Vue 核心技术与实战

day05

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

目录

Contents

内置指令
- v-html
- v-model
- v-for
- ...

自定义指令
- v-focus
- v-loading
- v-lazy
- ...

每个指令有着自己各自独立的功能

自定义指令

自定义指令：自己定义的指令，可以封装一些 dom 操作， 扩展额外功能

- 全局注册 - 语法

```
Vue.directive('指令名', {
  "inserted" (el) {
    // 可以对 el 标签，扩展额外功能
    el.focus()
  }
})
```

- 局部注册 - 语法

```
directives: {
  "指令名": {
    inserted () {
      // 可以对 el 标签，扩展额外功能
      el.focus()
    }
  }
}
```

需求：当页面加载时，让元素将获得焦点

(autofocus 在 safari 浏览器有兼容性)

操作dom：dom元素.focus()

```
mounted () {
  this.$refs.inp.focus()
}
```

麻烦

使用：

```
<input v-指令名 type="text">
```

简洁

小结

自定义指令的作用?

封装一些 dom 操作，扩展额外功能，例如获取焦点

自定义指令的使用步骤?

1. 注册 (全局注册 或 局部注册)

   在 inserted 钩子函数中，配置指令dom逻辑

2. 标签上 v-指令名 使用

自定义指令 - 指令的值

需求：实现一个 color 指令 - 传入不同的颜色，给标签设置文字颜色

- 语法：在绑定指令时，可以通过“等号”的形式为指令 绑定 具体的参数值

```
<div v-color="color">我是内容</div>
```

指令的值

- 通过 binding.value 可以拿到指令值，指令值修改会 触发 update 函数。

```
directives: {
  color: {
    inserted (el, binding) {
      el.style.color = binding.value
    },
    update (el, binding) {
      el.style.color = binding.value
    }
  }
}
```

指令的值

小结

1. 通过指令的值相关语法，可以应对更复杂指令封装场景

2. 指令值的语法：

   ① v-指令名 = "指令值"，通过 等号 可以绑定指令的值

   ② 通过 binding.value 可以拿到指令的值

   ③ 通过 update 钩子，可以监听指令值的变化，进行dom更新操作

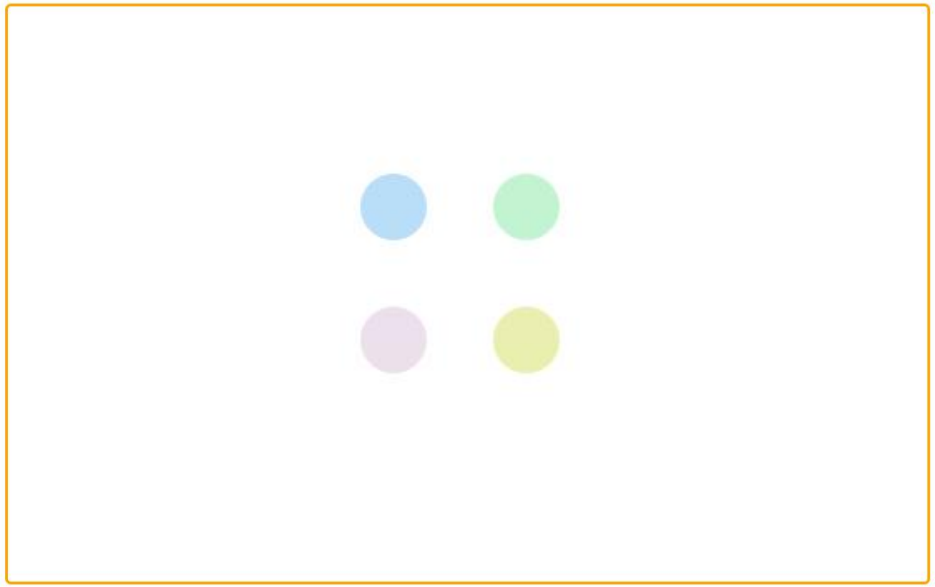
自定义指令 - v-loading 指令封装

场景：实际开发过程中，发送请求需要时间，在请求的数据未回来时，页面会处于空白状态 => 用户体验不好

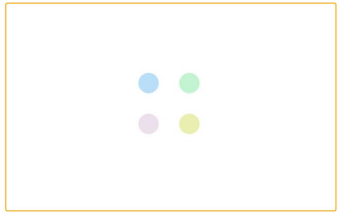
自定义指令 - v-loading 指令封装

场景：实际开发过程中，发送请求需要时间，在请求的数据未回来时，页面会处于空白状态 => 用户体验不好

需求：封装一个 v-loading 指令，实现加载中的效果



自定义指令 - v-loading 指令封装



分析：

1. 本质 loading 效果就是一个蒙层，盖在了盒子上
2. 数据请求中，开启loading状态，添加蒙层
3. 数据请求完毕，关闭loading状态，移除蒙层

实现：

1. 准备一个 loading 类，通过伪元素定位，设置宽高，实现蒙层
2. 开启关闭 loading 状态（添加移除蒙层），本质只需要添加移除类即可
3. 结合自定义指令的语法进行封装复用

```
.loading:before {
  content: "";
  position: absolute;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  background: #fff url("./loading.gif")
no-repeat center;
}
```

小结

1. 通过指令相关语法，封装了指令 v-loading 实现了请求的loading效果
2. 核心思路：

   (1) 准备类名 loading，通过伪元素提供遮罩层

   (2) 添加或移除类名，实现loading蒙层的添加移除

   (3) 利用指令语法，封装 v-loading 通用指令

   inserted 钩子中，binding.value 判断指令的值，设置默认状态

   update 钩子中，binding.value 判断指令的值，更新类名状态

目录

Contents

插槽 - 默认插槽

作用：让组件内部的一些 结构 支持 自定义

需求：将需要多次显示的对话框，封装成一个组件

问题：组件的内容部分，不希望写死，希望能使用的时候自定义。怎么办？

插槽 - 默认插槽

插槽基本语法：

1. 组件内需要定制的结构部分，改用<slot></slot>占位

2. 使用组件时，<MyDialog></MyDialog>标签内部，传入结构替换slot

```
<template>
  <div class="dialog">
    <div class="dialog-header">
      <h3>友情提示</h3>
      <span class="close">✖  </span>
    </div>

    <div class="dialog-content">
      <slot></slot>
    </div>
    <div class="dialog-footer">
      <button>取消</button>
      <button>确认</button>
    </div>
  </div>
</template>
```

```
<MyDialog>
  你确认要退出本系统么？
</MyDialog>
```

小结

场景：当组件内某一部分结构不确定，想要自定义怎么办？

用插槽 slot 占位封装

使用：插槽使用的基本步骤？

1. 先在组件内用 slot 占位

2. 使用组件时，传入具体标签内容插入

插槽 - 后备内容（默认值）

通过插槽完成了内容的定制，传什么显示什么，但是如果不传，则是空白

能否给插槽设置 默认显示内容 呢？

插槽 - 后备内容（默认值）

插槽后备内容：封装组件时，可以为预留的 `<slot>` 插槽提供后备内容（默认内容）。

- 语法: 在 <slot> 标签内，放置内容，作为默认显示内容
- 效果:
  - 外部使用组件时，不传东西，则slot会显示后备内容

```
<MyDialog></MyDialog>
```

  - 外部使用组件时，传东西了，则slot整体会被换掉

```
<MyDialog>我是内容</MyDialog>
```

```
<template>
  <div class="dialog">
    <div class="dialog-header">
      <h3>友情提示</h3>
      <span class="close">✖  </span>
    </div>

    <div class="dialog-content">
      <slot>我是后备内容</slot>
    </div>
    <div class="dialog-footer">
      <button>取消</button>
      <button>确认</button>
    </div>
  </div>
</template>
```

默认显示的内容

小结

如何给插槽设置默认显示内容?

在slot标签内，写好后备内容

什么时候插槽后备内容会显示?

当使用组件并未给我们传入具体标签或内容时

插槽 - 具名插槽

需求：一个组件内有多处结构，需要外部传入标签，进行定制

默认插槽：一个的定制位置

插槽 - 具名插槽

具名插槽语法：

1. 多个slot使用name属性区分名字

```
<div class="dialog-header">
  <slot name="head"></slot>
</div>
<div class="dialog-content">
  <slot name="content"></slot>
</div>
<div class="dialog-footer">
  <slot name="footer"></slot>
</div>
```

2. template配合v-slot:名字来分发对应标签

```
<MyDialog>
  <template v-slot:head>
    大标题
  </template>
  <template v-slot:content>
    内容文本
  </template>
  <template v-slot:footer>
    <button>按钮</button>
  </template>
</MyDialog>
```

插槽 - 具名插槽

具名插槽简化语法：

1. 多个slot使用name属性区分名字

```
<div class="dialog-header">
  <slot name="head"></slot>
</div>
<div class="dialog-content">
  <slot name="content"></slot>
</div>
<div class="dialog-footer">
  <slot name="footer"></slot>
</div>
```

2. template配合v-slot:名字来分发对应标签

```
<MyDialog>
  <template #head>
    大标题
  </template>
  <template #content>
    内容文本
  </template>
  <template #footer>
    <button>按钮</button>
  </template>
</MyDialog>
```

3. v-slot:插槽名 可以简化成 #插槽名

组件内 有多处不确定的结构 怎么办?

具名插槽

1. slot占位, 给name属性起名字来区分

2. template配合 v-slot:插槽名 分发内容

v-slot:插槽名 可以简化成什么?

#插槽名

插槽 - 作用域插槽

作用域插槽：定义 slot 插槽的同时，是可以传值的。给 插槽 上可以 绑定数据，将来 使用组件时可以用。

场景：封装表格组件

1. 父传子，动态渲染表格内容
2. 利用默认插槽，定制操作列
3. 删除或查看都需要用到 当前项的 id，属于 组件内部的数据

   通过 作用域插槽 传值绑定，进而使用

```
<MyTable :list="list">
  <button>删除</button>
</MyTable>

<MyTable :list="list2">
  <button>查看</button>
</MyTable>
```

序号	姓名	年纪	操作
1	张小花	18	删除
2	孙大明	19	删除
3	刘德忠	17	删除

序号	姓名	年纪	操作
1	赵小云	18	查看
2	刘蓓蓓	19	查看
3	姜肖泰	17	查看

插槽 - 作用域插槽

基本使用步骤：

1. 给 slot 标签，以 添加属性的方式传值

```
<slot :id="item.id" msg="测试文本"></slot>
```

2. 所有添加的属性，都会被收集到一个对象中

```
{ id: 3, msg: '测试文本' }
```

3. 在template中，通过 `#插槽名="obj"` 接收，默认插槽名为 default

```
<MyTable :list="list">
  <template #default="obj">
    <button @click="del(obj.id)">删除</button>
  </template>
</MyTable>
```

小结

作用域插槽的作用是什么？

可以给插槽上绑定数据，供将来使用组件时使用

作用域插槽使用步骤？

（1）给 slot 标签, 以 添加属性的方式传值

（2）所有属性都会被收集到一个对象中

（3）template中, 通过 ` #插槽名= "obj" ` 接收

目录

Contents

综合案例 - 商品列表

需求说明：

1. my-tag 标签组件封装
   (1) 双击显示输入框，输入框获取焦点
   (2) 失去焦点，隐藏输入框
   (3) 回显标签信息
   (4) 内容修改，回车 → 修改标签信息
2. my-table 表格组件封装
   (1) 动态传递表格数据渲染
   (2) 表头支持用户自定义
   (3) 主体支持用户自定义

编号	图片	名称	标签
101		梨皮朱泥三绝清代小品壶经典款紫砂壶	茶具
102		全防水HABU旋钮牛皮户外徒步鞋山宁泰抗菌	男鞋
103		毛茸茸小熊出没，儿童羊羔绒背心73-90cm	儿童服饰
104		基础百搭，儿童套头针织毛衣1-9岁	儿童服饰

小结

商品列表的实现封装了几个组件？

2个组件，标签组件 和 表格组件

封装用到的核心技术点有哪些？

（1）props父传子 $emit子传父 v-model

（2）$nextTick 自定义指令

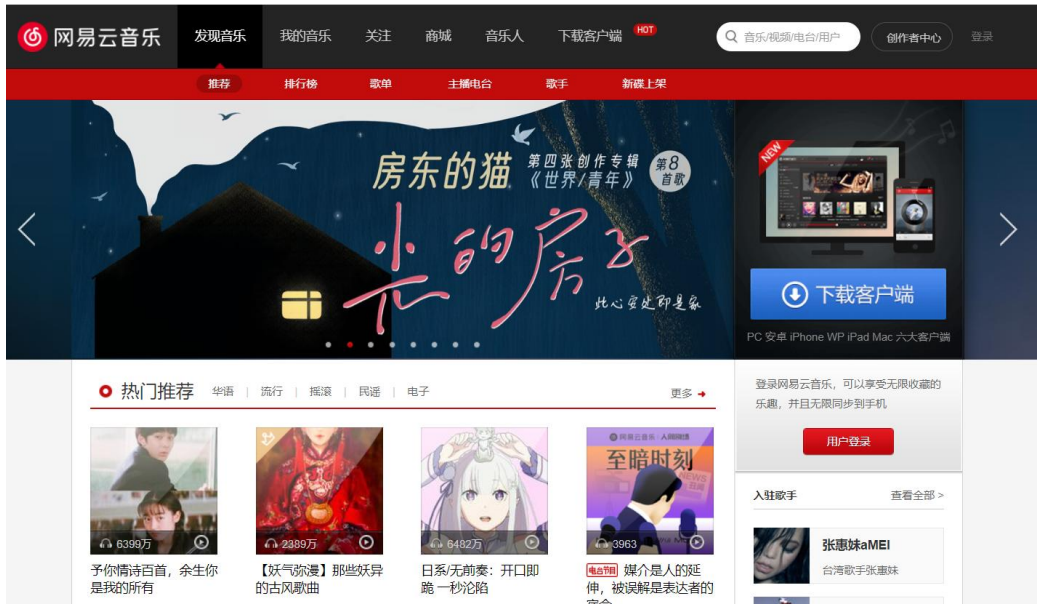
（3）插槽：具名插槽，作用域插槽

目录

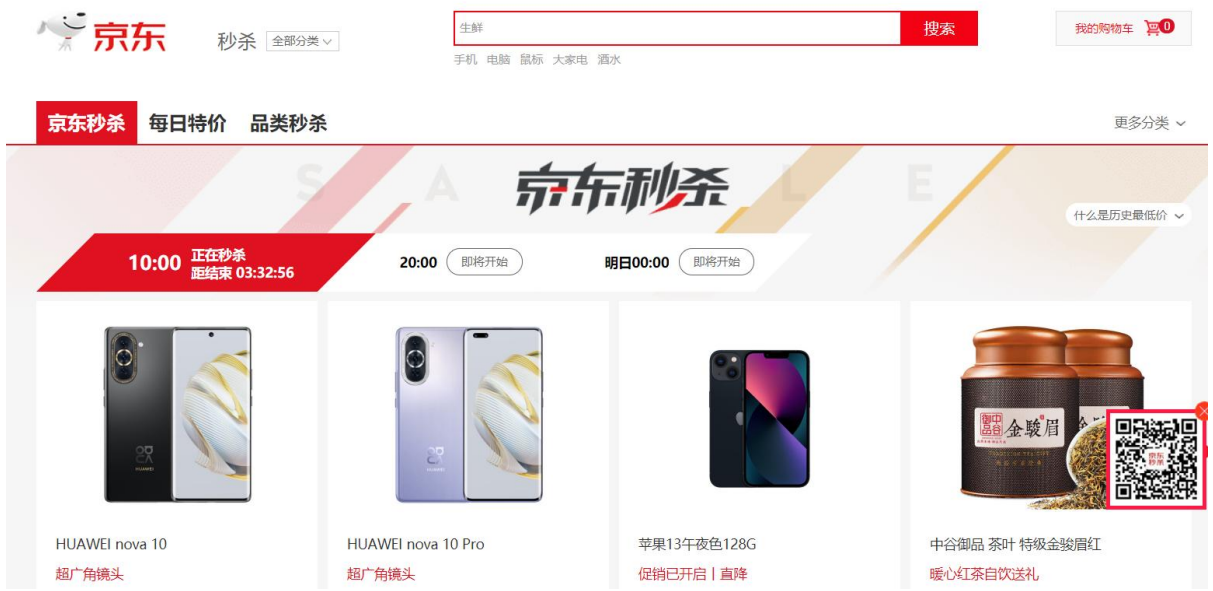
Contents

单页应用程序: SPA - Single Page Application

- 单页面应用(SPA): 所有功能在 一个html页面 上实现
- 具体示例: 网易云音乐 https://music.163.com/



单页面应用



多页面应用

单页应用程序: SPA - Single Page Application

开发分类	实现方式	页面性能	开发效率	用户体验	学习成本	首屏加载	SEO
单页	一个html页面	按需更新 性能高	高	非常好	高	慢	差
多页	多个html页面	整页更新 性能低	中等	一般	中等	快	优

单页面应用　VS　多页面应用

系统类网站 / 内部网站 / 文档类网站 /移动端站点

公司官网 / 电商类网站

小结

1. 什么是单页面应用程序?

   所有功能在一个html页面上实现

2. 单页面应用优缺点?

   优点：按需更新性能高，开发效率高，用户体验好

   缺点：学习成本，首屏加载慢，不利于SEO

3. 应用场景?

   系统类网站 / 内部网站 / 文档类网站 /移动端站点

思考

单页面应用程序，之所以开发效率高，性能高，用户体验好

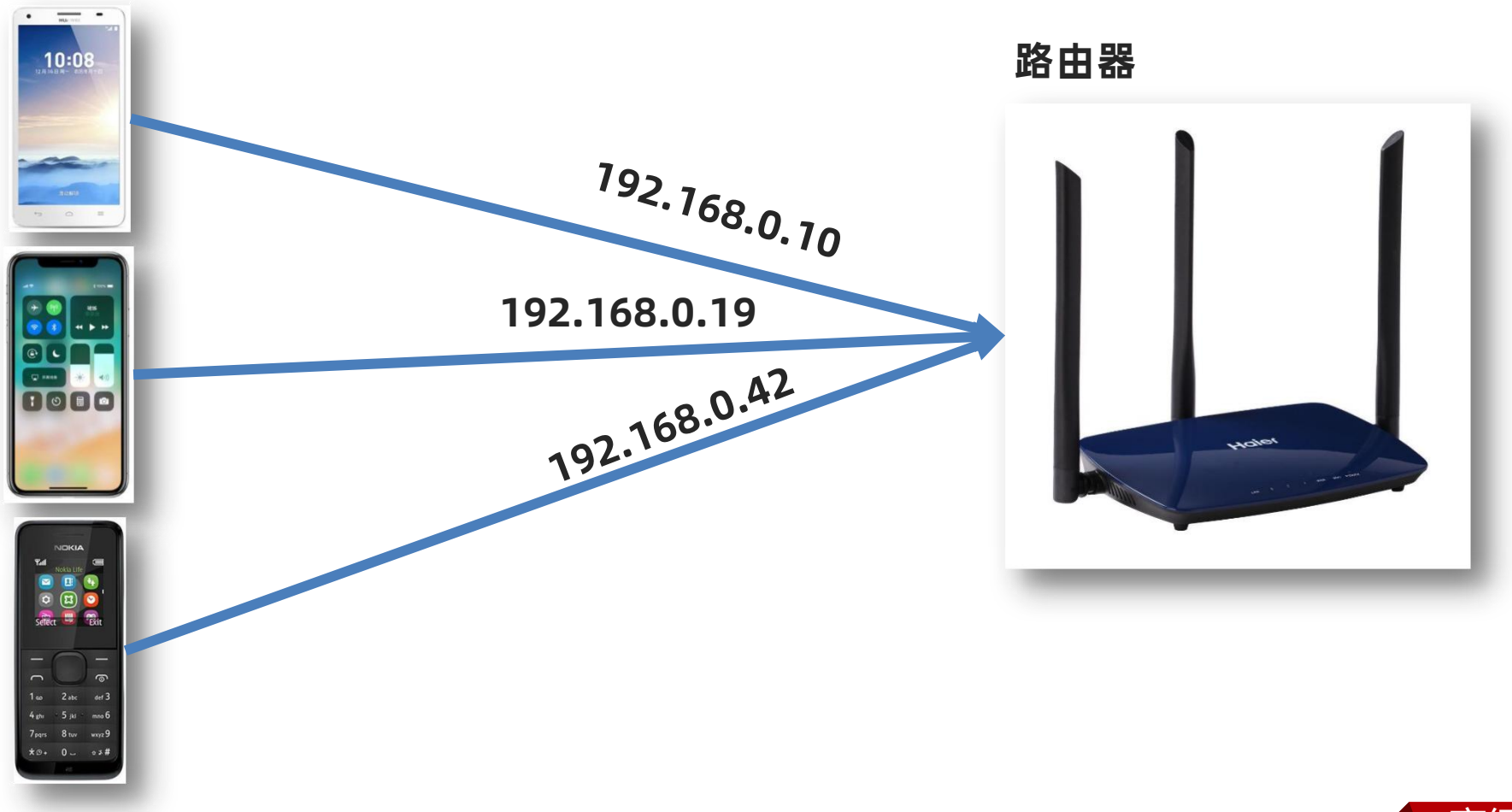
最大的原因就是：页面按需更新

要按需更新，首先就需要明确：访问路径 和 组件的对应关系！

访问路径 和 组件的对应关系如何确定呢？ 路由

路由的介绍

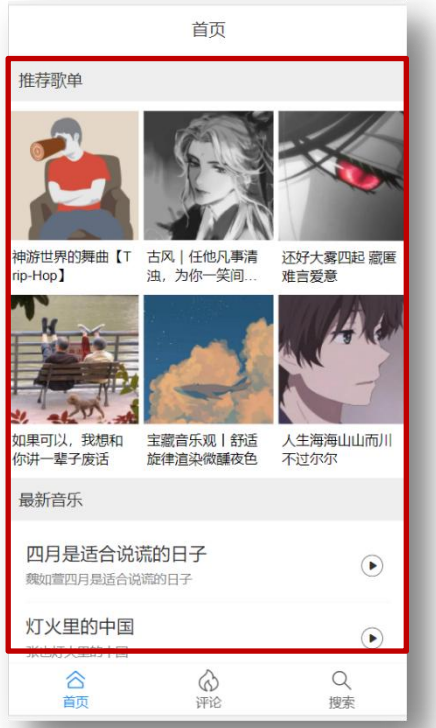
生活中的路由：设备和ip的映射关系



路由的介绍

Vue中路由：路径 和 组件 的 映射 关系

http://localhost:8080/#/home



Home.vue 组件

http://localhost:8080/#/comment

Comment.vue组件

http://localhost:8080/#/search

Search.vue组件

小结

1. 什么是路由？

   路由是一种映射关系

2. Vue中的路由是什么？

   路径 和 组件 的映射关系

   根据路由就能知道不同路径的，应该匹配渲染哪个组件

VueRouter 的 介绍

目标：认识插件 VueRouter，掌握 VueRouter 的基本使用步骤

作用：修改地址栏路径时，切换显示匹配的组件

说明：Vue 官方的一个路由插件，是一个第三方包

官网：https://v3.router.vuejs.org/zh/

Vue Router

Vue.js 官方的路由管理器。

VueRouter 的 使用 (5 + 2)

5个基础步骤 (固定)

① 下载： 下载 VueRouter 模块到当前工程，版本3.6.5

```
yarn add vue-router@3.6.5
```

② 引入

```
import VueRouter from 'vue-router'
```

③ 安装注册

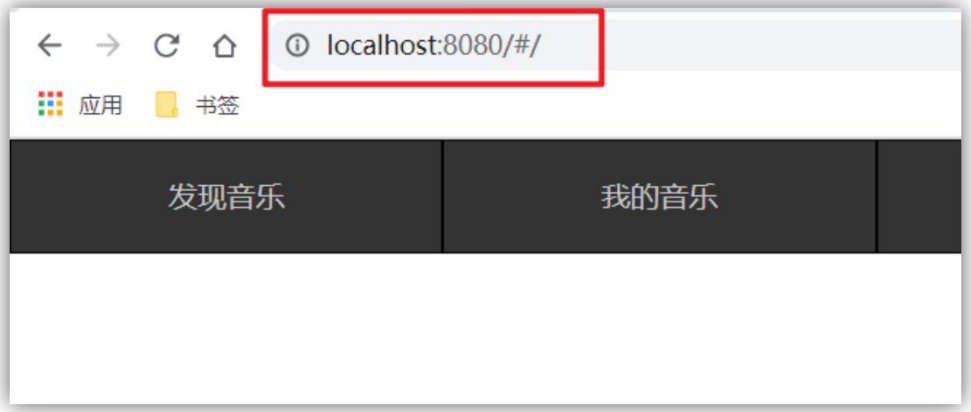
```
Vue.use(VueRouter)
```

④ 创建路由对象

```
const router = new VueRouter()
```

⑤ 注入，将路由对象注入到new Vue实例中，建立关联

```
new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```



VueRouter 的 使用 (5 + 2)

2 个核心步骤

① 创建需要的组件 (views目录)，配置路由规则

Find.vue　My.vue　Friend.vue

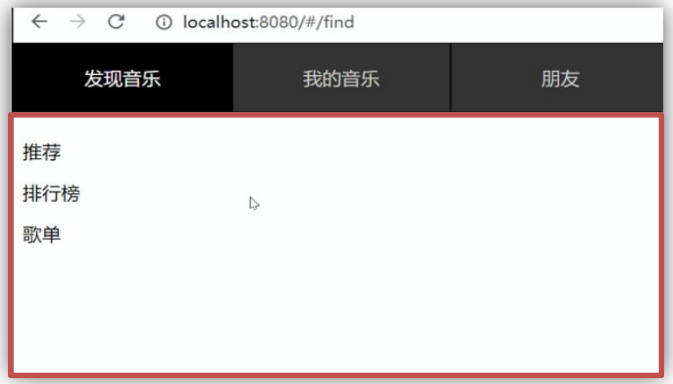
```
import Find from './views/Find.vue'
import My from './views/My.vue'
import Friend from './views/Friend.vue'

const router = new VueRouter({
  routes: [
    { path: '/find', component: Find },
    { path: '/my', component: My },
    { path: '/friend', component: Friend },
  ]
})
```

地址栏路径　　组 件

② 配置导航，配置路由出口(路径匹配的组件显示的位置)

```
<div class="footer_wrap">
  <a href="#/find">发现音乐</a>
  <a href="#/my">我的音乐</a>
  <a href="#/friend">朋友</a>
</div>
<div class="top">
  <router-view></router-view>
</div>
```



1. 如何实现 路径改变，对应组件 切换？

   Vue 官方插件 VueRouter

2. VueRouter 的使用基本步骤？(5 + 2)

   5个基础步骤

   ① 下包　② 引入　③ Vue.use 安装注册

   ④ 创建路由对象　⑤ 注入Vue实例

   2个核心步骤

   ① 创建组件，配置规则 (路径组件的匹配关系)

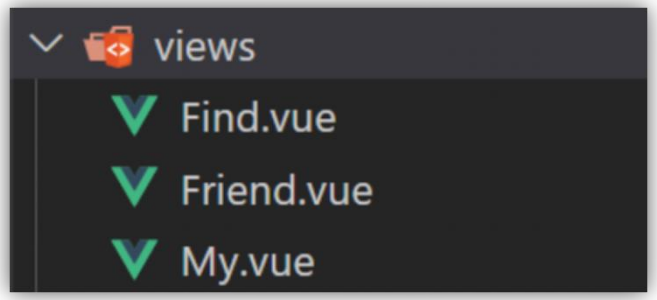
   ② 配导航，配置路由出口　router-view (组件展示的位置)

组件存放目录问题

注意：.vue文件 本质无区别。

路由相关的组件，为什么放在 views 目录呢？ 组件分类

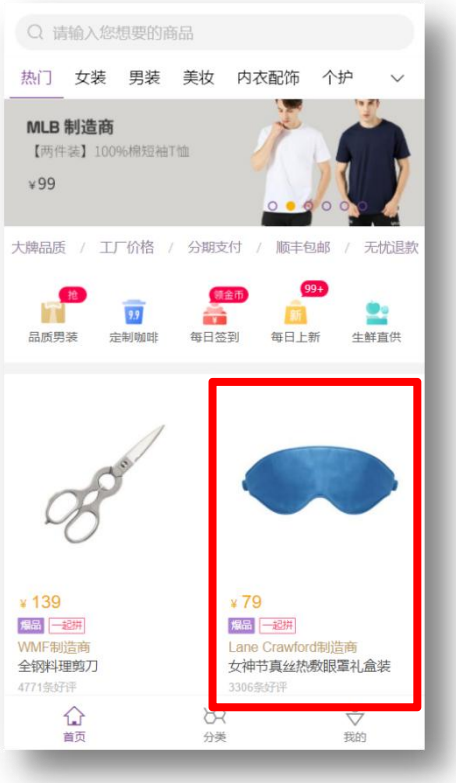
```
import Find from './views/Find.vue'
import My from './views/My.vue'
import Friend from './views/Friend.vue'
```



组件存放目录问题 (组件分类)

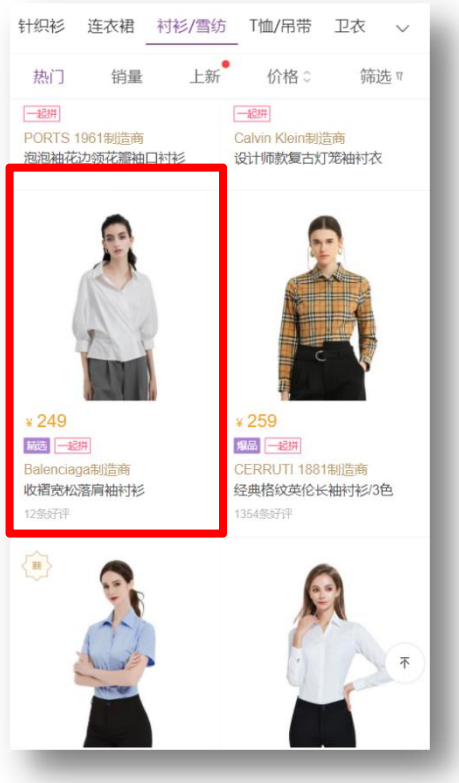
组件分类： .vue文件分2类； 页面组件 & 复用组件　　注意：都是 .vue文件 (本质无区别)

页面组件

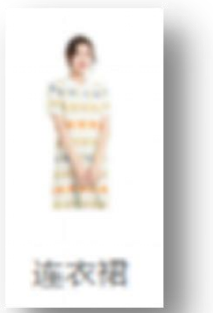


复用

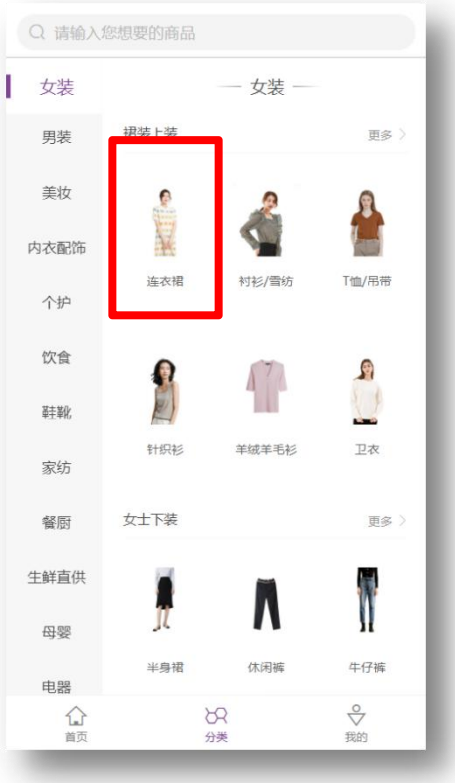
页面组件



复用



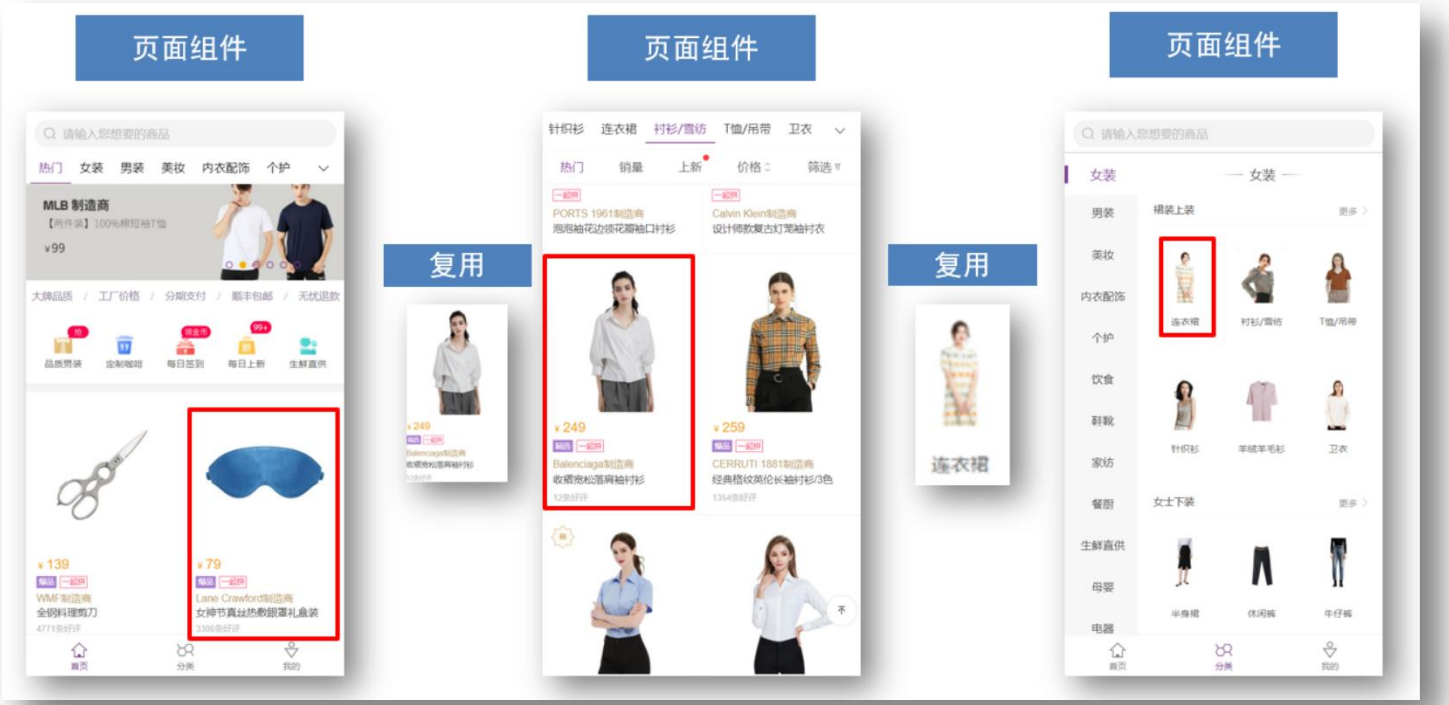
页面组件



组件存放目录问题 (组件分类)

分类开来 更易维护

- src/views文件夹
  - **页面组件** - 页面展示 - 配合路由用
- src/components文件夹
  - **复用组件** - 展示数据 - 常用于复用



组件存放目录问题 (组件分类)

小练习：以下 .vue 文件，属于什么分类组件？应该放在哪个目录？

Home.vue (首页)

Category.vue (分类页)

页面组件 配合路由用

放在 views 目录

组件存放目录问题 (组件分类)

小练习：以下 .vue 文件，属于什么分类组件？应该放在哪个目录？

Product.vue (商品组件)

Comment.vue (评价组件)

复用组件

放在 components 目录

1. 组件分类有哪两类？分类的目的？

    页面组件 和 复用组件，便于维护

2. 放在什么文件夹？作用分别是什么？

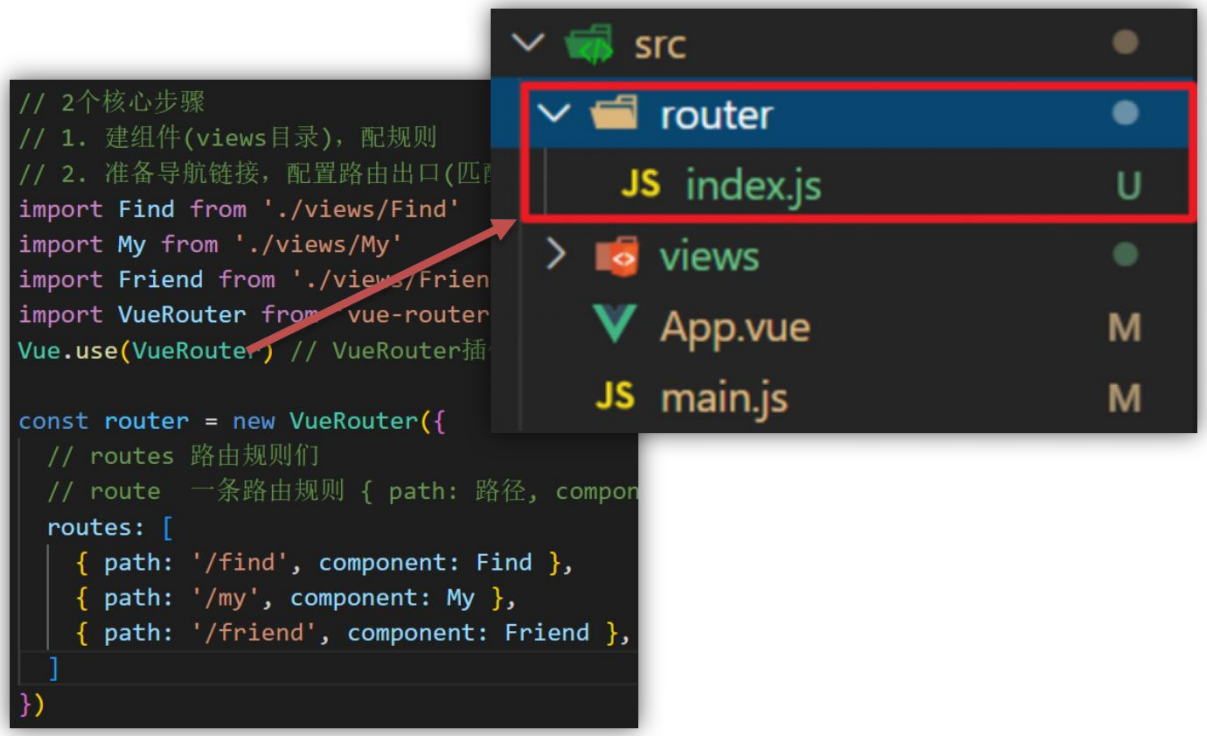
    页面组件 - views文件夹 => 配合路由，页面展示

    复用组件 - components文件夹 => 封装复用

路由的封装抽离

问题：所有的路由配置都堆在main.js中合适么？

目标：将路由模块抽离出来。 好处：拆分模块，利于维护




```
import router from './router/index.js'

new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```

绝对路径：@指代src目录，可以用于快速引入组件

小结

路由模块的封装抽离的好处是什么？

拆分模块，利于维护

以后如何快速引入组件？

基于 @ 指代 src 目录，从 src 目录出发找组件

传智教育旗下高端IT教育品牌