2021大厂前端核心面试题详解 二

一. 有做过前端加载优化相关的工作吗? 都做过哪些努力

1. 常见的优化手段

   做优化首先要有目的, 即你做优化是为了什么, 是把某个比较关注的指标提高吗?

   页面性能检测: https://developers.google.com/speed/pagespeed/insights/

   1. 只请求当前需要的资源
      异步加载, 懒加载, polyfill的优化 https://polyfill.io/v3/url-builder/
   2. 缩减资源体积
      打包压缩
      gzip
      图片格式优化, 压缩, 根据屏幕分辨率展示不同分辨率的图片
      尽量控制cookie大小
   3. 时序优化
      js中promise.all
      ssr
      prefetch、prerender、preload
      <link rel="dns-prefetch" href="xxxxxx" />
      <link rel="preconnect" href="xxxxxxx" />
      <link rel="preload" as="image" href="xxxxxxxxx" />
   4. 合理利用缓存
      cdn
      http缓存
      localStorage, sessionStorage

2. 如果一段js执行时间非常长, 怎么去分析?

   代码题, 装饰器计算函数执行时间

3. 阿里云oss支持通过链接后拼参数实现图片格式转换, 尝试写一下, 把图片转为webp格式? 需要注意什么?

代码题, 判断浏览器是否支持webp & webp格式转换

4. 如果有巨量的图片需要展示在页面, 除了懒加载这种方式, 还有什么好的方法限制其同一时间加载的数量?

   代码题, 使用promise实现并发控制

二. 平时有关注过前端的内存处理吗?

1. 你了解js中的内存管理吗? 什么情况会导致内存泄露?

1. 内存的生命周期

   内存分配：当我们申明变量、函数、对象的时候，系统会自动为他们分配内存
   内存使用：即读写内存，也就是使用变量、函数等
   内存回收：使用完毕，由垃圾回收机制自动回收不再使用的内存

2. Js中的内存分配

```
const n = 123; // 给数值变量分配内存
const s = “azerty”; // 给字符串分配内存

const o = {
    a: 1,
    b: null
}; // 给对象及其包含的值分配内存
```

3. Js中的内存使用

   使用值的过程实际上是对分配内存进行读取与写入的操作。 读取与写入可能是写入一个变量或者一个对象的属性值，甚至传递函数的参数。

```
var a = 10; // 分配内存
console.log(a); // 对内存的使用
```

4. Js中的垃圾回收机制

垃圾回收算法主要依赖于引用的概念。

在内存管理的环境中，一个对象如果有访问另一个对象的权限（隐式或者显式），叫做一个对象引用另一个对象。

例如，一个Javascript对象具有对它原型的引用（隐式引用）和对它属性的引用（显式引用）。

在这里，“对象”的概念不仅特指 JavaScript 对象，还包括函数作用域（或者全局词法作用域）。

4.1 引用计数垃圾回收

引用计数算法定义“内存不再使用”的标准很简单，就是看一个对象是否有指向它的引用。 如果没有其他对象指向它了，说明该对象已经不再需了。

但它却存在一个致命的问题：循环引用。

如果两个对象相互引用，尽管他们已不再使用，垃圾回收不会进行回收，导致内存泄露。

4.2 标记清除算法

标记清除算法将“不再使用的对象”定义为“无法达到的对象”。 简单来说，就是从根部（在JS中就是全局对象）出发定时扫描内存中的对象。 凡是能从根部到达的对象，都是还需要使用的。 那些无法由根部出发触及到的对象被标记为不再使用，稍后进行回收。

4.2.1 垃圾收集器在运行的时候会给存储在内存中的所有变量都加上标记。

4.2.2 从根部出发将能触及到的对象的标记清除。

4.2.3 那些还存在标记的变量被视为准备删除的变量。

4.2.4 最后垃圾收集器会执行最后一步内存清除的工作，销毁那些带标记的值并回收它们所占用的内存空间。

5. 常见的内存泄露

5.1 全局变量

```
function foo() {
        bar1 = 'some text'; // 没有声明变量 实际上是全局变量 => window.bar1
```

```
        this.bar2 = 'some text' // 全局变量 => window.bar2
    }
    foo();
```

5.2 未被清理的定时器和回调函数

如果后续 renderer 元素被移除，整个定时器实际上没有任何作用。 但如果你没有回收定时器，整个定时器依然有效, 不但定时器无法被内存回收， 定时器函数中的依赖也无法回收。在这个案例中的 serverData 也无法被回收。

```
var serverData = loadData();
    setInterval(function() {
        var renderer = document.getElementById('renderer');
        if(renderer) {
            renderer.innerHTML = JSON.stringify(serverData);
        }
    }, 5000); // 每 5 秒调用一次
```

5.3 闭包

在 JS 开发中，我们会经常用到闭包，一个内部函数，有权访问包含其的外部函数中的变量。 下面这种情况下，闭包也会造成内存泄露

```
var theThing = null;
    var replaceThing = function () {
    var originalThing = theThing;
    var unused = function () {
        if (originalThing) // 对于 'originalThing'的引用
        console.log("hi");
    };
    theThing = {
        longStr: new Array(1000000).join('*'),
        someMethod: function () {
        console.log("message");
        }
    };
```

```
    };
    setInterval(replaceThing, 1000);
```

这段代码，每次调用 replaceThing 时，theThing 获得了包含一个巨大的数组和一个对于新闭包 someMethod 的对象。 同时 unused 是一个引用了 originalThing 的闭包。

这个范例的关键在于，闭包之间是共享作用域的，尽管 unused 可能一直没有被调用，但是 someMethod 可能会被调用，就会导致无法对其内存进行回收。 当这段代码被反复执行时，内存会持续增长。

5.3 DOM引用

很多时候, 我们对 Dom 的操作, 会把 Dom 的引用保存在一个数组或者 Map 中。

```
var elements = {
        image: document.getElementById('image')
    };
    function doStuff() {
        elements.image.src = 'http://example.com/image_name.png';
    }
    function removeImage() {
        document.body.removeChild(document.getElementById('image'));
        // 这个时候我们对于 #image 仍然有一个引用, Image 元素, 仍然无法被内存回收.
    }
```

上述案例中，即使我们对于 image 元素进行了移除，但是仍然有对 image 元素的引用，依然无法对齐进行内存回收。

6. 如何避免内存泄露

减少不必要的全局变量，使用严格模式避免意外创建全局变量。
在你使用完数据后，及时解除引用（闭包中的变量，dom引用，定时器清除）。
组织好你的逻辑，避免死循环等造成浏览器卡顿，崩溃的问题。

2. 实现sizeOf函数, Get size of a JavaScript object in Bytes

既然对内存这么了解，那么来一道代码题：
实现sizeOf函数, 计算传入的对象所占的Bytes数值.

三. 来聊一下前端HTTP请求相关吧

1. 平时怎么解决跨域问题的?

1. jsonp
2. cors
3. Node 正向代理 利用服务端不跨域的特性
4. nginx 反向代理 proxy_pass
5. img标签

2. 有做过全局的请求处理吗? 比如统一请求并设置登录态, 比如报错统一弹toast?

Axios的request interceptor 和 response interceptor, 单例

3. 你能给xhr添加hook, 实现在各个阶段打印日志吗?

代码题, 实现页面上通过xhr发请求的时候, 在xhr的生命周期里, 能够实现自定义的行为触发。

四. 平时用过发布订阅模式吗? 比如Vue的event bus, node的eventemitter3

1. 这种模式, 事件的触发 和 回调之间是同步的还是异步的?

2. 实现一个简单的EventEmitter?

代码题，实现eventEmitter, 包含on emit once, off四个方法

五. 来一道算法 01背包问题

经典的0-1背包问题, 动态规划。