第5章 Ability Kit

本章要点

- Ability Kit 概述。
- Stage 模型开发。
- UlAbility 和 ExtensionAbility。
- FA 模型。

本章知识结构图(图 5-1)

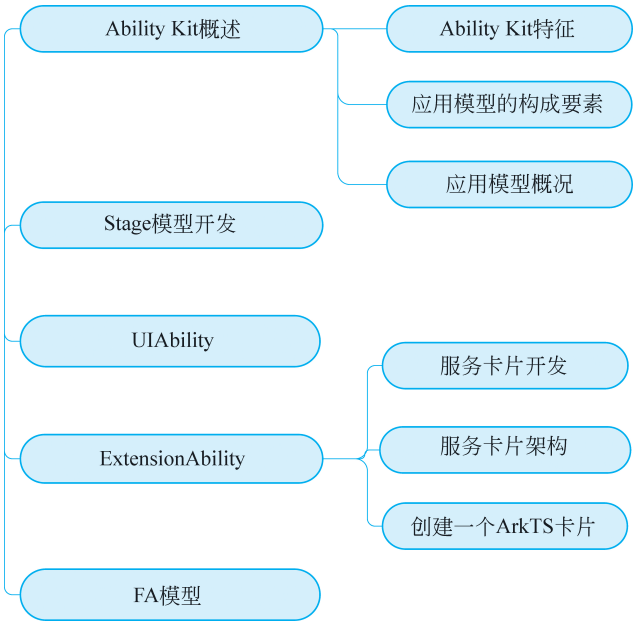

图 5-1 本章知识结构图

本章简介

本章重点解析 HarmonyOS 的 Ability Kit 框架及其核心应用模型,系统阐述 Stage 模型的设计理念与开发实践。通过对比 FA 模型的局限性,深入探讨 Stage 模型在组件共享、跨端协同、多窗口管理及生命周期优化等方面的技术优势。详细讲解 UIAbility 与 ExtensionAbility 组件的功能差异与应用场景,其中,UIAbility 专注于用户交互与界面管理,ExtensionAbility 则服务于卡片、输入法等特定场景的扩展能力。同时,结合服务卡片的开发实例,剖析其架构设计、数据绑定与动态更新机制,展现 HarmonyOS 在分布式

数据同步与原子化服务上的创新特性。本章还明确了 FA 模型因架构陈旧、维护成本高等问题被 Stage 模型全面替代的技术演进路径，为开发者构建高效、跨端协同的 HarmonyOS 应用提供完整的理论支持与实践指导。

5.1　Ability Kit 概述

Ability Kit(程序框架服务)提供了应用程序开发和运行的应用模型，是系统为开发者提供的应用程序所需能力的抽象提炼，它提供了应用程序必备的组件和运行机制。有了应用模型，开发者可以基于一套统一的模型进行应用开发，使应用开发更简单、高效。

5.1.1　Ability Kit 特征

HarmonyOS 相关程序框架服务 Ability Kit 具有以下几点特征。

(1) 为复杂应用而设计。

在 Ability Kit 中，多个应用组件共享同一个 ArkTS 引擎(运行 ArkTS 语言的虚拟机)实例，应用组件之间可以方便地共享对象和状态，同时减少复杂应用运行对内存的占用。并且 Ability Kit 采用面向对象的开发方式，提供模块化能力开发的支持，原生支持应用组件级的跨端迁移和多端协同，使得复杂应用代码可读性高、易维护性好、可扩展性强。

(2) Stage 模型实现了应用组件与 UI 解耦。

基于 Stage 模型开发应用支持多设备和多窗口形态。在跨端迁移场景下，系统在多设备的应用组件之间迁移数据/状态后，UI 便可利用 ArkUI 的声明式特点，通过应用组件中保存的数据/状态恢复 UI，便捷实现跨端迁移。同时，在多端协同场景下，应用组件具备组件间通信的 RPC 调用能力，天然支持跨设备应用组件的交互。

(3) 应用组件管理和窗口管理在架构层面解耦。

Stage 模型通过支持应用组件的动态裁剪(如无屏设备可移除窗口组件)和灵活扩展窗口形态，有效适配多样化设备形态需求，同时实现跨设备(桌面与移动终端)统一的生命周期管理机制，降低多端开发复杂度。在此基础上，该模型通过精准管控应用后台行为与资源占用，在保障功能扩展性的同时优化系统治理效能，达成应用能力开放与系统管控成本的最佳平衡，既避免了资源滥用风险，又为多场景协同提供了轻量化技术支撑。

(4) Stage 模型重新定义应用能力的边界，平衡应用能力和系统管控成本。

Stage 模型提供特定场景(如服务卡片、输入法)的应用组件，以便满足更多的使用场景。同时为保障用户体验，Stage 模型对后台应用进程进行了有序治理，应用程序不能随意驻留在后台，同时应用后台行为受到严格管理，防止恶意应用行为，使得后台进程管理规范化。

5.1.2　应用模型的构成要素

应用模型是系统为开发者提供的应用程序所需能力的抽象提炼，它提供了应用程序必备的组件和运行机制。有了应用模型，开发者可以基于一套统一的模型进行应用开发，使应用开发更简单、高效。

应用模型的构成要素包括应用组件、应用进程模型、应用线程模型、应用任务管理模型、应用配置文件。下面将按顺序逐一对其进行介绍。

1. 应用组件

应用组件是应用的基本组成单位,是应用的运行入口。在用户启动、使用和退出应用过程中,应用组件会在不同的状态间切换,这些状态称为应用组件的生命周期。应用组件提供生命周期的回调函数,开发者通过应用组件的生命周期回调感知应用的状态变化。应用开发者在编写应用时,首先需要编写的就是应用组件,同时还需编写应用组件的生命周期回调函数,并在应用配置文件中配置相关信息。这样,操作系统在运行期间通过配置文件创建应用组件的实例,并调度它的生命周期回调函数,从而执行开发者的代码。

2. 应用进程模型

应用进程模型定义应用进程的创建和销毁方式,以及进程间的通信方式。

3. 应用线程模型

应用线程模型定义应用进程内线程的创建和销毁方式、主线程和UI线程的创建方式、线程间的通信方式。

4. 应用任务管理模型(仅对系统应用开放)

应用任务管理模型定义任务的创建和销毁方式,以及任务与组件间的关系。所谓任务,即用户使用一个应用组件实例的记录。每次用户启动一个新的应用组件实例,都会生成一个新的任务。例如,用户启动一个视频应用,此时在“最近任务”界面,将会看到视频应用这个任务,当用户单击这个任务时,系统会把该任务切换到前台,如果这个视频应用中的视频编辑功能也是通过应用组件编写的,那么在用户启动视频编辑功能时,会创建视频编辑的应用组件实例,在“最近任务”界面中,将会展示视频应用、视频编辑两个任务。

5. 应用配置文件

应用配置文件中包含应用配置信息、应用组件信息、权限信息、开发者自定义信息等,这些信息在编译构建、分发和运行阶段分别提供给编译工具、应用市场和操作系统使用。

5.1.3 应用模型概况

随着系统的演进发展,HarmonyOS开发先后提供了两种应用模型,分别是FA(Feature Ability)模型和Stage模型。

(1) FA(Feature Ability)模型:从API 7开始支持的模型,已经不再主推。

(2) Stage模型:从API 9开始新增的模型,是目前主推且会长期演进的模型。在该模型中,由于提供了AbilityStage、WindowStage等类作为应用组件和Window窗口的“舞台”,因此称这种应用模型为Stage模型。Stage模型作为主推的应用模型,开发者通过它能够更加便利地开发出分布式场景下的复杂应用。

HarmonyOS主推Stage模型的原因有以下几点。

1. 为复杂应用而设计

Stage模型专为处理复杂应用而设计,它允许多个应用组件共享同一个ArkTS引擎实例,从而减少复杂应用运行时对内存的占用。这种设计不仅提升了内存使用效率,还降低了系统资源消耗。采用面向对象的开发方式,Stage模型使得复杂应用的代码结构更

加清晰，提高了代码的可读性和易维护性。此外，这种模型的可扩展性强，便于开发者根据需要添加新功能或进行功能扩展。

2. 原生支持应用组件级的跨端迁移和多端协同

Stage 模型实现了应用组件与 UI 的解耦合，这意味着应用组件可以在不同的设备和形态之间自由迁移，而无须进行大量修改。这种跨端迁移能力极大地增强了应用的灵活性和可用性。多端协同也是 Stage 模型的一大特点，它允许应用组件在多个设备之间协同工作，实现数据和状态的同步，为用户提供无缝的跨设备体验。

3. 支持多设备和多窗口形态

在 Stage 模型中，应用组件管理和窗口管理在架构层面被解耦合，这为系统提供了更大的灵活性，便于扩展窗口形态和支持更多设备类型。由于这种解耦合，应用组件可以在多种设备上使用同一套生命周期管理，简化了开发流程并提高了开发效率。

4. 平衡应用能力和系统管控成本

Stage 模型提供了特定场景下的应用组件，并规范化了后台进程管理。这有助于平衡应用的功能需求和系统资源的管控成本。通过这种方式，开发者可以更精确地控制应用组件的资源使用，同时确保系统整体的稳定性和性能。这种平衡对于资源有限的设备尤其重要，可以确保设备运行流畅且不会因资源过载而影响用户体验。

5.2　Stage 模型开发

在 HarmonyOS 中，Stage 模型和 FA 模型是两种不同的应用开发模型，它们在设计思想、组件类型、资源共享和内存占用、系统管理和控制能力，以及模型演进和主推程度等方面存在显著的差异。

Stage 模型是 HarmonyOS 3.1　Developer Preview 版本开始新增的模型，是目前主推且会长期演进的模型。在该模型中，由于提供了 AbilityStage、WindowStage 等类作为应用组件和窗口的“舞台”，因此称这种应用模型为 Stage 模型。Stage 模型之所以成为主推模型，源于其设计思想。它为复杂应用而设计，多个应用组件共享同一个 ArkTS 引擎实例，应用组件之间可以方便地共享对象和状态，同时减少复杂应用运行对内存的占用。采用面向对象的开发方式，使得复杂应用代码可读性高、易维护性好、可扩展性强。相比于 FA 模型，Stage 模型提供了更灵活的开发方式、更低的内存占用和更规范化的系统管理机制。未来 HarmonyOS 将在兼容 FA 模型的基础上，持续演进 Stage 模型。

它提供了一种更好的开发方式，更适用于多设备、分布式场景。Stage 模型的主要特点如下。

（1）组件共享：多个应用组件共享同一个 ArkTS 引擎实例，使得应用组件之间可以方便地共享对象和状态，同时减少复杂应用运行对内存的占用。

（2）多窗口管理：Stage 模型将应用的界面划分为多个独立的 Stage，每个 Stage 都有自己的窗口和界面布局，可以单独显示、隐藏和关闭。不同的 Stage 之间可以进行切换和互动，提供了更加丰富的用户交互方式。

（3）组件类型：提供 UIAbility 和 ExtensionAbility 两种类型的组件。UIAbility 组件用于与用户交互，而 ExtensionAbility 组件则提供场景化的服务扩展机制，但不提供自

定义服务的能力。

(4) 生命周期管理：UIAbility组件的生命周期包含创建、销毁、前台、后台状态，与界面强相关的获焦、失焦状态都放在窗口管理对象中，实现与窗口之间的弱耦合。

(5) 系统管理能力：对后台应用进程进行了有序治理，应用程序不能随意留驻在后台，同时应用后台行为受到严格管理，以防止恶意应用行为。

Stage模型提供UIAbility和ExtensionAbility两种类型的组件，这两种组件都有具体的类承载，支持面向对象的开发方式。

5.3 UIAbility

UIAbility组件是一种包含UI界面的应用组件，提供展示UI的能力，主要用于和用户交互。例如，图库类应用可以在UIAbility组件中展示图片瀑布流，在用户选择某个图片后，在新的页面中展示图片的详细内容。同时，用户可以通过返回键返回瀑布流页面。UIAbility组件的生命周期只包含创建/销毁/前台/后台等状态，与显示相关的状态通过WindowStage的事件暴露给开发者。

UIAbility也是系统调度的单元，为应用提供窗口在其中绘制界面。每一个UIAbility实例，都对应于一个最近任务列表中的任务。一个应用可以有一个UIAbility，也可以有多个UIAbility。

例如，浏览器应用可以通过一个UIAbility结合多页面的形式让用户进行搜索和浏览内容。而聊天应用增加一个“外卖功能”的场景，则可以将聊天应用中“外卖功能”的内容独立为一个UIAbility，当用户打开聊天应用的“外卖功能”，查看外卖订单详情，此时有新的聊天消息，即可以通过最近任务列表切换回聊天窗口继续进行聊天对话。

这样做的好处就是把功能解耦，进行一个模块一个模块的管理，这样极大地提高了项目的可维护性。

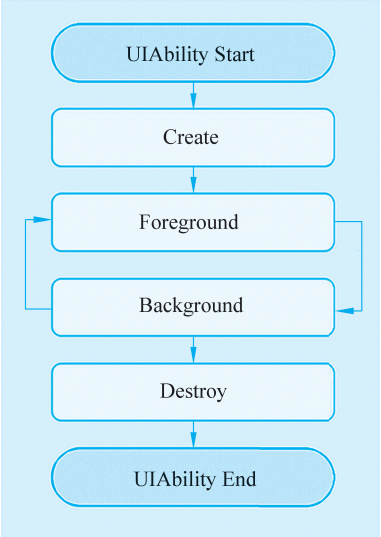

图5-2 UIAbility生命周期状态转换

对于一个UIAbility来说，通常是对应于多个页面。建议将一个独立的功能模块放到一个UIAbility中，以多页面的形式呈现。例如，新闻应用在浏览内容的时候，可以进行多页面的跳转使用。

接下来，介绍一下UIAbility的生命周期，当用户浏览、切换和返回到对应应用的时候，应用中的UIAbility实例会在其生命周期的不同状态之间转换。UIAbility类提供了很多回调，通过这些回调可以知晓当前UIAbility的某个状态已经发生改变，例如，UIAbility的创建和销毁，或者UIAbility发生了前后台的状态切换，如图5-2所示。

UIAbility的生命周期由系统回调管理，开发者需覆写以下方法。

onCreate()：初始化资源(如加载布局)。

onForeground()：界面即将可见时恢复数据或

动画。

onBackground()：界面退到后台时释放非必要资源。

onDestroy()：销毁资源，防止内存泄漏，如程序清单 5-1 所示。

程序清单 5-1　HM_App_2\ch5\MyApplication_5\entry\src\main\ets\entryability\EntryAbility.ets

```
export default class EntryAbility extends UIAbility {
  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam) {
    //初始化逻辑
  }
  onForeground() {
    //恢复前台状态
  }
  onBackground() {
    //进入后台
  }
  onDestroy() {
    //清理资源
  }
}
```

在 module.json5 中声明 UIAbility，需指定名称、入口路径等，如程序清单 5-2 所示。

程序清单 5-2　HM_App_2\ch5\MyApplication_5\entry\src\main\module.json5

```
{
  "module": {
    "abilities": [
      {
      "name": "EntryAbility",
      "srcEntry": "./ets/entryability/EntryAbility.ts",
      "description": "$string:description_entryability",
      "icon": "$media:icon",
      "label": "$string:entryability_label",
      "startWindowIcon": "$media:icon",
      "startWindowBackground": "$color:start_window_background"
      }
    ]
  }
}
```

1. 指定 UIAbility 的启动页面

应用中的 UIAbility 在启动过程中，需要指定启动页面，否则应用启动后会因为没有默认加载页面而导致白屏。可以在 UIAbility 的 onWindowStageCreate()生命周期回调中，通过 WindowStage 对象的 loadContent()方法设置启动页面，如程序清单 5-3 所示。

程序清单 5-3　HM_App_2\ch5\MyApplication_5\entry\src\main\ets\entryability\EntryAbility.ets

```
import { UIAbility } from '@kit.AbilityKit';
import { window } from '@kit.ArkUI';

export default class EntryAbility extends UIAbility {
```

```
    onWindowStageCreate(windowStage: window.WindowStage): void {
      //Main window is created, set main page for this ability
      windowStage.loadContent('pages/Index', (err, data) =>{
        //…
      });
    }
    //…
  }
```

2. 获取 UIAbility 的上下文信息

UIAbility 类拥有自身的上下文信息,该信息为 UIAbilityContext 类的实例,通过 UIAbilityContext 可以获取 UIAbility 的相关配置信息,如果需要在页面中获得当前 Ability 的 Context,可调用 getContext 接口获取当前页面关联的 UIAbilityContext 或 ExtensionContext。

在 UIAbility 中可以通过 this.context 获取 UIAbility 实例的上下文信息,如程序清单 5-4 所示。

程序清单 5-4　HM_App_2\ch5\MyApplication_5\entry\src\main\ets\entryability\EntryAbility.ets

```
import { UIAbility, AbilityConstant, Want } from '@kit.AbilityKit';

export default class EntryAbility extends UIAbility {
  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
    //获取 UIAbility 实例的上下文
    let context =this.context;
    //…
  }
}
```

通过@Entry 装饰的组件,负责具体 UI 布局和交互。UIAbility 通过 windowStage.loadContent()加载 Page,如程序清单 5-5 所示。

程序清单 5-5　HM_App_2\ch5\MyApplication_5\entry\src\main\ets\entryability\EntryAbility.ets

```
//UIAbility 中加载页面
onWindowStageCreate(windowStage: window.WindowStage) {
  windowStage.loadContent('pages/IndexPage', (err) =>{
    if (err) console.error('加载页面失败');
  });
}
```

通过 Want 对象启动其他 UIAbility,并支持参数传递:Want 是组件间通信的核心对象,用于启动其他组件(如 UIAbility、ServiceAbility)并传递数据。当一个应用内包含多个 UIAbility 时,存在应用内启动 UIAbility 的场景。

5.4　ExtensionAbility

ExtensionAbility 组件是一种面向特定场景的应用组件。开发者并不直接从 ExtensionAbility 组件派生,而是需要使用 ExtensionAbility 组件的派生类。目前,

ExtensionAbility 组件有用于卡片场景的 FormExtensionAbility，用于输入法场景的 InputMethodExtensionAbility，用于闲时任务场景的 WorkSchedulerExtensionAbility 等多种派生类，这些派生类都是基于特定场景提供的。例如，用户在桌面创建应用的卡片，需要应用开发者从 FormExtensionAbility 派生，实现其中的回调函数，并在配置文件中配置该能力。ExtensionAbility 组件的派生类实例由用户触发创建，并由系统管理生命周期。

在 Stage 模型上，三方应用开发者不能开发自定义服务，而需要根据自身的业务场景通过 ExtensionAbility 组件的派生类来实现。

UIExtensionAbility 提供了 onCreate、onSessionCreate、onSessionDestroy、onForeground、onBackground 和 onDestroy 生命周期回调，根据需要重写对应的回调方法。

onCreate：当 ExtensionAbility 创建时回调，执行初始化业务逻辑操作。

onSessionCreate：当 ExtensionAbility 界面内容对象创建后调用。

onSessionDestroy：当 ExtensionAbility 界面内容对象销毁后调用。

onForeground：当 ExtensionAbility 从后台转到前台时触发。

onBackground：当 ExtensionAbility 从前台转到后台时触发。

onDestroy：当 ExtensionAbility 销毁时回调，可以执行资源清理等操作。

5.4.1　服务卡片开发

鸿蒙服务卡片是华为 HarmonyOS 操作系统中的一个创新功能，它旨在提升用户体验和设备操作效率。服务卡片是一种全新的交互方式，能够将应用程序的关键信息和服务以小部件的形式展现在桌面或其他界面中，让用户无须直接打开应用程序，就能快速预览和使用所需的功能或内容。

5.4.2　服务卡片架构

鸿蒙服务卡片的架构体现了 HarmonyOS 系统的设计理念，即通过组件化、分布式和原子化服务来优化用户体验和提升交互效率。

在服务卡片的运行机制中，卡片使用方和卡片提供方通过标准化接口协同工作，形成松耦合的组件化架构：前者负责卡片的展示管理与宿主环境适配，后者专注于卡片业务逻辑与动态交互实现。这种分离设计既保证了卡片的跨场景复用性（如不同设备桌面均可嵌入同一卡片），又通过分布式能力实现多端数据同步（如手机与平板卡片状态实时联动）。同时，原子化服务特性使卡片能够脱离应用独立运行，按需触达用户，精准匹配“服务找人”的智慧化体验目标。

卡片使用方是承载服务卡片显示的宿主应用（如桌面、负一屏等），负责管理卡片在宿主中的布局、位置及生命周期。开发者需在配置文件中声明卡片的基本信息（如尺寸、更新周期等），用户可通过长按桌面进入编辑模式，将卡片拖曳至指定区域，系统将根据配置动态渲染卡片视图。使用方通过卡片代理（FormManager）与提供方交互，实现卡片的添加、删除、更新等操作，但不干涉卡片内部业务逻辑。例如，在桌面场景中，卡片使用方根

据用户设置的窗口小工具尺寸,自动调整卡片占位区域,确保视觉一致性,如图5-3所示。

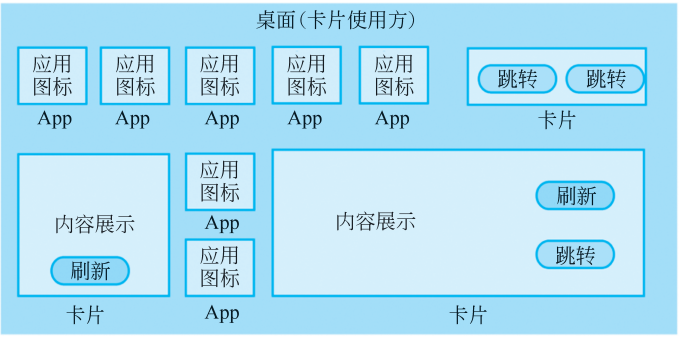

图5-3 桌面场景中卡片布局

卡片提供方是实际开发卡片功能的应用,需通过ArkUI框架实现卡片的UI构建、数据绑定及事件响应。开发者使用@Component装饰器定义卡片组件,在build()方法中通过Row、Column等容器组件与Text、Image等控件组合布局,并利用$r语法引用资源文件实现多端适配。卡片逻辑层通过postCardAction处理用户交互(如单击跳转Ability),同时可借助FormProvider能力实现动态数据更新。例如,当用户单击卡片中的"刷新"按钮时,提供方通过updateForm()方法向使用方推送最新数据,触发界面重绘。卡片提供方需在应用配置中导出卡片能力,声明支持的卡片规格,确保与卡片使用方的兼容性,如图5-4所示。

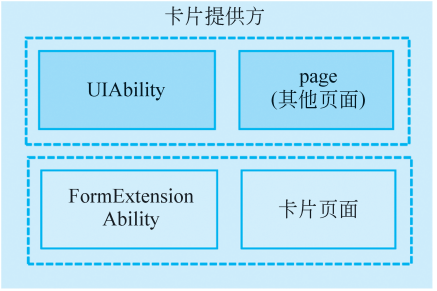

图5-4 卡片提供方

接下来,介绍与卡片相关的一些模块及其功能,包括卡片扩展模块(FormExtensionAbility)、其上下文环境(FormExtensionContext)、卡片信息(formInfo)、卡片数据绑定(formBindingData)、卡片提供方接口(formProvider)以及页面布局(Card etcs)等,并说明了卡片及相关模块的配置文件位置。

卡片扩展模块(FormExtensionAbility)是一个核心模块,负责卡片的创建、销毁以及刷新等生命周期的操作。就像是一个卡片的"管家",管理着卡片从出生(创建)到死亡(销毁)以及在生命周期中需要更新(刷新)的各种操作。例如,在一个天气应用中,当用户打开应用时需要创建显示当前天气的卡片(创建操作),当天气数据有更新时需要刷新卡片显示新的天气信息(刷新操作),当用户关闭应用或者切换到其他页面不需要显示该卡片时则可以销毁卡片(销毁操作)。

卡片扩展模块的上下文环境(FormExtensionContext)为卡片扩展模块提供了必要的接口和能力支持,是卡片扩展模块运行的"环境基础"。可以理解为卡片扩展模块在执行各种操作(如创建、销毁、刷新等)时需要依赖的一些条件和工具,而这些条件和工具就是由上下文环境来提供的。例如,在执行卡片创建操作时,可能需要获取当前设备的屏幕分辨率等信息来确定卡片的尺寸和布局,这些信息就是通过上下文环境提供的接口来获取的。

卡片信息(formInfo)主要提供关于卡片的信息以及状态等相关类型和枚举。这就像是卡片的“身份证”,记录着卡片自身的一些属性和状态信息。如卡片的类型(是天气卡片、新闻卡片还是日历卡片等)、卡片当前的显示状态(是展开状态还是折叠状态等)等信息都是通过卡片信息模块来提供的。

卡片数据绑定(formBindingData)负责卡片数据的绑定能力,包括创建FormBindingData 对象以及相关信息的描述。可以简单理解为将卡片要显示的数据和卡片的显示元素进行关联的过程。在一个新闻卡片中,将新闻的标题、内容、发布时间等数据与卡片上相应的文本框、标签等显示元素进行绑定,使得这些数据能够正确地显示在卡片上。

卡片提供方接口(formProvider)提供了与卡片提供方相关的接口,通过该接口可以实现更新卡片、设置卡片更新时间、获取卡片信息、请求发布卡片等操作。这就像是一个对外的“服务窗口”,其他模块或外部应用可以通过这个接口来对卡片进行各种操作。一个定时任务模块可以通过 formProvider 接口来设置卡片的更新时间,当到达设定时间时自动触发卡片的更新操作;或者一个管理模块可以通过该接口获取卡片的信息来进行统计和管理等。

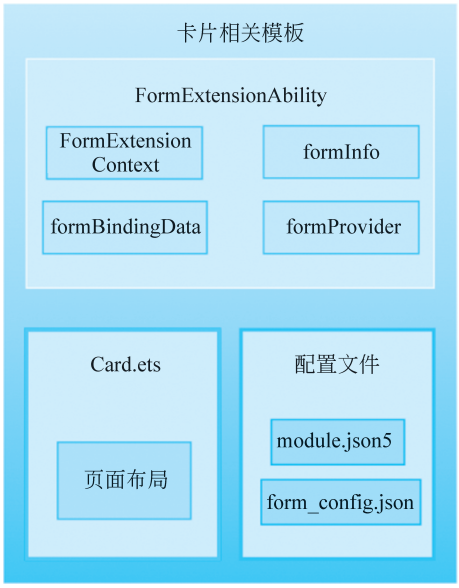

图 5-5　卡片相关模板

页面布局(Card etcs)提供声明式范式的 UI 接口能力,用于对卡片的布局进行设置。可以理解为是对卡片的外观和显示样式进行设计和安排的模块。确定卡片上各个元素(如图片、文字、按钮等)的位置、大小、颜色等样式信息,使得卡片能够以美观、合理的方式呈现给用户,如图 5-5 所示。

ArkTS 卡片模块如图 5-6 所示。

在 module.json5 配置文件中的 extensionAbilities 标签下配置 FormExtensionAbility 相关信息;在 resources/base/profile/目录下的 form_config.json 配置文件中配置卡片(WidgetCards 等)相关信息。这些配置文件就像是卡片及相关模块的“说明书”和“设置手册”,通过在这些文件中进行相应的设置来确定各个模块的具体行为和卡片的各种属性等。

5.4.3　创建一个 ArkTS 卡片

接下来,我们需要在已有的项目中创建卡片,本节详细介绍一下 ExtensionAbility 应用组件示例。

首先,打开 DevEco Stdio,创建一个 MyApplication5 项目。创建好项目后,在 page 中右键单击 New,当光标移动到 New 上后,可以在末尾看到 Service Widget。将光标移