Vue预习课：路由

Vue Router

Vue Router 是 Vue.js 官方的路由管理器。

安装

```
vue add router
```

基础

起步

路由规划、配置，router/index.js

商品列表（home） - 商品管理（about)

路由出口、导航，App.vue

```
<nav>
    <router-link to="/">首页</router-link>
    <router-link to="/about">管理</router-link>
</nav>

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

商品管理，About.vue

```
<template>
  <div>
    <message ref="msgSuccess" class="success">
      <!-- 命名为title插槽内容 -->
      <template v-slot:title="slotProps">
        <strong>{{slotProps.title}}</strong>
      </template>
      <!-- 默认插槽内容 -->
      <template v-slot:default>新增课程成功!</template>
    </message>

    <message ref="msgWarning" class="warning">
      <!-- 命名为title插槽内容 -->
      <template v-slot:title>
        <strong>警告</strong>
      </template>
      <!-- 默认插槽内容 -->
      <template v-slot:default>请输入课程名称!</template>
    </message>

    <cart-add v-model="course" @add-course="addCourse"></cart-add>
    <course-list :courses="courses"></course-list>
  </div>
</template>

<script>
import CartAdd from "@/components/CartAdd.vue";
import CourseList from "@/components/CourseList.vue";
import Message from "@/components/Message.vue";
import { getCourses } from "@/api/course";

export default {
  name: "app",
  data() {
    return {
      course: "",
      courses: []
    };
  },
  components: {
    CartAdd,
    CourseList,
    Message
  },
  async created() {
    // 组件实例已创建，由于未挂载，dom不存在
    const courses = await getCourses();
    this.courses = courses;
  },
  methods: {
    addCourse() {
      if (this.course) {
        // 添加course到数组
        this.courses.push({ name: this.course, price: 8999 });
        this.course = "";
```

```
        // 显示提示信息
        // this.show = true
        this.$refs.msgSuccess.toggle();
      } else {
        // 显示错误信息
        // this.showWarn = true
        this.$refs.msgWarning.toggle();
      }
    }
  }
};
</script>
```

动态路由匹配

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个 `User` 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 `vue-router` 的路由路径中使用“动态路径参数”(dynamic segment) 来达到这个效果：

```
{ path: '/user/:id', component: User }
```

范例：查看课程详情，views/Detail.vue

```
  <div>
    <h2>detail page</h2>
    <p>{{$route.params.name}} ...</p>
  </div>
```

router/index.js

```
{
    path: '/course/:name',
    component: () => import('../views/Detail.vue')
  }
```

列表中的导航，About.vue

```
<router-link :to="`/course/${c.name}`">
    {{ c.name }} - {{ c.price | currency('￥') }}
</router-link>
```

通配符

适合做404页面路由

```
{
  // 会匹配所有路径
  path: '*',
  component: () => import('../views/404.vue')
}
```

嵌套路由

实际生活中的应用界面，通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件，例如：

```
/user/foo/profile                     /user/foo/posts
+------------------+                  +-----------------+
| User             |                  | User            |
| +--------------+ |                  | +-------------+ |
| | Profile      | |  +------------>  | | Posts       | |
| |              | |                  | |             | |
| +--------------+ |                  | +-------------+ |
+------------------+                  +-----------------+
```

范例：嵌套方式显示课程详情

```
<router-link :to="`/about/${c.name}`">
    {{ c.name }} - {{ c.price | currency('￥') }}
</router-link>

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

路由配置

```
{
    path: '/about',
    name: 'about',
    component: () => import(/* webpackChunkName: "about" */
'../views/About.vue'),
    children: [
      {
        path: ':name',
        component: () => import('../views/Detail.vue')
      },
    ]
  }
```

响应路由参数变化，Detail.vue

```
export default {
  watch: {
    $route: {
      handler: () => {
        console.log("$route change");
      },
      immediate: true
    }
  }
};
```

编程导航

借助 router 的实例方法，可编写代码来实现编程式导航

`router.push(location, onComplete?, onAbort?)`

```
// 字符串
router.push('home')

// 对象
router.push({ path: 'home' })

// 命名的路由
router.push({ name: 'user', params: { userId: '123' }})

// 带查询参数，变成 /register?plan=private
router.push({ path: 'register', query: { plan: 'private' }})
```

范例：修改为课程详情跳转为编程导航

```
<div @click="selectedCourse = c;$router.push(`/about/${c.name}`)">
    {{ c.name }} - {{ c.price | currency('￥') }}</div>
```

命名路由

通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。你可以在创建 Router 实例的时候，在 `routes` 配置中给某个路由设置名称。

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

要链接到一个命名路由，可以给 `router-link` 的 `to` 属性传一个对象：

```
<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

调用 `router.push()` 时：

```
router.push({ name: 'user', params: { userId: 123 }})
```

进阶

路由守卫

`vue-router` 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中：全局的, 单个路由独享的, 或者组件级的。

全局守卫

```
router.beforeEach((to, from, next) => {
  // ...
  // 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: Function: 一定要调用该方法来 resolve 这个钩子。
})
```

范例：守卫About.vue

```
router.beforeEach((to, from, next) => {
    if (to.meta.auth) {
        if (window.isLogin) {
            next()
        } else {
            next('/login?redirect='+to.fullPath)
        }
    } else {
        next()
    }
})
```

```
{
    path: '/about',
    meta: {
        auth: true
    }
},
{
    path: '/login',
    component: () => import('../views/Login.vue')
},
```

```
<template>
  <div>
    <button @click="login" v-if="!isLogin">登录</button>
    <button @click="logout" v-else>登出</button>
  </div>
</template>

<script>
  export default {
    methods: {
      login() {
        window.isLogin = true
        this.$router.push(this.$route.query.redirect)
      },
      logout() {
        window.isLogin = false
      }
    },
    computed: {
      isLogin() {
        return window.isLogin
      }
    },
  }
</script>
```

路由独享的守卫

可以路由配置上直接定义 `beforeEnter` 守卫：

```
  {
    path: '/about',
    name: 'about',
    // ...
    beforeEnter(to, from, next) {
      if (to.meta.auth) {
        if (window.isLogin) {
          next()
        } else {
          next('/login?redirect=' + to.fullPath)
        }
      } else {
        next()
```

```
      }
    }
  },
```

组件内守卫

可以在路由组件内直接定义以下路由导航守卫：

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

```
// About.vue
beforeRouteEnter(to, from, next) {
    if (window.isLogin) {
        next();
    } else {
        next("/login?redirect=" + to.fullPath);
    }
}
```

数据获取

路由激活时，获取数据的时机有两个：

- 路由导航前

```
// 组件未渲染，通过给next传递回调访问组件实例
beforeRouteEnter (to, from, next) {
    getPost(to.params.id, post => {
        next(vm => vm.setData(post))
    })
},
// 组件已渲染，可以访问this直接赋值
beforeRouteUpdate (to, from, next) {
    this.post = null
    getPost(to.params.id, post => {
        this.setData(post)
        next()
    })
},
```

- 路由导航后

```
created () {
    this.fetchData()
},
watch: {
    '$route': 'fetchData'
}
```

动态路由

通过router.addRoutes(routes)方式动态添加路由

```
// 全局守卫修改为：要求用户必须登录，否则只能去登录页
router.beforeEach((to, from, next) => {
    if (window.isLogin) {
      if (to.path === '/login') {
        next('/')
      } else {
        next()
      }
    } else {
      if (to.path === '/login') {
        next()
      } else {
        next('/login?redirect=' + to.fullPath)
      }
    }
})
```

```
// Login.vue用户登录成功后动态添加/about
login() {
    window.isLogin = true;

    this.$router.addRoutes([
        {
            path: "/about", //...
        }
    ]);

    const redirect = this.$route.query.redirect || "/";
    this.$router.push(redirect);
}
```

路由组件缓存

利用keepalive做组件缓存，保留组件状态，提高执行效率

范例：缓存about组件

```
<keep-alive include="about">
    <router-view></router-view>
</keep-alive>
```

> 使用include或exclude时要给组件设置name

> 两个特别的生命周期：activated、deactivated

路由懒加载

路由组件的懒加载能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

```
() => import("../views/About.vue")
```

> 把组件按组分块
>
> ```
> () => import(/* webpackChunkName: "group-about" */ "../views/About.vue")
> ```