typescript+vue项目最佳实践

知识点

- vue+ts开发环境准备
- ts核心语言特性
- 组件编写
- 路由声明
- 全局状态管理
- 数据请求
- 第三方模块

开发环境准备

vue-cli：4.x

新建一个基于ts的vue项目

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Linter
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript for auto-detected polyfills? Yes
? Pick a linter / formatter config: Basic
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert sele
n)Lint on save
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) n
```

在已存在项目中安装typescript

```
vue add @vue/typescript
```

```
📦  Installing @vue/cli-plugin-typescript...

?  Your connection to the default yarn registry seems to be slow.
   Use https://registry.npm.taobao.org for faster installation? Yes
+ @vue/cli-plugin-typescript@4.2.3
added 26 packages from 30 contributors in 6.016s

19 packages are looking for funding
  run `npm fund` for details

✔  Successfully installed plugin: @vue/cli-plugin-typescript

? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, t
ranspiling JSX)? Yes
? Convert all .js files to .ts? No
? Allow .js files to be compiled? No
```

组件编写

ts编写组件有三种常见方式：`Vue.extend()`、`@Component`和`tsx`，以下案例以装饰器为主演示。

修改HelloWorld.vue组件

```
import { Component, Vue } from 'vue-property-decorator';

@Component
export default class HelloWorld extends Vue {
  msg = 'ts+vue实战'
}
```

组件的选项依然可以使用

```
@Component({
  componxents:{
    // ...
  }
})
export default class HelloWorld extends Vue {}
```

核心语言特性

类型注解、泛型、装饰器是最重要的三个语言特性

类型注解

使用类型注解引入强类型系统，保证程序健壮性。

使用类型注解

```
export default class HelloWorld extends Vue {
  msg: string = 'ts+vue实战'; // 类型注解
}
```

类型数组：数组内元素类型约束

```
heros: string[] = ['卤蛋', '后羿']
```

```
<div v-for="hero in heros" :key="hero">
  {{ hero }}
</div>
```

类型别名：使用类型别名定义自己的类型

```
// @/types/index.ts
export type Hero = {
  id: number,
  name: string,
  classify: string
}
```

```
import { Hero } from "@/types";

//...
heros: Hero[] = [
  { id: 1, name: "卤蛋", classify: "射手" },
  { id: 2, name: "后羿", classify: "射手" }
];
```

```
<div v-for="hero in heros" :key="hero.id">{{ hero.name }}</div>
```

函数中的类型

函数的形参和返回值也可以使用类型注解

```
onClick(hero: Hero): void {
  alert('我选'+hero.name+', 我走发育路')
}
```

```
<div @click="onClick(hero)">{{ hero.name }}</div>
```

事件的类型

```
addHero(e: KeyboardEvent) {
  const inp = e.target as HTMLInputElement;
  this.heros.push({
    id: this.heros.length + 1, name: inp.value, classify: this.classify });
}
```

```
<input type="text" @keydown.enter="addHero" />
```

生命周期函数

```
heros: Hero[] = [];

// 生命周期钩子名字对应即可
created() {
  this.heros = [
    { id: 1, name: "卤蛋", classify: "射手" },
    { id: 2, name: "后羿", classify: "射手" }
  ];
}
```

存取器：会作为计算属性对待

```
get total() {
  return this.heros.length
}
```

```
<p>共{{total}}个可选英雄</p>
```

泛型

泛型使我们定义的各种类型更加通用。

```ts
// @/types/index.ts
export interface Result<T> {
  ok: 0 | 1;
  data: T;
}
```

```ts
// @/api/hero.ts
import { Result } from '@/types'

export function getResult<T>(): Result<T> {
  const data: any = [
    { id: 1, name: "卤蛋", classify: "射手" },
    { id: 2, name: "后羿", classify: "射手" }
  ]
  return { ok: 1, data};
}
```

```ts
// HelloWorld.vue
heros: Hero[] = [];

created() {
  this.heros = getResult<Hero[]>().data;
}
```

泛型还能嵌套

```ts
// @/api/hero.ts
export function getResult<T>(): Promise<Result<T>> {
  //...
  return Promise.resolve<Result<T>>({ ok: 1, data});
}
```

```ts
// HelloWorld.vue
 async created() {
   this.heros = (await getResult<Hero[]>()).data;
 }
```

真正的网络请求

```
import axios from 'axios'

export function getResult<T>() {
  return axios.get<T>('/api/heros')
}
```

装饰器

装饰器用于扩展装饰的目标，常用于定义组件、属性和侦测器等

定义属性

```
import { Prop } from "vue-property-decorator";

@Component({
  components: {}
})
export default class HelloWorld extends Vue {
  @Prop({type: String, required: true})
  msg!: string;
}
```

派发事件

```
// HelloWorld.vue
@Emit()
addHero(e: KeyboardEvent) {
  // 返回事件载荷
  return inp.value
}
```

```
<!-- App.vue -->
<HelloWorld @add-hero="addHero"/>
```

路由声明

依靠各种强类型加持，编写路由变得更加轻松、准确。

```
// router/index
import VueRouter, { RouteConfig } from 'vue-router'

routes: RouteConfig[] = [
  //...
]

router.beforeEach((to, from, next)=>{
  console.log(to.path);
  next()
})
```

全局状态管理

全局状态管理推荐使用vuex-module-decorators，模块化开发轻而易举，还能很好的利用ts类型推断。

```
// store/index.ts
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({})
```

```
// store/user.ts
import { VuexModule, Module, Mutation } from 'vuex-module-decorators'
import store from './index'

@Module({dynamic: true, name: 'user', store})
class User extends VuexModule {
  username = 'tom'

  @Mutation
  setUsername(name: string) {
    this.username = name;
  }
}

export default User
```

使用：

```
import { getModule } from "vuex-module-decorators";
import User from "../store/user";
```

```
const user = getModule(User);

@Component({
  components: {}
})
export default class HelloWorld extends Vue {
  get username() {
    return user.username;
  }
  async created() {
    user.setUsername('Jerry')
  }
}
```

模块扩展

有些属性在某种类型上不存在时需要做扩展，否则会有错误提示。

以axios挂载为例

```
// HelloWorld.vue
created() {
  // 如果不做扩展，编译器无法识别$http
  this.$http.get<Hero[]>('/api/heros').then(res=> {
    this.heros = res.data
  })
}
```

```
// shims-vue.d.ts
import { AxiosInstance } from 'axios'

declare module 'vue/types/vue' {
  interface Vue {
    $http: AxiosInstance
  }
}
```

第三方模块

UI库等第三方库引入时，常用库都是有声明文件的，如果没有就手动安装：`npm i -D @types/xxx`

以moment为例

```
import moment from 'moment'

@Component
export default class App extends Vue {
  msg = moment().format('YYYY-MM-DD')
}
```

以element-ui为例

```
this.$notify.success("新增成功");
```