WebMooc Restaurant

1 项目设置

```
yarn install
```

1.1 开发时编译及热重载

```
yarn serve
```

1.2 编译成最终的文件

```
yarn build
```

2 项目结构

```
├── README.md----------------------------------------------------项目文档
├── deploy.sh----------------------------------------------------部署脚本
├── public-------------------------------------------------------图片等资源
│   ├── coin.png
│   ├── consumer-1.png
│   ├── consumer-2.png
│   ├── consumer-3.png
│   ├── consumer-4.png
│   ├── consumer-5.png
│   ├── consumer-6.png
│   ├── consumer-7.png
│   ├── favicon.ico
│   ├── food-ready.png
│   ├── index.html
│   └── seat.png
├── src----------------------------------------------------------项目源代码
│   ├── App.vue--------------------------------------------------游戏主体
│   ├── assets---------------------------------------------------图片等资源
│   │   ├── background.jpg
│   │   ├── chef.png
│   │   ├── iconfinder_Instagram_UI-07_2315589.png
│   │   ├── logo.png
│   │   └── restaurant_prototype.png
│   ├── components-----------------------------------------------游戏所用组件
│   │   ├── Chefs------------------------------------------------厨师组件
│   │   │   └── RestaurantChef.vue
│   │   ├── Consumers--------------------------------------------顾客组件
│   │   │   └── RestaurantConsumers.vue
```

```
│   │   ├── ProgressBar.vue----------------------------------进度条组件
│   │   ├── RestaurantButton.vue-----------------------------按钮组件
│   │   ├── RestaurantDialog.vue-----------------------------对话框组件
│   │   └── RestaurantSeat.vue-------------------------------座位组件
│   ├── hooks------------------------------------------------勾子函数
│   │   ├── Consumers----------------------------------------顾客相关的勾子函数
│   │   │   └── index.ts
│   │   └── index.ts-----------------------------------------游戏主体相关的勾子函数
│   ├── main.ts----------------------------------------------游戏入口文件
│   ├── shims-vue.d.ts---------------------------------------声明文件
│   ├── store------------------------------------------------模块化的全局数据
│   │   ├── Chefs--------------------------------------------厨师的数据
│   │   │   └── index.ts
│   │   ├── Consumers----------------------------------------顾客的数据
│   │   │   └── index.ts
│   │   └── index.ts-----------------------------------------游戏主体数据
│   └── types------------------------------------------------定义的类型信息
│       ├── Chef
│       │   └── index.ts
│       ├── Consumer
│       │   └── index.ts
│       ├── Meal
│       │   └── index.ts
│       └── index.ts
├── tsconfig.json----------------------------------------------Typescript编译相关信息
├── vue.config.js----------------------------------------------Webpack编译相关信息
└── yarn.lock--------------------------------------------------所用库的版本信息
```

3 组件介绍

3.1 `ProgressBar` 组件

3.1.1 实现功能

在界面中显示一个大小可控，时间可控，随时可以暂停、开始，显示文字可调节的进度条

3.1.2 可配置参数

Name	Type	Function
start	Boolean	控制进度条开始
time	Number	进度条从开始到结束所需时间
size	Size	进度条的大小
finishedColor	String	进度条完成部分的颜色
unfinishedColor	String	进度条未完成部分的颜色
text	String	进度条中显示的文本
rate	Number	进度条每次增加的步长
isFinished	Boolean	标记进度条是否完成
reset	Boolean	重设进度条
stop	Boolean	暂停进度条

3.2 `RestaurantButton` 组件

3.2.1 实现功能

封装了一个风格统一的按钮，通过 `slot` 实现所需要的文字

3.3 `RestaurantDialog` 组件

3.3.1 实现功能

封装了一个风格统一的对话框，通过 `slot` 实现各部分所需要的功能

3.3.2 `slot` 介绍

3.3.2.1 `header` 部分

主要是对话框的标题

3.3.2.2 `content` 部分

主要是对话框的主体内容

3.3.2.3 `action` 部分

主要是对话框的操作部分（如按钮等）

3.3.3 可配置参数

Name	Type	Function
active	Boolean	控制对话框的显示
showClose	Boolean	控制是否显示右上角的关闭按钮

3.4 `RestaurantSeat` 组件

3.4.1 实现功能

实现了一个大小可调且自适应的座位组件

3.4.2 可配置参数

Name	Type	Function
size	String	控制座位的大小

余下组件是由基础Html部件和以上组件组合形成，逻辑较为复杂，详情见源码

4 勾子函数介绍

勾子函数的目的是将代码中的部分逻辑抽离出来，方便维护与代码编写

4.1 时间自增

```
import { ref } from 'vue'

export function useTimeAutoIncr (w = 1, d = 1) {
  const day = ref<number>(d)
  const week = ref<number>(w)
  const timer = setInterval(() => {
    if (day.value < 7) {
      day.value++
    } else {
      day.value = 1
      week.value += 1
    }
  }, 240 * 1000)

  return {
    day,
    week,
    timer
  }
}
```

4.1.1 实现功能

该函数实现了游戏左上角的时间自增的逻辑，通过timer来取消定时器以实现暂停效果，通过指定函数参数可以实现继续的效果

4.2 顾客自增

```
import { useConsumersStore } from '@/store/Consumers'
import { WaitingConsumer } from '@/types/Consumer'
```

```
let $id = 0

function addConsumer (max: number) {
  const store = useConsumersStore()

  if (store.ConsumerWaitList.length < 5) {
    $id++

    const consumer: WaitingConsumer = {
      seat: $id,
      consumer: $id,
      picId: Math.floor(Math.random() * 6 + 1)
    }

    store.ConsumerWaitList.unshift(consumer)
  }

  setTimeout(addConsumer, Math.floor(Math.random() * max * 1000), max)
}

export function useAutoAddConsumer (max: number) {
  setTimeout(addConsumer, 0, max)
}
```

4.2.1 函数参数

Name	Type	Function
max	number	下一次增加顾客的最大时间(在可以增加情况下)

4.2.2 实现功能

在等位队列长度小于5的时候，客人会在随机时间后(小于max)进入等位队列，递归调用`setTimeout`函数能一定程度上保证每一次顾客进入队列中的时间都是随机的

5 状态管理

5.1 餐厅状态管理

```
import { defineStore } from 'pinia'
import { useTimeAutoIncr } from '@/hooks'
import { Meal, MealType } from '@/types/Meal'
import { ref } from 'vue'
import { Seat } from '@/types'

export const useRestaurantStore = defineStore('Restaurant', () => {
  // 钱
```

```
const money = ref<number>(500)

// 菜单
const dishes = ref<Map<number, Meal>>(new Map<number, Meal>())
const dish = [
  {
    id: 0,
    name: '凉拌黄瓜',
    price: 8,
    type: MealType.PRE,
    time: 30
  },
  {
    id: 1,
    name: '凉拌凤爪',
    price: 16,
    type: MealType.PRE,
    time: 40
  },
  {
    id: 2,
    name: '花生米',
    price: 4,
    type: MealType.PRE,
    time: 5
  },
  {
    id: 10,
    name: '红烧肉',
    price: 68,
    type: MealType.MAIN,
    time: 60
  },
  {
    id: 11,
    name: '番茄炖牛腩',
    price: 98,
    type: MealType.MAIN,
    time: 80
  },
  {
    id: 12,
    name: '酸菜鱼',
    price: 58,
    type: MealType.MAIN,
```

```
    time: 50
  },
  {
    id: 13,
    name: '农家小炒肉',
    price: 48,
    type: MealType.MAIN,
    time: 45
  },
  {
    id: 14,
    name: '清爽时蔬',
    price: 24,
    type: MealType.MAIN,
    time: 30
  },
  {
    id: 101,
    name: '可口可乐',
    price: 3,
    type: MealType.DRINK,
    time: 10
  },
  {
    id: 102,
    name: '鲜榨果汁',
    price: 20,
    type: MealType.DRINK,
    time: 30
  }
] as Meal[]
dish.forEach(value => dishes.value.set(value.id, value))

// 时间
const time = ref({
  ...useTimeAutoIncr(),
  stop: false
})

function Stop () {
  clearInterval(time.value.timer)
  time.value.stop = true
}

function Proceed () {
```

```
  if (time.value.stop) {
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    time.value = {
      ...useTimeAutoIncr(time.value.week, time.value.day),
      stop: false
    }
  }
}

// 点菜队列
const MealWaitList = ref<{ id: number, target: number }[]>([])

// 餐桌比例
const radio = ref<number>()

// 座位
const seats = ref<Seat[]>([
  {
    id: 0,
    consumer: null,
    picId: null,
    meals: []
  },
  {
    id: 1,
    consumer: null,
    picId: null,
    meals: []
  },
  {
    id: 2,
    consumer: null,
    picId: null,
    meals: []
  },
  {
    id: 3,
    consumer: null,
    picId: null,
    meals: []
  }
])

return {
```

```
    // 钱
    money,
    // 菜
    dishes,
    // 时间
    time,
    Stop,
    Proceed,
    // 点菜队列
    MealWaitList,
    // 餐桌比例
    radio,
    // 座位
    seats
  }
})
```

5.2 厨师状态管理

```
import { defineStore } from 'pinia'
import { ref } from 'vue'
import { useRestaurantStore } from '@/store'
import { Chef } from '@/types/Chef'

export const useChefsStore = defineStore('Chefs', () => {
  const restaurantStore = useRestaurantStore()

  let $id = 0

  // 初始状态存在一个厨师
  const chefs = ref<Map<number, Chef>>(new Map<number, Chef>([
    [0, {
      id: $id,
      working: null,
      serve: null
    }]
  ]))

  function add (): boolean {
    if (chefs.value.size >= 6) {
      return false
    }

    if (restaurantStore.money < 100) {
```

```
      return false
    }

    const id = ++$id

    restaurantStore.money -= 100

    chefs.value.set(id, {
      id,
      working: null,
      serve: null
    })

    return true
  }

  function remove (id: number): boolean {
    restaurantStore.money -= 50
    return chefs.value.delete(id)
  }

  return {
    chefs,
    add,
    remove
  }
})
```

5.3 顾客状态管理

```
import { defineStore } from 'pinia'
import { ref } from 'vue'
import { WaitingConsumer } from '@/types/Consumer'

export const useConsumersStore = defineStore('Consumers', () => {
  // 客人等位队列
  const ConsumerWaitList = ref<WaitingConsumer[]>([])

  return {
    // 客人等位队列
    ConsumerWaitList
  }
})
```

6 感想和体会

MVVM实现了数据变动直接反映到UI的变化，组件化提高了代码复用率，较为统一的状态管理实现了数据流的单向流动，方便管理，这是使用框架的优势，而传统写脚本来对UI和逻辑进行操作的话，不仅效率较低，代码较难管理，也比较容易出Bug，想必这也是现如今基本上所有大公司都开始使用框架对前端进行开发了吧