Vue3——Router4教程(小满版本)

> Vue 路由允许我们通过不同的 URL 访问不同的内容。通过 Vue 可以实现多视图的单页 Web 应用

路由导读

1-2.1、什么是路由?

1. 一个路由（route）就是一组映射关系（key - value），多个路由需要路由器（router）进行管理。
2. key 为路径, value 可能是 function 或 component

1-2.2、路由分类

1、后端路由：

1）理解：value 是 function, 用于处理客户端提交的请求。

2）工作过程：服务器接收到一个请求时, 根据**请求路径**找到匹配的**函数** 来处理请求, 返回响应数据。

2、前端路由：

1）理解：value 是 component，用于展示页面内容。

2）工作过程：当浏览器的路径改变时, 对应的组件就会显示。

第一章节(上)——入门（安装及配置）

1. 安装 vue-router

```
npm i vue-router --save
//注意Vue2与Vue3的路由是互不兼容的，使用Vue3请使用Router4
```

2. 配置 vue-router

新建 `router` 文件夹

路径： `/src/router`

新建 `index.js` 文件

路径： `/src/router/index.js`

1. 步骤

- 在`router`文件夹下的`index.js`中我们进行引入路由，然后创建一个路由器，在路由器中配置路由
- 然后要在`main.js`中进行引入router并进行使用
- 在App.vue中使用进行存放路由，这个是一个容器(内置组件，可以写在任何位置)
- 有了容器我们就可以往里面放东西了，还记得我们在index.js配置的路由信息吗?

- 使用，`to`决定了我们要跳向哪个位置，to里面填写的内容就是我们填在路由器中的路由的path。通过点击触发path路径然后激活component进行路由跳转。请注意这这个`router-link`是一个a标签

2. RouteRecordRaw属性通过源码可以看出联合了`RouteRecordSingleView`，`RouteRecordMultipleViews`，`RouteRecordRedirect`这三个类型，在继续往下，我们可以看到很多个配置，其中`path`与`component`是必填的

```
//index.js文件
import {createRouter,createWebHistory } from 'vue-router'

const routes = [
    {
        path:'/',//这两个属性是必传的，这个是显示在路径中的内容
        component:() => import('../pages/login.vue')//这个是要通过路由跳转过去的组件位置(就是说我们要去的地方)
    }
]

const router = createRouter({
    history:createWebHistory(),
    routes  //路由信息
})
export default router//记得将路由暴露出去
```

第一章节(下)——路由模式

我们平时在路径中有看到/#/的就是哈希模式(hash)，这个哈希不是算法

```
//Vue2 mode history Vue3 createWebHistory(历史)，兼容性较差一点，但路径不显示#
//Vue2 mode hash Vue3 createHashHistory(哈希)
//Vue2 mode abstact Vue3 createMemoryHistory
```

- 这个/#/是因为哈希模式是通过`location.hash`去匹配的，我们在控制台中可以打印出'#/'的结果
- 如果在控制台中修改`location.hash`的话，页面中也会进行实时跳转
- 通过监听window上的一个属性`hashchange`我们可以看到跳转之间的变化(`哈希模式`)

```
window.addEventListener('hashchange',(e)=>{
    console.log(e)
})
//哈希原理：通过监听hashchange我们可以看到HashChangeEvent这个属性里面的
//newURL与oldURL，其实也就是新地址跟旧地址(此地址其实就是路由跳转的变化)
```

hash 实现

hash 是 URL 中 hash (#) 及后面的那部分，常用作锚点在页面内进行导航，**改变 URL 中的 hash 部分不会引起页面刷新**

通过 hashchange 事件监听 URL 的变化，改变 URL 的方式只有这几种：

1. 通过浏览器前进后退改变 URL
2. 通过 `<a>` 标签改变 URL
3. 通过window.location改变URL

> 在上面我们演示了哈希的原理，接下来我们看另外的一个常用的，也就是路径中不带#号的`History`的原理，是通过`popstate`来看的

```
window.addEventListener('popstate',(e)=>{
    console.log(e)
})
//通过此属性得到的是PopStateEvent，在里面的state里面我们可以看到
//back(返回路径)current(当前页面)forward(上一个页面)

//监听跳转：通过history.pushState({存点东西},'第二个没啥用','第三个是跳转路径')，跳转后不会被浏览器监听到
```

第二章——命名路由-编程式导航

> 作用：可以简化路由的跳转。
>
> 不借助`<router-link>`实现路由跳转，让路由跳转更加灵活
>
> 除了使用 `<router-link>` 创建 a 标签来定义导航链接，我们还可以借助 router 的实例方法，通过编写代码来实现。
>
> 在路由器中的路由中进行使用，name:"xxx，爱叫什么叫什么，你叫yupi也无所谓，英文"

- 使用name命名路由后，我们第一件事情就是去使用了对应路由的地方进行修改
  - 比如我们原本是，现在就不是路径了，现在得变成名字了=>
  - 注意点：to前面多了冒号，是变成响应式的标记，然后写法变为对象形式
- 稍微提一嘴的a标签(这个router-link会自动转化为a标签，但是跟你直接使用a标签是不一样的，使用router-link的才会是单页面，如果直接使用a标签的话，那其实实际感官上是会抖动一下，然后会刷新页面)

```
<a href="/reg">rrr</a>
```

没错啦，上面都是铺垫，接下来才是`编程式的路由导航`写法

命名路由

除了 `path` 之外，你还可以为任何路由提供 `name`。这有以下优点：

- 没有硬编码的 URL

- `params` 的自动编码 / 解码。
- 防止你在 url 中出现打字错误。
- 绕过路径排序（如显示一个）

步骤

1. 引入useRouter

```
import {useRouter} from 'vue-router'
```

2. 使用router

```
const router = useRouter()
```

3. 字符串用法

```
//import { useRouter } from 'vue-router'
//const router = useRouter()
 //请注意，这个toPage是一个按钮，在页面中使用@click进行绑定
const toPage = () => {
  router.push('/reg')
}
//TS写法
const toPage = (url:string) => {
  router.push(url)//这种写法的话就需要自己在上面@click="toPage('这里传入路径')"，进行配置了
}
```

4. 对象模式

```
import { useRouter } from 'vue-router'
const router = useRouter()

const toPage = () => {
  router.push({//对象形式的写法
    path: '/reg'//path是路径，这里一样是写死了，可以将参数传进来
  })
}
```

5. 命名式写法

```
import { useRouter } from 'vue-router'
const router = useRouter()

const toPage = () => {
  router.push({
    name: 'Reg'//这里需要注意，在前面设置的也需要进行修改，不能够再传入url了，因为我们这里是命名式写法，需要传入命名(此处是写死的做法，下面是传值式的写法)
  })
}

const toPage = (Name:string) => {//TS写法，不用TS写法的话写个Name就行了
  router.push({//一样是对象形式写法
```

```
    name: Name//传值式写法
  })
}
```

第三章——历史记录

`<router-link>`的replace属性

1. 作用：控制路由跳转时操作浏览器历史记录的模式
2. 浏览器的历史记录有两种写入方式：分别为`push`和`replace`，`push`是追加历史记录，`replace`是替换当前记录。路由跳转时候默认为`push`
3. 如何开启`replace`模式：`<router-link replace .....>News</router-link>`
4. replace官方的解释就是替换当前的记录，可以简单的理解为无痕模式。你无法回退到之前的页面中，你做出的每一个指令都是没有回头路的

- router-link使用方法

```
    <router-link replace to="/">Login</router-link>
    <router-link replace style="margin-left:10px" to="/reg">Reg</router-link>
```

- 编程式导航

```
  <button @click="toPage('/')">Login</button>
  <button @click="toPage('/reg')">Reg</button>

import { useRouter } from 'vue-router'
const router = useRouter()

const toPage = (url: string) => {
  router.replace(url)//与router-link的区别就是，编程式导航的replace隐藏在js里面
}
```

> 补充小知识点：
>
> 1. 为什么有router-link了，还需要有编程式导航？
>
>   因为router-link最终会转化为a标签，而当你其实并不是需要a标签的时候，比如上方的例子是按钮，那如何实现？那就得靠路由导航了，路由导航就是没有router-link，所有的逻辑都写在了js里面(想要跳转到哪个页面就靠方法参数传值到js部分来决定)，然后在html页面中进行调用
>
> 2. 注意点：push是对象式的写法(命名路由)，而replace不是噢

go与back

> ### 横跨历史
>
> 该方法采用一个整数作为参数，表示在历史堆栈中前进或后退多少步

```
<button @click="next">前进</button>
<button @click="prev">后退</button>
```

```
const next = () => {
  //前进 数量不限于1
  router.go(1)//不知能前进，还可以后退，后退就是负数，数字的大小决定前进与后退几步
}

const prev = () => {
  //后退，这个就是纯后退了，按一下退一步
  router.back()
}
```

第四章——(路由传参)

Query 路由传参

> 编程式导航 使用 router push 或者 replace 的时候 改为对象形式新增 query 必须传入一个对象

使用前步骤(复习)

1. 引入useRouter，在vue3中需要使用的都要针对性的引入
2. 使用(我更愿意称为激活使用)，const router = useRouter()，这样后面进行使用的时候就直接router.xxx。本质上其实就是调用useRouter了

- query是写在push里面的，且只能接受一个对象

```
//子组件login.vue
const toDetail = (item:Item) =>{
    router.push({
        path:'/reg',
        query:item//item是一个对象
    })
}
//通过url路径，我们可以看到信息其实已经传过来了
```

```
//子组件reg.vue
//<template></template>部分
<template>
<div>价格:{{route.query.name}}</div> //我们通过这种方式取到了query中的值
</template>


//引入路由，注意啦，这是路由不是路由器
import {useRoute} from 'vue-router'

const route = useRoute()
```

Params路由传参

> 编程式导航 使用 router push 或者 replace 的时候 改为对象形式并且只能使用 `name`，`path` 是无效的，然后传入 params

- 使用Params来进行传参的话，不能使用path而是使用name。与此同时，query要变化为params，因为我们已经换频道了，现在是params专场，记得在html页面(或者说是template部分)要将 `route.query.name` 啥的换成 `route.pamas.name`

```
const toDetail = (item: Item) => {
    router.push({
        name: 'Reg',
        params: item
    })
}
```

> 注意点：
>
> 1. pamas的内容是存在内存当中的，这个也是与query传参的一个不同之处，但由此也会造成一个问题，那就是我们一但刷新，这个值是会丢失的

为了解决这个值会丢失的问题，我们延伸出了第三种方式 => 动态路由传参

动态路由传参

> 很多时候，我们需要将给定匹配模式的路由映射到同一个组件。例如，我们可能有一个 `User` 组件，它应该对所有用户进行渲染，但用户 ID 不同。在 Vue Router 中，我们可以在路径中使用一个动态字段来实现，我们称之为 *路径参数*
>
> *路径参数* 用冒号 `:` 表示。当一个路由被匹配时，它的 *params* 的值将在每个组件

步骤：

1. 修改router.push中的params

```
params:item//虽然item本身也是对象，但是换成对象形式写会更好
//修改后
params:{
    id:item.id//这里取名定义的要跟下面 代码块1中的`动态路由参数`部分的 path对应上(取名是指id，而不是item.id)
}
```

2. 进一步改造

```
import { useRoute } from 'vue-router';
import { data } from './list.json'
const route = useRoute()
//我们从data中获取值
data.find(v=>v.id === route.params.id)//直接找find，会返回一个对象
//这个时候提示"此条件将始终返回false"，因为类型"number"和string|string[]没有重叠
//我们知道他的类型是RouteParams，所以我们要进入源码进行寻找原因了，此时进度条在8分08秒
//我们在源码中找到RouteParams，看到他的类型是RouteParamValue|RouteParamValue[]
//RouteParamValue类型是等于string的，数字的话，三个等于号的情况下将永远不会等于字符串。所以我们这里使用Number
const item = data.find(v=>v.id === Number(route.params.id))//在data中找我们需要的东西
```

3. 此时需要同步进行改变的部分

```
//改变前
//<div>品牌：{{ route.params?.name }}</div>
//<div>价格：{{ route.params?.price }}</div>
//<div>ID：{{ route.params?.id }}</div>
//改变后
<div>品牌：{{ item?.name }}</div> //记得要加上问号，这是可选的意思，不然要是没有name岂不是要爆错，所以要是找不到就不要了，给自己留个余地
<div>价格：{{ item?.price }}</div>
<div>ID：{{ item?.id }}</div>
```

```
//代码块1
const routes:Array<RouteRecordRaw> = [
    {
        path:"/",
        name:"Login",
        component:()=> import('../components/login.vue')
    },
    {
        //动态路由参数
        path:"/reg/:id",//注意冒号后面的id，这个就是要对应上的部分
        name:"Reg",
        component:()=> import('../components/reg.vue')
    }
]
```

> find知识点补充：
>
> find 函数基本格式：`let obj=this.list.find(item=>item.code===val)`
> 首先在这里我们要知道的两个基础知识
> 一、find 是一个查找函数。
> 二、箭头函数 `find(item=>item.code===val)` 相当于 `find(item){item.code===val}`
>
> 其中 `list` 是数组，`this.list.find()` 是指在 `list` 数组中找某样东西，`item` 是 `find()` 函数的寻找某样东西的根据，也可以说是 `id` 或者是主键。后面 `item.code===val` 是查找这样东西的条件，只有这个 `item.code` 完全等于 `val` 的时候，才算是找到，才能赋值给 `obj`
>
> ```
> //举例
> //定义一个数组，有两条数据
> companyOptions: [
>  {
>    label: '饿了么',
>    value: 0,
>  },
>  {
>    label: '美团',
>    value: 1,
>  },
> ]
> //需要找到value为1，label为美团的数组
> let obj=this.companyOptions.find(item=>item.value===1)
> ```

小总结

二者的区别

1. query 传参配置的是 path，而 params 传参配置的是 name，在 params 中配置 path 无效
2. query 在路由配置不需要设置参数，而 params 必须设置
3. query 传递的参数会显示在地址栏中
4. params 传参刷新会无效，但是 query 会保存传递过来的值，刷新不变；
5. 路由配置

第五章——嵌套路由

> 一些应用程序的 UI 由多层嵌套的组件组成。在这种情况下，URL 的片段通常对应于特定的嵌套组件结构

在原有基础上的变化

1. 我们原本是路由器里面放路由，而嵌套路由就是在路由器里的路由里面继续放路由的一个过程。所以他叫嵌套路由
2. 写法上是在路由里面的`children`属性，英文翻译过来就是孩子们的意思，其实也就是子路由(嵌套路由)的意思
3. 注意点：
    1. 你在路由里面继续放路由了，那此时这个二级路由记得要给他们找个容器进行显示
    2. 你要通过跳转的时候，路径上不能直接写二级路由的path，要先写上一级路由的path然后/二级路由的path。因为他这个寻找逻辑是一层层寻找的，先找到一级路由再继续往下寻找(直接写二级路由的话，在一级路由肯定是找不到的，就会噶了)
    3. `children`配置只是另一个路由数组，就像`routes`本身一样。因此，你可以根据自己的需要，不断地嵌套视图

```
const routes: Array<RouteRecordRaw> = [
    {
        path: "/user",//一级路由
        component: () => import('../components/footer.vue'),
        children: [//注意看，嵌套路由的写法是数组里面装对象的形式
            {
                path: "",//嵌套路由1号(也叫做二级路由)
                name: "Login",
                component: () => import('../components/login.vue')
            },
            {
                path: "reg",//嵌套路由二号
                name: "Reg",
                component: () => import('../components/reg.vue')
            }
        ]
    },

]
```

第六章——命名视图

> default：默认的意思，这个具体是什么意思呢？就是说你默认显示出来的就是这个default
>
> </router-view name="名字，看你在components是怎么去命名的">，就可以具体显示你想要的页面(你可以定义多个view也就是视图)，跳转的时候同时显示多个视图的内容
>
> 命名视图可以在同一级（同一个组件）中展示更多的路由视图，而不是嵌套显示。 命名视图可以让一个组件中具有多个路由渲染出口，这对于一些特定的布局组件非常有用。 命名视图的概念非常类似于 “具名插槽”，并且视图的默认名称也是 `default`

```
import { createRouter, createWebHistory, RouteRecordRaw } from 'vue-router'


const routes: Array<RouteRecordRaw> = [
    {
        path: "/",
        components: {//注意这里发生了变化，多了一个s
            default: () => import('../components/layout/menu.vue'),//转到默认的
            header: () => import('../components/layout/header.vue'),//router-view里面的name填header，下面content同理，这个一旦填写了，我们在template中可以定义多个<router-view>视图，然后跳转的时候同时显示多个视图的内容
            content: () => import('../components/layout/content.vue'),
        }
    },
]

const router = createRouter({
    history: createWebHistory(),
    routes
})

export default router
```

> 对应 Router-view 通过 name 对应组件

```
    <div>
        <router-view></router-view>
        <router-view name="header"></router-view>
        <router-view name="content"></router-view>
    </div>
```

- 注意事项
- 这个是components而不是component，因为通过这个多加一个s的，我们可以一次性跳转到多个路由视图中，需要在template中去额外定义多的路由视图，并且通过你在components中取的名字来进行绑定，比如在上面代码块中，我们使用了header跟content这两个名字，那在使用的时候就需要name绑定这两个名字

第七章——重定向-别名

重定向 - redirect

> 字符串形式配置，访问 / 重定向到 /user （地址栏显示 /, 内容为 /user 路由的内容）
>
> 他的属性名就是redirect:

三次书写形式

- 字符串的写法：
  - ```
    const routes: Array<RouteRecordRaw> = [
        {
            path:'/',
            component:()=> import('../components/root.vue'),
            redirect:'/user1',//写法上的区别在这里，这redirect是字符串的写法
            children:[
                {
                    path:'/user1',
                    components:{
                        default:()=> import('../components/A.vue')
                    }
                },
                {
                    path:'/user2',
                    components:{
                        bbb:()=> import('../components/B.vue'),
                        ccc:()=> import('../components/C.vue')
                    }
                }
            ]
        }
    ]
    ```
- 对象的写法：
  - ```
    const routes: Array<RouteRecordRaw> = [
        {
            path: '/',
            component: () => import('../components/root.vue'),
            redirect: { path: '/user1' },//这里是对象形式的写法，path也会自动对应到下方children的path(子路由)中
            children: [
                {
                    path: '/user1',
                    components: {
                        default: () => import('../components/A.vue')
                    }
                },
                {
                    path: '/user2',
                    components: {
                        bbb: () => import('../components/B.vue'),
                        ccc: () => import('../components/C.vue')
                    }
                }
            ]
        }
    ]
    ```
- 函数模式（可以传参）=> 回调形式
  - ```
    const routes: Array<RouteRecordRaw> = [
    ```
    ```

```
    {
        path: '/',
        component: () => import('../components/root.vue'),
        redirect: (to) => {//写法是以to的形式，把to括起来的这个括号可加可不加
            return {//必须要返回值，也就是return
                path: '/user1',
                query: to.query//由于这里to没有传值过来，我们可以暂时自己写一个
                //query:{name:"小余"}，这个name=小余会在URL，也就是路径中体现
出来
            }
        },
        children: [
            {
                path: '/user1',
                components: {
                    default: () => import('../components/A.vue')
                }
            },
            {
                path: '/user2',
                components: {
                    bbb: () => import('../components/B.vue'),
                    ccc: () => import('../components/C.vue')
                }
            }
        ]
    }
]
```

别名 alias

> **将 `/` 别名为 `/`root，意味着当用户访问 `/`root 时，URL 仍然是 `/user`，但会被匹配为用户正在访问 `/`**
>
> 通俗的说就是给路由取多个名字，但访问的组件都是同一个(比如我们都知道鲁迅跟周树人，这是两个名字，但其实都是指同一个人)

```
const routes: Array<RouteRecordRaw> = [
    {
        path: '/',//我就是下面说的path路径
        component: () => import('../components/root.vue'),//访
问/root，/root2，/root3都是展示这个组件
        alias:["/root","/root2","/root3"],//三个名字，或者说是上面这个path路径的外号或
者说是别名
        children: [
            {
                path: 'user1',
                components: {
                    default: () => import('../components/A.vue')
                }
            },
            {
                path: 'user2',
                components: {
                    bbb: () => import('../components/B.vue'),
```

```
                    ccc: () => import('../components/C.vue')
                }
            }
        ]
    }
]
```

第八章——导航守卫-前置守卫

全局前置守卫

> router.beforeEach
>
> 小满称呼：中间件

```
//所有跳转、后退都会走这个函数
router.beforeEach((to, form, next) => {
    console.log(to, form);
    next()
})
```

> 小满在这里做的是一个登录的校验，使用了组件库这里都不进行说明

小满在这里使用了一个@别名，在这里对这个进行一个解释

- 在vite.config.ts文件下，代码如下

```
export default defineConfig({
    plugins:[vue(),vueJsx()],
    resolve:{
        alias:{
            '@':fileURLToPath(new URL('./src',import.meta.url)),
        }
    }
})
```

- 这样的作用是对你的文件起了一个别名，此处的@的别名对应的是./src。从上面的路径中我们也可以看到

resolve.alias

在组件之间相互引用时，可能是下面这样的：

```
import Hello from '../src.components/Hello';
```

其中的路径是相对于当前页面的。 但是如果嵌套等更为复杂，那么写起来会比较麻烦。但是如果我们通过这样的配置：

```
  resolve: {
    extensions: ['.js', '.vue', '.json'],
    alias: {
      'vue$': 'vue/dist/vue.esm.js',
      '@pages': path.join(__dirname, "..", "src", "pages"),
      "@components": path.join(__dirname, "..", "src", "components"),
      // 注意： 静态资源通过src，不能这么设置。
      // "@assets": path.join(__dirname, "..", "src", "assets"),
    }
```

其中 vue$ 表示引入 vue，就可以像下面这么写：

```
import Vue from 'vue'
```

另外，对于 @pages 和 @components 我们就可以直接引用了，而省去了一大堆的复杂应用，另外通过 @可以消除歧义。如下所示：

```
  import Hello from '@components/Hello';
import App from '@pages/App'
```

值得注意的时： 在 webpack.config.js 中我们不能使用../ 以及./ 这种形式的路径方式，而是通过 path.join 和 __dirname 这种形式来表示路径，否则会报错。

另外： 在组件中，我们会引用一些静态文件，即 static 下的文件， 这时我们就不能用 alias 下的配置了，而必须使用一般的配置方式。

> 我没找到vite的，根据webpack触类旁通

```
const rules = reactive({
    user:[
        {
            required:true,//证明是不是必填
            message:"失败给提示信息",
            type:"string",//类型
            tiggle:"change"//触发的条件
        }
    ]
})
```

每个守卫方法接收三个参数：

```
to: Route， 即将要进入的目标 路由对象；
from: Route，当前导航正要离开的路由；
next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 confirmed (确认的)。
next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。
next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。
```

案例 权限判断

```
const whileList = ['/']//白名单

router.beforeEach((to, from, next) => {
    let token = localStorage.getItem('token')
    //白名单 有值 或者登陆过存储了token信息可以跳转 否则就去登录页面
    if (whileList.includes(to.path) || token) {
        next()//next()是放行的意思
    } else {
        next({
            path:'/'//回到登录界面
        })
    }
})
```

> 呃，这个使用组件库的过程不好写笔记，建议自己看看视频

全局后置守卫(内容较深，后续回来补笔记)

> 使用场景一般可以用来做 loadingBar
>
> 你也可以注册全局后置钩子，然而和守卫不同的是，这些钩子不会接受 `next` 函数也不会改变导航本身：
>
> 和前置路由守卫相比为什么没有next功能呢？那是因为前置路由守卫是看大门前面的，而后置路由守卫则是在大门的后面。前面的保安如果都放行进来了，那后面的保安还要再确认一遍就太过繁复也没有必要啦

```
router.afterEach((to,from)=>{
    Vnode.component?.exposed?.endLoading()
})
```

> loadingBar 组件

```
<template>
    <div class="wraps">
        <div ref="bar" class="bar"></div>
    </div>
</template>

<script setup lang='ts'>
import { ref, onMounted } from 'vue'
let speed = ref<number>(1)//这里是通过TS的规范类型为数字number类型，以下类似
let bar = ref<HTMLElement>()
let timer = ref<number>(0)//默认为0
const startLoading = () => {//开始进度条
    let dom = bar.value as HTMLElement;//因为有可能为undefined，所以我们使用as进行一个断言为HTMLElement
    speed.value = 1//初始化speed的值
    timer.value = window.requestAnimationFrame(function fn()
{//requestAnimationFrame会将回流重绘收集起来只走一次，性能会更好。这里可以使用箭头函数，但我们选择使用function。回调函数只走一次，所以我们起了一个fn的名字拿去递归
        if (speed.value < 90) {
```

```
            speed.value += 1;
            dom.style.width = speed.value + '%'
            timer.value = window.requestAnimationFrame(fn)//递归
        } else {
            speed.value = 1;
            window.cancelAnimationFrame(timer.value)
        }
    })

}

const endLoading = () => {//结束进度条
    let dom = bar.value as HTMLElement;
    setTimeout(() => {
        window.requestAnimationFrame(() => {//箭头函数的写法
            speed.value = 100;//其实显示的就是进度条的进度，结束了就是100%
            dom.style.width = speed.value + '%'
        })
    }, 500)

}


defineExpose({//defineExpose 可以将方法主动暴露出来，通过对象的形式。然后能够在父组件中进行接收
    //当父组件通过模板引用的方式获取到当前组件的实例，获取到的实例会像这样 { startLoading: number, endLoading: number } (ref 会和在普通实例中一样被自动解包)
    startLoading,
    endLoading
})
</script>

<style scoped lang="less">//样式
.wraps {
    position: fixed;
    top: 0;
    width: 100%;//这些都是决定进度条的样式的
    height: 2px;
    .bar {
        height: inherit;
        width: 0;
        background: blue;
    }
}
</style>
```

main.ts

```
import loadingBar from './components/loadingBar.vue'
const Vnode = createVNode(loadingBar)//createVNode是Vue3中提供的方法，用于创造DOM节点
render(Vnode, document.body)
console.log(Vnode);

router.beforeEach((to, from, next) => {
    Vnode.component?.exposed?.startLoading()//问号为可选的意思
})

router.afterEach((to, from) => {
    Vnode.component?.exposed?.endLoading()
})
```

> 上述牵扯到的知识点补充：
>
> 计时器一直是javascriptz动画的核心技术。而编写动画循环的关键是要知道延迟时间多长合适。一方面，循环间隔必须足够短，这样
> 才能让不同的动画效果显得平滑流畅；另一方面，循环间隔还要足够长，这样才能确保浏览器有能力渲染产生的变化
>
> 大多数电脑显示器的刷新频率是60Hz，大概相当于每秒钟重绘60次。大多数浏览器都会对重绘操作加以限制，不超过显示器的重绘频率，因为即使超过那个频率用户体验也不会有提升。因此，最平滑动画的最佳循环间隔是1000ms/60,约等于16.6ms
>
> 而setTimeout和setIntervall的问题是，它们都不精确。它们的内在运行机制决定了时间间隔参数实际上只是指定了把动画代码添加到浏览器U线程队列中以等待执行的时间。如果队列前面已经加入了其他任务，那动画代码就要等前面的任务完成后再执行
>
> requestAnimationFrame.采用系统时间间隔，保持最佳绘制效率，不会因为间隔时间过短，造成过度绘制，增加开销；也不会因为间隔时间太长，使用动画卡顿不流畅，让各种网页动画效果能够有一个统一的刷新机制，从而节省系统资源，提高系统性能，改善视觉效果
>
> ---
>
> ### requestAnimationFrame 比起 setTimeout、setInterval 的优势主要有两点：
>
> **1、requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来，在一次重绘或回流中就完成，并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率，一般来说，这个频率为每秒 60 帧。**
>
> **2、在隐藏或不可见的元素中，requestAnimationFrame 将不会进行重绘或回流，这当然就意味着更少的的 cpu，gpu 和内存使用量。**

第九章——路由元信息

路由元信息

通过路由记录的 `meta` 属性可以定义路由的**元信息**。使用路由元信息可以在路由中附加自定义的数据，例如：

- 权限校验标识。
- 路由组件的过渡名称。
- 路由组件持久化缓存 (keep-alive) 的相关配置。
- 标题名称

> 有时我们想将一些信息附加到路由上，如过渡名称、谁可以访问路由等。这些事情可以通过接收属性对象的 `meta` 属性来实现，并且它可以在路由地址和导航守卫上都被访问到。定义路由的时候你可以这样配置 `meta` 字段

我们可以在**导航守卫**或者是**路由对象**中访问路由的元信息数据。

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      component: () => import('@/views/Login.vue'),
      meta: {
        title: "登录"//这里定义什么都行，最终这里的数据是会被获取到的
      }
    },
    {
      path: '/index',
      component: () => import('@/views/Index.vue'),
      meta: {
        title: "首页",
      }
    }
  ]
})
```

使用 TS 扩展

> 如果不使用扩展 将会是 unknow 类型

```
declare module 'vue-router' {
  interface RouteMeta {//接口写法，将title变成是否可选的string类型
    title?: string//这个问号不写也没有关系，意思就变为title必须为string(字符串)类型
  }
}
```

如何获取元信息

在组件中可以通过 this.$route.meta 来获取

在路由中获取:

```
 router.beforeEach((to, from, next) => {
 //通过to.meta来获取路由元信息
 console.log(to.meta.requiresAuth);
})
```

- 通过这类方法我们可以在配置文件中通过router.beforeEach，也就是前置路由守卫中进行设置
  - 前置路由守卫有三个参数分别为`to`，`from`，`next`。我们这里使用到to，意思是将要去哪里的意思，我们通过to.meta.title拿到了我们在routers中对应的路由中配置的mate中的title，然后将这个值赋值给document.title
  - 产生的效果就是能够在路由跳转后，网页的标题也随之发生改变，变成对应路由的信息(比如我们设定好的登录和首页)

第十章——路由过渡动效

过渡动效

想要在你的路径组件上使用转场，并对导航进行动画处理，你需要使用 v-slot API：

使用组件库的时候记得先安装(npm安装方式：npm install animate.css)后import引入

```
    <router-view #default="{route,Component}">//#default，像这种使用井号的是简写方式
        //能解出来两个属性，route为当前路由的信息，Component就是当前的组件
        <transition  :enter-active-class="`animate__animated ${route.meta.transition}`">//animate__animated是动画组件库的前缀，然后active-class就是高亮样式的选项，动画组件库和路由的结合形成的过度效果
            <component :is="Component"></component>
        </transition>
    </router-view>
```

上面的用法会对所有的路由使用相同的过渡。如果你想让每个路由的组件有不同的过渡，你可以将元信息和动态的 `name` 结合在一起，放在 `<transition>` 上：

```
declare module 'vue-router'{
  interface RouteMeta {
     title:string,
     transition:string,
  }
}

const router = createRouter({
history: createWebHistory(import.meta.env.BASE_URL),
routes: [
 {
   path: '/',
   component: () => import('@/views/Login.vue'),
   meta:{
      title:"登录页面",
      transition:"animate__fadeInUp",//动画组件库
   }
 },
 {
   path: '/index',
   component: () => import('@/views/Index.vue'),
   meta:{
      title:"首页！！！",
      transition:"animate__bounceIn",
   }
 }
]
})
```

第十一章——滚动行为

使用前端路由，当切换到新路由时，想要页面滚到顶部，或者是保持原先的滚动位置，就像重新加载页面那样。vue-router 可以自定义路由切换时页面如何滚动。

> 当创建一个 Router 实例，你可以提供一个 `scrollBehavior` 方法

```
const router = createRouter({
  history: createWebHistory(),
  scrollBehavior: (to, from, savePosition) => {//scrollBehavior滚动的属性，
savePosition参数为标记的距离
    if(savePosition){
        return savePosition
    }else{
        return{
            top:0//或者可以不进行判断直接return你想要的数据，就跟随你的心意距离顶部还有多少
距离
        }
    }
  },
```

```
const router = createRouter({
  history: createWebHistory(),
  scrollBehavior: (to, from, savePosition) => {//scrollBehavior滚动的属性，
savePosition参数为标记的距离
    console.log(to, '===============>', savePosition);
    return new Promise((r) => {//异步
      setTimeout(() => {
        r({
          top: 10000//top是Vue3的写法
        })
      }, 2000);
    })
  },
```

scrollBehavior 方法接收 to 和 from 路由对象。第三个参数 savedPosition 当且仅当 popstate 导航 (通过浏览器的 前进 / 后退 按钮触发) 时才可用。

scrollBehavior 返回滚动位置的对象信息，长这样：

- { left: number, top: number }

```
const router = createRouter({
  history: createWebHistory(),
  scrollBehavior: (to, from, savePosition) => {
    return {
       top:200//这个就是跟随心意的写法了，不进行if判断，直接根据自己喜好返回距离顶部的距离
    }
  },
```

第十二章——动态路由

> 我们一般使用动态路由都是后台会返回一个路由表前端通过调接口拿到后处理 (后端处理路由)
>
> 主要使用的方法就是 router.addRoute
>
> 调用接口需要安装axios，命令：npm install axios -S
>
> import引入axios

添加路由

动态路由主要通过两个函数实现。`router.addRoute()` 和 `router.removeRoute()`。它们只注册一个新的路由，也就是说，如果新增加的路由与当前位置相匹配，就需要你用 `router.push()` 或 `router.replace()` 来手动导航，才能显示该新路由

```
router.addRoute({ path: '/about', component: About })
```

删除路由

有几个不同的方法来删除现有的路由：

- 通过添加一个名称冲突的路由。如果添加与现有途径名称相同的途径，会先删除路由，再添加路由：

  ```
  router.addRoute({ path: '/about', name: 'about', component: About })
  // 这将会删除之前已经添加的路由，因为他们具有相同的名字且名字必须是唯一的
  router.addRoute({ path: '/other', name: 'about', component: Other })
  ```

- 通过调用 `router.addRoute()` 返回的回调：

  ```
  const removeRoute = router.addRoute(routeRecord)
  removeRoute() // 删除路由如果存在的话
  ```

  - 当路由没有名称时，这很有用。
- 通过使用 `router.removeRoute()` 按名称删除路由：

  ```
  router.addRoute({ path: '/about', name: 'about', component: About })
  // 删除路由
  router.removeRoute('about')
  ```

  > 需要注意的是，如果你想使用这个功能，但又想避免名字的冲突，可以在路由中使用 `Symbol` 作为名字。

当路由被删除时，**所有的别名和子路由也会被同时删除**

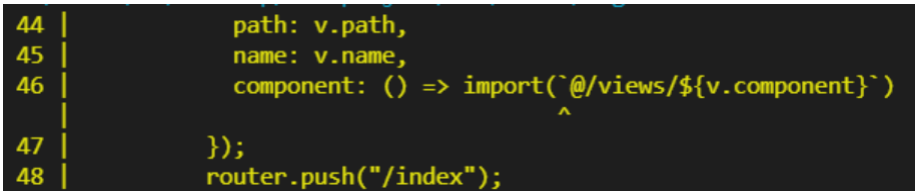
查看现有路由

Vue Router 提供了两个功能来查看现有的路由：

- router.hasRoute()：检查路由是否存在。
- router.getRoutes()：获取一个包含所有路由记录的数组。

案例

> ## 前端代码
>
> **注意一个事项 vite 在使用动态路由的时候无法使用别名 @ 必须使用相对路径**




```
const initRouter = async () => {
    const result = await axios.get('http://localhost:9999/login', { params: formInline });//get是params参数，post没有限制。formInline是表单的信息，通过.user.password之类的能获取到账号密码。
    result.data.route.forEach((v: any) => {//后端返回的信息进行动态添加
        router.addRoute({
            path: v.path,
            name: v.name,
                                    //这儿不能使用@
            component: () => import(`../views/${v.component}`)//最关键的组件需要我们动态的去拼接，这里使用模板字符串
            //然后就是import()动态引入
        })
        router.push('/index')//跳转页面，验证通过，进入登录页面
    })
    console.log(router.getRoutes());

}
```

> 后端代码 nodejs express

```
import express, { Express, Request, Response } from 'express'

const app: Express = express()

app.get('/login', (req: Request, res: Response) => {//这里使用get请求所以必须使用query参数
    res.header("Access-Control-Allow-Origin", "*");
    if (req.query.user == 'admin' && req.query.password == '123456') {//根据前端传过来的信息返回不同的信息回去
        res.json({
            route: [
                {
                    path: "/demo1",
                    name: "Demo1",
                    component: 'demo1.vue'
                },
                {
                    path: "/demo2",
                    name: "Demo2",
                    component: 'demo2.vue'
                },
                {
                    path: "/demo3",
                    name: "Demo3",
                    component: 'demo3.vue'
                }
            ]
        })
```

```
    }else{
        res.json({
            code:400,
            mesage:"账号密码错误"
        })
    }
})

app.listen(9999, () => {
    console.log('http://localhost:9999');

})
```