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 08：源码阅读：… 10：源码阅读：… 下一节

09：框架实现：组件生命周期回调处理逻辑

明确好了源码的生命周期处理之后，那么接下来我们来实现一下对应的逻辑。

我们本小节要处理的生命周期有四个，首先我们先处理前两个 `beforeCreate` 和 `created`，我们知道这两个回调方法是在 `applyOptions` 方法中回调的：

1. 在 `packages/runtime-core/src/component.ts` 的 `applyOptions` 方法中：

代码块

```
function applyOptions(instance: any) {
    const {
        data: dataOptions,
        beforeCreate,
        created,
        beforeMount,
        mounted
    } = instance.type

    // hooks
    if (beforeCreate) {
        callHook(beforeCreate)
    }

    // 存在 data 选项时
    if (dataOptions) {
        ...
    }

    // hooks
    if (created) {
        callHook(created)
    }
}
```

2. 创建对应的 `callHook`：

代码块

```
/**
 * 触发 hooks
 */
function callHook(hook: Function) {
    hook()
}
```

至此， `beforeCreate` 和 `created` 完成。

接下来我们来去处理 `beforeMount` 和 `mounted`，对于这两个生命周期而言，他需要先注册，在触发。

那么首先我们先来处理注册的逻辑：

1. 首先我们需要先创建 `LifecycleHooks`

    1. 在 `packages/runtime-core/src/component.ts` 中：

索引目录

09：框架实现：组

```
/**
 * 生命周期钩子
 */
export const enum LifecycleHooks {
    BEFORE_CREATE = 'bc',
    CREATED = 'c',
    BEFORE_MOUNT = 'bm',
    MOUNTED = 'm'
}
```

2. 在生成组件实例时，提供对应的生命周期相关选项：

代码块

```
/**
 * 创建组件实例
 */
export function createComponentInstance(vnode) {
    const type = vnode.type

    const instance = {
        ...
+       // 生命周期相关
+       isMounted: false, // 是否挂载
+       bc: null, // beforeCreate
+       c: null, // created
+       bm: null, // beforeMount
+       m: null // mounted
    }

    return instance
}
```

3. 创建 `packages/runtime-core/src/apiLifecycle.ts` 模块，处理对应的 `hooks` 注册方法：

代码块

```
import { LifecycleHooks } from './component'

/**
 * 注册 hook
 */
export function injectHook(
    type: LifecycleHooks,
    hook: Function,
    target
): Function | undefined {
    // 将 hook 注册到 组件实例中
    if (target) {
        target[type] = hook
        return hook
    }
}

/**
 * 创建一个指定的 hook
 * @param lifecycle 指定的 hook enum
 * @returns 注册 hook 的方法
 */
export const createHook = (lifecycle: LifecycleHooks) => {
    return (hook, target) => injectHook(lifecycle, hook, target)
}

export const onBeforeMount = createHook(LifecycleHooks.BEFORE_MOUNT)
export const onMounted = createHook(LifecycleHooks.MOUNTED)
```

这样我们注册 `hooks` 的一些基础逻辑完成。

那么下面我们就可以 `applyOptions` 方法中，完成对应的注册：

代码块

```
function applyOptions(instance: any) {
    ...
    function registerLifecycleHook(register: Function, hook?: Function) {
        register(hook, instance)
    }

    // 注册 hooks
    registerLifecycleHook(onBeforeMount, beforeMount)
    registerLifecycleHook(onMounted, mounted)
}

```

将 `bm` 和 `m` 注册到组件实例之后，下面就可以在 `componentUpdateFn` 中触发对应 `hooks` 了：

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) {
        // 获取 hook
        const { bm, m } = instance

        // beforeMount hook
        if (bm) {
            bm()
        }

        // 从 render 中获取需要渲染的内容
        const subTree = (instance.subTree = renderComponentRoot(instance))

        // 通过 patch 对 subTree，进行打补丁。即：渲染组件
        patch(null, subTree, container, anchor)

        // mounted hook
        if (m) {
            m()
        }

        // 把组件根节点的 el，作为组件的 el
        initialVNode.el = subTree.el
    } else {
    }
}
```

至此，生命周期逻辑处理完成。

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

代码块

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

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

```
    },
    // 组件被挂载之前
    beforeMount() {
      alert('beforeMount')
    },
    // 组件被挂载之后
    mounted() {
      alert('mounted')
    },
  }

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

测试成功

08：源码阅读：组件生命周期回调处理逻辑 ◂ 上一节　　下一节 ▸ 10：源码阅读：生命回调钩子中访问响应性...

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

意见反馈　收藏教程　标记书签