全部开发者教程

05：局部总结：无状态组件的挂载、更新、卸载总结

06：源码阅读：有状态的响应性组件挂载逻

07：框架实现：有状态的响应性组件挂载逻

08：源码阅读：组件生命周期回调处理逻辑

09：框架实现：组件生命周期回调处理逻辑

10：源码阅读：生命回调钩子中访问响应性数据

11：框架实现：生命回调钩子中访问响应性数据

12：源码阅读：响应性数据改变，触发组件的响应性变化

13：框架实现：响应性数据改变，触发组件的响应性变化

14：源码阅读：composition API，setup 函数挂载逻辑

Sunday • 更新于 2022-10-19

上一节 12：源码阅读：... 14：源码阅读：... 下一节

13：框架实现：响应性数据改变，触发组件的响应性变化

明确好了组件响应性更新，那么下面我们来实现下 `vue-next-mini` 的对应代码逻辑。

1. 在 `packages/runtime-core/src/renderer.ts` 的 `componentUpdateFn` 方法中，加入如下逻辑：

代码块

```
// 组件挂载和更新的方法
const componentUpdateFn = () => {
    // 当前处于 mounted 之前，即执行 挂载 逻辑
    if (!instance.isMounted) {
        ...
    // 修改 mounted 状态
    instance.isMounted = true
    } else {
        let { next, vnode } = instance
        if (!next) {
            next = vnode
        }

        // 获取下一次的 subTree
        const nextTree = renderComponentRoot(instance)

        // 保存对应的 subTree，以便进行更新操作
        const prevTree = instance.subTree
        instance.subTree = nextTree

        // 通过 patch 进行更新操作
        patch(prevTree, nextTree, container, anchor)

        // 更新 next
        next.el = nextTree.el
    }
}
```

至此，代码完成。

创建对应测试实例 `packages/vue/examples/runtime/redner-component-hook-data-change.html`：

代码块

```
<script>
  const { h, render } = Vue

  const component = {
    data() {
      return {
        msg: 'hello component'
      }
    },
    render() {
      return h('div', this.msg)
    },
    // 组件实例处理完所有与状态相关的选项之后
    created() {
```

索引目录

13：框架实现：响

```
        }, 2000);
      }
    }

    const vnode = h(component)
    // 挂载
    render(vnode, document.querySelector('#app'))
  </script>
```

得到响应性的组件更新。

12：源码阅读：响应性数据改变，触发组件... 上一节 下一节 14：源码阅读：composition API，setup ...