AOJS: アスペクトを完全分離記述可能なJavaScriptアスペクト指向プログラミング・フレームワーク

Aspect-Oriented Programming Framework for JavaScript with Completely Separated Aspect Description

久保 淳人[*] 水町 友彦[*] 鷲崎 弘宜[*] 深澤 良彰[*] 鹿糠 秀行[†]
小高 敏裕[‡] 杉本 信秀[§] 永井 洋一[¶] 山本 里枝子[‖] 吉岡 信和[**]

あらまし JavaScript はスクリプト言語の一つで，Web におけるクライアントサイドプログラミングに広く用いられる．プログラミングにおける目標の一つとして横断的関心事の分離実装があり，その手段としてアスペクト指向プログラミングが提案された．JavaScript 言語においてアスペクト指向プログラミングを実現する枠組みは存在するが，アスペクトの完全分離記述は実現されていない．本稿では，アスペクトを完全に分離可能な JavaScript アスペクト指向プログラミング・フレームワーク AOJS (Aspect-Oriented JavaScript) を提案する．AOJS は，既存のフレームワークでは不可能な，変数代入に関するアスペクト織り込みも扱うことができ，また，織り込み処理をプロキシを用いて行うことにより，完全分離記述の強制を実現する．

1 はじめに

JavaScript はスクリプト言語の一つで，World Wide Web(WWW) におけるクライアントサイド・プログラミングに広く用いられている．

プログラミングにおいては，複数のモジュールに同様の処理が出現することがある．複数のモジュールに出現する代表的な処理として，ログ出力，セッション処理，キャッシュ等がある．これらの処理は，複数のモジュールに散らばり，かつ，各モジュールの本質的処理と混在するため，モジュールの独立性を低下させ，結果として各モジュールやプログラム全体の保守性を低下させる．このような，複数のモジュールに出現する処理を横断的関心事と呼ぶ．

アスペクト指向プログラミング (Aspect-oriented Programming; AOP) は横断的関心事の分離実装を可能にするプログラミング方式である [1]．アスペクト指向プログラミングでは，横断的関心事をそれぞれ独立にアスペクトとして分離記述することで，モジュールの独立性を高める．記述されたアスペクトは，記述から実行までのいずれかの段階で織り込み対象プログラムに合成 (weaving; 以下，織り込み) され，実行される．

WWW クライアントサイド・プログラミングにおいてアスペクト指向プログラミングを実現するフレームワークは多数存在する．JavaScript に関係するアスペクト指向プログラミングは，HTML 中のイベントハンドラとして JavaScript コードを織り込むもの [2–5] と，JavaScript コードに対する JavaScript コードの織り込みを

[*]Waseda University, 早稲田大学
[†]Hitachi, Ltd., 株式会社日立製作所
[‡]Fujitsu Laboratories ltd., 株式会社富士通研究所
[§]Toshiba Solutions Corporation, 東芝ソリューション株式会社
[¶]NEC Corporation, 日本電気株式会社 中央研究所
[‖]Fujitsu Laboratories ltd., 株式会社富士通研究所
[**]National Institute of Infomatics, 国立情報学研究所

行うもの [6–10] に大別される．これらの枠組みのうち，本稿では後者に注目する．

一般的に，JavaScript プログラムは，プログラマの手で記述され，Web サーバに配置され，Web サーバから JavaScript プログラムが送信され，送信された JavaScript プログラムをブラウザが解釈し実行する，という段階を経る．これらの手順には，JavaScript の記述から JavaScript プログラムの実行までにプログラム変換を行う段階が存在せず，従って，コンパイル型言語のようなアスペクトの織り込みが不可能である．このことから，JavaScript プログラムに対する JavaScript コードの織り込みを実現する既存のアスペクト指向プログラミング・フレームワークでは，JavaScript の言語機能のみを用いてアスペクト織り込みを実現している．

しかし，これらの既存フレームワークにおいては，織り込み対象プログラムに織り込みに関する記述を行う必要があり，結果としてアスペクト織り込みに関する記述を完全に分離することができない．また，例えばクライアントの動作ログ送信を目的とした場合，各変数の値の変化に対するアスペクト織り込みが必要になるが，上述のフレームワークでは，いずれも変数の値の変化には対応していない．

もう一つの問題点として，JavaScirpt におけるアスペクトの完全分離記述が実現されたとしても，記述の分離を開発期間にわたって強制するのは困難である．前処理系を用いて織り込みされた JavaScript コードを Web サーバ上に配備する開発プロセスにおいて,「必ず前処理系を通す」といった規約を用いて統制を行うことは可能である．しかし，開発規約による統制は，緊急保守時に処理後の JavaScript のみを修正するなど，規約が守られなくなる状況が容易に想定できる．従って，何らかの方法で，織り込み処理後の JavaScript コードの直接編集を不可能にする必要があるが，そのような取り組みはまだなされていない．

本稿では，JavaScript アスペクト指向プログラミング・フレームワーク AOJS(Aspect-Oriented JavaScript) を提案する．AOJS ではプロキシ上で透過的にアスペクト織り込みが処理される．また，変数代入および関数呼び出しに対するアスペクト織り込みが可能である．

AOJS を用いることで，アスペクトと織り込み対象プログラムの完全分離記述が可能になり，織り込み対象の JavaScript プログラムを全く変更せずにアスペクト指向プログラミングを実現できる．JavaScript を用いる Web アプリケーションにおける Web ブラウザの動作ログ取得コードを，JavaScript プログラムを変更せずに織り込むといった利用が考えられる．また，AOJS はプロキシ上で織り込み処理をするため，織り込み処理後の JavaScript プログラムが HTTP 通信上のみに存在することになり，直接編集を事実上不可能にし，開発規約による統制よりも強く分離記述の維持に貢献すると考えられる．

2 AOJS の概要

本章では，提案フレームワーク AOJS の概要を説明する．アスペクト指向プログラミングにおける要素を下記に示す．

ジョインポイント 織り込み指定の可能な位置の全体
アドバイス 織り込まれるコード断片
ポイントカット ジョインポイントから実際の織り込み対象を指定する条件
アスペクト 上記の各要素をまとめたモジュール

2.1 構成

AOJS におけるアスペクト織り込みは，Web サーバと Web ブラウザとの間に介在するプロキシサーバで行われる．

AOJS の概要を図 1 に示す．本稿では，Web サーバと同一の計算機で稼働するリバースプロキシとして AOJS を実現した．リバースプロキシとすることで，当該 Web サーバについて，JavaScript プログラムを含む全ての通信に介入することがで

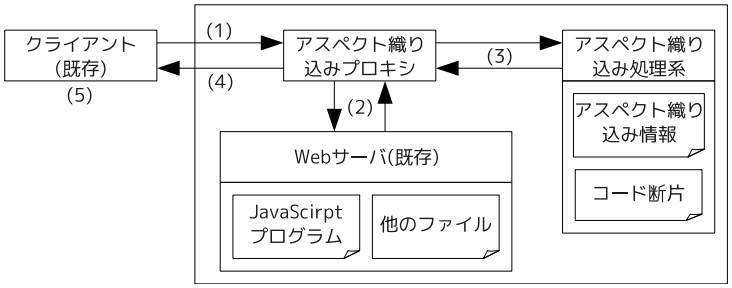

図 1　提案フレームワーク概要

きる．

Web ブラウザによる JavaScript プログラムの送信要求に対して，アスペクトが織り込まれた JavaScript が送信されるまでの順序を以下に示す．項番は図 1 の各数字に対応する．

1. **Web ブラウザからの JavaScript プログラム送信要求:** HTML ファイル等から参照されている JavaScript プログラムファイルについて，Web ブラウザはサーバに送信要求を行う．実際には，Web サーバの前段にあるアスペクト織り込みプロキシに対して送信要求がなされる．
2. **JavaScript 織り込みプロキシによるファイルの代理取得:** アスペクト織り込みプロキシは，Web ブラウザの代理として Web サーバに当該ファイルの代理取得を行う．
3. **アスペクト織り込み:** 代理取得したファイルが JavaScript プログラムファイルだった場合，アスペクト織り込み処理系を起動してアスペクト織り込みを行い，結果を取得する．開発者は，アスペクト織り込み処理系に対して，アドバイスとポイントカットを含むアスペクト指定することができる．
4. **アスペクト織り込み済 JavaScript プログラムの送信:** アスペクト織り込み済み JavaScript ファイルを Web ブラウザに送信する．JavaScript プログラム以外のファイルは，変更無しに Web ブラウザに送信される．
5. **アスペクト織り込み済 JavaScript プログラムの実行:** Web ブラウザは受信した JavaScript プログラムを実行する．

2.2　ポイントカット

AOJS では，ポイントカットとして JavaScript プログラム中の以下の要素を指定することができる．

- 変数への値の代入直前，直後
- 関数の実行直前，直後
- 処理対象 JavaScript ファイルの冒頭

図 2 は，AOJS における，アドバイスの織り込みを指定する XML ファイルである．after アドバイスにおいては，ポイントカットの実行結果として得られる値を`__retvalue__`変数として参照することができる．

`var` **要素**　変数への値の代入を指定するポイントカット．`varname` 属性で変数名を指定することができる．下位の `before` 要素および `after` 要素中にアドバイスを直接記述する．

`function` **要素**　関数呼び出しを指定するポイントカット．`functionname` 属性で関数名を指定することができる．下位の `before` 要素および `after` 要素中にアドバイスを直接記述する．

`inilializeFile` **要素**　処理対象 JavaScript ファイルの冒頭を指定するポイントカット．`inilializeFile` 要素でアドバイスを含むファイルを指定する．

```
<?xml version="1.0" ?>
<aspectsetting>
  <initializeFile>init.js</initializeFile>
  <var varname="/fib_gen_1/ret">
    <before><![CDATA[window.alert("/ret@before: " + ret +  "<br />");]]></before>
    <after><![CDATA[window.alert("/ret@after: " + ret +  "<br />");]]></after>
  </var>
  <var varname= "/y">
    <before><![CDATA[document.write("/y@before: " + y + "<br />");]]></before>
    <after><![CDATA[document.write("/y@after: " + y + "<br />");]]></after>
  </var>
  <function functionname = "/fib_gen_2">
    <before><![CDATA[var beg = (new Date()).getTime();]]></before>
    <after><![CDATA[var end = (new Date()).getTime(); sendLog( __retvalue__
      + ", " + (end - beg) + "ms");]]></after>
  </function>
</aspectsetting>
```

図 2　アスペクトの例

AOJS では，1 つの JavaScript ソースファイルについて処理を行い，プログラムのインクルードや HTML の script 要素によって読み込まれる他の JavaScript プログラムについては考慮しない．1 つの JavaScript プログラムにおいて，プログラムの各要素は木構造をとる．最上位の要素として，グローバルな変数や関数等が定義できる．関数中にはさらに変数や関数を定義することができる．従って，プログラム中の各要素は，ルート要素から各要素に至るパスによって特定可能である．ポイントカットはパスを用いて指定する．

AOJS におけるパス記述は，例えば `/myfunc/foo/x` のように行う．パス記述はスラッシュ記号から始まり，スラッシュ記号を区切りとして JavaScript プログラムファイル中に出現する各要素について，上位から下位に向かって記述する．パス記述においては，変数・関数を区別しない．パス記述は，`var` 要素や `function` 要素の `varname` 属性および `functionname` 属性で用いられる．

パス記述の例を以下に示す．

`/y`　グローバルな変数または関数 `y`

`/fib_gen_1/ret`　グローバルスコープに存在する `fib_gen_1` 関数に含まれる変数または関数 `x`

2.3　織り込み対象の抽出

アスペクト織り込み処理系は，出現した変数や関数について，ポイントカットに適合するジョインポイントであるか否かを判定する必要がある．アスペクト織り込み処理系は，処理対象となる JavaScript プログラムとアスペクトを入力として受け取る．入力を受けた AOJS は，2 段階処理によってポイントカットに適合するジョインポイントを抽出する．

第 1 段階では，JavaScript プログラムを先頭から順に解析し，それぞれの変数スコープ内で定義されている変数を抽出する．図 3 の JavaSciript プログラムを入力としたとき，図 4 の構造が得られる．入力 JavaScript プログラム全体は仮想ルートノードとして表現される．グローバル変数は仮想ルートスコープに含まれる．

第 2 段階では，JavaScript プログラム中での変数・関数の参照を抽出し，実際に参照・実行される関数・変数が，入力されたポイントカットに適合する場合にアスペクト織り込みを行う．例として，図 2 のアスペクトの各ポイントカットに適合するジョインポイントを表 1 に示す．

```
var x = 0; var y = 1;
function fib_gen_1() {
  var dummy, ret;
  ret = x + y; x = y;
  dummy = y = ret;
  return ret;
}
var z = 1;
function fib_gen_2() {
  function fib_2(x) {
    if(x <= 0) { return 1; }
    else if(x == 1) { return 1; }
    else { return fib_2(x - 1) + fib_2(x - 2); }
  }
  return fib_2(z++);
}
function fib_1() {
  var ret; ret = 100; y = 200;
  for(var i = 0; i < 30; ++i) document.form1.result.value = fib_gen_1();
}
function fib_2() {
  for(var i = 0; i < 30; ++i) document.form2.result.value = fib_gen_2();
}
```

図 3　フィボナッチ数計算を行う JavaScript プログラム

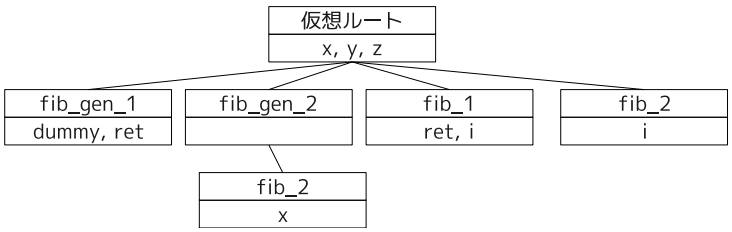

図 4　図 3 におけるスコープの包含関係と変数

表 1　各ポイントカットに適合するジョインポイント

ポイントカット	ジョインポイント	位置
<inilializeFile>init.js</inilializeFile>	-	図 3/先頭
<var varname=''/fib_gen_1/ret''>	ret = x + y	図 3/6 行目
<var varname= ``/y''>	y = ret	図 3/5 行目
<var varname= ``/y''>	y = 200	図 3/18 行目
<function functionname = ``/fib_gen_2''>	fib_gen_2()	図 3/21 行目

2.4　織り込み処理

抽出されたジョインポイントを置換するように織り込みを行う．織り込みにおいて共通利用するコードテンプレートとして，図 5 を用いる．図中の関数リテラルは，ジョインポイントを包み込み，関数本体の実行前後に before/after アドバイスをそれぞれ含む．ジョインポイントとなるコード断片が返り値を持つ場合は一旦保存して，アスペクトの実行終了後に返却する．定義された無名関数は，テンプレート最後部の () によって即座に実行される．式の値を持つ点においては，代入式も関数呼び出し式も同等であるので，同一のコードテンプレートを用いることができる．

織り込み処理で得られるコード断片は全体として関数呼び出し式になり，織り込み前の代入式/関数呼び出し式を置換コードの関数呼び出し式で置換するため，プ

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

図 5　織り込み用コードテンプレート

```
function sendLog(string) {
    var httpoj = createXMLHttpRequest();
    httpoj.open( "GET" , "/log.cgi?message=" + string, false );
    httpoj.send( "" );
}
(中略)
function fib_gen_1() {
  var dummy, ret;
  (function(){ window.alert("/ret@before: " + ret +  "<br />");
    var __retvalue__=ret = x + y;window.alert("/ret@after: " + ret +  "<br />");
    return __retvalue__;})();
  x = y;
  dummy = (function(){ document.write("/y@before: " + y + "<br />");
    var __retvalue__=y = ret;document.write("/y@after: " + y + "<br />");
    return __retvalue__;})();
  return ret;
}
(中略)
function fib_2() {
  for(var i = 0; i < 30; ++i) document.form2.result.value = (function() {
      var beg = (new Date()).getTime();var __retvalue__=fib_gen_2();
      var end = (new Date()).getTime(); sendLog( __retvalue__ + ", "
      + (end - beg) + "ms");return __retvalue__; })();
}
```

図 6　織り込み処理された JavaScript プログラム (一部)

ログラムの他の部分への影響がない．従って，図 3/5 行目のような，多重代入においても織り込み処理が可能である．また，置換された部分が独立の変数スコープを形成するため，アドバイス内で自由に変数を定義できる．

例として，図 3 について，図 2 で指定されたアスペクトを織り込む．各ポイントカットで指定されるグローバル変数 (`/y`)，ローカル変数 (`/fib_gen_1/ret`)，関数呼び出し (`/fib_gen_2`) について，コードテンプレートを用いた織り込みが行われ，結果として図 6 の JavaScript プログラムが得られる．

2.5　制限

2.5.1　無名関数

無名関数を変数に代入することで関数を作成している場合，関数に名前がないので階層に名前をつけることができない．無名関数については，例えば定義順に$1,$2 など関数名を与えて特定する方法が考えられるが，現時点では実装されていない．

2.5.2　変数名の参照時修飾

配列やオブジェクトプロパティのように，宣言時と使用時で文字列が異なる場合，同変数として判定できない問題である．たとえば，配列についてはシンボルが `x` だが，代入の際は，`x[0]` のようなシンボルで参照される．また，プロパティは，使用する際，プロパティ名の前に呼び出し元のオブジェクトを記述しなければならない．このように，宣言時と変数使用時で変数の記述が部分的に異なる場合は，同一の変数として判定できない．

2.5.3　曖昧なコーディング

Web ブラウザに含まれる JavaScript 実行環境は、多くが曖昧な構文解析を行う．寛容な JavaScript パーサに依存して，多くのライブラリにおいて，行末セミコロン

```
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 1, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 2, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 3, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 5, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 8, 0ms
(省略)
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 196418, 161ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 317811, 266ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 514229, 431ms
[Fri Jun 27 09:34:26 2008] [133.9.74.xx] 832040, 694ms
[Fri Jun 27 09:34:27 2008] [133.9.74.xx] 1346269, 1086ms
```

図 7　サーバに記録されたログ

の省略など文法に沿わない部分があるため，提案システムで構文解析できない場合がある．前処理による文法に沿った形式への変換か，提案システムのパーサを変更するなどの対策が必要である．

2.5.4　関数実行ジョインポイント

提案システムは，関数実行ジョインポイントを定義できない．このことから，現状「あるライブラリで定義された関数のすべての実行前に特定のコードを実行する」といった織り込み処理が不可能である．関数実行ジョインポイントは契約による設計 (Design by Contract) の実現に必要であるため，今後の実装を予定している．

2.5.5　インクルード順序の考慮

ライブラリを利用する JavaScript プログラムにおいては，JavaScript プログラムが Web ブラウザ上で統合される．したがって，複数の同名関数定義が存在した場合，実際に参照される関数がどれになるかは JavaScript ファイルのインクルード順序から影響を受ける．しかし，AOJS は JavaScript ファイル単体での織り込み処理を行うため，クライアント上での関数定義の上書きを考慮できない．この点については，クライアント上で織り込み処理を行う既存アスペクト処理系を組み合わせて用いることで解決できる．

3　実験

Java 言語を用いてアスペクト織り込み処理を，Perl 言語を用いて織り込みプロキシをそれぞれ実装した．JavaScript の解析処理には，JavaCC を用いて生成したコード雛形を変更して用いた．AOJS について，機能性および効率に関する実験を行った．

実験環境は，以下の通りである．**サーバ:** AMD Athlon 3500+，4GB Memory，Debian/GNU Linux 4.0r3，Apache 2.2.3，Squid 2.6．**クライアント:** Intel Core2Duo U7500，2GB Memory，Ubuntu Desktop 8.04.

3.1　機能性

図 3 の JavaScript ファイルと AOJS を Web サーバ上に配備し，再帰型のフィボナッチ数計算のパフォーマンスログをサーバに送信する実験を行った．JavaScript ファイルは図 2 のアスペクトに従って織り込み処理され，図 6 の JavaScript ファイルが Web ブラウザに送信される．`fib_2` 関数中の `fib_gen_2` 関数呼び出しの前後で時刻を計測し，その差分をサーバに送信している．`sendLog` 関数は `init.js` ファイル中で定義されており，XmlHttpRequest を用いてサーバにログを送信する．

実験の結果，サーバに記録されたログを図 7 に示す．計算の進行に伴って計算時間が増大することがわかる．

この実験から，AOJS においてアスペクト (図 2) の完全なモジュール化と分離に成功したことがわかる．また，既存プログラムの修正が不要なため既存の Web アプリケーションに容易に適用可能である．さらに，変数値の変化など，ジョインポイ

表 2　織り込み処理時間の比較

対象	行数	所要時間 [秒]
図 3	23	0.516
prototype.js [11]	2517	20.6

表 3　パフォーマンス計測結果

	Web サーバ	+織り込みプロキシ	+キャッシュプロキシ
処理リクエスト数 [個/秒] (HTML)	1272	143.6	1290
(JavaScript)	1232	1.77	1327
応答時間 [中間値; ミリ秒] (HTML)	0	4	0
(JavaScript)	0	560	1

ントの拡充により詳細なログ記録などの要求に合致した織り込みができることを確認した.

3.2　織り込み処理効率

実験用 Web サーバ上で，図 3 の JavaScript プログラム，および，prototype.js [11] に対して，それぞれ図 2 のアスペクト織り込み処理を行った．所要時間は各 10 回計測し平均値を算出した.

2.5.3 章で述べたように，prototype.js のような既存のライブラリについて構文解析に失敗するため，エラー箇所を手作業で訂正した．訂正点は，行末セミコロンの補完が 9 箇所，文字列リテラル中の非対応文字の削除が 3 箇所である.

結果を表 2 に示す．結果から，特に大規模な JavaScript プログラムについて，織り込み処理に要する時間が実用範囲を超えることが分かる．大規模 JavaScript プログラムへの織り込みパフォーマンスは今後の課題である.

3.3　リクエスト処理効率

実験用 Web サーバ上に存在する HTML ファイルおよび JavaScript ファイルを，(1)Web サーバから直接取得した場合，(2) 同一コンピュータで稼働するアスペクト織り込みプロキシ経由で取得した場合，(3) さらに同一コンピュータ上でキャッシュプロキシを追加した場合について，実験を行った．それぞれの場合について，Web サーバ Apache 付属の Apache Benchmark Tool を用いて，1 秒あたりのリクエスト処理能力と，応答時間の中間値を計測した.

実験における総リクエスト数を 10^5，並列数を 1 とした．通常，Web サーバの性能測定では並列でリクエストを発行するが，実験時における AOJS プロキシはシングルスレッド実装であったため，条件を合わせるために並列数を 1 とした.

表 3 に結果を示す．単純にアスペクト織り込みプロキシを追加した場合，大幅な性能低下が観察された．HTML ファイルの取得では織り込み処理が行われないため，性能低下は Web サーバへの代理接続が原因であると考えられる．直接取得の場合と比較して，リクエストごとに数ミリ秒の遅延がある．JavaScript ファイルの取得では織り込み処理が追加されるため，さらに性能が低下している．HTML ファイルと JavaScript ファイルの応答時間の差約 550 ミリ秒が，織り込みにかかった時間である．なお，応答時間中間値は表示が 1ms 精度であるため，応答時間中間値の値 0 および 1 は測定限界以下と見なす.

Squid によるキャッシュプロキシを追加した場合，HTML ファイルの取得については直接取得に準ずる性能を得た．JavaScript ファイルの取得では，織り込み処理が最初の 1 回だけになるため，アスペクト織り込みプロキシのみの場合と比較して性能が向上し，実用上問題ない水準に達した.

4 関連研究

AspectJS [2]，ulibjs Aspect [3] および Cerny.js [5] は，ポイントカットとしてオブジェクト・メソッドを指定可能なアスペクト指向プログラミング・フレームワークである．アスペクトの指定に先立って特定のライブラリを取り込んでおき，アスペクト全体を JavaScript コードとして記述する．AspectJS は JavaScript 仕様の範疇のみで実現可能であるが，織り込み対象のプログラムを変更する必要がある．

Dojo [6]，Ext JS [7]，The Yahoo! Uesr Interface Library (YUI) [8]，ajaxpect [9] は，id 属性や name 属性など，特定の条件を満たす HTML タグのイベントハンドラをジョインポイントと見なして JavaScript コードを埋め込む．岡本らのアスペクト指向プログラミングフレームワーク [4] では，上記のイベントハンドラ埋め込みの条件指定の制約を緩和し，任意の HTML タグのイベントハンドラをジョインポイントとして指定可能である．

Mozilla Firefox は，JavaScript の独自拡張として `watch/unwatch` 関数 [10] を提供している．これらの関数を用いることで，オブジェクトのプロパティについて値の変化時に呼び出されるフックを設定および解除することができる．対して AOJS は，オブジェクトのプロパティでない通常の変数について代入時を指定してコード織り込みが可能である．また，AOJS は Web ブラウザの独自拡張機能を使わないため，より汎用的である．

上記の関連研究では，織り込み対象のプログラムや HTML ファイルに対して何らかの追加記述が必要である．典型的な追加記述として，当該アスペクト指向プログラミングフレームワークを実現する外部 JavaScript プログラムを呼び出す `link` 要素が必要とされる．AOJS は，織り込みに際して既存プログラムや HTML ファイルの変更が不要である．反面，AOJS では通信に介在するプロキシとしてアスペクト織り込みを実現しているため，JavaScript の言語機能のみを用いたアスペクト指向プログラミングと比較して性能が大幅に低下する．

大村らは，Web ブラウザと Web サーバの通信を関数呼び出しと見なすアスペクト指向ウェブフレームワークを提案した [12]．大村らのフレームワークと AOJS は，プロキシの形態で実現されている点で共通しているが，織り込み処理対象が HTML と JavaScript プログラムである点で異なっている．

Stamey [13] らは，PHP を用いて HTML 中に JavaScript コードを織り込む手法 Aspect-oriented PHP(AOPHP) を提案した．対象言語が PHP である点で AOJS と異なっているが，サーバからの送出時に織り込み処理を行う点で共通している．

5 おわりに

本稿では，アスペクトを完全に分離記述可能な JavaScript アスペクト指向プログラミング・フレームワーク AOJS を提案した．AOJS はリバースプロキシとして動作し，Web サーバからの JavaScript コード送出に割り込んでアスペクト織り込みを行うため，既存コードの変更が一切不要である．また，AOJS は，プロキシを用いて透過的に織り込みを行うため，従来の JavaScript-AOP フレームワークと容易に組み合わせて利用することができる．さらに，織り込み処理後の JavaScript プログラムが HTTP 通信上にしか存在しないため，織り込み処理結果の直接編集が事実上不可能になり，開発規約による統制よりも強くコードの分離記述の維持に貢献できると考えられる．

実験により，AOJS によるアスペクトの完全なモジュール化と分離を確認した．さらに，変数値の変化など，ジョインポイントの拡充により詳細なログ記録などの要求に合致した織り込みができることを確認した．プロキシによる織り込み処理は性能を低下させるが，前段にキャッシュプロキシを設置することで，小規模なプログラムであれば実用上問題ない性能を得られることを確認した．数千行におよぶ大規模な JavaScript プログラムについての織り込み処理時間は非実用的な水準に到達す

るため，織り込み処理の効率化が必要である．

AOJS を用いることで，JavaScript によるアスペクト指向フレームワークの導入および管理が容易になることが期待される．

今後の課題として，以下がある．

- 無名関数における変数スコープ：無名変数への参照を保持する変数を経由して呼び出す必要があるため，参照を保持する変数の変数名を階層名として利用できるが，より詳細な解析が必要である．
- 変数名の参照時修飾への対応：より詳細な構文解析や，各変数が同一かどうかの判定処理が必要である．
- アスペクト織り込みプロキシを考慮したキャッシュ方法の検討が必要である．
- 文法に厳密に沿わないコーディングへの対策：前処理によりソースコードを変更する方法と，提案システムの構文解析部を寛容にする方法がある．
- 関数実行ジョインポイントの追加：特定の関数のすべての実行に関するアスペクト織り込みに対応したい．このジョインポイントを用いることで，契約による設計の実現が可能になる．
- 織り込み処理のパフォーマンス：大規模な JavaScript プログラムに関して織り込み処理時間が非実用的な水準まで増大することが分かったため，織り込み処理のパフォーマンス改善が必要である．

謝辞

本研究の一部は，財団法人 情報科学国際交流財団・産学戦略的研究フォーラム (SSR)2008 年度の助成を受けました．また，本研究の一部は国立情報学研究所・GRACE センターの支援を受けました．

参考文献

[1] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, Vol. 1241, pp. 220–242. Springer-Verlag, 1997.

[2] Zero. Aspectjs. `http://zer0.free.fr/aspectjs/`.

[3] 岡田耕一. ulibjs aspect. `http://www.sip.eee.yamaguchi-u.ac.jp/kou/JavaScript/Aspect.html`.

[4] 岡本隆史, 鷲崎弘宜, 深澤良彰. Javascript におけるアスペクト指向プログラミングの応用. ウィンターワークショップ 2008・イン・道後 論文集, pp. 69–70. 情報処理学会 ソフトウェア工学研究会, 1 2008.

[5] Robert Cerny. Cerny.js. `http://www.cerny-online.com/cerny.js`.

[6] Dojo. `http://dojotoolkit.org/`.

[7] Ext js. `http://extjs.com/`.

[8] The yahoo! user interface library. `http://developer.yahoo.com/yui/`.

[9] Google. ajaxpect. `http://code.google.com/p/ajaxpect/`.

[10] Mozilla Foundation. Core javascript 1.5 reference:global objects:object:watch. `http://developer.mozilla.org/ja/docs/Core_JavaScript_1.5_Reference:Global_Objects:Object:watch`.

[11] Prototype Core Team. prototype.js. `http://prototypejs.org/`.

[12] 大村裕, 山岡順一. アスペクト指向ウェブフレームワークによる情報共有システムの実現. ソフトウェア工学の基礎 XII, pp. 13–18, 2005.

[13] John Stamey, Bryan Saunders, and Simon Blanchard. The aspect-oriented web. In *the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, pp. 89–95, 2005.