信息工程系

教

案

课程名称：前端框架

教　　师：黄苗苗

总 学 时 ：72

理论学时：36

实训学时：36

上课班级：计算机 241、3+ 241

授课学期：25-26（1）

教 案

黄 苗 苗

课 程 性 质

Vue.js 是一个构建数据驱动的 web 界面的渐进式框架。Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它不仅易于上手，还便于与第三方库或既有项目整合。

另一方面，当与单文件组件和 Vue 生态系统支持的库结合使用时，Vue 也完全能够为复杂的单页应用程序提供驱动。

Vue.js 自身不是一个全能框架——它只聚焦于视图层。因此它非常容易学习，非常容易与其它库或已有项目整合。另一方面，在与相关工具和支持库一起使用时，Vue.js 也能完美地驱动复杂的单页应用。

课程要求侧重掌握为将来从事网页开发、手机应用开发、网络程序应用开发等提供必要的基础知识，打下良好的基础。因此要求上述专业学生都必须掌握本课程的内容。

课程要求：

1. 熟练使用 HTML、CSS 技术
2. 熟练掌握 JavaScript 技术
3. 掌握 Vue.js 的基本语法、框架应用
4. 应用 Vue.js 进行简单 SPA 开发应用设计

课程性质：

计算机基础专业课程，本课程是基于 JavaScript 技术基础上的延伸课程，是当前网络应用开发的前端开发的知识准备，需要学生对此投入较大的学习量，并将其应用到实际操作中才能切实掌握该技术。

课程教学手段

依据课程性质及周课时安排：

理论课（2 节/周）：

课程主要是从总体概述之后，首先导入 Vue.js 的基础语法，最后结合应用，介绍如何使用 Vue.js 框架式前端开发设计。

结合课程的各层结构，在每个章节的授课中，分几个主要方式进行：

- **应用环境导入**：结合知识点，导入可能见到的情境，对情景进行剖析，结合学生掌握点，分析已经解决问题及函待处理问题，由此导入新知识点。
- **承上启下式进入**：对于上一个章节，进行总结，然后在上一章节的基础上，对新的可能出现的情境进行引导式引入，然后就新的问题的本质、现象出现的方式以及其如何工作引入思考，然后进入章节介绍。

引入章节之后，对于每个技术要点，依据各个特点进行逐一详细解析，帮助学习者尽快进入新知识的学习。主要的各要点可归纳总结如下：

- 技术解决问题情境
- Vue.js 相关知识点
- 应用实例简介
- Vue.js 该技术综合实例设计剖析

实验课（2 节/周）：

针对本课程特性，实操性实验课是检验学生掌握程度和加深学生技能的重要过程。

实际结合网络知识以及日常工作可能布置网络应用设计情况设计。需要从浅入深，有基础语法的小实验一步一步扎实练习，最后才能汇总为大的设计完成。

实验课的难度系数属于中等难度系数，需学生课后配合多加练习。

课程思政体现：

课程结合当前应用情景及设计者（中国人）特征，倡导学生热爱祖国，热爱专业。并倡导专业学习尊重学生个性及兴趣爱好。以实际应用情境为设计目的，辅助学生专业素养能力培养。

第〇篇 课程概述

★ 计划课时：

1 学时

★ 教学目的：

1. 介绍课程
2. 认识课程地位

★ 教学重点、难点：

① 认识 Vue.js

★ 教学内容：

❀ 课程设置

- 课时：72
- 实验基本要求独立完成
- 考查

❀ 课程教学

- 引入
- 基础语法
- 重点/常用技能
- 综合运用

❀ 课程建议参考资源

- 《Vue.js 从入门到项目》
  - 清华大学出版社 刘汉伟编
  - 图书馆有关于 Vue.js 源代码分析书籍

- Vue.js 教程|菜鸟教程
  - https://www.runoob.com/vue2/vue-tutorial.html
- 知乎
- 微信公众号
- Vue.js 官方网站 https://cn.vuejs.org/

第一篇 运行环境

★ 计划课时：

2 学时

★ 教学目的：

1. 了解什么是前端开发
2. 认识 Vue.js 的整体框架

★ 教学重点、难点：

① 前端开发术语
② 数据反应
③ 环境配置

★ 教学内容：

- Vue 简介
- 运行环境
- 安装环境

❀ 简介

❖ 什么是 Vue

❖ Vue：
- Vue (读音 /vju:/，类似于 view)
- 是一套用于构建用户界面的渐进式框架——JavaScript 框架
- Vue 被设计为可以自底向上逐层应用

❖ Vue 有什么：

- **Have a look:**
  - **What is vue?**
    - **https://player.youku.com/embed/XMzMwMTYyODMyNA==?autoplay=true&client_id=37ae6144009e277d**

前端技术简介

- **前端开发是创建 Web 页面或 app 等前端界面呈现给用户的过程，通过 HTML、CSS 及 JavaScript 以及衍生出来的各种技术、框架、解决方案，来实现互联网产品的用户界面交互。**
- **前端开发从网页制作演变而来，早期网站主要内容都是静态，以图片和文字为主，用户使用网站的行为也以浏览为主。随着互联网技术的发展和 HTML5、CSS3 的应用，现代网页更加美观，交互效果显著，功能更加强大。**
- **Vue.js**
  - 是用于构建交互式的 Web 界面的库。
  - 它提供了 MVVM 数据绑定和一个可组合的组件系统，具有简单、灵活的 API
- **MVC**
  - Model-View-Controller
  - 模型-视图-控制器 模式
  - MVC 中：
    - M 专注于数据，
    - V 专注于表达，
    - C 则在 M 与 V 之间架起了一座桥梁，主要负责收集用户数据的数据，想相关模型请求数据并返回响应的视图来完成交互请求
- **MVVM**
  - Model-View-ViewMode
  - 模型-视图-视图模型
    - 

  - 区别于 MVC

- 分离视图 view 和模型 model
- Viewmodel 层封装了界面展示和操作的属性和接口
  - ViewModel 是 Vue.js 的核心，它是一个 Vue 实例
  - ViewModel 是双向绑定的

❖ **使用 Vue 的过程就是定义 MVVM 各个组成部分的过程**

- **定义 View→定义 Model →创建一个 Vue 实例或”ViewModel”→Vue 选项设置**

❖ **当前主要流行框架：--摘自简书**

■ Angular：Angular 是一个完整的框架
- 未来的发展趋势是前端后端只靠 json 数据来进行通信：后端只处理和发送一段 json 数据到前端，然后计算和模板渲染都在前端进行。而前端的改动后，形成 json 数据然后传回到后端
- AngularJS 的作用简单说就是就是把后台的 json 值直接用 html 进行渲染，然后 html 的操作又直接在形成 json 传回后台。未来的后台 MVC，试图不再是模板了，而是一段结构整齐标准的 JSON，而这个 JSON 作为前台的 model 直接在 AngularJS 直接使用。或者说后台的试图是前台的模型，而整个前台就是后台的视图。后台程序再也不做模板的任何处理了。

■ Reac：
- React 以 JavaScript 为中心
- React 是一个类库

■ Vue:
- 渐进式的！
- 一个轻量级的 MVVM 框架

❖ **单页应用：**

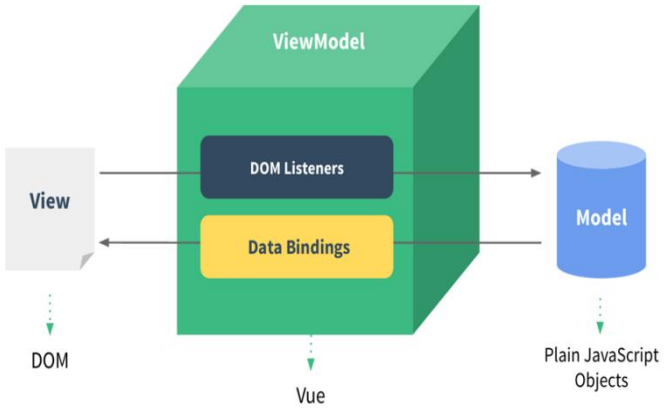
■ **SPA**: Single Page Web Application
- 单页面跳转仅仅刷新局部资源，公共资源等仅需加载一次
- 对应的传统页面方式：**MPA**——需要刷新页面上所有资源

❀ 运行环境

❖ NPM

❖ **VUE 2.0 推荐开发环境**

❖ **Vue 的 MVVP 模型**

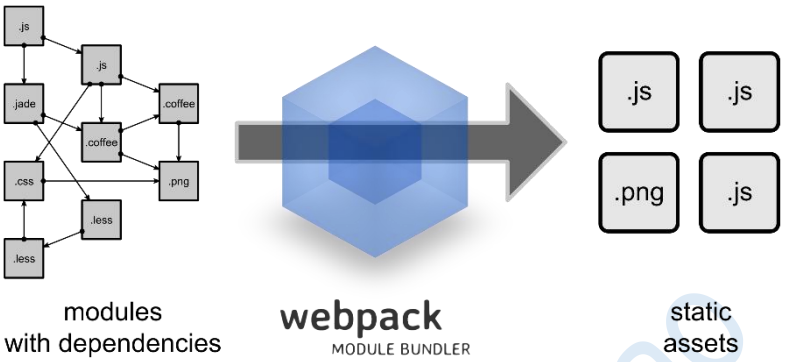


- Node.js：
  - Node.js® 是一个基于 Chrome V8 引擎 的 JavaScript 运行时。
    - node.js 是一个让 JavaScript 运行在服务端的开发平台，JavaScript 成为与 PHP、Python、Perl、Ruby 等服务端语言平起平坐的脚本语言
    - nodejs 作为一个新兴的前端框架，后台语言，有很多吸引人的地方：
      - RESTful API\单线程\非阻塞 IO\V8 虚拟机\事件驱动
- NODE.JS
  - 官网：
    - https://nodejs.org/zh-cn/
- NPM
  - Node Package Manager
  - 是开源库生态系统，集成在 Node.js 中，是它的包管理器
    - NPM 是随同 NodeJS 一起安装的包管理工具
  - 推荐学习网址：
    - https://www.runoob.com/nodejs/nodejs-npm.html
- NPM 仓库在国外，通常使用的是国内淘宝镜像源的 CNPM 来下载依赖
- NPM 主要管理命令：
  - npm –version
    - 测试是否成功安装
  - npm init
    - 创建项目
  - npm istall XXX --save
    - 安装项目依赖，模块开发完成后依旧需要
  - npm istall XXX --save -dev
    - 安装项目依赖，模块开发完成后不在需要

WEBPACK

- WebPack 可以看做是模块打包机
  - 它做的事情是：
    - 分析你的项目结构，找到 JavaScript 模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript 等），并将其打包为合适的格式以供浏览器使用
- Why Webpack?
  - 简化开发的复杂度

- 模块化
- 使我们能够实现目前版本的 JavaScript 不能直接使用的特性，并且之后还能能装换为 JavaScript 文件使浏览器可以识别
  - Webpack 可以将多种静态资源 js css less 等转换为一个静态文件，减少页面的请求，同时，也减少转义 less 或 ES6 语法等工作，大大提高开发效率



VISO STUDIO CODE

简介：

- Visual Studio Code （简称 VS Code）
  - 是由微软研发的一款免费的、轻量级的 Web 集成开发环境，且开源的跨平台文本（代码）编辑器，
  - 能运行在 Linux、Mac 、Windows 系统上。
  - 在智能提示变量类型,函数定义,模块方面继承了 visio studio 的优秀传统，
  - 在断点调试上也有不错的表现
    - 原文链接：https://blog.csdn.net/dong123dddd/java/article/details/53688917

建议安装插件：

- **Vetur** —— 语法高亮、智能感知、Emmet 等
- **EsLint** —— 语法纠错
- **Debugger for Chrome** —— 映射 vscode 上的断点到 chrome 上，方便调试
- **Auto Close Tag** —— 自动闭合 HTML/XML 标签
- **Auto Rename Tag** —— 自动完成另一侧标签的同步修改
- **JavaScript(ES6) code snippets** —— ES6 语法智能提示以及快速输入
- **Path Intellisense** —— 自动路劲补全
- **HTML CSS Support** —— 让 html 标签上写 class 智能提示当前项目所支持的样式

安装环境

- 下载 VS CODE STUDIO
- 下载安装 node.js

- 下载安装 GIT BASH 脚手架架构器
- 在 GIT BASH 里安装 NPM
- 建立项目，搭造依赖
- 运行

第二篇 基础语法

★ 计划课时：

10 学时

★ 教学目的：

1. 为全面使用 Vue.js 准备
2. 基础语法

★ 教学重点、难点：

① ES6
② Mustache
③ 指令
④ 属性

★ 教学内容：

① ECMAScript 6语法简介

❀ 简介

○ **ES6**

☆ 全称 ECMAScript 6.0 ，是 JavaScript 的下一个版本标准，2015.06 发版。主要是为了解决 ES5 的先天不足。

○ **目标：**

☆ 适应更复杂的应用；
☆ 实现代码库之间的共享；
☆ 不断迭代维护新版本

❀ 本章主要内容

○ **let VS const**

- 模板字面量
- 参　数
- 展开运算符
- 对象字面量语法
- 解构赋值
- 箭头函数
- promise

let VS const

- **ES2015(ES6) 新增加了两个重要的 JavaScript 关键字**
  - ☆ let：
    - ● 声明的变量只在 let 命令所在的代码块内有效
  - ☆ const：
    - ● 声明一个只读的常量，一旦声明，常量的值就不能改变
- **let 命令：**
  - ☆ let 是在代码块内有效，var 是在全局范围内有效
  - ☆ let 只能声明一次 var 可以声明多次
  - ☆ let 不存在变量提升，var 会变量提升

```
let a = 1;
let a = 2;
var b = 3;
var b = 4;
a  // Identifier 'a' has already been declared
b  // 4
```

  - ☆
- **const 命令**
  - ☆ 声明一个只读变量，声明之后不允许改变
    - ● 意味着，一旦声明必须初始化，否则会报错
  - ☆ 其实 const 其实保证的不是变量的值不变，而是保证变量指向的内存地址所保存的数据不允许改动。
    - ● **简单类型（数值 number、字符串 string 、布尔值 boolean）**
      - ● 值就保存在变量指向的那个内存地址，因此 const 声明的简单类型变量等同于常量
    - ● **复杂类型（对象 object，数组 array，函数 function）**
      - ● 变量指向的内存地址其实是保存了一个指向实际数据的指针
      - ● 所以 const 只能保证指针是固定的，至于指针指向的数据结构变不变就无法控制了，所以使用 const 声明复杂类型对象时要慎重
- **ES6 明确规定：**
  - ☆ 代码块内如果存在 let 或者 const，代码块会对这些命令声明的变量从块的开始就形成一个**封闭**作用域

模板字面量

- 模板字面量：用反撇号替换了单、双引号
- 多行字符串指令
- 字符串占位符

参 数

- 默认参数 rest 参数

- 默认参数
  - ☆ 规则：
    - 只有在未传递参数，或者参数为 undefined 时，才会使用默认参数，null 值被认为是有效的值传递
    - 使用函数默认参数时，不允许有同名参数
    - 函数参数默认值存在**暂时性死区**（**TDZ**, Temporal Dead Zone），在函数参数默认值表达式中，还未初始化赋值的参数值无法作为其他参数的默认值

- rest 参数
  - ☆ 在函数命名参数前 添加…运算符表示一个不定参数
    - 该参数是一个数组
    - 包含自它之后传入的所有参数
    - 规则：
      - ➢ 每个函数最多只能声明一个不定参数
      - ➢ 不定参数放在所有参数的末尾
      - ➢ 不定参数不能用于对象字面量的 setter 中

展开运算符

- 功能：
  - ☆ 对运算的指定数组，将它们打散后，作为各自独立的参数传入函数

对象字面量

- 属性
  - ☆ ES6 允许对象的属性直接写变量
    - 这时候属性名是变量名，属性值是变量值

```
const age = 12;
const name = "Amy";
const person = {age, name};
person    //{age: 12, name: "Amy"}
//等同于
const person = {age: age, name: name}
```

- ☆ **ES6 允许用表达式作为属性名，但是一定要将表达式放在方括号内**

```
const obj = {
 ["he"+"llo"](){
   return "Hi";
  }
}
obj.hello();  //"Hi"
```

- 方法名
- 方法
  - ☆ Object.assign()
    - 语法：
      - ➢ Object.assign(target, source_1, •••)
    - 用于将源对象的所有可枚举属性复制到目标对象中
    - assign 的属性拷贝是浅拷贝
  - ☆ 注意点：
    - 如果目标对象和源对象有同名属性，或者多个源对象有同名属性，则后面的属性会覆盖前面的属性
    - 如果该函数只有一个参数，当参数为对象时，直接返回该对象；当参数不是对象时，会先将参数转为对象然后返回

```
let target = {a: 1};
let object2 = {b: 2};
let object3 = {c: 3};
Object.assign(target,object2,object3);
// 第一个参数是目标对象，后面的参数是源对象
target;  // {a: 1, b: 2, c: 3
```

❀ 解构赋值

- **解构赋值:**
  - ☆ 是对赋值运算符的扩展。
  - ☆ 是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。
  - ☆ 在代码书写上简洁且易读，语义更加清晰明了
  - ☆ 也方便了复杂对象中数据字段获取
  - ☆ 解构的源，解构赋值表达式的右边部分

☆ 解构的目标，解构赋值表达式的左边部分

- **数组解构**
- **对象解构**

箭头函数

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：

```
参数 => 函数体
```

- **规则：**
  - ☆ 当箭头函数没有参数或者有多个参数，要用 () 括起来
  - ☆ 当箭头函数函数体有多行语句，用 {} 包裹起来，表示代码块，
    - ● 当只有一行语句，并且需要返回结果时，可以省略 {}，结果会自动返回
  - ☆ 当箭头函数要返回对象的时候，为了区分于代码块，要用 () 将对象包裹
  - ☆ 箭头函数体中的 this 对象，是定义函数时的对象，而不是使用函数时的对象
  - ☆ 不可以作为构造函数，也就是不能使用 new 命令，否则会报错

基础语法

本章主要内容

- 如何引入 Vue.js
- Vue 实例
- 模板语法
- 指令绑定
- 条件语句
- 循环语句

如何引入 Vue.js

- **必须先准备好的环境**
  - ☆ Node.js
  - ☆ npm & cnpm
    - ● 使用其安装 Vue.js
      - ➢ cnpm install vue

☆ 引入：

- 通过<script>标签即可引入
  - Vue 会被注册为全局变量

<script>

Vue 实例

实例

☆ 每个 Vue 应用都是通过用 Vue 函数创建一个新的 Vue 实例开始的

- 语法：var vm = new Vue({…})

☆ 数据：

- 当一个 Vue 实例被创建时，它将 data 对象中的所有的 property 加入到 Vue 的响应式系统中。当这些 property 的值发生改变时，视图将会产生“响应”，即匹配更新为新的值。
  - 当这些数据改变时，视图会进行重渲染。
  - 值得注意的是只有当实例被创建时就已经存在于 data 中的 property 才是响应式的。

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5  </head>
6  <body>
7  <div id="app">
8      <h1>{{ title }}</h1>
9  </div>
10 <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
11 <script type="text/javascript">
12     var vm = new Vue({
13         el: '#app', // mount到DOM上
14         data () {
15             return {
16                 title: 'Hello World'
17             }
18         }
19     })
20 </script>
21 </body>
22 </html>
```

Vue 实例：

☆ el:

- 挂载的 DOM 元素

☆ data:

- 一般该对象所有的数据都需要在该选项中进行定义
- 定义后的数据可共后续其他 HTML 中脚本捆绑使用
- 数据，一般情况下，习惯定义为一个函数属性
- 语法：**data(){……}**

☆ ……其他选项

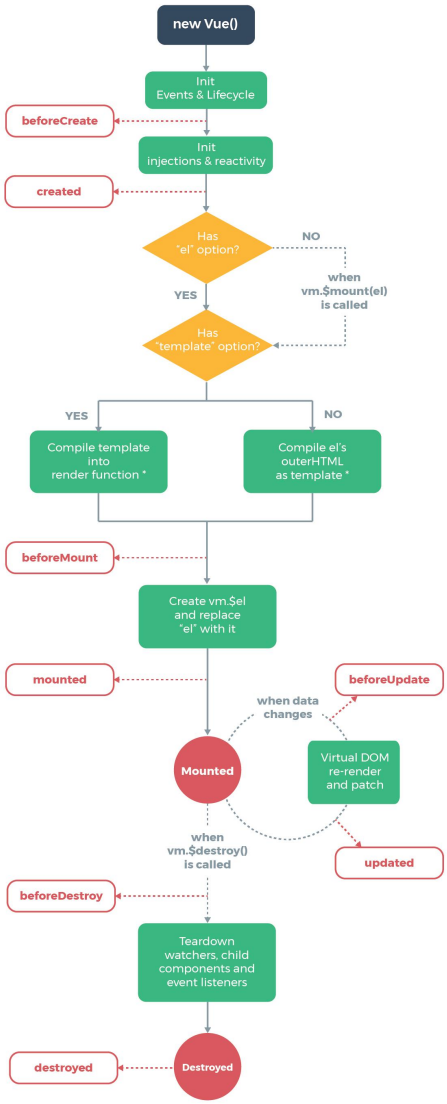
生命周期

☆ Vue 实例在初始化时需要经历一系列过程：编译模板、渲染虚拟 DOM 树、将实例挂载到 DOM 上、设置数据监听和数据绑定等

☆ 钩子函数：

☆ 在上述这些过程中允许开发者注入自己代码的地方

☆ 例子：



-

响应式：

☆ Vue.js 的核心包括一套“响应式系统”

- “响应式”，是指当数据改变后，Vue 会通知到使用该数据的代码。例如，视图渲染中使用了数据，数据改变后，视图也会自动更新。

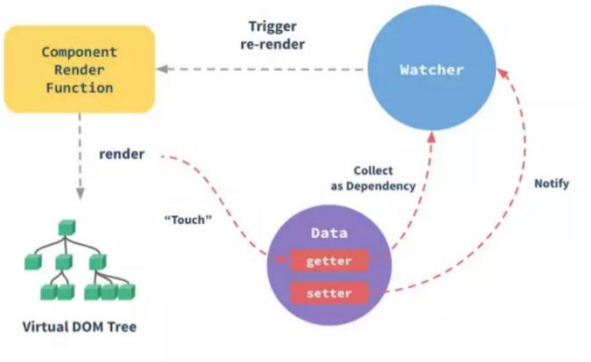
☆ Vue 的响应式，核心机制是 观察者模式

- 数据是被观察的一方，发生改变时，通知所有的观察者，这样观察者可以做出响应
  - ➢ 我们把依赖数据的观察者称为 watcher，那么这种关系可以表示为 data ->watcher

数据响应式原理

☆ 衍生数据和元数据之间的响应——数据链

☆ 视图和数据之间的绑定



● Vue 响应式原理

模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

☆ 所有 Vue.js 的模板都是合法的 HTML，

☆ 所以能被遵循规范的浏览器和 HTML 解析器解析。

插值——文本

☆ **文本插值是最常见的数据绑定**

☆ **格式：**

- “Mustache”语法 (双大括号)
- {{…}}
- eg:
- Message: {{ msg }}
- 说明：
  - Mustache 标签将会被替代为对应数据对象上 msg property 的值。无论何时，绑定的数据对象上 msg property 发生了改变，插值处的内容都会更新
  - MustacheMustache 语法不能作用在 HTML 的标签属性中
  - 语法中只能包含单个表达式进行解析

插值——HTNL

☆ **HTML 插值：**

- 使用 v-html 指令用于输出 html 代码
- 它可以动态渲染 DOM 节点
- 常用于处理开发者无可预支和难以控制的 DOM 结构

☆ **格式：**

- 在合法的 HTML 标签中捆绑
- eg:
- <dive id=“app” v-html=“blog”>

指令

☆ **什么是指令：**

- 指令 (Directives)
- 是带有 v- 前缀的特殊 attribute

- 指令 attribute 的值预期是**单个** JavaScript 表达式

☆ **指令的作用：**

- 指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM
- 从 2.6.0 开始，可以用方括号括起来的 JavaScript 表达式作为一个指令的参数

☆ **常见指令：**

- v-bind
- v-on
- v-model
- v-show
- v-html
- …

☆ **指令修饰符：**

- 修饰符 (modifier)
- 以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定

❀ 指令绑定

属性绑定

☆ **如何绑定属性：——Mustache 语法不能作用在 HTML attribute 上**

- 解决方法：
  - **DOM 节点的属性基本都可以使用指令 v-bind 进行绑定**

☆ **类名和样式绑定：**

- class 与 style 是 HTML 元素的属性，用于设置元素的样式，我们可以用 v-bind 来设置样式属性
  - Vue 做了增强:
    - 表达式结果的类型除了字符串之外，还可以是对象或数组

绑定 HTML Class：

☆ 对象语法：

☆ 我们可以传给 v-bind:class 一个对象，以动态地切换 class

```
data: {
  isActive: true,
  hasError: false
}
```

☆

```
<div
  class="static"
  v-bind:class="{ active: isActive, 'text-danger': hasError }"
></div>
```

☆

☆ 数组语法：

- 可以将多个样式对象应用到同一个元素上
  - eg：
  - data: { activeClass: 'active', errorClass: 'text-danger' }
- 渲染为：
  -

绑定样式 CSS：

☆ 语法：

- **v-bind:style——v-bind:style 的对象语法十分直观——看着非常像 CSS，但其实是一个 JavaScript 对象。CSS property 名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名**
  - CSS property 名可以这样命名：
    - 驼峰式 (camelCase) 或
    - 短横线分隔 (kebab-case，记得用引号括起来)
    - 同样，也可以使用数组语法方式绑定

条件语句

v-if 是一个指令

☆ 功能：用于条件性的渲染一块内容

- 故此，使用的时候，一般需要添加到一个元素上
- 可以为元素增加 **key** 属性
- Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染
  - 这样就导致某些元素，如<input>在切换之后仍然保留原有内容
  - key 的绑定可以让元素重新更新

```
<div id="app">
    <p v-if="seen">现在你看到我了</p>
    <template v-if="ok">
      <h1>菜鸟教程</h1>
      <p>学的不仅是技术，更是梦想！</p>
      <p>哈哈哈，打字辛苦啊！！！</p>
    </template>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    seen: true,
    ok: true
  }
})
</script>
```

☆

v-else

☆ **语法：**

- 与其他语言类似，必须与 v-if 或 v-else-if 配合使用

v-show

☆ **v-show 也是根据条件来展示元素**

☆ **区别：**

- v-show 的元素始终会被渲染并保存在 DOM 中
  - v-if 是惰性的
  - v-show 渲染开销高 VS v-if 切换开销高
- v-show 只是简单地切换元素的 CSS property display
- v-show 不支持<template>元素

循环语句

v-for

☆ **语法：**

- v-for 指令需要以 **site in sites** 形式的特殊语法:
- sites 是源数据数组
- site 是数组元素迭代的别名
- v-for 可以绑定数据到数组来渲染一个列表
- eg:

```
<div id="app">
  <ol>
    <li v-for="site in sites">
      {{ site.name }}
    </li>
  </ol>
</div>
<script>
new Vue({
  el: '#app',
  data: {
    sites: [
      { name: 'Runoob' },
      { name: 'Google' },
      { name: 'Taobao' }
    ]
  }
})
```

☆ **维护：**

- 当 Vue 正在更新使用 v-for 渲染的元素列表时，它默认使用“就地更新”的策略:
- 如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染
- 为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 **key** attribute
- eg：
  - <div v-for="item in items" v-bind:**key**="item.id"> <!-- 内容 -->
  - 建议尽可能在使用 v-for 时提供 key attribute
  - 除非遍历输出的 DOM 内容非常简单，或者是刻意依赖默认行为以获取性能上的提升

第三篇 事件&表单

★ 计划课时：

6 学时

★ 教学目的：

1. 事件处理器
2. 表单

★ 教学重点、难点：

① 如何表达事件
② 捆绑监听的命令如何增添
③ 表单各形式如何灵活运用

★ 教学内容：

❀ 本章主要内容

- 选项
- 事件
- 表单

❀ Vue 选项

- 常用选项：
  - ☆ el
  - ☆ data
  - ☆ methods
  - ☆ props
  - ☆ computed
  - ☆ watch
  - ☆ components
  - ☆ directives
  - ☆ filters

☆ …

- **el 选项**
  - ☆ Element
  - ☆ 可用于指定 Vue 实例的挂载目标
    - 属性值仅限于：
      - CSS 选择器
      - DOM 节点对象
  - ☆ 此外，Vue 也允许使用$mount 方法挂载实例
- **data 选项**
  - ☆ Vue 会递归地将 data 选项中的数据加入响应式系统
    - 前提：——这些数据应该在声明时即存在的
  - ☆ 一般可以接受两种类型：
    - 对象
    - 函数
  - ☆ **主要区别**
    - 对象方式，会让所有 Vue 实例都共享同一个内存
    - 组件——必须使用函数形式
- **computed 选项　　P102**
  - ☆ 可能、可能与 method 有点类似
    - 效果上两个都是一样的
  - ☆ 区别：
    - computed ：计算属性是基于它们的响应式依赖进行缓存的
    - 只在相关响应式依赖发生改变时它们才会重新求值
    - methods ：在重新渲染的时候，函数总会重新调用执行
  - ☆ 计算属性只有 getter，因此不能直接修改计算属性
    - 可使用 setter
  - ☆ 可以对同一数据设置多个 computed 属性
- **watch 选项　　P114**
  - ☆ Vue 提供了一种更通用的方式来观察和响应 Vue 实例上的数据变动：侦听属性
    - 当需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的
    - 大多数情况下，选择 computed
  - ☆ 明显优势：
    - 可以有中间过渡缓冲的一个状态
      - watch 更注重于处理数据变化时的业务——非常适合异步修改数据
      - computed 更注重于衍生数据

❀ 事件

- **监听事件**
  - ☆ **语法：**
    - 使用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码，一般：
      - 可以是 JavaScript 语句
      - 可以是函数方法

- Eg:

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

➢

☆ **可监听的 DOM 事件：**

- click
- keyup/down
- load
- submit
- ……

☆ **Vue.js 为 v-on 提供了事件修饰符**

- 修饰符是由点开头的指令后缀来表示的
- 常见修饰符：
  - .stop
  - .prevent
  - .capture
  - .self
  - .once
  - .passive
- 语法：

➢

○ **事件处理方法**

☆ **当监听到某元素上的 DOM 事件之后，通常，会采取调用某处理方法的方式来响应事件处理**

- 语法：
  - ➢ 这个事件处理方法一般紧跟在 v-on 的=之后
- Eg:
  - ➢ https://www.runoob.com/try/try.php?filename=vue2-v-on2

☆ 除了直接绑定到一个方法，也可以在内联 JavaScript 语句中调用方法

○ **实例分析**

表单

- **基本用法**
  - ☆ **主要双向绑定的表单：**
    - 文本
    - 复选框
    - 按钮
- **表单双向捆绑**
  - ☆ **语法：**
  - ☆ v-model
    - 使用 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
    - 它负责监听用户的输入事件以更新数据
  - ☆ 注意点：
    - v-model 会忽略所有表单元素的 value、checked、selected attribute 的初始值而总是将 Vue 实例的数据作为数据来源
    - 应该通过 JavaScript 在组件的 data 选项中声明初始值
  - ☆ **v-model 对应元素的事件：**
    - text 和 textarea 元素
      - ➢ 使用 value property 和 input 事件；
    - checkbox 和 radio
      - ➢ 使用 checked property 和 change 事件；
    - select 字段
      - ➢ 将 value 作为 prop 并将 change 作为事件。
  - ☆ **文本**
  - ☆ **复选框**
  - ☆ **按钮**
  - ☆ **选择框**
- **值绑定**
  - ☆ **通常情况下：**
    - v-model 绑定的值都是静态的
  - ☆ **动态绑定**
    - 将表单值绑定到 Vue 实例的一个动态 property 上
    - 语法：
      - ➢ v-bind
  - ☆
-

第四篇 组件

★ 计划课时：

6 学时

★ 教学目的：

1. 组件

★ 教学重点、难点：

① 认识什么是组件
② prop 属
③ 插槽
④ 动态组件

★ 教学内容：

❀ 本章主要内容

- 组 件
- prop 属性
- 插 槽
- 动态组件

❀ 组 件

- 组件
  - ☆ Component
  - ☆ 组件可以理解为预定义好行为的 ViewModel 类
  - ☆ 组件是可复用的 Vue 实例，且带有一个名字
  - ☆ 可在一个通过 new Vue 创建的 Vue 根实例中，把组件作为自定义元素来使用
  - ☆ 组件可以扩展 HTML 元素，封装可重用的代码
  - ☆ 强大的作用：
    - ● 组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型

的应用的界面都可以抽象为一个组件树

- 可以预定义的选项：
  - ☆ template 模板：必须是一个 HTML 元素
  - ☆ data 初始数据：必须是一个函数
  - ☆ pros 接受的外部参数：
  - ☆ methods 方法：
  - ☆ lifecycle hooks 生命周期钩子函数
- **第一个组件实例：**

```
// 定义一个名为 button-counter 的新组件
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">You clicked me {{ count }} times.</button>'
})
```

- ☆
- data 选项
  - ☆ 一个组件的 data 选项必须是一个函数
  - ☆ 每个实例可以维护一份被返回对象的独立的拷贝
  - ☆ 使用：
    - ● 为了能在模板中使用，这些组件必须先注册以便 Vue 能够识别
- **组件的注册&使用**
  - ☆ **注册类型**
    - ● 定义：
    - ● 给组件一个名字
    - ● 定义组件名：
      - ➢ **使用 kebab-case：**短横线分隔命名，必须在引用这个自定义元素时使用 kebab-case
      - ➢ **使用 PascalCase ：**首字母大写命名，在引用这个自定义元素时两种命名法都可以使用
      - ➢ !直接在 DOM (即非字符串的模板) 中使用时只有 kebab-case 是有效的
  - ☆ 注册方式：2 种
    - ● **全局注册**:
      - ➢ 全局注册的组件可以用在其被注册之后的任何 (通过 new Vue) 新创建的 Vue 根实例，也包括其组件树中的所有子组件的模板中
    - ● **局部注册:**
      - ➢ 局部注册的组件在其子组件中不可用
  - ☆ **全局注册：**
    - ● 语法：
    - ● 可以在一个 Vue 实例中注册多个组件
    - ● 组件注册完毕之后，可以相互使用
  - ☆ **局部注册：**
    - ● 语法:
    - ● 以通过一个普通的 JavaScript 对象来定义组件
    - ● 要定义之后，在被 Vue 实例 component 选项包含

☆ **组件的使用：**

- 语法：
  - 在需要使用到组件的地方，使用 HTML 尖括号<>括起组件名，即可使用该组件
  - <componentName>…</componentName>　仍然遵循 HTML 的成对出现匹配法则
  - 可以相互嵌套使用组件——父组件 VS 子组件
  - 组件实例的作用域是**孤立**的！
- 实例：

prop 属性

数据传递

☆ **Vue.js 组件之间有三种数据传递方式，通常父组件的模板中包含子组件，父组件要正向地向子组件传递数据以及参数：**

- props：
  - 父子组件之间正向传递数据的过程是通过 props 实现的
- 组件通信：
  - 子组件可以使用 **this.$parent** 访问其父组件，父组件也有数组 **this.$children** 包含其所有子元素
- slot：
  - 内容分发

☆ **三种数据传递方式**

- props：
- 组件通信：
  - 每个 Vue 实例都是一个事件触发器：
    - **$on()** 监听事件
    - **$emit()** 把事件沿着作用域链向上派送
    - **$dispatch()** 派发事件，事件沿着父链冒泡
    - **$broadcast()** 广播事件，事件向下传导给所有后代
- slot：

Vue 的属性选项 props

☆ props：

- 这是 Vue 为组件开发提供的属性选项
- 用途：
  - 它为组件注册动态特性，以处理业务间差异
  - 通过 props 选项，组件可以接受多态的数据
  - props 它是组件数据的一个字段，期望从父组件传下来的数据

☆ 语法：

- ——其值传递方式是：按引用传递
  - 以字符串数组形式列出的 prop：
    -

```
props: ['title', 'likes', 'isPublished', 'commentIds', 'author']
```

- 以对象形式列出 prop:

```
props: {
  title: String,
  likes: Number,
  isPublished: Boolean,
  commentIds: Array,
  author: Object,
  callback: Function,
  contactsPromise: Promise // or any other constructor
}
```

☆ **静态**值传递：

- 语法：直接将字符串赋值

```
<blog-post title="My journey with Vue"></blog-post>
```

☆ 可以动态绑定：

- 可以类似 v-bind 形式，**动态**为 props 绑定到父组件的数据——这样，每次父组件数据变化，则会自动传到给子组件
- .sync：双向绑定，会把子组件的数据属性同步回父组件的属性
- .once：单次绑定，建立之后不会同步之后的变化

```
<!-- 动态赋予一个变量的值 -->
<blog-post v-bind:title="post.title"></blog-post>

<!-- 动态赋予一个复杂表达式的值 -->
<blog-post
  v-bind:title="post.title + ' by ' + post.author.name"
></blog-post>
```

☆ **props 的单向数据流：**

- 所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：
- 父级 prop 的更新会向下流动到子组件中，但是反过来则不行

☆

props 事件

☆ 为什么需要事件？

- 父组件是使用 props 传递数据给子组件，是单向数据流！
- 但如果子组件要把数据传递回去，就需要使用自定义事件！

☆ 子组件-->父组件

- 是什么样的事件？
- 需要子组件自己触发！

☆ 子组件触发事件：

- 语法：
  - 使用 $on(eventName)监听事件
  - 使用 $emit (eventName) 触发事件
- 说明：

- 父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件
- 例子：

☆

Vue 的其他属性选项

插　　槽

slot

☆ 内容分发

☆ slot 是一个内置的自定义元素指令

☆ 语法：

- Vue 实现了一套内容分发的 API，将<slot>元素作为承载分发内容的出口
- 插槽内可以包含任何模板代码，包括 HTML

☆ 实例：

- !!!
- 在组件的的 template 中没有包含一个 <slot> 元素，则该组件起始标签和结束标签之间的任何内容都会被抛弃

作用域：

☆ 在使用组件时，组件<>对中间出现任何东西都属于组件的作用域

☆ 规则：

- 父级模板里的所有内容都是在父级作用域中编译的
- 子模板里的所有内容都是在子作用域中编译的

☆ 实例：

v-slot

动态组件

应用场景：

☆ 在某些场景，需要动态切换页面部分区域的视图

is：

☆ component 接受名为 is 的属性

- is 的值应该为父组件中注册过的组件的名称

☆ 语法：

```
<div id="app">
  <ul class="tabs">
    <li class="per-tab" @click="toggleView('Home')">Home</li><!--
    --><li class="per-tab" @click="toggleView('About')">About</li>
  </ul>
  <div class="tab-content">
    <component :is="view"></component> <!-- view为变量 -->
  </div>
</div>
```

第五篇 路由

★ 计划课时：

4 学时

★ 教学目的：

1. 路由

★ 教学重点、难点：

① 什么是前端路由
② 如何配置
③ 利用工具构建项目

★ 教学内容：

❀ 本章主要内容

- 构建项目
- 前端路由

❀ 构建项目

- 参考前面环境搭建
- 主要工具 Vue CLi
- 一个完整的前端开发环境应该具备：
  - ☆ 预编译模板
  - ☆ 注入依赖
  - ☆ 合并压缩资源
  - ☆ 分离开发和生产环境
  - ☆ 提供一个模拟的服务端环境等
  - ☆ ----------
  - ☆ **index.html**
  - ☆ **main.js**

- ☆ **app.vue**
- ☆ **index.js**
- ☆ **HelloWorld.vue**
- ☆ ----------
- ☆

前端路由

- **什么是前端路由：**
  - ☆ 路由：
    - ● 路由的概念来源于服务端，在服务端中路由描述的是 URL 与处理函数之间的映射关系
  - ☆ 前端路由：
    - ● 在 Web 前端单页应用 SPA(Single Page Application)中，路由描述的是 URL 与 UI 之间的映射关系，这种映射是单向的，即 URL 变化引起 UI 更新（无需刷新页面）
- **前端路由的实现，需要解决两个核心问题？**
  - ☆ 如何改变 URL 却不引起页面刷新
  - ☆ 如何检测 URL 变化了
- **如何实现？**
  - ☆ hash
  - ☆ history
- **hash 实现前端路由：**
  - ☆ hash 是 URL 中 hash (#)及后面的那部分，常用作锚点在页面内进行导航，改变 URL 中的 hash 部分不会引起页面刷新
  - ☆ 通过 hashchange 事件监听 URL 的变化，改变 URL 的方式只有这几种：——几种情况改变 URL 都会触发 hashchange 事件
    - ● 通过浏览器前进后退改变 URL
    - ● 通过标签改变 URL
    - ● 通过 window.location 改变 URL
- **history 实现前端路由：**
  - ☆ history 提供了 pushState 和 replaceState 两个方法，这两个方法改变 URL 的 path 部分不会引起页面刷新
  - ☆ history 提供类似 hashchange 事件的 popstate 事件
    - ● 通过浏览器前进后退改变 URL 时会触发 popstate 事件
- **在 Vue 中的前端路由是。。。**
  - ☆ Vue 的 vue-router 提供了前端路由在 Vue 中的实现
- **vue-router 是如何定义的？**
  - ☆ 例子：
    - ● import VueRouter from 'vue-router’
    - ● Vue.use(VueRouter)
    - ● const router = new VueRouter({
    - ● mode: 'history',
    - ● routes: [...]})

- new Vue({
- router
- ...})

☆ **设置和使用：**

- Vue Router 是 Vue.js 提供的路由管理器，它与 Vue.js 的核心深度集成，主要功能有：
- 嵌套的路由表和视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- ……

☆ 如何使用？

- **导入 Vue Router**
  - 语句：
  - **<script src="https://cdn.staticfile.org/vue-router/2.7.0/vue-router.min.js"></script>**
  - 或：
  - **import Router from 'vue-router'**
  -
- **Vue 实例引用路由管理器 Vue Router**
  - 语句：
  - **Vue.use(Router)**
  -
- **实例化路由对象**
  - 语句
  - **const router = new VueRouter({**
  - **routes //（缩写）相当于 routes: routes**
  - **})**
  -
- **在其中添加需要的路由规则**
  - 语句：
  - **export default new Router({**
  - **routes: [**
  - **{**
  - **path: '/',**
  - **name: 'Index',**
  - **component:Com**
  - **},**
  - **……**
  - **] })**
  -
- **使用**
  - 语法：
  - <router-link></router-link>

- <router-view></router-view>

- ☆ **大重要命名：**
  - **<router-link>**
    - 命名路由，是一个组件，该组件用于设置一个导航链接，切换不同 HTML 内容
    - **to** :属性为目标地址， 即要显示的内容
  - **<router-view>**
    - 命名视图，写在组件想要渲染的地方，当读写到该语句，则渲染组件
    - 一个视图，使用一个组件渲染
    - <router-view> 是最顶层的出口，渲染最高级路由匹配到的组件
- ☆

第六篇 Vuex

★ 计划课时：

学时

★ 教学目的：

1.
2.

★ 教学重点、难点：

①
②
③
④

★ 教学内容：