05 工具

目录(Catalog)

生词(New Words)

- **shallow ['ʃæləʊ] --adj.浅的, 肤浅的 --n.浅滩**
  - Tony seemed very shallow and immature. 托尼看起来好像很肤浅, 不够成熟.
- **expect [ɪk'spekt]** vt.期望, 期待, 预期**

内容(Content)

5.1 单文件组件

5.1.1 介绍

- 在很多 Vue 项目中, 我们使用 `Vue.component` 来定义全局组件, 紧接着用 `new Vue({ el: '#container '})` 在每个页面内指定一个容器元素.

  这种方式在很多中小规模的项目中运作的很好, 在这些项目里 JavaScript 只被用来加强特定的视图. 但当在更复杂的项目中, 或者你的前端完全由 JavaScript 驱动的时候, 下面这些缺点将变得非常明显:

  - **全局定义 (Global definitions)** 强制要求每个 component 中的命名不得重复;
  - **字符串模板 (String templates)** 缺乏语法高亮, 在 HTML 有多行的时候, 需要用到丑陋的 `\`;
  - **不支持 CSS (No CSS support)** 意味着当 HTML 和 JavaScript 组件化时, CSS 明显被遗漏;
  - **没有构建步骤 (No build step)** 限制只能使用 HTML 和 ES5 JavaScript, 而不能使用预处理器, 如 Pug (formerly Jade) 和 Babel.

  文件扩展名为 `.vue` 的 **single-file components(单文件组件)** 为以上所有问题提供了解决方法, 并且还可以使用 webpack 或 Browserify 等构建工具.

  这是一个文件名为 `Hello.vue` 的简单实例: 示例代码 略...

  单文件组件的示例 (点击查看文本版的代码

  现在我们获得:

  - 完整语法高亮
  - CommonJS 模块
  - 组件作用域的 CSS

  正如我们说过的, 我们可以使用预处理器来构建简洁和功能更丰富的组件, 比如 Pug, Babel (with ES2015 modules), 和 Stylus.

  带预处理器的单文件组件的示例(点击查看文本版的代码)

  这些特定的语言只是例子, 你可以只是简单地使用 Babel, TypeScript, SCSS, PostCSS - 或者其他任何能够帮助你提高生产力的预处理器. 如果搭配 vue-loader 使用 webpack, 它也能为 CSS Modules 提供头等支持.

5.1.1.2 怎么看待关注点分离?

- 一个重要的事情值得注意, 关注点分离不等于文件类型分离. 在现代 UI 开发中, 我们已经发现相比于把代码库分离成三个大的层次并将其相互交织起来, 把它们划分为松散耦合的组件再将其组合起来更合理一些. 在一个组件里, 其模板、逻辑和样式是内部耦合的, 并且把他们搭配在一起实际上使得组件更加内聚且更可维护.

  即便你不喜欢单文件组件, 你仍然可以把 JavaScript、CSS 分离成独立的文件然后做到热重载和预编译.

  ```
  <!-- my-component.vue -->
  <template>
      <div>This will be pre-compiled</div>
  </template>
  <script src="./my-component.js"></script>
  <style src="./my-component.css"></style>
  ```

5.1.2 起步

5.1.2.1 例子沙箱

- 如果你希望深入了解并开始使用单文件组件, 请来 CodeSandbox [看看这个简单的 todo 应用]https://codesandbox.io/s/o29j95wx9().

  TIP: 此处给出另外一个版本的 TodoList 应用的示例:

  ```
  
  ```

5.1.2.2 针对刚接触 JavaScript 模块开发系统的用户

- 有了 .vue 组件, 我们就进入了高级 JavaScript 应用领域. 如果你没有准备好的话, 意味着还需要学会使用一些附加的工具:
  - Node Package Manager (NPM): 阅读 Getting Started guide 中关于如何从注册地 (registry) 获取包的章节.
  - Modern JavaScript with ES2015/16: 阅读 Babel 的 Learn ES2015 guide. 你不需要立刻记住每一个方法, 但是你可以保留这个页面以便后期参考.

  在你花一天时间了解这些资源之后, 我们建议你参考 Vue CLI 3. 只要遵循指示, 你就能很快地运行一个带有 .vue 组件、ES2015、webpack 和热重载 (hot-reloading)的 Vue 项目!

5.1.2.3 针对高级用户

- CLI 会为你搞定大多数工具的配置问题, 同时也支持细粒度自定义配置项.

  有时你会想从零搭建你自己的构建工具, 这时你需要通过 Vue Loader 手动配置 webpack. 关于学习更多 webpack 的内容, 请查阅其官方文档和 Webpack Academy.

5.2 单元测试

- Vue CLI 拥有开箱即用的通过 Jest 或 Mocha 进行单元测试的内置选项. 我们还有官方的 Vue Test Utils 提供更多详细的指引和自定义设置.

5.2.1 简单的断言

- 你不必为了可测试性在组件中做任何特殊的操作, 导出原始设置就可以了:

```
<template>
    <span>{{ message }}</span>
</template>

<script>
    export default {
        data() {
            return {
                message: 'hello!'
            }
        },
        created() {
            this.message = 'bye!'
        }
    }
</script>
```

然后随着 **Vue Test Utils** 导入组件, 你可以使用许多常见的断言(这里我们使用的是 Jest 风格的 `expect` 断言作为示例):

```
// - 导入 Vue Test Utils 内的 `shallowMount` 和待测试的组件
import {shallowMount} from '@vue/test-utils';
import MyComponent from './MyComponent.vue';

// - 挂载这个组件
const wrapper = shallowMount(MyComponent);

// - 这里是一些 Jest 的测试, 你也可以使用你喜欢的任何断言库或测试
describe('MyComponent', () => {
    // - 检查原始组件选项
    it('has a created hook(有一个 created 钩子)', () => {
        expect(typeof MyComponent.created).toBe('function');
    })

    // - 评估原始组件选项中的函数的结果
    it('sets the correct default data(设置正确的默认 data)', () => {
        expect(typeof MyComponent.data).toBe('function');
        const defaultData = MyComponent.data();
        expect(defaultData.message).toBe('hello!');
    })

    // - 检查 mount 中的组件实例
    it('correctly sets the message when created', () => {
        expect(wrapper, vm.$data.message).toBe('bye!')
    })

    // - 创建一个实例并检查渲染输出
    it('renders the correct message', () => {
```

```
            expect(wrapper.text()).toBe('bye!')
        })
    })
```

5.2.2 编写可被测试的组件

- 很多组件的渲染输出由它的 `props` 决定. 事实上, 如果一个组件的渲染输出完全取决于它的 `props` 的话, 那么它会让测试变得简单, 就好像断言不同参数的纯函数的返回值. 看下面这个例子:

```
<template>
    <p>{{ msg }}</p>
</template>
<script>
    export default {
        props: ['msg']
    }
</script>
```

你可以使用 Vue Test Utils 来在输入不同 prop 时为渲染输出下断言:

```
import {shallowMount} from '@vue/test-utils';
import MyComponent from './MyComponent.vue';

// - 挂载元素并返回已渲染的组件的工具函数
function getMountedComponent(Component, propsData) {
    return shallowMount(Component, {
        propsData
    })
}

describe('MyComponent', () => {
    it('renders correctly with different props', () => {
        expect(
            getMountedComponent(MyComponent, {
                msg: 'Hello'
            }).text()
        ).toBe('Hello')
    })
    expect(
        getMountedComponent(MyComponent, {
            msg: 'Bye'
        }).text()
    ).toBe('Bye')
})
```

5.2.3 断言异步更新

- 由于 Vue 进行 **异步更新 DOM** 的情况, 一些依赖 DOM 更新结果的断言必须在 `vm.$nextTick()` resolve 之后进行:

```
// - 在状态更新后检查生成的 HTML
it('updates the rendered message when wrapper.message', async() => {
    const wrapper = shallowMount(MyComponent);
    wrapper.setData({message: 'foo'});

    // - 在状态改变后和断言 DOM 更新前等待一刻
    await wrapper.vm.$nextTick()
    except(wrapper.test().toBe('foo'))
})
```

关于更深入的 Vue 单元测试的内容, 请移步 Vue Test Utils 以及我们关于 Vue 组件的单元测试 的 Cookbook 文章.

5.3 `TypeScript` 支持

- 暂略. Vue 2.x 对 TS 的支持并不友好, 将来 Vue 3.0 普及后会提供对 TS 更好的支持.

5.3.1 发布为 NPM 包的官方声明文件

5.3.2 推荐配置

5.3.3 开发工具链

5.3.3.1 工程创建

5.3.3.2 编辑器支持

5.3.4 基本用法

5.3.5 基于类的 Vue 组件

5.3.6 增强类型以配合插件使用

5.3.7 标注返回值

5.4 生产环境部署

- > 以下大多数内容在你使用 Vue CLI 时都是默认开启的. 该章节仅跟你自定义的构建设置有关.

5.4.1 开启生产环境模式

- 开发环境(development)下, Vue 会提供很多警告来帮你对付常见的错误与陷阱. 而在生产环境(production)下, 这些警告语句却没有用, 反而会增加应用的体积. 此外, 有些警告检查还有一些小的运行时开销, 这在生产环境模式下是可以避免的.

5.4.1.1 不适用构建工具

- 如果用 Vue 完整独立版本, 即直接用 `<script>` 元素引入 Vue 而不提前进行构建, 请记得在生产环境下使用压缩后的版本 (vue.min.js). 两种版本都可以在 安装指导 中找到.

5.4.1.2 使用构建工具

- 当使用 webpack 或 Browserify 类似的构建工具时, Vue 源码会根据 `process.env.NODE_ENV` 决定是否启用生产环境模式, 默认情况为开发环境模式. 在 webpack 与 Browserify 中都有方法来覆盖此变量, 以启用 Vue 的生产环境模式, 同时在构建过程中警告语句也会被压缩工具去除. 所有这些在 `vue-cli` 模板中都预先配置好了, 但了解一下怎样配置会更好.

- webpack

  在 webpack 4+ 中, 你可以使用 `mode` 选项:

  ```
  module.exports = {
      mode: 'production'
  }
  ```

  但是在 webpack 3 及其更低版本中, 你需要使用 **DefinePlugin**:

  ```
  var webpack = require('webpack');
  module.exports = {
      // ...
      plugins: [
          // ...
          new webpack.DefinePlugin({
              'process.env.NODE_ENV': JSON.stringify('production');
          })
      ]
  }
  ```

- Browserify

  在运行打包命令时将 `NODE_ENV` 设置为 `"production"`. 这等于告诉 `vueify` 避免引入热重载和开发相关的代码.

  对打包后的文件进行一次全局的 **envify** 转换. 这使得压缩工具能清除掉 Vue 源码中所有用环境变量条件包裹起来的警告语句. 例如:

  ```
  NODE_ENV = production browserify -g envify -e main.js | uglifyjs -c -m > build.js
  ```

  或者在 Gulp 中使用 `envify`:

  ```
  // 使用 envify 的自定义模块来定制环境变量
  var envify = require('envify/custom')

  browserify(browserifyOptions)
  .transform(vueify)
  .transform(
      // 必填项, 以处理 node_modules 里的文件
      { global: true },
      envify({ NODE_ENV: 'production' })
  )
  .bundle()
  ```

  或者配合 Grunt 和 **grunt-browserify** 使用 envify:

```
// 使用 envify 自定义模块指定环境变量
var envify = require('envify/custom')

browserify: {
dist: {
    options: {
        // 该函数用来调整 grunt-browserify 的默认指令
        configure: b => b
        .transform('vueify')
        .transform(
            // 用来处理 `node_modules` 文件
        { global: true },
        envify({ NODE_ENV: 'production' })
        )
        .bundle()
    }
}
}
```

- Rollup

  使用 @rollup/plugin-replace：

```
const replace = require('@rollup/plugin-replace')
rollup({
// ...
plugins: [
    replace({
    'process.env.NODE_ENV': JSON.stringify( 'production' )
    })
]
}).then(...)
```

5.4.2 模板编译器

- 当使用 DOM 内模板或 JavaScript 内的字符串模板时，模板会在运行时被编译为渲染函数. 通常情况下这个过程已经足够快了，但对性能敏感的应用还是最好避免这种用法.

  预编译模板最简单的方式就是使用 单文件组件 —— 相关的构建设置会自动把预编译处理好，所以构建好的代码已经包含了编译出来的渲染函数而不是原始的模板字符串.

  如果你使用 webpack，并且喜欢分离 JavaScript 和模板文件，你可以使用 vue-template-loader，它也可以在构建过程中把模板文件转换成为 JavaScript 渲染函数.

5.4.3 提取组件的 CSS

- 当使用单文件组件时，组件内的 CSS 会以 `<style>` 标签的方式通过 JavaScript 动态注入. 这有一些小小的运行时开销，如果你使用服务端渲染，这会导致一段 "无样式内容闪烁 (fouc)". 将所有组件的 CSS 提取到同一个文件可以避免这个问题，也会让 CSS 更好地进行压缩和缓存.

  查阅这个构建工具各自的文档来了解更多：

  - [webpack] + [vue-loader] (`vue-cli` 的 webpack 模板已经预先配置好)

    - [Browserify] + [vueify]
    - [Rollup] + [rollup-plugin-vue]

5.4.4 跟踪运行时错误

- 如果在组件渲染时出现运行错误, 错误将会被传递至全局 `Vue.config.errorHandler` 配置函数 (如果已设置). 利用这个钩子函数来配合错误跟踪服务是个不错的主意. 比如 [Sentry], 它为 Vue 提供了 官方集成 .