一、CSS

1、介绍一下标准的CSS的盒子模型？与低版本IE的盒子模型有什么不同的？

标准盒子模型：宽度=内容的宽度（content）+ border + padding + margin

低版本IE盒子模型：宽度=内容宽度（content+border+padding）+ margin

2、box-sizing属性？

用来控制元素的盒子模型的解析模式，默认为content-box
context-box：W3C的标准盒子模型，设置元素的 height/width 属性指的是content部分的高/宽
border-box：IE传统盒子模型。设置元素的height/width属性指的是border + padding + content部分的高/宽

3、 CSS选择器有哪些？哪些属性可以继承？

CSS选择符：id选择器(#myid)、类选择器(.myclassname)、标签选择器(div, h1, p)、相邻选择器(h1 + p)、子选择器（ul > li）、后代选择器（li a）、通配符选择器（*）、属性选择器（a[rel=”external”]）、伪类选择器（a:hover, li:nth-child）
4、请回答，css代码，有几种常见的引入方式，分别是什么？

5、display:none与visibility：hidden的区别？

display：none 不显示对应的元素，在文档布局中不再分配空间（回流+重绘）
visibility：hidden 隐藏对应元素，在文档布局中仍保留原来的空间（重绘）

6、使用 CSS 预处理器吗？

7、三栏布局

8、有一个高度自适应的div，里面有两个div，一个高度100px，希望另一个填满剩下的高度

外层div使用position：relative；高度要求自适应的div使用position: absolute; top: 100px; bottom: 0; left: 0

二、HTML

1、什么是语义化？为什么需要语义化

2、元素包含关系

以前：块级元素可以包含行内元素，行内元素不可以包含块级元素，a元素除外

现在：元素的包含关系由元素的内容类别决定。

三、JavaScript

1、写出JavaScript语言typeof可能返回的结果

2、简述call和apply方法的用途、区别

3、为一个数组去重 `let arr = ['a', 'b', '234', '23', 'a', 'b'];`

4、运行 `test()` 和 `new test()` 的结果是什么？

```
var a = 5;
function test () {
  a = 0;
  alert(a);
  alert(this.a);
  var a;
  alert(a);
}
```

5、一串连续的数字，请实现打点功能。如：已知1000000000转换成1.000.000.000（用正则做可得满分，其他方法酌情给分）

6、Javascript作用链域?

全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节。 当需要从局部函数查找某一属性或方法时，如果当前作用域没有找到，就会上溯到上层作用域查找， 直至全局函数，这种组织形式就是作用域链。

7、new一个对象的过程

①、创建(或者说构造)一个全新的对象

②.这个新对象会被执行[Prototype]连接到构造函数的原型

③.将构造函数的作用域赋给新对象（因此 this 就指向了这个新对象）；

④.执行构造函数中的代码（为这个新对象添加属性）。如果函数没有显式返回其他对象,那么new表达式中的函数调用会自动返回这个新对象。

8、JavaScript中垃圾回收机制?

9、请你简述一下`childNodes`和`children`的区别

`childNodes` 查找直系子节点（不包括孙节点），除了元素节点，还会加上元素节点之间的文本节点、属性节点、注释节点等等。

`children` 只返回当前元素的元素字节点

10、`querySelector`、`querySelectorAll()`与其他DOM选择器方法选出的DOM类数组有什么不同吗?

`querySelector`、`querySelectorAll()`等`CSS`规则选择器方法选出的DOM类数组是静态的，在后续是不可改变的。而其他的DOM选择器是实时的。

11、暂时性死区

只要块级作用域内存在`let`或`const`命令，它所声明的变量就绑定了这个区域，不再受外部影响。这个区块对这些命令声明的变量形成了封闭的作用域，总之在代码块内，使用`let`或`const`命令声明变量前，该变量都是不可用的。

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

12、写出下面代码的打印结果，并说明原因

```
function fn (a, b) {
  argument[0] = 1;
  console.log(a);
}
fn(2, 1);
```

13、

```
function foo(x = 5) {
  let x = 1;
}
```

四、操作系统

1、操作系统的功能

2、中断技术和分时技术

CPU收到外部信号（中断信号）后，停止当前工作，转去处理外部事件，处理完毕后，回到原来工作的中断处（断点）继续原来的工作。

- 主机以很短的“时间片”为单位，把CPU轮流分配给每个终端使用，直到全部作业被运行完。
- 由于时间片很短，在终端数量不多的情况下，每个终端都能很快获得CPU，使得每个终端都能及时响应。
- 等待周期 = 时间片 X 终端数量

3、段式系统的缺点

- 段需要连续的存储空间
- 段的最大尺寸受到内存大小的限制
- 在辅存中管理可变尺寸的段比较困难

4、虚拟内存管理方法

- 页式虚拟存储管理
- 段式虚拟存储管理
- 段页式虚拟存储管理

5、进程切换

- 概念：内核挂起当前CPU上的进程并恢复之前挂起的某个进程
- 与中断上下文的切换有差别
    - 中断前后在同一进程上下文中，只是用户态转向内核态执行

6、死锁的概念

- 两个或者多个进程无限期地等待永远不会发生的条件的一种系统状态（结果：每个进程都永远阻塞)
- 在两个或多个进程中，每个进程都持有某种资源，但又继续申请其他进程已持有的某种资源。此时每个进程都拥有其运行所需的一部分资源，但是又都不够，从而每个进程都不能向前推进，陷入阻塞状态。这种状态称为死锁。

7、进程通信（管道通信和信号通信机制）

管道是进程间的一种通信机制。一个进程（A）可以通过管道把数据传输给另外一个进程（B）。前者（A）向管道输入数据，后者（B）从管道读取数据。

8、同步和互斥

进程的互斥关系

- 多个进程由于共享了**独占性资源**，必须协调各进程对资源的存取顺序：确保没有任何两个或以上的进程同时进行存取操作

进程的同步关系

- 若干**合作进程**为了完成一个共同的任务，需要相互协调运行步伐：**一个进程**开始某个操作**之前**必须要求**另一个**进程已经完成某个操作，否则前面的进程只能**等待**

五、计算机网络

1、tcp的三次握手

1. 第一次握手：建立连接时,客户端发送**syn包(syn=j)**到服务器,并进入**SYN_SEND**状态,等待服务器确认； SYN：同步序列编号(Synchronize Sequence Numbers)
2. 第二次握手：服务器收到syn包,必须确认客户的**SYN（ack=j+1）**,同时自己也发送一个**SYN包（syn=k）**,即**SYN+ACK包**,此时服务器进入SYN_RECV状态；
3. 第三次握手：客户端收到服务器的SYN + ACK包,向服务器发送**确认包ACK(ack=k+1)**,此包发送完毕,客户端和服务器进入ESTABLISHED状态,完成三次握手.完成三次握手,客户端与服务器开始传送数据

2、http的缺点

（1）通信使用明文(不加密),内容可能会被窃听

（2）不验证通信方的身份,因此有可能遭遇伪装

（3）无法证明报文的完整性,所以有可能已遭篡改

3、HTTP缓存

当客户端向服务器请求资源时，会先抵达浏览器缓存，如果浏览器有“要请求资源”的副本，就可以直接从浏览器缓存中提取而不是从原始服务器中提取这个资源。

4、持久连接（Connection：keep-alive）

- 特点：只要任意任意一端没有明确提出断开连接，则保持TCP连接状态。
- 好处：在于减少了TCP连接的重复建立和断开所造成的额外开销。（在http/1.1中，所有连接默认都是持久连接）

5、使用cookie的状态管理

Cookie会根据从服务端发送的响应报文内的一个叫做**Set-Cookie**的首部字段信息，通知客户端**保存Cookie**。当下次客户端再往该服务器发送请求时，客户端会自动**在请求报文中加入Cookie值（键值对形式）**后发送出去。服务端发现客户端发送过来的Cookie后，会去检查究竟是从哪一个客户端发送过来的连接请求，然后对比服务器上的记录，最后得到之前的状态信息。

6、公开秘钥加密

公开密钥加密使用一对非对称的密钥。一把叫做私有密钥pnvateky).另一把叫做公开密钥(publickey)。顾名思义,私有密钥不能其他任何人知道,而公开密钥则可以随意发布,任何人都可以获得。使用公开密钥加密方式,发送密文的一方使用对方的公开密钥进行加密处理,对方收到被加密的信息后,再使用自己的私有密钥进行解密。利用这种方式,不需要发送用来解密的私有密钥,也不必担心密钥被攻击者窃听而盗走。

7、简述一下TCP和UDP协议

可靠、按序的交付服务

- 拥塞控制
- 流量控制

- 连接控制

- 基于“尽力而为”的网络层IP协议，没有做（可靠性方面的）扩展
  - 复用/分用
  - 简单的错误校验
- `Best effort`服务，UDP段可能
  - 丢失
  - 非按序到达
- 无连接
  - UDP发送方和接收方之间不需要握手
  - 每个UDP段的处理独立于其他段

8、UDP的优点

- 无需建立连接（减少延迟）
- 实现简单：无需维护连接状态
- 头部开销少
- 没有拥塞控制：应用可更好地控制发送时间和速率

六、数据结构

1、数组和链表的区别。

从逻辑结构上来看，数组必须实现定于固定的长度，不能适应数据动态增减的情况，即数组的大小一旦定义就不能改变。当数据增加是，可能超过原先定义的元素的个数；当数据减少时，造成内存浪费；链表动态进行存储分配，可以适应数据动态地增减的情况，且可以方便地插入、删除数据项。

从内存存储的角度看；数组从栈中分配空间（用new则在堆上创建），对程序员方便快速，但是自由度小；链表从堆中分配空间，自由度大但是申请管理比较麻烦。

从访问方式类看，数组在内存中是连续的存储，因此可以利用下标索引进行访问；链表是链式存储结构，在访问元素时候只能够通过线性方式由前到后顺序的访问，所以访问效率比数组要低。

2、一个栈的输入序列为123、、、、、n，若输出序列的第一个元素是n，输出第i（1<=i<=n）个元素是（）

```
n-i+1
```

3、数组元素的地址计算与数组的存储方式无关。请问这句话的说法是正确的吗？（否）

维数组存储方式可以分为行优先和列优先俩种，所以数组的地址计算和数组的存储方式有关。

4、列举数据结构中的线性结构?

线性表（顺序存储）、链表（链式存储）、栈、队列

5、**解决哈希冲突的方法**

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。

1） 线性探测法

2） 平方探测法

3） 伪随机序列法

4） 拉链法

七、算法

1、

2、简述快速排序过程

1）选择一个基准元素,通常选择第一个元素或者最后一个元素,

2）通过一趟排序将待排序的记录分割成独立的两部分，其中一部分记录的元素值均比基准元素值小。另一部分记录的元素值比基准值大。

3）此时基准元素在其排好序后的正确位置

4）然后分别对这两部分记录用同样的方法继续进行排序，直到整个序列有序。

3、图的遍历

- 深度优先搜索遍历(DFS)
- 广度优先搜索遍历(BFS

4、爬楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

5、给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

6、有效括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足:

左括号必须用相同类型的右括号闭合。
左括号必须以正确的顺序闭合。

```
function isValid (s) {
  let obj = {
    ')': '(',
    ']': '[',
    '}': '{'
  };
  const { length } = s;
  let stack = [];
  for (let i = 0; i < length; i++) {
    let current = s[i];
    if (current in obj) {
      // 当栈的长度不为零时，取出栈顶元素赋给comparis
      let comparison = stack.length !== 0 ? stack.pop() : '#';
      if (obj[current] !== comparison) {
        return false;
      }
    } else {
      // 当匹配到左括号时，将该字符压到栈中
      stack.push(current);
    }
  }
```

```
    return stack.length === 0;
}
```

7、给定一个字符串 `s`，找到 `s` 中最长的回文子串。你可以假设 `s` 的最大长度为 1000。

八、数据库

1.主键、外键、超键、候选键

> **超键**：在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。
>
> **候选键**：是最小超键，即没有冗余元素的超键。
>
> **主键**：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。
>
> **外键**：在一个表中存在的另一个表的主键称此表的外键。

2、触发器的作用?

> 触发器是一种特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

3、非关系型数据库和关系型数据库区别，优势比较?

> **非关系型数据库的优势:**
>
> - **性能:** NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。
> - **可扩展性:** 同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。
>
> **关系型数据库的优势:**
>
> - **复杂查询:** 可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。
> - **事务支持:** 使得对于安全性能很高的数据访问要求得以实现。
>
> **其他:**
>
> **1.**对于这两类数据库，对方的优势就是自己的弱势，反之亦然。
>
> **2.**NOSQL数据库慢慢开始具备SQL数据库的一些复杂查询功能，比如MongoDB。
>
> **3.**对于事务的支持也可以用一些系统级的原子操作来实现例如乐观锁之类的方法来曲线救国，比如Redis set nx。

4、**SQL语言共分为四大类:**

- 数据查询语言DQL
- 数据操纵语言DML
- 数据定义语言DDL
- 数据控制语言DCL。

九、框架

1、MVVM定义

MVVM是Model-View-ViewModel的简写。即模型-视图-视图模型。【模型】指的是后端传递的数据。【视图】指的是所看到的页面。【视图模型】mvvm模式的核心，它是连接view和model的桥梁。

2、Vue的生命周期

3、vue中data必须是一个函数

对象为引用类型，当重用组件时，由于数据对象都指向同一个data对象，当在一个组件中修改data时，其他重用的组件中的data会同时被修改；而使用返回对象的函数，由于每次返回的都是一个新对象（Object的实例），引用地址不同，则不会出现这个问题。

4、$route$和 **router的区别**

$route$是“路由信息对象”，包括$path$，$params$，$hash$，$query$，$fullPath$，$matched$，$name$等路由信息参数。而router是“路由实例”对象包括了路由的跳转方法，钩子函数等

5、**v-if 和 v-show 区别**

v-if按照条件是否渲染，v-show是display的block或none

6、computed、watch、methods的区别

7、vue 组件通信

1.父组件与子组件传值

父组件传给子组件：子组件通过props方法接受数据;

子组件传给父组件：$emit方法传递参数

2.非父子组件间的数据传递，兄弟组件传值

eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。也可使用vuex

8、vue中的 ref 是什么?

ref 被用来给元素或子组件注册引用信息。引用信息将会注册在父组件的 $refs 对象上。如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例。

9、vue中数据响应式原理