**HTML:**

1. Web语义化
2. 含义：使用语义恰当的标签，让页面具有良好的结构与含义，让人和机器都能快速理解网页的内容
3. 优点：在无样式的情况下能够让页面呈现出清晰的结构；有利于SEO，和搜索引擎建立良好的沟通，有助于爬虫抓取更多有效信息；方便屏幕阅读器等其他设备的解析；使代码解构清晰，更具可读性，便于团队的开发和维护
4. 常见语义标签：title、hn、p、header、nav、main、article、section、aside、footer、strong、em、cite
5. W3C规范
6. 声明DOCTYPE
7. 定义语言编码
8. 正确的JavaScript和CSS定义，并尽量使用外链的样式表和脚本
9. 所有标签都要有结束标签
10. 所有标签的元素和属性名都要小写
11. 所有标签都必须合理嵌套
12. 所有属性必须用引号括起来
13. 所有属性必须带有相应的值
14. 为图片添加有意义的alt属性
15. 在表单中添加label
16. Doctype相关
17. 概念：document type文档类型，用来指定浏览器使用哪个文档标准来解析文档
18. HTML4.01是基于SGML（标准通用标记语言）的，在<!DOCTYPE>声明中引用DTD（文档类型定义），有Strict、Transitional和Frameset三种
19. HTML5声明<!DOCTYPE html>
20. 块级元素和内联元素
21. 块级元素：独占一行、可以设置宽高和内外边距
22. 内联元素：不会独占一行、不可以设置宽高且竖直方向上内外边距无效
23. 常见内联元素：a、img、span、i、input、select、label
24. src与href的区别

href表示超文本引用，用来建立当前文档和引用文档之间的联系；src表示source，src的内容是页面不可缺少的一部分，是引入的关系，阻塞解析和渲染

1. meta标签
2. 用处：用于定义页面的description、keywords和最后修改时间等元数据，这些元数据将服务于浏览器、搜索引擎和其他网络服务
3. 组成：必选的content和可选的name及http-equiv
4. charset属性：声明文档的字符编码
5. name属性：keywords页面关键字、description页面内容描述、viewport移动端窗口、robots搜索引擎抓取
6. http-equiv属性：X-UA-Compatible选择渲染版本、cache-control指定缓存机制（优先级高于http header的设置）
7. link和import
8. link属于HTML标签，没有兼容问题，而import是CSS提供的语法规则，IE5以上才能发识别
9. 页面被加载时，link会同时被加载，而import引用的CSS会等到页面加载完后加载
10. link样式的权重高于import
11. HTML全局属性：id、class、title、style、data-\*、draggable
12. cookie、sessionStorage和localStorage异同
13. 相同：三者都会在客户端缓存，都有同源策略和大小的限制；
14. 发送不同：Cookie会随着http或https请求发送到服务器，会占用带宽，而web storage只保存在本地，不会占用带宽；
15. 大小不同：cookie数据不超过4K，web storage数据存储可达5M；
16. 有效期不同：cookie在过期之前一直有效，如果未设置过期时间则关闭浏览器前有效；sessionStorage仅在关闭浏览器前有效；localStorage数据永久有效；
17. 作用域不同：sessionStorage不在不同的浏览器窗口之间共享，即时是同一个页面，cookie在同源的路径或子路径下是共享的（不需要端口相同），localStorage在同源窗口中都是共享的；
18. web storage的API更丰富，cookie要自己封装操作
19. WebSocket
20. 概念：基于TCP的新网络协议，实现了客户端与服务器的全双工通信——允许服务器主动发送信息给客户端
21. API：
22. 实例： var Socket = new WebSocket(“ws://localhost:8080/socket”);
23. 属性：Socket.readyState——0-建立、1-可通信、2-正在关闭、3-已关闭
24. 事件：Socket.onopen、Socket.onmessage、Socket.onerror、Socket.onclose
25. 方法：Socket.send()、Socket.close()
26. WebWork
27. 概念：为了利用多核CPU的计算能力，H5提出了Web Worker标准，允许JS脚本创建多个线程，但是子线程完全受主线程的控制且不得操作DOM
28. API：
29. 实例：var Worker = new Worker(“worker.js”);
30. 事件：Worerk.onmessage
31. 方法：Worker.postMessage()

**CSS：**

1. CSS选择器优先级
2. 不同级别：!important > 内联样式 > ID选择器 > 类/伪类/属性选择器 > 标签/伪元素选择器 > 通配符 > 继承值 > 用户代理
3. 同一级别：就近原则
4. CSS继承属性：font、line-height、color、visbility、cursor
5. 清除浮动
6. 目的：主要解决高度塌缩和文字绕图问题
7. 方法：添加clear样式；插入清除浮动的块级元素；伪元素清除浮动；overflow不为visible
8. Flex布局属性
9. 容器属性：flex-direction、flex-warp、flex-flow（flex-direction+flex-warp）、justify-content、align-content、align-items
10. 项目属性：order、flex-grow、flex-shrink、flex-basis、flex（flex-grow+flex-shrink+flex-basis）、align-self
11. FOUC
12. 概念：FOUC（Flash Of Unstyled Content）无样式内容闪烁
13. 原因：用import导入样式表、将样式表放在页面底部、有几个样式表放在不同位置
14. 解决方法：使用link标签将样式表放在文档head中
15. BFC
16. 概念：BFC（Block Formatting Context）格式化上下文，指一个独立的渲染区域或者说一个隔离的独立容器
17. 形成条件：
18. 浮动元素（float不为none）
19. 定位元素（absolute、fixed）
20. display值为inline-block、table-cell、table-caption
21. overflow值不为visible
22. 特性：
23. 内部盒子垂直排列、左对齐
24. 外边距不会折叠
25. 不被浮动元素覆盖
26. 高度不会塌缩
27. 重排与重绘
28. 概念：重排是DOM的变化导致部分渲染树需要重新构造，而重绘只是屏幕上的部分内容需要更新，但是渲染树结构不受影响
29. 触发重排：
30. 页面初始化渲染
31. 浏览器窗口尺寸改变
32. 添加或删除可见的DOM元素(display：none)
33. 元素位置、尺寸或内容改变
34. 减少重排重绘：
35. 使用translate代替top
36. visibility代替display: none
37. 把DOM离线后修改，修改完毕后再display出来
38. 使用requestAnimationFrame
39. 用will-change属性提升渲染层
40. display: none与visibility: hidden的区别
41. display;none会让元素完全从渲染树中消失，渲染时不占据任何空间；visibility:hidden不会让元素消失，渲染时占据空间只是内容不可见
42. display:none是非继承属性，后代节点消失是因为该元素消失，修改后代节点属性无法显示；visibility:hidden是继承属性，修改后代节点属性可以使之显示
43. 修改常规流中元素的display属性通常造成文档重排；修改visibility属性只会造成本元素的重绘
44. 读屏器会读取visibility:hidden元素的内容，不会读取display:none元素的内容
45. display、float和position关系
46. 当display: none时，不产生Box，float和position都不起作用
47. 当position为absolute或fixed时，float不起作用，display的值按下表转换
48. 当float不为none时，Box浮动且display的值按下表转换
49. 如果元素是根元素，display的值按下表转换

|  |  |
| --- | --- |
| **设定值** | **转换后** |
| inline-table | table |
| inline,inline-block,table-row... | block |
| 其他 | 保持设定值 |

1. CSS常见hack方法
2. 概念：由于不同厂商的浏览器或浏览器的不同版本，对CSS对支持、解析不一样，导致在不同的浏览器环境中呈现出不一致的页面展现效果，因而为了获得统一的页面效果，就需要对不同的浏览器或不同版本写特定的CSS样式
3. hack分类：
4. 条件注释：<!-- [if IE]>这段文字只在IE浏览器显示<![endif]-->，针对IE浏览器特有的Hack方式
5. 属性前缀：在样式属性名前面加上一些特定浏览器才能识别的前缀，例如-color（IE6）、\*color、color: red\0等
6. 选择器前缀：在CSS选择器前面加上一下特定浏览器才能识别的前缀，例如\*html（IE6）、\*+html（IE7）、@media screen\0
7. 媒体查询
8. 作用：使用媒体查询，可以针对不同的媒体类型定义不同的样式，是响应式设计的关键组成部分
9. 方法：
10. 在link中判断设备类型，然后引用不同的CSS文件：<link ref=”stylesheet” type=”text/css” media=”screen and (max-width: 400px)” href=”styleA.css”>
11. 在CSS文件中判断设备类型，然后决定样式：@media screen and (max-width: 400px) {...}
12. 文字缩略

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

1. 居中
2. 水平居中：
3. 内联元素：text-align:center
4. 块级元素：margin:0 auto
5. 多个块级：text-align:center + display:inline-block或者display:flex + justify-content
6. 垂直居中：
7. 内联元素：height = line-height
8. 多行内联元素：display:flex + justify-content + flex-direction:column
9. 固定高度块级元素：position:absolute + top:50% + margin-top:-height/2
10. 未知高度块级元素：position:absolute + top:50% + transform:translateY(-50%)
11. 水平垂直居中：
12. 固定宽高元素：position:absolute + top:50% + left:50% + margin-top:-height/2 + margin-left:-width/2或者position:absolute + top:0 + left:0 + bottom:0 + right:0 + margin：auto
13. 未知宽高元素：position:absolute + top:50% + left:50% + transform:translate(-50%, -50%)
14. Flex布局：display:flex + justify-content:center + align-item:center
15. 两栏布局
16. 浮动布局：aside块向左浮动，main块设置aside块大小的margin-left值；
17. 绝对布局：aside块绝对定位到左边，main块设置aside块大小的margin-left值；
18. Flex布局：容器块设置display:flex，aside块设置固定宽度，main块flex-grow设置为1；
19. 双飞翼布局：为main块添加容器块，aside块向左浮动，margin-right设置为-100%，容器块向左浮动且设置aside块大小的margin-left值；

（5）表格布局：容器块设置display:table，对两个子块设置display:table-cell，对aside块设置固定宽度

1. 三栏布局

（1）浮动布局：left块向左浮动，right块向右浮动，main块设置大小为left、right块width值的margin-left和margin-right值；

（2）绝对定位：left块绝对定位到左边，right块绝对定位到右边，为main块设置大小为left、right块width值的margin-left和margin-right值；

（3）Flex布局：容器块设置displ:flex，left和right块设置固定宽度，main块的flex-grow设置为1；

（4）圣杯布局：容器块设置等于left、right块的内边距（为两栏腾出空间），将三个子块向左浮动，left块的margin-left值设置为-100%并相对定位左移width大小的距离，right块的margin-left值设置为-width并相对定位右移width大小的距离，main块左浮动且width为100%；

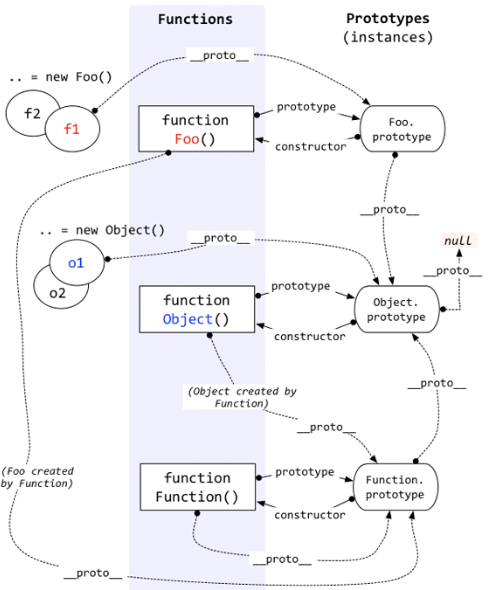
1. 双飞翼布局：为main块添加容器块，将容器、left及right块向左浮动，left块的margin-left值设置为-100%，right块的margin-left值设置为-width，为容器块设置100%宽度及大小为left、right块width值的margin-left和margin-right值并清除浮动；
2. 表格布局：容器块设置display:table，对三个子块设置display:table-cell，对left和right块设置固定宽度
3. 等高布局
4. padding补值：容器块设置overflow: hidden，在对aside或main块设置padding-bottom: 9999px，再用margin-bottom：-9999px调整间距（缺点是等高div没有底边的边框，可以通过绝对框模拟底边的边框）
5. flex布局：对容器块设置display: flex及align-items: stretch，再对main设置flex-grow: 1即可
6. Table-cell:容器块设置display: table及width: 100%，对aside块和main块设置display: table-cell即可

**JavaScript：**

1. JS引擎解析过程
2. 语法检查：词法分析 -> 语法分析
3. 预编译：创建执行环境 -> 属性填充（形参 -> 函数声明 -> 变量声明）
4. 脚本预编译（脚本执行前）：创建VO对象即全局对象 -> 查找函数声明，函数名做为全局对象属性，值为函数引用 -> 查找全局变量声明（包括隐式全局变量声明），变量名作为全局对象属性，值为undefined
5. 函数预编译（函数执行前）：创建AO对象 -> 查找函数形参，形参值为AO对象属性，值为实参值，如果没有传递实参则为undefined -> 查找函数声明，函数名做为AO对象属性，值为函数引用 -> 查找变量声明，变量名作为AO对象属性，值为undefined

（3）执行代码：此阶段程序真正进入执行，JS引擎一行一行读取并运行代码，此时执行环境中的变量都会被重新赋值，不仅仅是VO/AO，this和作用域链也会因某些语句而改变

1. 执行环境、变量对象及作用域
2. 执行环境（Execution Context一个内部对象）：JS引擎初始执行代码时，它首先默认进入全局执行环境，函数的每次调用都会创建一个新的执行环境，且这个执行环境会被推入一个执行环境栈Execution Stack中，在该函数被执行完后，执行栈将这个环境弹出
3. 变量对象（Variable Object）：在执行环境的创建阶段即预编译阶段，JS解释器会为这个环境创建一个变量对象VO，然后将定义在这个执行环境中的变量、函数（这也是声明提升的原因）以及this（this是在运行时绑定的）放入这个对象中，并且会将自身作用域中的VO/AO到保存到每一个函数的[[scope]]属性当中,在函数的执行环境中JS引擎会创建一个活动对象AO作为变量对象VO使用，活动对象AO中除了变量、函数声明和this外，还有各个参数及arguments对象
4. 作用域：一套保证对执行环境有权访问的所有变量和函数的有序访问的规则。作用域中变量的查找分为取值RHS和赋值LHS两种查找，不成功的取值查找会抛出ReferenceError错误（取值成功但对值的不正确操作会报TypeError错误，如果对变量做函数调用），而不成功的赋值查找会导致隐式创建全局（严格模式下还是报错的）。当一个函数执行环境创建完开始执行后，执行环境将取得函数内部[[scope]]属性所保存的作用域链（由一系列VO组成的类似链表结构）并加入执行环境中，同时将这个阶段创建的AO推入这个作用域的最前端，形成这个执行环境的作用域链。此后执行阶段在查找变量的时候，会从当前执行环境中的作用域链最前端即自身的AO中查找变量，如果没有就往作用域链的下一个VO/AO中查询，直至作用域链顶端及全局对象（在原型链中查找不到变量时会返回undefined，在作用域链顶层即全局对象中查找不到该变量时会去Object.prototype中查找，因为全局对象也是继承自Object.prototype，再查找不到则会报错ReferenceError而不是返回undefined）（执行环境的作用域链会随着执行环境的运行而产生，也会在执行环境的出栈而销毁，而每个函数内部的[[scope]]属性永远保存着）
5. 闭包
6. 概念：在一个函数A内定义一个函数B，函数B访问了函数A中的变量并被返回，使得函数B依然持有对函数A的作用域的引用，而这个引用就叫作闭包（定时器、事件监听等回调函数基本都会形成闭包）
7. 机制：在JS中作用域链只能向上访问不能向下访问，所以在函数A内部定义的函数B可以访问函数A内的局部变量，而函数B被返回后使得在外部可以访问函数A内部的局部变量，且这些局部变量一直保存在内存中
8. 用途：可以读取函数内部的变量，实现安全的私有属性和方法；让这些变量始终保存在内存中
9. 缺点：使用不当容易造成内存泄漏
10. 垃圾回收机制
11. 概念：JS具有自动垃圾回收机制，垃圾收集器会周期性地找出不再使用的变量，释放其内存。
12. 方式：JS使用标记清除的方式进行垃圾回收，即垃圾回收器遍历内存中所有可访问的变量并标记，之后垃圾回收器扫描并回收所有未被标记的变量。
13. 内存泄露的情况
14. 意外的全局变量
15. 页面元素被移除或替换时，元素的引用或元素绑定的事件仍没有被移除
16. 在函数中为元素添加事件，形成闭包
17. 子元素存在引用则它的所有父元素都不会被删除
18. 解决作用域污染
19. 定义一个全局变量，把其他全局变量追加到该命名空间下（但是外部可以随意修改内部成员）
20. 利用匿名自执行函数将脚本包裹起来，形成闭包（外部无法修改没有暴露的变量和函数）
21. 模块化
22. this的指向（this使得我们不用老是显示传递上下文对象）
23. 构造函数：新创建的对象
24. 显示绑定：通过call、apply方法显示地设置为第一个参数对象
25. 方法调用：方法的所属对象（在回调中会丢失所属对象，因为传递是函数的引用）
26. 函数调用：全局对象（window）
27. 全局范围：全局对象（window）
28. new关键字动作
29. 创建一个新对象
30. 将新对象\_\_proto\_\_指向构造函数的原型对象，建立原型链
31. 将this指向该对象，属性和方法被添加到这个对象中
32. 在没有显式返回其他对象的情况下，返回该新对象
33. 对象属性描述符
34. 数据属性：configurable、enumerable、writable及value
35. 访问器属性：configurable、enumerable、get及set
36. API：Object.defineProperty(obj, prop, descriptor)
37. 原型、原型链
38. 核心：Function.prototype和Object.prototype是两个特殊的对象，他们由JS引擎来创建，并且JS引擎将Object.prototype.\_\_proto\_\_设置 为null，将Function.prototype.\_\_proto\_\_指向Object.prototype，有了这两个对象之后所有的构造函数和实例都由构造器new和原生的Object及Function构造函数创建，（function Foo() {} == var Foo = new Function() ）
39. 原型：每个JS对象中都有一个[[prototype]]内部属性（object.\_\_proto\_\_内置于Object.prototype的访问器属性，）指向该对象的原型对象，以及一个constructor属性（来自prototype.constructor）指向该对象的构造函数；每个JS函数都有一个prototype属性指向原型对象，当这个函数被用作构造函数时，prototype属性值会被作为原型复制给所有对象实例；原型对象中有一个constructor属性指向它的构造函数
40. 原型链：本质上就是对象之间的关联关系。每个对象和原型对象都有原型，这种原型层层连接起来就构成了原型链，当查找一个对象的属性时，如果实例中没有改属性，则JS会向上遍历原型链（发生委托行为），直到找到该属性或在到达Object.prototype顶部（Object.prototype.\_\_proto\_\_ == null）仍没有找到该属性时放回undefined



1. 类和继承（以下是面向对象的实现，JS还可以利用对象关联即委托机制来实现类和继承Object.create是关键）
2. 创建类：
3. 工厂模式：在一个函数内创建好对象，然后把对象返回，不需要new关键字（缺点：无法识别对象）
4. 构造函数模式：没有return语句，将属性和方法赋给this对象（缺点：无法实现函数复用）
5. 原型模式：将属性和方法直接添加到构造函数的原型对象上（缺点：所有实例共享相同的属性值）
6. 组合模式：构造函数模式来定义实例属性，原型模式来定义方法和共享属性
7. 动态原型模式：将原型模式放入构造函数中，封装性更好
8. 实现继承：
9. 原型链继承：new一个父类的实例作为子类的原型（缺点：引用类型被实例共享，不能向父类构造函数传参）
10. 构造继承：在子类的构造函数中使用父类构造函数call方法，复制了父类的实例属性和方法给子类（缺点：只能继承父类的实例属性和方法，无法实现函数复用）
11. 实例继承：在子类构造函数中创建父类实例并添加新属性和方法，作为子类实例返回（缺点：实例是父类的实例，只能单继承）
12. 组合继承：结合原型链继承和构造继承（缺点：调用两次父类构造函数，生成两份实例）
13. 寄生组合继承：使用构造继承，在匿名自执行函数中创建空白类，将空白类构造函数原型对象设置为父类构造函数的原型对象，再将空白类实例作为子类的原型对象（缺点：实现比较复杂）
14. 异步编程
15. Event Loop机制：
16. 脚本执行时，如果JS遇到同步代码则按照顺序将之加入执行栈中并执行，如果当前执行的是函数，则向执行栈中添加这个函数的执行环境，并进入这个环境执行代码，当这个执行环境的代码执行完毕，则这个执行环境将出栈，返回上一个环境，这个过程反复进行，直至全部同步代码执行完毕（这里属于宏任务）
17. 当主线程遇到异步任务则将任务放到Event Table中注册，并继续往下执行代码，当这个异步任务了有运行结果时，根据这个异步任务的类型（Macrotask：script、setTimeout和setInterval；Microtask：Promise），JS将会把它放入相应的Task Queue中
18. 当执行栈中的所有同步代码执行完毕，JS会查看微任务队列是否有任务存在，如果有则将队列中的任务入栈并执行直至队列为空，
19. GUI接管线程，检查GUI任务队列，如果需要渲染UI则渲染UI（document是否需要更新、是否有resize、scroll事件触发、是否有requestAnimationFrame回调，但是如果脚本执行时间超过16.7ms会造成跳帧，可以用setTimeout(0)解决）
20. 开始下一轮Event Loop，JS引擎查看宏任务队列并取出队列中最前面的一个任务，入栈并始执行
21. 异步编程的四种方式：回调函数、事件监听、发布/订阅、Promise对象
22. 异步加载JS文件
23. async属性：JS文件一旦可用就会执行，但是文件的加载是并行进行的
24. defer属性：文件的加载是并行进行的，但是JS文件的执行要在所有元素解析完成之后
25. onload时动态创建DOM
26. DOM操作API
27. 查找节点：getElementById()、getElementsByTagName()、getElementsByClassName()、querySelector()、querySelectorAll()
28. 创建节点：createElement()、createTextNode()、createDocumentFragment()
29. 修改节点：appendChild()、removeChild()、insertBefore()、replaceChild()
30. 节点关系：parentNode、childNodes、children、firstChild、lastChild、firstElementChild、lastElementChild、previousSibling、nextSibling、previousElementSibling、nextElementSibling
31. 节点属性：getAttribute()、setAttribute()
32. 节点样式：element.style、element.classList、getComputedStyle()、getBoundingClientRect()
33. DOM事件与事件委托机制
34. 事件流：事件流描述的是从页面中接收事件的顺序，一个DOM事件流分为三个阶段：事件捕获阶段 --> 处于目标阶段 --> 事件冒泡阶段，此外还有有捕获和冒泡两种事件流,默认情况下冒泡流
35. event对象：event.currentTarget是当前事件处理程序所属的元素，即事件处理程序中this的值、event.target事件产生的目标元素、event.preventDefault()阻止事件默认行为、event.stopPropagation取消事件冒泡
36. 事件委托：利用事件冒泡原理，将事件绑定在目标元素的父节点上（优点：减少事件注册，节省内存；无需对新增子节点绑定事件）
37. 事件API：element.addEventListener(type, callback, useCapture)以及element.onclick
38. JS自定义事件
39. 创建事件：var myEvent = document.createEvent(eventType);（可选值HTMLEvents、MouseEvents、UIEvents）
40. 初始化事件：myEvent.initEvent(eventName, canBubble, preventDefault);
41. 触发事件：element.dispatchEvent(myEvent);
42. AJAX的原理及优缺点
43. 概念：AJAX（异步JS和XML）是一种用于创建交互式动态网页的技术，通过在后台与服务器进行少量的数据交换，使网页实现异步更新
44. XMLHttpRequest对象：
45. 创建：var xhr = window.XMLHttpRequest ? new XMLHttpRequest() :

new ActiveXObject(“Microsof.XMLHTTP”);

1. 属性：xhr.readyState——0-未初始化、1-连接已建立、2-请求已接受、3-请求处理中、4-请求已完成；xhr.status——200-OK、404-Not Found；xhr.responseText；xhr.responseXML
2. 事件：xhr.onreadystatechange
3. 方法：xhr.open(method, url, async)、xhr.send(string)、xhr.setRequestHeader()、xhr.getResponseHeader()
4. 步骤：创建XMLHttpRequest对象 -> 判断数据传输方式（GET/POST） -> 打开链接 -> 发送 -> 监听XMLHttpRequest对象的readyState并判断status状态码执行回调函数
5. 优点：页面无需刷新，在页面内与服务器通信，用户体验好；异步方式不需要打断用户的操作，响应能力更好；按需请求数据，减少服务器负担和带宽；标准化且广泛支持
6. 缺点：无法使用后退按钮回退；对搜索引擎支持弱；不支持移动设备；存在安全问题
7. XHR2.0
8. 超时设置：xhr.timeout和xhr.ontimeout
9. 新增FormData对象: var fd = new FormData(); fd.append(age, 11)
10. 进度信息：xhr.onprogress下载进度及xhr.upload.onprogress上传进度（var percentComplete = event.loaded / event.total）
11. 新增事件：load、abort、error、loadstart及loadend事件
12. BOM
13. 概念：Browser Object Model即浏览器对象模型，BOM提供了JS与浏览器交互的对象（BOM包含了DOM）
14. window对象（window既是JS访问浏览器的接口，也是JS中的全局对象）：
15. navigator对象:cookieEnabled-是否启用cookie、userAgent-请求报文user-agent首部值、geolocation地理位置、onLine设备在线情况
16. location对象：protocol-协议、host-主机名+端口、port-端口、search-查询字段、reload()-重新加载页面、replace(url)-替换当前页面
17. history对象：back()-返回到前一个URL、forward-前进到下一个URL、go()-返回到某个页面
18. screen对象：height屏幕像素高度、width屏幕像素宽度
19. 窗口控制：open(ulr, name, style)打开一个新的窗口、close()关闭窗口、scrollTo(x, y)窗口滚动到指定位置、resizeTo(x, y)窗口大小调整至
20. 定时器：setTimeout()、clearTimeout()、setInterval()、clearInterval()
21. 对话框：alert、confirm、prompt
22. ES6新特性
23. const和let关键字
24. 默认参数值
25. 箭头函数
26. 拓展运算符
27. 解构赋值
28. 模板字符串
29. 类
30. Promise
31. 对象字面增强
32. 模块
33. Map和Set
34. 箭头函数与普通函数的区别
35. 不绑定this，自己捕获上下文的this值（取决它外面第一个不是箭头函数的函数的this）
36. 不绑定arguments，可以用reset参数解决
37. 不能作为构造函数
38. 没有原型属性
39. Promise对象
40. 回调缺点：
41. 回调函数的异步流程的方式是非线性的、非顺序的，这使得正确推导这样的代码难度很大，程序员难于理解的代码，且错误很难捕捉
42. 回调会受到控制反转的影响，因为回调暗中把控制权交给第三方来调用你的代码，从而导致信任问题，比如回调被调用的次数是否会超出预期

（2）概念：一种新的异步编程方式，Promise在回调代码和将要执行这个任务的异步代码之间提供了一种可靠的中间机制来管理回调。Promise构造函数接受一个以resolve和reject为参数的函数作为参数，返回一个Promise对象，包含panding、resolved和rejeced三种状态，对象的状态不受外界的影响，且一旦状态改变就不会再变化，任何时刻都可以得到这个结果，可以实现扁平链式地执行异步编程。

（3）Promise API：

1. new Promise((resolve, reject) => {})构造函数，reject()就是拒绝这个promise；如果传给resolve()的是一个非 Promise、非 thenable 的立即值，这个promise就会用这个值完成，如果传给resolve()的是一个真正的Promise或thenable值，这个值就会被递归展开，并且要构造的promise将取用其最终决议值或状态
2. Promise.reject()创建一个已经被拒绝的Promise，Promise.resolve()创建一个已经完成的Promise，但可能是完成也可能是拒绝
3. then(resolve, reject)以及catch(reject)
4. Promise.all()以及Promise.race()

（4）Promise实现：

i.用setTimeout来延迟resolve和reject的执行

ii.加入状态：pending、resolved、rejected

iii.实现链式调用——处理返回值是不是Promise对象两种情况

19.模块化

（1）目的：CommonJS是同步加载模块，当要用到模块时现加载现用，无法解决以下两个问题，因而RequireJS和SeaJS分别提出了AMD和CMD规范：

i.多个JS文件之间可能有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器

ii.JS加载的时候浏览器会停止页面渲染，加载文件越多，页面失去响应时间越长

（2）AMD（Asynchronous Module Definition异步模块定义）：

i.定义模块：define(id?, dependencies?, factory)

ii.加载模块：require([dependencies], function() {})

（3）CMD（Common Module Definition通用模块定义）：

i.定义模块：define(function(require, exports, module))（在CMD中不在define中使用id和dependencies，依赖的模块在factory中用require定义）

ii.加载模块：seajs.use([dependencies], function() {})

（4）区别：

i.AMD推崇依赖前置，CMD推崇就近依赖

ii.AMD和CMD都会预加载依赖模块，但是CMD需要解析factory函数中的require关键字才能确定依赖

iii.最大区别是对依赖模块的执行时机处理不同：AMD是提前执行，各个依赖文件异步加载完便执行，所有依赖文件加载执行完才执行require的回调函数；CMD是延迟执行，所有依赖文件加载完后不执行，在require的函数中，被require时才执行

iv.AMD用户体验好，CMD性能好

**HTTP：**

1. HTTP报文
2. 请求报文：请求行 + 请求头 + 请求体

POST /index.html HTTP/1.1

HOST: www.demo.com

User-Agent: Mozilla/5.0 (Windows NT 6.1;rv:15.0) Firefox/15.0

username=admin&password=123

1. 响应报文：状态行 + 响应头 + 响应体

HTTP/1.1 200 OK

Content-Encoding: gzip

Content-Type: text/html;charset=utf-8

<!DOCTYPE html>

<html lang=”en”>

...

</html>

1. HTTP方法：GET、POST、HEAD、OPTIONS、PUT、DELETE、TRACE、CONNECT
2. 常见请求首部：Host、User-Agent、Referer、Cookie、Accept、Accept-Language、If-None-Match、If-Modified-Since
3. 常见响应首部：Server、Location、Set-Cookie、ETag、Cache-Control
4. 常见实体首部：Content-Type、Content-Length、Content-Encoding、Expires、Last-Modified
5. 常见HTTP状态码

（1）1XX信息：100 -- 继续

（2）2XX成功：200 -- OK；204 -- 没有内容；

（3）3XX重定向：301 -- 永久重定向；302 -- 临时重定向；304 -- Not modified；

1. 4XX客户端错误：400 -- Bad Request；401 -- 认证失败；403 -- Forbidden；404 -- Not Found

（5）5XX服务器错误：503 -- 服务器超载

1. get请求和post请求的区别
2. get用于获取数据，post用于修改数据
3. get参数通过url传递，post则放在request body中
4. get在url的参数大小为2K，post没有大小限制
5. get只接受ASCII字符参数，只进行url编码，post没有类型限制，支持多种编码方式（form-data、json）
6. get参数直接暴露在url，安全性较差（抓包是一样的）
7. get请求会被缓存，post不会
8. get在浏览器回退时是无害的，而post会再次提交请求
9. get产生一个TCP数据包，post产生两个TCP包
10. 两者都基于TCP/IP没有本质区别
11. HTTP缓存机制
12. 强缓存：在第一次请求中，服务器在响应header中设置了Expires/Cache-Control字段标明失效规则，在下次请求时，若缓存为失效则客户端直接使用缓存，否则重新请求数据，并相应地缓存数据
13. 协商缓存：在第一次请求中，服务器在响应header中设置了ETag/Last-Modifeed字段设置缓存标志，在下次请求时，在缓存中获取缓存标志，在请求header中设置If-None-Match/If-Modified-Since字段验证缓存是否失效，若缓存未失效，则服务器放回状304报文，客户端直接使用缓存，若缓存失效，则服务器直接返回200及数据，客户端相应地缓存数据
14. 当浏览器强缓存过期后就会触发协商缓存
15. HTTPS与HTTP
16. HTTPS：在HTTP应用层的基础上加入SSL层，实现安全版的HTTP
17. SSL主要机制：证书+非对称密钥+会话加密+消息校验
18. SSL动作：验证服务器端 --> 选择双方都支持的加密算法 --> 使用公钥加密技术加密数据 --> 创建一个加密是SSL连接 --> 基于该SSL连接传输HTTP请求
19. 区别：
20. HTTP明文传输数据，HTTPS会对传输的数据进行加密
21. HTTPS需要付费申请证书
22. HTTP使用80端口，HTTPS使用443端口

HTTPS协议由SSL+HTTP协议构建，会进行身份验证、加密传输和信息完整性校验，比HTTP协议安全

1. HTTP2.0新特性
2. 二进制分帧传输
3. 首部压缩
4. 多路复用
5. 请求优先级
6. 服务器推送

**其他：**

1. 浏览器内核
2. IE：Trident内核
3. Firefox：Gecko内核
4. Safari：Webkit内核
5. Opera、Chrome：Blink内核
6. 会话跟踪方式：cookie、session、url重写、隐藏input、ip地址
7. JS动画 VS CSS动画
8. JS动画占用主线程，受到主线程中其他任务的干扰可能导致线程阻塞而丢帧
9. JS动画代码复杂度高，CSS动画代码逻辑相较简单
10. JS代码的控制能力强，兼容性好且能实现的动画效果比CSS动画丰富
11. CSS动画如果不触发layout或paint的话，只会触发在渲染线程的合成器线程，而JS动画占用渲染线程的主线程，相较CSS动画性能更好
12. JS脚本可以为通过click事件为元素添加class样式而使得元素开始CSS动画，同时animation提供了animationstart、animationend及animationiteration事件API来控制CSS动画
13. 优雅降级和渐进增强
14. 优雅降级：一开始就构建站点的完整功能，然后针对浏览器进行测试和修复，比如一开始使用 CSS3 的特性构建了一个应用，然后逐步针对各大浏览器进行 hack 使其可以在低版本浏览器上正常浏览。
15. 渐进增强：一开始就针对低版本浏览器进行构建页面，完成基本的功能，然后再针对高级浏览器进行效果、交互、追加功能达到更好的体验。
16. MVC和MVVM的区别
17. MVC（Model-View-Controller）：用户对View的操作交给Controller处理，在Controller中响应View的事件并调用Model的接口对数据进行操作，一旦Model发生变化便通知View进行更新
18. MVVM（Model-View-ViewModel）：在Model和View之间没有直接的联系，但Model和View与ViewModel之间是双向的，通过双向数据绑定，实现了View和Model的同步
19. 区别：MVVM简化了View和Model的依赖，耦合度底，而且解决了数据频繁更新时，大量DOM操作使得页面渲染性能降低，影响用户体验的问题
20. 设计模式
21. 发布订阅模式：大大降低程序模块之间的耦合度，便于拓展和维护
22. 工厂模式：工厂模式定义一个用于创建对象的接口，这个接口由子类决定实例化哪一个类
23. 中介者模式
24. 代理模式
25. 单例模式
26. 跨域
27. 概念：请求不同源（协议、域名、端口）的资源时将受到同源策略的限制

（2）CORS：在AJAX中使用绝对路径即可，对于简单请求，客户端为请求配置Origin头部，服务器根据这个值绝对是否同意这次请求，如果服务器返回的响应首部没有Access-Control-Allow-Orign字段，客户端抛出一个错误给XMLHttpRequest对象；对于非简单请求，客户端配置Origin头部，用OPTIONS方法进行一次预检，服务器同意后，接下来的请求和简单请求一样

（3）JSONP：script标签不受同源策略的限制，用script标签从服务器请求数据，服务器放回一个带有方法和数据的JS代码（缺点：无法发送POST请求，不容易确定是否失败）

（4）服务器代理：利用反向代理技术，保证客户端和第三方服务是同源的。

（5）document.domain：把document.domain设置为自身或更高一级的父域，进行父子window之间的通信

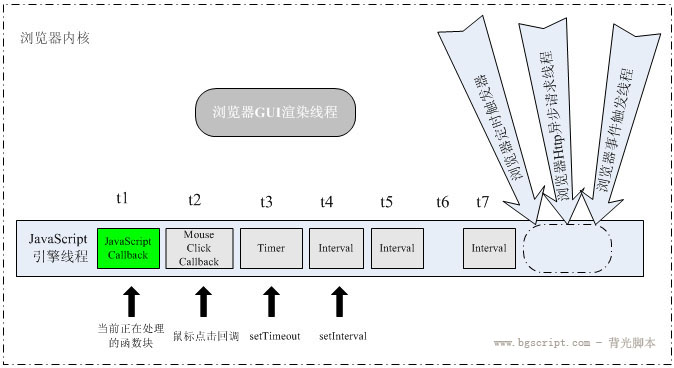
（6）window.name+iframe：用iframe请求页面，读取其中window.name的信息

（7）postMessage：H5新增概念，用postMessage和onmessage进行window之间的通信

1. Web安全

（1）XSS（跨站脚本）：

1. 概念：攻击者通过注入非法的html标签或JS代码，从而在用户浏览网页时，对用户浏览器进行控制或者偷取用户隐私数据，分为反射型、存储型和DOM型
2. 防范：设置cookie的httpOnly属性；检查、过滤和转义用户的输入和服务端输出
3. CSRF（跨站请求伪造）:
4. 概念：冒充受信任的用户向服务器发送非预期的请求，完成一些违背用户意愿的操作
5. 防范：验证码；检查referer首部字段；在cookie中添加token验证
6. SQL注入：
7. 概念：把SQL命令插入请求的查询字符串中，欺骗服务器执行恶意的SQL命令
8. TCP与UDP的区别
9. TCP面向链接（TCP握手）；UDP是无连接的，即发送数据之前不需要建立链接
10. TCP提供可靠的服务，即通过TCP传送的数据，无差错、不丢失、不重复，且按序到达；UDP尽最大努力交付，即不保证可靠交付
11. TCP面向字节流，具有拥塞控制；UDP面向报文，没有拥塞控制
12. TCP连接只能是点到点的；UDP支持多种
13. TCP首部开销20字节；UDP首部开销8字节
14. offset、client和scrollHeight的区别
15. offsetHeight返回值包含content+padding+border；
16. clientHeight返回值只包含content+padding，如果有滚动条也不包含滚动条；
17. scrollHeight返回值包含content+padding+溢出内容的尺寸
18. 虚拟DOM
19. 通过JS对象来模拟DOM对象：
20. 定义tag、props和children等属性
21. 实现一个JS对象转DOM对象的的createElement内部函数及render函数
22. 当render函数被调用时，判断前后虚拟DOM的差异：
23. 树的递归遍历
24. 判断属性的更改
25. 判断节点列表差异
26. 判断子元素差异
27. 局部更新，渲染差异
28. 浏览器渲染原理
29. 每个DOM上的可视化节点对应一个LayoutObject，这个LayoutObject知道如何在屏幕上绘制该节点，LayoutObject也呈现树的结构，也就是渲染树
30. 为满足形成层叠上下文的LayoutObject创建渲染层PaintLayer，一般而言，拥有相同坐标空间的LayoutObject同属于一个渲染层，此外会为一些特殊的LayoutObject创建自己的渲染层：
31. 有fixed、absolute明确的定位属性
32. 透明度opacity小于1
33. CSS filter滤镜
34. CSS transform
35. 有对于opacity、transform、filter的动画
36. overflow不为visible
37. 某些特殊的渲染层会被认为是合成层（合成层的位图会交由GPU合成，比CPU快；当需要重绘时，只需要重绘本身，不会影响其他层；对于transform和opacity效果不会触发重排和重绘，因而提升合成层对于提升页面性能有很大的帮助，设置will-change属性是最好的提升合成层方法），而拥有单独的GraphicsLayer，而其他不是合成层的渲染层，则和第一个拥有GraphicsLayer的父层公用一个，而每个GraphicsLayer都有一个GraphicsContext，GraphicsContext会负责输出该层的位图，上传到GPU中，最后由GPU将多个位图合并，然后draw到屏幕上。渲染层提升为合成层：
38. 硬件加速的iframe元素
39. video元素
40. 3D或硬件加速的2D Canvas元素
41. 3D transform
42. 有对于opacity、transform、filter的动画
43. will-change设置opacity、transform、top、left等
44. overlap重叠（这可能会随随便便就导致大量的合成层，导致层爆炸而消耗CPU和内存，因为浏览器有自己的层压缩机制）
45. 浏览器线程
46. 浏览器的内核是多线程的，这些线程在内核的控制下通过Event Loop异步机制相互配合以保持同步。一个浏览器至少实现五个常驻线程：javascript引擎线程，GUI渲染线程，浏览器事件触发线程、定时器触发线程、http异步请求线程（<img/> <link/>这类请求）
47. javascript引擎是基于事件驱动单线程执行的，JS引擎一直等待着任务队列中任务的到来，然后加以处理，浏览器无论什么时候都只有一个JS线程在运行JS程序
48. 当界面需要重绘或由于某种操作引发回流时,GUI线程就会执行。但需要注意 GUI渲染线程与JS引擎是互斥的，当JS引擎执行时GUI线程会被挂起，GUI更新会被保存在一个队列中等到JS引擎空闲时立即被执行
49. 当一个事件被触发时该线程会把事件添加到待处理队列的队尾，等待JS引擎的处理。这些事件可来自JavaScript引擎当前执行的代码块如setTimeout、也可来自浏览器内核的其他线程如鼠标点击、AJAX异步请求等，但由于JS的单线程关系所有这些事件都得排队等待JS引擎处理。



1. 前端SEO
2. 书写语义化的HTML代码，符合W3C规范
3. 设置合理的title、description和keywords
4. 重要内容的HTML代码放在前面：搜索自上而下抓取，且抓取长度有限
5. 重要内容不要用JS输出：搜索引擎不会抓取其中的内容
6. 少用iframe：搜索引擎不会抓取其中的内容
7. 非装饰性的图片必须添加alt书写
8. 提高网站速度：网站速度是搜索引擎排序的一个重要指标
9. Web性能优化
10. 内容方面：
11. 减少HTTP请求：合并JS和CSS、CSS-Sprite、图片懒加载
12. 减少DNS查询：进行DNS缓存、恰当数量的主机名
13. 减少资源体积：JS混淆压缩、CSS压缩、图片压缩
14. 减少DOM数量：减少不必要内容、分页、按需加载
15. 网络方面：
16. 使用CDN；
17. 配置ETag/Last-Modified，利用HTTP缓存
18. 使用可缓存的AJAX
19. Gzip压缩
20. 减少cookie大小
21. CSS方面：
22. 从外部引入样式表，放到页面顶部
23. 不适用CSS表达式
24. 使用link而不使用@import
25. 减少重排重绘
26. CSS选择符从右往左匹配，避免深度过深，权重够了即可
27. JS方面：
28. 从外部引入脚本，放到页面底部
29. 删除不需要的脚本
30. 减少DOM操作
31. 图片方面：
32. 选择合适的图片格式如WebP
33. 用SVG或Canvas代替图片
34. 使用iconfont



1. URL请求过程
2. 在浏览器地址栏输入URL
3. 浏览器查看缓存：
4. 如果请求资源未缓存，发起新请求
5. 如果请求资源在缓存中且新鲜，则读取缓存，并跳转到解码步骤
6. 如果请求资源在缓存中但已过期，则取出缓存中的Etag或Last-modify首部信息与服务器进行验证
7. 浏览器解析URL，获取协议、主机、端口和路径
8. 浏览器组装一个HTTP请求报文
9. 浏览器获取主机IP地址即进行DNS：
10. 浏览器缓存
11. 本机缓存
12. 路由器缓存
13. ISP DNS缓存
14. DNS递归查询
15. 打开一个socket与该IP及端口建立TCP链接，TCP握手：
16. 浏览器发送SYN=1，Seq=X的包到服务器端口
17. 服务器返回SYN=1，ACK=X+1，Seq=Y的响应包
18. 浏览器发送ACK=Y+1，Seq=Y的包
19. TCP链接建立后在这条TCP上发送HTTP请求
20. 服务器接受请求，并将请求转发给服务程序解析
21. 服务器检查HTTP请求头是否包含缓存验证信息，如果验证缓存新鲜，则返回304等对应状态码
22. 服务器将响应报文通过TCP链接发送给浏览器
23. 浏览器接受HTTP响应，然后根据情况选择关闭或保留重用TCP，TCP挥手：
24. 主动发发送Fin=1，ACK=Z，Seq=X包
25. 被动方发送ACK=X+1，Seq=Z包
26. 被动方发送Fin=1，ACK=X，Seq=Y包
27. 主动发发送ACK=Y，Seq=X包
28. 浏览器检查响应状态码，3XX、4XX、5XX的处理情况与2XX不同
29. 浏览器将资源缓存进内存中，再根据响应首部进行硬盘缓存
30. 浏览器对资源进行解码
31. 浏览器根据资源类型绝对如何处理，假设资源为HTML文档
32. 渲染引擎开始自上而下解析HTML文档，构建DOM树：字符流解析 -> 词法分析 -> 节点组成DOM树
33. 解析HTML文档过程中遇到图片、样式表启动http异步请求线程下载资源，不会阻塞HTML文档的解析
34. 下载完CSS文件后开始构建CSSOM树：字符流解析 -> 词法分析 -> 节点组成CSSOM树
35. 根据DOM树和CSS树构建渲染树：遍历DOM树可见节点构建渲染树 -> 布局及重排 -> 绘制及重绘
36. 下载完JS文件后开始JS引擎解析JS文件：语法检查 -> 预编译 -> 执行代码
37. JS执行过程中可能会再次引发渲染树的一系列的重排、重绘
38. 将前面进行一系列布局绘制的渲染层根据条件提升会合成层，再由该合成层所属的图象层生成位图交给GPU和并各个位图
39. 最后在屏幕上显示页面（HTML解析过程中会逐步显示页面）
40. H5及CSS3新特性
41. H5新特性：
42. nav、header、footer、article、section等新的语义化标签
43. 表单2.0（新标签、新属性）
44. 音频和视频
45. 拖放API
46. 绘画Canvas
47. 地理位置Geoloaction
48. 跨窗口通信postMessage
49. 跨域资源共享CORS
50. 离线应用manifest
51. 本地存储WebStorage
52. 多线程WebWorker
53. 全双工通信协议WebSocket
54. CSS3新特性：
55. :first-child和nth-of-type等新的选择器
56. 颜色rgba
57. 文本效果word-wrap
58. 字体@font-face
59. 圆角border-radius
60. 阴影box-shadow、text-shadow
61. 盒子模型box-sizing
62. 背景background-origin、backgro-clip
63. 渐变linear-gradient、radial-gradient
64. 弹性盒flex
65. 转换transform
66. 过渡transition
67. 动画animation
68. 媒体查询@media

**前端人员核心素质：**

1. 注重用户的体验——美感 + 代码能力，即你写的东西好看吗？好用吗？
2. 学习新兴的技术——技术迭代更新，新框架能带来可维护性、性能更好的产品
3. 与他人的交流——对美工：这个实现不了or这个我们只能渐进增强，对后台：这个接口文档要改一下
4. typeof的字符串返回值
5. undefined —— 如果这个值未初始化或未声明（可以用来阻止报错）
6. boolean —— 如果这个值为布尔值
7. string —— 如果这个值是字符串
8. number —— 如果这个值是数值，包括NaN
9. object —— 如果这个值是对象，包括Null、Array、Date
10. function —— 如果这个值是函数
11. 重写原型对象需要自己配置constructor属性，而且重写原型对象切断了现有原型与任何之前已经存在的对象实例之间的联系，它们引用的仍然是最初的原型
12. for循环头部的let不仅将i绑定了到for循环的块中，事实上它将其重新绑定到了的每一个迭代中，确保使用上一个循环迭代结束时的值重新进行赋值
13. JS内建函数：String、Number、Boolean、Object、Array、Function、Date、RegExp、Error
14. Object.prototype.toString.call(obj)可以取出对象内部的[[Class]]属性