前端基础面试题

11，事件模型
12，new操作符具体干了什么呢?
13，Ajax原理
14，Ajax解决浏览器缓存问题
15，如何解决跨域问题?
16，说说你对AMD和Commonjs的理解
17，js的7种基本数据类型
18，介绍js有哪些内置对象
19，JS有哪些方法定义对象
20，你觉得jQuery源码有哪些写的好的地方
21，null，undefined 的区别
22，谈谈你对ES6的理解
23，面向对象编程思想
24，如何通过JS判断一个数组
25，异步编程的实现方式
26，对原生Javascript了解方向
27，sort 快速打乱数组
28，数组去重操作
29，JS 原生拖拽节点
30，深拷贝、浅拷贝
31，节流防抖
32，变量提升
33，JS中的垃圾回收机制
34，如何理解前端模块化
35，js单线程
36，说说event loop
37，描述下this
38，ajax、axios、fetch区别

浏览器

1，浏览器存储的方式有哪些
2，浏览器内核的理解
3，HTTP 的请求方式场景
4，HTTP状态码
5，从浏览器地址栏输入URL后发生了什么?
6，请你谈谈Cookie的优缺点
7，cookies， sessionStorage 和 localStorage 的区别
8，浏览器缓存
9，浏览器渲染的步骤
10，GET 和 POST 请求的区别
11，什么是reflow
12，什么时候会导致reflow发生呢?
13，减少reflow对性能的影响

性能优化

1，SEO优化
2，server优化
3，css优化
4，js方面
5，webpack优化点
6，加载优化:
7，页面渲染优化
8，图片优化
9，脚本优化
10，为什么利用多个域名来存储网站资源会更有效?

其他问题

HTML

1，你如何理解HTML结构的语义化

- 更符合W3C统一的规范标准，是技术趋势。
- 没有样式时浏览器的默认样式也能让页面结构很清晰。
- 对功能障碍用户友好。屏幕阅读器（如果访客有视障）会完全根据你的标记来“读”你的网页。
- 对其他非主流终端设备友好。例如机顶盒、PDA、各种移动终端。
- 对SEO友好。

2，说说 title 和 alt 属性

- 两个属性都是当鼠标滑动到元素上的时候显示
- `alt` 是 `img` 的特有属性，是图片内容的等价描述，图片无法正常显示时候的替代文字。
- `title` 属性可以用在除了base，basefont，head，html，meta，param，script和title之外的所有标签，是对dom元素的一种类似注释说明，

3，Html5 有哪些新特性、移除了哪些元素

- 新增元素：
  - 绘画 `canvas`
  - 用于媒介回放的 `video 和 audio` 元素
  - 本地离线存储 `localStorage` 长期存储数据，浏览器关闭后数据不丢失
  - `sessionStorage` 的数据在浏览器关闭后自动删除
  - 语义化更好的内容元素，比如 `article 、footer、header、nav、section`
  - 表单控件，`calendar 、 date 、 time 、 email 、 url 、 search`
  - 新的技术 `webworker 、 websocket 、 Geolocation`
- 移除的元素：
  - 纯表现的元素：`basefont 、 big 、 center 、 font 、 s 、 strike 、 tt 、 u`
  - 对可用性产生负面影响的元素：`frame 、 frameset 、 noframes`
- 支持 HTML5 新标签：
  - `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

4，Label的作用是什么？是怎么用的?

label标签来定义表单控制间的关系,当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

```
<label for="Name">Number:</label>
<input type=“text“name="Name" id="Name"/>
<label>Date:<input type="text" name="B"/></label>
```

5，浏览器是怎么对 Html5 的离线储存资源进行管理和加载的呢

如何使用：

- 页面头部像下面一样加入一个 `manifest` 的属性；
- 在 `cache.manifest` 文件的编写离线存储的资源
- 在离线状态时，操作 `window.applicationCache` 进行需求实现

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。

6，iframe有那些缺点?

- `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
- 使用 `iframe` 之前需要考虑这两个缺点。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

7，HTML W3C的标准

标签闭合、标签小写、不乱嵌套、使用外链 css 和 js 、结构行为表现的分离

8，Doctype作用? 严格模式与混杂模式如何区分？它们有何意义?

- `<!DOCTYPE>` 声明位于文档中的最前面，处于 `html` 标签之前。告知浏览器的解析器， 用什么文档类型、规范来解析这个文档
- 严格模式的排版和 `JS` 运作模式是 以该浏览器支持的最高标准运行
- 在混杂模式中，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 `DOCTYPE` 不存在或格式不正确会导致文档以混杂模式呈现

9，HTML全局属性(global attribute)有哪些

- `class` :为元素设置类标识
- `data-*` : 为元素增加自定义属性
- `draggable` : 设置元素是否可拖拽
- `id` : 元素 id ， 文档内唯一
- `lang` : 元素内容的的语言
- `style` : 行内 css 样式
- `title` : 元素相关的建议信息

10，viewport的content属性作用

```
<meta name="viewport" content="" />
    width viewport的宽度[device-width | pixel_value]width如果直接设置pixel_value数值，大部分的安卓手机不支持，但是ios支持；
    height - viewport 的高度 （范围从 223 到 10,000 ）
    user-scalable [yes | no]是否允许缩放
    initial-scale [数值] 初始化比例（范围从 > 0 到 10）
    minimum-scale [数值] 允许缩放的最小比例
    maximum-scale [数值] 允许缩放的最大比例
    target-densitydpi 值有以下（一般推荐设置中等响度密度或者低像素密度，后者设置具体的值dpi_value，另外webkit内核已不准备再支持此属性）
          -- dpi_value 一般是70-400//没英寸像素点的个数
          -- device-dpi设备默认像素密度
          -- high-dpi 高像素密度
          -- medium-dpi 中等像素密度
          -- low-dpi 低像素密度
```

附带问题：怎样处理 移动端 1px 被 渲染成 2px 问题?

局部处理：

- `mate` 标签中的 `viewport` 属性， `initial-scale` 设置为 1
- `rem` 按照设计稿标准走，外加利用`transfrome 的 scale(0.5)`缩小一倍即可；

全局处理：

- `mate` 标签中的 `viewport` 属性， `initial-scale` 设置为 0.5
- `rem` 按照设计稿标准走即可

11，meta 相关

```
<!DOCTYPE html> <!--H5标准声明，使用 HTML5 doctype，不区分大小写-->
<head lang="en"> <!--标准的 lang 属性写法-->
<meta charset='utf-8'> <!--声明文档使用的字符编码-->
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/> <!--优先使用指定浏览器使用特定的文档模式-->
<meta name="description" content="不超过150个字符"/> <!--页面描述-->
<meta name="keywords" content=""/> <!-- 页面关键词-->
<meta name="author" content="name, email@gmail.com"/> <!--网页作者-->
<meta name="robots" content="index,follow"/> <!--搜索引擎抓取-->
<meta name="viewport" content="initial-scale=1, maximum-scale=3, minimum-sc
<meta name="apple-mobile-web-app-title" content="标题"> <!--iOS 设备 begin-->
<meta name="apple-mobile-web-app-capable" content="yes"/> <!--添加到主屏后的标是否启用 WebApp 全屏模式，删除苹果默认的工具栏和菜单栏-->
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
<meta name="renderer" content="webkit"> <!-- 启用360浏览器的极速模式(webkit)-->
<meta http-equiv="X-UA-Compatible" content="IE=edge"> <!--避免IE使用兼容模式-->
<meta http-equiv="Cache-Control" content="no-siteapp" /> <!--不让百度转码-->
<meta name="HandheldFriendly" content="true"> <!--针对手持设备优化，主要是针对一些老的不识别viewport的浏览器-->
<meta name="MobileOptimized" content="320"> <!--微软的老式浏览器-->
<meta name="screen-orientation" content="portrait"> <!--uc强制竖屏-->
<meta name="x5-orientation" content="portrait"> <!--QQ强制竖屏-->
<meta name="full-screen" content="yes"> <!--UC强制全屏-->
<meta name="x5-fullscreen" content="true"> <!--QQ强制全屏-->
```

```
<meta name="browsermode" content="application"> <!--UC应用模式-->
<meta name="x5-page-mode" content="app"> <!-- QQ应用模式-->
<meta name="msapplication-tap-highlight" content="no"> <!--windows phone
设置页面不缓存-->
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="expires" content="0">
```

12，div+css的布局较table布局有什么优点

- 改版的时候更方便 只要改 css 文件。
- 页面加载速度更快、结构化清晰、页面显示简洁。
- 表现与结构相分离。
- 易于优化（ seo ）搜索引擎更友好，排名更容易靠前。

13，简述一下src与href的区别

- `src` 用于替换当前元素，`href` 用于在当前文档和引用资源之间确立联系。
- `src` 是 `source` 的缩写，指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求 `src` 资源时会将其指向的资源下载并应用到文档内，例如 `js` 脚本，`img` 图片和 `frame` 等元素

> 当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部

`href 是 Hypertext Reference` 的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果我们在文档中添加 `link href="common.css" rel="stylesheet"` 那么浏览器会识别该文档为 css 文件，就会并行下载资源并且不会停止对当前文档的处理。这也是为什么建议使用 link方式来加载 css，而不是使用 @import 方式

14，知道的网页制作会用到的图片格式有哪些

png-8，png-24，jpeg，gif，svg。

但是上面的那些都不是面试官想要的最后答案。面试官希望听到是Webp。（是否有关注新技术，新鲜事物）

科普一下Webp：WebP格式，谷歌（google）开发的一种旨在加快图片加载速度的图片格式。图片压缩体积大约只有JPEG的2/3，并能节省大量的服务器带宽资源和数据空间。Facebook Ebay等知名网站已经开始测试并使用WebP格式。

在质量相同的情况下，WebP格式图像的体积要比JPEG格式图像小40%

15， 如何在 HTML5页面中嵌入音频与视频?

HTML 5 包含嵌入音频文件的标准方式，支持的格式包括 MP3、Wav 和 Ogg：

```
<audio controls>
 <source src="jamshed.mp3" type="audio/mpeg">
  Your browser does'nt support audio embedding feature.
</audio>
```

和音频一样，HTML5 定义了嵌入视频的标准方法，支持的格式包括：MP4、WebM 和 Ogg：

```
<video width="450" height="340" controls>
 <source src="jamshed.mp4" type="video/mp4">
  Your browser does'nt support video embedding feature.
</video>
```

CSS

1，水平居中的方法

- 元素为行内元素，设置父元素 `text-align:center`
- 如果元素宽度固定，可以设置左右 `margin 为 auto`;
- 如果元素为绝对定位，设置父元素 `position 为 relative`，元素设 `left:0;right:0;margin:auto;`
- 使用 `flex-box` 布局，指定 `justify-content` 属性为`center`
- `display` 设置为 `tabel-ceil`

2，垂直居中的方法

- 将显示方式设置为表格， `display:table-cell`,同时设置 `vertial-align：middle`
- 使用 `flex` 布局，设置为 `align-item：center`
- 绝对定位中设置 `bottom:0,top:0`,并设置 `margin:auto`
- 绝对定位中固定高度时设置`top:50%，margin-top`值为高度一半的负值
- 文本垂直居中设置 `line-height` 为 `height`值

3，link 与 @import 的区别

- `link` 是 `HTML` 方式， `@import` 是`CSS`方式
- `link` 最大限度支持并行下载， `@import` 过多嵌套导致串行下载，出现 FOUC (文档样式短暂失效)
- `link`可以通过 `rel="alternate stylesheet"`指定候选样式
- 浏览器对 `link`支持早于 `@import` ，可以使用 `@import` 对老浏览器隐藏样式
- `@import` 必须在样式规则之前，可以在`css`文件中引用其他文件
- 总体来说： `link` 优于 `@import` ， `link`优先级也更高

4，三列布局（中间固定两边自适应宽度）

1）采用浮动布局（左边左浮动，右边右浮动，中间margin：0 宽度值）

2）绝对定位方式（左右绝对定位，左边left0右边right0，中间上同）

3）圣杯布局与双飞翼布局

5，BFC 有什么用

- 创建规则：
  - 根元素
  - 浮动元素（ `float` 不取值为 `none` ）
  - 绝对定位元素（ `position` 取值为 `absolute` 或 `fixed` ）
  - `display` 取值为 `inline-block` 、 `table-cell` 、 `table-caption` 、 `flex` 、`inline-flex` 之一的元素
  - `overflow` 不取值为 `visible` 的元素
- 作用
  - 可以包含浮动元素
  - 不被浮动元素覆盖
  - 阻止父子元素的 `margin` 折叠

6，清除浮动的几种方式

- 父级 `div` 定义 `height`
- 结尾处加空 `div` 标签 `clear:both`
- 父级 `div` 定义伪类 `:after` 和 `zoom`
- 父级 `div` 定义 `overflow:hidden`
- 父级 `div` 也浮动，需要定义宽度
- 结尾处加 `br` 标签 `clear:both`

7，CSS3有哪些新特性

- 新增各种CSS选择器 （: not(.input)：所有 class 不是“input”的节点）
- 圆角 （border-radius:8px）
- 多列布局 （column）
- 阴影和反射 （Shadow\Reflect）
- 文字特效 （text-shadow）
- 线性渐变 （gradient）
- 旋转，缩放,定位,倾斜 （transform）
- 动画 （Animation）
- 多背景，背景裁剪

8，css3 新增伪类 - 伪元素

- `p:first-of-type` 选择属于其父元素的首个

  元素的每个

  元素。
- `p:last-of-type` 选择属于其父元素的最后

  元素的每个

  元素。
- `p:only-of-type` 选择属于其父元素唯一的

  元素的每个

  元素。
- `p:only-child` 选择属于其父元素的唯一子元素的每个

  元素。
- `p:nth-child(2)` 选择属于其父元素的第二个子元素的每个

  元素。
- `:enabled` 已启用的表单元素。
- `:disabled` 已禁用的表单元素。
- `:checked` 单选框或复选框被选中。
- `::before` 在元素之前添加内容。
- `::after` 在元素之后添加内容,也可以用来做清除浮动。
- `::first-line` 添加一行特殊样式到首行。
- `::first-letter` 添加一个特殊的样式到文本的首字母。
- 伪类语法一个：，它是为了弥补css常规类选择器的不足
- 伪元素语法两个：，它是凭空创建的一个虚拟容器生成的元素

9，为什么要初始化CSS样式

因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对CSS初始化往往会出现浏览器之间的页面显示差异。

当然，初始化样式会对SEO有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。

最简单的初始化方法：

```
* {
padding: 0;
margin: 0;
}
```

淘宝的样式初始化代码：

```
body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li, pre,
form, fieldset, legend, button, input, textarea, th, td {
margin:0;
padding:0;
}

body, button, input, select, textarea {
```

```
font:12px/1.5tahoma, arial;
}

h1, h2, h3, h4, h5, h6{
font-size:100%;
}

address, cite, dfn, em, var {
font-style:normal;
}

code, kbd, pre, samp {
font-family:couriernew, courier, monospace;
}

small{
font-size:12px;
}

ul, ol {
list-style:none;
}

a {
text-decoration:none;
}

a:hover {
text-decoration:underline;
}

sup {
vertical-align:text-top;
}

sub{
vertical-align:text-bottom;
}

legend {
color:#000;
}

fieldset, img {
border:0;
}

button, input, select, textarea {
font-size:100%;
}

table {
border-collapse:collapse;
border-spacing:0;
}
```

10，IE盒子模型 、W3C盒子模型

- W3C盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border)；
  - `box-sizing: content-box`
  - width = content width;
- IE盒子模型： IE 的content 部分把 border 和 padding 计算了进去;
  - `box-sizing: border-box`
  - width = border + padding + content width

11，请解释一下CSS3的Flexbox（弹性盒布局模型）,以及适用场景

一个用于页面布局的全新CSS3功能，Flexbox可以把列表放在同一个方向（从上到下排列，从左到右），并让列表能延伸到占用可用的空间。

较为复杂的布局还可以通过嵌套一个伸缩容器（flex container）来实现。

采用Flex布局的元素，称为Flex容器（flex container），简称"容器"。

它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。

常规布局是基于块和内联流方向，而Flex布局是基于flex-flow流可以很方便的用来做局中，能对不同屏幕大小自适应。

在布局上有了比以前更加灵活的空间

12，display:inline-block 什么时候不会显示间隙?

- 移除空格
- 使用`margin`负值
- 使用`font-size:0`
- `letter-spacing`
- `word-spacing`

13，行内元素float:left后是否变为块级元素?

行内元素设置成浮动之后变得更加像是`inline-block`（行内块级元素，设置 成这个属性的元素会同时拥有行内和块级的特性，最明显的不同是它的默认宽度不是 100% ），这时候给行内元素设置`padding-top`和`padding-bottom`或者`width 、 height`都是有效果的

14，如果需要手动写动画，你认为最小时间间隔是多久，为什么?

多数显示器默认频率是 60Hz ，即 1 秒刷新 60 次，所以理论上最小间隔为 1/60*1000ms = 16.7ms

15，display:inline-block 什么时候会显示间隙?

移除空格、使用margin负值、使用font-size:0、letter-spacing、word-spacing、浮动

16，CSS权重，计算规则

- `! important` > 内联样式 > ID > 类属性、属性选择器或者伪类选择器 > 标签选择器

`CSS REFACTORING`中提到了算法的过程，取决于A、B、C、D的值，abcd是什么呢?

符号	计算方式
A (内联样式)	有内联样式时 A=1 否则0
B（ID选择器出现次数）	有两层ID选择器 B=2
C （类选择器出现的次数）	类选择器1个， 属性选择器1个，伪类选择器2个 C=4
D （标签、伪类选择器出现次数）	标签选择器2个，伪类选择器1个 D=3

```
li                                  /* (0, 0, 0, 1) */
ul li                               /* (0, 0, 0, 2) */
ul ol+li                            /* (0, 0, 0, 3) */
ul ol+li                            /* (0, 0, 0, 3) */
h1 + *[REL=up]                      /* (0, 0, 1, 1) */
ul ol li.red                        /* (0, 0, 1, 3) */
li.red.level                        /* (0, 0, 2, 1) */
a1.a2.a3.a4.a5.a6.a7.a8.a9.a10.a11  /* (0, 0, 11,0) */
#x34y                               /* (0, 1, 0, 0) */
li:first-child h2 .title            /* (0, 0, 2, 2) */
#nav .selected > a:hover            /* (0, 1, 2, 1) */
html body #nav .selected > a:hover  /* (0, 1, 2, 3) */
```

- 最终从A开始逐层比较，A => B =>C =>D 哪个大优先哪个样式生效，否则后面覆盖前面样式，这也是为什么有的嵌套多层样式可以实现覆盖的原因。
- 样式名称也有就近原则，作用在当前标签的能覆盖继承来的样式
- 最终将这几个条件合并起来就是css的权重问题和计算规则

17，Sass、LESS是什么？大家为什么要使用他们?

他们是CSS预处理器。他是CSS上的一种抽象层。他们是一种特殊的语法/语言编译成CSS。

例如Less是一种动态样式语言. 将CSS赋予了动态语言的特性，如变量，继承，运算， 函数. LESS 既可以在客户端上运行 (支持IE 6+, Webkit, Firefox)，也可一在服务端运行 (借助 Node.js)。

为什么要使用它们?

结构清晰，便于扩展。

可以方便地屏蔽浏览器私有语法差异。这个不用多说，封装对浏览器语法差异的重复处理，减少无意义的机械劳动。

可以轻松实现多重继承。

完全兼容 CSS 代码，可以方便地应用到老项目中。LESS 只是在 CSS 语法上做了扩展，所以老的 CSS 代码也可以与 LESS 代码一同编译。

18，stylus，sass，less区别

- 均具有“变量”、“混合”、“嵌套”、“继承”、“颜色混合”五大基本特性
- `Sass` 和 `LESS` 语法较为严谨， `LESS` 要求一定要使用大括号“{}”， `Sass` 和 `Stylus` 可以通过缩进表示层次与嵌套关系
- `Sass` 无全局变量的概念， `LESS` 和 `Stylus` 有类似于其它语言的作用域概念
- `Sass` 是基于 `Ruby` 语言的，而 `LESS` 和 `Stylus` 可以基于 `NodeJS NPM` 下载相应库后进行编译；这也是为什么安装Sass的时候有时候会报错，需要安装python脚本

优点：就不用我多说了，谁用谁知道，真香。

19，rgba() 和 opacity 的透明效果有什么不同?

- `rgba()` 和 `opacity` 都能实现透明效果，但最大的不同是 `opacity` 作用于元素，以及元素内的所有内容的透明度，
- 而 `rgba()` 只作用于元素的颜色或其背景色。（设置 rgba 透明的元素的子元素不会继承透明效果！）

20，px和em的区别

px和em都是长度单位，区别是：

px值固定，容易计算。

em值不固定，是相对单位，其相对应父级元素的字体大小会调整

21，使用css实现一个持续的动画效果

```
animation:mymove 5s infinite;
@keyframes mymove {
from {top:0px;}
to {top:200px;}
}
```

22，，重排和重绘

部分渲染树（或者整个渲染树）需要重新分析并且节点尺寸需要重新计算。这被称为重排。注意这里至少会有一次重排-初始化页面布局。

由于节点的几何属性发生改变或者由于样式发生改变，例如改变元素背景色时，屏幕上的部分内容需要更新。这样的更新被称为重绘。

23，什么情况会触发重排和重绘?

1）添加、删除、更新 DOM 节点

2）通过 display: none 隐藏一个 DOM 节点-触发重排和重绘

3）通过 visibility: hidden 隐藏一个 DOM 节点-只触发重绘，因为没有几何变化

4）移动或者给页面中的 DOM 节点添加动画

5）添加一个样式表，调整样式属性

6）用户行为，例如调整窗口大小，改变字号，或者滚动

JavaScript

1，说几条JavaScript的基本规范

- 不要在同一行声明多个变量
- 请是用 ===/!== 来比较 `true/false` 或者数值
- 使用对象字面量替代 `new Array` 这种形式
- 不要使用全局函数
- `Switch` 语句必须带有 `default` 分支
- `If` 语句必须使用大括号
- `for-in` 循环中的变量 应该使用 `let` 关键字明确限定作用域，从而避免作用域污染

2，闭包

- 闭包就是能够读取其他函数内部变量的函数
- 闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个
- 函数内创建另一个函数，通过另一个函数访问这个函数的局部变量,利用闭包可以突破作用链域
- 闭包的特性：
    - 函数内再嵌套函数
    - 内部函数可以引用外层的参数和变量
    - 参数和变量不会被垃圾回收机制回收
- 优点：能够实现封装和缓存等
- 缺点：消耗内存、使用不当会内存溢出，
- 解决方法：在退出函数之前，将不使用的局部变量全部删除

3，变量对象

变量对象，是执行上下文中的一部分，可以抽象为一种 数据作用域，其实也可以理解为就是一个简单的对象，它存储着该执行上下文中的所有 变量和函数声明(不包含函数表达式)。

活动对象 (AO): 当变量对象所处的上下文为 active EC 时，称为活动对象。

4，作用域

执行上下文中还包含作用域链。理解作用域之前，先介绍下作用域。作用域其实可理解为该上下文中声明的 变量和声明的作用范围。可分为 块级作用域 和 函数作用域

特性:

声明提前: 一个声明在函数体内都是可见的, 函数优先于变量

非匿名自执行函数，函数变量为 只读 状态，无法修改

```
let foo = function() { console.log(1) };
(function foo() {
    foo = 10  // 由于foo在函数中只为可读，因此赋值无效
    console.log(foo)
}())
// 结果打印：  ƒ foo() { foo = 10 ; console.log(foo) }
```

5，说说你对作用域链的理解

- 作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到 `window` 对象即被终止，作用域链向下访问变量是不被允许的。
- 简单的说，作用域就是变量与函数的可访问范围，即作用域控制着变量与函数的可见性和生命周期

6，JavaScript原型，原型链？有什么特点?

- 每个对象都会在其内部初始化一个属性，就是 `prototype` (原型)，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去 `prototype` 里找这个属性，这个 `prototype` 又会有自己的 `prototype` ，于是就这样一直找下去，也就是我们平时所说的原型链的概念
- 关系： `instance.constructor.prototype = instance._*proto*_`
- 特点： `JavaScript` 对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变当我们需要一个属性的时， `Javascript` 引擎会先看当前对象中是否有这个属性， 如果没有的,就会查找他的 `Prototype` 对象是否有这个属性，如此递推下去，一直检索到 `Object` 内建对象

7，什么是事件委托

- 事件代理 `（ Event Delegation ）` ，又称之为事件委托。是 `JavaScript` 中常用的绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是 `DOM` 元素的事件冒泡。使用事件代理的好处是可以提高性能
- 可以大量节省内存占用，减少事件注册，比如在 `table` 上代理所有 `td 的 click` 事件就非常棒
- 可以实现当新增子对象时无需再次对其绑定

8，类的创建

类的创建（es5）：new一个function，在这个function的prototype里面增加属性和方法。

下面来创建一个Animal类：

```
// 定义一个动物类
function Animal (name) {
// 属性
this.name = name || 'Animal';
// 实例方法
this.sleep = function(){
console.log(this.name + '正在睡觉! ');
}
}

// 原型方法
Animal.prototype.eat = function(food) {
console.log(this.name + '正在吃: ' + food);
};
```

这样就生成了一个Animal类，实力化生成对象后，有方法和属性。

9，如何实现继承?

- 构造继承
- 原型继承
- 实例继承
- 拷贝继承
- 原型 `prototype` 机制或 `apply 和 call` 方法去实现较简单，建议使用构造函数与原型混合方式

```
function Parent(){
this.name = 'wang';
}
function Child(){
 this.age = 28;
}
Child.prototype = new Parent();//继承了Parent，通过原型
var demo = new Child();
alert(demo.age);
alert(demo.name);//得到被继承的属性
```

10，谈谈This对象的理解

- `this` 总是指向函数的直接调用者（而非间接调用者）
- 如果有 `new` 关键字， `this` 指向 `new` 出来的那个对象
- 在事件中， `this` 指向触发这个事件的对象，特殊的是， `IE` 中的 `attachEvent 中的this` 总是指向全局对象 `Window`

11，事件模型

> `W3C` 中定义事件的发生经历三个阶段：捕获阶段（`capturing`）、目标阶段 （`targetin`）、冒泡阶段（`bubbling`）

- 冒泡型事件：当你使用事件冒泡时，子级元素先触发，父级元素后触发
- 捕获型事件：当你使用事件捕获时，父级元素先触发，子级元素后触发
- `DOM` 事件流：同时支持两种事件模型：捕获型事件和冒泡型事件

- 阻止冒泡：在 `W3c` 中，使用 `stopPropagation()` 方法；在IE下设置 `cancelBubble =true`
- 阻止捕获：阻止事件的默认行为，例如 `click - a` 后的跳转。在 `W3c` 中，使用 `preventDefault()` 方法，在 IE 下设置 `window.event.returnValue = false`

12，new操作符具体干了什么呢?

- 创建一个空对象，并且 `this` 变量引用该对象，同时还继承了该函数的原型
- 属性和方法被加入到 `this` 引用的对象中
- 新创建的对象由 `this` 所引用，并且最后隐式的返回 `this`

13，Ajax原理

- `Ajax` 的原理简单来说是在用户和服务器之间加了一个中间层(AJAX 引擎)，通过 `XmlHttpRequest` 对象来向服务器发异步请求，从服务器获得数据，然后用 `javascript` 来操作 `DOM` 而更新页面。使用户操作与服务器响应异步化。这其中最关键的一步就是从服务器获得请求数据
- `Ajax` 的过程只涉及 `JavaScript 、 XMLHttpRequest 和 DOM 。 XMLHttpRequest` 是 `ajax` 的核心机制

14，Ajax解决浏览器缓存问题

在ajax发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。

在ajax发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。

在URL后面加上一个随机数： "fresh=" + Math.random()。

在URL后面加上时间搓："nowtime=" + new Date().getTime()。

如果是使用jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有ajax都会执行这条语句就是不需要保存缓存记录。

15，如何解决跨域问题?

> 首先了解下浏览器的同源策略 同源策略 `/SOP（Same origin policy）` 是一种约定，由Netscape公司1995年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到 `XSS 、 CSFR` 等攻击。所谓同源是指 `"协议+域名+端口"` 三者相同，即便两个不同的域名指向同一个ip地址，也非同源

- 通过 `jsonp` 跨域

```
var script = document.createElement('script');
script.type = 'text/javascript';
// 传参并指定回调执行函数为onBack
script.src = 'http://www.....:8080/login?user=admin&callback=onBack';
document.head.appendChild(script);
// 回调执行函数
function onBack(res) {
 alert(JSON.stringify(res));
}
```

- `document.domain + iframe`跨域

```
//父窗口：(http://www.domain.com/a.html)
<iframe id="iframe" src="http://child.domain.com/b.html"></iframe>
<script>
 document.domain = 'domain.com';
 var user = 'admin';
</script>

//子窗口：(http://child.domain.com/b.html)
document.domain = 'domain.com';
// 获取父窗口中变量
alert('get js data from parent ---> ' + window.parent.user);
```

- `nginx`代理跨域
- `nodejs`中间件代理跨域
- 后端在头部信息里面设置安全域名

16，说说你对AMD和Commonjs的理解

- `CommonJS`是服务器端模块的规范，`Node.js`采用了这个规范。`CommonJS`规范加载模

块是同步的，也就是说，只有加载完成，才能执行后面的操作。`AMD`规范则是非同步加载 模块，允许指定回调函数

- `AMD`推荐的风格通过返回一个对象做为模块对象，`CommonJS`的风格通过对

`module.exports 或 exports`的属性赋值来达到暴露模块对象的目的

17，js的7种基本数据类型

`Undefined 、 Null 、Boolean 、Number 、String 、Bigint 、Symbol`

18，介绍js有哪些内置对象

- `Object`是`JavaScript`中所有对象的父对象
- 数据封装类对象：`Object 、 Array 、 Boolean 、 Number 和 String`
- 其他对象：`Function 、 Arguments 、 Math 、 Date 、 RegExp 、 Error`

19，JS有哪些方法定义对象

- 对象字面量：`var obj = {};`
- 构造函数：`var obj = new Object();`
- Object.create(): `var obj = Object.create(Object.prototype);`

20，你觉得jQuery源码有哪些写的好的地方

- `jquery` 源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入 `window` 对象参数，可以使 `window` 对象作为局部变量使用，好处是当 `jquery` 中访问 `window` 对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问 `window` 对象。同样，传入 `undefined` 参数，可以缩短查找 `undefined` 时的作用域链
- `jquery` 将一些原型属性和方法封装在了 `jquery.prototype` 中，为了缩短名称，又赋值给了 `jquery.fn`，这是很形象的写法
- 有一些数组或对象的方法经常能使用到， `jQuery` 将其保存为局部变量以提高访问速度
- `jquery` 实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率

21，null，undefined 的区别

- `undefined` 表示不存在这个值。
- `undefined` :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。尝试读取时会返回 `undefined`
- 例如变量被声明了，但没有赋值时，就等于 `undefined`
- `null` 表示一个对象被定义了，值为“空值”
- `null` : 是一个对象(空对象, 没有任何属性和方法)
- 例如作为函数的参数，表示该函数的参数不是对象；
- 在验证 `null` 时，一定要使用 `===` ，因为 `==` 无法分别 `null 和 undefined`

22，谈谈你对ES6的理解

- 新增模板字符串（为 `JavaScript` 提供了简单的字符串插值功能）
- 箭头函数
- `for-of` （用来遍历数据—例如数组中的值。）
- `arguments` 对象可被不定参数和默认参数完美代替。
- `ES6` 将 `promise` 对象纳入规范，提供了原生的 `Promise` 对象。
- 增加了 `let 和 const` 命令，用来声明变量。
- 还有就是引入 `module` 模块的概念

23，面向对象编程思想

- 基本思想是使用对象，类，继承，封装等基本概念来进行程序设计
- 易维护
- 易扩展
- 开发工作的重用性、继承性高，降低重复工作量。
- 缩短了开发周期

24，如何通过JS判断一个数组

- `instanceof` 运算符是用来测试一个对象是否在其原型链原型构造函数的属性

```
var arr = [];
arr instanceof Array; // true
```

- `isArray`

```
Array.isArray([]) //true
Array.isArray(1) //false
```

- `constructor` 属性返回对创建此对象的数组函数的引用，就是返回对象相对应的构造函数

```
var arr = [];
arr.constructor == Array; //true
```

- `Object.prototype`

```
Object.prototype.toString.call([]) == '[object Array]'
// 写个方法
var isType = function (obj) {
 return Object.prototype.toString.call(obj).slice(8,-1);
 //return Object.prototype.toString.apply([obj]).slice(8,-1);
}
isType([])  //Array
```

25，异步编程的实现方式

- 回调函数
  - 优点：简单、容易理解
  - 缺点：不利于维护，代码耦合高
- 事件监听(采用时间驱动模式，取决于某个事件是否发生)
  - 优点：容易理解，可以绑定多个事件，每个事件可以指定多个回调函数
  - 缺点：事件驱动型，流程不够清晰
- 发布/订阅(观察者模式)
  - 类似于事件监听，但是可以通过‘消息中心’，了解现在有多少发布者，多少订阅者
- Promise 对象
  - 优点：可以利用`then`方法，进行链式写法；可以书写错误时的回调函数；
  - 缺点：编写和理解，相对比较难
- Generator 函数
  - 优点：函数体内外的数据交换、错误处理机制
  - 缺点：流程管理不方便
- async 函数
  - 优点：内置执行器、更好的语义、更广的适用性、返回的是`Promise`、结构清晰。
  - 缺点：错误处理机制

26，对原生Javascript了解方向

数据类型、运算、对象、`Function`、继承、闭包、作用域、原型链、事件、`RegExp`、`JSON`、`Ajax`、`DOM`、`BOM`、内存泄漏、跨域、异步装载、模板引擎、`前端MVC`、路由、模块化、`Canvas`、`ECMAScript`

27，sort 快速打乱数组

```
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(()=> Math.random() - 0.5)
//利用sort return 大于等于0不交换位置，小于0交换位置
// [5, 8, 4, 3, 2, 9, 10, 6, 1, 7]
```

28，数组去重操作

- `ES6 Set`
- `for` 循环 `indexOf`
- `for` 循环 `includes`
- `sort`

29，JS 原生拖拽节点

- 给需要拖拽的节点绑定 `mousedown ，mousemove ，mouseup` 事件
- `mousedown` 事件触发后，开始拖拽
- `mousemove` 时，需要通过 `event.clientX 和 clientY` 获取拖拽位置，并实时更新位置
- `mouseup` 时，拖拽结束
- 需要注意浏览器边界值，设置拖拽范围

30，深拷贝、浅拷贝

- 所有的基础数据类型的赋值操作都是深拷贝
- 通常利用上面这点，来对引用数据类型做递归深拷贝
- 浅拷贝：`Object.assign` 或者 扩展运算符
- 深拷贝：JSON.parse(JSON.stringify(object)) 深层递归
    - 局限性：会忽略 undefined，不能序列化函数，不能解决循环引用的对象

31，节流防抖

- 节流：每隔一段时间执行一次，通常用在高频率触发的地方，降低频率。--如：鼠标滑动 拖拽
- 防抖：一段时间内连续触发，不执行，直到超出限定时间执行最后一次。--如：`input` 模糊搜索

32，变量提升

当执行 `JS` 代码时，会生成执行环境，只要代码不是写在函数中的，就是在全局执行环境中，函数中的代码会产生函数执行环境，只此两种执行环境

```
js b() // call b console.log(a) // undefined var a = 'Hello world' function b()
{
console.log('call b')
}
```

变量提升 这是因为函数和变量提升的原因。通常提升的解释是说将声明的代码移动到了顶部，这其实没有什么错误，便于大家理解。

但是更准确的解释应该是：在生成执行环境时，会有两个阶段。

第一个阶段是创建的阶段，`JS`解释器会找出需要提升的变量和函数，并且给他们提前在内存中开辟好空间，函数的话会将整个函数存入内存中，变量只声明并且赋值为`undefined`，所以在第二个阶段，也就是代码执行阶段，我们可以直接提前使用

```
b() // call b second
function b() {
 console.log('call b fist')
}
function b() {
 console.log('call b second')
}
var b = 'Hello world'
```

33，JS中的垃圾回收机制

必要性：由于字符串、对象和数组没有固定大小，所有当他们的大小已知时，才能对他们进行动态的存储分配。

JavaScript程序每次创建字符串、数组或对象时，解释器都必须分配内存来存储那个实体。只要像这样动态地分配了内存，最终都要释放这些内存以便他们能够被再用，否则，JavaScript的解释器将会消耗完系统中所有可用的内存，造成系统崩溃。

这段话解释了为什么需要系统需要垃圾回收，JS不像C/C++，他有自己的一套垃圾回收机制（Garbage Collection）。

JavaScript的解释器可以检测到何时程序不再使用一个对象了，当他确定了一个对象是无用的时候，他就知道不再需要这个对象，可以把它所占用的内存释放掉了。

例如：

```
var a="hello world";
var b="world";
var a=b;
//这时，会释放掉"hello world"，释放内存以便再引用
```

垃圾回收的方法：标记清除、计数引用。

标记清除

这是最常见的垃圾回收方式，当变量进入环境时，就标记这个变量为"进入环境",从逻辑上讲，永远不能释放进入环境的变量所占的内存，永远不能释放进入环境变量所占用的内存，只要执行流程进入相应的环境，就可能用到他们。当离开环境时，就标记为离开环境。

垃圾回收器在运行的时候会给存储在内存中的变量都加上标记（所有都加），然后去掉环境变量中的变量，以及被环境变量中的变量所引用的变量（条件性去除标记），删除所有被标记的变量，删除的变量无法在环境变量中被访问所以会被删除，最后垃圾回收器，完成了内存的清除工作，并回收他们所占用的内存。

引用计数法

另一种不太常见的方法就是引用计数法，引用计数法的意思就是每个值没引用的次数，当声明了一个变量，并用一个引用类型的值赋值给改变量，则这个值的引用次数为1,;

相反的，如果包含了对这个值引用的变量又取得了另外一个值，则原先的引用值引用次数就减1，当这个值的引用次数为0的时候，说明没有办法再访问这个值了，因此就把所占的内存给回收进来，这样垃圾收集器再次运行的时候，就会释放引用次数为0的这些值。

用引用计数法会存在内存泄露，下面来看原因：

```
function problem() {
var objA = new Object();
var objB = new Object();
objA.someOtherObject = objB;
objB.anotherObject = objA;
}
```

在这个例子里面，objA和objB通过各自的属性相互引用，这样的话，两个对象的引用次数都为2，在采用引用计数的策略中，由于函数执行之后，这两个对象都离开了作用域，函数执行完成之后，因为计数不为0，这样的相互引用如果大量存在就会导致内存泄露。

特别是在DOM对象中，也容易存在这种问题：

```
var element=document.getElementById（’‘）；
var myObj=new Object();
myObj.element=element;
element.someObject=myObj;
```

这样就不会有垃圾回收的过程。

34，如何理解前端模块化

前端模块化就是复杂的文件编程一个一个独立的模块，比如js文件等等，分成独立的模块有利于重用（复用性）和维护（版本迭代），这样会引来模块之间相互依赖的问题，所以有了commonJS规范，AMD，CMD规范等等，以及用于js打包（编译等处理）的工具webpack

35，js单线程

- 单线程 - 只有一个线程，只能做一件事
- 原因 - 避免 DOM 渲染的冲突
  - 浏览器需要渲染 `DOM`
  - `JS` 可以修改 `DOM` 结构
  - `JS` 执行的时候，浏览器 `DOM` 渲染会暂停
  - 两段 `JS` 也不能同时执行（都修改 `DOM` 就冲突了）
  - `webworker` 支持多线程，但是不能访问 `DOM`
- 解决方案 - 异步

36，说说event loop

> 首先，`js`是单线程的，主要的任务是处理用户的交互，而用户的交互无非就 是响应`DOM`的增删改，使用事件队列的形式，一次事件循环只处理一个事件 响应，使得脚本执行相对连续，所以有了事件队列，用来储存待执行的事件， 那么事件队列的事件从哪里被`push`进来的呢。那就是另外一个线程叫事件触 发线程做的事情了，他的作用主要是在定时触发器线程、异步`HTTP`请求线程 满足特定条件下的回调函数`push`到事件队列中，等待`js`引擎空闲的时候去 执行，当然`js`引擎执行过程中有优先级之分，首先`js`引擎在一次事件循环中， 会先执行js线程的主任务，然后会去查找是否有微任务`microtask（promise）`，如果有那就优先执行微任务，如果没有，在去查找 宏任务`macrotask（setTimeout、setInterval）`进行执行

37，描述下this

> `this`，函数执行的上下文，可以通过`apply ， call ， bind`改变`this`的指向。对于匿名函数或者直接调用的函数来说，this指向全局上下文（浏览 器为`window，NodeJS为 global`），剩下的函数调用，那就是谁调用它，`this`就指向谁。当然还有`es6`的箭头函数，箭头函数的指向取决于该箭头函 数声明的位置，在哪里声明，`this`就指向哪里

38，ajax、axios、fetch区别

ajax：

- 本身是针对`MVC`的编程,不符合现在前端`MVVM`的浪潮
- 基于原生的`XHR`开发，`XHR`本身的架构不清晰，已经有了`fetch`的替代方案
- `JQuery`整个项目太大，单纯使用`ajax`却要引入整个`JQuery`非常的不合理（采取个性化打包的方案又不能享受`CDN`服务）

axios：

- 从浏览器中创建`XMLHttpRequest`
- 从`node.js`发出`http`请求
- 支持`Promise API`
- 拦截请求和响应
- 转换请求和响应数据
- 取消请求
- 自动转换`JSON`数据
- 客户端支持防止`CSRF/XSRF`

fetch：

- 只对网络请求报错，对 400 ， 500 都当做成功的请求，需要封装去处理
- 这里对于`cookie`的处理比较特殊，不同浏览器对`credentials`的默认值不一样，也就使得默认情况下`cookie`变的不可控。
- 本身无自带`abort`，无法超时控制，可以使用`AbortController`解决取消请求问题。
- 没有办法原生监测请求的进度，而`XHR`可以

浏览器

1，浏览器存储的方式有哪些

特性	cookie	localStorage	sessionStorage	indexedDB
数据生命周期	一般由服务器生成，可以设置过期时间	除非被清理，否则一直存在	页面关闭就清理	除非被清理，否则一直存在
数据存储大小	4K	5M	5M	无限
与服务端通信	每次都会携带在 header，中，对于请求性能影响	不参与	不参与	不参与

补充：cookie 原本并不是用来储存的，而是用来与服务端通信的，需要存取请自行封装 api。
而 localStorage 则自带 getItem 和 setItem 方法，使用很方便。

localStorage 注意点：

1. localStorage 只能存字符串，存取 JSON 数据需配合 JSON.stringify() 和 JSON.parse()
2. 遇上禁用 setItem 的浏览器，需要使用 try...catch 捕获异常

2，浏览器内核的理解

- 主要分两个部分：`渲染引擎`、`js引擎`
- `渲染引擎：`负责取得网页的内容（html css img ...），以及计算网页的显示方式，然后会输出至显示器或者打印机。浏览器的内核不同对于网页的语法解释也不同，所以渲染的效果也不一样
- `js引擎：` 解析和执行javascript 来实现网页的动态效果
- 最开始渲染引擎和js引擎并没有区分的很明确，后来js引擎越来越独立，内核就倾向于只值渲染引擎
- `IE : trident` 内核
- `Firefox ： gecko` 内核
- `Safari : webkit` 内核
- `Opera` :以前是 `presto` 内核， `Opera` 现已改用`Google - Chrome` 的 `Blink` 内核
- `Chrome:Blink`(基于 `webkit` ， `Google与Opera Software` 共同开发)

3，HTTP 的请求方式场景

- `Get` 方法：获取数据通常(查看数据)-查看
- `POST` 方法：向服务器提交数据通常(创建数据)-create
- `PUT` 方法：向服务器提交数据通常(更新数据)-update，与 `POST` 方法很像，也是提交数据，但 `PUT` 制定了资源在服务器上的位置，常用在修改数据
- `HEAD` 方法：只请求页面的首部信息
- `DELETE` 方法：删除服务器上的资源
- `OPTIONS` 方法：用于获取当前 `URL` 支持的请求方式
- `TRACE` 方法：用于激活一个远程的应用层请求消息回路
- `CONNECT` 方法：把请求链接转换到透明的 `TCP/IP` 的通道

4，HTTP状态码

- 1XX：信息状态码
  - `100 continue` 继续，一般在发送 post 请求时，已发送了 http header 之后服务端将返回此信息，表示确认，之后发送具体参数信息
- 2XX：成功状态码
  - `200 ok` 正常返回信息
  - `201 created` 请求成功并且服务器创建了新资源
  - `202 accepted` 服务器已经接收请求，但尚未处理
- 3XX：重定向
  - `301 move per` 请求的网页已经永久重定向
  - `302 found` 临时重定向
  - `303 see other` 临时冲重定向，且总是使用get请求新的url
  - `304 not modified` 自从上次请求后，请求的网页未修改过
- 4XX：客户端错误
  - `400 bad request` 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求
  - `401 unauthorized` 请求未授权
  - `403 forbidden` 禁止访问
- 404：not found 找不到如何与url匹配的资源
- 5XX：服务器错误
  - `500 internal server error` 最常见的服务器端的错误
  - `503 service unacailable` 服务器端暂时无法处理请求（可能是过载活维护）

5，从浏览器地址栏输入URL后发生了什么?

基础版本

- 的浏览器根据请求的 `URL` 交给 `DNS` 域名解析，找到真实 `IP`，向服务器发起请求；
- 服务器交给后台处理完成后返回数据，浏览器接收文件 `（ HTML、JS、CSS 、图象等）`；
- 浏览器对加载到的资源 `（ HTML、JS、CSS 等）` 进行语法解析，建立相应的内部数据结构 `（如 HTML 的 DOM ）`；
- 载入解析到的资源文件，渲染页面，完成。

详细版

- 从浏览器接收 `url` 到开启网络请求线程（这一部分可以展开浏览器的机制以及进程与线程之间的关系）
- 开启网络线程到发出一个完整的 `HTTP` 请求（这一部分涉及到dns查询， `TCP/IP` 请求，五层因特网协议栈等知识）
- 从服务器接收到请求到对应后台接收到请求（这一部分可能涉及到负载均衡，安全拦截以及后台内部的处理等等）
- 后台和前台的 `HTTP` 交互（这一部分包括 `HTTP` 头部、响应码、报文结构、 `cookie` 等知识，可以提下静态资源的 `cookie` 优化，以及编码解码，如 `gzip` 压缩等）
- 单独拎出来的缓存问题， `HTTP` 的缓存（这部分包括 `http缓存头部`， `ETag` ， `catchcontrol` 等）
- 浏览器接收到 `HTTP` 数据包后的解析流程（解析 `html` -词法分析然后解析成 `dom` 树、解析 `css` 生成 `css` 规则树、合并成 `render` 树，然后 `layout 、 painting` 渲染、复合图层的合成、 `GPU` 绘

制、外链资源的处理、 `loaded 和 DOMContentLoaded` 等）
- `CSS` 的可视化格式模型（元素的渲染规则，如包含块，控制框， `BFC`， IFC 等概念）
- `JS` 引擎解析过程（`JS` 的解释阶段，预处理阶段，执行阶段生成执行上下文，`VO`，作用域链、回收机制等等）
- 其它（可以拓展不同的知识模块，如跨域，`web` 安全， `hybrid` 模式等等内容）

详细升级版

- 在浏览器地址栏输入`URL`
- 浏览器查看缓存，如果请求资源在缓存中并且新鲜，跳转到转码步骤
  - 如果资源未缓存，发起新请求
  - 如果已缓存，检验是否足够新鲜，足够新鲜直接提供给客户端，否则与服务器进行验证。
  - 检验新鲜通常有两个HTTP头进行控制 Expires 和 Cache-Control：
    - 2.3.1 `HTTP1.0` 提供 `Expires`，值为一个绝对时间表示缓存新鲜日期
    - 2.3.2 `HTTP1.1` 增加了 `Cache-Control: max-age=`,值为以秒为单位的最大新鲜时间
- 浏览器解析 `URL` 获取协议，主机，端口， `path`
- 浏览器组装一个 `HTTP（GET）` 请求报文
- 浏览器获取主机ip地址，过程如下：
  - 浏览器缓存
  - 本机缓存
  - hosts文件
  - 路由器缓存
  - ISP DNS缓存
  - DNS递归查询（可能存在负载均衡导致每次IP不一致）
- 打开一个socket与目标IP地址，端口建立TCP链接，三次握手

  如下：
  - 客户端发送一个 `TCP的SYN=1，Seq=X` 的包到服务器端口
  - 服务器发回 `SYN=1，ACK=X+1,Seq=Y` 的相应包
  - 客户端发送 `ACK=Y+1，Seq=Z`
- `TCP` 链接建立后发送 `HTTP` 请求
- 服务器接收请求后解析，将请求转发到服务器程序，如虚拟主机使用 `HTTP Host` 头部判断请求的服务程序
- 服务器检测 `HTTP` 请求头是否包含缓存验证信息，如果验证缓存新鲜，返回304等对应状态
- 出合理程序读取完整请求并准备 `HTTP` 相应，可能需要查询数据库等操作
- 服务器将相应报文通过 `TCP` 链接发送回浏览器
- 浏览器接收HTTP相应，然后根据情况选择关闭TCP链接或者保留重用，关闭TCP链接的四次握手如下：
  - 主动方发送 `Fin=1,ACK=z,Seq=x` 报文
  - 被动方发送 `ACK=X+1,Seq=Y` 报文
  - 被动方发送 `Fin=1,ACK=X,Seq=Y` 报文
  - 主动方发送 `ACK=Y,Seq=x` 报文
- 浏览器检查相应状态码
- 如果资源可缓存，进行缓存
- 对相应进行解码
- 根据资源类型决定如何处理

- 解析`HTML`文档，构建`DOM`树，下载资源，构建`CSSOM`树，执行js脚本，这些操作每月严格的先后顺序
- 构建DOM树：
  - Tokenizing：根据HTML规范将字符流解析为标记
  - Lexing：词法分析将标记转换为对象并定义属性和规则
  - DOM construction：根据HTML标记关系将对象组成DOM树
- 解析过程中遇到图片、样式表、js文件，启动下载
- 构建CSSOM树：
  - `Tokenizing`：字符流转换为标记流
  - `Node`：根据标记创建节点
  - `CSSOM`：节点创建CSSOM树
- 根据`DOM树和CSSOM树`构建渲染树
  - 从`DOM树`的根节点遍历所有可见节点，不可见节点包括：1）`script ，meta`这样本身不可见的标签。2)被css隐藏的节点，如 display: none
  - 对每一个可见节点，找到恰当的`CSSOM`规则并应用
  - 发布可视节点的内容和计算样式
- js解析如下
  - 浏览器创建`Document对象`并解析`HTML`，将解析到的元素和文本节点添加到文档中，此时`document.readystate为loading`
  - HTML解析器遇到没有`async和defer的script时`，将他们添加到文档中，然后执行行内或外部脚本。这些脚本会同步执行，并且在脚本下载和执行时解析器会暂停。这样就可以用`document.write()`把文本插入到输入流中。同步脚本经常简单定义函数和注册事件处理程序，他们可以遍历和操作script和他们之前的文档内容
  - 当解析器遇到设置了`async属性的script`时，开始下载脚本并继续解析文档。脚本会在它下载完成后尽快执行，但是解析器不会停下来等它下载。异步脚本禁止使用`document.write()`，它们可以访问自己script和之前的文档元素
  - 当文档完成解析，`document.readState变成interactive`
  - 所有`defer脚本`会按照在文档出现的顺序执行，延迟脚本能访问完整文档树，禁止使用`document.write()`
  - 浏览器在`Document`对象上触发`DOMContentLoaded事件`
  - 此时文档完全解析完成，浏览器可能还在等待如图片等内容加载，等这些内容完成载入并且所有异步脚本完成载入和执行，`document.readState变为complete，window触发load事件`
- 显示页面（HTML解析过程中会逐步显示页面）

6，请你谈谈Cookie的优缺点

优点：极高的扩展性和可用性

1) 数据持久性。

2) 不需要任何服务器资源。 Cookie 存储在客户端并在发送后由服务器读取。

3) 可配置到期规则。 控制 cookie 的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的 cookie 。

4) 简单性。 基于文本的轻量结构。

5) 通过良好的编程，控制保存在 cookie 中的 session 对象的大小。

6) 通过加密和安全传输技术（ SSL ），减少 cookie 被破解的可能性。

7) 只在 cookie 中存放不敏感数据，即使被盗也不会有重大损失。

缺点：

1) Cookie 数量和长度的限制 。

数量：每个域的 cookie 总数有限。

a) IE6 或更低版本最多 20 个 cookie

b) IE7 和之后的版本最后可以有 50 个 cookie

c) Firefox 最多 50 个 cookie

d) chrome 和 Safari 没有做硬性限制

长度：每个 cookie 长度不超过 4KB （ 4096B ），否则会被截掉。

2) 潜在的安全风险 。 Cookie 可能被拦截、篡改。如果 cookie 被拦截，就有可能取得所有的 session 信息。

3) 用户配置为禁用 。有些用户禁用了浏览器或客户端设备接受 cookie 的能力，因此限制了这一功能。

4) 有些状态不可能保存在客户端 。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

7，cookies， sessionStorage 和 localStorage 的区别

- `cookie` 是网站为了标示用户身份而储存在用户本地终端上的数据（通常经过加密）
- `cookie` 数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递（优化点）
- `sessionStorage` 和 `localStorage` 不会自动把数据发给服务器，仅在本地保存
- 存储大小：
  - `cookie` 数据大小不能超过4k
  - `sessionStorage 和 localStorage` 虽然也有存储大小的限制，但比 cookie 大得多，可以达到5M或更大
- 有期时间：

`localStorage` 存储持久数据，浏览器关闭后数据不丢失除非主动删除数据

`sessionStorage` 数据在当前浏览器窗口关闭后自动删除

`cookie 设置的 cookie` 过期时间之前一直有效，即使窗口或浏览器关闭

8，浏览器缓存

> 浏览器缓存分为强缓存和协商缓存。当客户端请求某个资源时，获取缓存的流程如下

- 先根据这个资源的一些 `http header` 判断它是否命中强缓存，如果命中，则直接从本地获取缓存资源，不会发请求到服务器；
- 当强缓存没有命中时，客户端会发送请求到服务器，服务器通过另一些 `request header` 验证这个资源是否命中协商缓存，称为 `http` 再验证，如果命中，服务器将请求返回，但不返回资源，而是告诉客户端直接从缓存中获取，客户端收到返回后就会从缓存中获取资源；
- 强缓存和协商缓存共同之处在于，如果命中缓存，服务器都不会返回资源； 区别是，强缓存不对发送请求到服务器，但协商缓存会。

- 当协商缓存也没命中时，服务器就会将资源发送回客户端。
- 当 `ctrl+f5` 强制刷新网页时，直接从服务器加载，跳过强缓存和协商缓存；
- 当 `f5` 刷新网页时，跳过强缓存，但是会检查协商缓存；

9，浏览器渲染的步骤

1. HTML 解析出 DOM Tree
2. CSS 解析出 Style Rules
3. 两者关联生成 Render Tree
4. Layout（布局）根据 Render Tree 计算每个节点的信息
5. Painting 根据计算好的信息进行渲染整个页面

浏览器解析文档的过程中，如果遇到 script 标签，会立即解析脚本，停止解析文档（因为 JS 可能会改变 DOM 和 CSS,如果继续解析会造成浪费）。
如果是外部 script, 会等待脚本下载完成之后在继续解析文档。现在 script 标签增加了 defer 和 async 属性，脚本解析会将脚本中改变 DOM 和 css 的地方> 解析出来，追加到 DOM Tree 和 Style Rules 上

10，GET 和 POST 请求的区别

- GET 参数通过 url 传递，POST 放在 body 中。（http 协议规定，url 在请求头中，所以大小限制很小）
- GET 请求在 url 中传递的参数是有长度限制的，而 POST 没有。原因见上↑↑↑
- GET 在浏览器回退时是无害的，而 POST 会再次提交请求
- GET 请求会被浏览器主动 cache，而 POST 不会，除非手动设置
- GET 比 POST 更不安全，因为参数直接暴露在 url 中，所以不能用来传递敏感信息
- 对参数的数据类型，GET 只接受 ASCII字符，而 POST 没有限制
- GET 请求只能进行 url(x-www-form-urlencoded)编码，而 POST 支持多种编码方式
- **GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。**对于 GET 方式的请求，浏览器会把 http 的 header 和 data 一并发送出去，服务器响应200（返回数据）。而对于 POST，浏览器先发送 header，服务器响应100 continue，浏览器再发送 data，服务器响应200 ok（返回数据）

11，什么是reflow

浏览器为了重新渲染部分或整个页面，重新计算页面元素位置和几何结构的进程叫做 reflow . 通俗点说就是当开发人员定义好了样式后(也包括浏览器的默认样式),浏览器根据这些来计算并根据结果将元素放到它应该出现的位置上，这个过程叫做 reflow . 由于reflow是一种浏览器中的用户拦截操作，所以我们了解如何减少 reflow 次数，及DOM的层级，css 效率对 refolw 次数的影响是十分有必要的。 reflow (回流)是导致DOM脚本执行效率低的关键因素之一，页面上任何一个节点触发了 reflow，会导致它的子节点及祖先节点重新渲染。 简单解释一下 Reflow：当元素改变的时候，将会影响文档内容或结构，或元素位置，此过程称为 Reflow。

```
<body>
  <div class="hello">
    <h4>hello</h4>
    <p><strong>Name:</strong>BDing</p>
    <h5>male</h5>
    <ol>
      <li>coding</li>
      <li>loving</li>
    </ol>
  </div>
</body>
```

当p节点上发生reflow时，hello和body也会重新渲染，甚至h5和ol都会收到影响。

12，什么时候会导致reflow发生呢?

- 改变窗口大小
- 改变文字大小
- 添加/删除样式表
- 内容的改变，(用户在输入框中写入内容也会)
- 激活伪类，如:hover
- 操作class属性
- 脚本操作DOM
- 计算offsetWidth和offsetHeight
- 设置style属性

常见的重排元素			
width	height	padding	margin
display	border-width	border	top
position	font-size	float	text-align
overflow-y	font-weight	overflow	left
font-family	line-height	vertical-align	right
clear	white-space	bottom	min-height

13，减少reflow对性能的影响

- 不要一条一条地修改 DOM 的样式，预先定义好 class，然后修改 DOM 的 `className`
- 把 DOM 离线后修改，比如：先把 DOM 给 `display:none` (有一次 Reflow)，然后你修改100次，然后再把它显示出来
- 不要把 DOM 结点的属性值放在一个循环里当成循环里的变量
- 尽可能不要修改影响范围比较大的 DOM
- 为动画的元素使用绝对定位 `absolute / fixed`
- 不要使用 `table` 布局，可能很小的一个小改动会造成整个 table 的重新布局

- 尽可能限制reflow的影响范围，尽可能在低层级的DOM节点上，上述例子中，如果你要改变p的样式，class就不要加在div上，通过父元素去影响子元素不好。
- 避免设置大量的 style 属性，因为通过设置 style 属性改变结点样式的话，每一次设置都会触发一次 reflow，所以最好是使用 class 属性
- 实现元素的动画，它的position属性，最好是设为absoulte或fixed，这样不会影响其他元素的布局
- 动画实现的速度的选择。比如实现一个动画，以1个像素为单位移动这样最平滑，但是reflow就会过于频繁，大量消耗CPU资源，如果以3个像素为单位移动则会好很多。
- 不要使用 table 布局，因为table中某个元素旦触发了 reflow，那么整个 table 的元素都会触发 reflow 。那么在不得已使用 table 的场合，可以设置 table-layout:auto; 或者是 table-layout:fixed 这样可以让table一行一行的渲染，这种做法也是为了限制reflow的影响范围
- 如果CSS里面有计算表达式，每次都会重新计算一遍，出发一次reflow。

性能优化

1，SEO优化

- 合理的 title 、 description 、 keywords ： 搜索对着三项的权重逐个减小， `title` 值强调重点即可，重要关键词出现不要超过2次，而且要靠前，不同页面 title 要有所不同； `description` 把页面内容高度概括，长度合适，不可过分堆砌关键词，不同页面 `description` 有所不同； `keywords` 列举出重要关键词即可
- 语义化的 `HTML` 代码，符合W3C规范：语义化代码让搜索引擎容易理解网页
- 重要内容 `HTML` 代码放在最前：搜索引擎抓取 `HTML` 顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取
- 重要内容不要用 `js` 输出：爬虫不会执行js获取内容
- 少用 `iframe` ：搜索引擎不会抓取 `iframe` 中的内容
- 非装饰性图片必须加 `alt`
- 提高网站速度：网站速度是搜索引擎排序的一个重要指标

2，server优化

- 减少 `HTTP` 请求，合并文件、雪碧图
- 减少 `DNS` 查询，使用缓存
- 减少 `Dom` 元素的数量
- 使用 `CDN`
- 配置 `ETag` ,http缓存的手段
- 对组件使用 `Gzip` 压缩
- 减少 `cookie` 的大小

3，css优化

- 将样式表放在页面顶部
- 使用 `less scss` 表达式
- 使用 `link` 不适用 `@import` 引入样式
- 压缩 `css`
- 禁止使用 `gif` 图片实现 `loading` 效果（降低 CPU 消耗，提升渲染性能）

- 使用 `CSS3` 代码代替 `JS` 动画（尽可能避免重绘重排以及回流）
- 对于一些小图标，可以使用 `base64` 位编码，以减少网络请求。
- 页面头部的 `<style> <script>` 会阻塞页面；（因为 Renderer进程中 JS 线程和渲染线程是互斥的）
- 当需要设置的样式很多时设置 `className` 而不是直接操作 `style`

4，js方面

- 将脚本放到页面底部
- 将 `js` 外部引入
- 压缩 `js`
- 使用 `Eslint` 语法检测
- 减少 `Dom` 的操作
- 熟练使用设计模式
- 禁止使用 `iframe`（阻塞父文档 `onload` 事件）
- 页面中空的 `href 和 src` 会阻塞页面其他资源的加载
- 网页 `gzip ， CDN` 托管， `data` 缓存 ，图片服务器

5，webpack优化点

- 代码压缩插件 `UglifyJsPlugin`
- 服务器启用 `gzip` 压缩
- 按需加载资源文件 `require.ensure`
- 优化 `devtool` 中的 `source-map`
- 剥离 `css` 文件，单独打包
- 去除不必要插件，通常就是开发环境与生产环境用同一套配置文件导致
- 开发环境不做无意义的工作如提取 `css` 计算文件hash等
- 配置 `devtool`
- 优化构建时的搜索路径 指明需要构建目录及不需要构建目录

6，加载优化：

- 合并CSS、JavaScript
- 合并小图片、使用精灵图
- 缓存一切可缓存的资源
- 使用外链式引用CSS、JavaScript
- 压缩HTML、CSS、JavaScript
- 启用GZip
- 使用首屏加载、按需加载、滚屏加载
- 通过Media Query加载
- 增加Loading进度条
- 减少Cookie
- 避免重定向

- 异步加载第三方资源

7，页面渲染优化

- HTML 文档结构层次尽量少，最好不深于 6 层
- 脚本尽量放后边，避免组织页面加载
- 少量首屏样式可以放在便签内
- 样式结构层次尽量简单
- 脚本减少 DOM 操作，减少回流，尽量缓存访问 DOM 的样式信息
- 尽量减少 JS 修改样式，可以通过修改 class 名的方式解决
- 减少 DOM 查找，缓存 DOM 查找结果
- 动画在屏幕外或页面滚动时，尽量停止

8，图片优化

- 使用智图
- 使用（CSS3、SVG、IconFont）代替图片
- 使用Srcset
- webP优于JPG
- PNG8优于GIF
- 图片不宽于640

9，脚本优化

- 减少重绘和回流
- 缓存Dom选择与计算
- 尽量使用事件处理，避免批量绑定事件
- 尽量使用ID选择器
- 使用touchstart、touchend代替click

10，为什么利用多个域名来存储网站资源会更有效?

- `CDN` 缓存更方便
- 突破浏览器并发限制
- 节约 `cookie` 带宽
- 节约主域名的连接数，优化页面响应速度
- 防止不必要的安全问题

其他问题

- 自我介绍
- 面试完你还有什么问题要问的吗
- 你有什么爱好?
- 你最大的优点和缺点是什么?
- 你为什么会选择这个行业，职位?
- 你觉得你适合从事这个岗位吗?
- 你有什么职业规划?
- 你对工资有什么要求?
- 如何看待前端开发?
- 你的项目中技术难点是什么？遇到了什么问题？你是怎么解决的?
- 你认为哪个项目做得最好?
- 说下工作中你做过的一些性能优化处理
- 最近在看哪些前端方面的书?
- 平时是如何学习前端开发的?
- 你最有成就感的一件事
- 你为什么要离开前一家公司?
- 你对加班的看法
- 你希望通过这份工作获得什么?