kitety刘小生 Lv2
2018年11月09日 阅读 3686

关注

[译]JS箭头函数三连问：为何用、怎么用、何时用

在现代JS中最让人期待的特性就是关于箭头函数，用`=>`来标识。箭头函数有两个主要的优点：其一是非常简明的语法，另外就是直观的作用域和`this`的绑定。

因为这些优点，箭头函数比起其他形式的函数声明更加受欢迎。比如，受欢迎的`airbnb eslint configuration库`会强制使用JavaScript箭头函数创建匿名函数。

然而，就像世间万物一样，箭头函数有一些优点也有一些"缺点"，这就需要在使用的时候做一些权衡了。

学习如何权衡是使用好箭头函数的关键。在这篇文章中我们将回顾箭头函数是怎样工作的，然后深入探讨，实际代码中箭头函数是如何改进我们代码的，以及一些箭头函数不推荐的情况。

什么才是箭头函数

JS的箭头函数大概就像python中的`lambda(python定义匿名函数的关键字)`和ruby中的`blocks(类似于闭包)`一样。 这些匿名函数都有他们特殊的语法：首先接收一定数目的参数，然后在定义它们的函数的作用域或就近作用域中执行。

接下来我们将详细探讨这些。

箭头函数的语法

/

箭头函数有一个大体的结构，同时也有很多的特殊情况可以简化。 核心的结构如下：

```javascript
(argument1, argument2, ... argumentN) => {
    // function body
}
```


在括号里面有一系列的参数，接着跟着一个箭头符号 `=>`，最后是函数体。这跟传统的函数很相像，只是我们省略了 `function` 关键字，并且添加了一个 `=>` 在参数后面。

并且，这里也有很多种情况，让箭头函数结构变得更加的简洁。

首先，如果函数体里面是一个单独的表达式，你可以省略大括号直接将表达式写在一行，并且表达式的结果也将会被函数直接返回。比如：

```javascript
const add = (a, b) => a + b;
```


其次，如果这传入的是一个单独的参数，你也可省略参数部分的括号。比如：

```javascript
const getFirst = array => array[0];
```


如你所见，这样就看起来更加的简洁了，我们也将在后面说明更多的特性。

高级语法

如果你了解这些高级语法之后将十分受用。

首先，如果你尝试在一行书写函数，但是返回的值却是一个对象内容，你原想这样写：

```javascript 复制代码
(name, description) => {name: name, description: description};
```

而问题就是这样的语法会引起歧义，会误以为你在写一个函数的函数体。 如果想返回的是单个的对象，请用括号包装该对象：

```JavaScript 复制代码
(name, description) => ({name: name, description: description});
```

封闭的上下文作用域

不像其他形式的函数，箭头函数并没有他们自己的执行上下文。实际上，这就意味着代码中的 `this` 和 `arguments` 都是继承自他们的父函数。

比如，比较下面箭头函数和传统函数的区别：

```javascript 复制代码
const test = {
  name: 'test object',
  createAnonFunction: function() {
    return function() {
      console.log(this.name);
      console.log(arguments);
    };
  },
  createArrowFunction: function() {
    return () => {
      console.log(this.name);
      console.log(arguments);
    };
```

/

```
  }
};
```

我们有一个有两个方法的对象，每个方法都返回了一个匿名函数。区别在于第一个方法里面用了传统的函数表达式，后面的用了箭头函数表达式。如果我们在传入同样的参数运行，我们得到了两个不同的结果。

```javascript 复制代码
const anon = test.createAnonFunction('hello', 'world');
//返回匿名函数
const arrow = test.createArrowFunction('hello', 'world');
anon();
//undefined
//{}
// this->window
arrow();
//test object
//object { '0': 'hello', '1': 'world' }
//this->test
```

第一个匿名函数有自己的上下文（指向并非test对象），当你调用的时候没有参考的 `this.name` 的属性，（注意：现在 `this` 指向 `window` ），也没有创建它时调用的参数。另一个，箭头函数与创建它的函数有相同的上下文，让其可以访问参数arguments和对象。

箭头函数改进您的代码

传统 `lambda` 函数的主要用例之一，就是将函数用于数组的遍历，现在用JavaScript箭头函数实现。 比如你有一个有值的数组，你想去 `map` 遍历每一项，这时箭头函数是非常推荐的：

```javascript 复制代码
const words = ['hello', 'WORLD', 'Whatever'];
const downcasedWords = words.map(word => word.toLowerCase());
```

/

一个及其常见的例子就是返回一个对象的某个值：

```javascript
const names = objects.map(object => object.name);
```

类似的，当用 `forEach` 来替换传统 `for` 循环的时候，实际上箭头函数会直观的保持 `this` 来自于父一级

```javascript
this.examples.forEach(example => {
  this.runExample(example);
});
```

Promise和Promise链

当在编写异步程序的时候，箭头函数也会让代码更加直观和简洁。

Promise可以更简单的编写异步程序。虽然你乐意去使用 `async/await` ，你也需要好好理解 `promise` ,因为这是他们的基础。

使用promise，仍然需要定义你的代码执行完成之后的回调函数。 这是箭头函数的理想位置，特别是如果您生成的函数是有状态的，同时想引用对象中的某些内容。

```javascript
this.doSomethingAsync().then((result) => {
  this.storeResult(result);
});
```

对象转换

箭头函数的另一个常见而且十分有用的地方就是用于封装的对象转换。 例如在Vue.js中，有一种通用模式，就是使用 `mapState` 将Vuex存储的各个部分，直接包含到Vue组件中。 这涉及到定义一套 `mappers` ，用于从原对象到完整的转换输出，这在组件问题中实十分有必要的。这一系列简单的转换,使用箭头函数是最合适不过的。比如：

```javascript
export default {
  computed: {
    ...mapState({
      results: state => state.results,
      users: state => state.users,
    });
  }
}
```

你不应该使用箭头函数的情景

这里有许多箭头函数不推荐的场景，这种情况之下不仅没有帮助，而且还会造成不必要的麻烦。

首先就是对象中的方法。这里有一个函数上下文的例子，对于我们理解很有帮助。 曾经流行一种趋势，用 `class` 类的语法和箭头函数，为其自动绑定方法。比如:事件方法可以使用，但是仍然绑定在class类中。 看起来就像下面的例子:

```javascript
class Counter {
  counter = 0;
  handleClick = () => {
    this.counter++;
  }
}
```

在这种方法中,如果被一个点击事件函数调用了，它虽然不是 `Counter` 的上下文中，它仍旧可以访问实例的数据，这种方式的缺点不言而喻。

用这种方式的确提供了一种绑定函数的快捷方式，但是函数的表达形式多种多样，相当不直观。如果你尝试在原型使用这种对象，这将不利于测试，同时也会产生很多问题。 相反，推荐用一种常规的绑定方式，如有必要可以绑定在实例的构造函数中:

```javascript 复制代码
class Counter {
  counter = 0;
  handleClick() {
    this.counter++;
  }
  constructor() {
    this.handleClick = this.handleClick.bind(this);
  }
}
```

深层调用

另一种使用箭头函数会让你头疼的地方，就是你去用很多函数的组合调用，尤其是函数的深层调用。 简单的理由跟匿名函数一样，堆栈的追踪很复杂。

如果你的函数仅仅在一层之下，而不是深层的迭代，这倒不是什么问题。但是如果你将函数定义为箭头函数，并且在他们之间来回调用，当你调试bug的时候你将被代码困惑，甚至得到如下的错误信息：

```javascript 复制代码
{anonymous}()
{anonymous}()
{anonymous}()
{anonymous}()
```

/

```
{anonymous}()
//anonymous 匿名
```

有动态上下文的函数

还有最有一种箭头函数会让你困惑的情形，就是`this`是动态绑定的时候。 如果你在以下情形使用箭头函数，那么this的动态绑定不会如期工作，并且你也会困惑这些代码为什么不像预期那样工作，也会给你之后工作的人造成麻烦。 一些典型的例子：

- 事件的调用函数，this指向当前的目标属性
- 在jquery中，大多数时候this指向的是当前被选择的元素
- 在vue中，`methods`和`computed`中的`this`指向的是vue的组件。

当然你也可以在上面的情形之下谨慎的使用箭头函数。但特别是在`jquery`和`vue`的情况下, 这通常会干扰正常功能, 并使您感到困惑：为什么看起来跟别人代码一样的代码就是不工作。

总结

箭头函数是JS语言中十分特别的属性，并且使很多情形中代码更加的变化莫测。尽管如此，就像其他的语言特性，他们有各自的优缺点。因此我们使用它应该仅仅是作为一种工具，而不是无脑的简单的全部替换为箭头函数。

本文只是个人兴趣翻译，如有错误之处还望各位斧正。文章版权属于原文作者。

> 原文地址：codeburst.io/javascript-...

> **关注下面的标签，发现更多相似文章**

Vue.js JavaScript jQuery Promise

kitety刘小生 Lv2 前端工程师 @ simviso

发布了 4 篇专栏 · 获得点赞 126 · 获得阅读 9,072

关注

输入评论...

ositowang WorkAround Engineer @ 北美...

emmmm 把事件函数用箭头函数绑定组件上 不应该也算是。。react官方文档上的一种推荐吗。。。

2年前 回复

kitety刘小生 Lv2 (作者) 前端工程师 @ simviso

绑定在组件上也是为了绑定this，与之类似的有xxx.bind(this)写法。这个的确是一个应用，而英文原文中没有涉及到，我也就没有加上了。 😃😂

2年前 回复