把这种脚本语言嵌入到了 Navigator 2.0 之中，使其能在浏览器中运行。

与此相对的是，1996 年，微软发布了 VBScript 和 JScript。JScript 是对 JavaScript 进行逆向工程的实现，并内置于 Internet Explorer 3 中。但是 JavaScript 与 JScript 两种语言的实现存在差别，这导致了程序员开发的网页不能同时兼容 Navigator 和 Internet Explorer 浏览器。 Internet Explorer 开始抢夺 Netscape 的市场份额，这导致了第一次浏览器战争。

### 2.2第一次浏览器战争

1996 年 11 月，为了确保 JavaScript 的市场领导地位，网景将 JavaScript 提交到欧洲计算机制造商协会（European Computer Manufacturers Association）以便将其进行国际标准化。

1996.12 W3C 推出了 CSS 1.0 规范

1997.1 HTML3.2 作为 W3C 推荐标准发布

1997.6 ECMA 以 JavaScript 语言为基础制定了 ECMAScript 1.0 标准规范

1997 年 6 月，ECMA 以 JavaScript 语言为基础制定了 ECMAScript 标准规范 ECMA-262。JavaScript 是 ECMAScript 规范最著名的实现之一，除此之外，ActionScript 和 JScript 也都是 ECMAScript 规范的实现语言。自此，浏览器厂商都开始逐步实现 ECMAScript 规范。

1997.12 HTML 4.0 规范发布

1998 W3C 推出了 CSS 2.0 规范

1998.6 ECMAScript 2 规范发布

1998 年 6 月，ECMAScript 2 规范发布，并通过 ISO 生成了正式的国际标准 ISO/IEC 16262 。

1999.12 ECMAScript 3 规范发布

1999 年 12 月，ECMAScript 3 规范发布，在此后的十年间，ECMAScript 规范基本没有发生变动。ECMAScript 3 成为当今主流浏览器最广泛使用和实现的语言规范基础。

第一次浏览器战争以 IE 浏览器完胜 Netscape 而结束，IE 开始统领浏览器市场，份额的最高峰达到 2002 年的 96%。随着第一轮大战的结束，浏览器的创新也随之减少。

### 2.3 XHTML

1999 W3C 发布 HTML 4.01 标准，同年微软推出用于异步数据传输的 ActiveX，随即各大浏览器厂商模仿实现了 XMLHttpRequest（AJAX 雏形）。

2000: W3C 采用了一个大胆的计划，把 XML 引入 HTML，XHTML1.0 作为 W3C 推荐标准发布

2001.5 W3C 推出了 CSS 3.0 规范草案

2002-2006 XHTML 2.0 最终放弃

2009 W3C 宣布 XHTML2.0 不再继续，宣告死亡

### 2.4动态页面的崛起

JavaScript 诞生之后，可以用来更改前端 DOM 的样式，实现一些类似于时钟之类的小功能。那时候的JavaScript 仅限于此，大部分的前端界面还很简单，显示的都是纯静态的文本和图片。这种静态页面不能读取后台数据库中的数据，为了使得 Web 更加充满活力，以 PHP、JSP、ASP.NET 为代表的动态页面技术相继诞生。

PHP（PHP：Hypertext Preprocessor）最初是由 Rasmus Lerdorf 在 1995 年开始开发的，现在PHP 的标准由 PHP Group 维护。PHP 是一种开源的通用计算机脚本语言，尤其适用于网络开发并可嵌入 HTML 中使用。PHP 的语法借鉴吸收 C 语言、Java 和 Perl 等流行计算机语言的特点，易于一般程序员学习。PHP 的主要目标是允许网络开发人员快速编写动态页面。

JSP（JavaServer Pages）是由 Sun 公司倡导和许多公司参与共同创建的一种使软件开发者可以响应客户端请求，从而动态生成 HTML、XML 或其他格式文档的 Web 网页的技术标准。JSP 技术是以 Java 语言为基础的。1999 年，JSP 1.2 规范随着 J2EE 1.2 发布。

ASP（Active Server Pages）1.0 在 1996 年随着 IIS 3.0 而发布。2002 年，ASP.NET 发布，用于替代 ASP。

随着这些动态服务器页面技术的出现，页面不再是静止的，页面可以获取服务器数据信息并不断更新。以 Google 为代表的搜索引擎以及各种论坛相继出现，使得 Web 充满了活力。

随着动态页面技术的不断发展，后台代码变得庞大臃肿，后端逻辑也越来越复杂，逐渐难以维护。此时，后端的各种 MVC 框架逐渐发展起来，以 JSP 为例，Struct、Spring 等框架层出不穷。

从 Web 诞生至 2005 年，一直处于后端重、前端轻的状态。

#### 2.4.1 AJAX 的流行

在 Web 最初发展的阶段，前端页面要想获取后台信息需要刷新整个页面，这是很糟糕的用户体验。

Google 分别在 2004 年和 2005 年先后发布了两款重量级的 Web 产品：Gmail 和 Google Map。这两款 Web 产品都大量使用了 AJAX 技术，不需要刷新页面就可以使得前端与服务器进行网络通信，这虽然在当今看来是理所应当的，但是在十几年前AJAX却是一项革命性的技术，颠覆了用户体验。

随着 AJAX 的流行，越来越多的网站使用 AJAX 动态获取数据，这使得动态网页内容变成可能，像 Facebook 这样的社交网络开始变得繁荣起来，前端一时间呈现出了欣欣向荣的局面。

AJAX 使得浏览器客户端可以更方便地向服务器发送数据信息，这促进了 Web 2.0 的发展。

### 2.5第二次浏览器大战

#### 2.5.1前端兼容性框架的出现

IE 在第一次浏览器大战中击败 Netscape 赢得胜利，垄断了浏览器市场。作为独裁者，IE 并不遵循 W3C 的标准，IE 成了事实标准。

Netscape 于 1998 年被 AOL 收购前创建了 Mozilla 社区，Firefox 于 2004 年 11 月首次发布，并且 9 个月内下载量超过 6000 万，获取了巨大的成功，IE 的主导地位首次受到了挑战， Firefox 被认为是 Netscape 的精神续作。

之后 Firefox 浏览器一路奋起直追，逐渐蚕食 IE 市场份额，这引发了第二次浏览器战争。在 2008 年底时，Firefox 的市场份额达到了 25% 以上，IE 则跌至 65% 以下。

第二次浏览器战争中，随着以 Firefox 和 Opera 为首的 W3C 阵营与 IE 对抗程度的加剧，浏览器碎片化问题越来越严重，不同的浏览器执行不同的标准，对于开发人员来说这是一个恶梦。

为了解决浏览器兼容性问题，Dojo、jQuery、YUI、ExtJS、MooTools 等前端 Framework 相继诞生。前端开发人员用这些 Framework 频繁发送 AJAX 请求到后台，在得到数据后，再用这些 Framework 更新 DOM 树。

其中，jQuery 独领风骚，几乎成了所有网站的标配。Dojo、YUI、ExtJS 等提供了很多组件，这使得开发复杂的企业级 Web 应用成为可能。

### 2.6 HTML5

1999年，W3C发布了 HTML 4.01 版本，在之后的几年，没有再发布更新的 Web 标准。随着Web的迅猛发展，旧的Web标准已不能满足 Web 应用的快速增长。

2004 年 6 月，Mozilla 基金会和 Opera 软件公司在万维网联盟（W3C）所主办的研讨会上提出了一份联合建议书，其中包括 Web Forms 2.0 的初步规范草案。建议举行一次投票，以表决 W3C 是否应该扩展 HTML 和 DOM，从而满足 Web 应用中的新需求。研讨会最后以 8 票赞成，14 票反对否决此建议，这引起一些人的不满，不久后，部分浏览器厂商宣布成立网页超文本技术工作小组（WHATWG），以继续推动该规范的开发工作，该组织再度提出 Web Applications 1.0 规范草案，后来这两种规范合并形成 HTML5。2007 年，获得 W3C 接纳，并成立了新的 HTML 工作团队。2008 年 1 月 22 日，第一份正式草案发布。

#### 2.6.1 2008.12 Chrome 发布，JavaScript 引擎 V8

HTML5 草案发布不久，Google 在 2008 年 12 月发布了 Chrome 浏览器，加入了第二次浏览器大战当中。Chrome 使用了 Safari 开源的 WebKit 作为布局引擎，并且研发了高效的 JavaScript 引擎 V8。

尽管 HTML5 在网络开发人员中非常出名了，但是它成为主流媒体的一个话题是在 2010 年的 4 月，当时苹果公司的 CEO 乔布斯发表一篇题为“对 Flash 的思考”的文章，指出随着 HTML5 的发展，观看视频或其它内容时，Adobe Flash 将不再是必须的。这引发了开发人员间的争论，包括 HTML5 虽然提供了加强的功能，但开发人员必须考虑到不同浏览器对标准不同部分的支持程度的不同，以及 HTML5 和 Flash 间的功能差异。

在第二次浏览器大战中，各个浏览器厂商都以提升 JavaScript 运行效率和支持 HTML5 各种新特性为主要目标，促进了浏览器的良性竞争。在这一场战争中，Chrome 攻城略地，抢夺 IE 市场份额。2013 年，Chrome 超过 IE，成为市场份额最高的浏览器。2016 年，Chrome 占据了浏览器市场的半壁江山。

自 2008 年以来，浏览器中不断支持的 HTML5 新特性让开发者激动不已：WebWorker 可以让 JavaScript 运行在多线程中，WebSocket 可以实现前端与后台的双工通信，WebGL 可以创建 Web3D 网页游戏…

#### 2.6.2桌面浏览器对 HTML5 支持程度（2009-2017）

2009.12 ECMAScript 5.0 规范发布

2011.6 ECMAScript 5.1 规范发布

2012.10 微软发布 TypeScript 公开版

TypeScript 是一种由微软开发的自由和开源的编程语言。它是 JavaScript 的一个超集，而且本质上向这个语言添加了可选的静态类型和基于类的面向对象编程。

TypeScript 扩展了 JavaScript 的语法，所以任何现有的 JavaScript 程序可以不加改变的在 TypeScript 下工作。TypeScript 是为大型应用之开发而设计，而编译时它产生 JavaScript 以确保兼容性。

2013.6.19 TypeScript 0.9 正式版

2014.10.28 W3C 正式发布 HTML 5.0 推荐标准

2014 年 10 月 28 日，W3C 正式发布 HTML 5.0 推荐标准

### 2.7 Node.js 的爆发

早在 1994 年，Netspace 就公布了其 Netspace Enterprise Server 中的一种服务器脚本实现，叫做 LiveWire，是最早的服务器端 JavaScript，甚至早于浏览器中的 JavaScript。对于这门图灵完备的语言，Netspace 很早就开始尝试将它用在后端。

微软在 1996 年发布的 IE 3.0 中内嵌了自己的 JScript语言，其兼容 JavaScript 语法。1997 年年初，微软在它的服务器 IIS 3.0 中也包含了 JScript，这就是我们在 ASP 中能使用的脚本语言。

1997 年，Netspace 为了用 Java 实现 JavaScript 而创建了 Rhino 项目，最终 Rhino 演变成一个基于 Java 实现的 JavaScript 引擎，由 Mozilla 维护并开源。Rhino 可以为 Java 应用程序提供脚本能力。2006 年 12 月，J2SE 6 将 Rhino 作为 Java 默认的脚本引擎。

SpiderMonkey 是 Mozilla 用 C/C++ 语言实现的一个 JavaScript 引擎，从 Firefox 3.5 开始作为 JavaScript 编译引擎，并被 CouchDB 等项目作为服务端脚本语言使用。

可以看到，JavaScript 最开始就能同时运行在前后端，但时在前后端的待遇却不尽相同。随着 Java、PHP、.Net 等服务器端技术的风靡，与前端浏览器中的 JavaScript 越来越流行相比，服务端 JavaScript 逐渐式微。

2008 年 Chrome 发布，其 JavaScript 引擎 V8 的高效执行引起了 Ryan Dahl 的注意。2009 年，Ryan 利用 Chrome 的 V8 引擎打造了基于事件循环的异步 I/O 框架 —— Node.js 诞生。

Node.js 具有以下特点：

基于事件循环的异步 I/O 框架，能够提高 I/O 吞吐量

单线程运行，能够避免了多线程变量同步的问题

使得 JavaScript 可以编写后台代码，前后端编程语言统一。

Node.js 的出现吸引了很多前端开发人员开始用 JavaScript 开发服务器代码，其异步编程风格也深受开发人员的喜爱。Node.js 的伟大不仅在于拓展了 JavaScript 在服务器端的无限可能，更重要的是它构建了一个庞大的生态系统。

2010 年 1 月，NPM 作为 Node.js 的包管理系统首次发布。开发人员可以按照 CommonJS 的规范编写 Node.js 模块，然后将其发布到 NPM 上面供其他开发人员使用。目前 NPM 具有 40 万左右的模块，是世界上最大的包模块管理系统。

#### 2.7.1 前端 MV\* 架构

随着 HTML5 的流行，前端不再是人们眼中的小玩意，以前在 C/S 中实现的桌面软件的功能逐步迁移到了前端，前端的代码逻辑逐渐变得复杂起来。

以前只用于后台的 MV\* 等架构在前端逐渐使用起来，以下列举了部分常用的 MV\* 框架。



随着这些 MV\* 框架的出现，网页逐渐由 Web Site 演变成了 Web App，最终导致了复杂的单页应用（ Single Page Application）的出现。

### 2.8移动 Web 和 Hybrid App

随着 iOS 和 Android 等智能手机的广泛使用，移动浏览器也逐步加强了对 HTML5 特性的支持力度。

移动浏览器对 HTML5 支持程度（2009-2017）

移动浏览器的发展，导致了流量入口逐渐从 PC 分流到移动平台，这是 Web 发展的新机遇。移动 Web 面临着更大的碎片化和兼容性问题，jQuery Mobile、Sencha Touch、Framework7、Ionic 等移动 Web 框架也随之出现。

相比于 Native App，移动 Web 开发成本低、跨平台、发布周期短的优势愈发明显，但是 Native App的性能和 UI 体验要远胜于移动 Web。移动 Web 与 Native App 孰优孰劣的争论愈演愈烈，在无数开发者的实践中，人们发现两者不是替代关系，而是应该将两者结合起来，取长补短，Hybrid 技术逐渐得到认同。

Hybrid 技术指的是利用 Web 开发技术，调用 Native 相关 API，实现移动与 Web 二者的有机结合，既能体现 Web 开发周期短的优势，又能为用户提供 Native 体验。

根据实现原理，Hybrid 技术可以分为两大类：

将 HTML 5 的代码放到 Native App 的 WebView 控件中运行，WebView 为 Web 提供宿主环境，JavaScript 代码通过 WebView 调用 Native API。典型代表有 PhoneGap(Cordova) 以及国内的 AppCan 等。

将 HTML 5 代码针对不同平台编译成不同的原生应用，实现了 Web 开发，Native 部署。这一类的典型代表有 Titanium 和 NativeScript。

Hybrid 一系列技术中很难找出一种方案适应所有应用场景，我们需要根据自身需求对不同技术进行筛选与整合。

ECMAScript 6

JavaScript 语言是 ECMAScript 标准的一种实现，截止 2017 年 2 月，ECMAScript 一共发布了 7 个版本。

1997 年 6 月， ECMAScript 1.0 标准发布。

1998 年 6 月，ECMAScript 2.0 发布。

1999 年 12 月，ECMAScript 3.0 发布。

2007 年 10 月，Mozilla 主张的 ECMAScript 4.0 版草案发布，对 3.0 版做了大幅升级，该草案遭到了以 Yahoo、Microsoft、Google 为首的大公司的强烈反对，JavaScript 语言的创造者 Brendan Eich 和 IE 架构师 Chris Wilson 甚至在博客上就ES4向后兼容性问题打起了口水仗，最后由于各方分歧太大，ECMA 开会决定废弃中止 ECMAScript 4.0 草案。经各方妥协，在保证向下兼容的情况下，将部分增强的功能放到 ECMAScript 3.1 标准中，将原有 ECMAScript 4.0 草案中激进的功能放到以后的标准中。不久，ECMAScript 3.1 就改名为 ECMAScript 5。

2009 年 12 月，本着’Don’t break the web’原则，ECMAScript 5 发布。新增了 strict 模式、属性 getter 和 setter 等。

2011 年 6 月，ECMAScript 5.1 发布。

2015 年 6 月，ECMAScript 6.0 发布。该版本增加了许多新的语法，包括支持 let、const、Arrow function、Class、Module、Promise、Iterator、Generator、Set、Map、async、Symbol、Proxy、Reflect、Decorator 等。TC39 委员会计划以后每年都发布一个新版本的 ECMAScript，所以 ECMAScript 6.0 改名为 ECMAScript 2015。

2016 年 6 月，在 ECMAScript 2015 的基础上进行了部分增强，发布了 ECMAScript 2016。

在 ECMAScript 的各个版本中，ECMAScript 6.0 无疑最受人瞩目的，它增加了许多新特性，极大拓展了 JavaScript 语法和能力，以至于许多浏览器都只能支持部分 ES6 中的新特性。随之，Babel 和 TypeScript 逐渐流行起来，编写 ES6 代码，然后用 Babel 或 TypeScript 将其编译为 ES5 等浏览器支持的 JavaScript。

ECMAScript 以后每年将会发布一个新版本，这无疑将持续促使浏览器厂商不断为 JavaScript 注入新的功能与特性，JavaScript走上了快速发展的正轨

## 3.HTML元素整理

### 3.1基本介绍

以段落元素为例学习元素组成：



元素的主要部分有：

1. **开始标签**（Opening tag）：包含元素的名称（本例为 p），被大于号、小于号所包围。表示元素从这里开始或者开始起作用 —— 在本例中即段落由此开始。
2. **结束标签**（Closing tag）：与开始标签相似，只是其在元素名之前包含了一个斜杠。这表示着元素的结尾 —— 在本例中即段落在此结束。初学者常常会犯忘记包含结束标签的错误，这可能会产生一些奇怪的结果。
3. **内容**（Content）：元素的内容，本例中就是所输入的文本本身。
4. **元素**（Element）：开始标签、结束标签与内容相结合，便是一个完整的元素。

元素也可以有属性（Attribute）

属性应该包含：

1. 在属性与元素名称（或上一个属性，如果有超过一个属性的话）之间的空格符。
2. 属性的名称，并接上一个等号。
3. 由引号所包围的属性值。

**嵌套元素**

也可以将一个元素置于其他元素之中 —— 称作**嵌套**。要表明猫咪非常暴躁，可以将 “爆” 用 [<strong>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/strong) 元素包围，爆字将突出显示

<p>我的猫咪脾气<strong>爆</strong>:)</p>

以上介绍了一些基本的 HTML 元素，但孤木不成林。现在来看看单个元素如何彼此协同构成一个完整的 HTML 页面。回顾 [文件处理](https://developer.mozilla.org/zh-CN/docs/Learn/Getting_started_with_the_web/Dealing_with_files) 小节中创建的 index.html 示例：

#### 3.1.1元信息标签

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>测试页面</title>

</head>

<body>

<img src="images/firefox-icon.png" alt="测试图片">

</body>

</html>

* <!DOCTYPE html> — 文档类型。混沌初分，HTML 尚在襁褓（大约是 1991/92 年），DOCTYPE 用来链接一些 HTML 编写守则，有点像自动校正等。然而现在已经没有人关心这些，只是因为历史原因必须将它们保留，但没有实际作用。现在你只需要知道这些就可以。
* <html></html> — [<html>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/html) 元素。这个元素包含了整个页面的内容，有时也被称作根元素。
* <head></head> — [<head>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/head) 元素。这个元素放置的内容不是展现给用户的，而是包含例如面向搜索引擎的搜索关键字（[keywords](https://developer.mozilla.org/en-US/docs/Glossary/keyword)）、页面描述、CSS 样式表和字符编码声明等。
* <meta charset="utf-8"> — 这个元素指定了当前文档使用 UTF-8 字符编码 ，UTF-8 包括绝大多数人类已知语言的字符。基本上 UTF-8 可以处理任何文本内容，还可以避免以后出现某些问题，我们没有任何理由再选用其他编码。元信息标签
* <title></title> — [<title>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/title) 元素。这个元素设置页面的标题，显示在浏览器标签页上，同时作为收藏网页的描述文字。元素 [<title>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/title) 是用来表示整个HTML文档大致内容的元数据（不是文档的内容）
* <body></body> — [<body>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/body) 元素。这个元素包含期望让用户在访问页面时看到的内容，可以是文本、图像、视频、游戏、可播放的音轨或其他内容。

### 3.2语义标签

图像[节](https://developer.mozilla.org/zh-CN/docs/Learn/Getting_started_with_the_web/HTML_basics#图像)

重温一下 [<img>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/img) 元素：

<img src="images/firefox-icon.png" alt="测试图片">

像之前所讲，该元素通过包含图像文件路径的地址属性 src，可在所在位置嵌入图像。

该元素还包括一个替换文字属性 alt，是图像的描述内容，用于当图像不能被用户看见时显示，不可见的原因可能是：

1. 用户有视觉障碍。视障用户可以使用屏幕阅读器来朗读 alt 属性的内容。
2. 有些错误使图像无法显示。可以试着故意将 src 属性里的路径改错。保存并刷新页面就可以在图像位置看到：

**标题（Heading）**

标题元素可用于指定内容的标题和子标题。就像一本书的书名、每章的大标题、小标题，等。HTML 文档也是一样。HTML 包括六个级别的标题， [<h1>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/h1)–[<h6>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/h6) ，一般最多用到 3-4 级标题。

<h1>主标题</h1>

<h2>顶层标题</h2>

<h3>子标题</h3>

<h4>次子标题</h4>

**段落（Paragraph）**

<p>这是一个段落</p>

**列表（List）**

Web 上的许多内容都是列表，HTML 有一些特别的列表元素。标记列表通常包括至少两个元素。最常用的列表类型为：

1. **无序列表（Unordered List）**中项目的顺序并不重要，就像购物列表。用一个 [<ul>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/ul) 元素包围。
2. **有序列表（Ordered List）**中项目的顺序很重要，就像烹调指南。用一个 [<ol>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/ol) 元素包围。

列表的每个项目用一个列表项目（List Item）元素 [<li>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/li) 包围

<p>Mozilla 是一个全球社区，这里聚集着来自五湖四海的</p>

<ul>

<li>技术人员</li>

<li>思考者</li>

<li>建造者</li>

</ul>

<p>我们致力于……</p>

#### 3.2.1链接

链接非常重要 — 它们赋予 Web 网络属性。要植入一个链接，我们需要使用一个简单的元素 — [<a>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/a) — a 是 "anchor" （锚）的缩写。要将一些文本添加到链接中，只需如下几步

<a href="https://www.mozilla.org/zh-CN/about/manifesto/">Mozilla 宣言</a>

如果网址开始部分省略了 https:// 或者 http://，可能会得到错误的结果。在完成一个链接后，可以试着点击它来确保指向正确。

href 这个名字可能开始看起来有点令人费解。如果你觉得不好记，那么记住它代表的是超文本引用（ **h**ypertext **ref**erence）。

网页的外观多种多样，但是除了全屏视频或游戏，或艺术作品页面，或只是结构不当的页面以外，都倾向于使用类似的标准组件：

**页眉**

通常横跨于整个页面顶部有一个大标题 和/或 一个标志。 这是网站的主要一般信息，通常存在于所有网页。

**导航栏**

指向网站各个主要区段的超链接。通常用菜单按钮、链接或标签页表示。类似于标题栏，导航栏通常应在所有网页之间保持一致，否则会让用户感到疑惑，甚至无所适从。许多 web 设计人员认为导航栏是标题栏的一部分，而不是独立的组件，但这并非决对；还有人认为，两者独立可以提供更好的 [无障碍访问特性](https://developer.mozilla.org/zh-CN/docs/learn/Accessibility)，因为屏幕阅读器可以更清晰地分辨二者。

**主内容**

中心的大部分区域是当前网页大多数的独有内容，例如视频、文章、地图、新闻等。这些内容是网站的一部分，且会因页面而异。

**侧边栏**

一些外围信息、链接、引用、广告等。通常与主内容相关（例如一个新闻页面上，侧边栏可能包含作者信息或相关文章链接），还可能存在其他的重复元素，如辅助导航系统。

**页脚**

横跨页面底部的狭长区域。和标题一样，页脚是放置公共信息（比如版权声明或联系方式）的，一般使用较小字体，且通常为次要内容。 还可以通过提供快速访问链接来进行 [SEO](https://developer.mozilla.org/en-US/docs/Glossary/SEO)。

一个“典型的网站”可能会这样布局：



为了实现语义化标记，HTML 提供了明确这些区段的专用标签，例如：

* [<header>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/header)：页眉。
* [<nav>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/nav)：导航栏。
* [<main>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/main)：主内容。主内容中还可以有各种子内容区段，可用[<article>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/article)、[<section>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/section) 和 [<div>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/div) 等元素表示。
* [<aside>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/aside)：侧边栏，经常嵌套在 [<main>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/main) 中。
* [<footer>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/footer)：页脚。

上图的示例可用下面的代码表示（完整代码请参见 [GitHub](https://github.com/roy-tian/mdn-examples/blob/master/html/site-structure/index.html)），请结合图片观察代码，并找出代码中可见的区段：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>二次元俱乐部</title>

<link href="https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300|Sonsie+One" rel="stylesheet">

<link href="https://fonts.googleapis.com/css?family=ZCOOL+KuaiLe" rel="stylesheet">

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

</head>

<body>

<header> <!-- 本站所有网页的统一主标题 -->

<h1>聆听电子天籁之音</h1>

</header>

<nav> <!-- 本站统一的导航栏 -->

<ul>

<li><a href="#">主页</a></li>

<!-- 共n个导航栏项目，省略…… -->

</ul>

<form> <!-- 搜索栏是站点内导航的一个非线性的方式。 -->

<input type="search" name="q" placeholder="要搜索的内容">

<input type="submit" value="搜索">

</form>

</nav>

<main> <!-- 网页主体内容 -->

<article>

<!-- 此处包含一个 article（一篇文章），内容略…… -->

</article>

<aside> <!-- 侧边栏在主内容右侧 -->

<h2>相关链接</h2>

<ul>

<li><a href="#">这是一个超链接</a></li>

<!-- 侧边栏有n个超链接，略略略…… -->

</ul>

</aside>

</main>

<footer> <!-- 本站所有网页的统一页脚 -->

<p>© 2050 某某保留所有权利</p>

</footer>

</body>

</html>

**无语义元素**

有时你会发现，对于一些要组织的项目或要包装的内容，现有的语义元素均不能很好对应。有时候你可能只想将一组元素作为一个单独的实体来修饰来响应单一的用 [CSS](https://developer.mozilla.org/en-US/docs/Glossary/CSS) 或 [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript)。为了应对这种情况，HTML提供了 [<div>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/div) 和 [<span>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/span) 元素。应配合使用 [class](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Global_attributes#attr-class) 属性提供一些标签，使这些元素能易于查询。

[<span>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/span) 是一个内联的（inline）无语义元素，最好只用于无法找到更好的语义元素来包含内容时，或者不想增加特定的含义时。例如：

<p>国王喝得酩酊大醉，在凌晨 1 点时才回到自己的房间，踉跄地走过门口。<span class="editor-note">[编辑批注：此刻舞台灯光应变暗]</span>.</p>

这里，“编辑批注”仅仅是对舞台剧导演提供额外指引；没有具体语义。对于观众，CSS 应将批注内容与主内容稍微隔开一些。

[<div>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/div) 是一个块级无语义元素，应仅用于找不到更好的块级元素时，或者不想增加特定的意义时。例如，一个电子商务网站页面上有一个一直显示的购物车组件。

<div class="shopping-cart">

<h2>购物车</h2>

<ul>

<li>

<p><a href=""><strong>银耳环</strong></a>：$99.95.</p>

<img src="../products/3333-0985/" alt="Silver earrings">

</li>

<li>

...

</li>

</ul>

<p>售价：$237.89</p>

</div>

这里不应使用 <aside>，因为它和主内容并没有必要的联系（你想在任何地方都能看到它）。甚至不能用 <section> ，因为它也不是页面上主内容的一部分。所以在这种情况下就应使用 <div>，为满足无障碍使用特征，还应为购物车的标题设置一个可读标签。

<div> 非常便利但容易被滥用。由于它们没有语义值，会使 HTML 代码变得混乱。要小心使用，只有在没有更好的语义方案时才选择它，而且要尽可能少用， 否则文档的升级和维护工作会非常困难。

**换行与水平分割线**

有时会用到 [<br>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/br) 和 [<hr>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/hr) 两个元素，需要介绍一下。

<br> 可在段落中进行换行；<br> 是唯一能够生成多个短行结构（例如邮寄地址或诗歌）的元素。比如：

<p>从前有个人叫小高<br>

他说写 HTML 感觉最好<br>

但他写的代码结构语义一团糟<br>

他写的标签谁也懂不了。</p>

没有 <br> 元素，这段会直接显示在长长的一行中（如前文所讲，[HTML会忽略大部分空格](https://developer.mozilla.org/zh-CN/docs/learn/HTML/Introduction_to_HTML/Getting_started#HTML中的空白)）；使用 <br> 元素，才使得诗看上去像诗：

从前有个人叫小高  
他说写 HTML 感觉最好  
但他写的代码结构语义一团糟  
他写的标签谁也懂不了。

<hr> 元素在文档中生成一条水平分割线，表示文本中主题的变化（例如话题或场景的改变）。一般就是一条水平的直线。例如：

<p>原来这唐僧是个慈悯的圣僧。他见行者哀告，却也回心转意道：“既如此说，且饶你这一次。再休无礼。如若仍前作恶，这咒语颠倒就念二十遍！”行者道：“三十遍也由你，只是我不打人了。”却才伏侍唐僧上马，又将摘来桃子奉上。唐僧在马上也吃了几个，权且充饥。</p>

<hr>

<p>却说那妖精，脱命升空。原来行者那一棒不曾打杀妖精，妖精出神去了。他在那云端里，咬牙切齿，暗恨行者道：“几年只闻得讲他手段，今日果然话不虚传。那唐僧已此不认得我，将要吃饭。若低头闻一闻儿，我就一把捞住，却不是我的人了。不期被他走来，弄破我这勾当，又几乎被他打了一棒。若饶了这个和尚，诚然是劳而无功也。我还下去戏他一戏。”</p>

将渲染成：

原来这唐僧是个慈悯的圣僧。他见行者哀告，却也回心转意道：“既如此说，且饶你这一次。再休无礼。如若仍前作恶，这咒语颠倒就念二十遍！”行者道：“三十遍也由你，只是我不打人了。”却才伏侍唐僧上马，又将摘来桃子奉上。唐僧在马上也吃了几个，权且充饥。



却说那妖精，脱命升空。原来行者那一棒不曾打杀妖精，妖精出神去了。他在那云端里，咬牙切齿，暗恨行者道：“几年只闻得讲他手段，今日果然话不虚传。那唐僧已此不认得我，将要吃饭。若低头闻一闻儿，我就一把捞住，却不是我的人了。不期被他走来，弄破我这勾当，又几乎被他打了一棒。若饶了这个和尚，诚然是劳而无功也。我还下去戏他一戏。”



### 3.3 web 中的音频和视频

web 开发者们一直以来想在 Web 中使用音频和视频，自21世纪初以来，我们的带宽开始能够支持任意类型的视频（视频文件比文本和图片要大的多）。在早些时候，传统的 web 技术（如 HTML ）不能够在 Web 中嵌入音频和视频，所以一些像 [Flash](https://en.wikipedia.org/wiki/Adobe_Flash) (后来有 [Silverlight](https://en.wikipedia.org/wiki/Microsoft_Silverlight) ) 的专利技术在处理这些内容上变得很受欢迎。这些技术能够正常的工作，但是却有着一系列的问题，包括无法很好的支持 HTML/CSS 特性、安全问题，以及可行性问题。

传统的解决方案能够解决许多这样的问题，前提是它能够正确的工作。幸运的是，几年之后 [HTML5](https://developer.mozilla.org/en-US/docs/Glossary/HTML5) 标准提出，其中有许多的新特性，包括 [<video>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/video) 和 [<audio>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/audio) 标签，以及一些 [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) 和 [APIs](https://developer.mozilla.org/en-US/docs/Glossary/API) 用于对其进行控制。在这里，我们不讨论有关 JavaScript 的问题，仅仅讲解有关 HTML 的基础。

**<video> 标签**

[<video>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/video) 允许你简单的嵌入一段视频。一个简单的例子如下：

<video src="rabbit320.webm" controls>

<p>你的浏览器不支持 HTML5 视频。可点击<a href="rabbit320.mp4">此链接</a>观看</p>

</video>

[**controls**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/video#attr-controls)

用户必须能够控制视频和音频的回放功能。你可以使用浏览器提供的控制接口，同时你也可以使用合适的 [JavaScript API](https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement)构建控制接口。至少，这些媒体应该包括开始和停止，以及调整音量的功能

**<video> 标签内的段落**

这个叫做**后备内容** — 当浏览器不支持 <video> 标签的时候，它将会显示出来，它使我们能够对旧的浏览器做一些兼容处理。你可以添加任何后备内容，在这个例子中我们提供了一个指向这个视频文件的链接，从而使用户可以至少访问到这个文件，而不会局限于浏览器的支持。

**<audio> 标签**

[<audio>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/audio) 标签与 [<video>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/video) 标签的使用方式几乎完全相同，有一些细微的差别比如下面的边框不同，一个典型的例子如下：

<audio controls>

<source src="viper.mp3" type="audio/mp3">

<source src="viper.ogg" type="audio/ogg">

<p>你的浏览器不支持 HTML5 音频，可点击<a href="viper.mp3">此链接</a>收听。</p>

</audio>

### 3.4 HTML表单

[**<form>**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/form)**元素**

form action="/my-handling-form-page" method="post">

</form>

这个元素正式定义了一个表单。就像[<div>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/div)元素或[<p>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/p)元素，它是一个容器元素，但它也支持一些特定的属性来配置表单的行为方式。它的所有属性都是可选的，但实践中最好至少要设置action属性和method属性。

* action 属性定义了在提交表单时,应该把所收集的数据送给谁(/那个模块)(URL)去处理。.
* method 属性定义了发送数据的HTTP方法(它可以是“get”或“post”).

[**<label>**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/label)**,**[**<input>**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/input)**和**[**<textarea>**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/textarea)**元素**

我们的联系人表单非常简单，包含三个文本字段，每个字段都有一个标签。该名称的输入字段将是一个基本的单行文本字段，电子邮件的输入字段将是一个只接受电子邮件地址的单行文本字段，而消息的输入字段将是一个基本的多行文本字段。

就HTML代码而言，我们需要如下的东西来实现这些表单小部件：

<form action="/my-handling-form-page" method="post">

<div>

<label for="name">Name:</label>

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

</div>

<div>

<label for="mail">E-mail:</label>

<input type="email" id="mail">

</div>

<div>

<label for="msg">Message:</label>

<textarea id="msg"></textarea>

</div>

</form>

更新您的表单代码，使其看起来像上面的代码。

使用[<div>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/div) 元素可以使我们更加方便地构造我们自己的代码，并且更容易样式化(参见本文后面的文章)。注意在所有[<label>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/label)元素上使用for属性；它是将标签链接到表单小部件的一种正规方式。这个属性引用对应的小部件的id。这样做有一些好处。最明显的一个好处是允许用户单击标签以激活相应的小部件。如果您想更好地理解这个属性的其他好处，您可以找到[如何构造HTML表单的详细信息](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/How_to_structure_an_HTML_form)。

在 [<input>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/input)元素中，最重要的属性是type 属性。这个属性非常重要，因为它定义了[<input>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/input)属性的行为方式。它可以从根本上改变元素，所以要注意它。稍后您将在[原生表单控件](https://developer.mozilla.org/zh-CN/docs/Learn/HTML/Forms/The_native_form_widgets)文章中找到更多关于此的内容。

* 在我们的简单示例中，我们使用值 text 作为第一个输入——这个属性的默认值。它表示一个基本的单行文本字段，接受任何类型的文本输入。
* 对于第二个输入，我们使用值email，它定义了一个只接受格式正确的电子邮件地址的单行文本字段。这会将一个基本的文本字段转换为一种“智能”字段，该字段将对用户输入的数据进行一些检查。在稍后的表单数据验证文章中，您将了解到更多关于[表单验证](https://developer.mozilla.org/zh-CN/docs/Learn/HTML/Forms/Data_form_validation)的信息。

最后但同样重要的是，要注意<input> 和 <textarea></textarea>的语法。这是HTML的一个奇怪之处。 <input> 标签是一个空元素，这意味着它不需要关闭标签。相反， [<textarea>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/textarea)不是一个空元素，因此必须使用适当的结束标记来关闭它。这对HTML表单的特定特性有影响:定义默认值的方式。要定义[<input>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/input)的默认值，你必须使用value 属性，如下所示：

<input type="text" value="by default this element is filled with this text" />

相反，如果您想定义[<textarea>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/textarea)的默认值，您只需在[<textarea>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/textarea)元素的开始和结束标记之间放置默认值，就像这样:

<textarea>by default this element is filled with this text</textarea>

[**<button>**](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/button)**元素**  
我们的表格已经快准备好了，我们只需要再添加一个按钮，让用户在填写完表单后发送他们的数据。这是通过使用 [<button>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/button) 元素完成的。在 </form>这个结束标签上方添加以下内容：

<div class="button">

<button type="submit">Send your message</button>

</div>

您会看到[<button>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/button)元素也接受一个 type属性，它接受submit, reset或者 button 三个值中的任一个。

* 单击 type 属性定义为 submit 值(也是默认值)的按钮会发送表单的数据到[<form>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/form)元素的action 属性所定义的网页。
* 单击 type 属性定义为 reset 值的按钮 将所有表单小部件重新设置为它们的默认值。从用户体验的角度来看，这被认为是一种糟糕的做法。
* 单击 type 属性定义为 button 值的按钮……不会发生任何事！这听起来很傻，但是用JavaScript构建定制按钮非常有用。

### 3.5 HTML 表格

<table>

<tr>

<td>&nbsp;</td>

<td>Knocky</td>

<td>Flor</td>

<td>Ella</td>

<td>Juan</td>

</tr>

<tr>

<td>Age</td>

<td>16</td>

<td>9</td>

<td>10</td>

<td>5</td>

</tr>

<tr>

<td>Owner</td>

<td>Mother-in-law</td>

<td>Me</td>

<td>Me</td>

<td>Sister-in-law</td>

</tr>

</table>

## 4.CSS

### 4.1基本定义

* CSS 指层叠样式表 (**C**ascading **S**tyle **S**heets)
* 样式定义**如何显示** HTML 元素
* 样式通常存储在**样式表**中
* 把样式添加到 HTML 4.0 中，是为了**解决内容与表现分离的问题**
* **外部样式表**可以极大提高工作效率
* 外部样式表通常存储在 **CSS 文件**中
* 多个样式定义可**层叠**为一个

CSS 实例

CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明:



选择器通常是您需要改变样式的 HTML 元素。

每条声明由一个属性和一个值组成。

属性（property）是您希望设置的样式属性（style attribute）。每个属性有一个值。属性和值被冒号分开。

CSS声明总是以分号(;)结束，声明总以大括号({})括起来:

p {color:red;text-align:center;}

为了让CSS可读性更强，你可以每行只描述一个属性:

p  
{  
color:red;  
text-align:center;  
}

CSS 注释

注释是用来解释你的代码，并且可以随意编辑它，浏览器会忽略它。

CSS注释以 "/\*" 开始, 以 "\*/" 结束, 实例如下:

/\*这是个注释\*/  
p  
{  
text-align:center;  
/\*这是另一个注释\*/  
color:black;  
font-family:arial;  
}

### 4.2 CSS Id 和 Class

如果你要在HTML元素中设置CSS样式，你需要在元素中设置"id" 和 "class"选择器

id 选择器

id 选择器可以为标有特定 id 的 HTML 元素指定特定的样式。

HTML元素以id属性来设置id选择器,CSS 中 id 选择器以 "#" 来定义。

以下的样式规则应用于元素属性 id="para1":

#para1 { text-align:center; color:red; }

class 选择器

class 选择器用于描述一组元素的样式，class 选择器有别于id选择器，class可以在多个元素中使用。

class 选择器在HTML中以class属性表示, 在 CSS 中，类选择器以一个点"."号显示：

在以下的例子中，所有拥有 center 类的 HTML 元素均为居中。

.center {text-align:center;}

你也可以指定特定的HTML元素使用class。

在以下实例中, 所有的 p 元素使用 class="center" 让该元素的文本居中:

p.center {text-align:center;}

### 4.3 CSS应用

插入样式表的方法有三种:

* 外部样式表(External style sheet)
* 内部样式表(Internal style sheet)
* 内联样式(Inline style)

外部样式表

当样式需要应用于很多页面时，外部样式表将是理想的选择。在使用外部样式表的情况下，你可以通过改变一个文件来改变整个站点的外观。每个页面使用 <link> 标签链接到样式表。 <link> 标签在（文档的）头部

<head> <link rel="stylesheet" type="text/css" href="mystyle.css"> </head>

浏览器会从文件 mystyle.css 中读到样式声明，并根据它来格式文档。

外部样式表可以在任何文本编辑器中进行编辑。文件不能包含任何的 html 标签。样式表应该以 .css 扩展名进行保存。下面是一个样式表文件的例子：

hr {color:sienna;}

p {margin-left:20px;}

body {background-image:url("/images/back40.gif");}

*不要在属性值与单位之间留有空格（如："margin-left: 20 px" ），正确的写法是 "margin-left: 20px" 。*

内部样式表

当单个文档需要特殊的样式时，就应该使用内部样式表。你可以使用 <style> 标签在文档头部定义内部样式表，就像这样

<head>

<style> hr {color:sienna;} p {margin-left:20px;} body {background-image:url("images/back40.gif");}

</style>

</head>

内联样式

由于要将表现和内容混杂在一起，内联样式会损失掉样式表的许多优势。请慎用这种方法，例如当样式仅需要在一个元素上应用一次时。

要使用内联样式，你需要在相关的标签内使用样式（style）属性。Style 属性可以包含任何 CSS 属性。本例展示如何改变段落的颜色和左外边距：

<p style="color:sienna;margin-left:20px">这是一个段落。</p>

多重样式

如果某些属性在不同的样式表中被同样的选择器定义，那么属性值将从更具体的样式表中被继承过来。

例如，外部样式表拥有针对 h3 选择器的三个属性：

h3 { color:red; text-align:left; font-size:8pt; }

而内部样式表拥有针对 h3 选择器的两个属性

h3 { text-align:right; font-size:20pt; }

假如拥有内部样式表的这个页面同时与外部样式表链接，那么 h3 得到的样式是：

color:red; text-align:right; font-size:20pt;

即颜色属性将被继承于外部样式表，而文字排列（text-alignment）和字体尺寸（font-size）会被内部样式表中的规则取代

多重样式优先级

样式表允许以多种方式规定样式信息。样式可以规定在单个的 HTML 元素中，在 HTML 页的头元素中，或在一个外部的 CSS 文件中。甚至可以在同一个 HTML 文档内部引用多个外部样式表。

一般情况下，优先级如下：

**内联样式）Inline style > （内部样式）Internal style sheet >（外部样式）External style sheet > 浏览器默认样式**

<head> <!-- 外部样式 style.css --> <link rel="stylesheet" type="text/css" href="style.css"/> <!-- 设置：h3{color:blue;} --> <style type="text/css"> /\* 内部样式 \*/ h3{color:green;} </style> </head> <body> <h3>测试！</h3> </body>

***注意：****如果外部样式放在内部样式的后面，则外部样式将覆盖内部样式。*

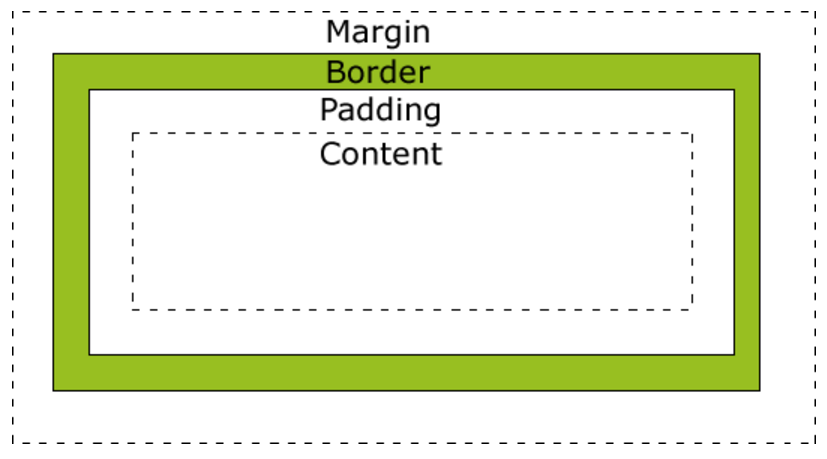
### 4.4CSS 盒子模型

所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容。

盒模型允许我们在其它元素和周围元素边框之间的空间放置元素。

下面的图片说明了盒子模型(Box Model)：



不同部分的说明：

* **Margin(外边距)** - 清除边框外的区域，外边距是透明的。
* **Border(边框)** - 围绕在内边距和内容外的边框。
* **Padding(内边距)** - 清除内容周围的区域，内边距是透明的。
* **Content(内容)** - 盒子的内容，显示文本和图像

为了正确设置元素在所有浏览器中的宽度和高度，你需要知道的盒模型是如何工作的。