

43 | 元编程：通过Proxies和Reflect赋能元编程

2022-12-27 石川 来自北京



《JavaScript进阶实战课》

[课程介绍 >](#)



讲述：石川

时长 10:01 大小 9.16M



你好，我是石川。

今天，我们来到了这个单元最后的一课。前面的两节课中，我们分别学习了微前端和大前端“一大一小”两个趋势，今天我们再来看看“元”编程。那么，元编程中的“元”代表什么呢？“元”有“之上”或“超出一般限制”的意思。听上去，比较玄乎，实际上，要理解和使用元编程，并不难，甚至你可能每天都在用，只是没有觉察到而已。所以今天，就让我们一步步了解下元编程的概念及使用。

在 JavaScript 中，我们可以把元编程的功能分为几类：第一类是查找和添加对象属性相关的功能；第二类是创建 DSL 这样的特定领域语言；第三类就是可以作为代理用于对象的装饰器。今天，就让我们一一来看一下。

对象属性的属性

1. 对象属性的设置

首先，我们先来看下对象相关属性的查找和添加。我们都知道 Javascript 对象的属性包含了名称和值。但是另外，我们需要了解的是，每个属性本身也有三个相关的属性，它们分别为**可写属性（writable）**、**可枚举属性（enumerable）**以及**可配置属性（configurable）**。

这三个属性指定了该属性的行为方式，以及我们可以使用它们做什么。这里的可写属性指定了属性的值是否可以更改；可枚举属性指定了属性是否可以由 `for/in` 循环和 `Object.keys()` 方法枚举；可配置属性指定了是否可以删除属性或更改属性的属性。

这里，需要注意的是，我们自定义的对象字面量和通过赋值定义的对象属性都是可写、可枚举和可配置的，但 JavaScript 标准库中定义的很多对象属性都不是。从这里，你应该可以看出，只要你使用过 `for/in`，那么，恭喜你，基本你已经使用过元编程开发了。

那么对属性的查询和设置有什么用呢？这对于很多第三方库的开发者来说是很重要的，因为它允许开发者向原型对象中添加方法，而且也可以像标准库中的很多内置方法一样，将它们设置成不可枚举；同时，它也可以允许开发者“锁定”对象，让属性定义无法被更改或删除。下面，我们就来看看可以为 JavaScript 第三方库的开发赋能的查询和设置属性的 API。

我们可以将属性分为两类，一类是“**数据属性**”，一类是“**访问器属性**”。如果我们把值、getter、setter 都看做是值的话，那么，数据属性就包含了值、可写、可枚举和可配置这 4 个属性；而访问器属性则包含了 `get`、`set`、可枚举和可配置这 4 个属性。其中可写、可枚举和可配置属性是布尔值，`get` 和 `set` 属性是函数值。

我们可以通过 `Object.getOwnPropertyDescriptor()` 来获取对象属性的属性描述，顾名思义，这个方法只适用于获取对象自身的属性，如果要查询继承属性的属性，就需要通过 `Object.getPrototypeOf()` 的方法来遍历原型链。如果要设置属性的属性或者使用指定的属性创建新属性，就需要用到 `Object.defineProperty()` 的方法。

2. 对象的可延展性

除了对对象的属性的获取和设置外，我们对对象本身也可以设置它的可延展性。我们可以通过 `Object.isExtensible()` 让一个对象可延展，同样的，我们也可以通过 `Object.preventExtensions()` 将一个对象设置为不可延展。

不过这里需要注意的是，一旦我们把一个对象设置为不可延展，我们不仅不可以在对象上再设置属性，而且我们也不可以再把对象改回为可延展。还有就是这个不可延展只影响对象本身的属性，对于对象的原型对象上的属性来说，是不受影响的。



对象的可延展性通常的作用是用于对象状态的锁定。通常我们把它和属性设置中的可写属性和可配置属性结合来使用。在 JavaScript 中，我们可以通过 `Object.seal()` 把不可延展属性和不可配置属性结合；通过 `Object.freeze()` 我们可以把不可延展、不可配置和不可写属性结合起来。

对于 JavaScript 第三方库的编写来说，如果库将对象传递给库的回调函数，那么我们就可以使用 `Object.freeze()` 来防止用户的代码对它们进行修改了。不过需要注意的是，这样的方法可能对 JavaScript 测试策略形成干扰。

3. 对象的原型对象

前面，我们介绍的 `Object.freeze()`、`Object.seal()` 和属性设置的方法一样，都是仅作用于对象本身的，都不会对对象的原型造成影响。我们知道，通过 `new` 创建的对象会使用创建函数的原型值作为自己的原型，通过 `Object.create()` 创建的对象会使用第一个参数作为对象的原型。

我们可以通过 `Object.getPrototypeOf()` 来获取对象的原型；通过 `isPrototypeOf()` 我们可以判断一个对象是不是另外一个对象的原型；同时，如果我们想要修改一个对象的原型，可以通过 `Object.setPrototypeOf()`。不过有一点需要注意的是，通常在原型已经设置后，就很少被改变了，使用 `Object.setPrototypeOf()` 有可能对性能产生影响。

用于 DSL 的模版标签

我们知道，在 JavaScript 中，在反引号内的字符串被称为模板字面量。当一个值为函数的表达式，并且后面跟着一个模板字面量时，它会变成一个函数被调用，我们将它称之为“带标签的模板字面量”。

为什么我们说定义一个新的标签函数，用于标签模板字面量可以被当做是一种元编程呢？因为标签模板通常用于定义 DSL，也就是域特定语言，这样定义新的标签函数就如同向 JavaScript 中添加了新的语法。标签模板字面量已被许多前端 JavaScript 库采用。GraphQL 查询语言通

过使用 `gql`` 标签函数，可以使查询被嵌入到 JavaScript 代码中。Emotion 库使用 `css`` 标签函数，使 CSS 样式同样可以被嵌入到 JavaScript 中。



当函数表达式后面有模板字面量时，该函数将被调用。第一个参数是字符串数组，后面是零或多个其它参数，这些参数可以具有任何类型的值。参数的数量取决于插入到模板字面量的值的数量，模板字面量的值始终是字符串，但是标签模板字面量的值是标签函数返回的值。它可能是一个字符串，但当使用标签函数实现 DSL 时，返回值通常是一个非字符串数据结构，它是字符串的解析表示。

当我们想将一个值安全地插入到 HTML 字符串中时，模版会非常得有用。我们拿 `html`` 为例，在使用标签构建最终字符串之前，标签会对每个值执行 HTML 转义。

复制代码

```
1 function html(str, ...val) {
2   var escaped = val.map(v => String(v)
3     .replace("&", "&amp;")
4     .replace("'", "&#39;"));
5   var result = str[0];
6   for(var i = 0; i < escaped.length; i++) {
7     result += escaped[i] + str[i+1];
8   }
9   return result;
10 }
11
12 var operator = "&";
13 html`<b>x ${operator} y</b>` // => "<b>x &amp; y</b>"
```

下面，我们再来看看 Reflect 对象。Reflect 并不是一个类，和 Math 对象类似，它的属性只是定义了一组相关的函数。ES6 中添加的这些函数都在一个命名空间中，它们模仿核心语言的行为，并且复制了各种预先存在于对象函数的特性。

尽管 Reflect 函数没有提供任何新功能，但它们确实将这些功能组合在一个 API 中方便使用。比如，我们在上面提到的对象属性的设置、可延展性以及对象的原型对象在 Reflect 中都有对应的方法，如 `Reflect.set()`、`Reflect.isExtensible()` 和 `Reflect.getPrototypeOf()`，等等。下面，我们会看到 Reflect 函数集与 Proxy 的处理程序方法集也可以一一对应。

Proxy 和 Reflect

在 ES6 和更高版本中提供的 Proxy 类可以算是 JavaScript 中最强大的元编程功能了。它允许我们编写改变 JavaScript 对象基本行为的代码。我们在前面提到的 Reflect API 是一组函数，它使我们可以直接访问 JavaScript 对象上的一组基本操作。当我们创建 Proxy 对象时，我们指定了另外两个对象，目标对象和处理程序对象。

 复制代码

```
1 var target = {
2   message1: "hello",
3   message2: "world",
4 };
5 var handler = {};
6
7 var proxy = new Proxy(target, handler);
```

生成的代理对象没有自己的状态或行为。无论何时对其执行操作（读取属性、写入属性、定义新属性、查找原型、将其作为函数调用），它都会将这些操作分派给处理程序对象或目标对象。代理对象支持的操作与 Reflect API 定义的操作相同。Proxy 的工作机制是，如果 handler 是空的，那么代理对象只是一层透明的装饰器。所以在上面的例子中，如果我们执行代理，那么它返回的结果就是目标对象上本来自有的属性。

 复制代码

```
1 console.log(proxy.message1); // hello
2 console.log(proxy.message2); // world
```

通常，我们会把 Proxy 和 Reflect 结合起来使用，这样的好处是，对于我们不想自定义的部分，我们可以使用 Reflect 来调用对象内置的方法。

 复制代码

```
1 const target = {
2   message1: "hello",
3   message2: "world",
4 };
5
6 const handler = {
7   get(target, prop, receiver) {
8     if (prop === "message2") {
9       return "Jackson";
10    }
11    return Reflect.get(...arguments);
12  },
```

```
13 };  
14 var proxy = new Proxy(target, handler);  
15 console.log(proxy.message1); // hello  
16 console.log(proxy.message2); // Jackson
```



总结

通过今天这节课，我们看到了 **JavaScript** 对象的属性本身也带有可以查找和添加的属性。同时，我们学习了，**JavaScript** 定义的函数允许我们遍历对象的原型链，甚至更改对象的原型。

标签模板字面量是一种函数调用语法，我们可以用它来定义新的标签函数，这样做有点像向语言中添加新的字面量语法。通过定义一个解析其模板字符串参数的标签函数，我们就可以在 **JavaScript** 代码中嵌入 **DSL**。标签函数还提供对字符串字面量的原始、非转义形式的访问，其中反斜杠不帶有任何特殊含义。


最后，我们又看了 **Proxy** 类和相关的 **Reflect API**。**Proxy** 和 **Reflect** 允许我们对 **JavaScript** 中的对象的基本行为进行低级控制。**Proxy** 对象可以用作可选的、可撤销的包装器，以改进代码封装，还可以用于实现非标准对象行为（如早期 **Web** 浏览器定义的一些特殊情况下的 **API**）。

思考题

我们知道 **Symbol** 对象的属性的值可以用于定义对象和类的属性或方法的名称。这样做可以控制对象如何与 **JavaScript** 语言特性和核心库交互。那么，你觉得 **Symbol** 在这种使用场景下算不算是一种元编程呢？

欢迎在留言区分享你的观点、交流学习心得或者提出问题，如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。我们下节课再见！

分享给需要的人，Ta购买本课程，你将得 18 元

 生成海报并分享

 赞 1  提建议

上一篇 42 | 大前端：通过一云多端搭建跨PC/移动的平台应用

下一篇 结束语 | JavaScript的未来之路：源于一个以终为始的初心



精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。