**面试题总结**

**Css部分**

**1 display:none 和 visibility:hidden的区别**

都是元素隐藏，在文档流中，display:none不占位置。visibility:hidden占位置。

性能上，display比visibility差。display控制显示隐藏时，页面会产生回流（当页面中的一部分元素需要改变规模尺寸、布局、显示隐藏等，页面重新构建，此时就是回流。所有页面第一次加载时需要产生一次回流）

**2 CSS和JS的位置会影响页面的效率，为什么？**

css加载过程中不会影响到DOM树的生成，但是会影响到Render树的生成，进而影响到layout

所以一般来说，style的link标签尽量放在head里面，因为在解析DOM树的时候是自上而下的，而css样式又是通过异步加载的，这样的话，解析DOM树的body节点和加载CSS样式尽可能的并行，加快Render树生成的速度。js脚本应该放在底部，原因在于js线程和GUI渲染是互斥的关系，如果js放在首部。当下载js的时候，会影响渲染线程绘制页面，js的作用主要是处理交互，而交互必须得先让页面呈现才能进行，所以为了保证用户体验，尽量让页面先绘制出来。

**3 rem的理解**

相对长度单位，相对于根元素(即html元素)font-size计算值的倍数。简单来说，就是一个相对单位，相对根元素子体大小的倍数。

例如：html的font-size:12px；那么使用rem作为单位的，1rem等价于12px，2rem等价于24px;

按定高宽设计出来页面，然后转换为rem单位，配合js查询屏幕大小来改变html的font-size，完美自适应大屏小屏

**4 盒模型**

html文档中的每个元素都被描绘成矩形盒子，这些矩形盒子通过一个模型来描述其占用空间，这个模型称为盒模型。盒模型通过四个边界来描述：margin（外边距），border（边框），padding（内边距），content（内容区域）

padding，border，margin都是可选的，默认值为0，但是浏览器会自行设置元素的margin和padding

**5 flex (弹性盒)**

flex（ flexible box：弹性布局盒模型）,是2009年w3c提出的一种可以简洁、快速弹性布局的属性。主要思想是给予容器控制内部元素高度和宽度的能力

其中在webkit内核的浏览器中使用时，必须加上**-webkit-**前缀

**（1） flex-direction 属性决定主轴的方向（即项目的排列方向）**

row（默认值）：主轴为水平方向，起点在左端。

row-reverse：主轴为水平方向，起点在右端。

column：主轴为垂直方向，起点在上沿。

column-reverse：主轴为垂直方向，起点在下沿

**（2） flex-wrap 默认情况下，项目都排在一条线（又称”轴线”）上。flex-wrap属性定义，如果一条轴线排不下，如何换行。**

nowrap（默认值）：不换行。

wrap：换行，第一行在上方。

wrap-reverse：换行，第一行在下方。

**（3） flex-flow属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。**

**（4） justify-content属性定义了项目在主轴上的对齐方式。**

flex-start（默认值）：左对齐

flex-end：右对齐

center： 居中

space-between：两端对齐，项目之间的间隔都相等。

space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

**（5）align-items属性定义项目在交叉轴上如何对齐。**

**（6） align-content属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用**

**6 css中的常见单位**

**px**

CSS中最为常用的一种单位，传统上一个像素对应计算机屏幕上的一个点。

**百分比%**

子元素相对父容器的占比

设置子元素的font-size为50%，那么子元素的font-size等于父容器的font-size\*50%。

**Em**

em是一个相对单位 1em等于其父元素的字体大小 如果该元素自己设置了大小则不继承父元素大小 父容器没有设置font-size，子盒子它的font-size为16px\*2=32px

**优势：1 用户可以设置字体大小 2 修改方便快捷 直接改父容器的值 px就要一个一个改不足：嵌套很深的结构 不容易立刻得到值**

**Rem**

一级元素上1个rem都等于跟元素html的font-size

**vw和vh**

1vw = 视口宽度的1% 1vw = 视口高度的1%

**视口**的宽度和高度不是指屏幕的高宽，更不是浏览器的高宽，而是浏览器内部的可视区域大小，即window.innerWidth/window.innerHeight

**7 css选择器**

CSS的选择器分为两大类：基本选择题和扩展选择器。

标签选择器：针对一类标签-------------- ‘p’‘span’

ID选择器：针对某一个特定的标签使用 ------------- ‘#’

类选择器：针对你想要的所有标签使用 ---------------‘ . ’

通用选择器（通配符）：针对所有的标签都适用（不建议使用） -------- ‘ \* ’ **标签越多，效率越低**

**8 清除浮动的方法（**什么是浮动？为什么要清除？浮动就是因为有元素使用了float属性，使得元素处于半脱离文档流的状态，无法撑开父元素。**）**

1. 给父元素设置overflow:hidden
2. 给父元素设置 float:left 就是父元素跟着一起浮动 会产生新的浮动问题
3. 设置父元素的高
4. 父级div定义 伪类:after 和 zoom （zoom是为了适配IE低版本）

.clearfloat:after {

display:block;

clear:both;

content:"";

display:hidden;

height:0 }

页面父盒子class 上加入 .clear float

（5）**父级div定义 overflow: auto （**内部宽高超过父级div时，会出现滚动条**）**

**9 BFC**

BFC 即 Block Formatting Contexts (块级格式化上下文)，具有 BFC 特性的元素可以看作是隔离了的独立容器，容器里面的元素不会在布局上影响到外面的元素，并且 BFC 具有普通容器所没有的一些特性。通俗一点来讲，可以把 BFC 理解为一个封闭的大子，箱子内部的元素无论如何翻江倒海，都不会影响到外部。就像js的函数作用域一样，BFC是页面上的作用域。

**触发BFC**

只要元素满足下面任一条件即可触发 BFC 特性：

body 根元素

浮动元素：float 除 none 以外的值

绝对定位元素：position (absolute、fixed)

display 为 inline-block、table-cells、flex

overflow 除了 visible 以外的值 (hidden、auto、scroll)

**BFC特性**

同一个 BFC 下外边距会发生折叠

BFC 可以包含浮动的元素（清除浮动）

BFC 可以阻止元素被浮动元素覆盖

**10 层叠上下文**

层叠上下文(stacking context)，是HTML中一个三维的概念。在CSS2.1规范中，每个盒模型的位置是三维的，分别是平面画布上的X轴，Y轴以及表示层叠的Z轴，一般情况下，元素在页面上沿X轴Y轴平铺，我们察觉不到它们在Z轴上的层叠关系

最终表现就是它离屏幕观察者更近

**如何产生“层叠上下文”**

1 ----- HTML中的根元素<html></html>本身j就具有层叠上下文，称为“根层叠上下文”。

2 ----- 普通元素设置position属性为非static值并设置z-index属性为具体数值，产生层叠上下文。

3 ----- CSS3中的新属性也可以产生层叠上下文。

**11 常见的css布局**

**网站页面布局**

1 T形布局 这个是我们比较常见的布局，其页面的顶部一般放置横网站的标志或Banner广告，下方左侧是导航栏菜单，下方右侧则用于放置网页正文等主要内容

2. 三栏型布局 国字型布局下最上面是网站的标题以及横幅广告条，接下来是网站的主要内寄，左右分列一些小条内容，中问是主要部分，与左右一起罗列到底，最下方是网站的一些基本信息、联系方式、版权声明等

3. POP布局 POP布局指页面布局像一张宣传海报，以一张精美图片作为页面的设计中心。常用于时尚类站点。优点显而易见：漂亮吸引人。缺点就是速度慢。人大的主页就类似这种布局。

4.左右布局型 顾名思义，就是网页主体分为左右两大块，多见为后台管理系统页面。一般左右布局型的页面需要做到两列等高。

5. 上下布局型 参见苹果的官网，类似于整屏显示的网页都为上下布局。

**布局方法**

1 圣杯布局 2 双飞翼布局 3 flex布局

**响应式布局**

**使用 @media媒体查询，你可以针对不同的媒体类型定义不同的样式。**

@media only screen and (max-width: 1200px)

@media only screen and (max-width: 980px)

以上两种宽度分别设置不同的样式布局

Screen -🡪 用于电脑屏幕，平板电脑，智能手机等。

All --🡪 用于所有设备

max-width ---🡪 定义输出设备中的页面最大可见区域宽度。

**弹性布局（rem/em布局）**

**rem,em区别：**rem,em都是顺应不同网页字体大小展现而产生的。其中，em是相对其父元素，在实际应用中相对而言会带来很多不便；而rem是始终相对于html大小，即页面根元素。

**静态布局**

即传统Web设计，网页上的所有元素的尺寸一律使用px作为单位。

**流式布局（Liquid Layout）**

流式布局（Liquid）的特点（也叫"Fluid") 是页面元素的宽度按照屏幕分辨率进行适配调整，但整体布局不变。代表作栅栏系统（网格系统）。

**响应式和弹性布局之间的对比：**

响应式布局：改变浏览器宽度，“布局”会随之变化，不是一成不变的，例如导航栏在大屏幕下是横排，在小屏幕下是竖排，在超小屏幕下隐藏为菜单，也就是说如果有足够的耐心，在每一种屏幕下都应该有合理的布局，完美的效果。

rem布局：改变浏览器宽度，页面所有元素的高宽都等比例缩放，也就是大屏幕下导航是横的，小屏幕下还是横的只不过变小了。

**12 Css预处理器、后处理器**

**css预处理器**：用一种专门的编程语言，为CSS增加了一些编程的特性，将CSS作为目标生成文件，然后开发者就只要使用这种语言进行编码工作。

css存在的问题：语法不够强大，比如无法嵌套书写导致模块化开发中需要书写很多重复的选择器；

没有变量和合理的样式复用机制，使得逻辑上相关的属性值必须以字面量的形式重复输出，导致难以维护。

所以需要预处理器提供 CSS 缺失的样式层复用机制、减少冗余代码，提高样式代码的可维护性。

目前比较主流的三个处理器分别是 Less、Sass、Stylus

**css后处理器**是对css进行处理，并最终生成css预处理器，它属于广义上的css预处理器

举例：css压缩工具（clean-css）,Autoprefixer(以Can I Use上的浏览器支持数据为基础，自动处理兼容问题)

**13 animation和transiton的相关属性animate和translate和transform**

**Transform（变形）**

[Transform](http://www.w3.org/TR/css3-2d-transforms/" \t "_blank)字面上就是变形，改变的意思。在[CSS3](http://www.w3.org/TR/css3-roadmap/" \t "_blank)中[transform](http://www.w3.org/TR/css3-2d-transforms/" \t "_blank)主要包括以下几种：**旋转rotate**、**扭曲skew**、**缩放scale**和**移动translate**以及**矩阵变形matrix**。

**一、旋转rotate（肉体特）**

transform:rotate(30deg) ---------- 旋转30度-- deg是单位 度

**二、移动translate**

transform:translate(100px,20px): -------------- > x轴向右平移100 y轴向上平移20

transform:translateX(100px): -------------- > x轴向右平移100

transform:translateY(20px): -------------- > y轴向上平移20

**三、缩放scale（司改u）**

transform:scale(2,1.5): -------------- > x轴缩放2倍 y轴缩放1.5倍

transform:scaleX(2): -------------- > x轴缩放2倍

transform:scaleY(2): -------------- > y轴缩放1.5倍

**四、扭曲skew（思qu）**

transform:skew(30deg,10deg): -------------- > x轴扭曲30度y轴扭曲10度

transform:skewX(30deg) -------------- > x轴扭曲30度

transform:skewY（10deg）-------------- > y轴扭曲10度

**transition（过渡）**

transition主要包含四个属性值：执行变换的属性：transition-property,变换延续的时间：transition-duration,在延续时间段，变换的速率变化transition-timing-function,变换延迟时间transition-delay。

transition: all 1s ease .1s;

**transition-property**是用来指定当元素其中一个属性改变时执行transition效果，其主要有以下几个值：none(没有属性改变)；all（所有属性改变）这个也是其默认值；indent（元素属性名）。

**transition-timing-function**的值允许你根据时间的推进去改变属性值的变换速率，transition-timing-function有6个可能值：

1、ease：（逐渐变慢）默认值，ease函数等同于贝塞尔曲线(0.25, 0.1, 0.25, 1.0).

2、linear：（匀速），linear 函数等同于贝塞尔曲线(0.0, 0.0, 1.0, 1.0).

3、ease-in：(加速)，ease-in 函数等同于贝塞尔曲线(0.42, 0, 1.0, 1.0).

4、ease-out：（减速），ease-out 函数等同于贝塞尔曲线(0, 0, 0.58, 1.0).

5、ease-in-out：（加速然后减速），ease-in-out 函数等同于贝塞尔曲线(0.42, 0, 0.58, 1.0)

6、cubic-bezier：（该值允许你去自定义一个时间曲线）， 特定的[cubic-bezier曲线](http://en.wikipedia.org/wiki/B%C3%A9zier_curve" \t "_blank)。 (x1, y1, x2, y2)四个值特定于曲线上点P1和点P2。所有值需在[0, 1]区域内，否则无效。

**Animation（动画）**

**Keyframes**

命名是由"@keyframes"开头，后面紧接着是这个“动画的名称”加上一对花括号“{}”，fromt、to来代表一个动画是从哪开始，到哪结束，也就是说这个 "from"就相当于"0%"而"to"相当于"100%" （只接受百分比 不接受0）

**animation和transition区别**

animation类似于transition属性，他们都是随着时间改变元素的属性值。他们主要区别是transition需要触发一个事件(hover事件或click事件等)才会随时间改变其css属性；而animation在不需要触发任何事件的情况下也可以显式的随着时间变化来改变元素css的属性值，从而达到一种动画的效果

**14 display哪些取值**

None -----------🡪此元素不会被显示。

block-----------🡪 此元素将显示为块级元素，此元素前后会带有换行符。

inline-----------🡪此元素会被显示为内联元素，元素前后没有换行符。

inline-block-----------🡪行内块元素。

list-item-----------🡪此元素会作为列表显示。

run-in-----------🡪此元素会根据上下文作为块级元素或内联元素显示。

table-----------🡪此元素会作为块级表格来显示（类似 <table>），表格前后带有换行符。

inherit-----------🡪 规定应该从父元素继承 display 属性的值。

table-cell-----------🡪 此元素会作为一个表格单元格显示（类似 <td> 和 <th>）

**15 相邻的两个inline-block节点为什么会出现间隔，该如何解决**

**产生原因**

西文排版问题-------🡪它由空格、换行或回车所产生空白符所致

**解决方案**

方法1:改变书写方式（将行内块元素写在代码的同一行）

方法2：font-size（给父元素设置font-size = 0）

方法2：设置margin负值（不推荐使用）margin-right: -5px;

**16 移动端适配**

**通过媒体查询的方式即CSS3的meida queries**

它主要是通过查询设备的宽度来执行不同的 css 代码，最终达到界面的配置。

**优势** 方法简单，成本低，特别是对移动和PC维护同一套代码的时候。目前像Bootstrap等框架使用这种方式布局

**缺点** 代码量比较大，维护不方便

**以天猫首页为代表的 flex 弹性布局**

它的viewport是固定的：<meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1,user-scalable=no">

**扩展**

initial-scale ------------> 表示初始的缩放比例

maximum-scale -----------🡪最大缩放比例

*minimum-scale=0.5------------🡪*最小缩放比例

*user-scalable=yes---------->*用户是否可以自己调整缩放比例

**特点** 高度定死，宽度自适应，元素都采用px做单位。

Scale 为当前缩放比例 device-width **定义输出设备的屏幕可见宽度。**

device-width的计算公式为：设备的物理分辨率/(devicePixelRatio \* scale)，

**devicePixelRatio**

UI设计师要求的1px是指设备的物理像素1px，而CSS里记录的像素是逻辑像素，它们之间存在一个比例关系，Window 属性 devicePixelRatio（简称dpr） 能够返回当前显示设备的物理像素分辨率与 CSS 像素分辨率的比率。此值也可以解释为像素大小的比率：一个 CSS 像素的大小与一个物理像素的大小的比值。简单地说，这告诉浏览器应该使用多少个屏幕的实际像素来绘制单个 CSS 像素。

**以淘宝首页为代表的 rem+viewport缩放**

根据rem将页面放大dpr倍, 然后viewport设置为1/dpr.

如iphone6 plus的dpr为3, 则页面整体放大3倍, 1px(css单位)在plus下默认为3px(物理像素)   
然后viewport设置为1/3, 这样页面整体缩回原始大小. 从而实现高清。

1. **rem 方式**

详见文件

**Viewport**

**layout viewport**(浏览器默认的宽度是大于浏览器可视区域的宽度的)  通过document.documentElement.clientWidth 获取

**visual viewport**（代表 浏览器可视区域的大小）window.innerWidth 获取

**ideal viewport** （移动设备的理想viewport）ideal viewport并没有一个固定的尺寸，不同的设备拥有有不同的ideal viewport

**17 rem em适配的优缺点**

Rem作用于非根元素 相对于根元素

**rem优点：1** 大大减少我们代码的重复性，是我们的代码更兼容 **2** 兼容大部分机型 3 计算方便 拥有统一的参考系（根元素大小）**缺点** 就是在pc端的应用比较少

**Em优点:**在于相对于写死的px,em更加富有张力，改动父元素的字体大小，子元素会等比例变化 **缺点** 在于这样的结构牵一发而动全身

**18  transparen**

 transparent 它代表着全透明黑色，即一个类似rgba(0,0,0,0)这样的值

应用场景：

background:transparent

border: solid transparent;

**19 box-shadow**

box-shadow: h-shadow v-shadow blur spread color inset;

|  |  |
| --- | --- |
| *h-shadow* | 必需。水平阴影的位置。允许负值。 |
| *v-shadow* | 必需。垂直阴影的位置。允许负值。 |
| *blur* | 可选。模糊距离。 |
| *spread* | 可选。阴影的尺寸。 |
| *color* | 可选。阴影的颜色。请参阅 CSS 颜色值。 |
| inset | 可选。将外部阴影 (outset) 改为内部阴影。 |

**20 overflow**

|  |  |
| --- | --- |
| visible | 默认值。内容不会被修剪，会呈现在元素框之外。 |
| hidden | 内容会被修剪，并且其余内容是不可见的。 |
| scroll | 内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容。 |
| auto | 如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容。 |
| inherit | 规定应该从父元素继承 overflow 属性的值。 |

**21 图片格式**

**JPEG（jpg）**

图片压缩，尤其是大图，就用它，其他都比不上它。

**PNG**

屏幕截图，图片处理，个人认为是目前为止，除了照片大图不如JPEG，动画不如GIF之外，所有图片中最好的格式就PNG了。

**GIF**

动图就用它吧，上面那俩都不支持，起码到目前为止不支持。

**22 position**

Position relative 相对定位 相对于其原来的位置

Position absolute 绝对定位 相对于其父盒子进行定位

Position fixed 绝对定位 相对浏览器窗口进行定位

Position static 默认值 没有定位

Position inherit 继承父盒子的定位属性值

**HTML部分**

**1 标签语义化（合理的标签做合理的事情）**

常用的语义化标签包括 **见2**

**标签语义化的意义**

1、去掉或丢失样式的时候，能够让界面呈现清晰的结构

2、有利于SEO:和搜索引擎建立更好的沟通，有助于爬虫抓取更多的有效信息，爬虫依赖标签确定上下文和各种关键字的权重。

3、方便其他设备解析（如屏幕阅读器，盲人阅读器，移动设备）以意义的方式渲染网页

4、便于团队开发和维护，语义化使得网页更具可读性，是进一步开发网页的步骤，遵循W3c标准的团队，都遵循这个标准，可以减少差异化。

**2 新标签总结**

**块标签**

<head>定义页面或者区域头部 <main>规定文档主要内容

<footer>定义页面或者区域底部 <article>定义一篇文章

<nav>定义导航链接 <scetion>定义一个区域

<aside>定义页面内容部分侧边栏 <hgroup>用于对网页或者区域段的标题组合

<figure>定义一组媒体内容以及标题 <figcaption>定义figure元素标题

<address>定义文章拥有者的联系信息

**行标签**

<time>定义时间、日期 <mark>高亮显示文字

**多媒体交互标签**

<video>定义一个视频 <audio>定义一个音频

<source>定义媒体资源标签 <canvas>定义图形 <embed>定义可交互的内容或者插件

**web应用标签**

<progress>状态标签、进度条 <mark>定义有标记的文本（默认是黄色选中内容）

**3 [<input type="text">和<textarea>的区别](https://www.cnblogs.com/yonglin/p/7041423.html)**

text标签是单行文本框，不会换行

<textarea>是多行文本输入框，文本区中可容纳无限数量的文本

**4 div 实现 textarea**

contenteditable="true" ---------- > 改变当前内容为可以编辑

outline: none; -----------🡪 定义当前元素轮廓为none 改变为可编辑之后编辑时会有蓝色轮廓

**5 移动端避免将数字识别为电话**

<meta name = "format-detection" content = "telephone=no">

**Meta标签中的format-detection属性及含义**

**format-detection属性：**翻译成中文的意思是“格式检测”，顾名思义，它是用来检测html里的一些格式的

**http-equiv属性**：<meta http-equiv='参数' content='参数变量值'>

1.<meta http-equiv=”Set-Cookie” content=”cookievalue=xxx; expires=Friday,12-Jan-2001 18:18:18 GMT; path=/”>:如果网页过期，那么存盘的cookie将被删除。必须使用GMT的时间格式。

2.**<meta http-equiv='expires' content='时间' >**：用于设定网页的到期时间。一旦网页过期，必须到服务器上重新传输。

3.**<meta http-equiv=”Refresh” content=”5;URL”>**：告诉浏览器在【数字】秒后跳转到【一个网址】

4.<meta http-equiv=”content-Type” content=”text/html; charset=utf-8″>：设定页面使用的字符集。

<meta charset=”utf-8″>：在HTML5中设定字符集的简写写法。

5.<meta http-equiv=”Pragma” content=”no-cache”>：禁止浏览器从本地计算机的缓存中访问页面内容。访问者将无法脱机浏览。

6**.<meta http-equiv=”Window-target” content=”\_top”>**：用来防止别人在iframe(框架)里调用自己的页面，这也算是一个非常实用的属性。

7**.<metahttp-equiv='X-UA-Compatible'content='IE=edge,chrome=1'>** :强制浏览器按照特定的版本标准进行渲染。但不支持IE7及以下版本。如果是ie浏览器就用最新的ie渲染，如果是双核浏览器就用chrome内核。

**name属性**： <meta name='参数' content='具体的参数值'>

1.<meta name=”viewport” content=”width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no”>：在移动设备浏览器上，禁用缩放（zooming）功能，用户只能滚动屏幕。

2.<meta name=”description” content=””>：告诉搜索引擎，当前页面的主要内容是xxx。

3.<meta name=”keywords” content=””>：告诉搜索引擎，当前页面的关键字。

4.<meta name=”author” content=””>：告诉搜索引擎，标注网站作者是谁。

5.<meta name=”copyright” content=””>：标注网站的版权信息。

<meta> 元素可提供有关页面的元信息（meta-information），比如针对搜索引擎和更新频度的描述和关键词。

<meta> 标签位于文档的头部，不包含任何内容。<meta> 标签的属性定义了与文档相关联的名称/值对。

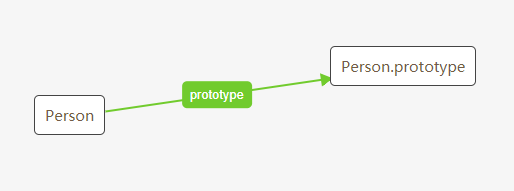
**JS部分**

**1 原型 原型链 构造函数 实例 继承**

**原型**

原型指的就是一个对象，实例“继承”那个对象的属性。在原型上定义的属性，通过“继承”，实例也拥有了这个属性。“继承”这个行为是在 new 操作符内部实现的

**原型与构造函数的关系 （构造函数通过prototype这个属性能够访问到原型）**

****

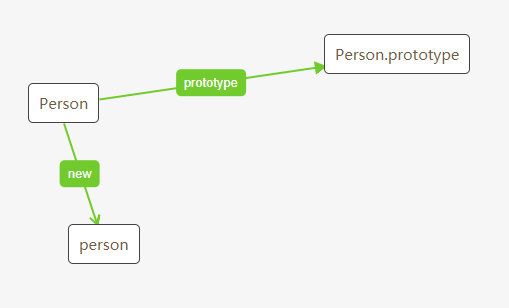
**instance 实例**

有个构造函数，我们就可以在原型上创建可以“继承”的属性，并

通过 new 操作符创建实例

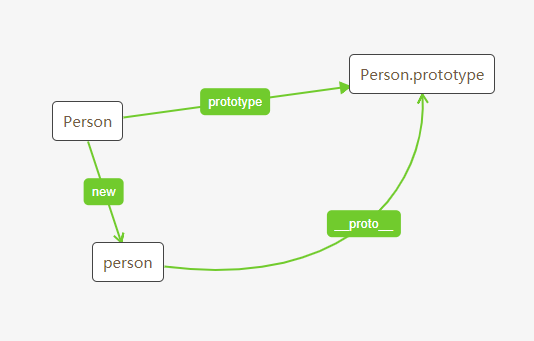
**instanceof** 我们可以通过这个属性验证当前实例是否属于这个函数 xx instanceof yy

我们在构造函数的原型上定义一个属性，在当前实例中也能访问到

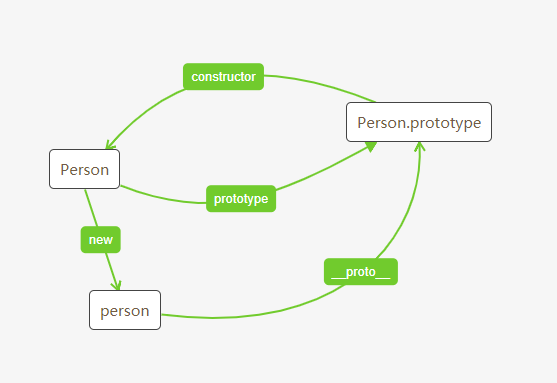
****

**proto 隐式原型**

当前实例通过 \_\_proto\_\_ 访问构造函数的原型

****

**constructor 构造函数**

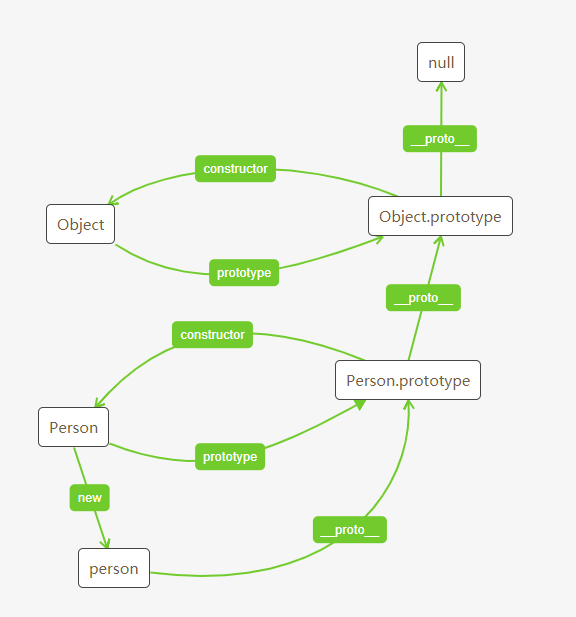
既然构造函数通过 prototype 来访问到原型，那么原型也应该能够通过某种途径访问到构造函数，这就是 constructor

**原型链**

原型同样也可以通过 \_\_proto\_\_ 访问到原型的原型，比方说这里有个构造函数 Person 然后“继承”前者的有一个构造函数 People，然后 new People 得到实例 p

当访问 p 中的一个非自有属性的时候，就会通过 \_\_proto\_\_ 作为桥梁连接起来的一系列原型、原型的原型、原型的原型的原型直到 Object 构造函数为止。

这个搜索的过程形成的链状关系就是原型链



**实现继承的几种方式**

**继承的概念**

**传统面向对象语言：继承是类与类之间的关系。**

**而在js中由于es6之前没有类的概念，所以继承是对象与对象之间的关系。**

**在js中，继承就是指使一个对象有权去访问另一个对象的能力。**

**比如：比如对象a能够访问对象b的成员（属性和方法），那么就说对象a继承于对象b；**

**1 原型链继承**

**基本思想**：

利用原型链让一个引用类型继承另一个引用类型的属性和方法。

**缺点：**

**1. 来自原型对象的引用属性是所有实例共享的。**

**2. 创建子类实例时，无法向父类构造函数传参。**

### 2. 借用构造函数

**基本思想：**

**在子类构造函数的内部通过call()以及apply()调用父类构造函数。**

**缺点：**

**1. 如果方法都在构造函数中定义，那么就无法复用函数。每次构建实例时都会在实例中保留方法函数，造成了内存的浪费，同时也无法实现同步更新，因为每个实例都是单独的方法函数。如果方法写在prototype上，就只会有一份，更新时候会做到同步更新。**

### 3. 组合继承

**基本思想：**

**将原型链和借用构造函数的技术组合到一块，从而发挥二者之长的一种继承模式。**

使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承

**缺点：**

**1. 调用了两次父类构造函数，一次通过SuperType.call(this)调用，一次通过new SuperType()调用。**

### 4. 原型式继承

**基本思想：**

**不使用严格意义上的构造函数，借助原型可以基于已有的对象创建新的对象，同时还不必因此创建自定义类型。**

// 在object函数内部，先创建了一个临时的构造函数，然后将传入的对象作为这个构造函数的原型，最后返回这个临时类型的一个新实例。

// 从本质上讲，object()对传入其中的对象执行了一次浅复制。

缺点：

**1. 和原型链继承一样，所有子类实例共享父类的引用类型。**

### 5. 寄生式继承

**基本原理：**

**寄生式继承是与原型式继承紧密相关的一种思路，创建一个仅用于封装继承过程的函数，该函数内部以某种形式来做增强对象，最后返回对象。**

**缺点：**

**1. 和原型链式继承一样，所有子类实例共享父类引用类型。**

**2. 和借用构造函数继承一样，每次创建对象都会创建一次方法。**

### 6. 寄生组合式继承

**基本思想：**

**将寄生式继承和组合继承相结合，解决了组合式继承中会调用两次父类构造函数的缺点。**

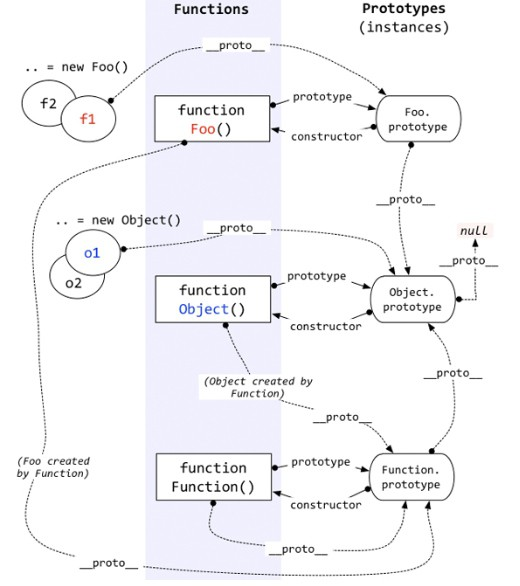
组合继承是JavaScript最常用的继承模式，它最大的问题就是无论在什么情况下，都会调用两次父类构造函数：一次是在创建子类原型的时候，另一次是在子类构造函数内部。

**开发人员普遍认为寄生组合式继承是引用类型最理想的继承方式。**

### 7 ES6实现继承

Class 创建类 extends继承类

**原理图**

****

### 2 isPrototypeOf、instanceof、hasOwnProperty

### isPrototypeOf

作用：检测一个对象是否是另一个对象的原型。或者说一个对象是否被包含在另一个对象的原型链中

### instanceof

instanceof运算符希望左操作数是一个对象，右操作数标识对象的类。如果左侧对象是右侧类的实例，则表达式返回为true，否则返回false。

### hasOwnProperty

检测集合成员的所属关系，判断某个属性是否存在于某个对象中。可以通过in运算符，hasOwnProperty()来完成。

**3 闭包**

**变量的作用域**

全局作用域 ---- > window

函数私有作用域 ---🡪function

私有作用域可以读取全局作用域的变量 而在私有作用域外面一般比较不容易获取私有作用域中的属性

**读取私有属性 在私有作用域中声明一个函数 子函数可以读取父函数中的属性**

**闭包的概念**

函数A包含函数B，函数B中调用函数A中的变量，函数B是闭包

**闭包的作用**

1 访问上级函数中的变量

2 保存上级函数中被调用的变量，使之不会被 垃圾回收机制 删除

**闭包的弊端**

由于闭包会导致被调用的函数变量一直保存，对内存的消耗比较大，可能会造成内存泄漏，解决方法就是在退出函数之前，将不使用的局部变量全部删除

**闭包的形成条件**

1 自执行函数2 匿名函数

**4 对象的深拷贝与浅拷贝**

浅拷贝：只拷贝对象的基础属性值，对属性值为对象或数组的属性则拷贝指针。   
深拷贝：拷贝对象的所有属性作为一个全新的对象。拷贝前后的对象互不影响。

**浅拷贝仅仅是指向被复制的内存地址，如果原地址中对象被改变了，那么深拷贝出来的对象也会相应改变。**

**5 call和apply bind**

apply：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.apply(A, arguments);即A对象应用B对象的方法。

call：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.call(A, args1,args2);即A对象调用B对象的方法

从定义中可以看出，call和apply都是调用一个对象的一个方法，用另一个对象替换当前对象。而不同之处在于传递的参数，**apply最多只能有两个参数——新this对象和一个数组argArray，如果arg不是数组则会报错TypeError；**

**call则可以传递多个参数，第一个参数和apply一样，是用来替换的对象，后边是参数列表**

**bind()方法调用并改变函数运行时的上下文的之后，返回一个新的函数，在我们需要调用的地方去调用他。**

#### 对比

call，apply，bind 三者用法和区别：参数、绑定规则（显示绑定和强绑定）、运行效率（最终都会转换成一个一个的参数去运行）、运行情况（call，apply 立即执行，bind 是return 出一个 this “固定”的函数，这也是为什么 bind 是强绑定的一个原因）。

注：“固定”这个词的含义，它指的固定是指只要传进去了 context，则 bind 中 return 出来的函数 this 便一直指向 context，除非 context 是个变量。

**6 this 对象**

1. 函数运行过程中如果没有指定this那么当前this指向的就是window对象

2.全局作用域——>指向window

3.call/apply会改变函数运行的指向

4.obj.function(); function()里面的this指向的就是obj.谁调用了这个方法，this就指向谁

每个对象都有 \_\_proto\_\_ 属性，但只有函数对象才有 prototype 属性

**6 同源策略**

**概念**：1995年，同源政策由 Netscape 公司引入浏览器。目前，所有浏览器都实行这个政策。

最初，它的含义是指，A网页设置的 Cookie，B网页不能打开，除非这两个网页"同源"。所谓"同源"指的是"三个相同"。 **协议相同 --- 端口相同 --- 域名相同**

**目的**：同源政策的目的，是为了保证用户信息的安全，防止恶意的网站窃取数据。

设想这样一种情况：A网站是一家银行，用户登录以后，又去浏览其他网站。如果其他网站可以读取A网站的 Cookie，会发生什么？

很显然，如果 Cookie 包含隐私（比如存款总额），这些信息就会泄漏。更可怕的是，Cookie 往往用来保存用户的登录状态，如果用户没有退出登录，其他网站就可以冒充用户，为所欲为。因为浏览器同时还规定，提交表单不受同源政策的限制。

由此可见，"同源政策"是必需的，否则 Cookie 可以共享，互联网就毫无安全可言了

**限制范围**：随着互联网的发展，"同源政策"越来越严格。目前，如果非同源，共有三种行为受到限制。

（1） Cookie、LocalStorage 和 IndexDB 无法读取。

（2） DOM 无法获得。

（3） AJAX 请求不能发送。

**解决方法**

1、 cookie

Cookie 是服务器写入浏览器的一小段信息，只有同源的网页才能共享。但是，两个网页一级域名相同，只是二级域名不同，浏览器允许通过设置document.domain共享 Cookie。

举例来说，A网页是http://w1.example.com/a.html，B网页是http://w2.example.com/b.html，那么只要设置相同的document.domain，两个网页就可以共享Cookie。

2、window.name

浏览器窗口有window.name属性。这个属性的最大特点是，无论是否同源，只要在同一个窗口里，前一个网页设置了这个属性，后一个网页可以读取它。

父窗口先打开一个子窗口，载入一个不同源的网页，该网页将信息写入window.name属性。

这种方法的优点是，window.name容量很大，可以放置非常长的字符串；缺点是必须监听子窗口window.name属性的变化，影响网页性能.

3、window.postMessage

HTML5为了解决这个问题，引入了一个全新的API：跨文档通信 API（Cross-document messaging）。

这个API为window对象新增了一个window.postMessage方法，允许跨窗口通信，不论这两个窗口是否同源。

举例来说，父窗口http://aaa.com向子窗口http://bbb.com发消息，调用postMessage方法就可以了。

postMessage方法的第一个参数是具体的信息内容，第二个参数是接收消息的窗口的源（origin），即"协议 + 域名 + 端口"。也可以设为\*，表示不限制域名，向所有窗口发送。

子窗口向父窗口发送消息的写法类似。

window.opener.postMessage('Nice to see you', 'http://aaa.com');

4、jsonp (只能发送get请求)

JSONP是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，老式浏览器全部支持，服务器改造非常小。

它的基本思想是，网页通过添加一个<script>元素，向服务器请求JSON数据，这种做法不受同源政策限制；服务器收到请求后，将数据放在一个指定名字的回调函数里传回来。

首先，网页动态插入<script>元素，由它向跨源网址发出请求。

5、websocket

WebSocket是一种通信协议，使用ws://（非加密）和wss://（加密）作为协议前缀。该协议不实行同源政策，只要服务器支持，就可以通过它进行跨源通信。

6、CORS

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。

它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制

**简介：**

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

整个CORS通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS通信与同源的AJAX通信没有差别，代码完全一样。浏览器一旦发现AJAX请求跨源，就会自动添加一些附加的头信息，有时还会多出一次附加的请求，但用户不会有感觉。

因此，实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信

**7 事件模型**

事件：浏览器、window、document、HTML元素发生事情，浏览器封装事情信息，就是事件。

事件源：引发事件的对象，就是事件源。

事件类型：每个事件都有相应的类型，比如："click"、"load"等，是一个说明事件类型的字符串。

事件处理器：JS中处理器和监听器是一回事。绑定到对象上，用来对事件进行响应。一般为一个函数。

**事件代理**

事件委托即事件代理，就是把多个节点的事件根据事件冒泡的原理绑定到其父节点上，以减少不必要的性能损耗。

window.event.srcElement -------🡪 获取当前触发事件的对象 = 获取事件源

**8** **window的onload事件和domcontentloaded**

1、当 onload 事件触发时，页面上所有的DOM，样式表，脚本，图片，flash都已经加载完成了。

2、当 DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash。

**9 for...in迭代和for...of有什么区别**

区别1 for…of 是es6 中引入的新的循环方式

区别2 for…in 循环的是 key for…of 循环的是 value

区别3 for…in 多用于循环对象 for..of 多用于循环数组

**10 函数柯里化**

**概念**

实际上就是把add函数的x，y两个参数变成了先用一个函数接收x然后返回一个函数去处理y参数。现在思路应该就比较清晰了，就是只传递给函数一部分参数来调用它，让它返回一个函数去处理剩下的参数。

**实例**

function curryingAdd(x) {

return function (y) {

return x + y

}

}

好处

##### 1. 参数复用

##### 2. 提前确认

##### 3. 延迟运行

**11 async/await**

**异步**

所谓"异步"，简单说就是一个任务不是连续完成的，可以理解成该任务被人为分成两段，先执行第一段，然后转而执行其他任务，等做好了准备，再回过头执行第二段。

async函数就是将 Generator 函数的星号（\*）替换成async，将yield替换成await，仅此而已。

async函数对 Generator 函数的改进，体现在以下四点。

（1）内置执行器。

Generator 函数的执行必须靠执行器，所以才有了co模块，而async函数自带执行器。也就是说，async函数的执行，与普通函数一模一样，只要一行。

（2）更好的语义。

async和await，比起星号和yield，语义更清楚了。async表示函数里有异步操作，await表示紧跟在后面的表达式需要等待结果。

（3）更广的适用性。

co模块约定，yield命令后面只能是 Thunk 函数或 Promise 对象，而async函数的await命令后面，可以是 Promise 对象和原始类型的值（数值、字符串和布尔值，但这时会自动转成立即 resolved 的 Promise 对象）。

（4）返回值是 Promise。

async函数的返回值是 Promise 对象，这比 Generator 函数的返回值是 Iterator 对象方便多了。你可以用then方法指定下一步的操作。

进一步说，async函数完全可以看作多个异步操作，包装成的一个 Promise 对象，而await命令就是内部then命令的语法糖

**特点**

 async/await更加语义化，async 是“异步”的简写，async function 用于申明一个 function 是异步的； await，可以认为是async wait的简写， 用于等待一个异步方法执行完成；

 async/await是一个用同步思维解决异步问题的方案（等结果出来之后，代码才会继续往下执行）

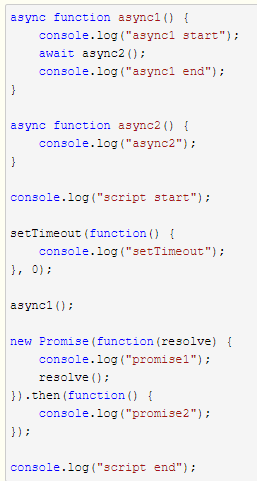
 可以通过多层 async function 的同步写法代替传统的callback嵌套

**async function语法**

* 自动将常规函数转换成Promise，返回值也是一个Promise对象
* 只有async函数内部的异步操作执行完，才会执行then方法指定的回调函数
* 异步函数内部可以使用await

**await语法**

* await 放置在Promise调用之前，await 强制后面代码等待，直到Promise对象resolve，如果promise出现报错，后面的代码将不再执行，得到resolve的值作为await表达式的运算结果
* await只能在async函数内部使用,用在普通函数里就会报错



script start async1 start async2 promise1 script end promise2 async1 end setTimeout

**12 立即执行函数（自执行函数）**

立即执行函数特点就是当JavaScript引擎解析到这个语句的时候就会马上执行，执行结束后马上把自己的执行上下文都销毁。这样就可以释放这里的内存，立即执行函数可以有返回值

以及形参等。

**作用**

1. 不必为函数命名，避免了污染全局变量
2. 立即执行函数内部形成了一个单独的作用域，可以封装一些外部无法读取的私有变量
3. 封装变量

**写法**

(function(){

//code

}())

(function (){

//code

})()

**使用场景**

1、你的代码在页面加载完成之后，不得不执行一些设置工作，比如时间处理器，创建对象等等。

2、所有的这些工作只需要执行一次，比如只需要显示一个时间。

3、但是这些代码也需要一些临时的变量，但是初始化过程结束之后，就再也不会被用到，如果将这些变量作为全局变量，不是一个好的注意，我们可以用立即执行函数——去将我们所有的代码包裹在它的局部作用域中，不会让任何变量泄露成全局变量。

**13 设计模式（6种）**1、 构造模式   2 工厂模式 3 模块模式 4单例设计模式 5 混合模式（构造模式+原型模式） 6 订阅-发布模式

1、单例模式

顾名思义，单例就是单一的意思，单例模式的定义是：保证一个类仅有一个一个实例，并提供一个访问它的全局访问点。

单例模式的应用： 全局中只需要一个公用的实例对象，比如登录弹窗组件。

**14 Iframe 优缺点**

**iframe的优点：**

1.iframe能够原封不动的把嵌入的网页展现出来。  
2.如果有多个网页引用iframe，那么你只需要修改iframe的内容，就可以实现调用的每一个页面内容的更改，方便快捷。  
3.网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用iframe来嵌套，可以增加代码的可重用。  
4.如果遇到加载缓慢的第三方内容如图标和广告，这些问题可以由iframe来解决。  
**iframe的缺点：**  
1.会产生很多页面，不容易管理。  
2.iframe框架结构有时会让人感到迷惑，如果框架个数多的话，可能会出现上下、左右滚动条，会分散访问者的注意力，用户体验度差。  
3.代码复杂，无法被一些搜索引擎索引到，这一点很关键，现在的搜索引擎爬虫还不能很好的处理iframe中的内容，所以使用iframe会不利于搜索引擎优化。  
4.很多的移动设备（PDA手机）无法完全显示框架，设备兼容性差。  
5.iframe框架页面会增加服务器的http请求，对于大型网站是不可取的。  
分析了这么多，现在基本上都是用Ajax来代替iframe，所以iframe已经渐渐的退出了前端开发。

**15 数组常用方法**

Array.map()

此方法是将数组中的每个元素调用一个提供的函数，结果作为一个新的数组返回，并没有改变原来的数组

Array.forEach()

此方法是将数组中的每个元素执行传进提供的函数，没有返回值，直接改变原数组，注意和map方法区分

Array.filter()

此方法是将所有元素进行判断，将满足条件的元素作为一个新的数组返回

Array.every()

此方法是将所有元素进行判断返回一个布尔值，如果所有元素都满足判断条件，则返回true，否则为false：

Array.some()

此方法是将所有元素进行判断返回一个布尔值，如果存在元素都满足判断条件，则返回true，若所有元素都不满足判断条件，则返回false：

Array.reduce()

此方法是所有元素调用返回函数，返回值为最后结果,传入的值必须是函数类型：

Array.push()

此方法是在数组的后面添加新加元素，此方法改变了数组的长度：

Array.pop()

此方法在数组后面删除最后一个元素，并返回数组，此方法改变了数组的长度：

Array.shift()

此方法在数组后面删除第一个元素，并返回数组，此方法改变了数组的长度：

Array.unshift()

此方法是将一个或多个元素添加到数组的开头，并返回新数组的长度：

Array.isArray()

判断一个对象是不是数组，返回的是布尔值

Array.concat()

此方法是一个可以将多个数组拼接成一个数组：

Array.toString()

此方法将数组转化为字符串

Array.join()

此方法也是将数组转化为字符串

.Array.splice(开始位置， 删除的个数，元素)

万能方法，可以实现增删改：

indexOf 方法可返回某个指定的字符串值在字符串中首次出现的位置。

lastIndexOf 两个参数 1 检索的字符串 2 开始检索的位置 它的合法取值是 0 到 stringObject.length – 1 默认是最后一个

**数组扁平化**

数组扁平化是指将一个多维数组变为一维数组

1. reduce

遍历数组每一项，若值为数组则递归遍历，否则concat。

2. toString & split

调用数组的toString方法，将数组变为字符串然后再用split分割还原为数组

3. join & split

和上面的toString一样，join也可以将数组转换为字符串

4. 递归

递归的遍历每一项，若为数组则继续遍历，否则concat

5. 扩展运算符

es6的扩展运算符能将二维数组变为一维

**16 BOM属性和方法**

**定义**：BOM(Browser Object Model) 是指浏览器对象模型,是用于描述这种对象与对象之间层次关系的模型

**Window对象**

**方法**：

confirm

console.log(window.confirm('我是小仙女'));

open:

window.open("http://www.baidu.com","baidu")

close

window.close()

**属性**

closed

**Navigator对象**

**属性**

appName（现在一般浏览器除了ie(Internet explore)以外差不多打出来的appName几乎都是"Netscape"）

online：指明系统是否处于脱机状态（就是检查现在是否有网)要是脱机状态也就是没网返回false，有网返回true，

userAgent：返回客户端的完整信息，可以判断机型，浏览器，手机端还是PC端。

**Screen对象**

获取当前屏幕信息 （宽高等等）

**History对象**

History对象属性

* length在同一个标签页，跳转了多少次，length就是多少

History对象方法

* back()和forward() go()这个需要添加参数数字

**Location对象**

Location对象属性

* host：设置或返回主机名和当前 URL 的端口号。
* hostname：设置或返回当前 URL 的主机名
* protocol：设置或返回当前 URL 的协议。（http或者https）
* href：当前页面url

5. hash：设置或返回从井号 (#) 开始的 URL（锚）。其实就相当于锚点

**17 浏览器内核划分**

**Blink**

Blink是一个由Google和Opera Software开发的浏览器排版引擎

**Webkit**

Webkit（Safari内核，Chrome内核原型，开源）**:**它是苹果公司自己的内核，也是苹果的[Safari](https://baike.baidu.com/item/Safari" \t "_blank)浏览器使用的内核 在脚本理解方面，Chrome使用自己研发的V8引擎

**Gecko**

Gecko（Firefox内核）：Netscape6开始采用的内核，后来的[Mozilla FireFox](https://baike.baidu.com/item/Mozilla%20FireFox" \t "_blank)（[火狐浏览器](https://baike.baidu.com/item/%E7%81%AB%E7%8B%90%E6%B5%8F%E8%A7%88%E5%99%A8" \t "_blank)） 也采用了该内核 代码完全公开

**Trident**

IE浏览器

**18 垃圾回收机制**

1）问什么是垃圾

一般来说没有被引用的对象就是垃圾，就是要被清除， 有个例外如果几个对象引用形成一个环，互相引用，但根访问不到它们，这几个对象也是垃圾，也要被清除。

2）如何检垃圾

一种算法是标记 标记-清除 算法

**18 event Loop**

**进程与线程**

两个名词都是 CPU **工作时间片**的一个描述

进程描述了 CPU 在运行指令及加载和保存上下文所需的时间，放在应用上来说就代表了一个程序。线程是进程中的更小单位，描述了执行一段指令所需的时间。

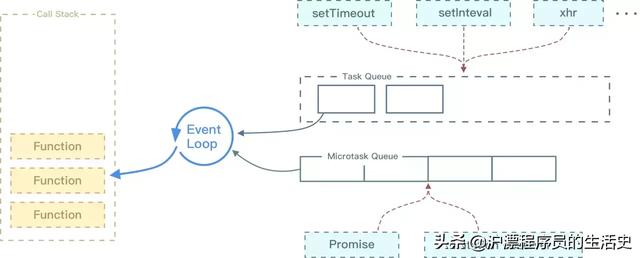
把这些概念拿到浏览器中来说，当你打开一个 Tab 页时，其实就是创建了一个进程，一个进程中可以有多个线程，比如渲染线程、JS 引擎线程、HTTP 请求线程等等。当你发起一个请求时，其实就是创建了一个线程，当请求结束后，该线程可能就会被销毁。

**执行栈**

可以把执行栈认为是一个存储函数调用的**栈结构**，遵循先进后出的原则。

当开始执行 JS 代码时，首先会执行一个 main 函数，然后执行我们的代码。根据先进后出的原则，后执行的函数会先弹出栈，

**浏览器中的 Event Loop（事件循环）**



不同的任务源会被分配到不同的 Task 队列中，任务源可以分为 **微任务**（microtask） 和 **宏任务**（macrotask）。在 ES6 规范中，microtask 称为 jobs，macrotask 称为 task

**Node 中的 Event Loop**

Node 中的 Event Loop 和浏览器中的是完全不相同的东西。

Node 的 Event Loop 分为 6 个阶段，它们会按照顺序反复运行。每当进入某一个阶段的时候，都会从对应的回调队列中取出函数去执行。当队列为空或者执行的回调函数数量到达系统设定的阈值，就会进入下一阶段。

**19 网页从输入网址到渲染完成经历了哪些过程？**

大致可以分为如下7步：

1 输入网址；

2 发送到DNS服务器，并获取域名对应的web服务器对应的ip地址，我们通常所见到的是[www.baidu.com](http://www.baidu.com) DNS叫域名解析协议，就是通过DNS将域名解析成对应的IP地址

3 与web服务器建立TCP连接；

4 浏览器向web服务器发送http请求；

5 web服务器响应请求，并返回指定url的数据（或错误信息，或重定向的新的url地址）；

6 浏览器下载web服务器返回的数据及解析html源文件；

7 生成DOM树，解析css和js，渲染页面，直至显示完成；

（

1，浏览器通过HTML parse 根据深度遍历的原则把html 节点遍历成DOM tree

2，将css 解析成CSS DOM tree

3，将DOM tree 和 css DOM tree 构造 render tree

4，layout  根据得到的render tree 计算所有节点在屏幕中的位置）

OSI协议

1 物理层----比特流传输

2 数据链路层----提供介质访问、链路管理等

3 网络层----寻址和路由选择 ---- IP

4 传输层----建立主机端到端链接 ---- TCP（三次握手，四次挥手）

5 会话层----建立维护和管理回话 ---- HTTP DNS 这三层可以统一叫应用层

6 表示层----处理数据格式、数据加密 ---- HTTP DNS这三层可以统一叫应用层

7 应用层----提供应用程序之间的通信 ---- HTTP DNS这三层可以统一叫应用层

HTTP协议（HyperText Transfer Protocol，超文本传输协议）是用于从WWW服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，

　　还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。

请求报文的格式 是行头体

**20 input 事件**

Oninput ---- 监听用户输入，随着用户的输入立即触发

Onchange ---- 监听用户的输入，当失去焦点时触发

**21 排序算法**

一. 冒泡排序

原理：简单来说就是相邻两个元素进行对比，按照你需要的排序方式（升序or降序）进行位置替换，替换时需要额外一个变量当作中间变量去暂存值。

二：选择排序

原理：首先从原始数组中找到最小的元素，并把该元素放在数组的最前面，然后再从剩下的元素中寻找最小的元素，放在之前最小元素的后面，直到排序完毕

三：快速排序

**22 sort深度解析**

Sort是基于原地算法实现的

一个原地算法（in-place algorithm）是一种使用小的，固定数量的额外之空间来转换资料的算法。当算法执行时，输入的资料通常会被要输出的部份覆盖掉。

原理：选择一个基准，将比基准小的放左边，比基准小的放在右边（基准处在中间位置）

如果指明了 compareFunction ，那么数组会按照调用该函数的返回值排序。即 a 和 b 是两个将要被比较的元素：

* 如果 compareFunction(a, b) 小于 0 ，那么 a 会被排列到 b 之前；
* 如果 compareFunction(a, b) 等于 0 ， a 和 b 的相对位置不变。备注： ECMAScript 标准并不保证这一行为，而且也不是所有浏览器都会遵守（例如 Mozilla 在 2003 年之前的版本）；
* 如果 compareFunction(a, b) 大于 0 ， b 会被排列到 a 之前。
* compareFunction(a, b) 必须总是对相同的输入返回相同的比较结果，否则排序的结果将是不确定的。

**23 eval**

eval() 函数可计算某个字符串，并执行其中的的 JavaScript 代码。

**24 let 和 var**

ES6 新增了let命令，用来声明变量。它的用法类似于var，但是所声明的变量，只在let命令所在的代码块内有效。

{

let a = 10;

var b = 1;

}

a // ReferenceError: a is not defined.

b // 1

let 的特点

（1）作用域是块级作用域（在ES6之前，js只存在函数作用域以及全局作用域）

（2）不存在 变量提升 也就是let的变量不会在提升到window属性中

（3）不能重复定义也就是不能重名

（4）存在暂时性死区

if(1){

 console.log(a);

  let a=2;

}

**块级作用域以及暂时性死区**

 ① 在一个块级作用域中，变量唯一存在，一旦在块级作用域中用let声明了一个变量，那么这个变量就唯一属于这个块级作用域，不受外部变量的影响；

② 无论在块中的任何地方声明了一个变量，那么在这个块级作用域中，任何使用这个名字的变量都是指这个变量，无论外部是否有其他同名的全局变量；

③ 暂时性死区的本质就是，只要一进入当前作用域，所要使用的变量就已经存在了，但是不可获取，只有等到声明变量的那一行代码出现，才可以获取和使用该变量。

④ 暂时性死区的意义：让我们标准化代码。将所有的变量的声明放在作用域的最开始

**暂时性死区详细概述**

var tmp = 123;

if (true) {

tmp = 'abc'; // ReferenceError

let tmp;

}

上面代码中，存在全局变量tmp，但是块级作用域内let又声明了一个局部变量tmp，导致后者绑定这个块级作用域，所以在let声明变量前，对tmp赋值会报错。

**块级作用域为什么出现**

1内层变量可能会覆盖外层变量。

var tmp = new Date();

function f() {

console.log(tmp);

if (false) {

var tmp = 'hello world';

}

}

f(); // undefined

2 用来计数的循环变量泄露为全局变量

var s = 'hello';

for (var i = 0; i < s.length; i++) {

console.log(s[i]);

}

console.log(i); // 5

**块级作用域与函数声明**

ES5 规定，函数只能在顶层作用域和函数作用域之中声明，不能在块级作用域声明

ES6 引入了块级作用域，明确允许在块级作用域之中声明函数。ES6 规定，块级作用域之中，函数声明语句的行为类似于let，在块级作用域之外不可引用

考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。

// 函数声明语句

{

let a = 'secret';

function f() {

return a;

}

}

// 函数表达式

{

let a = 'secret';

let f = function () {

return a;

};

}

ES6 的块级作用域允许声明函数的规则，只在使用大括号的情况下成立，如果没有使用大括号，就会报错。

// 不报错

'use strict';

if (true) {

function f() {}

}

// 报错

'use strict';

if (true)

function f() {}

**25 es6 之 Set**

ES6 提供了新的数据结构 Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。

Set本身是一个构造函数，用来生成 Set 数据结构

Set 结构的实例有以下属性。

* Set.prototype.constructor：构造函数，默认就是Set函数。
* Set.prototype.size：返回Set实例的成员总数。

Set 实例的方法分为两大类：操作方法（用于操作数据）和遍历方法（用于遍历成员）。下面先介绍四个操作方法。

* Set.prototype.add(value)：添加某个值，返回 Set 结构本身。
* Set.prototype.delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。
* Set.prototype.has(value)：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，没有返回值。

Set 结构的实例有四个遍历方法，可以用于遍历成员。

* Set.prototype.keys()：返回键名的遍历器
* Set.prototype.values()：返回键值的遍历器
* Set.prototype.entries()：返回键值对的遍历器
* Set.prototype.forEach()：使用回调函数遍历每个成员

由于 Set 结构没有键名，只有键值（或者说键名和键值是同一个值）

**26 Map结构，键名可以是任何类型的键值对集合。**

**Object 和 Map 比较**

对象的键只能为字符串,但是Map的键可以是任意值

可以通过map.size来获取Map的个数,而Object必须要遍历来手动获取

set() 方法为Map对象添加一个指定键（key）和值（value）的新元素。返回Map对象.  
若已经存在键,则更新键值,否则,重新建立新键  
myMap.set(key, value);  
get() 方法用来获取一个 Map 对象中指定的元素。  
myMap.get(key);

**27 es6 之 Symbol**

ES5 的对象属性名都是字符串，这容易造成属性名的冲突。比如，你使用了一个他人提供的对象，但又想为这个对象添加新的方法（mixin 模式），新方法的名字就有可能与现有方法产生冲突。如果有一种机制，保证每个属性的名字都是独一无二的就好了，这样就从根本上防止属性名的冲突。这就是 ES6 引入Symbol的原因。

ES6 引入了一种新的原始数据类型Symbol，表示独一无二的值。它是 JavaScript 语言的第七种数据类型，前六种是：undefined、null、布尔值（Boolean）、字符串（String）、数值（Number）、对象（Object）。

**27 promise**

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。它由社区最早提出和实现，ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象。

所谓Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果。从语法上说，Promise 是一个对象，从它可以获取异步操作的消息。Promise 提供统一的 API，各种异步操作都可以用同样的方法进行处理。

Promise对象有以下两个特点。

**（1）对象的状态不受外界影响**。Promise对象代表一个异步操作，有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。

**（2）一旦状态改变，就不会再变**，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从pending变为fulfilled和从pending变为rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果，这时就称为 resolved（已定型）。如果改变已经发生了，你再对Promise对象添加回调函数，也会立即得到这个结果。这与事件（Event）完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

注意，为了行文方便，本章后面的resolved统一只指fulfilled状态，不包含rejected状态。

有了Promise对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数(回调地狱)。此外，Promise对象提供统一的接口，使得控制异步操作更加容易。

Promise缺点。首先，无法取消Promise，一旦新建它就会立即执行，无法中途取消。其次，如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。第三，当处于pending状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。

基本用法

ES6 规定，Promise对象是一个构造函数，用来生成Promise实例。

const promise = new Promise(function(resolve, reject) {

// ... some code

if (/\* 异步操作成功 \*/){

resolve(value);

} else {

reject(error);

}

});

resolve函数的作用是，将Promise对象的状态从“未完成”变为“成功”（即从 pending 变为 resolved），在异步操作成功时调用，并将异步操作的结果，作为参数传递出去；reject函数的作用是，将Promise对象的状态从“未完成”变为“失败”（即从 pending 变为 rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去。

**例子 通过promise封装ajax**

const getJSON = function(url) {

const promise = new Promise(function(resolve, reject){

const handler = function() {

if (this.readyState !== 4) {

return;

}

if (this.status === 200) {

resolve(this.response);

} else {

reject(new Error(this.statusText));

}

};

const client = new XMLHttpRequest();

client.open("GET", url);

client.onreadystatechange = handler;

client.responseType = "json";

client.setRequestHeader("Accept", "application/json");

client.send();

});

return promise;

};

getJSON("/posts.json").then(function(json) {

console.log('Contents: ' + json);

}, function(error) {

console.error('出错了', error);

});

**Promise.prototype.then()**

then方法返回的是一个新的Promise实例（注意，不是原来那个Promise实例）。因此可以采用链式写法，即then方法后面再调用另一个then方法。

getJSON("/post/1.json").then(function(post) {

return getJSON(post.commentURL);

}).then(function (comments) {

console.log("resolved: ", comments);

}, function (err){

console.log("rejected: ", err);

});

**Promise.prototype.catch()**

Promise.prototype.catch方法是.then(null, rejection)或.then(undefined, rejection)的别名，用于指定发生错误时的回调函数。

如果 Promise 状态已经变成resolved，再抛出错误是无效的。

Promise 对象的错误具有“冒泡”性质，会一直向后传递，直到被捕获为止。也就是说，错误总是会被下一个catch语句捕获。Promise会自己吃掉错误

getJSON('/post/1.json').then(function(post) {

return getJSON(post.commentURL);

}).then(function(comments) {

// some code

}).catch(function(error) {

// 处理前面三个Promise产生的错误

});

上面代码中，一共有三个 Promise 对象：一个由getJSON产生，两个由then产生。它们之中任何一个抛出的错误，都会被最后一个catch捕获。**Promise.prototype.finally()**

finally方法用于指定不管 Promise 对象最后状态如何，都会执行的操作。该方法是 ES2018 引入标准的。

**28 Generator(宅呢瑞特)**

Generator 函数是 ES6 提供的一种异步编程解决方案Generator 函数可以返回一系列的值，因为可以有任意多个yield（呀的）。从另一个角度看，也可以说 Generator 生成了一系列的值，这也就是它的名称的来历（英语中，generator 这个词是“生成器”的意思）。done属性的值false，表示遍历还没有结束。

function\* helloWorldGenerator() {

yield 'hello';

yield 'world';

return 'ending';

}

var hw = helloWorldGenerator();

hw.next()

// { value: 'hello', done: false }

hw.next()

// { value: 'world', done: false }

hw.next()

// { value: 'ending', done: true }

hw.next()

// { value: undefined, done: true }

yield表达式与return语句既有相似之处，也有区别。相似之处在于，都能返回紧跟在语句后面的那个表达式的值。区别在于每次遇到yield，函数暂停执行，下一次再从该位置继续向后执行，而return语句不具备位置记忆的功能。一个函数里面，只能执行一次（或者说一个）return语句，但是可以执行多次（或者说多个）yield表达式

**29 异步概念**

异步

所谓"异步"，简单说就是一个任务不是连续完成的，可以理解成该任务被人为分成两段，先执行第一段，然后转而执行其他任务，等做好了准备，再回过头执行第二段。

比如，有一个任务是读取文件进行处理，任务的第一段是向操作系统发出请求，要求读取文件。然后，程序执行其他任务，等到操作系统返回文件，再接着执行任务的第二段（处理文件）。这种不连续的执行，就叫做异步。

相应地，连续的执行就叫做同步。由于是连续执行，不能插入其他任务，所以操作系统从硬盘读取文件的这段时间，程序只能干等着。

JavaScript 语言对异步编程的实现，就是回调函数

Es6之前解决异步编程的方法

* 回调函数
* 事件监听
* 发布/订阅
* Promise 对象

**协程**

传统的编程语言，早有异步编程的解决方案（其实是多任务的解决方案）。其中有一种叫做"协程"（coroutine），意思是多个线程互相协作，完成异步任务。

协程有点像函数，又有点像线程。它的运行流程大致如下。

* 第一步，协程A开始执行。
* 第二步，协程A执行到一半，进入暂停，执行权转移到协程B。
* 第三步，（一段时间后）协程B交还执行权。
* 第四步，协程A恢复执行。

Generator 函数是协程在 ES6 的实现，最大特点就是可以交出函数的执行权（即暂停执行）

**30 class**

**基本写法**

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}

// 正确

var point = new Point(2, 3);

prototype对象的constructor属性，直接指向“类”的本身，这与 ES5 的行为是一致的。

**取值函数（getter）和存值函数（setter）**

与 ES5 一样，在“类”的内部可以使用get和set关键字，对某个属性设置存值函数和取值函数，拦截该属性的存取行为。

class MyClass {

constructor() {// ...

}

get prop() {

return 'getter';

}

set prop(value) {

console.log('setter: '+value);

}

}

let inst = new MyClass();

inst.prop = 123;

// setter: 123

inst.prop

// 'getter'

继承

class ColorPoint extends Point {

constructor(x, y, color) {

super(x, y); // 调用父类的constructor(x, y)

this.color = color;

}

toString() {

return this.color + ' ' + super.toString(); // 调用父类的toString()

}

}

上面代码中，constructor方法和toString方法之中，都出现了super关键字，它在这里表示父类的构造函数，用来新建父类的this对象。

**子类必须在constructor方法中调用super方法**，否则新建实例时会报错。这是因为子类自己的this对象，必须先通过父类的构造函数完成塑造，得到与父类同样的实例属性和方法，然后再对其进行加工，加上子类自己的实例属性和方法。如果不调用super方法，子类就得不到this对象。

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

}

class ColorPoint extends Point {

constructor(x, y, color) {

this.color = color; // ReferenceError

super(x, y);

this.color = color; // 正确

}

}

**31 严格模式主要有以下限制。**

* 变量必须声明后再使用
* 函数的参数不能有同名属性，否则报错
* 不能使用with语句
* 不能对只读属性赋值，否则报错
* 不能使用前缀 0 表示八进制数，否则报错
* 不能删除不可删除的属性，否则报错
* 不能删除变量delete prop，会报错，只能删除属性delete global[prop]
* eval不会在它的外层作用域引入变量
* eval和arguments不能被重新赋值
* arguments不会自动反映函数参数的变化
* 不能使用arguments.callee
* 不能使用arguments.caller
* 禁止this指向全局对象
* 不能使用fn.caller和fn.arguments获取函数调用的堆栈
* 增加了保留字（比如protected、static和interface）

**32 export import export default**

模块功能主要由两个命令构成：export和import。export命令用于规定模块的对外接口，import命令用于输入其他模块提供的功能

为了给用户提供方便，让他们不用阅读文档就能加载模块，就要用到export default命令，为模块指定默认输出。export 之后加上default意指默认接口的意思，

export 与import是es6中新增模块功能最主要的两个命令。我们要知道在es6中，实现了模块功能，而且相当简单，意在取代commonjs和AMD规范。成为一种模块化的通用解决方案。

其设计思想为尽量的静态化，我们可以理解为在加载之前就知道自己所需要依赖的是哪个文件，而非在运行的时候才知道依赖谁。es6的那模块不是对象，通过export输出指令，通过import输入。就目前来说 不用一些编译器的话 最新版的浏览器是不识别import这个关键字的

在es6模块中是自动采用严格模式的

**33 解构赋值**

**数组的解构赋值**

let [a, b, c] = [1, 2, 3];

本质上，这种写法属于“模式匹配”，只要等号两边的模式相同，左边的变量就会被赋予对应的值。

如果等号的右边不是数组（或者严格地说，不是可遍历的结构，参见《Iterator》一章），那么将会报错。

// 报错

let [foo] = 1;

let [foo] = false;

let [foo] = NaN;

let [foo] = undefined;

let [foo] = null;

let [foo] = {};

对于 Set 结构，也可以使用数组的解构赋值。

let [x, y, z] = new Set(['a', 'b', 'c']);

x // "a"

let [x = 1] = [undefined];

x // 1

let [x = 1] = [null];

x // null

上面代码中，如果一个数组成员是null，默认值就不会生效，因为null不严格等于undefined。

**对象的解构赋值**

解构不仅可以用于数组，还可以用于对象。

let { foo, bar } = { foo: 'aaa', bar: 'bbb' };

foo // "aaa"

bar // "bbb"

对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值。

默认值生效的条件是，对象的属性值严格等于undefined。

var {x = 3} = {x: undefined};

x // 3

var {x = 3} = {x: null};

x // null

**注意点**

（1）如果要将一个已经声明的变量用于解构赋值，必须非常小心。

// 错误的写法

let x;

{x} = {x: 1};

// SyntaxError: syntax error

// 正确的写法

let x;

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

上面代码的写法会报错，因为 JavaScript 引擎会将{x}理解成一个代码块，从而发生语法错误。只有不将大括号写在行首，避免 JavaScript 将其解释为代码块，才能解决这个问题。

**字符串的解构赋值**

字符串也可以解构赋值。这是因为此时，字符串被转换成了一个类似数组的对象。

const [a, b, c, d, e] = 'hello';

a // "h"

b // "e"

c // "l"

d // "l"

**34 for …of 和for…in**

JavaScript 原有的for...in循环，只能获得对象的键名，不能直接获取键值。ES6 提供for...of循环，允许遍历获得键值。

一个数据结构只要部署了Symbol.iterator属性，就被视为具有 iterator（伊特瑞德） 接口，就可以用for...of循环遍历它的成员。也就是说，for...of循环内部调用的是数据结构的Symbol.iterator方法。

**Set 和 Map 结构**

Set 和 Map 结构也原生具有 Iterator 接口，可以直接使用for...of循环。

var engines = new Set(["Gecko", "Trident", "Webkit", "Webkit"]);

for (var e of engines) {

console.log(e);

}

// Gecko

// Trident

// Webkit

var es6 = new Map();

es6.set("edition", 6);

es6.set("committee", "TC39");

es6.set("standard", "ECMA-262");

for (var [name, value] of es6) {

console.log(name + ": " + value);

}

// edition: 6

// committee: TC39

// standard: ECMA-262

上面代码演示了如何遍历 Set 结构和 Map 结构。值得注意的地方有两个，首先，遍历的顺序是按照各个成员被添加进数据结构的顺序。其次，Set 结构遍历时，返回的是一个值，而 Map 结构遍历时，返回的是一个数组，该数组的两个成员分别为当前 Map 成员的键名和键值。

For …of 实现循环对象

一种解决方法是，使用Object.keys方法将对象的键名生成一个数组，然后遍历这个数组。

for (var key of Object.keys(someObject)) {

console.log(key + ': ' + someObject[key]);

}

**25 axios**

特点（体积小 提供了并发请求的封装）

1从浏览器中创建 [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \t "_blank)

2从 node.js 创建 [http](http://nodejs.org/api/http.html" \t "_blank) 请求

3支持 [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise" \t "_blank) API

4拦截请求和响应

5转换请求数据和响应数据

6取消请求

7自动转换 JSON 数据

8客户端支持防御 [XSRF](http://en.wikipedia.org/wiki/Cross-site_request_forgery" \t "_blank)

基本使用

1配置全局默认选项

axios.defaults.baseURL = 'https://api.example.com';

// `transformRequest` 允许在向服务器发送前，修改请求数据

// 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法

// 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream

transformRequest: [function (data) {

// 对 data 进行任意转换处理

return data;

}],

// `transformResponse` 在传递给 then/catch 前，允许修改响应数据

transformResponse: [function (data) {

// 对 data 进行任意转换处理

return data;

}],

拦截器interceptors(in特塞克特)

// 添加请求拦截器

axios.interceptors.request.use(function (config) {

// 在发送请求之前做些什么

return config;

}, function (error) {

// 对请求错误做些什么

return Promise.reject(error);

});

// 添加响应拦截器

axios.interceptors.response.use(function (response) {

// 对响应数据做点什么

return response;

}, function (error) {

// 对响应错误做点什么

return Promise.reject(error);

});

**Fetch**

fetch号称是AJAX的替代品，是在ES6出现的，使用了ES6中的promise对象。Fetch是基于promise设计的。Fetch的代码结构比起ajax简单多了，参数有点像jQuery ajax。但是，一定记住**fetch不是ajax的进一步封装，而是原生js，没有使用XMLHttpRequest对象**。

**jQuery ajax**

$.ajax({

type: 'POST',

url: url,

data: data,

dataType: dataType,

success: function () {},

error: function () {}

});

传统 Ajax 指的是 XMLHttpRequest（XHR）， 最早出现的发送后端请求技术，隶属于原始js中，核心使用XMLHttpRequest对象，多个请求之间如果有先后关系的话，就会出现**回调地狱**。

**浏览器与网络**

**1 回流与重绘**

重绘与回流

* 重绘(repaint): 当元素样式的改变不影响布局时，浏览器将使用重绘对元素进行更新，此时由于只需要UI层面的重新像素绘制，因此 损耗较少
* 回流(reflow): 当元素的尺寸、结构或触发某些属性时，浏览器会重新渲染页面，称为回流。此时，浏览器需要重新经过计算，计算后还需要重新页面布局，因此是较重的操作。会触发回流的操作:  
  \* 页面初次渲染  
  \* 浏览器窗口大小改变  
  \* 元素尺寸、位置、内容发生改变  
  \* 元素字体大小变化  
  \* 添加或者删除可见的 dom 元素  
  \* 激活 CSS 伪类（例如：:hover）  
  \* 查询某些属性或调用某些方法  
  \* clientWidth、clientHeight、clientTop、clientLeft  
  \* offsetWidth、offsetHeight、offsetTop、offsetLeft  
  \* scrollWidth、scrollHeight、scrollTop、scrollLeft  
  \* getComputedStyle()  
  \* getBoundingClientRect()  
  \* scrollTo()  
  回流必定触发重绘，重绘不一定触发回流。重绘的开销较小，回流的代价较高。

**2 浏览器状态码**

**1XX信息**

服务器收到请求，需要请求者继续执行操作

100:Continue继续。客户端应继续其请求

101:Switching Protocols切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议

**2XX 成功**

http状态码 200 （成功） 服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

http状态码 201 （已创建） 请求成功并且服务器创建了新的资源。

http状态码 202 （已接受） 服务器已接受请求，但尚未处理。

http状态码 203 （非授权信息） 服务器已成功处理了请求，但返回的信息可能来自另一来源。

http状态码 204 （无内容） 服务器成功处理了请求，但没有返回任何内容。

http状态码 205 （重置内容） 服务器成功处理了请求，但没有返回任何内容。

http状态码 206 （部分内容） 服务器成功处理了部分 GET 请求。

**3XX 重定向**

http状态码 300 （多种选择） 针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。

http状态码 301 （永久移动） 请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

http状态码 302 （临时移动） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

http状态码 303 （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。

http状态码 304 （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。

http状态码 305 （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。

http状态码 307 （临时重定向） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

**4XX 客户端错误**

http状态码 400 （错误请求） 服务器不理解请求的语法。

http状态码 401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

http状态码 403 （禁止） 服务器拒绝请求。

http状态码 404 （未找到） 服务器找不到请求的网页。

http状态码 405 （方法禁用） 禁用请求中指定的方法。

http状态码 406 （不接受） 无法使用请求的内容特性响应请求的网页。

http状态码 407 （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。

http状态码 408 （请求超时） 服务器等候请求时发生超时。

http状态码 409 （冲突） 服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。

http状态码 410 （已删除） 如果请求的资源已永久删除，服务器就会返回此响应。

http状态码 411 （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。

http状态码 412 （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。

http状态码 413 （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。

http状态码 414 （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。

http状态码 415 （不支持的媒体类型） 请求的格式不受请求页面的支持。

http状态码 416 （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。

http状态码 417 （未满足期望值） 服务器未满足”期望”请求标头字段的要求。

**5XX 服务器错误**

500 （服务器内部错误） 服务器遇到错误，无法完成请求。

501 （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

502 （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。

503 （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。

504 （网关超时） 服务器作为网关或代理，但是没有及时从上游服务器收到请求。

505 （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。

**3 前端安全XSS、CSRF**

Csrf ---- 跨站请求伪造

受害者必须满足两个必要的条件：

（1）登录受信任网站A，并在本地生成Cookie。（如果用户没有登录网站A，那么网站B在诱导的时候，请求网站A的api接口时，会提示你登录）

（2）在不登出A的情况下，访问危险网站B（其实是利用了网站A的漏洞）。

Csrf如何防御

方法一、Token 验证：（用的最多）

（1）服务器发送给客户端一个token；

（2）客户端提交的表单中带着这个token。

（3）如果这个 token 不合法，那么服务器拒绝这个请求。

方法二：隐藏令牌：

把 token 隐藏在 http 的 head头中。

方法二和方法一有点像，本质上没有太大区别，只是使用方式上有区别。

**4 get和post**



Get 长度限制原因

浏览器地址栏输入的参数是有限的

本质问题

GET 和 POST 只是 HTTP 协议中两种请求方式（异曲同工），而 HTTP 协议是基于 TCP/IP 的应用层协议，无论 GET 还是 POST，用的都是同一个传输层协议，所以在传输上，没有区别

安全问题

有人说POST 比 GET 安全，因为数据在地址栏上不可见。

然而，从传输的角度来说，他们都是不安全的，因为 HTTP 在网络上是明文传输的，只要在网络节点上捉包，就能完整地获取数据报文。

要想安全传输，就只有加密，也就是 HTTPS。

**5 即时通信 socket.io**

Socket.io将数据传输部分独立出来形成engine.io，engine.io对WebSocket和AJAX轮询进行了封装，形成了一套API，屏蔽了细节差异和兼容性问题，实现了跨浏览器/跨设备进行双向数据通信。

**Vue**

**1 vue解决了什么问题**

vue是一套构建用户界面的 渐进式框架 采用自底向上增量开发的设计。Vue 的核心库只关注视图层

vue 的双向数据绑定是通过 数据劫持 + 发布 订阅模式 实现的

数据劫持 object.defineProperty 中的 getter 和 setter

**2 如何实现一个自定义组件，不同组件之间如何通信的**

组件需要注册后才可以使用，有全局注册和局部注册两种方式

Vue.component（’组件名’,{}）注册全局组件

Let app ={} 注册局部组件

**组件之间的通信**

父子组件----- 父组件--🡪 子组件通过props 接收数据

子组件 -🡪 父组件通过 $on 响应 子组件通过 $emit 发送

**3 MVVM开发模式的理解**

1 MVVM分为Model、View、ViewModel三者。  
Model 代表数据模型，数据和业务逻辑都在Model层中定义；  
View 代表UI视图，负责数据的展示；  
ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；  
Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。  
这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 dom。

**4 v-if 和 v-show 有什么区别？**

v-show 仅仅控制元素的显示方式，将 display 属性在 block 和 none 来回切换；而v-if会控制这个 DOM 节点的存在与否。当我们需要经常切换某个元素的显示/隐藏时，使用v-show会更加节省性能上的开销；当只需要一次显示或隐藏时，使用v-if更加合理。

**5 delete和Vue.delete删除数组的区别**

delete只是被删除的元素变成了 empty/undefined 其他的元素的键值还是不变。  
Vue.delete直接删除了数组 改变了数组的键值。

**小程序和H5的区别**

运行环境

原生 App 直接运行在操作系统的单独进程中（在 Android 中还可以开启多进程），而小程序只能运行在微信的进程中。

开发成本

原生 App 的开发涉及到 Android/iOS 多个平台、开发工具、开发语言、不同设备的适配等问题；而小程序只需要开发一个就可以在 Android/iOS 等不同平台不同设备上运行。

原生 App 需要在商店上架（Android 需要上架各种商店）；小程序只能在微信平台发布。

系统权限

原生 App 调用的是系统资源，也就是说系统提供给开发的的 API 都可以使用；而小程序是基于微信的，小程序所有的功能都受限于微信，也就是说微信给开发者提供 API 才可以使用，不能绕过微信直接使用系统提供的 API。

原生 App 可以给用户推送消息；小程序不允许主动给用户发送消息，只能回复模版消息

原生 App 有独立的数据库，可以做离线存储；小程序只能存储到 LocalStorage，无法做离线存储。

原生 App 需要下载，安装包比较大；小程序无需下载，可以通过小程序码等方式通过微信直接打开。

运行流畅度

原生 App 运行在操作系统中，所有的原生组件可以直接调用 GPU 进行渲染；而小程序运行在微信的进程中，只能通过 WebView 进行渲染。

**扩展**

**1 %d,%c,%s,%x是程序汇编语言中的格式符**

1、%d表示按整型数据的实际长度输出数据。

2、%c用来输出一个字符。

3、%s用来输出一个字符串。

var name = 788

console.log('你想要的是%s' , name)

4、%x表示以十六进制数形式输出整数。

**Git基本命令**

1.本地初始化一个git库

在你要上传的文件夹中打开命令窗口：git init；然后他会自动生成.git文件

2.关联远程库:命令行：git remote add origin xxxxx

3. 将修改过的文件提交暂存区  git add .   (这里的  .  是指提交全部文件)

4.git status 查看状态，绿色  new file:test2.html，是指没有确定提交

5.接下来就是确定提交了。git commit -m "test2"

6将本地主分支推到远程：git push -u origin master

7通过git pull --rebase origin master命令来更新，你会发现你的Test2目录下面多了一个文件（git pull命令的作用是，取回远程主机某个分支的更新，再与本地的         指定分支合并）

8 git config --global user.name”用户名(建议是英文)”设置全局变量

**Unicode**

Unicode（统一码、万国码、单一码）是一种在计算机上使用的字符编码。它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。1990年开始研发，1994年正式公布。随着计算机工作能力的增强，Unicode也在面世以来的十多年里得到普及。

utf-8作为现在最广泛使用的编码方式，了解一下其工作原理还是有必要的。utf-8离不开unicode，utf-8编码出现的目的是为对unicode的编码值进行压缩，这才是utf-8编码的最核心意义。

##### 说出至少5个ES6的新特性，并简述它们的作用。（简答题）

答：  
     1、let关键字，用于声明只在块级作用域起作用的变量。  
     2、const关键字，用于声明一个常量。  
     3、解构赋值，一种新的变量赋值方式。常用于交换变量值，提取函数返回值，设置默认值。  
     4、Symbol数据类型，定义一个独一无二的值。  
     5、Proxy代理，用于编写处理函数，来拦截目标对象的操作。  
     6、for...of遍历，可遍历具有iterator 接口的数据结构。  
     7、Set结构，存储不重复的成员值的集合。  
     8、Map结构，键名可以是任何类型的键值对集合。  
     9、Promise对象，更合理、规范地处理异步操作。  
     10、Class类定义类和更简便地实现类的继承。