

Xcrypt マニュアル

京大中島研 e-science グループ

2009 年 10 月 27 日

目次

第 I 部	総論	3
第 1 章	はじめに	4
1.1	機能	4
1.2	動作環境	4
第 2 章	Xcrypt スクリプトの記述	5
2.1	モジュールに関する記述	5
2.2	大域変数に関する記述	5
2.3	ジョブ定義ハッシュに関する記述	6
2.4	ジョブ処理に関する記述	7
2.5	記述例	7
第 3 章	ジョブ処理の流れ	8
3.1	概念図	8
3.2	実行方法	8
3.3	作成物	9
第 II 部	各論	11
第 4 章	モジュール	12
4.1	core	12
4.2	dry	12
第 5 章	大域変数	13
5.1	smph	13
5.2	separator	13
5.3	separator_nocheck	13
5.4	allkeys	13
第 6 章	ジョブ定義ハッシュ	15
6.1	id	15
6.2	exe	15

6.3	arg0,...,arg255	15
6.4	linkedfile0,...,linkedfile255	15
6.5	copiedfile0,...,copiedfile255	15
6.6	copieddir0,...,copieddir255	16
6.7	stdofile	16
6.8	stdefile	16
6.9	queue	16
6.10	cpu	16
6.11	proc	16
6.12	option	16
第 7 章	組込み関数	17
7.1	prepare	17
7.2	submit	20
7.3	sync	21
7.4	addkeys	21
7.5	prepare_submit	22
7.6	submit_sync	22
7.7	prepare_submit_sync	22

メモ 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

はじめに

1.1 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．

1.2 動作環境

- ・ sh 系（sh, bash, zsh 等）または csh 系（csh, tcsh 等）シェル
- ・ Perl 5.10.0（GUI 利用時には Perl/Tk 8.4 も要）

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\$separator）は

```
$separator = '-';
```

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' => '100',  
    'arg1' => 'plasma.inp',  
    'linkedfile0' => 'kempo',  
    'copiedfile0' => 'plasma.inp',  
    'stdofile' => 'hogeout',  
    'stdefile' => 'hogeerr',  
    'queue' => 'gh10034',  
    'option' => '# @$-g gh10034'  
);
```

と記述する。デフォルトで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。ジョブ定義ハッシュキー（例えば userdefinedkey0 と userdefinedkey1）をユーザ定義するにはあらかじめ

```
addkeys('userdefinedkey0', 'userdefinedkey1');
```

と書いておく（7 章を参照のこと）か、または、ジョブ定義ハッシュのキーを与える際、

```
':userdefinedkey0' => 0
```

とキー名を:で始める。

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(core);

$smph=Thread::Semaphore->new(10);
$separator = '-';

%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'
);

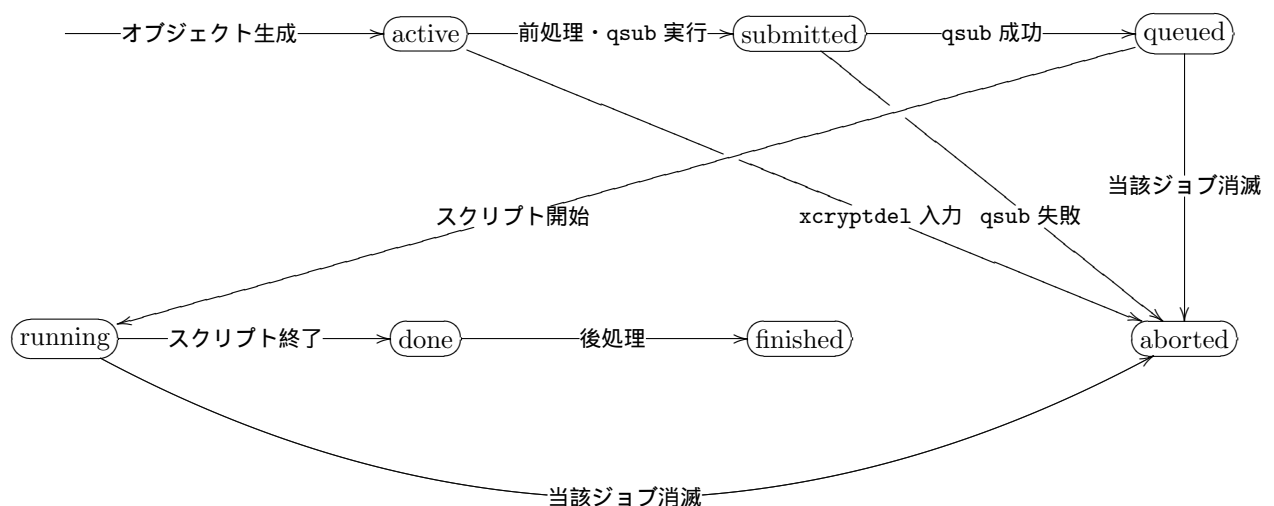
&prepare_submit_sync(%xyz, 'arg1@' => [2,4]);
```


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 概念図



3.2 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
```

環境変数 XCRJOBSCHED をジョブスケジューラの名前^{*1}に設定する。シェルが bash なら，例えば，

```
$ XCRJOBSCHED=SGE; export XCRJOBSCHED
```

とする。

^{*1} NQS と SGE とが利用可能である。また，環境変数 XCRJOBSCHED に sh も利用可能である。この場合，ジョブを OS のプロセスとして扱い，ジョブスケジューラが導入されていない環境におけるジョブの投入・削除・状態取得を行う。

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/bin/xcrypt sample.xcr
```

3.3 作成物

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。

(Xcrypt スクリプト名).pl

Xcrypt が Xcrypt スクリプトから作成する Perl スクリプトである。Xcrypt が行う Xcrypt スクリプトの実行は、Perl が行うこの Perl スクリプトの実行を含む。

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。

sh.sh

シェルをジョブスケジューラとして仮想的に利用す際、シェル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（コマンドの実行のみ行わないモード）で動作させることができる．

ジョブ定義ハッシュに `dry` というキーが使えるようになる．ジョブ定義ハッシュに

```
...  
'dry' => 1,  
...
```

と記述することでジョブがドライモードで動作する．

第 5 章

大域変数

5.1 smph

一度に投入されるジョブの数の上限を指定する .

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

5.2 separator

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 `$separator` で区切った名前のディレクトリを作成する . `$separator` のデフォルト値は `'_'` である . `'-'` に変えたい場合は ,

```
$separator = '-';
```

とする .


使用できるシンボルは `["$%&'/:;<=>?[\] '{|} 除く ASCII 印字可能文字とする .`

5.3 separator_nocheck

この値を 1 にすると `$separator` が使用できるシンボルであるかのチェックを飛ばす . デフォルト値は 0 である .

5.4 allkeys

ジョブ定義ハッシュにおいて利用可能なキーを格納している .



```
push(@allkeys, 'userdefinedkey');
```

とすることで userdefinedkey がキーとして利用可能になる .

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デフォルトで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。

使用できるシンボルは `␣"$%&'/:;<=>?[\\]`{|}` を除く ASCII 印字可能文字とする。

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 copieddir0,...,copieddir255

この値の名前であるディレクトリ中のファイルから作業ディレクトリにコピーをつくる。

6.7 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。

6.8 stderrfile

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。

6.9 queue

実行するジョブを投入するキューの名前を記述する。

6.10 cpu

使用するコア数を指定する。

6.11 proc

使用するプロセス数を指定する。

6.12 option

ジョブスケジューラのオプションを記述する。

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（リファレンス^{*1}またはスカラで与えられる）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' => (配列リファレンス)]
        ..., 'RANGE255' => (配列リファレンス)]
        [, 'ジョブ定義ハッシュキー@' => (リファレンス||スカラ)]
        ..., 'ジョブ定義ハッシュキー@' => (リファレンス||スカラ));
```

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

RANGE0, ..., RANGE255 の値であるところの配列リファレンスにおける配列に使用できるシンボルは「"\$%&'/:;<=>?[\]‘{|}」を除く ASCII 印字可能文字とする。

^{*1} 本稿ではリファレンスは型グロブを含まないものとする。

7.1.2 記述例

```
@jobs = prepare('id' => 'xyz', 'exe' => './kempo', 'arg0@' => [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',  
    'arg0@' => [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, 'arg0@' => [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], 'arg0@' => 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, 'arg0@' => [0,1], 'arg1@' => [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掛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],  
    'arg0@' => 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'});
```

引数がリファレンスでなくスカラである時、引数は eval される。ゆえに、上記は、

```
%abc = (
    'id' => 'xyz',
    'exe' => './kempo'
);
@jobs = prepare(%abc, 'RANGE0' => [0,1], 'RANGE1' => [2,4],
    'arg0@' => '$_[0] + $_[1]');
```

とも書くこともできる。

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、ジョブリファレンスの配列を返す。

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', 'arg0@' => [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

ジョブハッシュの配列を受け取り，そのジョブの後処理を行う．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

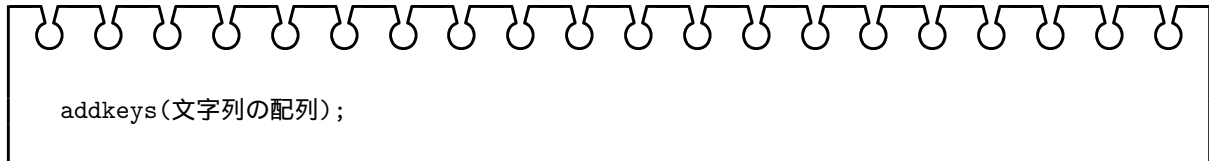
典型的には submit の返り値を引数にとる．

```
@jobs = prepare('id' => 'xyz', 'exe' => './kempo', 'arg0@' => [10,20]);  
@objs = submit(@jobs);  
sync(@objs);
```

7.4 addkeys

文字列の配列を受け取り，それらをジョブ定義ハッシュキーとして利用可能にする．

7.4.1 書式



```
addkeys(文字列の配列);
```

7.5 prepare_submit

prepare, submit を順に行う。ただし, prepare で生成したジョブリファレンスを即座に submit する (全てのジョブリファレンスが生成されるのを待たない)。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 prepare_submit_sync

prepare_submit, sync を順に行う。書式は prepare に準ずる。