全部开发者教程

06：响应性数据的编辑器处理：generate 生成 render 函数

07：响应性数据的编辑器处理：render 函数的执行处理

08：多层级模板的编辑器处理：多层级的处理逻辑

09：基于编辑器的指令(v-xx)处理：指令解析的整体逻辑

10：基于编辑器的指令(v-xx)处理：AST 解析逻辑（困难）

11：基于编辑器的指令(v-xx)处理：JavaScript AST，构建 vif 转化模块（困难）

12：基于编辑器的指令(v-xx)处理：JavaScript AST，transform 的转化逻辑

13：基于编辑器的指令(v-xx)处理：生成 render 函数

14：总结

Sunday • 更新于 2022-10-19

上一节 14：总结 02：基于 rende… 下一节

01：前言

到目前位置我们已经完成了：

1. 响应性
2. 运行时
3. 编辑器

三大模块。这三大模块基本上描述了 `vue` 的核心业务逻辑。

但是对于目前而言，这三大模块还是完全独立的系统。比如：如果想要渲染，那么必须要单独的导入 `render`。

如有大家有过 `vue 3` 的使用经验，那么我们知道，当我们构建一个 `vue 3` 实例时，可以这么做：

代码块

```
<script>
  const { createApp } = Vue

  const APP = {
    template: `<div>hello world</div>`
  }

  const app = createApp(APP)
  app.mount('#app')
</script>
```

那么这里就会涉及到两个方法：

1. `createApp`：创建 `app` 实例
2. `mount`：挂载

通过这两个方法，我们就可以直接关联上 **运行时 + 编译器**，直接实现模板的渲染。

咱们的框架目前还不支持这样的使用方式，所以本章我们就要去实现这个功能。

对于上述的功能我们需要分成两块来看：

1. 构建 `createApp` 通过 `render` 进行渲染：

代码块

```
<script>
  const { createApp, h } = Vue
  const APP = {
    render() {
      return h('div', 'hello world')
    }
  }

  const app = createApp(APP)
  app.mount('#app')
</script>
```

索引目录

01：前言

代码块

```
<script>
  const { createApp } = Vue

  const APP = {
    template: `<div>hello world</div>`
  }

  const app = createApp(APP)
  app.mount('#app')
</script>
```

那么明确好了以上内容之后，接下来我们就分别去进行对应的渲染。

14：总结 上一节　　下一节 02：基于 render 渲染的 createApp 的构建...

我要提出意见反馈