A-007

JavaScript へのオウン変数機能と関数クローニング機能の追加
Addition of own variables and function cloning to JavaScript

柏倉 歩† 大谷 真†
Ayumu Kashiwakura Makoto Oya

1. はじめに

JavaScript は、クラスレスのオブジェクト指向言語であるとともに、第 1 級関数をサポートした関数型言語である。グローバル変数や変数スコープの問題、セミコロン挿入や==の一貫性上の問題などが指摘されているものの、ECMAScript5[1]による標準化とその実装も進み充実した言語に成長している。フロントエンドプログラミングを中心に世界的に広く使われており、最近では Node.js にみられるようにサーバプログラミングにも使われ始めている。JavaSctipt の素晴らしい点は、その技術的特徴だけでなく、これまで一部の専門家しか使っていなかった第 1 級関数を使った関数型プログラミングを一般プログラマが普通に使うまでに大衆化させた点にある。実際に一般の Web プログラムが何の違和感なく関数オブジェクトを変数に代入したり関数の引数に指定している。

しかし、JavaScript の関数型プログラミング機能に関して、一般プログラマの視点から次の 2 つの問題点があると我々は考えている。

(1) オウン変数を使うのが難しい

クリック処理関数内や Ajax コールバック関数内では、オウン変数(関数呼び出しを超えて値が引き継がれる関数内ローカル変数)が必要になることが多い。JavaScript でオウン変数を使うには、その変数を宣言してから当該関数を返す関数を作り即時実行することで、クロージャ機能を利用して実現するのが妥当とされている[2][3]。しかし一般の JavaScript プログラマには、関数オブジェクトを代入したり引数に指定したりするのは日常よく行っているものの、関数を返す関数を自分で作ってそれを即時実行するのは技術的に極めて難しい。実際のところこのようなクロージャ機能は一般 javaScript プログラマ向けの書籍には殆ど書かれていない(注：Web プログラマ向け国内書籍を調査したところ、JavaScript 書籍のクロージャ掲載率は 15%(6/40 冊)、JavaScript を活用した HTML 作成書籍にクロージャの掲載はなかった(0/14 冊))。このため多くの Web プログラマは、プログラムの保守性や拡張性を犠牲に、関数インスタンスごとにグローバル変数を宣言することで済ませていることが多い。

(2)オウン変数を持つ関数のコピーができない。

同じ処理内容だが異なるオウン変数を持つ関数を複数個所のコールバック関数として使うことは多い。しかし JavaScript には関数をコピーする機能がない。類似のことはプロトタイプオブジェクトを使ってできるように思われるがオウン変数が共有されてしまうなどの問題があり妥当でない。

この研究では上記 2 つの問題点を解決するために、(1)JavaScript の関数定義機能を拡張し一般プログラマにも分かりやすい形でオウン変数が定義・利用できるようにした。また、(2)オウン変数を含む関数オブジェクトをコピーするための関数 clone を新たに導入した。さらに、(3)評価のためにこれらの拡張をサポートしたプリプロセッサを開発した。

†湘南工科大学工学研究科, Graduate School of Engineering, Shonan Institute of Technology

2. 問題点と解決アプローチ

2.1 オウン変数

オウン変数とは、ALGOL60[4]で使用された用語であり、C 言語のスタティック変数に類似している。つまり、関数呼び出しを超えて値の持続する変数のことである(図 1)。これは従来の JavaScript には存在しない。

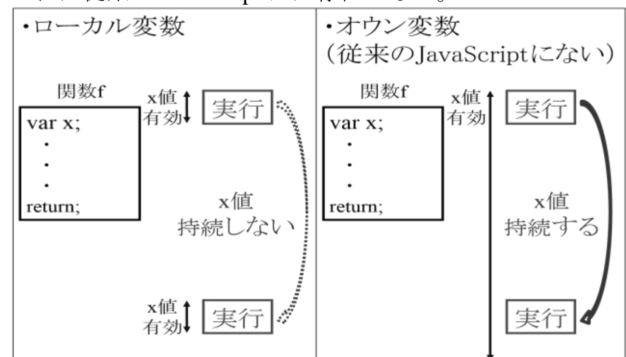

図 1 ローカル変数とオウン変数の違い

2.2 クロージャ

クロージャとは、関数定義時に定義されているコンテキスト(変数など)は、その関数の実行時のコンテキストに含まれるという性質を活用した関数呼び出しを超えて値の持続する変数を持つ関数である。これを使用してオウン変数を実現するには、図 2 のように関数内で関数呼び出しを超えて値の持続する変数、本体となる関数を定義し、その関数を返す、という複雑な関数定義をしなければならない。

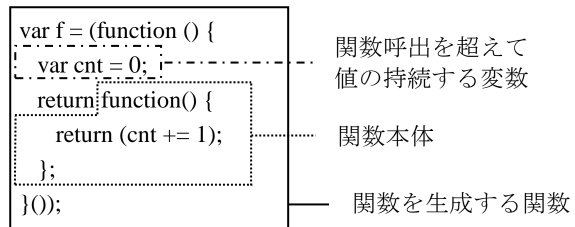

図 2 クロージャ

2.3 解決アプローチ

本研究では既に開発されつつあるクラスベースではなく JavaScript の柔軟さを活かすことのできるクラスレスベース(プロトタイプベース)プログラミングで解決する。その前提のもと、現在の関数定義に新たな文法を追加し、"関数を生成する関数"という複雑な概念を使用せず、オウン変数が定義できるようにする。

(1) オウン変数機能の追加

プログラマはオウンブロックを使用し、その中でオウン変数(関数・オブジェクトも可)を定義する。使い易さを考え、関数呼出しは通常の JavaScript と同じとする。これ

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

によりプログラマは見かけ上クロージャを使用せず、オウン変数を使用できるようになる。

(2) 関数クローニング機能の追加

2 つのボタンのクリック回数を別々に記憶しておきたい場合などに、関数の処理は同じだが別の関数を作りたい場合がある。クラスベースの言語の場合は同じクラスのインスタンスを作ることで対処できるが、クラスベースでないオブジェクト指向言語である JavaScript では一般的に Web プログラマに分かり易い手段がない。そこで関数クローニングという新パラダイムを導入する。

(3) プリプロセッサの開発

追加した関数機能を標準の JavaScript 処理系で処理させるため、標準的な JavaScript に変換するプリプロセッサを開発する。また、変換後の可読性保持のため、できるだけ元のコードを生かした変換とする。

3. オウン変数機能

3.1 仕様

関数定義の中にオウン変数宣言用にオウンブロックを追加した。一般 Web プログラマにも理解しやすくするため、従来 JavaScript 関数定義の拡張形とした。拡張した文法を図 3 に示す。

```
function 関数名([仮引数リスト]) {
  @own
    var オウン変数宣言 1;
    var オウン変数宣言 2;      ---- オウンブロック
     ・・・                        と呼ぶ
  @end
  文
}
```

図 3 オウンブロックの追加

オウン変数宣言が 1 つの場合、以下のようなシンタックスシュガーを使用することができる。

```
@own オウン変数宣言;
```

オウンブロック内で宣言された変数はローカルスコープにあるものの、関数が返った後も持続し、次の関数呼び出し時に値が引き継がれる。なお、オウンブロックには変数以外に関数、オブジェクトを宣言することもできる。使用例を図 4 に示す。

```
function fn() {
  @own
   var cnt = 0;   } オウン変数が 1 つなので以下でも可
  @end              @own cnt = 0;
  return (cnt += 1);
}
fn(); //1 が返る  } 実行例
fn(); //2 が返る
```

図 4 オウンブロックの使用例

3.2 変換方式

図 3 をプリプロセッサで図 5 の通り変換する。

```
var 関数名 = (function() {
  var オウン変数宣言 1;
  var オウン変数宣言 2;
   ・・・
  return function 関数名([仮引数リスト]) {
    文
  };
}());
```

図 5 オウンブロックの変換規則

例えば、図 4 は図 6 のように変換する。

```
var fn = (function() {
  var cnt = 0;
  return function fn() {
    return (cnt += 1);
  };
}());
fn(); //1 が返る
fn(); //2 が返る
```

図 6 オウンブロックの変換例

4. 関数クローニング機能

4.1 仕様

関数定義を行うと関数クローニングを行う clone メソッドが自動的に生成されるようにした。図 7 では関数名の関数をクローン関数名にクローニングしている。

```
function 関数名([仮引数リスト]) {
  @own
   var オウン変数宣言 1;
   var オウン変数宣言 2;
    ・・・
  @end
  文
}
var クローン関数名 = 関数名.clone();
```

図 7 関数クローニング

clone メソッドはクローン元関数の関数本体をコピーする。この際、ローカル変数の値はコピーせず、オウン変数は初期状態でコピーする。使用例を図 8 に示す。

```
function f(v) {
  @own max = undefined;
  if (max < v) {
    max = v;
  }
  return max;
}
var g = f.clone();
f(20); f(10); f(40);  //順に 20、20、40 が返る
g(10); g(40); g(20); //順に 10、40、40 が返る
```

図 8 関数クローニングの例

更にクローニングの際にクローン元関数内にあるオウン変数の値を引数で指定できるようにした。実際には「@cloning_arguments」という変数を用意した。この変数をクローン元関数のオウンブロック内で使用すると、引数に与えられた値をオウン変数に渡すことができる。使用例を図 9 に示す。

```
function f(v) {
  @own max = @cloning_arguments[0] || undefined;
  if (max < v) {
    max = v;
  }
  return max;
}
var g = f.clone(30);
f(20); f(10); f(40);  //順に 20、20、40 が返る
g(10); g(40); g(20); //順に 30、40、40 が返る
```

図 9 クローニング時引数の例

4.2 変換方式

図 7 は図 10 の通り変換する。

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
var 関数名_kgen = (function() {
  var オウン変数宣言 1;
  var オウン変数宣言 2;
  ・・・
  return function 関数名([仮引数リスト]) {
    文
  };
});
var 関数名 = 関数名_kgen();
関数名.clone = 関数名_kgen;
```

図 10 関数クローニングの変換規則

例えば、図 8 は図 11 のように変換する。

```
var f_kgen = (function() {
  var max = undefined;
  return function f(v) {
    if (max < v) {
      max = v;
    }
    return max;
  };
});
var f = f_kgen();
f.clone = f_kgen;
var g = f.clone();
f(20); f(10); f(40);  //順に 20、20、40 が返る
g(10); g(40); g(20); //順に 10、40、40 が返る
```

図 11 関数クローニングの変換例

また、図 9 は図 12 のように変換する。

```
var f_kgen = (function() {
  var max = arguments[0] || undefined;
  return function f(v) {
    if (max < v) {
      max = v;
    }
    return max;
  };
});
var f = f_kgen();
f.clone = f_kgen;
var g = f.clone(30);
f(20); f(10); f(40);  //順に 20、20、40 が返る
g(10); g(40); g(20); //順に 30、40、40 が返る
```

図 12 クローニング時引数の変換例

5. プリプロセッサの開発

5.1 様々な関数宣言方法に対応

4.2、5.2 で述べた形式で変換する。JavaScript の関数宣言の 5 つの方法(関数宣言文、有名関数リテラル、無名関数リテラル、無名関数、Function コンストラクタ)、全てをサポートした。

ただし、現在のプリプロセッサはオウン変数、関数クローニング共にサポートしているため、実際には 5.2 の形式で変換している。

5.2 関数クローニング機能の変換

図 7 から図 10 へ変換は、以下の方法で行っている。

まず function キーワードと{ }の数から関数定義の範囲を求める。function キーワード行から関数名を求め、変数に格納する。「@own」行から「@end」行前までをオウンブロックとして変数に格納する。この際、「@cloning_arguments」がある場合は arguments オブジェクト「arguments」に置換する。function キーワード行と関数本体(オウンブロックを除く)、関数定義終了行を抜き出し、関数ブロックとして変数に格納する。その後、変数に格納したオウンブロック、関数ブロックに必要なコードを付加して繋ぎ合わせ、変換済み関数ブロック(図 13)を作成する。この変換済み関数ブロックは関数定義のみを行い、関数の生成は行わない(図 13)。

```
(function() {
  var オウン変数宣言 1;
  var オウン変数宣言 2;
  ・・・
  return function 関数名([仮引数リスト]) {
    文
  };
});
```

図 13 変換済み関数ブロック(定義のみ)

ここで図 13 は、元の関数とは異なる"関数を生成する関数"という関数のため、別の関数名に代入することとした。実際には変数に格納された関数名に「_kgen」という文字列を付加したものに代入する。そして、元の関数名には関数名_kgen を実行した、つまり生成された関数を代入する(図 14)。

```
var 関数名_kgen = (function() {
  var オウン変数宣言 1;
  var オウン変数宣言 2;
  ・・・
  return function 関数名([仮引数リスト]) {
    文
  };
});
var 関数名 = 関数名_kgen();
```

図 14 関数を生成する関数と生成された関数

その後、生成された関数に clone メソッドを追加する。clone メソッドの内容は関数名_kgen そのまま、つまり関数オブジェクトを代入する(図 15)。

```
関数名.clone = 関数名_kgen;
```

図 15 clone メソッドの追加

5.3 関数宣言文以外で定義された関数の変換

関数リテラルで宣言された関数の変換方法は関数宣言文で宣言された関数の場合とほぼ同じである。Function コンストラクタで宣言された関数に関しては、一旦変換メソッドを通し、関数宣言文で宣言された関数にしてから変換を行うようにしている。ただし、Function コンストラクタの引数に文字列以外の値が渡された場合は実行時の値が変換時には不明であるため、そのまま出力する仕様となっている。

また、関数内にオウンブロックがない場合、無名関数はそのまま、有名関数は関数クローニングを可能にし、出力する。

5.4 その他の関数の変換

オブジェクト内関数、return 文で返される関数、関数引数に与えられた関数もサポートした。

オブジェクト内関数は、まずオブジェクトのプロパティ名を「プロパティ名_kgen」に直し、プロパティ値である関数は前述の通り変換を行う。変換オブジェクト宣言後にオブジェクトに元のプロパティ名(プロパティ値はプロパティ名_kgen を実行したもの)を追加するようにしている(図 16)。

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
var obj = {x : 1, y : 2,
          f : function() {
               @own cnt = 0;
               return (cnt += 1);
           }};
obj.f();
↓変換後
var obj = {x : 1, y : 2, f_kgen : (function() {
  var cnt = 0;
  return function() {
    return (cnt += 1);
  };
})};
obj.f = obj.f_kgen();
obj.f.clone = obj.f_kgen;
obj.f();
```

図 16 オブジェクト内関数の例

return 文で返される関数(図 17)、関数引数に与えられた関数(図 18)の場合も同様に変換するようにしている。ただし、それらの関数が有名関数の場合、関数クローニングを行えるようにしなければならないため、先に関数を認識させ、そこに clone メソッドを追加するようにしている。関数の前行、return 文の前行に変換済み関数を出力し、関数名を return 文、関数引数に与えるという処理を行っている。

```
return function f() {
  @own cnt;
  return (cnt += 1);
};
↓変換後
var f_kgen = (function() {
  var cnt;
  return function f() {
    return (cnt += 1);
  };
});
var f = f_kgen();
f.clone = f_kgen;
return f;
```

図 17 return 文で返される関数の例

```
$("#btn1").bind("click",function f() {
  @own cnt;
  cnt += 1;
  $("#disp").text(cnt + " times.");
});
↓変換後
var f_kgen = (function() {
  var cnt;
  return function f() {
    cnt += 1;
    $("#disp").text(cnt + " times.");
  };
});
var f = f_kgen();
f.clone = f_kgen;
$("#btn1").bind("click", f);
```

図 18 関数引数に与えられた関数の例

5.5 開発結果

プリプロセッサは JavaScript を使用し、開発した。このプリプロセッサによってオウン変数、関数クローニングを実現できた。　実際にはプリプロセッサを組み込んだ変換用 Web ページ(図 19)を用意した。

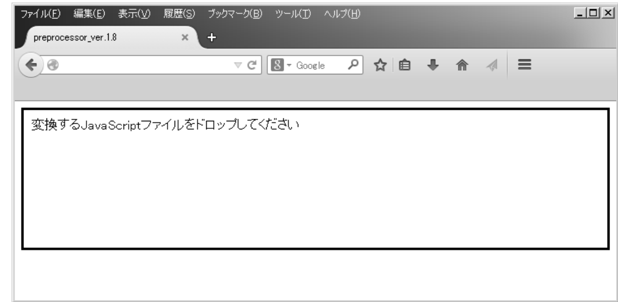

図 19 変換用 Web ページ

変換の際はページ上にあるドロップ領域に拡張関数を用いたプログラムのソースファイルをドロップすることで変換を行うことができる。変換済みのソースファイルは、ページ上に生成されるリンク(図 20)よりダウンロードすることができる。

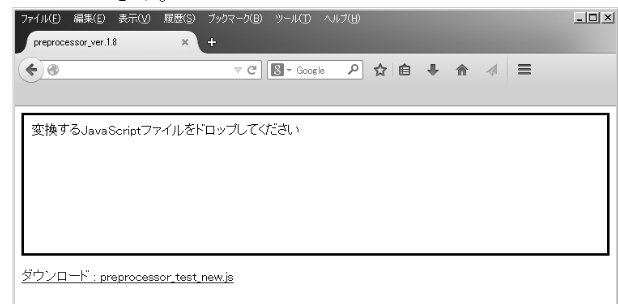

図 20 変換用 Web ページ(ソースファイルドロップ後)

6. 考察と今後の課題

関数定義の自然な拡張としてオウンブロックを追加したことによって、一般の JavaScript プログラマでも、そのままでは難度が高いオウン変数を、容易に宣言し扱えるようになった。さらに、オウン変数を含む関数のクローニング機能を追加したことにより、インスタンスとしては異なるが同一処理内容の関数を複数個所で利用できるようになった。

本論文では、JavaScript におけるオウン変数の追加とクローニングという関数機能の拡張だけを述べた。これによりクローニングを繰り返すことにより同様の関数を作りだすことができるようになる。関数だけでなく一般のオブジェクトについてもクローニング機能を持たせることで、従来のコンストラクタによる継承やプロトタイプチェインによる継承とは異なる新たな継承パラダイムを導入できるのではないかと我々は考えている。ただし、関数の場合とは異なりクローニングの方法についてより緻密な考察が必要である。その仕様と実現方法が今後の課題である。

参考文献

[1] ECMAScript Language Specification - ECMA-262 Edition 5.1、http://www.ecma-international.org/ecma-262/5.1/

[2] Douglas Crockford、JavaScript:The Good Parts -Unearthing the Excellence in JavaScript-、O'Reilly Media、2008

[3] Stoyan Stefanov、JavaScript Patterns -Build Better Applications with Coding and Design Patterns-、O'Reilly Media、2010

[4] SDS ALGOL 60 REFERENCE MANUAL、http://bitsavers.trailing-edge.com/pdf/sds/9xx/lang/900699C_Algol60_Ref_Nov66.pdf

[5] Nicholas C. Zakas、THE PRINCIPLES OF OBJECT-ORIENTED JAVASCRIPT、No Starch Press、2014

Copyright © 2015 by Information Processing Society of Japan and The Institute of Electronics, Information and Communication Engineers All rights reserved.