ارائه یک روش جدید برای تحلیل ایستا برای یافتن استفاده ناامن از متغیرهای برنامه‌نویسی در برنامه‌های جاوا

سید محمدمهدی احمدپناه ۹۰۳۱۸۰۶

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر – تهران – ایران – اردیبهشت ۱۳۹۳

چکیده

زبان جاوا از جمله زبان‌های type safe به شمار می‌رود. به دلیل کاربرد زیاد این زبان در نوشتن برنامه‌های گوناگون، با ایجاد یک ابزار برای تحلیل ایستا در خصوص یافتن خطاهای برنامه‌نویسی از جمله استفاده غیرمطمئن از متغیرها و Exception Handling نادرست می‌توان از ایجاد اینگونه خطاها در زمان اجرا (Runtime) جلوگیری کرد. مزیت این کار این است که یافتن و اصلاح خطاهای برنامه نویسی در زمان اجرا هزینه‌بر و مشکل‌تر از قبل از زمان اجرا است. در موارد حساس مانند برنامه‌های نظامی و فضایی یا برنامه‌هایی که با جان و مال مردم مرتبط است، نمی‌توان خطایابی برنامه را در زمان اجرا انجام داد. در این گونه موارد است که اهمیت تحلیل ایستا بیش از پیش مشخص می‌شود. ابزار و برنامه‌های نسبتاً زیادی برای حل این مسئله ساخته شده‌اند که هر کدام مشکلاتی دارند که باعث می‌شود که برنامه‌نویس نتواند این اطمینان را پیدا کند که قبل از اجرای برنامه، همه خطاهای احتمالیِ کد نوشته شده کشف می‌شود. از مهم‌ترین خطاهایی که ابزارهای کنونی قادر به کشف آنها نیستند، می توان به خطاهای ناشی از Exception Handling نادرست اشاره کرد. در این روش می‌خواهیم با استفاده از مفاهیم صوری (Formal) نظیر استفاده از گراف و الگوریتم‌های مرتبط با آن و بررسی جریان داده‌ها و به خصوص بررسی خطاهای محتمل رایج در برنامه نوشته شده، به تشخیص خطاهای ناشی از استفاده غیرمطمئن از متغیرها به خصوص خطاهای مرتبط با Exception Handling نامناسب بپردازیم که قبل از زمان اجرا بتوان نشان داد که برنامه نوشته شده توسط برنامه‌نویس عاری از این گونه خطاها است.

کلمات کلیدی: کنترل جریان اطلاعات، تحلیل ایستا، مدیریت استثناها در جاوا، استفاده ناامن از متغیرها.

Keywords: Information Flow Control, Static Analysis, Exception Handling in Java, Unsafe Use of Variables

مقدمه

تولید نرم‌افزار مقاوم یک چالش همیشگی است. برای سادگی تولید نرم‌افزار مقاوم، زبان‌های برنامه‌نویسی، مانند c++ و جاوا، ساختار Exception Handling را فراهم کرده‌اند که به برنامه‌نویس این اجازه را می‌دهد که خطاهای احتمالی برنامه را راحت‌تر و سریع‌تر مدیریت کند. گرچه ساختار Exception تشخیص خطاها و شکست‌های برنامه را راحت‌تر و ساده‌تر می‌کند، ولی کنترل جریان تلویحی با استفاده از Exception را دشوارتر می‌کند.

در حالت کلی، برای یافتن و برطرف کردن خطاهای برنامه‌نویسی در زمان اجرا پرهزینه‌تر از همین‌کار در حالت ایستا است. چرا که برای این کار در زمان اجرا، باید به ازای هربار اجرای برنامه، برطرف کردن خطاها انجام شود که یعنی هزینه یکبار عیب‌یابی و برطرف کردن آن در تعداد بار اجرای برنامه ضرب می‌شود که در برنامه‌های پرکاربرد عدد بسیار بزرگی می‌شود؛ در حالی که در حالت ایستا، فقط یکبار این عیب‌یابی و بهبود آن انجام می‌شود و در مرحله اجرا نیازی به debug کردن نیست.

در سال‌های اخیر، ابزارها و تکنیک‌های زیادی برای یافتن خودکار bugها به‌وسیله تحلیل source code یا تحلیل استاتیک کد میانی ساخته شده است. خیلی از این تکنیک‌ها هم‌اکنون با کامپایلرها تجمیع شده‌اند. از جمله این ابزارها برای جاوا می‌توان به JLint و FindBugs اشاره کرد. به عنوان مثال، FindBugs از تکنیک‌های ad-hoc برای تعادل دقت، کارایی و قابلیت‌استفاده بهره می‌برد. اکثر این ابزارها bugهای معرفی شده برای ساختار Exception Handling مثل استفاده ناامن از متغیرها را در نظر نمی‌گیرد.

unsafe use of variables به معنای استفاده نامطمئن متغیرها است؛ یعنی برنامه‌نویس در طول برنامه ممکن است در قسمت‌های مختلف کد، از متغیری به نادرستی یا به‌طور ناامن استفاده کند که این ممکن است باعث پایان ناگهانی و غیرمنتظره برنامه در زمان اجرا شود. ناامنی می‌تواند به دلیل عدم رعایت قوانین type safety باشد. در مسیر اجرای برنامه، در صورتی که گزاره‌ای منجر به گرفتن یک مقدار از متغیر شود و قبل از آن حالت آن متغیر به دلیل رخداد Exception به درستی و با موفقیت تغییر نکند، استفاده ناامن تلقی می‌شود که می‌توان این حالت را با ارائه تعاریفی به طور فرمال تعریف کرد.

کارهای مرتبط

ابزارهای مختلفی برای یافتن اشکال‌ها به‌طور خودکار با استفاده از تحلیل ایستای کد برنامه یا کد میانی تولیده شده است. از جمله این ابزارها می‌توان به FindBugs، Jlint، PMD، ESC/Java2 و Bandera اشاره کرد.

اما آنچه در این‌جا مطرح است این است که همه ابزارها باید تعادلی بین یافتن اشکال‌های درست با تولید مثبت‌های غلط (هشدارهایی در خصوص کد درست) و منفی‌های غلط (عدم هشدار در خصوص کد نادرست) ایجاد کنند.

Bug Category	Example	ESC/Java	FindBugs	JLint	PMD
General	Null dereference	√*	√*	√*	√
Concurrency	Possible deadlock	√*	√	√*	√
Exceptions	Possible unexpected exception	√*			
Array	Length may be less than zero	√		√*	
Mathematics	Division by zero	√*		√	
Conditional, loop	Unreachable code due to constant guard		√		√*
String	Checking equality using == or !=		√	√*	√
Object overriding	Equal objects must have equal hashcodes		√*	√*	√*
I/O stream	Stream not closed on all paths		√*		
Unused or duplicate statement	Unused local variable		√		√*
Design	Should be a static inner class		√*		
Unnecessary statement	Unnecessary return statement				√*

√ - tool checks for bugs in this category * - tool checks for this specific example

معرفی روش پیشنهادی

هدف این است که به طور خودکار، مغایرت‌ها با امنیت وابستگی در برنامه‌های جاوا در حالت متغیر تشخیص داده شود. بعضی از شکست‌های برنامه، یافتن و ردیابی آن‌ها بسیار سخت است، چون ممکن است فقط در زمانی که واقعا رخ می‌دهند، قابل تشخیص باشند. ابتدا یک تعریف فرمال برای استفاده ناامن بر پایه گسترش عملگرها بر روی متغیرهای استفاده‌شده در تحلیل جریان داده مرسوم ارائه می‌شود. در تحلیل جریان داده مرسوم، عملگرها روی متغیرها به دو دسته تقسیم می‌شوند: define، use و kill

عملگرهای define، در گزاره‌هایی حالت متغیر را تغییر می‌دهند، رخ می‌دهند؛ مثل assignment.

عملگرهای use در گزاره‌هایی رخ می‌دهد که منجر به گرفتن یک مقدار از یک متغیر می‌شود.

عملگرهای kill وقتی رخ می‌دهد که متغیر آزاد شده است و دیگر قابل دسترسی نیست.

حالت متغیر ممکن است در هنگام رخ دادن Exception تغییر کند.

عملگرهای define به دو دسته زیر تقسیم می‌شوند:

۱) sDef؛ نوعی از عملگرهای define است که زمانی رخ می‌دهد که حالت متغیر به‌درستی و با موفقیت تغییر کند.

۲) eDef؛ نوعی از عملگرهای define است که زمانی رخ می‌دهد که حالت متغیر به دلیل وقوع Exception، با موفقیت و به درستی تغییر نکند.

در طول اجرا، عملگرهای define یا sDef یا eDef هستند.

استفاده ناامن، نوعی از عملگرهای use در مسیر اجرا است که با یک عملگر eDef در قبل ظاهر می‌شود و هیچ عملگر define دیگری بین این دو عملگر رخ نمی‌دهد. عملگر استفاده ناامن، باعث نقض ویژگی امنیت وابستگی می‌شود.

برای یافتن زوج‌های eDef در برنامه‌ها مانند تحلیل زوج‌های define-use مرسوم، ابتدا برای برنامه جاوا با ساختار Exception Handling، گراف کنترل جریان می‌سازیم و سپس زوج‌های eDef را با الگوریتم تحلیل جریان داده‌ی سلسله‌مراتبی تشخیص می‌دهیم.

```
Algorithm: detect unsafe use of variables for method M
Input:     Control flow graph of M
Output:    Nodes where unsafe use of variables occur

Begin
/* Step1: Generate variable set for the operator of sDef,
eDef, Use and Kill */
    For (each node in control flow graph){
        Divide each variable into the set of sDef, eDef,
      Use and Kill.
    }
/* Step2: Generate operation traces for each variable */
    Traverse the control flow graph to generate the
  operation traces for each variable;
/* Step3: Detect unsafe use on each trace */
    For (each trace) {
        Detect all appearance of EU pairs.
        For (each appearance of EU in the trace) {
            Locate unsafe use node in program;
    }}
End
```

نتایج

در این مقاله، روش مورد نظر تحت مثال‌هایی توضیح داده شده است. در واقع، این مثال‌ها به عنوان آزمایش‌های این مقاله هستند. آزمایش‌ها بر روی قسمت‌هایی از کد موجود در پکیج org.hsqldb.util.QueryTool و بخش‌هایی از کد پکیج com.daffodilwoods.daffodildb.server.datadiction arysystem.information از پکیج add(E e) و متد java.util.Vector صورت گرفته است. در واقع، این روش را می‌توان بر روی نرم‌افزارهای تشخیص اشکال نظیر FindBugs پیاده‌سازی کرد و بر روی آنها آزمایش کرد. در این مقاله، صرفا به ارائه روش و توضیح آن بسنده شده است که با استفاده از گراف کنترل جریان و مفاهیم تحلیل ایستا قابل بیان است و پیاده‌سازی برای آن انجام نشده است که می‌تواند به عنوان کارهای آینده آن نیز معرفی شود.

منابع

•N. Rutar, et al., "A Comparison of Bug Tools for Java", Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on, 2004, pp. 245-256.Static Detection of Unsafe Use of Variables in Java, Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC), pp. 439-443, 2010.

• W. Wosgerer, "A Survey of Static Program Analysis Techniques", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , Volume:27, Issue: 7 , 2005.

•N. Ayewah, et al., "Using Static Analysis to Find Bugs," Software, IEEE, vol. 25, pp. 22-29, 2008.