(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 109857489 B
(45)授权公告日 2022.05.27

(21)申请号 201711243058.9
(22)申请日 2017.11.30
(65)同一申请的已公布的文献号
申请公布号 CN 109857489 A
(43)申请公布日 2019.06.07
(73)专利权人 阿里巴巴集团控股有限公司
地址 英属开曼群岛大开曼资本大厦一座四层847号邮箱
(72)发明人 谭博颖
(74)专利代理机构 北京润泽恒知识产权代理有限公司 11319
专利代理师 苏培华
(51)Int.Cl.
G06F 9/451(2018.01)
G06F 9/48(2006.01)

(56)对比文件
匿名.单元测试Robolectric的使用详解.《CSDN博客》.2017,
匿名.Android 业务组件化开发实践.《博客园》.2017,

审查员 刘迪

权利要求书4页 说明书23页 附图9页

(54)发明名称

基于Android系统的开发系统、解耦方法和装置

(57)摘要

本申请实施例提供了一种基于Android系统的开发系统、解耦方法和装置,所述开发系统包括:组件管理单元Controller,用于管理Android系统中的业务组件,所述Controller具有与注册后的业务组件的生命周期相适配的相关属性;生命周期管理单元LifeCycle,用于提供所述业务组件的生命周期接口;以及,将所述业务组件的生命周期分发到对应的Controller中;消息管理单元Messager,用于处理所述Controller之间的通信。本实施例通过采用模块化编程解决方案,可以实现大规模应用的组件化,降低了编程复杂度以及依赖耦合度,使得业务组件之间互不依赖,从而业务组件的灵活度也更高。

CN 109857489 B

1.一种基于Android系统的开发系统,其特征在于,所述开发系统包括:

组件管理单元Controller,用于管理Android系统中的业务组件,所述Controller具有与注册后的业务组件的生命周期相适配的相关属性;所述业务组件包括Activity组件和Fragment组件;

生命周期管理单元LifeCycle,用于提供所述业务组件的生命周期接口;以及,将所述业务组件的生命周期分发到对应的Controller中;

消息管理单元Messager,用于处理所述Controller之间的通信。

2.根据权利要求1所述的开发系统,其特征在于,所述Fragment组件被嵌入所述Activity组件中,通过执行所述Fragment组件的生命周期回调方法以实现相应的功能。

3.根据权利要求2所述的开发系统,其特征在于,所述组件管理单元Controller包括:

组件管理子单元ControllerManager,用于管理所述业务组件的业务功能;

基础管理子单元BaseController,用于管理所述业务组件的基础功能。

4.根据权利要求3所述的开发系统,其特征在于,所述业务功能包括如下任意一种或多种:

实例化业务组件、增加业务组件、修改业务组件、删除业务组件,或,查找业务组件。

5.根据权利要求3所述的开发系统,其特征在于,所述基础功能包括如下任意一种或多种:

消息注册、消息反注册、发送局部消息,或,发送全局消息。

6.根据权利要求3所述的开发系统,其特征在于,所述组件管理单元Controller还包括:

Activity组件实现子单元BaseActivityController,用于实现所述Activity组件对应的生命周期接口,以调用所述Activity组件生命周期接口中的方法;

Fragment组件实现子单元BaseFragmentController,用于实现所述Fragment组件对应的生命周期接口,以调用所述Fragment组件生命周期接口中的方法。

7.根据权利要求6所述的开发系统,其特征在于,还包括:

组件管理创建单元ControllerCreator,用于在所述业务组件注册时,创建所述组件管理单元Controller;

Activity组件管理创建单元BaseActivityControllerCreator,用于创建所述Activity组件实现子单元BaseActivityController;

Fragment组件管理创建单元BaseFragmentControllerCreator,用于创建所述Fragment组件实现子单元BaseFragmentController。

8.根据权利要求2所述的开发系统,其特征在于,所述消息管理单元Messager包括:

局部消息管理子单元LocalMessager,用于与任一Activity组件或Fragment组件绑定,以管理被绑定的Activity组件或Fragment组件对应的Controller中的消息;

全局消息管理子单元MessageManager,用于与Activity组件绑定,以管理被绑定的Activity组件中的消息。

9.根据权利要求8所述的开发系统,其特征在于,所述消息管理单元Messager还包括:

消息管理回调子单元MessageManagerListener,用于将所述局部消息管理子单元LocalMessager注册到所述全局消息管理子单元MessageManager中,以实现所述

MessageManager对所述LocalMessager的管理。

10.根据权利要求9所述的开发系统，其特征在于，所述局部消息管理子单元LocalMessager，还用于接收所述业务组件管理单元Controller发送的消息，并将所述消息发送至全局消息管理子单元MessageManager；

所述全局消息管理子单元MessageManager，还用于将接收到的所述消息转发至其他业务组件对应的局部消息管理子单元LocalMessager。

11.根据权利要求2所述的开发系统，其特征在于，所述生命周期管理单元LifeCycle包括：

生命周期管理子单元，用于提供所述业务组件的生命周期接口；

生命周期分发子单元，用于将所述业务组件的生命周期分发到对应的Controller中。

12.根据权利要求11所述的开发系统，其特征在于，所述生命周期管理子单元包括：

Activity生命周期管理子单元，用于提供所述Activity组件的生命周期接口；

Fragment生命周期管理子单元，用于提供所述Fragment组件的生命周期接口。

13.根据权利要求11所述的开发系统，其特征在于，所述生命周期分发子单元包括：

Activity生命周期分发子单元，用于将所述Activity组件的生命周期分发到对应的Controller中；

Fragment生命周期分发子单元，用于将所述Fragment组件的生命周期分发到对应的Controller中。

14.根据权利要求2所述的开发系统，其特征在于，还包括：

代理对接单元，用于实现与所述Android系统中业务组件的对接。

15.根据权利要求14所述的开发系统，其特征在于，所述代理对接单元包括：

Activity代理对接单元GrandPiano，用于实现与所述Android系统中的Activity组件的对接；

Fragment代理对接单元GrandPianoFgt，用于实现与所述Android系统中的Fragment组件的对接。

16.一种基于Android系统的解耦方法，其特征在于，所述Android系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager；所述组件管理单元Controller，用于管理Android系统中的业务组件，所述Controller具有与注册后的业务组件的生命周期相适配的相关属性；所述业务组件包括Activity组件和Fragment组件；所述生命周期管理单元LifeCycle，用于提供所述业务组件的生命周期接口；以及，将所述业务组件的生命周期分发到对应的Controller中；所述消息管理单元Messager，用于处理所述Controller之间的通信；所述方法包括：

当消息管理单元接收到针对业务对象的操作事件时，所述消息管理单元确定目标业务对象；

所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息；

所述目标业务对象依据所述事件消息，执行相应的操作。

17.一种基于Android系统的解耦方法，其特征在于，所述Android系统中包括业务组件，所述业务组件包括Activity组件和Fragment组件，所述业务组件具有生命周期；所述Android系统中还包括一开发系统，所述开发系统中设置有组件管理单元Controller、生命

周期管理单元LifeCycle和消息管理单元Messager，所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager，当所述业务组件被注册到Controller中，所述Controller具有与所述生命周期相适配的相关属性；所述方法包括：

当接收到针对业务对象的点击事件时，所述业务对象对应的消息管理单元确定目标业务对象；

所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息；

所述目标业务对象依据所述事件消息，执行相应的操作。

18.根据权利要求17所述的方法，其特征在于，所述业务对象由所述业务组件生成，所述当接收到针对业务对象的点击事件时，所述业务对象对应的消息管理单元确定目标业务对象，包括：

当接收到针对业务对象的点击事件时，所述业务对象对应的组件管理单元针对所述点击事件生成事件消息，并将所述事件消息发送至所述消息管理单元；

所述消息管理单元依据所述事件消息，确定目标业务对象。

19.根据权利要求18所述的方法，其特征在于，在所述业务对象对应的组件管理单元针对所述点击事件生成事件消息之前，还包括：

所述组件管理单元注册到所述消息管理单元，所述消息管理单元用于对各个业务对象生成的事件消息进行管理。

20.根据权利要求18所述的方法，其特征在于，所述目标业务对象由目标业务组件生成，所述目标业务对象具有对应的目标消息管理单元，所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息，包括：

所述消息管理单元向所述目标业务对象对应的目标消息管理单元发送针对所述点击事件生成的事件消息。

21.根据权利要求20所述的方法，其特征在于，所述消息管理单元向所述目标业务对象对应的目标消息管理单元发送针对所述点击事件生成的事件消息，包括：

所述消息管理单元判断目标业务对象对应的业务组件与当前的业务对象对应的业务组件是否位于同一Activity组件中；

若是，则所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元，由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标消息管理单元；

若否，则所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元，由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标全局消息管理单元，所述目标全局消息管理单元用于将所述事件消息分发至目标业务对象对应的目标消息管理单元。

22.根据权利要求20所述的方法，其特征在于，所述目标业务对象还具有对应的目标组件管理单元，所述目标业务对象依据所述事件消息，执行相应的操作，包括：

所述目标消息管理单元向所述目标组件管理单元转发所述事件消息；

所述目标组件管理单元依据所述事件消息，控制所述目标业务对象执行相应操作。

23.一种基于Android系统的解耦装置，其特征在于，所述Android系统中设置有组件管

理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager;所述组件管理单元Controller,用于管理Android系统中的业务组件,所述Controller具有与注册后的业务组件的生命周期相适配的相关属性;所述业务组件包括Activity组件和Fragment组件;所述生命周期管理单元LifeCycle,用于提供所述业务组件的生命周期接口;以及,将所述业务组件的生命周期分发到对应的Controller中;所述消息管理单元Messager,用于处理所述Controller之间的通信;所述装置包括:

目标业务对象确定模块,用于当消息管理单元接收到针对业务对象的操作事件时,所述消息管理单元确定目标业务对象;

事件消息发送模块,用于所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息;

执行模块,用于所述目标业务对象依据所述事件消息,执行相应的操作。

24.一种基于Android系统的解耦装置,其特征在于,所述Android系统中包括业务组件,所述业务组件包括Activity组件和Fragment组件,所述业务组件具有生命周期;所述Android系统中还包括一开发系统,所述开发系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager,所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager,当所述业务组件被注册到Controller中,所述Controller具有与所述生命周期相适配的相关属性;所述装置包括:

确定模块,用于当接收到针对业务对象的点击事件时,采用所述业务对象对应的消息管理单元确定目标业务对象;

发送模块,用于采用所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息;

执行模块,用于采用所述目标业务对象,依据所述事件消息,执行相应的操作。

25.一种基于Android系统的解耦装置,其特征在于,包括一个或多个处理器;和,其上存储的一个或多个计算机可读介质中的指令,当由所述一个或多个处理器执行时,使得所述装置执行如权利要求16或17-22中一个或多个的方法。

26.一个或多个计算机可读介质,其上存储有指令,当由一个或多个处理器执行时,使得终端执行如权利要求16或17-22中一个或多个的方法。

基于Android系统的开发系统、解耦方法和装置

技术领域

[0001] 本申请涉及计算机技术领域，特别是涉及一种基于Android系统的开发系统、一种基于Android系统的解耦方法、一种基于Android系统的解耦装置和一个或多个计算机可读介质。

背景技术

[0002] 在Android系统中，Activity是最基本也最为常用的组件，Activity能够提供一个屏幕，每一个Activity被给予一个窗口，在该窗口上面可以绘制用户接口。窗口通常充满屏幕，但也可以小于屏幕而浮于其他窗口之上。Activity是一个负责与用户交互的组件，其中所有的操作都与用户密切相关。为了完成某项任务，用户可以通过Activity来进行交互。例如，拨号、拍照、发送E-mail、查看地图等等。Activity上面可以显示一些控件(如通过setContentView(View)来显示指定控件)，也可以监听并处理用户事件并作出响应。在一个Android应用中，一个Activity通常就是一个单独的屏幕，各个Activity之间通过Intent进行通信。

[0003] 自Android 3.0后引入了Fragment，Fragment主要用于在不同的屏幕尺寸中展现不同的内容。Fragment必须被嵌入Activity中使用，并总是作为Activity的组成部分。但是，由于Activity和Fragment之间错综复杂的生命周期关系，在已有的Android系统开发过程中，将不可避免地会遇到以下问题：

[0004] 1、在Activity和Fragment创建、绑定或配置View时，包含大量的视图相关的代码，如果要对其进行写单元测试，必须首先对业务逻辑与视图代码进行解耦，否则很难完成写单元测试。

[0005] 2、由于Activity和Fragment中需要创建、绑定和配置View，导致各个类中十分膨胀，Activity或Fragment之间的耦合无法实现复用，可读性和可维护性较低。

[0006] 3、在实际业务中，各个组件之间的相互依赖、状态传递等联系导致关系链复杂、状态传递路径过深、状态更新通知变得极为复杂。

发明内容

[0007] 鉴于上述问题，提出了本申请实施例以便提供一种克服上述问题或者至少部分地解决上述问题的一种基于Android系统的开发系统、一种基于Android系统的解耦方法、一种基于Android系统的解耦装置和相应的一个或多个计算机可读介质。

[0008] 为了解决上述问题，本申请公开了一种基于Android系统的开发系统，所述开发系统包括：

[0009] 组件管理单元Controller，用于管理Android系统中的业务组件，所述Controller具有与注册后的业务组件的生命周期相适配的相关属性；

[0010] 生命周期管理单元LifeCycle，用于提供所述业务组件的生命周期接口；以及，将所述业务组件的生命周期分发到对应的Controller中；

[0011] 消息管理单元Messager,用于处理所述Controller之间的通信。

[0012] 可选地,所述业务组件包括Activity组件和Fragment组件,所述Fragment组件被嵌入所述Activity组件中,通过执行所述Fragment组件的生命周期回调方法以实现相应的功能。

[0013] 可选地,所述组件管理单元Controller包括:

[0014] 组件管理子单元ControllerManager,用于管理所述业务组件的业务功能;

[0015] 基础管理子单元BaseController,用于管理所述业务组件的基础功能。

[0016] 可选地,所述业务功能包括如下任意一种或多种:

[0017] 实例化业务组件、增加业务组件、修改业务组件、删除业务组件,或,查找业务组件。

[0018] 可选地,所述基础功能包括如下任意一种或多种:

[0019] 消息注册、消息反注册、发送局部消息,或,发送全局消息。

[0020] 可选地,所述组件管理单元Controller还包括:

[0021] Activity组件实现子单元BaseActivityController,用于实现所述Activity组件对应的生命周期接口,以调用所述Activity组件生命周期接口中的方法;

[0022] Fragment组件实现子单元BaseFragmentController,用于实现所述Fragment组件对应的生命周期接口,以调用所述Fragment组件生命周期接口中的方法。

[0023] 可选地,还包括:

[0024] 组件管理创建单元ControllerCreator,用于在所述业务组件注册时,创建所述组件管理单元Controller;

[0025] Activity组件管理创建单元BaseActivityControllerCreator,用于创建所述Activity组件实现子单元BaseActivityController;

[0026] Fragment组件管理创建单元BaseFragmentControllerCreator,用于创建所述Fragment组件实现子单元BaseFragmentController。

[0027] 可选地,所述消息管理单元Messager包括:

[0028] 局部消息管理子单元LocalMessager,用于与任一Activity组件或Fragment组件绑定,以管理被绑定的Activity组件或Fragment组件对应的Controller中的消息;

[0029] 全局消息管理子单元MessageManager,用于与Activity组件绑定,以管理被绑定的Activity组件中的消息。

[0030] 可选地,所述消息管理单元Messager还包括:

[0031] 消息管理回调子单元MessageManagerListener,用于将所述局部消息管理子单元LocalMessager注册到所述全局消息管理子单元MessageManager中,以实现所述MessageManager对所述LocalMessager的管理。

[0032] 可选地,所述局部消息管理子单元LocalMessager,还用于接收所述业务组件管理单元Controller发送的消息,并将所述消息发送至全局消息管理子单元MessageManager;

[0033] 所述全局消息管理子单元MessageManager,还用于将接收到的所述消息转发至其他业务组件对应的局部消息管理子单元LocalMessager。

[0034] 可选地,所述生命周期管理单元LifeCycle包括:

[0035] 生命周期管理子单元,用于提供所述业务组件的生命周期接口;

[0036] 生命周期分发子单元,用于将所述业务组件的生命周期分发到对应的Controller中。

[0037] 可选地,所述生命周期管理子单元包括:

[0038] Activity生命周期管理子单元,用于提供所述Activity组件的生命周期接口;

[0039] Fragment生命周期管理子单元,用于提供所述Fragment组件的生命周期接口。

[0040] 可选地,所述生命周期分发子单元包括:

[0041] Activity生命周期分发子单元,用于将所述Activity组件的生命周期分发到对应的Controller中;

[0042] Fragment生命周期分发子单元,用于将所述Fragment组件的生命周期分发到对应的Controller中。

[0043] 可选地,还包括:

[0044] 代理对接单元,用于实现与所述Android系统中业务组件的对接。

[0045] 可选地,所述代理对接单元包括:

[0046] Activity代理对接单元GrandPiano,用于实现与所述Android系统中的Activity组件的对接;

[0047] Fragment代理对接单元GrandPianoFgt,用于实现与所述Android系统中的Fragment组件的对接。

[0048] 为了解决上述问题,本申请公开了一种基于Android系统的解耦方法,所述Android系统中设置有消息管理单元Messager;所述方法包括:

[0049] 当消息管理单元接收到针对业务对象的操作事件时,所述消息管理单元确定目标业务对象;

[0050] 所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息;

[0051] 所述目标业务对象依据所述事件消息,执行相应的操作。

[0052] 为了解决上述问题,本申请公开了一种基于Android系统的解耦方法,所述Android系统中包括业务组件,所述业务组件具有生命周期;所述Android系统中还包括一开发系统,所述开发系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager,所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager,当所述业务组件被注册到Controller中,所述Controller具有与所述生命周期相适配的相关属性;所述方法包括:

[0053] 当接收到针对业务对象的点击事件时,所述业务对象对应的消息管理单元确定目标业务对象;

[0054] 所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息;

[0055] 所述目标业务对象依据所述事件消息,执行相应的操作。

[0056] 可选地,所述业务对象由所述业务组件生成,所述当接收到针对业务对象的点击事件时,所述业务对象对应的消息管理单元确定目标业务对象,包括:

[0057] 当接收到针对业务对象的点击事件时,所述业务对象对应的组件管理单元针对所述点击事件生成事件消息,并将所述事件消息发送至所述消息管理单元;

[0058] 所述消息管理单元依据所述事件消息,确定目标业务对象。

[0059] 可选地,在所述业务对象对应的组件管理单元针对所述点击事件生成事件消息之

前,还包括:

[0060] 所述组件管理单元注册到所述消息管理单元,所述消息管理单元用于对各个业务对象生成的事件消息进行管理。

[0061] 可选地,所述目标业务对象由目标业务组件生成,所述目标业务对象具有对应的目标消息管理单元,所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息,包括:

[0062] 所述消息管理单元向所述目标业务对象对应的目标消息管理单元发送针对所述点击事件生成的事件消息。

[0063] 可选地,所述消息管理单元向所述目标业务对象对应的目标消息管理单元发送针对所述点击事件生成的事件消息,包括:

[0064] 所述消息管理单元判断目标业务对象对应的业务组件与当前的业务对象对应的业务组件是否位于同一Activity组件中;

[0065] 若是,则所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元,由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标消息管理单元;

[0066] 若否,则所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元,由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标全局消息管理单元,所述目标全局消息管理单元用于将所述事件消息分发至目标业务对象对应的目标消息管理单元。

[0067] 可选地,所述目标业务对象还具有对应的目标组件管理单元,所述目标业务对象依据所述事件消息,执行相应的操作,包括:

[0068] 所述目标消息管理单元向所述目标组件管理单元转发所述事件消息;

[0069] 所述目标组件管理单元依据所述事件消息,控制所述目标业务对象执行相应操作。

[0070] 为了解决上述问题,本申请公开了一种基于Android系统的解耦装置,所述Android系统中设置有消息管理单元Messager;所述装置包括:

[0071] 目标业务对象确定模块,用于当消息管理单元接收到针对业务对象的操作事件时,所述消息管理单元确定目标业务对象;

[0072] 事件消息发送模块,用于所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息;

[0073] 执行模块,用于所述目标业务对象依据所述事件消息,执行相应的操作。

[0074] 为了解决上述问题,本申请公开了一种基于Android系统的解耦装置,所述Android系统中包括业务组件,所述业务组件具有生命周期;所述Android系统中还包括一开发系统,所述开发系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager,所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager,当所述业务组件被注册到Controller中,所述Controller具有与所述生命周期相适配的相关属性;所述装置包括:

[0075] 确定模块,用于当接收到针对业务对象的点击事件时,采用所述业务对象对应的消息管理单元确定目标业务对象;

[0076] 发送模块，用于采用所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息；

[0077] 执行模块，用于采用所述目标业务对象，依据所述事件消息，执行相应的操作。

[0078] 为了解决上述问题，本申请公开了一种基于Android系统的解耦装置，包括一个或多个处理器；和，其上存储的一个或多个计算机可读介质中的指令，当由所述一个或多个处理器执行时，使得所述装置执行如上述基于Android系统的解耦方法中一个或多个的方法。

[0079] 为了解决上述问题，本申请公开了一个或多个计算机可读介质，其上存储有指令，当由一个或多个处理器执行时，使得终端执行如上述基于Android系统的解耦方法中一个或多个的方法。

[0080] 与背景技术相比，本申请实施例包括以下优点：

[0081] 首先，本实施例提供的开发系统是一种全新的Android项目架构形式，通过采用模块化编程解决方案，可以实现大规模应用的组件化，降低了编程复杂度以及依赖耦合度，使得业务组件之间互不依赖，从而业务组件的灵活度也更高，进而应用程序也可以轻松地实现管理、维护、复用组件等功能。

[0082] 其次，本实施例提供的开发系统可以将Controller作为容器进而接管Activity组件和Fragment组件，通过Messager实现Controller之间的通信，使得Controller之间可以相互通信但不再直接依赖，从而达到Controller之间完全解耦的状态，无论对于Activity组件或Fragment组件，其复用、维护及可读性都能得到有效保障。

附图说明

[0083] 图1是Activity生命周期的过程示意图；

[0084] 图2是Fragment生命周期的过程示意图；

[0085] 图3是Fragment与Activity生命周期的对比示意图；

[0086] 图4A是卡包收起状态的示意图；

[0087] 图4B是卡包展开且卡片未展开状态的示意图；

[0088] 图4C是卡包展开且卡片展开状态的示意图；

[0089] 图5是本申请的一种基于Android系统的开发系统的系统架构示意图；

[0090] 图6是本申请的一种基于Android系统的解耦方法实施例一的步骤流程图；

[0091] 图7是本申请的一种基于Android系统的解耦方法实施例二的步骤流程图；

[0092] 图8是本申请的一种基于Android系统的解耦装置实施例一的结构框图；

[0093] 图9是本申请的一种基于Android系统的解耦装置实施例二的结构框图；

[0094] 图10是可被用于实现本申请中所述的各个实施例的示例性系统的示意图。

具体实施方式

[0095] 为使本申请的上述目的、特征和优点能够更加明显易懂，下面结合附图和具体实施方式对本申请作进一步详细的说明。

[0096] 本申请的构思易于进行各种修改和替代形式，其具体实施例已经通过附图的方式示出，并将在这里详细描述。然而，应该理解，上述内容并不是用来将本申请的构思限制为所公开的具体形式，相反地，本申请的说明书和附加权利要求书意欲覆盖所有的修改、等同

和替代的形式。

[0097] 本说明书中的“一个实施例”,“实施例”,“一个具体实施例”等,表示所描述的实施例可以包括特定特征、结构或特性,但是每个实施例可以包括或可以不必然包括该特定特征、结构或特性。此外,这样的短语不一定指的是同一实施例。另外,在联系一个实施例描述特定特征、结构或特性的情况下,无论是否明确描述,可以认为本领域技术人员所知的范围内,这样的特征、结构或特性也与其他实施例有关。另外,应该理解的是,“在A,B和C的至少一个”这种形式所包括的列表中的条目中,可以包括如下可能的项目:(A);(B);(C);(A和B);(A和C);(B和C);或(A,B和C)。同样,“A,B或C中的至少一个”这种形式列出的项目可能意味着(A);(B);(C);(A和B);(A和C);(B和C);或(A,B和C)。

[0098] 在一些情况下,所公开的实施例可以被实施为硬件、固件、软件或其任意组合。所公开的实施例也可以实现为携带或存储在一个或多个暂时的或者非暂时的机器可读(例如计算机可读)存储介质中的指令,该指令可以被一个或多个处理器执行。机器可读存储介质可以实施为用于以能够被机器读取的形式存储或者传输信息的存储装置、机构或其他物理结构(例如易失性或非易失性存储器、介质盘、或其他媒体其他物理结构装置)。

[0099] 在附图中,一些结构或方法特征可以以特定的安排和/或排序显示。然而,优选地,这样的具体安排和/或排序并不是必要的。相反,在一些实施方案中,这样的特征可以以不同的方式和/或顺序排列,而不是如附图中所示。此外,特定的附图中的结构或方法特征中所包含的内容,不意味着暗示这种特征是在所有实施例是必须的,并且在一些实施方案中,可能不包括这些特征,或者可能将这些特征与其他特征相结合。

[0100] 为使本领域技术人员更好地理解本申请,首先对Activity和Fragment的生命周期作一说明:

[0101] 在Android系统中,Activity拥有四种基本状态:

[0102] 1.active or running

[0103] 当一个新Activity启动入栈后,它会显示在屏幕最前端(例如就是手机当前的屏幕),处理时处于栈的最顶端(Activity栈顶),此时它处于可见并可以与用户交互的激活状态,称作活动状态或者运行状态(active or running)。

[0104] 2.Paused

[0105] 当Activity失去焦点,被一个新的非全屏的Activity或者一个透明的Activity被放置在栈顶,此时的状态称作暂停状态(Paused)。此时它依然与窗口管理器保持连接,Activity依然保持活力(保持所有的状态,成员信息,和窗口管理器保持连接),但是,在系统内存极端低下的时候将被强行终止掉。所以它仍然可见,但已经失去了焦点故不可与用户进行交互。

[0106] 3.Stoped

[0107] 如果一个Activity被另外的Activity完全覆盖掉,则称作停止状态(Stopped)。它依然保持所有状态和成员信息,但是它不再可见(用户看不见),所以它的窗口被隐藏,当系统内存需要被用在其他地方的时候,Stopped的Activity将被强行终止掉。

[0108] 4.Killed

[0109] 如果一个Activity是Paused或者Stopped状态,系统可以将该Activity从内存中删除。Android系统采用两种方式进行删除,要么要求该Activity结束,要么直接终止它的

进程。当该Activity再次显示给用户时，它必须重新开始，并重置前面的状态。

[0110] 进一步地，可以参考图1所示的Activity生命周期的过程示意图，Activity的生命周期通常会涉及如下过程：

[0111] S1.启动Activity(Activity starts)：系统会先调用onCreate()方法，然后调用onStart()方法，最后调用onResume()方法，使Activity进入运行状态(Activity is running)。

[0112] S2.当前Activity被其他Activity覆盖或被锁屏(Another activity comes in front of activity)：系统会调用onPause()方法，暂停当前Activity的执行。

[0113] S3.当前Activity由被覆盖状态回到前台或解锁屏(The activity comes to the foregroud)：系统会调用onResume()方法，再次进入运行状态(Activity is running)。

[0114] S4.当前Activity转到新的Activity界面或按Home键回到主屏，自身退居后台(The activity is no longer visible)：系统会先调用onPause()方法()，然后调用onStop()方法，进入停滞状态。

[0115] S5.用户后退回到此Activity(The activity comes to the foregroud)：系统会先调用onRestart()方法，然后调用onStart()方法，最后调用onResume()方法，再次进入运行状态(Activity is running)。

[0116] S6.当前Activity处于被覆盖状态或者后台不可见状态，即第S2步和第S4步时，若系统内存不足(Other applications need memory)，则杀死当前Activity(Process is killed)；而后用户退回当前Activity(User navigates back to the activity)时：再次调用onCreate()方法、onStart()方法、onResume()方法，使Activity进入运行状态(Activity is running)；

[0117] S7.用户退出当前Activity：系统先调用onPause()方法，然后调用onStop()方法，最后调用onDestory()方法，结束当前Activity(Activity is shut down)。

[0118] 在图1中，Activity有三个关键的循环过程：

[0119] 第一，整个的生命周期，从onCreate(Bundle)开始到onDestroy()结束。Activity在onCreate()设置所有的“全局”状态，在onDestory()释放所有的资源。例如，某个Activity有一个在后台运行的线程，用于从网络下载数据，则该Activity可以在onCreate()中创建线程，在onDestory()中停止线程。

[0120] 第二，可见的生命周期，从onStart()开始到onStop()结束。在这段时间内，可以看到Activity在屏幕上，尽管有可能不在前台，不能和用户交互。在这两个接口之间，需要保持显示给用户的UI数据和资源等。例如，可以在onStart()中注册一个IntentReceiver来监听数据变化导致UI的变动，当不再需要显示时候，可以在onStop()中注销它。onStart()，onStop()都可以被多次调用，因为Activity随时可以在可见和隐藏之间转换。

[0121] 第三，前台的生命周期，从onResume()开始到onPause()结束。在这段时间内，该Activity处于所有Activity的最前面，可以与用户进行交互。Activity可以经常性地在Resumed和Paused状态之间切换。例如，当设备准备休眠时，当一个Activity处理结果被分发时，当一个新的Intent被分发时。所以在这些接口方法中的代码应该属于非常轻量级的。

[0122] Android系统在发布Android3.0时推出了Fragment，一个Fragment代表在一个Activity中用户界面的行为或一部分。可以在单独的Activity中组合多个Fragment来构建

一个多面板的UI界面，也可以在多个Activity中重用一个Fragment。因此，可以认为一个Fragment是Activity的模块化部分，它有自己的生命周期，可以接收自己的输入事件，这些事件可以在Activity运行时进行添加或删除(类似于“子活动”，可以在不同的Activity中重用)。

[0123] 一个Fragment必须始终潜入在一个Activity中，并且Fragment的生命周期直接受主Activity生命周期的影响。例如，当一个Activity暂停时，所有内部的各个Fragment都暂停了，当一个Activity销毁时，所有的Fragment都被销毁了。然而，当一个Activity在运行时，可以独立的操纵每个Fragment，例如，添加或删除它们。

[0124] 当添加一个Fragment作为Activity布局的一部分时，该Fragment生活在ViewGroup里，该ViewGroup属于Activity的View层次部分并且在Fragment定义里有它自己的view布局。可以通过在Activity的布局文件中添加<fragment>元素声明这个Fragment，或是在应用程序代码中将它添加到已存在的ViewGroup中，从而将一个Fragment插入到Activity布局中。然而，一个Fragment并不是Activity布局的必需部分，也可以将没有UI的Fragment作为Activity的后台工作者。

[0125] 通常，Fragment主要用于在大屏幕上支持更具活力和灵活性的UI设计，例如tablets(平板设备)。因为一个tablets的屏幕要比handset(手持设备)的屏幕大很多，有更大的空间用于组合和交互UI组件。Fragments允许这样的设计，当view的层次级别发生复杂变化而不需要管理。通过将Activity的布局分割成多个Fragments，就可以在运行时修改Activity的外观，以及，将这些变化保留在由Activity管理的回堆栈中。

[0126] 例如，一个消息应用程序可以用一个Fragment在左边显示文章列表，另一个Fragment在右边显示一个文章。这两个Fragment在一个Activity中并排展现，并且每个Fragment有其自己的一套生命周期回调方法，能够独自处理自己的用户输入事件。这样，代替使用在一个Activity选择一个文章，在另一个Activity读这个文章的方式，从而用户可以在同一个Activity中选择一篇文章并阅读。

[0127] 在这种情况下，可以将每个Fragment作为一个模块或可重用Activity组件来设计。也就是说，因为每个Fragment可以独自定义它自己的布局和自己的行为，通过生命周期来回调这些行为，因此可以在多个Activity中包含一个Fragment，因此应该为了重用和避免直接从一个Fragment操作另一个Fragment而设计。这样的设计是非常重要的，因为模块化的Fragment允许在不同的屏幕大小下改变Fragment组合，当设计的应用程序都支持tablets和handsets时，可以在不同的布局配置中重用Fragments以便在现有的屏幕空间中优化用户体验。例如，在一个handset中，当不止一个不能在同一个Activity中适合时，就需要分割Fragments以提供一个单独的UI面板。

[0128] 例如，以新闻应用程序为例，应用程序当在tablet大小的设备上运行时，能在ActivityA中嵌入两个Fragment，然而，在handset大小的屏幕中，由于没有足够的空间来展示两个Fragment，因此，ActivityA只能包含文章列表的fragment。当用户选择一篇文章，启动ActivityB，这个包含了第二个Fragment，可以用来读这篇文章。这样，通过在不同的组合重用Fragment，来支持在tablets和handset中的显示。

[0129] 参考图2所示的Fragment生命周期的过程示意图，由于Fragment不是独立的组件，需要被添加到Acitvity中，所以Fragment的生命周期和Activity存在一定的关联。

[0130] 以下着重介绍和Fragment生命周期有关的重要回调方法：

[0131] onAttach()(Activity)：Fragment被添加到Activity时被调用，执行该方法时，Fragment与Activity已经完成绑定，该方法有一个Activity类型的参数，代表绑定的Activity，这时候可以执行诸如mActivity＝activity的操作。

[0132] onCreate()(Bundle)：Fragment创建时调用，初始化Fragment。可以通过参数savedInstanceState获取之前保存的值。

[0133] onCreateView()(LayoutInflater,ViewGroup,Bundle)：Fragment初始化界面时调用，初始化Fragment的布局。加载布局和findViewById的操作通常在此函数内完成，但是不建议执行耗时的操作，比如读取数据库数据列表。

[0134] onActivityCreat()(Bundle)：包含Fragment的Activity的onCreate()方法执行完毕后调用，执行该方法时，与Fragment绑定的Activity的onCreate()方法已经执行完成并返回。在该方法内可以进行与Activity交互的UI操作，所以在该方法之前Activity的onCreate()方法并未执行完成，如果提前进行交互操作，会引发空指针异常。

[0135] onStart()：Fragment可见时，和Activitiy的onStart()相关联，执行该方法时，Fragment由不可见变为可见状态。

[0136] onResume()：和Activitiy的onResume()相关联，执行该方法时，Fragment处于活动状态，用户可与之交互。

[0137] onPause()：Activity onPause或者Fragment被移除时，执行该方法时，Fragment处于暂停状态，但依然可见，用户不能与之交互。

[0138] onStop()：Activity onStop或者Fragment被移除时，执行该方法时，Fragment完全不可见。

[0139] onDestroyView()：Fragment被移除时，关联的view正在被移除时调用，销毁与Fragment有关的视图，但未与Activity解除绑定，依然可以通过onCreateView()方法重新创建视图。通常在ViewPager+Fragment的方式下会调用此方法。

[0140] onDestroy()：Activity onDetach()或者Fragment被移除时调用，销毁Fragment。通常按Back键退出或者Fragment被回收时调用此方法。

[0141] onDetach()：解除与Activity的绑定。在onDestroy()方法之后调用。

[0142] 通过上面的描述能够看出，Fragment和Activity的生命周期十分相似了，区别仅仅在于Fragment中包含有几个Activity中没有的新方法，即：

[0143] onAttach()方法：Fragment和Activity建立关联的时候调用(获得Activity的传递的值)

[0144] onCreateView()方法：为Fragment创建视图(加载布局)时调用(给当前的Fragment绘制UI布局，可以使用线程更新UI)

[0145] onActivityCreated()方法：当Activity中的onCreate()方法执行完后调用(表示Activity执行onCreate()方法完成了的时候会调用此方法)

[0146] onDestroyView()方法：Fragment中的布局被移除时调用(表示Fragment销毁相关联的UI布局)

[0147] onDetach()方法：Fragment和Activity解除关联的时候调用(脱离Activity)

[0148] 可以看出，Fragment生命周期类似于Activity的生命周期。生命周期方法的对应

非常重要,因为Fragment代表Activity在工作,它的状态应该也反应了Activity的状态。因而,Fragment需要对应的生命周期方法来处理Activity的工作。以下分三个方面来详细说明:

[0149] (一)Activity

[0150] 1)打开应用

[0151] onCreate()方法执行!

[0152] onStart()方法执行!

[0153] onResume()方法执行!

[0154] 2)按下主屏幕键/锁屏

[0155] onPause()方法执行!

[0156] onStop()方法执行!

[0157] 3)重新打开应用

[0158] onRestart()方法执行!

[0159] onStart()方法执行!

[0160] onResume()方法执行!

[0161] 4)按下后退键

[0162] onPause()方法执行!

[0163] onStop()方法执行!

[0164] onDestroy()方法执行!

[0165] (二)Fragment

[0166] 1)界面打开

[0167] onCreate()方法执行!

[0168] onCreateView()方法执行!

[0169] onActivityCreated()方法执行!

[0170] onStart()方法执行!

[0171] onResume()方法执行!

[0172] 2)按下主屏幕键/锁屏

[0173] onPause()方法执行!

[0174] onStop()方法执行!

[0175] 3)重新打开

[0176] onStart()方法执行!

[0177] onResume()方法执行!

[0178] 4)按下后退键

[0179] onPause()方法执行!

[0180] onStop()方法执行!

[0181] onDestroyView()方法执行!

[0182] onDestroy()方法执行!

[0183] onDetach()方法执行!

[0184] (三)在Activity中加入Fragment,对应的生命周期

[0185] 1)打开

[0186] Fragment onCreate()方法执行!

[0187] Fragment onCreateView()方法执行!

[0188] Activity onCreate()方法执行!

[0189] Fragment onActivityCreated()方法执行!

[0190] Activity onStart()方法执行!

[0191] Fragment onStart()方法执行!

[0192] Activity onResume()方法执行!

[0193] Fragment onResume()方法执行!

[0194] 2)按下主屏幕键/锁屏

[0195] Fragment onPause()方法执行!

[0196] Activity onPause()方法执行!

[0197] Fragment onStop()方法执行!

[0198] Activity onStop()方法执行!

[0199] 3)再次打开

[0200] Activity onRestart()方法执行!

[0201] Activity onStart()方法执行!

[0202] Fragment onStart()方法执行!

[0203] Activity onResume()方法执行!

[0204] Fragment onResume()方法执行!

[0205] 4)按下后退键

[0206] Fragment onPause()方法执行!

[0207] Activity onPause()方法执行!

[0208] Fragment onStop()方法执行!

[0209] Activity onStop()方法执行!

[0210] Fragment onDestroyView()方法执行!

[0211] Fragment onDestroy()方法执行!

[0212] Fragment onDetach()方法执行!

[0213] Activity onDestroy()方法执行!

[0214] 可以看出,Activity的FragmentManager负责调用队列中Fragment的生命周期方法,添加Fragment供FragmentManager管理时,onAttach(Activity)、onCreat(Bundle)以及onCreatView()方法会被调用。

[0215] 托管Activity的onCreate()方法执行后,onActivityCreated()方法也会被调用。因为正在向Activity的onCreat()方法中添加Fragment,所以Fragment被添加后,该方法会被调用。

[0216] 在Activity处于停止、暂停或运行状态下时,添加Fragment会发生什么呢?此种情况下,FragmentManager立即驱使Fragment快速跟上Activity的步伐,直到与Activity的最新状态保持同步。例如,向运行状态的Activity中添加Fragment时,以下Fragment生命周期方法会被依次调用:onAttch()、onCreat()、onCreatView()、onActivityCreated()、

onStart()，以及onResume()方法。

[0217] 只要Fragment的状态与Activity的状态保持了同步，托管Activity的FragmentManager便会继续调用其他生命周期方法以继续保持Fragment与Activity的状态一致，而几乎同时，它接收到了从操作系统发出的相应调用。但Fragment方法究竟是在Activity方法之前还是之后调用的这一点是无法保证的。

[0218] Fragment生命周期与Activity生命周期的一个关键区别就在于，Fragment的生命周期方法是由托管Activity而不是操作系统调用的。操作系统无从知晓Activity用来管理视图的Fragment。Fragment的使用是Activity自己内部的事情。可以发现，Activity中的生命周期方法都是私有的，而Fragment的则是公开的，这就是因为Fragment是由Activity来管理的，Activity需要调用这些方法。

[0219] 参考图3所示的Fragment与Activity生命周期的对比示意图。

[0220] 在创建的过程中，由于是Activity带领Fragment执行生命周期中的各个方法，所以生命周期方法的执行顺序是这样的：

[0221] 1.Activity--onCreate()；

[0222] 2.Fragment--onAttach()；

[0223] 3.Fragment--onCreate()；

[0224] 4.Fragment--onCreateView()；

[0225] 5.Fragment--onActivityCreated()；

[0226] 接着是这样的：

[0227] 6.Activity--onStart()；

[0228] 7.Fragment--onStart()；

[0229] 8.Activity--onResume()；

[0230] 9.Fragment--onResume()；

[0231] 无论对于Activity还是对于Fragment，onResume这个生命周期都是他们执行时间最长的，当Activity或者Fragment打开之后，它就一直处于这个生命周期中。

[0232] 当销毁的时候，是Fragment先感知到，于是销毁的时候就是Fragment带领Activity执行如下的各个方法：

[0233] 10.Fragment--onPause()；

[0234] 11.Activity--onPause()；

[0235] 12.Fragment--onStop()；

[0236] 13.Activity--onStop()；

[0237] 14.Fragment--onDestroyView()；

[0238] 15.Fragment--onDestroy()；

[0239] 16.Fragment--onDetach()；

[0240] 17.Activity--onDestroy()；

[0241] 上面这个顺序有一个前提，就是所有的日志打印代码都是紧挨着super()方法来写的。因为如果把Fragment写在了布局文件中，同时又在Activity的onCreate()方法中的setContentView之后打印日志，那么看到的生命周期的执行顺序就会有所不同，不过只是细微的差别。总之，在创建的过程中，是Activity带领着Fragment来进行，而在销毁的过程

中,是Fragment带领着Activity来进行的。

[0242] 以下通过几个具体示例来说明Activity与Fragment的一般开发过程。

[0243] 如图4A-4C所示,在会员+页面中有一个卡包组件,而卡包最基本的状态就是展开和收起。其中,图4A是卡包收起状态的示意图,图4B是卡包展开且卡片未展开状态的示意图,而图4C则是卡包展开且卡片展开状态的示意图。

[0244] 在点击卡包中的卡片时,卡包要依次自动展开如图4A→4B→4C的状态。卡包收起时卡片要依次自动收起如图4C→4B→4A的状态。

[0245] 示例一:

[0246] 基于现有的Activity与Fragment处理机制,图4A→4B→4C的实现涉及如下处理过程:

[0247] 卡包(CardBag),卡片列表(CardBagList),卡片(CardItem)都在Activity中创建,这3个是普通的业务类,卡片(CardItem)是卡片列表(CardBagList)的内部类。

[0248] 相应的执行流程是:点击卡片——卡片响应点击——读取卡包展开状态(CardBag.isPopup)——若卡包未展开则进行展开(CardBag.popup())——卡片列表展开

[0249] CardBagList作为卡片列表,聚合了卡片CardItem,在CardItem中监听点击事件onClick,点击时需要首先读取卡包CardBag中的状态是否已经展开CardBag.isPopup,然后再调用卡包的展开方法CardBag.popup()。由于在卡片响应点击时需要直接读取卡包的状态以及执行卡包的展开动作,所以在这个状态传递、方法调用中就将CardItem、CardBagList和CardBag耦合起来了,CardItem、CardBagList和CardBag无法单独运行和使用。

[0250] 基于现有的Activity与Fragment处理机制,图4C→4B→4A的实现涉及如下处理过程:

[0251] 用户点击收起卡包CardBag,卡包内部的onClick()方法被触发,首先调用CardBag.packup()方法和CardBagList.isListPopup()方法,然后再调用CardBagList.packupList()方法收起卡片,所以在这个过程中,CardBagList和CardBag是强耦合在一起的。

[0252] 示例二:

[0253] 在页面切换到后台后,要恢复为卡包收起状态,即图4B、图4C→4A。

[0254] 基于现有的Activity与Fragment处理机制,图4B、图4C→4A的实现涉及如下处理过程:

[0255] 在该Activity的生命周期方法onpause()中调用CardBag.isPopup()、CardBag.packup()、CardBagList.isListPopup()、CardBagList.packupList()。随着组件越来越多,这样将会有大量业务逻辑代码堆叠在onPause()中,Activity将因为有大量业务逻辑代码堆叠而变的臃肿难以维护。并且,由于业务组件之间不相关,可移植性和可维护性都将大大降低。

[0256] 示例三:

[0257] 基于现有的Activity与Fragment处理机制,在业务变化时,新建其他页面复用组件涉及如下处理过程:

[0258] 新建一个页面,将CardBagList拿过来放到Activity页面中,但是由于

CardBagList和CardBag强引用,需要将和CardBag相关的逻辑代码全部移除才能通过编译,同时还需要在新的页面Activity的onPause()方法中添加CardBagList.isListPopup()调用以及CardBagList.packupList()调用,移植成本较高,且容易功能丢失。

[0259] 因此,提出了本申请实施例的基于Android系统的开发系统的核心构思在于,在Android系统下,采用模块化编程解决方案,专注于实现大规模应用的组件化,降低编程复杂度以及依赖耦合度,使得组件之间互不依赖,组件的灵活度也更高,进而应用程序可以轻松地实现管理、维护、复用组件等功能。

[0260] 参照图5,示出了本申请的一种基于Android系统的开发系统的系统架构示意图,在Android系统中包括有业务组件,任一业务组件均具有对应的生命周期。

[0261] 在本申请实施例中,开发系统中设置有组件管理单元Controller,Controller是对应的业务组件的具体实现,可以用于管理相应的业务组件。例如,对于卡包,其组件管理单元(CardBagController)是卡包组件可复用的最小单元。

[0262] 在本申请实施例中,当业务组件被注册到Controller中,可以使得Controller具有与注册后的该业务组件的生命周期相适配的相关属性。

[0263] 在本申请实施例中,业务组件可以实现多种不同的功能。例如,基础功能和业务功能。

[0264] 业务组件的基础功能可以包括消息注册、消息反注册、发送局部消息,或,发送全局消息等等,基础功能是业务组件必须要实现的功能。而业务功能则可以包括实例化业务组件、增加业务组件、修改业务组件、删除业务组件,或,查找业务组件中的任意一种或多种,

[0265] 在本申请实施例中,组件管理单元Controller可以包括组件管理子单元ControllerManager和基础管理子单元BaseController。

[0266] 其中,ControllerManager可以用于管理业务组件的业务功能,即管理实例化业务组件、增加业务组件、修改业务组件、删除业务组件,或,查找业务组件中的任意一种或多种功能。而BaseController则可以用于实现及管理业务组件的上述基础功能。

[0267] 在本申请实施例中,业务组件可以包括Activity组件和Fragment组件。其中,Activity组件可以单独存在被实现相应的功能,而Fragment组件则只能被嵌入到Activity组件中,才能够通过执行Fragment组件的生命周期回调方法以实现相应的功能。

[0268] 因此,在本申请实施例中,组件管理单元Controller还可以包括Activity组件实现子单元BaseActivityController和Fragment组件实现子单元BaseFragmentController。

[0269] BaseActivityController继承于BaseController,可以用于实现Activity组件对应的生命周期接口,以调用上述Activity组件生命周期接口中的方法;而BaseFragmentController同样继承于BaseController,则可以用于实现Fragment组件对应的生命周期接口,以调用上述Fragment组件生命周期接口中的方法。

[0270] 在本申请实施例中,开发系统还可以包括组件管理创建单元ControllerCreator、Activity组件管理创建单元BaseActivityControllerCreator,以及,Fragment组件管理创建单元BaseFragmentControllerCreator。

[0271] ControllerCreator可以在业务组件注册时,创建与该业务组件对应的Controller,并由所创建的Controller对该业务组件进行管理。

BaseActivityControllerCreator则可以用于创建BaseActivityController;而BaseFragmentControllerCreator则用于创建BaseFragmentController。

[0272] 在本申请实施例中,开发系统还可以包括生命周期管理单元LifeCycle。LifeCycle可以提供业务组件的生命周期接口,并将该业务组件的生命周期分发到对应的Controller中。

[0273] 具体地,提供业务组件的生命周期接口的功能可以通过生命周期管理子单元实现,而将业务组件的生命周期分发到对应的Controller中则可以通过生命周期分发子单元实现。即,LifeCycle可以包括生命周期管理子单元和生命周期分发子单元。

[0274] 由于业务组件可以分为Activity组件和Fragment组件,为了实现分别对Activity组件和Fragment组件进行生命周期管理,生命周期管理子单元可以进一步包括Activity生命周期管理子单元和Fragment生命周期管理子单元。前者可以用于提供Activity组件的生命周期接口,而后者则用于提供Fragment组件的生命周期接口。

[0275] 相应地,为了实现与Activity生命周期管理子单元和Fragment生命周期管理子单元对接,生命周期分发子单元也可以进一步包括Activity生命周期分发子单元和Fragment生命周期分发子单元。前者可以用于将Activity组件的生命周期分发到对应的Controller中,而后者则用于将Fragment组件的生命周期分发到对应的Controller中。

[0276] 例如,LifeCycle中的Activity生命周期管理子单元提供Activity组件的生命周期接口后,可以通过Activity生命周期分发子单元将该Activity组件的生命周期分发到对应的Controller中;而Fragment生命周期管理子单元提供Fragment组件的生命周期接口后,则是通过Fragment生命周期分发子单元将该Fragment组件的生命周期分发到对应的Controller中,使得Activity组件和Fragment组件之间生命周期管理互不影响。

[0277] 在本申请实施例中,开发系统还可以包括消息管理单元Messager,用于处理Controller之间的通信。即,通过Messager直接负责各个业务组件内部或不同业务组件之间的消息的发送和接收等过程,使得Controller之间不再直接依赖,减少各个业务组件之间的耦合。

[0278] 在本申请实施例中,Messager可以包括局部消息管理子单元LocalMessager以及全局消息管理子单元MessageManager。

[0279] 在实际应用中,LocalMessager可以与任一Activity组件或Fragment组件绑定,当某一Activity组件或Fragment组件与一个LocalMessager绑定后,该LocalMessager可以管理被绑定的Activity组件或Fragment组件对应的Controller中的消息。

[0280] 需要说明的是,MessageManager则只能与Activity组件绑定,当某一Activity组件与一个MessageManager绑定后,该MessageManager可以管理被绑定的Activity组件中的消息。

[0281] 由于Fragment组件只有被嵌入到Activity组件中才能够实现相应的功能,而Activity组件则可以独立存在。因此,当一个Activity组件种嵌入有Fragment组件时,与该Activity组件绑定的MessageManager不仅可以管理该Activity组件中的消息,也可以管理嵌入该该Activity组件中的其他Fragment组件中的消息。

[0282] 在本申请实施例中,MessageManager可以接收LocalMessager发送的消息,也可以将消息发送至LocalMessager。因此,MessageManager可以实现不同业务组件对应的

LocalMessager之间的消息的传递。即,MessageManager在接收到任一业务组件发送的消息后,可以在确定出这一消息所要传递至的其他业务组件后,可以将接收到的消息转发至其他业务组件对应的LocalMessager。

[0283] 在本申请实施例中,为了实现同一Activity组件中,LocalMessager与MessageManager之间的消息的传递,消息管理单元Messager还提供了消息管理回调子单元MessageManagerListener。MessageManagerListener可以用于将LocalMessager注册到MessageManager中,并在完成注册后,实现MessageManager对LocalMessager的管理。

[0284] 需要说明的是,本实施例中的开发系统是基于Android系统的,为了实现与Android系统的对接,开发系统还提供了代理对接单元,可以用于实现与Android系统中业务组件的对接。

[0285] 具体地,该代理对接单元包括Activity代理对接单元GrandPiano和Fragment代理对接单元GrandPianoFgt。通过GrandPiano,实现与Android系统中的Activity组件的对接,通过GrandPianoFgt,实现与Android系统中的Fragment组件的对接。

[0286] 在本申请实施例中,开发系统中还可以包括一辅助日志输出单元,用于输出相应的日志。

[0287] 在本申请实施例中,提供了一种基于Android系统的开发系统,Android系统中包括业务组件,该业务组件具有生命周期,该开发系统中设置有组件管理单元Controller,当业务组件被注册到Controller中,该Controller具有与生命周期相适配的相关属性;本实施例提供的开发系统还包括生命周期管理单元LifeCycle和消息管理单元Messager。LifeCycle可以用于提供业务组件的生命周期接口并将该业务组件的生命周期分发到对应的Controller中,Messager可以用于处理Controller之间的通信。首先,本实施例提供的开发系统是一种全新的Android项目架构形式,通过采用模块化编程解决方案,可以实现大规模应用的组件化,降低了编程复杂度以及依赖耦合度,使得业务组件之间互不依赖,从而业务组件的灵活度也更高,进而应用程序也可以轻松地实现管理、维护、复用组件等功能。

[0288] 其次,本实施例提供的开发系统可以将Controller作为容器进而接管Activity组件和Fragment组件,通过Messager实现Controller之间的通信,使得Controller之间可以相互通信但不再直接依赖,从而达到Controller之间完全解耦的状态,无论对于Activity组件或Fragment组件,其复用、维护及可读性都能得到有效保障。

[0289] 参照图6,示出了本申请的一种基于Android系统的解耦方法实施例一的步骤流程图。所述Android系统中包括业务组件,所述业务组件具有生命周期;所述Android系统中还包括一开发系统,所述开发系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager,所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager,当所述业务组件被注册到Controller中,所述Controller具有与所述生命周期相适配的相关属性;所述方法具体可以包括如下步骤:

[0290] 步骤601,当接收到针对业务对象的点击事件时,所述业务对象对应的消息管理单元确定目标业务对象;

[0291] 在本申请实施例中,业务对象由业务组件生成。例如,对于如图4A-4C所示的会员+页面中的卡包组件,其卡包、卡片列表及卡片可以分别看作是一个业务对象。

[0292] 在本申请实施例中，业务对象对应的组件管理单元可以首先注册到消息管理单元，从而在后续操作中，可以通过消息管理单元对操作过程中形成的消息进行管理。

[0293] 在本申请实施例中，当某个业务对象接收到用户的点击事件时，该业务对象对应的组件管理单元可以针对当前的点击事件生成事件消息，并将该事件消息发送至消息管理单元；然后，消息管理单元可以依据接收到的事件消息，确定目标业务对象。

[0294] 例如，对于图4A所示的卡包，当用户点击其中某张卡片时，该卡片对应的组件管理单元可以生成相应的事件消息，并将该时间消息发送至该卡片的消息管理单元，然后，由消息管理单元确定需要将该事件消息发送至何处。

[0295] 步骤602，所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息；

[0296] 在本申请实施例中，目标业务对象可以由目标业务组件生成，目标业务对象具有对应的目标消息管理单元。

[0297] 需要说明的是，目标业务组件与业务组件类似，目标消息管理单元与消息管理单元类似，使用“目标”一词仅仅为了区别不同的业务对象。

[0298] 在本申请实施例中，业务对象对应的消息管理单元可以向目标业务对象对应的目标消息管理单元发送针对点击事件生成的事件消息。

[0299] 在具体实现中，消息管理单元可以首先判断目标业务对象对应的业务组件与当前的业务对象对应的业务组件是否位于同一Activity组件中。

[0300] 如果二者位于同一Activity组件中，则当前的业务对象对应的消息管理单元可以将该事件消息发送至该Activity组件对应的全局消息管理单元，然后由全局消息管理单元将该事件消息转发至目标业务对象对应的目标消息管理单元。

[0301] 如果二者并不位于同一Activity组件中，则当前的业务对象对应的消息管理单元可以将该事件消息发送至该Activity组件对应的全局消息管理单元，然后由全局消息管理单元将该事件消息转发至目标业务对象对应的目标全局消息管理单元，进而目标全局消息管理单元可以将接收到的事件消息分发至目标业务对象对应的目标消息管理单元。

[0302] 例如，在图4A所示的卡包中，当用户点击其中某张卡片后，可以相应执行展开卡片列表，进而展开相应的卡片的操作。在点击卡片后，该卡片对应的消息管理单元可以首先判断卡片列表对应的业务组件与卡片对应的业务组件是否位于同一Activity组件中。

[0303] 若二者位于同一Activity组件中，则卡片对应的局部消息管理单元与卡片列表对应的局部消息管理单元可以通过与该Activity组件绑定的全局消息管理单元进行消息之间的相互传递，从而卡片对应的局部消息管理单元可以将针对点击事件生成的事件消息发送至与该Activity组件绑定的全局消息管理单元，并由全局消息管理单元将该事件消息转发至卡片列表对应的局部消息管理单元。

[0304] 若二者并不位于同一Activity组件中，则卡片对应的局部消息管理单元需要首先将针对点击事件生成的事件消息发送至卡片所在的Activity组件对应的全局消息管理单元，然后全局消息管理单元需要将该事件消息转发至卡片列表所在的Activity组件对应的全局消息管理单元，从而卡片列表所在的Activity组件对应的全局消息管理单元可以将该事件消息分发至卡片列表对应的局部消息管理单元。

[0305] 步骤603，所述目标业务对象依据所述事件消息，执行相应的操作。

[0306] 在本申请实施例中，接收到事件消息的目标消息管理单元可以首先向目标组件管理单元转发该事件消息，然后目标组件管理单元可以依据接收到的事件消息，控制目标业务对象执行相应操作。

[0307] 例如，卡片列表对应的消息管理单元接收到事件消息后，可以将该事件消息转发至管理该卡片列表的组件管理单元，从而组件管理单元在接收到事件消息后，可以展开卡片列表。

[0308] 参照图7，示出了本申请的一种基于Android系统的解耦方法实施例二的步骤流程图，Android系统中可以设置有消息管理单元Messager；所述方法具体可以包括如下步骤：

[0309] 步骤701，当消息管理单元接收到针对业务对象的操作事件时，所述消息管理单元确定目标业务对象；

[0310] 通常，针对业务对象的操作事件可以在由某个业务组件生成的视图上的操作。例如，点击、滑动等等。

[0311] 当用户在业务对象上进行操作时，生成该业务对象的业务组件对应的消息管理单元能够接收到相应的消息，并基于该消息确定出目标业务对象。

[0312] 步骤702，所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息；

[0313] 在本申请实施例中，可以通过消息管理单元处理各个业务对象之间的消息。因此，在产生上述消息后，消息管理单元需要将该消息传递至目标业务对象。

[0314] 步骤703，所述目标业务对象依据所述事件消息，执行相应的操作。

[0315] 在本申请实施例中，目标业务对象对应的业务组件在接收到上述消息后，可以控制目标业务对象实现相应的操作。

[0316] 本实施例中步骤701-步骤703与实施例一中步骤601-步骤603较为相似，可以互相参阅，本实施例对此不再赘述。

[0317] 为了便于理解，以下通过几个具体的示例来说明基于本申请的开发系统的Activity与Fragment的开发过程。

[0318] 针对如图4A-4C所示的会员+页面中的卡包组件，在点击卡包中的卡片时，卡包要依次自动展开如图4A→4B→4C的状态。卡包收起时卡片要依次自动收起如图4C→4B→4A的状态。

[0319] 示例一：

[0320] 基于本申请的开发系统的Activity与Fragment的处理机制，图4A→4B→4C的实现涉及如下处理过程：

[0321] 卡包(CardBag)继承于BaseActivityController，属于Controller；卡片列表(CardBagList)继承于BaseActivityController，属于Controller，卡片是卡片列表的内部类。

[0322] SwitchCardBagEvent属于消息机制中的一种消息类型，该消息可以由任何Controller发送，根据发送方式不同可以是局部消息(LocalMessage)或者全局消息(GloabMessage)，该消息将被卡包接收并响应，该消息可以控制卡包状态变化(展开、收起)。

[0323] 相应的执行流程是：点击卡片——卡片响应点击——实例化SwitchCardBagEvent

消息(将控制卡包为展开状态)——卡片将消息发送到卡包中——卡包接收消息且响应——卡片列表展开

[0324] 由于在卡片响应点击时发送消息,通过消息传递到卡包,卡包接收到消息后进行响应,所以卡包和卡片并不耦合,可以单独运行和使用。

[0325] 示例二:

[0326] 在页面切换到后台后,要恢复为卡包收起状态,即图4B、图4C→4A。

[0327] 基于本申请的开发系统的Activity与Fragment的处理机制,图4B、图4C→4A的实现涉及如下处理过程:

[0328] CardBag继承于BaseController,由于BaseController实现了对应的lifecycle接口,所以GrandPiano会将onPause()生命周期方法分发到对应的LifeCycleDispatcher,然后会分发到BaseController中,所以仅需要在CardBag内部实现onPause()方法中调用this.isPopup()、this.packup(),CardBagList的处理过程同上。

[0329] 这样,可以将生命周期触发的逻辑分散到各个组件内部自行维护,互不干扰,Activity也不会有大量业务逻辑代码堆叠而变的臃肿难维护。

[0330] 示例三:

[0331] 基于本申请的开发系统的Activity与Fragment的处理机制,在业务变化时,新建其他页面复用组件涉及如下处理过程:

[0332] 在getControllers()方法中注册相应组件;

[0333] 在原始页面中组件的注册代码为:

[0334] controllersList.add(CardBag.class);

[0335] controllersList.add(CardBagList.class);

[0336] 而在新页面中组件组册代码为:

[0337] controllersList.add(CardBagList.class);

[0338] 因此,无须做其他逻辑上的修改,功能完全不丢失,移植成本极低,添加极少代码即可实现。

[0339] 参照图8,示出了本申请的一种基于Android系统的解耦装置实施例一的结构框图。所述Android系统中包括业务组件,所述业务组件具有生命周期;所述Android系统中还包括一开发系统,所述开发系统中设置有组件管理单元Controller、生命周期管理单元LifeCycle和消息管理单元Messager,所述消息管理单元Messager包括局部消息管理单元LocalMessager和全局消息管理单元MessageManager,当所述业务组件被注册到Controller中,所述Controller具有与所述生命周期相适配的相关属性;所述装置具体可以包括如下模块:

[0340] 确定模块801,用于当接收到针对业务对象的点击事件时,采用所述业务对象对应的消息管理单元确定目标业务对象;

[0341] 发送模块802,用于采用所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息;

[0342] 执行模块803,用于采用所述目标业务对象,依据所述事件消息,执行相应的操作。

[0343] 在本申请实施例中,所述业务对象由所述业务组件生成,所述当接收到针对业务对象的点击事件时,所述确定模块801具体可以包括如下子模块:

[0344] 生成发送子模块，用于当接收到针对业务对象的点击事件时，采用所述业务对象对应的组件管理单元针对所述点击事件生成事件消息，并将所述事件消息发送至所述消息管理单元；

[0345] 确定子模块，用于采用所述消息管理单元，依据所述事件消息，确定目标业务对象。

[0346] 在本申请实施例中，所述装置还可以包括如下模块：

[0347] 注册模块，用于将所述组件管理单元注册到所述消息管理单元。

[0348] 在本申请实施例中，所述目标业务对象由目标业务组件生成，所述发送模块802具体可以包括如下子模块：

[0349] 发送子模块，用于采用所述消息管理单元向所述目标业务对象对应的目标消息管理单元发送针对所述点击事件生成的事件消息。

[0350] 在本申请实施例中，所述发送子模块具体可以包括如下单元：

[0351] 判断单元，用于采用所述消息管理单元判断目标业务对象对应的业务组件与当前的业务对象对应的业务组件是否位于同一Activity组件中；

[0352] 第一发送单元，用于若是，则由所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元，由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标消息管理单元；

[0353] 第二发送单元，用于若否，则由所述业务对象对应的消息管理单元将所述事件消息发送至所述Activity组件对应的全局消息管理单元，由所述全局消息管理单元将所述事件消息转发至目标业务对象对应的目标全局消息管理单元，所述目标全局消息管理单元用于将所述事件消息分发至目标业务对象对应的目标消息管理单元。

[0354] 在本申请实施例中，所述目标业务对象还具有对应的目标组件管理单元，所述执行模块803具体可以包括如下子模块：

[0355] 转发子模块，用于采用所述目标消息管理单元向所述目标组件管理单元转发所述事件消息；

[0356] 执行子模块，用于采用所述目标组件管理单元，依据所述事件消息控制所述目标业务对象执行相应操作。

[0357] 参照图9，示出了本申请的一种基于Android系统的解耦装置实施例二的结构框图，Android系统设置有消息管理单元Messager，所述装置具体可以包括如下模块：

[0358] 目标业务对象确定模块901，用于当消息管理单元接收到针对业务对象的操作事件时，所述消息管理单元确定目标业务对象；

[0359] 事件消息发送模块902，用于所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息；

[0360] 执行模块903，用于所述目标业务对象依据所述事件消息，执行相应的操作。

[0361] 对于装置实施例而言，由于其与方法实施例基本相似，所以描述的比较简单，相关之处参见方法实施例的部分说明即可。

[0362] 本申请的实施例可被实现为使用任意适当的硬件，固件，软件，或及其任意组合进行想要的配置的系统。图10示意性地示出了可被用于实现本申请中所述的各个实施例的示例性装置(或系统)100。

[0363] 对于一个实施例，图10示出了示例性装置100，该装置具有一个或多个处理器1002、被耦合到(一个或多个)处理器1002中的至少一个的系统控制模块(芯片组)1004、被耦合到系统控制模块1004的系统存储器1006、被耦合到系统控制模块1004的非易失性存储器(NVM)/存储设备1008、被耦合到系统控制模块1004的一个或多个输入/输出设备1010，以及被耦合到系统控制模块1006的网络接口1012。

[0364] 处理器1002可包括一个或多个单核或多核处理器，处理器1002可包括通用处理器或专用处理器(例如图形处理器、应用处理器、基频处理器等)的任意组合。

[0365] 在一些实施例中，系统100可包括具有指令的一个或多个计算机可读介质(例如，系统存储器1006或NVM/存储设备1008)以及与该一个或多个计算机可读介质相合并被配置为执行指令以实现模块从而执行本申请中所述的动作的一个或多个处理器1002。

[0366] 对于一个实施例，系统控制模块1004可包括任意适当的接口控制器，以向(一个或多个)处理器1002中的至少一个和/或与系统控制模块404通信的任意适当的设备或组件提供任意适当的接口。

[0367] 系统控制模块1004可包括存储器控制器模块，以向系统存储器1006提供接口。存储器控制器模块可以是硬件模块、软件模块和/或固件模块。

[0368] 系统存储器1006可被用于例如为系统100加载和存储数据和/或指令。对于一个实施例，系统存储器1006可包括任意适当的易失性存储器，例如，适当的DRAM。在一些实施例中，系统存储器1006可包括双倍数据速率类型四同步动态随机存取存储器(DDR4SDRAM)。

[0369] 对于一个实施例，系统控制模块1004可包括一个或多个输入/输出控制器，以向NVM/存储设备1008及(一个或多个)输入/输出设备1010提供接口。

[0370] 例如，NVM/存储设备1008可被用于存储数据和/或指令。NVM/存储设备1008可包括任意适当的非易失性存储器(例如，闪存)和/或可包括任意适当的(一个或多个)非易失性存储设备(例如，一个或多个硬盘驱动器(HDD)、一个或多个光盘(CD)驱动器和/或一个或多个数字通用光盘(DVD)驱动器)。

[0371] NVM/存储设备1008可包括在物理上作为系统100被安装在其上的设备的一部分的存储资源，或者其可被该设备访问而不必作为该设备的一部分。例如，NVM/存储设备1008可通过网络经由(一个或多个)输入/输出设备1010进行访问。

[0372] (一个或多个)输入/输出设备1010可为系统100提供接口以与任意其他适当的设备通信，输入/输出设备1010可以包括通信组件、音频组件、传感器组件等。网络接口1012可为系统100提供接口以通过一个或多个网络通信，系统100可根据一个或多个无线网络标准和/或协议中的任意标准和/或协议来与无线网络的一个或多个组件进行无线通信，例如接入基于通信标准的无线网络，如WiFi，2G或3G，或它们的组合进行无线通信。

[0373] 对于一个实施例，(一个或多个)处理器1002中的至少一个可与系统控制模块1004的一个或多个控制器(例如，存储器控制器模块)的逻辑封装在一起。对于一个实施例，(一个或多个)处理器1002中的至少一个可与系统控制模块1004的一个或多个控制器的逻辑封装在一起以形成系统级封装(SiP)。对于一个实施例，(一个或多个)处理器1002中的至少一个可与系统控制模块1004的一个或多个控制器的逻辑集成在同一模具上。对于一个实施例，(一个或多个)处理器1002中的至少一个可与系统控制模块1004的一个或多个控制器的逻辑集成在同一模具上以形成片上系统(SoC)。

[0374] 在各个实施例中，系统100可以但不限于是：工作站、台式计算设备或移动计算设备(例如，膝上型计算设备、手持计算设备、平板电脑、上网本等)。在各个实施例中，系统1000可具有更多或更少的组件和/或不同的架构。例如，在一些实施例中，系统100包括一个或多个摄像机、键盘、液晶显示器(LCD)屏幕(包括触屏显示器)、非易失性存储器端口、多个天线、图形芯片、专用集成电路(ASIC)和扬声器。

[0375] 本申请实施例还提供了一种非易失性可读存储介质，该存储介质中存储有一个或多个模块(programs)，该一个或多个模块被应用在终端设备时，可以使得该终端设备执行本申请实施例中各方法步骤的指令(instructions)。

[0376] 在本申请的一个示例中，提供了一种装置，包括一个或多个处理器；和，其上存储的有指令的一个或多个机器可读介质，当由所述一个或多个处理器执行时，使得所述装置执行如本申请实施例中的方法。

[0377] 在本申请的一个示例中还提供了一个或多个机器可读介质，其上存储有指令，当由一个或多个处理器执行时，使得装置执行如本申请实施例中的方法。

[0378] 本说明书中的各个实施例均采用递进的方式描述，每个实施例重点说明的都是与其他实施例的不同之处，各个实施例之间相同相似的部分互相参见即可。

[0379] 本领域内的技术人员应明白，本申请实施例的实施例可提供为方法、装置、或计算机程序产品。因此，本申请实施例可采用完全硬件实施例、完全软件实施例、或结合软件和硬件方面的实施例的形式。而且，本申请实施例可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质(包括但不限于磁盘存储器、CD-ROM、光学存储器等)上实施的计算机程序产品的形式。

[0380] 在一个典型的配置中，所述计算机设备包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。内存可能包括计算机可读介质中的非永久性存储器，随机存取存储器(RAM)和/或非易失性内存等形式，如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括，但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光盘只读存储器(CD-ROM)、数字多功能光盘(DVD)或其他光学存储、磁盒式磁带，磁带磁磁盘存储或其他磁性存储设备或任何其他非传输介质，可用于存储可以被计算设备访问的信息。按照本文中的界定，计算机可读介质不包括非持续性的电脑可读媒体(transitory media)，如调制的数据信号和载波。

[0381] 本申请实施例是参照根据本申请实施例的方法、终端设备(系统)、和计算机程序产品的流程图和/或方框图来描述的。应理解可由计算机程序指令实现流程图和/或方框图中的每一流程和/或方框、以及流程图和/或方框图中的流程和/或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理终端设备的处理器以产生一个机器，使得通过计算机或其他可编程数据处理终端设备的处理器执行的指令产生用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的装置。

[0382] 这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理终端设备以特定方式工作的计算机可读存储器中，使得存储在该计算机可读存储器中的指令产生包括指令装置的制造品，该指令装置实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能。

[0383] 这些计算机程序指令也可装载到计算机或其他可编程数据处理终端设备上，使得在计算机或其他可编程终端设备上执行一系列操作步骤以产生计算机实现的处理，从而在计算机或其他可编程终端设备上执行的指令提供用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的步骤。

[0384] 尽管已描述了本申请实施例的优选实施例，但本领域内的技术人员一旦得知了基本创造性概念，则可对这些实施例做出另外的变更和修改。所以，所附权利要求意欲解释为包括优选实施例以及落入本申请实施例范围的所有变更和修改。

[0385] 最后，还需要说明的是，在本文中，诸如第一和第二等之类的关系术语仅仅用来将一个实体或者操作与另一个实体或操作区分开来，而不一定要求或者暗示这些实体或操作之间存在任何这种实际的关系或者顺序。而且，术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、物品或者终端设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、物品或者终端设备所固有的要素。在没有更多限制的情况下，由语句“包括一个……”限定的要素，并不排除在包括所述要素的过程、方法、物品或者终端设备中还存在另外的相同要素。

[0386] 以上对本申请所提供的一种基于Android系统的开发系统、一种基于Android系统的解耦方法、一种基于Android系统的解耦装置，和一个或多个计算机可读介质，进行了详细介绍，本文中应用了具体个例对本申请的原理及实施方式进行了阐述，以上实施例的说明只是用于帮助理解本申请的方法及其核心思想；同时，对于本领域的一般技术人员，依据本申请的思想，在具体实施方式及应用范围上均会有改变之处，综上所述，本说明书内容不应理解为对本申请的限制。

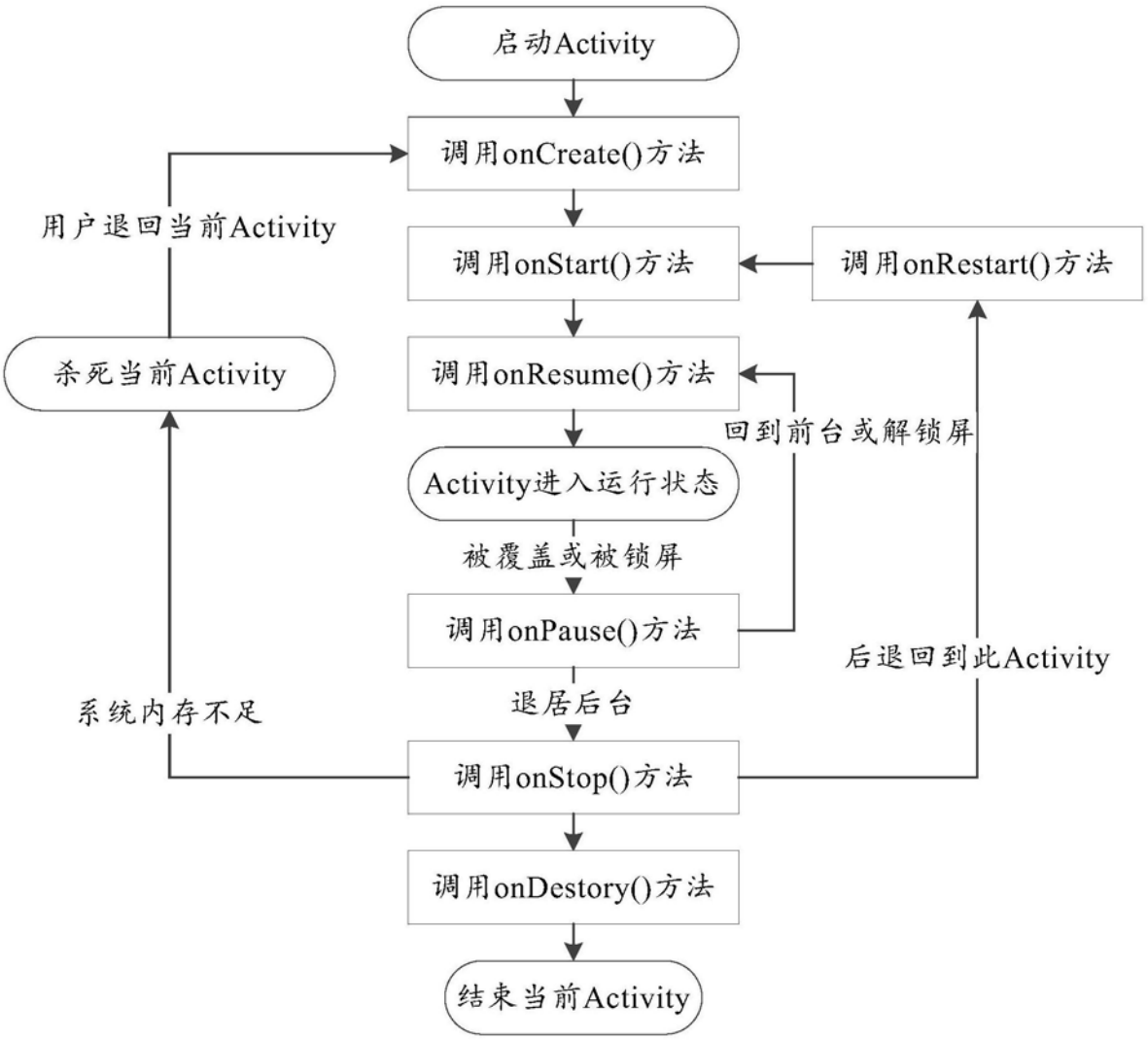
启动Activity
调用onCreate()方法
调用onStart()方法
调用onRestart()方法
调用onResume()方法
Activity进入运行状态
调用onPause()方法
调用onStop()方法
调用onDestory()方法
结束当前Activity
杀死当前Activity
用户退回当前Activity
回到前台或解锁屏
被覆盖或被锁屏
后退回到此Activity
退居后台
系统内存不足

图1

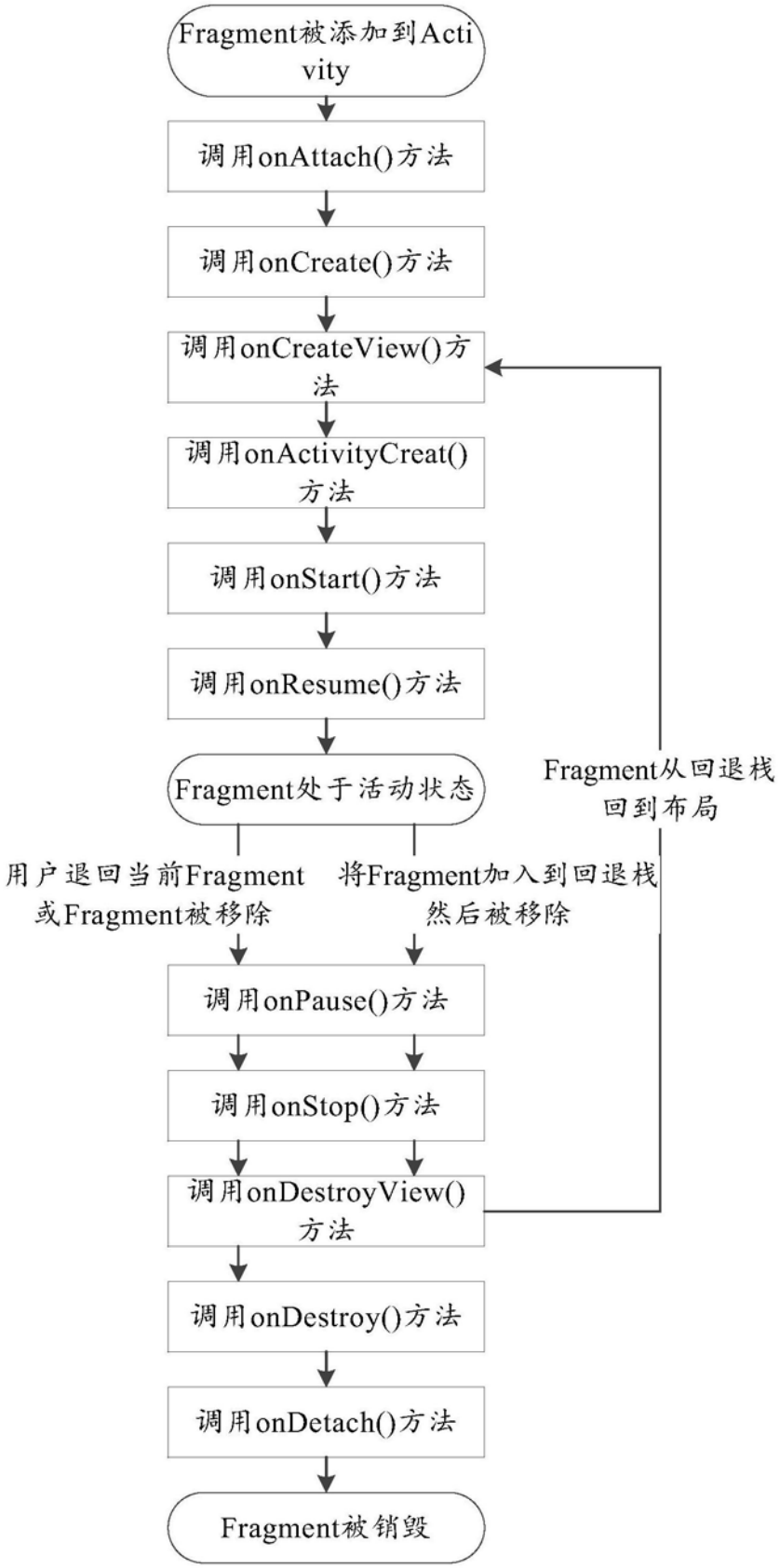
Fragment被添加到Activity
调用onAttach()方法
调用onCreate()方法
调用onCreateView()方法
调用onActivityCreat()方法
调用onStart()方法
调用onResume()方法
Fragment处于活动状态
Fragment从回退栈回到布局
用户退回当前Fragment或Fragment被移除
将Fragment加入到回退栈然后被移除
调用onPause()方法
调用onStop()方法
调用onDestroyView()方法
调用onDestroy()方法
调用onDetach()方法
Fragment被销毁

图2

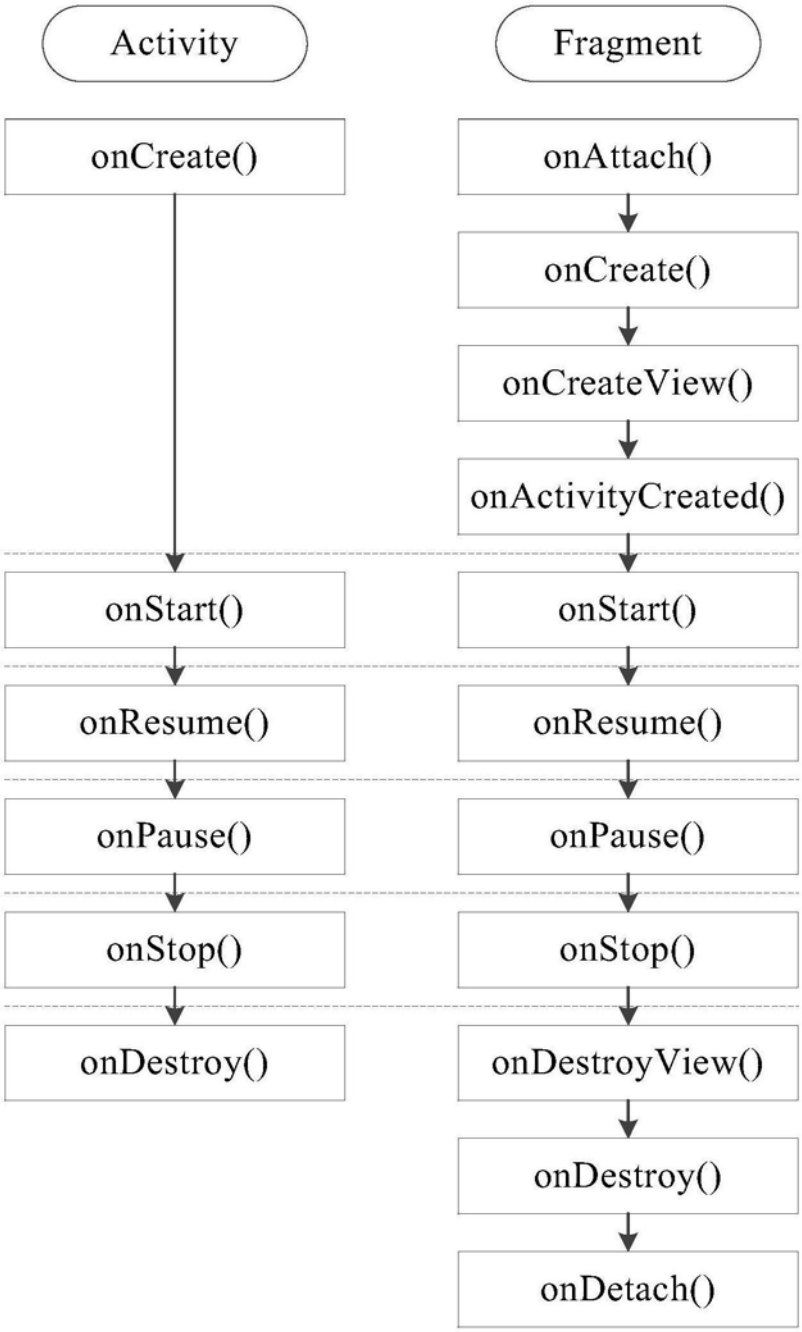
Activity
Fragment
onCreate()
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onStart()
onResume()
onResume()
onPause()
onPause()
onStop()
onStop()
onDestroy()
onDestroyView()
onDestroy()
onDetach()

图3

图4A


MARIE DALGAR
玛丽黛佳
CLARINS
娇韵诗
茵曼内衣
茵曼
谜秀
MISHOW
谜秀
FAIRWHALE
马克华菲
准会员
积分0
点击查看专属特权>
填资料送积分
收起卡包

图4B


CLARINS
娇韵诗
体验会员
积分0
点击查看专属特权>
入会试用装
茵曼内衣
茵曼
谜秀
MISHOW
谜秀
FAIRWHALE
马克华菲
收起卡包

图4C

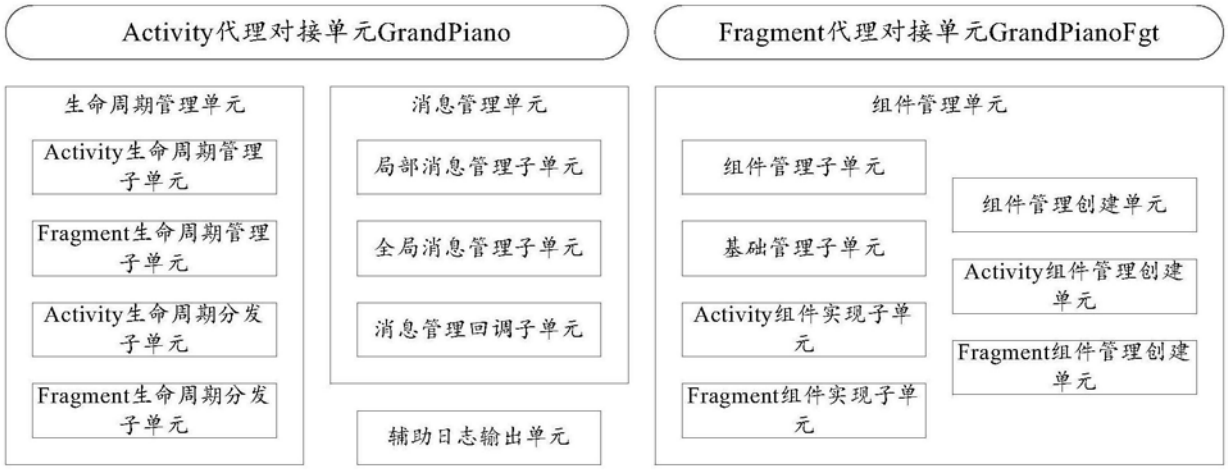
Activity代理对接单元GrandPiano
Fragment代理对接单元GrandPianoFgt
生命周期管理单元
Activity生命周期管理子单元
Fragment生命周期管理子单元
Activity生命周期分发子单元
Fragment生命周期分发子单元
消息管理单元
局部消息管理子单元
全局消息管理子单元
消息管理回调子单元
辅助日志输出单元
组件管理单元
组件管理子单元
基础管理子单元
Activity组件实现子单元
Fragment组件实现子单元
组件管理创建单元
Activity组件管理创建单元
Fragment组件管理创建单元

图5

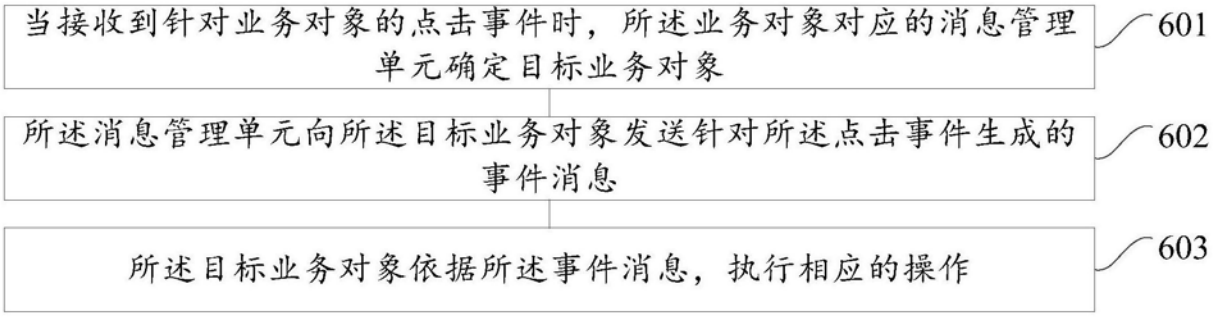
当接收到针对业务对象的点击事件时，所述业务对象对应的消息管理单元确定目标业务对象
601
所述消息管理单元向所述目标业务对象发送针对所述点击事件生成的事件消息
602
所述目标业务对象依据所述事件消息，执行相应的操作
603

图6

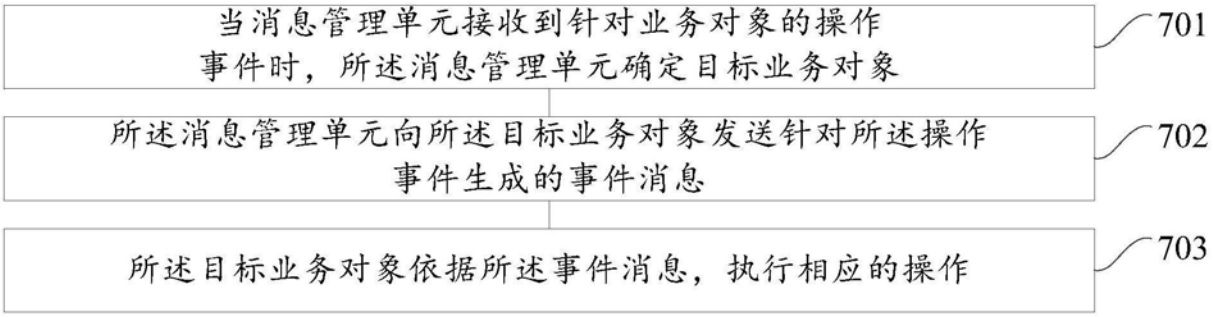
当消息管理单元接收到针对业务对象的操作事件时，所述消息管理单元确定目标业务对象
701
所述消息管理单元向所述目标业务对象发送针对所述操作事件生成的事件消息
702
所述目标业务对象依据所述事件消息，执行相应的操作
703

图7


确定模块
801
发送模块
802
执行模块
803

图8


目标业务对象确定模块
901
事件消息发送模块
902
执行模块
903

图9

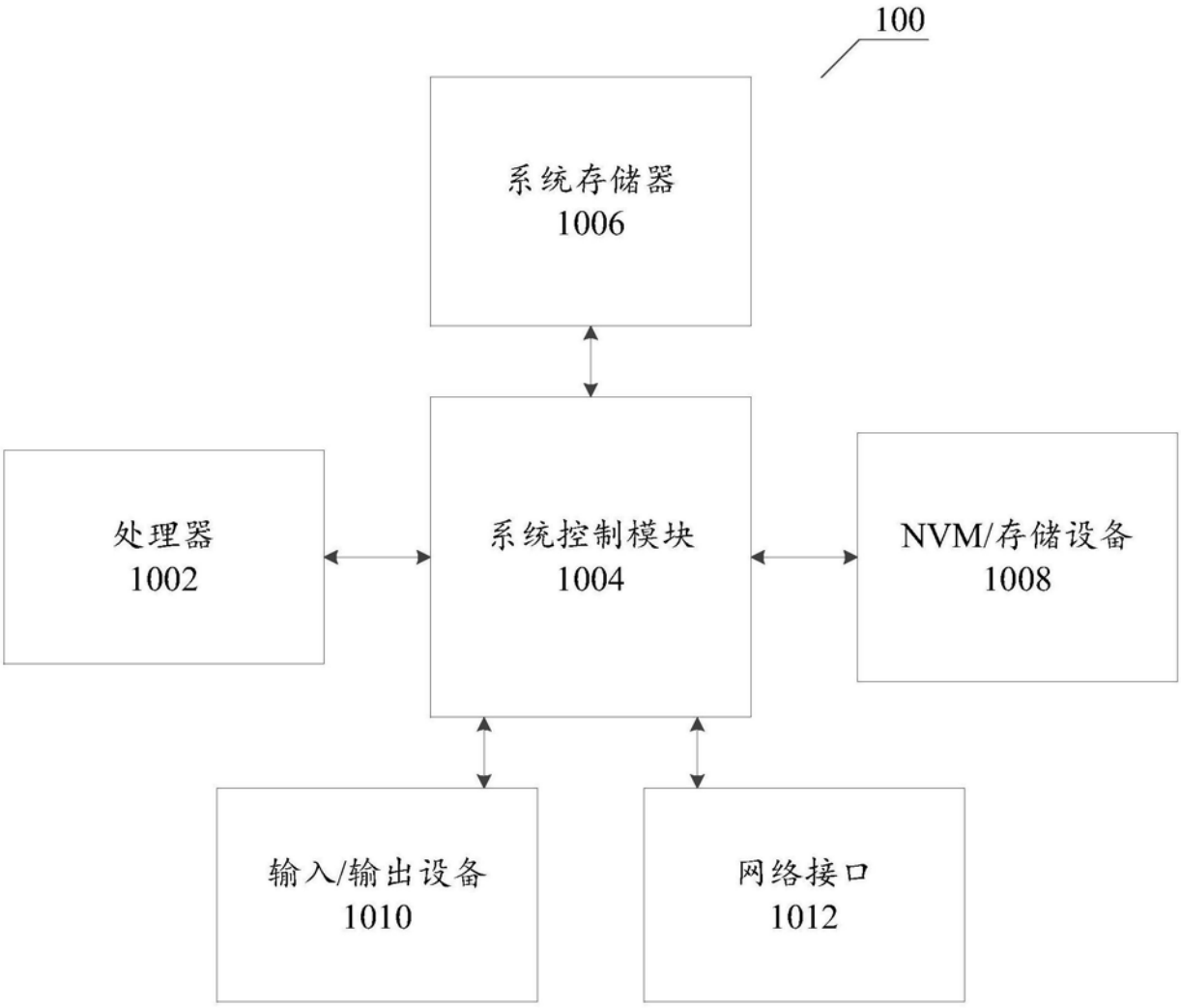
100
系统存储器
1006
处理器
1002
系统控制模块
1004
NVM/存储设备
1008
输入/输出设备
1010
网络接口
1012

图10