框架：framework有着自己的语法 ，都有对应的各个模块

库：library专注于一点

框架的好处：1、提高代码的质量，开发速度

2、提高代码的复用率

3、降低模块之间的耦合度（高内聚低耦合）

vue cn.vue.org 渐进式构建用户界面的js框架

简单的表单处理到复杂的数据操作复杂的单页面引用陈序

方便阅读的中文文档

容易上手

体积小

基于组件化的开发方式

代码的可读性、可维护性得到了提高

工作方式：丰富的指令去扩展模板，可以通过各种各样的插件增强功能

搭建环境：官网

双花括号mustache(胡子)/interop（差值表达式）

<any>{{表达式}}</any>

作用：将表达式执行的结果输出给调用元素的innerHTML中

<div>{{表达式}}</div>花括号内的运输与js无意，只是数据取自绑定当前dom的data中

建立Vue（）对象

new Vue({el: 选择器 ，data:{msg:xxx }});

指令-循环指令

<any v-for=’i in array’>{{i}}</any>=== for(let i of array){html += <any>i</any>}

<any v-for=’(value, index) in arry’></any>=== for(let index =0;index<array.length; index ++){html += <any>i</any>}

给dom绑定个key可以更好的定位

<any v-for=”” key=””>

指令-选择指令

<any v-if=”表达式”></any>根据表达式执行结果的真假，来决定是否将元素挂载到DOM树上

<any v-else-if=”表达式”></any>

<any v-else=”表达式”></any>

指令-事件绑定

<button v-on/@:事件名=”事件函数”></button>

new Vue({

el:’选择器’，

data:{

}

methods:{

事件函数:function(){

方法处理函数

this.数据：修改data中的数据

}

}

})

指令-属性绑定

<any v-bind:/:myProp=”表达式”></any>

作用：将表达式执行的结果绑定到当前元素的myProp属性

<any :style=”{css样式：Vue值}”></any>

<any class=”{属性：true}”></any>

{}是以对象的形式绑定数据，还可以用数组形式与直接写值

调用绑定属性才能从Vue中取值

双向数据绑定：

双向1：数据绑定到视图

双向2：视图（表单元素）绑定到数据

<表单元素 v-model=”变量”(为value值)>

<any v-show=’boolean’></any>改变display

<any v-html=”dat”>改变HTML</any>

组件化

组件：就是可以被反复使用的，带有特定功能的视图

像玩积木一样，把封装的组件复用，构成复杂页面

组件创建

Vue.component(‘id指定的命名’，{ 为全局组件

data:function(){ return obj }组件中初始数据应该放在function中

template:HTML模板,

watch:{监听数据：function(newValue, oldValue){}}

});

<script id=”1xx”type=”text/x-template”></script>

Vue.component(‘’，{

template: #1xx

})

组件使用：HTML调用：在Vue定义的HTML内<id指定的命名><、id指定的命名>

new Vue({

component:{

‘id定义的HTML标签’：{

template:HTML模板

}

}

})

！！组件只能存在一个根HTML标签

！！组件的HTML标签命名最好为xx-xxxx

修饰符 .xxxxx

<form @submit.prevent>阻止表单默认行 == e.preventDefault();

<any @xxx.stop>取消冒泡 == e.stopPropagation();

<input v-model.number=”name”>将v-model监听对象转为数字

自定义指令： ！！！要在Vue对象之前创建

Vue.directive(‘change’, {

bind:function(el(调用此方法的dom), bindings(含指令原始名，加工名，值-value)){} 首次调用执行

})

inserted(){}插入父节点时候调用

componentUpdate（）{} :被绑定的元素所在模板完成一次更新更新周期的时候调用

unbind:function(){}解除绑定时调用的函数

update:function(el(调用此方法的dom), bindings(多一个oldValue)){}有数据变化都会调用

调用自定义指令： <any v-change=“参数”></any>

过滤器：针对一些数据进行筛选，过滤，格式化等相关的处理，变成我们想要的数据

Vue.filter(‘名’,function(参数，之后的参数){

return 处理后的结果

});

<any>{{参数 | 过滤器（之后的参数）}}</any>

复合组件：在一个组件中调用其他组件

template: `其他的组件`

生命周期

1. beforeCreate/create 准备工作 初始方法于此
2. beforeMount/mount 挂载前后针对元素的操作 DOM操作于此
3. beforeUpdata/update 数据变化处理
4. beDestroy/destroy 清理工作 定时器之类

都可以接 ：function(){}

组件是基于Vue的实例而言的，所以Vue实例都可以用

常用属性

new Vue({

el:

data:

methods:

watch(监听属性,用于双向监听):{

监听数据对象data：function(newValue, oldValue){}

computed(计算属性，模板中复杂计算应该使用 依赖缓存效率高):function(){

函数名：function(){}

component:

driective:

}

}

})

组件间通信

父子通信：父 pass props 子， 子 emit event 父

父到子 子被动

父<子组件 参数=“属性”></子组件>

子： Vue.component({

props:[‘参数’] 与Vue中的data差不多可以直接输入参数名就能得到值

template: {{参数}}

})

父到子 子主动拿数据

子组件中：

组件周期函数：function(){

this.$parent.方法、名称

}

子到父，父组件被动拿值

1. 绑定
   1. methods:{函数（）{this.msg= 值}}
   2. data:function(){retrun {msg}}
   3. <father>
   4. <son @customEvent=’函数($event不写好一点)’></son>
   5. </father>
2. 触发
   1. 子组件内部：this.$emit(‘customEvent’，值);

子到父，父组件主动拿值

组件周期函数：funciton(){

this.$ref.名称(子组件).子组件里的值/方法

}

<father>

<son ref=”名称”></son>

</father>

兄弟组件通信

1、给父组件传值在通信

2、bus

创建bus :var bus= new Vue();

接受组件：beforeCreate:function(){bus.$on(‘事件名’， function(msg){接收后的事件})}在bus.$on中操作data数据要记得提前分装this

发送组件：methods: {发送函数：function(){bus.$emit(‘接收组件的方法名’，值)}}

<any @事件名=“发送函数”>

路由模块：建立起url和页面的映射关系

1、spa的基本概念和工作原理

single page application 但以页面应用程序，他加载页面只会跟新局部的界面不会刷新整个页面。

解析地址栏：完整页面地址、路由地址

根据路由地址，从路由词典中找到真的加载页面

发起ajax请求

指定位置插入请求文件

2、路由模块的基本使用

引入vue.js vue-router.js

指定一个容器 router-view

<router-view><router-view>

创建业务所需要的组件类

let 变量= Vue.component(‘’, {})

配置路由词典

const myRoutes= [{path:’/路径’, component: 变量}，。。。] (需要指定path为空时的组件，还需要指定路由为\*的404)路由数组

const myRouter = new VueRouter({routes: myRoutes})

new Vue({router: myRouter });

测试

使用路由器模块来实现页面跳转：

1. 网址修改
2. 定义方法：funciton(){this.$router（路由器）.push(path)}
3. <router-link to=”路由地址”></router-link>

完成参数的传递

1. 配置接收方的路由地址
   1. let myRoutes = [‘/login/:参数名’，component:组件];
2. 取得传参的值：
   1. beforeCreate:function(){

this.$route(路由传来的).params.index

}

路由嵌套：在一个路由中，path对应一格component，如果这个component需要根据不同的url再加载其他的component，就为路由嵌套

a组件中加载B、C组件

给a组件指定<router-view></router-view>

对a编辑路由词典{path:’/a’, component”a, children:[a组件下B、C组件的路由path:’/b’, component:b]}

子路由下根据需求空路由与\*路由可携带父路由地址

由于一个组件内存在其独特的data与methods，所以一个组件不能被多个父级调用

异步方式获取data。this.$nextTick或者setTimeout都行。相当于在初始化前告诉容器，等全执行完了再跑里面的代码。这种方式别说拿data了，拿渲染完DOM都OK~

同步方式的话，是要了解框架内部原理的。在beforeCreate前，所有的options都会先存到vm.$options中，在beforeCreate之后，将$options里的data啦，props啦，methods啦等等一个个附到vm上，然后再触发created钩子。所以在beforeCreate的时候，通过this.message是拿不到值的，在created的时候就能通过this.message拿到值了。

一定要在beforeCreate的时候就同步去拿data里的值的话，就是直接从this.$options.data里去拿。如果data中的初始值是简单的string，那直接this.$options.data()["message"]就好.涉及到复杂点的情况，建议看看源码里是怎么处理的，具体在core/instance/state.js中的initData(vm)里。

但是实际情况中从来没遇到过需要在组件还没初始化就去拿data的……

与远程服务器通信 vue-resouce.js

测试服务器网址：http://jsonplaceholder.tyicode.com

绑定事件methods:{

this.$http.get(‘服务器’,{params}).then((res) => {})

this.$http.post(‘服务器’,{params},{emulateJSON:true}).then((res) => {数据在 res.data中})

}

搭建命令行CIL开发环境

tpls.zip

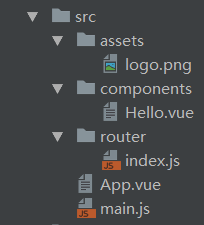
解压，在此文件夹内打开cmd npm install

启动环境npm start

通过node执行build/dev-server.js指定端口号

加载index.html与mian.js

加载src中对应内容



src:开发资源存放

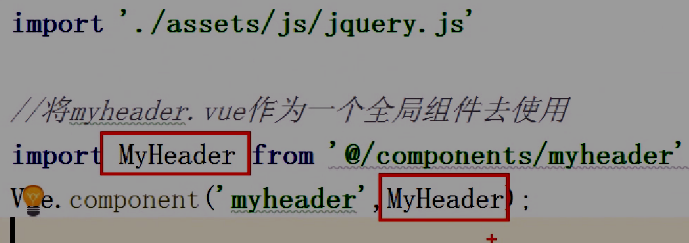
assets静态资源存放

components组件存放

router路由设置存放

app.vue根组件

main.js设置根组件css文件，js文件，创建vue实例



组件的创建方式：单文件组件 xxx.vue

<template></template> === Vue.component.template

<script>

export default{

components:{组件名}调用子组件

}

</script> === vue.component中的其他方法

<style scoped(仅本组件使用)>@import ‘path’</style> === css样式

配置路由

router的index下import 此组件

Vue中的bom操作

this.$router.back()退一页

this.$router.forward()前进一页

this.$router.go(n)走n页

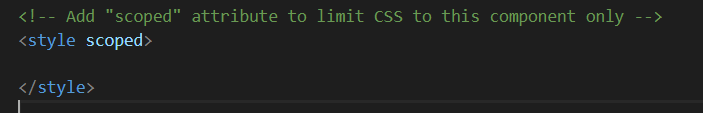
搭建项目脚手架

1. 启动apache和mysql
2. 初始化数据库
3. 需求分析
4. 路由嵌套
5. 搭建脚手架
   1. assets目录下创建css文件夹、js文件夹、img文件夹
   2. 将相应的插件与静态资源拷贝到对应文件夹
6. 创建组件
7. 配置路由词典

用vue脚本记得用Vue.use();

webpack.base.conf.js改设定

动态图片加入 ：src=require(“path”)



如果需要使用less，则安装：

npm install less --save-dev

npm install less-loader --save-dev

搭建最新的vue环境：

npm install vue –g 搭建全局vue

npm install vue-cli –g 搭建全局vue cli

vue init webpack-simple (name) 创建vue工程

