2025/6/24 上海道客（DaoCloud）一面

牛客面经

1.自我介绍

2.一个数组遍历一次找出最小正整数的下标，不存在返回-1（手撕）

3.async和await的使用

这里我举一个例子吧，假如我需要做一个删除购物车中某一个数据的功能，我需要现在vuex中cart模块定义一个delSelect函数，函数加上async前缀，函数里面将删除数据请求所需要的参数都拿过来，最后在函数里面用await delSelect(参数) 发请求删除数据，当然，如果有返回值也可以直接定义变量在await后面接收

async和await相对于原生的axios，避免了.then() .catch() 嵌套，看起来更像同步代码，更加清晰自然

async函数返回的是一个 Promise 对象

store/modules/cart.js

```
// 删除购物车数据
async delSelect (context) {
  const selCartList = context.getters.selCartList
  const cartIds = selCartList.map(item => item.id)
  await delSelect(cartIds)
  Toast('删除成功')

  // 重新拉取最新的购物车数据（重新渲染）
  context.dispatch('getCartAction')
}
```

views/layout/cart.vue

```
async handleDel () {
  if (this.selCount === 0) return
  await this.$store.dispatch('cart/delSelect')
  this.isEdit = false
}
```

4.Promise介绍

Promise 是 JavaScript 中用于处理异步操作的对象。它代表一个尚未完成但会在未来完成的操作结果。这是Mdn网站的定义

Promise有三种状态，待定（pending）：初始状态，既没有被兑现，也没有被拒接；已兑现（fulfilled）：操作成功完成 => 执行 .then() 回调；已拒绝（rejected）：操作失败 => 执行 .catch() 回调。Promise的回调都是微任务优先级大于宏任务

> 作用：状态改变后，调用关联的处理函数
>
> 注意：Promise对象一旦被兑现 / 拒接 就是已敲定了，状态无法再被改变了
>
> Promise对象刚创建时状态为 pending，但是对象创建时，回调函数里的代码就会执行了

Promise也有一些常用的静态方法，比如 Promise.all() 合并多个 Promise 对象，当所有 Promise 都成功时 => .then()，如果有任何一个 Promise 失败 => .catch()、Promise.race() 多个 Promise 对象谁先返回结果（无论成功或失败），就用谁的结果

Promise 链式调用可以解决回调函数地狱的问题

5.垂直居中的实现

- flex 居中

```
<div class="box">
    <div class="son"></div>
</div>

.box {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

- 定位 + transform

```
<div class="box">
    <div class="son"></div>
</div>

.box {
    position: relative;
}
.box .son {
    position: aboulate;
    left: 50%;
    right: 50%;
    transform: translate(-50%, -50%);;
}
```

> 通常使用 margin: 0 auto; 实现水平居中
>
> margin-left 和 margin-right 自动均分剩余空间

6.介绍一下flex布局

flex 布局：

- 主轴对齐方式（默认主轴是水平方向）

  justify-content

  flex-start 弹性盒子从**起点**开始依次排列

  flex-end 弹性盒子从**终点**开始依次排列

  center 弹性盒子沿主轴**居中**排列

  space-between 两个盒子先靠左右两边，空白间距均分在盒子之间

  space-around 两端出现空白间距，视觉效果：弹性盒子之间的间距是两端间距的两倍

  space-evently 两端同样出现空白间距，不过弹性盒子之间的间距与两端间距相等

- 侧轴对齐方式

  > 父级盒子开启flex布局之后，子级盒子如果不设置高度，默认侧轴方向会沿轴线铺满容器

  align-items 设置给所有盒子（给弹性容器设置）

  align-self 只设置给一个盒子（给那个盒子单独设置）

  flex-start 弹性盒子从**起点**开始依次排列

  flex-end 弹性盒子从**终点**开始依次排列

  stretch 沿着侧轴线**拉伸至铺满容器**（不能设置侧轴方向的尺寸）

  center 沿侧轴**居中**排列

- 修改主轴方向

  flex-direction: column 改变主轴方向为垂直方向，侧轴自动变换为水平方向/ ˈkɒləm /

  flex-direction: row 改变主轴方向为水平方向，侧轴自动变换为垂直方向（默认为水平方向）

- 弹性伸缩比

  > 默认情况下，主轴方向尺寸是靠内容撑开的，侧轴默认拉伸

  控制弹性盒子主轴方向的尺寸，因为默认是靠内容撑开的

  A盒子 flex: 1 B盒子 flex: 2 C盒子 flex: 1

  占空白比例 1/4 1/2 1/4

- 弹性换行

  flex-wrap: nowrap 不换行（默认）

  flex-wrap: wrap 换行

- 行对齐方式

align-content

flex-start 弹性盒子从**起点**开始依次排列

flex-end 弹性盒子从**终点**开始依次排列

center 弹性盒子沿主轴**居中**排列

space-between 两个盒子先靠左右两边，空白间距均分在盒子之间

space-around 两端出现空白间距，视觉效果：弹性盒子之间的间距是两端间距的两倍

space-evently 两端同样出现空白间距，不过弹性盒子之间的间距与两端间距相等

> align-items 和 align-content 的区别：
>
> align-items 是对单行生效的；align-content 是对多行生效的，需要开启弹性换行wrap

```
// 用flex布局把元素搞到右上角
<div class="container">
  <div class="box">右上角元素</div>
</div>

.container {
    display: flex;
    justify-content: flex-end; /* 横向靠右 */
    align-items: flex-start;
    height: 200px;
    width: 200px;
    background: pink;
}
.box {
    width: 50px;
    height: 50px;
    background: blue;
}
```

7.具体介绍一下flex:1

弹性伸缩比

8.ref和reactive

ref和reactive都是Vue3组合式API编码风格下创建数据的方式

reactive接受一个对象类型的数据，返回一个响应式对象

ref接受简单类型和复杂类型，本质其实还是在原有传入数据的基础上借助 reactive，将其包成对象实现的响应式，不过使用ref有个注意点：在脚本中访问数据，需要通过 .value，在 template 中不需要 .value

平常声明数据，常用ref

vue3中数据需要响应式才需要用ref和reactive包一层，如果只是定义一个不需要响应式的字符串就不需要ref和reactive，但是假如字符串是和 v-model 绑定的输入框联动，那也是需要ref和reactive的

9.computed属性

computed 主要是在已有的响应式数据的基础上派生出新的值，而且具有缓存特性，依赖不变不会重新计算

watch 是一个监听器，绑定的响应式数据发生变化时触发对应的回调函数

10.v-model的原理

v-model 是 :value 和 @input 语法糖，实现数据双向绑定，它可以把父组件中的数据通过 value 传递给子组件，同时也可以监听子组件内部的 input 事件来更新数据；当然如果绑定在 input 输入框当中也是如此

- vue2

```
<input type="text" v-model="message">
<input
type="text"
:value="message"
@input="message = $event.target.value"
>
```

```
// 父组件
<!-- <BaseSelect :value="selectId" @input="selectId = $event"></BaseSelect>
-->
<BaseSelect v-model="selectId"></BaseSelect>

// 子组件
export default {
  props: {
    value: String
    },
  methods: {
    handleChange(e){
    this.$emit('input', e.target.value)
  }
}
```

- vue3

> Vue3 的 `v-model` 允许自定义参数名：`v-model:title="bookTitle"` 绑定的就是 `:title` 和 `@update:title`

```
// 父组件
<!-- <CustomInput :modelValue="msg" @update:modelValue="msg = $event" /> -->
<CustomInput v-model="msg" />
```

```
// 子组件
<script setup>
  // 对于props传递过来的数据，模版中可以直接使用，脚本中需要 props.modelValue
    const props = defineProps(['modelValue'])
    const emit = defineEmits(['update:modelValue'])
</script>
```

11.git的操作

git -v

初始化仓库 git init

工作区书写代码

暂存所有文件 git add .

提交产生版本记录 git commit -m "1.第一次提交"

查看提交记录-简略信息 git log --oneline

添加远程仓库地址 git remote add 仓库别名 仓库地址

拉取 git pull 仓库别名 分支名

推送 git push 仓库别名 分支名

克隆 git clone 远程仓库地址

创建分支 git branch "分支名"

切换分支 git checkout "分支名"

删除分支 git branch -d “分支名”

把分支提交记录合并到当前所在分支 git merge “分支名”

12.拷打项目