

HarmonyOS 开发文档 (一)

V1.0

鸿蒙学堂 hmxt.org 整理

2020年9月10日

目 录

1	Ability.....	1
1.1	Ability	1
1.1.1	概述.....	1
1.1.2	Page Ability.....	2
1.1.3	Service Ability.....	13
1.1.4	Data Ability	21
1.1.5	Intent.....	32
1.1.6	Ability Form	36
1.2	分布式任务调度.....	42
1.2.1	概述.....	42
1.2.2	开发指导.....	43
1.3	公共事件与通知.....	57
1.3.1	概述.....	57
1.3.2	公共事件开发指导.....	58
1.3.3	通知开发指导.....	68
1.3.4	IntentAgent 开发指导.....	75
1.4	剪贴板.....	79
1.4.1	概述.....	79
1.4.2	开发指导.....	80
2	线程.....	86
2.1	线程管理.....	87
2.1.1	概述.....	87
2.1.2	开发指导.....	87
2.2	线程间通信.....	96
2.2.1	概述.....	96
2.2.2	开发指导.....	98
3	UI.....	110
3.1	Java UI 框架.....	110
3.1.1	概述.....	110
3.1.2	组件与布局开发指导.....	112
3.1.3	常用组件开发指导.....	121
3.1.4	常用布局开发指导.....	141
3.1.5	动画开发指导.....	163
3.1.6	可见即可说开发指导.....	166
3.2	JS UI 框架.....	167

3.2.1	概述.....	167
3.2.2	初步体验 JS FA 应用.....	169
3.2.3	构建用户界面.....	181
3.2.4	自定义组件.....	206
3.2.5	JS FA 如何调用 PA.....	208
3.3	多模输入.....	217

声明：所有内容均来自华为官方网站，如有错误，欢迎指正。

1 Ability

1.1 Ability

1.1.1 概述

Ability 是应用所具备能力的抽象，也是应用程序的重要组成部分。一个应用可以具备多种能力（即可以包含多个 Ability），HarmonyOS 支持应用以 Ability 为单位进行部署。Ability 可以分为 FA（Feature Ability）和 PA（Particle Ability）两种类型，每种类型为开发者提供了不同的模板，以便实现不同的业务功能。

- FA 支持 Page Ability:

Page 模板是 FA 唯一支持的模板，用于提供与用户交互的能力。一个 Page 实例可以包含一组相关页面，每个页面用一个 AbilitySlice 实例表示。

- PA 支持 Service Ability 和 Data Ability:

- Service 模板：用于提供后台运行任务的能力。
- Data 模板：用于对外部提供统一的数据访问抽象。

在配置文件（config.json）中注册 Ability 时，可以通过配置 Ability 元素中的“type”属性来指定 Ability 模板类型，示例如下。

其中，“type”的取值可以为“page”、“service”或“data”，分别代表 Page 模板、Service 模板、Data 模板。为了便于表述，后文中我们将基于 Page 模板、Service 模板、Data 模板实现的 Ability 分别简称为 Page、Service、Data。

```
1. {  
2.     "module": {  
3.         ...
```

```
4.     "abilities": [  
5.         {  
6.             ...  
7.             "type": "page"  
8.             ...  
9.         }  
10.    ]  
11.    ...  
12. }  
13. ...  
14. }
```

1.1.2 Page Ability

1.1.2.1 基本概念

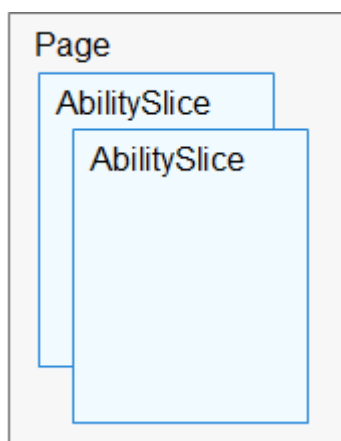
Page 与 AbilitySlice

Page 模板（以下简称“Page”）是 FA 唯一支持的模板，用于提供与用户交互的能力。一个 Page 可以由一个或多个 AbilitySlice 构成，AbilitySlice 是指应用的单个页面及其控制逻辑的总和。

当一个 Page 由多个 AbilitySlice 共同构成时，这些 AbilitySlice 页面提供的业务能力应具有高度相关性。例如，新闻浏览功能可以通过一个 Page 来实现，其中包含了两个 AbilitySlice：一个 AbilitySlice 用于展示新闻列表，另一个 AbilitySlice 用于展示新闻详情。Page 和

AbilitySlice 的关系如图 1 所示。

图 1 Page 与 AbilitySlice



相比于桌面场景，移动场景下应用之间的交互更为频繁。通常，单个应用专注于某个方面的能力开发，当它需要其他能力辅助时，会调用其他应用提供的能力。例如，外卖应用提供了联系商家业务功能入口，当用户在使用该功能时，会跳转到通话应用的拨号页面。与此类似，HarmonyOS 支持不同 Page 之间的跳转，并可以指定跳转到目标 Page 中某个具体的 AbilitySlice。

AbilitySlice 路由配置

虽然一个 Page 可以包含多个 AbilitySlice，但是 Page 进入前台时界面默认只展示一个 AbilitySlice。默认展示的 AbilitySlice 是通过 `setMainRoute()` 方法来指定的。如果需要更改默认展示的 AbilitySlice，可以通过 `addActionRoute()` 方法为此 AbilitySlice 配置一条路由规则。此时，当其他 Page 实例期望导航到此 AbilitySlice 时，可以在 Intent 中指定 Action，详见不同 Page 间导航。

`setMainRoute()` 方法与 `addActionRoute()` 方法的使用示例如下：

```
1. public class MyAbility extends Ability {
2.     @Override
3.     public void onStart(Intent intent) {
4.         super.onStart(intent);
```

```
5.     // set the main route
6.     setMainRoute(MainSlice.class.getName());
7.
8.     // set the action route
9.     addActionRoute("action.pay", PaySlice.class.getName());
10.    addActionRoute("action.scan", ScanSlice.class.getName());
11.    }
12. }
```

addActionRoute()方法中使用的动作命名，需要在应用配置文件（config.json）中注册：

```
1.  {
2.    "module": {
3.      "abilities": [
4.        {
5.          "skills": [
6.            {
7.              "actions": [
8.                "action.pay",
9.                "action.scan"
10.             ]
11.           }
12.         ]
13.       }
14.     ]
15.   }
16.   ...
17. }
18. ...
19. }
```

1.1.2.2 生命周期

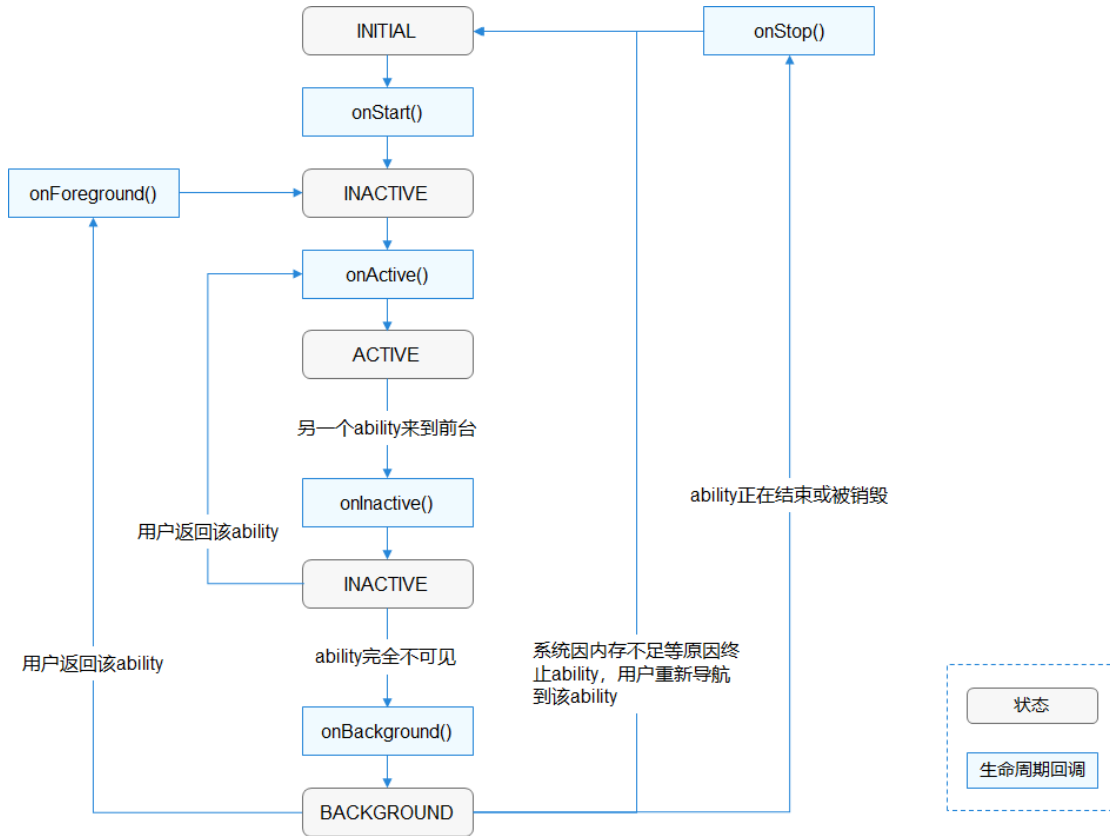
系统管理或用户操作等行为均会引起 Page 实例在其生命周期的不同状态之间进行转换。

Ability 类提供的回调机制能够让 Page 及时感知外界变化，从而正确地应对状态变化（比如释放资源），这有助于提升应用的性能和稳健性。

Page 生命周期回调

Page 生命周期的不同状态转换及其对应的回调，如图 1 所示。

图 1 Page 生命周期



- **onStart()**

当系统首次创建 Page 实例时，触发该回调。对于一个 Page 实例，该回调在其生命周期过程中仅触发一次，Page 在该逻辑后将进入 INACTIVE 状态。开发者必须重写该方法，并在此配置默认展示的 AbilitySlice。

```

1.  @Override
2.  public void onStart(Intent intent) {
3.      super.onStart(intent);
4.      super.setMainRoute(FooSlice.class.getName());
5.  }
    
```

- **onActive()**

Page 会在进入 INACTIVE 状态后来回到前台，然后系统调用此回调。Page 在此之后进入 ACTIVE 状态，该状态是应用与用户交互的状态。Page 将保持在此状态，除非某类事件发生导致 Page 失去焦点，比如用户点击返回键或导航到其他 Page。当此类事件发生时，会触发 Page 回到 INACTIVE 状态，系统将调用 onInactive()回调。此后，Page 可能重新回到 ACTIVE 状态，系统将再次调用 onActive()回调。因此，开发者通常需要成对实现 onActive()和 onInactive()，并在 onActive()中获取在 onInactive()中被释放的资源。

- **onInactive()**

当 Page 失去焦点时，系统将调用此回调，此后 Page 进入 INACTIVE 状态。开发者可以在此回调中实现 Page 失去焦点时应表现的恰当行为。

- **onBackground()**

如果 Page 不再对用户可见，系统将调用此回调通知开发者用户进行相应的资源释放，此后 Page 进入 BACKGROUND 状态。开发者应该在此回调中释放 Page 不可见时无用的资源，或在此回调中执行较为耗时的状态保存操作。

- **onForeground()**

处于 BACKGROUND 状态的 Page 仍然驻留在内存中，当重新回到前台时（比如用户重新导航到此 Page），系统将先调用 onForeground()回调通知开发者，而后 Page 的生命周期状态回到 INACTIVE 状态。开发者应当在此回调中重新申请在 onBackground()中释放的资源，最后 Page 的生命周期状态进一步回到 ACTIVE 状态，系统将通过 onActive()回调通知开发者用户。

- **onStop()**

系统将要销毁 Page 时，将会触发此回调函数，通知用户进行系统资源的释放。销毁 Page 的可能原因包括以下几个方面：

- 用户通过系统管理能力关闭指定 Page，例如使用任务管理器关闭 Page。
- 用户行为触发 Page 的 terminateAbility()方法调用，例如使用应用的退出功能。

- 配置变更导致系统暂时销毁 Page 并重建。
- 系统出于资源管理目的，自动触发对处于 BACKGROUND 状态 Page 的销毁。

AbilitySlice 生命周期

AbilitySlice 作为 Page 的组成单元，其生命周期是依托于其所属 Page 生命周期的。

AbilitySlice 和 Page 具有相同的生命周期状态和同名的回调，当 Page 生命周期发生变化时，它的 AbilitySlice 也会发生相同的生命周期变化。此外，AbilitySlice 还具有独立于 Page 的生命周期变化，这发生在同一 Page 中的 AbilitySlice 之间导航时，此时 Page 的生命周期状态不会改变。

AbilitySlice 生命周期回调与 Page 的相应回调类似，因此不再赘述。由于 AbilitySlice 承载具体的页面，开发者必须重写 AbilitySlice 的 onStart()回调，并在此方法中通过 setUIContent()方法设置页面，如下所示：

```
1.  @Override
2.  protected void onStart(Intent intent) {
3.      super.onStart(intent);
4.
5.      setUIContent(ResourceTable.Layout_main_layout);
6.  }
```

AbilitySlice 实例创建和管理通常由应用负责，系统仅在特定情况下会创建 AbilitySlice 实例。

例如，通过导航启动某个 AbilitySlice 时，是由系统负责实例化；但是在同一个 Page 中不同的 AbilitySlice 间导航时则由应用负责实例化。

Page 与 AbilitySlice 生命周期关联

当 AbilitySlice 处于前台且具有焦点时，其生命周期状态随着所属 Page 的生命周期状态的变化而变化。当一个 Page 拥有多个 AbilitySlice 时，例如：MyAbility 下有 FooAbilitySlice 和 BarAbilitySlice，当前 FooAbilitySlice 处于前台并

获得焦点，并即将导航到 BarAbilitySlice，在此期间的生命周期状态变化顺序为：

1. FooAbilitySlice 从 ACTIVE 状态变为 INACTIVE 状态。
2. BarAbilitySlice 则从 INITIAL 状态首先变为 INACTIVE 状态，然后变为 ACTIVE 状态（假定此前 BarAbilitySlice 未曾启动）。
3. FooAbilitySlice 从 INACTIVE 状态变为 BACKGROUND 状态。

对应两个 slice 的生命周期方法回调顺序为：

```
FooAbilitySlice.onInactive() --> BarAbilitySlice.onStart() --> BarAbilitySlice.onActive() -  
-> FooAbilitySlice.onBackground()
```

在整个流程中，MyAbility 始终处于 ACTIVE 状态。但是，当 Page 被系统销毁时，其所有已实例化的 AbilitySlice 将联动销毁，而不仅是处于前台的 AbilitySlice。

1.1.2.3 AbilitySlice 间导航

同一 Page 内导航

当发起导航的 AbilitySlice 和导航目标的 AbilitySlice 处于同一个 Page 时，您可以通过 present()方法实现导航。如下代码片段展示通过点击按钮导航到其他 AbilitySlice 的方法：

```
1. @Override  
2. protected void onStart(Intent intent) {  
3.  
4.     ...  
5.     Button button = ...;  
6.     button.setOnClickListener(listener -> present(new TargetSlice(), new Intent()));  
7.     ...  
8.  
9. }
```

如果开发者希望在用户从导航目标 AbilitySlice 返回时，能够获得其返回结果，则应当使用 `presentForResult()` 实现导航。用户从导航目标 AbilitySlice 返回时，系统将回调 `onResult()` 来接收和处理返回结果，开发者需要重写该方法。返回结果由导航目标 AbilitySlice 在其生命周期内通过 `setResult()` 进行设置。

```
1. @Override
2. protected void onStart(Intent intent) {
3.
4.     ...
5.     Button button = ...;
6.     button.setOnClickListener(listener -> presentForResult(new TargetSlice(), new Intent(), 0));
7.     ...
8.
9. }
10.
11. @Override
12. protected void onResult(int requestCode, Intent resultIntent) {
13.     if (requestCode == 0) {
14.         // Process resultIntent here.
15.     }
16. }
```

系统为每个 Page 维护了一个 AbilitySlice 实例的栈，每个进入前台的 AbilitySlice 实例均会入栈。当开发者在调用 `present()` 或 `presentForResult()` 时指定的 AbilitySlice 实例已经在栈中存在时，则栈中位于此实例之上的 AbilitySlice 均会出栈并终止其生命周期。前面的示例代码中，导航时指定的 AbilitySlice 实例均是新建的，即便重复执行此代码（此时作为导航目标的这些实例是同一个类），也不会导致任何 AbilitySlice 出栈。

不同 Page 间导航

不同 Page 中的 AbilitySlice 相互不可见，因此无法通过 `present()`或 `presentForResult()`方法直接导航到其他 Page 的 AbilitySlice。AbilitySlice 作为 Page 的内部单元，以 Action 的形式对外暴露，因此可以通过配置 Intent 的 Action 导航到目标 AbilitySlice。Page 间的导航可以使用 `startAbility()`或 `startAbilityForResult()`方法，获得返回结果的回调为 `onAbilityResult()`。在 Ability 中调用 `setResult()`可以设置返回结果。详细用法可参考[根据 Operation 的其他属性启动应用](#)中的示例。

1.1.2.4 跨设备迁移

跨设备迁移（下文简称“迁移”）支持将 Page 在同一用户的不同设备间迁移，以便支持用户无缝切换的诉求。以 Page 从设备 A 迁移到设备 B 为例，迁移动作主要步骤如下：

1. 设备 A 上的 Page 请求迁移。
2. HarmonyOS 处理迁移任务，并回调设备 A 上 Page 的保存数据方法，用于保存迁移必须的数据。
3. HarmonyOS 在设备 B 上启动同一个 Page，并回调其恢复数据方法。

开发者可以参考以下详细步骤开发具有迁移功能的 Page。

实现 IAbilityContinuation 接口

- **onStartContinuation()**

Page 请求迁移后，系统首先回调此方法，开发者可以在此回调中决策当前是否可以执行迁移，比如，弹框让用户确认是否开始迁移。

- **onSaveData()**

如果 `onStartContinuation()` 返回 `true`，则系统回调此方法，开发者在此回调中保存必须传递到另外设备上以便恢复 Page 状态的数据。

- **`onRestoreData()`**

源侧设备上 Page 完成保存数据后，系统在目标侧设备上回调此方法，开发者在此回调中接受用于恢复 Page 状态的数据。注意，在目标侧设备上的 Page 会重新启动其生命周期，无论其启动模式如何配置。且系统回调此方法的时机在 `onStart()` 之前。

- **`onCompleteContinuation()`**

目标侧设备上恢复数据一旦完成，系统就会在源侧设备上回调 Page 的此方法，以便通知应用迁移流程已结束。开发者可以在此检查迁移结果是否成功，并在此处理迁移结束的动作，例如，应用可以在迁移完成后终止自身生命周期。

- **`onRemoteTerminated()`**

如果开发者使用 `continueAbilityReversibly()` 而不是 `continueAbility()`，则此后可以在源侧设备上使用 `reverseContinueAbility()` 进行回迁。这种场景下，相当于同一个 Page（的两个实例）同时在两个设备上运行，迁移完成后，如果目标侧设备上 Page 因任何原因终止，则源侧 Page 通过此回调接收终止通知。

说明

一个应用可能包含多个 Page，仅支持迁移的 Page 需要实现 `IAbilityContinuation` 接口。同时，此 Page 所包含的所有 `AbilitySlice` 也需要实现此接口。

请求迁移

实现 `IAbilityContinuation` 的 Page 可以在其生命周期内，调用 `continueAbility()` 或 `continueAbilityReversibly()` 请求迁移。两者的区别是，通过后者发起的迁移此后可以进行回迁。

```
1. try {
2.     continueAbility();
3. } catch (IllegalStateException e) {
```

```

4.     // Maybe another continuation in progress.
5.     ...
6. }

```

以 Page 从设备 A 迁移到设备 B 为例，详细的流程如下：

1. 设备 A 上的 Page 请求迁移。
2. 系统回调设备 A 上 Page 及其 AbilitySlice 栈中所有 AbilitySlice 实例的 `IAbilityContinuation.onStartContinuation()` 方法，以确认当前是否可以立即迁移。
3. 如果可以立即迁移，则系统回调设备 A 上 Page 及其 AbilitySlice 栈中所有 AbilitySlice 实例的 `IAbilityContinuation.onSaveData()` 方法，以便保存迁移后恢复状态必须的数据。
4. 如果保存数据成功，则系统在设备 B 上启动同一个 Page，并恢复 AbilitySlice 栈，然后回调 `IAbilityContinuation.onRestoreData()` 方法，传递此前保存的数据；此后设备 B 上此 Page 从 `onStart()` 开始其生命周期回调。
5. 系统回调设备 A 上 Page 及其 AbilitySlice 栈中所有 AbilitySlice 实例的 `IAbilityContinuation.onCompleteContinuation()` 方法，通知数据恢复成功与否。

请求回迁

使用 `continueAbilityReversibly()` 请求迁移并完成后，源侧设备上已迁移的 Page 可以发起回迁，以便使用户活动重新回到此设备。

```

1. try {
2.     reverseContinueAbility();
3. } catch (IllegalStateException e) {
4.     // Maybe another continuation in progress.
5.     ...
6. }

```

以 Page 从设备 A 迁移到设备 B 后并请求回迁为例，详细的流程如下：

1. 设备 A 上的 Page 请求回迁。
2. 系统回调设备 B 上 Page 及其 AbilitySlice 栈中所有 AbilitySlice 实例的 `IAbilityContinuation.onStartContinuation()` 方法，以确认当前是否可以立即迁移。

3. 如果可以立即迁移，则系统回调设备 B 上 Page 及其 AbilitySlice 栈中所有 AbilitySlice 实例的 `IAbilityContinuation.onSaveData()` 方法，以便保存回迁后恢复状态必须的数据。
4. 如果保存数据成功，则系统在设备 A 上 Page 恢复 AbilitySlice 栈，然后回调 `IAbilityContinuation.onRestoreData()` 方法，传递此前保存的数据。
5. 如果数据恢复成功，则系统终止设备 B 上 Page 的生命周期。

1.1.3 Service Ability

1.1.3.1 基本概念

基于 Service 模板的 Ability（以下简称“Service”）主要用于后台运行任务（如执行音乐播放、文件下载等），但不提供用户交互界面。Service 可由其他应用或 Ability 启动，即使用户切换到其他应用，Service 仍将在后台继续运行。

Service 是单实例的。在一个设备上，相同的 Service 只会存在一个实例。如果多个 Ability 共用这个实例，只有当与 Service 绑定的所有 Ability 都退出后，Service 才能够退出。由于 Service 是在主线程里执行的，因此，如果在 Service 里面的操作时间过长，开发者必须在 Service 里创建新的线程来处理（详见线程间通信），防止造成主线程阻塞，应用程序无响应。

1.1.3.2 创建 Service

介绍如何创建一个 Service。

1. 创建 Ability 的子类，实现 Service 相关的生命周期方法。Service 也是一种 Ability，Ability 为 Service 提供了以下生命周期方法，用户可以重写这些方法来添加自己的处理。
 - `onStart()`
该方法在创建 Service 的时候调用，用于 Service 的初始化，在 Service 的整个生命周期只会调用一次。

- `onCommand()`

在 Service 创建完成之后调用，该方法在客户端每次启动该 Service 时都会调用，用户可以在该方法中做一些调用统计、初始化类的操作。

- `onConnect()`

在 Ability 和 Service 连接时调用，该方法返回 `IRemoteObject` 对象，用户可以在该回调函数中生成对应 Service 的 IPC 通信通道，以便 Ability 与 Service 交互。Ability 可以多次连接同一个 Service，系统会缓存该 Service 的 IPC 通信对象，只有第一个客户端连接 Service 时，系统才会调用 Service 的 `onConnect` 方法来生成 `IRemoteObject` 对象，而后系统会将同一个 `RemoteObject` 对象传递至其他连接同一个 Service 的所有客户端，而无需再次调用 `onConnect` 方法。

- `onDisconnect()`

在 Ability 与绑定的 Service 断开连接时调用。

- `onStop()`

在 Service 销毁时调用。Service 应通过实现此方法来清理任何资源，如关闭线程、注册的侦听器等等。

2. 创建 Service 的代码示例如下：

```
1. public class ServiceAbility extends Ability {
2.     @Override
3.     public void onStart(Intent intent) {
4.         super.onStart(intent);
5.     }
6.
7.     @Override
8.     public void onCommand(Intent intent, boolean restart, int startId) {
9.         super.onCommand(intent, restart, startId);
10.    }
11.
12.    @Override
```

```
13. public IRemoteObject onConnect(Intent intent) {
14.     super.onConnect(intent);
15.     return null;
16. }
17.
18. @Override
19. public void onDisconnect(Intent intent) {
20.     super.onDisconnect(intent);
21. }
22.
23. @Override
24. public void onStop() {
25.     super.onStop();
26. }
27. }
```

3. 注册 Service。

Service 也需要在应用配置文件中进行注册，注册类型 type 需要设置为 service。

```
0. {
1.     "module": {
2.         "abilities": [
3.             {
4.                 "name": ".ServiceAbility",
5.                 "type": "service",
6.                 "visible": true
7.             }
8.         ]
9.     }
10.     ...
11. }
12. ...
13. }
```

1.1.3.3 启动 Service

介绍通过 `startAbility()` 启动 Service 以及对应的停止方法。

- 启动 Service

Ability 为开发者提供了 `startAbility()` 方法来启动另外一个 Ability。因为 Service 也是 Ability 的一种，开发者同样可以通过将 [Intent](#) 传递给该方法来启动 Service。不仅支持启动本地 Service，还支持启动远程 Service。

开发者可以通过构造包含 DeviceId、BundleName 与 AbilityName 的 Operation 对象来设置目标 Service 信息。这三个参数的含义如下：

- DeviceId：表示设备 ID。如果是本地设备，则可以直接留空；如果是远程设备，可以通过 `ohos.distributedschedule.interwork.DeviceManager` 提供的 `getDeviceList` 获取设备列表，详见《API 参考》。
- BundleName：表示包名称。
- AbilityName：表示待启动的 Ability 名称。

启动本地设备 Service 的代码示例如下：

```
1. Intent intent = new Intent();
2. Operation operation = new Intent.OperationBuilder()
3.     .withDeviceId("")
4.     .withBundleName("com.huawei.hiworld.himusic")
5.     .withAbilityName("com.huawei.hiworld.himusic.entry.ServiceAbility")
6.     .build();
7. intent.setOperation(operation);
8. startAbility(intent);
```

启动远程设备 Service 的代码示例如下：

```
1. Operation operation = new Intent.OperationBuilder()
2.     .withDeviceId("deviceId")
3.     .withBundleName("com.huawei.hiworld.himusic")
4.     .withAbilityName("com.huawei.hiworld.himusic.entry.ServiceAbility")
5.     .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE) // 设置支持分布式调度系统多设备启动的标识
6.     .build();
7. Intent intent = new Intent();
8. intent.setOperation(operation);
```

```
9. startAbility(intent);
```

执行上述代码后，Ability 将通过 startAbility() 方法来启动 Service。

- 如果 Service 尚未运行，则系统会先调用 onStart()来初始化 Service，再回调 Service 的 onCommand()方法来启动 Service。
- 如果 Service 正在运行，则系统会直接回调 Service 的 onCommand()方法来启动 Service。
- 停止 Service

Service 一旦创建就会一直保持在后台运行，除非必须回收内存资源，否则系统不会停止或销毁 Service。开发者可以在 Service 中通过 terminateAbility()停止本 Service 或在其他 Ability 调用 stopAbility()来停止 Service。

停止 Service 同样支持停止本地设备 Service 和停止远程设备 Service，使用方法与启动 Service 一样。一旦调用停止 Service 的方法，系统便会尽快销毁 Service。

1.1.3.4 连接 Service

如果 Service 需要与 Page Ability 或其他应用的 Service Ability 进行交互，则应创建用于连接的 Connection。Service 支持其他 Ability 通过 connectAbility()方法与其进行连接。

在使用 connectAbility()处理回调时，需要传入目标 Service 的 Intent 与 IAbilityConnection 的实例。IAbilityConnection 提供了两个方法供开发者实现：onAbilityConnectDone()用来处理连接的回调，onAbilityDisconnectDone()用来处理断开连接的回调。

连接 Service 的代码示例如下：

```
1. // 创建连接回调实例
2. private IAbilityConnection connection = new IAbilityConnection() {
3.     // 连接到 Service 的回调
4.     @Override
5.     public void onAbilityConnectDone(ElementName elementName, IRemoteObject iRemoteObject, int resultCode) {
```

```
6.         // 在这里开发者可以拿到服务端传过来 IRemoteObject 对象，从中解析出服务端传过来的信息
7.     }
8.
9.     // 断开与连接的回调
10.    @Override
11.    public void onAbilityDisconnectDone(ElementName elementName, int resultCode) {
12.    }
13. };
14. // 连接 Service
15. connectAbility(intent, connection);
```

同时，Service 侧也需要在 onConnect()时返回 IRemoteObject，从而定义与 Service 进行通信的接口。onConnect()需要返回一个 IRemoteObject 对象，HarmonyOS 提供了

IRemoteObject 的默认实现，用户可以通过继承 RemoteObject 来创建自定义的实现类。

Service 侧把自身的实例返回给调用侧的代码示例如下：

```
1. // 创建自定义 IRemoteObject 实现类
2. private class MyRemoteObject extends RemoteObject {
3.     public MyRemoteObject() {
4.         super("MyRemoteObject");
5.     }
6. }
7.
8. // 把 IRemoteObject 返回给客户端
9. @Override
10. protected IRemoteObject onConnect(Intent intent) {
11.     return new MyRemoteObject();
12. }
```

1.1.3.5 生命周期

与 Page 类似，Service 也拥有生命周期，如[图 1](#)所示。根据调用方法的不同，其生命周期有

以下两种路径：

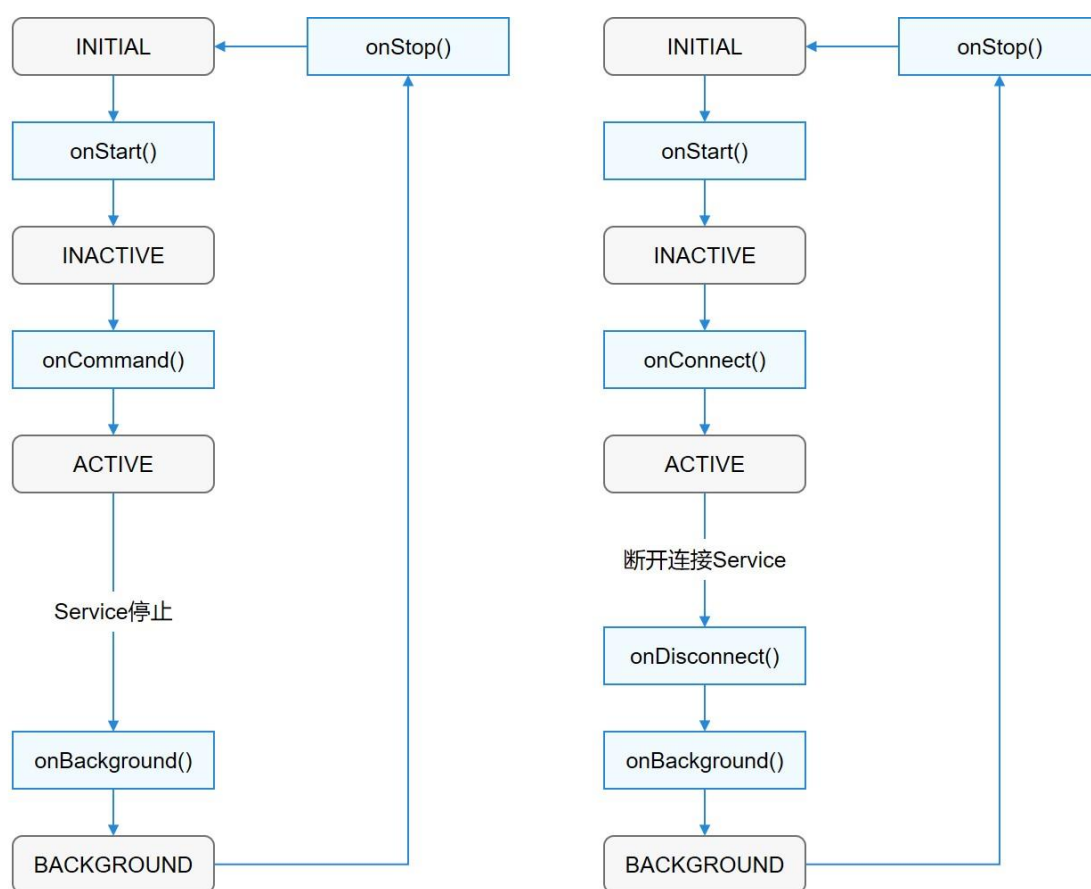
- 启动 Service

该 Service 在其他 Ability 调用 startAbility()时创建，然后保持运行。其他 Ability 通过调用 stopAbility()来停止 Service，Service 停止后，系统会将其销毁。

- 连接 Service

该 Service 在其他 Ability 调用 connectAbility()时创建，客户端可通过调用 disconnectAbility()断开连接。多个客户端可以绑定到相同 Service，而且当所有绑定全部取消后，系统即会销毁该 Service。

图 1 Service 生命周期



1.1.3.6 前台 Service

一般情况下，Service 都是在后台运行的，后台 Service 的优先级都是比较低的，当资源不足时，系统有可能回收正在运行的后台 Service。

在一些场景下（如播放音乐），用户希望应用能够一直保持运行，此时就需要使用前台

Service。前台 Service 会始终保持正在运行的图标在系统状态栏显示。

使用前台 Service 并不复杂，开发者只需在 Service 创建的方法里，调用

`keepBackgroundRunning()`将 Service 与通知绑定。调用 `keepBackgroundRunning()`方法

前需要在配置文件中声明 `ohos.permission.KEEP_BACKGROUND_RUNNING` 权限，该权限

是 normal 级别，同时还需要在配置文件中添加对应的 `backgroundModes` 参数。在

`onStop()`方法中调用 `cancelBackgroundRunning()`方法可停止前台 Service。

使用前台 Service 的 `onStart()`代码示例如下：

```
1. // 创建通知，其中 1005 为 notificationId
2. NotificationRequest request = new NotificationRequest(1005);
3. NotificationRequest.NotificationNormalContent content = new
   NotificationRequest.NotificationNormalContent();
4. content.setTitle("title").setText("text");
5. NotificationRequest.NotificationContent notificationContent = new
   NotificationRequest.NotificationContent(content);
6. request.setContent(notificationContent);
7.
8. // 绑定通知，1005 为创建通知时传入的 notificationId
9. keepBackgroundRunning(1005, request);
```

在配置文件中配置如下：

```
1. {
2.     "name": ".ServiceAbility",
3.     "type": "service",
4.     "visible": true,
5.     "backgroundModes": ["dataTransfer","location"]
6. }
```

1.1.4 Data Ability

1.1.4.1 基本概念

使用 Data 模板的 Ability（以下简称“Data”）有助于应用管理其自身和其他应用存储数据的访问，并提供与其他应用共享数据的方法。Data 既可用于同设备不同应用的数据共享，也支持跨设备不同应用的数据共享。

数据的存放形式多样，可以是数据库，也可以是磁盘上的文件。Data 对外提供对数据的增、删、改、查，以及打开文件等接口，这些接口的具体实现由开发者提供。

URI 介绍

Data 的提供方和使用方都通过 URI（Uniform Resource Identifier）来标识一个具体的数据，例如数据库中的某个表或磁盘上的某个文件。HarmonyOS 的 URI 仍基于 URI 通用标准，格式如下：

- scheme: 协议方案名，固定为“dataability”，代表 Data Ability 所使用的协议类型。
- authority: 设备 ID，如果为跨设备场景，则为目的设备的 IP 地址；如果为本地设备场景，则不需要填写。
- path: 资源的路径信息，代表特定资源的位置信息。
- query: 查询参数。
- fragment: 可以用于指示要访问的子资源。

URI 示例：

- 跨设备场景：dataability://device_id/com.huawei.dataability.persondata/person/10
- 本地设备：dataability:///com.huawei.dataability.persondata/person/10

1.1.4.2 访问 Data

开发者可以通过 `DataAbilityHelper` 类来访问当前应用或其他应用提供的共享数据。

`DataAbilityHelper` 作为客户端，与提供方的 `Data` 进行通信。`Data` 接收到请求后，执行相应的处理，并返回结果。`DataAbilityHelper` 提供了一系列与 `Data Ability` 对应的方法。

下面介绍 `DataAbilityHelper` 具体的使用步骤。

声明使用权限

如果待访问的 `Data` 声明了访问需要权限，则访问此 `Data` 需要在配置文件中声明需要此权限。

声明请参考[权限申请字段说明](#)。

```
1. "reqPermissions": [  
2.   {  
3.     "name": "com.example.myapplication5.DataAbility.DATA"  
4.   }  
5. ]
```

创建 `DataAbilityHelper`

`DataAbilityHelper` 为开发者提供了 `creator()`方法来创建 `DataAbilityHelper` 实例。该方法为静态方法，有多个重载。最常见的方法是通过传入一个 `context` 对象来创建 `DataAbilityHelper` 对象。

获取 helper 对象示例：

```
1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
```

访问 Data Ability

DataAbilityHelper 为开发者提供了一系列的接口来访问不同类型的数据（文件、数据库等）。

- **访问文件**

DataAbilityHelper 为开发者提供了 FileDescriptor openFile(Uri uri, String mode)方法来操作文件。此方法需要传入两个参数，其中 uri 用来确定目标资源路径，mode 用来指定打开文件的方式，可选方式包含 “r”（读），“w”（写），“rw”（读写），“wt”（覆盖写），“wa”（追加写），“rwt”（覆盖写且可读）。

该方法返回一个目标文件的 FD（文件描述符），把文件描述符封装成流，开发者就可以对文件流进行自定义处理。

访问文件示例：

```
1. // 读取文件描述符
2. FileDescriptor fd = helper.openFile(uri, "r");
3. FileInputStream fis = new FileInputStream(fd);
```

- **访问数据库**

DataAbilityHelper 为开发者提供了增、删、改、查以及批量处理等方法来操作数据库。

方法	描述
ResultSet query(Uri uri, String[] columns, DataAbilityPredicates predicates)	查询数据库
int insert(Uri uri, ValuesBucket value)	向数据库中插入单条数据
int batchInsert(Uri uri, ValuesBucket[] values)	向数据库中插入多条数据
int delete(Uri uri, DataAbilityPredicates predicates)	删除一条或多条数据
int update(Uri uri, ValuesBucket value, DataAbilityPredicates predicates)	更新数据库
DataAbilityResult[] executeBatch(ArrayList<DataAbilityOperation> operations)	批量操作数据库

这些方法的使用说明如下：

- query()

查询方法，其中 uri 为目标资源路径，columns 为想要查询的字段。开发者的查询条件可以通过 DataAbilityPredicates 来构建。查询用户表中 id 在 101-103 之间的用户，并把结果打印出来，代码示例如下：

```
1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
2.
3. // 构造查询条件
4. DataAbilityPredicates predicates = new DataAbilityPredicates();
5. predicates.between("userId", 101, 103);
6.
7. // 进行查询
8. ResultSet resultSet = helper.query(uri, columns, predicates);
9.
10. // 处理结果
11. resultSet.goToFirstRow();
12. do{
13.     // 在此处理 ResultSet 中的记录;
14. }while(resultSet.goToNextRow());
```

- insert()

新增方法，其中 uri 为目标资源路径，ValuesBucket 为要新增的对象。插入一条用户信息的代码示例如下：

```
1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
2.
3. // 构造插入数据
4. ValuesBucket valuesBucket = new ValuesBucket();
5. valuesBucket.putString("name", "Tom");
6. valuesBucket.putInteger("age", 12);
7. helper.insert(uri, valuesBucket);
```

- batchInsert()

批量插入方法，和 insert()类似。批量插入用户信息的代码示例如下：

```
1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
2.
```

```

3. // 构造插入数据
4. ValuesBucket[] values = new ValuesBucket[2];
5. value[0] = new ValuesBucket();
6. value[0].putString("name", "Tom");
7. value[0].putInteger("age", 12);
8. value[1] = new ValuesBucket();
9. value[1].putString("name", "Tom1");
10. value[1].putInteger("age", 16);
11. helper.batchInsert(uri, values);

```

- delete()

删除方法，其中删除条件可以通过 `DataAbilityPredicates` 来构建。删除用户表中 id 在 101-

103 之间的用户，代码示例如下：

```

1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
2.
3. // 构造删除条件
4. DataAbilityPredicates predicates = new DataAbilityPredicates();
5. predicates.between("userId", 101,103);
6. helper.delete(uri,predicates);

```

- update()

更新方法，更新数据由 `ValuesBucket` 传入，更新条件由 `DataAbilityPredicates` 来构建。更

新 id 为 102 的用户，代码示例如下：

```

1. DataAbilityHelper helper = DataAbilityHelper.creator(this);
2.
3. // 构造更新条件
4. DataAbilityPredicates predicates = new DataAbilityPredicates();
5. predicates.equalTo("userId",102);
6.
7. // 构造更新数据
8. ValuesBucket valuesBucket = new ValuesBucket();
9. valuesBucket.putString("name", "Tom");
10. valuesBucket.putInteger("age", 12);
11. helper.update(uri, valuesBucket, predicates);

```

- executeBatch()

此方法用来执行批量操作。DataAbilityOperation 中提供了设置操作类型、数据和操作条件的方法，开发者可自行设置自己要执行的数据库操作。插入多条数据的代码示例如下：

```

1. DataAbilityHelper helper = DataAbilityHelper.creator(abilityObj, insertUri);
2.
3. // 构造批量操作
4. ValuesBucket value1 = initSingleValue();
5. DataAbilityOperation opt1 = DataAbilityOperation.newInsertBuilder(insertUri).withValuesBucket(value1).build();
6. ValuesBucket value2 = initSingleValue2();
7. DataAbilityOperation opt2 = DataAbilityOperation.newInsertBuilder(insertUri).withValuesBucket(value2).build();
8. ArrayList<DataAbilityOperation> operations = new ArrayList<DataAbilityOperation>();
9. operations.add(opt1);
10. operations.add(opt2);
11. DataAbilityResult[] result = helper.executeBatch(insertUri, operations);
    
```

1.1.4.3 创建 Data

使用 Data 模板的 Ability 形式仍然是 Ability，因此，开发者需要为应用添加一个或多个 Ability 的子类，来提供程序与其他应用之间的接口。Data 为结构化数据和文件提供了不同 API 接口供用户使用，因此，开发者需要首先确定好使用何种类型的数据。本章节主要讲述了创建 Data 的基本步骤和需要使用的接口。

确定数据存储方式

确定数据的存储方式，Data 支持以下两种数据形式：

- 文件数据：如文本、图片、音乐等。
- 结构化数据：如数据库等。

实现 UserDataAbility

UserDataAbility 接收其他应用发送的请求，提供外部程序访问的入口，从而实现应用间的数据访问。Data 提供了文件存储和数据库存储两组接口供用户使用。

文件存储

开发者需要在 Data 中重写 FileDescriptor openFile(Uri uri, String mode)方法来操作文件：

uri 为客户端传入的请求目标路径；mode 为开发者对文件的操作选项，可选方式包含

“r”（读），“w”（写），“rw”（读写）等。

MessageParcel 类提供了一个静态方法，用于获取 MessageParcel 实例。通过

dupFileDescriptor()函数复制待操作文件流的文件描述符，并将其返回，供远端应用使用。

示例：根据传入的 uri 打开对应的文件

```

1. // 创建 messageParcel
2. MessageParcel messageParcel = MessageParcel.obtain();
3. File file = new File(uri.getDecodedPathList().get(1));
4. if (mode == null || !"rw".equals(mode)) {
5.     file.setReadOnly();
6. }
7. FileInputStream fileIs = new FileInputStream(file);
8. FileDescriptor fd = fileIs.getFD();
9.
10. // 绑定文件描述符
11. return messageParcel.dupFileDescriptor(fd);
    
```

数据库存储

1. 初始化数据库连接。

系统会在应用启动时调用 onStart()方法创建 Data 实例。在此方法中，开发者应该创建数据库连接，并获取连接对象，以便后续和数据库进行操作。为了避免影响应用启动速度，开发者应当尽可能将非必要的耗时任务推迟到使用时执行，而不是在此方法中执行所有初始化。

示例：初始化的时候连接数据库

```

1. private static final String DATABASE_NAME = "UserDataAbility.db";
2. private static final String DATABASE_NAME_ALIAS = "UserDataAbility";
3. private OrmContext ormContext = null;
4.
5. @Override
6. public void onStart(Intent intent) {
7.     super.onStart(intent);
8.     DatabaseHelper manager = new DatabaseHelper(this);
9.     ormContext = manager.getOrmContext(DATABASE_NAME_ALIAS, DATABASE_NAME, BookStore.class);
10. }
    
```

2. 编写数据库操作方法。

Ability 定义了 6 个方法供用户处理对数据库表数据的增删改查。这 6 个方法在 Ability 中已默认实现，开发者可按需重写。

方法	描述
ResultSet query(Uri uri, String[] columns, DataAbilityPredicates predicates)	查询数据库
int insert(Uri uri, ValuesBucket value)	向数据库中插入单条数据
int batchInsert(Uri uri, ValuesBucket[] values)	向数据库中插入多条数据
int delete(Uri uri, DataAbilityPredicates predicates)	删除一条或多条数据
int update(Uri uri, ValuesBucket value, DataAbilityPredicates predicates)	更新数据库
DataAbilityResult[] executeBatch(ArrayList<DataAbilityOperation> operations)	批量操作数据库

这些方法的使用说明如下：

- query()

该方法接收三个参数，分别是查询的目标路径，查询的列名，以及查询条件，查询条件由类

DataAbilityPredicates 构建。根据传入的列名和查询条件查询用户表的代码示例如下：

```

1. public ResultSet query(Uri uri, String[] columns, DataAbilityPredicates predicates) {
2.     if (ormContext == null) {
3.         HiLog.error(this.getClass().getSimpleName(), "failed to query, ormContext is null");
    
```

```
4.     return null;
5. }
6.
7. // 查询数据库
8. OrmPredicates ormPredicates = DataAbilityUtils.createOrmPredicates(predicates, User.class);
9. ResultSet resultSet = ormContext.query(ormPredicates, columns);
10. if (resultSet == null) {
11.     HiLog.info(this.getClass(), "resultSet is null");
12. }
13.
14. // 返回结果
15. return resultSet;
16. }
```

- insert()

该方法接收两个参数，分别是插入的目标路径和插入的数据值。其中，插入的数据由 ValuesBucket 封装，服务端可以从该参数中解析出对应的属性，然后插入到数据库中。此方法返回一个 int 类型的值用于标识结果。接收到传过来的用户信息并把它保存到数据库中的代码示例如下：

```
1. public int insert(Uri uri, ValuesBucket value) {
2.     // 参数校验
3.     if (ormContext == null) {
4.         HiLog.error(this.getClass().getSimpleName(), "failed to insert, ormContext is null");
5.         return -1;
6.     }
7.     String path = uri.getPath();
8.
9.     if (buildPathMatcher().getPathId(path) != PathId) {
10.        HiLog.info(this.getClass(), "UserDataAbility insert path is not matched");
11.        return -1;
12.    }
13.
14.    // 构造插入数据
15.    User user = new User();
16.    user.setUserId(value.getInteger("userId"));
17.    user.setFirstName(value.getString("firstName"));
```



```
18.     user.setLastName(value.getString("lastName"));
19.     user.setAge(value.getInteger("age"));
20.     user.setBalance(value.getDouble("balance"));
21.
22.     // 插入数据库
23.     boolean isSucceeded = true;
24.     try {
25.         isSucceeded = ormContext.insert(user);
26.     } catch (DataAbilityRemoteException e) {
27.         HiLog.error(TAG, "insert fail: " + e.getMessage());
28.         throw new RuntimeException(e);
29.     }
30.     if (!isSucceeded) {
31.         HiLog.error(this.getClass().getSimpleName(), "failed to insert");
32.         return -1;
33.     }
34.     isSucceeded = ormContext.flush();
35.     if (!isSucceeded) {
36.         HiLog.error(this.getClass().getSimpleName(), "failed to insert flush");
37.         return -1;
38.     }
39.     DataAbilityHelper.creator(this, uri).notifyChange(uri);
40.     int id = Math.toIntExact(user.getRowId());
41.     return id;
42. }
```

- batchInsert()

该方法为批量插入方法，接收一个 ValuesBucket 数组用于单次插入一组对象。它的作用是提高插入多条重复数据的效率。该方法系统已实现，开发者可以直接调用。

- delete()

该方法用来执行删除操作。删除条件由类 DataAbilityPredicates 构建，服务端在接收到该参数之后可以从中解析出要删除的数据，然后到数据库中执行。根据传入的条件删除用户表数据的代码示例如下：

```
1. public int delete(Uri uri, DataAbilityPredicates predicates) {
2.     if (ormContext == null) {
```

```

3.     HiLog.error(this.getClass().getSimpleName(), "failed to delete, ormContext is null");
4.     return -1;
5. }
6.
7.     OrmPredicates ormPredicates = DataAbilityUtils.createOrmPredicates(predicates, User.class);
8.     int value = ormContext.delete(ormPredicates);
9.     DataAbilityHelper.creator(this, uri).notifyChange(uri);
10.    return value;
11. }
    
```

- update()

此方法用来执行更新操作。用户可以在 ValuesBucket 参数中指定要更新的数据，在

DataAbilityPredicates 中构建更新的条件等。更新用户表的数据的代码示例如下：

```

1. public int update(Uri uri, ValuesBucket value, DataAbilityPredicates predicates) {
2.     if (ormContext == null) {
3.         HiLog.error(this.getClass().getSimpleName(), "failed to update, ormContext is null");
4.         return -1;
5.     }
6.
7.     OrmPredicates ormPredicates = DataAbilityUtils.createOrmPredicates(predicates, User.class);
8.     int index = ormContext.update(ormPredicates, value);
9.     HiLog.info(this.getClass(), "UserDataAbility update value:" + index);
10.    DataAbilityHelper.creator(this, uri).notifyChange(uri);
11.    return index;
12. }
    
```

- executeBatch()

此方法用来批量执行操作。DataAbilityOperation 中提供了设置操作类型、数据和操作条件的方法，用户可自行设置自己要执行的数据库操作。该方法系统已实现，开发者可以直接调用。

注册 UserDataAbility

和 Service 类似，开发者必须在配置配置文件中注册 Data。并且配置以下属性：

- type: 类型设置为 data
- uri: 对外提供的访问路径，全局唯一

- permissions: 访问该 data ability 时需要申请的访问权限

```

1. {
2.     "name": ".UserDataAbility",
3.     "type": "data",
4.     "visible": true,
5.     "uri": "dataability://com.example.myapplication5.DataAbilityTest",
6.     "permissions": [
7.         "com.example.myapplication5.DataAbility.DATA"
8.     ]
9. }
    
```

1.1.5 Intent

1.1.5.1 基本概念

Intent 是对象之间传递信息的载体。例如，当一个 Ability 需要启动另一个 Ability 时，或者一个 AbilitySlice 需要导航到另一个 AbilitySlice 时，可以通过 Intent 指定启动的目标同时携带相关数据。Intent 的构成元素包括 Operation 与 Parameters，具体描述参见表 1。

属性	子属性	描述
Operation	Action	表示动作，通常使用系统预置 Action，应用也可以自定义 Action。例如 IntentConstants.ACTION_HOME 表示返回桌面动作。
	Entity	表示类别，通常使用系统预置 Entity，应用也可以自定义 Entity。例如 Intent.ENTITY_HOME 表示在桌面显示图标。
	Uri	表示 Uri 描述。如果在 Intent 中指定了 Uri，则 Intent 将匹配指定的 Uri 信息，包括 scheme, schemeSpecificPart, authority 和 path 信息。
	Flags	表示处理 Intent 的方式。例如 Intent.FLAG_ABILITY_CONTINUATION 标记在本地一个 Ability 是否可以迁移到远端设备继续运行。

属性	子属性	描述
	BundleName	表示包描述。如果在 Intent 中同时指定了 BundleName 和 AbilityName，则 Intent 可以直接匹配到指定的 Ability。
	AbilityName	表示待启动的 Ability 名称。如果在 Intent 中同时指定了 BundleName 和 AbilityName，则 Intent 可以直接匹配到指定的 Ability。
	Deviceld	表示运行指定 Ability 的设备 ID。
Parameters	-	Parameters 是一种支持自定义的数据结构，开发者可以通过 Parameters 传递某些请求所需的信息。

表 1 Intent 的构成元素

当 Intent 用于发起请求时，根据指定元素的不同，分为两种类型：

- 如果同时指定了 BundleName 与 AbilityName，则根据 Ability 的全称（例如，“com.demoapp.FooAbility”）来直接启动应用。
- 如果未同时指定 BundleName 和 AbilityName，则根据 Operation 中的其他属性来启动应用。

1.1.5.2 根据 Ability 的全称启动应用

通过构造包含 BundleName 与 AbilityName 的 Operation 对象，可以启动一个 Ability、并

导航到该 Ability。示例代码如下：

```

1. Intent intent = new Intent();
2.
3. // 通过 Intent 中的 OperationBuilder 类构造 operation 对象，指定设备标识（空串表示当前设备）、应用包名、Ability 名称
4. Operation operation = new Intent.OperationBuilder()
5.     .withDeviceld("")
6.     .withBundleName("com.demoapp")
7.     .withAbilityName("com.demoapp.FooAbility")
8.     .build();
9.
10. // 把 operation 设置到 intent 中
    
```

```
11. intent.setOperation(operation);
12. startAbility(intent);
```

作为处理请求的对象，会在相应的回调方法中接收请求方传递的 Intent 对象。以导航到另一个 Ability 为例，导航的目标 Ability 可以在其 onStart()回调的参数中获得 Intent 对象。

1.1.5.3 根据 Operation 的其他属性启动应用

有些场景下，开发者需要在应用中使用其他应用提供的某种能力，而不感知提供该能力的具体是哪一个应用。例如开发者需要通过浏览器打开一个链接，而不关心用户最终选择哪一个浏览器应用，则可以通过 Operation 的其他属性（除 BundleName 与 AbilityName 之外的属性）描述需要的能力。如果设备上存在多个应用提供同种能力，系统则弹出候选列表，由用户选择由哪个应用处理请求。以下示例展示使用 Intent 跨 Ability 查询天气信息。

请求方

在 Ability 中构造 Intent 以及包含 Action 的 Operation 对象，并调用 startAbilityForResult() 方法发起请求。然后重写 onAbilityResult()回调方法，对请求结果进行处理。

```
1. private void queryWeather() {
2.     Intent intent = new Intent();
3.     Operation operation = new Intent.OperationBuilder()
4.         .withAction(Intent.ACTION_QUERY_WEATHER)
5.         .build();
6.     intent.setOperation(operation);
7.     startAbilityForResult(intent, REQ_CODE_QUERY_WEATHER);
8. }
9.
10. @Override
11. protected void onAbilityResult(int requestCode, int resultCode, Intent resultData) {
12.     switch (requestCode) {
13.         case REQ_CODE_QUERY_WEATHER:
```

```

14.         // Do something with result.
15.         ...
16.         return;
17.     default:
18.         ...
19.     }
20. }
    
```

处理方

1. 作为处理请求的对象，首先需要在配置文件中声明对外提供的能力，以便系统据此找到自身并作为候选的请求处理者。

```

1. {
2.     "module": {
3.         ...
4.         "abilities": [
5.             {
6.                 ...
7.                 "skills": [
8.                     {
9.                         "actions": [
10.                            "ability.intent.QUERY_WEATHER"
11.                        ]
12.                    }
13.                ]
14.            }
15.        ]
16.    }
17.    ...
18. }
19. ...
20. }
    
```

2. 在 Ability 中配置路由以便支持以此 action 导航到对应的 AbilitySlice。

```

1. @Override
2. protected void onStart(Intent intent) {
3.     ...
4.     addActionRoute(Intent.ACTION_QUERY_WEATHER, DemoSlice.class.getName());
5.     ...
6. }
    
```

3. 在 Ability 中处理请求，并调用 setResult()方法暂存返回结果。

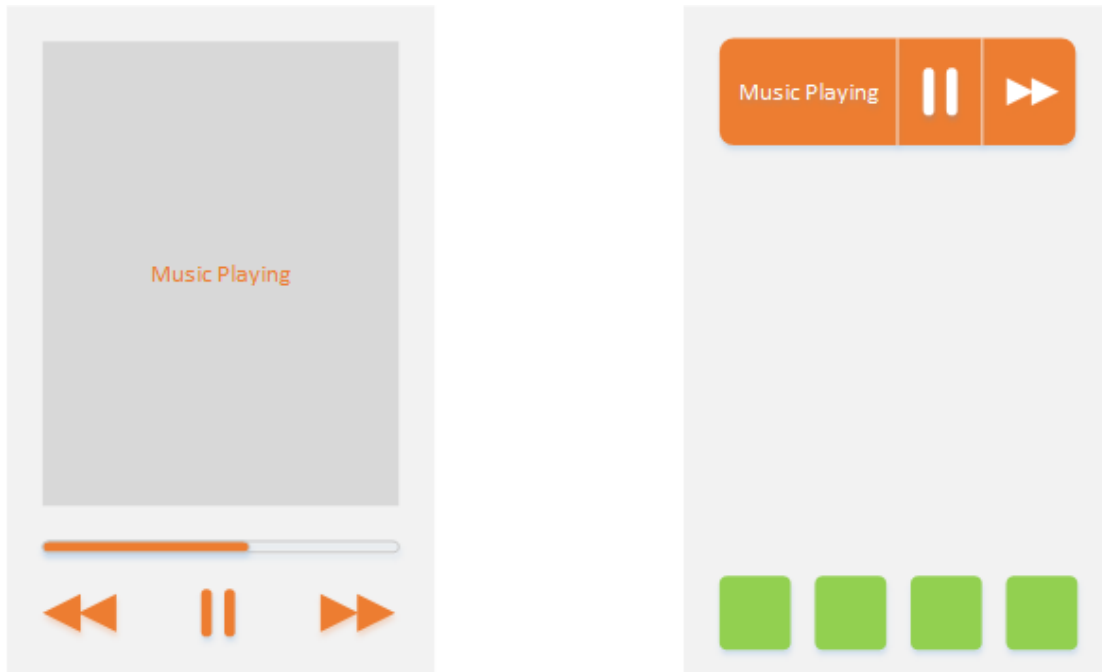
```
1. @Override
2. protected void onActive() {
3.     ...
4.     Intent resultIntent = new Intent();
5.     setResult(0, resultIntent);
6.     ...
7. }
```

1.1.6 Ability Form

1.1.6.1 基本概念

Ability Form，即表单，是 Page 形态的 Ability 的一种界面展示形式，用于嵌入到其他应用中作为其界面的一部分显示，并支持基础的交互功能。表单使用方作为表单展示的宿主负责显示表单，表单使用方的典型应用就是桌面。下图展示一种音乐播放应用 Page 的完整显示及其微缩展示效果。

图 1 Page 及其表单



1.1.6.2 表单提供方

表单提供方是一个 Page 形态的 Ability，需要实现 onCreateForm()方法，并返回一个

AbilityForm 对象。创建 AbilityForm 对象时需要指定表单布局文件。

1. 为表单定义布局文件。创建布局文件 *form_layout.xml*:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionLayout xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:width="match_parent"
4.     ohos:height="match_parent"
5.     ohos:orientation="horizontal">
6.     <Text
7.         ohos:id="+$+id:text01"
8.         ohos:width="match_content"
9.         ohos:height="match_content"
10.        ohos:text="Counter"
11.        ohos:text_color="#FF555555"
12.        ohos:text_size="20fp"/>
13.     <Text
14.         ohos:id="+$+id:text02"
15.         ohos:width="match_content"
16.         ohos:height="match_content"
    
```



```
17.         ohos:text_color="#FF000FF"  
18.         ohos:text_size="20fp"/>  
19. </DirectionLayout>
```

2. 实现 onCreateForm()方法，并为表单视图控件注册回调。

```
1. public class FormAbility extends Ability {  
2.     private static AbilityForm abilityForm;  
3.     private static int clickTimes = 0;  
4.  
5.     ...  
6.     @Override  
7.     public AbilityForm onCreateForm() {  
8.         abilityForm = new AbilityForm(ResourceTable.Layout_form_layout, this);  
9.  
10.        abilityForm.setText(ResourceTable.Id_text02, generateFormText());  
11.        abilityForm.registerViewListener(ResourceTable.Id_text02, new OnClickListener() {  
12.            @Override  
13.            public void onClick(int viewId, AbilityForm form, ViewsStatus viewsStatus) {  
14.                clickTimes++;  
15.                form.setText(viewId, generateFormText());  
16.                if (FormSlice.text != null) {  
17.                    FormSlice.text.setText("Client.Counter: " + clickTimes);  
18.                }  
19.            }  
20.        });  
21.        return abilityForm;  
22.    }  
23.  
24.     private static String generateFormText() {  
25.         return "total: " + clickTimes;  
26.     }  
27. }
```

3. 在 FormSlice 实现 text，用于展示 Form 点击效果（用例展示效果，非实现 Form 必须步骤）。

```
1. public class FormSlice extends AbilitySlice {  
2.     public static Text text;  
3.  
4.     @Override  
5.     public void onStart(Intent intent) {
```

```
6.     super.onStart(intent);
7.
8.     PositionLayout positionLayout = new PositionLayout(this);
9.
10.    ShapeElement background = new ShapeElement();
11.    background.setShape(ShapeElement.RECTANGLE);
12.    background.setRgbColor(new RgbColor(0xFFFFFFFF));
13.    positionLayout.setBackground(background);
14.
15.    text = new Text(this);
16.    text.setTextSize(30);
17.    text.setTop(400);
18.    text.setLeft(200);
19.    text.setText(assembleText(FormAbility.getClickTimes()));
20.    positionLayout.addComponent(text);
21.
22.    super.setUIContent(positionLayout);
23. }
24. ...
25. }
```

4. 表单提供方在配置文件中声明，将 form-enabled 设置为 true，并提供表单尺寸信息。

```
1. {
2.     "module": {
3.         ...
4.         "abilities": [
5.             {
6.                 ...
7.                 "form-enabled": true,
8.                 "form": {
9.                     "default-height": 200,
10.                    "default-width": 300,
11.                    "min-height": 60,
12.                    "min-width": 80
13.                },
14.                ...
15.            }
16.        ]
17.    }
18. }
```

```
18.     }
19.     ...
20. }
```

说明

由于当前暂未支持 Form 缩放，即无法调整其显示尺寸，因此 min-height 和 min-width 字段暂无实际用途，可以省略。

1.1.6.3 表单使用方

表单使用方通常是桌面类应用。以下示例展示如何获取并展示表单。

1. 获取表单需要具有 ohos.permission.REQUIRE_FORM 权限，注意该权限仅限系统应用获取。在配置文件中声明需要此权限。

```
1. "reqPermissions": [
2.     {
3.         "name": "ohos.permission.REQUIRE_FORM"
4.     }
5. ]
```

2. 调用 AbilitySlice 类的 acquireAbilityFormAsync()方法异步获取表单，该方法需要通过 Intent 指定获取的目标表单，并提供一个回调用于接收表单。注意，获取到 AbilityForm 实例并未立即显示到当前页面布局中，开发者需要继续后续步骤把表单添加到视图中才会显示。

```
1.     private void acquireForm() throws RemoteException {
2.         Intent intent = new Intent();
3.         Operation operation = new Intent.OperationBuilder()
4.             .withDevicelId("")
5.             .withBundleName("ohos.formsupplier.ability")
6.             .withAbilityName("ohos.formsupplier.ability.FormAbility")
7.             .build();
8.         intent.setOperation(operation);
9.
10.        // Get form size.
11.        formAbilityInfo =
12.        getBundleManager().queryAbilityByIntent(intent,IBundleManager.GET_ABILITY_INFO_WITH_PERMISSION,userId).get(0);
13.
14.        this.acquireAbilityFormAsync(intent, new AbilityForm.OnAcquiredCallback() {
15.            @Override
16.            public void onAcquired(AbilityForm abilityForm) {
```

```

16.         // store as a class field
17.         form = abilityForm;
18.     }
19.
20.     @Override
21.     public void onDestroyed(AbilityForm abilityForm) {
22.         form = null;
23.     }
24. });
25. }
    
```

3. 创建布局，作为表单的视图容器。

```

1.     private PositionLayout createFormHostLayout() {
2.         PositionLayout formLayout = ...;
3.
4.         ...
5.         // set size same with default of form
6.         formLayout.setHeight(formAbilityInfo.getDefaultFormHeight());
7.         formLayout.setWidth(formAbilityInfo.getDefaultFormWidth());
8.         ...
9.
10.        return formLayout;
11.    }
    
```

4. 将表单的视图容器添加到当前视图中，以便显示表单。

```

1.     private void showForm() {
2.         PositionLayout hostLayout = createFormHostLayout();
3.
4.         // attach form to host layout
5.         hostLayout.addComponent(form.getView());
6.
7.         // attach host layout to root layout
8.         rootLayout.addComponent(hostLayout);
9.     }
    
```

如果开发者需要移除表单，调用 AbilitySlice 类的 releaseAbilityForm()方法，并以此前获取的 AbilityForm 对象作为参数。

1.2 分布式任务调度

1.2.1 概述

在 HarmonyOS 中，分布式任务调度平台对搭载 HarmonyOS 的多设备构筑的“超级虚拟终端”提供统一的组件管理能力，为应用定义统一的能力基线、接口形式、数据结构、服务描述语言，屏蔽硬件差异；支持远程启动、远程调用、业务无缝迁移等分布式任务。

分布式任务调度平台在底层实现 [Ability](#)（分布式任务调度的基本组件）跨设备的启动/关闭、连接及断开连接以及迁移等能力，实现跨设备的组件管理：

- 启动和关闭：向开发者提供管理远程 Ability 的能力，即支持启动 Page 模板的 Ability，以及启动、关闭 Service 和 Data 模板的 Ability。
- 连接和断开连接：向开发者提供跨设备控制服务（Service 和 Data 模板的 Ability）的能力，开发者可以通过与远程服务连接及断开连接实现获取或注销跨设备管理服务的对象，达到和本地一致的服务调度。
- 迁移能力：向开发者提供跨设备业务的无缝迁移能力，开发者可以通过调用 Page 模板 Ability 的迁移接口，将本地业务无缝迁移到指定设备中，打通设备间壁垒。

1.2.1.1 约束与限制

- 开发者需要在 Intent 中设置支持分布式的标记（例如：Intent.FLAG_ABILITYSLICE_MULTI_DEVICE 表示该应用支持分布式调度），否则将无法获得分布式能力。
- 开发者通过在 config.json 中添加分布式数据传输的权限申请：{"name": "ohos.permission.servicebus.ACCESS_SERVICE"}，获取跨设备连接的能力。
- PA（Particle Ability，Service 和 Data 模板的 Ability）的调用支持连接及断开连接、启动及关闭这四类行为，在进行调度时：
 - 开发者必须在 Intent 中指定 PA 对应的 bundleName 和 abilityName。
 - 当开发者需要跨设备启动、关闭或连接 PA 时，需要在 Intent 中指定对端设备的 deviceId。开发者可通过如设备管理类 DeviceManager 提供的 getDeviceList 获取指定条件下匿名化处理的设备列表，实现对指定设备 PA 的启动/关闭以及连接管理。
- FA（Feature Ability，Page 模板的 Ability）的调用支持启动和迁移行为，在进行调度时：
 - 当启动 FA 时，需要开发者在 Intent 中指定对端设备的 deviceId、bundleName 和 abilityName。
 - FA 的迁移实现相同 bundleName 和 abilityName 的 FA 跨设备迁移，因此需要指定迁移设备的 deviceId。

1.2.2 开发指导

1.2.2.1 场景介绍

开发者在应用中集成分布式调度能力，通过调用指定能力的分布式接口，实现跨设备能力调度。根据 Ability 模板及意图的不同，分布式任务调度向开发者提供以下六种能力：启动远程 FA、启动远程 PA、关闭远程 PA、连接远程 PA、断开连接远程 PA 和 FA 跨设备迁移。下面以设备 A（本地设备）和设备 B（远端设备）为例，进行场景介绍：

1. 设备 A 启动设备 B 的 FA：在设备 A 上通过本地应用提供的启动按钮，启动设备 B 上对应的 FA。例如：设备 A 控制设备 B 打开相册，只需开发者在启动 FA 时指定打开相册的意图即可。
2. 设备 A 启动设备 B 的 PA：在设备 A 上通过本地应用提供的启动按钮，启动设备 B 上指定的 PA。例如：开发者在启动远程服务时通过意图指定音乐播放服务，即可实现设备 A 启动设备 B 音乐播放的能力。
3. 设备 A 关闭设备 B 的 PA：在设备 A 上通过本地应用提供的关闭按钮，关闭设备 B 上指定的 PA。类似启动的过程，开发者在关闭远程服务时通过意图指定音乐播放服务，即可实现关闭设备 B 上该服务的能力。
4. 设备 A 连接设备 B 的 PA：在设备 A 上通过本地应用提供的连接按钮，连接设备 B 上指定的 PA。连接后，通过其他功能相关按钮实现控制对端 PA 的能力。通过连接关系，开发者可以实现跨设备的同步服务调度，实现如大型计算任务互助等价值场景。
5. 设备 A 与设备 B 的 PA 断开连接：在设备 A 上通过本地应用提供断开连接的按钮，将之前已连接的 PA 断开连接。
6. 设备 A 的 FA 迁移至设备 B：设备 A 上通过本地应用提供的迁移按钮，将设备 A 的业务无缝迁移到设备 B 中。通过业务迁移能力，打通设备 A 和设备 B 间的壁垒，实现如文档跨设备编辑、视频从客厅到房间跨设备接续播放等场景。

1.2.2.2 接口说明

分布式调度平台提供的连接和断开连接 PA、启动远程 FA、启动和关闭 PA 以及迁移 FA 的能力，是实现更多价值性场景的基础。

连接远程 PA

connectAbility(Intent intent, IAbilityConnection conn)接口提供连接指定设备上 PA 的能力，Intent 中指定待连接

PA 的设备 deviceId、bundleName 和 abilityName。当连接成功后，通过在 conn 定义的 onAbilityConnectDone 回调中获取对端 PA 的服务代理，两者的连接关系则由 conn 维护。具体的参数定义如下表所示：

参数名	类型	说明
intent	ohos.aafwk.content.Intent	开发者需在 intent 对应的 Operation 中指定待连接 PA 的设备 deviceId、bundleName 和 abilityName。
conn	ohos.aafwk.ability.IAbilityConnection	当连接成功或失败时，作为连接关系的回调接口。该接口提供连接完成和断开连接完成时的处理逻辑，开发者可根据具体的场景进行定义。

启动远程 FA/PA

startAbility(Intent intent)接口提供启动指定设备上 FA 和 PA 的能力，Intent 中指定待启动 FA/PA 的设备 deviceId、bundleName 和 abilityName。具体参数定义如下表所示：

参数名	类型	说明
intent	ohos.aafwk.content.Intent	当开发者需要调用该接口启动远程 PA 时，需要指定待启动 PA 的设备 deviceId、bundleName 和 abilityName。若不指定设备 deviceId，则无法跨设备调用 PA。类似地，在启动 FA 时，也需要开发者指定启动 FA 的设备 deviceId、bundleName 和 abilityName。

分布式调度平台还会提供与上述功能相对应的断开远程 PA 的连接和关闭远程 PA 的接口，相关的参数与连接、启动的接口类似。

- 断开远程 PA 连接：disconnectAbility(IAbilityConnection conn)。
- 关闭远程 PA：boolean stopAbility(Intent intent)。

迁移 FA

continueAbility(String deviceId)接口提供将本地 FA 迁移到指定设备上的能力，需要开发者在调用时指定目标设备的 deviceId。具体参数定义如下表所示：

说明

Ability 和 AbilitySlice 类均需要实现 IAbilityContinuation 及其方法，才可以实现 FA 迁移。

参数名	类型	说明
deviceId	String	当开发者需要调用该接口将本地 FA 迁移时，需要指定目标设备的 deviceId。

1.2.2.3 开发步骤

1. 导入功能依赖的包。

```

1. // 以下依赖包含分布式调度平台开放的接口，用于：连接/断开连接远程 PA、启动远程 FA、通过连接关系注册的回调函数
   onAbilityConnectDone 中返回的对端 PA 的代理，实现对 PA 的控制
2. import ohos.aafwk.ability.AbilitySlice;
3. import ohos.aafwk.ability.IAbilityConnection;
4. import ohos.aafwk.content.Intent;
5. import ohos.aafwk.content.Operation;
6. import ohos.bundle.ElementName;
7. // 为了实现迁移能力，需要引入传递迁移所需数据的包以及实现迁移能力的接口。
8. import ohos.aafwk.ability.IAbilityContinuation;
9. import ohos.aafwk.content.IntentParams;
10. // 为了实现跨设备指令及数据通信，需要集成 HarmonyOS 提供的 RPC 接口
11. import ohos.rpc.IRemoteObject;
12. import ohos.rpc.IRemoteBroker;
13. import ohos.rpc.MessageParcel;
14. import ohos.rpc.MessageOption;
15. import ohos.rpc.RemoteException;
16. import ohos.rpc.RemoteObject;
17. // (可选) 多设备场景下涉及设备选择，为此需要引入组网设备发现的能力
18. import ohos.distributedschedule.interwork.DeviceInfo;
19. import ohos.distributedschedule.interwork.DeviceManager;
20. // (可选) 设计界面相关的包函数，对 FA 界面及按钮进行绘制
21. import ohos.agp.components.Button;
22. import ohos.agp.components.Component;
23. import ohos.agp.components.Component.ClickedListener;
24. import ohos.agp.components.ComponentContainer.LayoutConfig;
25. import ohos.agp.components.element.ShapeElement;
26. import ohos.agp.components.PositionLayout;

```

2. (可选) 编写一个基本的 FA 用于使用分布式能力。

```

1. // 调用 AbilitySlice 模板实现一个用于控制基础功能的 FA
2. // Ability 和 AbilitySlice 类均需要实现 IAbilityContinuation 及其方法，才可以实现 FA 迁移。AbilitySlice 的代码示例如下
3. public class SampleSlice extends AbilitySlice implements IAbilityContinuation {
4.     @Override
5.     public void onStart(Intent intent) {

```



```
6.     super.onStart(intent);
7.     // 开发者可以自行进行界面设计
8.     // 为按钮设置统一的背景色
9.     // 例如通过 PositionLayout 指定大小可以实现简单界面
10.    PositionLayout layout = new PositionLayout(this);
11.    LayoutConfig config = new LayoutConfig(LayoutConfig.MATCH_PARENT, LayoutConfig.MATCH_PARENT);
12.    layout.setLayoutConfig(config);
13.    ShapeElement buttonBg = new ShapeElement();
14.    buttonBg.setRgbColor(new RgbColor(0,125,255));
15.    addComponents(layout, buttonBg, config);
16.    super.setUIContent(layout);
17.    }
18.
19.    @Override
20.    public void onInactive() {
21.        super.onInactive();
22.    }
23.
24.    @Override
25.    public void onActive() {
26.        super.onActive();
27.    }
28.
29.    @Override
30.    public void onBackground() {
31.        super.onBackground();
32.    }
33.
34.    @Override
35.    public void onForeground(Intent intent) {
36.        super.onForeground(intent);
37.    }
38.
39.    @Override
40.    public void onStop() {
41.        super.onStop();
42.    }
43. }
```

说明

此步骤展示了一个简单 FA 的实现过程，实际开发中请开发者根据需要进行设计。

3. (可选) 为不同的能力设置相应的控制按钮。

```
1. // 建议开发者按照自己的界面进行按钮设计
2. // 开发者可以自行实现如 createButton 的方法，新建一个显示文字 text，背景色为 buttonBg 以及大小尺寸位置符合 config 设置的按钮，用来
   与用户发生交互
3. // private Button createButton(String text, ShapeElement buttonBg, LayoutConfig config)
4. // 按照顺序在 PositionLayout 中依次添加按钮的示例
5. private void addComponents(PositionLayout linear, ShapeElement buttonBg, LayoutConfig config) {
6.     // 构建远程启动 FA 的按钮
7.     btnStartRemoteFA = createButton("StartRemoteFA", buttonBg, config);
8.     btnStartRemoteFA.setClickedListener(mStartRemoteFAListener);
9.     linear.addComponent(btnStartRemoteFA);
10.    // 构建远程启动 PA 的按钮
11.    btnStartRemotePA = createButton("StartRemotePA", buttonBg, config);
12.    btnStartRemotePA.setClickedListener(mStartRemotePAListener);
13.    linear.addComponent(btnStartRemotePA);
14.    // 构建远程关闭 PA 的按钮
15.    btnStopRemotePA = createButton("StopRemotePA", buttonBg, config);
16.    btnStopRemotePA.setClickedListener(mStopRemotePAListener);
17.    linear.addComponent(btnStopRemotePA);
18.    // 构建连接远程 PA 的按钮
19.    btnConnectRemotePA = createButton("ConnectRemotePA", buttonBg, config);
20.    btnConnectRemotePA.setClickedListener(mConnectRemotePAListener);
21.    linear.addComponent(btnConnectRemotePA);
22.    // 构建控制连接 PA 的按钮
23.    btnControlRemotePA = createButton("ControlRemotePA", buttonBg, config);
24.    btnControlRemotePA.setClickedListener(mControlPAListener);
25.    linear.addComponent(btnControlRemotePA);
26.    // 构建与远程 PA 断开连接的按钮
27.    btnDisconnectRemotePA = createButton("DisconnectRemotePA", buttonBg, config);
28.    btnDisconnectRemotePA.setClickedListener(mDisconnectRemotePAListener);
29.    linear.addComponent(btnDisconnectRemotePA);
30.    // 构建迁移 FA 的按钮
31.    btnContinueRemoteFA = createButton("ContinueRemoteFA", buttonBg, config);
32.    btnContinueRemoteFA.setClickedListener(mContinueAbilityListener);
33.    linear.addComponent(btnContinueRemoteFA);
34. }
```

说明

此处只展示了基于按钮控制的能力调度方法，实际开发中请开发者根据需要选择能力调度方式。代码示例中未体现按钮如位置、样式等具体的设置方法，详情请参考 [JAVA UI 框架](#)。

4. 通过设备管理 DeviceManager 提供的 getDeviceList 接口获取设备列表，用于指定目标设备。

```

1. // ISelectResult 是一个自定义接口，用来处理指定设备 deviceId 后执行的行为
2. interface ISelectResult {
3.     void onSelectResult(String deviceId);
4. }
5.
6. // 获得设备列表，开发者可在得到的在线设备列表中选择目标设备执行操作
7. private void scheduleRemoteAbility(ISelectResult listener) {
8.     // 调用 DeviceManager 的 getDeviceList 接口，通过 FLAG_GET_ONLINE_DEVICE 标记获得在线设备列表
9.     List<DeviceInfo> onlineDevices = DeviceManager.getDeviceList(DeviceInfo.FLAG_GET_ONLINE_DEVICE);
10.    // 判断组网设备是否为空
11.    if (onlineDevices.isEmpty()) {
12.        listener.onSelectResult(null);
13.        return;
14.    }
15.    int numDevices = onlineDevices.size();
16.    ArrayList<String> deviceIds = new ArrayList<>(numDevices);
17.    ArrayList<String> deviceNames = new ArrayList<>(numDevices);
18.    onlineDevices.forEach((device) -> {
19.        deviceIds.add(device.getDeviceId());
20.        deviceNames.add(device.getDeviceName());
21.    });
22.    // 以选择首个设备作为目标设备为例
23.    // 开发者也可按照具体场景，通过别的方式进行设备选择
24.    String selectDeviceId = deviceIds.get(0);
25.    listener.onSelectResult(selectDeviceId);
26. }

```

上述实例中涉及对在线组网设备的查询，该项能力需要开发者在对应的 config.json 中声明获

取设备列表及设备信息的权限，如下所示：

```

1. {
2.     "reqPermissions": [
3.         {
4.             "name": "ohos.permission.DISTRIBUTED_DEVICE_STATE_CHANGE"

```

```
5.     },
6.     {
7.         "name": "ohos.permission.GET_DISTRIBUTED_DEVICE_INFO"
8.     },
9.     {
10.        "name": "ohos.permission.GET_BUNDLE_INFO"
11.    }
12. ]
13. }
```

2. 为启动远程 FA 的按钮设置点击回调，实现启动远程 FA 的能力。

```
1. // 启动一个指定 bundleName 和 abilityName 的 FA
2. private ClickedListener mStartRemoteFAListener = new ClickedListener() {
3.     @Override
4.     public void onClick(Component arg0) {
5.         // 启动远程 PA
6.         scheduleRemoteAbility(new ISelectResult() {
7.             @Override
8.             void onSelectResult(String deviceId) {
9.                 if (deviceId != null) {
10.                    Intent intent = new Intent();
11.                    // 通过 scheduleRemoteAbility 指定目标设备 deviceId
12.                    // 指定待启动 FA 的 bundleName 和 abilityName
13.                    // 例如: bundleName = "com.huawei.helloworld"
14.                    //      abilityName = "com.huawei.helloworld.SampleFeatureAbility"
15.                    // 设置分布式标记, 表明当前涉及分布式能力
16.                    Operation operation = new Intent.OperationBuilder()
17.                        .withDeviceId(deviceId)
18.                        .withBundleName(bundleName)
19.                        .withAbilityName(abilityName)
20.                        .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
21.                        .build();
22.                    intent.setOperation(operation);
23.                    // 通过 AbilitySlice 包含的 startAbility 接口实现跨设备启动 FA
24.                    startAbility(intent);
25.                }
26.            }
27.        });
28.     }
```

```
29.  };
```

3. 为启动和关闭 PA 定义回调，实现启动和关闭 PA 的能力。

对于 PA 的启动、关闭、连接等操作都需要开发者提供目标设备的 deviceId。开发者可以通过 DeviceManager 相关接口得到当前组网下的设备列表，并以弹窗的形式供用户选择，也可以按照实际需要实现其他个性化的处理方式。在点击事件回调函数中，需要开发者指定得到 deviceId 后的处理逻辑，即实现类似上例中 listener.onSelectResult(String deviceId)的方法，代码示例如下：

```
1. // 启动远程 PA
2. private ClickedListener mStartRemotePAListener = new ClickedListener() {
3.     @Override
4.     public void onClick(Component arg0) {
5.         // 启动远程 PA
6.         scheduleRemoteAbility(new ISelectResult() {
7.             @Override
8.             void onSelectResult(String deviceId) {
9.                 if (deviceId != null) {
10.                    Intent intentToStartPA = new Intent();
11.                    // bundleName 和 abilityName 与待启动 PA 对应
12.                    // 例如: bundleName = "com.huawei.helloworld"
13.                    //      abilityName = "com.huawei.helloworld.SampleParticleAbility"
14.                    Operation operation = new Intent.OperationBuilder()
15.                        .withDeviceId(deviceId)
16.                        .withBundleName(bundleName)
17.                        .withAbilityName(abilityName)
18.                        .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
19.                        .build();
20.                    intentToStartPA.setOperation(operation);
21.                    startAbility(intentToStartPA);
22.                }
23.            }
24.        });
25.     }
26. };
27.
```

```

28. // 关闭远程 PA，和启动类似开发者需要指定待关闭 PA 对应的 bundleName 和 abilityName
29. private ClickedListener mStopRemotePAListener = new ClickedListener() {
30.     @Override
31.     public void onClick(Component arg0) {
32.         scheduleRemoteAbility(new ISelectResult() {
33.             @Override
34.             void onSelectResult(String deviceId) {
35.                 if (deviceId != null) {
36.                     Intent intentToStopPA = new Intent();
37.                     // bundleName 和 abilityName 与待关闭 PA 对应
38.                     // 例如: bundleName = "com.huawei.helloworld"
39.                     //      abilityName = "com.huawei.helloworld.SampleParticleAbility"
40.                     Operation operation = new Intent.OperationBuilder()
41.                         .withDeviceId(deviceId)
42.                         .withBundleName(bundleName)
43.                         .withAbilityName(abilityName)
44.                         .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
45.                         .build();
46.                     intentToStopPA.setOperation(operation);
47.                     stopAbility(intentToStopPA);
48.                 }
49.             }
50.         });
51.     }
52. };

```

说明

启动和关闭的行为类似，开发者只需在 Intent 中指定待调度 PA 的 deviceId、bundleName 和 abilityName，并以 operation 的形式封装到 Intent 内。通过 AbilitySlice (Ability) 包含的 startAbility()和 stopAbility()接口即可实现相应功能。

4. 设备 A 连接设备 B 侧的 PA，利用连接关系调用该 PA 执行特定任务，以及断开连接。

```

1. // 当连接完成时，用来提供管理已连接 PA 的能力
2. private MyRemoteProxy mProxy = null;
3. // 用于管理连接关系
4. private IAbilityConnection conn = new IAbilityConnection() {
5.     @Override
6.     public void onAbilityConnectDone(ElementName element, IRemoteObject remote, int resultCode) {
7.         // 跨设备 PA 连接完成后，会返回一个序列化的 IRemoteObject 对象
8.         // 通过该对象得到控制远端服务的代理

```

```

9.     mProxy = new MyRemoteProxy(remote);
10.     btnConnectRemotePA.setText("connectRemoteAbility done");
11. }
12.
13. @Override
14. public void onAbilityDisconnectDone(ElementName element, int resultCode) {
15.     // 当已连接的远端 PA 关闭时，会触发该回调
16.     // 支持开发者按照返回的错误信息进行 PA 生命周期管理
17.     disconnectAbility(conn);
18. }
19. };

```

仅通过启动/关闭两种方式对 PA 进行调度无法应对需长期交互的场景，因此，分布式任务调度平台向开发者提供了跨设备 PA 连接及断开连接的能力。为了对已连接 PA 进行管理，开发者需要实现一个满足 IAbilityConnection 接口的连接状态检测实例，通过该实例可以对连接及断开连接完成时设置具体的处理逻辑，例如：获取控制对端 PA 的代理等。进一步为了使用该代理跨设备调度 PA，开发者需要在本地及对端分别实现对外接口一致的代理。一个具备加法能力的代理示例如下：

```

1. // 以连接提供加法计算能力的 PA 为例。为了提供跨设备连接能力，需要在本地发起连接侧和对端被连接侧分别实现代理。
2. // 发起连接侧的代理示例如下
3. public class MyRemoteProxy implements IRemoteBroker{
4.     private static final int ERR_OK = 0;
5.     private static final int COMMAND_PLUS = IRemoteObject.MIN_TRANSACTION_ID;
6.     private final IRemoteObject remote;
7.
8.     public MyRemoteProxy(
9.         /* [in] */ IRemoteObject remote) {
10.         this.remote = remote;
11.     }
12.
13.     @Override
14.     public IRemoteObject asObject() {
15.         return remote;
16.     }
17.
18.     public int plus(
19.         /* [in] */ int a,
20.         /* [in] */ int b) throws RemoteException {
21.         MessageParcel data = MessageParcel.obtain();
22.         MessageParcel reply = MessageParcel.obtain();

```

```
23. // option 不同的取值，决定采用同步或异步方式跨设备控制 PA
24. // 本例需要同步获取对端 PA 执行加法的结果，因此采用同步的方式，即 MessageOption.TF_SYNC
25. // 具体 MessageOption 的设置，可参考相关 API 文档
26. MessageOption option = new MessageOption(MessageOption.TF_SYNC);
27. data.writeInt(a);
28. data.writeInt(b);
29.
30. try {
31.     remote.sendRequest(COMMAND_PLUS, data, reply, option);
32.     int ec = reply.readInt();
33.     if (ec != ERR_OK) {
34.         throw new RemoteException();
35.     }
36.     int result = reply.readInt();
37.     return result;
38. } catch (RemoteException e) {
39.     throw new RemoteException();
40. } finally {
41.     data.reclaim();
42.     reply.reclaim();
43. }
44. }
45. }
```

此外，对端待连接的 PA 需要实现对应的客户端，代码示例如下所示：

```
1. // 以计算加法为例，对端实现的客户端如下
2. public class MyRemote extends RemoteObject implements IRemoteBroker{
3.     private static final int ERR_OK = 0;
4.     private static final int ERROR = -1;
5.     private static final int COMMAND_PLUS = IRemoteObject.MIN_TRANSACTION_ID;
6.
7.     public MyRemote() {
8.         super("MyService_Remote");
9.     }
10.
11.     @Override
12.     public IRemoteObject asObject() {
13.         return this;
14.     }
}
```



```

15.
16.     @Override
17.     public boolean onRemoteRequest(int code, MessageParcel data, MessageParcel reply, MessageOption option) {
18.         if (code != COMMAND_PLUS) {
19.             reply.writeInt(ERROR);
20.             return false;
21.         }
22.         int value1 = data.readInt();
23.         int value2 = data.readInt();
24.         int sum = value1 + value2;
25.         reply.writeInt(ERR_OK);
26.         reply.writeInt(sum);
27.         return true;
28.     }
29. }

```

对端除了要实现如上所述的客户端外，待连接的 PA 还需要作如下修改：

```

1. // 为了返回给连接方可调用的代理，需要在该 PA 中实例化客户端，例如作为该 PA 的成员变量
2. private MyProxy remote = new MyProxy();
3. // 当该 PA 接收到连接请求时，即将该客户端转化为代理返回给连接发起侧
4. @Override
5. protected IRemoteObject onConnect(Intent intent) {
6.     super.onConnect(intent);
7.     return remote.asObject();
8. }

```

完成上述步骤后，可以通过点击事件实现连接、利用连接关系控制 PA 以及断开连接等行为，代码示例如下：

```

1. // 连接远程 PA
2. private ClickedListener mConnectRemotePAListener = new ClickedListener() {
3.     @Override
4.     public void onClick(Component arg0) {
5.         scheduleRemoteAbility(new ISelectResult() {
6.             @Override
7.             void onSelectResult(String deviceId) {
8.                 if (deviceId != null) {
9.                     Intent connectPAIntent = new Intent();
10.                    // bundleName 和 abilityName 与待连接的 PA 一一对应
11.                    // 例如: bundleName = "com.huawei.helloworld"
12.                    //      abilityName = "com.huawei.helloworld.SampleParticleAbility"
13.                    Operation operation = new Intent.OperationBuilder()

```

```
14.         .withDeviceId(deviceId)
15.         .withBundleName(bundleName)
16.         .withAbilityName(abilityName)
17.         .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
18.         .build();
19.     connectPAIntent.setOperation(operation);
20.     connectAbility(connectPAIntent, conn);
21.     }
22. }
23. });
24. }
25. };
26. // 控制已连接 PA 执行加法
27. private ClickedListener mControlPAListener = new ClickedListener() {
28.     @Override
29.     public void onClick(Component arg0) {
30.         if (mProxy != null) {
31.             int ret = -1;
32.             try {
33.                 ret = mProxy.plus(10, 20);
34.             } catch (RemoteException e) {
35.                 e.printStackTrace();
36.             }
37.             btnControlRemotePA.setText("ControlRemotePA result = " + ret);
38.         }
39.     }
40. };
41. // 与远程 PA 断开连接
42. private ClickedListener mDisconnectRemotePAListener = new ClickedListener() {
43.     @Override
44.     public void onClick(Component arg0) {
45.         // 按钮复位
46.         btnConnectRemotePA.setText("ConnectRemotePA");
47.         btnControlRemotePA.setText("ControlRemotePA");
48.         disconnectAbility(conn);
49.     }
50. };
```

说明

通过连接/断开连接远程 PA，与跨设备 PA 建立长期的管理关系。例如在本例中，通过连接关系得到远程 PA 的控制代理后，实现跨设备计算加法并将结果返回到本地显示。在实际开发中，开发者可以根据需要实现多种分布式场景，例如：跨设备位置/电量等信息的采集、跨设备计算资源互助等。

2. 设备 A 将运行时的 FA 迁移到设备 B，实现业务在设备间无缝迁移。

```

1. // 跨设备迁移 FA
2. // 本地 FA 设置当前运行任务
3. private ClickedListener mContinueAbilityListener = new ClickedListener() {
4.     @Override
5.     public void onClick(Component arg0) {
6.         // 用户选择设备后实现业务迁移
7.         scheduleRemoteAbility(new ISelectResult() {
8.             @Override
9.             public void onSelectResult(String deviceId) {
10.                 continueAbility(deviceId);
11.             }
12.         });
13.     }
14. };
    
```

此外，不同于启动行为，FA 的迁移还涉及到状态数据的传递。为此，继承的 IAbilityContinuation 接口为开发者提供迁移过程中特定事件的管理能力。通过自定义迁移事件相关的行为，最终实现对 Ability 的迁移。具体的定义可以参考相关的 API 文档，此处主要以较为常用的两个事件，包括迁移发起端完成迁移的回调 onCompleteContinuation(int result) 以及接收到远端迁移行为传递数据的回调 onRestoreData(IntentParams restoreData)。其他还包括迁移到远端设备的 FA 关闭的回调 onRemoteTerminated()、用于本地迁移发起时保存状态数据的回调 onSaveData(IntentParams saveData)和本地发起迁移的回调 onStartContinuation()。按照实际应用自定义特定场景对应的回调，可以完成多种场景下 FA 的迁移任务。

```

1. @Override
2. public boolean onSaveData(IntentParams saveData) {
3.     String exampleData = String.valueOf(System.currentTimeMillis());
4.     saveData.setParam("continueParam", exampleData);
5.     return true;
6. }
7.
8. @Override
9. public boolean onRestoreData(IntentParams restoreData) {
10.     // 远端 FA 迁移传来的状态数据，开发者可以按照特定的场景对这些数据进行处理
11.     Object data = restoreData.getParam("continueParam");
12.     return true;
13. }
    
```

```

14.
15. @Override
16. public void onCompleteContinuation(int result) {
17.     btnContinueRemoteFA.setText("ContinueAbility Done");
18. }
    
```

说明

- FA 迁移可以打通设备间的壁垒，有助于不同能力的设备进行互助。前文以一个简单的例子介绍如何通过分布式任务调度提供的能力，实现 FA 跨设备的迁移（包括 FA 启动及状态数据的同步）。
- FA 迁移过程中，远端 FA 首先接收到发起端 FA 传输的数据，再执行启动，即 onRestoreData()发生在 onStart()之前，详见 API 参考。

1.3 公共事件与通知

1.3.1 概述

HarmonyOS 通过 CES（Common Event Service，公共事件服务）为应用程序提供订阅、发布、退订公共事件的能力，通过 ANS（Advanced Notification Service，即高级通知服务）系统服务来为应用程序提供发布通知的能力。

- 公共事件可分为系统公共事件和自定义公共事件。
 - 系统公共事件：系统将收集到的事件信息，根据系统策略发送给订阅该事件的用户程序。例如：用户可感知亮灭屏事件，系统关键服务发送的系统事件（例如：USB 插拔，网络连接，系统升级等）。
 - 自定义公共事件：应用自定义一些公共事件用来处理业务逻辑。
- 通知提供应用的即时消息或通信消息，用户可以直接删除或点击通知触发进一步的操作。
- IntentAgent 封装了一个指定行为的 Intent，可以通过 IntentAgent 启动 Ability 和发送公共事件。

应用如果需要接收公共事件，需要订阅相应的事件。

1.3.1.1 约束与限制

公共事件的约束与限制

- 目前公共事件仅支持动态订阅。部分系统事件需要具有指定的权限，具体的权限见 API 参考。
- 目前公共事件订阅不支持多用户。
- ThreadMode 表示线程模型，目前仅支持 HANDLER 模式，即在当前 UI 线程上执行回调函数。
- deviceId 用来指定订阅本地公共事件还是远端公共事件。deviceId 为 null、空字符串或本地设备 deviceId 时，表示订阅本地公共事件，否则表示订阅远端公共事件。

通知的约束与限制

- 通知目前支持六种样式：普通文本、长文本、图片、社交、多行文本和媒体样式。创建通知时必须包含一种样式。
- 通知支持快捷回复。
- 目前通知订阅不支持多用户。
- 通知的订阅目前仅支持系统应用，不支持第三方应用。

IntentAgent 的限制

使用 IntentAgent 启动 Ability 时，Intent 必须指定 Ability 的包名和类名。

1.3.2 公共事件开发指导

1.3.2.1 场景介绍

每个应用都可以订阅自己感兴趣的公共事件，订阅成功后且公共事件发布后，系统会把其发送给应用。这些公共事件可能来自系统、其他应用和应用自身。HarmonyOS 提供了一套完整的 API，支持用户订阅、发送和接收公共事件。发送公共事件需要借助 CommonEventData 对

象，接收公共事件需要继承 `CommonEventSubscriber` 类并实现 `onReceiveEvent` 回调函数。

1.3.2.2 接口说明

公共事件相关基础类包含 `CommonEventData`、`CommonEventPublishInfo`、`CommonEventSubscribeInfo`、`CommonEventSubscriber` 和 `CommonEventManager`。

基础类之间的关系如下图所示：

图 1 公共事件基础类关系图



- **CommonEventData**

`CommonEventData` 封装公共事件相关信息。用于在发布、分发和接收时处理数据。在构造 `CommonEventData` 对象时，相关参数需要注意以下事项：

- `code` 为有序公共事件的结果码，`data` 为有序公共事件的结果数据，仅用于有序公共事件场景。
- `intent` 不允许为空，否则发布公共事件失败。

接口名	描述
<code>CommonEventData()</code>	创建公共事件数据。
<code>CommonEventData(Intent intent)</code>	创建公共事件数据指定 <code>Intent</code> 。
<code>CommonEventData(Intent intent, int code, String data)</code>	创建公共事件数据，指定 <code>Intent</code> 、 <code>code</code> 和 <code>data</code> 。

接口名	描述
getIntent()	获取公共事件 intent。
setCode(int code)	设置有序公共事件的结果码。
getCode()	获取有序公共事件的结果码。
setData(String data)	设置有序公共事件的详细结果数据。
getData()	获取有序公共事件的详细结果数据。

表 1 CommonEventData 主要接口

- **CommonEventPublishInfo**

CommonEventPublishInfo 封装公共事件发布相关属性、限制等信息，包括公共事件类型（有序或粘性）、接收者权限等。

- 有序公共事件：主要场景是多个订阅者有依赖关系或者对处理顺序有要求，例如：高优先级订阅者可修改公共事件内容或处理结果，包括终止公共事件处理；或者低优先级订阅者依赖高优先级的处理结果等。

有序公共事件的订阅者可以通过 `CommonEventSubscribeInfo.setPriority()`方法指定优先级，缺省为 0，优先级范围[-1000, 1000]，值越大优先级越高。

- 粘性公共事件：指公共事件的订阅动作是在公共事件发布之后进行，订阅者也能收到的公共事件类型。主要场景是由公共事件服务记录某些系统状态，如蓝牙、WLAN、充电等事件和状态。不使用粘性公共事件机制时，应用可以通过直接访问系统服务获取该状态；在状态变化时，系统服务、硬件需要提供类似 `observer` 等方式通知应用。

发布粘性公共事件可以通过 `setSticky()`方法设置，发布粘性公共事件需要申请如下权限。声明请参考表 1。

```

1. "reqPermissions": [{
2.     "name": "ohos.permission.COMMONEVENT_STICKY",
3.     "reason": "get right",
4.     "usedScene": {
5.         "ability": [
6.             ".MainAbility"
7.         ],
8.         "when": "inuse"
9.     }

```

```

10. }, {
11. ...
12. }]
    
```

接口名	描述
CommonEventPublishInfo()	创建公共事件发送信息。
CommonEventPublishInfo(CommonEventPublishInfo publishInfo)	拷贝一个公共事件发送信息。
setSticky(boolean sticky)	设置公共事件的粘性属性。
setOrdered(boolean ordered)	设置公共事件的有序属性。
setSubscriberPermissions(String[] subscriberPermissions)	设置公共事件订阅者的权限，多参数仅第一个生效。

表 2 CommonEventPublishInfo 主要接口

- **CommonEventSubscribeInfo**

CommonEventSubscribeInfo 封装公共事件订阅相关信息，比如优先级、线程模式、事件范围等。

线程模式 (ThreadMode)：设置订阅者的回调方法执行的线程模式。ThreadMode 有 HANDLER, POST, ASYNC, BACKGROUND 四种模式，目前只支持 HANDLER 模式。

- HANDLER：在 Ability 的主线程上执行。
- POST：在事件分发线程执行。
- ASYNC：在一个新创建的异步线程执行。
- BACKGROUND：在后台线程执行。

接口名	描述
CommonEventSubscribeInfo(MatchingSkills matchingSkills)	创建公共事件订阅器指定 matchingSkills。
CommonEventSubscribeInfo(CommonEventSubscribeInfo)	拷贝公共事件订阅器对象。
setPriority(int priority)	设置优先级，用于有序公共事件。

接口名	描述
setThreadMode(ThreadMode threadMode)	指定订阅者的回调函数运行在哪个线程上。
setPermission(String permission)	设置订阅者的权限。
setDeviceId(String deviceId)	指定订阅哪台设备的公共事件。

表 3 CommonEventSubscribeInfo 主要接口

- **CommonEventSubscriber**

CommonEventSubscriber 封装公共事件订阅者及相关参数。

- CommonEventSubscriber.AsyncCommonEventResult 类处理有序公共事件异步执行，详见 API 参考。
- 目前只能通过调用 CommonEventManager 的 subscribeCommonEvent()进行订阅。

接口名	描述
CommonEventSubscriber(CommonEventSubscribeInfo subscribeInfo)	构造公共事件订阅者实例。
onReceiveEvent(CommonEventData data)	由开发者实现，在接收到公共事件时被调用。
AsyncCommonEventResult goAsyncCommonEvent()	设置有序公共事件异步执行。
setCodeAndData(int code, String data)	设置有序公共事件的异步结果。
setData(String data)	设置有序公共事件的异步结果数据。
setCode(int code)	设置有序公共事件的异步结果码。
getData()	获取有序公共事件的异步结果数据。
getCode()	获取有序公共事件的异步结果码。
abortComonEvent()	取消当前的公共事件，仅对有序公共事件有效，取消后，公共事件不再向下一个订阅者传递。
getAbortCommonEvent()	获取当前有序公共事件是否取消的状态。

接口名	描述
clearAbortCommonEvent()	清除当前有序公共事件 abort 状态。
isOrderedCommonEvent()	查询当前公共事件的是否为有序公共事件。
isStickyCommonEvent()	查询当前公共事件是否为粘性公共事件。

表 4 CommonEventSubscriber 主要接口

- **CommonEventManager**

CommonEventManager 是为应用提供订阅、退订和发布公共事件的静态接口类。

方法	描述
publishCommonEvent(CommonEventData event)	发布公共事件。
publishCommonEvent(CommonEventData event, CommonEventPublishInfo publishinfo)	发布公共事件指定发布信息。
publishCommonEvent(CommonEventData event, CommonEventPublishInfo publishinfo, CommonEventSubscriber resultSubscriber)	发布有序公共事件，指定发布信息和最后一个接收者。
subscribeCommonEvent(CommonEventSubscriber subscriber)	订阅公共事件。
unsubscribeCommonEvent(CommonEventSubscriber subscriber)	退订公共事件。

表 5 CommonEventManager 主要接口

1.3.2.3 发布公共事件

开发者可以发布四种公共事件：无序的公共事件、带权限的公共事件、有序的公共事件、粘性的公共事件。

发布无序的公共事件：构造 CommonEventData 对象，设置 Intent，通过构造 operation 对象把需要发布的公共事件信息传入 intent 对象。然后调用 CommonEventManager.publishCommonEvent(CommonEventData) 接口发布公共事件。

```
1. try {
2.     Intent intent = new Intent();
```

```

3.     Operation operation = new Intent.OperationBuilder()
4.         .withAction("com.my.test")
5.         .build();
6.     intent.setOperation(operation);
7.     CommonEventData eventData = new CommonEventData(intent);
8.     CommonEventManager.publishCommonEvent(eventData);
9. } catch (RemoteException e) {
10.     HiLog.info(LABEL, "publishCommonEvent occur exception.");
11. }
    
```

发布携带权限的公共事件：构造 CommonEventPublishInfo 对象，设置订阅者的权限。

- 订阅者在 config.json 中申请所需的权限，各字段含义详见权限定义字段说明。

```

1. {
2.     "reqPermissions": [{
3.         "name": "com.example.MyApplication.permission",
4.         "reason": "get right",
5.         "usedScene": {
6.             "ability": [
7.                 ".MainAbility"
8.             ],
9.             "when": "inuse"
10.        }
11.    }, {
12.        ...
13.    }]
14. }
    
```

- 发布带权限的公共事件示例代码如下：

```

1. Intent intent = new Intent();
2. Operation operation = new Intent.OperationBuilder()
3.     .withAction("com.my.test")
4.     .build();
5. intent.setOperation(operation);
6. CommonEventData eventData = new CommonEventData(intent);
7. CommonEventPublishInfo publishInfo = new CommonEventPublishInfo();
8. String[] permissions = {"com.example.MyApplication.permission"};
9. publishInfo.setSubscriberPermissions(permissions); // 设置权限
10. try {
11.     CommonEventManager.publishCommonEvent(eventData, publishInfo);
12. } catch (RemoteException e) {
    
```

```
13.     HiLog.info(LABEL, "publishCommoneEvent occur exception.");
14. }
```

发布有序的公共事件: 构造 `CommonEventPublishInfo` 对象，通过 `setOrdered(true)`指定公共事件属性为有序公共事件，也可以指定一个最后的公共事件接收者。

```
1.  CommonEventSubscriber resultSubscriber = new MyCommonEventSubscriber();
2.  CommonEventPublishInfo publishInfo = new CommonEventPublishInfo();
3.  publishInfo.setOrdered(true); // 设置属性为有序公共事件
4.  try {
5.      CommonEventManager.publishCommonEvent(eventData, publishInfo, resultSubscriber); // 指定 resultSubscriber 为有序公共事件
        最后一个接收者。
6.  } catch (RemoteException e) {
7.      HiLog.info(LABEL, "publishCommoneEvent occur exception.");
8.  }
```

发布粘性公共事件: 构造 `CommonEventPublishInfo` 对象，通过 `setSticky(true)`指定公共事件属性为粘性公共事件。

1. 发布者首先在 `config.json` 中申请发布粘性公共事件所需的权限，各字段含义详见[权限申请字段说明](#)。

```
1.  {
2.      "reqPermissions": [{
3.          "name": "ohos.permission.COMMONEVENT_STICKY",
4.          "reason": "get right",
5.          "usedScene": {
6.              "ability": [
7.                  ".MainAbility"
8.              ],
9.              "when": "inuse"
10.         }
11.     }, {
12.     ...
13.     }]
14. }
```

2. 发布粘性公共事件。

```
1.  CommonEventPublishInfo publishInfo = new CommonEventPublishInfo();
2.  publishInfo.setSticky(true); // 设置属性为粘性公共事件
3.  try {
4.      CommonEventManager.publishCommonEvent(eventData, publishInfo);
5.  } catch (RemoteException e) {
6.      HiLog.info(LABEL, "publishCommoneEvent occur exception.");
```

```
7. }
```

1.3.2.4 订阅公共事件

1. 创建 CommonEventSubscriber 派生类，在 onReceiveEvent()回调函数中处理公共事件。

说明

此处不能执行耗时操作，否则会阻塞 UI 线程，产生用户点击没有反应等异常。

```
1. class MyCommonEventSubscriber extends CommonEventSubscriber {
2.     MyCommonEventSubscriber(CommonEventSubscriberInfo info) {
3.         super(info);
4.     }
5.     @Override
6.     public void onReceiveEvent(CommonEventData commonEventData) {
7.     }
8. }
```

2. 构造 MyCommonEventSubscriber 对象，调用 CommonEventManager.subscribeCommonEvent()接口进行订阅。

```
1. String event = "com.my.test";
2. MatchingSkills matchingSkills = new MatchingSkills();
3. filter.addEvent(event); // 自定义事件
4. filter.addEvent(CommonEventSupport.COMMON_EVENT_SCREEN_ON); // 亮屏事件
5. CommonEventSubscriberInfo subscribeInfo = new CommonEventSubscriberInfo(matchingSkills);
6. MyCommonEventSubscriber subscriber = new MyCommonEventSubscriber(subscribeInfo);
7. try {
8.     CommonEventManager.subscribeCommonEvent(subscriber);
9. } catch (RemoteException e) {
10.     HiLog.info(LABEL, "subscribeCommonEvent occur exception.");
11. }
```

如果订阅拥有指定权限应用发布的公共事件，发布者需要在 config.json 中申请权限，各字段含义详见 [权限申请字段说明](#)。

```
1. "reqPermissions": [
2.     {
3.         "name": "ohos.abilitydemo.permission.PROVIDER",
4.         "reason": "get right",
5.         "usedScene": {
6.             "ability": ["com.huawei.hmi.ivi.systemsetting.MainAbility"],
7.             "when": "inuse"
```

```

8.     }
9.     }
10. ]

```

如果订阅的公共事件是有序的，可以调用 `setPriority()` 指定优先级。

```

1. String event = "com.my.test";
2. MatchingSkills matchingSkills = new MatchingSkills();
3. matchingSkills.addEvent(event); // 自定义事件
4.
5. CommonEventSubscribeInfo subscribeInfo = new CommonEventSubscribeInfo(matchingSkills);
6. subscribeInfo.setPriority(100); // 设置优先级，优先级取值范围[-1000, 1000]，值默认为 0。
7. MyCommonEventSubscriber subscriber = new MyCommonEventSubscriber(subscribeInfo);
8. try {
9.     CommonEventManager.subscribeCommonEvent(subscriber);
10. } catch (RemoteException e) {
11.     HiLog.info(LABEL, "subscribeCommonEvent occur exception.");
12. }

```

2. 针对在 `onReceiveEvent` 中不能执行耗时操作的限制，可以使用 `CommonEventSubscriber` 的

`goAsyncCommonEvent()` 来实现异步操作，函数返回后仍保持该公共事件活跃，且执行完成后必须调用 `AsyncCommonEventResult.finishCommonEvent()` 来结束。

```

1. EventRunner runner = EventRunner.create(); //EventRunner 创建新线程，将耗时的操作放到新的线程上执行
2. MyEventHandler myHandler = new MyEventHandler(runner); //MyEventHandler 为 EventHandler 的派生类，在不同线程间分发和处理事件和 Runnable 任务
3. @Override
4. public void onReceiveEvent(CommonEventData commonEventData){
5.     final AsyncCommonEventResult result = goAsyncCommonEvent();
6.
7.     Runnable task = new Runnable() {
8.         @Override
9.         public void run() {
10.             ..... // 待执行的操作，由开发者定义
11.             result.finishCommonEvent(); // 调用 finish 结束异步操作
12.         }
13.     };
14. myHandler.postTask(task);
15. }

```

1.3.2.5 退订公共事件

在 Ability 的 onStop()中调用 CommonEventManager.unsubscribeCommonEvent()方法来退订公共事件。调用后，之前订阅的所有公共事件均被退订。

```
1. try {
2.     CommonEventManager.unsubscribeCommonEvent(subscriber);
3. } catch (RemoteException e) {
4.     HiLog.info(LABEL, "unsubscribeCommonEvent occur exception.");
5. }
```

1.3.3 通知开发指导

1.3.3.1 场景介绍

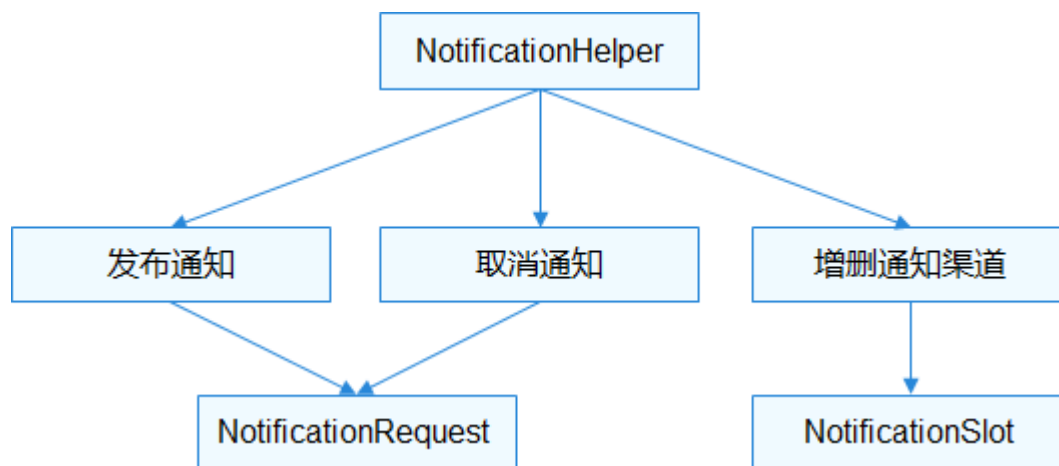
HarmonyOS 提供了通知功能，即在一个应用的 UI 界面之外显示的消息，主要用来提醒用户有来自该应用中的信息。当应用向系统发出通知时，它将先以图标的形式显示在通知栏中，用户可以下拉通知栏查看通知的详细信息。常见的使用场景：

- 显示接收到短消息、即时消息等。
- 显示应用的推送消息，如广告、版本更新等。
- 显示当前正在进行的事件，如播放音乐、导航、下载等。

1.3.3.2 接口说明

通知相关基础类包含 NotificationSlot、NotificationRequest 和 NotificationHelper。基础类之间的关系如下所示：

图 1 通知基础类关系图



- **NotificationSlot**

NotificationSlot 可以对提示音、振动、锁屏显示和重要级别等进行设置。一个应用可以创建一个或多个 NotificationSlot，在发送通知时，通过绑定不同的 NotificationSlot，实现不同用途。

说明

NotificationSlot 需要先通过 NotificationHelper 的 addNotificationSlot(NotificationSlot)方法发布后，通知才能绑定使用；所有绑定该 NotificationSlot 的通知在发布后都具备相应的特性，对象在创建后，将无法更改这些设置，对于是否启动相应设置，用户有最终控制权。

不指定 NotificationSlot 时，当前通知会使用默认的 NotificationSlot，默认的 NotificationSlot 优先级为 LEVEL_DEFAULT。

接口名	描述
NotificationSlot(String id, String name, int level)	构造 NotificationSlot。
setLevel(int level)	设置 NotificationSlot 的级别。
setName(String name)	设置 NotificationSlot 的命名。
setDescription(String description)	设置 NotificationSlot 的描述信息。
enableBypassDnd(boolean bypassDnd)	设置是否绕过系统的免打扰模式。
setEnableVibration(boolean vibration)	设置收到通知时是否使能振动。
setLockscreenVisiblness(int visiblness)	设置在锁屏场景下，收到通知后是否显示，以及显示的效果。

接口名	描述
setEnableLight(boolean isLightEnabled)	设置收到通知时是否开启呼吸灯，前提是当前硬件支持呼吸灯。
setLedLightColor(int color)	设置收到通知时的呼吸灯颜色。
setSlotGroup(String groupId)	绑定当前 NotificationSlot 到一个 NotificationSlot 组。

表 1 NotificationSlot 主要接口

NotificationSlot 的级别目前支持如下几种，由低到高：

- LEVEL_NONE：表示通知不发布。
- LEVEL_MIN：表示通知可以发布，但是不显示在通知栏，不自动弹出，无提示音；该级别不适用于前台服务的场景。
- LEVEL_LOW：表示通知可以发布且显示在通知栏，不自动弹出，无提示音。
- LEVEL_DEFAULT：表示通知发布后可在通知栏显示，不自动弹出，触发提示音。
- LEVEL_HIGH：表示通知发布后可在通知栏显示，自动弹出，触发提示。
- **NotificationRequest**

NotificationRequest 用于设置具体的通知对象，包括设置通知的属性，如：通知的分发时

间、小图标、大图标、自动删除等参数，以及设置具体的通知类型，如普通文本、长文本等。

接口名	描述
NotificationRequest()	构建一个通知。
NotificationRequest(int notificationId)	构建一个通知，指定通知的 id。通知的 Id 在应用内容具有唯一性，如果不指定，默认为 0。
setNotificationId(int notificationId)	设置当前通知 id。
setAutoDeletedTime(long time)	设置通知自动取消的时间戳。
setContent(NotificationRequest.NotificationContent content)	设置通知的具体类型。
setCreateTime(long createTime)	设置通知的创建的时间戳。

接口名	描述
setDeliveryTime(long deliveryTime)	设置通知分发的时间戳。
setSlotId(String slotId)	设置通知的 NotificationSlot id。
setTapDismissed(boolean tapDismissed)	设置通知在用户点击后是否自动取消。
setLittleIcon(PixelMap smallIcon)	设置通知的小图标，在通知左上角显示。
setBigIcon(PixelMap bigIcon)	设置通知的大图标，在通知的右边显示。
setGroupValue(String groupValue)	设置分组通知，相同分组的通知在通知栏显示时，将会折叠在一组应用中显示。
addActionButton(NotificationActionButton actionButton)	设置通知添加通知 ActionButton。
setIntentAgent(IntentAgent agent)	设置通知承载指定的 IntentAgent，在通知中实现即将触发的事件。

表 2 NotificationRequest 主要接口

具体的通知类型：目前支持六种类型，包括普通文本 NotificationNormalContent、长文本 NotificationLongTextContent、图片 NotificationPictureContent、多行 NotificationMultiLineContent、社交 NotificationConversationalContent、媒体 NotificationMediaContent。

类名	接口名	描述
NotificationNormalContent	setTitle(String title)	设置通知标题。
NotificationNormalContent	setText(String text)	设置通知内容。
NotificationNormalContent	setAdditionalText(String additionalText)	设置通知次要内容，是对通知内容的补充。
NotificationPictureContent	setBriefText(String briefText)	设置通知概要内容，是对通知内容的总结。

类名	接口名	描述
NotificationPictureContent	setExpandedTitle(String expandedTitle)	设置通知一旦设置了，折叠时显示 setTitle(String) 的值，展开时当前设置的展开标题。
NotificationPictureContent	setBigPicture(PixelMap bigPicture)	设置通知的图片内容，附加在 setText(String text) 下方。
NotificationLongTextContent	setLongText(String longText)	设置通知的长文本。
NotificationConversationalContent	setConversationTitle(String conversationTitle)	设置社交通知的标题。
NotificationConversationalContent	addConversationalMessage(ConversationalMessage message)	通知添加一条消息。
NotificationMultiLineContent	addSingleLine(String line)	在当前通知中添加一行文本。
NotificationMediaContent	setAVToken(AVToken avToken)	将媒体通知绑定指定的 AVToken。
NotificationMediaContent	setShownActions(int[] actions)	设置媒体通知待展示的按钮。

表 3 通知类型的主要接口

说明

通知发布后，通知的设置不可修改。如果下次发布通知使用相同的 id，就会更新之前发布的通知。

- **NotificationHelper**

NotificationHelper 封装了发布、更新、订阅、删除通知等静态方法。订阅通知、退订通知和查询系统中所有处于活跃状态的通知，有权限要求需为系统应用或具有订阅者权限。

接口名	描述
publishNotification(NotificationRequest request)	发布一条通知。

接口名	描述
publishNotification(String tag, NotificationRequest)	发布一条带 TAG 的通知。
cancelNotification(int notificationId)	取消指定的通知。
cancelNotification(String tag, int notificationId)	取消指定的带 TAG 的通知。
cancelAllNotifications()	取消之前发布的所有通知。
addNotificationSlot(NotificationSlot slot)	创建一个 NotificationSlot。
getNotificationSlot(String slotId)	获取 NotificationSlot。
removeNotificationSlot(String slotId)	删除一个 NotificationSlot。
getActiveNotifications()	获取当前应用发的活跃通知。
getActiveNotificationNums()	获取系统中当前应用发的活跃通知的数量。
setNotificationBadgeNum(int num)	设置通知的角标。
setNotificationBadgeNum()	设置当前应用中活跃状态通知的数量在角标显示。

表 4 NotificationHelper 主要接口

1.3.3.3 开发步骤

通知的开发指导分为创建 NotificationSlot、发布通知和取消通知等开发场景。

创建 NotificationSlot

NotificationSlot 可以设置公共通知的震动，锁屏模式，重要级别等，并通过调用 NotificationHelper.addNotificationSlot()发布 NotificationSlot 对象。

```

1. NotificationSlot slot = new NotificationSlot("slot_001", "slot_default", NotificationSlot.LEVEL_MIN); // 创建 notificationSlot 对象
2. slot.setDescription("NotificationSlotDescription");
3. slot.setEnableVibration(true); // 设置振动提醒
4. slot.setLockscreenVisiblness(NotificationRequest.VISIBLNESS_TYPE_PUBLIC); //设置锁屏模式
    
```

```

5. slot.setEnableLight(true); // 设置开启呼吸灯提醒
6. slot.setLedLightColor(Color.RED.getValue()); // 设置呼吸灯的提醒颜色
7. try {
8.     NotificationHelper.addNotificationSlot(slot);
9. } catch (RemoteException ex) {
10.     HiLog.warn(LABEL, "addNotificationSlot occur exception.");
11. }

```

发布通知

1. 构建 NotificationRequest 对象，应用发布通知前，通过 NotificationRequest 的 setSlotId()方法与 NotificationSlot 绑定，使该通知在发布后都具备该对象的特征。

```

1. int notificationId = 1;
2. NotificationRequest request = new NotificationRequest(notificationId);
3. request.setSlotId(slot.getId());

```

2. 调用 setContent()设置通知的内容。

```

1. String title = "title";
2. String text = "There is a normal notification content.";
3. NotificationNormalContent content = new NotificationNormalContent();
4. content.setTitle(title)
5.     .setText(text);
6. NotificationContent notificationContent = new NotificationContent(content);
7. request.setContent(notificationContent); // 设置通知的内容

```

3. 调用 publishNotification()发送通知。

```

1. try {
2.     NotificationHelper.publishNotification(request);
3. } catch (RemoteException ex) {
4.     HiLog.warn(LABEL, "publishNotification occur exception.");
5. }

```

取消通知

取消通知分为取消指定单条通知和取消所有通知，应用只能取消自己发布的通知。

- 调用 cancelNotification()取消指定的单条通知。

```

1. int notificationId = 1;
2. try {
3.     NotificationHelper.cancelNotification(notificationId);
4. } catch (RemoteException ex) {
5.     HiLog.warn(LABEL, "cancelNotification occur exception.");

```

6. }

- 调用 `cancelAllNotifications()`取消所有通知。

```

1. try {
2.     NotificationHelper.cancelAllNotifications();
3. } catch (RemoteException ex) {
4.     HiLog.warn(LABEL, "cancelAllNotifications occur exception.");
5. }
    
```

1.3.4 IntentAgent 开发指导

1.3.4.1 场景介绍

IntentAgent 封装了一个指定行为的 [Intent](#)，可以通过 `triggerIntentAgent` 接口主动触发，也可以与通知绑定被动触发。具体的行为包括：启动 Ability 和发送公共事件。例如：收到通知后，在点击通知后跳转到一个新的 Ability，不点击则不会触发。

1.3.4.2 接口说明

IntentAgent 相关基础类包括 `IntentAgentHelper`、`IntentAgentInfo`、`IntentAgentConstant` 和 `TriggerInfo`，基础类之间的关系如下图所示：

图 1 IntentAgent 基础类关系图

- **IntentAgentHelper**

`IntentAgentHelper` 封装了获取、激发、取消 `IntentAgent` 等静态方法。

接口名	描述
<code>getIntentAgent(Context context, IntentAgentInfo paramsInfo)</code>	获取一个 <code>IntentAgent</code> 实例。
<code>triggerIntentAgent(Context context, IntentAgent agent, IntentAgent.OnCompleted onCompleted, EventHandler handler, TriggerInfo paramsInfo)</code>	主动激发一个 <code>IntentAgent</code> 实例。

接口名	描述
cancel(IntentAgent agent)	取消一个 IntentAgent 实例。
judgeEquality(IntentAgent agent, IntentAgent otherAgent)	判断两个 IntentAgent 实例是否相等。
getHashCode(IntentAgent agent)	获取一个 IntentAgent 实例的哈希码。
getBundleName(IntentAgent agent)	获取一个 IntentAgent 实例的包名。
getUid(IntentAgent agent)	获取一个 IntentAgent 实例的用户 ID。

表 1 IntentAgentHelper 主要接口

- **IntentAgentInfo**

IntentAgentInfo 类封装了获取一个 IntentAgent 实例所需的数据。使用构造函数 IntentAgentInfo(int requestCode, OperationType operationType, List<Flags> flags, List<Intent> intents, IntentParams extraInfo) 获取 IntentAgentInfo 对象。

- requestCode: 使用者定义的一个私有值。
- operationType: 为 IntentAgentConstant.OperationType 枚举中的值。
- flags: 为 IntentAgentConstant.Flags 枚举中的值。
- intents: 将被执行的意图列表。operationType 的值为 START_ABILITY, START_SERVICE 和 SEND_COMMON_EVENT 时, intents 列表只允许包含一个 Intent; operationType 的值为 START_ABILITIES 时, intents 列表允许包含多个 Intent
- extraInfo: 表明如何启动一个有页面的 ability, 可以为 null, 只在 operationType 的值为 START_ABILITY 和 START_ABILITIES 时有意义。

- **IntentAgentConstant**

IntentAgentConstant 类中包含 OperationType 和 Flags 两个枚举类:

类名	枚举值
IntentAgentConstant.OperationType	<p>UNKNOWN_TYPE: 不识别的类型。</p> <p>START_ABILITY: 开启一个有页面的 Ability。</p> <p>START_ABILITIES: 开启多个有页面的 Ability。</p> <p>START_SERVICE: 开启一个无页面的 ability。</p> <p>SEND_COMMON_EVENT: 发送一个公共事件。</p>
IntentAgentConstant.Flags	<p>ONE_TIME_FLAG: IntentAgent 仅能使用一次。只在 operationType 的值为 START_ABILITY, START_SERVICE 和 SEND_COMMON_EVENT 时有意义。</p> <p>NO_BUILD_FLAG: 如果描述 IntentAgent 对象不存在，则不创建它，直接返回 null。只在 operationType 的值为 START_ABILITY, START_SERVICE 和 SEND_COMMON_EVENT 时有意义。</p> <p>CANCEL_PRESENT_FLAG: 在生成一个新的 IntentAgent 对象前取消已存在的一个 IntentAgent 对象。只在 operationType 的值为 START_ABILITY, START_SERVICE 和 SEND_COMMON_EVENT 时有意义。</p> <p>UPDATE_PRESENT_FLAG: 使用新的 IntentAgent 的额外数据替换已存在的 IntentAgent 中的额外数据。只在 operationType 的值为 START_ABILITY, START_SERVICE 和 SEND_COMMON_EVENT 时有意义。</p> <p>CONSTANT_FLAG: IntentAgent 是不可变的。</p> <p>REPLACE_ELEMENT: 当前 Intent 中的 element 属性可被 IntentAgentHelper.triggerIntentAgent() 中 Intent 的 element 属性取代。</p> <p>REPLACE_ACTION: 当前 Intent 中的 action 属性可被 IntentAgentHelper.triggerIntentAgent() 中 Intent 的 action 属性取代。</p> <p>REPLACE_URI: 当前 Intent 中的 uri 属性可被 IntentAgentHelper.triggerIntentAgent() 中 Intent 的 uri 属性取代。</p> <p>REPLACE_ENTITIES: 当前 Intent 中的 entities 属性可被 IntentAgentHelper.triggerIntentAgent() 中 Intent 的 entities 属性取代。</p> <p>REPLACE_BUNDLE: 当前 Intent 中的 bundleName 属性可被 IntentAgentHelper.triggerIntentAgent() 中 Intent 的 bundleName 属性取代。</p>

- **TriggerInfo**

TriggerInfo 类封装了主动激发一个 IntentAgent 实例所需的数据，使用构造函数 TriggerInfo(String permission,

IntentParams extraInfo, Intent intent, int code)获取 TriggerInfo 对象。

- permission: IntentAgent 的接收者的权限名称, 只在 operationType 的值为 SEND_COMMON_EVENT 时, 该参数才有意义。
- extraInfo: 激发 IntentAgent 时用户自定义的额外数据。
- intent: 额外的 Intent。如果 IntentAgentInfo 成员变量 flags 包含 CONSTANT_FLAG, 则忽略该参数; 如果 flags 包含 REPLACE_ELEMENT, REPLACE_ACTION, REPLACE_URI, REPLACE_ENTITIES 或 REPLACE_BUNDLE, 则使用额外 Intent 的 element, action, uri, entities 或 bundleName 属性替换原始 Intent 中对应的属性。如果 intent 是空, 则不替换原始 Intent 的属性。
- code: 提供给 IntentAgent 目标的结果码。

1.3.4.3 开发步骤

获取 IntentAgent 的代码示例如下:

```

1. // 指定要启动的 Ability 的 BundleName 和 AbilityName 字段
2. // 将 Operation 对象设置到 Intent 中
3. Operation operation = new Intent.OperationBuilder()
4.     .withDeviceld("")
5.     .withBundleName("com.huawei.testintentagent")
6.     .withAbilityName("com.huawei.testintentagent.entry.IntentAgentAbility")
7.     .build();
8. intent.setOperation(operation);
9. List<Intent> intentList = new ArrayList<>();
10. intentList.add(intent);
11. // 定义请求码
12. int requestCode = 200;
13. // 设置 flags
14. List<Flags> flags = new ArrayList<>();
15. flags.add(Flags.UPDATE_PRESENT_FLAG);
16. // 指定启动一个有页面的 Ability
17. IntentAgentInfo paramsInfo = new IntentAgentInfo(requestCode, IntentAgentConstant.OperationType.START_ABILITY, flags,
    intentList, null);
18. // 获取 IntentAgent 实例
19. IntentAgent agent = IntentAgentHelper.getIntentAgent(this, paramsInfo);
    
```

通知中添加 IntentAgent 的代码示例如下:

```
1. int notificationId = 1;
2. NotificationRequest request = new NotificationRequest(notificationId);
3. String title = "title";
4. String text = "There is a normal notification content.";
5. NotificationNormalContent content = new NotificationNormalContent();
6. content.setTitle(title)
7.     .setText(text);
8. NotificationContent notificationContent = new NotificationContent(content);
9. request.setContent(notificationContent); // 设置通知的内容
10. request.setIntentAgent(agent); // 设置通知的 IntentAgent
```

主动激发 IntentAgent 的代码示例如下：

```
1. int code = 100;
2. IntentAgentHelper.triggerIntentAgent(this, agent, null, null, new TriggerInfo(null, null, null, code ));
```

1.4 剪贴板

1.4.1 概述

用户通过系统剪贴板服务，可实现应用之间的简单数据传递。例如：在应用 A 中复制的数据，可以在应用 B 中粘贴，反之亦可。

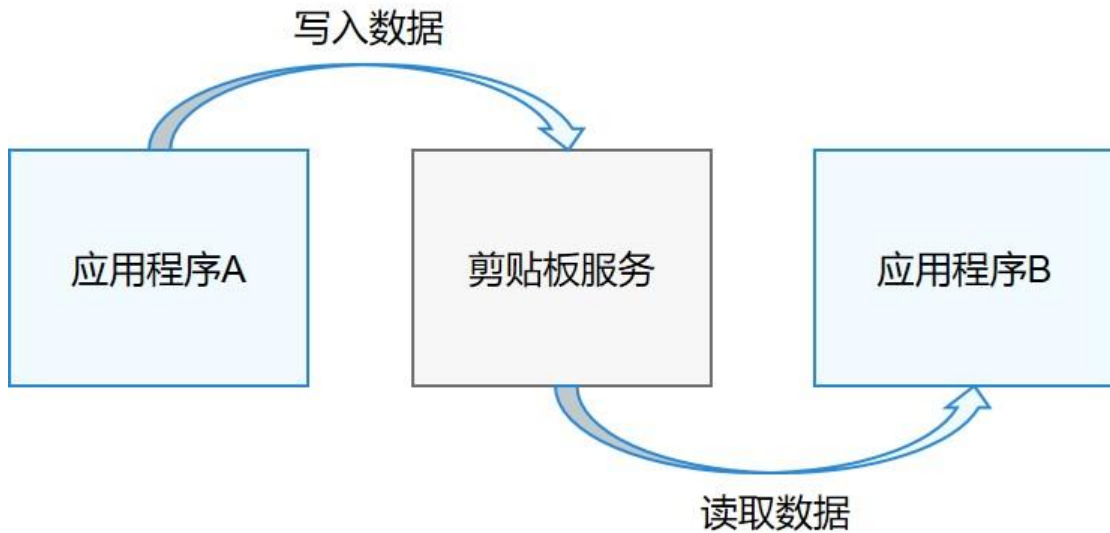
- HarmonyOS 提供系统剪贴板服务的操作接口，支持用户程序从系统剪贴板中读取、写入和查询剪贴板数据，以及添加、移除系统剪贴板数据变化的回调。
- HarmonyOS 提供剪贴板数据的对象定义，包含内容对象和属性对象。
- HarmonyOS 支持跨设备分布式剪贴板服务。

1.4.2 开发指导

1.4.2.1 场景介绍

同一设备的应用程序 A、B 之间可以借助系统剪贴板服务完成简单数据的传递，即应用程序 A 向剪贴板服务写入数据后，应用程序 B 可以从中读取数据。在满足分布式剪贴板服务的使用条件时，应用程序 A、B 也可以来自组网内的不同设备。

图 1 剪贴板服务示意图



在使用剪贴板服务时，需要注意以下几点：

- 只有在前台获取到焦点的应用才有读取系统剪贴板的权限（系统默认输入法应用除外）。
- 写入到剪贴板服务中的剪贴板数据不会随应用程序结束而销毁。
- 对同一用户而言，写入剪贴板服务的数据会被下一次写入的剪贴板数据所覆盖。
- 如果设备满足分布式组网条件，且进行复制操作的设备打开了剪贴板分布式开关，未配置“仅在本地”标志位的剪贴板数据里的 MIME 类型为纯文本和 HTML 的内容可以被组网内其他打开了剪贴板分布式开关的设备粘贴出来。
- 在同一设备内，剪贴板单次传递内容不应超过 800KB。在分布式场景下多设备间传递时，每次传递内容不应超过 64KB。

1.4.2.2 接口说明

SystemPasteboard 提供系统剪贴板操作的相关接口，比如复制、粘贴、配置回调等。

PasteData 是剪贴板服务操作的数据对象，一个 PasteData 由若干个内容节点

(PasteData.Record) 和一个属性集合对象 (PasteData.DataProperty) 组成。Record 是存

放剪贴板数据内容信息的最小单位，每个 Record 都有其特定的 MIME 类型，如纯文本、

HTML、URI、Intent。剪贴板数据的属性信息存放在 DataProperty 中，包括标签、时间戳、

“仅在本机” 标记位等。

SystemPasteboard

SystemPasteboard 提供系统剪贴板服务的操作接口，比如复制、粘贴、配置回调等。

接口名	描述
getSystemPasteboard(Context context)	获取系统剪贴板服务的对象实例。
getPasteData()	读取当前系统剪贴板中的数据。
hasPasteData()	判断当前系统剪贴板中是否有内容。
setPasteData(PasteData data)	将剪贴板数据写入到系统剪贴板。
clear()	清空系统剪贴板数据。
addPasteDataChangeListener(IPasteDataChangeListener listener)	用户程序添加系统剪贴板数据变化的回调，当系统剪贴板数据发生变化时，会触发用户程序的回调实现。
removePasteDataChangeListener(IPasteDataChangeListener listener)	用户程序移除系统剪贴板数据变化的回调。

表 1 SystemPasteboard 的主要接口

PasteData

PasteData 是剪贴板服务操作的数据对象，其中内容节点定义为 PasteData.Record，属性集合定义为

PasteData.DataProperty.

接口名	描述
PasteData()	构造器，创建一个空内容数据对象。
createPlainTextData(CharSequence text)	构建一个包含纯文本内容节点的数据对象。
creatHtmlData(String htmlText)	构建一个包含 HTML 内容节点的数据对象。
creatUriData(Uri uri)	构建一个包含 URI 内容节点的数据对象。
creatIntentData(Intent intent)	构建一个包含 Intent 内容节点的数据对象。
getPrimaryMimeType()	获取数据对象中首个内容节点的 MIME 类型，如果没有查询到内容，将返回一个空字符串。
getPrimaryText()	获取数据对象中首个内容节点的纯文本内容，如果没有查询到内容，将返回一个空对象。
addTextRecord(CharSequence text)	向数据对象中添加一个纯文本内容节点，该方法会自动更新数据属性中的 MIME 类型集合，最多只能添加 128 个内容节点。
addRecord(Record record)	向数据对象中添加一个内容节点，该方法会自动更新数据属性中的 MIME 类型集合，最多只能添加 128 个内容节点。
getRecordCount()	获取数据对象中内容节点的数量。
getRecordAt(int index)	获取数据对象在指定下标处的内容节点，如果操作失败会返回空对象。
removeRecordAt(int index)	移除数据对象在指定下标处的内容节点，如果操作成功会返回 true，操作失败会返回 false。
getMimeTypes()	获取数据对象中上所有内容节点的 MIME 类型列表，当内容节点为空时，返回列表为空对象。
getProperty()	获取该数据对象的属性集合成员。

表 2 PasteData 的主要接口

接口名	描述
MIMETYPE_TEXT_PLAIN= "text/plain"	纯文本的 MIME 类型定义。
MIMETYPE_TEXT_HTML= "text/html"	HTML 的 MIME 类型定义。
MIMETYPE_TEXT_URI= "text/uri"	URI 的 MIME 类型定义。
MIMETYPE_TEXT_INTENT= "text/ohos.intent"	Intent 的 MIME 类型定义。
MAX_RECORD_NUM=128	单个 PasteData 中所能包含的 Record 的数量上限。

表 3 PasteData 中定义的常量

PasteData.Record

一个 PasteData 中包含若干个特定 MIME 类型的 PasteData.Record，每个 Record 是存放剪贴板数据内容信息的最小单位。

接口名	描述
createPlainTextRecord(CharSequence text)	构造一个 MIME 类型为纯文本的内容节点。
createHtmlTextRecord(String htmlText)	构造一个 MIME 类型为 HTML 的内容节点。
createUriRecord(Uri uri)	构造一个 MIME 类型为 URI 的内容节点。
createIntentRecord(Intent intent)	构造一个 MIME 类型为 Intent 的内容节点。
getPlainText()	获取该内容节点中的文本内容，如果没有内容将返回空对象。
getHtmlText()	获取该内容节点中的 HTML 内容，如果没有内容将返回空对象。
getUri()	获取该内容节点中的 URI 内容，如果没有内容将返回空对象。

接口名	描述
getIntent()	获取该内容节点中的 Intent 内容，如果没有内容将返回空对象。
getMimeType()	获取该内容节点的 MIME 类型。
convertToText(Context context)	将该内容节点的内容转为文本形式。

表 4 PasteData.Record 的主要接口

PasteData.DataProperty

每个 PasteData 中都有一个 PasteData.DataProperty 成员，其中存放着该数据对象的属性集合，例如自定义标签、MIME 类型集合列表，“仅在本本地”标记位等。

接口名	描述
getMimeTypes()	获取所属数据对象的 MIME 类型集合列表，当内容节点为空时，返回列表为空对象。
hasMimeType(String mimeType)	判断所属数据对象中是否包含特定 MIME 类型的内容。
getTimestamp()	获取所属数据对象被写入系统剪贴板时的时间戳，如果该数据对象尚未被写入，则返回 0。
setTag(CharSequence tag)	设置自定义标签。
getTag()	获取自定义标签。
setAdditions(PacMap extraProps)	设置一些附加键值对信息。
getAdditions()	获取附加键值对信息。
setLocalOnly(boolean isLocalOnly)	配置“仅在本本地”标志位，默认配置为 false，表示此数据对象能在分布式剪贴板场景下跨设备传递，否则只在本地设备使用。
isLocalOnly()	查询“仅在本本地”标志位。

接口名	描述
表 5 PasteData.DataProperty 的主要接口	

IPasteDataChangeListener

IPasteDataChangeListener 是定义剪贴板数据变化回调的接口类，开发者需要实现此接口来编码触发回调时的处理逻辑。

接口名	描述
onChanged()	当系统剪贴板数据发生变化时的回调接口。

表 6 IPasteDataChangeListener 的主要接口

1.4.2.3 开发步骤

1. 应用 A 获取系统剪贴板服务。

```
1. SystemPasteboard pasteboard = SystemPasteboard.getSystemPasteboard(appContext);
```

2. 应用 A 向系统剪贴板中写入一条纯文本数据。

```
1. if (pasteboard != null) {
2.     pasteboard.setPasteData(PasteData.creatPlainTextData("Hello, world!"));
3. }
```

3. 应用 B 从系统剪贴板读取数据，将数据对象中的首个文本类型（纯文本/HTML）内容信息在控件中显示，忽略其他类型内容。

```
1. PasteData pasteData = pasteboard.getPasteData();
2. if (pasteData == null) {
3.     return;
4. }
5. DataProperty dataProperty = pasteData.getProperty();
6. boolean hasHtml = dataProperty.hasMimeType(PasteData.MIMETYPE_TEXT_HTML);
7. boolean hasText = dataProperty.hasMimeType(PasteData.MIMETYPE_TEXT_PLAIN);
8. if (hasHtml || hasText) {
9.     for (int i = 0; i < pasteData.getRecordCount(); i++) {
```



```

10.     Record record = pasteData.getRecordAt(i);
11.     String mimeType = record.getMimeType();
12.     if (mimeType.equals(PasteData.MIMETYPE_TEXT_HTML)) {
13.         text.setText(record.getHtmlText());
14.         break;
15.     } else if (mimeType.equals(PasteData.MIMETYPE_TEXT_PLAIN)) {
16.         text.setText(record.getPlainText().toString());
17.         break;
18.     }
19. }
20. }

```

4. 应用 C 注册添加系统剪贴板数据变化回调，当系统剪贴板数据发生变化时触发处理逻辑。

```

1.  IPasteDataChangeListener listener = new IPasteDataChangeListener() {
2.     @Override
3.     public void onChanged() {
4.         PasteData pasteData = pasteboard.getPasteData();
5.         if (pasteData == null) {
6.             return;
7.         }
8.         // Operations to handle data change on the system pasteboard
9.     }
10. };
11. pasteboard.addPasteDataChangeListener(listener);

```

2 线程

2.1 线程管理

2.1.1 概述

不同应用在各自独立的进程中运行。当应用以任何形式启动时，系统为其创建进程，该进程将持续运行。当进程完成当前任务处于等待状态，且系统资源不足时，系统自动回收。

在启动应用时，系统会为该应用创建一个称为“主线程”的执行线程。该线程随着应用创建或消失，是应用的核心线程。UI 界面的显示和更新等操作，都是在主线程上进行。主线程又称 UI 线程，默认情况下，所有的操作都是在主线程上执行。如果需要执行比较耗时的任务（如下载文件、查询数据库），可创建其他线程来处理。

2.1.2 开发指导

2.1.2.1 场景介绍

如果应用的业务逻辑比较复杂，可能需要创建多个线程来执行多个任务。这种情况下，代码复杂难以维护，任务与线程的交互也会更加繁杂。要解决此问题，开发者可以使用“TaskDispatcher”来分发不同的任务。

2.1.2.2 接口说明

TaskDispatcher 是一个任务分发器，它是 Ability 分发任务的基本接口，隐藏任务所在线程的实现细节。

为保证应用有更好的响应性，我们需要设计任务的优先级。在 UI 线程上运行的任务默认以高优先级运行，如果某个任务无需等待结果，则可以用低优先级。

优先级	详细描述
HIGH	最高任务优先级，比默认优先级、低优先级的任务有更高的几率得到执行。
DEFAULT	默认任务优先级，比低优先级的任务有更高的几率得到执行。
LOW	低任务优先级，比高优先级、默认优先级的任务有更低的几率得到执行。

表 1 线程优先级介绍

TaskDispatcher 具有多种实现，每种实现对应不同的任务分发器。在分发任务时可以指定任务的优先级，由同一个任务分发器分发出的任务具有相同的优先级。系统提供的任务分发器有 GlobalTaskDispatcher、ParallelTaskDispatcher、SerialTaskDispatcher、SpecTaskDispatcher。

- **GlobalTaskDispatcher**

全局并发任务分发器，由 Ability 执行 getGlobalTaskDispatcher() 获取。适用于任务之间没有联系的情况。一个应用只有一个 GlobalTaskDispatcher，它在程序结束时才被销毁。

```
1. TaskDispatcher globalTaskDispatcher = getGlobalTaskDispatcher(TaskPriority.DEFAULT);
```

- **ParallelTaskDispatcher**

并发任务分发器，由 Ability 执行 createParallelTaskDispatcher() 创建并返回。与 GlobalTaskDispatcher 不同的是，ParallelTaskDispatcher 不具有全局唯一性，可以创建多个。开发者在创建或销毁 dispatcher 时，需要持有对应的对象引用。

```
1. String dispatcherName = "parallelTaskDispatcher";
2. TaskDispatcher parallelTaskDispatcher = createParallelTaskDispatcher(dispatcherName, TaskPriority.DEFAULT);
```

- **SerialTaskDispatcher**

串行任务分发器，由 Ability 执行 createSerialTaskDispatcher() 创建并返回。由该分发器分发的所有的任务都是按顺序执行，但是执行这些任务的线程并不是固定的。如果要执行并行任务，应使用 ParallelTaskDispatcher 或者 GlobalTaskDispatcher，而不是创建多个 SerialTaskDispatcher。如果任务之间没有依赖，应使用 GlobalTaskDispatcher 来实现。它的创建和销毁由开发者自己管理，开发者在使用期间需要持有该对象引用。

```
1. String dispatcherName = "serialTaskDispatcher";
2. TaskDispatcher serialTaskDispatcher = createSerialTaskDispatcher(dispatcherName, TaskPriority.DEFAULT);
```

- **SpecTaskDispatcher**

专有任务分发器，绑定到专有线程上的任务分发器。目前已有的专有线程是主线程。

UITaskDispatcher 和 MainTaskDispatcher 都属于 SpecTaskDispatcher。建议使用

UITaskDispatcher。

UITaskDispatcher: 绑定到应用主线程的专有任务分发器，由 Ability 执行

getUITaskDispatcher()创建并返回。由该分发器分发的所有的任务都是在主线程上按顺序执

行，它在应用程序结束时被销毁。

```
1. TaskDispatcher uiTaskDispatcher = getUITaskDispatcher();
```

MainTaskDispatcher: 由 Ability 执行 getMainTaskDispatcher()创建并返回。

```
2. TaskDispatcher mainTaskDispatcher= getMainTaskDispatcher()
```

2.1.2.3 开发步骤

- **syncDispatch**

同步派发任务：派发任务并在当前线程等待任务执行完成。在返回前，当前线程会被阻塞。

如下代码示例展示了如何使用 GlobalTaskDispatcher 派发同步任务：

```
1. globalTaskDispatcher.syncDispatch(new Runnable() {
2.     @Override
3.     public void run() {
4.         HiLog.info(label, "sync task1 run");
5.     }
6. });
7. HiLog.info(label, "after sync task1");
8.
9. globalTaskDispatcher.syncDispatch(new Runnable() {
10.    @Override
11.    public void run() {
12.        HiLog.info(label, "sync task2 run");
13.    }
14. });
15. HiLog.info(label, "after sync task2");
```

```

16.
17. globalTaskDispatcher.syncDispatch(new Runnable() {
18.     @Override
19.     public void run() {
20.         HiLog.info(label, "sync task3 run");
21.     }
22. });
23. HiLog.info(label, "after sync task3");
24.
25. // 执行结果如下:
26. // sync task1 run
27. // after sync task1
28. // sync task2 run
29. // after sync task2
30. // sync task3 run
31. // after sync task3

```

说明

如果对 syncDispatch 使用不当，将会导致死锁。如下情形可能导致死锁发生：

- 在专有线程上，利用该专有任务分发器进行 syncDispatch。
- 在被某个串行任务分发器（dispatcher_a）派发的任务中，再次利用同一个串行任务分发器（dispatcher_a）对象派发任务。
- 在被某个串行任务分发器（dispatcher_a）派发的任务中，经过数次派发任务，最终又利用该（dispatcher_a）串行任务分发器派发任务。例如：dispatcher_a 派发的任务使用 dispatcher_b 进行任务的派发，在 dispatcher_b 派发的任务中又利用 dispatcher_a 进行派发任务。
- 串行任务分发器（dispatcher_a）派发的任务中利用串行任务分发器（dispatcher_b）进行同步派发任务，同时 dispatcher_b 派发的任务中利用串行任务分发器（dispatcher_a）进行同步派发任务。在特定的线程执行顺序下将导致死锁。

- **asyncDispatch**

异步派发任务：派发任务，并立即返回，返回值是一个可用于取消任务的接口。

如下代码示例展示了如何使用 GlobalTaskDispatcher 派发异步任务：

```

0. Revocable revocable = globalTaskDispatcher.asyncDispatch(new Runnable() {
1.     @Override
2.     public void run() {
3.         HiLog.info(label, "async task1 run");
4.     }

```

```

5.     });
6.     HiLog.info(label, "after async task1");
7.
8.     // 执行结果可能如下:
9.     // after async task1
10.    // async task1 run

```

- **delayDispatch**

异步延迟派发任务：异步执行，函数立即返回，内部会在延时指定时间后将任务派发到相应队列中。延时时间参数仅代表在这段时间以后任务分发器会将任务加入到队列中，任务的实际执行时间可能晚于这个时间。具体比这个数值晚多久，取决于队列及内部线程池的繁忙情况。

如下代码示例展示了如何使用 GlobalTaskDispatcher 延迟派发任务：

```

0.     final long callTime = System.currentTimeMillis();
1.     final long delayTime = 50;
2.     Revocable revocable = globalTaskDispatcher.delayDispatch(new Runnable() {
3.         @Override
4.         public void run() {
5.             HiLog.info(label, "delayDispatch task1 run");
6.             final long actualDelayMs = System.currentTimeMillis() - callTime;
7.             HiLog.info(label, "actualDelayTime >= delayTime : %{public}b" + (actualDelayMs >= delayTime));
8.         }
9.     }, delayTime );
10.    HiLog.info(label, "after delayDispatch task1");
11.
12.    // 执行结果可能如下:
13.    // after delayDispatch task1
14.    // delayDispatch task1 run
15.    // actualDelayTime >= delayTime : true

```

- **Group**

任务组：表示一组任务，且该组任务之间有一定的联系，由 TaskDispatcher 执行

createDispatchGroup 创建并返回。将任务加入任务组，返回一个用于取消任务的接口。

如下代码示例展示了任务组的使用方式：将一系列相关联的下载任务放入一个任务组，执行完下载任务后关闭应用。

```
0. void groupTest(Context context) {
1.     TaskDispatcher dispatcher = context.createParallelTaskDispatcher(dispatcherName, TaskPriority.DEFAULT);
2.     // 创建任务组。
3.     Group group = dispatcher.createDispatchGroup();
4.     // 将任务 1 加入任务组，返回一个用于取消任务的接口。
5.     dispatcher.asyncGroupDispatch(group, new Runnable(){
6.         public void run() {
7.             HiLog.info(label, "download task1 is running");
8.             downLoadRes(url1);
9.         }
10.    });
11.    // 将与任务 1 相关联的任务 2 加入任务组。
12.    dispatcher.asyncGroupDispatch(group, new Runnable(){
13.        public void run() {
14.            HiLog.info(label, "download task2 is running");
15.            downLoadRes(url2);
16.        }
17.    });
18.    // 在任务组中的所有任务执行完成后执行指定任务。
19.    dispatcher.groupDispatchNotify(group, new Runnable(){
20.        public void run() {
21.            HiLog.info(label, "the close task is running after all tasks in the group are completed");
22.            closeApp();
23.        }
24.    });
25. }
26.
27. // 可能的执行结果:
28. // download task1 is running
29. // download task2 is running
30. // the close task is running after all tasks in the group are completed
31.
32. // 另外一种可能的执行结果:
33. // download task2 is running
34. // download task1 is running
35. // the close task is running after all tasks in the group are completed
```

- **Revocable**

取消任务：Revocable 是取消一个异步任务的接口。异步任务包括通过 `asyncDispatch`、`delayDispatch`、`asyncGroupDispatch` 派发的任务。如果任务已经在执行中或执行完成，则会返回取消失败。

如下代码示例展示了如何取消一个异步延时任务：

```
0. void postTaskAndRevoke(Context context) {
1.     TaskDispatcher dispatcher = context.getUITaskDispatcher();
2.     Revocable revocable = dispatcher.delayDispatch(new Runnable(){
3.         HiLog.info(label, "delay dispatch");
4.     }, 10);
5.     boolean revoked = revocable.revoke();
6.     HiLog.info(label, "%{public}b", revoked);
7. }
8.
9. // 一种可能的结果如下：
10. // true
```

- **syncDispatchBarrier**

同步设置屏障任务：在任务组上设立任务执行屏障，同步等待任务组中的所有任务执行完成，再执行指定任务。

说明

在全球并发任务分发器（GlobalTaskDispatcher）上同步设置任务屏障，将不会起到屏障作用。

如下代码示例展示了如何同步设置屏障：

```
0. TaskDispatcher dispatcher = context.createParallelTaskDispatcher(dispatcherName, TaskPriority.DEFAULT);
1. // 创建任务组。
2. Group group = dispatcher.createDispatchGroup();
3. // 将任务加入任务组，返回一个用于取消任务的接口。
4. dispatcher.asyncGroupDispatch(group, new Runnable(){
5.     public void run() {
6.         HiLog.info(label, "task1 is running"); // 1
7.     }
8. });
9. dispatcher.asyncGroupDispatch(group, new Runnable(){
10.    public void run() {
11.        HiLog.info(label, "task2 is running"); // 2
```



```

12.     }
13. });
14.
15. dispatcher.syncDispatchBarrier(new Runnable() {
16.     public void run() {
17.         HiLog.info(label, "barrier"); // 3
18.     });
19.     HiLog.info(label, "after syncDispatchBarrier"); // 4
20. }
21.
22. // 1 和 2 的执行顺序不定; 3 和 4 总是在 1 和 2 之后按顺序执行。
23.
24. // 可能的执行结果:
25. // task1 is running
26. // task2 is running
27. // barrier
28. // after syncDispatchBarrier
29.
30. // 另外一种执行结果:
31. // task2 is running
32. // task1 is running
33. // barrier
34. // after syncDispatchBarrier

```

- **asyncDispatchBarrier**

异步设置屏障任务：在任务组上设立任务执行屏障后直接返回，指定任务将在任务组中的所有任务执行完成后再执行。

说明

在全局并发任务分发器（GlobalTaskDispatcher）上异步设置任务屏障，将不会起到屏障作用。可以使用并发任务分发器（ParallelTaskDispatcher）分离不同的任务组，达到微观并行、宏观串行的行为。

如下代码示例展示了如何异步设置屏障：

```

0.     TaskDispatcher dispatcher = context.createParallelTaskDispatcher(dispatcherName, TaskPriority.DEFAULT);
1.     // 创建任务组。
2.     Group group = dispatcher.createDispatchGroup();
3.     // 将任务加入任务组，返回一个用于取消任务的接口。
4.     dispatcher.asyncGroupDispatch(group, new Runnable(){
5.         public void run() {

```

```

6.         HiLog.info(label, "task1 is running"); // 1
7.     }
8. });
9.     dispatcher.asyncGroupDispatch(group, new Runnable(){
10.         public void run() {
11.             HiLog.info(label, "task2 is running"); // 2
12.         }
13.     });
14.
15.     dispatcher.asyncDispatchBarrier(new Runnable() {
16.         public void run() {
17.             HiLog.info(label, "barrier"); // 3
18.         }
19.     });
20.     HiLog.info(label, "after syncDispatchBarrier"); // 4
21. }
22.
23. // 1 和 2 的执行顺序不定，但总在 3 和 4 之前执行；4 可能在 3 之前执行
24.
25. // 可能的执行结果:
26. // task1 is running
27. // task2 is running
28. // after syncDispatchBarrier
29. // barrier

```

- **applyDispatch**

执行多次任务：对指定任务执行多次。

如下代码示例展示了如何执行多次任务：

```

0.     final int total = 10;
1.     final CountDownLatch latch = new CountDownLatch(total);
2.     final ArrayList<Long> indexList = new ArrayList<>(total);
3.
4.     // 执行任务 total 次
5.     dispatcher.applyDispatch((index) -> {
6.         indexList.add(index);
7.         latch.countDown();
8.     }, total);
9.

```

```
10. // 设置任务超时
11. try {
12.     latch.await();
13. } catch (InterruptedException exception) {
14.     HiLog.info(label, "latch exception");
15. }
16. HiLog.info(label, "list size matches, %{public}b", (total == indexList.size()));
17.
18. // 执行结果:
19. // list size matches, true
```

2.2 线程间通信

2.2.1 概述

在开发过程中，开发者经常需要在当前线程中处理下载任务等较为耗时的操作，但是又不希望当前的线程受到阻塞。此时，就可以使用 `EventHandler` 机制。`EventHandler` 是 HarmonyOS 用于处理线程间通信的一种机制，可以通过 [EventRunner](#) 创建新线程，将耗时的操作放到新线程上执行。这样既不阻塞原来的线程，任务又可以得到合理的处理。比如：主线程使用 `EventHandler` 创建子线程，子线程做耗时的下载图片操作，下载完成后，子线程通过 `EventHandler` 通知主线程，主线程再更新 UI。

2.2.1.1 基本概念

`EventRunner` 是一种事件循环器，循环处理从该 `EventRunner` 创建的新线程的事件队列中获取 `InnerEvent` 事件或者 `Runnable` 任务。`InnerEvent` 是 `EventHandler` 投递的事件。

`EventHandler` 是一种用户在当前线程上投递 `InnerEvent` 事件或者 `Runnable` 任务到异步线程上处理的机制。每一个 `EventHandler` 和指定的 `EventRunner` 所创建的新线程绑定，并且该新

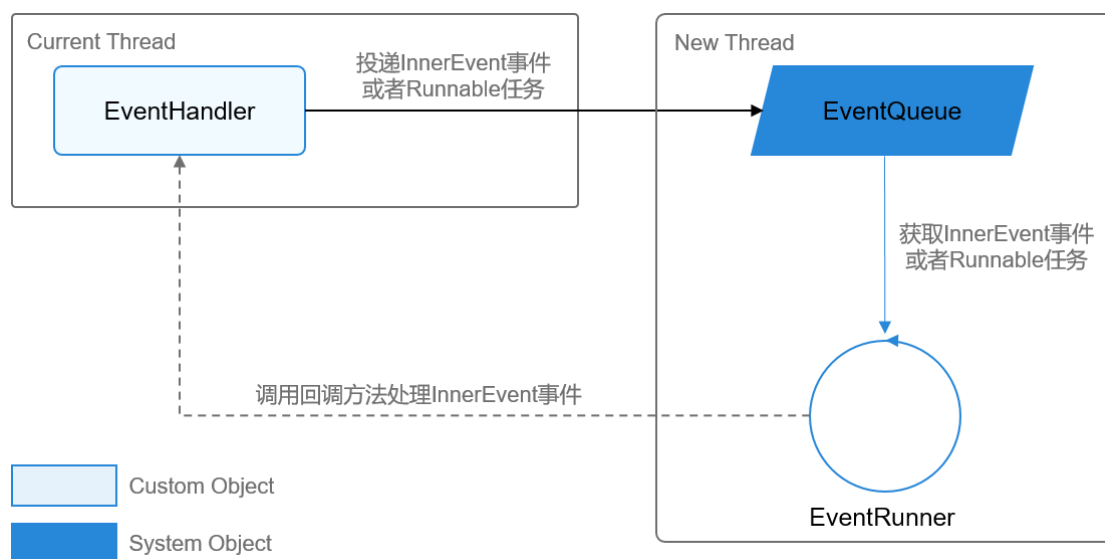
线程内部有一个事件队列。EventHandler 可以投递指定的 InnerEvent 事件或 Runnable 任务到这个事件队列。EventRunner 从事件队列里循环地取出事件，如果取出的事件是 InnerEvent 事件，将在 EventRunner 所在线程执行 processEvent 回调；如果取出的事件是 Runnable 任务，将在 EventRunner 所在线程执行 Runnable 的 run 回调。一般，EventHandler 有两个主要作用：

- 在不同线程间分发和处理 InnerEvent 事件或 Runnable 任务。
- 延迟处理 InnerEvent 事件或 Runnable 任务。

2.2.1.2 运作机制

EventHandler 的运作机制如下图所示：

图 1 EventHandler 的运作机制



使用 EventHandler 实现线程间通信的主要流程：

1. EventHandler 投递具体的 InnerEvent 事件或者 Runnable 任务到 EventRunner 所创建的线程的事件队列。
2. EventRunner 循环从事件队列中获取 InnerEvent 事件或者 Runnable 任务。
3. 处理事件或任务：
 - 如果 EventRunner 取出的事件为 InnerEvent 事件，则触发 EventHandler 的回调方法并触发 EventHandler 的处理方法，在新线程上处理该事件。
 - 如果 EventRunner 取出的事件为 Runnable 任务，则 EventRunner 直接在新线程上处理 Runnable 任务。

2.2.1.3 约束限制

- 在进行线程间通信的时候，EventHandler 只能和 EventRunner 所创建的线程进行绑定，EventRunner 创建时需要判断是否创建成功，只有确保获取的 EventRunner 实例非空时，才可以使用 EventHandler 绑定 EventRunner。
- 一个 EventHandler 只能同时与一个 EventRunner 绑定，一个 EventRunner 上可以创建多个 EventHandler。

2.2.2 开发指导

2.2.2.1 场景介绍

EventHandler 开发场景

EventHandler 的主要功能是将 InnerEvent 事件或者 Runnable 任务投递到其他的线程进行处理，其使用的场景包括：

- 开发者需要将 InnerEvent 事件投递到新的线程，按照优先级和延时进行处理。投递时，EventHandler 的优先级可在 IMMEDIATE、HIGH、LOW、IDLE 中选择，并设置合适的 delayTime。
- 开发者需要将 Runnable 任务投递到新的线程，并按照优先级和延时进行处理。投递时，EventHandler 的优先级可在 IMMEDIATE、HIGH、LOW、IDLE 中选择，并设置合适的 delayTime。
- 开发者需要在新创建的线程里投递事件到原线程进行处理。

EventRunner 工作模式

EventRunner 的工作模式可以分为托管模式和手动模式。两种模式是在调用 EventRunner 的 create()方法时，通过选择不同的参数来实现的，详见 API 参考。默认为托管模式。

- 托管模式：不需要开发者调用 run()和 stop()方法去启动和停止 EventRunner。当 EventRunner 实例化时，系统调用 run()来启动 EventRunner；当 EventRunner 不被引用时，系统调用 stop()来停止 EventRunner。
- 手动模式：需要开发者自行调用 EventRunner 的 run()方法和 stop()方法来确保线程的启动和停止。

2.2.2.2 接口说明

EventHandler

- EventHandler 的属性 Priority(优先级)介绍:

EventRunner 将根据优先级的高低从事件队列中获取事件或者 Runnable 任务进行处理。

属性	描述
Priority.IMMEDIATE	表示事件被立即投递
Priority.HIGH	表示事件先于 LOW 优先级投递
Priority.LOW	表示事件优于 IDLE 优先级投递，事件的默认优先级是 LOW
Priority.IDLE	表示在没有其他事件的情况下，才投递该事件

表 1 EventHandler 的属性

- EventHandler 的主要接口介绍:

接口名	描述
EventHandler(EventRunner runner)	利用已有的 EventRunner 来创建 EventHandler
current()	在 processEvent 回调中，获取当前的 EventHandler
processEvent(InnerEvent event)	回调处理事件，由开发者实现
sendEvent(InnerEvent event)	发送一个事件到事件队列，延时为 0ms，优先级为 LOW
sendEvent(InnerEvent event, long delayTime)	发送一个延时事件到事件队列，优先级为 LOW
sendEvent(InnerEvent event, long delayTime, EventHandler.Priority priority)	发送一个指定优先级的延时事件到事件队列
sendEvent(InnerEvent event, EventHandler.Priority priority)	发送一个指定优先级的事件到事件队列，延时为 0ms
sendSyncEvent(InnerEvent event)	发送一个同步事件到事件队列，延时为 0ms，优先级为 LOW

接口名	描述
sendSyncEvent(InnerEvent event, EventHandler.Priority priority)	发送一个指定优先级的同步事件到事件队列，延时为 0ms，优先级不可以是 IDLE
postSyncTask(Runnable task)	发送一个 Runnable 同步任务到事件队列，延时为 0ms，优先级为 LOW
postSyncTask(Runnable task, EventHandler.Priority priority)	发送一个指定优先级的 Runnable 同步任务到事件队列，延时为 0ms
postTask(Runnable task)	发送一个 Runnable 任务到事件队列，延时为 0ms，优先级为 LOW
postTask(Runnable task, long delayTime)	发送一个 Runnable 延时任务到事件队列，优先级为 LOW
postTask(Runnable task, long delayTime, EventHandler.Priority priority)	发送一个指定优先级的 Runnable 延时任务到事件队列
postTask(Runnable task, EventHandler.Priority priority)	发送一个指定优先级的 Runnable 任务到事件队列，延时为 0ms
sendTimingEvent(InnerEvent event, long taskTime)	发送一个定时事件到队列，在 taskTime 时间执行，如果 taskTime 小于当前时间，立即执行，优先级为 LOW
sendTimingEvent(InnerEvent event, long taskTime, EventHandler.Priority priority)	发送一个带优先级的事件到队列，在 taskTime 时间执行，如果 taskTime 小于当前时间，立即执行
postTimingTask(Runnable task, long taskTime)	发送一个 Runnable 任务到队列，在 taskTime 时间执行，如果 taskTime 小于当前时间，立即执行，优先级为 LOW
postTimingTask(Runnable task, long taskTime, EventHandler.Priority priority)	发送一个带优先级的 Runnable 任务到队列，在 taskTime 时间执行，如果 taskTime 小于当前时间，立即执行
removeEvent(int eventId)	删除指定 id 的事件
removeEvent(int eventId, long param)	删除指定 id 和 param 的事件
removeEvent(int eventId, long param, Object object)	删除指定 id、param 和 object 的事件

接口名	描述
removeAllEvent()	删除该 EventHandler 的所有事件
getEventName(InnerEvent event)	获取事件的名字
getEventRunner()	获取该 EventHandler 绑定的 EventRunner
isIdle()	判断队列是否为空
hasInnerEvent(Runnable runnable)	是否有还未被处理的这个任务

表 2 EventHandler 的主要接口

EventRunner

- EventRunner 的主要接口介绍：

接口名	描述
create()	创建一个拥有新线程的 EventRunner
create(boolean isDeposited)	创建一个拥有新线程的 EventRunner，isDeposited 为 true 时，EventRunner 为托管模式，系统将自动管理该 EventRunner；isDeposited 为 false 时，EventRunner 为手动模式。
create(String newThreadName)	创建一个拥有新线程的 EventRunner，新线程的名字是 newThreadName
current()	获取当前线程的 EventRunner
run()	EventRunner 为手动模式时，调用该方法启动新的线程
stop()	EventRunner 为手动模式时，调用该方法停止新的线程

表 3 EventRunner 主要接口

InnerEvent

- InnerEvent 的属性介绍：

属性	描述
eventId	事件的 ID, 由开发者定义用来辨别事件
object	事件携带的 Object 信息
param	事件携带的 long 型数据

表 4 InnerEvent 的属性

• InnerEvent 的主要接口介绍:

接口名	描述
drop()	释放一个事件实例
get()	获得一个事件实例
get(int eventId)	获得一个指定的 eventId 的事件实例
get(int eventId, long param)	获得一个指定的 eventId 和 param 的事件实例
get(int eventId, long param, Object object)	获得一个指定的 eventId, param 和 object 的事件实例
get(int eventId, Object object)	获得一个指定的 eventId 和 object 的事件实例
PacMap getPacMap()	获取 PacMap, 如果没有, 会新建一个
Runnable getTask()	获取 Runnable 任务
PacMap peekPacMap()	获取 PacMap
void setPacMap(PacMap pacMap)	设置 PacMap

表 5 InnerEvent 的接口

2.2.2.3 开发步骤

EventHandler 投递 InnerEvent 事件

EventHandler 投递 InnerEvent 事件，并按照优先级和延时进行处理，开发步骤如下：

1. 创建 EventHandler 的子类，在子类中重写实现方法 processEvent()来处理事件。

```
1. private class MyEventHandler extends EventHandler {
2.     private MyEventHandler(EventRunner runner) {
3.         super(runner);
4.     }
5.     // 重写实现 processEvent 方法
6.     @Override
7.     public void processEvent(InnerEvent event) {
8.         super.processEvent(event);
9.         if (event == null) {
10.             return;
11.         }
12.         int eventId = event.eventId;
13.         long param = event.param;
14.         switch (eventId | param) {
15.             case CASE1:
16.                 // 待执行的操作，由开发者定义
17.                 break;
18.             default:
19.                 break;
20.         }
21.     }
22. }
```

2. 创建 EventRunner，以手动模式为例。

```
1. EventRunner runner = EventRunner.create(false); // create()的参数是 true 时，则为托管模式
2. // 需要对 EventRunner 的实例进行校验，因为创建 EventRunner 可能失败，如创建线程失败时，创建 EventRunner 失败。
3. if (runner == null) {
4.     return;
5. }
```

3. 创建 EventHandler 子类的实例。

```
1. MyEventHandler myHandler = new MyEventHandler(runner);
```

4. 获取 InnerEvent 事件。

```
1. // 获取事件实例，其属性 eventId, param, object 由开发者确定，代码中只是示例。
2. int eventId1 = 0;
3. int eventId2 = 1;
4. long param = 0;
5. Object object = null;
6. InnerEvent event1 = InnerEvent.get(eventId1, param, object);
7. InnerEvent event2 = InnerEvent.get(eventId2, param, object);
```

5. 投递事件，投递的优先级以 IMMEDIATE 为例，延时选择 0ms 和 2ms。

```
1. // 优先级 immediate，投递之后立即处理，延时为 0ms，该语句等价于同步投递 sendSyncEvent(event1,
    EventHandler.Priority.immediate);
2. myHandler.sendEvent(event1, 0, EventHandler.Priority.IMMEDIATE);
3. myHandler.sendEvent(event2, 2, EventHandler.Priority.IMMEDIATE); // 延时 2ms 后立即处理
```

6. 启动和停止 EventRunner，如果为托管模式，则不需要此步骤。

```
1. runner.run();
2. //待执行操作
3. runner.stop();// 开发者根据业务需要在适当时机停止 EventRunner
```

EventHandler 投递 Runnable 任务

EventHandler 投递 Runnable 任务，并按照优先级和延时进行处理，开发步骤如下：

1. 创建 EventHandler 的子类，创建 EventRunner，并创建 EventHandler 子类的实例，步骤与 EventHandler 投递 InnerEvent 场景的步骤 1-3 相同。
2. 创建 Runnable 任务。

```
1. Runnable task1 = new Runnable() {
2.     @Override
3.     public void run() {
4.         // 待执行的操作，由开发者定义
5.     }
6. };
7. Runnable task2 = new Runnable() {
8.     @Override
9.     public void run() {
10.        // 待执行的操作，由开发者定义
11.    }
```

```
12.  };
```

3. 投递 Runnable 任务，投递的优先级以 IMMEDIATE 为例，延时选择 0ms 和 2ms。

```
1. //优先级为 immediate, 延时 0ms, 该语句等价于同步投递 myHandler.postSyncTask(task1, EventHandler.Priority.immediate);
2. myHandler.postTask(task1, 0, EventHandler.Priority.IMMEDIATE);
3.
4. myHandler.postTask(task2, 2, EventHandler.Priority.IMMEDIATE);// 延时 2ms 后立即执行
```

4. 启动和停止 EventRunner，如果是托管模式，则不需要此步骤。

```
1. runner.run();
2. //待执行操作
3.
4. runner.stop();// 停止 EventRunner
```

在新创建的线程里投递事件到原线程

EventHandler 从新创建的线程投递事件到原线程并进行处理，开发步骤如下：

1. 创建 EventHandler 的子类，在子类中重写实现方法 processEvent()来处理事件。

```
1. private class MyEventHandler extends EventHandler {
2.     private MyEventHandler(EventRunner runner) {
3.         super(runner);
4.     }
5.     // 重写实现 processEvent 方法
6.     @Override
7.     public void processEvent(InnerEvent event) {
8.         super.processEvent(event);
9.         if (event == null) {
10.             return;
11.         }
12.         int eventId = event.eventId;
13.         long param = event.param;
14.         Object object = event.object;
15.         switch (eventId | param) {
16.             case CASE1:
17.                 // 待执行的操作, 由开发者定义
18.                 break;
19.             case CASE2:
20.                 // 将原先线程的 EventRunner 实例投递给新创建的线程
```

```

21.         if (object instanceof EventRunner) {
22.             EventRunner runner2 = (EventRunner)object;
23.         }
24.         // 将原先线程的 EventRunner 实例与新创建的线程的 EventHandler 绑定
25.         EventHandler myHandler2 = new EventHandler(runner2) {
26.             @Override
27.             public void processEvent(InnerEvent event) {
28.                 //需要在原先线程执行的操作
29.             }
30.         };
31.         int eventId = 1;
32.         long param = 0;
33.         Object object = null;
34.         InnerEvent event2 = InnerEvent.get(eventId, param, object);
35.         myHandler2.sendEvent(event2); // 投递事件到原先的线程
36.         break;
37.     default:
38.         break;
39.     }
40. }
41. }

```

2. 创建 EventRunner，以手动模式为例。

```

1. EventRunner runner1 = EventRunner.create(false); // create()的参数是 true 时，则为托管模式。
2. // 需要对 EventRunner 的实例进行校验，不是任何线程都可以通过 create 创建，例如：当线程池已满时，不能再创建线程。
3. if (runner1 == null) {
4.     return;
5. }

```

3. 创建 EventHandler 子类的实例。

```

1. MyEventHandler myHandler1 = new MyEventHandler(runner1);

```

4. 获取 InnerEvent 事件。

```

1. // 获取事件实例，其属性 eventId, param, object 由开发者确定，代码中只是示例。
2. int eventId1 = 0;
3. long param = 0;
4. Object object = (Object) EventRunner.current();
5. InnerEvent event1 = InnerEvent.get(eventId1, param, object);

```

5. 投递事件，在新线程上直接处理。

```
1. // 将与当前线程绑定的 EventRunner 投递到与 runner1 创建的新线程中
2. myHandler.sendEvent(event1);
```

6. 启动和停止 EventRunner，如果是托管模式，则不需要此步骤。

```
1. runner.run();
2. //待执行操作
3.
4. runner.stop();// 停止 EventRunner
```

2.2.2.4 完整代码示例

- 非托管情况:

```
1. //全局:
2. EventRunner runnerA
3.
4. //线程 A:
5. runnerA = EventRunner.create(false);
6. runnerA.run(); // run 之后一直循环卡在这里，所以需要新建一个线程 run
7.
8. //线程 B:
9. //1.创建类继承 EventHandler
10. public class MyEventHandler extends EventHandler {
11.     public static int CODE_DOWNLOAD_FILE1;
12.     public static int CODE_DOWNLOAD_FILE2;
13.     public static int CODE_DOWNLOAD_FILE3;
14.     private MyEventHandler(EventRunner runner) {
15.         super(runner);
16.     }
17.
18.     @Override
19.     public void processEvent(InnerEvent event) {
20.         super.processEvent(event);
21.         if (event == null) {
22.             return;
23.         }
24.
25.         int eventId = event.eventId;
26.         if (STOP_EVENT_ID != eventId) {
```

```

27.         resultEventIdList.add(eventId);
28.     }
29.
30.     switch (eventId) {
31.         case CODE_DOWNLOAD_FILE1: {
32.             ... // your process
33.             break;
34.         }
35.         case CODE_DOWNLOAD_FILE1: {
36.             ... // your process
37.             break;
38.         }
39.         case CODE_DOWNLOAD_FILE1: {
40.             ... // your process
41.             break;
42.         }
43.         default:
44.             break;
45.     }
46. }
47. }
48.
49. //2.创建 MyEventHandler 实例
50. MyEventHandler handler = new MyEventHandler(runnerA);
51.
52. // 3.向线程 A 发送事件
53. handler.sendEvent(CODE_DOWNLOAD_FILE1);
54. handler.sendEvent(CODE_DOWNLOAD_FILE2);
55. handler.sendEvent(CODE_DOWNLOAD_FILE3);
56. .....
57.
58. // 4.runnerA 不再使用后，退出
59. runnerA.stop();

```

- 托管情况:

```

1. //1.创建 EventRunner A:
2. EventRunner runnerA = EventRunner.create("downloadRunner");// 内部会新建一个线程
3.
4. //2.创建类继承 EventHandler

```

```
5. public class MyEventHandler extends EventHandler {
6.     public static int CODE_DOWNLOAD_FILE1;
7.     public static int CODE_DOWNLOAD_FILE2;
8.     public static int CODE_DOWNLOAD_FILE3;
9.     private MyEventHandler(EventRunner runner) {
10.         super(runner);
11.     }
12.
13.     @Override
14.     public void processEvent(InnerEvent event) {
15.         super.processEvent(event);
16.         if (event == null) {
17.             return;
18.         }
19.
20.         int eventId = event.eventId;
21.         if (STOP_EVENT_ID != eventId) {
22.             resultEventIdList.add(eventId);
23.         }
24.
25.         switch (eventId) {
26.             case CODE_DOWNLOAD_FILE1: {
27.                 ... // your process
28.                 break;
29.             }
30.             case CODE_DOWNLOAD_FILE2: {
31.                 ... // your process
32.                 break;
33.             }
34.             case CODE_DOWNLOAD_FILE3: {
35.                 ... // your process
36.                 break;
37.             }
38.             default:
39.                 break;
40.         }
41.     }
42. }
```



```
43.  
44. //3.创建 MyEventHandler 实例  
45. MyEventHandler handler = new MyEventHandler(runnerA);  
46.  
47. //4.向线程 A 发送事件  
48. handler.sendEvent(CODE_DOWNLOAD_FILE1);  
49. handler.sendEvent(CODE_DOWNLOAD_FILE2);  
50. handler.sendEvent(CODE_DOWNLOAD_FILE3);  
51. ....  
52.  
53. //5.runnerA 没有任何对象引入时，线程会自动回收  
54. runnerA = null;
```

3 UI

3.1 Java UI 框架

3.1.1 概述

应用的 Ability 在屏幕上将显示一个用户界面，该界面用来显示所有可被用户查看和交互的内容。应用中所有的用户界面元素都是由 Component 和 ComponentContainer 对象构成。

Component 是绘制在屏幕上的一个对象，用户能与之交互。ComponentContainer 是一个用于容纳其他 Component 和 ComponentContainer 对象的容器。

Java UI 框架提供了一部分 Component 和 ComponentContainer 的具体子类，即创建用户界面（UI）的各类组件，包括一些常用的组件（比如：文本、按钮、图片、列表等）和常用的布局（比如：DirectionalLayout 和 DependentLayout）。用户可通过组件进行交互操作，并获得响应。所有的 UI 操作都应该在主线程进行设置。

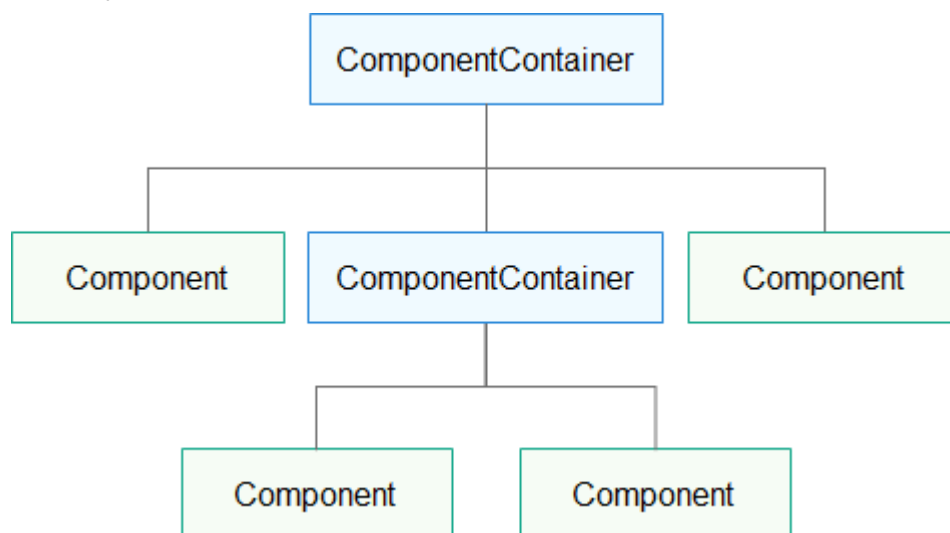
3.1.1.1 组件和布局

用户界面元素统称为组件，组件根据一定的层级结构进行组合形成布局。组件在未被添加到布局中时，既无法显示也无法交互，因此一个用户界面至少包含一个布局。在 UI 框架中，具体的布局类通常以 `XXLayout` 命名，完整的用户界面是一个布局，用户界面中的一部分也可以是一个布局。布局中容纳 `Component` 与 `ComponentContainer` 对象。

3.1.1.2 Component 和 ComponentContainer

- `Component`: 提供内容显示，是界面中所有组件的基类，开发者可以给 `Component` 设置事件处理回调来创建一个可交互的组件。Java UI 框架提供了一些常用的界面元素，也可称之为组件，组件一般直接继承 `Component` 或它的子类，如 `Text`、`Image` 等。
- `ComponentContainer`: 作为容器容纳 `Component` 或 `ComponentContainer` 对象，并对它们进行布局。Java UI 框架提供了一些标准布局功能的容器，它们继承自 `ComponentContainer`，一般以 “Layout” 结尾，如 `DirectionalLayout`、`DependentLayout` 等。

图 1 Component 结构

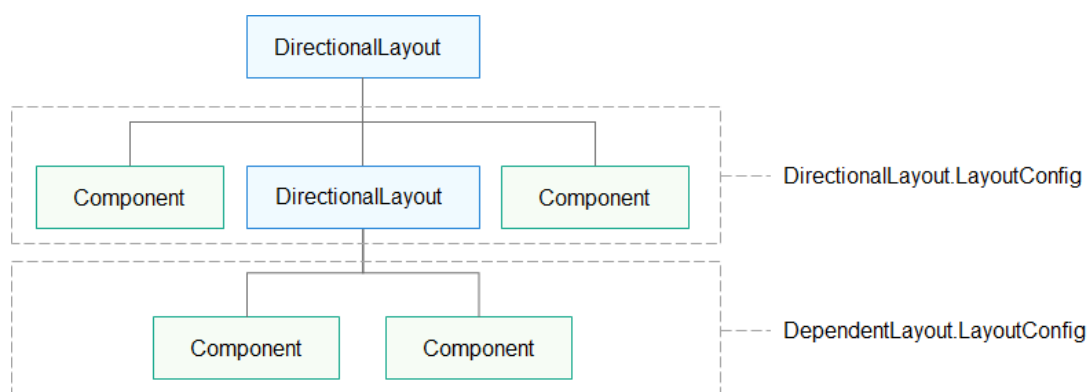


3.1.1.3 LayoutConfig

每种布局都根据自身特点提供 LayoutConfig 供子 Component 设定布局属性和参数，通过指定布局属性可以对子 Component 在布局中的显示效果进行约束。例如：“width”、

“height” 是最基本的布局属性，它们指定了组件的大小。

图 2 LayoutConfig



3.1.1.4 组件树

布局把 Component 和 ComponentContainer 以树状的层级结构进行组织，这样的—个布局就称为组件树。组件树的特点是仅有一个根组件，其他组件有且仅有一个父节点，组件之间的关系受到父节点的规则约束。

3.1.2 组件与布局开发指导

3.1.2.1 开发说明

HarmonyOS 提供了 Ability 和 AbilitySlice 两个基础类。有界面的 Ability 绑定了系统的

Window 进行 UI 展示，且具有[生命周期](#)。AbilitySlice 主要用于承载 Ability 的具体逻辑实现

和界面 UI，是应用显示、运行和跳转的最小单元。AbilitySlice 通过 setUIContent()为界面设置布局。

接口声明	接口描述
setUIContent(ComponentContainer root)	设置界面入口，root 为界面组件树根节点。

表 1 AbilitySlice 的 UI 接口

组件需要进行组合，并添加到界面的布局中。在 Java UI 框架中，提供了两种编写布局的方式：

- 在代码中创建布局：用代码创建 Component 和 ComponentContainer 对象，为这些对象设置合适的布局参数和属性值，并将 Component 添加到 ComponentContainer 中，从而创建出完整界面。
- 在 XML 中声明 UI 布局：按层级结构来描述 Component 和 ComponentContainer 的关系，给组件节点设定合适的布局参数和属性值，代码中可直接加载生成此布局。

这两种方式创建出的布局没有本质差别，在 XML 中声明布局，在加载后同样可在代码中对该布局进行修改。

组件分类

根据组件的功能，可以将组件分为布局类、显示类、交互类三类：

组件类别	组件名称	功能描述
布局类	PositionLayout、DirectionalLayout、StackLayout、DependentLayout、TableLayout、AdaptiveBoxLayout	提供了不同布局规范的组件容器，例如以单方向排列的 DirectionalLayout、以相对位置排列的 DependentLayout、以确切位置排列的 PositionLayout 等。

组 件 类 别	组件名称	功能描述
显 示 类	Text、Image、Clock、TickTimer、ProgressBar	提供了单纯的内容显示，例如用于文本显示的 Text，用于图像显示的 Image 等。
交 互 类	TextField、Button、Checkbox、RadioButton/RadioContainer、Switch、ToggleButton、Slider、Rating、ScrollView、TabList、ListContainer、PageSlider、PageFlipper、PageSliderIndicator、Picker、TimePicker、DatePicker、SurfaceProvider、ComponentProvider	提供了具体场景下与用户交互响应的功能，例如 Button 提供了点击响应功能，Slider 提供了进度选择功能等。

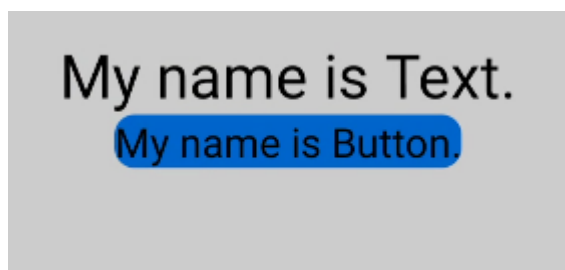
表 2 组件分类

框架提供的组件使应用界面开发更加便利，这些组件的具体功能说明及属性设置详见 API 参考。

3.1.2.2 代码创建布局

开发如下图所示界面，需要添加一个 Text 组件和 Button 组件。由于两个组件从上到下依次居中排列，可以选择使用竖向的 DirectionalLayout 布局来放置组件。

图 1 开发样例图



代码创建布局需要分别创建组件和布局，并将它们进行组织关联。

创建组件

1. 声明组件

```
1. Button button = new Button(context); // 参数 context 表示 AbilitySlice 的 Context 对象
```

2. 设置组件大小

```
1. button.setWidth(ComponentContainer.LayoutConfig.MATCH_CONTENT);
2. button.setHeight(ComponentContainer.LayoutConfig.MATCH_CONTENT);
```

3. 设置组件属性及 ID

```
1. button.setText("My name is Button.");
2. button.setTextSize(25);
3. button.setId(ID_BUTTON);
```

创建布局并使用

1. 声明布局

```
1. DirectionalLayout directionalLayout = new DirectionalLayout(context);
```

2. 设置布局大小

```
1. directionalLayout.setWidth(ComponentContainer.LayoutConfig.MATCH_PARENT);
2. directionalLayout.setHeight(ComponentContainer.LayoutConfig.MATCH_PARENT);
```

3. 设置布局属性及 ID

```
1. directionalLayout.setOrientation(Component.VERTICAL);
```

4. 将组件添加到布局中（视布局需要对组件设置布局属性进行约束）

```
1. directionalLayout.addComponent(button);
```

5. 将布局添加到组件树中

```
1. setUIContent(directionalLayout);
```

完整代码示例：

```
1. @Override
2. public void onStart(Intent intent) {
3.     super.onStart(intent);
4.     // 步骤 1 声明布局
```

```

5.     DirectionalLayout directionalLayout = new DirectionalLayout(context);
6.     // 步骤 2 设置布局大小
7.     directionalLayout.setWidth(ComponentContainer.LayoutConfig.MATCH_PARENT);
8.     directionalLayout.setHeight(ComponentContainer.LayoutConfig.MATCH_PARENT);
9.     // 步骤 3 设置布局属性及 ID (ID 视需要设置即可)
10.    directionalLayout.setOrientation(Component.VERTICAL);
11.    directionalLayout.setPadding(32, 32, 32, 32);
12.
13.    Text text = new Text(context);
14.    text.setText("My name is Text.");
15.    text.setTextSize(50);
16.    text.setId(100);
17.    // 步骤 4.1 为组件添加对应布局的布局属性
18.    DirectionalLayout.LayoutConfig layoutConfig = new DirectionalLayout.LayoutConfig(LayoutConfig.MATCH_CONTENT,
19.        LayoutConfig.MATCH_CONTENT);
20.    layoutConfig.alignment = LayoutAlignment.HORIZONTAL_CENTER;
21.    text.setLayoutConfig(layoutConfig);
22.
23.    // 步骤 4.2 将 Text 添加到布局中
24.    directionalLayout.addComponent(text);
25.
26.    // 类似的添加一个 Button
27.    Button button = new Button(context);
28.    layoutConfig.setMargins(0, 50, 0, 0);
29.    button.setLayoutConfig(layoutConfig);
30.    button.setText("My name is Button.");
31.    button.setTextSize(50);
32.    button.setId(100);
33.    ShapeElement background = new ShapeElement();
34.    background.setRgbColor(new RgbColor(0, 125, 255));
35.    background.setCornerRadius(25);
36.    button.setBackground(background);
37.    button.setPadding(10, 10, 10, 10);
38.    button.setClickListener(new Component.ClickedListener() {
39.        @Override
40.        // 在组件中增加对点击事件的检测
41.        public void onClick(Component Component) {
42.            // 此处添加按钮被点击需要执行的操作

```

```
43.     }
44.   });
45.   directionalLayout.addComponent(button);
46.
47.   // 步骤 5 将布局作为根布局添加到视图树中
48.   super.setUIContent(directionalLayout);
49. }
```

根据以上步骤创建组件和布局后的界面显示效果如[图 1](#)所示。其中，代码示例中为组件设置了一个按键回调，在按键被按下后，应用会执行自定义的操作。

在代码示例中，可以看到设置组件大小的方法有两种：

- 通过 `setWidth/setHeight` 直接设置宽高。
- 通过 `setLayoutConfig` 方法设置布局属性来设定宽高。

这两种方法的区别是后者还可以增加更多的布局属性设置，例如：使用“`alignment`”设置水平居中的约束。另外，这两种方法设置的宽高以最后设置的作为最终结果。它们的取值一致，可以是以下取值：

- 具体以像素为单位的数值。
- `MATCH_PARENT`：表示组件大小将扩展为父组件允许的最大值，它将占据父组件方向上的剩余大小。
- `MATCH_CONTENT`：表示组件大小与它内容占据的大小范围相适应。

3.1.2.3 XML 创建布局

XML 声明布局的方式更加简便直观。每一个 `Component` 和 `ComponentContainer` 对象大部分属性都支持在 XML 中进行设置，它们都有各自的 XML 属性列表。某些属性仅适用于特定的组件，例如：只有 `Text` 支持“`text_color`”属性，但不支持该属性的组件如果添加了该属性，该属性则会被忽略。具有继承关系的组件子类将继承父类的属性列表，`Component` 作为组件的基类，拥有各个组件常用的属性，比如：ID、布局参数等。

ID


```
1. ohos:id="$+id:text"
```

在 XML 中使用此格式声明一个对开发者友好的 ID，它会在编译过程中转换成一个常量。尤其在 DependentLayout 布局中，组件之间需要描述相对位置关系，描述时要通过 ID 来指定对应组件。

布局中的组件通常要设置独立的 ID，以便在程序中查找该组件。如果布局中有不同组件设置了相同的 ID，在通过 ID 查找组件时会返回查找到的第一个组件，因此尽量保证在所要查找的布局中为组件设置独立的 ID 值，避免出现与预期不符合的问题。

布局参数

```
1. ohos:width="20vp"  
2. ohos:height="10vp"
```

与代码中设置组件的宽度和高度类似，在 XML 中它们的取值可以是：

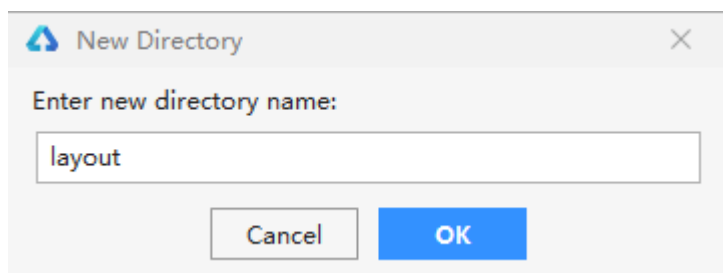
- 具体的数值：10（以像素为单位）、10vp（以屏幕相对像素为单位）。
- MATCH_PARENT：表示组件大小将扩展为父组件允许的最大值，它将占据父组件方向上的剩余大小，在 XML 中用“match_parent”表示。
- MATCH_CONTENT：表示组件大小与它的内容占据的大小范围相适应，在 XML 中用数值“match_content”表示。

更多的组件属性列表可参考组件的 XML 属性文档。

创建 XML 布局文件

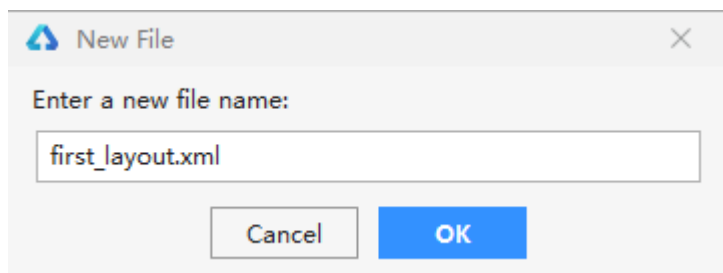
1. 在 DevEco Studio 的“Project”窗口，打开“entry > src > main > resources > base”，右键点击“base”文件夹，选择“New > Directory”，命名为“layout”。

图 1 设置 Directory 名称



2. 右键点击“layout”文件夹，选择“New > File”，命名为“first_layout.xml”。

图 2 设置 File 名称



修改 XML 布局文件

打开新创建的 first_layout.xml 布局文件，修改其中的内容，对布局和组件的属性和层级进行描述。

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="match_parent"
6.     ohos:orientation="vertical"
7.     ohos:padding="32">
8.     <Text
9.         ohos:id="$+id:text"
10.        ohos:width="match_content"
11.        ohos:height="match_content"
12.        ohos:layout_alignment="horizontal_center"
13.        ohos:text="My name is Text."
14.        ohos:text_size="25vp"/>
15.     <Button
16.         ohos:id="$+id:button"
17.         ohos:width="match_content"

```

```
18.     ohos:height="match_content"
19.     ohos:layout_alignment="horizontal_center"
20.     ohos:text="My name is Button."
21.     ohos:text_size="50"/>
22. </DirectionalLayout>
```

加载 XML 布局

在代码中需要加载 XML 布局，并添加为根布局或作为其他布局的子 Component。

```
1. @Override
2. public void onStart(Intent intent) {
3.     super.onStart(intent);
4.     // 加载 XML 布局作为根布局
5.     super.setUIContent(ResourceTable.Layout_first_layout);
6.     // 查找布局中组件
7.     Button button = (Button) findComponentById(ResourceTable.Id_button);
8.     if (button != null) {
9.         // 设置组件的属性
10.        ShapeElement background = new ShapeElement();
11.        background.setRgbColor(new RgbColor(0,125,255));
12.        background.setCornerRadius(25);
13.        button.setBackground(background);
14.
15.        button.setClickedListener(new Component.ClickedListener() {
16.            @Override
17.            // 在组件中增加对点击事件的检测
18.            public void onClick(Component Component) {
19.                // 此处添加按钮被点击需要执行的操作
20.            }
21.        });
22.    }
23. }
```

3.1.3 常用组件开发指导

3.1.3.1 Text

文本 (Text) 是用来显示字符串的组件，在界面上显示为一块文本区域。Text 作为一个基本组件，有很多扩展，常见的有按钮组件 Button，文本编辑组件 TextField。

使用 Text

- 创建 Text

```
1. <Text
2.     ohos:id="$+id:text"
3.     ohos:width="match_content"
4.     ohos:height="match_content"
5.     ohos:text="Text"
6.     ohos:background_element="$graphic:color_gray_element"/>
```

color_gray_element.xml:

```
7. <?xml version="1.0" encoding="utf-8"?>
8. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
9.     ohos:shape="rectangle">
10.     <solid
11.         ohos:color="#ff888888"/>
12. </shape>
```

图 1 创建一个 Text

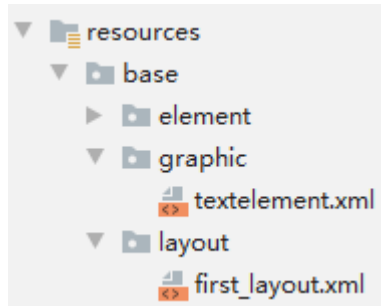


- 设置背景

常用的背景如常见的文本背景、按钮背景，可以采用 XML 格式放置在 [graphic](#) 目录下。

在 “Project” 窗口，打开 “entry > src > main > resources > base”，右键点击 “base” 文件夹，选择 “New > Directory”，命名为 “graphic”。右键点击 “graphic” 文件夹，选择 “New > File”，命名为 “textelement.xml”。

图 2 创建 textelement.xml 文件后的 resources 目录结构



在 textelement.xml 中定义文本的背景:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <corners
5.         ohos:radius="20"/>
6.     <solid
7.         ohos:color="#ff888888"/>
8. </shape>
    
```

在 first_layout.xml 中引用上面定义的文本背景:

```

9. <Text
10.     ohos:id="$+id:text"
11.     ohos:width="match_content"
12.     ohos:height="match_content"
13.     ohos:text="Text"
14.     ohos:background_element="$graphic:textelement"/>
    
```

- 设置字体大小和颜色

```

1. <Text
2.     ohos:id="$+id:text"
3.     ohos:width="match_content"
4.     ohos:height="match_content"
5.     ohos:text="Text"
6.     ohos:text_size="28fp"
7.     ohos:text_color="blue"
    
```

```
8.   ohos:left_margin="15vp"  
9.   ohos:bottom_margin="15vp"  
10.  ohos:right_padding="15vp"  
11.  ohos:left_padding="15vp"  
12.  ohos:background_element="$graphic:textelement"/>
```

图 3 设置字体大小和颜色的效果



- 设置字体风格和字重

```
1.  <Text  
2.   ohos:id="$+id:text"  
3.   ohos:width="match_content"  
4.   ohos:height="match_content"  
5.   ohos:text="Text"  
6.   ohos:text_size="28fp"  
7.   ohos:text_color="blue"  
8.   ohos:italic="true"  
9.   ohos:text_weight="700"  
10.  ohos:text_font="serif"  
11.  ohos:left_margin="15vp"  
12.  ohos:bottom_margin="15vp"  
13.  ohos:right_padding="15vp"  
14.  ohos:left_padding="15vp"  
15.  ohos:background_element="$graphic:textelement"/>
```

图 4 设置字体风格和字重的效果



- 设置文本对齐方式

```
1.  <Text
```

```
2.   ohos:id="$+id:text"  
3.   ohos:width="300vp"  
4.   ohos:height="100vp"  
5.   ohos:text="Text"  
6.   ohos:text_size="28fp"  
7.   ohos:text_color="blue"  
8.   ohos:italic="true"  
9.   ohos:text_weight="700"  
10.  ohos:text_font="serif"  
11.  ohos:left_margin="15vp"  
12.  ohos:bottom_margin="15vp"  
13.  ohos:right_padding="15vp"  
14.  ohos:left_padding="15vp"  
15.  ohos:text_alignment="horizontal_center|bottom"  
16.  ohos:background_element="$graphic:textelement"/>
```

图 5 设置文本对齐方式的效果



- 设置文本换行和最大显示行数

```
1.  <Text  
2.   ohos:id="$+id:text"  
3.   ohos:width="75vp"  
4.   ohos:height="match_content"  
5.   ohos:text="TextText"  
6.   ohos:text_size="28fp"  
7.   ohos:text_color="blue"  
8.   ohos:italic="true"  
9.   ohos:text_weight="700"  
10.  ohos:text_font="serif"
```

```

11.   ohos:multiple_lines="true"
12.   ohos:max_text_lines="2"
13.   ohos:background_element="$graphic:textelement"/>

```

图 6 设置文本换行和最大显示行数的效果



自动调节字体大小

Text 对象支持根据文本长度自动调整文本的字体大小和换行。

1. 设置自动换行、最大显示行数和自动调节字体大小。

```

1.   <Text
2.     ohos:id="$+id:text1"
3.     ohos:width="90vp"
4.     ohos:height="match_content"
5.     ohos:min_height="30vp"
6.     ohos:text="T"
7.     ohos:text_color="blue"
8.     ohos:italic="true"
9.     ohos:text_weight="700"
10.    ohos:text_font="serif"
11.    ohos:multiple_lines="true"
12.    ohos:max_text_lines="1"
13.    ohos:auto_font_size="true"
14.    ohos:right_padding="8vp"
15.    ohos:left_padding="8vp"
16.    ohos:background_element="$graphic:textelement"/>

```


2. 通过 `setAutoFontSizeRule` 设置自动调整规则，三个入参分别是最小的字体大小、最大的字体大小、每次调整文本字体大小的步长。

```
1. // 设置自动调整规则
2. text.setAutoFontSizeRule(30, 100, 1);
3. // 设置点击一次增多一个"T"
4. text.setClickListener(new Component.ClickedListener() {
5.     @Override
6.     public void onClick(Component Component) {
7.         text.setText(text.getText() + "T");
8.     }
9. });
```

图 7 自动调节字体大小



跑马灯效果

当文本过长时，可以设置跑马灯效果，实现文本滚动显示。前提是文本换行关闭且最大显示行数为 1，默认情况下即可满足前提要求。

```
1. <Text
2.     ohos:id="$+id:text"
3.     ohos:width="75vp"
4.     ohos:height="match_content"
5.     ohos:text="TextText"
6.     ohos:text_size="28fp"
7.     ohos:text_color="blue"
8.     ohos:italic="true"
9.     ohos:text_weight="700"
10.    ohos:text_font="serif"
11.    ohos:background_element="$graphic:textelement"/>
1. // 跑马灯效果
2. text.setTruncationMode(Text.TruncationMode.AUTO_SCROLLING);
```

3. `// 启动跑马灯效果`
4. `text.startAutoScrolling();`

图 8 跑马灯效果



场景示例

利用文本组件实现一个标题栏和详细内容的界面。

图 9 界面效果



源码示例：

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <DependentLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="match_content"
6.     ohos:background_element="$graphic:color_light_gray_element">
7.     <Text
8.         ohos:id="$+id:text1"
9.         ohos:width="match_parent"
10.        ohos:height="match_content"
11.        ohos:text_size="25fp"
```

```
12.     ohos:top_margin="15vp"
13.     ohos:left_margin="15vp"
14.     ohos:right_margin="15vp"
15.     ohos:background_element="$graphic:textelement"
16.     ohos:text="Title"
17.     ohos:text_weight="1000"
18.     ohos:text_alignment="horizontal_center"/>
19. <Text
20.     ohos:id="$+id:text3"
21.     ohos:width="match_parent"
22.     ohos:height="120vp"
23.     ohos:text_size="25fp"
24.     ohos:background_element="$graphic:textelement"
25.     ohos:text="Content"
26.     ohos:top_margin="15vp"
27.     ohos:left_margin="15vp"
28.     ohos:right_margin="15vp"
29.     ohos:bottom_margin="15vp"
30.     ohos:text_alignment="center"
31.     ohos:below="$id:text1"
32.     ohos:text_font="serif"/>
33. <Button
34.     ohos:id="$+id:button1"
35.     ohos:width="75vp"
36.     ohos:height="match_content"
37.     ohos:text_size="15fp"
38.     ohos:background_element="$graphic:textelement"
39.     ohos:text="Previous"
40.     ohos:right_margin="15vp"
41.     ohos:bottom_margin="15vp"
42.     ohos:left_padding="5vp"
43.     ohos:right_padding="5vp"
44.     ohos:below="$id:text3"
45.     ohos:left_of="$id:button2"
46.     ohos:text_font="serif"/>
47. <Button
48.     ohos:id="$+id:button2"
49.     ohos:width="75vp"
```

```
50.     ohos:height="match_content"
51.     ohos:text_size="15fp"
52.     ohos:background_element="$graphic:textelement"
53.     ohos:text="Next"
54.     ohos:right_margin="15vp"
55.     ohos:bottom_margin="15vp"
56.     ohos:left_padding="5vp"
57.     ohos:right_padding="5vp"
58.     ohos:align_parent_end="true"
59.     ohos:below="$id:text3"
60.     ohos:text_font="serif"/>
61. </DependentLayout>
```

color_light_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffeeeeee"/>
6. </shape>
```

textelement.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <corners
5.         ohos:radius="20"/>
6.     <solid
7.         ohos:color="#ff888888"/>
8. </shape>
```

3.1.3.2 Button

按钮（Button）是一种常见的组件，点击可以触发对应的操作，通常由文本或图标组成，也可以由图标和文本共同组成。

图 1 文本按钮



图 2 图标按钮



图 3 图标和文本共同组成的按钮



创建 Button

使用 Button 组件，可以生成形状、颜色丰富的按钮。

```
1. <Button
2.     ohos:id="$+id:button_sample"
3.     ohos:width="match_content"
4.     ohos:height="match_content"
5.     ohos:text_size="27fp"
6.     ohos:text="button"
7.     ohos:background_element="$graphic:button_element"
8.     ohos:left_margin="15vp"
9.     ohos:bottom_margin="15vp"
10.    ohos:right_padding="8vp"
11.    ohos:left_padding="8vp"
```

```

12.     ohos:element_left="$graphic.ic_btn_reload"
13. />

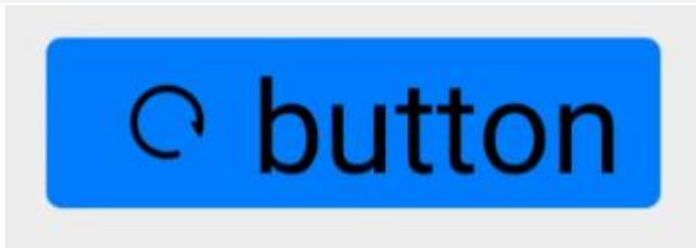
```

button_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <corners
5.         ohos:radius="10"/>
6.     <solid
7.         ohos:color="#FF007DFF"/>
8. </shape>

```



响应点击事件

按钮的重要作用是在用户单击按钮时，会执行相应的操作或者界面出现相应的变化。实际上用户单击按钮时，Button 对象将收到一个点击事件。开发者可以自定义响应点击事件的方法。例如，通过创建一个 Component.ClickedListener 对象，然后通过调用 setClickedListener 将其分配给按钮。

```

1. //从定义的 xml 中获取 Button 对象
2. Button button = (Button) rootLayout.findViewById(ResourceTable.Id_button_sample);
3. // 为按钮设置点击事件回调
4. button.setClickedListener(new Component.ClickedListener() {
5.     public void onClick(Component v) {
6.         // 此处添加点击按钮后的事件处理逻辑
7.     }
8. });

```

不同类型的按钮

按照按钮的形状，按钮可以分为：普通按钮，椭圆按钮，胶囊按钮，圆形按钮等。

- 普通按钮



普通按钮和其他按钮的区别在于不需要设置任何形状，只设置文本和背景颜色即可，例如：

```

1. <Button
2.     ohos:width="150vp"
3.     ohos:height="50vp"
4.     ohos:text_size="27fp"
5.     ohos:text="button"
6.     ohos:background_element="$graphic:color_blue_element"
7.     ohos:left_margin="15vp"
8.     ohos:bottom_margin="15vp"
9.     ohos:right_padding="8vp"
10.    ohos:left_padding="8vp"
11. />
    
```

color_blue_element.xml:

```

12. <?xml version="1.0" encoding="utf-8"?>
13. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
14.     ohos:shape="rectangle">
15.     <solid
16.         ohos:color="#FF007DFF"/>
17. </shape>
    
```

- 椭圆按钮



椭圆按钮是通过设置 background_element 来实现的，background_element 的 shape 设

置为椭圆 (oval)，例如：

```

1. <Button
2.     ohos:width="150vp"
3.     ohos:height="50vp"
4.     ohos:text_size="27fp"
5.     ohos:text="button"
6.     ohos:background_element="$graphic:oval_button_element"
7.     ohos:left_margin="15vp"
8.     ohos:bottom_margin="15vp"
9.     ohos:right_padding="8vp"
10.    ohos:left_padding="8vp"
11.    ohos:element_left="$graphic:ic_btn_reload"
12. />

```

oval_button_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="oval">
4.     <solid
5.         ohos:color="#FF007DFF"/>
6. </shape>

```

- **胶囊按钮**



胶囊按钮是一种常见的按钮，设置按钮背景时将背景设置为矩形形状，并且设置

ShapeElement 的 radius 的半径，例如：

```

1. <Button
2.     ohos:id="$+id:button"
3.     ohos:width="match_content"
4.     ohos:height="match_content"
5.     ohos:text_size="27fp"
6.     ohos:text="button"
7.     ohos:background_element="$graphic:capsule_button_element"
8.     ohos:left_margin="15vp"
9.     ohos:bottom_margin="15vp"
10.    ohos:right_padding="15vp"

```

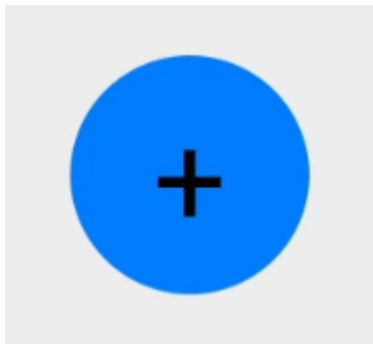


```
11.     ohos:left_padding="15vp"
12. />
```

capsule_button_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <corners
5.         ohos:radius="100"/>
6.     <solid
7.         ohos:color="#FF007DFF"/>
8. </shape>
```

- 圆形按钮



圆形按钮和椭圆按钮的区别在于组件本身的宽度和高度需要相同，例如：

```
1. <Button
2.     ohos:id="$+id:button4"
3.     ohos:width="50vp"
4.     ohos:height="50vp"
5.     ohos:text_size="27fp"
6.     ohos:background_element="$graphic:circle_button_element"
7.     ohos:text="+"
8.     ohos:left_margin="15vp"
9.     ohos:bottom_margin="15vp"
10.    ohos:right_padding="15vp"
11.    ohos:left_padding="15vp"
12. />
```

circle_button_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
```

```

3.   ohos:shape="oval">
4.   <solid
5.     ohos:color="#FF007DFF"/>
6. </shape>

```

场景示例

利用圆形按钮，胶囊按钮，文本组件可以绘制出如下拨号盘的 UI 界面。

图 4 界面效果



源码示例：

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout
3.   xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.   ohos:width="match_parent"
5.   ohos:height="match_parent"
6.   ohos:background_element="$graphic:color_light_gray_element"
7.   ohos:orientation="vertical">
8.   <Text
9.     ohos:width="match_content"
10.    ohos:height="match_content"
11.    ohos:text_size="20fp"
12.    ohos:text="0123456789"
13.    ohos:background_element="$graphic:green_text_element"

```

```
14.     ohos:text_alignment="center"
15.     ohos:layout_alignment="horizontal_center"
16. />
17. <DirectionalLayout
18.     ohos:width="match_parent"
19.     ohos:height="match_content"
20.     ohos:alignment="horizontal_center"
21.     ohos:orientation="horizontal"
22.     ohos:top_margin="5vp"
23.     ohos:bottom_margin="5vp">
24.     <Button
25.         ohos:width="40vp"
26.         ohos:height="40vp"
27.         ohos:text_size="15fp"
28.         ohos:background_element="$graphic:green_circle_button_element"
29.         ohos:text="1"
30.         ohos:text_alignment="center"
31.     />
32.     <Button
33.         ohos:width="40vp"
34.         ohos:height="40vp"
35.         ohos:text_size="15fp"
36.         ohos:background_element="$graphic:green_circle_button_element"
37.         ohos:text="2"
38.         ohos:left_margin="5vp"
39.         ohos:right_margin="5vp"
40.         ohos:text_alignment="center"
41.     />
42.     <Button
43.         ohos:width="40vp"
44.         ohos:height="40vp"
45.         ohos:text_size="15fp"
46.         ohos:background_element="$graphic:green_circle_button_element"
47.         ohos:text="3"
48.         ohos:text_alignment="center"
49.     />
50. </DirectionalLayout>
51. <DirectionalLayout
```

```

52.     ohos:width="match_parent"
53.     ohos:height="match_content"
54.     ohos:alignment="horizontal_center"
55.     ohos:orientation="horizontal"
56.     ohos:bottom_margin="5vp">
57.     <Button
58.         ohos:width="40vp"
59.         ohos:height="40vp"
60.         ohos:text_size="15fp"
61.         ohos:background_element="$graphic:green_circle_button_element"
62.         ohos:text="4"
63.         ohos:text_alignment="center"
64.     />
65.     <Button
66.         ohos:width="40vp"
67.         ohos:height="40vp"
68.         ohos:text_size="15fp"
69.         ohos:left_margin="5vp"
70.         ohos:right_margin="5vp"
71.         ohos:background_element="$graphic:green_circle_button_element"
72.         ohos:text="5"
73.         ohos:text_alignment="center"
74.     />
75.     <Button
76.         ohos:width="40vp"
77.         ohos:height="40vp"
78.         ohos:text_size="15fp"
79.         ohos:background_element="$graphic:green_circle_button_element"
80.         ohos:text="6"
81.         ohos:text_alignment="center"
82.     />
83. </DirectionalLayout>
84. <DirectionalLayout
85.     ohos:width="match_parent"
86.     ohos:height="match_content"
87.     ohos:alignment="horizontal_center"
88.     ohos:orientation="horizontal"
89.     ohos:bottom_margin="5vp">

```

```
90.     <Button
91.         ohos:width="40vp"
92.         ohos:height="40vp"
93.         ohos:text_size="15fp"
94.         ohos:background_element="$graphic:green_circle_button_element"
95.         ohos:text="7"
96.         ohos:text_alignment="center"
97.     />
98.     <Button
99.         ohos:width="40vp"
100.        ohos:height="40vp"
101.        ohos:text_size="15fp"
102.        ohos:left_margin="5vp"
103.        ohos:right_margin="5vp"
104.        ohos:background_element="$graphic:green_circle_button_element"
105.        ohos:text="8"
106.        ohos:text_alignment="center"
107.    />
108.    <Button
109.        ohos:width="40vp"
110.        ohos:height="40vp"
111.        ohos:text_size="15fp"
112.        ohos:background_element="$graphic:green_circle_button_element"
113.        ohos:text="9"
114.        ohos:text_alignment="center"
115.    />
116. </DirectionalLayout>
117. <DirectionalLayout
118.     ohos:width="match_parent"
119.     ohos:height="match_content"
120.     ohos:alignment="horizontal_center"
121.     ohos:orientation="horizontal"
122.     ohos:bottom_margin="5vp">
123.     <Button
124.         ohos:width="40vp"
125.         ohos:height="40vp"
126.         ohos:text_size="15fp"
127.         ohos:background_element="$graphic:green_circle_button_element"
```

```

128.         ohos:text="*"
129.         ohos:text_alignment="center"
130.     />
131.     <Button
132.         ohos:width="40vp"
133.         ohos:height="40vp"
134.         ohos:text_size="15fp"
135.         ohos:left_margin="5vp"
136.         ohos:right_margin="5vp"
137.         ohos:background_element="$graphic:green_circle_button_element"
138.         ohos:text="0"
139.         ohos:text_alignment="center"
140.     />
141.     <Button
142.         ohos:width="40vp"
143.         ohos:height="40vp"
144.         ohos:text_size="15fp"
145.         ohos:background_element="$graphic:green_circle_button_element"
146.         ohos:text="#"
147.         ohos:text_alignment="center"
148.     />
149. </DirectionalLayout>
150. <Button
151.     ohos:width="match_content"
152.     ohos:height="match_content"
153.     ohos:text_size="15fp"
154.     ohos:text="CALL"
155.     ohos:background_element="$graphic:green_capsule_button_element"
156.     ohos:bottom_margin="5vp"
157.     ohos:text_alignment="center"
158.     ohos:layout_alignment="horizontal_center"
159.     ohos:left_padding="10vp"
160.     ohos:right_padding="10vp"
161.     ohos:top_padding="2vp"
162.     ohos:bottom_padding="2vp"
163. />
164. </DirectionalLayout>

```

color_light_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffeeeeee"/>
6. </shape>
```

green_text_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <corners
5.         ohos:radius="20"/>
6.     <stroke
7.         ohos:width="2"
8.         ohos:color="#ff008B00"/>
9.     <solid
10.        ohos:color="#ffeeeeee"/>
11. </shape>
```

green_circle_button_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="oval">
4.     <stroke
5.         ohos:width="5"
6.         ohos:color="#ff008B00"/>
7.     <solid
8.         ohos:color="#ffeeeeee"/>
9. </shape>
```

green_capsule_button_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
```

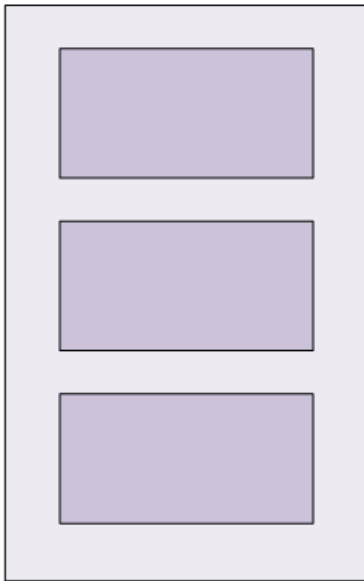
```
3.   ohos:shape="rectangle">
4.   <corners
5.     ohos:radius="100"/>
6.   <solid
7.     ohos:color="#ff008B00"/>
8. </shape>
```

3.1.4 常用布局开发指导

3.1.4.1 DirectionalLayout

DirectionalLayout 是 Java UI 中的一种重要组件布局，用于将一组组件(Component)按照水平或者垂直方向排布，能够方便地对齐布局内的组件。该布局和其他布局的组合，可以实现更加丰富的布局方式。

图 1 DirectionalLayout 示意图



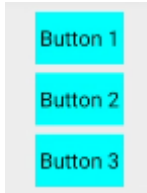
排列方式

DirectionalLayout 的排列方向 (orientation) 分为水平 (horizontal) 或者垂直 (vertical) 方向。使用 orientation 设置布局内组件的排列方式，默认为垂直排列。

- 垂直排列

垂直方向排列三个按钮，效果如下：

图 2 三个垂直排列的按钮



```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <DirectionalLayout
3.      xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.      ohos:width="match_parent"
5.      ohos:height="match_content"
6.      ohos:orientation="vertical">
7.      <Button
8.          ohos:width="33vp"
9.          ohos:height="20vp"
10.         ohos:bottom_margin="3vp"
11.         ohos:left_margin="13vp"
12.         ohos:background_element="$graphic:color_cyan_element"
13.         ohos:text="Button 1"/>
14.      <Button
15.          ohos:width="33vp"
16.          ohos:height="20vp"
17.          ohos:bottom_margin="3vp"
18.          ohos:left_margin="13vp"
19.          ohos:background_element="$graphic:color_cyan_element"
20.          ohos:text="Button 2"/>
21.      <Button
22.          ohos:width="33vp"
23.          ohos:height="20vp"
24.          ohos:bottom_margin="3vp"
25.          ohos:left_margin="13vp"
26.          ohos:background_element="$graphic:color_cyan_element"
27.          ohos:text="Button 3"/>
28.  </DirectionalLayout>
    
```

color_cyan_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ff"/>
6. </shape>
    
```

- **水平排列**

水平方向排列三个按钮，效果如下：

图 3 三个水平排列的按钮



```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="match_content"
6.     ohos:orientation="horizontal">
7.     <Button
8.         ohos:width="33vp"
9.         ohos:height="20vp"
10.        ohos:left_margin="13vp"
11.        ohos:background_element="$graphic:color_cyan_element"
12.        ohos:text="Button 1"/>
13.     <Button
14.         ohos:width="33vp"
15.         ohos:height="20vp"
16.         ohos:left_margin="13vp"
17.         ohos:background_element="$graphic:color_cyan_element"
18.         ohos:text="Button 2"/>
19.     <Button
20.         ohos:width="33vp"
21.         ohos:height="20vp"
22.         ohos:left_margin="13vp"
23.         ohos:background_element="$graphic:color_cyan_element"
24.         ohos:text="Button 3"/>
25. </DirectionalLayout>
    
```

color_cyan_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>
    
```

DirectionalLayout 不会自动换行，其子组件会按照设定的方向依次排列，若超过布局本身的大小，超出布局大小的部分将不会被显示，例如：

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="20vp"
6.     ohos:orientation="horizontal">
7.     <Button
8.         ohos:width="166vp"
9.         ohos:height="match_content"
10.        ohos:left_margin="13vp"
11.        ohos:background_element="$graphic:color_cyan_element"
12.        ohos:text="Button 1"/>
13.     <Button
14.         ohos:width="166vp"
15.         ohos:height="match_content"
16.         ohos:left_margin="13vp"
17.         ohos:background_element="$graphic:color_cyan_element"
18.         ohos:text="Button 2"/>
19.     <Button
20.         ohos:width="166vp"
21.         ohos:height="match_content"
22.         ohos:left_margin="13vp"
23.         ohos:background_element="$graphic:color_cyan_element"
24.         ohos:text="Button 3"/>
25. </DirectionalLayout>
    
```

color_cyan_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>
    
```

此布局包含了三个按钮，但由于 DirectionalLayout 不会自动换行，超出布局大小的组件部分无法显示。界面显示如下：

图 4 DirectionalLayout 不自动换行示例



对齐方式

DirectionalLayout 中的组件使用 layout_alignment 控制自身在布局中的对齐方式，当对齐方

式与[排列方式](#)方向一致时，对齐方式不会生效，如设置了水平方向的排列方式，则左对齐、右

对齐将不会生效。常用的对齐参数见[表 1](#)。

参数	作用	可搭配排列方式
left	左对齐	垂直排列
top	顶部对齐	水平排列
right	右对齐	垂直排列
bottom	底部对齐	水平排列
horizontal_center	水平方向居中	垂直排列
vertical_center	垂直方向居中	水平排列
center	垂直与水平方向都居中	水平/垂直排列

表 1 常用的对齐参数

三种对齐方式的示例代码：

```

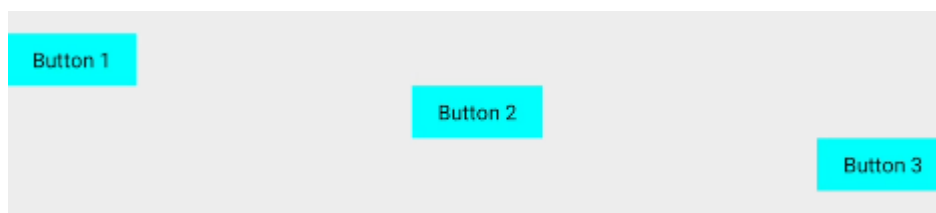
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <DirectionalLayout
3.      xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.      ohos:width="match_parent"
5.      ohos:height="60vp">
6.      <Button
7.          ohos:width="50vp"
8.          ohos:height="20vp"
9.          ohos:background_element="$graphic:color_cyan_element"
10.         ohos:layout_alignment="left"
11.         ohos:text="Button 1"/>
12.     <Button
13.         ohos:width="50vp"
14.         ohos:height="20vp"
15.         ohos:background_element="$graphic:color_cyan_element"
16.         ohos:layout_alignment="horizontal_center"
17.         ohos:text="Button 2"/>
18.     <Button
19.         ohos:width="50vp"
20.         ohos:height="20vp"
21.         ohos:background_element="$graphic:color_cyan_element"
22.         ohos:layout_alignment="right"
23.         ohos:text="Button 3"/>
24. </DirectionalLayout>
    
```

color_cyan_element.xml：

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.      ohos:shape="rectangle">
4.      <solid
5.          ohos:color="#ff0fffff"/>
6.  </shape>
    
```

图 5 三种对齐方式效果示例



权重

权重 (weight) 就是按比例来分配组件占用父组件的大小，在水平布局下计算公式为：

父布局可分配宽度=父布局宽度-所有子组件 width 之和；

组件宽度=组件 weight/所有组件 weight 之和*父布局可分配宽度；

实际使用过程中，建议使用 width=0 来按比例分配父布局的宽度，1:1:1 效果如下：



```
1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="match_content"
6.     ohos:orientation="horizontal">
7.     <Button
8.         ohos:width="0vp"
9.         ohos:height="20vp"
10.        ohos:weight="1"
11.        ohos:background_element="$graphic:color_cyan_element"
12.        ohos:text="Button 1"/>
13.     <Button
14.         ohos:width="0vp"
15.         ohos:height="20vp"
16.         ohos:weight="1"
17.         ohos:background_element="$graphic:color_gray_element"
```

```
18.     ohos:text="Button 2"/>
19.     <Button
20.         ohos:width="0vp"
21.         ohos:height="20vp"
22.         ohos:weight="1"
23.         ohos:background_element="$graphic:color_cyan_element"
24.         ohos:text="Button 3"/>
25. </DirectionalLayout>
```

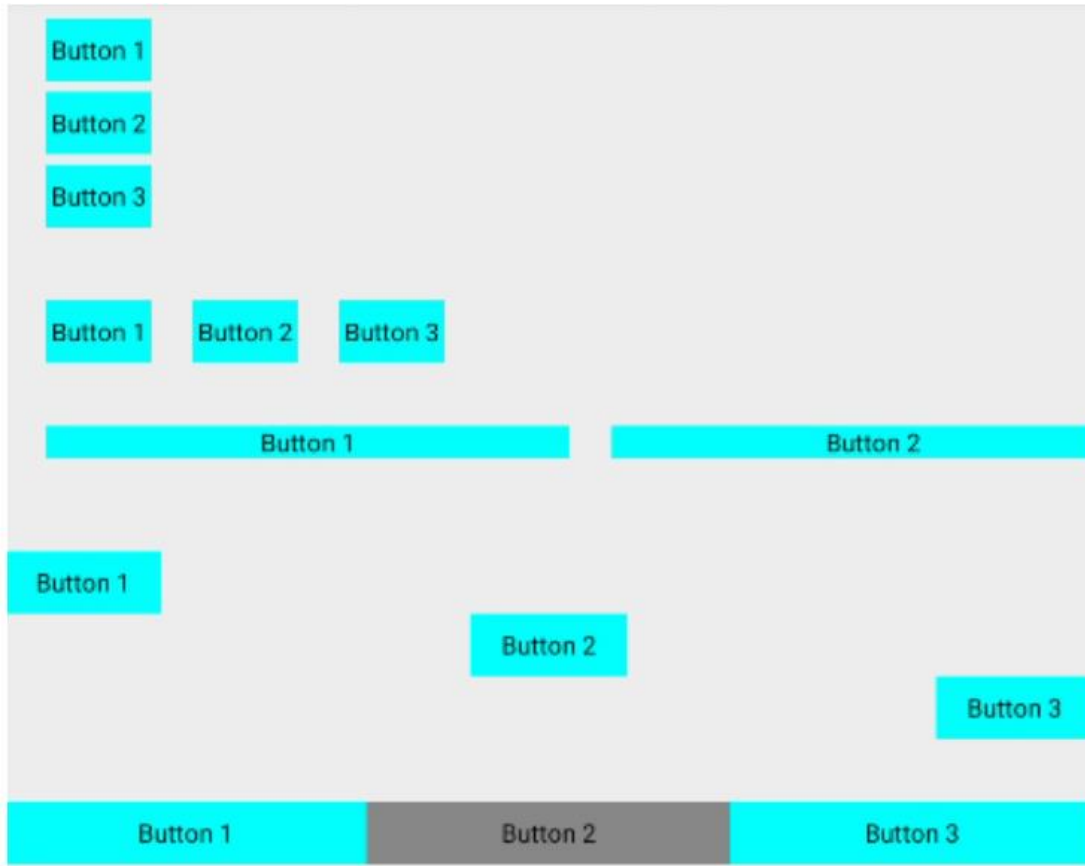
color_cyan_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>
```

color_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff888888"/>
6. </shape>
```

场景示例



源码示例：

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <DirectionalLayout xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:width="match_parent"
4.     ohos:height="match_parent"
5.     ohos:background_element="$graphic:color_light_gray_element">
6.     <DirectionalLayout
7.         ohos:width="match_parent"
8.         ohos:height="match_content"
9.         ohos:orientation="vertical">
10.        <Button
11.            ohos:width="33vp"
12.            ohos:height="20vp"
13.            ohos:bottom_margin="3vp"
14.            ohos:left_margin="13vp"
15.            ohos:background_element="$graphic:color_cyan_element"
```



```

16.         ohos:text="Button 1"/>
17.     <Button
18.         ohos:width="33vp"
19.         ohos:height="20vp"
20.         ohos:bottom_margin="3vp"
21.         ohos:left_margin="13vp"
22.         ohos:background_element="$graphic:color_cyan_element"
23.         ohos:text="Button 2"/>
24.     <Button
25.         ohos:width="33vp"
26.         ohos:height="20vp"
27.         ohos:bottom_margin="3vp"
28.         ohos:left_margin="13vp"
29.         ohos:background_element="$graphic:color_cyan_element"
30.         ohos:text="Button 3"/>
31. </DirectionalLayout>
32. <Component ohos:height="20vp"/>
33. <DirectionalLayout
34.     ohos:width="match_parent"
35.     ohos:height="match_content"
36.     ohos:orientation="horizontal">
37.     <Button
38.         ohos:width="33vp"
39.         ohos:height="20vp"
40.         ohos:left_margin="13vp"
41.         ohos:background_element="$graphic:color_cyan_element"
42.         ohos:text="Button 1"/>
43.     <Button
44.         ohos:width="33vp"
45.         ohos:height="20vp"
46.         ohos:left_margin="13vp"
47.         ohos:background_element="$graphic:color_cyan_element"
48.         ohos:text="Button 2"/>
49.     <Button
50.         ohos:width="33vp"
51.         ohos:height="20vp"
52.         ohos:left_margin="13vp"
53.         ohos:background_element="$graphic:color_cyan_element"

```

```
54.         ohos:text="Button 3"/>
55.     </DirectionalLayout>
56.     <Component ohos:height="20vp"/>
57.     <DirectionalLayout
58.         ohos:width="match_parent"
59.         ohos:height="20vp"
60.         ohos:orientation="horizontal" >
61.         <Button
62.             ohos:width="166vp"
63.             ohos:height="match_content"
64.             ohos:left_margin="13vp"
65.             ohos:background_element="$graphic:color_cyan_element"
66.             ohos:text="Button 1"/>
67.         <Button
68.             ohos:width="166vp"
69.             ohos:height="match_content"
70.             ohos:left_margin="13vp"
71.             ohos:background_element="$graphic:color_cyan_element"
72.             ohos:text="Button 2"/>
73.         <Button
74.             ohos:width="166vp"
75.             ohos:height="match_content"
76.             ohos:left_margin="13vp"
77.             ohos:background_element="$graphic:color_cyan_element"
78.             ohos:text="Button 3"/>
79.     </DirectionalLayout>
80. <Component ohos:height="20vp"/>
81. <DirectionalLayout
82.     ohos:width="match_parent"
83.     ohos:height="60vp" >
84.     <Button
85.         ohos:width="50vp"
86.         ohos:height="20vp"
87.         ohos:background_element="$graphic:color_cyan_element"
88.         ohos:layout_alignment="left"
89.         ohos:text="Button 1"/>
90.     <Button
91.         ohos:width="50vp"
```

```

92.         ohos:height="20vp"
93.         ohos:background_element="$graphic:color_cyan_element"
94.         ohos:layout_alignment="horizontal_center"
95.         ohos:text="Button 2"/>
96.     <Button
97.         ohos:width="50vp"
98.         ohos:height="20vp"
99.         ohos:background_element="$graphic:color_cyan_element"
100.        ohos:layout_alignment="right"
101.        ohos:text="Button 3"/>
102. </DirectionalLayout>
103. <Component ohos:height="20vp"/>
104. <DirectionalLayout
105.     ohos:width="match_parent"
106.     ohos:height="match_content"
107.     ohos:orientation="horizontal">
108.     <Button
109.         ohos:width="0vp"
110.         ohos:height="20vp"
111.         ohos:weight="1"
112.         ohos:background_element="$graphic:color_cyan_element"
113.         ohos:text="Button 1"/>
114.     <Button
115.         ohos:width="0vp"
116.         ohos:height="20vp"
117.         ohos:weight="1"
118.         ohos:background_element="$graphic:color_gray_element"
119.         ohos:text="Button 2"/>
120.     <Button
121.         ohos:width="0vp"
122.         ohos:height="20vp"
123.         ohos:weight="1"
124.         ohos:background_element="$graphic:color_cyan_element"
125.         ohos:text="Button 3"/>
126. </DirectionalLayout>
127. </DirectionalLayout>

```

color_light_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffeeeeee"/>
6. </shape>
```

color_cyan_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>
```

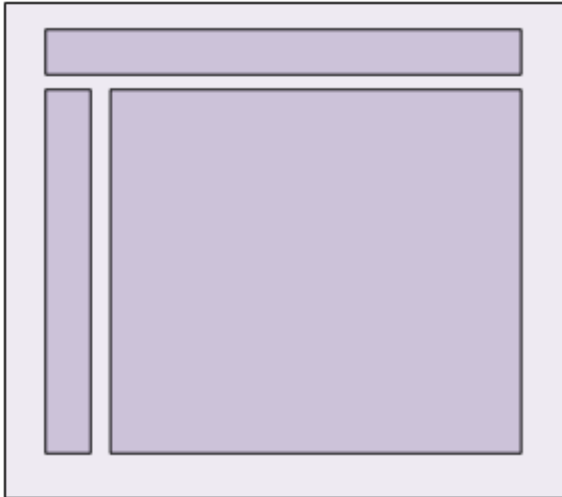
color_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff888888"/>
6. </shape>
```

3.1.4.2 DependentLayout

DependentLayout 是 Java UI 系统里的一种常见布局。与 DirectionalLayout 相比，拥有更多的排布方式，每个组件可以指定相对于其他同级元素的位置，或者指定相对于父组件的位置。

图 1 DependentLayout 示意图



排列方式

DependentLayout 的排列方式是相对于其他同级组件或者父组件的位置进行布局。

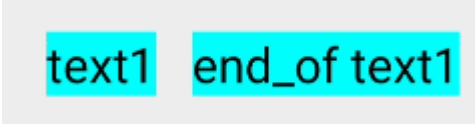
- 相对于同级组件

相对于同级组件的位置布局见表 1。

位置布局	描述
above	处于同级组件的上侧。
below	处于同级组件的下侧。
start_of	处于同级组件的起始侧。
end_of	处于同级组件的结束侧。
left_of	处于同级组件的左侧。
right_of	处于同级组件的右侧。

表 1 相对于同级组件的位置布局

end_of:



```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <DependentLayout
3.      xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.      ohos:width="match_content"
5.      ohos:height="match_content"
6.      ohos:background_element="$graphic:color_light_gray_element">
7.      <Text
8.          ohos:id="$+id:text1"
9.          ohos:width="match_content"
10.         ohos:height="match_content"
11.         ohos:left_margin="15vp"
12.         ohos:top_margin="15vp"
13.         ohos:bottom_margin="15vp"
14.         ohos:text="text1"
15.         ohos:text_size="20fp"
16.         ohos:background_element="$graphic:color_cyan_element"/>
17.     <Text
18.         ohos:id="$+id:text2"
19.         ohos:width="match_content"
20.         ohos:height="match_content"
21.         ohos:left_margin="15vp"
22.         ohos:top_margin="15vp"
23.         ohos:right_margin="15vp"
24.         ohos:bottom_margin="15vp"
25.         ohos:text="end_of text1"
26.         ohos:text_size="20fp"
27.         ohos:background_element="$graphic:color_cyan_element"
28.         ohos:end_of="$id:text1"/>
29. </DependentLayout>
    
```

color_light_gray_element.xml:

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.      ohos:shape="rectangle">
4.      <solid
5.          ohos:color="#ffeeeeee"/>
    
```

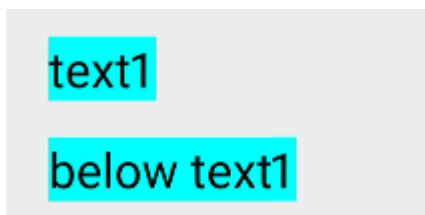
6. `</shape>`

color_cyan_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>
    
```

below:



```

1. <?xml version="1.0" encoding="utf-8"?>
2. <DependentLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_content"
5.     ohos:height="match_content"
6.     ohos:background_element="$graphic:color_light_gray_element">
7.     <Text
8.         ohos:id="$+id:text1"
9.         ohos:width="match_content"
10.        ohos:height="match_content"
11.        ohos:left_margin="15vp"
12.        ohos:top_margin="15vp"
13.        ohos:right_margin="40vp"
14.        ohos:text="text1"
15.        ohos:text_size="20fp"
16.        ohos:background_element="$graphic:color_cyan_element"/>
17.     <Text
18.         ohos:id="$+id:text3"
19.         ohos:width="match_content"
20.         ohos:height="match_content"
21.         ohos:left_margin="15vp"
22.         ohos:top_margin="15vp"
23.         ohos:right_margin="40vp"
    
```

```

24.     ohos:bottom_margin="15vp"
25.     ohos:text="below text1"
26.     ohos:text_size="20fp"
27.     ohos:background_element="$graphic:color_cyan_element"
28.     ohos:below="$id:text1"/>
29. </DependentLayout>

```

color_light_gray_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffeeeeee"/>
6. </shape>

```

color_cyan_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff00ffff"/>
6. </shape>

```

其他的 above、start_of、left_of、right_of 等参数可分别实现类似的布局。

- **相对于父组件**

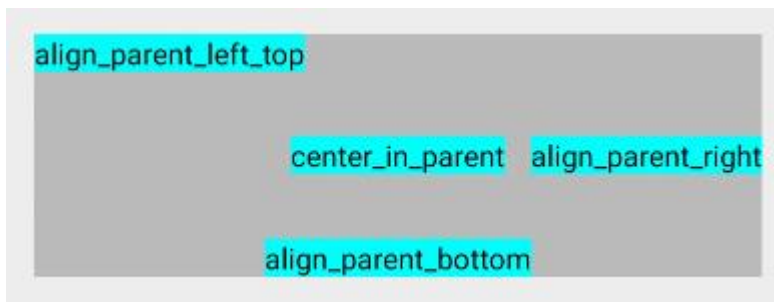
相对于父组件的位置布局见表 2。

位置布局	描述
align_parent_left	处于父组件的左侧。
align_parent_right	处于父组件的右侧。
align_parent_start	处于父组件的起始侧。
align_parent_end	处于父组件的结束侧。
align_parent_top	处于父组件的上侧。

位置布局	描述
align_parent_bottom	处于父组件的下侧。
center_in_parent	处于父组件的中间。

表 2 相对于父组件的位置布局

以上位置布局可以组合，形成处于左上角、左下角、右上角、右下角的布局。



```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <DependentLayout
3.      xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.      ohos:width="300vp"
5.      ohos:height="100vp"
6.      ohos:background_element="$graphic:color_background_gray_element" >
7.      <Text
8.          ohos:id="$+id:text6"
9.          ohos:width="match_content"
10.         ohos:height="match_content"
11.         ohos:text="align_parent_right"
12.         ohos:text_size="12fp"
13.         ohos:background_element="$graphic:color_cyan_element"
14.         ohos:align_parent_right="true"
15.         ohos:center_in_parent="true"/>
16.     <Text
17.         ohos:id="$+id:text7"
18.         ohos:width="match_content"
19.         ohos:height="match_content"
20.         ohos:text="align_parent_bottom"
21.         ohos:text_size="12fp"
22.         ohos:background_element="$graphic:color_cyan_element"

```

```

23.     ohos:align_parent_bottom="true"
24.     ohos:center_in_parent="true"/>
25. <Text
26.     ohos:id="$+id:text8"
27.     ohos:width="match_content"
28.     ohos:height="match_content"
29.     ohos:text="center_in_parent"
30.     ohos:text_size="12fp"
31.     ohos:background_element="$graphic:color_cyan_element"
32.     ohos:center_in_parent="true"/>
33. <Text
34.     ohos:id="$+id:text9"
35.     ohos:width="match_content"
36.     ohos:height="match_content"
37.     ohos:text="align_parent_left_top"
38.     ohos:text_size="12fp"
39.     ohos:background_element="$graphic:color_cyan_element"
40.     ohos:align_parent_left="true"
41.     ohos:align_parent_top="true"/>
42. </DependentLayout>

```

color_background_gray_element.xml:

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffbbbbbb"/>
6. </shape>

```

color_cyan_element.xml:

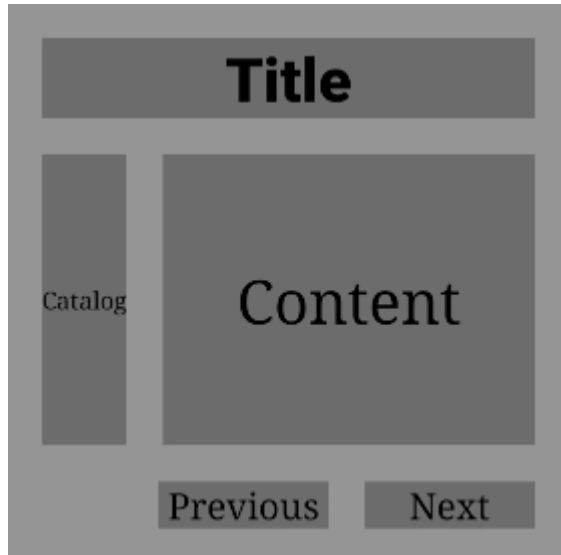
```

1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff0fffff"/>
6. </shape>

```

场景示例

使用 `DependentLayout` 可以轻松实现内容丰富的布局。



```
1. <?xml version="1.0" encoding="utf-8"?>
2. <DependentLayout
3.     xmlns:ohos="http://schemas.huawei.com/res/ohos"
4.     ohos:width="match_parent"
5.     ohos:height="match_content"
6.     ohos:background_element="$graphic:color_background_gray_element">
7.     <Text
8.         ohos:id="$+id:text1"
9.         ohos:width="match_parent"
10.        ohos:height="match_content"
11.        ohos:text_size="25fp"
12.        ohos:top_margin="15vp"
13.        ohos:left_margin="15vp"
14.        ohos:right_margin="15vp"
15.        ohos:background_element="$graphic:color_gray_element"
16.        ohos:text="Title"
17.        ohos:text_weight="1000"
18.        ohos:text_alignment="horizontal_center"
19.    />
20.    <Text
21.        ohos:id="$+id:text2"
22.        ohos:width="match_content"
```

```

23.     ohos:height="120vp"
24.     ohos:text_size="10vp"
25.     ohos:background_element="$graphic:color_gray_element"
26.     ohos:text="Catalog"
27.     ohos:top_margin="15vp"
28.     ohos:left_margin="15vp"
29.     ohos:right_margin="15vp"
30.     ohos:bottom_margin="15vp"
31.     ohos:align_parent_left="true"
32.     ohos:text_alignment="center"
33.     ohos:multiple_lines="true"
34.     ohos:below="$id:text1"
35.     ohos:text_font="serif"/>
36. <Text
37.     ohos:id="$+id:text3"
38.     ohos:width="match_parent"
39.     ohos:height="120vp"
40.     ohos:text_size="25fp"
41.     ohos:background_element="$graphic:color_gray_element"
42.     ohos:text="Content"
43.     ohos:top_margin="15vp"
44.     ohos:right_margin="15vp"
45.     ohos:bottom_margin="15vp"
46.     ohos:text_alignment="center"
47.     ohos:below="$id:text1"
48.     ohos:end_of="$id:text2"
49.     ohos:text_font="serif"/>
50. <Button
51.     ohos:id="$+id:button1"
52.     ohos:width="70vp"
53.     ohos:height="match_content"
54.     ohos:text_size="15fp"
55.     ohos:background_element="$graphic:color_gray_element"
56.     ohos:text="Previous"
57.     ohos:right_margin="15vp"
58.     ohos:bottom_margin="15vp"
59.     ohos:below="$id:text3"
60.     ohos:left_of="$id:button2"

```

```
61.     ohos:italic="false"
62.     ohos:text_weight="5"
63.     ohos:text_font="serif"/>
64. <Button
65.     ohos:id="$+id:button2"
66.     ohos:width="70vp"
67.     ohos:height="match_content"
68.     ohos:text_size="15fp"
69.     ohos:background_element="$graphic:color_gray_element"
70.     ohos:text="Next"
71.     ohos:right_margin="15vp"
72.     ohos:bottom_margin="15vp"
73.     ohos:align_parent_end="true"
74.     ohos:below="$id:text3"
75.     ohos:italic="false"
76.     ohos:text_weight="5"
77.     ohos:text_font="serif"/>
78. </DependentLayout>
```

color_background_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ffbbbbbb"/>
6. </shape>
```

color_gray_element.xml:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
3.     ohos:shape="rectangle">
4.     <solid
5.         ohos:color="#ff888888"/>
6. </shape>
```

3.1.5 动画开发指导

动画是组件的基础特性之一，精心设计的动画使 UI 变化更直观，有助于改进应用程序的外观并改善用户体验。Java UI 框架提供了数值动画（AnimatorValue）和属性动画（AnimatorProperty），并提供了将多个动画同时操作的动画集合（AnimatorGroup）。

数值动画（AnimatorValue）

AnimatorValue 数值从 0 到 1 变化，本身与 Component 无关。开发者可以设置 0 到 1 变化过程的属性，例如：时长、变化曲线、重复次数等，并通过值的变化改变组件的属性，实现组件的动画效果。

1. 声明 AnimatorValue。

```
1. AnimatorValue animator = new AnimatorValue();
```

2. 设置变化属性。

```
1. animator.setDuration(2000);
2. animator.setDelay(1000);
3. animator.setLoopedCount(2);
4. animator.setCurveType(Animator.CurveType.BOUNCE);
```

3. 添加回调事件。

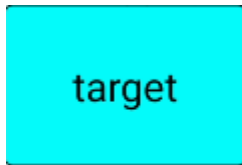
```
1. animator.setValueUpdateListener(new AnimatorValue.ValueUpdateListener() {
2.     @Override
3.     public void onUpdate(AnimatorValue animatorValue, float value) {
4.         button.setContentPosition((int) (800 * value), button.getContentPositionY());
5.     }
6. });
```

4. 启动动画或对动画做其他操作。

```
1. animator.start();
```

AnimatorGroup 动画效果如图所示：

图 1 数值动画效果



属性动画 (AnimatorProperty)

为 Component 的属性设置动画是非常常见的需求，Java UI 框架可以为 Component 设置某一个属性或多个属性的动画。

1. 声明 AnimatorProperty。

```
1. AnimatorProperty animator = button.createAnimatorProperty();
```

2. 设置变化属性，可链式调用。

```
1. animator.moveFromX(50).moveToX(1000).rotate(180).alpha(0).setDuration(2500).setDelay(500).setLoopedCount(5);
```

3. 启动动画或对动画做其他操作。

```
1. animator.start();
```

可以使用 `setTarget()` 改变关联的 Component 对象。

```
2. animator.setTarget(button2);
```

动画效果如图所示：

图 2 属性动画效果



动画集合 (AnimatorGroup)

如果需要使用一个组合动画，可以把多个动画对象进行组合，并添加到使用 AnimatorGroup 中。AnimatorGroup 提供了两个方法：runSerially() 和 runParallel()，分别表示动画按顺序开始和动画同时开始。

1. 声明 AnimatorGroup。

```
1. AnimatorGroup animatorGroup = new AnimatorGroup();
```

2. 添加要按顺序或同时开始的动画。

```
1. // 4 个动画按顺序播放
2. animatorGroup.runSerially(am1, am2, am3, am4);
3. // 4 个动画同时播放
4. animatorGroup.runParallel(am1, am2, am3, am4);
```

3. 启动动画或对动画做其他操作。

```
1. animatorGroup.start();
```

为了更加灵活处理多个动画的播放顺序，例如一些动画顺序播放，一些动画同时播放，Java UI 框架提供了更方便的动画 Builder 接口：

1. 声明 AnimatorGroup Builder。

```
1. AnimatorGroup.Builder animatorGroupBuilder = animatorGroup.build();
```

2. 按播放顺序添加多个动画。

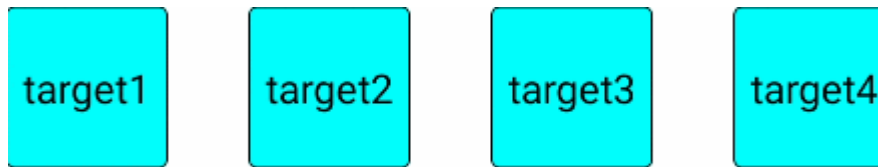
```
1. // 4 个动画的顺序为: am1 -> am2/am3 -> am4
2. animatorGroupBuilder.addAnimators(am1).addAnimators(am2, am3).addAnimators(am4)
```

3. 启动动画或对动画做其他操作。

```
1. animatorGroup.start();
```

动画集合的动画效果如下：

图 3 动画集合效果



3.1.6 可见即可说开发指导

可见即可说是要求 Component 中通过与热词关联，从而达到指定的效果。例如：在浏览图片时，说出图片的名字或角标序号，从而实现打开图片的效果。

说明

该功能目前仅在智慧屏产品上支持。

热词注册

开发者首先需要进行 Component 的热词注册，即告诉设备，哪些热词是这个 Component 所需要响应的。

1. 构建 Component.VoiceEvent 对象，需要设置热词，中英文都可以。

```
1. Component.VoiceEvent eventKeys = new Component.VoiceEvent("ok");
```

2. 如果一个 Component 的同一 VoiceEvent 存在多个热词匹配，可以通过 addSynonyms 方法增加 eventKeys 的热词。

```
1. eventKeys.addSynonyms("确定");
```

3. 当 Component.VoiceEvent 对象操作完成后，使用 Component 的 subscribeVoiceEvents 方法来发起注册。

```
1. Component.subscribeVoiceEvents(eventKeys);
```

4. 如果一个 Component 有多个事件需要响应，需要创建不同的事件来进行注册。

事件响应

开发者完成热词注册后，需要关注的是对应于不同热词所需要处理的事件。事件响应回调的

SpeechEvent 对象仅包含一个热词。

1. 首先需要实现 SpeechEventListener 接口。

```
1. private Component.SpeechEventListener speechEventListener = new Component.SpeechEventListener(  
2.     @Override  
3.     public boolean onSpeechEvent(Component v, SpeechEvent event) {  
4.         if (event.getActionProperty().equals("ok")) {  
5.             ... // 检测注册的热词，进行相应的处理  
6.         }  
7.     });
```

2. 通过 setSpeechEventListener 方法实现回调注册。

```
1. Component.setSpeechEventListener(speechEventListener);
```

3.2 JS UI 框架

3.2.1 概述

JS UI 框架是一种跨设备的高性能 UI 开发框架，支持声明式编程和跨设备多态 UI。

阅读本开发指南前，开发者需要掌握以下基础知识：

- HTML5
- CSS
- JavaScript

基础能力

- 声明式编程

JS UI 框架采用类 HTML 和 CSS 声明式编程语言作为页面布局和页面样式的开发语言，页面业务逻辑则支持 ECMAScript 规范的 JavaScript 语言。JS UI 框架提供的声明式编程，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。

- **跨设备**

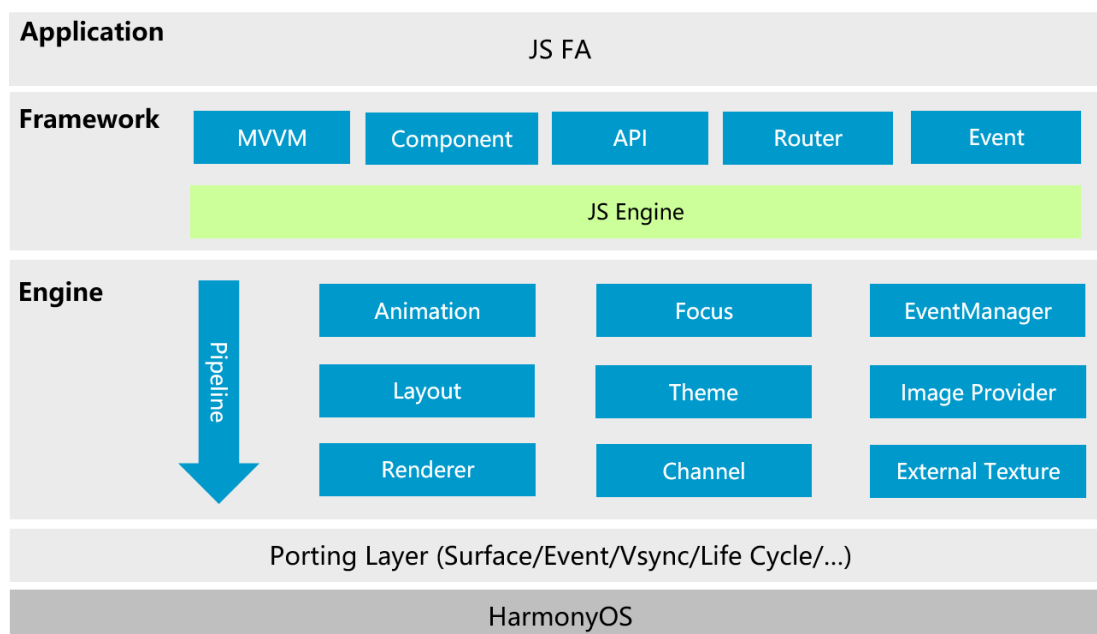
开发框架架构上支持 UI 跨设备显示能力，运行时自动映射到不同设备类型，开发者无感知，降低开发者多设备适配成本。

- **高性能**

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

整体架构

JS UI 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)。



- **Application**

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

Java 开发 FA 应用请参考 Java UI 框架。

- **Framework**

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

- **Engine**

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

- **Porting Layer**

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

3.2.2 初步体验 JS FA 应用

3.2.2.1 JS FA 概述

JS UI 框架支持纯 JavaScript、JavaScript 和 Java 混合语言开发。JS FA 指基于 JavaScript 或 JavaScript 和 Java 混合开发的 FA，下面主要介绍：JS FA 在 HarmonyOS 上运行时需要的基类 `AceAbility`、加载 JS FA 主体的方法、JS FA 开发目录。

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，代码示例如下：

```
1. public class MainAbility extends AceAbility {  
2.     @Override  
3.     public void onStart(Intent intent) {  
4.         super.onStart(intent);  
}
```

```
5.     }
6.
7.     @Override
8.     public void onStop() {
9.         super.onStop();
10.    }
11. }
```

如何加载 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` 窗口进行显示以及全局应用生命周期管理。

`setInstanceName(String name)`的参数 “name” 指实例名称，实例名称与 `config.json` 文

件中 `profile.application.js.name` 的值对应。若开发者未修改实例名，而使用了缺省值

`default`，则无需调用此接口。若开发者修改了实例名，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。

说明

多实例应用的 `profile.application.js` 字段中有多个实例项，使用时请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，代码示例如下：

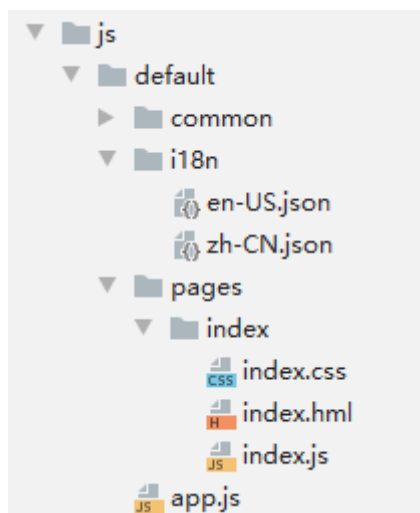
```
1. public class MainAbility extends AceAbility {
2.     @Override
3.     public void onStart(Intent intent) {
4.         setInstanceName("JSComponentName"); // config.json 配置文件中 ability.js.name 的标签值。
5.         super.onStart(intent);
6.     }
7. }
```

说明

需在 `super.onStart(Intent)`前调用此接口。

JS FA 开发目录

新建工程的 JS 目录如下图所示。



在工程目录中：common 文件夹主要存放公共资源，如图片、视频等；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。

- **main > js > default > i18n > en-US.json**: 此文件定义了英文模式下页面显示的变量内容。同理，zh-CN.json 中定义了中文模式下的页面内容。

```
1. {
2.   "strings": {
3.     "hello": "Hello",
4.     "world": "World"
5.   },
6.   "files": {
7.   }
8. }
```

- **main > js > default > pages > index > index.html**: 此文件定义了 index 页面的布局、index 页面中用到的组件，以及这些组件的层级关系。例如：index.html 文件中包含了一个 text 组件，内容为“Hello World”文本。

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

- **main > js > default > pages > index > index.css:** 此文件定义了 index 页面的样式。例如：index.css 文件定义了“container”和“title”的样式。

```
1. .container {  
2.   flex-direction: column;  
3.   justify-content: center;  
4.   align-items: center;  
5. }  
6. .title {  
7.   font-size: 100px;  
8. }
```

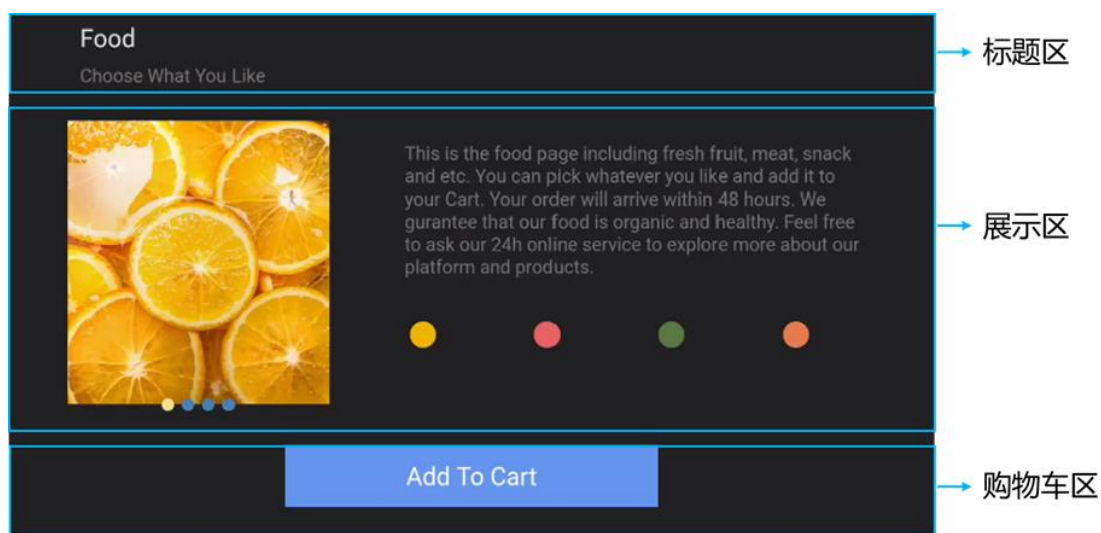
- **main > js > default > pages > index > index.js:** 此文件定义了 index 页面的业务逻辑，比如数据绑定、事件处理等。例如：变量“title”赋值为字符串“World”。

```
1. export default {  
2.   data: {  
3.     title: "",  
4.   },  
5.   onInit() {  
6.     this.title = this.$t('strings.world');  
7.   },  
8. }
```

3.2.2.2 开发一个 JS FA 应用

本章节主要介绍如何开发一个 JS FA 应用。此应用相对于 Hello World 应用模板具备更复杂的页面布局、页面样式和页面逻辑。该页面可以通过将焦点移动到不同颜色的圆形来选择不同的食物图片，也可以进行添加到购物车操作，应用效果图如下。

图 1 JS FA 应用效果图



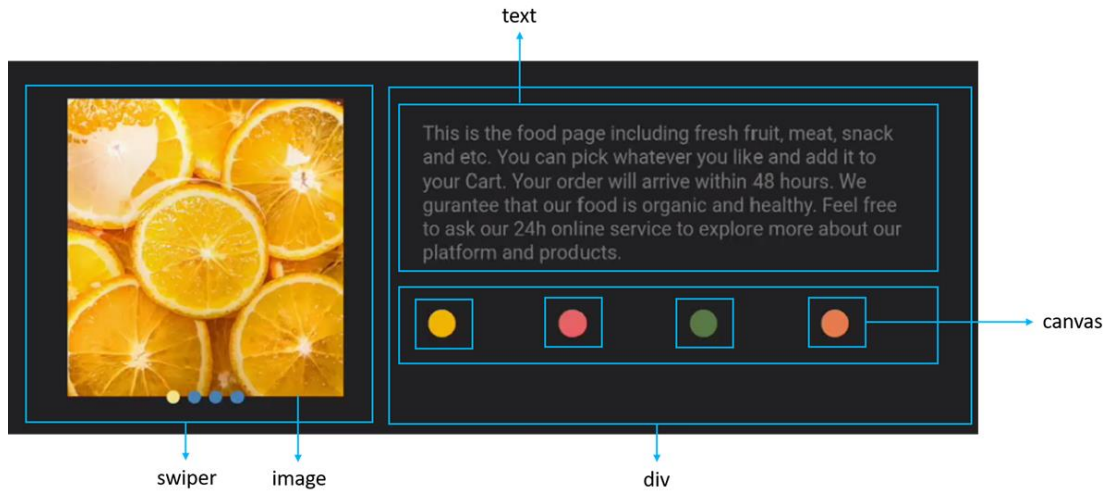
构建页面布局

开发者在 index.html 文件中构建页面布局。在进行代码开发之前，首先要对页面布局进行分析，将页面分解为不同的部分，用容器组件来承载。根据 JS FA 应用效果图，此页面一共分成三个部分：标题区、展示区和购物车区。根据此分区，可以确定根节点的子节点应按列排列。

标题区是由两个按列排列的 text 组件实现，购物车区由一个 text 组件构成。展示区由按行排列的 swiper 组件和 div 组件组成，如下图所示：

- 第一部分是由一个容器组件 swiper，包含了四个 image 组件构成；
- 第二部分是由一个容器组件 div，包含了一个 text 组件和四个画布组件 canvas 绘制的圆形构成。

图 2 展示区布局



根据布局结构的分析，实现页面基础布局的代码示例如下（其中四个 image 组件和 canvas 组件通过 for 指令来循环创建）：

```

1. <!-- index.html -->
2. <div class="container">
3.   <div class="title-section">
4.     <div class="title">
5.       <text class="name">Food</text>
6.       <text class="sub-title">Choose What You Like</text>
7.     </div>
8.   </div>
9.   <div>
10.    <swiper id="swiperImage" class="swiper-style">
11.      <image src="{{item}}" class="image-mode" focusable="true" for="{{imageList}}"></image>
12.    </swiper>
13.    <div class="container">
14.      <div class="description-first-paragraph">
15.        <text class="description">{{descriptionFirstParagraph}}</text>
16.      </div>
17.      <div class="color-column">
18.        <canvas id="{{item.id}}" onfocus="swipeToIndex({{item.index}})" class="color-item" focusable="true"
19.          for="{{canvasList}}"></canvas>
20.      </div>
21.    </div>
22.  </div>
23. <div class="cart">

```

```
24. <text class="{{cartStyle}}" onclick="addCart" onfocus="getFocus" onblur="lostFocus" focusable="true">
25.     {{cartText}}</text>
26. </div>
27. </div>
```

说明

common 目录用于存放公共资源，swiper 组件里展示的图片需要放在 common 目录下。

构建页面样式

开发者在 index.css 文件中需要设定的样式主要有 flex-direction（主轴方向），padding（内边距），font-size（字体大小）等。在构建页面样式中，还采用了 css 伪类的写法，当焦点移动到 canvas 组件上时，背景颜色变成白色，也可以在 js 中通过 focus 和 blur 事件动态修改 css 样式来实现同样的效果。

```
1. /* index.css */
2. .container {
3.     flex-direction: column;
4. }
5.
6. .title-section {
7.     flex-direction: row;
8.     height: 60px;
9.     margin-bottom: 5px;
10.    margin-top: 10px;
11. }
12.
13. .title {
14.    align-items: flex-start;
15.    flex-direction: column;
16.    padding-left: 60px;
17.    padding-right: 160px;
18. }
19.
20. .name {
21.    font-size: 20px;
22. }
```

```
23.
24. .sub-title {
25.     font-size: 15px;
26.     color: #7a787d;
27.     margin-top: 10px;
28. }
29.
30. .swiper-style {
31.     height: 250px;
32.     width: 350px;
33.     indicator-color: #4682b4;
34.     indicator-selected-color: #f0e68c;
35.     indicator-size: 10px;
36.     margin-left: 50px;
37. }
38.
39. .image-mode {
40.     object-fit: contain;
41. }
42.
43. .color-column {
44.     flex-direction: row;
45.     align-content: center;
46.     margin-top: 20px;
47. }
48.
49. .color-item {
50.     height: 50px;
51.     width: 50px;
52.     margin-left: 50px;
53.     padding-left: 10px;
54. }
55.
56. .color-item:focus {
57.     background-color: white;
58. }
59.
60. .description-first-paragraph {
```

```
61. padding-left: 60px;
62. padding-right: 60px;
63. padding-top: 30px;
64. }
65.
66. .description {
67.   color: #7a787d;
68.   font-size: 15px;
69. }
70.
71. .cart {
72.   justify-content: center;
73.   margin-top: 20px;
74. }
75.
76. .cart-text {
77.   font-size: 20px;
78.   text-align: center;
79.   width: 300px;
80.   height: 50px;
81.   background-color: #6495ed;
82.   color: white;
83. }
84.
85. .cart-text-focus {
86.   font-size: 20px;
87.   text-align: center;
88.   width: 300px;
89.   height: 50px;
90.   background-color: #4169e1;
91.   color: white;
92. }
93.
94. .add-cart-text {
95.   font-size: 20px;
96.   text-align: center;
97.   width: 300px;
98.   height: 50px;
```

```
99.   background-color: #ffd700;
100.  color: white;
101. }
```

构建页面逻辑

开发者在 `index.js` 文件中构建页面逻辑，主要实现的是两个逻辑功能：

- 当焦点移动到不同颜色的圆形，swiper 滑动到不同的图片；
- 当焦点移动到购物车区时，“Add To Cart”背景颜色从浅蓝变成深蓝，点击后文字变化为“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。

逻辑页面代码示例如下：

```
1.  // index.js
2.  export default {
3.    data: {
4.      cartText: 'Add To Cart',
5.      cartStyle: 'cart-text',
6.      isCartEmpty: true,
7.      descriptionFirstParagraph: 'This is the food page including fresh fruit, meat, snack and etc. You can pick whatever you like and add it to your Cart. Your order will arrive within 48 hours. We gurantee that our food is organic and healthy. Feel free to ask our 24h online service to explore more about our platform and products.',
8.      imageUrl: ['/common/food_000.JPG', '/common/food_001.JPG', '/common/food_002.JPG', '/common/food_003.JPG'],
9.      canvasList: [{
10.        id: 'cycle0',
11.        index: 0,
12.        color: '#f0b400',
13.      }, {
14.        id: 'cycle1',
15.        index: 1,
16.        color: '#e86063',
17.      }, {
18.        id: 'cycle2',
19.        index: 2,
20.        color: '#597a43',
21.      }, {
22.        id: 'cycle3',
```

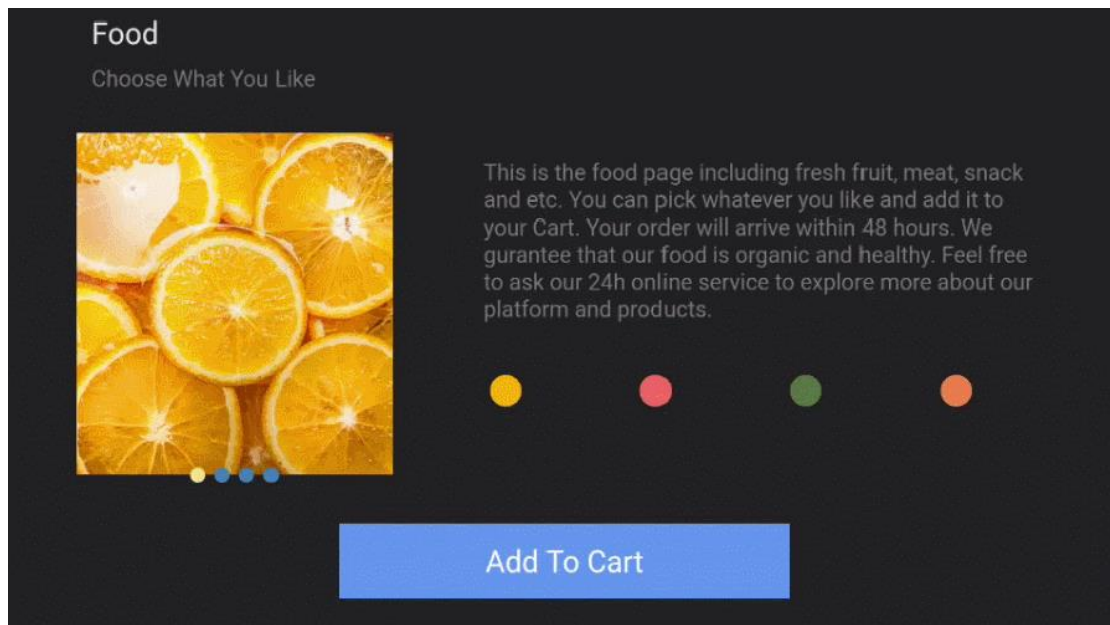
```
23.     index: 3,
24.     color: '#e97d4c',
25.   }],
26. },
27.
28. onShow() {
29.   this.canvasList.forEach(element => {
30.     this.drawCycle(element.id, element.color);
31.   });
32. },
33.
34. swipeToIndex(index) {
35.   this.$element('swiperImage').swipeTo({index: index});
36. },
37.
38. drawCycle(id, color) {
39.   var greenCycle = this.$element(id);
40.   var ctx = greenCycle.getContext("2d");
41.   ctx.strokeStyle = color;
42.   ctx.fillStyle = color;
43.   ctx.beginPath();
44.   ctx.arc(15, 25, 10, 0, 2 * 3.14);
45.   ctx.closePath();
46.   ctx.stroke();
47.   ctx.fill();
48. },
49.
50. addCart() {
51.   if (this.isCartEmpty) {
52.     this.cartText = 'Cart + 1';
53.     this.cartStyle = 'add-cart-text';
54.     this.isCartEmpty = false;
55.   }
56. },
57.
58. getFocus() {
59.   if (this.isCartEmpty) {
60.     this.cartStyle = 'cart-text-focus';
```

```
61.     }  
62.   },  
63.  
64.   lostFocus() {  
65.     if (this.isCartEmpty) {  
66.       this.cartStyle = 'cart-text';  
67.     }  
68.   },  
69. }
```

效果示例

实现此实例后，效果示例如下图所示。

图 3 运行效果



3.2.3 构建用户界面

3.2.3.1 组件介绍

组件 (Component) 是构建页面的核心，每个组件通过对数据和方法的简单封装，实现独立的可视、可交互功能单元。组件之间相互独立，随取随用，也可以在需求相同的地方重复使用。开发者还可以通过组件间合理的搭配定义满足业务需求的新组件，减少开发量，自定义组件的开发方法详见[自定义组件](#)。

组件分类

根据组件的功能，可以将组件分为以下四大类：

组件类型	主要组件
基础组件	text、image、progress、rating、span、marquee、image-animator、divider、search、menu、chart
容器组件	div、list、list-item、stack、swiper、tabs、tab-bar、tab-content、popup、list-item-group、refresh、dialog
媒体组件	video
画布组件	canvas

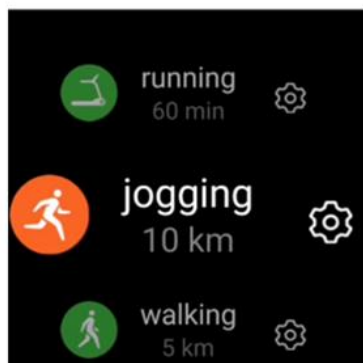
3.2.3.2 构建布局

布局说明

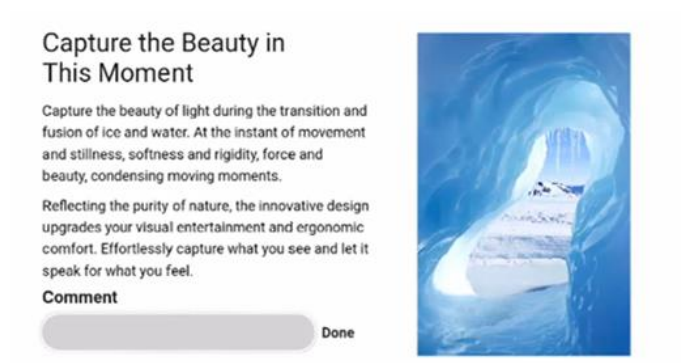
JS UI 框架以 720px (px 指逻辑像素，非物理像素) 为基准宽度，根据实际屏幕宽度进行缩放，例如当 width 设为 100px 时，在 1440px 宽度屏幕上，实际显示的宽度为 200px。JS UI 框架在不同设备的布局示例如下：

图 1 不同设备的布局示例

智能穿戴页面效果



智慧屏页面效果



绘制布局图

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

- 该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，方便对页面做后续的调整。以下图为例进行分解：

图 1 页面布局分解

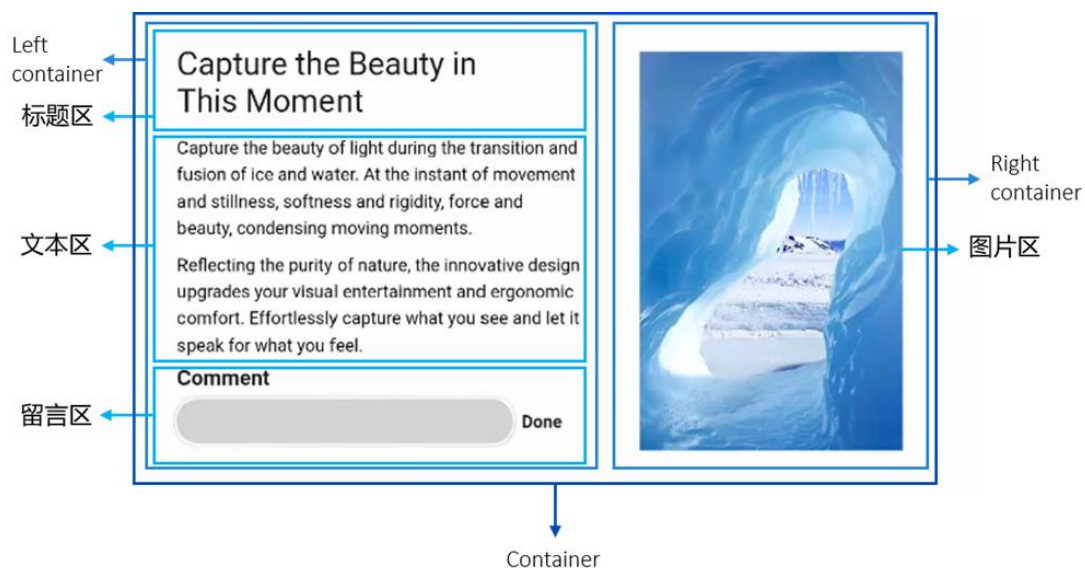
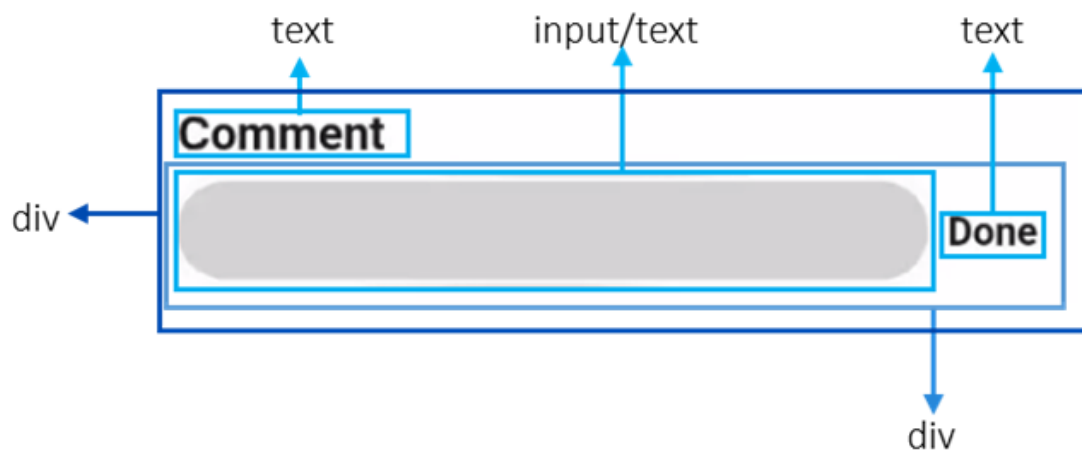


图 2 留言区布局分解



添加标题行和文本区域

实现标题和文本区域最常用的是基础组件 `text`。`text` 组件用于展示文本，可以设置不同的属性和样式，文本内容需要写在标签内容区，完整属性和样式信息请参考 `text`。在页面中插入标题和文本区域的示例如下：

```

1. <!-- xxx.html -->
2. <div class="container">
3.   <div class="left-container">
4.     <text class="title-text">{{headTitle}}</text>
5.     <text class="paragraph-text">{{paragraphFirst}}</text>
6.     <text class="paragraph-text">{{paragraphSecond}}</text>

```

```
7. </div>
```

```
8. </div>
```

```
1. /* xxx.css */
```

```
2. .container {
```

```
3.   margin-top: 24px;
```

```
4.   background-color: #ffffff;
```

```
5. }
```

```
6. .left-container {
```

```
7.   flex-direction: column;
```

```
8.   margin-left: 48px;
```

```
9.   width: 460px;
```

```
10. }
```

```
11. .title-text {
```

```
12.   color: #1a1a1a;
```

```
13.   font-size: 36px;
```

```
14.   height: 90px;
```

```
15.   width: 400px;
```

```
16. }
```

```
17. .paragraph-text {
```

```
18.   color: #000000;
```

```
19.   margin-top: 12px;
```

```
20.   font-size: 20px;
```

```
21.   line-height: 30px;
```

```
22. }
```

```
1. // xxx.js
```

```
2. export default {
```

```
3.   data: {
```

```
4.     headTitle: 'Capture the Beauty in This Moment',
```

```
5.     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.',
```

```
6.     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.',
```

```
7.   },
```

```
8. }
```

添加图片区域

实现图片区域通常用 image 组件来实现，使用的方法和 text 组件类似。图片资源放在

common 目录下，图片的路径要与图片实际所在的目录一致。具体示例如下：

```
1. <!-- xxx.html -->
2. <!-- 插入图片 -->
3. <div class="right-container">
4.   <image class="img" src="{{middleImage}}"></image>
5. </div>
```

```
1. /* xxx.css */
2. .right-container {
3.   width: 432px;
4.   justify-content: center;
5. }
6. .img {
7.   margin-top: 10px;
8.   object-fit: contain;
9.   height: 450px;
10. }
```

```
1. // xxx.js
2. export default {
3.   data: {
4.     middleImage: '/common/ice.png',
5.   },
6. }
```

添加留言区域

留言框的功能为：用户输入留言后点击完成，留言区域即显示留言内容；用户点击右侧的删除按钮可删除当前留言内容重新输入。

留言区域由 div、text、input 关联 click 事件实现。开发者可以使用 input 组件实现输入留言的部分，使用 text 组件实现留言完成部分，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。具体的实现示例如下：

```

1. <!-- xxx.html -->
2. <div class="container">
3.   <div class="left-container">
4.     <text class="comment-title">Comment</text>
5.     <div if="!{{commentText}}">
6.       <input class="comment" value="{{inputValue}}" onchange="updateValue()"></input>
7.       <text class="comment-key" onclick="update" focusable="true">Done</text>
8.     </div>
9.     <div if="{{commentText}}">
10.      <text class="comment-text" focusable="true">{{inputValue}}</text>
11.      <text class="comment-key" onclick="update" focusable="true">Delete</text>
12.    </div>
13.   </div>
14. </div>

```

```

1. /* xxx.css */
2. .container {
3.   margin-top: 24px;
4.   background-color: #ffffff;
5. }
6. .left-container {
7.   flex-direction: column;
8.   margin-left: 48px;
9.   width: 460px;
10. }
11. .comment-title {
12.   font-size: 24px;
13.   color: #1a1a1a;
14.   font-weight: bold;
15.   margin-top: 10px;
16.   margin-bottom: 10px;

```

```
17. }
18. .comment-key {
19.   width: 74px;
20.   height: 50px;
21.   margin-left: 10px;
22.   font-size: 20px;
23.   color: #1a1a1a;
24.   font-weight: bold;
25. }
26. .comment-key:focus {
27.   color: #007dff;
28. }
29. .comment-text {
30.   width: 386px;
31.   height: 50px;
32.   text-align: left;
33.   line-height: 35px;
34.   font-size: 20px;
35.   color: #000000;
36.   border-bottom-color: #bcbcbc;
37.   border-bottom-width: 0.5px;
38. }
39. .comment {
40.   width: 386px;
41.   height: 50px;
42.   background-color: lightgrey;
43. }
```

```
1. // xxx.js
2. export default {
3.   data: {
4.     inputValue: "",
5.     commentText: false,
6.   },
7.   update() {
8.     this.commentText = !this.commentText;
9.   },
10.  updateValue(e) {
```

```
11.   this.inputValue = e.text;
12.   },
13. }
```

添加外部容器

要将页面的基本元素组装在一起，需要使用容器组件。在页面布局中常用到三种容器组件，分别是 `div`、`list` 和 `tabs`。在页面结构相对简单时，可以直接用 `div` 作为容器，因为 `div` 作为单纯的布局容器，使用起来更为方便，可以支持多种子组件。

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。但是，`list` 组件仅支持 `list-item` 作为子组件，因此使用 `list` 时需要留意 `list-item` 的注意事项。具体的使用示例如下：

```
1.  <!-- xxx.html -->
2.  <list class="list">
3.    <list-item type="listItem" for="{{textList}}">
4.      <text class="desc-text">{{$item.value}}</text>
5.    </list-item>
6.  </list>
```

```
1.  /* xxx.css */
2.  desc-text {
3.    width: 683.3px;
4.    font-size: 35.4px;
5.    color: #000000;
6.  }
```

```
1.  // xxx.js
2.  export default {
3.    data: {
```

```
4.   textList: [{value: 'JS FA'}],
5.   },
6. }
```

为避免示例代码过长，以上示例的 list 中只包含一个 list-item，list-item 中只有一个 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。

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。具体的使用示例如下：

```
1. <!-- xxx.html -->
2. <tabs>
3.   <tab-bar class="tab-bar">
4.     <text style="color: #000000">tab-bar</text>
5.   </tab-bar>
6.   <tab-content>
7.     <image src="{{tabImage}}"></image>
8.   </tab-content>
9. </tabs>
```

```
1. /* xxx.css */
2. .tab-bar {
3.   background-color: #f2f2f2;
4.   width: 720px;
5. }
```

```
1. // xxx.js
2. export default {
3.   data: {
4.     tabImage: '/common/image.png',
```



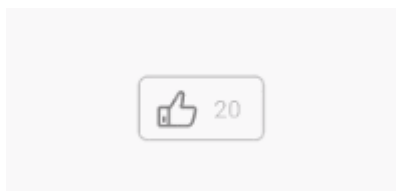
```
5.   },
6.   }
```

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

3.2.3.3 添加交互

添加交互通过在组件上关联事件实现。本节将介绍如何用 div、text、image 组件关联 click 事件，构建一个如下图所示的点赞按钮。

图 1 点赞按钮效果



点赞按钮通过一个 div 组件关联 click 事件实现。div 组件包含一个 image 组件和一个 text 组件：

- image 组件用于显示未点赞和点赞的效果。click 事件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，从而更新显示的 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。具体的实现示例如下：

```
1. <!-- xxx.hml -->
2. <!-- 点赞按钮 -->
3. <div>
4.   <div class="like" onclick="likeClick">
5.     <image class="like-img" src="{{likeImage}}" focusable="true"></image>
6.     <text class="like-num" focusable="true">{{total}}</text>
7.   </div>
8. </div>
```

```
1. /* xxx.css */
2. .like {
3.     width: 104px;
4.     height: 54px;
5.     border: 2px solid #bcbcbc;
6.     justify-content: space-between;
7.     align-items: center;
8.     margin-left: 72px;
9.     border-radius: 8px;
10. }
11. .like-img {
12.     width: 33px;
13.     height: 33px;
14.     margin-left: 14px;
15. }
16. .like-num {
17.     color: #bcbcbc;
18.     font-size: 20px;
19.     margin-right: 17px;
20. }
```

```
1. // xxx.js
2. export default {
3.     data: {
4.         likeImage: '/common/unLike.png',
5.         isPressed: false,
6.         total: 20,
7.     },
8.     likeClick() {
9.         var temp;
10.         if (!this.isPressed) {
11.             temp = this.total + 1;
12.             this.likeImage = '/common/like.png';
13.         } else {
14.             temp = this.total - 1;
15.             this.likeImage = '/common/unLike.png';
16.         }
17.         this.total = temp;
18.     }
19. }
```

```
18.   this.isPressed = !this.isPressed;
19.   },
20. }
```

JS UI 框架还提供了很多表单组件，例如开关、标签、滑动选择器等，以便于开发者在页面布局时灵活使用和提高交互性，详见容器组件。

3.2.3.4 动画

动画分为静态动画和连续动画。

静态动画

静态动画的核心是 `transform` 样式，主要可以实现以下三种变换类型，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**：将指定组件沿横轴或纵轴或中心点旋转指定的角度。

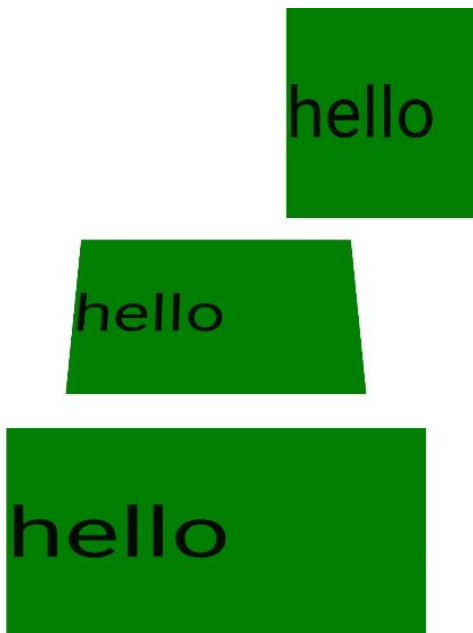
静态动画只有开始状态和结束状态，没有中间状态，如果需要设置中间的过渡状态和转换效果，需要使用连续动画实现。具体的使用示例如下，更多信息请参考组件。

```
1. <!-- xxx.html -->
2. <div class="container">
3.   <text class="translate">hello</text>
4.   <text class="rotate">hello</text>
5.   <text class="scale">hello</text>
6. </div>
```

```
1. /* xxx.css */
2. .container {
3.   flex-direction: column;
4.   align-items: center;
5. }
6. .translate {
```

```
7. height: 300px;
8. width: 400px;
9. font-size: 100px;
10. background-color: #008000;
11. transform: translate(300px);
12. }
13. .rotate {
14. height: 300px;
15. width: 400px;
16. font-size: 100px;
17. background-color: #008000;
18. transform-origin: 200px 100px;
19. transform: rotateX(45deg);
20. }
21. .scale {
22. height: 300px;
23. width: 400px;
24. font-size: 100px;
25. background-color: #008000;
26. transform: scaleX(1.5);
27. }
```

图 1 静态动画效果图



连续动画

连续动画的核心是 `animation` 样式，它定义了动画的开始状态、结束状态以及时间和速度的变化曲线。通过 `animation` 样式可以实现的效果有：

- **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`，在 `keyframe` 中设置动画的过渡效果，并通过一个样式类型在 `html` 文件中调用。`animation-name` 的使用示例如下：

```
1. <!-- xxx.html -->
2. <div class="item-container">
3.   <div class="group">
4.     <text class="header">animation-name</text>
5.     <div class="item {{colorParam}}">
6.       <text class="txt">color</text>
7.     </div>
8.     <div class="item {{opacityParam}}">
9.       <text class="txt">opacity</text>
10.    </div>
11.    <input class="button" type="button" name="" value="show" onclick="showAnimation"/>
12.  </div>
13. </div>
```

```
1. /* xxx.css */
2. .item-container {
3.   margin-bottom: 50px;
4.   margin-right: 60px;
5.   margin-left: 60px;
6.   flex-direction: column;
7.   align-items: flex-start;
8. }
9. .group {
```

```
10. margin-bottom: 150px;
11. flex-direction: column;
12. align-items: flex-start;
13. }
14. .header {
15.   margin-bottom: 20px;
16. }
17. .item {
18.   background-color: #f76160;
19. }
20. .txt {
21.   text-align: center;
22.   width: 200px;
23.   height: 100px;
24. }
25. .button {
26.   width: 200px;
27.   font-size: 30px;
28.   color: #ffffff;
29.   background-color: #09ba07;
30. }
31. .color {
32.   animation-name: Color;
33.   animation-duration: 8000ms;
34. }
35. .opacity {
36.   animation-name: Opacity;
37.   animation-duration: 8000ms;
38. }
39. @keyframes Color {
40.   from {
41.     background-color: #f76160;
42.   }
43.   to {
44.     background-color: #09ba07;
45.   }
46. }
47. @keyframes Opacity {
```

```
48. from {  
49.   opacity: 0.9;  
50. }  
51. to {  
52.   opacity: 0.1;  
53. }  
54. }
```

```
1. // xxx.js  
2. export default {  
3.   data: {  
4.     colorParam: "",  
5.     opacityParam: "",  
6.   },  
7.   showAnimation: function () {  
8.     this.colorParam = "";  
9.     this.opacityParam = "";  
10.    this.colorParam = 'color';  
11.    this.opacityParam = 'opacity';  
12.  },  
13. }
```

图 2 连续动画效果图



3.2.3.5 事件

事件主要包括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。

手势事件

手势表示由单个或多个事件识别的语义动作（例如：点击、拖动和长按）。一个完整的手势也可能由多个事件组成，对应手势的生命周期。JS UI 框架支持的手势事件有：

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，如来电提醒、弹窗。
- touchend: 手指触摸动作结束。

点击

click: 用户快速轻敲屏幕。

长按

longpress: 用户在相同位置长时间保持与屏幕接触。

具体的使用示例如下：

```
1. <!-- xxx.html -->
2. <div class="container">
3.   <div class="text-container" onclick="click">
4.     <text class="text-style">{{onclick}}</text>
5.   </div>
6.   <div class="text-container" ontouchstart="touchStart">
7.     <text class="text-style">{{touchStart}}</text>
```



```

8.   </div>
9.   <div class="text-container" ontouchmove="touchMove">
10.    <text class="text-style">{{touchMove}}</text>
11.  </div>
12.  <div class="text-container" ontouchend="touchEnd">
13.    <text class="text-style">{{touchEnd}}</text>
14.  </div>
15.  <div class="text-container" ontouchcancel="touchCancel">
16.    <text class="text-style">{{touchCancel}}</text>
17.  </div>
18.  <div class="text-container" onlongpress="longPress">
19.    <text class="text-style">{{onLongPress}}</text>
20.  </div>
21. </div>

```

```

1.  /* xxx.css */
2.  .container {
3.    flex-direction: column;
4.    justify-content: center;
5.    align-items: center;
6.  }
7.  .text-container {
8.    padding-top: 10px;
9.    flex-direction: column;
10.  }
11. .text-style {
12.   padding-top: 20px;
13.   padding-left: 100px;
14.   width: 750px;
15.   height: 100px;
16.   text-align: center;
17.   font-size: 50px;
18.   color: #ffffff;
19.   background-color: #09ba07;
20. }

```

```

1.  // xxx.js
2.  export default {
3.    data: {

```

```
4.     textData: "",
5.     touchStart: 'touchstart',
6.     touchMove: 'touchmove',
7.     touchEnd: 'touchend',
8.     touchCancel: 'touchcancel',
9.     onClick: 'onclick',
10.    onLongPress: 'onlongpress',
11.  },
12.  onInit() {
13.    this.textData = 'initdata';
14.  },
15.  onReady: function () {},
16.  onShow: function () {},
17.  onHide: function () {},
18.  onDestroy: function () {},
19.  touchCancel: function (event) {
20.    this.touchCancel = 'canceled';
21.  },
22.  touchEnd: function(event) {
23.    this.touchEnd = 'ended';
24.  },
25.  touchMove: function(event) {
26.    this.touchMove = 'moved';
27.  },
28.  touchStart: function(event) {
29.    this.touchStart = 'touched';
30.  },
31.  longPress: function() {
32.    this.onLongPress = 'longpressed';
33.  },
34.  click: function() {
35.    this.onClick = 'clicked';
36.  },
37. }
```

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。用户点击一个遥控器按键，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) 以实现不同的功能，常用的按键值请参考[组件通用事件](#)。具体的使用示例如下：

```
1. <!-- xxx.html -->
2. <div class="card-box">
3.   <div class="content-box">
4.     <text class="content-text" onkey="keyUp" onFocus="focusUp" onBlur="blurUp">{{up}}</text>
5.   </div>
6.   <div class="content-box">
7.     <text class="content-text" onkey="keyDown" onFocus="focusDown" onBlur="blurDown">{{down}}</text>
8.   </div>
9. </div>
```

```
1. /* xxx.css */
2. .card-box {
3.   flex-direction: column;
4.   justify-content: center;
5. }
6. .content-box {
7.   align-items: center;
8.   height: 200px;
9.   flex-direction: column;
10.  margin-left: 200px;
11.  margin-right: 200px;
12. }
13. .content-text {
14.  font-size: 40px;
15.  text-align: center;
16. }
```

```
1. // xxx.js
2. export default {
3.   data: {
4.     up: 'up',
5.     down: 'down',
6.   },
7.   focusUp: function() {
8.     this.up = 'up focused';
9.   },
10.  blurUp: function() {
11.    this.up = 'up';
12.  },
13.  keyUp: function() {
14.    this.up = 'up keyed';
15.  },
16.  focusDown: function() {
17.    this.down = 'down focused';
18.  },
19.  blurDown: function() {
20.    this.down = 'down';
21.  },
22.  keyDown: function() {
23.    this.down = 'down keyed';
24.  },
25. }
```

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。点按上下键选中 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。

3.2.3.6 页面路由

很多应用由多个页面组成，比如用户可以从音乐列表页面点击歌曲，跳转到该歌曲的播放界面。开发者需要通过页面路由将这些页面串联起来，按需实现跳转。

页面路由 router 根据页面的 uri 来找到目标页面，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：

1. 创建两个页面。
2. 修改配置文件 config.json。
3. 调用 router.push()路由到详情页。
4. 调用 router.back()回到首页。

创建两个页面

创建 index 和 detail 页面，这两个页面均包含一个 text 组件和 button 组件：text 组件用来指

明当前页面，button 组件用来实现两个页面之间的相互跳转。hml 文件代码示例如下：

```
1. <!-- index.hml -->
2. <div class="container">
3.   <div class="text-div">
4.     <text class="title">This is the index page.</text>
5.   </div>
6.   <div class="button-div">
7.     <button type="capsule" value="Go to the second page" onclick="launch"></button>
8.   </div>
9. </div>
```

```
1. <!-- detail.hml -->
2. <div class="container">
3.   <div class="text-div">
4.     <text class="title">This is the detail page.</text>
5.   </div>
6.   <div class="button-div">
7.     <button type="capsule" value="Go back" onclick="launch"></button>
8.   </div>
9. </div>
```

修改配置文件

config.json 文件是配置文件，主要包含了 JS FA 页面路由信息。开发者新创建的页面都要在配置文件的 pages 标签中进行注册，处于第一位的页面为首页，即点击图标后的主页面。

```
1. {
2. ...
3.   "pages": [
4.     "pages/index/index",
5.     "pages/detail/detail"
6.   ],
7. ...
8. }
```

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。代码示例如下：

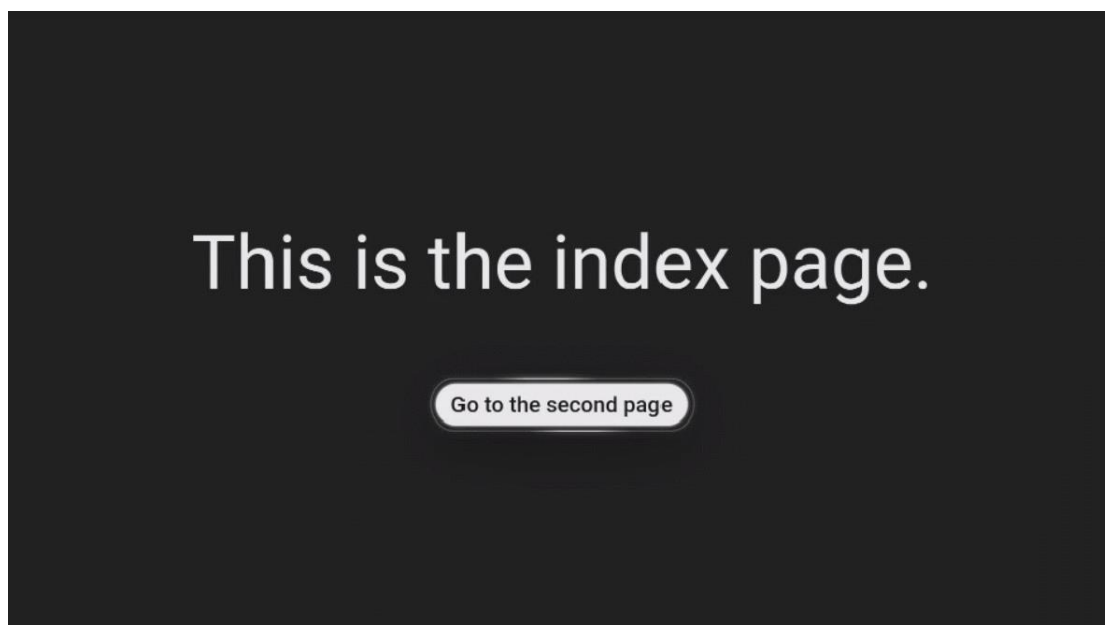
```
1. // index.js
2. import router from '@system.router';
3. export default {
4.   launch: function() {
5.     router.push ({
6.       uri: 'pages/detail/detail',
7.     });
8.   },
9. }
```

```
1. // detail.js
2. import router from '@system.router';
3. export default {
4.   launch: function() {
5.     router.back();
6.   },
7. }
```

7. }

运行效果如下图所示：

图 1 页面路由效果



3.2.3.7 焦点逻辑

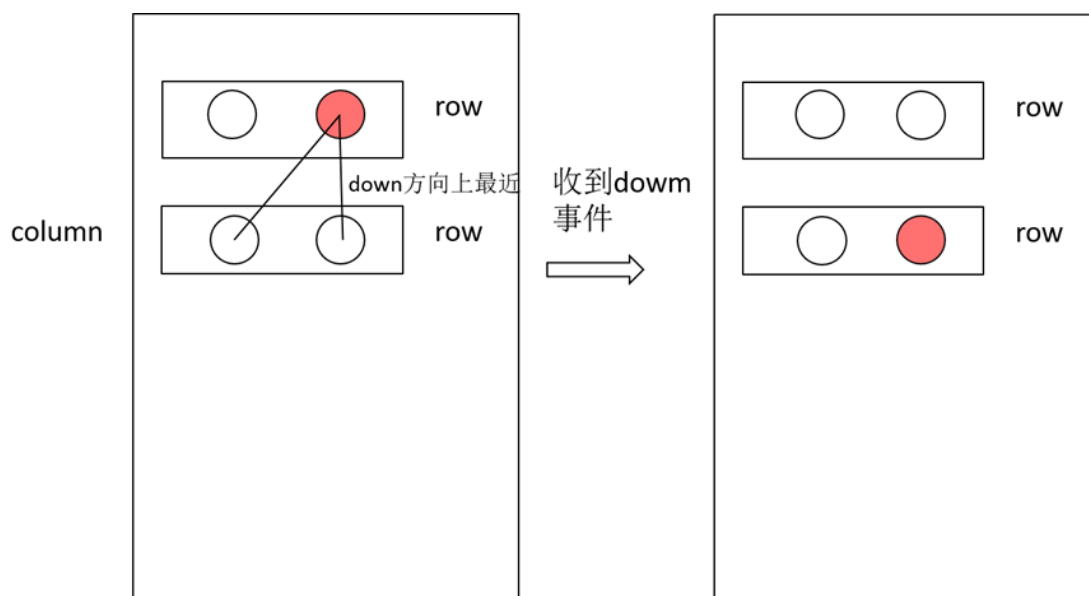
焦点移动是智慧屏的主要交互方式，本节将介绍焦点逻辑的主要规则。

- 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都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，以下分别说明常用容器组件的焦点分发逻辑。

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 1 中焦点在上方的横向 div 的第二个子组件上，当点击 down 按键时，焦点要移动到下方的横向 div 中。这时下方的横向 div 中的子组件会与当前焦点所在的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 1 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 的页面的焦点逻辑都与 div 的相同。
5. tabs 组件包含 tab-bar 与 tab-content，tab-bar 中的子组件默认都能获焦，与是否有可获焦的叶子结点无关。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。

- **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。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。特别的是，如果在没有使用 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。

容器组件是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。当容器组件 focusable 属性设置为 false，则它本身和它所包含的所有组件都不可获焦。

3.2.4 自定义组件

JS UI 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。具体的封装方法示例如下：

构建自定义组件

```
1. <!-- comp.html -->
2. <div class="item">
3.   <text class="title-style">{{title}}</text>
4.   <text class="text-style" onclick="childClicked" focusable="true">点击查看隐藏文本</text>
5.   <text class="text-style" if="{{show}}">hello world</text>
6. </div>
7. /* comp.css */
8. .item {
9.   width: 700px;
10.  flex-direction: column;
11.  height: 300px;
12.  align-items: center;
13.  margin-top: 100px;
14. }
15. .text-style {
16.  width: 100%;
17.  text-align: center;
18.  font-weight: 500;
19.  font-family: Courier;
20.  font-size: 36px;
21. }
22. .title-style {
23.  font-weight: 500;
24.  font-family: Courier;
25.  font-size: 50px;
26.  color: #483d8b;
27. }
```

```

28. // comp.js
29. export default {
30.   props: {
31.     title: {
32.       default: 'title',
33.     },
34.     showObject: {},
35.   },
36.   data() {
37.     return {
38.       show: this.showObject,
39.     };
40.   },
41.   childClicked () {
42.     this.$emit('eventType1', {text: '收到子组件参数'});
43.     this.show = !this.show;
44.   },
45. }

```

- 引入自定义组件

```

1. <!-- xxx.html -->
2. <element name='comp' src='../common/component/comp.html'></element>
3. <div class="container">
4.   <text>父组件: {{text}}</text>
5.   <comp title="自定义组件" show-object="{{show}}" @event-type1="textClicked"></comp>
6. </div>
7. /* xxx.css */
8. .container {
9.   background-color: #f8f8ff;
10.  flex: 1;
11.  flex-direction: column;
12.  align-content: center;
13. }
14. // xxx.js
15. export default {
16.  data: {
17.    text: '开始',
18.    show: false,
19.  },

```

```

20.   textClicked (e) {
21.     this.text = e.detail.text;
22.   },
23. }
    
```

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，要绑定子组件事件，父组件事件命名必须遵循事件绑定规则，详见 [自定义组件开发规范](#)。自定义组件效果如下图所示：

图 1 自定义组件静态效果



图 2 自定义组件动态效果



3.2.5 JS FA 如何调用 PA

JS UI 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来传递方法调用、数据返回以及订阅事件上报。

当前提供 Ability 和 Internal Ability 两种调用方式，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。

JS 端与 Java 端通过 `bundleName` 和 `abilityName` 来进行关联。在系统收到 JS 调用请求后，根据开发者在 JS 接口中设置的参数来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。详细信息请参考 JS FA 调用 Java PA 机制。

FA 调用 PA 接口

FA 端提供以下三个 JS 接口：

- `FeatureAbility.callAbility(OBJECT)`：调用 PA 能力。
- `FeatureAbility.subscribeAbilityEvent(OBJECT, Function)`：订阅 PA 能力。
- `FeatureAbility.unsubscribeAbilityEvent(OBJECT)`：取消订阅 PA 能力。

PA 端提供以下两类接口：

- `boolean IRemoteObject.onRemoteRequest(int code, MessageParcel data, MessageParcel reply, MessageOption option)`：Ability 调用方式，FA 使用远端进程通信拉起并请求 PA 服务。
- `boolean AceInternalAbility.AceInternalAbilityHandler.onRemoteRequest(int code, MessageParcel data, MessageParcel reply, MessageOption option)`：Internal Ability 调用方式，采用内部函数调用的方式和 FA 进行通信。

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，具体注册可参考 Internal Ability 的示例代码。
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 Ability 误填写为 Internal Ability 方式。

Ability 和 Internal Ability 是两种不同的 FA 调用 PA 的方式。[表 1](#)列举了在开发时各方面的差异，供开发者参考，避免开发时将两者混淆使用：

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

表 1 Ability 和 InternalAbility 差异项

- FeatureAbility.callAbility 中 syncOption 参数说明：
 - 对于 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，根据该参数的取值来选择：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，具体请参见接口通用规则。

示例参考

- FA JavaScript 端

```

1. // abilityType: 0-Ability; 1-Internal Ability
2. const ABILITY_TYPE_EXTERNAL = 0;
3. const ABILITY_TYPE_INTERNAL = 1;
4. // syncOption(Optional, default sync): 0-Sync; 1-Async
5. const ACTION_SYNC = 0;
6. const ACTION_ASYNC = 1;
7. const ACTION_MESSAGE_CODE_PLUS = 1001;
8. export default {
9.   plus: async function() {
10.     var actionData = {};

```

```

11.    actionData.firstNum = 1024;
12.    actionData.secondNum = 2048;
13.
14.    var action = {};
15.    action.bundleName = 'com.huawei.hiacservice';
16.    action.abilityName = 'CalcServiceAbility';
17.    action.messageCode = ACTION_MESSAGE_CODE_PLUS;
18.    action.data = actionData;
19.    action.abilityType = ABILITY_TYPE_EXTERNAL;
20.    action.syncOption = ACTION_SYNC;
21.
22.    var result = await FeatureAbility.callAbility(action);
23.    var ret = JSON.parse(result);
24.    if (ret.code && ret.code == 0) {
25.        console.info('plus result is:' + JSON.stringify(ret.abilityResult));
26.    } else {
27.        if (ret.code) {
28.            console.error('plus error code:' + JSON.stringify(ret.code));
29.        } else {
30.            console.error('plus error undefined.');
```

- PA 端 (Ability 方式)

功能代码实现：

CalcServiceAbility.java

```

1.    // ohos 相关接口包
2.    import ohos.aafwk.ability.Ability;
3.    import ohos.aafwk.content.Intent;
4.    import ohos.rpc.IRemoteBroker;
5.    import ohos.rpc.IRemoteObject;
6.    import ohos.rpc.RemoteObject;
7.    import ohos.rpc.MessageParcel;
8.    import ohos.rpc.MessageOption;
9.    import ohos.utils.zson.ZSONObject;
10.
```

```
11. import java.util.HashMap;
12. import java.util.Map;
13.
14. public class CalcServiceAbility extends Ability {
15.     private static final String TAG = "CalcServiceAbility";
16.     private MyRemote remote = new MyRemote();
17.     // FA 在请求 PA 服务时会调用 AbilityconnectAbility 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
18.     @Override
19.     protected IRemoteObject onConnect(Intent intent) {
20.         super.onConnect(intent);
21.         return remote.asObject();
22.     }
23.     class MyRemote extends RemoteObject implements IRemoteBroker {
24.         private static final int ERROR = -1;
25.         private static final int SUCCESS = 0;
26.         private static final int PLUS = 1001;
27.
28.         MyRemote() {
29.             super("MyService_MyRemote");
30.         }
31.
32.         @Override
33.         public boolean onRemoteRequest(int code, MessageParcel data, MessageParcel reply, MessageOption option) {
34.             switch (code) {
35.                 case PLUS: {
36.                     String zsonStr = data.readString();
37.                     RequestParam param = ZSONObject.stringToClass(zsonStr, RequestParam.class);
38.
39.                     // 返回结果仅支持可序列化的 Object 类型
40.                     Map<String, Object> zsonResult = new HashMap<String, Object>();
41.                     zsonResult.put("code", SUCCESS);
42.                     zsonResult.put("abilityResult", param.getFirstNum() + param.getSecondNum());
43.                     reply.writeString(ZSONObject.toZSONString(zsonResult));
44.                     break;
45.                 }
46.                 default: {
47.                     Map<String, Object> zsonResult = new HashMap<String, Object>();
```

```

48.         zsonResult.put("abilityError", ERROR);
49.         reply.writeString(ZSONObject.toZSONString(zsonResult));
50.         return false;
51.     }
52. }
53.     return true;
54. }
55.
56.     @Override
57.     public IRemoteObject asObject() {
58.         return this;
59.     }
60. }
61. }

```

请求参数代码：

RequestParam.java

```

1. public class RequestParam {
2.     private int firstNum;
3.     private int secondNum;
4.
5.     public int getFirstNum() {
6.         return firstNum;
7.     }
8.
9.     public void setFirstNum(int firstNum) {
10.        this.firstNum = firstNum;
11.    }
12.
13.    public int getSecondNum() {
14.        return secondNum;
15.    }
16.
17.    public void setSecondNum(int secondNum) {
18.        this.secondNum = secondNum;
19.    }
20. }

```

- PA 端 (Internal Ability 方式)

功能代码实现：

CalcInternalAbility.java

```
1. // ohos 相关接口包
2. import ohos.ace.ability.AceInternalAbility;
3. import ohos.app.AbilityContext;
4. import ohos.rpc.IRemoteObject;
5. import ohos.rpc.MessageOption;
6. import ohos.rpc.MessageParcel;
7. import ohos.rpc.RemoteException;
8. import ohos.utils.zson.ZSONObject;
9.
10. import java.util.HashMap;
11. import java.util.Map;
12.
13. public class CalcInternalAbility extends AceInternalAbility {
14.     private static final String TAG = CalcInternalAbility.class.getSimpleName();
15.     private static final String BUNDLE_NAME = "com.huawei.hiacservice";
16.     private static final String ABILITY_NAME = "CalcInternalAbility";
17.     private static final int ERROR = -1;
18.     private static final int SUCCESS = 0;
19.     private static final int PLUS = 1001;
20.
21.     private static CalcInternalAbility instance;
22.     private AbilityContext abilityContext;
23.
24.     // 如果多个 Ability 实例都需要注册当前 InternalAbility 实例，需要更改构造函数，设定自己的 bundleName 和 abilityName
25.     public CalcInternalAbility() {
26.         super(BUNDLE_NAME, ABILITY_NAME);
27.     }
28.
29.     public boolean onRemoteRequest(int code, MessageParcel data, MessageParcel reply, MessageOption option) {
30.         switch (code) {
31.             case PLUS: {
32.                 String zsonStr = data.readString();
33.                 RequestParam param = ZSONObject.stringToClass(zsonStr, RequestParam.class);
34.                 // 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
35.                 Map<String, Object> zsonResult = new HashMap<String, Object>();
36.                 zsonResult.put("code", SUCCESS);
```

```

37.         zsonResult.put("abilityResult", param.getFirstNum() + param.getSecondNum());
38.         // SYNC
39.         if (option.getFlags() == MessageOption.TF_SYNC) {
40.             reply.writeString(ZSONObject.toZSONString(zsonResult));
41.         } else {
42.             // ASYNC
43.             MessageParcel reponseData = MessageParcel.obtain();
44.             reponseData.writeString(ZSONObject.toZSONString(zsonResult));
45.             IRemoteObject remoteReply = reply.readRemoteObject();
46.             try {
47.                 remoteReply.sendRequest(0, reponseData, MessageParcel.obtain(), new MessageOption());
48.                 reponseData.reclaim();
49.             } catch (RemoteException exception) {
50.                 return false;
51.             }
52.         }
53.         break;
54.     }
55.     default: {
56.         Map<String, Object> zsonResult = new HashMap<String, Object>();
57.         zsonResult.put("abilityError", ERROR);
58.         reply.writeString(ZSONObject.toZSONString(zsonResult));
59.         return false;
60.     }
61. }
62. return true;
63. }
64.
65. /**
66.  * Internal ability registration.
67.  */
68. public static void register(AbilityContext abilityContext) {
69.     instance = new CalcInternalAbility();
70.     instance.onRegister(abilityContext);
71. }
72.
73. private void onRegister(AbilityContext abilityContext) {
74.     this.abilityContext = abilityContext;

```

```
75.     this.setInternalAbilityHandler((code, data, reply, option) -> {
76.         return this.onRemoteRequest(code, data, reply, option);
77.     });
78. }
79.
80. /**
81.  * Internal ability deregistration.
82.  */
83. public static void deregister() {
84.     instance.onDeregister();
85. }
86.
87. private void onDeregister() {
88.     abilityContext = null;
89.     this.setInternalAbilityHandler(null);
90. }
91. }
```

Internal Ability 注册：修改继承 AceAbility 工程中的代码

```
1. public class HiAceInternalAbility extends AceAbility {
2.
3.     @Override
4.     public void onStart(Intent intent) {
5.         super.onStart(intent);
6.         // 注册
7.         CalcInternalAbility.register(this);
8.         ...
9.     }
10.    @Override
11.    public void onStop() {
12.        // 去注册
13.        CalcInternalAbility.deregister();
14.        super.onStop();
15.    }
16. }
```

3.3 多模输入

3.3.1.1 概述

HarmonyOS 旨在为开发者提供 NUI (Natural User Interface) 的交互方式，有别于传统操作系统的输入划分方式，在 HarmonyOS 上，我们将多种维度的输入整合在一起，开发者可以借助应用程序框架、系统自带的 UI 控件或 API 接口轻松地实现具有多维、自然交互特点的应用程序。

具体来说，HarmonyOS 目前不仅支持传统的输入交互方式，例如按键、触控、键盘、鼠标等，同时也支持语音等新型的输入交互方式。

约束与限制

- 多模输入事件在不同形态产品支持的情况如下表。

多模输入事件	智慧屏	车机	智能穿戴
按键输入事件	支持	支持	支持
触屏输入事件	支持	支持	支持
鼠标事件	部分支持	不支持	不支持
语音事件	支持	不支持	不支持

表 1 多模输入事件在不同形态产品支持的情况

- 说明**
- 智慧屏产品对鼠标事件只支持鼠标左键事件，鼠标右键以及滚轮等事件暂不支持。
- 目前多模输入不支持生成事件（即开发者无法创建事件）和注入事件（即开发者无法模拟注入事件验证应用程序功能）。
- 使用多模输入相关功能需要获取多模输入权限：`ohos.permission.MULTIMODAL_INTERACTIVE`。

3.3.1.2 开发指导

场景介绍

多模输入使 HarmonyOS 的 UI 控件能够响应多种输入事件，事件来源于用户的按键、点击、触屏、语音等。例如用户希望通过语音操作 UI 控件，那么开发者可以通过多模输入在智慧屏产品上提供的语音事件达到“[可见即可说](#)”的效果。

接口说明

多模输入的接口设计是基于多模事件基类（MultimodalEvent），派生出操作事件类（ManipulationEvent）、按键事件类（KeyEvent）、语音事件（SpeechEvent）等，详细见

[图 1](#)。

图 1 多模输入事件类的派生关系

- MultimodalEvent 是所有事件的基类，该类中定义了一系列高级事件类型，这些事件类型通常是对某种行为或意图的抽象。

功能分类	接口名	描述
设备信息相关	getDeviceId()	获取输入设备所在的承载设备 id，如当同时有两个鼠标连接到一个机器上，该机器为这两个鼠标的承载设备。
	getInputDeviceId()	获取产生当前事件的输入设备 id，该 id 是该输入设备的唯一标识，如两个鼠标同时输入时，它们会分别产生输入事件，且从事件中获取到的 deviceId 是不同的，开发者可以将此 id 用来区分实际的输入设备源。
	getSourceDevice()	获取产生当前事件的输入设备类型。
时间	getOccurredTime()	获取产生当前事件的时间。

功能分类	接口名	描述
事件	getUuid()	获取事件的 UUID。
	isSameEvent(UUID id)	判断当前事件与传入 id 的事件是否为同一事件。

表 1 MultimodalEvent 的主要接口

- CompositeEvent 处理常用设备对应的事件，目前暂时只有 MouseEvent 事件继承该类。
- RotationEvent 处理由旋转器件产生的事件，比如智能穿戴上的数字表冠。

功能分类	接口名	描述
旋转器信息	getRotationValue()	获取旋转器件旋转产生的值。

表 2 RotationEvent 的主要接口

- SpeechEvent 处理语音事件，开发者可以通过该类获取语音识别结果。

功能分类	接口名	描述
构造函数	public static Optional<SpeechEvent> createEvent(long occurTime, int action, String value)	SpeechEvent 构造函数。
获取语音事件参数值	public int getAction()	获取当前动作的类型，如打开、关闭、命中热词。
	public int getScene()	获取当前动作时的场景。
	public String getActionProperty()	获取动作所携带的属性值。
	public int getMatchMode()	获取识别结果的匹配模式。

表 3 SpeechEvent 的主要接口

- ManipulationEvent 操作类事件主要包括手指触摸事件等事件，是对这些事件的一个抽象。该事件会持有事件发生的位置信息和发生的阶段等信息。通常情况下，该事件主要是作为操作回调接口的入参，开发者通过回调接口捕获及处

理事件。回调接口将操作分为开始、操作过程中、结束。例如对于一次手指触控，手指接触屏幕作为操作开始，手指在屏幕上移动作为操作过程，手指抬起作为操作结束。

功能分类	接口名	描述
手指信息	getPointerCount()	获取一次事件中触控或轨迹追踪的指针数量。
	getPointerId(int index)	获取一次事件中，指针的唯一标识 Id。
	setScreenOffset(float offsetX, float offsetY)	设置相对屏幕坐标原点的偏移位置信息。
	getPointerPosition(int index)	获取一次事件中触控或轨迹追踪的某个指针相对于偏移位置的坐标信息。
	getPointerScreenPosition(int index)	获取一次事件中触控或轨迹追踪的某个指针相对屏幕坐标原点的坐标信息。
	getRadius(int index)	返回给定 index 手指与屏幕接触的半径值。
	getForce(int index)	获取给定 index 手指触控的压力值。
时间	getStartTime()	获取操作开始阶段时间。
阶段	getPhase()	事件所属阶段。

表 4 ManipulationEvent 的主要接口

- KeyEvent 对所有按键类事件的定义，该类继承 MultimodalEvent 类，并对按键类事件做了专属的 Keycode 定义以及方法封装。

功能分类	接口名	描述
KeyCode	getKeyCode()	获取当前按键类事件的 keycode 值。
	getMaxKeyCode()	获取当前定义的按键类事件的最大 keycode 值。
按键按下状态	getKeyDownDuration()	获取当前按键截止该接口被调用时被按下的时长。

功能分类	接口名	描述
	isKeyDown()	获取当前按键事件的按下状态。

表 5 KeyEvent 的主要接口

- TouchEvent 处理手指触控相关事件。

功能分类	接口名	描述
触控行为	getAction()	获取当前触摸行为。
	getIndex()	获取发生行为的对应指针。

表 6 TouchEvent 的主要接口

- KeyboardEvent 处理键盘类设备的事件。

功能分类	接口名	描述
输入法编辑器	enableIme()	启动输入法编辑器。
	disableIme()	关闭输入法编辑器。
	isHandledByIme()	判断输入法编辑器是否在使用。
NoncharacterKey 行为	isNoncharacterKeyPressed(int keycode)	判定输入的单个 NoncharacterKey 是否处于按下状态。
	isNoncharacterKeyPressed(int keycode1, int keycode2)	判定输入的两个 NoncharacterKey 是否都处于按下状态。
	isNoncharacterKeyPressed(int keycode1, int keycode2, int keycode3)	判定输入的三个 NoncharacterKey 是否都处于按下状态。
按键 Unicode 码	getUnicode()	获取按键对应的 Unicode 码。

表 7 KeyboardEvent 的主要接口

- 说明

- NoncharacterKey 为除了文本可见字符（A-Z，0-9，空格，逗号，句号等）以外的按键码，例如：Ctrl，Alt，Shift 等。
- MouseEvent 处理鼠标的事件。

功能分类	接口名	描述
鼠标行为	getAction()	获取鼠标设备产生事件的行为。
鼠标按键	getActionButton()	获取状态发生变化的鼠标按键。
	getPressedButtons()	获取所有按下状态的鼠标按键。
鼠标指针/位置	getCursor()	获取鼠标指针的位置。
	getCursorDelta(int axis)	获取鼠标指针位置相对上次的变化值。
	setCursorOffset(float offsetX, float offsetY)	设置相对屏幕的偏移位置信息。
鼠标滚轮	getScrollingDelta(int axis)	获取滚轮的滚动值。

表 8 MouseEvent 的主要接口

- MmiPoint 处理在指定给定的坐标系中的 x,y 和 z 坐标。

功能分类	接口名	描述
构造函数	MmiPoint(float px, float py)	创建一个只包含 x 和 y 坐标的 MmiPoint 对象。
	MmiPoint(float px, float py, float pz)	创建一个包含 x, y 和 z 坐标的 MmiPoint 对象。
坐标值	getX()	获取 x 坐标值。
	getY()	获取 y 坐标值。
	getZ()	获取 z 坐标值。
	toString()	返回包含 x、y、z 坐标值信息的字符串

表 9 MmiPoint 的主要接口

开发步骤

处理按钮事件

1. 参考 HarmonyOS 的 Component 的 API 创建 KeyEventListener;
2. 重写实现 KeyEventListener 类中的 onKeyEvent(Component component, KeyEvent event)方法;
3. 开发者根据自身需求处理存在按键被按下以及 KEY_DPAD_CENTER、KEY_DPAD_LEFT 等按键被按下后的具体实现。

```
1. private Component.KeyEventListener onKeyEvent = new Component.KeyEventListener()
2. {
3.     @Override
4.     public boolean onKeyEvent(Component component, KeyEvent keyEvent) {
5.         if (keyEvent.isKeyDown()) {
6.             ... // 检测到按键被按下，开发者根据自身需求进行实现
7.         }
8.         int keycode = keyEvent.getKeyCode();
9.         switch (keycode) {
10.            case KeyEvent.KEY_DPAD_CENTER:
11.                ... // 检测到 KEY_DPAD_CENTER 被按下，开发者根据自身需求进行实现
12.                break;
13.            case KeyEvent.KEY_DPAD_LEFT:
14.                ... // 检测到 KEY_DPAD_LEFT 被按下，开发者根据自身需求进行实现
15.                break;
16.            case KeyEvent.KEY_DPAD_UP:
17.                ... // 检测到 KEY_DPAD_UP 被按下，开发者根据自身需求进行实现
18.                break;
19.            case KeyEvent.KEY_DPAD_RIGHT:
20.                ... // 检测到 KEY_DPAD_RIGHT 被按下，开发者根据自身需求进行实现
21.                break;
22.            case KeyEvent.KEY_DPAD_DOWN:
23.                ... // 检测到 KEY_DPAD_DOWN 被按下，开发者根据自身需求进行实现
24.                break;
25.            default:
26.                break;
27.        }
28.        ...
29.    }
30.};
```

处理语音事件

使用多模输入的语音事件实现“可见即可说”的效果简易开发样例参考可见即可说开发指导。