**Vue技术栈(全家桶)**

尚硅谷前端研究院

# 第1章：Vue核心

## Vue的基本认识

### 官网

1. 英文官网: <https://vuejs.org/>
2. 中文官网: <https://cn.vuejs.org/>

### 介绍描述

1. 动态构建用户界面的**渐进式**JavaScript库
2. 作者: 尤雨溪(一位华裔前Google工程师)

### Vue的特点

1. 遵循MVVM模式
2. 编码简洁, 体积小, 运行效率高, 适合移动/PC端开发
3. 它本身只关注UI, 可以轻松引入vue插件或其它第三库开发项目

### 与其它前端JS框架的关联

1. 借鉴Angular的**结构模板**和**数据绑定**技术
2. 借鉴React的**组件化**和**虚拟DOM**技术

### Vue扩展插件

1. vue-cli: vue脚手架
2. vue-resource、axios: ajax请求
3. vue-router: 路由
4. vuex: 状态管理
5. mint-ui: 基于vue的UI组件库(移动端)
6. element-ui: 基于vue的UI组件库(PC端)

……

## 初识Vue

### 效果



## 模板语法

### 效果



### 模板的理解

html中包含了一些JS语法代码，语法分为两种，分别为：

* 1. 插值语法（双大括号表达式）
  2. 指令（以v-开头的标签属性）

### 插值语法

1. 功能: 用于解析标签体内容
2. 语法: {{xxx}} ，xxxx会作为js表达式解析

### 指令

1. 功能: 解析标签属性、解析标签体内容、绑定事件的回调
2. 举例：v-bind:href = 'xxxx' ，xxxx会作为js表达式被解析
3. 说明：Vue中有有很多的指令，此处只是用v-bind举个例子

## 数据绑定

### 效果



### 单向数据绑定

1. 语法：v-bind:href ="xxx" 或简写为 :href

2. 特点：数据只能从data流向页面

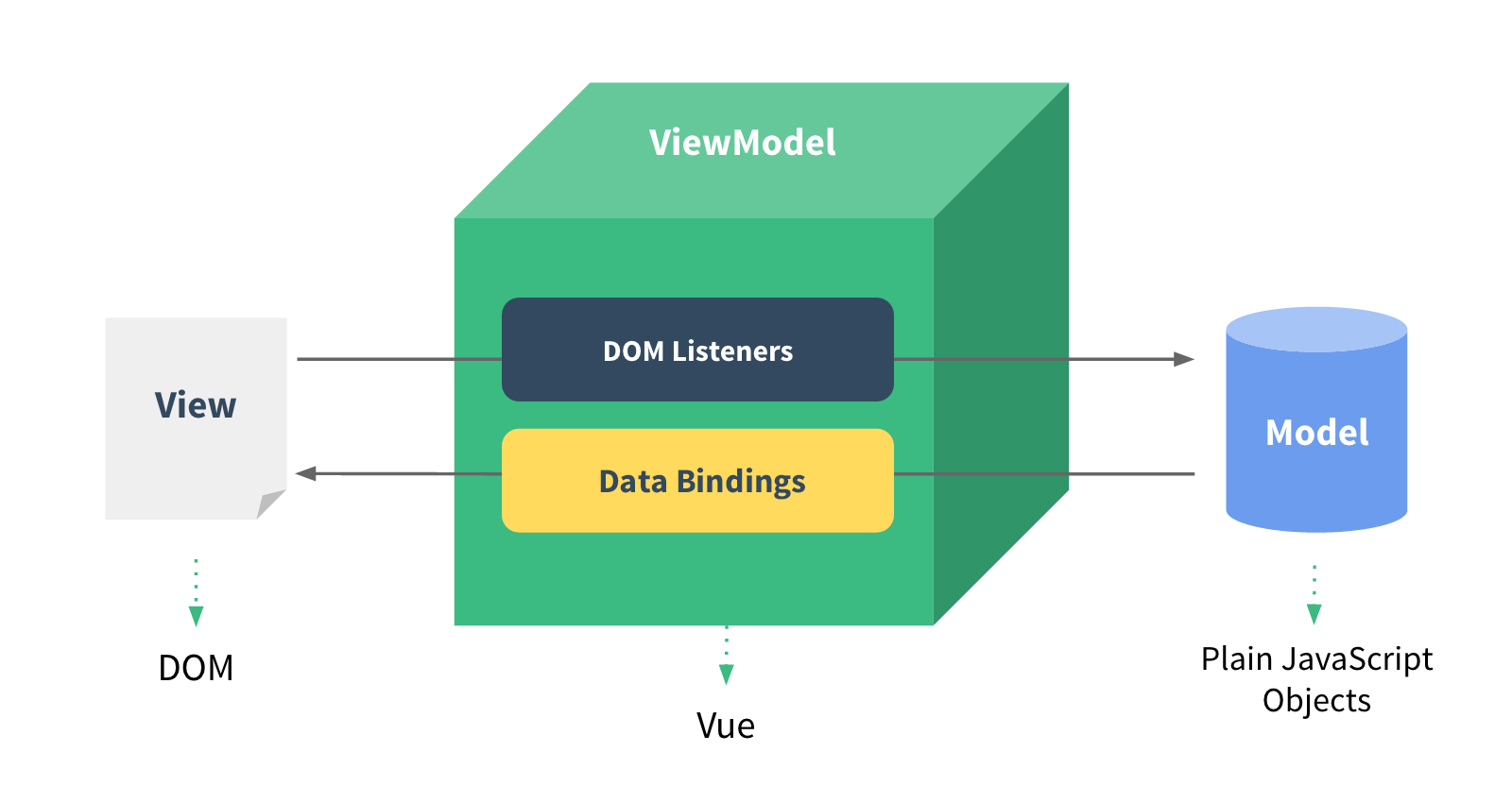
### 双向数据绑定

1. 语法：v-mode:value="xxx" 或简写为 v-model="xxx"

2. 特点：数据不仅能从data流向页面，还能从页面流向data

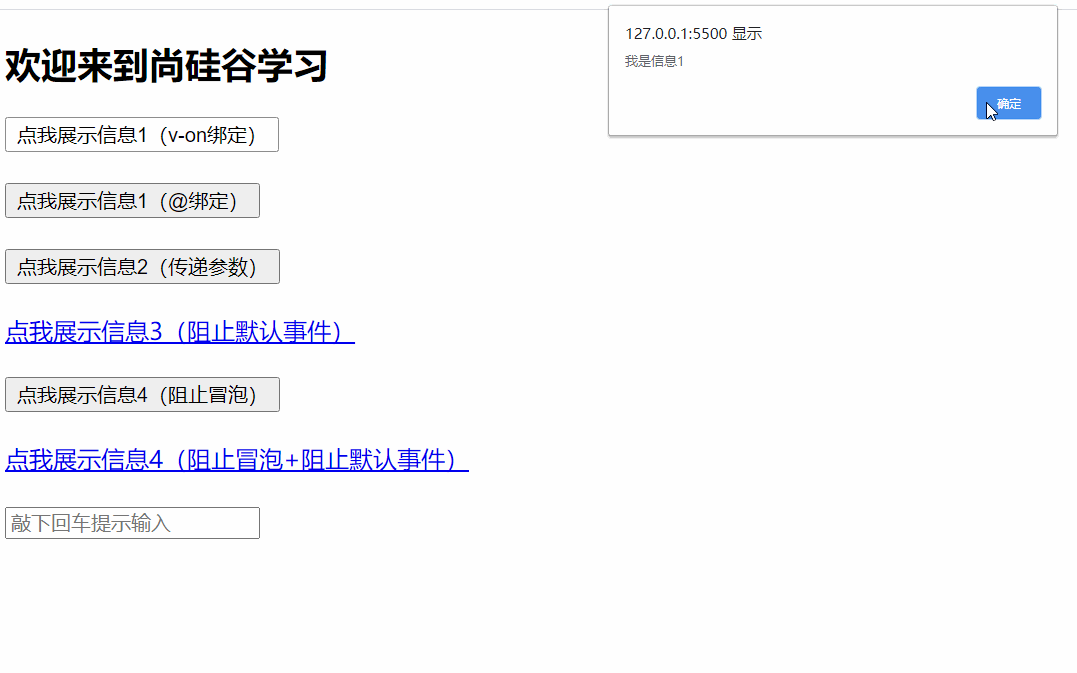
## MVVM模型

1. M：模型(Model) ：对应data中的数据
2. V：视图(View) ：模板代码
3. VM：视图模型(ViewModel) ： Vue实例对象



## 事件处理

### 效果



### 绑定监听:

1. v-on:xxx="fun"
2. @xxx="fun"
3. @xxx="fun(参数)"
4. 默认事件形参: event
5. 隐含属性对象: $event

### 事件修饰符

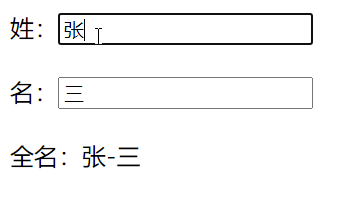
1. .prevent : 阻止事件的默认行为 event.preventDefault()
2. .stop : 停止事件冒泡 event.stopPropagation()

### 按键修饰符

1. keycode : 操作的是某个keycode值的键
2. .keyName : 操作的某个按键名的键(少部分)

## 计算属性与监视

### 效果



### 计算属性-computed

1. 要显示的数据不存在，要通过计算得来。
2. 在computed对象中定义计算属性。
3. 在页面中使用{{方法名}}来显示计算的结果。

### 监视属性-watch

1. 通过通过vm对象的$watch()或watch配置来监视指定的属性
2. 当属性变化时, 回调函数自动调用, 在函数内部进行计算

## class与style绑定

### 理解

1. 在应用界面中, 某个(些)元素的样式是变化的
2. class/style绑定就是专门用来实现动态样式效果的技术

### class绑定

1. :class='xxx'
2. 表达式是字符串: 'classA'
3. 表达式是对象: {classA:isA, classB: isB}
4. 表达式是数组: ['classA', 'classB']

### style绑定

1. :style="{ color: activeColor, fontSize: fontSize + 'px' }"
2. 其中activeColor/fontSize是data属性

## 条件渲染

### 效果



### 条件渲染指令

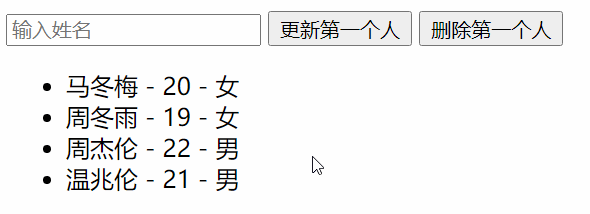
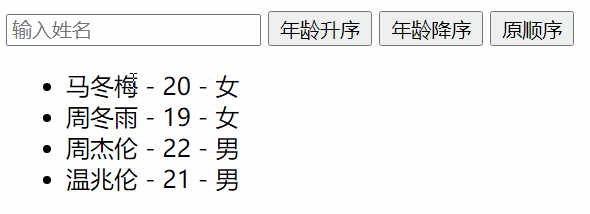
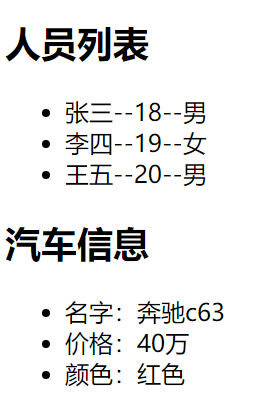
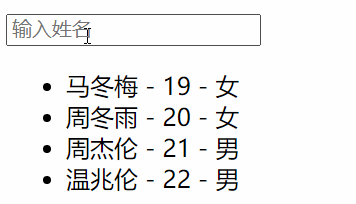
1. v-if与v-else
2. v-show

### 比较v-if与v-show

1. 如果需要频繁切换 v-show 较好
2. 当条件不成立时, v-if的所有子节点不会解析(项目中使用)

## 列表渲染

### 效果



### 列表显示指令

遍历数组: v-for / index

遍历对象: v-for / key

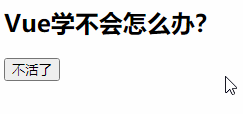
## 收集表单数据

### 效果



## Vue实例生命周期

### 效果



### 生命周期流程图

**

### vue生命周期分析

1. 初始化显示

\* beforeCreate()

\* created()

\* beforeMount()

\* mounted()

1. 更新状态: this.xxx = value

\* beforeUpdate()

\* updated()

1. 销毁vue实例: vm.$destory()

\* beforeDestory()

\* destoryed()

### 常用的生命周期方法

1. mounted(): 发送ajax请求, 启动定时器等异步任务
2. beforeDestory(): 做收尾工作, 如: 清除定时器

## 过渡&动画

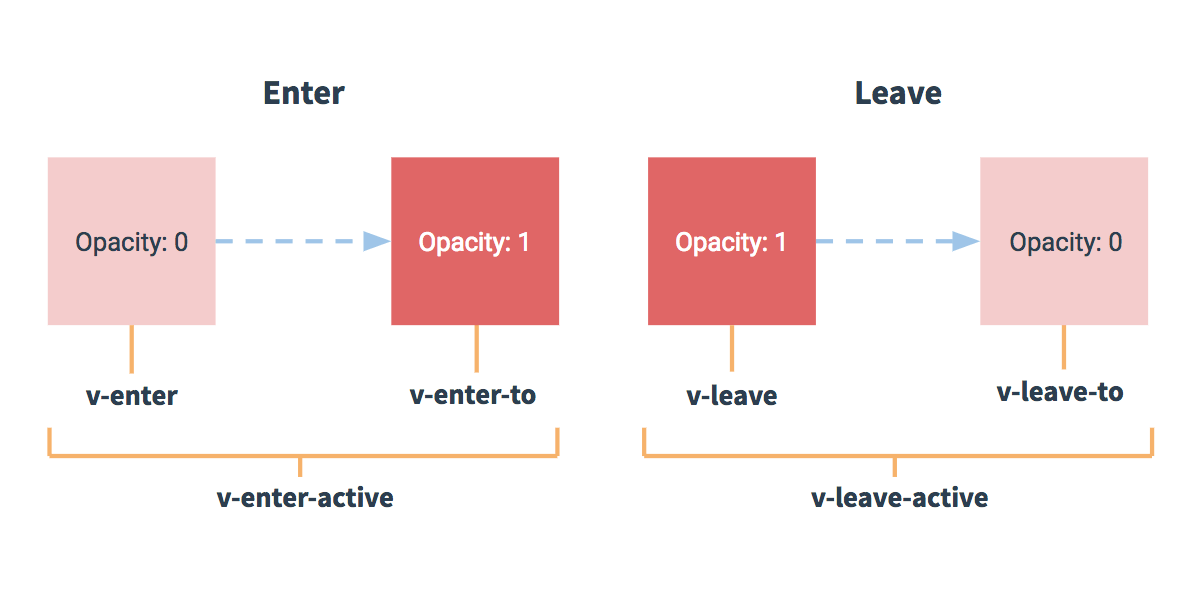
### 效果





### vue动画的理解

1. 操作css的trasition或animation
2. vue会给目标元素添加/移除特定的class
3. 过渡的相关类名：
4. xxx-enter-active: 指定显示的transition
5. xxx-leave-active: 指定隐藏的transition
6. xxx-enter/xxx-leave-to: 指定隐藏时的样式

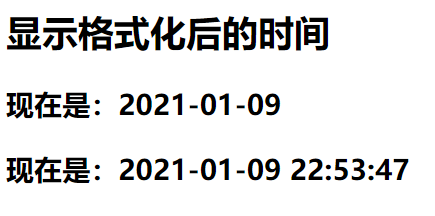


### 基本过渡动画的编码

1. 在目标元素外包裹<transition name="xxx">
2. 定义class样式
   1. 指定过渡样式: transition
   2. 指定隐藏时的样式: opacity/其它

## 过滤器

### 效果



### 理解过滤器

1. 功能: 对要显示的数据进行特定格式化后再显示
2. 注意: 并没有改变原本的数据, 是产生新的对应的数据

## 内置指令与自定义指令

### 常用内置指令

1. v-text : 更新元素的 textContent
2. v-html : 更新元素的 innerHTML
3. v-if : 如果为true, 当前标签才会输出到页面
4. v-else: 如果为false, 当前标签才会输出到页面
5. v-show : 通过控制display样式来控制显示/隐藏
6. v-for : 遍历数组/对象
7. v-on : 绑定事件监听, 一般简写为@
8. v-bind : 强制绑定解析表达式, 可以省略v-bind
9. v-model : 双向数据绑定
10. ref : 指定唯一标识, vue对象通过$els属性访问这个元素对象
11. v-cloak : 防止闪现, 与css配合: [v-cloak] { display: none }

### 自定义指令

1. **注册全局指令**

Vue.directive('my-directive', function(el, binding){

el.innerHTML = binding.value.toupperCase()

})

1. **注册局部指令**

directives : {

'my-directive' : {

bind (el, binding) {

el.innerHTML = binding.value.toupperCase()

}

}

}

1. 使用指令

v-my-directive='xxx'

## 自定义插件

### 说明

1. Vue插件是一个包含install方法的对象
2. 通过install方法给Vue或Vue实例添加方法, 定义全局指令等

# 第2章：Vue组件化编程

## 模块与组件、模块化与组件化

### 模块

1. 理解: 向外提供特定功能的js程序, 一般就是一个js文件
2. 为什么: js代码更多更复杂
3. 作用: 复用js, 简化js的编写, 提高js运行效率

### 组件

1. 理解: 用来实现特定(局部)界面功能效果的代码集合(html/css/js/image…..)
2. 为什么: 一个界面的功能很复杂
3. 作用: 复用编码, 简化项目编码, 提高运行效率

### 模块化

当应用的js都以模块来编写的, 这个应用就是一个模块化的应用

### 组件化

当应用是以多组件的方式实现, 这个应用就是一个组件化的应用, 应用的开发方式就是组件化的

## 组件定义与使用——(非单文件)

### 存在的问题

1. 模板编写没有提示
2. 没有构建过程, 无法将ES6转换成ES5
3. 不支持组件的CSS
4. 真正开发中几乎不用

## 组件定义与使用——(单文件组件)

### 一个.vue文件的组成(3个部分)

1. **模板页面**

<template>

页面模板

</template>

1. **JS模块对象**

<script>

export default {

data() {return {}},

methods: {},

computed: {},

components: {}

}

</script>

1. **样式**

<style>

样式定义

</style>

### 基本使用

1. 引入组件
2. 映射成标签
3. 使用组件标签

# 第3章：使用Vue脚手架

## 3.1 使用脚手架创建模板项目

### 3.1.1 说明

1. vue-cli是vue官方提供的脚手架工具 command line interface
2. 最新的版本是4,
3. 3.x版本与4.x版本变化不大, 但3.x相对于2.x的版本变化特别大
4. 在线文档: <https://cli.vuejs.org/zh/>

### 3.1.2 创建vue项目

1. 创建脚手架4/3的vue项目, 并运行

npm install -g @vue/cli

vue create vue-demo

npm run serve

1. 创建脚手架2的vue项目

npm install -g @vue/cli-init

vue init webpack vue-demo

npm run dev

1. 访问: http://localhost:8080/

### 3.1.3 模板项目的结构

vue-cli2脚手架项目结构

gshop

|-- build : webpack相关的配置文件夹(基本不需要修改)

|-- config: webpack相关的配置文件夹(基本不需要修改)

|-- index.js: 指定的后台服务的端口号和静态资源文件夹

|-- node\_modules

|-- src : 源码文件夹

|-- main.js: 应用入口js

|-- static: 静态资源文件夹

|-- .babelrc: babel的配置文件

|-- .editorconfig: 通过编辑器的编码/格式进行一定的配置

|-- .eslintignore: eslint检查忽略的配置

|-- .eslintrc.js: eslint检查的配置

|-- .gitignore: git版本管制忽略的配置

|-- index.html: 主页面文件

|-- package.json: 应用包配置文件

|-- README.md: 应用描述说明的readme文件

vue-cli3脚手架项目结构

gshop

|-- node\_modules

|-- public

|-- index.html: 主页面文件

|-- src

|-- main.js: 应用入口js

|-- babel.config.js: babel的配置文件

|-- vue.config.js: vue的配置文件，需要手动添加

|-- .gitignore: git版本管制忽略的配置

|-- package.json: 应用包配置文件

|-- README.md: 应用描述说明的readme文件

### 3.1.4 脚手架3相对于脚手架2的变化

1. webpack配置
   1. 2: 配置是暴露的, 我们可以直接在里面修改配置
   2. 3: 配置是包装隐藏了, 需要通过脚手架扩展的vue.config.js来配置
2. 运行启动命令
   1. 2: npm run dev
   2. 3: npm run serve

## 3.2 todo list案例

完成todolist案例

## 3.3 ref与props

### 3.3.1 ref

1. **作用：**用于给DOM节点打标识
2. **读取方式：**this.$refs.xxxxxx

### 3.3.2 props

1. **作用：**用于父组件给子组件传递数据
2. **读取方式一:** **只指定名称**

props: ['name', 'age', 'setName']

1. **读取方式二: 指定名称和类型**

props: {

name: String,

age: Number,

setNmae: Function

}

1. **读取方式三: 指定名称/类型/必要性/默认值**

props: {

name: {type: String, required: true, default:xxx},

}

## 3.4 Vue中的自定义事件

### 3.4.1 绑定事件监听

|  |
| --- |
| <**Header @addTodo="**addTodo**"**/>  或者  <**Header ref="header"**/>**this**.**$refs**.header.$on(**'addTodo'**, **this**.addTodo) |

### 3.4.2 触发事件

|  |
| --- |
| **this**.$emit(**'addTodo'**, todo) |

## 3.5 消息订阅与发布

### 3.5.1 理解

1. 这种方式的思想与全局事件总线很相似
2. 它包含以下操作:
   1. 订阅消息 --对应绑定事件监听
   2. 发布消息 --分发事件
   3. 取消消息订阅 --解绑事件监听
3. 需要引入一个消息订阅与发布的第三方实现库: [**PubSubJS**](https://github.com/mroderick/PubSubJS)

### 3.5.2使用PubSubJS

1. 在线文档: <https://github.com/mroderick/PubSubJS>
2. 下载: npm install -S pubsub-js
3. 相关语法
   1. import PubSub from 'pubsub-js' // 引入
   2. PubSub.subscribe(‘msgName’, functon(msgName, data){ })
   3. PubSub.publish(‘msgName’, data): 发布消息, 触发订阅的回调函数调用
   4. PubSub.unsubscribe(token): 取消消息的订阅

## 3.6 slot插槽

### 3.6.1 理解

此方式用于父组件向子组件传递带数据的标签，当一个组件有不确定的结构时, 就需要使用slot技术了，注意：插槽内容是在父组件中编译后, 再传递给子组件的。

### 3.6.2 分类

1. 默认插槽
2. 命名插槽
3. 作用域插槽

## 3.7全局事件总线

### 3.7.1 理解

1. Vue原型对象上包含事件处理的方法
   1. $on(eventName, listener): 绑定自定义事件监听
   2. $emit(eventName, data): 分发自定义事件
   3. $off(eventName): 解绑自定义事件监听
   4. $once(eventName, listener): 绑定事件监听, 但只能处理一次
2. 所有组件对象的原型对象的原型对象就是Vue的原型对象
   1. 所有组件对象都能看到Vue原型对象上的属性和方法
   2. Vue.prototype.$bus = new Vue(), 所有的组件对象都能看到$bus这个属性对象
3. 全局事件总线
   1. 包含事件处理相关方法的对象(只有一个)
   2. 所有的组件都可以得到

### 3.7.2 指定事件总线对象

|  |
| --- |
| **new** Vue({  beforeCreate () { *// 尽量早的执行挂载全局事件总线对象的操作* Vue.**prototype**.**$globalEventBus** = **this** },}).$mount(**'#root'**) |

### 3.7.3 绑定事件监听

|  |
| --- |
| **this**.**$globalEventBus**.$on(**'deleteTodo'**, **this**.deleteTodo) |

### 3.7.4 分发事件监听

|  |
| --- |
| **this**.**$globalEventBus**.$emit(**'deleteTodo'**, **this**.**index**) |

### 3.7.5 解绑事件监听

|  |
| --- |
| **this**.**$globalEventBus**.$off(**'deleteTodo'**) |

# 第4章：Vue中的ajax

## 4.1 vue项目中常用的2个ajax库

### 4.1.1 vue-resource

vue插件, 非官方库, vue1.x使用广泛，现已弃用。

### 4.1.2 axios

通用的ajax请求库, 官方推荐, vue2.x使用广泛

## 4.2 解决开发环境Ajax跨域问题

使用代理服务器

## 4.3 github users案例

接口地址: <https://api.github.com/search/users?q=xxx>

# 第5章：vuex

## 5.1 vuex理解

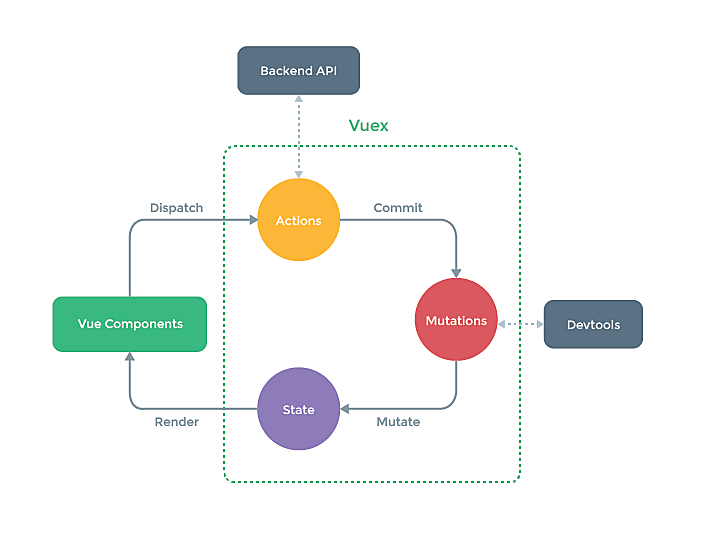
### 5.1.1 vuex是什么

1. 专门在Vue中实现集中式状态管理的一个插件，对vue应用中多个组件的共享状态进行集中式的管理(读/写)，也可以认为是一种组件间通信的方式，且适用于任意组件间通信。
2. github站点: <https://github.com/vuejs/vuex>
3. 在线文档: <https://vuex.vuejs.org/zh-cn/>

### 5.1.2 什么时候使用Vuex

1. 多个组件依赖于同一状态
2. 来自不同组件的行为需要变更同一状态
3. 多个组件要共享状态

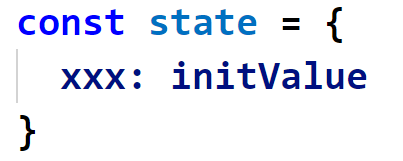
### 5.1.3 Vuex工作原理图



## 5.2 vuex核心概念和API

### 5.2.1 state

1. vuex管理的状态对象
2. 它应该是唯一的
3. 示例代码：

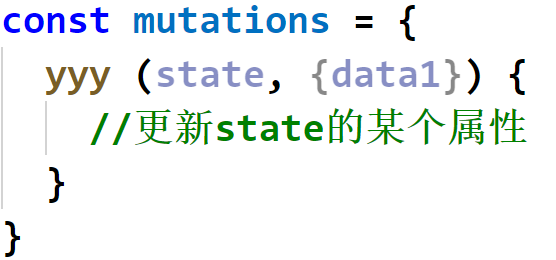


### 5.2.2 mutations

1. 值是一个对象，包含多个直接更新state的函数（又称方法）
2. 谁能调用mutations中的方法？如何调用？

在action中使用：**commit('对应的mutations方法名')** 触发

1. mutations中方法的特点：不能写异步代码、只能单纯的操作state
2. 示例代码：

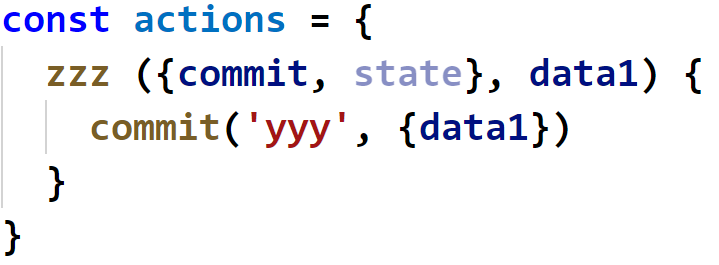


### 5.3.3 actions

1. 值为一个对象，包含多个响应用户动作的回调函数
2. 通过commit( )来触发mutation中函数的调用, 间接更新state
3. 如何触发actions中的回调？

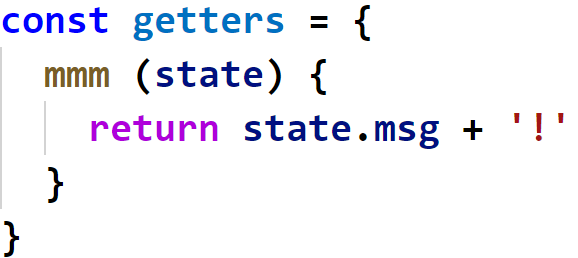
在组件中使用: **$store.dispatch('对应的action回调名')** 触发

1. 可以包含异步代码（定时器, ajax等等）
2. 示例代码：



### 5.3.4 getters

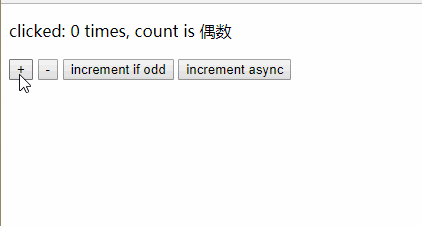
1. 值为一个对象，包含多个用于返回数据的函数
2. 如何使用？—— **$store.getters.xxx**
3. 示例代码：



### 5.3.5 modules

1. 包含多个module
2. 一个module是一个store的配置对象
3. 与一个组件（包含有共享数据）对应

### 5.3.6 计数器案例



# 第6章：vue-router

## 6.1相关理解

### 6.1.1 vue-router的理解

1. vue的一个插件库
2. 专门用来实现一个SPA应用
3. 基于vue的项目基本都会用到此库
4. 中文文档: <http://router.vuejs.org/zh-cn/>
5. 下载: npm install vue-router -S

### 6.1.2 SPA的理解

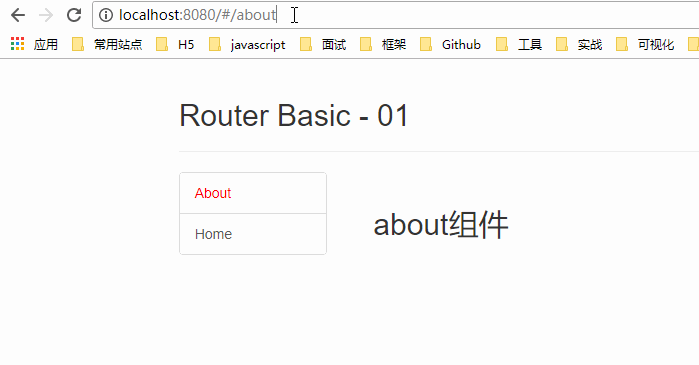
1. 单页Web应用（single page web application，SPA）。
2. 整个应用只有**一个完整的页面**。
3. 点击页面中的链接**不会刷新**页面，只会做页面的**局部更新。**
4. 数据都需要通过ajax请求获取, 并在前端异步展现。

### 6.1.3 路由的理解

1. **什么是路由?**
   1. 一个路由就是一个映射关系(key:value)
   2. key为路径, value可能是function或component
2. **路由分类**
   1. 后端路由：
      1. 理解： value是function, 用来处理客户端提交的请求。
      2. 注册路由： router.get(path, function(req, res))
      3. 工作过程：当服务器接收到一个请求时, 根据请求路径找到匹配的路由, 调用路由中的函数来处理请求, 返回响应数据
   2. 前端路由：
      1. 浏览器端路由，value是component，用于展示页面内容。
      2. 注册路由: <Route path="/test" component={Test}>
      3. 工作过程：当浏览器的path变为/test时, 当前组件就会变为Test组件

## 6.2 基本路由

### 6.2.1 效果



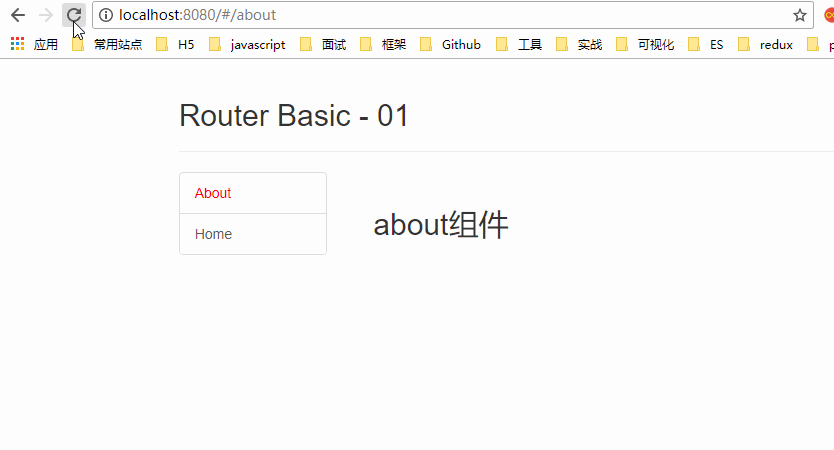
### 6.2.2 总结: 编写使用路由的3步

1. 定义路由组件
2. 注册路由
3. 使用路由

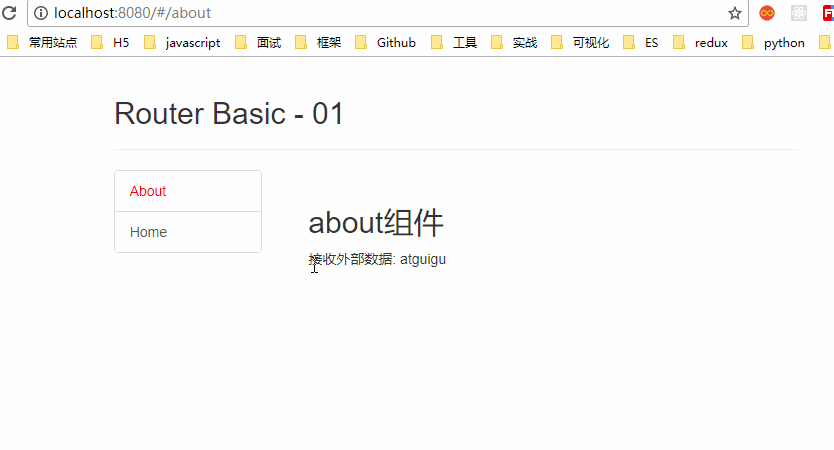
<router-link>

<router-view>

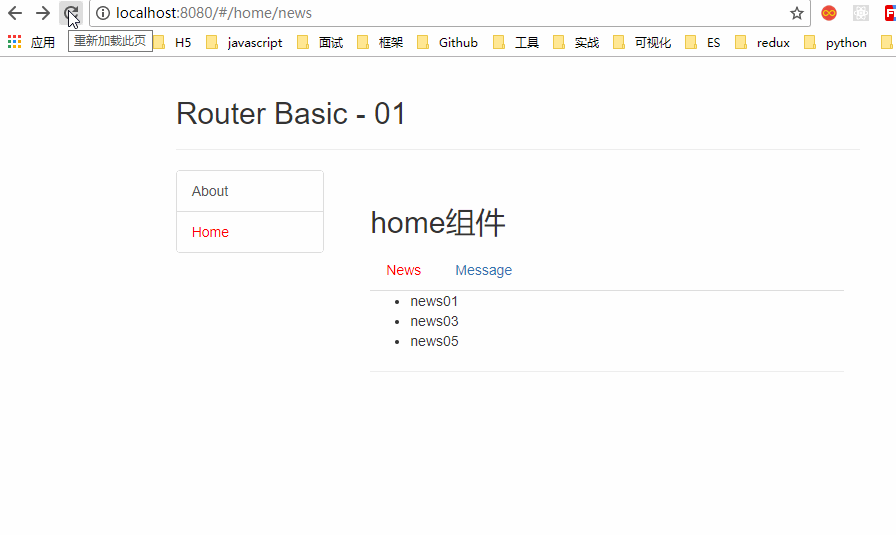
## 6.3 嵌套路由



## 6.4 路由传参



## 6.5 编程式路由导航



**相关API：**

1. this.$router.push(path): 相当于点击路由链接(可以返回到当前路由界面)
2. this.$router.replace(path): 用新路由替换当前路由(不可以返回到当前路由界面)
3. this.$router.back(): 请求(返回)上一个记录路由
4. this.$router.go(-1): 请求(返回)上一个记录路由
5. this.$router.go(1): 请求下一个记录路由

# 第7章：Vue UI组件库

## 7.1常用的vue UI组件库

### 7.1.1移动端UI组件库

1. Vant <https://youzan.github.io/vant/#/zh-CN/>
2. Cube UI <https://didi.github.io/cube-ui/#/zh-CN>
3. Mint UI <http://mint-ui.github.io/> (经常不能访问)

### 7.1.2 PC端UI组件库

1. Element UI <https://element.eleme.cn/#/zh-CN>
2. IView UI <https://www.iviewui.com/>

## 7.2使用 Element UI

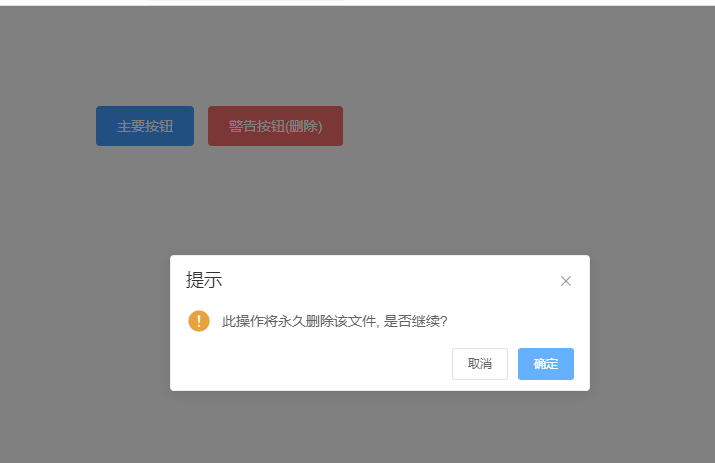
### 7.2.1 下载

|  |
| --- |
| npm i element-ui -S |

### 7.2.2 完整引入Element

|  |
| --- |
| **import** ElementUI **from 'element-ui' import 'element-ui/lib/theme-chalk/index.css'**  Vue.use(ElementUI) |

### 7.2.3 使用UI组件



### 7.2.4 实现按需打包

1. 下载

|  |
| --- |
| npm install babel-plugin-component -D |

1. 添加babel相关配置: webpack.config.js

|  |
| --- |
| **plugins**: [  [  **"component"**, *// 为babel-plugin-components配置* {  **"libraryName"**: **"element-ui"**,  **"styleLibraryName"**: **"theme-chalk"** }  ] ] *// 一旦我们需要一个另外babel插件, 需要在此配置* |