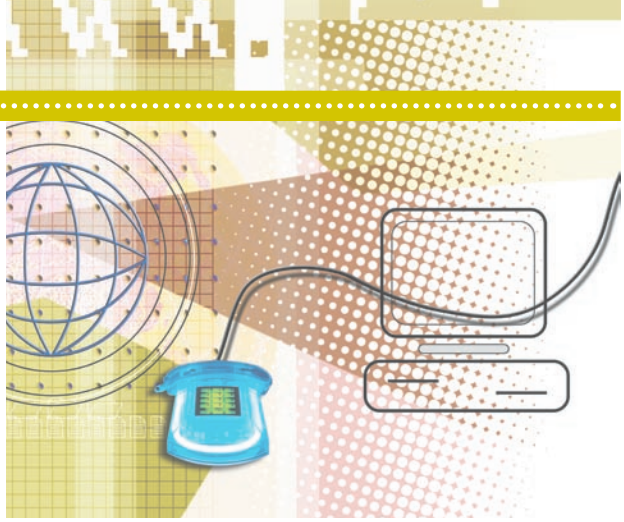
解説

プログラミング言語 Erlangの動向



井上　武　NTT未来ねっと研究所

はじめに

筆者の職場で，さまざまなプログラミング言語によるフィボナッチ数列の計算速度を比較したことがあった．指定された長さの数列が得られるまでの時間を測定したところ，驚いたことに，C言語を抑えて最速の座を勝ち取ったのはErlang（アーラン）☆1であった．怪訝に思われた読者のために種明かしをする．計算に使われたコンピュータは，クアッドコアCPUを2つ搭載していた．Erlangは，この8つのコアを使って，フィボナッチ数列を「並行して（concurrently）」計算していたのである．もちろん，他のプログラミング言語でも並行処理は可能であるが，ここでのポイントは，Erlangではそれをとても簡単に記述できる点にある．

Googleトレンドによると，日本でErlangが注目を集め始めたのは，2007年春のようである．Rubyの生みの親であるまつもとゆきひろ氏のブログで紹介されたことがきっかけと思われる．その記事[1]では，「次世代言語の本命」として，Erlangが次のように紹介されている．

- 次にくるトレンドは「関数型」と「並列」
- 両方を押さえたErlangが本命．歴史も信頼性もあり，知名度上昇中

ここには「並列」というキーワードが現れている☆2．

ここ数年，プロセッサのクロック向上は頭打ちとなり，半導体メーカ各社の戦略は，複数のコア（マルチコア）による並列動作へと変化している．Erlangが注目を集め始めた時期が，マルチコアへの移行と一致しているのは偶然ではない．Erlang開発者であるJoe Armstrong氏の言葉を借りると，「プロセッサのコアが増えるにつれ，並行プログラムは速くなるが，逐次的なプログラムは（相対的に）遅くなる」のである[2]．理想的には，Erlangでは，N個のコアがあれば，プログラムはN倍速くなる．

筆者がErlangを使い始めた理由は，まさにマルチコアにある．以前は，select/poll関数を利用したイベント駆動プログラムを書くことが多かった．しかし，このアプローチでは1つのコアしか利用することができない．マルチスレッドを使えば複数のコアを利用できるが，後述するように共有メモリを利用するため危険が多く，できれば避けたい．そう考えていたときに出会ったのがErlangであった．

本稿では，Erlangの特徴や最近の動向について紹介する．プログラミングテクニックには触れないが，興味のある読者のために，文献3）を紹介しておく．

Erlangとは

Erlangは，1987年に通信機器メーカであるエリクソンで開発されたプログラミング言語である．名前の由来は数学者のAgner Krarup Erlangにあると言われているが，ERicsson LANGuageの略や，電話の通信トラフィック量の単位（erlang）をも連想させる．1998年からはオープンソースとして公開されている．

Erlangは，動的型付けの関数型言語である．ソースコードをコンパイルし，BEAMと呼ばれるErlang VM上でオブジェクトコードを実行する．HiPE（High-Performance Erlang）☆3と呼ばれるネイティブコンパイラも開発されている．Erlangシェルやescriptにより，ソースコードを直接実行することもできる．Linux，Mac OS Xを含むUNIX系OS，Windowsなど，多くのOSで動作する．Wikipediaによる Erlangの紹介を表-1に示す．

分散した電話交換機を想定して開発されたプログラミング言語であるため，並列処理や耐障害性に多くの配慮がなされている．次章では，それらについて説明する．

☆1 http://www.erlang.org/
☆2 「並行（concurrent）」と「並列（parallel）」は厳密には異なる概念を指すが，本稿ではErlang関連の文献にならい，読みやすさを優先して用語の統一を行わない．
☆3 http://www.it.uu.se/research/group/hipe/

パラダイム	マルチパラダイム，並列処理指向，関数型言語
登場時期	1987 年
開発者	エリクソン
型付け	動的型付け，強い型付け
主な処理系	Erlang
影響を受けた言語	Prolog
影響を与えた言語	Clojure, Scala
プラットフォーム	UNIX 系 OS, Windows, Mac OS X

表 -1　Wikipedia による Erlang の紹介

アクターモデル

Erlang による並行プログラミングについて述べる．

Erlang VM は，デフォルトでコア数と同じスケジューラを用意する [4]．それぞれのスケジューラは独立に動作し，キューで待機している実行可能プロセスを取り出して，実行する．十分に多くのプロセスがあれば，統計的な効果により，遊んでいるコアがなくなる．

Erlang が採用しているプログラミングスタイルはアクターモデル [5] と呼ばれ，多くのプロセスを生成し，メッセージ交換によって協調動作させる．アクターモデルで記述されたプログラムを上で述べたスケジューラによって実行すると，マルチコアを有効に活用できる．これが，コアが増えるにつれ，Erlang のプログラムが速くなる原理である．

Erlang では，アクターモデルを実現するために，プロセスやメッセージ，変数に工夫を凝らしている．

プロセス

Erlang のプロセスは，OS のプロセスやスレッドとは異なり，Erlang VM によってスケジュールされる．共有メモリを持たないため，スレッドではなくプロセスと呼ばれる．

Erlang のプロセスはとても軽量であり，オーバヘッドはデフォルトで 309 ワード（32bit アーキテクチャであれば約 1.2KB）しかない．また，起動にかかる時間も数マイクロ秒と短い．アクターモデルのように多くのプロセスを用いるスタイルであっても，性能の劣化を抑えることができる．実験的ではあるが，4,000 万プロセスを起動した例もある [6]．

メッセージ

伝統的な UNIX プログラミングにおいて，複数のプロセスを起動してメッセージ交換を行うには，次のように複雑な手順が要求される．まず，fork 関数によりプロセスを起動し，関数の戻り値でプロセス種を区別する．何らかの方法で宛先プロセスへの識別子（ファイルディスクリプタなど）を取得し，必要に応じて接続を確立する．最後に，データを直列化して，送信する．受信プロセスは，データを復旧し，構造を調べ，適切な処理へと振り分ける．MPI（Message Passing Interface）[☆4] という並列コンピューティングのための標準もあるが，基本的な手順はよく似ている．

これに対し，Erlang のメッセージ交換は簡潔に抽象化されている．プロセスに種類はなく，区別は不要である．宛先プロセス ID あるいは名前を指定し，データを直列化せずにそのまま送ればよい．受信プロセスでは，データに対してパターン照合を行い，直観的に処理を振り分ける．

図 -1，2 に，値の増減を行うクライアント・サーバの例を示す．クライアントは，宛先プロセス ID（Pid）に対して，"!" によってメッセージ（{add, 3} と {subtract, 1}）を送信する．宛先プロセスに名前が与えられているときは，プロセス ID の代わりに名前を指定してもよい．このような簡潔な抽象化により，メッセージを送信できる．なお，メッセージは非同期に送信される．

サーバは，メッセージを受信すると，パターン照合により条件分岐を行う．この例では，メッセージの第 1 項によって照合が行われ，add であれば第 2 項の値（N）を加算し，subtract であれば減算する．Erlang のパターン照合は任意のデータ型に対して適用でき，特定のビットや数値の範囲を区別することも可能である．この柔軟かつ強力なパターン照合により，メッセージ受信処理を簡潔に記述できる．

メッセージ交換のオーバヘッドは適度に最適化される．たとえば，同じ Erlang VM のプロセスを宛先とするときは，大きなバイナリデータを送信しても実体はコピーされず，参照のみが渡される．

このメッセージ交換方法は，宛先プロセスが異なるコンピュータにあってもまったく同様に記述できる．直列化は不要であり，エンディアンの違いは隠蔽される．つまり，Erlang のプログラムは，ほとんど変更を伴わずに分散環境へと拡張することができる．

変数

マルチスレッドによる並行プログラムは，スレッド間で情報をやりとりするために，変更可能な共有メモリを利用することが多い．このモデルでは，あるプロセスが誤ったデータを書き込むと，他のプロセスがクラッシュする可能性がある．よって，ロックなどの排他制御メカ

☆4 http://www.mpi-forum.org/

```
1 > Pid = spawn(calc_server, loop, [0]).    % サーバの起動
<0.36.0>
2 > Pid ! {add, 3}.                         % 加算メッセージの送信
New value is 3
3 > Pid ! {subtract, 1}.                    % 減算メッセージの送信
New value is 2
```

図 -1　クライアントによるメッセージ送信の例（Erlang シェル）

```
-module(calc_server).
-compile(export_all).

% loop 関数の定義
loop(Value) ->
    NewValue =
        receive                                  % メッセージの受信とパターン照合
            {add,      N} -> Value + N;          % 値の加算
            {subtract, N} -> Value - N           % 値の減算
        end,
    io:format("New value is ~p~n", [NewValue]),  % 更新した値を出力
    loop(NewValue).                              % 再帰呼び出し
```

図 -2　サーバの例

```
fac(0) -> 1;                % 引数が 0 のとき
fac(N) -> N * fac(N-1).     % 引数が 0 以外のとき
```

図 -3　階乗の例

ニズムを利用して，競合を回避しなければならない．しかし，経験のある読者はよく理解できると思うが，間違いを犯さずに排他制御を行うことはきわめて難しい．

Erlang は関数型言語として設計されている．変数への代入は 1 回限りで，値の変更はできない．たとえば，N=N−1 のような代入は許可されない．このため，繰り返し処理を行うときは，for ループのようにカウンタ変数を増減させるのではなく，引数を変化させながら関数を再帰的に呼び出す．図 -3 は階乗を計算する例である．

変数の値が不変であるため，排他制御は不要である．Erlang には共有メモリの危険がなく，並行処理を安全に記述できる．

その他の特徴

Erlang は，並行処理以外にも魅力的な特徴を多く揃える．それらについて，簡単に説明する．

再帰呼び出し

前章で述べたように，Erlang では関数の再帰呼び出しによって繰り返しを実現する．このため，再帰呼び出しの効率は重要である．図 -2 に示したサーバプロセスは，メッセージを受信するたびに再帰呼び出しを行い，終了することがない．一見すると，呼び出しスタックが無限に成長するようにみえる．Erlang では，末尾再帰（関数の最後で行われる再帰呼び出し）にすることにより，スタックが積まれなくなる．サーバプロセスや階乗の例のように，ほとんどの再帰呼び出しは末尾再帰として記述できるため，スタックの問題はなくなる．

耐障害性・保守性

Erlang の異常処理は簡単である．プロセスが異常終了すると，あらかじめ指定しておいたプロセスに終了シグナルが発生する．このとき，異常が発生したプロセスは対処することなく，終了してしまってかまわない．終了シグナルの発生したプロセスで異常処理を行う．処理を行うプロセスは，異なるコンピュータにあってもよい．この遠隔処理モデルにより，Erlang は高い耐障害性と保守性を実現している．

また，プログラム実行中にコードを交換できるという特徴があり，システムを停止することなく障害に対処できる．

OTP

OTP（Open Telecom Platform）は，Erlang とともに配布されるライブラリである．サーバやプロセス監視，データベースなど，ソフトウェア開発に欠かせない機能が数多く含まれている．完成度は高く，動作は安定して

モジュール名	説明
gen_tcp	TCP/IP ソケット
gen_server	サーバ
gen_fsm	有限状態マシン
Supervisor	プロセス監視
Ets	メモリストレージ
Dets	ディスクストレージ
Mnesia	データベース
Sasl	アラーム，エラーロガー

表 -2　主な OTP モジュール

製品・サービス名	説明
Ericsson AXD 301	ATM スイッチ
Ericsson GPRS system	携帯電話パケットデータ通信装置
Nortel SSL accelerator/ VPN	SSL 関連装置
Tera Bank credit card system	クレジットカード決済システム
T-mobile SMS system	ショートメッセージシステム
Amazon SimpleDB ☆5	分散データベース

表 -3　Erlang を用いた製品・サービス名の例

いる．表 -2 に主な OTP モジュールを示す．英語ではあるが，ドキュメントも整備されている．

OTP の特徴の 1 つに，デザインパターンを実装するためのモジュールが含まれている点が挙げられる．これらは，behaviour モジュールと呼ばれる．このモジュールを利用すると，アクターモデルに準じた設計が容易になる．表に示したモジュールのうち，gen_server, gen_fsm, supervisor などが behaviour モジュールである．

OTP は，Erlang の一部といえるほど密接な関係にある．Erlang の配布ファイル名は otp_src_<version>.tar.gz であり，Erlang ではなく OTP を冠している．また，後述の技術者会議の正式名称は，Erlang/OTP User Conferences である．

Erlang を取り巻く状況

製品・サービス

冒頭で引用したまつもとゆきひろ氏の言葉にもあるように，Erlang は歴史と信頼性のあるプログラミング言語である．欧米ではすでに一定の地位を得ているように思われる．英国には，Skills Matter Ltd. や Erlang Training and Consulting Ltd. など，Erlang に関するサポートサービスを提供する企業がある．後述の Erlang eXchange というイベントは，これらの企業により主催された．

プロジェクト名	説明
Apache CouchDB ☆6	ドキュメント指向データベース
Apache Thrift ☆7	RPC フレームワーク
ejabberd ☆8	インスタントメッセージングサーバ
Yaws ☆9	Web アプリケーションフレームワーク
Mochiweb ☆10	Web アプリケーションフレームワーク
Kai ☆11	分散データストア
ermlia ☆12	分散ハッシュテーブル

表 -4　Erlang によるオープンソースプロジェクトの例

開発元であるエリクソンをはじめ，多くの企業が製品・サービスの開発に Erlang を用いている．表 -3 にいくつか例を示す．製品・サービスは多岐にわたるが，Erlang の高い並行性を活かした，多数の接続を行うようなネットワークサーバが多い．

オープンソースプロジェクト

Erlang によるオープンソースプロジェクトは少なくない．表 -4 にその例を示す．

Yaws や Mochiweb など，Web サーバに関連するソフトウェアが多いのは，いわゆる C10K 問題を反映してのことと思われる．C10K とは，10,000 クライアントからのアクセスをどのようにして受け付けるか，という問題である．通常，Web サーバはクライアントごとにスレッドやプロセスを割り当てるが，オーバヘッドが大きいため，10,000 ものクライアントをさばくことは難しい．Erlang のプロセスはオーバヘッドが小さく，また並行処理に適したプログラミングモデルを採用しているため，C10K 問題を解決できるとして期待されている．

また，製品・サービスで紹介した Amazon SimpleDB をはじめ，分散データストアにもよく利用されている．分散データストアでは，コミットプロトコルなどで非同期通信が多用される．Erlang はこれらのメッセージを簡潔に記述できるため，採用されたと考えられる．

日本国内での利用も徐々に増えつつある．KLab（株）は，ejabberd プロジェクトに貢献するとともに，それを利用したインスタントメッセージングサービスを提供

☆5 http://aws.amazon.com/simpledb/
☆6 http://couchdb.apache.org/
☆7 http://incubator.apache.org/thrift/
☆8 http://www.ejabberd.im/
☆9 http://yaws.hyber.org/
☆10 http://code.google.com/p/mochiweb/
☆11 http://kai.sf.net/
☆12 http://github.com/cooldaemon/ermlia/tree/master/

図 -4　Erlang 開発者会議の様子

している[☆13]．分散データストアの Kai は，日本発のプロジェクトであり，後述の日本ユーザコミュニティ活動の中心になっている．Kai は，ポータルサイト goo での導入が検討されている．

コミュニティ

欧米を中心に，コミュニティ活動は活発である．エリクソン本社のあるストックホルムでは，1994 年以降，ほぼ毎年，Erlang/OTP User Conference[☆14] という開発者会議が開催されている．また，2008 年には，Erlang eXchange[☆15] というイベントがロンドンで開催され，3 セッションが並列で行われるという盛況ぶりであった．2009 年は，サンフランシスコでも開催される予定である．

開発者メーリングリストでは，日々活発な議論が行われている．筆者が投稿したときには，多くの歓迎，提案，コードをいただき，暖かいコミュニティという印象を持っている．

日本のコミュニティ活動も活発になってきている．2008 年には 6 月と 11 月に有志による開発者会議が開催された．図 -4 は，6 月に（株）ミクシィで開催された会議の様子である．会場の都合により定員を 30 名に抑えなければならなかったが，募集開始後すぐに満員となるなど，Erlang への注目の高さが窺えた．2009 年も定期的に開催される予定である．なお，日本でのコミュニティ活動は，erlang-users.jp[☆16] にまとめられている．興味のある読者はこのサイトを参照されたい．

他のプログラミング言語からの注目も少なくない．Revactor[☆17] という，Ruby による Erlang 実装が開発されている．また，YAPC::Asia という Perl 開発者の国際会議では，Erlang を主題とした講演が行われた．

おわりに

本稿では，Erlang の特徴と現状について簡単に紹介した．Erlang は現在も熱心に開発が進められている．今後は，VM スケジューラの改良やユニコードのサポートなどが予定されている．プロセッサのマルチコア化が進むにつれ，製品・サービスへの利用やコミュニティ活動はますます盛んになると期待される．

謝辞　幾田雅仁さん，中居良介さん，橋本智哉さん，濱野司さんには原稿執筆にあたり有益な助言をいただきました．竹迫良範さんには開発者会議の写真を提供いただきました．この場を借りて，お礼申し上げます．

参考文献

1）まつもとゆきひろ：Matz にっき：「次」の言語，http://www.rubyist.net/~matz/20070418.html#p05 (Apr. 2007).
2）Armstrong, J. : Erlang Keynote : Armstrong on Software : Erlang & SMP, Erlang eXchange 2008 (June 2008).
3）Armstrong, J. : プログラミング Erlang, オーム社 (Feb. 2008).
4）Lundin, K. : Inside the Erlang VM, Erlang/OTP User Conference, Stockholm (Nov. 13, 2008).
5）Hewitt, C., Bishop, P. and Steiger, R. : A Universal Modular Actor Formalism for Artificial Intelligence, Proc. of IJCAI (1973).
6）Erlang Process - KLablabWiki, http://lab.klab.org/wiki/Erlang_Process (Feb. 2008).

（平成 20 年 12 月 19 日受付）

井上　武
takeru.inoue@ieee.org
1998 年京大・工・物理工卒業．2000 年同大学院工学研究科修士課程修了．2006 年同大学院情報学研究科博士後期課程修了．博士（情報学）．2000 年 NTT 入社．現在，NTT 未来ねっと研究所研究主任．オーバレイマルチキャスト，分散データベースなどの大規模分散システムの研究開発に従事．2001，2004 年度電子情報通信学会 IN 研究会研究賞受賞．2005 年 APCC Best Paper Award 受賞．

☆13 http://www.jabber.jp/
☆14 http://www.erlang.se/euc/
☆15 http://www.erlang-exchange.com/
☆16 http://erlang-users.jp/
☆17 http://revactor.org/