Web工程中基于不变性的元数据检查和测试

傅 腾 高建华

(上海师范大学计算机科学与工程系 上海 200234)

摘 要 元数据在Web工程中起到了十分重要的作用。随着元数据规模的增加,对元数据的维护会花费很多的时间和精力。而目前的编译器不能对元数据不一致所导致的错误进行提示,也不能罗列出元数据和代码之间隐藏的关系。通过实验,使用基于框架和框架无关两种方式来发现元数据不变性,研究并验证了元数据不变性。当用户重构或者增强程序时,元数据不变性会被检查,如果违反了不变性,则对用户进行提示。

关键词 元数据,不变性,Java,软件测试,软件错误

中图法分类号 TP302.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.048

Metadata Checking and Testing of Web Application Based on Invariance

FU Teng GAO Jian-hua

(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)

Abstract Metadata plays a very important role in Web application. With the increasing scale of metadata, maintaining metadata will spend intensive time and effort. The current compiler cannot notify the faults caused by metadata inconsistency, and the relationship between metadata and codebase is hidden. This paper extracted all the invariance of the projects based on the research regarding with the metadata invariant by experiments. It illustrated that the metadata invariants are respectively extracted and compared through such different methods as framework-based invariant discovering and frameless-based invariant discovering. When the program is refactored or enhanced, the metadata invariance will be determined. A message will be given if the programs violate invariance.

Keywords Metadata, Invariance, Java, Software testing, Software bugs

元数据(metadata)的运用已经成为当今Web工程中不可或缺的部分,起到了配置文件和管理项目之间依赖等作用。随着业务逻辑的扩展和工程规模的增加,项目结构变得纷繁复杂,程序员往往要花很多时间和精力维护项目中的元数据。而传统的语法或错误检查工具不能自动识别元数据导致的错误。

Ali Mesbah, Arie van Deurse和Danny Roest[2]利用元数据进行了Web工程的自动化测试。弗吉尼亚理工大学的M. Song和E. Tilevich[1]提出了一种DSL(领域专用语言)来检查元数据的正确性,这种DSL名为MIL(Metadate Invariants Language)。该方法能够检查元数据造成的错误,但缺少一定的扩展性。在国内,主要倾向于研究元数据的应用[3],对于元数据的正确性,没有很多的研究。

本文提出了一种新的方式来提取工程中的元数据不变性,主要展开以下5个方面的工作:(1)提出了基于不变性的元数据的检查和测试方法来检测和测试工程中的元数据,以求增强系统对于代码修改的健壮性;(2)使用基于框架和框架无关两种方式寻找元数据不变性;(3)检测因元数据导致的错误,提高可靠性;(4)自动向程序员提示元数据的错误,使其能减少因疏忽遗漏而导致的错误;(5)最后,通过对开源工程的测试,验证本文方法的有效性。

本文第1节介绍了相关的背景知识;第2节对XML的依赖关系进行分类;第3节定义及描述了不变性;第4节介绍了元数据不变性程序的实现方法;第5节是元数据不变性程序的实验分析并说明了本文研究的有效性;最后给出总结并提出展望。

1 相关知识

1.1 元数据

元数据(metadata)是一种记录了数据的数据,在很多领域都有着重要的作用,如航天、医疗、机械等[6]。在计算机领域,Web工程中的元数据有json、XML、annotation等。本文研究的元数据为XML。

XML全名为可扩展标记语言,XML可以用来定义数据和存储数据,是一种可以让用户自定义标签及其内容的源语言。在Web工程中,XML文件可以用来配置文件或者文件之间的依赖关系,几乎所有的Java开源框架都可以用XML或者json来配置。尽管如此,但当XML内容改变时,没有成

到稿日期:2013-10-21 返修日期:2013-12-23 本文受国家自然科学基金项目(61073163),上海市引进技术的吸收与创新年度计划项目(12CH-19)资助。

傅 腾(1990—),男,硕士生,主要研究方向为软件可靠性设计理论与方法,E-mail:liok3187@gmail.com;高建华(1963—),男,博士,教授,CCF会员,主要研究方向为软件可靠性理论与设计、软件开发环境与开发技术、数据安全与计算机安全、网络测试、LSI/VLSI测试等。

熟的工具可以对 XML 及其它的依赖文件进行校验和测试。本文研究如何读取 XML 的内容及其相关的文件，记录它们的不变性，从而在 XML 文件或其关联文件改变时，可以进行校验。

1.2 元数据的不变性

不变性是指程序中一些恒定不变的关系。举例来说，一个加法函数，其功能为输入两个浮点数，相加后输出浮点数的结果。对于这个函数，输入和输出的参数为浮点型，这是一个不变性；输出的结果为输入结果的和，这也是一个不变性。有了不变性的关系后，就可以根据这些规律或规则进行下一步的操作[1]。

不变性常用来检测程序的正确性、效率等问题。几乎从高级语言发明之初，不变性的概念就已经存在了，最早的相关研究始于 1991 年，当时 J. D. Fokker，H. Zantema 和 S. D. Swierstra[7]提出了用不变性来测试程序的正确性。本文主要研究了与元数据相关的不变性，先找出元数据相关的不变性，然后对代码进行检查和测试，并列举出元数据之间的关系[3]。

1.3 Web 工程中元数据的应用

随着计算机和信息技术的发展，现代企业级的信息系统渐渐从 C/S 结构转化为 B/S 结构，如 J2EE 是一个常用的实现方法。但当工程变得复杂和庞大后，软件开发团队往往会面临一些问题：任务重，时间紧；需求变化频繁；软件后续维护困难；人员流动等。为了解决这些问题，需要使用一些通用的特性：Java 的 EJB(Enterprise Java Bean)、JSP、XML、Servlet 等技术；也可以使用一些框架，比如：JUnit、hibernate、Spring 等，这些特性和框架提供了比较成熟的解决方案或开发工具，可以提高工程的效率和质量。随着这些特性和框架的使用，项目中会包含很多元数据，对这些元数据的管理和维护会存在很大的困难。本文从元数据的不变性出发，研究了一种可以检查和测试元数据的方法。

2 XML 依赖关系分类

2.1 按类型分类

XML 文件中的配置可以按照类型分为以下几类：

第一类(文件的对应关系)。在 XML 的配置中，一个节点的配置可以针对于一个单独的文件。图 1 为一段配置文件，是对 js 脚本文件的依赖关系：

```
〈PackageFile path="editor/js/fckeditorcode_ie. js"〉
    〈File path="editor/_source/fckconstants. js"/〉
    〈File path="editor/_source/fckjscoreextensions. js"/〉
    〈File path="editor/_source/internals/fckplugins. js"/〉
〈PackageFile/〉
```

图 1 含有 js 文件的 XML 片段

XML 配置文件中的依赖关系可以是内容简单的单个文件，如图 1 所示的 js 文件，也可以是打包过的 jar 包。这类文件一般只对文件名有着依赖关系，对文件中的内容没有依赖关系，所以只需要对文件的路径以及名字建立起对应的映射关系。

第二类(类的对应关系)。这里的类特指以. java 结尾的文件，这类文件有其严格的规范，可以通过 Java 的反射机制把它转化为树状的结构，以提取其中的内容。在 XML 的映射中，最关心的就是这类 Java 的源文件。

从图 2 的 struts2 的配置文件中可以看到这个依赖名叫做“nameRequest”。当从 jsp 页面上进行 nameRequest 的请求时，编译器会从 struts2 的配置文件中寻找这个依赖，然后根据这个依赖，在源文件下的 app/action/目录中寻找 registerAction. java 类，如果该类名找不到对应的 Java 类，则会报错。

```
〈action name="nameRequest" method="nameExcute"
    class="app. action. RegisterAction"〉
    〈result type="json"〉〈/result〉
〈/action〉
```

图 2 struts2 框架配置片段

当重构 registerAction. java 的命名时，如果将 registerAction. java 改成 registerUserAction，往往会忘记修改配置文件中的 class="app. action. registerAction"，而只有到了运行时，这个错误才会被发现。对此可以通过元数据依赖检测程序来记录程序中的元数据依赖，在修改或增加代码时，一旦违反了元数据的依赖关系，就会在控制台进行提示。

第三类(方法、域、属性的依赖关系)。这三者也是元数据中可能出现的依赖关系上文已经提到 XML 中的依赖关系对应了 app/action/目录中的 registerAction. java 类，更进一步来说，通过 method="nameExecute"这个配置，使 Java 文件对应了 registerAction. java 下的 nameExecute 方法。因为一个类中会有多个方法，当需要指定具体调用的方法时，就需要增加这个依赖关系。方法、域和属性属于同一个级别下的依赖关系，在一个类下可以有多个方法、域或者属性。在描述元数据的依赖关系时，类包含了方法、域和属性，而方法、域和属性是并列的，它们之间也可能有关系。

2.2 按重要性分类

XML 文件中的依赖关系可以按其重要性分为需要记录和无需记录两类。前文提到的依赖关系都是配置文件和 Java 文件(或其他类型文件)的对应关系，当其中一个修改或重构时，很有可能导致依赖关系的改变，从而导致程序的错误，这类关系需要记录；而形如图 3 所示的 XML 配置则无需记录。图 3 所示的依赖关系为 maven 的依赖配置，它也是一种对应关系，但与项目的源文件并无关系，maven 框架也可以自动检测和更新这类依赖关系，所以这类依赖关系属于无需记录的关系。

```
〈dependency〉
    〈groupId〉com. google. inject. extensions〈/groupId〉
    〈artifactId〉guice-persist〈/artifactId〉
    〈version〉3. 0〈/version〉
〈/dependency〉
```

图 3 maven 框架配置片段

3 不变性的定义及描述

3.1 框架无关的元数据不变性

3.1.1 框架无关的元数据不变性分析

如果用户没有对框架进行配置，或者元数据并非在框架

中，可以使用框架无关的元数据发现的方法来寻找元数据不变性，图 4 是普通元数据的片段。

```
〈Templates imagesBasePath="fck_template/images/"〉
    〈Template title="Image and Title"
        image="template1. gif"〉
        〈Description〉One main image with a title and
            that surround the image. 〈/Description〉
        〈Html〉
            〈h3〉Type the title here〈/h3〉
            type the text here]]〉
        〈/Html〉
    〈/Template〉
⋮
```

图 4　普通元数据片段

对于一般的元数据，它的不变性与标签并没有直接关系，因为 XML 文件中的标签是固定的，一般不会发生变动，本文所关心的是标签中的属性和标签里的内容；同样地，本文也不比较属性名，判断的是属性中的值。

3.1.2　元数据不变性符号定义

表 1 为元数据符号定义。

表 1　元数据不变性符号定义

符号	含义	符号	含义
S	源程序	c	类
X	XML 文件	m	方法
L	XML 文件中的标签〈L〉…〈/L〉	a	属性
L(a)	标签 L 中的属性	p	参数
L(v)	标签 L 中的值	Pf	文件 f 的路径
F	java 文件		

3.1.3　框架无关的元数据不变性

(1)*Class*(X,F)：表示对于一个属性 a(或者值 v)，如果匹配文件的类名和路径，则结果为 1，否则为 0。

(2)*MethodDistance*(X,F)：表示 X 中的一组属性$\{A_1,A_2,\cdots,A_n\}$(或一组值$\{T_1,T_2,\cdots,T_m\}$)与 F 中的一组方法 $C\{c_1,c_2,\cdots,c_m\}$的平均距离。

(3)*FieldDistance*(X,F)：表示 X 中的一组属性$\{A_1,A_2,\cdots,A_n\}$(或一组值$\{T_1,T_2,\cdots,T_m\}$)与 F 中的一组属性 $A\{a_1,a_2,\cdots,a_m\}$的平均距离。

(4) *ConDistance*(X,F)：表示 X 中的一组属性$\{A_1,A_2,\cdots,A_n\}$(或一组值$\{T_1,T_2,\cdots,T_m\}$)与 F 中的一组参数 $P\{p_1,p_2,\cdots,p_m\}$的平均距离。

(5)*Match*(X,Y)：表示如果 X 与 Y 相匹配结果为 1，否则结果为 0。

一组属性$\{A_1,A_2,\cdots,A_n\}$和一组方法 $C\{c_1,c_2,\cdots,c_m\}$(或属性 $A\{a_1,a_2,\cdots,a_m\}$或参数 $P\{p_1,p_2,\cdots,p_m\}$)的平均距离记作 Dis，表示为：$Dis=\frac{\sum_{i=1}^{n}Match(A_n,C)}{n}$。

本文把 XML 文件和其他文件的相似度记作 Sim，表示为：

$$Sim=Class(X,F)+MethodDistane(X,F)+FieldDistance(X,F)+ConDistance(X,F)$$

在元数据不变性发现程序中，用户可以给定一个相似度阈值，如果相似度高于此值，可以认为这是一个元数据不变性，应该记录这个关系和具体的信息。

3.2　基于框架的元数据不变性

3.2.1　基于框架的元数据不变性定义

一个 XML 文件 X 含有一组标签〈L〉…〈/L〉，当这组标签中的属性 $L(a)$与一个 java 文件 F 中的类 c(或方法 m，或参数 p，或属性 a)有着一一对应的关系时，本文称 X 中的 L 和 F 中的 c 有着元数据不变性，并将这种关系记作 $X_{L(a)}\leftrightarrow F_c$，同时记录路径 P_X,P_L,P_F,P_c。

3.2.2　基于框架的元数据不变性描述

如果 java 文件 F 和元数据 X 存在不变性，则满足：

$$\exists F\in S$$
$$\exists F_c\leftrightarrow X_L \text{ or}$$
$$\exists[F_m|F_p|F_a]\leftrightarrow X_L$$

其中，$F_m,F_p,F_a\in F_c$。

文件 F 是源代码库 S 里的一个文件，若存在一个文件中的类 F_c 与元数据 X 中的标签 L 有着元数据不变性，或者文件中的类 F_c 下的方法 F_m 或参数 F_p 或属性 F_a 与元数据 X 中的标签 L 有着元数据不变性，则元数据不变性成立。

对于图 3 的配置文件，struts2 的不变性可以描述为 $F_c\leftrightarrow X_{L(a)}$ 和 $F_m\leftrightarrow X_L$(其中 $F_m\in F_c$)。具体可以写成：

$$\text{RegisterAction}_{\text{public class RegisterAction}}\leftrightarrow$$
$$\text{Action}_{\text{class="app. action. RegisterAction"}}$$
$$\text{RegisterAction}_{\text{public String nameExecute()}}\leftrightarrow$$
$$\text{Action}_{\text{method="nameExecute"}}$$

4　元数据不变性实现

4.1　元数据不变性活动图

元数据不变性的实现如图 5 所示，具体步骤如下：

(1)在 eclipse 里导入一个 Web 工程；

(2)运行“元数据检测”程序，获得此工程里的所有元数据的不变式；

(3)对不变式进行检测，排除无用的关系，确认有用的关系并保存；

(4)运行“查看元数据关系”程序，查看此工程中的元数据依赖关系；

(5)当对程序进行改动时，“元数据依赖检查”程序判断不变性是否被破坏，如果不变性不成立，则对用户进行提示。

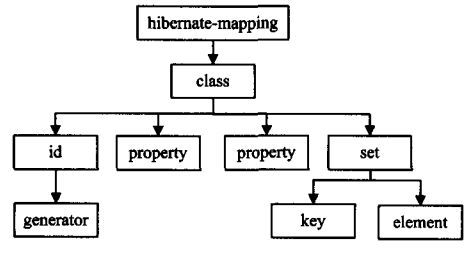

图 5　元数据不变性活动图

4.1.1　元数据不变性发现

元数据检测可以检测出工程中所有的元数据并记录下

来。元数据不变性程序首先找出工程中所有的 XML 文件，再提取这些文件中可能存在的元数据不变性，然后通过与其他文件进行比较和判断，确定元数据的不变性。本文使用基于框架和框架无关两种方式提取元数据不变性，其中基于框架的不变性选用的框架为 Hibernate。如果需要添加其他的基于框架的元数据不变性，可以通过继承和接口进行扩展。

4.1.2 元数据不变性确认

在记录了元数据的不变性后，需要确认该元数据不变性的正确性。遍历得到的所有元数据不变性，包括一般的和特定框架下的元数据不变性。如果一个元数据或者代码库中的其他文件不止一次出现，则说明该元数据与对应的其他文件并非是一一对应的关系，则这个元数据不变性是错误的，必须把这些关系从元数据不变性中删除。

4.1.3 元数据不变性检查

在记录了元数据的不变性后，如果新的代码对程序进行了修改或者重构，并且破坏了元数据的不变性，则元数据不变性程序会进行提示。程序员可以自行判断其破坏元数据不变性的行为是错误的还是想要的结果，如果是错误的，则可以根据提示来修改程序。

4.2 元数据不变性 UML 类图

图 6 给出了元数据不变性的 UML 类图。

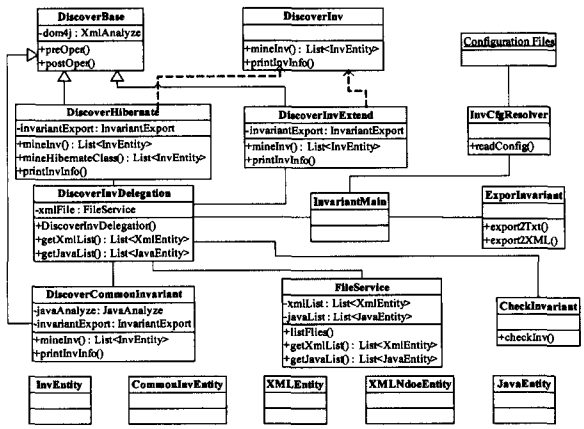

图 6 元数据不变性 UML 类图

(1)Entity 实体

图 6 中的 3 个实体分别为记录 XML、Java 和不变性的实体。XML 和 Java 的实体包含了一个文件的名字、完整路径和子节点等信息。一个文件就是一个实体，而不变性实体则是为了记录不变性，一个不变性的关系为一个实体。不变性实体间存在着包含的关系，比如一个类的不变性中包含了方法的不变性；一个 XML 或者 Java 文件可能含有多个不变性，对应了多个不变性实体。

(2)元数据发现代理

图 6 中的 DiscoverInvDelegtion 是元数据发现代理，该代理对于用户是透明的。对元数据进行配置后，元数据代理类就会根据配置信息挖掘元数据的不变性。每种元数据都继承了一个基类，实现了一个接口。在基类中，抽象出了元数据发现前后需要的两个方法，即准备元数据和元数据后续处理；同时也要实现接口，在接口中可以具体地定义第一种元数据的发现方法。本文的程序基于面向对象的设计模式，如果需要增加一个元数据的不变性，只需要继承基类实现接口就可以进行扩展，不会改变原来的代码。

5 实验与讨论

本文的实验验证基于框架和框架无关的元数据的检查和测试方法的有效性，选取了开源 Web 工程 myblog，重点关注以下两个问题：

(1)本文的元数据检查和测试程序能否有效地找出元数据不变性？

(2)基于框架和框架无关的不变性提取的准确性对比。

5.1 实验准备

实验的环境如表 2 所列。该项目为一个开源的 Java 软件 myblog，其中包含了 31 个 XML 文件和 130 个 Java 文件。实验主要从两个角度发现和测试元数据的不变性：基于框架的和框架无关的。实验对应的框架采用了 Hibernate，它是一个数据库对象关系映射的框架。图 7 上是 Hibernate 配置文件的片段，图 7 下对应了 java 程序的片段。如基于 Hibernate 来发现不变性，类名 Comment 存在不变性；属性 authorName 也存在着不变性；如果用框架无关的方式发现不变性，则根据前文提到的算法来计算 XML 文件和 Java 文件之间的相似度，如果该文件相似度大于相似度阈值，则记录对应的 XML 文件和 Java 文件，再通过扫描代码库来确认这个元数据的不变性。

表 2 实验环境

使用软件	eclipse
开源项目	myblog
使用语言	Java
jdk 版本	1.6.0
项目使用框架	Hibernate
Xml 文件数	31
Java 文件数	130

```
〈compass-core-mapping package="com. jdkcn. domain"〉
〈class name="Comment" alias="comment"〉
    〈property name="authorName"〉
        〈meta-data〉authorName〈/meta-data〉
    〈/property〉
```

```
public class Comment extends BaseDomain{
    private String authorName;
    public String getAuthorName(){
        return authorName;}
    public void setAuthorName(String authorName){
        this. anthorName = authorName;}
}
```

图 7 Hibernate 配置片段和 Java 程序片段

5.2 实验结果

5.2.1 Hibernate 框架的元数据发现

若寻找基于 Hibernate 框架的不变性，共计可以找到 48 种不变性，如表 3 所列。使用了 Hibernate 配置的 Web 工程，可以找出所有的基于 Hibernate 的不变性。如果这些不变性

发生了改变，元数据的检查和测试程序就会提醒程序员，元数据的不变性被破坏了，此时程序员可以选择修改程序或忽略提示。

表 3 Hibernate 实验结果

	类	方法	属性	总计
不变性	12	12	24	48

5.2.2 框架无关的元数据发现

以 Comment. cpm. xml 为例，经过框架无关的元数据发现程序寻找不变性后，结果如表 4 所列。随着相似度阈值的增加，寻找到的相关的文件数会不断减少。当不限相似度时，有 46 个相关的文件；当相似度减少到 1.5 的时候，相关的文件减少到 3 个；其中，相似度最大的文件是 Comment. java，其相似度为 2.38。

表 4 框架无关的不变性单个实验结果

相似度阈值	不限	>0.1	>1.2	>1.5	>2
候选文件数	46	29	13	3	2

手工查看 Hibernate 的配置，我们发现 Comment. cpm. xml 的确是对应了 Comment. java。这两个文件存在着元数据不变性。从实验中可知，对于 Comment. cpm. xml 元数据，与其相似度最大的 Java 文件有元数据不变性。下面我们通过全部元数据文件相似度的发现与比较来验证：与元数据相似度最大的 Java 文件是否有着元数据不变性。

5.2.3 总体实验结果

表 5 为框架无关的元数据发现实验发现的相似度个数。

表 5 框架无关的元数据相似度数量

	类	方法	属性	参数
无限制的相似度	559	2056	608	652

可以看出框架无关的元数据发现方法，从单独的类或方法、属性、参数上来看，获取的不变性过多，有些并不是有效的。对于框架无关的元数据，需要结合所有的类、方法、属性和参数，同时利用阈值来决定不变性，不变性的结果与给定的阈值有较大的关系。

选取相似度最大的文件，调整相似度取值，框架无关的元数据发现程序获得的结果如表 6 所列。

表 6 框架无关的元数据发现总结果

阈值	不限	>1	>1.2	>1.5	>2
不变性	32	24	21	14	2
准确率	37.5%	50%	57.1%	85.7%	100%

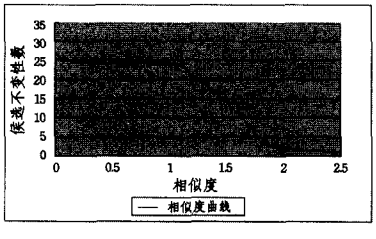

图 8 框架无关的元数据发现相似度曲线

框架无关的元数据发现程序扫描了项目中所有的元数据，计算了该元数据与其他文件的相似度，选取相似度最大的文件作为元数据不变性的候选。从实验结果中可以看出，当相似度取到 2 的时候，准确率为 100%，但是过大的相似度取值会遗漏很多元数据不变性。对于本次实验，为了取得高的准确率，逐步增加相似度的阈值，从表 6 中可以看出，在相似度为 1.5 时，效果最佳。该阈值发现了所有的不变性，准确率也达到了 85.7%。选取阈值时，若增加了很小的相似度阈值，而获取的元数据不变性骤减，则该拐点为最佳的阈值取值。也可以从图 8 所示的框架无关的元数据发现相似度曲线中看出，当相似度为 1.5 的时候，斜率最大，则该点为最佳取值。

在了解使用框架的情况下，采用基于框架的元数据不变性发现，效果较好。但是在不使用框架，或者对于框架不了解的情况下，采用框架无关的元数据发现就能挖掘出工程中的元数据不变性。

结束语 本文重点从 3 个方面研究了 Web 工程中的元数据不变性：(1)对元数据的特性进行了归纳和抽象；(2)研究和设计了元数据分析和提取算法的方法和流程，把元数据不变性的提取分为基于框架的和框架无关的两类；(3)给出了算法的核心思想，分析了核心的框架。

经过初步实验，表明了算法的适用性，展示了元数据不变性程序的功能，列举出了工程中元数据的依赖关系。日后的研究方向为：更准确地提取出元数据不变性；针对不同框架，尽可能抽象出相同的功能，使扩展需要的代价最小化。

参 考 文 献

[1] Song M, Tilevich E. Metadata Invariants: Checking and Inferring Metadata Coding Conventions[C]// IEEE International Conference on Software Engineering. Zurich, Switzerland, 2012: 694-704

[2] Mesbah A, van Deursen A, Roest D. Invariant-Based Automatic Testing of Modern Web Applications[J]. IEEE Trans-actions on Software Engineering, 2012, 38(1): 35-53

[3] 潘有能，滕海明. 基于语义标记树的 XML 文档聚类研究[J]. 情报学报，2012，31(5)：508-514

[4] Costa G, Ortale R. Structure-oriented clustering of xml documents: A transactional approach[C]// IEEE International Conference on Intelligent Systemsm. Sofia, 2012: 188-193

[5] Posonia A M, Jyothi V L. Structural-based Clustering Technique of XML Documents[C]// International Conference on Power and Computing Technologies. Nagercoil, 2013: 1239-1242

[6] 查礼. 基于 Hadoop 的大数据计算技术[J]. 科研信息化技术与应用，2012，3(6)：26-33

[7] Fokker J D, Zantema H, Swierstra S D. Iteratie en invariatie [M]. Programmeren en Correctheid, Academic Service, 1991