JAIST Repository
https://dspace.jaist.ac.jp/

Title	ネットワーク上に静的な領域をもつプログラミング言語JavaPANの提案
Author(s)	鴨川, 雄; 由井薗, 隆也
Citation	電子情報通信学会論文誌 D, J88-D1(2): 326-329
Issue Date	2005-02-01
Type	Journal Article
Text version	publisher
URL	http://hdl.handle.net/10119/4731
Rights	Copyright (C)2005 IEICE. 鴨川雄, 由井薗隆也, 電子情報通信学会論文誌 D, J88-D1(2), 2005, 326-329. http://www.ieice.org/jpn/trans_online/
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

ネットワーク上に静的な領域をもつプログラミング言語 JavaPAN の提案

鴨川　雄[†]　由井薗隆也[††]（正員）

A Proposal of Programming Language JavaPAN with Static Network Scope

Takeshi KAMOGAWA[†], *Nonmember* and Takaya YUIZONO[††], *Member*

[†] 島根大学大学院総合理工学研究科，松江市
Interdisciplinary Graduate School of Science and Engineering, Shimane University, Matsue-shi, 690–8504 Japan
[††] 島根大学総合理工学部，松江市
Interdisciplinary Faculty of Science and Engineering, Shimane University, Matsue-shi, 690–8504 Japan

あらまし　本論文では，ネットワークで結合された複数の計算機を用いたアプリケーションプログラムの開発を支援するために，ネットワーク上に静的な領域をもつプログラミング言語 JavaPAN を提案する．ネットワーク上の静的な領域には，各プログラムがほかのプログラムと共有可能な資源を明示的に置くことができる．共有可能な資源は，変数と関数である．これにより，従来の遠隔手続き呼出しを用いたプログラミングに加えて，遠隔上の変数の直接的な操作ができるプログラミングも可能になる．

キーワード　ネットワーク公開領域，ネットワーク公開資源，端末資源，Java 言語，コンパイラ

1. まえがき

近年，マイクロプロセッサと高速ネットワークの発達により，ネットワークで結合した計算機群を用いて高度な情報サービスを提供する技術が広く普及している．したがって，そのようなサービスを実現するためのソフトウェア開発環境は重要である．分散処理環境下において，計算機間で行われる最も基本的な処理は，計算機間でのデータ通信処理である．分散処理プログラムの開発では，処理中で行われる外部との通信の回数や扱う送受信データの種類が少なくても，プログラム中の通信処理に関する記述を容易にする記法が望まれる．また，プログラム間で行われる処理内容を明確にする記法も望まれる．

従来のプログラム開発 [1] には，1 対 1 のデータ通信を支援するソケット API，そして手続き概念などを用いた抽象化記述が行える RPC（Remote Procedure Call）や RMI（Remote Method Invocation）[2] が用いられている．また，分散オブジェクト，移動オブジェクト，エージェントなどの研究が盛んに行われている．本研究では，Java 言語 [3] に対して「ネットワークでつながったプログラム間でアクセス可能な領域」を実装した拡張言語 JavaPAN（Public Areas on Network）を開発した．この言語では，プログラマは分散処理を構成するプログラム間で共有可能な資源（変数や関数）をネットワーク上の静的な領域に置き，それらの操作を通してプログラム間での通信を実現する．また，資源の操作には，専用のデータ通信ライブラリを用いず，ローカル上に存在する資源を操作するのと同様の記述を用いる．このような支援により，分散処理プログラムの開発において，従来のプログラムよりも処理の記述量を少なくすることと，プログラム間で行われる処理内容を明確にすることをねらう．

2. JavaPAN の設計方針

2.1 プログラムの構成

JavaPAN では，プログラマは作成するプログラムすべてに対して，固有の識別子とそれを走らせる計算機の IP アドレスとポート番号を専用のファイルに定義する．図 1 は，分散処理を構成するプログラム数が 4 個の場合のイメージである．

プログラムに与える識別子は，分散処理環境下で一意のプログラムを指すものとする．また，IP アドレスとポート番号は，各プログラムがもつ共有資源にアクセスする際に用いられる．

2.2 ネットワーク公開領域

ネットワーク公開領域（以下，ネット領域）とは，分散処理を構成するプログラム間で共有可能な資源を置くための領域である．この領域は，分散処理を構成するすべてのプログラムに与えられる．図 2 は，分散処理を構成するプログラム数が N 個である場合の各領域とネット領域との関係を表したものである．

2.3 プログラミング規則

・ネットワーク公開資源の宣言方法

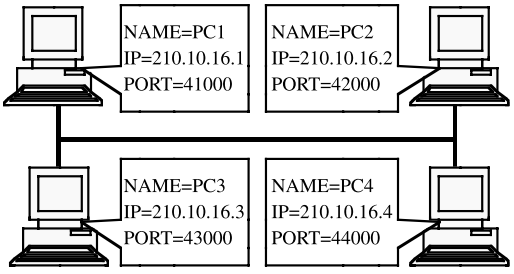

図 1　JavaPAN のプログラムのイメージ
Fig. 1　An image of JavaPAN programs.

 © (社)電子情報通信学会 2005

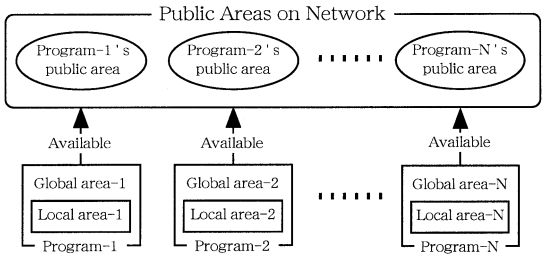

図 2　ネットワーク公開領域の概念
Fig. 2　Concept of public area on network.

ネットワーク公開資源（以下，ネット資源）とは，ネット領域上に置かれた資源のことである．ネット資源として宣言できるものは変数と関数であり，その型には基本型とクラス（ユーザ定義型も含む）の 2 種類が使用できる．また，ネット資源として宣言された変数をネットワーク公開変数（以下，ネット変数），関数をネットワーク公開関数（以下，ネット関数）と呼ぶ．ネット資源の宣言方法は，分散処理を構成する各プログラムの識別子と同じ名前のクラスを定義し，その中で修飾子が public であり static であるものがネット資源として認識される．つまり，ネット領域の実体は「分散処理を構成するプログラムがもつ大域領域の一部」であり，ネット資源の実体は「分散処理を構成する各プログラムがもつ大域領域上に置かれた一部の変数や関数の集合」である．

例として，図 1 に示した分散処理プログラムの一つである PC1 が整数型の変数 i と，戻り値が整数型で引数が整数型一つである関数 f をネット資源として宣言する場合，プログラマはクラス PC1 中で以下のように記述すればよい．

```
public final class PC1{
    ....
    public static int i ;
    public static int f ( int x ) {
        ....
    }
    ....
}
```

・ネットワーク公開資源の操作記述

ネット資源の識別子は，以下のように定義される．

```
＜ ネット資源の識別子 ＞
::= ＜ 資源を宣言したプログラムの識別子 ＞
  . ＜ 宣言時の識別子 ＞
```

プログラマは，この識別子を使ってネット資源を操作する．例として，図 1 に示した分散処理プログラムの一つ（PC1 も含む）が PC1 から宣言されたネット変数 i とネット関数 f を利用する場合，ソースコード中で以下に示すような処理を記述すればよい．

```
int x = PC1.i + 4 ;
int y = PC1.f(8) ;
```

最初の文は，PC1 がネット資源として宣言した変数 i がもつ値に 4 加算した値をローカル変数 x に代入する処理の記述である．次の文は，PC1 がネット資源として宣言した関数 f に引数 8 を与えて呼出し，その戻り値をローカル変数 y に代入する処理の記述である．

・ネットワーク公開資源の同期

ネット変数は，原則としてそれを利用しようとするプログラム単位で同期がとられるが，同期されたプログラム内のスレッド間では同期されない．しかし，型がクラスである場合には，synchronized-文 [3] を用いることでスレッド単位での同期も可能である．

また，ネット関数は原則として複数のプログラムから同時に使用可能である．しかし，ネット関数を定義する際に synchronized-修飾子 [3] を付けた場合には，その関数は複数のプログラムから同時に使用不可能となる．

3. 実 装 内 容

3.1 JavaPAN コンパイラ

JavaPAN は，Java 言語の拡張言語であるが，開発にあたって新たに追加された予約語や構文規則は全くない．**2.3** で触れたネット資源の宣言方法とその識別子も，Java の構文規則の範囲内である．また，JavaPAN コンパイラは，Java のソースコードとそのオブジェクトコードから，分散処理環境用に調整された Java のソースコードを目的コードとして生成する．コンパイラが行う処理 [4] は，コード解析，通信モジュールの生成と組み込み，そして目的コードの生成である．

このコンパイラの開発は，Java 言語を用いて行った．コンパイラの大きさは，全体で約 7000 行であり，その中でコードの解析部分は約 2400 行，通信モジュールの生成部分は約 2400 行，目的コードの生成部分は

約 1600 行である．

3.2 コード解析

コード解析では，与えられた Java のソースコードとそのオブジェクトコードを解析し，ソースコード中からネット資源を操作する処理の記述を見つけ，操作対象であるネット資源の識別子とそれに対する操作内容（ネット変数の場合は値の参照や書換え，ネット関数の場合はその呼出し）に関する情報を集める．

3.3 通信モジュールの生成と組込み

通信モジュールは，外部のプログラムがもつ資源を操作するためのプログラムである．このプログラムは，コード解析によって集められた情報から生成される．通信モジュールの種類は，クライアントプログラムとサーバプログラムの二つである．

クライアントプログラムは，操作対象の資源をもつプログラムと同じ名前をもつクラスである．このクラスは，操作対象のネット変数の値を操作する命令とその値をローカル上に取り込むためのバッファ，そして呼出し対象のネット関数を呼び出すための遠隔手続き呼出し（RPC）をもつ．これらの命令やバッファ，遠隔手続き呼出しのことを端末資源と呼ぶ．

操作対象のネット変数に対しては，それと同じ型の変数（バッファ）と対応するネット変数の値をその変数に取り込む命令，対応するネット変数の値をその変数の値で書き換える命令をもつ．この変数のことを，端末変数と呼ぶ．また，呼出し対象のネット関数に対しては，それと同戻り値，同引数の関数をもつ．この関数は，呼出し対象の関数に対応した遠隔手続き呼出しである．この関数のことを，端末関数と呼ぶ．このプログラムは，ネット資源を操作しようとするすべてのプログラムに組み込まれる．

サーバプログラムは，クライアントプログラムが実行した命令に対応した処理を行う．処理の種類は，指定されたネット変数の値の返信や値の書換えと，指定されたネット関数を呼び出し，その戻り値を返信することである．このプログラムは，ネット資源を所有するすべてのプログラムに組み込まれ，そのプログラムに与えられている IP アドレスとポート番号で接続待ちする．

3.4 目的コードの生成

JavaPAN コンパイラは，ネット資源の識別子を含む Java のソースコードから分散処理環境用に調整された Java のソースコードを生成する．このコードは，もともとのコード中で，ネット資源を操作する記述を，

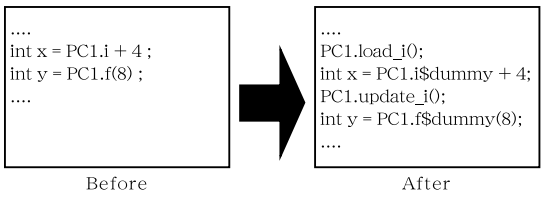

図 3 目的コードの生成例
Fig. 3 An example of generated object code.

表 1 プログラム作成の比較
Table 1 Comparison of JavaPAN and JavaRMI.

		JavaPAN	JavaRMI
画像処理	ソースコード数	317 行	547 行
	全体処理時間（ワーカ 1 台）	62 秒	47 秒
	全体処理時間（ワーカ 2 台）	35 秒	26 秒
	全体処理時間（ワーカ 3 台）	27 秒	22 秒
チャット処理	ソースコード数	97 行	171 行
	応答速度	問題なし	問題なし

分散処理環境用に調整したものである．

ネット変数を操作する処理の記述に対しては，その前後に操作対象のネット変数の値を端末変数に取り込む命令とネット変数の値を端末変数の値で書き換える命令を挿入し，そのネット変数の識別子を対応する端末変数の識別子に書き換える処理を行う．また，ネット関数を呼び出す処理の記述に対しては，そのネット関数の識別子を対応する端末関数の識別子に書き換える処理を行う．図 3 は，ソースコード中でネット変数 PC1.i とネット関数 PC1.f を利用する記述が見つかった個所を，分散処理環境用に調整した例である．

4. 評　　価

本論文では，開発した JavaPAN 言語用のコンパイラを評価するために，JavaRMI を用いて開発された分散処理プログラムと同等なプログラムの作成を試みた．作成したプログラムは，ワーカ複製形式を用いたマンデルブロー集合の画像処理プログラム [5] とマルチユーザ対応のチャットプログラムの 2 種類である．表 1 は，JavaPAN と JavaRMI で作成された分散処理プログラムの大きさ（行数）と処理時間の比較結果である．この結果から，以下に示すことが分かった．ただし，画像処理の結果は，800×640 の画像を 10 分割してワーカに処理させたものである．

（1） JavaPAN で開発されたプログラムのシステム記述量（行数）は，JavaRMI で開発されたものより少なかった．

（2） JavaPAN で開発されたプログラムは，

JavaRMI で開発されたプログラムよりも処理が遅かった．

5. 考　　察

JavaPAN 言語では，外部のプログラムが所有する資源を操作する方法として専用のデータ通信処理ライブラリを用いるのではなく，大域領域よりも大きな領域であるネットワーク公開領域という概念を導入した．そして，その領域上に置かれた資源を操作する処理の記述に，ローカル上の資源を操作する処理の記述と同じものを採用した．これにより，通信専用の処理の記述や命令を用意する必要がなくなったため，プログラムの大きさが小さくなったと考えられる．

従来の手続き概念による抽象化を用いた RPC によるプログラミングの場合，遠隔上に存在するプログラムがもつ関数を，ローカル上に存在する関数を呼び出すのと同じ方法で行うことが可能になる．しかし，遠隔上に存在するプログラムがもつ変数の操作に関しては，その変数名を用いてローカル上に存在する変数を操作することと同じ方法で行うことはできない．同様な処理の実現には，専用の手続きを用意する必要がある．

また，JavaPAN を用いて作成された画像処理プログラムが JavaRMI を用いて作成されたものよりも処理時間が長い原因は，通信処理におけるオーバヘッドが JavaRMI で作成されたものよりも大きいことにあると考えられる．

JavaPAN 言語で作成されたプログラムは，基本的には開発時に与えられた IP アドレスをもつ計算機上に束縛されるため，動的なシステムの構築には向いていないと考えられる．しかし，タップル空間 [1], [5] のような疎な結合による通信を行う場合には，システムの動的な変化が可能である．

ほかの研究においても，JavaPAN と同様に分散処理用に Java 言語を拡張したものに mJava/LR [6] と MetaVM [7] がある．mJava/LR では，do-at または do-in という構文によって，限定的にネットワーク上で実行したいコードを指定する機能がある．また，metaVM では，遠隔上のオブジェクトをローカルオブジェクトと全く同様に扱うことができる機能がある．これら二つは，Java の構文規則や実行環境である JVM を改良したものであるため，習得するには Java 言語の知識に加えていくつか覚えなくてはならないことがある．一方，JavaPAN は一部の変数や関数に対してネットワーク上での識別子が追加されただけである．

6. む す び

本論文では，分散処理環境下で走らせることを想定したプログラム言語 JavaPAN を開発した．JavaPAN では，ネットワークでつながったプログラム間でアクセス可能な領域を導入した．これにより，プログラム間で行われる通信処理について，プログラマ側で通信処理専用の命令を定義することなく作成することを可能にした．その結果，今回評価対象としたプログラムにおいて，処理の記述量を少なくできることが分かった．

今後は，第三者によるネットワークプログラム作成の評価と，開発するネットワークサービスに対応した生成コードの最適化を検討する予定である．

文　　献

[1] A.S. Tannenbaum, M.V. Steen, 分散システム原理とパラダイム，ピアソン・エデュケーション，東京，2003.

[2] Sun Microsystems, Java Remote Method Invocation Specification JDK 1.2, 1998.

[3] K. Arnold, J. Gosling, and D. Holmes, プログラミング言語 Java 第 3 版，ピアソン・エデュケーション，東京，2001.

[4] 中田育男，コンパイラの構成と最適化，朝倉書店，東京，1999.

[5] E. Freeman, S. Hupfer, and K. Arnold, JavaSpaces Principles, Patterns, and Practices, Addison Wesley, 1999.

[6] 渡辺昌寛，伊藤貴康，“場所の概念を備えた Java 言語とその処理系，” 情処学論，vol.40, no.SIG7(PRO4), pp.51–65, 1999.

[7] 首藤一幸，根山　亮，村岡洋一，“プログラマに単一マシンビューを提供する分散オブジェクトシステムの実現，” 情処学論，vo1.40, no.SIG7(PRO4), pp.66–79, 1999.

（平成 16 年 5 月 21 日受付）