

# 前端基础笔记

曹思远

2021 年 1 月 8 日

## 目录

<b>1 软件安装</b>	<b>4</b>
1.1 必备软件列表 . . . . .	4
1.1.1 必备软件配置 . . . . .	4
<b>2 Git 入门</b>	<b>6</b>
2.1 命令行入门 . . . . .	6
2.2 本地仓库 . . . . .	6
2.3 远程仓库 . . . . .	6
<b>3 HTML</b>	<b>6</b>
3.1 概览 . . . . .	6
3.2 标签 . . . . .	6
3.3 重难点 . . . . .	6
3.4 实践和手机调试 . . . . .	6
<b>4 CSS</b>	<b>6</b>
4.1 基础 . . . . .	6
4.1.1 语法 . . . . .	6
4.1.2 Border 调试法 . . . . .	6
4.1.3 文档流 . . . . .	6
4.1.4 盒模型 . . . . .	7
4.1.5 margin 合并 . . . . .	8
4.1.6 基本单位 . . . . .	8
4.1.7 练手项目 . . . . .	8
4.2 布局 . . . . .	8
4.2.1 布局分类 . . . . .	8
4.2.2 两种布局思路 . . . . .	9
4.2.3 float 布局 . . . . .	9
4.2.4 Flex 布局 . . . . .	10
4.2.5 Grid 布局 . . . . .	10
4.3 定位 . . . . .	11
4.3.1 一个 div 的分层 . . . . .	11
4.3.2 positon 的五个取值 . . . . .	11
4.3.3 层叠上下文 . . . . .	11
4.4 动画 . . . . .	11
4.4.1 动画的原理 . . . . .	11

4.4.2 浏览器渲染的原理 . . . . .	11
4.4.3 CSS 动画优化 . . . . .	12
4.4.4 transition . . . . .	12
4.4.5 transition 过渡 . . . . .	14
4.4.6 红心实践 . . . . .	14
<b>5 HTTP</b>	<b>14</b>
5.1 URL . . . . .	14
5.1.1 IP . . . . .	14
5.1.2 端口 . . . . .	15
5.1.3 域名 . . . . .	15
5.1.4 HTTP 协议 . . . . .	17
5.2 请求响应和 NodeJS Sever . . . . .	17
<b>6 JS</b>	<b>17</b>
6.1 概览 . . . . .	17
6.1.1 硬要求 . . . . .	17
6.1.2 JS 的历史 . . . . .	18
6.2 内存图与 JS 世界 . . . . .	19
6.2.1 操作系统常识 . . . . .	19
6.2.2 内存图 . . . . .	20
6.2.3 JS 的世界是怎样的 . . . . .	21
6.2.4 原型链 . . . . .	23
6.3 Canvas 实践—画图板 . . . . .	25
6.4 JS 语法 . . . . .	25
6.4.1 JS 版本 . . . . .	26
6.4.2 语法 . . . . .	26
6.4.3 JS 数据 . . . . .	28
6.4.4 JS 对象 . . . . .	29
6.4.5 JS 对象分类 . . . . .	31
<b>7 算法与数据结构</b>	<b>32</b>
<b>8 JS 编程接口</b>	<b>32</b>
8.1 DOM 编程 . . . . .	32
8.1.1 DOM 简介 . . . . .	32
8.1.2 获取元素的 API . . . . .	33
8.1.3 元素的 6 层原型链 . . . . .	33
8.1.4 创建元素的 API . . . . .	33
8.1.5 查看元素的 API . . . . .	33
8.1.6 DOM 操作跨线程 . . . . .	33
8.2 手写 DOM 库 . . . . .	33
8.3 jQuery 中的设计模式 . . . . .	33
8.4 DOM 事件与事件委托 . . . . .	33
<b>9 前端站点导航</b>	<b>39</b>

<b>10 前后分离</b>	<b>39</b>
10.1 AJAX 的原理 . . . . .	39
10.2 异步与 Promise . . . . .	39
10.3 跨域, CORS, JSONP . . . . .	39
10.4 静态服务器 . . . . .	44
10.5 Ajax 实战: Cookie, Session . . . . .	44
<b>11 MVC</b>	<b>44</b>
<b>12 Webpack</b>	<b>44</b>
<b>13 虚拟 DOM 与 DOM diff</b>	<b>44</b>
<b>14 Vue</b>	<b>44</b>
<b>15 React</b>	<b>44</b>
<b>16 NodeJS</b>	<b>44</b>
<b>17 Vue3 造轮子</b>	<b>44</b>

# 1 软件安装

```

Name.prototype = {
  methodName: function(params){
    var doubleQuoteString = "some text";
    var singleQuoteString = 'some more text';
    // this is a comment
    if(this.confirmed != null && typeof(this.confirmed) == Boolean && this.confirmed == true){
      document.createElement('h3');
      $('#system').append("This looks great");
      return false;
    } else {
      throw new Error;
    }
  }
}

```

Listing 1: My Javascript Example

## 1.1 必备软件列表

1. vscode
2. cmder
3. chrome
4. clash
5. NodeJS, Yarn

### 1.1.1 必备软件配置

任何软件都需要配置

#### vscode

1. 必备插件
  - Chinese (Simplified) Language Pack for Visual Studio Code
  - Code Spell Checker
  - Git Easy
  - Latex Workshop
  - Markdown All in One
  - Prettier - Code formatter
2. 环境和快捷键配置
  - 具体看我知乎保存的 setting.json 和 keybinding.json
3. 快捷键
  - ctrl + p - 找文件
  - ctrl + shift + p 或 F1 - 输入命令
  - alt + 单击 - 多位置输入

## cmder

1. 配置内容较多，直接看我保存的配置文件。
2. 将 cmder 加入右键菜单，加入环境变量。

## chrome

1. 可选插件
  - Proxy SwitchOmega
  - uBlock
2. 高级用户配置
  - 在开发者工具[里面](#)按 ESC 可以新建控制台
  - Sources 面板可以保存代码片段 (Snippets)
  - Network 关掉 show overview, filter 可以搜索，右击勾选 Method
  - Network 可模拟慢网速/断网
  - Preserve log 不会清空当前请求数据
  - Disable cache 清除缓存
3. chrome 常用快捷键
  - 鼠标中键单击 - 打开或关闭
  - ctrl + T - 新开标签
  - ctrl + shift + T - 撤销关闭
  - ctrl + 点击 - 在新标签打开
  - ctrl + W - 关闭当前标签
  - ctrl + Reload 或者 F5 - 刷新
  - ctrl + Location - 输入网址
  - ctrl + shift + Inspector 或 F12 - 打开开发者工具
  - alt + 左右 - 前进后退
  - alt + 回车 - 在新标签打开
  - shift + 回车 - 在新窗口打开
  - ctrl + shift + delete - 删除历史浏览数据

## windows

1. 关掉任务栏无用标签，卸载无用软件。(如搜索框，任务视图，微软小娜 Cortana，开始菜单里的各种贴图)
2. 可安装 TranslucentTB 使任务栏透明
3. 理解用户目录，即 C:\Users\Jony，分别右击用户目录里的下载和文件，属性-> 位置-> 移动到 E 盘。
4. 显示文件后缀，打开查看-> 选项-> 查看，去掉隐藏已知文件扩展名，勾选显示已知文件，文件夹和驱动器。
5. 记住组合键
  - Win 组合键

- (a) win + Desktop - 展示桌面
  - (b) win + 方向键 - 移动窗口
  - (c) alt + tab - 切换窗口
  - (d) win + tab - 不怎么常用的切换窗口
  - (e) win + ctrl + 方向键 - 切换桌面
- Ctrl 组合键
    - (a) ctrl + All/ctrl + Copy/ctrl + V/ctrl + Z/ctrl + Y
    - (b) ctrl + Reload/F5
    - (c) ctrl + P - 打印

## NodeJS, Yarn

### 1. 开发必装的东西

- nrm
- tlldr

## 2 Git 入门

### 2.1 命令行入门

### 2.2 本地仓库

### 2.3 远程仓库

## 3 HTML

### 3.1 概览

### 3.2 标签

### 3.3 重难点

### 3.4 实践和手机调试

## 4 CSS

### 4.1 基础

#### 4.1.1 语法

#### 4.1.2 Border 调试法

#### 4.1.3 文档流

##### 1. 文档流的基本概念

- 流动方向
  - inline 元素从左到右，到达最右边才会换行
  - block 元素从上到下，每一个都另起一行
  - inline-block 也是从左到右

- 宽度
  - inline 宽度为内部 inline 元素的和，不能用 width 指定
  - block 默认自动计算宽度，可用 width 指定
  - inline-block 结合前两者特点，可用 width
- 高度
  - inline 高度由 line-height 间接确定，跟 height 无关
  - block 高度由文档流元素决定，可以设 height
  - inline-block 跟 block 类似，可以设置 height

## 2. overflow 溢出

- 当内容大于容器
  - 等内容的宽度或高度大于容器，会溢出
  - 可用 overflow 来设置是否显示滚动条
  - auto 是灵活设置
  - scroll 是永远显示
  - hidden 是直接隐藏溢出部分
  - visible 是直接显示溢出部分
  - overflow 可以分为 overflow-x 和 overflow-y

## 3. 脱离文档流

- 有些元素可不在文档流
  - 原因是 block 高度由内部文档流元素决定，可以设 height
- 以下元素脱离文档流
  - float
  - position:absolute/fixed
- 不用以上属性就不脱离文档流

### 4.1.4 盒模型

1. 两种
  - content-box 内容盒 - 内容就是盒子的边界
  - border-box 边框盒 - 边框才是盒子的边界
2. 公式
  - content-box width = 内容宽度
  - border-box width = 内容宽度 + padding + border
3. 哪个好用？
  - border-box 好用
  - 因为可以同时指定 padding, width, border

#### 4.1.5 margin 合并

##### 1. 哪些情况会合并

- 父子 margin 合并
- 兄弟 margin 合并

##### 2. 如何阻止合并

- 父子合并用 padding/border 挡住
- 父子合并用 overflow:hidden 挡住
- 父子合并用 display:flex
- 兄弟合并是符合预期的
- 兄弟合并可以用 inline-block 消除
- css 属性逐年增多，每年都有新的，死记就完事了

#### 4.1.6 基本单位

##### 1. 长度单位

- px 像素
- em 相对于自身 font-size 的倍数
- 百分数
- 整数
- rem
- vw 和 vh

##### 2. 颜色

- 十六进制 #FF6600 或者 #F60
- RGBA 颜色 rgb(0,0,0) 或者 rgba(0,0,0,1)
- hsl 颜色 hsl(360,100%, 100%)

#### 4.1.7 练手项目

彩虹 demo

## 4.2 布局

### 4.2.1 布局分类

##### 1. 两种

- 固定宽度布局，一般宽度为 960/1000/1024px
- 不固定宽度布局，主要靠文档流的原来布局

##### 2. 回顾

- 文档流本来就是自适应的，不需要加额外的样式

##### 3. 响应式布局

- PC 上固定宽度，手机上不固定宽度
- 也就是一种混合布局

#### 4.2.2 两种布局思路

##### 1. 从大到小

- 先定下大局
- 然后完善每个部分的小布局

##### 2. 从小到大

- 先完成小布局
- 然后组合成大布局

##### 3. 两种均可

- 新人推荐第二种，因为小的简单
- 老手一般用第一种，因为熟练有大局观

#### 4.2.3 float 布局

一图流 (图片以后贴出)

##### 1. float 布局

- 子元素加上 float:left 和 width
- 在父元素上加.clearfix

##### 2. float 布局经验

- 留一些空间或者最后一个不设 width
- 不需要做响应式，因为手机上没有 IE，而这个布局是专门为 IE 准备的
- 解决 IE6/7 存在的双倍 margin bug 如下
  - 一是将错就错，针对 IE6/7 把 margin 减半
  - 二是神来一笔，再加一个 display:inline-block

#### float 布局实践

##### 1. 不同布局

- 用 float 做两栏布局 (如顶部条)
- 用 float 做三栏布局 (如内容区)
- 用 float 做四栏布局 (如导航)
- 用 float 做平均布局 (如产品展示区)
- 

##### 2. 实践经验

- 加上头尾，即可满足所有 PC 页面需求
- 手机页面傻子采用 float
- float 要程序员自己计算宽度，不灵活
- float 用来应付 IE 足矣

#### 4.2.4 Flex 布局

##### 1. 重点

- display: flex
- flex-direction: row/column
- flex-wrap: wrap
- justify-content: center/space-between
- align-items: center

##### 2. 颜色

- 十六进制 #FF6600 或者 #F60
- RGBA 颜色 rgb(0,0,0) 或者 rgba(0,0,0,1)
- hsl 颜色 hsl(360,100%, 100%)

##### 3. 实践

- 用 flex 做两栏布局
- 用 flex 做三栏布局
- 用 flex 做四栏布局
- 用 flex 做平均布局
- 用 flex 组合使用，做更复杂的布局
- 

##### 4. 经验

- 永远不要把 width 和 height 写死，除非特殊说明
- 用 min-width/max-width/min-height/max-height
- flex 可以基本满足所有需求
- flex 和 margin-xxx: auto 配合有意外的效果

##### 5. 什么是写死

- width:100px

##### 6. 不写死

- width:50%
- max-width:100px
- width:30vw
- min-width:80%
- 特点：不使用 px，或者加 min max 前缀

#### 4.2.5 Grid 布局

二维布局用 Grid，一维布局用 Flex

语法

## 例子和语法

### 4.3 定位

布局与定位的区别是：布局是屏幕平面上的，定位是垂直于屏幕的

#### 4.3.1 一个 div 的分层

#### 4.3.2 position 的五个取值

#### 4.3.3 层叠上下文

### 4.4 动画

#### 4.4.1 动画的原理

#### 4.4.2 浏览器渲染的原理

### 浏览器渲染过程

1. 根据 HTML 构建 HTML 树 (DOM)
2. 根据 CSS 构建 CSS 树 (CSSOM)
3. 将两棵树合并成一颗渲染树 (render tree)
4. Layout 布局 (文档流，盒模型，计算大小和位置)
5. Paint 绘制 (把边框颜色，文字颜色，阴影等画出来)
6. Compose 合成 (根据层叠关系展示画面)

### 三棵树 图片以后放

如何更新样式 一般我们采用 JS 来更新样式

1. 比如 `div.style.background='red'`
2. 比如 `div.style.display='none'`
3. 比如 `div.classList.add('red')`
4. 比如 `div.remove()` 直接删掉节点

### 三种更新方式

1. JS/CSS > 样式 > 布局 > 绘制 > 合成
2. JS/CSS > 样式 > 绘制 > 合成
3. JS/CSS > 样式 > 合成

## 三种更新方式区别

### 1. 第一种，全走

- div.remove() 会触发当前消失，其他元素 relayout
- 

### 2. 第二种，跳过 layout

- 改变背景颜色，直接 repaint+composite
- 

### 3. 第三种，跳过 layout 和 paint

- 改变 transform，只需 composite
- 注意必须全屏查看效果，在 iframe 里看有问题
- 

## 4.4.3 CSS 动画优化

### JS 优化

#### 1. 使用 requestAnimationFrame 代替 setTimeout 或 setInterval

### JS 优化

#### 1. 使用 will-change 或 translate

### 参考文章

## 4.4.4 transition

位移 translate 缩放 scale 旋转 rotate 倾斜 skew

### 经验

1. 一般都不需要配合 transition 过度
2. inline 元素不支持 transform，需要先变成 block

### translate

#### 1. 常用写法

- translateX(<length-percentage>)
- translateY(<length-percentage>)
- translate(<length-percentage>, <length-percentage>?)
- translateZ(<length>) 且父容器 perspective
- translate3d(x,y,z)
- 演示

#### 2. 经验

- 看懂 MDN 语法示例
- translate(-50%, -50%) 可做绝对定位元素的居中

## scale

### 1. 常用写法

- scaleX(<number>)
- scaleY(<number>)
- scale(<number>, <number>?)
- 演示

### 2. 经验

- 用的少

## rotate

### 1. 常用写法

- rotate([<angle>|<zero>])
- rotateZ([<angle>|<zero>])
- rotateX([<angle>|<zero>])
- rotateY([<angle>|<zero>])
- rotate3d 太复杂
- 演示

### 2. 经验

- 一般用于 360 度选择制作 loading
- 用到的时候查 rotate MDN 文档

## skew

### 1. 常用写法

- skewX([<angle>|<zero>])
- skewY([<angle>|<zero>])
- skew([<angle>|<zero>],[<angle>|<zero>]?)
- 演示

### 2. 经验

- 用的较少
- 用到的时候查 skew MDN 文档

## transform 多重效果

### 1. 组合使用

- transform:scale(0.5) translate(-100%, -100%);
- transform:none; 取消所有

## 参考文章

#### 4.4.5 transition 过渡

作用是补充中间帧

#### 4.4.6 红心实践

css 需要想象力

## 5 HTTP

Hyper Text Transfer Protocol

### 5.1 URL

Uniform Resource Locator

协议 + 域名或 IP+ 端口号 + 路径 + 查询字符串 + 锚点

#### 5.1.1 IP

Internet Protocol

1. 约定了两件事

- 如何定位一台设备
- 如何封装数据报文，以跟其他设备交流

2. 外网 IP

- 从电信租用带宽，一年一千多。
- 买了路由器，然后用电脑和手机分别连接路由器广播出来的无线 WIFI。
- 路由器连上电信服务器，路由器有一个外围 IP，这是你互联网中的地址。
- 重启路由器可能会被重新分配外围 IP，也就是路由器没有固定的外网 IP。
- 连接路由器的手机和电脑是内网 IP。

3. 内网 IP

- 路由器会在家里创建一个内网，内网设备使用内网 IP，一般是 192.169.xxx.xxx。
- 一般路由器会给自己分配一个好记的内网 IP，如 192.168.1.1。
- 然后路由器会给每一个内网中的设备分配不同的内网 IP。
- 如电脑是 192.168.1.2，手机是 192.168.1.3。

4. 路由器的功能

- 现在路由器会有两个 IP，一个外网 IP 和一个内网 IP。
- 内网的设备可以互相访问，但是不能直接访问外网。
- 内网设备想要访问外围，就必须经过路由器中转。
- 外网中的设备可以互相访问，但是无法访问你的内网。
- 外网设备想要把内容送到内网，也必须通过路由器。
- 也就是说内网和外网就像两个隔绝的空间，无法互通，唯一的联通点就是路由器。

- 所以路由器有时候也被叫做网关。

## 5. 几个特殊的 IP

- 127.0.0.1 表示自己。
- localhost 通过 hosts 指定为自己。
- 0.0.0.0 不表示任何设备。

### 5.1.2 端口

一台机器可以提供很多服务，每个服务一个号码，这个号码就叫端口号 port

#### 1. 一个比喻

- 麦当劳提供两个窗口，一号快餐，二号咖啡。
- 你去快餐窗口点咖啡会被拒绝，让你去两一个窗口。
- 你去咖啡窗口点快餐结果一样。

#### 2. 一台机器可以提供不同服务

- 要提供 HTTP 服务最好使用 80 端口。
- 要提供 HTTPS 服务最好使用 443 端口。
- 要提供 FTP 服务最好使用 21 端口。
- 一共有 65535 个端口 (基本够用)。

#### 3. 端口使用规则

- 0 到 1023(2 的 10 次方减 1) 号端口是留给系统使用的。
- 你只有拥有了管理员权限后，才能使用这 1024 个端口。
- 其他端口可以给普通用户使用。
- 比如 http-server 默认使用 8080 端口。
- 一个端口如果被占用，你就只能换一个端口。

### 5.1.3 域名

#### 1. 域名就是 IP 的别称

- baidu.com 对应的什么 IP -> ping baidu.com
- qq.com 对应的什么 IP -> ping qq.com
- 一个域名可以对应不同 IP。
- 这个叫做均衡负载，防止一台机器扛不住。
- 一个 IP 可以对应不同域名。
- 这个叫做共享主机，穷开发者会这么做。

#### 2. 域名和 IP 是怎么对应起来的

- 通过 DNS

#### 3. 当你输入 qq.com 的过程

- 你的 Chrome 浏览器会向电信提供的 DNS 服务器询问 qq.com 对应什么 IP。

- 电信会回答一个 IP(具体过程很复杂，不研究)。
- 然后 Chrome 才会向对应 IP 的 80/443 端口发送请求。
- 请求内容是查看 qq.com 的首页。

#### 4. 为什么是 80 或 443 端口

- 服务器默认用 80 提供 http 服务。
- 服务器默认用 443 提供 https 服务。
- 可以在开发者工具看到具体的端口。

#### 5. 题外话

- www.caosiyuan.com 和 caosiyuan.com 不是同一域名。
- comn 是顶级域名。
- caosiyuan.com 是二级域名 (俗称一级域名)。
- www.caosiyuan.com 是三级域名 (俗称二级)。
- 他们是父子关系
- 比如 github.io 把子域名 xx.github.io 免费给你使用
- 但 www.caosiyuan.com 和 caosiyuan.com 可以不是同一家公司，也可以是。
- www 非常多余

#### 6. 如何请求不同的页面

- 路径可以做到
- <https://developer.mozilla.org/zh-CN/docs/Web/HTML>
- <https://developer.mozilla.org/zh-CN/docs/Web/CSS>
- 使用 chrome 开发者工具 Network 面板看区别。
- 有点类似爬虫找规律。

#### 7. 同一个页面，不同内容

- 查询参数可以做到
- <http://www.baidu.com/s?wd=hi>
- <http://www.baidu.com/s?wd=hello>

#### 8. 同一个页面，不同位置

- 锚点可以做到
- <https://developer.mozilla.org/zh-CN/docs/Web/CSS#>
- <https://developer.mozilla.org/zh-CN/docs/Web/CSS#>
- 注意，锚点看起来有中文，但实际不支持中文。
- # 参考书会变成 #E5%8F...。
- 锚点是无法在 Network 面板看到。
- 锚点不会传给服务器。

#### 5.1.4 HTTP 协议

基于 TCP 和 IP 两个协议，规定了请求的格式是什么，响应的格式是什么

##### 1. 用 curl 可以发 HTTP 请求

- curl -v http://baidu.com
- curl -s -v https://www.baidu.com

##### 2. 理解一下概念

- url 会被 curl 工具重写，先请求 DNS 获得 IP
- 先进行 TCP 连接，TCP 连接成功后，开始发送 HTTP 请求
- 请求内容看一眼
- 响应内容看一眼
- 响应结束后，关闭 TCP 连接（看不出来）
- 真正结束

## 5.2 请求响应和 NodeJS Sever

# 6 JS

## 6.1 概览

JS 需要一点逻辑能力，数学学的好不用担心，因为比数学简单太多了。

### 6.1.1 硬要求

#### 1. 足够的代码量

- 达到 1000 行 -> 新手
- 达到 10000 行 -> 熟手
- 达到 50000 行 -> 专业选手
- 只能靠时间积累，人生就是奋斗，最快一年就可达到。

#### 2. 如何统计自己的代码行数

- 安装 yarn global add cloc
- 在项目文件下使用 cloc --vcs=git.
- 注意把仓库里 node\_modules 等不想关内容写入.gitignore

#### 3. 了解最够多的概念，不仅会写，还要会说

- 常用考点：闭包，原型，类，继承，MVC，Flux，高阶函数，前端工程化
- 博客总结，代码实践，多多积累。

#### 4. 有足够的踩坑经验

- 把该领域内所有的错误都犯完的人，就是专家。
- 多做个人项目，全方位踩坑。

## 6.1.2 JS 的历史

### 1. JavaScript 的诞生

- 布兰登生平自行了解，我的总结是成为了领导后千万不能犯错。
- 牛逼的程序员不怕辞退，很容易创业，可以干到 50 岁以上。
- 公司要求 JS 的命名蹭 Java 的流量，现在各行各业也存在者这种营销。
- 由于版权问题，JS 又叫 ECMAScript。
- 布兰登十天设计了 JS 最初版本（不是实现），所以 JS 有很多 bug。
- 网景被微软收购，IE6 如日中天。
- 2004 谷歌雇佣了一些 Firefox 和 IE 的开发者。
- 2016 年 Chrome 全球份额 62%，横空出世。
- 移动市场智能手机的崛起。

### 2. JavaScript 的兴起

- 2004 年愚人节，谷歌发布 Gmail 在线网页，当时人们认为网页只能看新闻和图片。
- 2005 年，Jesse 将谷歌用到的技术命名为 AJAX，从此，前端技术正式出现。
- 用历史唯物主义的观念看，正如现在很多前端概念就是过去技术的打包。
- 在此之前的网页开发都是由后端和设计师完成。
- 2006 年，jQuery 发布，是目前最长寿的 JS 库。
- 后来的十年，jQuery 大放异彩，直到 IE 不行了，才稍微没有那么火。

### 3. 中国的前端

- 正式出现时间是 2010 年左右，中国才有专门的前端岗位。
- 可以用百度指数关键词搜索趋势。
- 早期的前端是一些自学前端的后端程序员，他们把 Java 思想带入 JS
- 因此面向对象成了 JS 的主流思想。
- 行业还是很缺前端。

### 4. JavaScript 的爆发

- Chrome 的 JS 引擎叫做 V8，V8 原本是跑车引擎的叫法，快如闪电。
- 2009 年，Ryan 基于 V8 创建了 Node.js。
- 2010 年，Isaac 基于 Node.js 写出了 npm。
- 前端工程师可以在浏览器之外执行 JS 了，Node.js 快速风靡。
- 同年，TJ 受 Sinatra 启发，发布了 Express.js。
- 至此，前端工程师可以愉快的写后端应用了。
- 至此，爆发了很多技术了，gulp，grunt，yeoman，requireJs，webpack 等。
- JS 是历史的选择，一开始是玩具，但 JS 走对了风口，所以活到了最后。
- 总结：类似考研政治，历史人物可以影响事物的进程，但决定不了历史的发展的方向。

## 6.2 内存图与 JS 世界

### 6.2.1 操作系统常识

一切都运行在内存里

#### 1. 开机

- 操作系统在 C 盘里 (macOS 的在根目录下多个目录里)
- 当按下开机键，主板通电，开始读取固件
- 固件就是固定在主板上的存储设备，里面有开机程序
- 开机程序会将文件里的操作系统加载到内存中运行

#### 2. 操作系统 (以 Linux 为例)

- 首先加载操作系统内核
- 然后启动初始化进程，编号为 1，每个进程都有编号
- 启动系统服务：文件，安全，联网
- 等待用户登录：输入密码登录/ssh 登录
- 登录后，运行 shell，用户就可以和操作系统对话了
- bash 是一种 shell，图形化界面可认为是一种 shell

#### 3. 打开浏览器 (chrome.exe)

- 你双击 Chrome 图标，就会运行 chrome.exe 文件
- 开启 Chrome 进程，作为主进程
- 主进程会开启一些辅助进程，如网络服务，GPU 加速
- 你每新建一个网页，就有可能会开启一个子进程

#### 4. 浏览器的功能

- 发起请求，下载 HTML，解析 HTML，下载 CSS，解析 CSS
- 渲染界面，下载 JS，解析 JS，执行 JS 等
- 功能模块：用户界面，渲染引擎，JS 引擎，存储等
- 以上功能模块一般各处于不同的线程 (比进程更小)
- 如果进程是车间，那么线程就是车间里的流水线

#### 5. JS 引擎

- Chrome 用的是由 C++ 编写的 V8 引擎
- 网景用的是 SpiderMonkey，后被 Firefox 使用
- Safari 用的是 JavaScriptCore
- IE 用的是 Chakra(JScript9)
- Edge 用的是 Chakra(JavaScript)
- Node.js 用的是 V8 引擎

#### 6. JS 引擎的功能

- 编译：把 JS 代码翻译为机器能执行的字节码或机器码

- 优化：改写代码，使其更高效
- 执行：执行上面的字节码或者机器码
- 垃圾回收：把 JS 用完的内存回收，方便之后再次使用

## 7. 执行 JS 代码的准备工作

- 浏览器提供 API: window/document setTimeout
- 没错，上面东西都不是 JS 自身具备的功能
- 我们将这些功能称为运行环境 runtime env
- 一旦把 JS 放进页面，就开始执行 JS
- JS 代码在内存里运行，看下部分内存图

### 6.2.2 内存图

要求会画内存图

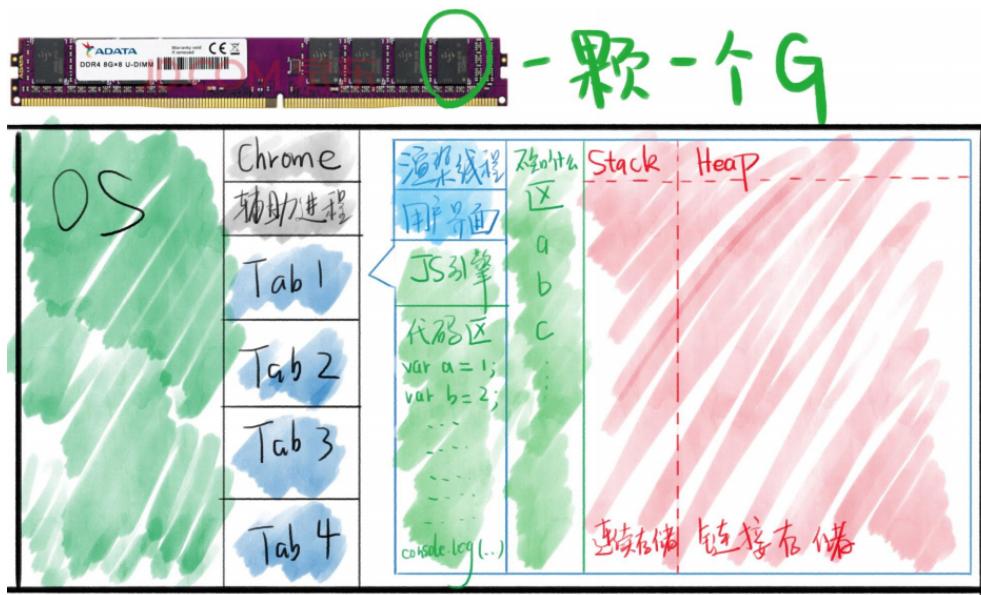


图 1: 瓜分内存图

#### 1. 红色区域的作用

- 红色专门用来存放数据，我们目前只研究该区域
- 红色区域并不存变量名，变量名在不知什么区
- 每种浏览器的分配规则并不一样
- 还有调用栈，任务队列尚未画出

#### 2. Stack 和 Heap

- 红色区域分为 Stack 栈和 Heap 堆
- 栈和堆需要用到数据结构知识
- Stack 区特点：每个数据顺序存放
- Heap 区特点：每个数据随机存放

#### 3. js 代码在 Heap 和 Stack 区的执行过程

```

var a = 1
var b = a
var person = {name: 'syuancao', hobby: 'coding'}
var person2 = person

```

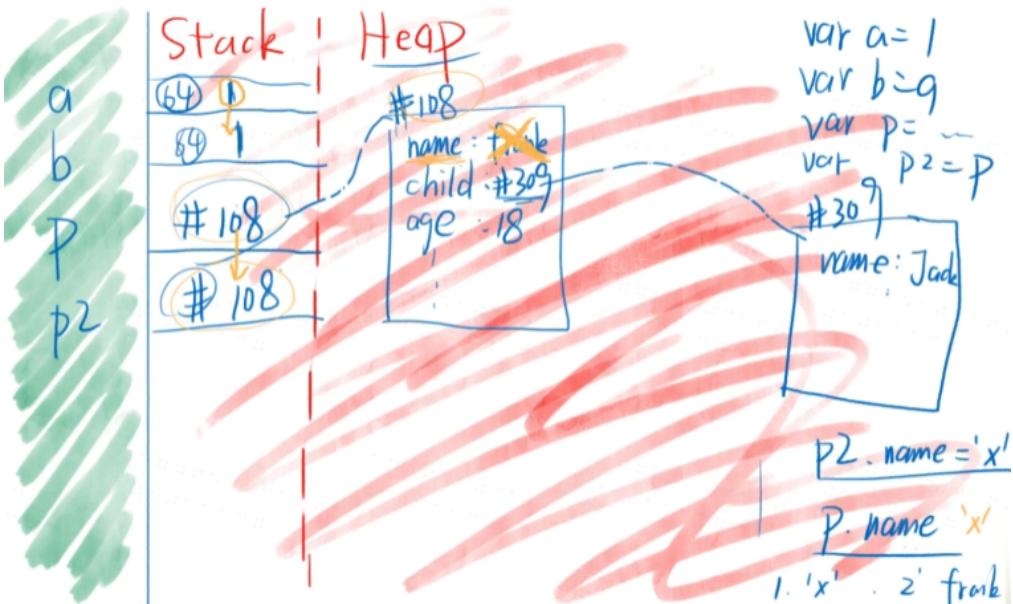


图 2: js 代码在 Heap 和 Stack 区执行的内存图

#### 4. 规律

- 数据分两种：非对象和对象
- 非对象都存在 Stack (数字, 字符串, 布尔不是对象)
- 对象都存在 Heap (数组是对象, 函数是对象)
- = 号总是会把右边的东西复制到左边 (不存在什么传值和传址, 是直接拷贝, 图中箭头指向是虚的)
- 很多书上会让你区分值和地址, 只有不会画内存图的人才需要做这件事

#### 5. 对象被篡改, 结合内存图很好分析

```

var person = {name:'caosiyuan'}
var person2 = person
person2.name = 'syuancao'
console.log(person.name) // syuancao

```

### 6.2.3 JS 的世界是怎样的

神说要有光, 就有了光, JS 开发者说要有 window, 就有了 window(浏览器提供)

#### 1. JS 世界还需要什么

- 要有 console, 并且挂到 window 上
- 要有 document, 并且挂到 window 上
- 要有对象, 于是就有了 Object, 并且挂到 window 上
- var person = {} 等价于 var person = new Object()
- 要有数组 (一种特殊的对象), 于是有了 Array, 并且挂到 window 上
- var a = [1, 2, 3] 等价于 var a = new Array(1, 2, 3)

- 要有函数（一种特殊的对象），于是有了 Function，并且挂到 window 上

- function f(){} 等价于 var f = new Function()

## 2. 题外话

- 为什么有 var a = []，还要提供 var a = new Array() 呢
- 因为后者是正规写法，但是没人用，前者不正规，但是好用
- 为什么有 function f(){}，还要提供 var f = new Function() 写法呢
- 原因同上

## 3. 把 window 用内存图画出来

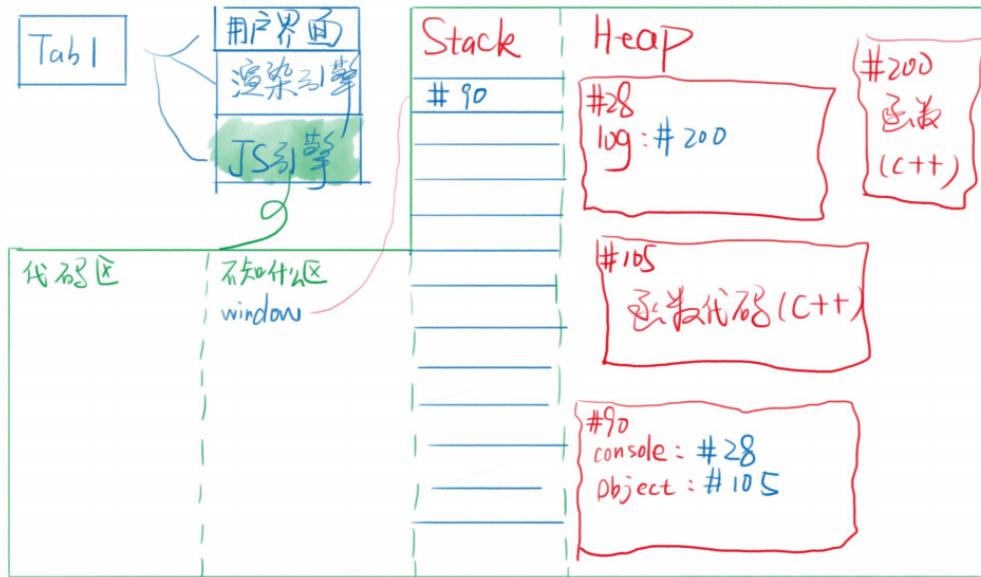


图 3: window 内存图

## 4. 更简单的画法

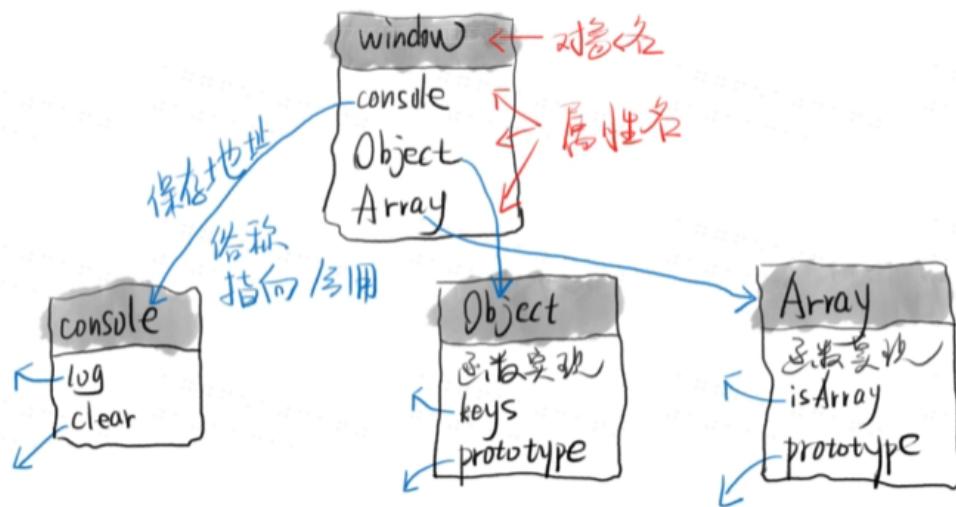


图 4: 更简单的 window 内存图

- 可以用 console.dir(window.Array) 看属性
- 如果第一个字母是大写比如 Object, Array，那么会就有 prototype 属性

## 5. 细节

- window 变量和 window 对象是两个东西
- window 变量是一个容器，存放 window 对象的地址
- window 对象是 Heap 里的一块数据
- 不信的话，可以让 var x = window，那么这个 x 就指向 window 对象，window 变量就可以去死了
- 但这样的代码会弄晕新手，所以不要这样写
- 但是 jQuery 就是这样挂到 window 上的函数，`window.jQuery=function(){}`
- 但是 jQuery 我们平时用 \$，用 \$ 去调用，`var $ = jQuery, $(())`
- 同理，`console(属性)` 和 `console` 对象不是同一个东西
- Object 和 Object 函数对象不是同一个东西
- 前者是内存地址，后者是内存对应的一块数据也就是一块内存

### 6.2.4 原型链

是 JS 里最重要的，也是新手最难懂的之一（注：JS 有三个最难懂的，分别是 this，原型，AJAX）

#### 1. 内存图里的 prototype 是干什么用的

- 可以打印出来看看，`console.dir(window.Object.prototype)`，window 可以省略
- 只是看起来是一块无用函数

#### 2. `var obj={} obj.toString()` 为什么不报错？为什么可以运行？

- obj 有一个隐藏属性
- 隐藏属性存储了 `Object.prototype` 对象的地址
- `obj.toString()` 发现 obj 上没有 `toString`
- 就去隐藏属性对应的对象里面找
- 于是就找到了 `Object.prototype.toString` 里面的 `toString`
- 也就是 `obj.toString === window.prototype.toString`

#### 3. 类似的 `var arr=[] arr.join('?)` 为什么不报错？为什么可以运行？

- arr 有一个隐藏属性
- 隐藏属性存储了 `Array.prototype` 对象的地址
- `arr.join()` 发现 arr 上没有 `join`
- 就去隐藏属性对应的对象里面找
- 于是就找到了 `Array.prototype.join` 里面的 `join`
- 也就是 `Array.prototype.join === window.prototype.join`

#### 4. JS 的光

- 下面一张图可以解释上面的问题

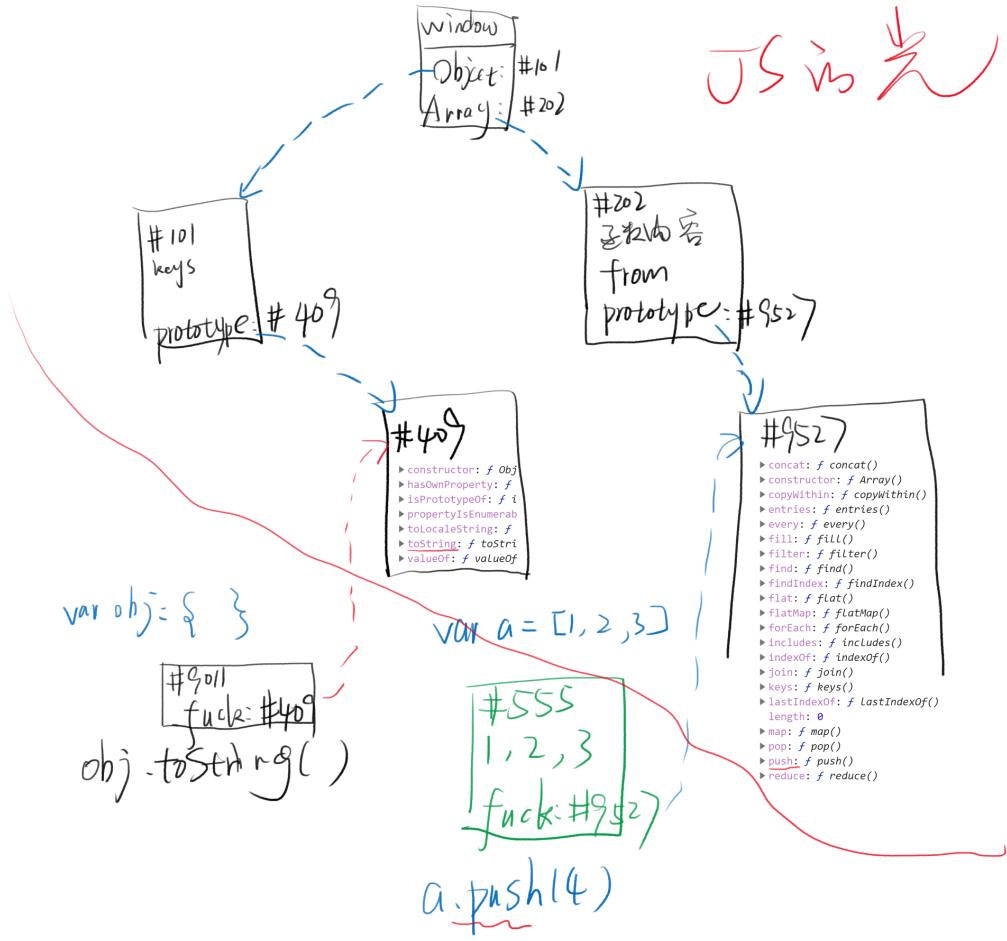


图 5: JS 的光

5. var obj2={} obj2.toString() obj 和 obj2 有什么联系

- 相同点: 都调用.toString
- 不同点: 地址不同 obj!==obj2, 可以拥有不同的属性
- obj.prototype 存储了 obj 对象的共同属性, 这就是原型

6. 原型的好处

- 如果没有原型, 声明一个对象

```

var obj = {
  toString: window.Object.prototype.toString,
  hasOwnProperty: window.Object.....
}
obj.toString()
var obj2 = {
  toString: window.Object.prototype.toString,
  hasOwnProperty: window.Object.....
}
obj2.toString()

```

- 你是不是想累死自己
- 原型让你无需重复声明共有属性, 省代码, 省内存

7. 关于隐藏属性 \_\_proto\_\_

- 每个对象都有一个隐藏属性, 用来保存其原型的地址, 这个隐藏属性的名字叫做 \_\_proto\_\_

- 大写的不要关心隐藏属性，这涉及 js 哲学问题不用关心，关心小写的隐藏属性
- 如果没有隐藏属性，obj 就不知道共有属性在哪，就没法调用 `toString` 等

## 8. prototype 和 `_proto_` 的区别是什么

- 都存着原型的地址，即相同的地址
- 只不过 `prototype` 挂在函数上，通常是大写的 (`Array`, `Object`, `Function`) 上面
- `_proto_` 挂在每个新生成的对象上，也就是小写的 (`var a = {}`, `var b = []`) 上面

## 9. 犀利的提问

- 类似之前提过篡改对象的例子

```

var obj = {}
var obj2 = {}

obj.toString === obj2.toString // 输出为 true
obj.toString = 'fuck' // 输出为 "fuck"
obj.toString // 也等于 'fuck' 吗?

```

- 不废话一图解决问题

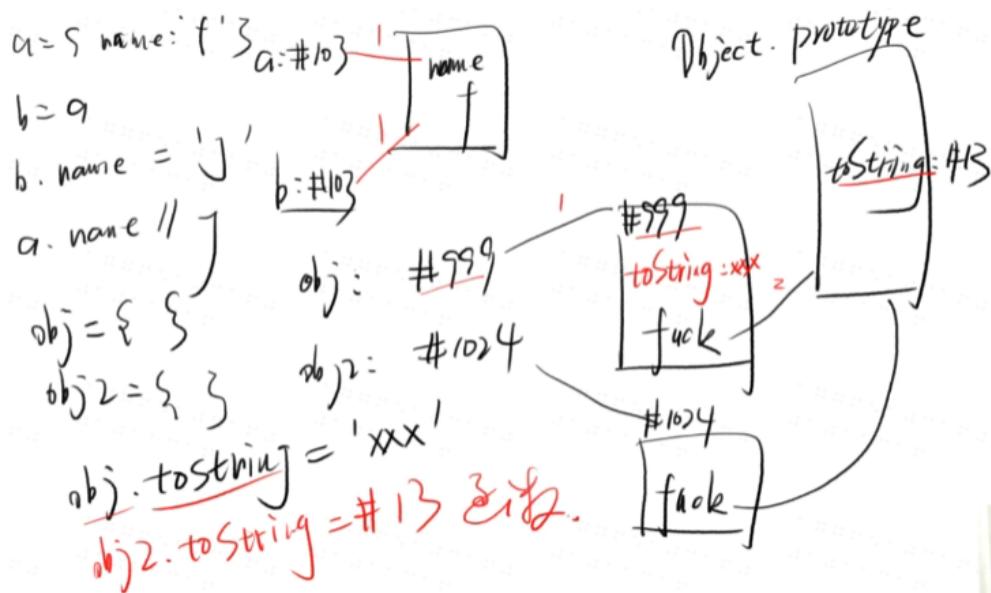


图 6: `toString` 篡改

- 原因解释
  - 这个和之前的不一样，这里 `toString` 是隐藏属性
  - 结合上图相当于写了两层
  - 所以一层是可以篡改的，两层就不可以

## 6.3 Canvas 实践—画图板

[项目地址](#)

[预览效果](#)

## 6.4 JS 语法

[es6 是最低要求](#)

### 6.4.1 JS 版本

#### 1. 历史版本

- ES3, IE6 支持, 总体评价: 垃圾
- ES5, 总体评价: 还是垃圾
- ES6, 大部分浏览器支持, 总体评级: 一半垃圾一半好
- ES2019 与 ES6 差别不大

#### 2. 为什么说 ES6 一半垃圾

- 因为 ES 不能删除以前的特性, 需要兼容旧网站
- 也就是说以前能运行的网站, 以后都要能运行
- 对比 Python3 你就能知道兼容的好处: 稳定

#### 3. 一门语言的价值

- 是由其产生的价值决定
- JS 是世界上使用最广的语言
- JS 是门槛极低的语言 (只要你不学糟粕)
- JS 是一门能产生价值的语言 (虽然不美)
- 它的优秀之处并非原创, 它的原创之处并不优秀 – 来自 JS 之父的评价

### 6.4.2 语法

#### 1. 表达式

- `1+2` 表达式的值为 3
- `add(1,2)` 表达式的值为函数的返回值
- `console.log` 表达式的值为函数本身
- `console.log(3)` 表达式的值为多少? (`undefined`)

#### 2. 语句

- `var a = 1` 是一个和语句

#### 3. 二者的区别

- 表达式一般都有值, 语句可能有也可能没有
- 语句一般会改变环境 (声明, 赋值)
- 上面两句话并不是绝对的

#### 4. 大小写敏感

- `var a` 和 `var A` 是不同的
- `object` 和 `Object` 是不同的
- `function` 和 `Function` 是不同的
- 具体含义后面说

#### 5. 空格

- 大部分空格没有实际意义
- var a = 1 和 var a=1 没有区别
- 加回车大部分时候也不影响
- 只有一个地方不能加回车，那就是 return 后面

## 6. 标识符的规则

- 第一个字符，可以是 Unicode 字母或 \$ 或中字
- 后面的字符，除了上所说，还可以有数字
- 变量名是标识符

## 7. 区块 block

- 把代码包在一起
- 常常与 if/for/while 合用

## 8. if 语句

- if(表达式) 语句 1else 语句 2
- {} 在语句只有一句的时候可以省略，但不建议这么做

## 9. if 语句变态情况

- 表达式里可以非常变态，如 a = 1
- 语句 1 里可以非常变态，如嵌套 if else
- 语句 2 里可以非常变态，如嵌套 if else
- 缩进也可以很变态，如面试常常下套

```
a = 1
if (a === 2)
  console.log('a')
  console.log('a等于2')
```

## 10. while 循环

- while(表达式) 语句
- 当表达式为真，执行语句，执行完再判断表达式真假
- 当表达式为假，执行后面的语句
- do ... while 可以做用户输入判定

## 11. for 循环

- for 循环是 while 的方便写法
- for(语句 1; 表达式 2; 语句 3) 循环体
- 不多说了，和 C 语言一样

## 12. label 语句

- 语法

```

foo: {
    console.log(1);
    break foo;
    console.log('本行不会输出');
}
console.log(2);

foo: 1;

{
    foo: 1;
}

```

- 面试：下面的东西是什么？下面是代码块，是 label，不是对象

```

{
    foo: 1
}

```

### 6.4.3 JS 数据

#### 1. 7 种数据类型 (大小写无所谓)

- 数字 number
- 字符串 string
- 布尔 bool
- 符号 symbol
- 空 undefined
- 空 null
- 对象 object
- 总结：四基两空一对象

#### 2. 以下不是数据类型

- 数组，函数，日期
- 它们都属于 object

#### 3. 5 个 falsy 值

- falsy 就是相当于 false 但又不是 false 值
- 分别是 undefined null 0 NaN ”
- ”，即空字符串和空格字符串不是一个玩意

#### 4. undefined 和 null 的区别

- 这是 js 的垃圾之处，没有本质区别
- 如果一个变量声明了，但没有赋值，那么默认值就是 undefined 而不是 null
- 如果一个函数，没有写 return，那么默认 return undefined，而不是 null
- 前端程序员习惯上，把非对象的空值写成 undefined，把对象空值写为 null
- 仅仅是习惯而已

#### 5. let 声明

- 遵循块作用域，即使用范围不能超出
- 不能重复申明
- 可以赋值，也可以不赋值
- 必须先声明再使用，否则报错
- 全局声明的 let 变量，不会变成 window 的属性
- for 循环配合 let 有奇效

## 6. 类型转换

- number => string
  - String(n)
  - n + ”
- string => number
  - Number(s)
  - parseInt(s)/parseFloat(s)
  - s - 0
- x => bool
  - Boolean(x)
  - x.toString()

### 6.4.4 JS 对象

第七种数据类型，唯一一种复杂类型

#### 1. 定义

- 无序的数据集合
- 键值对的集合

#### 2. 写法

```
let obj = {'name': 'caosiyuan', 'age': '27'}
let obj = new Object({'name': 'caosiyuan'})
console.log({'name': 'caosiyuan', 'age': 18})
```

#### 3. 细节

- 键名是字符串，不是标识符，可以包含任意字符
- 引号可以省略，省略之后就只能写标识符
- 就算引号省略了，键名也还是字符串

#### 4. 变量作属性名

- 不加 [] 的属性名会自动变成字符串
- 加了 [] 则会当作变量求值
- 值如果不是字符串，则会自动变成字符串
- 除了字符串，symbol 也能做属性名

#### 5. 对象的隐藏属性

- JS 中每一个对象都有一个隐藏属性
- 这个隐藏属性储存着其**共有属性组成的对象**的地址
- 这个**共有属性组成的对象**叫做原型
- 也就是说，隐藏属性储存着原型的地址

## 6. 删除属性

- `delete obj.xxx` 或 `obj['xxx']`
- 即可删除 `obj` 的 `xxx` 属性，请区分属性值为 `undefined` 和不含属性名
- 不含属性名`'xxx'` in `obj` === false
- 含有属性名，但是值为 `undefined`, `'xxx' in obj && obj.xxx === undefined`
- 注意 `obj.xxx === undefined` 不能断定`'xxx'` 是否为 `obj` 的属性

## 7. 查看所有属性 (读属性)

- 查看自身所有属性 `Object.keys(obj)`
- 查看自身 + 共有属性 `console.dir(obj)`
- 或者自己依次用 `Object.keys` 打印出 `obj.__proto__`
- `obj.hasOwnProperty('toString')` 判断一个属性是自身的还是共有的

## 8. 原型

- 每个对象都有原型，原型里存着对象的共有属性
- 比如 `obj` 的原型就是一个对象
- `obj.__proto__` 存着这个对象的地址
- 这个对象里有 `toString/constructor/valueOf` 等属性
- 对象的原型也是对象，所以对象的原型也有对象
- `obj=` 的原型即为所有对象的原型
- 这个原型包含所有对象的共有属性，是对象的根
- 这个原型也有原型，是 `null`

## 9. 查看属性

- 中括号语法: `obj['key']`
- 点语法: `obj.key`
- 坑新人语法: `obj[key]` // 变量 `key` 值一般不为`'key'`
- 优先使用中括号语法，点语法会误导你，让你以为 `key` 不是字符串

## 10. 修改或增加属性 (写属性)

- 直接赋值 `obj.name = 'caosiyuan'`
- 批量赋值 `Object.assign(obj, age: 27, gender: 'man')`

## 11. 修改或增加共有属性

- 无法通过自身修改或增加共有属性
- `let obj = , obj2 = //共有 toString`

- `obj.toString = 'xxx'` 只会再改 `obj` 自身属性
- `obj.toString` 还是在原型上

## 12. 我偏要修改或增加原型上的属性

- `obj.__proto__.toString = 'xxx' // 不推荐用 __proto__`
- `Object.prototype.toString = 'xxx'`
- 一般来说，不要修改原型，这会引起很多问题

## 13. 修改隐藏属性

- 不推荐用 `__proto__`

```
let obj = {'name': 'caosiyuan'}
let obj2 = {'name': 'syuancao'}
let common = {kind: 'human'}
obj.__proto__ = common
obj2.__proto__ = common
```

- 推荐使用 `Object.create`

```
let obj = Object.create(common)
obj.name = 'caosiyuan'
let obj2 = Object.create(common)
obj2.name = 'syuancao'
```

## 14. 'name' in `obj` 和 `obj.hasOwnProperty('name')` 的区别

- `'name' in obj` 会检查对象隐藏属性即原型链
- `obj.hasOwnProperty('name')` 只会检查自身，即该属性必须是对象本身的成员

## 6.4.5 JS 对象分类

**你可以不会 class，但是一定要学会 prototype**

### 1. 历史版本

- ES3, IE6 支持，总体评价：垃圾
- ES5，总体评价：还是垃圾
- ES6，大部分浏览器支持，总体评级：一半垃圾一半好
- ES2019 与 ES6 差别不大

### 2. 为什么说 ES6 一半垃圾

- 因为 ES 不能删除以前的特性，需要兼容旧网站
- 也就是说以前能运行的网站，以后都要能运行
- 对比 Python3 你就能知道兼容的好处：稳定

### 3. 一门语言的价值

- 是由其产生的价值决定
- JS 是世界上使用最广的语言
- JS 是门槛极低的语言（只要你不学糟粕）
- JS 是一门能产生价值的语言（虽然不美）
- 它的优秀之处并非原创，它的原创之处并不优秀

## 7 算法与数据结构

## 8 JS 编程接口

### 8.1 DOM 编程

#### 8.1.1 DOM 简介

1. 网页其实是一棵树

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <meta http-equiv="X-UA-Compatible" content="ie=edge">
8    <title>标题</title>
9  </head>
10
11 <body>
12   <header>
13     <h1>文字1</h1>
14   </header>
15   <main>
16     <h2>文字2</h2>
17     <p>文字3 <span>文字4</span> 文字5</p>
18   </main>
19 </body>
20
21 </html>
```

图 7: HTML 树 1

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
```

```
19 </body>
20
21 </html>
```

图 8: HTML 树 2

2. 为什么说 ES6 一半垃圾

- 因为 ES 不能删除以前的特性，需要兼容旧网站
- 也就是说以前能运行的网站，以后都要能运行
- 对比 Python3 你就能知道兼容的好处：稳定

### 3. 一门语言的价值

- 是由其产生的价值决定
- JS 是世界上使用最广的语言
- JS 是门槛极低的语言 (只要你不学糟粕)
- JS 是一门能产生价值的语言 (虽然不美)
- 它的优秀之处并非原创，它的原创之处并不优秀

#### 8.1.2 获取元素的 API

#### 8.1.3 元素的 6 层原型链

#### 8.1.4 创建元素的 API

#### 8.1.5 查看元素的 API

#### 8.1.6 DOM 操作跨线程

### 8.2 手写 DOM 库

### 8.3 jQuery 中的设计模式

### 8.4 DOM 事件与事件委托

#### 1. 从点击事件开始研究

- 看如下代码

```
<div class="爷爷">
    <div class="爸爸">
        <div class="儿子">文字</div>
    </div>
</div>
```

即. 爷爷 >. 爸爸 >. 儿子, 给三个 div 分别添加事件监听 fnYe/fnBa/fnEr

- 提问一：点击了谁？

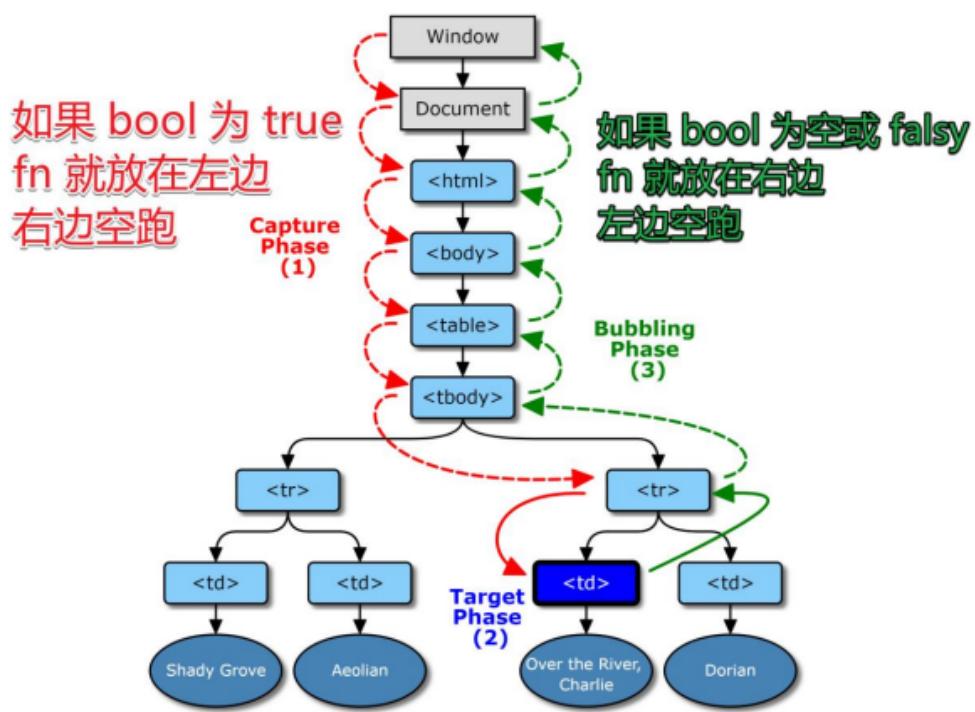
- 点击文字，算不算点击儿子？
  - 点击文字，算不算点击爸爸？
  - 点击文字，算不算点击爷爷？
  - 答案：都算

- 提问二：调用顺序

- 点击文字，最先调用 fnYe/fnBa/fnEr 中的哪一个函数？
  - 答案：都行
  - IE5 认为先调 fnEr，网景认为先调 fnYe，然后搞上了
  - 最后闹到了 W3C

#### 2. 和事佬 W3C

- 2002 年，W3C 发布标准
  - 文档名为 DOM Level 2 Events Specification
  - 规定浏览器应该同时支持两种调用顺序
    - 首先按爷爷 => 爸爸 => 儿子顺序看有没有函数监听
    - 然后按儿子 => 爸爸 => 爷爷顺序看有没有函数监听
    - 有监听函数就调用，并提供事件信息，没有就跳过
- 术语
  - 从外向内找监听函数，叫事件捕获
  - 从内向外找监听函数，叫事件冒泡
- 疑问：那岂不是 fnYe/fnBa/fnEr 都调用两次？非也！
  - 开发者自己选择把 fnYe 放在捕获阶段还是放在冒泡阶段
- 示意图



### 3. addEventListener

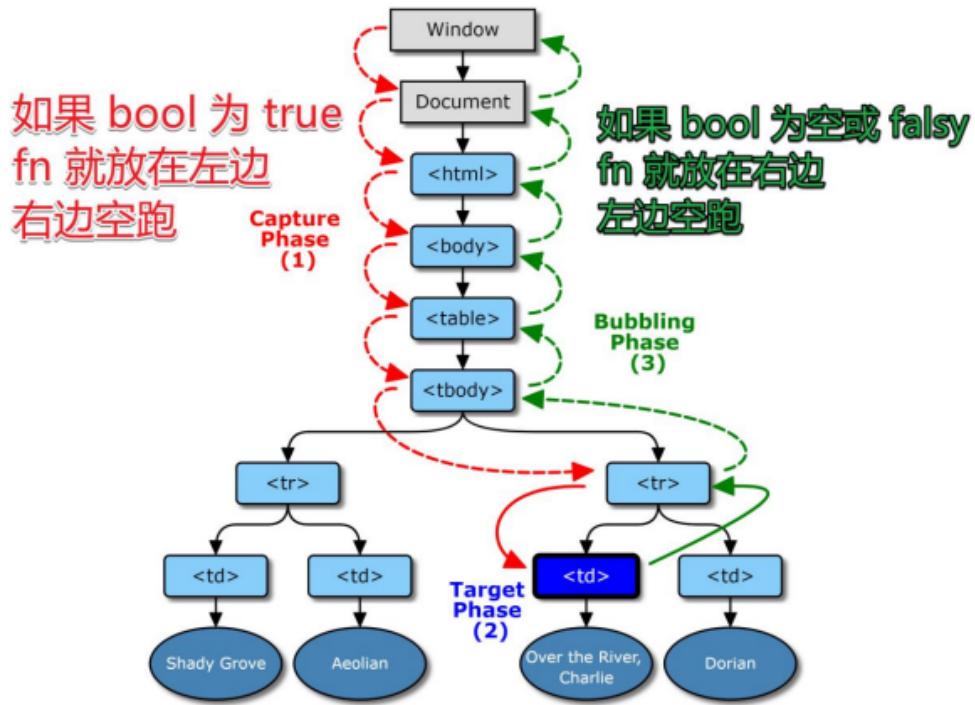
- 事件绑定 API
  - IE5
 

```
baba.attachEvent('onclick', fn) // 冒泡
```
  - 网景
 

```
baba.addEventListener('click', fn) // 捕获
```
  - W3C
 

```
baba.addEventListener('click', fn, bool) // 不填支持ie, 填true支持网景
```
- 如果 bool 不传或为 falsy
  - 就让 fn 走冒泡，即当浏览器在冒泡阶段发现 baba 有 fn 监听函数，就会调用 fn，并提供事件信息
- 如果 bool 为 true
  - 就让 fn 走捕获，即当浏览器在捕获阶段发现 baba 有 fn 监听函数，就会调用 fn，并提供事件信息

#### 4. 你可以选择把 fn 放在哪边



#### 5. 代码示例

The screenshot shows a browser developer tools interface with the following sections:

- HTML:** Displays the HTML code structure.
- JavaScript:** Displays the following JavaScript code:
- Output:** Displays the result of the click event on the target element, showing a colorful target icon.

```
HTML :
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>JS Bin</title>
6   </head>
7   <body>
8     <div class="level1 x">
9       <div class="level2 x">
10      <div class="level3 x">
11        <div class="level4 x">
12          <div class="level5 x">
13            <div class="level6 x">
14              <div class="level7 x">
15                ...
16              </div>
17            </div>
18          </div>
19        </div>
20      </div>
21    </div>
22  </body>
23 </html>
```

```
JavaScript :
1 const level1 = document.querySelector('.level1')
2 const level2 = document.querySelector('.level2')
3 const level3 = document.querySelector('.level3')
4 const level4 = document.querySelector('.level4')
5 const level5 = document.querySelector('.level5')
6 const level6 = document.querySelector('.level6')
7 const level7 = document.querySelector('.level7')
8
9 let n = 1
10
11 level1.addEventListener('click', (e)=>{
12   const t = e.currentTarget
13   setTimeout(()=>{
14     t.classList.remove('x')
15   },n*1000)
16   n+=1
17 })
18 level2.addEventListener('click', (e)=>{
19   const t = e.currentTarget
20   setTimeout(()=>{
21     t.classList.remove('x')
22   },n*1000)
23   n+=1
24 })
25 level3.addEventListener('click', (e)=>{
26   const t = e.currentTarget
```

Output 736px: A colorful target icon with concentric rings in red, yellow, green, and blue.

#### 6. 代码图解

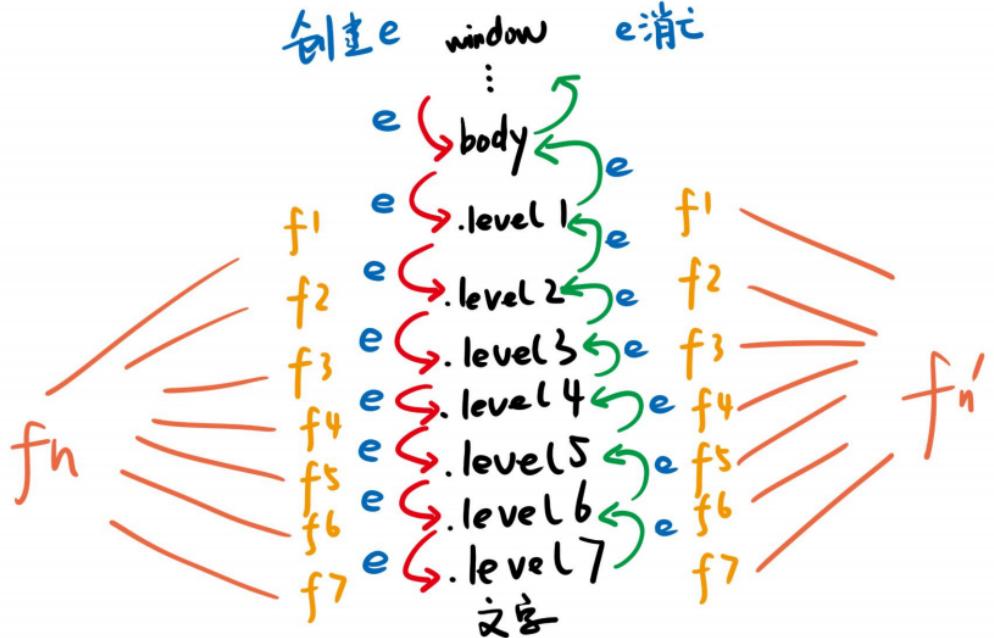


图 9: 先捕获后冒泡

## 7. 小结

- 两个疑问
  - 儿子被点击了，算不算点击老子？(算)
  - 那么先调用老子的函数还是先调用儿子的函数？(不确定，ie 调用儿子，firefox 调用老子)
- 捕获与冒泡
  - 捕获说先调用爸爸的监听函数
  - 冒泡说先调用儿子的监听函数
- W3C 事件模型
  - 先捕获 (先爸爸 => 儿子) 再冒泡 (再儿子 => 爸爸)
  - 注意 e 对象被传给所有监听函数
  - 事业结束后，e 对象就不存在了

## 8. target v.s. currentTarget

- 区别
  - e.target - 用户操作的元素
  - e.currentTarget - 程序员监听的元素
  - this 是 e.currentTarget, 我个人不推荐使用它
- 举例
  - div > span{文字}, 用户点击文字
  - e.target 就是 span
  - e.currentTarget 就是 div

## 9. 一个特例

- 背景
  - 只有一个 div 被监听 (不考虑父子同时被监听)

- fn 分别在捕获阶段和冒泡阶段监听 click 事件
- 用户点击的元素就是开发者监听的
- 代码

```

    div.addEventListener('click', f1)

    div.addEventListener('click', f2, true)
  
```

- 请问, f1 先执行还是 f2 先执行?
- 如果把两行调换位置后, 请问哪个先执行?
- 错误答案: f2 先执行
- 正确答案: 谁先监听谁先执行
- 总结: 这是一个特例

## 10. 取消冒泡

- 捕获不可以取消, 但冒泡可以
- e.stopPropagation() 可中断冒泡, 浏览器不再向上走
- 通俗来说: 有人打我, 我自己解决, 别告诉我老子
- 一般用于封装某些独立的组件

## 11. 不可阻止默认动作

- 有些事件不能阻止默认动作
- MDN 搜索 scroll event, 看到 Bubbles 和 Cancelable
- Bubbles 的意思是该事件是否冒泡, 所有冒泡都可取消
- Cancelable 的意思是开发者是否可以阻止默认事件
- Cancelable 与冒泡无关
- 推荐看 MDN 英文版, 中文版内容不全

The scroll event fires when the document view or an element has been scrolled.

<b>Bubbles</b>	Yes
<b>Cancelable</b>	No
<b>Interface</b>	Event
<b>Event handler property</b>	onscroll

## 12. 插曲: 如何阻止滚动

- scroll 事件不可阻止默认动作
- 阻止 scroll 默认动作没用, 因为先有滚动才有滚动事件
- 要阻止滚动, 可阻止 wheel 和 touchstart 的默认动作
- 注意你需要找准滚动条所在的元素
- 但是如果用鼠标滚, 滚动条还能用, 可用 CSS 让滚动条 width: 0

- CSS 也行
- 使用 overflow:hidden 可以直接取消滚动条
- 但此时 JS 依然可以修改 scrollTop
- [代码示例](#)

## 13. 小结

- target 和 currentTarget
- 一个是用户点击的，一个是开发者监听的
- 取消冒泡
- e.stopPropagation()
- 事件的特性
  - Bubbles 表示是否冒泡
  - Cancelable 表示是否支持开发者取消冒泡
  - 如 scroll 不支持取消冒泡
- 如何禁用滚动
- 取消特定元素的 wheel 和 touchstart 的默认动作，而不是阻止冒泡

## 14. 自定义事件

- 浏览器自带事件
  - 一共 100 多种事件，[列表](#)在 MDN 上
- 提问
  - 开发者能不能在自带事件之外，自定义一个事件
  - 答：可以，见[示例](#)

## 15. 事件委托 (装逼名词)

- 场景一
  - 你要给 100 个按钮添加点击事件，咋办？
  - 答：监听这 100 个按钮的祖先，等冒泡的时候判断 target 是不是这 100 个按钮中的一个
- 场景二
  - 你要监听目前不存在的元素的点击事件，咋办？
  - 答：监听祖先，等点击的时候看看是不是我想要监听的元素即可
- 优点
  - 省监听数 (内存)
  - 可以监听动态元素
- [代码示例](#)

## 16. 封装事件委托

- 要求
  - 写出这样一个函数

```
on('click', '#div1', 'button', fn)
```

- 当用户点击 #div1 里的 button 元素时，调用 fn 函数

- 要求用到事件委托
- 答案一
  - 判断 target 是否匹配的button
  - [代码示例](#)
  - **注：**这样封装其实是错的，如果 button 里包 span，点击的是 span，当前元素不匹配 button，就不执行
  - 所以只能用答案二里的递归操作
- 答案二
  - 递归判断 target/target 的爸爸/target 的爷爷
  - [代码示例](#)
- 整合进 jQuery
  - 有兴趣可以自己实现

```
$('#xxx').on('click', 'button', fn)
```

## 17. JS 支持事件吗？

- [答](#)
- 支持，也不支持。
- DOM 事件不属于 JS 的功能，属于浏览器提供的 DOM 的功能
- JS 只是调用了 DOM 提供的 addEventListener 而已
- [追问](#)
- 如何让 JS 支持事件？请手写一个事件系统
- 目前水平可能写不出来，但可以先思考一段时间

## 9 前端站点导航

### 1. 设计需要用到的网站

- <https://www.figma.com/>
- <https://www.iconfont.cn/>

### 2. 一些细节

- 引入 jQuery 用 <https://www.bootcdn.cn/>
- background-color 会自动填满整个页面

## 10 前后分离

### 10.1 AJAX 的原理

### 10.2 异步与 Promise

### 10.3 跨域，CORS，JSONP

面试必必必问，菜鸟必定不回答

### 1. 跨域关键知识

- 同源策略
- 浏览器故意设计的一个功能限制
- CORS
- 突破浏览器限制的一个方法
- JSONP
- IE 时代的妥协

## 2. 同源的定义

- 源
  - window.origin 或 location.origin 可以得到当前源
- 源 = 协议 + 域名 + 端口号
- 如果两个 url 的
  - 协议
  - 域名
  - 端口号
- 完全一致，那么这两个 url 就是同源
- 举例
  - https://qq.com, https://www.baidu.com 不同源
  - https://baidu.com, https://www.baidu.com 不同源
  - 完全一致才算同源

## 3. 同源策略的定义

- 浏览器规定
- 如果 JS<sub>运行在</sub>源 A 里，那么就只能获取源 A 的数据
- 不能获取源 B 的数据，即不允许跨域
- 举例（省略 http 协议）
  - 假设 syuancao.com/index.html 引用了 cdn.com/1.js
  - 所以 1.js 就只能获取 syuancao.com 的数据
  - 不能获取 1.syuancao.com 或者 qq.com 的数据
  - 这是浏览器的功能！
  - 浏览器故意要这样设计的
  - 同源策略就是不同源的页面之间，不准互相访问数据

## 4. 浏览器这样做的目的是啥？

- 为了保护用户隐私！怎么保护的？

## 5. 如果没有同源策略

- 以 QQ 空间为例
- 源为 https://user.qzone.qq.com
- 假设，当前用户已经登录（用 Cookie，后面讲）

- 假设，AJAX 请求 /friends.json 可获取用户好友列表
- 到目前为止都很正常
- 黑客来了
- 假设你的女神分享了 <https://qzone-qq.com> 给你
- 实际上这个是一个钓鱼网站
- 你点开这个网页，这样网页也请求你的好友列表
- <https://user.qzone.qq.com/friends.json>
- 请问，你的好友列表是不是就把黑客偷偷偷走了？
- 好像是哦.....

## 6. 问题的根源

- 无法区分发送者
- QQ 空间页面里的 JS 和黑客网页里的 JS
- 发的请求几乎没有区别 (referrer 有区别)
- 所以，没有同源策略，任何页面都能偷 QQ 空间的数据
- 甚至支付宝余额！
- 那检查 referer 不就号了？
- 安全原则：安全链条的强度取决于最弱的一环
- 万一这个网站的后端开发工程师是个傻 X 呢
- 所以浏览器应该主动预防这个偷数据的行为
- 总之，浏览器为了用户隐私，设置了严格的同源策略

## 7. 代码演示步骤

- 创建目录
- qq-com 里面有一个 server.js，用来模拟 QQ 空间
- syuancao-com 里面有一个 server.js，用来模拟坏人网站
- qq-com
  - /index.html 是首页
  - /qq.js 是 JS 脚本文件
  - /friends.json 是模拟的好友数据
  - 端口监听为 8888，访问 <http://127.0.0.1:8888>
- syuancao-com
  - /index.html 是首页
  - /syuancao.js 是 JS 脚本文件
  - 端口监听为 9999，访问 <http://127.0.0.1:9999>

## 8. hosts

- 设置本地域名映射
- 让 qq.com 映射到 127.0.0.1
- 就可以访问 <http://qq.com:8888/index.html>

- 让 syuancao.com 映射到 127.0.0.1
- 就可以访问<http://syuancao.com:9999/index.html>
- **设置 hosts**
- 让两个<http://127.0.0.1>变成不同的

## 9. 跨域 AJAX

- **正常使用 AJAX**
- 在 qq.com:8888 里运行的 JS 可以访问 /friends.json
- **黑客偷数据**
- 在 syuancao.com:9999 里运行的 JS 不能访问！
- 浏览器需要**CORS**
- **提问**
- 黑客的请求成功了没有？
- 答：成功了，因为 qq.com 后台有 log
- 黑客拿到了响应没有？
- 答：没有，因为浏览器不给数据给它
- 就没有浏览器不限制跨域么？
- 答：如果不限制，就是浏览器 bug 了，快向浏览器反馈

## 10. 其他新手疑问

- **为什么 a.qq.com 访问 qq.com 也算跨域？**
- 答：因为历史上出现过不同公司的共用域名，这两个不一定是同一网站
- 浏览器为了谨慎起见，认为这是不同的源
- **为什么不同端口也算跨域？**
- 答：原因同上，一个端口一个公司。记住安全链条的强度取决于最弱的一环
- 任何给安全相关的问题都要谨慎对待
- **为什么两个网站的 IP 是一样的，也算跨域？**
- 答：原因同上，IP 可以公用。
- **为什么可以跨域使用 CSS, JS 和图片等？**
- 答：同源策略限制的是数据访问，我们引用 CSS, JS 和图片的时候，其实并不知道其内容，我们只是在引用。不信我问你，你能知道 CSS 的第一个字符是什么吗？
- **现实却是：请问怎么跨域？**
- 面试官会问你，工作中也会遇到

## 11. 解法一：CORS

- **问题根源**
- 浏览器默认不同源之间不能互相访问数据
- 但是 qq.com 和 syuancao.com 其实都是方方的网站
- 方方就是想要两个网站互相访问，浏览器为什么阻止
- **好吧，用 CORS**

- 浏览器说，如果要共享数据，需要提前声明！
- 哦，那怎么声明呢？
- 浏览器说，qq.com 在响应头里写 syuancao.com 可以访问
- 哟，具体语法呢？
- Access-Control-Allow-Origin:http://foo.example
- 浏览器说：都在 MDN 文档里
- 注意：CORS 分为简单请求和复杂请求，具体看文档

## 12. 解法二：JSONP（可兼容 IE）

- 定义
- JSONP 和 JSON 半毛线关系都没有
- 由于前端水平低下，错误地将其称为 JSONP
- 具体看代码
- 我们现在面临的问题是什么？
- 程序员常常面临奇葩需求
- 没有 CORS，怎么跨域
- 记不得我们可以随意引用 JS
- 虽然我们不能访问 qq.com:8888/friends.json
- 但是我们能用 qq.com:8888/friends.js 啊！
- 这有什么用？JS 又不是数据
- 我们让 JS 包含数据不就好了...
- 试试看吧！明天就要上线啦！

## 13. JSONP 步骤

- syuancao.com 访问 qq.com
- qq.com 将数据写到/friends.js
- syuancao.com 用 script 标签引用/friends.js
- /friend.js 执行，执行什么呢？
- syuancao 事先定义号 window.xxx 函数
- /friend.js 执行 window.xxx(friends:[...])
- 然后 syuancao.com 就是通过 window.xxx 获取到数据了
- window.xxx 就是一个回调啊！
- 这 TM 都能想到，人才啊！
- 这是很多前端工程师一起想出来的

## 14. 优化

- xxx 能不写死吗？
- window.xxx 能不能改其他名字？
- 其实名字不重要，只要 syuancao.com 定义的函数名
- 和 qq.com/friends.js 执行的函数名是同一个即可！

- 那就把名字传给 /friends.js

- 具体看代码

## 15. 再优化

- **封装！**

- 初级程序员学 API，中级程序员学封装

- 封装成 `jsonp('url').then(f1, f2)`

## 16. JSONP 是什么？(面试回答，需要回答优缺点)

- JSONP 就是我们在跨域的时候由于当前浏览器不支持 CORS

- 或者因为某些条件不支持 CORS，我们必须使用另外一种方式

- 来跨域，于是我们就请求一个 JS 文件，这个 JS 文件会执行一个回调，

- 回调里面有我们的数据。回调的名字是可以随机生成的，给个随机数

- 然后我们把这个名字以 callback 的名字传给后台，

- 后台会把这个函数返回给我们并执行。

- JSONP 优点是兼容 IE，也可以成功跨域。

- JSONP 缺点是因为它是 script 标签，读不到 AJAX 那么精确的状态，不知道

- 状态码和成功的头是什么，只知道响应和失败。

- 由于它是 script 标签，所以它只能发 get 请求，即 JSONP 不支持 post

## 17. **代码示例**

### 10.4 静态服务器

### 10.5 Ajax 实战：Cookie, Session

## 11 MVC

## 12 Webpack

## 13 虚拟 DOM 与 DOM diff

## 14 Vue

## 15 React

## 16 NodeJS

## 17 Vue3 造轮子