初探 TypeScript

物流组凌晨轩
chenxuan.ling@ele.me

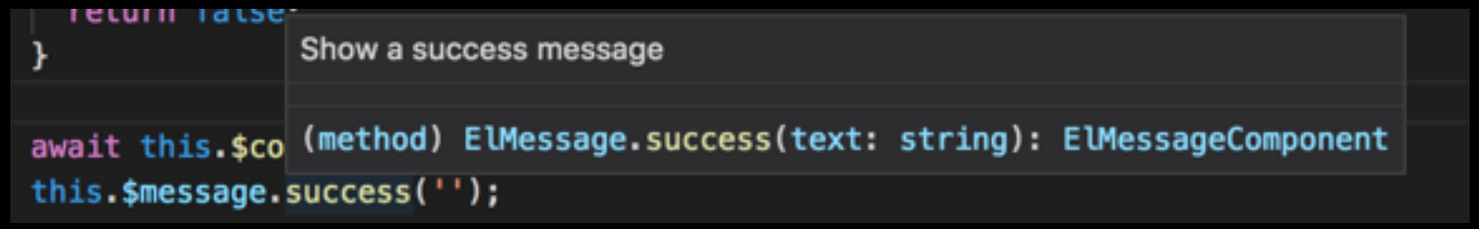
简介

- TypeScript 是微软开发维护的一种编程语言。
- 它是 JavaScript 的超集，并添加了类型机制等特性。

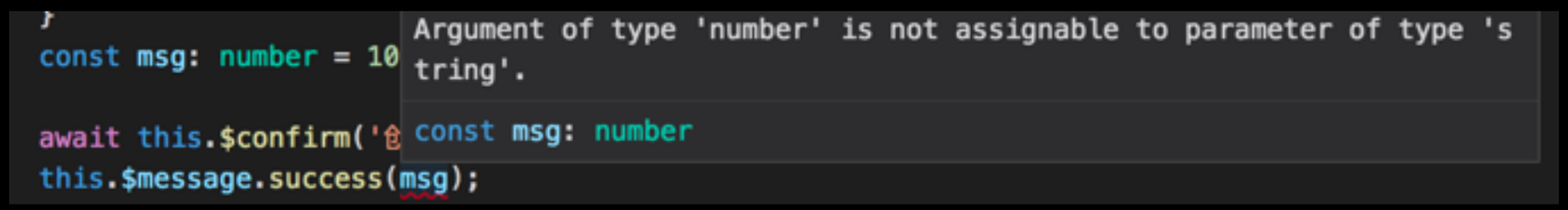
为什么使用TS

- JS是弱类型语言，容易出现bug，不易维护。
- TS的类型机制可以提升代码的可维护性，提升团队开发的效率。
- IDE和编辑器有更好的自动补全和类型提示。

vscode的API提示



vscode的错误提示



基础类型

- 布尔值 `let isDone: boolean = false;`
- 数字 `let decLiteral: number = 6;`
- 字符串 `let name: string = "bob";`
- 数组 `let list: number[] = [1, 2, 3];`
- 元组 `let x: [string, number];`

- 枚举

```
enum Color {Red, Green, Blue};
let c: Color = Color.Green;
```

- Any
- Void
- Null 和 Undefined

类型断言

```
let someValue: any = "this is a string”;
let strLength: number = (<string>someValue).length;
```

另外一种写法：

```
let strLength: number = (someValue as string).length;
```

函数

我们可以给函数参数和返回值加上类型说明：

```
function add(x: number, y: number): number {
    return x + y;
}
```

```
let myAdd = function(x: number, y: number): number { return x + y; };
```

函数的完整类型：

```
let myAdd: (baseValue: number, increment: number) => number =
function(x: number, y: number): number { return x + y; };
```

可选参数和默认参数

```
function buildName(firstName: string, lastName?: string) {
    // ...
}

function buildName(firstName: string, lastName = "Smith") {
    // ...
}
```

接口

TypeScript的核心原则之一是对值所具有的结构进行类型检查。在TypeScript里，接口的作用就是为这些类型命名和为你的代码或第三方代码定义契约。

```
interface Person{
    name: string;
    age?: number;
}
function printInfo(info: Person){
    console.log(info);
}
let info = {
    "name": "mu",
    "age": 23
};
printInfo(info);
```

```
function printInfo(info: {
    name: string;
    age?: number;
}){
    console.log(info);
}
```

可索引类型

```
interface StringArray {
    [index: number]: string;
}

let myArray: StringArray;
myArray = ["Bob", "Fred"];

let myStr: string = myArray[0];
```

接口继承类

```
interface ClockInterface{
    currentTime: Date;
    setTime(d: Date);
}
class Clock implements ClockInterface{
    currentTime: Date;
    setTime(d: Date){
        this.currentTime = d;
    }
    constructor(h: number, m: number) {}
}
```

泛型

软件工程中，我们不仅要创建一致的定义良好的API，同时也要考虑可重用性。 组件不仅能够支持当前的数据类型，同时也能支持未来的数据类型，这在创建大型系统时为你提供了十分灵活的功能。

泛型可以用来代表任意一种类型：

```
function identity<T>(arg: T): T {
    return arg;
}
```

在Vue中使用的方法

- 安装依赖：TypeScript ts-loader
- 配置webpack和tsconfig.json

```
import Vue, { ComponentOptions }  from 'vue'
export default {
    props: ['message'],
    template: '<span>{{ message }}</span>'
} as ComponentOptions<Vue>
```

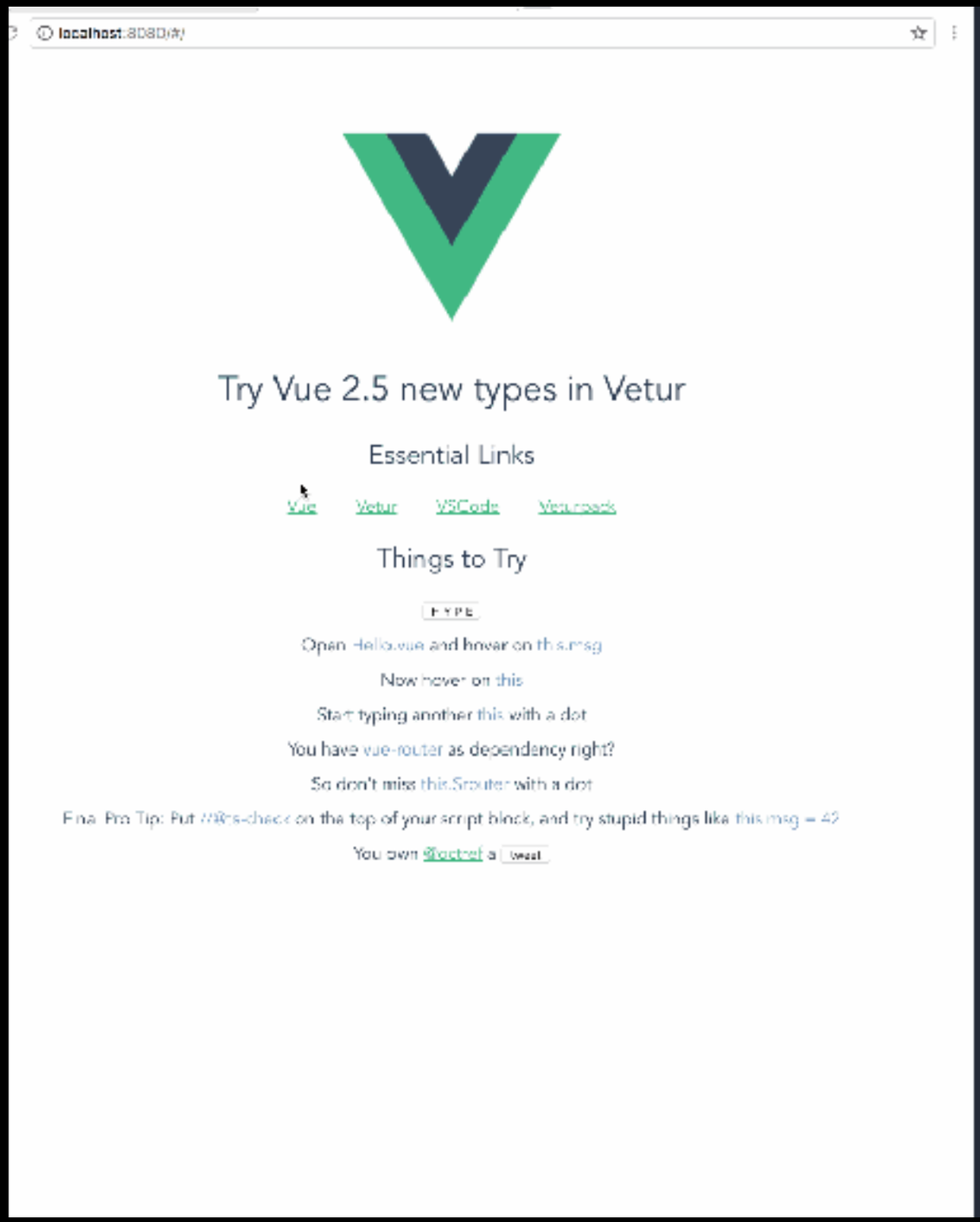
这种方法不能推断出data和method的类型。
但是在即将到来的Vue 2.5中，因为TS的新特性，得以自动推断这些属性的类型。

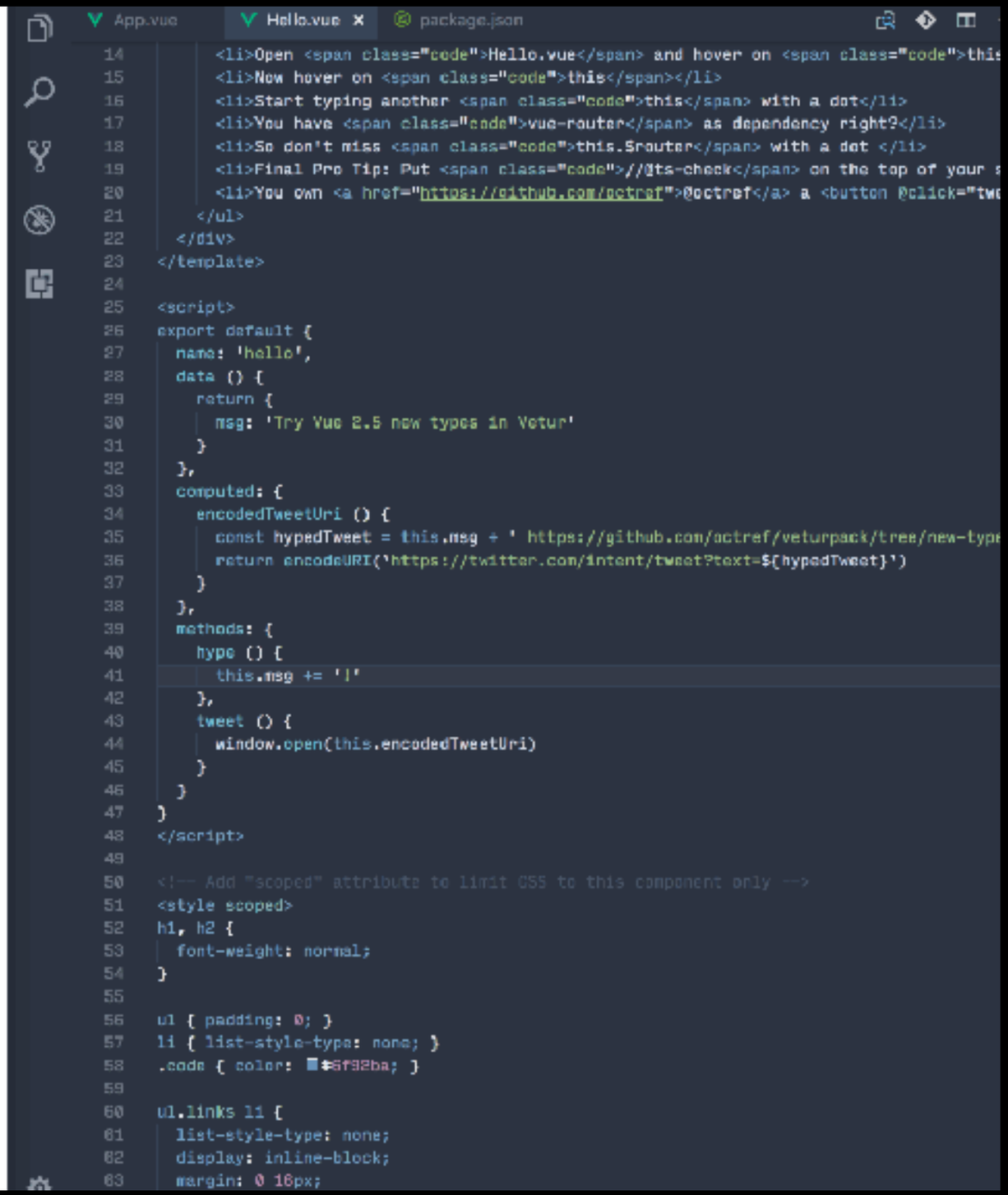
或者使用 vue-class-component

```
import Vue from 'vue'
import Component from 'vue-class-component'
// @Component 修饰符注明了此类为一个 Vue 组件
@Component({
  // 所有的组件选项都可以放在这里
  template: '<button @click="onClick">Click!</button>'
})
export default class MyComponent extends Vue {
  // 初始数据可以直接声明为实例的属性
  message: string = 'Hello!'
  // 组件方法也可以直接声明为实例的方法
  onClick (): void {
    window.alert(this.message)
  }
}
```

声明第三方插件

```ts
// 1. 确保在声明补充的类型之前导入 'vue'
import Vue from 'vue'
// 2. 定制一个文件，设置你想要补充的类型
//    在 types/vue.d.ts 里 Vue 有构造函数类型
declare module 'vue/types/vue' {
// 3. 声明为 Vue 补充的东西
  interface Vue {
    $myProperty: string
  }
}
```


localhost:8080/#/
Try Vue 2.5 new types in Vetur
Essential Links
Vue
Vetur
VSCode
Vetur-pack
Things to Try
TYPE
Open Hello.vue and hover on this.msg
Now hover on this
Start typing another this with a dot
You have vue-router as dependency right?
So don't miss this.$router with a dot
Final Pro Tip: Put //@ts-check on the top of your script block, and try stupid things like this.msg = 42
You own @octref a tweet


App.vue
Hello.vue
package.json
Open Hello.vue and hover on this
Now hover on this
Start typing another this with a dot
You have vue-router as dependency right?
So don't miss this.$router with a dot
Final Pro Tip: Put //@ts-check on the top of your
You own @octref a <button @click="tw

</template>
<script>
export default {
name: 'hello',
data () {
return {
msg: 'Try Vue 2.5 new types in Vetur'
}
},
computed: {
encodedTweetUri () {
const hypedTweet = this.msg + ' https://github.com/octref/veturpack/tree/new-typ
return encodeURI('https://twitter.com/intent/tweet?text=${hypedTweet}')
}
},
methods: {
hype () {
this.msg += '!'
},
tweet () {
window.open(this.encodedTweetUri)
}
}
}
</script>
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h1, h2 {
font-weight: normal;
}
ul { padding: 0; }
li { list-style-type: none; }
.code { color: #6f92ba; }
ul.links li {
list-style-type: none;
display: inline-block;
margin: 0 10px;

再见