第 5 章

JS UI

5.1 关于 JS UI

5.1.1 JS UI 框架介绍

JS UI 框架是一种跨设备的高性能 UI 开发框架，支持声明式编程和跨设备多态 UI。它的基础能力主要体现在 3 个方面：

1）声明式编程

JS UI 框架采用类 HTML 和 CSS 声明式编程语言作为页面布局和页面样式的开发语言，让开发者避免编写 UI 状态切换的代码，页面业务逻辑则支持 ECMAScript 规范的 JS 语言。

2）跨设备

开发框架架构上支持 UI 跨设备显示能力，运行时自动映射到不同设备类型，开发者无感知，从而降低开发者多设备适配成本。

3）高性能

开发框架包含了许多核心的控件，如列表、图片和各类容器组件等，针对声明式语法进行了渲染流程的优化。

JS UI 整体架构如图 5.1 所示，包括应用层（Application）、前端框架层（Framework）、引擎层（Engine）和平台适配层（Porting Layer）。

1）Application

应用层表示开发者使用 JS UI 框架开发的 FA 应用，这里的 FA 应用特指 JS FA 应用。

2）Framework

前端框架层主要完成前端页面解析，以及提供 MVVM（Model-View-ViewModel）开发模式、页面路由机制和自定义组件等能力。

3）Engine

引擎层主要提供动画解析、DOM（Document Object Model）树构建、布局计算、渲染命令构建与绘制、事件管理等能力。

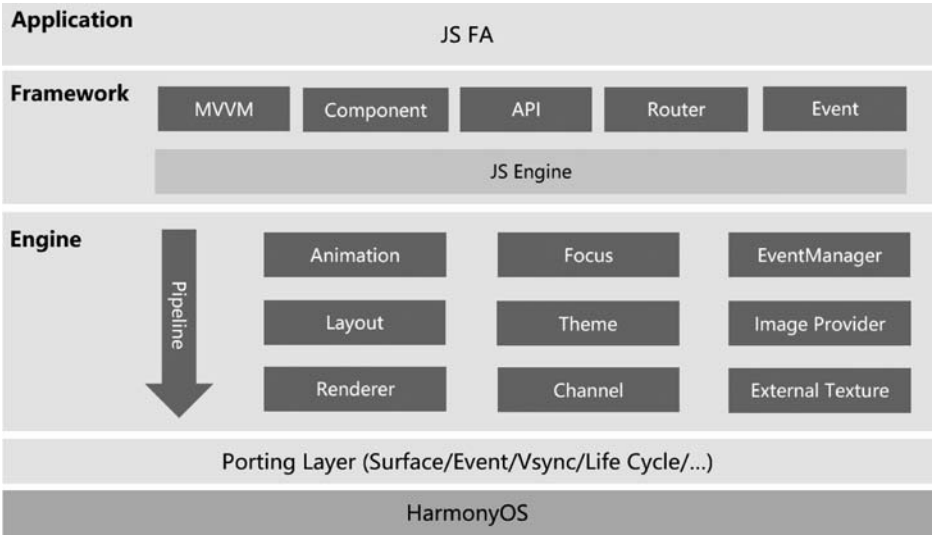

图 5.1 JS UI 整体架构图

4) Porting Layer

适配层主要完成对平台层进行抽象,提供抽象接口,可以对接到系统平台。例如:事件对接、渲染管线对接和系统生命周期对接等。

5.1.2 JS UI 主体介绍

JS UI 框架支持纯 JS、JS 和 Java 混合语言开发。JS FA 指基于 JS 或 JS 和 Java 混合开发的 FA。

新建一个工程,选择 Phone 设备下的 Empty Feature Ability (JS)模板,输入工程名称和包名。新建后的 entry 包结构如图 5.2 所示。

图 5.2 JS 工程目录

可以看出其中包含 java 和 js 两个文件夹，选择 entry→src→main→java→包名→MainAbility，可以看到自动创建的代码如下：

```
//MainAbility 中的示例代码
public class MainAbility extends AceAbility {
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
    }

    @Override
    public void onStop() {
        super.onStop();
    }
}
```

其中，JS FA 在 HarmonyOS 上运行时，必须需要基类 AceAbility，其继承自 Ability 类，所有应用运行入口类都应该从 AceAbility 类派生。

应用通过 AceAbility 类中 setInstanceName()接口设置该 Ability 的实例资源，并通过 AceAbility 窗口进行显示及全局应用生命周期管理，因此当加载 JS FA 时，需要通过 setInstanceName(String name)的参数 name 指明实例名称，实例名称与 config.json 文件中 profile.application.js.name 的值对应。若开发者未修改实例名，而使用了默认值 default，则无须调用此接口，如上述代码，新建工程时默认实例名为 default，因此不需要调用 setInstanceName()接口。若开发者修改了实例名，则需要在应用 Ability 实例的 onStart()中调用此接口，并将参数 name 设置为修改后的实例名称。多实例应用的 profile.application.js 字段中有多个实例项，使用时应选择相应的实例名称。如实例名称为 JSComponentName 时，必须在 super.onStart(Intent)前调用此接口，代码如下：

```
//MainAbility.java
public class MainAbility extends AceAbility {
    @Override
    public void onStart(Intent intent) {
setInstanceName("JSComponentName");     //参数为 config.json 配置文件中
module.js.name 的标签值
        super.onStart(intent);
    }
}
```

选择 entry→src→main→js，工程的 JS FA 开发目录如图 5.3 所示。

这里对每个文件进行具体介绍。

(1) i18n 文件夹用于存放多语言的 json 文件。en-US.json 文件定义了在英文模式下页面显示的变量内容，zh-CN.json 文件定义了中文模式下的页面内容。如 en-US.json 文件的代码如下：

图 5.3 JS FA 开发目录

```
{
  "strings": {
    "hello": "Hello",
    "world": "World"
  },
  "Files": {
  }
}
```

(2) pages 文件夹用于存放多个页面,每个页面由 hml、css 和 js 文件组成。

index.hml 文件定义了 index 页面的布局、index 页面中用到的组件,以及这些组件的层级关系。以下面的代码为例,包含了一个 text 组件,内容为文本 Hello World,其中{{title}}采用了变量赋值的方式,在 index.js 文件中进行赋值,代码如下:

```
<div class = "container">
<text class = "title">
        {{ $t('strings.hello') }} {{title}}
</text>
</div>
```

index.css 文件定义了 index 页面的样式。如下面代码所示,该 css 文件定义了 index.hml 文件中 class=container 和 class=title 的容器或组件样式,代码如下:

```
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
}

.title {
    font-size: 100px;
}
```

index.js 文件定义了 index 页面的业务逻辑，例如数据绑定、事件处理等。为 index.hml 文件中的变量 title 赋值字符串 World，代码如下：

```
export default {
    data: {
        title: ""
    },
    onInit() {
        this.title = this.$t('strings.world');
    }
}
```

5.2 开发第一个 JS FA 应用

5.2.1 页面布局说明

本节将逐步介绍如何开发一个 JS FA 应用。一个页面的基本元素包含标题区域、文本区域、图片区域等，每个基本元素内还可以包含多个子元素，开发者根据需求还可以添加按钮、开关、进度条等组件。在构建页面布局时，需要对每个基本元素思考以下几个问题：

(1) 该元素的尺寸和排列位置。

(2) 是否有重叠的元素。

(3) 是否需要设置对齐、内间距或者边界。

(4) 是否包含子元素及其排列位置。

(5) 是否需要容器组件及其类型。

在进行代码开发之前，首先要对整体页面布局进行分析，将页面分解为不同的部分，用容器组件来承载。将页面中的元素分解之后再对每个基本元素按顺序实现，可以减少多层嵌套造成的视觉混乱和逻辑混乱，提高代码的可读性，方便对页面进行后续的调整。应用的分解效果图如图 5.4 所示，其中，最上方的图片区可以通过滑动来观看不同的图片，中间的标题区可以进行收藏和取消收藏，最下方则为描述区，可以进行文字介绍。

根据 JS FA 应用效果图，此页面一共分成 3 个部分：图片区、标题区、描述区。根据此分区，根节点的子节点应按列排列。

图片区和描述区分别使用 swiper 组件和 text 组件实现。标题区由两部分组成，以行来排列，其中

图 5.4 JS FA 应用效果图

第一部分由两个 text 组件组成，分别为商品名称和商品标语，以列排列。第二部分由 image 组件和 text 组件组成，分别为代表收藏功能的星号和代表收藏次数的数字，以行排列，如图 5.5 所示。

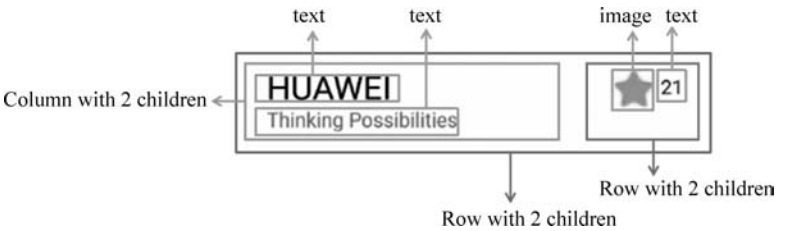

图 5.5 JS FA 标题区布局分析

5.2.2 构建布局

根据布局结构的分析，首先构建页面的基础布局。其中，实现图片区域通常用 image 组件实现，由于需要左右滑动图片，因此在 image 外层加入 swiper 滑动容器，swiper 容器提供了切换子组件显示的能力。图片资源放在 common 目录下，图片的路径要与图片实际所在的目录一致，需要注意，common 需要新建，级别须与 pages 目录平级。

标题区和描述区采用了最常用的基础组件 text，text 组件用于展示文本，文本内容需要写在标签内容区。要将页面的基本元素组装在一起，需要使用容器组件，如上述 swiper 容器。在页面布局中常用到 3 种容器组件，分别是 div、list 和 tabs。在页面结构相对简单时，可以直接用 div 作为容器，因为 div 作为单纯的布局容器使用起来更为方便，可以支持多种子组件。

在 index.hml 文件中实现页面基础布局，具体代码如下：

```
<!-- index.hml 文件代码示例 -->
<div class="container">
<swiper class="swiper-style">
<image src="/common/Phone_00.jpg" class="image-mode"></image>
<image src="/common/Phone_01.jpg" class="image-mode"></image>
<image src="/common/Phone_02.jpg" class="image-mode"></image>
<image src="/common/Phone_03.jpg" class="image-mode"></image>
</swiper>
<div class="title-section">
<div class="phone-title">
<text class="phone-name">
                HUAWEI
</text>
<text class="phone-definition">
                Thinking Possibilities
</text>
```

```
</div>
<div class = "favorite - image">
<image src = "{{unFavoriteImage}}" class = "image - size"onclick = "favorite"></image>
</div>
<div class = "favorite - count">
<text>{{number}}
</text>
</div>
</div>
<div class = "description - first - paragraph">
<text class = "description">{{descriptionFirstParagraph}}
</text>
</div>
<div class = "description - second - paragraph">
<text class = "description">{{descriptionSecondParagraph}}
</text>
</div>
</div>
```

在 index.hml 中，为每个组件和容器都定义了一个 class＝" *** "的样式，需要在 css 文件中依次对样式进行构建。如本例，在 index.css 文件中，需要设定的样式主要有：flex-direction 用于设置子组件容器的排列方式，paddind 用于设置内边距，font-size 用于设置字体大小，以及 swiper 组件的一些私有属性，如 indicator-color 用于设置导航点指示器的填充颜色，indicator-selected-color 用于设置导航点指示器选中的颜色，indicator-size 用于设置导航点指示器的直径大小等。具体代码如下：

```
/* index.css 文件代码示例 */
.container {
    flex - direction: column;
}
.swiper - style {
    height: 700px;
    indicator - color: #4682b4;
    indicator - selected - color: #ffffff;
    indicator - size: 20px;
}
.title - section {
    flex - direction: row;
    height: 150px;
}
.phone - title {
    align - items:flex - start;
    flex - direction: column;
    padding - left: 60px;
```

```
    padding-right: 160px;
    padding-top: 50px;
}
.phone-name {
    font-size: 50px;
    color: #000000;
}
.phone-definition {
    font-size: 30px;
    color: #7A787D;
}
.favorite-image {
    padding-left: 70px;
    padding-top: 50px;
}
.favorite-count {
    padding-top: 60px;
    padding-left: 10px;
}
.image-size {
    object-fit: contain;
    height: 60px;
    width: 60px;
}
.description-first-paragraph {
    padding-left: 60px;
    padding-top: 50px;
    padding-right: 60px;
}
.description-second-paragraph {
    padding-left: 60px;
    padding-top: 30px;
    padding-right: 60px;
}
.description {
    color: #7A787D;
}
.image-mode {
    object-fit: contain;
}
```

在 index.hml 中，收藏的图片来源采用了 src="{{unFavoriteImage}}"，text 标签也采用了{{descriptionFirstParagraph}}和{{descriptionSecondParag -raph}}的数据绑定形式，所以需要在 index.js 中对其进行赋值。index.js 代码如下：

```
//index.js 文件代码示例
export default {
    data: {
        unFavoriteImage: "/common/unfavorite.png",
        isFavorite: false,
        number: 20,
        descriptionFirstParagraph:"The breakthrough of visual boundaries, the exploration of photography and videography, the liberation of power and speed, and the innovation of interaction are now ready to be discovered. Embrace the future with new possibilities.",
        descriptionSecondParagraph: "Lighting up infinite possibilities. The quad camera of HUAWEI is embraced by the halo ring. It is a perfect fusion of reflection and refraction. Still Mate, but a new icon.",
    },
}
```

以上代码完成了基础的布局构建,接下来对页面中的收藏交互进行实现。

5.2.3 添加交互

添加交互通过在组件上关联事件实现,本节将介绍如何关联 click 事件,构建上述页面中的收藏功能,即单击星星图片,图片变成黄色,表示收藏,收藏数加 1,如图 5.6 所示。再次单击,星号恢复成原本颜色,表示取消收藏,收藏数减 1。

收藏前　　收藏后

图 5.6　收藏过程分析

收藏按钮通过一个 div 组件关联 click 事件实现。div 组件包含一个 image 组件和一个 text 组件。image 组件用于显示未收藏和收藏后的效果。click 事件函数会交替更新收藏和未收藏图片的路径。text 组件用于显示收藏数,收藏数也会在 click 事件的函数中同步更新。

index.js 文件用于构建页面逻辑,click 事件作为一个函数定义在 index.js 文件中,可以更改 isFavorite 的状态,从而更新显示的 image 组件和 text 组件。如果 isFavorite 为真,则更改收藏后的图片路径,并将点赞数加 1。该函数在 hml 文件中对应的 div 组件上生效。在 index.js 中加入代码如下:

```
//index.js 文件代码实现
export default {
    data: {
…
    },

    favorite() {
```

```
            var tempTotal;
            if (!this.isFavorite) {
                this.unFavoriteImage = "/common/favorite.png";
                tempTotal = this.number + 1;
            } else {
                this.unFavoriteImage = "/common/unfavorite.png";
                tempTotal = this.number - 1;
            }
            this.number = tempTotal;
            this.isFavorite = !this.isFavorite;
        }
    }
```

运行程序,实现效果如图 5.7 所示。

顶部的 swiper 图片可进行左右滑动,且中间的标题区可进行收藏操作,收藏后效果如图 5.8 所示。

图 5.7 JS FA 运行效果图

图 5.8 JS FA 收藏运行效果图

5.3 常用组件

16min

组件(Component)是构建页面的核心,每个组件通过对数据和方法的简单封装,实现独立的可视、可交互功能单元。组件之间相互独立,随取随用,也可以在需求相同的地方重复

使用。开发者还可以通过组件间合理的搭配定义满足业务需求的新组件，从而减少开发工作量。

根据组件的功能，可以将组件分为以下四大类，如表 5.1 所示。

表 5.1　组件的分类

组 件 类 型	主 要 组 件
基础组件	text、image、progress、rating、span、marquee、image-animator、divider、search、menu、chart
容器组件	div、list、list-item、stack、swiper、tabs、tab-bar、tab-content、popup、list-item-group、refresh、dialog
媒体组件	video
画布组件	canvas

合理使用控件可以编写出丰富多样的界面，下面介绍几种常用控件的使用方法。

5.3.1　基础组件

1. Text

实现标题和文本区域最常用的是基础组件 Text。Text 组件用于展示文本，可以设置不同的属性和样式，文本内容需要写在标签内容区。在页面中插入标题和文本区域的代码如下：

```
<!-- index.hml 实现在页面中插入标题和文本 -->
<div class = "container">
<!-- 标题区域 -->
<text class = "title_text">{{headTitle}}</text>
<!-- 第一段文字 -->
<div class = "paragraph">
<text class = "paragraph_text">{{paragraphFirst}}</text>
</div>
<!-- 第二段文字 -->
<div class = "paragraph">
<text class = "paragraph_text">{{paragraphSecond}}</text>
</div>
</div>

<!-- index.js -->
export default {
    data: {
        headTitle:"HUAWEI",
        paragraphFirst:"The breakthrough of visual boundaries, the exploration of photography
and videography, the liberation of power and speed, and the innovation of interaction are now
ready to be discovered. Embrace the future with new possibilities.",
```

```
        paragraphSecond: "Lighting up infinite possibilities. The quad camera of HUAWEI is
embraced by the halo ring. It is a perfect fusion of reflection and refraction. Still Mate, but a
new icon."
    },
}
<!-- index.css -->
.container{
    flex-direction: column;
}
.title_text{
    align-items:flex-start;
    flex-direction: column;
    padding-left: 60px;
    padding-right: 160px;
    padding-top: 50px;
    font-size: 50px;
}
.paragraph{
    padding-left: 60px;
    padding-top: 50px;
    padding-right: 60px;
}
```

示例效果图如图 5.9 所示。

图 5.9 Text 组件示例效果图

2. Button

Button 是实现用户交互最常用的组件，可用于触发 JS 中的函数。可以使用 JS 提供的各种按钮（包括胶囊按钮、圆形按钮、文本按钮、弧形按钮、下载按钮）实现很多有趣的功能。例如，在 hml 文件中定义上述各种按钮，此外，在圆形按钮中加入了一张图标，将下载按钮的 onClick 属性绑定到了 js 文件中的 setProgress 函数中，将第二个胶囊按钮的 waiting 属性定义为 true，让它一直处于等待状态。实现代码如下：

```
<! -- index.hml 定义多种按钮 -- >
< div class = "div - button">
< button class = "button" type = "capsule" value = "胶囊按钮"></button >
< button class = "button circle" type = "circle" icon = "common/logo.png"></button >
< button class = "button text" type = "text">文本按钮</button >
< button class = "button download" type = "download" id = "download - btn"
            onClick = "setProgress">{{downloadText}}</button >
< button class = "button" type = "capsule" waiting = "true">载入中...</button >
< button class = "button" type = "arc">弧形按钮</button >
</div >
```

上述下载按钮的 onClick 属性绑定到了 js 文件中的 setProgress 函数中，js 文件的代码如下：

```
//index.js
export default {
    data: {
        progress: 5,
        downloadText: "下载"
    },
    setProgress(e) {
        this.progress += 10;                    //每次单击增加 10 % 的进度
        this.downloadText = this.progress + " % ";
        if (this.progress > = 100) {
            this.downloadText = "完成";         //到达 100 % 时显示完成
        }
    }
}
```

css 文件中定义了按钮的样式，具体实现代码如下：

```
/ * index.css 定义按钮的样式 * /
.div - button {
    flex - direction: column;
    align - items: center;
}
```

```
.button {
    margin-top: 15px;
}
.button:waiting {
    width: 280px;
}
.circle {
    background-color: #007dff;
    radius: 72px;
    icon-width: 72px;
    icon-height: 72px;
}
.text {
    text-color: red;
    font-size: 40px;
    font-weight: 900;
    font-family: sans-serif;
    font-style: normal;
}
.download {
    width: 280px;
    text-color: white;
    background-color: #007dff;
}
```

运行上述代码,最终的效果如图 5.10 所示。

图 5.10 Button 组件示例效果图

3. Menu

Menu 提供菜单组件，作为临时性弹出窗口，用于展示用户可执行的操作。当菜单中某个值被单击选中时，selected 事件将会被触发，同时返回 value 的值。在页面中插入菜单组件的示例代码如下：

```
<!-- index.hml 在页面中插入 Menu 组件 -->
<div class="container">
<text onclick="onTextClick" class="title-text">Show popup menu.</text>
<menu id="apiMenu" onselected="onMenuSelected">
<option value="Item 1">Item 1</option>
<option value="Item 2">Item 2</option>
<option value="Item 3">Item 3</option>
</menu>
</div>

/* index.js */
import prompt from '@system.prompt';
export default {
    onMenuSelected(e) {
        prompt.showToast({
            message: e.value
        })
    },
    onTextClick() {
        this.$element("apiMenu").show({x:280,y:120});
    }
}
```

运行上述代码，示例效果图如图 5.11 所示，可以看到实现了最简单的菜单效果。

图 5.11　Menu 组件示例效果图

5.3.2 List 组件

现如今，购物 App 已经成为生活中不可或缺的一部分，滑动浏览的商品列表是必不可少的基础组件，如图 5.12 所示。在本节将介绍 JS UI 中的 List 组件，用于呈现多行连续同类的数据。

图 5.12 List 组件的实际应用实例

List 列表组件包含两类子组件，分别是 list-item 列表项和 list-item-group 列表项组。下面对两个子组件分别进行介绍。

1. list-item 列表项

这里先举一个简单的 list-item 例子，在 hml 文件中，编写代码如下：

```
<!-- list-item 示例 -->
<div class="container">
<list>
<list-item><text>列表项文本 1</text></list-item>
<list-item><text>列表项文本 2</text></list-item>
<list-item><text>列表项文本 3</text></list-item>
<list-item><text>列表项文本 4</text></list-item>
```

```
<list-item><text>列表项文本 5</text></list-item>
<list-item><text>列表项文本 6</text></list-item>
<list-item><text>列表项文本 7</text></list-item>
<list-item><text>列表项文本 8</text></list-item>
<list-item><text>列表项文本 9</text></list-item>
<list-item><text>列表项文本 10</text></list-item>
<list-item><text>列表项文本 11</text></list-item>
<list-item><text>列表项文本 12</text></list-item>
        ...
<list-item><text>列表项文本 32</text></list-item>
</list>
</div>
```

运行上述代码,效果如图 5.13 所示。

图 5.13　List 组件示例效果图

示例中枚举了 32 个列表项文本,一般情况下,都会在 js 文件中,通过数组的形式来存储列表项文本内容,代码如下:

```
<!-- index.hml -->
<div class="container">
<list>
<list-item><text>{{data}}</text></list-item>
</list>
</div>
```

```
/* index.js */
export default {
    data: {
        title: "",
        data : ["列表项文本 1","列表项文本 2","列表项文本 3","列表项文本 4"...,"列表项文
本 32"]
    }
}
```

事实上,32 个列表项文本长度已经超过了屏幕长度,超出部分被自动隐藏起来,当手指滑动屏幕时显示剩余部分,这一过程一般称为 scroller。在 list-item 属性设置中还有几个比较常用的属性:

(1) scrollbar 用于设置侧边滑动栏的显示模式,默认值为 off,即不显示,另外还可以设置为 on(表示常驻显示)和 auto(表示按需显示)。当设置为 auto 时,触摸屏幕时会显示滑动条且 2s 后自动消失。

(2) scrolleffect 用于显示滑动效果,默认值为 spring,是一个弹性物理动效,当滑动到边缘后可以根据初始速度或通过触摸事件继续滑动一段距离,然后松手后回弹。另外还可以设置为 fade,实现渐隐动效,活动到边缘后展示一个波浪形的渐隐,根据速度和滑动距离的变化,渐隐也会发生一定的变化,也可以设置为 no,即不设置滑动边缘效果。

(3) indexer 用于展示侧边栏快速字母索引栏,一般需要配合 item 中的 section 属性一起使用。可能很多读者会对这个快速字母索引栏有些不解,不知为何物。其实这个索引栏类似于计算机文件中的按类型分类,可以将相同类型的文件分为一类,这里的索引栏也有异曲同工之妙,根据 section 中设置的值,将相同 section 的条目分为一类。indexer 的参数可以为 true,表示使用默认字母索引表,也可以为 false,表示无索引,还可以自定义索引表,这里值得注意的是在自定义时“#”必须存在。实现自定义索引表的代码如下:

```
<!-- index.hml -->
<div class="container">
<list scrollbar="on" scrolleffect="fade" indexer="{{index}}">
<list-item section="a"><text>列表项文本 1</text></list-item>
<list-item section="b"><text>列表项文本 2</text></list-item>
<list-item section="c"><text>列表项文本 3</text></list-item>
<list-item section="d"><text>列表项文本 4</text></list-item>
<list-item section="a"><text>列表项文本 1</text></list-item>
<list-item section="b"><text>列表项文本 2</text></list-item>
<list-item section="c"><text>列表项文本 3</text></list-item>
<list-item section="d"><text>列表项文本 4</text></list-item>
    ...
</list>
</div>
```

上述代码中可以看到 indexer 需要配合 section 共同使用，indexer 设置的是 index 数组，由用户自定义，section 设置的是该 list-item 所属的索引列。index 数组定义在 js 文件中，代码如下：

```
/* index.js */
export default {
    data: {
        title: "",
        index : ["#","a","b","c"]
    }
}
```

上述代码的执行效果如图 5.14 所示。

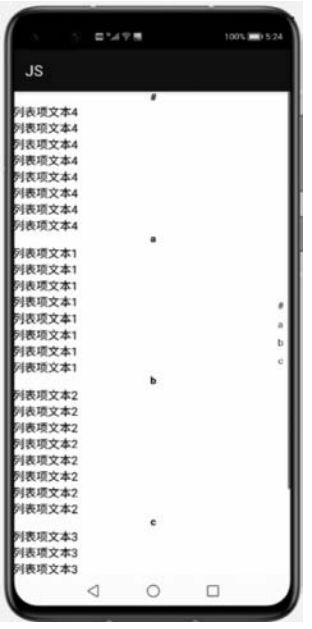

图 5.14　带索引的 List 组件示例效果图

此外，除了按照索引分类，indexer 还会在侧边栏添加索引值查找的功能，可以单击跳转到具体的索引列表。

(4) updateeffect 属性用于设置当 list 内部的 item 发生删除或新增时是否支持动效。当设置为 true 时，新增或删除 item 时播放过程动效。

2. list-item-group 列表项组

list-item-group 是一个具有折叠效果 list 列表，配合 list-item 使用，以 list-item-group 中的第一个 list-item 作为分类标准，以下都折叠进分类中，单击具有折叠和扩展的效果，代码如下：

```
<!-- index.hml -->
<div class="container">
<list>
<list-item-group>
<list-item class="item_style"><text class="big_size">水果</text></list-item>
<list-item><text class="font_size">苹果</text></list-item>
<list-item><text class="font_size">香蕉</text></list-item>
</list-item-group>
<list-item-group>
<list-item class="item_style"><text class="big_size">饮料</text></list-item>
<list-item><text class="font_size">可乐</text></list-item>
<list-item><text class="font_size">咖啡</text></list-item>
</list-item-group>
<list-item-group>
    <list-item class="item_style"><text class="big_size">零食</text></list-item>
<list-item><text class="font_size">薯片</text></list-item>
<list-item><text class="font_size">锅巴</text></list-item>
</list-item-group>
    </list>
</div>

/* index.css */
.item_style {
    background-color: pink;
}
.big_size {
    font-size: 50px;
}
```

运行上述代码,结果如图 5.15 所示,实现了一个具有折叠效果的列表。

图 5.15 可折叠的 List 组件示例效果图

5.3.3 Tabs 组件

有时需要在 App 的界面中滑动或者单击某个特定区域来显示不同的内容，类似于购物 App 中显示推送的功能，这时就需要用到 JS 提供的 Tabs 组件了。Tabs 组件仅包含一个 tab bar 和一个 tab content，其中，tab bar 可以用来显示一些简讯，而 tab content 可以用来显示主要的内容。

接下来开发一个观察华为手机三视图的小程序，在观察某个视图时界面能显示当前视图的名称及内容。其中，当前视图的名称可以用 tab bar 组件来显示，而视图内容可以用 tab content 来显示，hml 文件代码如下：

```
<! -- .hml 文件定义 Tabs 组件 -- >
< div class = "container">
< tabs class = "tabs" vertical = "false" >
< tab - bar class = "tab - bar" mode = "fixed">
< text class = "tab - text"> black </text >
< text class = "tab - text"> orange </text >
< text class = "tab - text"> white </text >
</tab - bar >
< tab - content class = "tab - content" scrollable = "true">
< div class = "item - content" >
< image class = "item - image" src = "/common/1. png"></image >
</div >
< div class = "item - content" >
< image class = "item - image" src = "/common/2. png"></image >
</div >
< div class = "item - content" >
< image class = "item - image" src = "/common/2. jpg"></image >
</div >
</tab - content >
</tabs >
</div >
```

由代码可知，定义了一个 Tabs 组件，并且在 Tabs 组件内定义了一个 tab-bar 组件用于显示各视图的名称(black、orange、white)，还定义了一个 tab-content 组件用于显示 3 个视图。在 tab-bar 组件中定义了 3 个 Text 来显示视图名称，在 tab-content 中定义了 3 个 Image 来显示内容。

此外，还需要在 css 文件中设置具体的样式，css 文件的代码如下：

```
/ * .css 文件代码设置样式 * /
.container {
    flex - direction: column;              / * 调整主轴方向，将容器中的内容按列摆放 * /
    justify - content: flex - start;       / * 将内容放在容器开头 * /
```

```
    align-items: center;                    /* 内容对齐方式 */
}
.tabs {
    width: 100%;
}
.tab-bar {
    margin: 10px;
    height: 60px;
    border-color: #963C71;
    border-width: 1px;
}
.tab-text {
    width: 300px;
    text-align: center;
}
.tab-content {
    width: 100%;
    height: 80%;
    justify-content: center;                /* 将内容放在容器中央 */
}
.item-content {
    height: 100%;
    justify-content: center;
}
.item-image {
    object-fit: contain;                    /* 将内容缩放到全部显示 */
}
```

运行上述代码,效果如图 5.16 所示。

图 5.16 Tabs 组件示例效果图

可以看到定义的 tab-bar 在屏幕上方并显示了视图名称，通过单击 tab-bar 中各部分可以切换视图，同时通过拖动屏幕中内容的方法也可以切换视图（前提是在 tab-content 定义时将 scrollable 属性设置为 true）。

5.3.4 自定义组件

自定义组件是用户根据业务需求，将已有的组件组合起来，封装成新的组件，并作为新组件可以在工程中被多次调用，从而提高代码的可读性和可扩展性。自定义组件通过 element 引入宿主页面，使用方法的示例代码如下：

```
<element name = "comp" src = "../comp.hml"></element>
<div>
<comp prop1 = 'xxxx' @child1 = "bindParentVmMethod"></comp>
</div>
```

（1）name 属性指定自定义组件名称（非必填，若不填组件名称则为 hml 文件名）。src 属性指自定义组件 hml 文件路径，为了准确定位该组件位置，src 属性内容必须填写，需要注意，src 路径中"../"代表上一级目录索引。

（2）prop 属性用于组件之间的通信，可以通过<tag xxxx='value'>方式传递给组件，名称必须用小写。

（3）事件绑定：自定义组件中绑定子组件事件使用（on 或@）child1 语法，子组件中通过 this.$emit('child1', { params: '传递参数' }）触发事件并进行传值，父组件执行 bindParentVmMe-thod 方法并接收子组件传递的参数。

下面尝试创建一个自定义组件，并在父组件中引入自定义组件的事件响应。首先在 page 文件夹中新建自定义组件目录 comp，同时创建自定义组件的基础 hml、css 和 js 文件，代码如下：

```
<!-- 新建 comp.hml 文件 -->
<div class = "item">
<text class = "title_style">{{title}}</text>
<text class = "text-style" onclick = "childClicked">单击这里查看隐藏文本</text>
<text class = "text-style" if = "{{showword}}"> hello world </text>
</div>

/* 新建 comp.css */
.item {
    width: 700px;
    flex-direction: column;
    height: 300px;
    align-items: center;
    margin-top: 100px;
}
```

```
.text-style {
    font-weight: 500;
    font-family: Courier;
    font-size: 40px;
}

/* 新建 comp.js */
export default {
    props: {
        title: {
            default: 'title'
        },
    },
    data: {
        showword: false,
    },
    childClicked () {
        this.$emit('eventType1', {text: "收到子组件参数"});
        this.showword = !this.showword;
    },
}
```

上述代码新创建了一个 comp 自定义组件，组件中设置了一个具有单击属性的文本内容，该单击效果会将自定义组件的数据传递给父组件，并将 show 值从初始化的 false 转换为 true，即显示 hello world 文本。

在父组件中通过 element 引入自定义组件，代码如下：

```
<!-- index.hml -->
<element name = "comp" src = "../comp/comp.hml"></element>
<div class = "container">
<text>父组件：{{text}}</text>
<comp title = "自定义组件" @event-type1 = "textClicked"></comp>
</div>
/* index.js */
export default {
    data: {
        text: "开始"
    },
    textClicked (clicked) {
        this.text = clicked.detail.text;
    },
}
/* index.css */
.container {
    background-color: #f8f8ff;
```

```
        flex: 1;
        flex-direction: column;
        align-content: center;
    }
```

运行上述代码，运行效果如图 5.17 所示。

图 5.17　自定义组件示例效果图

本示例中父组件通过添加自定义属性向子组件传递了名为 title 的参数，自定义子组件在 props 中接收，自定义组件可以通过 props 声明属性，父组件通过设置的属性向子组件传递参数，注意在命名 prop 时需要使用 camelCase 即驼峰式命名法，在外部父组件传递参数时需要使用 kebab-case 即用短横线分割命名，例如在上面示例代码中属性 eventTypel 在父组件引用时需要转换为 event-typel。子组件也通过事件绑定向上传递了参数 text，父组件接收时通过 clicked. detail 获取数据。

父子组件之间的数据传输是单向的，一般只能从父组件传递给子组件，而子组件如果需要向上传递必须绑定事件，通过事件的 $ emit 来传输。子组件获取来自父组件数据后，子组件不能直接修改父组件传递下来的值，可以通过将 props 传入的值用 data 接收后作为默认值，然后再对 data 的值进行修改。

如果需要观察组件中属性的变化，可以通过 $ watch 方法增加属性变化回调，代码如下：

```
//comp.js
export default {
    props: ['title'],
    onInit() {
        this.$watch('title', 'onPropertyChange');
    },
    onPropertyChange(newV, oldV) {
        console.info('title 属性变化 ' + newV + '' + oldV);
    },
}
```

5.4 添加用户交互

5.4.1 手势事件

提到如何与 App 进行交互，首先想到的就是 Button 组件，其实 HarmonyOS 中绝大多数组件可以与用户进行交互，最常用的就是设置组件的 onclick 属性，代码如下：

```
<div class = "click - test" onclick = "Click" vertical = "false">
```

上述代码中，将 click-test 组件的 onclick 属性设置为 Click，这里的 Click 其实是在 js 文件中定义的一个函数，当此组件所包含的区域被单击时就会触发 Click 函数。这里依旧以 5.3.3 节中的 3 种不同颜色的手机为例，代码如下：

```
<! -- .hml 文件代码 -- >
<div class = "container">
<div class = "click - test" onclick = "Click" vertical = "true">
<image class = "image" src = "{{Image}}"></image>
<text class = "image - name">{{Name}}</text>
</div>
</div>
```

在 hml 文件代码中定义了一个 div 组件，并将其 onclick 属性连接到了 js 文件中定义的 Click 函数，在 Click 函数中会根据 tmp 的值来决定每次单击后显示的视图及显示的视图名字，Click 函数在 js 文件中的实现代码如下：

```
//.js 文件代码
export default {
    data: {
        Image: "/common/1.png",                    //最开始显示 black 视图
        Name: "black",
```

```
        tmp: 1,
    },
Click() {
        if ( this.tmp == 0) {
            this.Image = "/common/1.png";              //第 3 次单击跳到第一视图
            this.Name = "black";
        }
        else if ( this.tmp == 1 ) {
            this.Image = "/common/2.png";              //第 1 次单击跳到第二视图
            this.Name = "orange";
        }
        else if ( this.tmp == 2 ) {
            this.Image = "/common/3.png";              //第 2 次单击跳到第三视图
            this.Name = "white";
        }
        this.tmp = ( this.tmp + 1) % 3 ;
    }
}
```

在 css 文件中对布局样式进行简单设计，代码如下：

```
.container {
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
}
.click-test {
    width: 100%;
}
.image {
    height: 80%;
    justify-content: center;
}
.image-name {
    color: #BCBCBC;
    width: 300px;
    text-align: center;
}
```

运行上述代码，当应用程序打开时，开始界面如图 5.18 所示。

当第 1 次单击界面容器之后，图片会进行跳转，从图 5.18 所示效果跳转至图 5.19 所示效果，如图 5.19 所示。

同理，当第 2 次单击之后效果如图 5.20 所示。第 3 次单击之后又会回到开始界面，即图 5.18 所示的效果，之后不断循环。

图 5.18 组件示例效果图

图 5.19 第 1 次手势交互后组件示例效果图

图 5.20 第 2 次手势交互后组件示例效果图

5.4.2 按键事件

按键事件是智慧屏上特有的手势事件，会在用户操作遥控器按键时触发。当用户单击一个遥控器按钮时，通常会触发两次 key 事件：先触发 action 为 0 即触发按下事件，再触发 action 为 1 即手指抬起事件。action 等于 2 的场景较少出现，一般为用户按下按键后不抬起即长按，此时 repeatCount 将返回次数。每个物理按键对应各自的按键 keycode 以实现不同的功能，代码如下：

```
<!-- index.hml -->
<div class="card-box">
<div class="content-box">
<text class="content-text" onkey="keyUp" onfocus="focusUp"
onblur="blurUp">{{up}}</text>
</div>
<div class="content-box">
<text class="content-text" onkey="keyDown" onfocus="focusDown"
onblur="blurDown">{{down}}</text>
</div>
</div>
```

```
/* index.js */
export default {
    data: {
        up: 'up',
        down: 'down',
    },
    focusUp: function() {
        this.up = 'up focused';
    },
    blurUp: function() {
        this.up = 'up';
    },
    keyUp: function() {
        this.up = 'up keyed';
    },
    focusDown: function() {
        this.down = 'down focused';
    },
    blurDown: function() {
        this.down = 'down';
    },
    keyDown: function() {
        this.down = 'down keyed';
    },
}
```

按键事件通过获焦事件向下分发,因此上述示例中使用了 focus 事件和 blur 事件来明确当前焦点位置。当按上下键时,相应的 focused 状态将会响应。当失去焦点按键时,恢复到正常的 up 或 down 按键文本。按确认键后该键变为 keyed 状态。

5.4.3 页面路由

很多情况下,在开发 App 时不只使用一个页面,例如在购物 App 中有时需要从商品详情页面跳转到购物车页面,这时就需要在商品详情页面设置一个能跳转到购物车页面的入口,在浏览完购物车页面之后又需要回到之前的商品详情页面,此时就需要页面路由功能。

在页面路由中需要定义两个或两个以上页面,然后在各自页面的 js 文件中定义相应的路由函数来使用目标页面的 uri 跳转到目标页面。定义两个页面,两个页面的 hml 文件代码分别如下:

```
<!-- first.hml -->
<div class="container">
<div class="text-div">
<text class="title">
//这是第一个页面
```

```
</text>
</div>
<div class="button-div">
<button type="capsule" value="跳转到第二页面" class="button" onclick="launch"></button>
</div>
</div>

<!-- second.hml -->
<div class="container">
<div class="text-div">
<text class="title">
//这是第二个页面
</text>
</div>
<div class="button-div">
<button type="capsule" value="回到第一个页面 by router.back" class="button" onclick="launch"></button>
<button type="capsule" value="回到第一个页面 by router.push" class="button" onclick="launch2"></button>
</div>
</div>
```

在第一个页面中定义了一个 button 并将其与 first.js 中的 launch 函数相关联，第二个页面中的两个 button 分别与 second.js 中的 launch 函数与 launch2 函数相关联，两个 js 文件中的代码如下：

```
//first.js
import router from '@system.router';
//指定的页面.在调用 router 方法之前,需要导入 router 模块
export default {
    launch: function () {
        router.push({
            uri: 'pages/second/second',                    //目标页面的路径
        })
    },
}

//second.js
import router from '@system.router'
export default {
    launch: function () {
        router.back();                                     //回到路由前的页面
    },
    launch2: function () {
```

```
            router.push({
                uri: 'pages/first/first',                    //目标页面的路径
            })
        }
    }
```

在第一个页面的 first.js 文件中，使用 router.push 来跳转到第二个页面，uri 代表目标页面的路径，然而回到之前的页面有两条途径，也就是在第二个页面中所使用的两种方法，router.back 和 router.push。其中，router.back 可以实现直接跳回原页面，所以不需要任何 uri，而 router.push 利用原页面的 uri 再次跳转到原页面。

下面是两个页面的 css 文件，代码如下：

```
/* first.css */
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
.text-div{
    justify-content: center;
}
.button-div{
    justify-content: center;
}

/* second.css */
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
.text-div{
    justify-content: center;
}
.button-div{
    justify-content: center;
}
```

为了实现页面跳转，需要将每个页面的 3 个文件(hml、css、js)都放入各自独立的文件夹，如图 5.21 所示。

之后还需要在 App 模块的 config.json 文件中注册已编写好的页面，代码如下：

```
"js": [
  {
    "pages": [
```

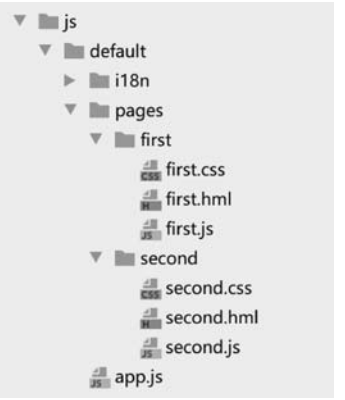

图 5.21 页面路由示例工程结构图

```
        "pages/first/first",
        "pages/second/second"
      ],
      ...
    }
  ]
```

pages 中排在第一位的页面将作为应用程序的默认页面，也就是打开应用程序后显示的第一个页面。运行上述代码，首先显示的第一个页面如图 5.22 所示。

单击“跳转到第二个页面”按钮将会跳转到第二个页面，如图 5.23 所示。

图 5.22 页面路由示例的第一个页面

图 5.23 页面路由示例的第二个页面

单击页面上的两个按钮都会跳转到第一个页面，不同的地方是左边的按钮使用返回方式回到原页面，因此更像返回操作，而右边的按钮使用 uri 跳转到原页面，因此更像前进操作。

5.5 动画

5.5.1 transform 静态动画

静态动画的核心是 transform 样式，其中包含 3 种变换类型，且每一次的样式设置只能支持一种类型的变化。下面对 3 种类型进行逐一讲解。

首先，translate 变换类型可以将组件沿水平或垂直方向移动一定的距离，创建 JS 项目工程，修改 index.hml 和 index.css 文件，示例是一个水平向右移动的示例代码，代码如下：

```
<!-- index.hml 实现水平向右移动 -->
<div class = "container">
<text class = "translate"> hello </text >
</div >

/* index.css */
.container {
    flex - direction: column;
    align - items: center;
}
.translate {
    height: 300px;
    width: 400px;
    font - size: 100px;
    background - color: #008000;
    transform: translateX(300px);
}
```

上述代码实现了一个 text 组件水平向右移动的效果，translateX(300px)将 text 组件水平从基准线向右移动 300px，其中右为正值，左为负值。同理，若修改为 translateY()，则是以 Y 轴基准线为标准，下为正值，上为负值。运行结果如图 5.24 所示，灰线为基准线。

scale 样式可将组件沿横向或纵向，缩小或放大一定比例。下面对 text 组件进行横向放大，代码如下：

```
<!-- index.hml 实现横向放大 -->
<div class = "container">
<text class = "scale"> hello </text >
</div >
```

```
/* index.css */
.container {
    flex-direction: column;
    align-items: center;
}
.scale {
    height: 300px;
    width: 400px;
    font-size: 100px;
    background-color: #008000;
    transform: scaleX(1.5);
}
```

其中,scaleX(1.5)表示将 text 文本横向放大 1.5 倍,运行上述代码可以直观看到效果,如图 5.25 所示。

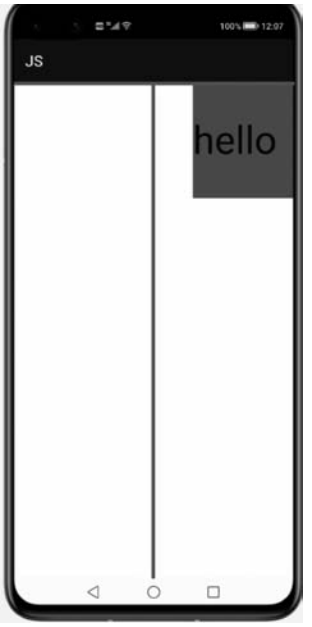

图 5.24 静态动画平移示例效果图

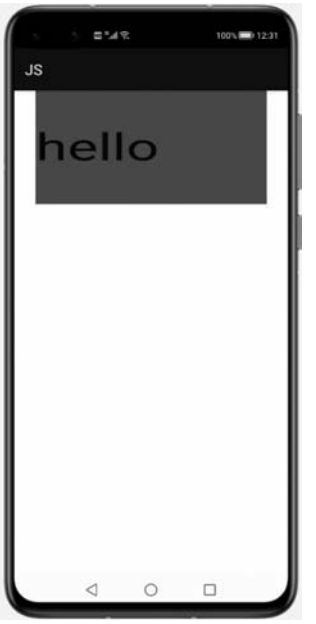

图 5.25 静态动画放大示例效果图

rotate 样式可以将组件沿横轴或纵轴或中心点,旋转一定的角度,如下示例将 text 文本绕 X 轴顺时针旋转 45°,代码如下:

```
<!-- index.hml 实现绕 X 轴顺时针旋转 -->
<div class="container">
<text class="rotate">hello</text>
</div>
```

```
/* index.css */
.container {
    flex-direction: column;
    align-items: center;
}
.rotate {
    height: 300px;
    width: 400px;
    font-size: 100px;
    background-color: #008000;
    transform-origin: 200px 100px;
    transform: rotateX(45deg);
}
```

运行上述代码，结果如图 5.26 所示。

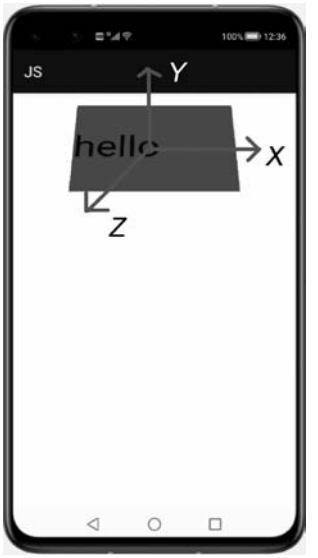

图 5.26 静态动画旋转示例效果图

图 5.26 中 Z 轴垂直穿出屏幕。一般的 rotateX()和 rotateY()表示绕 X 和 Y 轴顺时针旋转，而 rotate()表示绕 Z 轴旋转。

与连续动画不同，静态动画只有开始状态和结束状态，而不能设置中间状态，如果需要设置中间的过渡状态和转换效果，则只能由连续动画实现。

5.5.2 animation 连续动画

前面讲过的静态动画只有开始状态和结束状态，而没有中间状态，因此静态动画看起来更像是图片之间的切换而不是真正意义上的动画。为了使动画变得连贯，HarmonyOS 中的 JS 引入了连续动画。连续动画最主要的功能贡献者就是 animation 样式，通过它可以定义动画的开始、结束状态及期间变化的速度。在引入连续动画之后，可以定义组件的宽、高、颜色和透明度等的变化速度和程度，利用这一点可以实现一些有趣的功能，例如进度条、渐变色块等。在本节中利用 animation 实现颜色、透明度和宽度变化的功能。首先新建 JS 工程，修改代码如下：

```
<!-- animation.hml 创建页面 -->
<div class="item-container">
<div class="group">
<text class="header">
<!-- 动画演示 -->
</text>
<div class="item {{colorParam}}">
```

```
<text class="txt">
<!-- 颜色 -->
</text>
</div>
<div class="item {{opacityParam}}">
<text class="txt">
<!-- 透明度 -->
</text>
</div>
<input class="button" type="button" name="" value="开始" onclick="show01"/>
</div>
</div>
```

在页面中定义了两个文本组件，分别用来演示颜色变化，以及透明度和宽度变化。input 组件的单击事件 show()在文件中的实现代码如下：

```
//animation.js
export default {
    data: {
        colorParam: '',
        opacityParam: '',
    },
    show: function () {
        this.colorParam = ''
        this.opacityParam = ''
        this.colorParam = 'color'
        this.opacityParam = 'opacity'
    },
}
```

其中，colorParam 和 opacityParam 分别与 hml 文件中的样式进行数据绑定，在 show()方法中，首先将这两个参数设置为默认格式，并在 css 文件中进行动画实现，css 文件代码如下：

```
/* animation.css */
.item-container {
    margin-bottom: 50px;
    margin-right: 60px;
    margin-left: 60px;
    flex-direction: column;
    align-items: flex-start;
}
.group {
    margin-bottom: 150px;
    flex-direction: column;
```

```
    align-items: flex-start;
}
.header {
    margin-bottom: 20px;
}
.item {
    background-color: #191FF7;
}
.txt {
    text-align: center;
    width: 200px;
    height: 100px;
}
.button {
    width: 200px;
    font-size: 30px;
    color: #ffffff;
    background-color: #09ba07;
}
.color {
    animation-name: Color-frames;              /* 动画由 Color-frames 定义 */
    animation-duration: 8000ms;
}
.opacity {
    animation-name: Opacity-frames;            /* 动画由 Opacity-frames */
    animation-duration: 8000ms;
}
@keyframes Color-frames {                      /* 颜色变换效果动画 */
    from {
        background-color: #191FF7;             /* 初始颜色 */
    }
    to {
        background-color: #09ba07;             /* 最终颜色 */
    }
} @keyframes Opacity-frames {                  /* 透明度、宽度变换效果动画 */
    from {
        width: 600px;                          /* 初始宽度 */
        opacity: 0.9;                          /* 初始透明度 */
    }
    to {
        width: 0px;                            /* 最终宽度 */
        opacity: 0.0;                          /* 最终透明度 */
    }
}
```

在.color 和.opacity 样式中，使用 animation-name 属性定义了各自的动画样式

(@keyframes)。其中@keyframes 可以自定义动画样式,例如代码中定义的 Color-frames 和 Opacity-frames,from 代表动画的开始状态,to 代表结束状态,期间的过渡动画由系统自动计算完成,当然也可以使用 animation-timing-function 属性来描述动画执行的速度曲线,使动画更加平滑。运行上述代码,初始页面如图 5.27 所示。

单击“开始”按钮后,会播放定义的动画(按时间顺序排列),如图 5.28 所示。

图 5.27 连续动画示例的初始状态

(a) 动画过程 (b) 动画结束状态

图 5.28 连续动画示例的过程(a)和结束状态(b)

可以看到“颜色”块的背景色从蓝色最终变为绿色,“透明度”块的透明度从不透明变为全透明,并且宽度从最开始一直变为 0,类似一个反向的进度条。在这里还可以使用 animation 的其他属性来定义更多有趣的动画效果,读者可以自行尝试。