基于数据库模式的源代码数据语义恢复*)

吴浩　彭鑫　杨益明　赵文耘
复旦大学计算机科学技术学院，上海 200433
Email: {wuhao, pengxin, 051021056, wyzhao}@fudan.edu.cn

摘　要　源代码中的数据语义，即各种程序变量的含义，对于程序理解具有重要的意义。然而现有的程序和数据逆向分析方法中，源代码和数据模型（例如数据库模式）的分析往往是割裂开的，因此很难实现源代码中的数据语义逆向恢复。针对这一问题，本文结合数据库数据和代码逆向工程方法，从数据库表结构定义出发，提出了一种恢复遗产系统源代码中数据语义方法。该方法首先从数据库定义文件中获取数据库字段的语义信息，通过对源代码中与数据库交互的代码片断进行分析，利用系统依赖图和字符串静态计算方法，发现程序中变量和数据库中表字段间的对应关系。然后，通过变量和上下文的数据依赖关系，将数据库定义中的语义信息在程序中进行传递。

关键词　数据语义、恢复、静态分析、系统依赖图、字符串计算
中图法分类号：TP311　　文献标识码：A

Recovering data semantics in source code based on database schemes

WU Hao PENG Xin　YANG Yi-ming ZHAO Wen-yun

(School of Computer Science and Technology, Fudan University, Shanghai 200433, China)

Abstract　Data semantics in source code, i.e. meanings of those variables, are important for program comprehension. However, in existing program and data reverse analysis methods, source code and data model (e.g. database schema) are separated, making it difficult to recovery data semantics from source code. To solve the problem, this paper, based on the database scheme files, considering both database information and program reverse engineering techniques, proposes a method to determine the meaning of data in programs. The method first gets the semantics of data in database from database scheme, and then explores the code snippets that interact with DBMS to find the direct mapping of the program variable and the database columns utilizing the system dependency graph and string expression analysis technique. And then data semantics are assigned or transmitted according to the dependence relationships of the variables in the context.

Key words　Data Semantics, Recovery, Static Analysis, System Dependency Graph, String Expression Analysis

1 引言

以数据为中心的遗产系统，如商业信息系统，通常依赖于大量的数据库表（数据文件）。这些系统中既包含了大量的商业逻辑，又含有至关重要的商业数据，而其庞大的规模，使得丢弃这些系统或对其进行重新开发，成为不可能的事情。为此，越来越多的信息系统部门陷入系统维护的泥沼。当重新购买和替换这些已有的系统变为不切实际的目标时，对已有的系统进行逆向工程就成为唯一的选择。通过系统逆向工程，理解既有代码的结构和数据的意义，以降低维护的成本。对代码逆向工程的研究工作已经持续了很多年，并在不断的完善。但数据逆向工程的出现则相对晚一些。但它从另一个方面扩展了软件逆向工程，对于以数据为中心的遗产系统有着重要的意义。

在信息管理系统或者是以数据为中心（数据密集型）的应用程序中，数据的逆向工程可以分为单独的两个方面，一是对数据库进行逆向工程，二是对程序中的数据进行逆向工程。其中，前者对应于数据库逆向工程（Database Reverse Engineering, DBRE），是数据库方向的研究者进行数据逆向工程时的主要关注点。DBRE 是通过数据库管理系统(Database Management System, DBMS)中数据定义语言(DDL)文本和使用这些数据的源代码中恢复出对应数据库模式的过程[1]，包括 E-R 模型的恢复[2][3]，表间依赖关系和约束的发现[4]等。而后者主要对应的是软件工程研究者的工作,通过对源代码的分析[5][6][7]，发现数据的上下文依赖关系。目前，软件工程方向对逆向工程的研究主要关注于代码的逆向工程，以理解程序的模块的功能，恢复设计模式，定位特征。很少结合数据库中的数据模型，进行逆向分析，对数据逆向工程方向的研究也相对较少。

通过对以上的相关工作的研究，我们发现当前的研究工作中主要存在以下不足和问题：1)对于源代码的理解是独立于外部数据进行的，而外部数据中通常包含了理解源代码所必需的信息；2)在对源代码进行理解的过程中，无法识别出具有相同语义却命名不同的变量或对象；3)尽管提出了源代码中数据依赖的发现和恢复方法，但缺乏对分析源代码与数据库之间依赖的方法和技术的研究。针对当前研究工作中存

*)项目资助：国家自然科学基金(60703092)、国家863计划(2006AA01Z189,2007AA01Z125)。**吴浩**　男（1982-），硕士生，主要研究方向为软件再工程等；**彭鑫**　男（1979-），讲师，博士，主要研究方向为软件产品线、软件再工程等；**杨益明**　男（1983-），博士生，主要研究为软件再工程等。赵文耘 男(1964-)，博士生导师 ，主要研究为软件工程等

在的问题，本文的研究根据数据库模式，通过分析获得数据库中数据表的意义及其字段的意义，通过构建实体-关系模型，表示数据的语义。然后，在系统依赖图的基础上，自动的分析源程序与数据库进行交互的代码，把源程序中的数据元素同数据库中的数据元素进行关联，进而把数据库中已知的数据语义传递到源程序的数据元素中。这样，系统不同模块之间的数据元素定义统一到数据库的定义文件中，方便维护工作的进行。

本文的组织结构如下：第一章引言主要介绍了数据逆向工程的研究对象和现状，并说明本文研究的动机和方法。第二章概述了本文所用的相关概念和技术。第三章详细阐述了方法的主要步骤和关键技术，重点讨论了确定源代码和数据库交互片段的方法，SQL 值的静态计算技术，SQL 解析树构造技术。第四章讨论实现方法和初步的应用分析结果。最后，第五章总结了本文研究的意义，指明了存在的不足和将来的研究方向。

2 相关技术

系统依赖图由 Horwitz 在文献[8]中提出，用于进行过程间的切片。它在 Ottenstein[9]所提出的程序依赖图的基础上进行扩充，包含了过程间的依赖关系。Larsen[10]对 SDG 进行进一步扩展，使其可以表示面向对象程序的系统依赖关系。Walkinshaw [11]中描述了构造 Java 系统的 SDG 的方法，并提供了原型实现。

Java String Analysis（JSA）[12]库开发了一种产生 Java 字符串模型的机制。特别的，JSA 对应用程序静态的执行保守的字符串分析，创建描述特定字符串在给定点所有可能值的自动机。

Soot[13]是一个 Java 程序分析优化框架，它提供了四种中间表示，用以分析和转化的 Java 字节码。Soot 可以作为一个工具单独，用来优化或检查分析类文件；也可以作为一个框架使用，通过其提供的 API 对 Java 字节码进行分析和转换，以获得需要的结果。

3 方法流程

我们提出的源代码数据语义恢复方法的第一步是提取 E-R 模型，这一步的输入为数据库模式，输出为数据库中数据的语义信息。构建 E-R 模型的过程，主要是对数据库模式进行分析，通过理解表、表中字段的含义和表与表之间、字段与字段之间的关系，来确定其所反映的客观世界的实体以及实体间的关系。在获得 E-R 模型的基础上，我们对数据库数据的语义进行定义，并记录。方法的第二步是对源代码进行静态分析，这一步的输入为程序源代码，输出为包含源代码变量和数据库列直接映射信息的系统依赖图。方法第三步为映射传递数据语义，其输入为前两步的分析结果，输出为扩展的系统依赖图。这里，扩展的系统依赖图是在传统系统依赖图的基础上，增加数据库相关的依赖边，并标注相应变量的语义。最后，利用源代码中数据的上下文依赖关系，从已获得语义的变量出发，把数据语义进一步进行传递。

3.1 提取E-R模型

本文以下所采用的数据语义，表示的就是通过 E-R 模型来定义的数据对象的构成，和它们之间的关系。从数据库中数据出发，进行数据语义的理解，主要基于以下的考虑。首先，数据库模式所定义的数据模型，更加接近于人们对现实世界的认识，更容易理解。另外，数据库模式和数据模型定义文件，独立于应用程序，更加易于维护，因此，在多数系统中这一部分文当相对完整和准确。理解数据库数据的工作量远远小于对代码的理解。E-R 模型中的实体和关系与数据库模式中的表以及表之间的关系存在着一一对应的关系。一般来说，从数据库的模式出发，恢复出数据模型-E-R 模型是容易做到的，并且理解数据库中数据的语义是容易的。

Dowming Yeh[2]提出了从遗留数据库的表结构中提取 E-R 模型的方法。在当前的数据库逆向工程研究中，通常假设数据库中关于属性、主键、引用等的语义信息是完备的。但实际情况可能并非如此。以数据库中的表名、字段名来直接表示数据的语义信息，可以自动的完成 E-R 图的构建过程。Alhajj [3]提出了从遗产数据库中恢复出 E-R 模型的方法。这里，可以利用其恢复出的初步 E-R 模型，然后对其上的实体名、属性名等进行修正，使其对应与我们观念中的术语，并增加领域信息，描述实体和属性。

3.2数据库与源代码间的交互分析

在数据库应用系统中，源代码可以通过不同的方式访问数据库。如利用数据操作语言（DML），结构查询语言（SQL）或是直接的 API 进行访问。根据数据操作语言和源代码数据库访问代码间的相似性，人们一般把源代码与数据库的交互方式分为四类。它们是直接的数据库访问、内建的数据库访问、嵌入式数据库访问以及基于调用的数据库访问。其中，直接的数据库访问是在进行标准的数据库文件访问时，源代码中使用直接的数据操作语句对数据库进行操作（如 COBOL 语言中的 READ, WRITE, DELETE, …）；内建的数据库访问是对某些特定的 DBMS，程序语言中提供了访问数据库的内建指令进行访问（如 FIND,STORE ， …）；嵌入式的数据库访问通常是指通过 SQL 对数据库进行访问，即在代码中嵌入了一些属于数据库操作的 SQL 代码，通过调用提供的方法，并将 SQL 语句作为参数传递给这些方法进行数据库的访问；基于调用的数据库访问是指通过调用数据访问模块（Database Access Module, DAM）提供的方法，来对数据库进行访问。在现代的数据库应用程序设计，主要采用后两种方式。并且应用程序和 DBMS 的交互是通过应用程序向 DBMS 发送数据操作语句（SQL 语句是现在数据库操作的主要标准），DMBS 返回处理结果进行的。在本文中，着重针对 Java 语言讨论的嵌入式的数据库访问模式。

这一步骤的主要流程如图 1 所示，首先对嵌入式数据库访问方式进行分析，利用系统依赖图，发现数据库与源代码之间的交互，并通过对 SQL 语句的分析，确定数据库字段和源代码变量之间的映射关系，并进一步构造扩展的系统依赖图（Extended System Dependency Graph, eSDG）。

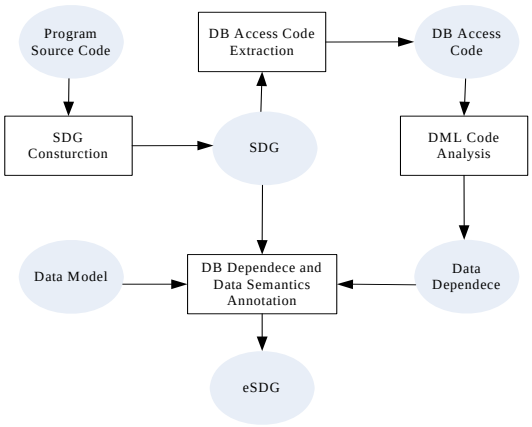

图 1　数据库与源代码映射提取流程

在 Java 语言中，与数据库的交互主要通过 JDBC 进行。类 Statement 等提供了一系列进行数据库查询和更新的方法，如 executeQuery(String sql)等。(在下文中，我们将采用正则表达式的方法对这类方法进行统一描述，如对于 Statement 类中的数据库操作方法，便可记为 java.sql.Statement->execute*(...))。通过简单匹配的方法，查找程序中出现以上类和方法出现的地方，可以发现源代码与数据库交互的片断。

我们当前的实现关注于 JDBC 的方法调用接口，由于其 SQL 语句以字符串的方式传递，我们利用 Java String Analysis（JSA）库在特定的调用点收集字符数据。这一步是对每个 JDBC 方法调用者构造自动机。 这一自动机集合描述了应用程序所有可能执行的 SQL 查询。JSA 库对每个接收 String 参数的方法，提供了字符层次上的自动机。更具体的说，他们的技术首先创建系统的控制流图，然后从控制流图创建非确定性有限自动机(NDFA),来表示 String 参数的所有可能值。进一步通过把字符层级上的自动机，转换为记号级别的自动机，把结果进行精化。通过对字符级的自动机执行深度优先遍历，识别出 SQL 关键词，操作，或是文字作为记号，并以这些记号作为变迁，构建新的自动机。例如，字符级上的带有字符‘F’，‘R’，‘O’,和‘M’一系列变迁，被识别为 SQL “From” 关键字，并以标记为 FROM 的单个变迁进行替换。结果为以 SQL 关键字，操作符和数据库对象标识为变迁的 NDFA。.附加标志 VAR 被用来表示无法通过静态分析识别的用户输入。我们通过对源代码进行插装，在输入变量前加入其名称，这样就可在生成的自动机获得输入变量的信息。例如，如下方法调用 executeQuery("SELECT NAME FROM CUSTOMER")将产生如图所示的自动机：

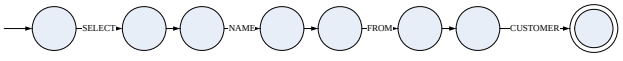

图 2　SQL 语句对应的自动机

获得了以上自动机之后，利用 JSA 库中的 Automation->getStrings(...)方法,获得对应的字符串。通过对 SQL 代码进行解析，我们希望获得 SQL 语句中每个字段及其所在的表。在 3.1 节中，我们通过 E-R 模型定义了数据库中数据语义。这里，利用嵌入式 SQL 和程序间的数据依赖关系，把数据语义传递到程序中。对 SQL 代码解析这一步骤可以通过对 SQL 语句静态的文本分析并结合数据库定义文件获得。构建 SQL 解析树如图 3 所示：

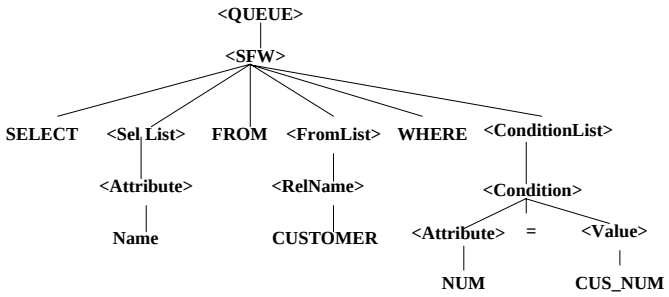

图 3　SQL 解析树示例

程序和 DBMS 间的数据交换主要是通过宿主变量(host variables)来实现的。DBMS 通过输出宿主变量(output host variables)传递数据和状态到源程序。源程序通过输入宿主变量（input host variables）把数据传递给 DBMS。通过对 SQL 的解析，发现输入宿主变量的过程比较简单。

在确定了输入宿主变量后，我们下一步的任务是发现输出宿主变量。这一过程便是从数据库交互的节点出发发现数据依赖，确定对应关系的过程,这一流程如下：

对 SDG 图中的每个 DBAN: DBAccessNode
计算解析 SQL 语句
添加 DBAN 对数据库表的数据依赖
确定 SDG 图上，所有依赖于 DBAN 的语句节点
对于每个上述节点，设为 n,
　　If n 调用了 rg: Resultset->get*(...)方法
　　　　根据 rg 方法的参数查找其对应的 SPT 节点
　　　　查找数据库数据语义定义
　　　　标注 rg->creturn_out
　　If n 调用了 ps: PreparedStatement->set*(...)方法
　　　　根据 set 方法的参数查找其对应的 SPT 节点
　　　　查找数据库数据语义定义
　　　　标注 ps->creturn_out
　　设 m 为语句 n 所在的方法，增加 m 对数据库的依赖信息

在上面的步骤中，我们标注了直接对应与数据库操作的节点。这里注意到，上述过程中的宿主输出变量，并没有直接对应到源代码中存在的变量，而是对应到 SDG 图上的 return_out 节点。在有些情况下，Resultset->get*(...)方法没有直接的宿主变量与其对应，例如，语句：

Total = rs.getInt(“amount”) * rs.getDouble(“unit_price”);

其中，rs（Resultset 类的一个实例）为某个 SQL 语句查询返回的结果集，amount、unit_price 分别对应 SQL 中查询的所返回的某个属性值。此时，为了确定对应关系，通常需要创建临时节点，并增加依赖边来表示。由于 SDG 本身包含了这样的节点，可以避免了另外创建临时节点的过程。

在完成以上步骤之后，SDG 图上包含了一些标注了数据语义的节点。接下来便是把相关的节点也进行标注。过程如下：对于每个已经进行标注的节点 n，查找 n 所依赖的节点，和依赖于 n 的节点，对应已定义数据间关系，标注相应节点。对于相互之间直接赋值的情况，语义的传递过程是直接的。对于，通过运算才得到的情况，则要依靠已定义的数据间的关系获得。至此，我们得到了一个标注数据语义的 SDG 图。

4 系统实现

我们使用基于 Soot Analysis Framework[13]获得系统的数据流和控制流图，并构造 SDG。在使用 JSA 库对数据库访问方法传入的 SQL 参数值进行计算前，我们首先对 Java 源程序进行插装,通过 Java 的反射机制可以获得用户输入变量（方法参数）的名称，把变量的名称以字符串的形式和原变量值进行连接。图 4 中给出了一个代码插装的示例。

在获得进行交互的 SQL 语句后，使用 JavaCC 构造 SQL 的解析树，并通过简单的搜索把查询列和数据库定义文件中的列进行匹配以获得对应的表，并在 SDG 图上标注和传递数据语义。

```
public String getPassword(String id) {
    String sqlQuery =
      "SELECT password FROM users WHERE id = '"
       + id + "' ";
    Statement stmt = dbConn.createStatement();
    ResultSet rs =  stmt.executeQuery(sqlQuery);
    if(rs != null) return "";
    else return rs.getString("password");
}
```

```
public String getPassword(String id) {
    id = " id " + id;
    String sqlQuery =
      "SELECT password FROM users WHERE id = '"
       + id + "' ";
    Statement stmt = dbConn.createStatement();
    ResultSet rs =  stmt.executeQuery(sqlQuery);
    if(rs != null) return "";
    else return rs.getString("password");
}
```

图 4　代码插装示例

5　总结

本文从结合数据库数据知识和源代码理解的角度出发提出了一种基于数据库模式的数据语义逆向恢复方法。其目的是更好的理解以数据为中心的遗产系统，恢复其源代码中的数据资产，在此同时，利用这些包含有数据语义信息的源代码，方便对于系统结构的理解和进行再工程。我们的方法仍有不少地方需要进一步的研究和完善。在接下来的研究工作中，主要有以下需要解决的问题或者进一步的研究方向：1)

针对调用式的数据库访问方式，进行数据库和源代码之间交互点的发现和分析。此时的交互点将是调用 DAM 的点，而非直接的数据库请求点；2)构建的数据库数据模型语义不够完备，程序中仍存在着许多无法关联的变量。在数据库数据模型的基础上，增加相关的领域知识模型，可以更好的对源代码进行理解。

参 考 文 献

[1] K. H. Davis and P. H. Aiken. Data Reverse Engineering: A Historical Survey [J]. Seventh Working Conference on Reverse Engineering ,2000

[2] Dowming Yeh , Yuwen Li. Extracting Entity Relationship Diagram from a Table-Based Legacy Database [J]. Ninth European Conference on Software Maintenance and Reengineering, 2005

[3] Alhajj, R. Extracting the extended entity-relationship model from a legacy relational database [J]. Information Systems, Elsevier Science Ltd., 2002, pp.597-618

[4] Cleve, A. Henrard, J. Hainaut, J.-L. Data Reverse Engineering using System Dependency Graphs[J]. Working Conference on Reverse Engineering 2006

[5] Von Mayrhauser A., Vans A. M. Program Understanding -- A Survey. Technical Report CS-94-120, Colorado State University, Computer Science Department, 1994

[6] Young P. Program Comprehension. Technical report, Center for Software Maintenance, 1996

[7] Rugaber S. Program Comprehension. Technical report, Georgia Institute of Technology, 1995

[8] S. Horwitz, T. Reps, D. Binkley. Interprocedural slicing using dependence graphs[J]. In Proceedings of the ACM SIGPLAN8, 1988

[9] Karl J. Ottenstein, Linda M. Otenstein. The program dependence graph in a software development environment[J]. ACM SIGPLAN Notices,1984, volume19(5)

[10]L.Larsen , M. Harrold, Slicing object oriented software[J]. in 18th International Conference on SoftwareEngineering, pp. 495－505, 1996.

[11]N. Walkinshaw, M. Roper, M. Wood. The Java system dependence graph[J]. Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation.20 03, 55 -6

[12]Java String Analysis http://www.brics.dk/JSA/.

[13]Soot Analysis Framework http://www.sable.mcgill.ca/soot/