

# 加餐 | 实战篇思考题答疑（上）

2023-02-01 杨文坚 来自北京

天下无鱼  
<https://shikey.com/>

《Vue 3 企业级项目实战课》

课程介绍 >



讲述：杨文坚

时长 01:38 大小 1.49M



你好，我是杨文坚。

实战篇学到现在，我们已经从前端到后端，基于 Vue.js 和 Node.js，实现了一套全栈服务功能，这些功能覆盖了运营搭建平台的基本功能链路。

我们首先定制了运营拖拽组件，承接基础组件库的开发，也开启了运营搭建业务功能实现，为后面实战的高阶开发做准备。

接着，我们基于 Koa.js 开发 Node.js Web 服务，同时结合基于 MySQL 的项目数据库设计，打通了项目平台服务的全栈链路。通过这个全栈 Node.js Web 服务开发，我们扩展出面向企业用户管理的后台服务和面向客户展示页面的前台服务，核心的技术思路是做全栈服务的技术分层，以及面向不同场景的服务项目拆分。

然后，我们从业务视角，划分了运营搭建平台的功能维度，分别是用户、物料和页面这三大功能维度。我们基于搭建好的全栈服务底座，实现了用户和物料两大维度的基础业务功能，核心的技术思路是数据库的“增删改”的操作，以及前端静态资源的共享管理。



今天我们对已经学过的实战篇内容做一次统一答疑，如果还有疑问，也欢迎你留言。

## 15

课程：🔗 [15 定制运营拖拽组件](#)

提问：我们实现了拖拽布局组件，只是通过拖拽调整了布局，那么，如果要从一个布局拖拽到另外一个布局里，应该怎么实现这个“拖放”布局布局组件呢？

拖放操作，其实跟拖拽操作类似，技术实现逻辑只有两个步骤。

- 第一步，要确定“可拖的模块”和“可放的目标区域”。
- 第二步，注册“拖”和“放”的 DOM 事件。

第一步，“可拖的模块”需要设置 `draggable` 属性。“可放的目标区域”不需要注册特殊属性，只需在第二步注册“监听放”的事件。

第二步，“可拖的模块”注册拖拽 `ondrapstart` 事件，“可放的目标区域”需要注册 `ondrop` 事件，就是监听“拖放”的事件。同时要监听 `ondrapover` 事件，就是监听“可拖的模块”是否到达“可放”的区域中。

这里我实现了一个代码案例。

📄 复制代码

```
1 <html>
2   <head>
3     <meta charset="utf8" />
4     <style>
5       .area {
6         width: 400px; height: 80px; background: #f0f0f0; border: 1px solid #e7e
7       }
8       .area-target {
9         background: #f0a49e;
10      }
```

```

11     .module {
12         width: 100px; height: 40px; background: #bddef9; border: 1px solid #222
13     }
14 </style>
15 </head>
16 <body>
17
18     <div class="area">
19         <div>原始区域</div>
20         <div id="module" draggable="true" ondragstart="dragModuleStart(event)" cl
21             可拖放模块
22         </div>
23     </div>
24
25     <div id="target-area" ondrop="dropArea(event)" ondragover="dragOverArea(eve
26         <div>目标区域</div>
27         <!-- 模块可以拖方这里 -->
28     </div>
29
30 </body>
31 <script>
32     function dragModuleStart(event) {
33         // 将拖拽模块元素的 id 添加到数据传输对象
34         event.dataTransfer.setData("application/my-app", event.target.id);
35         event.dataTransfer.dropEffect = "move";
36     }
37     function dragOverArea(event) {
38         // 处理拖拽模块到目标区域时候，目标区域的拖拽悬浮事件
39         event.preventDefault();
40         event.dataTransfer.dropEffect = "move"
41     }
42     function dropArea(event) {
43         event.preventDefault();
44         // 获取拖放模块的id， 并将移动的元素添加到目标的 DOM
45         const data = event.dataTransfer.getData("application/my-app");
46         event.target.appendChild(document.getElementById(data));
47     }
48 </script>
49
50 </html>

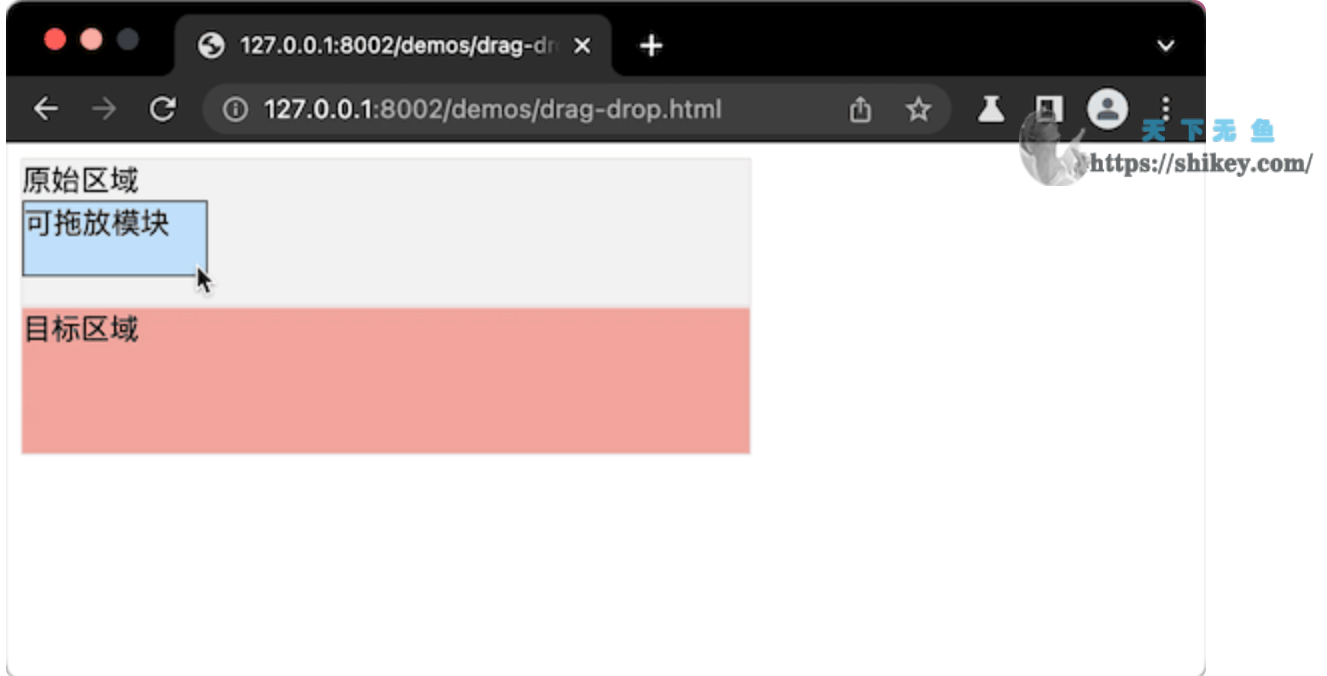
```



天下无鱼

<https://shikey.com/>

代码案例的最终实现效果图，就像这样：



## 16

课程：🔗 [16 开发单页面应用](#)

提问：如何处理单页面和多页面项目共存？

要让单页面（应用）和多页面在同一个项目中共存，需要前端和后端一起配合实现，具体实现可以分成三步。

第一步，前端和后端要约定单页面和多页面的路径规范。例如约定 `/page/mange/` 路径是单页面路径，凡是 URL 路径以 `/page/mange/` 开头的都是单页面应用。剩下 `/page/xxx` 都是多页面路径。

第二步，后端路由分别处理单页面和多页面的服务端渲染。服务端路由判断请求 URL 路径是以 `/page/mange/` 开头的，就渲染同一个单页面的 HTML，其它多页面路径就渲染对应页面的 HTML。

第三步，前端路由处理单页面渲染。前端对页面为 `/page/mange/` 的路径，用 hash 或者 history 的前端浏览器路由，处理对应的单页面操作。

总结一下，单页面（应用）和多页面共存，不是一个单纯技术问题，更多是一个全栈项目的前端后端的规范约定问题。

提问: Koa.js 和 Express.js 的中间件模型有什么区别吗?

Koa.js 和 Express.js 的中间件模型是不一样的。Koa.js 是基于 Promise 的嵌套实现“洋葱中间件模型”，Express.js 是基于“回调”的嵌套来实现中间件模型。

Koa.js 的中间件模型，通过 Promise 的嵌套，处理实现了洋葱模型中间件。不管怎么“use”中间件，最后都是结构成一层层 Promise 的嵌套，大致解构出来代码就像这样。

[📄 复制代码](#)

```
1 // Koa.js中间件的使用
2 app.use(middleware1);
3 app.use(middleware2);
4 app.use(middleware3);
5
6 // 可以解构类比成
7 Promise.resolve(
8   middleware1(context, async () => {
9     return Promise.resolve(
10       middleware2(context, async () => {
11         return Promise.resolve(
12           middleware3(context, async () => {
13             return Promise.resolve();
14           })
15         );
16       })
17     );
18   })
19 ).then(() => {});
```

Express.js 通过“回调方法”的嵌套形成中间件数组，再自上而下“线性”执行中间件，通过 next 这个方法来控制“线性”的过程。看大致解构出来代码。

[📄 复制代码](#)

```
1 // Express.js中间件的使用
2
3 app.use((req, res, next)=> {
4   console.log('001');
5   next();
6   console.log('004');
```

```
7 });
8 app.use((req, res, next)=> {
9   console.log('002');
10  next();
11  console.log('003');
12 });
13
14 // 可以解构成以下代码
15 app.use((req, res, next)=>{
16   console.log('001');
17   ((req, res, next)=>{
18     console.log('002');
19     // 其它中间件
20     console.log('003');
21   })()
22   console.log('004');
23 });
```



## 18

课程: [🔗 18 Node.js 服务端渲染页面](#)

提问: 单页面应用如何优雅设计 Vue.js 项目的 SSR 和 CSR?

多页面的 SSR 和 CSR 结合, 主要是从编译视角做功夫, 把一套 Vue.js 的前端代码, 稍做编译处理, 输出另一份代码, 来支持 Node.js 服务环境渲染 HTML。那么单页面应用, 也可以遵循这个思路来低成本地优雅实现。

在单页面应用中, 每个子页面都是就是基于 CSR 进行修改路由, 然后渲染的。这时候需要 SSR 的出现场景, 就只是“刷新”或“首次”加载子页面时候, 在服务端输出子页面的 HTML。

如果单页面应用要支持 SSR, 那就需要支持每个“子页面”的服务渲染 HTML。这时候最低成本的方式, 或者说是最优雅的方式, 就是从编译视角做修改。


也就是一套 Vue.js 的单页面应用代码, 在编译时候做处理, 输出每个子页面的 Node.js 代码, 在 Node.js 服务对应子页面路由上做处理。

## 19

课程: [🔗 19 搭建 Vue.js 的前后台全栈项目](#)



提问：为什么不能用读写静态文件的方式来代替数据库使用？

用静态文件，理论上是可以当做“数据库”使用的，就当做是用一个 Excel 文件来管理数据。

如果用静态文件管理数据，那就需要自己写一堆服务端代码实现数据的“新增”、“搜索”、“更新”的操作支持。

但是，数据量积累到一定量的程度，“搜索”数据的代码将会遇到性能瓶颈，这时如果要实现数据“更新”，必须基于“搜索”来找到数据，然后做“更新”操作，将会叠加了性能瓶颈问题。此时，操作静态文件的代码，将会大大增加维护成本，甚至做性能优化都无从下手。

数据库的出现，其实就是解决了数据的“管理”问题，一般会内置一个或者多个“引擎”来处理数据的“搜索”操作。而且数据库是独立的服务，独立处理服务器里资源开销问题。如果自己读写静态文件来管理数据，必将占用同一个服务的资源（例如内存和 CPU）。

当然，数据库也不是处处适用，如果实际需求只需要管理少量数据，用静态文件的读写作为数据管理反而更省事。因为数据库是需要独立开启服务的，也需要运维成本的。

所以，如果数据量极少时候，可以用静态文件的读写方式来管理数据。如果数据量多了，就尽量选择用数据库来管理数据。

## 20

课程：[🔗 20 设计运营搭建平台的数据库](#)

提问：如何用 SQL 实现数据库联表查询操作？例如，联合员工用户表、物料信息表和物料快照表，查询具体用户的操作记录日志。

可以通过“JOIN”的方式来联表查询。我们先用 SQL 来插入模拟数据。

 复制代码

```
1 -- 插入模拟用户数据
2 INSERT INTO user_info
3     (uuid, username, password, status, info)
4 VALUES
5     (
6         '00000000-aaaa-bbbb-cccc-ddddeeee0001',
```



```
7      'admin001',
8      '1f22f6ce6e58a7326c5b5dd197973105',
9      1,
10     '{}',
11   ),
12   (
13     '00000000-aaaa-bbbb-cccc-ddddeeee0002',
14     'admin002',
15     '1f22f6ce6e58a7326c5b5dd197973105',
16     1,
17     '{}',
18   );
19
20
21 -- 插入模拟的物料数据
22 INSERT INTO material_info
23   (uuid, name, current_version, status, info, extend )
24 VALUES
25   (
26     '11110000-aaaa-bbbb-cccc-ddddeeee0001',
27     '@my-material/promoted-products',
28     '1.2.3',
29     1,
30     '{}',
31     '{}',
32   ),
33   (
34     '11110000-aaaa-bbbb-cccc-ddddeeee0002',
35     '@my-material/promoted-banner',
36     '4.5.6',
37     1,
38     '{}',
39     '{}',
40   );
41
42 --- 插入模拟的物料快照数据
43 INSERT INTO material_snapshot
44   (version, user_uuid, material_uuid, material_data, status, extend )
45 VALUES
46   (
47     '0.1.0',
48     '00000000-aaaa-bbbb-cccc-ddddeeee0001',
49     '11110000-aaaa-bbbb-cccc-ddddeeee0001',
50     '{"a":123}',
51     1,
52     '{}',
53   ),
54   (
55     '0.2.0',
56     '00000000-aaaa-bbbb-cccc-ddddeeee0001',
57     '11110000-aaaa-bbbb-cccc-ddddeeee0001',
58     '{"a":123}',
```



```
59     1,  
60     '{}'  
61 ),  
62 (  
63     '1.2.0',  
64     '00000000-aaaa-bbbb-cccc-ddddeeee0002',  
65     '11110000-aaaa-bbbb-cccc-ddddeeee0001',  
66     '{"a":123}',  
67     1,  
68     '{}'  
69 ),  
70 (  
71     '1.2.1',  
72     '00000000-aaaa-bbbb-cccc-ddddeeee0002',  
73     '11110000-aaaa-bbbb-cccc-ddddeeee0001',  
74     '{"a":123}',  
75     1,  
76     '{}'  
77 ),  
78 (  
79     '3.2.1',  
80     '00000000-aaaa-bbbb-cccc-ddddeeee0001',  
81     '11110000-aaaa-bbbb-cccc-ddddeeee0002',  
82     '{"a":123}',  
83     1,  
84     '{}'  
85 ),  
86 (  
87     '4.3.2',  
88     '00000000-aaaa-bbbb-cccc-ddddeeee0001',  
89     '11110000-aaaa-bbbb-cccc-ddddeeee0002',  
90     '{"a":123}',  
91     1,  
92     '{}'  
93 )  
94
```

如果要联合员工用户表、物料信息表和物料快照表，查询具体用户的操作记录日志，需要分步操作。可以分成三步。

- 第一步，联合物料信息表和物料快照表，整合一个物料快照详细信息的“临时物料快照联合表”。
- 第二步，联合员工用户表和“临时物料快照联合表”，整合成一个“临时完整员工操作物料日志表”。
- 第三步，按需查询“临时完整员工操作物料日志表”。

第一步，联合物料信息表和物料快照表的 SQL 实现。



```
1 SELECT
2     mi.name AS material_name,
3     ms.user_uuid AS user_uuid,
4     mi.current_version AS current_version,
5     ms.version AS snapshot_version,
6     ms.create_time AS create_time
7 FROM
8     material_snapshot as ms
9 LEFT JOIN material_info AS mi ON mi.uuid = ms.material_uuid
```

第二步，联合员工用户表和“临时物料快照联合表”。

复制代码

```
1 SELECT * FROM (
2     SELECT
3         mi.name AS material_name,
4         ms.user_uuid AS user_uuid,
5         mi.current_version AS current_version,
6         ms.version AS snapshot_version,
7         ms.create_time AS create_time
8     FROM
9         material_snapshot as ms
10    LEFT JOIN material_info AS mi ON mi.uuid = ms.material_uuid
11 ) AS m
12 LEFT JOIN user_info AS u ON u.uuid = m.user_uuid;
```

第三步，按需查询“临时完整员工操作物料日志表”。

复制代码

```
1 SELECT
2     u.username AS username,
3     m.material_name AS material_name,
4     m.snapshot_version AS snapshot_version,
5     m.create_time AS time
6 FROM (
7     SELECT
8         mi.name AS material_name,
9         ms.user_uuid AS user_uuid,
10        mi.current_version AS current_version,
11        ms.version AS snapshot_version,
12        ms.create_time AS create_time
```

```
13 FROM
14 material_snapshot as ms
15 LEFT JOIN material_info AS mi ON mi.uuid = ms.material_uuid
16 ) AS m
17 LEFT JOIN user_info AS u ON u.uuid = m.user_uuid;
```



## 21

课程：[🔗21 结合 Vue 3 和 Koa.js 实现注册登录](#)

提问：通过 **Cookie** 作为登录态唯一数据存在浏览器里，如果 **Cookie** 被其他人拿到，是否能伪造 **Cookie** 所属用户的登录行为？有办法避免出现这个问题吗？从 **Web** 服务平台提供方视角出发，一般怎么做登录态的保护？

通过 **Cookie** 作为登录态唯一数据存在浏览器里，如果 **Cookie** 被其他人拿到，是否能伪造 **Cookie** 所属用户的登录行为？

- 能伪造，理论上讲，如果 **Cookie** 被别人拿到了，可以被“种”到他自己的浏览器里，那么就等于别人的浏览器也有登录态，也就能拥有 **Cookie** 用户的登录权限了。

如果 **Cookie** 被其他人拿到了，伪造了登录，那有办法避免出现该问题吗？大厂一般怎么做登录态的保护？

有办法避免的。一般在大厂里，会有专门的部门做统一用户登录等权限管理。而且会从多个维度来进行校验用户登录态的有效性。例如这三种维度：

- 用户 **IP** 维度。每次登录或者 **HTTP** 访问时候，会在数据库、或者缓存、或者日志记录登录用户当前的 **IP** 地址，并且与上次记录的做比对，如果判断不是同一个地区的 **IP** 地址，就需要用户做二次验证，例如邮箱验证码，或者短信验证码验证。上述用户所在的 **IP** 地址是可以从 **HTTP** 请求里获取到的。
- 用户常用浏览器标记维度。例如用浏览器的存储能力，例如 **localStorage** 来存储一些用户标记，下次请求时候，判断是否为常用浏览器，如果不是，就进行二次验证。
- 用户 **Cookie** 的时效性维度。对 **Cookie** 进行短时间内的定时更新，即使某个时间的 **Cookie** 被别人获取，也只是在短时间内的泄露问题。

大厂里除了上述维度进行验证用户是否异常，还有更高阶的维度判断，例如基于用户 HTTP 请求的算法分析等，就是算法分析实时日志时候，发现当前时间段内用户请求某个 HTTP 的 API 异常多，就会对用户进行二次验证。



## 22

课程：🔗 [22 物料组件的编译和管理，处理组件的多种模块格式](#)

提问：前台场景运行页面时，通过 ESM 或者 AMD 格式进行异步加载物料的代码文件，如果页面依赖的物料变多了，物料文件请求也会变多，这会影响页面打开时间吗？有什么办法可以提高页面打开时间吗？

页面上物料请求文件变多，会导致用户加载页面时间变长。

**解决思路就是减少首屏文件请求，来提高用户体验。**核心的解决点就是“减少首屏文件请求”，那么就要在物料组装的运行时中，做好物料的“懒加载”，主要实现方式是首屏只渲染首屏的物料组件，第二屏开始监听页面滚动情况，按需加载加载对应的物料文件。

## 23

课程：🔗 [23 运营物料的后台管理，全栈化实现列表分页的功能](#)

提问：在今天的技术方案设计中，物料组件版本号只能升级不能降级，这是为什么呢？

物料组件的升级版本号，只能升级不能降级，这是为了让物料版本号跟 CDN 和 NPM 远程静态资源最新版本保持一致。**页面级别的数据发布和管理才是允许版本降级，所以这不是一个技术问题，而是一个项目的业务逻辑问题。**

物料最终是给页面组装使用的，那么最终上线的数据是页面级别的版本数据，每个页面的每个版本数据里，使用了哪个物料都是有对应版本号。如果使用版本出问题，就在页面级别上，把版本号降级或升级。

思考题就解答到这里，下一讲我们继续实战篇的学习，下一讲见。

生成海报并分享



天下无鱼

<https://shikey.com/>

👍 赞 0

🔗 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 运营物料的后台管理：如何全栈化实现列表分页的功能？

下一篇 围炉夜话 | 学了新技术，在公司用不上怎么办？

## 精选留言

💬 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。