巡回するオブジェクトによるデータの収集

京都情報大学院大学　教授
渡邉 勝正

■ 概　要 ■

インターネットでは，様々な機能の Web アプリケーションや Web サービスが公開されている。それらを利用するには，定められた API（Application Program Interface）に従ってサービスの提供を求めることが必要である。それは，プログラムの面からは，遠隔メソッドの起動（RMI： Remote Method Invocation あるいは RPC： Remote Procedure Call）によるものである。

本論文では，まず，RMI の動きを，一つの例を挙げて述べる。それを基にして，RMI を 2 方向の RMI に拡張して，新しいサービス形態の可能性を探ると共に，使用に向けての問題点を明らかにする。その問題点を解消する一つの方策として，RMI を用いて，複数のサイトを移動して巡回するオブジェクトの実現を示す。これは，たとえば，複数の顧客における発注情報を集めてくる機能として適用できる。

■ 1. はじめに ■

インターネット上には種々の Web サービスあるいは Web アプリケーション（以降は，「Web サービス」とする）が開発されている。広く公開されているもの，または，対象とする利用者を限定して通知しているものがある。通知の方法は違っているかもしれないが，Web サービスと連携して利用する形式の根底には，RMI（または，RPC）がある。

RMI は，使用側（クライアント）が，遠隔のサーバ上にあるメソッド（または，関数）を，あたかも手元にあるかのように起動して，結果の値を受け取るものである。クライアントは，受け取った結果によって，別の Web サービスを呼び出すことで，自分のサイトに新しい機能を構築していくことができる。単純な手順であり，現在の Web サービスは，殆ど，この形に基づいて利用されている。

本論文では，それを次の二つに適用して，新しい Web サービスの開発の可能性を探ることを目指している。

（1）RMI を双方向にして，サーバが処理の途中で，クライアント側にあるメソッドを呼び出すことを可能にする。(RMI2)

（2）RMI の機構を利用して，オブジェクトが複数のサイトを巡回移動して，それぞれのサイトで，あらかじめ用意したメソッドを実行する。たとえば，サイトのファイルを読みだして，データを収集する。これは，以前には「移動エージェント」と呼ばれていたものである（たとえば，文献［1］）。ここでは，その動きを「巡回オブジェクト」として実現した例を取り上げる。それにより，複数のサイトでサービスを連続して提供する形態を実現することの基盤とする。

これら二つの方法がどのように応用できるかは別にして，Web サービスの実現の形態を拡張して，今後の Web サービスの発展を考える素材とする。

■ 2. 遠隔メソッドの起動（RMI：Remote Method Invocation）■

■ 2.1 RMIの手順 ■

RMI は，クライアント（C）とサーバ（S）間の通信のプログラム手順を，ソケットによる場合に比べて簡潔にしている。

(1s) 提供するメソッドをもつオブジェクトインスタンスを，サーバの名前とキーワード（合言葉）で結び付ける（rebind)。これは名前の登録になる。

(2c) サーバの名前とキーワードによって，メソッドをもつオブジェクトインスタンスを探し出す（lookup)。これは名前による検索で行える。

(3c) メソッドの入出力仕様を定義したインタフェースに基づいて，遠隔のメソッドを呼び出す。これは，手元のメソッドを呼び出すのと同じ形になる。

結果の型の変数 = オブジェクト . メソッド (引数のリスト) ;

(4s) 受け取った引数に対する処理をして，その結果の値を送り返す。これは，関数値を変数に代入する文で表現できる。

名前の登録(1s)と検索(2c)は，rmiregistry に任せる。メソッドの入出力仕様（3c, 4s）は，インタフェース（Interface）で定義する。これらの関係は，図 1 のように表せる。

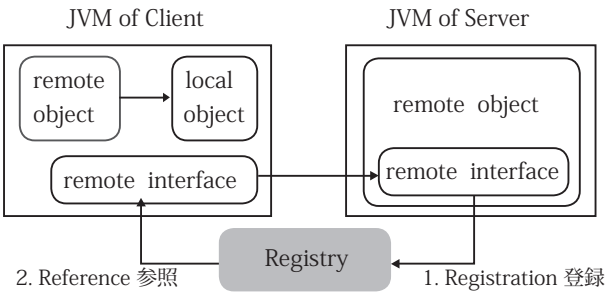

図1　名前の登録と参照　Fig.1 Name Registration and Reference

クライアントとサーバのつながりは，メソッドを含むクラス（class）から，rmic（RMI compiler）によって生成したスタブ（Stub）を介して行われる。

■ 2.2 RMIの例 ■

上述した手順を示す例として，次の仕様を持つメソッドの呼び出しを取り上げる。

「クライアントが指定した名前に対して，業績と生年月日を返す」プログラムの構成とそれぞれの役割を示す。
WhoIntf.java：メソッドのインタフェースを定義する。
WhoClient.java：名前を指定して，サーバに尋ねる。
WhoServer.java：名前から，その人の成果と生年月日を回答として送る。
Personal.java + BDate.java：人のデータをまとめたクラスと生年月日のクラス。
(0s&c) メソッドのインタフェースを定義する。
public Personal sayWho（String RecvMess）throws java.rmi.RemoteException;
(1s) サーバのインスタンス h を生成して，キーワード "who" で登録する。
```
WhoServer h ＝ new WhoServer（）；
Naming.rebind（"who", h）; //bind keyword
```
(2c) クライアントが約束したキーワードでサーバインスタンスを検索する。
```
WhoIntf h＝(WhoIntf)Naming.lookup("//"+args[0]+ "/who");
```
(3c) 実行引数に与えられたサーバ名（2c）と名前（たとえば，Turing）で要求を出す。
```
Personal message＝h.sayWho(args[1]+ " を紹介してください !")；
```
(4s) サーバが sayWho の中で，受け取った名前に対応するデータオブジェクトを返す。
```
Personal m1＝new Personal("Alan Mathison Turing","Turing Machine",
                new BDate(1912,6,23));
return(m1);
```
(3c+) クライアントは得られたPersonal型のデータを分解する。
実行結果は，たとえば，次のようになる。
サーバ側 ：
```
>javac WhoServer.java
>rmic WhoServer
>start rmiregistry
>java WhoServer 172.16.1.136
Fine ! Who Server is ready on   kcgi-sotec1
Please call me by：java WhoClient kcgi-sotec1
Receive Message：Turing を紹介してください !
```
クライアント側：
```
>java WhoClient 172.16.1.136 Turing
Start Who Client to Server：172.16.1.136
for Person：Turing
link to server 172.16.1.136
WhoClient 名前：Alan Mathison Turing
  his  id：Turing Machine
  his  生年月日：1912-6-23
```
クライアントの要求が満たされている。

■ 2.3 RMIの特長と問題点 ■

RMI はキーワードによって，クライアントとサーバを結び付けることができる。関数を呼び出す形式なので，プログラミング操作も簡単になる。

ただし，通信形態がメソッド呼び出しの形式に限定されているため，対話的に連続的なやり取りで機能を進めたい場合には工夫が必要である。

たとえば，第 1 のメソッド func1 の結果に基づいて，第 2 のメソッド func2 の機能を要求するとき，func1 と func2 の継続性が確保できるようにすることが求められる。

■ 3. 双方向遠隔メソッドの起動 (RMI2) ■

■ 3.1 RMI2の手順 ■

複数のメソッド呼び出しの継続性を確保する方法の一つとして，双方向の RMI を考える。これは，クライアントからの RMI の中で，サーバが処理の途中でクライアントの中のメソッドを呼び出す形態である。その手順は次のステップになる。図 2
(1) C→S：クライアントがサーバのメソッドfuncSを起動する。
(2) C←S：メソッドの中で，それまでの中間結果を用いて，クライアント側のあらかじめ約束されたメソッドfuncCを起動する。
(3) C→ S：クライアントがfuncCの結果をサーバに返送する。
(4) C← S：funcSは，funcCの結果を用いて処理を続けて，結果をクライアントに返す。これが最初のRMI(1)の結果になる。

サーバにおける(1)から(2)の間の処理と，(3)から(4)の間の処理が継続できる。

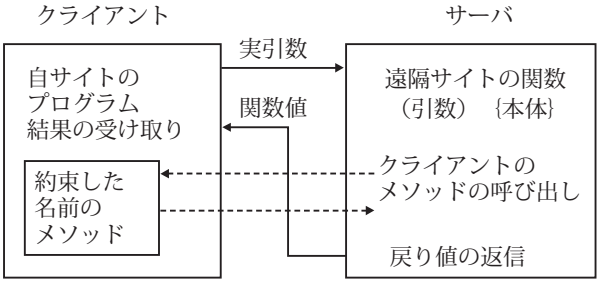

図 2　双方向遠隔メソッド起動 RMI2 の関係
Fig.2 Step of Bi-directional Remote Method Invocation

これを実現するために，次の対称的なペアが必要になる。
S：funcSへのキーワードとfuncSのスタブ(Stub)をクライアントに知らせる。
C：funcCへのキーワードとfuncCのスタブ(Stub)をサーバに知らせる。

また，双方のサイトで rmiregistry を起動して，提供するメソッドのキーワードの登録と検索に備える。

■ 3.2 RMI2の例 ■

クライアントとサーバが協調して「日時にあった挨拶を作る」動きをRMI2 で構成する。これは，文献［3］を基にして変更したものである。
(1) C → S：クライアントは，サーバに曜日（Weekday）または時刻（Time）を訊ねる。
(2) C ← S：サーバは曜日の番号または時刻をクライアントに知らせる。

(3) C→ S ：クライアントは曜日または時刻の日本語の挨拶をサーバに返送する。
(4) C← S ：サーバは，クライアントからの起動（1)の結果として，日時と挨拶を返す。
プログラムの構成とそれぞれの役割を示す。
(0s) ServerFuncInf：サーバメソッドのインタフェースを定義する。
　引数に，クライアントのIP を送って，サーバで検索する方法(cip)と，クライアントオブジェクトを送る方法（cob)がある。cip では，RMI 操作が対称的になる。
cip：public String ServerFunction（String cip, String WeekdayorTime）
cob：public String ServerFunction（ClientFuncInf pm, String WeekdayorTime）
(0c) ClientFuncInf：クライアントメソッドのインタフェースを定義する。
　public String ClientFunction(String Message, int WeekdayorTimeValue)
(1s) ServerImp：サーバのmain（）とServerFunction（）メソッドをもつ。
main()では、ServerImp のインスタンスを作り，名前を登録する。
　ServerImp oneServer ＝ new ServerImp（）;
　Naming.rebind（"toServer", oneServer）;　// 名前は"toServer"
ServerFunction（）では，前半の処理をして，クライアントのメソッドを名前”toClient” で検索して，前半の中間結果を付けて起動する。ここでは，引数にClientFuncInf pm を受け取る（0s)方法（cob)で行う。引数と検索結果を同じ名前（pm)にしている。
　//pm＝(ClientFuncInf)Naming.lookup("//" + cip + "/toClient");
　fromClient ＝pm.ClientFunction（WeekdayorTime, week);
クライアントから得た挨拶と日時を合わせて，後半の結果とする。
(2c) ClientImp：クライアントは，まず，自分が行うメソッドをもつクラス（3f)のインスタンスを生成して，約束した名前 "toClient" で登録する。
ClientFunctionImp onepm ＝
　new ClientFunctionImp("We response in English to your question");
Naming.rebind（"toClient", onepm);
次いで，名前”toServer” でサーバインスタンスを検索して，要求のメッセージを付けて起動する。引数に，上で生成したオブジェクトを含める。
ServerFuncInf cliServer ＝
　(ServerFuncInf)Naming.lookup("//" + args[0] + "/toServer");
String rmiResult=cliServer.ServerFunction(onepm, weekortime);
(3f) ClientFunctionImp：サーバから前半の処理結果を受けて起動されると，後半の処理へメッセージを送り返す。
　return（clientResult);
それを受けて，サーバは（1s)で後半の処理をして，(2c)の起動に対する結果を返す。
　クライアントサイトとサーバサイトには，それぞれServerImp_Stub.class，あるいは，ClientFunctionImp_Stub.classを伝えておく。
　たとえば，日本語の挨拶を返す実行結果の一部は，次のようになる。
サーバ側 ：

```
￥RMI2_Server>java ServerImp
ServerDate1 - 2010 年11 月8 日 13：09：34 GMT+09：00
in try of Server main - create ServerImp
ServerMain2 - bound in registry by toServer-key
+1+ ServerImp - RMI を受け入れる準備ができました
ServerMain3 - End of ServerImp main（)!
+4+ ServerFunction - RMI を受け入れました
+4+ ServerFunction - 引数はClientFuncInf pm です
ServerFunction - Request ：week
weekday number ＝ 1 daytime ＝ 13
+4b sendMessage - pm.ClientFunction を呼び出します
+4c from ClientFunction ：美しい月の月曜日 for 1
sendCallDate - 2010 年11 月8 日 13：10：19 GMT+09：00
```

クライアント側：

```
￥RMI2_ClientJapanese>java ClientImp 172.17.21.14 week
ClientMain1 - request ＝ week
+2+ ClientFunctionImp - インスタンスを作成しました
ClientMain2 + これから ServerFunction(onepm) を実行します
+2+ ClientImp - RMI to ServerFunction for week
ClientFunction get ：week with 1
send to Server >> 美しい月の月曜日 for 1
ClientMain3 - return from ServerFunction and ClientFunction
ClientRMI：2010 年11 月8 日13：10：19 GMT+09：00：美しい月の月曜日 for 1
ClientMain4 - End of ClientImp main（)!
```

　このClientFunction は，サーバからの起動に対して日本語で応答するが，別のクライアントでは，サーバからの起動に対して英語で応答させることができる。

```
￥RMI2_ClientEnglish>java ClientImp 192.168.0.2 time
クライアントの起動状態
ClientMain1 - link to ＝192.168.0.2
ClientMain1 - request ＝time
ClientFunction が返送したメッセージとServerFunction の結果
send to Server >> Good Morning for 8
ClientRMI：2010年11月14日 8:33:43 JST：Good Morning for 8
```

　このように，それぞれのクライアントは，サーバ機能ServerFunction の中間の結果に対して，ClientFunction の中で異なった対応をすることができる。
　RMI2 では，サーバに対して依頼の連続性を維持できるので，次のようなビジネスでの対話通信に適用できる。

	例 1：商品の発注	例 2：貨幣の変換
(1) C → S	商品見積りを依頼	2 国間の為替レートを問う
(2) C ← S	見積書を送付	レートの情報を通知
(3) C → S	発注する	貨幣の変換を依頼する
(4) C ← S	発送通知と請求書を送付	変換の通知と請求書を送付

■ 3.3 RMI2の特長と問題点 ■

クライアント側のメソッド名（ClientFunction）を約束しておくことにより，サーバ側の複数の機能を連続して起動する形態を実現できる。

ただし，サーバは，それぞれのクライアントの機能と連携するクラスのスタブ（Stub）を備える必要がある。

そのため，複数のクライアントが同時に依頼を出すことに対応するには，それぞれのクライアントへのスタブを識別することが要求される。

■ 4. 移動・巡回するオブジェクト ■

■ 4.1 オブジェクトの移動 ■

遠隔メソッド起動RMI では，クライアントが要求するときにサーバの機能を起動した。逆に，サーバ側が要求するときに，サーバのオブジェクトをクライアント側に移して，そこで機能を発揮することが考えられる。これについては，以前に，モバイル・エージェント（Mobile Agent）として研究が進められていた。たとえば，文献［1］［2］。

その一つの方法は，あらかじめエージェントオブジェクトを受け入れるサイト（Place）を予定しておいて，RMI によってオブジェクトをPlace に移すことである。

AgentSystem as＝（AgentSystem）Naming.lookup（place）; // 移動先のPlace

as.receiveAgent（ag, 2）; // 移動先の受け入れメソッドを起動する。

ここで，AgentSystem は，receiveAgent（）を定義しているインタフェースである。

このように，サーバからオブジェクトを順にサイトを移動させて，たとえば，各クライアントの状態情報や発注情報を集めることができる。

■ 4.2 巡回オブジェクトの動きと手順 ■

問題として，「クライアントの発注情報を，各 Place のファイルから取り出して集めてくる。」サーバ（オブジェクト）の働きを，一例として考える。ここでは，移動するオブジェクト（mobile object）を Mobje と略称する。

プログラムの構成：文献［1］の例を参考にして，それを，同じ名前で，拡張変更した。

（0）AgentSystem：Place が Mobje を受け入れるメソッド Place.receiveAgent（）のインタフェースを定義する。

（1）Place：AgentSystem インタフェースの実装クラスである。Mobje の送受信（send/receive）と保存取り出し操作（push/pop）のメソッドをもつ。

本例では，Kyoto, Osaka, Nara の３つの Place サイトを定めた。

（2）AgentKicker：引数に与えられた Mobje を , 新しいスレッドとして働かせる。

（3）Agent：Mobje の抽象クラスであり，Mobje と Place を結び付けるメソッドと，Mobje の累積データをもつ。たとえば，本例では次の変数を定義している。

public int cmax＝4;//MobjeがPlaceを移動する回数を指定する

public int count;//Mobjeの移動回数を数える count〈＝ cmax

public String trace ＝ new String（""）; // 移動したPlace の経歴を記録する

public String order＝ new String（""）; // 取り込んだ発注情報を累積する

（4）TestAgent：Mobje の具体的な実装クラスである。Mobje の初期設定（init）あるいは Mobje の本体動作（come），および，Mobje の移動の指定（go）のメソッドをもつ。Mobje が果たす機能を come（）で記述する。

（5）AgentStarter：新しい Mobje を作り出して , 指定した最初の Place に送り出す。このクラスは，Mobje を作り出すサイト（start）だけに必要で，各 Place（Kyoto, Osaka, Nara）にはいらない。

それぞれのサイトにPlace_Stubを持たせて，Place.receiveAgent()の実行を支える。

また，各サイトに発注データを記録したテキストファイルを準備しておく。

サイト間の制御の流れは，大まかに，次のようになる。

Step0：AgentStarterを実行するサイト（start）で，一つのMobjeを作り出して，最初の Place（たとえば，Kyoto）に送り出す。

Step1：Placeのサイトでは，受け入れた Mobjeを一旦キューに入れて，キューの先頭から Mobjeを取りだす。これは，複数のMobjeが巡回する場合に備えている。

Step2：Placeで取り出した Mobjeを，AgentKickerが新しいスレッドとして働かせる。

Step3：Mobje（本例では，TestAgent）の come（）で，定められた仕事を，現在の Place（サイト）で実行する。これがMobjeの役割を表している。

Step4：移動回数 countが，定めた回数 cmaxに達していなければ，Mobjeを次の Placeに移す。移動先で Step1から Step4が行われる。count〉＝ cmaxならば実行を終了する。

■ 4.3 巡回オブジェクトの実行例 ■

一つのMobjeが巡回を実行したとき，Mobjeを作り出すstart サイト，巡回の最初と最後のPlaceになるKyoto，途中のPlaceの Osaka，での表示結果の一部を示す。

［0］startサイトでの実行表示結果：

```
¥w3start>java AgentStarter TestAgent
 5,  AS1 - main(args[0]) TestAgent
10,  Ta1 - init() of TestAgent ：count ＝ 1
11,  Ta2 - init() and go(here, 41) here ＝ rmi:
     //172.17.21.39/Kyoto
13,  AS5 - sendAgent(iag, place) iag  ＝ TestAgent@15dfd77
14,  AS6 - sendAgent(iag, place) place ＝ rmi:
     //172.17.21.39/Kyoto
```

16,　AS8 - sendAgent to receiveAgent(iag, 1)TestAgent@15dfd77
[1] Place Kyotoサイトでの実行表示結果:
¥w3Kyoto>java Place Kyoto
 1,　PL1：main(args[0]) Kyoto
 4,..*PL7：wait in popAgent(ifirst) from Place.start()1
17,..*PL5：receiveAgent(ag, ifrom) AgentStarter
TestAgent@1a16869 ifrom = 1
trace = /* 最初にKyotoに来たとき, 発注データを取り出す */
 1:Kyoto1 + 2 <<rmi://172.17.21.39/Kyoto>>
for GetOrder local filename = OrderKyoto1.txt
K:Pnasonic Note PC, 20個;
K:サーバマシン DELL, 5台;
K:Hard Disk 4TB, 8台;
+++24+Ta3 + 2 到着時刻 - 2010年11月15日 11:45:10 GMT+09:00
 ,25*Ta4 - come() and go(here, 42) here = rmi://172.17.21.6/Osaka
trace =　 /* 二回目にKyotoに来たとき, 集めてきた発注データを表示する */
 1:Kyoto1 + 2 <<rmi://172.17.21.39/Kyoto>>
 2:Osaka1 + 3 <<rmi://172.17.21.6/Osaka>>
 3:Nara1 + 4 <<rmi://172.17.21.7/Nara>>
 4:Kyoto102 + 5 <<rmi://172.17.21.39/Kyoto>>
for GetOrder local filename = OrderKyoto102.txt
ORDER state =
rmi://172.17.21.39/Kyoto;
K:Pnasonic Note PC, 20個;
K:サーバマシン DELL, 5台;
K:Hard Disk 4TB, 8台;
rmi://172.17.21.6/Osaka;
O:Dell Desktop Machine, 12個;
O:サーバマシン DELL, 5台;
O:Hard Disk 4TB, 8台;
O:USB メモリ 64GB, 20個
rmi://172.17.21.7/Nara;
N:Note PC, 22個;
N:Hard Disk 4TB, 6台;
N:USB メモリ 64GB, 10個
rmi://172.17.21.39/Kyoto;
K:一巡してきました;
end of order
+++24+Ta3 + 5 到着時刻 - 2010年11月15日 11:45:11 GMT+09:00
 ,25*Ta4 - come() and go(here, 42) here = rmi://172.17.21.6/Osaka
 ,ee*Tae - in come() over count > cmax：4

最初の訪問では, 発注データを読み取り, 一巡した後の二回目の訪問では, 各サイトの発注データをまとめて表示している。

[2] Place Osakaサイトでの実行表示結果:
¥w3Osaka>java Place Osaka
17,28*PL5：receiveAgent(ag, ifrom) Place TestAgent@1a16869 ifrom = 2
trace = /* KyotoからOsakaに来た履歴を示す */
 1:Kyoto1 + 2 <<rmi://172.17.21.39/Kyoto>>
 2:Osaka1 + 3 <<rmi://172.17.21.6/Osaka>>
for GetOrder local filename = OrderOsaka1.txt
O:Dell Desktop Machine, 12個;
O:サーバマシン DELL, 5台;
O:Hard Disk 4TB, 8台;
O:USB メモリ 64GB, 20個
+++24+Ta3 + 3 到着時刻 - 2010年11月15日 11:45:10 GMT+09:00
 ,25*Ta4 - come() and go(here, 42) here = rmi://172.17.21.7/Nara

なお, たとえば, PlaceNaraでの発注データのファイルは次のものであった。

| N:Note PC, 22個
| N:Hard Disk 4TB, 6台
| N:USB メモリ 64GB, 10個

本例と同じPlaceを対象として, 巡回の経路が違う, あるいは, 違った動作をする移動オブジェクトMobjeを形成できる。たとえば, 上記の例のように, Kyotoから出発して発注を収集する(file read)Mobjeとは逆に, Osakaから出発して商品見積書を配布する(file write)Mobjeを構成できる。これは, 各Placeに商品カタログデータを残していく。たとえば, 各サイトのファイルReceivePost.txtに次のデータを残していく。

| 01:ノートPC, Dell2010, 45000
| 02:大容量USBメモリ, Sharp2011, 6800
| 03:新規サーバマシン, Fujitsu888, 128000
| end of 3-items 2010.11.16

■　4.4 巡回オブジェクトの特長と問題点　■

移動するオブジェクト Mobje は，約束されたサイト間で，オブジェクトを移動させて，Mobje がもっているメソッドを，移動先のサイトで実行させることができた。登録された会員，あるいは，顧客（クライアント）のサイトを周って，取り決めた機能を提供することができる。これによって，インターネットを介した新しい応用の可能性が広がることを目指している。

ただし，そのためには，Place に相当するサイトで，Mobje を受け入れるプログラムを準備して，定時に実行していることが必要である。クライアントが好きな時にサーバの機能を使う場合は，Web サービスを提供するサーバは，RMI の受け入れをできるように待機している。逆に，サーバ機能が移動あるいは巡回する場合は，Web サービスを受けるクライアントが，約束した時刻には稼働していて，待機している必要がある。

巡回の時刻に停止しているクライアントをスキップする方法が求められる。

本例で示したような，データの収集やデータの配布の他に，

クライアントの定期診断などのサービスが考えられる。

■ 5. おわりに ■

インターネット上の Web サービスあるいは Web アプリケーションを利用する基盤になる遠隔メソッド起動（RMI）に注目した。RMI の動きを明確にして，それを双方向 RMI に拡張すること（RMI2）と，RMI を用いてサイト間でオブジェクトを移動させること（Mobje）を導入した。RMI2 によって，一つのクライアントからの複数のメソッド起動に連続性を保つことができる。Mobje によって，サーバがクライアント側に移動してサービスを提供することができる。それぞれの適用に当たっては問題点もあるが，２つの形態を用いて，新しい Web サービスを開発することの可能性を探った。どのような機能を実現するかは，今後に取り組む興味ある問題である。

【参考文献】
1. 岩井俊弥 , “Java モバイル・エージェント ”, ソフト・リサーチ・センタ , 1998.
2.Jeff Nelson, “Programming Mobile Objects with Java”, John Willy & Sons, 1999.
3.Charles W. Kann,“Creating Components ： Object Oriented, Concurrent, and Distributed Computing in Java”, Auerbach Publication, 2003.

渡邉　勝正
Watanabe　Katsumasa
京都大学大学院工学研究科博士課程修了（数理工学専攻），工学博士。
元京都大学助教授，元福井大学教授，元奈良先端科学技術大学院大学教授，
社団法人情報処理学会フェロー。