# Particle Ability（无UI界面）

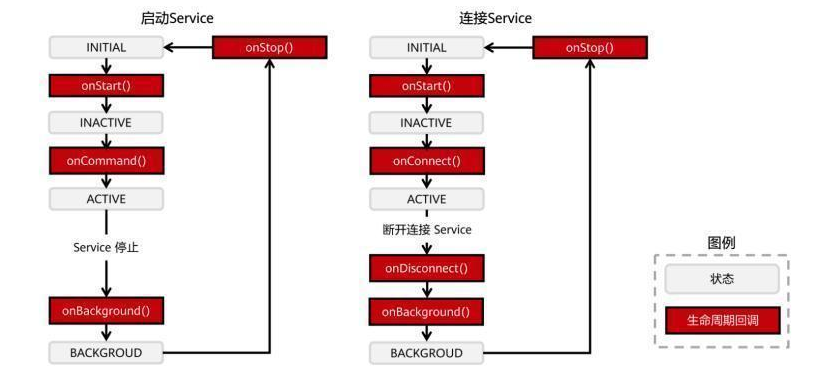
* PA具有后台运行任务的能力
* 元服务：
  + 由三方开发，实现单一功能的程序实体
  + 无UI，仅对系统服务有依赖关系（系统服务：由OS提供的基础软件服务和硬件服务）
* PA支持Service模板和Data模板

## Service Ability

* 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
* 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，防止造成主线程阻塞，应用程序无响应
* 创建Service：
  + 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，开发者可以重写这些方法，来添加其他Ability请求与Service Ability交互时的处理方法：
    - 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，调用时传入的Intent应为空
    - 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，开发者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，开发者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
* 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
* 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

## Service Ability生命周期

* 调用Service的方法不同，其生命周期有以下两种路径：启动Service和连接Service



* 启动Service：
  + 通过startAbility()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
    - BundleName：表示包名称
    - AbilityName：表示待启动的Ability名称
* 连接Service：
  + 如果Service需要与Page Ability或其他应用的Service Ability进行交互，则须创建用于连接的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()是用来处理连接Service成功的回调（连接的回调）
    - onAbilityDisconnectDone()是用来处理Service异常死亡的回调（断开连接的回调）
  + 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LocalRemoteObject来创建自定义的实现类
  + 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
  + connectAbility()也可以连接通过startAbility()创建的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。要手动停止Service，有以下两种方法：
    - 在Service中通过terminateAbility()停止本Service
    - 在其他Ability调用stopAbility()来停止Servic
  + 停止Service同样支持停止本地设备Service和停止远程设备Service，使用方法和启动Service一样
  + 一旦调用停止Service的方法，系统便会尽快销毁Service

## 前台Service

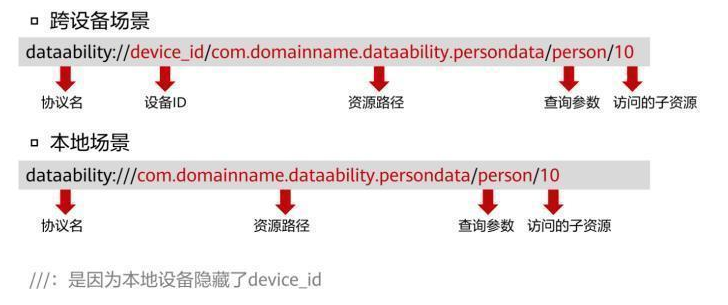
* 一般情况下，Service都是在后台运行的，后台Service的优先级都是比较低的，当资源不足时，系统有可能回收正在运行的后台Service
* 在一些场景下（如播放音乐），用户希望应用能够一直保持运行，此时就需要使用前台Service。前台Service会始终保持正在运行的图标在系统状态栏显示
* 使用前台Service并不复杂，开发者只需在Service创建的方法里，调用keepBackgroundRunning()将Service与通知绑定。调用keepBackgroundRunning()方法前需要在配置文件中声明ohos.permission.KEEP\_BACKGROUND\_RUNNING权限，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

# Data Ability

* Data Ability用于对外部提供统一的数据访问抽象
* 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
* 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
* 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

# URL

* Data的提供方和使用方都通过URL（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，则为目标设备的ID；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



# 创建Data

* 实现UserDataAbility：
  + UserDataAbility用于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
  + 实现UserDataAbility，需要在“Project”窗口当前工程的主目录（“entry > src > main > java > com.xxx.xxx”）选择“File > New > Ability > Empty Data Ability”，设置“Data Name”后完成UserDataAbility的创建
  + 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
* ohos.rpc.MessageParcel类提供了一个静态方法，用于获取MessageParcel实例。开发者可通过获取到的MessageParcel实例，使用dupFileDescriptor()函数复制待操作文件流的文件描述符，并将其返回，供远端应用访问文件
  + - 数据库存储：
      * 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
* 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：
    - query()：查询数据库
      * ResultSet query(Uri uri, String[] columns, DataAbilityPredicates predicates)
      * 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
    - insert()：向数据库中插入单条数据
      * int insert(Uri uri, ValuesBucket value)
      * 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
    - batchInsert()：向数据库中插入多条数据
      * int batchInsert(Uri uri, ValuesBucket[] values)
      * 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()：删除一条或多条数据
      * int delete(Uri uri, DataAbilityPredicates predicates)
      * 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
    - update()：更新数据库
      * int update(Uri uri, ValuesBucket value, DataAbilityPredicates predicates)
      * 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
    - executeBatch()：批量操作数据库
      * DataAbilityResult[]executeBatch(ArrayList<DataAbilityOperation>operations)
      * 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，开发者必须在配置文件中注册Data
  + 配置文件中该字段在创建Data Ability时会自动创建，name与创建的Data Ability一致
  + 需要关注以下属性：
    - 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

# 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具体的使用步骤：

1. 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
2. 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：DataAbilityHelper helper = DataAbilityHelper.creator(this);
3. 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：
         * 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,这些方法的使用说明如下:
         * query()：查询数据库

ResultSet query(Uri uri, String[] columns, DataAbilityPredicates predicates)

该方法接收三个参数，uri是查询的目标路径，columns是查询的列名（字段），以及查询条件查询条件由类DataAbilityPredicates构建

* + - * + insert()：向数据库中插入单条数据

int insert(Uri uri, ValuesBucket value)

新增方法，该方法接收两个参数，其中uri为目标资源路径，ValuesBucket为要新增的数据值

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

* + - * + batchInsert()：向数据库中插入多条数据

int batchInsert(Uri uri, ValuesBucket[] values)

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()：删除一条或多条数据

int delete(Uri uri, DataAbilityPredicates predicates)

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

* + - * + update()：更新数据库

int update(Uri uri, ValuesBucket value, DataAbilityPredicates predicates)

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

* + - * + executeBatch()：批量操作数据库

DataAbilityResult[]executeBatch(ArrayList<DataAbilityOperation>operations)

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