面向普适计算环境的Android平台服务编排框架

顾敬潇 彭 鑫 赵文耘

(复旦大学软件学院 上海 201203) (上海市数据科学重点实验室(复旦大学) 上海 201203)

摘 要 普适计算环境下的智能移动设备是面向终端用户的服务资源聚集和编排的主要载体。普适计算环境中的服务资源具有多种不同的形态,包括基于互联网提供的Web服务、终端设备自身服务和资源(例如本地应用、自带传感器)以及所处环境中可访问的服务(例如环境传感器)。此外,不断变化的上下文环境对软件本身的自适应能力提出了新的要求,而移动设备上的服务编排受设备计算能力和资源的限制。为了解决上述问题,提出了一个面向普适计算环境的Android平台服务编排框架ASOF。通过ASOF,移动终端可在运行时获取所需业务流程的服务模板,并对该模板中的抽象服务进行服务绑定,实现轻量级的混合服务编排,使终端能够动态获得调用普适计算环境中各种类型的服务的能力。随后,基于OSGi Felix框架给出了一套ASOF的标准实现,并以一个具体案例验证其有效性。

关键词 普适计算,面向服务的架构,安卓,服务编排

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.012

Pervasive Computing Environment Oriented Service Orchestration Framework for Android

GU Jing-xiao PENG Xin ZHAO Wen-yun

(Software School, Fudan University, Shanghai 201203, China)

(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract Mobile devices and smart phones are the main carriers of service aggregation and orchestration in pervasive computing environment. Services in pervasive computing environment have diverse forms, including remote Web services, local services within devices (such as local app and sensors inside) and context-aware services provided by the surrounding environment (such as environmental sensors). In addition, changing contexts raise new demands for the adaptive ability of software itself, but service orchestration on mobile devices is limited by computing power and resources. In order to solve the above problems, this paper addressed a pervasive computing environment oriented service orchestration framework on Android platform based on SOA, named ASOF. With the help of ASOF, mobile devices are able to obtain service templates of required business processes and bind services for abstract services in the templates. In this way, the devices can perform lightweight mixed service orchestration and dynamically gain the ability to invoke all types of services in pervasive computing environment. After that, a standard implementation of ASOF based on OSGi Felix framework was given and validated by a concrete case.

Keywords Pervasive Computing, SOA, Android, Service orchestration

1 引言

近年来,随着计算机科学领域的不断发展,硬件设施取得了长足的进步,移动设备的生产成本在逐年降低,但其计算能力却在持续提升,普适计算领域逐渐进入了人们的视野。普适计算强调在能够和用户无缝衔接起来、同时融入了计算和沟通能力的环境体系中,移动计算设备借助自身嵌入传感器感知环境上下文,根据上下文的不同采取不同的业务逻辑来服务用户[1]。同时,在软件领域,面向服务的架构(SOA)也逐渐兴起。凭借耦合松散、灵活性复用性高以及业务逻辑与实现技术分离等特点,SOA在大型企业级软件开发过程中得到了广泛的应用。面对上述两种趋势,将面向服务的架构应用到普适计算环境便成为了一个值得研究的领域。

Android作为主流的移动终端操作系统之一,具有开源可深度定制、应用开发成本低、不同平台的兼容性良好等优势。然而,传统Android应用并不是采用面向服务的架构进行开发的,通常开发人员将应用的全部代码一次性编写完成,编译后,打包成扩展名为.apk的文件,发布到应用市场上供用户下载。但软件开发是一个不断迭代的过程,随着软件演化,新的功能点不断被加入,于是整个应用就需要重新编译、打包、发布,然后重新在用户终端部署,这无疑增加了终端上的资源占用和网络消耗。另外,在普适计算的条件下,为适应动态变化的环境,应用就需要具备在不同环境下实现相同或相近的功能效果的能力。传统的Android应用需将不同环境

到稿日期:2015-02-12 返修日期:2015-07-13 本文受国家自然科学基金(61361120097),国家高技术研究发展计划(863)(2013AA01A605)资助。

顾敬潇(1992—),男,硕士生,主要研究方向为自适应软件系统,E-mail:14212010005@fudan.edu.cn;彭 鑫(1979—),男,副教授,博士生导师,主要研究方向为软件维护、自适应软件、软件产品线;赵文耘(1964—),男,教授,博士生导师,主要研究方向为软件维护、自适应软件、软件产品线。

下不同业务逻辑的代码悉数打包到 apk 文件中并静态部署到用户终端上。不能动态获取能力的应用往往是冗余的，而且灵活性也比较差，一旦用户在开发人员没有预想到的环境下使用某个功能，应用则会失效，这在一定程度上降低了用户的体验效果。

因此一个能够分离业务流程与具体实现的关注点、能够动态获取能力、在运行时加载功能点的 Android 应用的运行框架在一定程度上可以解决上述问题。本文针对这一领域存在的问题，提出了一套基于面向服务架构的在运行时进行混合服务组装的轻量级 Android 应用开发模型框架 ASOF(Android Service Orchestration Framework)。

2 背景和相关工作

本节首先概述与本文工作相关的技术背景，然后介绍一些相关研究工作，并与本文的工作进行了对比。

2.1 背景知识

OSGi[2](Open Service Gateway initiative)是一种 Java 动态组件模型规范，其中的组件可以以服务的形式相互调用。OSGi 为开发人员提供了以下几个特性[3]：模块化开发、相同模块多版本同时部署在一个 JVM 中、模块运行时热部署、模块实现细节的封装、面向服务开发以及模块间的依赖管理。OSGi 模型中，OSGi 服务平台是基于 Java 语言实现的动态模块化系统。OSGi 将模块统称为组件(Bundle)，同时提供了一套用来管理 Bundle 的运行框架，每个 Bundle 在实现过程中被打包为普通的 JAR 文件，其中的清单文件(manifest)包含了对整个 Bundle 的元信息和对 OSGi 框架的依赖情况。每个 Bundle 可以以声明式服务(declared service)的形式向 OSGi 服务平台的服务注册中心(service registry)注册服务，其他 Bundle 在运行时可以向注册中心获取当前在注册中心注册的、实现了某个特定接口或者包含某些特定规约的服务。

2.2 相关工作

普适计算环境下混合服务组合的相关领域中，部分研究只关注于服务的发现和选取过程。EASY[4]是一套针对普适计算环境在现有服务发现协议之上基于语义、上下文感知和服务质量感知的高效服务发现机制，没有涉及混合服务组合。而文献[5]给出一套普适计算环境下基于上下文协商的动态服务组合方法，关注点在于权衡设备可用资源和用户期望的服务选取算法。

也有部分研究关注于服务组合过程中组合逻辑和算法。PICO[6]是一套面向服务的中间件，它将普适计算环境下的服务和资源抽象为一系列连通图，并将其保存在服务资源库中；再根据服务质量和输入输出等约束构建复合服务，实现服务组合，这套服务组合算法是其主要贡献。MUSIC[7]是一套移动设备上的服务组件自适应中间件，同样基于 OSGi，主要关注根据应用所处上下文环境自主替换服务组件，提升应用整体的可用性，不涉及混合服务组合。文献[8]针对在普适计算环境下服务组合过程中存在的服务的易挥发性，提出了一种机会式的服务绑定的策略，即将每个子服务完成服务绑定后立即执行该服务，而不是在所有子服务绑定完成后再统一执行，从而降低了已绑定的子服务在其他子服务执行过程中挥发的可能性，使得复合服务质量得以提升。

还有一些研究关注于面向服务，以器端的混合服务组合。SOCRADES[9]关注于将物联网中设备对外提供的服务转化为 Web 服务，以供服务调用者发现、查询、选取和调用。OSGi4HSI[10]采用 SOA 的思想，基于 OSGi 提出了一种普适计算环境下的异构服务集成框架，主要面向基于 CORBA、Web 开发的遗留应用，不针对移动平台。

3 ASOF —— Android 服务编排框架

ASOF 开发模型的业务逻辑如图 1 所示。通过 ASOF，开发人员不必像以往开发 Android 应用一样考虑整个应用的业务逻辑实现，而是开发一些基础的功能组件并将其保存在具体服务资源库服务器中，这些组件可以是调用 Android API 对外提供的本地服务，也可以是作为 Web 服务的客户端对外提供的 Web 服务，还可以是封装外部传感器的通信逻辑对外提供的上下文感知相关的服务。另一方面，复合服务定义者根据所需的业务逻辑，定义所需的服务模板并将其保存到服务模板资源库服务器中。而最终用户则通过一个普通的 Android 客户端与整个系统交互。客户端首先根据用户输入或行为获取用户需求，然后向服务模板资源库服务器发起复合服务选取操作，根据模板中的抽象服务描述，客户端服务流程引擎与具体服务资源库服务器交互，服务器匹配到合适的具体服务组件后，将其返回给客户端，服务流程引擎动态加载，随即绑定服务模板中的抽象服务与具体服务组件，形成一套可执行的复合服务模板，最终执行整个模板以完成整个复合服务调用过程。这样的开发模式将 Android 应用开发过程转化为面向服务化的过程，业务逻辑和具体实现的关注点自然分离开来。为了适应普适计算环境下应用所处上下文多变而且不稳定的问题，整个服务流程引擎支持在某些特定情况下失效服务的恢复功能，一旦某个服务组件在运行时失效，服务引擎会主动捕捉这一失效，卸载现有失效服务并重新向具体服务资源库服务器发出请求，获取其他可用的相似服务组件并加载到本地，替换原有失效服务组件，保证服务流程的正常执行。

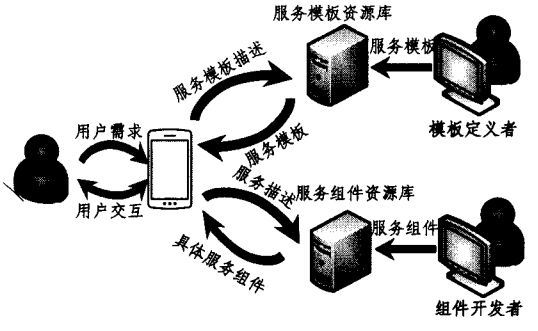

图 1 ASOF 开发模式

ASOF 参考传统的 Web 服务组合引擎的实现，同时整合 OSGi 在 Android 部署的核心框架，具有 5 层体系结构层次，如图 2 所示。

User Behavior Interpreter 即用户行为解释器，这一层次用于获取最终用户与框架直接交互的行为，将其中的有效行为转化为所需复合服务统一的可解析的复合服务描述，并传递给下一层次。这里的有效用户行为可以包括用户对图形化界面的操作、用户的文字输入，甚至用户的语音输入等。

Template Engine 即模板(复合服务)引擎，该层根据上层给定的规范化复合服务描述与模板资源库服务器(图 2 中的 Template Server)交互，以获取与复合服务描述匹配度最高的复合服务模板。限于移动终端有限的计算资源，并出于减少

网络开销的目的，服务模板匹配的工作交由云端执行。模板资源库服务器中模板匹配器(图 2 中的 Template Matcher)通过模板管理器(图 2 中的 Template Manager)检索资源库中的服务模板及其描述，匹配与请求最接近的服务模板返回给模板引擎。随后，模板引擎将加载到的模板保存至本地并使用类加载器将模板载入 JVM，之后交给服务注入引擎完成服务注入，模板引擎便可执行该服务模板。

Service Injection Engine 即服务注入引擎。服务模板中仅包含所依赖服务的抽象描述及调用关系，在运行时，服务引擎需将具体服务实例注入到服务模板，完成服务绑定。这首先要获取服务模板中所有抽象服务的服务描述，这个描述可以是自然语言的描述，也可以是基于本体的，包括服务输入输出、效用、前置后置条件和 QoS(Quality of Service)等约束。服务注入引擎会将每个抽象服务的描述依次传递给组件控制器，获取具体服务实例，再将服务实例注入模板，最后将完成服务绑定的复合服务返回给模板引擎。

Bundle Controller 即组件控制器。这一层次的作用是与服务组件资源库(图 2 中的 Service/Bundle Server)通信，获取对应服务的 OSGi Bundle，同时监控每个 Bundle 的生命周期。每当服务注入引擎调用组件控制器时，组件控制器就会根据服务描述向服务组件资源库提交一次获取服务组件的请求，与模板资源库服务器相类似，服务组件资源库也利用组件匹配模块(图 2 中的 Service Matcher)调用组件管理器(图 2 中的 Bundle Manager)匹配到最符合服务描述的 Bundle。匹配的过程要将服务描述中的硬性指标(如服务质量约束、输入输出和前置后置条件)作为筛选要素，剔除不满足约束的 Bundle，最后在筛选过的 Bundle 中选取服务效用相似度最高的 Bundle 作为响应，返回给组件控制器。随后，组件控制器会将 Bundle 保存至由 OSGi 框架(图 2 中 OSGI Framework)管理的本地目录下，并向 OSGi 注册回调方法。OSGi 在完成对 Bundle 依赖包的编织和加载过程后，会调用回调方法通知组件控制器相应的服务已加载完成，从而，组件控制器就能够获取具体服务的实例并将其返回给服务注入引擎。服务调用完成后，组件控制器还需将服务对应的 Bundle 从 Felix 中卸载，以释放资源。如果服务在调用过程中出现异常，如服务组件与当前 Android 系统不兼容或发生网络通信异常等，那么当前服务组件被视为失效，为了保障复合服务的正常执行，组件控制器需再次向服务组件资源库服务器发起请求，获取其他有效的服务组件。如此反复尝试，直到组件能够正常完成服务，但如果最终服务组件资源库服务器不能匹配到合适的服务组件，那么组件控制器便会通知上层该复合服务中存在无法绑定的抽象服务，即服务模板在当前环境下失效，最后反馈给用户。

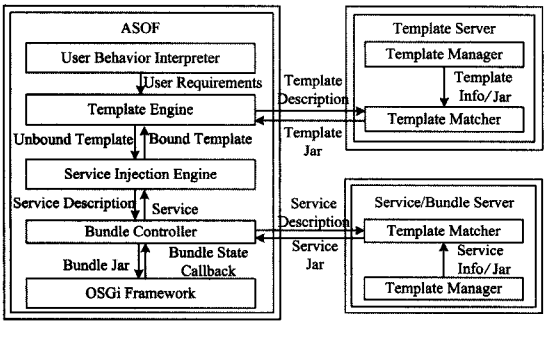

图 2　ASOF 体系结构

4　ASOF 具体实现

本节介绍了 ASOF 的一个具体实现。

4.1　ASOF 核心类

ASOF 的实现基于复合服务(服务模板)、抽象服务和具体服务这 3 个基本概念。

4.1.1　复合服务的表示

Template 类是服务组合过程中表示复合服务的抽象类，每一个定义好的服务模板都需要继承该类，图 3 给出了 ASOF 实现中的一个 Template 实例。

```
1. public class ATemplate extends Template{
2.     @ServiceDescritpion{
3.         description="some descriptions",
4.         input={"var"},
5.         output={"result1","results"}
6.     }AbstractService service1;
7.
8.     @ServiceDescritpion{
9.         description="some other descritpions",
10.        input={"var1","var2"},
11.        output={"resutls3"}
12.    }AbstractService service2;
13.
14.    @Override
15.    protected void orchestra() throws TemplateInvalidException{
16.        String var=requestUserInput("Please Input Var");
17.        ReturnType output1=service1.invokeService(var);
18.        Object var1=output1.get("result1");
19.        Object var2=output1.get("result2");
20.        ReturnType output2 = service2.invokeService(var1,
           var2);
21.        showMessage(output2.get("result3"));
22.    }
23. }
```

图 3　一个简单的复合服务模板

Template 类对外提供一个 orchestraServices()方法，该方法会在服务模板中的全部抽象服务完成服务绑定后，由 Template Engine 调用，启动整个复合服务。该类为子类提供了一系列阻塞式的用户输入交互方法来获取用户输入，并借助 JDK 的 FutureTask 类将异步输入过程转为同步调用。

Template 的子类中，声明抽象服务是以声明类的字段的形式实现的，其中每个抽象服务字段的类型都是 AbstractService，而服务的描述则使用了 Java 的注解(Annotation)特性。模板中每个 AbstractService 字段都需要配上 ServiceDescription 注解才能够在运行时被服务引擎发现并注入。在运行时，Service Injection Engine 会获得每个 AbstractService 的 ServiceDescription 注解以作为服务匹配的依据与服务器交互。

4.1.2　抽象服务和具体服务的表示

AbstractService 类是服务组合过程中表示抽象服务的抽象类，每一个具体服务都需要继承该类。ASOF 中每一个具体服务都是以 OSGi Bundle 的形式存在，因此 AbstractService 类实现了 Felix 要求每个 Bundle 都要实现的 BundleActivator 接口，并在 Bundle 初始化方法中将 AbstractService.class 作为 Bundle 对外暴露的服务注册到 Felix 框架中，以供随后模板引擎调用。

AbstractService 类对外的 public 方法只有表示调用服务的 invokeService()，接受任意类型和数量的不定参数。为提升服务匹配成功率，ASOF 提供了服务适配。当前的实现只考虑了参数和返回值的适配。

AbstractService 为子类提供了 startServiceActivity()方法，用于服务启动 Activity 组件与用户交互。但在现有 Android 操作系统下是不能动态创建 Activity 的，这里利用一个作为适配器存在的类 ServiceActivity 和一个在 manifest. xml 文件中声明的 Activity，使具体服务能在在运行时动态创建一个 Activity。如果服务 Bundle 开发人员希望在 Bundle 执行过程中创建 Activity 与用户更友好地交互，则只需在 Bundle 的 Jar 包的 Manifest 文件中指定 Service-Activity 属性，标记继承了 ServiceActivity 的子类的类名，ASOF 即可在运行时通过反射机制将该类实例化，适配已在系统注册过的那个 Activity 中，从而实现 Activity 的动态创建。

此外，出于安全性考虑，Android 系统不允许应用 manifest. xml 中未声明的系统权限，服务组件因此不能在运行时动态获取 Android 系统权限。ASOF 当前的实现在默认服务 Bundle 无恶意的前提下，其 Android 客户端申请了全部系统权限，在真正实施的过程中可以采用类似 iOS 应用发布平台 iTunes 的 App Review 机制来保障服务组件的无害性。

4.2 复合服务和具体服务组件打包与发布

复合服务和具体服务组件打成 Jar 包后，才能发布到对应的服务器上，以供 ASOF Android 客户端下载。由于服务器是通过 Jar 包的原信息进行匹配的，因此在打包前，需为 Jar 包添加 manifest 文件。对于复合服务，需在原信息中指定模板名称、模板功能的自然语言描述和继承 Template 类子类的类名等字段；对于具体服务组件，需指定继承 AbstractService 类子类的类名、服务 Bundle 实现过程中 import 的所有包名、服务的功能性自然语言描述、服务 Bundle 在运行时要创建的 ServiceActivity 子类的类名以及服务输入参数和返回值的名称序列等字段。再使用 Android SDK 中的 dx 和 aapt 工具将 Bundle Jar 包转化为 Dalvik 所能识别的 Jar 包，就能发布。

4.3 ASOF 各阶段各层次业务逻辑实现

本文 ASOF 实现作为一个服务编排引擎，在服务编排过程中主要经历 3 个阶段：服务模板获取阶段、服务绑定阶段和服务调用阶段。

4.3.1 服务模板获取阶段业务逻辑

首先，用户行为解释器获取用户关于所期望的复合服务功能的自然语言描述的语音输入，将其转化为字符串。接着，用户行为解释器将服务描述传递给模板引擎，由模板引擎向模板资源库服务器发起请求，服务器通过余弦相似性算法选取服务器端与服务描述最接近的服务模板返回给 ASOF Android 客户端。最后，客户端将获得的模板 Jar 文件保存至本地，利用 Android 类加载器将该 Jar 文件中的 Template 的子类加载到内存，并创建一个 Template 对象。

4.3.2 服务绑定阶段业务逻辑

模板引擎在创建 Template 实例之后会调用服务注入引擎，为 Template 实例注入服务字段。随后服务注入引擎会通过反射获取所有的 AbstractService 字段，同时向 Template 实例注册 Bundle 卸载监听器供复合服务调用，完成后卸载所有依赖的服务 Bundle。接着服务注入引擎对每个 AbstractService 字段进行如下操作：首先获取其 ServiceDescription 注解，根据其中服务描述(包括功能性自然语言描述、服务输入参数的名称序列以及服务返回值的名称序列)，由 Bundle 控制器向服务组件资源库服务器发起请求。服务组件资源库服务器首先匹配参数和返回值，获得服务器端的每个具体服务组件的上述一一映射关系。对于能找到这样的映射的备选 Bundle，用余弦相似性计算 Bundle 的服务描述与请求之间的相似度作为匹配度。最后将匹配度最高的 Bundle 的 Jar 文件和表示映射关系的两个数组 inputMatch 和 outputMatch 返回给客户端。之后，客户端会将对应的服务 Bundle 的 Jar 文件保存到 OSGi 管理的动态发现 Bundle 的目录下，紧接着以该 Bundle 的唯一标示符作为索引，向 OSGi 注册 Bundle 监听器。当 OSGi 加载了这个 Bundle 后，监听器便会触发，通知 Bundle 控制器相应的 Bundle 已经加载完毕。这时，Bundle 控制器就可以获得服务 Bundle 注册的 AbstractService 服务的实例，随后通过服务注入引擎注册的服务监听器返回给服务注入引擎，服务注入引擎即可通过反射将 AbstractService 实例注入对应的 AbstractService 字段。

4.3.3 服务调用阶段业务逻辑

模板引擎得知 Template 实例已经完成服务绑定时，会调用该实例的 orchestraService()方法，随后组合服务开始运行。运行过程中，模板可获得用户输入，一旦其中某一个抽象服务在执行过程中发生异常，说明该服务实例在当前运行环境下失效，此时，AbstractService 的子类实例就会直接与 Bundle 控制器交互，将失效的服务组件卸载，将自己的服务描述作为参数一起注册过去，完成与服务注入引擎在服务绑定阶段相同的操作，获取一个新的服务组件。获取服务组件成功后调用新服务的 invokeService()方法，实现服务的修复工作。最后，调用在服务绑定阶段服务注入引擎向服务模板注册的 Bundle 卸载监听器的 uninstallBundles()方法，根据服务模板绑定的所有 Bundle 的唯一标示符，调用 Bundle 控制器卸载对应 Bundle 释放资源。

5 案例分析

本文通过一个在普适计算环境下利用 ASOF 实现的网上购书的服务组合案例来验证其可行性。

传统网上购书的业务流程主要包括以下 5 步：进入图书网站→搜索图书→填写收货人信息→下单→支付。而在移动环境下，借助终端传感器和云端资源可将上述步骤简化为：首先用户输入书名，随后通过位置服务获得用户当前的地理位置信息，然后通过位置转化服务获得当前用户的地址，并将它作为收件人的地址，接着通过个人信息服务获得本机号码，再将上述信息传给购书服务生成一条订单，最后通过第三方支付服务完成支付。

为验证 ASOF 在普适环境下对由多种类型的服务(包括 Android 本地服务、Web 服务以及上下文相关的外部传感器服务)组成的复合服务可用性的提升作用，服务组件资源库中每种服务分别实现如下。位置服务由两种组件实现，一种是基于 Android 系统本地的多位置源的位置服务实现定位组件，另一种是基于蓝牙射频技术实现的室内定位组件。前者定位精度低，但在室外可用性比较好；而后者定位精度高，但需要附近存在蓝牙定位热点。位置转化服务由利用的百度地

(下转第 64 页)

[2] Scott D. Assessing the costs of application downtime[OL]. http://citeseerx.ist.pus.edu/showciting?cid=3757589

[3] Ma X, Baresi L, Ghezzi C, et al. Version-consistent dynamic reconfiguration of component-based distributed systems[C]// Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011: 245-255

[4] Kramer J, Magee J. The evolving philosophers problem: Dynamic change management[J]. IEEE Transactions on Software Engineering, 1990, 16(11): 1293-1306

[5] Vandewoude Y, Ebraert P, Berbers Y, et al. Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates[J]. IEEE Transactions on Software Engineering, 2007, 33(12): 856-868

[6] Su Ping, Cao Chun, Ma Xiao-xing, et al. Automated Management of Dynamic Component Dependency for Runtime System Reconfiguration[C]// Software Engineering Conference (APSEC 2013). IEEE, 2013: 450-458

(上接第 55 页)

图反向地址解析 API 发起 HTTP 请求将经纬度转化为地址信息的组件实现。用户信息服务由能够获取本机号码的服务组件实现。购书服务由两种与部署在不同 Tomcat 中的购书 Web 服务 A、B 进行 SOAP 通信的服务组件实现，其中 Web 服务 A 可用性较差，B 较好。第三方支付服务由一个与 Web 服务通信的组件实现。

实验过程从两个维度考虑：1)附近有无蓝牙定位热点，2)服务 A 所在的 Tomcat 是否开启。因此就有了 4 组对比实验，系统日志如图 4 所示，运行环境(a)为有蓝牙定位热点且购书服务 A 可用，(b)为附近无蓝牙定位热点但购书服务 A 可用，(c)为附近有蓝牙定位热点但购书服务 A 不可用，(d)为附近无蓝牙定位热点且购书服务 A 不可用。从框出部分可以看出，ASOF 能够实现在不同的移动环境下替换失效的服务绑定，在一定程度上实现系统的自修复，从而提升组合服务整体上的可用性。

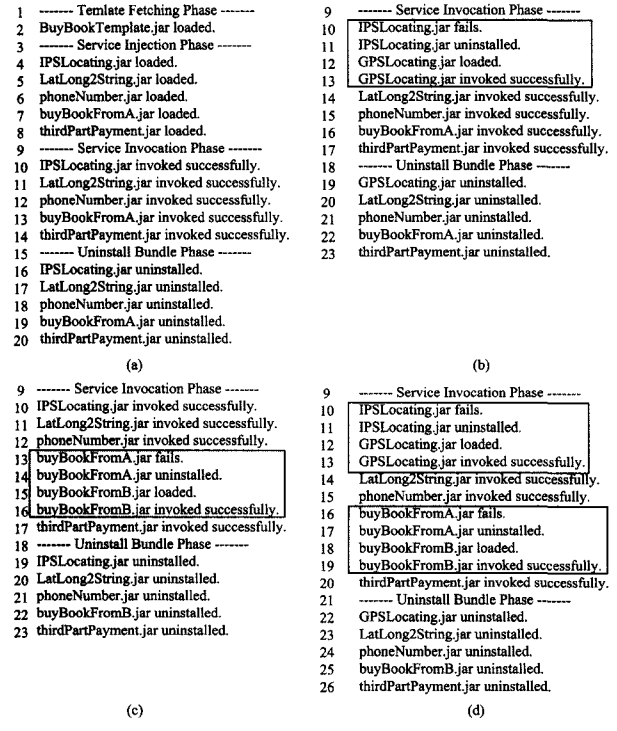

图 4　4 组对比实验系统日志

结束语　本文提出了一套 Android 移动终端服务编排框架 ASOF，该框架借助 OSGi 规范，实现了普适计算环境下 Android 平台上不同类型服务的混合组装。在服务失效时，该框架能够自主替换失效的服务组件，实现复合服务的自修复。由于模板和服务匹配都是在服务器端进行的，因此客户端的资源消耗得以降低，服务组合的过程也就更加轻量。同时，本文也给出了一套标准的 ASOF 实现，并为 ASOF 的两个重要参与者(模板定义者和服务开发者)提供了一套开发工具包。基于工具包，模板定义者可以定义符合自己业务逻辑需求的组合服务模板，服务开发者可以实现具有一定功能的具体服务组件，以对应资源库作为平台发布。最后以一个具体案例验证了框架在不同的普适计算环境下复合服务的自修复能力。

参 考 文 献

[1] Satyanarayanan M. Pervasive computing: Vision and challenges [J]. Personal Communications, IEEE, 2001, 8(4): 10-17

[2] OSGi Homepage[OL]. http://www.osgi.org

[3] Knoernschild K. Java Application Architecture: Modularity Patterns with Examples Using OSGi[M]. Prentice Hall Press, 2012

[4] Mokhtar S B, Preuveneers D, Georgantas N, et al. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support[J]. Journal of Systems and Software, 2008, 81(5): 785-808

[5] 唐磊，淮晓永，李明树. 一种基于上下文协商的动态服务组合方法[J]. 计算机研究与发展，2008，45(11)：1902-1910
Tang Lei, Huai Xiao-yong, Li Ming-shu. An approach to Dynamic Service Composition Based on Context Negotiation[J]. Journal of Computer Research and Development, 2008, 45(11): 1902-1910

[6] Kalasapur S, Kumar M, Shirazi B. Dynamic service composition in pervasive computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(7): 907-918

[7] Rouvoy R, Barone P, Ding Y, et al. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments [M]// Software engineering for self-adaptive systems. Springer Berlin Heidelberg, 2009: 164-182

[8] Groba C, Clarke S. Opportunistic composition of sequentially-connected services in mobile computing environments[C]// 2011 IEEE International Conference on International Conference on Web Services (ICWS). IEEE, 2011: 17-24

[9] Guinard D, Trifa V, Karnouskos S, et al. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services[J]. IEEE Transactions on Services Computing, 2010, 3(3): 223-235

[10] 张威，史殿习. OSGi4HSI：普适计算环境下的异构服务集成框架[OL]. http://cpfd.cnki.com.cn/Article/CPFDTOTAL-JDMT-201010001006.htm