HUAWEI

Stage应用模型

华为云 一切皆服务

前言

- Stage应用模型是HarmonyOS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，为开发者提供了一个统一的开发框架。
- UIAbility组件是Stage应用模型的重要功能模块，专门负责处理与用户界面相关的事务；基于Stage应用模型，开发者可以便捷使用UIAbility来设计和管理应用的用户界面。
- 本章将带领开发者们初识Stage应用模型，了解Stage应用模型的核心概念，学习在Stage应用模型下，使用UIAbility组件进行应用开发的基本流程及相关概念。

课程目标

- 学完本课程后，您将能够：
  - 了解Stage应用模型的核心概念；
  - 了解UIAbility应用组件的核心概念；
  - 掌握在Stage模型下，使用UIAbility组件进行应用开发的基本流程。

目录

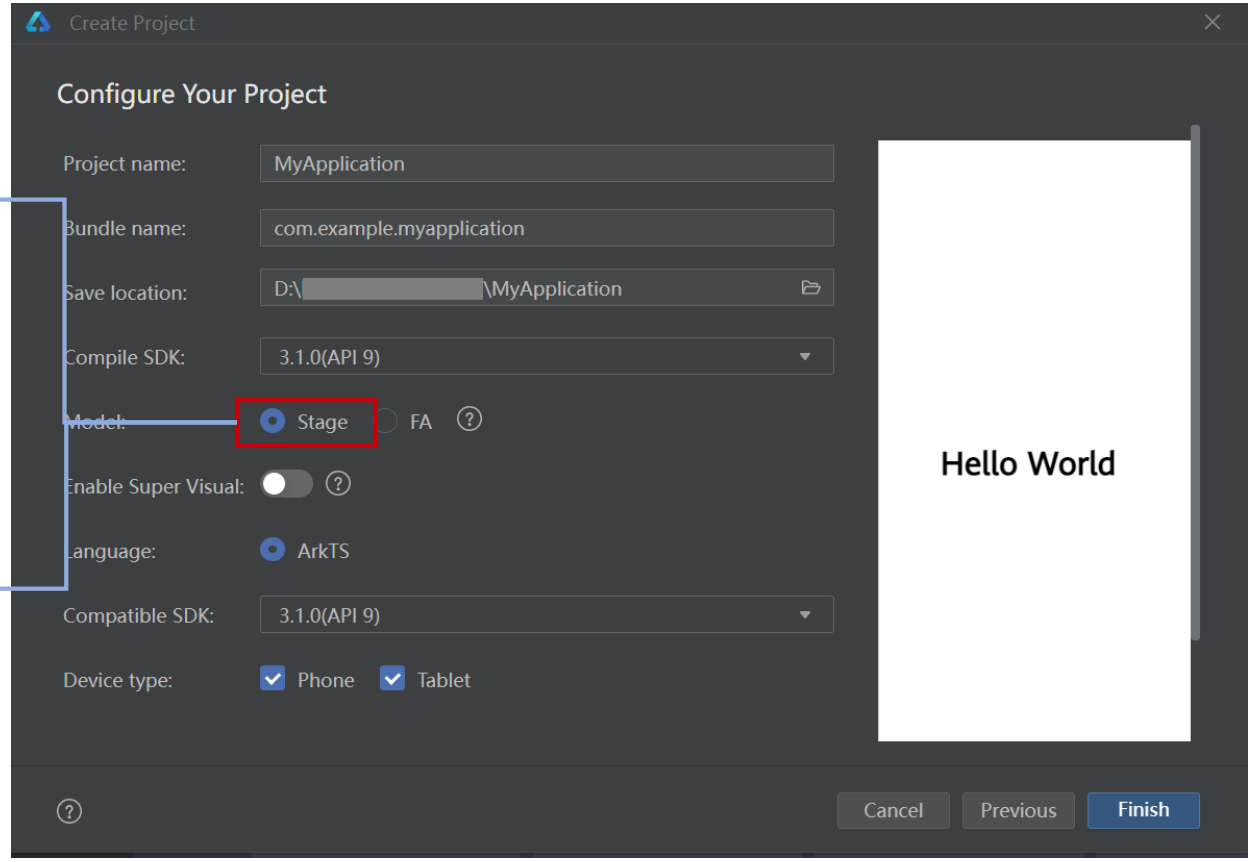
1. Stage应用模型概述
2. UIAbility应用组件
   - UIAbility组件概述
   - UIAbility组件生命周期
   - UIAbility组件启动模式
   - UIAbility基本用法
   - UIAbility组件间交互（设备内）

什么是Stage应用模型？

- 在前面的章节学习中，每当我们创建应用工程时，应用模型都会选择Stage模型。

Stage模型到底是什么?

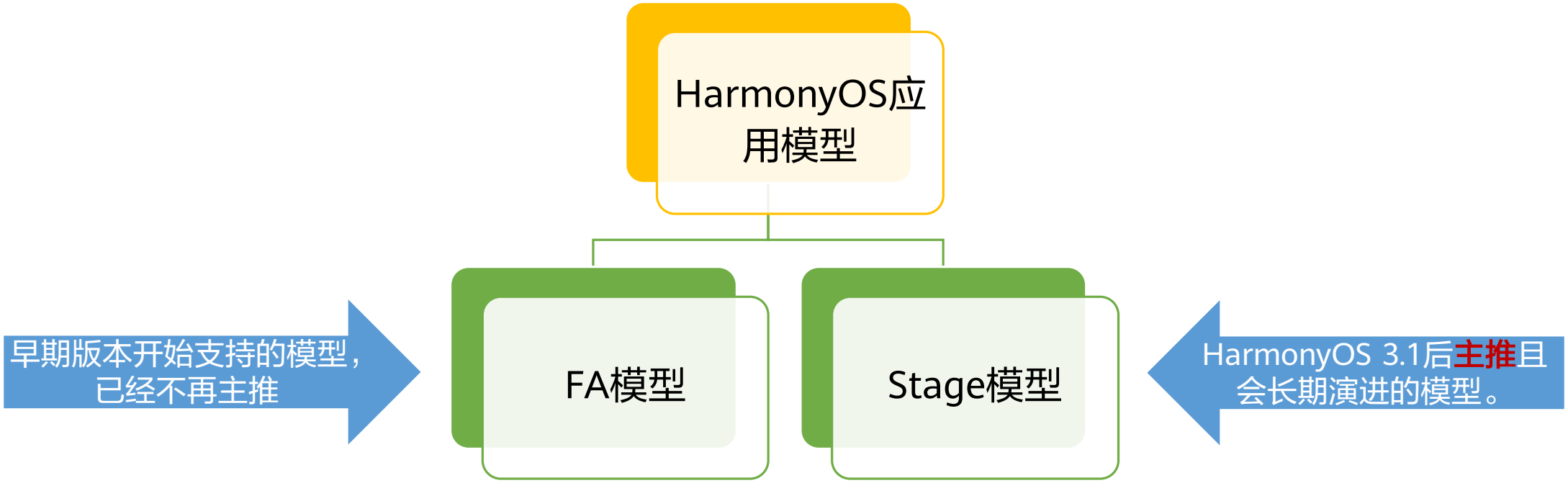
如何基于Stage模型进行应用开发?



HUAWEI

Stage模型的定义

- 应用模型是HarmonyOS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。而Stage作为应用模型，开发者可以基于这一套统一的模型进行应用开发，使应用开发更简单、高效。



Stage应用模型的设计思想

- Stage模型之所以成为主推模型，源于其设计思想。Stage模型的设计基于如下出发点：

为复杂应用而设计

多组件共享同一个ArkTS引擎实例（运行ArkTS语言的虚拟机），组件间共享对象和状态，减少复杂应用运行内存的占用。

面向对象的开发方式，使得代码的可读性高、易维护性好、可扩展性强。

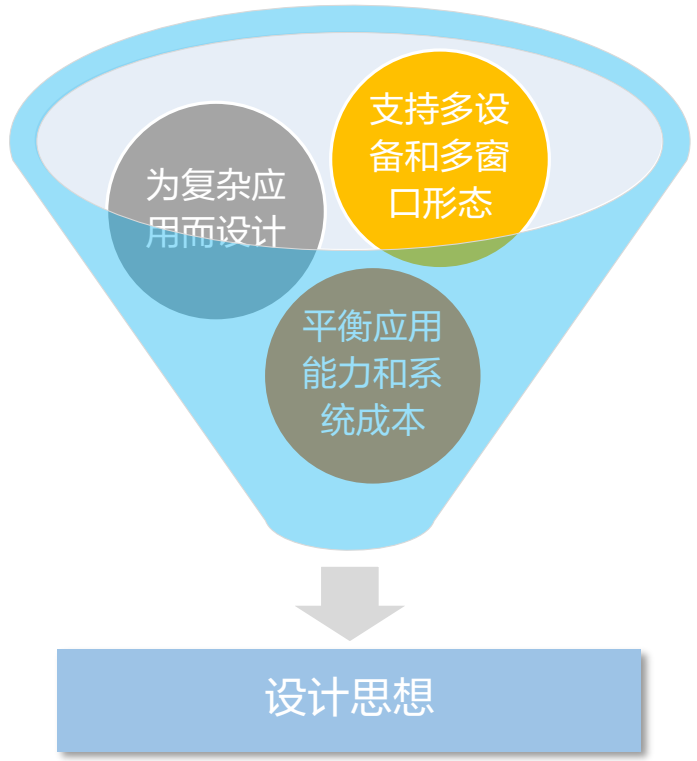
支持多设备和多窗口形态

便于系统对应用组件进行裁剪，便于系统扩展窗口形态，在多设备上，应用组件可使用同一套生命周期。

平衡应用能力和系统成本

提供特定场景（如卡片、输入法）的应用组件，以满足更多的使用场景。

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。



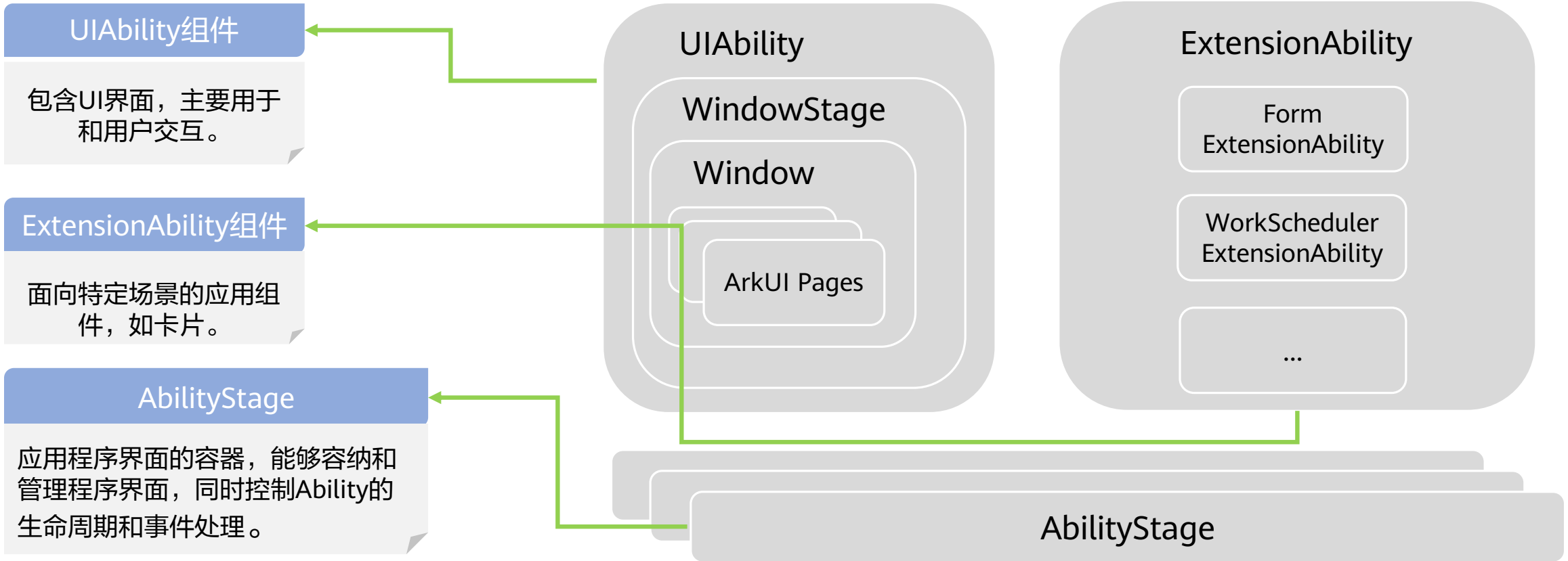
HUAWEI

Stage应用模型的特点

- HarmonyOS的Stage应用模型具有多个显著特点，这些特点使其在构建和管理复杂应用时能够展现出强大的优势。

Stage应用模型的主要结构（1）

- Stage模型中的应用组件是由Ability这个基础概念演化而来，Ability是应用程序框架中最基本的抽象单位，是能够完成独立功能的应用组件，在 Stage 模型中，Ability分两大类：



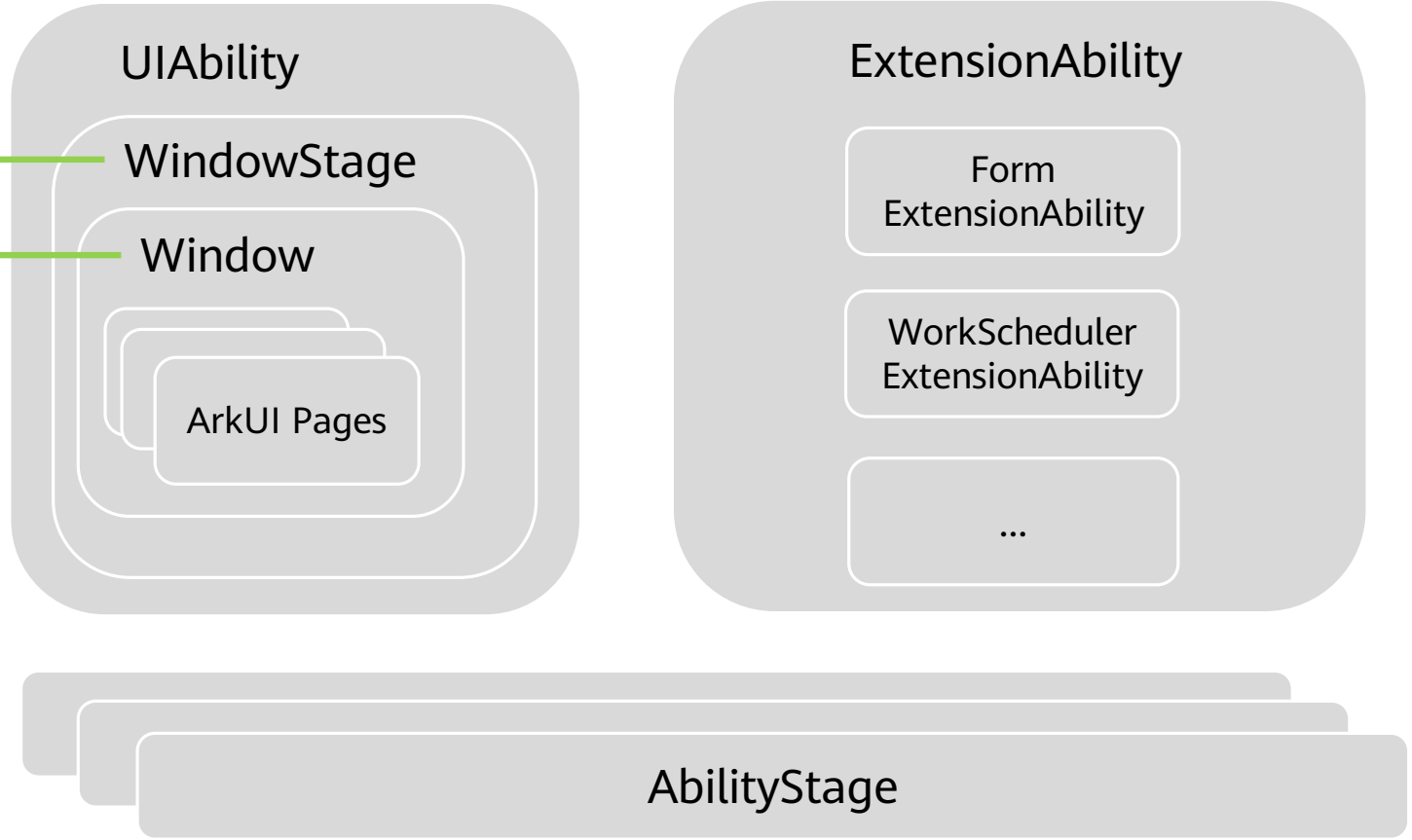
Stage应用模型的主要结构 (2)

WindowStage

应用程序内部的窗口管理器，每个UIAbility实例都与一个WindowStage实例绑定。UIAbility通过WindowStage来持有一个窗口。

Window

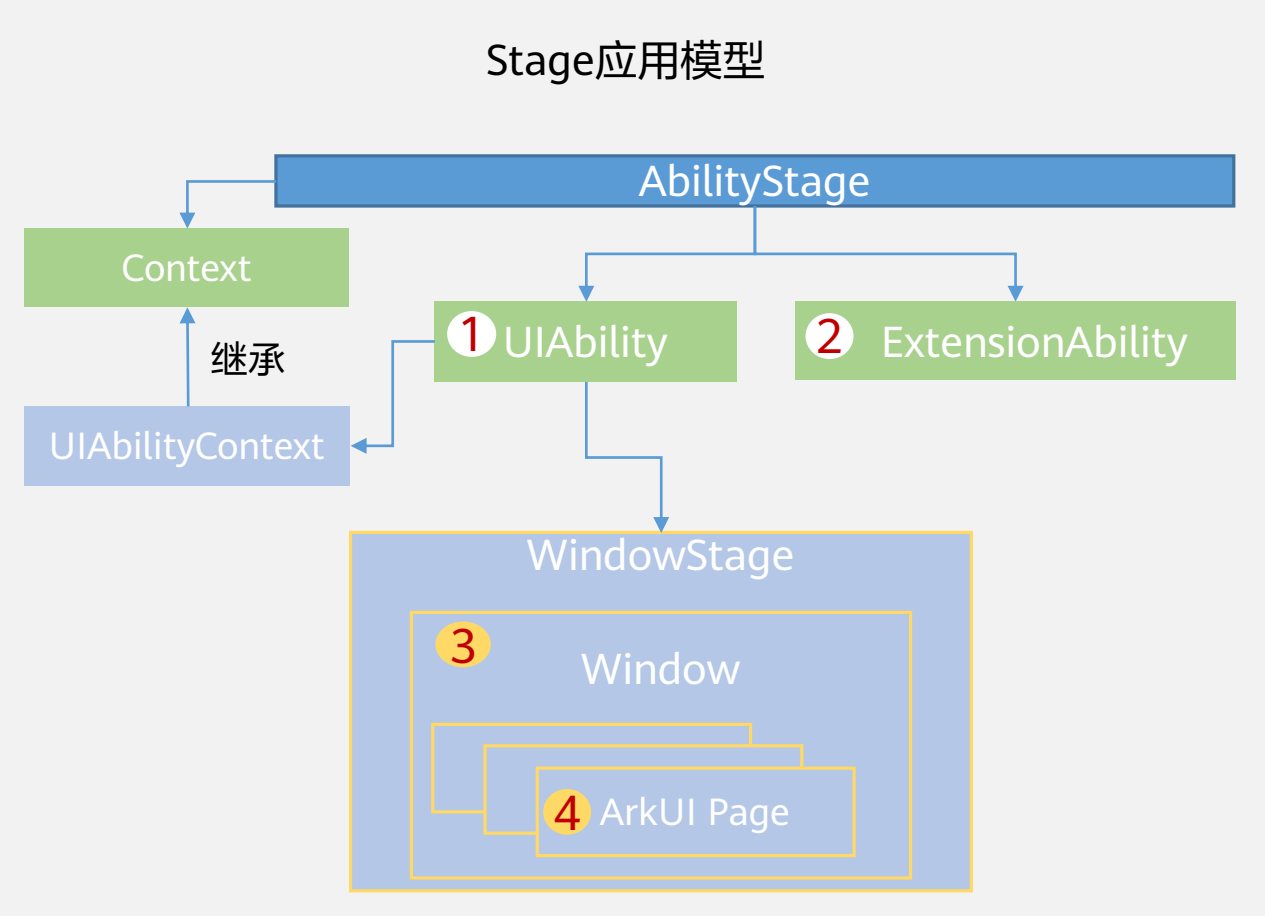
为ArkUI提供了绘制区域，是应用与用户进行交互的界面基础。



HUAWEI

Stage应用模型的主要结构 (3)

- 下面通过一个天气应用的实例，来介绍Stage模型结构中的基本概念。



HUAWEI

目录

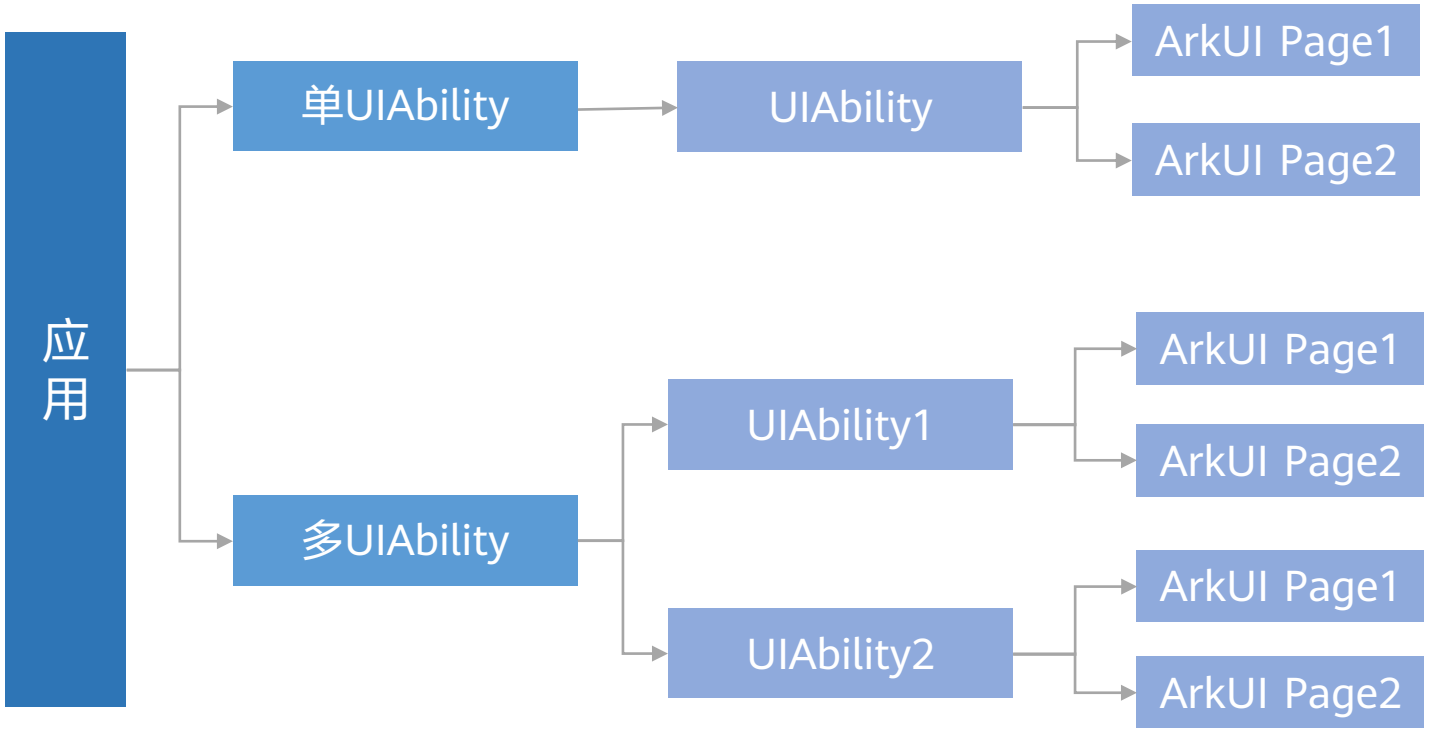
1. Stage应用模型概述

2. **UIAbility应用组件**

   - **UIAbility组件概述**
   - UIAbility组件生命周期
   - UIAbility组件启动模式
   - UIAbility基本用法
   - UIAbility组件间交互（设备内）

UIAbility应用组件的定义

- UIAbility组件是一种包含UI界面的应用组件，主要用于和用户交互。一个UIAbility组件中可以通过多个页面来实现一个功能模块。每一个UIAbility组件实例，都对应于一个最近任务列表中的任务。



UIAbility应用组件的声明配置

- 为使应用能够正常使用UIAbility，需要在module.json5配置文件的abilities标签中声明UIAbility的名称、入口、标签等相关信息。

```
{
  "module": {
  "abilities": [
    {
      "name": "EntryAbility", // UIAbility组件的名称
      "srcEntry": "./ets/entryability/EntryAbility.ts", // UIAbility组件的代码路径
      "description": "$string:EntryAbility_desc", // UIAbility组件的描述信息
      "icon": "$media:icon", // UIAbility组件的图标
      "label": "$string:EntryAbility_label", // UIAbility组件的标签
      "startWindowIcon": "$media:icon", // UIAbility组件启动页面图标资源文件的索引
      "startWindowBackground": "$color:start_window_background",  // UIAbility组件启动页面背景颜色资源文件的索引
    }
  ]
 }
}
```

目录

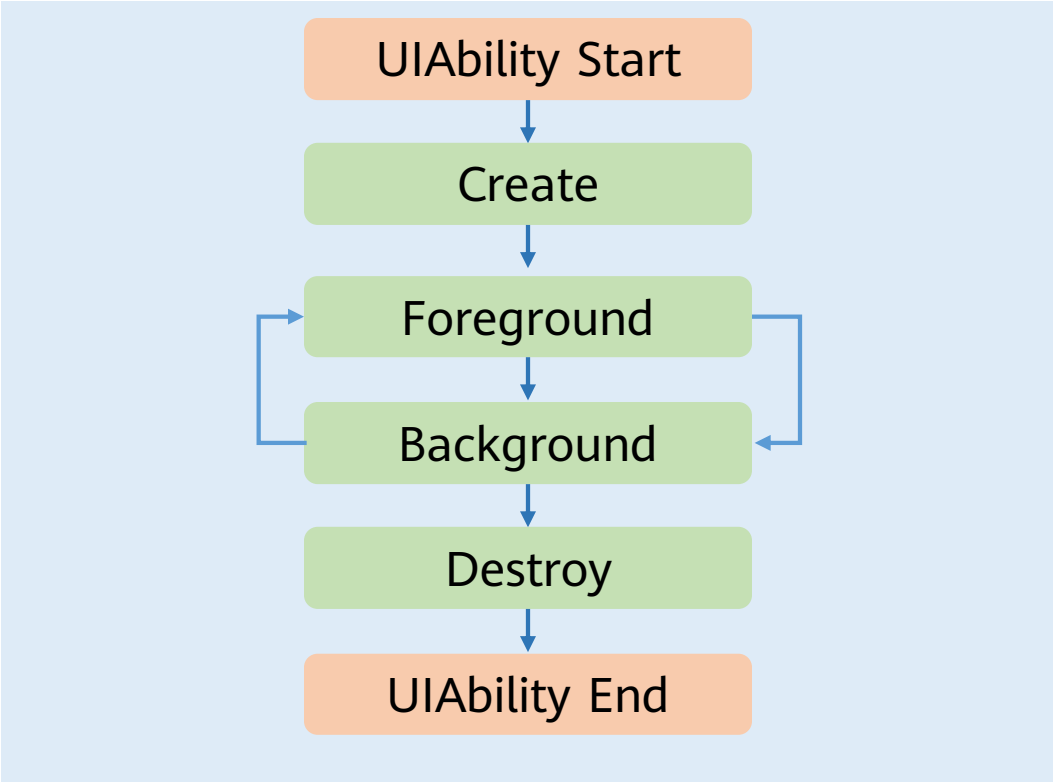
1. Stage应用模型概述
2. **UIAbility应用组件**
   - UIAbility组件概述
   - **UIAbility组件生命周期**
   - UIAbility组件启动模式
   - UIAbility基本用法
   - UIAbility组件间交互（设备内）

初探UIAbility生命周期

- 当我们在打开、切换和返回到某款应用时，应用中的UIAbility实例都会在其生命周期的不同状态之间转换，以使用华为天气应用为例。

UIAbility应用组件生命周期概述

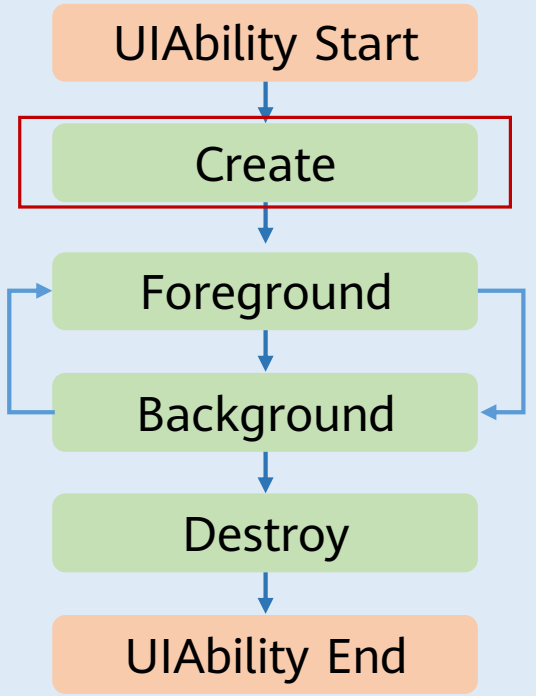
- UIAbility类提供了一系列回调，通过这些回调可以知道当前UIAbility实例的某个状态发生改变，会经过UIAbility实例的创建和销毁，或者UIAbility实例发生了前后台的状态切换。



Create状态

- Create状态为在应用加载过程中，UIAbility实例创建完成时触发，系统会调用onCreate()回调。可以在该回调中进行页面初始化操作，例如变量定义资源加载等，用于后续的UI界面展示。

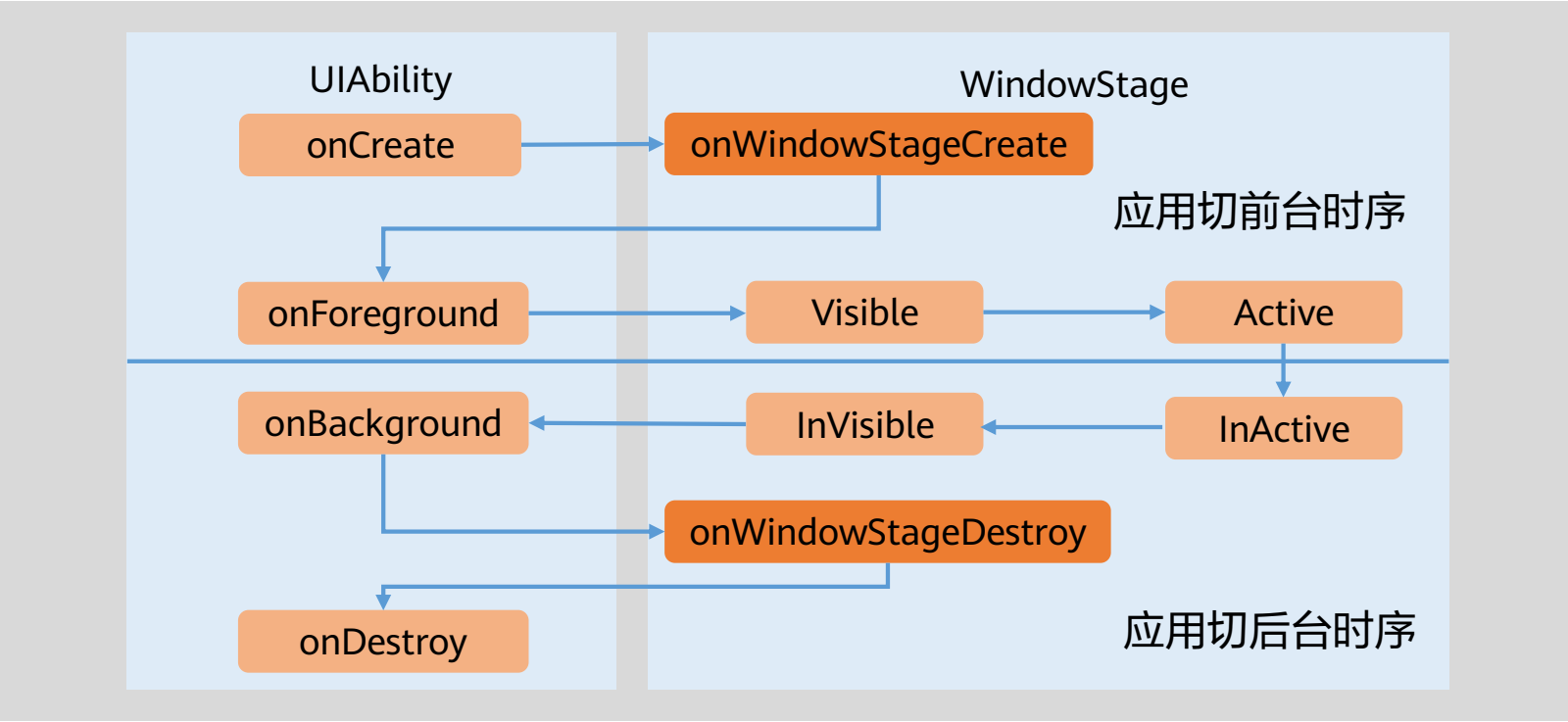
```
// EntryAbility.ts文件

import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class EntryAbility extends UIAbility {
    onCreate(want, launchParam) {
        // 页面初始化
    }
    ...
}
```



HUAWEI

WindowStageCreate和WindowStageDestroy状态 (1)

- UIAbility实例创建完成之后，在进入Foreground之前，系统会创建一个WindowStage。WindowStage创建完成后会进入onWindowStageCreate()回调，可以在该回调中设置UI界面加载、设置WindowStage的事件订阅。



WindowStageCreate和WindowStageDestroy状态 (2)

- 在onWindowStageCreate()回调中通过loadContent()方法设置应用要加载的页面并根据需要订阅WindowStage的事件（获焦/失焦、可见/不可见）。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class EntryAbility extends UIAbility {

    onWindowStageCreate(windowStage: Window.WindowStage) {
        // 设置WindowStage的事件订阅（获焦/失焦、可见/不可见）
        // 设置UI界面加载
        windowStage.loadContent('pages/Index', (err, data) => {
            ...
        });
    }
}
```

WindowStageCreate和WindowStageDestroy状态 (3)

- 在UIAbility实例销毁之前，则会先进入onWindowStageDestroy()回调，可以在该回调中释放UI界面资源。例如在onWindowStageDestroy()中注销获焦/失焦等WindowStage事件。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class EntryAbility extends UIAbility {
    ...
    onWindowStageDestroy() {
        // 释放UI界面资源
    }
}
```

Foreground和Background状态 (1)

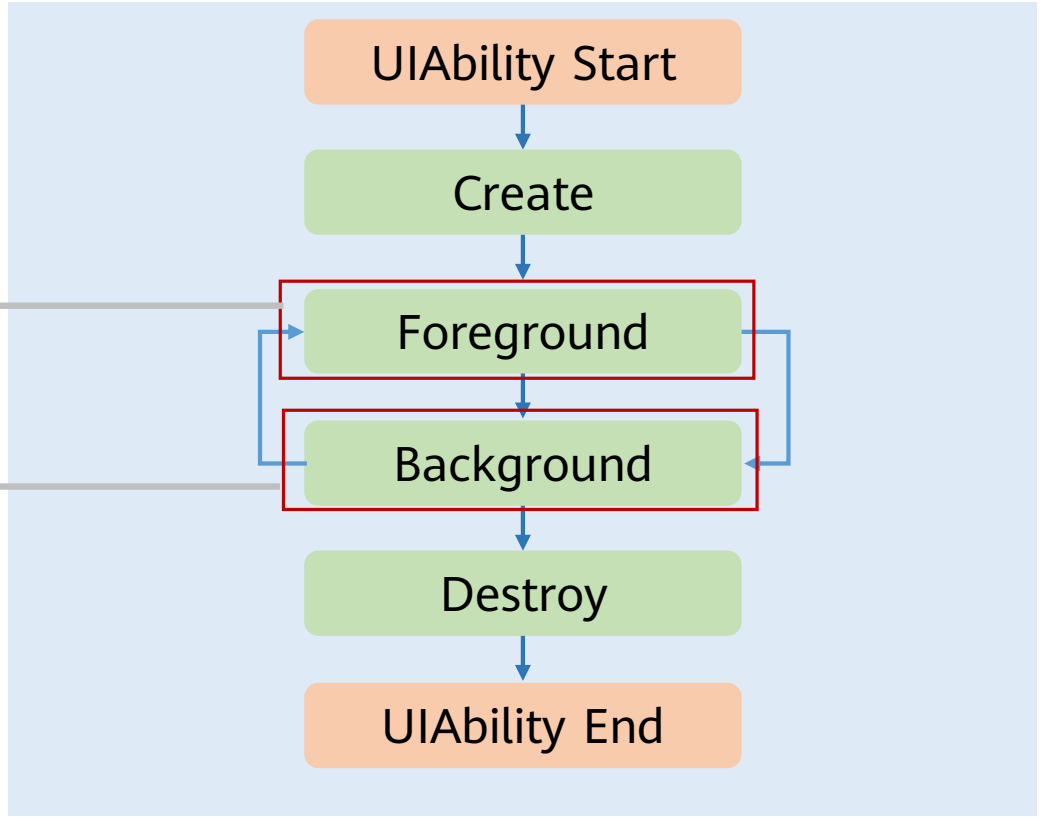
- Foreground和Background状态分别在UIAbility实例切换至前台和切换至后台时触发，对应于onForeground()回调和onBackground()回调。

onForeground( )回调

在UIAbility的UI界面可见之前，如UIAbility切换至前台时触发。

onBackground( )回调

在UIAbility的UI界面完全不可见之后，如UIAbility切换至后台时候触发。



Foreground和Background状态 (2)

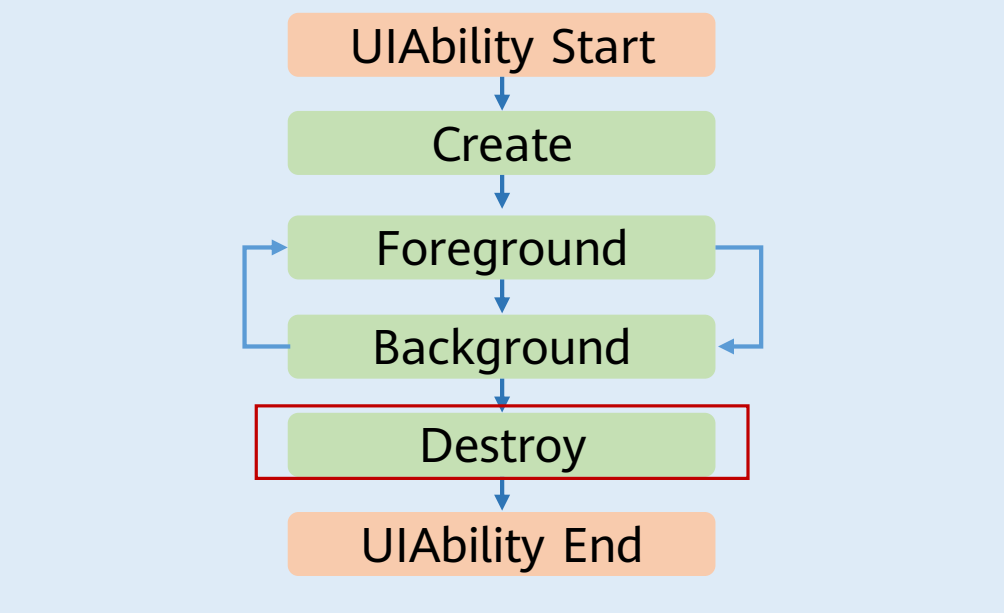
- 以应用使用定位功能为例，假设应用已获得用户的定位权限授权。在UI界面显示之前，可以在onForeground()回调中开启定位功能，从而获取到当前的位置信息。当应用切换到后台状态，可以在onBackground()回调中停止定位功能，以节省系统的资源消耗。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';

export default class EntryAbility extends UIAbility {
    onForeground() {
        // 申请系统需要的资源，或者重新申请在onBackground中释放的资源
    }

    onBackground() {
        // 释放UI界面不可见时无用的资源，或者在此回调中执行较为耗时的操作
        // 例如状态保存等
    }
}
```

Destory状态

- Destroy状态在UIAbility实例销毁时触发。可以在onDestroy()回调中进行系统资源的释放、数据的保存等操作。例如调用terminateSelf()方法停止当前UIAbility实例，从而完成UIAbility实例的销毁；或者用户使用最近任务列表关闭该UIAbility实例，完成UIAbility的销毁。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class EntryAbility extends UIAbility {
    onDestroy() {
        // 系统资源的释放、数据的保存等
    }
}
```

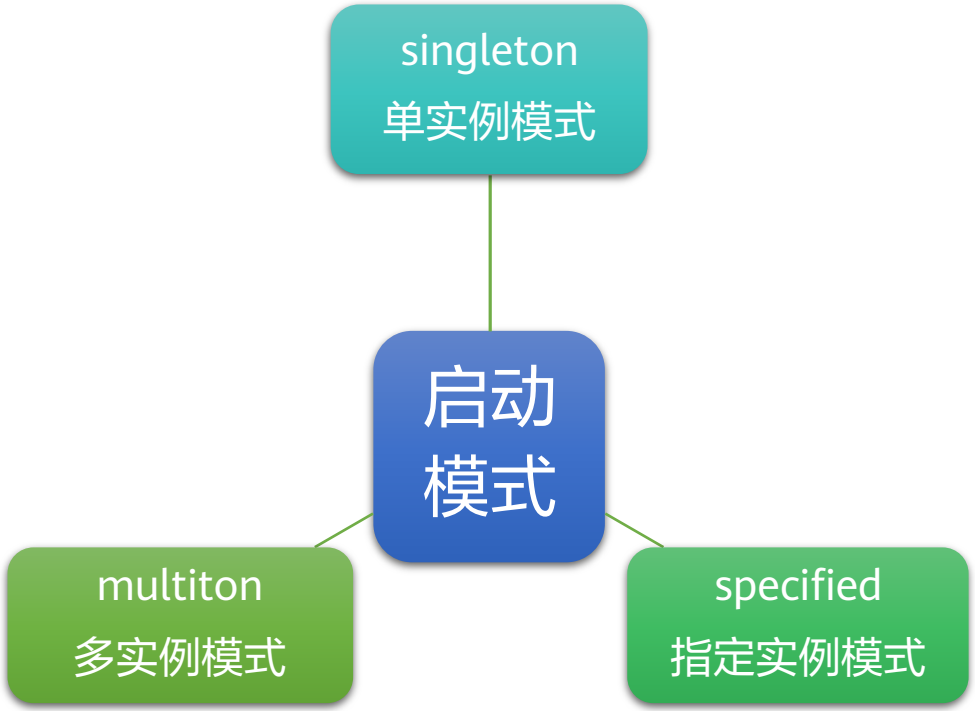


目录

1. Stage应用模型概述
2. **UIAbility应用组件**
   - UIAbility组件概述
   - UIAbility组件生命周期
   - **UIAbility组件启动模式**
   - UIAbility基本用法
   - UIAbility组件间交互（设备内）

UIAbility组件启动模式

- 当启动一个UIAbility时，系统就会产生一个UIAbility实例。 UIAbility的启动模式是指UIAbility实例在启动时的不同呈现状态。针对不同的业务场景，系统提供三种启动模式。



singleton启动模式

- singleton启动模式为单实例模式（默认模式），UIAbility启动时，若同类型实例已存在，则复用该实例。系统中同一UIAbility类型仅有一个实例，最近任务列表中也只显示一个该类型实例。

示例理解

multiton启动模式

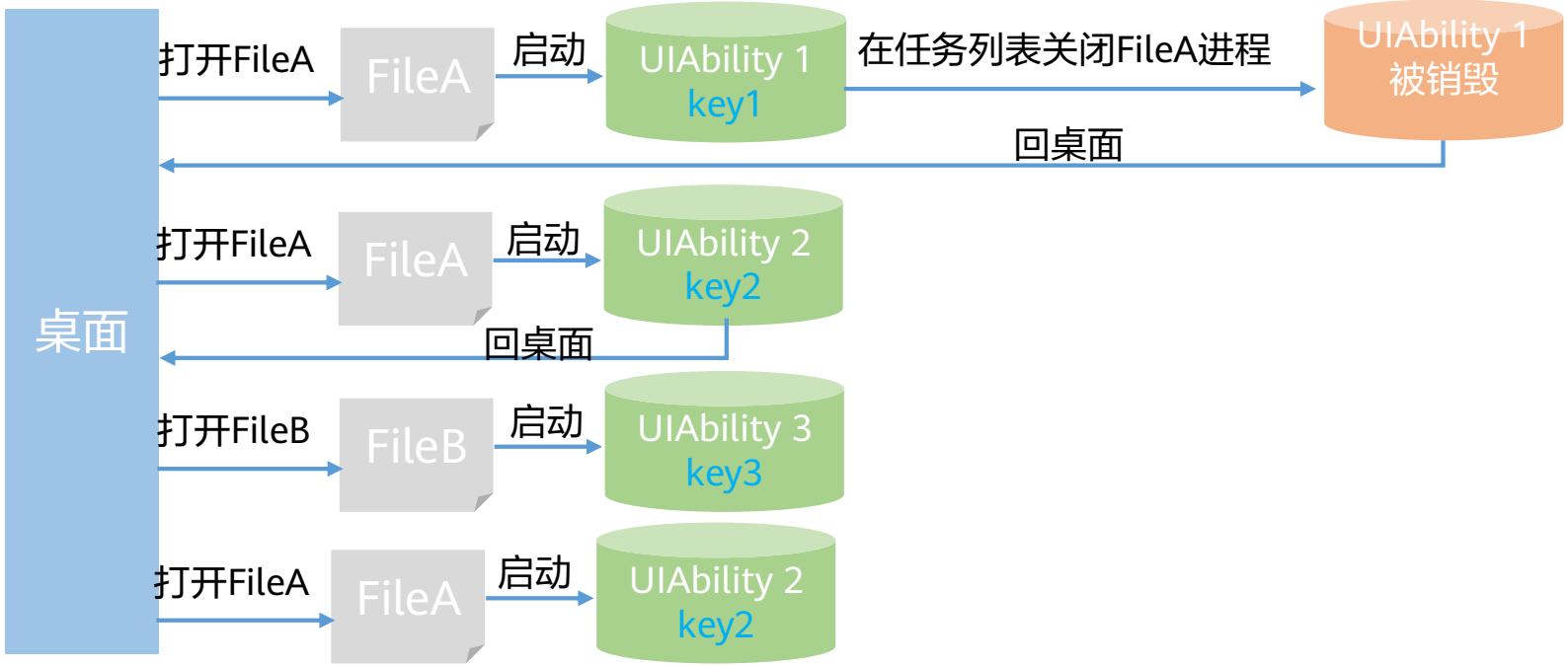
- multiton启动模式为多实例模式，每次启动UIAbility时，都会在应用进程中创建一个新的该类型UIAbility实例，即在最近任务列表中可以看到有多个该类型的UIAbility实例。

示例理解

HUAWEI

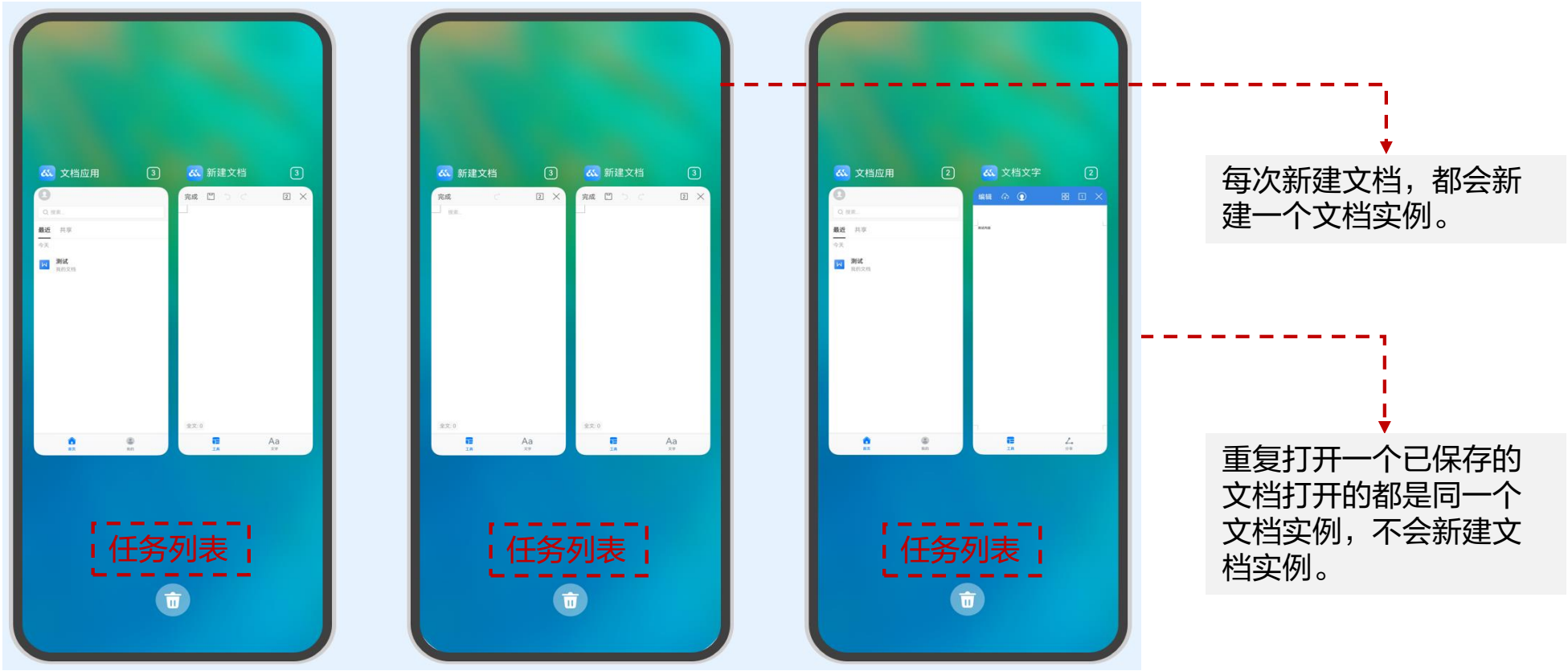
specified启动模式（1）

- specified启动模式为指定实例模式，针对一些特殊场景使用（例如文档应用中每次新建文档希望都能新建一个文档实例，重复打开一个已保存的文档希望打开的都是同一个文档实例）。



specified启动模式 (2)

- 在文档应用中，每次新建文档希望都能新建一个文档实例，重复打开一个已保存的文档希望打开的都是同一个文档实例，文档应用实际演示效果如下图所示：



启动模式的相关配置

- 如果需要使用某一种启动模式，必须在module.json5配置文件的"launchType"字段配置为该模式，例如使用singleton启动模式，则将"launchType"字段配置为"singleton"即可。

```
{
  "module": {
    ...
    "abilities": [
      {
        "launchType": "singleton",
       // "launchType": " multiton ",
       // "launchType": " specified ",
        ...
      }
    ]
  }
}
```

目录

1. Stage应用模型概述

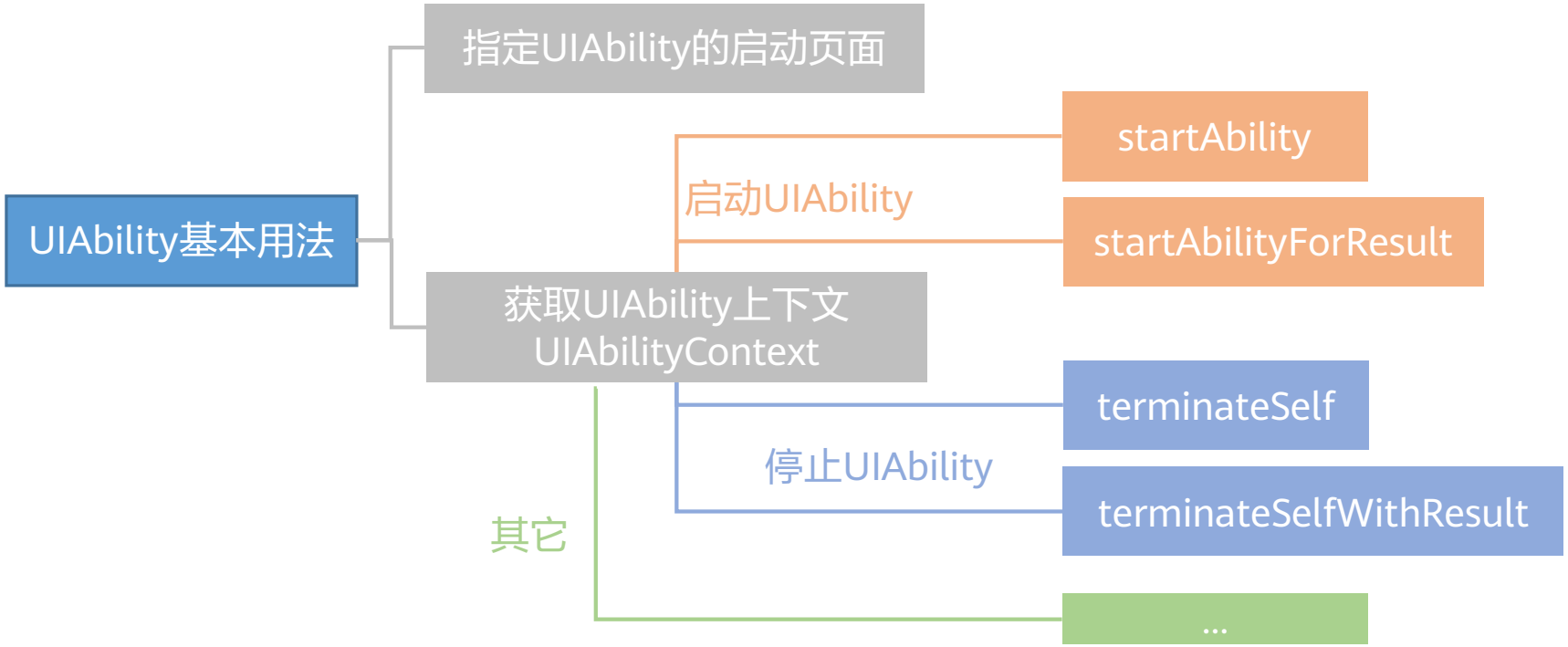
2. **UIAbility应用组件**

- UIAbility组件概述
- UIAbility组件生命周期
- UIAbility组件启动模式
- **UIAbility基本用法**
- UIAbility组件间交互（设备内）

HUAWEI

UIAbility组件的基本用法

- 指定UIAbility的启动页面和获取UIAbility的上下文UIAbilityContext是UIAbility组件的两种基本用法。 通过UIAbilityContext可以获取UIAbility的相关配置信息，提供操作UIAbility实例的方法，比如UIAbility的启动、停止等。



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。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class EntryAbility extends UIAbility {
    onWindowStageCreate(windowStage: Window.WindowStage) {
        // Main window is created, set main page for this ability
        windowStage.loadContent('pages/Index', (err, data) => {
            ...
        });
    } ...
}
```

获取UIAbility的上下文信息 (1)

- UIAbility类拥有自身的上下文信息，通过UIAbilityContext可以获取UIAbility的相关配置信息，如包代码路径、Bundle名称、Ability名称和应用程序需要的环境状态等属性信息，同时还可以获取操作UIAbility实例的方法。
  - 在UIAbility中可以通过this.context获取UIAbility实例的上下文信息。

```
// EntryAbility.ts文件
import UIAbility from '@ohos.app.ability.UIAbility';
export default class EntryAbility extends UIAbility {
    onCreate(want, launchParam) {
        // 获取UIAbility实例的上下文
        let context = this.context;
        ...
    }
}
```

获取UIAbility的上下文信息 (2)

- 在页面文件中获取UIAbility实例的上下文信息，包括导入依赖资源context模块和在组件中定义一个context变量两个部分。

```
//ArlTS的页面文件 xxx.ets, ， 这里是Index.ets
import common from '@ohos.app.ability.common';
@Entry
@Component
struct Index {
  private context = getContext(this) as common.UIAbilityContext;
  startAbilityTest() {
    let want = {
      // Want参数信息
    };
    this.context.startAbility(want);
  }
   build() {// 页面展示
        ...
    }
}
```

获取UIAbility的上下文信息 (3)

- 在导入依赖资源context模块后，定义context变量，还可以在具体使用UIAbilityContext前进行变量定义。

```
//ArlTS的页面文件 xxx.ets，这里是Index.ets
import common from '@ohos.app.ability.common';
@Entry
@Component
struct Index {
  startAbilityTest() {
    let context = getContext(this) as common.UIAbilityContext;
    let want = {
      // Want参数信息
    };
    context.startAbility(want);
  }
   build() { // 页面展示
   ...
   }
}
```

目录

1. Stage应用模型概述

2. **UIAbility应用组件**

   - UIAbility组件概述
   - UIAbility组件生命周期
   - UIAbility组件启动模式
   - UIAbility基本用法
   - **UIAbility组件间交互（设备内）**

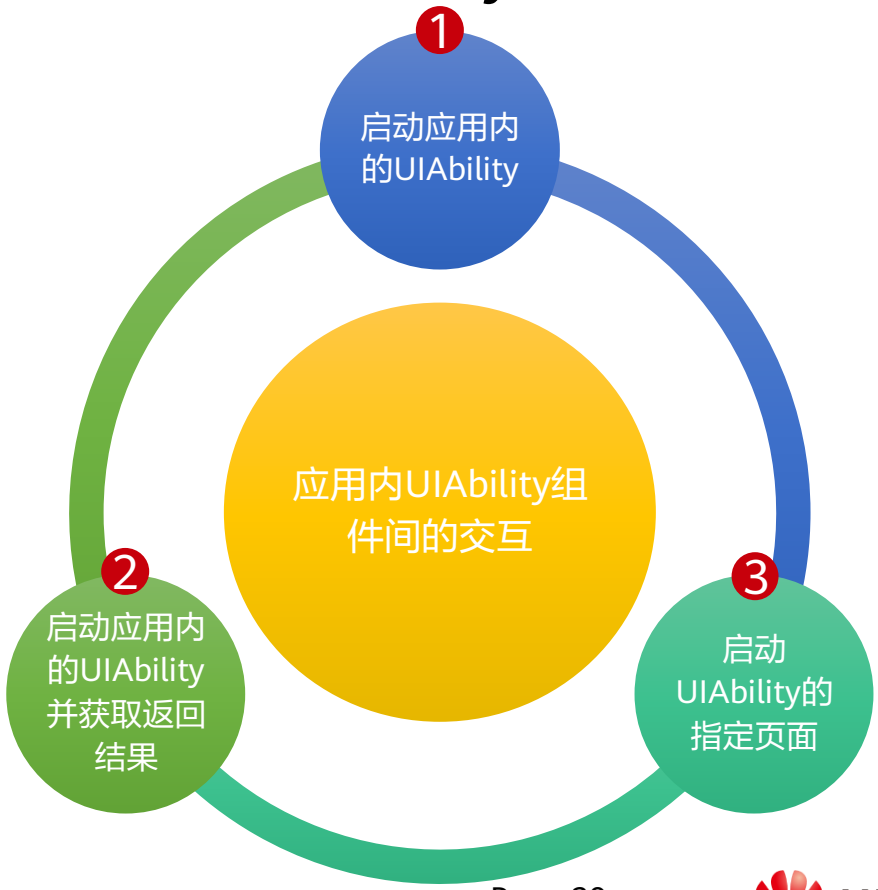
UIAbility组件间交互

- 在UIAbility的使用过程中，当在不同功能模块之间跳转时，常遇到启动特定UIAbility的场景，以电商应用为例，UIAbilityA负责购物车功能模块，UIAbilityB负责订单功能模块。

1. 希望在UIAbilityA中完成商品结算后，自动生成订单。

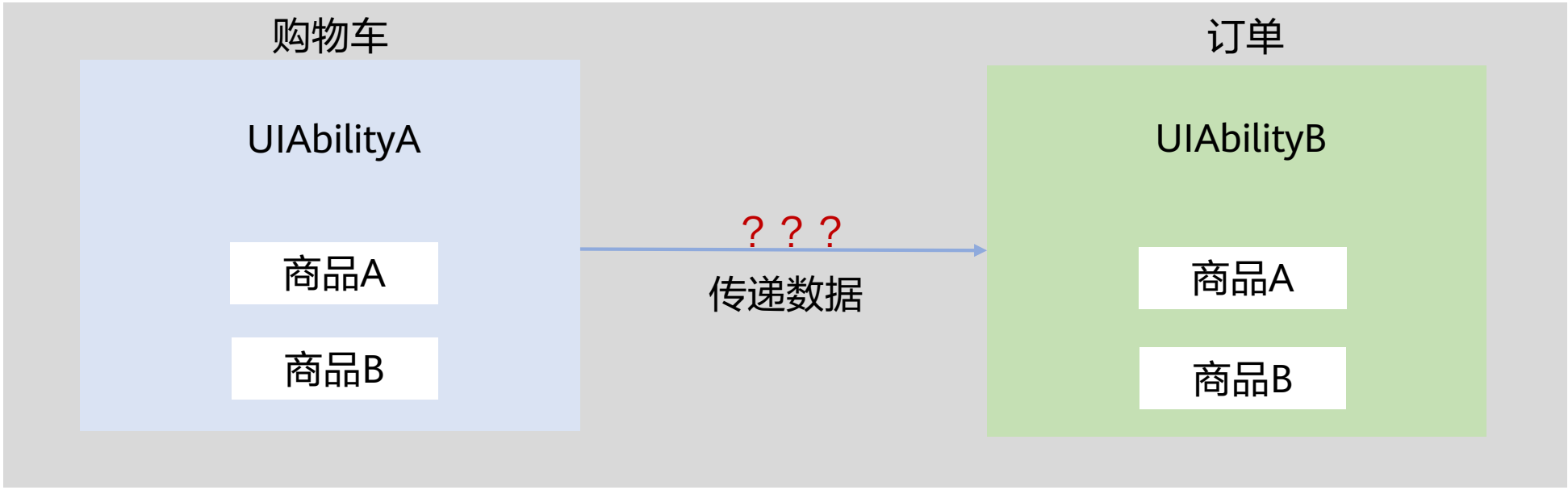
2. 希望在UIAbiltiyA中收到订单生成的提示信息。

3. 希望在UIAbilityA中完成商品结算后，生成订单时是跳转到订单浏览页面，而不是UIAbilityB模块的其它页面（如订单修改页面）。



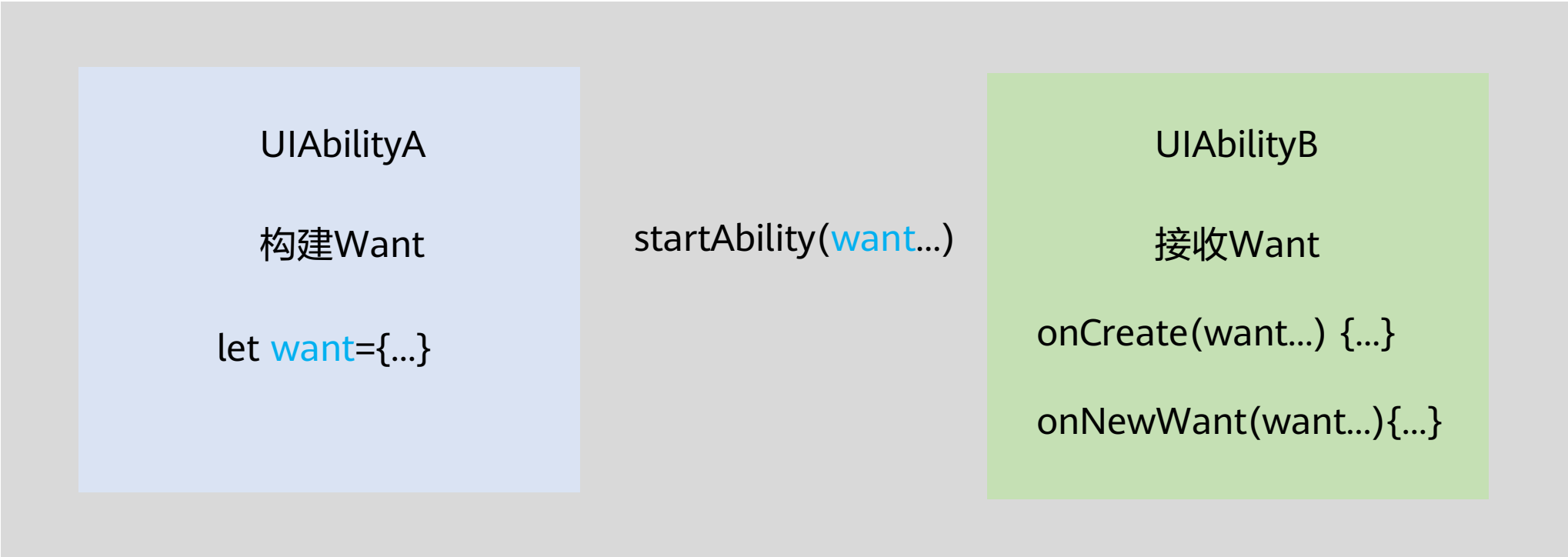
启动UIAbility时的数据传递

- 仍然以电商应用为例，UIAbilityA负责购物车功能模块， UIAbilityB负责订单功能模块。当我们在UIAbilityA中启动UIAbilityB时，就能实现结算商品并自动生成订单的效果。此时会发现生成的订单中会含有购物车结算的商品数据，那么这个数据的传递是如何实现的呢?



Want的定义与用途

- Want是对象间信息传递的载体，可以用于应用组件间的信息传递。当UIAbilityA启动UIAbilityB并需要传入一些数据给UIAbilityB时，Want可以作为一个载体将数据传给UIAbilityB。



Want的类型 - 显式Want

- 显式Want是在启动Ability时指定了abilityName和bundleName的Want，则称为显式Want。
- 当有明确处理请求的对象时，通过提供目标Ability所在应用的包名信息（bundleName），并在Want内指定abilityName便可启动目标Ability。显式Want通常用于在当前应用开发中启动某个已知的Ability。

```
let wantInfo = {
    deviceId: '  ', // deviceId为空表示本设备
    bundleName: 'com.example.myapplication', // 待启动应用的Bundle名称
    abilityName: 'FuncAbility', // 待启动的UIAbility名称
}
```

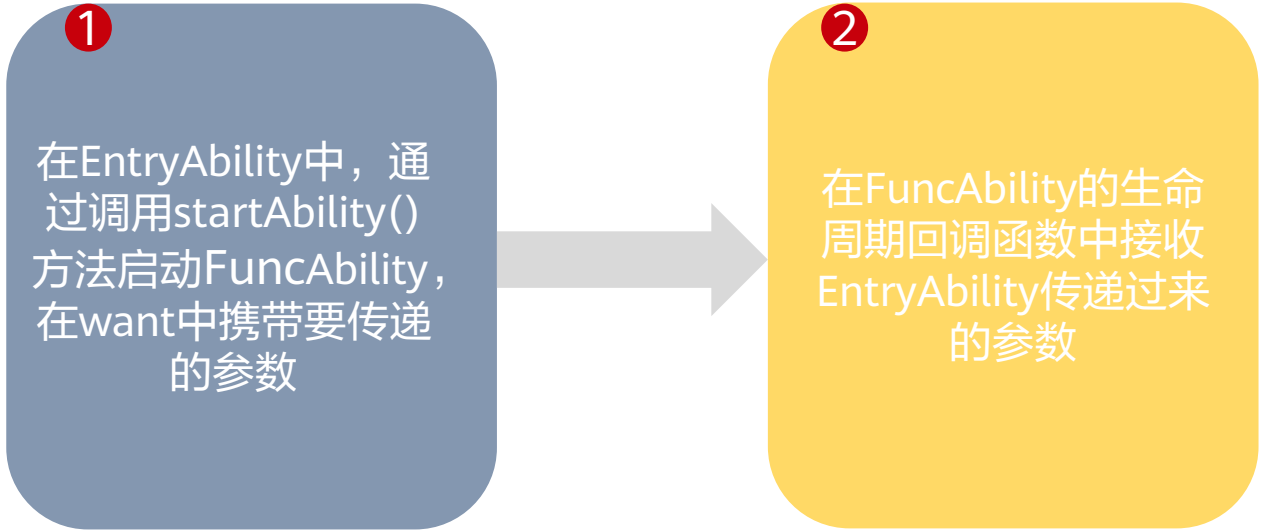
Want的类型 - 隐式Want

- 隐式Want是在启动UIAbility时未指定abilityName的Want，则称为隐式Want。当请求处理的对象不明确时，希望在当前应用中使用其他应用提供的某个能力（通过skills标签定义），而不关心提供该能力的具体应用，可以使用隐式Want。

```
let wantInfo = {
        //表示要执行的通用操作（如：查看、分享、应用详情）
        action: 'ohos.want.action.search',
        //表示目标Ability额外的类别信息,在隐式Want中是对action的补充可省略
        entities: ['entity.system.browsable' ],
        ...
        };
```

启动应用内的UIAbility（1）

- 当一个应用内包含多个UIAbility时，存在应用内启动UIAbility的场景。
  - 例如应用中有两个UIAbility：EntryAbility和FuncAbility，需要从EntryAbility的页面中启动FuncAbility，流程如下：



备注

在FuncAbility业务完成之后，如需要停止当前UIAbility实例

在FuncAbility中通过调用terminateSelf()方法实现

启动应用内的UIAbility (2)

- 在EntryAbility中，通过调用startAbility()方法启动UIAbility。

```
// EntryAbility的ArkTS页面文件，如Index.ets
let wantInfo = {
    deviceId: ' ', // deviceId为空表示本设备
    bundleName: ' com.example.myapplication ', // 待启动应用的Bundle名称

    abilityName: ' FuncAbility ', // 待启动的UIAbility名称
    moduleName: 'module1', // moduleName非必选
    parameters: { // 自定义信息
        info: '来自EntryAbility Index页面',
    },
}
// context为调用方UIAbility的AbilityContext
this.context.startAbility(wantInfo).then(() => {
  ...
}).catch((err) => {
  ...
 }
)
```

启动应用内的UIAbility (3)

- 在FuncAbility.ts文件的生命周期回调函数中接收EntryAbility传递过来的参数。
- 在FuncAbility业务完成之后，如需要停止当前UIAbility实例，在FuncAbility中通过调用terminateSelf()方法实现。

```
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class FuncAbility extends UIAbility {
    onCreate(want, launchParam) {
    // 接收调用方UIAbility传过来的参数
        let funcAbilityWant = want;
        let info = funcAbilityWant?.parameters?.info;
        ...
    }
}
```

```
// context为需要停止的UIAbility实例的AbilityContext
this.context.terminateSelf((err) => {
    ...
});
```

启动应用内的UIAbility (4)

- 在FuncAbility.ts文件的生命周期回调函数中接收到EntryAbility传递过来的参数后，如想要在FuncAbility的页面文件中使用该参数，只需将参数存放在AppStorage中即可。

```
import UIAbility from '@ohos.app.ability.UIAbility';
import Window from '@ohos.window';

export default class FuncAbility extends UIAbility {
    onCreate(want, launchParam) {
    // 接收调用方UIAbility传过来的参数
        let funcAbilityWant = want;
        let info = funcAbilityWant?.parameters?.info;
    // 将参数存放到AppStorage中，方便在页面文件中使用
    AppStorage.SetOrCreate('info', info)
}
}
```

```
// 在页面文件中获取存在AppStorage中的参数
let info= AppStorage.Get<string>('info')
@Entry
@Component
struct Index {
  build() {
      Column{
        Text(info) // 使用参数
  }
}
```

启动应用内的UIAbility并获取返回结果（1）

- 当一个EntryAbility启动另外一个FuncAbility时，希望在被启动的FuncAbility完成相关业务后，能将结果返回给调用方,实现流程如下：

启动应用内的UIAbility并获取返回结果 (2)

- 在EntryAbilit中，调用startAbilityForResult()接口启动FuncAbility，异步回调中的data用于接收FuncAbility停止自身后返回给EntryAbility的信息。

```
// EntryAbility的ArkTS页面文件，如Index.ets
let wantInfo = {
    deviceId: '', // deviceId为空表示本设备
    bundleName: 'com.example.myapplication',
    abilityName: 'FuncAbility',
    moduleName: 'module1', // moduleName非必选
    parameters: { // 自定义信息
        info: '来自EntryAbility Index页面',
    },
}
// context为调用方UIAbility的AbilityContext
this.context.startAbilityForResult(wantInfo).then((data) => {
    ...
}).catch((err) => {
   ...
})
```

启动应用内的UIAbility并获取返回结果 (3)

- 在FuncAbility停止自身时，只需在其页面文件中调用terminateSelfWithResult()方法即可，入参abilityResult为FuncAbility需要返回给EntryAbility的信息。

```
const RESULT_CODE: number = 1001;
let abilityResult = {
    resultCode: RESULT_CODE,
    want: {
        bundleName: 'com.example.myapplication',
        abilityName: ‘EntryAbility',
        ...
            },
        },
}
// context为被调用方UIAbility的AbilityContext
   this.context.terminateSelfWithResult(abilityResult, (err) => {
    ...
});
```

启动应用内的UIAbility并获取返回结果 (4)

- 在FuncAbility停止自身后，EntryAbility通过startAbilityForResult()方法回调接收被FuncAbility返回的信息，RESULT_CODE需要与前面的数值保持一致。

```
const RESULT_CODE: number = 1001;
// context为调用方UIAbility的AbilityContext
this.context.startAbilityForResult(want).then((data)
=> {
    if (data?.resultCode === RESULT_CODE) {
        // 解析被调用方UIAbility返回的信息
        let info = data.want?.parameters?.info;
        ...
    }
}).catch((err) => {
        ...
})
```

启动UIAbility的指定页面（1）

- 一个UIAbility可以对应多个页面，在不同的场景下启动该UIAbility时需要展示不同的页面，例如从一个UIAbility的页面中跳转到另外一个UIAbility时，希望启动目标UIAbility的指定页面。

启动UIAbility的指定页面 (2)

- 调用方UIAbility启动另外一个UIAbility时，通常需要跳转到指定的页面。如FuncAbility包含两个页面（Index对应首页，Second对应功能A页面），此时需要在传入的want参数中配置指定的页面路径信息，可以通过want中的parameters参数增加一个自定义参数传递页面跳转信息。

```
let wantInfo = {
    deviceId: '', // deviceId为空表示本设备
    bundleName: 'com.example.myapplication',
    abilityName: 'FuncAbility',
    parameters: { // 自定义参数传递页面信息
        router: 'funcA',
    },
}
  // context为调用方UIAbility的AbilityContext
    this.context.startAbility(wantInfo).then(() => {
      ...
   }).catch((err) => {
      ...
})
```

启动UIAbility的指定页面 (3)

- 目标UIAbility首次启动时，在目标UIAbility的onWindowStageCreate()生命周期回调中，解析EntryAbility传递过来的want参数，获取到需要加载的页面信息url，传入windowStage.loadContent()方法。

```
import UIAbility from '@ohos.app.ability.UIAbility'
import Window from '@ohos.window'
export default class FuncAbility extends UIAbility {
onCreate(want, launchParam) {
    let funcAbilityWant = want; // 接收调用方UIAbility传过来的参数
   }
onWindowStageCreate(windowStage: Window.WindowStage) {
    let url = 'pages/Index';
      if (this.funcAbilityWant?.parameters?.router) {
          if (this.funcAbilityWant.parameters.router === 'funcA') {
              url = 'pages/Second';
          }}
              windowStage.loadContent(url, (err, data) => { ...
           });
    }
}
```

本章小结

- 本章主要介绍了Stage模型与UIAbility应用组件的核心概念。
- 讲解了UIAbility的生命周期与启动模式。
- 讲解了UIAbility应用组件的基本用法。
- 讲解了常见场景下组件间的交互（设备内）。

思考题

1. （单选题）在UIAbility的生命周期回调函数中，哪个回调在UIAbility实例创建时完成触发？（ ）

A. onCreate()

B. onForeground()

C. onBackground()

D. onDestroy()

思考题

2. （单选题） UIAbility如果想使用多实例启动模式，则需要在module.json5配置文件中将“launchType”字段配置为？（ ）

A. singleton

B. " "

C. multiton

D. specified

学习推荐

- 官方学习网站
  - HarmonyOS官网：https://developer.harmonyos.com/
  - HarmonyOS应用开发文档： https://developer.huawei.com/consumer/cn/
  - OpenHarmony官网：https://edu.huaweicloud.com/
  - 华为开发者论坛： https://developer.huawei.com/consumer/cn/forum/

华为云开发者学堂

感谢