Vue3快速上手指南

-- 楼兰

你只要会基础的HTML，JS，CSS，那么就可以上手Vue了。如果你会Java，那么上手Vue非常轻松。如果你会Vue2，那么上手Vue3会更加舒服。

Vue3简介

- 官网地址：https://vuejs.org/ 。中文官网 https://cn.vuejs.org/
- Vue是什么？易学易用，性能出色，适用场景丰富的 Web 前端框架。
- Vue2已经于2023年12月31日停止维护。建议升级到Vue.js3.0版本。打包更小，内存更少，渲染更快。`好消息是，vue3向下兼容vue2的语法`

- Vue3于2020年9月18日发布，代号： `One Piece` 海贼王。 `久经磨砺`
- Vue3新特性：组合式API(重点)，更好的支持TypeScript(熟悉)，状态存储框架Pinia(重点)，新组件(了解)。。。。。详见官网

一、整体认识Vue3项目

1、创建Vue3工程

前置：安装NodeJS。NodeJS版本18.0以上。

使用官方脚手架创建Vue工程[推荐]。

```
# 使用官方脚手架
npm create vue@latest

# 按照脚手架要求选择是否启用相关组件
Vue.js - The Progressive JavaScript Framework
✔ 请输入项目名称： … myVue3
✔ 请输入包名称： … myvue3
✔ 是否使用 TypeScript 语法？ … 否 / 是  # 选是
✔ 是否启用 JSX 支持？ … 否 / 是
✔ 是否引入 Vue Router 进行单页面应用开发？ … 否 / 是
✔ 是否引入 Pinia 用于状态管理？ … 否 / 是
✔ 是否引入 Vitest 用于单元测试？ … 否 / 是
✔ 是否要引入一款端到端（End to End）测试工具？ › 不需要
✔ 是否引入 ESLint 用于代码质量检测？ … 否 / 是 # 选是
✔ 是否引入 Prettier 用于代码格式化？ … 否 / 是
✔ Add Vue DevTools extension for debugging? (experimental) … 否 / 是

# 启动项目
npm install
npm run dev


#  VITE v5.1.6  ready in 315 ms
#
#  ➜  Local:   http://localhost:5173/
#  ➜  Network: use --host to expose
#  ➜  press h + enter to show help
```

> 1、所有功能组件都可以后续手动添加。
>
> 关于TypeScript，在Vue中的TypeScript可以认为是在JS的基础上，增加面向对象的能力。可以定义接口、类、抽象类等。
>
> 2、npm install过程中会去node仓库下载很多依赖库，放到项目本地node_modules目录。建议将npm源设定为淘宝提供的国内镜像，可以下载快一点。
>
> ```
> npm config get registry https://registry.npmmirror.com
> ```

补充：vue2时提供了另外一个脚手架vue-cli，也可以用来创建vue3项目。但是vue-cli已经处于停止维护状态。

```
# 安装或者升级脚手架
npm install -g @vue/cli

# 查看脚手架版本，确保版本在4.5.0以上
vue --version

# 创建应用
vue create vue_test

# 创建时选择3.x
#  Choose a version of Vue.js that you want to start the project with (Use arrow keys)
#  > 3.x
#    2.x

# 启动
cd vue_test
npm run serve
```

另外，官方还有其他一些集成vue的方法，自行参考。

2、主要工程结构

官方建议开发IDE： vscode。提供了辅助开发插件 vue-official。`在这之前有个插件叫volar，现在已经停用`

主要代码结构如下图

- 典型的Vue项目，都是在index.html这一个单页面里形成各种交互，这也就是所谓的SPA(Single Page Application)
- Vue3的核心是通过createApp函数创建一个应用实例，在这个实例中构建各种应用。(main.ts中)
- 每个vue文件就是一个页面上的组件，组件可以嵌套使用。
- vue中的组件分为<template>页面模板，<script>脚本和<style>样式三个部分。Vue2中要求<template>下必须有一个唯一的根元素，Vue3中则没有了这个限制。

二、数据双向绑定

双向绑定是Vue最为核心的功能。简单理解就是<template>中的页面数据和<script>中的脚本数据进行绑定，其中任何一个数据发生了变化，另一个数据也随之发生变化。

1、vue2语法的双向绑定

```
<template>
  <div>
    姓名: <input v-model="userName" /> {{ userName }} <br />
```

```
    薪水：<input type="number" v-model="salary" /> <button @click="addSalary">薪水加1000</button> {{ salary }}
  </div>
</template>
<script lang="ts">
  export default{
    //数据
    data() {
        return {
          userName:"王一",
          salary:15000
        }
    },
    //方法
    methods:{
      addSalary(){
        this.salary += 1000
      }
    }
  }
</script>

<style scoped>
</style>
```

数据双向绑定可以说是整个Vue的核心。例如，我们可以用数据双向绑定实现一些更为复杂的表单。

```
<template>
  <div>
    姓名：<input v-model="userName"/> {{ userName }}<br />
    薪水：<input type="number" v-model="salary"/> {{ salary }}<br />
    <button v-on:click="addSalary">提交</button> <button @click="changeUserInfo">查看个人信息</button>
  </div>
  <hr />
  <div class="userInfo" v-if="showUserInfo">
    <h2>个人信息</h2>
    年龄：<input type="number" v-model="userInfo.age" /><br />
    性别：<input type="radio" value="1" v-model="userInfo.sex">男<input type="radio" value="2" v-model="userInfo.sex">女<br/>
    岗位：<select v-model="userInfo.department">
              <option value="dev">开发</option>
              <option value="test">测试</option>
              <option value="maintain">运维</option>
            </select><br />
    技术：<span v-for="skill in userInfo.skills" :key="skill">{{ skill }}</span><br />
    学习新技术：  <input v-model="newSkill" /> <button @click="learnSkill">学习</button><br />
    个人信息汇总：{{ userInfo }}
  </div>
</template>
```

```
<script lang="ts">
  export default{
    data(){
      return{
        userName:'roy',
        salary:15000,
        userInfo:{
          age:0,
          sex:1,
          skills:['java','vue','python'],
          department:''
        },
        newSkill:'',
        showUserInfo:false
      }
    },
    methods:{
      addSalary(){
        this.salary +=1000
      },
      learnSkill(){
        if(this.newSkill)
        this.userInfo.skills.push(this.newSkill)
      },
      changeUserInfo(){
        this.showUserInfo= !this.showUserInfo
      }
    }
  }
</script>

<style scoped>
.userInfo{
  background-color: bisque;
  width: 80%;
}
.userInfo span{
  background-color: yellow;
  margin-left: 10px;
  border: 1px;
  border-radius: 5px;
}
</style>
```

这样的表单，如果要用纯JS实现，就会相当困难。但是，用双向绑定就简单很多。

2、OptionsAPI和CompositionAPI

Vue2中常用的这种编写方式称为OptionsAPI，配置式。其实现方式是用一个统一的配置对象来实现全部代码逻辑。在这个对象中，通过data、methods、computed等配置选项来控制逻辑。

OptionsAPI是Vue2时的标准API编写方式。Vue3向下兼容了Vue2的API。因此，Vue2的老项目，在Vue3中基本可以无缝迁移。`实际上，OptionsAPI是在CompositionAPI的基础上实现的。关于Vue的基础概念和知识，在这两种API之间是通用的`。另外，官方建议，如果采用Vue构建完整的SPA应用，那么更建议使用CompositionAPI。

但是，OptionsAPI所有逻辑都混在一起，不便于维护和复用。 Vue3另外通过了一种更方便的API，Composition API，混合式API。

上面同样的示例，用Composition API的写法如下：

```
<template>
  <div>
    姓名：<input v-model="userName" /> {{ userName }} <br />
    薪水：<input type="number" v-model="salary" /> <button @click="addSalary">薪水加
1000</button> {{ salary }}
  </div>
</template>
<script lang="ts">
  export default{
   setup(){
    //现在声明的变量还不具备双向绑定
    let userName="王一"
    let salary=15000

    function addSalary(){
      salary += 1000
      console.log("salary = " + salary)
    }
    //模板要用哪些，就返回哪些
    return {userName,salary,addSalary}
   }
  }
</script>

<style scoped>
</style>
```

> 1、setup是Vue3中的一个生命周期函数，他会在组件加载时执行。后面会细讲生命周期。
>
> 2、setup可以返回对象或者函数。如果是一个对象，则对象中的属性、方法等，可以在模板中直接使用(常用)。如果返回一个函数，则通过函数的返回值直接渲染页面，不经过模板。例如 `setup(){return ()=>"直接渲染"}`
>
> 3、setup是一个普通的函数，不能使用this。`OptionsAPI中可以通过this访问脚本本身的数据` 同时 `setup中不处理this，意味着setup编写可以更灵活，不需要依赖当前页面上下文`
>
> 4、此时声明的userName， salary等变量不具备双向绑定。Vue3对双向绑定做了重新设计，后面会详细分享。
>
> 5、setup有一种简写的方式<script setup lang="ts">。这样就不需要写函数了，标签内部直接写函数体。在标签内部声明的对象，函数等，都会直接return出去。`项目中常用`

```
<script setup lang="ts">
//现在声明的变量还不具备双向绑定
let userName="王一"
let salary=15000

function addSalary(){
 salary += 1000
 console.log("salary = " + salary)
}
</script>
```

在CompositionAPI中，由于setup是一个不同的函数，不需要处理this。这也意味着setup函数编写可以更加灵活，不需要依赖当前页面上下文。例如：将示例中的脚本单独写到一个ts文件中。

```
// MySalary.ts
import { onMounted, ref } from "vue"

export default function(){

    //现在声明的变量还不具备双向绑定。添加ref函数才能具备响应式
    const userName=ref("王一")
    const salary=ref(15000)

    function addSalary(){
      salary.value += 1000
      console.log("salary = " + salary.value)
    }

    onMounted(()=>{
        console.log("加载了外部脚本")
    });

    return {userName,salary,addSalary}
}
```

然后，在App.vue中就可以直接引用脚本

```
<template>
  <div>
    姓名：<input v-model="userName" /> {{ userName }} <br />
    薪水：<input type="number" v-model="salary" /> <button @click="addSalary">薪水加
1000</button> {{ salary }}
  </div>
</template>
<script setup lang="ts">
import MySalary from './components/MySalary';
let {userName,salary,addSalary} = MySalary()
</script>

<style scoped>
</style>
```

如果App.vue的逻辑越来越复杂，通过这种方式，就更易于将相关的属性和方法整理到一起，从而实现一个特定的业务功能。

> 1、ref函数让变量具备了双向绑定功能。后面详细分析。
>
> 2、复杂页面可以用这种方式。一般情况下，显然是将MySalary的模板和脚本封装到一起，这就是自定义组件了。

3、Vue3中的数据双向绑定

3.1 ref定义基础类型响应式数据

- 语法： let userName=ref(初始值)。
- 返回值：一个RefImpl的实例对象，值被包裹在对象的value属性中。
- 注意点：
  - 脚本中要用ref对象的value属性访问值，例如userName.value。但是模板中可以直接用。
  - ref对象本身不是响应式的，value属性是响应式的。例如js中修改值，要通过userName.value="xxx"，而不能userName="xxx"。
  - vue-official插件中可以选择自动添加value属性。(需要手动勾选)

```
<template>
  <div>
    姓名：<input v-model="userName" /> <button @click="changeName">名字后面加1</button> {{
userName }} <br />
    薪水：<input type="number" v-model="salary" /> <button @click="addSalary">薪水加
1000</button> {{ salary }}
  </div>
</template>
<script setup lang="ts">
import { ref } from 'vue';
//基础类型用ref声明响应式
let userName=ref("王一")
let salary=ref(15000)
```

```
function changeName(){
  userName.value += "一"
  //userName不是响应式的，userName.value才是响应式的。重新定义userName就无法双向绑定
  //userName = ref("王一一")
}

function addSalary(){
  //脚本中操作数据要加.value
  salary.value += 1000
  //观察salary对象结构
  console.log("salary = " , salary)
}
</script>
```

3.2 reactive定义对象型响应式数据

- 语法： let salaryInfo = reactive({userName:"王一",salary:15000})
- 返回值：一个Proxy实例对象，具有双向绑定能力。

```
<template>
  <div>
    姓名：<input v-model="salaryInfo.userName" /> <button @click="changeName">名字后面加1</button> {{ salaryInfo.userName }} <br />
    薪水：<input type="number" v-model="salaryInfo.salary" /> <button @click="addSalary">薪水加1000</button> {{ salaryInfo.salary }}
  </div>
</template>
<script setup lang="ts">
import { reactive } from 'vue';
// 对象类型用reactive声明响应式
let salaryInfo = reactive({userName:"王一",salary:15000})

function changeName(){
  salaryInfo.userName+="一"
}

function addSalary(){
  salaryInfo.salary+=1000
  //观察SalaryInfo对象
  console.log("salaryInfo",salaryInfo)
}
</script>

<style scoped>
</style>
```

3.3 ref对比reactive

这两者都是用来声明响应式数据的。但是也有一些需要注意的地方。

- ref也可以用来声明对象型响应式数据。其内部也是使用reactive实现。例如下面的写法效果是一样的

```
<script setup lang="ts">
import { ref } from 'vue';
// 对象类型用reactive声明响应式
let salaryInfo = ref({userName:"王一",salary:15000})

function changeName(){
  salaryInfo.value.userName+="一"
}

function addSalary(){
  salaryInfo.value.salary+=1000
  //观察SalaryInfo对象
  console.log("salaryInfo",salaryInfo)
}
</script>
```

  其中salaryInfo.value其实拿到的就是一个Reactive对象。

- 基础类型响应式数据，只能用ref声明。对象型响应式数据，ref，reactive都可以。通常大对象推荐使用reactive。
- 对象型响应数据，如果将各个属性拆解出来，是不具备响应式的。如果需要响应式属性，可以使用toRefs或者toRef函数进行转换。例如

```
<template>
  <div>
    姓名：<input v-model="name" /> <button @click="changeName">名字后面加1</button> {{ name }} <br />
    薪水：<input type="number" v-model="money" /> <button @click="addSalary">薪水加1000</button> {{ money }}
  </div>
</template>
<script setup lang="ts">
import { reactive, toRef, toRefs } from 'vue';
// 对象类型用reactive声明响应式
let salaryInfo = reactive({userName:"王一",salary:15000})
// 拆解出来的属性，是基础数据，不具备响应式
// let name = salaryInfo.userName
// let money = salaryInfo.salary
// toRef将对象的属性转为一个响应式数据
let name = toRef(salaryInfo,'userName')
let money = toRef(salaryInfo,'salary')
// 将对象的所有属性一起转换成响应式数据
// let {userName,salary} = toRefs(salaryInfo)
function changeName(){
  name.value +="一"
  console.log("name",name)
}
```

```
function addSalary(){
  money.value +=1000
  //观察SalaryInfo对象
  console.log("money",money)
}
</script>

<style scoped>
</style>
```

3.4 标签的ref属性

在<template>中定义模板时，可以通过ref属性将当前DOM元素绑定给响应式变量。

```
<template>
    姓名：<input ref="name" abc="aaaaa"/> <button @click="showRes">分析输入框</button>
</template>
<script setup lang="ts">
import { ref } from 'vue';

let name = ref()

function showRes(){
  console.log(name) //RefImpl ref对象
  console.log(name.value) //<input> dom元素
  console.log(name.value.value) //输入框的值
  console.log(name.value.getAttribute("abc")) //自定义属性的值
}
</script>

<style scoped>
</style>
```

如果只是针对普通元素，还体现不出Ref的作用。如果配合自定义组件，则更能体现Ref属性的作用。例如，针对薪水信息，可以自己写一个简单组件，把多个输入框整合到一起。

```
<!-- 自定义的薪水信息输入组件 -->
<template>
    姓名：<input v-model="userName"><br />
    薪水：<input type="number" v-model="salary">
</template>

<script lang="ts">
    //组件名默认是文件名。如果不希望用文件名，也可以自定义
    export default {
        name:"SalaryInfo"
    }
</script>
```

```
<script setup lang="ts">
import { ref } from 'vue';
    //响应式数据默认值
    let userName = ref("unknown")
    let salary = ref(1000)
    //对外暴露属性。只有暴露出去，组件外部才能访问
    defineExpose({userName,salary})
</script>

<style></style>
```

然后，在App.vue中，就可以通过ref属性获取薪水输入框的值。

```
<!-- App.vue -->
<template>
    <MySalaryInfo ref="salaryInfo"/><button @click="showRes">查看薪水信息</button>
</template>
<script setup lang="ts">
  //引入子组件
import MySalaryInfo from '@/components/MySalaryInfo.vue';
import { ref } from 'vue';
  //获取绑定对象
let salaryInfo = ref()

function showRes(){
  console.log(salaryInfo) //RefImpl ref对象
  console.log(salaryInfo.value) //Proxy 子组件的响应式数据
  console.log(salaryInfo.value.userName) //输入框的值
  console.log(salaryInfo.value.salary)
}
</script>

<style scoped>
</style>
```

3.5自定义组件的props属性

上面的示例相当于是子组件将属性暴露给父组件。那如果想要父组件给子组件赋值呢？这就可以用到组件的props属性。

```
<!-- App.vue -->
<template>
  <MySalaryInfo :salaryInfo="salaryInfo"/> <br /><button @click="setSalary">修改薪水信息
</button>
</template>
<script setup lang="ts">
//引入子组件
import MySalaryInfo from '@/components/MySalaryInfo.vue';
import { reactive } from 'vue';
```

```
let salaryInfo = reactive({
    userName:"王一",salary:15000
  })
function setSalary(){
  salaryInfo.salary+=1000
  console.log(salaryInfo)
}
</script>

<style scoped>
</style>
```

此时，在MySalaryInfo组件内，就可以通过definePropss函数，获取属性。

```
<!-- MySalaryInfo.vue -->
<template>
    {{ salaryInfo }} <br />
    <!-- 父组件传进来的值，不建议直接用，eslint会报红提示 -->
    姓名：<input v-model="salaryInfo.userName"><br />
    薪水：<input type="number" v-model="salaryInfo.salary">
</template>

<script lang="ts">
    //组件名默认是文件名。如果不希望用文件名，也可以自定义
    export default {
        name:"SalaryInfo"
    }
</script>

<script setup lang="ts">
import type { SalaryInfo } from '@/types/SalaryInfo';

//直接接收，不限定类型
// defineProps(["salaryInfo"])
//接收参数，限定类型
defineProps<{salaryInfo:SalaryInfo}>()
</script>

<style></style>
```

三、VUE3生命周期

每个 Vue 组件实例在创建时都需要经历一系列的初始化步骤，比如设置好数据侦听，编译模板，挂载实例到 DOM，以及在数据改变时更新 DOM。在此过程中，它也会运行被称为生命周期钩子的函数，让开发者有机会在特定阶段运行自己的代码。

生命周期有四个阶段：创建，挂载，更新，销毁。每个阶段有一前一后两个函数

OptionsAPI的生命周期函数：

- 创建阶段：`beforeCreate`、`created`
- 挂载阶段：`beforeMount`、`mounted`
- 更新阶段：`beforeUpdate`、`updated`
- 销毁阶段：`beforeDestroy`、`destroyed`

CompositionAPI的生命周期函数：

- 创建阶段：`setup`
- 挂载阶段：`onBeforeMount`、`onMounted`
- 更新阶段：`onBeforeUpdate`、`onUpdated`
- 卸载阶段：`onBeforeUnmount`、`onUnmounted`

示例

```
<template>
  <div>
    薪水：<input type="number" v-model="salary" /> <br />
    <button @click="addsum">薪水+1000</button>
  </div>
</template>

<!-- vue3写法 -->
<script lang="ts" setup>
  import {
    ref,
    onBeforeMount,
    onMounted,
    onBeforeUpdate,
    onUpdated,
    onBeforeUnmount,
    onUnmounted
  } from 'vue'

  // 数据
  let salary = ref(0)
  // 方法
  function addsum() {
    salary.value += 1000
  }
  console.log('setup')
  // 生命周期钩子
  onBeforeMount(()=>{
    console.log('挂载之前')
  })
  onMounted(()=>{
    console.log('挂载完毕')
  })
  onBeforeUpdate(()=>{
    console.log('更新之前')
  })
```

```
  onUpdated(()=>{
    console.log('更新完毕')
  })
  onBeforeUnmount(()=>{
    console.log('卸载之前')
  })
  onUnmounted(()=>{
    console.log('卸载完毕')
  })
</script>
```

四、Vue-Router组件路由机制

Vue项目虽然只有index.html一个页面，但是可以通过多路由机制实现多页面跳转的效果。访问不同链接，展示不同的页面内容，形成多页面的效果。

Vue官方提供了Vue-Router组件实现路由管理，官网地址：https://router.vuejs.org/zh/ 。该组件可以在创建Vue项目时选择引入。如果创建时没有安装，也可以手动安装。

```
npm install vue-router@4
```

> vue3要求使用router组件最新版本。目前最新版本是4

1、基础使用

首先要在ts脚本中配置router组件。

main.ts

```
import { createApp } from 'vue'
import App from './App.vue'

import { createRouter,createWebHistory } from "vue-router";

import HomePage from "@/pages/Home.vue"
import AboutPage from "@/pages/About.vue"
import NewsPage from "@/pages/News.vue"
//配置路由规则
const routes = [
    { path: '/',redirect: '/home'}, //默认跳转都首页
    { path: '/home', component: HomePage },
    { path: '/about', component: AboutPage, name:'about' }, //命名路由
    { path: '/news', component: NewsPage },
  ]
//创建路由器
const router = createRouter({
    history: createWebHistory(),//路由器工作模式
    routes,
})
//项目中，通常将两个配置项放到单独的ts文件中
```

```
const app = createApp(App)
//加载路由器
app.use(router)

app.mount('#app')
```

然后，在Vue模板中，配置跳转链接(<router-link>标签)以及跳转页面(<router-view>标签)。

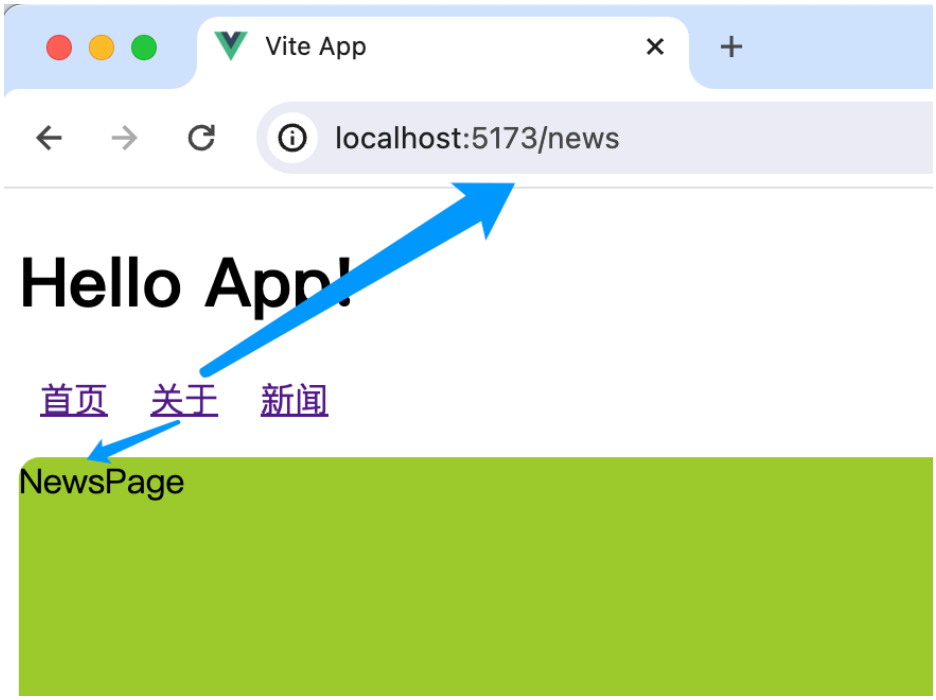
App.vue

```
<template>
  <div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- 使用 router-link 组件进行导航 -->
    <!-- 通过传递 `to` 来指定链接 -->
    <!-- `<router-link>` 将呈现一个带有正确 `href` 属性的 `<a>` 标签 -->
    <router-link to="/home">首页</router-link> <!-- 直接跳转 -->
    <router-link :to="{ path:'/about'}">关于</router-link> <!-- 路径跳转 -->
    <router-link replace :to="{ name:'news'}">新闻</router-link> <!-- 命名跳转 -->
  </p>
  <div class="content">
    <!-- 路由出口 -->
    <!-- 路由匹配到的组件将渲染在这里 -->
    <router-view></router-view>
  </div>
</div>
</template>

<!-- vue3写法 -->
<script lang="ts" setup >
</script>

<style>
  a{
    margin: 10px;
  }
  .content{
    background: yellowgreen;
    widows: 10%;
    height: 400px;
    border: 1cap;
    border-radius: 10px;
  }
</style>
```

启动后，点击页面上方的菜单，下方内容页就会显示相对应的内容。同时注意观察上方路径变化。



2、路由工作模式

在router配置中的history项为路由工作模式。Vue提供了两种工作模式：

- history模式

> 访问路径：URL不带#，斜杠链接，接近传统网站。缺点：容易产生404错误。

```
const router = createRouter({
  history:createWebHistory(), //history模式
  /******/
})
```

- hash模式

> 访问路径：URL带有#。缺点：对SEO不太友好。比较适合内部系统。

```
const router = createRouter({
  history:createWebHashHistory(), //hash模式
  /******/
})
```

3、replace属性

<route-link>标签可以添加replace属性。有两种可选配置： push和replace

- push 追加浏览器历史记录(默认值)。追加历史记录后，可以使用浏览器的返回按钮，跳回历史页
- replace 替换浏览器历史记录。替换历史记录后，浏览器的返回按钮不可用。

4、嵌套路由

<route-view>标签嵌入的页面中支持进一步嵌套子菜单。例如，新闻页希望进一步嵌套新闻路由。新闻页有多条新闻，希望在新闻页展示多条新闻的标题。点击标题，可以查看对应新闻的详情。

首先，定义三个新闻对应的详情页。每个详情页包含简单的内容

```
<!-- NewsDetail1.vue -->
<template>
    <p>新闻ID:   1</p>
    <p>新闻标题:  1 </p>
    <p>新闻内容:   1 </p>
</template>
<script lang="ts" setup>

</script>
<style>

</style>
```

并配置到路由规则中

```
import News1 from "@/pages/NewsDetail1.vue"
import News2 from "@/pages/NewsDetail2.vue"

const routes = [
    { path: '/',redirect: '/home'}, //默认跳转都首页
    { path: '/home', component: HomePage },
    { path: '/about', component: AboutPage, name:'about' }, //命名路由
    {
      path: '/news',
      component: NewsPage,
      name:'news',
      children:[ //子路由
        {
          name: "xinwen1",
          path: "1",
          component: News1
        },
        {
          name: "xinwen2",
          path: "2",
          component: News2
        }
      ] },
  ]
```

然后，在新闻详情页增加嵌套路由

```
<template>
    <div class="news">
    <!-- 导航区 -->
```

```
    <ul>
        <li><RouterLink to="/news/1">新闻1</RouterLink></li>
        <li><RouterLink to="/news/2">新闻2</RouterLink></li>
    </ul>
    <!-- 展示区 -->
    <div class="news-content">
      <RouterView></RouterView>
    </div>
  </div>
</template>

<script setup lang="ts">

</script>
<style scoped>
/* 新闻 */
.news {
  padding: 0 20px;
  display: flex;
  justify-content: space-between;
  height: 100%;
}
.news ul {
  margin-top: 30px;
  list-style: none;
  padding-left: 10px;
}
.news li>a {
  font-size: 18px;
  line-height: 40px;
  text-decoration: none;
  color: #64967E;
  text-shadow: 0 0 1px rgb(0, 84, 0);
}
.news-content {
  width: 90%;
  height: 90%;
  border: 1px solid;
  margin-top: 20px;
  border-radius: 10px;
}
</style>
```

这样就实现了新闻页内的嵌套路由。点击新闻标题，会跳到对应的新闻详情页。

5、路由传参

上面的示例显然太呆板，现实的场景当然是希望查出一个完整的新闻列表，然后每个新闻页都是展示新闻列表中的内容，而不是每个组件内固定的内容。这也就需要进行路由传参，也就是NewsDetail中的内容是从新闻列表中传递进来的。

Vue3中提供了两种传参方式，query传参和param传参。

1、query传参

News.vue传参

```
<!-- 字符串传参 -->
<router-link to="/news/1?id=1&title=新闻1&content=asdfasdf"

<!-- 对象传参 -->
<RouterLink
  :to="{
    path:'/news/1',
    query:{
      id:'1',
      title:'新闻1',
      content:'asdfasdf'
    }
  }">
  新闻1
</RouterLink>
```

NewsDetail.vue接收参数

```
import {useRoute} from 'vue-router'
import {toRefs} from 'vue'
const route = useRoute()
// 打印query参数
console.log(route.query)
//配置双向绑定数据
let {query} = toRefs(route)
```

2、params传参

params传参方式表示所有参数都拼接到URL上。

首先需要在route配置中预设占位符

```
{
      path: '/news',
      component: NewsPage,
      name:'news',
      children:[ //子路由
        {
          name: "xinwen2",
          // Param传参，URL预设占位符，?表示参数可选
          path: "2/:id/:title/:content",
          component: News2
        }
      ] },
```

然后，传参时，在RouteLink中直接传到预设的URL，或者用name属性指定目标。

```
<!-- params传参 -->
<RouterLink to="/news/2/1/新闻2/qowuieoiu">param路径传参</RouterLink>
<!-- params传参 -->
<RouterLink
  :to="{
    name:'xinwen2',
    params:{
      id:2,
      title :'新闻2',
      content :'qowiueoiqu'
    }
    }">
  param对象传参
</RouterLink>
```

接下来NewsDetail2.vue中通过路由的params属性接收参数

```
import {useRoute} from 'vue-router'
import {toRefs} from 'vue'
const route = useRoute()
// 打印params参数
console.log(route.params)
//配置双向绑定数据
let {params} = toRefs(route)
```

五、Pinia集中式状态存储

1、理解状态

在任意Vue页面之间共享的存储数据。简单理解：在当前Vue项目中使用的MySQL数据库。例如登录信息，只要完成了登录，所有Vue页面都能读取到当前登录用户。

Vue2中提供的集中状态存储框架是Vuex，Vue3中新提供了Pinia。如果你使用的还是Vue2，那么主要下，Vuex和Pinia不能一起使用。

2、创建store

Pinia可以在创建应用时选择引入。如果创建时没有引入，那就需要手动引入一下。

```
npm install pinia
```

Pinia的使用方式和Route组件基本相似。需要在启动的ts文件中使用use函数引入。

main.ts

```
import {createPinia} from 'pinia'
//加载pinia
const pinia = createPinia()
app.use(pinia)
```

接下来使用pinia需要创建Store。一个Store可以理解为MySQL中的一个库，保存一部分数据。Pinia的Store中有三个概念： state, getter , action。这三个概念也可以类比于熟悉的MVC。state相当于是数据；getter相当于是服务，用来获取并返回数据；action相当于是Controller，组织业务逻辑。

创建定义store的文件 store/user.ts

```
import { defineStore } from 'pinia'

export const userStore = defineStore('userStore',{
    //action封装修改state的业务动作
    actions:{
        changeUsername(value:string){
            if(value && value.length<10){
                this.username = value
            }
        }
    },
    //getters读取state的计算值
    getters:{
        getUsername():string{
            return this.username.toUpperCase()
        }
    },
    //state定义要保存的数据结构
    state(){
        return{
            //给定默认值
            username:'--'
        }
    }
})
```

> store中最为核心的就是state，而在定义state时，可以利用TypeScript的类型定制功能，对复杂数据结构进行规范。例如
>
> ```
> const useStore = defineStore('storeId', {
> state: () => {
> return {
> // 用于初始化空列表
> userList: [] as UserInfo[],
> // 用于尚未加载的数据
> user: null as UserInfo | null,
> }
> },
> })
> ```

```
interface UserInfo {
name: string
age: number
}
```

如果你熟悉Java后端开发，有没有觉得接口很熟悉？那么接下来，类、抽象类这些呢？也可以尝试尝试。

3、使用store操作数据

App.vue中修改stroe的数据

```
<script lang="ts" setup >
//获取store
import { userStore } from "@/store/user";
const user = userStore()
//修改store中的值
//1、直接修改某一个state
user.username='roy'
//2、修改完整的state
user.$patch({
  username:'roy2'
})
//3、通过action进行修改 推荐方式
user.changeUsername('roy')
// 获取store中的数据
console.log(user.username)
// 通过getter获取state数据 推荐方式
console.log(user.getUsername)
</script>
```

pinia的使用几乎没有门槛，相比vuex要简单很多，所以官方对Pinia的定义是符合直觉的状态管理库。因此，在使用pinia时，更应该是注意使用规范。

4、storeToRefs声明响应式数据

如果需要将store中的数据声明成响应式数据，供Vue的模板使用，可以使用pinia提供的storeToRefs函数。他和Vue提供的toRefs函数的区别在于，stroeToRefs只将store中的数据转换成响应式数据。而toRefs会将store对象中很多隐藏的方法和属性页转换出来。

```
<template>
  <div id="app">
  <h1>Hello {{ userInfo.username.value }}</h1>
</div>
</template>

<!-- vue3写法 -->
<script lang="ts" setup >
//获取store
import { userStore } from "@/store/user";
```

```
import { storeToRefs } from "pinia";
import { toRefs } from "vue";
const user = userStore()
//storeToRefs转换后只有username和getUsername
let userInfo = storeToRefs(user)
console.log(userInfo)
//toRefs转换后包含了很多隐藏方法和属性，比如$patch
let userInfo2 = toRefs(user)
console.log(userInfo2)
</script>

<style>
</style>
```

5、store的混合式写法

store也有一种混合式的写法，将各种组件混合到一起。

```
import { defineStore } from 'pinia'
import { reactive } from 'vue'

export const userStore = defineStore('userStore',()=>{
    //相当于是state
    const userInfo = reactive({username:"---"})
    //相当于action
    function changeUsername(value:string){
        if(value && value.length<10){
            userInfo.username = value
        }
    }
    //相当于getters
    function getUsername():string{
            return userInfo.username.toUpperCase()
    }
    //不用区分什么类型，返回出去的就可以用
    return {userInfo,changeUsername,getUsername}
})
```

在App.vue中，也可以像使用普通对象一样，使用store中的方法和对象。

```
<template>
  <div id="app">
  <!-- 注意对象拆包过程 -->
  <h1>Hello {{ res.userInfo.value.username }}</h1>
</div>
</template>
<!-- vue3写法 -->
<script lang="ts" setup >
//获取store
import { userStore } from "@/store/user2";
```

```
import { storeToRefs } from "pinia";
const user = userStore()
//修改store中的值
//通过action进行修改 推荐方式
user.changeUsername('roy')
// 获取store中的数据
console.log(user.userInfo)
// 通过getter获取state数据 推荐方式
console.log(user.getUsername())
//混合式store转成Ref后，只有数据的ref
let res = storeToRefs(user)
console.log(res)
</script>

<style>
</style>
```

这种方式相当于在做MVC开发时，将Controller\Service\Dao这些组件写到一起。

> 复杂项目当中，不太建议这样用。但是如果别人这么用了，你要能看懂。

六、快速上手Element-Plus

ElementUI是饿了么开源的一套基于Vue2的经典UI库。针对Vue3，升级成为了ElementPlus。熟悉ElementPlus库，不但可以节省大量前端项目的开发时间，同时也是深入了解Vue3复杂组件开发的很好途径。

ElementPlus官网地址：https://element-plus.gitee.io/zh-CN/ 。 目前还在迭代更新过程当中。

1、安装ElementPlus

```
npm install element-plus --save
```

2、引入ElementPlus

main.ts

```
import { createApp } from 'vue'
import App from './App.vue'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'
const app = createApp(App)
app.use(ElementPlus)
app.mount('#app')
```

3、使用ElementPlus组件 参见官方文档。

```
<template>
  <div class="mb-4">
    <el-button>Default</el-button>
    <el-button type="primary">Primary</el-button>
    <el-button type="success">Success</el-button>
```

```
    <el-button type="info">Info</el-button>
    <el-button type="warning">Warning</el-button>
    <el-button type="danger">Danger</el-button>
  </div>
</template>

<!-- vue3写法 -->
<script lang="ts" setup >
import { ElButton } from 'element-plus';
</script>

<style>
</style>
```

或者，你也可以直接使用element-plus提供的Demo：https://github.com/element-plus/element-plus-vite-starter 。 里面有更多现成的案例。

> ElementUI针对Vue2还推出过一个vue-admin模版，里面案例更丰富，集成度也更高。很多企业内部项目都可以直接拿来用。有兴趣可以了解一下。而针对Vue3，只推出了一个将ElementUI从Vue2升级到Vue3的迁移工具，尚未提供Vue3的版本。

> 类似的UI框架还有很多，给大家例举几个常用的
>
> - Ant Design Vue(https://www.antdv.com/docs/vue/getting-started-cn) 经典老框架
> - Native UI(https://www.naiveui.com/zh-CN/light) 仅支持Vue3的一个新的UI库
> - Tdesign(https://tdesign.tencent.com/) 腾讯开源的前端UI框架 包含桌面与移动端
> - NutUI(https://nutui.jd.com/#/) 京东开源的前端UI框架
> - uvuewui(https://www.uviewui.com/) 适合移动端uni-app开发