如何从 OS 框架层面实现应用服务功能解耦

作者：赵俊民

01 背景

在 HarmonyOS 中，将 Ability 作为应用的基本构成单元，调度单元，迁移单元，服务单元。

何为服务单元呢？在 Android 上开发一个应用，此应用需要去查看和编辑一个图片，如何实现这个功能？大概有两种实现方式：

1. 应用开发者集成相应的 jar/so SDK 库，通过调用 API 完成查看和编辑功能

2. 应用开发者通过调用另外一个应用（如图库）提供的图片查看和编辑功能完成

在 Android 系统上，应用的一些功能都是通过方式二实现的。方式二有如下几个好处：

1. 查看和编辑图片，可能不是应用的核心业务，无需要浪费太多的时间

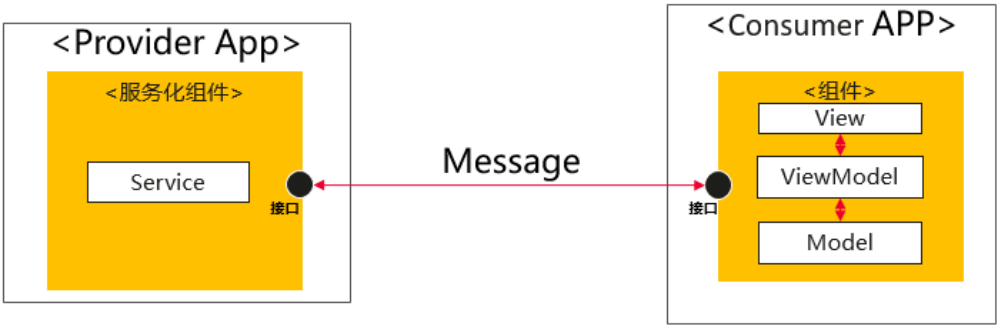
2. 应用本身 ROM 发布包体较小

3. 应用可以借助 OS 系统提供的能力，在不同的国家调用不同的图库应用，避免一些安全隐私问题

我们将方式二称之为图库应用为应用开发者提供了图片服务。

图库应用就是“服务化应用”。

来源：HarmonyOS 开发者微信号 https://mp.weixin.qq.com/s/PoYHQHeUgUkTEihwelBJrA



- Consumer App：服务的使用方
- Provider App：服务的提供方

基于上面的场景解释，本文重点讨论的问题是：如何从 OS 应用框架层面提供能力，解耦服务的提供方和使用方？是否在移动应用/IOT 领域存在服务中间商的商业模式可能性？

要回答这个问题，让我们先调研一下业界现有应用服务调用模式。

02 业界技术调研

2.1Android 系统中应用服务化设计

Android 的最大创新之一就是应用服务化，单一 UI 入口变为服务多入口，Android 应用模型不仅为用户提供了 UI 入口，而且可以暴露出更多的服务入口给其他应用。Android 的 activity，service，provider 都可以 exported，被外部应用进行调用。Android 中不同的组件类型，提供了不同的调用方式：

- **Activity 组件**

通过 Intent 方式进行调用，Intent 描述启动那个 Activity 和携带必要的启动数据。Intent 分为显示和隐式类型，其中隐式 Intent 类型不需要显示指定组件的名称，只需要描述需要组件完成的 Action 就可以。这种方式就可以把使用方和提供方进行松绑。Android 中定义了非常多的 Action：

https://developer.android.com/reference/android/content/Intent

ACTION_VIEW: Display the data to the user

ACTION_OPEN_DOCUMENT : Allow the user to select and return one or more existing documents.

ACTION_EDIT：Provide explicit editable access to the given data.

...

我们以 ACTION_VIEW 为例，Android 如何写一个被服务 Activity 被外部调用

Android 给的一个记事本 App 例子，在 Manifest 文件里面声明了一个支持 mimetype 的文件

//https://android.googlesource.com/platform/development/+/05523fb/samples/NotePad

```
<activity android:name="NoteEditor"
    android:theme="@android:style/Theme.Light"
    android:configChanges="keyboardHidden|orientation">
```

```
        <!-- This filter says that we can view or edit the data of
             a single note -->
        <intent-filter android:label="@string/resolve_edit">
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name="com.android.notes.action.EDIT_NOTE" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
        ...
 </activity>
```

如果使用方需要服务方 Activity 返回接口，调用

https://developer.android.com/training/basics/intents/filters

```
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION", Uri.parse("content://re
sult_uri"));
setResult(Activity.RESULT_OK, result);
finish();
```

- Service 组件

Service 组件被外部人调用有两种方式：startService 和 bindService。前者适合比较简单的交互通信，后者可以提供 IDL 更加丰富的交互。Service 是 Android APP 向外提供无界面服务的主要组件。GMS 提供的大部分功能都是基于 SDK(封装了 IDL) + Service 方式。可以参考：

https://developers.google.com/android/guides/overview

Google 在 Android 上通过 Service 组件调用方式，从架构上实现了 App 与 GMS 的解耦，从而实现了自己的导流的商业模式。

- **Provider 组件**

Provider 是 Android 提供的一种数据组件。如果应用需要有些数据可以被外部其他应用进行使用，基本都是通过 Provider 组件进行暴露。

providers and provider clients offer a consistent,

standard interface to data that also handles inter-process communication and secure data access.

Android 提供统一的 ContentResolver 机制访问一个 Provider

```
//https://developer.android.com/guide/topics/providers/content-provider-basics
// Does a query against the table and returns a Cursor object
mCursor = contentResolver.query(
    UserDictionary.Words.CONTENT_URI,  // The content URI of the words table
```

```
    projection,                 // The columns to return for each row
    selectionClause,            // Either null, or the word the user entered
    selectionArgs,              // Either empty, or the string the user entered
    sortOrder                   // The sort order for the returned rows
)
```

总体看，Android 的 Activity，Service 和 Provider 这三种应用组件都可以导出某种服务能力，以不同的方式被其他应用使用：

- Activity：携带 UI 界面，OS 通过 Intent 消息在提供方和消费方进行调用和回复。
- Service：无 UI 界面，

应用通过 Client Library+Service 方式对外提供服务

- Provider：应用通过 ContentResolver 机制对外暴露数据。

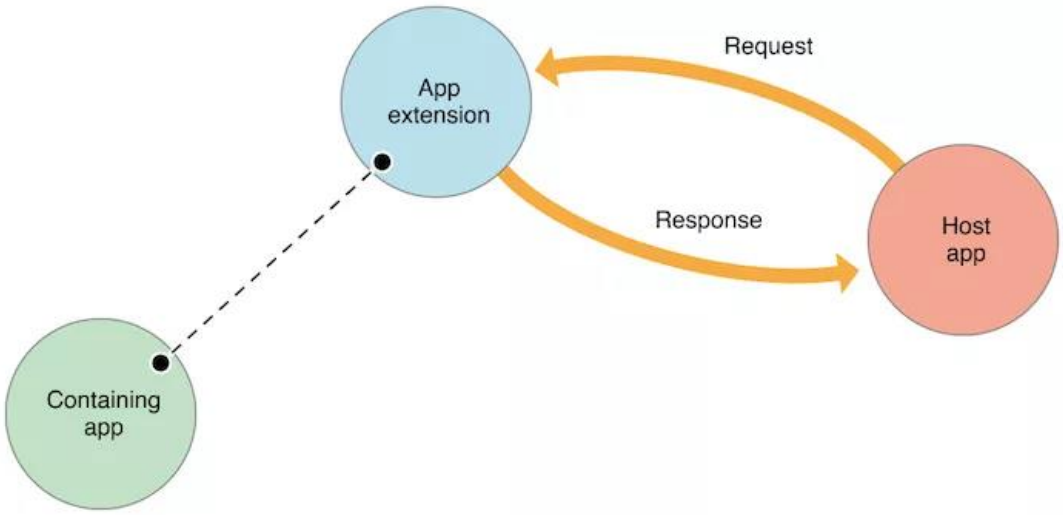
Android 提供了各种灵活的机制实现了应用服务化，部分解耦了服务的使用方和服务提供方，总体来说，功能是比较强大的，但我个人认为提供的调用机制比较复杂，是否可以简化有待商榷。

2.2 IOS 系统中应用服务化设计

IOS 并没有像 Android 一样提供如此灵活的应用组件框架，IOS 中提供了 App Extensions 机制，用于一个 APP 导出功能弥补系统功能或者给其他 APP 使用。

An app extension lets you extend custom functionality and content beyond your

app and make it available to users while they’ re interacting with other apps or the system.



https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/ExtensionOverview.html#//apple_ref/doc/uid/TP40014214-CH2-SW2

从上图可以看出，理解 App extension 需要理解三个概念：

• Host App：用户正在使用的应用，可以启动 extension，如在图库中使用分享扩展，图库即 HostApp。

• App extension: 一个独立的编程组件，跟 App 有不相同的声明周期，声明周期基本伴随 Host App 调用进行启动，调用完即销毁。

• Containing App：包含 App extension 功能代码和可以共享 Shared resources 的 App，本身 App extension 跟 Containing App 在运行中交互通过 IPC 机制。App extension 和 Containing app，Hostapp 分别运行在不同的进程。

IOS 中定义了很多类型的 App extension，

包括 Action，Audio Unit，Broadcast UI，Broadcast Upload，...,Photo Editing,Share (iOS and macOS)。

IOS App extension 设计可以看出：IOS 提供统一的 App extension 设计（组件定义，生命周期管理，通信机制，系统管理）便于三方应用开发者向系统注册能力，给其他应用进行使用。

2.3 Windows 系统中应用服务化设计

- **COM 组件模型**

Windows 系统中大家所知就是 COM（Component Object Model）组件化模型，允许应用从二进制软件组件构建应用。COM 模型定义了一套标准，按照 COM 标准实现的组件，提供统一的组件定义，注册，查询，调用，版本兼容方法。

COM has a number of parts that work together to enable the creation of applications that are built from reusable components: 1.A host system that provides a run-time environment that conforms to the COM specification. 2.Interfaces that define feature contracts, and components that implement interfaces. 3.Servers that provide components to the system, and clients that use the features provided by components. 4.A registry that tracks where components are deployed on local and remote hosts.5.A Service Control Manager that locates components on local and remote hosts and connects servers to

clients. 6.A structured storage protocol that defines how to navigate the contents of files on the host's file system.

在《Registering COM Applications - Win32 apps》中介绍了，如何注册一个 COM 组件并且提供给其他 App 开发者。COM 里面思考的问题维度和解决方案非常有借鉴意义。

- **UWP app services and app extensions**

微软的 UWP 应用框架提供了 App Services 和 AppExtensions 两种方式，可以让一个应用调用另外一个应用。

App Services 可以让一个应用为另外一个 UWP 应用提供服务。详细参考：App Service

App services are UWP apps that provide services to other UWP apps. an app service might provide a bar code scanner service that other apps could use.

App services let you create UI-less services that apps can call on the same device

App Services 运行在 HOST APP 进程，即提供服务的应用进程，不在调用者进程。跟 IOS 一样 App Services 需要在声明文件进行服务声明。调用者和服务方通过异步的通信方式完成：AppServiceConnection 和 AppServiceRequestReceivedEventArgs。

AppExtensions 允许一个应用只读访问另外一个应用提供的内容，详细参考：AppExtensions

enable your UWP app to host content provided by other UWP apps. Discover, enumerate, and access read-only content from those apps.

AppService 本身是一个后台 task，无 UI 界面，跟 Android 的 Service 实现的功能类似。

2.4 PalCom 架构

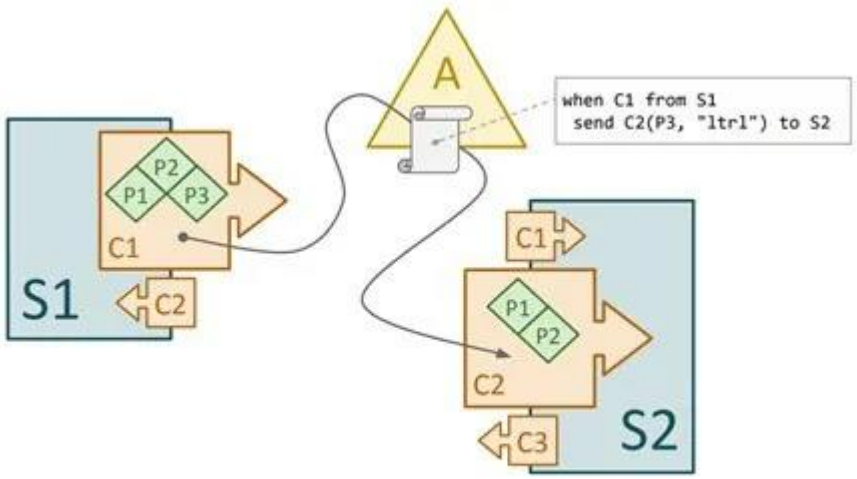


Figure 1.4: Assembly (A) *configuration* of services (S) and command (C) *coordination* with selective parameter (P) passing

来自《Factoring out Glue-code in Systems of IoT Devices》

在论文《Factoring out Glue-code in Systems of IoT Devices》中讲了一套已经商用的 IOT 设备开发系统，IOT 设备对外提供服务（Service-based），设备间采用消息通讯，服务间通过胶水代码(Assembly)协调,把流程串起来引入了一套逻辑比较简单的 DSL 支持 invert live programming：• 传统正向编程：先把控件排布好，再填充代码逻辑把流程和呈现做好反向编程：• 用 assembly 将业务流程连好，再选择适用哪种控件呈现结果• 降低了 IOT

应用的开发难度，非软件工程师也能玩溜 IOT 互联。

由于此系统并没有开源，只能从一篇博士论文《Assemblies of Pervasive Services》中看出一些设计思考，核心概念: • Devices：execution platform for the services, and for middleware components.• Service：fundamental interaction point and building block in Pal-Com systems. The service description makes the service self-describing in network。• Connections：paths of communication between PalCom services• Tunnels：secure connections between local PalCom networks• assembly：enables combinations of services.• Versioning：many very similar assemblies, solving similar problems

其中核心有价值的思考：

1.跨设备的服务发现和调用机制：Jini，UPnP，OSGi（SOA）2.服务通信通过异步 Communication between services is asynchronous, and services are in a peer-to-peer relationship when they communicate。

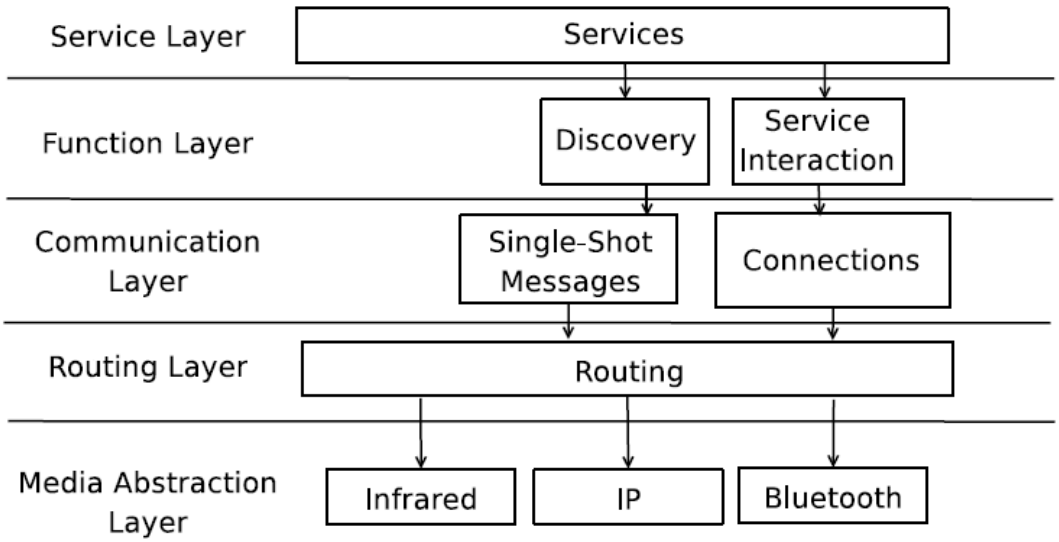


Figure 5.1: *The PalCom Communication Model.*

来源于《Assemblies of Pervasive Services》

2.5 Service-oriented architecture（SOA）

根据 wiki 解释

https://en.wikipedia.org/wiki/Service-oriented_architectureService-oriented architecture(SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. A SOA service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online. SOA is also intended to be independent of vendors, products and technologies.[1]

SOA 服务化架构常常用于云计算领域，其核心思想通过松耦合的交互机制进行服务互操作："inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language"

03 设计实现思考

现在回归到本文开头提到的两个问题：如何从 OS 应用框架层面提供能力，解耦服务的提供方和使用方？是否在移动应用/IOT 领域存在服务中间商的商业模式可能性？

3.1OS 应用框架层面提供能力分析

先从技术上回答第一个问题，从第二章节中的一些分析可以看出，应用服务化框架设计需要 OS 考虑如下几个方面能力：1.基础组件模型定义：服务导出的基本编程单元（例如 Activity，App Extension）2.组件管理：应用可以先系统注册服务组件，并且可以被使用方发现；组件选择，同类型组件服务选择。3.组件调用：提供组件调用的机制（如 Intent 机制）4.组件生命周期管理：管理组件的启动，销毁 5.组件下载和安装：设备上服务组件如果不存在，需要上云进行搜索和安装 6.组件版本管理：管理各种版本的应用组件，尤其第三方提供的服务组件存在多个版本。7.组件授权：只允许付费的，有权限的应用进行调用 8.1-6 中定义的跨设备分布式场景下支持。

- 基础组件模型构造一个应用服务化框架首先要构思这个编程单元是什么？在 HarmonyOS 中显然服务导出的编程单元就是 Ability，应用开发人员可以通过定义 Ability

给其他三方提供服务。

• 组件管理 App 中需要通过描述文件，描述这个应用中支持那些服务。在 HarmonyOS 中显然服务导出的编程单元就是 Ability 在 Profile 文件中进行描述，通过组件安装管理服务发现那个 App 中包括那些对外提供的 Ability。

• 组件调用从上面看无非是两种方式：基于消息的调用和基于服务 IDL RPC API 方式。消息是异步和松耦合的；IDL 接口定义语言（如 Android ClientLibrary）方式在兼容性方面要求更高。

• 跨设备组件调用机制跨设备服务组件调用支持，需要 OS 提供跨设备的通信机制，串行化数据。

• 组件版本管控如何实现多个版本的组件版本管理，这个是比较困难的一件事情，最近看到 nixos 引入了一个 Purely Functional Package Manager。

"Nix is a purely functional package manager. This means that it treats packages like values in purely functional programming languages such as Haskell — they are built by functions that don’ t have side-effects, and they never change after they have been built. Nix stores packages in the Nix store, usually the directory /nix/store, where each package has its own unique subdirectory such as /nix/store/b6gvzjyb2pg0kjfwrjmg1vfhh54ad73z-firefox-33.1/

where b6gvzjyb2pg0… is a unique identifier for the package that captures all its dependencies (it’ s a cryptographic hash of the package’ s build dependency graph). This enables many powerful features.

3.2 服务中间商的商业模式探讨

是否在移动应用/IOT 领域存在服务中间商的商业模式可能性？刚开头我们提到了目前很多专注算法或者服务公司给应用开发者提供 jar/so SDK 方式，直接运行在应用进程，导致应用发布的包体变大，更新慢。另外基于服务化的应用调用方式，还可以针对不同的区域调用不同的服务。例如应用开发者想使用地图服务，地图服务可能非常多，国内用 baidu，国外用 Google，开发者可以不关心使用什么地图，比较出海的隐私和安全是非常重要的。

如果在基于支持应用服务化框架的 OS 上，是否存在一个服务中间商，用于负责对应用提供的服务进行版本管理，兼容性测试呢？这个问题在 “操作系统这些事儿"群里面做了简单讨论：

康老师：版本兼容性问题才是根本，开发者不愿意接受你给增加开发负担以及部署负担。开发者不说我者东西只能在什么版本上跑，中间层哪敢下结论。如果一个服务中间商存在，关键看你付出的成本和解决的问题对比。直接给我自己打包一个 so 就能解决问题，你非得让我和中间商协调，省那点内存和我开发者何干。只要收益大于风险就可以做。比如 reactnative 和 flutter 这种东西，尽量抹平安卓/ios 差异，虽然也有问题，但是大家扑上去用。

康老师提了一些关键问题，如版本兼容性问题。针对服务提供方如何赚钱，服务中间商如何赚钱也是关键问题。针对服务中间商需要提供如下的能力：• 服务计费设施：如果进行服务

授权，费用计费，保证服务提供方的商业诉求• 服务测试设施：如何进行服务的兼容性，可靠性测试• 服务版本管理：不同版本的服务进行管理，及其跟设备侧的自动版本下载和管理• 服务 API DOC：API Doc 管理设施• 调试设施：实现服务调试，试用等机制。

本文参考文献

1.https://developer.android.com/guide/components/intents-filters

2.https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/index.html

3.https://dl.acm.org/doi/10.1145/3328433.3328436

4.https://stackoverflow.com/questions/25240106/in-which-process-runs-ios-app-extension

5.https://developers.google.com/android/guides/overview

6.https://www.cs.umd.edu/~pugh/com/

7.https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4267594

8.https://robtiffany.com/wp-content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf

9.https://docs.microsoft.com/en-us/windows/uwp/launch-resume/app-services