Xcrypt マニュアル

京大中島研 e-science グループ

2009 年 7 月 10 日

目次

■メモ 定義すべき用語ジョブスレッド

重要

Daniel Muey 氏の Recursive.pm を使用している．ライセンス的には問題ないはず.

第Ⅰ部

総論

第 1 章

機能

通常，ある一つの大きな処理を行うには，その処理を小分け（ジョブと呼ぶ）にし，それらの処理をジョブスケジューラに依頼する．具体的には，

1. ジョブスケジューラが理解できるスクリプトを作成し，
2. そのスクリプトをジョブスケジューラに渡し，
3. ジョブスケジューラが返す結果からまた別のスクリプトを作成し，それをジョブスケジューラに渡す，

という一連の操作を繰り返すというものである．

しかし，この方法は処理の繰り返しごとに人手による介入を要し，作業効率が悪い．そこで，人手で行うところを適当なスクリプト言語により記述することで自動実行を実現することが考えられる．Xcrypt はそのスクリプト言語として Perl を採用し，パラメタ指定によるジョブの生成や投入を Perl の関数として提供することで，ユーザがジョブ処理を容易に行うことを補助する．

TODO: サーチアルゴリズム等のアルゴリズムモジュールの提供についても記述する．

第 2 章

Xcrypt スクリプトの記述

Xcrypt は Perl の拡張である．よって，全ての Perl スクリプトは Xcrypt スクリプトである．しかし，Xcrypt はジョブに関する操作の補助を行うものと考えられ，Xcrypt スクリプトは典型的には以下の順に記述される．

1. モジュールに関する記述
2. 大域変数に関する記述
3. ジョブ定義ハッシュに関する記述
4. ジョブ処理に関する記述

2.1 モジュールに関する記述

モジュールは

```
use base qw(core);
```

と記述して読み込む．複数読み込む場合は読み込みたい順に

```
use base qw(dry core);
```

と記述して読み込む．利用可能なモジュールに関しては 4 章で述べる．

2.2 大域変数に関する記述

Xcrypt スクリプトの動作全体に関わる大域変数（例えば`$sge`）は

```
$sge = 1;
```

と記述してセットする．

2.3 ジョブ定義ハッシュに関する記述

Xcrypt ではジョブはハッシュ（ジョブハッシュと呼ぶ）で実現されている．ジョブを定義するにあたり，ハッシュ自身を記述してもよいが，ジョブ定義ハッシュと呼ばれるハッシュを記述すれば，ジョブを生成する際，さらにパラメタを与えることで多数のジョブを一度に生成できる．例えば，

```
%xyz = (
    'id' => 'job100',
    'exe' => './kempo',
    'arg0' => 'plasma.inp',
    'arg1' => '100',
    'copieddir0' => 'forcopieddir',
    'linkedfile0' => 'kempo',
    'copiedfile0' => 'plasma.inp',
    'stdofile' => 'hogeout',
    'stdefile' => 'hogeerr',
    'queue' => 'gh10034',
    'option' => '# @$-g gh10034'
);
```

と記述する．定義可能なジョブ定義ハッシュのキーに関しては 6 章で述べる．

2.4 ジョブ処理に関する記述

通常，人手で行う処理で，今回，Xcrypt に行わせたい処理について記述する．Xcrypt で利用可能な関数については 7 章で述べる．

2.5 記述例

本章で前節までの説明を踏まえたスクリプト記述例を以下に示す．

```
use base qw(limit core);

$limit::smph=Thread::Semaphore->new(10);
$sge = 1;

%xyz = (
    'id' => 'job100',
    'exe' => './kempo',
    'arg0' => 'plasma.inp',
    'arg1' => '100',
    'copieddir0' => 'forcopieddir',
    'linkedfile0' => 'kempo',
    'copiedfile0' => 'plasma.inp',
    'stdofile' => 'hogeout',
    'stdefile' => 'hogeerr',
    'queue' => 'gh10034',
    'option' => '# @$-g gh10034'
);

my @jobs = &prepare(%xyz, 'arg1s' => [2,4]);
my @thrds = &submit(@jobs);
my @results  = &sync(@thrds);

foreach (@results) { print $_->stdout , "\n"; }
```

第 3 章

ジョブ処理の流れ

実際のジョブ処理の流れについて概観する．

3.1 実行方法

環境変数 XCRYPT を Xcrypt をインストールしたディレクトリで定義する（ここでは /usr/share/xcrypt と仮定する）．シェルが bash なら，

```
$ XCRYPT=/usr/share/xcrypt; export XCRYPT
```

作業ディレクトリに移動する（ここでは $HOME/wd とする）．

```
$ cd $HOME/e-science/wd
```

Xcrypt スクリプト（2.5 節参照）を作成する（ここでは sample.xcr とする）．
実行する．

```
$ $XCRYPT/xcrypt sample.xcr
```

3.2 作成される物

Xcrypt を実行した際，作業ディレクトリ以下に作成される物について説明する．

ディレクトリ

Xcrypt はジョブごとにディレクトリ（ジョブ作業ディレクトリと呼ぶ）を作成する．ディレクトリの名前はジョブハッシュの id キーの値である．ジョブ処理はジョブ作業ディレクトリで行われる．

ジョブリンク・ジョブファイル

ジョブ作業ディレクトリからジョブハッシュの linkedfile0,...,linkedfile255 キーの値で指定されているファイルへシンボリックリンクを張り，copiedfile0,...,copiedfile255 キーの値で指定されている

作業ディレクトリ中のファイルのコピーをジョブ作業ディレクトリに作成する．

`nqs.sh`

ジョブスケジューラが NQS である際，NQS に渡されるスクリプトである．

`sge.sh`

ジョブスケジューラが Sun Grid Engine である際，Sun Grid Engine に渡されるスクリプトである．

`request_id`

Xcrypt によるジョブの投入に対し，ジョブスケジューラが返すジョブのリクエスト ID を格納する．

`stdout`

ジョブの実行コマンドの標準出力が格納される．ジョブハッシュの `stdofile` キーの値が指定されている場合，その値のファイル名で作成される．

`stderr`

ジョブの実行コマンドの標準エラー出力が格納される．ジョブハッシュの `stdefile` キーの値が指定されている場合，その値のファイル名で作成される．

第 II 部

各論

第 4 章

モジュール

4.1 core

コアモジュールを全て読み込む．Xcrypt 特有のもの（ジョブ定義ハッシュ等）を使用する場合には必ず読み込まないといけない．

4.2 dry

Xcrypt をドライモード（実行コマンドなしで）で動作させることができる．
Xcrypt スクリプトに

```
$dry::dry = 1;
```

と記述することで Xcrypt がドライモードで動作する．

4.3 limit

一度に投入されるジョブの数の上限を指定する．

```
$limit::smph=Thread::Semaphore->new(100);
```

第 5 章

大域変数

5.1 sge

ジョブスケジューラが Sun Grid Engine の場合は，Xcrypt スクリプトに

```
$sge = 1;
```

と記述することで Sun Grid Engine に対して Xcrypt を正常に動作させることができる．デフォルト値は 0 である．

5.2 separation_symbol

ジョブ作業ディレクトリ以下にジョブ定義ハッシュにおける id,arg0,...,arg255 を$separation_symbol で区切った名前のディレクトリを作成する．$separation_symbol のデフォルト値は「!」である．「-」に変えたい場合は，

```
$jobsched::separation_symbol = '-';
```

とする．

第 6 章

ジョブ定義ハッシュ

ジョブ定義ハッシュにおいて利用可能なキーについて紹介する．

6.1 id

実行されるジョブを識別する語を記述する．

6.2 exe

実行されるジョブの実行コマンドを記述する．後述の arg0,...,arg255 とともに

```
$ exe arg0 arg1 ...  arg255
```

といった形で実行される．

6.3 arg0,...,arg255

実行されるジョブの実行コマンドの引数を記述する．前述の exe とともに

```
$ exe arg0 arg1 ...  arg255
```

といった形で実行される．

6.4 linkedfile0,...,linkedfile255

この値のリンク名でジョブ作業ディレクトリから作業ディレクトリのファイルへシンボリックリンクを張る．

6.5 copiedfile0,...,copiedfile255

この値のファイル名でジョブ作業ディレクトリから作業ディレクトリにコピーをつくる．

6.6 stdofile

この値のファイル名で実行プログラムとジョブスケジューラの標準出力を格納する．空の場合は「stdout」というファイル名になる．

6.7 stdefile

この値のファイル名で実行プログラムとジョブスケジューラの標準エラー出力を格納する．空の場合は「stderr」というファイル名になる．

6.8 ofile

実行プログラムの出力ファイルを記述する．

6.9 odelimiter

実行プログラムの出力ファイルの行を区切る際の区切り文字を指定する．

6.10 ocolumn

実行プログラムの出力ファイルの列を指定する．

6.11 queue

実行するジョブを投入するキューの名前を記述する．

6.12 cpu

使用するコア数を指定する．

6.13 proc

使用するプロセス数を指定する．

6.14 option

ジョブスケジューラのオプションを記述する．

6.15 predecessor

ジョブを，値であるところのジョブたちの処理が終わるまで投入されず，

6.16 successor

ジョブ処理後，値であるところのジョブたちを生成する．

第 7 章

組込み関数

Xcrypt で利用可能な組込み関数のうち，Perl の組込み関数でないものについて紹介する．

7.1 prepare

ジョブ定義ハッシュとパラメタを受け取り，適当なジョブリファレンスの配列を返す．特に，ジョブの id は後述の RANGE0,...,RANGE255 により生成される．

7.1.1 書式

```
prepare(ジョブ定義ハッシュ [,'RANGE0' => リファレンス]
                        ...[,'RANGE255' => リファレンス]
                           [,'ジョブ定義ハッシュキー S' => リファレンス]
                        ...[,'ジョブ定義ハッシュキー S' => リファレンス]);
```

ただし，「ジョブ定義ハッシュキー S」はジョブ定義ハッシュキー（arg0 等）の語尾に S をつけ加えたもの（arg0S 等）を意味するものとする．

7.1.2 記述例

```
@jobs = prepare('id' => 'xyz', 'exe' => './kempo', 'arg0S' => [10,20]);
```

これは以下と同義である．

```
@jobs = ();
push(@jobs, ['id' => 'xyz_0', 'exe' => './kempo', 'arg0' => '10']);
push(@jobs, ['id' => 'xyz_1', 'exe' => './kempo', 'arg0' => '20']);
```

宣言的に書くこともできる.

```
%abc = (
    'id' => 'xyz',
    'exe' => './kempo',
    'arg0S' => [10,20]
);
prepare(%abc);
```

ジョブ定義ハッシュとパラメタを分けて書くこともできる.

```
%abc = (
    'id' => 'xyz',
    'exe' => './kempo'
);
prepare(%abc, 'arg0S' => [10,20]);
```

RANGE0 と関数リファレンスを使うことでさまざまなパラメタでジョブを生成することができる. 例えば,

```
@jobs = prepare(%abc, 'RANGE0' => [0..99], 'arg0S' => sub { 2 * $_[0] });
```

は prepare(%abc, 0), prepare(%abc, 2),..., prepare(%abc, 198) を順番に行ったものと同義である.

7.1.3 発展

パラメタは複数書くことができる. 複数パラメタの配列の頭からジョブは生成される. 例えば,

```
%abc = (
    'id' => 'xyz',
    'exe' => './kempo'
);
@jobs = prepare(%abc, 'arg0s' => [0,1], 'arg1S' => [2,3]);
```

は以下と同義である.

```
@jobs = ();
push(@jobs, ['id' => 'xyz_0', 'exe' => './kempo', 'arg0' => '0', 'arg1' => '2']);
push(@jobs, ['id' => 'xyz_1', 'exe' => './kempo', 'arg0' => '1', 'arg1' => '3']);
```

RANGE0 等と関数リファレンスを使うことでパラメタをかけ合わせてジョブを生成することができる. 例えば,

```
%abc = (
    'id' => 'xyz',
    'exe' => './kempo'
);
@jobs = prepare(%abc, 'RANGE0' => [0,1], 'RANGE1' => [2,4],
                'arg0S' => sub { $_[0] + $_[1] });
```

は以下と同義である.

```
@jobs = ();
push(@jobs, ['id' => 'xyz_2', 'exe' => './kempo', 'arg0' => '2']);
push(@jobs, ['id' => 'xyz_4', 'exe' => './kempo', 'arg0' => '4']);
push(@jobs, ['id' => 'xyz_3', 'exe' => './kempo', 'arg0' => '3']);
push(@jobs, ['id' => 'xyz_5', 'exe' => './kempo', 'arg0' => '5']);
```

7.2 submit

ジョブリファレンスの配列を受け取り，各ジョブをジョブスケジューラに渡し，ジョブスケジューラからジョブスレッドを受け取り，それらの配列を返す.

7.2.1 書式

submit(**ジョブリファレンスの配列**);

7.2.2 記述例

典型的には prepare の返り値を引数にとる.

```
@jobs = prepare('id' => 'xyz', 'exe' => './kempo', 'arg0S' => [10,20]);
submit(@jobs);
```

自力でジョブリファレンスの配列を書いてもよい.

```
submit({'id' => 'xyz_0', 'exe' => './kempo', 'arg0' => '10'},
       {'id' => 'xyz_1', 'exe' => './kempo', 'arg0' => '20'});
```

7.3 sync

ジョブスレッドの配列を受け取り，ジョブ処理後のジョブリファレンスの配列を返す.

7.3.1 書式

sync(**ジョブスレッドの配列**);

7.3.2 記述例

典型的には submit の返り値を引数にとる.

```
@jobs = prepare('id' => 'xyz', 'exe' => './kempo', 'arg0S' => [10,20]);
@thrds = submit(@jobs);
sync(@thrds);
```

7.4 prepare_submit_sync

prepare, submit, sync を順に行う. 書式は prepare に準ずる.

7.5 prepare_submit

prepare, submit を順に行う. 書式は prepare に準ずる.

7.6 submit_sync

submit, sync を順に行う. 書式は submit に準ずる.

7.7 pickup

ファイル名と区切り文字を受け取り, ファイルの先頭行を区切り文字で分割し, それらからなる配列を返す.

7.7.1 書式

```
pickup(ファイル名, 区切り文字);
```

7.8 repickup

ジョブハッシュリファレンスの配列を受け取り, ofile キーの値であるところのファイルの先頭行を odelimiter キーの値であるところの区切り文字で区切った際の ocolumn キーの値であるところの数番目の列からなる配列を返す.

7.8.1 書式

```
repickup(ジョブハッシュリファレンスの配列);
```