双11返场进行中！优惠Go on，快来捡漏！ 02 天 15 时 07 分 19 秒 Go >>>

Sunday • 更新于 2022-10-19

上一节 03：框架实现：... 05：源码阅读：... 下一节

04：总结：ref 复杂数据类型的响应性

根据以上代码实现我们知道，针对于 `ref` 的复杂数据类型而言，它的响应性本身，其实是 **利用** `reactive` **函数** 进行的实现，即：

代码块

```
const obj = ref({
    name: '张三'
})

const obj = reactive({
    name: '张三'
})
```

本质上的实现方案其实是完全相同的，都是利用 `reactive` 函数，返回一个 `proxy` 实例，监听 `proxy` 的 `getter` 和 `setter` 进行的依赖收集和依赖触发。

但是它们之间也存在一些不同的地方，比如：

1. `ref`：
    1. `ref` 的返回值是一个 `RefImpl` 类型的实例对象
    2. 想要访问 `ref` 的真实数据，需要通过 `.value` 来触发 `get value` 函数，得到被 `toReactive` 标记之后的 `this._value` 数据，即：`proxy` 实例
2. `reactive`：
    1. `reactive` 会直接返回一个 `proxy` 的实例对象，不需要通过 `.value` 属性得到

同时我们也知道，对于 `reactive` 而言，它是不具备 **简单数据类型** 的响应性呢，但是 `ref` 是具备的。

那么 `ref` 是如何处理 **简单数据类型** 的响应性的呢？我们继续来往下看~~~

03：框架实现：ref 函数 - 构建复杂数据类型... 上一节 下一节 05：源码阅读：ref 简单数据类型的响应性

我要提出意见反馈

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备110108020301151号