第 3 章

◂ Webpack+Vue.js开发准备 ▸

所有的 Vue.js 项目都是在 Webpack 的框架下进行开发的。可以说 vue-cli 直接把 Webpack 做了集成。在开发时，一边享受着飞一般的开发速度，一边体验着 Webpack 带来的便利。

本章将介绍如何使用 Webpack+Vue.js 进行开发的基本知识。

3.1 学习过程

在学习任何一种框架的时候，都是按照循序渐进的顺序来的。

（1）安装。
（2）新建一个页面。
（3）做一些简单变量的渲染。
（4）实现页面的跳转（路由）。
（5）实现页面间的参数传递（路由）。
（6）实现真实的 http 请求（访问接口）。
（7）提交表单。
（8）使用一些技巧让代码层次化（组件）。

按照笔者之前在惠普、联通、移动等公司的讲课经验，只需要一天的时间就可以把本章内容学会，并且上手开发 Vue.js 项目。

3.1.1 可以跳过的章节

对于有一定 Node 基础的读者，可以跳过“NVM 的安装”。对于使用 Linux/Mac 的读者，可以跳过“Git Bash 的安装”。

3.1.2 简写说明

由于本章是 Webpack + Vue.js 下的实战开发，所以统一使用 Vue.js 来代替冗长的 Webpack + Vue.js。

例如，在 Vue.js 中创建页面需要以下两步。

（1）新建路由。

（2）新建 vue 页面。

3.1.3　本书例子文件下载

本章中的所有例子，由于都需要与 Webpack 相结合，因此笔者将其单独做成一个项目：https://github.com/sg552/vue_js_lesson_demo，读者可以下载后直接运行。

```
$ git clone https://github.com/sg552/vue_js_lesson_demo.git
$ npm install -v
$ npm run dev
```

可以在 http://localhost:8080/#/中看到效果，如图 3-1 所示。

图 3-1　页面效果

3.2　NVM、NPM 与 Node

NVM（Node Version Manager）是非常好用的 Node 版本管理器。这个技术出现的原因，是由于不同的项目 node 版本也不同，有的是 5.0.1，有的是 6.3.2。如果 node 版本不对，运行某个应用时，很可能就会遇到各种莫名其妙的问题。

因此，我们需要在同一台机器上同时安装多个版本的 Node。NVM 应用而生，很好地帮我们解决了这个问题。

Linux/Mac 下的 NVM 官方网址：https://github.com/creationix/nvm。

Windows 下的 NVM 官方网址：https://github.com/coreybutler/nvm-windows。

NPM（Node Package Manager）只要安装了 node，就会捆绑安装该命令。它的作用与

Ruby 中的 bundler 及 Java 中的 maven 相同，都是对第三方依赖进行管理的。

3.2.1 Windows 下的安装

步骤 01 使用浏览器打开 https://github.com/coreybutler/nvm-windows/releases，如图 3-2 所示。

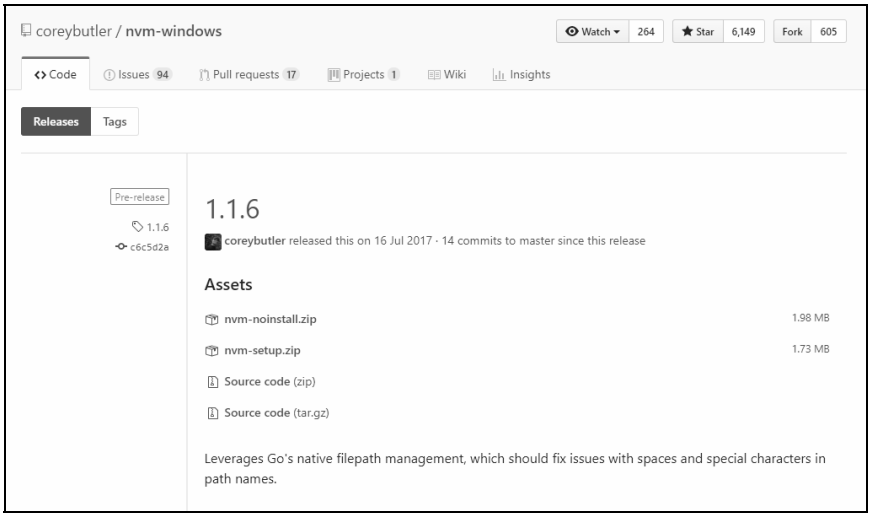

图 3-2 打开下载网址

步骤 02 单击最新的 release 版本进行下载“1.1.6 中的 nvm-setup.zip”文件。

步骤 03 解压下载文件，双击其中的 nvm-setup.exe 文件，就可以开始安装了，如图 3-3 所示。

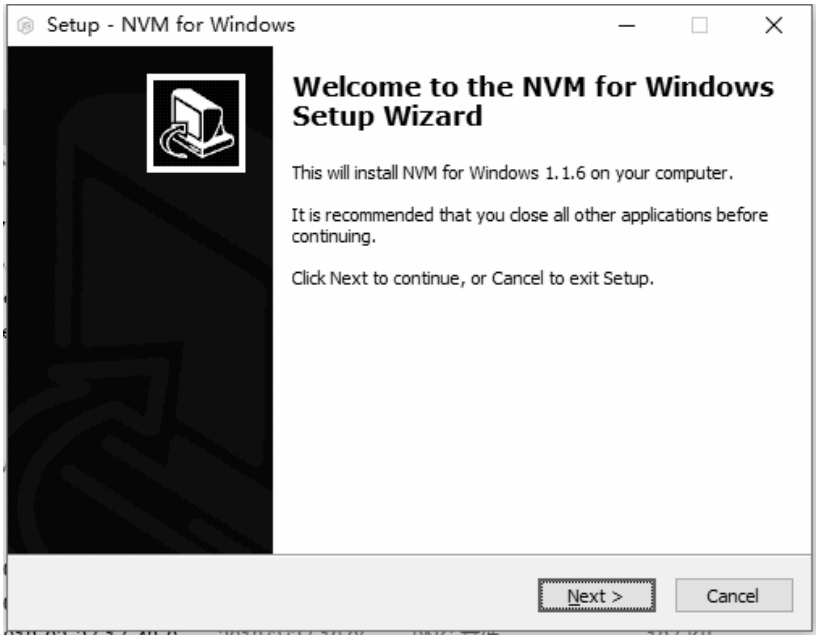

图 3-3 开始安装

步骤 04 单击 Next 按钮，在打开的对话框中选中 I accept the agreement 单选按钮，如图 3-4 所示。

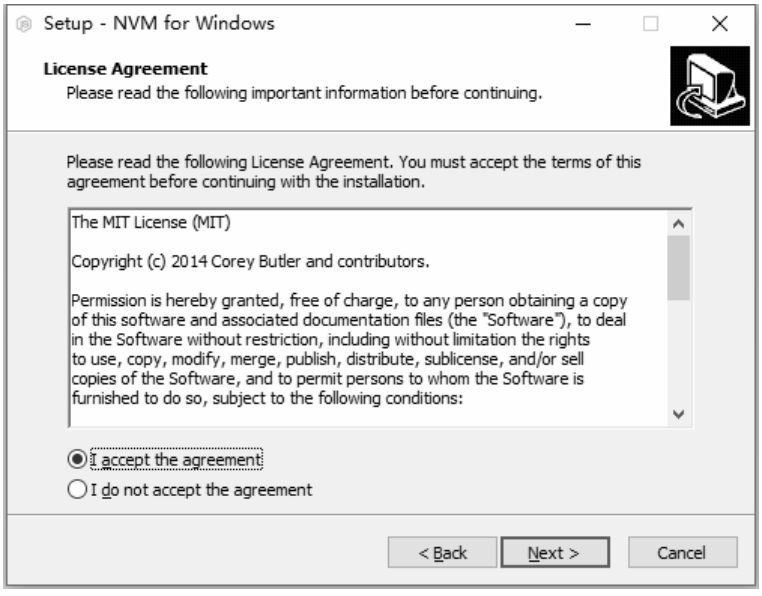

图 3-4　选择接受

步骤 05 继续单击 Next 按钮，在打开的对话框中选择安装路径。这里选择安装到 D:\nvm，如图 3-5 所示。

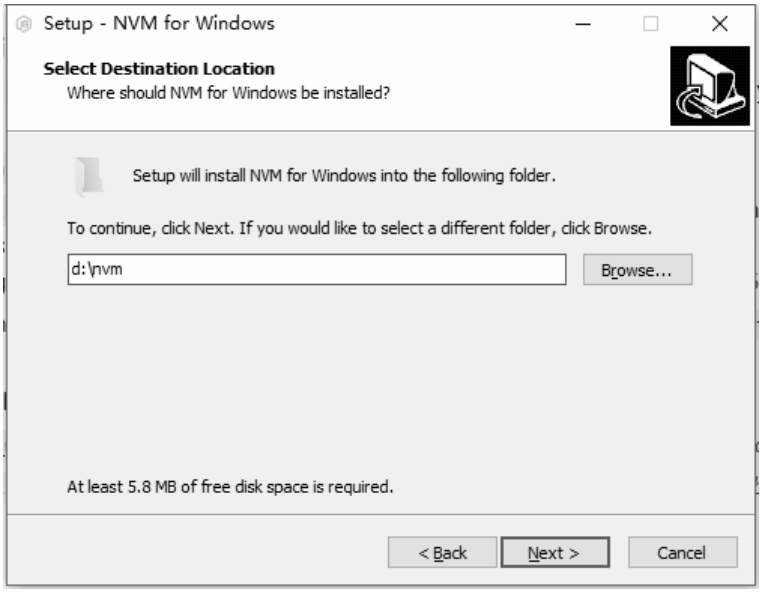

图 3-5　选择安装路径

步骤 06 继续单击 Next 按钮，在打开的对话框中询问把 nvm 的快捷方式放在哪里（symlink 的作用同快捷方式，允许我们在任意路径下都可以调用 nvm 命令），不用修改，直

接单击 Next 按钮，如图 3-6 所示。

图 3-6　默认快捷方式位置

步骤 07 然后弹出确认安装对话框，继续单击 Next 按钮就可以了。

步骤 08 最后设置环境变量。

```
NVM_HOME        D:\nvm
NVM_SYMLINK     C:\Program Files\nodejs
```

从控制面板中进入到“所有控制面板项目”→“高级系统配置”→“环境变量”，如图 3-7 所示。

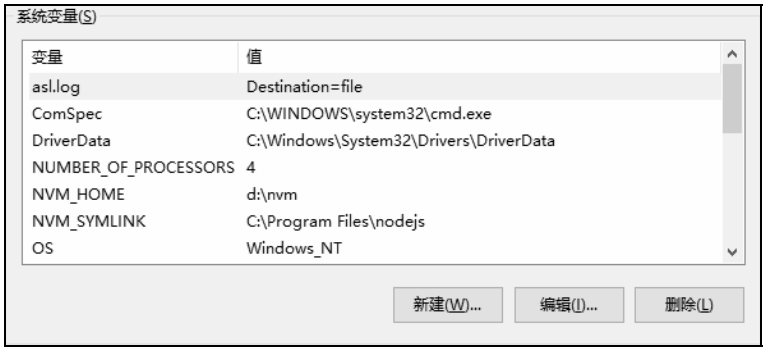

图 3-7　系统变量

对 PATH 的修改则是在原有值的基础上添加%NVM_HOME%、%NVM_SYMLINK%，如图 3-8 所示。

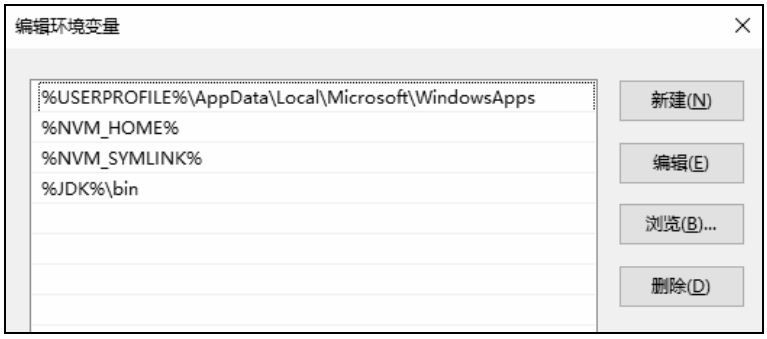

图 3-8　编辑环境变量

3.2.2　Linux、Mac 下的安装

（1）下载 nvm 的源代码，运行下面命令即可。

```
$ git clone https://github.com/creationix/nvm.git ~/.nvm && cd ~/.nvm && git
checkout `git describe --abbrev=0 --tags`
```

（2）Linux、Mac 的用户：为脚本设置启动时加载（对于使用 Windows 的读者，可以直接跳过第二步，到官方网站下载 exe 安装文件就可以了）。

把下面的代码放到~/.bashrc 或~/.bash_profile 或~/.zshrc 中。

```
$ source ~/.nvm/nvm.sh
```

3.2.3　运行

不能使用$ which nvm 验证安装是否成功，因为即使成功了，也不会返回结果。直接在命令行输入以下代码即可。

```
$ nvm
```

如果安装成功，就会看到以下英文。

```
Running version 1.1.6.


Usage:


  nvm arch                    : Show if node is running in 32 or 64 bit mode.
  nvm install <version> [arch] : The version can be a node.js version or
"latest" for the latest stable version....
  nvm list [available]        : List the node.js installations. Type
```

```
"available" at the end to see what can be ...
  nvm on                    : Enable node.js version management.
```

3.2.4 使用 NVM 安装或管理 node 版本

（1）列出所有可以安装的 node 版本。

Windows 下的命令：

```
$ nvm list available
```

Linux/Mac 下的命令：

```
$ nvm list-remote
```

可以看到安装的所有版本。下面是 Windows 中的例子，Linux、Mac 类似。

```
$ nvm list available

|   CURRENT    |    LTS     |  OLD STABLE  | OLD UNSTABLE |
|--------------|------------|--------------|--------------|
|    10.5.0    |   8.11.3   |   0.12.18    |   0.11.16    |
|    10.4.1    |   8.11.2   |   0.12.17    |   0.11.15    |
|    10.4.0    |   8.11.1   |   0.12.16    |   0.11.14    |
|    10.3.0    |   8.11.0   |   0.12.15    |   0.11.13    |
|    10.2.1    |   8.10.0   |   0.12.14    |   0.11.12    |
|    10.2.0    |   8.9.4    |   0.12.13    |   0.11.11    |
|    10.1.0    |   8.9.3    |   0.12.12    |   0.11.10    |
|    10.0.0    |   8.9.2    |   0.12.11    |    0.11.9    |
|    9.11.2    |   8.9.1    |   0.12.10    |    0.11.8    |

This is a partial list. For a complete list, visit
https://nodejs.org/download/release
```

（2）列出本地安装好的版本。

```
$ nvm list
```

结果形如：

```
$ nvm list
```

```
  * 10.5.0 (Currently using 64-bit executable)
    6.9.1
```

在上面的结果中，表示当前系统安装了两个 node 版本：6.9.1 和 10.5.0。默认的 node 版本是 10.5.0。

（3）安装 node。

选择一个版本号就可以安装了。

```
$ nvm install 10.5.0
```

安装好之后，退出命令行并重新进入即可。

（4）使用 node。

下面的命令是为当前文件夹指定 node 的版本。

```
$ nvm use 10.5.0
```

对于 Linux、Mac，如果希望为系统全局使用某个版本，就可以运行下面的命令。

```
$ nvm alias default 10.5.0
```

在 Linux、Mac 下，还可以将其放到~/.bashrc、~/.bash_profile 中。这样系统每次启动，都会自动指定 node 作为全局的版本。

3.2.5　删除 NVM

对于 Linux、Mac，直接手动删掉对应的配置文件（如果有的话）即可。

- ~/.nvm
- ~/.npm
- ~/.bower

对于 Windows，可直接在控制面板中卸载该软件。

3.2.6　加快 NVM 和 NPM 的下载速度

由于某些原因，在国内连接国外的服务器会比较慢，所以我们使用下面的命令，就可以解决这个问题（默认是使用国外的服务器，现在改成使用国内的镜像服务器）。感谢淘宝提供了这个方法。

对于 NVM，使用 NVM_NODEJS_ORG_MIRROR 这个变量作为前缀。

```
$ NVM_NODEJS_ORG_MIRROR=https://npm.taobao.org/dist nvm install
```

对于 NPM，使用 cnpm 代替 npm 命令。

```
$ npm install -g cnpm --registry=https://registry.npm.taobao.org
```

对于 Linux、Mac 用户，可以直接创建一个"alias"命令。

```
alias cnpm="npm --registry=https://registry.npm.taobao.org \
--cache=$HOME/.npm/.cache/cnpm \
--disturl=https://npm.taobao.org/dist \
--userconfig=$HOME/.cnpmrc"
```

然后通过国内的淘宝服务器安装 node 包。例如：

```
$ cnpm install vue-cli -g
```

3.3 Git 在 Windows 下的使用

在《程序员修炼之道： 从小工到专家》中提到了一个让程序员非常尴尬的局面：老板要看进度，结果程序员拿不出来，只好跟老板撒谎：我的代码被猫吃了。

虽然我们的代码不会被猫吃掉，但是几乎每个程序员都会犯的错误就是：在下班的时候忘记保存，或者突然断电，结果导致写了几个小时的代码就这样没有了。因此，每个程序员必须要对自己的代码做版本控制。

在 2009 年之前，国内的人大部分都用 SVN。从 2010 年开始，越来越多的人开始使用 Git。本节专门为 Windows 程序员准备。因为对于 Linux 和 Mac 用户来说，Git 都是现成的一行命令就能搞定。

3.3.1 为什么要使用 Git Bash

Git Bash 不但提供了 Git，还提供了 bash，一种非常不错的类似于 Linux 的命令行。在 Windows 环境下，命令行模式与 Linux/Mac 是相反的。例如：

- Linux/Mac 下：（使用左斜线作为路径分隔符）

```
$ cd /workspace/happy_book_Vue.js
```

- Windows 下：（使用右斜线作为路径分隔符，并且要分成 C、D 等盘 ）

```
C:\Users\dashi>d:                          (进入 D 盘)
D:\>cd workspace\happy_book_Vue.js         (进入到对应目录)
```

只要不是做.NET/微信小程序/安卓开发，都应该转移到 Linux 平台上。原因是：代码被编译后，会运行在 Linux+Nginx 的服务器中。最好的办法就是从现在开始就适应 Linux 的环境。另外，命令行在绝大多数情况下比“图形化”的操作界面好用。

3.3.2　安装 git 客户端

在 Windows 下选择 Git Bash。官方网址为 https://gitforwindows.org/。

步骤 01 打开下载页面后就可以看到 Logo，如图 3-9 所示。

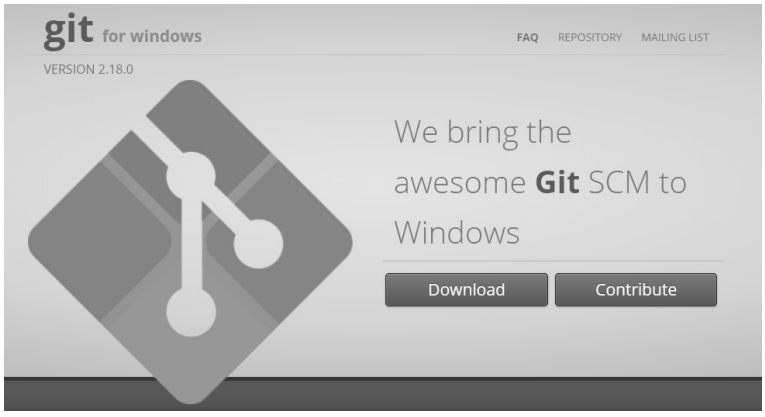

图 3-9　下载页面

步骤 02 选择最新版本，这里下载 2.16.2。

步骤 03 下载并运行，可以看到欢迎对话框，如图 3-10 所示。

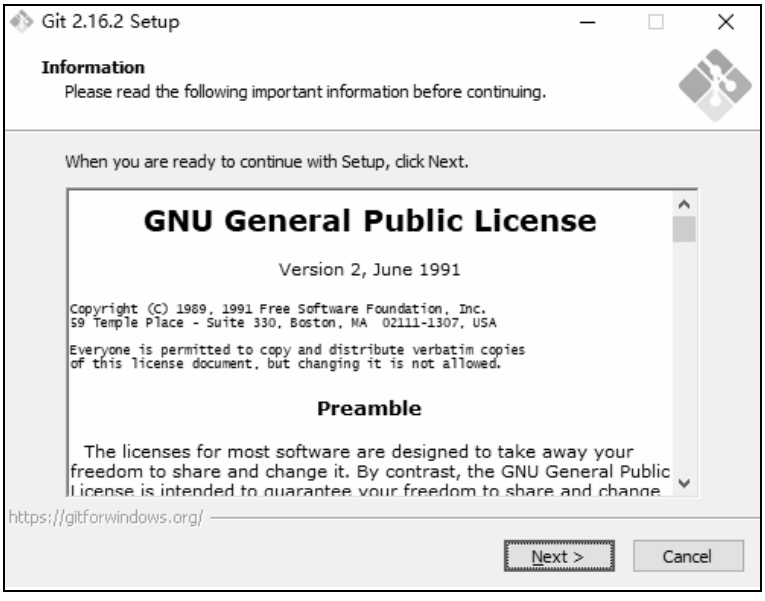

图 3-10　欢迎对话框

步骤 04 单击 Next 按钮，在打开的对话框中看到选择安装的内容，保持默认就好，如图 3-11 所示。

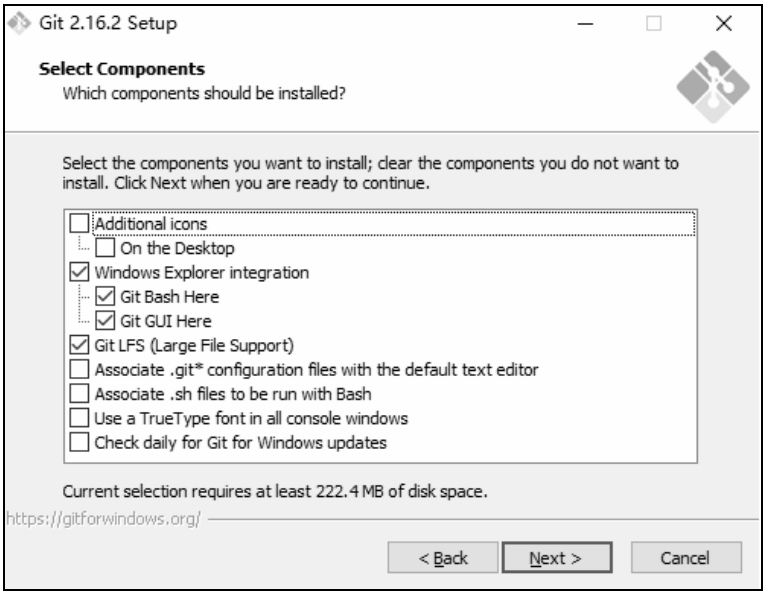

图 3-11　选择安装内容

步骤 05 单击 Next 按钮，在打开的对话框中选择哪个编辑器作为 git 消息编辑器。

- nano：最简单的 Linux 下的编辑器，同 Windows 下的记事本。学习曲线是 0。
- vim：需要长时间学习的编辑器，被称为“编辑器之神”。
- notepad++：加强型记事本，也很好用，学习曲线是 0。
- visual studio code：　一款 IDE。

对于新手来说，建议选择 notepad ++，如图 3-12 所示。

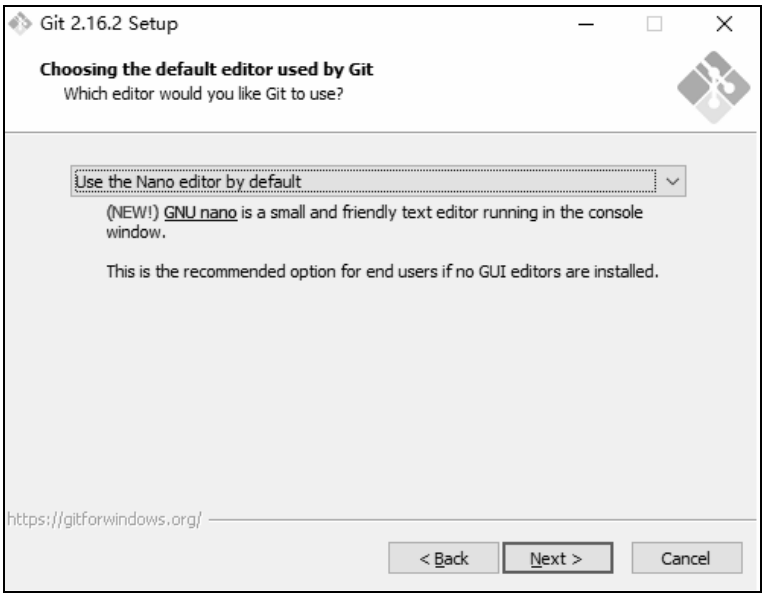

图 3-12　选择编辑器

步骤 06 单击 Next 按钮，询问使用什么风格的命令行。这里建议选择默认的 Use Git from Windows Command Prompt，如图 3-13 所示。

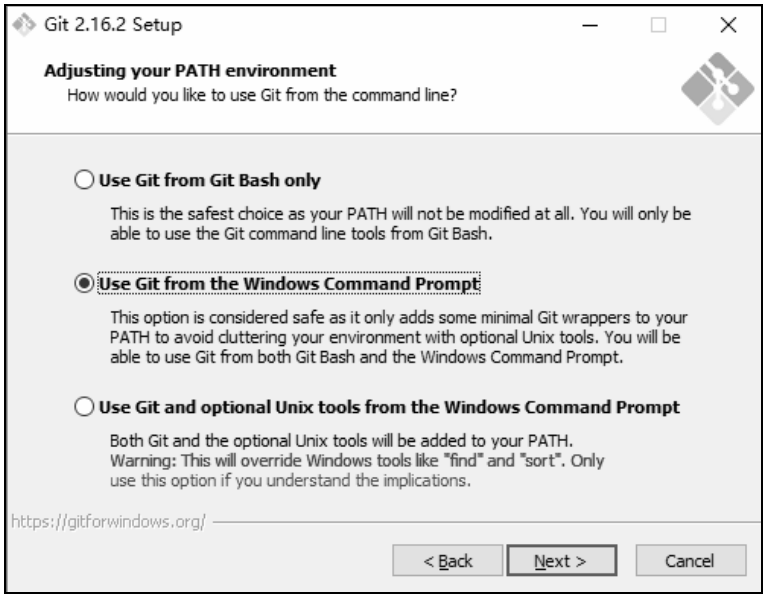

图 3-13　选择默认命令行

步骤 07 单击 Next 按钮，询问使用什么风格的 SSH 连接程序，如图 3-14 所示。

- OpenSSH SSH 的首选，是 Git bash 自带的。
- Plink 第三方用户自己安装的 SSH 连接程序。

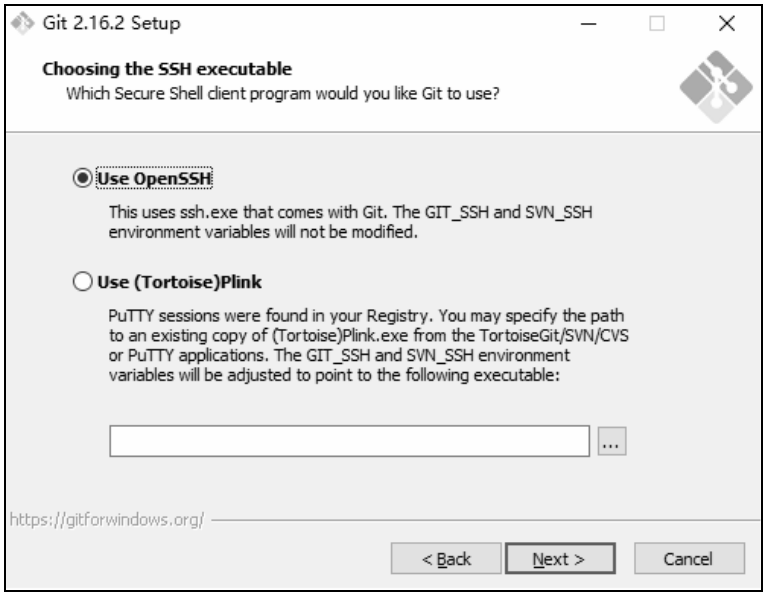

图 3-14　选择 SSH 连接程序

步骤 08 单击 Next 按钮，询问使用什么 SSH 后端，这里选择默认的 OpenSSL，如图 3-15 所示。

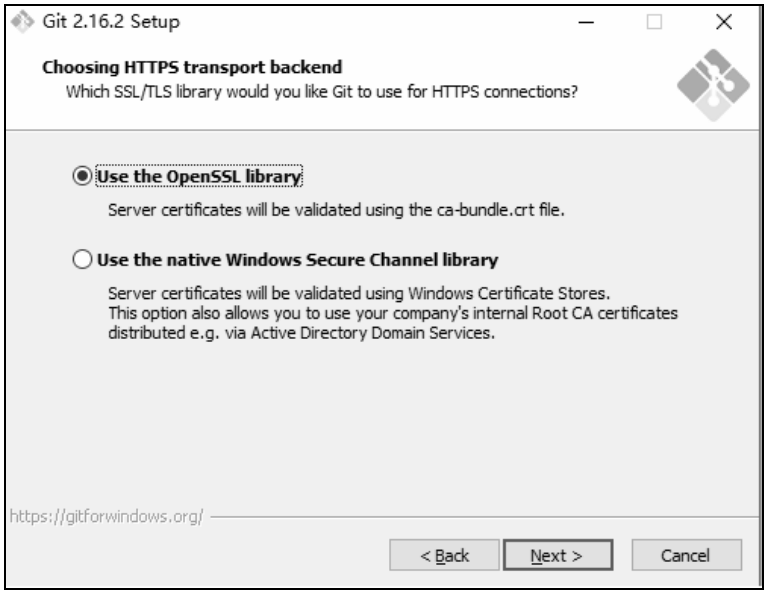

图 3-15　选择 SSH 后端

步骤 09 单击 Next 按钮，询问使用什么 Checkout/commit 风格。因为 Windows 与 linux 对待文件的处理是不同的，如回车在 Windows 下是\r\n，而在 linux 下就是\n，所以这里选择默认的第一项即可（用 Windows 风格 checkout，用 unix 风格 commit），如图 3-16 所示。

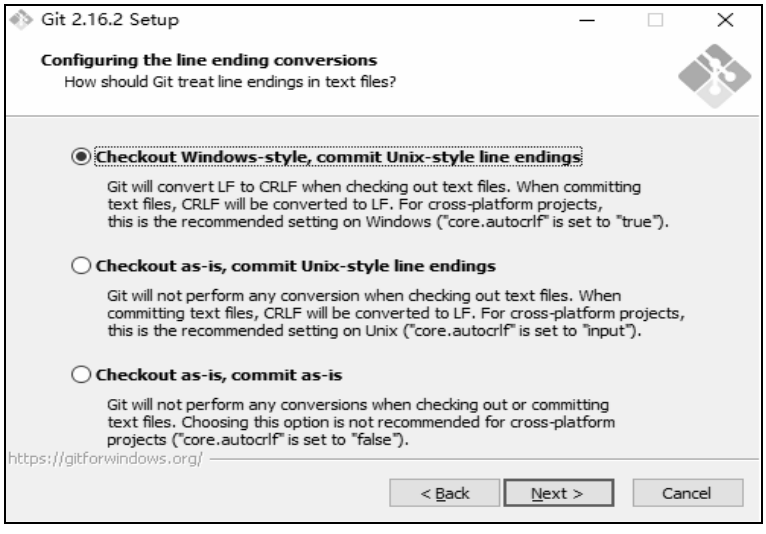

图 3-16　选择 Checkout/commit

步骤 10 单击 Next 按钮，询问用什么风格的 console（命令行）。这里一定要选择 Use MinTTY（the default terminal of MSYS2），也就是类似于 Linux 风格的命令行。可以说，它就是非常著名的 Cygwin，如图 3-17 所示。

图 3-17　选择 console（命令行）

步骤 11 单击 Next 按钮，询问其他配置项目。直接选择默认即可，如图 3-18 所示。

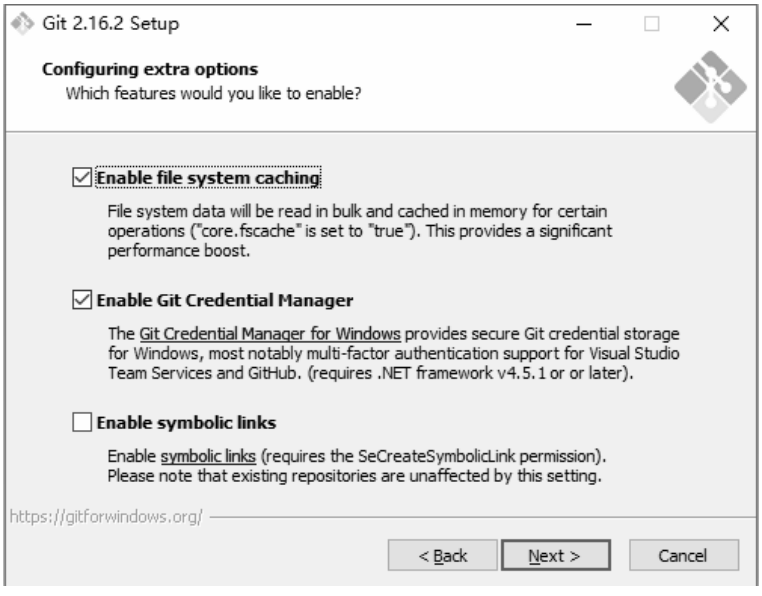

图 3-18　设置其他配置项目

步骤 12 继续单击 Next 按钮，就安装成功了，如图 3-19 所示。

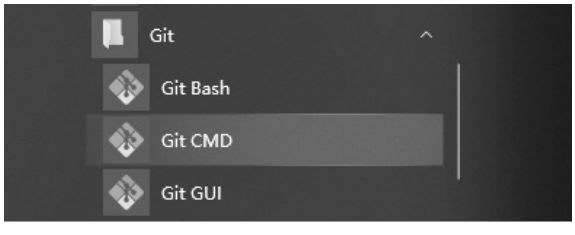

图 3-19　安装成功

3.3.3 使用 Git Bash

（1）打开 Git Bash。

打开 Git Bash，可以看到这个页面，如图 3-20 所示。

图 3-20　打开 Git Bash

一片空荡荡。估计习惯了鼠标和我的电脑的读者会非常不习惯。不要紧，我们慢慢来。

（2）查看当前路径：pwd。

输入 $ pwd 就可以知道当前位置了。

```
dashi@i5-16g MINGW64 ~
$ pwd
/c/Users/dashi
```

在上面的结果中可以看到：

- dashi 是 window 系统的用户名（笔者的外号叫大师）。
- i5-16g 是电脑在局域网的名字。
- MINGW64 是操作系统的名字。可以认为它是 Linux、Windows、Mac 之外的第 4 种操作系统。
- $是命令行的前缀。后面的 pwd 就是输入的命令。
- /c/Users/dashi 就是当前位置。这个是 Linux 风格。实际上，它对应的 Windows 的标准路径是 C:\Users\dashi

每次打开 Git Bash 的时候，都是默认的“当前用户在 Windows”中的用户文件夹。如果我们在一个窗口中打开这个路径，就可以看到我的用户文件夹了，如图 3-21 所示。

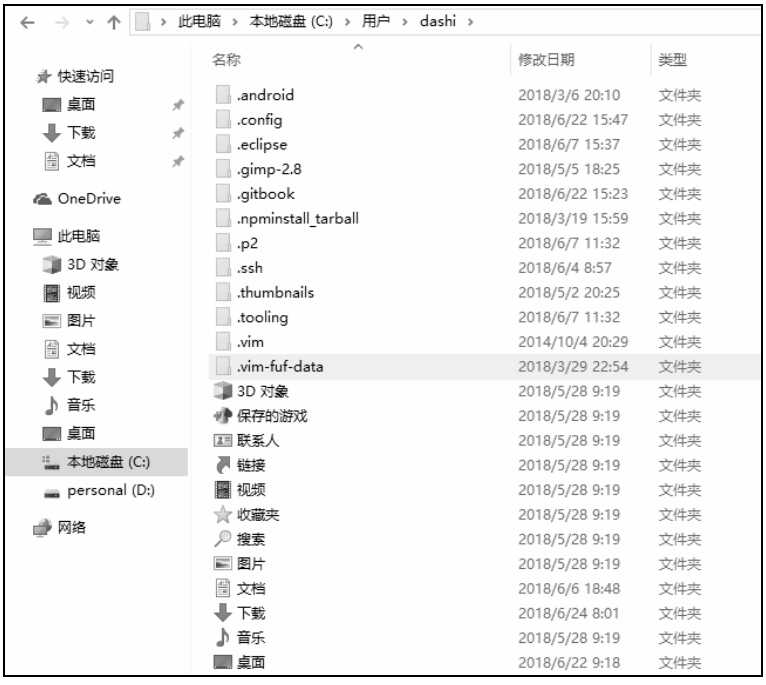

图 3-21　用户文件夹

可以看到输入的路径是 C:\Users\dashi，结果在 GUI 中显示的文字是 "> 此电脑 > 本地磁盘(C:) > 用户 > dashi"。

（3）切换路径：cd。

例如，想进入工作目录（位于 D:\workspace\happy_book_Vue.js），继续写关于 Vue.js 书，就可以这样：

```
dashi@i5-16g MINGW64 ~
$ cd /d/workspace/happy_book_Vue.js/

dashi@i5-16g MINGW64 /d/workspace/happy_book_Vue.js (master)
$
```

大家可以看到，D:\在 Git Bash 中对应的地址是/d，这个就是唯一需要注意的点了。

其他 git 基本知识（git clone、git commit、git push 等）就不在本书中赘述了。

3.4 Webpack

随着 SPA（Single Page App）单页应用的发展，可以发现，使用的 js/css/png 等文件特别多，比较难管理，文件夹结构也很容易混乱。很多时候我们希望项目可以具备压缩 css，压缩

js，正确地处理各种 js/css 的 import，以及相关的模板 html 文件。

在最开始的一段时间里，可以说每个 SPA 项目发展到一定规模，都会遇到这样的瓶颈和痛点。为了解决这个问题，就出现了 Webpack，其官方网站为 https://webpack.js.org/，github 为 https://github.com/webpack/ webpack。

Webpack 官方网站页面如图 3-22 所示。

图 3-22　Webpack 官方页面

Webpack 是一个打包工具，可以把 js、css、node module、coffeescrip、scss/less、图片等都打包在一起，简直是模块化开发 SPA 福音。因此，现在几乎所有的 SPA 项目、JS 项目都会用到 Webpack。

在前面的入门知识中，我们看到 Vue.js 只需要引入一个外部的 js 文件就可以工作了。不过，在实际开发中，情况就复杂了很多倍，我们都是统一使用 Webpack + Vue.js 的方式来做项目的，这样才可以做到“视图”“路由”“component”等的分离，以及快速打包、部署及项目上线。

3.4.1　Webpack 功能

Webpack 功能非常强大，对各种技术都提供了支持，仿佛是一个“万能胶水”，把所有的技术都结合到了一起。

1. 对文件的支持

- 支持普通文件
- 支持代码文件
- 支持文件转 url（支持图片）

2. 对 JSON 的支持

- 支持普通 JSON

- 支持 JSON5
- 支持 CSON

3. 对 JS 预处理器的支持

- 支持普通 JavaScript
- 支持 Babel（使用 ES2015+）
- 支持 Traceur（使用 ES2015+）
- 支持 Typescript
- 支持 Coffeescript

4. 对模板的支持

- 支持普通 HTML
- 支持 Pug 模板
- 支持 JADE 模板
- 支持 Markdown 模板
- 支持 PostHTML
- 支持 Handlebars

5. 对 Style 的支持

- 支持普通 style
- 支持 import
- 支持 LESS
- 支持 SASS/SCSS
- 支持 Stylus
- 支持 PostCSS

6. 对各种框架的支持

- 支持 Vue.js
- 支持 Angular2
- 支持 Riot

3.4.2　Webpack 安装与使用

Webpack 安装命令如下：

```
$ npm install --save-dev webpack
```

因为 Webpack 自身是支持 Vue.js 的，所以 Webpack 与 Vue.js 已经结合到很难分清谁是谁，我们就索性不区分。知道做什么事情需要运行什么命令就可以了。

在接下来内容中，读者会看到 Webpack+Vue.js 共同开发的方法和步骤。

3.5 开发环境的搭建

- NPM：3.10.8 (NPM > 3.0)
- Node：v6.9.1 (Node > 6.0)
- Vue.js：2.0+

总体来说，只要能安装上 Node 和 Vue.js 就可以。

3.5.1 安装 Vue.js

需要同时安装 vue 和 vue-cli 这两个 node package。

运行下面的命令：

```
$ npm install vue vue-cli -g
```

`-g` 表示这个包安装后可以被全局使用。

3.5.2 运行 vue

创建一个基于 Webpack 模板的新项目。

```
$ vue init webpack my-project
```

注意，我们使用 Vue 都是在 Webpack 前提下使用的。

安装依赖：

```
$ cd my-project
$ cnpm install
```

在本地以默认端口来运行。

```
$ npm run dev
```

然后就可以看到在本地已经运行起来了。

```
> test_vue_0613@1.0.0 dev /workspace/test_vue_0613
> node build/dev-server.js


> Starting dev server...



 DONE  Compiled successfully in 12611ms   12:16:47 PM
```

```
> Listening at http://localhost:8080
```

打开 http://localhost:8080 就可以看到刚才创建的项目欢迎页，如图 3-23 所示。

图 3-23　项目欢迎页

3.6 Webpack 下的 Vue.js 项目文件结构

前面我们已经安装了 Webpack、vue-cli 并运行成功，看到了 Vue.js 的第一个页面。那么我们首先需要对 Webpack 下的 Vue.js 有一个全面的了解。

运行下面命令后：

```
$ vue init webpack my-project
```

会生成一个崭新的项目。它的文件结构如下：

```
▸ build/              // 编译用到的脚本
  config/             // 各种配置
  dist/               // 打包后的文件夹
  node_modules/       // node 第三方包
  src/                // 源代码
  static/             // 静态文件，暂时无用
  index.html          // 最外层文件
  package.json        // node 项目配置文件
```

下面我们针对重要的文件和文件夹依次说明。

3.6.1 build 文件夹

build 文件夹中保留了各种打包脚本，是不可或缺的，不可随意修改。

build 文件夹展开后如下：

```
▾ build/
    build.js
    check-versions.js
    dev-client.js
    dev-server.js
    utils.js
    vue-loader.conf.js
    webpack.base.conf.js
    webpack.dev.conf.js
    webpack.prod.conf.js
```

- build.js：打包使用，不能修改。
- check-versions.js：检查 npm 的版本，不能修改。
- dev-client.js 和 dev-server.js：是在开发时使用的服务器脚本，不能修改（借助于 node 后端语言，在做 Vue.js 开发时，可以通过$npm run dev 命令打开一个小的 server 运行 Vue.js）。
- utils.js：不能修改，做一些 css/sass 等文件的生成。
- vue-loader.conf.js：非常重要的配置文件，不能修改。内容是用于辅助加载 Vue.js 用到的 css source map 等。
- Webpack.base.conf.js、Webpack.dev.conf.js、Webpack.prod.conf.js：这三个都是基本的配置文件，不能修改。

3.6.2 config 文件夹

与部署和配置相关。

```
▾ config/
    dev.env.js
    index.js
    prod.env.js
```

- dev.env.js：开发模式下的配置文件，一般不用修改。
- prod.env.js：生产模式下的配置文件，一般不用修改。
- index.js：很重要的文件，定义了开发时的端口（默认 8080）、图片文件夹（默认 static）、开发模式下的代理服务器。

对于这个文件夹的内容，会在后续的章节中陆续进行解释（如对于某个页面的渲染过程、如何做代理转发）。

3.6.3　dist 文件夹

打包之后的文件所在目录如下：

```
▾ dist/
    static/
      css/
        app.d41d8cd98f00b204e9800998ecf8427e.css
        app.d41d8cd98f00b204e9800998ecf8427e.css.map
      js/
        app.c482246388114c3b9cf0.js
        app.c482246388114c3b9cf0.js.map
        manifest.577e472792d533aaaf04.js
        manifest.577e472792d533aaaf04.js.map
        vendor.5f34d51c868c93d3fb31.js
        vendor.5f34d51c868c93d3fb31.js.map
    index.html
```

可以看到，对应的 css、js、map 都在这里。

注意，文件名中无意义的字符串是随机生成的。目的是为了让文件名发生变化，方便部署，同时方便 Nginx 服务器重新对该文件进行缓存。

- app.css：编译后的 CSS 文件。
- app.js：最核心的 js 文件，几乎所有的代码逻辑都会打包到这里。
- manifest.js：生成的周边 js 文件。
- vendor.js：生成的 vendor.js 文件。

另外，每个.map 文件都非常重要。可以简单地认为，有了.map 文件，浏览器就可以先下载整个.js 文件，然后在后续的操作中“部分加载”对应的文件。

切记这个文件夹不要放到 git 中。因为每次编译之后，这里的文件都会发生变化。

3.6.4　node_modules 文件夹

node 项目所用到的第三方包特别多，也特别大。这些文件是由命令$ npm install 产生的。所有在 package.json 中定义的第三方包都会被自动下载，保存在该文件夹下。

package.json：

```
// ...
```

```
"dependencies": {
  "vue": "^2.3.3",
  "vue-resource": "1.3.3",
  "vue-router": "^2.3.1",
  "vuex": "^2.3.1"
},
"devDependencies": {
  "autoprefixer": "^6.7.2",
  "babel-core": "^6.22.1",
  "babel-loader": "^6.2.10",
  "babel-plugin-transform-runtime": "^6.22.0",
  "babel-preset-env": "^1.3.2",
  "babel-preset-stage-2": "^6.22.0",
  "babel-register": "^6.22.0",
  //...
}
// ...
```

node_modules 文件夹中往往会有几百个文件夹，如图 3-24 所示。

图 3-24　node_modules 文件夹

注意，这个文件夹不能放到 git 中。

3.6.5　src 文件夹

src 文件夹是核心源代码的所在目录。展开后如下所示（不同版本的 vue-cli 生成的目录会稍有不同，不过核心都是一样的）：

```
▾ src/
    assets/
      logo.png
    components/
      Book.vue
      BookList.vue
      Hello.vue
    router/
      index.js
   App.vue
   main.js
```

- assets 文件夹用到的图片都可以放在这里。
- components 用到的“视图”和“组件”所在的文件夹（核心）。
- router/index.js 是路由文件，定义了各个页面对应的 url。
- 如果 index.html 是一级页面模板的话，App.vue 就是二级页面模板。 所有的其他 Vue.js 页面都作为该模板的一部分被渲染出来。
- main.js 没有实际的业务逻辑，但是为了支撑整个 Vue.js 框架，很有必要存在。

第 4 章

◂ Webpack+Vue.js实战 ▸

4.1 创建一个页面

激动人心的时刻到来了，接下来我们需要通过自己动手开始下一步的学习。

请读者务必准备好电脑，只有一边学习一边编写代码，才能真正看到效果，因为调试代码的过程是无法脑补出来的。

在 Vue.js 中创建页面需要以下两步。

（1）新建路由。

（2）新建 vue 页面。

4.1.1 新建路由

默认的路由文件是 src/router/index.js，将其打开之后，我们增加两行：

```
import Vue from 'vue'
import Router from 'vue-router'

// 增加这一行，作用是引入 SayHi 这个 component
import SayHi from '@/components/SayHi'

Vue.use(Router)
export default new Router({
  routes: [

    // 增加下面几行，表示定义了  /#/say_hi 这个 url
    {
```

```
    path: '/say_hi',
    name: 'SayHi',
    component: SayHi
  },
 ]
})
```

上面的代码中：

```
import SayHi from '@/components/SayHi'
```

表示从当前目录下的 components 引入 SayHi 文件，@表示当前目录。

然后利用下面的代码定义一个路由：

```
routes: [
  {
    path: '/say_hi',   // 对应一个 url
    name: 'SayHi',     // Vue.js 内部使用的名称
    component: SayHi   // 对应的.vue 页面的名字
  },
]
```

也就是说，每当用户的浏览器访问 http://localhost:8080/#/say_hi 时，就会渲染./components/SayHi.vue 文件。name: 'SayHi' 定义了该路由在 Vue.js 内部的名称。

4.1.2　创建一个新的 Component

由于我们在路由中引用了 component： src/components/SayHi.vue，接下来，就创建这个文件。代码如下：

```
<template>
 <div >
  Hi Vue!
 </div>
</template>

<script>
export default {
 data () {
  return { }
```

```
  }
}
</script>

<style>
</style>
```

在上面的代码中：

- <template></template>代码块中表示的是 HTML 模板，里面写的就是最普通的 HTML 。
- <script/>表示的是 js 代码，所有的 js 代码都写在这里。这里使用的是 EMScript。
- <style>表示所有的 CSS/SCSS/SASS 文件都可以写在这里。

现在，可以直接访问 http://localhost:8080/#/say_hi 了，页面如图 4-1 所示。

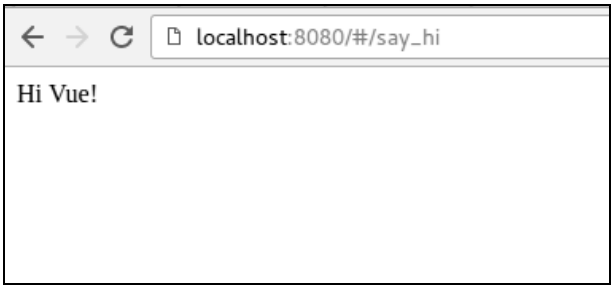

图 4-1　页面效果

4.1.3　为页面添加样式

我们可以为页面添加一些样式，让它变得好看一些。

```
<template>
  <div class='hi'>
    Hi Vue!
  </div>
</template>

<script>
export default {
  data () {
    return { }
  }
```

```
}
</script>

<style>
.hi {
  color: red;
  font-size: 20px;
}
</style>
```

注意上面代码中的<style>标签，里面与普通的 CSS 一样定义了样式。

```
.hi {
  color: red;
  font-size: 20px;
}
```

刷新浏览器，可以看到文字加了颜色，如图 4-2 所示。

图 4-2　为文字添加颜色

4.1.4　定义并显示变量

如果要在 vue 页面中定义一个变量，并显示出来，就需要事先在 data 中定义。

```
export default {
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
    }
  }
}
```

可以看到，上面的代码是通过 Webpack 的项目来写的。 回忆一下，在原生的 Vue.js 中是如何定义一个变量（property）的？

答案是：

```
var app = new Vue({
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
    }
  }
})
```

我们可以认为，之前在“原生的 Vue.js”的代码中存在于 new Vue({...})中的代码，在 Webpack 框架下，都应该放到 export default{ .. }代码块中。

完整的代码（src/components/SayHi.vue）如下所示：

```
<template>
  <div>
    <!--  步骤 2: 在这里显示 message -->

  </div>
</template>

<script>
export default {
  data () {
    return {
      // 步骤 1: 这里定义了变量 message 的初始值
      message: '你好 Vue! 本消息来自于变量'
    }
  }
}
</script>

<style>
</style>
```

页面效果如图 4-3 所示。

图 4-3　页面效果

4.2 Vue.js 中的 ECMAScript

有一定编程经验的读者会发现，我们使用的不是“原生的 JavaScript”，而是一种新的语言，这个语言就是 ECMAScript。

严格来说，ECMAScript 是 JavaScript 的规范，JavaScript 是 ECMA 的实现。

ECMAScript 的简称是 ES，其版本比较多，有 ES 2015、ES 2016、ES 2017 等，很多时候我们用 ES6 来泛指这三个版本。从 http://kangax.github.io/compat-table/es6/ 可以看到，ES6 的 90%的特性都已经被各大浏览器实现了。

具体的细节不去深究，我们就暂且认为 ECMAScript 实现了很多普通 js 无法实现的功能。同时，在 Vue.js 项目中大量使用了 ES 的语法。

下面是极简版的 ES6 入门知识。大家只要看懂这些代码，就可以继续阅读本书。

4.2.1 let、var、常量与全局变量

声明本地变量，使用 let 或 var，两者的区别如下。

- var：有可能引起变量提升，或者块级作用域的问题。
- let：就是为了解决以上两个问题存在的。

最佳实践：多用 let，少用 var，遇到诡异变量问题时，就查一查是不是 var 的问题。下面是三个对比：

```
var title = '标题';   // 没问题
let title = '标题';   // 没问题
title = '标题';       // 这样做会报错.
```

在 Webpack 下的 Vue.js 中使用任何变量，都要使用 var 或 let 来声明。常量：

```
const TITLE='标题';
```

对于全局变量，直接在 index.html 中声明即可。例如：

```
window.title = '我的博客列表'
```

4.2.2 导入代码：import

import 用于导入外部代码。例如：

```
import Vue from 'vue'
import Router from 'vue-router'
```

上面的代码，目的是引入 vue 和 vue-router（由于它们是在 package.json 中定义的，因此可以直接 import ... from <包名>，否则要加上路径）。

```
import SayHi from '@/components/SayHi'
```

在 from 后面添加@符号，表示是在本地文件系统中引入文件。@代表源代码目录，一般是 src。

@出现之前，我们在编码时也会这样写：

```
import Swiper from '../components/swiper'
import SwiperItem from '../components/swiper-item'
import XHeader from '../components/header/x-header'
import store from '../vuex/store'
```

因为大量使用了../..，这样的代码会引起混乱，所以推荐使用@方法。

4.2.3 方便其他代码使用自己: export default {..}

在每个 vue 文件的<script>中，都会存在 export default {..}代码，作用是方便其他代码对这个代码进行引用。对于 Vue.js 程序员来说，记住这个写法就可以了。

在 ES6 之前，js 没有统一的模块定义方式，流行的定义方式有 AMD、CommonJS 等，这些方式都是以一种“打补丁”的形式实现这个功能。ES6 从语言层面对定义模块的方式进行了统一。

假设有 lib/math.js 文件，其内容如下：

```
export function sum(x, y) {
  return x + y
}
export var pi = 3.141593
```

lib/math.js 文件可以导出两个内容，一个是 function sum，另一个是 var pi。

我们可以定义一个新的文件：app.js，内容如下：

```
import * as math from "lib/math"
alert("2π = " + math.sum(math.pi, math.pi))
```

在上面的代码中， 可以直接调用 math.sum 和 math.pi 方法。

新建一个文件：other_app.js，内容如下：

```
import {sum, pi} from "lib/math"
alert("2π = " + sum(pi, pi))
```

在上面的代码中，通过 import {sum, pi} from "lib/math"可以在后面直接调用 sum()和 pi。而 export default { ... } 则是暴露出一段没有名字的代码，不像 export function sum(a,b){ .. }有一个名字（sum）。

在 Webpack 下的 Vue.js，会自动对这些代码进行处理，属于框架内的工作。读者只要按照这个规则来写代码，就一定没有问题。

4.2.4　ES 中的简写

有时我们会发现这样的代码：

```
<script>
export default {
  data () {
    return { }
  }
}
</script>
```

实际上，上面的代码是一种简写形式，等同于下面的代码：

```
<script>
export default {
  data: function() {
    return { }
  }
}
</script>
```

4.2.5　箭头函数=>

与 coffeescript 一样，ES 也可以通过箭头表示函数。

```
  .then(response => ... );
```

等同于：

```
  .then(function (response) {
    // ...
```

```
  })
```

这样写的好处就是强制定义了作用域。使用=>之后，可以避免很多由作用域产生的问题，建议大家多使用。

4.2.6 hash 中同名的 key、value 的简写

```
let title = 'triple body'


return {
  title: title
}
```

等同于：

```
let title = 'triple body'


return {
  title
}
```

4.2.7 分号可以省略

例如：

```
var a = 1
var b = 2
```

等同于：

```
var a = 1;
var b = 2;
```

4.2.8 解构赋值

我们先定义好一个 hash：

```
let person = {
  firstname : "steve",
  lastname : "curry",
  age : 29,
  sex : "man"
};
```

然后可以这样做定义：

```
let {firstname, lastname} = person
```

上面一行代码等同于：

```
let firstname = person.firstname
let lastname = person.lastname
```

可以这样定义函数：

```
function greet({firstname, lastname}) {
  console.log(`hello,${firstname}.${lastname}!`);
};

greet({
  firstname: 'steve',
  lastname: 'curry'
});
```

但是不建议读者这样使用。另外，浏览器和一些第三方支持应用的不是太好，我们在实际项目中，曾经遇到过与之相关的很奇葩的问题。

有关 ECMAScript，国内比较好的中文教材，是阮一峰编写的 《ES6 标准入门（第 3 版）》，书中非常详实地阐述了相关的内容、概念和用法，也可以在网上直接查看这本书的电子版：http://es6.ruanyifeng.com。

4.3 Vue.js 渲染页面的过程和原理

只有知道一个页面是如何被渲染出来的，才能更好地理解框架和调试代码。下面就来学习一下这个过程。

4.3.1 渲染过程 1：js 入口文件

首先我们要知道./build/webpack.base.conf.js 是 webpack 打包的主要配置文件。一个典型的代码如下：

```
var path = require('path')
var utils = require('./utils')
```

```
var config = require('../config')
var vueLoaderConfig = require('./vue-loader.conf')

function resolve (dir) {
  return path.join(__dirname, '..', dir)
}

module.exports = {
  entry: {
    app: './src/main.js'
  },
  output: {
    path: config.build.assetsRoot,
    filename: '[name].js',
    publicPath: process.env.NODE_ENV === 'production'
      ? config.build.assetsPublicPath
      : config.dev.assetsPublicPath
  },
  resolve: {
    extensions: ['.js', '.vue', '.json'],
    alias: {
      'vue$': 'vue/dist/vue.esm.js',
      '@': resolve('src')
    }
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: vueLoaderConfig
      },
      {
        test: /\.js$/,
        loader: 'babel-loader',
        include: [resolve('src'), resolve('test')]
```

```
    },
    {
      test: /\.(png|jpe?g|gif|svg)(\?.*)?$/,
      loader: 'url-loader',
      options: {
        limit: 10000,
        name: utils.assetsPath('img/[name].[hash:7].[ext]')
      }
    },
    {
      test: /\.(woff2?|eot|ttf|otf)(\?.*)?$/,
      loader: 'url-loader',
      options: {
        limit: 10000,
        name: utils.assetsPath('fonts/[name].[hash:7].[ext]')
      }
    }
  ]
 }
}
```

在上面的代码中：

```
module.exports = {
  entry : {
    app: './src/main.js'   // 这里就定义了 Vue.js 的入口文件
  }
}
```

其中 app：'./src/main.js'就定义了 Vue.js 的入口文件。

4.3.2　渲染过程 2：静态的 HTML 页面（index.html）

因为我们每次打开的都是'http://localhost/#/'，实际上打开的文件是'http://localhost/#/index.html'，所以找到 index.html，就可以看到其内容。

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title>演示 Vue 的 demo</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

这里的 就是将来会动态变化的内容。特别类似于 Rails 的 layout（模板文件）。

这个 index.html 文件是最外层的模板。

4.3.3 渲染过程 3：main.js 中的 Vue 定义

我们来看看 src/main.js 文件，其内容如下：

```
import Vue from 'vue'
import App from './App'
import router from './router'
import VueResource from 'vue-resource';
Vue.use(VueResource);

Vue.config.productionTip = false
Vue.http.options.emulateJSON = true;

new Vue({
  el: '#app',   // 这里, #app 对应着 <div id=app></div>
  router,
  template: '<App/>',
  components: { App }  // 这里, App 就是 ./src/App.vue
})
```

熟悉 jquery、css selector 的读者可以看到，el: '#app'就是 index.html 中的

上面的 App.vue 会被 main.js 加载。App.vue 的内容如下：

```
<template>
  <div id="app">
       <router-view class="view"></router-view>
  </div>
```

```
</template>
<script>
</script>
<style>
</style>
```

上面代码中的<template/>就是第二层模板，可以认为该页面的内容就是在这个位置被渲染出来的。

在上面的代码中，还有 ... ，该元素与 index.html 中的是同一个元素（暂且这样理解）。

所有<router-view>中的内容都会被自动替换。<script>中的代码则是脚本代码。至此，整个过程就出来了。

4.3.4　渲染原理与实例

Vue.js 就是最典型的 Ajax 工作方式，即只渲染部分页面。

浏览器的页面从来不会整体刷新，所有的页面变化都限定在 index.html 中的 代码中。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>演示 Vue 的 demo</title>
  </head>
  <body>
    <div id="app"></div>   <!-- 都在这一行 -->
  </body>
</html>
```

所有的动作都可以靠 url 来触发。例如：

- /#/books_list 对应某个列表页。
- /#/books/3 对应某个详情页。

这个技术就是靠 Vue.js 的核心组件 vue-router 来实现的。

不使用 router 的技术：QQ 邮箱

QQ 邮箱是属于 url 无法与某个页面一一对应的项目。所有页面的跳转，都无法根据 url 来判断。

最大的特点是不能保存页面的状态，难以调试，无法根据 url 进入某个页面。

4.4 视图中的渲染

前面我们介绍了项目的运行（hello world）、文件夹的结构及 index.html 中的内容是如何一点点渲染出来的。下面学习 Vue.js 中视图的操作。

4.4.1 渲染某个变量

假设定义了一个变量：

```
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
}
</script>
```

可以这样来显示它：

```
<div></div>
```

完整代码如下：（src/components/Hello.vue）

```
<template>
  <div>


  </div>
</template>


<script>
export default {
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
```

```
    }
  }
}
</script>

<style>
</style>
```

上面的代码显示定义了 message 变量，然后将其在<h1> </h1>中显示出来。

打开 http://localhost:8080/#/say_hi_from_variable 页面，就可以看到如图 4-4 所示的结果。

图 4-4　运行结果

4.4.2　方法的声明和调用

声明一个 show_my_value 方法。

```
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    show_my_value: function(){
      // 注意下面的  this.my_value, 要用到 this 关键字
      alert('my_value: ' + this.my_value);
    },
  }
}
</script>
```

调用上面的方法。

```
<template>
  <div>
    <input type='button' @click="show_my_value()" value='...'/>
  </div>
</template>
```

对于有参数的方法，直接传递参数就可以了。例如：

```
<template>
  <div>
    <input type='button' @click="say_hi('Jim')" value='...'/>
  </div>
</template>
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    say_hi: function(name){
      alert('hi, ' + name)
    },
  }
}
</script>
```

上面的代码中：

```
<input type='button' @click="say_hi('Jim')" value='...'/>
```

就会调用 say_hi 方法，传入参数'Jim'。

4.4.3 事件处理：v-on

很多时候，@click 等同于 v-on:click。下面两个是一样的：

```
<input type='button' @click="say_hi('Jim')" value='...'/>
```

```
<input type='button' v-on:click="say_hi('Jim')" value='...'/>
```

4.5 视图中的 Directive（指令）

我们在学习 Java 的时候，知道有 jsp 页面，对于.net 语言，有.asp、aspx 页面， 对于 Ruby，有 erb 页面，在 Vue.js 中也有类似的编程能力。但是因为 Vue.js 是一种 javascript 框架，所以只能与标签结合使用，叫做 Directive（指令）。

我们之前看到的 v-on、v-bind、v-if、v-for 等，只要是以 v 开头的，都是 Directive。

原理：

（1）用户在浏览器中输入网址，按回车键。

（2）浏览器加载所有的资源（js、html 内容），此时尚未解析。

（3）浏览器加载 Vue.js。

（4）Vue.js 程序被执行，发现页面中的 Directive 并进行相关的解析。

（5）HTML 中的内容被替换，展现给用户。

因此，我们在开发一个 Vue.js 项目的时候，会看到大量的 Directive。这里的基础务必打好。

4.5.1 前提：在 directive 中使用表达式（Expression）

- 表达式：a>1（有效）。
- 普通语句：a=1;（这是个声明，不会生效）。
- 控制语句：returna;（不会生效）。

在所有的 Directive 中，只能使用表达式。

正确的：

```
<div v-if="a > 100">
</div>
```

错误的：

```
<div v-if="return false">
</div>
```

4.5.2 循环：v-for

完整的代码如下：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p>Vue.js 周边的技术生态有： <p>
        <br/>
        <ul>
            <li v-for="tech in technologies">
                {{tech}}
            </li>
        </ul>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                technologies: [
                    "nvm", "npm", "node", "webpack", "ecma_script"
                ]
            }
        })
    </script>
</body>
</html>
```

上面代码中的 technologies 是在 data 中被定义的。

```
data: {
    technologies: [
        "nvm", "npm", "node", "webpack", "ecma_script"
    ]
}
```

然后在下面的代码中被循环显示。

```
<li v-for="tech in technologies">
```

```
</li>
```

使用浏览器打开上面代码后，结果如图 4-5 所示。

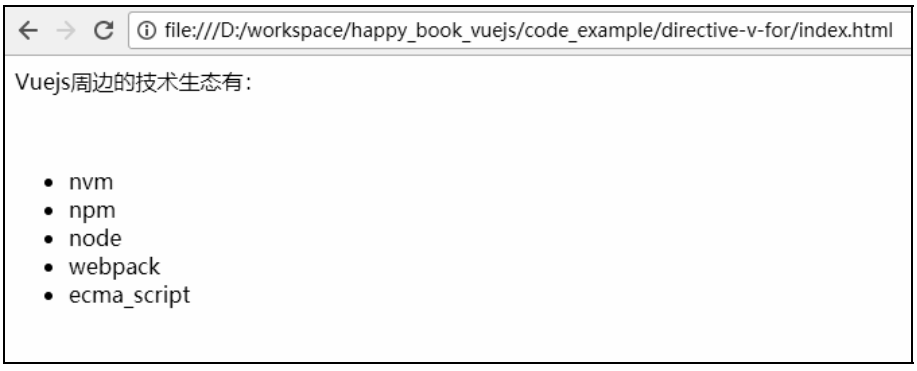

图 4-5　运行结果

4.5.3　判断：v-if

条件 Directive 是由 v-if、v-else-if、v-else 配合完成的。下面是一个完整的例子：

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>
      <p>我们要使用的技术是： <p>

      <div v-if="name === 'Vue.js'">
         Vue.js !
      </div>
      <div v-else-if="name === 'angular'">
         Angular
      </div>
      <div v-else>
         React
      </div>
   </div>
   <script>
```

```
        var app = new Vue({
            el: '#app',
            data: {
                name: 'Vue.js'
            }
        })
    </script>
</body>
</html>
```

注意，v-if 后面的引号中是 name=== 'Vue.js'，===是 Ecmascript 的语言，表示严格判断（由于 js 的==有先天缺陷，因此在 95%的情况下，都是使用三个等号的形式）。使用浏览器打开上面的代码后，结果如图 4-6 所示。

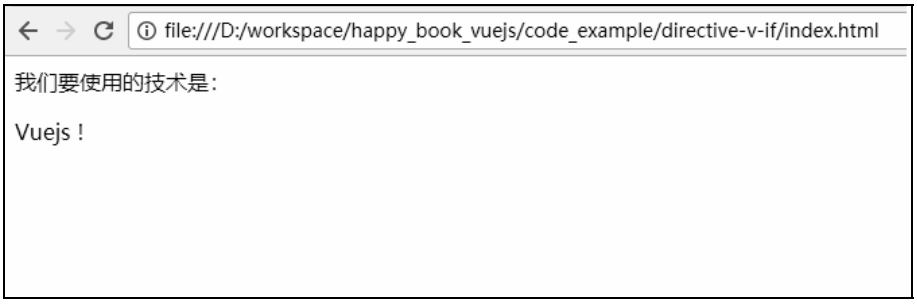

图 4-6　运行结果

4.5.4　v-if 与 v-for 的优先级

当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级。也就是说，Vue.js 会先执行 v-for，再执行 v-if。

下面是个完整的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p> 全部的技术是:  </p>
        <p> v-for 与 v-if 的结合使用, 只打印出 以"n"开头的技术:  <p>
        <ul>
```

```
            <li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
                {{tech}}
            </li>
        </ul>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                technologies: [
                    "nvm", "npm", "node", "webpack", "ecma_script"
                ]
            }
        })
    </script>
</body>
</html>
```

可以看到，在上面的代码中，v-if 与 v-for 结合使用了，先是做了循环 tech in technologies，然后对当前的循环对象 tech 做了判断。核心代码如下所示：

```
<li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
    {{tech}}
</li>
```

使用浏览器打开上面的代码后，结果如图 4-7 所示。

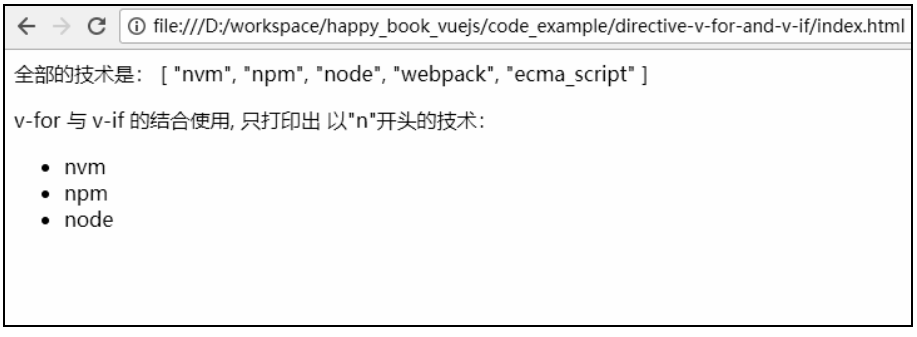

图 4-7　运行结果

4.5.5 v-bind

v-bind 指令用于把某个属性绑定到对应的元素属性。例如：

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>
      <p v-bind:style="'color:' + my_color">Vue.js 学起来好开心~ </p>
   </div>
   <script>
      var app = new Vue({
         el: '#app',
         data: {
            my_color: 'green'
         }
      })
   </script>
</body>
</html>
```

在上面的代码中，通过 v-bind 把<p>元素的 style 的值绑定成了'color:' + my_color 表达式。当 my_color 的值发生变化时，对应<p>的颜色也会发生变化。

例如：默认页面打开后，文字是绿色的，如图 4-8 所示。

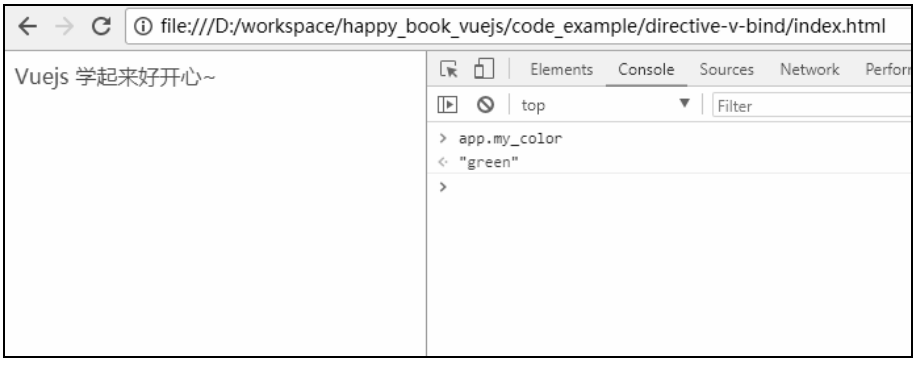

图 4-8 默认文字是绿色的

如何知道变量 my_color 已经绑定到<p>上了呢？我们在 console 中做修改，让 app.my_color = "red"，就可以看到对应的文字颜色变成了红色，如图 4-9 所示。

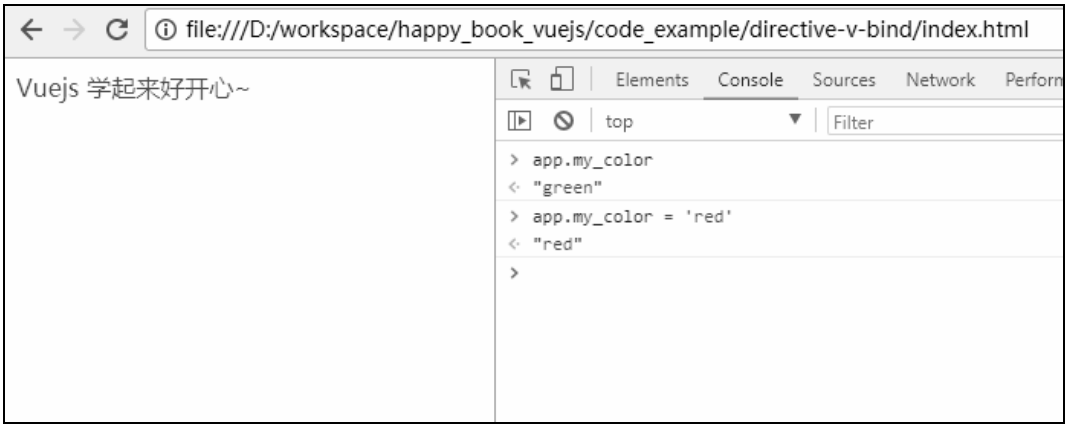

图 4-9　文字变成了红色

对于所有的属性，都可以使用 v-bind。例如：

```
<div v-bind:style='...'>
</div>
```

会生成：

```
<div style='...'>  </div>
<div v-bind:class='...'> </div>
```

会生成：

```
<div class='...'>  </div>
<div v-bind:id='...'> </div>
```

会生成：

```
<div id='...'>  </div>
```

4.5.6　v-on

v-on 指令用于触发事件。例如：

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>

      <br/>
```

```
        <button v-on:click='highlight' style='margin-top: 50px'>真的吗</button>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                message: '学习 Vue.js 使我快乐~ '
            },
            methods: {
                highlight: function() {
                    this.message = this.message + '是的， 工资还会涨~!'
                }
            }

        })
    </script>
</body>
</html>
```

在上面的代码中，通过 v-on:click 的声明，当被单击（click）后，就会触发 highlight 方法。

单击前的页面如图 4-10 所示。

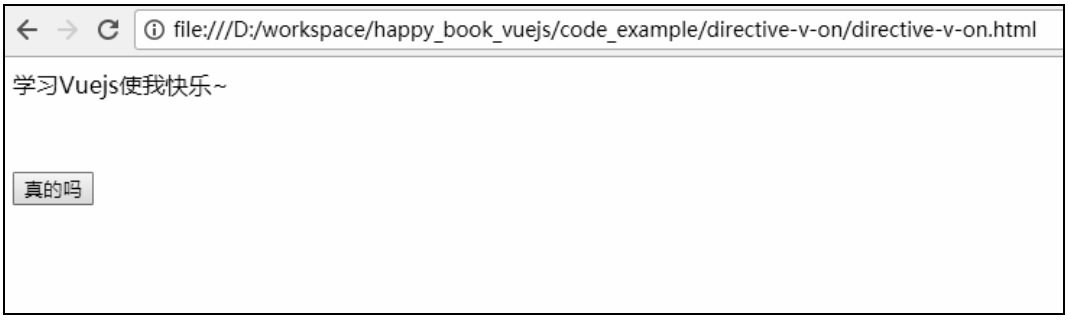

图 4-10　单击前的页面

单击后的页面如图 4-11 所示。

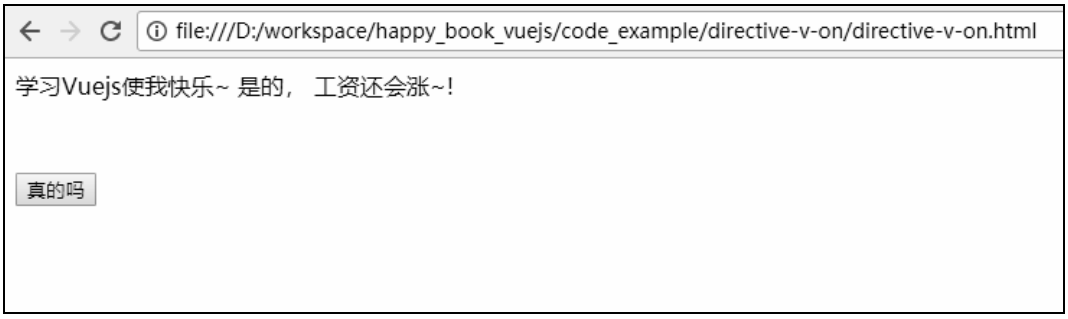

图 4-11　单击后的页面

v-on 后面可以接 HTML 的标准事件。例如：

- click（单击鼠标左键）
- dblclick（双击鼠标左键）
- contextmenu（单击鼠标右键）
- mouseover（指针移到有事件监听的元素或其子元素内）
- mouseout（指针移出元素，或者移到其子元素上）
- keydown（键盘动作：按下任意键）
- keyup（键盘动作：释放任意键）

对于 v-on 的更多说明，请参看 Event 的对应章节。

注意：v-on 可以简写，v-on:click 往往会写成@click，v-on:dblclick，也会写成@dblclick，读者看代码的时候要注意。

4.5.7　v-model 与双向绑定

v-model 往往用来做“双向绑定”（two way binding）。双向绑定的含义如下：

（1）可以通过表单（用户手动输入的值）来修改某个变量的值。

（2）可以通过程序的运算来修改某个变量的值，并且影响页面的展示。

双向绑定可以大大方便我们的开发。例如，在制作一个天气预报的软件时，需要在前台展示“当前温度”，如果后台代码做了某些操作后，就会发现“当前温度”发生了变化。

如果没有双向绑定，代码就会比较膨胀，做各种条件判断。有了双向绑定，就可以做到后台变量一变化，前台对于该变量的展示就会发生变化。

下面是一个完整的页面源代码。

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
```

```
    <div id='app'>
        <p> 你好， {{name}} ! </p>
        <input type='text' v-model="name" />
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                // 下面一行代码不能省略。 这里声明了 name 变量
                name: 'Vue.js（默认)'
            }
        })
    </script>
</body>
</html>
```

在上面的代码中，可以看到：

（1）使用了 <input type='text' v-model="name" />把变量 name 绑定在<input>输入框上（可以在这里看到 name 的值）

（2）使用了 你好，{{name}} ! 把变量 name 显示在页面上。

（3）在初始化中，使用 data: { name: '...' } 的方式对变量 name 进行了初始化。

使用浏览器打开该 html 页面，可以看到最初的状态如图 4-12 所示。

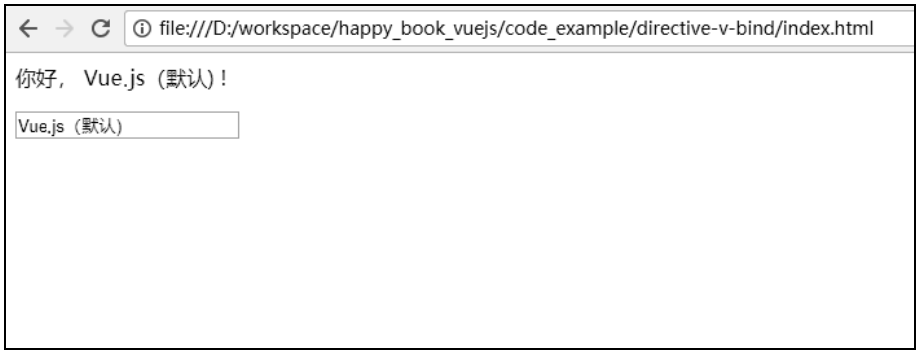

图 4-12　最初的状态

可以看到，图 4-12 中显示的文字是“你好，Vue.js!”。

然后在输入框中把内容改为“Vue.js 和 Webpack”，于是页面就发生了变化，如图 4-13 所示。

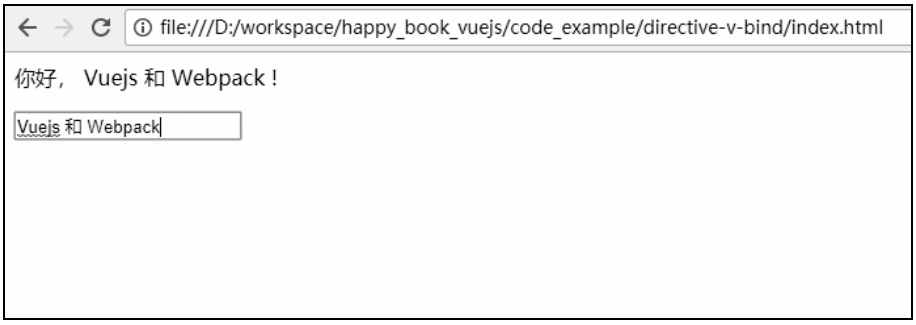

图 4-13　页面发生变化

这就说明，我们通过输入框来改变某个变量的值是成功的。

打开浏览器中的 Peveloper Tools（建议用 Chrome，Chrome 下的操作方式是按 F12 键）。在 console 中输入 app.name = "明日 Vue.js 高手"，就会看到如图 4-14 所示的效果。

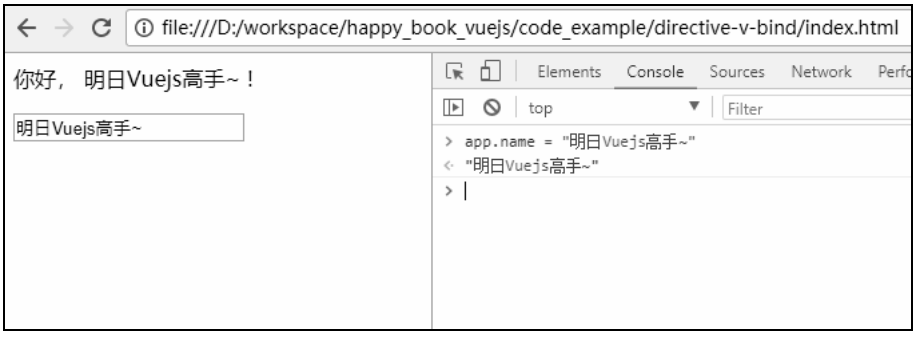

图 4-14　页面效果

这就说明，我们通过运算来改变某个变量的值是成功的。

4.6 发送 http 请求

每个 SPA 项目都要使用 http 请求，这些请求从服务器读取数据，然后:

（1）在前端页面进行展示，如论坛应用中显示文章列表。
（2）做一些逻辑判断，如注册页面需要判断某个用户名是否已经存在。
（3）做一些数据库的保存操作，如修改密码。

所以，http 请求非常重要。

在进行下面的学习之前，我们需要为当前的 SPA 项目加上 http 请求的支持。修改 src/main.js 文件，增加如下内容即可。

```
import VueResource from 'vue-resource';
Vue.use(VueResource);
```

4.6.1 调用 http 请求

Vue.js 内置了对发送 http 请求的支持，只需要在对应页面的 script 标签内加上对应的代码就可以。

例如，我们新增一个页面为“博客列表页”：src/components/BlogList.vue，其作用是从我的个人网站（http://siwei.me）上读取文章的标题并显示出来。

代码如下：

```
<template>
  <div >
    <table>
      <tr v-for="blog in blogs">
        <td></td>
      </tr>
    </table>
  </div>
</template>

<script>
export default {
  data () {
    return {
      title: '博客列表页',
      blogs: [
      ]
    }
  },
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
       console.info(response.body)
       this.blogs = response.body.blogs
    }, (response) => {
       console.error(response)
    });
  }
}
</script>
```

```
<style >

td {
  border-bottom: 1px solid grey;
}
</style>
```

在上面的代码中， 我们先看 <script/>代码段：

```
export default {
  data () {
    return {
      title: '博客列表页',
      blogs: [
      ]
    }
  },
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
       console.info(response.body)
       this.blogs = response.body.blogs
    }, (response) => {
       console.error(response)
    });
  }
}
```

上面的代码先定义了两个变量：title 和 blogs，然后定义了一个 mounted 方法。该方法表示当页面加载完毕后应该做哪些事情，是一个钩子方法。

```
this.$http.get('api/interface/blogs/all').then((response) => {
   console.info(response.body)
   this.blogs = response.body.blogs
}, (response) => {
   console.error(response)
});
```

上面的代码是发起 http 请求的核心代码。访问的接口地址是 api/interface/blogs/all，然后

使用 then 方法做下一步的事情， then 方法接受两个函数作为参数，第一个是成功后做什么，第二个是失败后做什么。

成功后的代码如下：

```
this.blogs = response.body.blogs
```

然后在对应的视图部分显示：

```
<tr v-for="blog in blogs">
  <td></td>
</tr>
```

4.6.2 远程接口的格式

在远程服务器上读取个人博客标题的接口已经提前做好了，是 http://siwei.me/interface/blogs/all。其内容如下：

```
{
  blogs: [
    {
      id: 1516,
      title: "网络安全资源",
      created_at: "2018-06-24T09:36:20+08:00"
    },
    {
      id: 1515,
      title: "github - 邀请伙伴后，需要修改权限",
      created_at: "2018-06-20T15:03:33+08:00"
    },
    {
      id: 1514,
      title: "ruby/rails - 根据浏览器的语言自动识别",
      created_at: "2018-06-19T08:28:44+08:00"
    },
    {
      id: 1513,
      title: "google cloud - 申请 VM 的经验",
      created_at: "2018-06-09T16:42:08+08:00"
    },
    {
      id: 1512,
      title: "验证码 - 使用 geetest 或网易云盾提供的动态二维码",
```

```
      created_at: "2018-06-07T09:28:14+08:00"
    }
    // 更多内容……
  ]
}
```

在浏览器中打开后，页面效果如图 4-13 所示（使用 jsonview 插件做了 json 的代码格式化）。

图 4-13　页面效果

4.6.3　设置 Vue.js 开发服务器的代理

正常来说，JavaScript 在浏览器中是无法发送跨域请求的，我们需要在 Vue.js 的“开发服务器”上做转发配置。

修改 config/index.js 文件，增加下列内容：

```
module.exports = {
  dev: {
    proxyTable: {
      '/api': {       // 1. 对所有以 "/api" 开头的 url 做处理
        target: 'http://siwei.me',   // 3. 转发到 siwei.me 上
        changeOrigin: true,
        pathRewrite: {
          '^/api': ''  // 2. 把 url 中的 "/api" 去掉
```

```
      }
    }
  },
}
```

上面的代码做了以下三件事。

（1）对所有以 "/api" 开头的 url 做处理。

（2）把 url 中的 "/api" 去掉。

（3）把新的 url 请求转发到 siwei.me 上。

例如：

- 原请求：http://localhost:8080/api/interface/blogs/all。
- 新请求：http://siwei.me/interface/blogs/all。

注 意

以上代理服务器内容只能在“开发模式”下才能使用。在生产模式下，只能靠服务器的 nginx 特性来解决 js 跨域问题。

修改后的 config/index.js 文件的完整内容如下：

```
var path = require('path')

module.exports = {
  build: {
    env: require('./prod.env'),
    index: path.resolve(__dirname, '../dist/index.html'),
    assetsRoot: path.resolve(__dirname, '../dist'),
    assetsSubDirectory: 'static',
    assetsPublicPath: '/',
    productionSourceMap: true,
    productionGzip: false,
    productionGzipExtensions: ['js', 'css'],
    bundleAnalyzerReport: process.env.npm_config_report
  },
  dev: {
    env: require('./dev.env'),
    port: 8080,
    autoOpenBrowser: true,
    assetsSubDirectory: 'static',
    assetsPublicPath: '/',

    proxyTable: {
      '/api': {
        target: 'http://siwei.me',
        changeOrigin: true,
```

```
      pathRewrite: {
        '^/api': ''
      }
    }
  },

  cssSourceMap: false
 }
}
```

重启服务器，可以看到转发设置已经生效。

```
$ npm run dev
...
[HPM] Proxy created: /api  ->  http://siwei.me
[HPM] Proxy rewrite rule created: "^/api" ~> ""
> Starting dev server...
...
```

4.6.4　打开页面，查看 http 请求

接下来，访问 http://localhost:8080/#/blogs/ 。

打开 chrome developer tools 就可以看到，"Network"中已经有请求发出去了，如图 4-14 所示的结果。

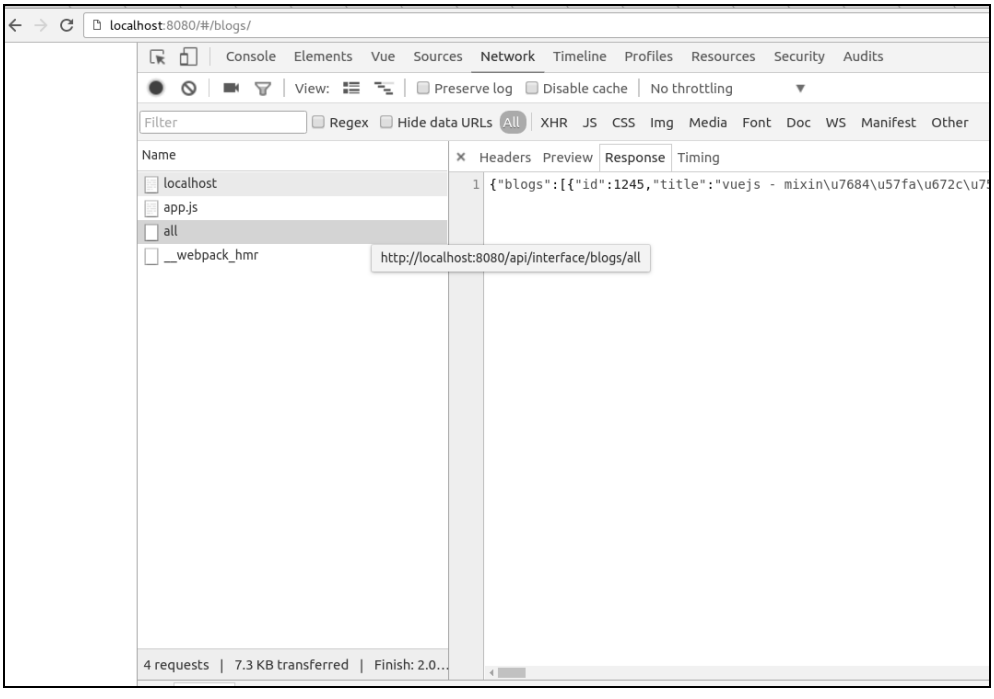

图 4-14　发出请求后

也可以直接在浏览器中输入要打开的链接，结果（该浏览器使用了 json view 插件）如图 4-15 所示。

图 4-15　直接打开链接

4.6.5　把结果渲染到页面中

在 export 代码段中有以下两个部分。

```
<script>
export default {
  data () { },
  mounted() { }
}
</script>
```

实际上，在上面代码中：

- data 方法用于“声明页面会出现的变量”，并且赋予初始值。
- mounted 表示页面被 vue 渲染好之后的钩子方法，会立刻执行。

所以，我们要把发送 http 的请求写到 mounted 方法中（钩子方法还有 created，可以暂且认为 mounted 方法与 created 方法基本一样）。

```
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
      this.blogs = response.body.blogs
    }, (response) => {
      console.error(response)
    });
  }
```

在上面的代码中：

- this.$http 中的 this 表示当前的 vue 组件（即 BookList.vue）；$http 表示所有以 $开头的变量都是 vue 的特殊变量，往往是 vue 框架自带。这里的$http 就是可以发起 http 请求的对象。
- $http.get 是一个方法，可以发起 get 请求，只有一个参数即目标 url。
- then()方法来自 promise，可以把异步请求写成普通的非异步形式。第一个参数是成功后的 callback，第二个参数是失败后的 callback。
- this.blogs = response.body.blogs 中，是把远程返回的结果（json）赋予到本地。因为 JavaScript 的语言特性能直接支持 JSON ，所以才可以这样写。

然后，我们通过这个代码进行渲染。

```
<tr v-for="blog in blogs">
  <td></td>
</tr>
```

在上面的代码中：

- v-for 是一个循环语法，可以把这个元素进行循环。注意，这个叫 directive 指令，需要与标签一起使用。
- blog in blogs: 前面的 blog 是一个临时变量，用于遍历使用；后面的 blogs 是 http 请求成功后，this.blogs = ...变量。同时，this.blogs 是声明于 data 钩子方法中。
- {{blog.title}} 用于显示每个 blog.title 的值。

4.6.6　如何发起 post 请求

与 get 类似，就是第二个参数是请求的 body。

在 vue 的配置文件中（如 Webpack 项目的 src/main.js 中）增加下面一句：

```
import VueResource from 'vue-resource';
Vue.use(VueResource);
....

//增加下面这句：
Vue.http.options.emulateJSON = true;
```

目的是为了能够让发出的 post 请求不会被浏览器转换为 option 请求。然后就可以按照下面的代码发送请求了。

```
this.$http.post('api/interface/blogs/all', {title: '', blog_body: ''})
    .then((response) => {
```

```
        ...
    }, (response) => {
        ...
    });
```

关于发送 http 请求的更多内容，可查看官方文档：https://github.com/pagekit/vue-resource

4.7 不同页面间的参数传递

在普通的 web 开发中，参数传递有以下几种形式。

- ur：/another_page?id=3。
- 表单：<form>...</form>。

而在 Vue.js 中，不会产生表单的提交（会引起页面的整体刷新），有以下两种。

- url：同传统语言，参数体现在 url 中。
- Vue.js 内部的机制（无法在 url 中体现，可以认为是由 js 代码隐式实现的）。

我们用一个实际的例子说明：之前实现了“博客列表页”，接下来要实现“单击博客列表页中的某一行，就显示博客详情页”。

4.7.1 回顾：现有的接口

我们已经做好了一个接口：文章详情页。地址为：

```
/interface/blogs/show
```

该接口接收一个参数：id，使用 HTTP GET 请求进行访问。

下面是该接口的一个完整形式：http://siwei.me/interface/blogs/show?id=1244。

返回结果如下：

```
{
    "result":{
        "body":"<p>这个问题很常见，解决办法就是禁止硬件加速...</p>",
        "id":1244,
        "title":"android - 在 view pager 中的 webview，切换时，会闪烁的问题。"这个问题很常见，解决办法就是禁止硬件加速...</p>
    }
}
```

在浏览器中打开，结果如图 4-16 所示。

图 4-16　页面效果

4.7.2　显示博客详情页

我们需要新增 Vue 页面： src/components/Blog.vue，用于显示博客详情。

```
<template>
  <div >
    <div>
      <p> 标题：    </p>
      <p> 发布于：  </p>
      <div>


      </div>
    </div>
  </div>
</template>
```

```
<script>
export default {
  data () {
    return {
      blog: {}
    }
  },
  mounted() {

this.$http.get('api/interface/blogs/show?id='+this.$route.query.id).then((resp
onse) => {
      this.blog = response.body.result
    }, (response) => {
      console.error(response)
    });
  }
}
</script>

<style>
</style>
```

在上面的代码中：

- data(){ blog: {}}用于初始化 blog 页面用到的变量。
- {{blog.body}}、{{blog.title}}等用于显示 blog 相关的信息。
- mounted...中定义了发起 HTTP 的请求。
- this.$route.query.id 获取 url 中的 id 参数，如/my_url?id=333，'333'就是取到的结果。

4.7.3 新增路由

修改 src/router/index.js 文件。添加如下代码：

```
import Blog from '@/components/Blog'

export default new Router({
  routes: [
      // ...
    {
```

```
    path: '/blog',
    name: 'Blog',
    component: Blog
  }
 ]
} )
```

上面的代码就是让 Vue.js 可以对形如 http://localhost:8080/#/blog 的 url 进行处理。 对应的 vue 文件是 @/components/Blog.vue。

4.7.4　修改博客列表页的跳转方式 1：使用事件

我们需要修改博客列表页，增加跳转事件。修改 src/components/BlogList.vue，代码如下。

```
<template>
 ...
    <tr v-for="blog in blogs">
      <td @click='show_blog(blog.id)'></td>
    </tr>
 ...
</template>
<script>
export default {
 methods: {
   show_blog: function(blog_id) {
     this.$router.push({name: 'Blog', query: {id: blog_id}})
   }
 }
}
</script>
```

在上面的代码中：

- <td @click='show_blog(blog.id)'...</td>表示该<td>标签在被单击时会触发一个事件: show_blog，并且以当前正在遍历的 blog 对象的 id 作为参数。
- methods: {} 是比较核心的方法，vue 页面中用到的事件都要写在这里。
- show_blog: function...就是我们定义的方法。该方法可以通过@click="show_blog"调用。
- this.$router.push({name: 'Blog', params: {id: blog_id}})中，this.$router 是 vue 的内置对

象，表示路由。

- this.$router.push 表示让 vue 跳转，跳转到 name: Blog 对应的 vue 页面，参数是 id: blog_id 。

1. 演示结果

打开“博客列表页”，可以看到对应的文章，然后单击其中一篇文章的标题，就可以打开对应的文章详情页，如图 4-17 所示。

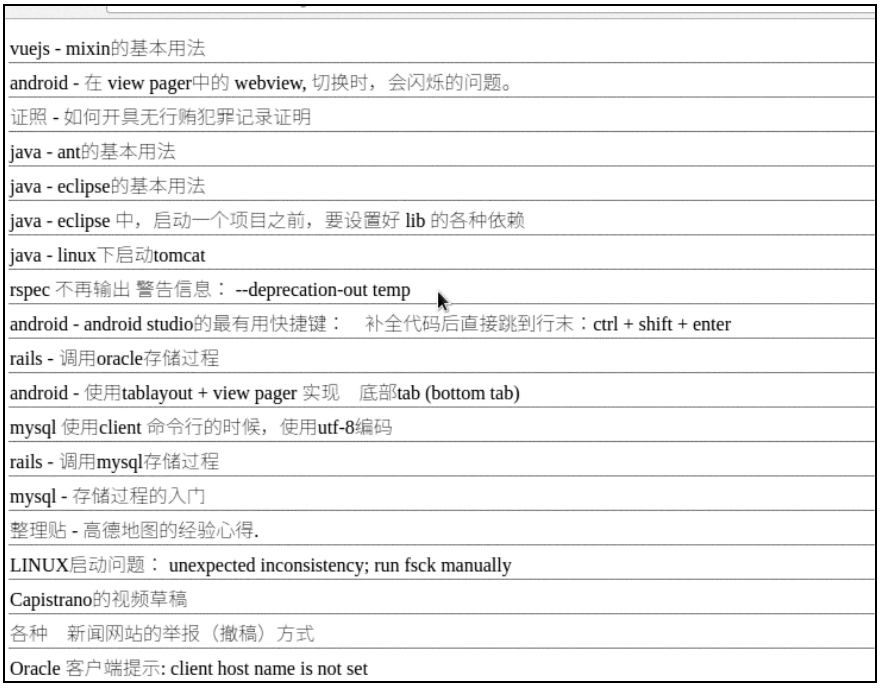

图 4-17 文章列表

2. 不经过 HTML 转义，直接打印结果

我们发现，HTML 的源代码在页面显示时被转义了，如图 4-18 所示。

图 4-18 HTML 源代码被转义

所以，我们把它修改一下，即不转义。

```
<template>
  .....
    <div v-html='blog.body'>
    </div>
  .....
</template>
```

上面的 v-html 就表示不转义。页面效果如图 4-19 所示。

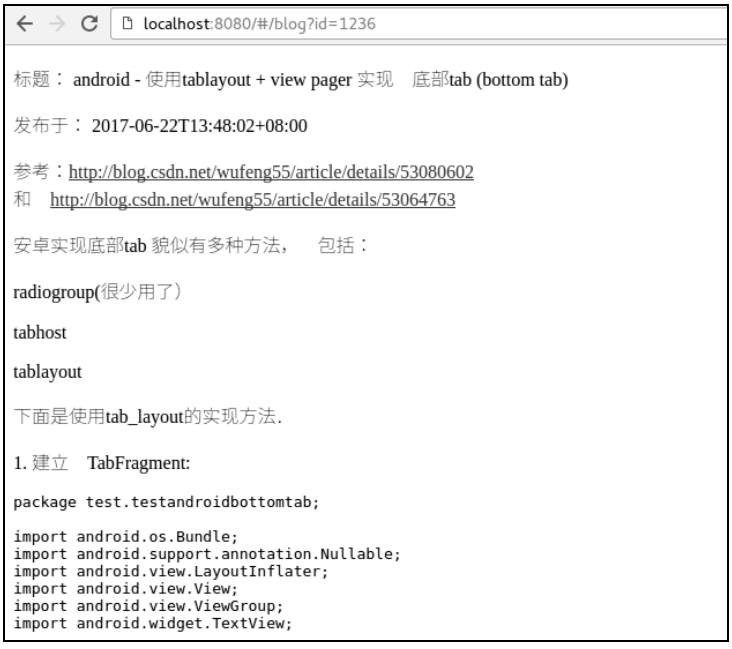

图 4-19　页面效果

4.7.5　修改博客列表页的跳转方式 2：使用 v-link

<router-link>比起<a href="...">要好一些。

因为无论是 HTML5 history 模式还是 hash 模式，其表现行为一致，所以当要切换路由模式，或者在 IE9 降级使用 hash 模式时，无须做任何变动。

在 HTML5 history 模式下，router-link 会拦截单击事件，让浏览器不再重新加载页面。

当用户在 HTML5 history 模式下使用 base 选项之后，所有的 to 属性都不需要写（基路径）了。

```
<td>
  <router-link :to="{name: 'Blog', query: {id: blog.id}}">


  </router-link>
```

```
</td>
```

然后就可以看到生成的 HTML 形如：

```
<a href="#/blog?id=1239" class="">
   1239
</a>
```

单击之后，有同样的跳转功能。感兴趣的读者，可以查看下面的链接：https://router.Vue.js.org/zh-cn/api/router-link.html。

4.8 路由

路由是所有前端框架中必须具备的元素，其定义了对于哪个 URL（页面）应该由哪个文件来处理。

在 Vue.js 中，路由专门独立成为了一个项目: vue-router。

4.8.1 基本用法

每个 vue 页面都要对应一个路由。 例如，我们要做一个“博客列表页”，就需要具备以下两个条件。

- vue 文件，如 src/components/books.vue 负责展示页面。
- 路由代码，让/#/books 与上面的 vue 文件对应。

下面的代码就是一个完整的路由文件。

```
import Vue from 'vue'
import Router from 'vue-router'

// 增加这一行，作用是引入 SayHi 这个 component
import SayHi from '@/components/SayHi'

Vue.use(Router)
export default new Router({
  routes: [

    // 增加下面几行，表示定义了  /#/say_hi 这个 url
    {
```

```
    path: '/say_hi',
    name: 'SayHi',
    component: SayHi
  },
 ]
})
```

写法是固定的，其中：

- path：定义了链接地址，如/#/say_hi。
- name 表示为这个路由加名字，方便以后引用，如 this.$router.push({name: 'SayHi'})。
- component 表示该路由由哪个 component 来处理。

4.8.2　跳转到某个路由时带上参数

有路由就会有参数，下面来看一下路由是如何处理参数的。

1. 对于普通的参数

例如：

```
routes: [
  {
    path: '/blog',
    name: 'Blog'
  },
]
```

在视图中，我们这样做：

```
<router-link :to="{name: 'Blog', query:{id: 3} }">User</router-link>
```

当用户单击这个代码生成的 html 页面时，就会触发跳转。

在<script/>中也可以这样做：

```
this.$router.push({name: 'Blog', query: {id: 3}})
```

都会跳转到/#/blog?id=3。

2. 对于在路由中声明的参数

例如：

```
routes: [
  {
```

```
    path: '/blog/:id',
    name: 'Blog'
  },
]
```

在视图中，我们这样做：

```
<router-link :to="{name: 'Blog', params: {id: 3} }">User</router-link>
```

在 script 中也可以这样做：

```
this.$router.push({name: 'Blog', params: {id: 3}})
```

都会跳转到/#/blog/3。

4.8.3 根据路由获取参数

在 Vue 的路由中，获取参数有两种方式：query 和 params。

1. 获取普通参数

对于/#/blogs?id=3 中的参数，可以这样获取：

```
this.$route.query.id  // 返回结果 3
```

2. 获取路由中定义好的参数

对于/#/blogs/3 这样的参数，可以对应的路由应该是：

```
  routes: [
    {
      path: '/blogs/:id',  // 注意这里的 :id
      ...
    },
  ]
```

这个 named path 就可以通过下面的代码来获取 id。

```
this.$route.params.id  // 返回结果 3
```

4.9 使用样式

样式用起来特别简单，直接写到<style>段落里面即可。代码如下：

```
<template>
  <div class='hi'>
    Hi Vue!
  </div>
</template>

<script>
export default {
  data () {
    return { }
  }
}
</script>

<style>
.hi {
  color: red;
  font-size: 20px;
}
</style>
```

使用浏览器打开上述代码，就可以看到一个红色的，字体大小为 20px 的 Hi Vue!，如图 4-20 所示。

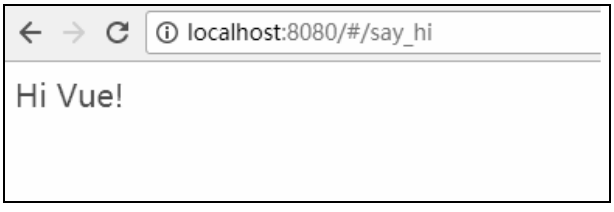

图 4-20　页面效果

1. 使用全局

```
<style >
td {
  border-bottom: 1px solid grey;
}
</style>
```

2. 使用局部的 CSS

```
<style scoped>
td {
  border-bottom: 1px solid grey;
}
</style>
```

这段 CSS 只对当前的 component 适用。

也就是说，当我们有两个不同的页面（page1 和 page2）时，如果两个页面中都定义了某个样式（如上面的 td），是不会互相冲突的。

因为 Vue.js 会这样解析：

```
page1 的 DOM:
<div data-v-7cfd41e ... ></div>

page2 的 DOM:
<div data-v-3389dfw ... ></div>
```

而我们使用的"scoped style"就可以存在于不同的页面（component）上了。

4.10 双向绑定

现在，双向绑定的概念越来越普及。

在 Angular 出现的时候，双向绑定就作为宣传的王牌概念，现在几乎每个 js 前端框架就有该功能。它的概念是某个变量定义于 <script/>，需要展现在 <template/>中的话：

- 如果在代码层面进行修改，页面的值就会发生变化。
- 如果在页面进行修改（如在 input 标签中），代码的值就会发生变化。

在我们的项目中，增加一个 vue 页面：src/components/TwoWayBinding.vue。

```
<template>
  <div>
    <!-- 显示  this.my_value 变量 -->
    <p>页面上的值:  </p>

    <p> 通过视图层，修改 my_value: </p>
```

```
    <input v-model="my_value" style='width: 400px'/>

    <hr/>
    <input type='button' @click="change_my_value_by_code()" value='通过控制代码修改 my_value'/>
    <hr/>
    <input type='button' @click="show_my_value()" value='显示代码中的 my_value'/>
  </div>
</template>

<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    show_my_value: function(){
      alert('my_value: ' + this.my_value);
    },
    change_my_value_by_code: function(){
      this.my_value += ",  在代码中做修改, 666."
    }
  }
}
</script>
```

在上面的代码中显示定义了一个变量：my_value，该变量可以在 <script/>中访问和修改，也可以在<template/>中访问和修改。

- 在代码（<script/>）中访问，就是 this.my_value。
- 在视图（<template/>）中访问，就是<input v-model=my_value />。

这个就是双向绑定的方法。

接下来，修改路由文件: src/router/index.js。

```
import TwoWayBinding from '@/components/TwoWayBinding'
```

```
export default new Router({
  routes: [
    {
      path: '/two_way_binding',
      name: 'TwoWayBinding',
      component: TwoWayBinding
    }
  ]
})
```

然后可以使用浏览器访问路径：http://localhost:8080/#/two_way_binding。

效果 1：通过页面修改 js 代码的值，可以看到，一旦代码中的 my_value 发生改变，视图中的 my_value 就发生变化，如图 4-21 所示。

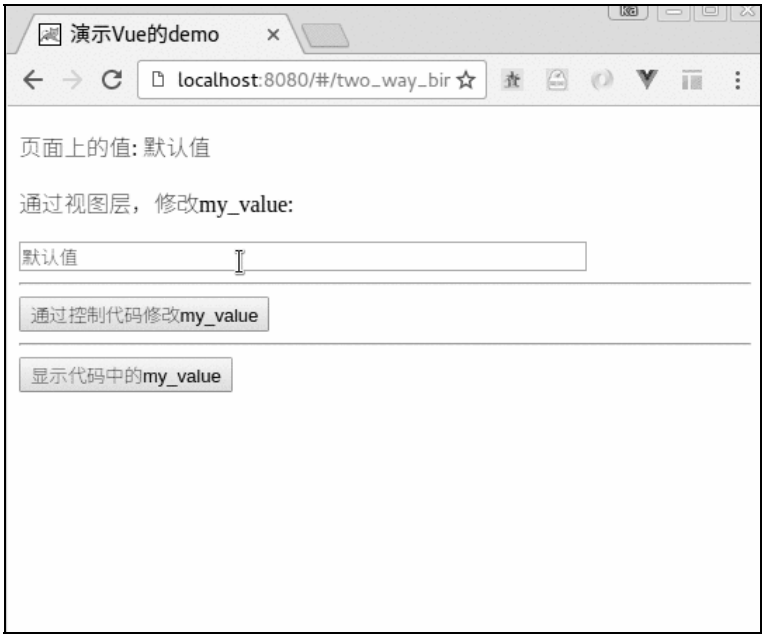

图 4-21　效果 1

效果 2：通过代码层面的改动影响页面的值，如图 4-22 所示。

图 4-22　效果 2

这个特性是 Vue.js 自带的，我们不需要刻意学习，只需要知道它可以达到这个目就可以了。读者以后会发现，这种思想和现象在 Vue.js 等前端框架中特别常用。

4.11 表单项目的绑定

所有的表单项，无论是<input/>还是<textarea/>，基本上都需要使用 v-model 来绑定。

1. 表单项 input、textarea、select 等

使用 v-model 来绑定输入项。

```
<input v-model="my_value" style='width: 400px'/>
```

可以在代码中获取到 this.my_value 的值。

3. 表单项的完整例子

```
<template>
  <div>

    input:  <input type='text' v-model="input_value"/>,
    输入的值：
    <hr/>
```

```
    text area:  <textarea v-model="textarea_value"></textarea>,
    输入的值：
    <hr/>
    radio:
    <input type='radio' v-model='radio_value' value='A'/> A,
    <input type='radio' v-model='radio_value' value='B'/> B,
    <input type='radio' v-model='radio_value' value='C'/> C,
    输入的值：

    <hr/>
    checkbox:
    <input type='checkbox' v-model='checkbox_value'
      v-bind:true-value='true'
      v-bind:false-value='false'
      /> ,
    输入的值：

    <hr/>
    select:
    <select v-model='select_value'>
      <option v-for="e in options" v-bind:value="e.value">

      </option>
    </select>
    输入的值：

  </div>
</template>

<script>
export default {
  data () {
    return {
      input_value: '',
      textarea_value: '',
      radio_value: '',
```

```
    checkbox_value: '',
    select_value: 'C',
    options: [
      {
        text: '红烧肉', value: 'A'
      },
      {
        text: '囊包肉', value: 'B'
      },
      {
        text: '水煮鱼', value: 'C'
      }
    ]
  }
 },
 methods: {
 }
}
</script>
```

对于 select 的 option，使用 v-bind:value 来绑定 option 的值，效果如图 4-23 所示。

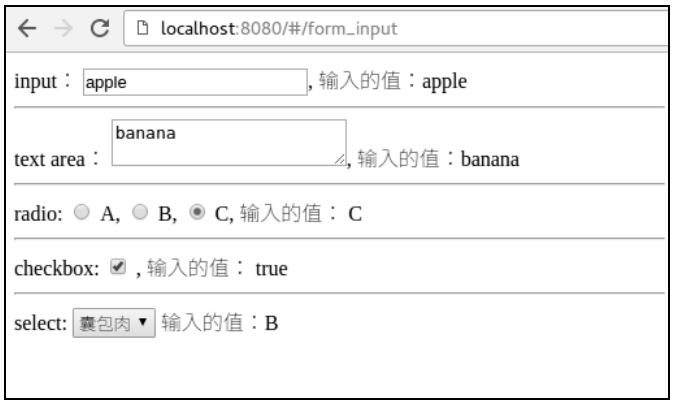

图 4-23　页面效果

4. Modifiers （后缀词）

（1）.lazy

在用户对某个文本框做输入的时候，文本框中的值不会随着用户按下的每一个键立刻发生变化，而是等焦点彻底离开文本框后（触发 blur()事件后）触发视图中值的变化。

使用方式如下：

```
<input type='text' v-model.lazy="input_value"/>
```

这个可以用在某些需要等待用户输入完字符串再需要给出反应的情况，如“搜索”。

（2）.number

强制要求输入数字。使用方式如下：

```
<input type='text' v-model.lazy="input_value" type="number"/>
```

（3）.trim

强制对输入的值去掉前后的空格。使用方式如下：

```
<input type='text' v-model.trim="input_value" />
```

4.12 表单的提交

在任何 Single Page App 中，js 代码都不会产生一个传统意义的 form 表单提交（这会引起整个页面的刷新），一般用事件来实现（桌面开发思维）。

例如，在远程有一个接口，可以接受别人的留言：

- URL：http://siwei.me/interface/blogs/add_comment。
- 参数：content（留言的内容）。
- 请求方式：POST。
- 返回结果的例子如下。

```
{"result":"ok","content":"(留言的内容)"}
```

例如，下面的代码就是把输入的表单提交到后台。

新增加一个/src/components/FormSubmit.vue 文件，内容如下：

```
<template>
  <div>
    <textarea v-model='content'>
    </textarea>
    <br/>
    <input type='button' @click='submit' value='留言'/>
  </div>
</template>
<script>

export default {
```

```
  data () {
    return {
      content: ''
    }
  },
  methods: {
    submit: function(){
      this.$http.post('/api/interface/blogs/add_comment',
        {
          content: this.content
        }
      )
      .then((response) => {
          alert("提交成功!, 刚才提交的内容是：" + response.body.content)
        },
        (response) => {
          alert("出错了")
        }
      )
    }
  }
}
</script>
```

从上面的代码中可以看到：

（1）下面的代码是待输入的表单项。

```
    <textarea v-model='content'>
    </textarea>
```

（2）下面的代码则是按钮被单击后触发提交表单的函数 submit。

```
<input type='button' @click='submit' value='留言'/>
```

（3）下面的代码定义了提交表单的函数 submit。

```
    submit: function(){
      this.$http.post('/api/interface/blogs/add_comment',
        {
```

```
        content: this.content
      }
    )
    .then((response) => {
        alert("提交成功!, 刚才提交的内容是: " + response.body.content)
      },
      (response) => {
        alert("出错了")
      }
    )
  }
```

- this.$http.post 表示发起的 HTTP 类型是 post。
- post 函数的第一个参数是 url，第二个参数是 json，{ content: this.content}代表了要提交的数据。
- then 函数的处理同 HTTP get 请求。

接下来，修改路由 src/router/index.js 文件，增加内容如下：

```
import FormSubmit from '@/components/FormSubmit'

export default new Router({
  routes: [
    {
      path: '/form_submit',
      name: 'FormSubmit',
      component: FormSubmit
    }
  ]
} )
```

访问 url: http://localhost:8080/form_submit ，输入一段字符串，如图 4-24 所示。

图 4-24　输入一段字符串

单击提交按钮就可以看到，内容已经提交，并且得到了返回的 response，触发了 alert，如图 4-25 所示。

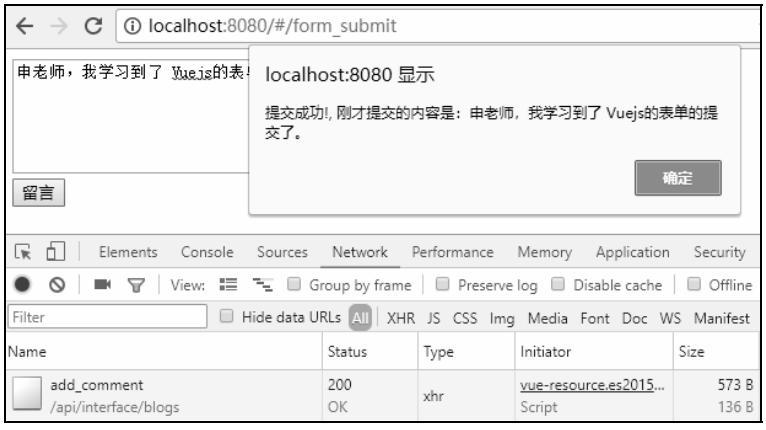

图 4-25　提交成功

查看一下返回的 JSON：

```
{"result":"ok","content":"\u7533\u8001\u5e08\uff0c\u6211\u5b66\u4e60\u5230\u4e
86 Vue.js\u7684\u8868\u5355\u7684\u63d0\u4ea4\u4e86\u3002"}
```

至此，完成了一个完整的输入表单、提交表单的过程。

4.13 Component 组件

组件是 Vue.js 中最重要的部分之一，学好组件知识需要一定的时间投入。在 Webpack 项目中，每一个页面文件（.vue）都可以认为是一个组件。

在 Vue.js 1.x 中，组件与视图是分别放到不同的文件夹下面的，在 Vue.js 2.8 以后，所有的视图文件都保存在 components 目录下。

可见，组件的概念已经越来越普及了。

注　意

这里的内容，与官方文档中的“原始组件”不一样。本章的内容，仅使用于 Webpack 项目中的组件，官方网站中对应的文档是单文件组件。

4.13.1　如何查看文档

先快速查看官方文档中关于“原始组件”的页面：https://cn.Vue.js.org/v2/guide/single-file-components.html，对所有的概念有所了解，因为这个“原始组件”的开发环境与 webpack 下项目的开发环境不太一样，所以很多以 webpack 作为入门的读者（如本书读者）会感到迷

茫。再看“单文件组件”：https://cn.Vue.js.org/v2/guide/single-file-components.html，就可以对 webpack 项目下的组件有清晰地认识。

遇到问题之后，再看“原始组件”，这里包括很多 API 级别的概念和解释。

4.13.2 Component 的重要作用：重用代码

我们可以想象一个场景：有两个页面，每个页面的头部都有一张 Logo 图片。如果每次都写成原始 HTML 的话，代码就会比较重复。页面 1 的代码如下：

```
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
fyfkzhs9232258.jpg'>
  </div>
  页面 1 的其他代码
```

页面 2 的代码如下：

```
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
fyfkzhs9232258.jpg'>
  </div>
  页面 2 的其他代码
```

因此，我们应该把这段代码抽取出来成为一个新的组件(component)。

4.13.3 组件的创建

新建一个 src/components/Logo.vue 文件。

```
<template>
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png'/>
  </div>
</template>
```

该文件中定义了一个比较简单的 component。然后修改对应的页面：

```
<template>
  <div >
    <my-logo>
    </my-logo>
       ...
```

```
</template>

<script>
import MyLogo from '@/components/Logo'

export default {
    ...
  components: {
    MyLogo
  }
}
```

上面代码中的 components: { MyLogo }必须是这个写法，等同于：

```
components: {
  MyLogo: MyLogo   // 前面的 MyLogo 是 template 中的名字
                   // 后面的 MyLogo 是 import 进来的代码
}
```

虽然上面代码中定义的组件名字为 MyLogo，但是在<template/>中使用时需要写为<my-logo></my-logo>。

保存代码并刷新一次，发现两个页面都发生了变化，如图 4-26 所示。

图 4-26　两个页面都发生了变化

4.13.4 向组件中传递参数

如果希望两个页面中都有一个 title，内容却不同，该怎么办呢？这时就需要向 Component 传递参数了。

声明组件时，需要修改 src/components/Logo.vue 文件。

```
<template>
  <div class='logo'>
    <h1></h1>
        ...
  </div>
</template>

<script>
export default {
  props: ['title']    // 加上这个声明.
}
</script>
```

可以看到，在上面的代码中增加了以下几行代码。

```
export default {
  props: ['title']
}
```

上面的代码表示为该 component 增加了一个 property（属性），属性的名字为 title。

1. 组件接收字符串作为参数

在调用时传递字符串就可以了。

```
<my-logo title="博客列表页">
</my-logo>
```

2. 组件接收变量作为参数

如果要传递的参数是一个变量，就可以这样写：

```
<template>
    <my-logo :title="title">
    </my-logo>
    <input type='button' @click='change_title' value='单击修改标题'/><br/>
</template>
```

```
<script>
export default {
  data: function() {
    return {
      title: '博客列表页',
    }
  },
  methods: {
    change_title: function(){
      this.title = '好多文章(标题被代码修改过了)'
    }
  },
</script>
```

如图 4-27 所示。

图 4-27　标题已被修改

4.13.5 脱离 Webpack，在原生 Vue.js 中创建 component

在原生 Vue.js 中创建 component 的过程非常简单。代码如下：

```
<html>
<head>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
  <div id='app'>
    <study-process></study-process>
  </div>
  <script>
    Vue.component('study-process', {
      data: function () {
        return {
          count: 0
        }
      },
      template: '<button v-on:click="count++">我学习到了第  章.</button>'
    })
    var app = new Vue({
      el: '#app',
      data: {
      }
    })
  </script>
</body>
</html>
```

该代码首先声明了一个 component：

```
    Vue.component('study-process', {
      data: function () {
        return {
          count: 0
        }
      },
```

```
    template: '<button v-on:click="count++">我学习到了第  章.</button>'
  })
```

可以看出，该 component 定义了一个 data 代码段，其中有一个 count 变量，然后定义一个 template 段落即可。