Web アプリケーションにおける JavaScript 計算の移送機構*

石橋 崇† 小宮常康‡ 廣津登志夫§
豊橋技術科学大学† 電気通信大学‡ 豊橋技術科学大学§

1 背景

近年，Ajax の普及により高度な機能を持った Web アプリケーションが増加している．Ajax はデスクトップアプリケーションに匹敵するインタラクション性および応答性を持つ Web アプリケーションを可能にしたが，我々は JavaScript に移送機能を持たせることにより，さらに高機能な Web アプリケーションを簡潔に構築できると考えている．例えば，クライアントとサーバのどちらでも行える処理をどちらで実行するかは性能に影響する重要な問題と考えられる．しかし，クライアントの性能およびネットワークの速度は様々であり，その分担を静的に決めることはできない．移送機能があれば任意のタイミングで計算をサーバで実行することが可能になる．

2 設計方針

移送機能を実現するにあたって，Web アプリケーションのプラットフォームに依存しないという利点を損なわないことを目標とした．したがって，言語処理系に依存しない移送方式を実現する必要がある．そこで，移送の実現に第一級継続 (first-class continuation) をプラットフォームに依存しない形で取得し，他の計算機に転送・実行させる方法を採った．JavaScript は継続を第一級データとして持たないが，そのような言語でもコード変換により処理系を変更せずに第一級継続を実現できることが知られている．この方法ならば既存の Web ブラウザに搭載された JavaScript 処理系を一切改良せずに移送を実現することが可能である．継続を採用するもう 1 つの理由としては，JavaScript プログラムから「計算」を柔軟かつ動的に扱うプリミティブ機能として適していると考えたためである．

また，移送機能を利用した Web アプリケーションでは別の場所へ移送した計算の結果をブラウザで受け取るといった利用方法が想定される．それを実現するには，物理的に離れた場所で実行される JavaScript 計算のプロセス同士が協調して作業を行える必要がある．そこで，移送を利用した Web アプリケーションの分散モデルを設計し，このモデルに対応する分散処理言語としての機能を JavaScript へ追加した．

本稿では分散処理言語として拡張した JavaScript を「JavaScript/MC^2 (**JavaScript** with Migratable first-**C**lass Continuation)」と呼ぶ．

3 JavaScript/MC^2

JavaScript/MC^2 では複数のプロセスがネットワークを介したメッセージ交換を行いながら分散処理を行う．プロセスは互いに実行コンテキストを共有しない独立した計算処理である．これはブラウザであれば 1 つの Web ページに相当する．プロセスは識別子としてプロセス ID を持つ．これはプロセスが実行されている場所（ノード）の識別子を含み，分散システム全体で一意に定まる．ノードはプロセスを実行する計算機であり，Web サーバもしくはブラウザが実行環境として存在している．このノードの識別には IP アドレスが使用される．メッセージの送信はプロセス ID を指定して行う．メッセージは非同期通信で送られ，各プロセスがもつ mailbox に到着順に保存される．

言語の特徴的として (1) 第一級継続の操作，(2) プロセスの生成，(3) プロセス間のメッセージ通信，の 3 つの機能を有する．プロセス生成は任意のノードで指定した JavaScript コードを実行する機能である．この機能と継続の取得を組み合わせて移送を実現する．具体的には第一級継続により計算の継続を取得し，その継続を再開するプロセスを他のノードで生成する．

次に JavaScript/MC^2 の言語機能の一部を説明する．

■継続 **callcc**(*fun*): 現在の継続を取得する関数である．Scheme の call-with-current-continuation (call/cc) と同等の機能を持っている．

■メッセージ通信 **send**(*id*, *msg*) / **recv**(): send は *id* で識別されるプロセスへメッセージ *msg* を送信する．recv はメッセージを mailbox から受信する．

■プロセス **self**() / **spawn**(*node*, *fun*): self はプロセ

*Migration facility on JavaScript for Web-based computation
†So Ishibashi, Toyohashi University of Technology
‡Tsuneyasu Komiya, The University of Electro-Communications
§Toshio Hirotsu, Toyohashi University of Technology

スIDを返す関数である．spawnは*node*中に新たにプロセスを生成し，そのプロセスで関数*fun*を実行する．戻り値は新規に作られたプロセスのIDである．

4 実装

移送機構は大きく分けてコード変換器，JavaScriptのライブラリ関数，サーバサイドシステムから構成される．コード変換器はJavaScriptコードを第一級継続を扱える形式に変換する．これは，移送機構とは独立したツールとして実装した．そのため，アプリケーションサービスを開始する前に，ソースコードを変換する必要がある．なお，変換方法はPettyjohnらの方法を参考にした[1]．

ライブラリ関数は前節で述べたようなプリミティブ関数群を提供する．ブラウザへ提供されるライブラリ関数の一部はWebサーバの特定のURLにアクセスすることでサーバサイドシステムと連携して動作する．これはブラウザの処理系に手をいれずに機能を実現するため一部の処理をサーバサイドで行っているためである．

サーバサイドシステムはspawnで渡されたコードの実行，mailboxの生成，通信のプロキシなどを行う．通信のプロクシはXMLHttpRequestオブジェクトのクロスドメイン制限を回避するため必要となる．また，ブラウザ上のプロセスのmailboxはサーバサイドに置かれる．これは，Webシステムがクライアントサーバモデルであるためブラウザ側でメッセージを待ち受けることができないためである．

5 記述例

JavaScript/MC^{2}は移送機能があるため，サーバサイドの処理もJavaScript/MC^{2}によって記述する方が自然である．そこで，最初にWebページにアクセスされたときのサーバサイドの処理はActive Server Pages (ASP) 形式の記述を利用する．以下に例を示す．

```
<%
  var server = spawn(
    'localhost',
    function() {
      for(;;){
        var req = recv()
        //メッセージは
        //{id:(プロセスID),cont:(継続)}と仮定
        //継続を実行し結果を id へ返す
        send(req.id, req.cont());
      }
    });
%>
<script>
...
function migrate(){
  callcc(function(k){
    send(<%= server %>, {id:self(), cont:k});
    display(recv());
    halt();
  });
}
function exec(){
  var me = self();
  //flag == true なら以降の処理をサーバで実行
  if (flag) migrate();
  var rval = some_caluclation();
  if (me == self()) //サーバかブラウザか判定
    display(rval);  //ブラウザなら結果を表示
  else
    return rval;    //サーバなら結果を return
}
</script>
...
```

サーバは記号<% %>で囲まれた部分を実行する．<% %>の外側はHTMLテキストとしてブラウザに転送される．このコードは flag によって関数 some_caluclation をサーバとブラウザのどちらで実行するか変更している．関数 migrate は呼び出された以降の処理を継続として取得し server が示すサーバプロセスへ送信する．そして，サーバで実行された継続の結果を受け取り関数 display へ渡している．ここで，display は渡された値を画面に表示する関数とする．サーバは画面への表示機能を持っていないため，継続として渡されるコードは実行されている場所をプロセスIDで判断している．

このようにJavaScript/MC^{2}のプリミティブ機能を組み合わせて動的な移送を実現することができる．

6 まとめ

Webアプリケーションで移送を扱うための言語，JavaScript/MC^{2}の設計および実装を行った．また，プロセスの生成やメッセージ通信をサポートするサーバシステムの実装も行った．

JavaScript/MC^{2}で提供するプリミティブ関数は移送を実現するには十分な機能を持つが，単純な機能しか持たないため実用的には複数の関数を組み合わせて使う必要がある．また，継続は強力な反面，熟練したプログラマでも扱うには難しい面がある．そのためプリミティブ機能を抽象化するためのフレームワークとしての機能が必要だと考えられる．フレームワークの設計・開発は今後の課題である．

参考文献

[1] G. Pettyjohn, J. Clements, J. Marshall, S. Krishnamurthi, M. Felleisen. Continuations from Generalized Stack Inspection. *ACM SIGPLAN ICFP 2005*, pages 216-227, September 2005