1.6 路由

Vue Router是官方路由库，它和Vue.js深度整合，能轻易创建单页应用程序，和vue3搭配的版本将是vue-router 4.x。

1.6.1 快速起步

快速体验vue-router 4的方式：

- CDN: `<script src="https://unpkg.com/vue-router@next"></script>`
- 在我们的vue项目中使用：

```
npm i vue-router@next -S
```

下面我们做第一个使用范例：将课程列表修改为多页面

添加入口，App.vue

```
<router-view></router-view>
```

配置路由，router/index.js

```
import { createRouter, createWebHashHistory } from "vue-router";
import CourseList from "/comps/CourseList.vue";
import CourseAdd from "/comps/CourseAdd.vue";

// 1. 配置路由
const routes = [
  { path: "/", component: CourseList },
  { path: "/add", component: CourseAdd },
];

// 2. 创建路由器实例
export default createRouter({
  // 3. 提供一个history实现
  history: createWebHashHistory(),
  routes,
});
```

引入vue-router插件，main.js

```
import router from "./router/index";

createApp(App)
  .use(router)
  .mount("#app");
```

CourseList现在需要自己获取并维护课程数据，并且不再需要传递属性：

```
<script>
import { getCourses } from "../api/course";
export default {
data() {
 return {
   selectedCourse: "",
 };
},
setup() {
 const courses = ref([]);
 getCourses().then(result => (courses.value = result));
 return { courses };
},
// props: {
//   courses: {
//     type: Array,
//     required: true,
//   },
// },
};
</script>
```

定义接口，api/course.js

```
const courses = ["web全栈架构师", "web高级工程师"];

export function getCourses() {
return Promise.resolve(courses);
}
```

跳转新增课程，CourseList.vue

```
<p><router-link to="/add">新增</router-link></p>
```

CourseAdd.vue自己负责course新增逻辑，不需要在传入属性

```
<template>
```

```
<input type="text" v-model="course" @keydown.enter="addCourse" />
<button v-on:click="add">新增课程</button>
</template>
<script>
import { ref } from "vue";
import { addCourse } from "../api/course";
import { useRouter } from "vue-router";

export default {
setup() {
 const course = ref("");
 const router = useRouter();
 const add = () => {
   addCourse(course.value);
   router.push("/");
 };
 return { course, add };
},
// props: {
//   // model-value
//   course: {
//     type: String,
//     required: true,
//   },
// },
// data() {
//   return {
//     course: ''
//   }
// },
// emits: ["update:course", "add"],
// methods: {
//   addCourse() {
//     this.$emit("add");
//     // this.course = ''
//   },
// },
};
</script>
```

1.6.2 动态路由匹配

下面我们想看看课程详情，我们的链接会是这个样子`/course/1`，不同的课程id是不一样的，vue-router中解决此类问题使用`动态路由匹配`特性。

基本用法

- 路由配置：`{ path: "/course/:id", component: CourseDetail }`
- 参数获取：`this.$route.params.id` 或 `useRoute().params.id`

范例：课程详情页

修改路由配置，router/index.js

```
import CourseDetail from "/comps/CourseDetail.vue";
const routes = [
  { path: "/", component: CourseList },
  { path: "/about", component: CourseAdd },
  { path: "/course/:id", component: CourseDetail },
];
```

> 顺便修改一下数据结构，api/course.js：
>
> ```
> const courses = [
> { id: 1, name: "web全栈架构师", price: 99 },
> { id: 2, name: "web高级工程师", price: 99 },
> ];
> ```

导航，CourseList.vue

```
<li
    v-for="c in courses"
    :key="c.id"
    :class="{ active: selectedCourse === c }"
    @click="selectedCourse = c"
    >
  <router-link :to="'/course/' + c.id">{{ c.name }}</router-link>
</li>
```

详情页，CourseDetail.vue

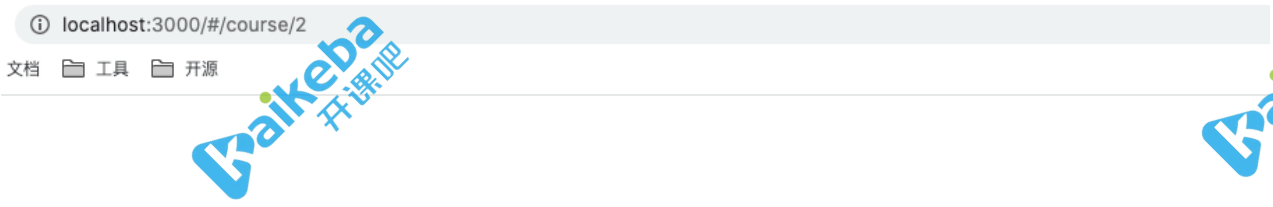
```
<template>
  <div>
    <h3>{{ course.name }}</h3>
    <p>id: {{ $route.params.id }}</p>
    <p>price: {{ course.price }}</p>
  </div>
</template>
```

```
<script>
import { ref } from "vue";
import { useRoute } from "vue-router";
import { getCourseById } from "../api/course";
export default {
  setup() {
    const course = ref({ name: "", price: "" });
    const route = useRoute();
    console.log(route.params.id);
    getCourseById(route.params.id).then(ret => {
      console.log(ret);
      course.value = ret;
    });
    return { course };
  },
};
</script>
```

> api定义，api/course.js
>
> ```
> export function getCourseById(id) {
> return Promise.resolve(courses.find(c => c.id == id));
> }
> ```

响应参数变化

参数变化时，组件实例会复用，导致页面不会响应参数的变化。注意下图id变为2，显示内容还是1的。



解决方案：

- 监听params：`this.$watch(() => this.$route.params, (params, prevParams) => {})`
- 利用导航钩子

```
async beforeRouteUpdate(to, from) {
  // 响应路由变化
  this.userData = await fetchUser(to.params.id)
},
```

改进我们案例，CourseDetail.vue

```
watch(
  () => route.params,
  () => {
    getCourseById(route.params.id).then(ret => {
      course.value = ret;
    });
  }
);
```

通配或404处理

```
const routes = [
  // 下面配置会匹配所有path，匹配内容放入`$route.params.pathMatch`
  { path: '/:pathMatch(.*)*', name: 'NotFound', component: NotFound },
  // 匹配所有以`/user-`开头path，匹配内容放入`$route.params.afterUser`
  { path: '/user-:afterUser(.*)', component: UserGeneric },
]
```

范例中的404处理

```
import NotFound from "/comps/NotFound.vue";
const routes = [{ path: "/:pathMatch(.*)*", component: NotFound },];
```

NotFound.vue

```
<template>
  <div>
    <h1>404 Not Found!</h1>
    <p>{{ $route.params.pathMatch }}</p>
  </div>
</template>

<script>
export default {};
</script>
```

1.6.3 嵌套路由

组件之间的嵌套常常会用嵌套路由形式与之对应。

范例：我们修改列表页、详情页和新增页路由配置，使他们产生嵌套关系

```
const routes = [
  {
    path: "/",
    redirect: "/course",
  },
  {
    path: "/course",
    component: CourseList,
    children: [
      { path: "/course/:id", component: CourseDetail },
      { path: "/course/add", component: CourseAdd },
    ],
  },
  { path: "/:pathMatch(.*)*", component: NotFound },
];
```

CourseList中发生两个变化：

- 需要添加一个`router-view`显示嵌套路由内容
- 导航也随之变化

```
<p><router-link to="/course/add">新增</router-link></p>
<router-view></router-view>
```

> 一个功能在一个页面内出现，用户体验更好了。

1.6.4 编程导航

除了router-link导航之外，其实还能使用router实例提供的方法进行编程导航。

可用方法如下：

声明式	编程式
`<router-link :to="...">`	`router.push(...)`
`<router-link :to="..." replace>`	`router.replace(...)`

常见用法：

```
// 传递path
router.push(`/user/${username}`)
// 或者
router.push({ path: `/user/${username}` })
```

范例：新增课程按钮形式实现，CourseList.vue

```
<button @click="$router.push('/add')">新增</button>
```

导航时传参：

```
// 使用`name`和`params`搭配，以利用自动的URL编码；不能使用`path`和`params`搭配
router.push({ name: 'user', params: { username } })
```

范例：修改列表跳转链接为点击导航，CourseList.vue

```
<ul>
  <li @click="showDetail(c)">
    {{ c.name }}
  </li>
</ul>
```

```
setup() {
  const router = useRouter();
  const selectedCourse = ref(null);
  const showDetail = c => {
    selectedCourse.value = c;
    router.push({ name: "detail", params: { id: c.id } });
  };

  return { courses, showDetail };
},
```

路由定义时需要一个name，router/index.js

```
{ path: "/course/:id", name: "detail", component: CourseDetail }
```

> 使用命名路由有以下好处：
>
> - 无需编写复杂URL
> - `params`自动编码/解码
> - 避免path之间的排名竞争

router-link中也可使用：

```
<router-link :to="{ name: 'bar', params: { param: 'abc' }}">Bar</router-link>
```

1.6.5 路由守卫

路由守卫用于守卫路由导航、重定向或取消，有以下几种守卫路由的方式：

- 全局守卫
- 单个路由守卫
- 组件内路由钩子

全局守卫

范围最大，任何路由导航都会触发回调

```
router.beforeEach((to, from) => {})
```

范例：守卫新增课程页面，若未登录不能访问

```
const router = createRouter({});

router.beforeEach((to, from, next) => {
  if (to.path === "/add") {
    if (localStorage.getItem("token")) {
      next();
    } else {
      next({ path: "/login", query: { redirect: to.path } });
    }
  } else {
    next();
  }
});
export default router;
```

单个路由守卫

可以在路由配置中定义 `beforeEnter` 守卫，它仅作用于该路由。

```
const routes = [
  {
    path: '/users/:id',
    component: UserDetails,
    beforeEnter: (to, from) => {},
  },
]
```

组件内路由守卫

组件可使用以下钩子定义路由守卫：

- `beforeRouteEnter`
- `beforeRouteUpdate`
- `beforeRouteLeave`

范例：把之前的弹窗提示从App中移过来，CourseList.vue：

```
<message v-if="showMsg" @close="showMsg = false">
  <template v-slot:title> 恭喜 </template>
  <template v-slot:default> 新增课程成功！ </template>
</message>
```

```
// message显示控制从App.vue中移过来
beforeRouteEnter(to, from, next) => {
  // 如果从add页面过来，且操作成功则显示弹窗
  if (from.path === '/course/add' && to.query.action === "success") {
    next(vm => vm.showMsg.value = true)
  }
}
setup() {
  const showMsg = ref(false);
  return { showMsg };
}
```

新增成功传递一个action参数出去，CourseAdd.vue

```
const add = () => {
  addCourse(course.value);
  router.push({ path: "/course", query: { action: "success" } });
};
```

1.6.6 路由元数据

有时需要在定义路由时附加额外信息，此时可以利用路由元数据，这样导航时可以访问这些信息。

```
{
  path: '/about',
  component: About
  meta: { foo: 'bar' }
}
```

范例：元数据定义路由权限验证要求，提高代码通用性

```
{
  path: "/course/add",
  component: CourseAdd,
  meta: { requiresAuth: true }
},
```

```
router.beforeEach((to, from, next) => {
  // 这样就不用像之前那样写死了
  if (to.meta.requiresAuth) {
```

1.6.7 路由懒加载

打包时将单个路由组件分片打包，访问时才异步加载，可以有效降低app尺寸和加载时间。

定义异步路由

```
const UserDetails = () => import('./views/UserDetails')
const router = createRouter({
  // ...
  routes: [{ path: '/users/:id', component: UserDetails }],
})
```

1.6.8 composition api

随着composition api诞生，我们就有在setup中获取router或者route实例的需求。

setup中获取router或者route实例方法

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

export default {
  setup() {
    const router = useRouter()
    const route = useRoute()
    //...
  }
}
```

setup里面获取导航钩子

```
import { onBeforeRouteLeave, onBeforeRouteUpdate } from 'vue-router'

export default {
  setup() {
    // 等效于beforeRouteLeave，但是不能访问`this`
    onBeforeRouteLeave((to, from) => {})

    // 等效于beforeRouteUpdate，但是不能访问`this`
    onBeforeRouteUpdate((to, from) => {})
  },
}
```

> beforeRouteEnter没有等效方法，文档没有明确这样做的原因，猜测是由于setup执行时已经创建组件实例，此时刻过于靠后，已经失去了守卫的意义。

useLink()暴露Router-Link内部行为，常用于使用composition方式自定义Link组件

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

export default {
  setup() {
    const { route, href, isActive, isExactActive, navigate } = useLink(props)

  }
}
```

1.6.9 缓存和过度动画

缓存

结合keep-alive可以对路由组件做缓存以保存组件状态，优化用户体验。v4中使用方式有比较大的变化，它们必须出现在router-view内部结合v-slot api使用：

```
<router-view v-slot="{ Component }">
  <keep-alive>
    <component :is="Component" />
  </keep-alive>
</router-view>
```

过度动画

要制作路由导航的过度动画，我们还是要结合v-slot api：

```
<router-view v-slot="{ Component }">
  <transition name="fade">
    <component :is="Component" />
  </transition>
</router-view>
```

添加css测试一下

```
<style scoped>
.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.5s ease;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}
</style>
```

单个路由动画

```
const routes = [
  {
    path: '/custom-transition',
    component: PanelLeft,
    meta: { transition: 'slide-left' },
  },
  {
    path: '/other-transition',
    component: PanelRight,
    meta: { transition: 'slide-right' },
  },
]
```

```
<router-view v-slot="{ Component, route }">
  <transition :name="route.meta.transition || 'fade'">
    <component :is="Component" />
  </transition>
</router-view>
```

甚至可以根据路由之间关系动态决定动画类型

```
<router-view v-slot="{ Component, route }">
  <transition :name="route.meta.transition">
    <component :is="Component" />
  </transition>
</router-view>
```

```
router.afterEach((to, from) => {
  const toDepth = to.path.split('/').length
  const fromDepth = from.path.split('/').length
  to.meta.transitionName = toDepth < fromDepth ? 'slide-right' : 'slide-left'
})
```

1.6.10 动态路由

有时希望在app正在运行时动态添加路由到router，vue-router提供如下api：

新增路由

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

移除路由

- 通过name删除：`router.removeRoute('about')`
- 通过addRoute()返回的回调：

```
const removeRoute = router.addRoute(routeRecord)
removeRoute() // 如果路由存在移除之
```

- 通过添加一个name冲突的路由：

```
router.addRoute({ path: '/about', name: 'about', component: About })
// 下面操作将移除之前路由并添加新路由，因为他们name冲突
router.addRoute({ path: '/other', name: 'about', component: Other })
```

范例：用户登录之后动态添加权限路由

```
// 权限路由
const authRoutes = [{ path: "/add", component: CourseAdd }];
// 是否添加权限路由
let hasAuth = false;
router.beforeEach((to, from, next) => {
  if (localStorage.getItem("token")) {
    if (hasAuth) { // 添加过直接放行
      next();
    } else { // 动态添加权限路由
      hasAuth = true;
      authRoutes.forEach(route => router.addRoute(route));
      next({ ...to, replace: true });
    }
```

```
  } else {
    next({ path: "/login", query: { redirect: to.path } });
  }
});
```

添加嵌套路由

通过参数1传递父路由name即可

```
router.addRoute('parentRouteName', {...})
```

查找已存在路由

- router.hasRoute(): 判断是否存在某个路由
- router.getRoutes(): 获取所有路由数组

> 更多vue-router4迁移说明