## **认识 Flow**

# **认识 Flow**

[Flow](https://flow.org/en/docs/getting-started/" \t "https://coding.imooc.com/lesson/_blank) 是 facebook 出品的 JavaScript 静态类型检查工具。Vue.js 的源码利用了 Flow 做了静态类型检查，所以了解 Flow 有助于我们阅读源码。

## **为什么用 Flow**

JavaScript 是动态类型语言，它的灵活性有目共睹，但是过于灵活的副作用是很容易就写出非常隐蔽的隐患代码，在编译期甚至看上去都不会报错，但在运行阶段就可能出现各种奇怪的 bug。

类型检查是当前动态类型语言的发展趋势，所谓类型检查，就是在编译期尽早发现（由类型错误引起的）bug，又不影响代码运行（不需要运行时动态检查类型），使编写 JavaScript 具有和编写 Java 等强类型语言相近的体验。

项目越复杂就越需要通过工具的手段来保证项目的维护性和增强代码的可读性。 Vue.js 在做 2.0 重构的时候，在 ES2015 的基础上，除了 ESLint 保证代码风格之外，也引入了 Flow 做静态类型检查。之所以选择 Flow，主要是因为 Babel 和 ESLint 都有对应的 Flow 插件以支持语法，可以完全沿用现有的构建配置，非常小成本的改动就可以拥有静态类型检查的能力。

## **Flow 的工作方式**

通常类型检查分成 2 种方式：

****类型推断****：通过变量的使用上下文来推断出变量类型，然后根据这些推断来检查类型。

* ****类型注释****：事先注释好我们期待的类型，Flow 会基于这些注释来判断。

### **类型判断**

它不需要任何代码修改即可进行类型检查，最小化开发者的工作量。它不会强制你改变开发习惯，因为它会自动推断出变量的类型。这就是所谓的类型推断，Flow 最重要的特性之一。

通过一个简单例子说明一下：

/\*@flow\*/``

function split(str) {

return str.split(' ')

}

split(11)

Flow 检查上述代码后会报错，因为函数 split 期待的参数是字符串，而我们输入了数字。

### **类型注释**

如上所述，类型推断是 Flow 最有用的特性之一，不需要编写类型注释就能获取有用的反馈。但在某些特定的场景下，添加类型注释可以提供更好更明确的检查依据。

考虑如下代码：

/\*@flow\*/

function add(x, y){

return x + y

}

add('Hello', 11)

Flow 检查上述代码时检查不出任何错误，因为从语法层面考虑， + 即可以用在字符串上，也可以用在数字上，我们并没有明确指出 add() 的参数必须为数字。

在这种情况下，我们可以借助类型注释来指明期望的类型。类型注释是以冒号 : 开头，可以在函数参数，返回值，变量声明中使用。

如果我们在上段代码中添加类型注释，就会变成如下：

/\*@flow\*/

function add(x: number, y: number): number {

return x + y

}

add('Hello', 11)

现在 Flow 就能检查出错误，因为函数参数的期待类型为数字，而我们提供了字符串。

上面的例子是针对函数的类型注释。接下来我们来看看 Flow 能支持的一些常见的类型注释。

#### 数组

/\*@flow\*/

var arr: Array<number> = [1, 2, 3]

arr.push('Hello')

数组类型注释的格式是 Array<T>，T 表示数组中每项的数据类型。在上述代码中，arr 是每项均为数字的数组。如果我们给这个数组添加了一个字符串，Flow 能检查出错误。

#### 类和对象

/\*@flow\*/

class Bar {

x: string; // x 是字符串

y: string | number; // y 可以是字符串或者数字

z: boolean;

constructor(x: string, y: string | number) {

this.x = x

this.y = y

this.z = false

}

}

var bar: Bar = new Bar('hello', 4)

var obj: { a: string, b: number, c: Array<string>, d: Bar } = {

a: 'hello',

b: 11,

c: ['hello', 'world'],

d: new Bar('hello', 3)

}

类的类型注释格式如上，可以对类自身的属性做类型检查，也可以对构造函数的参数做类型检查。这里需要注意的是，属性 y 的类型中间用 | 做间隔，表示 y 的类型即可以是字符串也可以是数字。

对象的注释类型类似于类，需要指定对象属性的类型。

#### Null

若想任意类型 T 可以为 null 或者 undefined，只需类似如下写成 ?T 的格式即可。

/\*@flow\*/

var foo: ?string = null

此时，foo 可以为字符串，也可以为 null。

目前我们只列举了 Flow 的一些常见的类型注释。如果想了解所有类型注释，请移步 Flow 的[官方文档](https://flow.org/en/docs/types/" \t "https://coding.imooc.com/lesson/_blank)。

## **Flow 在 Vue.js 源码中的应用**

有时候我们想引用第三方库，或者自定义一些类型，但 Flow 并不认识，因此检查的时候会报错。为了解决这类问题，Flow 提出了一个 libdef 的概念，可以用来识别这些第三方库或者是自定义类型，而 Vue.js 也利用了这一特性。

在 Vue.js 的主目录下有 .flowconfig 文件， 它是 Flow 的配置文件，感兴趣的同学可以看[官方文档](https://flow.org/en/docs/config/" \t "https://coding.imooc.com/lesson/_blank)。这其中的 [libs] 部分用来描述包含指定库定义的目录，默认是名为 flow-typed 的目录。

这里 [libs] 配置的是 flow，表示指定的库定义都在 flow 文件夹内。我们打开这个目录，会发现文件如下：

flow

├── compiler.js # 编译相关

├── component.js # 组件数据结构

├── global-api.js # Global API 结构

├── modules.js # 第三方库定义

├── options.js # 选项相关

├── ssr.js # 服务端渲染相关

├── vnode.js # 虚拟 node 相关

可以看到，Vue.js 有很多自定义类型的定义，在阅读源码的时候，如果遇到某个类型并想了解它完整的数据结构的时候，可以回来翻阅这些数据结构的定义。

## **总结**

通过对 Flow 的认识，有助于我们阅读 Vue 的源码，并且这种静态类型检查的方式非常有利于大型项目源码的开发和维护。类似 Flow 的工具还有如 TypeScript，感兴趣的同学也可以自行去了解一下。

****任务****

请仔细阅读此文档