TypeScript

资源

1. TypeScript参考
2. vue中的TypeScript

知识点

准备工作

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
```

类型注解和编译时类型检查

类型注解：变量后面通过冒号+类型来做类型注解

```
// test.ts
let title1: string; // 类型注解
title1 = "开课吧"; // 正确
title1 = 4; // 错误

let title2 = "xx"; // 类型推论
title2 = 2;// 错误

//数组类型
```

```
let names: string[];
names = ['Tom'];//或Array<string>

//任意类型
let foo:any;
foo = 'xx'
foo = 3

//any类型也可用于数组
let list: any[];
list = [1, true, "free"];
list[1] = 100;

//函数中使用类型
function greeting(person: string): string {
  return 'Hello, ' + person;
}
//void类型，常用于没有返回值的函数
function warnUser(): void { alert("This is my warning message"); }
```

函数

必填参：参数一旦声明，就要求传递，且类型需符合

```
function greeting(person: string): string {
  return "Hello, " + person;
}
greeting('tom')
```

可选参数：参数名后面加上问号，变成可选参数

```
function greeting(person: string, msg?: string): string {
  return "Hello, " + person;
}
```

参数默认值

```
function greeting(person: string, msg = ''): string {
  return "Hello, " + person;
}
```

函数重载

```
// 声明1
function info(a: {name: string}): string;
// 声明2
function info(a: string): {name: string};
// 实现
function info(a: {name: string} | string): {name: string} | string {
  if (typeof a === "object") {
    return a.name;
  } else {
    return { name: a };
```

```
  }
}
console.log(info({ name: "tom" }));
console.log(info("tom"));
```

类

class的特性

ts中的类和es6中大体相同，这里重点关注ts带来的特性

```
class MyComp {
  private _foo: string; // 私有属性，不能在类的外部访问
  protected bar: string;// 保护属性，可以在子类中访问

  // 构造函数参数加修饰符，能够定义为成员属性
  constructor(public tua = "tua") {}

  // 方法也有修饰符
  private someMethod() {}

  // 存取器：属性方式访问，可添加额外逻辑，控制读写性
  get foo() { return this._foo }
  set foo(val) { this._foo = val }
}
```

接口

接口仅约束结构，不要求实现，使用更简单

```
interface Person {
  firstName: string;
  lastName: string;
}
function greeting(person: Person) {
  return 'Hello, ' + person.firstName + ' ' + person.lastName;
}
const user = {firstName: 'Jane', lastName: 'User'};
console.log(user);
console.log(greeting(user));
```

泛型

泛型（Generics）是指在定义函数、接口或类的时候，不预先指定具体的类型，而在使用的时候再指定类型的一种特性。以此增加代码通用性。

```
// 不用泛型
// interface Result {
//   ok: 0 | 1;
//   data: Feature[];
```

```
// }

// 使用泛型
interface Result<T> {
  ok: 0 | 1;
  data: T;
}

// 泛型方法
function getData<T>(): Result<T> {
  const data: any = [
    { id: 1, name: "类型注解", version: "2.0" },
    { id: 2, name: "编译型语言", version: "1.0" }
  ];
  return { ok: 1, data }
}
```

装饰器

装饰器用于扩展类或者它的属性和方法。@xxx就是装饰器的写法

组件声明：@Component

典型应用是组件装饰器@Component

```
@Component
export default class Hello extends Vue {}
```

属性声明：@Prop

除了在@Component中声明，还可以采用@Prop的方式声明组件属性

```
export default class HelloWorld extends Vue {
  // Props()参数是为vue提供属性选项
  // !称为明确赋值断言，它是提供给ts的
  @Prop({type: String, required: true})
  private msg!: string;
}
```

事件处理：@Emit

新增特性时派发事件通知，Hello.vue

```
@Emit()
private addFeature(event: any) {// 若没有返回值形参将作为事件参数
    const feature = { name: event.target.value, id: this.features.length + 1 };
    this.features.push(feature);
    event.target.value = "";
    return feature;// 若有返回值则返回值作为事件参数
}
```

变更监测：@Watch

```
@Watch('msg')
onRouteChange(val:string, oldVal:any){
    console.log(val, oldVal);
}
```

vuex使用：vuex-class

`vuex-class`为vue-class-component提供Vuex状态绑定帮助方法。

安装依赖

```
npm i vuex-class -S
```

定义状态，store.ts

```
import Vuex from "vuex";
import Vue from "vue";

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    features: [
      { id: 1, name: "类型", version: "1.0" },
      { id: 2, name: "编译型语言", version: "1.0" },
    ],
  },
  mutations: {
    addFeatureMutation(state: any, featureName) {
      state.features.push({ id: state.features.length + 1, name: featureName });
    },
  },
  actions: {
    addFeatureAction({ commit }, featureName) {
      commit("addFeatureMutation", featureName);
    },
  },
});
```

使用，Hello.vue

```
import { State, Action, Mutation } from "vuex-class";

@Component
export default class Feature extends Vue {
  // 状态、动作、变更映射
  @State features!: string[];
  @Action addFeatureAction: any;
  @Mutation addFeatureMutation: any;

  private addFeature(event) {
    console.log(event);
    // this.features.push(event.target.value);
    this.addFeatureAction(event.target.value);
    // this.addFeaturMutation(event.target.value);
    event.target.value = "";
  }
}
```

装饰器原理

类装饰器

```
//类装饰器表达式会在运行时当作函数被调用，类的构造函数作为其唯一的参数。
function log(target: Function) {
    // target是构造函数
    console.log(target === Foo); // true
    target.prototype.log = function() {
      console.log(this.bar);
    }
    // 如果类装饰器返回一个值，它会使用提供的构造函数来替换类的声明。
}

@log
class Foo {
  bar = 'bar'
}

const foo = new Foo();
// @ts-ignore
foo.log();
```

方法装饰器

```
function dong(target: any, name: string, descriptor: any) {
    // 这里通过修改descriptor.value扩展了bar方法
    const baz = descriptor.value;
    descriptor.value = function(val: string) {
        console.log('dong~~');
        baz.call(this, val);
    }
    return descriptor
```

```
}

class Foo {
  @dong
  setBar(val: string) {
    this.bar = val
  }
}

foo.setBar('lalala')
```

属性装饰器

```
// 属性装饰器
function mua(target, name) {
    target[name] = 'mua~~~'
}
class Foo {
  @mua ns!:string;
}

console.log(foo.ns);
```

稍微改造一下使其可以接收参数

```
function mua(param:string) {
    return function (target, name) {
        target[name] = param
    }
}
```

实战一下Component，新建Decor.vue

```
<template>
  <div>{{msg}}</div>
</template>

<script lang='ts'>
import { Vue } from "vue-property-decorator";

function Component(options: any) {
  return function(target: any) {
    return Vue.extend(options);
  };
}

@Component({
  props: {
    msg: {
      type: String,
      default: ""
```

```
    }
  }
})
export default class Decor extends Vue {}
</script>
```

显然options中的选项都可以从Decor定义中找到，去源码中找答案吧~

作业

1. 把手头的小项目改造为ts编写
2. 探究vue-property-decorator中各装饰器实现原理，能造个轮子更佳