

全部开发者教程

09: 框架实现：初步实现 watch 数据监听器

10: 问题分析：watch 下的依赖收集

11: 框架实现：完成 watch 数据监听器的依赖收集

12: 总结：watch 数据侦听器

13: 总结

第八章：runtime 运行时 - 运行时核心设计原则

01: 前言

02: HTML DOM 节点树与虚拟 DOM 树


03: 挂载与更新

04: h 函数与 render 函数

05: 运行时核心设计原则

06: 总结

第九章：runtime 运行时 - 构建 h 函数，生成 Vnode

 Sunday • 更新于 2022-10-19

◀ 上一节 10: 问题分析：... 12: 总结：watch... 下一节 ▶

11：框架实现：完成 watch 数据监听器的依赖收集

根据上一节所说，我们接下来就需要去实现 `traverse` 函数，在 `packages/runtime-core/src/apiWatch` 中，创建 `traverse` 方法：

<> 代码块

```
1  /**
2   * 依次执行 getter，从而触发依赖收集
3   */
4  export function traverse(value: unknown) {
5    if (!isObject(value)) {
6      return value
7    }
8
9    for (const key in value as object) {
10     traverse((value as any)[key])
11   }
12   return value
13 }
```

在 `doWatch` 中通过 `traverse` 方法，构建 `getter`：

<> 代码块

```
1  // 存在回调函数和deep
2  if (cb && deep) {
3    // TODO
4    const baseGetter = getter
5    getter = () => traverse(baseGetter())
6  }
```

此时再次运行测试实例，`watch` 成功监听。

同时因为我们已经处理了 `immediate` 的场景：

<> 代码块

```
1  if (cb) {
2    if (immediate) {
3      job()
4    } else {
5      oldValue = effect.run()
6    }
7  } else {
8    effect.run()
9  }
```

所以，目前 `watch` 也支持 `immediate` 的配置选项，`ref` 场景下的处理也可以得到支持，具体可见如下测试案例 `packages/vue/examples/reactivity/watch-2.html`

<> 代码块

```
1  <script>
2    const { ref, watch } = Vue
3
4    const obj = ref({
```

索引目录

11: 框架实现：...

📄

?

📱

💬

```
6    })
7
8    watch(obj.value, (value, oldValue) => {
9      console.log('watch 监听被触发');
10     console.log('value', value);
11   }, {
12     immediate: true
13   })
14
15   setTimeout(() => {
16     obj.value.name = '李四'
17   }, 2000);
18 </script>
```

10: 问题分析: watch 下的依赖收集 ◀ 上一节 下一节 ▶ 12: 总结: watch 数据侦听器

 我要提出意见反馈

[企业服务](#) [网站地图](#) [网站首页](#) [关于我们](#) [联系我们](#) [讲师招募](#) [帮助中心](#) [意见反馈](#) [代码托管](#)

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号



 意见反馈

 收藏教程

 标记书签