从零开发企业级中后台项目（基础篇）

这是从零开发企业级中后台项目系列中的基础篇，其中的内容很大一部分来自于官方文档。这里只是提及在中后台搭建中的必知必会，但笔者仍然建议查阅官方文档来入门。

一、Vue 3.0

本章主要介绍 vue 3.0 与 vue 2.0 的区别，以及笔者比较推荐一种编码方式。

1.1 Vue 3.0 带来了什么

Vue 3.0 终于发布正式版了（截止 2020-9-18，2600 多次的提交，628 次 PR）。这意味着我们可以尝试着将其投入到生产环境中。到目前为止（2021-06-21）整个生态不断趋于完善，现在都还不开始学习么，**机会总是留给有准备的人的**。

先来了解一下 Vue 3.0 带来了哪些新的变化：

- 性能提升
- 组合式 API（重中之重）
- 更好的 TS 支持

性能就不说了，在项目中去感受吧。值得一提的是，在使用 Vue 2.0 开发复杂组件的时候，业务逻辑多且复杂的情况下，难以复用。新推出的组合式 API 解决了这一问题。

用过 React 的同学，可能会比较好理解，其实跟 Hook 异曲同工（以组合的方式，复用业务逻辑）。另外就是可以稍微更友好地使用 TS 了（React 推崇纯函数天然支持，也是笔者为什么更偏爱 React 的原因）。

1.2 Vue 3.0 组合式 API

先来看一下什么叫组合 API（Composition API），Vue 3.0 将 Vue 2.0 的选项 API（options API）制作成一个个 hook （钩子）函数，如 watch、computed 等方法，在 Vue 2.0 中是以选项 API 的形式出现，如下：

CSS

```
// options API
export default {
  name: "App",
  watch: {},
  computed: {},
};
```

而 Vue 3.0 新增的 setup 方法，也是以选项的形式出现在抛出的对象中，但是诸如上述代码中的 watch、computed 等属性，都变成 hook 函数的形式，通过 vue 解构出来，在 setup 方法中使用，如下所示：

CoffeeScript

```
// Composition API
import { watch, computed } from "vue";
export default {
  name: "App",
  setup() {
    watch(
      () => {
        // do something
      },
      () => {
        // do something
      }
    );
    const a = computed(() => {
      // do something
    });
  },
};
```

setup 存在的意义，就是为了让你能够使用新增的组合 API。并且这些组合 API 只能在 setup 函数内使用。

下表包含如何在 setup () 内部调用生命周期钩子：

	A	B	C
1	**选项式 API**	Hook inside setup	**描述**
2	beforeCreate	Not needed	
3	created	Not needed	
4	beforeMount	onBeforeMount	组件挂载前
5	mounted	onMounted	组件挂载完成后
6	beforeUpdate	onBeforeUpdate	组件更新前
7	updated	onUpdated	组件更新完成后
8	beforeUnmount	onBeforeUnmount	组件卸载前
9	unmounted	onUnmounted	组件卸载完成后
10	errorCaptured	onErrorCaptured	当捕获来自子孙组件的异常时
11	renderTracked	onRenderTracked	状态跟踪
12	renderTriggered	onRenderTriggered	状态触发
13	activated	onActivated	被 keep-alive 缓存的组件激活时
14	deactivated	onDeactivated	被 keep-alive 缓存的组件停用时

你可以通过在生命周期钩子前面加上 “on” 来访问组件的生命周期钩子。

1.3 Vue 3.0 相关文档

虽然本教程是以分享中后台实战过程为目的，但本着专业的学习精神，文档还是要看的。

便于大家查阅：

- Vue3.0 官方文档
- Composition-API 手册
- Vue 3.0 源码学习
- Vue-Router 4.0 官方文档
- Vuex 4.0
- vue-devtools
- Vite 中文文档
- Vite 源码学习
- Vant 3.0
- Ant Design Vue 2.0
- Element-plus
- Taro

🍉 TODO，如有必要，这里会添加更多关于 vue 3.0 的最佳实践内容。

二、Vue-Router 4.x

Vue-router 是我们在实现单页应用中必不可少的库，推荐大家通过约定 官方文档 来进行深入学习。下面简单介绍一下大概内容，方便快速进入中后台系统的搭建。

2.1 初始化

新增 Vue-Router 依赖：

```Bash
yarn add vue-router@next
```

新建 `src/router/index.js`，并添加如下代码：

JavaScript

```javascript
import { createRouter, createWebHashHistory } from "vue-router";

import Home from "../views/Home.vue";
import Dashboard from "../views/Dashboard.vue";

// 参考: https://next.router.vuejs.org/zh/
const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    {
      path: "/",
      name: "/",
      component: Home,
    },
    {
      path: "/dashboard",
      name: "dashboard",
      component: Dashboard,
    },
  ],
});

export default router;
```

在 `src/main.js` 中添加如下代码：

JavaScript

```javascript
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";

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

通过调用 `app.use(router)`，我们可以在任意组件中以 `this.$router` 的形式访问它，并且以 `this.$route` 的形式访问当前路由：

JavaScript

```
export default {
  computed: {
    username() {
      return this.$route.params.username;
    },
  },
  methods: {
    goToDashboard() {
      this.$router.push("/dashboard");
    },
  },
};
```

> 注意：要在 setup 函数中访问路由，请调用 useRouter 或 useRoute 函数，可在 Composition API 中了解更多信息。

2.2 路由跳转

想要导航到不同的 URL，可以通过下面两种方法。两种方式都会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，会回到之前的 URL。

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

下面是一些样例：

JavaScript

```javascript
// 字符串路径
router.push("/users/eduardo");

// 带有路径的对象
router.push({ path: "/users/eduardo" });

// 命名的路由，并加上参数，让路由建立 url
router.push({ name: "user", params: { username: "eduardo" } });

// 带查询参数，结果是 /register?plan=private
router.push({ path: "/register", query: { plan: "private" } });

// 带 hash，结果是 /about#team
router.push({ path: "/about", hash: "#team" });
```

想了解更多可以参考官方文档

2.3 路由守卫

路由守卫，先来了解一下完整的导航解析流程：

- 导航被触发。
- 在失活的组件里调用 beforeRouteLeave 守卫。
- 调用全局的 beforeEach 守卫。
- 在重用的组件里调用 beforeRouteUpdate 守卫(2.2+)。
- 在路由配置里调用 beforeEnter。
- 解析异步路由组件。
- 在被激活的组件里调用 beforeRouteEnter。
- 调用全局的 beforeResolve 守卫(2.5+)。
- 导航被确认。
- 调用全局的 afterEach 钩子。
- 触发 DOM 更新。
- 调用 beforeRouteEnter 守卫中传给 next 的回调函数，创建好的组件实例会作为回调函数的参数传入。

接着来看一下，如何使用 `router.beforeEach` 注册一个全局前置守卫：

JavaScript

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

router.beforeEach((to, from) => {
  // ...
  // 返回 false 以取消导航
  return false
})
```

每个守卫方法接收两个参数：

- to: 即将要进入的目标
- from: 当前导航正要离开的路由

to、from 的数据格式：routelocationnormalized

可以返回的值如下：

- false: 取消当前的导航。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。
- 一个路由地址: 通过一个路由地址跳转到一个不同的地址，就像你调用 router.push() 一样，你可以设置诸如 replace: true 或 name: 'home' 之类的配置。当前的导航被中断，然后进行一个新的导航，就和 from 一样。

可选的第三个参数 next：

JavaScript

```
router.beforeEach((to, from, next) => {
  if (to.name !== "Login" && !isAuthenticated) next({ name: "Login" });
  else next();
});
```

注意：确保 next 在任何给定的导航守卫中都被严格调用一次。

2.4 路由元信息

有时，你可能希望将任意信息附加到路由上，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：

YAML

```yaml
const routes = [
  {
    path: '/posts',
    component: PostsLayout,
    children: [
      {
        path: 'new',
        component: PostsNew,
        // 只有经过身份验证的用户才能创建帖子
        meta: { requiresAuth: true }
      },
      {
        path: ':id',
        component: PostsDetail
        // 任何人都可以阅读文章
        meta: { requiresAuth: false }
      }
    ]
  }
]
```

可通过下面的方法来访问 `meta` 字段：

JavaScript

```javascript
router.beforeEach((to, from) => {
  // 而不是去检查每条路由记录
  // to.matched.some(record => record.meta.requiresAuth)
  if (to.meta.requiresAuth && !auth.isLoggedIn()) {
    // 此路由需要授权，请检查是否已登录
    // 如果没有，则重定向到登录页面
    return {
      path: "/login",
      // 保存我们所在的位置，以便以后再来
      query: { redirect: to.fullPath },
    };
  }
});
```

三、vuex 4.x

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。

可通过 Vue 的官方调试工具 devtools extension 来获得零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

3.1 初始化

安装：

Bash

```
yarn add vuex@next
```

每一个 Vuex 应用的核心就是 store（仓库），它和单纯的全局对象不同的是：

- Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
- 你不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

示例：

JavaScript

```javascript
import { createStore } from "vuex";

// 创建一个新的 store 实例
const store = createStore({
  state() {
    return {
      count: 0,
    };
  },
  mutations: {
    increment(state) {
      state.count++;
    },
  },
});

// 通过 store.commit 方法触发状态变更
store.commit("increment");
```

在 Vue 组件中， 可以通过 this.$store 访问 store 实例：

JavaScript

```javascript
methods: {
  increment() {
    this.$store.commit('increment')
    console.log(this.$store.state.count)
  }
}
```

由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。为了解决这个问题，Vuex 允许我们将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割：

JavaScript

```javascript
const moduleA = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... }
}

const store = createStore({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> moduleA 的状态
store.state.b // -> moduleB 的状态
```

3.2 项目结构

Vuex 并不限制你的代码结构。但是，它规定了一些需要遵守的规则：

- 应用层级的状态应该集中到单个 store 对象中。
- 提交 mutation 是更改状态的唯一方法，并且这个过程是同步的。
- 异步逻辑都应该封装到 action 里面。

结构示例：

Bash

```
├──── index.html
├──── main.js
├──── api
│   └──── ... # 抽取出 API 请求
├──── components
│   ├──── App.vue
│   └──── ...
└──── store
├──── index.js # 我们组装模块并导出 store 的地方
├──── actions.js # 根级别的 action
├──── mutations.js # 根级别的 mutation
└──── modules
├──── cart.js # 购物车模块
└──── products.js # 产品模块
```

3.3 常用API

一些有用的 API：

- mapState
- mapGetters
- mapMutations
- mapActions

组合式 API：

可以通过调用 useStore 函数，来在 setup 钩子函数中访问 store。下面是简单的示例（需要更多示例参阅）：

JavaScript

```js
import { computed } from 'vue'
import { useStore } from "vuex";

export default {
  setup() {
    const store = useStore();

    return {
      // 在 computed 函数中访问 state
      count: computed(() => store.state.count),

      // 在 computed 函数中访问 getter
      double: computed(() => store.getters.double)

      // 使用 mutation
      increment: () => store.commit("increment"),

      // 使用 action
      asyncIncrement: () => store.dispatch("asyncIncrement"),
    };
  },
};
```

vuex 内容不属于本次教程的重点内容，想继续深入学习的同学可以参阅官网文档，这里就不再重复搬运。

四、其他配置项

通过配置 `jsconfig.json` 来提升 vue 的开发体验：

JSON

```json
{
  "compilerOptions": {
    "target": "es6",
    "baseUrl": ".",
    "paths": {
      "@/*": ["src/*"] // 这样项目中，通过 @ 配置的路径就能得到 vscode 的引用跳转，否则编辑器不认识 @
    }
```