1. **你所熟悉(用过)的前端开发工具有哪些(写在简历)?**

Editplus：EditPlus是一款的小巧但是功能强大的可处理文本、HTML和程序语言的Windows编辑器，你甚至可以通过设置用户工具将其作为C,Java,Php等等语言的一个简单的IDE。

Notepad++：Notepad++是Windows操作系统下的一套文本编辑器，有完整的中文化接口及支持多国语言编写的功能。

Adobe Dreamweaver：DW是集网页制作和管理网站于一身的所见即所得网页代码编辑器。利用对 HTML、CSS、JavaScript等内容的支持，设计师和程序员可以在几乎任何地方快速制作和进行网站建设。

**HBuilder**：HBuilder是DCloud（数字天堂）推出的一款支持HTML5的Web开发IDE。 HBuilder的编写用到了Java、C、Web和Ruby。HBuilder本身主体是由Java编写，它基于Eclipse，所以顺其自然地兼容了Eclipse的插件。

HBuilderX：当前最快的HTML开发工具,强大的代码助手帮你快速完成开发,最全的语法库和浏览器兼容性数据让浏览器碎片化不再头痛。

**Sublime Text**：一个代码编辑器，也是HTML和散文先进的文本编辑器。器！轻量级IDE，神器神在可扩展性，工具包含大量实用插件，我们可以通过安装自己领域的插件来提高工作效率。支持VIM模式（LINUX推荐）、支持宏（批处理）。

**WebStorm**：jetbrains公司旗下一款JavaScript 开发工具。目前已经被广大中国JS开发者誉为“Web前端开发神器”、“最强大的HTML5编辑器”、“最智能的JavaScript IDE”等。与IntelliJ IDEA同源，继承了IntelliJ IDEA强大的JS部分的功能。优秀的代码定位能力，能轻松将光标定位到变量/属性/方法的定义处，对阅读代码非常有用。nodejs 最好的开发工具。

**Visual Studio Code**：微软推出的良心之作。是一款免费开源的现代化轻量级代码编辑器，支持几乎所有主流的开发语言的语法高亮、智能代码补全、自定义快捷键、括号匹配和颜色区分。

1. **世界五大主流浏览器?**

目前最为主流浏览器有五大款，分别是IE、Firefox、Google Chrome、Safari、Opera。

1. **浏览器内核的作用（\*）?**

浏览器最重要或者说核心的部分是“Rendering Engine”，可大概译为“渲染引擎”，不过我们一般习惯将之称为“浏览器内核”。**负责对网页语法的解释（如标准通用标记语言下的一个应用HTML、JavaScript）并渲染（显示）网页。** 所以，通常所谓的浏览器内核也就是浏览器所采用的渲染引擎，渲染引擎决定了浏览器如何显示网页的内容以及页面的格式信息。**不同的浏览器内核对网页编写语法的解释也有不同，因此同一网页在不同的内核的浏览器里的渲染（显示）效果也可能不同，这也是网页编写者需要在不同内核的浏览器中测试网页显示效果的原因。**

1. **四大内核以及对应浏览器（\*）?**

四大内核分别是：Trident（也称IE内核）、webkit、Blink、Gecko。

五大浏览器采用的都是单内核，而是随着浏览器的发展现在也出现了双内核。像360浏览器、QQ浏览器都是采用双内核。

1、IE浏览器内核：Trident内核，也是俗称的IE内核；

2、Chrome浏览器内核：统称为Chromium内核或Chrome内核，以前是Webkit内核，现在是Blink内核；

3、Firefox浏览器内核：Gecko内核，俗称Firefox内核；

4、Safari浏览器内核：Webkit内核；

5、Opera浏览器内核：最初是自己的Presto内核，后来是Webkit，现在是Blink内核；

6、360浏览器、猎豹浏览器内核：IE+Chrome双内核；

7、搜狗、遨游、QQ浏览器内核：Trident（兼容模式）+Webkit（高速模式）；

8、百度浏览器、世界之窗内核：IE内核；

9、2345浏览器内核：以前是IE内核，现在也是IE+Chrome双内核；

1. **HTML概念?**

HTML 指的是超文本标记语言 (Hyper Text Markup Language)，“超文本”就是指页面内可以包含图片、链接，甚至音乐、视频等非文字元素。

1. **HTML5之前都在使用什么版本?**

HTML4.01 XHTML1.0

1. **前端页面三层构成（\*）?**

页面的构成被分为三个主要部分：结构(Structure)、表现(Presentation)和行为(Behavior)。

结构层：HTML/XHTML/HTML5等超文本标记语言，通过使用标记标签来描述网页，定义网页结构；

表现层：层叠样式表（Cascading Style Sheets）,是用于（增强）控制网页样式并允许将样式信息与网页内容分离的一种标记性语言。

行为层：JavaScript脚本语言，用来给页面添加动态效果和动态交互。

1. **url的概念以及组成部分（\*）?**

Uniform Resource Locator; 统一资源定位器（URL）指的是Internet文件在网上的地址。

它从左到右由下述部分组成：

协议：http通信协议。

服务器地址（host）：指出WWW页所在的服务器域名。

端口（port）：有时（并非总是这样），对某些资源的访问来说，需给出相应的服务器提供端口号。

路径（path）：指明服务器上某资源的位置（其格式与DOS系统中的格式一样，通常有目录/子目录/文件名这样结构组成）。与端口一样，路径并非总是需要的。

1. **DOCTYPE 的作用是什么（\*）？**

**<!DOCTYPE> 声明一般位于文档的第一行，它的作用主要是告诉浏览器以什么样的模式来解析文档。**一般指定了之后会以标准模式来 进行文档解析，否则就以兼容模式进行解析。

1. **标准模式与兼容模式各有什么区别（\*）？**

标准模式的渲染方式和 JS 引擎的解析方式都是以该浏览器支持的最高标准运行。

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

1. **DTD 介绍（\*）？**

DTD（ Document Type Definition 文档类型定义）是一组机器可读的规则，它们定义 XML 或 HTML 的特定版本中所有允许元 素及它们的属性和层次关系的定义。在解析网页时，浏览器将使用这些规则检查页面的有效性并且采取相应的措施。

DTD 是对 HTML 文档的声明，还会影响浏览器的渲染模式（工作模式）。

1. **为什么HTML5不需要dtd约束（\*）?**

HTML5 不基于 SGML，因此不需要对 DTD 进行引用，但是需要 DOCTYPE 来规范浏览器的行为（让浏览器按照它们应该的方式来运 行）。

而 HTML4.01 基于 SGML ，所以需要对 DTD 进行引用，才能告知浏览器文档所使用的文档类型。

1. **SGML 、 HTML 、XML 和 XHTML 的区别（\*）？**

SGML 是标准通用标记语言，是一种定义电子文档结构和描述其内容的国际标准语言，是所有电子文档标记语言的起源。

HTML 是超文本标记语言，主要是用于规定怎么显示网页。

XML 是可扩展标记语言是未来网页语言的发展方向，XML 和 HTML 的最大区别就在于 XML 的标签是可以自己创建的，数量无限多， 而 HTML 的标签都是固定的而且数量有限。

XHTML 也是现在基本上所有网页都在用的标记语言，他其实和 HTML 没什么本质的区别，标签都一样，用法也都一样，就是比 HTML 更严格，比如标签必须都用小写，标签都必须有闭合标签等。

1. **HTML5的新特性有哪些（\*）?**

用以绘画的 canvas 元素

用以媒介播放的 video 和 audio 元素

对本地离线存储的更好的支持

新的语义化标签 例如:article footer header nav section

新的表单控件 例如:date time email url search

1. **HTML标签的分类及其特点（\*）?**

html标签又叫做html元素，它分为块级元素、行内块元素和行内元素。

块级元素是指本身样式属性为display:block;的元素。其特点是：独占一行,可以设置宽高。

行内(内联)元素是指本身属性为display:inline;的元素。其特点是:有多大占多大,不可以设置宽高。

行内块元素是指本身属性为display:inline-block;的元素。其特点是:有多大占多大,可以设置宽高。

1. **HTML和CSS里面常用的长度单位,分别介绍（\*）?**

**px**：像素或许被认为是最好的“设备像素”，而这种像素长度和你在显示器上看到的文字屏幕像素无关。在web上，像素仍然是典型的度量单位，很多其他长度单位直接映射成像素，最终，他们被按照像素处理，javascript语言里的单位就是使用的像素。

in：英寸是一个物理度量单位，但是在CSS领域，英寸只不过被直接映射成像素罢了。但是，到目前为止我还没有见过哪个实例用的是英寸，当然也没见过使用下面那些物理度量单位的实例。1in == 96px

cm：对于世界上大多数的人来说，厘米是比较熟悉有用的物理度量单位。它也映射成像素。1cm == 37.8px

mm：毫米是个小数量级的物理度量单位。1mm == 0.1cm == 3.78px

**em**：em 是一个相对单位。起初排版度量时是基于当前字体大写字母”M”的尺寸的。当改变font-family时，它的尺寸不会发生改变，但在改变font-size的大小时，它的尺寸就会发生变化。

**rem**：rem和em一样也是一个相对单位，但是和em不同的是rem总是相对于根元素(如:root{})，而不像em一样使用级联的方式来计算尺寸。这种相对单位使用起来更简单。

**vw**：vw是可视区宽度单位。1vw等于可视区宽度的百分之一。

**vh**：vh和vw（viewport widht）单位一样，不同的是vh是相对于可视区的高度。

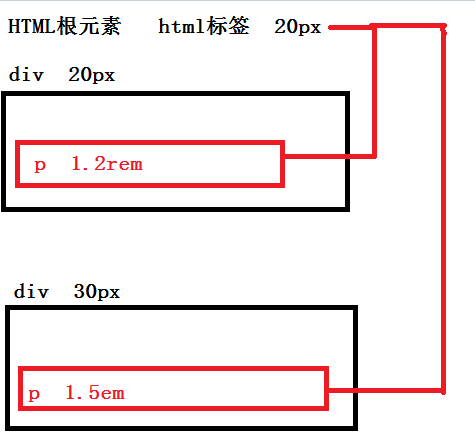
**百分比**：以百分比为单位的长度值是基于具有相同属性的父元素的长度值。

1. **em和rem单位的区别（\*）?**

Em图解:



rem图解:



em和rem都是相对单位。

em和rem相对于px更具有灵活性，他们是相对长度单位，意思是长度不是定死了的，更适用于响应式布局。

对于em和rem的区别一句话概括：em相对于父元素，rem相对于根元素。

1. **CSS中能不能实现合并单元格?为什么（\*）?**

不能。

CSS是层叠样式表，是一种用来控制HTML元素样式的代码；通过CSS控制，你可以改变页面上任何元素的样式，同样也要知道，它所能做的，也仅仅是改变元素的样式，样式以外的东西是没法改变的；因为合并单元格，要删除多余的单元格，那么对于改变元素的结构，CSS是心有余而力不足的。

1. **现实生活中,那么操作是向服务器发送请求呢?而且是什么方式的请求?**
2. 表单提交；可以是get方式也可以是post方式，有method决定。
3. 点击超链接；get方式
4. 地址栏输入地址回车；get方式
5. 异步请求，ajax
6. **form表单的常用属性有哪些，分别做什么?**

action 指定表单的发送地址(服务器地址)

method 表单数据发送至服务器的方法,常用的有两种 get(默认)/post

注：

**表单控件的常用属性?** type、name、required、placeholder等

**常见的表单控件类型有哪些? 下拉框、密码框、文本框等**

1. **表单常见提交方式是什么?有什么区别（\*）?**

get提交:用户输入的信息,显示在地址栏中,不安全,请求内容长度有限制,请求速度快

post提交:用户输入的信息,不会显示在地址栏，安全,请求内容长度无限制(重要数据)，请求速度慢。

1. **label标签的作用以及用法（\*）?**

label 元素不会向用户呈现任何特殊效果。不过，它为鼠标用户改进了可用性。如果您在 label 元素内点击文本，就会触发此控件。就是说，当用户选择该标签时，浏览器就会自动将焦点转到和label标签相关的表单控件上。

方式一:

for属性,让标签和指点的input元素建立关联,多数使用在单选或复选

,给单选或复选后面的文字加入label标签,for属性值是input的id值

方式二:

将input元素包含在lable标签中；如果将input放置在label标签之间,那么for属性就可以不用

注:和单选、复选结合使用,提高用户体验

1. **表单的执行原理解析?**
2. 用户在客户端填写表单数据；
3. 通过客户端浏览器把数据提交给服务端

表单的action属性

异步ajax

1. 服务端解析请求对象获取表单提交的数据并处理
2. 根据提交的请求给客户端一个响应.

**请求/响应模式:客户端浏览器发送请求给服务器;服务器给予客户端浏览器响应;**

1. **<img>的title和alt有什么区别（\*）？**

title是全局属性之一，用于为元素提供附加的额外信息。通常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，用于图片无法加载时显示、读屏器阅读图片。可提图片高可访问性，除了纯装饰图片外都必须设置有意义的值，搜索引擎会重点分析。

1. **超链接target属性的值有哪些?**

target：超链接的打开方式

\_blank 新窗口打开

\_self 当前窗口打开(默认)

\_parent 父窗口打开

\_top 顶层窗口打开

iframeName 在指定的ifranem框架打开

1. **iframe的优缺点（\*）？**

优点：

a. 解决加载缓慢的第三方内容如图标和广告等的加载问题

b. iframe无刷新文件上传

c. iframe跨域通信

缺点：

a. iframe会阻塞主页面的Onload事件

b. 无法被一些搜索引擎索引到

c. 页面会增加服务器的http请求

d. 会产生很多页面，不容易管理。

1. **b和strong的区别（\*）?**

b和strong在效果上都是呈现加粗文本。

b没有任何语义，strong代表语气更强烈的强调文本。

我们通常标签 <strong>替换加粗标签<b>来使用。

1. **i和em的区别（\*）?**

i和em在效果上都是呈现斜体文本。

i没有任何语义，em代表强调文本。

我们通常标签<em>替换<i>标签使用。

1. **常用的特殊符号?**

空格： &nbsp;

小于号：&lt;

大于号：&gt;

&：&amp;

双引号：&quot;

版权：&copy;

注册：&reg;

1. **H5 是什么？**

H5是HTML5的简称，就是“HTML”的第5个版本，也就是第5个版本的超文本标记语言。

1. **你是如何理解语义化的（\*）？**

语义化，顾名思义，就是你写的HTML结构，是用相对应的有一定语义的英文字母（标签）表示的，标记的，因为HTML本身就是标记语言。不仅对自己来说，容易阅读，书写。别人看你的代码和结构也容易理解，甚至对一些不是做网页开发的人来说，也容易阅读。那么，我们以后再开发的过程中，一定要注意了，尽量使用官方的有语义的标签，不要再使用一堆无意义的标签去堆你的结构。

语义化，也无非就是自己在使用标签的时候多使用有英文语义的标签，比如h标签，在HTML中就是就是用来定义标题，还有p标签，英文是paragraph段落，table表格标签,等等。

1. **HTML5有哪些新的布局标签?好处是什么（\*）?**

<header>：标签描述了文档的头部区域

<nav>：标签定义导航链接的部分。

<article>：标签定义独立的内容。

<section>：标签定义文档中的节（section、区段）。

<aside>：标签定义页面主区域内容之外的内容（比如侧边栏）。

<footer>：标签描述了文档的底部区域.

HTML5新增的还有article、nav、header、footer.....等等等，其实现效果都是和div一样没有效果，但是合理使用却能让页面结构更加清晰有逻辑。一方面程序可读性和维护性加强，另一方面，可以用下级选择器来写样式，减少了class和id的命名，可以精简网页的结构；而且语义化结构 有条理才能让搜索引擎更加友好。

1. **HTML5新的语义化标签有哪些（\*）？**

|  |  |
| --- | --- |
| <article> | 定义独立的内容 |
| <aside> | 定义页面内容之外的内容 |
| <audio> | 定义声音内容 |
| <bdi> | 定义文本的文本方向，使其脱离其周围文本的方向设置 |
| <canvas> | 定义图形 |
| <command> | 定义命令按钮 |
| <datalist> | 定义数据列表 |
| <details> | 定义元素的细节 |
| <embed> | 定义外部交互内容或插件 |
| <figcaption> | 定义figure元素的标题 |
| <figure> | 定义媒介内容的分组，以及他们的标题 |
| <footer> | 定义section或page的页脚 |
| <header> | 定义section或page的页眉 |
| <hgroup> | 定义有关文档中的section信息 |
| <keygen> | 定义生成密钥 |
| <mark> | 定义有记号的文本 |
| <meter> | 定义预定义范围内的度量 |
| <nav> | 定义导航链接 |
| <output> | 定义输出的一些类型 |
| <progress> | 定义任何类型的任务的进度 |
| <rp> | 定义若浏览器不支持ruby元素显示的内容 |
| <rt> | 定义ruby注释的解释 |
| <ruby> | 定义ruby注释 |
| <section> | 定义section |
| <source> | 定义媒介源 |
| <summary> | 定义details元素的标题 |
| <time> | 定义日期/时间 |
| <track> | 定义用在媒体播放器中的文本轨道 |
| <video> | 定义视频 |

1. **HTML5新增表单控件有哪些?分别做什么（\*）?**

input[type=email] 表示输入一个电子邮箱(有校验)

input[type=color] 用于规定颜色框

input[type=date/datetime-local/month/week/time] 用于规定日期/日期时间/年月/年中周数/时间

input[type=number] 用于包含数字值的输入字段

input[type=range] 用于应该包含指定范围值的输入字段

input[type=search] 用于搜索字段

input[type=tel] 用于应该包含电话号码的输入字段,但是不会进行校验

input[type=url] 用于包含 URL 地址的输入字段

<datalist>：元素规定输入域的选项列表。

1. **HTML5新增表单控件属性（\*）?**

input **placeholder** 属性:提供一种提示（hint），描述输入域所期待的值。

input **required** 属性:规定必须在提交之前填写输入域

input step 属性:step 属性为输入域规定合法的数字间隔。

input **autofocus** 属性:autofocus 属性规定在页面加载时，域自动地获得焦点。

input form 属性:form 属性规定输入域所属的一个或多个表单。

input **list** 属性:list 属性规定输入域的 datalist。datalist 是输入域的选项列表。

input **multiple** 属性:规定<input>元素中可选择多个值。

1. **HTML5规范对于哪些音频和视频格式支持好?哪些好的图片格式?**

视频格式:

**WebM由Google提出，是一个开放、免费的媒体文件格式。**WebM标准的网络视频更加偏向于开源并且是**基于HTML5标准的**，WebM 项目旨在为对每个人都开放的网络开发高质量、开放的视频格式，其重点是解决视频服务这一核心的网络用户体验。

音频格式:

Ogg全称是OGGVobis(oggVorbis)是一种音频压缩格式，类似于MP3等的音乐格式。Ogg是完全免费、开放和没有专利限制的。OggVorbis文件的扩展名是".ogg"。Ogg文件格式可以不断地进行大小和音质的改良，而不影响旧有的编码器或播放器。

图片格式：

WebP (发音"weppy")，是一种同时提供了有损压缩与无损压缩的图片文件格式，派生自图像编码格式VP8 。

与JPEG相同，WebP是一种有损压缩利用预测编码技术。但谷歌表示，这种格式的主要优势在于高效率。他们发现，“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%。

1. **H5已经弃用的标签（\*）?(5种)**

<basefont> 规定页面上的默认字体颜色和字号

<big> 让文本比常规的字体大一号

<center> 将 HTML 网页中的文本进行水平居中处理

<dir> 定义目录列表。

<font> 规定文本的字体、字体尺寸、字体颜色。

<frame> 定义 frameset 中的一个特定的窗口（框架）。

<frameset> 可定义一个框架集。它被用来组织多个窗口（框架）。

<strike> 可定义加删除线文本定义。

<tt> 呈现类似打字机或者等宽的文本效果。

1. **SVG 与 Canvas两者间的区别（\*）?**

SVG 是一种使用 XML 描述 2D 图形的语言。Canvas 通过 JavaScript 来绘制 2D 图形。

SVG不依赖分辨率。Canvas依赖分辨率。

SVG支持事件处理器。Canvas 不支持事件处理器。

SVG不适合游戏应用。Canvas适合图像密集型的游戏。

1. **CSS样式的三种来源,分别代表什么（\*）?**

CSS中的样式一共有三种来源：创作人员、读者和用户代理，来源的不同会影响到样式的层叠方式。

创作人员（author's+style）样式应该是我们最熟悉的，如果你是一个前端开发者，那么你写的那些样式就叫做创作人员样式。

用户代理样式（agent's+style），用户代理也就是我们通常所说的浏览器（IE、Firefox等等），这些浏览器会提供一些默认的样式。

读者样式（reader's+style），所谓读者自然就是浏览网页的用户，有些时候这些用户里可能会有人不满意网页的配色，或者字体大小，这时候他们就是通过浏览器提供的接口为网站添加读者样式。

1. **CSS三大机制?分别解释（\*）+ 优先级算法?**

CSS的三大机制是：特殊性[冲突]、继承、层叠。

所谓的特殊性，其实是CSS样式一种计算声明权重的规则。特殊性是由规则的选择器确定的，然后用户代理会将计算出来的特殊性附加给相应的每一个声明上。

继承机制，即样式不仅会应用到指定的样式，而且会应用到其后代的元素。并不是所有的属性都可以继承，一般的，大多数框模型属性（包括外边距、内边距、背景、边框）都不能继承。

文档中的一个元素可能同时被多个css选择器选中，每个选择器都有一些css规则，这就是层叠;

1. **页面导入样式时，使用 link 和 @import 有什么区别（\*）？**

从属关系区别。@import是CSS提供的语法规则，只有导入样式表的作用；link是HTML提供的标签，不仅可以加载CSS文件，还可以定义RSS、rel连接属性、引入网站图标等。

加载顺序区别。加载页面时，link标签引入的CSS文件被同时加载；而@import引入的CSS文件将在页面加载完毕后被加载。

兼容性区别。@import时CSS 2.1才有的语法，故只可在IE5+才能识别；link标签作为HTML标签，不存在兼容性问题。

DOM可控性区别。可以通过JS操作DOM，插入link标签来改变样式；由于DOM方法是基于文档的，无法使用@import的方法插入样式。

<https://blog.csdn.net/Dora_5537/article/details/92792949>

1. **CSS盒模型的组成部分,以及相关样式（\*）?**

所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容 。

实际内容：width 设置宽度、height 设置高度

填充(内边距)：padding

边框：border

外边距：margin

1. **CSS盒模型padding和margin不同值代表的含义（选择题 \*）?**

margin : 10px 5px 8px; 左外边距是多少? 5px

元素在页面中最终占据的大小是由盒模型来决定的,不是由宽高;

\* 表示四周的内/外边距

\*\* 表示上下、左右的内/外边距

\*\*\* 表示上、左右、下的内/外边距

\*\*\*\* 表示上、右、下、左的内/外边距

1. **IE盒模型和W3C盒模型的概念（\*）?**

w3c盒子模型的范围包括margin、border、padding、content,并且content部分不包含其他部分。

IE盒子模型的范围包括margin、border、padding、content,和w3c盒子模型不同的是，IE盒子模型的content部分包含了padding和border。

在标准兼容模式下使用的是W3C的盒模型标准。

css3的box-sizing属性给了开发者选择盒模型解析方式的权利。W3C的盒模型方式被称为“content-box”，IE的被称为“border-box”，使用box-sizing: border-box;就是为了在设置有padding值和border值的时候不把宽度撑开。

1. **设置超链接的四种状态,但是效果不正确,怎么解决（\*）?**

通过选择器按照顺序设置超链接的四种状态、

顺序是：

a:link - 正常，未访问过的链接

a:visited - 用户已访问过的链接

a:hover - 当用户鼠标放在链接上时

a:active - 链接被点击的那一刻

1. **CSS清除浮动影响的方式有哪些?以及他们的原理（\*\*\*）?**
2. 给父级元素单独定义高度（height）；

原理：如果父级元素没有定义高度，父元素的高度完全由子元素撑开时，父级div手动定义height，就解决了父级div无法自动获取到高度的问题。

优点：简单、代码少、容易掌握。

缺点：只适合高度固定的布局，要给出精确的高度，如果高度和父级div不一样时，会产生问题。对于响应式布局会有很大影响。

1. 在标签结尾处前加空div标签 clear:both

原理：添加一个空div，利用css提供的clear:both清除浮动，让父级div能自动获取到高度。

优点：简单、代码少、浏览器支持好、不容易出现怪问题

缺点：不少初学者不理解原理；如果页面浮动布局多，就要增加很多空div，不利于页面的优化。

1. 父级div定义 伪元素:after 和 zoom

原理：元素生成伪类的作用和效果相当于方法2中的原理，但是IE8以上和非IE浏览器才支持:after，zoom(IE专有属性)可解决ie6,ie7浮动问题。

优点：浏览器支持好、不容易出现怪问题，写法是固定的，不理解也可以直接复制使用；（推荐使用此种方法，简单便捷，只需添加一个class即可解决问题）

缺点：css代码多、不少初学者不理解原理，要两句代码结合使用才能让主流浏览器都支持。

1. 父级div定义 overflow:hidden

原理: 当添加overflow:hidden的时候,父元素会去适配所有子元素,即使脱离文档流的子元素,那么适配之后,父元素就有了高度;

优点：简单、代码少、浏览器支持好

缺点：不能和position配合使用，因为超出的尺寸的会被隐藏。

1. **display:none和visibility:hidden两者的区别（\*）？**

给一个元素设置了display: none，那么该元素以及它的所有后代元素都会隐藏。

给元素设置visibility: hidden也可以隐藏这个元素，但是隐藏元素仍需占用与未隐藏时一样的空间，也就是说虽然元素不可见了，但是仍然会影响页面布局。

1. **rgba()设置透明度和opacity设置透明度的区别?**

rgba()设置的透明度只是针对于元素本身,不会作用于内容

opacity设置透明度不仅作用于元素本身,还作用于内容

1. **如何在页面中设置元素不可见?**

visibility:hidden 隐藏元素,但是还在页面中占据位置

display:none 隐藏元素,不在页面中占据位置

opacity:0 隐藏元素,但是还在页面中占据位置

1. **请阐述CSS精灵截图技术(图片拼合技术)概念以及实现（应用题 \*）?**

css sprites直译过来就是CSS精灵。通常被解释为“CSS图像拼合”或“CSS贴图定位”。其实就是通过将多个图片融合到一张图里面，然后通过CSS background背景定位技术技巧布局网页背景。这样做的好处也是显而易见的，因为图片多的话，会增加http的请求，无疑促使了网站性能的减低，特别是图片特别多的网站，如果能用css sprites降低图片数量，带来的将是速度的提升。

css sprites是什么通俗解释：**CSS Sprites其实就是把网页中一些背景图片整合拼合成一张图片中，再利用DIV CSS的“background-image”，“background- repeat”，“background-position”的组合进行背景定位，background-position可以用数字能精确的定位出背景图片在布局盒子对象位置。**

1. **在开发过程中,有哪些方法可以提高网站的性能?**
2. 使用新的语义化标签,有利于浏览器抓取和解析
3. 使用小体积的图片格式,比如webp
4. 使用基于H5标准的音频和使用,比如ogg和webm
5. 提高CSS代码复用性
6. 图片拼合技术
7. **CSS3新增选择器有哪些（至少5种 \*）?**

属性选择器：

attribute^=value 选择具有指定属性并且属性值以特定字符开头的元素。

attribute$=value 选择具有指定属性并且属性值以特定字符结尾的元素。

attribute\*=value 选择具有指定属性并且属性值以包含特点字符的元素。

结构伪类选择器：

:only-child(number/odd/even/表达式) 选择父元素的第n个子元素

:last-child 选择父元素的最后一个子元素

伪元素选择器：

::selection 匹配元素中被用户选中或处于高亮状态的部分

表单选择器：

:enabled 选择每一个已启用的输入元素

:disabled 选择每一个禁用的输入元素

:checked 选择每个选中的输入元素

其他选择器：

:empty 选择没有任何子级的元素

:not(selector) 选择非指定参数选择器元素

1. **CSS3新增样式有哪些(至少5种 \*)?**

border-radius 用于创建圆角。

box-shadow 设置一个或多个下拉阴影的框

border-image 设置图片边框

background-size 指定背景图像的大小

background-image: linear-gradient(direction/degree ,color-stop1, color-stop2, ...); 设置线性渐变

background-image: radial-gradient(shape size at position, start-color, ..., last-color); 设置径向渐变

text-shadow 设置文本阴影。

text-overflow 指定应向用户如何显示溢出内容。

transform 用于设置2D或3D转换的元素

transition 用于设置过渡效果。

@keyframes 用于设置关键帧动画

1. **牢记border-radius定义不同值代表的含义?**

border-radius: 10px 5px 6px;

\* 设置四个角圆角

\*\* 设置左上右下、左下右上的圆角

\*\*\* 设置左上、左下右上、右下的圆角

\*\*\*\* 设置左上、右上、右下、左下的圆角

1. **display:flex 和 display: inline-flex的区别?**

display:flex： 将对象作为弹性伸缩盒显示

display:inline-flex：将对象作为内联块级弹性伸缩盒显示

1. **请详细阐述Flex布局（\*\*\*）?**

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

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"。

容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。

容器样式属性：

flex-direction：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。

flex-wrap：定义如果一条轴线排不下，如何换行。

nowrap（默认）：不换行。

wrap：换行，第一行在上方。

wrap-reverse：换行，第一行在下方。

flex-flow：flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

justify-content：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。

align-items：定义项目在交叉轴上如何对齐。

flex-start：交叉轴的起点对齐。

flex-end：交叉轴的终点对齐。

center：交叉轴的中点对齐。

baseline: 项目的第一行文字的基线对齐。

stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

align-content：定义了多根轴线的对齐方式。

flex-start：与交叉轴的起点对齐。

flex-end：与交叉轴的终点对齐。

center：与交叉轴的中点对齐。

space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。

space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。

stretch（默认值）：轴线占满整个交叉轴。

项目样式属性：

order：定义项目的排列顺序。数值越小，排列越靠前，默认为0。

flex-grow：定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。

flex-shrink：定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小

flex-basis：定义了在分配多余空间之前，项目占据的主轴空间（main size）。

flex:flex属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

align-self:允许单个项目有与其他项目不一样的对齐方式.

1. **什么是CSS Reset（\*），为什么使用?**

**HTML标签在浏览器中都有默认的样式，不同的浏览器的默认样式之间存在差别。**例如ul默认带有缩进样式，在IE下，它的缩进是由margin实现的，而在Firefox下却是由padding实现的。**开发时浏览器的默认样式可能会给我们带来多浏览器兼容性问题，影响开发效率。所以解决的方法就是一开始就将浏览器的默认样式全部去掉，更准确说就是通过重新定义标签样式。**“覆盖”浏览器的CSS默认属性。最最简单的说法就是**把浏览器提供的默认样式覆盖掉！这就是CSS reset。**

1. **你能描述一下渐进增强和优雅降级之间的不同吗（\*）?**

|  |
| --- |
| //渐进增强  .transition{  -webkit-transition: all .5s;  -moz-transition: all .5s;  -o-transition: all .5s;  transition: all .5s;  }  //优雅降级  .transition{  　　 transition: all .5s;  　　 -o-transition: all .5s;  　-moz-transition: all .5s;  -webkit-transition: all .5s;  } |

渐进增强（progressive enhancement）：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。（从被所有浏览器支持的基本功能开始，逐步地添加那些只有新式浏览器才支持的功能，向页面添加无害于基础浏览器的额外样式和功能。当浏览器支持时，它们会自动地呈现出来并发挥作用。）

　　优雅降级（graceful degradation）：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。（Web站点在所有新式浏览器中都能正常工作，如果用户使用的是老式浏览器，则代码会检查以确认它们是否能正常工作。由于IE独特的盒模型布局问题，针对不同版本的IE的hack实践过优雅降级了，为那些无法支持功能的浏览器增加候选方案，使之在旧式浏览器上以某种形式降级体验却不至于完全失效。）

　　区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的、能够起作用的版本开始，并不断扩充，以适应未来环境的需要。

1. **什么是弱类型语言，什么是强类型语言?**

强类型: 定义变量,必须指明其类型;

byte b = 10; String str = “1”; char=’1’

弱类型: 定义变量,无序指明其类型,统一使用var来定义变量;

var b = 10; var str = “abc”;

1. **JavaScript的组成部分（\*）?**

JavaScript是ECMAScript、文档对象模型（document object model:DOM）、浏览器对象模型（brower object model:BOM）由三部分构成。

ECMAScript，描述了该语言的语法和基本对象(内置对象)。

文档对象模型（DOM），描述处理网页内容的方法和接口。通过 DOM，可以访问所有的 HTML 元素，连同它们所包含的文本和属性，可以对其中的内容进行修改和删除，同时也可以创建新的元素；

浏览器对象模型（BOM），描述与浏览器进行交互的方法和接口。BOM提供了独立于内容而与浏览器窗口进行交互的对象，例如可以移动，调整浏览器大小的window对象，可以用于导航的location对象与history对象，可以获取浏览器，操作系统与用户屏幕信息的navigator与screen对象。

1. **JavaScript特点或特性?**

JavaScript是一种属于网络的脚本语言,已经被广泛用于Web应用开发,常用来为网页添加各式各样的动态功能,为用户提供更流畅美观的浏览效果。

脚本语言。JavaScript是一种解释型的脚本语言,C、C++等语言先编译后执行,而JavaScript是在程序的运行过程中逐行进行解释。

基于对象。JavaScript是一种基于对象的脚本语言,它不仅可以创建对象,也能使用现有的对象(内置对象)。

简单性。JavaScript语言中采用的是弱类型的变量类型,对使用的数据类型未做出严格的要求,是基于Java基本语句和控制的脚本语言,其设计简单紧凑。

动态性。JavaScript是一种采用事件驱动的脚本语言,它不需要经过Web服务器就可以对用户的输入做出响应。在访问一个网页时,鼠标在网页中进行鼠标点击或上下移、窗口移动等操作JavaScript都可直接对这些事件给出相应的响应（针对于用户在页面中的操作,都是基于事件驱动）。

跨平台性：JavaScript脚本语言不依赖于操作系统,仅需要浏览器的支持。因此一个JavaScript脚本在编写后可以带到任意机器上使用,前提上机器上的浏览器支 持JavaScript脚本语言,目前JavaScript已被大多数的浏览器所支持。

1. **JavaScript ECMAScript包含什么内容?**

基础语法和内置对象

基础语法：变量、数据类型、运算符和表达式、流程控制语句

内置对象：Date、String、Array、Global、Boolean、Number等

1. **JavaScript数据类型有哪些（\*）?**

值类型（原始类型）：数字Number、字符串String、布尔Boolean、空类型Null、未定义Undefined

引用类型（对象类型）：Object、Array等内置对象、Function函数对象

1. **JavaScript里面的NaN代表什么意思?属于什么类型（\*）?**

NaN：Not a Number,不是一个数值

在JS里面属于Number类型。

1. **JavaScript 中的强制转型是指什么（\*）？**

在 JavaScript 中，两种不同的内置类型间的转换被称为强制转型。强制转型在 JavaScript 中有两种形式：显式和隐式。

这是一个显式强制转型的例子：

var a = "42";

var b = Number( a );

这是一个隐式强制转型的例子：

var a = "42";

var b = a - 1; // "42" 隐式转型成 42

1. **以下两条语法,分别表示的是什么?**

var arr2 = new Array(5); 定义长度为5的JS数组

var arr6 = [5]; 定义数组并给其下标0赋值为5

1. **Null和Undefined的区别（\*）?**

JavaScript 中有两种原始类型：null 和 undefined。

Null 已定义，并初始化值为null;

Undefined:未定义，或者未初始化 。

null和undefined是两个不同的对象, 有图为证:



1. **在JS里面== 和 ===的区别（\*）?**

相等运算符 == ：判断两个操作数是否相等。不同的数据类型会自动转换为相等的数据类型再做比较。

等同(恒等)运算符=== ：全等（值和类型），严格意义的相等，两个操作数的值和他们的类型必须完全一致。

1. **如下语句得到的结果分别是什么(js里面的特殊比较 选择题)?**

null == undefined true

NaN == NaN false

undefined == 0 false

null == 0 false

1. **Switch语句和if多分支语句的区别（\*）?**

相同点：他们都是用来处理多分支条件的结构。

不同点：switch语句只能处理等值条件判断的情况，而且条件只能是整型变量或字符型变量。多重if选择结构的限制，特别适合某个变量处于某个连续区间时的情况。

1. **While 和 do while的区别（\*）?**

while语句和 do…while语句类似，都是要判断循环条件是否为真。如果为真，则执行循环体，否则退出循环。

while语句和 do…while语句的区别在于：do…while语句是先执行一次循环体，然后再判断。因此 do…while语句至少要执行一次循环体。而 while是先判断后执行，如果条件不成立或不满足，则一次循环体也不执行。

1. **JavaScrip的一些编写原则（\* - 开发中的细节）?**

1、不要在同一行声明多个变量

2、使用===或!==来比较

3、使用字面量的方式来创建对象、数组，替代new Array这种形式

4、switch语句必须要带default分支

5、fon-in循环中的变量，用var关键字说明作用域，防止变量污染

6、三元表达式可以替代if双分支语句

7、比较数据类型以下6中情况是false，其他都是true------false、""、0、null、undefined、NaN

8、数据类型检测用typeof，对象类型检测用instanceof

9、异步加载第三方的内容

10、单行注释//，多行注释/\*\*/

1. **循环的使用场合?**

已知循环次数用for循环

不知道循环次数且可能一次都不循环用while循环

不知道循环次数且至少循环一次用do-while循环

1. **怎么通过break结束指定的循环（\*）?**

|  |
| --- |
| //给for循环定义标记,然后在break后面指明要结束的循环标记  tt:for(var i = 0;i < 10;i++){  for(var j = 0;i < 10;j++){   if(j == 5){   break tt; //结束外层for循环                   }                    document.write("Hell0<br>");              }                  document.write("<hr>");   } |

1. **break和continue的区别（\*）?**

break将直接跳出并结束当前循环结构。

continue用于跳过循环中的一个迭代。

continue 语句只能用在循环中；break只能用在循环或 switch 中。

1. **形参和实参的区别（\*）?**

1、出现的地方不一样，形参是在函数的声明中，实参出现在函数的调用中。

2、代表的意义不一样，形参是声明一个变量，实参是该变量的值。

1. **JS局部变量和全局变量的区别（\*）?**

在JavaScript函数内部声明的变量是局部变量，所以只能在函数内部访问它，该变量的作用域是局部的。只要函数运行完毕，局部变量就会被删除。

在函数外声明的变量是全局变量，网页上的所有脚本和函数都能访问它。 全局变量会在页面关闭后被删除。

1. **匿名函数的五种用法?**

方式一：把它放进一个变量里，这个变量就相当于一个函数名了。没有名字的匿名函数，变成有“名”函数了，我们把这种方式称之为函数字面量。

|  |
| --- |
| var myFun = function( a,b ){  console.info( a+b);  };  myFun( 10,30 ); |

方式二：干脆不要名字，直接执行~！-- 这么做可以在内部形成局部变量和局部函数，防止全局污染。

|  |
| --- |
| (function(a,b){  console.info( a+b );  }(10,30)); |

方式三：利用事件去调用。

|  |
| --- |
| var btn = document.getElementById("btn"); // 找到页面某个标签  // 添加事件  btn.onclick = function(){  console.info("你点了我！");  } |

方法四：作为对象的方法调用

|  |
| --- |
| var myObj = {  name : "John",  sayHello:function(){  console.info("Hello,"+ this.name );  }  };  myObj.sayHello(); |

方法五：作为另一个函数的参数。

函数也可以作为另一个函数的参数的。当然有名函数也可以做参数。

|  |
| --- |
| function myFun(fn){  fn();  }  myFun( function(){  console.info("这个匿名函数是个参数");  }); |

1. **值传递和引用传递的区别（\*）?**

值传递和引用传递都是发生在函数调用中；

**值传递：**实参传递给形参的是值，形参和实参在内存上是两个独立的变量，对形参做任何修改不会影响实参

**引用传递：**实参传递给形参的是参数对于内存上的引用地址，实参和 形参在内存上指向了同一块区域，对形参的修改会影响实参

1. **介绍js有哪些常用内置对象（\*）?**

1.String(字符串对象),

2.Number(数值对象),

3.Boolean（布尔对象）,

4.Function(函数对象),

5.Array(数组),

6.Math(数学运算对象),

7.Object(Object根对象,所有对象的父对象),

8.RegExp（正则表达式对象）,

9.Date(日期对象)，

10.Error(异常对象)

1. **JS中,如何把数组转换为指定字符串,如何把指定字符串拆分成一个数组（\*）?**

Array对象里面的join() 可以把数组转换为指定字符串

String对象里面的split() 可以把字符串转换为数组

1. **Array里面push()和concat()的区别?**

push() 向原数组里面添加元素;如果参数是一个数组,那么会把数组当成一个整体;

concat() 不修改原数组,生成一个新的数组;如果参数是一个数组,那么会把数组里面的元素单独提取出来;

1. **pop()、push()、shift()、unshift()分别做什么的（\*）?**

pop() 删除数组的最后一个元素

push() 添加元素到数组的末尾

shift() 删除数组的第一个元素

unshift() 添加元素到数组的开头

1. **Array对象里面哪些方法可以删除数组的元素（\*）?**

pop() 删除数组的最后一个元素

shift() 删除数组的第一个元素

splice() 删除数组任意下标，任意个数的元素

1. **Array对象里面的哪些方法是添加数组的元素（\*）?**

push() 添加元素到数组的末尾

unshift() 添加元素到数组的开头

splice() 添加元素到数组的任意位置

1. **Array对象里面splice()和slice()的区别（\*）?**

slice() 从原数组截取出来一个新的数组,可以传递开始下标和结束下标

splice() 可以添加数组元素、删除数组元素和替换数组元素；

1. **如下代码的执行结果：**

var arr8 = ["xx", "yy", "zz", "kk", "jj"];

var sliceArr4 = arr8.slice(-4, 3); // ["yy", "zz"]

1. **Math对象里面ceil()、floor()、random()、round()分别做什么使用（\*）?**

ceil()：向上取整

floor()：向下取整

random()：随机0-1的伪随机数

round()：四舍五入

1. **在JS里面如何把字符转换为Unicode编码；如果把Unicode编码转换为字符？**

str.charCodeAt(1) 字符转换为Unicode

String.fromCharCode(97) Unicode转换为字符

1. **JS String对象里面slice()、substr()、substring()有什么区别（\*）?**

都是用来截取字符串的

如果三个方法都是传递一个参数,那么都表示开始下标;表示从开始下标截取到末尾；

如果有第二个参数,slice第二个参数表示结束下标;substr第二个参数表示截取长度;substring它的两个参数谁大谁就是结束下标;

1. **Number()、parseInt()、parseFloat()的区别（\*）?**

Number() 当参数为数值类型字符串的时候,才能正确转换为数值,否则都是NaN

parseInt() 把字符串转换为整数数值,哪怕参数是一个浮点类型;如果字符串前面是数值后面是其他字符,那么会对数值部分进行转换

parseFloat() 把字符串转换为浮点数值;如果字符串前面是数值后面是其他字符,那么会对数值部分进行转换

1. **URL 如何编码解码？为什么要编码（\*）？**

URL 只能使用 ASCII 字符集来通过因特网进行发送。也就是说URL只能使用英文字母、阿拉伯数字和某些标点符号，不能使用其他文字和符号。这意味着如果URL存在汉字，就必须编码后使用。

例如说我们有这样一条 URL ：www.hahaha.com/你好?a=1&b=2，我们如何可以把它合法的在因特网中传播呢？

使用 encodeURIComponent(str) 这个方法来将 utf-8 的字符编码为合法的 URL 。

使用decodeURIComponent(str)这个方法就可以解码你所得到的URL。

<https://www.jianshu.com/p/cfd3c8abb2c4>

1. **什么是 IIFE（立即调用函数表达式）（知道IIFE代表啥意思）？**

它是立即调用函数表达式（Immediately-Invoked Function Expression），简称 IIFE。函数被创建后立即被执行：

|  |
| --- |
| (function IIFE(){  console.log( "Hello!" );  })();  // "Hello!" |

在避免污染全局命名空间时经常使用这种模式，因为 IIFE（与任何其他正常函数一样）内部的所有变量在其作用域之外都是不可见的。

1. **解释什么是回调函数，并提供一个简单的例子（\*）。**

回调函数是可以作为参数传递给另一个函数的函数，并在某些操作完成后执行。下面是一个简单的回调函数示例，这个函数在某些操作完成后打印消息到控制台。

|  |
| --- |
| function modifyArray(arr, callback) {  // 对 arr 做一些操作  arr.push(100);  // 执行传进来的 callback 函数  callback();  }  var arr = [1, 2, 3, 4, 5];  modifyArray(arr, function() {  console.log("array has been modified", arr);  }); |

1. **BOM由哪些对象组成（\*）?**

1.window(核心)

2.document对象 (文档对象)

3.location对象（地址栏对象）

4.navigation对象（浏览器本身相关信息）

5.screen对象（屏幕相关）

6.history对象（浏览器历史相关）

7.frames对象 (内嵌框架集)

1. **JS定时器有哪些?**

延时型定时器: 延迟指定时间后,执行一次

setTimeout()

间隔型定时器: 每隔指定时间后,指向一次

setInterval()

注：定时器都是异步的,不会阻塞后续代码都的执行

1. **如何获取当前窗口文档显示区域的宽高（\*）?**

window.innerWidth 返回窗口的文档显示区的宽度

window.innerHeigt 返回窗口的文档显示区的高度。

1. **请分别阐述Location对象的assign()和replace()的区别（\*）？**

两者共同点都是可以实现页面跳转。

window.location.assign(url) ： 加载 URL 指定的新的 HTML 文档。就相当于一个链接，跳转到指定的url，当前页面会转为新页面内容，可以点击后退返回上一个页面。

​ window.location.replace(url) ： 通过加载 URL 指定的文档来替换当前文档，这个方法是替换当前窗口页面，前后两个页面共用一个窗口，所以是没有后退返回上一页的。

1. **innerText和innerHTML的区别（\*）?**

innerText：用来获取或设置标签的文本内容,不会解析标签。

InnerHTML：用来获取或设置标签的HTML内容,可以解析标签。

1. **获取页面元素的方法有哪些?**

getElementById() 根据id获取页面唯一元素

getElementsByName() 根据name获取页面指定元素

getElementsByTagName() 根据标签名获取页面指定元素

getElementsByClassName() 根据类名获取页面元素

querySelector() 根据选择器选择页面的第一个元素

querySelectorAll() 根据选择器选择页面的指定所有元素

children 获取当前元素的所有子元素

nextElementSibling 获取后一个兄弟元素节点

previousElementSibling 获取前一个兄弟元素节点

firstElementChild 获取第一个子元素

lastElementChild 获取最后一个子元素

**.....**

1. **nodeList和HTMLCollection的区别（\*）?**

nodeList,节点列表;节点列表里面会存在各种节点;

HTMLCollection,HTML集合;集合列表里面之后存在元素节点;

1. **常见节点类型有哪些?nodeName、nodeType、nodeValue分别得到什么（\*）?**

节点类型 nodeName nodeType nodeValue

元素 元素名称 1 null

属性​ 属性名称​ 2​ 属性值

文本​ #text​ 3​ 文本内容

注释​ #comment​ 8 ​ 注释内容

1. **onmouseover、onmouseout、onmouseenter、onmouseleave的作用及区别（\*）** onmouseover和onmouseenter都是鼠标移入元素的时候触发

onmouseout和onmouseleave都是鼠标移除元素的时候触发

onmouseover和onmouseout支持事件冒泡;onmouseenter和onmouseleave不支持冒泡

1. **keydown 和 keypress的区别（\*）?**

两者都是键盘按下未释放的时候触发;

1. keydown按任意键都可以触发;keypress按字符键可以触发;

2. keydown优先于keypress触发;

3. keydown获取的键码不区分大小写; keypress获取的键码区分大小写

1. **常见的事件类型有哪些（\*）？**

鼠标事件:

click 鼠标单击事件,举例:点击显示/隐藏百度登录窗口、点击回到顶部、点击切换登录方式...

dblclick 鼠标双击事件

mouseover 鼠标移入事件

mouseout 鼠标移出事件

案例: 移入显示/移除隐藏、移入停止轮播/移出继续轮播

mouseenter 鼠标移入事件

mouseleave 鼠标移出事件

mouseover/mouseout可以实现的动态效果,它都可以实现;

mousemove 鼠标移动事件

案例: 放大镜

mousedown 鼠标按下事件

案例: 拖拽 - 百度登录窗口

mouseup 鼠标弹起事件

案例: 拖拽 - 百度登录窗口

键盘事件:

keydown 键盘按下事件

keypress 键盘按下事件

keyup 键盘弹起事件

案例: 页面小游戏、输入框的强控制

keydown->所有键可以触发,获取得到的键码不区分大小写

keypress->字符键可以触发,获取得到的键码区分大小写

keydown先于keypress触发

框架/对象事件:

load 页面加载完毕事件

unload 页面卸载事件

beforeunload 页面卸载之前的时间

hashchange URL锚链接改变事件

resize 窗口尺寸改变事件

scroll 内容滚动事件

懒加载

表单事件:

change 内容改变事件;修改了输入框的内容并且失去焦点的时候触发

案例: 级联操作

select 内容字符选择事件,选择了输入框的内容并释放鼠标按键触发

submit 表单提交事件

案例: 客户端校验; 输入的数据不符合要求,阻止表单提交

reset 表单重置事件

focus 获取焦点事件

案例: 获取焦点提示规则

blur 失去焦点事件

案例: 失去焦点验证输入数据

input 输入事件

案例: 百度密码

1. **什么是事件冒泡和事件捕获(捕捉)（\*）?**

事件冒泡：是指嵌套最深的元素触发一个事件，然后这个事件顺着嵌套顺序在父元素上触发。

事件的捕获：与事件冒泡相反，事件会从最外层开始发生，直到最具体的元素。

1. **事件的三个阶段?**

事件的处理过程主要有三个阶段：捕获阶段，目标阶段，冒泡阶段；

捕获阶段：当我们在 DOM 树的某个节点发生了一些操作（例如单击、鼠标移动上去），就会有一个事件发射过去。这个事件从 Window 发出，不断经过下级节点直到触发的目标节点。在到达目标节点之前的过程，就是捕获阶段（Capture Phase）。事件由页面元素接收，逐级向下，到具体的元素。

目标阶段：当事件不断的传递直到目标节点的时候，最终在目标节点上触发这个事件，就是目标阶段。具体的元素本身。

冒泡阶段：事件冒泡即事件开始时，由最具体的元素接收（也就是事件发生所在的节点），然后逐级传播到较为不具体的节点。跟捕获相反，具体元素本身，逐级向上，到页面元素（我们平时用的事件绑定就是利用的事件冒泡的原理）。

1. **什么时候产生事件对象?**

当事件函数触发的时候,该函数的形参接收一个event对象.Event 对象代表事件的状态，比如事件在其中发生的元素、键盘按键的状态、鼠标的位置、鼠标按钮的状态。

1. **事件对象的属性（\*）？**

event.eventPhase 返回事件传播的当前阶段。

**event.target || event.srcElement** 返回触发此事件的元素（事件的目标节点）。

**event.currentTarget** 返回其事件监听器触发该事件的元素

event.bubbles 返回布尔值，指示事件是否是冒泡事件类型

event.cancelable 返回布尔值，指示事件是否可拥可取消的默认动作。

**event.type 获取事件类型**

event.timeStamp 返回事件生成的日期和时间。

**event.clientX/clientY** 所有浏览器都支持，窗口位置

**event.pageX/pageY** IE9以前不支持，页面位置

1. **说明如何使用JavaScript提交表单（\*）？**

要使用JavaScript提交表单，请使用

document.form [0] .submit();

1. **DOM里面哪些API需要进行兼容性处理?**

innerText

|  |
| --- |
| // 设置文本内容兼容性函数  function setInnerText(ele, text) {  if (ele.innerText) { // 如果没有innerText属性,返回undefined  ele.innerText = text;  } else {  ele.textContent = text;  }  }  // 获取文本内容兼容性函数  function getInnerText(ele) {  if (ele.innerText) { // 如果没有innerText属性,返回undefined  return ele.innerText;  } else {  return ele.textContent;  }  } |

firstElementChild/lastElementChild

|  |
| --- |
| // firstElementChild兼容性处理  function getFirstElementChild(element) {  var node, nodes = element.childNodes,  i = 0;  while (node = nodes[i++]) {  if (node.nodeType === 1) {  return node;  }  }  return null;  }  // lastElementChild兼容性处理  function getLastElementChild(element) {  var node, nodes = element.childNodes,  i = nodes.length - 1;  while (node = nodes[i--]) {  if (node.nodeType === 1) {  return node;  }  }  return null;  } |

nextElementSibling/previousElementSibling

|  |
| --- |
| // nextElementSibling兼容性处理  function getNextElementSibling(element) {  var el = element;  while (el = el.nextSibling) {  if (el.nodeType === 1) {  return el;  }  }  return null;  }  // previousElementSibling兼容性处理  function getPreviousElementSibling(element) {  var el = element;  while (el = el.previousSibling) {  if (el.nodeType === 1) {  return el;  }  }  return null;  } |

addEventListener/removeEventListener

|  |
| --- |
| // 添加事件的兼容性处理  function addEventListener(element, type, fn) {  if (element.addEventListener) {  element.addEventListener(type, fn);  } else if (element.attachEvent) {  element.attachEvent("on" + type, fn)  } else {  element["on" + type] = fn;  }  }  // 移除事件的兼容性处理  function removeEventListener(element, type, fn) {  if (element.removeEventListener) {  element.removeEventListener(type, fn);  } else if (element.detachEvent) {  element.detachEvent("on" + type, fn)  } else {  element["on" + type] = null;  }  } |

pageX/pageY

|  |
| --- |
| // 封装获取滚动出去的距离的函数  function getScroll() {  var scrollLeft = document.body.scrollLeft || document.documentElement.scrollLeft;  var scrollTop = document.body.scrollTop || document.documentElement.scrollTop;  return {  scrollLeft: scrollLeft,  scrollTop: scrollTop  }  }  // 封装获取页面坐标的函数,必须用在事件函数里面,依赖于event对象  function getPage(e) {  e = e || window.event;  var pageX = e.pageX || getScroll().scrollLeft + e.clientX;  var pageY = e.pageY || getScroll().scrollTop + e.clientY;  return {  pageX: pageX,  pageY: pageY  }  } |

1. **什么是事件委托（\*）?**

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

1. **为什么要用事件委托（\*\*\*）？**

在JavaScript中，添加到页面上的事件处理程序数量将直接关系到页面的整体运行性能，因为需要不断的与dom节点进行交互，访问dom的次数越多，引起浏览器重绘与重排的次数也就越多，就会延长整个页面的交互就绪时间，这就是为什么性能优化的主要思想之一就是减少DOM操作的原因；如果要用事件委托，就会将所有的操作放到js程序里面，与dom的操作就只需要交互一次，这样就能大大的减少与dom的交互次数，提高性能；

1. **事件委托的原理（\*）?**

事件委托是利用事件的冒泡原理来实现的

1. **如何阻止事件传播（冒泡）（\*）?**

event.stopPropagation()

1. **阻止浏览器的默认行为（\*）?**

event.preventDefault();

事件函数return false;

1. **offsetParent和parentNode的区别（\*）?**

parentNode就是父节点

offsetParent是具有定位的父级节点

1. **offset、client、scroll三大家对比（\*\*\*）?**

offset获取位置和大小:

offsetLeft/offsetTop是相对于offsetParent的位置,前提是父元素必须具有定位属性;否则继续向上查找,直到body

offsetWidth/offsetHeight是获取元素在页面中占据的实际大小

offsetWidth/offsetHeight = width/height + padding + border

client获取位置和大小:

clientLeft/clientTop是获取左边框和上边框的宽度

clientWidth/clientHeight是获取元素的宽高 + 内边距

clientWidth/clientHeight = width/height + padding

scroll获取位置和大小:

scrollLeft/scrollTop是获取左侧和顶部滚动隐藏进去的距离

scrollWidth/scrollHeight是获取元素的宽高 + 内边距 + 未显示的内容,不包含滚动条

scrollWidth/scrollHeight = width/height + padding + 未显示内容 - 滚动条宽度

1. **请分别解释JavaScript内核和JavaScript引擎（\*）?**

JavaScript内核，主要指渲染引擎，负责对网页语法的解释并渲染（显示）网页。

JavaScript引擎用来解释并执行嵌入在网页中的JavaScript（又称ECMAScript）代码

1. **请阐述浏览器渲染过程?**

1. 解析HTML，生成DOM树，解析CSS，生成CSSOM树

2. 将DOM树和CSSOM树结合，生成渲染树(Render Tree)

3. Layout(回流):根据生成的渲染树，进行回流(Layout)，得到节点的几何信息（位置，大小）

4. Painting(重绘):根据渲染树以及回流得到的几何信息，得到节点的绝对像素

5. Display:将像素发送给GPU，展示在页面上。

1. **回流和重绘概念（\*）?**

可见DOM节点以及它对应的样式结合起来生成渲染树,计算它们在设备视口(viewport)内的确切位置和大小，这个计算的阶段就是回流。

通过构造渲染树和回流阶段，我们知道了哪些节点是可见的，以及可见节点的样式和具体的几何信息(位置、大小)，那么我们就可以将渲染树的每个节点都转换为屏幕上的实际像素，这个阶段就叫做重绘节点

1. **如何减少回流和重绘（\*）?**
2. 合并多次对DOM和样式的修改，然后一次处理掉
3. 元素脱离文档流,进行修改,然后带回文档流

隐藏元素，应用修改，重新显示

使用文档片段(document fragment)在当前DOM之外构建一个子树，再把它拷贝回文档

将原始元素拷贝到一个脱离文档的节点中，修改节点后，再替换原始的元素

1. css3硬件加速
2. **原型和原型链的理解（\*）?**

每创建一个函数，该函数都会自动带有一个prototype属性。该属性是一个指针，指向一个对象，该对象称之为原型对象；原型对象定义的所有属性和方法都会被函数所拥有；

1、构造函数的prototype属性指向 原型对象

2、通过new调用构造函数得到 实例对象

3、实例对象的\_\_proto\_\_属性指向 原型对象

4、实例对象的constructor指向 构造函数

5、原型对象的constructor指向 构造函数

当对象访问属性和方法的时候，会往自身查找，如果没有才会去原型中找。（一级一级传递 形成了原型链）

1. **JS实现继承（\*）?**

|  |
| --- |
| function Person(name, age) {  this.type = 'human'  this.name = name  this.age = age  }  Person.prototype.sayName = function () {  console.log('hello ' + this.name)  }  function Student(name, age) {  Person.call(this, name, age)  }  // 利用原型的特性实现继承  Student.prototype = new Person();  Student.prototype.constructor = Student; |

https://www.cnblogs.com/ranyonsue/p/11201730.html

1. **call、apply、bind的区别（\*\*\*）?**

call 和 apply 特性一样

都是用来调用函数，而且是立即调用

但是可以在调用函数的同时，通过第一个参数指定函数内部 `this` 的指向

call 调用的时候，参数必须以参数列表的形式进行传递，也就是以逗号分隔的方式依次传递即可

apply 调用的时候，参数必须是一个数组，然后在执行的时候，会将数组内部的元素一个一个拿出来，与形参一一对应进行传递

如果第一个参数指定了 `null` 或者 `undefined` 则内部 this 指向 window

bind

可以用来指定内部 this 的指向，然后生成一个改变了 this 指向的新的函数

它和 call、apply 最大的区别是：bind 不会调用

bind 支持传递参数，它的传参方式比较特殊，一共有两个位置可以传递

在 bind 的同时，以参数列表的形式进行传递

在调用的时候，以参数列表的形式进行传递

到底以谁 bind 的时候传递的参数为准呢还是以调用的时候传递的参数为准

两者合并：bind 的时候传递的参数和调用的时候传递的参数会合并到一起，传递到函数内部

1. **XMLHttpRequest的理解?**

- XMLHttpRequest是Ajax的基础。

- 所有现代浏览器均支持XMLHttpRequest对象（IE5和IE6使用ActiveXObject）。

- XMLHttpRequest用于在后台与服务器交换数据。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新

XMLHttpRequest 对象属性:

onreadystatechange 属性存有处理服务器响应的函数。

readyState 属性存有服务器响应的状态信息。每当 readyState 改变时，onreadystatechange 函数就会被执行。

readyState 属性可能的值：

1. 请求未初始化（在调用 open() 之前）
2. 请求已提出（调用 send() 之前）
3. 请求已发送（这里通常可以从响应得到内容头部）
4. 请求处理中（响应中通常有部分数据可用，但是服务器还没有完成响应）
5. 请求已完成（可以访问服务器响应并使用它）

responseText 属性来取回由服务器返回的数据。

XMLHttpRequest 对象方法:

open() 方法:open() 有三个参数。第一个参数定义发送请求所使用的方法，第二个参数规定服务器端脚本的URL，第三个参数规定应当对请求进行异步地处理。

send() 方法将请求送往服务器。

abort() 方法停止当前请求

1. **如何理解隐式迭代和链式编程（\*）?**

在方法的内部会为匹配到的所有元素进行循环遍历，执行相应的方法，而不用我们再进行循环，简化我们的操作，方便我们调用,这就叫做隐式迭代。

链式编程，是将多个操作（多行代码）通过点号"."链接在一起成为一句代码。链式编程的原理是：函数返回的就是调用对象;

1. **$(document).ready() 是个什么函数？为什么要用它（\*）？**

**ready() 函数用于在文档进入ready状态时执行代码**。当DOM 完全加载（例如HTML被完全解析DOM树构建完成时），jQuery允许你执行代码。使用**$(document).ready()的最大好处在于它适用于所有浏览器，jQuery帮你解决了跨浏览器的难题。**

1. **Window.onload和$(document).ready()的区别（\*）?**

JavaScript window.onload 事件和 jQuery ready 函数之间的主要区别是，前者除了要等待 DOM 被创建还要等到包括大型图片、音频、视频在内的所有外部资源都完全加载。如果加载图片和媒体内容花费了大量时间，用户就会感受到定义在 window.onload 事件上的代码在执行时有明显的延迟。

另一方面，jQuery ready() 函数只需对 DOM 树的等待，而无需对图像或外部资源加载的等待，从而执行起来更快。使用 jQuery $(document).ready() 的另一个优势是你可以在网页里多次使用它，浏览器会按它们在 HTML 页面里出现的顺序执行它们，相反对于 onload 技术而言，只能在单一函数里使用。鉴于这个好处，用 jQuery ready() 函数比用 JavaScript window.onload 事件要更好

1. **jQuery 库中的 $() 是什么（\*）？**

**$() 函数是 jQuery() 函数的别称，$() 函数用于将任何对象包裹成 jQuery 对象，接着你就被允许调用定义在 jQuery 对象上的多个不同方法。**你甚至可以将一个选择器字符串传入 $() 函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。

1. **jQuery对象和DOM对象的相互转换（\*）?**

jQuery对象转换为DOM对象: jQuery对象[下标] / jQuery对象.get(下标)

DOM对象转换为jQuery对象: $(DOM对象)

1. **jQuery中width()/height()、innerWidth()/innerHeight()、outerWidth()/outerHeight分别代表什么意思?**

width()/height() 设置或获取内容区域的宽/高

innerWidth()/innerHeight() 获取内部区域(内容+内边距)的宽/高

outerWidth()/outerHeight() 获取外部区域(内容+内边距+边框)的宽/高,如果参数设置为true,也会包含外边距

1. **jQuery里面的attr()和prop()的区别（\*）?**

两者都是用来操作属性的;

1、attr()可以操作标签的自定义属性和固有属性,一般用来操作自定义属性;prop()只能操作标签的固有属性,一般用来操作固有属性;

2、attr()是操作标签的属性; prop()是操作的jQuery对象的属性

3、对于无需设定属性值的属性,比如disabled、selected、checked等; attr()获取的是和属性名一样的值;但是prop()获取布尔值

4、对于标签没有定义的属性,attr()获取的是undefined;prop()获取的是""

1. **分别使用JS和jQuery设置标签属性，以及操作样式、CSS类?**

JS操作:

Element.属性名 获取或设置标签属性

Element.setAttribute() 给标签设置属性

Element.getAttribute() 获取标签属性值

Element.removeAttribute() 移除标签属性值

Element.className 获取或设置标签的css类

Element.style.样式名 获取或设置样式

jQuery操作:

jQuery对象.attr()/prop() 获取或设置标签属性

jQuery对象.addClass() 给标签添加css类

jQuery对象.removeClass() 移除标签css类

jQuery对象.toggleClass() 添加或移除css类

jQuery对象.css() 获取或设置样式

1. **$(this) 和 this 关键字在 jQuery 中有何不同（\*）？**

$(this) 返回一个 jQuery 对象，你可以对它调用多个 jQuery 方法，比如用 text() 获取文本，用val() 获取值等等。

而 this 代表当前元素，它是 JavaScript 关键词中的一个，表示上下文中的当前 DOM 元素。你不能对它调用 jQuery 方法，直到它被 $() 函数包裹，例如 $(this)。

1. **jQuery中remove()和detach()的区别（\*）?**

共同点: 都是从从DOM中删除所有匹配的元素。

区别点:

remove方法不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。但除了这个元素本身得以保留之外，其他的比如绑定的事件，附加的数据等都会被移除。

detach方法不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。与remove()不同的是，所有绑定的事件、附加的数据等都会保留下来。

1. **jQuery 筛选包含哪些?**

过滤、查找、串联

1. **jQuery过滤和查找的区别?**

过滤得到的元素都是当前jQuery对象的子集

查找得到的元素可能是当前jQuery对象的子集也可以不是;

1. **jQuery的on()和bind()的区别?**

两者都可以用来绑定事件及事件处理函数

on()还可以实现事件委托,但是bind()不行

1. **jQuery中trigger和triggerHandler的区别?**

trigger和triggerHandler都是用来触发事件;

Trigger会触发浏览器的默认行为，会执行事件冒泡

triggerHandler不会触发浏览器的默认行为，不会执行事件冒泡

1. **如何定义jQuery插件（\*）?**

|  |
| --- |
| /基于jQuery插件机制定义的插件              $.fn.extend({                  setRedColor:function(){                      this.css("background","red");                  },                  setColor:function(e){                      this.css("background",e);                  }              })              // $("#div1").setRedColor();              $("#div1").setColor("yellow");  jQuery.fn.extend({  // 注意:如果我们定义的插件没有写return,那么就没有办法继续调用其他的jQuery方法了  // 为了更好的使用jQuery的链式编程,我们定义的插件最好也把jQuery对象给return  check: function () {  return this.each(function () {  this.checked = true;  });  },  uncheck: function () {  return this.each(function () {  this.checked = false;  });  },  invert: function () {  return this.each(function () {  this.checked = !this.checked;  });  }  }); |

1. **$.get() 和 $.ajax() 方法之间的区别是什么（\*）?**

$.ajax() 方法更强大，更具可配置性, 让你可以指定等待多久，以及如何处理错误。$.get()/$.post() 方法是一个只获取一些数据的专门化方法。

1. **HTTP请求方式（\*）？**

GET是最常用的方法，通常用于请求服务器发送某个资源。

POST 方法，常用来创建一个新资源。

PUT 方法，常用来更新已有资源，若资源不存在，它也会进行创建。

PATCH是一个新方法，可以当作是PUT方法的补充，主要用来做局部更新。

DELETE 方法，常用来删除已有资源

1. **meta viewport 是做什么的？怎么写（\*）?**

移动端浏览器通常都在一个比屏幕更宽的虚拟窗口中渲染页面，这个虚拟窗口就是viewport，目的是正常展示没有做移动端适配的网页，可以让他们完整的展现给用户。我们有时用移动设备访问桌面版网页就会看到一个横向滚动条，这里可显示区域的宽度就是viewport的宽度。

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

该meta标签的作用是让当前viewport的宽度等于设备的宽度，同时不允许用户手动缩放。当然maximum-scale=1.0, user-scalable=0不是必需的，是否允许用户手动播放根据网站的需求来定，但把width设为width-device基本是必须的，这样能保证不会出现横向滚动条。

meta viewport 的6个属性：

|  |  |
| --- | --- |
| width | 设置layout viewport 的宽度，为一个正整数，或字符串”width-device” |
| initial-scale | 设置页面的初始缩放值，为一个数字，可以带小数 |
| minimum-scale | 允许用户的最小缩放值，为一个数字，可以带小数 |
| maximum-scale | 允许用户的最大缩放值，为一个数字，可以带小数 |
| height | 设置layout viewport 的高度，这个属性并不重要，很少使用 |
| user-scalable | 是否允许用户进行缩放，值为”no”或”yes”, no 代表不允许，yes代表允许 |

1. **在做响应式页面的时候,有哪几种设备,分别的阀值是什么（\*）？**

四种设备:

超大屏幕,阀值大于等于1200px

中等屏幕,阀值大于等于992px和小于1200px

小屏幕,阀值大于等于768px和小于992px

超小屏幕,阀值小于768px

1. **Bootstrap .container容器的阈值以及固定宽度是?**

阈值 固定宽度

>=1200 1170

>=992&<1200 970

>=768 750

<768 100%

1. **什么是Bootstrap？以及为什么要使Bootstrap？**

Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。 Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的。Bootstrap具有移动设备优先、浏览器支持良好、容易上手、响应式设计等优点，所以Bootstrap被广泛应用。

1. **使用Bootstrap时，要声明的文档类型是什么？以及为什么要这样声明？**

**使用Bootstrap时，需要使用 HTML5 文档类型（Doctype）**。<!DOCTYPE html>因为Bootstrap 使用了一些 HTML5 元素和 CSS 属性，如果在 Bootstrap 创建的网页开头不使用 HTML5 的文档类型（Doctype），可能会面临一些浏览器显示不一致的问题，甚至可能面临一些特定情境下的不一致，以致于代码不能通过 W3C 标准的验证。

1. **.container 和 .container-fluid区别（\*）？**

.container 类用于固定宽度并支持响应式布局的容器。

.container-fluid 类用于 100% 宽度，占据全部视口（viewport）的容器。

1. **Bootstrap 网格系统（Grid System）的工作原理？**

（1）行必须放置在 .container class 内，以便获得适当的对齐（alignment）和内边距（padding）。

（2）使用行来创建列的水平组。

（3）内容应该放置在列内，且唯有列可以是行的直接子元素。

（4）预定义的网格类，比如 .row 和 .col-xs-4，可用于快速创建网格布局。LESS 混合类可用于更多语义布局。

（5）列通过内边距（padding）来创建列内容之间的间隙。该内边距是通过 .rows 上的外边距（margin）取负，表示第一列和最后一列的行偏移。

（6）网格系统是通过指定您想要横跨的十二个可用的列来创建的。例如，要创建三个相等的列，则使用三个 .col-xs-4。

1. **Bootstrap 网格系统列与列之间的间隙宽度是多少（\*）？**

间隙宽度为30px（一个列的每边分别是15px）。

1. **如果通过CSS实现三角符号?**

|  |
| --- |
| /\* 三角符号的实现 \*/  span {  display: inline-block;  width: 0;  height: 0;  /\* 下三角 \*/  /\* border-top: 5px solid black; \*/  /\* 上三角 \*/  border-bottom: 5px solid black;  border-left: 5px solid transparent;  border-right: 5px solid transparent;  }  /\*具备兼容性的三角符号实现\*/  #wrap .triangle {  width: 0;  height: 0;  border-width: 20px;  border-style: dashed dashed solid dashed;  border-color: transparent transparent red transparent;  /\* 大部分ie的布局拉伸问题,都直接添加overflow:hidden \*/  overflow: hidden;  } |

1. **layui 与 layuiAdmin 有什么区别？**

layui 是一款开源免费的前端 UI 框架，遵循 MIT 协议，可随意使用。

layuiAdmin 是完全基于 layui 搭建的一款通用型后台管理模板系统，由 layui 官方出品，需购买授权使用。

1. **Layui数据表格的用法（\*）？**

1、使用layui.use()加载table模块

2、使用table.render()渲染，render()中需要指定一系列参数：

2.1 elem: '绑定的表格元素'

2.2 height: 表格的高度

2.3 url： 请求数据的url

2.4 page： 分页相关设置（对象写法），每页显示的条目数limit，每页显示条目数列表limits，首页链接的的名称first，尾页链接的名称last，分页布局的顺序layout

2.5 toolbar: 是否开启工具条

2.6 cols: 表头设置

2.7 parseData：对响应数据的解析

2.8 request： 请求名称的变更

3、监听工具条

table.on('tool(数据表格id)', function(){})

1. **你用过Layui吗？都是怎么用的（\*）**

用过，直接使用开源的后端管理系统进行修改使用

1. **原生js实现图片懒加载原理（\*）?**

背景：页面图片多，加载的图片就多。服务器压力就会很大。不仅影响渲染速度还会浪费带宽。比如一个1M大小的图片，并发情况下，达到1000并发，即同时有1000个人访问，就会产生1个G的带宽。为了解决以上问题，提高用户体验，就出现了懒加载方式来减轻服务器的压力，优先加载可视区域的内容，其他部分等进入了可视区域再加载，从而提高性能。

懒加载原理

一张图片就是一个<img>标签，浏览器是否发起请求图片是根据<img>的src属性，所以实现懒加载的关键就是：

在图片没有进入可视区域时，先不给<img>的src赋值，这样浏览器就不会发送请求了，等到图片进入可视区域再给src赋值。

懒加载思路及实现

实现懒加载有四个步骤，如下：

1.加载loading图片

2.判断哪些图片要加载【重点】

3.隐形加载图片

4.替换真图片

More:<https://www.cnblogs.com/cat-eol/p/11747362.html>

1. **什么是版本控制?**

版本控制是指对软件开发过程中各种程序代码、配置文件及说明文档等文件变更的管理，是软件配置管理的核心思想之一。

简而言之：版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统.

1. **版本控制工具的两大核心作用（\*）?**

追踪文件的变更(历史记录)

并行开发

检入检出控制

1. **什么是单点故障?**

单点故障（英语：single point of failure，缩写SPOF）是指系统中一点失效，就会让整个系统无法运作的部件，换句话说，单点故障即会整体故障。

1. **集中式版本控制工具的缺点（\*）?**

服务器宕机,代码没有办法提交,代码没有办法得到保障

服务器挂了（硬盘坏了）,所有的历史记录和代码都没了，代码在客户端可以拿到，但是历史记录肯定是没了;

1. **分布式版本控制工具的优点（\*）?**

可以解决集中式版本控制工具的缺点

分布式版本控制工具可以完美解决上述的两个问题;

因为存在本地仓库,所以服务器宕机,也可以正常提交代码

因为本地仓库是中央仓库的完整备份,所以服务器挂了,本地仓库即包含了代码,也包含了历史记录;

1. **Git三大分区（\*）?**

Git 本地数据管理，大概可以分为三个区，工作区,暂存区和版本库。

工作区（Working Directory）:直接编辑的地方，肉眼可见，直接操作。

暂存区（Stage）:数据暂时存放的区域。在项目目录下使用指令“git add fileName”就可以把代码存入缓存区，这里的fileName是指的要往缓存区添加的文件名，当然也可以使用“.”代表当前的整个目录。

版本库（commit History）:存放已经提交的数据，push 的时候，就是把这个区的数据 push 到远程git仓库了。而.git这个目录就是本地仓库区，你commit的一切更改都会存储到本地仓库中

1. **Git本地库与远程库（\*）？**

git本地库，适合个人在本地使用版本管理工具来管理自己的资料

git远程库，适合团队的协作，成员之间共享资料

1. **Git三种状态(已跟踪文件的三种状态)（\*）?**

工作目录下面的所有文件都不外乎这两种状态：已跟踪或未跟踪。

已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是**已提交，已修改或者已暂存**。

已提交（committed）：已提交表示数据已经安全的保存在本地数据库中。

已修改（modified）：已修改表示修改了文件，但还没保存到数据库中。

已暂存（staged）： 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

1. **Git三大对象（\*）?**

git对象：代表的是文件的内容 （blob）

tree对象：代表的是项目版本的快照 （tree）

commit对象：代表的是对tree的简单包装,里面包含了tree对象、父提交对象、以及作者、提交者、提交时间、注释信息等 （commit）

1. **git add 命令到底做了什么事情（\*）?**

简单来说：把文件或目录下面的所有文件都放到暂存区

底层：把文件内容以键值的方式存储在本地库中，键是hash，值是git对象；然后把对应的git对象拉到暂存区以指定文件来存储;

1. **Git快进合并和典型合并（\*）?**

快进合并：当你试图合并两个分支时，如果顺着一个分支走下去能够到达另一个分支，那么 Git在合并两者的时候，只会简单的将指针向前推进（指针右移），因为这种情况下的合并操作没有需要解决的分歧——这就叫做 “快进（fast-forward）。【如果可以从一条线顺下来,就是快进合并】

典型合并：你的开发历史从一个更早的地方开始分叉开来（diverged ），分支A所在提交并不是分支B所在提交的直接祖先，Git不得不做一些额外的工作。出现这种情况的时候，Git会使用两个分支的末端所指的快照以及这两个分支的工作祖先，做一个简单的三方合并。【如果不可以从一条线顺下来,就是典型合并】

1. **分支的本质（\*）?**

Git的分支，其实本质上仅仅是**指向提交对象的可变指针**。

1. **git checkout [branch]与运行 git reset --hard [commithash]（\*\*\*）?**

首先不同于 reset --hard，checkout对工作目录是安全的，它会通过检查来确保不会将已更改的文件弄丢。而 reset --hard则会不做检查就全面地替换所有东西。

第二个重要的区别是如何更新 HEAD。reset会移动HEAD分支的指向，而checkout只会移动HEAD自身来指向另一个分支。

1. **Git特点(比如：git存储版本的方式)（\*）?**

直接记录项目快照，而非差异比较

近乎所有操作都是本地执行

时刻保持数据完整性

多数操作仅添加数据

文件的三种状态

1. **列举工作中常用的几个git命令（\*\*\*）？**

新增文件的命令：git add file或者git add ./

提交文件的命令：git commit –m或者git commit –a

查看工作区状况：git status

拉取合并远程分支的操作：git fetch/git merge或者git pull

查看提交记录命令：git reflog

切换分支: git checkout 分支名

创建分支: git branch 分支名

创建分支并切换: git checkout -b 分支名

提交文件到远程库：git push 库别名 分支

1. **如果本次提交误操作，如何撤销 - reset三部曲（\*\*\*）？**

如果想撤销提交到索引（暂存）区的文件，可以通过git restore --staged 文件名 或 git reset HEAD file；

如果想撤销提交到本地仓库的文件，可以通过git reset –soft HEAD~[^n]恢复当前分支的版本库至上一次提交的状态，索引区和工作空间不变更；

可以通过git reset –mixed HEAD~[^n]恢复当前分支的版本库和索引区至上一次提交的状态，工作区不变更；

可以通过git reset –hard HEAD~[^n]恢复当前分支的版本库、索引区和工作空间至上一次提交的状态。

1. **提交时发生冲突，你能解释冲突是如何产生的吗？你是如何解决的（\*\*\*）？**

开发过程中，我们都有自己的特性分支，所以冲突发生的并不多，但也碰到过。诸如公共类的公共方法，我和别人同时修改同一个文件，他提交后我再提交就会报冲突的错误。

发生冲突，在IDE里面一般都是对比本地文件和远程分支的文件，然后把远程分支上文件的内容手工修改到本地文件，然后再提交冲突的文件使其保证与远程分支的文件一致，这样才会消除冲突，然后再提交自己修改的部分。特别要注意下，修改本地冲突文件使其与远程仓库的文件保持一致后，需要提交后才能消除冲突，否则无法继续提交。必要时可与同事交流，消除冲突。

发生冲突，也可以使用命令。

通过git stash命令，把工作区的修改提交到栈区，目的是保存工作区的修改；

通过git pull命令，拉取远程分支上的代码并合并到本地分支，目的是消除冲突；

通过git stash pop命令，把保存在栈区的修改部分合并到最新的工作空间中；

1. **如果我想修改提交的历史信息，应该用什么命令？**

如果修改最近一次提交的历史记录，就可以用git commit –amend命令；vim编辑的方式；

1. **使用过git merge和git rebase吗？它们之间有什么区别（\*\*\*）？**

git merge和git rebase都是合并分支的命令。

git merge branch会把branch分支的差异内容pull到本地，然后与本地分支的内容一并形成一个committer对象提交到主分支上，合并后的分支与主分支一致；

git rebase branch会把branch分支优先合并到主分支，然后把本地分支的commit放到主分支后面，合并后的分支就好像从合并后主分支又拉了一个分支一样，本地分支本身不会保留提交历史。

1. **你使用过git stash命令吗？你一般什么情况下会使用它（\*\*\*）？**

命令git stash是把工作区修改的内容存储在栈区。

以下几种情况会使用到它：

解决冲突文件时，会先执行git stash，然后解决冲突；

遇到紧急开发任务但目前任务不能提交时，会先执行git stash，然后进行紧急任务的开发，然后通过git stash pop取出栈区的内容继续开发；

切换分支时，当前工作空间内容不能提交时，会先执行git stash再进行分支切换；

1. **能不能说一下git fetch和git pull命令之间的区别（\*\*\*）？**

git fetch branch是把名为branch的远程分支拉取到本地；而git pull branch是在fetch的基础上，把branch分支与当前分支进行merge；因此pull = fetch + merge。

1. **能说一下git系统中HEAD、工作树和索引之间的区别吗（\*）？**

HEAD文件包含当前分支的引用（指针）；

工作树是把当前分支检出到工作空间后形成的目录树，一般的开发工作都会基于工作树进行；

索引index文件是对工作树进行代码修改后，通过add命令更新索引文件；GIT系统通过索引index文件生成tree对象；

1. **Git和SVN有什么区别（\*）？**

| ****Git**** | ****SVN**** |
| --- | --- |
| 1. Git是一个分布式的版本控制工具 | 1. SVN 是集中版本控制工具 |
| 2.它属于第3代版本控制工具 | 2.它属于第2代版本控制工具 |
| 3.客户端可以在其本地系统上克隆整个存储库 | 3.版本历史记录存储在服务器端存储库中 |
| 4.即使离线也可以提交 | 4.只允许在线提交 |
| 5.Push/pull 操作更快 | 5.Push/pull 操作较慢 |
| 6.工程可以用 commit 自动共享 | 6.没有任何东西自动共享 |

1. **我们在本地工程常会修改一些配置文件，这些文件不需要被提交，而我们又不想每次执行git status时都让这些文件显示出来，我们该如何操作（\*\*\*）？**

在.git/info/创建.gitignore(目前自带有exclude文件)

然后往文件中添加需要忽略哪些文件夹下的什么类型的文件

1. **提交对象包含什么（\*）？**

Commit 对象包含以下组件，你应该提到以下这三点：

一组文件(树对象)，表示给定时间点的项目状态

引用父提交对象

SHAI 名称，一个40个字符的字符串，提交对象的唯一标识。

作者、提交者、提交注释

1. **如果分支是否已合并为master，你可以通过什么手段知道(\*)？**

要知道某个分支是否已合并为master，你可以使用以下命令：

git branch --merged 它列出了已合并到当前分支的分支。

git branch --no--merged 它列出了尚未合并的分支。

1. **cookies，sessionStorage和localStorage的区别（\*\*\*）？**

共同点：都是保存在浏览器端，且是同源的。

区别：

cookies是为了标识用户身份而存储在用户本地终端上的数据，始终在同源http请求中携带，即cookies在浏览器和服务器间来回传递，而sessionstorage和localstorage不会自动把数据发给服务器，仅在本地保存。

存储大小的限制不同。cookie保存的数据很小，不能超过4k，而sessionstorage和localstorage保存的数据大，可达到5M。

数据的有效期不同。cookie在设置的cookie过期时间之前一直有效，即使窗口或者浏览器关闭。sessionstorage仅在浏览器窗口关闭之前有效。localstorage始终有效，窗口和浏览器关闭也一直保存，用作长久数据保存。

作用域不同。cookie在所有的同源窗口都是共享；sessionstorage不在不同的浏览器共享，即使同一页面；localstorage在所有同源窗口都是共享

1. **如何实现浏览器内多个标签页之间的通信(\*)？**

调用localstorge、cookies等本地存储方式

1. **简单阐述jsonServer（\*）？**

一个在前端本地运行，可以存储json数据的server。

1. **ajax概述以及ajax语法（\*\*\*）？**

Ajax 即“Asynchronous Javascript And XML”（异步 JavaScript 和 XML），是指一种创建交互式网页应用的网页开发技术。

Ajax 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。

**$.ajax([settings]) 通过 HTTP 请求加载远程数据。jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等**。

**url:一个用来包含发送请求的URL字符串。**

settings:AJAX 请求设置。所有选项都是可选的。

async: (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。

contentType: (默认: "application/x-www-form-urlencoded") 发送信息至服务器时内容编码类型。

**data: Object,String;发送到服务器的数据。**

**dataType:预期服务器返回的数据类型。**

"json": 返回 JSON 数据 。

"jsonp": JSONP 格式。使用 JSONP 形式调用函数时

**success(data):求成功后的回调函数。**

**type:(默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET"。注意：其它 HTTP 请求方法，如 PUT 和 DELETE 也可以使用，但仅部分浏览器支持**。

**url:(默认: 当前页地址) 发送请求的地址。**

**语法:**

**$.ajax({**

**url:”http://localhost:3004/friues”,**

**type:”GET”,**

**data:{key:value}**

**dataType:”jsonp”,**

**success:function(res){**

**}**

**})**

1. **如何解决跨域问题（\*\*\*）？**

跨域是指一个域下的文档或脚本试图去请求另一个域下的资源，这里跨域是广义的。

Jsonp方式：

jsonp的原理就是利用<script>标签没有跨域限制，通过<script>标签src属性，发送带有callback参数的GET请求，服务端将接口返回数据拼凑到callback函数中，返回给浏览器，浏览器解析执行，从而前端拿到callback函数返回的数据。

使用Access-Control-Allow-Origin解决跨域:

在服务器端设置响应头:

response.setHeader("Access-Control-Allow-Origin",允许跨域的资源);

nginx代理跨域：

nginx代理跨域，实质和CORS跨域原理一样，通过配置文件设置请求响应头Access-Control-Allow-Origin…等字段。

<https://www.jianshu.com/p/ad799a7801e1>

<https://segmentfault.com/a/1190000011145364>

<https://www.cnblogs.com/sdcs/p/8484905.html>

<https://www.cnblogs.com/itmacy/p/6958181.html>

1. **什么是同源策略（\*）？**

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。

1. **如何进行网站性能优化？**

减少 HTTP 请求：CSS 精灵、inline Image

减少 DOM 元素数量

减少 iframe 数量

非必须组件延迟加载

将样式表放到页面顶部

不使用@import

将脚本放到页面底部

将 javascript 和 css 从外部引入

压缩 javascript 和 css

减少 DOM 访问

合理设计事件监听器

1. **CSS实现垂直水平居中?**

1、使用transforms来实现垂直水平居中（不兼容老版本浏览器）：

|  |
| --- |
| <style>  \* {  padding: 0;  margin: 0;  }  #wrap {  width: 300px;  height: 300px;  border: 1px solid black;  margin: 0 auto;  }  .inner {  width: 100px;  height: 100px;  background: yellow;  margin: 50% auto 0;  transform: translateY(-50%);  }  </style>  <div id="wrap">  <div class="inner">  </div>  </div> |

2、通过定位实现（兼容老版本浏览器）

|  |
| --- |
| #wrap {  width: 300px;  height: 300px;  border: 1px solid black;  margin: 0 auto;  position: relative;  }  .inner {  width: 100px;  height: 100px;  background: yellow;  position: absolute;  left: 0;  right: 0;  top: 0;  bottom: 0;  margin: auto;  }  <div id="wrap">  <div class="inner">  </div>  </div> |

1. **什么是Less?**

LESS是一个CSS预处理器，可以为网站启用可自定义，可管理和可重用的样式表。

LESS是一种动态样式表语言，扩展了CSS的功能。 LESS也是跨浏览器友好。

1. **什么是Sass？**

SASS（Syntactically Awesome Stylesheet）是一个CSS预处理器，有助于减少CSS的重复，节省时间。它是更稳定和强大的CSS扩展语言，描述文档的样式干净和结构。

1. **什么是 CSS 预处理器呢？**

CSS预处理器是一种脚本语言，可扩展CSS并将其编译为常规CSS语法，以便可以通过Web浏览器读取。 它提供诸如变量，函数， mixins 和操作等功能，可以构建动态CSS。

1. **Less/Sass如何声明变量？**

LESS允许使用 @ 符号定义变量。 变量分配使用冒号(:)完成。

LESS中声明变量的格式为“@变量名：变量值”。

Sass使用$符号来标识变量。变量分配使用冒号(:)完成。

1. **什么是Mixin?**

混合类似于编程语言中的函数。 Mixins是一组CSS属性，允许您将一个类的属性用于另一个类，并且包含类名作为其属性。 在LESS中，可以使用类或id选择器以与CSS样式相同的方式声明mixin。 它可以存储多个值，并且可以在必要时在代码中重复使用。

1. **Mixin的形式?**

普通混合

不输出混合

带参数的混合

带参数且有默认值的混合

命名参数

匹配模式

等

1. **定义带参数且与默认值的混合?**

Less:

|  |
| --- |
| .juzhong(@w:10px,@h:10px,@c:yellow){  position: absolute;  left: 0;  right: 0;  top: 0;  bottom: 0;  margin: auto;  background: @c;  height: @h;  width: @w;  } |

Sass:

|  |
| --- |
| @mixin juzhong($w:10px,$h:10px,$c:yellow){  position: absolute;  left: 0;  right: 0;  top: 0;  bottom: 0;  margin: auto;  background: $c;  height: $h;  width: $w;  } |

1. **sass,less,stylus优缺点？**

Less 从语言特性的设计到功能的健壮程度和另外两者相比都有一些缺陷，但用 Less 可以满足大多数场景的需求。

但相比另外两者，基于 Less 开发类库会复杂得多，实现的代码会比较脏，能实现的功能也会受到 DSL 的制约。

比 Stylus 语义更清晰、比 Sass 更接近 CSS 语法，使得刚刚转用 CSS 预编译的开发者能够更平滑地进行切换。

Sass 在三者之中历史最久，也吸收了其他两者的一些优点。

从功能上来说 Sass 大而全，语义明晰但是代码很容易显得累赘。

主项目基于 Ruby 可能也是一部分人不选择它的理由（Less 开始也是基于 Ruby 开发，后来逐渐转到 less.js 项目中）。

Sass 有一个「事实标准」库——Compass，于是对于很多开发者而言省去了选择类库的烦恼，对于提升开发效率也有不小的帮助。

Stylus 的语法非常灵活，很多语义都是根据上下文隐含的。

基于 Stylus 可以写出非常简洁的代码，但对使用团队的开发素养要求也更高，更需要有良好的开发规范或约定。

总的来说，三种预处理器百分之七八十的功能是类似的。

Less 适合帮助团队更快地上手预处理代码的开发，而 Sass 和 Stylus 的差异更在于口味。

1. **为什么要使用CSS预处理器？**

作为前端开发人员，大家都知道，Js中可以自定义变量，而CSS仅仅是一个标记语言，不是编程语言，因此不可以自定义变量，不可以引用等等。

CSS有具体以下几个缺点：

  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。

  这就导致了我们在工作中无端增加了许多工作量。而使用CSS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。大大提高了我们的开发效率。

  但是，CSS预处理器也不是万金油，CSS的好处在于简便、随时随地被使用和调试。预编译CSS步骤的加入，让我们开发工作流中多了一个环节，调试也变得更麻烦了。更大的问题在于，预编译很容易造成后代选择器的滥用。

  所以我们在实际项目中衡量预编译方案时，还是得想想，比起带来的额外维护开销，CSS预处理器有没有解决更大的麻烦。

1. **Sass和Less的比较？**

相同点：

1、混入(Mixins)——class中的class；

2、参数混入——可以传递参数的class，就像函数一样；

3、嵌套规则——Class中嵌套class，从而减少重复的代码；

4、运算——CSS中用上数学；

5、颜色功能——可以编辑颜色；

6、名字空间(namespace)——分组样式，从而可以被调用；

7、作用域——局部修改样式；

8、JavaScript 赋值——在CSS中使用JavaScript表达式赋值。

区别点：

Less是基于JavaScript，是在客户端处理的。

Sass是基于Ruby的，是在服务器端处理的。

关于变量在Less和Sass中的唯一区别就是Less用@，Sass用$。

1. **列出SASS支持的两种语法？**

SASS支持两种语法，即 SCSS 和缩进语法。

SCSS（Sassy CSS）是CSS语法的扩展，可以更容易地维护大型样式表，并且可以识别供应商特定的语法和许多CSS。 SCSS文件使用扩展名 .scss 。

缩进是较旧的语法，有时仅称为 Sass 。 使用这种形式的语法，可以简洁地编写CSS。 SASS文件使用扩展名 .sass 。

1. **解释使用Sass @import函数？**

它用于扩展CSS导入规则。 为此需要启用Sass和SCSS文件的导入。

它可以将所有导入的文件合并为一个输出的CSS文件。

1. **Sass有哪四种输出格式，分别说说它们的特点？**

1：嵌套格式 :nested格式

该格式能清晰的反应CSS与HTML的结构关系，选择器和属性等单独占一行，缩进量与Sass文件一致。

2：展开格式 :expanded格式

选择器和属性等单独占一行，属性根据选择器缩进，而选择器不做任何缩进。

3：紧凑格式 ：compact格式

这种格式比上边两种格式都要节省空间，每条CSS规则只占用一行，嵌套过的选择器之间是没有隔行的，没有嵌套过的选择器会一行留白。

4：压缩格式 :compressed格式

Compressed 输出方式删除所有无意义的空格、空白行、以及注释，力求将文件体积压缩到最小，同时也会做出其他调整，比如会自动替换占用空间最小的颜色表达方式。

1. **进程和线程的概念?**

进程是一个应用程序的运行实例，是一个动态抽象概念，竞争计算机系统资源的基本单位。

程序运行时系统就会创建一个进程，并为它分配资源，然后把该进程放入进程就绪队列，进程调度器选中它的时候就会为它分配CPU时间，程序开始真正运行。

线程是程序执行时的最小单位，它是进程的一个执行流，是CPU调度和分派的基本单位。一个进程可以由很多个线程组成，线程间共享进程的所有资源，每个线程有自己的堆栈和局部变量。

线程由CPU独立调度执行，在多CPU环境下就允许多个线程同时运行。同样多线程也可以实现并发操作，每个请求分配一个线程来处理。

**一个程序至少一个进程，一个进程至少一个线程(进程和线程的关系)**。

1. **进程与线程的区别?**

1. 线程是程序(进程)执行的最小单位，而进程是操作系统分配资源的最小单位；

2. 一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线

3. 进程之间相互独立，但同一进程下的各个线程之间共享程序的内存空间(包括代码段，数据集，堆等)及一些进程级的资源(如打开文件和信号等)，某进程内的线程在其他进程不可见；

4. 调度和切换：线程上下文切换比进程上下文切换要快得多

1. **JavaScript是单线程还是多线程?**

JavaScript是单线程；

JavaScript的单线程，与它的用途有关。作为浏览器脚本语言，JavaScript的主要用途是与用户互动，以及操作DOM。这决定了它只能是单线程，否则会带来很复杂的同步问题。

1. **NodeJS概述?**

Node.js是一个能够在服务器端运行JavaScript的开放源代码、跨平台JavaScript运行环境。

1. **为什么使用NodeJS?**

nodejs会让我们的编程工作变得简单，它主要包含如下几点几个好处:

执行快速。

永远不会阻滞。

JavaScript是通用的编程语言。

异步处理机制。

避免并行所带来的问题。

1. **NodeJS特点?**

是单线程的，但是有很高的可扩展性，使用JavaScript作为主流编程语言。使用的是异步处理机制和事件驱动。处理高效。

1. **为什么NodeJS是单线程的？**

Nodejs使用的是单线程没错，但是通过异步处理的方式，可以处理大量的数据吞吐量，从而有更好的性能和扩可扩展性。

1. **什么叫做回调地狱?**

回调地狱是由嵌套的回调函数导致的。这样的机制会导致有些函数无法到达，并且很难维护。

1. **如何避免回调地狱？**

模块化：将回调函数转换为独立的函数

使用Promise

使用aync/await （暂且没讲）

1. **NodeJS API函数的类型有哪些?**

有两种，

一种是同步（阻滞型）函数。阻滞型函数会等待操作完成以后再进行下一步。

另外一种是异步（非阻滞型）函数。这种函数使用回调函数来处理当前函数获取的结果。

1. **NodeJS回调函数的第1个参数是什么?**

通常是错误对象。如果这个参数为空，表示没有错误。

1. **什么是错误优先的回调函数？**

错误优先的回调函数用于传递错误和数据。第一个参数始终应该是一个错误对象， 用于检查程序是否发生了错误。其余的参数用于传递数据。例如：

fs.readFile(filePath, function(err, data) {

if (err) {

//handle the error

}

// use the data object

});

解析：这个题目的主要作用在于检查被面试者对于Node中异步操作的一些基本知识的掌握。

1. **模块化的概念？**

模块化是一种处理复杂系统分解为更好的可管理模块的方式。每个模块完成一个特定的子功能，所有的模块按某种方法组装起来，成为一个整体，完成整个系统所要求的功能。

1. **NodeJS中模块定义、模块引用、模块标识的理解?**

**模块定义:**

在Node中一个文件就是一个模块。默认情况下在js文件中编写的内容，都是运行在一个独立的函数中,那么在模块中定义的函数或变量,就是私有的;

在运行环境中，提供了exports对象用于导出当前模块的函数或者变量，并且它是唯一的导出的出口。

在模块中还存在一个module对象，它代表模块自身，而exports是module的属性。

**模块引用:**

在规范中，定义了require()方法，这个方法接受模块标识，以此将一个模块引入到当前运行环境中。

require()方法返回的代表引入模块的模块对象,我们可以通过这个模块对象使用暴露出来的变量或者函数;

**模块标识:**

模块的标识就是模块的名字或路径，也就是传递给require()方法的参数，它必须是符合驼峰命名法的字符串，或者是以.、..开头的相对路径、或者绝对路径。

1. **Node中exports和module.exports的区别?**

如果使用exports,来暴露变量或函数,那么只能通过

exports.变量名 = 值;

exports.函数名 = function(){}

但是使用module.exports,来暴露变量或函数,那么可以通过如下方式:

方式一:

module.exports.变量名 = 值;

module.exports.函数名 = function(){}

方式二:

module.exports.变量名 = 值;

module.exports = {

变量名: 值;

函数名: function(){}

}

**推荐：不要省略module**

1. **NPM是什么?**

npm全称为Node Package Manager，是一个基于Node.js的包管理器，也是整个Node.js社区最流行、支持的第三方模块最多的包管理器。为了让JavaScript开发人员更容易分享和重用代码;

允许用户从NPM服务器下载别人编写的第三方包到本地使用。

允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。

允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

1. **加载模块时，为什么每个模块都有\_\_dirname,\_\_filename属性呢，这两个属性是从哪里来的？**

module模块相当于被包装了，包装形式如下

(function (exports, require, module, \_\_filename, \_\_dirname) {

});

也就是说，每个module里面都会传入\_\_filename, \_\_dirname参数，这两个参数并不是module本身就有的，是外界传入的

1. **node导出模块有两种方式，一种是exports.xxx=xxx和module.exports={}有什么区别吗？**

通过exports只能使用.的方式来向外暴露内部变量

exports.xxx = xxx

module.exports既可以通过.的形式，也可以直接赋值

module.exports.xxx = xxxx

module.exports = {}

因为module.exports.xxx或者exports.xxx都没有改变exports的指向;

但是exports={},改变了exports的指向;

1. **setImmediate()和setTimeout() 区别在哪里?**

setImmediate就是马上执行的意思。setTimeout, 时间参数传为0，也想获得同样的功能。只不过前者要快一些。

1. **nodejs和ajax的区别是什么?**

Nodejs和ajax也就是asynchronous JavaScript and xml，都是通过JavaScript来表现的，但是他们的目的截然不同。

Ajax是设计用来动态的更新页面的某个区域，从而不需要更新整个页面。

Nodejs是用来开发客户服务器类型应用的。

1. **V8的内存限制是多少，为什么V8这样设计？**

64位系统下是1.4GB， 32位系统下是0.7GB。因为1.5GB的垃圾回收堆内存，V8需要花费50毫秒以上，做一次非增量式的垃圾回收甚至要1秒以上。这是垃圾回收中引起Javascript线程暂停执行的事件，在这样的花销下，应用的性能和影响力都会直线下降。

1. **新建Buffer会占用V8分配的内存吗？**

不会，Buffer属于堆外内存，不是V8分配的。

1. **Buffer.alloc和Buffer.allocUnsafe的区别？**

Buffer.allocUnsafe创建的 Buffer 实例的底层内存是未初始化的。 新创建的 Buffer 的内容是未知的，可能包含敏感数据。 使用 Buffer.alloc() 可以创建以零初始化的 Buffer 实例。

1. **express项目的目录大致是什么样子的？**

参照Express生成器目录：

app.js, package.json, bin/www, public, routes, views.

1. **express常用函数？**

express.Router()路由组件

app.get/post/all()路由定向

app.configure()配置

app.set()设定参数

app.use()使用中间件

1. **express中如何获取路由的参数?**

获取RESTful风格路由参数：

Route path: /users/:userId/books/:bookId

Request URL: http://localhost:3000/users/34/books/8989

req.params: { "userId": "34", "bookId": "8989" }

获取Get请求参数：

req.query

获取Post请求参数：

req.body

1. **express response有哪些常用方法?**

res.download() 弹出文件下载

res.end() 结束response

res.json() 返回json

res.jsonp() 返回jsonp

res.redirect() 重定向请求

res.render() 渲染模板

res.send() 返回多种形式数据

res.sendFile 返回文件

res.sendStatus() 返回状态

1. **什么是MongoDB？**

MongoDB是一个文档数据库，提供好的性能，领先的非关系型数据库。采用BSON存储文档数据。

BSON（）是一种类json的一种二进制形式的存储格式，简称Binary JSON.

相对于json多了date类型和二进制数组。

1. **MongoDB的优势有哪些？**

面向文档的存储：以 JSON 格式的文档保存数据。

任何属性都可以建立索引。

复制以及高可扩展性。

自动分片。

丰富的查询功能。

快速的即时更新。

1. **什么是数据库？**

数据库可以看成是一个电子化的文件柜,用户可以对文件中的数据运行新增、检索、更新、删除等操作。数据库是一个

所有集合的容器，在文件系统中每一个数据库都有一个相关的物理文件。

1. **什么是集合(表)？**

集合就是一组 MongoDB 文档。它相当于关系型数据库（RDBMS）中的表这种概念。集合位于单独的一个数据库中。

一个集合内的多个文档可以有多个不同的字段。一般来说，集合中的文档都有着相同或相关的目的。

1. **什么是文档(记录)？**

文档由一组key value组成。文档是动态模式,这意味着同一集合里的文档不需要有相同的字段和结构。在关系型数据库中table中的每一条记录相当于MongoDB中的一个文档

1. **什么是非关系型数据库？**

非关系型数据库的显著特点是不使用SQL作为查询语言，数据存储不需要特定的表格模式。

1. **为什么用MOngoDB？**

架构简单

没有复杂的连接

深度查询能力,MongoDB支持动态查询。

容易调试

容易扩展

不需要转化/映射应用对象到数据库对象

使用内部内存作为存储工作区,以便更快的存取数据。

1. **MongoDB支持哪些数据类型？**

String

Integer

Double

Boolean

Object

Object ID

Arrays

Min/Max Keys

Datetime

Code

Regular Expression等

1. **为什么要在MongoDB中用"Code"数据类型**

"Code"类型用于在文档中存储 JavaScript 代码。

1. **为什么要在MongoDB中用"Regular Expression"数据类型?**

"Regular Expression"类型用于在文档中存储正则表达式

1. **为什么在MongoDB中使用"Object ID"数据类型?**

"ObjectID"数据类型用于存储文档id

1. **"ObjectID"有哪些部分组成?**

一共有四部分组成:时间戳、客户端ID、客户进程ID、三个字节的增量计数器

1. **在MongoDb中什么是索引?**

索引用于高效的执行查询,没有索引的MongoDB将扫描整个集合中的所有文档,这种扫描效率很低,需要处理大量的数据.

索引是一种特殊的数据结构,将一小块数据集合保存为容易遍历的形式.索引能够存储某种特殊字段或字段集的值,并按照索引指定的方式将字段值进行排序.

1. **对文档进行CRUD的API是什么？**

新增C:

db.collectionName.insert({key:value...})

查询R：

db.collectionName.find({key:value},{映射})

更新U: db.collectionName.update({key:value},{$set:{newkey:newValue}})

删除D：

db.collectionName.remove({key:value})

1. **如何添加索引?**

使用db.collection.createIndex()在集合中创建一个索引

1. **在MongoDB中如何排序？**

Sort()并使用 1 和 -1 来指定排序方式，其中 1 表示升序，而 -1 表示降序。

db.connectionName.find({key:value}).sort({columnName:1})

1. **什么是NoSQL数据库？NoSQL和RDBMS有什么区别？在哪些情况下使用和不使用NoSQL数据库？**

NoSQL是非关系型数据库，NoSQL = Not Only SQL。

关系型数据库采用的结构化的数据，NoSQL采用的是键值对的方式存储数据。

在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用NoSQL数据库。

在考虑数据库的成熟度；支持；分析和商业智能；管理及专业性等问题时，应优先考虑关系型数据库。

1. **Zepto和jQuery的区别?**

相同点:

都是优秀的js函数库,都是对JS的轻量级封装

语法，API大部分都一样（zepto是按照jquery的思路来设计的）

zepto相当于jquery的子集

同jquery一样都是以$为核心函数

不同点:

Jquery:针对的更多的是PC端、体积较重、API较为完善

Zepto:针对的是移动端、体积较轻、有自己独特的移动端事件

1. **JSON字符串和JS对象的相互转换?**

JSON字符串转换为对象：

JSON.parse(json字符串)

对象转换为JSON字符串:

JSON.stringify(对象)

1. **Zepto和jQuery不同的API?**

$.each(collection, function(index, item){ ... })

jQuery中只能用来遍历数组或对象

Zepto可以遍历数组、对象、字符串

clone()

jQuery中可以设置参数为true,来指示事件处理函数被复制

Zepto中只能复制集合中的所有元素,此方法不会将数据和事件处理程序复制到新的元素

attr()

jQuery中获取未定义的属性(selected/checked...),那么得到的是undefined,如果定义属性,那么得到的是和属性名一样的属性值

Zepto中获取未定义的属性(selected/checked...),那么得到的是false,如果定义属性,那么得到的是和属性名一样的属性值

DOM操作

jquery中插入DOM元素的时候添加配置对象(比如：id，class等。。。)不起作用,除非插入的DOM元素是一个空节点。

插入DOM元素的时候添加配置对象（id，class等。。）的时候可以添加进去，并且会直接显示在标签属性内，可以操作，影响DOM元素。

offset()

jQuery中获取得到的对象只包含了left、top

Zepto中获取得到的对象包含left、top、width、height

left和top和jQuery里面的结果一样;

width和height包含内容区域的高度+内边距+边框

尺寸:

jQuery里面有width()/height()、innerWidth()/innerHeight()、outerWidth()/outerHeight

Zepto只有width()/height(),没有innerWidth()/innerHeight()、outerWidth()/outerHeight

但是jQuery里面的width()/height()获取的是内容区域的宽/高;可以获取隐藏元素的宽/高

但是Zepto里面的width()/height()获取的是内容区域+内边距+边框;不可以获取隐藏元素的宽/高

事件委托:

jQuery中的事件委托:找对应的event.target触发对应的函数即可,其他不关心;

Zepto中的事件委托:

委托的事件先被依次放入数组队列里，然后由自身开始往后找直到找到最后，期间符合条件的元素委托的事件都会执行。

1、委托在同一个父元素,或者触发的元素的事件范围小于同类型事件(冒泡能冒到自身范围)

2、同一个事件

3、委托关联 操作的类要进行关联

4、绑定顺序---从当前的位置往后看

1. **jQuery事件委托和Zepto事件委托的区别?**

jQuery中的事件委托:找对应的event.target触发对应的函数即可,其他不关心;

Zepto中的事件委托:委托的事件先被依次放入数组队列里，然后由自身开始往后找直到找到最后，期间符合条件的元素委托的事件都会执行。

1、委托在同一个父元素,或者触发的元素的事件范围小于同类型事件(冒泡能冒到自身范围)

2、同一个事件

3、委托关联 操作的类要进行关联

4、绑定顺序---从当前的位置往后看

1. **Zepto新增的API？**

$.camelCase

将一组字符串变成“骆驼”命名法的新字符串，如果该字符已经是“骆驼”命名法，则不变化。

jQuery没有该API

concat(nodes, [node2, ...])

添加元素到一个Zepto对象集合形成一个新数组。如果参数是一个数组，那么这个数组中的元素将会合并到Zepto对象集合中。

indexOf(element, [fromIndex])

在当前对象集合中获取一个元素的索引值（注：从0开始计数）。如果给定formindex参数，从该位置开始往后查找，返回基于0的索引值，如果没找到，则返回-1。

pluck(property)

获取对象集合中每一个元素的属性值。返回值为 null或undefined值得过滤掉。

push(element, [element2, ...])

添加元素到当前对象集合的最后。

1. **如何取消上次发送的异步请求?**

abort()方法：该方法可以暂停一个HttpRequest的请求发送或者HttpResponse的接收，并且将XMLHttp Request对象设置 为初始化状态。

1. **确保发送一条验证码的案例?**

问法:点击获取验证码的按钮，用户十秒以后可以再次获取，当十秒过后第一次请求没有返回，用户再次点击获取;然后因为网速好或者其他原因，两次请求同时返回，该怎么解决？（可能不太能确保到底哪个验证码是最终的验证码）

解决方法:

1、5分钟之内的验证码都是一致的 （服务器端实现）

2、只返回一条验证码 （客户端实现）

代码实现:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width,initial-scale=1,minimum-scale=1,maximum-scale=1,user-scalable=no" />  <title>ajax容易忽视的重点</title>  <style type="text/css">  #btn {  width: 220px;  height: 40px;  background: red;  border-radius: 20px;  text-align: center;  line-height: 40px;  font-size: 16px;  }  </style>  </head>  <body>  <!--  \* 如何取消一个ajax请求 ----abort()方法--取消当前请求。  场景：点击获取验证码的按钮，用户十秒时候可以再次获取，当十秒过后第一次请求没有返回，用户再次点击获取  然后因为网速好或者其他原因，两次请求同时返回，该怎么解决  //需求用户可以再次点击的时候取消之前的请求。  \* disabled 禁止用户点击，操作按钮，表单项的时候只是针对click事件，并没有对touch事件作出处理。  -->  <button id="btn">获取验证码</button>  <script src="../js/zepto.js" type="text/javascript" charset="utf-8"></script>  <script src="../js/touch.js" type="text/javascript" charset="utf-8"></script>  <script type='text/javascript'>  $(function () {  // 倒计时的变量  var time = 3;  // 定时器的变量  var intervalId;  // 判断是否可以点击发送  var isSend = true; // 默认可以发送  var xhr = null;  // 当触摸按钮的时候,发送ajax请求  $("#btn").on("touchstart", function () {  if (isSend == false) {  return;  }  // 点击发送验证码,按钮呈现倒计时的效果【重点是：如果处于灰色状态,禁止点击】  // 禁止点击: 简单来说就是不执行事件处理函数  $(this).css('background', "gray");  isSend = false;  // 提示文本  $(this).html(time + "s之后再次发送");  intervalId = setInterval(() => {  $("#btn").html(--time + "s之后再次发送");  // 倒计时结束了  if (time == 0) {  clearInterval(intervalId)  $(this).html("获取验证码");  $(this).css('background', "red");  time = 3  isSend = true;  }  }, 1000);  // 第一次发送  if (xhr == null) {  xhr = sendSecurityRequest();  } else {  xhr.abort();  xhr = sendSecurityRequest();  }  })  // 获取验证码的请求  function sendSecurityRequest() {  // 发送ajax请求  var xhr = null;  xhr = $.ajax({  type: "get",  url: "http://127.0.0.1:3000/",  // 我们已经在服务器端进行跨域支持了,所以不需要写json  // dataType:"jsonp",  success: function (msg) {  console.log(msg);  },  error: function () {  }  })  return xhr;  }  })  </script>  </body>  </html> |

1. **AngularJS和jQuery的比较?**

jQuery:JS函数库、封装简化dom操作

Angular：JS结构化框架、主体不再是DOM, 而是页面中的动态数据

1. **什么是SPA?**

单页面应用（single page application，SPA），就是只有一张Web页面的应用。

单页应用程序 (SPA) 是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。

浏览器一开始会加载必需的HTML、CSS和JavaScript，所有的操作都在这张页面上完成，都由JavaScript来控制。因此，对单页应用来说模块化的开发和设计显得相当重要。

1. **什么是单向数据绑定?**

数据模型（Module）和视图（View）之间的单向绑定。

需要我们先写好模板，然后把模板和数据（可能来自后台）整合到一起形成HTML代码，然后把这段HTML代码插入到文档流里面。

单向数据绑定的缺点：一旦HTML代码生成好后，就没有办法再进行改变了，如果有新的数据出现，那就必须要先把之前的HTML代码删掉，然后重新把新的数据和模板一起整合形成新的HTML代码，再插入到文档流中。

1. **什么是双向数据绑定?**

数据模型和视图之间的双向绑定。

当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化；可以这样说用户在视图上的修改会自动同步到数据模型中去，数据模型也是同样的变化。

1. **AngularJS的单向数据绑定和双向数据绑定是什么?**

单向数据绑定:

当改变View中的数据, Model对象的对应属性也会随之改变

数据从View==>Model : ng-init

当Model域对象的属性发生改变时, 页面对应数据随之更新

数据从Model==>View : {{}}表达式

双向数据绑定:

数据可以从View流向Model, 也可以从Model流向View

ng-model是双向数据绑定

1. **什么是依赖注入?**

依赖注入（DI）是用来创建对象及其依赖的其它对象的一种方式。 当依赖注入系统创建某个对象实例时，会负责提供该对象所依赖的对象（称为该对象的依赖）。

1. **引入依赖对象的方式?**

方式一: 内部自己创建 : 不动态

方式二: 全局变量 : 污染全局命名空间

方式三: 形参引入依赖 : 依赖注入使用的方式

1. **AngularJS的依赖注入?**

angular的 ‘$scope’对象就是依赖对象，并且是依赖注入的形式进行使用。

形参必须是特定的名称, 否则Angular无法注入抛异常;

1. **JS中哪些行为是依赖注入?**

回调函数的event的就是依赖对象

回调函数有形参就是依赖注入

1. **什么是耦合、什么是解耦?**

耦合是指两个或两个以上的体系或两种运动形式间通过相互作用而彼此影响以至联合起来的现象。在软件工程中，对象之间的耦合度就是对象之间的依赖性。对象之间的耦合越高，维护成本越高，因此对象的设计应使类和构件之间的耦合最小。

解耦，字面意思就是解除耦合关系。在软件工程中，降低耦合度即可以理解为解耦，模块间有依赖关系必然存在耦合，理论上的绝对零耦合是做不到的，但可以通过一些现有的方法将耦合度降至最低。

1. **AngularJS的指令是干什么的?**

指令是Angular的一个特殊标志，也是有别于其他框架的一个重要特征，Angular之所以功能强大，在很大程度上得益于它拥有大量内置的指令，也能通过语法自定义指令。从字面意义来说，“指令”是一种执行的信号，一旦发布了这个指令，就要执行某项动作。Angular中的指令是一个在特定DOM元素上执行的函数。

1. **Angular表达式是干什么的?**

AngularJS 使用表达式 把数据绑定到 HTML。

1. **AngularJS的常用指令?**

ng-app 初始化一个 AngularJS 应用程序。

ng-model 绑定 HTML 控制器的值到应用数据

ng-controller 定义应用的控制器对象

ng-init 初始化应用程序数据。

ng-bind 绑定 HTML 元素到应用程序数据

ng-click 定义元素被点击时的行为

ng-repeat 定义集合中每项数据的模板

ng-show 显示或隐藏 HTML 元素

ng-hide 隐藏或显示 HTML 元素

ng-class 指定 HTML 元素使用的 CSS 类

ng-style 指定元素的 style 属性

ng-include 在应用中包含 HTML 文件

ng-click 定义元素被点击时的行为

ng-blur 规定 blur 事件的行为

ng-keydown 规定按下按键事件的行为

ng-keyup 规定按下按键事件的行为

ng-mouseover 规定鼠标指针位于元素上方时的行为

ng-mousemove 规定鼠标指针在指定的元素中移动时的行为

ng-mouseout 规定鼠标指针离开元素上方时的行为

ng-mousedown 规定按下鼠标按键时的行为

ng-mouseup 规定当在元素上松开鼠标按钮时的行为

ng-mouseenter 规定鼠标指针穿过元素时的行为

1. **简单介绍AngularJS的表单验证?**

Angular 支持非常强大的内置表单验证，maxlength、minlength、required 以及 pattern。使用 Angular 的内置表单校验能够完成绝大多数的业务场景的校验需求，但有时我们还需要实现更为复杂的表单校验功能，这时可以使用 Angular 提供的表单自定义校验（Custom Validator）。

1. **AngularJS对Ajax做了哪些封装?**

Angular提供了基本的 AJAX 封装，你直接面对 promise 对象，使用起来还是很方便的。

1. **AngularJS的三个重要对象?**

**作用域对象(scope)：**一个js实例对象, ng-app指令默认会创建一个根作用域对象($rootScope)；它的属性和方法与页面中的指令或表达式是关联的

**控制器对象(controller)：**用来控制AngularJS应用数据的实例对象

ng-controller : 指定控制器构造函数, Angular会自动new此函数创建控制器对象;同时Angular还有创建一个新的域对象$scope, 它是$rootScope的子对象在控制器函数中声明$scope形参, Angular会自动将$scope传入;

**模块对象**：

模块定义了一个应用程序。

模块是应用程序中不同部分的容器。

模块是应用控制器的容器。

控制器通常属于一个模块。

1. **声明式和命令式的区别?**

命令式更注重的执行的过程

声明式更注重的执行的结果

声明式是对命令式的局部包装

1. **什么是MVC？**

Model(数据模型)：是应用程序中用于处理应用程序数据逻辑的部分

View（视图）：是应用程序中处理数据显示的部分

Controller(控制)：负责从视图读取数据，控制用户输入，并向模型发送数据；

1. **什么MVVM?**

Model(数据模型)：是应用程序中用于处理应用程序数据逻辑的部分

View（视图）：应用程序的业务逻辑相关的数据的封装载体

ViewModel(视图模型)：负责View和Model的交互和协作

1. **简单介绍Angular的MVVM代表的是哪些部分?**

Model/模型:

scope中的各个数据对象

数据模型

View/视图:

html/css/directive/expression

显示VM中的数据

与用户交互

ViewModel/视图模型:

scope对象

View和Model的交互和协作

给View提供显示的数据

提供了View中Command事件操作Model的途径

Controller/控制层：

在angular中controller不再是架构的核心，在MVVM中只是起辅助作用，用来辅助$scope对象，即VM层

初始化Model数据

为Model添加行为方法

1. **ECMAScript 和 JavaScript 的关系?**

ECMAScript 和 JavaScript 的关系是，前者是后者的规范，后者是前者的一种实现

1. **JavaScript作用域?**

变量和函数能作用到的范围，在这个范围内起作用，它就是所谓的作用域。

全局作用域: 在整个页面可以访问的变量和函数就是全局作用域

函数作用域: 在函数中可以访问的变量和函数就是函数作用域

块级作用域: 在块里面可以访问的变量和函数就是块级作用域

1. **如何定义块级作用域?**

块级作用域可通过新增命令 let 和 const 声明，所声明的变量在指定块的作用域外无法被访问

1. **请介绍let和const?**

let 关键字用来声明变量，使用 let 声明的变量有几个特点：

不允许重复声明

块级作用域

不存在变量提升

不影响作用域链

const 关键字用来声明常量，const 声明有以下特点

声明必须赋初始值

标识符一般为大写

不允许重复声明

值不允许修改

块级作用域

声明对象类型一般使用const;

声明非对象类型一般使用let;

1. **Var、let、const的区别?**

var：允许重复定义，污染同一作用域下的变量，没有块级作用域

let：不允许重复定义，识别块级作用域

const：不能重复定义，定义的是常量，识别块级作用域

1. **请阐述对闭包的理解?**

闭包：内部函数总是可以访问其所在的外部函数中声明的参数和变量，即使在其外部函数被返回（寿命终结）了之后。

在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

闭包函数：声明在一个函数中的函数，叫做闭包函数。

1. **闭包的用途?**

可以在函数外部读取函数内部成员

让函数内成员始终存活在内存中

1. **闭包的坏处?**

闭包使函数内部的变量不能被内存释放，这些变量就会占用内存，内存消耗大，可能会导致内存泄露,甚至于堆栈溢出；

1. **手写一个简单的闭包?**

|  |
| --- |
| function fn1(a, b) { // a 和 b 是fn1的参数  let c = 10; // fn1内部的变量  // fn2是fn1函数内部的函数,它可能是闭包函数,也可能不是;  // fn2简称叫内部函数  // 内部函数总是可以访问其所在的外部函数中声明的参数和变量  function fn2() {  // fn2可以访问a,b,c  // console.log(a, b, c)  return {  a: a,  b: b,  c: c  }  }  return fn2; // 把fn2return出去,那么fn2就是一个闭包函数  // 闭包函数的特点是:  // 可以访问外部函数的参数和变量,即使外部函数执行完毕,也可以访问外部函数的参数和变量  }  var rfn = fn1(1, 2);  console.log("------------------");  var innerVar = rfn();  console.log(innerVar) |

1. **闭包代码阅读题?**

|  |
| --- |
| console.log("第一道:");  function funA() {  var a = 10; // funA的活动对象之中;  return function () { //匿名函数的活动对象;  console.log(a);  }  }  var b = funA();  b(); //10  console.log("第二道:");  function outerFn() {  var i = 0;  function innerFn() {  i++;  console.log(i);  }  return innerFn;  }  //每次外部函数执行的时候,都会开辟一块内存空间,外部函数的地址不同，都会重新创建一个新的地址  var inner = outerFn();  inner(); // 1  inner(); // 2  inner(); // 3  var inner2 = outerFn();  inner2(); // 1  inner2(); // 2  inner2(); // 3  console.log("第三道:");  var i = 0;  function outerFn(){  function innnerFn(){  i++;  console.log(i);  }  return innnerFn;  }  var inner1 = outerFn();  var inner2 = outerFn();  inner1(); // 1  inner2(); // 2  inner1(); // 3  inner2(); // 4  console.log("第四道:");  function fn(){  var a = 3;  return function(){  return ++a;  }  }  console.log(fn()()); // 4  console.log(fn()()); // 4  console.log("第五道:");  function outerFn(){  var i = 0;  function innnerFn(){  i++;  console.log(i);  }  return innnerFn;  }  var inner1 = outerFn();  var inner2 = outerFn();  inner1(); // 1  inner2(); // 1  inner1(); // 2  inner2(); // 2  console.log("第六道:");  (function() {  var m = 0;  function getM() { return m; }  function seta(val) { m = val; }  window.g = getM;  window.f = seta;  })(); // IIFE 立即调用函数  // 因为在立即调用函数里面,把f绑定给window,所以f方法就是全局方法  f(100); // 把100 赋值给 m  console.info(g()); // 100  // 闭包找到的是同一地址中父级函数中对应变量最终的值  console.log("第八题");  function m1(){  var x = 1;  return function(){  console.log(++x);  }  }    m1()(); //2  m1()(); //2  m1()(); //2    var m2 = m1();  m2(); //2  m2(); //3  m2(); //4  console.log("第九题");  var fn = (function(){  var i=10;  function fn(){  console.log(++i);  }  return fn;  })();  fn(); // 11  fn(); // 12  console.log("第十题")  function love1(){  var num = 223;  var me1 = function() {  console.log(num);  }  num++; // 224  return me1;  }  var loveme1 = love1(); // 调用外部方法  loveme1(); // 224  // 阿里巴巴面试题  console.log("第十一题");  function fun(n,o) {  console.log(o);  return {  fun:function(m) {  return fun(m,n);  }  };  }  /\*  var a = fun(0); 调用了外部函数fun,把0传递给了n,o没有传递参数,那么o是undefined  n = 0, o = undefined  a = {  fun:function(m) {  return fun(m,n);  }  }    a.fun(1) => function(m) {return fun(m,n);}  m = 1,n = 0 fun(1,0)  a.fun(2) => function(m) {return fun(m,n);}  m = 2,n = 0 fun(1,0)    a.fun(3) => function(m) {return fun(m,n);}  m = 3,n = 0 fun(1,0)  \*/  var a = fun(0); // undefined  a.fun(1); // 0  a.fun(2); // 0  a.fun(3); // 0  console.log("-------------------------")  /\*  var b = fun(0); 调用了外部函数fun,把0传递给了n,o没有传递参数,那么o是undefined  n = 0, o = undefined  b = {  fun:function(m) {  return fun(m,n);  }  }  var b = fun(0).fun(1) => function(m) {return fun(m,n);}  m = 1,n = 0 fun(1,0)  m => n n=>o 打印0  b = {  fun:function(m) {  return fun(m,n);  }  }  var b = fun(0).fun(1).fun(2)  m = 2,n = 1  m => n n = >o , n = 2 ,o = 1  b = {  fun:function(m) {  return fun(m,n);=>fun(n,o)  }  }  var b = fun(0).fun(1).fun(2).fun(3)  m = 3,n = 2  m => n,n => o  \*/  var b = fun(0).fun(1).fun(2).fun(3); // undefined 0 1 2  console.log("-------------------------")  var c = fun(0).fun(1); // undefinde 0  c.fun(2); // 1  c.fun(3); // 1 |

1. **什么是递归?**

递归就是调用自身的一种编程技巧，在程序设计中应用广泛。递归函数就是函数对自身的调用，是循环运算的一种算法模式。

1. **求n的阶乘?**

|  |
| --- |
| function factorial1(n) {  return n == 1 ? 1 : n \* factorial1(n - 1);  }  function factorial2(m, n) {  return m == 1 ? n : factorial2(m - 1, m \* n);  } |

1. **统计指定节点及其所有后代元素节点的个数？**

|  |
| --- |
| function f(n) {  var l = 0; //初始化计数变量  if (n.nodeType == 1) {  l++  }; //如果是元素节点，则计数  var child = n.childNodes; //获取子节点集合  for (var i = 0; i < child.length; i++) { //遍历所有子节点  l += f(child[i]); //递归运算，统计当前节点下所有子节点数  }  return l; //返回节点数  }  window.onload = function () {  // 元素节点的节点类型 1  console.log(f(document.body)); //返回2，即body和script两个节点  } |

1. **JS尾递归的理解?**

尾递归是递归的一种优化算法，递归函数执行时会形成一个调用函数，当子一层的函数代码执行完成之后，父一层的函数才会销毁调用记录，这样就形成了调用栈，栈的叠加可能会产生内存溢出。而尾递归函数的每子一层函数不再需要使用父一层的函数执行完毕就会销毁栈记录，避免了内存溢出，节省了内存空间。

1. **什么是解构赋值?**

解构赋值是对赋值运算符的扩展。

它是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。

let [a, b, c] = [11, 22, 33];

let {b,a} = {a: 1,b: 2}

1. **简单介绍ES6模版字符串?**

模版字符串是字符串的增强版,特点如下:

1、模版字符串中可以出现换行符

2、可以使用${变量名}来拼接变量

1. **箭头函数的特性?**

1、this指向问题，this 是静态的. this 始终指向函数声明时所在作用域下的 this 的值

2、箭头函数不能new

3、不能使用arguments

4、箭头函数简写

1. **JS中this指向都有哪些?**

对象里面的this,代表的就是该对象

构造函数里面的this,代表的就是实例化的对象

事件函数里面的this,代表的就是触发事件函数的元素对象

箭头函数里面的this,this始终指向函数声明时所在作用域下的 this 的值

1. **ES6数组新增API有哪些?作用是什么?**

1、every() 方法用于检测数组所有元素是否都符合指定条件（通过函数提供）。 every()它的返回值是boolean类型,如果每个元素都符合条件,则true,只要一个不满足就是false

|  |
| --- |
| const ages = [32, 33, 16, 40];  // 检测数组里面的元素是否都大于18  let isGt = ages.every(function (item, index, arr) {  // 在函数里面指定条件,需要使用return返回  return item > 12;  })  console.log(isGt); |

2、some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。some()它的返回值是boolean类型,只要有一个元素符合条件,则true,都不符合才false；

|  |
| --- |
| let isLt = ages.some(function (item, index, arr) {  // 在函数里面指定条件,需要使用return返回  return item < 18;  })  console.log(isLt) |

3、fill() 方法用于将一个固定值替换数组的元素。

|  |
| --- |
| let fruits = ["Banana", "Orange", "Apple", "Mango"];  // fruits.fill("XiGua");  // fruits.fill("XiGua", 2);  fruits.fill("XiGua", 1, 3);  console.log(fruits) |

1. filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

|  |
| --- |
| // 筛选出年龄大于20的元素  const result = ages.filter((item, index, arr) => item > 20)  console.log(result) |

1. forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

|  |
| --- |
| const arr = [11, 22, 33, 44];  /\* for (let i = 0; i < arr.length; i++) {  console.log(arr[i]);  }  for (let i in arr) {  console.log(arr[i]);  } \*/  arr.forEach((item) => {  console.log(item);  }) |

6、map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

|  |
| --- |
| const numArr = [3, 4, 5];  // 求数组里面每个元素的平方,得到一个新的数组  const newNumArr = numArr.map((item) => {  return item \* item;  })  console.log(newNumArr) |

1. **JS数组方法some()和every()的区别？**

共同点：

1.遍历数组；

2. 三个参数分别是item,index,arr(数组具体项，位置，数字本身)；

3.返回的都是布尔值；

区别：

some()方法，遍历数组的每一项，若其中一项为 true，则返回true；

every()方法，遍历数组每一项，若全部为true，则返回true；

1. **JS数组方法map()和filter()的区别?**

map() 方法创建一个新数组，其结果是该数组中的每个元素都调用一个提供的函数,回调函数return的结果是什么就存储什么，它更多是用来对数组里面的元素进行处理。

filter()方法创建一个新数组，其结果是该数组中的每个元素都调用一个提供的函数,回调函数return结果为true,则存储这个值，它更多是用来对数组里面的元素进行过滤筛选；

1. **合并数组的实现?**

|  |
| --- |
| const chopsticks = ['王太利', '肖央'];  const phoenix = ['曾毅', '玲花'];  // 使用Array.concat()  // const newGroup = chopsticks.concat(phoenix);  const newGroup = [...phoenix, ...chopsticks]; // ['曾毅', '玲花','王太利', '肖央']  console.log(newGroup) |

1. **JS伪数组转换为数组?**

|  |
| --- |
| const divs = document.getElementsByTagName("div");  console.log(divs) // divs是一个伪数组  const divArr = [...divs];  console.log(divArr) |

1. **请简单阐述对Symbol的理解?**

ES6 引入了一种新的原始数据类型`Symbol`，表示独一无二的值;为了从根本上防止属性名的冲突；

Symbol 值通过`Symbol`函数生成。这就是说，对象的属性名现在可以有两种类型，一种是原来就有的字符串，另一种就是新增的 Symbol 类型。凡是属性名属于 Symbol 类型，就都是独一无二的，可以保证不会与其他属性名产生冲突。

1. **请简单阐述ES6迭代器?**

遍历器（Iterator）就是一种机制。它是一种接口，为各种不同的数据结构提供统一的访问机制。任何数据结构只要部署 Iterator 接口，就可以完成遍历操作。

ES6 创造了一种新的遍历命令 for...of 循环，Iterator 接口主要供 for...of 消费

原生具备 iterator 接口的数据(可用 for of 遍历)

1. **ES6迭代器的工作原理?**

a) 创建一个指针对象，指向当前数据结构的起始位置

b) 第一次调用对象的 next 方法，指针自动指向数据结构的第一个成员

c) 接下来不断调用next 方法，指针一直往后移动，直到指向最后一个成员

d) 每调用 next 方法返回一个包含 value 和 done 属性的对象

1. **ES6迭代器的作用?**

迭代器的作用就是供for of消费的；当你需要自定义数据遍历的时候,就要想到迭代器

1. **请简单介绍ES6生成器?**

生成器函数是 ES6 提供的一种**异步编程解决方案**，语法行为与传统函数完全不同

1. **ES6 Promise的作用?**

Promise 是 ES6 引入的异步编程的新解决方案。语法上 Promise 是一个构造函数，用来封装异步操作并可以获取其成功或失败的结果。

1. **通过Promise实现异步文件读取?**

先读取为学、再读取插秧诗、再读取观书有感；把读取得到的数据放到一起

|  |
| --- |
| const fs = require('fs');  /\* fs.readFile("./resources/为学.md", function (err1, result1) {  if (err1) {  throw err1;  } else {  fs.readFile("./resources/插秧诗.md", function (err2, result2) {  if (err2) {  throw err2;  } else {  fs.readFile("./resources/观书有感.md", function (err3, result3) {  if (err3) {  throw err3;  } else {  let result = result1 + "\r\n" + result2 + "\r\n" + result3;  console.log(result)  }  })  }  })  }  }) \*/  const promise = new Promise((resolve, reject) => {  fs.readFile("./resources/为学.md", (err, data) => {  if (err) reject("为学->读取失败");  resolve(data)  })  })  promise.then(value => { // 第一次读取得到的数据value  // 继续读取第二本  return new Promise((resolve, reject) => {  fs.readFile("./resources/插秧诗.md", (err, data) => { // 第二次读取得到的数据  if (err) reject("插秧诗->读取失败");  resolve([value, data])  })  })  }).catch(reason => {  console.log(reason);  })  .then(value => { // 第二次读取的封装 [value, data]  return new Promise((resolve, reject) => {  fs.readFile("./resources/观书有感.md", (err, data) => { // 第二次读取得到的数据  if (err) reject("观书有感->读取失败");  value.push(data)  resolve(value)  })  })  }).catch(reason => {  console.log(reason);  }).then(value => {  console.log(value.join("\r\n"))  }) |

1. **Set集合概述?**

ES6 提供了新的数据结构 Set（集合）。它类似于数组，允许你存储任何类型的唯一值，无论是原始值或者是对象引用，集合实现了 iterator 接口，所以可以使用『扩展运算符』和『for…of…』进行遍历

1. **数组和Set集合相互转换?**

数组转换为Set集合,通过Set构造函数传递数组参数:

const arr1 = ["张三", "李四", "王武"];

const set1 = new Set(arr1);

Set集合转换为数组,通过扩展运算符：

const arr2 = [...set2];

1. **JS实现数组去重?**

|  |
| --- |
| const numArr = [1, 2, 3, 3, 2, 1];  const resultArr = [...new Set(numArr)];  console.log(resultArr)  /\* 原生JS去重 \*/  function unique(arr) {  for (var i = 0; i < arr.length; i++) {  for (var j = i + 1; j < arr.length; j++) {  if (arr[i] == arr[j]) { //第一个等同于第二个，splice方法删除第二个  arr.splice(j, 1);  j--;  }  }  }  return arr;  }  var arr = [1, 1, 'true', 'true', true, true, 15, 15, false, false, undefined, undefined, null, null, NaN, NaN,  'NaN', 0, 0, 'a', 'a', {}, {}  ];  console.log(unique(arr)) |

1. **使用JS求两个数组的并集、交集、差集?**

|  |
| --- |
| /\* 并集：把两个数组合并为一个数组去重即可 \*/  const arrA = [1, 2, 3];  const arrB = [2, 3, 4, 5];  const union = [...new Set([...arrA, ...arrB])];  console.log(union)  /\* 交集:使用filter()结合集合的hash() \*/  const unite = arrA.filter(item => new Set(arrB).has(item))  console.log(unite)  /\* 差集: 使用filter()结合集合的hash()\*/  /\* 差集的话,必须使用值多的来调用filter \*/  const diff1 = arrB.filter(item => !(new Set(arrA).has(item)))  const diff2 = arrA.filter(item => !(new Set(arrB).has(item)))  const diff = [...diff1, ...diff2];  console.log(diff) |

1. **Array和Set的区别？**

Array数组是表示一种储存在连续空间中的结构类型（内容可以为number,object等）

Set更像是一种抽象的数据类型。它只包含不同的元素/对象，不需要连续分配存储空间。

它们之间最大的差别就是Array中的元素是可以重复的，而Set中的元素不可重复。除此之外，Array被认为是一种索引集合，而Set是一种键的集合。

1. **Map集合概述?**

Map对象保存键值对。任何值(对象或者原始值)都可以作为一个键或一个值。

1. **Map 和 Object 的区别？**

Map对象是一种有对应键/值对的对象，JS的Object也是 键/值对的对象 ；

在Object对象中，只能把String和Symbol作为key值， 但是在Map中，key值可以是任何基本类型;

通过Map中的size属性， 可以很方便地获取到Map长度， 要获取Object的长度， 你只能用别的方法了；

1. **ES6实现继承的方式?**

|  |
| --- |
| // 定义手机父类  class Phone {  constructor(brand, price) {  this.brand = brand;  this.price = price;  }  call() {  console.log("我可以打电话!");  }  }  // 定义子类,子类想要继承父类,必须通过extends来实现继承  class SmartPhone extends Phone {  constructor(brand, price, color, size) {  // 调用父类的构造函数  super(brand, price)  this.color = color;  this.size = size;  }  // 我可以重写父类继承过来的方法  call() {  console.log("我可以视频通话!");  }  photo() {  console.log("我可以拍照!");  }  game() {  console.log("我可以玩游戏!");  }  }  const xm = new SmartPhone("小米", 1999, "黑色", "5.5")  console.log(xm)  xm.call();  xm.game(); |

1. **JS实例对象和函数对象分别指的是什么?**

实例对象：通过 new 函数产生的对象称为实例对象，简称对象。

函数对象：将函数作为对象使用时

1. **编码题：0.1+0.2和0.3进行等值比较,要求得到true;**

|  |
| --- |
| function numEquals(a, b) {  if (Math.abs(a - b) < Number.EPSILON) {  return true;  } else {  return false;  }  }  console.log(0.1 + 0.2 == 0.3);  console.log(numEquals(0.1 + 0.2, 0.3)); |

1. **JS模块化规范?**

CommonJs规范

​ 根据CommonJs规范，每个文件就是一个模块，有自己的作用域。

​ 在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见，CommonJS规范加载模块是同步的，也就是说，加载完成才可以执行后面的操作。

在服务器端: 模块的加载是运行时同步加载的。

在浏览器端: 模块需要提前编译打包处理。

AMD规范

​ AMD是"Asynchronous Module Definition（异步模块定义）"的简写，也就是它采用异步方式加载模块，依赖前置，特点是准备充分，但加载会较慢。通过define方法去定义模块，require方法去加载模块。

​ 专门用于浏览器端。

CMD（用于浏览器环境）

​ CMD是“Common Module Definition（通用模块定义）”的简写，CMD规范也会异步加载，不同的是CMD依赖就近，特点是首次加载很快。CMD规范也是使用define()定义模块，require()加载模块。但和AMD规范思想不同，写法也不同。

​ 专门用于浏览器端, 模块的加载是异步的，模块使用时才会加载执行

**ES6模块规范**

​ ES6 模块的设计思想是尽量的静态化，使得编译时就能确定模块的依赖关系。使用import和export来导入导出，ES6还提供了一个default，用来提供默认的export。特点是加载模块存储的是值的引用，所以全局只有一份；加载模块也是异步的。ES6的module吸收了CommoneJS和AMD两者的优点，兼容两标准的规范。

导出：exprot来实现导出,分为分别暴露、统一暴露、默认暴露;

导入：import来实现导入,通用导入方式、解构赋值导入方式

1. **目前会的技术？**

第一阶段：HTML、HTML5、CSS、CSS3.0、JavaScript；

老师18、闪银、云联数据、好望角、屏芯科技;

第二阶段：jQuery、AJAX、json-server、Bootstrap、Layui、Git、Sass、Less

jQuery是JS的轻量级框架;

AJAX是用来发送异步请求

Bootstrap是HTML、CSS 和 JS 进行封装的前端UI框架; - 前台页面

HTML:修改了标签的默认样式

CSS:提供了很多类

JS: 封装了很多动态特性

前台页面是给用户使用

Layui是HTML、CSS 和 JS 进行封装的前端UI框架; - 后台页面

核心就是内置模块（模版引擎、数据表格、表单等）;

后台页面是给管理员或后台人员使用;

Git是版本控制工具,对于项目进行管控;

Sass、Less是CSS预处理器;

第三阶段：NodeJS、Express、art-template、MongoDB、Zepto、AngularJS

第四阶段：ES6

前端页面基础：HTML/HTML5/CSS/CSS3/JavaScript

JS类库：jQuery

异步请求：AJAX

UI框架：Bootstrap、Layui

版本控制工具：Git

数据mock：json-server

预处理CSS：Sass、Less