11 | 布局组件：如何实现自研组件库的布局方案？

2022-12-16 杨文坚 来自北京

《Vue 3 企业级项目实战课》

讲述：杨文坚

时长 13:24　大小 12.24M

你好，我是杨文坚。

在做前端页面开发的过程中，无论你用的是 Vue.js、React.js 还是小程序，都是通过一个个组件的排列、嵌套或者动态渲染等方式的组合，实现一个前端页面的渲染和功能效果。

既然页面都是由一个个组件用各种方式组合而成的，你有没有想过，这个组合形式有没有什么规范和讲究？能不能形成通用组件来重复使用呢？

其实，在企业里用 Vue.js 或者 React.js 来开发页面，对组件布置组合形式是有规范、有讲究的。一般这类“规范”或者“讲究”被称为“页面布局”，能提供布局能力的组件，也就称为“布局组件”。

目前主流的 Vue.js 或者 React.js 的开源组件库里，都会提供一些基础的布局组件，利用这些布局组件，我们可以拼接出页面的大致“骨架”，然后在布局“骨架”里“填充”所需要的功能组件，

从而组合成业务需要实现的页面。

这么说估计有点不好理解，我们代入一个工作中常见的开发场景，来看看布局组件到底有什么用，如何实现自己的布局组件。

布局组件

假设你开发了一个电商首页，从上到下是轮播图片区块、商品类目区块、促销商品区块和热销商品区块。你完成代码后，交接给另外一个前端同事维护，他接到需求，要在轮播图片和类目模块左右两侧各加一个广告位区块。这位前端同事完成需求后，再转交给你维护，你再接到需求，在轮播图顶部再加一个广告位区块……

就这样，不停地更换开发者或者几个前端开发者一起维护，不停“拆掉”页面“模块”，在指定“插入”新增广告位区块，“打乱”原有各个模块的排布。

再加上，每个前端开发者的对布局的代码实现“习惯”不一样，这个经过多次转手维护的页面布局代码，变得十分混乱，甚至再有加“广告位”的需求，新人开发已经难以下手了。

这种情况，如果给你一次重新实现的机会，你会怎么做？

重新开发这个页面，最好的方式就是**一开始“约定好布局规范”，根据规范，用组件方式“搭建好布局骨架”，开发时就在“布局骨架”里填充需要的功能代码**。

如果遇到业务需求，要变更布局，例如在页面某个位置插入一个广告位布局，就先根据“布局规范”来调整“布局骨架”，也就是调整布局组件，而布局组件里的功能组件不做变动。布局组件排布调整完后，在新加入的布局组件里填充新需求功能的组件，例如广告组件。

这样，以后需求要在页面“添加广告位”，改变布局组件的排布，就可以调整页面布局，不需要担心布局混乱和频繁修改原有功能组件的代码。

这个例子可以看出布局组件的几点重要性：

- 统一前端页面布局的开发规范；
- 降低设计师、业务方和前端开发者在对接页面布局变更的沟通成本；

- 降低前端页面修改布局的开发成本；
- 降低前端页面合作开发的理解成本。

总结一句话就是，**布局组件可以用“规范”来“降低成本”**。那么，我们应该如何实现 Vue.js 3.x 自研组件库里的布局组件呢？

在讲解实现之前，我们先老规矩分析一下需要准备什么。

实现布局组件需要准备什么？

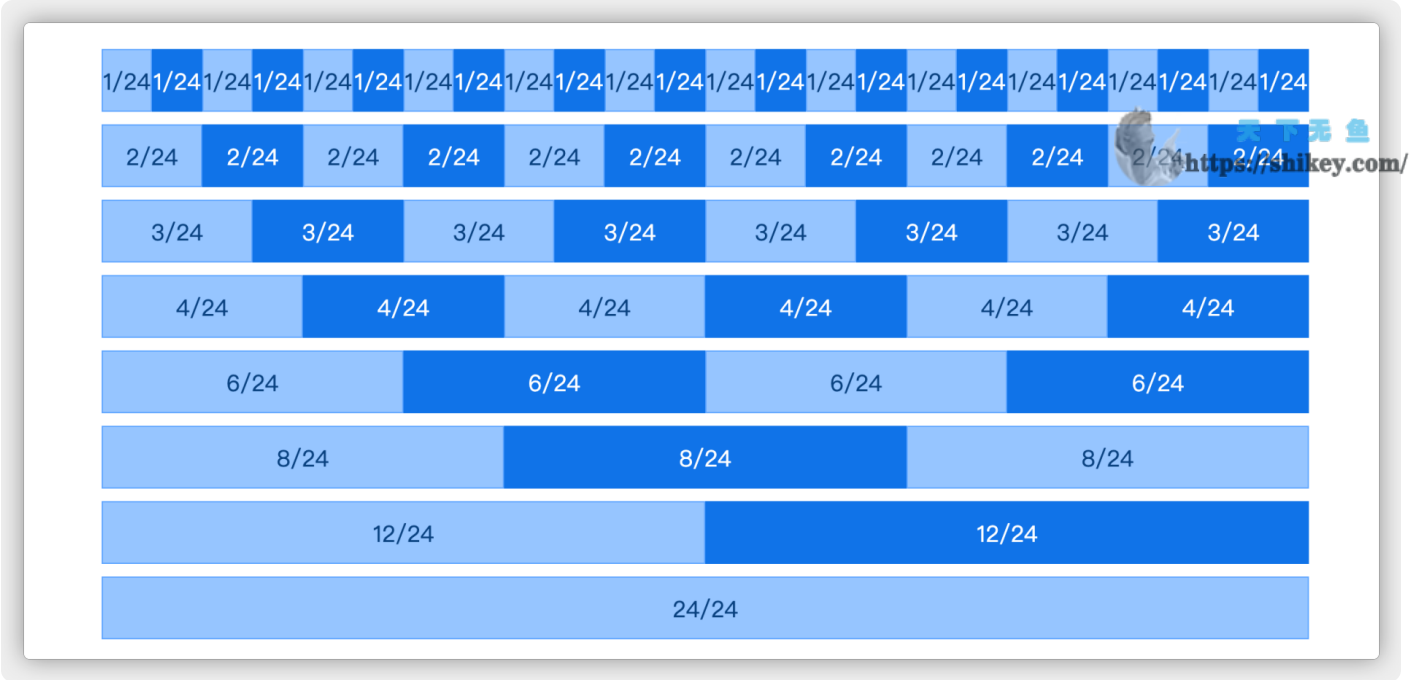
在实现布局组件前，我们首先要做的当然是**布局的规范设计**。布局的规范设计就跟“一千个读者就有一千个哈姆雷特”一样，在开发过程中，不同前端开发甚至是网页设计师都有不同见解。

这里，基于 Ant Design 和 Element Plus 这两个主流的开源组件库，我总结出布局规范设计的三个准备点：

- 栅格化系统的设计；
- 不同布局组件的语义化定义；
- 根据栅格组件和语义布局组件，组合实现各种页面布局。

第一个准备点“**栅格化系统设计**”。

听起来是不是有点“高大上”的味道，其实理解起来非常简单。通俗点说，“栅格化”就是将一个平面进行划分成行（Row）和列（Column）来管理布局，每一行分成一个个等分的格子，一个个格子排列起来就是“栅格化”，如下图所示：



前端领域，“栅格化系统设计”，就是将页面在横向视角分成一行行（Row），在每一行里划分成指定数量的纵向等分格子列（Column），每一个格子就是后续定义布局大小的一个“单位”。

总之，将页面等分成格子，并且将格子作为布局单位，就是“栅格化系统”。同时，栅格化组件也是一种布局组件，只不过是一种没有语义的基础布局组件，用来规范布局基本尺寸或位置。

第二个点，就是“**不同布局组件语义化定义**”。

这里的不同布局组件，就是布局中的一些有语义场景的组件，例如头部组件（Header）、内容组件（Content）和底部组件（Footer）等有一定布局含义的组件，就是语义化的布局组件。

不同企业或者不同业务场景，可能需要定制不同语义的布局组件。我刚刚举例的只是一些常见的布局组件，日常开发中还需要一些特定场景的布局组件，例如一些“悬浮球”的固定悬浮的布局组件。

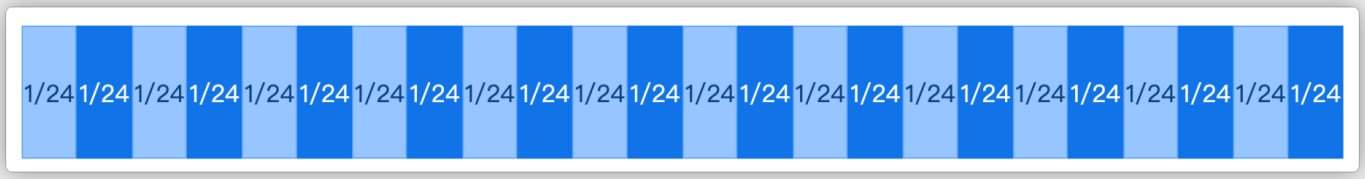
第三个点，**根据栅格化系统来再结合不同语义的布局组件，就可以实现各种业务中所需要的页面布局格式。**例如，开发一个卡片布局组件，要划分卡片的“面积划分”，就可以使用栅格化组件进行二次封装开发。

说了这么多规范，那么我们如何在 Vue.js 3.x 的技术体系内实现上述规范下的布局组件呢？我就从最基础的布局组件——栅格组件讲起。

如何实现栅格组件？

上面提到过，栅格化系统设计，就是将页面等分成一个个格子，每个格子作为布局单位。那么栅格组件就是用来**实现和定义页面中某个区块布局尺寸等于“多少个格子”**。

目前页面的栅格化系统设计，将页面分成 12 等分或者 24 等分。分得越多，布局尺寸的处理就可以越精细。所以我选择 24 等分的栅格设计方式，也就是将页面栅格化成“24 个格子”。



现在来实现栅格组件。首先，我们需要实现一个栅格的行组件（Row），用来作为列组件（Column）的外部容器，具体实现方式如下所示：

Vue 代码：

复制代码

```vue
<template>
  <div :class="{ [className]: true }">
    <slot v-if="$slots.default"></slot>
  </div>
</template>
<script lang="ts" setup>
import { prefixName } from '../theme';
const className = `${prefixName}-row`;
</script>
```

Less 代码：

复制代码

```less
@import '../../theme/variable.less';

.@{prefix-name}-row {
  display: flex;
  flex-flow: row wrap;
  min-width: 0;
}
```

接下来就是实现 Row 组件里的列组件 Col（Column 的缩写）。

还记得我所说过的 24 等分吗？这个 24 等分不是体现在组件的 HTML 代码上，而是体现在 CSS 代码里。设计 Col 组件的 Props，传入 span 参数来控制等分尺寸，例如，传入 1 就使用外围容器 1/24 等分宽度的 className，传入 6 就使用 6/24 等分宽度的 className。

需要实现以下代码，Vue 代码：

复制代码

```vue
<template>
  <div :class="{ [baseClassName]: true, [spanClassName]: true }">
    <slot v-if="$slots.default"></slot>
  </div>
</template>
<script lang="ts" setup>
import { prefixName } from '../theme';
const props = defineProps({
  span: Number
});

function getSpan(propSpan: number | undefined): number {
  if (typeof propSpan === 'number') {
    const span: number = Math.ceil(Number(propSpan));
    if (span >= 1 && span <= 24) {
      return span;
    }
  }
  return 1;
}

const baseClassName = `${prefixName}-col`;
const spanClassName = `${prefixName}-col-${getSpan(props.span)}`;
</script>
```

上述 Vue 代码里，基于 Props 传入的 span 参数选择对应栅格 CSS 的 className，不同 className 对应的不同栅格化尺寸的 CSS 代码的实现，如下 Less 代码所示：

Less 代码：

复制代码

```less
@import '../../theme/variable.less';

.generate-col(@num) {
```

```
  &.@{prefix-name}-col-@{num} {
    width: percentage((1 / 24 * @num));
    flex: 0 0 percentage((1 / 24 * @num));
  }
}

.generate-col-list(@count) when (@count > 0) {
  .generate-col-list((@count - 1));
  .generate-col(@count);
}

.@{prefix-name}-col {
  position: relative;
  display: block;
  box-sizing: border-box;
  .generate-col-list(24);
}
```


上述代码是用 Less 里的函数语法，批量循环实现 24 栅格化的 CSS 样式。

最后就是使用上述两个组件（Row 和 Col）来使用组合 Vue.js 3.x 下的栅格化系统，代码如下：

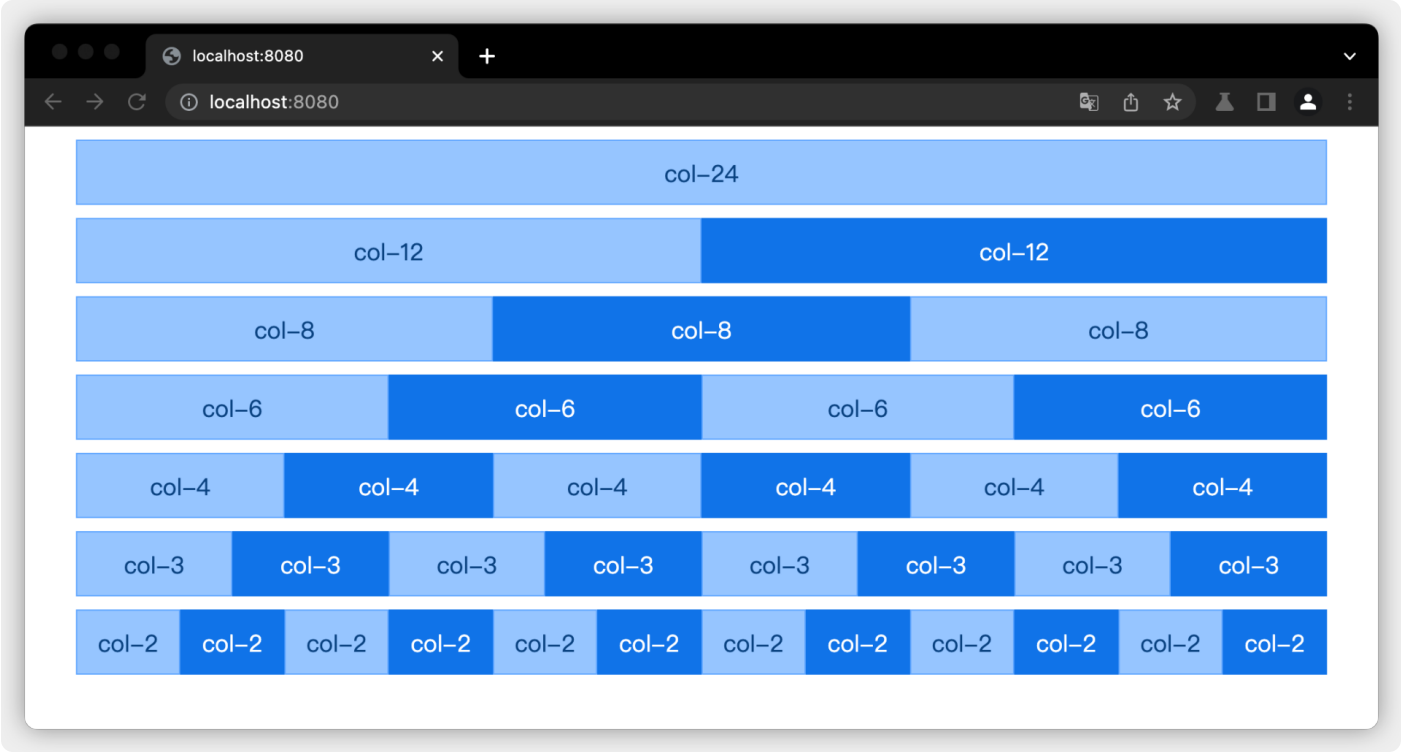
复制代码

```
<template>
    <Row>
      <Col class="gird" :span="24">col-24</Col>
    </Row>
    <Row>
      <Col class="gird :span="12">col-12</Col>
      <Col class="gird gird-dark" :span="12">col-12</Col>
    </Row>
    <Row>
      <Col class="gird" :span="8">col-8</Col>
      <Col class="gird gird-dark" :span="8">col-8</Col>
      <Col class="gird" :span="8">col-8</Col>
    </Row>
    <Row>
      <Col class="gird" :span="6">col-6</Col>
      <Col class="gird gird-dark" :span="6">col-6</Col>
      <Col class="gird" :span="6">col-6</Col>
      <Col class="gird gird-dark" :span="6">col-6</Col>
    </Row>
    <Row>
      <Col class="gird" :span="4">col-4</Col>
      <Col class="gird gird-dark" :span="4">col-4</Col>
      <Col class="gird" :span="4">col-4</Col>
      <Col class="gird gird-dark" :span="4">col-4</Col>
```

```
      <Col class="gird" :span="4">col-4</Col>
      <Col class="gird gird-dark" :span="4">col-4</Col>
    </Row>
    <Row>
      <Col class="gird" :span="3">col-3</Col>
      <Col class="gird gird-dark" :span="3">col-3</Col>
      <Col class="gird" :span="3">col-3</Col>
      <Col class="gird gird-dark" :span="3">col-3</Col>
      <Col class="gird" :span="3">col-3</Col>
      <Col class="gird gird-dark" :span="3">col-3</Col>
      <Col class="gird" :span="3">col-3</Col>
      <Col class="gird gird-dark" :span="3">col-3</Col>
    </Row>
    <Row>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
      <Col class="gird" :span="2">col-2</Col>
      <Col class="gird gird-dark" :span="2">col-2</Col>
    </Row>
</template>

<script setup lang="ts">
import { Col, Row } from '../src';
</script>

<style lang="less">
.gird {
  background: #0092ff6b;
  border: 1px #0092ff6b solid;
  margin-bottom: 10px;
  height: 50px;
  color: #035593;
  font-size: 18px;
  // font-weight: bolder;
  text-align: center;
  line-height: 50px;

  &.gird-dark {
    background: #068aed;
    border: 1px #068aed solid;
    color: #ffffff;
    margin-bottom: 10px;
  }
}
```

```
77 </style>
78
```


最后实现效果如图：



不过，不知道你有没有这样的疑问：为什么栅格组件按比例等分？而不是固定尺寸，例如固定 20px 等分呢？

这是因为**组件库需要有通用性，如果限死栅格单元的固定“格子”尺寸，就不能适用大部分前端开发场景**。这里按比例等分，可以基于“父容器”的固定尺寸，来控制子容器的栅格尺寸。也就是说，如果你想显示宽度为 300px 和 100px 两个子容器，就可以设置父容器 Row 宽度 400px，两个子容器 Col 的栅格 span 参数配置为 span=18 和 span=6。

现在我们就实现了最基础的布局组件，但仅仅靠这个，要实现具体页面布局还是不够的，我们还需要依赖具体有一定语义化的布局组件来组合实现。语义场景下的布局组件，最常见的就是 PC 页面语义化布局组件，

如何实现 PC 端布局组件？

首先需要哪些 PC 布局组件呢？常见的有 5 种：

- 页面容器组件——Layout；

- 页面头部组件——Header；
- 页面内容组件——Content；
- 页面侧边组件——Sider；
- 页面底部组件——Footer。

每个业务组件实现代码，跟前面栅格组件类似，根据不同语义结合 CSS 的 Flex 布局方式，实现对应的“格子”样式就好。有了基础布局组件栅格组件，我们实现其它布局组件会方便许多，跟“搭积木”一样，实现 PC 布局组件，就是基于栅格组件来“搭积木”。

其中 Layout 容器组件比较特殊，我们具体看下。

因为要考虑到有侧边组件时候，内部会从默认的纵向排列变成横向排列。这时候布局容器组件就需要有个侧边栏的控制参数 hasSider 来做判断处理，代码实现如下所示。

Vue 代码：

复制代码

```
<template>
  <section
    :class="{
      [className]: true,
      [hasSiderClassName]: props.hasSider
    }"
  >
    <slot v-if="$slots.default"></slot>
  </section>
</template>
<script lang="ts" setup>
import { prefixName } from '../theme';
const props = defineProps<{ hasSider?: boolean }>();
const className = `${prefixName}-layout`;
const hasSiderClassName = `${className}-has-sider`;
</script>
```

Less 代码：

复制代码

```
@import '../../theme/variable.less';

```

```less
.@{prefix-name}-layout {
  display: flex;
  flex-direction: column;
  flex: auto;
  box-sizing: border-box;
  min-height: 0;
  margin: 0;
  padding: 0;

  &.@{prefix-name}-layout-has-sider  {
    flex-direction: row;
  }
}
```


剩下的 Header、Content、Footer 和 Sider 组件，实现代码都比 Layout 简单，去掉 hasSider 判断逻辑，保留 Flex 样式格式，再使用对应语义化 HTML 标签就可以使用，具体可以你看课后的完整代码案例。

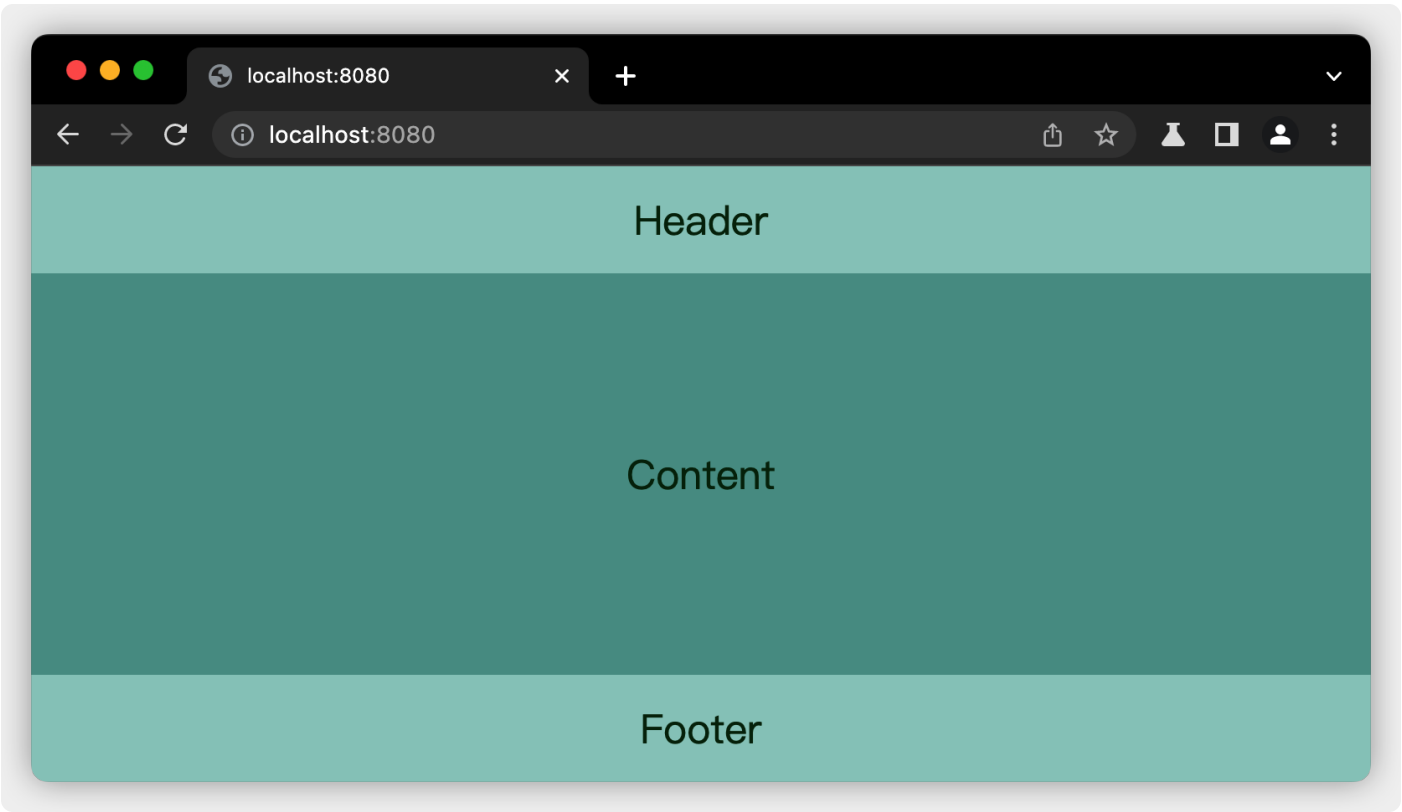
实现所有语义化的布局组件后，我们就可以用 PC 布局组件来组合实现想要的布局，例如这个使用案例代码：

复制代码

```vue
<template>
  <Layout class="example">
    <Header class="example-header">Header</Header>
    <Content class="example-content">Content</Content>
    <Footer class="example-footer">Footer</Footer>
  </Layout>
</template>

<script setup lang="ts">
import { Layout, Header, Footer, Content } from '../src';
</script>

<style lang="less">
.example {
  width: 100%;
  height: 100%;
  margin: 0 auto;
  margin-top: 10px;
  font-size: 24px;
  color: #02290a;
  text-align: center;
}

.example-header {
  background: #00968880;
```

```
  height: 40px;
  justify-content: center;
  align-items: center;
}

.example-content {
  background: #007a6ec0;
  height: 100px;
  justify-content: center;
  align-items: center;
}

.example-footer {
  background: #00968880;
  height: 40px;
  justify-content: center;
  align-items: center;
}

.example-sider {
  background: #106d64d6;
  width: 200px;
  justify-content: center;
  align-items: center;
}
</style>
```

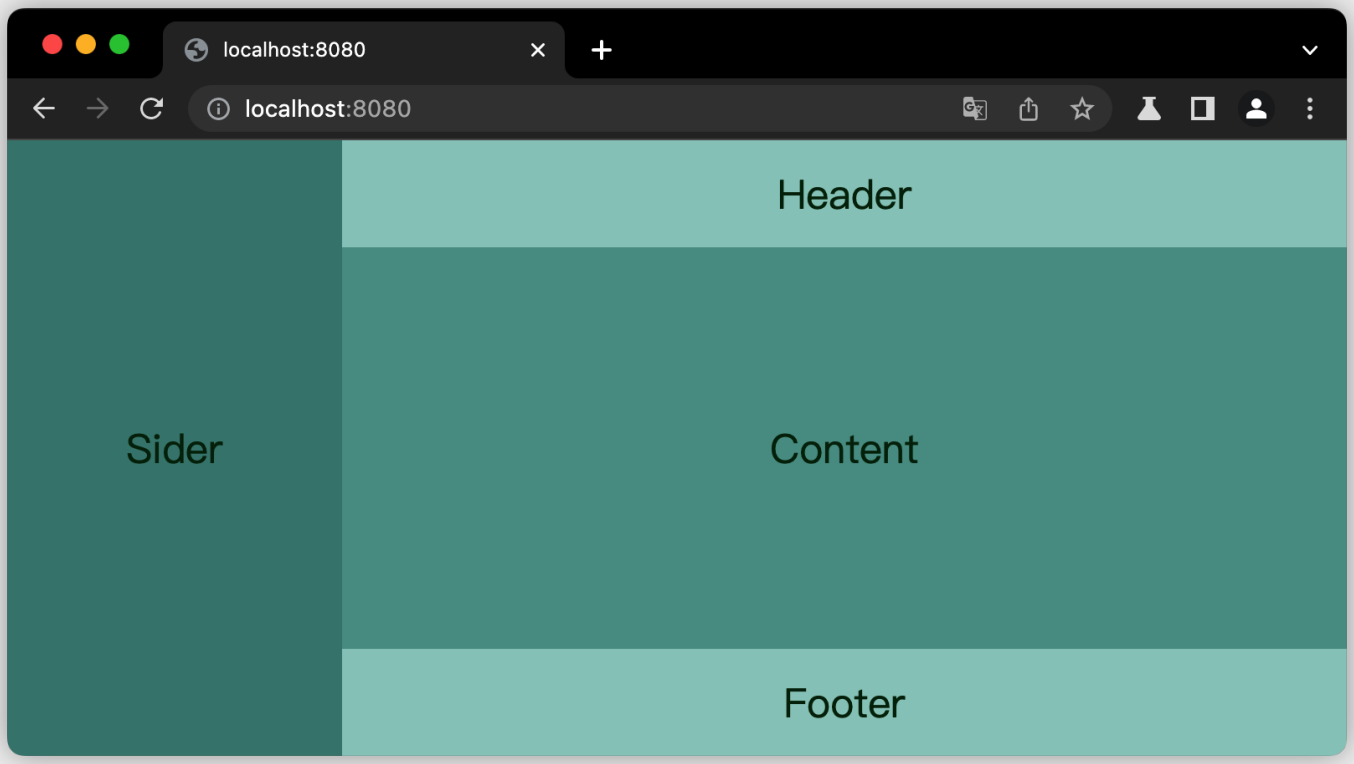

上述代码在浏览器渲染效果如下图所示：



我们把布局可以调整一下，添加 Sider 组件，调整成下布局排序：


```
<Layout class="example" :hasSider="true">
  <Sider class="example-sider">Sider</Sider>
  <Layout>
    <Header class="example-header">Header</Header>
    <Content class="example-content">Content</Content>
    <Footer class="example-footer">Footer</Footer>
  </Layout>
</Layout>
```

可以显示成这个布局效果，如下图所示：



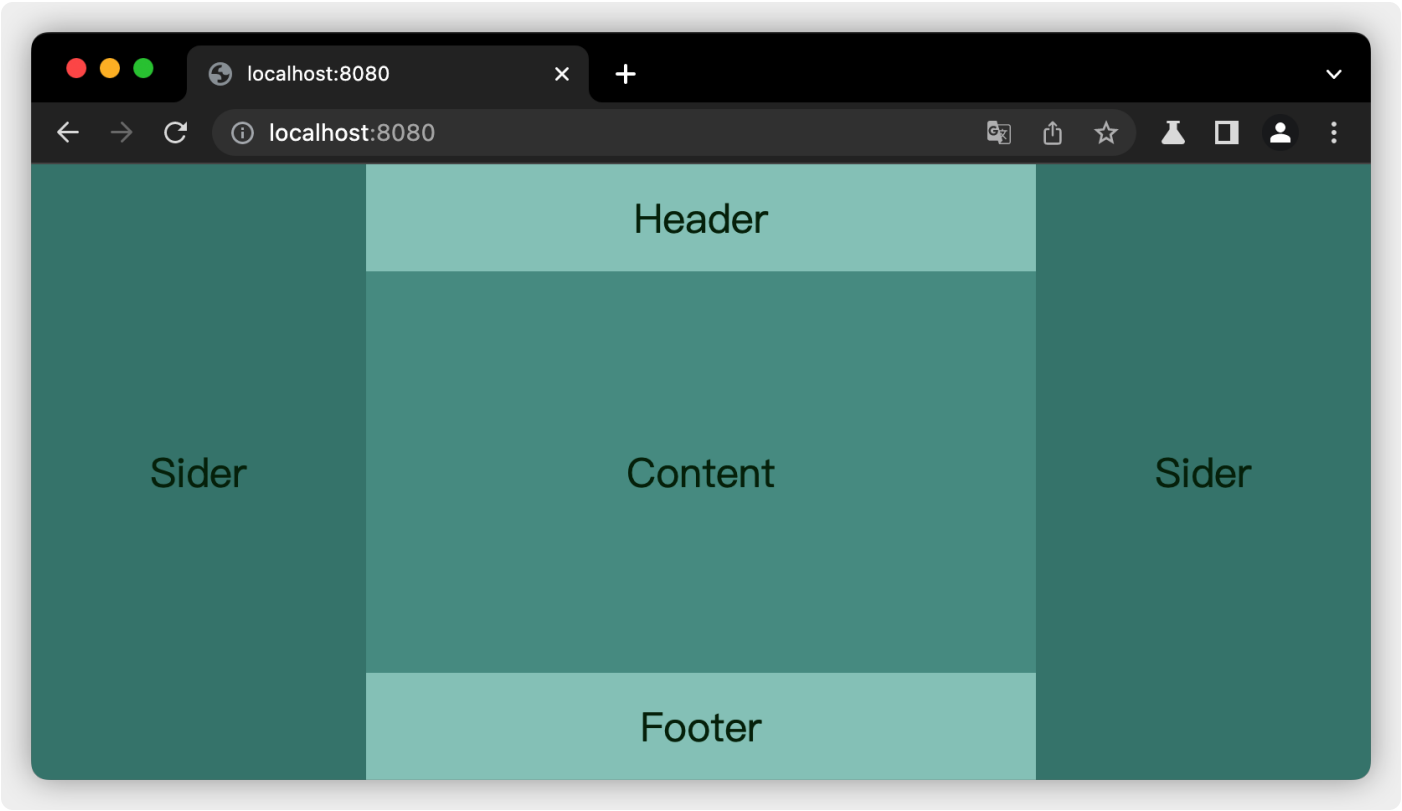
我们还可以再调整一下布局，换成双侧边栏形式，代码如下所示：

复制代码

```
<Layout class="example" :hasSider="true">
  <Sider class="example-sider">Sider</Sider>
  <Layout>
    <Header class="example-header">Header</Header>
    <Content class="example-content">Content</Content>
    <Footer class="example-footer">Footer</Footer>
  </Layout>
  <Sider class="example-sider">Sider</Sider>
```

```
9  </Layout>
```


最后的布局效果，如下图所示：



我们再结合栅格组件一起使用，可以搭建更复杂的布局，如下图所示：

通过 PC 布局组件和栅格组件，结合实现了页面布局后，你就可以在这个“布局骨架”中填充你需要的页面功能组件或代码了。

如果你想改变页面布局，直接调整布局组件排布就行。如果你在工作中需要用到其他 PC 布局类型，甚至是移动端页面的布局组件，可以按照今天的实现方式举一反三。

说到这里，不知道你有没有发现一个隐患，**布局组件内填充的功能组件尺寸实现是不可控的，**这也是我们自研组件库中最容易出现的一个问题。

比如，布局内部填充的功能组件，写死了固定尺寸，有一天你又要减少这个布局组件栅格，或者调整语义化布局组件的一些尺寸，那么里面的功能组件就会“撑出”这个布局组件，或者在布局组件里有大量“留白”。

当然这里只是一种可能的问题场景，实际开发中还有很多其他布局组件调整后与功能组件的尺寸发生冲突的问题，这个从组件技术实现角度上是无法限制的，所以目前只能在组件的使用角度来约定一些规范。

总结一下，在自研组件库的时候，我们需要围绕着内部填充的功能组件来做若干开发规范约定：

- 组件尽量不要写死尺寸，需要用弹性的尺寸样式，建议直接使用栅格组件；
- 组件里的文字内容要考虑换行处理，特别是纯字母和数字的显示组件；
- 组件里图片显示尽量在控制好边缘尺寸限制；
- 组件里绝对定位注意固定位置的偏移是否受外界布局组件容器的影响。

总结

你已经掌握了 Vue.js 3.x 自研组件库的布局组件的规范设计、技术实现，最后我们做个总结。

布局组件的规范设计：

- 需要对页面的做栅格化系统设计；
- 栅格化可以选择主流的 12 等分或 24 等分；

- 基于栅格化设计，根据不同场景来设计不同语义的布局组件。

布局组件的技术实现流程：

- 基于栅格化系统设计来实现栅格组件，作为布局基础组件；
- 再实现不同类型语言的布局组件并结合栅格组件搭配使用。

企业内的需求变化是不可预测的，所以你也无法预测会遇到什么布局需求，但是学会今天的布局组件的设计规范和技术实现，即使你遇到特殊布局场景，也可以根据 PC 端页面布局组件实现和多种布局组合搭配，举一反三，实现自己所需要的布局内容。

思考题

如何实现一套布局组件方式，同时兼容 PC 页面和移动端页面的布局？

期待看到你的思考，如果觉得今天的内容对你有帮助，也欢迎分享给身边的朋友一起学习。我们下节课见。

完整的代码在这里

上一篇　10 | 动态渲染组件：如何实现Vue的动态渲染组件？

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。