红色字：重点 黄色字：暂时忽略 绿色字：有待研究 蓝色：方法名

目录

[第一章 HTML 13](#_Toc4560)

[1. bgcolor,bgColor和background-color的区别 13](#_Toc30111)

[2. HTML全局属性（16个） 13](#_Toc27198)

[3. 水平线如何创建,如何换行？ 16](#_Toc24940)

[4. HTML注释 16](#_Toc7466)

[5.HTML多余的空格和空行会如何处理？浏览器窗口大小改变，段落行数是否出现变化？ 16](#_Toc25801)

[5. 文本格式化 16](#_Toc16725)

[6. 计算机输出标签 17](#_Toc26026)

[7. 引用和术语定义 17](#_Toc3630)

[8. HTML的target，name属性 18](#_Toc18891)

[9. HTML图像 20](#_Toc22117)

[10 HTML表格 22](#_Toc5496)

[11 HTML列表 24](#_Toc8)

[12 HTML块级元素和内联元素；span元素 25](#_Toc1458)

[13. HTML类 26](#_Toc11925)

[14 HTML5语义化 27](#_Toc10356)

[15 HTML响应式web设计 28](#_Toc14000)

[16. HTML框架 30](#_Toc19488)

[17 HTML头部元素 31](#_Toc28756)

[18. HTML字符实体 31](#_Toc10079)

[19. URL 32](#_Toc25527)

[20. HTML表单 32](#_Toc25849)

[21. HTML<input>输入类型 36](#_Toc28224)

[22. web存储 38](#_Toc23409)

[23.主流浏览器的内核 39](#_Toc10304)

[24. HTML5创建新元素 39](#_Toc13920)

[25. HTML5 Canvas 40](#_Toc22855)

[26. SVG 43](#_Toc18415)

[27. 拖放 43](#_Toc30797)

[第二章 CSS 45](#_Toc10978)

[1. 样式表划分 45](#_Toc24076)

[2. 选择器的分组 45](#_Toc25088)

[3. CSS继承 45](#_Toc17212)

[4. 派生选择器；子元素选择器；id选择器；类选择器；相邻兄弟选择器 46](#_Toc18416)

[5. 属性选择器 47](#_Toc12684)

[6. 背景 48](#_Toc31421)

[7. 文本 50](#_Toc1425)

[8. 字体 52](#_Toc2195)

[9. 链接（锚） 54](#_Toc20737)

[10. 列表 55](#_Toc3975)

[11. 表格 56](#_Toc10229)

[12. 轮廓 59](#_Toc20571)

[13. CSS盒模型 60](#_Toc30983)

[14. 定位 63](#_Toc29853)

[15. 伪类，伪元素 67](#_Toc20351)

[16. 水平对齐 68](#_Toc2713)

[17. CSS尺寸 69](#_Toc6141)

[18. 媒介类型 70](#_Toc3565)

[19. CSS3弹性盒子 71](#_Toc819)

[20. 水平垂直居中 73](#_Toc18893)

[21. nth-child(CSS3) 77](#_Toc16302)

[22. nth-of-type(CSS3) 77](#_Toc2603)

[23. 77](#_Toc19144)

[第三章 JavaScript 78](#_Toc31586)

[1. 输出 78](#_Toc23088)

[2. JS数据类型 78](#_Toc21537)

[3. 对象 80](#_Toc10927)

[4. 事件句柄 81](#_Toc921)

[5. 字符串方法 83](#_Toc9741)

[6. 数字 86](#_Toc11846)

[7. 数组 88](#_Toc8528)

[8. 数组排序 92](#_Toc21956)

[9. 日期 92](#_Toc15777)

[10. 数学 93](#_Toc11409)

[11. Switch 93](#_Toc2455)

[12. 类型转换 93](#_Toc23732)

[13. 正则表达式 94](#_Toc11715)

[14. 关键词 95](#_Toc143)

[15. let,const 96](#_Toc1711)

[16. 浏览器内核 96](#_Toc18292)

[17. JSON 97](#_Toc24488)

[18. 回调函数 99](#_Toc19380)

[19. 闭包 99](#_Toc23821)

[20. 浏览器兼容 99](#_Toc3640)

[21. 创建对象的几种方法 100](#_Toc23043)

[22. JS继承 105](#_Toc27631)

[23. websocket 108](#_Toc23729)

[24. 事件机制 108](#_Toc17296)

[25. 性能优化 110](#_Toc7537)

[26. 状态码 111](#_Toc30336)

[27. JS进阶 112](#_Toc16552)

[28. async defer 113](#_Toc1505)

[第四章 计算机网络 114](#_Toc8213)

[1. DNS 114](#_Toc21774)

[2. 浏览器输入一个URL之后会发生什么 114](#_Toc10198)

[3. GET和POST 115](#_Toc9494)

[第五章 AngularJS 117](#_Toc11391)

[1. 重点 117](#_Toc14580)

[2. angular.js导入 117](#_Toc2802)

[3. 基本指令 117](#_Toc20627)

[4. 表达式 121](#_Toc10195)

[5. 应用 121](#_Toc22749)

[6. 数字，字符串，数组，对象 122](#_Toc21458)

[7. 作用域 122](#_Toc24938)

[8. 控制器 123](#_Toc807)

[9. 过滤器 123](#_Toc11408)

[10. 服务 123](#_Toc31209)

[(1) $http 124](#_Toc5330)

[(2) $timeout 125](#_Toc23596)

[11. 选择框 125](#_Toc6925)

[12. 表格 125](#_Toc18446)

[13. HTML DOM 126](#_Toc19157)

[14. AngularJS事件 126](#_Toc4369)

[15. AngularJS模块 126](#_Toc20184)

[16. 依赖注入 126](#_Toc12158)

[17. 路由 127](#_Toc7496)

[18. console.log和alert测试的区别 128](#_Toc19364)

[19. ng-class 128](#_Toc9365)

[第六章 ES6 130](#_Toc19493)

[1. let和const 130](#_Toc25344)

[2. 变量的解构赋值 130](#_Toc18363)

[3. 字符串的扩展 134](#_Toc26581)

[4. 字符串新增方法 136](#_Toc32281)

[5. 正则的扩展 137](#_Toc29507)

[6. 数值的扩展 139](#_Toc22738)

[7. 函数的扩展 141](#_Toc21040)

[8. 数组的扩展 144](#_Toc15464)

[9. 对象的扩展 146](#_Toc15741)

[10. 对象的新增方法 148](#_Toc29429)

[11. Symbol 150](#_Toc148)

[12. Set数据结构 152](#_Toc1299)

[13. WeakSet数据结构 153](#_Toc2356)

[14. Map数据结构 154](#_Toc15073)

[15. WeakMap 156](#_Toc21212)

[16. Proxy 156](#_Toc3674)

[17. Reflect 157](#_Toc29246)

[18. Promise对象 158](#_Toc24934)

[19. Iterator 160](#_Toc27044)

[20. for...of 161](#_Toc7482)

[21. Generator 161](#_Toc30002)

[22. Generator 异步编程 165](#_Toc19983)

[23. async 166](#_Toc13916)

[24. Class 166](#_Toc9525)

[第七章 TypeScript 170](#_Toc8220)

[1. ： 170](#_Toc23813)

[2. 编译 170](#_Toc20892)

[3. 原始数据类型 170](#_Toc5457)

[4. 任意值 170](#_Toc15221)

[5. 类型推论 171](#_Toc13361)

[6. 联合类型 171](#_Toc2999)

[7. 对象的类型-接口 171](#_Toc30307)

[8. 数组的类型 172](#_Toc32594)

[9. 函数的类型 173](#_Toc29601)

[10. 类型断言 175](#_Toc27539)

[11. 引用声明文件 175](#_Toc16013)

[12. 内置对象 175](#_Toc10459)

[13. 类型别名 175](#_Toc17709)

[14. 字符串字面量类型 176](#_Toc4024)

[15. 元祖 176](#_Toc32262)

[16. 枚举 176](#_Toc1023)

[第八章 前端3D建模 177](#_Toc3930)

[1. 基本原理 177](#_Toc9638)

[2. WebGL图形流水线总体结构 177](#_Toc18713)

[3. 顶点着色器 178](#_Toc15718)

[4. 图元装配 178](#_Toc20350)

[5. 光栅化 179](#_Toc27500)

[6. 片段着色器（像素着色器） 179](#_Toc881)

[7. 逐片段操作 180](#_Toc4263)

[第九章 less 181](#_Toc24357)

[1. 变量名 181](#_Toc28534)

[2. 函数 181](#_Toc28819)

[3. 模板 181](#_Toc9916)

[4. 扩展 182](#_Toc10420)

[5. Mixins(混合) 185](#_Toc679)

[6. 导入 187](#_Toc11692)

[7. Guard运算符 187](#_Toc14515)

[8. 循环 188](#_Toc3916)

[9. 合并 188](#_Toc4264)

[10. 父选择器 189](#_Toc30861)

[第十章 Node.js（网上教程） 190](#_Toc19109)

[1. 概念 190](#_Toc3569)

[2. nvs 190](#_Toc21119)

[3. 组成部分 190](#_Toc26612)

[4. 创建简单的服务器 190](#_Toc4844)

[5. NPM 191](#_Toc20972)

[6. REPL（交互式解释器） 192](#_Toc3939)

[7. 回调函数 193](#_Toc30989)

[8. 事件循环 193](#_Toc24977)

[9. 事件 193](#_Toc21613)

[10. Buffer（缓冲区） 194](#_Toc24526)

[11. Stream（流） 196](#_Toc5688)

[12. 模块系统 197](#_Toc7172)

[13. 函数 198](#_Toc14197)

[14. 路由 198](#_Toc9941)

[15. 全局对象 198](#_Toc31501)

[16. util 199](#_Toc8464)

[17. 文件系统 200](#_Toc27123)

[18. 工具模块 201](#_Toc13749)

[19. Web模块 202](#_Toc6557)

[20. Express框架 202](#_Toc10509)

[21. RESTful API 204](#_Toc22318)

[22. 多进程 204](#_Toc19886)

[23. JXcore 205](#_Toc25851)

[第十一章 《深入浅出Node.js》 206](#_Toc30296)

[1. Node的特点 206](#_Toc9570)

[2. 模块 206](#_Toc29716)

[3. 异步I/O 207](#_Toc9468)

[4. 非I/0的异步API 210](#_Toc27136)

[(3) setImmediate 210](#_Toc18350)

[5. 异步编程注意点 211](#_Toc19619)

[6. 事件发布/订阅模式 211](#_Toc14138)

[7. Promise/Deferred模式 212](#_Toc20674)

[8. 流程控制 212](#_Toc26413)

[9. 异步并发控制 214](#_Toc28179)

[10. 内存控制概要 214](#_Toc24927)

[11. 垃圾回收 214](#_Toc31800)

[12. 内存泄漏 216](#_Toc26468)

[13. 大内存应用 216](#_Toc25836)

[14. Buffer结构 217](#_Toc6304)

[15. Buffer的转换 218](#_Toc31821)

[16. Buffer的拼接 218](#_Toc2524)

[17. Buffer与性能 219](#_Toc22684)

[18. 网络编程 219](#_Toc25223)

[19. TCP 220](#_Toc3316)

[20. UDP 221](#_Toc29967)

[21. HTTP 222](#_Toc1734)

[22. WebSocket服务 224](#_Toc13046)

[23. 网络服务安全 225](#_Toc3098)

[24. cookie 226](#_Toc8841)

[25. Session 226](#_Toc2270)

[26. 缓存 227](#_Toc27497)

[27. 数据上传 227](#_Toc3032)

[28. MVC 228](#_Toc30791)

[29. RESTful 228](#_Toc6131)

[30. 中间件 228](#_Toc24443)

[31. 页面渲染 229](#_Toc21742)

[32. 进程 229](#_Toc4451)

[33. 单元测试 231](#_Toc17099)

[第十二章 React 233](#_Toc684)

[1. 初识 233](#_Toc28013)

[2. 元素渲染 233](#_Toc31219)

# HTML

## bgcolor,bgColor和background-color的区别

bgcolor和bgColor都是设置某些标签的背景颜色。bgcolor是静态（HTML），bgColor是动态（JScript）。注：<p>标签不支持bgcolor。

Background-color属于CSS样式表。

用法：bgcolor=”背景颜色” style=“background-color:背景颜色”

注：不要使用bgcolor，color，align等HTML属性，用css代替；能用css代替的HTML标签也用css实现。

## HTML全局属性（16个）

accesskey: 使用快捷键激活元素 例如accesskey=’h’,用快捷键alt+’h’可直接激活（访问）当前元素（链接）。

class: 类名 大多用于样式表中的类，也可以利用它通过Javascript来改变带有指定class的HTML元素。注：class不能在HTML标签中使用，如head、html、script、style，以及其他某些元素，如base、meta、param、title。

contenteditable（HTML5）: 规定元素内容是否可编辑，既可以直接在网页上修改内容而不用再在HTML文件内修改元素内容。用法：contenteditable="true|false"。注：此属性可从父元素继承。

contextmenu（HTML5）: 规定元素的上下文菜单，上下文菜单在用户点击元素时显示。注：只有FireFox支持。

data-\*（HTML5）： 用于存储页面或应用程序的私有自定义数据，即HTML元素的自定义属性。可以被JavaScript调用。用法：data-animal-type=”bird”;

dir: 规定元素内容在浏览器文本方向。ltr为默认，浏览器显示从左向右；rtl浏览器显示是从右向左。注：dir不能用于<base>, <br>, <frame>, <frameset>, <hr>, <iframe>, <param> 以及 <script>；dir指的不是文字的方向，而是在浏览器从左边开始显示还是从右边开始显示。

draggable（HTML5）: 规定元素在浏览器中是否可拖动。用法：draggable="true|false|auto"，即可拖动、不可拖动、浏览器默认行为。复杂，暂时不会使用中。

dropzone: 规定在拖动数据时是否对该数据进行复制、移动或链接。注：主流浏览器都不支持。

hidden（HTML5）: 规定元素仍未或不再相关。hidden是布尔属性。设置后浏览器不会显示该属性，除非匹配某些条件。JavaScript可以删除该属性，使元素可见。用法：hidden="hidden"。

id: 规定元素的唯一id。id在文档中必须唯一。

lang: 规定元素内容的语言。注：lang 属性在以下标签中无效：<base>, <br>, <frame>, <frameset>, <hr>, <iframe>, <param> 以及 <script>。

spellcheck(HTML5): 规定是否对元素进行拼写和语法检查。只对三种内容进行拼写检查：1.input元素中的文本值；2.<textarea>元素中的文本；3.可编辑元素中的文本（contenteditable）注：不怎么好用，还未成功实现过。

style： 规定元素的行内CSS样式。注：style属性覆盖任何全局的样式设定。

tabindex: 规定元素的tab键次序。用法：tabindex="number"，number是控制次序，1是第一个。注：以下元素支持 tabindex 属性：<a>, <area>, <button>, <input>, <object>, <select> 以及 <textarea>。

title: 规定元素的额外信息。通常鼠标移到元素上会显示。

translate（HTML5）: 规定不应该翻译某些元素。用法：translate="yes|no"。注：所有主流浏览器不支持。

## 水平线如何创建,如何换行？

<hr />标签在HTML页面中创建水平线。

<br />标签用于换行。

## HTML注释

<!--\*\*\*-->

## 5.HTML多余的空格和空行会如何处理？浏览器窗口大小改变，段落行数是否出现变化？

HTML会将多余的空格显示为一个，并移除空行。

调节浏览器窗口大小，会改变段落中的行数。

## 文本格式化

部分已不被使用，部分建议用CSS样式替代

<b>: 加粗

<big>: 加大

<em>: 着重（略微加粗）

<i>: 斜体

<small>: 变小

<strong>: 加重语气

<sub>: 下标

<sup>: 上标

<ins>: 插入字（下划线，配合del使用）

<del>: 删除字（配合ins使用）

## 计算机输出标签

本质上就是改变文本格式，即css样式或格式化。类似语义化，但附加了css效果？

<code>: 定义计算机代码

<kbd>: 定义键盘码

<samp>: 定义计算机代码样本

<tt>: 定义打字机代码

<var>: 定义变量

<pre>: 定义预格式文本

## 引用和术语定义

本质上就是改变文本格式，即css样式或格式化。类似语义化，但附加了css效果？

<abbr>: 定义缩写(语义化)

<acronym>: 定义首字母缩写

<bdo>: 定义文字方向。ltr:默认，从左向右；rtl:从右向左。

<blockquote>: 定义长的引用（缩进）

<q>: 定义短的引用语（加引号）

<cite>: 定义引用，引证（斜体）

<dfn>: 定义一个项目的定义（斜体，可用abbr代替使用）

<address>: 定义文档或文章的联系信息。（斜体）

## HTML的target，name属性

超链接也叫锚

1. 超链接默认在原页面打开，相当于target=”\_self”。
2. 若target=”\_blank”,则链接会在新窗口打开。
3. target=“\_parent”,文档载入父窗口或父框架，若不存在父窗口或父框架则相当于“\_self”。
4. target=“\_top”，清除所有被包含的框架并将文档载入整个浏览器窗口。
5. target=“框架名字”，用在在指定框架中打开窗口，既页面窗口的切换。

例如：

<frameset cols="100,\*">

<frame src="toc.html">

<frame src="pref.html" name="view\_frame">

</frameset>

“toc.html”:

<ul>

<li><a href="pref.html" target="view\_frame">Preface</a></li>

<li><a href="chap1.html" target="view\_frame">Chapter 1</a></li>

<li><a href="chap2.html" target="view\_frame">Chapter 2</a></li>

<li><a href="chap3.html" target="view\_frame">Chapter 3</a></li>

</ul>

注：frame和frameset已被删除

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

name规定链接的名字，通常用于页内跳转，也可以用id属性代替。

例如：

<a name=”tips”>跳转</a>

<a href=”#tips”>至此</a>

也可以跳转其他页面的某处，例如：

<a href="http://www.w3school.com.cn/html/html\_links.asp#tips">跳转至其他页某处</a>

注：正斜杠（“/”）需要用于子文件夹前

## HTML图像

用法：<img src=”url” />。<img>是空标签，只包含属性，没有结束标签。src的值是图像的URL地址，例如<http://www.w3school.com.cn/images/boat.gif>。

alt属性：当图像无法加载时，在原位置出现的提示文本信息。例如alt=”wrong”

opacity属性：改变透明度。用法：

img{

opacity:0.4;

}

浏览器背景图片用法：<body background="/i/eg\_background.jpg">，gif和jpg文件均可用作HTML背景，若图像较小则会重复填满。

align属性：可设置图像相对文本的位置。bottom为默认底部；middle为文本在图像中段；top为文本在图像顶端；left为图像在文本左侧；right为图像在文本右侧。

usemap属性：创建带有可点击区域的图像映射。<map>为定义图像地图，<area>为定义图像地图中的可点击区域。用法：

<img src=”” usemap=”#planetmap”>

<map name=”planetmap” id=”planetmap”>

//name是为了对应usemap，id是为了引入CSS

<area shape="circle" coords="180,139,14" href ="venus.html" alt="Venus" />

<area shape="circle" coords="129,161,10" href ="mercur.html" alt="Mercury" />

<area shape="rect" coords="0,0,110,260" href ="sun.html" alt="Sun" />

</map>

//shape属性定义映射区域的形状，rect为矩形；circle为圆形；poly为多边形。

//coords定义了形状的坐标与半径，矩形为“x1,x2,y1,y2”即左上角和右下角的坐标；圆形为”x1,y1,r”即圆心坐标和半径；多边形为“x1,y1,x2,y2,…,xn,yn,x1,y1”即n个角的坐标

ismap属性：点击图像某处，浏览器会把坐标以URL查询字符串的形式发给服务器。用法：<img ismap="ismap" />。注：只有当 <img> 元素属于带有有效 href 属性的 <a> 元素的后代时，才允许 ismap 属性。

## 10 HTML表格

格式：

<table>

<caption>表格标题</caption>

<thead> //thead定义表格的页眉

<th>

<td></td>

</th>

</thead>

<tbody> //tbody定义表格的主题

<tr>

<td></td>

</tr>

</tbody>

<tfoot> //tfoot定义表格的页脚

<tr>

<td></td>

</tr>

</tfoot>

</table>

border=”1”: 边框宽度大小。默认为“0”即不显示边框。

&nbsp: 空格占位符。用于空的单元格，便于正确显示单元格边框。

表头可水平可垂直。

colspan属性：合并列单元格 rowspan属性: 合并行单元格

用法实例：<th colspan=”number”>,number表示合并单元格的数量，超出则不会显示。

cellpadding:单元格内容与边框的距离 cellspacing:单元格之间的距离（包括与主框架的距离）

frame：控制围绕表格的边框。box是四周都显示；above只显示上面的；below只显示下面的；hsides只显示上下的；vsides只显示左右的。

用法示例：<table frame=”box”>

注：<col> <colgroup>暂未实现

## 11 HTML列表

无序列表，黑色圆点为行头标记。用法：

<ul>

<li> </li>

<li> </li>

</ul>

type属性可改变行头标记形状。例如：<ul type=”circle”>，标记为空心圆。

有序列表，数字为行头标记。用法：

<ol>

<li> </li>

<li> </li>

</ol>

type属性可改变行头标记形状。例如：<ol type=”a”>，标记为小写英文字母a,b…。”A”,标记为大写英文字母；”I”,标记为罗马字母列表；”i”,标记为小写罗马字母列表。

自定义列表，列表项加列表项的定义，主要多了个列表项。用法：

<dl>

<dt>计算机</dt>

<dd>用来计算的仪器 ... ...</dd>

<dt>显示器</dt>

<dd>以视觉方式显示信息的装置 ... ...</dd>

</dl>

## 12 HTML块级元素和内联元素；span元素

HTML块级元素:<address>、<center>、<h1>-<h6>、<hr>、<p>、<pre>、<blockqupte>、<marquee>、<ul>、<ol>、<dl>、<table>、<form>、<div>

总是从新的一行开始

高度与宽度总是可控的

宽度没有设置时，默认为100%

块级元素可以包含块级元素和内联元素

HTML内联元素：<span>、<a>、<br>、<img>、文本格式化、<input>、<textarea>、<select>

与其他元素在同一行

宽度，高度和内边距是不可控的

Margin和padding只有左右有效

宽高就是内容高度

行内元素只能嵌套行内元素

<span>：使框定的内联元素显示特定效果，一般是特别的CSS样式。

## 13. HTML类

1. 分类块级元素

用法：

<style>

div.xx{ //或者.xx{}

}

</style>

…..

<div class=”xx”>

1. 分类内联元素

用法：

<style>

span.xx{

}

</style>

…..

<h1><span class=”xx”>\*\*</span></h1>

## 14 HTML5语义化

语义元素清晰地向浏览器和开发者描述其意义。让机器可以读懂内容。

HTML5语义元素：

header: 定义文档或小节的页眉

hgroup: 组合标题

section: 定义文档的小节

article: 定义独立包含的文章

main: 规定主内容

footer: 定义文档或小节的页脚

nav: 定义导航链接

aside: 定义内容之外的东西（比如侧栏）

details: 定义额外的细节

summary: 定义details元素的标题

figure: 组合图片（img）和描述（figcaption）

figcaption: 为图片添加可见的描述或解释

time: 定义时间/日期

mark: 定义强调的文本

bdi: 允许设置一段文本，使其脱离父元素的文本方向设置。

command: 定义命令按钮

dialog: 定义对话框

meter： 定义度量衡（数据范围）

progress: 定义进度

ruby: 定义音标之类

用法：<ruby>

汉 <rt>Han</rt><rp>无法使用ruby</rp>

</ruby>

<rt>: 定义字符的发音，ruby联动使用。

<rp>: 定义不支持ruby时显示的内容

<wbr>: 规定何处适合添加换行符（宽度足够不换行，宽度不够则换行）

优点：

1. 优化内容、代码结构
2. 更有可读性
3. 便于搜索引擎搜索

## 15 HTML响应式web设计

响应式Web设计是指以可变尺寸传递网页。多用于电脑、平板、移动设备显示相同网页时因为不同屏幕大小而使网页显示不同结构。

1. 自己创建。使用CSS样式：float:left
2. Bootstrap: 最流行的开发响应式web的HTML,CSS,JS框架
3. @media query: CSS3的媒体查询功能

用法：

<style>

@media screen and (min-width: 1000px) {

//像素大于1000px时显示

.nav{}

}

@media screen and (min-width: 640px) and (max-width: 1000px) {

//像素小于1000px,大于640px时显示

.nav{}

}

@media screen and (max-width: 640px) {

//像素小于640px时显示

.nav{}

}

</style>

## 16. HTML框架

在一个浏览器窗口中显示多个HTML页面

frameset: 框架结构标签。可嵌套。

frame： 框架标签

cols垂直划分窗口，rows水平划分窗口，值为占据屏幕的面积。

用法示例1：

<frameset cols="25%,75%">

<frame src="frame\_a.html">

<frame src="frame\_b.html">

</frameset>

用法示例2：

<frameset cols="120,\*">

<frame src="frame\_a.html">

<frame src="frame\_b.html">

</frameset>

注：<body>不能与<frameset>同时使用

frame可跳转至框架内的某个id处

用法：<frame src=”/a.html#C10”>

noframes: 浏览器不支持框架时运行的内容

noresize: 禁止在浏览器窗口拖动框架边框。用法：noresize=”noresize”

iframe: 定义内联框架。用于在网页内显示网页。

frameborder：是否显示iframe周围边框，0不显示，1显示。

iframe可作为target的目标，即内联框架内跳转。

注：内联框架用于<body>标签内

## 17 HTML头部元素

head: 定义关于文档的信息。下面元素都可放入head中。

title： 定义文档标题

base： 定义页面上所有链接的默认地址或默认target属性值

link: 定义文档和外部资源的关系。多用于链接CSS样式表。

meta: 定义关于文档的数据信息。对于机器是可读的。

script: 定义脚本内容。例如JavaScript

style: 定义CSS样式。

## 18. HTML字符实体

HTML中有些字符是不能直接使用的，主要是标签会用到。若要用这些字符则要用字符实体代替。

用法示例：

”<”小于号：&entity\_name

不间断空格：&nbsp

## 19. URL

统一资源定位器，也叫网址

因特网服务类型：

http: 超文本传输协议。普通网页，不加密。

https: 安全超文本传输协议。安全网页，加密所有信息交换。

ftp: 文件传输协议。用于将文件下载或上传至网站。

file： 本地计算机上的文件。

## 20. HTML表单

HTML表单用于搜集不同类型的用户输入

form: 定义表单。用法：<form></form>

action属性: 定义在提交表单时执行的动作。通常是使用提交按钮，也可以指定某个服务器脚本来处理被提交的表单。

method：规定提交表单用哪种HTTP方法(GET/POST)。默认GET。

accept-charset: 规定在被提交表单中使用的字符集。默认页面字符集。

autocomplete(HTML5)：规定浏览器应该自动完成表单。默认开启。当页面返回时依旧能显示原本填写的数据。

enctype: 规定被提交数据的编码。默认URL-encoded.

novalidate: 规定浏览器不验证表单。

target： 规定action属性中地址的目标。默认\_self。

name: 用于表单元素。注：若没有name属性则该字段不会被提交。

autofocus(HTML5): 用于表单元素。在页面加载时自动获得焦点。即不用找寻输入的地方，页面加载完成后直接就可以输入。一般用于input。注：这个属性只对第一个有这个属性的表单元素生效。

novalidate(HTML5): 提交表单时不会验证元素格式（如email格式等）。

formaction(HTML5): 表单提交时本属性会覆盖原本的action属性。多用于提交到不同的URL。注：只能用于submit和image。

formenctype(HTML5):表单提交到服务器的数据编码。覆盖enctype属性。

注：只能用于submit和image；只能用于POST方法。

formmethod(HTML5):表单提交方式。覆盖method属性。注：只能用于submit和image。

formnovalidate(HTML5):表单提交无需被验证。覆盖novalidate属性。注：只能用于submit。

formtarget(HTML5):覆盖target属性。注：只能用于submit和image。

fieldset：为包围起来的表单元素添加外边框。

legend: 为fieldset添加标题。

用法：

<form>

<fieldset>

<legand>标题</legand>

<input>

<input>

</fieldset>

</form>

<input>: 最重要的表单元素。根据不同的type属性变换形态。（详情见21）

<select>：定义下拉列表。

<option>: 定义选项。通常首个选项为默认被选选项。亦或使用selected定义默认选项。

用法：

<select name="cars">

<option value="volvo">Volvo</option> //无人工定义，则为默认选项

<option value="saab">Saab</option>

<option value="fiat">Fiat</option>

<option value="audi" selected>Audi</option> //人工定义默认选项

</select>

<textarea>: 定义多行输入，即文本域。

用法：

<textarea rows=”10” cols=”30”>

\*\*\*\*\*\*\*\*\*

<textarea>

<button>: 定义可点击的按钮。用法：<button type=”button”>按钮<button>

<datalist>(HTML5): 规定输入域的选项列表。使用list属性绑定输入域和datalist，列表元素由option创建。

list(HTML5): 用于规定datalist。

用法：

<form action=" ">

<input list="browsers">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

</form>

注：option元素永远要设置value属性。

<keygen>(HTML5): 提供一种验证用户的方法。注：不是很有用，也未研究。

<output>(HTML5): 用于不同类型的输出。自动输出结果。

## 21. HTML<input>输入类型

定义模板：<input type=”text” />

text: 定义文本输入的单行输入字段

password:定义密码字段。浏览器中会以星号做代替显示。

submit: 定义提交表单按钮。

radio: 定义单选按钮

checkbox: 定义复选框

button：定义按钮。

number（HTML5）: 定义数字的输入字段。可限制范围。

date（HTML5）: 定义日期的输入字段。有日期选择器（年月日）。

color（HTML5）： 定义颜色的输入字段。有颜色选择器。

range(HTML5): 定义一定范围取值字段。有滑块。

month(HTML5): 月份年份字段。有日期选择器（月和年）。

week(HTML5): 周和年字段。有日期选择器（周和年）。

time（HTML5）： 时间字段。有时间选择器（时和分）。

datetime(HTML5): 日期和时间字段（有时区）。有日期选择器。注：大部分浏览器不能用。

datetime-local(HTML5): 日期和时间（无时区）字段。有日期选择器（年月日，时分）。

email（HTML5）: 定义电子邮件地址字段。提交时会自动验证。

search(HTML5): 定义搜索字段。注：语义化？还未理解。

tel(HTML5): 定义电话号码字段。注：大部分浏览器不支持

url(HTML5): 定义URL地址的输入字段。提交时自动验证url字段。

min,max,step(HTML5):包含数字或日期的input的约束。最小，最大，步伐。

multiple(HTML5): input可以选择多个值。适用于email和file（选取多个文件或图片等）。

pattern(HTML5): 一个正则表达式用于验证元素值。

placeholder：

## 22. web存储

cookie：

浏览器在本地存储的文本字符串，向浏览器发送请求时会附带这些数据。（用于为HTTP的无状态附带状态信息）

过了指定时间才会被删除。

不适合大量数据的存储。

数据在同源窗口都是共享的（协议，主机名，端口相同）。

localStorage：

只在本地保存。

存储的数据没有时间限制。

存储数据量大。

数据在同源窗口都是共享的（协议，主机名，端口相同）。

sessionStorage：

只在本地保存。

当用户关闭浏览器窗口后，数据会被删除。

存储数据量大。

数据不能在不同浏览器中共享，即即使是相同域名下的两个页面，只要不在同一个浏览器窗口打开，它们的sessionStorage内容便无法共享。

（协议，主机名，端口，浏览器窗口相同）

## 23.主流浏览器的内核

五大主流浏览器和四大内核

IE Trident IE、猎豹、360、百度

firefox Gecko

Safari webkit

chrome Blink(webkit分支)

Opera blink

## 24. HTML5创建新元素

HTML5可以创建新的元素。

示例：

<style>

newSection{ display:block}

</style>

……….

<newSection> \*\*\*\* </newSection>

## 25. HTML5 Canvas

用于在网页中绘制所需要的图形。

首先创建一个画布（一个矩形框），接受style属性：

<canvas id=”myCanvas” width=”200” height=”100”></canvas>

然后使用JS画图。

var c=document.getElementById(“myCanvas”);

var ctx=c.getContext(“2d”); //getContext(“2d”)是HTML5对象，用于绘制路径，矩形，圆形，字符，图像等。

ctx.fillstyle=”red”;

ctx.fillRect(0,0,150,75)

fillStyle: 填充画的颜色，渐变或模式。

fillRect: 画的填充方式（画布上画一个矩形）。第一个参数和第二个参数是画（矩形图像）的左上角的起始坐标，第三个参数和第四个参数是宽和高

路径：画布上画线。

moveTo(x,y): 定义线条开始坐标

lineTo(x,y): 定义线条结束坐标

arc(): 定义圆形

stroke(): 绘制图像

画线示例：

ctx.moveTo(0,0);

ctx,lineTo(200,100);

ctx.stroke();

画圆示例：

ctx.beginPath();

ctx.arc(95,50,40,0,2\*Math.PI);

ctx.stroke();

文本：画布上写字

font: 定义字体

fillText(text,x,y) 实心文本

strokeText(text,x,y) 空心文本

示例：

ctx.font=”30px Arial”;

ctx.fillText(“hello world”,10,50);

渐变：填充颜色渐变。

createLinearGradient(x,y,x1,y1): 创建线性渐变

createRadialGradient(x,y,r,x1,y1,r1) 创建一个径向/圆渐变。x,y,r是开始圆的参数（圆心，半径）；x1,y1,r1是结束圆的参数

addColorStop(0~1,color): 渐变颜色，0代表起始颜色，1代表结束颜色，若中间要插入渐变则取0到1之间的值

fillStyle值为渐变对象时使用渐变

示例:

grd=ctx.createLinearGradient(0,0,200,0);

grd.addColorStop(0,”red”);

grd.addColorStop(0.5,”black”);

grd.addColorStop(1,”white”);

ctx.fillstyle=grd;

ctx.fillRect(10,10,150,80);

图像：将一幅图像放置到画上

drawImage(image,x,y):放图像

示例：

var img=document.getElementById(“scream”);

ctx.drawImage(img,10,10);

## 26. SVG

SVG和Canvas都是2D做图

SVG表示可缩放矢量图形（矢量图形放大或缩小不会失真）

也是一种绘图工具，通过XML来描述。

DOM中的每个元素都是可以被JS调用的（Canvas不行）

SVG重绘是实时的，Canvas发生变化则整个场景都要重绘。

用法：

<svg width=”100%” height=”100%”>

…………….

</svg>

## 27. 拖放

draggable: 设置该元素允许被拖放（链接和图片默认可拖动）。

用法：<img draggable=”true” />

ondragstart: 开始拖动元素或选择文本时触发

ondrag: 元素拖动过程中触发

ondragend: 完成元素拖动后触发

ondragenter:拖动对象进入容器范围触发

ondragover: 拖动对象在另一个容器范围触发

ondragleave:拖动对象离开容器范围触发

ondrop: 拖动过程中释放鼠标时触发。

示例：

<div ondragstart=”allowStart(event)”>

……..

function allowStart(event){}

setData: 设置被托数据的数据类型和值

示例：

function allowStart(event){

event.dataTranser.setData(“Text”,event.target.id);

}

addEventListener:向指定元素添加事件句柄。

用法：

document.getElementById(“\*\*”).addEventListener(“ondragstart”,function(){})

# 第二章 CSS

## 样式表划分

1. 浏览器缺省设置
2. 外部样式表（link）
3. 内部样式表（head->style）
4. 内联样式（style）

优先级如上所示，4的优先级最高。

## 选择器的分组

即多个标签共享一个声明。示例：

h1,h2,h3{

color:green;

}

## CSS继承

父子：元素嵌套

继承：子元素未定义的某些属性从已定义的父元素中继承

可继承的属性：

所有字体属性。

大部分文本属性：text-indent、text-align、line-height、word-spacing、letter-spacing、text-transform、direction、color。

元素可见性：visibility。

表格布局属性：caption-side、border-collapse、border-spacing、empty-cells、table-layout

列表布局属性：list-style-type、list-style-image、list-style-position、list-style.

生成内容属性：quotes

光标属性：cursor

页面样式属性：page、page-break-inside、windows、orphans

声音样式属性：speak、speak-punctuation、speak-numeral、speak-header、speech-rate、volume、voice-family、pitch、pitch-range、stress、richness、、azimuth、elevation

## 派生选择器；子元素选择器；id选择器；类选择器；相邻兄弟选择器

派生选择器（后代选择器）示例：

li strong{} //对li标签包含的所有strong标签内的元素有效。

子元素选择器示例：

li>strong{} //当strong标签为li标签的子元素时有效（孙子不行）。

id选择器示例：

#red{color：red}

#green{color:green} //不同id不同CSS

id派生选择器示例：

#other p{} //只能在id为other的p标签生效，区分大小写。

类选择器示例：

.center{} //红色的点就是类选择器，后面跟类名，区分大小写。

.center.other{} //同时包含两个类的标签

td.fancy{} //类名为fancy的td标签才生效,多用于对多个相同类型的标签在不同地方的使用

td.fancy2{}

注：id不能重名，类可以重名即不同标签用同一个类。

相邻兄弟选择器示例：

h1+p{} //紧接着h1后面的p标签的CSS，且h1和p有共同的父元素。

## 属性选择器

为拥有指定属性的HTML元素设置样式。

示例：

[title]{} //为带有title属性的元素设置样式

[title=”\*\*”]{} //为带有title属性且值为\*\*的元素设置样式

选择器可以带有符号：

[title ~= “\*\*”] // ”~=”用于选取属性值中包含\*\*这个单词且单词之间以空格隔开的元素

[title |= “\*\*”] // ”|=”用于选取属性值中只有一个\*\*或一个\*\*-开头的单词的元素

[title ^= “\*\*”] // ”^=”用于选取属性值中以\*\*开头的元素

[title $= “\*\*”] // ”$=”用于选取属性值中以\*\*结尾的元素

[title \*= “\*\*”] // ”\*=”用于选取属性值中包含\*\*的元素

属性选择器多用于配合id选择器和类选择器，示例：

input[type=”text”]{ }

input[type=”text”][value]{} //属性可以有多条。

特别的：\*[lang|=”en”]{} //所有元素特定属性选择器。

## 背景

background-color: 背景颜色。用法：{background-color:red}

background-image： 背景图像。用法：{background-image:url(1.jpg)}

background-repeat: 背景图像进行平铺。repeat-x是在水平方向上，repeat-y是在垂直方向上，no-repeat则不允许平铺，默认是铺满整个背景。

用法：{background-image:url(1.jpg); //注：格式范例url(1.jpg)

background-repeat:repeat-x;}

background-position: 改变图像在背景中的位置。具体值分为关键词、百分数值和长度值。关键词有center、top、bottom、right、left;百分数值\*%，\*%，第一个百分数值是水平方向的，第二个百分数值是垂直方向的，默认为0%，0%；长度值\*px,\*px，第一个长度值是向右长度，第二个长度值是向下长度。

示例用法：{background-image：url（1.jpg）;

background-repeat:no-repeat;

background-position:center;} //图像居中

background-position:50%,50% //图像居中

background-position:50px,100px //从内边距左上角向右50像素，向下100像素。

background-attachment:可使图像一直显示在窗口上。用法：background-attachment：fixed。默认为scroll，即当页面下滑时，图像会随文档上滑，即图像固定在页面上。

background-size（CSS3）: 规定背景图片的尺寸。

用法：{background-size：40% 100%} //值为宽度，高度

background-origin（CSS3）： 规定背景图片的地位区域,规定background-position属性相对于什么位置来定位。值有content-box（内容框）,padding-box（默认，内边距框）,border-box（边框盒）。

用法：{background-origin:content-box}

注：边框盒包括内边距框和内容盒，内边距框包括内容框。

注：未理解

background-clip：规定背景的绘制区域，即图像被裁剪。值有content-box（内容框）,padding-box（内边距框）,border-box（默认，边框盒）。

用法：{background-clip:content-box}

注：边框盒包括内边距框和内容盒，内边距框包括内容框。

## 文本

text-indent: 缩进文本。值可以为正值长度、负值长度、百分比值（相对于父元素）

示例：p{text-indent:5em}

text-align：水平对齐。值有left（左对齐）、right（右对齐）、center（居中），justify(两端对齐)，inherit（从父元素继承）

word-spacing: 字之间的标准间隔。默认为0。正值间隔增加，负值间隔减小（字可以重叠）。

letter-spacing: 字母之间的标准间隔。默认为0。正值间隔增加，负值间隔减小（字可以重叠）

text-transform: 处理文本大小写，即大小写转换。值有uppercase(全部大写)、lowercase(全部小写)、capitalize(首字母大写)

text-decoration: 文本装饰。值有underline（加下划线）、overline（加上划线）、line-through(文本中间加贯穿线)、blink（让文本闪烁）、默认为none，可用来清空装饰。

一个声明可用多个装饰。

本属性在父子选择器时子值会完全替换父值而不会继承。

white-space: 影响浏览器处理空白符的方式。值有默认normal（所有空白符合并为一个空格，防止换行除非使用br标签）、pre（空白符和换行不会被忽略）、nowrap（防止换行除非使用br标签）

color： 文本颜色

direction： 设置文本方向

line-height: 设置行高

text-shadow(CSS3): 设置文本阴影。值为向左偏移，向下偏移，模糊距离，阴影颜色。

用法：{text-shadow:5px 5px 5px red}

word-wrap(CSS3): 强制文本进行换行，会拆分单词。

用法：p {word-wrap:break-word;}

text-overflow(CSS3): 当文本溢出时。值有clip（多余的裁剪）、ellipsis（多余的以省略号代替）、多余以给定字符串代替。

用法示例：{ text-overflow: clip|ellipsis|string;}

注：不是很好用

## 字体

font-family: 定义文本的字体。例如宋体之类，也可为元素指定一系列类似的字体。当第一个字体不能用则延续到第二个，若都不能用则为默认字体

用法：

{font-family:Times,TimesNR}

font-style:规定斜体字。值有normal（正常）、italic（斜体）、oblique（倾斜）。italic和oblique基本类似。

font-variant: 设定小型大写字母，即将字母都转为大写字母且大小变小。值为small-caps。用法：{font-variant:small-caps}

font-weight: 设置文本的粗细。值为normal（正常）、值（100~900）、bold（加粗700）。

font-size: 设置文本大小。单位允许em、px、百分比。单位建议用em,1em=16px。em允许浏览器调整字体大小。

font：设置文本的简写属性，即把所有针对字体的属性设置在一个声明当中。顺序是style->varient->weight->size/line-height->family（line-height设置行间距）。至少要设置size和family，其余会自动设置为默认。

cursor: 改变鼠标悬停时的光标。用法：span{cursor:\*\*}.

visibility: 设置元素是否可见。用法：p{visibility:hidden} //元素不可见

特殊：tr.one{visibility:collapse} //可使列表的某行或某列不可见

@font-face(CSS3): 可使用自己下载的字体文件。

用法：

@font-face

{

font-family: myFirstFont;

url('Sansation\_Light.eot');

}

## 链接（锚）

链接可用所处的状态来设置样式。

a:link: 普通的，未被访问的链接

a:visited: 用户已访问的链接

a:hover: 鼠标位于链接上时

a:active: 链接被点击时

示例：

a:link {color:#FF0000;}

a:visited {color:#00FF00;}

a:hover {color:#FF00FF;}

a:active {color:#0000FF;}

注：a:hover 必须位于 a:link 和 a:visited 之后；a:active 必须位于 a:hover 之后。假设顺序是a:visited、a:hover、a:link，结果是把鼠标放到link的链接上时，它并不变成hover时的样式，只有放在visited的链接上，链接才会变成hover时的样式。原因是一个鼠标经过的未访问链接同时拥有a:link、a:hover两种属性，a:link离它最近，所以它优先满足a:link，而放弃a:hover的重复定义。CSS遵循的“就近原则”。

## 列表

list-style-type: 改变列表项的标志类型（圆点，数字）。值有square（方块）、none（无标记）等。用法：ul{list-style-type:square}

list-style-image: 改变标志类型为图像。用法：ul li{list-style-image:url(1.gif)}

list-style-position: 确定标志出现在列表项内容之外还是内容内部。值为inside和outside（默认值）。

list-style: 上述三个属性合并为一个属性。顺序不记，值若忽略则为默认值，且值都可以忽略。

用法：li {list-style : url(1.gif) square inside}

## 表格

border: 设置表格边框。可单独设置某条边框。

用法：

table,th,td{border:1px solid blue;} //粗细，线类型，颜色

table{border-bottom:1px solid blue} //设置下边框

注：表格显示双线条边框，因为table、th、td都有独立的边框。

border-style: 定义边框的样式。值有none（无边框）、hidden（无边框，不能用于表）、dotted（点状边框）、solid（实线）、groove（3d凹槽边框）等。

五种用法：

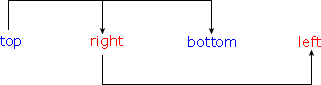
p{border-style: solid;} //设置四边为相同样式

p{border-style: dotted solid double dashed;} //设置四边为不同样式（上右下左）

p{border-style: dotted solid double;} //上边框为dotted,左右边框为solid，下边框为double

p{border-style:dotted solid;} //上下为dotted，左右为solid

注：规则：缺少左，右补；缺少下，上补；缺少右，上补；不能缺少上！



p{

border-style:solid;

border-top-style:none;

border-right-style:none;

border-bottom-style:none;

border-left-style:none;

} //后面属性会覆盖前面属性。注：当边框样式为none时，宽度一定为0

border-color: 边框颜色（必须先设置样式border-style）。用法如boder-style。特别值：transparent（透明边框，有宽度）。

border-width: 边框宽度（必须先设置样式border-style）。用法如border-style。特别值：thin（细）、medium（默认，中等）、thick（粗）。

border-collapse: 是否将表格边框折叠为单一边框。值为separate（默认值）、collapse（折叠）。

用法：table{border-collapse：collapse;}

text-align: 设置表格中文本的水平对齐方式。（见7.文本）

padding：内边距，即表格中内容与边框的距离。

border-spacing: 设置分隔单元格边框的距离。值为\*px,\*px。第一个是水平间隔，第二个是垂直间隔。若只有\*px,则水平垂直都定义了。

caption-side: 设置表格标题的位置。值为bottom、top。

empty-cells: 设置是否显示表格中的空单元格。值为show或hide。

table-layout: 设置显示单元，行和列的算法。值为automatic(默认，由内容决定)、fixed（列宽由表格宽度和列宽度决定）

border-radius（CSS3）: 添加圆角。值为圆角程度。可为长度或百分比。

用法：{border-radius:25px}

box-shadow(CSS3): 向框添加阴影。值为向左偏移程度，向下偏移距离，模糊距离（可选），阴影大小（可选），阴影颜色（可选）和将外部阴影改为内部阴影（inset，可选）。

用法示例：{box-shadow:10px 15px 5px black}

box-image(CSS3): 用图片创建边框。包含五个值：图片存放路径，向内偏移距离，图片边框的宽度，边框图像区域超出边框的长度，图像边框处理方式（平铺repeated、铺满rounded、拉伸stretched）。

用法示例：url(/i/border\_image\_button.png) 0 14 0 14 stretch;

注：未理解

## 轮廓

outline: 设置所有轮廓。顺序是color->style->width。若未设置则为默认值。样式必须有。

示例：

p{

border:red solid thin;

outline:blue dotted thick

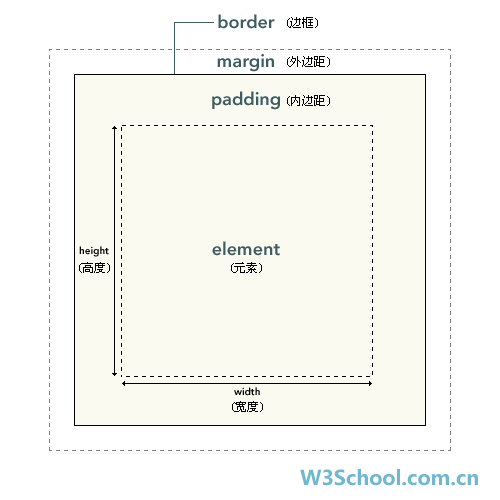
}

outline-color: 设置轮廓颜色（要先设置样式）。

outline-style: 设置轮廓的样式。

outline-width: 设置轮廓的宽度（要先设置样式）。

## CSS盒模型



可以使用通用选择器对所有元素进行设置。

用法：

\*{

margin：0；

padding：0；

}

注1：CSS中width和height指的是内容区域的宽度和高度。增加padding，margin，border不会影响width和height，但是会增加元素框总尺寸。

注2：内边距、边框、外边距可以应用于一个元素的所有边或单独的边。

注3：外边距margin可以为负值，很多情况都要用到负值的外边距（生成边栏）。负值的外边距就是将框架整体缩小，某条边的负值外边距则在那个方向上缩小。

注4：添加背景，背景应用于内容，内边距和边框组成的区域。

注3：margin-right:-150px



padding: 内边距，即内容与边框的距离。值接受长度值和百分比值，不允许负值。百分比值是相对其父元素的width，父元素的width改变，使用百分比值的padding也会改变。

用法：

h1 {padding：10px;}

h1{padding：10px 0.25em 2ex 20%} //上 右 下 左

h1{

padding-top:10px;

padding-ringht:0.25em;

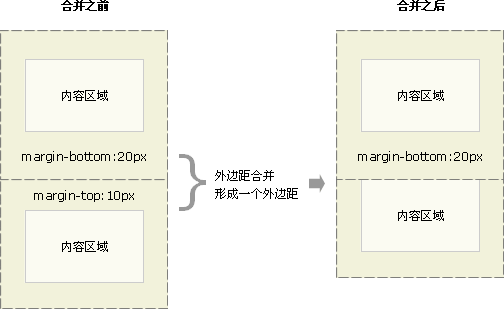
padding-bottom:2ex;

padding-left:20%;

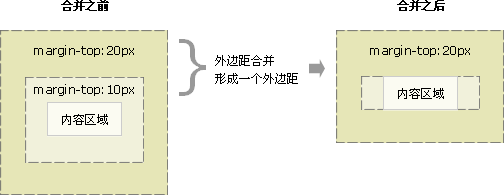
}

border： 边框。边框绘制在”元素背景之上”。（具体见11.表格）

外边距合并：当两个垂直外边距相遇时，将形成一个外边距。合并后的高度等于两个外边距中高度较大的值。特殊：空元素（有外边距但是没有内容和边框）可以重复合并，类似俄罗斯方块。



特殊的垂直外边距：当一个元素包含在另一个元素中时，上下外边距也会合并。



## 定位

一切皆为框，包括块级元素和内联元素。

display: 设置框的类型。用于定义布局元素。关键值有none（元素不被显示）、block（显示为块级元素，带有换行符）、inline（默认，显示为内联元素，没有换行符）等。可以实现块级元素和内联元素的性质的转换（是否换行）。

CSS定位机制：普通流，浮动和绝对定位。默认所有框在普通流定位。

三种方式使一个元素脱离文档流：浮动，绝对定位（absolute）和固定定位（fixed）

position: 选择定位类型。只有static、relative、absolute、fixed。

static：正常生成，文档流内。

relative（相对定位）: 框偏移，形状不变。相对于它的起点移动。

用法：

p{

position:relative;

left:-20px //向左移20px

}

absolute（绝对定位）：从文档流删除。相对于最近的已定位父元素或以上元素，如果元素没有已定位的祖先元素，那么它的位置相对于最初的包含块。

用法：

p{

position:absolute;

left:100px;

top:150px;

} //距离左侧100px,距离顶部150px

<body>

<p>绝对定位</p>

</body>

fixed（固定定位）：表现类似absolute，相对于整个视窗。

overflow属性: 当内容溢出元素内容区域时。值有visible(默认，多余内容在元素之外)、hidden（多余内容不可见）、scroll（元素以滚动条呈现）、auto（若有溢出，则以滚动条呈现，否则正常）。

clip: 设置元素形状，将元素裁剪入一个矩形再显示。

用法：

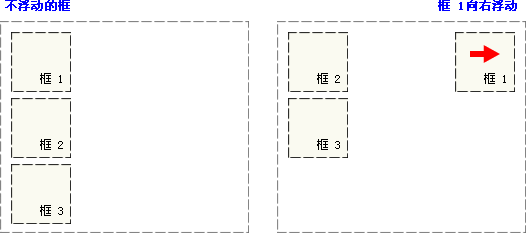
clip:rect(\*px,\*px,\*px,\*px) //值含义：从上裁剪图片长度；裁剪后图片宽度（右边被裁剪）；裁剪后图片高度（下边被裁剪）；从左边裁剪图片长度。

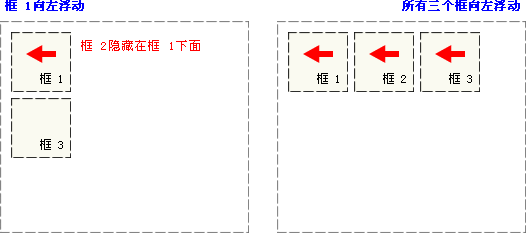
vertical-align: 设置元素垂直对齐方式。关键值有text-top（顶端与父元素顶端对齐）、middle（放在父元素中部）、text-bottom（底部与父元素底部对齐）等。注：vertical-align：middle是子元素相对父元素垂直居中，text-align：center是内容相对框架水平居中，有差别。

z-index: 设置元素堆叠顺序，高堆叠顺序总是在低堆叠顺序前面。值为数值，可以为负值。默认为同一框架下先来在前。

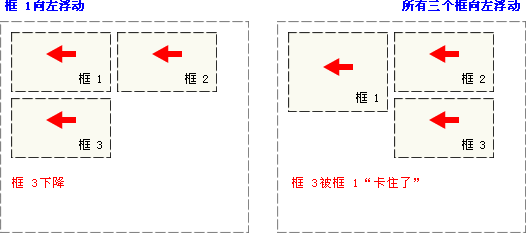
float: 实现元素的浮动。多用于文本围绕图像。

浮动原理：





大框横向放不下三个小框：



clear: 表示框的哪些边附加不应该出现浮动元素。值有left、right、both、none。

## 伪类，伪元素

向某些选择器添加CSS效果。

伪类用法：选择器名：伪类名{}

示例1：通常伪类。值有:active、:focus(有键盘输入焦点)、:hover、:link、:visited

a:hover{} //鼠标悬停态的CSS

a.red:hover{} //类名为red的a标签的鼠标悬停态的CSS

示例2：first-child伪类

p:first-child{} //p标签成为某个父元素的第一个子元素时的CSS

p>i:first-child{} //所有p标签中的第一个i标签内的元素的CSS

p:first-child i{} //所有元素的第一个子元素为p标签内的i标签的元素的CSS。

示例3：lang伪类

p:lang(no){} //p标签中有lang属性且值”|=no”的元素的CSS

伪元素用法：选择器名：伪元素名{}

:first-line伪元素，用于对文本的首行设置CSS。注：只适用块级元素

用法：p:first-line{} //所有p标签的第一行文本的CSS

:first-letter伪元素，用于向文本的首字母设置特殊格式。用法同:first-line。注：只适用块级元素

:before伪元素，用于在元素内容前插入新内容。

用法：h1:before{content:url(1.jpg)} //插入图片

:after伪元素，用于在元素内容后插入新内容。

用法：h1:after{content:url(1.jpg)} //插入图片

伪元素可以结合各种选择器使用。例如：p.article:first-letter{}

## 水平对齐

对齐块元素：

文本对齐：text-align:center

方法1：使用margin属性：左右外边距设置为auto:。可居中对齐。注：宽度不能为100%,不然就没有必要居中了

.center{

width:70%;

margin-left:auto;

margin-right:auto;

}

方法2：使用绝对定位。可左对齐，右对齐，居中对齐。

方法3：使用float属性。可左对齐，右对齐。

.right{

float:right

width:300px

}

## CSS尺寸

height： 元素的高度

width： 元素的宽度

line-height：行高

max-height: 元素的最大高度

max-width: 元素的最大宽度

min-height: 元素的最小高度

min-width: 元素的最小宽度

值可以用像素值（px）,百分比，em

## 媒介类型

@media：在样式表中使用不同的样式规则来针对不同的媒介（电脑，手机等）

方法一：直接使用值。如screen（电脑屏幕）、handheld（手持设备）等

用法：

<style>

@media screen

{

p{}

}

@media handle

{

p{}

}

</style>

方法二：添加限制条件

<style>

@media screen and (min-width: 1000px) {

//最小宽度大于1000px时显示

.nav{}

}

@media screen and (min-width: 640px) and (max-width: 1000px) {

//最小宽度小于1000px,大于640px时显示

.nav{}

}

@media screen and (max-width: 640px) {

//最小宽度小于640px时显示

.nav{}

}

</style>

## CSS3弹性盒子

新的布局模式，用于页面需要适应不同的屏幕大小及设备类型时确保元素拥有恰当的布局方式。

flex/inlineflex: 通过display设置为flex或inline-flex定义为弹性容器

弹性容器包含一个或多个弹性子元素

弹性子元素通常在弹性盒子内一行显示，默认每个容器只有一行

flex-direction: 顺序指定弹性子元素在父容器中的位置

用法：

flex-direction：row|row-reverse|column|column-reverse

row:横向从左到右（默认）；row-reverse:横向从右到左；column:纵向从上到下；column-reverse:纵向从下到上

justify-content: 内容沿主轴线对齐（水平对齐）。

align-items: 设置或检索弹性盒子在纵轴方向上的对齐方式（垂直对齐）

flex-wrap: 指定弹性盒子的子元素换行方式。有单行，多行或反转多行。

align-content: 设置各个行对齐

order: 根据order大小排列顺序，小在前，大在后。

margin-right:auto: 将剩余空间放置在该元素的右侧。

margin:auto: 在弹性盒子中完美居中。

align-self: 设置弹性元素自身在纵轴方向上的对齐方式。

## 水平垂直居中

1. 水平居中margin法

三个条件：

1. 元素定宽
2. 块级元素或设置display:block
3. margin-left和margin-right为auto

示例：

.son

margin:0 auto;

}

1. 水平居中定位法

三个条件

* 1. 元素定宽
  2. 元素绝对定位，并设置left:50%
  3. 元素左边距margin-left为宽度的一半且为负的

示例：

.father{

position:relative

}

.son{

width:100px;

position:absolute;

left:50%;

margin-left:-50px;

}

元素宽度不固定则用CSS3中的transform。

1. 单行文字水平居中

text-align:center

多行文字看作块级元素

1. 垂直居中定位法

与水平居中类似

.father{

position:relative;

}

.son{

width:100px;

position:absolute;

top:50%;

margin-top:-50px;

}

1. 垂直居中定位法简化版

.father{

position:relative;

}

.son{

width:100px;

position:absolute;

top:calc(50% -50px);

}

1. 单行文本垂直居中

两个条件

* 1. 元素内容单行，高度不变
  2. line-height和height相同

示例：

.father{

height:300px;

}

.son{

line-height:300px;

}

1. 水平垂直居中弹性盒子法
   1. 块级元素

.father{

display:flex

}

.son{

margin:auto;

}

* 1. 文本

son{

display:flex;

align-items:center;

justify-content:center;

margin:auto;

}

1. 水平垂直居中定位法

.father{

position:relative

}

.son{

position:absolute;

top:0;

left:0;

right:0;

bottom:0;

margin:auto;

}

## nth-child(CSS3)

:nth-child(n) 匹配属于其父元素的第n个子元素

n可以为数字，关键词，公式

特殊：当n为odd时匹配下标为奇数的子元素；当n为even时匹配下标为偶数的子元素。

## nth-of-type(CSS3)

span:nth-of-type(n): 匹配属于其父元素的第n个span类型的子元素。

span.one:nth-of-type(n): 不会匹配属于其父元素的第n个span类型的而且带有.one CSS类的子元素

## 

# 第三章 JavaScript

## 1. 输出

JavaScript能够以不同的方案”显示”数据.

innerHTML: 写入HTML元素。

访问HTML元素，可用document.getElementById(id)的方法。

用法示例：document.getElementById(“demo”).innerHTML=5+6;

document.write(): 写入HTML元素（测试用）。

用法：document.write(5+6);

注：当HTML文档完全加载后使用document.write()将删除所有已有的HTML。

window.alert(): 警告框显示数据。一般alert();

用法：alert（5+6）；

console.log(): 控制台显示数据（测试用）。

用法：console.log(5+6);

## JS数据类型

声明数据类型只需要”var”。

数据类型共八种：

原始数据类型七种：Number（数值），String（字符串），Boolean（布尔），Undefined（未被赋值），Null（NULL），BigInt（任意精度的整数），Symbols（符号）

引用类型一种:Object(对象) ，数组和函数都是对象。

JS拥有动态类型，同一个变量可用作不同类型。

用法：

var x; //undefined

var x=1; //数值

var x=”bill” //字符串值

var x=7+”vovle” //JS会把数值看作字符串，结果为”7vovle”

原始数据类型的复制是相互独立的，复制会自动创建新的内存来保存。区别于对象。

JS对象是name:value对。

用法1：

var per={firstname:”bill”,lastname:”gates”,age:88}

用法2：

var per=new Object();

per.name=”BILL”;

typeof:确定JS变量类型。

用法：typeof “” 注：typeof对数组返回object（对象）。

## 对象

对象方法：对象中的函数。

用法示例1：

var per={

firstname:”bill”,

lastname:”na”,

id:78,

full:function(){

return this.firstname+this.lastname; //this指的是person

}

或者

get full(){

return this.firstname+this.lastname;

}

}

访问方式：person.lastname或者person[“lastname”]。

第一种对象函数访问方式：person.full(),没有括号返回函数具体定义内容。

第二种get访问方式：person.full。更加简洁。

注：不要把字符串，数值，布尔值声明为对象，降低效率。new关键词会把变量创建为对象。

this关键词： this引用该函数的”拥有者”。

当复制保存对象的某个变量时，操作的是对象的引用；但为对象添加属性时，操作的是实际的对象。

不是在函数内部用var定义的变量都会被视为全局变量。

## 事件句柄

定义事件的行为。

onchange: HTML元素已发生改变。

onclick: HTML元素被点击

onmouseover: 鼠标悬停在该元素上

onmouseout: 鼠标移开HTML元素

onmousemove： 鼠标被移动

onmouseup: 鼠标按键被送开

onreset： 重置按钮被点击

onresize： 窗口或框架被重新调整大小

onselect： 文本被选中

onsubmit： 确认按钮被点击

onunload： 用户退出页面

onload： 页面或图像完成加载

onkeydown: 用户按下键盘按键

onabort: 图像加载被中断

onblur: 元素失去焦点

ondbclick: HTML元素被双击

onerror： 加载文档或图像时发生错误

onfocus: 元素获得焦点

onkeypress: 用户按下键盘按钮并松开

onkeyup: 用户送开键盘按钮

## 字符串方法

字符串一旦被创建，就无法修改。

length： 长度

indexof():返回字符串中指定文本首次出现的位置。

用法：

var str.indexof(“a”);

var str.indexof(“a”,10) //可指定开始的起始位置，往后找

var str.lastindexof(“a”,10) //可指定的起始位置，往前找

lastindexof(): 返回字串中指定文本最后一次出现的位置

两个方法未找到都返回-1.

search(): 和indexof类似。

search()无法设置第二个位置参数；indexof()无法使用正则表达式

slice(): 提取字符串某个部分并返回被提取的部分。

用法：var str=””; var res=str.slice(7,13);

若值为负，则从结尾计算。

用法：var res.slice(-13,-7)

若省略第二个参数，则之前部分被裁剪。

substring(): 类似slice(),但值不能为负。

若省略第二个参数，则之前部分被裁剪。

substr(): 类似slice(),第二参数为长度。

若省略第二个参数，则之前部分被裁剪。

replace() 用另一个值替换字符串中指定的值

用法：var n=str.replace(“1”,”b”);

注：原字符串不变，会返回新字符串。

注：只替换第一个匹配的。

注：对大小写敏感

toUpperCase(): 把字符串转换为大小写。

用法：var text1.toUpperCase();

concat(): 连接多个字符串

用法：text4=text3.concat(test1,text2)

注：所有方法不会改变原始字符串，只会返回新字符串。

trim(): 删除字符串两端的空白符

用法：var str=” x”;var str.trim();

charAt(): 返回字符串指定位置的字符；

用法：str.charAt(8);

charCodeAt(): 返回字符串指定位置的字符的Unicode编码

用法：str.charCodeAt();

split(): 将字符串转换为数组。划分数组的标准为具体的值。

用法：var txt=”a,b,c” var arr=txt.split(“,”) //以”,”划分字符串，形成abc数组

var arr=txt.split(“|”) //以”|”划分字符串(或其他没有出现的字符)，数组只有一个值就是原字符串。

var arr=txt。split(“”) //””为值时，字符串所有字符划为数组。

## 数字

小数计算不准确，使用乘除法。用法：var x=(0.2\*10+0.1\*10)/10

var x=10;var y=20;var z=”30”;var last=x+y+z //只有加号这样。结果为3030，·从左向右编译。

var x=”100”；var y=”10”;var z=x/y; //结果为10，其他数学运算JS会把字符串转换为数字。

NaN: 某个数不是合法数字。一般当做一个数值。

用法：

var x=NaN；var y=1;var z=x+y; //值为NaN,与NaN数值运算值都为NaN

var x=NaN;var y=”5”;var z=x+y; //值为NaN5

isaNaN(): 确定某个值不是数值。不是是True,是是False。

用法：isNaN(x);

Infinity: 计算超出范围时返回的值（包括除以0）。

用法：if(num==Infinity)

toString(): 进制转换。

用法：num.toString(16) //num转换为16进制

toExponential(): 返回字符串值。先转换为只有各位的数字，再计算精度，值为精度，四舍五入，精度大时往后加0。

用法：

var num=90.656

num.toExponential(2) //最后值为9.07e+1,”+1”表示乘以0个10。

toFixed(): 返回字符串值。值为小数位数字，四舍五入，精度大时往后加0。

用法：

var x=9.656；

x.toFixed(2) //返回9.66

toPrecision(): 返回字符串值。值为长度，四舍五入。

用法：

var x=9.656;

x.toPrecision(2) //返回9.7

Number(): 变量转为数值，无法转则为NaN。原变量为日期时则转为1970年1月1日至日期的毫秒数

用法：var x=”1”;Number(x);

parseInt(): 变量转为整数，无法转则为NaN,只返回首个数字。

用法：var x=”10.33”;parseInt(x); //值为10

parseFloat(): 变量转为浮点数，无法转则为NaN,只返回首个数字。

用法：var x=”10.33”;parseFloat(x); //值为10.33

MIN\_VALUE和MAX\_VALUE返回可能的最小和最大数字。

用法：var x=y.MIN\_VALUE；

POSITIVE\_INFINITY或NEGATIVE\_INFINITY:

溢出时返回POSITIVE\_INFINITYNEGATIVE\_INFINITY

## 数组

数组是特殊类型的对象

可以在数组中保存函数，也可以保存数组

注：数组不支持散列。

push(): 数组末尾添加新元素，返回值为新数组长度。

pop(): 删除数组最后一个元素，返回值该元素。

toString(): 数组转为字符串，字符串中用逗号分隔。

join(): 类似toString(),分隔号可自定义。

用法：str.join(“|”);

shift(): 删除首个数组元素，并将后面元素前移。返回该元素。

unshift(): 在开头添加新元素，原元素后移。返回新数组长度。

splice():数组指定一段位置添加一系列新值。返回新数组。

用法：

a.splice(2,0,”kad”,”bob”); //第一个参数定义位置，第二个参数定义删除元素数，其余为添加的新元素。

a.splice(0,1); //若无新元素则只删除。

slice(): 裁剪数组。

用法：

a.slice(1,3) //删除1-3的数组元素

a.slice(2) //删除位置为2的元素

foreach(): 为每个数组元素调用一次函数。

注：改变原数组

用法：

arr.foreach(bianli());

function bianli(value){text+=value+” ”;}

map(): 每个数组元素都执行一次函数，以创造新的数组。

注：没有值的数组元素不会执行函数；不会改变原始数组。

示例：

arr2=arr1.map(myfunction);

function myfunction(value,index,array){ //value是项目值，index是项目索引，array是数组本身,没有用到的参数可省略。

return value\*2;

}

filter(): 用通过测试函数的数组元素创建一个新的数组。

示例：

arr2=arr1.filter(myfunction);

function myfunction(value,index,array){

return value>18;

}

reduce(): 在每个数组元素上运行函数，最后生成一个值。

注：从左到右（从右到左是reduceright()）；不改变原始数组。

用法：

var num=arr.reduce(myfucntion);

function myfunction(total,value,index,array){ //total为初始值以及之后的返回值

return total+value;

}

every(): 检测是否数组所有元素通过测试函数，返回true或false。

some(): 检测是否存在数组元素通过检测，返回true或false.

用法：

var result=arr.every(myfunction);

function myfunction(value,index,arr){

return value>18;

}

indexof(): 搜索元素值并返回其首个出现的位置（类似字符串）。

lastindexof(): 类似indexof从末尾开始搜索。

用法：

var num=arr.indexof(“apple”)

find(): 通过测试函数的第一个数组元素的值

findIndex():通过测试函数的第一个数组元素的索引

示例：

var num=arr.find(myfunction);

function myfunction(value,index,arr){

return value>18;

}

## 数组排序

sort(): 以字母顺序排序（默认升序）

reverse(): 逆序数组

sort(function(a,b){return a-b}): 比值函数。若返回负值则a在b之前，返回正值则a在b之后。

Math.max(): 数学方法找最大值。

用法：Math.max.apply(null,数组名)

注：Math.max.apply([1,2,3])等价于Math.max(1,2,3)

## 日期

new Date(年，月，日，小时，分钟，秒，毫秒)，后面可省略

用法：var d=new Date(2018,11,24,10,33,30,10)

缺少学习

## 数学

缺少学习

## Switch

格式：

switch（）{

case n:

代码；

break;

default:

代码；

}

注：多个匹配，匹配第一个。

注：case使用严格匹配（”===”:类型和值都要相同）

## 类型转换

Number()转换为数值，String()转换为字符串，Boolean:转换为布尔值

constructor: 检查变量的构造器函数（array,date之类更加具体的类型）

## 正则表达式

正则表达式构成一种搜索模式，用于文本搜索和文本替换。

语法：/模式/修饰符

修饰符：i(大小写不敏感)、g(全局匹配，不在第一个匹配成功后停止)、m(多行匹配)

一般方法：search（）、match（）、replace（）、test（）

search(): 使用表达式来匹配字符串。返回匹配的位置。

replace(): 返回模式被替换修改后的字符串

match(): 类似search()。返回匹配的元素。

test(): 检验字符串的一部分是否符合规则。返回true或false

用法：

if(模式.test(字符串)){}

模式：

1. 括号用于查找一定范围的字符串：[abc](找括号之间的任何字符)、[a-z][0-9]（找0到9的数字）、(x|y)（查找x或者y）

[^a-z]: 排除a-z。

^a: 行首，从行首开始匹配

a$: 行尾 匹配到行尾结束

1. 元字符查找特殊含义：”\d”(数字)、”\s”(空白字符)、”\b”(匹配单词边界)、\uxxxx(十六进制)、.（点代表任意字符,\.表示点）、\w（英文，数字，下划线）、\D(除了\d)、\S、\W。

\元字符表示查找元字符本身

1. 量词：n{m}(正好出现m次)、n{m,p}(最少m次，最多p次)、n{m,}(最少m次)、n+(匹配任何至少包含一个n的字符串)、n\*(零个或多个)、n?(零个或一个)

示例：

英文，数字 [0-9a-z]

电话号码 [1-9]\d{7}

可能有区号的电话号码 （0\d{2,3}）?[1-9]{8}(0\d{1,5})?

邮箱：^\w+@[a-z0-9]+\.[a-z]+$

## 关键词

this：

在对象的方法中，this 指的是该方法所属的对象。

单独的情况下，this 指的是全局对象。

在函数中，this 指的是函数的所有者（一般是全局对象）。

在函数中，严格模式下，this 是 undefined。

在触发事件中，this 指的是绑定触发事件的元素。

call()和apply()方法可以将修改this的指向

## let,const

let在局部变量中再次声明不会改变原全局变量的内容

示例1： 示例2：

var x=1; var x=7;

{ for(let x=0;x<10;x++){}

let x=2 //此处x=7

}

//此处x=1

const定义的变量不能重新赋值。

在块作用域中const声明重新变量不会改变原变量的值

示例：

var x=1;

{

const x=2

}

//此处x=1

const赋值必须在声明变量同时。

## 浏览器内核

浏览器主要进程：Browser进程（主进程）、Renderer（内核,渲染进程）等。

内核主要线程有5个：

GUI渲染线程

JS引擎线程

事件触发线程

定时器触发线程

异步HTTP请求线程

注：GUI渲染线程和JS引擎线程是互斥的，因为逻辑上这两个线程不应该允许同时执行，即渲染和改变DOM属性不能同时执行，否则会出现数据不一致。在JS引擎线程运行时，GUI渲染线程会挂起，直到JS引擎线程执行完毕。可能会出现JS没有遵循自上而下的执行顺序。

## JSON

数据是名称/值对

数据由逗号分隔

花括号保存对象

方括号保存数组

浏览器和服务器之间传数据必须用文本格式，JSON就是文本格式且便于转换。

JSON值必须是：字符串、数字、JSON对象、数组、布尔、NULL

字符串必须用双引号：{“name”:”json”}

数字必须是整数或浮点数

JSON类似JS对象，但键必须用双引号：{“name”:”json”}。

parse: JSON转化为JavaScript 用法：JSON.parse(\*)

stringify: JavaScript转化为JSON字符串？ 用法：JSON.stringify()

JSONP解决跨域传送JSON数据问题。

允许从另一个域请求外部脚本->script标签代替XMLHttpRequest对象

最简单的例子：

调外域文件的函数来引入外域文件的内容

<script>

function myFunc(myObj) {

document.getElementById("demo").innerHTML = myObj.name;

} //外域文件的函数

</script>

<script src="/demo/demo\_php\_jsonp.php"></script> //外域文件的地址

## 回调函数

函数a有一个参数，这个参数是个函数b，当函数a执行完以后执行函数b。那么这个过程就叫回调。

用法：

a(b()){}

a(function(){}) 常见

回调函数是闭包

注意对象的函数成为回调函数时会出现的问题

## 闭包

函数可以使用此函数外定义的变量，例如：函数可以使用全局变量；有时可以使用其他函数的变量。

常用方式是出现在函数内再创建一个函数时候。

## 浏览器兼容

HTML5：

1. 旧的浏览器不支持HTML5新的语义元素，可以用display:block来告知。
2. 条件判断浏览器类型，条件使用某些语句

示例：

<head>

………..

<!--[if It IE 9]>

<script src=””></script>

<![endif]-->

………..

</head>

## 创建对象的几种方法

1. 创建实例，再添加属性和方法。

示例：

var person=new Object();

person.name=”Brad”;

person.age=29;

person.myfunction=function(){alert(this.name)}

1. 字面量创建（直接创建）

示例：

var person={

name=”brad“,

age=29,

myfunction:function(){

alert(this.name);

}

};

1. 工厂模式

function person(name.age){

var o=new Object();

o.name=name;

o.age=age;

o.myfunction=function(){

alert(o.name);

}

return o;

}

var newperson=person(“brad”,29); //工厂模式

var secondperson=person(“hell”,30);

var newperson=new person(“brad”,29) //寄生构造函数模式,不建议使用

1. 构造函数（简化版工厂模式）

function Person(name,age){ //构造函数名应该以大写开头

this.name=name;

this.age=age;

this.myfunction=function(){

alert(this.name);

}

}

var newperson=new person(“brad”,29);

1. 原型函数

function Person(){

person.prototype.name=”brad”;

person.prototype.age=29;

person.prototype.myfunction=function(){

alert(this.name);

}

}

var newperson=new Person()

每创建一个构造函数都会随之创建一个原型对象，构造函数和原型对象都有指针互指。对象实例和原型有指针互指。有些通用的属性和方法就可以放到原型中方便调用。

简化版：

function Person(){

Person.prototype={ //放在函数内和函数外都可以

constructor:Person,

name:”brad”,

age:29,

myfunction:function(){

alert(this.name);

}

}

}

注：已经创建实例的情况下重写原型对象（字面量方法）会切断现有原型和之前存在的对象实例之间的联系。

1. 原型模式+构造函数

原型函数中存放公共的用法

构造函数使用多变的变量

fucntion Person(name,age){

this.name=name;

this.age=age;

}

Person.prototype={

constructor:Person,

myfunction:function(){

alert(this.name);

}

}

var newperson=Person(“brad”,29);

1. 如果添加一个判断条件：如果myfunction函数不存在的情况下则添加到原型中。称之为动态原型模式。

fucntion Person(name,age){

this.name=name;

this.age=age;

if(typeof this.myfunction != ”function”){

Person.prototype.myfunction=function(){

alert(this.name);

};

}

}

1. 稳妥构造函数

不允许外部访问某些变量，只能通过方法获得。（类似私有成员）

function Person(name,age){

var o=new Object();

var address=”china”;

o.sayName=fucntion(){

alert(name);

}

o.sayAddress=function(){

alert(address);

}

return o;

}

var me=Person(“brad”,29);

me.sayName();

## JS继承

1. 原型继承

继承通过原型链实现，即用原型继承原型，一个原型包含一个指向另一个原型的指针。

示例：

function Person(){

this.a=true;

}

Person.prototype.getA=function(){

return this.a;

}

function Man(){

this.b=false;

}

Man.prototype=new Person(); //继承

Man.prototype.getB=function(){

return this.b

}

var me=new Man();

alert(me.getA());

1. 借用构造函数

function Person(){

this.a=[1,2,3];

}

function Man(){

Person.call(this); //或apply()，区别是apply第二个参数是参数数组，call是一系列参数

}

var me=new Man();

me.a.push(4);

alert(me.a); //1,2,3,4

var me2=new Man();

alert(me2.a) //1,2,3

1. 组合继承（原型链+借用构造函数）

用原型链实现对原型属性和方法的继承，用构造函数实现对实例属性的继承。

原型链继承不会变化的公共的方法。

构造函数继承多变的变量。

function Person(name){

this.name=name;

this.color=[1,2,3];

}

Person.prototype.sayName=function(){

alert(this.name);

}

function Man(name,age){

Person.call(this,name) //借用构造函数传参

this.age=age;

}

Man.prototype=new Person();

Man.prototype.constructor=Man();

Man.prototype.sayAge=function(){

alert(this.age);

}

var me=new Man(“brad”,29);

me.color.push(4);

alert(me.color); //1,2,3,4

me.sayName(); //brad

me.sayAge(); //29

var me2=new Man(“hell”,30);

alert(me2.color); //1,2,3

me2.sayName(); //hell

me2.sayAge(); //30

## websocket

通信只能由客户端发起。若要持续检测服务端状态变化，则要进行轮询。

而websocket是客户端可以主动传信息给服务端；服务端可以主动发信息给客户端。

没有同源限制，可以跨域。

都用TCP，都要三次握手。

Http长连接，指的是传递一次大量数据保持长时间连接。

且是无状态的连接，每次都要重新传输身份信息。

Websocket持续连接，指的是除非主动关闭，否则都处于连接状态。

是服务器主动传送消息。

## 事件机制

假设一个div中有一个p，且两个元素都有click处理函数，那么哪个函数会首先被触发呢？

涉及两种截然相反的事件流：事件冒泡和事件捕获

事件冒泡从内向外，事件捕获从外向内

DOM2级事件同时支持了两种事件流，供开发者自行选择。

从兼容性来讲，使用事件冒泡

DOM事件流有三个阶段：事件捕获阶段，处于目标阶段，事件冒泡阶段

事件捕获阶段：document->html->body

处于目标阶段：body->div,事件处理

冒泡阶段：div->body->html->document

1. HTML事件处理（应该摒弃）

用on-event方式处理事件

<p onclick=””></p>

1. DOM0级事件处理

通过JS进行事件处理

var btn=document.ElementById(“myBtn”);

btn.onclick=function(){

alert(this.id);

}

1. DOM2级事件处理

指定事件处理程序：addEventListener()

删除事件处理程序：removeEventListener()

3个参数：处理事件名，事件处理函数，布尔值

布尔值为true，在事件捕获阶段调用事件处理函数。

布尔值为false,在事件冒泡阶段调用事件处理函数。

注：一般设置为false

示例：

var btn=document.getElementById(“myBtn”);

var handler=function(){

alert(this.id);

};

btn.addEventListener(“click”,handler,false);

//可以添加多个事件处理程序，按顺序触发。

var handler2=function(){

alert(“world”);

};

btn.addEventListener(“click”,handler2,false);

btn.removeEventListener(“click”,handler,false);

btn.removeEventListener(“click”,handler2,false);

## 性能优化

1. 避免全局查找。将在一个函数中会用到多次的全局对象存储为局部变量。
2. 避免with语句
3. 避免不必要的属性查找。多次用到同一个对象属性，应该将其存放到一个局部变量中。
4. 优化循环：

从最大值往最小值迭代；

简化结束条件；

简化循环体；

若只有一个值循环，则使用类似do-while的后测试循环，避免使用for和while前测试循环；

循环不大则不用循环（循环不确定可以用Duff装置）；

避免双重解释（避免使用eval();Function构造函数简化；setTimeout传一个函数参数）；

复杂的if-else换成switch

考虑使用位运算符

1. 减少http请求次数

## 状态码

100 继续

200 正常返回信息

201 请求成功且服务器创建了新的资源

202服务器已接受请求，但尚未处理

301请求的网页已永久移到新位置

302临时性重定向

303临时性重定向且用get请求新的URI

304自从上次请求后，请求的网页未修改过

401请求未授权

403禁止访问

404找不到与URI匹配的资源

500 最常见的服务器端错误

## JS进阶

DOM：文档对象模型

BOM：浏览器对象模型

DOM把整个页面映射为一个多层次节点结构，HTML页面中的每个组成部分都是某种类型的节点，节点又包含不同类型的数据。

<script>标签写在<body>最后。如果写在之前，在js文件未加载完成之前，页面不会渲染（<body>呈现内容）。

函数在被调用的时候，可以传入任意数量任意类型参数，而不用在意定义时候的参数格式（伪）。因为定义时候参数最终是被设置为一个数组。函数体内可以用arguments来访问参数数组。

## async defer

涉及HTML文件引用JS文件

async和defer 让HTML和JS文件的加载异步

但defer是明确JS加载在HTML加载之后，一般用defer

# 第四章 计算机网络

## DNS

DNS用于把主机名转换为IP地址

DNS缓存指的是之前查询过的DNS信息会在本地DNS服务器端记录，下一次查询就可直接获得。

DNS服务器端从近到远有：本地DNS服务器->根DNS服务器->TLD DNS服务器->权威DNS服务器

## 浏览器输入一个URL之后会发生什么

域名解析->浏览器通过TCP/IP协议发送HTTP请求给服务器->将网页以数据包的形式传给浏览器->浏览器渲染网页/断开TCP连接

域名解析首先在DNS缓存查找，若有记录则直接用。没有则继续向更深层次的服务器端请求。

过程：

浏览器从URL中抽出主机名，并将主机名发给DNS应用客户端

DNS客户端向DNS服务器端发送一个包含主机名的请求

DNS客户端收到一份回答报文，包含对应该主机名的IP地址

HTTP使用TCP作为它的支撑运输协议。浏览器首先发起一个与服务器端的TCP连接。连接一旦建立，浏览器和服务器可以通过套接字访问TCP,通过套接字发送HTTP请求报文和HTTP响应报文（浏览器发送请求报文，接收响应报文；服务器发送响应报文，接受请求报文）。

浏览器渲染网页和断开TCP连接没有关系。断开连接一般是页面关闭或过了指定时间。

## GET和POST

定义来说：

GET:从服务器请求数据

POST:向服务器提交数据

面试说法：

1：GET在地址栏传输数据，POST在表单传输数据；

2：GET传输数据量小，POST传输数据量大；

3：GET传输数据可见，因此不安全；POST传输数据不可见，因此安全；

4：GET请求可被缓存；POST请求不会被缓存

一般说法：



# AngularJS

## 重点

controller   directive   factory   filter  constant

## angular.js导入

<script scr=”https://cdn.staticfile.org/angular.js/1.4.6/angular.min.js</script>”

注：其余脚本放在<body>元素底部，提高网页加载速度。

## 基本指令

通过内置指令来为应用添加功能

允许自定义指令

ng-app: 定义一个AngularJS应用程序

ng-model: 把该元素值绑定到应用程序。用于input,select,textarea

-这是一个双向绑定，在页面修改输入栏值得同时，显示部分也会同步改变;控制器中的值会映射到页面

-可以应用数据提供状态值:

valid(如果输入值是合法的则为true),

dirty（如果值改变则为true）,

touched（如果失去焦点则为true）,

error(如果输入值检查不通过则为true).

empty(输入值为空则为true),

no-empty,还没搞懂

untouched,

invalid,

pending(任何为满足$asyncValidators的情况),

prisitine(为初始状态则为true)。

-根据状态可以提供对应的CSS类，例如input.ng-valid{background-color:lightblue}等

示例1：

<div ng-app=””>

<input type=”text” ng-model=”name”>

<h1>{{name}}</h1>

</div>

注：若没有ng-app，则直接显示{{name}}。

示例2：

<form ng-app=”” name=”myForm” ng-init=”myText=’test@runoob.com’”>

<input type=”email” name=”myAddress” ng-model=”myText” required>

<p> Valid:{{myForm.myAddress.$valid}}</p>

</form>

ng-bind: 把应用程序数据绑定到HTML视图

ng-init: 初始化AngularJS应用程序变量。注：不常用

示例：

<div ng-app=”” ng-init=”firstname=’brad’”>

<p ng-bind=”firstname”></p>

注：HTML5允许的扩展属性以data-开头，AngularJS属性以ng-开头，综上可用data-ng-让AngularJS属性对HTML5有效。

ng-repeat: 重复一个HTML元素。可用于遍历显示数组元素。$index表示索引（0,1..），可用{{}}获取。

示例:

<div ng-app=”” ng-init=”name=[‘a’,’b’,’c’]”>

<ul>

<li ng-repeat=”x in names”>{{x}}</li>

</ul>

</div>

用directive自定义指令

示例：

<div ng-app=”myApp”>

<runoob-directive></runoob-directive> //调用指令

</div>

<script>

var app=angular.module(“myApp”,[]);

app.directive(“runoobDirective”,function(){ //定义指令

return{

template:”<h1>somthing</h1>”

};

});

</script>

可以通过元素名/属性/类名/注释来调用指令：

<runoob-directive></runoob-directive>

<div runoob-directive></div>

<div class=”runoob-directive”></div>

<!-- directive:runoob-directive>

限制使用：添加restrict属性，来限制自定义指令只能通过特定的方式来调用。

E：作为元素名

A：作为属性

C：作为类名

M：作为注释

EA：默认，可通过元素名和属性调用

示例：

var app=angular.module(“myApp”,[]);

app.directive(“runoobDirective”,function(){

return{

restrict:”A”,

template:”<h1>自定义指令</h1>”

};

});

## 表达式

表达式写在双大括号内：{{表达式}}

表达式可以包含文字、运算符和变量。

表达式不支持条件判断，循环和异常

表达式支持过滤器(参考过滤器)

示例：

<p>{{5+5}}</p> //10

<p>{{“ni”+5}}</p> //”ni5”

注：<p>{{a\*b}}</p> 和 <p ng-bind=”a\*b”></p>等价

## 应用

模块定义了AngularJS应用

控制器控制AngularJS应用

ng-app指明应用；ng-controller指明控制器

示例：

<div ng-app=”myApp” ng-controller=”myCtrl”>

<input type=”text” ng-model=”firstName”><br/>

<input type=”text” ng-model=”lastName”><br/>

{{firstName+” ”+lastname}}

</div>

<script>

var app=angular.module(‘myApp’,[])；

app.controller(‘myCtrl’,function($scope){

$scope.firstName=”brad”;

$scope.lastName=”doe”;

})；

</script>

## 数字，字符串，数组，对象

类似JavaScript

## 作用域

在AngularJS创建控制器时，可以将$scope对象当作一个参数传递。

AngularJS应用由View(视图，即HTML)、Model(模型，即视图中可使用的数据)、Controller(控制器，即JavaScript函数)组成。

$rootScope: 根作用域（相当于一个ng-app全局变量）

scope具体和控制器的关系有待研究！

## 控制器

控制器控制应用程序的数据

控制器是JavaScript对象，由对象构造函数创建

ng-controller指令定义了控制器

控制器可以带变量和方法

## 过滤器

过滤器用于转换数据，可用于表达式（{{}}）和指令中

格式： {{数据|过滤器}}

currency: 格式化数字为货币格式

filter: 从数组项中选择一个子集

lowercase: 格式化字符串为小写

orderBy: 根据某个表达式排列数组

uppercase: 格式化字符串为大写

自定义过滤器：app.filter(“过滤器名”，function(){});涉及注入依赖

## 服务

服务是个对象，AngularJS内建了30多个服务

$location：返回当前页面的URL地址

$http: 向服务器发送请求，应用响应服务器穿送过来的数据

$timeout 对应JS window.setTimeout函数，需学习

$interval 对应JS window.setInterval函数，需学习

创建自定义服务： app.server(‘自定义服务名’,function(){}); 涉及依赖

### $http

格式1：

$http({

method:’GET’, //或者POST

url:’/someUrl’

}).then(function successCallback(response){

请求成功后执行代码

}，function errorCallback(response){

请求失败后执行代码

}

)；

格式2：

$http.get(‘/someUrl’,config).then(function(response){},function(res ponse){})); //默认前面的函数为successCallback,后面的为 errorCallback.

$http.post(‘/someUrl’,data,config).then(function(response){},functi on(response){});

### $timeout

$timeout(回调函数，延迟时间（可选），布尔值（可选）)

延迟时间如果为空，则表示等线程空下来后就执行，比如当页面渲染完成后。

实战：onsearch() 子组件会调用父级的onSearch。但是 双向绑定的属性值 可能还是上上次的（即不是最新的），所以要加个$timeout。

$timeout指令是对window.setTimeout的一次封装，返回一个promise对象。当定义的时间到了，这个promise对象就会被resolve，回调函数就会被执行。

取消一个timeout: 调用 $timeout.cancel(promise)

## 选择框

AngularJS可以使用数组或对象创建一个下拉列表选项

用ng-option创建下拉列表的项或者ng-repeat循环。ng-option更好

数组格式： x for x in sites //前面的x代表显示的选项值

对象格式： x for (x,y) in sites //前面的x可以换成y或者y.key

## 表格

使用ng-repeat

奇数行CSS：table tr:nth-child(odd){}

偶数行CSS：table tr:nth-child(even){}

可以配合过滤器

显示序号：<td>{{$index+1}}</td>

## HTML DOM

ng-disabled：绑定应用程序数据到HTML的disabled属性，true时元素不可用

ng-show: 隐藏或显示一个HTML元素，true时元素可见

ng-hide: 同ng-show，true时元素不可见

## AngularJS事件

ng-click: 点击事件

## AngularJS模块

模块定义了一个应用程序

模块是控制器的容器

angular.module（“myApp”,[]）函数： 注册模块

有两个参数，第一个是模块名；第二个是依赖注入列表（空报表时不需要依赖关系）

## 依赖注入

使用服务和在控制器函数中创建函数：服务在整个应用都可以使用，控制器方法函数只能在控制器内使用。

依赖注入是一种设计模式，松耦合

依赖注入有三方，使用方，制造方，传递方。制造方创造函数，传递方传递函数给使用方，使用方使用函数。传递方又称为依赖注入

value: javascript对象，用于向控制器传值（配置阶段）

factory: 函数，用于计算或返回值。在service和controller需要时创建(工厂模式)

Provider：创建一个service,factory(配置阶段)。提供一个factory方法get()

constant: 常量，传递数值（配置阶段）

AngularJS工作流包括配置（config）和运行（run）。通过config方法实现对模块的配置，只有provider和constant能够注入config中。

## 路由

用于允许我们通过不同的URL访问不同的内容。

了解即可，这种方法不会用

首先载入实现了路由的JS文件

然后创建包含ngRoute模块作为主应用模块的依赖模块

使用ngView指令，内容会随路由变化而变

config里定义路由规则

when（path,object）和otherwise(object)定义不同URL时的路由规则

第一个参数是URL或者URL正则规则

第二个参数是路由配置对象

路由配置对象参数：

template: 在ng-view中插入简单的HTML内容

templateURL: 在ng-view中插入HTML模板文件

controller: function,string或数组类型，在当前模板上执行的controller函数，生成新的scope

controllerAs: string,为controller指定别名

redirectTo: 重定向的地址

resolve： 指定当前controller所依赖的其他模块

## console.log和alert测试的区别

console.log能打印任何类型的数据

alert只能打印字符串，若数据类型为对象，则自动调用toString()。

## ng-class

ng-class指令用于给HTML元素动态绑定一个或多个CSS（同一个ng-class可以选择不同的class）。

ng-class指令的值可以是字符串、对象或一个数组

如果是字符串，多个类名使用空格分隔。

如果是对象，需要使用key-value对，key为你想要添加的类名，value是个布尔值，只有值为true才会被添加。

如果是数组，可以由字符串或对象组合而成，数组的元素可以是字符串或对象。

# 第六章 ES6

## let和const

见上

## 变量的解构赋值

允许按照一定的模式，从数组和对象中提取值，对变量进行赋值，这被称为解构。

一种“模式匹配”

* 1. 数组

之前赋值：let a=1,b=2

es6: let[a,b]=[1,2] 一个数组包含两个变量

允许嵌套： let [a,[b,c]]=[1,[2,3]]

let [ , , a]=[1,2,3] <=> a=3

let [a,b]=[1,2,3] <=>a=1,b=2

只要符合Iterator接口，都可以采用数组的形式解构

function test(){

let a=0;

let b=1;

while(true){

yield a;

[a,b] = [b, a+b]

}

}

let[a,b,c,d,e,f] = test() <=> f=5

解构赋值允许指定默认值，有效条件：存在一个数组成员严格等于undefined时：

let [a=0] = [] ;

let [a=0]=[1] <=> a=0

let [x=f()] = [1] ; f(){return 2} ; <=> x=1 默认值是表达时，有赋值优先赋值（惰性）

* 1. 对象形式

let {a,b} = {a:0 , b:1} ;

右边对象没有左边对象的变量名，则不会赋值

现有对象的方法赋给某个对象

let {a,b,c}=Math;

左右变量名不同又要赋值时：

let {a:b} = {a:1} <=> b=1 a是模式，b才是变量

let c={a:’aa’,b:’bb’}

let {a:x,b:y} = c <=> x=’aa’,y=’bb’

嵌套：

let obj = {

p:[‘0’,{y:’1’}]

}

let { p:[x,{y}] } = obj;

let {p, p:[x,{y}] } = obj; //p也被赋值

对象的解构可以取到继承的值

也可以指定默认值，有效条件为对象的属性值严格等于undefined

已经声明的变量用于解构赋值时，需要整个赋值语句加括号，防止编译器解析错误。

let x;

({x} = {x: 1});

数组本质是对象，可以进行对象形式的解构

let arr= [1, 2, 3]; <=> first=1,last=3

let {0: first,[arr.length-1]:last} = arr; //方括号属于“属性名表达式” 详见《对象的扩展》

* 1. 字符串

字符串转换成了数组。

除了一般方法，用length：

let { length: len} = ‘hello’ <=> len=5

* 1. 布尔值\数值

右边为数值或布尔值时会转换成对象

let {toString: s} = 123

右边值为undefined或null时会报错

* 1. 函数参数

function move({x = 0, y = 0}={}){return [x,y]};

move({x:3,y:8}): //[3,8]

move({x:3}) //[3,0]

注意圆括号的使用，只能用在左边的非模式部分

* 1. 用途
     1. 交换变量的值 [x, y] = [y, x]
     2. 若从函数返回多个值，便于取值
     3. 函数参数的定义
     4. 提取JSON数据
     5. 函数参数的默认值
     6. 遍历Map解构

## 字符串的扩展

* 1. 字符的Unicode表示法

\uxxxx

xxxx表示字符的Unicode码点

范围：\u0000~\uFFFF 超出则用两个：\uxxxx\uxxxx（双字节）或者值用大括号括起来\u{xxxxx}

* 1. 遍历器

优化了字符串的for循环遍历，变为for..of遍历

使其可以识别大于0xFFFF的码点

* 1. 允许直接输入字符以及其转义形式
  2. JSON.stringify() 的改造
  3. 模板字符串

原本字符串输出遇到变量要用‘+’号:”xx”+a.x+”yy”。现在可以用模板字符串：`xx${a.x}yy` 注意是反引号，不是传统引号

可以当作一般字符串、可以定义多行字符串、可以在字符串中嵌入变量

多行字符串中的空格和换行都会保留，可以用 `字符串`.trim() 去除换行

嵌入变量写在${ }中，可放入字符串，表达式，运算，引用对象属性，调用函数

模板字符串可嵌套

<%...%>

* 1. 标签模板

alert`123` <=> alert(123)

是函数调用的一种特殊形式

多用于过滤字符串，防止用户输入恶意内容；多语言转换

函数参数也可以用模板字符串：func`a ${b} ${c} d`

<=>func([‘a ’, ’’, ‘ d’], b, c)

第一个参数是数组，元素是模板字符串中那些没有变量替换的部分，以及两个变量相邻时会添加’’（空字符）。其余参数是模板字符串变量被替换后的。

## 字符串新增方法

String.fromCodePoint(): 优化fromCharCode(),用于转换Unicode码点为字符，新增可识别大于0xFFFF的码点.

注：fromCodePoint用于String对象，fromCharCode用于字符串的实例对象

String.raw(): （原生的String对象可用）返回一个斜杠都被转义的字符串，多用于模板字符串的处理。

String.raw`hi\\n` <=> ‘hi\\\\n’ (显示的还是hi\\n)

codePointAt(): 优化字符转码点。能够正确处理4个字节存储的字符，返回一个字符的码点。

normalize() :Unicode正规化

includes(): 返回布尔值，表示是否找到了参数字符串

startsWith(): 返回布尔值，表示参数字符是否在原字符串的头部

endsWith(): 返回字符串，表示参数字符串是否在原字符串的尾部

第一个参数是参数字符串，第二个参数（可选）是搜索的起始位置（endwith第二个参数代表前n个字符）。

repeat(): 将原字符串重复n次，返回新字符串。小数取整，0~-1取0，再小就报错；NaN取0；字符串转换为数字，不是数字取0。

字符串补全长度，某个字符串不够指定长度，会在头部或尾部补全：

padStart(): 用于头部补全

padEnd(): 用于尾部补全

两个参数，第一个参数为最大长度，第二个参数（可选）是用于补全的字符串。

若原字符串足够长则无效，返回原字符串。

若用于补全的字符串过长，则会截去多余的补全字符串。

若第二个参数为空，则默认用空格补全。

trimStart(): 消除字符串头部空格（换行符、tab键也可）

trimEnd(): 消除字符串尾部空格（换行符、tab键也可）

返回新字符串

matchAll(): 返回一个正则表达式在当前字符串的所有匹配。

## 正则的扩展

* 1. Regexp: 被优化
  2. 字符串的正则方法优化。
  3. u修饰符 Unicode模式
  4. 新增unicode属性，用于确定是否设置了u修饰符
  5. y修饰符 与g类似，用于全局匹配，但匹配过程必须从剩余的第一个位置就开始。

let s=”aaa\_aa”

let x1=/a+/g

let x2=/a+/y

x1.exec(s) //aaa

x2.exec(s) //aaa

x1.exec(s) //a

x2.exec(s) //因为是\_aa所以不存在即null

y相比g，确保匹配不会出现漏掉的字符或非法字符。

* 1. sticky属性，用于判断是否有y修饰符
  2. flag属性，返回正则表达式的修饰符
  3. dotAll 用于判断是否有s修饰符
  4. 支持后行断言
  5. 新增Unicode类 \p{...}和\P{...}匹配（\P匹配不满足条件的字符） 允许正则表达式匹配符合Unicode某种属性的所有字符。
  6. 具名组匹配

正则表达式使用圆括号进行匹配，使用exec可以将三组匹配结果提取出来，但每组匹配含义不容易看出来。

具名组匹配允许为每一个组匹配指定一个名字，便于阅读代码和引用。

* 1. 解构赋值和替换

和具名组匹配合作，使用解构赋值直接从匹配结果上为变量赋值。

## 数值的扩展

* 1. 提供了二进制和八进制的新写法

二进制：0bxxx 八进制：0oxxx

转换为十进制：Number(‘ ’)

* 1. Number.isFinite() Number.isNaN()

只对数值有效

isFinite(number): 检查一个数值是否为有限的

isNaN(number): 检查一个值是否为NaN

* 1. Number.parseInt() Number.parseFloat()

将全局方法移植到Number对象上。

* 1. Number.isIntrger()

判断数值是否为整数（低精度）

* 1. Number.EPSILON

新增常量 表示1与大于1的最小浮点数之间的差

用于浮点数计算时设置的误差范围

* 1. 安全整数和Number.isSafeInteger()

整数上限：Number.MAX\_SAFE\_INTEGER

整数下限：Number.MIN\_SAFE\_INTEGER

isSafeInteger(): 判断一个整数是否在此范围内（比较上下限）

* 1. Math对象新增方法（只能在Math对象上调用）

Math.trunc( ): 去除一个数的小数部分

Math.sign( ): 判断数的性质。1（正数），-1（负数），0（0），-0（-0），NaN（其他值）

Math.cbrt( ): 计算一个数的立方根

Math.clz32( ): 将参数转为32位无符号整数的形式，返回这个32位值里有多少个前导0。

Math.imul( ，): 返回两个数相乘的结果，结果是一个32位的带符号整数

Math.fround( ): 将一个数转化为32位单精度浮点数形式。主要用于64位双精度浮点数转为32位单精度浮点数。

Math.hypot(... ): 返回所有参数的平方和的平方根

Math.expm1( ): 返回ex-1

Math.log1p( ): 返回log(1+x)。若x小于-1，返回NaN。

Math.log10(): 返回以10为底数的x的对数。

Math.log2(): 返回以2为底数的x的对数。

Math.sinh(x): 返回x的双曲正弦

Math.cosh(x): 返回x的双曲余弦

Math.tanh(x): 返回x的双曲正切

Math.asinh(x): 返回x的反双曲正弦

Math.acosh(x)； 返回x的反双曲余弦

Math.atanh(x): 返回x的反双曲正切

指数运算符：\*\*（右结合）

## 函数的扩展

* 1. 函数参数的默认值

老式：y = y || ‘a’

ES6: funtion test(x,y = ‘a’)

已默认声明的参数变量不能在函数体内 再次声明

每次调用函数都会重新计算默认值

参数默认值应放在末尾

函数的length将返回没有指定默认值的参数个数

如果设置了参数的默认值，函数进行声明初始化时，参数之间会形成一个单独的作用域，直到初始化结束。

示例：

var x = 1;

function test( x, y = x){ alert(y); } //x, y=x形成自己的作用域

test( 2 ); //alert(2)

默认值可以设为undefined表示参数可省略

默认值可以设置为一个报错函数，一旦调用时省略了这个参数就可以报错。

函数参数可以和解构默认值结合使用。

* 1. rest参数

用于获取函数被调用时多设置的参数（代替arguments对象）

形式： ...数组名

必须是最后一个参数；length属性不计算rest参数

* 1. 严格模式

以下情况下函数内部不可以显示设定为严格模式：使用默认值、解构赋值、扩展运算符。

* 1. name属性

函数的name属性，返回该函数的函数名。若被重新赋给一个变量，则返回该变量名。

构造函数name值为anonymous.

bind返回的函数，name属性值会加上bound前缀

* 1. 箭头函数

注意事项：

* + 1. this对象就是定义时所在的对象，而非使用时所在的对象。即this对象的指向是 固定的。（箭头函数没有自己的this，箭头函数的this就是外层代码块的this）
    2. 不可以当作构造函数，即不能用new命令
    3. 不可以使用arguments对象（用rest替代）
    4. 不可以使用yield命令
    5. 箭头函数没有自己的arguments、super、new.target，若定义了则指向外层代码块的同名关键词。
    6. 不应该在定义对象方法时使用箭头函数，因为对象不能构成单独的作用域，箭头函数作用域会指向全局作用域，这是不被允许的。
    7. 不应该在需要动态this时设置箭头函数（触发事件）
    8. 箭头函数在复杂情况下可读性低
  1. 尾调用优化

尾调用：某个函数的最后一步是调用另一个函数

* 1. 尾递归

函数在最后一步调用自身（递归+尾调用）

尾调用在最后一步只能纯粹调用一个函数，不能附带对这个函数的运算

* 1. 允许函数的最后一个参数有尾逗号
  2. Function.prototype.toString()

不再忽略注释和空格

* 1. try...catch

允许catch省略参数

## 数组的扩展

* 1. 扩展运算符

扩展运算符：...

将一个数组转为用逗号分隔的参数序列（参考rest），即数组的展开

主要用于函数调用和定义；替代函数的apply方法；复制数组（浅拷贝）；合并数组（浅拷贝）；结合解构赋值生成新数组；字符串转数组；实现Iterator接口的对象

只有函数调用时，扩展运算符才可以放在圆括号中

扩展运算符应放在参数的最后一位

* 1. Array.from()

将类似数组的对象或可遍历的对象转为真正的数组。

* 1. Array.of

将一组值转换为数组

* 1. copyWithin()

数组.copyWithin(target, start,end):在当前数组内部，将指定位置的成员复制到其他位置（会覆盖，且会修改原数组）。target是替换的初始位置（必选，可为负值即倒数），start是读取数据的初始位置（可选，可为负值即从末尾开始），end是读取数据的停止位置（可选，默认为数组长度，可为负值即从末尾开始）

* 1. find() findIndex()

参考之前内容

* 1. fill()

数组.fill(数值,start,end)：使用给定值填充一个数组。start是填充起始位置（可选），end是填充结束位置（可选）

* 1. entries() keys() values()

用于遍历数组，返回遍历器对象（Iterator）。entries()是对键值对的遍历，keys()是对键名的遍历，values()是对键值的遍历

* 1. includes()

数组.includes(数值，start)：表示某个数组是否包含给定的值

区分Map和Set数据结构的has方法

* 1. flat() flatMap()

数组.flat(数值||Infinity): 将多维数组降维，返回一个新数组。数值是降维的程度（可选。默认一维）。若最终要降成一维数组，则用Infinity。

数组.flatMap(函数): 对原数组的每个成员执行一个函数,然后对返回值组成的数组执行flat(),返回新数组。

* 1. 数组的空位

大多将空位转为undefined

避免出现空位

* 1. sort()的排序稳定性

排序稳定性：排序关键词相同的项目，排序前后的顺序不变

ES6确定sort()的默认排序算法必须稳定。

## 对象的扩展

* 1. 属性的简介表示法

允许在大括号内直接写入变量和函数，作为对象的属性和方法。

示例1：

const foo = ’bar’

const baz = {foo}; <=> const baz={ foo:foo };

示例2：

function f(x,y){ <=> function f(x,y){

return {x,y} return {x:x,y:y}

} }

示例3（方法简写）：

const o={ <=> const o={

method() { method:function(){

return “hello!”; return “hello”;

} }

} }

* 1. 属性名表达式

允许在字面量定义对象时，允许表达式作为键

示例：

let test={

[‘a’+’a’]:123,

[‘a’+1](){ return 1;},

}

属性名表达式和间接表示法不能同时使用

属性名表达式若是一个对象，则会转为字符串‘[object object]’

* 1. 方法的name属性

对象方法也有name属性，返回函数名（方法名）

若方法使用了getter或setter函数，则name属性在方法的get或set属性上。

bind方法创造的函数，name属性在bind()上

构造函数，name属性返回anonymous

如果方法时一个Symbol值，name属性返回Symbol值的描述。

* 1. 属性的可枚举性

for...in不会遍历到toString属性和length属性的enumerable

* 1. 属性的遍历方法
     1. for...in

循环遍历对象自身和继承的可枚举属性

* + 1. Object.keys(obj)

返回一个数组。包含对象自身的所有可枚举属性的键名（不包括Symbol属性）

* + 1. Object.getOwnPropertyNames(obj)

返回一个数组。包含对象自身的所有属性的键名（不包括Symbol属性）

* + 1. Object.getOwnPropertySymbols(obj)

返回一个数组。包含对象自身的所有Symbol属性的键名

* + 1. Reflect.ownKeys(obj)

返回一个数组，包含对象自身的所有键名（先是数值，然后字符串，最后Symbol）。

* 1. super关键词

指向当前对象的原型对象

只能用在对象的简写方法中

* 1. 扩展运算符

同数组的扩展运算符

取出参数对象的所有可遍历属性，拷贝到当前对象

## 对象的新增方法

* 1. Object.is()

Object.is(a,b)

比较两个值是否严格相等（===）

区别是：+0不等于-0；NaN===NaN

* 1. Object.assign()

对象的合并,将源对象所有可枚举属性复制到目标对象

Object.assign(a,b...)

第一个参数是目标对象，后面的参数都是源对象（会被覆盖）

浅拷贝

数组视为对象，键名为下标值

多用于为对象添加属性、添加方法；克隆对象；合并多个对象为属性指定默认值

* 1. Object.getOwnPropertyDescriptors()

返回指定对象所有自身属性的描述对象

* 1. Object.setPrototypeOf() Object.getPrototypeOf()

\_proto\_: 用来读取或设置当前对象的prototype对象。

只有浏览器才必须部署这个属性，一般看作不存在，使用新增的读写操作代替。

setPrototypeOf():写操作

getPrototypeOf():读操作

* 1. Object.keys() Object.values() Object.entries()

keys(): 返回一个数组，是参数对象自身的所有可遍历的属性的键名。

values(): 返回一个数组，是参数对象自身的所有可遍历对象的键值。

entries(): 返回一个数组，成员是参数对象自身的所有可遍历属性的键值对数组

* 1. Object.fromEntries()

将一个键值对数组转为对象。

是entries()的逆操作

## Symbol

Symbol是新的基础数据类型，表示独一无二的值，用于解决可能存在的冲突。

let a = Symbol(‘test’); 如果参数是对象，则调用toString

不能和其他类型的值进行运算，会报错。

可以转为字符串和布尔值

descriotion属性：返回Symbol的描述（参数）

可用于对象的属性命名，保证唯一性。读取时不能用点运算符。对象内部定义时也只能用方括号。

两种能获得Symbol属性的方法：

Object.getOwnPropertySymbols():获取指定对象的所有Symbol属性名，返回一个数组。

Reflect.ownkeys():返回所有类型的键名

在对象中，用Symbol定义一些希望只用于内部（非私有）的方法。

Symbols.for(): 接收一个字符串作为参数，搜索有没有以该参数作为名称的Symbols值。如果有，就返回这个Symbol值，否则新建一个并在全局作用域注册。

Symbol.keyFor(): 返回一个已登记的Symbol类型的key

11个内置的Symbol值：

hasInstance: 指向一个内部方法，用于判断某个对象是否是该对象的实例。

isConcatSpreadable: 等同于一个布尔值。该对象用于concat时是否可以展开。数组默认可以展开，对象默认不展开。

species: 指向一个构造函数，创建衍生对象会用到。

match: 指向一个函数。当执行str.match()时，如果该属性存在则调用并返回该方法的返回值。

replace: 指向一个方法。当该对象被String.prototype.replce方法调用时，会返回该方法的返回值。

search: 类似replace

split: 类似replace

iterator: 指向该对象的默认遍历器方法。

toPrimitive: 指向一个方法。该对象呗转为原始类型的值时，会调用这个方法，返回该对象对应的原始类型值。

toStringTag: 指向一个方法。在该对象上调用Objec.prototype.toString方法时，如果该属性存在，它的返回值会出现在toString返回的字符串之中，表示对象的类型。有对应属性值。

unscopables：指向一个对象。该对象指定了使用with关键词时，哪些属性会被with环境排除。

## Set数据结构

新的数据结构，类似数组，成员的值都是唯一的，没有重复。（重复的会自动删除）

set是一个构造函数，用来生成Set数据结构。

const test = new Set();

Set函数可以接受一个数组，或者具有iterable接口的其他数据结构用于初始化。

去除数组重复成员：[ ...new Set(array)]

去除字符串的重复字符：[ ...new Set(‘abcabc’)].join(‘’)

NaN===NaN,重复时会删除

两个对象总是不相等的

加入值时不会发生类型转换，5和‘5’是不同的值

属性：

Set.prototype.constructor: 构造函数，默认Set函数

Set.prototype.size: 返回Set实例的成员总数

操作数据方法：

Set.prototype.add(): 添加某个值，返回Set结构本身

Set.prototype.delete(): 删除某个值，返回一个布尔值，表示删除是否成功

Set.prototype.has(): 返回一个布尔值，表示该值是否为Set的成员

Set.prototype.clear(): 清除所有成员，无返回值。

Array.from() :将Set结构转为数组

-去除数组重复成员：array.from(new Set(array))

遍历方法：

Set.prototype.keys(): 遍历后返回键名

Set.prototype.values(): 遍历后返回键值

Set.prototype.entries(): 遍历后返回键值对

Set.prototype.forEach()； 每个成员都使用一次回调函数

keys()、values()、entries返回的都是遍历器对象，Set结构键名和键值相同。

可使用for...of遍历Set

扩展运算符...也可以用于Set结构

-去除数组的重复成员：let test = [...new Set(array)]

-数组的map和filter方法可以间接使用于Set

-实现并集、交集、差集

## WeakSet数据结构

不重复的对象的集合 （任何具有Iterable接口的对象）

对象都是弱引用，只有WeakSet引用某个对象，则该对象占用的内存会被回收。

WeakSet适合临时存放一组对象，以及和对象绑定的信息。

方法：

WeakSet.prototype.add(value): 向WeakSet实例添加一个新成员

WeakSet.prototype.delete(value): 清除WeakSet实例的指定对象

WeakSet.prototype.has(value): 返回一个布尔值，表示某个值是否在WeakSet实例之中

没有size属性，无法遍历

适合存储DOM节点

## Map数据结构

类似对象，键值对的集合 各种类型的值（包括对象）都可以当作键。

Map是一个构造函数，可以接受任何具有Iterator接口，且每个成员都是一个双元素的数组的数据结构（Set和Map都可以）作为参数。

对同一个对象的引用、0和-0、NaN，Map结构都将视其为同一个键，其余同样的值视为两个键

属性：

size： 返回Map结构的成员总数

Map.prototype.set（key,value）: 生成设置键名key对应键值为value,如果key已有，则替换键值。（可用链式写法类似变量）

Map.prototype.get(key): 读取key对应的键值，如果找不到返回undefined

Map.prototype.has(key): 返回一个布尔值，表示某个键是否在当前Map对象之中。

Map.prototype.delete(key): 删除某个键，返回布尔值表示是否删除成功

Map.prototype.clear(): clear方法清除所有成员，没有返回值

Map.prototype.keys(): 返回键名的遍历器

Map.prototype.values(): 返回键值的遍历器

Map.prototype.entries(): 返回所有成员的遍历器

Map.prototype.forEach(): 所有成员实施一次回调函数

Map的遍历顺序就是插入顺序

Map结构转为数组： [...map]

-结合数组方法，filter，实现Map的遍历和过滤

数组转Map: new Map(array)

Map转对象：for...of 非字符串的键名会转为字符串的键名

对象转为Map: for...of

Map转为JSON：

键名都是字符串：JSON.stringify(map) 对象JSON

键名有非字符串：JSON.stringify([...map]) 数组JSON

JSON转为Map：

所有键名都是字符串：JSON.parse(json)

JSON是一个数组，每个数组成员本身又有两个成员：一一对应转为Map

## WeakMap

类似Map,但只接受对象作为键名

若没有其他对象引用，键名指向的对象的内存会被自动回收。

可用于DOM元素添加数据

## Proxy

Proxy用于修改某些操作的默认行为，译为代理器

在目标对象之前假设一层拦截，外界对该对象的访问，都必须先通过这层拦截，因此可以对外界的访问进行过滤和改写。

new Proxy(target,handler)

target参数表示所要拦截的目标对象，handler参数也是一个对象，用来定制拦截行为。

拦截操作：

get(target,propKey,receiver): 拦截对象属性的读取

set(target,propKey,value,receiver): 拦截对象属性的设置，返回一个布尔值

has(target,propKey):拦截propKey in proxy的操作，返回一个布尔值

deleteProperty(target,propKey): 拦截delete proxy[propKey]的操作，返回一个布尔值。

ownKeys(target): 拦截循环，返回一个数组。

getOwnPropertyDescriptor(target,propKey): 拦截getOwnPropertyDescriptor,返回属性的描述对象

defineProperty(target,propKey,propDesc):拦截对应方法，返回一个布尔值

preventExtensions(target):拦截对应方法，返回一个布尔值

getPrototypeOf(target): 拦截对应方法，返回一个对象

isExtensible(target): 拦截对应方法，返回一个布尔值

setPrototypeOf(target,proto): 拦截对应方法，返回一个布尔值。若目标对象是函数，那么还有两种额外操作可以拦截。

apply(target,object,args): 拦截Proxy实例作为函数调用的操作。

construct(target,args): 拦截Proxy实例作为构造函数调用的操作。

Proxy代理的情况下，目标对象内部的this关键词会指向Proxy代理，若未指向Proxy，则Proxy失效。

可用于Web服务的客户端

## Reflect

将Object对象的一些明显属于语言内部的方法移到Reflect对象上

修改某些Object方法的返回结果，让其逻辑上变得合理

让Object操作都变成函数行为，取消命令式

Reflect对象的方法与Proxy的对象的方法一一对应，使Proxy对象能够方便调用对应方法

## Promise对象

异步编程的解决方案，优化传统的“回调函数和事件”，避免层层嵌套的回调函数。

三种状态：pending(进行中)、fulfilled(已成功)、rejected(已失败)。但只有异步操作的结果可以决定当前是哪一种状态。

Promise状态变化有两种情况：进行中到已成功；进行中到已失败。任何情况一旦状态改变，就不会再变，被称为resolved（已定型）

如果某件事件偶尔发生，用Promise,否则用Stream模式

Promise对象是一个构造函数。

const promise = new Promise(function (resolve, reject)){if reslove(value) else reject(error)}

resolve和reject是两个函数。resolve用于将状态从未完成变为成功，在异步操作成功时调用，并将结果作为参数传递出来；reject用于将状态从未完成变为失败，并把错误作为参数传递出去。且这两个函数总是在最后才执行。若是最后一个执行则用return resolve()

Promise实例生成后可以用then分别指定resolved状态和rejected状态的回调函数。

promise.then(function(value),function(error)) 第一个回调函数是状态为成功时调用；第二个回调函数是状态为失败时调用，第二个函数是可选的。

Promise新建后会立即执行，then里的会等所有同步操作结束再执行。

* 1. then(): 为Promise实例添加状态改变时的回调函数。返回的是新的Promise实例，所以可用链式写法（.then().then()）
  2. catch(): 用于指定发生错误时的回调函数，等同于then()的第二个回调函数。包括then()中的错误，也会被catch捕获。

一般用catch()替代then()中拒绝状态的回调函数。

* 1. finally(): 不管Promise对象最后状态如何都会执行的操作。
  2. all(): 将多个Promise实例，包装成一个新的Promise实例。参数必须具有Iterator接口。实例遵从&&运算规则。
  3. race(): 将多个Promise实例，包装成一个新的Promise实例。多个实例有一个改变状态，则新实例也改变状态，首先改变状态的实例的返回值传给新实例的回调函数。
  4. allSettled(): 接收一组Promise实例作为参数，包装成一个新的Promise实例，只有等到所有参数实例都返回结果，包装实例才会结束。

一般用于不关心异步操作的结果，只关心异步操作是否结束。

* 1. Promise.any(): 接收一组Promise实例作为参数，包装成一个新的Promise实例。只要参数实例有一个变成fulfilled状态。包装实例就会变成fulfilled状态；如果所有参数实例都变成rejected状态，包装实例就会变成rejected状态。
  2. Promise。resolve(): 将现有对象转为Promise对象。若参数是一个Promise实例，则不变；若是一个有then方法的对象，则转变完成后立即执行then方法；若是不具有then方法的对象或者不是对象，则Promise.resolved方法返回一个新的Promise对象，状态为resolved；不带有任何参数，返回一个resolved状态的Promise对象
  3. Promise.reject(): 返回一个新的Promise实例，状态为rejected
  4. Promise.try(): 让同步函数同步执行，让异步函数异步执行。便于更好的管理异常，等价于try代码块。

应用：加载图片、Generator函数管理、

## Iterator

Iterator指遍历器，遍历器本质是一个指针对象

使数据结构的成员能够按照某种次序排列

为各种数据结构提供统一的访问接口，即for...of。当for...of遍历某种数据结构时，会自动去寻找Iterator接口

一种数据结构只要部署了Iterator接口，则称这种数据结构是可遍历的。对应属性为Symbol.iterator,即有这个属性就是可遍历的。这个属性返回value和done,done表示是否遍历结束。

7种有Iterator接口的数据结构：数组、Map、Set、String、TypedArray、函数的arguments、NodeList。(一般对象没有)

可通过遍历器实现指针

默认调用Iterator接口：for...of;解构赋值；扩展运算符；yield\*;数组作为参数时候

可以重写原生Iterator接口，达到修改遍历器行为的目的。

方法：

next(): 必须的。下一个

return() 可选的。for...of提前退出时调用。

throw() 可选的。配合Generator使用。

## for...of

借鉴后端语言

for(let x of test){}

主要用于替代for...in来遍历数组

可实现forEach()的功能

for...in读取键名；for...of读取键值，且只读取具有数字索引的键值

## Generator

Generator函数是一种异步编程的解决方案

Generator封装了多个内部状态，用yield表达式定义不同的状态

实例：function \* test(){ yield ‘1’; return ‘2’} //两种状态 return表示结束执行状态

返回一个遍历器对象的指针，可以遍历Generator函数内部的每一个状态。

调用next()，使指针指向下一个状态，并运行经过的代码段。即Generator是分段执行的，yield/return暂停执行，next恢复执行。返回的值为yield表达式的结果

若yield在另一个表达式中，则必须放在圆括号内

alert(1+(yield));

若是函数参数或在赋值表达式右边可以不加括号

可以把Generator赋值给对象的Symbol.iterator属性，从而使对象具有iterator接口。

若yield表达式无返回值，则next的参数会被当作上一个yield表达式的返回值

for...of可以自动遍历Generator函数运行时生成的Iterator对象。

方法：

throw(): 在函数体外跑出错误，然后在Generator函数体内用try..catch捕获，会附带执行一次next。若已经捕获过或next无法执行，则转而让函数体外的try...catch捕获。若没有catch，则会报错。

return(): （函数体外）返回给定的值，并结束遍历Generator函数

yield\*: 解决Generator函数嵌套。next时，会遍历到表达式内部的值。任何数据结构有Iterator接口，就可以使用yield\*遍历。

可用于嵌套数据结构的一维展开（平铺）

二叉树的遍历：



若Generator函数变成对象的属性，则可以简写：let x={ \*test(){}; }

Generator函数没有自己的this，除非用call()方法绑定。

多用于Generator实现状态机，实现协程，异步操作（ajax），控制流管理，部署Iterator接口，作为数据结构

## **Generator 异步编程**

异步：一个任务不是连续完成的，先执行一部分，然后去执行其他任务，接着回头执行接下来的部分。一般是在等待需要的资源时候去执行其他任务，等资源到位了就回头执行。

用Generator函数替代Promise等过去的异步编程方法。

执行中遇到yield就去执行其他任务，即在需要异步操作的地方都用yield注明

Thunk函数是自动执行Generator函数的一种方法。

使用Thunkify模块

传名调用：将表达式传入函数体，只在用到的时候传值。

js中：Thunk替换多参数函数，替换成一个只接受回调函数作为参数的单参数函数。

任何函数，只要参数有回调函数，就能写出Thunk函数的形式。

Thunk函数用于Generator自动执行过程中的流程管理

co函数是一个用于Generator函数自动执行的工具

co函数不需要编写Generator函数的执行器，Generator函数传入co函数就会自动执行。co函数返回一个Promise对象。

使用co函数的前提是，Generator的yield命令后面是Thunk函数或Promise对象，亦或数组或对象的成员都是Promise对象。

Promise和Thunk本质都是对回调函数的包装

## async

Generator函数的语法糖

将Generator函数的星号替换成async，将yield替换成await。

优点：自带执行器（不用next和co）；语义清晰；await后面可以是原始类型的值，会自动转换成resolved的Promise对象。

async函数return的值会成为then方法回调函数的参数

自带执行器，所以必须等async函数内部的异步操作执行完才会执行then。若await后面的Promise对象变为reject状态，整个async函数都会中断执行，若希望不中断则放入try...catch结构或者紧跟一个catch函数去处理错误。

可用于休眠功能

如果await后面的异步操作出错，等同于async函数返回的对象呗reject.

## Class

ES5的语法糖，只是让对象原型的写法更加清晰。

构造函数 -> 类 ；

class test{

constructor(x,y){

this.x = x;

this.y = y;

}

toString(){

return x+y;

}

}

let b = new test();

b.toString();

类的所有方法都定义在类的prototype属性上面，除非使用this定义在本身。类的constructor属性指向类本身。

类的内部所有定义的方法都是不可枚举的。

constructor方法是类的默认方法，通过new生成对象实例时会自动调用。默认返回实例对象（this），也可以自定义。

类必须使用new调用。

所有实例共享一个原型对象。可以通过\_proto\_属性为类添加方法

类的可以使用get和set关键词

类的属性名可以采用表达式：let x=’aa’ class test{ [x](){}; } //方法名x采用aa

默认严格模式；

不存在变量提升；

name属性返回类名；

可以是存在Generator函数作为方法；

this默认指向类的实例；加上static的方法不会被实例继承（静态方法），父类的静态方法可以被子类继承；

实例属性可以被定义在constructor方法里，也可以定义在最外层；

静态属性老式写法（新式未定）：class test{} test.a = 1; //静态属性a

属性new.target: 返回new命令作用的构造函数，用于确定构造函数是如何调用的。

类可以通过extends关键字实现继承：class son extends father{}

子类必须在constructor方法中调用super方法，否则新建实例会报错，因为this指向不正确。子类通过super关键字新建父类的this对象。若子类没有定义constructor，则默认有constructor方法和super关键字。

Object.getPrototypeOf(): 从子类获取父类，借此判断继承关系。

super可用作函数或对象。函数就是在子类的constructor中使用；对象是在普通方法中，指向父类的原型对象，在静态方法中，指向父类。定义在父类实例上的方法和属性是无法通过super调用的

子类的\_proto\_总指向父类；子类prototype属性的\_proto\_属性，总是指向父类的prototype属性；子类实例的\_proto\_属性总是指向父类实例的\_proto\_

允许继承原生构造函数定义子类

# TypeScript

## ：

TypeScript中，使用:指定变量的类型

实例：test:string //变量test类型为字符串

## 编译

命令行编译方法： tsc 文件名

写的时候文件名后缀是.ts，编译后的文件名后缀是.js

## 原始数据类型

布尔值：boolean 数值：number 字符串：string 空值：void

void用undefined和null来代替

## 任意值

数据类型any表示允许赋值为任意值。

原始数据类型在赋值过程中改变类型是不被允许的（没有自动类型转换）

声明为一个变量为任意值后，对它的任何操作返回的内容类型都是任意值。

变量在声明时未指定类型且未赋值，会被识别为任意值

## 类型推论

如果没有明确的指定类型，那么TypeScript会依照推论的规则推断出一个类型。然后这个变量就自动被指定为推断的类型。

## 联合类型

联合类型表示取值可以为多种类型中的一种

用|分隔每个类型

只能访问联合类型所有类型共有的属性和方法，独有的是不能访问的。

## 对象的类型-接口

用接口定义对象的类型

实例：

interface IHuman{...}

let man:IHuman={...} //约定接口名加上I前缀

且字段名必须一致，少字段或多字段都是不被允许的。即形状一致。

如果不希望完全匹配，则字段名后加个?，变为可选字段。

如果希望可以有自定义的字段，则字段名定义为[propName:类型]：any; 但类型不能变。

如果希望某些字段只在创建时被赋值（只读），则在字段名前加上readonly，即readonly:字段名：number

## 数组的类型

类型+方括号表示法

实例：let test:number[]:[1,2,3]

不允许出现其他的类型

实例（任意类型）：let test:any[]:[1,’1’,false]

数组泛型

实例：let test Array<number>=[1,2,3]

用接口表示数组

实例：

interface InumberArray{

[index:number]:number

}

let test:InumberArray=[1,2,3,4,5]

类数组

实例(自定义)：

function sun() {

let args:{

[index:number]:number;length:number;callee:Function

} = arguments;

}

实例（已有）：

function sum(){

let args:IArguments = arguments

}

## 函数的类型

函数声明实例：

function test(x:number,y:number):number{

return x+y;

}

必须输入同样的参数，缺少或多余都是不被允许的

函数表达式实例：

let test:(x:number,y:number)=>number=function(x:number,y:number):number{

return x+y;

}

=>有TypeScript自己的定义，表示函数的定义，左边是输入类型，需要用括号括起来，右边是输出类型。

接口定义函数的形状实例：

interface ISearchFunc{

(source:string,subString:string):boolean;

}

let test:ISearchFunc;

test=function(souce:string,subString:string){

return source.search(subString)!==-1;

}

用？定义可选参数，可选参数后面不允许出现必选参数

参数允许添加默认值，有默认值的参数自动定义为可选参数（不受限制）。

可能有剩余参数

多态允许一个函数结构不同数量或类型的参数时作出不同的处理。

实例：

function test(x:number):number;

function test(x:string):string;

function test(x:number|string):number|string{

if(typeof x ===’number’){return x}

else if(typeof x===’string’){return -1;}

}

## 类型断言

手动指定一个值的类型

用法：<类型>值 或者 值as类型 //后面一种用途更加宽泛

类型断言可用于访问联合类型中某个类型独有的方法或属性

## 引用声明文件

使用第三方库时，我们需要引用它的的声明文件

通常会把声明语句放到一个单独的文件，即声明文件。以.d.ts为后缀

## 内置对象

在TypeScript中当作定义好了的类型

ECMAScript提供的内置对象：Boolean,Error,Date,RegExp

实例：let a:RegExp=/[a-z]/

DOM和BOM提供的内置对象：Document,HTMLElement,Event,NodeList

可以用TypeScript写Node.js，但是要引入第三方声明文件。

## 类型别名

类型别名用于给一个类型起个新名字，常用于联合类型。

实例：type Name = string|number;

## 字符串字面量类型

用于约束取值只能是某几个字符串中的一个

实例：type test=1|2|3;

## 元祖

数组合并了相同类型的对象，元祖合并了不同类型的对象

实例：let test:[string,number]=[‘1’,1];

若是直接赋值，则应该赋予所有不同类型的值

实例：

let test:[string,number]

test=[‘1’,1];

添加越界的元素时，类型会被限制为每个类型的联合类型。

## 枚举

枚举类型用于取值被限定在一定范围内的场景。

枚举使用enum关键词来定义

实例：enum Days {Mon,Tue}

枚举成员会被赋值从0开始递增的数字（反过来也可以），也可以手动赋值。

# 前端3D建模

## 基本原理

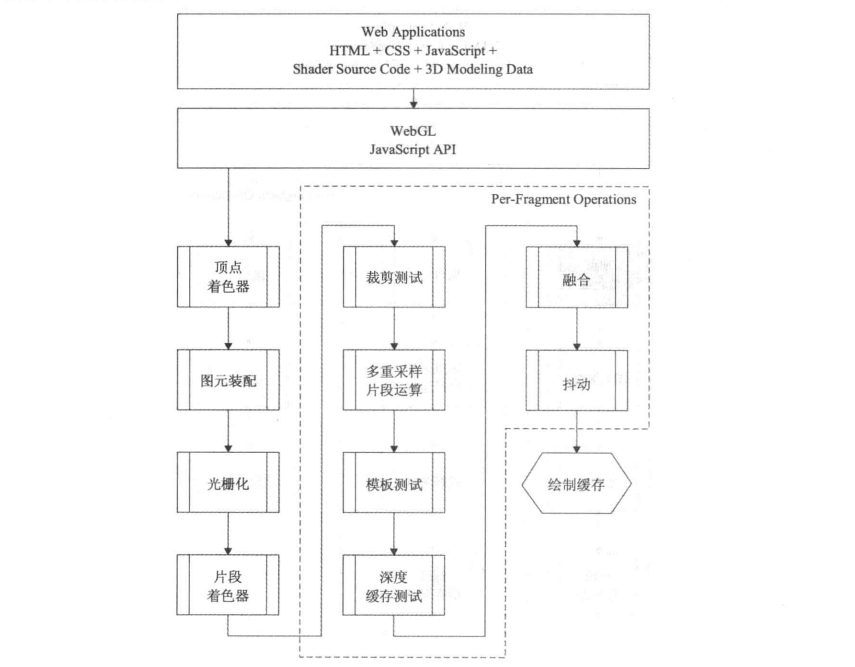
WebGL网页组成成分：HTML5、JavaScript和GLSL ES（着色器语言 OpenGL ES）

WebGL是基于光栅的API

WebGL只关注两个方面：投影矩阵的坐标和投影矩阵的颜色。

GPU有两个基础任务，第一个是将点处理为投影矩阵，第二个是基于第一点将相应的像素点秒回出来

## WebGL图形流水线总体结构



## 顶点着色器

顶点表示几何对象（点，线，圆，多边形等）的角点或交点

顶点着色器是对顶点着色的阶段

顶点着色器可以自己设计

顶点着色器的输入：

实际源代码

attribute变量：用户自定义的变量，通常用来包含特点于每个顶点的数据

uniform变量：表示所有顶点相同的数据，例如变换矩阵和光源位置。

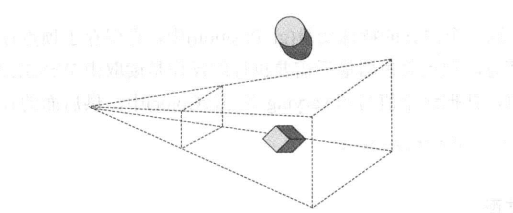
varying变量：顶点着色器向片段着色器发送的信息。

内置特殊变量：gl\_Position(最重要，保存顶点的位置信息)，gl\_FrontFacing，gl\_PointSize

## 图元装配

把已经着色的顶点装配成几何图元，然后判断它们当前是否在视锥体内，删除处于视锥体外的部分。

视锥体：



## 光栅化

图元转换为片段，然后把片段传送给片段着色器

## 片段着色器（像素着色器）

最终能够写入绘制缓存的片段对应屏幕上的一个像素

片段着色器的输入：

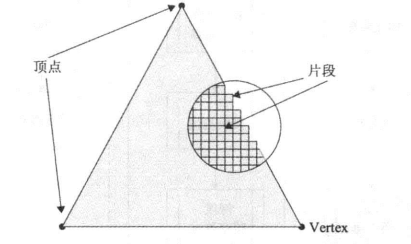
源代码

内置的特殊变量

varying变量（值由顶点着色器写入）

采样器（特殊的uniform变量）：用于纹理映射

顶点和片段的关系：



片段着色器通过特殊的内置gl\_FragColor变量输出结果，片段着色器把片段的颜色写到这些变量中。

WebGL要求所有片段着色器中都必须定义浮点类型数据的精度，通过精度限定符实现。

## 逐片段操作

# less

## 变量名

范围：优先本地搜索，若不可用，则在父作用域搜索

懒加载：可在使用后声明

* 1. 作为属性时：

@x :a

color:x;

* 1. 作为变量名时：

@x:a

@{x}

且可以嵌套使用：

@x:a

@y:x

@@{y}

## 函数

less预定义了几个函数用于操作css属性值

## 模板

模板：

.class1{

.val(@x){

color:@x;

fontsize:10px;

}

}

调用模板：

.myclass{

.class1 > .val(red)

}

## 扩展

* 1. :extend是一个伪类
  2. 使用extend的A 会使用 extend引用的B

less实例：

h{

&:extend(.style) //h的属性会包含style的属性

}

.style{

color: red;

}

转换后 => css实例：



逗号后面的代表被扩展的A，逗号之前的代表引用的B，总体表示属性原本是B的，现在A也有了。

=> A,B都有这个属性

* 1. 语法：

.a{ .b,

&:extend(.b) <=> .a:extend(.b){} <=> .a{}

}

* 1. :extend注意事项：

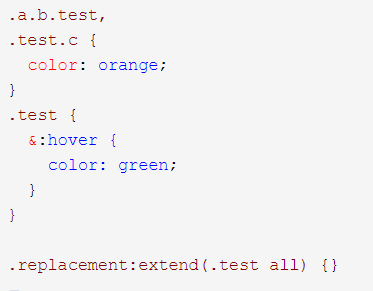
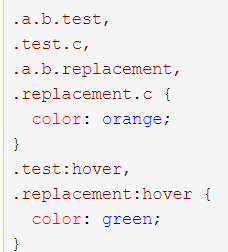
1. 在选择器之后
2. 选择器与:extend之间允许有空格
3. 允许多个扩展:extend():extend()... <=> :extend( , , ...)
4. :extend必须在末尾
5. 可扩展嵌套的选择器 a:extend(.b.c){}
6. 伪类在:extend中顺序不同则是指向不同匹配
7. :extend不能将选择器与变量相匹配，但变量可以使用:extend。
8. 在 media 媒体声明中写入的 extend 只匹配同一媒体声明中的选择器，；在媒体声明中写入的 extend ，不匹配其嵌套声明中的选择器；顶级 :extend 将匹配所有内容，包括嵌套媒体中的选择器
   1. ,a,b,c,d{x} <=> b,c,d拥有a的属性x => a,b,c,d拥有属性x
   2. 使用n相关的选择器时（css3），即使计算结果相同，但如果变量名不同也是不会匹配的。

而属性选择器中的引用类型不会影响：

[title=x]和[title=’x’]指向相同

* 1. :extend在末尾添加all关键字，它将替代另一个选择器的该部分匹配该选择器：

.a:extend(.x all) => .a会替代所有涉及.x的选择器作为新的选择 器。原来的.x相关的选择器依旧存在。

 <=> 

## Mixins(混合)

* 1. mixins是一组CSS属性，允许将一个类的属性用于另一个类。

简单使用：

.a(){} //mixins，内容是属性或者是扩展的其他选择器

.b{.a} //b使用a的属性，a不会出现在正式CSS文件中，类 似虚函数。

* 1. 命名空间用于外部封装mixin组，避免名称冲突：

#name(){ .a{} }

.b{ #name>.a }

保护的命名空间:增加guard条件，只有guard条件为true时才使用命名空间中的mixin

实例：

@color:blue;

#name when(@color = blue){

.a(){}

}

.b{ #name.a(); }

* 1. 当调用选择器使用!important时，被调用的mixin的所有属性都会加上！important

.b(.a()!)

* 1. 带参mixin,参数间逗号或分号分隔（一般分号）：

.class(@a;@b){}

.class2{.class(x,y);}

* 1. 用@arguments包括所有参数

.class(@a;@b;@c;@d){ x:@arguments}

* 1. mixin通过...提供可变数量的参数（ES6）、
  2. 模式匹配：通过传递不同的参数来选择不同的mixin（多态）

.a(x){ }

.a(y){ }

.b{ .a(x) }

* 1. 变量可以调用mixin里的对应值，但如果在本地就有变量的设置，则本地优先。
  2. 可以把mixin当作函数，入参后借用变量返回

.a(@x,@y){

@b:@x+@y;

}

.c{

.a(1,2);

d:@b

}

* 1. 可以在mixin内再使用mixin,且新的mixin可以被当作返回值。
  2. 可以把规则集传递给mixin,随之的所有内容都可以被mixin使用

## 导入

@import导入外部文件，没有扩展名，默认less格式文件。

@import(选项，选项) 选项会对导入文件附加限制条件，选项之间用逗号分隔。

选项：

reference:使用该文件，但不会表面表明使用

inline:会将CSS复制到输出CSS文件中，但不进行处理

less:无论扩展名是什么，都看作less文件

css:无论扩展名是什么，都看作css文件

once:(默认)该文件仅可以导入一次

multiple:该文件可以多次导入

optional：没有发现该文件时继续编译

## Guard运算符

* 1. Guard运算符：<, >, <=, >=, =

=可以比较数字，字符串，标识符，其余只能与数字一起使用。

* 1. 可以与mixin联动，用于多态: when (@a < 20px) and (@a > 5px)
  2. 否定：not(@a = 10px) and ...
  3. 类型检查：isxx(@a) 判断是否符合规则。

iscolor;isnumber;isstring;iskeyword;isurl;ispixel;ispercentage;ise m;isunit

## 循环

定义mixin: .mixin(@x){}

定义数组（存储变量名）：@List: a,b,c

循环：

.loop(@i) when(@i < length(@List)+1){

.mixin(extract(@List,@i));

.loop(@i+1);

}

.loop(1);

//length()获取列表长度，extract()获取列表元素，最后确定循环此处（.loop(1)）

## 合并

添加属性值到最后

.mixin(){ x:c}

.test(){

.mixin();

x+:d //结果：x:c,d

}

添加属性值和空格

x+\_: d

结果：x:c d

## 父选择器

&：用于引用父选择器，可重复使用

* 1. 减少冗余代码，一般伪类都要重启一行，用&可以直接在父选择器内定义伪类
  2. 替代相同的类名

&-x{} &-y{} ...

* 1. 改变选择器顺序,&前的选择器成为最外层的选择器

.a{ .b&{} } <=> .b .a{}

.a{ .b{ .c{ .d& }}} <=> .d .a .b .c{}

* 1. 排列组合

a,b{

&+&{}

}

a+a,a+b,b+b都会使用该选择器

# Node.js（网上教程）

## 概念

Node.js是一个基于Chrome V8引擎的JavaScript运行环境

Node.js是一个让JavaScript运行在服务端的开发平台

Node.js使用了一个事件驱动，非阻塞式I/O模型

Node.js对一些特殊用例提供特殊的API

Node.js是单线程，但借助事件循环来实现并发操作

## nvs

nvs是个Node.js版本管理工具，可以切换不同版本

使用nvs代替nvm

## 组成部分

* 1. 引入required模块：使用require指令来载入Node.js模块
  2. 创建服务器：服务器用于监听客户端的请求
  3. 接收请求与响应请求：客户端可以使用浏览器或终端发送HTTP请求，服务器接收请求后，返回响应数据。

## 创建简单的服务器

require指令载入http模块 =>

http.createServer()创建服务器 =>

通过request,response来接收和响应数据 =>

listen方法绑定服务器监听的端口号

## NPM

NPM是包管理工具，NPM是Node.js的一个“社区”，常用于：

* 1. 从NPM服务器下载他人编写的包或命令行程序
  2. 将自己编写的包或命令行上传到NPM服务器供他人使用

下载后的包/第三方程序会存放在工程目录的node\_modules目录中。

第三方包的依赖可以写在package.json文件中，从而NPM下载大包时会顺带把需要的包一同下载下来，而不用特地去下载该第三方包。

package.json必要字段：

name- 包名，在NPM服务器上需要保持唯一

version- 当前版本号

dependencies-第三包依赖，需要指定包名和版本号

main- 入口模块位置

bin- 命令行程序名和主模块位置

dependencies和devDependencies区别：使用--save-dev是将包写入devDependencies区域中，使用--save是写入dependencies区域中；devDependencies里的包只用于开发环境，不用于生产环境，而dependencies也要用于生产环境。一般devDependencies存放构建工具包（gulp.webpack），其余依赖包存在dependencies.

命令：

npm install 下载

npm publish 发布

npm help 查看所有命令

npm help<command> 查看某条命令的详细帮助

npm update<package> 把当前目录下node\_modules子目录中的对应模块更新至最新版本

npm update<package> -g 全局更新

npm cache clear 清空NPM本地缓存

npm unpublish<package>@<version> 撤销自己发布的某个版本代码

package.json所在目录下 npm install -g 可先在本地安装当前命令行程序，多用于发布前的本地测试

## REPL（交互式解释器）

REPL是Node.js自带的交互式解释器。我们可以在其中输入命令，并收到系统的响应。

cmd -> node进入->.exit退出

## 回调函数

Node.js异步编程寄托于回调函数实现

阻塞是按顺序执行；非阻塞先执行完当前代码，再去执行回调成功的函数。

## 事件循环

Node.js单线程进入一个类似while(true)的事件循环，直到没有事件观察者退出，每个异步事件都生成一个事件观察者，如果有事件发生就调用该回调函数。

事件驱动：每收到一个请求，放入事件列表，让主程序通过非阻塞I/O方式来处理请求。事件列表用于存放未能及时处理的事件。

非事件驱动的，会因为需要某个条件触发才能执行程序，而导致不断检查该条件，直到条件满足，十分浪费cpu;事件驱动会进入睡眠，直到事件触发而被唤醒。

## 事件

events模块只提供一个对象：events.EventEmitter。EventEmitter用于事件的发送和监听功能的封装。

对于每一个事件，EventEmitter允许设置多个事件监听器

当事件发送时，注册到这个事件的事件监听器被依次调用，事件参数作为回调函数参数传递。

EventEmitter常用的API：

EventEmitter.on(event,listen); emitter.addListener(event, listener): 为指定事件注册一个监听器，接收一个字符串event和一个回调函数listener

EventEmitter.emit(event,arg1,arg2...): 发送event事件，传递若干个可选参数到事件监听器的参数表。

EventEmitter.once(event,listener): 为指定事件注册一个单次监听器，即该监听器只会触发一次，触发后立刻被解除，

EventEmitter.removeListener(event,listener):移除指定事件的某个监听器，listener必须是该事件已经注册过的监听器。

EventEmitter.removeAllListener([event]):移除所有事件的所有监听器，如果指定event，则移除指定事件的所有监听器。

error事件：遇到异常时发送。如果没有响应的监听器，则Node.js会当作异常，退出程序并打印调用栈。所以要设置error的监听器，避免程序崩溃：emitter.emit(‘error’)

一般使用时,不直接使用EventEmitter，而是在对象中继承。这样的话事件的监听和发送是一个对象的方法，更加符合语义；而且不容易打乱对象原有的继承关系。

## Buffer（缓冲区）

JavaScript语言只有字符串数据类型，没有二进制数据类型。

处理TCP或文件时，必须用到二进制数据

=>Node.js定义一个Buffer类，该类用来创建一个专门存放二进制

数据的缓冲区。

* 1. 写入Node缓冲区：buf.write(string[,offset[,length]][,encoding])

string:写入缓冲区的字符串;offset:缓冲区开始写入的索引值，默认为0;length:写入的字节数，默认为buffer.length;encoding:使用的编码，默认为utf8。返回实际写入的大小，如果buffer空间不足，则只会写入部分字符串。

* 1. 从缓冲区读取数据:buf.toString([encoding[,start[,end]]])

encoding:使用的编码，默认为utf8;start:指定开始读取的索引位置，默认为0;end:结束位置，默认为缓冲区的末尾。

* 1. 将Buffer转换为JSON对象：buf.toJSON()
  2. 缓冲区合并:Buffer.concat(list[,totalLength])

list:用于合并的Buffer对象数据列表;totalLength:指定合并后Buffer对象的总长度。

* 1. 缓冲区比较:buf.compare(otherBuffer)

otherBuffer:与buf对象比较的另外一个Buffer对象。

返回0相同，返回小于0 buf在otherBuffer之前，反之在后，

* 1. 拷贝缓冲区:buf.copy(target[,targetStart[,sourceStart[,sourceEnd]]])

targetBuffer:要拷贝的Buffer对象；targetStart、sourceStart、sourceEnd:数字，可选，默认0；

* 1. 缓冲区裁剪：buf.slice([start[,end]])

start:数字，可选 默认0；end:数字，可选，默认buffer.length

* 1. 缓冲区长度（内存长度）：buf.length

等.....

## Stream（流）

Stream是Node.js中的一个模块

Stream实现数据从一个地方流动到另一个地方的功能

Stream有四种流类型：

Readable:可读 Writeable:可写

Duplex:可读可写 Transform:操作被写入数据，然后读出结果

所有Stream对象都是EventEmitter的实例。

常用的事件:

data:当有数据可读时触发

end:没有更多的数据可读时触发

error:在接收和写入过程中发生错误时触发

finish:所有数据已被写入到底层系统时触发

管道流提供了一个输出流到输入流的机制。（字面意思）

管道流能够从一个流中获取数据并将数据传递到另外一个流中（文件复制等）：readerStream.pipe(writeStream)

链式流是通过连接输出流到另外一个流并创建多个对个流操作链的机制。链式流一般用于管道操作。（压缩和解压文件）

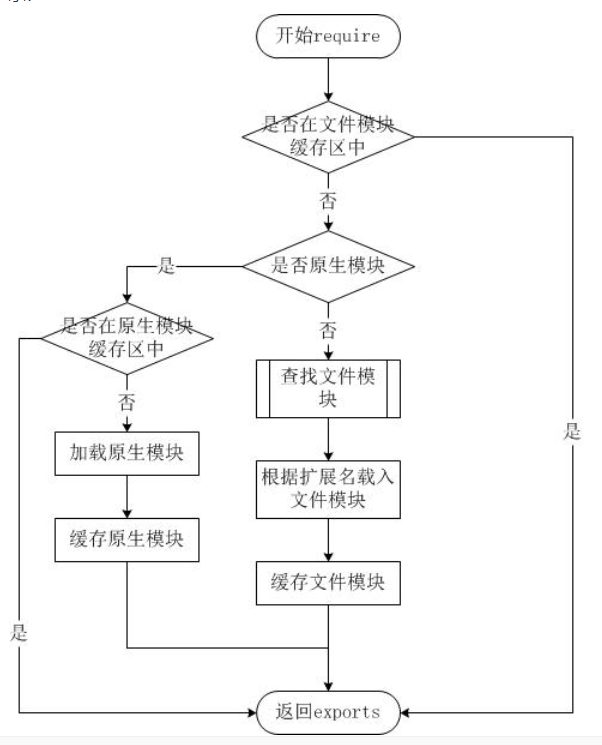
## 模块系统

Node.js为了便于文件相互间的调用，提供了一个简单的模块系统。

模块是Node.js程序的基本组成部分。

一个Node.js文件就是一个模块

require(import)加载优先级：



## 函数

把函数作为变量传递，简化代码

## 路由

我们要为路由提供请求的url和对应的GET(POST)参数，随后路由根据这些数据来执行相应的代码。

解析这些数据需要用到url模块和querystring模块。

## 全局对象

JS全局对象是window，Node.js全局对象是global，所有全局变量都是global对象的属性。

而在Node.js中能够直接访问到对象通常都是global的属性。

global最根本的作用是作为全局变量的宿主。

* 1. process：描述当前Node.js进程状态的对象

process.stdout:标准输出流。process.stdout.write()比console.log更加底层

process.stdin:标准输入流。一开始是被暂停的，若要从标准输入流读取数据，必须手动恢复流，并编写流的响应函数。

process.nextTick(callback):为事件循环设置一项任务，Node.js会在下次事件循环调响应时调用callback。注：Node.js一个编程原则就是尽量缩短每个事件的执行时间，而process.nextTick()提供这样一个工具，可以把复杂的工作拆散，变成一个个较小的事件。

* 1. console:提供控制台标准输出。

console.log():向标准输出流打印字符并以换行符结束。

console.error():向标准错误流输出

console.trace():向标准错误流输出当前的调用栈

## util

Node.js常用工具，提供常用函数的集合。

* 1. util.inherits（继承对象，原对象）是一个实现对象间原型继承的函数
  2. util.inspect（转换的对象，[showHidden],[depth],[colors]） 将任意对象转换为字符串，常用于调试和错误输出，至少接收一个object参数（转换的对象）。showHidden(可选)：如果值为true，将会输出更多的隐藏信息;depth:最大递归的层数，默认2层，指定null则不限；color:若为true,则输出格式将会以ANSI颜色编码，通常用于在终端显示更加漂亮的效果。
  3. util.isArray(object):判断参数是否是数组，是则返回true,不是返回false。
  4. util.isRegExp(object):判断参数是否是正则表达式，是则返回true，不是返回false。
  5. util.isDate(object):判断参数是否是一个日期，是则返回true，不是返回false。
  6. util.isError(object):判断参数是否是一个错误对象，是则返回true，不是返回false。

## 文件系统

Node.js文件系统被封装在fs模块，它提供了文件的读取，写入，更名，删除，遍历目录，链接等POSIX文件系统操作。

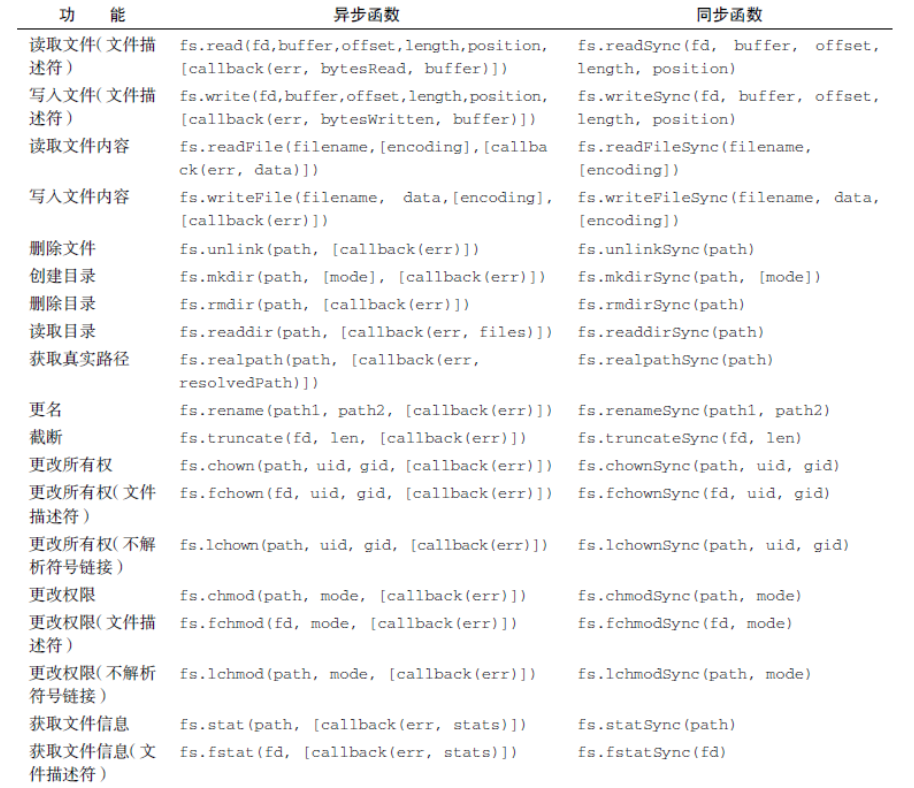
fs模块中所有的操作都提供了异步和同步的两个版本。

* 1. fs.readFile(filename,[encoding],[callback(err,data)]): 异步读取文件。filename(必选):读取的文件名；encoding(可选):表示文件的字符编码；callback:回调函数，接收文件的内容。

回调函数提供err和data,err表示有没有错误发生，data是文件内容。未指定encoding时data是以Buffer形式表示的二进制数据。

* 1. fs.readFileSync(filename,[encoding]):fs.readFile同步的版本
  2. fs.open(path,flags,[model],[callback(err,fd)]):打开文件。path(必选):文件的路径；flags(必选)：打开方式。r(读模式)、r+(读写模式)、w(写模式。文件不存在则新建)、w+(读写模式。文件不存在则新建)。
  3. fs.read(fd,buffer,offset,length,position,[callback(err,byteRead,buffer)]):读取数据。fd:读取数据并写入buffer指向的缓冲区对象；offset：buffer的写入偏移量；length：从文件中读取的字节数；position:文件读取的起始位置，若为null,则会从当前文件指针的位置读取；callback:回调函数传递bytesRead和buffer，分别表示读取的字节数和缓冲区对象。

fs模块函数表：



## 工具模块

os模块：提供基本的系统操作函数

Path模块：提供了处理和转换文件路的工具

Net模块：用于底层的网络通信，系统了服务端和客户端的操作

DNS模块：用于解析域名

Domain模块：简化异步代码的异常处理，可以捕捉处理try-catch无法捕捉的

## Web模块

Node.js提供了HTTP模块，http模块主要用于搭建HTTP服务端和客户端。

## Express框架

Express是一个为Node.js设计的web开发框架

使用Express可以快速搭建一个完整功能的网站：

设置中间件(body-parser,multer)来响应HTTP请求(JSON,Raw,Text,URL)

定义了路由表用于执行不同的HTTP请求动作

可以通过向模板传递参数来动态渲染HTML页面

Express需要下载cookie-parser解析Cookie

Express使用回调函数的参数：request和response对象来处理请求和响应的数据。

* 1. request常见属性：HTTP请求

req.app:当callback为外部文件时，使用此访问express实例。

req.baseUrl获取路由当前安装的URL路径。

req.body/req.cookie:获得[请求主体]/Cookies

req.fresh/req.stale:判断请求是否还是新的

req.hostname/req.ip:获取主机名和IP地址

req.originalUrl:获取原始请求URL

req.params:获取路由的parameters

req.path:获取请求路径

req.protocol:获取协议类型

req.query:获取URL的查询参数串

req.route:获取当前匹配的路由

req.subdomains:获取子域名

req.accpets:检查请求的Accept头的请求类型

req.acceptsCharsets/req.accrptsEncodings/req.acceptsLanguages

req.get:获取指定的HTTP请求头

req.is:判断请求头Content-Type的MIME类型

* 1. Response对象常见属性：HTTP响应，即接收待请求时向客户端发送的HTTP响应数据。

res.app:同req.app

res.append:追加指定HTTP头

res.set：在res.append后将重置之前设置的HTTP头

res.cookie(name,value[,option]):设置Cookie

res.clearCookie:清除Cookie

res.download:传送指定路径的文件

res.get:返回指定的HTTP头

res.json:传送JSON响应

res.jsonp:传功JSONP响应

res.location:只设置响应的Location HTTP头，不设置状态码或者close response

res.redirect:设置响应的Location HTTP头，并且设置状态码302

res.send:传送HTTP响应

res.sendFile(path,[,option][,fn]):传送指定路径的文件-会根据文件extension设定Content-Type

res.set:设置HTTP头，传入object可以一次设置多个头

res.status:设置HTTP状态码

res.type:设置Content-Type的MIME类型

* 1. Express提供了内置的中间件express.static来设置静态文件。

express.static(静态文件文件夹)

* 1. process\_get路由器处理GET请求；process\_post路由器处理POST请求

## RESTful API

REST:表述性状态传递 REST是一种设计风格

表述性状态转移是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是RESTful。

简单来说：同一个URL GET获取资源；POST新建资源；DELETE删除资源；PUT 更新资源。

## 多进程

在多核CPU的系统上创建多个子进程

每个子进程带有三个流对象：child.stdin,chile.stdout,child.stderr

Node.js提供了child\_process模块来创建子进程

child\_process.exec(command[,option],callback):使用子进程执行命令，缓存子进程的输出，并将子进程的输出以回调函数参数的形式返回。command 字符串，将要运行的命令；options 对象，设置需要的选项；callback 回调函数；等待进程结束，一次性返回最大的缓冲区内容。

child\_process.spawn(command[,args][,options]):使用指定的命令行参数创建新进程。command 将要运行的命令；args Array字符串参数数组；options 设置需要的选项；返回流，且进程开始执行spawn()时就开始接收响应，多用于进程返回大量数据。

child\_process.fork(modulePath[,args][,options]):是spawn的特殊形式，用于在子进程中运行的模块，同时在父子进程之间建立一个通信管道，用于进程之间的通信。modulePath 字符串，将要在子进程中运行的模块；args 字符串参数数组；options 设置需要的选项；返回的对象有ChildProcess实例的所有方法和一个内建的通信信道。

## JXcore

JXcore是Node.js的一个发行版本，支持多线程。

JXcore可以让你不需要对现有的代码做改动，直接以多线程运行。

# 《深入浅出Node.js》

## Node的特点

Node是当作后端JavaScript的运行平台

它将前端的思想迁移到服务器端

## 模块

Node对引入过的模块都会进行缓存，以减少二次引入时的开销。而且Node缓存的是编译和执行后的对象。

对相同模块二次引用都是缓存优先。每一个第一次编译成功的模块都会将其文件路径作为索引缓存在Module.cache对象上。

若标识符不含文件扩展名，Node会按照.js、.json、.node的次序依次补足扩展名。（为提高速度，标识符需带上扩展名）

每个文件都是一个模块，每一个文件模块又都是一个对象。

路径分析和文件定位

在编译过程中，Node对获取的JS文件内容进行了包装，所有JS文件外部包了层（function(exports,require,module,\_filename,\_dirname)）{}）。从而使JS文件可以使用这几个关键词，且模块文件之间都进行了作用域隔离。

核心模块中，C/C++文件在Node的项目的src目录下；JS文件在lib目录下。核心模块中的JS文件也需要进行包装。

C++模块完成核心，JS实现包装的模式是Node能够提高性能的常见方式。

纯C/C++编写的部分统一称为内建模块。不常被用户直接调用，如fs,os,evals,buffer等模块。内建模块也是最底层的模块。

脚本语言开发速度优于静态语言，静态语言性能优于脚本语言。

一般核心模块都封装了内建模块，若要调用内建模块可以调用封装后的核心模块。

总结：C/C++是最底层的模块，提供API给JS核心模块和第三方JS文件模块调用，但如果不是非常熟悉，不要直接调用；JS核心模块主要用于C/C++内建模块的封装层和桥接层以及纯粹的功能模块；文件模块通常由第三方编写，包括普通JS模块和C/C++扩展模块，主要调用方向为普通JS模块调用扩展模块。

C/C++

package.json是包描述文件

NPM所有行为都与该文件息息相关

-g是指将一个包安装为全局可引用的状态。

## 异步I/O

异步/同步和阻塞/非阻塞是两个不同的概念

同步指发出一个功能调用时，在没有得到结果之前，该调用就不返回。

阻塞指发出一个功能调用时，在没有得到结果之前，当前线程会被挂起，直到得到结果才返回。

区别：阻塞是从程序等待调用结果状态角度去看；同步是从消息通信机制去看 https://blog.51cto.com/yaocoder/1308899

操作系统内核对于I/O只有两种方式：阻塞和非阻塞。在阻塞I/0时，调用后一定要等到系统内核层面完成所有操作后，调用才结束；非阻塞I/0则是在调用后不带数据直接返回，要获取数据，需要再次读取。一般将非阻塞I/O重复调用的操作称为轮询。

Node是异步，非阻塞I/0。异步体现在回调函数等；非阻塞I/O体现在事件循环过程中的轮询机制。

Node是JS执行时是单线程的，但其他功能是多线程的（内部完成I/O任务），所以Node本身是多线程的。

进程启动时，Node会创建一个循环，每一次循环称为一次Tick。每个Tick的过程就是看是否有事件处理，有就执行并执行关联的回调函数，然后加入下次循环(continue)，没有事件直接加入下次循环(continue)。

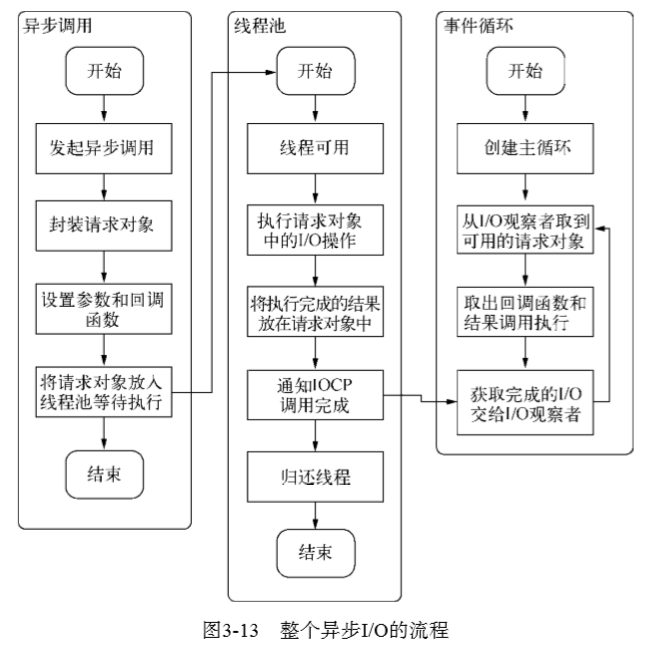
使用观察者来判断是否有事件需要处理，且不同类型事件会有不同的观察者来处理。

异步实现：在libuv封装层调用uv\_fs\_open方法，在调用过程中创建了一个FSRedWrap请求对象，JS传入的参数和回调等方法都被封装在这个请求对象中。在Windows下，调用QueueUserWorkItem方法将请求对象推入线程池等待。至此JS调用立即返回，继续执行JS的后续内容。而I/O操作则在线程池中等待，不会影响JS线程的后续执行。

回调实现：线程池中的I/0操作调用完毕之后，会通知IOCP，告知当前对象操作已经完成（向IOCP提交执行状态，并将线程归还线程池）。同时观察者检查到线程池中有未执行完的请求，并将请求对象加入处理队列，处理时执行对应回调函数（请求对象中保存的）。

IOCP：windows实现异步的内核。方法是事先开好N个线程，将所有用户请求都投递到一个消息队列中，而后让线程从消息队列中取出消息并处理。避免用完的线程的浪费。

对比传统：每个请求开个新线程。



事件循环、观察者、请求对象、I/0线程池共同构成了Node异步I/O模型的基本要素。

JS线程用于分配任务和处理结果，I/O线程池里的各个I/0线程负责完成分配来的任务。

## 非I/0的异步API

* 1. 定时器

setTimeout和setInterval分别用于单次和多次定时执行任务。与异步I/0类似，但不需要I/0线程池。

调用时创建的定时器会被插入定时器观察者的一个红黑树，每次Tick从红黑树迭代取出定时器对象，检查是否超过定时时间，如果超过，就生成一个事件去执行回调函数。

问题是时间不精确，决定何时生效的是每一次的Tick，而非本身指定的时间。

* 1. process.nextTick

process.nextTick用于替代setTimeout立即异步执行一个任务。

每次调用会将回调函数放入队列，在下轮Tick时取出执行。

## setImmediate

setImmediate功能和process.nextTick类似，都是将回调函数延迟执行。

但process.nextTick中的回调函数优先级高于setImmediate。原因是process.nextTick属于idle观察者，setImmediate属于check观察者。而对观察者的检查是有先后顺序的，idle观察者高于I/O观察者高于check观察者。

而且process.nextTick的回调函数保存在一个数组中，每次Tick会将数组中的回调函数执行完；setImmediate的是保存在链表中，每次Tick只执行一个回调函数。

## 异步编程注意点

* 1. Node的异步使得try/catch无法获取回调时的错误，所以将异常作为回调函数的第一个实参传回，只需正确传递异常，无需过多处理。
  2. Node函数容易嵌套过深，多出现于一个业务逻辑依赖于多个回调函数
  3. Node不能阻塞代码，难以实现同步需求：长时间的setTimeout替代
  4. JS单线程通常指的是JS执行线程和UI渲染共用一个线程。如果服务器是多核CPU，则单个Node进程没有充分利用。Web Workers提出将JS执行和UI渲染分离。

## 事件发布/订阅模式

事件发布/订阅模式替代事件冒泡和控制事件传递。可以实现一个事件与多个回调函数的关联，这些回调函数又被称为事件侦听器。

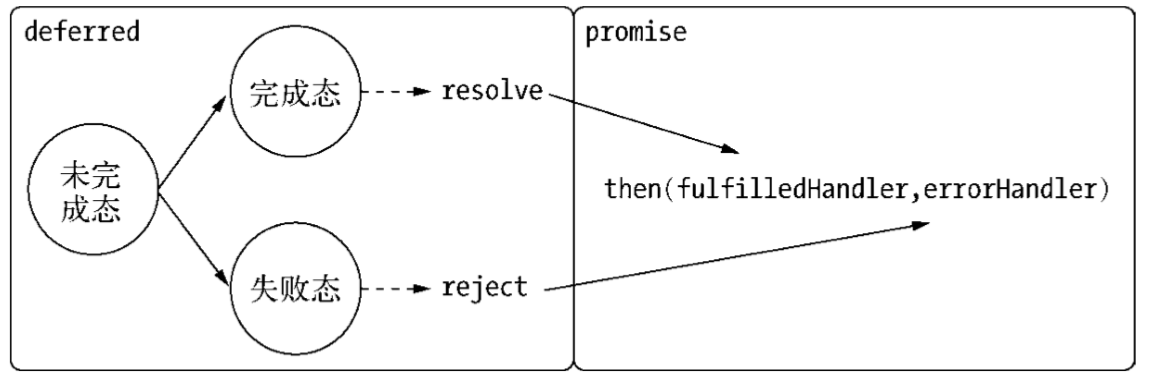
事件发布者无需关注订阅的侦听器如何实现业务逻辑，便于解耦业务逻辑，也是一种钩子机制，利用钩子导出内部数据或状态给外部的调用者。

利用事件队列解决雪崩问题。雪崩问题指高访问量、大并发量的情况下，缓存失效，大量请求同时涌入数据库，单条SQL反复查询，而数据库无法承受。用Once方法，它添加的侦听器只能执行一次，执行之后就会将它与事件的关联移除。

## Promise/Deferred模式

传统AJAX在异步调用前要把执行事件（success,error,complete）都准备好，Promise/Deferred模式允许先执行异步调用，再执行事件：$.get().success(1).success(2).error().complete()。

deferred主要用于内部维护异步模型状态；promise作用于外部，通过then方法暴露给外部以添加自定义逻辑。



when和Q模块是完整的Promise实现。

## 流程控制

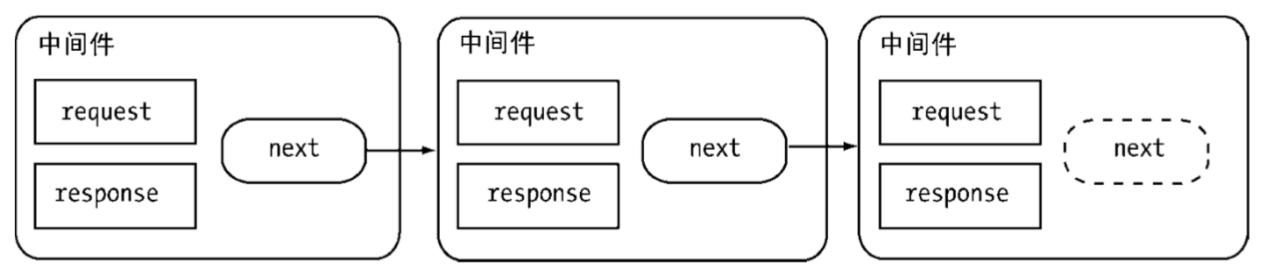
node是异步的，而有些事件必须有前置事件回调成功才能执行。一种方法是嵌套回调函数，更好的另一种方法就是流程控制。

* 1. 尾触发

尾触发：手动调用才能持续执行后续调用的，常见关键词是next。

适合处理网络请求的场合

中间件传递请求对象，响应对象和尾触发函数：



* 1. async

async提供了series方法实现一组任务的串行（顺序）执行。每次执行会将结果保存起来，然后执行下一次调用。

async提供了parallel方法实现异步操作的并行执行。只有所有异步调用都正常完成，才会将结果以数组的方式传入。

当前结果是下一次调用的输入，且是异步串行执行：waterfall。

auto(): 自动依赖处理,解决复杂的依赖关系。

* 1. Step

只有一个接口Step(task1,task2....)

接收任意数量的任务，所有任务都将会串行依次执行。

Step可以使用parallel使得多个异步任务并行执行（但Step至多会取回传的前两个参数）。

* 1. group

类似Step,最后回调函数接收的返回值存储方式不同。

step:function(err,result1,result2..)

group:function(err,results)

## 异步并发控制

Node可以使用async发起无数次异步调用，但如果并发量过大，下层服务器会难以承受,所以需要一定的过载保护。

async提供了parallellimit,parallelimit和parallel类似，但多了个限制并发数量的参数，使得任务只能同时并发一定数量。

如果又需要动态地增加并行任务，则可以使用queue，缺点是接收的参数是固定的。

## 内存控制概要

Node有内存消耗低的优点，适合处理海量的网络请求。内存控制是在海量请求和长时间运行的前提下进行探讨的。

V8有内存限制，所以Node也有内存限制（可以手动扩大V8内存限制）。

## 垃圾回收

V8按对象的存活时间将内存分为新生代和老生代，新生代中的对象为存活时间短的对象；老生代中的对象为存活时间长的对象；整体大小就是新生代加老生代的内存大小。

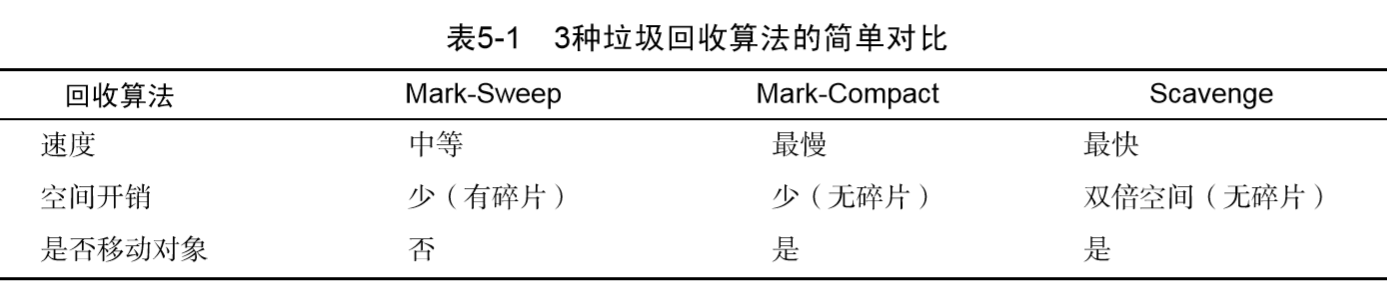
新生代使用Scavenge算法进行垃圾回收，Scavenge具体实现又采用了Cheney。

Cheney将堆内存一分为二，一个处于使用中（From空间），一个处于闲置状态（To空间）。分配对象先在From空间分配。垃圾回收时，检查From空间的存活对象，存活对象被复制到To空间，非存活对象呗释放，完成复制后两个空间角色互换。

新生代移到老生代内存的过程叫做晋升。晋升条件是对象建立过一次Scavenge回收或者新生代的To空间内存占比超过25%。

老生代采用Mark-Sweep和Mark-Compact相结合的方式进行垃圾回收。

Mark-Sweep是标记清除的意思，它分为标记和清除两个阶段。它在标记阶段遍历堆中的所有对象，并标记活着的对象，在随后的清除阶段，只清除没有标记的对象。但会产生内存碎片的问题，于是用到Mark-Compact，Mark-Compact是标记整理的意思。差别在于对象被标记为死亡后，在整理时，将活着的对象往一端移动，移动完成后，直接清理掉边界的内存。



V8主要使用Mark-Sweep，在空间不足时对从新生代晋升过来的对象进行分配时才使用Mark-Compact。

全停顿：垃圾回收时需要将应用逻辑暂停。

为降低全停顿时间，V8在标记阶段将原本需要一口气完成的，分为一段一段执行，端间让应用逻辑执行一会儿，这个动作称为增量标记。同样有清理阶段的延迟清理以及整理阶段的增量式整理。

查看垃圾回收日志：在启动时添加--trace\_gc参数，然后从标准输出中打印垃圾回收的日志信息。

V8执行时的性能分析数据：Node启动时使用==prof参数。能够查看垃圾回收执行时间。

V8提供了linux-tick-processor工具用于统计日志信息。

os模块的totalmem()和freemem()也可以查看内存使用情况，totalmem返回系统的总内存，freemem返回系统的闲置内存。

Node存在非V8分配的内存，称为堆外内存，例如Buffer对象。

## 内存泄漏

* 1. 限制将内存当作缓存的行为
  2. 消费者-生产者模型中，消费速度低于生产速度，队列会形成堆积，从而出现内存泄漏。需要监控队列的长度。

可以使用各种工具定位Node应用的内存泄漏。

## 大内存应用

Node提供了stream模块用于处理大文件。stream继承自EventEmitter,分为可读和可写两种。

Node中的大多数模块都有stream的应用。

## Buffer结构

Buffer占用的内存属于堆外内存

Node在进程启动时就已经加载了，无需require。

Buffer是一个经典的JS与C++结合的模块，性能相关用C++实现，非性能用JS实现。真正的内存是C++层面提供的，JS层面只是使用它。

Buffer对象类似数组，元素为16进制的两位数（0-255，如果不在范围内则自动加256或减256，使其在区间内；小数则只保留整数部分），不同类型字符串占用元素个数也不同。在utf-8编码下中文占3个元素，字母和标点符号各占1个元素。

可以通过访问length属性得到长度，也可以通过下标访问元素。

内存方面，采用在C++层面申请内存，在JS分配内存的策略。内存分配使用slab分配机制。

slab是一块申请好的固定大小的内存区域。有三种形态：full(完全分配状态)、partial(部分分配状态)、empty(没有被分配状态)。一开始为empty，分配内存后为partial

Node以8KB来区分Buffer是大对象还是小对象。因为8KB就是每个slab的大小，在JS层面作为单位单元进行分配的内存。

* 1. 小对象

创建的对象会使用一个局部变量pool作为中间处理对象，处于分配状态的slab单元会将新申请的SlowBuffer指向它。

第一次创建的Buffer对象在1个8KB的slab里，第二个会判断第一个是否放的下，放的下则放，放不下则再新开1个slab。

* 1. 大对象

直接分配一个SlowBuffer对象作为slab单元，这个slab单元将会被这个大对象独占。

SlowBuffer类都是在C++定义的。

## Buffer的转换

* 1. 字符串转Buffer

new Buffer(str,[encoding]) encoding不传时，默认UTF-8编码

* 1. Buffer转字符串

Buffer对象.toString([encoding],[start],[end])

* 1. 判断编码是否支持转换

Buffer.isEncoding([encoding])

对于不支持的编码类型，可以借助模块转换，比如iconv和iconv-lite。

## Buffer的拼接

直接用字符串方式拼接时data += chunk 等价于 data = data.toString()+chunk.toString()。英文字母（1个元素）不会有问题，中文作为宽字节（3个元素）可能会有问题。实际表现为原本的中文会变成�。

假设限制Buffer长度为11，则前3个中文（9个元素）会正常显示，后两个元素不能匹配任何中文就显示为乱码。

* 1. setEncoding

可以使用setEncoding让Buffer对象变为编码后的字符串，从而正确显示。setEncoding会在内部设置一个decoder对象，decoder对象知道宽字节在UTF-8下占3个字节。上例中的前9个字节转换为中文，后2个字节会保留到下次读取流。

* 1. 正确拼接Buffer

chunks.push(chunk)

size += chunk.length

用一个数组来存储接收到的所有Buffer片段并记录下所有片段的总长度。然后调用Buffer.concat生成一个合并的Buffer对象。Buffer封装了从小Buffer对象向大Buffer对象的赋值过程。

## Buffer与性能

在web应用中，字符串转换到Buffer是时时刻刻发生的，提高字符串到Buffer的转换效率，可以很大程度地提高网络吞吐率。

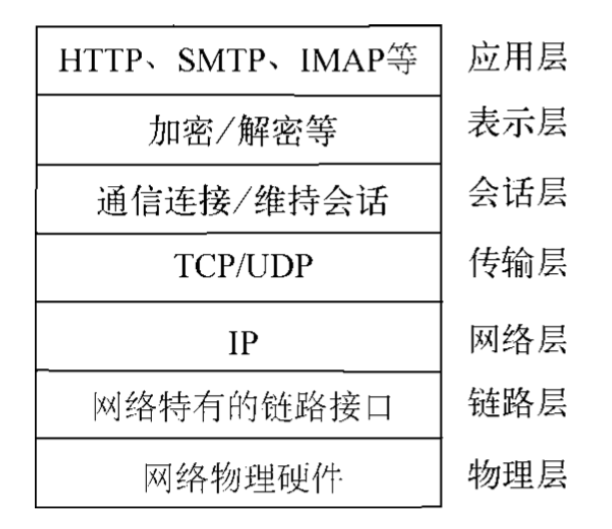
在不需要改变内容的场景下，尽量只读取Buffer，然后直接传输，不做额外的转换，避免损耗。

在文件读取时，highWaterMark参数在一定范围内设置的大一些

## 网络编程

Node提供了net模块处理TCP，dgram处理UDP，http处理HTTP,https处理HTTPS，且服务器端和客户端都适用。

OSI模型：



## TCP

TCP是面向连接的协议，显著特征是三次握手形成会话。只有会话形成后服务器端和客户端之间才能互相发送数据。在创建会话过程中，服务器端和客户端分别提供一个套接字，这两个套接字共同形成一个连接。服务器端与客户端则通过套接字实现两者之间连接的操作。

服务器事件

listening:在调用server.listen()绑定端口或者Domain Socket后触发

connection：每个客户端套接字连接到服务器端时触发

close：当服务器关闭时触发，调用后，服务器会停止接收新的套接字连接，但保持当前存在的连接，等待所有连接都断开后，会触发该事件

error:服务器发生异常触发。如果不监听则直接抛出异常。

连接事件

data:当一端调用write发送数据，另一端会触发data事件，事件传递的数据即是write发送的数据

end:当连接中的任意一端发送了FIN数据时触发

connect:当该事件用于客户端，当套接字与服务端连接成功时触发

drain:当任意一端调用write发送数据时触发

error:当发送异常时触发

timeout:当一定时间后连接不再活跃时触发，通知用户当前该连接已被闲置了。

Nagle算法：要求缓冲区的数据达到一定数量或一定时间后才发出，Node默认使用。

## UDP

UDP不面向连接，一个套接字可以和多个UDP通信，它提供不可靠消息传输。

套接字事件

message:当UDP套接字侦听网卡端口后，接收到消息时触发该事件，触发携带的数据为消息Buffer对象和一个远程地址信息。

listening:当UDP套接字开始侦听时触发

close：调用close方法时触发，并不再触发message事件

error：当异常发生时触发该事件

## HTTP

HTTP基于TCP

HTTP两端是服务器端和浏览器端，即B/S结构

HTTP任务：处理HTTP请求和发送HTTP响应

Node中的HTTP继承自TCP，它能与多个客户端保持连接，由于采用事件驱动的形式，并不为每个连接创建额外的线程或进程，保持很低的内存占用，从而实现高并发。

HTTP请求：ServerRequest对象。报文体中的数据要在报文头解析后再被读取。

HTTP响应：setHeader,writeHead。可以调用setHeader进行多次设置，但只有调用writeHead后，报头才会写入到连接中。报头会在报文体之前发送，一旦开始数据的发送，writeHead和setHeader将不再失效。注：无论什么情况，结束时调用res.end结束请求是必要的，否则客户端将一直处于等待的状态。

HTTP服务的事件：

connection：建立底层的TCP连接时触发一次，该连接可以在多次请求响应之间使用。

request：解析出HTTP头后触发。

close：调用server.close来停止接收新的连接，当已有的连接都断开时，触发该事件。

checkContinue:客户端在发送大的数据时，会先发送一个头部请求给服务器，此时服务器触发该事件。如果不接受大数据，则返回400，且不触发request，直到客户端收到100后重新发起请求才会触发request事件。

connect:客户端发起CONNECT请求后触发，通常在 HTTP代理时出现。

upgrade：当客户端要求升级连接的协议时，需要和服务器端协商，客户端会在请求头中带上Upgrade字段。

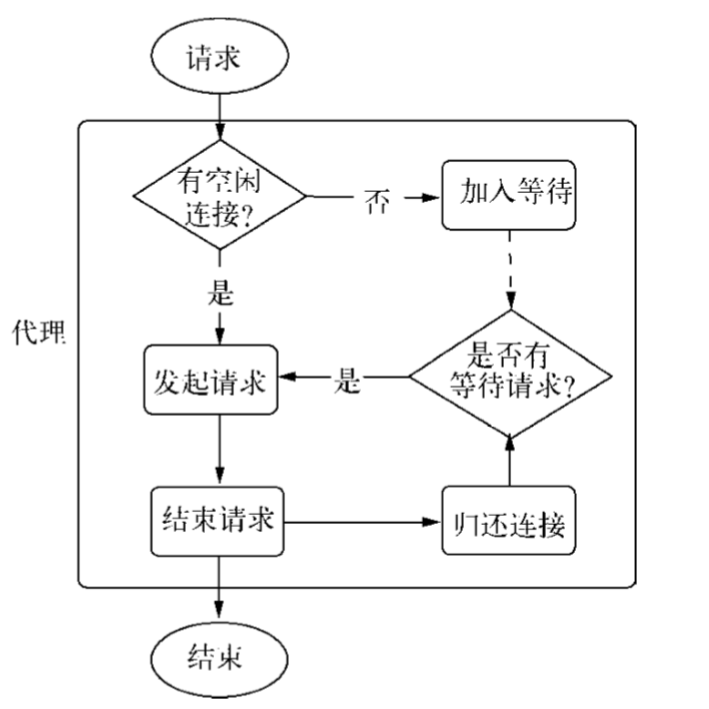
clientError:连接的客户端触发error事件，错误传递到服务器端时触发。

http模块提供了底层API-http.request(options,connect)用于构造HTTP客户端。options决定HTTP请求头中的内容。报文体的内容由请求对象的write和end实现，write向连接中写入数据，通过end方法告知报文结束。

ClientRequest:HTTP客户端响应对象，事件为response。与服务器端类似。

通过ClientRequest对象对同一个服务器端发起的HTTP请求默认可以创建5个，实质是一个连接池。

这意味着如果发起10个连接请求，实质只有5个处于并发态，其余在等待，代理示意图：



HTTP客户端事件与服务器端类似

## WebSocket服务

WebSocket实现了客户端与服务器端之间的长连接，而Node事件驱动的方式十分擅长与大量的客户端保持高并发连接。

优点：

* 1. 客户端与服务器只建立一个TCP连接
  2. 服务器端可以推送数据到客户端
  3. 更加轻量级的协议头，减少数据传送量。

HTTP握手时让服务器端切换到WebSocket协议，完成后开始WebSocket的数据帧协议实现数据交换：当客户端调用send发送数据，服务器端触发onmessage；当服务器端调用send发送数据，客户端的onmessage触发。

## 网络服务安全

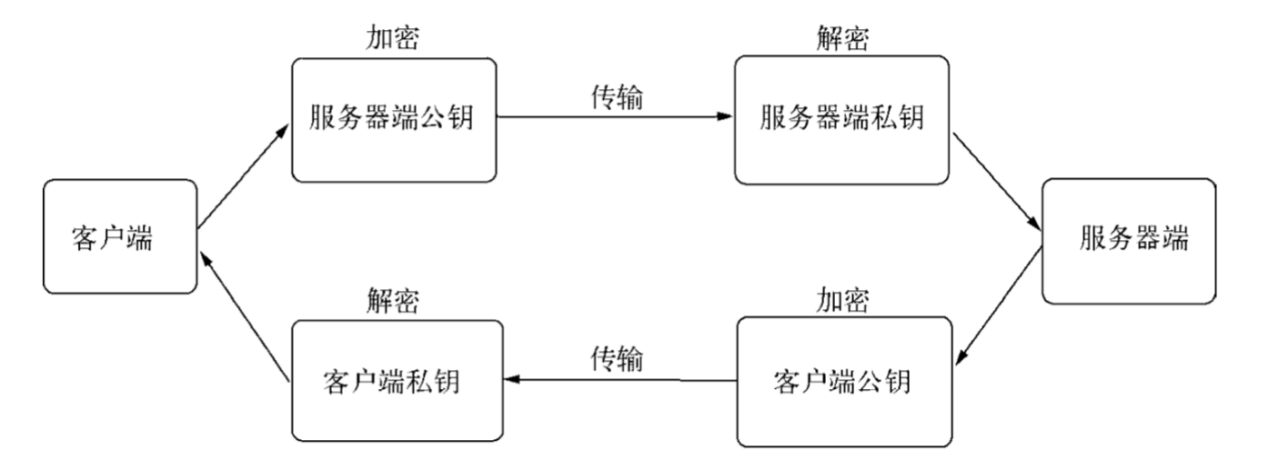
在传输层完成数据的加密解密，对于应用层的HTTP等还是透明的。

Node提供了3个模块用于网络安全：crypto,tls,https

tls模块提供了与net模块类似的功能，区别在于它建立在TLS/SSL加密的TCP连接上。

https和http模块接口完全一致，区别在于它建立在TLS/SSL加密的TCP连接上。

TLS/SSL：



## cookie

HTTP是个无状态协议，实际业务需要状态，如何标识和认证一个用户，使用Cookie。

处理步骤：

服务器向客户端发送cookie -> 浏览器保存cookie -> 之后每次浏览器都会将cookie发送给服务器

客户端发送的cookie在请求报文的cookie字段

Cookie可选选项：

path:表示这个Cookie影响到的路径，当前访问的路径不匹配时，浏览器不发送这个Cookie

Expires和Max-Age：告知浏览器这个Cookie何时过期，若不设置则关闭浏览器就丢失该Cookie

HttpOnly:告知浏览器不允许通过脚本document.cookie去更改这个Cookie值

Secure:当Secure值为true时，只有在HTTPS才有效，在HTTP中是无效的。

缺点：体积过大；cookie被允许在前后端修改，安全性差。

## Session

session数据只保留在服务器端，客户端无法修改。

session的有效期短，安全性高

## 缓存

由服务器和浏览器约定，大多只有GET请求才使用缓存。

浏览器请求某文件，本地没有就只能向服务器请求，然后该文件就会放置在本地的某个缓存文件中。第二次请求就基本可以使用本地文件，如果不确定是否可以使用，则会向服务器发送一次条件请求，若服务器响应304则允许使用本地文件，否则使用服务器的新文件。

条件请求可以用时间戳实现。替代条件请求可以是在服务器的响应头内加个Expires或Cache-Control头，浏览器根据该值进行缓存。一般用Cache-Control,本质是一个时间字符串，且以倒计时的方式，便于两端时间一致。

## 数据上传

单纯的HTTP头部报文无法携带大量的数据。

Node的http模块只会对HTTP报文的头部进行解析，然后触发request事件，如果请求中还带有内容部分则需要用户自行接收和解析。解析报头结束后，报文内容部分会通过data事件触发，然后用户以流的方式处理即可。

一般都是依靠报头中的Content-Type值判断是哪类数据并进行相应解析。

提交文件时表单中含有file类型的控件，以及需要指定特定的表单属性enctype。请求报文的报头也是特别的：

Content-Type:multipart/form-data;boundary=AaB03x

Content-Length:18231

boundary=AaB03x指定的是每部分内容的分界符，AaB03x是随机生成的一段字符串，报文体的内容将通过在它前面添加--进行分割，报文结束时在它前后都加上--表示结束。Content-Length必须是报文体的长度。

## MVC

Model:模型，数据的相关操作和封装

View:视图，视图的渲染

Controller:控制器，一组行为的集合

工作模式：

路由解析，根据URL 寻找到对应的控制器和行为

行为调用相关的模型，进行数据操作

数据操作结束后，调用视图和相关数据进行页面渲染，输出到客户端

## RESTful

同上

## 中间件

替代处理基础设施和业务逻辑之间的联系

让开发者关注在业务的开发

一般中间件要用next()来调下个中间件，中间件处理异常的方法是在next方法中添加err参数，并捕获中间件直接抛出的同步异常；异步异常则由中间件自己抛出。

## 页面渲染

* 1. 内容响应

客户端处理响应内容时，会根据报头Content-Type值来采取不同的处理方式。如当作纯文本处理或者当作HTML内容处理。这个值被称为MIME。

Content-Disposition：让客户端把响应内容当作附件下载而非即时浏览的内容。

* 1. 视图渲染

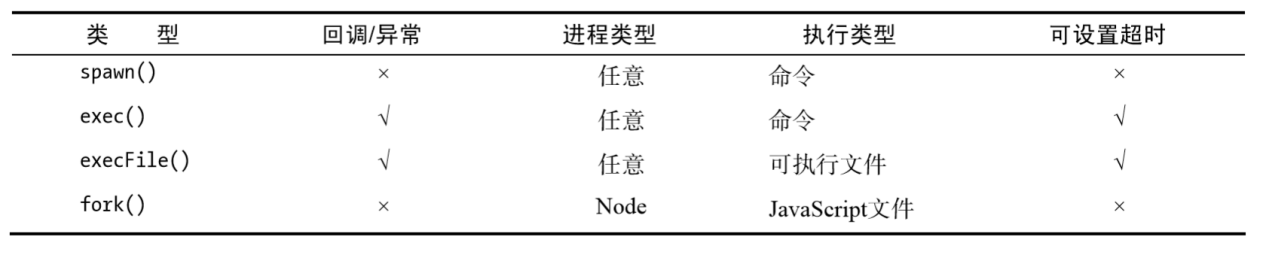
模板技术使得网页中的动态内容和静态内容变得不互相依赖。

模板本质上是拼接字符串。

为预防XSS漏洞，需要把能形成HTML的标签的字符转换成安全的字符，即为转义。

## 进程

Node提供了child\_process来提供进程的复制

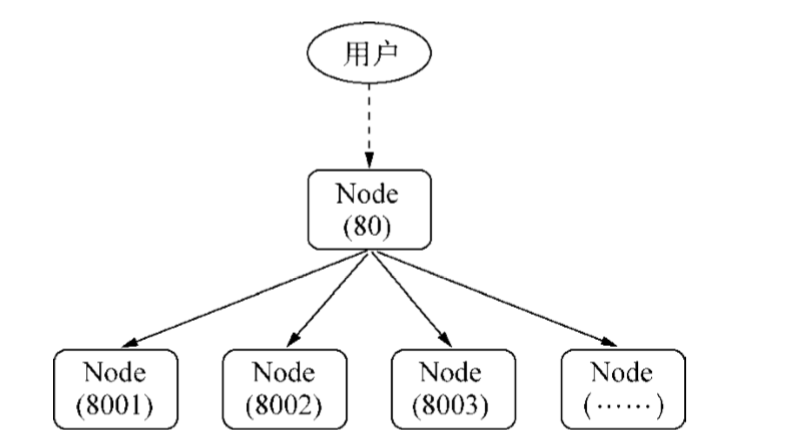


JS主线程和UI渲染共用同一个线程，长时间执行JS会造成UI渲染停滞。HTML5提出WwebWorker API,允许创建工作线程并在后台运行，使得一些阻塞较为严重的计算不影响主线程上的UI渲染。

父子进程之间用IPC通道通信，IPC通道在创建子进程之前就已经创建好了，创建完子进程后，会给子进程一个文件描述符去连接IPC，从而最终实现父子进程的连接。

windows下IPC通道由命名管道实现，是双向通信且不用通过网络层通信。

多个进程不能直接监听同一个端口。若要实现，则需要一个主进程监听一个主端口，对外则接收多个网络请求，再将这些网络请求分别代理到不同的端口的进程。



但是代理方式会比较浪费资源，所以后来发送句柄的方式取而代之。句柄是一种可以用来标识资源的引用，譬如标识一个服务器端对象，管道等。

之前问题中，可以使主进程接收到socket请求后，将这个socket直接发送给工作进程而无需再创建新的socket连接，句柄（文件描述符）在其中起关键作用。

Node进程之间只有消息传递，不会真正传递对象，传递对象的错觉是抽象封装的结果。

句柄总结：一个主机启动多个进程，让这些进程监听同一个端口号是不可能的。一个socket一个描述符只能用一个端口，而服务器的套接字文件描述符不同。Node为了解决这个问题，加了个进程间发送句柄的功能，句柄就是文件描述符，这样进程间的文件描述符就相同了，就可以监听相同端口，但在监听相同端口时，进程间只有一个能连接上，即进程服务是抢占式的。

Node的抢占式策略是轮叫调度，方式是由主进程接收连接，将其依次分发给工作进程。

## 单元测试

* 1. 断言

断言是单元测试中用来保证最小单元是否正常的检测方法。

断言用于检查程序在运行时是否满足期望。

Node中的assert模块用于断言。

检测方法：

ok(): 判断结果是否为真

equal(): 判断实际值与期望值是否相等

notEqual(): 判断实际值与期望值是否不相等

deepEqual(): 判断实际值与期望值是否深度相等（对象或数组元 素完全相等）

notDeepEqual(): 判断实际值与期望值是否不深度相等

strictEqual(): 判断实际值与期望值是否严格相等（相当于===）

notStrictEqual(): 判断实际值与期望值是否不严格相等

throws: 判断代码块是否抛出异常

doesNotThrow(): 判断代码块是否没有抛出异常

ifError(): 判断实际值是否为一个假值（null,undefined,0,’’,false）, 如果为真值则抛出异常

* 1. 测试框架

测试框架为测试服务，但本身不参与测试，主要用于管理测试用例和生成测试报告，提升测试用例的开发速度，提高测试用例的可维护性和可读性。

Node推荐用mocha模块

有TDD和BDD两种单元测试风格：

TDD指测试驱动开发，关注所有功能是否被正确实现，每一个功能都有对应的测试用例。TDD对测试用例的组织主要采用suite和test完成，suite实现多层级描述，测试用例用test。

BDD指行为驱动开发，关注整体行为是否符合预期，适合自顶向下的设计方式。BDD对测试用例的组织主要采用describe和it，describe可以描述多层级的结构，具体到测试用例时，用it。

# React

## 初识

React主要用于构建UI

## 元素渲染

用React开发时应用时一般只会定义一个根节点

React元素都是不可变的，当元素被创建之后，你是无法改变其内容或属性的。更新界面的唯一办法就是创建一个新的元素，然后将它传入ReactDOM.render（）方法。

## JSX

显示变量：<h1>{text}</h1>

显示字符串：<h1>{‘text’}</h1>

注：组件的render方法中只能返回个根节点,如果有多个并列的div，需要在外面包一个根div

html的class关键词在JSX中由className替代 for关键词由htmlFor替代

JSX中设置属性但没有设置值时 默认为true

没有设置属性时 默认值为false

属性赋值时 后面的值会把前面的值覆盖掉

希望使用自定义属性 可以使用data-

使用style属性方法：

style={{ color:’red’，fontSize:’1px’ }}

第一个{}是JSX语法 第二个为JS 使用小驼峰命名法使用

使用事件处理时也是小驼峰命名法使用：onClick={this.click}

## 迭代

迭代时每个项都要一个不同的key

## props state refs

props:父级传递给子级的数据 传入后就不能修改

state:组件本身的数据 随着使用者互动而改变

refs:用于绑定render()输出的任何组件上（当组件插入到DOM后， ref属性添加一个组件的引用于到this.refs，可以通过ref来获 取组件的引用 例：handleClick() {this.refs.value}）

注意：组件不是真实的DOM节点 而是存在于内存中的一种数据结构 叫做虚拟DOM，只有插入文档后才会变成真实的DOM。React中所有DOM变动，都先在虚拟DOM上发生，然后将实际发生变动的部分反映在真实DOM上，这种算法叫做DOM diff

如果要从组件获取真实DOM节点 需要用ref

## React Component(组件)生命周期

三种生命周期状态：

Mounting：已插入真实的DOM

Updating: 正在被重新渲染

Unmounting: 已移出真实的DOM

对应处理方法，will函数在进入状态之前调用，did函数在进入之后调用。常规五种，特殊两种：

1. Mounting
   1. componentWillMount()
   2. componentDidMount()
2. Updating
   1. componentWillReceiveProps(object nextProps):已载入组件收到新的参数时呼叫 （特殊）
   2. shouldComponentUpdate(object nextProps, object nextState):组件判断是否重新渲染时呼叫，起始不会呼叫除非呼叫forceUpdate() （特殊）
   3. componentWillUpdate(object nextProps, object nextState)
   4. componentDidUpdate(object prevProps, object prevState)
3. Unmounting

componentWillUnmount()

ajax非同步处理在componentDidMount

## 路由配置

已发现问题：

1. react路由分为V4.0前和V4.X（V4.0后） 教程一般是V4.0前 现在使用是V4.X
2. V4.X路由配置由react-router-dom实现
3. 官方文档很详细
4. activeClassName和activeStyle只有NavLink有

## ImmutableJS

用Imuutable定义时 就不能再修改 功能类似深拷贝

可以复制给其他变量 其他变量变化时 原变量不变

ImmutableJS定义了七种类型：

1. Map 类似于键值的对象 可参考ES6的Map
2. List 对应一般的数组,有序且可以有重复值
3. Set 没有顺序且不能重复的列表 可参考ES6的Set

4-7 (不常用)Stack OrderedMap OrderedSet Record

用七种类型定义时不支持数据嵌套(内部有JS对象，不会变为不可变数据 )

若要数据嵌套可用Immutable.fromJS()

从不可变数据生成JS：ImmutableData.toJS()

判断两个数据结构是否相等：Immutable.is A,B 或者ES6的PureRenderMixin

判断两个数据引用是否一致:A is B

## this.props.children

表示父组件的所有节点

没有节点值类型为undefined 一个节点值类型为object 多个节点值类型为array

## proptypes

验证组件实例属性是否符合要求

注V4.0前和V4.0后 检查方式不同

getDefaultProps可以设置组件属性的默认值

## 表单

表单值 要用回调函数并通过event.target.value读取

## Flux入门

一种软件架构

Flux将一个应用分为四个部分：

View：视图层

Action: 视图层发出的消息（click之类）

Dispatcher:用来接收Actions,并执行回调函数，每个APP只有一个

Store：用来存放应用的状态，一旦发生变动，就提醒Views要更新页面，数据也被存放在这儿.即负责操作和存储数据资料并为View监听（listen）。一个APP通常有多个store负责存放业务逻辑，不同业务会有不同store。

Flux的特点是数据的单向流动 ,任何相邻的部分都不会发生数据的双向流动：View->Action->Dispatcher->Store->View

## Redux入门（非必须情况下 只了解）

Redux是将Flux与函数式编程结合一起

大多时候并不需要Redux。即如果你不知道是否需要 Redux，那就是不需要它。例如UI层很简单，没有很多互动就没有必要使用。

多数据源，多交互时候需要用到Redux。

设计思想：

Web应用是一个状态机，视图和状态是一一对应的。

所有状态都保存在一个对象里