

# Vue 核心技术与实战

## day03



黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌



# 目录

Contents

## ◆ 生命周期

生命周期 & 生命周期四个阶段 / 生命周期钩子 / 生命周期案例

## ◆ 综合案例：小黑记账清单

列表渲染(请求) / 添加 / 删除 / 饼图渲染

## ◆ 工程化开发入门

工程化开发和脚手架 / 项目运行流程 / 组件化 / 组件注册

## ◆ 综合案例：小兔鲜首页

拆分模块-局部注册 / 结构样式完善 / 拆分组件 - 全局注册

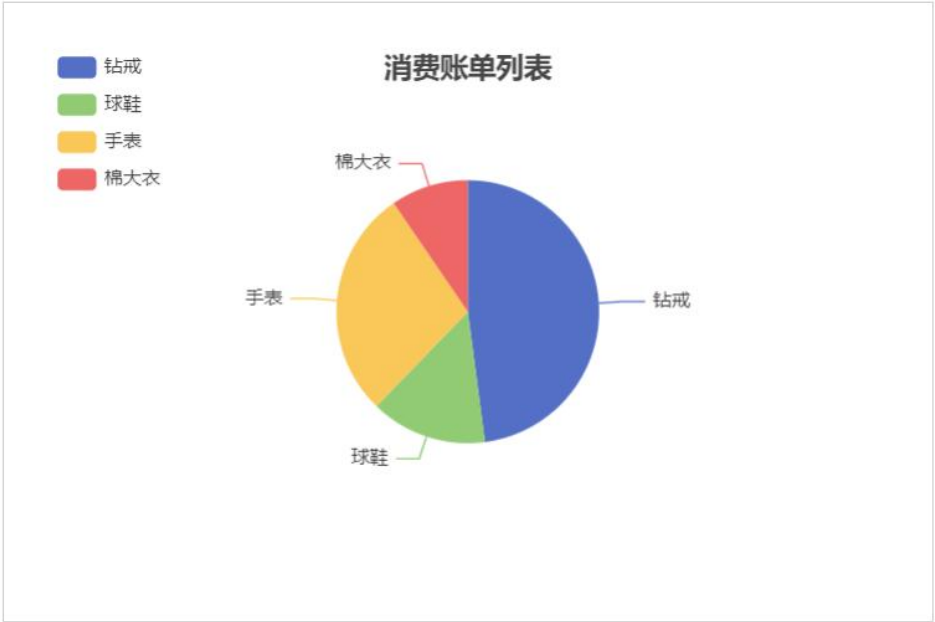
综合案例：小黑记账清单

消费名称

消费价格

添加账单

编号	消费名称	消费价格	操作
1	钻戒	999.00	删除
2	球鞋	299.00	删除
3	手表	588.00	删除
4	棉大衣	199.00	删除
消费总计：2085.00			



## 综合案例：小兔鲜主页 (组件拆分)





# 目录

Contents

## ◆ 生命周期

生命周期 & 生命周期四个阶段 / 生命周期钩子 / 生命周期案例

## ◆ 综合案例：小黑记账清单

列表渲染(请求) / 添加 / 删除 / 饼图渲染

## ◆ 工程化开发入门

工程化开发和脚手架 / 项目运行流程 / 组件化 / 组件注册

## ◆ 综合案例：小兔鲜首页

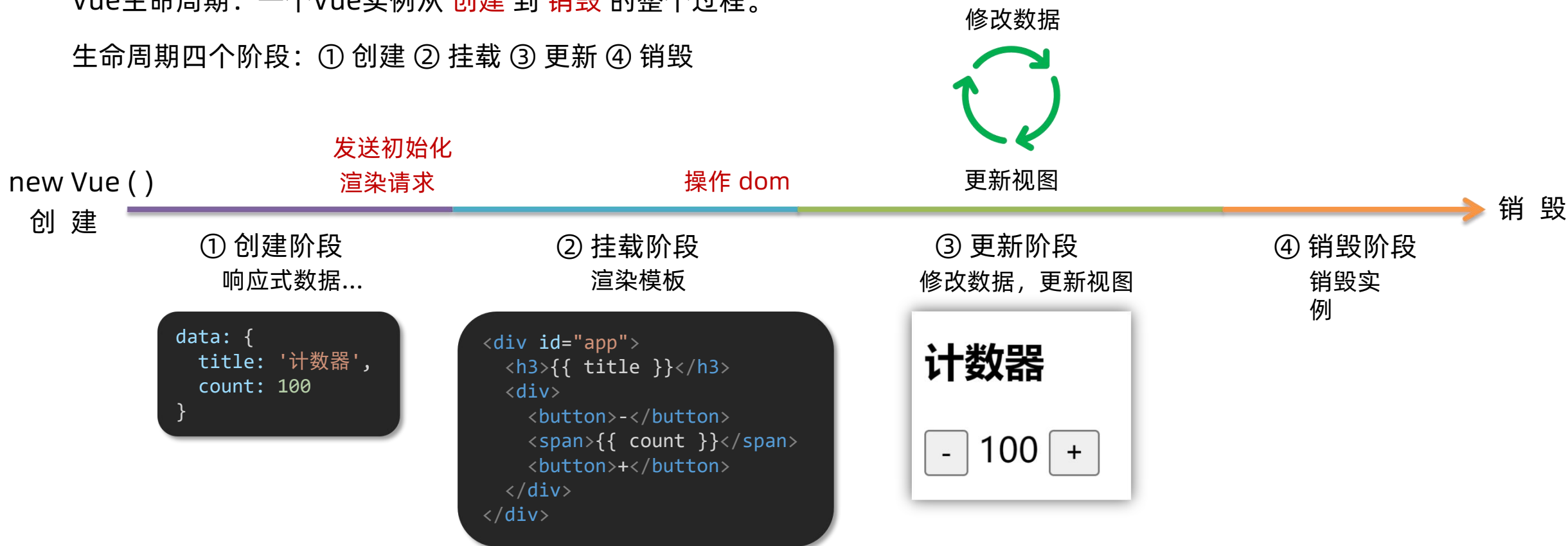
拆分模块-局部注册 / 结构样式完善 / 拆分组件 - 全局注册

## Vue 生命周期 和 生命周期的四个阶段

思考：什么时候可以发送**初始化渲染请求**？（越早越好） 什么时候可以开始**操作dom**？（至少dom得渲染出来）

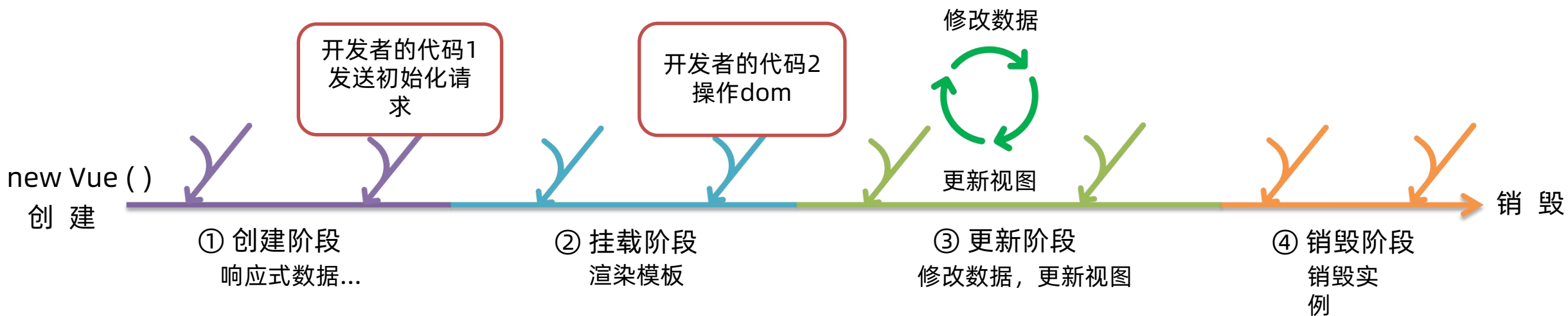
Vue生命周期：一个Vue实例从 **创建** 到 **销毁** 的整个过程。

生命周期四个阶段：① 创建 ② 挂载 ③ 更新 ④ 销毁



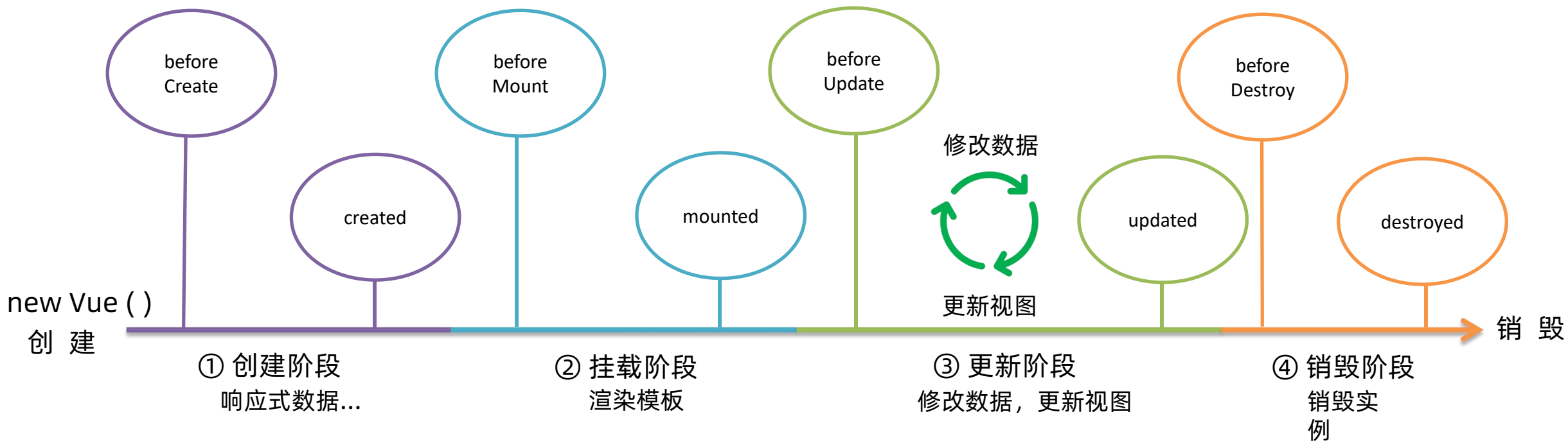
## Vue 生命周期函数（钩子函数）

Vue生命周期过程中，会自动运行一些函数，被称为【生命周期钩子】→ 让开发者可以在【特定阶段】运行自己的代码。



## Vue 生命周期函数（钩子函数）

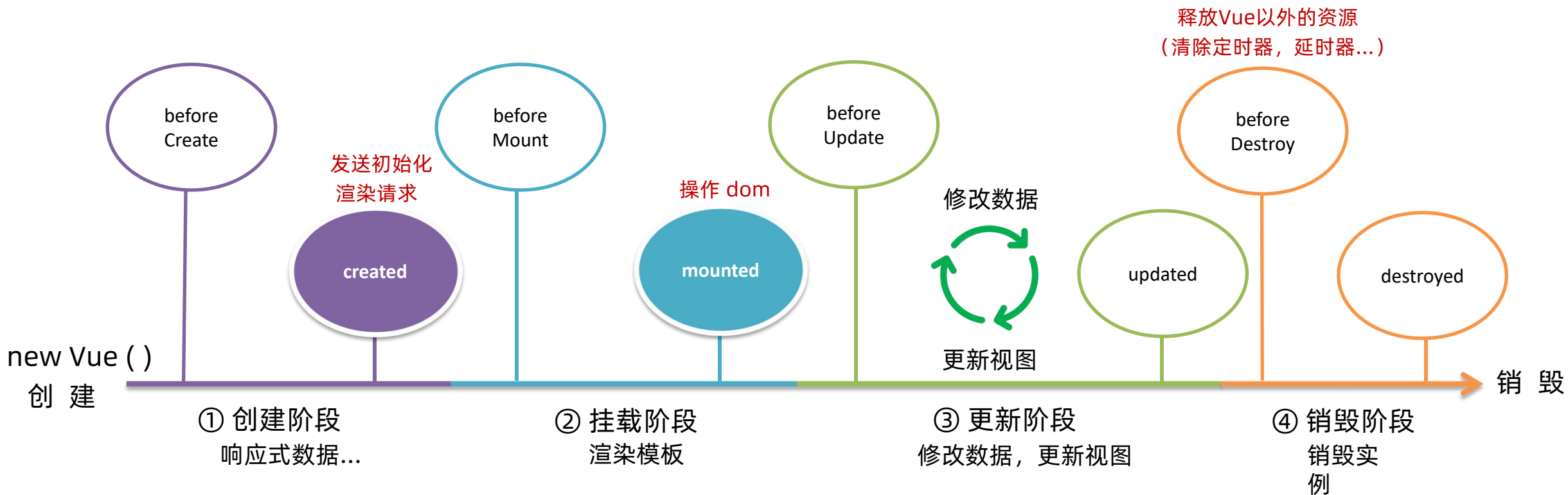
Vue生命周期过程中，会自动运行一些函数，被称为【生命周期钩子】→ 让开发者可以在【特定阶段】运行自己的代码。





## Vue 生命周期函数（钩子函数）

Vue生命周期过程中，会自动运行一些函数，被称为【生命周期钩子】→ 让开发者可以在【特定阶段】运行自己的代码。



## Vue 生命周期钩子案例 - 新闻列表 & 输入框自动聚焦



**mounted** 模板渲染完成，可以开始**操作DOM**了。

**created** 数据准备好了，可以开始发送**初始化渲染请求**。



# 目录

Contents

## ◆ 生命周期

生命周期 & 生命周期四个阶段 / 生命周期钩子 / 生命周期案例

## ◆ 综合案例：小黑记账清单

列表渲染(请求) / 添加 / 删除 / 饼图渲染

## ◆ 工程化开发入门

工程化开发和脚手架 / 项目运行流程 / 组件化 / 组件注册

## ◆ 综合案例：小兔鲜首页

拆分模块-局部注册 / 结构样式完善 / 拆分组件 - 全局注册

小黑记账清单

消费名称

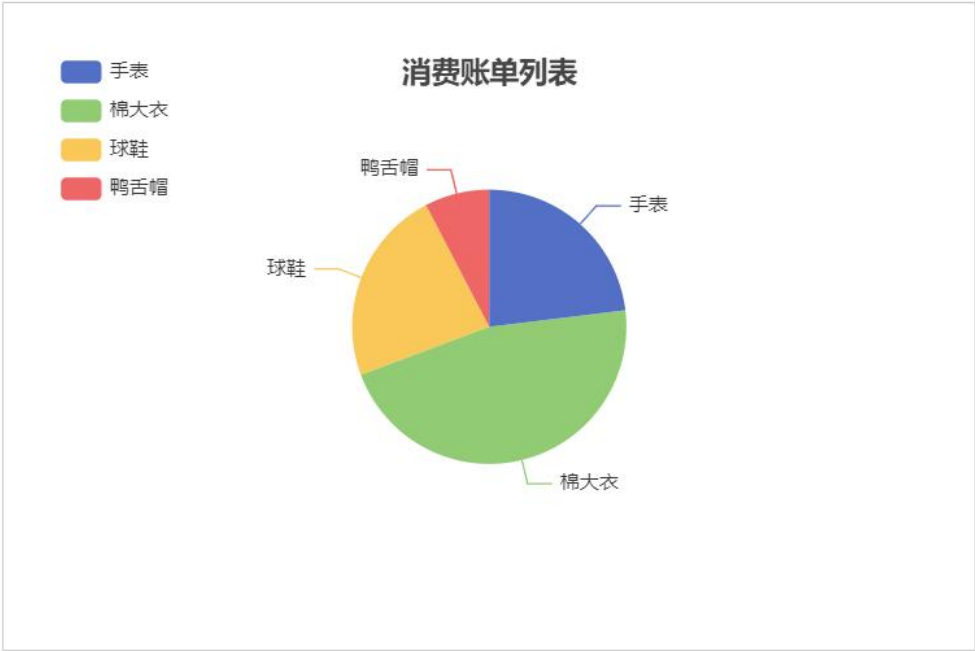
消费价格

添加账单

编号	消费名称	消费价格	操作
1	手表	299.00	删除
2	棉大衣	599.00	删除
3	球鞋	299.00	删除
4	鸭舌帽	99.00	删除
消费总计: 1296.00			

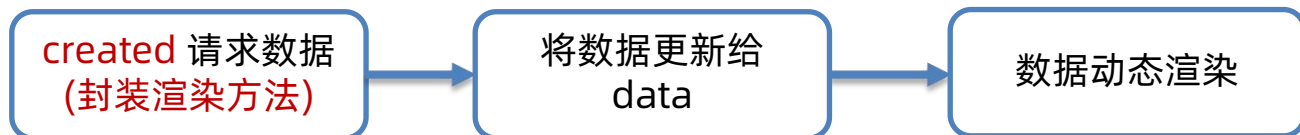
功能需求：

- 1. 基本渲染
- 2. 添加功能
- 3. 删除功能
- 4. 饼图渲染

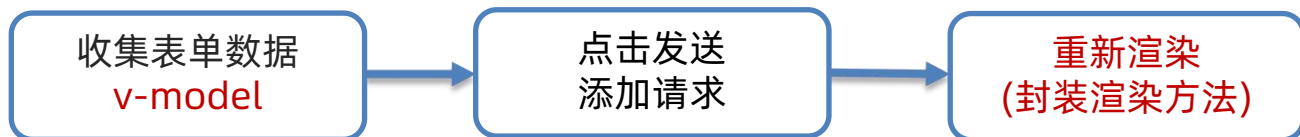


## 案例总结：

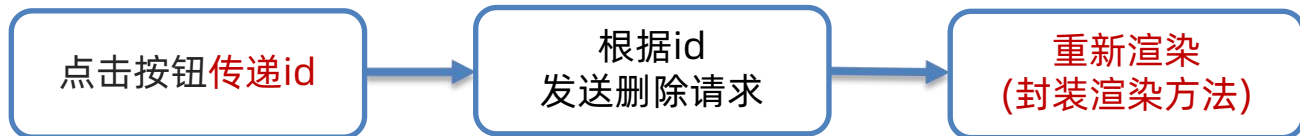
### 1. 基本渲染



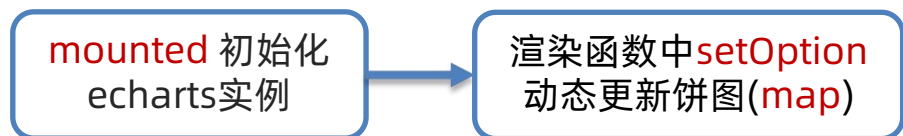
### 2. 添加功能



### 3. 删除功能



### 4. 饼图渲染





# 目录

Contents

## ◆ 生命周期

生命周期 & 生命周期四个阶段 / 生命周期钩子 / 生命周期案例

## ◆ 综合案例：小黑记账清单

列表渲染(请求) / 添加 / 删除 / 饼图渲染

## ◆ 工程化开发入门

工程化开发和脚手架 / 项目运行流程 / 组件化 / 组件注册

## ◆ 综合案例：小兔鲜首页

拆分模块-局部注册 / 结构样式完善 / 拆分组件 - 全局注册

## 工程化开发 & 脚手架 Vue CLI

### 开发 Vue 的两种方式：

1. 核心包传统开发模式：基于 html / css / js 文件，直接引入核心包，开发 Vue。
2. 工程化开发模式：基于构建工具（例如：webpack）的环境中开发 Vue。



### 问题：

- ① webpack 配置不简单
- ② 雷同的基础配置
- ③ 缺乏统一标准

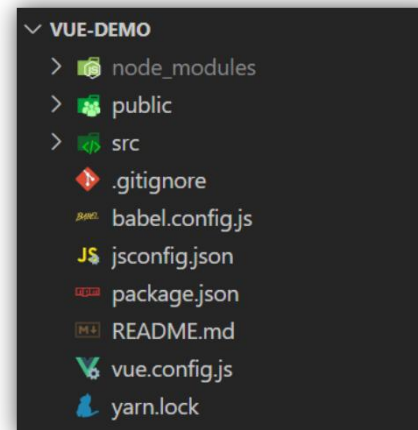
需要一个工具，生成标准化的配置！

## 工程化开发 & 脚手架 Vue CLI

### 基本介绍:

Vue CLI 是 Vue 官方提供的一个**全局命令工具**。

可以帮助我们**快速创建**一个开发 Vue 项目的**标准化基础架子**。【集成了 webpack 配置】



### 好处:

1. 开箱即用，零配置
2. 内置 babel 等工具
3. 标准化

### 使用步骤:

1. 全局安装 (一次): `yarn global add @vue/cli` 或 `npm i @vue/cli -g`
2. 查看 Vue 版本: `vue --version`
3. 创建项目架子: `vue create project-name` (项目名-不能用中文)
4. 启动项目: `yarn serve` 或 `npm run serve` (找package.json)



## 脚手架目录文件介绍 & 项目运行流程

### VUE-DEMO

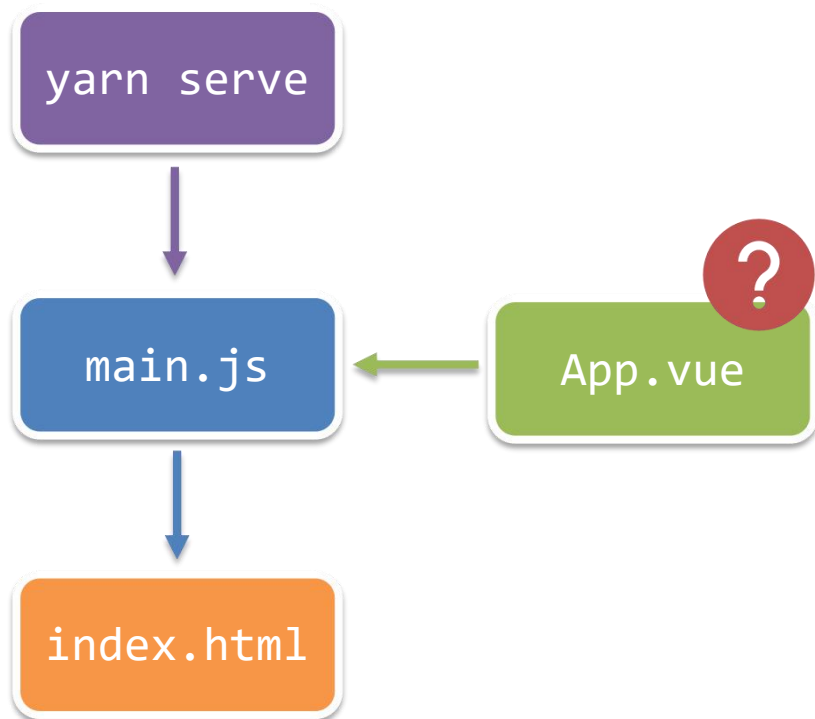
—node_modules	第三包文件夹
—public	放html文件的地方
—favicon.ico	网站图标
—index.html	index.html 模板文件 ③
—src	源代码目录 → 以后写代码的文件夹
—assets	静态资源目录 → 存放图片、字体等
—components	组件目录 → 存放通用组件
—App.vue	App根组件 → 项目运行看到的内容就在这里编写 ②
—main.js	入口文件 → 打包或运行，第一个执行的文件 ①
—.gitignore	git忽视文件
—babel.config.js	babel配置文件
—jsconfig.json	js配置文件
—package.json	项目配置文件 → 包含项目名、版本号、scripts、依赖包等
—README.md	项目说明文档
—vue.config.js	vue-cli配置文件
—yarn.lock	yarn锁文件，由yarn自动生成的，锁定安装版本

## 脚手架目录文件介绍 & 项目运行流程

### VUE-DEMO

—node_modules	第三包文件夹
—public	放html文件的地方
—favicon.ico	网站图标
—index.html	index.html 模板文件 ③
—src	源代码目录 → 以后写代码的文件夹
—assets	静态资源目录 → 存放图片、字体等
—components	组件目录 → 存放通用组件
—App.vue	App根组件 → 项目运行看到的内容就在这里编写 ②
—main.js	入口文件 → 打包或运行，第一个执行的文件 ①
—.gitignore	git忽视文件
—babel.config.js	babel配置文件
—jsconfig.json	js配置文件
—package.json	项目配置文件 → 包含项目名、版本号、scripts、依赖包等
—README.md	项目说明文档
—vue.config.js	vue-cli配置文件
—yarn.lock	yarn锁文件，由yarn自动生成的，锁定安装版本

## 脚手架目录文件介绍 & 项目运行流程



### main.js 核心代码

#### 1. 导入 Vue

```
• import Vue from 'vue'
```

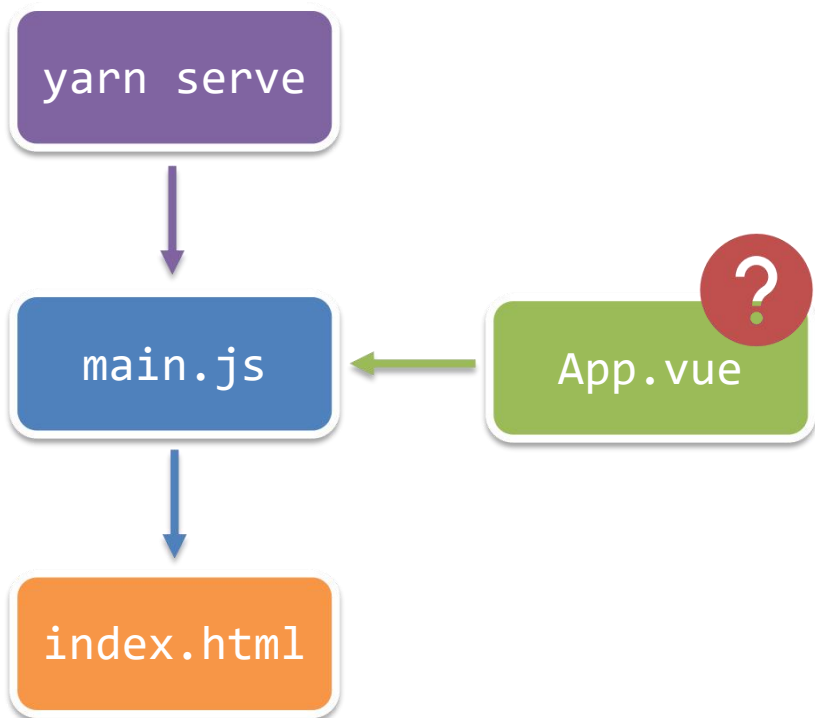
#### 2. 导入 App.vue

```
• import App from './App.vue'
```

#### 3. 实例化 Vue，将 App.vue 渲染到 index.html 容器中

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

## 组件化开发 & 根组件



## 组件化开发 & 根组件

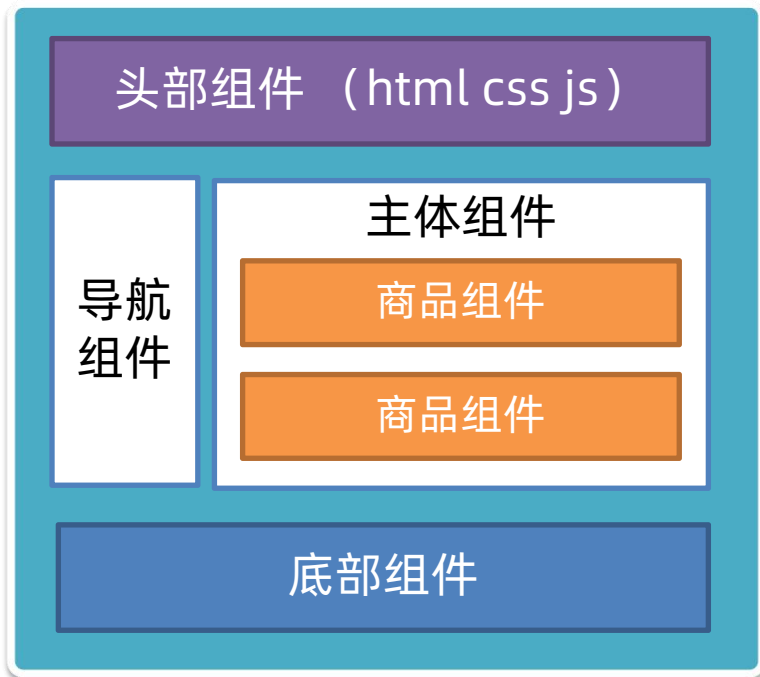
① **组件化**：一个页面可以拆分成一个个组件，每个组件有着自己独立的结构、样式、行为。

好处：便于维护，利于复用 → 提升开发效率。

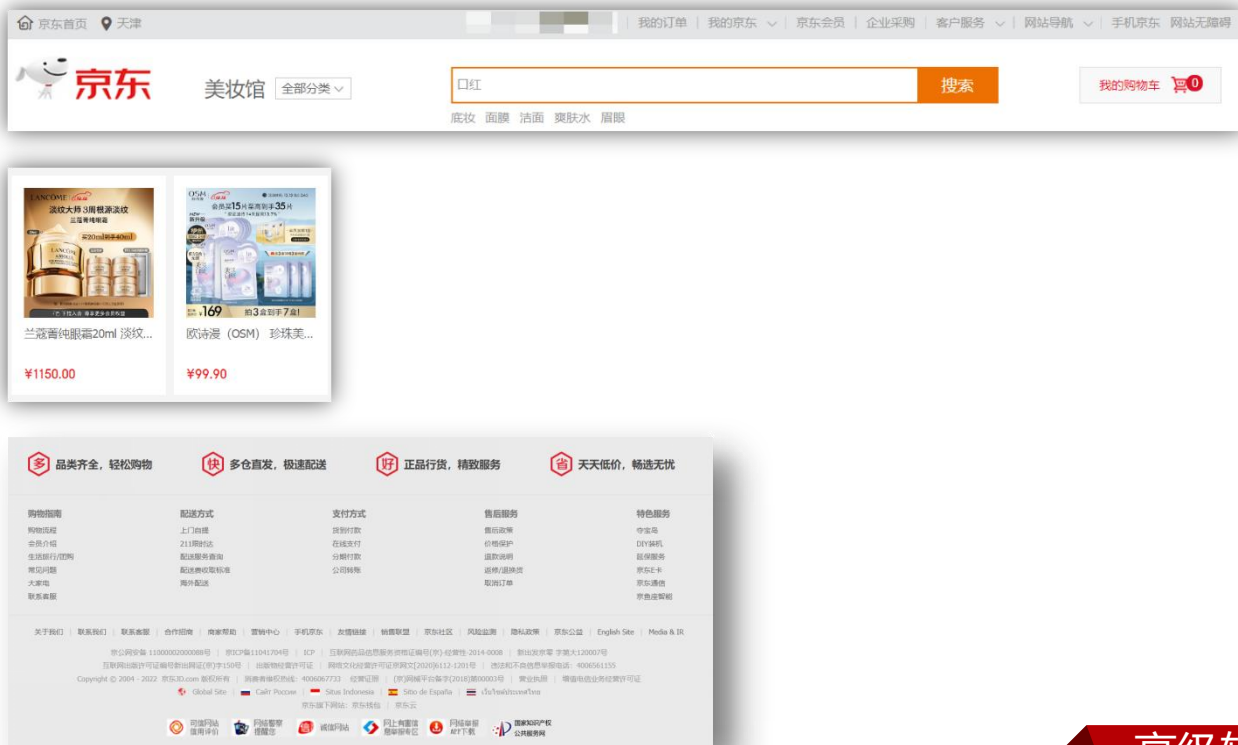
组件分类：普通组件、根组件。

② **根组件**：整个应用最上层的组件，包裹所有普通小组件。

App.vue



App.vue 根组件



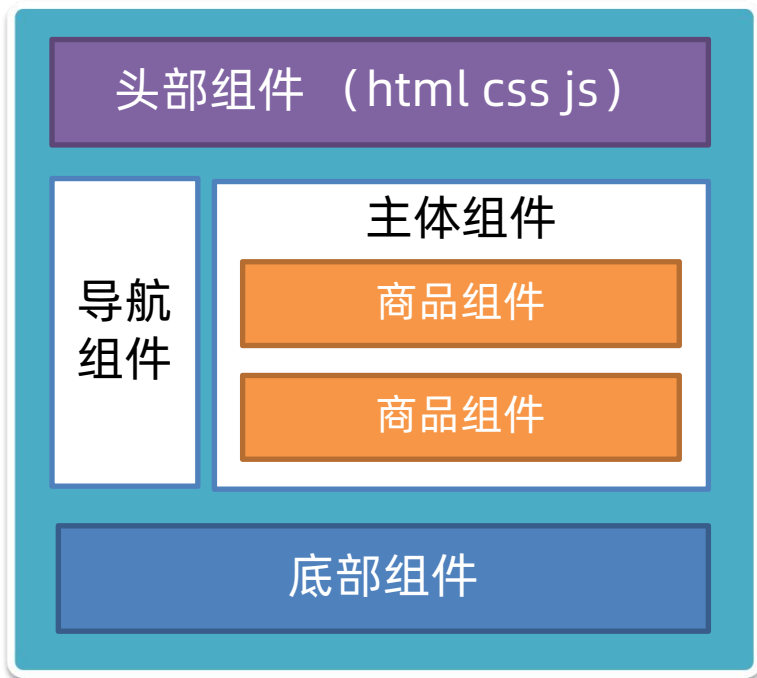
## 组件化开发 & 根组件

① **组件化**：一个页面可以拆分成一个个**组件**，每个组件有着自己独立的**结构、样式、行为**。

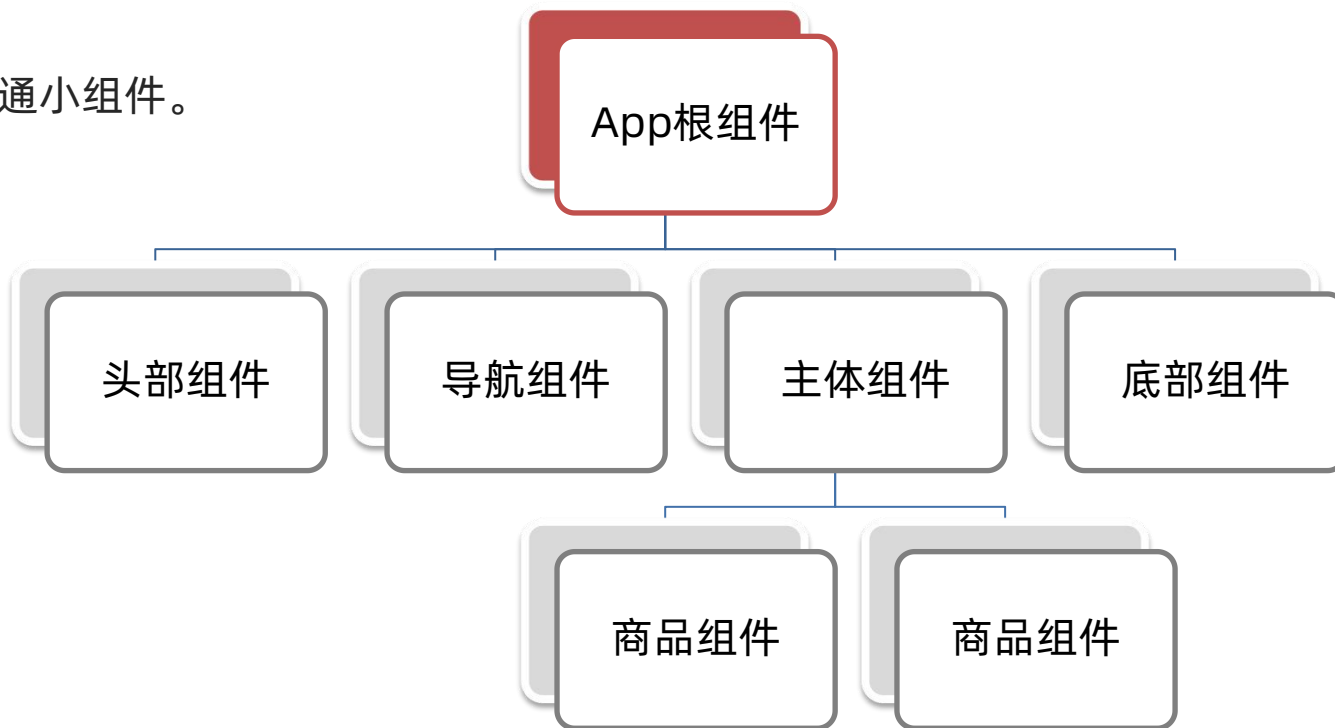
好处：便于**维护**，利于**复用** → 提升**开发效率**。

组件分类：普通组件、根组件。

② **根组件**：整个应用最上层的组件，包裹所有普通小组件。



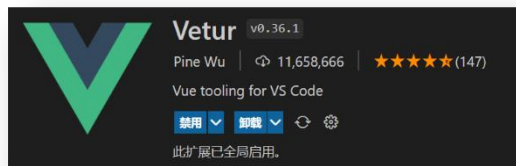
App.vue 根组件



组件树

## App.vue 文件（单文件组件）的三个组成部分

### 1. 语法高亮插件：

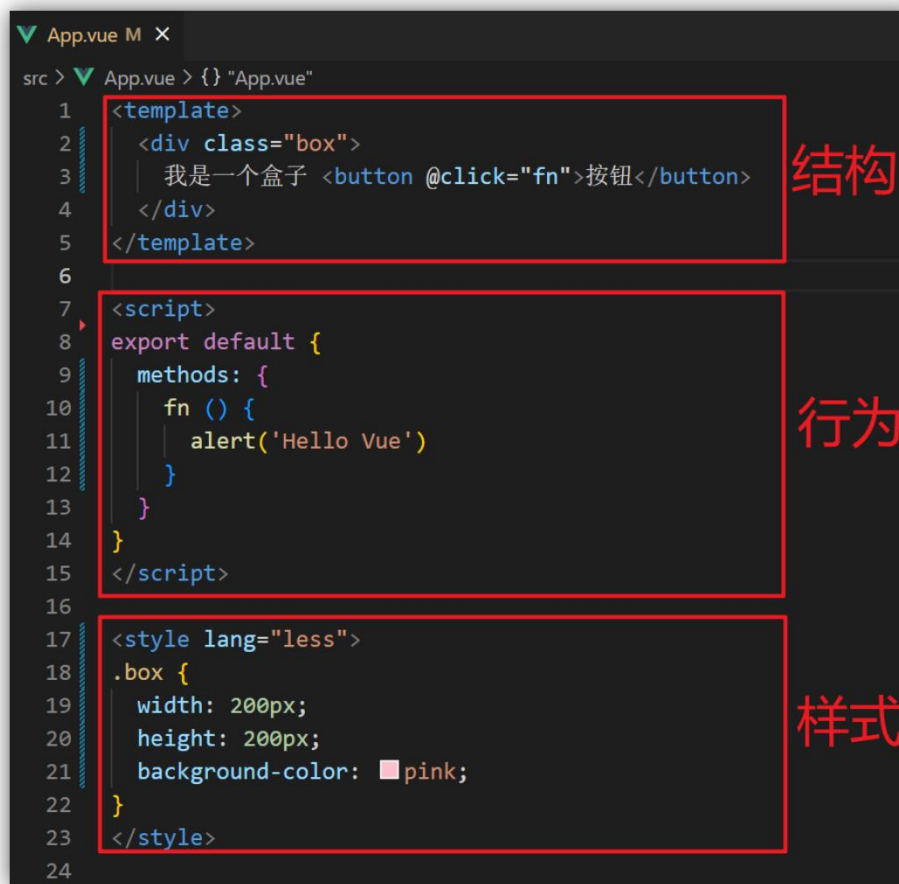


### 2. 三部分组成：

- ◆ template: 结构（有且只能一个根元素）
- ◆ script: js逻辑
- ◆ style: 样式（可支持less，需要装包）

### 3. 让组件支持 less

- (1) style标签, lang="less" 开启less功能
- (2) 装包: `yarn add less less-loader`

The image shows a code editor window titled 'App.vue M x' with the following code:

```
src > App.vue > {} "App.vue"
1  <template>
2    <div class="box">
3      我是一个盒子 <button @click="fn">按钮</button>
4    </div>
5  </template>
6
7  <script>
8    export default {
9      methods: {
10        fn () {
11          alert('Hello Vue')
12        }
13      }
14    }
15  </script>
16
17  <style lang="less">
18    .box {
19      width: 200px;
20      height: 200px;
21      background-color: pink;
22    }
23  </style>
24
```

Red boxes highlight the three sections of the code, with red text labels to their right: '结构' (Structure) for the template section, '行为' (Behavior) for the script section, and '样式' (Style) for the style section.



# 总结

## (1) 组件化：

页面可拆分成一个个组件，每个组件有着独立的结构、样式、行为

① 好处：便于维护，利于复用 → 提升开发效率。

② 组件分类：普通组件、根组件。

## (2) 根组件：

整个应用最上层的组件，包裹所有普通小组件。

一个根组件App.vue，包含的三个部分：

① **template** 结构 (只能有一个根节点)

② **style** 样式 (可以支持less，需要装包 less 和 less-loader )

③ **script** 行为



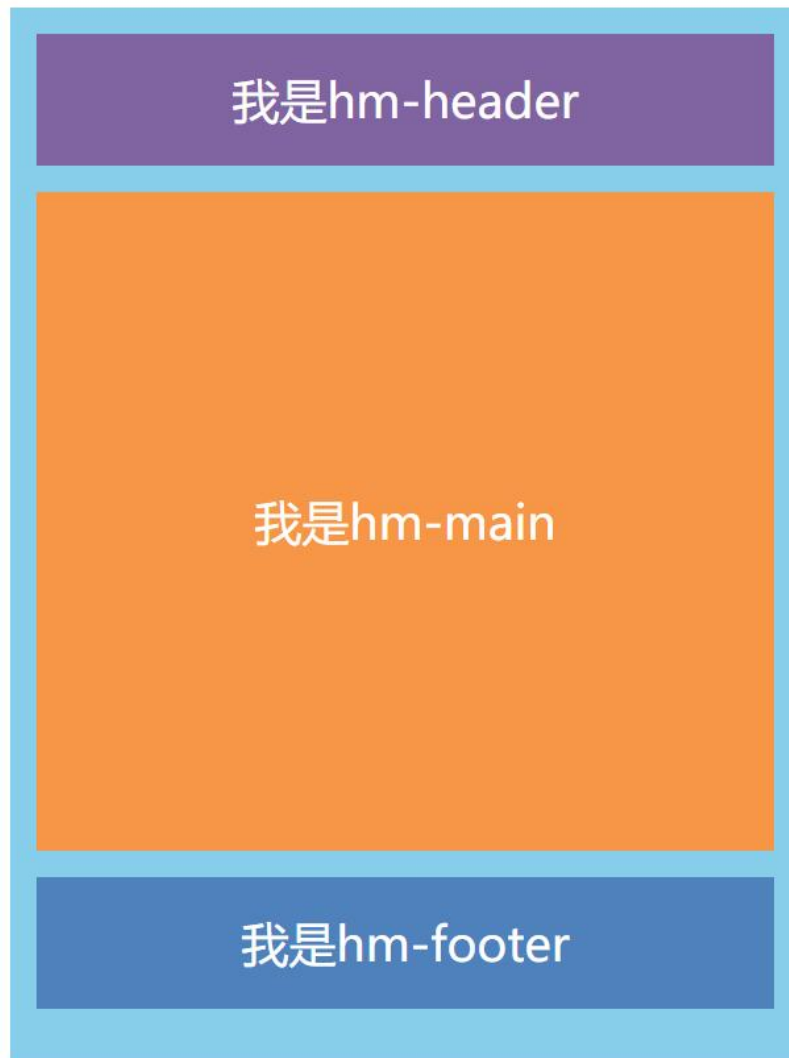
## 普通组件的注册使用

### 组件注册的两种方式：

#### 1. 局部注册：只能在注册的组件内使用

- ① 创建 .vue 文件 (三个组成部分)
- ② 在使用的组件内导入并注册

#### 2. 全局注册：所有组件内都能使用



**App.vue 根组件**

## 普通组件的注册使用

### 组件注册的两种方式：

#### 1. 局部注册：只能在注册的组件内使用

① 创建 .vue 文件 (三个组成部分)

② 在使用的组件内导入并注册

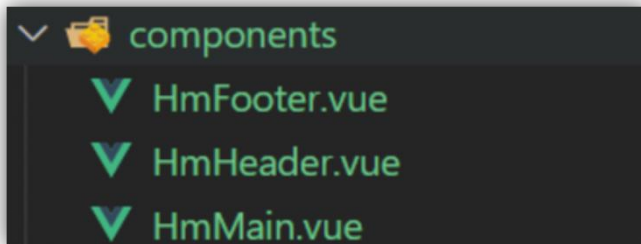
#### 2. 全局注册：所有组件内都能使用

### 使用：

◆ 当成 html 标签使用 `**<组件名></组件名>**`

### 注意：

◆ 组件名规范 → 大驼峰命名法，如：HmHeader



```
// 导入需要注册的组件
import 组件对象 from '.vue文件路径'
import HmHeader from './components/HmHeader'

export default {
  // 局部注册
  components: {
    '组件名': 组件对象,
    HmHeader: HmHeader
  }
}
```

**App.vue**

## 普通组件的注册使用

### 组件注册的两种方式：

#### 1. 局部注册：只能在注册的组件内使用

① 创建 .vue 文件 (三个组成部分)

② 在使用的组件内导入并注册

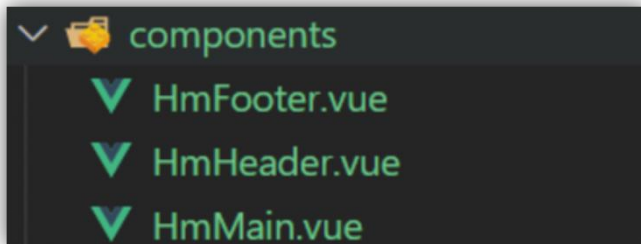
#### 2. 全局注册：所有组件内都能使用

### 使用：

◆ 当成 html 标签使用 `**<组件名></组件名>**`

### 注意：

◆ 组件名规范 → 大驼峰命名法，如：HmHeader



```
// 导入需要注册的组件
import 组件对象 from '.vue文件路径'
import HmHeader from './components/HmHeader'

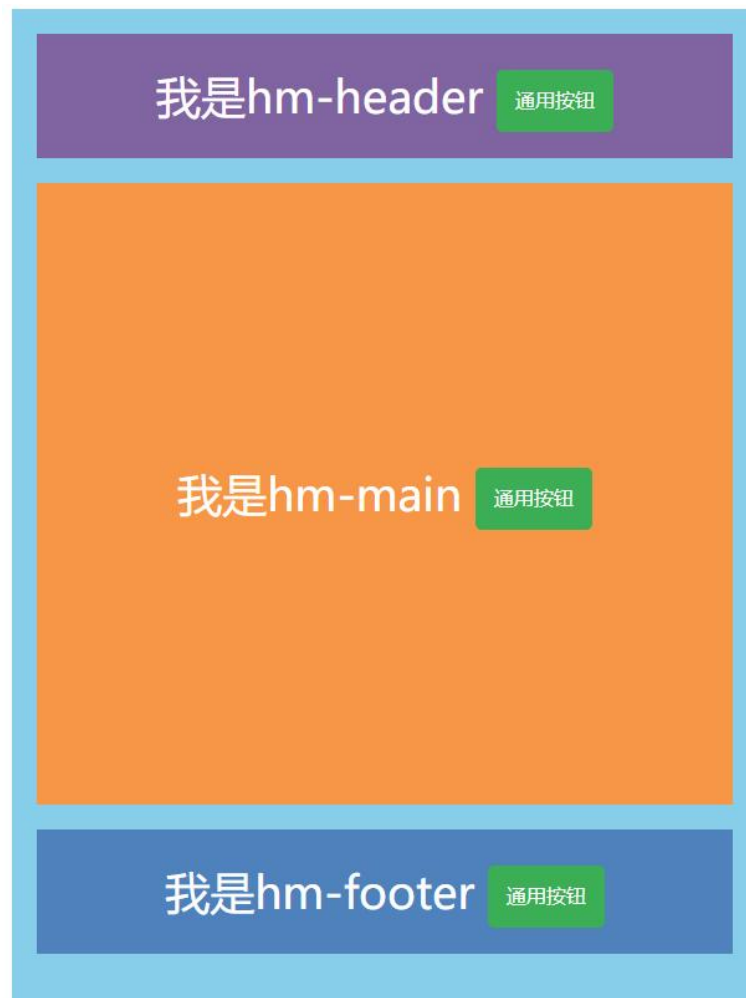
export default {
  // 局部注册
  components: {
    '组件名': 组件对象,
    HmHeader
  }
}
```

**App.vue**

## 普通组件的注册使用

### 组件注册的两种方式：

1. 局部注册：只能在注册的组件内使用
2. 全局注册：所有组件内都能使用
  - ① 创建 .vue 文件 (三个组成部分)
  - ② `main.js` 中进行全局注册



## 普通组件的注册使用

### 组件注册的两种方式：

1. 局部注册：只能在注册的组件内使用
2. 全局注册：所有组件内都能使用

- ① 创建 .vue 文件 (三个组成部分)
- ② main.js 中进行全局注册

### 使用：

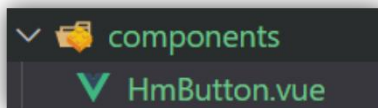
- ◆ 当成 html 标签使用 `**<组件名></组件名>**`

### 注意：

- ◆ 组件名规范 → 大驼峰命名法，如：HmHeader

### 技巧：

- ◆ 一般都用**局部注册**，如果发现确实是**通用组件**，再定义到全局。



```
// 导入需要全局注册的组件
import HmButton from './components/HmButton'

// 调用 Vue.component 进行全局注册
// Vue.component('组件名', 组件对象)
Vue.component('HmButton', HmButton)
```

main.js



# 总结

## 普通组件的注册使用：

### 1. 两种注册方式：

#### ① 局部注册：

(1) 创建.vue组件 (单文件组件)

(2) 使用的组件内导入，并局部注册 `components: { 组件名: 组件对象 }`

#### ② 全局注册：

(1) 创建.vue组件 (单文件组件)

(2) `main.js`内导入，并全局注册 `Vue.component(组件名, 组件对象)`

### 2. 使用：

`<组件名></组件名>`

## 技巧：

一般都用局部注册，如果发现确实是通用组件，再抽离到全局。



# 目录

Contents

## ◆ 生命周期

生命周期 & 生命周期四个阶段 / 生命周期钩子 / 生命周期案例

## ◆ 综合案例：小黑记账清单

列表渲染(请求) / 添加 / 删除 / 饼图渲染

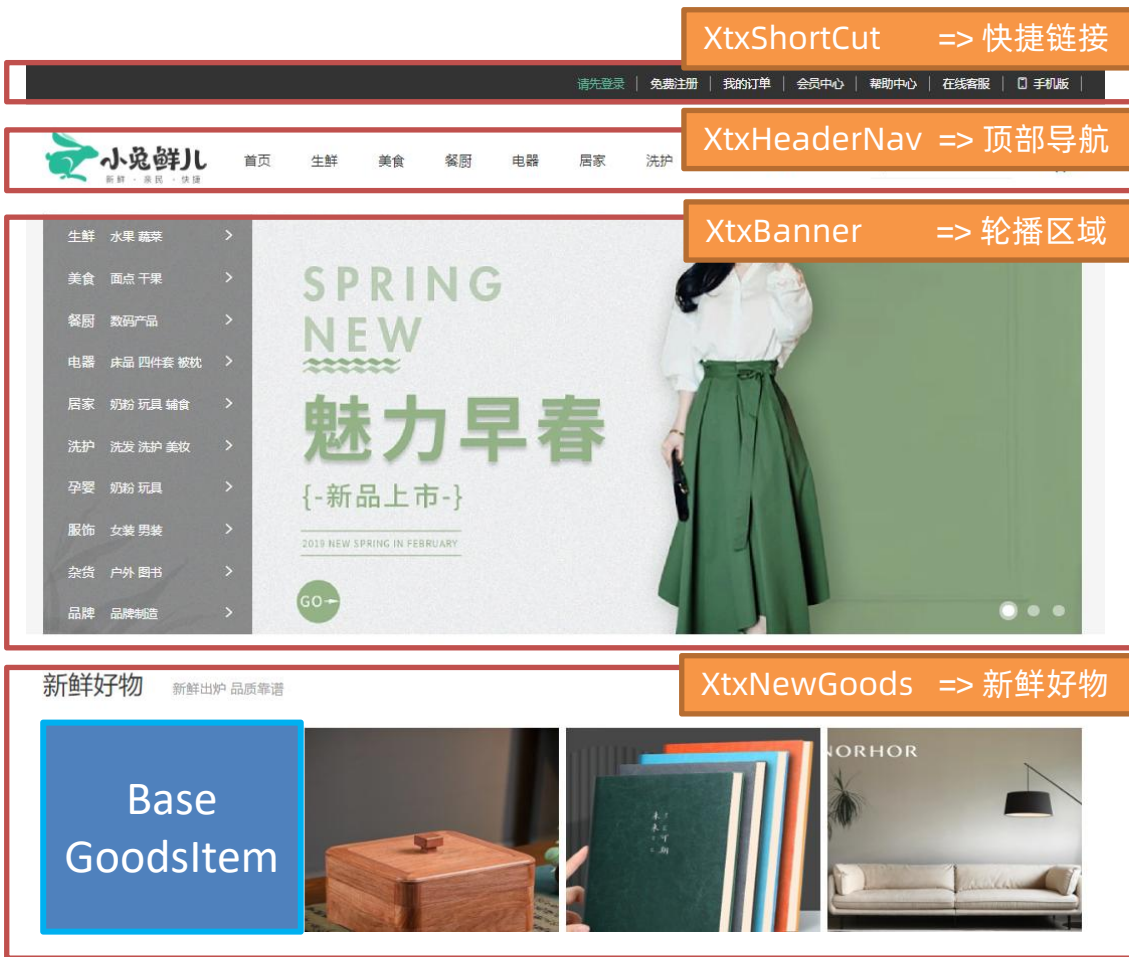
## ◆ 工程化开发入门

工程化开发和脚手架 / 项目运行流程 / 组件化 / 组件注册

## ◆ 综合案例：小兔鲜首页

拆分模块-局部注册 / 结构样式完善 / 拆分组件 - 全局注册

## 综合案例 - 小兔鲜首页 - 组件拆分



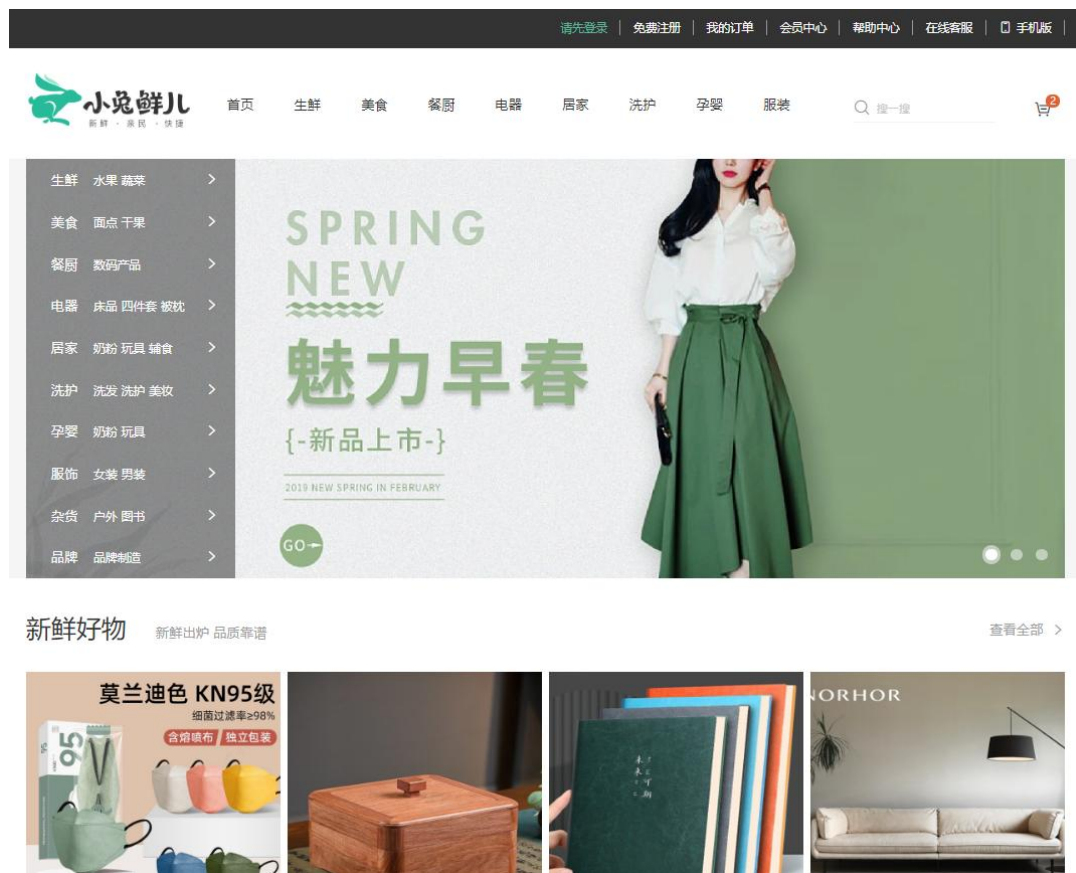


## 综合案例 - 小兔鲜首页 - 组件拆分

页面开发思路：

1. 分析页面，按模块拆分子组件，搭架子（局部或全局注册）
2. 根据设计图，编写组件 html 结构 css 样式（已准备好）
3. 拆分封装通用小组件（局部或全局注册）

将来 → 通过 js 动态渲染，实现功能





传智教育旗下高端IT教育品牌