# Vue\_04

# 目标

* 使用混入完成方法添加
* 掌握生命周期
* 能够使用axios发起网络请求
* 定义组件实现父子组件间传值(边界处理)
* 能够使用插槽对组件复用

# 一、混入 - Mixins

混入(mixins)是一种分发Vue组件中可复用功能的非常灵活的方式。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

混入分为：全局和局部

# 全局混入

Vue.mixin({

created(){},

methods:{}

})

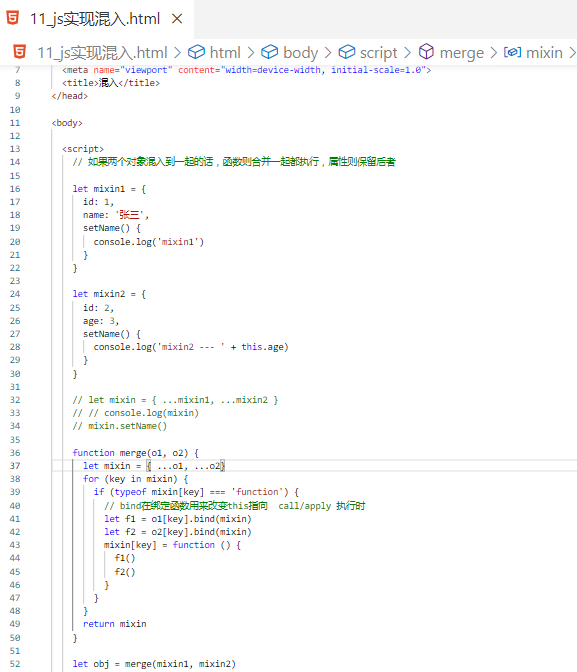
# 局部混入

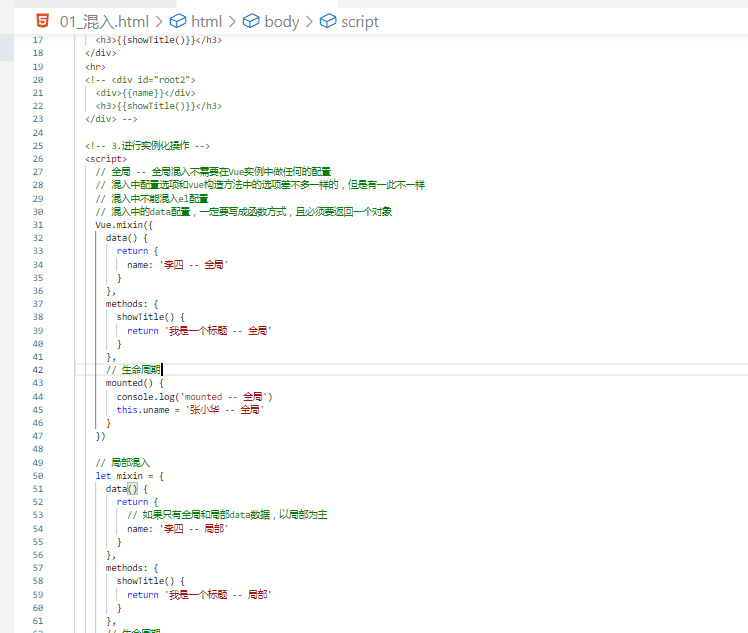
let mixinObj = {}

new Vue({

mixins:[mixinObj]

})





vue混入：

①、代码的复用

②、初始化方法和属性还有权限检测

# 二、实例生命周期

每个 Vue 实例在被创建之前都要经过一系列的初始化过程。例如需要设置数据监听、编译模板、挂载实例到 DOM、在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，给予用户机会在一些特定的场景下添加他们自己的代码。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | beforeCreate | 在实例初始化之后，数据观测和事件配置之前被调用 此时data 和 methods 以及页面的DOM结构都没有初始化 什么都做不了 执行1次 | | created | 在实例创建完成后被立即调用此时data 和 methods已经可以使用 但是页面还没有渲染出来 执行1次 用this对象 | | beforeMount | 在挂载开始之前被调用 此时页面上还看不到真实数据 只是一个模板页面而已 执行1次 | | 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。 数据已经真实渲染到页面上 在这个钩子函数里面可以进行数据请求等 执行1次 | | beforeUpdate | 数据更新时调用，页面上数据还是旧的 n次 | | updated | 由于数据更新完毕，页面上数据已经替换成最新的 n次 | | beforeDestroy | 实例销毁之前调用 执行1次 | | destroyed | 实例销毁后调用 执行1次 | | activated | keep-alive 组件激活时调用 | | deactivated | keep-alive 组件停用时调用 | | errorCaptured | 当捕获一个来自子孙组件的错误时被调用 | |

# 三、网络请求

## 3.1、XmlHttpRequest

var xmlhttp = new XMLHttpRequest()

xmlhttp.onreadystatechange = function () {

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

console.log(xmlHttp.responseText)

}

}

// GET

xmlhttp.open("GET", url, true)

xmlhttp.send(null)

// POST

// xhr.open('POST', 'url', true);

// xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded')

// var data = new FormData();

//data.append('user', 'person');

//xmlhttp.send(data)

## 3.2、Fetch

Fetch是新的ajax解决方案 Fetch会返回Promise对象。fetch不是ajax的进一步封装，而是原生js，没有使用XMLHttpRequest对象。

参数说明

1、第一个参数是URL:

2、第二个设置请求的参数，是可选参数

3、返回使用了Promise 来处理结果/回调

fetch(url, options).then(res=>res.json()/text()).then(ret=>console.log(ret))

Fetch它给我们提供了很多的方法，如POST，GET，DELETE，UPDATE，PATCH和PUT等。默认的是 GET 请求，可以在设置请求参数中指定请求方法，通过method来设置

// get请求

// 通过url表达式来传递数据

fetch('http://xxx/?id=123')

.then(res => res.json())

.then(data => console.log(data))

// post标准提交

fetch('http://xxxx/post', {

method: 'post',

body: 'uname=lisi&pwd=123',

headers: {

'Content-Type': 'application/x-www-form-urlencoded'

}

}).then(res => res.json()).then(data => console.log(data))

// post提交json数据

fetch('http://localhost:3000/books', {

method: 'post',

body: JSON.stringify({

uname: 'lisi',

pwd: '123'

}),

headers: {

'Content-Type': 'application/json'

}

}).then(res => res.json()).then(data => console.log(data))



## 3.3、axios

文档：<https://www.kancloud.cn/yunye/axios/234845>

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和node.js中。能拦截请求和响应，自动转换JSON数据。axios也是vue作者推荐使用的网络请求库。

**使用语法**

# GET

## 方法1

axios.get('/user?ID=12345').then(res=>{})

## 方法2

axios.get('/user', {

params: {

ID: 12345

}

}).then(res=>{})

# POST

axios.post('/user', {

firstName: 'aa',

lastName: 'bb'

}).then(res=>{})

# 标准写法

axios({

method: 'post',

url: '/user/12345',

timeout: 1000,

headers: {'X-Custom-Header': 'foobar'},

data: {

firstName: 'Fred',

lastName: 'Flintstone'

}

})

# axios 全局配置

# 配置公共的请求头

axios.defaults.baseURL = 'https://api.example.com';

# 配置 超时时间 毫秒数(0 表示无超时时间)

axios.defaults.timeout = 1000;

# 配置公共的请求头

axios.defaults.headers.token = 'token值';

get请求



# 四、vue组件

## 4.1、概述

组件 (Component) 是 Vue.js 最强大的功能之一，它是html、css、js等的一个聚合体。

组件化

* 将一个具备完整功能的项目的一部分分割多处使用
* 加快项目的进度
* 可以进行项目的复用

组件注册分为：全局注册和局部注册

## 4.2、全局注册

Vue.component('组件名称', { })第1个参数是标签名称，第2个参数是一个选项对象。全局组件注册后，任何vue实例都可以用。

**组件注意事项：**

* 构造 Vue 实例时传入的各种选项大多数都可以在组件里使用，只有一个例外：data必须是函数，同时这个函数要求返回一个对象
* 组件模板必须是单个根元素
* 组件模板的内容可以是模板字符串

Vue.component('HelloWorld', {

data: function(){

return {

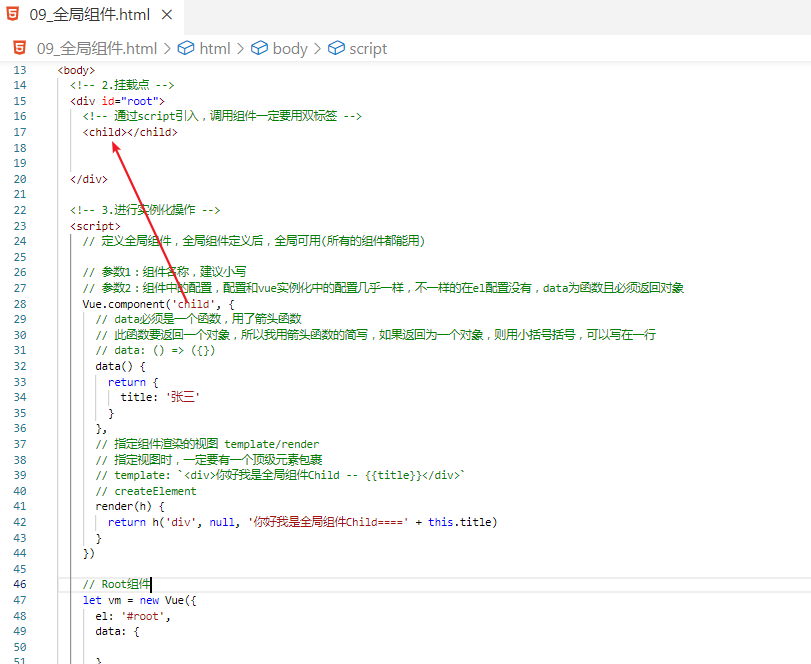
msg: 'HelloWorld'

}

},

template: '<div>{{msg}}</div>'

});



## 4.3、局部注册

只能在当前注册它的vue实例中使用，通过某个 Vue 实例/组件的实例选项 components 注册仅在其作用域中可用的组件

var Child = {

template: '<div>A custom component!</div>'

}

new Vue({

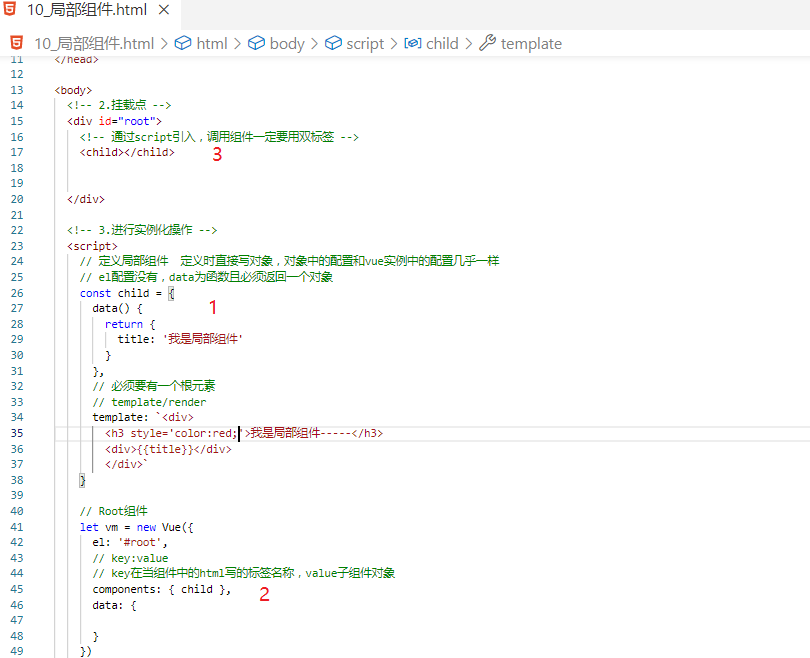
components: {

// <my-component> 将只在父组件模板中可用

'my-component': Child

}

})



## 4.4、组件间传值

### 4.4.1、父组件向子组件传值

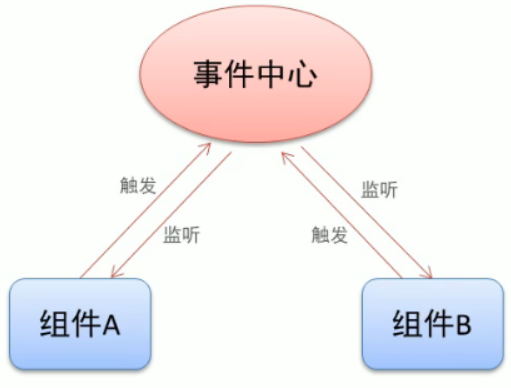
* 父组件以属性的形式绑定值到子组件身上
* 子组件通过使用属性props接收(props是单向绑定的(只读属性)：当父组件的属性变化时，将传导给子组件，但是反过来不会)

### 4.4.2、子组件向父组件传值

* 子组件用$emit()定义自定义事件，$emit()第一个参数为 自定义的事件名称第二个参数为需要传递的数据
* 父组件用v-on(@)绑定子组件定义的自定义事件名，监听子组件的事件，实现通信

### 4.4.3、事件总线

非父子组件或更多层级间组件间传值，在Vue中通过单独的事件中心来管理组件间的传值。



* 建立统一的事件中心

const bus = new Vue()

* 传递数据方，通过一个事件触发bus.$emit(方法名，传递的数据)
* 接收数据方，在生命周期函数中，通过bus.$on(方法名,[params])来监听
* 销毁事件，在接受数据方，通过bus.$off(方法名)销毁之后无法监听数据

### 4.4.4、ref

ref被用来给元素或子组件注册引用信息。引用信息将会注册在父组件的 $refs 对象上。如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例

# 普通dom元素上

<p ref="p">hello</p>

# 子组件上

<child-comp ref="child"></child-comp>

# 调用

this.$refs.引用名

### 4.4.5、root/parent/children

获取父组件对象或子组件对象集合

console.log(this.$parent) //获取父组件对象

console.log(this.$children) //获取子组件对象集合

console.log(this.$root) //获取根组件对象

### 4.4.6、provide/inject

发布与注入，一般用于组件封装中使用

//父组件

provide:{

name:"haha",

say:function(){

console.log("hello")

}

}

//子组件

inject:['name','say']

### 4.4.7、$attrs/$listeners

//在子组件中,通过$attrs可以获取父组件声明的自定义属性(props除外)

//在子组件中,通过$listeners可以获取父组件声明的自定义方法(可能打印不出来,但可以调用)

//$attrs和$listeners可以继续在子组件中绑定,一直传递给后代组件

<comp1 :a='b' :c='d' @test1="fn1" @test2="fn2" v-bind="$attrs" v-on="$listeners" />

## 4.5、内置组件

### 4.5.1、动态组件

通过使用保留的 <component> 元素，动态地绑定到它的 is 特性，我们让多个组件可以使用同一个挂载点，并动态切换

<component v-bind:is="currentView"></component>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'home'

},

components: {

home,

posts

}

})

</script>

### 4.5.2、keep-alive

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。

<keep-alive>

<component :is="currentView"></component>

</keep-alive>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'posts'

},

components: {

home,

posts

}

})

</script>

<!-- 子组件中 -->

<script>

export default {

activated: function () {

console.log("home组件被激活了");

},

deactivated: function () {

console.log("home组件被停用了");

},

};

</script>

### 4.5.3、动画组件

<keep-alive>

<!-- 使用过渡组件实现css过渡效果 -->

<transition name="cook" mode="out-in">

<component :is="currentView"></component>

</transition>

</keep-alive>

<style>

.cook-enter-active,

.cook-leave-active {

transition: all 0.5s;

}

.cook-enter {

opacity: 0;

transform: translateX(100%);

}

.cook-leave-to {

opacity: 0;

transform: translateX(-100%);

}

</style>

# 五、组件插槽

组件的最大特性就是复用性，而用好插槽能大大提高组件的可复用能力

## 5.1、单个插槽

当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

# 子组件

<div>

<h2>我是子组件的标题</h2>

<slot>

只有在没有要分发的内容时才会显示。

</slot>

</div>

# 父组件

<div>

<h1>我是父组件的标题</h1>

<my-component>

<p>这是一些初始内容</p>

<p>这是更多的初始内容</p>

</my-component>

</div>

## 5.2、具名插槽

有时我们需要多个插槽，来完成对应的数据自定义显示。

一个不带 name 的 <slot> 出口会带有隐含的名字“default”。

自2.6.0 起有所更新。已废弃的使用 slot

# 子组件

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

</div>

# 父组件

<app-layout>

// 老写法

<h1 slot="header">这里可能是一个页面标题</h1>

// 新写法

// v-slot 只能添加在 <template> 上

// 简写 v-slot:header == #header

<template v-slot:header>

<h1>这里可能是一个页面标题</h1>

</template>

<p>主要内容的一个段落。</p>

</app-layout>

## 5.3、作用域插槽

作用域插槽是一种特殊类型的插槽，用作一个 (能被传递数据的)可重用模板，来代替已经渲染好的元素。在子组件中，只需将数据传递到插槽，就像你将 prop 传递给组件一样

# 子组件中

Vue.component('child', {

template: `

<div class="child">

<slot name="default" text="我是子组件中的内容"></slot>

</div>

`

})

# 父组件中

<div class="parent">

<child>

// 老写法

<div name="default" slot-scope="props">

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

// 新写法

<template #default="props"

<div>

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

<template/>

</child>

</div>