计算机科学
COMPUTER SCIENCE

基于计算反射的Android应用程序接口自动生成方法

王毅, 陈迎仁, 陈星, 林兵, 马郓

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211100066

基于计算反射的 Android 应用程序接口自动生成方法

王 毅[1,2] 陈迎仁[1,2] 陈 星[1,2] 林 兵[2,3] 马 郓[4]
1 福州大学数学与计算机科学学院 福州 350116
2 福建省网络计算与智能信息处理重点实验室 福州 350116
3 福建师范大学物理与能源学院 福州 350117
4 北京大学信息科学技术学院 北京 100871
(396882243@qq.com)

摘 要 随着移动设备硬件技术和5G等通信技术的发展,智能应用软件不断涌现,其提供的功能已涉及人们生活和工作的方方面面。应用内功能众多,不仅可以满足应用使用者的需求,还能被进一步发布成应用程序接口(API)用于外部调用,例如应用发布的API可以被智能语音助手调用。然而,为了生成应用内功能的API,开发者通常需要在应用开发阶段通过手工编码来实现,对于开发时没有发布的API,在应用上线以后,其功能则无法被外部调用。针对此问题,文中提出了一种基于计算反射的Android应用API自动生成方法。该方法能够在不修改源代码的情况下,基于计算反射机制重建Android应用的Activity界面运行时软件体系结构;面向指定功能的测试用例,分析用户行为工作流以及对应的程序调用;通过模拟用户行为的方式调用指定功能,并生成对应的API。针对"豌豆荚"Android应用商店中的300个流行应用进行方法评估,实验结果显示,所提方法适用于其中的280个应用;对于指定功能,所提方法能够在15 min左右实现其API,且API的性能满足外部调用的需求。

关键词:API生成;Android应用;计算反射;运行时软件体系结构;工作流
中图法分类号 TP391

Automating Release of Android APIs Based on Computational Reflection

WANG Yi[1,2],CHEN Ying-ren[1,2],CHEN Xing[1,2],LIN Bin[2,3] and MA Yun[4]
1 College of Mathematics and Computer Science,Fuzhou University,Fuzhou 350116,China
2 Fujian Key Laboratory of Network Computing and Intelligent Information Processing,Fuzhou 350116,China
3 College of Physics and Energy,Fujian Normal University,Fuzhou 350117,China
4 School of Electronics Engineering and Computer Science,Peking University,Beijing 100871,China

Abstract With the development of mobile hardware and 5G communication technologies,smart applications are booming,which has penetrated into all the aspects of our life and work. There are many functions in applications,which can not only satisfy the user's requirements,but also be further released as APIs for external invocations. For instance,the APIs provided by applications can be invoked by the intelligent voice assistant. However,these functions must be released as APIs during the development phase,otherwise they cannot be used for external invocations. To address this problem,this paper proposes an approach to automatic release of APIs of Android applications based on computational reflection. It first rebuilds runtime software architecture for the activities of Android applications based on the reflection mechanism,without modifying the source code of application. Then,based on test cases of the specified function,it analyzes its user-behavior workflow and corresponding procedure calls. Finally,the function can be invoked by simulating the user behaviors,and then is released as the corresponding API. We evaluate our approach with 300 popular apps on Android app store Wandoujia,and the results show that our approach is effective for 280 of them. For the specified functions,APIs can be implemented by our approach in about 15 minutes,and their runtime performance is desirable.

到稿日期:2021-11-05 返修日期:2022-07-28
基金项目:国家重点研发计划(2018YFB1004800);福建省自然科学基金杰青项目(2020J06014);中央引导地方科技发展专项(2019L3002)
This work was supported by the National Key R&D Program of China(2018YFB1004800),Natural Science Foundation of Fujian Province for Distinguished Young Scholars(2020J06014) and Special Project on Science and Technology Development of Central Government Guiding Local Government(2019L3002).
通信作者:陈星(chenxing@fzu.edu.cn)

Keywords　API generation，Android app，Computational reflection，Runtime software architecture，Workflow

1　引言

近年来，随着移动互联网的蓬勃发展以及智能设备的快速普及，人们逐渐习惯通过移动智能设备来访问网络。为了满足人们的多样化需求，大量智能应用不断涌现，提供的功能涉及人们生活和工作的方方面面，这些智能应用已经成为人们通过移动端访问互联网的主要渠道[1-2]。各类智能应用提供了众多功能，这些功能不仅可以在应用内使用，还能被进一步发布成应用程序接口（Application Programming Interface，API）供外部调用。例如，Siri 等智能语音助手根据语音输入信息调用其他应用的功能，并跳转到目标功能页面（Activity）[3]；微信等社交应用根据输入的位置信息，跳转到第三方地图应用的导航页面；智能家居集成应用根据用户指令调用不同品牌的智能设备功能，并跳转到该品牌智能家居应用的目标功能页面。这些应用内功能的 API，需要由开发者在开发阶段通过编写代码实现，而对于应用开发时没有发布的 API，在应用上线以后，其功能则无法被外部调用。此时，若希望通过外部调用来访问这些没有 API 的应用内功能，则需要进行二次开发以实现其 API。

在前期工作[4]中，我们以“豌豆荚”应用商店[5]为例，对当前 Android 应用对外发布 API 的情况进行了实证研究。研究结论如下：1）对比“豌豆荚”下载量排名前 200 应用的初始版本和 2017 年 1 月的最新版本，对外发布 API 的应用数量从 30％提高到了 80％，这说明对外发布 API 已经成为当前 Android 应用的一种趋势；2）在“豌豆荚”下载量排名前 20000 的热门应用中，仅有不到 30％的应用对外发布 API，这说明当前仅有少部分 Android 应用已对外发布 API；3）通过对部分 Android 应用的研究发现，对外发布一个指定功能的 API 通常需要手动修改 45－411 行代码。

因此，将应用内功能发布成 API 已成为一种趋势，且具有极大需求。然而，现有的 API 开发方式需要对应用源代码进行修改，工作量大，难以用于第三方应用的 API 开发。深层链接（Deep Link）是移动智能应用领域的一个新兴概念[6]，支持直接跳转到第三方应用的指定 Activity。虽然现有研究工作[4,7]能够生成指定 Activity 的深层链接，但是对于需要经过多个 Activity 才能完成的复杂功能，如打开“QQ 音乐”并播放指定的歌曲，则没有提供良好的外部调用支持。

计算反射[8]指使计算系统具有精确“自表示”的能力，它要求系统的表示能够与系统的状态和行为保持一致。基于计算反射思想，一些研究工作[9-14]采用逆向工程技术实现了传统表单系统、Web 应用和工业软件等类型应用的 API 生成。本文基于计算反射思想研究 Android 应用的 API 自动生成方法，通过模拟用户行为的方式调用指定功能，并生成对应的 API。本文的主要贡献如下：

（1）提出了一种基于计算反射的 Android 应用程序接口自动生成方法。该方法能够在不修改源代码的情况下，基于计算反射机制重建 Android 应用的 Activity 界面运行时软件体系结构，以监听界面状态；面向指定功能的测试用例，分析用户行为工作流以及对应的系统调用；通过模拟用户行为的方式调用指定功能，并生成对应的 API。

（2）实现了一套 Android 应用程序接口自动生成工具，针对“豌豆荚”Android 应用商店中的 300 个流行应用进行方法评估。实验结果显示，该方法适用于其中 280 个应用。在实验过程中，能够在 15min 左右实现指定功能的 API，并且 API 性能满足外部调用的需求。

本文第 2 节概述了本文方法的整体框架；第 3 节介绍了基于计算反射的 Activity 界面运行时软件体系结构重建；第 4 节介绍了测试用例驱动的用户行为工作流获取；第 5 节介绍了基于模拟用户行为的 API 建模与执行；第 6 节对本文方法进行了评估；第 7 节介绍了相关工作；最后总结全文。

2　方法概览

图 1 给出了基于计算反射的 Android 应用程序接口自动生成方法概览。该方法能够在不修改源代码的情况下，基于计算反射机制重建 Android 应用的 Activity 界面运行时软件体系结构；针对指定应用内功能的测试用例，分析用户行为工作流以及对应的系统调用；通过模拟用户行为的方式调用应用内功能并自动生成其 API。该方法主要包含 3 个步骤：1）基于计算反射的 Activity 界面运行时软件体系结构重建；2）测试用例驱动的用户行为工作流获取；3）基于模拟用户行为的 API 建模与执行。

首先，图 1(a)给出了基于计算反射的 Activity 界面运行时软件体系结构重建方法，以监听界面状态。利用 Activity 界面中构件的构件名、组织结构、服务接口等元信息，创建相应的元对象；采用元建模技术，定义 Activity 界面运行时软件体系结构的语法和语义，通过计算反射机制建立运行时软件体系结构与运行系统之间的直接因果关联。

其次，图 1(b)给出了测试用例驱动的用户行为工作流获取方法，用于构造指定功能的系统调用工作流。基于 Activity 界面运行时软件体系结构，实现用户操作的监听；面向指定功能的一组测试用例，分析用户行为工作流以及对应的系统调用，获取目标功能的数据流和控制流。

最后，图 1(c)给出了基于模拟用户行为的 API 建模与执行方法。在 Activity 界面运行时软件体系结构的基础上，实现用户操作的模拟执行；根据指定功能的用户行为工作流，生成对应的 API，通过模拟用户行为的方式调用其功能。

基于上述方法，开发者能够在不修改源代码的情况下生成 Android 应用的 API，极大地降低了第三方开发 API 的难度和复杂度。

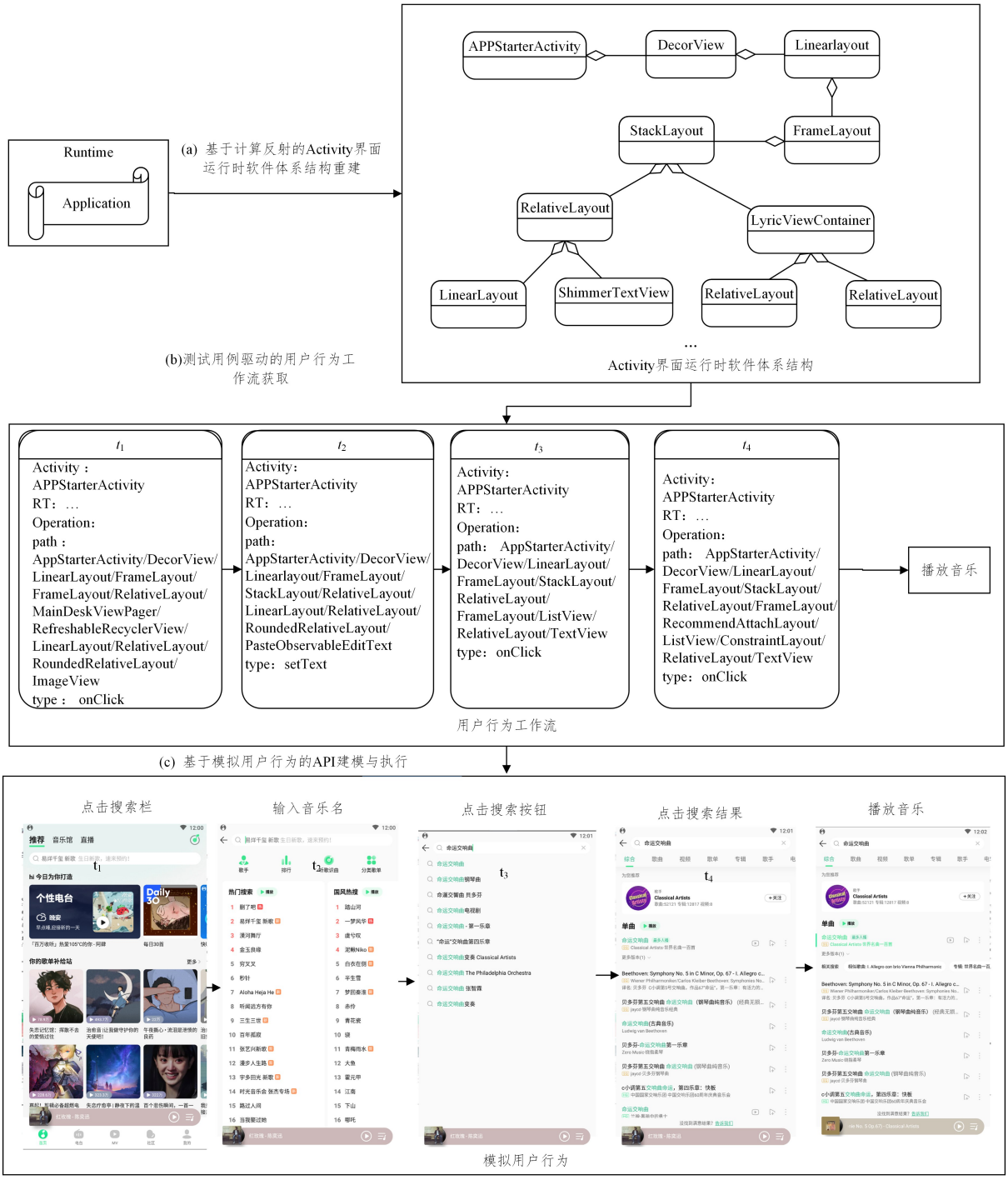


图 1 基于计算反射的 Android 应用程序接口自动生成方法概览

Fig. 1 Overview for the approach to automatic generation of Android APIs based on computational reflection

3 基于计算反射的 Activity 界面运行时软件体系结构重建

运行时软件体系结构[15-16]是对运行时系统的体系结构级别的动态表示，有助于在较高的抽象级别上监视和调整系统的运行时行为。为了通过模型读写来实现系统监控，运行时软件体系结构与运行系统需要保持因果关联：运行系统的任何信息均反映到运行时软件体系结构上，而运行时软件体系结构的任何变化也会作用到运行系统上。北京大学团队在运行时软件体系结构建模理论及构造方法方面进行了研究[17-19]：给定系统的元模型(描述了系统的结构信息)与访问模型(描述了系统的管理接口)，SM@RT 工具[20]可以自动生成代码，在保证性能的前提下实现模型到管理接口的映射。

基于 SM@RT 工具，本文提出了基于计算反射的 Activity 界面运行时软件体系结构重建方法，以支持对 Activity 界面状态的监听。

3.1 Activity 元模型

Activity 界面运行时软件体系结构是实际 Activity 界面的抽象，包括界面的构件和组织结构等信息。界面的构件包括界面(Activity)和其中的各种视图控件(View)。因此，在构建 Activity 界面运行时软件体系结构前需要对 Android 应用进行分析，通过 IC3 工具[21]和 GATOR 工具[22]能够分别获得

不同类型的 Activity 类和 View 类。

图 2 给出了 Activity 元模型，其主要组成部分包括界面(Activity)、视图控件(View)、组合视图控件(ViewGroup)、线性布局控件(LinearLayout)、搜索框控件(SearchView)、图像控件(ImageView)、按钮控件(Botton)等。其中，Activity 表示界面，描述界面的基本信息，同时包含该界面的所有视图控件。View 表示视图控件，描述视图控件的基本信息，包括控件的类名、所在路径、宽度、高度等。典型的视图控件包括按钮控件(Botton)、文本控件(TextView)等。ViewGroup 表示组合视图控件，包含多个子视图控件，其类型包括线性布局控件(LinearLayout)等。

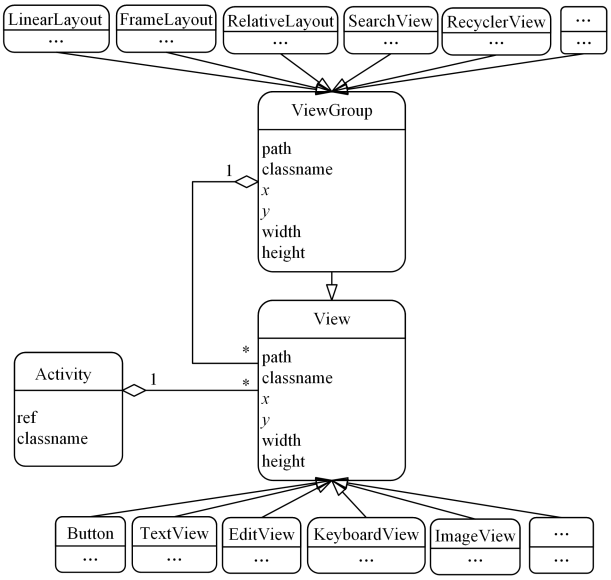


图 2　Activity 元模型

Fig. 2　Meta-model of Activity

定义 1　Activity 界面运行时软件体系结构。在 Activity 元模型的基础上，Activity 界面运行时软件体系结构表示为 $RT(E, HashMap\langle e, List(e)\rangle)$。其中，$E$ 表示元素的集合，$E=\{e_1, e_2, \cdots, e_n\}$，$e_i$ 表示运行时软件体系结构中的元素，其中 e_1 表示根节点，类型为 Activity，其他 e_i 类型为 View；$e_1=\{ref_1, classname_1\}$，其中 ref_1 表示根节点 e_1 在系统中对应对象的引用，$classname_1$ 表示 e_1 的类名；$e_i=\{path_i, classname_i, x_i, y_i, width_i, height_i\}$，其中 $path_i$ 表示元素 e_i 在运行时软件体系结构中的所在路径，$classname_i$ 表示 e_i 的类名，x_i 表示 e_i 在界面中的横坐标，y_i 表示 e_i 在界面中的纵坐标，$width_i$ 表示 e_i 在界面中的宽度，$height_i$ 表示 e_i 在界面中的高度。$HashMap\langle e, List(e)\rangle$ 表示元素间的聚合关系，$List(e_i)$ 表示元素 e_i 的子元素的集合，$List(e_i)=\{e_{i1}, e_{i2}, \cdots, e_{im}\}$。

3.2　Activity 访问模型

Activity 访问模型描述了模型操作所需的系统接口的调用方法，由于 Activity 界面运行时软件体系结构主要用于用户操作的监听，其模型操作主要包括 Get，List 和 Create 这 3 种类型。如表 1 所列，*Get* 操作表示读取运行时软件体系结构中元素的属性值；*Create* 操作表示创建运行时软件体系结构中的元素；*List* 操作表示获取运行时软件体系结构中元素的所有子元素。

表 1　Activity 界面运行时软件体系结构的模型操作

Table 1　Model operations of Activity runtime software architecture

操作	规则描述
Get	$Get\ e.property \rightarrow Get(e.ref, property)$
Create	$Create\ e \rightarrow Create(e.ref)$
List	$List\ e \rightarrow List(e.ref)$

基于计算反射机制实现上述模型操作，具体规则如下：

(1) $Get(e.ref, property)$。$e.ref$ 表示元素 e 在系统中对应对象的引用，基于计算反射机制，读取对应对象的属性 $property$ 的值。

(2) $Create(e.ref)$。$e.ref$ 表示系统中目标对象的引用，通过 *Get* 操作获取目标对象所有属性的值，并在运行时软件体系结构中创建对应的元素 e。

(3) $List(e.ref)$。$e.ref$ 表示元素 e 在系统中对应对象的引用，基于计算反射机制，获取其所有成员对象的引用，进一步通过 *Create* 操作，在运行时软件体系结构中创建其成员对象对应的元素。

3.3　Activity 界面运行时软件体系结构重建机制

当界面发生变化时，需要重建 Activity 界面运行时软件体系结构，来维护运行时软件体系结构与运行系统之间的双向同步。因此，对 Activity 类的 onResume 方法和 View 类的 onDraw 方法进行监听，当这两种方法被调用时，说明界面发生了变化，此时重建 Activity 界面运行时软件体系结构。

Activity 运行时软件体系结构重建算法如算法 1 所示。其输入为当前 Activity 的引用 ref_{root}，输出为运行时软件体系结构 RT，使用元素链表 *eList* 存放等待处理的元素，使用 *currentE* 表示当前要处理的元素，使用子元素链表 *childList* 存放 *currentE* 的子元素。

算法 1　运行时软件体系重构

输入：activity reference ref_{root}

输出：runtime software architecture RT

定义：

childList：the set that contains all subsequent elements of currentE.

eList：the set that contains all elements waiting to be processed.

currentE：the current element to be processed

RT：the runtime software architecture.

```
1. e1 = Create(ref_root);
2.    RT. E. add(e1);
3.    eList. add(e1);
4.    WHILE(eList ! = null) DO
5.          currentE=eList. remove(0);
6.          childList=List(currentE);
7.       FOREACH(eiin childList) DO
8.             RT. E. add(ei);
9.             eList. add(ei);
10.      END FOR
11.      RT. HashMap. add(currentE,childList);
12. END WHILE
```

(1)创建运行时软件体系结构的根元素(见第 1－3 行)。根据当前 Activity 引用 ref_{root}，通过 Create 操作创建 RT 的根元素 e_1，在 RT 中添加元素 e_1 并将其加入元素链表 *eList*。

(2)在运行时软件体系结构中创建一个元素的所有子元素(见第 5－11 行)。首先，取出元素链表 *eList* 中的第一个元

素作为当前要处理的元素 *currentE*，通过 *List* 操作创建其所有子元素并加入子元素链表 *childList*。然后，将子元素链表 *childList* 中的元素逐个添加到 *RT* 中并加入元素链表 *eList*。最后，在 *RT* 中建立元素 *currentE* 到子元素链表 *childList* 的映射。

(3)重复步骤(2)，直到元素链表 *eList* 为空，完成 Activity 界面运行时软件体系结构的重建。

4 测试用例驱动的用户行为工作流获取

用户行为工作流描述了指定应用内功能的执行过程。本文在 Activity 界面运行时软件体系结构的基础上，提出了一种测试用例驱动的用户行为工作流获取方法。该方法对用户操作进行监听，面向指定功能的一组测试用例，分析用户行为工作流以及对应的系统调用，从而获取目标功能的数据流和控制流。

4.1 用户操作的监听

为了获取用户行为工作流，需要对用户操作进行监听。用户操作的定义如下。

定义 2(用户操作) 用户操作表示为 $Operation=\{path,type\}$。其中，*path* 表示操作作用的元素 *e* 的所在路径(*e.path*)；*type* 表示操作类型，主要包括 *onClick* 和 *setText*。*onClick* 表示点击操作，无输入参数；*setText* 表示输入操作，输入参数为文本信息。通过对所有 View 类实例的 dispatchTouchEvent 方法[23]进行监听，获取点击操作；通过对所有 TextView 类实例的 setText 方法[23]进行监听，获取输入操作。

4.2 用户行为工作流的获取

用户行为工作流描述了指定应用内功能的执行过程，包括一组用户操作的触发顺序和触发条件。本文面向指定应用内功能的一组测试用例，监听它们的用户操作序列，并进一步归纳其用户行为工作流。

定义 3(用户操作序列) 用户操作序列是一个测试用例的执行过程，表示为 $TaskSeq=\{t_1,t_2,\cdots,t_n\}$。其中，*t* 表示任务，包括用户操作及其触发时的界面状态。$t=\langle Activity, RT, Operation\rangle$，其中，*Activity* 表示操作所在界面，*RT* 表示操作触发时的界面运行时软件体系结构，*Operation* 表示用户操作。

一个测试用例描述了应用内功能的一次执行过程，指定应用内功能的一组测试用例则描述了该功能的所有执行情况，其对应的用户操作序列集合表示为 $TaskSeqs=\{TaskSeq_1, TaskSeq_2,\cdots, TaskSeq_n\}$。本文针对应用内的指定功能，通过人工手动多次执行该功能来获得对应的测试用例集合。基于用户操作序列集合，分析用户操作的触发顺序和触发条件，进一步归纳用户行为工作流。

定义 4(用户行为工作流) 用户行为工作流表示为 $G(T, HashMap\langle t, List(t)\rangle)$。其中，$T=\{t_1,t_2,\cdots,t_n\}$，表示任务集合；*t* 表示任务，包括用户操作及其触发条件，组成元素与定义 3 一致；$HashMap\langle t, List(t)\rangle$ 表示任务执行顺序，$List(t_i)$ 表示 t_i 的后继任务的集合，$List(t_i)=\{t_{i1},t_{i2},\cdots,t_{im}\}$。

用户行为工作流构建算法如算法 2 所示。其输入为用户操作序列集合 *TaskSeqs*，输出为用户行为工作流 *G*，使用指针 *task* 指向用户行为工作流中的任务，使用链表 *tList* 存放 *task* 的后继任务。算法 2 的流程如下：

(1)初始化用户行为工作流 *G*(见第 2－4 行)。判断用户行为工作流是否为空。若为空，则将用户操作序列 $TaskSeq_i$ 的第一个任务作为用户行为工作流 *G* 的第一个任务。

(2)初始化 *task*，*tList*，并获取当前用户操作序列 $TaskSeq_i$ 的任务数(见第 5－7 行)。将 *task* 指向用户行为工作流 *G* 的第一个任务，将用户行为工作流的第一个任务加入到任务链表 *tList* 中，获取当前用户操作序列 $TaskSeq_i$ 的任务数 *num*。

(3)判断 *task* 后继任务链表 *tList* 中是否存在与 $TaskSeq_i.t_j$ 相同的任务(见第 10－17 行)。遍历 *task* 后继任务链表 *tList*，寻找是否存在任务 $G.T.t_k$ 与 $TaskSeq_i$ 中第 *j* 个任务 t_j 相同。若存在，则将 *isExist* 赋值为 1，将 *task* 指向任务 $G.T.t_k$。更新任务链表 *tList*，将 *j* 的值加 1。

(4)当 *task* 后继任务链表 *tList* 中不存在与 $TaskSeq_i.t_j$ 相同的任务时(具体处理方式见第 19－25 行)，将 $TaskSeq_i$ 中第 *j* 个任务 t_j 及其之后的任务作为当前 *task* 的一个分支，逐个加入用户行为工作流 *G*。

(5)重复执行第 3－4 步，直到遍历完 $TaskSeq_i$ 中的所有任务。

(6)重复执行第 1－5 步，依次处理 *TaskSeqs* 中每一个用户操作序列 $TaskSeq_i$，逐步构建用户行为工作流 *G*。

算法 2 构建用户行为工作流

```
输入:set of user-operation sequences TaskSeqs
输出:user-behavior workflow G
定义:
G:the user-behavior workflow.
task:the pointer that points to a task in the user-behavior workflow.
tList:the set that contains all subsequent tasks of task.
1. FOREACH(TaskSeqi in TaskSeqs) DO
2.    IF(G.T=null) THEN
3.       G.T.t1=TaskSeqi.t1;
4.    END IF
5.    task=G.T.t1;
6.    tList.add(G.T.t1);
7.    num=getTaskNumber(TaskSeqi);
8.    j=1;
9.    WHILE(j <= num) DO
10.      isExist=0;
11.      FOREACH(G.T.tk in tList) DO
12.         IF(equalT(G.T.tk,TaskSeqi.tj)=1) THEN
13.            isExist=1;
14.            task=G.T.tk;
15.            tList=G.HashMap.List(task);
16.            j++;
17.         END IF
18.   END FOR
19.   IF(isExist=0) THEN
20.         WHILE(j <= num) DO
21.            G.T.add(TaskSeqi.tj);
22.            Task=G.HashMap.List(task).add(TaskSeqi.tj);
23.            j++;
24.         END WHILE
25.      END IF
```

26.　END WHILE
27. END FOR

在算法 2 的第 12 行中，根据函数 $equalT(G.T.t_k, TaskSeq_i.t_j)$ 判断两个任务是否相同，判断规则如下。首先，判断两个任务的所在界面（Activity）和用户操作（Operation）是否相同；然后，判断两个任务用户操作触发时的界面状态（RT）是否相似，Activity 界面状态（RT）的相似度计算方法[24]如式(1)所示。当且仅当两个任务的所在界面和用户操作相同、用户操作触发时的界面状态相似时，两个任务相同。

$$sim(RT_k,RT_j)=\frac{\sum_{l=1}^{N}\sum_{q=1}^{M_l}\sum_{r=1}^{M_j}equalE(e_{k,l,q},e_{j,l,r})}{\max(N(RT_k),N(RT_j))} \tag{1}$$

其中，l 表示元素所在的层数，$e_{k,l,q}$ 表示 RT_k 中第 l 层的第 q 个元素，$e_{j,l,r}$ 表示 RT_j 的第 l 层节点的第 r 个元素。函数 $equalE()$ 用于判断两个元素是否相同。逐层计算 RT_k 和 RT_j 之间的相同元素的数量，遍历 RT_j 的第 l 层，寻找是否存在元素 $e_{j,l,r}$ 与 RT_k 中的元素 $e_{k,l,q}$ 相同，若存在，则 $equalE(e_{k,l,q},e_{j,l,r})=1$，表示 RT_j 中存在 $e_{k,l,q}$ 的相同元素。然后，将 RT_k 和 RT_j 之间的相同元素的数量除以两者元素数量的较大值，得到 RT_k 和 RT_j 之间的相似度。当相似度大于阈值时，认为两个界面状态相同（equalRT），本文根据使用经验，将界面状态相似度的阈值设为 0.5。

在比较界面状态相似度的过程中，需要判断两个元素是否相同，判断规则如下。首先，判断两个元素在运行时软件体系结构中的位置（path）和类名（classname）是否相同；然后，判断两个元素的外观是否相似，元素外观的相似度计算方法如式(2)所示。当且仅当两个元素位置和类名属性相同、外观相似时，两个元素相同。

$$\begin{aligned}sim(e_{k,l,q},e_{j,l,r})=&\frac{\min(e_{k,l,q}.x,e_{j,l,r}.x)}{\max(e_{k,l,q}.x,e_{j,l,r}.x)}*p_1+\\&\frac{\min(e_{k,l,q}.y,e_{j,l,r}.y)}{\max(e_{k,l,q}.y,e_{j,l,r}.y)}*p_2+\\&\frac{\min(e_{k,l,q}.width,e_{j,l,r}.width)}{\max(e_{k,l,q}.width,e_{j,l,r}.width)}*p_3+\\&\frac{\min(e_{k,l,q}.height,e_{j,l,r}.height)}{\max(e_{k,l,q}.height,e_{j,l,r}.height)}*p_4\end{aligned} \tag{2}$$

该公式是 4 个部分的加权求和，4 个部分分别计算了元素 $e_{k,l,q}$，$e_{j,l,r}$ 的横坐标、纵坐标、宽度和高度的相似程度。p_i 是它们的权重，根据经验进行设定，本文中 p_1，p_2，p_3，p_4 均设为 0.25。当相似度 sim 大于阈值时，认为两个元素相同，本文根据使用经验，将元素相似度的阈值设为 0.6。

5　测试用例驱动的用户行为工作流获取

本文提出了基于模拟用户行为的 API 建模与执行方法。在 Activity 界面运行时软件体系结构基础上，实现用户操作的模拟执行；根据指定功能的用户行为工作流，生成对应的 API；通过模拟用户行为的方式进行应用内功能的调用。

5.1　API 的建模

根据指定功能的用户行为工作流，建立对应的 API。API 的定义如下。

定义 5（应用程序接口）　应用程序接口表示为 $API=\langle Params,G\rangle$，其中 $Params$ 表示 API 的输入，即用户行为工作流 G 中所有输入操作参数的集合（见定义 2），表示为 $Params=\{\langle param_1:input_1\rangle,\langle param_2:input_2\rangle,\cdots,\langle param_n:input_n\rangle\}$，其中，$input_i$ 表示 G 中第 i 个 $setText$ 操作的输入参数 $param_i$ 的值；G 表示 API 对应的指定功能的用户行为工作流。

5.2　API 的执行

在 Activity 界面运行时软件体系结构的基础上，基于计算反射机制，实现单个用户操作的模拟执行。操作 $Operation$ 的执行方式如下：首先，获取运行时软件体系结构中根节点 e_1 在系统中对应对象的引用 $e_1.ref$；其次，根据操作作用的元素 e 的所在路径（$Operation.path$），找到元素 e 在系统中对应对象的引用 $e.ref$；最后，基于计算反射机制，调用 View 类实例的 dispatchTouchEvent 方法或 TextView 类实例的 setText 方法分别实现用户操作 $onClick$ 或 $setText$ 的模拟执行。

根据 API 对应的用户行为工作流，通过模拟用户行为的方式，调用应用内功能。API 执行算法如算法 3 所示，其输入为一个 API 调用实例，$task$ 指向 API 工作流中的任务，$tList$ 存放 $task$ 的所有后继任务。算法流程如算法 3 所示。

（1）初始化任务链表 $tList$（见第 1 行）。将 API 工作流的第一个任务加入到任务链表 $tList$ 中。

（2）监听 Activity 界面是否发生变化并重建其软件体系结构（见第 3－5 行）。当 Activity 界面发生变化时，根据当前 Activity 的引用 ref_{root}，使用算法 1 重建其运行时软件体系结构 RT。

（3）根据触发条件判断待执行的用户操作（见第 6－7 行）。遍历后继任务链表 $tList$，判断当前 Activity 的界面状态 RT 是否与任务 $api.G.T.t_k$ 的触发条件 $t_k.RT$ 相同。若相同，则执行步骤 4。

（4）用户操作的模拟执行（见第 8－15 行）。将 $task$ 指向 $api.G.T.t_k$，并判断 $task$ 的用户操作类型。若为输入操作，则从 API 的 $Params$ 中取出对应的 $input$ 作为输入参数，执行该操作。若为点击操作，则输入参数为 nul，执行该操作。然后，更新任务链表 tList。

（5）重复执行第 2－4 步，直到跳转并停留在目标 $Activity$，此时 tList 为空。

算法 3　API 生成

```
输入：API api
输出：go to the target activity
定义：
G：the user-behavior workflow.
task：the pointer that points to the current task in the API workflow.
tList：the set that contains all subsequent tasks of task.
RT：the runtime software architecture.
1. tList. add(api. G. T. t1);
2. i=1;
3. WHILE(true) DO
4.    IF(Activity is changed) THEN
5.       RT =算法 1(refroot);
6.          FOR(G. T. tk in tList) DO
7.          IF(equalRT(RT,api. G. T. tk. RT)=1) THEN
8.             task=api. G. T. tk ;
9.             IF(task. operation. type=setText) THEN
10.               performOperation(task. operation,api. Params. inputi);
11.               i++;
12.            ELSE IF(task. operation. type=onClick) THEN
```

```
13.            performOperation(task.operation,null);
14.          END IF
15.          tList=api.G.HashMap.List(task);
16.        END IF
17.      END IF
18.      IF(tList=null) THEN
19.        break;
20.      END IF
21.    END IF
22. END WHILE
```

6 方法评估

本节首先以“QQ音乐”应用中播放音乐功能为例对本文方法开展实例研究，然后从适用范围、API开发难度和API执行性能3个方面对方法进行评估。

6.1 实验设置

本文从“豌豆荚”应用商店的旅游出行、新闻阅读、影音播放、生活休闲、考试学习和健康运动6个类别应用中，分别选择各类别排名前50的应用程序(共300个)作为代表性应用集合，评估本文方法的适用范围。针对集合中的应用程序，当存在功能可以使用本文方法生成对应的API时，认为该方法能够适用该应用程序。分析集合中所有的应用程序，并计算方法的适用比率。

然后，从上述6个类别应用中选取50个重要功能作为代表性功能集合，采用Redmi Note 8 Pro手机(2.05 GHz CPU，8 GB RAM)作为实验设备，评估API的开发难度和执行性能。代表性功能集合包括30个简单功能和20个复杂功能。简单功能指经过1～2次点击操作即可完成的功能，复杂功能指从应用中选取的核心功能，包含输入操作和多次点击操作。

本文选择前期工作Aladdin[4]作为对比方法。给定一个Android应用程序，Aladdin首先使用静态分析来找到如何从应用程序的入口最有效地访问应用程序内的Activity(每个Activity都是Android应用程序的基本UI组件)。在开发人员选择需要深层链接动态位置的Activity之后，Aladdin会执行动态分析，以找到如何在每个Activity中访问片段(每个片段都是UI组件的一部分)。最后，Aladdin综合分析结果，获取跳转到目标Activity界面的深层链接，并生成对应的API。此外，Aladdin提供了一个与应用程序代码集成的深层链接代理来接管深层链接处理，因此不会检测应用程序的原始业务逻辑。

6.2 实例研究

本节以“QQ音乐”应用中播放音乐功能为例，展示本文方法如何生成API。图3给出了该应用内功能的执行过程。首先，打开应用首页，点击搜索栏(t1)，跳转到“搜索”页面；然后，在“搜索”页面的搜索框中输入音乐名(t2)，并点击“搜索”按钮(t3)；最后，若存在该音乐，则跳转到“搜索结果”页面，点击第一个搜索结果(t4)，播放该音乐，否则跳转到“未找到音乐”页面。

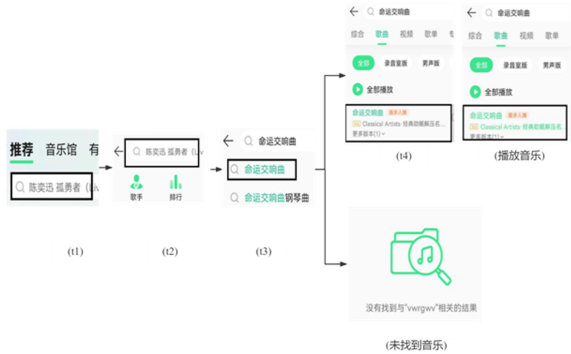


图3 “QQ音乐”应用中播放音乐功能的执行过程

Fig. 3 Execution process for music-play function of QQ music app

使用本文方法将上述应用内功能生成对应的API，其输入参数为“音乐名”，执行API将跳转到目标Activity界面。图4给出了该API的用户行为工作流。t_1判断当前界面是否为应用首页，执行搜索栏的点击操作，执行成功后跳转到“搜索”页面；t_2判断当前界面是否为“搜索”页面，执行搜索框的输入操作并输入“音乐名”，执行成功后仍位于“搜索”页面；t_3判断当前界面是否为“搜索”页面，执行“搜索”按钮的点击操作，若存在该音乐，则跳转到“搜索结果”页面，否则跳转到“未找到音乐”页面(目标Activity界面)；t_4判断当前界面是否为“搜索结果”页面，执行第一个搜索结果的点击操作，执行成功后跳转到“播放音乐”页面(目标Activity界面)。

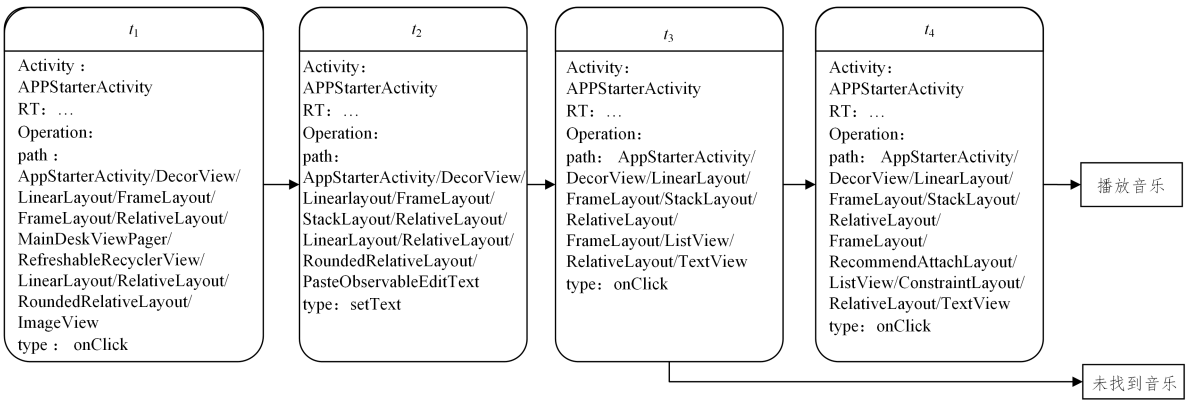


图4 “QQ音乐”应用中播放音乐功能的用户行为工作流

Fig. 4 User-behavior workflow for music-play function of QQ music app

6.3 方法适用范围

为了评估本文方法的适用范围，分析集合中所有的应用程序，并计算方法的适用比率。针对集合中的每个应用程序，选择其中一个功能，使用本文方法生成对应的API并进行

验证。实验结果如表 2 所列，本文方法与 Aladdin 能够适用其中的 280 个应用程序，占所有应用程序的 93%。本文对不适用的应用程序进行分析，发现这 20 个应用存在代码混淆的情况，因此无法通过本文方法生成 API。与 Aladdin 相同，本文方法使用 IC3 和 GATOR 等工具进行代码静态分析，以获取所有 Activity 类和 View 类，能够适用 Android 原生应用，但无法解决代码混淆的情况，因此两种方法均适用于大多数应用程序，适用范围相同。

表 2　本文方法和 Aladdin 的适用范围

Table 2　Scope of application of the proposed method and Aladdin

类别	代表性应用集合	本文方法的适用范围		Aladdin 的适用范围	
		应用数	百分比/%	应用数	百分比/%
新闻阅读	50	44	88	44	88
旅游出行	50	46	92	46	92
考试学习	50	48	96	48	96
影音播放	50	49	98	49	98
健康运动	50	47	94	47	94
生活休闲	50	46	92	46	92
总计	300	280	93	280	93

6.4　API 开发难度

从适用功能和 API 开发时间两方面评估方法的 API 开发难度。图 5 给出了本文方法和 Aladdin 适用的功能，本文方法能够实现代表性功能集合中所有 50 个功能的 API 开发，Aladdin 仅能实现其中 30 个简单功能的 API 开发。Aladdin 能够获取跳转到指定 Activity 界面的深层链接并生成对应的 API，能够实现简单功能的 API 开发；但是，复杂功能需要应用程序本身对输入参数进行处理和转换，而 Aladdin 所生成的 API 执行时并非根据应用程序的原有业务逻辑逐步执行，因此 Aladdin 不能实现复杂功能的 API 开发。本文方法通过模拟用户行为的方式调用指定功能，并生成对应的 API，避免了上述问题。图 6 给出了本文方法和 Aladdin 的 API 开发时间，对于简单功能，本文方法的 API 开发时间约为 15 min，Aladdin 的开发时间约为 20 min；对于复杂功能，本文方法的 API 开发时间约为 20 min，相比 Aladdin，本文方法进一步提高了 API 开发的效率。

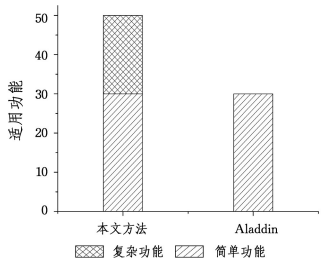


图 5　方法适用功能

Fig. 5　Functions applicable to the proposed method and Aladdin

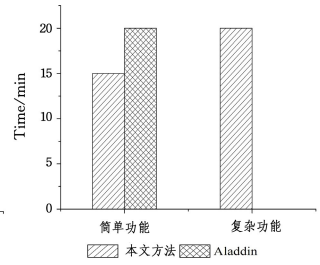
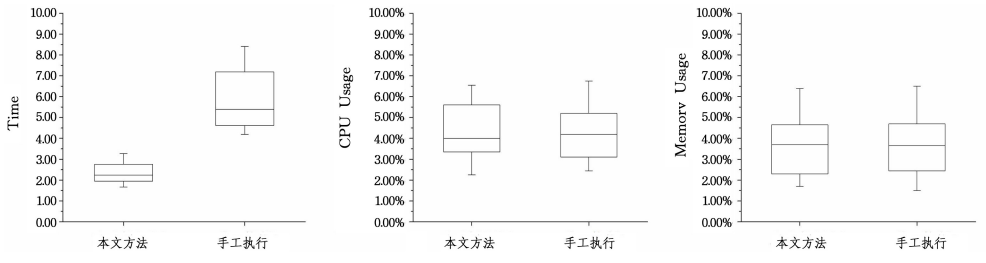


图 6　API 开发时间

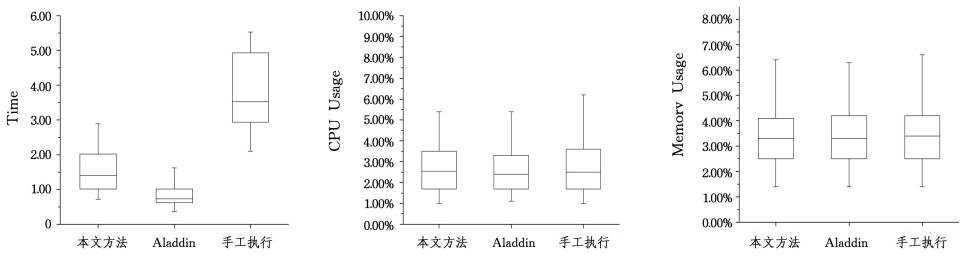
Fig. 6　Development time for API

6.5　API 开发难度

在 Redmi Note 8 Pro 手机上执行本文方法和 Aladdin 开发的 API，以手工执行作为基准，评估 API 的执行性能。图 7 给出了 API 的执行性能，包括执行时间、CPU 使用量和内存使用量。对于简单功能 API，本文方法的执行时间为 0.6～2.9 s，Aladdin 的执行时间为 0.4～1.6 s，手工执行的执行时间为 2.1～5.5 s；对于复杂功能 API，本文方法的执行时间为 1.5～3.2 s，手工执行的执行时间为 4.1～8.3 s，Aladdin 不能实现对应的 API。在执行时间方面，本文方法和 Aladdin 相比手工执行均具有明显优势，而 Aladdin 开发的 API 的执行时间略短于本文方法。本文方法通过模拟用户行为的方式调用指定功能，需要依次在不同 Activity 界面上执行操作，并最终跳转到目标 Activity 界面；Aladdin 则直接跳转到目标 Activity 界面，因此 API 执行时间较短。对于简单功能 API，本文方法的 CPU 使用量为 1.0%～5.3%，内存使用量为 1.4%～6.5%，Aladdin 的 CPU 使用量为 1.1%～5.3%，内存使用量为 1.3%～6.4%，手工执行的 CPU 使用量为 1.0%～6.2%，内存使用量为 1.3%～6.6%；对于复杂功能 API，本文方法的 CPU 使用量为 2.2%～6.5%，内存使用量为 1.6%～6.4%，手工执行的 CPU 使用量为 2.4%～6.7%，内存使用量为 1.5%～6.5%。在资源消耗方面，本文方法、Aladdin 和手工执行没有明显区别。



(a)简单功能 API 的执行性能



(b)复杂功能 API 的执行性能

图 7　API 的执行性能

Fig. 7　Execution performance of API

7 相关工作

深层链接(Deep Link)是移动智能应用领域的一个新兴概念。近年来,一些搜索领域的大型公司在深层链接方面投入了大量精力,并提出了各种深层链接标准。Google App Indexing[25]允许用户点击 Google 搜索结果中的链接,打开 Android 或 iOS 设备上对应的应用。Bing App Linking[26]将应用与 Bing 在 Windows 设备上的搜索结果相关联。Facebook App Links[27]是一个开放式跨平台的解决方案,能够通过深层链接跳转到第三方应用的指定内容页面。但是,这些解决方案都要求应用程序具有相应的网页,才能实现其深层链接,限制了它们的应用范围。Azim 等[7]设计并实现了 uLink,其是一种生成用户定义的深层链接的轻量级方法。uLink 被实现为一个 Android 库,开发者可以使用 uLink 重构应用程序,并生成指定 Activity 的深层链接。此外,一些研究者提出了基于记录和回放生成指定 Activity 深层链接的方法[28-30]。前期工作中,我们提出了一种实现深层链接的方法 Aladdin[4],给定 Android 应用以及指定 Activity,通过静态分析和动态分析就能获取指定 Activity 的深层链接。尽管上述方法能够通过深层链接跳转到应用中的指定 Activity,但对于需要经过多个 Activity 才能完成的复杂功能,则没有提供良好的外部调用支持。

在应用程序接口生成技术方面,Canfora 等[12-13]针对传统表单系统提出了一种 Web 服务发布方法,采用逆向工程技术理解和建模人机交互过程来构造包装器,以实现遗留系统接口到 Web 服务的转换。Rodríguez-Echeverría 等[9]针对 Web 应用提出了一种模型驱动的 API 生成方法,采用逆向工程技术对遗留 Web 应用建模,并进一步生成 REST API。Zhang 等[11]针对工业应用提出了一种基于计算反射的 API 生成方法,采用逆向工程技术对遗留工业应用建模,在此基础上开发指定功能的 API。前期工作中,我们针对 C/S 架构系统提出了一种 API 生成方法[14],重构系统客户端程序,采用逆向工程技术分析客户端运行时的数据流和控制流,通过模拟用户行为的方式进行后台数据读写和功能调用,从而生成对应的 API。上述方法针对传统表单系统、Web 应用、工业软件等应用类型,采用逆向工程技术支持从遗留应用生成 API,本文则研究 Android 应用的 API 自动生成方法。

北京大学团队在运行时软件体系结构理论及构造方法方面进行了研究[17-19]:给定系统元模型与一组管理接口,SM@RT 工具[20]可以自动生成代码,在保证性能的前提下实现模型到管理接口的映射;当系统元模型发生变化时,SM@RT 可以自动生成新的映射代码。在此基础上,本文建立 Activity 界面运行时软件体系结构,以监听界面状态。

结束语 智能应用提供了健康、旅游等涵盖生活和工作的丰富功能,这些功能不仅能满足使用者的需求,还可以被进一步发布成 API 供外部调用。本文提出了一种基于计算反射的 Android 应用程序接口自动生成方法。该方法能够在应用不修改源代码的情况下,基于计算反射机制重建其 Activity 界面运行时软件体系结构;面向应用内功能的测试用例,分析用户行为工作流以及对应的系统调用;通过模拟用户行为的方式调用指定功能,并生成对应的 API。在此基础上,实现了一套 Android 应用程序接口自动生成工具,针对"豌豆荚"Android 应用商店中的 300 个流行应用进行方法评估,结果显示,该方法适用于其中 280 个应用,能够在 15min 左右实现指定功能的 API,并且 API 性能满足外部调用的需求。

本文方法实现了 Android 应用程序接口的自动生成,实验结果验证了本文方法的有效性,但本文方法在实际应用中仍存在两方面问题。1)在实际应用中,本文方法针对应用内的指定功能,通过人工手动执行该功能来获取测试用例,在此基础上分析用户行为工作流,并生成对应的 API。因此,获取正确和完整的测试用例集合,是进一步生成正确 API 的前提。一个测试用例是从应用首页开始到目标页面为止的一次执行过程,不同的测试用例能够描述该功能不同的执行路径,完整的测试用例集合则需要覆盖该功能所有可能的执行路径。例如,实例研究中"QQ 音乐"应用的播放音乐功能包含"播放音乐"和"未找到音乐"两种执行路径,测试用例集合需要包含能够覆盖这两种执行路径的测试用例。大多数的应用内功能相对简单,其执行过程往往仅包含若干次用户操作和少量执行路径,获取完整的测试用例集合也相对容易;对于少数复杂的功能,则需要结合现有测试用例获取方法[31-32],以获取完整的测试用例集合。2)智能应用中存在丰富的用户操作,包括点击、长按、滑动、多点触控、输入等操作类型,其中最常用的是点击和输入操作。本文方法对点击和输入操作进行建模,实现两种操作的监听和模拟执行,能够支持大多数应用内功能的 API 生成。对于其他类型的用户操作,同样可以通过 View 类实例的 dispatchTouchEvent 方法实现其监听和模拟执行,在未来的工作中,我们拟进一步对这些操作进行建模,使建模方法更加完备,以支持所有应用内功能的 API 生成。

参 考 文 献

[1] Mobile Internet Use Passes Desktop[EB/OL]. https://techcrunch.com/2016/11/01/mobile-internet-use-passes-desktop-for-the-first-time-study-finds.

[2] WANG H Y, LIU Z, GUO Y, et al. An Explorative Study of the Mobile App Ecosystem from App Developers' Perspective[C]// Proceedings of the 26th International Conference on World Wide Web. Switzerland: WWW, 2017: 163-172.

[3] Activity[OL]. https://developer.android.google.cn/guide/components/activities/intro-activities?hl=zh_cn.

[4] MA Y, HU Z N, LIU Y Z, et al. Aladdin: Automating Release of DeepLink APIs on Android[C]// Proceedings of the 2018 World Wide Web Conference. Switzerland: WWW, 2018: 1469-1478.

[5] Wandoujia [OL]. https://www.wandoujia.com.

[6] Mobile deep linking[OL]. https://en.wikipedia.org/wiki/Mobile_deep_linking.

[7] AZIM T, RIVA O, NATH S. ULink: Enabling User-Defined Deep Linking to App Content[C]// Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. New York: Association for Computing Machinery, 2016: 305-318.

[8] MAES P. Concepts and Experiments in Computational Reflec-

tion[C]// Proceedings of Object-oriented Programming Systems, Languages and Applications. New York: Association for Computing Machinery, 1987: 147-155.

[9] RODRÍGUEZ-ECHEVERRÍA R, MACÍAS F, PAVÓN V M, et al. Model-Driven Generation of A Rest API from A Legacy Web Application[C]// Proceedings of the 9th International Workshop on Model-Driven and Agile Engineering for the Web. Berlin: Springer-Verlag, 2013: 133-147.

[10] JIANG Y, ELENI S. Towards Reengineering Web Sites to Web-Services Providers[C]// Proceedings of the 8th European Conference on Software Maintenance and Reengineering. Piscataway, NJ: IEEE, 2004: 296-305.

[11] ZHANG S, CAI H Q, MA Y, et al. SmartPipe: Towards Interoperability of Industrial Applications via Computational Reflection[J]. Journal of Computer Science & Technology, 2020, 35(1): 161-178.

[12] CANFORA G, FASOLINO A R, FRATTOLILLO G, et al. Migrating Interactive Legacy Systems to Web Services[C]// Proceedings of the 10th European Conference on Software Maintenance and Reengineering. Piscataway, NJ: IEEE, 2006: 24-36.

[13] CANFORA G, FASOLINO A R, FRATTOLILLO G, et al. A Wrapping Approach for Migrating Legacy System Interactive Functionalities to Service Oriented Architectures[J]. Journal of Systems and Software, 2008, 81(4): 463-480.

[14] LIU G G, HU C S M, CHEN S H, et al. Refactoring Java Code for Automatic API Generation[C]// Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain. Piscataway, NJ: IEEE, 2018: 1-6.

[15] BENCOMO N, BLAIR G, FRANCE R. Summary of the Workshop Models@Run. Time at MoDELS 2006[C]// Proceedings of the 2006 International Conference on Models in Software Engineering. Berlin: Springer-Verlag, 2006: 227-231.

[16] BLAIR G, BENCOMO N, FRANCE R. Models@Run. Time[J]. Computer, 2009, 42(10): 22-27.

[17] SONG H, HUANG G, CHAUVEL F, et al. Supporting Runtime Software Architecture: A Bidirectional-Transformation-Based Approach[J]. Journal of Systems & Software, 2011, 84(5): 711-723.

[18] HUANG G, SONG H, MEI H. SM@RT: Applying Architecture-Based Runtime Management of Internetware Systems[J]. International Journal of Software and Informatics, 2009, 3(4): 439-464.

[19] SONG H, HUANG G, WU Y H, et al. Modeling and Maintaining Runtime Software Architectures[J]. Ruan Jian Xue Bao/Journal of Software, 2014, 25(7): 1731-1745.

[20] Peking University. SM@RT: Supporting Models at Run-time [OL]. 2009. http://code.google.com/p/smatrt/.

[21] OCTEAU D, LUCHAUP D, DERING M, et al. Composite Constant Propagation: Application to Android Inter-Component Communication Analysis[C]// Proc of the 37th IEEE/ACM International Conference on Software Engineering. Piscataway, NJ: IEEE, 2015: 77-88.

[22] ROUNTEV A, YAN D C, YANG S Q, et al. GATOR: Program Analysis Toolkit for Android [EB/OL]. [2021-05-12]. http://web.cse.ohio-state.edu/presto/software/gator/.

[23] Android API Reference. Android Developers[EB/OL]. https://developer.android.google.cn/reference.

[24] HE X, XIE Z P. Structural Similarity Measurement of Web Pages Based on Simple Tree Matching Algorithm[J]. Journal of Computer Research and Development, 2007, 44(Z3): 1-6.

[25] Google App Indexing[EB/OL]. https://developers.google.com/app-indexing/.

[26] Bing App Linking[EB/OL]. https://msdn.microsoft.com/en-us/library/dn614167.

[27] Facebook App Links [EB/OL]. https://developers.facebook.com/docs/applinks.

[28] GOMEZ L, NEAMTIU I, AZIM T, et al. RERAN: Timing-and Touch-Sensitive Record and Replay for Android[C]// Proceedings of the 35th International Conference on Software Engineering. Piscataway, NJ: IEEE, 2013: 72-81.

[29] HU Y J, AZIM T, NEAMTIU I. Versatileyet Lightweight Record-and-Replay for Android[C]// Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York: Association for Computing Machinery, 2015: 349-366.

[30] ZHANG J H, SHEN L W, PENG X, et al. MashReDroid: enabling end-user creation of Android mashups based on record and replay[J]. Journal of Science China Information Sciences, 2020, 63(10): 50-69.

[31] KONG P, LI L, GAO J, et al. Automated Testing of Android Apps: A Systematic Literature Review[J]. Journal of IEEE Transactions on Reliability, 2019, 68(1): 45-66.

[32] LAI D, RUBIN J. Goal-Driven Exploration for Android Applications[C]// Proceedings of 34th IEEE/ACM International Conference on Automated Software Engineering. 2019: 115-127.

WANG Yi, born in 1996, postgraduate. His main research interests include Android application service generation and Android application service adaptation.

CHEN Xing, born in 1985, Ph.D, professor, Ph. D supervisor, is a member of China Computer Federation. His main research interests include system software, software self-adaptation and cloud computing.

（责任编辑：喻藜）