Web 应用中冗余代码检测方法研究

胡营营,赵逢禹
(上海理工大学 光电信息与计算机工程学院,上海 200093)

摘 要:为了提高 Web 开发效率,开发人员常常复用已有系统框架或成熟项目中现有的代码,但因此也导致了 Web 应用中总存在大量的冗余代码,冗余代码不仅影响程序的可读性和运行效率同时还会隐藏软件缺陷。通过研究 Web 应用源代码逻辑和框架的特性,提出了 Web 应用系统中基于源代码分析的冗余代码检测方法。从应用程序入口开始,根据代码之间的逻辑调用关系构建 Web 应用调用树,进而得到有效页面集、有效类与方法节点集;然后根据冗余检测算法检测出 Web 应用系统中冗余页面、冗余处理类与处理方法。为了评估冗余检测方法的有效性,包括漏检率与误检率,对两个 JavaWeb 应用进行冗余检测并通过人工注入冗余实验验证检测的有效性。实验结果证明,提出的冗余代码检测方法可以达到较高的检测效率。

关键词:Web 应用;冗余代码;抽象语法树;Web 应用调用树
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2020)03-0030-06
doi:10.3969/j.issn.1673-629X.2020.03.006

Research on Redundant Code Detection Method in Web Application

HU Ying-ying,ZHAO Feng-yu
(School of Optical-Electrical and Computer Engineering,University of Shanghai for Science and Technology,Shanghai 200093,China)

Abstract:In order to improve the efficiency of Web development,developers often reuse previous code in existing system frameworks or mature projects,but it also leads to a large amount of redundant code in Web applications. Redundant code not only affects the readability and operational efficiency of the program,but also hides software defects. We propose a redundant code detection method based on source code analysis in Web application system by studying the characteristics of web application source code logic and framework. The method constructs the Web application call tree according to the logical call relationship between the codes starting from the application entry,and then obtains the effective page set,class and method node set. A redundancy detection algorithm is presented to detect the redundant pages,the redundant service classes and the processing methods. In order to evaluate the effectiveness of the redundant detection method,including the missed detection rate and the false detection rate,two JavaWeb applications are used as the experiment to verify the redundant code detection method proposed,and the results show that the redundant detection efficiency is high.

Key words:Web application;redundancy code;abstract syntax tree;application call tree

0 引 言

为了快速开发 Web 应用系统,开发人员常常复用已有系统的框架或成熟项目中现有的代码,然后在此基础上进行完善与修改。这种代码与框架复用能够提高开发效率,节省开发时间,但同时也带来了诸多的副作用。一是代码冗余增加,即项目中存在冗余的页面代码、JavaScript 代码、CSS 代码、处理方法、控制类、支持类等;二是导致源代码臃肿,可读性差,增加了维护难度。在某些情况下,Web 应用系统中的冗余代码还会隐藏软件缺陷与安全漏洞[1-3],最近一项对 9 300 名开发人员进行的调查结果显示将冗余代码检测和代码拆分评为最高级别功能请求。

冗余检测一直受到国内外学者的关注。王伟使用 Lex 和 YACC 分别对 C 语言代码进行词法和语法分析,通过语法树检测代码中的幂等冗余、变量冗余和死代码冗余[4];苏小红等人利用 TOKEN 序列建立复合语句控制结构信息表,设计了基于控制结构的冗余代码检测模型,能对 C 语言中幂等、隐幂等、私有变量、

收稿日期:2019-03-14 **修回日期**:2019-07-18 **网络出版时间**:2019-11-07
基金项目:国家自然基金青年基金项目(61402288)
作者简介:胡营营(1993-),女,硕士,研究方向为软件工程;赵逢禹,博士,教授,研究方向为软件工程与软件质量控制。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20191107.0918.062.html

条件冗余、死代码和冗余传参进行检测[5-6];寿能为了揭示冗余与软件缺陷的关系,在冗余分类的基础上,研究了冗余特征与软件缺陷的关联关系,使用 NRefactory 设计了冗余检测算法,实现了一个冗余检测与缺陷提示的检测器[7];Wang Xing 结合静态切片和动态切片分析方法提出了一种基于程序切片的死代码检测方法,并在 LLVM 基础上设计了死代码检测框架[8];Leitao 通过对 C 语言代码构建抽象语法树,设计了 Retargetable Redundancy and Deceit Detector (R2D2),通过分析语法树中的子节点来检测冗余代码[9]。

Niels 针对 Web 中代码复用带来浏览器解析多余 JavaScript 死代码造成 Web 应用系统的整体性能下降问题,提出了一种基于静态和动态分析的 JavaScript 死代码消除方法 Lacuna,在执行时间和分析精度方面取得了可喜的成果[10]。虽然 Niels 针对 JavaScript 中的冗余代码进行了检测,但 Web 应用系统中还包括多个层面的代码如浏览器端的表现层代码 HTML 和 CSS、服务器端处理与控制类代码、数据库操作与服务支持代码等[11],Niels 的研究尚无法推广到对整个 Web 应用系统的冗余检测。而实际上,前后台交互的业务处理类与处理方法是 Web 应用系统的核心逻辑,对该类代码进行冗余检测在提高系统的可读性、可维护性以及性能上有重要作用,因此文中把研究的重点集中在前后台交互的业务处理类与处理方法冗余检测方面。

Web 应用系统开发中常用前端开发语言有 HTML、Javascript、CSS、C#、Jquery 和 Bootstrap 等;常用后端开发语言为 Java、ASP. NET、PHP、Python 和 Ruby 等。尽管一个 Web 应用系统可能由不同语言甚至多种语言组合开发,但前后台仍然是通过业务处理类与处理方法进行交互,对该类冗余的检测仍然适用于所有 Web 应用系统,因此文中以广泛使用的 HTML、Javascript、CSS 和 Java 语言为例,给出了 Web 应用系统中基于源代码分析的冗余代码检测(redundant code detection for web application, RCDWA)方法。

1　相关技术及概念

在 RCDWA 方法中,需要获取 Web 应用中前后台交互的业务处理类与处理方法,即表现层和业务逻辑层功能节点业务跳转的关联关系,这需要用到抽象语法树以实现对相关节点的提取和源代码的解析。

1.1　抽象语法树

抽象语法树[12](abstract syntax tree, AST)是源代码的抽象语法结构的树状表现形式。树上的每个节点都表示源代码中的一种结构。之所以说语法是“抽象”的,是因为这里的语法并不会表示出真实语法中出现的每个细节,一些语句被隐含在树的结构中,并没有以节点的形式呈现。

采用 Eclipse JDT 中的静态解析技术将源代码文件转化为抽象语法树,通过操纵抽象语法树,一方面可以精确地获得 Web 应用源代码中的相关节点,进而实现对代码的解析和利用;另一方面在语法分析过程中编译器会对文法进行等价的转换如消除左递归、回溯、二义性等[13],这样会给文法引入多余的成分,抽象语法树的结构采用的是上下文无关文法即不依赖于源语言的文法,因此选用抽象语法树可以避免干扰因素,更好地对处理类与处理方法节点进行提取。

1.2　冗余处理类与处理方法定义

一个 Web 应用系统是由页面、后台处理逻辑、数据库系统组成的有联系的代码集合。Web 应用从入口页面即主页面开始执行并根据用户交互情况动态生成不同的显示层页面[14]。代码 1 和代码 2 中的代码分别为 Web 应用系统前台 HTML、Javascript、CSS 代码和后台的 Java 代码举例。代码 1 中超链接的 href 会链接到页面 next. jsp,Javascript 代码的 myfunction 函数调用代码 2 中的业务处理方法 function1,代码 2 中的 Controller 和 Redundant 为业务处理类。从程序入口开始遍历所有源代码文件,遍历时被调用到的页面、被调用的处理类与方法就是 Web 应用中有效的代码集。遍历完整个 Web 应用系统,如果某些页面、类与方法都没有被调用,那它们就是冗余页面、冗余业务处理类与处理方法。在代码 1 和代码 2 给出的例子中,代码 2 中 Redundant、function2 和 function3 都没有被调用,就是冗余类或冗余方法。

代码 1:Web 应用系统的入口文件 index. jsp。

```
1 <html >
2 <head>
3 <link type = " text/css"  href = " css/bootstrap. css" >
4 <title>主页</title>
5 <style type = " text/css" ></style>
6 </head>
7 <body>
8 <a href = " next. jsp"  target = " main" >next. jsp</a>
9 <script type = " text/javascript" >
10 function myfunction( ) {
11 formObject. action = " URL" ; }
12 </script>
13 </body>
14 </html>
```

代码 2:Web 后台的 Java 代码 Controller. java。

```
1 public class Controller {
2 @ RequestMapping( value = " / URL" )
3 public ModelAndView function1 (request) {
```

```
4 …
5 }
6 public ModelAndView function2 (request){
7 …
8 }
9 }
10 class Redundant {
11 public ModelAndView function3 (request){
12 …
13 }
14 }
```

2 RCDWA 冗余检测方法

为了分析 Web 应用系统中业务处理类与处理方法的冗余问题,需要分析前后台交互的业务处理类与处理方法。图 1 给出了该冗余检测问题的整体流程框架。

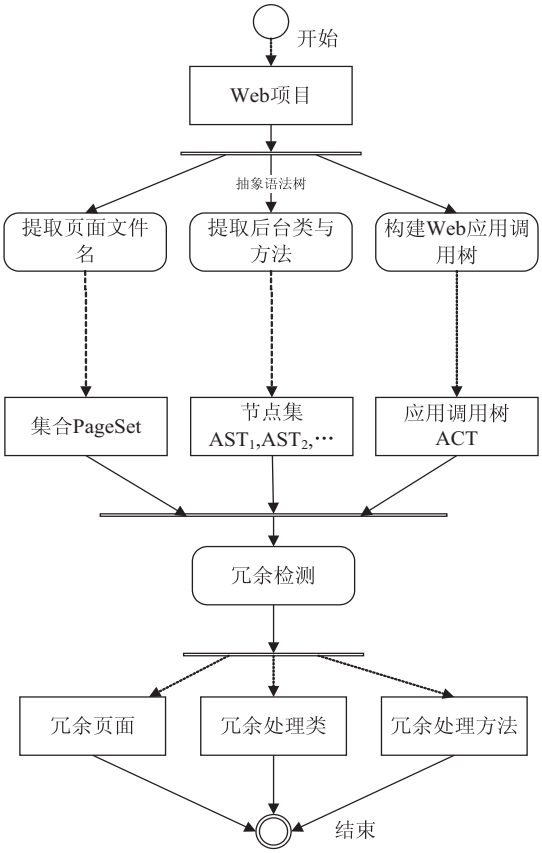

图 1 RCDWA 方法流程框架

(1)提取 Web 应用中的页面文件名,得到页面文件名集合 PageSet。在应用系统页面文件目录下新建一个. bat 文件,并且以编辑形式打开并输入代码@ ECHO OFF</回车>tree /F >页面文件名集合. txt,即可得到 PageSet 集合。

(2)提取后台文件中类和方法得到节点集合 AllSet = { AST_1 , AST_2 , … , AST_i … } ,其中节点 AST_i = <$Class_i$, $Method_{i1}$, $Method_{i2}$ … >, 这里 $Method_{ik}$ 是类 $Class_i$ 中的方法。算法 1 给出了获取集合 AllSet 的具体算法。

(3)搜索 Web 应用入口页面中对其他页面的调用或对后台类中方法的调用,递归构建 Web 应用调用树(application call tree,ACT)。算法 2 给出了构建 ACT 的详细过程。

(4)冗余检测。将集合 PageSet、AllSet 集合和 Web 应用调用树 ACT 作为输入进行冗余代码检测,输出冗余页面、冗余处理类和处理方法。算法 3 给出了冗余检测的具体过程。

3 RCDWA 冗余检测核心算法

在 RCDWA 方法中核心算法有节点构建算法、Web 应用调用树构建算法和冗余检测算法。节点构建算法利用源代码的抽象语法树查找 $Class_i$ 中的方法,得到节点 AST_i;调用树构建算法是为了构造出 Web 应用系统页面间、方法间、页面与方法间的调用关系;冗余检测算法利用 Web 应用调用树得到 Web 应用中有效的页面、处理类和处理方法节点集 ActivePage 和 ActiveSet,将有效节点集与 Web 应用中总的节点集 AllSet 对比来检测冗余代码。

3.1 节点 AST_i 构建

```
>TYPES(2)
>TypeDeclaration[158+99]
>type binding:cn. usst. market. controller. Controller
>BODY_DECLARATIONS(2)
>MethodDeclaration[186+65]
>method binding:Controller. function1( )
>MethodDeclaration[254+30]
>method binding:Controller. function2( )
>TypeDeclaration[290+49]
>type binding:cn. usst. market. controller. Redundant
>BODY_DECLARATIONS(1)
>MethodDeclaration[310+26]
>method binding:Redundant. function3( )
```

以上为 Controller. java 的抽象语法树,TYPES 为抽象语法树的根节点,TYPES(2)表示该属性下有两个 TypeDeclaration 子节点,TypeDeclaration 表示类声明或接口声明,在该树中表示类 Controller 和 Redundant;类声明的子节点 BodyDeclaration 表示类主体,即类大括号中的内容;BodyDeclaration 的子节点 MethodDeclaration 表示方法声明或构造器声明,在该树中代表方法 function1、function2 和 function3。将程序源码解析为抽象语法树 AST,生成的语法树可以方便地查找类与类中的方法,下面给出类与方法节点 AST_i 构建算法。

算法 1:节点构建算法。

输入:源码 code;

输出:AllSet。

处理:

(1)获取 Web 应用所有后台源代码文件。

(2)利用 Eclipse JDT 中的静态解析技术构建源代码文件类的抽象语法树 $Tree_i$。

(3)根据抽象语法树 $Tree_i$,查找类中的方法,并形成节点 $AST_i = <Class_i, Method_{i1}, Method_{i2}\cdots>$。

(4)返回节点 AST_i,并将节点添加到节点集合 AllSet 中。

利用节点构建算法处理后台源代码文件 Controller. java 可以得到集合 $AST_1 = <Controller, function1, function2>$ 和 $AST_2 = <Redundant, function3>$。

3.2　构建 Web 应用调用树 ACT

为了检测 Web 系统中从未调用的页面、处理类与处理方法,需要建立代码之间的逻辑调用关系,并基于代码间的调用关系构建 Web 应用调用树 ACT。Web 应用的主页面为 ACT 的根节点,主页面调用的其他页面 page(页面文件名)、调用的后台类中方法 Class. Method 都是该根节点的子节点。从程序入口页面开始将节点按调用关系逐步构建 Web 应用调用树,以便后文利用 Web 应用调用树来识别有效的页面、处理类与处理方法。

算法 2:Web 应用调用树构建算法。

输入:Web 应用入口页面文件;

输出:Web 应用调用树 ACT。

处理:

(1)将 Web 应用入口页面文件名作为树的根节点 Root,并标记该节点为“未访问”,得到初始 Web 应用调用树 ACT(每个 Web 应用系统有唯一入口页面)。

(2)采用广度优先算法,从 ACT 中获取第一个未访问的节点 Node,如果该节点是一个页面文件名,转步骤 3,如果该节点是一个方法,转步骤 4,如果该节点为 Null,则转步骤 5。

(3)搜索该页面中标签属性为 href 和 action 的节点作为 Node 节点的孩子节点,并将该 Node 节点标记为“已访问”,然后转步骤 2。

(4)利用抽象语法树查找 Node 节点调用的页面 page(页面文件名)与调用的方法 Class. Method,把这些节点作为该 Node 节点的孩子节点,并将该 Node 节点标记为“已访问”,然后转步骤 2。

(5)返回构建好的 ACT,结束算法。

Web 应用调用树是由 href 链接到的 page 和 action 逻辑跳转到的 Class. Method 2 种节点组成,如图 2 所示为前文代码 1 和代码 2 的 Web 应用调用树,根节点为入口文件 index. jsp,页面 index. jsp 中标签属性为 href 和 action 的节点分别调用页面 next. jsp 和后台方法 function1,依次递归直至构建为完整的 Web 应用调用树。

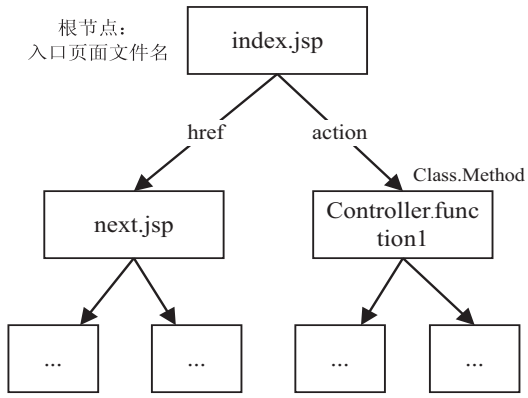

图 2　代码示例的 Web 应用调用树

3.3　冗余代码检测

将源程序中所有页面文件名存入集合 PageSet 中,到目前为止,已经获取集合 PageSet、集合 AllSet 和 Web 应用调用树 ACT,集合 PageSet 和 AllSet 中包含 Web 应用所有的页面文件、类和方法,ACT 中的节点为有效的页面文件、处理类和处理方法,通过计算可以得出冗余页面、冗余处理类与处理方法。

算法 3:冗余代码检测算法。

输入:PageSet、AllSet、ACT;

输出:冗余代码。

处理:

(1)设置有效的页面集合 ActivePage 与有效的方法集合 ActiveSet 为 Null。

(2)获取 ACT 中未访问过的节点 Node,如果 Node 为空,则转步骤 4。

(3)如果该节点是一个页面文件则存入集合 ActivePage 中,如果该节点是一个方法则存入集合 ActiveSet 中,然后把 Node 节点标记为“已访问”。转步骤 2。

(4)计算冗余页面集合 RedundantPage = PageSet – ActivePage,集合 RedundantPage 中节点对应的页面就是冗余页面。

(5)计算冗余方法集合 RedundantSet = AllSet – ActiveSet,集合 RedundantSet 中节点对应的方法就是冗余方法。

(6)如果一个类中的所有方法都是冗余方法,则该类记为冗余类

(7)输出代码冗余结果。

4　实　验

为了评估 RCDWA 方法的有效性,包括误检率和

漏检率,对两个 Web 应用系统进行了冗余检测,并对两个程序集分别进行不同数量的人工注入冗余实验。

4.1 实验对象

采用 UsstMarket 和 MovieBoot 两个 Web 应用进行实验分析,其中 UsstMarket 为笔者所在实验室开发的大型模拟创业网站,网站总共包括 6 个季度,模拟了现实世界创业中可能遇到的各种流程,目的是给各大高校学生当作一门创业课。MovieBoot 为 github 上的开源项目,是一个集电影、音乐和书籍于一体的 JavaWeb 应用,github 上显示最近一次代码更新是 1 个月前。两个 Web 应用都是由 HTML、JavaScript、CSS 和 Java 语言开发。表 1 为两个应用程序的基本信息。

表 1 程序集信息

程序	版本	文件数	总行数	Java 行数
UsstMarket	0.9.0	578	36 987	16 398
MovieBoot	2.1.0	224	14 718	9 765

4.2 实验过程与结果分析

实验 1:误检率检测。

利用 RCDWA 方法对 UsstMarket 和 MovieBoot 进行冗余检测,获取 Web 应用中的页面文件名集合,获取后台源代码文件并解析为抽象语法树得到类与方法节点集,然后构建 Web 应用调用树,进行冗余检测。

实验过程中两个程序集前台页面文件数、生成的抽象语法树 AST 个数、处理类与处理方法个数如表 2 所示。对检测出的页面冗余、类冗余和方法冗余进行了人工审查,发现确实是冗余代码,其中 UsstMarket 中的 49 个方法冗余是因为复用其它功能代码后需求改变所导致的冗余,2 个页面冗余是开发新页面时复用其它页面的代码导致的。

表 2 误检率检测结果

Web 应用	源程序中数量				检测出冗余数			误检率
	页面	AST	类	方法	页面	类	方法	
UsstMarket	121	249	339	1 674	7	23	147	0
MovieBoot	89	94	174	764	0	4	21	0

实验 2:漏检率检测。

为了统计漏检率,本文对两个 Web 应用进行了人工注入冗余实验,人工注入冗余时将冗余类和方法尽量分散到了不同文件里,表 3 列出了人工注入冗余页面、冗余处理类与处理方法的数量,注入冗余后利用 RCDWA 方法和文献[6][8]中的冗余代码检测方法对两个应用进行冗余检测,检测结果如表 3 所示,对 RCDWA 方法检测出的冗余进行人工审查,发现检测出总的冗余是人工注入冗余和实验 1 检测冗余结果之和。

表 3 漏检率检测结果

检测方法	Web 应用	人工注入冗余个数 M			检测出总冗余数			人工确认冗余数 N			$0 \leqslant$(漏检率 $(M-N)/M) \leqslant 1$		
		页面	类	方法	页面	类	方法	页面	类	方法	页面	类	方法
RCDWA	UsstMarket	30	95	200	37	118	347	37	118	347	0	0	0
	MovieBoot	20	75	100	20	79	121	20	79	121	0	0	0
RC-Finder	UsstMarket	30	95	200	0	36	41	0	36	41	-	0.62	0.79
	MovieBoot	20	75	100	0	13	21	0	13	21	-	0.82	0.79
DCDPS	UsstMarket	30	95	200	0	54	91	0	54	91	-	0.43	0.54
	MovieBoot	20	75	100	0	31	42	0	31	42	-	0.58	0.58

通过表 2 和表 3 可以看出,RCDWA 方法对两个应用程序冗余检测的误检率为 0%,对人工注入冗余的漏检率为 0%。针对实验 2 进行了多次试验,每次注入若干个页面、处理类与处理方法冗余,漏检率和误检率都为 0%。RC-Finder 和 DCDPS 方法对页面冗余检测数为 0 是因为文献[6]和文献[8]并未对页面冗余进行研究;对冗余类和冗余方法漏检率较高是因为 Web 应用使用面向对象语言开发,引入了封装、继承、多态等特性,文献[6]中 RC-Finder 方法不能完全适用于面向对象语言,文献[8]中 DCDPS 方法添加了动态分析技术,但由于动态程序切片的效率低导致只能分析较小的程序。从两个实验结果来看,文中提出的冗余代码检测方法针对 Web 应用系统中的页面冗余、处理类与处理方法冗余可以达到较高的检测效率。

5 结束语

冗余检测对 Web 应用系统的缺陷排除、系统维护有重要意义[15]。提出的基于源代码分析的冗余代码检测方法,从应用程序入口开始,根据代码之间的逻辑调用关系构建 Web 应用调用树,进而得到有效页面、类与方法节点集;然后将有效节点集与 Web 应用总的节点集根据冗余检测算法检测出冗余页面、冗余业务处理类与处理方法。基于该方法,对 UsstMarket 和 MovieBoot 两个中型 Web 应用进行了冗余检测实验,具有一定的代表性。尽管实验是针对 JavaWeb 应用系统进行的,但 RCDWA 方法完全适用于其他语言开发

的 Web 应用。

文中重点对 Web 应用系统中基于前后台交互的业务处理类、处理方法和页面进行冗余检测,对于 Web 应用系统中的其他冗余,如数据库操作冗余、CSS 冗余、Javascript 冗余等,并没有进行深入的研究,因此在后续工作中将进一步针对 Web 应用中的其他冗余进行研究。

参考文献:

[1] ENGLER D, XIE Y. Using redundancies to find errors[J]. IEEE Transactions on Software Engineering, 2003, 29(10): 915-928.

[2] 刘　伟,刘宏韬,胡志刚. 代码缺陷与代码味道的自动探测与优化研究[J]. 计算机应用研究,2014,31(1):170-176.

[3] 中国互联网协会. 互联网行业运行指数("网行指数")报告——中国网站[J]. 互联网天地,2018(2):18-25.

[4] 王　伟. C 冗余代码及相关缺陷检测方法研究[D]. 哈尔滨:哈尔滨工业大学,2010.

[5] 龚丹丹,王甜甜,苏小红,等. 冗余代码缺陷检测方法[J]. 哈尔滨工业大学学报,2012,44(7):58-63.

[6] GONG D, WANG T, SU X, et al. RC-finder: redundancy detection for large scale source code[C]//2012 second international conference on instrumentation, measurement, computer, communication and control. Harbin: IEEE, 2012: 243-248.

[7] 寿　能,赵逢禹. 基于 NRefactory 的冗余检测与缺陷研究[J]. 小型微型计算机系统,2015,36(9):1973-1976.

[8] WANG X, ZHANG Y, ZHAO L, et al. Dead code detection method based on program slicing[C]//2017 international conference on cyber-enabled distributed computing and knowledge discovery (CyberC). Guilin, China: IEEE, 2017.

[9] LEITAO A M. Detection of redundant code using R2D2[J]. Software Quality Journal, 2004, 12(4): 361-382.

[10] OBBINK N G, MALAVOLTA I, LUCA G, et al. An extensible approach for taming the challenges of JavaScript dead code elimination[C]//IEEE international conference on software analysis, evolution and reengineering. Antwerp, Belgium: IEEE, 2018: 291-401.

[11] TRIPP O, FERRARA P, PISTOIA M. Hybrid security analysis of web JavaScript code via dynamic partial evaluation[C]//Proceedings of the 2014 international symposium on software testing and analysis. [s. l.]: [s. n.], 2014: 49-59.

[12] TORRES R. Comparison of clang abstract syntax trees using string kernels[C]//2018 international conference on high performance computing & simulation (HPCS). Orléans, France: IEEE, 2018: 106-113.

[13] CODY M. Deep learning similarities from different representations of source code[C]//2018 ACM/IEEE 15th international conference on mining software repositories. Gothenburg, Sweden: IEEE, 2018: 542-553.

[14] KIRUTHIKA K. User experience design in web applications[C]//2016 IEEE international conference on computational science and engineering. Dubai: IEEE, 2016: 642-646.

[15] LIN B, PONZANELLI L, MOCCI A, et al. On the uniqueness of code redundancies[C]//2017 IEEE/ACM 25th international conference on program comprehension (ICPC). Buenos Aires: IEEE, 2017: 121-131.