Table of Contents

前端总结

HTML

HTML参考手册

概念

Doctype

`<!DOCTYPE>` 声明位于位于 HTML 文档中的第一行，处于 `<html>` 标签之前。

告知浏览器的解析器用什么文档标准解析这个文档。DOCTYPE 不存在或格式不正确会导致文档以兼容模式呈现。

在 HTML 4.01 中，<!DOCTYPE> 声明引用 DTD，因为 HTML 4.01 基于 SGML。DTD 规定了标记语言的规则，这样浏览器才能正确地呈现内容。

HTML5 不基于 SGML，所以不需要引用 DTD。

标准模式 && 兼容模式

标准模式的排版和 JS 运作模式都是以该浏览器支持的最高标准运行。

在兼容模式中，页面以宽松的向后兼容的方式显示,模拟老式浏览器的行为以防止站点无法工作。

内联元素

1. 和其他元素都在一行上；
2. 高，行高及外边距和内边距部分可改变；
3. 宽度只与内容有关；
4. 行内元素只能容纳文本或者其他行内元素。
5. 不可以设置宽高，其宽度随着内容增加，高度随字体大小而改变

> 内联元素可以设置外边界，但是外边界不对上下起作用，只能对左右起作用，也可以设置内边界，但是内边界在ie6中不对上下起作用，只能对左右起作用

常用的内联元素：a , b , br , em , font , img , input , label , select , small , big , span , textarea

块级元素

1. 总是在新行上开始，占据一整行；
2. 高度，行高以及外边距和内边距都可控制；
3. 宽带始终是与浏览器宽度一样，与内容无关；
4. 它可以容纳内联元素和其他块元素。

> 注意：h1--h4、p 标签都是块级元素

空元素

例如： br、hr、等

DOM事件

事件方法名	事件内容
onabort	图像的加载被中断。
onblur	元素失去焦点。
onchange	域的内容被改变。
onclick	当用户点击某个对象时调用的事件句柄。
ondblclick	当用户双击某个对象时调用的事件句柄。
onerror	在加载文档或图像时发生错误。
onfocus	元素获得焦点。

onkeydown	某个键盘按键被按下。
onkeypress	某个键盘按键被按下并松开。
onkeyup	某个键盘按键被松开。
onload	一张页面或一幅图像完成加载。
onmousedown	鼠标按钮被按下。
onmousemove	鼠标被移动。
onmouseout	鼠标从某元素移开。
onmouseover	鼠标移到某元素之上。
onmouseup	鼠标按键被松开。
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。

HTML 语义化

用正确的标签做正确的事情。

html 语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析;

即使在没有样式 CSS 情况下也以一种文档格式显示，并且是容易阅读的;

搜索引擎的爬虫也依赖于 HTML 标记来确定上下文和各个关键字的权重，利于 SEO;

使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

颜色

HTML 颜色由一个十六进制符号来定义，这个符号由红色、绿色和蓝色的值组成。

例如：#000000，rgb(0,0,0)。

另外，rgb(0,0,0,0.5)，这里的 0.5 表示透明度，取值为 0~1。

HTML 标签

主要是介绍了一些常用的、易混淆的 HMTL 标签。

label

为输入字段做相关标记。

例如：

```
<label for="male">Male</label>
<input type="radio" name="sex" id="male" />
<br />
<label for="female">Female</label>
<input type="radio" name="sex" id="female" />
```

这时候，点击 label 标签内容，后边的 input 会自动获得焦点。

> 注意 label 的 for 值要与后边 input 的 id 相同。class 不可以。

iframe

iframe 缺点

- iframe 会阻塞主页面的 Onload 事件。
- 搜索引擎的检索程序无法解读这种页面，不利于 SEO。
- iframe 和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。

建议

使用iframe之前需要考虑上边两个缺点。如果需要使用iframe，最好是通过 javascript，动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题。

iframeborder

设置 iframe 的边框。

br

在不产生新段落得情况下进行换行。一个 br 产生一个换行。

不常见格式化标签

标签	描述
`<b>`	定义粗体文本
`<em>`	定义着重文字
`<i>`	定义斜体字
`<small>`	定义小号字
`<strong>`	定义加重语气
`<sub>`	定义下标字
`<sup>`	定义上标字
`<ins>`	定义插入字
`<del>`	定义删除字

标签	描述

`<code>`	定义计算机代码
`<kbd>`	定义键盘码
`<samp>`	定义计算机代码样本
`<var>`	定义变量
`<pre>`	定义预格式文本

标签	描述
`<abbr>`	定义缩写
`<address>`	定义地址
`<bdo>`	定义文字方向
`<blockquote>`	定义长的引用
`<q>`	定义短的引用语
`<cite>`	定义引用、引证
`<dfn>`	定义一个定义项目

head

title

定义文档的标题，一般显示在浏览器的标签中。

base

定义页面中所有链接默认的链接地址。

```
<head>
<base href="http://www.runoob.com/images/" target="_blank">
</head>
```

meta

用来描述 HTML 文档的描述，关键词，作者，字符集等。

```
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<meta name="description" content="免费在线教程">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="runoob">
</head>
```

图标

```
<link rel="shortcup icon" href="img_url">
```

图片映射

实现图片的部分区域可点击。

```
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">

<map name="planetmap">
    <!-- x1,y1 左上坐标，x2,y2 右下坐标 -->
    <area shape="rect" coords="x1,y1,x2,y2" alt="Sun" href="sun.htm">
    <!-- 圆，x1，y1，半径 -->
    <area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm">
    <area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm">
    <!-- 多边形，每个点坐标（x1，y1）... -->
```

```
</map>
```

> 实际测试谷歌浏览器中，usemap="@planetmap" 中带有 # 与不带有 # 同样有效果，但是 map 只能使用 map，使用 id 和 class 不可以。

运行结果

table

表格标签。

tr：一行。

th：表头，单元格。

td：表体，单元格。

caption：定义表格标题。

colgroup：定义表格列的组。

col：定义表格列的属性。

thead：定义表格的页眉。

tbody：定义表格的主体。

tfoot：定义表格的页脚。

ul

ul + li，无序列表。

ol + li，有序列表。

dl + (dt + dd)，自定义列表，dt 为列表项，dd 为列表项描述。

input

使用 type 定义 input 的种类，例如：

Type 的值	**input** 的种类
text	输入框
password	密码字段
radio	单选按钮（对应的常见的属性值有 value、name、checked）
checkbox	多选按钮（属性值有 value、name、checked）
submit	提交按钮
reset	重置按钮

noscript

noscript 标签提供无法使用脚本时的替代内容，比方在浏览器禁用脚本时，或浏览器不支持客户端脚本时。

noscript 元素可包含普通 HTML 页面的 body 元素中能够找到的所有元素。

只有在浏览器不支持脚本或者禁用脚本时，才会显示

```
<script>
    document.write("Hello World!")
</script>

<noscript>抱歉，你的浏览器不支持 JavaScript!</noscript>
<!-- 此时浏览器如果不支持 javascript，网页就会显示 “抱歉，你的浏览器u支持 JavaScript！” -->
```

HTML 属性

title

title 属性规定关于 head 元素的额外信息。

浏览器的标签上会显示当前网页的 title 值，同时鼠标移到此标签上时，会显示 title 中的文本。

> 注意与 h1 区别，h1 是 HTML 中的一个标签，在 body 中，而 title 是一个 head 中的一个元素。

href 和 src 的区别

- href 是 Hypertext Reference 的缩写，表示超文本引用。用来建立当前元素和文档之间的链接。带有此属性的常见的标签有：link，a 等。例如 CSS 标签，浏览器会识别该文档为 css 文档，并行下载该文档，并且不会停止对当前文档的处理。
- src 是 source 的缩写，src 的内容是页面必不可少的一部分，是引入。带有此属性的常见的标签有：img、script、iframe 等。 当浏览器解析到该元素时，会暂停浏览器的渲染，知道该资源加载完毕。这也是将js脚本放在底部而不是头部得原因。

HTML5

HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

> 中文网页需要使用 `<mate charset="utf-8">` 进行声明。

HTML5 新特性

1. 绘画 canvas。
2. 用于媒介回放的 video 和 audio 元素。
3. 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失。
4. sessionStorage 的数据在浏览器关闭后自动删除。
5. 语意化更好的内容元素，比如 article、footer、header、nav、section。
6. 表单控件，calendar、date、time、email、url、search。

7.新的技术webworker, websocket, Geolocation。

HTML5 移除的元素

1. 纯表现的元素：basefont，big，center，font, s，strike，tt，u。
2. 对可用性产生负面影响的元素：frame，frameset，noframes。

HTML5 新元素

header, section, footer, aside, nav, main, article, figure等。

> 这些新元素都是块级元素。 更多新元素参考：http://www.runoob.com/html/html5-new-element.html

添加自定义元素

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>为 HTML 添加新元素</title>
<script>
    document.createElement("myHero")
</script>
<style>
    myHero {
        display: block;
        background-color: #ddd;
        padding: 50px;
        font-size: 30px;
    }
</style>
</head>
<body>
    <myHero>我的第一个新元素</myHero>
</body>
</html>
```

canvas

> canvas参考手册

HTML5 canvas 元素用于图形的绘制，通过脚本 (通常是 JavaScript)来完成.

canvas 标签只是图形容器，您必须使用脚本来绘制图形。

你可以通过多种方法使用 canvas 绘制路径,盒、圆、字符以及添加图像。

例如：

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

渐变

createLinearGradient(x,y,x1,y1) -> 创建线条渐变

createRadialGradient(x,y,r,x1,y1,r1) -> 创建一个径向/圆渐变

文字

font -> 定义字体和大小

fillText(text,x,y) -> 在 canvas 上绘制实心的文本

strokeText(text,x,y) -> 在 canvas 上绘制空心的文本font - 定义字体

fillText(text,x,y) -> 在 canvas 上绘制实心的文本

strokeText(text,x,y) -> 在 canvas 上绘制空心的文本

路径（直线）

moveTo(x,y) 定义线条开始坐标。

lineTo(x,y) 定义线条结束坐标。

图像

drawImage(image,x,y)

SVG

SVG 指可伸缩矢量图形 (Scalable Vector Graphics)。

SVG 用于定义用于网络的基于矢量的图形。

SVG 使用 XML 格式定义图形。

SVG 图像在放大或改变尺寸的情况下其图形质量不会有损失。

SVG 是万维网联盟的标准。

SVG优势

与其他图像格式相比（比如 JPEG 和 GIF），使用 SVG 的优势在于：

- SVG 图像可通过文本编辑器来创建和修改
- SVG 图像可被搜索、索引、脚本化或压缩
- SVG 是可伸缩的
- SVG 图像可在任何的分辨率下被高质量地打印
- SVG 可在图像质量不下降的情况下被放大

svg 标签

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">
  <polygon points="100,10 40,180 190,60 10,60 160,180"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;">
</svg>
```

SVG 与 Canvas两者间的区别

SVG 是一种使用 XML 描述 2D 图形的语言。

Canvas 通过 JavaScript 来绘制 2D 图形。

SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。您可以为某个元素附加 JavaScript 事件处理器。

在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

Canvas 是逐像素进行渲染的。在 canvas 中，一旦图形被绘制完成，它就不会继续得到浏览器的关注。如果其位置发生变化，那么整个场景也需要重新绘制，包括任何或许已被图形覆盖的对象。

MathML

> 参考文章

拖放（drag & drop）

> 参考文章

更多类型的 input

> 参考文章

Page Visibility API

document.hidden

返回表示页面是否隐藏的布尔值。

document.visibilityState

有下面 4 个可能状态的值：

1. hidden：页面在后台标签页中或者浏览器最小化
2. visible：页面在前台标签页中
3. prerender：页面在屏幕外执行预渲染处理 document.hidden 的值为 true
4. unloaded：页面正在从内存中卸载

Visibilitychange事件

当文档从可见变为不可见或者从不可见变为可见时，会触发该事件。

这样，我们可以监听 Visibilitychange 事件，当该事件触发时，获取 document.hidden 的值，根据该值进行页面一些事件的处理。

离线存储

在线的情况下，浏览器发现html头部有manifest属性，它会请求manifest文件，如果是第一次访问app，那么浏览器就会根据manifest文件的内容下载相应的资源并且进行离线存储。如果已经访问过app并且资源已经离线存储了，那么浏览器就会使用离线的资源加载页面，然后浏览器会对比新的manifest文件与旧的manifest文件，如果文件没有发生改变，就不做任何操作，如果文件改变了，那么就会重新下载文件中的资源并进行离线存储。

离线的情况下，浏览器就直接使用离线存储的资源。

HTML5 兼容

1. IE8/IE7/IE6支持通过 document.createElement 方法产生的标签，

   可以利用这一特性让这些浏览器支持HTML5新标签，浏览器支持新标签后，还需要添加标签默认的样式。

2. 直接使用成熟的框架、比如 html5shim。

例如：

```
<!--[if lt IE 9]>
    <script>src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>
<![endif]-->
```

> 以上代码是一个注释，作用是在 IE 浏览器的版本小于 IE9 时将读取 html5.js 文件，并解析它。另外，以上代码必须放在 head 中。

如何区分**HTML5**

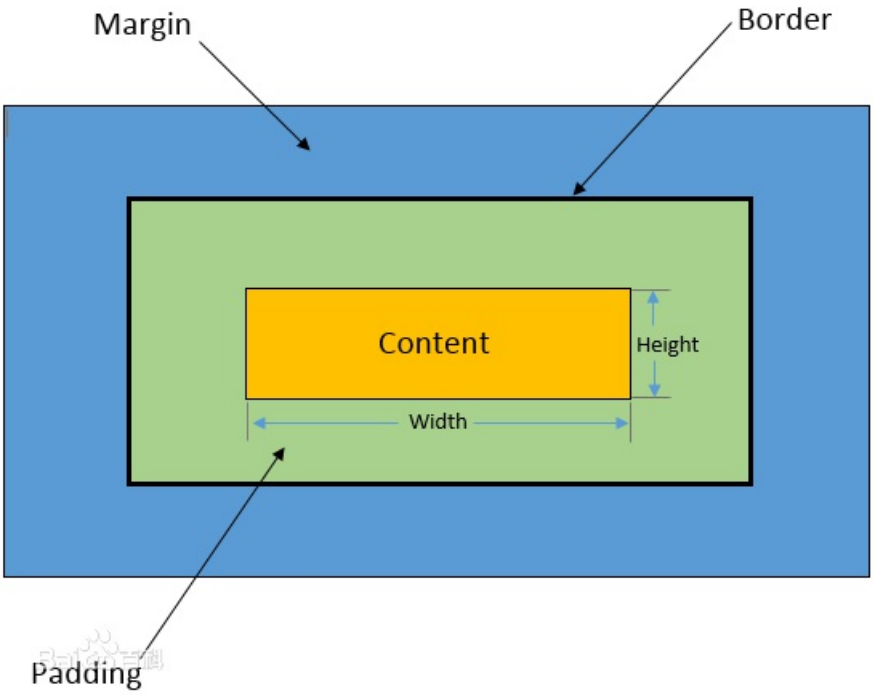
- DOCTYPE声明
- 新增的结构元素
- 功能元素

CSS

相关概念

盒子模型

1. 盒子模型中主要的属性有：
   - 内容(content)
   - 内边距(padding)
   - 边框(border)
   - 外边距(margin)
   - 宽度(width)
   - 高度(height)



1. 设置元素的盒子模型显示方式

可以通过 box-sizing(CSS3) 属性设置元素显示的盒子模型，可以选择的值有 content-box(default)、border-box、inherit。

```
- content-box, 设置的宽高应用于元素的内容
- border-box, 设置的宽高决定元素的边框和
- inherit, 继承父元素
```

页面引入样式

link && @import

1. link属于XHTML标签，除了加载CSS外，还能用于定义RSS,定义rel连接属性等作用；而@import是CSS提供的，只能用于加载CSS;
2. 页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载;
3. import是CSS2.1 提出的，只在IE5以上才能被识别，而link是XHTML标签，无兼容问题;

CSS选择器

- 通配符选择器（*）
- 类选择器

- ID选择器
- 属性选择器（*[name]）
- 后代选择器(.classname .otherclassname)
- 子元素选择器（h1>strong)
- 相邻兄弟选择器(.classname + .classname)
- 伪类（button:active, p:first-child, q:lang(no),focus,hover,link,vistied,)
- 伪元素（first-line, first-letter, :before, :after,)

> 1.后代选择器和子元素选择器的区别是，后代选择器的两个元素的层次间隔可以是任意的，而子元素选择器只能选择前边元素的子元素。
> 2.相邻兄弟选择器，选择的是后边的元素。

CSS属性

Position

- static: 默认值，没有定位，元素出现在正常流中，（忽略top、bottom、left、right、z-index）
- relative: 生成相对定位的元素，可以通过top、bottom、left、right相对于正常位置进行定位，可以通过z-index分层。正常流中仍然有relative的位置，只是进行了偏移。定位总是相对于最近的父元素，无论其是何种定位方式。
- absolute: 生成绝对定位的元素，相对于static定位外的第一个元素进行定位，元素的位置通过"left","top","right"以及"bottom"属性进行规定。可通过z-index进行层次分级。正常流中不再有absolute的位置，相当于元素悬浮。离其最近的absolute或者relative层，没有的话就以body定位。margin也符号上述规则。
- fixed： 生成绝对定位的元素，相对于浏览器窗口进行定位。元素的位置通过"left","top","right"以及"bottom"属性进行规定。可通过z-index进行层次分级。

border style

dotted	solid	double	dashed
点状	实线	双线	虚线

background-attachment

滚动视觉差。可以实现页面背景图固定的效果。

可能的属性值	效果
local	类似scroll
scroll	背景随内容滚动而滚动
fixed	背景固定于可视区域，不随内容的滚动而滚动

> 需要注意的是，此属性以 background-image 属性为前提。

1. local

背景图相对于元素内容固定。

1. scroll

默认值，背景图相对于元素固定，背景随页面滚动而滚动，可以理解为此时，元素内容于背景是绑定的。

1. fixed

背景图固定于可视区域，当前元素只要在可视区域内，则背景都是此元素的背景，不会随元素内容的滚动而滚动，知道遇到下一个元素的背景。

> 此时背景图的 z-index 相对较低，如果可视区域内同时存在了其他元素，其他元素的背景会遮盖住此元素的背景图。

visibility

用于设置 HTML 元素的可见性。

可选值	作用
visible	默认值。元素是可见的
hidden	元素是不可见的

collapse	当在表格元素中使用时，此值可删除一行或一列，但是它不会影响表格的布局。被行或列占据的空间会留给其他内容使用。如果此值被用在其他的元素上，会呈现为 "hidden"
inherit	规定应该从父元素继承 visibility 属性的值

> 注意与 display 区别，visibility 只是使元素不可见，该元素仍会占用空间。

CSS三大机制

1、特殊性

`特殊性`：一个元素可以被多少个规则设置，但是最终只有一个规则会起作用，那么该规则的特殊性最高，特殊性即CSS优先级。

2、继承

`CSS继承`：是从一个元素向其后代元素传递属性值所采用的机制。

- 继承是从父元素到子元素
- 所有关于文字图片大小样式的属性可以继承(例如：letter-spacing、word-spacing、white-space、line-height、color、font

无继承性的属性

1. display：规定元素应该生成的框的类型
2. 文本属性：

属性	含义
vertical-align	垂直文本对齐
text-decoration	规定添加到文本的装饰
text-shadow	文本阴影效果
white-space	空白符的处理
unicode-bidi	设置文本的方向

1. 盒子模型的属性：width、height、margin及margin相关、border及boder相关、padding及padding相关
2. 背景属性：background（background-color、background-image、background-repeat、background-position、background-attachment）
3. 定位属性：float、clear、position、top、right、bottom、left、min-width、min-height、max-width、max-height、overflow、clip、z-index
4. 生成内容属性：content、counter-reset、counter-increment
5. 轮廓样式属性：outline-style、outline-width、outline-color、outline
6. 页面样式属性：size、page-break-before、page-break-after
7. 声音样式属性：pause-before、pause-after、pause、cue-before、cue-after、cue、play-during

有继承性的属性

1. 字体系列属性：font及font相关、
2. 文本系列属性：text-indent：文本缩进、text-align、line-height、word-spacing：即字间隔、letter-spacing：字符间距、text-transform：控制文本大小写、direction：规定文本的书写方向、color
3. 元素可见性：visibility
4. 表格布局属性：caption-side、border-collapse、border-spacing、empty-cells、table-layout
5. 列表布局属性：list-style-type、list-style-image、list-style-position、list-style
6. 生成内容属性：quotes
7. 光标属性：cursor
8. 页面样式属性：page、page-break-inside、windows、orphans
9. 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

所有元素可以继承的属性

1. 元素可见性：visibility
2. 光标属性：cursor

内联元素可以继承的属性

1. 字体系列属性

2. 除text-indent、text-align之外的文本系列属性

块级元素可以继承的属性

1. text-indent、text-align

3、层叠

`层叠`：确定应当向一个元素应用哪些值时，浏览器不仅要考虑继承，还要考虑声明的特殊性，另外需要考虑声明本身的来源。这个过程就称为层叠。

- 规则的权重（!important），加了权重的优先级最高
- 当权重相同的时候，会比较规则的特殊性，根据前面的优先级计算规则决定哪条规则起作用
- 当特殊性值也一样的时候，css规则会按顺序排序，后声明的规则优先级高，

> 后声明的规则优先级高

三、css优先级

1、相关概念

`声明`：CSS选择器后边大括号里的样式叫做声明。
`声明块`： CSS选择器整个大括号叫做声明块。

2、优先级的计算

选择器的特殊值用4位标识，0，0，0，0，不同选择器对特殊值的影响如下：

选择器	影响
通配选择器*对特殊性没有贡献	即0,0,0,0
元素和伪元素	加0,0,0,1
类选择器、属性选择器或伪类	加0,0,1,0
ID选择器的特殊性值	加0,1,0,0
!important（权重）	它没有特殊性值，但它的优先级是最高的，可以认为1,0,0,0,0
继承的属性	没有特殊值,0,0,0,0

四、CSS实例

1. 纯CSS实现三角形

```
.triangle{
    width:0;
    height:0;
    border-width:20px;
    border-style:solid dashed dashed dashed;
    border-color:#e66161 transparent transparent transparent;
}
```

> 利用了 CSS 中的 border 属性。

2. 对话框下边的三角形

利用旋转操作将正方形旋转。

3. 不定高元素垂直居中

- 使用：vertical-align:(对内部的文字起作用，内部不能是 div)
    - 外层 div，display: table-cell => 对 div 中的文字设置垂直居中，。
    - 外层 div，display: inline => 可以通过设置 line-height 改变 div 的高度。

    - 外层 div, display: inline-block => 设置 line-height 和 height 相同的时候，vertical-align 作用才会显现出。
- 外层 div设置为 relative，内层div 设置为 left:50%, top:50%, absolute，transform：translate(-50%, -50%), 或者使用 transform: translate3d(-50%, -50%, 0) 一样道理。
- 外层 div，display: -webkit-box; -webkit-box-algin: center; -webkit-bok-pack: center;（外层 div 的内容可以是文字，也可以是div）

  > -webkit-box-glign 只有 chrome,oprate 支持)

CSS3

CSS3 完全向后兼容，因此升级到 CSS3 的代价是非常小的。

> 3CSCHOOL CSS3

简介

CSS3 被划分为模块。

其中最重要的 CSS3 模块包括：

- 选择器
- 框模型
- 背景和边框
- 文本效果
- 2D/3D 转换
- 动画
- 多列布局
- 用户界面

边框

通过 CSS3，可以方便的创建圆角边框，向矩形添加阴影，使用图片来绘制边框。

- border-radius，创建边框圆角
- border-shadow，设置边框阴影。 例如：

```
div {
    box-shadow: 10px 10px 5px #888888;
}
```

- border-img，使用图片创建边框。 例如：

```
div {
    border-image:url(border.png) 30 30 round;
}
```

flex-box 布局

flex-box(FlexibleBox/CSS3弹性布局) 是 CSS3中的一种新的布局模式，是可以自动调整子元素的高和宽，来很好的填充任何不同屏幕大小的显示设备中的可用显示空间。收缩内容防止内容溢出，确保元素拥有恰当的行为的布局方式，例如：对齐方式，排列方向，排列顺序。

> 参考文章

基础概念

主轴（main axis），交叉轴（cross axis）

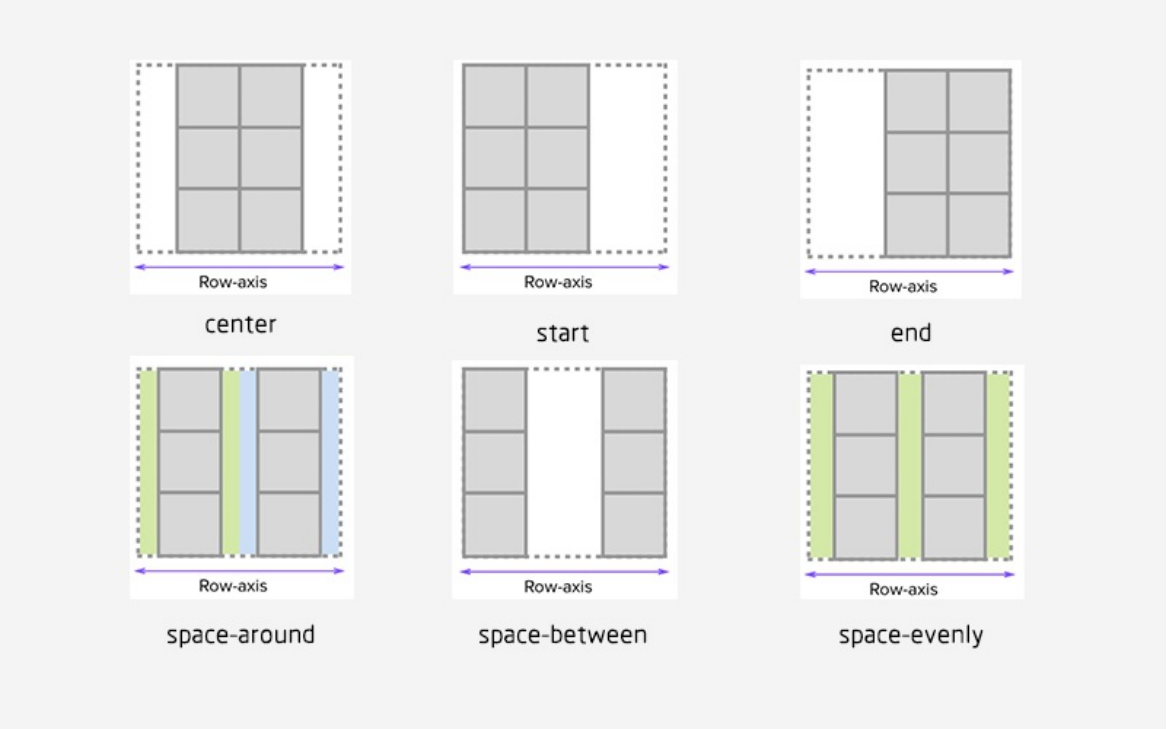
主轴起点为 main start，终点为 main end。交叉轴起点为 cross start，终点为 cross end。

> 主轴不一定水平，取决于 flex-direction

使用方法，外层容器 display 属性设置为 flex，使用 flex-direction 设置排列方向。

其他属性：

1. justify-content，可选值：flex-end，center，flex-end，space-evenly，space-between，space-around。(主轴方向)



2. align-items，可选值: start, end, center。（交叉轴）
3. align-self: 可选值：flex-start，center，flex-end。（应用于子元素，设置自身的对齐位置）
4. flex-wrap: wrap（设置 flex 可以多行排列，溢出元素即换行排列），wrap-reverse（多行排列，从末尾开始），
5. align-content（多行排列时在交叉轴上的排列方式），可选值：flex-start，flex-end，center，space-between，space-around ，space-evenly 和 stretch（默认）。
6. flex-grow（设置容器空间剩余时不同元素拉伸占剩余空间的比例，默认为1），值为数字。
7. flex-shrink（设置需要收缩时不同元素的比例），值为数字。
8. flex-basis（设置元素初始大小，默认值为 auto），可选值为 px，百分比。
9. flex: flex-grow-value flex-shrink-value flex-basis-value

属性

transform

旋转：transform：rotate(7deg);

> css中的相关度数：① deg：度数，总共360度; ② grad：梯度，一个圆400梯度； ③ rad：弧度，Π； ④ turn：圈，几圈。

less

1、特性

- 变量、Mixin、函数等特性
- lessc 命令进行编译
- 运行在node.js端或者浏览器端，浏览器加载less.js

sass

1、特性

- 变量，嵌套，混合
- sass命令进行编译

JavaScript

概念

AJAX

广义上的 AJAX 是指 XMLHttpRequest，也就是 ECMAScript 中用于网络交互的操作。

现在的 AJAX 大多数指的是 JQuery 中的 AJAX 函数。

JQuery 中的 AJAX 封装了大量网络交互操作，使其更加方便快捷。

this 关键字

首先介绍一下函数不同的调用方法：

普通函数调用；作为方法来调用；作为构造函数来调用；使用apply/call方法来调用；Function.prototype.bind方法；es6箭头函数。

> 谁调用了方法，this 就指向谁。

普通函数调用

```
function person() {
  this.name = "xiaoming";
  console.log(this);
  console.log(this.name);
}

person(); //print window, xiaoming
```

在这段代码中，person() 作为普通函数调用，实际上 person 是作为全局对象 window 的一个方法来进行调用的，即 window.person()。

所以在这里，是 window 调用了 person 方法，那么 person() 中的 this 就指向 window，并且 window 还有了一个属性 name，值为 xiaoming。

同样的：

```
var name = "xiaoming"
function person() {
  console.log(this);
  console.log(this.name);
}

person(); // print window, xiaoming
```

和上边代码的结果是相同的，因为 this 指向 window， 所以 this.name 的值还是 xiaomimg。

作为方法来调用

```
var name = "xiao hong";
var person = {
  name: "xiao ming",
  printName: function() {
    console.log(this.name);
  },
}

// print xiao ming
// person 对象调用 printName() 方法，所以 this 指向 person 对象，所以打印 xiao ming。
person.printName();

var printName = person.showName;  // 注意没有 （）

// print xiao hong
// 将 person 类中的方法 printName() 赋给了 printName 变量，当直接调用 printName() 的时候，this 指向的是 window，所以打印 xiao hong。
printName();
```

作为构造函数来调用

```
function Person(name) {
  this.name = name;
```

```
}

var person1 = Person("xiao ming");
console.log(person1.name);  // error:undefine
// 因为没有使用 new 操作，等同于 window 调用了 Person() 方法，因此 this 指向的是 window对象。
console.log(window.name); // xiao ming

var person2 = new Person("xiao hong")
console.log(person2.name);  // xiao hong
console.log(window.name); // xiao ming
```

下边讲解一下 new 操作：

```
function Person(name) {
  var o = {},
  o.__proto__ = Person.prototype; // 原型继承
  Person.call(o, name);
  return o;
}
// 注意，此处为伪代码，并不能实际运行，如此写是为了让读者更明白 new 操作符的作用。
```

call/apply 方法的调用

call、apply 方法的作用是改变 this 的作用域。

```
var name = "xiao hong";
var Person = {
  name: "xiao ming",
  printName: function() {
    console.log(this.name);
  }
}

Person.printName(); // xiao ming
Persin.printName.call();  // xiao hong
```

箭头函数**(ES6)**

箭头函数的 this 指向固定化，始终指向外部对象，因为箭头函数没有 this，因此它自身不能进行 new 操作实例化，同时也不能使用 call、apply、bind 来改变 this 的指向。

操作符

== && ===

例：value1 == value2 ， value3 === value4，

== ：

1. 如果两个值类型相同，再进行三个等号(===)的比较。
2. 如果两个值类型不同，也有可能相等，需根据以下规则进行类型转换在比较：
    i. 如果一个是null，一个是undefined，那么相等。
    ii. 如果一个是字符串，一个是数值，把字符串转换成数值之后再进行比较。

===：

1. 如果类型不同，则一定不相等。
2. 如果两个都是数值并且是同一个值，那么相等。
3. 如果一个是数值一个是 NaN，那么不相等。
4. 如果都是true 或者 false 相等。
5. 如果都是 null 或者 undefined，那么相等。注意，null == undefined => true, null === undefined => false.

总结：

1. string、number 等基础类型，== 与 === 的区别在于，== 会转换类型再进行比较，=== 不会转换类型比较。
2. array、object 等高级类型，== 与 === 没有区别。

3. 基础类型与高级类型比较，区别在于，== 将高级转换为基础类型，再进行比较，=== 因为类型不同，直接返回false

equals

JavaScript 中字符串没有 equals 方法，但是可以自己实现。

函数

ready() && onload()

- document.ready()

  是DOM结构绘制完成之后即可执行，不需要相关联资源下载完成，例如图片DOM结构绘制完成，而图片可能没有下载完成，其函数同样会执行。

- document.onload()

  相关资源都加载完毕之后，执行，一个页面中只能有一个onload

- load函数

  常用在图片的加载，比如 $('idName').load()，load同样是资源全部加载完成之后才会执行。

> JavaScript 没有 document.ready()，但是有 document.readyState，返回当前页面情况。

apply && call && bind

相同点：

- 都是用来改变函数的this对象的指向的
- 第一个参数都是this要指向的对象
- 都可以继续传递参数

```
var xb = {
    name: '小冰',
    gender: '女',
    say: function(){
    alert(this.name + ',' + this.gender);
    }
}
var other = {
    name: '小东',
    gender: '男',
}

xb.say(); ===> 小冰，女
xb.say.call(other); ==> 小东，男
xb.say.apply(other);
xb.say.bind(other)();
```

三个方法的不同点

- `call` 和 `apply` 方法，都是对函数的直接调用，但是 `bind()` 方法需要加上()来执行
- `call` 和 `apply` 方法传参的方法不同

  ```
  xb.say.call(other,'斯坦福','3')。
  xb.say.apply(other, 'sitanfu','third');
  ```

- `bind` 返回的是方法，所以需要后边加上()才能执行。

实现 **bind** 方法

```
if (!function() {}.bind) {
        Function.prototype.bind = function(context) {
        var self = this,
        args = Array.prototype.slice.call(arguments);
        return function() {
        return self.apply(context, args.slice(1));
```

```
        }
        };
}
```

Typeof

Typeof 把参数的类型信息作为字符串返回。

可能的值有：

- number
- string
- boolean
- object
- function
- undefined

> 注意：typeof(null) 返回object。

定时器

- `setTimeout(.... , time)` 在指定的时间（time） 后调用函数或者计算表达式。
- `setInterval(code,millisec,lang)` , lang传递给执行函数的其他参数。会在指定的周期（millisec）不断调用函数（code），直到clearInterval() 清除。

slice && splice （操作数组）

slice (截取数组，不改变原数组)

> 接收一个或两个参数,它可以创建一个由当前数组中的一项或多项组成的新数组,也就是说它不会修改原来数组的值。

用法:

- slice(para1),会截取从para1开始的到原数组最后的部分；
- slice（para1,para2）会截取原数组的从para1开始的para2-para1个数组。

> 注意：当两个参数中存在负数时，用原数组的长度加上两个负数的参数作为相应的参数来计算，并且得到的第一个参数必须小于第二个参数，否则返回一个空数组。

```
var a = [1,2,3,4,5,6];
a.slice(-2,-3)  //return []
a.slice(-3,-2)  //return [4]
```

splice (删除/替换 数组中的某几位，返回删除的数组)

用法：

- splice(para1,para2) : 删除数组中任意数量的项，从para1开始的para2项。

> 注意的是用splice删除数组中的值会直接将某几项从数组中完全删除，会导致数组length值的改变，这与delete的删除置为undefined是不一样的。

- splice(para1,para2,val1,val2...)：向数组中添加和删除项，para1表示可以添加的项数，para2表示删除的项数，后面的变量表示要添加的项的值，注意是从para1之后开始删除和添加的。

> 参数为负数的问题，如果para1为负数，则会加上数组的长度作为para1的值，而para2为负数或0的话不会执行删除操作，此时从 para1 当前位置开始添加变量。

split (操作字符串)

根据特定的字符切割字符串并返回切割后生成数组。

> 如果 a.split(''), 则切割每一个字符。

```
var string = 'abcde';
var array = string.split('');
// array = ['a','b','c','d','e']
```

reverse (反转数组中的元素)

```
var a = [1,2,3,4];
a.reverse();
//a = [4,3,2,1]
```

join (用于将数组转换为字符串)

> 不改变原数组

```
var a = [1,2,3,4];
a.join('') // return '1234'
a.join('.') // return '1.2.3.4'
//此时，a = [1,2,3,4]
```

综合

垃圾回收机制

Js具有自动回收垃圾的机制，垃圾收集器会按照固定的时间间隔周期性执行。

- 标记清除

  工作原理：当变量进入环境时，将这个变量标记为“进入环境”，当变量离开环境时，则将其标记为“离开环境”。标记为“离开环境”的就会被回收。

- 引用计数

  跟踪记录每个值被引用的次数，当一个值被引用的次数为0时，垃圾收集器下次运行时就会将值回收。

内存泄漏

问题	解决办法
全局变量引起的内存泄漏	严格使用全局变量
闭包引起的内存泄漏。闭包可以维持函数内部变量，使其得不到释放	避免闭包，或者在外部事件处理函数中删除对DOM的引用
被遗忘的定时器或者回调函数	删除定时器

JS对象属性遍历

1. 使用 Object.keys(objectName).forEach(function(element,index){ .... } 便利
2. for(var i in object) {...}
3. Object.getOwnPropertyNames(), 返回一个数组，包含自身的所有属性（不包含symbol属性，但是包含不可枚举属性）。
4. Reflect.ownKeys(obj) ，包含所有属性，包含symbol属性和不可枚举属性

```
var obj = {
  a: 'aaaaa',
  b: 'bbbbb',
  c: 'ccccc',
}

for ( var  i in obj ) {
  console.log(i, obj[i]);
}
//输出结果
//a aaaaa
//b bbbbb
//c ccccc

Object.keys(obj).forEach(
  function(ele,index) {
    console.log(ele,obj[ele],index)
  })
//输出结果
a dfafda 0
b afdaf 1
c ssssss 2
```

cookies && storage

`cookies`、`localStorage`、`sessionStorage` 都是在客户端以键值对存储的存储机制，并且只能将值存储为字符串。

`sessionStorage` 常用操作方法：

- session.clear(), 清空
- session.setItem('key','value')
- session.getItem('key')
- session.removeItem('key')
- session.length,

	cookies	localStorage	sessionStorage
初始化	客户端或服务器，服务器可使用Set-Cookies请求头	客户端	客户端
过期时间	手动设置	永不过期	当前页面刷新、关闭
与域名关联	是	否	否
容量	4kb	5mb	5mb
随请求发给服务器	是	否	否

冒泡/捕获/事件委托/事件代理

`事件委托` 就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。

举例：

有三个同事预计会在周一收到快递。为签收快递，有两种办法：一是三个人在公司门口等快递；二是委托给前台MM代为签收。现实当中，我们大都采用委托的方案（公司也不会容忍那么多员工站在门口就为了等快递）。前台MM收到快递后，她会判断收件人是谁，然后按照收件人的要求签收，甚至代为付款。这种方案还有一个优势，那就是即使公司里来了新员工（不管多少），前台MM也会在收到寄给新员工的快递后核实并代为签收。

1. 现在委托前台的同事是可以代为签收的，即程序中的现有的dom节点是有事件的；
2. 新员工也是可以被前台MM代为签收的，即程序中新添加的dom节点也是有事件的。

发生事件的元素可以使用 `ev.target` 获取到，利用事件冒泡。

参考文章

> 注意：$("ul").on("click","li",function(){});这样写有事件委托。$("ul li").on();这样写法则没有。

浏览器事件模型

1. 每个浏览器都有自己的事件模型，W3C为了兼顾标准，定义了三个阶段：捕获阶段，目标阶段，冒泡阶段。
2. elem.addEventListener("eventType", fn , boolean);
3. 参数说明: 事件类型，函数，true,表示在捕获阶段执行,false为在冒泡阶段执行(默认为false).
4. stopPropagation()可以阻止事件的传播，可以是捕获阶段也可以是冒泡阶段。

模块化

Commonjs、AMD、CMD、es6 modules

- CommonJS
  - 核心思想就是通过 require 方法来同步加载所要依赖的其他模块，然后通过 exports 或者 module.exports 来导出需要暴露的接口
  - 同步加载
  - 用于nodejs服务器端
- AMD规范
  - require.js
  - 非同步加载，允许指定回调函数

    - require([module],callback),引用函数
    - define(id, [dependences], callback), 定义函数
    - 支持 Commonjs 的导出方式
    - 只是提前加载所有依赖，不是按需加载
- CMD规范
    - sea.js
    - 按需加载
    - 依赖spm打包
    - ES6模块化
    - import
- es6 modules
    - 前端框架中常用

闭包

简答的说，如果一个function能除了能访问它的参数、局部变量或函数之外，还能访问外部环境变量（包括全局变量、DOM、外部函数的变量或者函数），那么它就可以成为一个闭包。

```
// 看代码猜结果：

function fun(n,o) {
  console.log(o);
  return {
    fun:function(m){
      return fun(m,n);
    }
  };
}

1. var a = fun(0); a.fun(1); a.fun(2); a.fun(3);//undefined,0,0,0
2. var b = fun(0).fun(1).fun(2).fun(3);//undefined，0,1,2,
3. var c = fun(0).fun(1); c.fun(2); c.fun(3);//undefined,0,1,1
```

> 参考文章

元素获取

原生

- getElementById()
- getElementsByName()
- getElementsByTagName()

模拟复杂元素获取方式

- getElementByClassNameAndTag

根据 `tagName` 和 `className` 获取元素

```
function getElementsByClassNameAndTag(tagName, className){
  var tags = document.getElemetsByTagName(tagName);
  var resultTags = [];
  for(var i = 0; i < tags.length; i++){
    if(tags[i].classname.indexOf(className) != -1){  /*     important, indexOf    */
      resultTags[resultTags.length] = tags[i];
    }
  }
  return resultTags;
}
```

- getElementsByClassName

只根据 `className` 来获取元素

```
function getElementsByClassName(className){
 var resultTags = []
```

```
 if( !document.getElementsByClassName ) {
   var allTags = document.getElementsByTagName('*');
   for ( var i = 0; i < allTags.length; i++ ) {
     if( allTags[i].className.indexOf(className) != -1 ) {
       resultTags[resultTags.length] = allTags[i];
     }
   }
 }else {
   var tags = document.getElementsByClassName(className);
   for ( var i = 0; i < tags.length; i++) {
     resultTags.push(tags[i]);
   }
 }
 return resultTags;
}
```

JQuery 相关知识

jsonp

jquery 中的 jsonp

`jsonp` 是JQuery 中解决 `跨域请求数据` 的一个解决办法。

```
$.ajax({
  url: "http://localhost:9090/student",
  type: "GET", //只支持GET方法
  dataType: "jsonp",  //指定服务器返回的数据类型
  jsonp: "callback",//传递给请求处理程序或页面的，用以获得jsonp回调函数名的参数名(一般默认为:callback)
  jsonpCallback: "showData",  //指定回调函数名称
  success: function (data) {
    console.info("调用success");
  },
  error: function () {
    console.log('error');
  }
});
```

> jsonp 只支持 GET 方法，因为 jsonp 的原理其实访问一个js文件然后调用回调函数的方式，所以并不能实现客户端向服务器传送文件。

JavaScript 实现 jsonp 方式的数据访问

- 客户端HTML文件

```
<script type="text/javascript">
  var localHandler = function(data){
      alert('我是本地函数，可以被跨域的remote.js文件调用，远程js带来的数据是：' + data.result);
  };
</script>
<script type="text/javascript" src="http://remoteserver.com/remote.js"></script>
```

- remote.js 文件

```
localHandler({"result":"我是远程js带来的数据"});
```

JavaScript 实现动态 jsonp

```
var flightHandler = function(data){
   alert('你查询的航班结果是：票价 ' + data.price + ' 元，' + '余票 ' + data.tickets + ' 张。');
};
// 提供jsonp服务的url地址（不管是什么类型的地址，最终生成的返回值都是一段javascript代码）
var url = "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998&callback=flightHandler";
// 创建script标签，设置其属性
var script = document.createElement('script');
script.setAttribute('src', url);
// 把script标签加入head，此时调用开始
document.getElementsByTagName('head')[0].appendChild(script);
```

JavaScript 编码实例

JavaScript 验证手机号

> 主要是正则表达式的书写。

```
var phoneNumber;

phoneNumber  = document.getElementById("phone-number-input");

phoneRole = /^(1[1-9][0-9]{9})$/;

if (phoneRole.test(phoneNumber) == false) {

  alert("phone number is wrong");

}else {

  alert ("phone number is correct")

}
```

JavaScript 拼接 URL

```
export function encodeSearchParams(obj) {

  const params = []

  Object.keys(obj).forEach((key) => {

    let value = obj[key]

    // 如果值为undefined我们将其置空

    if (typeof value === 'undefined') {

      value = ''

    }

    // 对于需要编码的文本（比如说中文）我们要进行编码

    params.push([key, encodeURIComponent(value)].join('='))

  })

  return params.join('&')

}
```

JavaScript 实现 ajax 方法

```
function getAJAX(fn,url){

  var xhr = createXHR();

  xhr.open("GET",url,true);

  xhr.onreadystatechange =  function(){

    if(xhr.readyState == 4){//异步请求时的状态码4代表数据接收完毕

      if(xhr.status == 200){//HTTP的状态 成功

        var data = eval("(" + xhr.responseText + ")");

        fn(data);//实现函数的回调,将结果返回

      }

    }

  }
```

```
  xhr.send(null);
}
```

JavaScript 如何实现一个栈

> todo

JavaScript 实现字符串反转

- 使用 `split` 将字符串转换为字符数组
- 使用 `reverse` 将字符数组反转
- 使用 `.join('')` 将字符数组转换为字符串

```
function stringReversal(string) {
  var stringArray = string.split('');
  var resultString= stringArray.reverse(stringArray);
  return resultString.join('');
}
```

合并两个有序数组（归并排序）

```
function mergeArray(array1, array2) {
  var index1 = 0, index2 = 0;
  var resultArray = [];

  while( index1 < array1.length && index2 < array2.length ) {
    if( array1[index1] < array2[index2] ) {
      resultArray.push(array1[index1]);
      index1 ++;
    }else {
      resultArray.push(array2[index2]);
      index2++;
    }
  }
  while( index1 < array1.length ) {
    resultArray.push(array1[index1]);
    index1++;
  }
  while( index2 < array2.length ) {
    resultArray.push(array2[index2]);
    index2++;
  }

  return resultArray;
}
```

ES6

Default Parameters（默认参数）

在 ES5 中定义默认参数需要一下这样：

```
function fun(a) {
    var a = a || 'default value';
    console.log(a);
}
fun('a'); //log a
fun();    //log default value
fun(0);   //log default vaule (error)
```

但是这样存在一个问题：如果传入的参数 a 为 0，那 fun() 方法就会出错，因为 0 代表 false。

在 ES6 中，我们可以这样定义默认参数：

```
function fun(a = 'default value') {
    console.log(a);
}
//不仅代码更加简洁，而且不容易出错。
fun('a');    //log a
fun();  //log default value
fun(0); //log 0
```

> 定义默认参数的方式类似于 Ruby。

Template Literals （模板文本）

在 ES6 之前用比变量拼接字符串：

```
var first = 'firstName';
var last = 'lastName';
var name = 'My name is:' + first + '.' + last;
```

在 ES6 中因为支持模板文本，所以可以写成这样：

```
var first = 'firstName';
var last = 'lastName';
var name = `My name is ${first}.${last}`;
```

> 注意：模板文本变量的开头和结尾不是单引号，而是反引号。

Multi-line Strings（多行字符串）

在 ES6 中支持反引号的多行字符串。

例如：

```
var str = `Hello,
        My name is RN.`
```

Destructuring Assignment（解构赋值）

我们在编码的过程中经常定义许多对象和数组，然后有组织地从中提取相关的信息片段。

在ES6中添加了可以简化这种任务的新特性：解构。

解构是一种打破数据结构，将其拆分为更小部分的过程。

下边举一个简单的例子：

在 ES5 中，我们如果想获取一个对象中的属性的值的话，一般写成这样：

```
var person = {
    name: 'MyName',
    age: 20,
}

var name = person.name;
var age = person.age;

console.log(name);  //log MyName
console.log(age);   //log 20
```

在 ES6 中我们有更简单的写法：

```
var person = {
    name: 'MyName',
    age: 20,
}

var { name, age } = person;
console.log(name);  //log MyName
console.log(age);   //log 20
```

> 结构赋值同样支持数组。

Enhanced Object Literals（增强的对象字面量）

Arrow Functions（箭头函数）

箭头函数，属于匿名函数，使用 => 来定义函数。

下边是同一个函数，使用不同写法的对比：

```
var name = 'XiaoMing';

//es5
(function (name) {
    console.log(`Hello, everyone! My name is ${name}`)
})(name)


//es6
var fun = (name) => {
    console.log(`Hello, everyone! My name is ${name}`)
}
fun()   //log Hello, everyone! My name is Xiaoming
```

Promises

Promise 对象用于一个异步操作的最终完成（或失败）及其结果值的表示。简单点说，它就是用于处理异步操作的，异步处理成功了就执行成功的操作，异步处理失败了就捕获错误或者停止后续操作。

状态

Promise 有三种状态，分别是 pending, fulfilled, rejected。

- pending，初始状态,也称为未定状态，就是初始化 Promise 时，调用 executor 执行器函数后的状态。
- fulfilled，完成状态，意味着异步操作成功。
- rejected，失败状态，意味着异步操作失败。

状态转化

- 操作成功，pending -> dfulfilled
- 操作失败，pending -> rejected

方法

then()

then() 调用后返回一个 Promise 对象，意味着实例化后的 Promise 对象可以进行链式调用，而且这个 then() 方法可以接收两个函数，一个是处理成功后的函数，一个是处理错误结果的函数。

例如：

```
var promise = new Promise(function(resolve, reject) {
  // 2秒后置为接收状态
  setTimeout(function() {
    resolve('success');
  }, 2000);
});

promise.then(function(data){
    console.log(data)
}, function(error) {
    console.log(error);
})
```

catch()

catch() 方法和 then() 方法一样，都会返回一个新的 Promise 对象，它主要用于捕获异步操作时出现的异常。因此，我们通常省略 then() 方法的第二个参数，把错误处理控制权转交给其后面的 catch() 函数。

例如：

```
var promise = new Promise(function(resolve, reject) {
    setTimeOut(function() {
        resolve('success');
    }, 2000);
});

promise.then(function(data) {
    console.log(data);
}).catch(function(error) {
    console.log(data);
})
```

Promise.all()

Promise.all()接收一个参数，它必须是可以迭代的，比如数组。

它通常用来处理一些并发的异步操作，即它们的结果互不干扰，但是又需要异步执行。它最终只有两种状态：成功或者失败。

它的状态受参数内各个值的状态影响，即里面状态全部为fulfilled时，它才会变成fulfilled，否则变成rejected。

成功调用后返回一个数组，数组的值是有序的，即按照传入参数的数组的值操作后返回的结果。

Promise.race()

Promise.race() 和 Promise.all() 类似，都接收一个可以迭代的参数，但是不同之处是 Promise.race() 的状态变化不是全部受参数内的状态影响，一旦参数内有一个值的状态发生的改变，那么该 Promise 的状态就是改变的状态。就跟 race 单词的字面意思一样，谁跑的快谁赢。

Promise.resolve()

Promise.resolve() 接受一个参数值，可以是普通的值，具有 then() 方法的对象和 Promise 实例。正常情况下，它返回一个 Promise 对象，状态为 fulfilled。但是，当解析时发生错误时，返回的 Promise 对象将会置为 rejected 态。

Promise.reject()

Promise.reject() 和 Promise.resolve() 正好相反，它接收一个参数值 reason，即发生异常的原因。此时返回的 Promise 对象将会置为 rejected 态。

Block-Scoped Constructs Let and Const（块作用域构造Let and Const）

let

let 类似 var，但是只对 {} 代码块中起作用。

const

声明一个只读的变量，并且需要在声明的时候赋值。

Classes（类）

通过class关键字,可以定义类。基本上,ES6的class可以看作只是一个语法糖,它的绝大部分功能,ES5都可以做到,新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法。

ES5 中定义一个类：

```
function Person(name,age) {
    this.name = name;
    this.age = age;
    ...
}
Person.prototype.toString = function() {
    return (this.name + '的年龄是' + this.age + '岁')
}
```

ES6 中定义一个类：

```
class Person{
    // 构造
    constructor(x,y){
        this.x = x;
        this.y = y;
    }

    toString(){
        return (this.x + "的年龄是" +this.y+"岁");
    }
}
```

Modules（模块）

- 每一个模块只加载一次， 每一个JS只执行一次， 如果下次再去加载同目录下同文件，直接从内存中读取；
- 每一个模块内声明的变量都是局部变量， 不会污染全局作用域；
- 模块内部的变量或者函数可以通过export导出；
- 一个模块可以导入别的模块

export && import

模块功能主要由两个命令构成：

1. export，用于规定模块的对外接口。
2. import，用于输入其他模块提供的功能。

> 可以在 import 模块的时候使用 as 为引入的模块重新命名。

说明

1. import 后面的 from 指定模块文件的位置，可以是相对路径，也可以是绝对路径，.js后缀可以省略。如果只是模块名，不带有路径，那么必须有配置文件，告诉 JavaScript 引擎该模块的位置。
2. import命令具有提升效果，会提升到整个模块的头部，首先执行。
3. 由于import是静态执行，所以不能使用表达式和变量。
4. import 除了指定加载某个输出值，还可以使用整体加载，即用星号（*）指定一个对象，所有输出值都加载在这个对象上面。
5. 13.export default命令，为模块指定默认输出，这样就可以在使用 import 命令的时候，不必知道所要加载的变量名或函数名。但是，一个模块只能有一个 export default。

6. 如果在一个模块之中，先输入后输出同一个模块，import语句可以与export语句写在一起。

   例如：

```
export { foo, bar } from 'module';

// 等同于
import { foo, bar } from 'module';
export { foo, bar };
```

7. const声明的常量只在当前代码块有效。如果想设置跨模块的常量（即跨多个文件），或者说一个值要被多个模块共享，可以采用下面的写法:

```
// constants.js 模块
export const A = 1;
export const B = 2;

// test1.js 模块
import * as constants from './constants';
console.log(constants.A); // 1
console.log(constants.B); // 2

// test2.js 模块
import {A, B} from './constants';
console.log(A); // 1
console.log(B); // 2
```

前端框架的比较

> VUE 的对比其他框架

几个框架修改数据的不同

Vue

1. 创建一个数据对象，其中的数据可以自由更改。

> this.name = 'a' 赋值。this.name = 'b'，更改。

React

1. 创建一个状态对象，更改数据需要更多的操作。
2. 例如：this.status(...)
3. React之所以使用状态的方式，是为了方便在数据进行了改变之后，及时追踪和运行声明周期hook。

> this.state.name = 'a' 赋值。 this.setState({ name:'b' })。

Angular

React

props && state

1. props是不可变的，而state是可以根据用户的交互来改变的。
2. props不可改变是指在组件本身中修改，不过因为组件中的props本质作用是一种父级向子级传递数据的方式，所以可以在父组件中修改。
3. React 中更新组件的state，会导致重新渲染用户界面（不需要操作DOM），简单来说就是用户界面会随着state变化而变化

路由实现原理

> TODO

虚拟DOM && Diff算法

虚拟DOM

React在Virtual DOM上实现了DOM diff算法，当数据更新时，会通过diff算法计算出相应的更新策略，尽量只对变化的部分进行实际的浏览器的DOM更新，而不是直接重新渲染整个DOM树，从而达到提高性能的目的。

Diff算法

> 相关网址

虚拟DOM树分层比较，忽略DOM节点跨层级的移动操作。React只会对相同颜色方框内的DOM节点进行比较，即同一个父节点下的所有子节点。当发现节点已经不存在，则该节点及其子节点会被完全删除掉，不会用于进一步的比较。这样只需要对树进行一次遍历，便能完成整个DOM树的比较。

如果真的发生跨级移动的话，React是删除原来的，再新建一个挂到DOM树上。

vue

数据双向绑定实现原理

vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。

> TODO

综合知识

缓存

当没有CDN时，客户端访问网站，先检查本地缓存是否过期，如果过期，则由网站返回新的数据，若没有过期，直接使用缓存资源（200）。

概念

强缓存

强缓存，指的是使用本地的缓存。

协商缓存

协商缓存，是指本地的缓存到期后，与服务器确认缓存是否真正过期，过期返回新的资源（200），没有过期（304资源仍有效）

CDN缓存

客户端浏览器先检查是否有本地缓存是否过期：

- 如果过期，则向CDN边缘节点发起请求，CDN边缘节点会检测用户请求数据的缓存是否过期，
- 如果没有过期，则直接响应用户请求，此时一个完成http请求结束；
- 如果数据已经过期，那么CDN还需要向源站发出回源请求（back to the source request）,来拉取最新的数据。

CDN缓存策略

一般通过http响应头中的 `Cache-control: max-age` 的字段来设置CDN边缘节点数据缓存时间。

自定义重写 **Select**

> TODO

正则表达式

常用字符

符号	符号意义
/...../	用来表示一个正则匹配表达式。
\	转义字符。
/^a/	表示当第一个字符为a时才能匹配。
/t$/	最后一个字符为t才会匹配。
*	一个字符出现任意次，即匹配。
？	匹配一个字符0次或者1次。
{n}	匹配前边的字符发生n次。例如，/e{3}/ 就会匹配 eee 及>3e 的字符串。
{n,m}	最少n次，最多m次
[abcd]	字符集和，匹配方括号中的任意字符
abcd	匹配不存在方括号中的任意字符
\d	匹配一个数字
\D	匹配一个非数字

前后端交互相关

RESTful 接口

`RESTful接口` 是一种软件架构设计风格，主要用于前后端的交互，它的重要一个原则是前端的请求是没有状态的，无状态的请求可以由任何服务器回答。 RESTful提倡使用标准的 HTTP 方法，GET对应取、PUT对应修改、POST新增 和 DELETE对应删除。

TCP和UDP的区别

TCP：

- 面向连接，进行数据传输前需要连接。因此连接是点对点的。
- TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达。

  > （Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。）

- TCP对系统资源要求较多。

UDP：

- 无连接的，进行传输前不需要建立连接。可以是一对一，一对多，多对一和多对多。
- 尽最大努力进行交付，但是不保证交付的可靠性。例如视频网站传输视频。
- UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。
- UDP对系统资源要求较少。

HTTP 状态码

代码	含义
1	消息
2	成功
3	重定向
4	请求错误
400	Bad Request
401	Unauthorized
402	PaymentRequired
403	Forbidden
404	Not Found
5	服务器错误
500	Internal Server Error

css、js加载会不会阻塞DOM解析和绘制

CSS不会阻塞DOM的解析，但是会阻塞DOM的绘制。

CSS会阻塞之后的JS的运行。

XSS和CSRF

XSS

`XSS` 是跨站脚本攻击。 例如：。也可以写成图片，iframe的形式

CSRF

`CSRF` 是跨站请求伪造。 攻击者盗用了用户的身份，以用户的名义发送恶意请求，包括：以用户的名义发送邮件，发消息，盗取用户账号等。

跨域问题场景【细看】

跨域详解

- Ajax直接请求普通文件存在跨域无权限访问的问题，甭管你是静态页面、动态网页、web服务、WCF，只要是跨域请求，一律不准。
- Web页面上调用js文件时则不受是否跨域的影响（不仅如此，我们还发现凡是拥有"src"这个属性的标签都拥有跨域的能力，比如 `<script>`、`<img>`、`<iframe>` ）

DOM同源策略

禁止对不同源页面DOM进行操作, 同源指，域名、协议、端口 如果非同源，js脚本执行时浏览器会报错，提示拒绝访问。（可以下载，但是不可以有数据操作）

跨域问题解决

- jsonp

原理：通过动态创建script标签,然后利用src属性进行跨域，设置一个回调函数。

```
// 定义一个fun函数
function fun(fata) {
    console.log(data);
};
// 创建一个脚本，并且告诉后端回调函数名叫fun
var body = document.getElementsByTagName('body')[0];
var script = document.gerElement('script');
script.type = 'text/javasctipt';
script.src = 'demo.js?callback=fun';
body.appendChild(script);
```

> jsonp不能发post请求，只能发get请求。因为它是调用回调函数的方式，在服务器看来只是请求了一个资源。

重绘 && 重排

一个页面由两部分组成：

- DOM：描述该页面的结构
- render：描述 DOM 节点 (nodes) 在页面上如何呈现

重绘 Repaint

当 DOM 元素的属性发生变化 (如 color) 时, 浏览器会通知 render 重新描绘相应的元素, 此过程称为 repaint。

重排 Reflow

如果该次变化涉及元素布局 (如 width), 浏览器则抛弃原有属性, 重新计算并把结果传递给 render 以重新描绘页面元素, 此过程称为 reflow。

> 重排必定会引发重绘，但重绘不一定会引发重排。
> 重排对性能的影响要大于重绘。

浏览器解析顺序

1. 解析HTML代码并生成一个 DOM 树。
2. 解析CSS文件，顺序为：浏览器默认样式->自定义样式->页面内的样式。
3. 生成一个渲染树（render tree）。这个渲染树和DOM树的不同之处在于，它是受样式影响的。它不包括那些不可见的节点。
4. 当渲染树生成之后，浏览器就会在屏幕上"画"出所有渲染树中的节点。

优化方法

（1）直接改变元素的className （2）display：none；先设置元素为display：none；然后进行页面布局等操作；设置完成后将元素设置为display：block；这样的话就只引发两次重绘和重排； （3）不要经常访问浏览器的flush队列属性；如果一定要访问，可以利用缓存。将访问的值存储起来，接下来使用就不会再引发回流； （4）使用cloneNode(true or false) 和 replaceChild 技术，引发一次回流和重绘； （5）将需要多次重排的元素，position属性设为absolute或fixed，元素脱离了文档流，它的变化不会影响到其他元素； （6）如果需要创建多个DOM节点，可以使用DocumentFragment创建完后一次性的加入document；

浏览器相关

浏览器内核

内核名称	代表浏览器
Trident	Internet Explore、搜狗
Gecko	Mozilla、Firefox
WebKit	Safari、Chrome
Presto	Opera

浏览器输入**URL**敲击回车后发生了什么？

> 这个问题可以理解成浏览器是怎么访问一个网站了。

分别有以下几个步骤：

1. 浏览器解析URL

一个URL通常包括以下几个参数：

- 传输协议
- 域名
- 端口
- 虚拟目录
- 文件名
- 参数
- 锚

浏览器解析URL，主要是解析出需要访问的服务器的域名。

1. DNS解析

DNS解析就是域名对应IP地址的查询。

浏览器DNS缓存（2-30分钟） > 系统DNS缓存(host文件) > 路由器DNS缓存 > 查询ISP DNS缓存(网络供应商) > 递归查询(从根域名服务器到顶级域名服务器再到极限域名服务器依次搜索哦对应目标域名的IP。)

1. 浏览器与网站建立TCP连接（三次握手）

> 三次握手，四次挥手

1. 请求和传输数据

> GET、POST等进行数据传输

1. 浏览器渲染页面

问题

你是如何学习前端的

大概就讲了视频（入门）=》文档（w3school/菜鸟教程/MDN/阮一峰）=》书（dom编程艺术/红宝书/understanding es6,/es6入门教程）=》官方文档（w3.org/vuejs.org）

没事转转博客、社区。

以及接下来打算学习什么，看什么书。

HTTP

HTTP 分为 HTTP1.0 和 HTTP1.1 两个版本。

生命周期

HTTP 1.0

HTTP 的生命周期通过 Request 来界定，也就是一个 Request 一个 Response ，那么在 HTTP1.0 中，这次HTTP请求就结束了。

HTTP 1.1

对 HTTP 1.0 进行了改进，使得有一个 keep-alive，也就是说，在一个HTTP连接中，可以发送多个Request，接收多个Response。

> 但是，Request = Response，在 HTTP 中永远是这样，也就是说一个 request 只能有一个 response。而且这个 response 也是被动的，不能主动发起。

http 方式的实时获取信息（仿 WebSOcket）

http long poll

`long poll` 其实原理跟 ajax轮询 差不多，都是采用轮询的方式。
不过采取的是阻塞模型（一直打电话，没收到就不挂电话），也就是说，客户端发起连接后，如果没消息，就一直不返回Response给客户端。直到有消息才返回，返回完之后，客户端再次建立连接，周而复始。

ajax 轮询

ajax轮询的原理非常简单，让浏览器隔个几秒就发送一次请求，询问服务器是否有新信息。

> 无论是 http long poll 还是 ajax 轮询都是被动的，都是由客户端不停的发起请求，都是非常消耗资源的。
> 不同的是，ajax轮询 需要服务器有很快的处理速度和资源（速度）。 而，long poll 需要有很高的并发，也就是说同时接待客户的能力。
> 而且，http连接是无状态的，每次连接都是一个新的连接。

HTTP1.0 和 HTTP1.1 和 HTTP2.0 的区别

HTTP1.0 和 HTTP1.1 主要区别

- 长连接

HTTP 1.0需要使用keep-alive参数来告知服务器端要建立一个长连接，而HTTP1.1默认支持长连接。

HTTP是基于TCP/IP协议的，创建一个TCP连接是需要经过三次握手的,有一定的开销，如果每次通讯都要重新建立连接的话，对性能有影响。因此最好能维持一个长连接，可以用个长连接来发多个请求。

- 节约带宽

HTTP 1.1支持只发送header信息(不带任何body信息)，如果服务器认为客户端有权限请求服务器，则返回100，否则返回401。客户端如果接受到100，才开始把请求body发送到服务器。这样当服务器返回401的时候，客户端就可以不用发送请求body了，节约了带宽。

另外HTTP1.1还支持传送内容的一部分。这样当客户端已经有一部分的资源后，只需要跟服务器请求另外的部分资源即可。这是支持文件断点续传的基础。

- HOST 域

现在可以web server例如tomat，设置虚拟站点是非常常见的，也即是说，web server上的多个虚拟站点可以共享同一个ip和端口。

HTTP1.0是没有host域的，HTTP1.1才支持这个参数。

HTTP1.1 和 HTTP2.0 主要区别

- 多路复用

HTTP2.0使用了多路复用的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。

当然HTTP1.1也可以多建立几个TCP连接，来支持处理更多并发的请求，但是创建TCP连接本身也是有开销的。

TCP连接有一个预热和保护的过程，先检查数据是否传送成功，一旦成功过，则慢慢加大传输速度。因此对应瞬时并发的连接，服务器的响应就会变慢。所以最好能使用一个建立好的连接，并且这个连接可以支持瞬时并发的请求。

> 二进制分帧,在 应用层(HTTP/2)和传输层(TCP or UDP)之间增加一个二进制分帧层,在这个二进制分帧层，所有传输的消息被分为更小的消息和帧，并对它们采用二进制编码，其中 http1.x的首部信息会被封装到HEADER frame，而相应的 request body 则被封装到 DATA frame 中。在建立连接之初，http2.0会限制连接的速度，当数据传输成功后，会逐步增大信息传输的速度。

- 数据压缩

HTTP1.1不支持header数据的压缩，HTTP2.0使用 `HPACK算法` 对header的数据进行压缩，这样数据体积小了，在网络上传输就会更快。

- 服务器推送

意思是说，当我们对支持HTTP2.0的web server请求数据的时候，服务器会顺便把一些客户端需要的资源一起推送到客户端，免得客户端再次创建连接发送请求到服务器端获取。这种方式非常合适加载静态资源。

http1.x 服务端只能对一个请求发生一个 response，但是 http2.0 可以对一个 request 返回多个 response。

> HTTP2.0 参考文章

WebSocket

`Websocket` 其实是一个新协议，跟 `HTTP` 协议基本没有关系，只是为了兼容现有浏览器的握手规范。由 `HTML5` 提出。

Websocket 实例

典型的 Websocket 握手（借用Wikipedia的）如下：

浏览器发送给服务器

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

- `Upgrade: websocket` 和 `Connection: Upgrade`，告诉 nginx、apache 等服务器，此次请求是 `websocket` 请求。
- `Sec-WebSocket-Key` 是一个 Base64 encode 的值，这个是浏览器随机生成的，告诉服务器：泥煤，不要忽悠窝，我要验证尼是不是真的是Websocket助理。
- `Sec_WebSocket-Protocol` 是一个用户定义的字符串，用来区分相同 URL 下，不同的服务所需要的协议。简单理解：今晚我要服务A，别搞错啦~
- `Sec-WebSocket-Version` 是告诉服务器所使用的 `Websocket Draft` （协议版本）

服务器返回：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

- 同样，`Upgrade: websocket` 和 `Connection: Upgrade`，告诉浏览器成功切换协议为 `websocket`.
- `Sec-WebSocket-Accept` 这个则是经过服务器确认，并且加密过后的 `Sec-WebSocket-Key`。服务器：好啦好啦，知道啦，给你看我的ID CARD来证明行了吧。
- `Sec-WebSocket-Protocol` 则是表示最终使用的协议. (至此，http作用已经完成)

WebSocket

websocket 的创建

```
var Socket = new WebSocket(url, [protocol] );
```

> 以上代码中的第一个参数 url, 指定连接的 URL。第二个参数 protocol 是可选的，指定了可接受的子协议。

websocktet 的状态

只读属性 `readyState` 表示连接状态，可以是以下值：

值	表示的状态
0	表示连接尚未建立。
1	表示连接已建立，可以进行通信。
2	表示连接正在进行关闭。
3	表示连接已经关闭或者连接不能打开。

websocket 的事件

事件	事件处理程序	描述
open	Socket.onopen	连接建立时触发
message	Socket.onmessage	客户端接收服务端数据时触发
error	Socket.onerror	通信发生错误时触发
close	Socket.onclose	连接关闭时触发

websocket 方法

方法	描述
Socket.send()	使用连接发送数据
Socket.close()	关闭连接

> 单个 TCP 连接上进行全双工通讯的协议。
> 获取 Web Socket 连接后，可以通过 send() 方法来向服务器发送数据，并通过 onmessage 事件来接收服务器返回的数据。
> 持久化

websocket 的实现

- python的一个开源项目 pywebsocket 可以实现简单的 websocket 服务端。
- WebSocket-Node 实现 websocket 服务端，需要依赖于底层的C++,Python的环境，支持以node做客户端的访问。
- faye-websocket-node 实现 websocket 服务端，是faye软件框架体系的一部分，安装简单，不需要其他依赖库。
- socket.io 实现 websocket 服务端，功能强大，支持集成websocket服务器端和Express3框架与一身。

webSocket 如何兼容低浏览器

Adobe Flash Socket 、 ActiveX HTMLFile (IE) 、 基于 multipart 编码发送 XHR 、 基于长轮询的 XHR。

算法

二分查找

查找数组一般顺序排列，设定min，max，middle，其中middle=(min+max)/2，当x < middle, max = middle - 1, 当 x>middle,min = min+1。

排序算法

> TODO

TCP/IP 协议

传输层

链路层，网络层，传输层，应用层。

- 应用层：常用的Http 通过Http协议设置相关数据，比如请求行，请求头，请求方法，请求体，请求参数等等（Https：Http+SSL，是在应用层和传输层之间对请求数据进行加密处理，和Http相比安全有证书的验证）
- 传输层：传输层是面向通信的最高层，也是用户功能的最底层。复用和分发，在发送端多个进程可复用一个传输层，在接收端为不同主机的不同进程进行分发数据。其中使用TCP协议、UDP协议等。
- 网络层：使用IP协议，这里面的IP协议不是指IP地址，IP协议包含IP地址和计算机的MAC地址，IP协议能够通过IP地址和MAC地址准确的找到你需求请求的服务器
- 物理层：就是我们平时接触的网卡和网卡的驱动程序等。

二、三次握手

> 参考链接

两个序号和三个标志位：

（1）序号：seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。（2）确认序号：ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，ack=seq+1。（3）标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，具体含义如下：（A）URG：紧急指针（urgent pointer）有效。（B）ACK：确认序号有效。（C）PSH：接收方应该尽快将这个报文交给应用层。（D）RST：重置连接。（E）SYN：发起一个新连接。（F）FIN：释放一个连接。

建立新连接过程

客户端 -> SYN=1,seq=x -> 服务端 客户端 <- SYN=1,ACK=1,seq=y,ack=x+1 <- 服务端 客户端 -> ACK=1,seq=x+1,ack=y+1 -> 服务端

> seq 为单方面消息序列号，ack为单方面待收消息序列号。

三、四次挥手

断开连接过程

client -> FIN=m -> server client <- ack=m+1 <- server client <- FIN=n <- server client -> ACK=1,ack n+1 -> server

为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。

而关闭连接时，收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，

己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，

再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送。

数据结构

树

> TODO

树的前序、中序、后序

- 深度优先

方便记忆，前序、中序、后序是指 `根节点` 在排序查找中的位置。

名称	遍历顺序
前序	节点--左孩子--右孩子
中序	左孩子--节点--右孩子
后序	左孩子--右孩子--节点

- 广度优先

从上往下逐层遍历，同一层中从左到右对节点逐一访问。

红黑树

- 红黑树是一种自平衡二叉查找树。
- 用途：是实现关联数组。
- 时间复杂度：在O(log n)时间内做查找，插入和删除，这里的n 是树中元素的数目。

串、栈、队列、链表、哈希表、树、图

哈希表是怎样的结构？

操作系统的内存管理

怎么实现一个服务器

如何删除一个dom节点 removeChild()或者innerHtml去替换

前端开发

前端开发流程

前端开发接口设计

前端开发要干什么

安全、性能、兼容

前端攻击

说下性能优化

主要就结合了网络和浏览器运行机制来讲述如何优化。

工具

Git

常用命令

常用命令	作用
git fetch	拉取
git pull	拉取合并
git push	推送
git status	当前状态
git add .	添加修改文件
git commit -m	暂存
git tag	打tag
git branch	新建分支
git checkout	切换分支、回滚
git remote	操作远程库
git config	操作git配置文件
git merge	合并

git fetch 与 **git pull** 的区别

一般认为， git pull 是 git fetch 和 git merge 的合并。

Linux 相关知识

常用命令

> TODO

相关概念

线程、进程

进程

是表示资源分配的基本单位。

线程

线程则是进程中执行运算的最小单位，即执行处理机调度的基本单位

Java开发中的线程并发

线程安全：当多个线程访问某个类时，这个类始终都能表现出正确的行为，那么就称这个类是线程安全的。理解为程序按照你的代码逻辑执行，并始终输出预定的结果。

进程状态

相关概念

进程状态：一个进程的生命周期可以划分为一组状态，这些状态刻画了整个进程。进程状态即体现一个进程的生命状态。

状态模型

- 三态模型：
  - 运行态：进程占有处理器正在运行。
  - 就绪态：进程具有运行条件，等待系统分配处理器运行。
  - 阻塞态：进程不具备运行条件，正在等待某个事件的完成。
- 五态模型：新建态、终止态，运行态，就绪态，阻塞态。

进程间的通信（IPC，Inter-process Communcation)

相关概念

IPC两个操作

- 发送(send message)
- 接受(receive message)

进程通信流程

- 在进程间建立通信链路
- 通过 send/receive 发送接收消息

进程通信方式

- 直接通信:(共享信道)
  - 进程必须正确命名对方，比如send（p，message）向p发送信息，receive（q，message）从q中接收信息
  - 通信链路有如下属性：自动建立连接；一条链路恰好对应一对通信进程；

    - 每对进程之间只有一个链接存在；
    - 链路可以是单向的，但是通常为双向的
- 间接通信:(通过OS内核)
    - 通过OS维护的消息队列实现进程之间的通信（接收发送信息）
    - 每个消息队列都有一个唯一标识，只有共享了相同消息队列的进程才能够通信
    - 通信链路有如下属性：只有共享了相同消息队列的进程才能建立连接，比如send（a，message）向消息队列a发送信息，receive（a，message）从消息队列a中接收信息；
    - 连接单向或者双向；
    - 消息队列可以与多个进程相互关联；
    - 每个进程共享多个消息队列（也就是进程和消息队列是多对多的关系）

进程间通信的**5**种方式

> TODO

线程并发

> TODO

Python3 相关知识

简单爬虫

urlib

使用urlib这个组件抓取网页，urllib是一个URL处理包，这个包中集合了一些处理URL的模块。

模块名	作用
urllib.request	模块是用来打开和读取URLs的
urllib.error	模块包含一些有urllib.request产生的错误，可以使用try进行捕捉处理
urllib.parse	模块包含了一些解析URLs的方法
urllib.robotparser	模块用来解析robots.txt文本文件.它提供了一个单独的RobotFileParser类，通过该类提供的can_fetch()方法测试爬虫是否可以下载一个页面

爬取一个网页的步骤

1. 使用 `urllib.request.urlopen()` 打开一个网页。

   > urlopen有一些可选参数，具体信息可以查阅Python自带的documentation。

2. 读取网页内容。

```
# -*- coding: UTF-8 -*-
from urllib import request

if __name__ == "__main__":
    response = request.urlopen("http://fanyi.baidu.com")
    html = response.read()
    html = html.decode('utf-8')
    print(html)
```

3.

Java 相关知识

JVM

`JVM` 是 `Java virtual machine` 的缩写，可以理解成一个虚构的计算机。

Java 在引入了Java虚拟机之后，Java代码就可以不需重新编译而在不同的平台上运行。

Java语言使用Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。这就是Java的能够“一次编译，到处运行”的原因。

> TODO

数据库

相关概念

死锁

> 集合中的每一个进程都在等待只能由本集合中的其他进程才能引发的事件，那么该组进程是死锁的

- 死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程.
- 不只是关系数据库管理系统，任何多线程系统上都会发生死锁，并且对于数据库对象的锁之外的资源也会发生死锁.例如，多线程操作系统中的一个线程要获取一个或多个资源（例如，内存块）

共享锁（读锁）

共享锁就是多个事务对于同一数据可以共享一把锁，都能访问到数据，但是只能读不能修改。

排他锁（写锁）

排他锁就是不能与其他所并存，如一个事务获取了一个数据行的排他锁，其他事务就不能再获取该行的其他锁，包括共享锁和排他锁，但是获取排他锁的事务是可以对数据就行读取和修改。

> 排他锁指的是一个事务在一行数据加上排他锁后，其他事务不能再在其上加其他的锁。mysql InnoDB引擎默认的修改数据语句，update,delete,insert都会自动给涉及到的数据加上排他锁，select语句默认不会加任何锁类型，如果加排他锁可以使用select ...for update语句，加共享锁可以使用select ... lock in share mode语句。

常用命令

> TODO