南京大學

研 究 生 毕 业 论 文
（申请工程硕士学位）

论　文　题　目　基于知识抽取的功能点测试跨应用复用技术

作　者　姓　名　曹振飞

学科、专业名称　工程硕士（软件工程领域）

研　究　方　向　软件工程

指　导　教　师　陈振宇教授，房春荣助理研究员

2022 年 **5** 月 **14** 日

学　　号：**MF20320008**

论文答辩日期：**2022** 年 **05** 月 **20** 日

指 导 教 师：　　（签字）

Cross-application functionality test reuse technique based on knowledge extraction

by

Zhenfei Cao

Supervised by

Professor Zhenyu Chen, Assistant Professor Chunrong Fang

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering

Software Institute
Nanjing University

May 20, 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于知识抽取的功能点测试跨应用复用技术

工程硕士（软件工程领域）专业 2020 级硕士生姓名：曹振飞

指导教师（姓名、职称）：陈振宇教授，房春荣助理研究员

摘　要

现如今，移动应用在越来越多的方面帮助大众用户便利与丰富其生活内容，各大应用市场上分门别类的软件五花八门。大众用户对移动应用软件的依赖性日趋增加也带来了对于移动应用软件质量的更高的质量保障需求，这其中又以应用功能的可靠性这一指标最为关键。为满足用户增长的使用需求，移动应用软件也需要进行快速的版本功能迭代升级，这无疑给功能测试带来了更大的困难。在尽快进行功能上线的前提下，仅依靠传统的手工测试，有限的测试人员和短暂的时间下往往无法对待测功能进行完备和充分的测试。由此诞生了大量针对移动应用的自动化遍历测试工具与方法。移动应用 GUI 自动化遍历测试是基于特定的遍历策略覆盖式的操作页面上的控件，通过高页面以及控件覆盖率确保测试效果的有效性。尽管传统的移动应用自动化遍历测试能够很好的完成对应用页面上的控件的覆盖，然而目前仍然没有任一遍历策略在遍历过程中体现了特点功能点的操作逻辑性。由于缺乏功能操作逻辑内涵，在穷尽页面组合与控件操作序列的遍历过程中既无法通过高覆盖率确保功能测试充分性，又会在复杂的页面跳转逻辑中产生指数增长的路径爆炸问题从而极其影响测试效率。而与此相对的是，与功能逻辑强耦合的脚本测试方法极其依赖测试人员的经验和编码能力，同时又由于版本迭代等极易失效，同样也是相当低效的测试方法。因此目前急需一种包含特定功能点逻辑内涵的高效自动化测试方法。

基于知识抽取的功能点测试跨应用复用技术创新性地在自动化测试的遍历过程中引入了知识图谱作为测试引导手段。通过大量前期积累的众测报告数据，针对不同功能分别构建知识图谱，基于知识推理等能力使得知识图谱学习到不同功能的操作逻辑。在具体的测试执行的过程中，通过图像识别的技术分析当前页面截图中所含的控件信息及特征，由已构建完成的知识图谱负责识别符合该功能操作逻辑链中的候选操作控件集合，基于图像识别到的控件边框，

利用 Appium 的自动化测试框架完成如点击、输入、滑动等控件操作。经过知识图谱剪枝的操作序列将与记录在图谱中功能执行逻辑路径进行匹配，最终判断应用的特定功能执行逻辑是否完备。

通过分析华为应用商城中的应用信息，本文选取了市面上最热门的功能及其代表性应用。本知识图谱引导的自动化测试工具对这些应用及功能上进行了测试，通过验证对于功能中不同场景的覆盖程度，证明了本工具能够通过扩展和丰富构建好的知识图谱充分和完备的测试应用特定功能。同时对于不同应用的相同功能，我们验证了其测试执行成功率，证明了本工具能够跨不同应用复用该测试方法和能力。以上两点结论都证明了本文技术的有效性，能够很好地补充传统移动应用遍历测试的不足，实现对功能的更完备和充分的测试，其无关应用的特性也大大提高了测试效率，降低了人力干预和脚本修复的开销。

关键词： 功能点测试；知识图谱；移动应用自动化测试；图像理解

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Cross-application functionality test reuse technique based on knowledge extraction

SPECIALIZATION: Software Engineering

POSTGRADUATE: Zhenfei Cao

MENTOR: Professor Zhenyu Chen, Assistant Professor Chunrong Fang

Abstract

Nowadays, different mobile applications help mass users to facilitate and enrich their life. There are various types of mobile applications in major application markets. The increasing dependence of mass users on mobile application has also brought stronger quality assurance requirements for them, among which the reliability of functions is the most critical metric. In order to meet the growing usage demands of users, the mobile application also needs to upgrade version and its function quickly, which undoubtedly brings greater difficulties to functional testing. On the premise of release versions as soon as possible, relying only on traditional manual testing, limited testers and short time are often unable to completely and adequately test the functions. As a result, a large number of automatic traversal testing tools and methods for mobile applications have been created. Mobile application GUI automatic traversal testing is based on a specific traversal strategy to cover the widgets on the activity, and to ensure the test effectivity by high page and widgets coverage. Although the traditional mobile application automatic traversal test can well complete the coverage of the widgets on activities, there still doesn't exist any traversal strategy that reflects the operational logic of the specific functionality in the traversal process. Due to the lack of functional operation logic connotation, in the process of traversing exhaustive page combinations and widgets operation sequences, it is not only unable to ensure the test effectivity of specific functionality through high coverage, and will also causes exponential path explosion problems in the complex page jump logic, which is hugely affect the test efficiency. In contrast, the script testing method is strongly coupled with functional logic and is thus extremely dependent on the experience and coding ability of the tester. At the same

time, script test is very easy to fail due to version iteration, etc. Therefore, there is an urgent need for an efficient automatic testing method that can reflect the operational logic connotation of specific functionality.

The cross-application functionality test reuse technique based on knowledge extraction innovatively applies knowledge graph as a test guidance method in the traversal process of automatic testing. Through a large amount of accumulated public testing report data accumulated, knowledge graphs are constructed for different functions, which can learn the operation logic of different functions based on knowledge reasoning and other capabilities. In the process of specific test execution, the image recognition technology is used to analyze the widgets information and features contained in the screenshot of the current activity, and the constructed knowledge graph is responsible for identifying the candidate widgets set to be operated, which exist similar objects in the logical chain of the function operation stored in knowledge graph. The test tool uses Appium's automatic testing framework to complete widgets operations such as click, input, and slide based on the frames of widgets recognized from screenshots. The operation sequences pruned by the knowledge graph will match the function execution logic path recorded in the graph, and finally determine whether the specific function execution logic of the application is complete.

By analyzing the application information on Google Play, this paper selects the most popular functions and their representative applications on the market. The automatic testing tool guided by the knowledge graph has tested these applications and its corresponding functions. By verifying the coverage of different scenarios of the function, it proves that this tool can . completely test specific function on different mobile applications by expanding and enriching the constructed knowledge graph. At the same time, for the same function of different applications, we verified its test execution success rate and prove that this tool can reuse the test method and capability across different applications. The above two conclusions have proved the effectiveness of the technology in this paper, which can well complement the shortcomings of traditional mobile application traversal testing and improve the completeness and sufficiency of functionality test by reducing cost of human intervention and test script fixes.

keywords: Functionality test, Knowledge Graph, Mobile application automatic testing, Image understanding

目　　录

插图清单

附表清单

第一章　引言

1.1　项目背景及意义

随着移动互联网的日益普及，大众用户在日常生活中能够接触到各式各样的移动应用，这些应用在购物、支付、阅读、音乐等各方面便利着人们的生活。目前移动应用开发上线数量仍然呈现井喷态势。在全球范围内，持有智能手机的用户人均使用各类应用的每日时长约为 3.7 小时，各类 APP 下载量超过两千亿次 [1]。而在国内，根据工信部统计结果来看，主流移动应用总数高达 367 万，各大应用商店应用下载量达到千亿次 [2]。而在这其中，安卓系统因为其生态丰富，开发简便的优势占到了 87% 的应用市场份额 [3]。

与安卓应用数量井喷相对应的，是移动应用质量保障的迫切性。面对版本迭代快速的应用，传统的手工测试显然无法起到很好的效果，因此如何采取有效的测试方法确保应用功能的高可用性成为了棘手的难题。移动应用自动化测试因此作为一种很好的测试补充手段 [4]。目前应用较为广泛的移动应用自动化测试方法主要分为两类，一类是依托于自动化测试框架的脚本测试，另一类为移动应用 GUI 遍历测试工具。对于脚本测试，测试人员通常需要优先选定依赖的自动化测试框架，如 Appium [5] 等，借助 UIAutomator [6] 等应用页面结构分析工具获取应用当前页面 XML 结构完成对具体控件的定位，通过编写与自动化测试框架对应的测试脚本语句实现对页面控件的操作，编写完成的附带完整控件操作逻辑序列的脚本经由自动化测试框架运行即可进行测试 [7]。如果说脚本测试是面向特定功能点的逻辑性测试，那 GUI 遍历测试则是面向应用整体的覆盖性测试，其通常基于一定的遍历策略，对应用中的页面跳转逻辑，控件组合进行穷尽式的探索，以求寻找到能够触发 bug 的操作逻辑序列，并将所遇到的 bug 以报告的形式呈现给开发人员进行复现和修复 [8]，例如基于深度优先遍历算法的测试工具 GUIRipper [9] 以及基于伪随机策略的 Dynodroid [10] 等。

尽管目前基于自动化测试框架的脚本测试和移动应用 GUI 遍历测试工具都应用的十分广泛，然而他们仍然存在着比较多的缺陷。首先对于脚本测试而言，编写测试脚本本身需要测试人员有一定的专业知识，这为脚本测试设定了

一定的门槛 [11]。同时尽管由于自动化测试框架的存在使得测试脚本有一定的跨平台性，但快速迭代的应用功能和频繁改版的应用页面使得在实际的工业界中测试脚本的时效性可能仅能持续一个到几个版本，无法实现同一测试脚本的长期沿用。而针对性地对测试脚本进行修复使之重新生效的成本有可能甚至比重新编写一份新的脚本更加高昂，因此脚本的失效率与丢弃率非常之高 [12]。其次对于移动应用 GUI 遍历测试工具而言，应用中出现的弹窗和广告、系统权限申请窗口等都有可能成为遍历工具除 bug 外提前终止遍历过程的原因，导致工具的实际控件和页面覆盖率并不理想 [13]。据调研，业界所应用的主流移动应用 GUI 遍历测试工具的覆盖率通常只能达到 50% 到 60% [14]。除此以外，目前工业界和学术界所用到的 GUI 遍历测试工具所采用的遍历策略之中，并没有能够像脚本测试一样体现具体功能的操作逻辑，其所执行过的控件集合序列以及页面跳转关系无法对应功能中的具体场景，因此测试工具所展示出的控件覆盖率和页面覆盖率在功能层面而言有一定的欺骗性，即高控件和页面覆盖率对于具体功能而言并不等同于其测试充分性 [15]。

针对上述提到的脚本测试易失效、复用难和 GUI 遍历测试无法体现应用功能逻辑的问题，本文创新性地在测试执行的过程中引入了知识图谱作为测试引导，提出了基于知识抽取的跨应用复用的功能点测试技术。首先通过众测报告等数据来源，针对不同功能点构建其功能逻辑知识图谱。在具体的功能点测试执行过程中，利用图像识别的技术获取页面中的控件信息，经过知识图谱匹配查询当前页面中能够触发该功能操作逻辑序列的控件对象。通过执行经过知识图谱剪枝过后的页面跳转逻辑和控件组合，基于是否完整覆盖了应用内涵盖的该功能点中的典型场景作为判断应用功能是否完备可靠的依据。本技术为功能点自动化测试开辟了一条有可能的新思路，也是知识图谱和自动化测试相结合的首次实践，希望能为相关科研工作和业界探索提供参考和帮助。

1.2 国内外研究现状

1.2.1 安卓自动化测试研究现状

安卓自动化测试是依托于大规模安卓真机集群，在不同的真机上对指定的应用进行交互，触发如点击、滑动、输入、双击等操作模拟用户的真实行为，同时监听并记录下执行时的页面截屏、堆栈信息、日志信息、应用状态、手机系统状态等信息，由测试人员复查这些信息以判断应用是否出现了异常 [16]。

而安卓自动化测试又可以分为两类，一类是依托于自动化测试框架的脚本测试，另一类是基于一定的遍历策略的 GUI 遍历测试工具。

对于脚本测试而言，目前所采用的主流的自动化测试框架有 Appium [5]、Robotium [17]、Instrumentation、UIAutomator [6] 等，其中又以 Appium 因为其可以跨 Android 和 iOS 平台的特性最为热门。利用自动化测试框架对外暴露的 API，测试人员可以编写相应的测试脚本，对页面上的控件进行定位和模拟用户的操作。通常来说，测试脚本中往往是为完成具备一定逻辑性的操作任务，因此脚本测试可以达到代替手工测试进行功能测试的目的。然而编写测试脚本需要测试人员具备一定的专业技能，非常熟悉自动化测试框架及相应接口。同时，如果需要完成相对复杂的功能的测试，测试脚本会显得非常冗长并且逻辑混乱，这会非常耗费时间和精力。并且由于脚本测试的修复成本高昂，通常来说对于每个版本的应用的每个功能都要独立开发一个新的测试脚本，这导致脚本的失效率非常高，不断废弃并编写新的脚本也带来了大量的重复工作。

GUI 遍历测试工具是作为基于随机策略的 Monkey 测试工具的升级与改进手段出现的。Monkey 测试工具并不尝试理解和分析安卓应用的页面控件结构，仅仅将页面视为像素点的集合，并基于完全随机的方式对页面上的任意位置进行单一点击类操作的模拟，并监控当前应用的状态是否出现异常或闪退等情况。这种类黑盒测试的方法并未能对应用控件做进一步的覆盖，测试的过程也不包含任何操作逻辑，因此 Monkey 测试工具通常是被应用在压力测试中 [18]。而在基于随机策略的 Monkey 测试工具之后，陆续出现了更多采用基于优先状态机的遍历策略的 GUI 遍历测试工具 [19]。

A. Machiry 等人提出了用于为未修改的 Android 应用程序生成相关输入的工具，Dynodroid [10]。Dynodroid 将应用程序视为事件驱动程序，它通过 Android 框架与一系列事件及其环境进行交互。通过一劳永逸地检测框架，Dynodroid 以轻量级的方式监控应用程序对每个事件的反应，并使用它来指导应用程序生成下一个事件。Dynodroid 还允许将来自机器的事件与更善于提供智能输入的人类事件交织在一起，用以生成大量简单的输入。D. Amalfitano 等人提出了一种通过图形用户界面 (GUI) 测试 Android 应用程序的工具 AndroidRipper [9]。其基于用户界面驱动的 ripper，自动探索应用程序的界面，旨在以结构化方式运行应用程序，能够检测到底层代码中的错误。T. Azim 等人提出了 A3E [20] 的测试工具，能够在实际手机上运行时系统地探索大量 Android 应用程序，而无需访问应用程序的源代码。其通过一种新颖的方式对应用程序字节码使用

静态的、污点式的数据流分析，以构建捕获应用页面之间合法转换的高级控制流图，并使用该图来应用基于目标的探索策略，快速、直接地探索页面。该工具还同时引入了深度优先探索的策略，模仿用户行为以更慢但更系统的方式探索页面及其组成部分。W. Yang 等人提出了一种新颖的灰盒测试工具 ORBIT [21]，用于自动提取给定移动应用程序的模型，使用静态分析提取应用程序的图形用户界面支持的事件集，然后通过动态抓取在运行的应用程序上系统地执行这些事件，对应用程序的模型进行逆向工程。Xiujiang Li 等人提出了一种新颖的用户引导自动化技术 [22]。该技术通过记录用户引导的应用程序执行将应用程序重播到某些停止点，从这些停止点系统地探索状态空间，利用用户引导补充自动化测试技术。

目前的脚本测试方法依然存在着失效率高，复用难的问题，这主要是由于应用迭代更新快，页面结构变化所导致的。这启示了本文不拘泥于应用不同版本的具体页面结构中，而是利用图像理解的手段分析应用页面控件信息，仅将自动化测试框架作为测试执行的手段，由此确保了测试方法的通用性和可复用性。而现有的各类 GUI 遍历测试工具的遍历策略，并未能融入具体功能的执行逻辑，因此然而只能作为整体性的应用覆盖性测试，无法聚焦到功能粒度。这促使本文引入知识图谱作为测试引导，细化测试过程，利用知识图谱和图像理解的能力将自动化测试应用在功能测试范畴内。

1.2.2 知识图谱研究现状

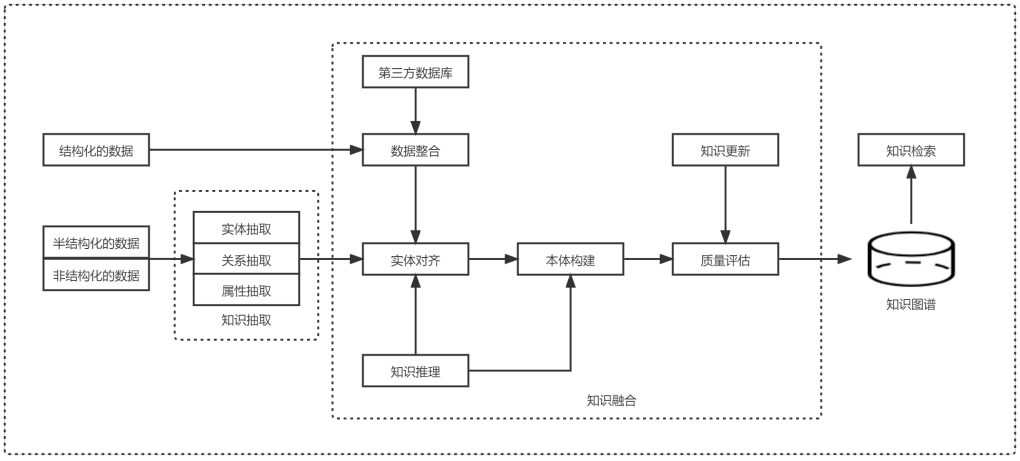

图 1-1: 知识图谱体系结构

知识图谱最早在 2012 年 5 月 17 日由 Google 正式提出，其最早是用于优化搜索引擎查询结果，提高搜索质量和用户体验。知识图谱是人工智能快速发展的一大产物，相较于人工智能利用统计知识以概率解释世界，知识图谱更具备分析和推理能力。知识图谱目前被广泛应用于信息检索、自然语言理解、问答系统、推荐系统、金融风控等领域中 [23]。知识图谱是一种具有很强表达能力和建模灵活性的语义网络，能对现实世界实体和关系进行建模 [24]。知识图谱同样也是一种数据建模的协议，通过知识抽取、知识推理、知识检索等能力对所存储的知识间的关联进行挖掘。知识图谱体系结构如图 1-1所示。

知识图谱其本质为揭示实体之间关系的语义网络。知识图谱有两种构建方法，分别为自顶向下和自底向上。自顶向下指的是先定义好知识图谱的数据模式，再将与之对应的实体嵌入到图谱之中，典型的如 Freebase [25] 这样的从维基百科中获取的通用知识库。更为主流的构建方法是自底向上，即从通用数据源中提取出实体对象，筛选出其中可信度高的信息加入到知识库中再尝试挖掘出实体与实体之间的联系。知识图谱的逻辑结构是由模式层与数据层构成的。数据层由一定的数据结构表示的知识构成，例如可以以实体、关系、属性类似的三元组表征，这类数据结构通常来说使用图数据库存储的。主流的图数据库包括 Neo4j [26]、FlockDB [27]、JanusGraph [28] 等。模式层基于数据层，主要用于规范数据层的知识表示。对于知识库的构建方法，往往需要多种技术的共同协作，这其中又以知识抽取、知识融合、知识推理、知识表示最为关键。

知识抽取指的是从开放数据源中利用自动化的方式归纳抽取目标知识结构的能力。知识结构中往往包括知识实体、知识关系以及知识属性这三大元素。具体到实体抽取，需要应用到自然语言处理中的命名实体识别技术 [29]，从大范围的杂乱语料中自动识别出命名实体，这是整个知识库构建中最为重要的一环，其识别的准确率将直接影响构建完成的知识库的有效性。关系抽取主要是利用实体间的关系模型搭建实体与实体之间的语义链接。

知识融合指的是对构建完成的知识库进行去重融合的步骤。由于开放数据源的数据质量不可控，刚构建完成的知识库往往存在知识冗余、关系不清晰的问题，因此为了提升构建完成的知识库的质量，知识融合是一个必要的步骤。通过不断的对库中的知识进行更高抽象层次的融合、消歧、验证更新，可以令知识库中的知识不断地进行迭代更新、与时俱进。

知识表示指的是利用深度学习技术，将知识实体表示为低维的数值向量，从而能够通过在向量空间中进行数值计算代表知识间关系和复杂语义关联的构

建与推理。Antoine Bordes 等人提出了以三元组（实体，关系，实体）表征知识的词嵌入模型 TransE [30]。Zhen Wang 等人基于 TransE 模型，为解决其在处理自反、一对多、多对一和多对多关系方面的不足提出了 TransH 模型 [31]，它将关系建模为超平面以及对其进行平移操作总而更好地保留上述关系的映射特性。Yankai Li 等人又在 TransE 和 TransH 的基础上提出了 TransR 模型 [32]，其在单独的实体空间和关系空间中构建实体和关系嵌入，再将实体从实体空间投影到相应的关系空间，然后在实体的投影之间建立联系来学习嵌入。

尽管在自动化测试和功能测试领域并没有知识图谱的应用，但考虑到功能测试领域的各功能点内部的强逻辑性以及功能点之间的依赖，属性多元化、逻辑关系强的功能测试适合应用知识图谱。

1.3 本文主要工作

针对上述研究现状，为综合性地解决脚本测试复用难、失效率高以及 GUI 自动化遍历测试缺乏功能逻辑的问题，在参考了知识图谱对于功能测试的帮助以及评估了其可行性之后，本文提出了基于知识抽取的功能点测试跨应用复用技术。该技术利用知识图谱构建的知识库作为测试引导在自动化遍历测试中引入了脚本测试的逻辑性，应用图像理解技术自动识别待操作的控件代替了依赖页面结构的模式从而保证其不会随着页面结构的变化而轻易失效。具体来说本文主要通过以下几个方面来实现目标：

Appium 服务链接：本文基于 Appium 自动化测试框架进行控件操作。通过与对应的安卓设备进行链接并启动相应的 Appium 服务，利用控件在页面上的位置调用相应 Appium 的接口进行例如点击、滑动、输入等控件操作。

页面信息提取：本文利用图像理解技术，对于每一步操作执行后的页面状态进行分析，主要包括布局识别、控件识别、文字提取等方式。布局识别是指对于页面截图，利用泛洪填充算法分析其横向和纵向的页面构成，划分相应的水平和垂直布局框。控件识别是指基于布局识别出的布局框，再利用边缘识别算法等图像分析方法，获取控件轮廓的位置信息。文字识别是指利用 OCR 的方法对于识别到的控件再分别获取其内部的文本用于理解控件的功能。最终整合识别到的页面布局、控件、文本形成结构化信息。

知识图谱引导测试执行：本文创新性地在自动化遍历测试的过程中引入知识图谱作为测试引导，对待测页面中的控件元素进行剪枝，通过对页面信息提

取中得到的控件信息与知识库中的功能逻辑序列进行匹配与检索，筛选出可触发功能逻辑的候选控件集合，利用 Appium 的相关接口进行相应具体操作。

深度优先遍历：对于知识图谱检索完成获取到的控件集合，每一个所返回的控件将对应一条功能逻辑分支。通过在全局信息中记录的路径分支情况，将采用深度优先遍历算法，对于每一功能逻辑分支优先完成整体遍历，再回溯到分叉节点继续其余功能分支的测试执行。最终应用的该功能可用性测试结果将结合具体各个功能逻辑分支的测试执行情况判断。

为验证和评估该测试方法的效果，本文收集了安卓应用市场当中最为主流和热门的功能及其相关应用，利用其中一定数量的应用进行相应功能点的知识图谱构建并在余下的应用上验证本测试方法的有效性。本文引入了功能点中的场景的定义，利用场景覆盖率确保其对功能测试的充分性，同时对于具备同一功能的不同应用，本文验证了该工具跨应用执行的能力，证明本工具的测试过程和能力支持在不同应用之间进行复用。

1.4 本文组织结构

本文共分为六个章节，主要组织结构如下。

第一章为引言部分。主要介绍了安卓自动化测试、知识图谱地国内外研究现状，介绍了脚本测试和 GUI 自动化遍历测试分别在复用性和功能逻辑性方面的问题，引出了本文的技术及主要工作。

第二章为相关技术概念介绍。介绍了项目后端框架 Django，前端框架 Vue，安卓自动化测试框架 Appium，ADB，UIAutomator，Neo4j 图数据库，以及对于项目中所应用的图像理解算法如 SIFT、泛洪填充等进行了介绍。

第三章为本系统的需求分析与概要设计。首先描述了系统整体的架构和大体流程。接着对系统的各个功能模块进行了划分，从系统架构、逻辑视图、开发视图、物理视图等方面，使用核心类图、流程图对各个模块进行了详细设计。最后对项目数据库中的实体、数据表和字段进行了设计。

第四章在第三章的基础上介绍了系统的具体实现。对第三章中划分的各个模块，使用顺序图详细介绍了其调用逻辑，对其中的关键代码进行了解释并附上系统的实现截图。

第五章为实验验证部分，详细解释了实验的设计过程和具体设置，对本工具进行了全面的评估，并对实验结果进行了深入分析。

第六章为总结和展望，对本文及本工具中的各项工作和贡献进行了总结，并且对于工具仍然存在的不足进行了分析，提出了未来有可能的改进方向。

第二章　相关技术概述

2.1　安卓自动化测试相关技术

2.1.1　安卓自动化测试框架 Appium

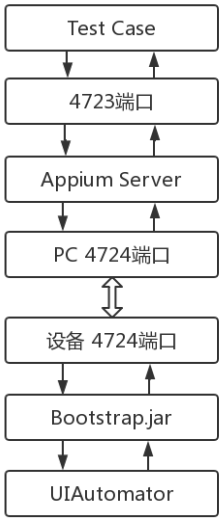

图 2-1: Appium 执行原理

Appium 是一个基于 node.js 的可应用于 IOS 设备、安卓设备、Windows 桌面应用、移动端 Web 应用以及混合应用的开源测试工具 [33]。其跨平台性能够使得测试人员在不同测试端复用同一套测试代码。Appium 使用并扩展了 WebDriver 的 json wire 协议用于驱动苹果系统和安卓系统的 UIAutomator 库。Appium 的另一大特性是能够同时支持多语言，例如 python、java、ruby、js 等，甚至 SeleniumWebDriver 相关 API 同样可以。而反观 Apple UIAutomator 仅能够使用 javascript 编写测试脚本，安卓 UIAuotmator 紧跟能够给使用 Java 编写测试脚本，这两者的局限性明显更大。Appium 的支持多语言特性在底层是通过 Client/Server 的设计模式实现的，因为客户端可以将任意语言实现的测试请

求通过 http 请求发送给服务端 [34]。由于其出色的跨平台和支持多语言的特性使得 Appium 已经成为了目前业界最为主流的自动化测试框架。

Appium 的运行原理如图 2-1所示。首先服务端会启动 Appium 服务，服务启动后将会监听 4723 与 4724 两个端口。其中 4723 端口负责监听客户端请求，4724 端口负责与已连接的设备通讯。客户端会将脚本请求以 json 数据格式通过 http request 发送给服务端。该请求参数中的 Desired Capabilities 字段通常为需要进行操作的上下文信息，例如本次操作的对应设备为移动端还是浏览器端、待测的 app 信息等。服务端即是通过该信息将测试设备与待测对象进行匹配，随后客户端将会创建一个 session 用于与服务端进行后续的通讯与请求传递。session 的作用为保持当前设备的状态信息，在整个会话期间设备与程序的联系不会断开，所以不用重复传递配置信息。当整个测试过程全部完成之后，当前 session 和所有关联窗口才会关闭，同时整个测试进程也会被终止。

在创建 session 成功之前 bootstrap.jar 就已植入安卓设备之中并开启了设备上的基于 Appium bootstrap 的 socket 服务，绑定本机以及 bootstrap 的端口号为 4724。服务端解析脚本请求后发送到 4724 端口，并向 socket 服务发送请求，bootstrap 会再将解析好的 Appium 命令转换为 UIAutomator 命令来让最底层的 UIAutomator 进行处理。socket 接收到请求之后会将响应结果返回给脚本。

本文所采用的测试执行底层就是基于 Appium 完成的，获取到的控件轮廓信息被 Appium 用于定位具体的待操作控件，并进行后续的具体操作。

2.1.2 ADB

ADB(Android Debug Bridge)，即安卓调试桥，是安卓提供的用于在电脑端与模拟器或真机设备间进行交互与通信的命令行窗口 [35]。其底层架构为 Client/Server 模式，分为三部分组成 [36]。第一部分为运行在 pc 端的 adb 客户端，其主要作用为运行 adb 命令。adb 程序首先会尝试定位主机端的 adb 服务器，如果找不到则将自动新启动 adb 服务。当安卓设备与 pc 端 adb 服务器建立连接之后，adb 客户端就可以向服务端陆续发送请求。第二部分为运行在 pc 端的 adb 服务端，其主要作用为检测 USB 端口设备的连接与断开以及模拟器的启动和关闭。adb 服务端会将从 adb 客户端接收到的请求通过 usb 或 tcp 的方式发送到对应的 adb 上。第三部分为运行在安卓设备上的 adb 守护进程，其主要作用为连接 adb 服务器，并且为运行在主机上的客户端提供一些服务。

本文采用 ADB 工具主要用于在安卓设备上执行各类命令，获取安卓设备

当前信息，如截图、页面 XML 结构等。

2.1.3 UIAutomator

UiAutomator 是谷歌在 Android4.1 版本发布时推出的一款用 Java 编写的 UI 测试框架，基于 Accessibility 服务。其最大的特点就是可以跨进程操作，可以使用 UiAutomator 框架提供的一些方便的 API 来对安卓应用进行一系列的自动化测试操作，如点击、滑动、键盘输入、长按以及常用的断言方法等。UIAutomator 测试框架所提供的 API 可用于构建在用户应用和系统应用上执行交互的界面测试，因此非常适合用来编写黑盒式自动化测试，此类测试的测试代码不依赖于目标应用的内部实现细节。UIAutomator 测试框架还可以用于检查布局层次结构以及检索状态信息并在目标设备上执行相应操作。

UIAutomator 还同时提供了一个用于分析应用当前页面结构的图形工具界面 UIAutomatorviewer。该工具能够扫描并分析安卓设备上当前显示的界面组件，利用工具所得到的布局层次结构与页面控件的属性可以进行更加精细的测试。UIAutomatorviewer 工具能够帮助测试人员进行控件的定位，其中定位方式包括利用控件 id 定位、利用控件上的文字进行定位、利用控件所属类进行定位、利用控件所属包定位、利用控件描述信息定位，这些定位方式都有其相应的定位 API 可供调用。该工具同样可以对得到的应用页面当前截图与分析得到的 XML 格式的页面布局文件进行持久化保存。

Appium 自动化测试框架底层也是基于 UIAutomator 实现的，本文主要应用 UIAutomator 获取应用页面布局结构辅助图像理解进行控件精准定位。

2.2 知识图谱相关技术

2.2.1 事件知识图谱

知识图谱被广泛应用于增强搜索引擎返回结果，然而当前知识库倾向于表示现实世界的实际状态，却并未能关注动态和时间变化信息。在信息爆炸的当下，仅关注静态信息的知识库将会流失大量的流动信息，构建以事件为中心的事件知识图谱能够更好的获取动态知识，有效地访问和分析大规模的以事件为中心的时序信息，促进知识图谱信息更新和知识网络扩展 [37]。

传统知识图谱表示以事件为中心的知识存在着许多难点 [38]。首先事件表

示和时序关系分布在异构信息源中，如 Wikidata、DBpedia 和 YAGO 等知识图谱通常以实体为中心，因此这类大规模数据源中以事件为中心的信息无法被清晰的表示。现有知识图谱中对事件的描述并不完整，缺乏如时间和空间信息等关键属性。这类事件和时序关系结构化表示的缺失将会阻碍其应用。此外采用语义技术从非结构化异构信息源中提取事件本身可能也存在着很高的噪声。事件知识图谱的提出则能很好地代替传统知识图谱并解决上述问题。

事件知识图谱通过简单事件模型（SEM）对事件进行建模 [39]。SEM 提供了一种通用的事件表示，包括事件的主题、地理信息、时间维度以及参与者，即参与事件的实体间的连接。事件知识图谱中的时间关系将分为事件-实体关系、实体-事件关系和实体-实体关系三类。事件-实体事件关系用于将事件与参与者进行联系，而实体间时间关系则用于捕获相关事件信息。参与者实体或事件的存在关系同样可以用来进行推理，以估计相关事件的未被明确提供的时间有效性。SEM 中的另一应用是衡量事件和实体之间的关系强度与事件流行度 [40]，可以通过一个实体描述引用另一个实体的频率以及外部信息源中引用的关系数计算得到。图 2-2所示为简单事件模型的实例。

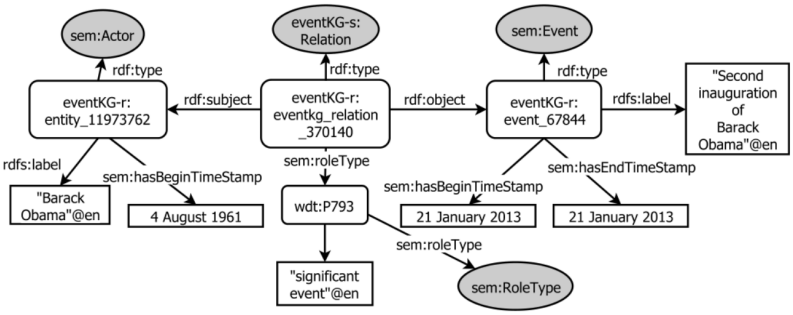

图 2-2: 简单事件模型实例

事件知识图谱的构建过程将分为信息输入、事件识别和提取、事件和实体关系提取、集成、融合以及图谱输出 [41]。事件知识图谱使用额外的事件识别启发式来提高事件识别的召回率。具体来说，其在参考源中传播了有关已识别事件的信息，其次使用依赖于语言的正则表达式与相关事件进行匹配。在事件和实体关系提取中，主要提取以下几类 [38]：基于时间有效性信息的有效性来识别事件关系、所有涉及事件的关系以及已知存在时间的实体之间的关系、其他事件和实体关系。通过为包含至少一个事件的每对互连实体，提取量化关系强度和事件受欢迎程度的信息。对于集成的阶段，从引用源提取的语句包含在命令图中，每个命名图将对应于一个引用源，基于描述、时间和链接的基于规

则的方法对从半结构化源中提取的事件进行集成。对于融合阶段，将使用基于规则的方法融合事件的时间、空间和类型信息。位置融合指从不同参考源中获取其位置的并集并将该集合减小到最小；时间融合是指对于具备已知存在或有效时间的每个实体、事件或关系利用特定规则进行整合。最后事件知识图谱将从单独的命名图中提供从每个参考源提取的信息和融合步骤的结果。

相较于传统知识图谱，事件知识图谱中以事件为中心、包含事件间时序关系的特点更为契合具备功能逻辑性的自动化遍历测试场景，因此本文将使用事件知识图谱替代传统知识图谱构建知识库。

2.2.2 Neo4j 图数据库

Neo4j 是一个高性能的 NOSQL 型图数据库，其能够将结构化的数据存储在图网络而非传统数据表中。Neo4j 同时也是一个嵌入式的、基于磁盘的、具备事务特性的 Java 持久化引擎。Neo4j 具备两种运行方式，一种是对外提供 REST 接口的服务，另一种是嵌入式模式，数据以文件形式直接存放在本地，可以直接对本地文件进行操作。Neo4j 因其嵌入式、高性能、轻量级等优势逐渐成为使用最广泛的开源图数据库。

现实世界中许多数据都是通过图表示的，例如人际关系网络图、地图数据等，传统关系型数据库并不适合用于存储此类数据结构。传统数据库着重表示实体内部属性，实体间关系通过数据库外键表示，关系的求解则要通过复杂的 join 操作完成，这对于大规模数据库而言是一个相当费时且低效的操作。NOSQL 型图数据库则更直观的表示实体间关系也更加轻量级。在 Neo4j 中，一个节点可以拥有一个以上的标签，标签对应着现实世界中的类别信息。节点则是指代具体的对象，节点之间的有向边代表实体间的关系。无论是节点还是节点间的边都可以拥有任意多的属性，属性的表示为 key-value 结构，在 java 中能够很方便的通过 hashmap 进行存储。整个图数据库通过一个个实体之间的连接将会形成一张复杂的数据网络。Neo4j 可以很直观的进行数据的查询而无需考虑数据表与表之间的关系。查询所依赖的图搜索和遍历方法也非常高效，相较于传统数据库能够更快执行查询任务。图 2-3所示为 Neo4j 运行示意图。

鉴于 Neo4j 适用于图形类数据存储以及能够直观表示实体间关系的特点，本文将采用 Neo4j 存储构建完成的知识图谱。Neo4j 中的节点存储的即是知识图谱中的实体，边则对应的图谱中的实体-实体关系。

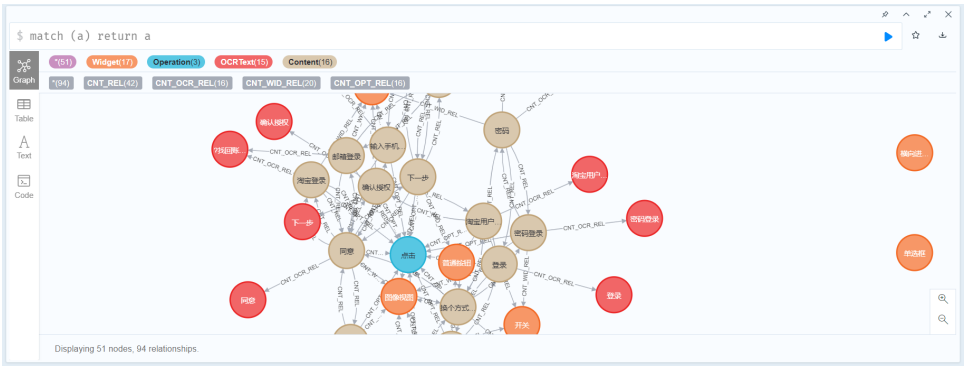

图 2-3: Neo4j 运行示例

2.3 图像理解相关技术

2.3.1 Canny 边缘检测算法

图像的边缘指局部亮度变化显著的部分，该区域会从一个灰度值在很小的缓冲区域内急剧变化到另一个灰度相差较大的灰度值。边缘部分集中了图像的大部分信息，准确地提取图像边缘对于图像的识别任务非常关键。Canny 边缘检测算法最早由 J Canny 提出 [42]，经过了多年的改进 Canny 及其变种依然是目前最为有效的边缘检测算法。Canny 算法将主要由高斯模糊、梯度幅值和方向计算、非极大值抑制、双阈值算法检测和连接边缘 4 个步骤实现。

高斯模糊的主要作用是去除图片噪声。噪声和图片边缘信息一样也集中在高频信号，很容易被错误地认为是图片边缘。选择合理的高斯模糊半径可以去除噪声而保留真正的边缘信息。具体地来说，对于位置为 (m,n) 的像素点，其灰度值为 $f(m,n)$，其经过高斯滤波后的灰度值将变为：

$$g_\sigma(m,n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{m^2+n^2}{2\sigma^2}} \cdot f(m,n)$$

图像边缘可以指向不同方向，因此梯度值以及梯度方向需要由水平方向差分 $g_x(m,n)$ 和垂直方向差分 $g_y(m,n)$ 综合表示，具体地：

$$G(m,n) = \sqrt{g_x(m,n)^2 + g_y(m,n)^2}, \theta = arctan\frac{g_y(m,n)}{g_x(m,n)}$$

图像梯度幅值矩阵中的元素值越大，图像中该点梯度越大，但并不一定就是边缘点。非极大值抑制则是寻找像素点局部最大值，将非极大值点所对应的

灰度值置为 0，可以剔除一大部分非边缘点。非极大值抑制后，会得到一个二值图像，这一结果仍然包含较多噪声及伪边缘，因此仍需进一步的处理。

Canny 算法使用双阈值法减少伪边缘数量。通过选取两个不相等的阈值，即一个高阈值和一个低阈值区分边缘像素点。如果边缘点像素值大于高阈值，则被认为是强边缘点。如果介于高阈值和低阈值之间，则被标记为弱边缘点。小于低阈值的点将被直接抛弃。

本文主要应用 Canny 边缘检测算法提取控件边缘识别其 bounding box 以及分析应用当前页面布局信息。

2.3.2 SIFT 尺度不变特征转换匹配算法

SIFT，即尺度不变特征变换，最早由 DG Lowe 等人提出 [43]，是用于在图像中检测出关键点的局部特征描述算子。SIFT 具有较好的稳定性和不变性，能够适应旋转、尺度缩放、亮度变化等，在一定程度上不受观察者视角、噪声等的影响。SIFT 算法所提取的尺度不变特征具备很好的区分性，能够在大规模检索系统中快速进行匹配。同时所提取出的特征丰富，即是待检图片内容单一，仍然能够产生大量特征向量。SIFT 特征检测的主要分为三个步骤：尺度空间上关键点的提取、特征点定位及方向赋值、特征点描述与匹配。

SIFT 算法本质上是在不同尺度空间上查找关键点并计算其方向。所谓尺度空间指的是通过图像的模糊程度模拟人在观察物体时的角度，即距离越近图像尺寸越大且越模糊。通过使用不同的高斯核与图像进行卷积运算能够得到不同模糊程度的图像，具体地：

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{\frac{x^2+y^2}{2\sigma^2}},L(x,y,\sigma)=G(x,y,\sigma)\cdot I(x,y)$$

其中 σ 为尺度空间因子，为高斯正态分布的标准差，反映图像被模糊的程度。$L(x,y,\sigma)$ 代表图像的高斯尺度控件，$I(x,y)$ 代表高斯核。构建尺度空间的目的是为了检测出不同尺度下都存在的特征点，而较好的算子为差分高斯 DoG。设 k 为相邻两高斯尺度空间的比例因子，则 DOG 的计算方式为：

$$D(x,y,\sigma)=[G(x,y,k\sigma)-G(x,y,\sigma)]\cdot I(x,y)=L(x,y,k\sigma)-L(x,y,\sigma)$$

为寻找尺度空间极值点，每个像素点需要与其同一尺度空间和相邻尺度空间的所有相邻点进行比较。此时所得到的局部极值点是在离散的空间搜索得到

的，由于离散空间的采样特性，并不一定找到所有局部极值点都为真正的极值点。因此还需要通过尺度空间的 DoG 函数进行曲线拟合以去除低对比度的特征点以及不稳定的边缘相应点。为实现图像的旋转不变性，需要给特征点方向进行赋值，利用特征点邻域像素的梯度分布特性来确定其方向参数，再利用图像的梯度直方图求取关键点局部结构的稳定方向。具体来说，每个点 (x,y) 其梯度的模 $m(x,y)$ 以及其方向 $\theta(x,y)$ 的计算方式为：

$$m(x,y) = \sqrt{[L(x+1,y)-L(x-1,y)]^2+[L(x,y+1)-L(x,y-1)]^2}$$

$$\theta(x,y) = arctan\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}$$

得到了每一特征点的位置、尺度、方向信息之后，最后需要使用一组向量来描述该关键点，该描述符既包含特征点本身，也包括特征点周围对其有影响的像素点。该描述符的生成步骤主要为：校正旋转主方向，确保其旋转不变性、生成 128 维的特征向量描述符、对特征向量长度进行归一化处理。

本文主要应用 SIFT 算法提取分析得到的控件图片的尺度不变特征，用于进行相似控件图片的匹配并在知识图谱记录的信息中进行大规模检索。

2.3.3 OCR 光学字符识别

OCR(光学字符识别) 指的是计算机用字符识别的方法识别图像中的文本的技术。经过长时间的技术改进，目前大部分场景下的 OCR 准确率已经非常高了，该技术也广泛应用在证件车牌识别、图片识别、物流分拣、文献资料检索等领域中。目前 OCR 文字识别主要分为印刷体文字识别和手写体文字识别，衡量 OCR 工具性能优劣的指标则主要包括：拒识别率、误识率、识别速度等。尽管在许多简单环境下的 OCR 识别准确度已经比较高，但在一些特定复杂场景下仍然有较大的改进空间，如何降低识别错误率和利用其它辅助信息帮助提高字符识别的成功率仍然是目前 OCR 技术的主要研究方向。

OCR 识别文字的过程主要分为图像处理和文字识别两大步骤。在图像处理阶段，首先会对图片进行灰度化将彩色图片转换为灰度图片并进行二值化。接着利用滤波降噪算法对图像进行降噪，由于图片有可能存在一定的拍摄角度，因此还需要进行图像的倾斜校正到正常的水平位置并按行和列切分字符。切分符完成后就可以利用深度学习技术训练的字符分类器对得到的字符进行识别。

目前业界各大公司都开放了自研的 OCR 接口服务，例如 Google 公司开放

的 Tesseract OCR 以及百度公司开放的 Baidu OCR 服务，并将功能归类为通用识别 OCR 和专用 OCR 分别用于处理普通场景与特定环境下的字符识别。本文选用了 Baidu OCR 高精度通用识别服务，应用在控件图片文本提取的场景中。

2.3.4 CNN 卷积神经网络

卷积神经网络 (CNN) 是一种前馈神经网络，其神经元可以相应一部分覆盖范围内的周围单元，在大尺寸图片处理任务中性能突出。其诞生主要是为解决普通神经网络处理大尺寸图像时将图片展开为向量丢失空间信息、参数过多效率低下训练困难以及大量参数导致网络过拟合的问题。卷积神经网络利用输入是图片的特点，将神经元设计为宽度 (width)、高度 (height)、深度 (depth) 三个维度，使得前馈函数更加有效率，并减少了大量参数。图 2-4所示为卷积神经网络图片识别与分类实例示意图。

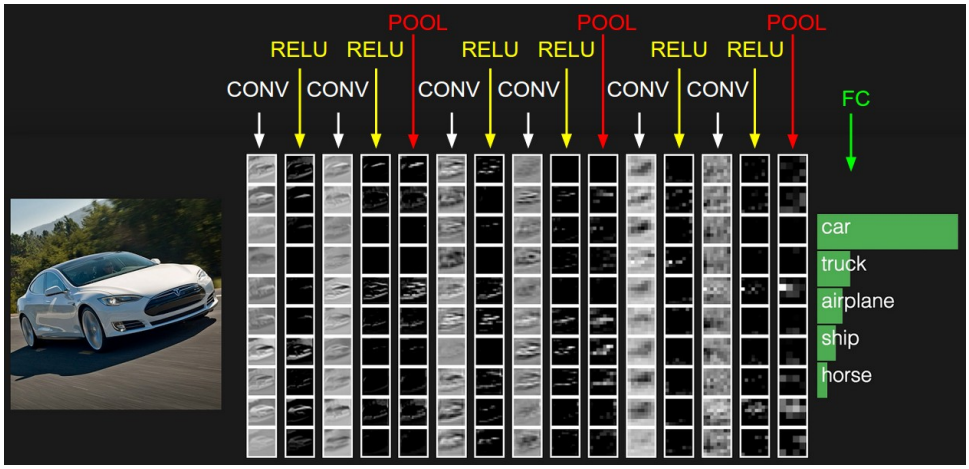

图 2-4: 卷积神经网络图片分类实例

卷积神经网络的结构主要由输入层、卷积层、池化层和全连接层构成。卷积层是其中的核心层，通过将原始图片与不同大小的卷积核之间的计算，学习到图片的某些视觉特征，并降低参数数量防止过拟合，不同卷积核代表每次卷积操作所接触到的神经元的感受野大小。通常在连续的卷积层之间会周期性地插入池化层，负责降低数据的空间尺寸减少参数数量以及计算量，能够有效地防止过拟合。池化层最常见的尺寸为 2*2 的滤波器，即每次从 4 个点中按照一定的规则生成一个输出作为特征值。池化策略一般包括最大池化、均值池化、高斯池化和可训练池化等，其中最大池化是最常见的池化策略。全连接层则可以与卷积层相互转换，将卷积层转化为全连接层只需要将权重变为一个巨大的

矩阵其中绝大多数位置都填充为 0，只有少量特定位置非零且权值相同。将全连接层转化为卷积层则只需要将 filter size 设置为整个输入层大小即可。

本文主要应用卷积神经网络训练控件图片分类器，以将从屏幕截图中裁剪得到的候选控件图片归类为其对应的控件类型。

2.4 本章小结

本章主要概述了项目所使用的相关技术和算法，并对其选用理由进行了阐述。首先介绍了本测试工具中测试执行部分所依赖的安卓自动化测试相关技术，即测试执行依赖的自动化测试框架 Appium、安卓设备交互监测工具 ADB、安卓设备页面结构分析工具 UIAutomator。其次介绍了本测试工具中测试引导部分所使用的事件知识图谱技术与其相应的存储工具图数据库 Neo4j 进行了介绍。最后介绍了本测试工具中测试执行部分图像分析阶段所使用的 Canny 边缘检测算法、SIFT 尺度不变特征点算法、OCR 文字识别服务以及控件图片分类模型卷积神经网络。

第三章　系统需求分析与概要设计

3.1　系统整体概述

随着移动应用软件的快速发展，为抢占大众市场、提前上线时间，移动应用的开发周期不断缩短，导致未经过充分测试的软件产品的功能性 Bug 频发。为了保障移动应用软件功能的高可用性，目前主流采用的自动化测试方法主要分为 GUI 自动化遍历测试以及自动化测试框架脚本测试两类。GUI 自动化遍历测试能够基于特定的遍历策略覆盖页面中的控件与页面间的跳转依赖关系，通过高控件与页面覆盖率验证其测试有效性，然而目前没有任何一款主流的自动化遍历测试工具的遍历策略中包含了功能逻辑性，其所覆盖的页面与控件仅是算法的中间产物，对测试人员验证功能有效性并没有帮助。自动化测试框架脚本测试是基于测试人员模拟人工操作过程编写的测试脚本完成的，其执行过程体现了特定功能的操作逻辑，然而由于测试脚本的编写过程中需要显示地指定待操作控件的 ID、XPath 等信息，随着软件产品的版本更迭等因素此类信息极易改变因此测试脚本的有效期也非常短暂，难以进行跨版本的复用。

本文针对以上两类测试方法的局限性，融合了其优势，创新性地引入了基于特定功能点操作逻辑构建知识图谱作为测试引导，学习到了功能操作逻辑性的图谱将基于目前已执行的测试上下文从下一步页面提取的布局与控件信息中筛选出待操作的控件集合，从而改进 GUI 自动化遍历测试中缺乏功能逻辑性的缺陷并很好地对遍历过程进行了剪枝大大减少了所需执行的控件和覆盖的页面。同时将基于图像理解技术形成控件识别和布局分析能力，基于图谱查询所得与控件、页面分析结果自动化生成执行语句通过自动化测试框架执行相应操作，改进了脚本测试需显式指定控件硬编码信息的不足，使得该测试过程能够在任意软件版本通用，大幅提高了复用性。

图 3-1所示为本系统的整体流程概述。首先测试人员将在系统上上传待测应用并选择待测功能点。测试过程开始后将于每一步测试操作执行前进行当前页面中的信息提取，分析页面布局结构以及控件识别。分析完成的信息将连同测试执行上下文一起经由已经构建完成的知识图谱进行匹配查询当前页面中

处于功能操作逻辑链中的控件元素，查询结果将以待操作控件位置信息、操作类型、待输入信息三元组集合的形式返回，同时也将图谱也将反馈该操作步骤是否为候选功能点终止步骤。返回的待操作控件信息将记录在测试执行上下文中，以树状结构的形式进行存储，所返回的每一待操作控件都将以一个独立分支进行操作序列的扩展。操作类型主要分为点击、输入、滑动三种，每一种最终都将通过 Appium 的相关接口执行，当前步骤操作执行完成后将继续迭代该页面信息提取与外部引导测试执行的过程。整个测试过程的终止将分为三类条件，分别为：图谱返回该步骤为候选功能点终止步骤，测试执行上下文中无未覆盖的分支，三元组序列无待操作控件返回，即应用功能点完备；图谱返回该步骤不为候选功能点终止步骤，三元组序列无待操作控件返回，即应用功能点不完善。功能点不完善还可以再区分为功能点逻辑错误与应用功能点缺失结果。功能点测试中间过程将实时展示在前端页面上，并且由于每一步需要经过页面图像分析与知识图谱查询匹配时间较长，中间过程中的分析阶段和中间产物也将实时反馈在前端页面上。最终执行成功的功能逻辑将以树状结构形式进行展示，帮助测试人员判断应用该功能是否存在逻辑问题。该技术通过基于不同功能点逻辑构建的知识库匹配测试过程解决了功能逻辑缺失的问题，通过图像理解能力自动化生成执行语句解决了测试脚本失效率高的问题，大大完善了功能点自动化测试过程，提高了测试能力的可复用性。

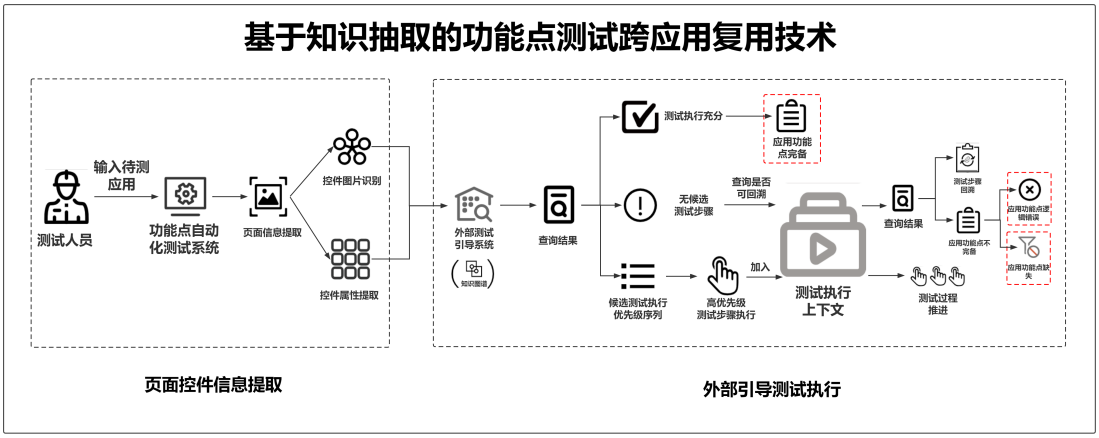

图 3-1: 基于知识抽取的跨应用复用功能点测试系统流程概述

3.2 系统需求分析

3.2.1 系统涉众分析

本系统作为基于知识抽取的跨应用复用功能点测试系统，所涉及的涉众如表 3-1所示，仅包含测试需求方这一单一涉众。对于测试需求方而言，他们需要通过该系统上传待测应用，选择待测试的功能点，查看该测试任务当前执行情况与所处阶段，查看功能点测试结果报告，对功能点测试结果进行分析等。此类测试需求方通常来说有可能是该应用的开发人员，需要对测试结果中的未完成的测试功能点进行 Bug 的定位和修复，也有可能是本应用的产品经理，他们需要对比该应用与其他应用在相同功能点上不同的操作逻辑，以分析和改进软件产品的用户体验和操作复杂性。他们所期待的是，基于知识抽取的跨应用复用功能点测试系统能够对给定的功能进行尽可能充分的探索，并对每个能够给完成本功能的路径分支，即功能场景，进行可达性检测，同时该测试过程能够在不同应用的相同功能点上进行复用，提高测试能力的通用性。

表 3-1: 系统涉众分析结果

涉众名称	涉众特征与期望
测试需求方	测试需求方希望在上传待测 APK 以及选定待测功能点后，本系统能够在上传的应用内对给定的功能进行尽可能充分的探索，并对每个能够给完成本功能的路径分支，即功能场景，进行可达性检测，同时该测试过程能够在不同应用的相同功能点上进行复用。测试需求方往往也是该应用的开发人员或产品经理，需根据功能点测试结果对功能逻辑缺陷进行 Bug 定位与修复；也可以根据本应用与其他同类应用的相同功能逻辑路径对比改进本产品的用户体验。

3.2.2 功能性需求分析

本节将在系统整体概述的基础上，基于系统所要实现的目标，对系统的功能性需求进行设计，并划分出相应的功能模块。图 3-2所示为系统模模块划分。基于知识抽取的跨应用复用功能点测试系统的功能性需求将主要由测试设

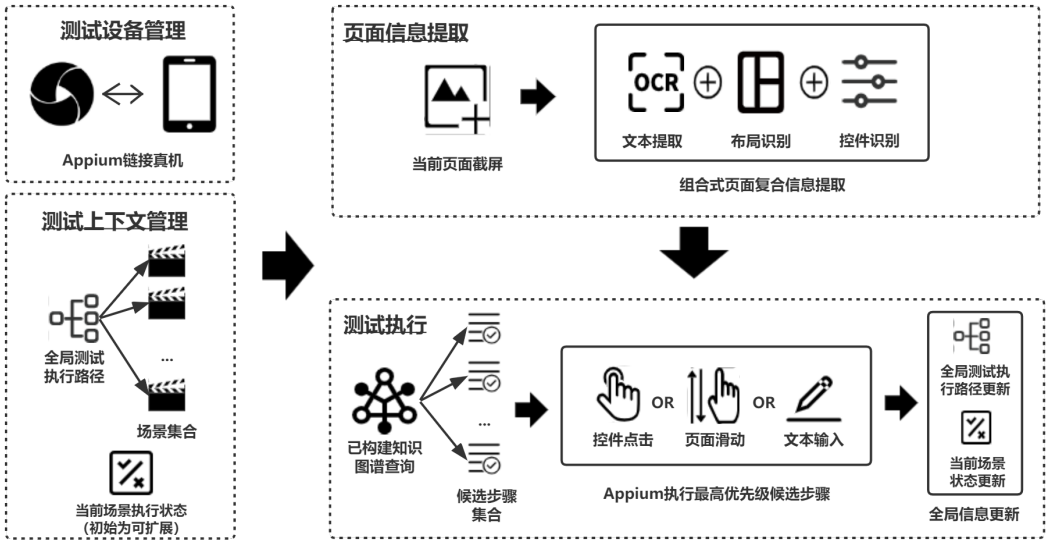

图 3-2: 系统模块划分

备管理模块、测试上下文管理模块、页面信息提取模块、测试执行模块以及测试结果可视化模块几部分组成。以下将分别阐述其具体的功能性需求。

表 3-2: 测试设备管理模块功能性需求表

需求编号	需求名称	需求内容	优先级
R1	查询已连接设备	获取当前已连接设备的详细信息以供用户选择测试执行设备。	高
R2	启动 Appium 服务	用户选定待测设备后系统根据相应设备信息启动 Appium 服务。	高
R3	获取待测 Apk 信息	系统对传入的 Apk 进行解析获取其包名与启动时页面等详细信息。	中
R4	获取当前设备页面截图	系统截取选定测试执行设备的当前页面截图并拉取到服务端。	高
R5	获取当前设备页面结构	系统获取选定测试执行设备的当前页面 xml 结构并拉取到服务端。	高

测试设备管理模块是系统用于管理连接自动化测试机柜的安卓设备并与之交互，获取设备当前状态信息的主要模块，表 3-2所示为测试设备管理模块的功能需求。用户在上传待测 Apk 已经选定了希望测试的功能点之后，系统将会

向用户展示目前已连接的可用设备信息，包括其品牌、型号、分辨率等，以供用户自行选择执行测试的设备。用户选定测试执行设备后，系统会将设备的相应信息传入 Appium 客户端中，并启动 Appium 服务链接安卓设备真机与服务端。用户上传待测 Apk 后，系统将会利用 aapt 工具对其进行解析，获取 Apk 包名和启动时页面信息，这两大信息也是 Appium 服务启动时必不可少的。在测试的执行过程中，由于测试执行语句的生成是基于图像理解技术的，因此需要与正在执行测试过程的安卓设备进行交互，包括截取当前页面截图以及分析当前页面的 XML 结构。获取完成的页面截图信息与页面结构信息将分别以 png 图片和 xml 文件的形式保存在安卓设备上，还需要将其拉取到系统的服务端。

测试上下文管理模块是系统用于管理测试过程的中间状态信息，并根据测试执行进度进行更新和切换的主要模块，表 3-3所示为测试上下文管理模块的功能需求。用户需要首先在系统上点击上传待测 Apk 按钮上传相应的待测应用，并新建相应的测试任务开始本次功能点测试。测试任务新建时还需要指定待测功能点，用户需要根据上传的 Apk 的具体情况，选择本 Apk 中包含的已存在的主要功能点进行测试。在测试过程中，经过页面信息提取与知识图谱引导的测试执行，在每一测试步骤时都有可能产生至少一个待操作的控件，从测试开始到测试结束将产生大量的可扩展的测试执行路径，系统将以树形结构记录测试执行的路径序列并进行扩展的更新操作，树形结构中所产生的分叉节点将会在后续进行场景回溯。图 3-3所示为以支付宝的登录功能点为例的全局测试执行路径树状图。全局测试执行上下文信息以测试执行路径树状图的形式进行存储，而局部测试状态即当前测试功能场景完备性也需要通过一个标志位表示，整个测试过程类似于深度优先遍历需要优先将当前执行的测试场景探索完成再回溯到上一分叉节点继续遍历，因此该测试功能场景完备与否的标志位也需要一并在测试过程中进行更新并随着场景的切换而切换。

页面信息提取模块是系统用于分析应用当前页面信息，解析页面结构并识别页面中的控件的主要模块，其生成的页面特征信息将作为自动化生成测试执行语句的基础。表 3-4所示为页面信息提取模块的功能性需求表。系统将在初始状态以及每一次测试操作执行完成后进行页面信息提取，提取的信息主要包括当前页面结构、当前页面中所包含的控件、控件中的文本信息三类。页面结构分析将主要利用将主要识别当前页面中的横向与纵向布局块。识别出的布局块中包含的内容即是控件元素，系统将进一步在布局块内部识别控件的轮廓信息，获取控件对应的 bounding box。得到其边缘后将对整张原始页面截图按照

表 3-3: 测试上下文管理模块功能性需求表

需求编号	需求名称	需求内容	优先级
R6	上传待测 Apk	用户在系统上上传待测 Apk 并新建一个测试任务开始功能点测试。	高
R7	选择待测功能点	用户在系统上根据所上传 Apk 的具体情况选择合适的功能点进行测试。	高
R8	全局测试执行路径管理	系统对测试过程中需要执行的测试路径和待操作的控件进行记录。	高
R9	测试场景状态管理	系统对测试过程中当前的功能场景的测试完备性状态进行更新和切换。	中

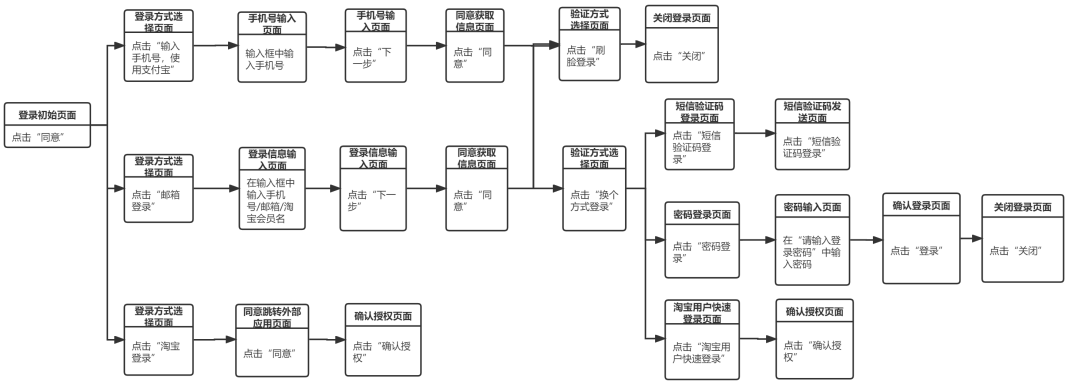

图 3-3: 支付宝登录功能测试路径树状图

不同控件轮廓进行裁剪，裁剪后的候选控件图片将通过训练好的模型识别其对应的类别。识别出的控件图片将进一步通过 OCR 识别其内部的文字内容，作为控件功能的描述信息。此时页面布局、页面中的控件元素以及控件内部文本都已经识别完成，但仍需要进行后续的信息匹配与整理，最终得到的形式化结构特征信息将作为页面信息提取结果以供知识图谱进行匹配和筛选。

测试执行模块是系统基于已提取的页面信息，结合知识图谱引导结果执行相应的控件操作推进测试逻辑过程的主要模块。表 3-5所示为测试执行模块的功能性需求表。系统将已提取的当前页面信息联合已执行的测试上下文在已构建完成的知识图谱中进行检索与匹配，知识图谱的匹配结果将返回待操作控件元素、操作类型、待输入信息三元组集合，从而筛选该页面中与本功能点相关且位于当前操作逻辑序列上的控件元素集合。系统对得到的筛选结果根据其不

表 3-4: 页面信息提取模块功能性需求表

需求编号	需求名称	需求内容	优先级
R10	页面布局结构分析	系统分析页面布局结构获取横向与纵向布局块内容。	高
R11	页面控件识别	系统识别页面中的控件元素的轮廓信息并对其类别进行分类。	高
R12	页面控件文本提取	系统对识别出的控件提取其内部的文本内容信息。	高
R13	页面特征信息融合	系统对识别得到的布局、控件、文本信息进行匹配与整理为形式化结果。	高

同的操作类型，基于特定操作类型利用底层 Appium 相关实现模拟用户执行相应控件操作，从而推进总体测试逻辑过程并继续下一测试过程迭代。

表 3-5: 测试执行模块功能性需求表

需求编号	需求名称	需求内容	优先级
R14	知识图谱查询交互	系统基于已提取页面信息利用知识图谱对当前页面待操作控件进行筛选。	高
R15	控件元素操作	系统根据不同的操作类型对具体控件进行相应操作。	高

测试结果可视化模块是系统创建与管理测试任务以及向用户反馈测试结果的主要模块。表 3-6所示为测试结果可视化模块的功能性需求表。用户需要在前端页面新建相应测试任务，并能选择任务相关设置例如测试设备信息、需要测试的功能点等。用户需要能在任务展示页面选择当前任务或历史任务进行查看，可查看任务详细详细包括当前执行状态、历史任务列表以及已经完成的任务的结果报告。当次测试任务完成后，系统需要向用户反馈测试结果可视化结果报告，结果报告内容包括本次测试执行路径图，测试执行结果和本功能点对应知识图谱以帮助用户理解测试结果并进行后续的相关 Bug 定位与修复。

表 3-6: 测试结果可视化模块功能性需求表

需求编号	需求名称	需求内容	优先级
R16	创建测试任务	用户在系统前端页面创建测试任务并提供测试设备、待测功能等选项。	高
R17	查看任务详情	系统展示用户任务的当前执行状态、历史任务和结果报告等信息。	中
R18	反馈测试任务结果报告	测试任务执行结束后系统向用户反馈测试结果可视化报告。	高

表 3-7: 系统非功能性需求列表

需求编号	需求名称	需求内容	优先级
R19	易用性	系统应该简洁易用，用户能在没有产品说明明书的情况下短时间内掌握使用方法。	中
R20	可扩展性	系统各项功能应该易于扩展，在发生需求变更时能够以较少工作量修改原有功能。	中
R21	可靠性	系统不会因错误的输入数据而产生异常，故障发生后能迅速恢复并无数据丢失。	高
R22	可用性	系统能够保证自身可以在较长的一段时间内稳定运行而不发生任何故障。	高
R22	可维护性	系统发生故障后可以通过脚本或重启等方式快速重新恢复到正常状态。	高

3.2.3 非功能性需求分析

为使系统能够正常运行、易于维护、使用便捷，对基于知识抽取的跨应用复用功能点测试系统的非功能性需求进行了设计，表 3-7所示为本系统的非功能性需求。本系统应该具备较高的易用性，界面简洁易懂，能够在不向用户提供产品使用说明书的情况下令用户快速掌握本系统基本功能的使用方法。本系统应该具备较高的可扩展性，系统实现对变更开放，在未来发生需求变更时能够以较小的代价在原有功能上进行修改或在不影响原有功能的基础上快速实现新功能。本系统应该具备较高的可靠性，系统不会因为错误输入数据或恶意攻击而产生异常甚至崩溃，万一发生崩溃时也能迅速恢复系统并做到历史数据不

丢失，不影响用户正常使用本系统。本系统应该具备较高的可用性，系统能够保证自身在较长的一段时间内稳定运行并且不发生任何技术性故障。本系统应该具备较高的可维护性，系统发生故障后能够通过预编写好的脚本或者重启服务器等方式快速，例如 5 分钟内，恢复系统正常运行状态。

3.3 系统用例分析

3.3.1 系统用例图

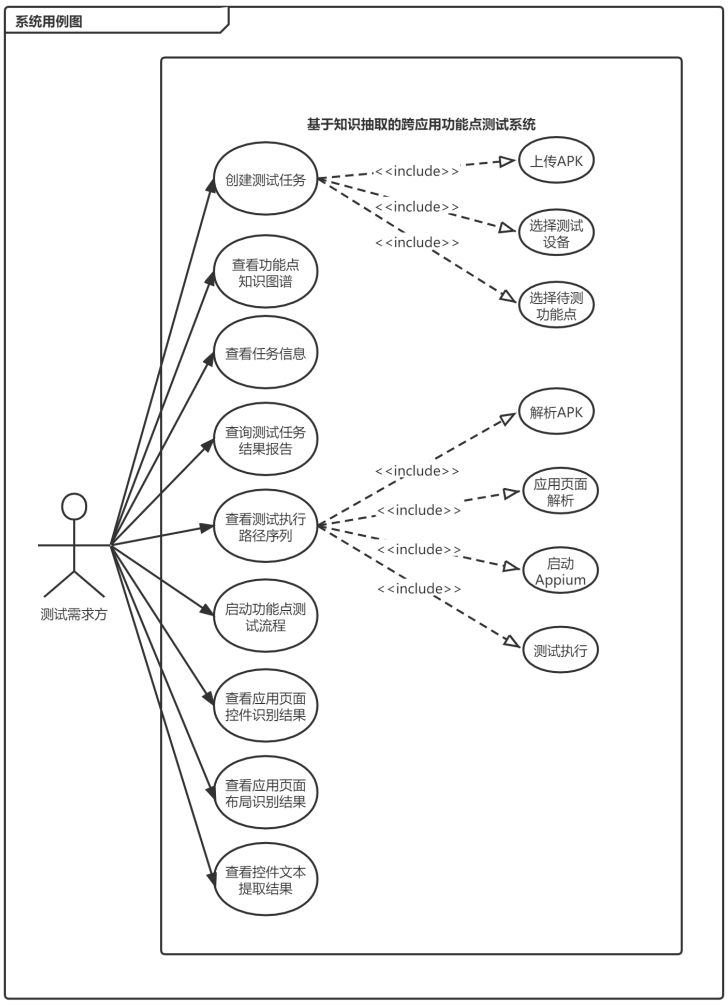

图 3-4: 系统用例图

基于知识抽取的跨应用功能点测试系统的用例图如图 3-4所示。本系统的服务人员为测试需求方，他们希望能够在系统中上传待测试的 APK 并选择需要

测试的功能点类型以此创建测试任务，在平台上能够查看当前任务进度、查询历史测试任务数据及其相应结果，测试任务完成之后能向其展示多维详细的测试结果报告。因此，本系统的业务目标即为提供安卓 APK 功能点测试平台，通过知识图谱引导以及图像分析方法自动化进行逻辑性功能场景覆盖测试，并向用户展示测试过程中分析信息支撑的测试结果。

3.3.2 系统用例总表

如表 3-8所示，基于知识抽取的跨应用复用功能点测试系统拥有 9 个主要的系统用例，分别为：创建测试任务、查看功能点知识图谱、查看任务信息、查询测试任务结果报告、查看测试执行路径、启动功能点测试流程、查看应用页面控件识别结果、查看应用页面布局识别结果以及查看控件文本提取结果。该表中阐述了每个用例和功能性需求之间的联系，并为每个用例进行了编号，在下一节中将以该编号对各个用例的具体描述进行解释。

表 3-8: 基于知识抽取的跨应用复用功能点测试系统用例总表

用例编号	用例名称	需求编号
UC1	创建测试任务	R6、R7、R16
UC2	查看功能点知识图谱	R14、R15
UC3	查看任务信息	R17
UC4	查询测试任务结果报告	R13、R18
UC5	查看测试执行路径序列	R8、R9
UC6	启动功能点测试流程	R1、R2、R3、R4、R5
UC7	查看应用页面控件识别结果	R11
UC8	查看应用页面布局识别结果	R10
UC9	查看控件文本提取结果	R12

3.3.3 系统用例描述

在本节中将会对上一节中所描述的 9 个用例进行详细的阐述：

创建测试任务是测试需求方使用本系统的第一步。用户需要在系统前端页面首先点击创建测试任务按钮，接着系统将会要求用户设定测试配置项，即上传待测应用与设定待测功能点。系统将会检测上传的待测应用文件格式，如

果不为.apk 则上传失败。系统同时会检测相同版本的相同应用文件是否已经存在，若已存在则无需重复上传。待测功能点将在系统提供的下拉框内进行选择。配置项设定完成后，系统将提示测试任务创建成功，同时系统中将能看到已创建的测试任务信息。该用例的用例描述如表 3-9所示。

表 3-9: 创建测试任务用例描述

描述项	说明
用例编号	UC1
用例名称	创建测试任务
参与者	测试需求方
优先级	高
用例描述	测试需求方在系统中设定各项配置项，创建一个测试任务。
前置条件	测试需求方已创建平台账户，拥有使用本平台的权限。
主事件流	1. 用户点击创建测试任务按钮。 2. 用户点击上传待测应用按钮，上传待测应用 APK。 3. 用户在下拉框中选择待测功能点。 4. 系统提示测试任务创建成功。
后置条件	系统提示测试任务创建成功，前端页面中能够看到已经创建的任务信息，系统后台 OSS 存储中能够找到已上传的 APK。
扩展事件流	3.a 系统检测所上传的应用文件不为.apk 格式，则上传失败 3.b 系统检测所上传的该版本的应用 apk 已存在，则上传失败。
特殊需求	测试需求方上传的待测应用必须为.apk 格式文件

查看功能点知识图谱是在测试任务执行完成之后用户可以查看的测试结果产物之一。用户选择已执行完成的测试任务查看其详细信息，接着点击查看功能点知识图谱按钮，系统查询用于存储该已构建完成的知识图谱的图数据库，并将知识图谱情况以可视化形式在前端反馈给用户。用户在系统中可查看的功能点列表包含了系统基于相应功能逻辑场景与构建完成的所有功能点名称。该功能点的该用例的用例描述如表 3-10所示。

查看任务信息主要用于测试需求方查看自己在系统中创建的所有历史测试任务的详细信息。用户在系统选择特定任务后可以查看其测试配置项，即待测应用 APK 及版本、选定测试功能点，当前测试执行状态以及若该任务已执行完

表 3-10: 查看功能点知识图谱用例描述

描述项	说明
用例编号	UC2
用例名称	查看功能点知识图谱
参与者	测试需求方
优先级	中
用例描述	测试需求方查看待测功能点的可视化知识图谱结果。
前置条件	测试需求方选定的该测试任务已经完成，得到测试结果。
主事件流	1. 用户选择已执行完成的测试任务。 2. 用户点击查看对应功能点知识图谱按钮。 3. 系统展示该功能点可视化已构建的知识图谱结果。
后置条件	系统成功获取该已构建知识图谱信息。
扩展事件流	无
特殊需求	测试需求方需要查看的功能点必须是系统已包含的。

成测试过程后的测试结果报告。系统通过查询数据库以及与当前测试执行进程通信获取相应信息并进行反馈。该功能点的用例描述如表 3-11所示。

查询测试任务结果报告主要用于测试需求方在平台上获得已创建测试任务的测试结果反馈。用户在系统中首先获取所有已创建测试任务，测试任务的当前状态将分为未开始、执行中以及已完成。用户可选择当前状态为已完成的测试任务点击其查看测试结果报告按钮获取系统可视化测试结果报告。系统将会获取该任务测试过程中产生的所有中间结果数据进行汇总与融合。系统分析完成后将形成可视化报告。该功能点的用例描述如表 3-12所示。

查看测试执行路径序列是在测试任务执行完成之后用户可以查看的测试结果产物之一。用户在系统前端页面获取已执行完成的测试任务，选择特定任务查看详情，点击查看测试执行路径序列按钮。系统获取其测试过程中执行情况，根据操作控件以及页面跳转关系进行链接，生成树状图可视化执行序列反馈给用户。该功能点的用例描述如表 3-13所示。

启动功能点测试流程主要用于测试需求方手动启动已创建任务的测试流程。已创建测试任务状态初始为未开始，用户启动测试流程需要先选择已与系统成功连接的可用测试设备。若当前没有设备已经成功连接或所有已连接设备都被其他任务占用将无法开始当前测试流程。用户选定对应设备后通过点击启

表 3-11: 查看任务信息用例描述

描述项	说明
用例编号	UC3
用例名称	查看任务信息
参与者	测试需求方
优先级	高
用例描述	测试需求方查看所有历史任务的详细信息。
前置条件	测试需求方已经在系统中有创建测试任务的记录。
主事件流	1. 用户查看历史测试任务记录。 2. 用户点击查看该任务测试应用、功能点、测试执行状态、测试结果报告等信息。 3. 系统显示该任务的相关信息。
后置条件	系统查询数据库与执行任务进程获取相关信息。
扩展事件流	无
特殊需求	测试需求方只能查看自己创建的有权限的任务详情。

表 3-12: 查询测试任务结果报告用例描述

描述项	说明
用例编号	UC4
用例名称	查询测试任务结果报告
参与者	测试需求方
优先级	高
用例描述	测试需求方查看已完成任务的结果报告。
前置条件	测试需求方选定的该测试任务已经完成。
主事件流	1. 用户选择已执行完成的测试任务。 2. 用户点击查看该任务的结果报告。 3. 系统获取所有测试中间结果数据进行汇总与融合。 4. 系统展示已执行完成的该任务的可视化结果报告。
后置条件	系统融合汇总所有测试过程中间结果形成可视化结果报告。
扩展事件流	3.a 若当前任务未执行完成，则无法查看其结果报告。
特殊需求	结果报告需要将测试中间数据结果进行可视化地展现。

表 3-13: 查看测试执行路径序列用例描述

描述项	说明
用例编号	UC5
用例名称	查看测试执行路径序列
参与者	测试需求方
优先级	高
用例描述	测试需求方查看已完成测试任务的测试执行路径序列。
前置条件	测试需求方选定的该测试任务已经完成。
主事件流	1. 用户选择已执行完成的测试任务。 2. 用户点击查看测试执行路径序列按钮。 3. 系统展示可视化测试执行路径序列树状图结果。
后置条件	系统获取测试执行情况生成路径序列树状图。
扩展事件流	无
特殊需求	系统需要以树状图的方式可视化展示测试执行路径序列。

动测试过程按钮即可开始该任务的测试执行。系统将会首先链接 Appium 服务与测试设备，系统后端开始执行信息提取与测试执行迭代，系统前端可见该任务进入执行状态。该功能点的用例描述如表 3-14所示。

查看应用页面控件识别结果是在测试任务执行完成之后用户可以查看的测试结果产物之一。用户选择已执行完成的测试任务，点击查看应用页面控件识别结果按钮。系统将会查询测试过程中执行涉及到的应用页面截图，根据控件识别结果将在截图中对所识别出的控件进行标识，标识后的结果图片将在系统前端向用户进行可视化反馈。该功能点的用例描述如表 3-15所示。

查看应用页面布局识别结果是在测试任务执行完成之后用户可以查看的测试结果产物之一。用户点击查看应用页面布局识别结果按钮，系统获取测试过程中的应用中间页面截图，根据页面对应布局识别结果在截图中标识所识别出的布局块，标识后的页面将进行可视化。该功能点的用例描述如表 3-16所示。

查看控件文本提取结果是在测试任务执行完成之后用户可以查看的测试结果产物之一。用户选择已执行完成的测试任务，点击查看控件文本提取结果按钮。系统获取测试过程中的应用中间页面截图及控件识别结果，根据控件元素位置信息将控件文本提取结果与对应控件进行匹配。控件文本提取结果将与控件识别结果融合后进行可视化。该功能点的用例描述如表 3-17所示。

表 3-14: 启动功能点测试流程用例描述

描述项	说明
用例编号	UC6
用例名称	启动功能点测试流程
参与者	测试需求方
优先级	高
用例描述	测试需求方使已创建的测试任务进入执行状态。
前置条件	测试需求方已经成功创建相应测试任务。
主事件流	1. 用户选择已创建的测试任务。 2. 用户选择已连接测试设备点击启动测试过程按钮。 3. 系统链接 Appium 服务与真机，开始信息提取与测试执行。 4. 系统反馈测试任务已成功启动，目前正在执行中。
后置条件	选定测试设备与 Appium 服务端成功链接，系统后端开始执行信息提取与测试执行迭代，系统前端可见该任务进入执行状态。
扩展事件流	2.a 系统目前无已连接设备，测试无法开始执行。
特殊需求	无。

3.4 系统概要设计

3.4.1 系统架构设计

基于知识抽取的跨应用复用功能点测试系统服务端基于 Django 框架实现，主要划分为页面信息提取模块、测试上下文维护模块、测试资源管理模块、测试执行模块以及服务于前端结果展示的测试结果可视化模块共 4 个模块。构建完成的知识图谱采用 Neo4j 存储。项目前后端分离，前端基于 Vue 框架开发。系统架构图如图 3-5所示。下面将对系统的主要模块进行详细描述。

前端方面，本系统采用 Vue 框架实现。Vue 是一个轻量级的数据驱动框架，通过将页面组件化，对各个组件以及其所需利用到的数据进了响应式的绑定，同时它易于上手能够高效地完成开发。前端 UI 组件采用了开箱即用的 iView 组件库，其提供的示例能够很好地帮助开发人员快速构建所需界面。前后端通信方面主要令服务端通过 Restful API 将已实现的服务暴露相应接口，前

表 3-15: 查看应用页面控件识别结果用例描述

描述项	说明
用例编号	UC7
用例名称	查看应用页面控件识别结果
参与者	测试需求方
优先级	高
用例描述	测试需求方查看已完成的测试任务的应用页面控件识别结果。
前置条件	测试需求方选定的该测试任务已经完成，得到测试结果。
主事件流	1. 用户选择已执行完成的测试任务。 2. 用户点击查看应用页面控件识别结果按钮。 3. 系统获取测试过程中间涉及的页面及其控件识别结果。 4. 系统在中间页面截图进行控件标识可视化反馈给用户。
后置条件	系统获取测试过程页面控件识别中间结果并进行可视化展示。
扩展事件流	无
特殊需求	应用页面控件识别结果需要在截图上标识以展示可视化结果。

表 3-16: 查看应用页面布局识别结果用例描述

描述项	说明
用例编号	UC8
用例名称	查看应用页面布局识别结果
参与者	测试需求方
优先级	中
用例描述	测试需求方查看应用页面布局识别结果。
前置条件	测试需求方选定的该测试任务已经完成，得到测试结果。
主事件流	1. 用户选择已测试完成的测试任务。 2. 用户点击查看应用页面布局识别结果按钮。 3. 系统获取测试过程中间涉及的页面及其布局识别结果。 4. 系统在截图中标识布局块进行可视化展示。
后置条件	系统获取应用页面布局识别结果后进行可视化展示。
扩展事件流	无
特殊需求	应用页面布局识别结果需要在截图上标识布局块以可视化。

表 3-17: 查看控件文本提取结果用例描述

描述项	说明
用例编号	UC9
用例名称	查看控件文本提取结果
参与者	测试需求方
优先级	中
用例描述	测试需求方查看控件文本提取结果。
前置条件	测试需求方选定的该测试任务已经完成，得到测试结果。
主事件流	1. 用户选择已测试完成的测试任务。 2. 用户点击查看控件文本提取结果按钮。 3. 系统获取测试过程中间页面及其控件识别结果。 4. 系统将控件文本提取结果与对应控件进行匹配后展示。
后置条件	系统获取控件文本提取结果并与对应控件完成匹配。
扩展事件流	无
特殊需求	无

端通过 Http 通信调用相应服务获取数据。测试任务执行结果可视化方面，主要利用了 Echarts 绘制相关图表，其提供了多种图表组合的组件式开箱即用的能力，支持 Canvas、SVG 双引擎一键切换，具有增强渲染和数据交互能力，帮助测试需求方获取对测试结果的多维度和专业化的数据分析。

本系统的后端主要基于 Django 框架开发。Django 是一个 python 编写的轻量级 web 框架，其基于 MVC 的设计模式实现，开发简单、组件划分明确同时具备很好的扩展能力。本系统服务端主要划分为五个模块。测试资源管理模块主要负责对与系统连接的测试设备以及上传到系统中的应用 APK 进行管理，提供 APK 解析、测试设备信息交互及状态获取、Appium 服务启动等能力，是服务端与物理设备端和存储端交互的核心模块。测试上下文维护模块主要负责记录与更新测试过程中产生的测试执行路径序列以及当前测试场景执行完成性，提供测试执行路径序列继承关系扩展、测试场景切换以及场景执行状态设置等能力，是服务测试过程推进、终止测试流程的核心控制模块。页面信息提取模块主要负责分析当前应用执行到的页面中的特征信息，提供页面布局识别、页面控件识别以及控件文字提取等能力，是为测试执行提供信息来源以及为测试结果可视化模块提供中间过程结果的数据生产者模块。测试执行模块主

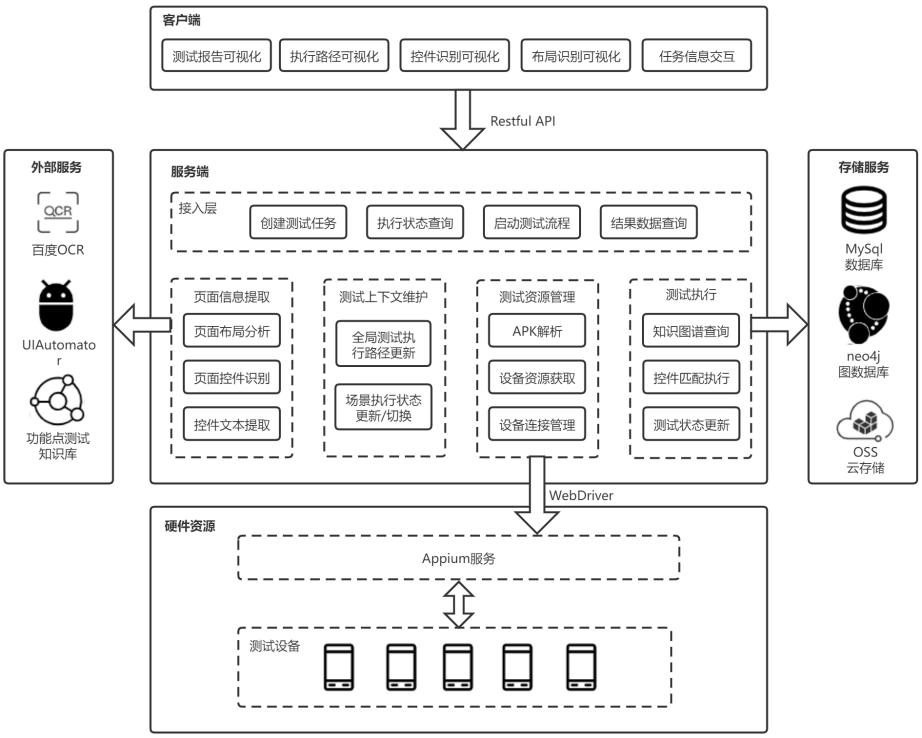

图 3-5: 系统架构图

要负责对获取的候选元素进行筛选，识别下一步待操作控件对象，提供知识图谱查询、控件操作模拟、测试状态更新等能力，是负责对接知识图谱完成功能逻辑匹配与检索的交互模块。测试结果可视化模块主要负责向前端提供可视化数据，提供融合测试过程中间结果以及生成可视化结果报告的能力。

本系统的存储端主要应用了 Mysql 数据库、Neo4j 图数据库以及云端 OSS 存储。Mysql 数据库主要用于存储系统中的测试任务相关信息，Neo4j 图数据库主要负责存储各个功能点已构建好的记录操作逻辑的功能点测试知识图谱，云端 OSS 存储主要负责管理测试过程中生成的应用页面截图，包括原始截图以及标识了页面中控件或布局结果的标识结果截图，和页面布局结构文件。

3.4.2 系统视图模型

本文将参考软件工程领域中使用最为广泛的“4+1”视图模型分别从场景视图、逻辑视图、开发视图、进程视图和物理视图五个方面对系统整体架构进行描述。其中场景视图在 3.3 节的系统用例分析中已经做了非常详细的解释，在此不再赘述，接下来将从其他 4 个视图对程序总体设计进行阐述。

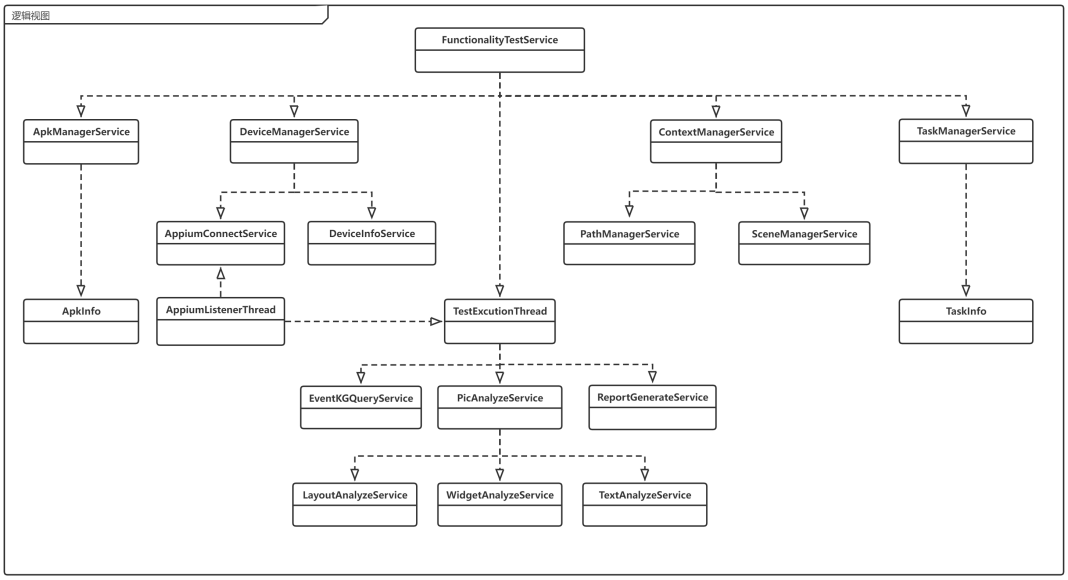

图 3-6: 系统逻辑视图

逻辑视图负责对系统职责进行划分并阐述各逻辑元素间的关系。如图 3-6所示为系统的逻辑视图。FunctionalityTestService 为本系统的核心服务，提供了 Apk 解析、设备管理、任务信息管理、测试上下文管理能力，并在系统启动时创建了两个守护进程，分别为 Appium 监听进程和测试执行进程。ApkManagerService 为 Apk 解析服务，主要负责解析上传到系统中的 Apk 详细信息，并生成 ApkInfo 类负责记录 Apk 包名、启动时页面、版本等信息。TaskManagerService 为测试任务管理服务，主要负责管理系统中创建的测试任务信息，并生成 TaskInfo 类负责记录任务创建时间、选定 APK、待测功能点、选定测试设备等信息。DeviceManagerService 服务负责管理测试设备，并分为 AppiumConnectService 和 DeviceInfoService 两个细分服务。AppiumConnectService 服务负责寻找可用测试设备并与 Appium 客户端进行链接，将由 AppiumListener 进程接收到系统启动测试流程请求后进行调用。DeviceInfoService 服务负责获取设备截图、状态信息等。ContextManagerService 服务负责维护测试上下文信息并分为 PathManagerService 和 SceneManagerService 两个细分服务。PathManagerService 服务更新测试执行路径，SceneManagerService 负责维护功能场景测试状态。测试执行开始后，TestExcution 进程将分别调用 EventKGQueryService，PicAnalyzeService 和 ReportGenerateService 服务。PicAnalyzeService 负责提取下一步待操作页面信息，将细分为 WidgetAnalyzeService，LayoutAnalyzeService，TextAnalyzeService 分别用于处理控件识别、布

局识别以及控件文字提取。EventKGQuery 负责查询已构建知识库进行功能逻辑检索与匹配，筛选出下一步待操作的控件集合。ReportGenerateService 负责在测试完成后融合测试产物生成测试结果报告。

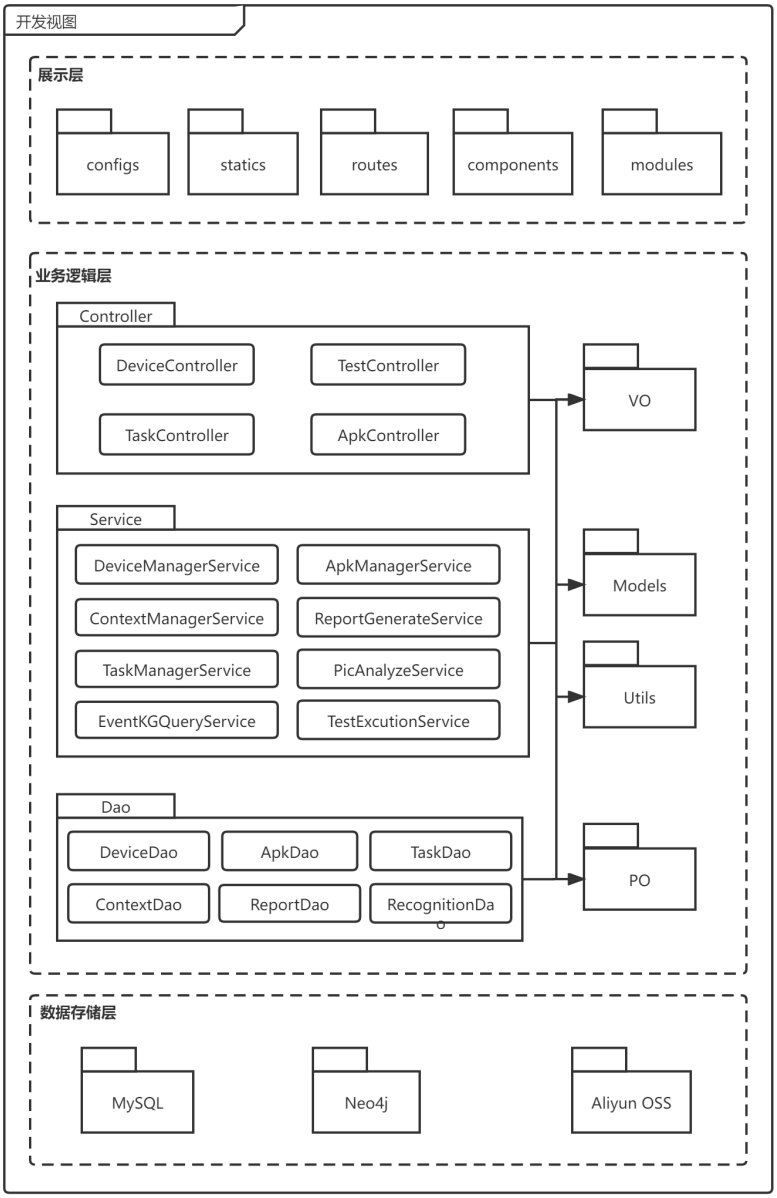

图 3-7: 系统开发视图

开发视图负责对系统中的的各个元素描述其在项目中的代码位置。如图 3-7所示为系统的开发视图。本系统采用分层结构。展示层中，configs 包存放的是系统前端的各配置项、statics 包存放的是前端的静态资源文件、routes 包存放的是前端页面的路由文件、components 包存放的是封装好的可复用组件、

modules 包存放的是前端的各个页面布局文件。业务逻辑层中，Controller 层负责将前端发送的请求分发到各个服务，Service 层负责实现项目所需的业务逻辑，Dao 层负责抽象数据库持久化存储的对象。VO 包中存放用于与前端页面展示的对象，PO 则为逻辑处理和数据存储相关对象。Models 包中存放了实现业务逻辑所需的自定义对象，而 Utils 包中存放了多个模块需要共用的工具类，包括日志打印、数据处理、文件解析等。数据存储层中主要包含 MySQL 负责存储系统业务逻辑持久化对象、Neo4j 存储构建完成的各功能点知识图谱、云端 OSS 存储测试过程中产生的如截图、文件等中间产物。

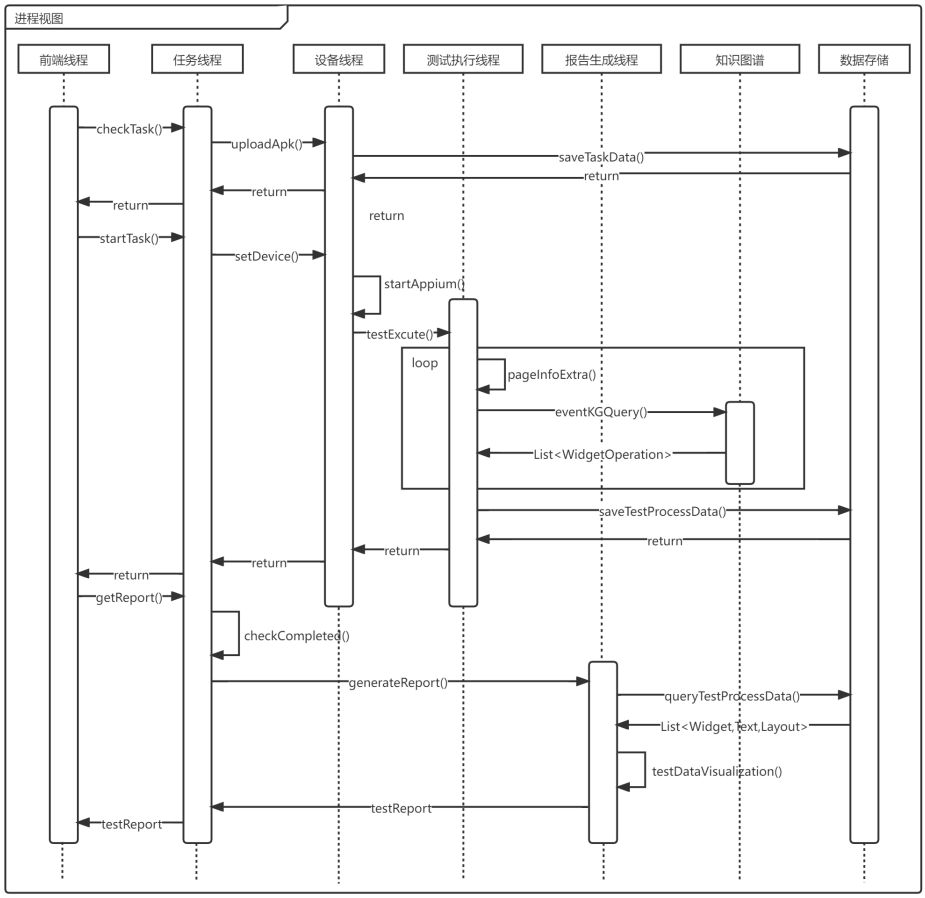

图 3-8: 系统进程视图

进程视图负责描述逻辑架构元素之间的交互关系。如图 3-8所示为系统的进程视图。用户在前端发起创建任务请求，任务线程收到请求后向设备线程发起上传 APK 调用，随后将创建完成的任务信息存储到数据库中。用户在前端发起启动测试任务请求，任务线程收到请求后向设备线程发起测试执行调用，设备线程启动 Appium 后向测试执行线程发起启动测试请求，经过其余知识图谱

的查询返回迭代后，在测试任务执行完成后将测试中间数据存储到数据库中。用户在前端发起获取结果报告请求，任务线程收到请求后进行任务完成与否自检，随后向报告生成线程发起生成报告请求，报告生成线程向数据库请求测试任务中间结果，经过融合汇总后形成可视化报告反馈给前端。

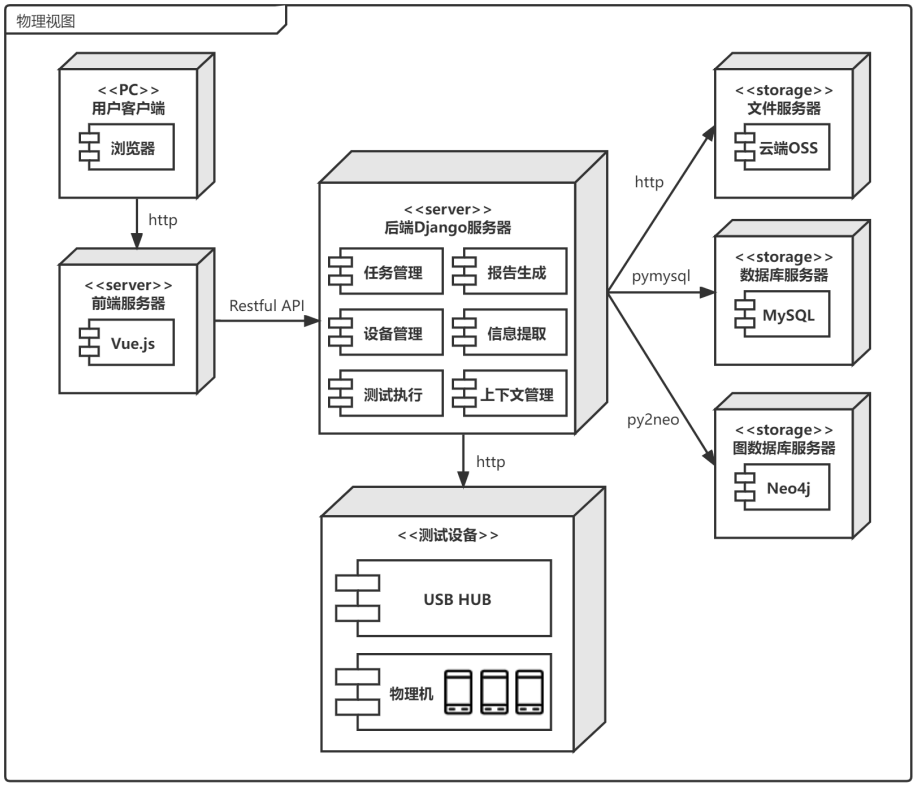

图 3-9: 系统物理视图

物理视图负责描述系统运行在物理或软件环境上的部署情况。如图 3-9所示为系统的物理视图。用户通过个人浏览器向前端服务器发送 http 请求，本系统的前端服务器采用 Vue 框架，前端接收到请求后通过 Restful API 将请求发送到后端服务器。本系统的后端服务器采用 Django 框架，接收到的请求将通过路由地址由后端的 controller 分配到具体的业务实现层处理相关业务逻辑。后端将包括如任务管理、设备管理、测试执行、报告生成、信息提取、上下文管理等一系列服务。具体的测试将利用物理机测试设备执行。物理机统一存放在测试机柜中，并通过 USB HUB 中间层实现多测试机的连接。后端服务器与测试机柜的交互通过 http 连接。后端处理业务逻辑需要利用知识图谱引导，构建完成的知识图谱存放在 Neo4j 图数据库服务器中，与后端 Django 服务器通过 py2neo

相关 API 进行调用。后端生成的截图、文件等数据将通过 http 连接存放在云端 OSS 存储中。其余业务数据将通过 pymysql 连接 MySQL 服务器进行存储。

3.4.3 系统实体类设计

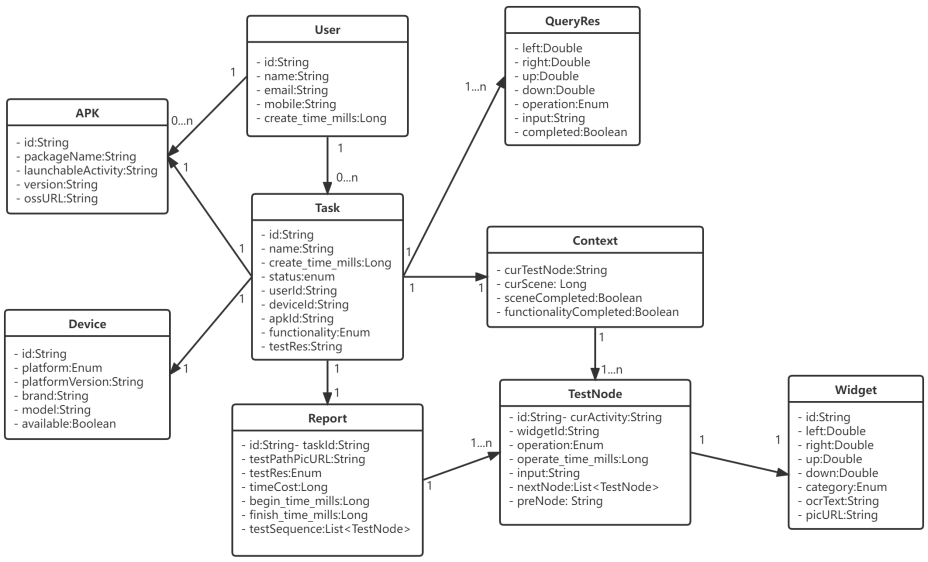

图 3-10: 系统实体类图

基于知识抽取的跨应用功能点测试系统中主要包含的实体类如图 3-10所示。其中重要的实体类主要包括测试需求方类 User，测试任务类 Task，待测应用类 APK，测试设备类 Device，测试结果报告类 Report，知识图谱查询结果类 QueryRes，测试上下文类 Context，测试执行节点类 TestNode，控件类 Widget。测试需求方能够创建 0-n 个测试任务。每个测试任务需要选定待测应用并指定测试设备，测试完成后将生成可视化结果报告。每个测试任务执行过程中需要维护全局测试执行上下文，并迭代地查询 1-n 次知识图谱获取测试引导。测试执行上下文会由多个执行节点构成执行序列，每个执行节点操作一个识别出的控件。可视化结果报告中会展示本次测试过程中执行的 1-n 个测试节点。

User 类：User 类为系统使用者测试需求方的抽象。测试需求方在系统上创建测试任务，上传待测应用，选择测试执行的设备并设定测试功能点，在测试任务完成后查看测试结果可视化报告。User 类具体内容如表 3-18所示。

APK 类：APK 类为测试需求方上传系统中的待测应用的抽象。上传的待测应用会优先存储在 OSS 云端，在测试任务启动时，根据 OSS 下载 URL 将远端 APK 下载并安装到测试机上进行测试。APK 类具体内容如表 3-19所示。

表 3-18: User 类设计描述

属性	数据类型	说明
id	String	测试需求方 ID
name	String	测试需求方名称
email	String	测试需求方邮箱
mobile	String	测试需求方手机号
create_time_mills	Long	测试需求方创建账户时间。

表 3-19: APK 类设计描述

属性	数据类型	说明
id	String	待测应用 ID
packageName	String	待测应用包名
launchableActivity	String	待测应用启动时页面名称
version	String	待测应用版本号
ossURL	String	待测应用在 OSS 云端存储的下载 URL

Device 类：Device 类为测试设备物理机的抽象。测试设备将统一连接到测试机柜，测试需求方在创建启动测试执行流程时需要指定本次测试任务待执行的具体设备。Device 类具体内容如表 3-20所示。

表 3-20: Device 类设计描述

属性	数据类型	说明
id	String	测试设备 ID
platform	Enum	测试设备操作系统（安卓/苹果）
platformVersion	String	测试设备操作系统版本
brand	String	测试设备品牌
model	String	测试设备型号
available	Boolean	测试设备当前是否可用

Task 类：Task 类为测试需求方创建的测试任务对象的抽象。测试需求方在创建测试任务时需要上传待测应用、选择测试设备、选择测试功能点。测试需

求方可以手动启动测试流程。Task 类具体内容如表 3-21所示。

表 3-21: Task 类设计描述

属性	数据类型	说明
id	String	测试任务 ID
name	String	测试任务名称
create_time_mills	Long	测试任务创建时间
status	Enum	测试任务当前状态（未开始/执行中/已完成）
userId	String	测试任务创建者
deviceId	String	测试任务执行设备
apkId	String	测试任务指定待测应用
functionality	Enum	测试任务待测功能点
testRes	String	测试任务结果报告

Report 类：Report 类为测试任务执行完成后的可视化结果报告的抽象。测试需求方在可视化报告中可以查看测试执行路径序列，获取布局以及控件等测试过程中的识别信息。Report 类具体内容如表 3-22所示。

表 3-22: Report 类设计描述

属性	数据类型	说明
id	String	结果报告 ID
taskId	String	结果报告对应测试任务
testPathPicURL	String	结果报告测试执行路径图片 OSS 下载链接
testRes	Enum	测试结果（功能点完备/应用功能点逻辑错误/应用功能点缺失）
timeCost	Long	测试任务执行共计花费时间
begin_time_mills	Long	测试任务执行开始时间
finish_time_mills	Long	测试任务执行结束时间
testSequence	List<TestNode>	测试任务执行序列集合

QueryRes 类：QueryRes 类为测试任务执行过程中查询知识图谱所得测试引导结果的抽象。系统将当前页面中提取所得到的的信息以及测试执行上下文与

知识图谱中记录的功能点操作逻辑知识进行匹配，从页面信息中筛选得到候选控件及操作集合。QueryRes 类具体内容如表 3-23所示。

表 3-23: QueryRes 类设计描述

属性	数据类型	说明
left	Double	待执行控件左侧坐标
right	Double	待执行控件右侧坐标
up	Double	待执行控件上侧坐标
down	Double	待执行控件下侧坐标
operation	Enum	待执行控件操作（点击/输入/滑动）
input	String	待输入信息
completed	Boolean	当前测试执行是否为候选结束操作

Context 类：Context 类为测试任务执行过程中维护的测试执行上下文信息的抽象。系统在测试执行的过程中为保证测试操作可回溯以及测试场景确认需要维护测试执行路径序列，在深度遍历完当前场景切换至下一场景时需要更新当前执行场景。Context 类具体内容如表 3-24所示。

表 3-24: Context 类设计描述

属性	数据类型	说明
curTestNode	String	当前执行到的测试执行节点
curScene	Long	当前正在执行测试的功能场景编号
sceneCompleted	Boolean	当前功能场景测试执行完成与否
functionalityCompleted	Boolean	当前功能点测试充分与否

TestNode 类：TestNode 类为测试任务执行过程中的测试执行节点的抽象。系统在生成测试执行路径序列时需要记录测试过程中执行的每一步所操作的控件、执行的操作类型、输入的信息，并记录其前继与后继节点，为此将这类信息结合封装为测试执行节点。TestNode 类具体内容如表 3-25所示。

Widget 类：Widget 类为测试任务执行过程中识别到的页面中的控件元素的抽象。系统在测试执行任务过程中需要对所覆盖的每一页面进行控件识别，对识别到的控件元素将进一步分析其类别以及提取其中的文字信息。从原截图中

表 3-25: TestNode 类设计描述

属性	数据类型	说明
id	String	测试执行节点 ID
curActivity	String	当前应用页面名称
widgetId	String	与当前测试执行节点关联的控件
operation	Enum	待执行控件操作（点击/输入/滑动）
operate_time_mills	Long	当前操作执行时间
input	String	当前节点执行测试的输入信息
nextNode	List<TestNode>	当前测试执行节点在总执行路径序列中的后继节点
preNode	String	当前测试执行节点在总执行路径序列中的前继节点

识别得到的控件图片将存储在 OSS 中。Widget 类具体内容如表 3-26所示。

表 3-26: Widget 类设计描述

属性	数据类型	说明
id	String	控件 ID
left	Double	控件左侧坐标
right	Double	控件右侧坐标
up	Double	控件上侧坐标
down	Double	控件下侧坐标
category	Enum	控件类型
ocrText	String	控件中提取得到的文本信息
picURL	String	控件截图在云端 OSS 存储中的下载 URL

3.4.4 系统数据库设计

数据库实体关系如图 3-11所示，其中实体总共有待测应用，测试设备，用户，测试任务，结果报告，测试执行节点，识别控件共 7 个。用户主键为用户 ID，每个用户将会上传多个待测应用，待测应用主键为应用 ID，用户 ID 为外

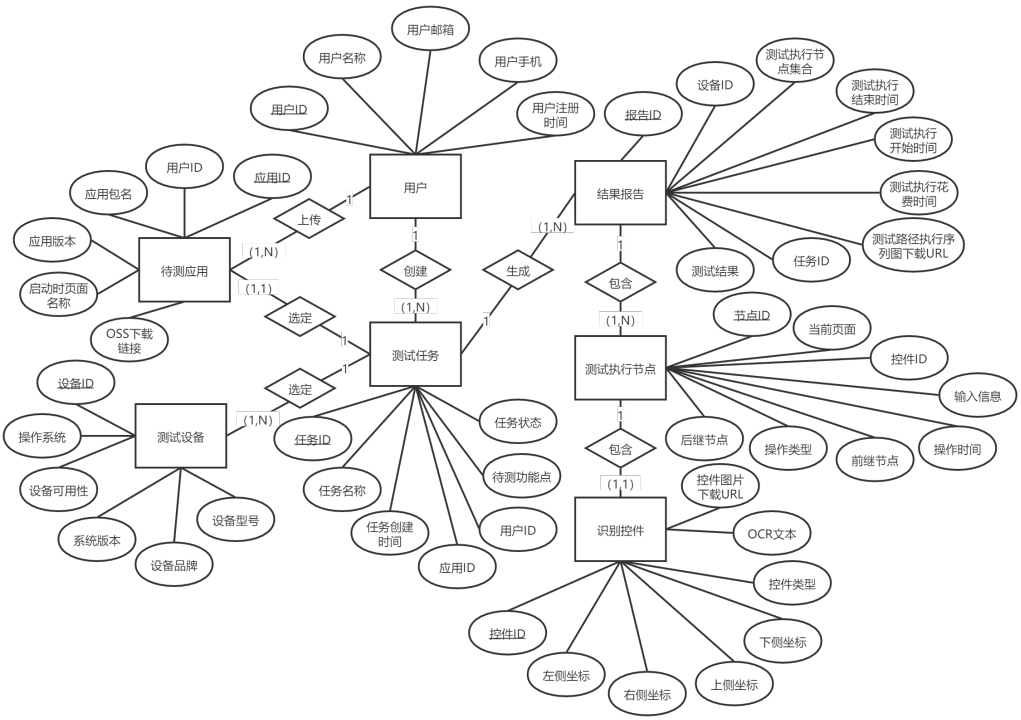

图 3-11: 数据库实体关系图

键。用户会创建多个测试任务，每个测试任务需选定一个待测应用，可以选择多个测试设备进行多次测试。测试任务主键为任务 ID，应用 ID、用户 ID 为外键。测试设备主键为设备 ID。测试任务可以生成多份结果报告，结果报告主键为报告 ID，设备 ID、任务 ID 为外键。结果报告中可以包含多个测试执行节点组成执行序列，测试执行节点主键为节点 ID，控件 ID 为外键。每个测试执行节点对应一次控件的操作，识别控件的主键为控件 ID。其余实体类的详细属性可以在图 3-11以及 3.4.3 节中找到详细解释，故在此不再赘述。

3.5 本章小结

本章对基于知识抽取的跨应用功能点测试系统的需求分析进行了概述。首先对系统涉众、功能性需求、非功能性需求进行了分析，接着使用系统用例图以及系统用例总表和分表的形式对系统各个用例以及功能进行了详细的描述，随后用系统架构图对系统整体架构进行了详细的介绍并且对系统模块进行了划分，之后利用“4+1”视图模型通过系统开发视图、进程视图、逻辑视图从多个角度对系统架构描述进行了补充。最后通过系统实体类设计图对系统运行中

涉及到的数据实体进行了说明，并将其映射为数据库表，通过 ER 图说明了数据库表之间的关系以及表中各个字段的意义，为第四章阐述系统的各个模块的详细设计以及实现打下了基础。

第四章　系统详细设计与实现

4.1　测试资源管理模块的设计与实现

4.1.1　测试资源管理模块概述

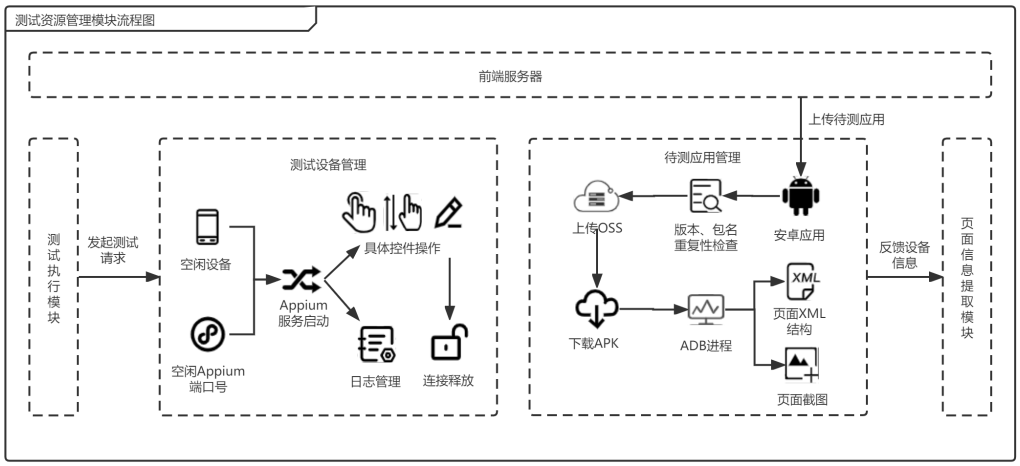

图 4-1: 测试资源管理模块流程图

测试资源管理模块是系统用于管理测试过程中需要用到的物理资源，作为测试执行模块的实际操作方以及页面信息提取模块的信息来源方。测试资源管理模块的流程图如图 4-1所示。本模块主要负责管理的测试资源包括测试设备机以及待测应用两类。测试设备管理主要与测试执行模块进行交互，测试执行模块发起执行测试需求，测试设备管理将会在测试设备机柜中检测目前处在空闲状态的机器，反馈给前端用户进行选择。测试流程开启时，测试设备管理部分将会不断顺延地查询目前空闲的 Appium 端口号，随后链接空闲设备与空闲端口，在该端口开启 Appium 服务，随后由该服务执行测试过程中的具体控件操作并拉取该端口的 Appium 日志统一在系统日志中进行管理。测试执行完成后，连接将被释放，占用的测试设备将重新回到空闲状态同时相应的 Appium 端口也会被释放。待测应用管理首先由用户在前端浏览器发起上传待测应用请

求，接着将调用 aapt 工具进行 apk 的解析，分析其包名、版本号，若云端 OSS 库中已有相同的包名与版本号的应用 APK 存在，则上传失败，系统将提示用户该应用已存在。若云端 OSS 库中未有该应用 APK 存在，则可以进行待测应用的上传，上传过程将首先继续利用 aapt 工具解析 APK 的诸如启动时页面等更多信息。解析完成后将 APK 文件上传至云端 OSS 存储，APK 相关信息则被存储在系统数据库中。当测试执行流程启动时，将首先从云端 OSS 下载 APK 文件并安装到测试机中，接着将开启当前执行设备状态检测进程，以在必要时获取页面 XML 结构文件以及屏幕截图反馈给页面信息提取模块。

4.1.2 测试资源管理模块核心类图

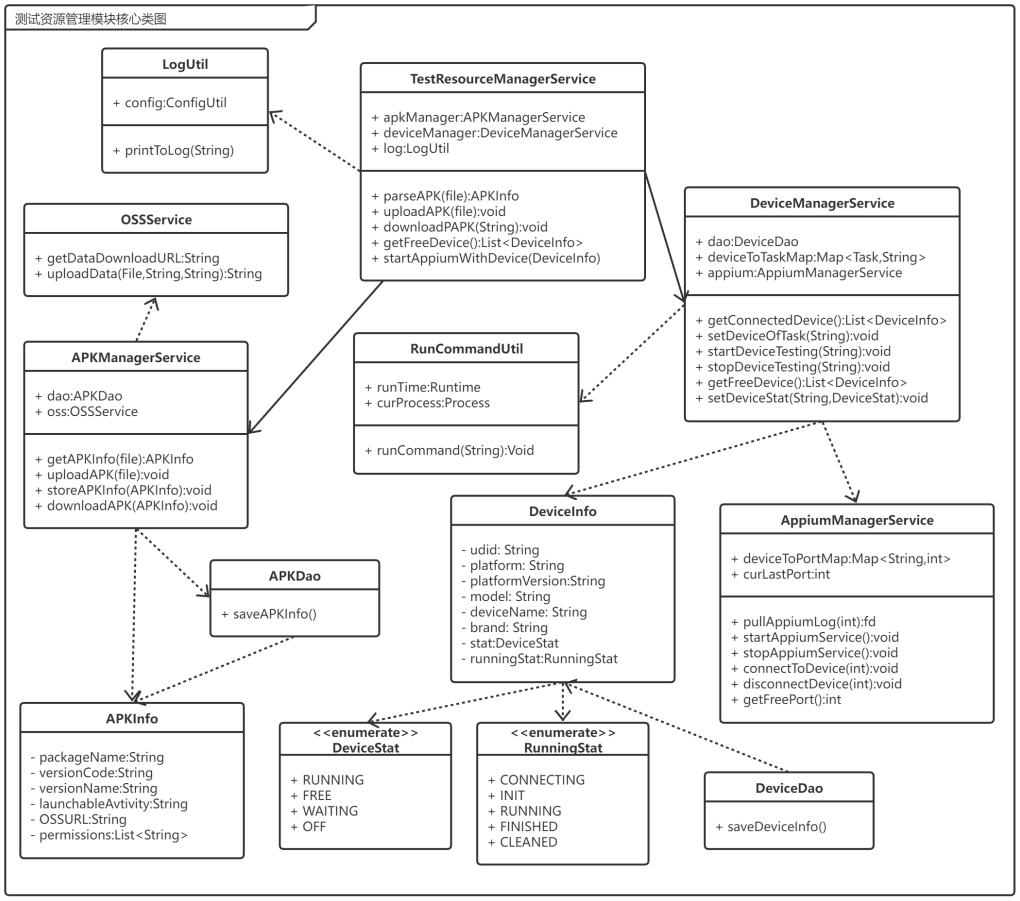

图 4–2: 测试资源管理模块核心类图

测试资源管理模块核心类图如图 4–2所示。其中 TestResourceManagerService 为该模块的核心服务类，其分别通过 APKManagerService 实现待测应用管

理服务和 DeviceManagerService 实现测试设备管理服务。LogUtil 类为日志打印类，主要负责对系统运行时数据按统一格式输出到日志文件中，系统日志按天分为不同的日志文件夹，每天内按照不同的测试任务拆分为不同的日志文件，系统最多保留三十天内的日志文件过期将自动清理。APKManagerService 主要负责解析、上传、下载待测应用，解析后的待测应用信息将会封装为 AppiumManagerService 并调用 APKDao 将其存储在数据库中。APKManagerService 调用 OSSService 上传 apk 文件到云端 OSS 存储，同时支持将远端文件通过链接下载至本地。DeviceManagerService 主要负责维护和更新测试机柜内的设备当前状态、开始和终止设备执行的状态、查询当前与系统已连接的设备、并为测试任务分配可用设备。测试设备相关信息被封装为 DeviceInfo 类，其中包含两个状态枚举类，分别为设备可用状态与设备任务执行状态。测试设备相关信息由 DeviceDao 类负责存储。AppiumManagerService 主要负责寻找空闲端口启动 Appium 服务与测试设备连接以及管理 Appium 连接的维护与释放。DeviceManagerService 完成功能时需要调用 RunCommandUtil 类执行特定系统命令如 ADB 命令、Appium 相关命令等。

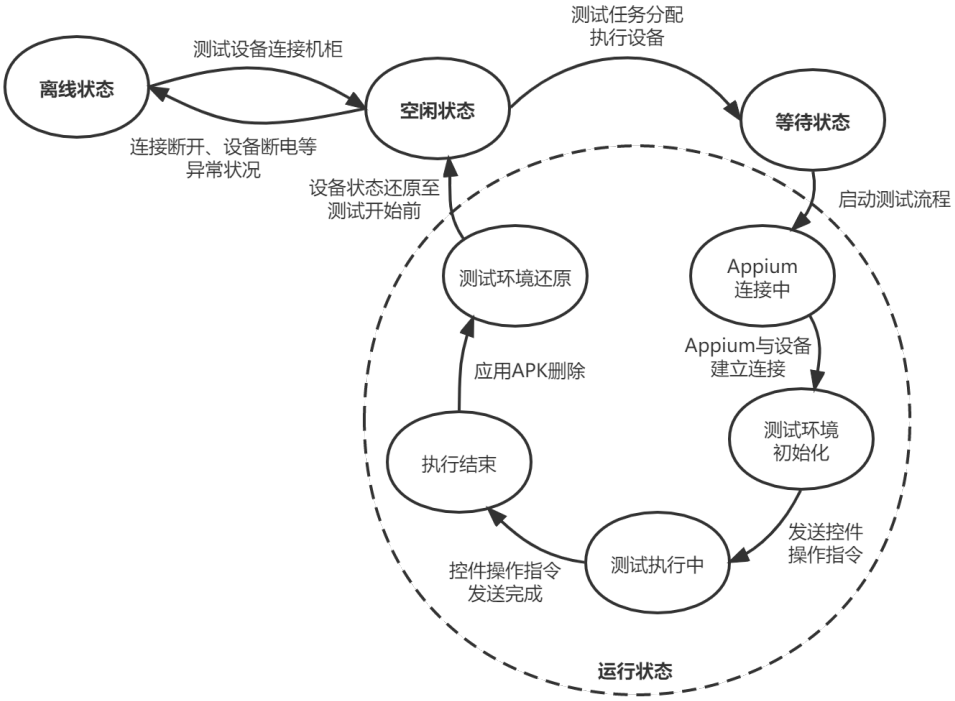

图 4-3: 测试设备状态转换图

测试设备状态详细转换图如图 4-3所示。测试设备分为离线状态、空闲状

态、等待状态、运行状态共四个状态。设备初始为离线状态，此时测试设备还未连接到测试设备集群，系统无法获取设备信息。当设备通过 USB Hub 连接到设备集群后，设备进入空闲状态。此时系统可以获取到设备的操作系统、版本、厂商、型号等信息，并同时开启监控线程监控设备当前状态。若突然发生连接断开、设备断电等异常状态，设备将重新回到离线状态。空闲状态中的设备可被系统获取反馈给用户进行测试任务的分配，被分配到测试任务的设备将进入到等待状态，用户启动测试流程后，设备就会从等待状态进入到运行状态。运行状态中的设备会再被细分为五个小状态，分别为连接中、初始化、执行中、执行结束、已还原。设备接收到测试流程启动指令后，将首先从等待状态进入连接中。该状态内的设备主要由 Appium 管理服务寻找空闲端口与设备建立连接，连接建立成功后设备就进入初始化状态。初始化状态内设备主要完成应用 APK 安装、输入法切换等测试执行准备工作。初始化完成后，系统就进入执行中状态，能够接受外部控件操作指令。整个测试流程完成没有后续控件操作指令传达后，设备就进入结束状态。结束状态中设备主要完成应用 APK 删除工作等设备还原测试环境的操作。测试设备环境还原至测试执行开始前后，该设备就将重新回到空闲状态，可以再次接受用户分配测试任务的请求。

4.1.3 测试资源管理模块顺序图

测试资源管理模块顺序图如图 4-4所示。TestResourceManagerService 首先向 APKManagerService 发起上传 APK 请求，APKManagerService 向 RunCommandUtil 发起解析 APK 信息请求。RunCommandUtil 返回 app 包名、版本号等信息，APKManagerService 在进行 APK 存在性检查。若相同的 apk 文件已存在，则向 TestResourceManagerService 返回待测应用 APK 已存在，无需重复上传。若 apk 文件未存在，则向 OSSService 发起上传 APK 请求，OSSService 返回上传结果。用户选择启动测试流程后，TestResourceManagerService 向 APKManagerService 发起启动测试流程，APKManagerService 向 OSSService 查询该 APK 的下载 URL，获得 OSS 返回的下载链接后，APKManagerService 将下载完成的 APK 发送给 DeviceManagerService。DeviceManagerService 向 AppiumManagerService 发起连接 Appium 与测试设备的请求，AppiumManagerService 向 RunCommandUtil 发起寻找空闲 Appium 端口号已经启动 Appium 服务的请求，RunCommandUtil 返回建立好的 Appium 连接。DeviceManagerSerivce 向 RunCommandUtil 发起初始化测试环境的请求，RunCommandUtil 进行 APK 安

装与输入法切换并返回初始化完成信号。DeviceManagerService 与 AppiumManagerSerivce 循环地进行控件操作指令发送与操作结果返回的过程直至测试流程终止，DeviceManagerService 向 AppiumManagerService 发起终止测试流程的请求，AppiumManagerService 向 RunCommandUtil 发起释放连接请求。RunCommandUtil 经过 Appium 服务关闭与测试环境还原后向 DeviceManagerService 返回测试环境已还原信号。DeviceManagerService 再向 TestResourceManagerService 返回测试设备已回到空闲状态。

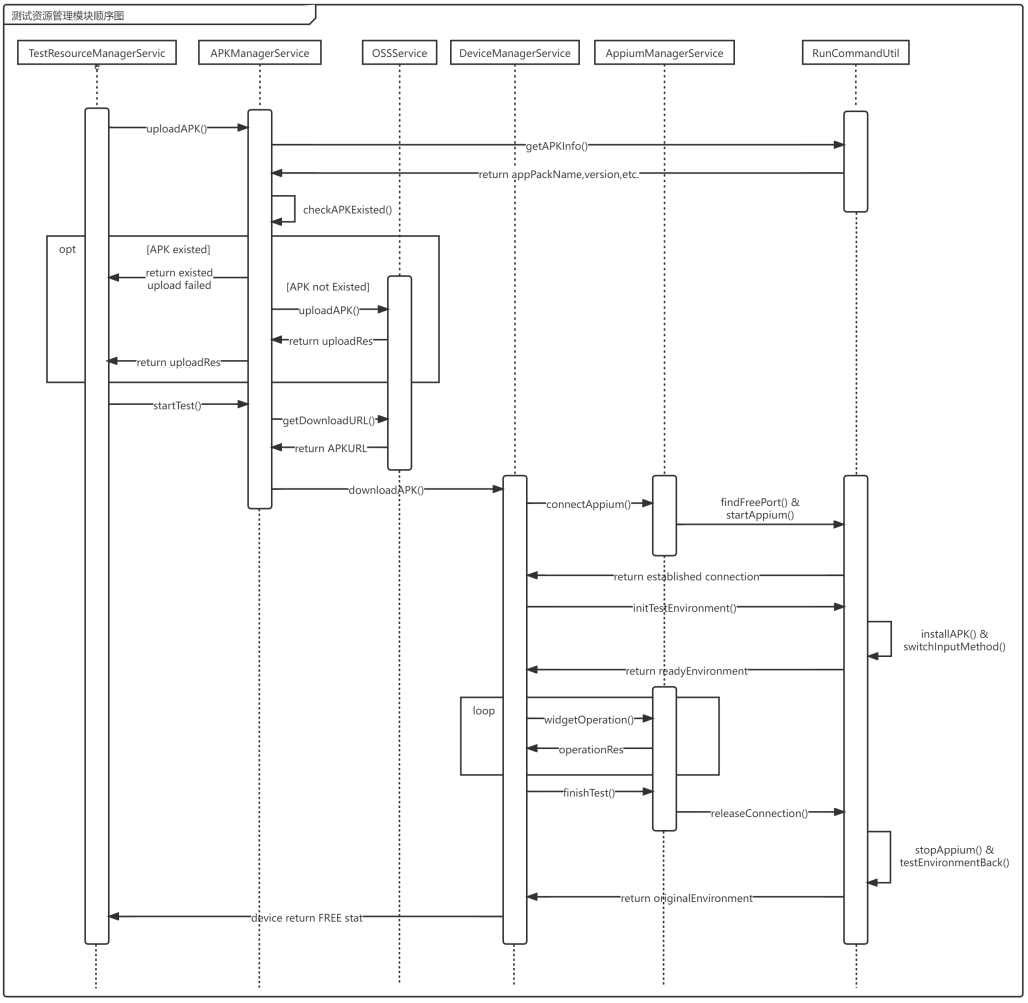

图 4-4: 测试资源管理模块顺序图

```
def get_app_info(apk_name, apk_path):
    # 检查 apk 文件路径合法性
    if not os.path.exists(apk_path):
        print("Apk file not exist!")
        return
    command = 'aapt dump badging {}'.format(apk_path)
    # 构造 popen 调用 aapt 工具解析 apk 信息
    popen = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
universal_newlines=True, bufsize=1, encoding='utf-8')
    out, err = popen.communicate()
    # aapt 返回结果解析
    items = out.split("\n")
    item_list = [x for x in items if x != '']
    for item in item_list:
        # 遍历每个 key-val 对象
        tmp = item.split(': ')
        if tmp[0] == 'package':
            val_list = item_vals.split(' ')
            for val in val_list:
                if val.split('=')[0] == 'name':
                    # 获取 apk 包名
                    package_name = val.split('=')[1]
                    package_name = package_name[1:len(package_name)-1]
        elif tmp[1] == 'launchable-activity':
            val_list = item_vals.split(' ')
            for val in val_list:
                if val.split('=')[0] == 'name':
                    # 获取 apk 启动时页面
                    activity = val.split('=')[1]
                    launchable_activity = activity[1:len(launchable_activity)-1]
    return package_name, launchable_activity, apk_path
```

图 4-5: APKManagerService 的 get_apk_info 方法核心代码

4.1.4 待测应用管理关键代码

图 4-5所示为 APKManagerService 的 get_apk_info 方法核心代码。该方法首先检查传入的 apk 文件路径参数合法性。解析 apk 信息需要利用 aapt 工具，该工具通过在命令行中输入 apk 路径，能够得到 key-val 形式的 apk 属性作为命令行显示结果。由于需要执行 aapt 系统命令并获取输出，因此需要使用系统的 popen 接口。由于 aapt 解析结果中有可能存在中文字符，为避免中文乱码问题

```
# OSS 秘钥等相关配置项
access_id = 'XXX'
access_secret = 'XXX'
bucket_name = 'XXX'
endpoint = 'XXX'

def upload_apk(path, package_name, version):
    # 创建 Bucket 对象
    bucket = oss2.Bucket(oss2.Auth(access_id, access_secret), endpoint, bucket_name)
    # 上传 apk 文件
    with open(path, "rb") as f:
        data = f.read()
    bucket.put_object(package_name+'_'+version, data)
    # 构造 OSS 云端存储文件的对应 URL
    url = endpoint+ package_name+'_'+version
    return url

def downloadFiles(local_path, file_name):
    # 检查 OSS 云端是否存在该下载文件
    if not bucket.object_exists(file_name):
        print("File {0} is not on the OSS!".format(tmp_file))
        return
    # 检查本地保存路径是否已存在
    if not os.path.exists(local_path):
        os.makedirs(local_path)
    start_time = time.time()
    download_times = 0
    # 允许下载失败重试
    while download_times <= threshold:
        oss2.resumable_download(bucket, file, path, progress_callback = percentage)
        if download_success:
            break
        download_times += 1
    print("Downloads finished, cost {0} Sec.".format(time.time() - start_time ))
```

图 4-6: APKManagerService 的上传与下载 APK 方法核心代码

需要使用能够设定输出编码格式的 subprocess.Popen 接口，该接口会在主进程的基础上产生一个子进程并连接到子进程的标准输入、输出、错误输出中，从而可以获取子进程命令执行的返回值。get_apk_info 方法通过构造 popen 并指定好标准输出为系统管道、输出编码格式为'utf-8' 参数，即能够获得 aapt 命令返回的结果。由于 aapt 命令所得为类似 key-val 的 json 格式数据，因此仍然需进

一步遍历每个 key-val 对象并检查键值从而获取所需的属性值。

图 4-6所示为 APKManagerService 的上传与下载 APK 方法核心代码。首先需要在 ConfigUtil 中获取 OSSkey、秘钥、bucket 名称、endpoint 等属性。上传时需要首先创建 Bucket 对象，将本地 APK 文件以二进制文件流的形式打开，构造 OSS 云端存储文件对应的下载 URL 为 endpoint 链接 APK 包名加版本号。下载 APK 时首先需要检查 OSS 云端是否存在该下载文件，同时检查本地文件下载目录是否已创建。本方法允许下载失败重试避免网络因素影响下载，并设定了最大重试次数将云端 APK 文件下载到给定的本地目录下。如果在最大重试次数内都没有成功完成 APK 文件下载，则会抛出 APK 下载未成功异常。

4.1.5 测试设备管理关键代码

图 4-7所示为 DeviceManagerService 的 get_connected_device 方法核心代码。该方法主要负责通过“adb devices”命令获取目前已连接测试设备集群的所有设备信息。由于同样需要执行系统命令以及获取输出结果，因此也必须要调用系统 popen 接口执行命令。由于 adb 命令的返回结果中没有中文字符存在，无须指定 utf8 编码格式，因此只需选用 os.popen 的普通接口即可。命令返回结果为设备 udid 以及设备名的多行文本因此需要进一步字符串处理切割换行以及分割行内的两个属性值，以获取连接系统的多个设备的 udid 信息。

```
def check_device_connected():
    with os.popen(r'adb devices', 'r') as f:
        text = f.read()
    # 输出结果字符串处理
    s = text.split("\n")   # 切割换行
    result = [x for x in s if x != '']   # 生成式去掉空行
    # 获取多台设备的信息
    devices = []
    for i in result:
        dev = i.split("\tdevice")
        if len(dev) >= 2:
            devices.append(dev[0])
    if not devices:
        return False
    else:
        return True
```

图 4-7: DeviceManagerService 的 get_connected_device 方法核心代码

图 4-8所示为 DeviceManagerService 的 appium_init 方法核心代码。本方法主要负责寻找空闲端口号开启 Appium 服务与该设备连接。本方法首先检查当前系统中是否有未释放的 Appium 连接，并检查给定设备先前建立的 Appium 连接是否已经释放了。接着寻找 Appium 服务的空闲端口号，并在 udid 和端口号的映射内添加该设备 udid 与新的空闲端口号的组合。之后设置 Appium 服务启动相关参数，包括设备名称等相关参数、APK 文件路径及启动时页面等相关参数与输入法相关属性。设置完成后通过远端 ip 地址与空闲端口号设置 Appium 服务器连接地址并传入设置好的启动参数连接 Appium 服务。

```
def appium_init(self, device_name, udid):
    # 检查当前是否有未释放的 Appium 连接
    if self.driver is not None:
        self.appium_quit()
    # 检查当前设备是否已经开启 Appium
    port = check_device_port(udid)
    # 寻找空闲端口号并在 udid 和端口号映射内添加该组合
    if self.cur_port is null:
        self.cur_port = get_free_port()
        self.udid_to_port_map[udid] = port
    desired_caps = {'browserName': '', 'platformName': 'Android',
                    'deviceName': device_name, 'appPackage': self.app_package,
                    'appActivity': self.app_activity, 'noSign': 'true',
                    'app': self.app_path,   # apk 路径
                    #设置使用 unicode 键盘，支持输入中文和特殊字符
                    'unicodeKeyboard': 'true',
                    'resetKeyboard': 'true',   # 设置用例执行完成后重置键盘}
    remote_url = "http://{0}:{1}/wd/hub".format(self.ip_address, self.cur_port)
    self.driver = webdriver.Remote(remote_url, desired_caps)   # 连接 Appium
    self.driver.implicitly_wait(8)
```

图 4-8: DeviceManagerService 的 appium_init 方法核心代码

4.2 测试上下文管理模块的设计与实现

4.2.1 测试上下文管理模块概述

测试上下文管理模块主要负责管理和维护测试过程中的测试执行路径与场景信息，为测试执行模块记录其从测试开始以来的执行路径分支信息，帮助测

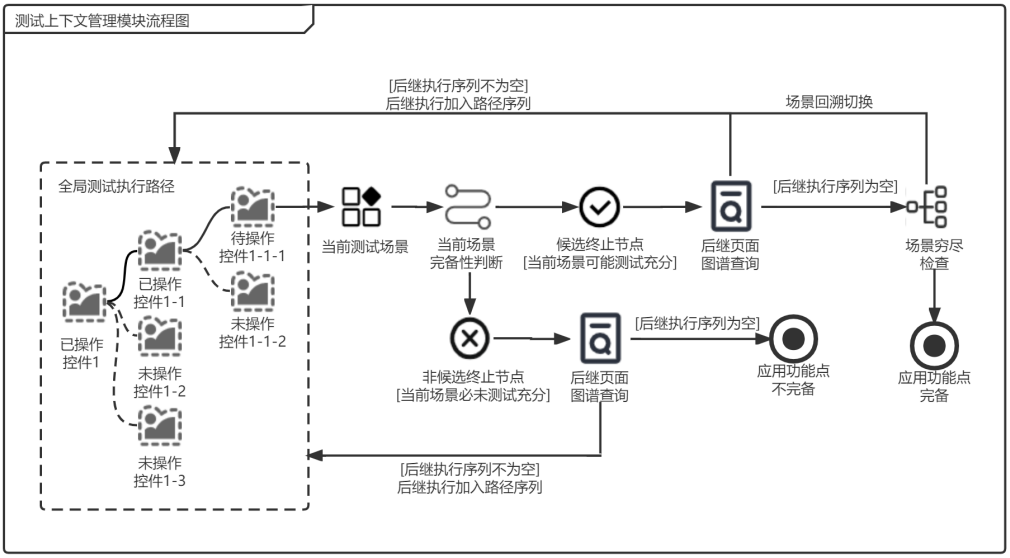

图 4-9: 测试上下文管理模块流程图

试执行模块划分功能场景并实时进行场景的更新与切换。测试上下文管理模块的流程图如图 4-9所示。测试上下文管理模块接受来自测试执行模块的执行信息并在整个测试过程中维护全局测试执行路径，全局测试执行路径中的测试执行节点主要记录已操作的控件位置信息、当前所处页面、所执行的操作以及输入的信息。整个测试执行的过程类似于深度优先遍历，每次需要将当前正在测试的功能点场景探索完成后再切换至下一场景。本文中对功能点场景的定义为在应用中能够实现功能的每一控件操作序列都能被称为该功能点的一个场景。全局测试执行路径中正在进行探索的路径序列则为当前测试场景。每次测试执行模块进行相应的控件操作后，测试上下文管理模块需要判断该测试执行节点是否为候选执行终止节点。判断的依据即为当前操作是否与知识图谱中记录的某些应用在实现本功能点的终止节点相似。若当前测试执行节点不为候选终止节点，则说明当前场景必未探索充分，因此仍然需要对后继页面进行页面信息提取以及图谱匹配检索。若图谱返回结果中的后继执行序列不为空则证明找到了匹配的后继可操作控件，因此执行相应的控件操作并将具体操作情况加入到全局测试执行路径中。若后继测试执行序列为空，则说明此时实际执行情况与知识库中学习到的功能点操作逻辑产生了冲突，图谱认为该功能点的该场景仍然可探索但已没有可执行的控件，证明应用该功能点在此场景上并不完备。若当前测试执行节点为候选终止节点，则说明该场景可能已探索充分。此时依然需要对后继页面进行页面信息提取以及图谱匹配检索。若图谱返回结果中的后

继执行序列不为空则说明该场景仍可继续探索、功能点尚未完成，前继节点并不为真正的终止节点，同样执行相应的控件操作并加入全局测试执行路径。若后继测试执行序列为空，说明知识库中认为前继节点为终止节点并且实际执行情况也佐证了这一点，此时可证明已通过当前测试场景完成了功能。接着需要判断全局测试执行路径中是否存在其他未探索场景待测。若全局测试执行路径中已无未遍历过的分支，说明该功能点的所有场景已经探索完成且每个场景的测试过程中均未出现问题，此时可输出应用功能点完备结果。若仍有分支未遍历到，说明该功能点仍有场景待测，此时需要将已执行的控件操作回溯至未遍历的分支处页面及应用状态，并切换当前场景继续完成测试。

4.2.2 测试上下文管理模块核心类图

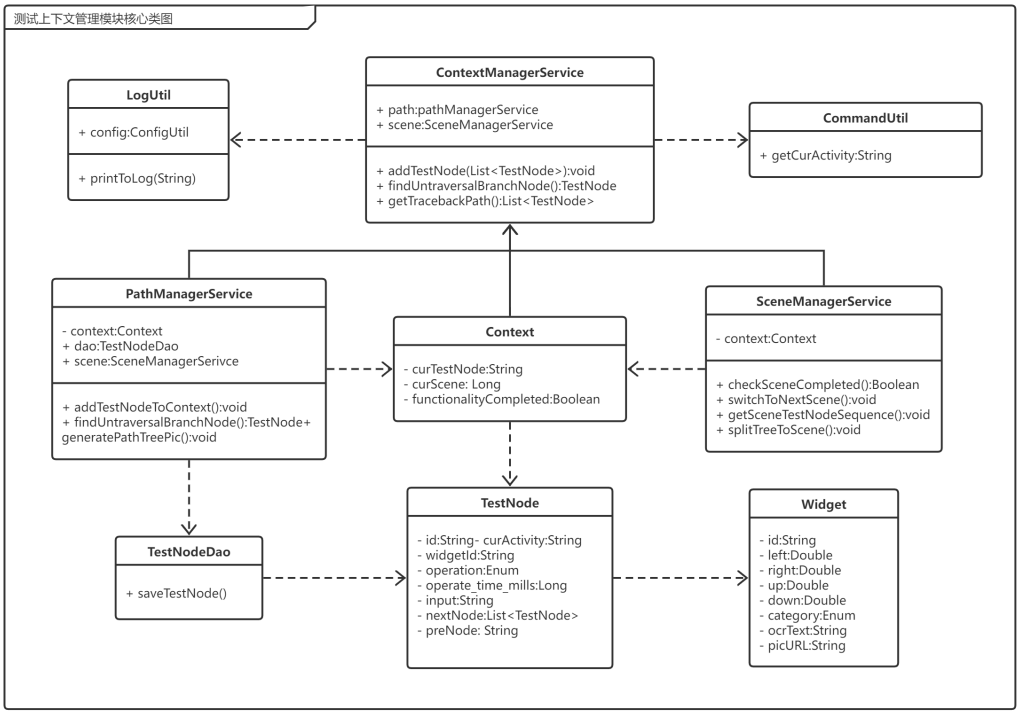

图 4-10: 测试上下文管理模块核心类图

测试上下文管理模块的核心类图如图 4-10所示。其中 ContextManagerService 为该模块的核心服务类，其分别通过 PathManagerService 类实现对全局测试执行路径的更新和维护以及通过 SceneManagerService 实现对场景的划分与切换。LogUtil 类为日志打印类，在多个模块中都需要引用，在此不再赘述。

CommandUtil 为工具类，负责执行系统命令获取当前应用页面名称，标注测试执行所处页面位置情况。PathManagerService 负责管理和维护当前测试过程中的全局测试执行路径信息，其中测试上下文信息抽象为 Context 类，主要负责记录当前执行到的测试执行节点、当前所在场景以及功能测试充分性。Context 类其中存储的当前测试执行节点属性主要抽象为 TestNode 类，保存如当前所执行的操作类型、操作的控件所在页面名称、操作的控件位置，而 TestNode 又依赖 Widget 类所封装的控件元素对象。Widget 类中的重要属性元素包括控件的上下左右位置坐标、控件类别以及控件 OCR 文本。PathManagerService 依赖 TestNodeDao 将记录的测试执行节点持久化保存用于生成测试结果报告。SceneManagerService 主要负责将全局测试执行路径划分为不同的场景，并在测试过程中判断当前场景的测试充分性以及完成场景的切换更新。

4.2.3 测试上下文管理模块顺序图

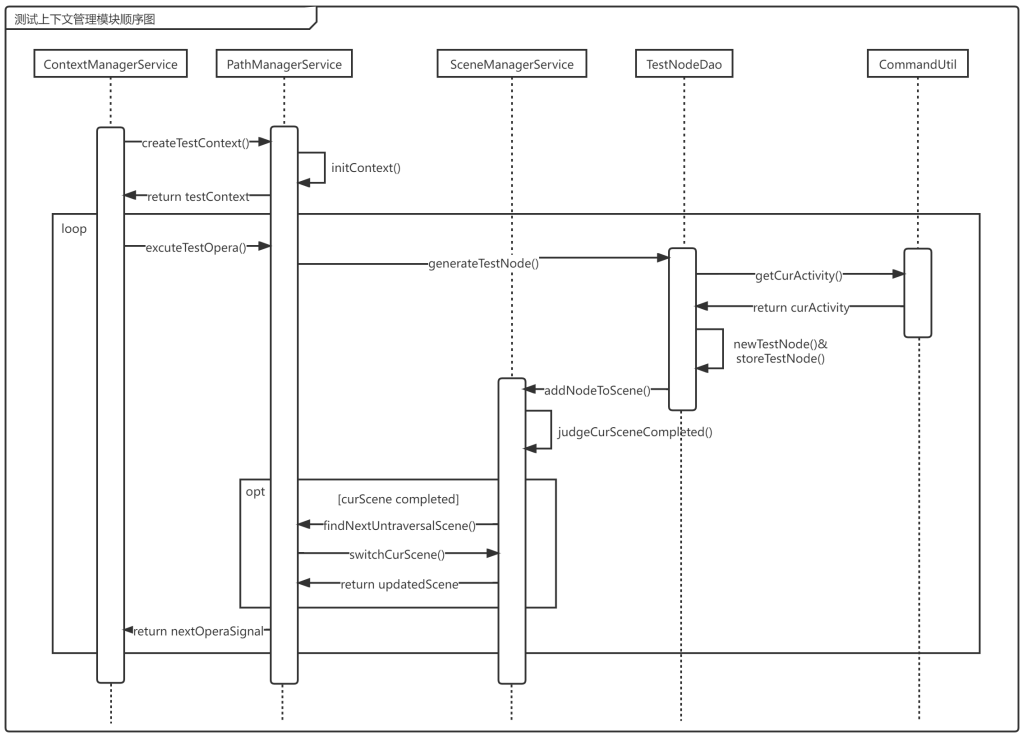

图 4-11: 测试上下文管理模块顺序图

测试上下文管理模块的顺序图如图 4-11所示。ContextManagerService 首先向 PathManagerService 发起创建测试执行上下文的请求，PathManagerService

进行全局测试执行上下文信息的初始化并返回初始化完成的已封装的 Context 对象。随后将进行迭代的完整测试过程，ContextManagerService 向 PathManagerSerivce 传递已经执行的测试操作信息，PathManagerService 随后向 TestNodeDao 发起新建测试执行节点的请求，TestNodeDao 向 CommandUtil 请求获取当前应用所执行到的页面名称，获取返回值后进行测试执行节点的新建以及存储并将该测试执行节点加入到当前测试执行场景中。随后 SceneManagerService 将判断当前场景是否已经探索完成。若当前场景已经探索完成，SceneManagerService 向 PathManagerService 调用获取全局测试执行中仍然未遍历的分支，并将当前场景进行切换，PathManagerService 则向 ContextManagerService 返回从根节点到新场景的回溯执行操作序列。最后 ContextManagerService 会释放可以进行下一次控件操作的讯号，等待后续操作执行。

4.2.4 测试上下文管理关键代码

图 4-12所示为 ContextManagerService 的 TestNode 类核心代码。TestNode 类主要负责封装测试执行信息，包括操作的控件类别、控件文本信息、操作类型、控件上下左右四点坐标。在创建 TestNode 对象时会初始化当前控件是否已操作状态为否、后继节点列表为空、前继节点为空。Context 类可以看做 TestNode 节点构成的树状结构，根节点为初始页面待操作的控件元素对象，后继节点信息通过知识图谱匹配并筛选页面信息提取模块分析出的当前页面中的控件元素对象。由于测试执行节点包括指向后继节点以及前前继节点的双向指针，因此能够非常方便的从当前正在执行的对象向前回溯或是从根节点向后获取到任意节点的执行序列用于回溯。由于初始化时，当前测试执行节点的前继节点、后继节点都为空，因此需要暴露该两个属性的接口，前后继关系都在知识图谱返回匹配结果后对 Context 类中的当前测试执行节点进行更新。同样由于初始化时当前测试执行节点并未执行，因此也许暴露更新已执行信息的接口用于在测试执行模块实际执行相应操作时再修改相应属性。

图 4-13所示为 ContextManagerService 的 Context 类核心代码。初始化方法中需要创建空 TestNode 对象做为假头节点，主要用于处理初始页面就存在分支的情况，有一个假头节点可以指向初始的各个分支。创建假头节点后，需要将其设置为根节点与当前操作节点。add_test_node 方法主要负责将图谱匹配的当前页面待操作控件封装为测试执行节点并加入测试执行上下文中。当前操作测试执行节点的后继节点列表需更新为传入的测试执行节点列表，同时传入的所

```
def TestNode:

    def __init__(self, category, text, operation, cnt, cnt_id, x1, y1, x2, y2,
                 pre_node, tag=-1):
        # 初始化设置测试执行节点的属性
        self.widget_category = category
        self.ocr_text = text
        self.widget_operation = operation
        self.cnt = cnt
        self.cnt_id = cnt_id
        self.left = x1
        self.right = x2
        self.top = y1
        self.bottom = y2
        self.already_opera = False
        self.next_nodes = []
        self.pre_node = pre_node
        self.id = tag

    def add_next_nodes(self, nodes_list):
        # 添加当前节点的后继节点
        self.next_nodes = nodes_list
        for node in nodes_list:
            node.set_pre_node(self)

    def set_pre_node(self, node):
        # 设置当前节点的前继节点
        self.pre_node = node

    def set_operated(self):
        # 设置当前节点已经操作完成
        self.already_opera = True
```

图 4-12: ContextManagerService 的 TestNode 类核心代码

有测试执行节点也需要设置其前继节点为当前操作测试执行节点。默认优先执行列表中的第一个测试执行节点，即优先探索第一个分支。完成上述操作后更新当前操作节点为列表中的第一个测试执行节点。find_traversal_node 方法主要负责回溯寻找当前未遍历到的分支节点。从当前操作测试执行节点开始，只要当前节点不为空则一直遍历寻找，若当前节点的后继节点列表长度大于 1，则说明当前节点后续存在分支需要检查。遍历当前节点的每个后继节点，若有节

```
class Context:

    def add_test_node(self, test_node_list):
        # 将图谱匹配结果加入测试执行上下文中
        self.cur_opera_node.add_next_nodes(test_node_list)
        for node in test_node_list:
            node.set_pre_node(self.cur_opera_node)
        # 优先执行待操作控件中的第一个节点，即优先探索第一个分支
        self.cur_opera_node.next_nodes[0].set_operated()
        # 更新当前操作节点
        self.cur_opera_node = self.cur_opera_node.next_nodes[0]

    def find_traversal_node(self):
        # 回溯寻找当前未遍历到的分支节点
        cur_node = self.cur_opera_node.pre_node
        # 若当前节点不为空则一直遍历寻找
        while cur_node is not None:
            if len(cur_node.next_nodes) > 1:
                for node in cur_node.next_nodes:
                    if not node.already_opera:
                        new_opera_node = node
                        test_sequence = []
                        # 构造从新节点到根节点的路径
                        while node.pre_node is not None:
                            test_sequence.append(node)
                            node = node.pre_node
                        # 反转路径得到从根节点开始的路径
                        test_sequence = test_sequence[::-1]
                        self.cur_opera_node = new_opera_node
                        return test_sequence
            # 未找到则继续向上回溯
            cur_node = cur_node.pre_node
    return None
```

图 4-13: ContextManagerService 的 Context 类核心代码

点当前未操作则说明已经找到了满足条件的节点。此时需要构造从新节点开始一直向上到根节点的路径。由于最终所需得到的为从根节点到新场景节点的测试执行操作序列，因此仍需将刚得到的路径进行翻转并返回。若当前未能找到满足条件的节点，则需将当前节点重置为其前继节点直至为空并继续搜索。

4.3 页面信息提取模块的设计与实现

4.3.1 页面信息提取模块概述

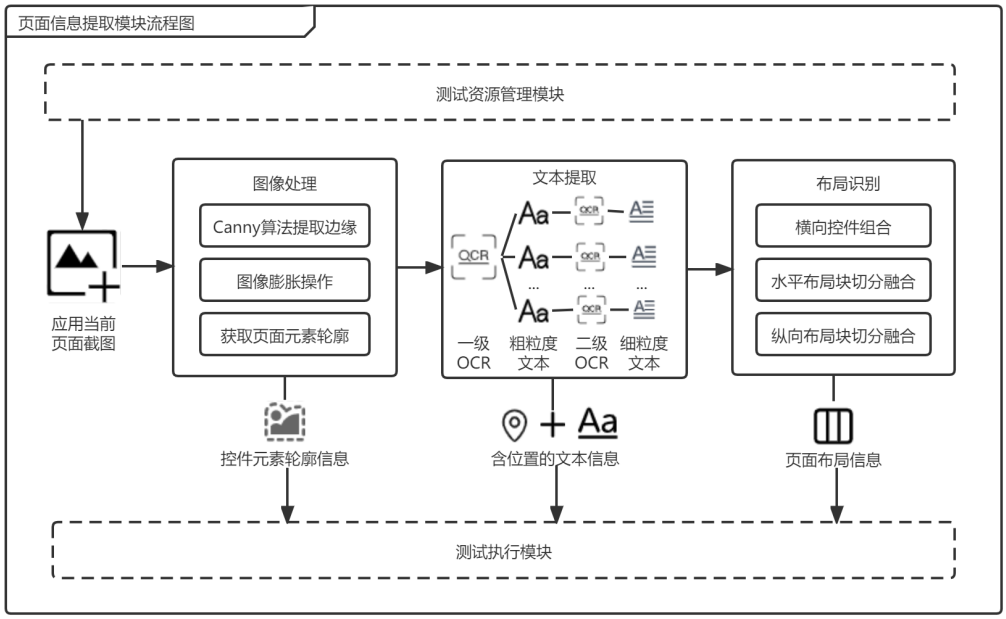

图 4-14: 页面信息提取模块流程图

页面信息提取是系统用于分析当前页面中的特征信息，利用识别到的布局及控件信息描述应用当前状态，为知识图谱判断当前在该功能点操作逻辑序列中的所处位置并筛选出待操作候选控件为测试引导的过程提供信息匹配来源。页面信息提取模块的流程图如图 4-14所示。系统在本模块内将首先调用测试资源管理模块的设备截图服务，获取应用当前页面截图。接着将对截图进行图像处理，首先利用 Canny 算法提取图像中的边缘信息，接着利用图像膨胀操作放大提取得到的边缘使其更易于提取，接着获取页面元素轮廓从而得到控件元素轮廓信息。处理完成的图像将继续进行页面中的文本信息提取，仅通过一次 OCR 识别出的结果有可能会产生误识别、错识别的情况，通过将第一次识别的 OCR 结果再次 OCR 构成的二级深度 OCR 能够减少产生误识别的几率，获得的文本信息将更加精确并且包含相应位置信息。布局识别部分将以控件元素识别结果为基础，通过控件的 y 坐标组合形成水平布局块再利用 x 坐标进行纵向布局块的切分从而得到页面布局信息。页面信息提取模块产生的控件信息、文本信息、布局信息将进一步通过在页面中的位置进行匹配、去重、组合形成更加

精确的结构化提取结果并反馈给测试执行模块以进行后续的测试执行。

4.3.2 页面信息提取模块核心类设计

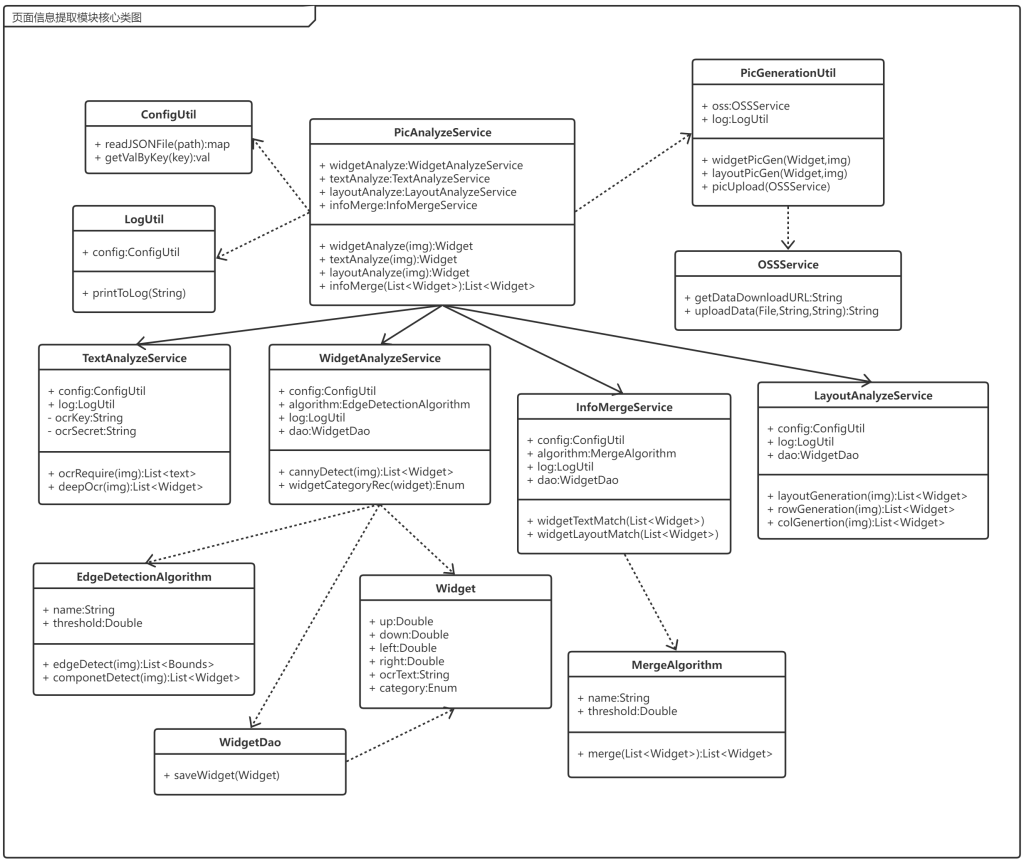

图 4-15: 页面信息提取模块核心类图

页面信息提取模块核心类图如图 4-15所示。其中 PicAnalyzeService 为本模块的核心服务类，其分别通过 WidgetAnalyzeService 实现控件识别、TextAnalyzeService 实现页面文本提取、LayoutAnalyzeService 实现布局识别能力。ConfigUtil 类为模块配置类，其中包含如默认算法、临时文件存储路径等信息，配置项存储在 json 文件中，系统可以通过指定对应键获取对应值。LogUtil 类为日志打印类，在多个模块中引用，在此不再赘述。PicGenerationUtil 为模块内工具类，主要功能为在原页面截图中标识系统的控件识别结果以及布局识别结果并保存为新的图片，生成的图片将调用 OSSUtil 类上传至云端 OSS 存储。TextAnalyzeService 主要负责提取页面中的控件文字，其内部包含调用百度 OCR 接口提取图片中文字的服务调用接口以及对初始 OCR 结果再次进行解

析合并的深度 OCR 接口。WidgetAnalyzeService 主要负责识别页面中的控件元素，其内部包括边缘检测接口以及控件类型识别接口。由于系统中配置了多种边缘检测算法，因此此处还需封装 EdgeDetectionAlgorithm 类负责抽象算法能力。识别得到的控件元素将封装为 Widget 类，并调用 WidgetDao 进行持久化存储。LayoutAnalyzeService 主要负责识别页面布局，其内部包括水平布局块生成，纵向布局块切分以及页面整体布局结构生成接口。InfoMergeService 主要负责对识别出的三类信息按照其页面位置进行匹配、去重、组合。

4.3.3 页面信息提取模块顺序图

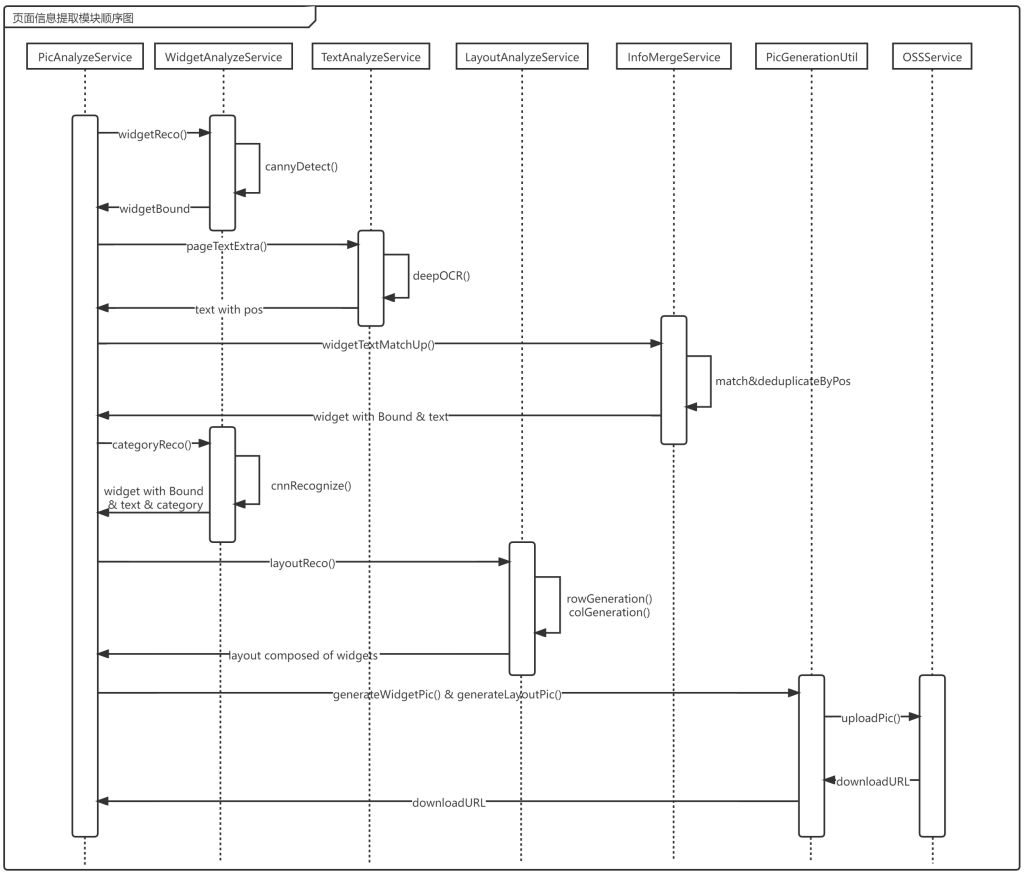

图 4-16: 页面信息提取模块顺序图

页面信息提取模块顺序图如图 4-15所示。PicAnalyzeService 将首先向 WidgetAnalyzeService 发起截图中控件识别服务调用。WidgetAnalyzeService 经过 Canny 边缘提取后将得到的控件 bounding box 信息返回给 PicAnalyzeService。随后 PicAnalyzeService 向 TextAnalyzeService 发起文本提取服务调用。

TextAnalyzeService 经过两次迭代 OCR 识别后将得到的带位置的文本信息返回给 PicAnalyzeService。PicAnalyzeService 随后将该文本信息以及控件 bounding box 信息传递给 InfoMergeService，其根据两个集合中的元素的 bounding box 先进行控件元素与其对应的文本内容的匹配，随后进行控件之间的去重，将得到的包含文本内容以及位置信息的控件元素属性封装为 Widget 类返回给 PicAnalyzeService。PicAnalyzeService 随后将包含文本内容以及位置信息的控件元素继续传递给 WidgetAnalyzeService，其在内部调用训练好的模型识别控件类别，并将具备位置、OCR 文本、类别完整三个属性的控件元素返回给 PicAnalyzeService。PicAnalyzeService 接着向 LayoutAnalyzeService 传递识别好的完整的控件元素对象，发起布局识别服务调用，LayoutAnalyzeService 经过横向纵向布局块识别获取得到布局识别结果并反馈给 PicAnalyzeService。PicAnalyzeService 最后将向 PicGenerationUtil 发送识别并匹配完成的控件元素和布局块结果，PicGenerationUtil 将在原截图的基础上表示识别结果并生成新的图片调用 OssService 上传到云端 OSS 存储，成功后将返回对应的下载 URL。

4.3.4 页面文字提取关键代码

图 4-17所示为 TextAnalyzeService 类中的 deep_ocr 方法核心代码。该方法首先将传入的页面截图调用一次百度 OCR 服务，获取整张图片的文本提取结果，OCR 服务返回的详细结果中包含文字及其对应的页面位置，即上下左右四个坐标，以及识别结果的准确概率。深度 OCR 则会继续对初始 OCR 结果进行过滤和核实，根据第一次的识别结果中的位置信息，调用 cv 相关接口裁剪原始页面截图，裁剪后的截图将再次调用 OCR 服务。与第一次初试结果返回页面中整句文本信息不同，第二次 OCR 将得到页面中每个整句的单字集合。对二次 OCR 所得结果，系统将首先过滤识别准确概率未达到阈值的。接着获取单字集合中的每两字间距，对于两字间距过小的则认为是 OCR 服务错误将实际一个字识别为了两个字，此时只能保留其中之一。完成识别结果过滤后，将最终所得 OCR 结果集合封装为包含文本的 Widget 对象进行存储。

4.3.5 页面控件识别关键代码

图 4-18所示为 WidgetAnalyzeService 类中的 canny_boundings 方法核心代码。该方法负责识别页面中的候选控件元素的轮廓信息。该方法首先将传入的

```
def deep_ocr(C, image_path, lang='CHN_ENG', prob=0.90, space_ratio=0.7):
    # 获取整张图片初始 OCR 结果
    result = ocr(open(image_path, 'rb').read(), lang)
    text_boxes = []
    image = cv.imread(image_path)
    for words in result['words_result']:
        left = words['location']['left']
        # 获取文字信息的左、右、上、下四个位置坐标
        # 基于获取到的位置信息对图片进行裁剪
        cropped = cv.imencode('.jpg', image[top + height, left + width, :])[1].tobytes()
        # 对裁剪的图片二次调用 OCR 服务获取结果
        words['details'] = details = ocr(cropped, show_char=True)
        for detailed_words in details['words_result']:
            # 根据 OCR 返回结果中的识别概率过滤可能误识别的
            if detailed_words['probability']['average'] < prob:
                continue
            split_idx = []
            for i, (p, q) in zip(detailed_words['chars'][:-1], detailed_words['chars'][1:]):
                # 再次调用的识别结果将从整句变为单字
                distance = q['location']['left'] - p['location']['left'] - p['location']['width']
                threshold = space_ratio * min(p['location']['height'], q['location']['height'])
                # 若字与字之间的距离过小，可能是将一个字错识别为两个
                # 需要将识别出的两个字过滤其中之一
                if distance > threshold:
                    split_idx.append(i + 1)
            for i, j in zip([0] + split_idx, split_idx + [len(detailed_words['chars'])]):
                # 生成包含文本的 Widget 对象
                text_boxes.append(Bbox(x, y, width + x, height + y, 'TEXT', texts))
    # 生成 OCR 识别结果标识图片
    draw_rectangle_show_save(image, text_boxes, path)
    return text_boxes
```

图 4-17: TextAnalyzeService 的 deep_ocr 方法核心代码

原始图片转化为灰度图，并利用预设的 canny_sigma 参数设定 Canny 算法中双阈值法的高低两个阈值。随后进行 Canny 边缘算法提取图像边缘信息并利用膨胀操作放大边缘部分信息量。最后就可以通过 findContours 接口获取截图中的边缘轮廓信息。获取到的控件元素 bounding box 信息会被封装在 Widget 对象内，同时在原始截图上标识生成控件识别结果并生成新的图片进行保存。

图 4-19所示为 InfoMergeService 类中的关键方法核心代码。merge 方法主要负责对 TextAnalyzeService 识别出的包含位置的 OCR 识别结果以及 Widget-

```
def canny_boundings(C, image_path, canny_sigma=0.33, dilate_count=4):
    image = cv.imread(image_path)
    # 检查路片路径是否合法
    assert image is not None, '图片路劲错误！'
    # 将原始图片转化为灰度图
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    v = np.median(gray)
    # 设定 Canny 双阈值法的高低阈值
    lower_threshold = int(max(0, (1 - canny_sigma) * v))
    upper_threshold = int(min(255, (1 + canny_sigma) * v))
    # Canny 边缘提取
    img_binary = cv.Canny(gray, lower_threshold, upper_threshold, -1)
    # 膨胀操作放大边缘信息
    dilated = cv.dilate(img_binary, None, iterations=dilate_count)
    # 获取边缘轮廓信息
    contours= cv.findContours(dilated, cv.RETR_EXTERNAL, cv .CHAIN_APPROX_SIMPLE)
    # Widget 对象封装
    boundings = [cv.boundingRect(c) for c in contours]
    bboxs = [Bbox(b[0], b[1], b[0] + b[2], b[1] + b[3], 'COMPONENT') for b in boundings]
    name = image_path.split('/')[-1][:-4]
    # 生成控件识别结果标识图片保存
    save_bboxs_as_json(bboxs, os.path.join(C.OUTPUT_COMPONENT_PATH, name + '.json'))
    draw_rectangle_show_save(cv.imread(image_path), bboxs, path)
    return bboxs
```

图 4-18: WidgetAnalyzeService 的 canny_bounding 方法核心代码

AnalyzeService 识别出的包含位置的控件识别结果进行匹配以及去重。该方法会对传入的 OCR 识别结果集合中的每个 bounding box 在控件识别结果集合中选择与其最为匹配的 bounding box。两个 bounding box 的匹配程度主要通过两个矩形的 IOU 值进行量化，并通过 intersect 方法实现。intersect 方法优先获取传入的两个 bounding box 矩形构成的相交矩形的宽和高，并计算该相交矩形的面积。相交矩形的面积的计算公式为两个矩形各自的面积相加再减去相交矩形的面积。intersect 方法返回三个值，分别为相交矩形面积与矩形 A 面积比值，相交矩形与矩形 B 面积比值以及相交矩形面积与并集矩形面积比值共三个值。找到与 OCR 结果 bounding box 的 IOU 值最大的控件识别结果的 bounding box 最为最佳匹配。接着 merge 方法会判断相交矩形面积与 OCR bounding box 自身矩形面积的比值，若比值高于阈值，则认为该最佳匹配合法，对应的原控件识别结果需要在集合中替换为该 OCR 识别结果并附带文本信息，同时过滤控件识

别结果集合中的其他相似 bounding box 进行删除；若比值低于阈值，则认为该最佳匹配不合法，即在原控件识别结果集合中未能找到与该 OCR 识别结果匹配的元素，因此需要在控件识别结果集合中新增该 OCR 识别结果。最终经过匹配、去重后的 Widget 集合为最终控件识别结果。

```
def merge(boundings, ocr_res, threshold=.70):
    mods = []
    for rect_o in ocr_res:
        for rect_c in boundings:
            ratio_o, ratio_c, ratio = intersect(rect_o, rect_c)
            # 获取与其 IOU 值最大的控件元素作为最佳匹配
            if ratio > best_match[3]:
                best_match = (rect_c, ratio_o, ratio_c, ratio)
        if rect_c is not None:
            if ratio_c > threshold:
                # 找到与 OCR 文本匹配的控件元素
                # 替换原控件元素为包含文本信息的控件元素
                mods.append(('replace', rect_c, rect_o))
                for rect_cc in boundings:
                    # 删除控件元素列表中多余相似的对象
                    _, ratio_cc, _ = intersect(rect_o, rect_cc)
                    if ratio_cc > threshold:
                        mods.append(('delete', rect_cc))
            else:
                # 未找到与 OCR 文本匹配的控件元素
                # 需要将该元素新加入控件元素列表
                mods.append(('add', rect_o))
        else:
            mods.append(('add', rect_o))
    for mod in set(mods):
        if mod[0] == 'replace':
            boundings[boundings.index(mod[1])] = mod[2]
        elif mod[0] == 'add':
            boundings.append(mod[1])
        elif mod[0] == 'delete':
            boundings.remove(mod[1])
    return boundings
```

图 4-19: InfoMergeService 的关键方法核心代码

4.3.6 页面布局识别关键代码

```
def layout(bounding, resolution):
    # 处理后的边界 enlarged boundings
    # 分辨率 resolution(即宽高(w,h)))
    for group in groups:
        # 将分组内所有控件的 y 坐标为分割线形成行，并去除高度过小的行
        group[0][0][1] = group[1]
        group[0][-1][2] = group[2]
        nodes = [(y, h) for node_row in group[0] for _, y, _, h in node_row[0]]
        lines = sorted(set([y for y, _ in nodes] + [y + h for y, h in nodes]+ [group[1], group[2]]))
        merge_close_lines(lines, line_merge_threshold * resolution[1] / 100)
        for top, bottom in zip(lines[:-1], lines[1:]):
            # 将与该行有重叠的所有控件的 x 坐标作为分割线形成列
            filtered_basic_rows = [row for row in group[0] if not
                                   (bottom <= row[1] or top >= row[2])]
            nodes = [(x, w) for row in filtered_basic_rows for x, y, w, h
                     in row[0] if not (y + h <= top or y >= bottom)]
            cols = sorted(set([x for x, _ in nodes]+[x + w for x, w in filtered_nodes]))
            if len(cols) == 0 or not cols[0] == 0:
                cols = [0] + cols
            if not cols[-1] == resolution[0]:
                cols.append(resolution[0])
            if len(cols) > 0:
                merge_close_lines(cols, column_merge_threshold * resolution[0] / 100)
                cols[0] = 0
                cols[-1] = resolution[0]
                cols = [[left, right] for left, right in zip(cols[:-1], cols[1:])]
            rows.append([cols, top, bottom])
        group[0] = rows
    return groups
```

图 4-20: LayoutAnalyzeService 的 layout 方法核心代码

图 4-20所示为 LayoutAnalyzeService 水平布局块生成的关键方法核心代码，basic_row_generation 方法主要负责生成初始水平布局块，其将传入的已识别控件元素对象按照垂直方向的坐标范围具备重叠的不同控件组合为一个初始水平块，即若控件 B 的垂直方向中点在控件 A 的顶部坐标与底部坐标之间则两者会融合成一个水平布局块。但笨方法求解出的初始水平布局块与块之间可能会存在 y 方向坐标的重叠，因此还需进一步调用 group_generation 方法进行布

局块再次划分。group_generation 方法会首先将垂直方向中点 y 坐标落在别的初始水平布局块中的布局块与其进行合并。其他垂直方向中点 y 坐标互不落在对方范围内的初始水平布局块相互分离，并且重新设置相应的顶部坐标与底部坐标。然后进行布局块整理，垂直方向的第一个水平布局块的上边界为 0，最后一个水平布局块的下边界为页面底部，同时所有相邻布局块的边界进行重合。对于高度低于阈值的布局块，如果其位于页面顶部则被下方布局块合并；若其位于页面底部则被上方布局块合并；若位于页面中间，则被上下布局块均分。整理完成的水平布局块进一步调用 layout 方法，以布局块内所有控件的 y 坐标为分割线形成行，并去除高度过小的行，同理将与该行有重叠的所有控件的 x 坐标作为分割线形成列并同样去除宽度过小的列。

4.4 测试执行模块的设计与实现

4.4.1 测试执行模块概述

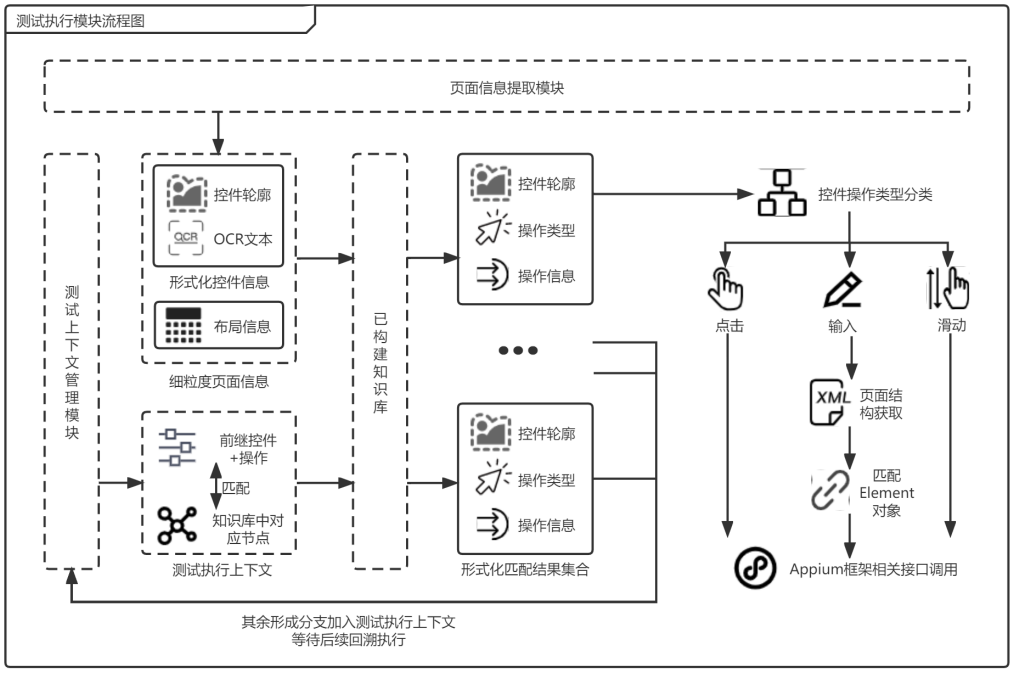

图 4-21: 测试执行模块流程图

测试执行模块主要负责接收页面信息提取模块分析得到的细粒度页面信息以及测试上下文管理模块记录的执行上下文，利用已构建知识图谱进行匹配和

筛选候选控件执行具体操作。测试执行模块流程图如图 4-21所示。页面信息提取模块得到的细粒度页面信息包括页面中的形式化控件信息集合以及页面布局信息。形式化控件信息指控件轮廓与 OCR 提取文本匹配后的封装好的控件元素对象。知识库匹配所需的另一部分测试执行上下文由测试执行上下文管理模块负责维护，主要包括前继控件操作以及与之匹配的知识库中节点信息。知识库构建时主要应用对应功能点的操作执行过程，因此已构建好的知识库中包含该功能点的逻辑信息。知识库匹配后的结果为控件轮廓、操作类型、操作信息的三元组形式化信息集合。测试执行模块默认执行第一个待执行操作，其余操作将形成分支加入测试执行上下文等待后续回溯执行。执行时需要判断控件操作类型，若操作为输入，则还需要拉取页面 XML 布局文件并根据控件轮廓矩形与 XML 中叶节点元素 bounding box 的相似程度从而匹配相应的 Element 对象能够调用相关 Appium 接口。若操作类型为点击或滑动，则可以直接调用。

4.4.2 测试执行模块核心类图

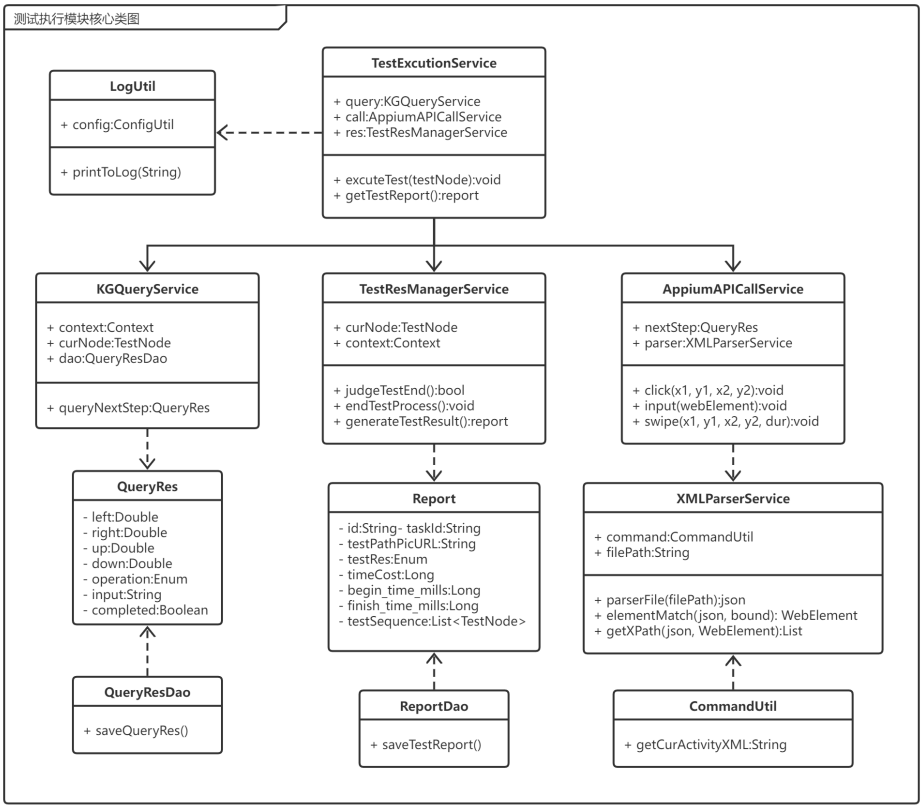

图 4-22: 测试执行模块核心类图

测试执行模块核心类图如图 4-23所示。TestExcutionService 为测试执行模块的核心服务类，其分别通过 KGQueryService 对已构建知识库进行匹配和检索，通过 AppiumAPICallService 调用不同的 Appium 接口执行具体控件操作，通过 TestManagerService 服务判断测试执行过程是否终止以及生成测试结果可视化报告。LogUtil 是系统中用于打印日志的服务类。KGQueryService 主要负责根据获取得到的细粒度页面信息以及测试执行上下文向已经构建好的功能点测试知识库查询与记录的知识匹配的位于当前操作序列上的候选下一页面待操作控件对象集合。知识库返回的匹配结果将会被封装在 QueryRes 类中，并由 QueryResDao 负责持久化保存。AppiumAPICallService 主要负责根据后继操作的不同类型调用不同的 Appium 接口进行具体的控件操作执行，其中点击操作与滑动操作都可以根据图谱匹配的控件边缘信息直接调用相应的接口，而输入操作则必须获取 WebElement 对象。XMLParserService 服务将主要负责拉取当前页面的 XML 结构文件，并根据控件的轮廓信息匹配最为接近的控件节点以获得 WebElement 控件对象并且计算其从页面根节点到匹配的叶节点的 XPath。CommandUtil 类主要负责拉取设备的 XML 布局结构文件。

4.4.3 测试执行模块顺序图

测试执行模块的顺序图如图 4-23所示。在测试执行的过程中，TestExcutionService 会迭代地在每次完成页面信息提取后向 KGQueryService 发起查询图谱请求，经过与构建好的知识库进行匹配获取位于当前执行逻辑上的控件对象与相应操作。获取返回的匹配结果后，TestExcutionService 向 AppiumCallService 发起执行具体操作的请求。AppiumCallService 首先会判断返回的控件操作类型，若当前操作类型为输入，则 AppiumCallService 优先向 CommandUtil 发起获取当前页面 XML 布局结构文件。得到的 XML 文件将由 XMLParserService 进行与文件中的节点的匹配，根据识别的控件轮廓比对节点的 bounding box，两者 IOU 值最大则被认为是最为匹配的节点。由根节点到该节点的访问路径即为该节点的 XPath，该 XPath 即可通过 API 的相应接口定位 WebElement 对象，根据 WebElement 对象的相应输入接口即可实现输入操作。若当前操作类型为点击，则根据识别的控件轮廓直接调用点击屏幕像素点的接口，若当前操作类型为滑动，则根据指定的滑动起始点与终止点坐标进行相应操作。AppiumAPICallService 调用 Appium 接口完成后则会调用 TestResManagerService 生成测试结果，若测试仍未完成则仍需继续迭代。测试完成后，

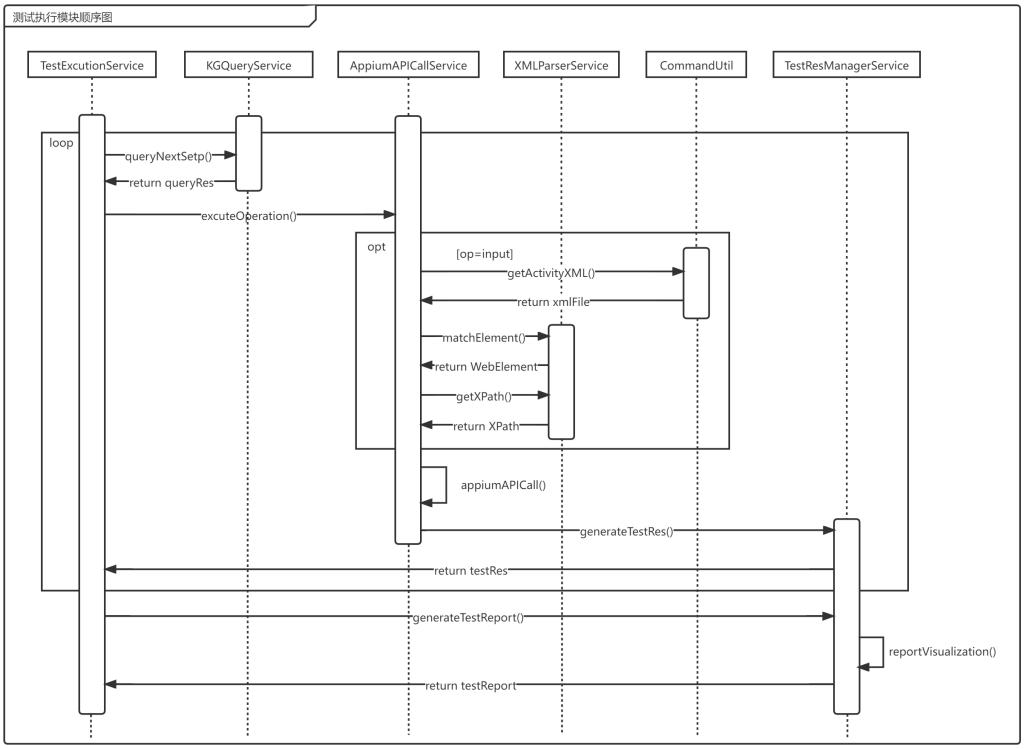

图 4-23: 测试执行模块顺序图

TestExcutionService 能够向 TestResManagerService 发起生成测试结果报告的请求，TestResManagerService 经过数据获取与分析生成可视化报告反馈给前端。

4.4.4 XML 页面结构解析关键代码

图 4-24所示为 XMLParserService 中 XMLNode 类核心代码。XMLNode 类为 XML 文件中所有节点属性集合封装成的类，该类中主要负责维护节点的安卓控件元素类型属性、兄弟节点中同类型元素的数量、从根节点到当前节点的 xpath 路径以及当前节点的 bounding box 位置坐标信息。其中兄弟节点中同类元素的数量主要是为了在设定当前节点 xpath 路径时进行运用。xpath 的设置需要判断当前 index 属性值是否为人工预设的-1，若为-1 说明当前节点为根节点，设置根节点的 xpath 为‘/’。若非根节点，则 xpath 的格式为‘父节点 xpath/当前节点 class[兄弟节点中同类元素的数量]’，根据该格式从父节点扩展 xpath 则可以跟随从根节点遍历 XML 文件的顺序从根节点开始得到每一节点的 xpath。由于 XML 文件中记录的 bounding box 信息是字符串格式，因此仍然需要进行字符串解析以获取当前元素的左、上、右、下四个坐标值。

```
class XMLNode:

    def __init__(self, element, index, father_xpath):
        self.element = element
        # 获取控件 class 属性
        self.class_name = self.element.attrib['class']
        # 父节点下第几个同类元素
        self.index = index
        if self.index == -1:
            # 根节点 xpath 置为'/'
            self.xpath = '/'
        else:
            # 当前节点 xpath 为父节点 xpath+当前节点元素类别+当前元素 index
            self.xpath = "{}/{}[{}]".format(father_xpath, self.class_name, self.index)
        # 解析 bounds 属性值
        bounds_str = self.element.attrib['bounds']
        str_list = re.split(',|\[|\]', bounds_str)
        pos_list = []
        for item in str_list:
            if item != "":
                pos_list.append(int(item))
        self.left = pos_list[0]
        self.up = pos_list[1]
        self.right = pos_list[2]
        self.down = pos_list[3]
```

图 4-24: XMLParserService 的 XMLNode 类核心代码

图 4-25所示为 XMLParserService 的 XMLParser 类核心代码。该类的主要作用是获取并分析应用当前页面的 XML 布局结构，匹配与传入位置最匹配的元素的 xpath 定位方式以便后续通过 Appium 相关接口定位控件元素实现输入操作。get_xml_leaf方法的主要作用是根据传入的 xml 文件路径，分析并返回该文件中所有 xml 叶子节点的 XMLNode 封装类。该方法利用 python 的 ElementTree 包辅助 xml 文件分析，首先调用 ElementTree 的 parse 接口获取 xml 树状结构，接着获取该树的根节点。创建用来遍历树的队列数据结构，并将得到的根节点封装成 XMLNode 节点放入队列中。只要队列不为空就持续进行遍历，获取队列头部的 XMLNode 节点。若当前节点为叶子节点则直接放入结果列表中。若当前节点不为叶子节点，则遍历当前节点的所有子节点，通过字典数据结构判断当前子节点是兄弟节点中的同类元素的第几个，并根据该 index 创建

```
class XMLParser:

    def get_xml_leaf(self, path):
        tree = ET.parse(path)
        # 获得根节点
        root = tree.getroot()
        # 创建 dummyHead
        root.attrib['class'] = ''
        root.attrib['bounds'] = '[0,0][0,0]'
        # 队列中存放初始节点
        stack = [XMLNode(root, -1, '')]
        while len(stack) > 0:
            # 获取队列头部节点
            cur_XMLNode = stack.pop(0)
            if len(cur_XMLNode.element) == 0:
                # 当前节点为叶子节点
                leafs.append(cur_XMLNode)
            else:
                for child in cur_XMLNode.element:
                    child_class_name = child.attrib['class']
                    # 判断当前节点在兄弟节点中是同类元素的第几个
                    stack.append(XMLNode(child, child_name, child_xpath))
        return leafs

    def get_matched_element_xpath(self, apk_name, x1, y1, x2, y2):
        # 调用 command_util 获取页面 XML 结构
        xml_path = get_xml(apk_name)
        leafs = self.get_xml_leaf(path)
        IOU_list = [self.intersect(node.left, node.up, node.right, node.down, x1, y1, x2, y2)
                                    for node in leafs]
        # 返回 IOU 值最大的叶子节点的 xpath 为最匹配 Element 的定位方式
        max_index = IOU_list.index(max(IOU_list))
```

图 4-25: XMLParserService 的 XMLParser 类核心代码

XMLNode 对象存放在队列尾部。get_matched_element_xpath 方法主要用于匹配与传入的 bounding box 信息最接近的 Element 对象以及其 xpath 定位方式。该方法首先调用 command_util 执行 adb 命令并拉取 xml 文件，接着通过刚刚的 get_xml_leaf 方法获取所有布局文件中的所有叶子节点的 XMLNode 封装类对象，最后计算所有叶子节点的 bounding box 与传入的待匹配控件的边缘矩形的 IOU 值，返回最大 IOU 值的记录在 XMLNode 中的 xpath 定位方式。

4.4.5 测试执行流程关键代码

图 4-26所示为测试执行模块活动图。本模块在得到知识图谱匹配完成返回的结果后首先判断当前测试执行节点是否为候选场景终止节点，若当前节点不为候选终止节点需要进一步判断图谱是否返回了有效的执行节点，若结果中并无可执行节点，则说明图谱认为当前节点并未终止但实际分析却未能进一步探索该场景，输出场景不完整的测试结果并继续其他场景的探索。若当前候选场景终止节点为候选终止节点则继续分析下一页面信息并进行图谱匹配，对于匹配的结果需要进行重复性的检查与过滤防止测试执行再次执行已经覆盖过的分支。若过滤后包含有效可执行节点，则说明当前场景并未探索完成，因此继续迭代测试执行。若过滤后没有有效可执行节点则说明当前测试执行场景已经探索完成，检查全局测试执行路径中是否存在仍未覆盖过的场景，若存在当前未遍历节点，则回溯至该节点并切换场景继续测试。若当前不存在未遍历的分支则说明所有场景已经探索完成，输出功能点场景完备的结果。

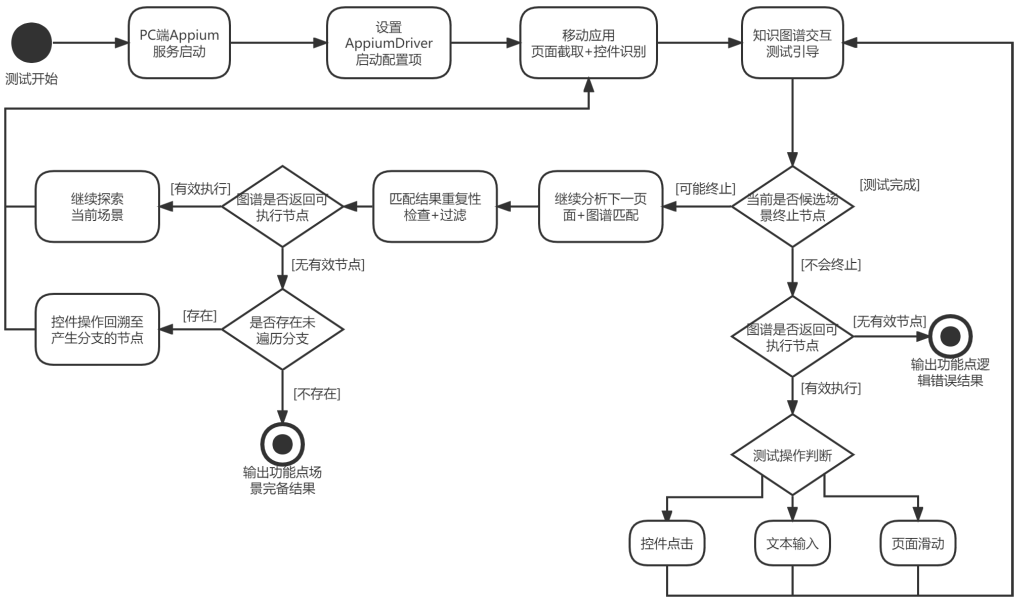

图 4-26: 测试执行活动图

图 4-27所示为 TestExcutionService 的 AutoTestTool 类初始化关键代码。init 方法中主要负责设置待测应用的相关属性（包括包名、应用名、初始化页面名、应用文件路径）、Appium Driver 对象、XMLParser 对象、配置相关属性以及不同控件操作对应方法。click 方法、swipe 方法、input 方法分别为调用相关

Appium 接口实现的控件操作方法并返回本操作执行成功与否状态。

```
class AutoTestTool:

    def __init__(self, app_package, app_activity, app_path):
        self.parser = XMLParser()
        self.test_context = Context()
        self.test_terminated = False
        self.config = Config()
        self.enum = Config_enum()
        self.extract_pic = ExtractPic(self.config, self.enum)
        self.operation = {"输入": self.input, "点击": self.click, "滑动": self.swipe}

    def click(self, x1, y1, x2, y2, content):
        try:
            self.driver.tap([(x1, y1), (x2, y2)])
            return True
        except Exception as e:
            return False

    def swipe(self, x1, y1, x2, y2, content):
        try:
            self.driver.swipe(x1, y1, x2, y2)
            return True
        except Exception as e:
            return False

    def input(self, x1, y1, x2, y2, content):
        xpath = self.parser.get_matched_element_xpath(self.app_name, x1, y1, x2, y2)
        try:
            element = self.driver.find_element_by_xpath(xpath)
            element.send_keys(content)
            return True
        except Exception as e:
            return False
```

图 4-27: TestExcutionService 的 AutoTestTool 类初始化关键代码

图 4-28所示为 TestExcutionService 的 auto_test 方法关键代码。该方法主要涉及测试执行的整体流程。测试终止标志初始化时为 False，若当前测试过程未终止，需要迭代的进行下述步骤。首先调用 command_util 类获取应用当前屏幕截图，接着分析该页面的布局和控件信息并查询图谱匹配下一步待操作的测试

```
def auto_test(self):
    while not self.test_terminated:
        # 获取当前步骤知识图谱匹配结果
        screenshot_path = get_screenshot(self.app_name)
        picture = cv2.imread(screenshot_path, 0)
        # 查询图谱匹配下一步待操作测试执行节点
        split_dic = split(screenshot_path)
        components = self.extract_pic.generate_widget_info(screenshot_path)
        res = next_step(picture, split_dic, components, self.test_context.last_cnt_id)
        # 检查节点是否已经执行过
        res = self.test_context.filter_test_node(res)
        # 解析图谱返回结果，创建 TestNode 测试执行节点对象
        test_node_list = [TestNode(item['category'], item['ocr'], item['operation'],
                                   item['cnt'],item['cnt_id'], item['x1'], item['y1'],
                                   item['x2'], item['y2'], item['state'],
                                   get_cur_activity_name()) for item in res['data']]
        traceback = True
        if len(test_node_list) > 0:
            # 优先执行第一个待测执行节点
            cur_test_node = test_node_list[0]
            res = self.widget_operation()
            # 更新测试执行上下文信息
            self.test_context.add_test_node(test_node_list)
            traceback = not res
        if traceback:
            # 寻找下一待回溯节点
            traceback_sequence = self.test_context.find_traversal_node()
            if traceback_sequence is None:
                self.test_terminated = True
            else:
                for test_node in traceback_sequence:
                    self.widget_operation()
```

图 4-28: TestExcutionService 的 auto_test 方法关键代码

执行节点，图谱匹配的结果需要进行已执行的检查。根据过滤重复节点后的结果创建对应的 TestNode 测试执行节点对象存储相应的测试执行信息。接着进一步判断当前图谱的匹配结果是否返回了有效节点，若结果列表中包含待执行的有效节点则优先执行第一个节点，若该控件操作执行成功，则说明该步骤不需要进行测试场景回溯。否则若控件操作执行失败或无有效节点，均说明当前场景已无法继续探索需要进行回溯，调用 Context 类的 find_traversal_node 方法获

取从根节点到待回溯节点的测试执行序列。若测试执行序列不为空则按顺序依次执行所有控件操作到达未遍历分支的回溯节点。若测试执行序列为空则说明当前功能点已无未遍历分支，所有测试场景已经探索完毕，当前功能点已经测试完成，因此只需更新测试终止标志并结束循环。

4.5 本章小结

本章对系统主要的四个功能模块：测试资源管理模块、测试上下文管理模块、页面信息提取模块、测试执行模块的详细设计与具体实现进行了介绍。首先，本章利用流程图描述了每个模块具体的流程，接着给出了每个模块的核心类图对模块内各个类的属性和行为进行了详细说明，其次利用顺序图说明了模块内各个类如何进行协作与交互，最后给出了模块内关键方法和类的实现代码并结合注释进行了详细解释。

第五章　实验设计与分析

5.1　实验数据收集

为了评估本工具，我们在华为应用商城收集了共 18802 个应用，所收集的应用包括了出行导航、教育学习、拍摄美化、便携生活、运动健康、美食、购物比价、影音娱乐、新闻阅读、商务共 10 个类别以确保实验结果的普遍性。应用分类占比以及应用中所包含的功能点占比情况如表 5-1所示，本文选择的五个功能点分别为登录、注册、搜索购物、机票查询与邮件发送。其中选择登录与注册功能点几乎在任何类别的绝大多数应用中都有包含，平均占比高达 95%。而购物功能点除了在购物比价类别中具有 86% 的占比，在其他各个功能点中都有所涉及，平均占比为 25%，足以说明该功能点在所有应用中的通用性。机票查询与邮件发送功能点的情况一致，分别在除了出行导航类别以及商务类别的其他类别占比极少，但在上述两个类别内又有着相当高的占比，分别为 85% 和 74%，属于在特定应用类别上使用的专用功能。因此，本文所选择的该五个功能点分别对应了在任何应用分类中适用性极强、在各大类别中较为普遍以及在特定类别中占比极高的三大不同特性以确保实验结果的可靠性与典型性。对于该五个功能点，我们总共收集了 96 个应用，其中 57 个应用用于构建各功能点对应的图谱，39 个应用用以评估本工具自动化执行能力与所构建图谱完备性。每个功能点分别以 10 个左右的应用构建功能点图谱再以不同的 10 个左右的应用进行测试。由于选择的功能点在不同应用类别中存在着一定的普遍性，因此部分实验用应用可能会产生重叠，即部分应用将会应用在多个不同的功能点的测试过程中，对于该部分应用本文进行了特别标识。构建图谱的应用详情如表 5-2所示，用于实验的应用详情如表 5-3所示。

表 5-1: 华为应用商城应用分类数据

类别名称	分类占比		功能点占比				
	应用数量	类别比例	登录	注册	搜索购物	机票查询	邮件发送
出行导航	2191	11.6%	96%	96%	12%	85%	0%
教育学习	1964	10.4%	92%	92%	32%	0%	0%
拍摄美化	1217	6.4%	95%	95%	26%	0%	0%
便携生活	2631	13.9%	94%	94%	17%	14%	3%
运动健康	1139	6.0%	93%	93%	8%	1%	0%
美食	1118	5.9%	92%	92%	11%	0%	0%
购物比价	3554	18.9%	100%	100%	86%	0%	0%
影音娱乐	1140	6.0%	93%	93%	29%	0%	0%
新闻阅读	2629	13.9%	97%	97%	22%	0%	0%
商务	1219	7.0%	100%	100%	7%	9%	74%

5.2 实验一：应用功能点测试充分性

5.2.1 实验目的

为了保证本工具在自动化测试的过程中能够对给定功能点进行有效的测试，我们设计了一个实验用来评估该工具在不同应用上对给定功能点测试的充分性，具体来说我们设置了该研究问题：

问题一：本工具在不同应用上对给定功能点的测试过程是否能覆盖用户实际完成该功能的不同方式，测试结果是否充分？

问题一的回答能够表现出本工具在功能点测试上的充分性，证明本自动化测试工具能够较好的将测试粒度限制在给定功能点的逻辑场景中。由于当前工业界与学术界并没有其他测试工具是以功能点为测试粒度的，因此本实验无法选择其他同类工具作为对比。为了区别于这些其他测试工具，本实验并未选择传统自动化测试工具的评估指标，而是定义了功能点中的场景，利用前期人工探索所得的场景集合计算本工具自动化测试执行过程的场景覆盖率，从而评价工具执行与实际用户操作的匹配程度。本实验的核心关注点在于已构建的功能点场景图谱的典型性，基于部分应用的功能点场景数据构建的图谱是否具备判断典型场景执行路径的知识。具体场景定义及评估指标将在下一节详细阐述。

表 5-2: 图谱构建应用数据集

ID	应用名	版本	功能点	ID	应用名	版本	功能点
1	长龙航空	3.5.0	登录 机票查询	2	春秋航空	7.0.24	登录 机票查询
3	东方航空	9.2.9	登录 机票查询	4	海南航空	8.14.1	登录 机票查询
5	吉祥航空	6.7.0	登录 机票查询	6	南方航空	4.2.8	登录 机票查询
7	深圳航空	5.6.5	登录 机票查询	8	祥鹏航空	3.8.1	登录 机票查询
9	中国国航	7.1.2	登录 机票查询	10	中国联合航空	10.8.6	登录 机票查询
11	香港航空	8.3.3	机票查询	12	流利说-英语	8.39.0	登录
13	扇贝单词英语版	4.4.103	登录	14	疯狂背单词	1.21.1	登录
15	乐词新东方背单词	4.3.2	登录	16	拓词	11.32	登录
17	扇贝阅读	4.3.7	登录	18	沪江网校	5.15.7	登录
19	流利说-阅读	2.22.1	登录	20	PTE 单词	1.5.2	登录
21	扇贝听力口语	4.1.101	登录	22	必剪	2.10.0	注册
23	倒数日	1.10.16	注册	24	得到	9.12.0	注册
25	丁香医生	10.1.2	注册	26	豆果美食	7.1.11.2	注册
27	货拉拉	6.6.36	注册	28	IKEA 宜家家居	2.30.0	注册
29	金山词霸	11.2	注册	30	酷我音乐	10.1.0	注册
31	漫画人	3.7.4.0	注册	32	猫眼	9.30.0	注册
33	美丽修行	5.3.1	注册	34	Q 房网	9.7.6	注册
35	人民日报	7.2.4.4	注册	36	瑞幸咖啡	5.0.35	注册
37	扫描全能王	6.11.0	注册	38	央视频	2.4.2	注册
39	医鹿	6.6.32	注册	40	转转	9.6.0	注册
41	孔夫子旧书网	3.15.1	注册 搜索购物	42	当当	12.3.0	搜索购物
43	京东	10.4.5	搜索购物	44	京喜	5.22.0	搜索购物
45	苏宁易购	9.5.68	搜索购物	46	淘宝	10.10.5	搜索购物
47	淘特	5.0.0	搜索购物	48	天猫	12.2.0	搜索购物
49	网易严选	6.12.0	搜索购物	50	一淘	9.11.4	搜索购物
51	139 邮箱	9.3.0	发送邮件	52	189 邮箱	8.3.4	发送邮件
53	2980 邮箱	6.0.3	发送邮件	54	花瓣邮箱	1.0.2	发送邮件
55	简信邮箱	2.4.7	发送邮件	56	沃邮箱	8.4.1	发送邮件
57	邮洽邮箱	4.4.1	发送邮件				

表 5-3: 工具实验应用数据集

ID	应用名	版本	功能点	ID	应用名	版本	功能点
58	小红书	7.27.0	登录	59	天眼查	12.43.3	登录
60	西瓜视频	6.4.6	登录	61	今日头条	8.7.1	登录
62	58 同城	11.0.2	登录	63	赶集直招	10.16.1	登录
			注册				注册
64	抖音	19.6.0	登录	65	知乎	8.6.0	登录
			注册				注册
66	soul	4.19.0	登录	67	得物	4.85.6.1	登录
			注册				注册
68	陌陌	9.2.10	注册	69	智联招聘	8.5.0	注册
70	贝壳找房	2.75.0	注册	71	YY	8.7.1	注册
72	四川航空	6.2.3	机票查询	73	厦门航空	6.6.2	机票查询
74	首都航空	3.7.29	机票查询	75	九元航空	2.2.2	机票查询
76	河北航空	1.7.6	机票查询	77	福州航空	4.4.8	机票查询
78	大连航空	2.0	机票查询	79	天津航空	02.00.14	机票查询
80	义乌购	3.7.4	搜索购物	81	IKEA 宜家家居	2.30.0	搜索购物
82	洋码头	6.8.69	搜索购物	83	小象优品	4.2.9	搜索购物
84	小米有品	5.1.0	搜索购物	85	唯品会	7.67.2	搜索购物
86	唯品仓	1.35.0	搜索购物	87	豌豆公主	6.25.2	搜索购物
88	考拉海购	5.1.1	搜索购物	89	东方购物	4.5.86	搜索购物
90	Gmail	4.3.0	发送邮件	91	Outlook	4.22.5	发送邮件
92	QQ 邮箱	6.3.2	发送邮件	93	搜狐邮箱	2.3.5	发送邮件
94	网易邮箱大师	7.8.14	发送邮件	95	完美邮箱	2.2.2	发送邮件
96	新浪邮箱	1.9.4	发送邮件				

5.2.2 实验设置

目前工业界与学术界的大部分自动化测试工具以应用整体为测试粒度，工具的探索范围并未进行限制，因此这类工具通常使用页面覆盖率与控件覆盖率作为评估测试充分性的指标。在测试前收集应用中的页面与控件集合：

$$A = \{A_1, A_2, A_3, ..., A_n\}, \quad A_i = \{W_{i_1}, W_{i_2}, W_{i_3}, ..., W_{i_n}\}$$

其中 A 代表应用页面集合，A_i 代表页面 A_i 上的控件集合：

$$A_t = \{A_{t_1}, A_{t_2}, A_{t_3}, ..., A_{t_k}\} \quad (t_1, t_2, ..., t_k \in \{1-n\})$$

$$A_{t_j} = \{W_{t\_l_1}, W_{t\_l_2}, W_{t\_l_3}, ..., W_{t\_l_j}\} \quad (l_1, l_2, ..., l_j \in \{1-|A_j|\})$$

测试过程中需要记录所覆盖的页面与控件集合，其中 A_t 代表测试过程中覆盖的页面集合，由离散的 A 中的页面的集合表示，A_{t_j} 代表测试过程中覆盖的页面 A_{t_j} 上已执行到的控件集合。得到应用页面集与控件集、测试页面与控件覆盖集后，页面与控件覆盖率将通过如下方式计算：

$$ActivityCoverage = \frac{|A_t|}{|A|} \times 100\%, \quad WidgetCoverage = \frac{\sum_{j=1}^{t_k} |A_{t_j}|}{\sum_{i=1}^{n} |A_i|} \times 100\%$$

由于传统自动化测试工具的测试粒度为应用整体，因此使用页面与控件覆盖率证明该工具在测试过程中对绝大多数的控件进行了相关操作，并且到达了大部分页面。然而该评估指标仅关注了覆盖的信息的数量，却未能体现执行过程中的逻辑，对于功能点测试而言，更需要关注测试过程中控件操作的先后依赖关系以及最终结果。因此本实验中使用页面与控件覆盖率作为评估指标显然是不合理的，本文为此引入了功能点场景的定义，我们将一组能够实现功能点的控件操作序列成为该功能点的一个场景，具体地来说：

$$Scenario_i =< (w_1, a_1, o_1), (w_2, a_2, o_2), ..., (w_n, a_n, o_n) > \quad (w_i \in W, a_i \in A, o_i \in O)$$

$$A = \{A_1, A_2, A_3, ..., A_n\}, \quad W = \{W_1, W_2, W_3, ..., W_n\}, \quad O = \{O_1, O_2, O_3, ..., O_n\}$$

其中 $Scenario_i$ 为功能点的一个场景由当前页面、待操作控件、所执行的具体操作三元组序列构成，其中每个 w_i 代表三元组中控件属于控件集合 W，每个 a_i 代表三元组中页面属于页面集合 A，每个 o_i 代表三元组中操作属于操作集合 O。应用功能点则视为一系列不同场景的集合，值得注意的是两个场景只要操作序列长度不相等或是序列中相同位置的三元组不相同则被视为两个不同的场景。通过测试前人工探索以及测试过程中收集到的场景数据，本文将以功能点场景覆盖率评估本工具自动化测试过程中对功能点测试的充分性，具体定义：

$$Func = \{Scenario_1, Scenario_2, ..., Scenario_n\}$$

$$S_t = \{Scenario_{t_1}, Scenario_{t_2}, ..., Scenario_{t_n}\}(t_i \in \{1-n\})$$

$$ScenarioCoverage = \frac{|S_t|}{|Func|} \times 100\%$$

其中 $Func$ 为人工探索所得的功能点中的所有场景集合，S_t 为测试执行过程中收集到的已覆盖场景，值得注意的是测试工具需要依次执行该场景中的所有三元组序列才能够视为该场景已被覆盖。$ScenarioCoverage$ 为场景覆盖率，计算方式为测试覆盖场景数量与功能点场景集合元素数的比值。

值得注意的是，本文在设计实验时并未引入其他测试工具作为对比，主要原因为以下三点：

（1）测试目的不同：目前学术界以及工业界的其他 GUI 遍历测试工具的主要测试目的为验证应用中的页面与控件的可达性，是否存在部分页面以及控件无法覆盖，在应用中作为无效的存在。同时通过穷举遍历过程中的各页面以及控件之间的跳转切换顺序以保障应用中各操作执行路径的可行性。而本测试工具是首次提出以应用功能点的可用性作为测试目的，所执行到的所有测试路径均以完成特定功能点为目标。考虑到测试目的并不完全一致，目前学术界与工业界的其他测试工具在严格意义上并不能作为同类工具进行实验对比。

（2）测试粒度不同：由于本工具与其他测试工具的测试目的不同，因此也导致了两者的测试粒度并不相同。传统 GUI 页面遍历测试工具以整体应用作为测试粒度，遍历的过程中需要覆盖应用中的所有页面及控件，然而本测试工具将测试过程收缩到特定功能点内部，以功能点作为粒度。最终实验结果若在两种粒度下进行对比实验将难以确保实验结果的可靠性与客观性。

（3）评估指标不一致：由本节上述内容所述，传统测试工具所采用的页面或控件覆盖率无法有效地评估本测试工具对功能点测试的充分性，因此本文创新性地定义了功能点场景覆盖度作为评估指标。两类工具的评价体系难以统一导致了对比实验无法使用同一指标作为最终结果说明两类工具的测试效果，因此两类测试工具的测试效果的优劣也无法进行有效对比。

综上三点所述，本文的实验将仅论证如何选用有效实验数据以及设计可靠的评价体系以验证本工具的测试有效性，与其他工具的对比将不涵盖在内。

5.2.3 实验结果分析

表 5-4所示为功能点覆盖率实验数据。如图表所示，针对上述选定的五个功能点，每个功能点分别选择了 10 个左右应用收集该应用在该功能点下的典型场景。最终在用于测试登录功能点的总共 10 个应用中收集到了 67 个场景，该

测试工具覆盖了其中的 55 个，达到了 82% 的覆盖率；在用于测试注册功能点的 10 个应用中收集到了 39 个场景，覆盖了其中的 33 个，达到了 84% 的覆盖率；在用于测试机票查询功能点的 8 个应用中收集到了 8 个场景，覆盖了其中的 8 个，达到了 100% 的覆盖率；在用于测试搜索购物功能点的 10 个应用中收集到了 15 个场景，覆盖了其中的 12 个，达到了 80% 的覆盖率；在用于测试发送邮件功能点的 10 个应用中收集到了 24 个场景，覆盖了其中的 20 个，达到了 83% 的覆盖率；所有实验应用总共收集了 153 个场景，覆盖了其中的 128 个，达到了 83% 的覆盖率。

本测试工具目前对于待测功能点中的场景覆盖率较高，说明本工具能够较好地对给定应用的待测功能点进行充分测试。然而，目前测试工具执行情况中仍然存在一些比较常见的场景未覆盖情况，主要由以下两类情况构成：

第一类未覆盖的场景通常为某些特定应用中才会包含的特殊场景。由于相应功能点对应的场景图谱在构建时使用的收集到的用于构建图谱的有限应用中的通用场景下产生的数据，尽管能够保障所构建的图谱中记录的功能点场景逻辑知识具备一定的典型性，却无法涵盖所有应用中的有可能存在的部分特殊场景。如图 5-1、图 5-2、图 5-3所示为应用赶集直招登录功能点的某一个特殊场景，该场景所涉及的部分操作包括首先点击图 5-1中的“不登录，我先看看”，接着点击图 5-2中的“我的”，再点击图 5-3中的“前往登录”。该场景的特殊之处在于已经进入到可以完成登录操作的页面时并不优先通过手机验证码登录或选择第三方登录，而是选择返回首页后续通过“我的”按钮再次返回该页面继续完成登录操作，然而目前业内的大多数应用必须完成登录才可以正常使用软件功能并未提供这样的暂不登录的方式，收集到的用于构建图谱的所有应用的登录功能中并未涉及类似这样的先不登录进入正常功能页面再选择新入口完成登录功能的场景，因此这是导致该场景并未能成功覆盖的原因。

第二类未覆盖的场景为图标类控件入口识别失败问题。图标类控件与文本类控件相比其特点为不存在控件上的提示性文本用于向用户解释该控件在所处页面上的作用，这同样为图谱匹配时判断该图标类控件是否为当前测试上下文环境下的后继执行节点带来了不小的难度。图 5-4所示为小红书应用的登录功能点的第三方登录页面，底部的三个图标类控件分别为 QQ、微博、华为第三方登录方式的图标控件功能入口，由于这三个图标不存在文本提示其作用，测试工具只能通过页面布局识别的方式来辅助理解其控件的意图，这间接导致了 QQ 图标未能成功识别。图 5-5所示为天眼查应用登录功能点的短信验证码

表 5-4: 功能点覆盖率实验数据

登录功能点					
应用名	总场景数	覆盖场景数	应用名	总场景数	覆盖场景数
58 同城	10	10	赶集直招	4	1
小红书	10	9	抖音	6	6
知乎	14	13	soul	3	3
得物	6	4	天眼查	3	2
西瓜视频	3	2	今日头条	7	5
累计场景数	67	累计覆盖场景	55	总覆盖率	82%
注册功能点					
应用名	总场景数	覆盖场景数	应用名	总场景数	覆盖场景数
58 同城	5	5	赶集直招	2	1
陌陌	4	4	抖音	2	2
知乎	10	10	soul	1	1
得物	4	2	智联招聘	3	3
贝壳找房	4	1	YY	4	4
累计场景数	39	累计覆盖场景	33	总覆盖率	84%
机票查询功能点					
应用名	总场景数	覆盖场景数	应用名	总场景数	覆盖场景数
四川航空	1	1	厦门航空	1	1
首都航空	1	1	九元航空	1	1
河北航空	1	1	福州航空	1	1
大连航空	1	1	天津航空	1	1
累计场景数	8	累计覆盖场景	8	总覆盖率	100%
搜索购物功能点					
应用名	总场景数	覆盖场景数	应用名	总场景数	覆盖场景数
义乌购	1	1	IKEA 宜家家居	1	1
洋码头	2	2	小象优品	4	2
小米有品	1	1	唯品会	1	0
唯品仓	1	1	豌豆公主	1	1
考拉海购	2	2	东方购物	1	1
累计场景数	15	累计覆盖场景	12	总覆盖率	80%
发送邮件功能点					
应用名	总场景数	覆盖场景数	应用名	总场景数	覆盖场景数
Gmail	3	2	Outlook	2	2
QQ 邮箱	4	3	搜狐邮箱	5	4
网易邮箱大师	2	2	完美邮箱	5	4
新浪邮箱	3	3			
累计场景数	24	累计覆盖场景	20	总覆盖率	83%

图 5-1: 非典型步骤 1

图 5-2: 非典型步骤 2

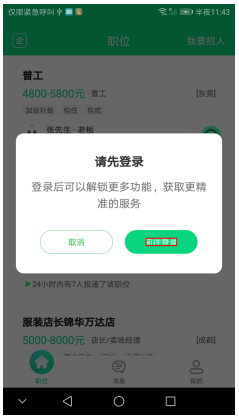

图 5-3: 非典型步骤 3

图 5-4: 未识别图标 1

图 5-5: 未识别图标 2

图 5-6: 未识别图标 3

登录页面，该页面底部存在一个华为图标用于指示华为第三方登录应用的功能入口。由于此图标只会在华为手机上出现，因此在构建登录功能点图谱时用的是通用场景，并未能包括点击华为图标实现登录的场景因此未能覆盖该场景。图 5-6所示为今日头条应用登录页面，页面底部分别有一个手机样式的图标以及一个三点样式图标，然而两个图标下方的文本，即“隐私设置”、“遇到问题”却并不能显示两个图标的功能。点击手机样式图标后会进入手机号密码登录页面，提示输入手机号与密码等信息。点击三点图标后会进入第三方登录方式选择页面，用户可以选择如 QQ、微博、微信等登录方式。因此两个图标真正的功能应该分别为：手机号密码登录、第三方登录，底部的文本不但无法揭示其真正作用甚至误导了匹配过程，使得工具未能成功识别该两个图标。

5.3 实验二：自动化测试执行成功率

5.3.1 实验目的

为了保证本工具在自动化测试过程中能够有效执行图谱匹配的操作，我们设计了一个实验用来评估该工具在不同应用的给定功能点的测试过程中的自动化程度，具体来说我们设置了该研究问题：

问题二：本工具在不同应用上对给定功能点的测试过程执行相应图谱匹配的控件操作的成功率如何，测试过程是否足够自动化？

问题二的回答通过测试工具执行的成功率证明本工具能够以较高的自动化程度执行所匹配出的控件操作。本实验沿用了 5.2.2 中关于场景的定义，利用场景三元组序列中的操作定义，根据实验前期人工探索得到的场景操作步骤集合，以及实验过程中成功执行的操作集合计算自动化测试执行成功率。本实验的核心关注点在于自动化测试工具的自动化程度，在图谱已经匹配出了正确的待操作控件信息后，测试工具能否通过自动化测试框架模拟用户各类操作推进测试过程。具体实验评估指标将在下一节中详细阐述。

5.3.2 实验设置

本节将沿用 5.2.2 中关于场景的定义，阐述自动化测试执行成功率这一评估指标。$Func$ 为人工探索所得的功能点中所有场景集合，同时场景 $Scenario_i$ 为 < 页面，操作，控件 > 的三元组序列，则当前应用中的该功能点所有操作集合即为所有场景三元组的并集，具体定义如下：

$$Scenario_i = \{triple_1, triple_2, ..., triple_n\}, \quad triple_i =< w_i, a_i, o_i >$$

$$ScenarioOp_i = \{triple_{i_1}, triple_{i_2}, ..., triple_{i_k}\}$$

$$(triple_{i_k} \in Scenario_i \quad and \quad \forall m, n \in \{i_1 - i_k\} \quad triple_m \neq triple_n)$$

$$FuncOp = \{ScenarioOp_1 \cup ... ScenarioOp_i \cup ... ScenarioOp_n\} \quad (ScenarioOp_i \in Func)$$

$Scenario$ 为沿用的场景定义，$tripe_i$ 则是为三元组设置的别称，$ScenarioOp$ 为场景内部对三元组进行去重后，将原三元组序列转为三元组集合，避免场景内部有多次执行的相同重复操作。值得一提的是，两个三元组相等的含义为三元

组的每一元素都分别相同，实际含义为同一页面上对同一控件进行的相同具体类型的操作。$FuncOp$ 为该功能点的所有操作集合，除了 $ScenarioOp$ 对场景内部操作进行去重的操作，$FuncOp$ 对各场景的 $ScenarioOp$ 取并集进行场景间操作，避免多个场景中有公共的部分序列引起重复计算某些操作。接着对测试执行过程中所收集到的成功执行的操作同样进行上面的去重操作，再次不再进行重复定义，最终获得成功执行操作集合 $TestOp$，则自动化测试执行成功率的定义为 $TestOp$ 中所包含的元素数量与 $FuncOp$ 中所包含的元素数量的比值。

$$ExecuteSuccess = \frac{|TestOp|}{|FuncOp|} \times 100\%$$

5.3.3 实验结果分析

表 5-5所示为自动化测试执行成功率实验数据。如表所示，最终在登录功能的测试过程中涉及了 192 个不同的操作，测试工具成功执行了其中的 163 个，测试执行成功率为 85%；在注册功能点的测试过程中涉及了 161 个不同的操作，测试工具成功执行了其中的 137 个，测试执行成功率为 85%；在机票查询功能的测试过程中涉及了 39 个不同的操作，测试工具成功执行了其中的 36 个，测试执行成功率为 92%；在搜索购物功能的测试过程中设计了 85 个不同的操作，测试工具成功执行了其中的 78 个，测试执行成功率为 91%；在发送邮件功能的测试过程中涉及了 69 个不同的操作，测试工具成功执行了其中的 61 个，测试执行成功率为 88%。所有试验应用的测试过程中总共需要执行 546 个操作，成功执行了其中的 475 个，测试执行成功率为 87%。

本测试工具目前自动化测试执行的成功率较高，说明本工具能够较高程度地实现测试过程的全自动化并有效推进测试过程。然而，目前测试工具执行过程中仍然存在一些无法成功执行的测试操作，主要分为以下几类情况：

第一类为设备权限弹窗，如图 5-7所示为抖音应用请求设备电话权限，这类弹窗通常出现在应用初次启动时请求设备允许应用获取某类权限，由于该类弹窗会遮挡背后的应用原生控件导致正常的该应用页面无法识别得到正确的待操作控件。同时设备原生弹窗又无法通过自动化测试框架调取相应接口进行关闭，因此目前此类权限弹窗只能由工具请求人工关闭。

第二类为应用广告弹窗，如图 5-8所示为 58 同城应用首页截图，截图中央显示的为应用广告弹窗，关闭按钮位于弹窗下方且是图标类按钮。此类弹窗同第一类弹窗一样，会遮挡背后的其他控件。并且由于弹窗模态框的存在，虽然

表 5-5: 自动化测试执行成功率实验数据

登录功能点					
应用名	总操作数	成功操作数	应用名	总操作数	成功操作数
58 同城	15	12	赶集直招	14	12
小红书	21	17	抖音	30	26
知乎	30	26	soul	29	25
得物	23	20	天眼查	13	11
西瓜视频	13	11	今日头条	22	19
累计操作数	192	累计成功操作	163	总成功率	85%
注册功能点					
应用名	总操作数	成功操作数	应用名	总操作数	成功操作数
58 同城	17	15	赶集直招	12	10
陌陌	11	10	抖音	16	13
知乎	22	19	soul	8	7
得物	30	25	智联招聘	11	9
贝壳找房	19	16	YY	15	13
累计操作数	161	累计成功操作	137	总成功率	85%
机票查询功能点					
应用名	总操作数	成功操作数	应用名	总操作数	成功操作数
四川航空	5	5	厦门航空	5	4
首都航空	5	4	九元航空	5	5
河北航空	5	5	福州航空	5	4
大连航空	4	4	天津航空	5	5
累计操作数	39	累计成功操作	36	总成功率	92%
搜索购物功能点					
应用名	总操作数	成功操作数	应用名	总操作数	成功操作数
义乌购	6	6	IKEA 宜家家居	6	6
洋码头	12	11	小象优品	18	16
小米有品	8	7	唯品会	5	5
唯品仓	7	6	豌豆公主	6	6
考拉海购	10	8	东方购物	7	7
累计操作数	85	累计成功操作	78	总成功率	91%
发送邮件功能点					
应用名	总操作数	成功操作数	应用名	总操作数	成功操作数
Gmail	8	7	Outlook	7	7
QQ 邮箱	12	10	搜狐邮箱	11	9
网易邮箱大师	9	8	完美邮箱	10	9
新浪邮箱	12	11			
累计操作数	69	累计成功操作	61	总成功率	88%

图 5-7: 设备权限弹窗

图 5-8: 应用广告弹窗

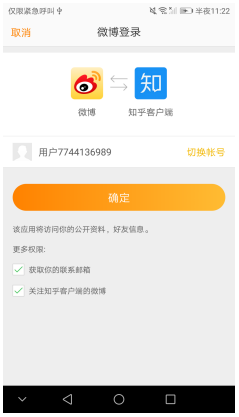

图 5-9: 跳转至外部应用

图 5-10: 验证码输入

图谱匹配出了需要待操作的控件，但当自动化测试工具按照要求执行时在不关闭弹窗的情况下是不会使页面产生变化的，这会导致自动化测试执行过程陷入循环，即在页面状态不发生改变的情况下始终返回相同的待操作控件使得测试过程难以继续。即便测试过程中添加了相应页面状态变化的检测，避免测试工具陷入循环，但仍然需要继续关闭该类弹窗使得测试过程推进。与第一类弹窗相比，应用广告弹窗出现更随机的特性使得此类弹窗的处理更加困难。

第三类为外部应用跳转，如图 5-9所示为知乎应用跳转至微博应用并请求第三方应用授权。跳转至外部应用时待测应用页面会被挂起，新应用内部逻辑更加不可控。跳转至新应用的原因通常为请求第三方应用授权，点击了跳转至第三方应用的链接或者某些网页链接。第三方应用与当前待测应用的功能点完成间并没有过多联系。因此当跳转至第三方应用时应当立即通过导航栏或者应用原生返回按键返回至原应用。

第四类为验证码输入，如图 5-10所示为今日头条应用发送验证码后请求用户输入验证码。验证码输入需要读取手机短信，而这一过程由于其身份验证的

设计原则天然无法进行自动化实现，因此在应用成功发送完验证码后测试工具会请求测试人员人工查验短信内容，获取验证码并进行手动输入。

5.4 本章小结

本章主要进行了两个实验分别评估本工具对待测功能点的测试充分性以及本工具测试执行的自动化程度。在实验一中，本文定义了功能点中的场景，以测试前人工探索的场景与测试过程中覆盖的已执行场景计算测试中的功能点场景覆盖率。对于实验结果的分析，我们发现本工具构建的知识图谱具备较为典型的功能点场景知识，能够得到充分的功能点测试结果。在实验二中，本文计算了功能点场景中的所有操作的执行成功率，通过结果分析，得到了该测试工具能够很好地自动化执行图谱匹配结果操作的结论，同时对于未能成功执行的典型失效场景进行了原因分析。

第六章　总结与展望

6.1　总结

随着移动应用的发展，大众用户在日常生活的方方面面能通过应用各类具体功能借用数字化方式完成。不断缩短的应用开发迭代周期给移动应用质量保障提出了更高的要求，因此各类移动应用自动化测试技术被不断提出与改进。目前学术界与工业界投入广泛应用的自动化测试方法主要分为自动化测试框架下的脚本测试以及 GUI 自动化遍历测试，然而这两类测试方法都有其局限性。GUI 自动化遍历测试基于特定遍历策略，例如深度优先遍历等对应用中的页面与控件进行探索，通过高页面与控件覆盖率确保应用中的控件可操作性与页面可达性，然而该测试方法仅关注了所探索的控件的数量而忽视了控件与控件、页面与控件之间的操作时序性以及上下文依赖性，即未能在探索过程中体现功能逻辑性。基于自动化测试框架的脚本测试则克服了这一缺陷，所编写的测试脚本通常为模拟用户实际使用应用完成某特定任务，如登录、购物等过程，测试脚本中编写的语句前后构成了完成功能的某路径的具体操作序列，然而该测试方法却受制于应用功能、页面的快速变化，极易失效的测试脚本无法跨应用的多个版本使用，这大大提高了测试过程中反复编写脚本的人工成本。

本文创新性地提出了基于知识抽取的功能点测试跨应用复用技术，在改进了 GUI 自动化遍历测试缺乏功能逻辑性以及自动化测试框架脚本测试易失效的缺陷的同时，通过知识图谱的测试引导加强了测试过程的功能逻辑性并将测试粒度限制在功能点层面，减少了传统自动化测试过程中遍历应用中所有页面与控件而产生的复杂分支。知识图谱的构建基于功能点，在前期收集不同应用中特定功能点的操作逻辑形成场景数据，构建完成的知识图谱能够基于功能点场景知识与已执行测试上下文自动匹配候选后继操作，扩展探索分支；自动化测试执行部分则区别于脚本测试中基于页面布局结构文件定位控件元素这样容易强依赖与页面结构的易失效方式，引入了图像理解技术识别页面中的控件元素并分析页面布局，从而可以尽可能地减少版本更新对应用自动化测试的影响。

本文首先介绍了项目的背景与意义，包括安卓自动化测试技术与知识图谱

技术的国内外研究现状，通过分析了现有安卓自动化测试技术的局限性以及知识图谱应用于功能点测试的可行性之后，提出了基于知识抽取的功能点测试跨应用复用技术。接着对于文本使用的安卓自动化测试框架与相关技术、知识图谱相关概念以及图数据库存储技术、图像理解相关算法等进行了详细的阐述并解释了技术选型的理由。接着，对本系统进行了需求分析，涉众分析和通过用例图以及用例表进行了用例分析，并通过系统架构图和 4+1 视图等对系统进行了架构设计以及模块划分，通过系统实体类分析以及 ER 图对数据库进行了相关设计。其次，对本系统所划分的测试资源管理模块、测试上下文管理模块、页面信息提取模块、测试执行模块分别通过模块流程图阐述模块运行流程，通过模块核心类图阐述模块详细功能划分，通过时序图阐述模块类间交互过程，并附上关键实现代码。最后在实证研究部分定义了功能点中的场景，并以场景覆盖率为评估指标验证了功能点知识图谱的充分性，通过自动化测试执行成功率验证了自动化测试工具的可靠性。

6.2 展望

目前经过实证研究证明了基于知识抽取的功能点测试跨应用复用工具能够对给定功能点进行较为充分的测试，高测试执行成功率也说明了本自动化测试工具的可靠性，然而受时间和能力所限，目前仍然存在几个方面待改进：

（1）所支持的待测功能点数量目前较少。由于知识图谱的构建是以待测功能点为粒度，因此每新增一个所支持待测功能点就需要新构建一个对应的知识图谱，考虑到图谱构建前期收集功能点场景数据以及图谱构建完成的维护和更新成本，目前本工具仅针对所选择的五个功能点构建了对应的知识图谱，即登录、注册、搜索购物、机票查询、发送邮件，这五个功能点在各类应用中具备一定的普遍性，然而目前市场上仍然存在大量应用以及未能支持的功能点需要进行可靠性测试。如何更好地定义与划分功能点的粒度并形成更加高效的图谱构建方式以支持更多功能的测试是本工具未来的主要努力方向。

（2）功能点场景覆盖率仍然需要提高。目前针对已支持的待测功能点在实验过程中发现仍然存在着一些较为特殊的场景未能覆盖，这类特殊场景或是应用流程的特殊设计，或是测试设备的独特特性，因为构建图谱前收集到的场景数据的局限性无法进行很好的覆盖。未来仍然需要继续扩大数据集收集范围或是更换更加丰富的数据源以确保所构建图谱中存储知识的丰富性，并且需要

思考引入有效数据扩增的方法来应对原始功能点点场景数据不够丰富的情况。

（3）自动化测试执行率仍然需要提高。目前针对本工具自动化测试执行成功率的实验过程中发现仍然存在着一些典型情形导致测试过程需要暂停以获取人工帮助，除了验证码输入这类流程设计时以身份认证为目的的特殊情形，其他例如各类应用弹窗或设备权限请求弹窗等在测试过程中也反复出现。后期仍然需要设计自动化弹窗识别与关闭机制以确保本工具的更高自动化的执行过程，确保测试效率。

致　　谢

在南京大学的两年研究生生涯终于告一段落了，这也已经是在南京大学的第六个年头了。从本科毕业那一年开始，新冠肺炎开始在全世界肆虐，这一时代特殊性也给我的研究生生涯多少造成了些不便，同样也赋予了这一段时光一些特别的含义。回首两年仙林以及四年鼓楼的学习生活，在这段旅程中遇到了许多对我给予鼓励和指导的人，因此想在这里感谢每一位帮助过我的人。

首先要感谢我的研究生导师陈振宇老师，在本科阶段陈老师教学期间所展示出的幽默风趣以及渊博的学识就令我留下了深刻的印象，这也是我在保研时选择了 iSE 实验室继续研究生阶段学习的原因。研究生期间，陈老师利用开阔的思路深入浅出地为我们答疑解惑，并不断鞭策和鼓励我们专注于学业，为实验室和所研究的领域贡献有价值的产出。同时陈老师的刻苦努力和勤勉不懈以身作则激励了我们，是我们永远学习的榜样。

其次我想感谢实验室的房春荣老师，房老师指导了我的本科与研究生毕设以及平时研究生期间的工作。令我印象最深刻的是在我完成本科的毕设后，房老师鼓励我以毕设项目投稿 ASE Demo，作为科研小白的我对学术论文写作以及实验等一无所知，是房老师的悉心指导帮助我完成了这篇论文，最后我也幸运地在本科阶段有了一篇一作身份的顶会 Demo 论文。如果不是房老师的鼓励和指导，我的履历上也不会有这么浓墨重彩的一笔，在此对房老师在整个研究生阶段对我的所有帮助表示最真挚的感谢。

同时我想感谢整个 iSE 大家庭中的所有同学。这里有给我提供过很多帮助和技术指导的学长，田元汗、李灏宇学长帮助我接过了移动应用自动化测试项目，张晨剑、尹子越、周赛学长帮助我掌握了 Web 自动化测试，学长们的帮助和指导让我快速融入了实验室。这里有与我一起协作完成项目的同届的同学，同一小组的王旭、张晶、恽叶霄、李彤宇同学提供了很多技术上的帮助，他们的一部分产出对我完成毕设有非常大的帮助，虞圣呈博士为我点拨了许多项目和实验的方向。在这些人身上我学到了努力、积极乐观、不畏困难等一系列的优秀品质，与他们相识是我在实验室中收获的最大的财富。

参考文献

[1] 郑三波. 10 月国内手机出货量 3357.5 万部 5G 手机占八成 [J]. 重庆商报, 2021, 04.

[2] 工信部运行监测协调局. 2021 年 1 到 11 月规模以上互联网企业业务收入同比增长 22.3%[J]. 中国电子报, 2012, 06.

[3] 陈永伟. 美国众议院《数字市场竞争状况调查报告》介评 [J]. 竞争政策研究, 2020, 05.

[4] BAEK Y-M, BAE D-H. Automated model-based Android GUI testing using multi-level GUI comparison criteria[C] // 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). 2016 : 238 – 249.

[5] SHAHà G, SHAHà P, MUCHHALAà R. Software Testing Automation using Appium[J]. International Journal of Current Engineering and Technology, 2014, 5 : 3528 – 3531.

[6] 刘升贵, 史梦安. 基于 UIAutomator 的 Android UI 自动化测试框架及其应用探索 [J]. 福建电脑, 2017, 33(06).

[7] 陈镔, 姬庆, 夏夜. 移动互联自动化测试框架创新研究与实践 [J]. 金融电子化, 2016, 5 : 69 – 70.

[8] MIRZAEI N, GARCIA J, BAGHERI H, et al. Reducing combinatorics in GUI testing of android applications[C/OL] // DILLON L K, VISSER W, WILLIAMS L A. Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016. [S.l.] : ACM, 2016 : 559 – 570. https://doi.org/10.1145/2884781.2884853.

[9] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. Using GUI ripping for automated testing of Android applications[C/OL] // GOEDICKE M,

MENZIES T, SAEKI M. IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012. [S.l.]: ACM, 2012: 258 – 261.
https://doi.org/10.1145/2351676.2351717.

[10] MACHIRY A, TAHILIANI R, NAIK M. Dynodroid: an input generation system for Android apps[C/OL] // MEYER B, BARESI L, MEZINI M. Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013. [S.l.]: ACM, 2013: 224 – 234.
https://doi.org/10.1145/2491411.2491450.

[11] JIANG C, LIU B, YIN Y, et al. Study on real-time test script in automated test equipment[C/OL] // 2009 8th International Conference on Reliability, Maintainability and Safety. 2009: 738 – 742.
http://dx.doi.org/10.1109/ICRMS.2009.5270090.

[12] GAO Z, CHEN Z, ZOU Y, et al. SITAR: GUI Test Script Repair[J/OL]. IEEE Transactions on Software Engineering, 2016, 42(2): 170 – 186.
http://dx.doi.org/10.1109/TSE.2015.2454510.

[13] HU Y, NEAMTIU I, ALAVI A. Automatically verifying and reproducing event-based races in Android apps[C/OL] // ZELLER A, ROYCHOUDHURY A. Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016. [S.l.]: ACM, 2016: 377 – 388.
https://doi.org/10.1145/2931037.2931069.

[14] CHEN N, KIM S. Puzzle-based automatic testing: bringing humans into the loop by solving puzzles[C/OL] // GOEDICKE M, MENZIES T, SAEKI M. IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012. [S.l.]: ACM, 2012: 140 – 149.
https://doi.org/10.1145/2351676.2351697.

[15] BAEK Y-M, BAE D-H. Automated model-based Android GUI testing using multi-level GUI comparison criteria[C] // 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). 2016 : 238 – 249.

[16] USMAN M, IQBAL M Z, KHAN M U. An automated model-based approach for unit-level performance test generation of mobile applications[J/OL]. J. Softw. Evol. Process., 2020, 32(1).
https://doi.org/10.1002/smr.2215.

[17] ZADGAONKAR H. Robotium automated testing for Android efficiently automate test cases for Android applications using Robotium[J], 2013.

[18] PAYDAR S, HOUSHMAND M, HAYERI E. Experimental study on the importance and effectiveness of monkey testing for android applications[C/OL] // 2017 International Symposium on Computer Science and Software Engineering Conference (CSSE). 2017 : 73 – 79.
http://dx.doi.org/10.1109/CSICSSE.2017.8364659.

[19] CHOUDHARY S R, GORLA A, ORSO A. Automated Test Input Generation for Android: Are We There Yet? (E)[C/OL] // COHEN M B, GRUNSKE L, WHALEN M. 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015. [S.l.] : IEEE Computer Society, 2015 : 429 – 440.
https://doi.org/10.1109/ASE.2015.89.

[20] AZIM T, NEAMTIU I. Targeted and depth-first exploration for systematic testing of android apps[C/OL] // HOSKING A L, EUGSTER P T, LOPES C V. Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013. [S.l.] : ACM, 2013 : 641 – 660.
https://doi.org/10.1145/2509136.2509549.

[21] YANG W, PRASAD M R, XIE T. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications[C/OL] // CORTELLESSA V, VARRÓ D. Lecture Notes in Computer Science, Vol 7793 : Fundamental Approaches to

Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. [S.l.] : Springer, 2013 : 250 – 265.
https://doi.org/10.1007/978-3-642-37057-1_19.

[22] LI X, JIANG Y, LIU Y, et al. User Guided Automation for Testing Mobile Apps[C/OL] // CHA S S, GUÉHÉNEUC Y, KWON G. 21st Asia-Pacific Software Engineering Conference, APSEC 2014, Jeju, South Korea, December 1-4, 2014. Volume 1: Research Papers. [S.l.] : IEEE Computer Society, 2014 : 27 – 34.
https://doi.org/10.1109/APSEC.2014.13.

[23] PUJARA J, MIAO H, GETOOR L, et al. Knowledge Graph Identification[J]. Springer-Verlag New York, Inc., 2013.

[24] XU Z, SHENG Y, HE L, et al. Review on Knowledge Graph Techniques[J]. Dianzi Keji Daxue Xuebao/Journal of the University of Electronic Science and Technology of China, 2016.

[25] BOLLACKER K D, COOK R P, TUFTS P. Freebase: A Shared Database of Structured General Human Knowledge[C/OL] // Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada. [S.l.] : AAAI Press, 2007 : 1962 – 1963.
http://www.aaai.org/Library/AAAI/2007/aaai07-355.php.

[26] WEBBER J. A programmatic introduction to Neo4j[C/OL] // LEAVENS G T. Conference on Systems, Programming, and Applications: Software for Humanity, SPLASH '12, Tucson, AZ, USA, October 21-25, 2012. [S.l.] : ACM, 2012 : 217 – 218.
https://doi.org/10.1145/2384716.2384777.

[27] DAS A, MITRA A, BHAGAT S N, et al. Issues and Concepts of Graph Database and a Comparative Analysis on list of Graph Database tools[C/OL] // 2020 International Conference on Computer Communication and Informatics (ICCCI). 2020 : 1 – 6.
http://dx.doi.org/10.1109/ICCCI48352.2020.9104202.

[28] ANIKIN D, BORISENKO O, NEDUMOV Y. Labeled Property Graphs: SQL or NoSQL?[C/OL] // 2019 Ivannikov Memorial Workshop (IVMEM). 2019 : 7 – 13. http://dx.doi.org/10.1109/IVMEM.2019.00007.

[29] KANYA N, RAVI T. Modelings and techniques in Named Entity Recognition-an Information Extraction task[C/OL] // IET Chennai 3rd International on Sustainable Energy and Intelligent Systems (SEISCON 2012). 2012 : 1 – 5. http://dx.doi.org/10.1049/cp.2012.2199.

[30] BORDES A, USUNIER N, GARCIA-DURAN A, et al. Translating Embeddings for Modeling Multi-relational Data[J]. Curran Associates Inc., 2013.

[31] WANG Z, ZHANG J, FENG J, et al. Knowledge Graph Embedding by Translating on Hyperplanes[C/OL] // BRODLEY C E, STONE P. Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. [S.l.] : AAAI Press, 2014 : 1112 – 1119. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

[32] LIN Y, LIU Z, SUN M, et al. Learning Entity and Relation Embeddings for Knowledge Graph Completion[C/OL] // BONET B, KOENIG S. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. [S.l.] : AAAI Press, 2015 : 2181 – 2187. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571.

[33] AMATYA S, KURTI A. Cross-Platform Mobile Development: Challenges and Opportunities[C/OL] // TRAJKOVIK V, ANASTAS M. Advances in Intelligent Systems and Computing, Vol 231 : ICT Innovations 2013 - ICT Innovations and Education, Ohrid, Macedonia, 12-15 September, 2013. [S.l.] : Springer, 2013 : 219 – 229. https://doi.org/10.1007/978-3-319-01466-1_21.

[34] CHOWDHURY R R, HOSSAIN S S, ARAFAT Y, et al. Configuring Appium for iOS Applications and Test Automation in Multiple Devices[C/OL] // ASSE 2020: Asia Service Sciences and Software Engineering Conference, Nagoya, Japan, May 13-15, 2020. [S.l.] : ACM, 2020 : 63 – 69. https://doi.org/10.1145/3399871.3399883.

[35] HWANG S, LEE S, KIM Y, et al. Bittersweet ADB: Attacks and Defenses[C/OL] // BAO F, MILLER S, ZHOU J, et al. Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015. [S.l.] : ACM, 2015 : 579 – 584. https://doi.org/10.1145/2714576.2714638.

[36] AMARANTE J, BARROS J P. Exploring USB Connection Vulnerabilities on Android Devices - Breaches using the Android Debug Bridge[C] // 14th International Conference on Security and Cryptography. 2017.

[37] ROSPOCHER M, van ERP M, VOSSEN P, et al. Building event-centric knowledge graphs from news[J/OL]. J. Web Semant., 2016, 37-38 : 132 – 151. https://doi.org/10.1016/j.websem.2015.12.004.

[38] GOTTSCHALK S, DEMIDOVA E. EventKG: A Multilingual Event-Centric Temporal Knowledge Graph[C/OL] // GANGEMI A, NAVIGLI R, VIDAL M, et al. Lecture Notes in Computer Science, Vol 10843 : The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings. [S.l.] : Springer, 2018 : 272 – 287. https://doi.org/10.1007/978-3-319-93417-4_18.

[39] GUAN S, CHENG X, BAI L, et al. What is Event Knowledge Graph: A Survey[J/OL]. CoRR, 2021, abs/2112.15280. https://arxiv.org/abs/2112.15280.

[40] LI Z, ZHAO S, DING X, et al. EEG: Knowledge Base for Event Evolutionary Principles and Patterns[C/OL] // CHENG X, MA W, LIU H, et al. Communications in Computer and Information Science, Vol 774 : Social Media Processing - 6th National Conference, SMP 2017, Beijing, China, September 14-17, 2017, Proceedings. [S.l.] : Springer, 2017 : 40 – 52. https://doi.org/10.1007/978-981-10-6805-8_4.

[41] COLAS A, SADEGHIAN A, WANG Y, et al. EventNarrative: A large-scale Event-centric Dataset for Knowledge Graph-to-Text Generation[J/OL]. CoRR, 2021, abs/2111.00276. https://arxiv.org/abs/2111.00276.

[42] CANNY J F. A Computational Approach to Edge Detection[J/OL]. IEEE Trans. Pattern Anal. Mach. Intell., 1986, 8(6): 679–698. https://doi.org/10.1109/TPAMI.1986.4767851.

[43] LOWE D G. Distinctive Image Features from Scale-Invariant Keypoints[J/OL]. Int. J. Comput. Vis., 2004, 60(2): 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94.

简历与科研成果

基本信息

曹振飞，男，汉族，1998 年 7 月 4 日出生，江苏省无锡市江阴市人。

教育背景

2020 年 9 月 — 2022 年 6 月 南京大学软件学院 硕士

20016 年 9 月 — 2020 年 9 月 南京大学软件学院 本科

攻读工程硕士学位期间完成的学术成果

1. Shengcheng Yu, Chunrong Fang, **Zhenfei Cao**, Xu Wang, Tongyu Li and Zhenyu Chen, ”Prioritize Crowdsourced Test Reports via Deep Screenshot Understanding,” 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 946-956.
2. **Zhenfei Cao**, Xu Wang, Shengcheng Yu, Yexiao Yun and Chunrong Fang, ”STIFA: Crowdsourced Mobile Testing Report Selection Based on Text and Image Fusion Analysis,” 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020, pp. 1331-1335.

攻读工程硕士学位期间参与的科研课题

1. 房春荣、王旭、**曹振飞**、虞圣呈、李彤宇、陈振宇，“一种基于图像文本融合分析的移动应用众包测试报告排序的方法”，专利申请号：202111471921.2，2021.
2. 房春荣、虞圣呈、恽叶霄、王旭、**曹振飞**、李彤宇，“一种基于图像理解技术的移动应用众包测试缺陷报告自动生成的方法”，专利申请号：2020104872052，2020.
3. 房春荣、**曹振飞**、王旭、虞圣呈、恽叶霄、李彤宇，“一种基于自然语言处理的众包测试报告相似度检测的方法”，专利申请号：2020104872029，2020.