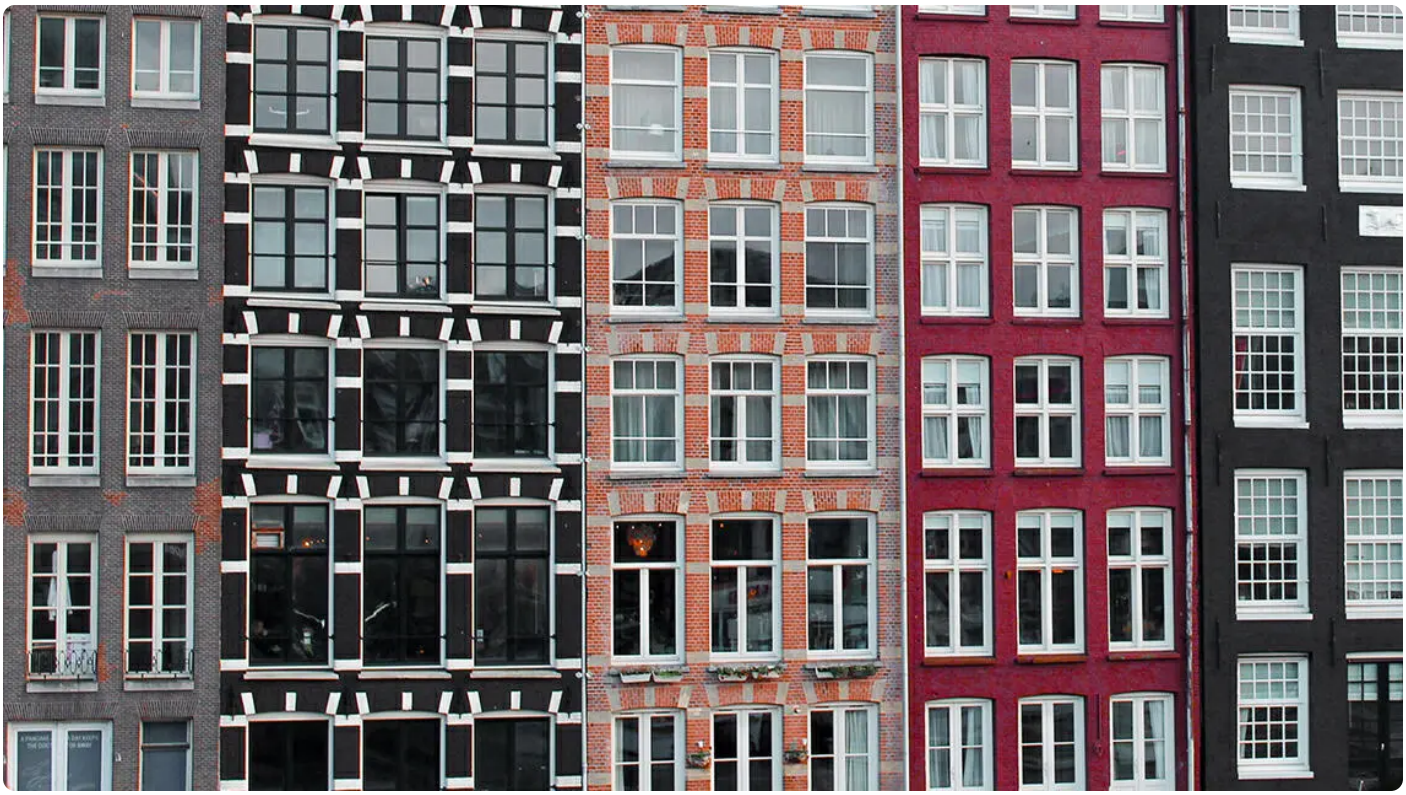
25丨后台搭建功能：如何设计和实现Vue.js运营后台的搭建功能？

2023-02-06 杨文坚 来自北京

《Vue 3 企业级项目实战课》 课程介绍 >



讲述：杨文坚

时长 14:08　大小 12.91M

你好，我是杨文坚。

在课程项目“运营搭建平台”的功能分析中，我们把平台功能分成三大功能维度，“用户维度”“物料维度”和“页面维度”。

页面的数据结构，其实就是物料数据源的组成，上节课我们用 JSON Schema 来描述物料数据源，等于描述了页面的数据结构。物料维度相关的内容和技术实现就告一段落。从今天开始，我们进入页面功能维度的学习。

页面功能维度，是我们运营搭建平台项目最后一个功能维度，也是最重要、最复杂的功能维度。

页面功能维度可以分成五大功能模块，“页面搭建”“页面编译和运行”“页面发布流程”“页面版本管理”和“页面渲染方式”，按照业务逻辑的操作顺序，先有“页面搭建”，才能生产页面数据，从

而带动后续的功能操作。所以，这节课我们就来学习运营搭建平台的“页面搭建”功能。

如何设计页面搭建的数据格式

之前我们说过，页面是由物料组成的，物料是由 Vue.js 组件和数据源组成的，而数据源构成页面的数据结构。所以，不管是页面还是物料，最底层的构成元素就是“数据”和“静态资源”，页面的搭建，也是围绕着“数据”和“静态资源”来实现的。

那么，我们要做的事情就是页面的数据格式设计。

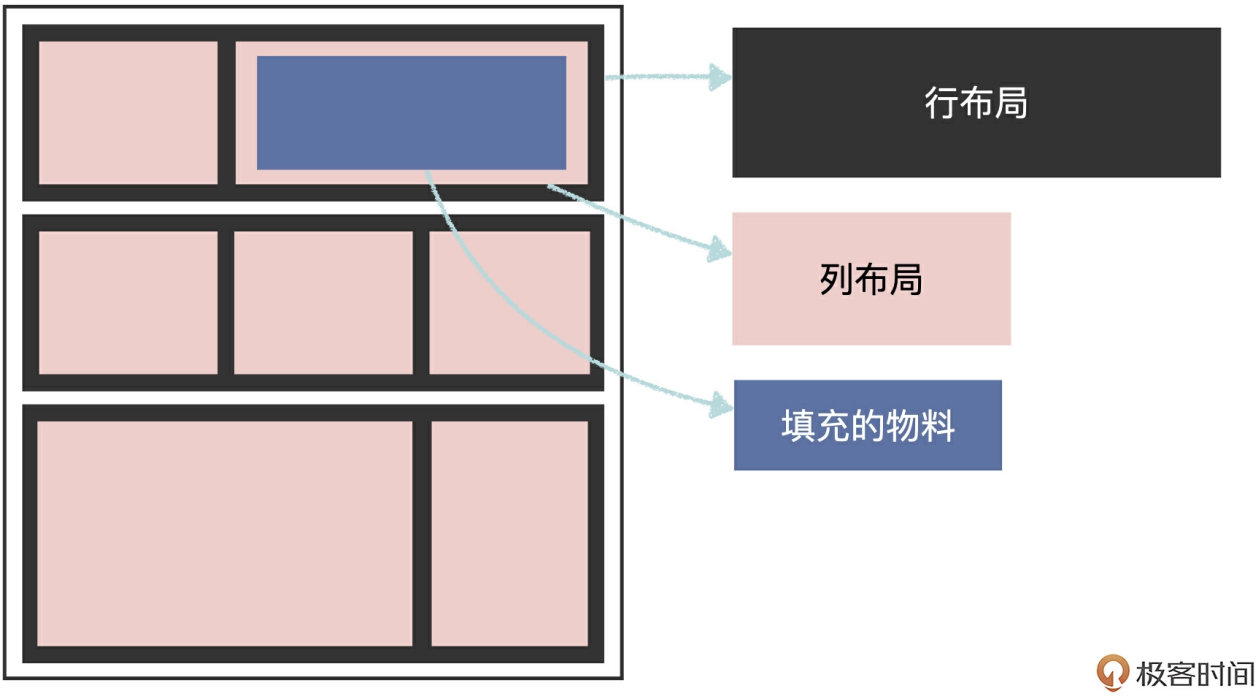
不过设计搭建页面的数据格式之前，**我们要先对页面布局结构做设计**。毕竟，页面不是简单的物料叠加组合，而是根据有规范的排列规则，来排列物料。这个排列规律就是布局设计。当布局设计出来后，我们根据布局里的“坑位”来填充物料，就能形成一张完整的页面。

那么要如何做布局设计呢？

首先，我们要做“布局规范”的选择。在前端领域里，页面布局规范有很多种，比如网格布局、绝对定位布局、流式布局等等。在常见的搭建场景中，现在比较主流的布局规范是网格布局。

网格布局，顾名思义就是把页面网格化，页面由“行”和“列”两种模块元素组成。一个或多个“列布局”模块沿着水平方向，横向组成一个“行布局”模块，一个或多个“行布局”模块沿着垂直方向，纵向组成页面。

有网格布局的规范选择，**接下来就是要做布局的数据设计了**。我这里画了一张图辅助你理解。



网格布局，你可以直接理解成“行列布局”，那么布局数据的格式就是“页面数据”嵌套了“行数据”，“行数据”里嵌套了“列数据”。

我们可以基于 TypeScript 来设计页面布局数据格式。

复制代码

```
// 列布局
interface LayoutColumn {
  name?: string;
}

// 行布局
interface LayoutRow {
  columns: LayoutColumn[];
}

// 布局
interface Layout {
  rows: LayoutRow[];
}
```

设计了布局的基本数据格式后，**还要考虑布局尺寸问题**。我们都知道，用浏览器访问页面，常规浏览的视角是页面从上到下，也就是说页面的高度是动态变化的。

这就意味着，在网格布局中，页面的每一行高度，我们可以根据列布局中物料的高度弹性变化，但是页面的宽度就要做限制，每一行的宽度，受到顶级布局宽度的限制，带动着行里的每一列就要分割限制的宽度。这时候布局数据的 TypeScript 数据类型可以扩展添加宽度属性。

复制代码

```
// 列布局
interface LayoutColumn {
  name?: string;
  width: string | number;
}

// 行布局
interface LayoutRow {
  columns: LayoutColumn[];
}

// 布局
interface Layout {
  width: number | string;
  rows: LayoutRow[];
}
```

现在我们有了完整的布局数据格式，接下来就要**把物料填充到网格布局中的每一列中**。

怎么操作呢？在布局数据中直接添加“物料数据格式”到“列数据格式”？理论上可以这么做，但是在“列数据”中添加“物料数据”，会导致整个“页面布局数据”的数据层级太多，实际 JSON 的“数据深度”太大。

我们可以在每一个列数据中留下一个 uuid，代表每一个物料模块，然后在布局数据格式的同个层级中，定义一个模块的 Map 对象数据，来包含所有物料模块数据格式。**模块 Map 对象数据的键值 Key 就是每列的 uuid，指向所引用的列。**

模块数据可以这么设计，每个独立的物料模块中，包含了物料组件的名称，版本号和物料数据源。看具体的 TypeScript 代码。

复制代码

```
// 物料模块
interface LayoutModule {
  materialName: string;
  materialVersion: string;
  materialData: Record<string, any>; // 物料数据源
```

```
}

// 物料模块Map
interface LayoutModuleMap {
  [uuid: string]: LayoutModule; // uuid指向每一列里的uuid
}
```

到了这里，我们有了网格布局数据格式，也有物料模块数据格式，把他们整合起来，就是完整的页面布局数据。

看完整的 TypeScript 数据类型代码。

复制代码

```
// 列布局
interface LayoutColumn {
  uuid: string; // 列唯一的uuid，指向物料模块的uuid
  name?: string;
  width: string | number;
}

// 行布局
interface LayoutRow {
  uuid: string; // 行唯一的uuid
  columns: LayoutColumn[];
}

// 布局
interface Layout {
  width: number | string;
  rows: LayoutRow[];
}

// 物料模块
interface LayoutModule {
  materialName: string;
  materialVersion: string;
  materialData: Record<string, any>; // 物料数据源
}

// 物料模块Map
interface LayoutModuleMap {
  [uuid: string]: LayoutModule;
}

// 完整的页面布局数据
interface PageLayoutData {
  layout: Layout;
  moduleMap: LayoutModuleMap;
```

```
}
```

基于这段完整的页面布局数据的 TypeScript 类型，我来实现一个页面搭建的 JSON 对象数据例子。

复制代码

```json
{
  "layout": {
    "rows": [
      {
        "uuid": "7a3dbfa7-29ca-4765-bd58-7a91abda7721",
        "columns": [
          {
            "name": "其它广告位1",
            "uuid": "2be63a53-8a16-43ba-8640-a1ab24bdbae3",
            "width": 600
          },
          {
            "name": "其它广告位2",
            "uuid": "0184e8ed-7df4-4742-9bb4-b8dfa1a0aca7",
            "width": 400
          }
        ]
      },
      {
        "uuid": "0fdbe663-51d8-4038-9d0d-bdeed0c6eb71",
        "columns": [
          {
            "name": "促销商品模块",
            "uuid": "26148a93-e66a-4448-b6a4-4bddbce9ae4f",
            "width": 1000
          }
        ]
      }
    ],
    "width": 1000
  },
  "moduleMap": {
    "26148a93-e66a-4448-b6a4-4bddbce9ae4f": {
      "materialData": {},
      "materialName": "@my/material-product-list",
      "materialVersion": "0.9.0"
    },
    "2be63a53-8a16-43ba-8640-a1ab24bdbae3": {
      "materialData": {},
      "materialName": "@my/material-banner-slides",
      "materialVersion": "0.9.0"
    },
    "0184e8ed-7df4-4742-9bb4-b8dfa1a0aca7": {
```

```
      "materialData": {},
      "materialName": "@my/material-banner-slides",
      "materialVersion": "0.9.0"
    }
  }
}
```

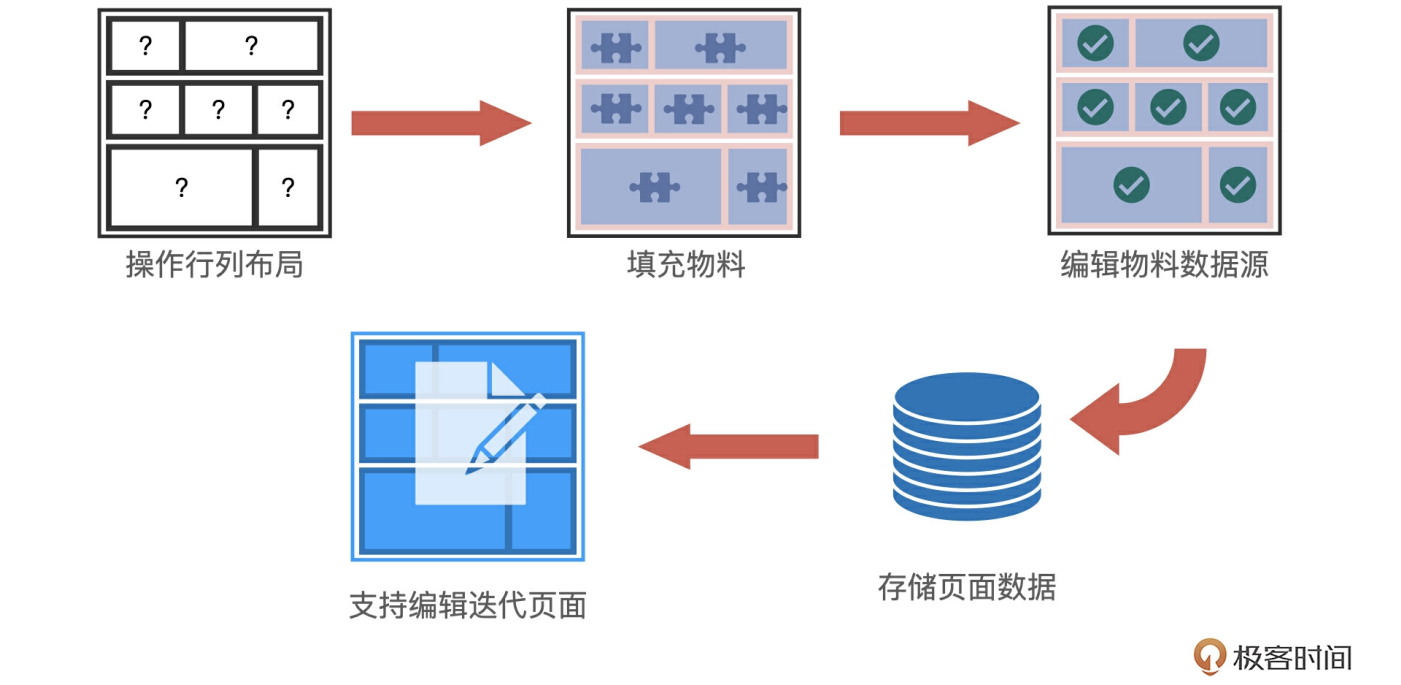
为了统一描述，这里的“布局数据”和“物料模块数据”整合后的数据，我们就统称为“页面布局数据”。

我们已经设计了用于页面搭建的“页面布局数据”，我也举例模拟搭建页面的 JSON 对象数据，接下来就可以实现页面搭建了。

页面搭建的实现原理，就是要基于“搭建页面数据”，再结合“物料组件资源”，也就是物料的 JavaScript 和 CSS 静态资源文件，来完成页面的搭建功能。按照我们之前的功能实现套路，首先要做功能逻辑链路设计，然后基于功能逻辑设计来做技术方案设计，最后落地成实际代码。那么，要如何设计页面搭建的功能链路呢？

如何设计页面搭建的功能链路

从页面布局数据的分析步骤可以看出，实现一个搭建的页面，需要先制定页面的布局，然后再向布局中填充物料。所以页面搭建功能链路，就是从布局到物料的操作过程。

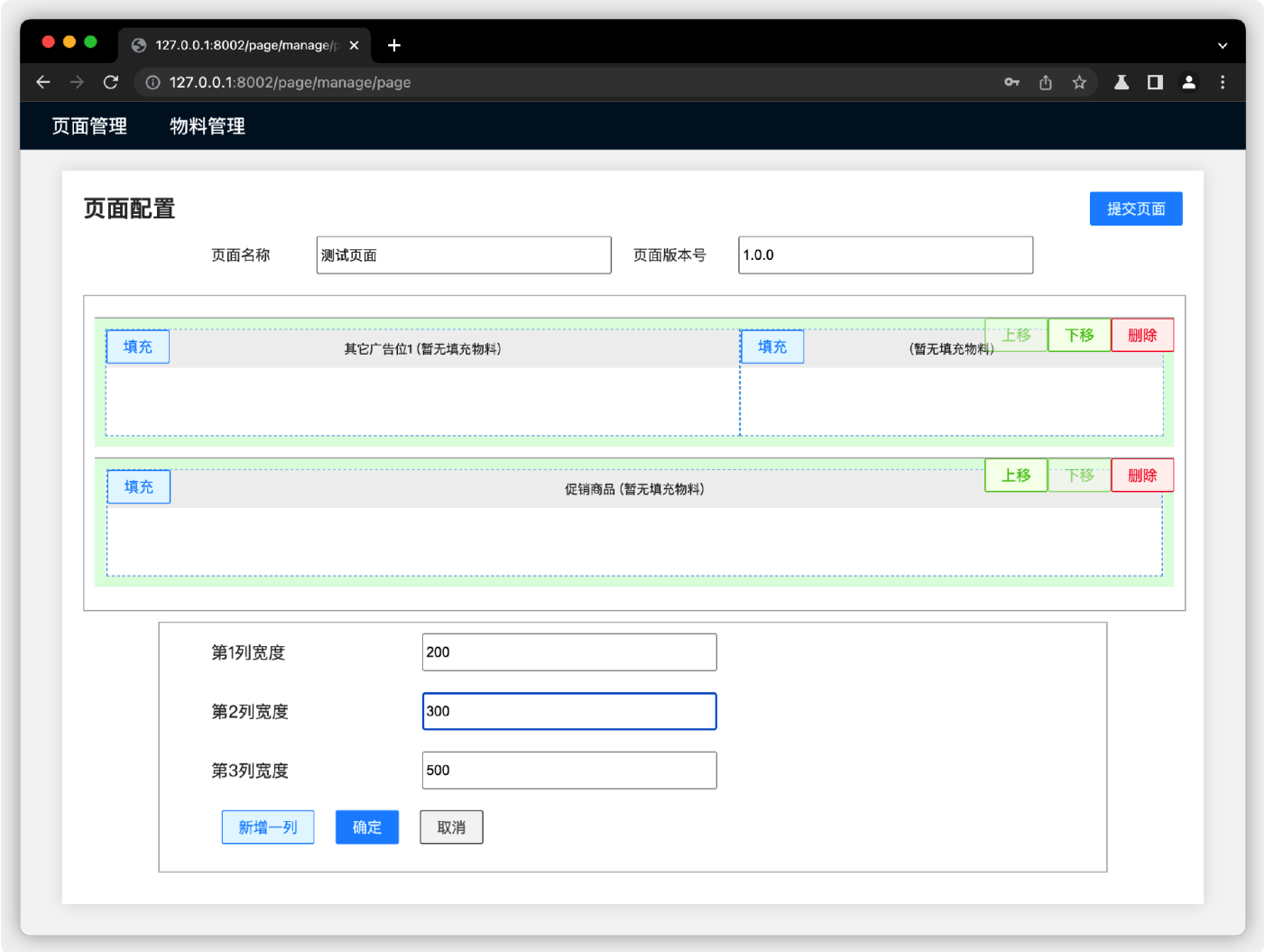


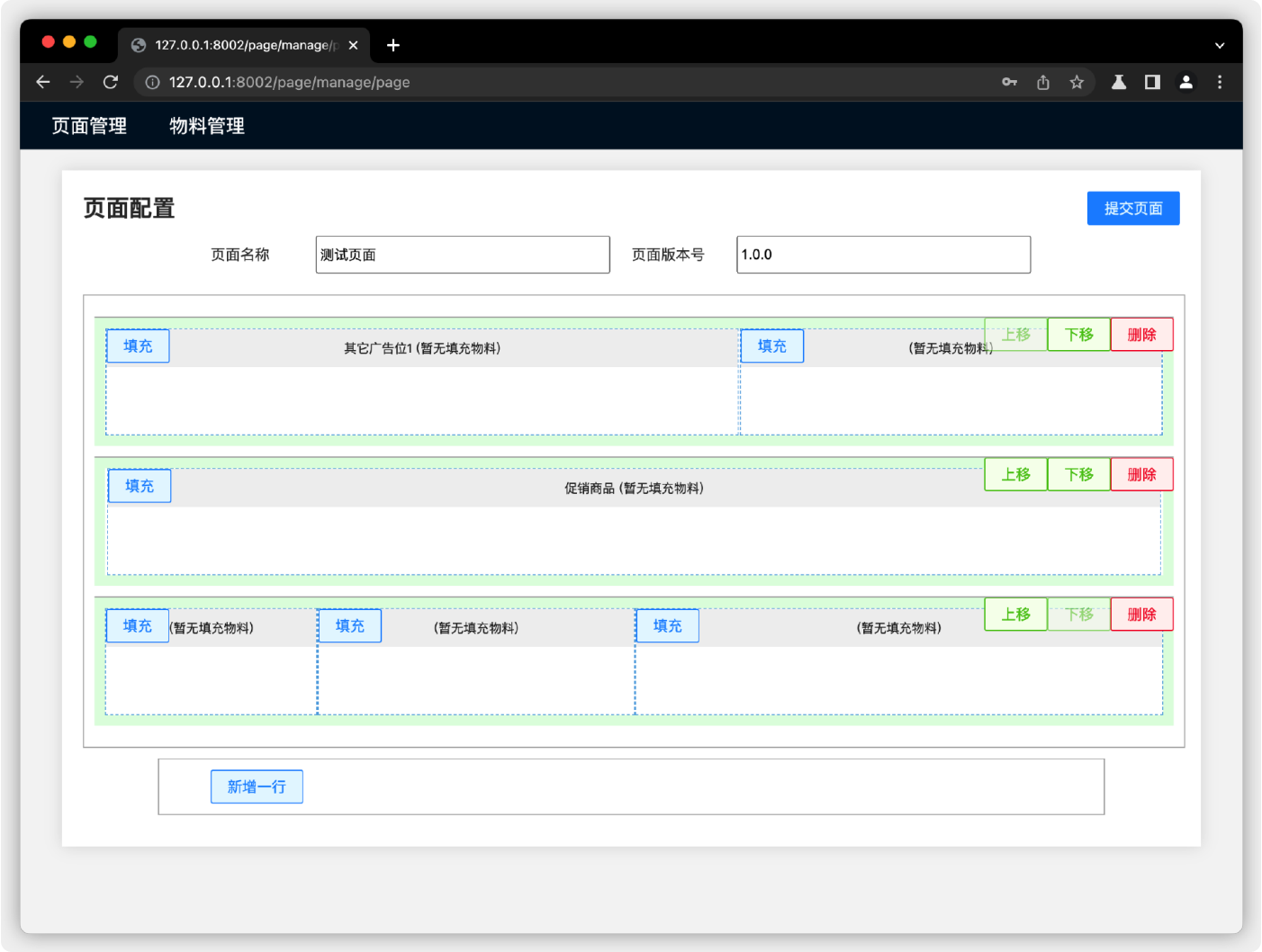
我们能很清晰地把页面搭建功能链路分成五个步骤。

- 第一步：操作页面的行列布局
- 第二步：填充物料模块到列布局中
- 第三步：编辑物料的数据源
- 第四步：发布页面布局数据
- 第五步：支持重新编辑页面布局数据

我们分析一下功能链路上每一步的操作逻辑。

第一步，操作页面的行列布局。这一步，主要功能是支持添加“行布局”，在新增“行布局”时候，支持定义所在“行布局”中的“列布局”。这里的定义“列布局”包括了“列”的个数和每一列分割的宽度。具体操作界面就像这样。

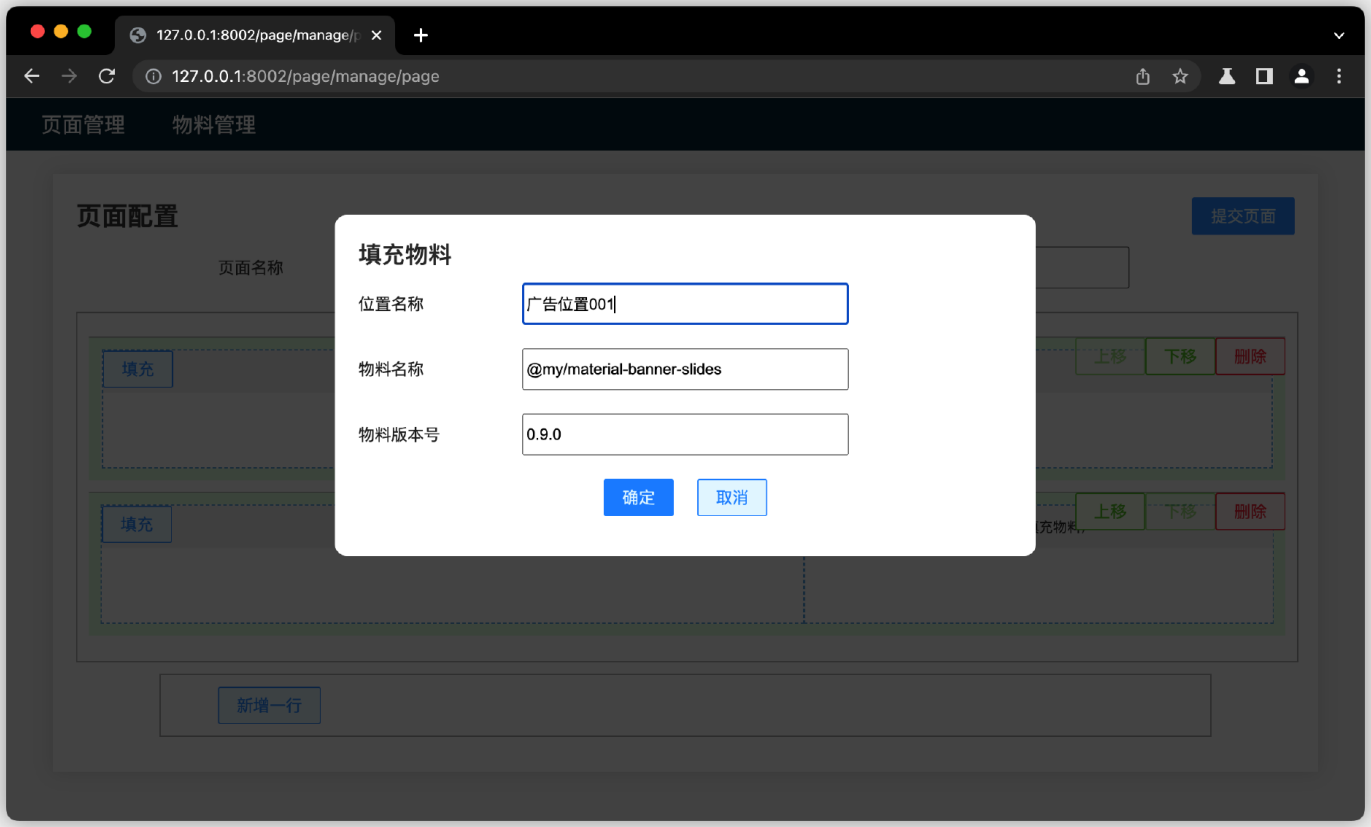


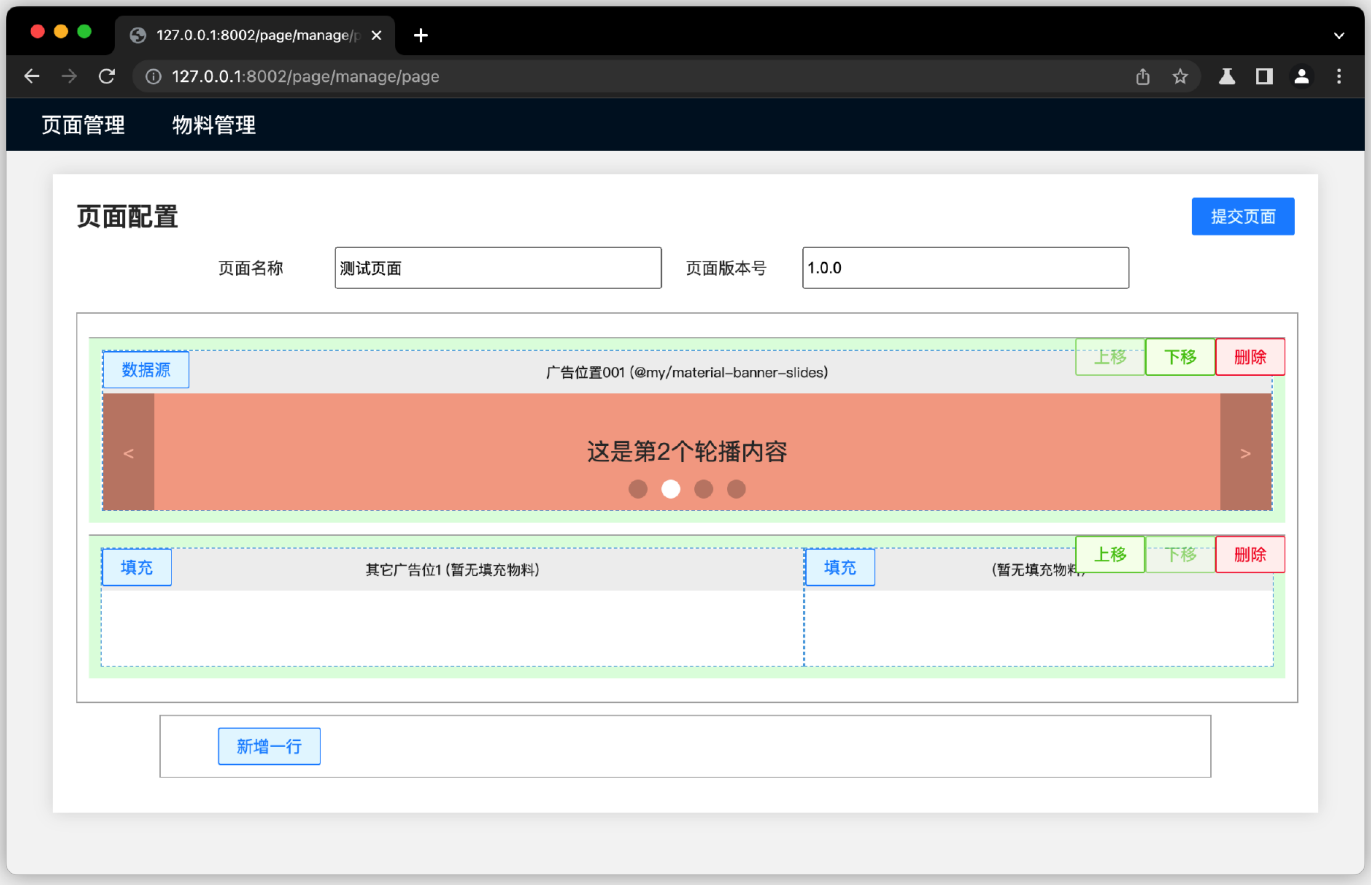


这两张图就是课程的代码案例，实现了行布局的新增和列布局的定义。

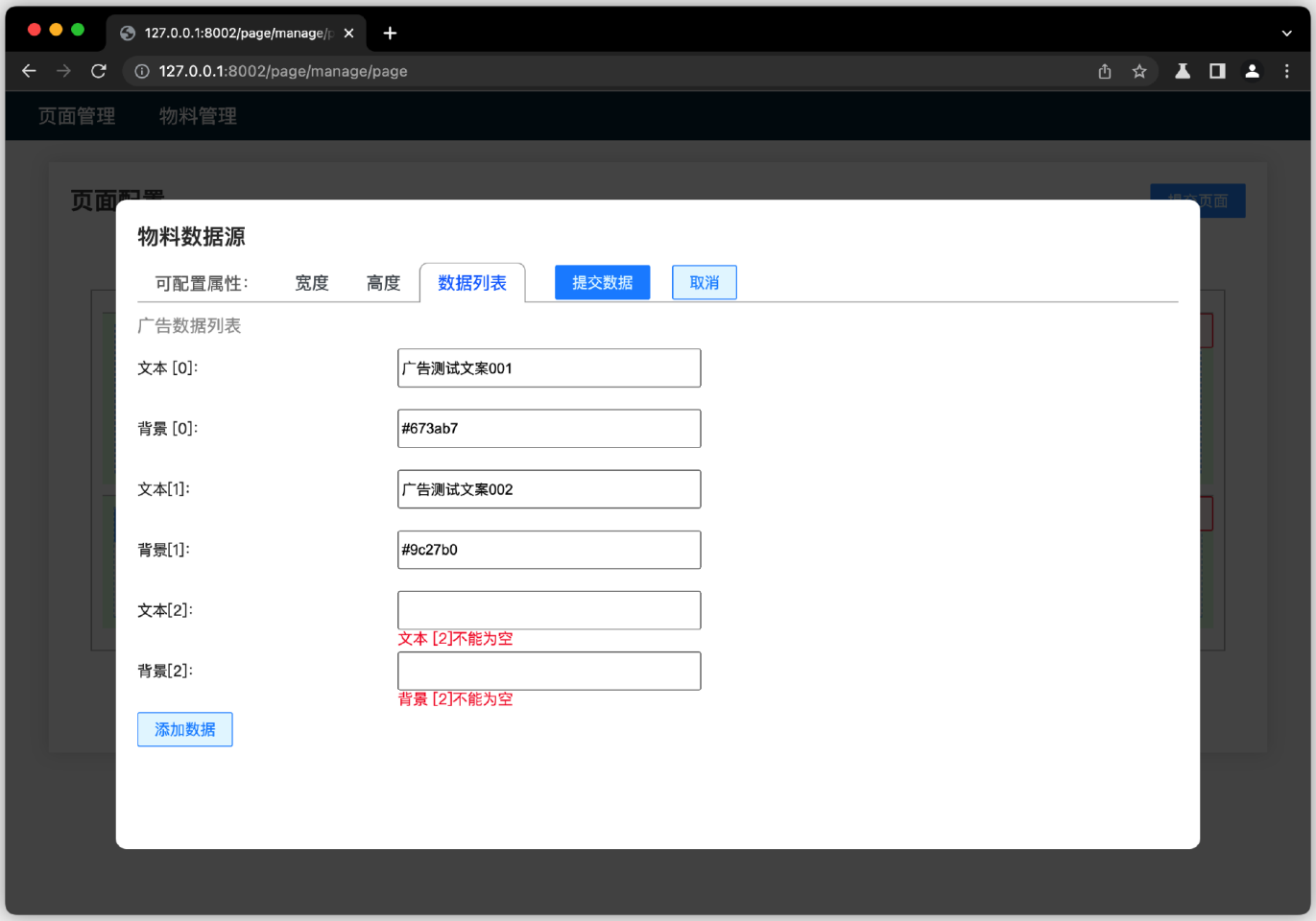
从上面页面布局数据定义过程中，我们可以知道，**“列”是整个页面布局中的最小单位，也就是说，最终物料是填充到“列布局”中的。**

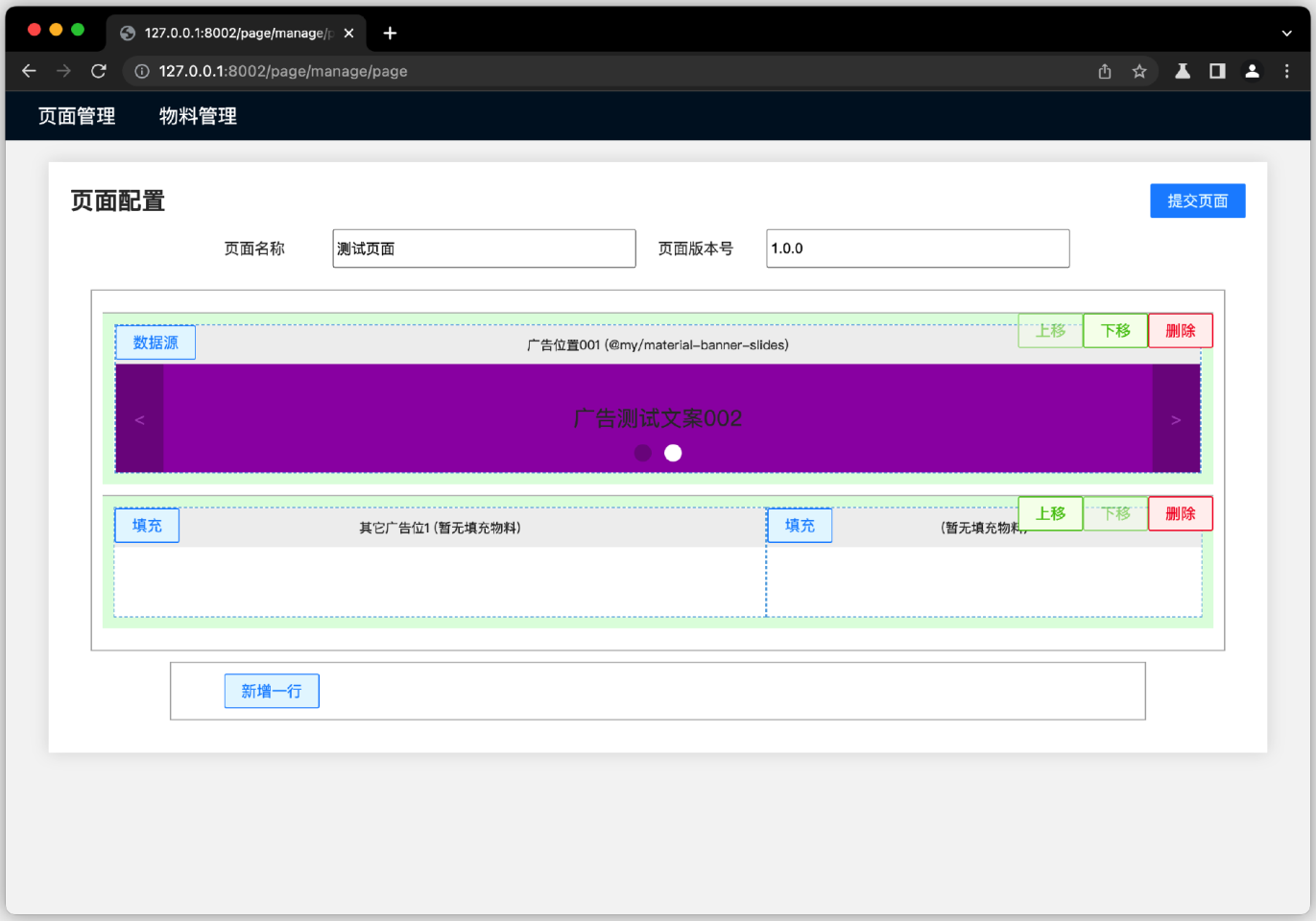
所以，第二步，填充物料模块到“列布局”中，我们先进行物料的查询，查询验证成功后就填充到“列布局”中。具体操作界面就像这样。





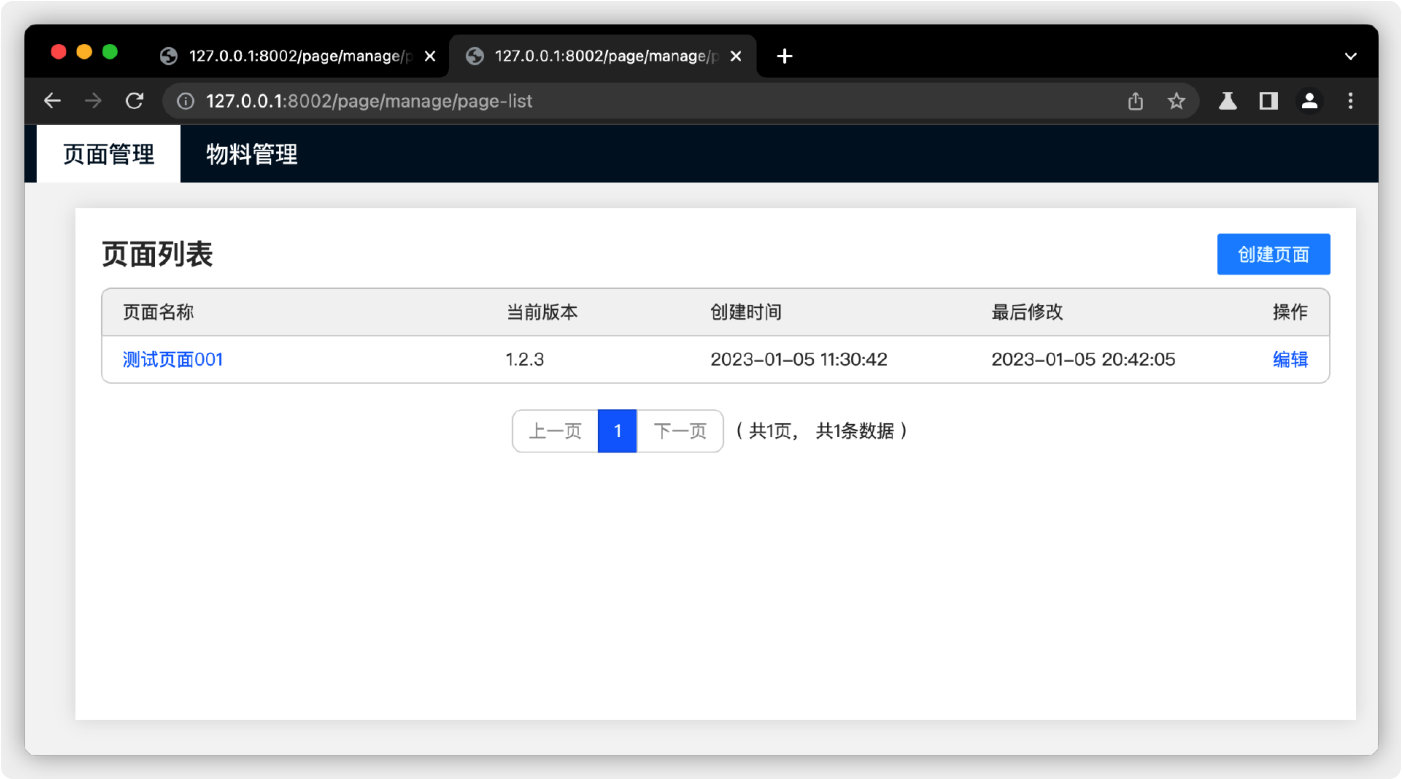
第三步，编辑物料的数据源。在这个步骤中，支持对每一列中的物料模块，进行数据源的编辑。根据已填充的物料组件，显示对应的数据源编辑表单，编辑对应的数据源。具体操作界面我也截图了。


127.0.0.1:8002/page/manage/p
127.0.0.1:8002/page/manage/page
页面管理
物料管理
物料数据源
可配置属性:
宽度
高度
数据列表
提交数据
取消
广告数据列表
文本 [0]:
广告测试文案001
背景 [0]:
#673ab7
文本[1]:
广告测试文案002
背景[1]:
#9c27b0
文本[2]:
文本 [2]不能为空
背景[2]:
背景 [2]不能为空
添加数据


127.0.0.1:8002/page/manage/p
127.0.0.1:8002/page/manage/page
页面管理
物料管理
页面配置
提交页面
页面名称
测试页面
页面版本号
1.0.0
数据源
广告位置001 (@my/material-banner-slides)
上移
下移
删除
广告测试文案002
填充
其它广告位1 (暂无填充物料)
填充
上移
下移
删除
新增一行

截图展示了根据对应的物料，显示对应的数据源编辑表单。当提交了数据源给物料后，渲染出新的效果。

第四步，发布页面布局数据。这一步也就是提交完整的页面布局数据（布局和物料模块数据），存储到数据中，支持下次编辑。发布成功后，跳转到页面列表页面。



第五步，支持编辑页面布局数据。这一步中，支持对已发布的页面数据进行重新修改，基本复用了上面的所有功能点，包括编辑行列布局、填充物料组件、编辑物料数据源和发布修改数据。

到这里，我们就完成了页面搭建的功能逻辑链路的设计，是不是逻辑很清晰。

接下来，我们就要把这些功能逻辑设计进行技术层面的“翻译”，也就是技术方案的设计。

如何做页面搭建的技术方案设计

按照我们之前的套路，功能的技术方案设计，其实就是根据功能逻辑设计的步骤，对应找到合适的或者熟悉的技术实现方式。那我们就根据上面的功能逻辑设计的步骤，来定制技术方案。

第一步，操作页面的行列布局。这一步骤中，主要是实现布局的操控。首先，是“行布局”的操控，我们可以**基于 Vue.js 的响应式数据**来实现“行布局”的“增删改”操作。然后是“列布局”的定义，我们在行布局的新增时候，就可以用**动态表单**来实现对应“列布局”的宽度定义。

“列布局”还可以用我们之前课程中实现的“**拖拽布局组件**”来支持“列布局”的位置拖拽。说到这里，你可能有疑问，为什么“行布局”不用“拖拽布局组件”，而是直接操作响应式数据来控制呢？

这是因为行布局是垂直排列的，超出浏览器屏幕高度就会出现页面滚动，不方便拖拽，所以用数据来操控实现更合适。如果硬要“拖拽布局组件”来实现，就要改造“拖拽布局组件”的深层嵌套操作。这会增大工作量，而且在超长页面中，拖拽操作也对用户不友好，开发收益不大。

具体的列布局拖拽效果就像这样。

第二步，填充物料模块到“列”布局中。这里就用到我们之间课程中，组件库相关的“对话框组件”和“动态表单组件”。今天的代码案例中，我已经改造了“对话框组件”，让它支持渲染嵌套子组件。最终技术逻辑是弹出一个对话框，嵌套一个动态表单来进行填充物料。

在这个步骤中，我们还要基于选中的填充物料，把对应组件的静态资源，也就是 JavaScript 和 CSS 文件进行加载，并且渲染到对应“列布局中”。

在代码案例中，我选用了物料 Vue.js 组件的 AMD 模块，进行组件动态加载和动态渲染。主要是封装了一个动态渲染物料模块的 JSX 组件。

```tsx
// packages/work-front/src/pages/manage/modules/page-editor.tsx
import * as Vue from 'vue';

// 动态初始化AMD 运行时环境
async function initAMDEnv(params: {
  materialName: string;
  materialVersion: string;
}) {
  const { materialName, materialVersion } = params;
  if (!window.requirejs) {
    await loadScript('/public/cdn/pkg/requirejs/2.3.6/require.js');
  }
  const paths: Record<string, string> = {};
  // 注册Vue的AMD模块，AMD和当前应用共用同一个Vue.js运行时
  window.define('vue', [], () => Vue);
  paths[materialName] = `/public/cdn/material/${materialName}/${materialVersion
  window.requirejs.config({
    paths
  });
}

// 创建动态物料预览模块 的 Vue.js组件
function createEditModule(params: {
  materialName?: string;
  materialVersion?: string;
  materialData?: any;
  // 省略其它参数...
}) {
  const { materialName, materialVersion, materialData } = params;
  // 物料挂载的 "外壳组件"
  const EditModule: DefineComponent = defineComponent({
    setup() {
      const container = ref<HTMLElement>();

      onMounted(async () => {
        if (!(materialName && materialVersion)) {
          return;
        }
        const params = {
          name: materialName,
          version: materialVersion
        };
        // 初始化 AMD 运行时环境
        await initAMDEnv({ materialName, materialVersion });
        // 加载物料样式
        await loadMaterialStyle(params);
        window.require( ['vue', materialName],
          (Vue: any, MaterialComponent: any) => {
            // 这里拿到的Vue是跟当前应用是同一个Vue.js运行时
            // 动态加载物料组件的AMD模块
            const App = Vue.h(MaterialComponent, materialData || {});
            const app = Vue.createApp(App, {});
```

```
            // 在"外壳组件"中动态挂载物料的AMD格式Vue.js组件
            app.mount(container.value);
          }
        );
      });
      // 中间省略其它代码 ....

      return () => {
        return (
          <div style={{ width }} class="module-page-edit-module">
            {/* 中间省略其它代码 ....   */}
            <div class="page-edit-module-content" ref={container}></div>
          </div>
        );
      };
    }
  });
  return EditModule;
}
```

从代码和注释，你可以看出，填充物料的技术实现，就是动态渲染物料的 Vue.js 组件。核心步骤是先初始化 AMD 运行时环境，然后动态加载 AMD 格式组件，再生成动态的外壳组件，最后在外壳组件里，挂载物料的 Vue.js 组件。

第三步，编辑物料的数据源。这一步骤，可以直接复用上节课编辑物料数据源的表单组件，结合对话框组件一起使用，实现动态获取数据源的 JSON Schema，动态渲染数据源表单，最后实现编辑数据源的功能。

第四步，发布页面布局数据。这要开发对应的 HTTP 服务接口，支持提交数据到服务端，并且存储到数据库中。具体技术实现都是重复性的 SQL 操作和 Node.js 逻辑代码实现，可以复制修改之前物料操作的代码，对数据库中的页面数据进行“增改查”操作。

第五步，支持编辑页面布局数据。这一步最简单，基于上述四个步骤实现的功能，添加默认数据填充和提交数据接口的修改。

到这里，基于页面搭建的功能逻辑设计，我们就实现了技术方案设计，最终的完整实现代码，你可以看课程对应的代码案例。

总结

通过今天的学习，相信你已经掌握了页面搭建的功能实现操作。从功能实现视角总结。

- 页面搭建流程主要分成两个部分，第一是要先做布局设计，第二是填充物料到布局中。所以页面搭建是围绕着“操作布局”和“填充物料”来进行的。
- 页面搭建核心点是“操作布局”和“填充物料”，底层核心数据就是“布局数据”和“填充的物料模块数据”。所以页面搭建的数据设计，首先要考虑到“布局”和“布局填充”的数据。

我们再从技术视角总结一下。

- 页面搭建过程中，对布局和物料的编辑等操作，会遇到大量表单场景，技术方案选择动态表单，能大大提高开发效率。
- 拖拽技术不一定能适用所有页面搭建功能，要考虑到页面超长出现浏览器滚动场景，拖拽操作就体验不友好。
- 页面布局操作，涉及大量的数据设计和数据操作，所以类似搭建项目，要尽量使用 TypeScript 进行开发，限制数据类型格式。

思考题

留给作业给你，在今天中的页面搭建技术方案设计中，为什么渲染物料组件考虑用 AMD 模块格式，而不选用 ESM 模块格式呢?

期待能在留言区看到你的想法。通过今天的学习，希望你能掌握页面搭建的实现要点，举一反三，不管之后遇到什么样的搭建场景，都能游刃有余地按照套路进行功能实现和技术思考。下节课见。

完整的代码在这里

上一篇　24 | 后台搭建数据源：如何设计运营搭建页面的数据结构？

下一篇 26丨页面编译和运行：如何设计Vue.js搭建页面的渲染策略？

精选留言 (1)

写留言

cyw0220

2023-02-13 来自浙江

AMD动态加载比较方便吧

作者回复: 您好，JS模块化文件动态加载“哪种更方便”，取决于实际场景。

比如假设要考虑“IE8”等旧版本浏览器场景，那么AMD比ES Modules兼容性更高，动态加载更方便。

又比如假设只需要考虑最高版本的Chrome浏览器，那么用ES Modules动态加载文件更方便，因为是原生的浏览器能力。

1