

Vue.js 3.0

源码组织方式

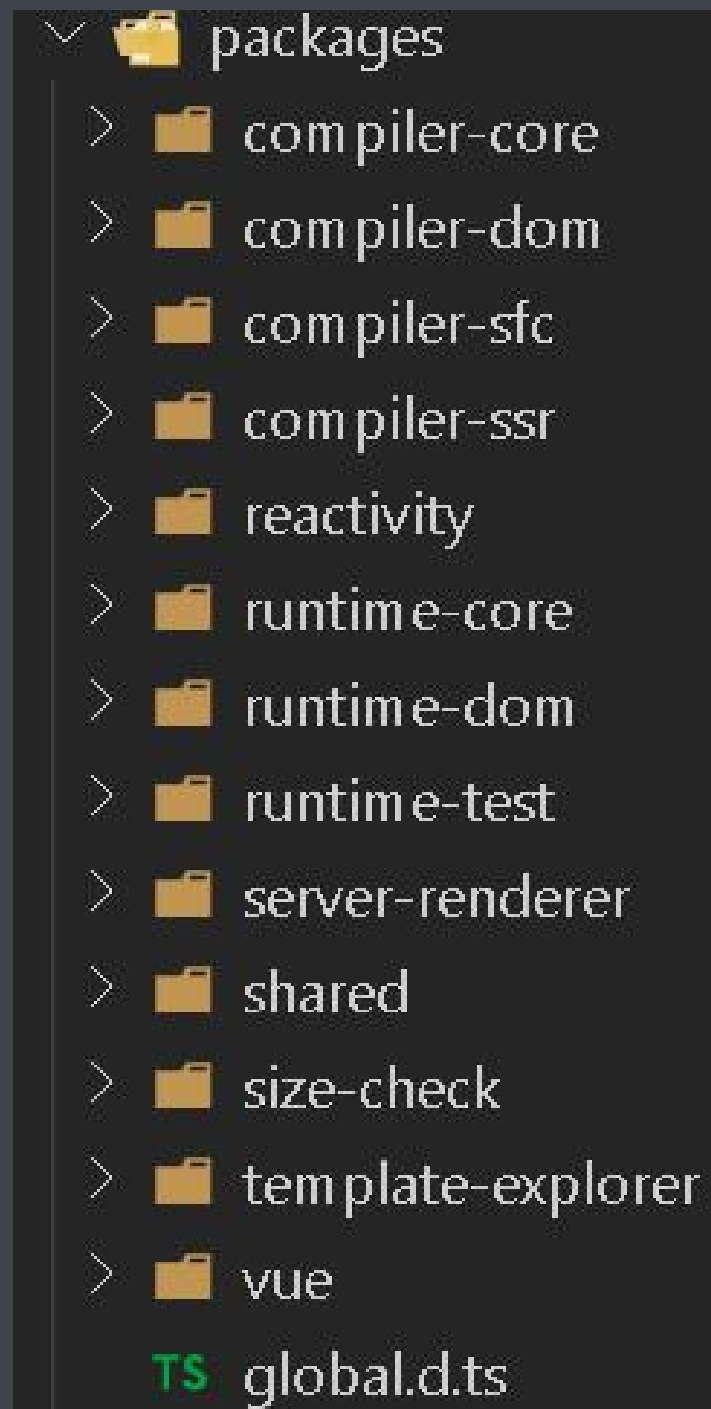
Vue.js 3.0

- 源码组织方式的变化
- Composition API
- 性能提升
- Vite

源码组织方式

- 源码采用 TypeScript 重写
- 使用 Monorepo 管理项目结构

packages 目录结构

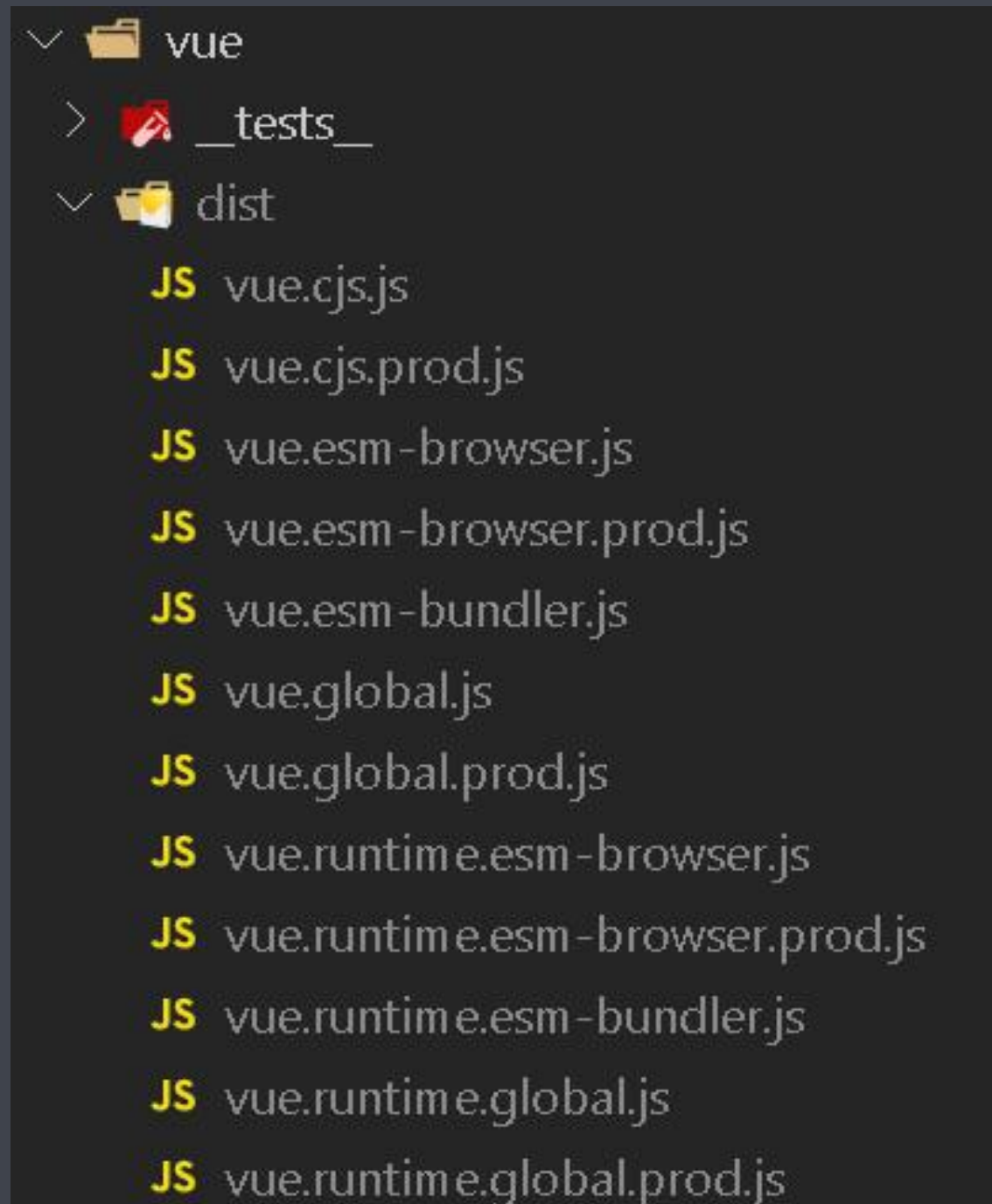


Vue.js 3.0

不同构建版本

构建版本

- packages/vue



构建版本

- cjs
 - vue.cjs.js
 - vue.cjs.prod.js
- global
 - vue.global.js
 - vue.global.prod.js
 - vue.runtime.global.js
 - vue.runtime.global.prod.js

构建版本

- browser
 - vue.esm-browser.js
 - vue.esm-browser.prod.js
 - vue.runtime.esm-browser.js
 - vue.runtime.esm-browser.prod.js
- bundler
 - vue.esm-bundler.js
 - vue.runtime.esm-bundler.js

Vue.js 3.0

Composition API

Composition API

- RFC (Request For Comments)
 - <https://github.com/vuejs/rfcs>
- Composition API RFC
 - <https://composition-api.vuejs.org>

设计动机

- Options API
 - 包含一个描述组件选项(data、methods、props等)的对象
 - Options API 开发复杂组件，同一个功能逻辑的代码被拆分到不同选项

Options API Demo

```
export default {
  data () {
    return {
      position: {
        x: 0,
        y: 0
      }
    }
  },
  created () {
    window.addEventListener('mousemove', this.handle)
  },
  destroyed () {
    window.removeEventListener('mousemove', this.handle)
  },
  methods: {
    handle (e) {
      this.position.x = e.pageX
      this.position.y = e.pageY
    }
  }
}
```

设计动机

- Composition API
 - Vue.js 3.0 新增的一组 API
 - 一组基于函数的 API
 - 可以更灵活的组织组件的逻辑

Composition API Demo

```
import { reactive, onMounted, onUnmounted } from 'vue'
function useMousePosition () {
  const position = reactive({
    x: 0,
    y: 0
  })
  const update = (e) => {
    position.x = e.pageX
    position.y = e.pageY
  }
  onMounted(() => {
    window.addEventListener('mousemove', update)
  })
  onUnmounted(() => {
    window.removeEventListener('mousemove', update)
  })
  return position
}

export default {
  setup () {
    const position = useMousePosition()
    return {
      position
    }
  }
}
```

Composition API

Vue.js 3.0

性能提升

性能提升

- 响应式系统升级
- 编译优化
- 源码体积的优化

响应式系统升级

- Vue.js 2.x 中响应式系统的核心 `defineProperty`
- Vue.js 3.0 中使用 `Proxy` 对象重写响应式系统
 - 可以监听动态新增的属性
 - 可以监听删除的属性
 - 可以监听数组的索引和 `length` 属性

编译优化



```
<template>
  <div id="app">
    <div>static root
      <div>static node</div>
    </div>
    <div>static node</div>
    <div>static node</div>
    <div>{{ count }}</div>
    <button @click="handler">button</button>
  </div>
</template>
```

编译优化

- Vue.js 2.x 中通过标记静态根节点，优化 diff 的过程
- Vue.js 3.0 中标记和提升所有的静态根节点，diff 的时候只需要对比动态节点内容
 - Fragments (升级 vetur 插件)
 - 静态提升
 - Patch flag
 - 缓存事件处理函数

优化打包体积

- Vue.js 3.0 中移除了一些不常用的 API
 - 例如: inline-template、filter 等
- Tree-shaking

Vue.js 3.0

Vite

ES Module

- 现代浏览器都支持 ES Module (IE 不支持)
- 通过下面的方式加载模块
 - `<script type="module" src="..." ></script>`
- 支持模块的 `script` 默认延迟加载
 - 类似于 `script` 标签设置 `defer`
 - 在文档解析完成后, 触发 `DOMContentLoaded` 事件前执行

Vue.js 3.0

Vite

Vite as Vue-CLI

- Vite 在开发模式下不需要打包可以直接运行
- Vue-CLI 开发模式下必须对项目打包才可以运行

Vite 特点

- 快速冷启动
- 按需编译
- 模块热更新

Vite as Vue-CLI

- Vite 在生产环境下使用 Rollup 打包
 - 基于 ES Module 的方式打包
- Vue-CLI 使用 Webpack 打包

Vite 创建项目

- Vite 创建项目



```
$ npm init vite-app <project-name>  
$ cd <project-name>  
$ npm install  
$ npm run dev
```

- 基于模板创建项目



```
$ npm init vite-app --template react  
$ npm init vite-app --template preact
```

Composition API

Composition API

生命周期钩子函数

生命周期钩子函数

Options API	Hook inside <code>setup</code>
<code>beforeCreate</code>	Not needed*
<code>created</code>	Not needed*
<code>beforeMount</code>	<code>onBeforeMount</code>
<code>mounted</code>	<code>onMounted</code>
<code>beforeUpdate</code>	<code>onBeforeUpdate</code>
<code>updated</code>	<code>onUpdated</code>
<code>beforeUnmount</code>	<code>onBeforeUnmount</code>
<code>unmounted</code>	<code>onUnmounted</code>
<code>errorCaptured</code>	<code>onErrorCaptured</code>
<code>renderTracked</code>	<code>onRenderTracked</code>
<code>renderTriggered</code>	<code>onRenderTriggered</code>

Composition API

reactive/toRefs/ref

Computed

Computed

- 第一种用法

- `watch(() => count.value + 1)`

- 第二种用法

```
const count = ref(1)
const plusOne = computed({
  get: () => count.value + 1,
  set: val => {
    count.value = val - 1
  }
})
```

Watch

Watch

- Watch 的三个参数
 - 第一个参数：要监听的数据
 - 第二个参数：监听到数据变化后执行的函数，这个函数有两个参数分别是新值和旧值
 - 第三个参数：选项对象，deep 和 immediate
- Watch 的返回值
 - 取消监听的函数

WatchEffect

WatchEffect

- 是 watch 函数的简化版本，也用来监视数据的变化
- 接收一个函数作为参数，监听函数内响应式数据的变化

ToDoList

功能演示

ToDoList 功能列表

- 添加待办事项
- 删除待办事项
- 编辑待办事项
- 切换待办事项
- 存储待办事项

ToDoList

项目结构

ToDoList

添加待办事项

ToDoList

删除待办事项

ToDoList

编辑待办事项

编辑待办事项

- 双击待办项，展示编辑文本框
- 按回车或者编辑文本框失去焦点，修改数据
- 按esc取消编辑
- 把编辑文本框清空按回车，删除这一项
- 显示编辑文本框的时候获取焦点

ToDoList

编辑待办事项

ToDoList

编辑待办事项

自定义指令

- Vue 2.x

```
Vue.directive('editingFocus', {  
  bind(el, binding, vnode, prevVnode) {},  
  inserted() {},  
  update() {}, // remove  
  componentUpdated() {},  
  unbind() {}  
})
```

- Vue 3.0

```
app.directive('editingFocus', {  
  beforeMount(el, binding, vnode, prevVnode) {},  
  mounted() {},  
  beforeUpdate() {}, // new  
  updated() {},  
  beforeUnmount() {}, // new  
  unmounted() {}  
})
```


自定义指令

- Vue 2.x



```
Vue.directive('editingFocus', (el, binding) => {  
  binding.value && el.focus()  
})
```

- Vue 3.0



```
app.directive('editingFocus', (el, binding) => {  
  binding.value && el.focus()  
})
```

ToDoList

切换待办事项状态

切换待办事项状态

- 点击 checkbox, 改变所有待办项状态
- All/Active/Completed
- 其它
 - 显示未完成待办项个数
 - 移除所有完成的项目
 - 如果没有待办项, 隐藏 main 和 footer

ToDoList

切换待办事项状态

ToDoList

切换待办事项状态

ToDoList

切换待办事项状态

ToDoList

本地存储

响应式原理

Vue.js 响应式回顾

- Proxy 对象实现属性监听
- 多层属性嵌套，在访问属性过程中处理下一级属性
- 默认监听动态添加的属性
- 默认监听属性的删除操作
- 默认监听数组索引和 length 属性
- 可以作为单独的模块使用

核心方法

- reactive/ref/toRefs/computed
- effect
- track
- trigger

响应式原理

Proxy 对象回顾

响应式原理

reactive

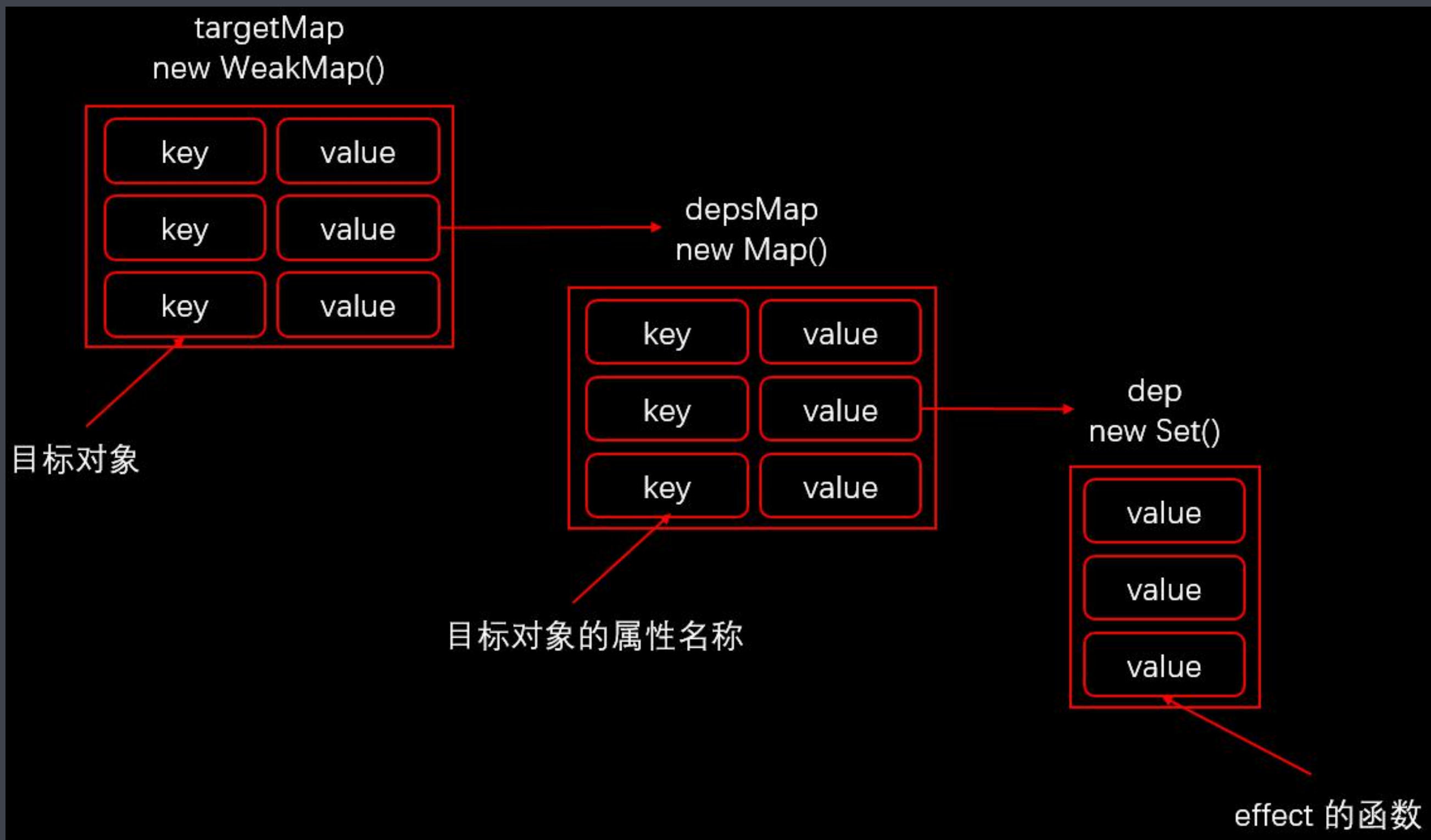
reactive

- 接收一个参数，判断这参数是否是对象
- 创建拦截器对象 handler，设置 get/set/deleteProperty
- 返回 Proxy 对象

响应式原理

收集依赖

收集依赖



响应式原理

effect && track

响应式原理

trigger

响应式原理

ref

reactive vs ref

- ref 可以把基本数据类型数据，转成响应式对象
- ref 返回的对象，重新赋值成对象也是响应式的
- reactive 返回的对象，重新赋值丢失响应式
- reactive 返回的对象不可以解构

reactive vs ref

- reactive

```
const product = reactive({  
  name: 'iPhone',  
  price: 5000,  
  count: 3  
})
```

- ref

```
const price = ref(5000)  
const count = ref(3)
```

响应式原理

toRefs

响应式原理

computed

响应式原理

computed

Vite

Vite 概念

- Vite 是一个面向现代浏览器的一个更轻、更快的 Web 应用开发工具
- 它基于 ECMAScript 标准原生模块系统（ES Modules）实现

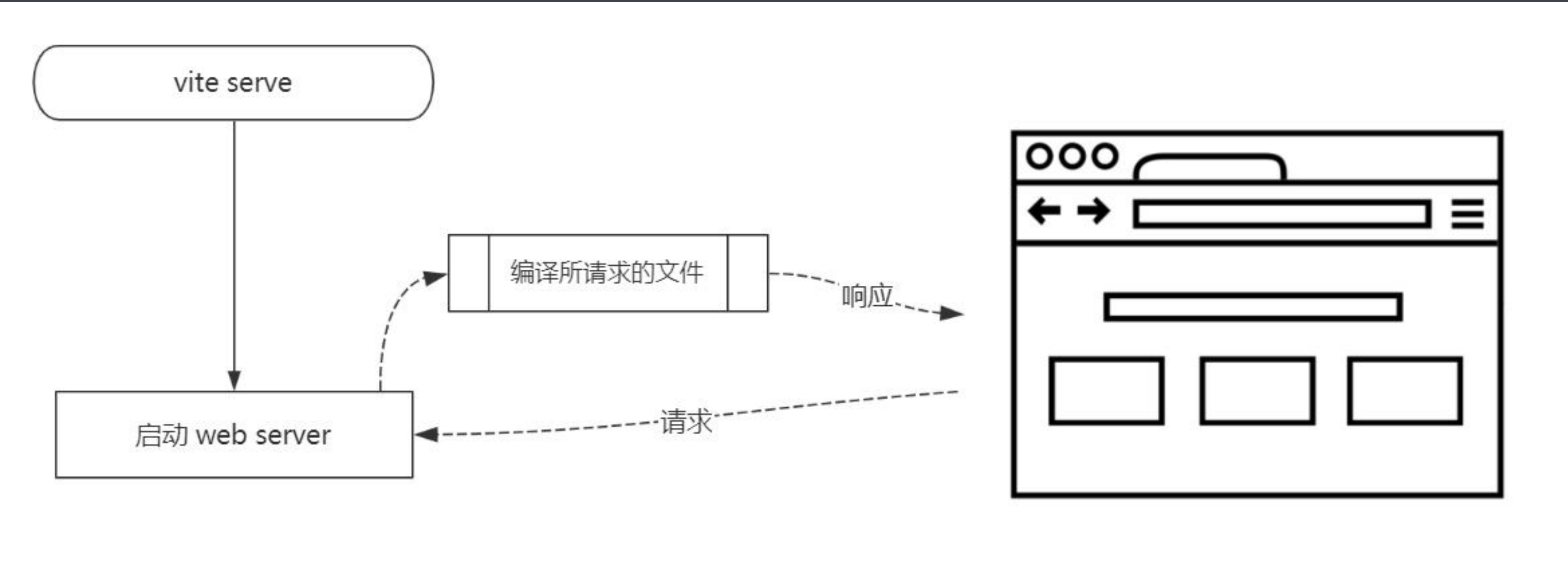
Vite 项目依赖

- Vite
- @vue/compiler-sfc

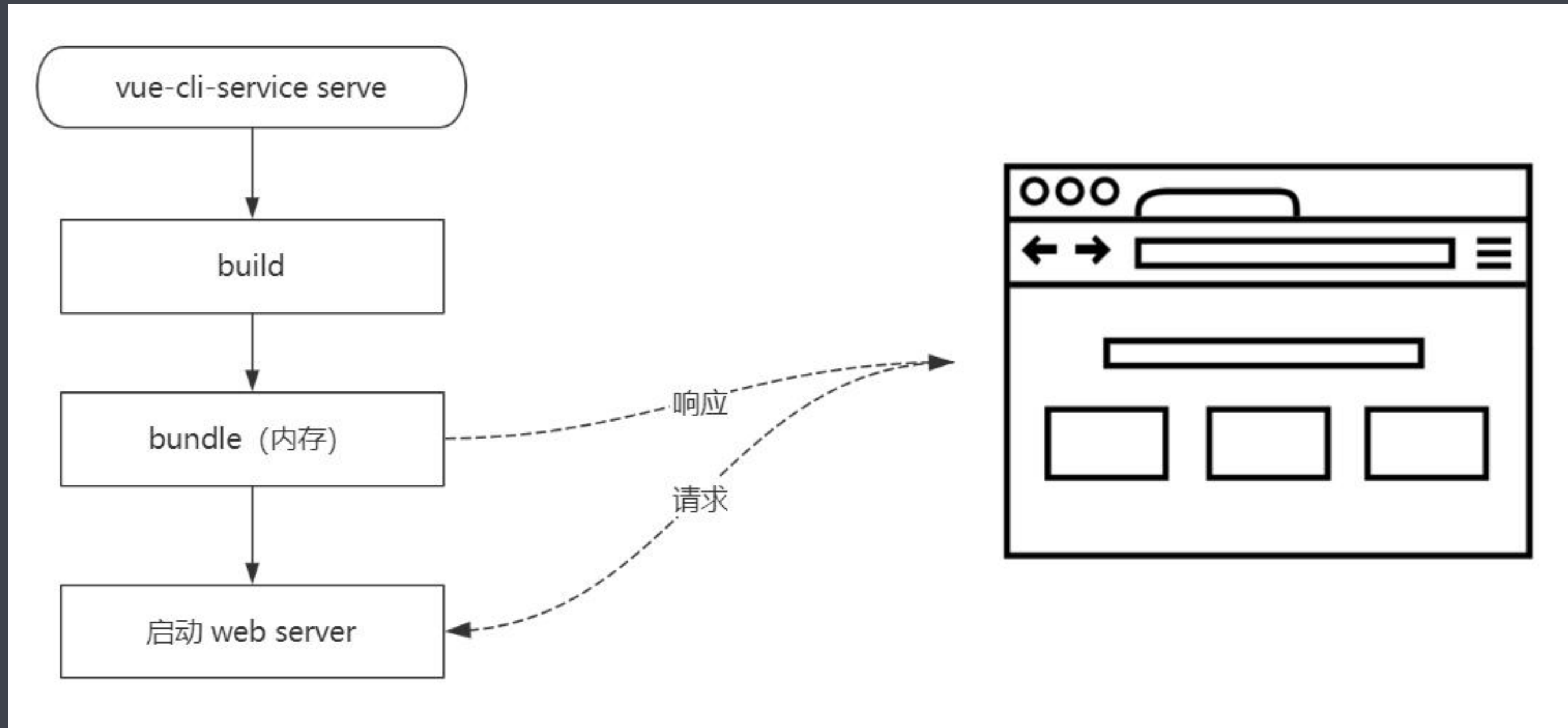
基础使用

- vite serve
- vite build

vite serve



vue-cli-service serve



HMR

- Vite HMR
 - 立即编译当前所修改的文件
- Webpack HMR
 - 会自动以这个文件为入口重写 build 一次，所有的涉及到的依赖也都会被加载一遍

Build

- vite build
 - Rollup
 - [Dynamic import](#)
 - [Polyfill](#)

打包 or 不打包

- 使用 Webpack 打包的两个原因：
 - 浏览器环境并不支持模块化
 - 零散的模块文件会产生大量的 HTTP 请求

浏览器对 ES Module 的支持



开箱即用

- TypeScript – 内置支持
- less/sass/stylus/postcss – 内置支持（需要单独安装）
- JSX
- Web Assembly

Vite 特性

- 快速冷启动
- 模块热更新
- 按需编译
- 开箱即用

Vite 实现原理

静态 Web 服务器

Vite 核心功能

- 静态 Web 服务器
- 编译单文件组件
 - 拦截浏览器不识别的模块，并处理
- HMR

Vite 实现原理

修改第三方模块的路径

Vite 实现原理

加载第三方模块

Vite 实现原理

编译单文件组件



扫码联系老师

技能评估、福利资料、课程优惠

Made with ❤️ by LagouFed