第5章 方舟开发框架(ArkUI)——基于JS扩展的类Web开发范式

方舟开发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,适用于手机、平板、智慧屏和智能穿戴应用开发。方舟开发框架包括基于JS扩展的类Web开发范式和TS扩展的声明式开发范式。本章对基于JS扩展的类Web开发范式进行描述。

5.1 开发概述

基于JS扩展的类Web开发基础功能如下。

1. 类Web范式编程

采用类HTML和CSS Web编程语言作为页面布局和页面样式的开发语言,页面业务逻辑则支持ECMAScript规范的JavaScript语言。方舟开发框架提供的类Web编程范式,可以让开发者避免编写UI状态切换的代码,视图配置信息更加直观。

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,降低多设备适配成本。

3. 高性能

开发框架包含许多核心的控件,例如列表、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。

使用基于JS扩展的类Web开发范式的方舟开发框架,包括应用层(Application)、前端框架层(Framework)、引擎层(Engine)和平台适配层(Porting Layer),如图5-1所示。

(1) 应用层。应用层表示开发的FA应用,这里的FA特指JS FA应用。

(2) 前端框架层。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功能。

(3) 引擎层。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、事件管理等功能。

(4) 平台适配层。平台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

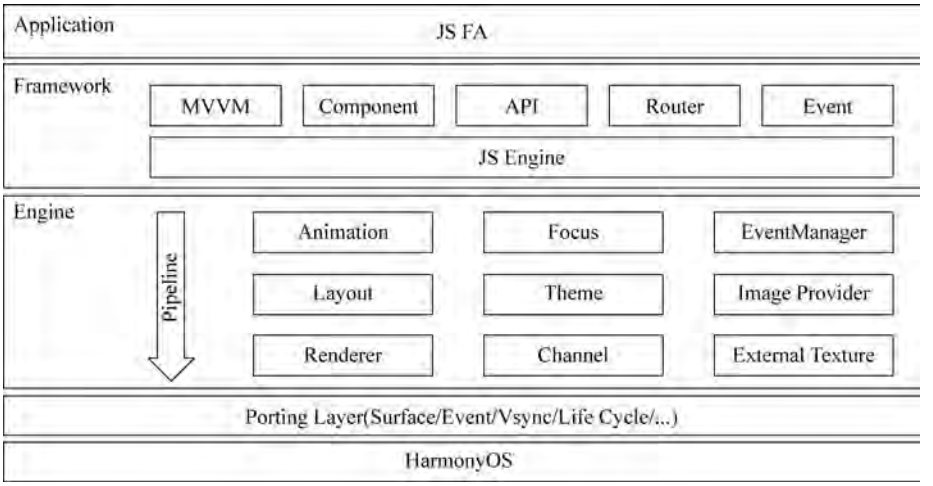

图 5-1　方舟开发框架

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。

5.2　JS FA 初步应用

基于 JS 扩展的类 Web 开发范式支持纯 JavaScript、JavaScript 和 Java 混合语言开发。JS FA 是基于 JavaScript、JavaScript 与 Java 混合开发的 FA。

5.2.1　JS FA 概述

JS FA 在 HarmonyOS 上运行时，需要基类 AceAbility、加载 JS FA 主体的方法、JS FA 开发目录，具体说明如下。

1. AceAbility

AceAbility 类是 JS FA 在 HarmonyOS 上运行环境的基类，继承自 Ability。开发者的应用运行入口类应该从该类派生，相关代码如下。

```
public class MainAbility extends AceAbility {
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
    }
    @Override
    public void onStop() {
        super.onStop();
    }
}
```

2. 加载 JS FA

JS FA 生命周期事件分为应用生命周期和页面生命周期，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。

setInstanceName(String name)的参数 name 是指实例名称，实例名称与 config.json 文件中 module.js.name 的值对应。若开发者未修改实例名称，而使用了默认值 default，则无须调用此接口。若已经修改，则需在应用 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 module.js 字段中有多个实例项，使用时选择相应的实例名称。

setInstanceName()接口使用方法：在 MainAbility 的 onStart()中的 super.onStart()前调用此接口。以 JSComponentName 作为实例名称，需在 super.onStart(Intent)前调用此接口，相关代码如下。

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

3. JS FA 开发目录

新建工程的 JS 目录如图 5-2 所示。

图 5-2　新建工程的 JS 目录

在工程目录中，i18n 下存放多语言的 json 文件；pages 文件夹下存放多个页面，每个页面由 HML、CSS 和 JS 文件组成。HML 是一套类 HTML 的标记语言，通过组件、事件构建出页面的内容。页面具备数据绑定、事件绑定、列表渲染、条件渲染等高级能力。

(1) main→js→default→i18n→en-US.json：此文件定义了在英文模式下页面显示的变量内容。

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

同理，zh-CN.json 中定义了中文模式下的页面内容。

(2) main→js→default→pages→index→index.hml：此文件定义了 index 页面的布局、

用到的组件，以及这些组件的层级关系。例如，index.hml 文件中包含了一个 text 组件，内容为 Hello World 文本。

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

(3) main→js→default→pages→index→index.css：此文件定义了 index 页面的样式(index.css 文件定义了 container 和 title)。

```
.container {
  flex - direction: column;
  justify - content: center;
  align - items: center;
}
.title {
  font - size: 100px;
}
```

(4) main→js→default→pages→index→index.js：此文件定义了 index 页面的业务逻辑(数据绑定、事件处理等)。示例：变量 title 赋值为字符串 World。

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

5.2.2 JS FA 开发应用

本节主要介绍 JS FA 开发应用。该应用通过 media query 同时适配手机和 TV，单击或者将焦点移动到食物的缩略图选择不同的食物图片，也可以添加到购物车，如图 5-3 和图 5-4 所示。

1. 构建页面布局

在 index.hml 文件中构建页面布局，进行代码开发之前，首先要对页面布局进行分析，将页面分解为不同区，用容器组件承载。根据 JS FA 应用效果图，此页面共分成三部分：标题区、展示区和详情区。展示区和详情区在手机和 TV 上分别是按列排列和按行排列。

标题区较为简单，由两个按列排列的 text 组件构成。展示区包含 4 个 image 组件的 swiper，详情区由 image 和 text 组件构成。下面以手机效果图为例，展示区和详情区布局如图 5-5 所示。

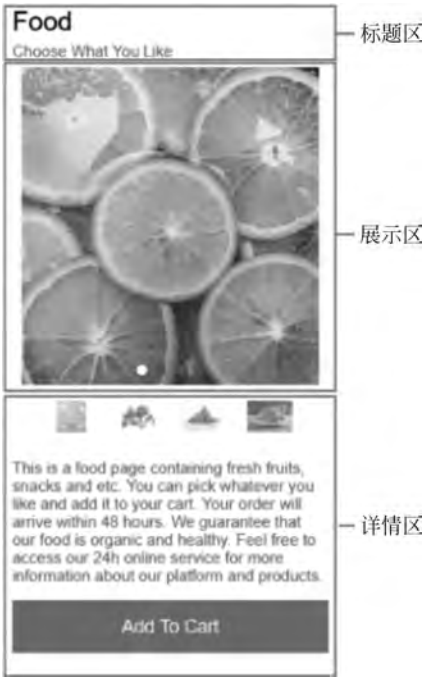

图 5-3　手机应用效果

图 5-4　TV 应用效果

根据布局结构的分析,实现页面基础布局的代码如下(其中 4 个 image 组件通过 for 指令循环创建)。

```
<!-- index.hml -->
<div class = "container">
  <!-- title area -->
  <div class = "title">
    <text class = "name"> Food </text>
```

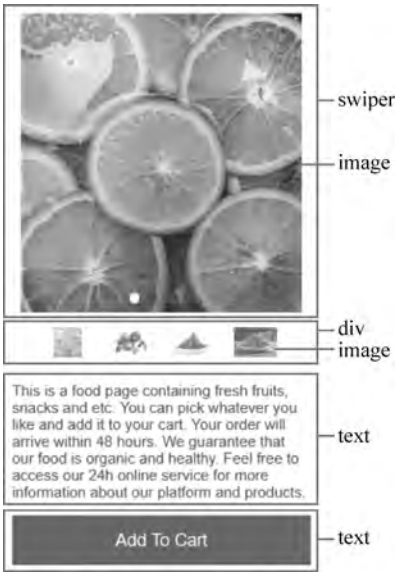

图 5-5　展示区和详情区布局

```
    <text class="sub-title">Choose What You Like</text>
  </div>
  <div class="display-style">
    <!-- display area -->
    <swiper id="swiperImage" class="swiper-style">
      <image src="{{ $item}}" class="image-mode" focusable="true" for="{{imageList}}">
</image>
    </swiper>
    <!-- product details area -->
    <div class="container">
      <div class="selection-bar-container">
        <div class="selection-bar">
          <image src="{{ $item}}" class="option-mode" onfocus="swipeToIndex({{ $idx}})"
 onclick="swipeToIndex({{ $idx}})" for="{{imageList}}"></image>
        </div>
      </div>
      <div class="description-first-paragraph">
        <text class="description">{{descriptionFirstParagraph}}</text>
      </div>
      <div class="cart">
        <text class="{{cartStyle}}" onclick="addCart" onfocus="getFocus" onblur=
"lostFocus" focusable="true">{{cartText}}</text>
      </div>
    </div>
  </div>
</div>
```

swiper组件里展示的图片需要自行添加图片资源，放置到js→default→common目录下，common目录需自行创建。

2. 构建页面样式

index.css文件通过media query管控手机和TV不同页面样式。此外，该页面样式还采用了CSS伪类的写法，当单击或者焦点移动到image组件上时，image组件由半透明变成不透明，以此实现选中的效果，相关代码请扫描二维码获取。

3. 构建页面逻辑

在index.js文件中构建页面逻辑，主要实现两个功能：单击或者焦点移动到不同的缩略图时，swiper滑动到相应的图片；焦点移动到购物车时，Add To Cart背景颜色从浅蓝色变成深蓝色，单击后文字变为Cart+1，背景颜色由深蓝色变成黄色。添加购物车不可重复操作，逻辑页面相关代码如下。

```
//index.js
export default {
  data: {
    cartText: 'Add To Cart',
    cartStyle: 'cart-text',
    isCartEmpty: true,
    descriptionFirstParagraph: 'This is a food page containing fresh fruits, snacks and etc.
You can pick whatever you like and add it to your cart. Your order will arrive within 48 hours. We
guarantee that our food is organic and healthy. Feel free to access our 24h online service for
more information about our platform and products.',
    imageList: ['/common/food_000.JPG', '/common/food_001.JPG', '/common/food_002.JPG',
'/common/food_003.JPG'],
  },
  swipeToIndex(index) {
    this.$element('swiperImage').swipeTo({index: index});
  },
  addCart() {
    if (this.isCartEmpty) {
      this.cartText = 'Cart + 1';
      this.cartStyle = 'add-cart-text';
      this.isCartEmpty = false;
    }
  },
  getFocus() {
    if (this.isCartEmpty) {
      this.cartStyle = 'cart-text-focus';
    }
  },
  lostFocus() {
    if (this.isCartEmpty) {
      this.cartStyle = 'cart-text';
    }
  },
}
```

4. 配置设备类型

在 config.json 的 deviceType 字段中添加手机和 TV 的设备类型。

```
{
  ...
  "module": {
    ...
    "deviceType": [
      "phone",
      "tv"
    ],
    ...
  }
}
```

5.3 构建用户界面

本节主要介绍组件、构建布局、添加交互、动画、事件、页面路由和焦点逻辑。

5.3.1 组件

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，既可以在需求相同的地方重复使用，也可以通过组件间合理的搭配定义满足业务需求的新组件，减少开发量，实现自定义开发的组件。组件根据功能，可以分为以下几类，如表 5-1 所示。

表 5-1 组件类型及组件名称

组件类型	组件名称
容器组件	badge、dialog、div、form、list、list-item、list-item-group、panel、popup、refresh、stack、stepper、stepper-item、swiper、tabs、tab-bar、tab-content
基础组件	button、chart、divider、image、image-animator、input、label、marquee、menu、option、picker、picker-view、piece、progress、qrcode、rating、richtext、search、select、slider、span、switch、text、textarea、toolbar、toolbar-item、toggle、web
媒体组件	camera、video
画布组件	canvas
栅格组件	grid-container、grid-row、grid-col
svg 组件	svg、rect、circle、ellipse、path、line、polyline、polygon、text、tspan、textPath、animate、animateMotion、animateTransform

5.3.2 构建布局

本节主要对布局说明、添加标题行和文本区域、添加图片区域、添加留言区域、添加容器

进行描述。

1. 布局说明

手机和智慧屏的基准宽度为720px(px为逻辑像素,非物理像素),实际显示效果会根据屏幕宽度进行缩放,其换算关系如下。

组件的width设为100px时,在宽度为720物理像素的屏幕上,实际显示为100物理像素;在宽度为1440物理像素的屏幕上,实际显示为200物理像素。智能穿戴的基准宽度为454px,换算逻辑同理。

一个页面的基本元素包含标题区域、文本区域和图片区域等,每个基本元素内还包含多个子元素,根据需求可以添加按钮、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:元素的尺寸和排列位置、是否有重叠的元素、是否需要设置对齐、内间距或者边界、是否包含子元素及其排列位置、是否需要容器组件及其类型。

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,方便对页面做后续调整,如图5-6和图5-7所示。

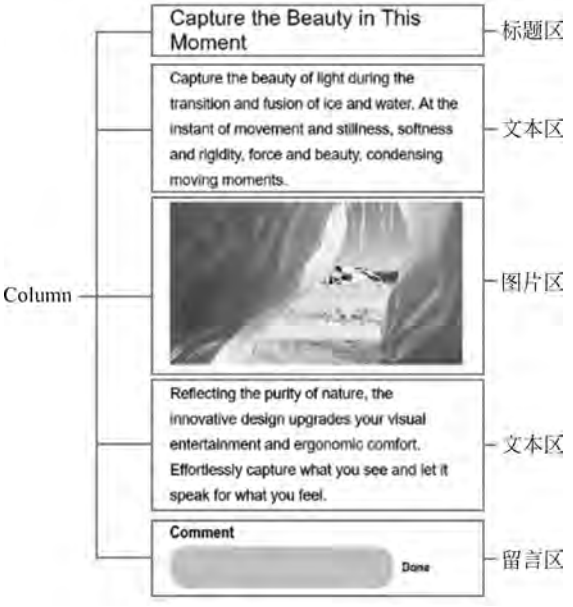

图5-6　页面布局分解

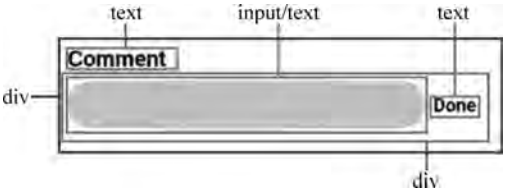

图5-7　留言区布局分解

2. 添加标题行和文本区域

实现标题和文本区域常用的是基础组件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,插入标题和文本区域的相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <text class="title-text">{{headTitle}}</text>
```

```
    <text class = "paragraph - text">{{paragraphFirst}}</text>
    <text class = "paragraph - text">{{paragraphSecond}}</text>
  </div>
  /* xxx.css */
  .container {
    flex - direction: column;
    margin - top: 20px;
    margin - left: 30px;
  }
  .title - text {
    color: #1a1a1a;
    font - size: 50px;
    margin - top: 40px;
    margin - bottom: 20px;
  }
  .paragraph - text {
    color: #000000;
    font - size: 35px;
    line - height: 60px;
  }
  //xxx.js
  export default {
    data: {
      headTitle: 'Capture the Beauty in This Moment',
      paragraphFirst: 'Capture the beauty of light during the transition and fusion of ice and
    water. At the instant of movement and stillness, softness and rigidity, force and beauty,
    condensing moving moments.',
      paragraphSecond: 'Reflecting the purity of nature, the innovative design upgrades your
    visual entertainment and ergonomic comfort. Effortlessly capture what you see and let it speak
    for what you feel.',
    },
  }
```

3. 添加图片区域

添加图片区域通常用 image 组件实现，使用方法与 text 组件类似。图片资源建议放在 js→default→common 目录下，common 目录需自行创建，相关代码如下。

```
<!-- xxx.hml -->
<image class = "img" src = "{{middleImage}}"></image>
/* xxx.css */
.img {
  margin - top: 30px;
  margin - bottom: 30px;
  height: 385px;
}
//xxx.js
export default {
```

```
  data: {
    middleImage: '/common/ice.png',
  },
}
```

4. 添加留言区域

用户输入留言后单击完成，留言区域即显示留言内容。用户单击右侧的“删除”按钮可删除当前留言内容并重新输入，留言区域由 div、text、input 关联 click 事件实现。开发者可以使用 input 组件实现输入留言，通过 text 组件实现留言完成，使用 commentText 的状态标记此时显示的组件(通过 if 属性控制)。在包含文本“完成”和“删除”的 text 组件中关联 click 事件，更新 commentText 状态和 inputValue 的内容，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <text class="comment-title">Comment</text>
  <div if="{{!commentText}}">
    <input class="comment" value="{{inputValue}}" onchange="updateValue()"></input>
    <text class="comment-key" onclick="update" focusable="true">Done</text>
  </div>
  <div if="{{commentText}}">
    <text class="comment-text" focusable="true">{{inputValue}}</text>
    <text class="comment-key" onclick="update" focusable="true">Delete</text>
  </div>
</div>
/* xxx.css */
.container {
  margin-top: 24px;
  background-color: #ffffff;
}
.comment-title {
  font-size: 40px;
  color: #1a1a1a;
  font-weight: bold;
  margin-top: 40px;
  margin-bottom: 10px;
}
.comment {
  width: 550px;
  height: 100px;
  background-color: lightgrey;
}
.comment-key {
  width: 150px;
  height: 100px;
  margin-left: 20px;
  font-size: 32px;
```

```
    color: #1a1a1a;
    font-weight: bold;
  }
  .comment-key:focus {
    color: #007dff;
  }
  .comment-text {
    width: 550px;
    height: 100px;
    text-align: left;
    line-height: 35px;
    font-size: 30px;
    color: #000000;
    border-bottom-color: #bcbcbc;
    border-bottom-width: 0.5px;
  }
  //xxx.js
  export default {
    data: {
      inputValue: '',
      commentText: false,
    },
    update() {
      this.commentText = !this.commentText;
    },
    updateValue(e) {
      this.inputValue = e.text;
    },
  }
```

5. 添加容器

将页面的基本元素组装在一起，需要使用容器组件。在页面布局中常用 div、list 和 tabs 3 种容器组件。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，可以支持多种子组件，使用起来更为方便。

1）list 组件

当页面结构较为复杂时，如果使用 div 循环渲染，容易出现卡顿，因此推荐使用 list 组件代替 div 组件实现长列表布局，从而实现更加流畅的列表滚动体验。list 仅支持 list-item 作为子组件，相关代码如下。

```
<!-- xxx.hml -->
<list class="list">
  <list-item type="listItem" for="{{textList}}">
    <text class="desc-text">{{ $item.value}}</text>
  </list-item>
</list>
```

```
/* xxx.css */
.desc-text {
  width: 683.3px;
  font-size: 35.4px;
}
//xxx.js
export default {
  data: {
    textList:  [{value: 'JS FA'}],
  },
}
```

为避免示例代码过长,以上示例的 list 中只包含一个 list-item 和一个 text 组件。在实际应用中可以在 list 中加入多个 list-item,同时 list-item 下可以包含多个其他子组件。

2) tabs 组件

当页面经常需要动态加载时,推荐使用 tabs 组件。tabs 组件支持 change 事件,在页签切换后触发。tabs 组件仅支持一个 tab-bar 和一个 tab-content,相关代码如下。

```
<!-- xxx.hml -->
<tabs>
  <tab-bar>
    <text>Home</text>
    <text>Index</text>
    <text>Detail</text>
  </tab-bar>
  <tab-content>
    <image src="{{homeImage}}"></image>
    <image src="{{indexImage}}"></image>
    <image src="{{detailImage}}"></image>
  </tab-content>
</tabs>
//xxx.js
export default {
  data: {
    homeImage: '/common/home.png',
    indexImage: '/common/index.png',
    detailImage: '/common/detail.png',
  },
}
```

tab-content 组件用来展示页签的内容区,支持 scrollable 属性,高度默认充满 tabs 剩余空间。

5.3.3　添加交互

添加交互可以通过在组件上实现关联事件。本节介绍如何用 div、text 和 image 组件关

联 click 事件，构建一个点赞按钮。点赞按钮通过一个 div 组件实现关联 click 事件。div 组件包含一个 image 和一个 text 组件。

image 组件用于显示未点赞和点赞的效果。click 事件的函数会交替更新点赞和未点赞图片的路径。text 组件用于显示点赞数，点赞数会在 click 事件的函数中同步更新。

click 事件作为一个函数定义在 JS 文件中，可以更改 isPressed 的状态，从而更新显示 image 组件。如果 isPressed 为真，则点赞数加 1。该函数在 HML 文件中对应的 div 组件上生效，点赞按钮各子组件的样式设置在 CSS 文件当中，相关代码如下。

```
<!-- xxx.hml -->
<!-- 点赞按钮 -->
<div>
  <div class="like" onclick="likeClick">
    <image class="like-img" src="{{likeImage}}" focusable="true"></image>
    <text class="like-num" focusable="true">{{total}}</text>
  </div>
</div>
/* xxx.css */
.like {
  width: 104px;
  height: 54px;
  border: 2px solid #bcbcbc;
  justify-content: space-between;
  align-items: center;
  margin-left: 72px;
  border-radius: 8px;
}
.like-img {
  width: 33px;
  height: 33px;
  margin-left: 14px;
}
.like-num {
  color: #bcbcbc;
  font-size: 20px;
  margin-right: 17px;
}
//xxx.js
export default {
  data: {
    likeImage: '/common/unLike.png',
    isPressed: false,
    total: 20,
  },
  likeClick() {
    var temp;
```

```
    if (!this.isPressed) {
      temp = this.total + 1;
      this.likeImage = '/common/like.png';
    } else {
      temp = this.total - 1;
      this.likeImage = '/common/unLike.png';
    }
    this.total = temp;
    this.isPressed = !this.isPressed;
  },
}
```

除此之外，还有很多表单组件(开关、标签、滑动选择器等)可在页面布局时灵活使用，提高交互性。

5.3.4　动画

动画分为静态动画和连续动画。

1. 静态动画

静态动画的核心是 transform 样式，可以实现以下 3 种变换类型，一次样式设置只能实现一种类型变换。

translate：沿水平或垂直方向将指定组件移动所需距离。scale：横向或纵向将指定组件缩小或放大到所需比例。rotate：将指定组件沿横轴、纵轴或中心点旋转指定的角度，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <text class = "translate"> hello </text>
  <text class = "rotate"> hello </text>
  <text class = "scale"> hello </text>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  align-items: center;
}
.translate {
  height: 150px;
  width: 300px;
  font-size: 50px;
  background-color: #008000;
  transform: translate(200px);
}
.rotate {
  height: 150px;
```

```
    width: 300px;
    font-size: 50px;
    background-color: #008000;
    transform-origin: 200px 100px;
    transform: rotateX(45deg);
  }
  .scale {
    height: 150px;
    width: 300px;
    font-size: 50px;
    background-color: #008000;
    transform: scaleX(1.5);
  }
```

2. 连续动画

静态动画只有开始状态和结束状态，没有中间状态。如果设置中间的过渡状态和转换效果，需要使用连续动画实现。

连续动画的核心是 animation 样式，它定义了动画的开始状态、结束状态及时间和速度的变化曲线，通过 animation 样式实现的效果如下。

animation-name：设置动画执行后应用到组件上的背景颜色、透明度、宽高和变换类型。

animation-delay 和 animation-duration：分别设置动画执行后元素延迟和持续的时间。

animation-timing-function：描述动画执行的速度曲线，使动画更加平滑。

animation-iteration-count：定义动画播放的次数。

animation-fill-mode：指定动画执行结束后是否恢复初始状态。

animation 样式需要在 CSS 文件中先定义 keyframe，然后设置动画的过渡效果，并通过一个样式类型在 HML 文件中调用，相关代码如下。

```
<!-- xxx.hml -->
<div class="item-container">
  <text class="header">animation-name</text>
  <div class="item {{colorParam}}">
    <text class="txt">color</text>
  </div>
  <div class="item {{opacityParam}}">
    <text class="txt">opacity</text>
  </div>
  <input class="button" type="button" name="" value="show" onclick="showAnimation"/>
</div>
/* xxx.css */
.item-container {
  margin-right: 60px;
  margin-left: 60px;
  flex-direction: column;
```

```
}
.header {
  margin-bottom: 20px;
}
.item {
  background-color: #f76160;
}
.txt {
  text-align: center;
  width: 200px;
  height: 100px;
}
.button {
  width: 200px;
  font-size: 30px;
  background-color: #09ba07;
}
.color {
  animation-name: Color;
  animation-duration: 8000ms;
}
.opacity {
  animation-name: Opacity;
  animation-duration: 8000ms;
}
@keyframes Color {
  from {
    background-color: #f76160;
  }
  to {
    background-color: #09ba07;
  }
}
@keyframes Opacity {
  from {
    opacity: 0.9;
  }
  to {
    opacity: 0.1;
  }
}
//xxx.js
export default {
  data: {
    colorParam: '',
    opacityParam: '',
  },
```

```
  showAnimation: function () {
    this.colorParam = '';
    this.opacityParam = '';
    this.colorParam = 'color';
    this.opacityParam = 'opacity';
  },
}
```

5.3.5 事件

事件主要为手势事件和按键事件。手势事件主要用于智能穿戴等具有触摸屏的设备，按键事件主要用于智慧屏设备。

1. 手势事件

手势表示由单个或多个事件识别的语义动作(触摸、单击和长按)。一个完整的手势可能由多个事件组成，对应手势的生命周期，支持的事件如下。

1) 触摸

touchstart：手指触摸动作开始。

touchmove：手指触摸后移动。

touchcancel：手指触摸动作被打断(来电提醒、弹窗)。

touchend：手指触摸动作结束。

2) 单击

click：用户快速轻敲屏幕。

3) 长按

longpress：用户在相同位置长时间保持与屏幕接触，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <div class="text-container" onclick="click">
    <text class="text-style">{{onClick}}</text>
  </div>
  <div class="text-container" ontouchstart="touchStart">
    <text class="text-style">{{touchstart}}</text>
  </div>
  <div class="text-container" ontouchmove="touchMove">
    <text class="text-style">{{touchmove}}</text>
  </div>
  <div class="text-container" ontouchend="touchEnd">
    <text class="text-style">{{touchend}}</text>
  </div>
  <div class="text-container" ontouchcancel="touchCancel">
    <text class="text-style">{{touchcancel}}</text>
  </div>
  <div class="text-container" onlongpress="longPress">
```

```
    <text class="text-style">{{onLongPress}}</text>
  </div>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
}
.text-container {
  margin-top: 10px;
  flex-direction: column;
  width: 750px;
  height: 50px;
  background-color: #09ba07;
}
.text-style {
  width: 100%;
  line-height: 50px;
  text-align: center;
  font-size: 24px;
  color: #ffffff;
}
//xxx.js
export default {
  data: {
    touchstart: 'touchstart',
    touchmove: 'touchmove',
    touchend: 'touchend',
    touchcancel: 'touchcancel',
    onClick: 'onclick',
    onLongPress: 'onlongpress',
  },
  touchCancel: function (event) {
    this.touchcancel = 'canceled';
  },
  touchEnd: function(event) {
    this.touchend = 'ended';
  },
  touchMove: function(event) {
    this.touchmove = 'moved';
  },
  touchStart: function(event) {
    this.touchstart = 'touched';
  },
  longPress: function() {
    this.onLongPress = 'longpressed';
```

```
  },
  click: function() {
    this.onClick = 'clicked';
  },
}
```

2. 按键事件

按键事件是智慧屏上特有的手势事件，当用户操作遥控器按键时触发。用户单击一个遥控器按键，通常会触发两次 key 事件：先触发 action 为 0，再触发 action 为 1，即先触发按下事件，再触发抬起事件。action 为 2 的场景比较少见，一般为用户按下按键且不松开，此时 repeatCount 将返回次数。每个物理按键对应各自的按键值(keycode)实现不同的功能，相关代码如下。

```
<!-- xxx.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>
/ * xxx.css * /
.card - box {
  flex - direction: column;
  justify - content: center;
}
.content - box {
  align - items: center;
  height: 200px;
  flex - direction: column;
  margin - left: 200px;
  margin - right: 200px;
}
.content - text {
  font - size: 40px;
  text - align: center;
}
//xxx.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,因此示例中使用了 focus 事件和 blur 事件明确当前焦点的位置。单击上下键选中 up 或 down 按键,即相应的 focused 状态,失去焦点的按键恢复正常的 up 或 down 按键文本,按确认键后该按键变为 keyed 状态。

5.3.6　页面路由

很多应用由多个页面组成,例如,用户可以从音乐列表页面单击歌曲,跳转到该歌曲的播放界面。此过程通过页面路由进行串联,按需实现跳转。

页面路由 router 根据页面的 URI 找到目标页面,从而实现跳转。以最基础的两个页面之间的跳转为例,具体实现步骤如下:在 Project 窗口,打开 entry→src→main→js→default,右击 pages 文件夹,选择 New→JS Page,创建详情页;调用 router.push()路由到详情页,调用 router.back()回到首页。

1. 构建页面布局

index 和 detail 页面均包含一个 text 组件和 button 组件。text 组件指明当前页面,button 组件实现两个页面之间的相互跳转,HML 文件相关代码如下。

```
<!-- index.hml -->
<div class = "container">
  <text class = "title"> This is the index page.</text >
  <button type = "capsule" value = "Go to the second page" class = "button" onclick = "launch">
</button >
</div >
<!-- detail.hml -->
<div class = "container">
```

```
  <text class = "title"> This is the detail page.</text >
  <button type = "capsule" value = "Go back" class = "button" onclick = "launch"></button >
</div >
```

2. 构建页面样式

构建 index 和 detail 页面样式，text 组件和 button 组件居中显示，两个组件间距为 50px。CSS 相关代码如下(两个页面样式代码一致)。

```
/ * index.css * /
/ * detail.css * /
.container {
  flex - direction: column;
  justify - content: center;
  align - items: center;
}
.title {
  font - size: 50px;
  margin - bottom: 50px;
}
```

3. 实现跳转

为了使 button 组件的 launch 方法生效，需要在页面的 JS 文件中实现跳转逻辑。调用 router.push()接口，将 URI 指定的页面添加到路由栈中，即跳转到 URI 指定的页面。在调用 router 方法之前，需要导入 router 模块，相关代码如下。

```
//index.js
import router from '@system.router';
export default {
  launch() {
    router.push ({
      uri: 'pages/detail/detail',
    });
  },
}
//detail.js
import router from '@system.router';
export default {
  launch() {
    router.back();
  },
}
```

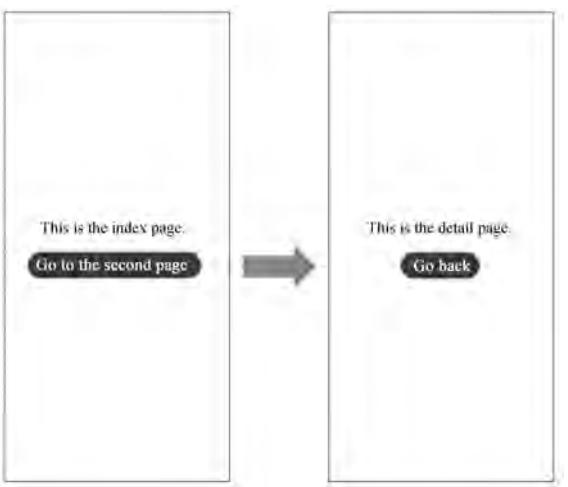

图 5-8　页面路由效果

运行效果如图 5-8 所示。

5.3.7　焦点逻辑

焦点移动是智慧屏的主要交互方式，本节介绍焦点逻辑的相关规则。

1. 容器组件焦点分发逻辑

容器组件在第一次获焦时焦点一般落在第一个可获焦的子组件上，再次获焦时焦点落在上一次失去焦点时获焦的子组件上。容器组件一般都有特定的焦点分发逻辑，常用容器组件的焦点分发逻辑说明如下。

(1) div 组件通过按键移动获焦时，焦点会移动到在移动方向上与当前获焦组件布局中心距离最近的可获焦叶子节点上。如图 5-9 所示，焦点在上方横向 div 的第二个子组件上，当单击 Down 按键时，焦点要移动到下方的横向 div 中，这时下方横向 div 中的子组件会与当前焦点所在的子组件进行布局中心距离的计算，其中距离最近的子组件获焦。

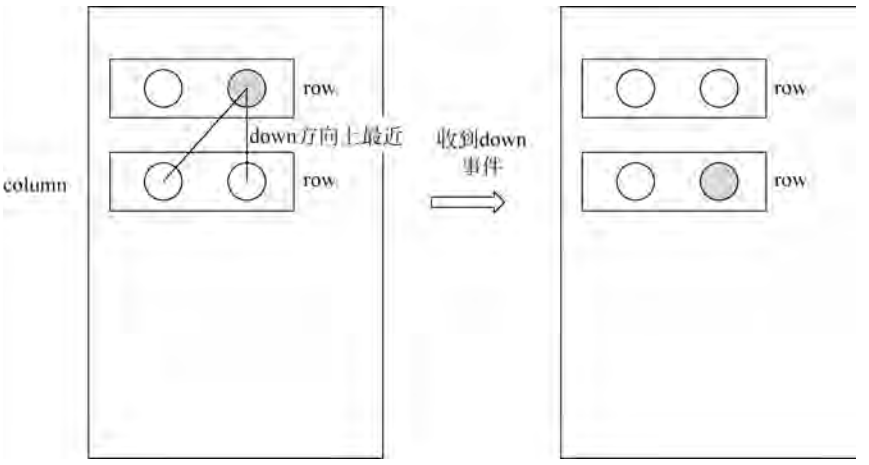

图 5-9　div 焦点移动时距离计算示例

(2) list 组件包含 list-item 与 list-item-group，list 组件每次获焦时会使第一个可获焦的 item 获焦。list-item-group 为特殊的 list-item，且两者都与 div 的焦点逻辑相同。

(3) stack 组件只能由自顶而下的第一个可获焦的子组件获焦。

(4) swiper 的每个页面和 refresh 页面焦点逻辑都与 div 的相同。

(5) tabs 组件包含 tab-bar 与 tab-content，tab-bar 中的子组件默认均能获焦，与是否有可获焦的叶子节点无关。tab-bar 与 tab-content 的每个页面都与 div 的焦点逻辑相同。

(6) dialog 的 button 可获焦，若有多个 button，默认初始焦点落在第二个 button 上。

(7) popup 无法获焦。

2. focusable 属性使用

通用属性 focusable 主要用于控制组件能否获焦，本身不支持焦点的组件在设置此属性后可以拥有获取焦点的能力。例如，text 组件本身不能获焦，焦点无法移动到它上面，设置 text 的 focusable 属性为 true 后，text 组件便可以获焦。如果在未使用 focusable 属性的情

况下，使用了 focus、blur 或 key 事件，会默认添加 focusable 属性为 true。

容器组件能否可获焦依赖于是否拥有可获焦的子组件。如果容器组件内没有可以获焦的子组件，即使设置了 focusable 为 true，依然不能获焦。若容器组件 focusable 属性设置为 false，则它本身和所包含的所有组件都不可获焦。

5.4 常见组件开发

常见组件包括 Text、Input、Button、List、Picker、Dialog、Form、Stepper、Tabs、Image。下面分别介绍功能及其相关代码。

5.4.1 Text

Text 是文本组件，用于呈现一段文本信息。

1. 创建 Text 组件

在 pages/index 目录下的 HML 文件中创建一个 Text 组件，相关代码如下。

```
<!-- xxx.hml -->
<div class="container" style="text-align: center;justify-content: center; align-items: center;">
  <text>
    Hello World
  </text>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  align-items: center;
  justify-content: center;
  background-color: #F1F3F5;
}
```

2. 设置 Text 组件样式和属性

设置 Text 组件样式和属性步骤如下。

1）添加文本样式

设置 color、font-size 和 allow-scale 属性分别为文本添加颜色、大小和缩放，相关代码如下。

```
<!-- xxx.hml -->
<div class="container" style="background-color: #F1F3F5; justify-content: center; align-items: center;">
  <text style="color: blueviolet; font-size: 40px; allow-scale:true">
    This is a passage
  </text>
</div>
```

2）添加画线

设置 text-decoration 属性为文本添加画线，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container" style = "background - color: # F1F3F5;">
  <text style = "text - decoration:underline">
    This is a passage
  </text>
  <text style = "text - decoration:line - through">
    This is a passage
  </text>
</div>
/ * xxx.css * /
.container {
  flex - direction: column;
  align - items: center;
  justify - content: center;
}
text{
  font - size: 50px;
}
```

3）隐藏文本内容

当文本内容过多而显示不全时，添加 text-overflow 属性将隐藏内容以省略号的形式展现，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <text class = "text">
    This is a passage
  </text>
</div>
/ * xxx.css * /
.container {
  flex - direction: column;
  align - items: center;
  background - color: # F1F3F5;
  justify - content: center;
}
.text{
  width: 200px;
  max - lines: 1;
  text - overflow:ellipsis;
}
```

text-overflow 样式需要与 max-lines 样式配套使用，max-lines 属性设置文本最多可以

展示的行数,在设置最大行数的情况下方可生效。

4) 设置文本折行

设置 word-break 属性对文本内容做断行处理,相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <div class="content">
    <text class="text1">
      Welcome to the world
    </text>
    <text class="text2">
      Welcome to the world
    </text>
  </div>
</div>
/* xxx.css */
.container {
  background-color: #F1F3F5;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.content{
  width: 50%;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.text1{
  height: 200px;
  border:1px solid #1a1919;
  margin-bottom: 50px;
  text-align: center;
  word-break: break-word;
  font-size: 40px;
}
.text2{
  height: 200px;
  border:1px solid #0931e8;
  text-align: center;
  word-break: break-all;
  font-size: 40px;
}
```

5) Text 组件支持 Span 子组件

相关代码如下。

```
<!-- xxx.hml -->
<div class = "container" style = "justify - content: center; align - items: center; flex -
direction: column;background - color: #F1F3F5;">
  <text style = "font - size: 45px;">
    This is a passage
  </text>
  <text style = "font - size: 45px;">
    <span style = "color: aqua;"> This </span>
    <span style = "color: #F1F3F5;">
      1
    </span>
    <span style = "color: blue;"> is a </span>
    <span style = "color: #F1F3F5;">
      1
    </span>
    <span style = "color: red;">  passage </span>
  </text>
</div>
```

当使用 Span 子组件组成文本段落时，如果 Span 属性样式异常(font-weight 设置为 1000)，将导致文本段落显示异常。在使用 Span 子组件时，Text 组件内不能存在文本内容；如果存在，只会显示子组件 Span 中的内容。

3. 场景示例

Text 组件通过数据绑定展示文本内容，Span 组件通过设置 show 属性实现文本内容的隐藏和显示，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div style = "align - items: center;justify - content: center;">
    <text class = "title">
      {{ content }}
    </text>
    <switch checked = "true" onchange = "test"></switch>
  </div>
  <text class = "span - container" style = "color: #ff00ff;">
    <span show = "{{isShow}}">  {{ content  }}  </span>
    <span style = "color: white;">
        1
    </span>
    <span style = "color: #f76160"> Hide clip </span>
  </text>
</div>
/* xxx.css */
.container {
  align - items: center;
  flex - direction: column;
```

```
  justify-content: center;
  background-color: #F1F3F5;
}
.title {
  font-size: 26px;
  text-align:center;
  width: 200px;
  height: 200px;
}
//xxx.js
export default {
  data: {
    isShow:true,
    content: 'Hello World'
  },
  onInit(){    },
  test(e) {
    this.isShow = e.checked
  }
}
```

5.4.2 Input

Input 是交互式组件，用于接收用户数据，其类型可设置为日期、多选框和按钮等。

1. 创建 Input 组件

在 pages/index 目录下的 HML 文件中创建 Input 组件，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <input type = "text">
    Please enter the content
  </input>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #F1F3F5;
}
```

2. 设置 Input 类型

通过设置 type 属性定义 Input 类型，将 Input 设置为 button、date 等，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
```

```
  <div class="div-button">
    <dialog class="dialogClass" id="dialogId">
      <div class="content">
        <text>this is a dialog</text>
      </div>
    </dialog>
    <input class="button" type="button" value="click" onclick="btnclick"></input>
  </div>
  <div class="content">
    <input onchange="checkboxOnChange" checked="true" type="checkbox"></input>
  </div>
  <div class="content">
    <input type="date" class="flex" placeholder="Enter data"></input>
  </div>
</div>
/* xxx.css */
.container {
  align-items: center;
  flex-direction: column;
  justify-content: center;
  background-color: #F1F3F5 ;
}
.div-button {
  flex-direction: column;
  align-items: center;
}
.dialogClass{
  width:80%;
  height: 200px;
}
.button {
  margin-top: 30px;
  width: 50%;
}
.content{
  width: 90%;
  height: 150px;
  align-items: center;
  justify-content: center;
}
.flex {
  width: 80%;
  margin-bottom:40px;
}
//xxx.js
export default {
  btnclick(){
```

```
    this.$element('dialogId').show()
  },
}
```

智能穿戴将 Input 类型设置为 button、radio 和 checkbox。当 Input 类型为 checkbox 和 radio 时，当前组件选中的属性为 checked 才生效，默认值为 false。

3. 事件绑定

向 Input 组件添加 search 和 translate 事件，相关代码如下。

```
<!-- xxx.hml -->
<div class="content">
  <text style="margin-left: -7px;">
    <span>Enter text and then touch and hold what you've entered</span>
  </text>
  <input class="input" type="text" onsearch="search" placeholder="search"></input>
  <input class="input" type="text" ontranslate="translate" placeholder="translate"
></input>
</div>
/* xxx.css */
.content {
  width: 100%;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  background-color: #F1F3F5;
}
.input {
  margin-top: 50px;
  width: 60%;
  placeholder-color: gray;
}
text{
  width:100%;
  font-size:25px;
  text-align:center;
}
// xxx.js
import prompt from '@system.prompt'
export default {
  search(e){
    prompt.showToast({
      message: e.value,
      duration: 3000,
    });
  },
  translate(e){
```

```
    prompt.showToast({
      message:  e.value,
      duration: 3000,
    });
  }
}
```

4. 设置输入提示

通过对 Input 组件添加 showError 方法提示输入的错误原因，相关代码如下。

```
<!-- xxx.hml -->
<div class="content">
  <input id="input" class="input" type="text"  maxlength="20" placeholder="Please
input text" onchange="change">
  </input>
  <input class="button" type="button" value="Submit" onclick="buttonClick"></input>
</div>
/* xxx.css */
.content {
  width: 100%;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  background-color: #F1F3F5;
}
.input {
  width: 80%;
  placeholder-color: gray;
}
.button {
  width: 30%;
  margin-top: 50px;
}
//xxx.js
import prompt from '@system.prompt'
export default {
   data:{
     value:'',
   },
   change(e){
     this.value = e.value;
     prompt.showToast({
     message: "value: " + this.value,
       duration: 3000,
      });
   },
   buttonClick(e){
```

```
      if(this.value.length > 6){
        this.$element("input").showError({
          error:  'Up to 6 characters are allowed.'
        });
      }else if(this.value.length == 0){
        this.$element("input").showError({
          error:this.value + 'This field cannot be left empty.'
        });
      }else{
        prompt.showToast({
          message: "success "
        });
      }
    },
}
```

该方法在 Input 类型为 text、email、date、time、number 和 password 时生效。

5. 场景示例

根据场景选择不同类型的 Input 输入框，完成信息录入，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "label - item">
    <label > memorandum </label >
  </div >
  <div class = "label - item">
    <label class = "lab" target = "input1"> content:</label >
    <input class = "flex" id = "input1" placeholder = "Enter content" />
  </div >
  <div class = "label - item">
    <label class = "lab" target = "input3"> date:</label >
    <input class = "flex"id = "input3" type = "date" placeholder = "Enter data" />
  </div >
  <div class = "label - item">
    <label class = "lab" target = "input4"> time:</label >
    <input class = "flex" id = "input4" type = "time" placeholder = "Enter time" />
  </div >
  <div class = "label - item">
    <label class = "lab" target = "checkbox1"> Complete:</label >
    < input class = "flex" type = "checkbox" id = "checkbox1" style = "width: 100px; height:
100px;" />
  </div >
  <div class = "label - item">
    <input class = "flex" type = "button" id = "button" value = "save" onclick = "btnclick"/>
  </div >
</div >
/ * xxx.css * /
```

```
.container {
  flex-direction: column;
  background-color: #F1F3F5;
}
.label-item {
  align-items: center;
  border-bottom-width: 1px;border-color: #dddddd;
}
.lab {
  width: 400px;}
label {
  padding: 30px;
  font-size: 30px;
  width: 320px;
  font-family: serif;
  color: #9370d8;
  font-weight: bold;
}
.flex {
  flex: 1;
}
.textareaPadding {
  padding-left: 100px;
}
//xxx.js
import prompt from '@system.prompt';
export default {
  data: {
  },
  onInit() {
  },
  btnclick(e) {
    prompt.showToast({
      message:'Saved successfully!'
    })
  }
}
```

5.4.3　Button

Button是按钮组件，其类型包括胶囊按钮、圆形按钮、文本按钮、弧形按钮、下载按钮。

1. 创建 Button 组件

在 pages/index 目录下的 hml 文件中创建 Button 组件，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
```

```
  <button  type = "capsule" value = "Capsule button"></button>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #F1F3F5;
}
```

2. 设置 Button 类型

通过设置 Button 的 type 属性选择按钮类型，例如定义 Button 为圆形按钮、文本按钮等，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <button class = "circle" type = "circle" > + </button>
  <button class = "text" type = "text"> button </button>
</div>
/* xxx.css */
.container {
  background-color: #F1F3F5;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.circle {
  font-size: 120px;
  background-color: blue;
  radius: 72px;
}
.text {
  margin-top: 30px;
  text-color: white;
  font-size: 30px;
  font-style: normal;
  background-color: blue;
  width: 50%;
  height: 100px;
}
```

胶囊按钮(type=capsule)不支持 border 相关样式。圆形按钮(type=circle)不支持文本相关样式。文本按钮(type=text)自适应文本大小，不支持尺寸样式设置(radius、width 和 height)，背景透明不支持 background-color 样式。Button 组件使用的 icon 图标如果来自云端路径，需要添加网络访问权限 ohos.permission.INTERNET，在 resources 文件夹下的 config.json 文件中进行权限配置，相关代码如下。

```
<!-- config.json -->
"module": {
  "reqPermissions": [{
    "name": "ohos.permission.INTERNET"
  }],
}
```

3. 显示下载进度

为 Button 组件添加 progress 方法，实时显示下载的进度，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <button class = "button download" type = "download" id = "download - btn" onclick = "setProgress">
{{downloadText}}</button>
</div>
/* xxx.css */
.container {
  background - color: #F1F3F5;
  flex - direction: column;
  align - items: center;
  justify - content: center;
}
.download {
  width: 280px;
  text - color: white;
  background - color: #007dff;
}
//xxx.js
import prompt from '@system.prompt';
export default {
  data: {
    percent: 0,
    downloadText: "Download",
    isPaused: true,
    intervalId : null,
  },
  star(){
    this.intervalId = setInterval(() =>{
      if(this.percent < 100){
        this.percent += 1;
        this.downloadText = this.percent + " % ";
       } else{
         prompt.showToast({
             message: "Download succeeded."
         })
         this.paused()
         this.downloadText = "Download";
```

```
            this.percent = 0;
            this.isPaused = true;
          }
      },100)
    },
    paused(){
      clearInterval(this.intervalId);
      this.intervalId = null;
    },
    setProgress(e) {
      if(this.isPaused){
        prompt.showToast({
          message: "Download started"
        })
        this.star();
        this.isPaused = false;
      }else{
        prompt.showToast({
          message: "Paused."
        })
        this.paused();
        this.isPaused = true;
      }
    }
}
```

setProgress 方法只支持 Button 的类型为 download。

4. 场景示例

在本场景中,可根据输入的文本内容进行 Button 类型切换,相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "input - item">
    <input class = "input - text" id = "change" type = "{{mytype}}"  placeholder = "{{myholder}}"
      style = "background - color:{{mystyle1}};
      placeholder - color:{{mystyle2}};flex - grow:{{myflex}};"name = "{{myname}}" value =
"{{myvalue}}"></input>
  </div>
  <div class = "input - item">
    <div class = "doc - row">
      < input type = "button" class = "select - button color - 3" value = "text" onclick =
"changetype3"></input>
      < input type = "button" class = "select - button color - 3" value = "data" onclick =
"changetype4"></input>
    </div>
  </div>
</div>
```

```
/* xxx.css */
.container {
  flex-direction: column;
  align-items: center;
  background-color: #F1F3F5;
}
.input-item {
  margin-bottom: 80px;
  flex-direction: column;
}
.doc-row {
  justify-content: center;
  margin-left: 30px;
  margin-right: 30px;
  justify-content: space-around;
}
.input-text {
  height: 80px;
  line-height: 80px;
  padding-left: 30px;
  padding-right: 30px;
  margin-left: 30px;
  margin-right: 30px;
  margin-top:100px;
  border: 3px solid;
  border-color: #999999;
  font-size: 30px;
  background-color: #ffffff;
  font-weight: 400;
}
.select-button {
  width: 35%;
  text-align: center;
  height: 70px;
  padding-top: 10px;
  padding-bottom: 10px;
  margin-top: 30px;
  font-size: 30px;
  color: #ffffff;
}
.color-3 {
  background-color: #0598db;;
}
//xxx.js
export default {
  data: {
    myflex: '',
```

```
    myholder: 'Enter text.',
    myname: '',
    mystyle1: "#ffffff",
    mystyle2: "#ff0000",
    mytype: 'text',
    myvalue: '',
  },
  onInit() {
  },
  changetype3() {
    this.myflex = '';
    this.myholder = 'Enter text.';
    this.myname = '';
    this.mystyle1 = "#ffffff";
    this.mystyle2 = "#FF0000";
    this.mytype = 'text';
    this.myvalue = '';
  },
  changetype4() {
    this.myflex = '';
    this.myholder = 'Enter a date.';
    this.myname = '';
    this.mystyle1 = "#ffffff";
    this.mystyle2 = "#FF0000";
    this.mytype = 'date';
    this.myvalue = '';
  },
}
```

其他组件介绍，请扫描二维码获取。

5.5 动效开发

本节主要介绍 CSS 和 JS 动画。其中，CSS 动画包括属性样式、transform 样式、background-position 样式动画。JS 动画包括组件和插值器动画。

5.5.1 CSS 动画开发

本部分包括属性样式动画、transform 样式动画和 background-position 样式动画。

1. 属性样式动画

在关键帧(Keyframes)中动态设置父组件的 width 和 height 值实现变大缩小，子组件设置 scale 属性实现父子组件同时缩放，再设置 opacity 实现父子组件的显示与隐藏，相关代码如下。

```
<!-- xxx.hml -->
```

```
<div class = "container">
  <div class = "fade">
    <text> fading away </text>
  </div>
  <div class = "bigger">
    <text> getting bigger </text>
  </div>
</div>
/* xxx.css */
.container {
  background-color: #F1F3F5;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}
.fade{
  width: 30%;
  height: 200px;
  left: 35%;
  top: 25%;
  position: absolute;
  animation: 2s change infinite friction;
}
.bigger{
  width: 20%;
  height: 100px;
  background-color: blue;
  animation: 2s change1 infinite linear-out-slow-in;
}
text{
  width: 100%;
  height: 100%;
  text-align: center;
  color: white;
  font-size: 35px;
  animation: 2s change2 infinite linear-out-slow-in;
}
/* 颜色变化 */
@keyframes change{
  from {
    background-color: #f76160;
    opacity: 1;
  }
  to {
    background-color: #09ba07;
    opacity: 0;
```

```
  }
}
/* 父组件大小变化 */
@keyframes change1{
  0% {
    width: 20%;
    height: 100px;
  }
  100% {
    width: 80%;
    height: 200px;
  }
}
/* 子组件文字缩放 */
@keyframes change2{
  0%{
   transform: scale(0);
  }
  100% {
    transform: scale(1.5);
  }
}
```

animation 取值不区分先后，duration(动画执行时间)/delay(动画延迟执行时间)按照出现的先后顺序解析。必须设置 animation-duration 样式，否则时长为 0 则不会有动画效果。当设置 animation-fill-mode 属性为 forwards 时，组件直接展示最后一帧的样式。

2. transform 样式动画

设置 transform 属性对组件进行旋转、缩放、移动和倾斜。

1) 设置静态动画

创建一个正方形并旋转 90°变成菱形，用下方的长方形将菱形下半部分遮盖形成屋顶。首先，设置长方形 translate 属性值为(150px，－150px)，确定坐标位置形成门；其次，使用 position 属性将横纵线跟随父组件(正方形)移动到指定坐标位置；再次，设置 scale 属性使父子组件一起变大形成窗户大小；最后，使用 skewX 属性使组件倾斜后设置坐标 translate (200px，－830px)得到烟囱，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "top"></div>
  <div class = "content"></div>
  <div class = "door"></div>
  <!-- 窗户 -->
  <div class = "window">
    <div class = "horizontal"></div>
    <div class = "vertical"></div>
```

```
    </div>
    <div class = "chimney"></div>
</div>
/* xxx.css */
.container {
    background-color:#F1F3F5;
    align-items: center;
    flex-direction: column;
}
.top{
    z-index: -1;
    position: absolute;
    width: 428px;
    height: 428px;
    background-color: #860303;
    transform: rotate(45deg);
    margin-top: 230px;
    margin-left: 266px;
}
.content{
    margin-top: 500px;
    width: 600px;
    height: 400px;
    background-color: white;
    border:  1px solid black;
}
.door{
    width: 100px;
    height: 150px;
    background-color: #1033d9;
    transform: translate(150px, -150px);
}
.window{
    z-index: 1;
    position: relative;
    width: 100px;
    height: 100px;
    background-color: white;
    border: 1px solid black;
    transform: translate(-150px, -400px) scale(1.5);
}
/* 窗户的横轴 */
.horizontal{
    position: absolute;
    top: 50%;
    width: 100px;
    height: 5px;
```

```
    background-color: black;
}
/* 窗户的纵轴 */
.vertical{
    position: absolute;
    left: 50%;
    width: 5px;
    height: 100px;
    background-color: black;
}
.chimney{
    z-index: -2;
    width: 40px;
    height: 100px;
    border-radius: 15px;
    background-color: #9a7404;
    transform: translate(200px, -830px) skewX(-5deg);
}
```

2）设置平移动画

小球下降动画，改变小球的 Y 轴坐标实现小球下落，在下一段时间内减小 Y 轴坐标实现小球回弹，让每次回弹的高度逐次减小直至回弹高度为 0，即模拟出小球下降的动画，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
    <div class="circle"></div>
    <div class="flower"></div>
</div>
/* xxx.css */
.container {
    background-color: #F1F3F5;
    display: flex;
    justify-content: center;
}
.circle{
    width: 100px;
    height: 100px;
    border-radius: 50px;
    background-color: red;
    /* forwards 停在动画的最后一帧 */
    animation: down 3s fast-out-linear-in forwards;
}
.flower{
    position: fixed;
    width: 80%;
```

```
    margin-left: 10%;
    height: 5px;
    background-color: black;
    top: 1000px;
  }
  @keyframes down {
    0% {
      transform: translate(0px,0px);
    }
    /* 下落 */
    15% {
      transform: translate(10px,900px);
    }
    /* 开始回弹 */
    25% {
      transform: translate(20px,500px);
    }
    /* 下落 */
    35% {
      transform: translate(30px,900px);
    }
    /* 回弹 */
    45% {
      transform: translate(40px,700px);
    }
    55% {
      transform: translate(50px,900px);
    }
    65% {
      transform: translate(60px,800px);
    }
    80% {
      transform: translate(70px,900px);
    }
    90% {
      transform: translate(80px,850px);
    }
    /* 停止 */
    100% {
      transform: translate(90px,900px);
    }
  }
```

3) 设置旋转动画

设置不同的原点位置(transform-origin)改变元素围绕的旋转中心。rotate3d 属性前三个参数值分别为 X 轴、Y 轴、Z 轴的旋转向量,第四个值为旋转角度,旋转角度可为负值,负

值代表旋转方向为逆时针，相关代码请扫描二维码获取。

transform-origin 变换对象的原点位置，如果仅设置一个值，另一个值为 50%，若设置两个值，则第一个值表示 X 轴的位置，第二个值表示 Y 轴的位置。

4）设置缩放动画

设置 scale 样式属性实现涟漪动画。首先，使用定位确定元素的位置，确定坐标后创建多个组件实现重合效果；其次，设置 opacity 属性改变组件不透明度实现组件隐藏与显示，同时设置 scale 值，使组件可以一边放大一边隐藏；最后，设置两个组件不同的动画执行时间，实现扩散效果。设置 sacle3d 中 X 轴、Y 轴、Z 轴的缩放参数实现动画，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "circle">
    <text > ripple </text >
  </div >
  <div class = "ripple"></div >
  <div class = "ripple ripple2"></div >
  <!-- 3d -->
  <div class = "content">
    <text > spring </text >
  </div >
</div >
/ * xxx.css * /
.container {
  flex - direction: column;
  background - color: # F1F3F5;
  width: 100 % ;
  position: relative;
}
.circle{
  margin - top: 400px;
  margin - left: 40 % ;
  width: 100px;
  height: 100px;
  border - radius: 50px;
  background:linear - gradient( # dcaec1, # d3a8e3);
  z - index: 1;
  position: absolute;
}
.ripple{
  margin - top: 400px;
  margin - left: 40 % ;
  position: absolute;
  z - index: 0;
  width: 100px;
  height: 100px;
```

```
  border-radius: 50px;
  background:linear-gradient(#dcaec1,#d3a8e3);
  animation: ripple 5s infinite;
}
/*设置不同的动画时间*/
.ripple2{
  animation-duration: 2.5s;
}
@keyframes ripple{
  0%{
    transform: scale(1);
    opacity: 0.5;
  }
  50%{
    transform: scale(3);
    opacity: 0;
  }
  100%{
    transform: scale(1);
    opacity: 0.5;
  }
}
text{
  color: white;
  text-align: center;
  height: 100%;
  width: 100%;
}
.content {
  margin-top: 700px;
  margin-left: 33%;
  width: 200px;
  height: 100px;
  animation:rubberBand 1s infinite;
  /*设置渐变色*/
  background:linear-gradient(#e276aa,#ec0d66);
  position: absolute;
}
@keyframes rubberBand {
  0% {
    transform: scale3d(1, 1, 1);
  }
  30% {
    transform: scale3d(1.25, 0.75, 1.1);
  }
  40% {
    transform: scale3d(0.75, 1.25, 1.2);
```

```
  }
  50% {
    transform: scale3d(1.15, 0.85, 1.3);
  }
  65% {
    transform: scale3d(.95, 1.05, 1.2);
  }
  75% {
    transform: scale3d(1.05, .95, 1.1);
  }
  100%{
    transform: scale3d(1, 1, 1);
  }
}
```

设置 transform 属性值后，子元素会随父元素一起改变，若只改变父元素其他属性值（如 height、width），则子元素不会改变。

5）设置 matrix 属性

matrix 是一个入参为 6 个值的矩阵，6 个值分别代表 scaleX、skewY、skewX、scaleY、translateX 和 translateY。下面示例中设置了 matrix 属性为 matrix(1,0,0,1,0,200)，使组件移动和倾斜，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <div class="rect"></div>
</div>
/* xxx.css */
.container{
  background-color: #F1F3F5;
  display: flex;
  justify-content: center;
}
.rect{
  width: 100px;
  height: 100px;
  background-color: red;
  animation: down 3s infinite forwards;
}
@keyframes down{
  0%{
    transform: matrix(1,0,0,1,0,0);
  }
  10%{
    transform: matrix(1,0,0,1,0,200);
  }
  60%{
```

```
    transform: matrix(2,1.5,1.5,2,0,700);
  }
  100 % {
    transform: matrix(1,0,0,1,0,0);
  }
}
```

6）整合 transform 属性

transform 可以设置多个值并且多个值可同时设置，下面示例中展示同时设置缩放(scale)、平移(translate)和旋转(rotate)属性时的动画效果。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "rect1"></div>
  <div class = "rect2"></div>
  <div class = "rect3"></div>
  <div class = "rect4"></div>
  <div class = "rect5"></div>
</div>
/ * xxx.css * /
.container{
  flex - direction:column;
  background - color: # F1F3F5;
  padding:50px;
}
.rect1{
  width: 100px;
  height: 100px;
  background:linear - gradient( # e77070, # ee0202);
  animation: change1 3s infinite forwards;
}
.rect2{
  margin - top: 50px;
  width: 100px;
  height: 100px;
  background:linear - gradient( # 95a6e8, # 2739de);
  animation: change2 3s infinite forwards;
}
.rect3{
  margin - top: 50px;
  width: 100px;
  height: 100px;
  background:linear - gradient( # 142ee2, # 8cb1e5);
  animation: change3 3s infinite;
}
.rect4{
  align - self: center;
```

```
  margin-left: 50px;
  margin-top: 200px;
  width: 100px;
  height: 100px;
  background:linear-gradient(#e2a8df, #9c67d4,#8245d9,#e251c3);
  animation: change4 3s infinite;
}
.rect5{
  margin-top: 300px;
  width: 100px;
  height: 100px;
  background:linear-gradient(#e7ded7, #486ccd, #94b4d2);
  animation: change5 3s infinite;
}
/*change1 change2 对比*/
@keyframes change1{
  0%{
    transform: translate(0,0);
    transform: rotate(0deg)
  }
  100%{
    transform: translate(0,500px);
    transform: rotate(360deg)
  }
}
/*change2 change3 对比属性顺序不同的动画效果*/
@keyframes change2{
  0%{
    transform:translate(0,0) rotate(0deg) ;
  }
  100%{
    transform: translate(300px,0) rotate(360deg);
  }
}
@keyframes change3{
  0%{
    transform:rotate(0deg) translate(0,0);
  }
  100%{
    transform:rotate(360deg)  translate(300px,0);
  }
}
/*属性值不对应的情况*/
@keyframes change4{
  0%{
    transform: scale(0.5);
  }
```

```
    100 % {
      transform:scale(2) rotate(45deg);
    }
  }
  /* 多属性的写法 */
  @keyframes change5{
    0 % {
      transform:scale(0) translate(0,0) rotate(0);
    }
    100 % {
      transform: scale(1.5) rotate(360deg) translate(200px,0);
    }
  }
```

当设置多个 transform 时，后续的 transform 值会将前面的覆盖。若想同时使用多个动画样式可用复合写法，例如 transform：scale(1) rotate(0) translate(0,0)。transform 进行复合写法时，变化样式内多个样式值顺序的不同会呈现不同的动画效果。transform 属性设置的样式值要一一对应，若前后不一致，则该动画不生效。若设置多个样式值，则只会呈现出已对应值的动画效果。

3. background-position 样式动画

通过改变 background-position 属性（第一个值为 X 轴的位置，第二个值为 Y 轴的位置）移动背景图片位置，若背景图位置超出组件，则超出部分的背景图不显示，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
  <div class = "content"></div>
  <div class = "content1"></div>
</div>
/* xxx.css */
.container {
  background - color: # F1F3F5;
  display: flex;
  flex - direction: column;
  justify - content: center;
  align - items: center;
  width: 100 % ;
}
.content{
  width: 400px;
  height: 400px;
  background - image: url('common/images/bg - tv.jpg');
  background - size: 100 % ;
  background - repeat: no - repeat;
  animation: change 3s infinite;
  border: 1px solid black;
```

```
}
.content1{
  margin-top:50px;
  width: 400px;
  height: 400px;
  background-image: url('common/images/bg-tv.jpg');
  background-size: 50%;
  background-repeat: no-repeat;
  animation: change1 5s infinite;
  border: 1px solid black;
}
/*背景图片移动出组件*/
@keyframes change{
  0%{
    background-position:0px top;
  }
  25%{
    background-position:400px top;
  }
  50%{
    background-position:0px top;
  }
  75%{
    background-position:0px bottom;
  }
  100%{
    background-position:0px top;
  }
}
/*背景图片在组件内移动*/
@keyframes change1{
  0%{
    background-position:left top;
  }
  25%{
    background-position:50% 50%;
  }
  50%{
    background-position:right bottom;
  }
  100%{
    background-position:left top;;
  }
}
```

background-position 仅支持背景图片的移动，不支持背景颜色(background-color)。

5.5.2　JS 动画

本部分介绍组件动画、插值器动画和动画帧。

1. 组件动画

在组件上创建和运行动画的快捷方式步骤如下。

1）获取动画对象

通过调用 animate 方法获得 animation 对象，animation 对象支持动画属性、动画方法和动画事件，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <div id="content" class="box" onclick="Show"></div>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100%;
}
.box{
  width: 200px;
  height: 200px;
  background-color: #ff0000;
  margin-top: 30px;
}
/* xxx.js */
export default {
  data: {
    animation: '',
  },
  onInit() {
  },
  onShow() {
    var options = {
      duration: 1500,
    };
    var frames = [
      {
        width:200,height:200,
      },
      {
        width:300,height:300,
      }
    ];
```

```
    this.animation = this. $element('content').animate(frames, options);
  //获取动画对象
  },
  Show() {
    this.animation.play();
  }
}
```

使用 animate 方法时必须传入 Keyframes 和 Options 参数。多次调用 animate 方法时，采用 replace 策略，即最后一次调用时传入的参数生效。

2）设置动画参数

在获取动画对象后，通过设置参数 Keyframes 设置动画在组件上的样式，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
   <div id = "content" class = "box" onclick = "Show"></div>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100%;
}
.box{
  width: 200px;
  height: 200px;
  background-color: #ff0000;
  margin-top: 30px;
}
/* xxx.js */
export default {
  data: {
    animation: '',
    keyframes:{},
    options:{}
  },
  onInit() {
    this.options = {
      duration: 4000,
    };
    this.keyframes = [
      {
        transform: {
          translate: '-120px -0px',
```

```
          scale: 1,
          rotate: 0
        },
        transformOrigin: '100px 100px',
        offset: 0.0,
        width: 200,
        height: 200
      },
      {
        transform: {
          translate: '120px 0px',
          scale: 1.5,
          rotate: 90
        },
        transformOrigin: '100px 100px',
        offset: 1.0,
        width: 300,
        height: 300
      }
    ];
  },
  Show() {
    this.animation = this.$element('content').animate(this.keyframes, this.options);
    this.animation.play();
  }
}
```

translate、scale 和 totate 的先后顺序会影响动画效果。transformOrigin 只对 scale 和 totate 起作用。在获取动画对象后，通过设置参数 Options 设置动画的属性，相关代码如下。

```
<!-- xxx.hml -->
<div class = "container">
   <div id = "content" class = "box" onclick = "Show"></div>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100%;
}
.box{
  width: 200px;
  height: 200px;
  background-color: #ff0000;
  margin-top: 30px;
```

```
}
/* xxx.js */
export default {
  data: {
    animation: '',
  },
  onInit() {
  },
  onShow() {
    var options = {
      duration: 1500,
      easing: 'ease-in',
      delay: 5,
      iterations: 2,
      direction: 'normal',
    };
    var frames = [
      {
        transform: {
          translate: '-150px -0px'
        }
      },
      {
        transform: {
          translate: '150px 0px'
        }
      }
    ];
    this.animation = this.$element('content').animate(frames, options);
  },
  Show() {
    this.animation.play();
  }
}
```

direction：指定动画的播放模式。normal：动画正向循环播放。reverse：动画反向循环播放。alternate：动画交替循环播放，奇数次正向播放，偶数次反向播放。alternate-reverse：动画反向交替循环播放，奇数次反向播放，偶数次正向播放。

3）添加事件和调用方法

animation 对象支持动画事件和动画方法。可以通过添加开始和取消事件，调用播放、暂停、倒放和结束方法实现预期动画，相关代码请扫描二维码获取。

2. 插值器动画

通过设置插值器实现动画效果，从 API Version 6 开始支持。

（1）创建动画对象。通过 createAnimator 创建一个动画对象，通过设置参数 options 设

置动画的属性，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <div style="width: 300px;height: 300px;margin-top: 100px;background: linear-gradient
(pink, purple);transform: translate({{translateVal}});">
  </div>
  <div class="row">
    <button type="capsule" value="play" onclick="playAnimation"></button>
  </div>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
button{
  width: 200px;
}
.row{
  width: 65%;
  height: 100px;
  align-items: center;
  justify-content: space-between;
  margin-top: 50px;
  margin-left: 260px;
}
/* xxx.js */
import animator from '@ohos.animator';
export default {
  data: {
    translateVal: 0,
    animation: null
  },
  onInit() {},
  onShow(){
    var options = {
      duration: 3000,
      easing:"friction",
      delay:"1000",
      fill: 'forwards',
      direction:'alternate',
      iterations: 2,
      begin: 0,
      end: 180
    }; //设置参数
```

```
    this.animation = animator.createAnimator(options)//创建动画
  },
  playAnimation() {
    var _this = this;
    this.animation.onframe = function(value) {
      _this.translateVal =  value
    };
    this.animation.play();
  }
}
```

使用 createAnimator 创建动画对象时必须传入 options 参数。begin 为插值起点，不设置时默认为 0；end 为插值终点，不设置时默认为 1。

（2）添加动画事件和调用接口。animator 支持事件和接口，可以通过添加 frame、cancel、repeat、finish 事件和调用 update、play、pause、cancel、reverse、finish 接口自定义动画效果，相关代码请扫描二维码获取。

3. 动画帧

本部分包括请求动画帧和取消动画帧。

1）请求动画帧

通过 requestAnimationFrame 函数逐帧回调，在调用该函数时传入一个回调函数。

runframe 在调用 requestAnimationFrame 时传入带有 timestamp 参数的回调函数 step，将 step 中的 timestamp 赋予起始的 startTime。当 timestamp 与 startTime 的差值小于规定的时间时将再次调用 requestAnimationFrame，最终动画将会停止，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <tabs onchange="changecontent">
    <tab-content>
      <div class="container">
        <stack style="width: 300px; height: 300px; margin-top: 100px; margin-bottom:
100px;">
          <canvas id="mycanvas" style="width: 100%; height: 100%; background-color:
coral;">
          </canvas>
          <div style="width: 50px; height: 50px; border-radius: 25px; background-color:
indigo; position: absolute; left: {{left}}; top: {{top}};">
          </div>
        </stack>
        <button type="capsule" value="play" onclick="runframe"></button>
      </div>
    </tab-content>
  </tabs>
</div>
/* xxx.css */
```

```
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100%;
  height: 100%;
}
button{
  width: 300px;
}
/* xxx.js */
export default {
  data: {
    timer: null,
    left: 0,
    top: 0,
    flag: true,
    animation: null,
    startTime: 0,
  },
  onShow() {
    var test = this.$element("mycanvas");
    var ctx = test.getContext("2d");
    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.lineTo(300, 300);
    ctx.lineWidth = 5;
    ctx.strokeStyle = "red";
    ctx.stroke();
  },
  runframe() {
    this.left = 0;
    this.top = 0;
    this.flag = true;
    this.animation = requestAnimationFrame(this.step);
  },
  step(timestamp) {
    if (this.flag) {
      this.left += 5;
      this.top += 5;
      if (this.startTime == 0) {
        this.startTime = timestamp;
      }
      var elapsed = timestamp - this.startTime;
        if (elapsed < 500) {
          console.log('callback step timestamp: ' + timestamp);
          this.animation = requestAnimationFrame(this.step);
```

```
        }
      } else {
        this.left -= 5;
        this.top -= 5;
        this.animation = requestAnimationFrame(this.step);
      }
      if (this.left == 250 || this.left == 0) {
        this.flag = !this.flag
      }
    },
    onDestroy() {
      cancelAnimationFrame(this.animation);
    }
}
```

requestAnimationFrame 函数调用回调函数时，在第一个参数位置传入 timestamp 时间戳，表示 requestAnimationFrame 开始执行回调函数的时刻。

2）取消动画帧

通过 cancelAnimationFrame 函数取消逐帧回调，在调用 cancelAnimationFrame 函数时取消 requestAnimationFrame 函数的请求，相关代码如下。

```
<!-- xxx.hml -->
<div class="container">
  <tabs onchange="changecontent">
    <tab-content>
      <div class="container">
        <stack style="width: 300px; height: 300px; margin-top: 100px; margin-bottom: 100px;">
          <canvas id="mycanvas" style="width: 100%; height: 100%; background-color: coral;">
          </canvas>
          <div style="width: 50px; height: 50px; border-radius: 25px; background-color: indigo; position: absolute; left: {{left}}; top: {{top}};">
          </div>
        </stack>
        <button type="capsule" value="play" onclick="runframe"></button>
      </div>
    </tab-content>
  </tabs>
</div>
/* xxx.css */
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 100%;
```

```
  height: 100 % ;
}
button{
  width: 300px;
}
/ * xxx. js * /
export default {
  data: {
    timer: null,
    left: 0,
    top: 0,
    flag: true,
    animation: null
  },
  onShow() {
    var test = this. $element("mycanvas");
    var ctx = test. getContext("2d");
    ctx. beginPath();
    ctx. moveTo(0, 0);
    ctx. lineTo(300, 300);
    ctx. lineWidth = 5;
    ctx. strokeStyle = "red";
    ctx. stroke();
  },
  runframe() {
    this. left = 0;
    this. top = 0;
    this. flag = true;
    this. animation = requestAnimationFrame(this. step);
  },
  step(timestamp) {
    if (this. flag) {
      this. left += 5;
      this. top += 5;
      this. animation = requestAnimationFrame(this. step);
    } else {
      this. left -= 5;
      this. top -= 5;
      this. animation = requestAnimationFrame(this. step);
    }
    if (this. left == 250 || this. left == 0) {
      this. flag = !this. flag
    }
  },
  onDestroy() {
    cancelAnimationFrame(this. animation);
  }
```

```
}
```

在调用该函数时需传入一个具有标识 ID 的参数。

5.6 自定义组件

使用基于 JS 扩展的类 Web 开发范式的方舟开发框架支持自定义组件，用户可根据业务需求将已有的组件进行扩展，增加自定义的私有属性和事件，封装成新的组件，方便在工程中多次调用，提高页面布局代码的可读性，具体封装方法如下。

1. 构建自定义组件

相关代码如下。

```
<!-- comp.hml -->
<div class = "item">
    <text class = "title - style">{{title}}</text>
    <text class = "text - style" onclick = "childClicked" focusable = "true">单击这里查看隐藏
文本</text>
    <text class = "text - style" if = "{{showObj}}"> hello world </text>
</div>
/ * comp.css * /
.item {
    width: 700px;
    flex - direction: column;
    height: 300px;
    align - items: center;
    margin - top: 100px;
}
.text - style {
    width: 100 % ;
    text - align: center;
    font - weight: 500;
    font - family: Courier;
    font - size: 36px;
}
.title - style {
    font - weight: 500;
    font - family: Courier;
    font - size: 50px;
    color: #483d8b;
}
//comp.js
export default {
    props: {
      title: {
        default: 'title',
```

```
    },
    showObject: {},
  },
  data() {
    return {
      showObj: this.showObject,
    };
  },
  childClicked () {
    this.$emit('eventType1', {text: '收到子组件参数'});
    this.showObj = !this.showObj;
  },
}
```

2. 引入自定义组件

相关代码如下。

```
<!-- xxx.hml -->
<element name = 'comp' src = '../../common/component/comp.hml'></element>
<div class = "container">
  <text>父组件:{{text}}</text>
  <comp title = "自定义组件" show-object = "{{isShow}}" @event-type1 = "textClicked"
></comp>
</div>
/* xxx.css */
.container {
  background-color: #f8f8ff;
  flex: 1;
  flex-direction: column;
  align-content: center;
}
//xxx.js
export default {
  data: {
    text: '开始',
    isShow: false,
  },
  textClicked (e) {
    this.text = e.detail.text;
  },
}
```

本示例中父组件通过添加自定义属性向子组件传递了名称为 title 的参数,子组件在 props 中接收。同时子组件也通过事件绑定向上传递了参数 text,接收时通过 e.detail 获取。如绑定子组件事件,父组件事件命名必须遵循事件绑定规则,自定义组件效果如图 5-10 所示。

图 5-10　自定义组件效果

5.7　JS FA 调用 PA

基于 JS 扩展的类 Web 开发范式的方舟开发框架，提供了 JS FA(Feature Ability)调用 Java PA(Particle Ability)的机制，该机制提供了一种通道传递方法，调用、处理数据返回，上报订阅事件。

当前提供 Ability 和 Internal Ability 两种方式，开发者可以根据业务场景选择合适的调用方式进行开发。

Ability：拥有独立的 Ability 生命周期，FA 使用远端进程通信拉起并请求 PA 服务，适用于基本服务供多 FA 调用或者服务在后台独立运行的场景。

Internal Ability：与 FA 共进程，采用内部函数调用的方式和 FA 进行通信，适用于对服务响应时延要求较高的场景，该方式下 PA 不支持其他 FA 访问调用。

对于 Internal Ability 调用方式的开发，可以使用 js2java-codegen 工具自动生成代码，提高开发效率。

JS 端与 Java 端通过 bundleName 和 abilityName 进行关联。在系统收到 JS 调用请求后，JS 接口中设置的参数选择对应的处理方式。开发者在 onRemoteRequest()中实现 PA 提供的业务逻辑。

1. FA 调用 PA 接口

本部分包括 FA 端和 PA 端提供的接口。

1) FA 端提供以下三个 JS 接口

FeatureAbility. callAbility(OBJECT)：调用 PA。

FeatureAbility. subscribeAbilityEvent(OBJECT,Function)：订阅 PA。

FeatureAbility. unsubscribeAbilityEvent(OBJECT)：取消订阅 PA。

2) PA 端提供以下两类接口

IRemoteObject. onRemoteRequest(int, MessageParcel, MessageParcel, MessageOption)：Ability 调用方式，FA 使用远端进程通信拉起并请求 PA 服务。

AceInternalAbility. AceInternalAbilityHandler. onRemoteRequest(int,MessageParcel,MessageParcel,MessageOption)：Internal Ability 调用方式，采用内部函数调用的方式和

FA 进行通信。

2. FA 调用 PA 常见问题

callAbility 返回报错：Internal ability not register。返回该错误说明 JS 接口调用请求未在系统中找到对应的 InternalAbilityHandler 进行处理，因此需要检查以下几点是否正确执行。

(1) 在 AceAbility 继承类中对 AceInternalAbility 继承类执行了 register 方法。

(2) JS 侧填写的 bundleName 和 abilityName 与 AceInternalAbility 继承类构造函数中填写的名称保持相同，大小写敏感。

(3) 检查 JS 端填写的 abilityType(0：Ability；1：Internal Ability)，确保没有将 AbilityType 缺省或误填写为 Ability 方式。

Ability 和 Internal Ability 是两种不同的 FA 调用 PA 的方式。Ability 和 InternalAbility 差异项如表 5-2 所示，避免开发时将两者混淆使用。

表 5-2　Ability 和 InternalAbility 差异项

差　异　项	Ability	InternalAbility
JS 端(abilityType)	0	1
是否需要在 config. json 的 abilities 中为 PA 添加声明	需要(有独立的生命周期)	不需要(和 FA 共生命周期)
是否需要在 FA 中注册	不需要	需要
继承的类	ohos. aafwk. ability. Ability	ohos. ace. ability. AceInternalAbility
是否允许被其他 FA 访问调用	是	否

FeatureAbility. callAbility 中 syncOption 参数说明如下。

JS FA 侧返回的结果都是 Promise 对象，因此无论该参数取何值，都采用异步方式等待 PA 侧响应。对于 JAVA PA 侧，在 Internal Ability 方式下收到 FA 的请求后，根据该参数的取值选择通过同步的方式获取结果后返回，或者异步执行 PA 逻辑，获取结果后使用 remoteObject. sendRequest 的方式将结果返回 FA。

使用 await 方式调用时 IDE 编译报错，需引入 babel-runtime/regenerator。

3. 示例参考

JS 端调用 FeatureAbility 接口，传入两个 Number 参数，Java 端接收后返回两个数的和。JS FA 应用的 JS 模块(entry/src/main)典型开发目录结构如图 5-11 所示。

1) FA JavaScript 端

使用 Internal Ability 方式时，需要将对应的 action. abilityType 值改为 ABILITY_TYPE_INTERNAL，相关代码如下。

```
//abilityType: 0 - Ability; 1 - Internal Ability
const ABILITY_TYPE_EXTERNAL = 0;
const ABILITY_TYPE_INTERNAL = 1;
//syncOption(Optional, default sync): 0 - Sync; 1 - Async
```

```
const ACTION_SYNC = 0;
const ACTION_ASYNC = 1;
const ACTION_MESSAGE_CODE_PLUS = 1001;
export default {
  plus: async function() {
    var actionData = {};
    actionData.firstNum = 1024;
    actionData.secondNum = 2048;
    var action = {};
      action.bundleName = 'com.example.
hiaceservice';
    action.abilityName = 'com.example.hiaceservice.
ComputeServiceAbility';
     action.messageCode = ACTION_MESSAGE_CODE_
PLUS;
    action.data = actionData;
    action.abilityType = ABILITY_TYPE_EXTERNAL;
    action.syncOption = ACTION_SYNC;
    var result = await FeatureAbility.callAbility
(action);
    var ret = JSON.parse(result);
    if (ret.code == 0) {
      console.info('plus result is: ' + JSON.
stringify(ret.abilityResult));
    } else {
      console.error('plus error code:' + JSON.
stringify(ret.code));
    }
  }
}
```

```
main
├ config.json
├ java
│ └ com.example.hiaceservice.ComputeServiceAbility
│           ├ ComputeInternalAbility.java
│           ├ ComputeServiceAbility.java
│           ├ MainAbility.java
│           ├ MyApplication.java
│           └ RequestParam.java
├ js
│ └ default
│   ├ app.js
│   ├ common (可选)
│   │ ├ component
│   │ │ └ component
│   │ │   ├ componentA.css
│   │ │   ├ componentA.hml
│   │ │   └ componentA.js
│   │ └ images
│   │   ├ ***.jpg
│   │   └ ***.png
│   ├ i18n (可选)
│   │ ├ en-US.json
│   │ └ zh-CN.json
│   └ pages
│     ├ index
│     │ ├ index.css
│     │ ├ index.hml
│     │ └ index.js
│     └ detail (可选)
│       ├ detail.css
│       ├ detail.hml
│       └ detail.js
└ resources (可选)
  ├ base
  │ ├ element
  │ │ └ string.json
  │ └ media
  │   └ icon.png
  └ rawfile
```

图 5-11 JS 模块典型开发目录结构

2）PA 端(Ability 方式)

功能代码实现：在 Java 目录下新建一个 Service Ability，文件命名为 ComputeServiceAbility.java，相关代码如下。

```
package com.example.hiaceservice;
//ohos 相关接口包
import ohos.aafwk.ability.Ability;
import ohos.aafwk.content.Intent;
import ohos.hiviewdfx.HiLog;
import ohos.hiviewdfx.HiLogLabel;
import ohos.rpc.IRemoteBroker;
import ohos.rpc.IRemoteObject;
import ohos.rpc.RemoteObject;
import ohos.rpc.MessageParcel;
import ohos.rpc.MessageOption;
import ohos.utils.zson.ZSONObject;
```

```
import java.util.HashMap;
import java.util.Map;
public class ComputeServiceAbility extends Ability {
  //定义日志标签
  private static final HiLogLabel LABEL = new HiLogLabel(HiLog.LOG_APP, 0, "MY_TAG");
  private MyRemote remote = new MyRemote();
  //FA 在请求 PA 服务时会调用 Ability.connectAbility 连接 PA,连接成功后,需要在 onConnect 返
  //回一个 remote 对象,供 FA 向 PA 发送消息
  @Override
  protected IRemoteObject onConnect(Intent intent) {
    super.onConnect(intent);
    return remote.asObject();
  }
  class MyRemote extends RemoteObject implements IRemoteBroker {
    private static final int SUCCESS = 0;
    private static final int ERROR = 1;
    private static final int PLUS = 1001;
    MyRemote() {
      super("MyService_MyRemote");
    }
    @Override
     public boolean onRemoteRequest(int code, MessageParcel data, MessageParcel reply,
MessageOption option) {
      switch (code) {
        case PLUS: {
          String dataStr = data.readString();
          RequestParam param = new RequestParam();
          try {
              param = ZSONObject.stringToClass(dataStr, RequestParam.class);
          } catch (RuntimeException e) {
              HiLog.error(LABEL, "convert failed.");
          }
         //返回结果当前仅支持 String,对于复杂结构可以序列化为 Zson 字符串上报
          Map<String, Object> result = new HashMap<String, Object>();
          result.put("code", SUCCESS);
          result.put("abilityResult", param.getFirstNum() + param.getSecondNum());
          reply.writeString(ZSONObject.toZSONString(result));
          break;
        }
        default: {
          Map<String, Object> result = new HashMap<String, Object>();
          result.put("abilityError", ERROR);
          reply.writeString(ZSONObject.toZSONString(result));
          return false;
        }
      }
      return true;
```

```
    }
    @Override
    public IRemoteObject asObject() {
      return this;
    }
  }
}
```

请求参数代码如下。

```
RequestParam.java
public class RequestParam {
  private int firstNum;
  private int secondNum;
  public int getFirstNum() {
    return firstNum;
  }
  public void setFirstNum(int firstNum) {
    this.firstNum = firstNum;
  }
  public int getSecondNum() {
    return secondNum;
  }
  public void setSecondNum(int secondNum) {
    this.secondNum = secondNum;
  }
}
```

3) PA 端(Internal Ability 方式)

功能代码实现可以使用 js2java-codegen 工具自动生成：在 Java 目录下新建一个 Service Ability，文件命名为 ComputeInternalAbility.java，相关代码请扫描二维码获取。

5.8 使用工具自动生成 JS FA 调用 PA 代码

JS FA 调用 PA 是基于 JS 扩展的类 Web 开发范式的方舟开发框架所提供的一种跨语言能力调用的机制，用于建立 JS 能力与 Java 能力之间传递方法调用、处理数据返回及订阅事件上报的通道。开发者可以使用 FA 调用 PA 机制进行应用开发，但直接使用需要手动撰写大量模板代码，且模板代码可能与业务代码相互耦合，使代码可维护性和可读性较差。

为提升开发效率，快速完成 FA 调用 PA 应用，可以在 DevEco Studio 环境中借助 js2java-codegen 工具自动生成 JS FA 调用 PA 代码(目前仅支持 InternalAbility 调用方式)。开发者只需添加简单的配置与标注即可利用该工具完成大部分 FA 调用 PA 模板代码的编写，同时也有效地将业务代码与模板代码相互分离。

1. js2java-codegen 工具简介

js2java-codegen 是工具链提供的自动生成 JS FA 调用 PA 代码的辅助开发工具。它可

以根据用户源码生成 FA 调用 PA 所需的、与用户编写的业务代码相互分离的模板代码。

js2java-codegen 工具所支持的 FA 调用 PA 实现方式为 InternalAbility 类型，目前尚不支持 Ability 类型。开发者完成设置后只需编写包含实际业务逻辑的 InternalAbility 类和需要注册的 Ability 类，并在 InternalAbility 类中加上对应注解，js2java-codegen 即可在编译过程中完成 FA 调用 PA 通道的建立。之后，只需在 JS 侧调用由 js2java-codegen 工具生成的 JS 接口即可调用 Java 一侧的能力。

js2java-codegen 工具所生成的模板代码包含 Java 代码和 JS 代码。其中，Java 代码会被直接编译成字节码文件，并且对应 Ability 类中会被自动添加注册与反注册语句，开发者无须关注；而 JS 代码则需要用户手动调用，因此需要在编译前设置好 JS 代码的输出路径。

注解使用说明如下：js2java-codegen 工具通过注解获取信息并生成开发者所需的代码。因此，用户如果使用该工具辅助开发，则需要了解以下三种用法。

1）@InternalAbility 注解

@InternalAbility 注解为类注解，用于 InternalAbility、包含实际业务代码的类（简称 InternalAbility 类），只支持文件中 public 的顶层类，不支持接口类和注解类，包含一个参数 registerTo，值为需要注册到 Ability 类的全名。示例如下，Service 类是一个 InternalAbility 类，注册到位于 com.example 包中的、名为 Ability 的 Ability 类。

```
@InternalAbility(registerTo = "com.example.Ability")
public class Service{}
```

2）@ExportIgnore 注解

@ExportIgnore 注解为方法注解，用于 InternalAbility 类中的某些方法，表示该方法不暴露给 JS 侧调用，仅对 public 方法有效。示例如下，service 方法不会被暴露给 JS 侧。

```
@ExportIgnore
public int service(int input) {
    return input;
}
```

3）@ContextInject 注解

@ContextInject 注解用于 AbilityContext 上的注解。该类由 HarmonyOS 的 Java API 提供，可通过它获取 API 中提供的信息。

可以借助 abilityContext 对象获取 API 中提供的信息，示例如下。

```
@ContextInject
AbilityContext abilityContext;
```

2. 新建工程

体验工具生成模板代码的功能，可使用 DevEco Studio 新建一个包含 JS 前端的简单手机项目，并用其开发一个简单的 FA 调用 PA 应用。

3. 工具开关与编译设置

快速验证功能可选择修改 entry 模块的 build.gradle，通过 entry 模块进行验证。

编译参数位于 ohos→defaultConfig 中，只需添加如下设置即可。开发者需在此处设置 JS 模板代码生成路径，即 jsOutputDir 对应的值。

```
//在文件头部定义 JS 模板代码生成路径
def jsOutputDir = project.file("src/main/js/default/generated").toString()
//在 ohos→defaultConfig 中设置 JS 模板代码生成路径
javaCompileOptions {
    annotationProcessorOptions {
        arguments = ["jsOutputDir": jsOutputDir] //JS 模板代码生成赋值
    }
}
```

工具开关位于 ohos 中，只需添加如下设置即可。值设为 true 则启用工具，值设为 false 则不进行配置，不启用工具。

```
compileOptions {
    f2pautogenEnabled true //此处为启用 js2java - codegen 工具的开关
}
```

4. Java 侧代码编写

模板代码的生成需要提供用于 FA 调用的 PA，因此需要编写 InternalAbility 类，在类上加@InternalAbility 注解，registerTo 参数设为将要注册到的 Ability 类的全称。Ability 类可使用项目中已有的 MainAbility 类，或创建新的 Ability 类。

注意，InternalAbility 类中需要暴露给 FA 调用的方法只能是 public 类型的非静态非 void 方法，若不是则不会被暴露。

一个简单的 InternalAbility 类实现如下，文件名为 Service.java，与 MainAbility 类同包，用注解注册到 MainAbility 类。类中包含一个 ADD 方法作为暴露给 JS FA 调用的能力，实现两数相加的功能，入参为两个 int 参数，返回值为两数之和，示例如下。

```
package com.example.myapplication;
import ohos.annotation.f2pautogen.InternalAbility;
@InternalAbility(registerTo = "com.example.myapplication.MainAbility")
//此处 registerTo 的参数为项目中 MainAbility 类的全称
public class Service {
    public int add(int num1, int num2) {
        return num1 + num2;
    }
}
```

5. 编译

js2java-codegen 工具在编译过程中会自动被调用、生成模板代码并完成整个通道建立的过程。

单击菜单栏中的 Build→Build HAP(s)/APP(s)→Build HAP(s)，即可完成项目编译，同时 js2java-codegen 工具会在编译过程中完成 FA 调用 PA 通道的建立。

编译过程会生成 Java 和 JS 的模板代码。其中，JS 的模板代码位于编译设置中的路径，名称与 InternalAbility 类相对应；而 Java 的模板代码位于 entry→build→generated→source→annotation→debug→InternalAbility 类同名包→InternalAbility 类名＋Stub. java，该类的调用语句会被注入 MainAbility 类的字节码当中，生成的模板如图 5-12 和图 5-13 所示。

```
package com.example.myapplication;

import ...

public class ServiceStub extends AceInternalAbility {
  public static final String BUNDLE_NAME = "com.example.myapplication";

  public static final String ABILITY_NAME = "com.example.myapplication.MainAbility";

  public static final int ERROR = -1;

  public static final int SUCCESS = 0;

  public static final int OPCODE_add = 0;

  private static ServiceStub instance;

  private Service service;

  private AbilityContext abilityContext;

  public ServiceStub() { super(BUNDLE_NAME, ABILITY_NAME); }
```

图 5-12　Java 模板代码示例

```
package com.example.myapplication;

import ...

public class ServiceStub extends AceInternalAbility {
  public static final String BUNDLE_NAME = "com.example.myapplication";

  public static final String ABILITY_NAME = "com.example.myapplication.MainAbility";

  public static final int ERROR = -1;

  public static final int SUCCESS = 0;

  public static final int OPCODE_add = 0;

  private static ServiceStub instance;

  private Service service;

  private AbilityContext abilityContext;

  public ServiceStub() { super(BUNDLE_NAME, ABILITY_NAME); }
```

图 5-13　JS 模板代码示例

6. JS 侧代码编写

为了简易直观地检验工具生成代码的可用性，开发者可以通过修改 entry→src→main→js→default→pages→index→index.js 调用 Java 侧的能力，并在前端页面展示效果。

可通过 import 方式引入 JS 侧 FA 接口，例如 import Service from '../../generated/Service.js'；（from 后的值需要与编译设置中的路径进行统一，生成的 JS 代码文件名及类名与 InternalAbility 类名相同）。

一个简单的 index.js 页面实现如下，调用 JS 侧接口，传入 1 和 10 两个参数，并把返回的结果打印在 title 中，这样只要运行该应用就可以验证 FA 调用 PA 是否成功。

```
import Service from '../../generated/Service.js';      //此处 FA 路径和类名对应之前的 jsOutput 路径及 InternalAbility 的名称
export default {
    data: {
        title: "Result:"
    },
    onInit() {
        const echo = new Service(); //此处新建 FA 实例
        echo.add(1,10)
            .then((data) => {
            this.title += data["abilityResult"];  //获取运算结果，并加到 title 之后
        });
    }
}
```

为了方便结果展示，这里对同目录下的 index.hml 也做一些修改，使页面中只显示 title 的内容。

```
<div class = "container">
    <text class = "title">
        {{ title }}
    </text>
</div>
```

7. 结果验证

启动手机模拟器，成功后运行，看到显示界面则说明 js2java-codegen 工具生成了有效的模板代码，建立了 FA 调用 PA 的通道。