计算机科学
COMPUTER SCIENCE

基于依赖模型的REST接口测试用例生成方法研究

刘盈盈, 杨秋辉, 姚邦国, 刘巧韵

引用本文

刘盈盈, 杨秋辉, 姚邦国, 刘巧韵. 基于依赖模型的REST接口测试用例生成方法研究[J]. 计算机科学, 2023, 50(9): 101-107.

LIU Yingying, YANG Qiuhui, YAO Bangguo, LIU Qiaoyun. Study on REST API Test Case Generation Method Based on Dependency Model [J]. Computer Science, 2023, 50(9): 101-107.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800071

基于依赖模型的 REST 接口测试用例生成方法研究

刘盈盈　杨秋辉　姚邦国　刘巧韵
四川大学计算机学院　成都 610065
(yyliu.eve@foxmail.com)

摘　要　REST 接口中普遍存在依赖关系,导致生成合理的接口调用序列与输入参数变得十分困难。现有的大多数方法只考虑了其中一种依赖关系,并需要人工执行繁杂的前置操作,生成的测试用例有效性仍然较低。针对以上问题,文中提出了一种基于依赖模型的测试用例生成方法。通过解析 OpenAPI 文档,该方法提取了接口内的操作间依赖关系与参数间依赖关系,并据此建立了两种依赖模型,从模型生成测试用例,最后从 3 方面确定测试预言。实验结果表明,该方法的输入度量覆盖率达到了 100%,状态码类别、状态码、响应资源类型的覆盖率分别达到了 100%,91.67%,83.33%,并能在限定时间内检出接口内部缺陷;与 RESTler 和 RESTest 相比,该方法的输出度量覆盖率最大提高了 36%,触发了最多次的异常响应状态码,检测到接口异常响应的比例最大提高了 10%。该方法为 REST 接口的测试用例生成问题提供了有价值的参考。

关键词:REST 接口;测试用例生成;操作间依赖;参数间依赖;BERT 模型
中图法分类号　TP311.5

Study on REST API Test Case Generation Method Based on Dependency Model

LIU Yingying, YANG Qiuhui, YAO Bangguo and LIU Qiaoyun
College of Computer Science, Sichuan University, Chengdu 610065, China

Abstract　The prevalence of dependencies in REST API makes it difficult to generate a reasonable sequence of API calls with input parameters. Most existing approaches only consider one of these dependencies and require cumbersome manual preliminaries, thus the generated test cases are still less effective. To address the above problem, a test case generation method based on dependency model is proposed. By parsing the OpenAPI documentation, this method extracts the inter-operation dependencies and inter-parameter dependencies, establishes two dependency models, generates test cases from the models, and determines test oracles. Experimental results show that the proposed method achieves 100% input metric coverage, and 100%, 91.67%, and 83.33% coverage for status code category, status code, and response resource type, respectively, and can detect internal interface defects within a limited time. Compared with RESTler and RESTest, the proposed method improves the maximum 36% of output metric coverage, triggeres the most number of abnormal response status codes, and detects a maximum of 10% increase in the percentage of abnormal responses. The method provides a valuable reference for the test case generation problem of REST API.

Keywords　REST API, Test case generation, Inter-operation dependency, Inter-parameter dependency, BERT model

1　引言

目前,大多数云服务都通过 REST(Representational State Transfer)接口[1]进行访问。REST 接口基于 HTTP/S 协议,提供了一种统一的方式来创建(POST/PUT)、管理(PATCH/PUT/POST)、删除(DELETE)以及获取(GET)云资源[2]。作为目前主流的接口开发风格,REST 接口具有结构清晰、易于理解和扩展方便等特点,在工业界和学术界一直广受关注。

为了保证 REST 接口的质量,开放软件在发布前必须经过详细的接口测试。2021 年,SaltSecurity 的接口安全报告中指出:过去一年中,有 91%的组织发生过接口安全事故,严重者甚至导致了敏感数据外泄[3],造成不可估量的损失。因此,关于 REST 接口的测试研究具有重要的实际意义。目前,已有多个研究提出了关于 REST 接口的黑盒测试方法[4-7]。然而,一方面,目前相关的黑盒测试方法大多针对接口的单个操作进行测试,难以检测出由多个连续操作所触发的缺陷;另一方面,由于可能的输入参数构成了一个庞大的参数空间,在不考虑参数间依赖关系限制的情况下,无法在有限时间成本内遍历整个空间,难以生成合法的参数组合。

到稿日期:2022-08-07　返修日期:2022-11-15
基金项目:四川省自然科学基金(23NSFSC3752);四川大学专职博士后研发基金(2022SCU12077)
This work was supported by the Natural Science Foundation of Sichuan Province, China(23NSFSC3752) and Sichuan University Postdoctoral Science Research Foundation(2022SCU12077).
通信作者:杨秋辉(yangqiuhui@scu.edu.cn)

针对以上问题，本文在已有研究的基础上，提出了一种基于依赖模型的测试用例生成方法。该方法主要针对遵循OpenAPI规范的REST接口应用开发，具有较广泛的应用前景。

2 相关工作

在REST接口的各操作之间，可能存在这样的关系：操作A的响应数据是操作B的输入数据，那么操作A必须在操作B之前执行。在进行REST接口测试时，类似$\langle B,A\rangle$这样不合理的操作序列将无法覆盖到待测接口的有效状态转移，导致大幅度降低测试效率。针对该问题，Atlidakis等提出了RESTler工具[4]，通过分析接口操作间的生产者-消费者依赖关系，从响应信息中学习无效的操作组合来动态生成操作序列；Viglianisi等提出了RESTESTGEN工具[5]，通过编码API文档中可用操作之间的数据依赖，来生成常规场景和错误场景的测试用例。

而在REST接口的各参数之间，可能存在这样的关系：参数Y的存在性会受到参数X的影响，例如，若参数X被设置，那么参数Y必须同时被设置。在测试时，若未考虑到此类依赖关系，输入参数的生成空间将会呈指数级增长，难以在有限时间内搜索到合法参数组合。针对该问题，Martin-Lopez等[6]建立了一套描述参数依赖的特定领域语言——IDL(Inter-parameter Dependency Language)，提取了7种类型的参数间依赖关系并形成IDL文档。基于OpenAPI与IDL文档，该研究实现了RESTest框架，通过生成合法与非法输入参数组合来更快速地对被测接口进行更深入的测试评估。

但是，目前针对REST接口内部依赖关系的研究，大部分仅考虑了其中一类依赖[8]；并且，在目前处理参数间依赖问题的流行方案中，大多数基于特定领域语言，需要测试人员手动执行繁杂的前置操作，自动化程度较低。因此，针对现有方法的不足，本文提出了一种基于依赖模型的测试用例生成方法。

3 方案描述

在本文提出的方案中，其输入只需要待测接口的OpenAPI文档，基于比较规则与自然语言处理模型，实现了接口操作间依赖与参数间依赖关系的自动提取，据此建立了依赖关系模型，从测试序列生成和接口参数生成两个维度生成测试用例。图1为方案的整体流程图。

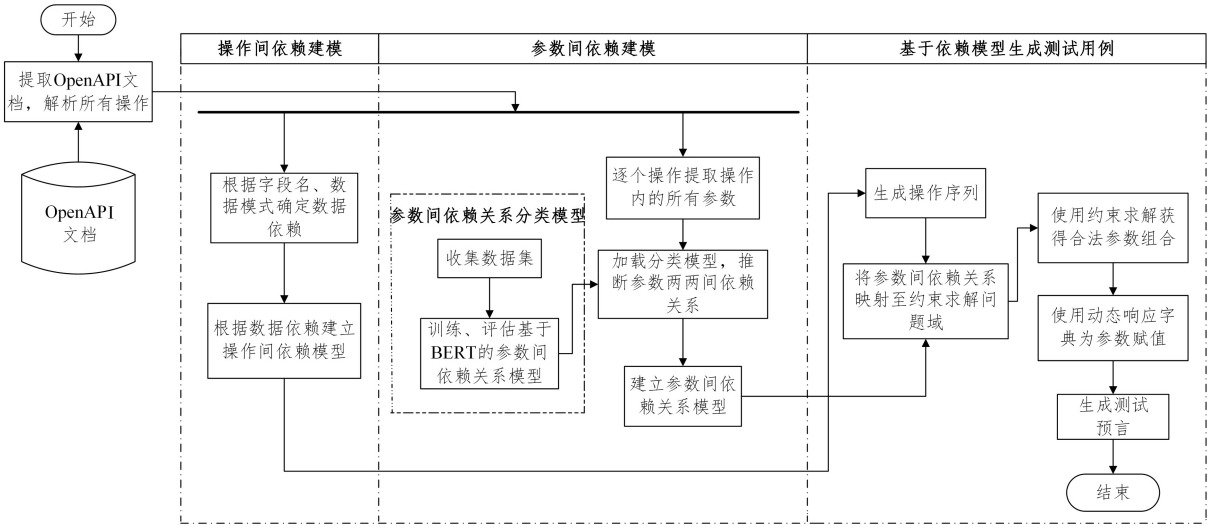


图1 方案的流程图

Fig. 1 Approach flow chart

3.1 操作间依赖建模

在REST接口的多个调用之间可能会存在共享数据，而这类共享数据隐式地确定了操作间的合理调用顺序，即产生该数据的调用应先于消费该数据的调用发生[4]。因此，根据此类数据依赖可以确定操作间的执行次序。

基于上述思路，本文根据相关研究[5]实现了数据依赖判断算法，如算法1所示。同时，考虑到不同对象的各字段可能会使用相同的名称命名，因此进入算法前，将数据对象名与字段名进行了合并，例如pet对象的id字段，将会被重命名为petid。另外，考虑到可能存在输入错误的情况，因此本文加入编辑距离作为比较规则。

算法1 判断是否存在数据依赖

输入：待分析的两个数据d_1，d_2

输出：是否存在数据依赖

```
1. If(d1,d2均为原子类型)：
2.     return IsSameType(d1,d2)
3. If(d1,d2均为复合类型)：
4.     return IsSameSchema(d1,d2)
5. return False
6. Procedure IsSameType(d1,d2)
7. If(d1字段名==d2字段名)
8.     or(编辑距离(d1,d2)≤thr)
9.     or(Snowball(d1)==Snowball(d2))：
10.        return True
11.    return False
12. Procedure IsSameSchema(d1,d2)
13.    isSame ← True
14.    For d1中的每一个子结构 sub：
15.        If d2中包含 sub 结构：
16.            isSame← isSame and 递归判断(d1,d2子结构是否相同)
17. return isSame
```

算法输入为待分析的两个数据，输出为判断其之间是否存在数据依赖关系。首先判断数据的数据类型，当均为原子类型时，进入算法 1 的第 7 行，比较两个数据的字段名是否完全相等或编辑距离是否小于等于阈值（本文设置为 2），若为假，则进一步使用 Snowball 算法提取字段名所代表的单词词干，根据词干比较结果判断是否存在数据依赖；当数据为复合类型时，进入算法 1 的第 12 行。由于 OpenAPI 规范中的复合类型使用 Schema 对象定义，因此算法 1 的第 16 行需要递归地比较两个 Schema 对象的子结构是否相同。

获得数据依赖后，建立操作间依赖图模型（Operation Dependency Graph，ODG）。ODG 为一个有向图，用 $G(N, N_0, N_f, E)$ 表示。其中，N 为接口所有操作的集合；N_0 为起始节点集合，表示不依赖任何操作的操作集合；N_f 为终止节点集合，表示不被任何操作所依赖的操作集合；E 为有向边集合，对于任意操作 o_1, o_2（$o_1, o_2 \in N$），若 o_2 存在对 o_1 的数据依赖，则建立一条有向边 $e = o_1 \rightarrow o_2, e \in E$。

3.2 参数间依赖建模

为了从 OpenAPI 文档中自动提取参数间的依赖关系，本文训练了一个基于 BERT 模型的参数间依赖关系分类模型。利用该分类模型对接口内的所有参数进行关系分类，建立参数间依赖模型。

基于现有研究[9]提出的多参数依赖关系类别，本文首先考虑了参数两两间的依赖关系。给定操作内任意两个参数 p_1 和 p_2，其间可能存在以下 4 种依赖关系中的任一种：

1）$Require(p_1, p_2) = p_1 \Rightarrow p_2$，表示当参数 p_1 指定时 p_2 必须指定，p_2 对 p_1 没有影响；

2）$Either(p_1, p_2) = p_1 \oplus p_2$，表示参数 p_1 或 p_2 只能指定一个；

3）$Both(p_1, p_2) = p_1 \wedge p_2$，表示参数 p_1 和 p_2 必须同时指定；

4）$None(p_1, p_2) = \varnothing$，表示参数 p_1 和 p_2 没有关系。

3.2.1 数据收集与预处理

从相关文献[10]以及 API 仓库[11]中收集大量参数描述文本，进行数据清洗后，手工标记文本中的参数实体：在第一个参数实体的前后添加 # 符号，在第二个参数实体的前后添加 $ 符号。若涉及的参数多于两个，则进行两两标记后，对结果进行合并，得到所有参数的依赖关系。处理完毕后，以 6∶2∶2 的比例将数据集划分为训练集、验证集和测试集。样本数据的基本情况如表 1 所列。

表 1 样本数据的基本情况

Table 1 Basic information of sample data

依赖关系	样本数量	示例数据
Require	254	(1)If # project_id # is specified, $ email $ and user_id is required (2)If project_id is specified, # email # and $ user_id $ is required (3)If # project_id # is specified, email and $ user_id $ is required
Either	234	You must specify one of # user_id #, $ user_name $
Both	220	# longitude # must be used with $ latitude $
None	1692	Apply an # offset # to the $ query $
总计	2400	—

3.2.2 构建分类模型

由于 OpenAPI 规范建议开发者以自然语言来描述参数间的约束关系，因此可以考虑使用自然语言处理技术来辅助提取和分析此类关系。参数间依赖关系的推断类似于一般的关系抽取任务，需要进行命名实体识别与关系分类两个步骤。因此，本文使用 Wu 等[12]提出的关系抽取模型 R-BERT 进行依赖关系分类。该模型主要分为两个部分：第一部分为特征提取，即获得参数描述文本的嵌入表示；第二部分为一个线性分类器，如图 2 所示。

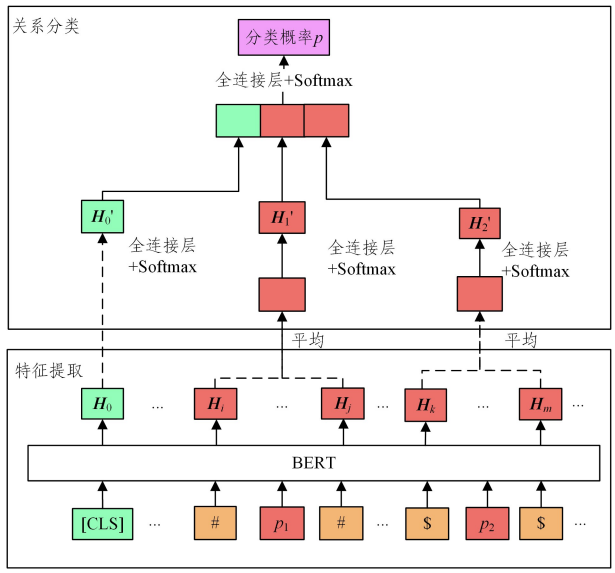


图 2 基于 BERT 模型的参数间依赖关系分类模型结构

Fig. 2 Structure of classification model based on the BERT model for inter-parameter dependency

模型以一段连续文本为输入，其中包含的两个参数文本已在预处理阶段进行标识，记为 p_1 和 p_2。同时，以[CLS]标识文本的起始位置。经过 BERT 模型后，获得该段文本的向量表示，其中向量 $\boldsymbol{H}_i$ 至 $\boldsymbol{H}_j$ 为参数 p_1 对应隐层输出，向量 $\boldsymbol{H}_k$ 至 $\boldsymbol{H}_m$ 为参数 p_2 的隐层输出，分别对两部分隐层输出进行平均后，获得两个参数的向量表示，经过激活操作后输入至一个全连接层，得到 $\boldsymbol{H}_1'$ 与 $\boldsymbol{H}_2'$；同样，[*CLS*]对应的隐层输出 $\boldsymbol{H}_0$ 进行激活操作后输入至全连接层，获得 $\boldsymbol{H}_0'$。最后，将 $\boldsymbol{H}_0'$，$\boldsymbol{H}_1'$，$\boldsymbol{H}_2'$ 进行拼接后，输入到全连接层，使用 Softmax 获得分类概率 p。模型使用交叉熵作为损失函数，在验证集上进行评估、调整后，得到主要超参数值：批量大小＝16，学习率＝0.05，训练轮次＝30，Dropout rate＝0.1。

由于参数间的依赖关系分类属于文本分类问题，因此考虑使用精确率 Precision、召回率 Recall 以及 F1 指标来评估模型性能。除此之外，由于使用的数据具有不平衡的特点，并且每个类别具有同等的重要性，因此使用 Macro-F1 指标来评估分类器的效果，计算式如式(1)所示：

$$Macro\text{-}F1 = \frac{\sum_{i=1}^{4} F1i}{4} \tag{1}$$

3.2.3 构建参数间的依赖模型

用构建的分类模型进行参数间依赖关系分类，建立参数间依赖模型。首先遍历整个操作列表，提取每个操作的所有

参数项与描述文本，筛选出包含多个参数名的文本，使用分类模型进行关系分类。例如，若参数 Y 的描述文本中包含字段“Required when using X”，而参数 X 的文本中不包含此类关系描述，则此时可通过分类模型得到依赖关系(X, Y, Require)。处理完毕所有操作后，对包含相同参数对的分类结果进行检查，去除矛盾的依赖关系，最终获得参数间的依赖关系模型。

3.3 基于依赖模型生成测试用例

一个完备的测试用例主要包含 3 部分，即操作序列、参数组合以及测试预言。

3.3.1 生成操作序列

生成新序列的过程，即从候选操作集中选择操作，附加至当前操作序列的最后。文献[4]提出使用广度优先搜索策略(BFS)，但随着操作序列不断生成，候选操作集将越来越大，使用 BFS 策略需要对候选集中的每一个操作生成新的操作序列，保存这些状态需要消耗巨大的资源。因此，本文使用一种基于随机搜索的序列生成方法。

算法 2 操作序列生成

输入：操作间依赖关系模型 G(N, N_0, N_f, E)；最大序列长度 L；最大运行时间 TimeLimit

输出：操作序列集合 seqs

```
1. seqs← {}
2. TimeStart ← time()
3. for i=1 to L do
4.       seqs ← seqs⋃RandomGenerateSeq()
5. repeat
6.       seq,satisfied ← Ø,{}
7.       seq ← RandomChoose(seqs)
8.       for Each n in N do
9.         If HasDependencyOn(n,seq) then
10.             satisfied ← satisfied⋃{n}
11.      seq ← Append(seq,RandomChoose(satisfied))
12.      seqs ← seqs⋃ seq
13.      TimeEnd=time()
14. Until TimeEnd−TimeStart=TimeLimit()
15. return seqs
```

算法 2 以操作间依赖关系模型 G、最大初始序列长度 L 以及最大运行时间 $TimeLimit$ 为输入，以生成的操作序列集合 $seqs$ 为输出。算法 2 中的第 4 行，从起始节点集 N_0 中随机选择若干节点，初始化若干长度小于预设值的序列，形成序列集合。算法 2 中的第 7 行，每次从序列集合中随机选择一条序列，根据依赖模型 G 可确定与该条序列中的操作存在数据依赖的操作集合，从该集合中随机选择一个操作，将其添加至序列末尾，完成一次对序列的扩展。每生成一个测试序列都需要判断时间成本是否耗尽，如算法 2 的第 14 行。

相比 BFS 算法，该方法由于并未记录遍历操作过程中可能产生的所有序列，因此有效减少了资源开销，运行效率更高。

3.3.2 确定参数组合与参数值

为了确定调用接口时应提供的输入参数组合，根据已有研究[9]，将 4 种参数间依赖关系映射至约束满足(Constraints Satisfaction Problem，CSP)问题域，用三元组(V, D, C)进行表示。其中，变量 V 为操作内的所有参数，定义域 D 为每个参数类型与布尔变量的并集，布尔变量用于记录该参数是否已被设置，约束条件 C 由参数间的依赖关系模型转化而来。针对具体的参数向量 $\boldsymbol{v}_i$，转化关系如表 2 所列，注意 None 关系对参数无约束作用，可以直接忽略。

表 2 参数间依赖映射为 CSP 约束

Table 2 Inter-parameter dependency mapping as a CSP constraint

参数间依赖关系	CSP 映射
[Require]: If $\boldsymbol{v}_i$ Then $\boldsymbol{v}_j$	$C \leftarrow C \cup \{map(\boldsymbol{v}_i) \Rightarrow map(\boldsymbol{v}_j)\}$
[Either]: $\boldsymbol{v}_i$ XOR $\boldsymbol{v}_j$	$C \leftarrow C \cup \{map(\boldsymbol{v}_i) \Rightarrow \neg map(\boldsymbol{v}_j)\}$
[Both]: $\boldsymbol{v}_i$ AND $\boldsymbol{v}_j$	$C \leftarrow C \cup \{map(\boldsymbol{v}_i) \cap map(\boldsymbol{v}_j)\}$

表 2 中，$map(k) = C \leftarrow C \cup \{vSet == \text{true}\}$，$1 \leqslant i \leqslant |V|$，$1 \leqslant j \leqslant |V|$，$vSet$ 代表参数 v 是否为该操作的必需参数。使用谷歌开源工具 OR-Tools[13] 进行约束求解，求解结果即为该操作的合法参数组合。

针对参数组合中各参数的赋值问题，使用动态响应字典进行数据复用。为了解决动态响应字典中初始数据的缺失问题，在测试运行前，首先对 REST 接口的 OpenAPI 文档进行一次静态分析，从中提取可用的示例数据，将其作为静态数据记录在动态响应字典中。在没有可复用数据时，根据不同的参数类型，采用不同的方法随机生成对应类型数据。所有参数赋值完毕后，需要构造 HTTP 请求并执行，若执行成功，则将该次操作的响应数据存至字典中，以便下次请求进行数据复用。

3.3.3 生成测试预言

由于本文主要针对接口的功能验证来生成测试用例，因此考虑接口在功能实现层面的测试预言。根据 OpenAPI 文档对接口功能的定义，可从状态码、JSON-Schema 和 Key-Value 这 3 个方面生成测试预言。

1)使用响应状态码生成测试预言：接口设计人员通常会在 OpenAPI 文档中标明正确或错误处理请求时，客户端将收到的响应状态码，据此可以确定测试预言。

2)使用 JSON-Schema 生成测试预言：JSON-Schema 是定义负载数据约束的标准，数据发送和接收者基于该约束对数据进行验证，以保证交换数据的正确性。因此，根据 JSON-Schema 中的限制关系可对接口调用结果生成测试预言。

3)使用 Key-Value 生成测试预言：完全使用自动化方法对响应信息进行检查是一项十分困难的工作。因此，本文提供了一种半自动化的 Key-Value 测试预言生成方法。该方法首先使用状态码和 JSON-Schema 约束生成测试预言，然后由测试人员手动根据操作序列设计输入参数并记录预期响应，最后扩展自动生成的测试预言。该方法生成的测试用例可用于 REST 接口的回归测试。

4 实验验证

4.1 实验数据

使用 Github 上已公开的 5 个 REST 接口项目进行实验，这些项目已被应用于相关研究[4,14-15]。表 3 列出了各项目的详细信息。其中，Petstore 项目为 OpenAPI 官方

示例项目，其数据直接保存于内存中，因此未使用其他数据库。本文选择的实验对象涉及不同编程语言、开发框架与数据库，并且具有不同复杂度，能够较好地代表真实世界的 REST 接口。

表 3 实验对象的详细信息

Table 3 Experimental subject details

实验对象	开发语言	开发框架	访问点数量	操作数量	数据库	规模(LOC)
BlogPosts	Python	Flask	2	6	Sqlite	189
Petstore	Java	Spring Boot	14	20	—	1235
Rest-news	Kotlin	Spring Boot	2	7	H2	679
Catwatch	Java	Spring Boot	6	6	PostgreSQL	7143
Cyclotron	Javascript	Express	40	52	MongoDB	5038

4.2 评估指标

1)OpenAPI 文档覆盖度量

由于本文方案基于 OpenAPI 文档对接口进行功能测试，因此考虑采用基于功能的测试覆盖率指标。当前应用最广的是 Martin-Lopez 等提出的覆盖度量[16]，该研究将覆盖度量分为两类：输入覆盖度量和输出覆盖度量。输入覆盖度量主要包括路径覆盖、操作覆盖、参数覆盖、参数值覆盖与请求资源类型覆盖；输出覆盖度量主要包括状态码覆盖、状态码类别覆盖以及响应资源类型覆盖。这些覆盖度量的计算式如式(2)所示：

$$Coverage(metric)=\frac{tested\_num(metric)}{total\_num(metric)} \tag{2}$$

其中，*metric* 表示所使用的覆盖标准，可取上述覆盖标准中的任一种，在选定的覆盖标准 *metric* 下，$total\_num(metric)$ 表示 OpenAPI 文档中定义的对应元素总数量，$tested\_num(metric)$ 表示测试过程中覆盖到的元素数量。

2)错误检测能力

5XX 响应状态码表示接口内部的实现缺陷[4]，该状态码在测试过程中出现的情况也反映了测试用例的检错能力。本实验计算了异常响应比率，即 5XX 响应状态码的占比，计算式如式(3)所示：

$$异常响应比率=\frac{5XX\ 响应状态码数量}{响应状态码总数} \tag{3}$$

该比率反映了已运行用例的总体检错能力，其值越高表示检错能力越强。

4.3 结果分析

4.3.1 实验 1：生成的测试用例质量评估

首先在 Blogpost 项目上对该方案进行了可行性验证。该项目包含一个人工植入的缺陷，且只有在考虑了操作间的依赖时，才能触发该缺陷。实验时，所使用的分类模型精确率为 0.810，Macro-F1 值为 0.805，召回率为 0.808。在该项目上运行方案 10 min，设置最大序列长度为 5，使用 Restats[17] 工具统计 OpenAPI 文档中各覆盖度量的覆盖率，结果如表 4 所列。

表 4 BlogPosts 项目 OpenAPI 文档的覆盖情况

Table 4 BlogPosts project's OpenAPI documentation coverage

类别	度量	总数	被覆盖数	覆盖率/%
输入覆盖	路径	3	3	100
	操作	6	6	100
	参数	2	2	100
	参数值	0	0	—
	请求资源类型	2	2	100
输出覆盖	9	9	100	
	状态码	12	11	91.67
	响应资源类型	6	5	83.33

从表 4 可知，本文方案能够对 BlogPosts 接口的输入度量实现完全覆盖，表明生成的测试用例能够对被测接口的不同部分进行测试；同时，输出度量覆盖率均大于 80%，这表明测试用例能够覆盖到接口内大部分的业务逻辑。

在第 8 个请求-响应循环后，本文方案发现了 BlogPosts 中的人工植入缺陷，并在第 278 次和第 465 次请求中重复检测到了该缺陷。

4.3.2 实验 2：本文方案与 RESTler 和 RESTest 工具的对比实验

本文主要关注 REST 接口的测试用例生成问题，故与同类型研究 RESTler[4] 和 RESTest[6] 进行对比实验，以验证方案的有效性。

RESTest 将 OpenAPI 文档翻译为 IDL 语言，进而建立测试模型，分别生成正确和错误场景的测试数据。本文方案的输入为原始的 OpenAPI 文档，无须对文档进行进一步处理。

RESTler 通过分析操作间的生产者-消费者关系，迭代构造请求序列，从响应信息中学习无效的操作组合来生成操作序列。本文方案使用 ODG 模型，根据到目前为止已选择的操作集合，动态地计算下一个要测试的操作。

由于输入度量覆盖仅与生成的测试用例有关，所有方案均能在限定时间内达到完全覆盖，因此实验时仅统计输出度量覆盖。在各方案运行 1 h 后，统计 4 个项目上的状态码、状态码类别以及响应资源类型覆盖，结果如表 5 所列。

表 5 不同方法在各项目上的输出度量覆盖

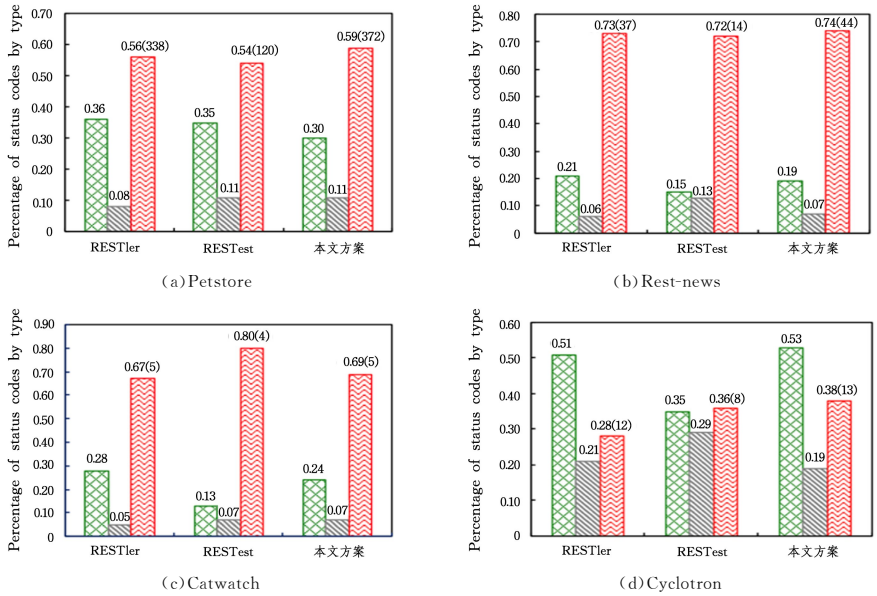
Table 5 Output metric coverage of different methods on each project

(单位：%)

项目	状态码类别覆盖			状态码覆盖			响应资源类型覆盖		
	RESTler	RESTest	本文方法	RESTler	RESTest	本文方法	RESTler	RESTest	本文方法
Petstore	64	**93**	91	60	**90**	88	79	87	**89**
Rest-news	83	**95**	**95**	85	**90**	85	80	**84**	83
Catwatch	64	**81**	80	81	**88**	**88**	88	78	**89**
Cyclotron	55	91	**91**	84	**90**	89	75	**87**	84

从表 5 可知，本文方案所生成的测试用例输出度量覆盖能力优于 RESTler，接近于 RESTest。由于 RESTest 在生成测试用例时考虑了两种场景，因此更容易获得不同类型的响应，故覆盖能力最佳。RESTler 主要使用静态设置的数据字典以及文档提供的样例输入来生成参数，相比之下，所触发的响应类型比较有限。而本文使用动态响应字典来记录运行时信息，在参数赋值阶段能够进行数据复用，因此可以充分利用前序信息来提高输出覆盖率。另外，当字典中不存在可复用数据时，本文方案使用随机方法生成数据，这类数据更容易导致接口返回错误类型的状态码。与 RESTler 相比，在状态码类别覆盖、状态码覆盖和响应资源类型的覆盖上，本文方案分别高出了 12%～36%，0%～28% 以及 1%～10%。

为了评估几种方案所生成测试用例的检错能力，统计测试完成后所收到的各类型状态码的占比，结果如图 3 所示。同时，由于各方案在相同时间内生成的测试用例数量存在较大差异，因此，在图 3 中对各方案触发的 5XX 状态码的数量进行了标注（括号内数值）。



(a) Petstore (b) Rest-news (c) Catwatch (d) Cyclotron

图 3 不同方案在不同项目上的响应状态码分布情况

Fig. 3 Distribution of response status codes for different programs on different projects

从 5XX 状态码的占比可以看出，除了在 Catwatch 项目上，RESTest 方案表现最优，在其余项目上，本文方案均优于 RESTler 和 RESTest。其中，在 Cyclotron 项目上，本文方案的检错效果最佳。通过分析发现，由于 Cyclotron 项目中操作数量和参数数量较多，因此包含了较多的操作间依赖与参数间依赖关系；而 Catwatch 中只包含了 6 个查询操作，各操作间不存在依赖关系，因此擅长对单个操作进行测试的 RESTest 的效果较好。同时，从括号内标注的 5XX 状态码的数量对比中可以看出，在相同时间内，本文方案生成的测试用例在各项目上触发的异常响应状态码数量最多，同样证明了本文方案的检错能力更佳。

可行性实验表明，本文方案生成的测试用例能够检测出 REST 接口中已知的缺陷，达到较高的 OpenAPI 文档覆盖率，具有实际业务意义。对比实验表明，在接口内访问点数量和操作数量较多的情况下，可能包含了大量操作间依赖与参数间依赖关系，本文方案由于综合考虑了两种依赖关系，因此无论是在 OpenAPI 文档的覆盖能力上，还是通过异常响应状态码检测接口错误的能力上，与现有流行方案相比都有较好的表现。

综上所述，在接口内包含了较多访问点与操作的应用中，本文方案能够充分利用依赖关系和响应信息，生成具有实际业务意义的操作序列，提高接口覆盖率与检错率。

结束语 针对如何生成具有实际业务意义的 REST 接口测试用例，提出了一种基于依赖模型的 REST 接口用例生成方法。实验结果表明，本文方案所生成的测试用例能够在有限时间内检出接口内部的缺陷；与现有流行方案的对比实验表明，所生成的测试用例不论在覆盖率还是检错率方面，都有一定程度的提升。

本文整体方案仍存在一定不足，未来需要继续改进，包括但不限于：本文方案在几个中小型项目中表现出了更好的测试效果，在大型项目中的有效性需要更多实验来验证；未来可以考虑应用更先进的自然语言处理技术来进一步处理 OpenAPI 文档，规范数据命名；未来可以根据被测接口的应用领域生成领域属性更强的测试数据；除了参数间存在性依赖关系，可以进一步考察参数间更多类型的依赖关系；可以将本文方案与变异测试相结合，对已生成用例进行变异，得到大量无效用例，进而测试接口鲁棒性。

参 考 文 献

[1] FIELDING R T. Architectural styles and the design of network-based software architectures[M]. Ann Arbor: University of California, Irvine, 2000: 76-105.

[2] ATLIDAKIS V, GODEFROID P, POLISHCHUNK M. Checking security properties of cloud service REST APIs[C]// 2020 IEEE 13th International Conference on Software Testing, Validation and Verification. 2020: 387-397.

[3] SaltSecurity. 91% of organizations had an API security incident last year[EB/OL]. The State of API Security. [2021-02-26]. https://salt. security/api-security-trends.

[4] ATLIDAKIS V, GODEFROID P, POLISHCHUNK M. Restler: Stateful rest API fuzzing[C]// 2019 IEE ACM 41st International Conference on Software Engineering. 2019: 748-758.

[5] VIGLIANISI E, DALLAGO M, CECCATO M. Resttestgen: automated black-box testing of restful apis[C]// 2020 IEEE 13th Interna-tional Conference on Software Testing, Validation and Verification. 2020: 142-152.

[6] MARTIN-LOPEZ A, SEGURA S, RUIZ-CORTES A. RESTest: automated black-box testing of RESTful web APIs[C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021: 682-685.

[7] LARANJEIRO N, AGNELO J, BERNARDINO J. A black box tool for robustness testing of REST services[J]. IEEE Access, 2021, 9: 24738-24754.

[8] CORRADINI D, ZAMPIERI A, PASQUA M, et al. Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIs[C]// 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation. 2021: 226-236.

[9] MARTIN-LOPEZ A. Automated analysis of inter-parameter dependencies in web APIs[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. 2020: 140-142.

[10] MARTIN-LOPEZ A, SEGURA S, RUIZ-CORTÉS A. A Catalogue of Inter-parameter Dependencies in RESTful Web APIs [M]. Service-Oriented Computing. Cham: Springer International Publishing, 2019: 399-414.

[11] APIs. GURU. Browse APIs. APIs. guru [EB/OL]. [2022-01-16]. https://apis. guru/.

[12] WU S, HE Y. Enriching pre-trained language model with entity information for relation classification[C]// Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2019: 2361-2364.

[13] Get Started with OR-Tools for Python[EB/OL]//Google Developers. [2022-01-17]. https://developers. google. com/optimization/introduction/python.

[14] ARCURI A. RESTful API Automated Test Case Generation with EvoMaster[J]. ACM Transactions on Software Engineering and Methodology, 2019, 28(1): 3: 1-3: 37.

[15] CORRADINID, ZAMPIERI A, PASQUA M, et al. Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIs[C]// 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation. 2021: 226-236.

[16] MARTIN-LOPEZ A, SEGURA S, RUIZ-CORTÉS A. Test coverage criteria for RESTful web APIs[C]// Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation. 2019: 15-21.

[17] CORRADINI D, ZAMPIERI A, PASQUA M, et al. Restats: A test coverage tool for RESTful APIs[C]// 2021 IEEE International Conference on Software Maintenance and Evolution. 2021: 594-598.

LIU Yingying, born in 1998, postgraduate. Her main research interest is software quality assurance.

YANG Qiuhui, born in 1970, Ph.D, associate professor. Her main research interests include software quality assurance and testing and software engineering.

（责任编辑：喻藜）