A Comparison of Implementations for Aspect-Oriented JavaScript

Wenhao Huang[1,2], Chengwan He[1,2]
[1]School of Computer Science and Engineering. Wuhan Institute of Technology
[2]Hubei Province Key Laboratory of Intelligent Robot
Wuhan, China

Zheng Li
School Computer and Information Engineering, Henan University
Henan, China

***Abstract*—JavaScript is widely used to build web applications. The increasing scale and complexity of JavaScript programs bring much difficulty to the subsequent development and maintenance because of the same functions scatter in the programs. We can use AOP to reduce the difficulty but there are few AOP frameworks for JavaScript and they have many shortages should be improve. In this paper we will compare AspectJS, AOJS and AspectScript these three mature AOP JavaScript frameworks, and give some suggestions for further research.**

Keywords—JavaScript; AOP; web application; aspect; comparison

I. INTRODUCTION

OOP(object-oriented programming) promoted the flexibility and maintainability of programs, and it is widely used in large-scale project design[1]. As the scale and complexity of software increased, OOP has revealed certain limitations, such as the system logging, modules access control and other issues[2]. Programmers who use the object-oriented language such as Java and JavaScript to develop the software need to write a lot of repetitive codes to achieve and manage these functions. Then AOP(aspect-oriented programming) was proposed to solve these problems. The same function that scatters in different modules are called "crosscutting concerns[3]" in AOP, and AOP provides some methods to define and control them. Crosscutting concerns greatly reduce the redundancy in the program, and also reduce the difficulty and complexity of the development.

In the actual programming development, AspectJ[4] is a good AOP extension for Java. It provides mature AOP supports for Java software development. JavaScript is a popular script language that is widely used to build increasingly complex Web applications. AspectJS[5], AOJS[8] and AspectScript[9] are AOP extensions for JavaScript but their performances are not well as AspectJ's.They have some problems need to be solved, such as AspectJS and AspectScript are invasive to the original programs, and AOJS does not support the dynamic aspects deployment.

In this paper, we will make a comparison and summary for the three AOP frameworks for JavaScript and put some ideas to improve the three frameworks. Section 2 presents the three AOP frameworks for JavaScript. Section 3 puts forward a criteria for comparison. Section 4 makes a comparison between the three frameworks using the criteria. Section 5 put forward two envisages and section 6 concludes.

II. OVERVIEW OF IMPLEMENTATIONS

A. AspectJS

AspectJS is an industrial-strength JavaScript component that gives full control over method-call interception[5]. Based on JavaScript's dynamic nature, it uses Method-Call Interception(MCI)[6] to execute some other functions as "prefixes" and/or "suffixes" around the target object-method. AspectJS adopts this method to realize AOP.

AspectJS provides full and thorough control over the function calls that are to be intercepted to facilitate AOP in JavaScript. It calls the methods that are added to the target functions "affix" and a call to an object-method can cause other functions to execute as "prefixes" to that method. When the AJS object applies an initial prefix to an object's method, it modifies the object's internal reference to the method body, such that it points to a "proxy" function that is generated by the AJS object. When client code invokes the method, the proxy function executes instead, and calls any prefixes that have been applied. Once the last prefix has returned, it executes the intercepted method itself[7]. Fig.1 shows how to add a prefix to an object-method.

```
function prefixFunc(){
    /* something */
}
function myFunc(){
    /* something */
}
AJS.addPrefix(this,"myFunc",prefixFunc);
myFunc();
```

Fig. 1. The method that add a prefix to an object-method

Function *prefixFunc* is a custom prefix applying and *myFunc* is the target object-method. Pass the name of the target method and prefix applying to the function *addPrefix* of AspectJS framework. When the function *myFunc* is called, function *prefixFunc* will run before the function *myFunc*. Fig.2 shows the program execution process.

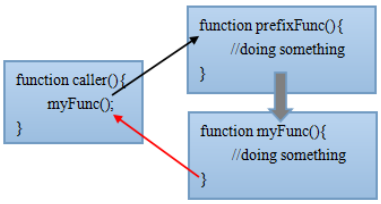

Fig. 2. The program execution process

B. AOJS

AOJS[8] is an aspect-oriented JavaScript framework that is differs from AspectJS. The most different place is that AOJS uses the XML files to define aspects and pointcuts, and it weaves aspects by a proxy server. In this way, AOJS realizes the complete separation of aspects from original programs. Other AOP frameworks for JavaScript can not separate aspects and original programs completely, and this shortage brings an issue that the frameworks can not ensure the consistency between original programs and the ones with aspects weaved. AOJS server consists of two parts:the reverse proxy for judging necessity of weaving and redirecting requests/outputs, and the weaver for weaving. Fig.3 shows the program flow.

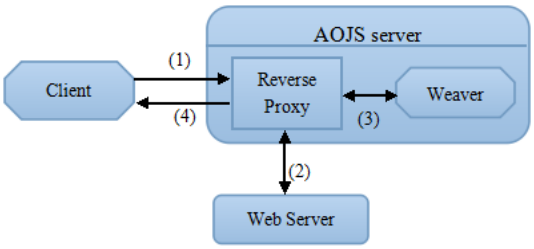

Fig. 3. AOJS program flow

Runtime behavior of AOJS is as follows[8]:

1) The client requests web pages with JavaScript programs to the reverse proxy via HTTP.

2) The reverse proxy fetches the target from the web server.

3) The reverse proxy passes the JavaScript programs to the weaver. The weaver weaves aspects which has been predefined in the XML files into the original JavaScript programs, then returns the results back to the reverse proxy.

4) The reverse proxy returns the results back to the client.

AOJS allows programmers to specify any variable assignment, function execution and the beginning part of the target JavaScript file as a joinpoint by using pointcuts <var>, <function> and <initializeFile>. The weaver of AOJS weaves aspects into the location where the initialization pointcuts specify and uses a code template for code replacement. The code template is shown by Fig.4.

```
(function(){
    <before>
     var _retvalue_ = <target>;
    <after>
     return _retvalue_;
})();
```

Fig. 4. The code template

The weaving process conducts the following four steps[8]:

1) Executes <before>.

2) Executes <target>, and stores the return value of the <target> expression into the temporal variable _retvalue_.

3) Executes <after>.

4) Returns the value stored in _retvalue_.

C. AspectScript

AspectScript is a full-fledged AOP extension of JavaScript that adopts higher-order programming and dynamicity as its core design principles[9]. Based on the dynamic characterizing feature of JavaScript, AspectScript supports the dynamically-deployed aspects. It also takes full advantages of the higher-order programming to make the definitions for pointcuts and aspects more precision. And integrates scoping strategies[10,11,12], it provides proper control over the scope of dynamically-deployed aspects. The way that adopts the hybrid join point model makes it possible to extend the framework richly. AspectScript also adopts the MCI to wrap the target functions. It uses *weave* to achieve aspects-weaving. Fig.3 shows the weaving process. Fig.5 shows the simplified definition of *weave*.

```
function weave(jp){
    var currentAspects = union(globalAspects,
        aspectsIn(ctxObj), aspectsIn(cxtFun));
    var advices = match(jp, currentAspects);
    return chainAndApply(advices);
}
```

Fig. 5. The simplified definition of function weave

The weaving process is as follows:

1) Computes current aspects to determine the set of aspects may apply.

2) Evaluates pointcuts of these aspects against the current join point.

3) Chains together these advises of aspects that matched the current join point and then applies them.

Fig.6 shows the weaving process.

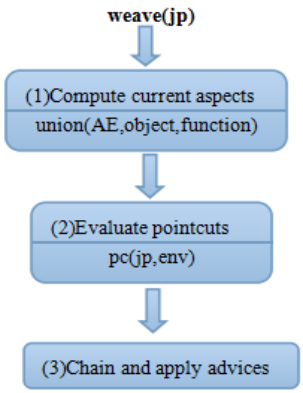

Fig. 6. Weaving process of AspectScript

III. COMPARISON CRITERIA

AOP is to make up for the inadequacy of OOP, it is the extension and complement of object-oriented programming. AspectJ is a mature AOP implementation for Java, and it has its own methods to define aspects, pointcuts and advises. Inspired by AspectJ, we think a full-fledged AOP JavaScript framework must have these points:

- Invasiveness. AOP is a supplement and extension for OOP. The original programs cannot be modify when programmers applied AOP to it, especially the original programs were under updating and maintaining. Because it would cause the secondary development. It is better to separate the aspects from original programs completely.
- Briefness. AOP is harder than OOP to be comprehend for those who never know it. Using JavaScript to define aspects in simple way can make it easy to apply AOP to JavaScript original programs. The pointcuts and advises should be defined accurately and simply.
- Maturity. The AOP framework for JavaScript should embrace the features of JavaScript and control aspects precisely and based on the dynamic nature of JavaScript, the framework should support dynamic aspect deployment.

IV. COMPARISON

A. Invasiveness

The codes for aspect-calls should be wrote in the original programs and it cannot be separated from the original programs completely. In other words, the original programs would be modified when programs used this framework to update it. This situation can be accepted in development but not the maintenance.

AOJS can separate the aspects from original programs completely by defining the pointcuts and aspects in XML files. Programmers who use this framework do not need to modify the original programs at all and the AOJS framework is transparent to users.

The working principle of AspectScript is same to the AspectJS's. So this framework also can not separate the aspects and original programs completely. It is invasive to the original programs.

B. Briefness

AspectJS is a lightweight AOP framework for JavaScript. It has a pre-built library for constructing aspects so programmers only need to define the advises with standard JavaScript functions and pass some parameters to the functions of AJS. It is easy to apply AOP with this framework.

AOJS can define aspects precisely in XML files by using some customized definition tags, such as <var>, <before> and <aspectsetting>. This method can separate the aspects from the original programs completely, but the it is complex to define aspect in XML files and deploy them. AOJS has its own server for weaving aspects into the original programs, its construction is more complex than others'.

AspectScript also has the pre-built library for constructing aspects. So programmers only need to define pointcuts and advises. AspectScript supports high-order programming and dynamic aspects deployment so that it has precise control over the aspects. But the programmers who use this framework need to define the advises carefully.

C. Maturity

AspectJS just uses MCI simply to wrap target functions.It can meet the fundamental AOP demands but it does not support higher-order programming nor dynamic deployment of aspects. There is no quantification in its simplest case.

AOJS takes quantification more seriously but does not embrace the full features of JavaScript. This framework can meet the practical requirements of basic AOP. But the way that specifies aspects in XML files brings an issue that aspects can not enjoy the full power of higher-order programming.

AspectScript is a more mature AOP framework for JavaScript. It supports higher-order programming and dynamic deployment of aspects. Pointcuts and advices in AspectScript are standard JavaScript functions and they take full advantages of higher-order programming patterns. This framework fully embraces the features of JavaScript and adopts hybrid join points to make the extension possible.

TABLE I below shows the results of the above discussion are summarized. "+" means that the framework meet the criteria well, and "-" otherwise.

TABLE I. Table aggregated summary of comparison

Framework	**Comparison Criteria**		
	Invasiveness	***Briefness***	***Maturity***
AspectJS	-	++	-
AOJS	+	+	-
AspectScript	-	++	++

V. RESEARCH DIRECTIONS

In this section, we put forward two Research Directions that can make the AOP implementation for JavaScript better.

A. Client Weaver

AOJS can separate aspects and original programs completely with XML files. But the extra server makes it not efficient. If the weaver that weaves aspects into original programs is deployed on the client, it can reduce the time of requests and responses to make the framework more efficient. The weaver can be achieved as a browser plug-in that need to be downloaded and installed the first time and can be disabled when the programs do not need it. The JavaScript aspects can be defined as standard JavaScript variables so that these aspects can embrace the features of JavaScript and support high-order programming.

B. *Precompiled Framework*

Because of the dynamic feature of JavaScript, the framework that similar to AspectJS and AspectScript can not dispel the invasiveness. JavaScript is an interpreted language so that it can not be executed after be compiled like static languages[13], such as C++, Java and so on. These is an idea that achieved a precompiled framework to weave the aspects into original programs before the programs executes. The precompiled framework can identify these aspects files with special marks that be defined in it. Then the framework weaves these aspects into original programs with the pointcuts. The precompiled framework is also a pre-built aspect library. Programmers only need to define the pointcuts and advises as the standard JavaScript functions and the percompiled framework can identify the join points to weave aspects into the original programs.

VI. CONCLUSION

Based on the features of JavaScript, the aspects that are standard JavaScript functions must be invasive to the original programs. So the aspect-oriented JavaScript framework should decrease the invasiveness. We can know that a AOP framework for JavaScript need a pre-built aspect library to simplify the definition of the aspects. If the framework adopts the method of AOJS to eliminate the invasiveness, the efficiency and cost must be taken into account. The AOP framework for JavaScript should fully embrace the features of JavaScript language and supports the dynamic deployment of aspects. In order to make the framework more all-sided, the support for higher-order programming must be one of the features of the framework. Inspired by AspectScript, the method that adopts hybrid join point is good to extend the framework richly. So a first-class AOP framework for JavaScript should support hybrid join points. At last, efficiency is very important to clients. One JavaScript program does not need the all modules of the framework. For high efficiency, the framework should have the ability to disable the unwanted modules.

ACKNOWLEDGMENT

This research project was supported by National Natural Science Foundation of China under Grant No.61272115, 60873024, 61402150, the Key science Technology Research Project of Hubei Provincial Department of Education under Grant No.D20121508.

REFERENCES

[1] Object-oriented programming. http://baike.baidu.com/view/324458.htm.

[2] Luo Hui-Ping. The Research and Implementation of AOP fundamental function on JavaScript[J]. Present in China High Technology Enterprises, 2007. 04. 049.

[3] Kiczales, G., Lamping, J., J. : Aspect oriented programming. In: Proceedings of ECOOP'97. Number 1241 in Lecture Notes in Computer Science, Springer Verlag (1997) 220-242.

[4] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold. An overview of AspectJ. In Jorgen L. Knudsen, editor, Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 2001), number 2072 in Lecture Notes in Computer Science, pages 327–353, Budapest, Hungary, June 2001. Springer-Verlag.

[5] Cerny. A javascript framework for method-call interception. http://www.cerny-online.com/cerny.js/.

[6] AspectJS. A function-call framework in JavaScript. http://www.aspectjs.com/.

[7] AspectJS. A JavaScript framework for aspect-oriented programming. http://www.aspectjs.com/AJS_Tutorial_01.htm.

[8] Hironori Washizaki, Atsuto Kubo, Tomohiko Mizumachi, Kazuki Eguchi, Yoshiaki Fukazawa, Nobukazu Yoshioka, Hideyuki Kanuka, Toshihiro Kodaka, Nobuhide Sugimoto, Yoichi Nagai, and Rieko Yamamoto. AOJS: aspect-oriented JavaScript programming framework for web development. In Proceedings of the 8th workshop on Aspects, components, and patterns for infrastructure software, pages 31–36, Charlottesville, Virginia, USA, 2009. ACM.

[9] Rodolfo Toledo, Paul Leger and Eric Tanter. AspectScript: Expressive Aspects for the Web. In Proceedings of the 9th International Conference on Aspect-Oriented Software Development, pages 13-24, New York, USA, 2010.

[10] Éric Tanter. Beyond static and dynamic scope. In Proceedings of the 5th ACM Dynamic Languages Symposium (DLS 2009), Orlando, FL, USA, October 2009. ACM Press. To appear.

[11] Éric Tanter.Execution levels for aspect-oriented programming.Technical Report TR/DCC-2009-8, University of Chile, October 2009.

[12] Éric Tanter, Johan Fabry, Rémi Douence, Jacques Noyé, and Mario Südholt. Expressive scoping of distributed aspects. In Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development (AOSD 2009), pages 27–38, Charlottesville, Virginia, USA, March 2009. ACM Press.

[13] Bruce Eckel. Scala: The Static Language that Feels Dynamic. http://www.artima.com/weblogs/viewpost.jsp?thread=328540.