A dynamic instrumentation tool for obtaining software logs

To cite this article: Hongchen Zhou and Jing Li 2020 J. Phys.: Conf. Ser. 1684 012056

View the article online for updates and enhancements.
A dynamic instrumentation tool for obtaining software logs

Hongchen Zhou\textsuperscript{1, a}\textsuperscript{*}, Jing Li\textsuperscript{1, b}

\textsuperscript{1}School of Software Engineering, South China University of Technology, Guangzhou, China
\textsuperscript{2}School of Software Engineering, South China University of Technology, Guangzhou, China
\textsuperscript{a*}Corresponding author: 940294396@qq.com
\textsuperscript{b}499672844@qq.com

Abstract—Compared with the software analysis based on source code, log based software analysis technology has gradually emerged in recent years, but it is not easy to obtain the log of software. To automatically obtain enough information on the software log, this paper proposes a dynamic instrumentation tool, it can dynamically instrument a software implemented in Java language, and get the log finally. This tool does not require users to be familiar with the software, and the instrumentation process will not cause an intrusive impact on the source code. It is a general-purpose automatic dynamic instrumentation tool.

1. INTRODUCTION

The log of software can reflect the structure and behavior characteristics of software to a great extent. At present, there are some relative researches on software analysis based on log, which have their advantages and disadvantages with software analysis based on source code. Based on the log, the structure of the software can be reconstructed and verified\cite{1}, and the running process of the software can be analyzed\cite{2} \cite{3}. Based on these products, the bottleneck of software can be further analyzed and optimized\cite{4} \cite{5}.

However, it is not easy to obtain a log of software\cite{6}. First of all, not all of the software implements the log acquisition function. Secondly, although some software has realized the log acquisition function, the information obtained is limited and cannot meet the needs of analysis work\cite{7}. Finally, considering that after the software is put into use, the maintenance of software may lead to software changes, but the log acquisition function often can not keep up with the speed of software changes. And the change of software results in the loss and omission of the original log acquisition function, and even the inconsistency of the format or information between the changed part of the log and the original log. Therefore, for a software that has been used for a while, its internal log acquisition function is often not good enough, and the operation log with missing information or inconsistent formal mode cannot be directly used for log analysis.

Therefore, in most cases, the software log acquisition module needs to be redeveloped\cite{8}. One of the original methods is to integrate the log acquisition function directly based on the software source code. However, this kind of development work not only requires developers to be familiar with the original system, but also needs a certain development cycle, and it is easy to introduce other errors. At the same time, when the software is updated and changed, we also need to follow up on the part of the log in time. Nowadays, Aspect Oriented Programming (AOP) has been widely used\cite{9}. In this paper,
the AOP idea is used for reference, and the non-invasive source code is considered. Finally, a tool is given, which can automatically instrument the software implemented in Java language and get the log. The contributions of this tool are as follows: 1) the software can be dynamically instrumented without modifying the source code of the software, to obtain sufficient software operation log with sufficient information; 2) using a configuration file to control the scope of the instrumentation and the object of the instrumentation, to instrument any Java language software, and the user does not need to be familiar with the software. The implementation does not need to pay attention to the later changes of the software, so it has wide applicability.

2. Relative definitions

This chapter will introduce the related concepts involved in this paper, mainly including the software objects targeted by the instrumentation tool, and the software log. Next, the above definitions are discussed in depth.

2.1 Instrumentation and instrumentation objects

This section will introduce the concepts and objects of instrumentation mentioned in this paper. Instrumentation refers to the process of embedding user-defined code into source code, intermediate code, or object code of software [10]. Dynamic instrumentation in this paper refers to the process that Java Software embeds bytecode when the virtual machine is running.

The instrumentation object refers to the software that needs to obtain the log. The software can be implemented by different languages, and the characteristics of different object-oriented programming languages are not the same. Therefore, the performance of software implemented by different programming languages may not be different, but the performance of the log is quite different. For example, the abstract class of Java and the virtual class of C++ have the same meaning, but their representation and keywords are not the same. Therefore, it is easy to find that the specific performance of the log is closely related to the language features of the software. In this paper, the software we study is implemented in the Java language, and the logs are obtained based on the relevant characteristics of Java [11]. On the other hand, considering that software in use is often deployed on multiple nodes, the instrumentation tool proposed in this paper can instrument distributed software programs.

2.2 Software log

The log refers to the relevant information of the software method at runtime [2]. In detail, the log is composed of records of method execution, and the records are sorted in the order in which they are generated. We call such a record an event. An event consists of several data items, which are called attributes of events[12].

As shown in Figure 1, each event consists of several data items. In this example, each event consists of event ID, timestamp, method, and lifecycle.

Therefore, we give a formal definition here: Definition 1 (event): an event is an information record generated by a method at run time, and it is also the basic unit of the log. An event is a seven tuple consisting of a series of predetermined data items. That is, event = < node X method_name X lifecycle X timestamp X thread_id X resource X class_message >, some of which can be empty. Detailed explanations are as follows:

- **node**: refers to the physical node running the current method;
- **method_name**: refers to the integrity description of the current running method, including the complete method name, modifier, return type, and parameter list of the method. If the constructor has no return type, the parameter list can be obtained only if there is a parameter method;
- **lifecycle**: indicates the start or end of the current method execution. Call and return are used to indicate the beginning and end of the current method. Besides, it also identifies the types of current methods, and the specific classification of methods will be discussed in detail in sections 2 and 3 of Chapter 3.
- **timestamp**: refers to the current running time;
- thread_id: refers to the system thread number of running the current method;
- resource: describes the system resources occupied by the current running method. For example, communication resources and thread resources. If these resources are not used by the current method, this field is empty;
- class_message: refers to the information about the class to which the currently running method belongs. If the target method of instrumentation is some well-known third-party library methods, then this field can be blank.

Among them, class_message data item is a more complex data item, which is composed of method class name, class attribute information, parent class information, interface information, and so on. Some data items can be null.

Definition 2 (software log): a software log is a collection of several events, and it is sorted in ascending order according to the chronological order of the events. In detail, the raw log is a collection of events generated by the software running the methods over a while. Record as log = \{event1, event2, event3..., eventn\}.

2.3 Monitored code
This section will introduce the monitored code mentioned in the paper. To obtain the data items in the events defined above, it is necessary to insert some code into the software, which we call monitored code.

![Figure 1. components of the log](image)

3. Theory and strategy of instrument
This chapter will mainly discuss three aspects, including the working theory of instrumentation, the positioning of instrumentation methods, and the strategies of instrument monitored code. In other words, we will first discuss how to implement instrumentation, then discuss which methods need to be instrumented, and finally discuss what monitored code needs to be embedded in a method.

3.1 Theory of instrumentation
This section mainly discusses how to implement non-invasive dynamic instrumentation of software. In short, the idea of getting the software log in this paper is to implant monitored code before and after the method in the software, and the software will execute the monitored code to generate a log when running this method [13]. The detailed theory is shown in Figure 2.
How to implant monitored code is the focus of this paper. For example, to obtain the time stamp information of the event, the simplest way is to manually add the monitored code to obtain the current timestamp at the beginning and end of the relevant method in the software. However, this method has serious limitations, we must get the source code of the software program and make changes in the source code. On the other hand, when the software is relatively large, this method undoubtedly needs a lot of work, and it is easy to miss methods and make errors. Therefore, it is unrealistic to process software manually. To solve this problem, the related concepts of AOP technology are introduced here [14].

AOP is a kind of programming thought. Different from the common object-oriented programming thought, AOP aims to separate general logic from business logic, like a programming method across programs. Referring to this way, this paper designs the log collection module as one aspect. So we design and implement a dynamic instrumentation tool based on an existing Xport tool for non-invasive code and wider applicability.

This tool is based on the dynamic proxy approach of instrumentation. The instrumentation technology is implemented by the java_agent framework [15]. This technology is essentially a bytecode operation technology. When the Java virtual machine loads the bytecode file, it intercepts the bytecode file to determine whether it needs to be instrumented. If instrumentation is needed, the bytecode is modified based on the methods provided by the javassist library, and the modified bytecode is finally loaded into the virtual machine by the class loader. The specific theory is shown in Figure 3.

It can be seen that this method has two obvious advantages: one is to modify the bytecode during the software running period without modifying the software source code; the other is that all methods can be instrumented through simple inspection rules. In short, we only need to know the structure of the software to automatically instrument any method, which is simple and efficient.
3.2 The positioning of instrumentation methods

This section discusses which methods in the software need instrumentation based on the previous section.

This paper classifies the methods involved in the software. Generally speaking, as a developer, the first thing to pay attention to is the member method of composing software. However, the operation of software inevitably needs to use the methods of system library, framework library, and other third-party libraries. Based on this classification method, the methods in the software are roughly divided into software member methods and third-party library methods.

For detailed, among all the methods that compose a software, one is the method directly existing in the software, that is, the method implemented by the developer. These methods are collectively referred to as software member methods. The other is a method called directly by developers. This method usually corresponds to a specific function and is developed by a certain organization and known by software developers. We call it the third-party library method. For the software member method, this paper further divides it into the construction method, interface method, and common method.

There are many reasons for this classification. The purpose of this paper is to obtain the software log that can be used for analysis. Although the third-party library method is also an important part of the software structure, these methods often do not directly reflect the business functions of the software, or the granularity is too small for the business functions. At the same time, the implementation details of third-party library methods are often known by most people, so most of the third-party library methods do not need to implant monitored code. The software member method is the method implemented by the software developers. This kind of method can reflect the software structure and the relationship between the structures [16], so it is necessary to obtain sufficient operation information. As a result, we need to insert the monitored code for all member methods in the software.

3.3 Strategies of instrument monitored code.

Based on the previous section, this section discusses which monitored code needs to be instrumented for a specific method to obtain the log.

Now explore what information needs to be collected for each of the above two methods. In Chapter 2, the exact definition of events has been given. Among all the data items of the event, some data items are the information that all methods need to collect when running, while some are not. We make a preliminary classification of these data items, in which node, method_name, lifecycle, timestamp, and thread_id these five data items are the necessary information for the method runtime, so all the interpolation methods need to obtain these five data items, which are abbreviated as E₁ in this paper. The resource data item is abbreviated as Eᵣ, the class_message data item is abbreviated as E₂.

According to the above, the software member method can not only reflect the structure of the software but also reflect the relationship between the internal structure of the software. Therefore, the software member method not only needs to obtain E₁, but also needs to obtain the specific information E₂ of the class to which the method belongs. The construction method is similar to the common method, which can obtain E₁ and E₂ directly. However, the methods in the interface will not run, so if the methods in the interface are directly instrumented, the relevant information cannot be obtained [17]. Therefore, based on the specific implementation of the interface, this paper proposes an instrumentation strategy: if a common method has the same name as a method in an interface implemented by the class, then the common method is instrumented to obtain the interface information, to determine the method information in the interface and the interface information in the E₂.

As mentioned above, most third-party library methods do not require instrumentation, but not all third-party library methods do not need to be embedded with monitored code. We carefully consider the resource data items in the event, mainly including two types of resources, one is thread resources, the other is communication resources. The so-called thread resources mainly refer to the information that can reflect the parent-child relationship between threads; communication resources are the information
that can reflect the communication between different processes. These two kinds of methods need to obtain $E^1$ and $E^2$.

The information reflecting the relationship between threads is to find the connection between threads, that is, when one thread starts another. Combining with the characteristics of Java language, it is necessary to instrument the thread creation method, that is, the thread start and other related methods, such as the normal start method, execute and submit method of the thread pool[18]. The information reflecting the relationship between different processes of multiple nodes is to find the communication between processes. This information needs to be obtained from the classes related to network communication. At present, the research in this paper is only aimed at the current mainstream communication protocol related methods.

4. EXPERIMENT AND ANALYSIS

In this chapter, the dynamic instrumentation tool is verified by experiments. We use the proposed tool to instrument a java software, and discuss the practicability of the dynamic instrumentation tool based on the experiment.

4.1 Experiment introduction

We use the dynamic instrumentation tool proposed in this paper to test a software based on Java language, verify whether it can successfully instrument different types of methods in the software according to the running log. The experimental process will be introduced below.

The experimental software is a distributed software composed of two sub-modules, which run on two virtual machines respectively. Submodule A creates a new thread to access submodule B and cooperatively completes the whole process. The experimental flow is shown in Figure 4.

4.2 Experimental analysis

Through the above experiments, we get the corresponding operation log information, and here we extract part of the log for display. As shown in Figure 5, the information for each event is represented in the form of key-value pairs. Each method executed in the software will produce two events, representing the start and end information of the method respectively. For software member methods, the method information and class-related information are obtained. In this experiment, threadID and newThreadId information are used to show the relationship between parent and child threads. The node number represents the method running on NodeA and NodeB.

Experiments show that the software has completed the function of instrumentation, and the parent-child contact information between threads has been successfully obtained. The practicability of the dynamic instrumentation tool proposed in this paper is fully verified, and the effect is good.
5. CONCLUSION
In this paper, the running log which can reflect the software structure is studied, it is found that there is still a big shortage in obtaining the running log which can reflect the software structure. Therefore, this paper proposes a dynamic instrumentation tool for Java software. It solves the problem of insufficient access to log information. At the same time, the plug-in process does not need to modify the source code, it is easy to operate and has good practicability. It can plug in any Java software and has wide applicability.

REFERENCES
[1] Van W. der A, Weijters T, Maruster L. Workflow mining: discovering process models from event logs[J]. IEEE Transactions on Knowledge & Data Engineering, 2004, 16(9): 1128–1142.
[2] Leemans M, Aalst W M P Van Der, Brand M G J V D B T-I I C on S A. Recursion aware modeling and discovery for hierarchical software event log analysis[A]. 2018.
[3] Lei Liang. Design and implementation of grid software log analysis system [D]. Harbin Institute of technology, 2014.(in Chinese)
[4] Hong Mei, Qianxiang Wang, Lu Zhang, et al. Progress of software analysis technology [J]. Acta Sinica Sinica, 2009, 32 (9): 1697 – 1710.(in Chinese)
[5] Liyuan Tai, Chunqi Tian, Wei Wang. Anomaly Detection of Large Scale Microservice Architecture Software System Based on Log Parsing. 2019, 09(12):2266-2276.
[6] Wei Wu. An implementation method of software log [J]. Journal of science and technology economy, 2017, 000 (5): p.45-45.(in Chinese)
[7] Xuhui Zhang. Design and implementation of SaaS oriented application software log Architecture [D]. Yanshan University.(in Chinese)
[8] Andrews R , Dun C G J V , Wynn M T , et al. Quality-Informed Semi-Automated Event Log Generation for Process Mining[J]. Decision Support Systems, 2020:113265.
[9] Harvey G, Salinas A, Suesaowaluk P. An aspect-oriented product line framework to support the development of software product lines of web applications[J].
[10] Huimei Liu, Huayu Xu. Research on code analysis and instrumentation technology in software testing [J]. Computer Engineering, 2007 (01): 92-94 + 97.(in Chinese)
[11] Java language programming: Java language programming: object oriented design ideas and practices [M]. China Machine Press, 2012.(in Chinese)
[12] Jing Li. Research on instantiation method of original log for software structure analysis [D]. South China University of technology, 2019.(in Chinese)
[13] Zhengtao Xiang, Yuping Miao, Dongming Lu. Agent oriented software analysis and design method [J]. Computer science, 2004 (6): 129 – 134.(in Chinese)
[14] Chengwan He, Zhipeng Ye. Detection method of SQL injection behavior based on AOP and dynamic Taint Analysis [J]. Acta electronica Sinica, 2019, 047 (011): 2413 – 2419.(in Chinese)
[15] Sebastian Theiss, Klaus Kabitzsch. A Java software agent framework for hard real-time manufacturing control. 2017, 65(11):749-765.
[16] Xiaoying Li. Research on Java based reverse transformation methods and tools [D]. Shantou University.(in Chinese)
[17] Yang J J. Design and application of communication interface class based on OOP[J]. Modern Electronics Technique, 2013.
[18] Verdu, Javier, Pajuelo,et al. Thread Assignment of Multithreaded Network Applications in Multicore/Multithreaded Processors[J]. IEEE Transactions on Parallel & Distributed Systems A Publication of the IEEE Computer Society, 2013.