

vue3全家桶+Ts



YaYa LV.3

2021年10月13日 08:50 · 阅读 466

关注

vue3全家桶+Ts

创建项目

让我们来下载一个vue3的项目

```
→ ~ vue create talk

Vue CLI v4.5.10

New version available 4.5.10 → 4.5.13
Run yarn global add @vue/cli to update!

? Please pick a preset:
  composition-api ([Vue 2] router, vuex, less, babel, eslint)
  vue3 preset ([Vue 3] less, babel, typescript, router, vuex, eslint)
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
? Check the features needed for your project:
  ☒ Choose Vue version
  ☒ Babel
  ☒ TypeScript
  ☐ Progressive Web App (PWA) Support
  ☒ Router
  ☒ Vuex
> ☒ CSS Pre-processors
  ☒ Linter / Formatter
  ☐ Unit Testing
  ☐ E2E Testing
? Choose a version of Vue.js that you want to start the project with
  2.x
```

```

? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) █
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default):
  Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
> Less
  Stylus
? Pick a linter / formatter config:
  ESLint with error prevention only
  ESLint + Airbnb config
> ESLint + Standard config
  ESLint + Prettier
  TSLint (deprecated)
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>● Lint on save
  ○ Lint and fix on commit
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json

```

@稀土掘金技术社区

基础语法

定义data

- script标签上lang="ts"
- 定义一个类型 `type` 或者接口 `interface` 来约束 `data`
- 可以使用 `ref` 或者 `toRefs` 来定义响应式数据
 - 使用 `ref` 在 `setup` 读取的时候需要获取 `xxx.value` ,但在 `template` 中不需要
 - 使用 `reactive` 时, 可以用 `toRefs` 解构导出, 在 `template` 就可以直接使用了

typescript 复制代码

```

<script lang="ts">
import { defineComponent, reactive, ref, toRefs } from 'vue';

type Todo = {
  id: number,
  name: string,
  completed: boolean
}

```

```
export default defineComponent({
  const data = reactive({
    todoList: [] as Todo[]
  })
  const count = ref(0);
  console.log(count.value)
  return {
    ...toRefs(data)
  }
})
</script>
```

基础语法

定义data

- script标签上lang="ts"
- 定义一个类型 `type` 或者接口 `interface` 来约束 `data`
- 可以使用 `ref` 或者 `toRefs` 来定义响应式数据
 - 使用 `ref` 在 `setup` 读取的时候需要获取 `xxx.value` ,但在 `template` 中不需要
 - 使用 `reactive` 时, 可以用 `toRefs` 解构导出, 在 `template` 就可以直接使用了

typescript 复制代码

```
<script lang="ts">
import { defineComponent, reactive, ref, toRefs } from 'vue';

type Todo = {
  id: number,
  name: string,
  completed: boolean
}

export default defineComponent({
  const data = reactive({
    todoList: [] as Todo[]
  })
  const count = ref(0);
  console.log(count.value)
  return {
```

```
  })  
</script>
```

定义props

props 需要使用 `PropType` 泛型来约束。

typescript 复制代码

```
<script lang="ts">  
import { defineComponent, PropType } from 'vue';  
  
interface UserInfo = {  
  id: number,  
  name: string,  
  age: number  
}  
  
export default defineComponent({  
  props: {  
    userInfo: {  
      type: Object as PropType<UserInfo>, // 泛型类型  
      required: true  
    }  
  },  
})  
</script>
```

定义methods

typescript 复制代码

```
<script lang="ts">  
import { defineComponent, reactive, ref, toRefs } from 'vue';  
  
type Todo = {  
  id: number,  
  name: string,  
  completed: boolean  
}  
  
export default defineComponent({  
  const data = reactive({  
    todoList: [] as Todo[]  
  })  
  // 约束输入和输出类型  
})  
</script>
```

```
      id: this.items.length + 1,
      name,
      completed: false
    });
  }
  const addTodo = (todo: Todo): void => {
    data.todoList.push(todo)
  }
  return {
    ...toRefs(data),
    newTodo,
    addTodo
  }
})
</script>
```

vue-router

- `createRouter` 创建 `router` 实例
- `router` 的模式分为：
 - `createWebHistory` -- history模式
 - `createWebHashHistory` -- hash模式
- `routes` 的约束类型是 `RouteRecordRaw`

javascript 复制代码

```
import { createRouter, createWebHistory, RouteRecordRaw } from 'vue-router';
import Home from '../views/Home.vue';
const routes: Array< RouteRecordRaw > = [
  {
    path: '/',
    name: 'Home',
    component: Home,
  },
  {
    path: '/about',
    name: 'About',
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')
  }
];
```

```
routes
});

export default router;
```

扩展路由额外属性

在实际项目开发中，常常会遇到这么一个场景，某一个路由是不需要渲染到侧边栏导航上的，此时我们可以给该路由添加一个hidden属性来实现。

在ts的强类型约束下，添加额外属性就会报错，那么我们就需要扩展 `RouteRecordRaw` 类型。

typescript 复制代码

```
// 联合类型
type RouteConfig = RouteRecordRaw & {hidden?: boolean}; //hidden 是可选属性
const routes: Array<RouteConfig> = [
  {
    path: '/',
    name: 'Home',
    component: Home,
    hidden: true,
    meta: {
      permission: true,
      icon: ''
    }
  }
];
```

在setup中使用

需要导入 `useRouter` 创建一个 `router` 实例。

xml 复制代码

```
<script lang="ts">
import { useRouter } from 'vue-router';
import { defineComponent } from 'vue';
export default defineComponent({
  setup () {
    const router = useRouter();
    goRoute(path) {
      router.push({path})
    }
  }
});
```

vuex

使用this.\$store

```
import { createStore } from 'vuex';

export type State = {
  count: number
}

export default createStore({
  state: {
    count: 0
  }
});
```

typescript 复制代码

需要创建一个声明文件 `vuex.d.ts`

```
// vuex.d.ts

import { ComponentCustomProperties } from 'vue';
import { Store } from 'vuex';
import { State } from './store'

declare module '@vue/runtime-core' {
  interface ComponentCustomProperties {
    $store: Store<State>
  }
}
```

typescript 复制代码

在setup中使用

1. 定义InjectionKey
2. 在安装插件时传入key
3. 在使用useStore时传入

```
import { InjectionKey } from 'vue';
import { createStore, Store } from 'vuex';

export type State = {
  count: number
}
```

typescript 复制代码

```
// 创建一个injectionKey
export const key: InjectionKey<Store<State>> = Symbol('key');
```

javascript 复制代码

```
// main.ts
import store, { key } from './store';
app.use(store, key);
```

xml 复制代码

```
<script lang="ts">
import { useStore } from 'vuex';
import { key } from '@store';
export default defineComponent({
  setup () {
    const store = useStore(key);
    const count = computed(() => store.state.count);
    return {
      count
    }
  }
})
</script>
```

模块

新增一个 `todo` 模块。导入的模块，需要是一个 `vuex` 中的interface `Module` 的对象,接收两个泛型约束，第一个是**该模块类型**，第二个是**根模块类型**。

typescript 复制代码

```
// modules/todo.ts
import { Module } from 'vuex';
import { State } from '../index.ts';

type Todo = {
  id: number,
  name: string,
  completed: boolean
}

const initialState = {
  todos: [] as Todo[]
};

export type TodoState = typeof initialState;
```



```
state: initialState,
mutations: {
  addTodo (state, payload: Todo) {
    state.todos.push(payload);
  }
}
} as Module<TodoState, State>; //Module<S, R> S 该模块类型 R根模块类型
```

typescript 复制代码

```
// index.ts
export type State = {
  count: number,
  todo?: TodoState // 这里必须是可选，不然state会报错
}

export default createStore({
  state: {
    count: 0
  }
  modules: {
    todo
  }
});
```

使用:

scss 复制代码

```
setup () {
  console.log(store.state.todo?.todos);
}
```

elementPlus

csharp 复制代码

```
yarn add element-plus
```

完整引入

javascript 复制代码

```
import { createApp } from 'vue'
import ElementPlus from 'element-plus';import 'element-plus/lib/theme-chalk/index.css';import App fr
import 'dayjs/locale/zh-cn'
import locale from 'element-plus/lib/locale/lang/zh-cn'
```

```
app.use(ElementPlus, { size: 'small', zIndex: 3000, locale })
app.mount('#app')
```

按需加载

需要安装 **babel-plugin-component** 插件:

```
yarn add babel-plugin-component -D
```

scss 复制代码

```
// babel.config.js
plugins: [
  [
    'component',
    {
      libraryName: 'element-plus',
      styleLibraryName: 'theme-chalk'
    }
  ]
]
```

```
import 'element-plus/lib/theme-chalk/index.css';
import 'dayjs/locale/zh-cn';
import locale from 'element-plus/lib/locale';
import lang from 'element-plus/lib/locale/lang/zh-cn';
import {
  ElAside,
  ElButton,
  ElButtonGroup,
} from 'element-plus';
```

ini 复制代码

```
const components: any[] = [
  ElAside,
  ElButton,
  ElButtonGroup,
];

const plugins: any[] = [
  ElLoading,
  ElMessage,
  ElMessageBox,
  ElNotification
]
```

```

const element = (app: any):any => {
  // 国际化
  locale.use(lang);
  // 全局配置
  app.config.globalProperties.$ELEMENT = { size: 'small' };

  components.forEach(component => {
    app.component(component.name, component);
  });

  plugins.forEach(plugin => {
    app.use(plugin);
  });
};

export default element;

```

javascript 复制代码

```

// main.ts
import element from './plugin/element'

const app = createApp(App);
element(app);

```

axios

axios的安装使用和vue2上没有什么大的区别，如果需要做一些扩展属性，还是需要声明一个新的类型。

```

type Config = AxiosRequestConfig & {successNotice? : boolean, errorNotice? : boolean}

```

java 复制代码

```

import axios, { AxiosResponse, AxiosRequestConfig } from 'axios';
import { ElMessage } from 'element-plus';
const instance = axios.create({
  baseURL: process.env.VUE_APP_API_BASE_URL || '',
  timeout: 120 * 1000,
  withCredentials: true
});

// 错误处理
const err = (error) => {
  if (error.message.includes('timeout')) {

```

php 复制代码

```
        type: 'error'
      });
    }
  }
  if (error.response) {
    const data = error.response.data;
    if (error.response.status === 403) {
      ElMessage({
        message: 'Forbidden',
        type: 'error'
      });
    }
  }
  if (error.response.status === 401) {
    ElMessage({
      message: 'Unauthorized',
      type: 'error'
    });
  }
}
return Promise.reject(error);
};
```

```
type Config = AxiosRequestConfig & {successNotice? : boolean, errorNotice? : boolean}
```

```
// 请求拦截
```

```
instance.interceptors.request.use((config: Config) => {
  config.headers['Access-Token'] = localStorage.getItem('token') || '';
  return config;
}, err);
```

```
// 响应拦截
```

```
instance.interceptors.response.use((response: AxiosResponse) => {
  const config: Config = response.config;

  const code = Number(response.data.status);
  if (code === 200) {
    if (config && config.successNotice) {
      ElMessage({
        message: response.data.msg,
        type: 'success'
      });
    }
    return response.data;
  } else {
    let errCode = [402, 403];
    if (errCode.includes(response.data.code)) {
      ElMessage({
        message: response.data.msg,
```

```
    }  
  }  
}, err);  
  
export default instance;
```

setup script

官方提供了一个**实验性**的写法，直接在 `script` 里面写 `setup` 的内容，即： `setup script` 。

之前我们写组件是这样的：

xml 复制代码

```
<template>  
  <div>  
    {{count}}  
    <ImgReview></ImgReview >  
  </div>  
</template>  
<script lang="ts">  
import { ref, defineComponent } from "vue";  
import ImgReview from "../components/ImgReview.vue";  
  
export default defineComponent({  
  components: {  
    ImgReview,  
  },  
  setup() {  
    const count = ref(0);  
    return { count };  
  }  
});  
</script>
```

启用 `setup script` 后：在 `script` 上加上 `setup`

xml 复制代码

```
<template>  
  <div>  
    {{count}}  
    <ImgReview></ImgReview>  
  </div>  
</template>  
<script lang="ts" setup>
```

```
const count = ref(0);  
</script>
```

是不是看起来简洁了很多，组件直接导入就行了，不用注册组件，数据定义了就可以用。其实我们可以简单的理解为 `script` 包括的内容就是 `setup` 中的，并做了 `return`。

导出方法

typescript 复制代码

```
<script lang="ts" setup>  
const handleClick = (type: string) => {  
  console.log(type);  
}  
</script>
```

定义props

使用 `props` 需要用到 `defineProps` 来定义，具体用法跟之前的 `props` 写法类似：

基础用法

xml 复制代码

```
<script lang="ts" setup>  
import { defineProps } from "vue";  
const props = defineProps(['userInfo', 'gameId']);  
</script>
```

构造函数进行检查 给props定义类型：

php 复制代码

```
const props = defineProps({  
  gameId: Number,  
  userInfo: {  
    type: Object,  
    required: true  
  }  
});
```

使用类型注解进行检查

```
defineProps<{
  name: string
  phoneNumber: number
  userInfo: object
  tags: string[]
}>()
```

可以先定义好类型：

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

```
defineProps<{
  name: string
  userInfo: UserInfo
}>()
```

defineEmit

```
<script lang="ts" setup>
import { defineEmit } from 'vue';

// expects emits options
const emit = defineEmit(['kk', 'up']);
const handleClick = () => {
  emit('kk', '点了我');
};
</script>
```

```
<Comp @kk="handleClick"/>

<script lang="ts" setup>
const handleClick = (data) => {
  console.log(data)
}
</script>
```

在标准组件写法里，setup 函数默认支持两个入参：

参数	类型	含义
props	object	由父组件传递下来的数据
context	object	组件的执行上下文

在setup script 中使用useContext获取上下文：

xml 复制代码

```
<script lang="ts" setup>
  import { useContext } from 'vue'
  const { slots, attrs } = useContext();
</script>
```

获取到的 slots , attrs 跟 setup 里面的是一样的。

分类： 前端 标签： 前端 Vue.js

评论

输入评论（Enter换行，Ctrl + Enter发送）

相关推荐

悠哉wu 2月前 前端 Vue.js

由于不好用，就用Vue3+Ts做了个视频网站

1.6w 177 69

清风夜半 11月前 前端 CSS

Vue+Echarts企业级大屏项目活配方案

日拱一寸 1年前 前端 Vite

体验vite + vue3 + ts搭建项目的全过程

1.7w 128 26

我要充满正能量 5月前 前端

Vue3拒绝写return, 用setup语法糖,让写Vue3更畅快

3.1w 498 126

shuan 2年前 数据可视化

Vue全家桶+Echarts数据可视化实践

3436 9 8

天明夜尽 1年前 Vue.js Element

Vue3 全家桶 + Element Plus + Vite + TypeScript + Eslint 项目配置最佳实践

1.9w 220 28

Web前端严选 1年前 Vue.js

Vue3+TS, 写一个逼格满满的项目

3.8w 256 48

小兮兮 3月前 前端 Vue.js

25个 Vue 技巧,学了这么久才知道还能这么用

5.6w 658 68

清夜 2月前 前端 JavaScript Vue.js

一定要优雅, 高端前端程序员都应该具备的基本素养

7.5w 662 230

杨村长 1年前 Vue.js 前端

还在迟疑是否上ts? 先上车再说! vue3+ts开发初体验

1.2w 185 29

UU 1年前 Vue.js

如何搭建一个完整的Vue3.0 + ts 的项目

木鱼123 1年前 ESLint

vite+vue3+ts+eslint+prettier+stylelint+husky+jest搭建项目

5894 40 3

一起去看星星吧 3月前 前端

ts保姆级教程，别再说你不会ts了

6.5w 1614 159

前端胖头鱼 4月前 前端 JavaScript Vue.js

11+ chrome高级调试技巧，学会效率直接提升666%

7.1w 2720 177

mapengfei 5月前 前端 Vue.js

大菠萝？Pinia已经来了，再不学你就outs了

9.0w 1264 219

liyoro 1年前 Vue.js ECharts

vue项目中封装echarts的比较优雅的方式

1.2w 210 38

webmote33 1年前 Vue.js 前端

关于微前端(阿里QianKun)的那点事——上线一个“微前端”逼走了2位90后

5.0w 591 544

张星宇 1年前 前端

vue3+g6 踩坑笔记2 (.d.ts 类型声明文件)

1040 1 评论

Jokcy 1年前 Vue.js JavaScript

为什么我推荐使用JSX开发Vue3

3.6w 431 327



7



评论



收藏

