360前端面试

360前端面试

1. 自我介绍
2. 聊项目
3. 登录功能怎么实现的
4. JWT的加密泄露怎么办?
5. JWT有哪些信息
6. 签名加密的算法
7. Vue-router 里面的hash模式和history router模式的区别
8. 项目用的是什么模式
9. 使用history模式时直接刷新带有路径的界面会发生什么?
10. 项目编译完了以后怎么部署的
11. Vite和webpack的区别
12. 聊一聊let、const、var 的区别
13. 如何做到项目保证在不同手机上的一致性的体验
14. 快速排序的实现

面试问题

自我介绍

聊项目

登录功能怎么实现的

JWT的加密泄露怎么办?

JWT有哪些信息

签名加密的算法

Vue-router 里面的hash模式和history router模式的区别

项目用的是什么模式

使用history模式时直接刷新带有路径的界面会发生什么?

项目编译完了以后怎么部署的

Vite和webpack的区别

聊一聊let、const、var 的区别

如何做到项目保证在不同手机上的一致性的体验

快速排序的实现

回答

1.自我介绍

这个比较建议大家提前写好自己的自我介绍，一般来说碰到了面试官问不太会卡壳

比如说：

我来自哪 平常对哪个内容很感兴趣 喜欢探索新技术 为什么选这个岗位 等等

一般来说，我建议大家在面试之前提前做好功课，比如说这家公司干什么的，这个岗位在这家公司要做什么，这样子提前准备会让面试官有个好的印象。

2.聊项目

项目这块我还是建议大家写点难点的项目，最好多写点复杂的组件，就好比方我后面说的登录注册这个功能，面试官会挖的非常深。从基础的前端连后端，到考虑存储位置，是否加密，加密用的方法，优缺点，全部都会问的清清楚楚。在项目这块面试官更注重你的深度，而不是广度。（我个人认为）

3.登陆注册功能的实现

这里略，相信大家的功能比我的更好。

4.JWT的加密泄露怎么办?

如果出现加密泄露的话基本上已经是很严重的问题了，因为我项目有说到jwt进行加密，因此问到了这个问题。

1. **重新生成密钥**：立即生成一个新的加密密钥，并将其用于签署新的JWT。这将使以前的JWT失效，并防止攻击者使用已泄露的密钥生成有效的JWT。
2. **吊销已发出的JWT**：如果可能的话，您可能需要在服务器端实施一种机制，使得已经发出的JWT失效。这可以通过在JWT中添加一个唯一的标识符（例如，UUID），并在服务器端跟踪这些标识符的列表来实现。
3. **增加监控和审计**：加强对JWT的监控和审计，以便及时发现任何异常活动。

5.JWT有哪些信息

是什么

说到JWT通常我们需要从概念了解清楚它究竟是什么，它的全名叫做JSON Web Token，它其实就是一串字符串，帮助我们加密在前后端传递信息。由于http的特性，服务器在接收请求的时候是无法分辨请求来源的，因此我们通常在第一次发送请求信息以后，服务器会返回给我们一个特定信息，证明这个请求是由这个用户发出的。

举个例子，甲乙同时给商城服务器发送登录请求，两个人都需要看看自己购物车有什么东西，但是服务器不知道到底该给谁的购物车信息，因此必须在两个人第一次登录以后返回一个信息，在后续两个人请求的时候辨明他们的身份。（这个就涉及到cookie这些东西，以后我专门出一期聊聊。）

那么对于登录返回的信息，服务器也不知道是不是用户自己发的，如果这个信息被黑客截取的的话就会导致用户信息泄露，那么我们就需要对它进行加密，防止被黑客拿走信息，这时候jwt就应运而生了。它可以对用户的专属信息进行加密。

包含内容

JWT（JSON Web Token）由三部分组成，它们用点号（.）分隔开来。这三部分分别是：

1. **Header（头部）**：包含了JWT的元数据信息，例如类型（typ）和所使用的算法（alg）。通常是一个JSON对象。
2. **Payload（载荷）**：包含了JWT的主要内容，例如用户的身份信息或其他数据。通常也是一个JSON对象。
3. **Signature（签名）**：由前两部分使用指定的算法进行签名生成的，用来验证JWT的真实性和完整性。签名是Base64编码后的字符串。

这三部分通过点号连接在一起就构成了一个完整的JWT。例如，一个JWT可能看起来像这样：

```
复制代码
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG
4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

其中，第一部分是头部，第二部分是载荷，第三部分是签名。

6.签名加密的算法

大家可以看到，面试官的问题非常精准，看到我们聊到了头部的加密以后，就开始聊它怎么加密的。

JWT的签名算法通常是使用HMAC（Hash-based Message Authentication Code）或RSA（Rivest-Shamir-Adleman）进行加密的。常见的算法包括：

1. **HMAC with SHA-256**：使用SHA-256哈希函数和一个密钥对JWT进行签名。

2. **HMAC with SHA-384**：使用SHA-384哈希函数和一个密钥对JWT进行签名。
3. **HMAC with SHA-512**：使用SHA-512哈希函数和一个密钥对JWT进行签名。
4. **RSA with SHA-256**：使用RSA算法和SHA-256哈希函数对JWT进行签名。
5. **RSA with SHA-384**：使用RSA算法和SHA-384哈希函数对JWT进行签名。
6. **RSA with SHA-512**：使用RSA算法和SHA-512哈希函数对JWT进行签名。

HMAC通常比RSA更快速，但RSA提供了更好的密钥管理和非对称加密的特性。

Vue-router 里面的hash模式和history router模式的区别

由于我在上面的算法里面卡壳比较严重，而且也没说出来很多算法，因此面试官转到了跳转方式的问题上面。

Vue Router 支持两种路由模式：hash 模式和 history 模式。它们的主要区别在于 URL 的表现形式和对浏览器特性的依赖。

Hash 模式：在 URL 中使用 来表示路由，如 `http://example.com/#/path/to/route`。在 hash 模式下，路由的改变不会引起浏览器向服务器发送请求，所有的路由都是在客户端进行处理的。这种模式的优点是兼容性好，支持老版本浏览器，并且部署简单。缺点是 URL 中会出现 ，看起来不够美观。

History 模式：利用了 HTML5 History API 来实现路由，去掉了 URL 中的 ，如 `http://example.com/path/to/route`。在 history 模式下，路由的改变会向服务器发送请求，因此需要服务器端的支持。这种模式的优点是 URL 较美观，不带 ，更符合传统 URL 的形式。缺点是需要服务器端支持，并且在一些特殊情况下可能会导致404错误。

因此，如果需要兼容老版本浏览器或部署简单，可以选择 hash 模式；如果需要更美观的 URL 或更接近传统 URL 的形式，可以选择 history 模式。

7.项目用的是什么模式

略过，通常大家默认都使用history模式，我也是这个

8.使用history模式时直接刷新带有路径的界面会发生什么?

在使用 Vue Router 的 history 模式时，当直接刷新带有路径的界面时，服务器会尝试去匹配相应的路由。如果服务器配置正确，能够处理这些路由，那么页面会正常加载。但是如果服务器没有正确配置，可能会导致404错误或者其他错误。

这是因为 history 模式使用了 HTML5 History API 来管理路由，它会将路由添加到浏览器的历史记录中，并且不会像 hash 模式那样在 URL 中包含 。因此，当直接刷新带有路径的界面时，浏览器会向服务器发送请求，服务器需要能够正确地处理这些请求并返回对应的页面内容。

为了避免这种问题，通常需要在服务器端配置，确保所有的路由都指向同一个 HTML 文件，以便 Vue 应用程序能够正确地处理这些路由。这通常称为单页应用程序（SPA）的服务器配置。

9.项目编译完了以后怎么部署的

由于我的项目没有部署到云服务上面，所以这方面没有答得很深入。我猜测如果答上来了会问些云服务器相关的问题。

10.Vite和webpack的区别

Vite 和 Webpack 都是现代 JavaScript 应用程序的构建工具，但它们在实现方式和目标方面有一些区别。

1. **实时开发服务器（Development Server）**：
   - Vite：Vite 内置了一个基于原生 ES 模块的开发服务器，利用浏览器原生支持的 ES 模块特性，在开发过程中能够实现快速的热更新，使得开发过程更加流畅。
   - Webpack：Webpack 也有自己的开发服务器 webpack-dev-server，但相比 Vite，它的热更新速度可能会稍慢一些，因为它是基于传统的模块热替换（HMR）技术实现的。
2. **构建速度（Build Speed）**：
   - Vite：由于 Vite 在开发模式下利用了浏览器的原生 ES 模块特性，可以实现更快的构建速度。在生产模式下，Vite 会使用 Rollup 进行打包，进一步提高了构建速度。
   - Webpack：Webpack 的构建速度可能会比 Vite 慢一些，尤其是在大型项目中，因为它需要对所有模块进行分析和打包。
3. **打包方式（Bundle）**：
   - Vite：在生产环境下，Vite 会使用 Rollup 进行打包，利用 Rollup 的 Tree Shaking 特性来实现更小的包大小。

- Webpack：Webpack 也支持 Tree Shaking，但需要通过配置来实现。同时，Webpack 还支持更多的高级特性和插件，使得它在一些复杂的场景下更具灵活性。

4. **配置方式（Configuration）**：
   - Vite：Vite 的配置相对简单，大部分情况下只需要一个简单的配置文件即可。Vite 的配置文件使用了 ES Module 的语法，更加清晰和直观。
   - Webpack：Webpack 的配置相对复杂，需要理解更多的概念和配置项。Webpack 的配置文件是一个 CommonJS 模块，需要使用 Node.js 的语法。

相对来说，Vite 更适合用于快速原型开发和小型项目，因为它具有快速的构建速度和简单的配置方式。而 Webpack 则更适合于大型项目和复杂的应用场景，因为它具有更多的高级特性和灵活的配置方式。

11.聊一聊let、const、var 的区别

12.如何做到项目保证在不同手机上的一致性的体验

因为我主要是前端方向，所以从前端聊聊

前端主要使用响应式设计 包括：

1. **弹性网格布局（Flexible Grid Layout）**：使用相对单位（如百分比、em 或 rem）而不是固定单位（如像素），使网页元素能够根据屏幕大小自动调整布局。
2. **弹性图片和媒体（Flexible Images and Media）**：使用 CSS 中的 `max-width: 100%;` 属性来确保图片和媒体元素在不同屏幕大小下能够自动缩放，避免图片溢出或失真。
3. **媒体查询（Media Queries）**：使用 CSS3 中的媒体查询功能，根据设备的屏幕大小和特性应用不同的样式，以适应不同的设备和分辨率。
4. **断点设计（Breakpoint Design）**：定义一些关键的断点（breakpoint），在这些断点上应用不同的样式，以实现在不同屏幕大小下的布局调整。
5. **流式布局（Fluid Layout）**：使用相对单位和百分比来定义页面元素的宽度，使页面能够随着屏幕大小的变化而自动调整布局。

```
1 html
2 复制代码
3 <!DOCTYPE html<html lang="en"<head<meta charset="UTF-8"<meta name="viewport"
  content="width=device-width, initial-scale=1.0"<titleResponsive Layout
  Example</title<style/* 基础样式 */body {  font-family: Arial, sans-serif;
  margin: 0;  padding: 0;}.container {  padding: 20px;  text-align: center;}/* 响
  应式布局 */@media screen and (min-width: 768px) {  .container {    width: 50%;
    margin: 0 auto;  }}@media screen and (min-width: 1024px) {  .container {
  width: 30%;  }}</style</head<body<div class="container"  <h1Responsive Layout
```

```
Example</h1  <pThis is a responsive layout example with different widths for
different screen sizes.</p</div</body</html
```

13.快速排序的实现

快速排序（Quick Sort）是一种常用的排序算法，它的基本原理是选择一个基准元素，然后将数组中小于基准元素的元素移到基准元素的左边，大于基准元素的元素移到基准元素的右边，最终将数组分成两个子数组，然后递归地对这两个子数组进行排序，直到整个数组有序。

下面是快速排序的实现代码（使用 JavaScript）：

```
1 javascript
2 复制代码
3 function quickSort(arr) {    if (arr.length <= 1) {        return arr;    }
  const pivot = arr[0]; // 选择第一个元素作为基准元素    const left = [];    const
  right = [];    for (let i = 1; i < arr.length; i++) {        if (arr[i] <
  pivot) {            left.push(arr[i]); // 小于基准元素的放在左边数组        } else
  {            right.push(arr[i]); // 大于等于基准元素的放在右边数组        }    }
   return [...quickSort(left), pivot, ...quickSort(right)]; // 递归地对左右子数组进
  行排序并合并}// 示例const array = [4, 2, 7, 1, 9, 5, 3, 8, 6];const sortedArray
  = quickSort(array);console.log(sortedArray); // 输出 [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

快速排序的时间复杂度为 O(n log n)（平均情况和最好情况），最坏情况下为 O(n^2)，空间复杂度为 O(log n)（递归调用栈的大小）。快速排序是一种原地排序算法，不需要额外的空间来存储临时数据，因此它的空间复杂度相对较低。

这也是面试官最常考的排序算法.