**Vue.js-Day02**

# 课程介绍

* 1. v-bind（掌握）
* 2. 其他指令（了解）
* 3. 过滤器filters（了解/掌握）
* 4. 侦听器watcher（了解/掌握）
* 5. Vue生命周期（了解/掌握）
* 6. 路由vue-router(重点掌握)

# v-bind【掌握】

## v-bind给属性绑定动态数据

语法： <标签 v-bind:属性=”表达式” ></标签>

简写： <标签 :属性=”表达式” ></标签>

## v-bind绑定style

语法： <标签 :style=”{ color: ‘red’, backgroundColor: ‘green’ }” ></标签>

## v-bind绑定class

语法：<标签 :class=”{ 类名1: flag1, 类名2: flag2 }” ></标签>

注意：flag是true，就有这个类，是false，就没有这个类

# 其他指令（了解）

## v-pre

不编译Mustache模板表达式，直接当成普通的字符串输出

## v-cloak

隐藏Mustache表达式，直到有数据后，才显示出来

## v-once

只渲染1次，后续数据改变，不会重新渲染。

# 过滤器filters(了解)

## 什么是filter

过滤器filters写法和methods一样，主要是在数据输出到页面之前，进行显示格式处理。

## 特点

**过滤器写法和computed一模一样，有一些区别：**

1. 过滤器，函数会接收一个参数，函数必须有返回值，对传入数据进行处理
2. 过滤器，无缓存机制，调用次数，取决于页面中有所多少过滤器
3. 过滤器，被作为一个特殊方法处理.

## 写法

<h1> {{ msg **| 过滤器函数名** }} </h1>

export default {

components: {}

data() {},

methods: {},

computed: {},

filters: { // 过滤器

函数名(msg) {

// 函数中，可以对msg进行各种格式处理，返回什么，页面就能输出什么

return xxx

}

}

}

# 4 侦听器Watch（了解）

## 4.1.什么是watch

侦听器watch写法和computed一样，主要是捕获指定数据的变化，进行相关的操作。

## 4.2.特点

**侦听器写法和computed一模一样，有一些区别：**

1. 侦听器，函数会接收两个参数()，内部可以根据新值进行处理，函数无需返回值，
2. 侦听器，自定捕获数据的变化，并作出响应。变化一次，执行一次。
3. 侦听器，被作为一个监听方法处理.

## 4.3.写法

export default {

components: {}

data() {},

methods: {},

computed: {},

filters: {},

watch:{

数据属性名称(newVal，oldVal) {

// 对数据进行各种处理，可以返回，页可以部返回。

return xxx

}

}

}

# 5. Vue生命周期（了解）

生命总共分为4大阶段**创建前/后**、**载入前/后**、**更新前/后**、**销毁前/后**共8个方法。

## 5.1创建前/后

创建前: :

在**beforeCreate**阶段，vue实例的挂载元素el和数据对象data都为undefined，还未初始化；

创建后 ：

在**created**阶段，vue实例的数据data有了，el还没有。

\*创建阶段：主要是处理数据（主要：对数据进行挟持，把data对象上的数据挂到了this实例上）！！

## 5.2挂载前/后

挂载前：

在beforeMount阶段，vue实例的$el和data都初始化了,单还没有挂载之前都是虚拟的dom阶段,data.message还未替换；

this.$refs.box ： 可以获取dom节点

挂载后：

在mounted阶段,vue实例挂载完后,data.message成功渲染。

\*挂载阶段：主要是合成数据和视图模板，先在内存完成虚拟dom解析，最后一次dom操作直接更新到页面。

## 5.3更新前/后

更新前**：**

beforeUpdate，当data变化后,这时视图尚未改变，依旧可以获得改变前的DOM内容

更新后**：**

updated，当data变化后，页面视图已经更新完毕，可以获得最新的DOM信息

\*更新阶段：主要是完成对视图的动态修改。

## 5.4销毁前/后

销毁前：

beforeDestroy，实例销毁之前调用。在这一步，实例仍然完全可用。这时data的改变不会再触发周期函数，说明此时vue实例已经结束了事件监听以及和dom的绑定，但是dom结构依然存在。

销毁后：

destroyed，Vue 实例销毁后调用。这时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。

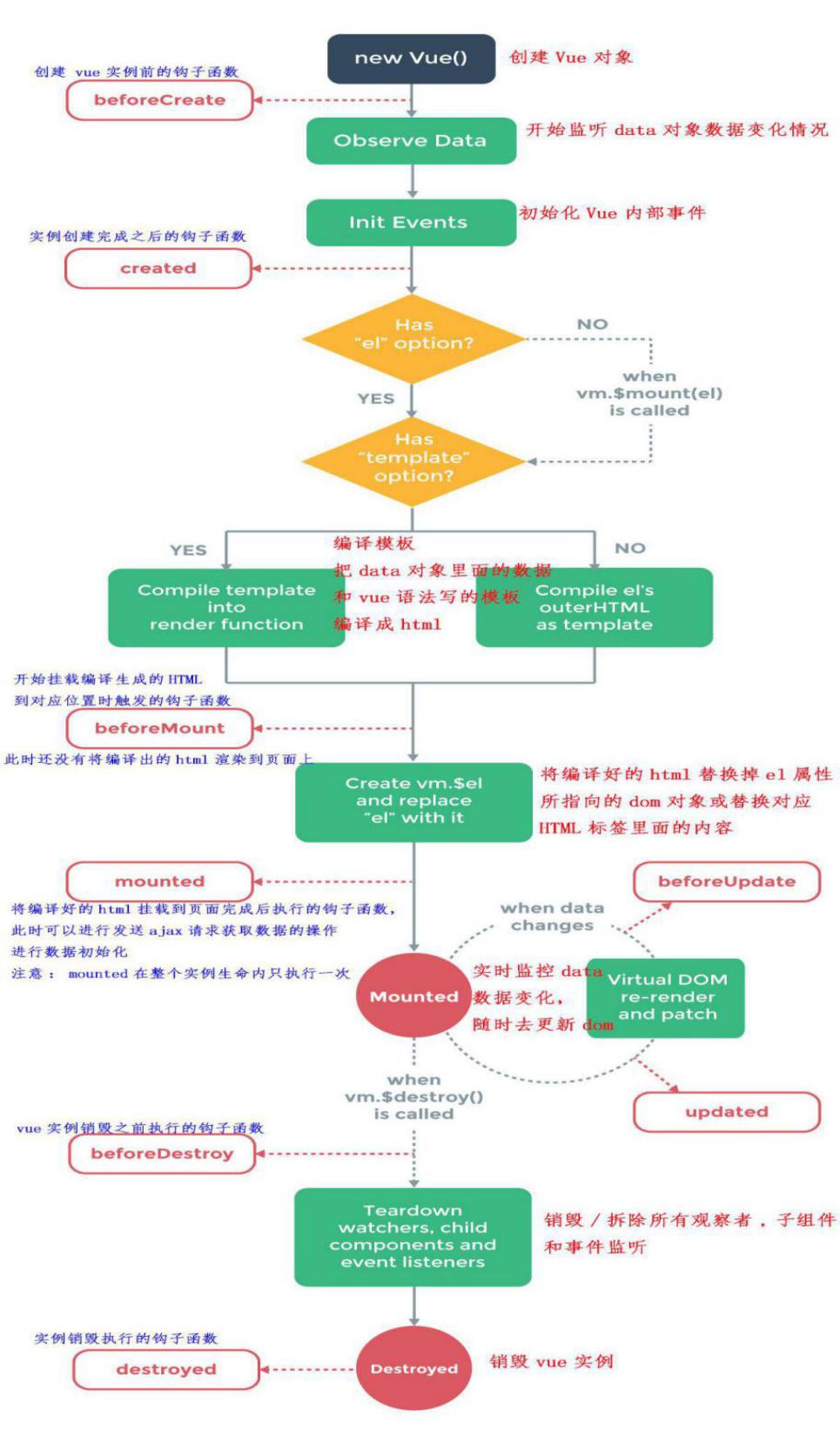
\*销毁阶段：主要是完成对组件的回收。

发请求最佳时间：如果你需要在created中，操作dom，必须添加一个函数

created(){

this.$nextTick( () => { /\* code \*/ } **)**

}



# 6. 前端路由Vue-router【掌握】

## 6.1什么是前端路由

前端路由定义了地址和组件的一一对应关系（一个url地址，对应一个页面级别的组件）

## 6.2前端路由优缺点

**优点**

1. 路由处理速度较快，不用每次都通过服务返回。
2. 前端拥有更多的主动权。

**缺点**

1. 使用浏览器的前进，后退键的时候会重新发送请求，没有合理地利用缓存。
2. 单页面无法记住之前滚动的位置，无法在前进，后退的时候记住滚动的位置。

## 6.3vue-router基本用法

Vue-Router 是 Vue官方的一个插件，可以按需安装。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。

### 6.3.1安装

方式1--手动安装： yarn add vue-router

方式2--脚手架安装： 直接使用脚手架集成

### 6.3.2配置

1. 创建router.js文件，引入路由插件，然后使用Vue.use()加载插件

// 引入核心库

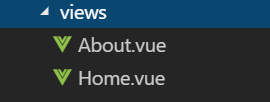
import Vue from 'vue'

import Router from 'vue-router'

// 全局注册路由

Vue.use(Router)

1. 创建两个组件文件,并编写内容



1. 在路由文件router.js里面，配置路由和组件的映射关系，每一个路由映射一个组件.

路由的配置有2种：

直接引入

按需加载

import Home from './views/Home.vue'

// 定义路线配置

const routes = [ // 将来只需修改路线集

{

path: '/',

redirect: { name: 'home' }

},

{

path: '/home',

name: 'home',

component: Home

},

{

path: '/about',

name: 'about',

//路由级别代码分割

//这会为该路由生成一个单独的块(about.[hash].js)

//当访问路由时，延迟加载。

component: () => import(/\* webpackChunkName: "about" \*/ './views/About.vue')

}

]

1. 然后，继续在router.js里面，完成配置和路由实例

// 根据路线配置，创建路由实例，并导出

export default new Router({

**routes** // routes即 ‘routes’:routes的简称

})

1. 回到入口文件main.js，在全局Vue实例上配置路由

import router from './router'

new Vue({

**router,** // router即 ‘router’:router的简称

render: h => h(App)

}).$mount('#app')

### 6.3.3使用

在App.vue里面，添加一个路由出口，使用<router-view><router-view>来挂载所有的路由组件。

<div id="app">

<**router-view**/>

</div>

<router-view>会根据当前路由动态渲染不同的页面组件

## 6.4路由导航

Vue-router里面有内置的组件<router-link>可以实现跳转，他会被渲染成一个a标签

<div id="app">

<div id="nav">

<**router-link** to="/">Home</router-link> |

<router-link to="/about">About</router-link>

</div>

<router-view/>

</div>

## 6.5路由嵌套

在某些场景中，页面只需要做局部更新。在路由中再定义子路由，叫做路由嵌套。内部使用children属性来定义。

var router = new VueRouter({

        routes:[

            {path:"/",component:index},

            {path:"/product",component:product},

            {path:"/about",component:about,

**children**: [

                    {path : "" , component :us},

                    {path : "company" , component :company},

                ]

            }

]});

如上所示，嵌套的子组件配置和普通路线配置完全一样，也是由2大核心配置“path（请求路径）”和“component（视图组件）”两部分组成。

## 6.6路由中的全局对象

### 6.6.1$router-全局路由对象

this.$router.push()： 跳转到不同的url，但这个方法会向history栈添加一个记录，点击后退会返回到上一个页面。

this.$router.replace()：替换当前路由，不会向history栈添加一个记录，无法通过history返回到上一个页面。

### 6.6.2$route-获取当前路线信息

this.$route.path : 获取当前页面对应的路线地址

# 7.课程总结

## 7.1重点

v-bind(掌握)

组件通信（掌握）

## 7.2难点

路由嵌套

## 7.3如何掌握？

交互代码敲2遍 写手一遍路由的配置

# 8.课后练习

**作业难度：☆☆☆**

**完成时间: 90分钟**

**简答题（手抄笔记）**

1. 简述vue的生命周期?

2、手写路由配置

**操作题（电脑操作）**

1、课堂代码（常见配置项、生命周期、路由）

2、使用v-bind / v-show / v-if 实现一个简单的走马灯效果。

3、使用路由来实现户籍系统改版。

1） 提供户籍列表和户籍添加2个组件

2） 把组件配置到路由

3） 列表组件提供添加按钮，能够切换到添加视图

4） 添加组件保存后，把数据追加到本地存储，跳回到列表页

5） 列表页在生命周期方法中，加载本地存储的数据，渲染列表

6） 点击删除，可以删除一条数据。

# 面试题

**1. 说说路由有哪些方式可以跳转 ?**

# 10.扩展知识