Sunday • 更新于 2022-10-19

上一节 06：vue3的响应... 08：总结 下一节

07：proxy的最佳拍档：Reflect—拦截 js 对象操作

当我们了解了 Proxy 之后，那么接下来我们需要了解另外一个 Proxy 的“伴生对象”：Reflect

Reflect 属性,多数时候会与 proxy 配合进行使用在 MDN Proxy 的例子中， Reflect 也有对此出现。

那么 Reflect 的作用是什么呢?

查看 MDN 的文档介绍，我们可以发现 Reflect 提供了非常多的静态方法，并且很巧的是这些方法与 Proxy 中 Handler 的方法类似：

> ### Reflect 静态方法
>
> ```
> Reflect.get(target, propertyKey[, receiver\])
> Reflect.has(target, propertyKey)
> Reflect.set(target, propertyKey, value[, receiver\])
> ...
> ```
>
> ### handler 对象的方法
>
> ```
> handler.has()
> handler.get()
> handler.set()
> ...
> ```

我们现在已经知道了 handler 中 get 和 set 的作用，那么 Reflect 中 get 和 set 的作用是什么呢?

我们来看一下代码：

代码块

```
<script>
  const obj = {
    name: '张三'
  }

  console.log(obj.name) // 张三
  console.log(Reflect.get(obj, 'name')) // 张三
</script>
```

由以上代码可以发现，两次打印的结果是相同的。这其实也就说明了 Reflect.get(obj, 'name') 本质上和 obj.name 的作用 **相同**

那么既然如此，我们为什么还需要 Reflect 呢?

根据官方文档可知，对于 Reflect.get 而言，它还存在第三个参数 receiver ，那么这个参数的作用是什么呢?

根据官网的介绍为：

> 如果 target 对象中指定了 getter， receiver 则为 getter 调用时的 this 值。

什么意思呢？我们来看以下代码：

代码块

```
<script>
  const p1 = {
    lastName: '张',
    firstName: '三',
    // 通过 get 标识符标记，可以让方法的调用像属性的调用一样
    get fullName() {
      return this.lastName + this.firstName
    }
  }

  const p2 = {
    lastName: '李',
    firstName: '四',
    // 通过 get 标识符标记，可以让方法的调用像属性的调用一样
    get fullName() {
      return this.lastName + this.firstName
    }
  }

  console.log(p1.fullName) // 张三
  console.log(Reflect.get(p1, 'fullName')) // 张三
  // 第三个参数 receiver 在对象指定了 getter 时表示为 this
  console.log(Reflect.get(p1, 'fullName', p2)) // 李四
</script>
```

在以上代码中，我们可以利用 `p2` 作为第三个参数 `receiver` ，以此来修改 `fullName` 的打印结果。即：**此时触发的 `fullName` 不是 `p1` 的而是 `p2` 的。**

那么明确好了这个之后，我们再来看下面这个例子：

代码块

```
<script>
  const p1 = {
    lastName: '张',
    firstName: '三',
    // 通过 get 标识符标记，可以让方法的调用像属性的调用一样
    get fullName() {
      return this.lastName + this.firstName
    }
  }

  const proxy = new Proxy(p1, {
    // target：被代理对象
    // receiver：代理对象
    get(target, key, receiver) {
      console.log('触发了 getter');
      return target[key]
    }
  })

  console.log(proxy.fullName);
</script>
```

在以上这个代码中，我问大家，此时我们触发了 `prox.fullName` ，在这个 `fullName` 中又触发了 `this.lastName + this.firstName` 那么问：**getter 应该被触发几次?**

此时 `getter` 应该被触发 3 次！ ，但是 **实际只触发了 1 次！** 。为什么?

可能有同学已经想到了，因为在 `this.lastName + this.firstName` 这个代码中，我们的 `this` 是 `p1` ，**而非** `proxy` ！所以 `lastName` 和 `firstName` 的触发，不会再次触发 `getter` 。

那么怎么办呢？我们如何能够让 `getter` 被触发三次?

想要实现这个想过，那么就需要使用到 `Reflect.get` 了。

我们已知， `Reflect.get` 的第三个参数 `receiver` 可以修改 `this` 指向，那么我们可不可以 **利用 Reflect.get 把 fullName 中的 this 指向修改为 proxy**，依次来达到触发三次的效果?

我们修改以上代码：

代码块

```
1    const proxy = new Proxy(p1, {
2        // target：被代理对象
3        // receiver：代理对象
4        get(target, key, receiver) {
5          console.log('触发了 getter');
6    +       // return target[key]
7    +       return Reflect.get(target, key, receiver)
8        }
9      })
```

修改代码之后，我们发现，此时 `getter` 得到了三次的触发!

总结

本小节的内容比较多，但是核心其实就是在描述一件事情，那就是 `Reflect` 的作用，我们为什么要使用它。

最后做一个总结：

当我们期望监听代理对象的 `getter` 和 `setter` 时，**不应该使用** `target[key]`，因为它在某些时刻（比如 `fullName`）下是不可靠的。而 **应该使用** `Reflect`，借助它的 `get` 和 `set` 方法，使用 `receiver（proxy 实例）` 作为 `this`，已达到期望的结果（触发三次 `getter`）。

06：vue3的响应性核心 API：proxy ‹ 上一节　　下一节 › 08：总结

我要提出意见反馈