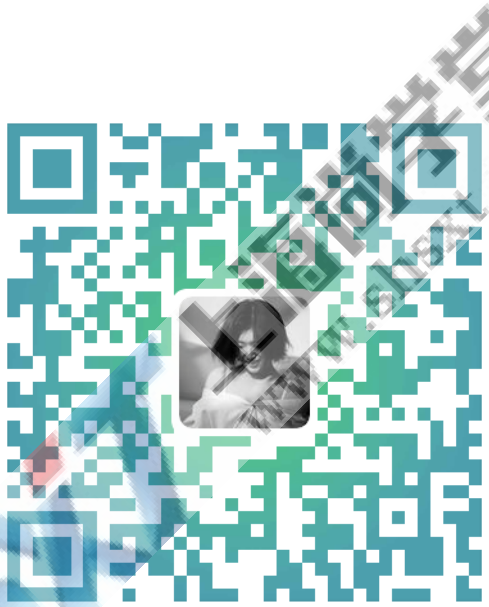
上海尚学堂
www.shsxt.com

上海尚学堂高级架构课程

正所谓“授人以鱼不如授人以渔”，你们想要的 Java 学习资料来啦！

不管你是学生，还是已经步入职场的同行，希望你们都要珍惜眼前的学习机会，奋斗没有终点，知识永不过时。

扫描下方二维码即可领取



乐字节晓啡

VUE

学习目标

前端知识体系

前端三要素

- HTML（结构）：超文本标记语言（Hyper Text Markup Language），决定网页的结构和内容
- CSS（表现）：层叠样式表（Cascading Style Sheets），设定网页的表现样式
- JavaScript（行为）：是一种弱类型脚本语言，其源代码不需经过编译，而是由浏览器解释运行，用于控制网页的行为

结构层（HTML）

略

表现层（CSS）

CSS 层叠样式表是一门标记语言，并不是编程语言，因此不可以自定义变量，不可以引用等，换句话说就是不具备任何语法支持，它主要缺陷如下：

强大，比如无法嵌套书写，导致模块化开发中需要书写很多重复的选择器；
- 没有变量和合理的样式复用机制，使得逻辑上相关的属性值必须以字面量的形式重复输出，导致难以维护；

这就导致了我们在工作中无端增加了许多工作量。为了解决这个问题，前端开发人员会使用一种称之为**【CSS 预处理器】**的工具，提供 CSS 缺失的样式层复用机制、减少冗余代码，提高样式代码的可维护性。大大提高了前端在样式上的开发效率。

什么是 CSS 预处理器

CSS 预处理器定义了一种新的语言，其基本思想是，用一种专门的编程语言，为 CSS 增加了一些编程的特性，将 CSS 作为目标生成文件，然后开发者就只要使用这种语言进行 CSS 的编码工作。转化成通俗易懂的话来说就是"**用一种专门的编程语言，进行 Web 页面样式设计，再通过编译器转化为正常的 CSS 文件，以供项目使用**"。

常用的 CSS 预处理器有哪些

- SASS：基于 Ruby，通过服务端处理，功能强大。解析效率高。需要学习 Ruby 语言，上手难度高于 LESS。
- LESS：基于 NodeJS，通过客户端处理，使用简单。功能比 SASS 简单，解析效率也低于 SASS，但在实际开发中足够了，所以我们后台人员如果需要的话，建议使用 LESS。

行为层（JavaScript）

JavaScript 一门弱类型脚本语言，其源代码在发往客户端运行之前不需经过编译，而是将文本格式的字符代码发送给浏览器由浏览器解释运行。

Native 原生 JS 开发

原生 JS 开发，也就是让我们按照 **【ECMAScript】** 标准的开发方式，简称是 ES，特点是所有浏览器都支持。截止到目前，ES 标准已发布如下版本：

- ES3
- ES4（内部，未正式发布）
- ES5（全浏览器支持）
- ES6（常用，当前主流版本）
- ES7
- ES8
- ES9（草案阶段）

区别就是逐步增加新特性。

TypeScript 微软的标准

TypeScript 是一种由微软开发的自由和开源的编程语言。它是 JavaScript 的一个超集，而且本质上向这个语言添加了可选的静态类型和基于类的面向对象编程。由安德斯·海尔斯伯格（C#、Delphi、TypeScript 之父；.NET 创立者）主导。

该语言的特点就是除了具备 ES 的特性之外还纳入了许多不在标准范围内的新特性，所以会导致很多浏览器不能直接支持 TypeScript 语法，需要编译后（编译成 JS）才能被浏览器正确执行。

JavaScript 框架

- jQuery：大家熟知的 JavaScript 框架，优点是简化了 DOM 操作，缺点是 DOM 操作太频繁，影响前端性能；在前端眼里使用它仅仅是为了兼容 IE6、7、8；
- Angular：Google 收购的前端框架，由一群 Java 程序员开发，其特点是将后台的 MVC 模式搬到了前端并增加了模块化开发的理念，与微软合作，采用 TypeScript 语法开发；对后台程序员友好，对前端程序员不太友好；最大的缺点是版本迭代不合理（如：1代 -> 2代，除了名字，基本就是两个东西）

Facebook 出品，一款高性能的 JS 前端框架；特点是提出了新概念 **【虚拟 DOM】** 用于减少真实 DOM 操作，在内存中模拟 DOM 操作，有效的提升了前端渲染效率；缺点是使用复杂，因为需要额外学习一门 **【JSX】** 语言；

- `Vue`：一款渐进式 JavaScript 框架，所谓渐进式就是逐步实现新特性的意思，如实现模块化开发、路由、状态管理等新特性。其特点是综合了 Angular（模块化） 和 React（虚拟 DOM） 的优点；
- `Axios`：前端通信框架；因为 `Vue` 的边界很明确，就是为了处理 DOM，所以并不具备通信能力，此时就需要额外使用一个通信框架与服务器交互；当然也可以直接选择使用 jQuery 提供的 AJAX 通信功能；

UI 框架

- Ant-Design：阿里巴巴出品，基于 React 的 UI 框架
- ElementUI：饿了么出品，基于 Vue 的 UI 框架
- Bootstrap：Twitter 推出的一个用于前端开发的开源工具包
- AmazeUI：又叫“妹子 UI”，一款 HTML5 跨屏前端框架

JavaScript 构建工具

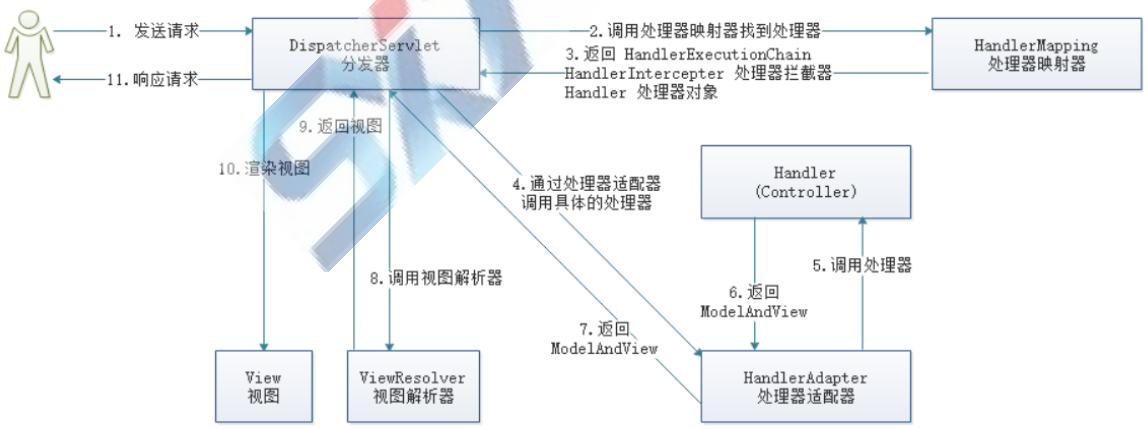
- Babel：JS 编译工具，主要用于浏览器不支持的 ES 新特性，比如用于编译 TypeScript
- WebPack：模块打包器，主要作用是打包、压缩、合并及按序加载

前后分离的演变史

后端为主的 MVC 时代

为了降低开发的复杂度，以后端为出发点，比如：Struts2、SpringMVC 等框架的使用，就是后端的 MVC 时代;

以 `SpringMVC` 流程为例：



- 发起请求到前端控制器(`DispatcherServlet`)
- 前端控制器请求 `HandlerMapping` 查找 `Handler`，可以根据 `xml` 配置、注解进行查找
- 处理器映射器 `HandlerMapping` 向前端控制器返回 `Handler`
- 前端控制器调用处理器适配器去执行 `Handler`
- 处理器适配器去执行 `Handler`
- `Handler` 执行完成给适配器返回 `ModelAndView`
- 处理器适配器向前端控制器返回 `ModelAndView`，`ModelAndView` 是 `SpringMVC` 框架的一个底层对象，包括 `Model` 和 `View`
- 前端控制器请求视图解析器去进行视图解析，根据逻辑视图名解析成真正的视图(`JSP`)
- 视图解析器向前端控制器返回 `View`
- 前端控制器进行视图渲染，视图渲染将模型数据(在 `ModelAndView` 对象中)填充到 `request` 域

器向用户响应结果

优点

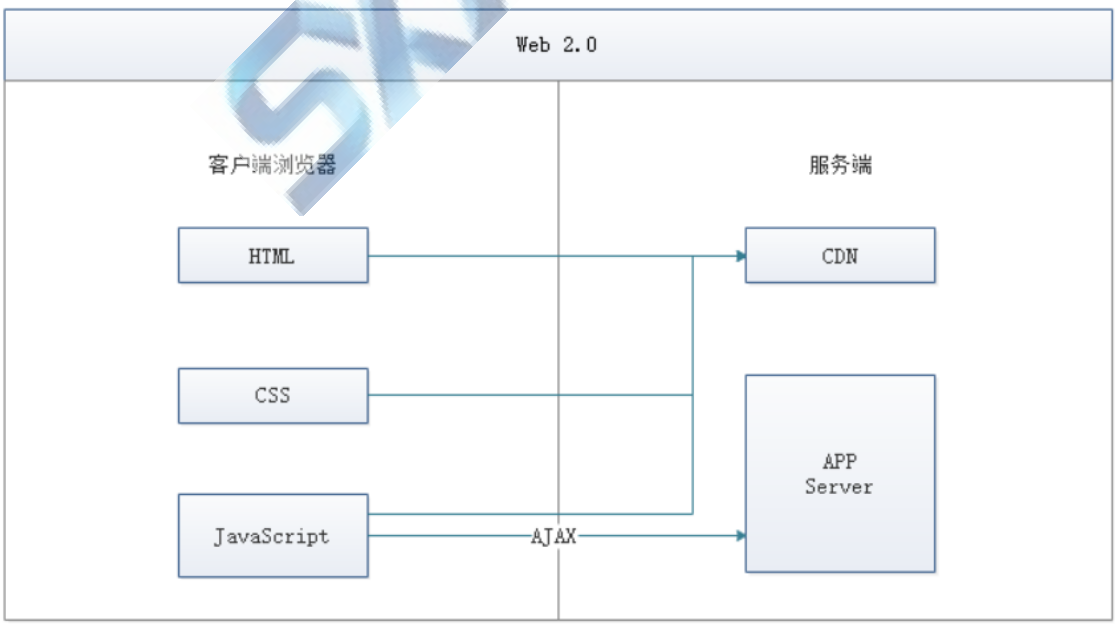
MVC 是一个非常好的协作模式，能够有效降低代码的耦合度，从架构上能够让开发者明白代码应该写在哪里。为了让 View 更纯粹，还可以使用 Thymeleaf、Freemarker 等模板引擎，使模板里无法写入 Java 代码，让前后端分工更加清晰。

缺点

- 前端开发重度依赖开发环境，开发效率低，这种架构下，前后端协作有两种模式：
  - 第一种是前端写 DEMO，写好后，让后端去套模板。好处是 DEMO 可以本地开发，很高效。不足是还需要后端套模板，有可能套错，套完后还需要前端确定，来回沟通调整的成本比较大；
  - 另一种协作模式是前端负责浏览器端的所有开发和服务器端的 View 层模板开发。好处是 UI 相关的代码都是前端去写就好，后端不用太关注，不足就是前端开发重度绑定后端环境，环境成为影响前端开发效率的重要因素。
- 前后端职责纠缠不清：模板引擎功能强大，依旧可以通过拿到的上下文变量来实现各种业务逻辑。这样，只要前端弱势一点，往往就会被后端要求在模板层写出不少业务代码。还有一个很大的灰色地带是 `Controller`，页面路由等功能本应该是前端最关注的，但却是由后端来实现。`Controller` 本身与 `Model` 往往也会纠缠不清，看了让人咬牙的业务代码经常会出现在 `Controller` 层。这些问题不能全归结于程序员的素养，否则 JSP 就够了。
- 对前端发挥的局限性：性能优化如果只在前端做空间非常有限，于是我们经常需要后端合作，但由于后端框架限制，我们很难使用 【Comet】、【BigPipe】 等技术方案来优化性能。

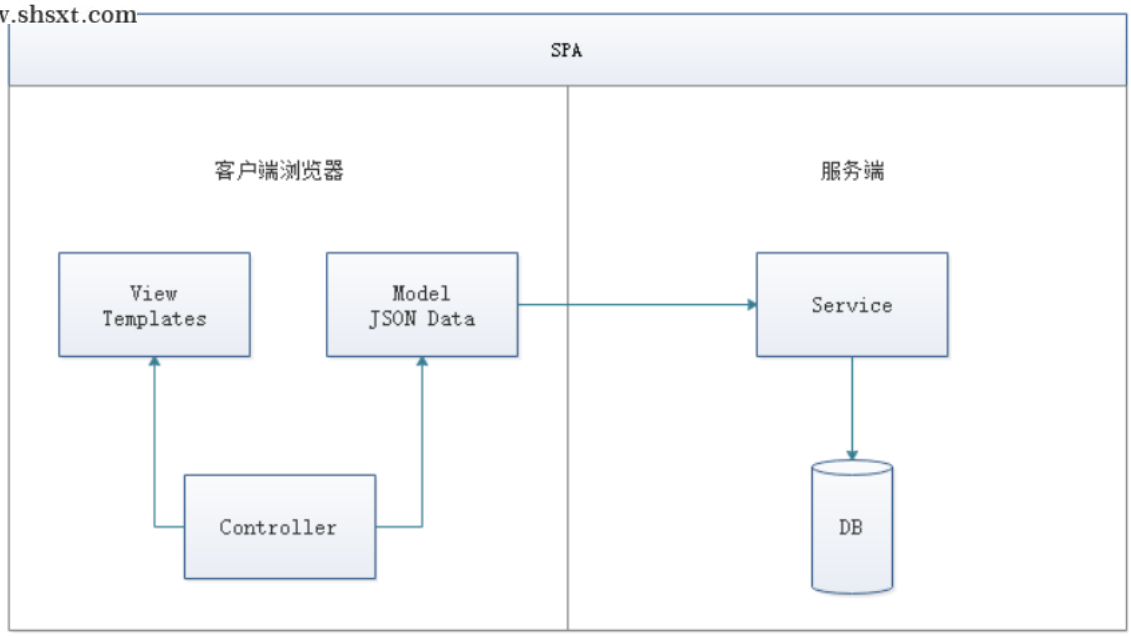
基于 AJAX 带来的 SPA 时代

时间回到 2005 年 `AJAX`（Asynchronous JavaScript And XML，异步 JavaScript 和 XML，老技术新用法）被正式提出并开始使用 `CDN` 作为静态资源存储，于是出现了 JavaScript 王者归来（在这之前 JS 都是用来在网页上贴狗皮膏药广告的）的 SPA（Single Page Application）单页面应用时代。



优点

这种模式下，**前后端的分工非常清晰，前后端的关键协作点是 `AJAX` 接口。**看起来是如此美妙，但回过头来看看的话，这与 JSP 时代区别不大。复杂度从服务端的 JSP 里移到了浏览器的 JavaScript，浏览器端变得很复杂。类似 Spring MVC，**这个时代开始出现浏览器端的分层架构**：



缺点

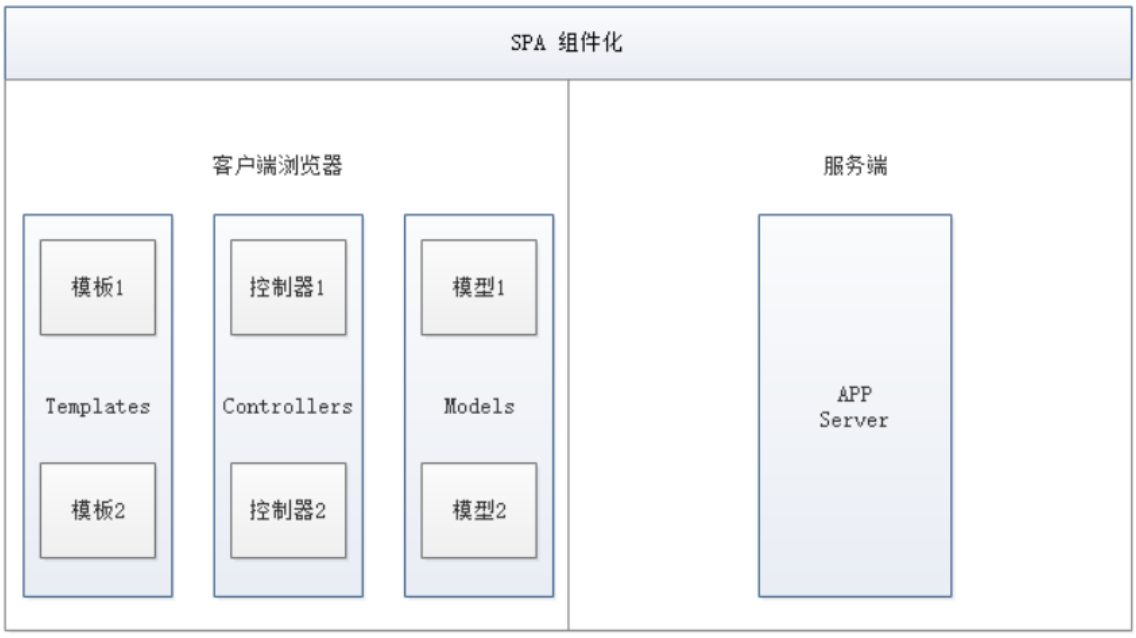
- **前后端接口的约定：** 如果后端的接口一塌糊涂，如果后端的业务模型不够稳定，那么前端开发会很痛苦；不少团队也有类似尝试，通过接口规则、接口平台等方式来做。**有了和后端一起沉淀的** `接口规则` **，还可以用来模拟数据，使得前后端可以在约定接口后实现高效并行开发。**
- **前端开发的复杂度控制：** SPA 应用大多以功能交互型为主，JavaScript 代码过十万行很正常。大量 JS 代码的组织，与 View 层的绑定等，都不是容易的事情。

前端为主的 MV* 时代

此处的 MV* 模式如下：

- MVC（同步通信为主）：Model、View、Controller
- MVP（异步通信为主）：Model、View、Presenter
- MVVM（异步通信为主）：Model、View、ViewModel

为了降低前端开发复杂度，涌现了大量的前端框架，比如：`AngularJS`、`React`、`Vue.js`、`EmberJS` 等，这些框架总的原则是先按类型分层，比如 Templates、Controllers、Models，然后再在层内做切分，如下图：



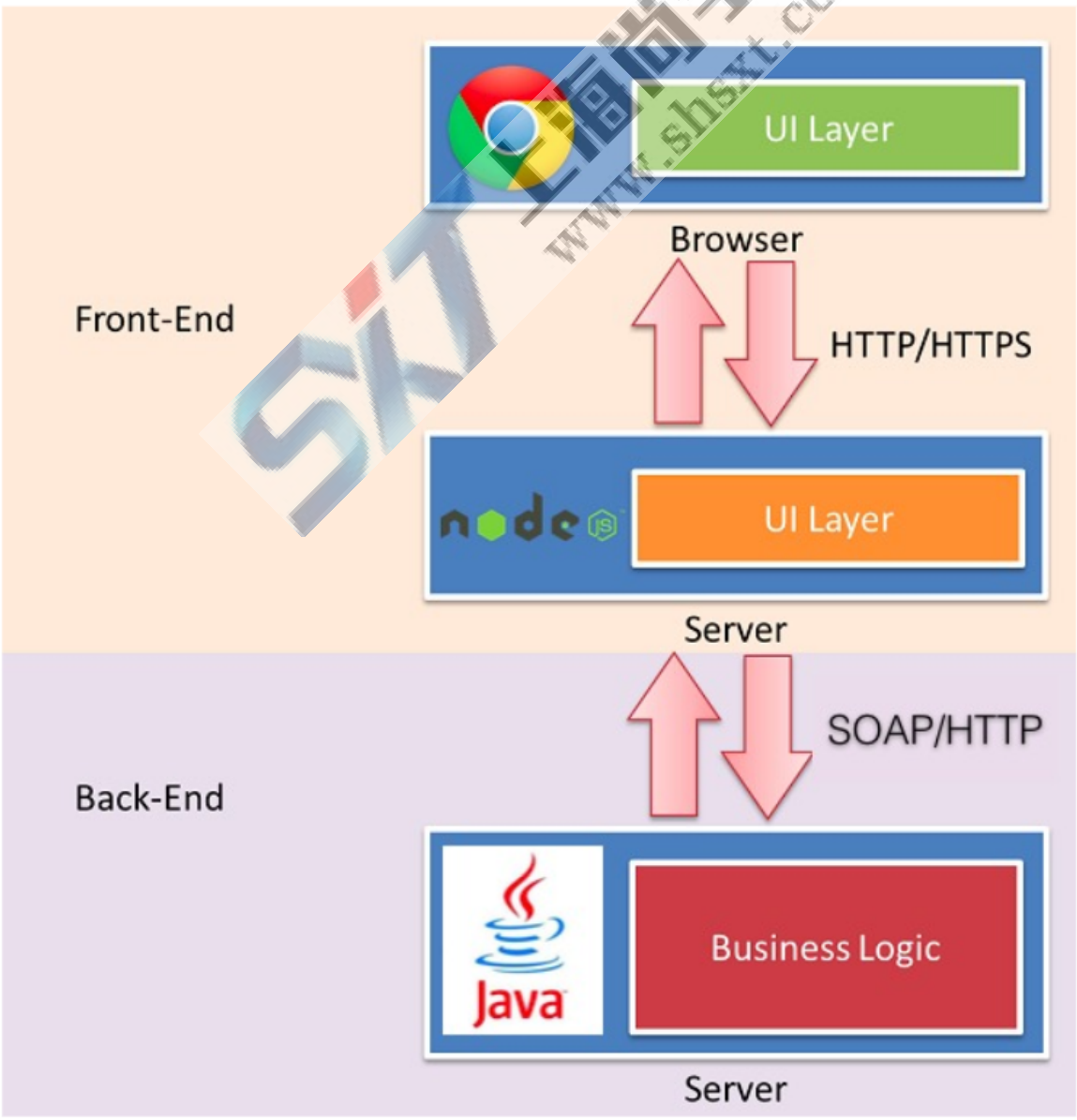
- **前后端职责很清晰：** 前端工作在浏览器端，后端工作在服务端。清晰的分工，可以让开发并行，测试数据的模拟不难，前端可以本地开发。后端则可以专注于业务逻辑的处理，输出 RESTful。
- **前端开发的复杂度可控：** 前端代码很重，但合理的分层，让前端代码能各司其职。这一块蛮有意思的，简单如模板特性的选择，就有很多很多讲究。并非越强大越好，限制什么，留下哪些自由，代码应该如何组织，所有这一切设计，得花一本书的厚度去说明。
- **部署相对独立：** 可以快速改进产品体验

缺点

- 代码不能复用。比如后端依旧需要对数据做各种校验，校验逻辑无法复用浏览器端的代码。如果可以复用，那么后端的数据校验可以相对简单化。
- 全异步，对 SEO 不利。往往还需要服务端做同步渲染的降级方案。
- 性能并非最佳，特别是移动互联网环境下。
- SPA 不能满足所有需求，依旧存在大量多页面应用。URL Design 需要后端配合，前端无法完全掌控。

NodeJS 带来的全栈时代

前端为主的 MV* 模式解决了很多很多问题，但如上所述，依旧存在不少不足之处。随着 NodeJS 的兴起，JavaScript 开始有能力运行在服务端。这意味着可以有一种新的研发模式：



式下，前后端的职责很清晰。对前端来说，两个 UI 层各司其职：

- Front-end UI layer 处理浏览器层的展现逻辑。通过 CSS 渲染样式，通过 JavaScript 添加交互功能，HTML 的生成也可以放在这层，具体看应用场景。
- Back-end UI layer 处理路由、模板、数据获取、Cookie 等。通过路由，前端终于可以自主把控 URL Design，这样无论是单页面应用还是多页面应用，前端都可以自由调控。后端也终于可以摆脱对展现的强关注，转而可以专心于业务逻辑层的开发。

通过 Node，Web Server 层也是 JavaScript 代码，这意味着部分代码可前后复用，需要 SEO 的场景可以在服务端同步渲染，由于异步请求太多导致的性能问题也可以通过服务端来缓解。前一种模式的不足，通过这种模式几乎都能完美解决掉。

与 JSP 模式相比，全栈模式看起来是一种回归，也的确是一种向原始开发模式的回归，不过是一种螺旋上升式的回归。

基于 NodeJS 的全栈模式，依旧面临很多挑战：

- 需要前端对服务端编程有更进一步的认识。比如 TCP/IP 等网络知识的掌握。
- NodeJS 层与 Java 层的高效通信。NodeJS 模式下，都在服务器端，RESTful HTTP 通信未必高效，通过 SOAP 等方式通信更高效。一切需要在验证中前行。
- 对部署、运维层面的熟练了解，需要更多知识点和实操经验。
- 大量历史遗留问题如何过渡。这可能是最大最大的阻力。

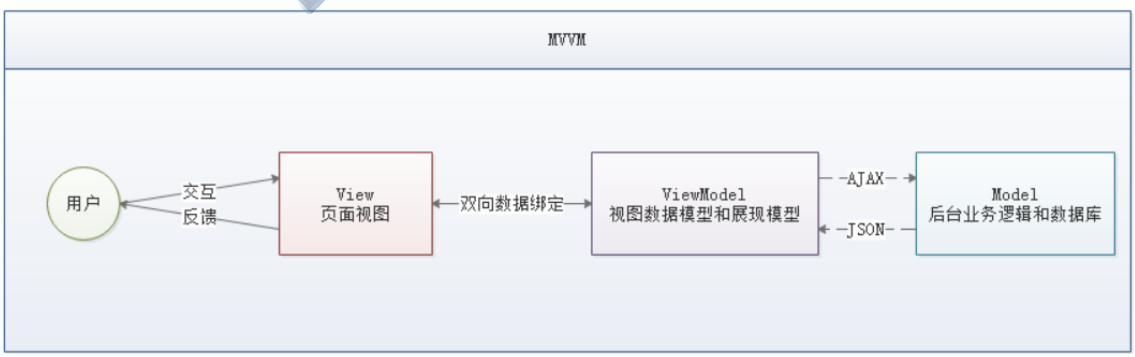
前端 MVVM 模式

什么是 MVVM

MVVM（Model-View-ViewModel）是一种软件架构设计模式，由微软 WPF（用于替代 WinForm，以前就是用这个技术开发桌面应用程序的）和 Silverlight（类似于 Java Applet，简单点说就是在浏览器上运行的 WPF） 的架构师 Ken Cooper 和 Ted Peters 开发，是一种简化用户界面的`事件驱动编程方式`。由 John Gossman（同样也是 WPF 和 Silverlight 的架构师）于 2005 年在他的博客上发表。

MVVM 源自于经典的 MVC（Model-View-Controller）模式（期间还演化出了 MVP（Model-View-Presenter） 模式）。`MVVM 的核心是 ViewModel 层`，负责转换 Model 中的数据对象来让数据变得更容易管理和使用，其作用如下：

- 该层向上与视图层进行双向数据绑定
- 向下与 Model 层通过接口请求进行数据交互



MVVM 已经相当成熟了，主要运用但不仅仅在网络应用程序开发中。当下流行的 MVVM 框架有`Vue.js`，`AngularJS`等。

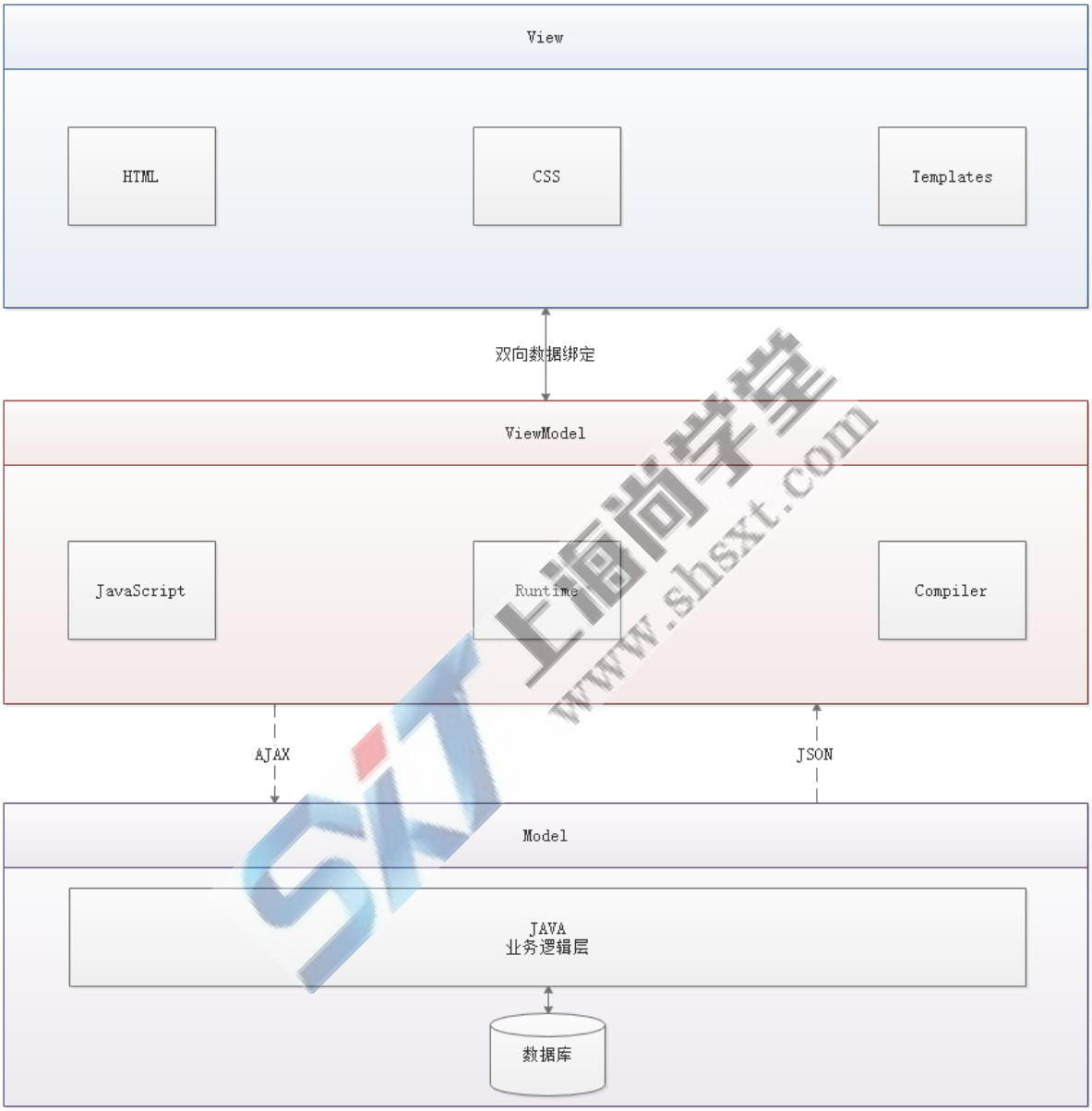
为什么要使用 MVVM

MVVM 模式和 MVC 模式一样，主要目的是分离视图（View）和模型（Model），有几大好处

视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的 View 上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

- **可复用：** 你可以把一些视图逻辑放在一个 ViewModel 里面，让很多 View 重用这段视图逻辑。
- **独立开发：** 开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。
- **可测试：** 界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来写。

MVVM 的组成部分



View

View 是视图层，也就是用户界面。前端主要由 HTML 和 CSS 来构建，为了更方便地展现 ViewModel 或者 Model 层的数据，已经产生了各种各样的前后端模板语言，比如 `FreeMarker`、`Thymeleaf` 等等，各大 MVVM 框架如 `Vue.js`，`AngularJS`，`EJS` 等也都有自己用来构建用户界面的内置模板语言。

Model

Model 是指数据模型，泛指后端进行的各种业务逻辑处理和数据操控，主要围绕数据库系统展开。这里的难点主要在于需要和前端约定统一的 `接口规则`

ViewModel

是由前端开发人员组织生成和维护的视图数据层。在这一层，前端开发者对从后端获取的 Model 数据进行转换处理，做二次封装，以生成符合 View 层使用预期的视图数据模型。

需要注意的是 ViewModel 所封装出来的数据模型包括视图的状态和行为两部分，而 Model 层的数据模型是只包含状态的

- 比如页面的这一块展示什么，那一块展示什么这些都属于视图状态（展示）
- 页面加载进来时发生什么，点击这一块发生什么，这一块滚动时发生什么这些都属于视图行为（交互）

视图状态和行为都封装在了 ViewModel 里。这样的封装使得 ViewModel 可以完整地去描述 View 层。由于实现了双向绑定，ViewModel 的内容会实时展现在 View 层，这是激动人心的，因为前端开发者再也不必低效又麻烦地通过操纵 DOM 去更新视图。

MVVM 框架已经把最脏最累的一块做好了，我们开发者只需要处理和维护 ViewModel，更新数据视图就会自动得到相应更新，真正实现`事件驱动编程`。

View 层展现的不是`Model`层的数据，而是`ViewModel`的数据，由`ViewModel`负责与`Model`层交互，这就完全解耦了`View`层和`Model`层，这个解耦是至关重要的，它是前后端分离方案实施的重要一环`。

概述

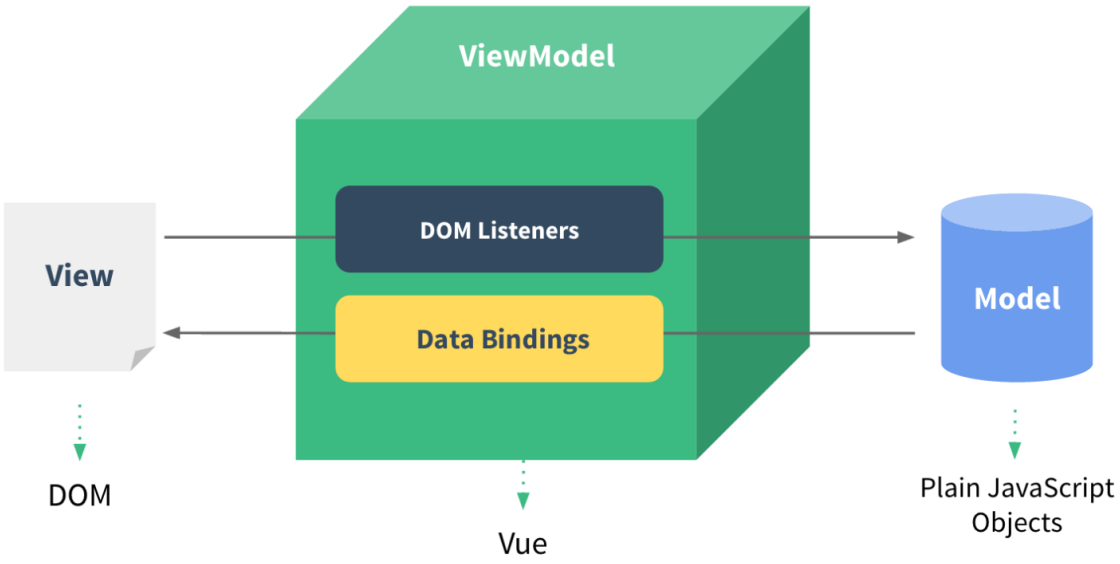
Vue (读音 /vjuː/，类似于 **view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库**只关注视图层**，不仅易于上手，还便于与第三方库（如：`vue-router`，`vue-resource`，`vuex`）或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

MVVM 模式的实现者

我们知道 MVVM 表示如下：

- Model：模型层，在这里表示 JavaScript 对象
- View：视图层，在这里表示 DOM（HTML 操作的元素）

del：连接视图和数据的中间件，**Vue.js 就是 MVVM 中的 ViewModel 层的实现者**



在 MVVM 架构中，是不允许 **数据** 和 **视图** 直接通信的，只能通过 `ViewModel` 来通信，而 ViewModel 就是定义了一个 `Observer` 观察者

- ViewModel 能够观察到数据的变化，并对视图对应的内容进行更新
- ViewModel 能够监听到视图的变化，并能够通知数据发生改变

至此，我们就明白了，Vue.js 就是一个 MVVM 的实现者，他的核心就是实现了 `DOM 监听` 与 `数据绑定`

其它 MVVM 实现者

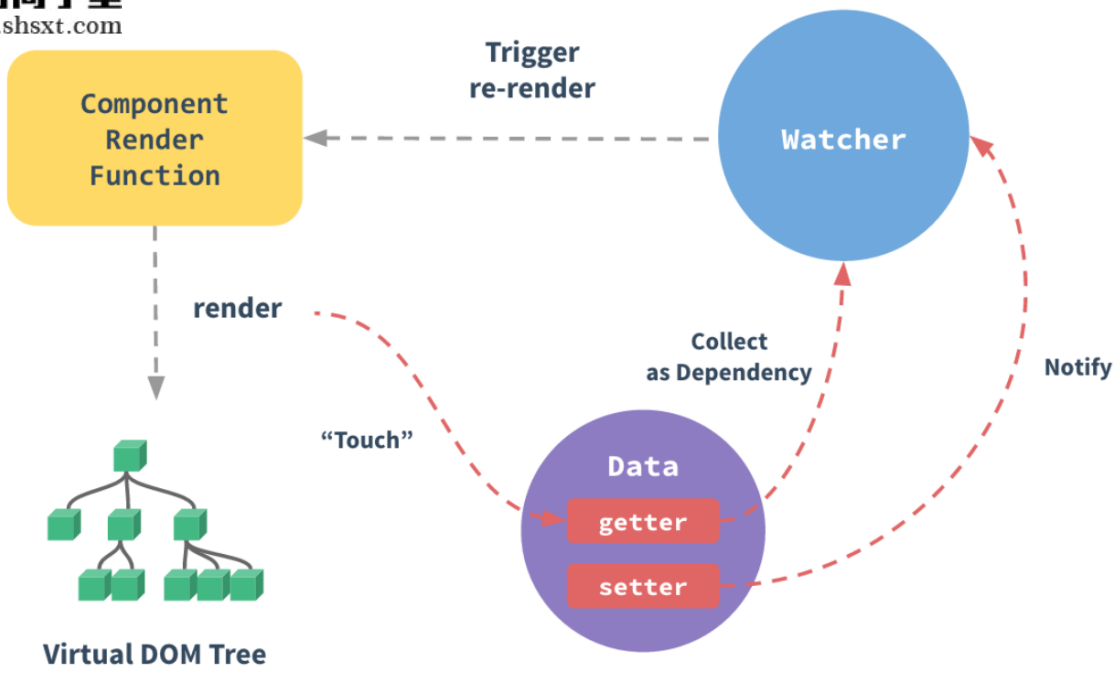
- AngularJS
- ReactJS
- 微信小程序

为什么要使用 Vue.js

- 轻量级，体积小是一个重要指标。Vue.js 压缩后有只有 **30多kb** （Angular 压缩后 **56kb+**，React 压缩后 **44kb+**）
- 移动优先。更适合移动端，比如移动端的 Touch 事件
- 易上手，学习曲线平稳，文档齐全
- 吸取了 Angular（`模块化`）和 React（`虚拟 DOM`）的长处，并拥有自己独特的功能，如：`计算属性`
- 开源，社区活跃度高

Vue.js 的两大核心要素

数据驱动



当你把一个普通的 JavaScript 对象传给 Vue 实例的 `data` 选项，Vue 将遍历此对象所有的属性，并使用 `Object.defineProperty` 把这些属性全部转为 `getter/setter`。**Object.defineProperty 是 ES5 中一个无法 shim 的特性，这也就是为什么 Vue 不支持 IE8 以及更低版本浏览器。**

这些 getter/setter 对用户来说是不可见的，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。这里需要注意的问题是浏览器控制台在打印数据对象时 getter/setter 的格式化并不同，所以你可能需要安装 `vue-devtools` 来获取更加友好的检查接口。

每个组件实例都有相应的 **watcher** 实例对象，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 `setter` 被调用时，会通知 `watcher` 重新计算，从而致使它关联的组件得以更新。

组件化

- 页面上每个独立的可交互的区域视为一个组件
- 每个组件对应一个工程目录，组件所需的各种资源在这个目录下就近维护
- 页面不过是组件的容器，组件可以嵌套自由组合（复用）形成完整的页面

第一个 Vue 应用程序

兼容性

Vue **不支持** IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。但它支持所有**兼容 ECMAScript 5 的浏览器。**

下载地址

- 开发版本
  - 包含完整的警告和调试模式
  - 删除了警告，30.96KB min + gzip
- CDN
  - ```
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    ```
  - ```
    <script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>
    ```

ue 应用程序

Vue.js 的核心是实现了 MVVM 模式，她扮演的角色就是 ViewModel 层，那么所谓的第一个应用程序就是展示她的 **数据绑定** 功能，操作流程如下：

创建一个 HTML 文件

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>第一个Vue应用程序</title>
        <!-- vue核心库 -->
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>

    </body>
</html>
```

引入 Vue.js

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

创建一个 Vue 的实例

```
<script >
    <!-- 编写vue组件 -->
        var app = new Vue({
            // element的简写    挂载元素
            el:'#app',
            // 组件内部数据
            data:{
                message:'Hello Vue!'
            }
        })
</script>
```

说明

- `el:'#app'`：绑定元素的 ID
- `data:{message:'Hello Vue!'}`：数据对象中有一个名为 `message` 的属性，并设置了初始值 `Hello Vue!`

将数据绑定到页面元素将数据绑定到页面元素

```
<div id="app">
    {{message}}
</div>
```

说明：只需要在绑定的元素中使用 `双花括号` 将 Vue 创建的名为 `message` 属性包裹起来，即可实现数据绑定功能，也就实现了 ViewModel 层所需的效果，是不是和 `EL` 表达式非常像?

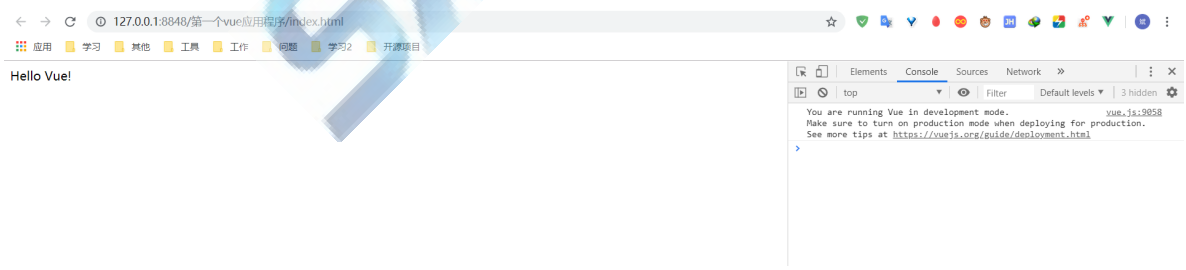
```
#{message} => {{message}}
```

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>第一个Vue应用程序</title>
        <!-- vue核心库 -->
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <!-- 准备DOM结构 -->
        <div id="app">
            {{message}}
        </div>
        <script type="text/javascript">
            <!-- 编写Vue组件 -->
            var app = new Vue({
                // element的简写    挂在元素
                el:'#app',
                // 组件内部数据
                data:{
                    message:'Hello Vue!'
                }
            })
        </script>
    </body>
</html>
```

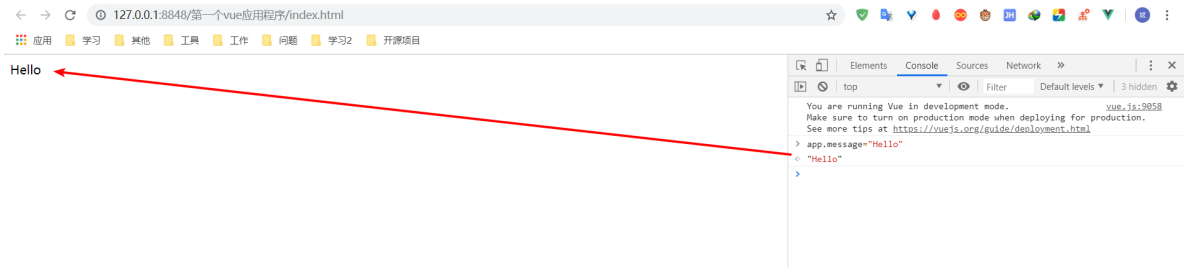
测试 Vue

为了能够更直观的体验 Vue 带来的数据绑定功能，我们需要在浏览器测试一番，操作流程如下：

- 在 `Chrome` 浏览器上运行第一个 Vue 应用程序，并按 `F12` 进入 `开发者工具`



- 在控制台输入 `app.message = 'Hello World'`，然后 `回车`，你会发现浏览器中显示的内容会直接变成 `Hello World`



说明

在之前的代码中，我们创建了一个名为 `app` 的 Vue 实例

```
new Vue({
    // element的简写    挂在元素
    el:'#app',
    // 组件内部数据
    data:{
        message:'Hello Vue!'
    }
})
```

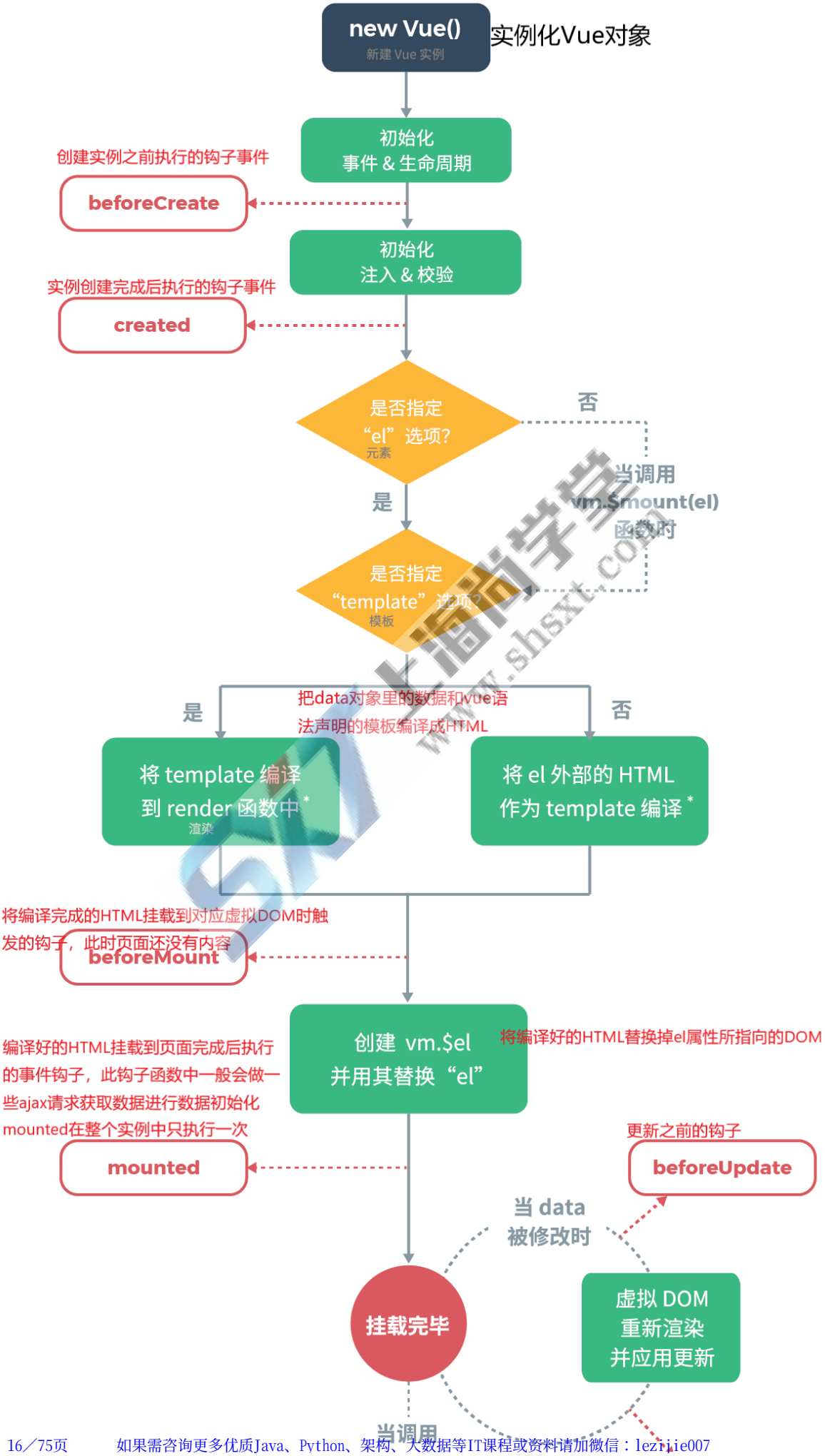
此时就可以在控制台直接输入 `vm.message` 来修改值，中间是可以省略 `data` 的，在这个操作中，我并没有主动操作 DOM，就让页面的内容发生了变化，这就是借助了 Vue 的 **数据绑定** 功能实现的；MVVM 模式中要求 ViewModel 层就是使用 `观察者模式` 来实现数据的监听与绑定，以做到数据与视图的快速响应。

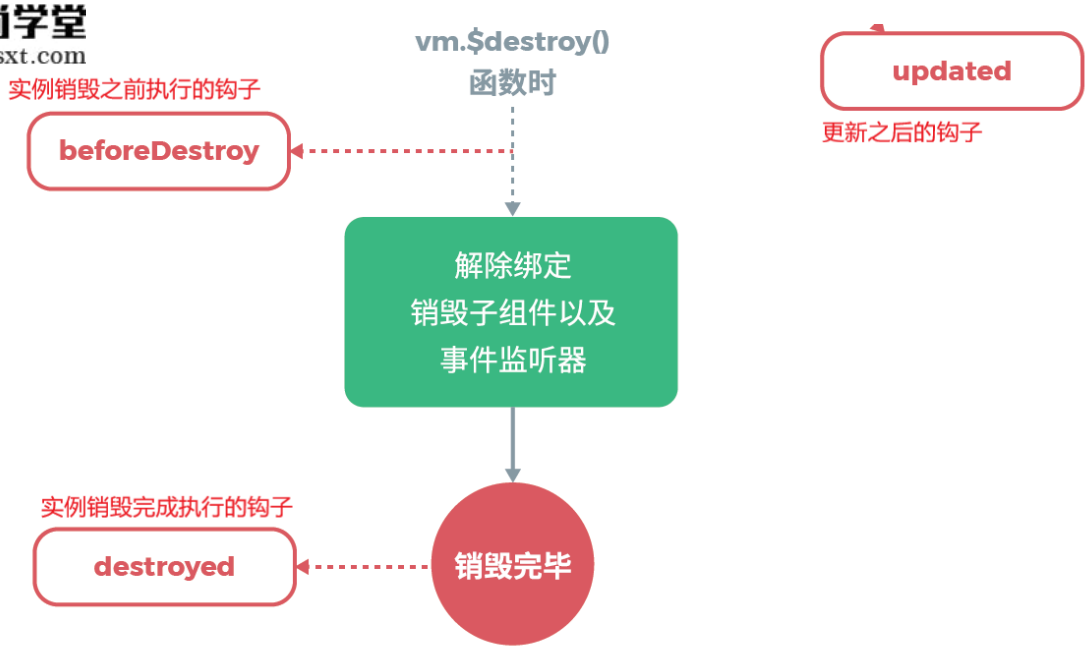
Vue 实例的生命周期

什么是生命周期

Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载 DOM、渲染→更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。通俗说就是 Vue 实例从创建到销毁的过程，就是生命周期。

在 Vue 的整个生命周期中，它提供了一系列的事件，可以让我们在事件触发时注册 JS 方法，可以让我们用自己注册的 JS 方法控制整个大局，在这些事件响应方法中的 this 直接指向的是 Vue 的实例。





* 如果使用构造生成文件（例如构造单文件组件），
模板编译将提前执行

钩子函数的触发时机

beforeCreate

在实例初始化之后，数据观测(data observer) 和 event/watcher 事件配置之前被调用。

created

实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测(data observer)，属性和方法的运算， watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

beforeMount

在挂载开始之前被调用：相关的 render 函数首次被调用。

mounted

el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。

beforeUpdate

数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。 你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。

updated

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。

beforeDestroy

调用。在这一步，实例仍然完全可用。

destroyed

Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

条件渲染

条件判断语句

- `v-if`
- `v-else`

什么是条件判断语句，就不需要我说明了吧，直接看语法上效果

HTML

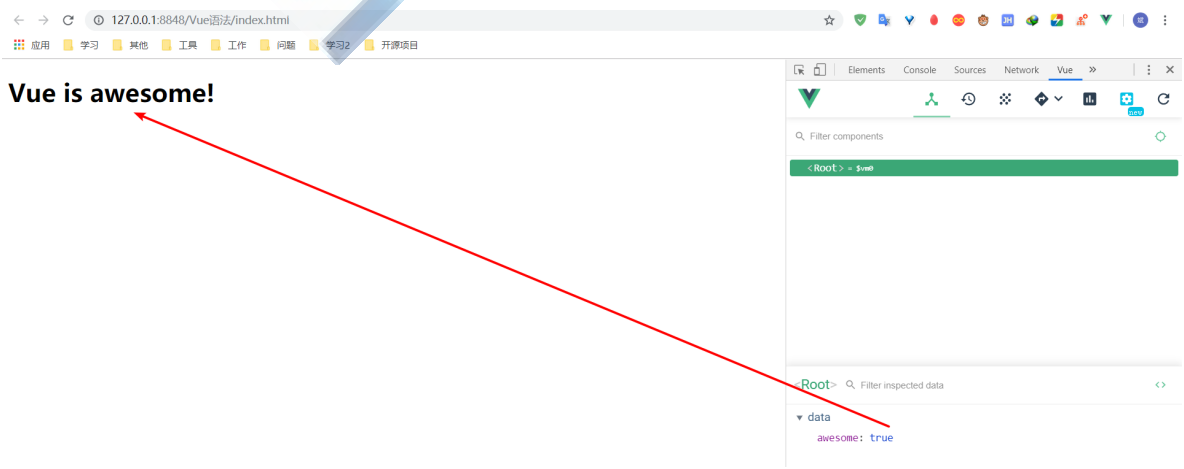
```
<div id="app">
    <h1 v-if="awesome">Vue is awesome!</h1>
    <h1 v-else>Oh no 😢</h1>
</div>
```

JavaScript

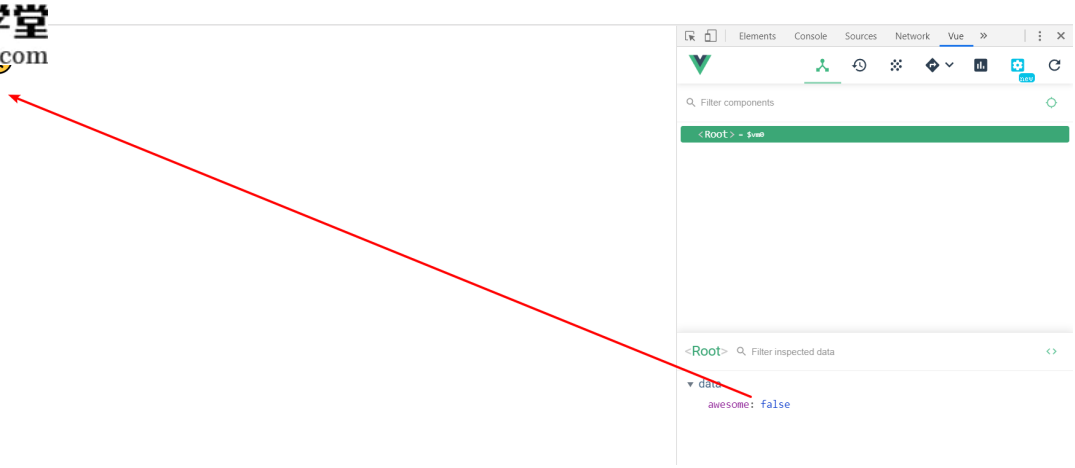
```
var app = new Vue({
    el:'#app',
    data:{
        awesome:true
    }
})
```

测试效果

- 在 `Chrome` 浏览器上运行，并按 `F12` 进入 `开发者工具`



- 在控制台输入 `app.awesome = false` ，然后 `回车` ，你会发现浏览器中显示的内容会直接变成 `Oh no`



注：使用 `v-*` 属性绑定数据是不需要 `双花括号` 包裹的

完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Vue语法</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <div id="app">
            <h1 v-if="awesome">Vue is awesome!</h1>
            <h1 v-else>Oh no 😢</h1>
        </div>
        <script type="text/javascript">
            var app = new Vue({
                el:'#app',
                data:{
                    awesome:true
                }
            })
        </script>
    </body>
</html>
```

连续的条件判断语句

- `v-if`
- `v-else-if`
- `v-else`

HTML

```
<div id="app">
    <div v-if="type === 'A'">
        A
    </div>
    <div v-else-if="type === 'B'">
        B
    </div>
    <div v-else-if="type === 'C'">
```

```
    </div>
    <div v-else>
        Not A/B/C
    </div>
</div>
```

注：`===` 三个等号在 JS 中表示绝对等于（就是数据与类型都要相等）

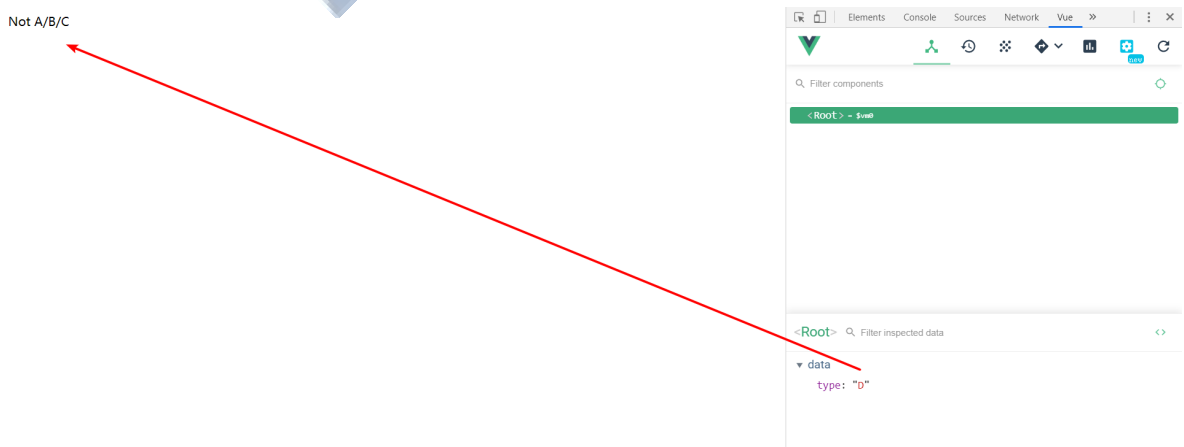
JavaScript

```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            type: 'A'
        }
    })
</script>
```

测试效果

- 在 `Chrome` 浏览器上运行，并按 `F12` 进入 `开发者工具`

- 分别观察在控制台输入 `app.type = 'B'、'C'` 的变化



完整的 HTML

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

```
        <meta charset="utf-8" />
        <title>Vue语法</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <div id="app">
            <div v-if="type === 'A'">
                A
            </div>
            <div v-else-if="type === 'B'">
                B
            </div>
            <div v-else-if="type === 'C'">
                C
            </div>
            <div v-else>
                Not A/B/C
            </div>
        </div>
        <script type="text/javascript">
            var app = new Vue({
                el:'#app',
                data:{
                    type: 'A'
                }
            })
        </script>
    </body>
</html>
```

列表渲染

循环遍历语句

- `v-for`

HTML

```
<ul id="example">
    <li v-for="item in items">
        {{item.message}}
    </li>
</ul>
```

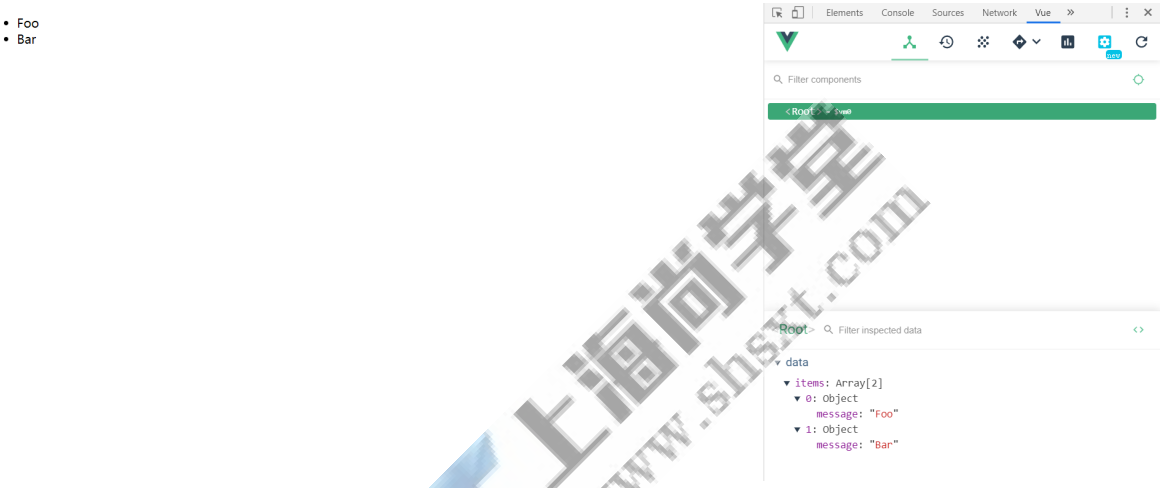
注：`items` 是源数据数组并且 `item` 是数组元素迭代的别名。是不是像极了 `Thymeleaf`

JavaScript

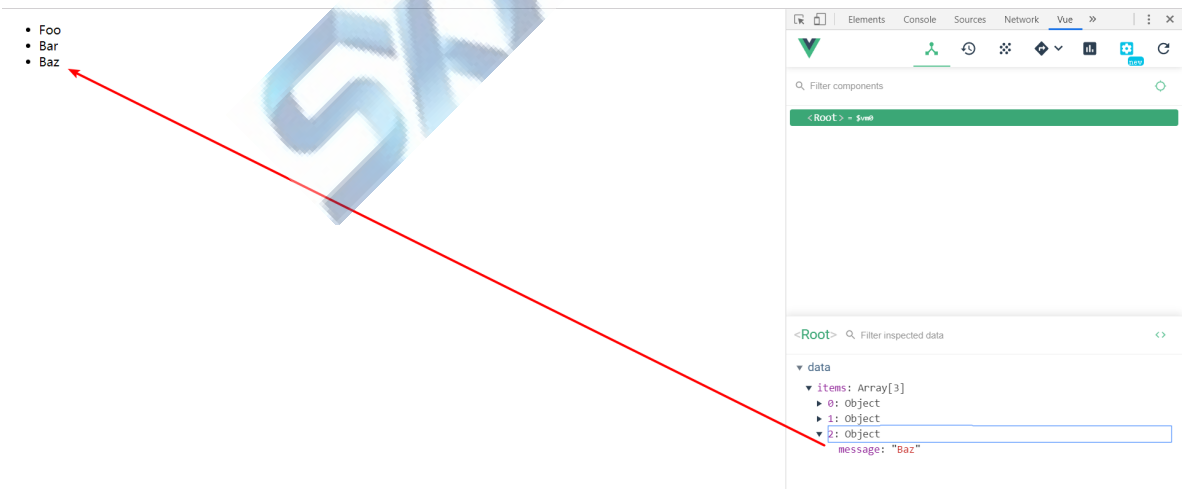

```
<script type="text/javascript">
    var example = new Vue({
        el:'#example',
        data:{
            items:[
                {message:'Foo'},
                {message:'Bar'},
            ]
        }
    })
</script>
```

测试效果

- 在 `Chrome` 浏览器上运行，并按 `F12` 进入 `开发者工具`



- 在控制台输入 `example.items.push({message: 'Baz'})` ，尝试追加一条数据，你会发现浏览器中显示的内容会增加一条 `Baz`



完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Vue语法</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
```

```
        <ul id="example">
            <li v-for="item in items">
                {{item.message}}
            </li>
        </ul>
        <script type="text/javascript">
            var example = new Vue({
                el:'#example',
                data:{
                    items:[
                        {message:'Foo'},
                        {message:'Bar'},
                    ]
                }
            })
        </script>
    </body>
</html>
```

事件处理

监听事件

- v-on

HTML

```
<div id="example">
    <!-- greet 是在下面定义的方法名 -->
    <button v-on:click="greet">greet</button>
</div>
```

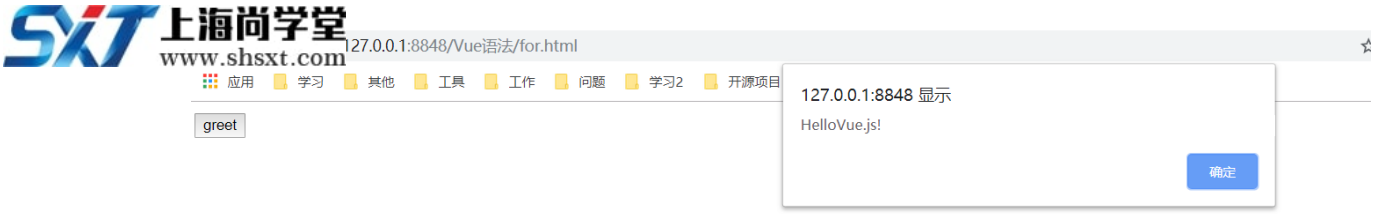
注：在这里我们使用了 v-on 绑定了 click 事件，并指定了名为 greet 的方法

JavaScript

方法必须定义在 Vue 实例的 methods 对象中

```
<script type="text/javascript">
    var example = new Vue({
        el:'#example',
        data:{
            name:'Vue.js'
        },
        // 在methods对象中定义方法
        methods:{
            greet:function(event){
                alert('Hello'+this.name+'!')
            }
        }
    })
</script>
```

测试效果



完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Vue语法</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <div id="example">
            <!-- greet 是在下面定义的方法名 -->
            <button v-on:click="greet">greet</button>
        </div>
        <script type="text/javascript">
            var example = new Vue({
                el:'#example',
                data:{
                    name:'Vue.js'
                },
                // 在methods对象中定义方法
                methods:{
                    greet:function(event){
                        alert('Hello'+this.name+'!')
                    }
                }
            })
        </script>
    </body>
</html>
```

使用Axios实现异步通信

什么是 Axios

Axios 是一个开源的可以用在浏览器端和 NodeJS 的异步通信框架，她的主要作用就是实现 AJAX 异步通信，其功能特点如下：

- 从浏览器中创建 `XMLHttpRequests`
- 从 `node.js` 创建 `http` 请求
- 支持 `Promise API`
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 `JSON` 数据

支持防御 XSRF（跨站请求伪造）

GitHub：https://github.com/axios/axios

为什么要使用 Axios

由于 Vue.js 是一个 **视图层框架** 并且作者（尤雨溪）严格遵守 SoC （关注度分离原则），所以 Vue.js 并不包含 AJAX 的通信功能，为了解决通信问题，作者单独开发了一个名为 vue-resource 的插件，不过在进入 2.0 版本以后停止了对该插件的维护并推荐了 Axios 框架

第一个 Axios 应用程序

咱们开发的接口大部分都是采用 JSON 格式，可以先在项目里模拟一段 JSON 数据，数据内容如下：

```
{
  "name": "百度",
  "url": "http://www.baidu.com",
  "page": 66,
  "isNonProfit": true,
  "address": {
    "street": "海定区",
    "city": "北京市",
    "country": "中国"
  },
  "links": [
    {
      "name": "Google",
      "url": "http://www.google.com"
    },
    {
      "name": "Baidu",
      "url": "http://www.baidu.com"
    },
    {
      "name": "Sougou",
      "url": "http://www.sougou.com"
    }
  ]
}
```

创建一个名为 data.json 的文件并填入上面的内容，放在项目的根目录下，如图所示：

创建 HTML

```
<div id="example">
    <div>
        名称:{{info.name}}
    </div>
    <div >
        地址:{{info.address.country}}-{{info.address.city}}-
{{info.address.street}}
    </div>
    <div>
        链接:<a v-bind:href="info.url" target="_blank">{{info.url}}</a>
    </div>
    <ul>
        <li v-for="link in info.links">{{link.name}}-{{link.url}}</li>
    </ul>
</div>
```

注：在这里使用了 `v-bind` 将 `a:href` 的属性值与 Vue 实例中的数据进行绑定

引入 JS 文件

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

JavaScript

```
<script type="text/javascript">
    var example = new Vue({
        el:'#example',
        data(){
            return{
                info:{
                    name:'',
                    address:{
```

```
                    country:'',
                    city:'',
                    street:''
                },
                links:[]
            }
        }
    },
    mounted() {
        axios
            .get('data.json')
            .then(response=>this.info=response.data)
    }
})
</script>
```

使用 `axios` 框架的 `get` 方法请求 AJAX 并自动将数据封装进了 Vue 实例的数据对象中

数据对象

这里的数据结构与 JSON 数据结构是匹配的

```
info:{
    name:'',
    address:{
        country:'',
        city:'',
        street:''
    },
    links:[]
}
```

调用 `get` 请求

调用 `axios` 的 `get` 请求并自动装箱数据

```
axios
    .get('data.json')
    .then(response=>this.info=response.data);
```

测试效果

名称:百度
地址:中国-北京市-海定区
链接:http://www.baicu.com

- Google-http://www.google.com
- Baidu-http://www.baidu.com
- Sougou-http://www.sougou.com

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Axios</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
        <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    </head>
    <body>
        <div id="example">
            <div>
                名称:{{info.name}}
            </div>
            <div >
                地址:{{info.address.country}}-{{info.address.city}}-
{{info.address.street}}
            </div>
            <div>
                链接:<a v-bind:href="info.url" target="_blank">{{info.url}}</a>
            </div>
            <ul>
                <li v-for="link in info.links">{{link.name}}-{{link.url}}</li>
            </ul>
        </div>
        <script type="text/javascript">
            var example = new Vue({
                el:'#example',
                data(){
                    return{
                        info:{
                            name:'',
                            address:{
                                country:'',
                                city:'',
                                street:''
                            },
                            links:[]
                        }
                    }
                },
                mounted() {
                    axios
                    .get('data.json')
                    .then(response=>this.info=response.data);
                }
            })
        </script>
    </body>
</html>
```

表单输入绑定

什么是双向数据绑定

MVVM 框架，即数据双向绑定，即当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化。这也算是 Vue.js 的精髓之处了。值得注意的是，**我们所说的数据双向绑定，一定是对于 UI 控件来说的**，非 UI 控件不会涉及到数据双向绑定。单向数据绑定是使用状态管理工具的前提。如果我们使用 `vuex`，那么数据流也是单项的，这时就会和双向数据绑定有冲突。

为什么要实现数据的双向绑定

在 Vue.js 中，如果使用 `vuex`，实际上数据还是单向的，之所以说是数据双向绑定，这是用的 UI 控件来说，对于我们处理表单，Vue.js 的双向数据绑定用起来就特别舒服了。即两者并不互斥，在全局性数据流使用单项，方便跟踪；局部性数据流使用双向，简单易操作。

在表单中使用双向数据绑定

你可以用 `v-model` 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 `v-model` 本质上不过是语法糖。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。

注意：`v-model` 会忽略所有表单元素的 `value`、`checked`、`selected` 特性的初始值而总是将 Vue 实例的数据作为数据来源。你应该通过 JavaScript 在组件的 `data` 选项中声明初始值。

单行文本

```
<div id="app">
    <p>单行文本</p>
    <input type="text" v-model="message" />
    <p>Message is :{{message}}</p>
</div>
```

```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            message:'Hello Vue'
        }
    })
</script>
```

多行文本

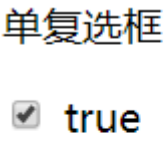
```
app">
    <p>多行文本</p>
    <span>Multiline message is:</span>
    <p>{{message}}</p>
    <br>
    <textarea v-model="message"></textarea>
</div>
```

```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            message:'Hello textarea'
        }
    })
</script>
```

单复选框

```
<div id="app">
    <p>单复选框</p>
    <input type="checkbox" id="checkbox" v-model="checked">
    <label for="checkbox">{{checked}}</label>
</div>
```

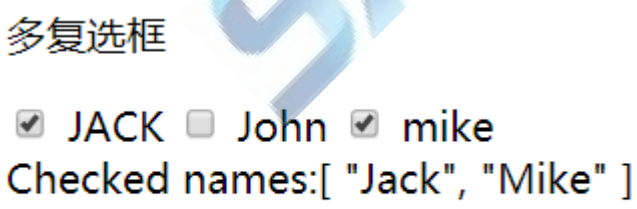
```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            checked:true
        }
    })
</script>
```



多复选框

```
<div id="app">
    <p>多复选框</p>
    <input type="checkbox" id="Jack" value="Jack" v-model="checkNames"/>
    <label for="jack">JACK</label>
    <input type="checkbox" id="John" value="John" v-model="checkNames"/>
    <label for="John">John</label>
    <input type="checkbox" id="Mike" value="Mike" v-model="checkNames"/>
    <label for="Mike">mike</label>
    <br>
    <span>Checked names:{{checkNames}}</span>
</div>
```

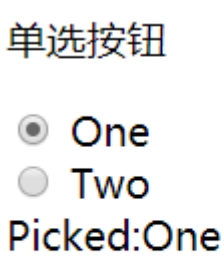
```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            checkNames:[]
        }
    })
</script>
```



单选按钮


```
app">
    <p>单选按钮</p>
    <input type="radio" id="one" value="One" v-model="picked"/>
    <label for="one">One</label>
    <br>
    <input type="radio" id="two" value="Two" v-model="picked"/>
    <label for="two">Two</label>
    <br>
    <span>Picked:{{picked}}</span>
</div>
```

```
<script type="text/javascript">
    var app = new Vue({
        el:'#app',
        data:{
            picked:''
        }
    })
</script>
```



选择框

```
<div id="app">
    <p>选择框</p>
    <select v-model="selected">
        <option disabled value =""></option>
        <option>A</option>
        <option>B</option>
        <option>C</option>
    </select>
    <span>Selected :{{selected}}</span>
</div>
```

```
<script type="text/javascript">
    var app = new Vue({
        el: '#app',
        data: {
            selected: ''
        }
    });
</script>
```

选择框

C ▾ Selected :C

注意：如果 `v-model` 表达式的初始值未能匹配任何选项，`<select>` 元素将被渲染为“未选中”状态。在 iOS 中，这会使用户无法选择第一个选项。因为这样的情况下，iOS 不会触发 change 事件。因此，更推荐像上面这样提供一个值为空的禁用选项。

组件基础

什么是组件

组件是可复用的 Vue 实例，说白了就是一组可以重复使用的模板，跟 `JSTL` 的自定义标签、`Thymeleaf` 的 `th:fragment` 以及 `Sitemesh3` 框架有着异曲同工之妙。通常一个应用会以一棵嵌套的组件树的形式来组织：

例如，你可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。

第一个 Vue 组件

注意：在实际开发中，我们并不会用以下方式开发组件，而是采用 `vue-cli` 创建 `.vue` 模板文件的方式开发，以下方法只是为了让大家理解什么是组件。

使用 `Vue.component()` 方法注册组件

JavaScript

```
<script type="text/javascript">
    // 定义一个名为component-li的组件
    Vue.component('component-li',{
    template:'<li>Hello li</li>'
})
var app = new Vue({
    el:'#app',
})
</script>
```

HTML

app">

        <component-li></component-li>


```

**说明**

Vue.component()：注册组件
component-li：自定义组件的名字
template：组件的模板

**测试效果**

---

- Hello li

## 使用 props 属性传递参数

像上面那样用组件没有任何意义，所以我们是需要传递参数到组件的，此时就需要使用 props 属性了

**注意：默认规则下 props 属性里的值不能为大写**

**JavaScript**

```
<script type="text/javascript">
    // 定义一个名为component-li的组件
    Vue.component('component-li',{
    props:['item'],
    template:'
Hello {{item}}
'
})
var app = new Vue({
    el:'#app',
    data:{
        items:['古力娜扎','迪丽热巴','玛尔扎哈']
    }
})
</script>
```

**HTML**

```


        <component-li v-for="item in items" v-bind:item="item"></component-li>


```

**说明**

v-for="item in items"：遍历 Vue 实例中定义的名为 items 的数组，并创建同等数量的组件
```

item="item"：将遍历的 item 项绑定到组件中 props 定义的名为 item 属性上； = 号左边的 item 为 props 定义的属性名，右边的为 item in items 中遍历的 item 项的值

测试效果

- Hello 古力娜扎
- Hello 迪丽热巴
- Hello 玛尔扎哈

完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Vue组件</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <div id="app">
            <ul>
                <component-li v-for="item in items" v-bind:item="item">
</component-li>
            </ul>
        </div>
        <script type="text/javascript">
            // 定义一个名为component-li的组件
            Vue.component('component-li',{
                props:['item'],
                template:'<li>Hello {{item}}</li>'
            })
            var app = new Vue({
                el:'#app',
                data:{
                    items:['古力娜扎','迪丽热巴','玛尔扎哈']
                }
            })
        </script>
    </body>
</html>
```

计算属性

什么是计算属性

来声明式的描述一个值依赖了其它的值。当你在模板里把数据绑定到一个计算属性上时，Vue 会在其依赖的任何值导致该计算属性改变时更新 DOM。这个功能非常强大，它可以让你的代码更加声明式、数据驱动并且易于维护。简单点说，它就是一个能够将计算结果缓存起来的属性（**将行为转化成了静态的属性**），仅此而已；

计算属性与方法的区别

完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title></title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
    </head>
    <body>
        <div id="app">
            <p>当前时间方法:{{getCurrentTime()}}</p>
            <p>当前时间属性:{{getCurrentTime1}}</p>
        </div>
        <script type="text/javascript">
            var app = new Vue({
                el:'#app',
                methods:{
                    getCurrentTime:function(){
                        return Date.now();
                    }
                },
                computed:{
                    getCurrentTime1:function(){
                        return Date.now();
                    }
                }
            });
        </script>
    </body>
</html>
```

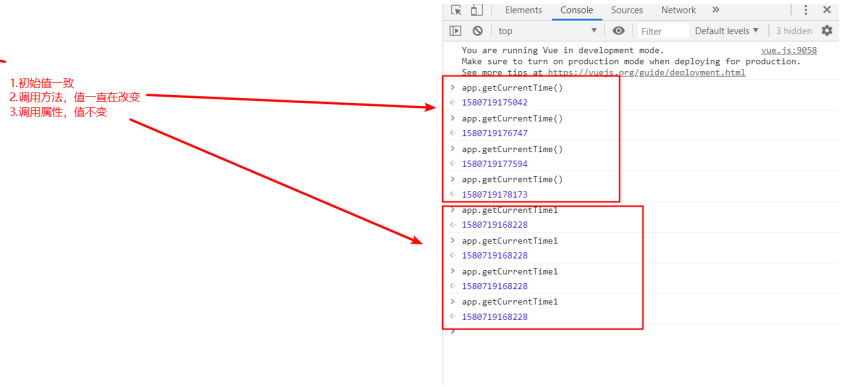
说明

methods：定义方法，调用方法使用 getCurrentTime()，需要带括号
computed：定义计算属性，调用属性使用 getCurrentTime01，不需要带括号;

注意：methods 和 computed 里不能重名

测试效果

仔细看图中说明，观察其中的差异



结论

调用方法时，每次都需要进行计算，既然有计算过程则必定产生系统开销，那如果这个结果是不经常变化的呢？此时就可以考虑将这个结果缓存起来，采用计算属性可以很方便的做到这一点；**计算属性的主要特性就是为了将不经常变化的计算结果进行缓存，以节约我们的系统开销**

插槽与自定义事件

插槽内容

Vue 实现了一套内容分发的 API，这套 API 的设计灵感源自 Web Components 规范草案，将 `<slot>` 元素作为承载分发内容的出口。

利用插槽功能实现一个组合组件

比如准备制作一个待办事项组件（`todo`），该组件由待办标题（`todo-title`）和待办内容（`todo-items`）组成，但这三个组件又是相互独立的，该如何操作呢？

定义一个名为 todo 的待办事项组件

```
Vue.component('todo',{
    template:'<div><slot name=\'todo-title\'></slot><ul><slot name=\'todo-items\'></slot></ul></div>'
});
```

该组件中放置了两个插槽，分别为 todo-title 和 todo-items

定义一个名为 todo-title 的待办标题组件

```
Vue.component('todo-title',{
    props:['title'],
    template:'<div>{{title}}</div>'
});
```

定义一个名为 todo-items 的待办内容组件

```
Vue.component('todo-items',{
    props:['item'],
    template:'<li>{{item}}</li>'
})
```

实例化 Vue 并初始化数据


```
new Vue({
    el:'#app',
    data:{
        title:'标题',
        items:["古力娜扎","迪丽热巴","玛尔扎哈"]
    }
});
```

HTML

```
<div id="app">
    <todo>
        <todo-title slot='todo-title' v-bind:title="title"></todo-title>
        <todo-items slot='todo-items' v-for="item in items" v-bind:item="item">
</todo-items>
    </todo>
</div>
```

此时，我们的 todo-title 和 todo-items 组件分别被分发到了 todo 组件的 todo-title 和 todo-items 插槽中

测试效果

标题

- 古力娜扎
- 迪丽热巴
- 玛尔扎哈

使用自定义事件删除待办事项

通过以上代码不难发现，数据项在 Vue 的实例中，但删除操作要在组件中完成，那么组件如何才能删除 Vue 实例中的数据呢？此时就涉及到参数传递与事件分发了，Vue 为我们提供了自定义事件的功能很好的帮助我们解决了这个问题；使用 this.$emit('自定义事件名', 参数)，操作过程如下

修改创建 Vue 实例代码

```
var app = new Vue({
    el:'#app',
    data:{
        title:'标题',
        items:["古力娜扎","迪丽热巴","玛尔扎哈"]
    },
    // 该方法可以被模板中自定义事件触发
    methods:{
        removeItem:function(index){
            // splice可以对数据添加/删除数据,然后返回被删除数据,index为下标,1为数量
            this.items.splice(index,1);
        }
```

```
});
```

增加了 methods 对象并定义了一个名为 removeItems 的方法

修改 todo-items 待办内容组件的代码

```
Vue.component('todo-items',{
    props:['item','index'],
    template:'<li>{{index}}-{{item}}<button v-on:click="remove">删除</button>
</li>',
    methods:{
        // remove是自定义事件名称,需要在HTML中使用v-on:remove的方式指派
        remove:function(){
            this.$emit("remove");
        }
    }
});
```

增加了 删除 元素并绑定了组件中定义的 remove 事件

修改 todo-items 待办内容组件的 HTML 代码

```
<todo-items slot='todo-items' v-for="(item,index) in items" v-bind:item="item"
v-bind:index="index" v-on:remove="removeItem"></todo-items>
```

增加了 v-on:remove="removeItems" 自定义事件，该事件会调用 Vue 实例中定义的名为 removeItems 的方法

说明

v-on:remove="removeItems 自定义事件不需要添加 index 参数，会通过 v-for="(item,index) in items" 自动带入，所以Vue实例代码里的 removeItem 可以直接传入 index 参数

测试效果

标题

- 0-迪丽热巴 删除
- 1-玛尔扎哈 删除

完整的 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Vue插槽与自定义事件</title>
        <script src="js/vue.js" type="text/javascript" charset="utf-8"></script>
```

```
d>
    <body>
        <div id="app">
            <todo>
                <todo-title slot='todo-title' v-bind:title="title"></todo-title>
                <todo-items slot='todo-items' v-for="(item,index) in items" v-
bind:item="item" v-bind:index="index" v-on:remove="removeItem()"></todo-items>
            </todo>
        </div>

        <script type="text/javascript">
            Vue.component('todo',{
                template:'<div><slot name=\'todo-title\'></slot><ul><slot
name=\'todo-items\'></slot></ul></div>'
            });
            Vue.component('todo-title',{
                props:['title'],
                template:'<div>{{title}}</div>'
            });
            Vue.component('todo-items',{
                props:['item','index'],
                template:'<li>{{index}}-{{item}}<button v-on:click="remove">删除
</button></li>',
                methods:{
                    // remove是自定义事件名称,需要在HTML中使用v-on:remove的方式指派
                    remove:function(){
                        this.$emit("remove");
                    }
                }
            });
            var app = new Vue({
                el:'#app',
                data:{
                    title:'标题',
                    items:["古力娜扎","迪丽热巴","玛尔扎哈"]
                },
                // 该方法可以被模板中自定义事件触发
                methods:{
                    removeItem:function(index){
                        // splice可以对数据添加/删除数据,然后返回被删除数据,index为下
标,1为数量
                        this.items.splice(index,1);
                    }
                }
            });
        </script>
    </body>
</html>
```

vue-cli

什么是 vue-cli

vue-cli 官方提供的一个脚手架（预先定义好的目录结构及基础代码，咱们在创建 Maven 项目时可以选择创建一个骨架项目，这个骨架项目就是脚手架），用于快速生成一个 vue 的项目模板

统一的目录结构
本地调试
热部署
单元测试
集成打包上线

环境准备

Node.js（>= 6.x，首选 8.x）
git

安装 vue-cli

安装 Node.js

官网下载地址 http://nodejs.cn/download

安装 Node.js 淘宝镜像加速器（cnpm）

```
npm install cnpm -g

# 或使用如下语句解决 npm 速度慢的问题
npm install --registry=https://registry.npm.taobao.org
```



```
s [版本 10.0.18362.592]
ft Corporation。保留所有权利。

C:\WINDOWS\system32>npm install cnpm -g
C:\Users\Administrator\AppData\Roaming\npm\cnpm -> C:\Users\Administrator\AppData\Roaming\npm\node_modules\cnpm\bin\cnpm
+ cnpm@6.1.1
added 686 packages from 944 contributors in 86.005s

C:\WINDOWS\system32>
```

安装 vue-cli

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

测试是否安装成功

```
# 查看可以基于哪些模板创建 vue 应用程序，通常我们选择 webpack
vue list
```

```
C:\WINDOWS\system32>vue list

 Available official templates:

 ★  browserify - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.
 ★  browserify-simple - A simple Browserify + vueify setup for quick prototyping.
 ★  pwa - PWA template for vue-cli based on the webpack template
 ★  simple - The simplest possible Vue setup in a single HTML file
 ★  webpack - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.
 ★  webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.
```

第一个 vue-cli 应用程序

创建一个基于 webpack 模板的 vue 应用程序

```
# 这里的 firstvue 是项目名称，可以根据自己的需求起名
vue init webpack firstvue
```

```
oft Corporation。保留所有权利。
D:\workspace\HBuilderProjects>vue init webpack firstvue

? Project name firstvue
? Project description A Vue.js project
? Author zsyue <244173220@qq.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) no

   vue-cli · Generated "firstvue".

# Project initialization finished!
# ========================

To get started:

  cd firstvue
  npm install (or if using yarn: yarn)
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack



D:\workspace\HBuilderProjects>
```

说明

Project name：项目名称，默认 回车 即可

Project description：项目描述，默认 回车 即可

Author：项目作者，默认 回车 即可

Install vue-router：是否安装 vue-router，选择 n 不安装（后期需要再手动添加）

Use ESLint to lint your code：是否使用 ESLint 做代码检查，选择 n 不安装（后期需要再手动添加）

Set up unit tests：单元测试相关，选择 n 不安装（后期需要再手动添加）

Setup e2e tests with Nightwatch：单元测试相关，选择 n 不安装（后期需要再手动添加）

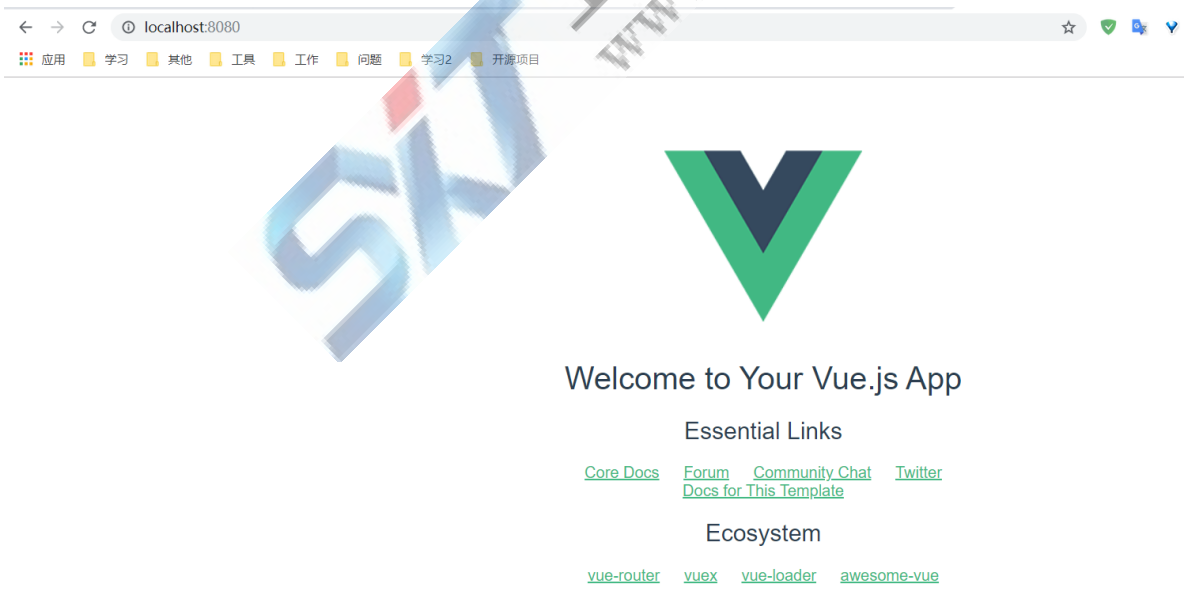
Should we run npm install for you after the project has been created：创建完成后直接初始化，选择 n，我们手动执行

初始化并运行

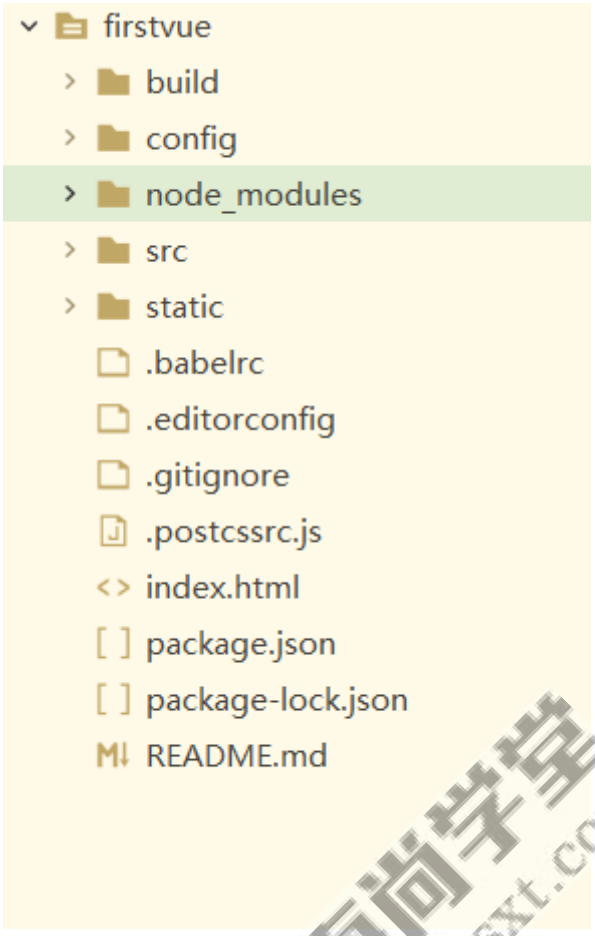
```
cd firstvue
npm install
npm run dev
```

```
ilderProjects\firstvue>npm install
ted extract-text-webpack-plugin@3.0.2: Deprecated. Please use https://github.com/webpack-contrib/mini-cs
s-extract-plugin
npm WARN deprecated browserslist@2.11.3: Browserslist 2 could fail on reading Browserslist >3.0 config used in other too
ls.
npm WARN deprecated bfj-node4@5.3.1: Switch to the `bfj` package for fixes and new features!
npm WARN deprecated core-js@2.6.11: core-js@<3 is no longer maintained and not recommended for usage due to the number o
f issues. Please, upgrade your dependencies to the actual version of core-js@3.
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tool
s.

> core-js@2.6.11 postinstall D:\workspace\HBuilderProjects\firstvue\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> ejs@2.7.4 postinstall D:\workspace\HBuilderProjects\firstvue\node_modules\ejs
> node ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)


> uglifyjs-webpack-plugin@0.4.6 postinstall D:\workspace\HBuilderProjects\firstvue\node_modules\webpack\node_modules\ugl
ifyjs-webpack-plugin
> node lib/post_install.js

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN ajv-keywords@3.4.1 requires a peer of ajv@^6.9.1 but none is installed. You must install peer dependencies your
self.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"an
y"} (current: {"os":"win32","arch":"x64"})

added 1217 packages from 669 contributors and audited 11953 packages in 420.904s

24 packages are looking for funding
  run `npm fund` for details

found 13 vulnerabilities (1 low, 8 moderate, 4 high)
  run `npm audit fix` to fix them, or `npm audit` for details

D:\workspace\HBuilderProjects\firstvue>dir
```

安装并运行成功后在浏览器输入：http://localhost:8080



vue-cli 目录结构



build 和 config：WebPack 配置文件

node_modules：用于存放 `npm install` 安装的依赖文件

src： 项目源码目录

static：静态资源文件

.babelrc：Babel 配置文件，主要作用是将 ES6 转换为 ES5

.editorconfig：编辑器配置

eslintignore：需要忽略的语法检查配置文件

.gitignore：git 忽略的配置文件

.postcssrc.js：css 相关配置文件，其中内部的 `module.exports` 是 NodeJS 模块化语法

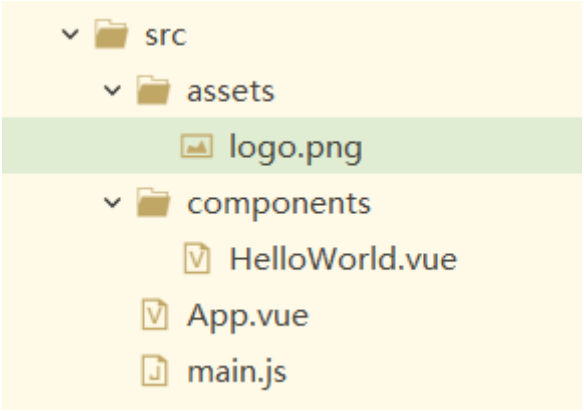
index.html：首页，仅作为模板页，实际开发时不使用

package.json：项目的配置文件

name：项目名称
version：项目版本
description：项目描述
author：项目作者
scripts：封装常用命令
dependencies：生产环境依赖
devDependencies：开发环境依赖

vue-cli src 目录

`src` 目录是项目的源码目录，所有代码都会写在这里



main.js

项目的入口文件，我们知道所有的程序都会有一个入口

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  components: { App },
  template: '<App/>'
})
```

`import Vue from 'vue'`：ES6 写法，会被转换成 `require("vue");` (require 是 NodeJS 提供的模块加载器)

`import App from './App'`：意思同上，但是指定了查找路径，`./` 为当前目录

`Vue.config.productionTip = false`：关闭浏览器控制台关于环境的相关提示

`new Vue({...})`：实例化 Vue

- `el: '#app'`：查找 index.html 中 id 为 app 的元素
- `template: '<App/>'`：模板，会将 index.html 中 `<div id="app"><div>` 替换为 `<App/>`
- `components: { App }`：引入组件，使用的是 `import App from './App'` 定义的 App 组件

App.vue

组件模板

```
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <HelloWorld/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld'
```

```
export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

template：HTML 代码模板，会替换 <App /> 中的内容

import HelloWorld from './components/HelloWorld'：引入 HelloWorld 组件，用于替换 template 中的 <HelloWorld/>

export default{...}：导出 NodeJS 对象，作用是可以通过 import 关键字导入

name: 'App'：定义组件的名称

components: { HelloWorld }：定义子组件

HelloWorld.vue

基本同上，不解释..

关于 <style scoped> 的说明：CSS 样式仅在当前组件有效，声明了样式的作用域

Webpack

Webpack简介

简介

本质上，webpack 是一个现代 JavaScript 应用程序静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

现状

伴随着移动互联网的大潮，当今越来越多的网站已经从网页模式进化到了 WebApp 模式。它们运行在现代浏览器里，使用 HTML5、CSS3、ES6 等新的技术来开发丰富的功能，网页已经不仅仅是完成浏览器的基本需求；WebApp 通常是一个 SPA （单页面应用），每一个视图通过异步的方式加载，这导致页面初始化和使用过程中会加载越来越多的 JS 代码，这给前端的开发流程和资源组织带来了巨大挑战。

前端开发和其他开发工作的主要区别，首先是前端基于多语言、多层次的编码和组织工作，其次前端产品的交付是基于浏览器的，这些资源是通过增量加载的方式运行到浏览器端，如何在开发环境组织好这些碎片化的代码和资源，并且保证他们在浏览器端快速、优雅的加载和更新，就需要一个模块化系统，这个理想中的模块化系统是前端工程师多年来一直探索的难题。

Script 标签

```
<script src="module1.js"></scirpt>
<script src="module2.js"></scirpt>
<script src="module3.js"></scirpt>
<script src="module4.js"></scirpt>
```

这是最原始的 JavaScript 文件加载方式，如果把每一个文件看做是一个模块，那么他们的接口通常是暴露在全局作用域下，也就是定义在 window 对象中，不同模块的调用都是一个作用域。

这种原始的加载方式暴露了一些显而易见的弊端：

全局作用域下容易造成变量冲突
文件只能按照 <script> 的书写顺序进行加载
开发人员必须主观解决模块和代码库的依赖关系
在大型项目中各种资源难以管理，长期积累的问题导致代码库混乱不堪

CommonsJS

服务器端的 NodeJS 遵循 CommonsJS 规范，该规范核心思想是允许模块通过 require 方法来同步加载所需依赖的其它模块，然后通过 exports 或 module.exports 来导出需要暴露的接口。

```
require("module");
require("../module.js");
export.doStuff = function() {};
module.exports = someValue;
```

优点

服务器端模块便于重用
NPM 中已经有超过 45 万个可以使用的模块包
简单易用

缺点

同步的模块加载方式不适合在浏览器环境中，同步意味着阻塞加载，浏览器资源是异步加载的
不能非阻塞的并行加载多个模块

实现

服务端的 NodeJS
Browserify，浏览器端的 CommonsJS 实现，可以使用 NPM 的模块，但是编译打包后的文件体积较大
modules-webmake，类似 Browserify，但不如 Browserify 灵活
wreq，Browserify 的前身

AMD

Asynchronous Module Definition 规范其实主要一个主要接口 define(id?, dependencies?, factory)；它要在声明模块的时候指定所有的依赖 dependencies，并且还要当做形参传到 factory 中，对于依赖的模块提前执行。

```
define("module", ["dep1", "dep2"], function(d1, d2) {
  return someExportedValue;
});
require(["module", "../file.js"], function(module, file) {});
```

适合在浏览器环境中异步加载模块
可以并行加载多个模块

缺点

提高了开发成本，代码的阅读和书写比较困难，模块定义方式的语义不畅
不符合通用的模块化思维方式，是一种妥协的实现

实现

RequireJS
curl

CMD

Commons Module Definition 规范和 AMD 很相似，尽量保持简单，并与 CommonsJS 和 NodeJS 的 Modules 规范保持了很大的兼容性。

```
define(function(require, exports, module) {
  var $ = require("jquery");
  var Spinning = require("./spinning");
  exports.doSomething = ...;
  module.exports = ...;
});
```

优点

依赖就近，延迟执行
可以很容易在 NodeJS 中运行

缺点

依赖 SPM 打包，模块的加载逻辑偏重

实现

Sea.js
coolie

ES6 模块

EcmaScript6 标准增加了 JavaScript 语言层面的模块体系定义。 ES6 模块的设计思想，是尽量静态化，使编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonsJS 和 AMD 模块，都只能在运行时确定这些东西。

```
import "jquery";
export function doStuff() {}
module "localModule" {}
```

优点

容易进行静态分析
面向未来的 EcmaScript 标准

缺点

原生浏览器端还没有实现该标准
全新的命令，新版的 NodeJS 才支持

实现

Babel

期望的模块系统

模块风格，尽量可以利用已有的代码，不仅仅只是 JavaScript 模块化，还有 CSS、图片、字体等资源也需要模块化。

安装 WebPack

安装

```
npm install webpack -g
npm install webpack-cli -g
```

配置

创建 webpack.config.js 配置文件

entry：入口文件，指定 WebPack 用哪个文件作为项目的入口
output：输出，指定 WebPack 把处理完成的文件放置到指定路径
module：模块，用于处理各种类型的文件
plugins：插件，如：热更新、代码重用等
resolve：设置路径指向
watch：监听，用于设置文件改动后直接打包

```
module.exports = {
    entry: "",
    output: {
        path: "",
        filename: ""
    },
    module: {
        loaders: [
            {test: /\.js$/, loader: ""}
        ]
    },
    plugins: {},
    resolve: {},
    watch: true
}
```

执行

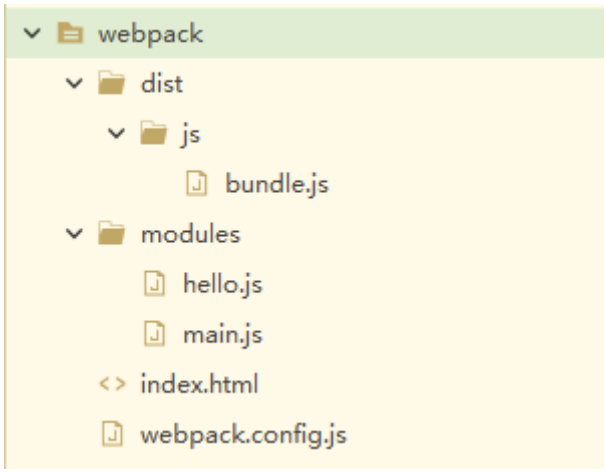
直接运行 webpack 命令打包

使用 WebPack

概述

使用 WebPack 打包项目非常简单，主要步骤如下：

创建项目
创建一个名为 modules 的目录，用于放置 JS 模块等资源文件
创建模块文件，如 hello.js ，用于编写 JS 模块相关代码
创建一个名为 main.js 的入口文件，用于打包时设置 entry 属性
创建 webpack.config.js 配置文件，使用 webpack 命令打包
创建 HTML 页面，如 index.html ，导入 WebPack 打包后的 JS 文件
运行 HTML 看效果



模块代码

创建一个名为 hello.js 的 JavaScript 模块文件，代码如下：

```
exports.sayHi = function () {
  document.write("<div>Hello WebPack</div>");
};
```

入口代码

创建一个名为 main.js 的 JavaScript 入口模块，代码如下：

```
var hello = require("./hello");
hello.sayHi();
```

配置文件

创建名为 webpack.config.js 的配置文件，代码如下：

```
module.exports = {
    entry: "./modules/main.js",
    output: {
        filename: "./js/bundle.js"
    }
};
```

HTML

创建一个名为 index.html，代码如下：


```
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <script src="dist/js/bundle.js"></script>
    </body>
</html>
```

打包

```
# 用于监听变化
webpack --watch
```

运行

运行 HTML 文件，你会在浏览器看到：

```
Hello WebPack
```

vue-router

什么是vue-router

简介

Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

嵌套的路由/视图表
模块化的、基于组件的路由配置
路由参数、查询、通配符
基于 Vue.js 过渡系统的视图过渡效果
细粒度的导航控制
带有自动激活的 CSS class 的链接
HTML5 历史模式或 hash 模式，在 IE9 中自动降级
自定义的滚动条行为

安装

vue-router 是一个插件包，所以我们还是需要用 npm/cnpm 来进行安装的。打开命令行工具，进入你的项目目录，输入下面命令。

```
npm install vue-router --save-dev
```

如果在一个模块化工程中使用它，必须要通过 `Vue.use()` 明确地安装路由功能：

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

Vue.use(VueRouter);
```

以下案例在 vue-cli 项目中使用 vue-router

创建组件页面

创建一个名为 src/components 的目录专门放置我们开发的 Vue 组件，在 src/components 目录下创建一个名为 Content.vue 的组件，代码如下：

```
<template>
  <div>
    我是内容页
  </div>
</template>

<script>
  export default{
    name:"Content"
  }
</script>
<style scoped>
</style>
```

安装路由

创建一个名为 src/router 的目录专门放置我们的路由配置代码，在 src/router 目录下创建一个名为 index.js 路由配置文件，代码如下：

```
import Vue from 'vue'
// 导入路由插件
import Router from 'vue-router'
// 导入自定义的组件
import Content from '../components/Content'
// 安装路由
Vue.use(Router);

// 配置路由
export default new Router({
  routes: [{
    // 路由路径
    path: '/content',
    // 路由名称
    name: 'Content',
    // 跳转的组件
    component: Content
  }]
});
```

配置路由

修改 main.js 入口文件，增加配置路由的相关代码

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'
import VueRouter from 'vue-router'
```

```
import router from './router'

Vue.use(VueRouter);

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  // 配置路由
  router,
  components: { App },
  template: '<App/>'
})
```

使用路由

修改 App.vue 组件，代码如下：

```
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <!-- 路由链接 -->
    <router-link to="/">首页</router-link>
    <router-link to="/content">内容页</router-link>
    <!-- 路由组件 -->
    <router-view />
  </div>
</template>

<script>

export default {
  name: 'App'
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

说明：

router-link： 默认会被渲染成一个 <a> 标签， to 属性为指定链接
router-view： 用于渲染路由匹配到的组件

效果演示

整合ElementUI

创建工程

创建一个名为 vue-elementui 的工程

```
# 使用 webpack 打包工具初始化一个名为 vue-elementui 的工程
vue init webpack vue-elementui
```

```
(c) 2019 Microsoft Corporation。保留所有权利。

D:\workspace\HBuilderProjects>vue init webpack vue-elementui

? Project name vue-elementui
? Project description A Vue.js project
? Author zsyue <244173220@qq.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) no

   vue-cli · Generated "vue-elementui".

# Project initialization finished!
# ========================

To get started:

  cd vue-elementui
  npm install (or if using yarn: yarn)
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack


D:\workspace\HBuilderProjects>
```

安装依赖

我们需要安装 vue-router 、 element-ui 、 sass-loader 和 node-sass 四个插件


```
cd vue-elementui
# 安装 vue-router
npm install vue-router --save-dev
# 安装 element-ui
npm i element-ui -S
# 安装 SASS 加载器
npm install sass-loader@7.3.1 node-sass --save-dev
# 安装项目所有依赖
npm install
```

说明

sass-loader 默认不指定版本是最新的 8.0.x， 运行可能会出现问题,所以指定版本安装

启动工程

```
npm run dev
```

```
ilderProjects\vue-elementui>npm run dev

> vue-elementui@1.0.0 dev D:\workspace\HBuilderProjects\vue-elementui
> webpack-dev-server --inline --progress --config build/webpack.dev.conf.js

 13% building modules 25/29 modules 4 active ...derProjects\vue-elementui\src\App.vue{ parser: "babylon" } is deprecated
; we now treat it as { parser: "babel" }.
 95% emitting

 DONE  Compiled successfully in 2802ms                                                                    16:51:56

 I  Your application is running here: http://localhost:8080
```

运行效果

在浏览器打开 http://localhost:8080 你会看到如下效果

附：NPM 相关命令说明

`npm install moduleName`：安装模块到项目目录下

`npm install -g moduleName`：-g 的意思是将模块安装到全局，具体安装到磁盘哪个位置，要看 npm config prefix 的位置

`npm install -save moduleName`：--save 的意思是将模块安装到项目目录下，并在 package 文件的 dependencies 节点写入依赖，`-S` 为该命令的缩写

`npm install -save-dev moduleName`：--save-dev 的意思是将模块安装到项目目录下，并在 package 文件的 devDependencies 节点写入依赖，`-D` 为该命令的缩写

编写ElementUI页面

目录结构

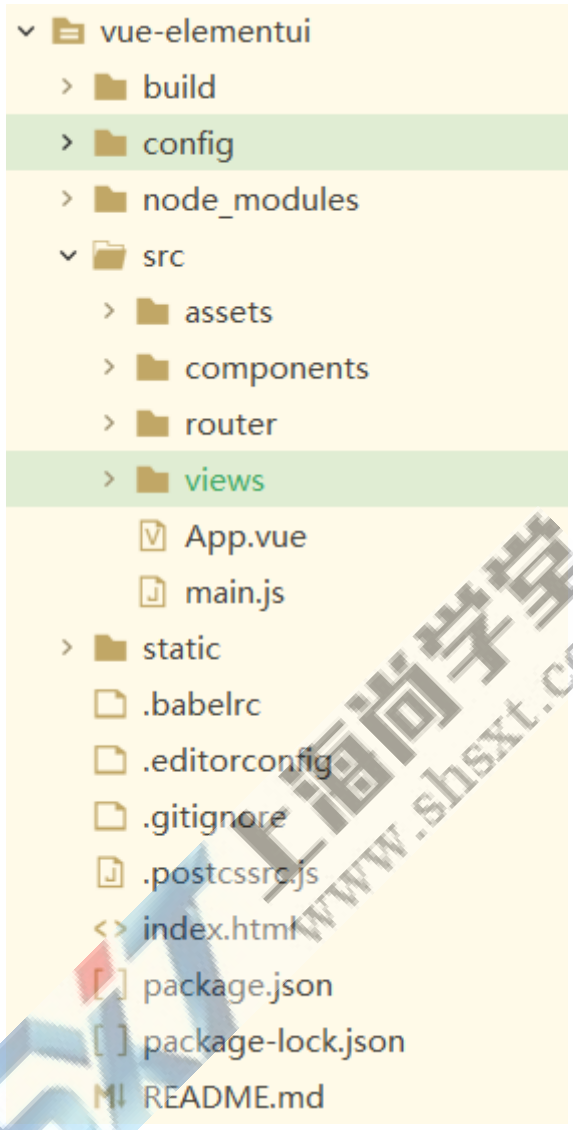
在源码目录中创建如下结构：

用于存放资源文件

components：用于存放 Vue 功能组件

views：用于存放 Vue 视图组件

router：用于存放 vue-router 配置



创建视图

创建首页视图

在 views 目录下创建一个名为 Main.vue 的视图组件；该组件在当前章节无任何作用，主要用于登录后展示登录成功的跳转效果；

```
<template>
  <div>
    我是首页
  </div>
</template>

<script>
  export default{
    name:'Main'
  }
</script>

<style>
</style>
```

在 views 目录下创建一个名为 Login.vue 的视图组件，其中 el-* 的元素为 ElementUI 组件；

```
<template>
  <div>
    <el-form ref="form" :model="form" :rules="rules" label-width="80px"
class="login-box">
      <h3 class="login-title">欢迎登录</h3>
      <el-form-item label="账号" prop="name">
        <el-input v-model="form.name" placeholder="请输入用户名" type="text"></el-
input>
      </el-form-item>
      <el-form-item label="密码" prop="password">
        <el-input v-model="form.password" placeholder="请输入密码"
type="password"></el-input>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" @click="onSubmit('form')">登录</el-button>
      </el-form-item>
    </el-form>
  </div>
</template>

<script>
  export default {
    name: 'Login',
    data() {
      return {
        form: {
          name: '',
          password: '',
        },
        rules: {
          name: [{
            required: true,
            message: '请输入用户名',
            trigger: 'blur'
          }],
          password: [{
            required: true,
            message: '请输入密码',
            trigger: 'blur'
          }]
        }
      };
    },
    methods: {
      onSubmit(form) {
        this.$refs[form].validate((valid) => {
          if (valid) {
            this.$router.push("/main");
          } else {
            this.$message.error('请输入用户名或密码');
            return false;
          }
        });
```

```
    }
  }
</script>

<style lang="scss" scoped>
  .login-box {
    // 宽度
    width: 400px;
    // 边距
    margin: 200px auto;
    // 边框
    border: 1px solid #DCDFE6;
    // 内边距
    padding: 40px;
    // 边框圆角
    border-radius: 10px;
    // 阴影
    box-shadow: 0 0 30px #DCDFE6;
  }

  .login-title {
    // 文本居中
    text-align: center;
  }
</style>
```

创建路由

在 router 目录下创建一个名为 index.js 的 vue-router 路由配置文件

```
import Vue from 'vue'
import Router from 'vue-router'
import Login from '../views/Login'
import Main from '../views/Main'

Vue.use(Router);

export default new Router({
  routes:[
    {
      // 登录页
      path:'/login',
      name:'Login',
      component:Login
    },
    {
      // 首页
      path:'/main',
      name:'Main',
      component:Main
    }
  ]
});
```

配置路由

修改 main.js 入口代码

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'
import VueRouter from 'vue-router'
// 引入ElementUI
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
import router from './router'

Vue.use(VueRouter);
// 安装ElementUI
Vue.use(ElementUI);

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  // 使用ELementUI
  render: h => h(App)
})
```

修改 App.vue 组件代码

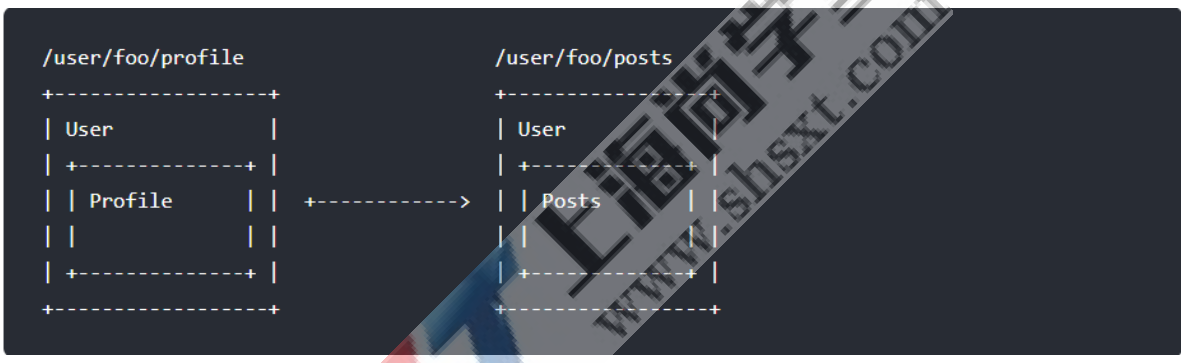
```
<template>
  <div id="app">
    <router-view />
  </div>
</template>

<script>

export default {
  name: 'App'
}
</script>
```

效果演示

在浏览器打开 http://localhost:8080/#/login 你会看到如下效果

嵌套路由

嵌套路由又称子路由，在实际应用中，通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件，例如：



创建嵌套视图组件

用户信息组件

在 views/members 目录下创建一个名为 MemberLevel.vue 的视图组件；该组件在当前章节无任何作用，主要用于展示嵌套效果；

```
<template>
  <div>
    会员等级
  </div>
</template>

<script>
  export default{
    name:'MemberLevel'
  }
</script>

<style>
</style>
```

用户列表组件

mbers 目录下创建一个名为 MemberList.vue 的视图组件；该组件无任何作用，主要用于展示嵌套效果；

```
<template>
  <div>
    会员列表
  </div>
</template>

<script>
  export default{
    name:'MemberList'
  }
</script>

<style>
</style>
```

配置嵌套路由

修改 router 目录下的 index.js 路由配置文件，代码如下：

```
import Vue from 'vue'
import Router from 'vue-router'
import Login from '../views/Login'
import Main from '../views/Main'
// 用于嵌套路由的组件
import MemberList from '../views/members/MemberList'
import MemberLevel from '../views/members/MemberLevel'

Vue.use(Router);

export default new Router({
  routes:[
    {
      // 登录页
      path:'/login',
      name:'Login',
      component:Login
    },
    {
      // 首页
      path:'/main',
      name:'Main',
      component:Main,
      // 嵌套路由
      children:[
        {
          // 会员等级
          path:'/members/level',
          name:'MemberLevel',
          component:MemberLevel
        },
        {
          // 会员列表
          path:'/members/list',
          name:'MemberList',
```

```
component:MemberList
            }
        ]
      }
    ]
})
```

说明：主要在路由配置中增加了 children 数组配置，用于在该组件下设置嵌套路由

修改首页视图

修改 Main.vue 视图组件，此处使用了 ElementUI 布局容器组件，代码如下：

```
<template>
  <div>
    <el-container>
      <el-aside width="200px">
        <el-menu :default-openeds="['1']">
          <el-submenu index="1">
            <template slot="title"><i class="el-icon-message"></i>会员管理
</template>
            <el-menu-item index="1-1">
              <router-link to="/members/list">会员列表</router-link>
            </el-menu-item>
            <el-menu-item index="1-2">
              <router-link to="/members/level">会员等级</router-link>
            </el-menu-item>
          </el-submenu>
          <el-submenu index="2">
            <template slot="title"><i class="el-icon-message"></i>商品管理
</template>
            <el-menu-item index="2-1">商品分类</el-menu-item>
            <el-menu-item index="2-2">商品列表</el-menu-item>
          </el-submenu>
        </el-menu>
      </el-aside>
      <el-container>
        <el-header style="text-align: right; font-size: 12px">
          <el-dropdown>
            <i class="el-icon-setting" style="margin-right: 15px"></i>
            <el-dropdown-menu slot="dropdown">
              <el-dropdown-item>用户中心</el-dropdown-item>
              <el-dropdown-item>退出登录</el-dropdown-item>
            </el-dropdown-menu>
          </el-dropdown>
          <span>admin</span>
        </el-header>
        <el-main>
          <router-view />
        </el-main>
      </el-container>
    </el-container>
  </div>
</template>

<script>
  export default {
```



```
    'Main'
  }
</script>

<style>
  .el-header {
    background-color: #B3C0D1;
    color: #333;
    line-height: 60px;
  }

  .el-aside {
    color: #333;
  }
</style>
```

说明：

在 <el-main> 元素中配置了 <router-view /> 用于展示嵌套路由

主要使用 <router-link to="/members/list">会员列表</router-link> 展示嵌套路由内容

效果演示

参数传递

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个 User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。此时我们就需要传递参数了；

使用路径匹配的方式

修改路由配置

```
{
    // 会员等级
    path:'/members/level/:id',
    name:'MemberLevel',
            component:MemberLevel
}
```

说明：主要是在 path 属性中增加了 :id 这样的占位符

router-link

```
<router-link :to="{name:'MemberLevel',params:{id:2}}">会员等级</router-link>
```

说明：此时我们将 to 改为了 :to，是为了将这一属性当成对象使用，注意 **router-link 中的 name 属性名称** 一定要和 **路由中的 name 属性名称** 匹配，因为这样 Vue 才能找到对应的路由路径；

代码方式

```
this.$router.push({ name: 'MemberLevel', params: {id: 2}});
```

接收参数

在目标组件中使用

```
{{ $route.params.id }}
```

来接收参数

使用 props 的方式

修改路由配置

```
{path: '/member/level/:id', name:'MemberLevel', component: UserProfile, props:
true}
```

说明：主要增加了 props: true 属性

传递参数

同上

接收参数

为目标组件增加 props 属性，代码如下：

```
export default {
  props: ['id'],
  name: "MemberLevel"
}
```

模板中使用

```
{{ id }}
```

接收参数

组件重定向

重定向的意思大家都明白，但 Vue 中的重定向是作用在路径不同但组件相同的情况下

配置重定向

修改路由配置

```
      path: '/main',
      name: 'Main',
      component: Main
    },
    {
      path: '/goMain',
      redirect: '/main'
    }
```

说明：这里定义了两个路径，一个是 /main ，一个是 /goMain，其中 /goMain 重定向到了 /main 路径，由此可以看出重定向不需要定义组件；

重定向到组件

设置对应路径即可

```
<router-link to="/goMain">回到首页</router-link>
```

带参数的重定向

修改路由配置

```
    {
      // 首页
      path: '/main/:name',
      name: 'Main',
      component: Main
    },
    {
      path: '/goMain/:name',
      redirect: '/main/:name'
    }
```

重定向到组件

```
<router-link to="/goMain/admin123">回到首页</router-link>
```

路由模式

路由模式有两种

hash：路径带 # 符号，如 http://localhost/#/login
history：路径不带 # 符号，如 http://localhost/login

修改路由配置，代码如下：

```
export default new Router({
  mode: 'history',
  routes: [
  ]
});
```

处理 404

NotFound.vue 的视图组件，代码如下：

```
<template>
  <div>
    404
  </div>
</template>

<script>
  export default{
    name:'NotFound'
  }
</script>

<style>
</style>
```

修改路由配置，代码如下：

```
{
    path: '*',
    component: NotFound
}
```

路由中的钩子函数

beforeRouteEnter：在进入路由前执行

beforeRouteLeave：在离开路由前执行

案例代码如下：

```
<script>
  export default{
    props:['id'],
    name:'MemberLevel',
    beforeRouteEnter: (to,from,next) => {
      console.log("进入会员等级页面");
      next();
    },
    beforeRouteLeave:(to,from,next)=>{
      console.log("离开会员等级页面");
      next();
    }
  }
</script>
```

参数说明：

to：路由将要跳转的路径信息

from：路径跳转前的路径信息

next：路由的控制参数

next() 跳入下一个页面

next('/path') 改变路由的跳转方向，使其跳到另一个路由

next(false) 返回原来的页面

xt((vm)=>{}) 仅在 beforeRouteEnter 中可用，vm 是组件实例

效果演示

在钩子函数中使用异步请求

安装 Axios

```
npm install axios -s
```

引用 Axios

```
import axios from 'axios'
Vue.prototype.axios = axios;
```

在 beforeRouteEnter 中进行异步请求，案例代码如下：

```
<script>
  export default{
    props:['id'],
    name:'MemberLevel',
    beforeRouteEnter: (to,from,next) => {
      console.log("进入会员等级页面");
      // 注意，一定要在 next 中请求，因为该方法调用时 Vue 实例还没有创建，此时无法获取到
this 对象，在这里使用官方提供的回调函数拿到当前实例
      next(vm=>{
        vm.getData();
      });
    },
    beforeRouteLeave:(to,from,next)=>{
      console.log("离开会员等级页面");
      next();
    },
    methods:{
      getData:function(){
        this.axios({
          method:'get',
          url:'http://localhost:8080/data.json',
          }).then(function(repos){
            console.log(repos);
          }).catch(function(error){
            console.log(error);
          });
      }
    }
  }
</script>
```

说明：出现这种情况说明已经请求成功了，但是遇到跨域问题。这个问题等我们到后面讲到后端再去解决跨域问题

vuex

Vuex 是一个专为 Vue.js 应用程序开发的 **状态管理模式。**它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

安装

在项目根目录执行如下命令来安装 Vuex

```
npm install vuex --save
```

修改 main.js 文件，导入 Vuex，关键代码如下：

```
import Vuex from 'vuex'
Vue.use(Vuex);
```

判断用户是否登录

我们利用路由钩子 beforeEach 来判断用户是否登录，期间会用到 sessionStorage 存储功能

修改 Login.vue

在表单验证成功方法内增加如下代码：

```
// 设置用户登录成功
sessionStorage.setItem('isLogin', 'true');
```

修改 main.js

利用路由钩子 beforeEach 方法判断用户是否成功登录，关键代码如下：

```
// 路由跳转前执行
router.beforeEach((to,from,next)=>{
  // 获取用户登录状态
    let isLogin = sessionStorage.getItem('isLogin');

    // 注销
    if(to.path=='/logout'){
    // 清空
        sessionStorage.clear();
    // 跳转到登录页面
```

```
        ext({path:'/login'});
      }else if(to.path=='/login'){
      if(isLogin != null){
        next({path:'/main'});
      }
    }else if(isLogin == null){
      next({path:'/login'});
    }
    // 下一个路由
      next();
  })
```

配置 vuex

创建 Vuex 配置文件

在 src 目录下创建一个名为 store 的目录并新建一个名为 index.js 文件用来配置 Vuex，代码如下：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex);

// 全局state对象,用于保存所有组件的公共数据
const state={
    user:{
        name:''
    }
};

// 监听state值的最新状态（计算属性）
const getters={
    getUser(state){
        return state.user;
    }
};

// 唯一可以改变state值的方法(同步执行)
const mutations={
    updateUser(state,user){
        state.user=user;
    }
}

// 异步执行mutations方法
const actions={
    asyncUpdateUser(context,user){
        context.commit('updateUser',user);
    }
}


export default new Vuex.Store({
    state,
    getters,
    mutations,
```

```
ns
})
```

修改 main.js 增加刚才配置的 store/index.js，关键代码如下：

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'
import VueRouter from 'vue-router'
// 引入ElementUI
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
import router from './router'
import axios from 'axios'
import Vuex from 'vuex'
import store from './store'

Vue.use(VueRouter);
// 安装ElementUI
Vue.use(ElementUI);
// 安装axios
Vue.prototype.axios=axios;
// 安装vuex
Vue.use(Vuex);

// 路由跳转前执行
router.beforeEach((to,from,next)=>{
  // 获取用户登录状态
    let isLogin = sessionStorage.getItem('isLogin');

    // 注销
    if(to.path=='/logout'){
    // 清空
        sessionStorage.clear();
    // 跳转到登录页面
        next({path:'/login'});
    }else if(to.path=='/login'){
    if(isLogin != null){
      next({path:'/main'});
    }
  }else if(isLogin == null){
    next({path:'/login'});
  }
  // 下一个路由
    next();
})

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  // 使用ELementUI
  render: h => h(App)
```


Login.vue

```
methods: {
      submitForm(formName) {
        this.$refs[formName].validate((valid) => {
          if (valid) {
            sessionStorage.setItem('isLogin','true');
            this.$store.dispatch('asyncUpdateUser',{name:this.form.name})
            this.$router.push("/main")
          } else {
            this.$message.error('请输入用户名或密码！');
            return false;
          }
        });
      },
    }
```

Main.vue

```
<span>{{$store.getters.getUser.name}}</span>
```

解决浏览器刷新后 Vuex 数据消失问题

问题描述

Vuex 的状态存储是响应式的，当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。但是有一个问题就是：vuex 的存储的数据只是在页面的中，相当于我们定义的全局变量，刷新之后，里边的数据就会恢复到初始化状态。但是这个情况有时候并不是我们所希望的。

解决方案

监听页面是否刷新，如果页面刷新了，将 state 对象存入到 sessionStorage 中。页面打开之后，判断 sessionStorage 中是否存在 state 对象，如果存在，则说明页面是被刷新过的，将 sessionStorage 中存的数据取出来给 vuex 中的 state 赋值。如果不存在，说明是第一次打开，则取 vuex 中定义的 state 初始值。

修改代码

在 App.vue 中增加监听刷新事件

```
export default {
  name: 'App',
  mounted() {
    window.addEventListener('unload', this.saveState);
  },
  methods: {
    saveState() {
      sessionStorage.setItem('state', JSON.stringify(this.$store.state));
    }
  }
}
```

修改 store/index.js 中的 state


```
  state = sessionStorage.getItem('state') ?
  JSON.parse(sessionStorage.getItem('state')) : {
    user: {
      username: ''
    }
  };
```

模块化

由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。当应用变得非常复杂时，store 对象就有可能变得相当臃肿。为了解决以上问题，Vuex 允许我们将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割

创建 user 模块

在 store 目录下创建一个名为 modules 的目录并创建一个名为 user.js 的文件，代码如下：

```
const user = {
  // 全局state对象,用于保存所有组件的公共数据
  state: sessionStorage.getItem('userState') ?
JSON.parse(sessionStorage.getItem('userState')) : {
    user: {
      name: ''
    }
  },

  // 监听state值的最新状态（计算属性）
  getters: {
    getUser(state) {
      return state.user;
    }
  },

  // 唯一可以改变state值的方法(同步执行)
  mutations: {
    updateUser(state, user) {
      state.user = user;
    }
  },

  // 异步执行mutations方法
  actions: {
    asyncUpdateUser(context, user) {
      context.commit('updateUser', user);
    }
  }
}


export default user;
```

修改 store/index.js

```
import Vue from 'vue'
import Vuex from 'vuex'
import user from './modules/user'

Vue.use(Vuex);

export default new Vuex.Store({
  modules:{
    user
  }
})
```

备注：由于组件中使用的是 getters 和 actions 处理，所以调用代码不变

修改 App.vue

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

<script>

export default {
  name: 'App',
  mounted() {
    window.addEventListener('unload',this.saveState);
  },
  methods:{
    saveState(){

 sessionStorage.setItem('userState',JSON.stringify(this.$store.state.user));
    }
  }
}
</script>
```