Sunday • 更新于 2022-10-19

上一节 02：源码阅读：... 04：总结：ref ... 下一节

03：框架实现：ref 函数 - 构建复杂数据类型的响应性

在上一小节中，我们已经查看了 `vue 3` 中 `ref` 函数针对复杂数据类型的响应性处理代码逻辑，那么这一小节，我们就可以实现一下对应的代码。

1. 创建 `packages/reactivity/src/ref.ts` 模块：

代码块

```ts
import { createDep, Dep } from './dep'
import { activeEffect, trackEffects } from './effect'
import { toReactive } from './reactive'

export interface Ref<T = any> {
    value: T
}

/**
 * ref 函数
 * @param value unknown
 */
export function ref(value?: unknown) {
    return createRef(value, false)
}

/**
 * 创建 RefImpl 实例
 * @param rawValue 原始数据
 * @param shallow boolean 形数据，表示《浅层的响应性（即：只有 .value 是响应性的）》
 * @returns
 */
function createRef(rawValue: unknown, shallow: boolean) {
    if (isRef(rawValue)) {
        return rawValue
    }
    return new RefImpl(rawValue, shallow)
}

class RefImpl<T> {
    private _value: T

    public dep?: Dep = undefined

    // 是否为 ref 类型数据的标记
    public readonly __v_isRef = true

    constructor(value: T, public readonly __v_isShallow: boolean) {
        // 如果 __v_isShallow 为 true，则 value 不会被转化为 reactive 数据，即如果当前 v
        this._value = __v_isShallow ? value : toReactive(value)
    }

    /**
     * get语法将对象属性绑定到查询该属性时将被调用的函数。
     * 即：xxx.value 时触发该函数
     */
    get value() {
```

```
        }

        set value(newVal) {}
    }

    /**
     * 为 ref 的 value 进行依赖收集工作
     */
    export function trackRefValue(ref) {
        if (activeEffect) {
            trackEffects(ref.dep || (ref.dep = createDep()))
        }
    }

    /**
     * 指定数据是否为 RefImpl 类型
     */
    export function isRef(r: any): r is Ref {
        return !!(r && r.__v_isRef === true)
    }
```

2. 在 `packages/reactivity/src/reactive.ts` 中，新增 `toReactive` 方法：

代码块

```
    /**
     * 将指定数据变为 reactive 数据
     */
    export const toReactive = <T extends unknown>(value: T): T =>
        isObject(value) ? reactive(value as object) : value
```

3. 在 `packages/shared/src/index.ts` 中，新增 `isObject` 方法：

代码块

```
    /**
     * 判断是否为一个对象
     */
    export const isObject = (val: unknown) =>
        val !== null && typeof val === 'object'
```

4. 在 `packages/reactivity/src/index.ts` 中，导出 `ref` 函数：

代码块

```
    ...
    export { ref } from './ref'
```

5. 在 `packages/vue/src/index.ts` 中，导出 `ref` 函数：：

代码块

```
    export { reactive, effect, ref } from '@vue/reactivity'
```

至此， `ref` 函数构建完成。

我们可以增加测试案例 `packages/vue/examples/reactivity/ref.html` 中：

代码块

```
  <script>
    const { ref, effect } = Vue

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

    // 调用 effect 方法
    effect(() => {
      document.querySelector('#app').innerText = obj.value.name
```

```
12
13      setTimeout(() => {
14        obj.value.name = '李四'
15      }, 2000);
16    </script>
```

可以发现代码测试成功。

02：源码阅读：ref 复杂数据类型的响应性 ◂ 上一节　　下一节 ▸ 04：总结：ref 复杂数据类型的响应性

我要提出意见反馈