第 5 章

JavaScript UI 设计

相对于 Java UI 来讲，JavaScript UI 非常类似于 Web 前端开发，毕竟使用的就是 JavaScript 前端技术的脚本语言。因此，如果你是一个成熟的前端开发者，或者是微信小程序开发者，那么 JavaScript UI 就是专门为你准备的技术框架了。不过，即使如此，你仍然需要学习许多 JavaScript UI 所特有的知识，包括 HML 语法、组件的基本用法等。

与前端技术一样，JavaScript UI 包含结构（HML）、表现（CSS）和逻辑（JavaScript）共 3 个主要部分：

（1）HML：鸿蒙标记语言（HarmonyOS Markup Language），用于表述用户界面的结构。通过 HML 编写的界面结构文件后缀名为.hml，因此 HML 既是一门语言，也是一种文件类型。要特别注意，虽然 HML 与 HTML 语法相似，但是仍然存在很多区别。

（2）CSS：层叠样式表（Cascading Style Sheets），用于表述用户界面的表现样式。

（3）JavaScript：一种解释性脚本语言，用于表述用户界面的简单业务逻辑，支持 ECMAScript 6 语法。

在学习本章内容之前，需要读者先学习并掌握 Web 前端的开发知识，特别是 JavaScript 和 CSS 的使用方法。对于 HTML，建议读者了解 HTML 中的一些基本标签的使用，因为 HML 与 HTML 的使用方法实在太相似了。

JavaScript UI 支持手机、平板计算机、智慧屏、智能穿戴设备、轻量级智能穿戴等设备上进行应用开发，似乎比 Java UI 的适用能力更加广泛，但是由于 JavaScript UI 更加倾向于界面开发，在许多涉及设备硬件等复杂业务逻辑方面，可能还需要 Java 来帮忙处理，这就涉及 JavaScript UI 和 Java UI 之间的交互方法了。值得注意的是，在轻量级智能穿戴设备上的 JavaScript UI 会更加轻量化，并不能使用所有的 JavaScript UI 特性。

5.1 初识 JavaScript UI

JavaScript UI 采用声明式编程范式，可以帮助开发者摆脱烦琐的 UI 状态切换，即当开发者修改数据时 UI 内容可以实现自动更新。这就省去了许多 Java UI 中的组件内容设置代码和监听器代码，对于开发而言方便了许多。

本节介绍 JavaScript UI 的基础知识，包括工程组织、基本语法、跳转路由等。

14min

5.1.1 JavaScript 实例与页面

让我们先熟悉一下“新面孔”。本节介绍新的 JavaScript 鸿蒙应用程序工程的基本结构，随后介绍 JavaScript 实例和页面的基本概念，以及 JavaScript 实例的配置选项。

1. 第一个 JavaScript 鸿蒙应用程序工程

打开 DevEco Studio，并创建一个新的鸿蒙应用程序，在 Create HarmonyOS Project 对话框中，在 Device 选项中选择 Phone，在 Template 选项中选择 Empty Feature Ability(JS)，并单击 Next 按钮，如图 5-1 所示。

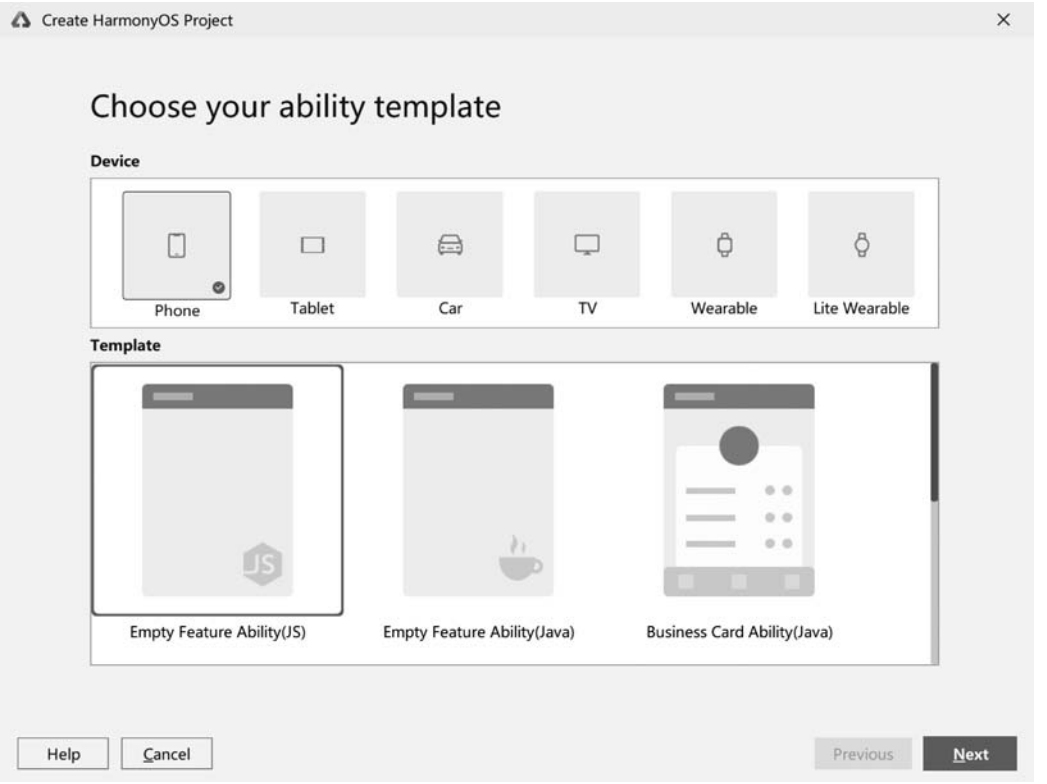


图 5-1 创建 Empty Feature Ability(JS)模板工程

在随后弹出的 Create HarmonyOS Project 对话框中，将工程名称(Project Name)设置为 HelloJavaScript，将包名(Package Name)设置为 com. example. helloJavaScript，其他选项保持默认，单击 Finish 按钮完成工程创建。

创建完成后，可以发现该工程和 Java 模板的鸿蒙应用程序工程并没有很大不同，唯一的区别就是在 main 目录下多出了 js 目录。js 目录是 JavaScript 鸿蒙应用程序开发的主战场，其默认创建的文件如图 5-2 所示。

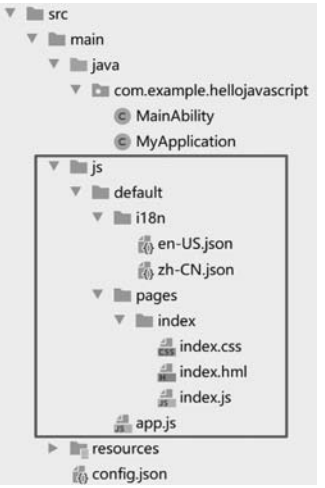


图 5-2 Empty Feature Ability(JS)模板工程结构

2. JavaScript 实例和页面的概念

js 目录下有一个 default 目录，这个目录属于一个 JavaScript 实例(以下简称实例)。在鸿蒙应用程序中，一个实例包含了一组与功能相关的用户界面和资源，默认的实例名称为 default。通常，每个实例都被独立的 Ability 管理，而 default 实例默认被 MainAbility 管理。

在实例中，可以创建多个用户界面，其中每个用户界面被称为一个页面(Page)。Ability、实例和页面的关系如图 5-3 所示。

如果开发者仅使用 JavaScript 语言开发鸿蒙应用程序，则只需使用默认的 default 实例，一般不需要创建多个 Ability，也不需要创建多个实例。此时，JavaScript 鸿蒙应用程序可以简化为如图 5-4 所示的结构。

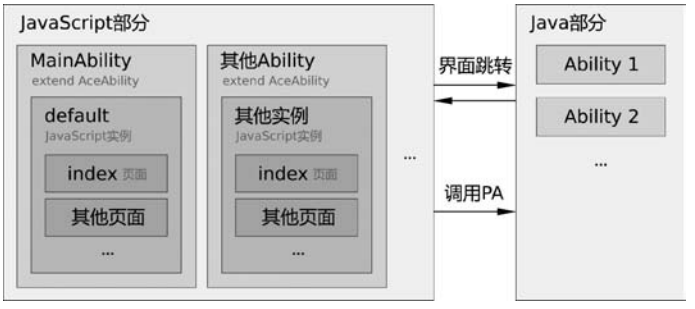


图 5-3 Ability、JavaScript 实例和页面的关系

图 5-4 JavaScript 鸿蒙应用程序简化结构

这样，我们只需要在 default 实例中创建所需要的页面，并实现页面的路由跳转。

注意：JavaScript UI 中的页面概念非常类似于 Java UI 中的 AbilitySlice 概念，都是承

载用户界面的最小单元。

3. 配置 JavaScript 实例

JavaScript 实例需要在 config.json 文件中 module 对象下的 js 对象内进行配置。js 对象为一个数组，其中每个对象都代表了一个实例。默认情况下，工程创建了一个 default 默认实例，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/config.json
"js": [
  {
    "pages": [
      "pages/index/index"
    ],
    "name": "default",
    "window": {
      "designWidth": 720,
      "autoDesignWidth": false
    }
  }
]
```

实例中包含以下常用属性：

- name：声明实例名称，默认为 default。
- pages：声明实例所包含的页面。
- window：声明与虚拟像素相关的选项。

pages 属性为一个由页面路径所组成的数组。页面路径是以实例目录为根目录的路径，且不带.hml 后缀名。数组中的第一个页面为该实例的主路由，即首先被打开的页面。

window 属性包含 designWidth 和 autoDesignWidth 两个子属性：

(1) autoDesignWidth 表示是否启用虚拟像素。与 Java UI 不同，在 JavaScript UI 中只有 px 这一种像素单位。当 autoDesignWidth 为 true 时，px 像素单位就不代表物理像素了，而是以 160ppi 为基准的虚拟像素(如同 Java UI 中的 vp)，读者可参见 3.1.5 节的相关内容。

(2) designWidth 表示屏幕的虚拟宽度(当 autoDesignWidth 为 false 时有效)。当指定了虚拟宽度后，px 像素单位也同样不代表实际的物理像素，而是会以虚拟宽度为基准缩放像素大小。例如，当 designWidth 为 720(默认的手机虚拟像素)时，10px 在实际像素宽度为 1440 的设备上会表现为 20px。

注意：如果 autoDesignWidth 为 false，且没有通过 designWidth 指定屏幕的虚拟宽度，则在开发中 px 像素单位表现为实际的物理宽度。

4. 实例的目录结构

在图 5-2 中已经展示了实例的基本目录结构，但是一个完整的 js 实例还可能包含

common 目录和 resources 目录。一个实例所包含的目录和文件的功能如下：

(1) pages 目录：以子目录的形式组织页面文件，每个子目录都代表一个页面，并且包含了 hml、js 和 css 文件。

(2) common 目录：存放通用的代码文件(js 文件、自定义组件等)，以及各种资源文件(图片、声频等)。

(3) i18n 目录：国际化目录，用于存放字符串资源等。

(4) resources 目录：资源配置目录，用于存放用于适配不同屏幕密度的资源配置文件等。

这些目录在学习和实践中会经常遇到，在这里不进行详细介绍。

5.1.2 新的 JavaScript 实例

3min

创建实例有两个主要方法：

(1) 创建新的使用 JavaScript UI 的 Page Ability，并同时创建实例。

(2) 仅创建一个新的实例。

这里建议使用第一种方法来创建实例，因为采用第二种方法创建新的实例后，还需要指定 Ability 来管理这个实例。

1. 创建新的使用 JavaScript UI 的 Page Ability

在 Project 工具窗体中，在 entry 目录上右击，选择 New→Ability→Empty Page Ability (JS)菜单，弹出如图 5-5 所示的对话框。

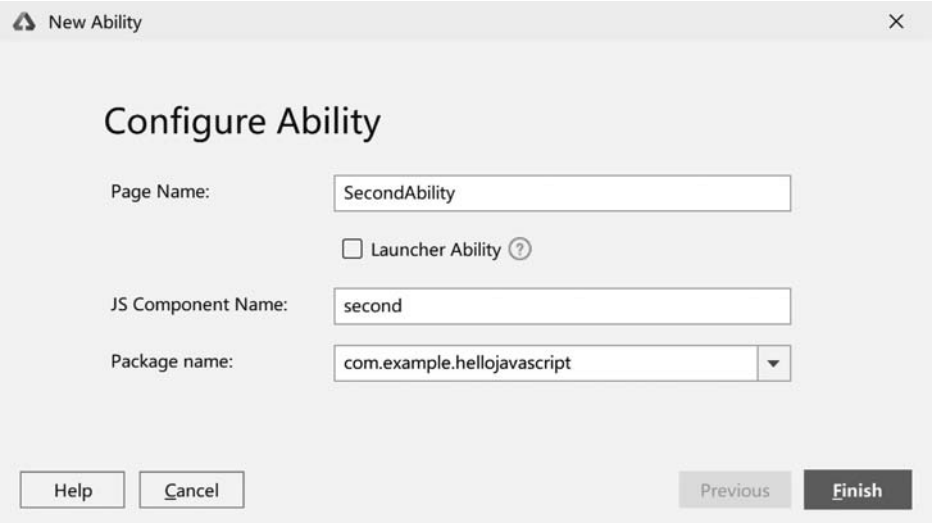


图 5-5 创建使用 JavaScript UI 的 Page Ability

在 Page Name 选项中输入 Ability 的名称 SecondAbility；在 JS Component Name 选项中输入默认的 JavaScript 实例名称 second；在 Package name 选项中选择包名(保持默认

即可),单击 Finish 按钮。

此时,在工程中会出现新创建的 SecondAbility 和 second 实例,如图 5-6 所示。

SecondAbility 和 second 实例会自动注册在 config.json 中。

2. 仅创建 JavaScript 实例

此处创建名为 foo 的实例。在 js 目录上右击,在菜单中选择 New→JS Component 即可弹出创建实例对话框 New JS Component,如图 5-7 所示。

在 JS Component Name 选项中输入实例名称 foo,单击 Finish 按钮。此时,该工程中即可出现刚刚所创建的 foo 实例目录,并自动生成了 index 页面,并且,foo 实例也会自动注册在 config.json 中。

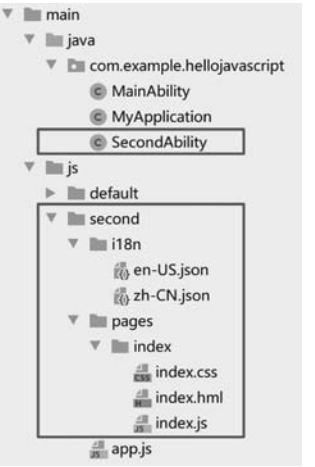


图 5-6 SecondAbility 和 second 实例

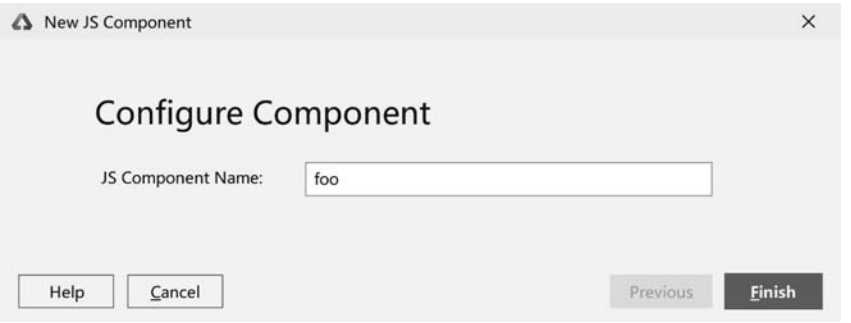


图 5-7 New JS Component 对话框

3. 指定 Ability 的默认实例

分析一下 MainAbility.java 这个文件,代码如下:

```
//chapter5/HelloJavaScript/entry/src/main/java/com/example/helloJavaScript/MainAbility.java
public class MainAbility extends AceAbility {
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
    }

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

这段代码和之前在 Java 环境下学习的 MainAbility 没有什么不同，只是其父类从 Ability 变成了 AceAbility。这里的 Ace 表示 Ability 跨平台环境（Ability Cross-Platform Environment），而 JavaScript UI 即运行在 ACE 基础之上。

默认情况下，它会加载名为 default 的实例，因此，通常 MainAbility 的代码不需要进行改动，只需把这个类放在这里就可以了。

但是，如果希望让这个 MainAbility 加载非 default 实例，则只需要在 onStart 方法的最前面（注意要在 super.onStart(intent)之前）调用 setInstanceName 方法，并传入实例名称字符串。例如，默认加载 foo 实例的代码如下：

```
@Override
public void onStart(Intent intent) {
    super.setInstanceName("foo");
    super.onStart(intent);
}
```

5.1.3 初识页面

4min

一个 JavaScript 页面由同一目录下的 HML 文件、CSS 文件和 JavaScript 文件组成，例如工程默认生成的 index 页面包含了 index.hml、index.css 和 index.js 文件。

index.hml 文件声明了页面的结构，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/js/default/pages/index/index.hml
<div class = "container">
    <text class = "title">
        {{ $t('strings.hello') }} {{title}}
    </text>
</div>
```

标签下包含了<text>标签，这些标签都是鸿蒙应用程序的组件，前者为块组件（容器），后者为文本组件。在文本组件中，通过双引用括号“{{ }}”来嵌入 JavaScript 代码。这种直接将变量填入“{{ }}”中被称为变量的动态绑定，当被绑定变量发生变化时，页面也会随之更新这个变量的显示效果。

在上面的代码中，“{{ $t('strings.hello') }}”引用了字符串资源中的 hello 字符串；“{{title}}”引用了 title 变量。title 变量在 index.js 中进行定义，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/js/default/pages/index/index.js
export default {
    data: {
        title: ""
    },
```

```
    onInit() {
        this.title = this.$t('strings.world');
    }
}
```

在上面的代码中，data 对象定义了 title 变量。然后，在页面生命周期方法 onInit()中将 title 变量设置为字符串资源中的 world 字符串。关于页面的生命周期方法将在 5.1.5 节进行详细介绍，这里只需知道页面在创建时会被调用一次（且仅被调用一次）onInit()方法用于初始化整个页面。

字符串资源由 i18n 目录下的 json 文件定义。json 文件的文件名由语言和地区两个部分组成，默认情况下工程自动生成了 zh-CN.json 和 en-US.json 两个字符串资源文件。

注意：i18n 为 internationalization 的缩写。由于这个单词实在太长了，因此中间的 18 个英文字符被简称为 18，然后加入首末两个字母 i 和 n，构建了这个非常经典的简写。

zh-CN.json 包含了语言为中文（zh）、地区为中国（CN）的字符串资源，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/js/default/i18n/zh-CN.json
{
  "strings": {
    "hello": "您好",
    "world": "世界"
  }
}
```

en-US.json 包含了语言为英文（en）、地区为美国（US）的字符串资源，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/js/default/i18n/en-US.json
{
  "strings": {
    "hello": "Hello",
    "world": "World"
  }
}
```

index.css 文件定义了页面的样式，代码如下：

```
//chapter5/HelloJavaScript/entry/src/main/js/default/pages/index/index.css
.container {
    flex-direction: column;        /* 垂直排列组件 */
    justify-content: center;       /* 垂直方向上居中显示组件 */
    align-items: center;           /* 组件水平居中 */
}

.title {
    font-size: 100px;              /* 文字大小 */
}
```

该页面的显示效果如图 5-8 所示。

图 5-8 index 页面的默认显示效果

注意：JavaScript UI 中的代码调试方法与 Java 代码的调试方法非常类似，读者可以参考 2.3.1 节中的相关内容。

5.1.4 页面的跳转

本节创建一个新的工程 JSRoute，并实现页面的跳转，即在 default 实例中创建一个新的页面 secpage，并实现 index 页面和 other 页面之间的跳转。

在学习本节内容之前，读者需要先创建一个目标为手机的 Empty Feature Ability(JS) 模板工程 JSRoute，具体方法可参考 5.1.1 节的相关内容。

1. 创建新的页面

在 default 实例中的 pages 目录上右击，选择 New→JS Page 选项，弹出创建页面对话框 New JS Page，如图 5-9 所示。

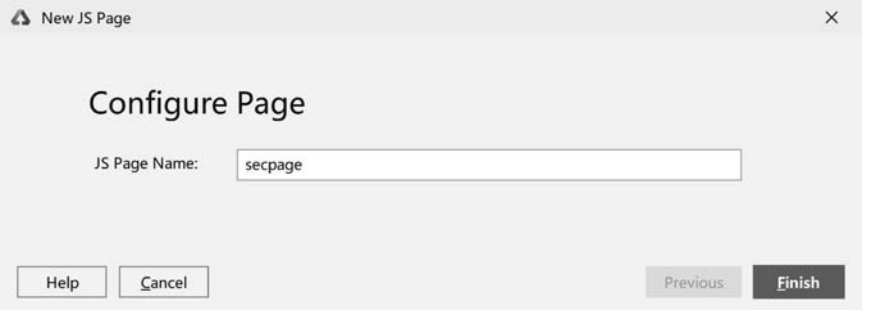


图 5-9 创建页面对话框

在 JS Page Name 选项中输入页面名称 secpage，单击 Finish 按钮即可创建名为 secpage 的页面，如图 5-10 所示。

图 5-10　新创建的 secpage 页面

2. 页面的跳转

在 JavaScript UI 中，支持页面的层级显示，即上层页面遮盖下层页面，并形成一个栈结构，称为页面栈。页面的跳转关系被称为页面路由，由 JavaScript UI 的 router 模块来管理，该模块需要在 js 文件的 export default 的代码块前进行导入，代码如下：

```
import router from '@system.router';
export default {
    ...
}
```

router 模块主要包含以下方法：

(1) push(obj: IForwardPara)：跳转到另外一个页面，而且原先的页面仍然存在，只是被遮盖而已。通过 IForwardPara 可以定义跳转的页面和传递的数据，分别通过其 URL 属性和 params 属性定义。

(2) replace(obj: IForwardPara)：跳转到另外一个页面，并销毁当前页面。通过 IForwardPara 可以定义跳转的页面和传递的数据，分别通过其 URL 属性和 params 属性定义。

(3) back(obj?: IBackPara)：返回上一个页面。通过 IBackPara 可以定义返回的页面路径(可选)，通过该对象内的 path 属性定义。

(4) clear()：清除被遮盖的页面，仅保留当前显示的页面。

(5) getLength()：获取当前页面栈长度，即栈内页面数量。

(6) getState()：获取当前页面栈状态，返回 IRouterState 对象，该对象包括 index、name 和 path 共 3 个变量。index 变量为整型，表示当前页面所在页面栈的位置，从底层到顶层是从 1 开始计数的。name 为字符串，表示当前页面文件名。path 为字符串，表示当前页面的路径。

注意：并不是在页面 secpage 中 router.getState().name 获取的字符串就一定是 secpage。例如，在本节所介绍的例子中，当从 index 页面跳转到 secpage 页面时，在 secpage 页面的 onInit()和 onReady()生命周期方法中 router.getState().name 获取的字符串为 index，这是因为此时 secpage 还并没有加入页面栈中，也没有显示在屏幕上。关于页面的生命周期方法参考 5.1.5 节的相关内容。

上述关于页面跳转的方法的实现效果如图 5-11 所示。

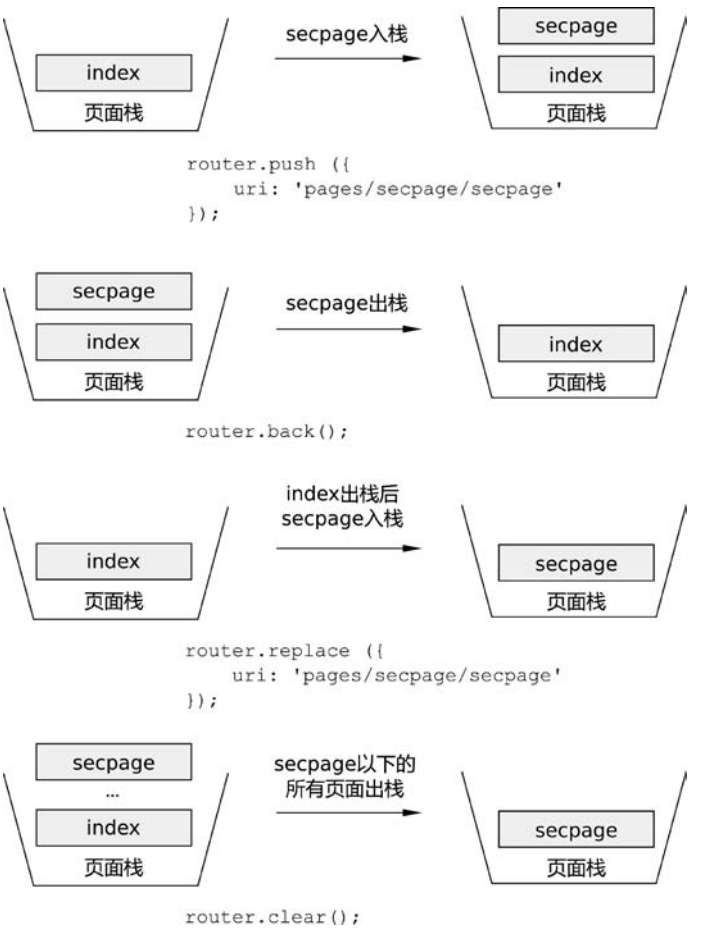


图 5-11 4 种关于页面跳转的方法

1）实现 index 页面跳转到 secpage 页面

首先，在 index.js 中实现跳转代码，代码如下：

```
//chapter5/JSRoute/entry/src/main/js/default/pages/index/index.js
//导入 router 模块
import router from '@system.router';
export default {
    data: {
        harmony: null            //定义 harmony 字符串
    },
    onInit() {
        //初始化 harmony 字符串
        this.harmony = "鸿蒙初生,连接万物";
    },
    //跳转到 secpage 的方法
    toSecPage() {
```

```
            //通过 push 方法入栈
            router.push ({
                //指定跳转位置
                uri: 'pages/secpage/secpage',
                //传递数据
                params: {
                    harmony: this.harmony
                }
            });
        }
    }
```

在上面的代码中,创建了跳转到 secpage 页面的方法 toSecPage()。在 toSecPage 方法中,调用了 router 模块的 push 方法,并指定了跳转路径(pages/secpage/secpage)和所需要传递的数据(harmony 字符串)。该字符串指定了本页面中的 harmony 字符串变量(this.harmony)。this.harmony 变量在 data 中进行了定义,并在 onInit()生命周期方法中初始化为"鸿蒙初生,连接万物"。关于页面的生命周期方法将在 5.1.5 节进行详细介绍,这里只需知道页面在创建时会被调用一次(且仅被调用一次)onInit()方法用于初始化整个页面。

接下来,在 index 页面中加入一个按钮,并在单击该按钮时触发 toSecPage()方法,代码如下:

```
//chapter5/JSRoute/entry/src/main/js/default/pages/index/index.hml
<div class = "container">
    <text class = "title">
        {{ this.harmony }}
    </text>
    <button onclick = "toSecPage">进入 SecPage 页面</button>
</div>
```

在上述代码中,<text>文本组件动态绑定了 harmony 变量的内容;定义了<button>按钮组件的 onclick 事件(即单击事件)的处理函数为 toSecPage 方法。

此时,编译并运行程序,单击【进入 SecPage 页面】即可跳转到 secpage 页面。

2) 使用被传递的数据,并实现从 secpage 页面返回 index 页面

接下来,在 secpage 页面的 js 文件中打印刚才传递来的 harmony 变量,并创建返回 index 页面的方法 back(),代码如下:

```
//chapter5/JSRoute/entry/src/main/js/default/pages/secpage/secpage.js
import router from '@system.router';
export default {
    data : {
    },
    onInit() {
```

```
        //输出刚被传递来的 harmony 字符串
        console.info(this.harmony);
    },
//返回之前的页面
    back() {
        router.back();
    }
}
```

可见，被传递来的数据可以直接使用，通过 this.harmony 即可引用被传递来的 harmony 变量。通过 console.info(message)方法可以将该字符串变量以 HiLog 的形式输出。

由于目前 secpage 页面处在 index 页面的顶端，因此在 back()方法中调用了 router 模块的 back()方法将当前页面出栈，index 页面将重见光日。

然后，在 secpage.hml 中添加一个按钮，单击触发 back()方法，代码如下：

```
//chapter5/JSRoute/entry/src/main/js/default/pages/secpage/secpage.hml
<div class = "container">
    <button onclick = "back">返回主页面</button>
</div>
```

编译并运行程序，在 index 页面中单击【进入 SecPage 页面】按钮后，就会进入 secpage 页面。此时，单击【返回主页面】即可返回 index 页面，如图 5-12 所示。

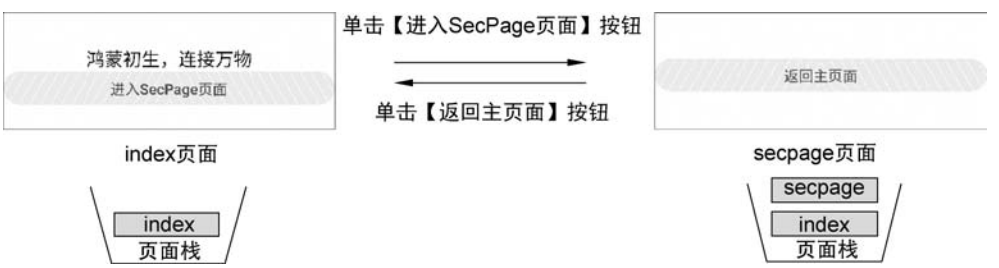


图 5-12 index 页面和 secpage 页面的跳转效果

另外，进入 secpage 页面时，DevEco Studio 的 HiLog 工具窗体会显示从 index 页面传递来的数据，如图 5-13 所示。

3. JavaScript UI 的 HiLog 输出

在上面的代码中，通过 console.info(message)方法就可以以 HiLog 的形式输出字符串信息。事实上，根据输出级别的不同，console 主要包括 log(日志)、info(一般信息)、debug(调试)、warn(警告)、error(错误)。这些级别可以参考 2.3.2 节的相关内容。

在开发者通过上述方法输出的信息中，都会在 HiLog 输出中加入 app Log 字符串，因此可以通过该字符串加以筛选，以显示所需要调试和观察的输出信息，如图 5-13 所示。

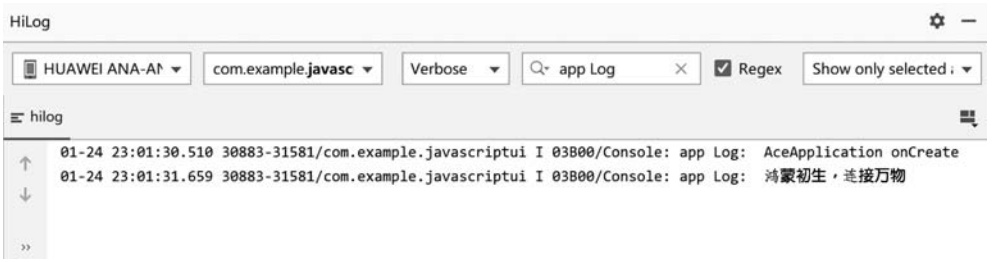


图 5-13　从 index 页面传递的 harmony 字符串


3min

5.1.5　页面的生命周期

与 Java UI 中的 Page Ability 一样，页面也存在从初始化到销毁的生命周期，这些生命周期方法如下所示：

- onInit()：页面初始化时调用。
- onReady()：页面创建完成时调用。
- onShow()：页面显示时调用。
- onHide()：页面隐藏时调用。
- onDestroy()：页面销毁时调用。
- onBackPress()：当用户按下系统的后退按钮后调用。

onBackPress()生命周期方法有些特殊，先介绍前 5 个生命周期方法，然后单独介绍 onBackPress()方法。

1. 生命周期方法的调用时机

onInit()、onReady()、onShow()、onHide()、onDestory()这 5 个生命周期方法调用时机如图 5-14 所示。

图 5-14　页面的生命周期方法调用时机

可见,在一个完整的页面生命周期中,这些生命周期方法都会至少被调用一次,并且onInit()、onReady()和onDestroy()仅能够被调用一次。

通过下面的实例来感受生命周期方法的调用。在该实例中,创建了名为JSLifecycle的JavaScript鸿蒙应用程序工程,并使用5.1.4节的方法创建了index和secpage两个页面,从而实现了页面的跳转(采用push和back方法进行跳转和跳回)。

在index.js中,实现了这5个生命周期方法,并在每个生命周期被调用时输出HiLog字符串,代码如下:

```
//chapter5/JSLifecycle/entry/src/main/js/default/pages/index/index.js
export default {
    …
    onInit() {
        console.info("Page index on init!");
    },
    onReady(){
        console.info("Page index on ready!");
    },
    onShow() {
        console.info("Page index on show!");
    },
    onHide() {
        console.info("Page index on hide!");
    },
    onDestroy() {
        console.info("Page index on destroy!");
    },
    …
}
```

同样地,在secpage页面中实现这5个生命周期方法,代码如下:

```
//chapter5/JSLifecycle/entry/src/main/js/default/pages/secpage/secpage.js
export default {
    …
    onInit() {
        console.info("Page secpage on init!");
    },
    onReady(){
        console.info("Page secpage on ready!");
    },
    onShow() {
        console.info("Page secpage on show!");
    },
    onHide() {
```

```
            console.info("Page secpage on hide!");
        },
        onDestroy() {
            console.info("Page secpage on destroy!");
        },
        …
    }
```

编译并运行该应用程序，观察在不同类型的操作中 HiLog 的提示：

(1) 启动应用程序时，在 HiLog 中会出现如下类似的提示：

```
16123 - 16590/? I 03B00/Console: app Log: Page index on init!
16123 - 16590/? I 03B00/Console: app Log: Page index on ready!
16123 - 16590/? I 03B00/Console: app Log: Page index on show!
```

这说明 index 页面经过了创建和显示步骤，调用了 onInit()、onReady()和 onShow()这 3 个生命周期方法。

(2) 单击系统的 Home 键，使应用进入后台，此时 HiLog 提示如下：

```
16123 - 16590/? I 03B00/Console: app Log: Page index on hide!
```

这说明该页面不再显示在屏幕上，调用了 onHide()方法。

(3) 重新进入该应用，使该应用进入前台，此时 HiLog 提示如下：

```
16123 - 16882/? I 03B00/Console: app Log: Page index on show!
```

这说明 index 重新回到屏幕并显示出来。

(4) 关闭应用程序，此时 HiLog 提示如下：

```
16123 - 16969/? I 03B00/Console: app Log: Page index on hide!
16123 - 16969/? I 03B00/Console: app Log: Page index on destroy!
```

这说明 index 页面被关闭时会依次调用 onHide()和 onDestory()方法。

(5) 接下来，试一下从 index 页面跳转到 secpage 页面(采用 router 模块的 push 方法跳转)，此时 HiLog 提示如下：

```
16123 - 17172/? I 03B00/Console: app Log: Page secpage on init!
16123 - 17172/? I 03B00/Console: app Log: Page secpage on ready!
16123 - 17172/? I 03B00/Console: app Log: Page index on hide!
16123 - 17172/? I 03B00/Console: app Log: Page secpage on show!
```

在跳转过程中，首先初始化并创建 secpage 页面，然后将 index 页面隐藏，最后将 secpage 页面显示出来。

(6) 从 secpage 页面回到 index 页面(采用 router 模块的 back 方法返回),此时 HiLog 提示如下:

```
16123 - 17172/? I 03B00/Console: app Log: Page secpage on hide!
16123 - 17172/? I 03B00/Console: app Log: Page secpage on destroy!
16123 - 17172/? I 03B00/Console: app Log: Page index on show!
```

在回调过程中,先销毁了 secpage 页面,然后显示 index 页面。

总结一下,常见的页面生命周期状态变化过程,如表 5-1 所示。

表 5-1 常见的页面生命周期状态变化过程

常 见 操 作	回调生命周期方法
进入页面	onInit()→onReady()→onShow()
进入后台	onHide()
重新进入前台	onShow()
退出页面	onHide()→onDestory()
从页面 A 跳转到页面 B(用 router 模块的 push 方法)	B. onInit()→B. onReady()→A. onHide()→B. onShow()
从页面 B 回跳到页面 A(采用 router 模块的 back 方法)	B. onHide()→B. onDestory()→A. onShow()

2. 通过 onBackPress()方法监听系统返回事件

当用户单击了系统的返回按钮,如果页面栈中当前页面下还存在其他页面,则会将当前页面出栈,显示出下面的页面。此时,可以通过 onBackPress()方法监听系统返回事件:在页面中实现该方法,代码如下:

```
onBackPress() {
    console.info("Page index : back button pressed!");
    return false;
}
```

当返回值为 false 时,可以正常地出栈当前页面。当返回值为 true 时,则不会出栈当前页面了,开发者可以自行在 onBackPress()方法中定义相关的执行动作,代码如下:

```
onBackPress() {
    //在这里实现返回执行动作
    return true;
}
```

5.1.6 应用对象

3min

在任何一个页面中,通过 this. $app 代码即可获取当前的应用对象。应用对象拥有自

身的生命周期,并且开发者可以在应用对象中实现 JavaScript 全局变量。

1. 应用的生命周期

应用的生命周期方法包括 onCreate()和 onDestory()方法。onCreate()方法在应用创建时调用,onDestory()方法则在应用销毁时调用。

应用对象由实例下的 app.js 进行定义,默认的代码如下:

```
//chapter5/JSRoute/entry/src/main/js/default/app.js
export default {
    onCreate() {
        console.info('AceApplication onCreate');
    },
    onDestroy() {
        console.info('AceApplication onDestroy');
    }
};
```

在上面的代码中,已经默认实现了应用的 2 个生命周期方法。这 2 个生命周期方法还是非常有用的:一方面,可以在这两种方法中实现数据库的管理。例如,在 onDestroy()中检查数据库是否关闭,如果未关闭则要立即关闭。另一方面,可以在 onCreate()方法中执行一些初始化操作,例如网络连接、账号核查等。

2. 共享应用对象的变量

应用对象是一个单例,在应用对象中定义的变量可以在所有页面中进行调用。在 5.1.4 节中,实现了页面跳转时的数据传递,但是并没有实现回跳时将数据传递。那么使用应用对象共享变量不失是一种数据传递的方法。

接下来,对 5.1.4 节中的 JSRoute 工程进行改造:在 app.js 中添加 1 个变量 jumpCount,用于记录用户的页面跳转次数。然后,创建该变量的 Get/Set 方法,代码如下:

```
//chapter5/JSRoute2/entry/src/main/js/default/app.js
export default {
    data : {
        jumpCount: null                    //页面的跳转次数
    },
    //获取页面的跳转次数
    getJumpCount(){
        return this.jumpCount;
    },
    //设置页面的跳转次数
    setJumpCount(count) {
        this.jumpCount = count;
    },
    //页面的跳转次数 + 1
    increaseJumpCount() {
```

```
        this.jumpCount ++;
    },
    onCreate() {
        this.jumpCount = 0;          //初始化页面的跳转次数为 0
        console.info('AceApplication onCreate');
    },
    onDestroy() {
        console.info('AceApplication onDestroy');
    }
};
```

随后,在跳转页面前调用 increaseJumpCount()方法即可记录页面的跳转次数。在 index.js 中,调用 router 模块的 push 方法前调用该方法,代码如下:

```
//chapter5/JSRoute2/entry/src/main/js/default/pages/index/index.js
//跳转到 secpage 的方法
toSecPage() {
    //页面的跳转次数 +1
    this.$app.increaseJumpCount();
    //输出当前的页面跳转次数
    console.info("getJumpCount: " + this.$app.getJumpCount());
    //通过 push 方法入栈
    router.push({
        //指定跳转位置
        uri: 'pages/secpage/secpage',
        //传递数据
        params: {
            "harmony": this.harmony
        }
    });
}
```

在 secpage.js 中,调用 router 模块的 back 方法前调用该方法,代码如下:

```
//chapter5/JSRoute2/entry/src/main/js/default/pages/secpage/secpage.js
//返回到之前的页面
back() {
    //页面的跳转次数 +1
    this.$app.increaseJumpCount();
    //输出当前的页面跳转次数
    console.info("getJumpCount: " + this.$app.getJumpCount());
    router.back();
}
```

编译并运行程序,此时每次在页面跳转时都会在 HiLog 中输出页面的跳转次数,典型

的输出如下：

```
21321 - 21971/? I 03B00/Console: app Log: AceApplication onCreate
21321 - 21971/? I 03B00/Console: app Log: getJumpCount: 1
21321 - 21971/? I 03B00/Console: app Log: getJumpCount: 2
21321 - 21971/? I 03B00/Console: app Log: getJumpCount: 3
...
```

5.2 常用组件和容器

组件是页面中用户界面的基本单位。在 JavaScript UI 中，通过 HML 定义页面结构，而页面中的组件通过 HML 中的标签定义。HML 和 HTML 非常类似，HML 定义了 HTML 中常用的< div >、< span >、< input >等组件，也定义了 HML 独有的< rating >、< chart >等组件。

容器是组件的一种。与其他组件不同的是，容器可以包含若干个组件，并且容器用于排布这些组件，类似于 Java UI 中的布局。例如，< div >就是一种常见容器。

在本节中，不再详细介绍< div >、< span >等 HTML 中已存在的组件（读者可以参考 HTML 语法及鸿蒙官方网站的 JS API 参考），本节详细介绍 HML 中常用且特有的组件。

9min

5.2.1 属性、事件和方法

通常，需要开发者通过属性定义组件的特征，通过事件绑定监听用户的交互信息，通过方法来控制组件的行为。

1. 属性

组件的属性是指组件的一些特性，例如 id 属性、style 属性、disabled 属性等。

id 属性可以唯一标识组件。在 js 代码中，可以通过 id 属性获取该组件的对象。例如，定义一个 id 属性为 mytext 的文本组件，代码如下：

```
<text id = "mytext" class = "title">
    这是一个文本
</text>
```

那么，在 js 代码中通过 $ element 方法即可获取该组件对象（DOM 元素），代码如下：

```
this. $ element('customMarquee')
```

另外，还可以通过 ref 属性标识组件。例如，定义文本组件的 ref 属性为 username，代码如下：

```
<text ref = "username" class = "title">
    这是一个文本
</text>
```

那么，在 js 代码中通过 $ ref 方法即可获取该组件对象(DOM 元素)，代码如下：

```
this. $ refs. username
```

这两种方法获取 DOM 元素的效果是一致的，只是风格不同，开发者可以二选其一。

另外，通过 style 属性可以定义组件的样式；通过 class 属性可以引用组件的样式；通过 disabled 属性可以定义组件是否可以被交互；通过 focusable 属性可以定义组件是否可以具有焦点；通过 show 属性可以定义组件的可见性。这些属性非常简单，并且多数与 HTML 中的各种元素属性非常类似，这里不进行详细介绍。

2. 事件

通过绑定事件可以获取用户的交互信息(单击、长按、按键等)。常见的通用事件如表 5-2 所示。

表 5-2 组件常见的通用事件

名 称	描 述	名 称	描 述
touchstart	刚触摸屏幕时触发	longpress	长按时触发
touchmove	触摸屏幕后移动时触发	focus	获得焦点时触发
touchcancel	触摸屏幕中动作被打断时触发	blur	失去焦点时触发
touchend	触摸结束离开屏幕时触发	key	当用户操作遥控器按键时触发(仅限智慧屏)
click	单击时触发		

组件在绑定事件时，需要子事件名称前加 on 或“@”来标示事件。例如，捕获文本组件的单击事件的典型代码如下：

```
<text class = "title" onclick = "textClicked">
    这是一个文本
</text>
```

或者，使用"@"来标示事件，代码如下：

```
<text class = "title" @click = "textClicked">
    这是一个文本
</text>
```

随后，在 js 文件中实现单击方法 textClicked()即可。

对于 key 事件来讲，可以通过其 KeyEvent 对象的 code 属性来判断操作的按键类型，通过 action 属性来判断按键操作。例如，当按下遥控器的确认键时才进行处理的代码如下：

```
onEnterKeyDown(keyevent){
    //当按下确认键才进行处理
    if (keyevent.code == 23 && keyevent.action == 0) {
        //处理代码
    }
}
```

当 action 为 0 时，表示按下按钮；当 action 为 1 时，表示松开按钮；当 action 为 2 时，表示长按按钮不松手。

code 属性所对应的按键类型如表 5-3 所示。

表 5-3 KeyEvent 的 code 属性及其所对应的物理按键

code 属性	对应的按键类型	code 属性	对应的按键类型
19	向上方向键	23	智慧屏遥控器的确认键
20	向下方向键	66	键盘回车键
21	向左方向键	160	键盘的小键盘回车键
22	向右方向键		

3. 方法

通过组件方法可以对组件进行控制。例如，对于跑马灯组件可以通过其 start 和 stop 方法来启动和结束跑马灯的滚动，其典型代码如下：

```
//启动跑马灯
this.$refs.mymarquee.start()
//结束跑马灯
this.$refs.mymarquee.stop()
```

组件没有通用的方法。在后面的章节中，会介绍常见组件的常用方法。

4. 样式

样式通过 css 文件定义，通用的样式可以设置组件的宽度、高度、边距、边框、背景、透明度、可见性等。

(1) 宽度和高度：通过 width 和 height 属性可以设置组件的宽度和高度。

(2) 边距：通过 padding 和 margin 属性可以设置组件的内边距和外边距。当然，还可以通过像素与 padding-top 等形式指定某个方向的边距。

(3) 边框：通过 border 属性可以定义边框的宽度、样式和颜色，也可以分别通过 border-width、border-style、border-color、border-radius 定义边框的宽度、样式、颜色、圆角半径等。

(4) 背景：通过 background、background-color、background-image 等属性可以定义背景样式。

另外，还可以通过 opacity 属性定义组件的透明度；通过 display 属性来定义组件为弹

性布局(flex)或者不渲染(none)；通过 visibility 属性定义组件的可见性等。

这些样式的用法不进行详细介绍，读者可以参考 Web 前端开发的相关资料和鸿蒙的 JS API 文档。

5.2.2 常用组件

19min

JavaScript UI 中的组件及其支持性如表 5-4 所示。

表 5-4 JavaScript UI 中的组件及其支持性

组 件	名 称	支持性		
		手机/平板/智慧屏	智能穿戴	轻量级智能穿戴
text	文本	√	√	√
span	文本行内修饰	√	√	
marquee	跑马灯	√	√	√
progress	进度条	√	√	√
divider	分隔器	√	√	
button	按钮	√	√	
input	输入(单选、多选、文本框、按钮等)	√	√	√
label	标注	√	√	
textarea	多行文本输入的文本框	√		
search	搜索框	√		
slider	滑动条	√	√	√
rating	评分条	√		
switch	开关选择器	√	√	√
picker	滑动选择器	√		
picker-view	嵌入页面的滑动选择器	√	√	√
menu	菜单	√		
select	下拉选择按钮	√		
option	select 和 menu 的子选项	√		
image	图像	√	√	√
image-animator	图片帧播放器	√	√	√
video	视频播放器	√		
chart	图表	√	√	√

本节介绍几种常用的组件。

1. 文本类组件

文本类组件包括< text >、< span >和< marquee >。< text >用于显示基本的文本内容，而< span >可以包括< text >中的一部分文本，并通过 CSS 类选择器定义其特殊的样式。< marquee >可以将文本以跑马灯的形式显示在页面中。

1）文本组件

<text>标签的内容即为文本组件的显示文本。接下来，用实例来讲明文本组件的用法，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_text/cpt_text.hml
<div class = "container">
    <text class = "title">
        这是一个文本
    </text>
    <text class = "title">
        <span class = "red">红色</span>
        <span class = "green">绿色</span>
        <span class = "blue">蓝色</span>
    </text>
</div>
```

在上面的代码中，第一个<text>组件显示文本“这是一个文本”；第二个<text>组件中包含了 3 个标签，并分别通过类选择器定义了其中不同文本的颜色。相关的类选择器代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_text/cpt_text.css
.title {
    font-size: 30px;                        /* 文本字号为 30px */
    text-align: center;                     /* 居中显示文本 */
}
.red {
    color: red;                             /* 文本颜色为红色 */
}
.blue {
    color: blue;                            /* 文本颜色为蓝色 */
}
.green {
    color: green;                           /* 文本颜色为绿色 */
}
```

以上代码的显示效果如图 5-15 所示。

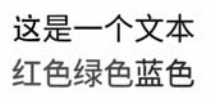


图 5-15　文本组件

注意：如果需要文本换行，则需要转义字符\r\n，而在<text>标签中的文本换行不会显示在最终显示效果中。

如果<text>标签内包含了至少一个标签，则没有被标签所包裹的文本部分将无法显示。例如，在下面的代码中，“这是一个”字符串没有包含在标签，代码如下：

```
<text class = "title">
    这是一个<span>文本</span>
</text>
```

那么，在实际的显示效果中，将不会显示“这是一个”字符串，只会显示“文本”字符串。

2）跑马灯组件

跑马灯组件能够将其中的文本以跑马灯的形式，从右到左（或从左到右）不断滚动，典型的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_text/cpt_text.hml
<marquee scrollamount = "50" loop = "3" direction = "right">
    这是一个跑马灯
</marquee>
```

跑马灯的3个属性及其功能分别为

（1）scrollamount：滚动速度，默认为6。

（2）loop：滚动次数，默认为－1。滚动次数为－1表示无限次滚动。

（3）direction：滚动方向，包括从左到右（left）、从右到左（right），默认为left。

跑马灯包括开始滚动（start）、滚动到末尾（bounce）和结束滚动（finish）等3个事件。另外，调用跑马灯的start和stop两种方法，可以开始和结束跑马灯的滚动。

2. 进度条

进度条组件为<progress>，包括横向进度条（horizontal）、无限进度条（circular）、环形进度条（ring）、带刻度环形进度条（scale-ring）、弧形进度条（arc）共5类，其类型可通过type属性进行定义。

除了无限进度条以外，均可通过present属性定义其进度。这其中，对于横向进度条、环形进度条和带刻度环形进度条来讲，还可以通过secondarypercent属性定义其副进度。

进度条的典型代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_progress/cpt_progress.hml
<div class = "container">
    <!-- 横向进度条 -->
    <progress type = "horizontal" percent = "50" secondarypercent = "70"/>
    <!-- 无限进度条 -->
    <progress type = "circular"/>
    <!-- 环形进度条 -->
    <progress type = "ring" percent = "50" secondarypercent = "70"/>
    <!-- 带刻度环形进度条 -->
    <progress class = "fixedSize" type = "scale-ring" percent = "50" secondarypercent = "70"/>
    <!-- 弧形进度条 -->
    <progress type = "arc" percent = "50"/>
</div>
```

其中，通过 fixedSize 类选择器固定了带刻度环形进度条的大小，代码如下：

```
.fixedSize {
    width: 260px;
    height: 260px;
}
```

上面的代码的显示效果如图 5-16 所示。

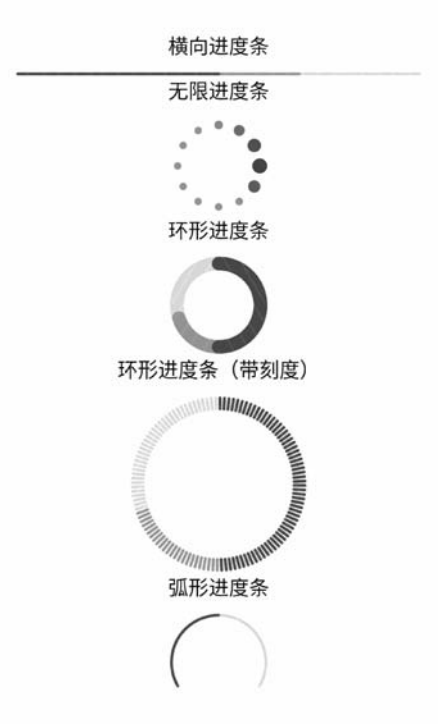


图 5-16　进度条组件

开发者可以根据需要选择所需要的进度条类型。

3. 常用交互类组件

交互类组件非常多，包括< button >按钮、< input >输入、< rading >评分条、< slider >滑动条、< switch >开关选择器等。

1）按钮

按钮可以通过按钮控件< button >或者输入控件< input >实现，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_interactive/cpt_interactive.hml
<!-- button 按钮 -->
< button > button 按钮</button >
<!-- input 按钮 -->
< input type = "button" value = "input 按钮"/>
```

这两个按钮的显示效果如图 5-17 所示。

通过输入控件< input >实现按钮，需要将其类型(type)设置为 button，并且其按钮内容需要通过 value 属性进行设置。

图 5-17 按钮组件的实现

除了 button 之外，输入控件< input >的类型还包括 checkbox(复选框)、radio(单选框)、text(普通文本框)、email(E-mail 文本框)、date(日期文本框)、time(时间文本框)、number(数字文本框)、password(密码文本框)。

2) 单选框与复选框

当将输入控件< input >的类型设置为 checkbox 时，该组件为复选框；当将其类型设置为 radio 时，该组件为单选框。

由于无法通过< input >设置单选框和复选框的文字提示，因此通常需要配合< label >组件使用。为了让< label >组件绑定到< input >组件，首先需要设置< input >组件的 id 属性，然后将< label >组件的 target 属性指定为< input >组件的 id，如图 5-18 所示。

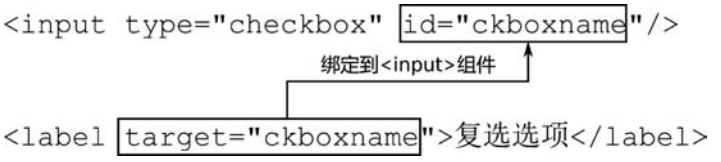


图 5-18 将< label >组件绑定到< input >组件

单选框组件通常为一组，同时需要将一组单选框组件的 name 属性设置为同一个字符串。

复选框和单选框的典型代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_interactive/cpt_interactive.hml
<!-- 复选框 -->
<text>复选框</text>
<div class="row">
    <input type="checkbox" id="checkbox" checked="true"/>
    <label target="checkbox">复选选项</label>
</div>
<!-- 单选框 -->
<text>单选框</text>
<div class="row">
    <input type="radio" id="radio1" name="group" value="1"/>
    <label target="radio1">单选选项 1 </label>
    <input type="radio" id="radio2" name="group" value="2" checked="true"/>
    <label target="radio2">单选选项 2 </label>
</div>
```

通过< input >的 check 属性可设置复选框或单选框的选中状态。通过< input >的 change 事件可监听其选中状态的变化情况。

另外,为了能够将< label >与其所在的< input >水平放置,因此将其放置在一个单独的< div >中,并将其类选择器设置为 row,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_interactive/cpt_interactive.css
.row {
    flex-direction: row;
    justify-content: center;
}
```

上述代码的显示效果如图 5-19 所示。

此时,单击【单选选项 1】时,【单选选项 2】会被自动取消。即【单选选项 1】和【单选选项 2】只能二选一,因为这两个组件的 group 属性相同。

图 5-19 复选框和单选框

3) 文本框

文本框也可以通过输入组件< input >实现,不过多行文本框和搜索文本框则分别需要通过< textarea >和< search >组件实现。

常用的文本框的代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_interactive/cpt_interactive.hml
<!-- 普通文本框 -->
<input type="text" value="" placeholder="请输入文本"/>
<!-- E-mail 文本框 -->
<input type="email" value="" placeholder="请输入 E-mail"/>
<!-- 数字文本框 -->
<input type="number" value="" placeholder="请输入数字"/>
<!-- 密码文本框 -->
<input type="password" value="" placeholder="请输入密码"/>
<!-- 日期文本框 -->
<input type="date" value="" placeholder="请输入日期"/>
<!-- 时间文本框 -->
<input type="time" value="" placeholder="请输入时间"/>
<!-- 多行文本框 -->
<textarea placeholder="多行文本框"></textarea>
<!-- 搜索文本框 -->
<search hint="搜索文本框"/>
```

以上代码的显示效果如图 5-20 所示。

注意:除了搜索文本框以外,文本框提示均可以通过 placeholder 属性进行设置。搜索文本框的提示通过 hint 属性进行设置。

通过<input>组件实现的文本框所涉及的 type 属性包括 text(普通文本框)、email(E-mail 文本框)、number(数字文本框)、password(密码文本框)、date(日期文本框)、time(时间文本框)。各类文本框的区别主要体现在弹出的键盘类型上。根据<input>组件类型的不同,文本框键盘类型也会适配。例如,普通文本框弹出的键盘为输入法的默认键盘类型,而数字文本框弹出的键盘类型为数字键盘,如图 5-21 所示。

图 5-20 各种文本框

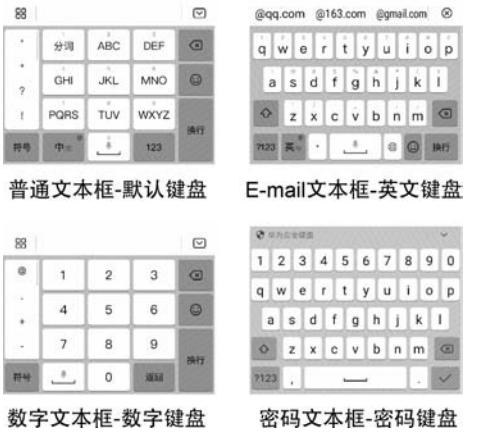


图 5-21 各类文本框及其所对应的键盘类型

对于密码文本框而言,默认不显示用户输入的字符,除非用户主动点选了文本框右侧的按钮将密码切换为可见状态。

<input>组件实现的文本框还可通过其 enterKeyType 属性定义其软键盘的回车类型,其值可以为 default(默认)、next(下一项)、go(前往)、done(完成)、send(发送)和 search(搜索)。

<input>组件实现的文本框主要包括 2 个事件:

(1) change(inputValue):当输入框的内容发生变化时触发,返回的参数为变化后的文本框内容字符串。

(2) enterkeyclick:当单击了软键盘的回车按钮(其文本由 enterKeyType 定义)时触发。

多行文本框<textarea>和搜索文本框<search>与<input>组件的外观和用法非常类似,因篇幅有限不进行详细叙述。

4）滑动条

滑动条组件< slider >可以让用户通过滑动的方式选择数值，通常可以用于调整音量、图片透明度、视频播放进度等，典型的代码如下：

```
< slider value = "40"/>
```

这里将滑动条选择的数值设置为 40（默认数值范围为 0～100），其显示效果如图 5-22 所示。

滑动条组件< slider >的常用属性如下：

- min：最小值，默认为 0。
- max：最大值，默认为 100。
- step：最小滑动步长，默认为 1。
- value：初始值，默认为 0。

另外，通过滑动条组件< slider >的 change（progressValue）事件可以监听用户的滑动数值变化。

5）评分条

评分条< rating >的功能性更加专一，通常用于评价目标（音乐、视频、应用等）的等级，典型的代码如下：

```
< rating rating = "3.5"/>
```

默认情况下，评分条的登记在 0～5 范围内，且步长为 0.5。通过 rating 属性可以定义评分的默认值，其值 3.5 表示三星半，上述代码的显示效果如图 5-23 所示。

图 5-22　滑动条组件

图 5-23　评分条组件

评分条组件< rating >的常用属性如下：

- numstars：评分最大值，默认为 5。
- rating：评分默认值，默认为 0。
- stepsize：评分步长，默认为 0.5。
- indicator：当该属性为 true 时，评分条无法交互，默认值为 false。

另外，通过评分条组件< rating >的 change（currentRating）事件可以监听用户的评分变化。

6）开关选择器

开关选择器< switch >具有打开和关闭两种状态，用户可以通过单击的方式切换开关选择器的开关状态，其典型的代码如下：

```
<switch checked = "true" showtext = "true" texton = "启动" textoff = "停用"/>
```

上述代码的显示效果如图 5-24 所示。

图 5-24 开关选择器组件

开关选择器<switch>的常用属性如下：

- checked：开关状态，默认值为 false。
- showtext：是否显示文本，默认值为 false。
- texton：打开时显示的文本内容，默认为 On。
- textoff：关闭时显示的文本内容，默认为 Off。

另外，通过开关选择器<switch>的 change(checkedValue)事件可以监听用户的开关交互动作。

4. 滑动选择器

滑动选择器组件<picker>可以让用户通过滑动的方式选择选项和数值，包括文本选择器、日期选择器等不同类型。滑动选择器的类型通过其 type 属性定义，其值及所对应的选择器类型如下所示：

- text：文本选择器。
- multi-text：多列文本选择器。
- date：日期选择器。
- time：时间选择器。
- datetime：日期时间选择器。

对于文本选择器和多列文本选择器来讲，还需要通过 range 属性定义其选择范围(数组)，通过 selected 和 value 属性定义选择的文本数组索引或其值。日期选择器、时间选择器和日期时间选择器也存在相应的选项用于设置其选择范围，这里不进行详述。

不过，在 HML 文件中定义的<picker>组件不会直接显示在界面中，需要通过代码的方式调用其 show()方法才会将其以模态的方式显示到界面中。

下面，分别创建 5 个按钮组件和 5 个不同类型的滑动选择器组件，单击按钮显示对应的滑动选择器，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_picker/cpt_picker.hml
<button @click = "showTextPicker">文本选择器</button>
<picker id = "picker - text" type = "text" range = "{{options}}"></picker>
<button @click = "showMultiTextPicker">多列文本选择器</button>
<picker id = "picker - multi - text" type = "multi - text" range = "{{multi_text_options}}">
</picker>
<button @click = "showDatePicker">日期选择器</button>
<picker id = "picker - date" type = "date"></picker>
<button @click = "showTimePicker">时间选择器</button>
<picker id = "picker - time" type = "time"></picker>
<button @click = "showDateTimePicker">日期时间选择器</button>
<picker id = "picker - datetime" type = "datetime"></picker>
```

在 js 文件中，创建 options 和 multi_text_options 数组，并且实现 5 个按钮单击事件的处理方法，分别获取对应的滑动选择器对象并显示在界面中，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_picker/cpt_picker.js
data: {
    options:['选项 1', '选项 2', '选项 3'],
    multi_text_options: [
        ['男', '女'],
        ['程序员', '项目经理', '学生', '公务员']
    ]
},
showTextPicker() {
    this.$element("picker-text").show();
},
showMultiTextPicker() {
    this.$element("picker-multi-text").show();
},
showDatePicker() {
    this.$element("picker-date").show();
},
showTimePicker() {
    this.$element("picker-time").show();
},
showDateTimePicker() {
    this.$element("picker-datetime").show();
}
```

编译运行程序并进入上述界面，可以在界面中看到 5 个用于打开滑动选择器的按钮，但是滑动选择器并没有显示在界面中，如图 5-25 所示。

图 5-25　显示滑动选择器的 5 个按钮

单击这 5 个按钮，其对应类型的选择器显示效果如图 5-26 所示。

如果开发者希望滑动选择器直接显示在界面上，而并不是以模态的方式弹出，则可以尝试使用 < picker-view >组件。< picker-view >组件和< picker >组件的属性、事件基本类似，不再赘述。

注意：< picker-view >组件的设备支持性更强，而< picker >组件不支持可穿戴设备和轻量级可穿戴设备。

5. 菜单与下拉选择按钮

菜单和下拉选择按钮的使用方法比较类似，都是以弹出选项按钮的方式让用户选择。

1）菜单

菜单< menu >组件需要包括多个< option >组件，其中每个< option >组件都是一个选

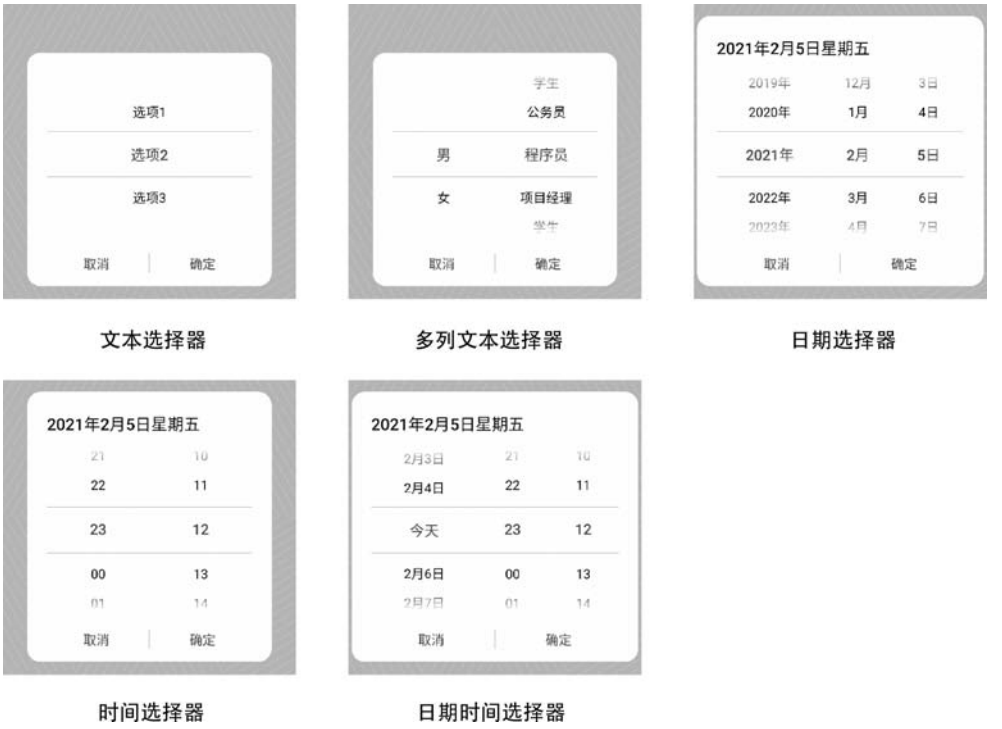


图 5-26 5 种类型的滑动选择器

项。不过,HML 中的< menu >组件并不会直接显示在界面中,需要调用其 show()方法才会弹出显示菜单。这里通过单击< button >按钮的方式弹出菜单,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_menuandselect/cpt_menuandselect.hml
< button @click = "showMenu">显示菜单</button >
< menu id = "menu">
    < option value = "opt1">菜单选项 1 </option >
    < option value = "opt2">菜单选项 2 </option >
    < option value = "opt3">菜单选项 3 </option >
</menu >
```

< button >按钮的单击处理方法,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_menuandselect/cpt_menuandselect.js
showMenu() {
    this. $ element('menu'). show();
}
```

运行以上代码,单击【显示菜单】后,弹出的菜单如图 5-27 所示。

2）下拉选择按钮

下拉选择按钮< select >的使用方法更加简单，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_menuandselect/cpt_menuandselect.hml
< select >
    < option value = "opt1">选项 1 </option >
    < option value = "opt2">选项 2 </option >
    < option value = "opt3">选项 3 </option >
</select >
```

上述代码会在界面中显示一个按钮，其右侧的▼图标标识了该按钮为一个下拉选择按钮，其显示效果如图 5-28 所示。

图 5-27　菜单组件

图 5-28　下拉选择组件

6. 图像

图像组件< image >可以加载并显示图形图像，支持 SVG、PNG 等多种格式，包括了 2 个常用属性：

（1）src：指定图形图像位置，通常图片资源需要放置在实例的 common 目录中。

（2）alt：在图形图像未加载完成前，占位显示的文字内容。图形图像加载成功可通过 complete 事件处理回调；图形图像加载失败则可通过 error 事件处理回调。

图像组件< image >的典型代码如下：

```
< image src = "/common/img.png" alt = "加载图片..."></image >
```

在上述代码中，图像组件< image >加载实例 common 目录中的 img.png 图片，效果如图 5-29 所示。

通常，图形图像的尺寸难以与图像组件的尺寸刚好吻合，因此常常需要通过缩放模式来改变图形图像的尺寸和位置。缩放模式可以通过其 object-fit 样式进行设置，其各个值所代表的意义如表 5-5 所示。

表 5-5 图像组件的缩放模式

值	描 述
cover	保持原始比例居中并填满组件大小
contain	保持原始比例居中并完整地显示图形图像内容。当图形图像比组件大时，则缩小图形图像直至能够完整显示图形图像
fill	拉伸图形图像充满整个组件大小
none	保持原始大小居中
scale-down	居中显示，保持原始比例填充组件的宽度或高度，并完整显示图形图像内容

为了能够便于开发者理解，这里通过 1 个比组件小的图标图像和 1 个比组件大的照片图像来演示这几种缩放模式的区别，如图 5-30 所示。

图 5-29 图像组件

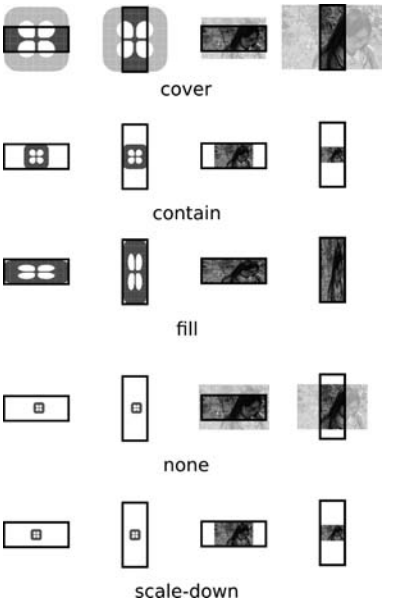


图 5-30 图像组件的缩放模式

图像组件也支持网络图片的加载，直接在 src 中填入网络图片的网址即可。例如，可以通过 https 协议获取互联网的图片，代码如下：

```
<image src="https://***/***.png" alt="加载图片..."></image>
```

不过，不要忘记在 config.json 中为该应用添加网络访问权限，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/config.json
{
  …
  "module": {
```

```
    "package": "com.example.JavaScriptui",
    "reqPermissions": [
      {
        "name": "ohos.permission.INTERNET"
      }
    ],
  }
  …
}
```

7. 视频播放器

视频播放器组件< video >的集成度非常高，与图像组件的用法也非常类似，通过 src 属性指定视频文件(同样可以为网络视频)。在默认情况下，< video >组件包含了控制栏(用于播放、暂停、拖动播放进度、全屏等)，不过也可以通过 start、pause 等方法控制视频组件的播放功能，典型的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_video/cpt_video.hml
<div class = "container">
    <video id = "video" src = "/common/test.mp4"></video>
    <button @click = "startVideo">开始视频</button>
    <button @click = "pauseVideo">暂停视频</button>
</div>
```

startVideo 和 pauseVideo 的方法实现代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_video/cpt_video.js
export default {
    startVideo() {
        this.$element('video').start();
    },
    pauseVideo() {
        this.$element('video').pause();
    }
}
```

为了控制视频播放器的尺寸，在 css 文件中的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/cpt_video/cpt_video.css
video {
    height: 540px;
    width: 720px;
}
```

上述代码的显示效果如图 5-31 所示。

视频组件< video >的常用属性如下：

- src：视频文件路径。
- muted：是否静音播放。
- autoplay：是否加载后自动开始播放。
- poster：视频停止时显示的预览图片。
- controls：是否显示视频播放控制栏。

视频组件< video >的常用事件如下：

- prepared：视频准备完成时触发该事件。
- start：播放时触发该事件。
- pause：暂停时触发该事件。
- finish：播放结束时触发该事件。
- error：播放失败时触发该事件。

图 5-31 视频组件

5.2.3 常用容器

14min

JavaScript UI 的常用容器包括基础容器< div >、列表容器< list >、堆叠容器< stack >、滑动容器< swiper >和页签容器< tabs >。其中，页签容器不支持可穿戴设备和轻量级可穿戴设备，其他容器支持各种设备。

1. 基础容器

与 HTML 类似，HML 中的< div >属于布局中的基础容器，也是应用最为广泛的容器。< div >通过 display 样式定义了 3 种布局类型：弹性布局(flex)、网络布局(grid)和 none。当 display 样式为 none 时，将不显示(隐藏)< div >容器及其内容。

1) 弹性布局

默认情况下，< div >容器为弹性布局。弹性布局是沿着某个方向(横向或纵向)依次排列组件的布局，这个方向被称为主轴。与主轴垂直的方向被称为交叉轴。

注意：从定义上看，弹性布局和 Java UI 中的定向布局非常类似。不过弹性布局更加灵活，如果在主轴方向上组件排列不下，则可以换行(换列)排列。从这个特性上来看，弹性布局又具备了 Java UI 中自适应布局的特点。

通过< div >的 flex-direction 样式定义弹性布局的主轴方向，其值可以为 column(纵向从上到下)和 row(横向从左到右)。当主轴为垂直方向时，交叉轴为水平方向，反之亦然。例如，将某个< div >的类选择器定义为 container，其中包含了 3 个文本组件，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_div_flex/ctn_div_flex.hml
<div class = "container">
    <text class = "title">组件 1 </text >
    <text class = "title">组件 2 </text >
    <text class = "title">组件 3 </text >
</div >
```

随后，定义 container 和 title 的类选择器样式，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_div_flex/ctn_div_flex.css
.container {
    display: flex;                          /* 弹性布局 */
    flex-direction: column;                 /* 弹性布局垂直方向 */
}

.title {
    width: 300px;                           /* 宽度 300px */
    height: 100px;                          /* 高度 100px */
    background-color: bisque;               /* 背景颜色 */
}
```

上述代码的显示效果如图 5-32 所示。

当基础组件< div >的 flex-direction 样式为 column 或 row 时，其主轴和交叉轴的方向如图 5-33 所示。

图 5-32　基础容器的弹性布局

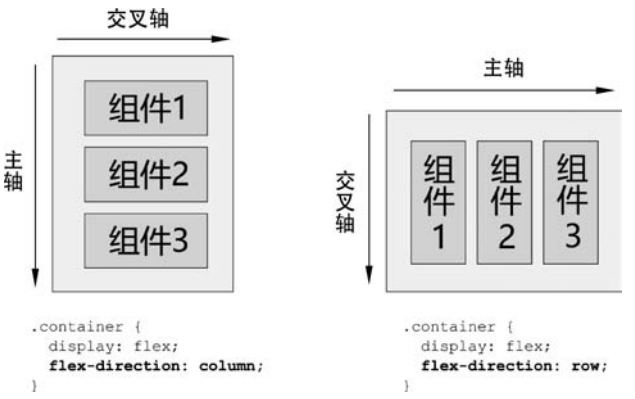


图 5-33　主轴与交叉轴

flex-wrap 样式定义了弹性容器是否可以换行，其值可以为 nowrap（不换行）和 wrap（换行）。当 flex-wrap 样式为 wrap 时，如果主轴没有足够空间放下所有的组件，则会换行（当弹性布局为纵向时为换列）显示组件，如图 5-34 所示。

除了上述样式以外，还包括几种重要的对齐方式样式，如下所示：

（1）justify-content 样式定义主轴的对齐方式，包括依靠起始位（flex-start）、依靠结束位（flex-end）、居中（center）、平均放置且前后端不留空白（space-between）、平均放置且前后端留空白（space-around）。这些对齐方式的显示效果如图 5-35 所示。

（2）align-items 样式定义了交叉轴对齐方式，包括拉伸组件到容器宽度（stretch）、依靠起始位（flex-start）、依靠结束位（flex-end）、居中（center）。

注意：align-items 样式的 stretch 值仅适用于弹性尺寸的组件和容器。

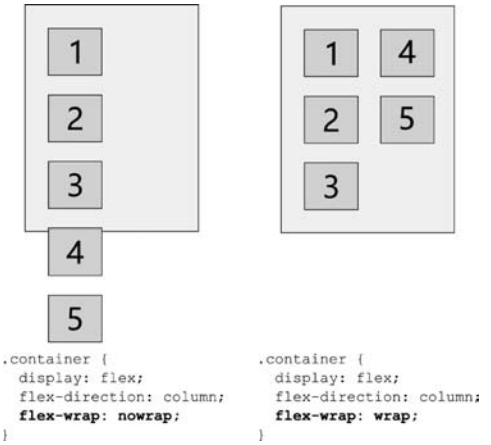


图 5-34 换行样式 flex-wrap

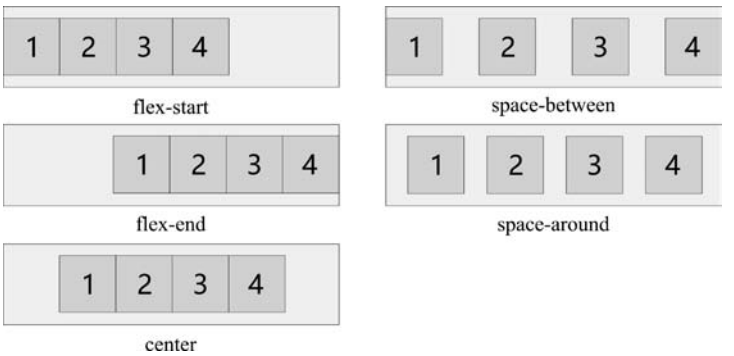


图 5-35 主轴对齐方式

(3) 换行样式为 wrap 时,align-content 定义了多行对齐方式,包括依靠起始位(flex-start)、依靠结束位(flex-end)、居中(center)、平均放置且前后端不留空白(space-between)、平均放置且前后端留空白(space-around)。这些对齐方式的效果与 justify-content 样式的相应对齐方式非常类似,如图 5-35 所示。

2) 网格布局

通过网格布局可以让其所包含的组件以对齐行列的方式显示在界面中。当< div >容器的 display 样式为 grid 时,该容器即为网格布局。与网格布局相关的主要样式如表 5-6 所示。

表 5-6 网格布局的相关样式

值		描 述
由网格布局定义	grid-template-columns	列数及其宽度
	grid-template-rows	行数及其宽度
	grid-columns-gap	列间距
	grid-rows-gap	行间距

续表

值		描　述
由网格布局内的组件定义	grid-row-start	组件所在网格布局的起始行号
	grid-row-end	组件所在网络布局的结束行号
	grid-column-start	组件所在网格布局的起始列号
	grid-column-end	组件所在网络布局的结束列号

接下来，创建一个 3 行 3 列网格布局，放置 4 个组件组成一个如图 5-36 所示的形状。其中，每个形状都跨行或跨列。左上角的组件 A 处在第 0 列且横跨第 0 行和第 1 行。右上角的组件 B 处在第 0 行且横跨第 1 列和第 2 列。左下角的组件 C 处在第 2 行且横跨第 0 列和第 1 列。右下角的组件 D 处在第 2 列且横跨第 1 行和第 2 行。

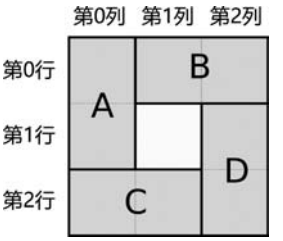


图 5-36　将 4 个组件放置到网格布局中（设计图稿）

为了实现上述的效果，创建 1 个网格布局< div >组件，其网格布局的相关属性由 container 类选择器定义。随后，在网格布局中创建 4 个子组件，其所处行列的位置由 left-top（对应组件 A）、right-top（对应组件 B）、left-bottom（对应组件 C）和 right-bottom（对应组件 D）这 4 个类选择器定义。代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_div_grid/ctn_div_grid.hml
<div class = "container">
    <div class = "grid left - top"></div>
    <div class = "grid left - bottom"></div>
    <div class = "grid right - top"></div>
    <div class = "grid right - bottom"></div>
</div>
```

在 CSS 文件中定义 container、grid 等类选择器样式，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_div_grid/ctn_div_grid.css
.container {
    width: 400px;
    height: 400px;
    display: grid;                                  /* 网格布局 */
    grid - template - columns: 1fr 1fr 1fr;         /* 分列 */
    grid - template - rows: 1fr 1fr 1fr;            /* 分行 */
    grid - columns - gap: 20px;                     /* 列间距 */
    grid - rows - gap: 20px;                        /* 行间距 */

}
.grid {
```

```
    width: 100%;
    height: 100%;
}
.left-top {
    grid-row-start: 0;
    grid-row-end: 1;
    grid-column-start: 0;
    grid-column-end: 0;
    background-color: red;
}
.left-bottom {
    grid-row-start: 2;
    grid-row-end: 2;
    grid-column-start: 0;
    grid-column-end: 1;
    background-color: green;
}
.right-top {
    grid-row-start: 0;
    grid-row-end: 0;
    grid-column-start: 1;
    grid-column-end: 2;
    background-color: blue;
}
.right-bottom {
    grid-row-start: 1;
    grid-row-end: 2;
    grid-column-start: 2;
    grid-column-end: 2;
    background-color: purple;
}
```

上述代码的实现效果如图 5-37 所示。

图 5-37 将 4 个组件放置到网格布局中

grid-template-columns 和 grid-template-rows 的定义方式存在以下几种：

(1) 通过像素定义。例如，定义行数为 3，且其高度分别为 50px、60px 和 70px，代码如下：

```
grid-template-rows: 50px 60px 70px;
```

如果需要定义多个固定高度的行，则可以尝试以下代码：

```
grid-template-rows: repeat(10,20px);
```

其中，repeat 的第 1 个参数为行数，第 2 个参数为每行的高度，因此上面的代码定义了 10 行，且每行高度均为 20px。这行代码等价于以下代码：

```
grid-template-rows: 20px 20px 20px 20px 20px 20px 20px 20px 20px 20px;
```

（2）通过行列占比定义。例如，定义行数为 3，且其所占比例分别为 20%、30%和 50%，代码如下：

```
grid-template-rows: 20% 30% 50%;
```

（3）通过行列权重定义。例如，定义行数为 3，且其所占布局权重分别为 1 份、2 份和 3 份，那么这 3 行分别占据整个布局高度的 1/6、1/3 和 1/2，代码如下：

```
grid-template-rows: 1fr 2fr 3fr;
```

另外，还可以通过 auto 来自适应宽度。例如，第 1 行的高度根据内部组件自适应，其余的空间分别按照 1 份和 2 份来分配高度，代码如下：

```
grid-template-rows: auto 1fr 2fr;
```

2. 列表容器

列表容器<list>可以方便地显示列表项<list-item>和列表项组<item-item-group>。列表容器只能直接包含<list-item>和<item-item-group>标签，且<item-item-group>只能直接包含<list-item>标签。列表内容所需要的组件，例如<text>、<image>等都需要放置到<list-item>中。

列表项组<item-item-group>可以包含多个<list-item>，单击列表项组可以折叠或者展开其内部的列表项。列表项组的第 1 个<list-item>会显示在列表项组上，并作为其内容显示在界面中。

列表容器的使用比较简单，这里仅用 1 个实例来展示其用法。首先，定义包含列表项和列表项组的列表容器，代码如下：

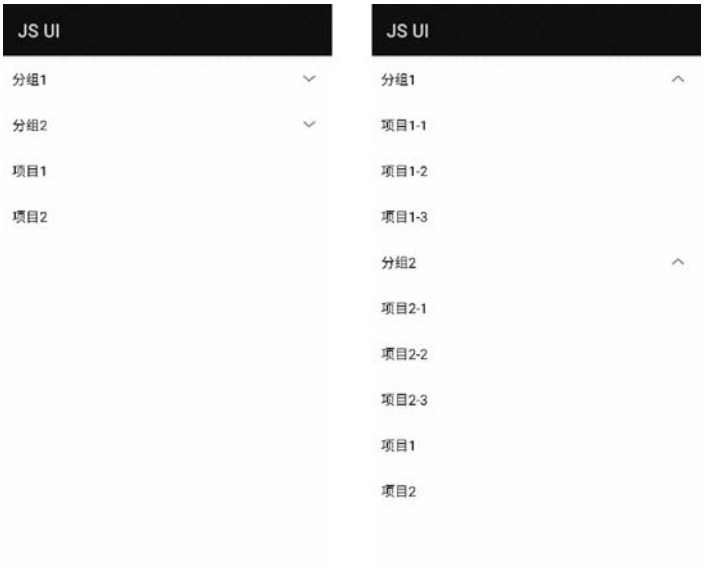
```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_list/ctn_list.hml
<list>
    <list-item-group>
```

```
        <list-item class="item"><text>分组 1</text></list-item>
        <list-item class="item"><text>项目 1-1</text></list-item>
        <list-item class="item"><text>项目 1-2</text></list-item>
        <list-item class="item"><text>项目 1-3</text></list-item>
    </list-item-group>
    <list-item-group>
        <list-item class="item"><text>分组 2</text></list-item>
        <list-item class="item"><text>项目 2-1</text></list-item>
        <list-item class="item"><text>项目 2-2</text></list-item>
        <list-item class="item"><text>项目 2-3</text></list-item>
    </list-item-group>
    <list-item class="item"><text>项目 1</text></list-item>
    <list-item class="item"><text>项目 2</text></list-item>
</list>
```

然后,在 CSS 文件中定义类选择器 item,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_list/ctn_list.css
.item {
    width: 100%;
    height: 100px;
    margin-left: 20px;
}
```

上述代码的显示效果如图 5-38 所示。



折叠效果　　展开效果

图 5-38　列表容器

3. 堆叠容器

堆叠容器中的组件会依次堆叠起来，非常类似于 Java UI 中的堆叠布局。接下来，以一个实例来展示堆叠容器的显示效果，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_stack/ctn_stack.hml
<stack class = "container">
    <div class = "stack1"></div>
    <div class = "stack2"></div>
    <div class = "stack3"></div>
    <div class = "stack4"></div>
    <div class = "stack5"></div>
</stack>
```

类选择器为 stack1、stack2 等的 组件按照次序堆叠到界面中。在 CSS 文件中，根据叠加次序将这 5 个 组件分别设置不同的宽度和高度，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_stack/ctn_stack.css
.container {
    width: 100 % ;
    height: 100 % ;
    justify - content: center;
    align - items: center;
}
.stack1 {
    background - color: yellow;
    width: 500px;
    height: 500px;
}
.stack2 {
    background - color: hotpink;
    width: 400px;
    height: 400px;
}
.stack3 {
    background - color: red;
    width: 300px;
    height: 300px;
}
.stack4 {
    background - color: blue;
    width: 200px;
    height: 200px;
}
.stack5 {
    background - color: green;
    width: 100px;
    height: 100px;
}
```

以上代码的最终显示效果如图 5-39 所示。

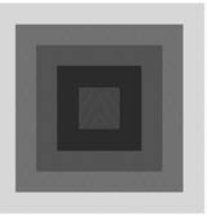


图 5-39 堆叠容器

4. 滑动容器

滑动容器可以让用户以滑动的方式切换用户界面，是一种常用且重要的内容展示容器。接下来，以滑动切换图片为例，介绍滑动容器的用法。

首先定义滑动容器及其内部组件，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_swiper/ctn_swiper.hml
<swiper class="swiper">
    <image src="/common/swiper1.png"/>
    <image src="/common/swiper2.png"/>
    <image src="/common/swiper3.png"/>
</swiper>
```

滑动容器的常用属性如下：

(1) index：当前显示的组件索引，默认为 0。

(2) autoplay：是否自动播放，默认值为 false。

(3) interval：自动播放时切换组件的时间间隔，单位为 ms，默认为 3000ms。

(4) loop：组件切换是否循环，默认值为 true。当该值为 true 时，如果当前显示的是最后一个组件，则继续切换组件即可显示第一个组件。

(5) duration：切换组件的动画时长。

(6) indicator：是否显示导航点指示器，默认值为 true。

(7) indicatormask：是否显示导航点指示器模板，默认值为 false。

(8) vertical：是否为纵向滑动，默认值为 false。

随后，定义类选择器 swiper，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_swiper/ctn_swiper.css
.swiper {
    width: 100%;
    height: 200px;
    indicator-color: white;
    indicator-selected-color: blue;
    indicator-size: 14px;
    indicator-bottom: 20px;
    indicator-right: 30px;
}
```

关于导航点指示器相关的样式说明如下：

(1) indicator-color：导航点的颜色。

(2) indicator-selected-color：选中导航点的颜色。

(3) indicator-size：导航点大小。

(4) indicator-left：导航点指示器距离< swiper >左侧的距离。

(5) indicator-right：导航点指示器距离< swiper >右侧的距离。

(6) indicator-top：导航点指示器距离< swiper >上方的距离。

(7) indicator-bottom：导航点指示器距离< swiper >下方的距离。

上述代码的显示效果如图 5-40 所示。

另外，还可以通过 swipeTo(index)方法跳转到指定的组件位置；通过 showNext()方法跳转到下一个组件；通过 showPrevious()方法跳转到上一个组件。

图 5-40 滑动容器

5. 页签容器

页签容器< tabs >可以容纳多个页面< tab-content >，用户可以通过选择< tab-bar >或滑动的方式切换这些页签，因此，< tabs >内只能直接容纳< tab-bar >和< tab-content >这两种组件。< tab-content >中的每个子组件(或容器)是用户切换页签的对象。

页签容器的典型代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_tabs/ctn_tabs.hml
<tabs>
    <tab-bar class="tabbar" mode="fixed">
        <text class="tab-text">页面 1</text>
        <text class="tab-text">页面 2</text>
        <text class="tab-text">页面 3</text>
    </tab-bar>
    <tab-content class="tabcontent" scrollable="true">
        <div class="item-content">
            <text class="item-title">页面 1</text>
        </div>
        <div class="item-content">
            <text class="item-title">页面 2</text>
        </div>
        <div class="item-content">
            <text class="item-title">页面 3</text>
        </div>
    </tab-content>
</tabs>
```

< tabs >的常见属性如下：

(1) index：当前显示的组件(容器)索引，默认为 0。

(2) vertical：是否纵向显示组件(容器)，默认值为 false，即横向显示组件(容器)。

< tab-bar >的 mode 属性包括 scrollable 和 fixed 两类。当 mode 为 scrollable 时，其内部的组件宽度由其自身组件大小决定，如果子组件大小超过了< tab-bar >的宽度，用户可以滑动< tab-bar >查看或选择这些子组件。

<tab-content>的 scrollable 属性指定了用户是否可以通过滑动的方式切换子组件(容器),默认值为 true。

在 CSS 文件中,定义 tabbar、tabcontent 等类选择器,使<tab-bar>占据 10%的屏幕空间,使<tab-content>占据 90%的屏幕空间,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/ctn_tabs/ctn_tabs.css
.tabbar {
    width: 100%;
    height: 10%;
}
.tabcontent {
    width: 100%;
    height: 90%;
}
.item-content {
    height: 100%;
    justify-content: center;
}
.item-title {
    font-size: 60px;
}
```

以上代码的显示效果如图 5-41 所示。

图 5-41 页签容器

5.2.4 对话框

4min

对话框可以通过两种方式实现:通过<dialog>组件实现并弹出对话框;通过 prompt 模块弹出对话框。相对来讲,使用 prompt 模块会更加方便一些,而通过<dialog>组件的方

式的自定义能力更强，下面介绍这两种实现方式。

1. 通过<dialog>组件实现并弹出对话框

开发者在 HML 中定义对话框组件<dialog>中的内容，包含 2 个重要的方法：显示对话框 show()和关闭对话框 close()。<dialog>并不会直接显示在界面中，而是需要调用<dialog>组件的 show()方法才能显示在界面中。使用<dialog>的典型代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/dialogtest/dialogtest.hml
<button class = "btn" @click = "showDialog">通过 dialog 组件显示对话框</button>
<dialog id = "dialog">
    <div class = "dialog - content">
        <text>提示信息</text>
        <button class = "btn" @click = "closeDialog">确认</button>
        <button class = "btn" @click = "closeDialog">取消</button>
    </div>
</dialog>
```

在 CSS 文件中定义 btn 和 dialog-content 类选择器，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/dialogtest/dialogtest.css
.btn {
    margin: 10px;
}

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

单击【通过 dialog 组件显示对话框】按钮后，调用 showDialog()方法显示对话框；单击对话框内的【确认】或【取消】按钮后，调用 closeDialog()方法关闭对话框，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/dialogtest/dialogtest.js
showDialog() {
    this.$element('dialog').show();
},
closeDialog() {
    this.$element('dialog').close();
}
```

上述代码的对话框显示效果如图 5-42 所示。

注意：对话框组件<dialog>不支持轻量级可穿戴设备。

2. 通过 prompt 模块弹出对话框

通过 prompt 模块弹出对话框非常简单，只需调用 prompt 的 showDialog 方法或

图 5-42 通过< dialog >组件实现并弹出对话框

showToast 方法,前者用于弹出一般对话框,后者用于弹出 Toast 对话框。

在使用这两种方法之前,需要在 js 文件中导入 prompt 模块,代码如下:

```
import prompt from '@system.prompt';
```

1) 通过 showDialog 方法弹出一般对话框

使用 showDialog 方法需要传入 Dialog 对象,主要包含以下几个属性:

(1) title: 对话框的标题。

(2) message: 对话框的提示信息。

(3) buttons: 对话框按钮数组,其中每个按钮对象都包括按钮文字 text 和按钮颜色 color。

(4) success: 当用户单击了对话框的按钮后回调并关闭对话框。

(5) cancel: 当用户单击了对话框之外的空白处,或触发了系统的返回事件后回调并关闭对话框。

(6) complete: 对话框关闭后回调。

通过 showDialog 方法弹出一般对话框的代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/dialogtest/dialogtest.js
showDialogByPrompt() {
    prompt.showDialog({
        title: "对话框标题",
        message: "对话框信息",
        buttons: [
            {text:'按钮 1', color: '#666666'},
            {text:'按钮 2', color: '#666666'}
        ],
        success: function(data) {
            console.info('对话框已选择,选择按钮为: ' + data.index);
        },
        cancel: function() {
            console.info('对话框已取消.');
        },
    })
}
```

上述代码的显示效果如图 5-43 所示。

2）通过 showToast 方法弹出 Toast 对话框

showToast 方法更加简单，只需传入 Toast 对象。Toast 对象包括 message 和 duration 两个属性：前者用于定义 Toast 对话框显示的字符串，后者用于定义 Toast 对话框显示的时间，单位为 ms。通过 showToast 方法弹出 Toast 对话框的典型代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/dialogtest/dialogtest.js
showToast() {
    prompt.showToast({
        message: 'Toast 信息',
        duration: 2000
    });
}
```

上述代码的显示效果如图 5-44 所示。

图 5-43　通过 showDialog 方法弹出一般对话框

图 5-44　通过 showToast 方法弹出 Toast 对话框

5.3　其他高级用法

本章开始部分的内容介绍了 JavaScript UI 的基本用法。虽然 JavaScript UI 冠以 UI 结尾，但是其功能和用法远不止如此，通过 JavaScript UI 可以管理资源、访问硬件传感器等功能，完全可以仅依靠 JavaScript UI 构建一个鸿蒙应用程序。本节介绍一些 JavaScript UI 的高级用法。虽然本节的标题冠以“高级”这个形容词，但是这些方法对于构建 JavaScript 鸿蒙应用程序来讲仍然非常重要。


5min

5.3.1　逻辑控制

在 5.1.3 节中介绍了 HML 动态绑定的概念，可以将 JavaScript 中的数据绑定到界面中，这是一个非常实用的功能。开发者直接修改 JavaScript 中的变量就可以实时将其更新到界面中。

注意：JavaScript UI 从设计上运用了 MVVM 模式，这种动态绑定的实现实际上是借助于 ViewModel 实现的。如果读者感兴趣可以参考鸿蒙 SDK 中的 viewmodel.d.ts 源代码，该文件处于鸿蒙 SDK 目录下的./js/<版本号>/api/common/@internal 目录（需要将“<版本号>”替换为开发者实际使用的 JavaScript SDK 版本号）。

借助动态绑定功能和逻辑控制，开发者可以实现更加复杂的组件动态展示能力。

1. 循环控制：for

通过 for 循环控制可以根据数组的长度创建对应属性的组件，而 for 循环控制是通过组件的 for 属性实现的。

为了演示循环控制的使用方法，首先创建一个 students 数组，students 数组中的每一项都包含 name 属性（代表姓名）和 age 属性（代表年龄），代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/logic/logic.js
export default {
    data: {
        students: [
            {name:'张三', age:17},
            {name:'李四', age:18},
            {name:'王五', age:16},
        ],
    }
}
```

在 HML 文件中，通过< text >组件的 for 属性动态绑定 students 数组，此时即可在其文本内容中通过 $ item 引用数组中的元素，通过 $ idx 获取当前元素的索引，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/logic/logic.hml
<div class = "container">
    <text for = "{{students}}">
        {{ $ idx + 1}} {{ $ item.name}} {{ $ item.age}}
    </text>
</div>
```

上述代码的显示效果如图 5-45 所示。

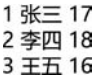


图 5-45 通过 for 循环控制显示数组内容

可见，通过 for 循环控制方法创建了 3 个< text >文本组件，并且每个组件都显示了数组元素的信息。

通过 in 关键词可以自定义数组元素变量名称，例如将 student 代替 $ item，代码如下：

```
<text for = "{{student in students}}">
    {{ $ idx + 1}} {{student.name}} {{student.age}}
</text>
```

除了可以自定义元素变量名称，还可以自定义索引变量名称，例如将 index 代替 $ idx，将 student 代替 $ item，代码如下：

```
<text for = "{{(index, student) in students}}">
    {{index + 1}} {{student.name}} {{student.age}}
</text>
```

在 for 属性和 if 属性(随后将介绍)中,动态绑定的符号“{{}}”可以省略,代码如下:

```
<text for = "(index, student) in students">
    {{index + 1}} {{student.name}} {{student.age}}
</text>
```

以上 4 段代码是等价的,显示效果相同。

2. 条件控制:if

通过 if 条件控制可以控制组件的显示与否。为了演示条件控制功能,首先在 JS 文件中创建 2 个变量:isBoy 和 isOldman,代码如下:

```
data: {
    isBoy: false,
    isOldman: false
}
```

随后,即可通过 if、elif 和 else 对组件的显示与否进行控制,代码如下:

```
<text if = "{{isBoy}}">你好,男孩!</text>
<text elif = "{{isOldman}}">您好,尊敬的前辈!</text>
<text else>你好,年轻人!</text>
```

由于 isBoy 和 isOldman 变量均为 false,因此最终将会在界面中显示“你好,年轻人!”字符串。当 isBoy 为 true 时,界面会显示“你好,男孩!”字符串。当 isBoy 为 false 且 isOldman 为 true 时,界面会显示“您好,尊敬的前辈!”字符串。

在连用 if、elif 和 else 时,组件的类型必须为兄弟节点,否则无法编译通过。

注意:通过 if 属性控制组件的显示与否,当判断结果为 false 时,该组件不仅不会显示在界面中,也无法通过 JS 方法获取其 DOM 元素,但是通过 show 属性控制组件的显示与否,当判断结果为 false 时,虽然其显示效果与前者相同,但是其 DOM 元素实则会被创建。

3. 逻辑控制块

逻辑控制块<block>是仅支持 for 和 if 属性的虚拟组件。<block>不会显示在界面中,仅仅用于逻辑控制。例如,可以通过<block>控制列表项的显示内容,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/logic/logic.hml
<list>
    <block for = "{{students}}">
```

```
        <list-item class="item">
            <text>{{ $item.name}}.</text>
            <text if="{{ $item.age >= 18}}">已成年</text>
            <text else>未成年</text>
        </list-item>
    </block>
</list>
```

上述代码的显示效果如图 5-46 所示。

图 5-46 通过逻辑控制块控制列表项的显示内容

5.3.2 代码资源

资源文件存储在 JavaScript 实例中的 common 目录中，可以包含图片、视频、代码文件等。在 5.2.2 节中介绍<image>和<video>组件时，已经介绍并使用了 common 目录中的图片和视频文件。本节介绍代码文件资源的使用方法。

代码文件资源可以为 js 文件、HML 文件和 CSS 文件。

1. js 文件资源

通过 export default 可以定义一个 js 模块。在 common 目录中创建一个名为 utils.js 的模块，并添加一个名为 getUserInformation()的方法，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/common/utils.js
export default {
    getUserInformation() {
        return {
            userid: "dongyu",
            username: "董昱",
            age: 18
        };
    }
}
```

然后，就可以在其他的 js 文件中导入这个模块，并可调用这个 getUserInformation()的方法，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/userinformation/userinformation.js
import utils from '../../common/utils.js'; //导入 utils 模块

export default {
    data: {
        userinformation: null
    },
    onInit() {
        //通过 getUserInformation()获取用户信息放置到 userinformation 变量中
        this.userinformation = utils.getUserInformation();
    }
}
```

2. HML 文件资源

在 common 目录中创建一个名为 information.hml 的文件，并添加一个< text >组件，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/common/information.hml
< text >年龄: {{age}}</text >
```

然后，在页面的 HML 文件中即可应用该资源，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/userinformation/userinformation.hml
< element name = "comp" src = "../../common/information.hml"></element >
< div class = "container">
    < text class = "title">用户 ID: {{userinformation.userid}}</text >
    < text class = "title">用户名称: {{userinformation.username}}</text >
    < comp class = "title" age = "{{userinformation.age}}"></comp >
</div >
```

在上面的代码中，首先通过< element >元素引用了 information.hml 文件，并将其元素名称定义为< comp >，然后，在页面中使用< comp >元素，并通过 age 属性填充了刚刚在 information.hml 中定义的 age 动态绑定变量。

3. CSS 文件资源

CSS 文件资源通过@import 语句进行引用。首先，在 common 目录中创建 customstyle.css 文件资源，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/common/customstyle.css
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
.title {
```

```
    font-size: 30px;
    text-align: center;
    width: 100%;
    height: 100px;
}
```

然后，即可在页面的 CSS 文件中引用这个资源文件，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/userinformation/userinformation.css
@import "../../common/customstyle.css";
```

在上面的例子中，同时在页面中引用了 JS 文件资源、HML 文件资源和 CSS 文件资源，其最终显示效果如图 5-47 所示。

用户ID: dongyu

用户名称： 董昱

年龄： 18

图 5-47 用户信息展示界面

5.3.3 设备适配

5min

鸿蒙操作系统是全场景分布式操作系统，其目标设备众多，因此通常要根据屏幕密度适配图片，以及通过媒体查询并根据屏幕的类型和尺寸适配样式文件。

1. 根据屏幕密度适配图片

DPI 分为 ldpi(低密度)、mdpi(中密度)、hdpi(高密度)、xhdpi(超高密度)、xxhdpi(超超高密度)和 xxxhdpi(超超超高密度)共 6 个等级，其中各分级所定义的 DPI 范围如图 3-29 所示。

在 JavaScript UI 中，根据屏幕密度适配图片的方法如下：

(1) 准备不同屏幕密度所需要的图片文件，例如 ic_test_xhdpi.png、ic_test_xxhdpi.png、ic_test_xxxhdpi.png 等。将这些文件放置到 JavaScript 实例的 common 目录下。

(2) 根据不同屏幕密度创建资源文件。在 JavaScript 实例 resources 目录下创建针对不同屏幕密度(DPI)的资源文件。资源文件为 JSON 文件，此类文件以 res-开头，后接 dpi 分级或者 defaults。

注意：如果系统根据当前的屏幕密度找不到所对应的资源，则会先查找并使用 res-defaults.json 中的图片资源。如果在 res-defaults.json 文件中仍然找不到相应的图片资源，则会选择最邻近 DPI 分级的资源。

为了简单起见，这里只在 resources 目录中创建了 res-defaults.json、res-xxhdpi.json 和 res-xxxhdpi.json 文件。此时，如果运行该程序的设备屏幕密度为 xxxhdpi，则会引用 res-xxxhdpi.json 文件的图片资源；如果运行该程序的设备屏幕密度为 xxhdpi，则会引用 res-xxhdpi.json 文件的图片资源；其他屏幕密度的设备则会直接使用 res-defaults.json 中的图片资源。

res-defaults.json 的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/resources/res-defaults.json
{
  "image": {
    "test": "common/ic_test_xhdpi.png"
  }
}
```

res-xxhdpi.json 的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/resources/res-xxhdpi.json
{
  "image": {
    "test": "common/ic_test_xxhdpi.png"
  }
}
```

res-xxxhdpi.json 的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/resources/res-xxxhdpi.json
{
  "image": {
    "test": "common/ic_test_xxxhdpi.png"
  }
}
```

在上面的 3 个文件中均定义了名为 test 的图片资源，并且根据屏幕密度的不同选择了 ic_test_xhdpi.png、ic_test_xxhdpi.png 和 ic_test_xxxhdpi.png 图片文件。

随后，在页面的 JS 文件和 HML 文件中即可通过 $r 方法获取适合当前屏幕密度的图片文件，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/adaption/adaption.hml
<div class="container">
    <image src="{{ $r('image.test') }}"/>
</div>
```

此时，<image>组件显示的图片内容会根据上述资源文件的定义选择合适的图片文件。

2. 媒体查询

媒体查询是 CSS 中的概念，即根据设备类型和屏幕参数有针对性地定义页面样式。媒体查询通过@media 语句实现。例如，通过@media 筛选手机设备的代码如下：

```
@media (device-type: phone) {
    /* 这里的 CSS 代码仅适配手机设备 */
    …
}
```

上述代码中，device-type 后接设备类型，可以为手机(phone)、智慧屏(tv)、可穿戴设备(wearable)等。

关于@media 语句的相关用法，读者可以参考 CSS 和鸿蒙 JS API 的相关文档。以下列举常见的用法：

(1) 根据屏幕的形状筛选设备。例如，筛选圆形屏幕设备的代码如下：

```
@media screen and (round-screen: true) { … }
```

(2) 根据屏幕的宽度筛选设备。例如，筛选页面宽度在 600 以内设备的代码如下：

```
@media (width <= 600) { … }
```

该代码为 CSS 4 语法，与下面的代码(CSS 3 语法)等价：

```
@media (max-width: 600) { … }
```

(3) 根据屏幕方向筛选设备。例如，筛选屏幕方向为横向且页面宽度大于 500 的代码如下：

```
@media screen and (orientation: landscape) and (width > 500) { … }
```

常用的媒体查询参数如表 5-7 所示。在多个媒体参数语句之间可以通过 and、or、not 等逻辑关系将其建立连接。

表 5-7 常用的媒体查询参数

类 型	说 明	类 型	说 明
height	页面高度	device-type	设备类型
min-height	页面最小高度	resolution	设备的分辨率
max-height	页面最大高度	min-resolution	设备的最小分辨率
width	页面宽度	max-resolution	设备的最大分辨率
min-width	页面最小宽度	device-height	设备的高度
max-width	页面最大宽度	min-device-height	设备的最小高度
aspect-ratio	页面宽高比	max-device-height	设备的最大高度
min-aspect-ratio	页面宽高比最小值	device-width	设备的宽度
max-aspect-ratio	页面宽高比最大值	min-device-width	设备的最小宽度
round-screen	屏幕是否为圆形	max-device-width	设备的最大宽度
orientation	屏幕方向，包括竖屏(portrait)和横屏(landscape)		

5.3.4 模块

模块是一系列 JavaScript 工具的集合，在 JavaScript 鸿蒙应用程序开发中，模块提供了 4min

各种各样的高级功能。严格上说，模块已经超出了 JavaScript UI 的范畴，而涉及网络访问、数据存储、设备管理等多种多样的业务功能。

常用的 JavaScript 模块如表 5-8 所示，其中前文已经介绍过页面路由(router)和页面弹窗(prompt)模块。因篇幅有限，本节无法完整地介绍这些模块，仅介绍一些常用的模块供读者参考。详细的模块使用方法可参考鸿蒙官方的 JS API 文档。

表 5-8　常用的 JavaScript 模块

模　块		模块名称	模块描述
应用	应用上下文	app	获取应用的名称、版本名称、版本号
	应用配置	configuration	获取应用当前的语言和地区
	应用管理	package	通过 BundleName 判断指定的应用是否已安装
	通知消息	notification	在状态栏上显示通知消息
	快捷方式	shortcut	创建某个页面的快捷方式
设备	设备信息	device	获取设备的品牌、生产商、型号、系统语言和地区、屏幕密度和形状等信息
	媒体查询	mediaquery	通过设备类型和屏幕尺寸查询匹配设备
	电量信息	battery	获取设备的电量信息
	屏幕亮度	brightness	获取和设置屏幕亮度和亮度模式、控制屏幕常亮等
	地理位置	geolocation	获取设备的地理位置信息
	传感器	sensor	获取加速度传感器、磁传感器、计步器、心率传感器等信息
	振动	vibrator	使设备振动
页面	页面路由	router	页面跳转和页面关系
	页面弹窗	prompt	弹出 Toast 对话框或一般对话框
网络	网络状态	network	获取当前的网络状态
	上传下载	request	实现数据的上传和下载
	HTTP 访问	fetch	实现 HTTP 访问
存储	数据存储	storage	通过键值对的方式存储数据
	文件存储	file	存储文件

使用 JavaScript 模块需要注意以下两个方面：

(1) 注意申请应用权限。使用某些模块(如震动、地理位置等)需要首先申请应用权限。如果开发者没有在 config.json 中添加应用权限，则这些模块的相关方法可能无法正常调用。

(2) 使用模块前需要导入模块。在使用模块的页面中，需要在 export default 语句块前导入模块，基本的导入方法代码如下：

```
import <模块名称> from '@system.<模块名称>';
```

导入具体的模块时，需要将“<模块名称>”替换为具体的模块名称。

1. 应用上下文与应用配置

通过应用上下文模块(app)和应用配置(configuration)模块可以获得应用的一些基本信息。在使用这两个模块前,需要导入 app 和 configuration 模块,代码如下:

```
import app from '@system.app';
import configuration from '@system.configuration';
```

然后,通过 app 和 configuration 中的方法即可获得应用名称、版本信息、区域语言信息等,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
console.info("应用名称: " + app.getInfo().appName);
console.info("版本号: " + app.getInfo().versionCode);
console.info("版本名称: " + app.getInfo().versionName);
console.info("区域: " + configuration.getLocale().countryOrRegion);
console.info("语言: " + configuration.getLocale().language);
console.info("阅读方向: " + configuration.getLocale().dir);
```

在 JavaScript UI 示例工程中,运行上述代码后在 HiLog 的输出信息如下:

```
27601-28040/* I 03B00/Console: app Log: 应用名称: JS UI
27601-28040/* I 03B00/Console: app Log: 版本号: 1
27601-28040/* I 03B00/Console: app Log: 版本名称: 1.0
27601-28040/* I 03B00/Console: app Log: 区域: CN
27601-28040/* I 03B00/Console: app Log: 语言: zh
27601-28040/* I 03B00/Console: app Log: 阅读方向: ltr
```

另外,通过应用上下文 app 还可以实现退出应用、请求全屏等功能。退出应用的代码如下:

```
app.terminate();
```

上述这种方法称为同步方法,方法结果会被直接返回。router 模块(在 5.1.4 节中已介绍)的 push、back 等方法,以及 prompt 模块(在 5.2.4 节中已介绍)的 showToast 等方法也均属于同步方法。这种方法在使用上非常简单直观,但是通常仅用于耗时较短的算法和信息获取功能。

注意:prompt 模块的 showDialog 方法属于异步方法。

2. 通知消息

通过通知消息(notification)模块可以在鸿蒙操作系统的通知栏中显示通知信息,具体的使用方法如下:

(1) 导入通知消息模块,代码如下:

```
import notification from '@system.notification';
```

(2) 通过 notification 模块的 show 方法即可显示通知消息,该方法需要传入一个 option 对象,包括以下参数:

- contentText:通知文本内容。
- contentTitle:通知标题。
- clickAction:单击通知后进入的页面,包括 bundleName、abilityName 和 uri 参数。bundleName 指定打开应用的 Bundle 名称;abilityName 指定 Ability 的全路径名称;uri 指定打开该 Ability 下的页面路径,当该路径为"/"时表示打开其主页面。

例如,在 JavaScript UI 应用中,通过 notification 模块创建一个通知栏信息,单击该通知后打开 5.2.4 节所介绍的对话框测试页面,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
notification.show({
    contentText: "单击进入对话框测试页面",
    contentTitle: "JavaScript UI 给您的通知",
    clickAction: {
        bundleName: "com.example.JavaScriptui",
        abilityName: "com.example.JavaScriptui.MainAbility",
        uri: "pages/dialogtest/dialogtest"
    }
})
```

运行上述代码,在通知栏中提示的信息如图 5-48 所示。

单击该通知后即可进入 5.2.4 节中所介绍的 dialogtest 对话框测试页面。

图 5-48 通过 notification 模块创建通知

3. 设备信息与异步方法的使用

通过设备信息(device)模块的 getInfo 方法即可获得设备品牌、生产商、型号、系统语言和地区、屏幕密度和形状等信息。

device 的 getInfo 方法是一个异步方法。这种方法并不返回任何信息。设备的具体信息是通过其回调方法实现的。在各种 JavaScript 模块中,异步方法通常包括 3 个主要的回调方法:

(1) success(data):信息获取成功后回调该方法,其中 data 参数包含了需要获取的信息。

(2) fail(data,code):信息获取失败后回调该方法,其中 data 参数为错误信息,code 参数为错误代码。参数代码可以为 200(通用错误)、202(参数错误)或 300(I/O 错误)等。如果调用异步方法时缺少了应用权限,则其错误参数代码为 200。

(3) cancel()：当用户主动取消了这个异步方法的执行时将会回调到该方法中。例如，prompt 模块的 showDialog()方法运用了这一回调。不过，在大多数模块中 cancel()回调并不常用。

(4) complete()：该方法执行完成后回调该方法。

注意：success()、fail()和 cancel()方法互斥，一次方法调用只能回调到这 3 种方法中的 1 种方法中。最后，将会调用 complete()方法。

在使用 device 模块前需要导入该模块，代码如下：

```
import device from '@system.device';
```

在 device 模块 getInfo()方法的 success(data)回调方法中，data 包含 brand、manufacturer、model 等参数，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
device.getInfo({
    success: function(data) {
        console.info('设备品牌：' + data.brand);
        console.info('设备生产商：' + data.manufacturer);
        console.info('设备型号：' + data.model);
        console.info('设备代号：' + data.product);
        console.info('系统语言：' + data.language);
        console.info('系统地区：' + data.region);
        console.info('可使用的窗口宽度：' + data.windowWidth);
        console.info('可使用的窗口高度：' + data.windowHeight);
        console.info('屏幕密度(dpi)：' + data.screenDensity);
        console.info('屏幕形状：' + data.screenShape);

    },
    fail: function(data, code) {
        console.info('设备信息获取错误。错误代码：'+ code + '错误信息：' + data);
    },
    complete: function(){
        console.info("设备信息获取完毕");
    }

});
```

在华为 P40 手机的应用程序中，运行上述代码后在 HiLog 的输出信息如下：

```
30979 - 31227/* I 03B00/Console: app Log: 设备品牌：HUAWEI
30979 - 31227/* I 03B00/Console: app Log: 设备生产商：HUAWEI
30979 - 31227/* I 03B00/Console: app Log: 设备型号：ANA - AN00
30979 - 31227/* I 03B00/Console: app Log: 设备代号：ANA - AN00
```

```
30979 - 31227/* I 03B00/Console: app Log: 系统语言: zh
30979 - 31227/* I 03B00/Console: app Log: 系统地区: CN
30979 - 31227/* I 03B00/Console: app Log: 可使用的窗口宽度: 1080
30979 - 31227/* I 03B00/Console: app Log: 可使用的窗口高度: 2043
30979 - 31227/* I 03B00/Console: app Log: 屏幕密度(dpi): 3.000000
30979 - 31227/* I 03B00/Console: app Log: 屏幕形状: rect
30979 - 31227/* I 03B00/Console: app Log: 设备信息获取完毕
```

其中,屏幕形状参数 screenShape 的取值可以为 rect(矩形屏)或 circle(圆形屏)。

4. 检查应用是否安装

通过应用管理 package 模块可以检查应用程序是否已安装,在使用该模块之前需要申请 GET_BUNDLE_INFO 权限,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/config.json
{
  ...
  "module": {
    ...
    "reqPermissions": [
      {
        "name": "ohos.permission.GET_BUNDLE_INFO"
      },
      ...
    ],
    ...
  }
}
```

然后,在使用该模块的页面中导入该模块,代码如下:

```
import pkg from '@system.package';
```

最后,通过 package 的 hasInstalled 方法即可判断指定 Bundle 名称的应用程序是否已安装,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
pkg.hasInstalled({
    bundleName: 'com.example.javaui',
    success: function(data) {
        console.info('Java UI 应用程序安装情况: ' + data);
    },
    fail: function(data, code) {
        console.info('安装信息获取错误。错误代码: '+ code + '错误信息: ' + data);
    },
    complete: function(){
```

```
            console.info("安装信息获取完毕");
        }
    });
```

其中，bundleName 参数即为需要检查的应用 Bundle 的名称。如果该应用程序已经安装，则上述代码的 HiLog 输出结果如下：

```
31812 - 32494/ * I 03B00/Console: app Log: JavaUI 应用程序安装情况: true
31812 - 32494/ * I 03B00/Console: app Log: 安装信息获取完毕
```

5. 地理位置模块与订阅

通过地理位置 geolocation 模块可以获取设备当前的地理位置。在使用该模块之前需要申请 ohos.permission.LOCATION 权限，代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/config.json
{
  …
  "module": {
    …
    "reqPermissions": [
      {
        "name": "ohos.permission.LOCATION"
      },
      …
    ],
    …
  }
}
```

地理位置 geolocation 模块的常用方法如下：

(1) getLocation()：获取当前的地理位置信息。

(2) getLocationType()：获取当前地理位置信息的定位方式(网络定位或 GPS 定位)。

(3) subscribe()：订阅地理位置信息。

(4) unsubscribe()：取消订阅地理位置信息。

(5) getSupportedCoordTypes()：获取支持的坐标类型(WGS-84 或 GCJ-02)。

注意：GCJ-02 是一个由中国国家测绘局制订的用于中国范围内民用地图(包括电子地图)的加密后地理坐标系统，其中 GCJ 的 3 个字母分别为“国家”“测绘”和“局”的首字母简写。GCJ-02 的加密算法是非线性的，很难通过加密坐标来反推原始的正确坐标。目前，谷歌、百度等公司发布的电子地图基本都采用了 GCJ-02 坐标系统并对原始数据进行了加密。

在使用地理位置 geolocation 模块的页面中导入该模块，代码如下：

```
import geolocation from '@system.geolocation';
```

下面介绍获取、订阅地理位置信息的方法。

1）获取地理位置信息

与其他异步方法类似，获取地理位置信息同样需要 success()、fail()和 complete()方法，典型的代码如下：

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
geolocation.getLocation({
    success: function(data) {
        console.info('地理位置信息获取成功。经度:' + data.longitude + " 纬度: " + data.
latitude);
    },
    fail: function(data, code) {
        console.info('地理位置信息获取错误。错误代码: '+ code + '错误信息: ' + data);
    },
    complete: function(){
        console.info("地理位置信息获取完毕");
    }
});
```

在地理位置信息获取成功后，data 变量包含经度(longitude)、纬度(latitude)、高度(altitude)、精度(accuracy)、地理位置获取时间(time)等参数。

在运行上述代码时，系统会询问用户是否授权获取地理位置信息，如图 5-49 所示。

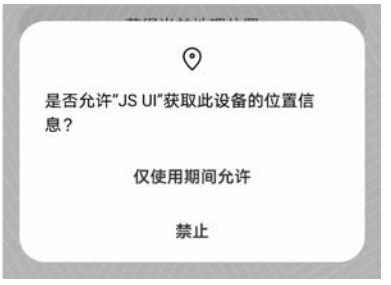


图 5-49　位置信息权限的动态申请

当用户同意授权后，地理位置获取成功后 HiLog 输出如下：

```
3895 - 4709/* I 03B00/Console: app Log: 地理位置信息获取成功。经度:** 纬度: **
3895 - 4709/* I 03B00/Console: app Log: 地理位置信息获取完毕
```

2）订阅地理位置信息

在运行应用时，用户可能要随时监听特定的信息，以便于应用或用户能够实时处理这些信息。例如，在出行导航时应用需要根据地理位置信息的变化为用户提供导航信息。类似的信息还包括电池电量、屏幕亮度及加速度计、磁传感器、心率传感器等各种各样的设备传感器信息。

这就需要使用 JavaScript 模块中的订阅功能。对于同一个需要订阅的信息,模块都会为其设置订阅方法(subscribe)和取消订阅方法(unsubscribe)。

订阅方法通常包括 success 回调和 fail 回调,这与模块的异步方法非常类似。例如,通过 geolocation 的 subscribe()方法即可订阅地理位置信息,代码如下:

```
//chapter5/JavaScriptUI/entry/src/main/js/default/pages/module/module.js
geolocation.subscribe({
    success: function(data) {
        console.info('地理位置信息更新成功。经度:' + data.longitude + " 维度: " + data.
latitude);
    },
    fail: function(data, code) {
        console.info('地理位置信息更新错误。错误代码: '+ code + '错误信息: ' + data);
    }
});
```

运行上述代码后,每次地理位置更新都会在 HiLog 出现相应的更新,典型的提示信息如下:

```
3895 - 4709/ * I 03B00/Console: app Log: 地理位置信息更新成功。经度: ** 纬度: **
```

取消订阅方法通常为同步方法。例如,通过 geolocation 的 unsubscribe()方法即可取消订阅地理位置信息,代码如下:

```
geolocation.unsubscribe();
```

6. 传感器的相关订阅方法

传感器模块 sensor 提供了各类传感器信息的订阅获取方法,读者可参考上文中对地理位置信息的方法来订阅这些传感器信息,这里不再赘述。传感器模块 sensor 的常用订阅方法如表 5-9 所示。

表 5-9 传感器模块 sensor 的常用订阅方法

方 法	描 述	方 法	描 述
subscribeAccelerometer()	订阅加速度计信息	subscribeLight()	订阅光线传感器信息
unsubscribeAccelerometer()	取消订阅加速度计信息	unsubscribeLight()	取消订阅光线传感器信息
subscribeCompass()	订阅磁传感器信息	subscribeStepCounter()	订阅计步器信息
unsubscribeCompass()	取消订阅磁传感器信息	unsubscribeStepCounter()	取消订阅计步器信息
		subscribeBarometer()	订阅气压计信息
subscribeProximity()	订阅距离传感器信息	unsubscribeBarometer()	取消订阅气压计信息
		subscribeHeartRate()	订阅心率传感器信息
unsubscribeProximity()	取消订阅距离传感器信息	unsubscribeHeartRate()	取消订阅心率传感器信息

注意：在开发订阅传感器的相关代码之前，需要确认相关传感器的应用权限的配置是否正确。

5.4 本章小结

恭喜你又获得了一项 UI 开发的技能！笔者认为 JavaScript UI 中各类组件和容器的默认样式更加美观。具备前端开发基础的开发者可能更加容易上手。

本章介绍了 JavaScript UI 中的各种基本概念、常见组件和容器的用法，以及控制逻辑、代码资源管理等高级功能，最后介绍了功能强大的各种各样的 JavaScript 模块。通过对模块的学习可以发现，JavaScript UI 不仅包含了 UI 设计的相关组件和容器，还包含了系统通知、设备管理等各种业务逻辑代码中所设计的功能。通过 JavaScript UI(不依赖 Java 语言)完全可以开发出独立的鸿蒙应用程序，因此，JavaScript UI 的称呼似乎并不准确，称其为 JavaScript API 更加名副其实。不过，对于复杂的业务逻辑，仍然需要 Java API 的加持才能完成。

本章作为较为独立的一章，难以完整地介绍 JavaScript UI 的各个方面，不过鸿蒙官方网站提供了中文的 JavaScript API 文档，相对于 Java API 来讲可能更加方便查阅和学习。希望读者能够举一反三，学有所获。