HarmonyOS应用开发基础

HUAWEI

前言

- 本章为HarmonyOS应用开发基础课程，简述HarmonyOS应用基础开发技术栈，并通过对分布式关键技术和应用场景的分析，为后续应用开发实战打下基础。

目标

- 了解HarmonyOS的理念及技术特性
- 掌握HarmonyOS应用基础开发
- HarmonyOS应用开发项目实践

目录

HarmonyOS概述

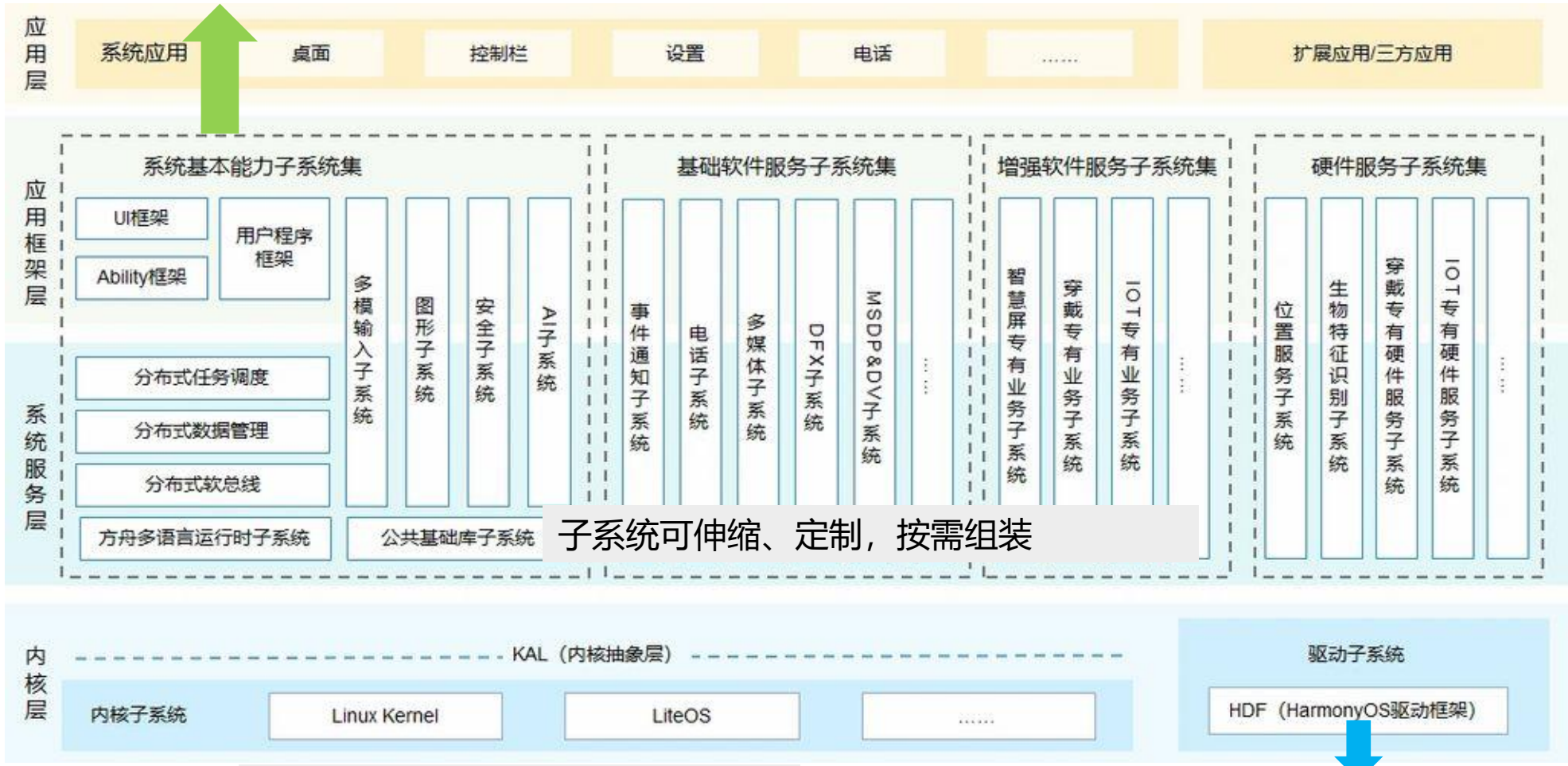
- HarmonyOS是一款“面向未来”、面向全场景（移动办公、运动健康、社交通信、媒体娱乐等）的分布式操作系统。
- 消费者：实现不同的终端设备之间的快速连接、能力互助、资源共享，提供流畅的全场景体验；
- 应用开发：提供多种分布式能力，可以快速将业务迁移到其他设备上；
- 设备开发：组件化的设计，根据设备资源和业务特征进行灵活裁剪，满足不同终端设备的要求；

系统特点：

- **1、硬件互助，资源共享：**分布式和设备虚拟化
- **2、一次开发，多端部署：**业务逻辑和界面逻辑进行复用
- **3、统一OS，弹性部署：**弹性部署，适配不同类别的硬件和功能

HarmonyOS架构

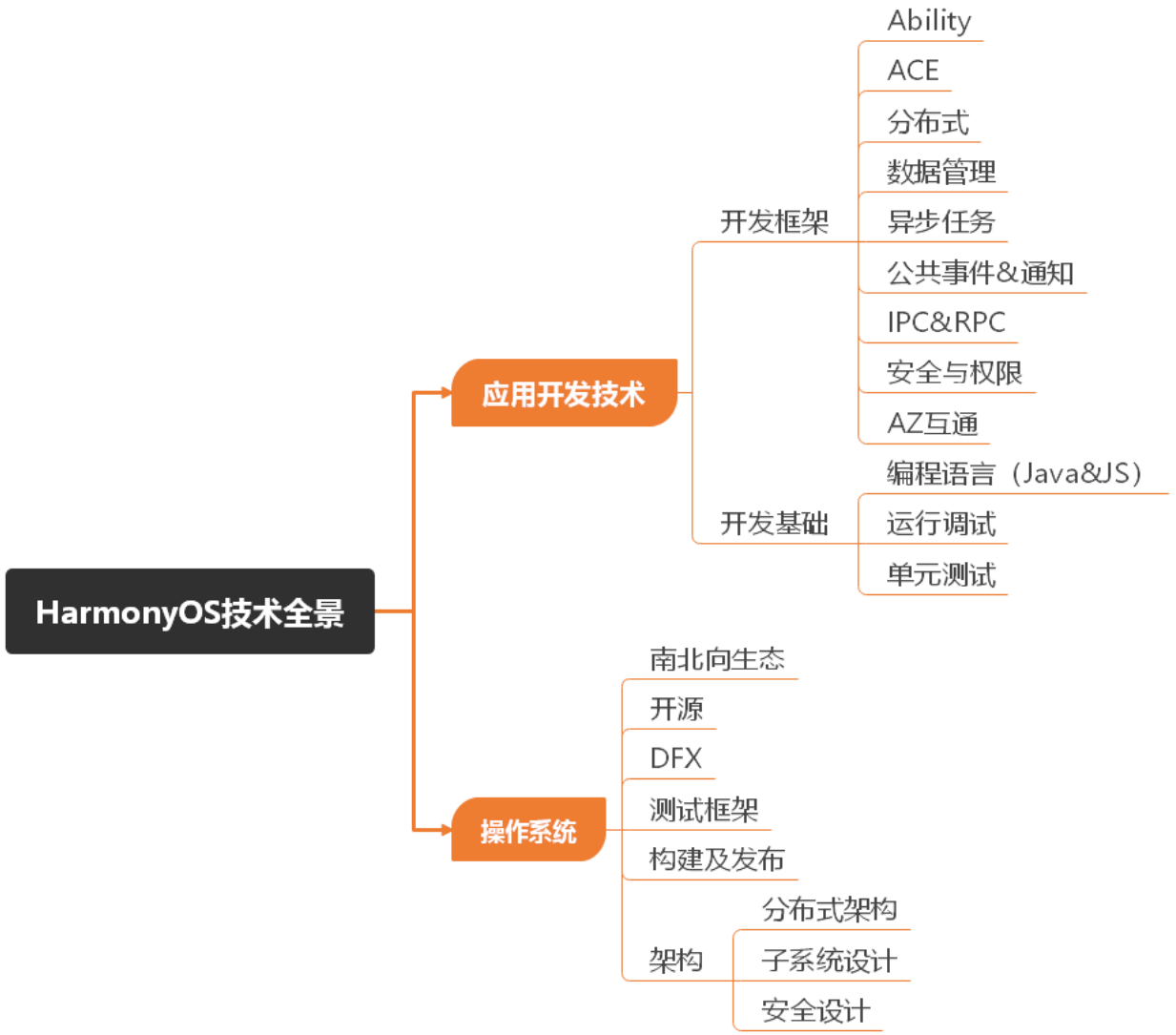
北向应用生态



支持多内核，可部署不同等级设备

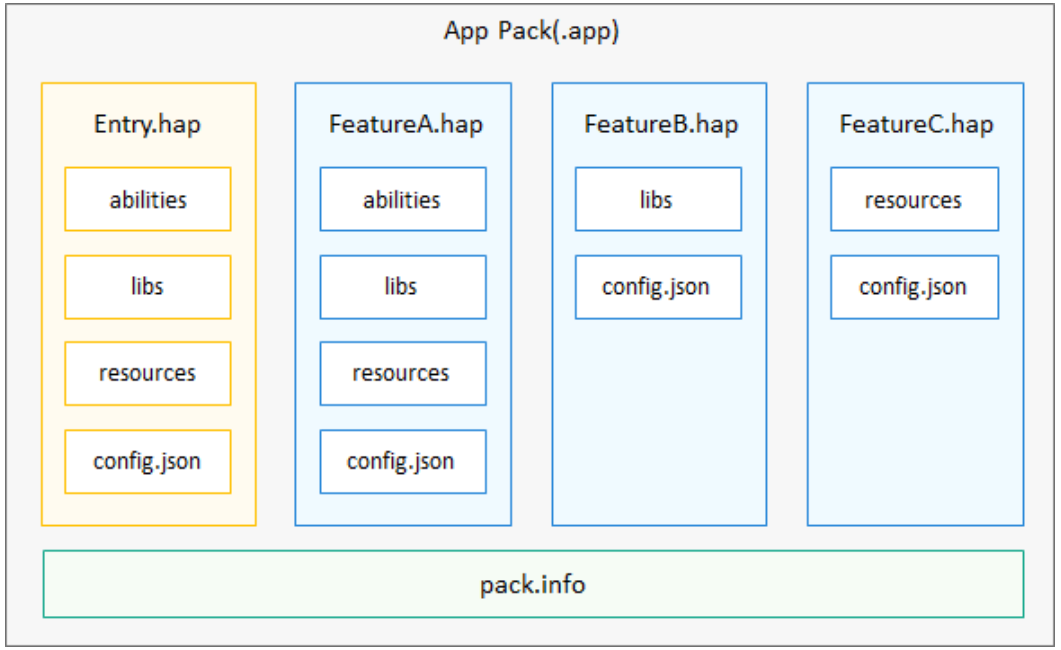
南向设备生态

应用开发基础 - 技术栈全景



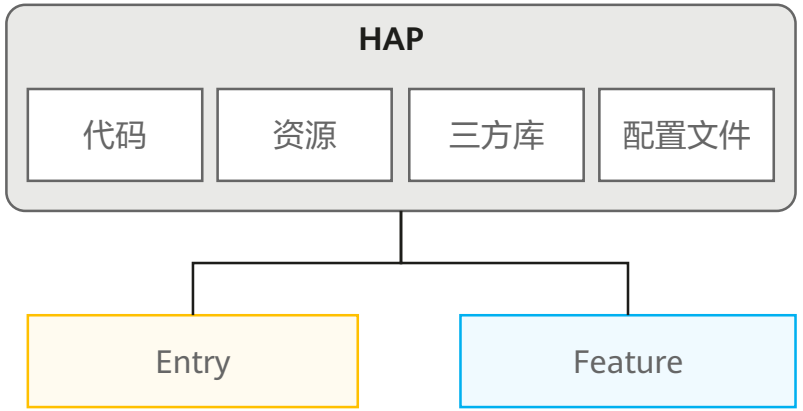
应用开发基础 - APP文件结构

HarmonyOS的应用软件包以**APP Pack**（Application Package）形式发布，它是由一个或多个**HAP**（HarmonyOS Ability Package）以及描述每个HAP属性的pack.info组成。HAP是Ability的部署包，HarmonyOS应用代码围绕Ability组件展开。



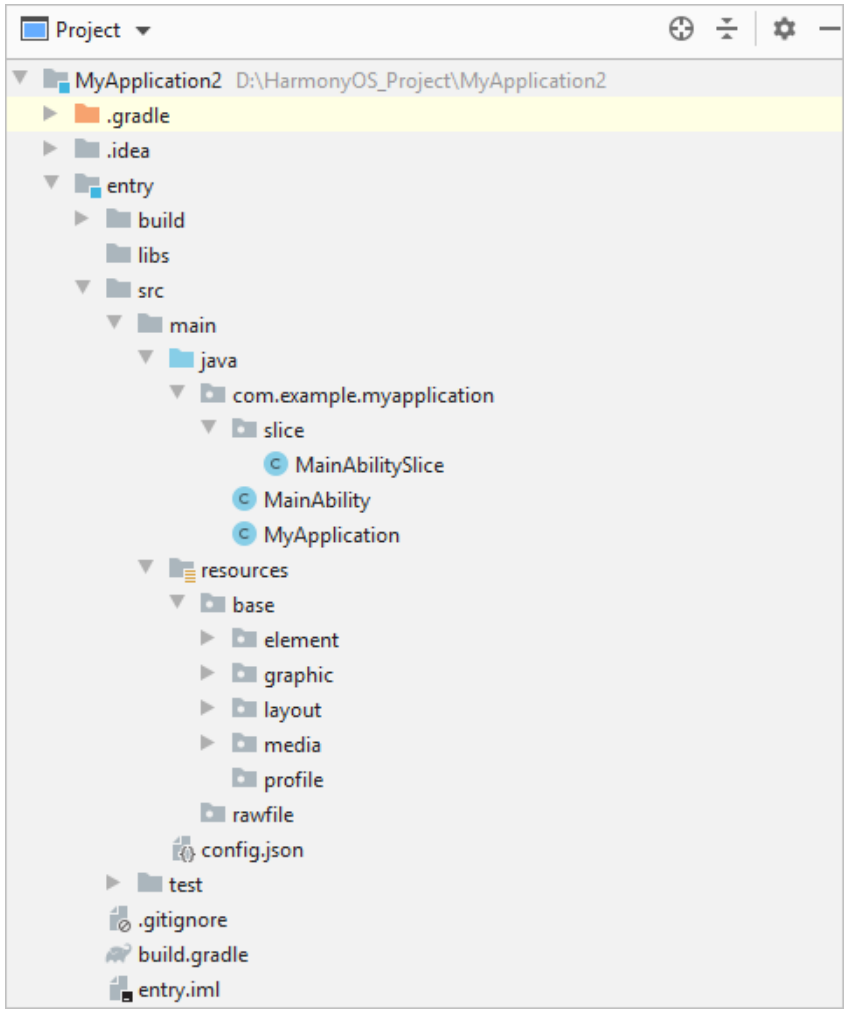
App逻辑视图

一个HAP是由代码、资源、第三方库及应用配置文件组成的模块包，可分为entry和feature两种模块类型。



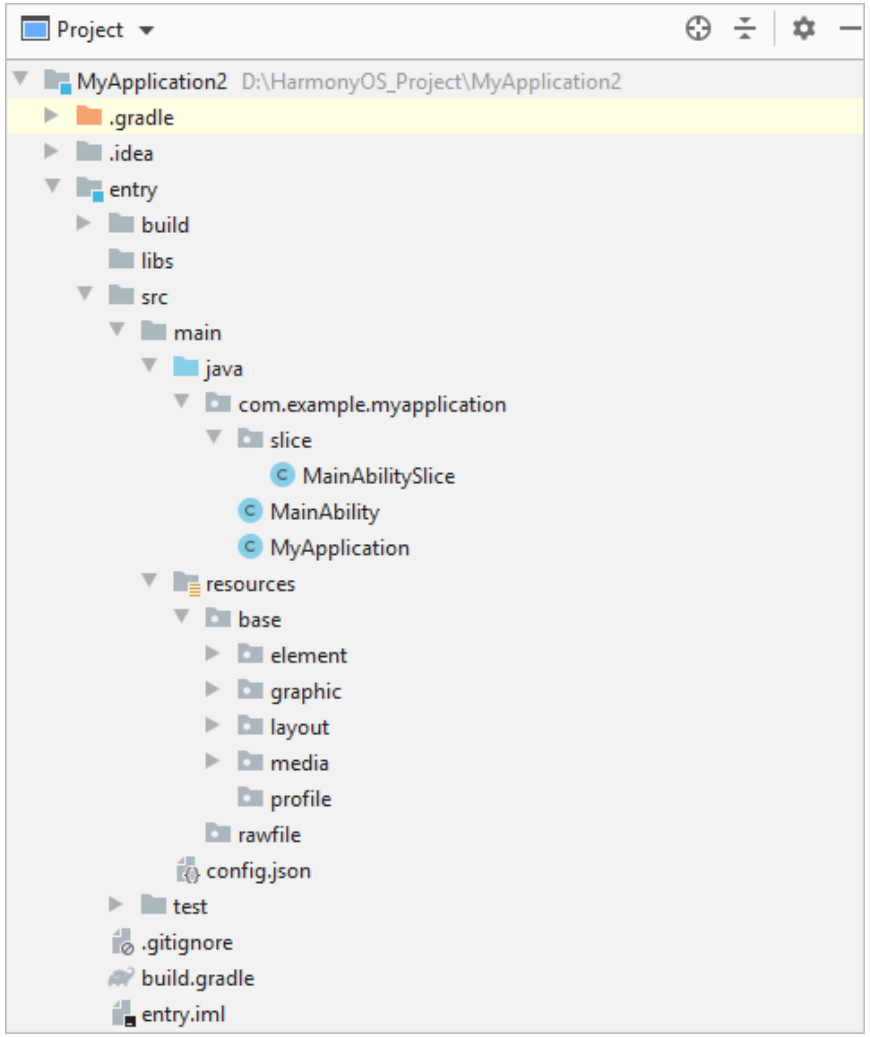
- Entry：应用的主模块。一个APP中，对于同一设备类型必须有且只有一个entry类型的HAP，可独立安装运行。
- Feature：应用的动态特性模块。一个APP可以包含一个或多个feature类型的HAP，也可以不含。只有包含Ability的HAP才能够独立运行。

应用开发基础 - Java工程结构 (1)



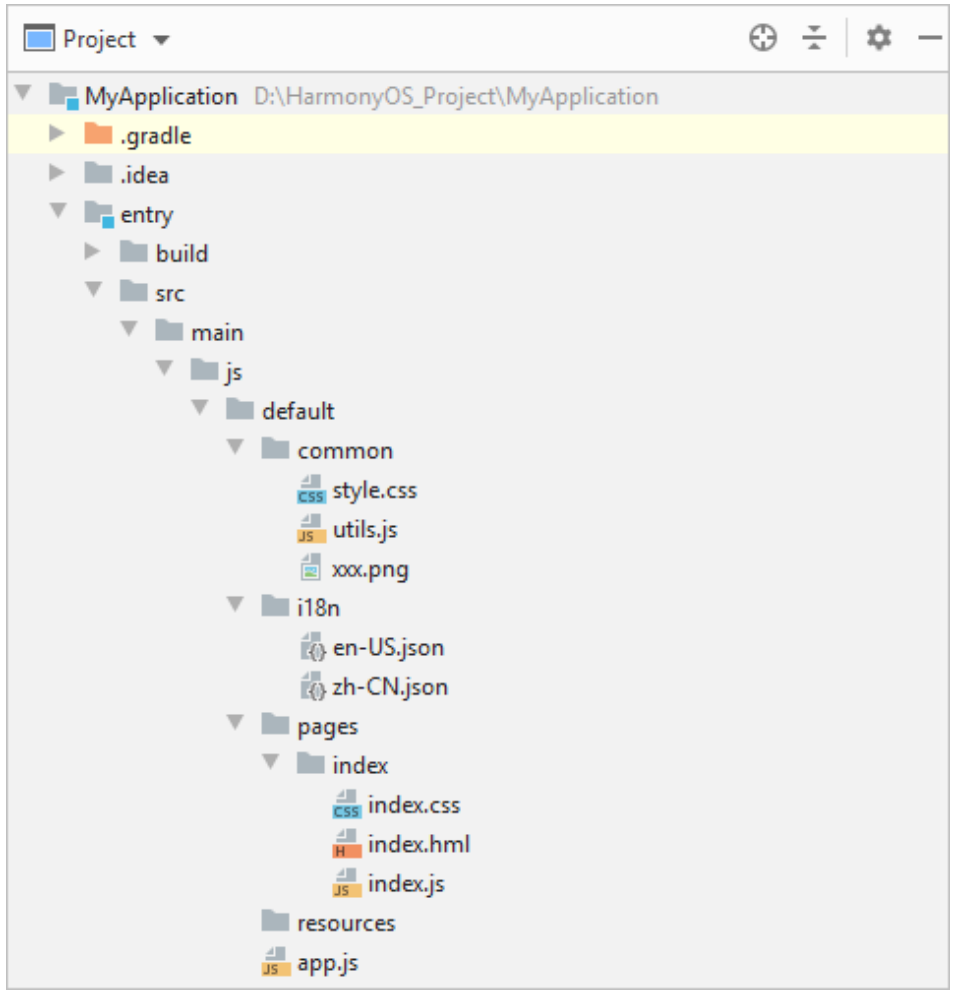
- **.gradle**：Gradle配置文件，由系统自动生成，一般情况下不需要进行修改。
- **entry**：默认启动模块（主模块），开发者用于编写源码文件以及开发资源文件的目录。
  - **entry>libs**：用于存放entry模块的依赖文件。
  - **entry>src>main>Java**：用于存放Java源码。
  - **entry>src>main>resources**：用于存放应用所用到的资源文件，如图形、多媒体、字符串、布局文件等。关于资源文件的详细说明请参考资源文件的分类。
  - **entry>src>main>config.json**：HAP清单文件，详细说明请参考**config.json配置文件介绍**。
  - **entry>src>test**：编写代码单元测试代码的目录，运行在本地Java虚拟机（JVM）上。
  - **entry>.gitignore**：标识git版本管理需要忽略的文件。
  - **entry>build.gradle**：entry模块的编译配置文件。

应用开发基础 - Java工程结构 (2)



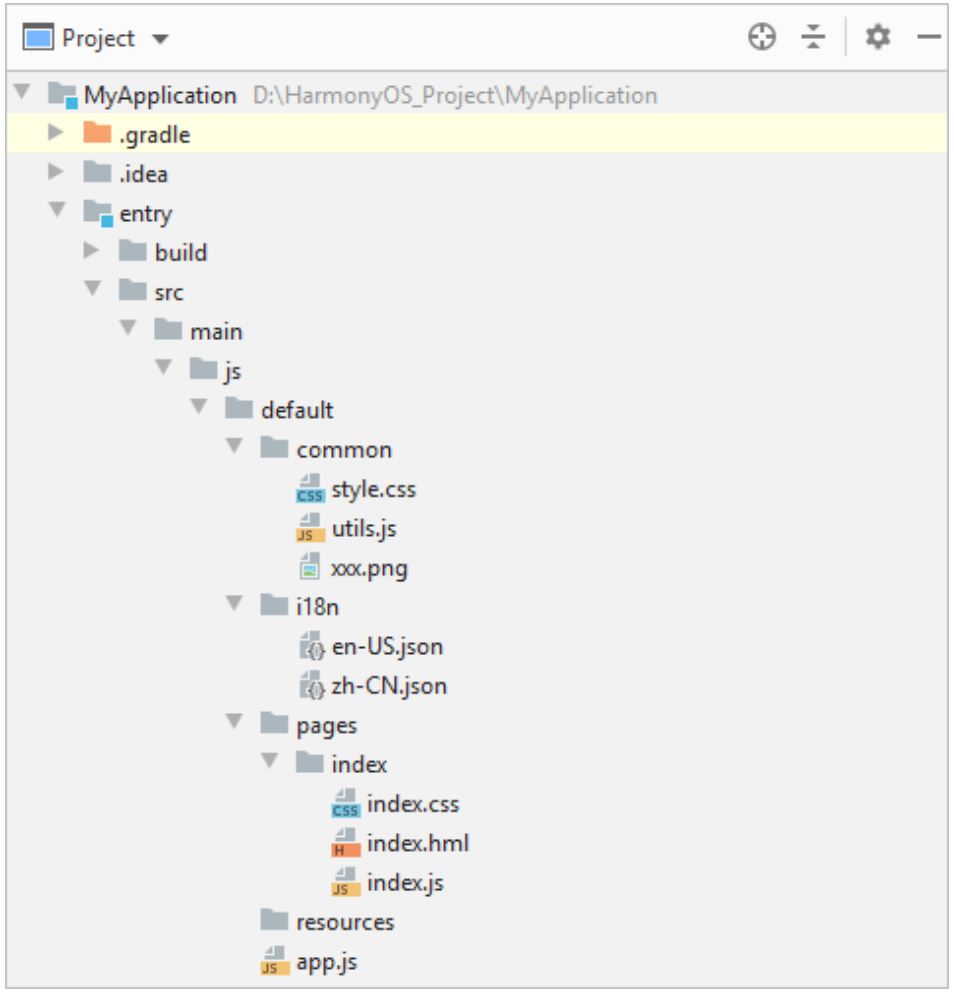
资源目录	资源文件说明
base>element	包括字符串、整型数、颜色、样式等资源的json文件。每个资源均由json格式进行定义，例如： •boolean.json：布尔型 •color.json：颜色 •float.json：浮点型 •intarray.json：整型数组 •integer.json：整型 •pattern.json：样式 •plural.json：复数形式 •strarray.json：字符串数组 •strings.json：字符串值。
base>graphic	xml类型的可绘制资源，如SVG（Scalable Vector Graphics）可缩放矢量图形文件、Shape基本的几何图形（如矩形、圆形、线等）等。
base>layout	xml格式的界面布局文件。
base>media	多媒体文件，如图形、视频、音频等文件，支持的文件格式包括：**.png**、**.gif**、**.mp3**、**.mp4**等。
base>profile	用于存储任意格式的原始资源文件。区别在于rawfile不会根据设备的状态去匹配不同的资源，需要指定文件路径和文件名进行引用。
rawfile	

应用开发基础 - JS工程结构 (1)



- 目录结构中文件分类如下：
  - .hml结尾的HML模板文件，这个文件用来描述当前页面的文件布局结构。
  - .css结尾的CSS样式文件，这个文件用于描述页面样式。
  - .js结尾的JS文件，这个文件用于处理页面和用户的交互。
- 各个文件夹的作用：
  - app.js文件用于全局JavaScript逻辑和应用生命周期管理。
  - pages目录用于存放所有组件页面。
  - common目录用于存放公共资源文件，比如：媒体资源和JS文件。
  - i18n目录用于配置不同语言场景资源内容，比如应用文本词条，图片路径等资源。

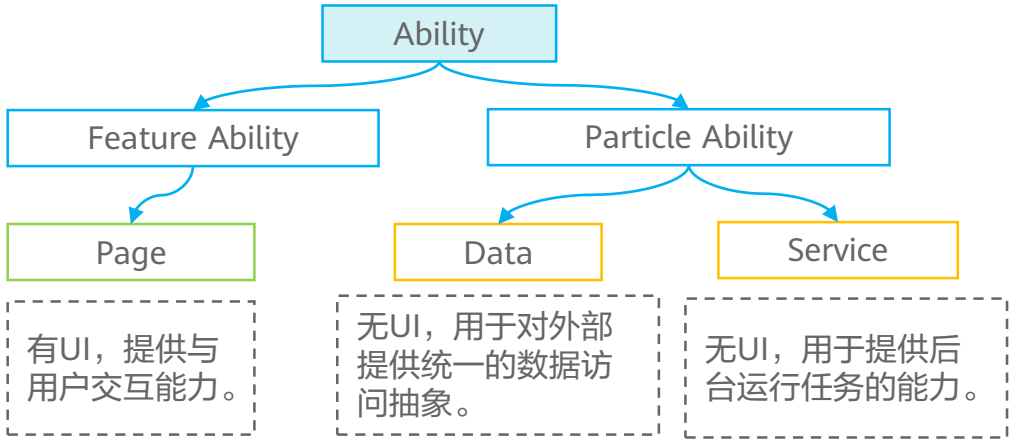
应用开发基础 - JS工程结构 (2)

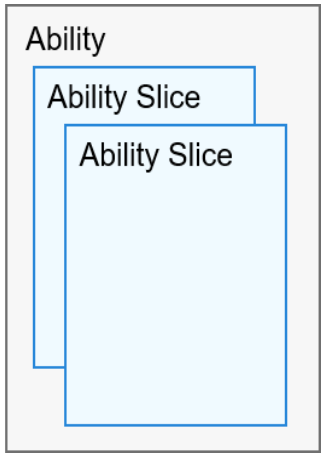


- common目录：可选，用于存放公共资源文件，如媒体资源、自定义组件和JS文档等。
- i18n目录：可选，用于存放多语言的json文件，可以在该目录下定义应用在不同语言系统下显示的内容，如应用文本词条、图片路径等，详情请参考多语言支持。
- pages目录：pages文件夹下可以包含1个或多个页面，每个页面都需要创建一个文件夹（如图中的index）。页面文件夹下主要包含3种文件类型：css、js和hml文件。
- pages>index>index.hml文件：hml文件定义了页面的布局结构，使用到的组件，以及这些组件的层级关系，详情请参考HML语法参考。
- pages>index>index.css文件：css文件定义了页面的样式与布局，包含样式选择器和各种样式属性等，详情请参考CSS语法参考。
- pages>index>index.js文件：js文件描述了页面的行为逻辑，此文件里定义了页面里所用到的所有的逻辑关系，比如数据、事件等，详情请参考JS语法参考。
- resources：可选，用于存放资源配置文件，比如：全局样式、多分辨率加载等配置文件。resources资源引用示例请参考根据设备分辨率加载图片。
- app.js文件：全局的JavaScript逻辑文件和应用的生命周期管理。

应用开发基础 - Ability能力

- Ability是应用所具备的能力的抽象，一个应用可以包含一个或多个Ability。Ability分为两种类型： FA（Feature Ability）和PA（Particle Ability）。
- FA/PA是应用的基本组成单元，能够实现特定的业务功能。
- FA有界面，而PA无界面。





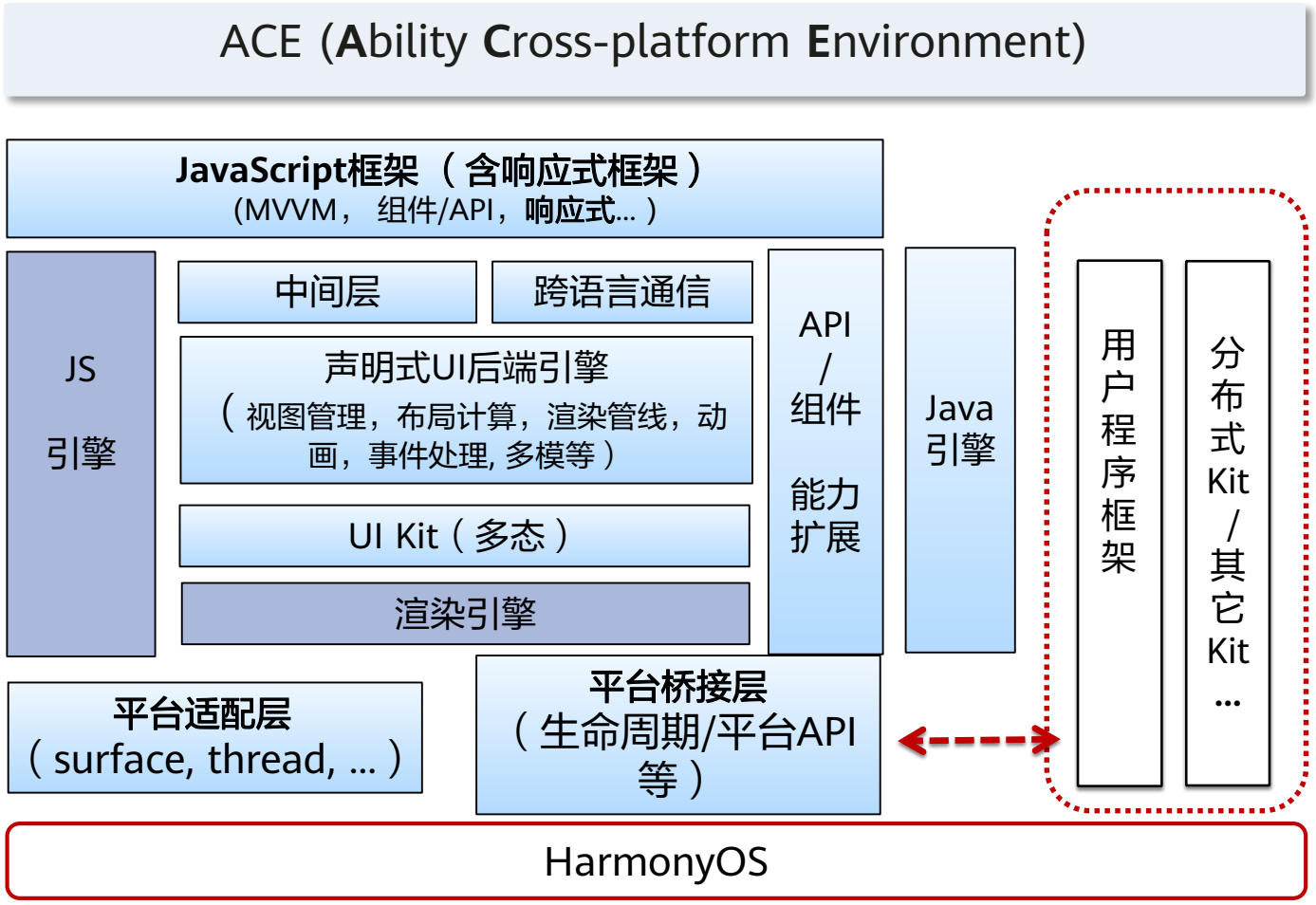
Page Ability又可以由一个或多个 AbilitySlice构成

config.json

```
"abilities": [
  {
    "name": "com.example.myapplication.MainAbility",
    "type": "page",
    "launchType": "standard"
  },
  {
    "name": "com.example.myapplication.DemoService",
    "type": "service"
  },
  {
    "name": "com.example.myapplication.DemoData",
    "type": "data “,
    "uri": "com.example.myapplication.Demo"
  },
  ...
]
```

以模板的形式提供给开发者使用，通过配置属性来区分不同的模板：“type”： page、data、service

应用开发基础 - UI框架



ACE是一个应用开发框架，在OS架构上属于上层框架，目前支持JAVA/JS语言开发；

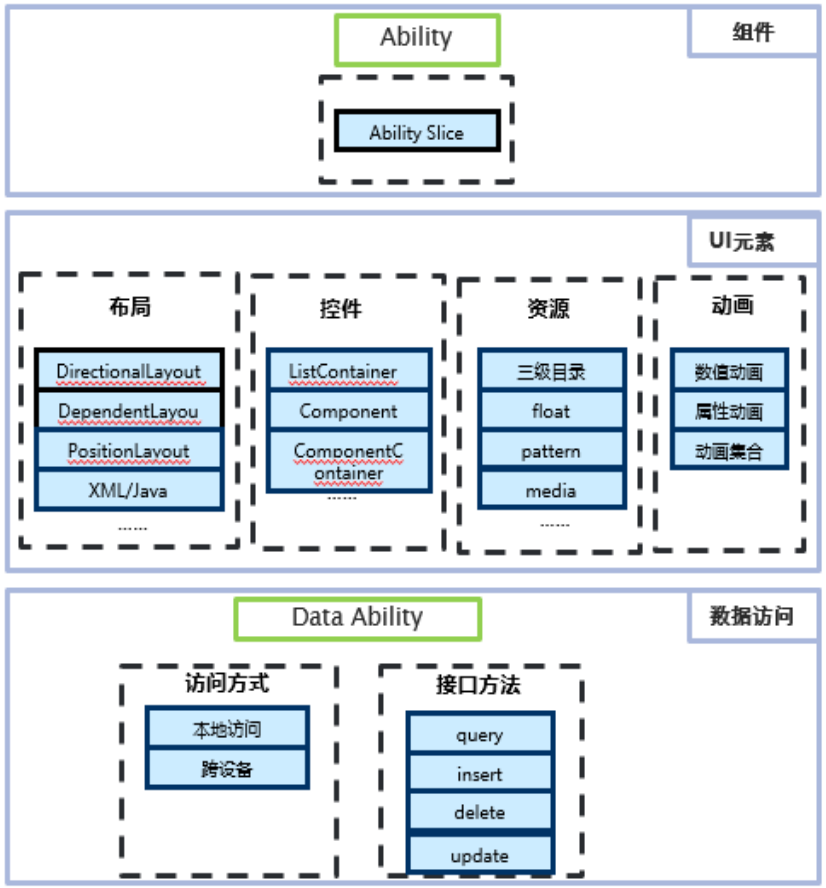
1、JS UI框架采用类HTML和CSS声明式编程语言作为页面布局和页面样式的开发语言，页面业务逻辑则支持ECMAScript 规范的JavaScript语言。

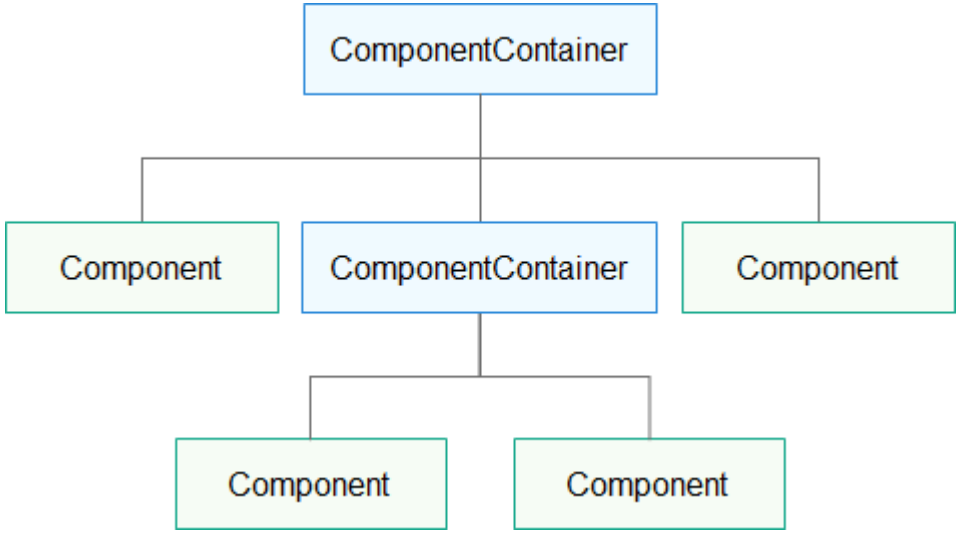
2、JAVA UI是以组件、布局的形式将界面绘制在窗口上。

应用开发基础 - Java UI

应用以Ability为维度，在屏幕上显示一个用户界面。

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。

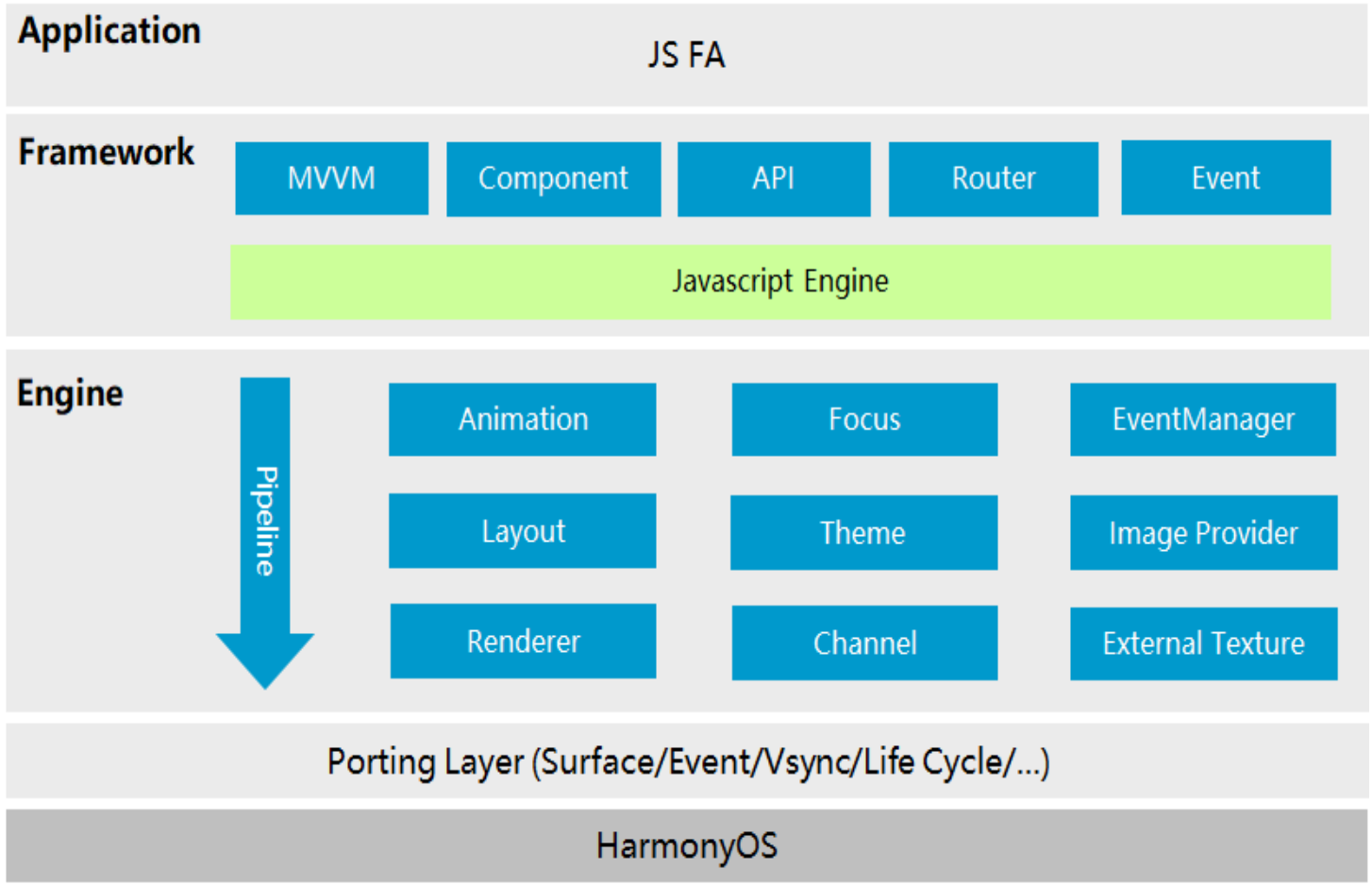




- Component：提供内容显示，是界面中所有组件的基类，开发者可以给Component设置事件处理回调来创建一个可交互的组件。Java UI框架提供了一些常用的界面元素，也可称之为组件，组件一般直接继承Component或它的子类，如Text、Image等。
- ComponentContainer：作为容器容纳Component或ComponentContainer对象，并对它们进行布局。Java UI框架提供了一些标准布局功能的容器，它们继承自ComponentContainer，一般以“Layout”结尾，如DirectionalLayout、DependentLayout等。

应用开发基础 - JS UI (1)

- JS UI框架包括前端框架层（Framework）、引擎层（Engine）和平台适配层（Porting Layer）。
- 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、事件管理和平台channel机制等能力。
- Porting Layer:适配层主要完成对平台层进行抽象，提供具体抽象接口，可以对接到不同系统平台。比如：事件对接、渲染管线对接和系统生命周期对接等。



应用开发基础 - JS UI (2)

index.hml

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

    <button class="btn_common"
            type="capsule"
            onclick="changeText">
        Click Me
    </button>
</div>
```

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

index.css

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

.title {
    font-size: 100px;
}

.btn_common {
    width: 50%;
    height: 100px;
    margin-top: 30px;
    font-size: 40px;
}
```

应用开发基础 - 并发 (1)

✓TaskDispatcher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；支持设置任务的优先级。

分类	功能	约束
GlobalTaskDispatcher	全局并发任务分发器	全局唯一，应用内只有一个；适用于任务之间没有联系的情况。
ParallelTaskDispatcher	并发任务分发器	非全局唯一性，应用可以创建多个
SerialTaskDispatcher	串行任务分发器	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
SpecTaskDispatcher	专有任务分发器	目前已有的专有线程是主线程。UITaskDispatcher和MainTaskDispatcher 都属于SpecTaskDispatcher。

任务派发：

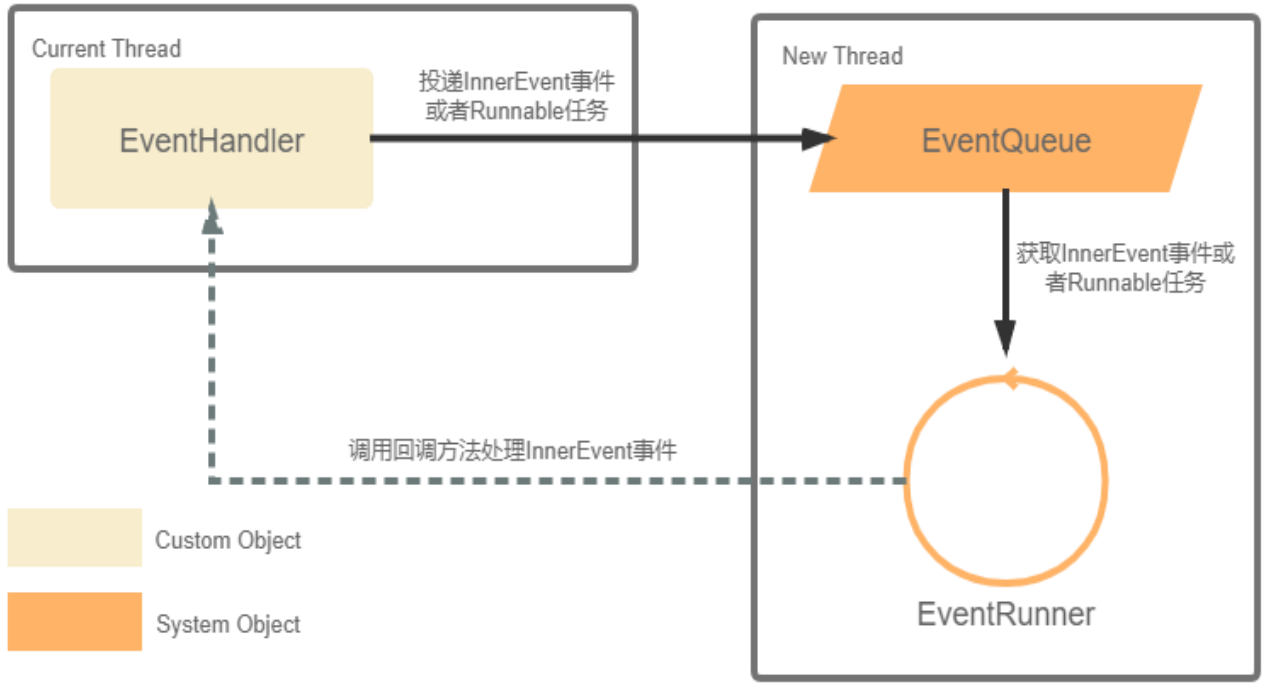
1、syncDispatch：派发任务并在当前线程等待任务执行完成。在返回前，当前线程会被阻塞。

2、asyncDispatch：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。

应用开发基础 - 并发 (2)

✓ EventHandler

用于处理线程间通信的一种机制，主要与EventRunner和InnerEvent两者进行配合使用来达到线程间通信的目的。



【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：

1）取出的事件为InnerEvent，则触发EventHandler的回调方法，在新线程上处理该事件。

2）取出的事件为Runnable任务，则EventRunner直接在新线程上处理Runnable任务。

应用开发基础 - 公共事件与通知 (1)

✓ CES

为应用程序提供订阅、发布、退订公共事件的能力

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

2、CommonEventData：封装公共事件相关信息。

3、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。

事件分类：有序/无序的公共事件、携带权限的公共事件、粘性公共事件

发布：publishCommonEvent

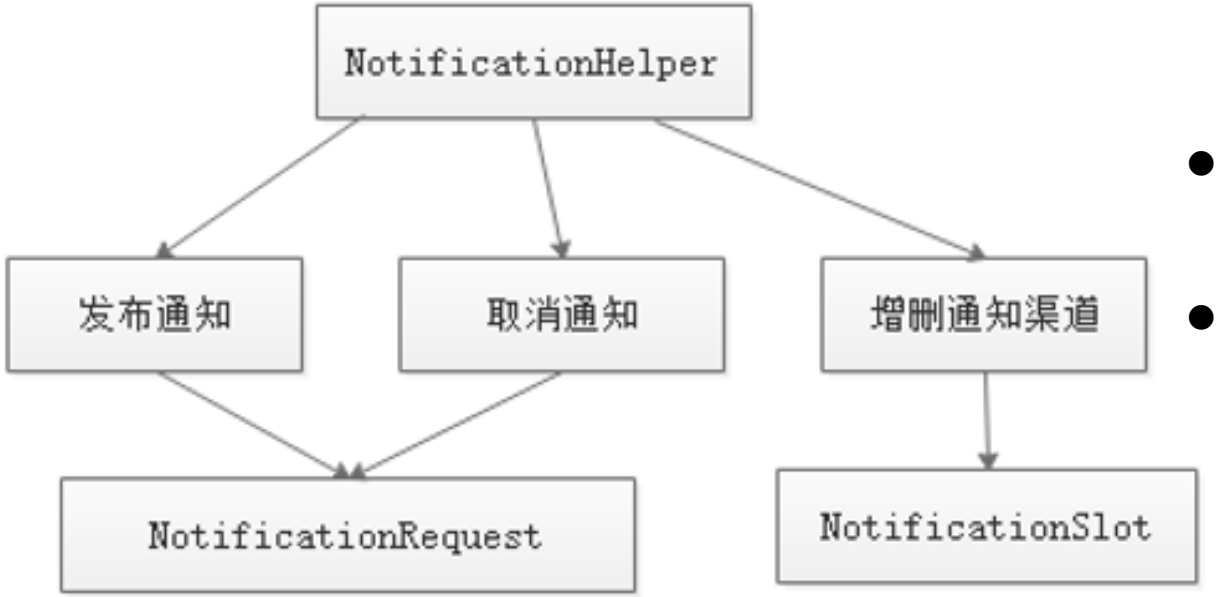
订阅：subscribeCommonEvent

退订：unsubscribeCommonEvent

应用开发基础 - 公共事件与通知 (2)

✓ANS（Advanced Notification Service）

为应用程序提供订阅、发布、退订通知的能力；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。



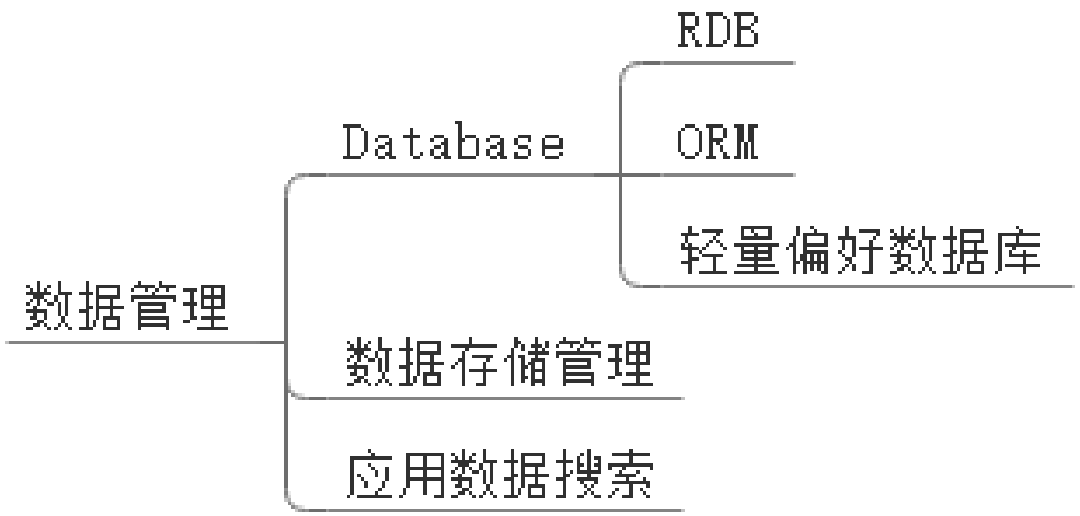
- 发送通知

  NotificationHelper.publishNotification(request);

- 取消通知

  NotificationHelper.cancelNotification(notification_id);

应用开发基础 - 数据管理



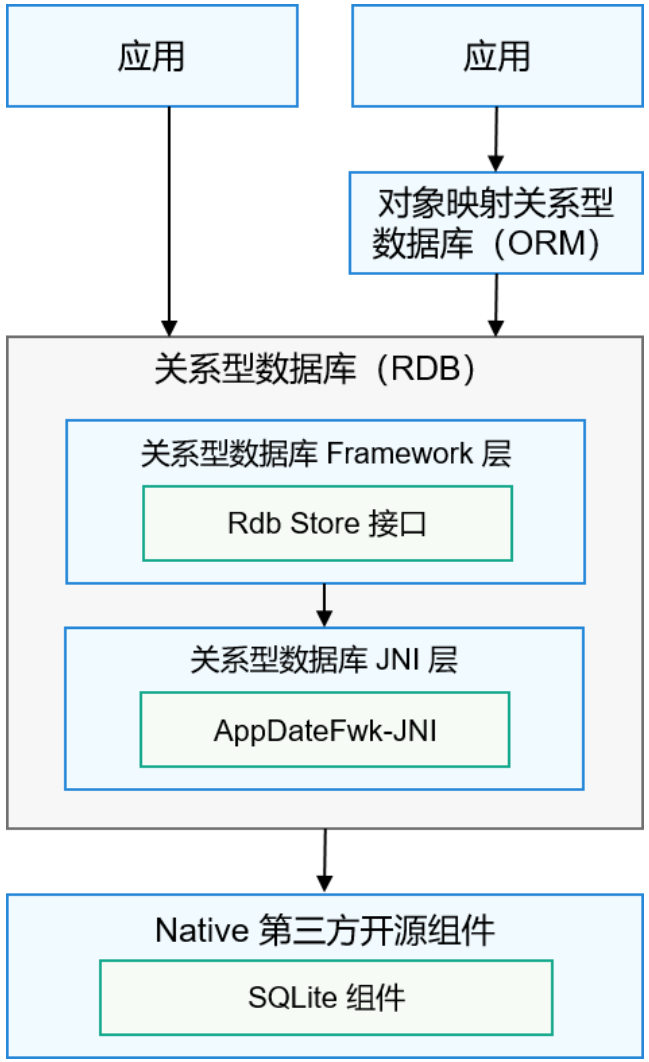
RDB：关系型数据库管理

ORM：对象映射数据管理，操作对象一样执行数据库操作；

轻量偏好数据库：KV模型，保存简单的持久化数据

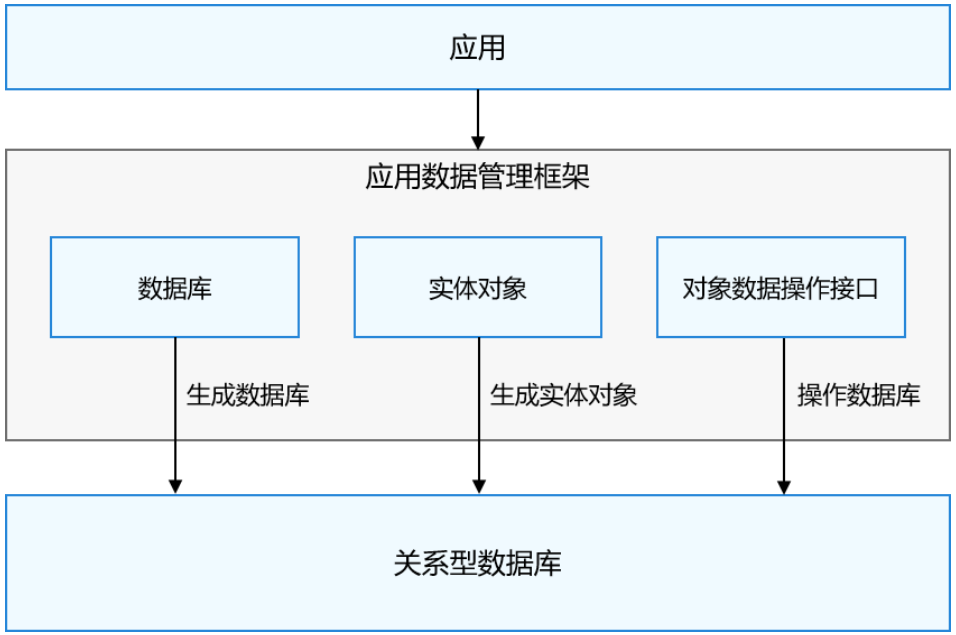
应用开发基础 - 关系型数据库

- HarmonyOS关系型数据库对外提供通用的操作接口，底层使用SQLite作为持久化存储引擎，支持SQLite具有的所有数据库特性，包括但不限于事务、索引、视图、触发器、外键、参数化查询和预编译SQL语句。
- 基本概念
  - 关系型数据库创建在关系模型基础上的数据库，以行和列的形式存储数据。
  - 谓词数据库中用来代表数据实体的性质、特征或者数据实体之间关系的词项，主要用来定义数据库的操作条件。
  - 结果集指用户查询之后的结果集合，可以对数据进行访问。结果集提供了灵活的数据访问方式，可以更方便的拿到用户想要的数据。
  - SQLite数据库一款轻型的数据库，是遵守ACID的关系型数据库管理系统。它是一个开源的项目。



应用开发基础 – ORM (1)

- HarmonyOS对象关系映射（Object Relational Mapping，ORM）数据库是一款基于SQLite的数据库框架，屏蔽了底层SQLite数据库的SQL操作，针对实体和关系提供了增删改查等一系列的面向对象接口。应用开发者不必再去编写复杂的SQL语句， 以操作对象的形式来操作数据库，提升效率的同时也能聚焦于业务开发。

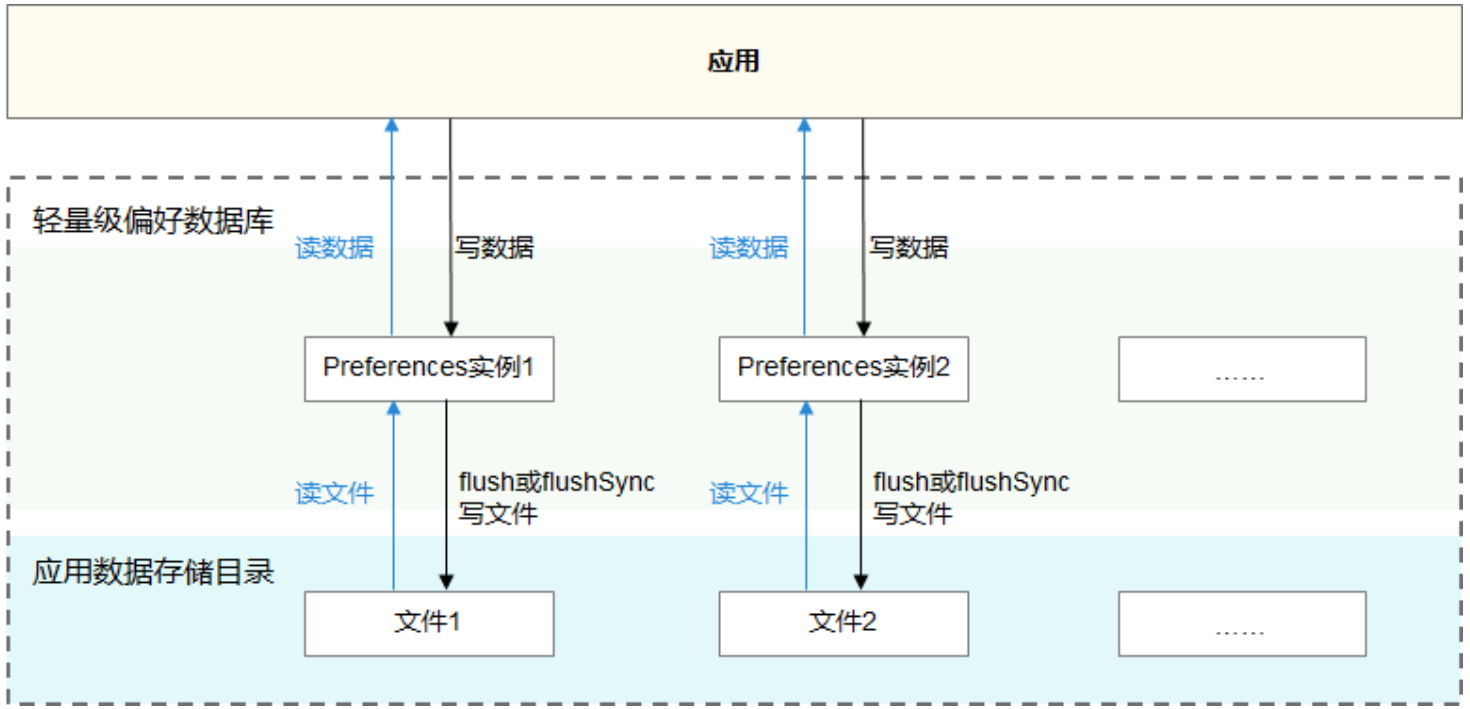


应用开发基础 – ORM (2)

- 对象关系映射数据库的三个主要组件：
  - 数据库：被开发者用@Database注解，且继承了OrmDatabase的类，对应关系型数据库。
  - 实体对象：被开发者用@Entity注解，且继承了OrmObject的类，对应关系型数据库中的表。
  - 对象数据操作接口：包括数据库操作的入口OrmContext类和谓词接口（ OrmPredicate ）等。
- 谓词 数据库中是用来代表数据实体的性质、特征或者数据实体之间关系的词项，主要用来定义数据库的操作条件。对象关系映射数据库将SQLite数据库中的谓词封装成了接口方法供开发者调用。开发者通过对象数据操作接口，可以访问到应用持久化的关系型数据。
- 对象关系映射数据库 通过将实例对象映射到关系上，实现使用操作实例对象的语法，来操作关系型数据库。它是在SQLite数据库的基础上提供的一个抽象层。
- SQLite数据库 一款轻型的数据库，是遵守ACID的关系型数据库管理系统。

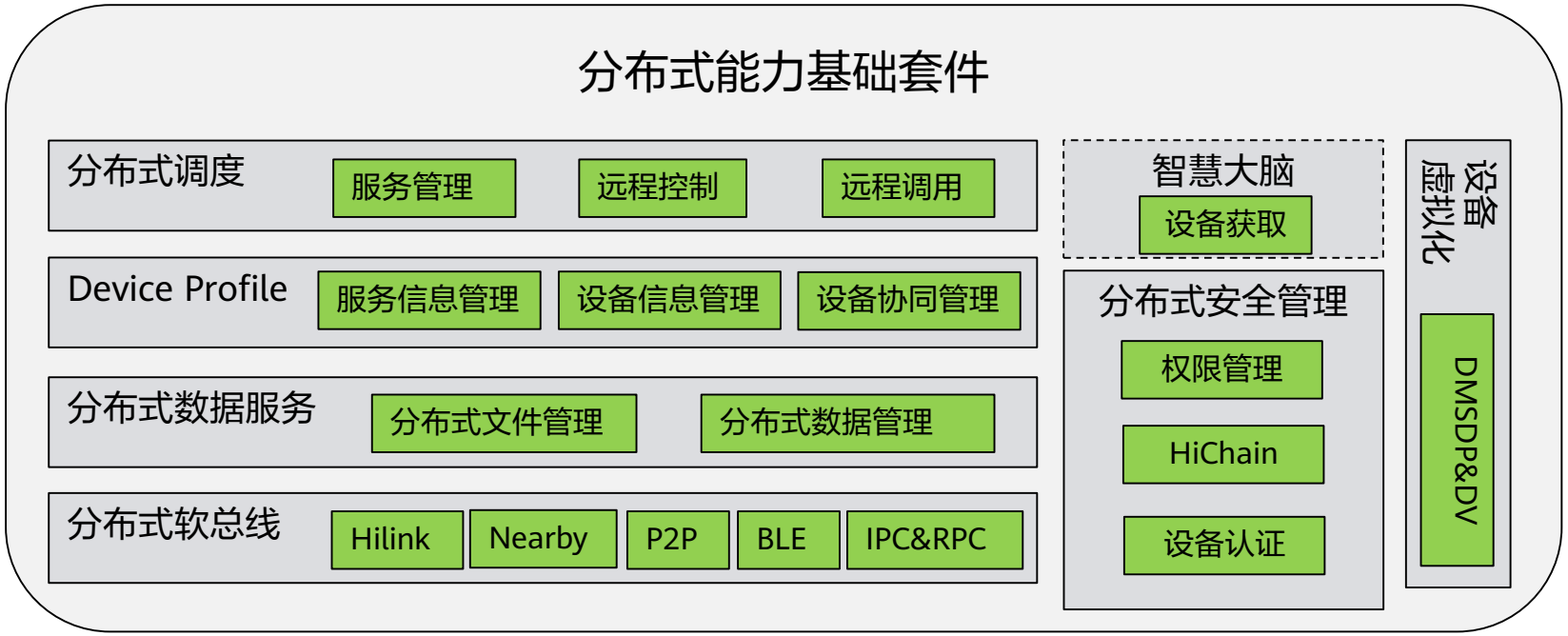
应用开发基础 - 轻量级偏好数据库

- 轻量级偏好数据库主要提供轻量级Key-Value操作，支持本地应用存储少量数据，数据存储在本地文件中，同时也加载在内存中的，所以访问速度更快，效率更高。轻量级偏好数据库属于非关系型数据库，不宜存储大量数据，经常用于操作键值对形式数据的场景。



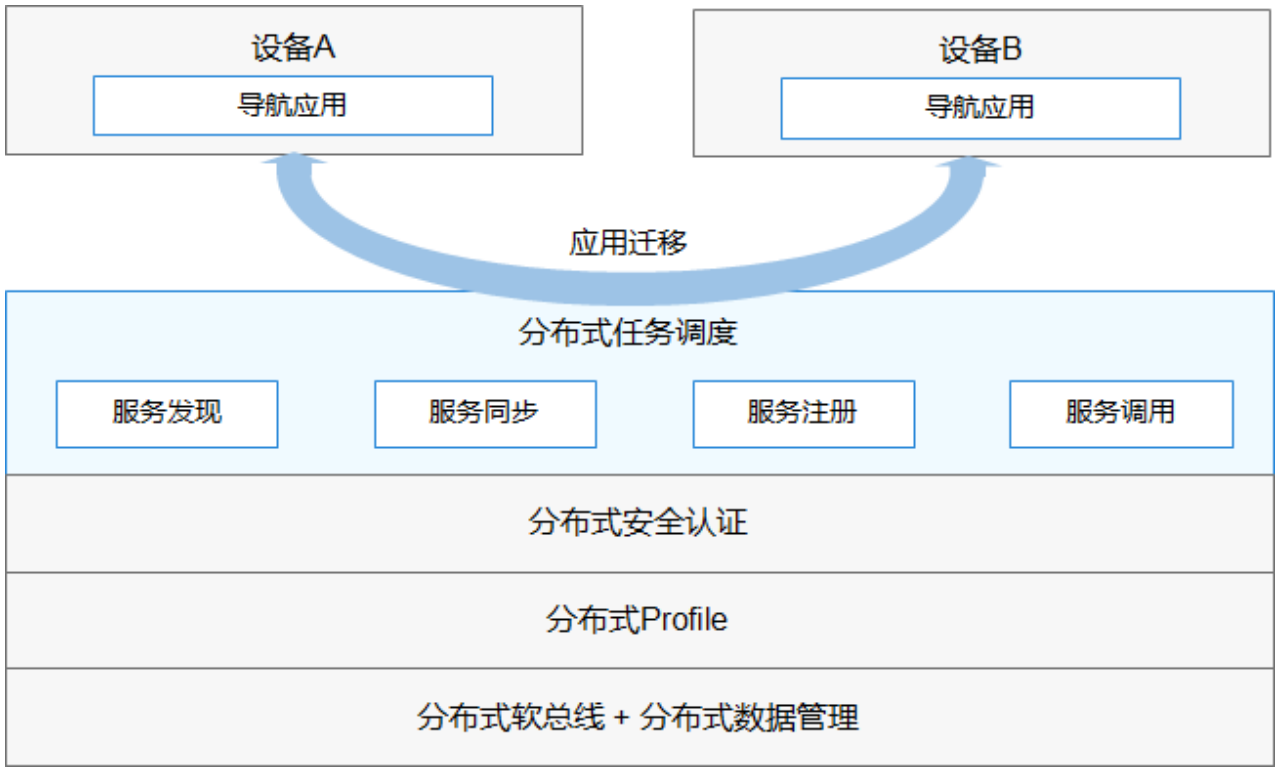
分布式关键技术及应用场景 - 分布式能力

软总线：分布式统一基座，提供统一的分布式通信能力，能够快速发现并连接设备，高效地分发任务和传输数据。



- 组网：
  - 同账号：同华为账号设备自发现、组网；
  - 异账号：支持通过扫码、NFC等方式异账号组网；
- 安全：
  - HiChain 负责设备的安全认证；
  - 总线传输通道加密；
  - 应用负责数据多设备间的安全可见；
- 简单易用：
  - 屏蔽底层通信协议；应用只处理自己关心的业务；

分布式关键技术及应用场景 - 分布式任务调度



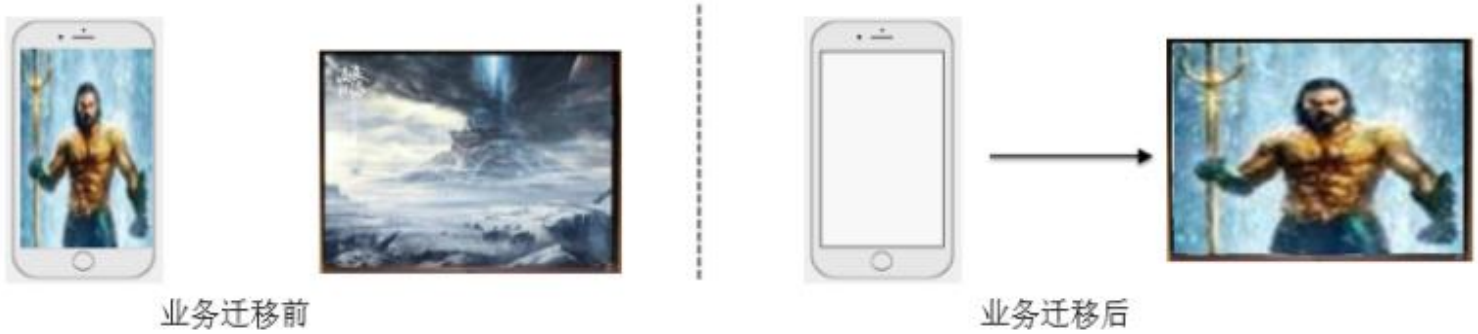
- 支持对跨设备的应用进行远程启动、远程调用、远程连接以及迁移等操作
  - 跨设备拉起用户界面
  - 跨设备访问服务
  - 跨设备访问数据
- 调用方法：
  - startAbility
  - connectAbility

分布式关键技术及应用场景 - 跨设备拉起场景

手表拉起手机畅连登录界面

```
Intent intent = new Intent();
// 指定目标设备deviceId
// 指定待启动FA的bundleName和abilityName
// 设置分布式标记，表明当前涉及分布式能力
Operation operation = new Intent.OperationBuilder()
.withDeviceId(deviceId).withBundleName(bundleName)
.withAbilityName(abilityName)
.withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE) .build();
intent.setOperation(operation);
// 通过AbilitySlice包含的startAbility接口实现跨设备启动FA
startAbility(intent);
```

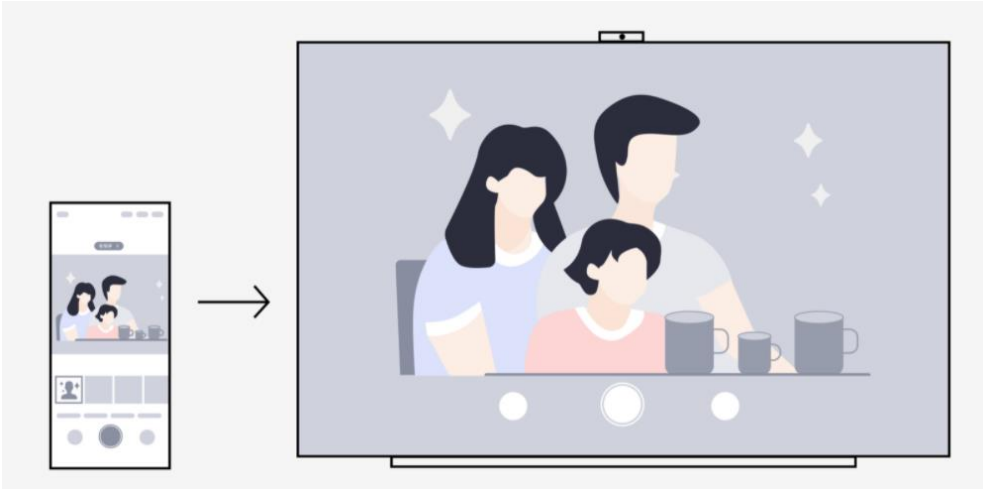
分布式关键技术及应用场景 - 跨设备迁移场景



视频接续，业务无缝迁移

```
try {
    continueAbility();
} catch (IllegalStateException e) {
// Maybe another continuation in progress.
...
}
```

分布式关键技术及应用场景 - 跨设备协同服务场景

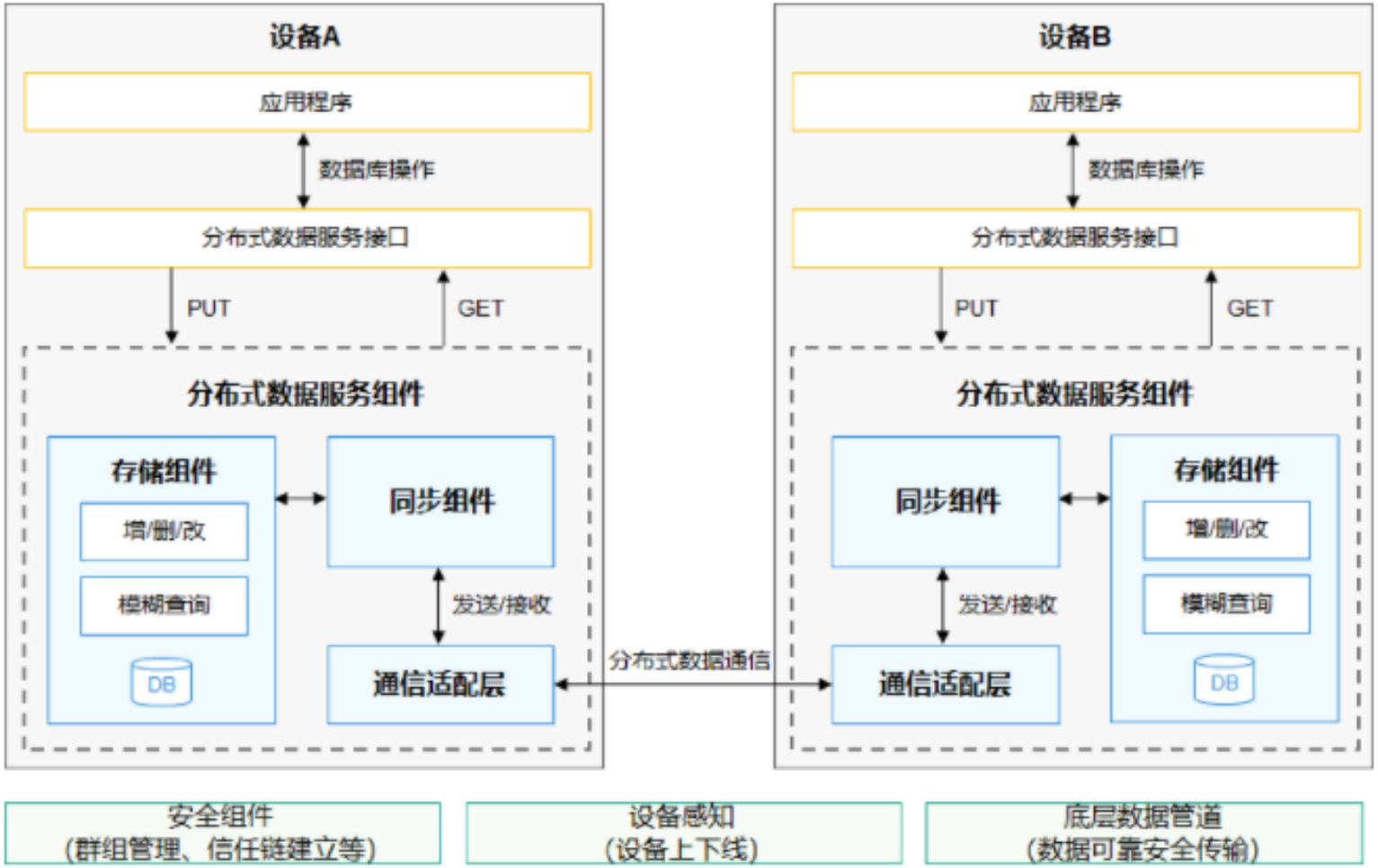


硬件互助，跨设备协同

```
Intent connectPaIntent = new Intent();
Operation operation = new Intent.OperationBuilder()
        .withDeviceId(deviceId)
        .withBundleName(bundleName)
        .withAbilityName(abilityName)
        .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
        .build();
connectPaIntent.setOperation(operation);
conn = new IAbilityConnection() {
    @Override
    public void onAbilityConnectDone(ElementName elementName, IRemoteObject
remote, int resultCode) {
        LogUtils.info(TAG, "===connectRemoteAbility done");
        proxy = new MyRemoteProxy(remote);
    }

    @Override
    public void onAbilityDisconnectDone(ElementName elementName, int resultCode)
{
        LogUtils.info(TAG, "onAbilityDisconnectDone.....");
        proxy = null;
    }
};
context.connectAbility(connectPaIntent, conn);
```

分布式关键技术及应用场景 - 分布式数据服务



分布式数据服务(Distributed Data Service)提供设备间分布式数据管理能力。

分布式关键技术及应用场景 - 分布式数据服务

```
// 创建KvManager
KvManagerFactory kvManagerFactory = KvManagerFactory.getInstance();
KvManagerConfig config = new KvManagerConfig(context);
KvManager mKvManager = kvManagerFactory.createKvManager(config);
// 设置数据库属性
Options options = new Options();
options.setCreateIfMissing(true).setEncrypt(false).setBackup(false).
setAutoSync(true).setKvStoreType(KvStoreType.SINGLE_VERSION);
// 获取SingleKvStore分布式数据库
SingleKvStore mSingleKvStore = mKvManager.getKvStore(options, STORE_NAME);
// 监听数据变化
KvStoreObserverClient kvStoreObserverClient = new KvStoreObserverClient();
mSingleKvStore.subscribe(SubscribeType.SUBSCRIBE_TYPE_REMOTE, kvStoreObserverClient);
// 数据变化回调处理
private static class KvStoreObserverClient implements KvStoreObserver {
    @Override
    public void onChange(ChangeNotification changeNotification) {// 处理变化数据
    }}
```

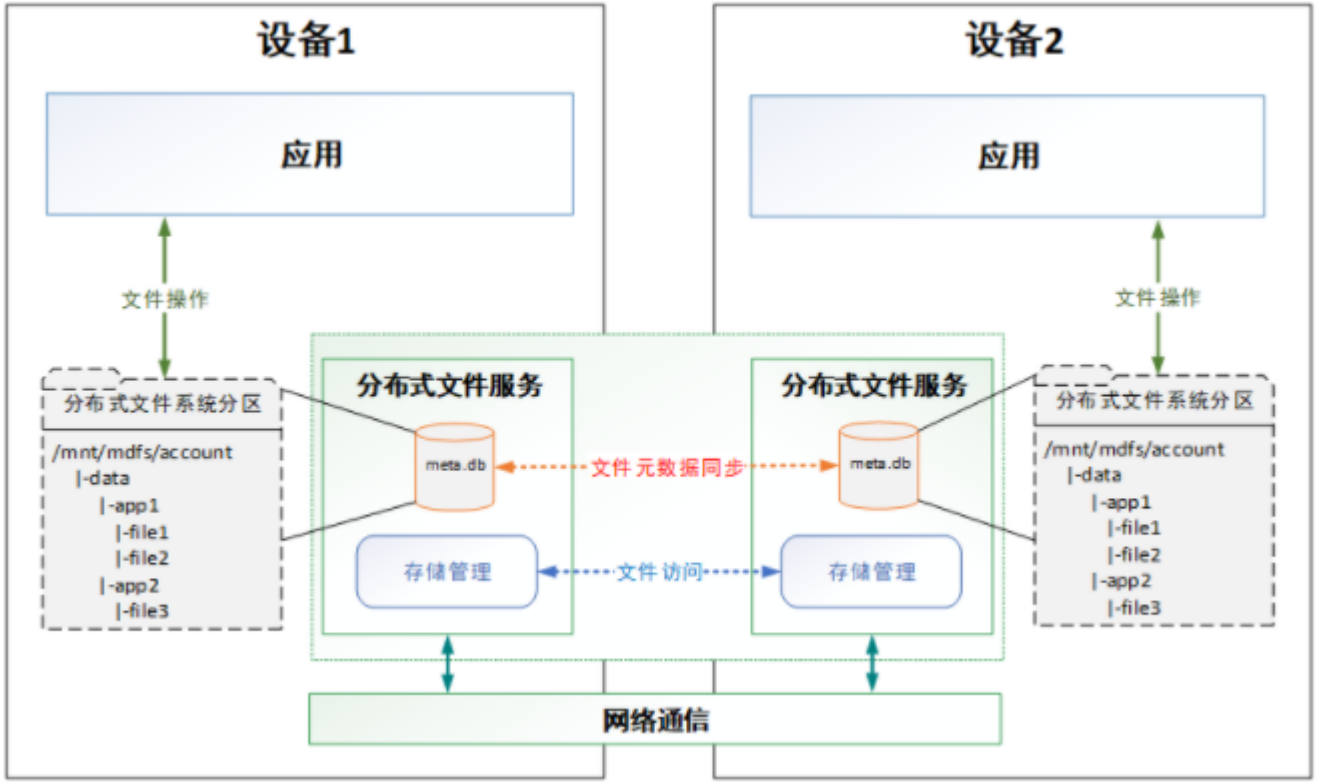
```
// 插入数据
mSingleKvStore.putString(key, value);
// 读取数据
String value = mSingleKvStore.getString(key);
// 删除数据
mSingleKvStore.delete(key);
// 删除远端设备的数据（不会同步到其他设备）
mSingleKvStore.removeDeviceData(key);
手动同步下三种模式
将远端设备数据 PULL 到本设备
List<String> deviceIdList = getConnectedDevices();
kvStore.sync(deviceIdList, SyncMode.PULL_ONLY);
将本设备数据 PUSH 到远端设备
List<String> deviceIdList = getConnectedDevices();
kvStore.sync(deviceIdList, SyncMode.PUSH_ONLY);
本设备与远端设备数据双向同步 PUSH_PULL
List<String> deviceIdList = getConnectedDevices();
kvStore.sync(deviceIdList, SyncMode.PUSH_PULL);
```

分布式关键技术及应用场景 - 联系人同步场景

HUAWEI

分布式关键技术及应用场景 - 分布式文件服务

- 分布式文件服务提供多设备之间的文件共享能力，支持相同帐号下同一应用文件的跨设备访问。
- 采用无中心节点的设计，每个设备都存储一份全量的文件元数据和本设备上产生的分布式文件，元数据在多台设备间互相同步，当应用需要访问分布式文件时，分布式文件服务首先查询本设备上的文件元数据，获取文件所在的存储设备，然后对存储设备上的分布式文件服务发起文件访问请求，将文件内容读取到本地。



分布式关键技术及应用场景 - 文件跨设备共享

图1 设备A完成邮件编写并选择附件，流转到另一设备

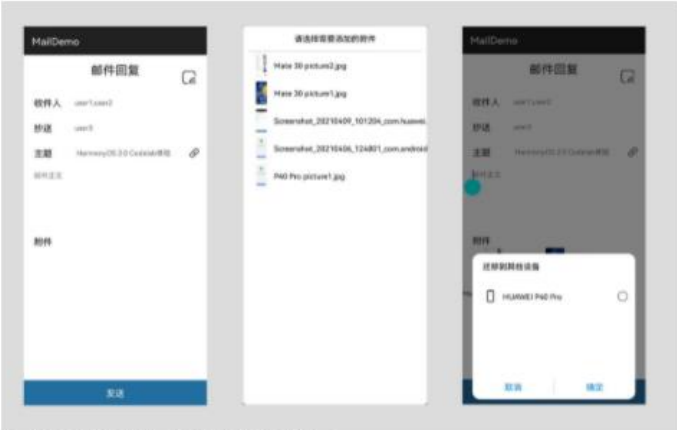


图2 设备B弹出邮件界面，可继续完成邮件编写

文件跨设备共享

```
// 设备A创建文件
Context context;
... // context初始化
File distDir = context.getDistributedDir();
String filePath = distDir + File.separator + “XX";
FileWriter fileWriter = new FileWriter(filePath, true);
fileWriter.write(XX);
fileWriter.close();
```

```
// 设备B读取文件
... // context初始化
File distDir = context.getDistributedDir();
String filePath = distDir + File.separator + “XX";
FileReader fileReader = new FileReader(filePath);
char[] buffer = new char[1024];
fileReader.read(buffer);
fileReader.close();
System.out.println(buffer);
```

开发工具简介 - HUAWEI DevEco Studio

- 基于IntelliJ IDEA Community开源版本打造，面向华为终端全场景多设备的一站式集成开发环境（IDE）
- 多设备统一开发环境
  - 包括Phone、Tablet、Car、TV、Wearable、LiteWearable、Smart Vision设备
- 支持多语言的代码开发和调试
  - Java、XML、C/C++ 、 JS、CSS、HML
- 支持FA和PA快速开发
  - 提供FA/PA工程模板，一键打包等。
- 支持分布式多端应用开发
  - 代码最大化重用，跨设备运行，不同设备差异化开发。
- 支持多设备模拟器
  - 提供多设备的模拟器，提高调试效率。
- 支持多设备预览器
  - 实时预览编码效果，支持多设备同时预览。

项目实践

参考： HarmonyOS应用开发实验手册.docx

思考题

1. 如何通过分布式能力充分利用现有设备?

更多信息

- 鸿蒙官网
  - https://www.harmonyos.com/cn/home/
- 开源代码仓库
  - https://openharmony.gitee.com
- 反馈issue
  - https://gitee.com/organizations/openharmony/issues
- 华为开发者联盟论坛
  - https://developer.huawei.com/consumer/cn/forum/block/harmonyos

学习推荐

HarmonyOS官网

HarmonyOS开发者
微信公众号

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and organization for a fully connected, intelligent world.

HUAWEI