Java 组件脆弱性数据库的设计与实现①

李远玲，刘 强，刘 丽，陈 华

(北京系统工程研究所, 北京 100101)

摘 要：JPA 作为 Java 的持久化解决方案能够直接将 Java 对象映射到关系数据库.主要论述了 Java 组件脆弱性数据库的结构、Java 组件数据的获取方式，以及应用 JPA 实现组件数据库的方法，重点论述了针对组件数据的分层结构，利用面向对象的持久化技术批量存储组件数据的过程。

关键词：组件脆弱性; Java 组件; 组件数据库; JPA; Soot

Design and Implementation of Java Component Vulnerability Database

LI Yuan-Ling,LIU Qiang,LIU Li,CHEN Hua

(Beijing Institute of System Engineering, Beijing 100101, China)

Abstract：JPA, as a persistence solution for Java, can directly map objects to relational databases. In this paper, we present the structure of Java component vulnerability database, the methods to acquire Java component data and vulnerability information, and methods to achieve the component database using JPA, especially procedures storing the component data based on hierarchical structure for the component data.

Key words：component vulnerability; java component; component database; JPA; Soot

软件脆弱性分析技术是计算机安全领域重要的研究课题之一。随着软件复杂度和规模的不断扩大，大型软件的脆弱性分析已成为热点。从软件结构上看，由于应用程序构建在平台架构之上，常常需要引用第三方函数库的功能，单纯分析应用程序的脆弱性会使分析结果不准确，因此需要分析运行环境和第三方函数库的脆弱性，这就大大增加了软件脆弱性分析的规模，并由此产生了巨大的数据量，如何有效地存储分析过程数据和结果数据成为软件脆弱性分析需要重点考虑的问题之一。

基于组件的脆弱性分析[1]是软件脆弱性分析的一个思路。组件是具有特定功能且可以独立开发并部署的基本程序单元，可以按照数据传播的脆弱性模式对组件进行分析以获取组件脆弱性信息。本文以 Java 软件作为分析对象，Java 类是 Java 虚拟机加载的基本单位，可将其作为组件的基本单位。方法是特定功能的实现，对外表现为 API 接口，对方法的分析可采用基于控制流、数据流等经典理论的控制依赖分析、数据传播分析、类型分析、调用分析等静态分析技术。

Java 组件脆弱性分析系统的核心部分由组件分析器、漏洞检测器和数据存储器构成。存储器使用数据库技术实现，称之为 Java 组件脆弱性数据库。数据库中包含组件的解析结果和脆弱性分析结果，实现了组件解析和脆弱性分析结果向组件知识库的转化，进而可实现结果数据的查询、统计和利用。组件脆弱性分析系统以 Java 项目（如 JBoss）、Java 包、类和方法为分析对象，分析过程分为三个步骤：步骤一是解析。为了能对无源代码 Java 程序做分析，系统以 soot 提供的解析功能作为分析前端，实现 java 字节码到用于程序分析的中间表示的转换[2]，输出 Java 项目、Java 包、Java 类和方法的解析结果。步骤二是污染传播分析。首先，在中间表示结果的基础上按照预设的规则标识方法的污染入口、出口和风险调用。然后以污染入口为起点使用控制流和数据流分析等多种静态分析技术

① 收稿时间:2011-11-23;收到修改稿时间:2011-12-21

分析得到体现污染数据在方法内部传播关系的方法污染传播图。接下来，通过对污染传播图的遍历，得到了方法的风险传播和传播效能，这两种关系分别体现了污染入口到风险调用之间传播关系和污染入口到污染出口之间的传播关系。最后，对控制流图和污染传播图进行图形化输出[3]，以方便分析人员对传播关系的查看。步骤三是存储和输出，将解析结果和风险传播等摘要信息以一对多的层次关系存储在 Java 组件脆弱性数据库中，并且进行抽取生成分析报告。

面向对象的软件开发是当今应用开发环境中最流行的开发方法，关系数据库则是应用系统中广泛使用的数据存储方式。JPA（Java Persistence API）是一个简单的实体持久化程序设计模式，它的 ORM（Object-Relational Mapping）框架能够直接将 Java 对象映射到关系数据库。原型系统使用 Java 开发，并使用 JPA 实现数据的持久化。本文主要介绍 Java 组件脆弱性数据库的结构设计，Java 组件数据获取方式，以及使用 JPA 构建组件数据库的过程和方法。

1 JPA框架简介

目前，使用 Java 语言编写数据库管理软件一般都采用被称为对象关系映射的数据持久化技术。对象关系映射是针对 JDBC 不能直接持久化 Java 对象的解决方案。一旦将对象和关系型数据库关联起来，操作对象就可以自动操作数据库，直接持久化复杂的 Java 对象。

现在存在一些成熟的 ORM 框架，如 Hibernate、Toplink 等，但它们没有统一的标准，框架之间的 API 存在着差异。JPA 规范解决了这些 ORM 框架之间不能够兼容的问题。JPA 是 Java EE 5.0 平台标准的 ORM 规范，也是 EJB 3.0 的重要组成部分。它是非侵入式的，很容易与其它框架和容器集成。JPA 作为对象持久化的标准，不但可以在 EJB 3.0 容器中应用，也可以脱离容器直接在标准的 Java 平台(Java SE)中使用。本文主要介绍在 Java SE 中的使用。

JPA 框架下的持久化实体是一个轻量级的 Java class，它的状态被持久化到关系型数据库的一个表中，一个实体的各个实例对应于表中的不同行。与数据库表相一致，实体与实体之间同样存在着关联关系，而这些关联关系可通过对象/关系元数据描述。

在 JPA 框架下创建实体和创建 Java 类一样简单，没有任何约束和限制，只需使用 javax.persistence。Entity 进行注释。

综合起来，JPA 框架包含三个方面的技术规范[4]：

1) ORM 映射元数据。JPA 支持 XML 和 JDK5.0 注解两种元数据形式，元数据描述对象和表之间的映射关系，框架据此将实体对象持久化到数据库表中;

2) JPA 的 API 用来操作实体对象，执行 CRUD（创建，查询，更新，删除）操作。框架在后台替开发者完成所有的事情，使开发者从繁琐的 JDBC 代码中解放出来。

3) JPA 自身定义的查询语言 JPQL（Java Persistence Query Language）。它是一种针对实体的查询语言，面向对象而非面向数据库，避免程序的 SQL 语句紧密耦合。

JPA 框架也支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，这些支持能够让开发者最大限度地使用面向对象的模型设计企业应用，而不需要自行处理这些特性在关系数据库中的持久化。

为了使用 JPA 操作一个数据库，需要用到一个实体管理器（Entity Manager），实体管理器可以执行针对数据库中实体的持久化、删除和查询等操作。实体管理器又通过实体管理器工厂（Entity Manager Factory）创建。实体管理工厂使用 javax.persistence.Persistence 类获取。它读取 META-INF 目录下 persistence。xml 文件中的持久化配置信息，创建命名的实体管理工厂。实体管理工厂发现并处理存储在 XML 文件中的或在程序代码中注解的持久化元数据。脆弱性数据库管理系统的开发仅使用在程序代码中注解持久化元数据的方式。图 1 表示这一过程[5]。

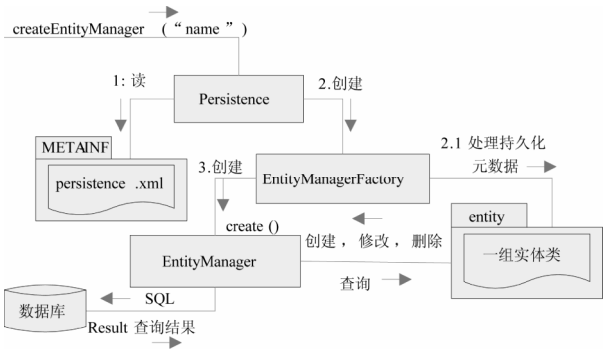

图 1 持久化序列

从图中也可以看出实体管理器作为 Java 实体对象

与数据库交互的中介，一方面管理一组实体并完成实体的 CRUD 操作; 另一方面将面向 Java 实体对象的操作转化为数据库识别的 SQL 语句以实现实体的持久化以及将返回的查询结果组装成实体。

2　Soot简介

Soot 是 Java 字节码分析工具[6]。它提供了多种字节码分析和变换功能，利用它可进行程序的分析、编译优化和变换。这里主要介绍 Soot 作为类库提供的 java 字节码相关分析功能。

Soot 有一个复杂的类层次结构[2,3]，它的应用程序接口所涉及的类包括 Scene、SootClass、SootMethod（类中的方法）、SootField（类中的域）、Body（方法体）等。Scene 类表示分析场景，它是一个数据大仓库，存储分析过程中的全局变量，包括预设的全局参数、分析过程中不断充实的类继承关系、调用关系等与全局相关的信息，以及当前被分析类的中间结果信息等。SootClass 表示由 Soot 解析或创建的单个 java 类对象。一个 SootClass 包含一个 Java 类的相关信息，如这个类的类名、修饰符、超类、SootField 链、SootMethod 链等。Scene 中的所有类都是 SootClass 类的实例。

Soot 对 Java 类的解析过程从 SootClass 的加载开始，代码如下所示：

Scence。v().loadClassAndSupport(className);

加载后对 SootClass 抽丝剥茧解析出更加细粒度的信息。例如使用 SootClass 的 getMethods()方法可得到该类所有 SootMethod 的列表，通过 SootMethod 的 retrieveActiveBody()可得到该方法的 Body 对象，之后再对 Body 对象进一步分析。

Java 组件的解析过程如下所示：

1) 通过 java.Util.jar.jarFile 从 jar 包中获取要分析的类文件；

2) 通过 Scence.v().loadClassAndSupport(className)构造类相关的 SootClass 对象;

3) 通过 SootClass、SootMethods 和 SootFields 获取相关的类、方法和域的信息。

表一列举了 Soot 对象包含的部分方法。

利用 Soot 对象的方法对 Java 类进行解析，将解析的结果数据以及解析过程中的统计数据存储在数据库中，以便进一步实现组件脆弱性分析。

表 1　Soot 对象主要方法说明

对象	包含的方法	说明
SootClass	isInterface(), isAbstract()等	判断这个类是否是接口、抽象类等
	getInterfaceCount() getFieldCount()	得到这个类直接实现的接口的个数、域的个数等
	getMethods()	得到这个类所有方法的对象
SootMethod	getName()	得到方法名
	isNative() isAbstract()等	判断这个方法是否是本地方法、抽象方法等
	getReturnType()	得到方法的返回类型

3　Java组件脆弱性数据库简介及结构设计

Java 组件脆弱性数据库用于存储 Java 组件解析结果数据和组件脆弱性分析结果数据。Java 组件解析后不仅数据量大，而且数据之间存在着复杂的关联关系。为实现数据的有效存储和合理利用，本系统从数据的性质出发，以分层方式构造数据的存储结构，以实体间双向映射策略实现数据间的映射关系。力求做到数据结构清晰，数据处理效率高。

Java 组件的解析以项目为单位，一个项目包含若干个 Jar 包，每个 Jar 包里有若干个类，每个类有若干个方法。数据存储同样以项目为单位，在数据库中建立项目与 Jar 包、Jar 包与类、类与方法之间的映射关系，以便有效地实现统计和分析工作。图 2 是 Java 组件脆弱性数据库的层次关系图。

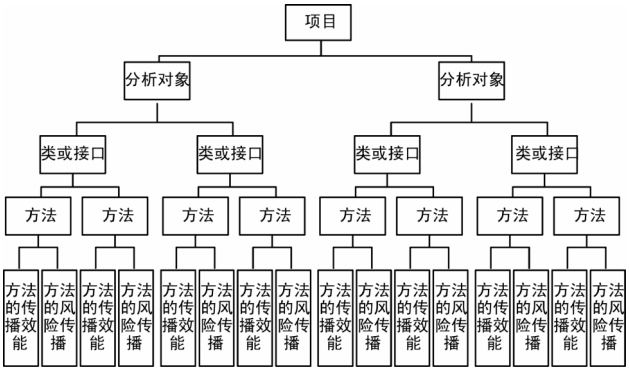

图 2　组件数据库实体之间层次关系图

在数据库中这些节点的信息存储在六个数据库表

中，它们是项目表、分析对象表、类列表、方法表、方法的风险传播表和方法的传播效能表。

由于一个项目包含多个 Jar 包，因此项目实体和 Jar 包实体是一对多的关系，通过项目可以检索到该项目下的所有 Jar 包信息。通过项目实体和 Jar 包实体在数据库中定义的关联关系，也可以反过来确定某个 Jar 包属于哪个项目。同样，Jar 包和类、类和方法、方法和方法的风险传播、方法和方法的传播效能也都是一对多的关系。图 3 是这些表之间的调用关系图。

脆弱性分析主要针对方法进行。分析结果包括：方法的控制流图信息、方法内污染传播信息、方法间污染传播信息，以及该方法的风险传播信息和方法的传播效能信息。用户在查询方法污染传播信息的同时，也可以通过这些调用关系，方便地查询到方法所隶属的类、Jar 包和项目。

通过合理地构造数据库表的调用关系，可以有效地提高数据的存储效率、减少数据存储的冗余度。

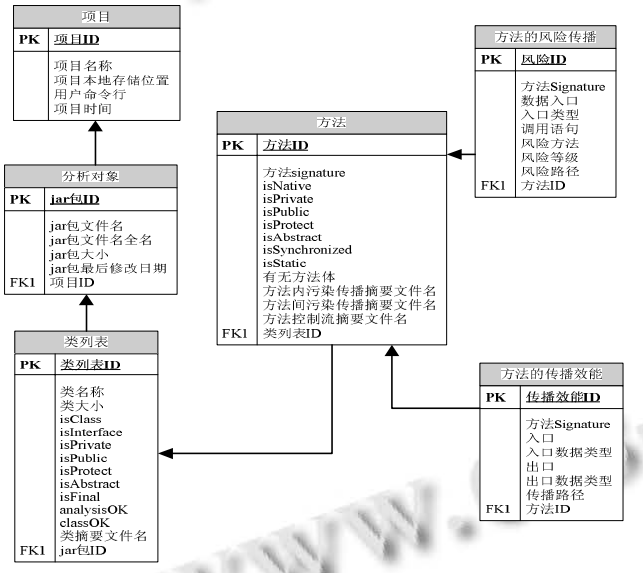

图 3 组件脆弱性数据库实体之间的调用关系图

4 JPA框架在组件脆弱性数据库中的应用

这一节将描述实现组件脆弱性数据库所涉及的持久化过程。首先描述与组件脆弱性数据库相关的持久化上下文，包括如何从 Java SE 创建实体管理工厂和实体管理器，以及持久化上下文与事务的关系。随后说明实体类与组件脆弱性数据库中各实体间的映射关系，以及为实现这些映射关系所提供的方法。最后详细叙述为实现数据库的数据关联所设计的数据存储过程。

4.1 编写持久化上下文

实体管理器是 JPA 重要的组成部分，任何对实体的操作都是通过它完成。实体管理器作为 Java 实体对象与数据库交互的中介，负责一组实体的创建、读取、更新和删除。它的具体功能和作用体现在以下两个方面：(1) 负责将对 Java 实体对象的操作转化成数据库识别的 SQL 脚本，以便实现实体的持久化。(2) 负责将针对实体进行查询的 JPQL（Java Persistence QL）语句转化成 SQL 脚本，并将返回的结果组装成实体。

所有应用程序管理的实体管理器对象都是通过实体管理器工厂对象创建的。EntityManagerFactory 对象通过 javax.Persistence.Persistence 类的静态方法 createEntityManagerFactory 来创建。下面的代码是从实体管理工厂对象获得实体管理器对象的过程。

EntityManagerFactory Emf = javax. Persistence. Persistence.createEntityManagerFactory("dataflowPersistenceUnitForUpdate");

EntityManager em = emf.createEntityManager();

一个实体管理器的配置与创建它的实体管理器工厂绑定在一起，它被定义为一个持久化单元（Persistence Unit）。在一个持久化单元里，将指定这个被创建的实体管理器管理的实体类、数据库定义和一个特定的持久化供应商。JPA 在 persistence.xml 中定义持久化单元的配置。persistence.xml 配置代码如图 4 所示：

```
<persistence-unit name="dataflowPersistenceUnitForUpdate"
                  transaction-type="RESOURCE_LOCAL">
  <class>entity.MethodInformation</class>
    ……
  <properties>
    <property name="hibernate.connection.driver_class"
              value="com.mysql.jdbc.Driver" />
    <property name="hibernate.connection.url"
              value="jdbc:mysql://192.168.40.201:3306/javaVulDB/>
    <property name="hibernate.connection.username" value="vuldb" />
      <property name="hibernate.connection.password" value="vuldb" />
      <property name="hibernate.dialect"
              value="org.hibernate.dialect.MySQLDialect" />
      <property name="hibernate.hbm2ddl.auto" value="update" />
  </properties>
</persistence-unit>
```

图 4 持久化单元的配置内容

每个持久化单元都会被命名。在一个 persistence.xml 文件中可以包含多个已命名的持久化单元，它们包含不同的配置。应用可根据不同的要求，通过名字引用不同的持久化单元配置。在上面的代码中，<persistence></ persistence >对中可由一个或多个<persistence-unit>对组成。每对< persistence-unit >中包

含一个持久化单元配置。name 元素指的就是持久化单元名字，也就是在创建实体管理工厂时指定的名字。我们这里用“dataflowPersistenceUnitForUpdate”作为名字。transaction-type 表示实体管理器对象的事务管理方式。脆弱性数据库管理系统的开发是在 J2SE 环境中进行的，而在 J2SE 环境中只能使用 RESOURCE_LOCAL 管理实体管理器事务。class 元素列出了持久化单元的所有实体，这是在 J2SE 环境中满足可移植性必须做的。另外，在 Java SE 环境中必须指定登录到数据库的参数，在<properties></properties>对中是一列供应商专有的成员属性，这些属性值指出数据库的连接方式。

4.2 定义持久化实体类

Java Persistence API 对象都是 POJO（Plain Old Java Object，简单 Java 对象），任何现有的应用对象都能够被持久化。

如前所述，Java Persistence API 的对象关系映射完全是由元数据驱动的，每个实体都与一组元数据相关联，这些元数据（重点使用默认值，坚持尽可能少的原则）用来描述实体。持久层通过元数据识别、解析和管理实体。实体元数据可以通过两种方式指定：(1) 在代码中增加注解。(2) 使用外部定义的 XML。注解元数据是一种语言特性，它将结构化和类型化的元数据附着在源代码上。

本项目定义了六个实体类，用于存放项目、jar 包、类、方法、方法的传播效能和方法风险传播的分析结果。以类实体为例，图 5 的代码片段说明了数据库中类实体的元数据。

```
@Entity
@Table(name = "classList")
public class ClassInformation {
  @Id
  @GeneratedValue
  @Column(nullable= false)
  int classID;
  @Column(length=256,nullable= false)
  private String className;
  ……
  @OneToMany(mappedBy="upclass")
  private Collection<MethodInformation> methods;
  @ManyToOne(optional = true)
  private JarFileInformation upfile;
  ……
  public Collection<MethodInformation> getMethods() {
    return methods;
  }
  public void appendMethod(final MethodInformation method){
    //将方法实体对象添加到本类实体的一对多方法实体集合中
   getMethods().add(method);
    // 并将方法实体指向类的设置为本实体
   method.setUpclass(this);
  } ……
}
```

图 5　类实体的注解元数据

注解元数据@Entity 标注这个类是一个可持久化的实体类。注解@Table(name = “classList”)表示该实体所对应的数据库表名为“ classList ”。如果没有这个注解，数据库表名的默认值就是实体类的绝对名字。另一个注解元数据也是必须的，它就是@Id，它将属性 classID 标记为实体的主键，一个实体类至少要有一个主键。其余属性如果不标注注解，将使用默认值。注解@OneToMany 说明了类实体与方法实体是一对多的关系，属性 mappedBy 说明了这是一个双向关联关系。多对一端为持有端，定义连接字段；一对多端是反向端，指定 mappedBy 元素。注解@ManyToOne 说明了类到 Jar 包是多对一的关系，类是多端，是持有端，定义了连接字段 upfile。

实体类中有许多方法定义了各个属性 set 和 get 的方法，由于篇幅的原因在上述代码段中被省略了，只是保留了操作一对多和多对一关系的代码段。appendMethod()方法首先得到该类所拥有的方法实体集合链表的指针，然后将新分析的方法实体添加到集合链表中，这样就建立了类到方法的一对多关系。反过来，方法到类是多对一的关系，在方法实体中有一个连接字段 upclasss，保存方法到类的连接关系，通过语句 method.setUpclass(this)，就可在方法实体中设置与当前类的连接。

```
em.getTransaction().begin();开启事务
分析项目;创建项目实体并设置项目实体的各个属性;获得该项目的所有jar文件
for(一个jar文件 : jar文件列表) {
    分析jar文件;创建jar文件实体并设置该实体的属性;
    将jar文件实体存入实体列表jarEntities;
    获取jar包中所有的类
    for(一个class文件 : class文件列表) {
        分析解析类文件;创建类文件实体并设置该实体的属性;
        将类文件实体存入实体列表classEntities;
        获取类文件中所有的方法
        for(一个method文件 : method文件列表) {
            ⑴分析解析该方法;创建方法文件实体并设置该实体的属性;
            将方法文件实体存入实体列表methodEntities
            ⑵分析该方法的传播效能;将该方法若干传播效能实体实例存入列表
            performanceEntities;将该方法的所有传播效能实体加入到该方法实体的
            一对多关系链中;将这个方法的所有传播效能实体持久化到数据库中
            ⑶分析该方法的风险传播;将该方法若干风险传播实体实例存入列表
            riskEntities;将该方法的所有风险传播实体加入到该方法实体的一对多
            关系链中;将这个方法的所有风险传播实体持久化到数据库中
        }
        将该类的所有方法实体加入到该类实体的一对多关系链中;
        将该类的所有方法实体持久化到数据库中。
    }
 将该jar包的所有类实体加入到该jar实体的一对多关系链中;
  将该jar的所有类实体持久化到数据库中.
}
将该项目实体持久化到数据库中
em.getTransation().commi();提交事务
```

图 6　分析数据的存储过程

4.3 数据存储的逻辑过程

数据存储过程的设计涉及数据的逻辑关系和数据结构，不仅要考虑数据的关联，还要考虑减少数据的冗余度。组件数据库各实体之间相互关联，以分层的方式存储在数据库中，因此数据存储过程采用嵌套的方式，图 6 的代码段描述这一存储过程。

在 JPA 中影响数据库中数据变化的操作必须使用事务管理保证其一致性。实体的 CRUD 基本操作的“C”、“U”、“D”需要事务管理，这样保证一旦操作失败，所有操作将回滚到初始状态，操作成功才完成提交，数据库内容才会被真正更新。J2SE 开发环境采用 RESOURCE_LOCAL 方式管理事务，该方式通过调用 EntityManager 的 getTransaction()方法获得本地事务对象。开始一个新事务，使用对象的 begin()方法；事务完成后，使用 commit()方法提交。

上述代码首先对 jar 文件进行分析，分析采用 java 本身提供的类，通过该内部类获取 jar 包中所有类文件。然后使用 java 字节码分析工具 Soot 对类和方法进行分析解析。

分析和存储的过程是一个多层嵌套的过程。比如，分析一个类文件时，首先分析该类的所有方法，将每个方法的实体对象先存储在链表 methodEntities 中，待所有方法分析完毕才会通过 em.persist()语句分别将所有方法实体持久化到数据库中。可见持久化过程是一个逆序过程，首先持久化方法实体，然后持久化类实体，再持久化 Jar 文件实体，最后再持久化项目实体。为了保证数据的完整性，只是在整个项目数据的持久化过程完成之后，才提交给数据库完成真正的数据持久化。通过这种多层嵌套方式也可以在分析存储过程中实现数据的关联关系。

由于实际分析中一次提交的存储数据量很大，为了尽可能地避免数据存储过程中 Java 虚拟机的堆空间产生 OutOfMemoryError，实际原型系统中利用两种方法解决问题：一是在 java 虚拟机启动时利用-Xms 和-Xmx 参数给 Java 堆尺寸指定一个较大的数值。二是通过分析将存储过程中后续不再使用的 JPA 实体的实例显式置 null，并通过编写的内存监控程序不断监视内存剩余情况，按照策略通过显式的调用 System.gc()来回收内存垃圾。

由于数据采用分层结构进行存储，因此数据的浏览可采用树形结构。项目名称位于树形结构的第一层，项目的子节点是 jar 包名称，jar 包的子节点是类，类的子节点是方法。在树形结构中点击相应的节点，可以获取该节点的完整信息，其中包括方法的风险传播信息等。分层存储结构也便于实现数据的统计和查询。比如可以统计某一个 jar 中包含 native 方法的数目以及通过查询获取某一方法的控制流图等。

5 结语

数据的组织不仅与数据本身的性质有关，而且与数据的使用密切相关。Java 组件分析结果的数据量大，特别是 JBoss 和 JDK 这些项目分析结果的数据量更大，而且这些数据需要以项目为单位一次性录入数据库并构造其相互关系。因此不仅需要设计良好的数据结构，而且需要有效的方法实现数据的存储过程，以实现数据的完整性和可用性。

参考文献

1 Zhao G, Chen H, Wang DX.Data-flow Based Analysis of Java Bytecode Vulnerability. The Ninth International Conference on Web-Age Information Management.647–653.

2 Einarsson A, Dam J, Brics N. A Survivor's Guide to Java Program Analysis with Soot. [2008-07-17].http://www. brics.dk/SootGuide/sootsurvivorsguide.pdf.

3 李远玲,陈华,刘丽.Soot 的 Java 程序控制流分析及图形化输出.计算机系统应用,2009,18(10):88–92.

4 冯曼菲,等.EJB JPA 数据库持久层开发实践详解.北京:电子工业出版社,2008.3–12.

5 JPA Tutorial.http://schuchert.wikispaces.com/JPA+Tutorial+1

6 Soot:a Java Optimization Framework. [2010-03-29].http://www.sable.mcgill.\ca/soot/