

Xcrypt マニュアル

京大中島研 e-science グループ

2009 年上半期

目次

第 I 部	総論	3
第 1 章	機能	4
第 2 章	実装	5
第 3 章	Xcrypt スクリプトの記述	7
3.1	モジュールに関する記述	7
3.2	ジョブ定義ハッシュに関する記述	7
3.3	ジョブの処理に関する記述	9
3.4	記述例	9
第 4 章	ジョブ処理の実際	10
4.1	実行方法	10
4.2	ジョブ処理後に作成されるもの	11
第 II 部	各論	12
第 5 章	モジュール	13
5.1	limit	13
5.2	function	13
5.3	successor	13
5.4	predecessor	13
5.5	dry	13
5.6	jobsched	14
5.7	UI	14
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	stdofile	16
6.7	stdefile	16
6.8	queue	16
6.9	cpu	16
6.10	proc	16
6.11	option	16
第 7 章	組込み関数	17
7.1	prepare	17
7.2	submit	20
7.3	sync	20
7.4	prepare_submit_sync	21
7.5	prepare_submit	21
7.6	submit_sync	21

TODO 定義すべき用語ジョブ定義ハッシュ, ジョブキー, ジョブリファレンス, ジョブスレッド

メモ 内輪向け解説

重要

constructor.pm に exit_cond に関するバグあり, exit_cond に必要性があるか? 直すにしてもどのように直すべきか? のどちらも不明.

第Ⅰ部

総論

第 1 章

機能

NQS スクリプト (nqs.sh) を作成し、それを NQS に渡し (`$ qsub nqs.sh`), NQS が返す結果からまた別の NQS スクリプトを作成し、それを NQS に渡し....., という一連の操作を、perl スクリプト user.pl を作成し、それを xcrypt に渡す (`$ xcrypt sample.pl`) だけで済むようにするモジュール群である。

ライトユーザ向けに、ジョブレベルでなく、ジョブセットレベルの記述もサポートしている。

第 2 章

実装

NQS スクリプト一つの作成をスレッド一つで行う。

モジュールにはコアモジュールと汎用目的モジュールとの二種類ある。

コアモジュールは `new start before after` という関数をのみを持つことができる。これらの関数はこの順番で呼び出される仮定の下で書かれている。

「呼び出す関数 → 呼び出される関数」と書くことにすると、`1.pm` の `start →, ..., → n.pm` の `start`、`top.pm` の `start → 1.pm` の `before →, ..., → n.pm` の `before` であり、さらに、`top.pm` の `start → n.pm` の `after →, ..., → 1.pm` の `after` である。

現在のコアモジュール間の継承関係は、子から親へ向かって `constructor.pm` `successor.pm` `dry.pm` `predecessor.pm` `limit.pm` `top.pm` である。

汎用目的モジュールは現在 `UI.pm` `function.pm` `jobsched.pm` である。

`xcrypt`

ユーザはモジュール群で処理されるフィールドとその値の組となるハッシュを記述し、そのリファレンスを `id` と同名のスカラー変数で定義する。

`constructor.pm`

`trigger.pl` で定義されたハッシュリファレンスからジョブを生成する。

オブジェクトの `exit_cond` と `trace` から処理を続けるかどうかを判定する。続ける場合は `change_arg` で `arg` を更新した別オブジェクト（その `id` は元オブジェクトの `id` の後ろに 0 を付加したもの）を生成し、その処理を行う。

`successor.pm`

`successor` から別オブジェクトを生成する。ただし `successor` が空の場合は出力列を `traces.log` に書き出して終了する。

`** dry.pm`

`./trigger -dry` でユーザ定義のオブジェクトのフィールド値を適当に上書きする。

`predecessor.pm`

`pjo_inventory_watch,write.pl` にジョブの処理を終了を判定してもらい、その確認ができるまでジョブの生成を待つ。

`limit.pm`

作成されて処理されるのを待っている NQS スクリプトの数をスレッドにまたがる変数の値で制御する。

`top.pm`

関数 `before` を呼び出すことで前処理をし、NQS スクリプトを作成し、それを `qsub` に渡し（ジョブ管理をするモジュール `jobsched.pm` 中の関数を呼び出すことで実現している）、関数 `after` を呼び出すことで後処理をする。

`UI.pm`

ユーザがオペレーションとして利用する関数を定義している。

`function.pm`

ユーザがデータとして利用する関数を定義している。

`jobsched.pm`

`pjo_inventory_watch,write.pl` と連携してジョブ管理を行う。

第 3 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ように記述される．

1. モジュールに関する記述
2. ジョブ定義ハッシュに関する記述
3. ジョブの処理に関する記述

3.1 モジュールに関する記述

利用可能なモジュールにはコアモジュールとそれ以外のモジュールの二種類がある．コアモジュールは自動的に読み込まれるため，明示的に読み込むための記述をする必要はない．コアモジュール以外に関しては，例えば

```
use モジュール名;
```

と記述する．利用可能なモジュールに関しては 5 で述べる．

3.2 ジョブ定義ハッシュに関する記述

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．定義可能なキーに関しては 6 で述べる．

3.3 ジョブの処理に関する記述

3.4 記述例

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

%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 4 章

ジョブ処理の実際

4.1 実行方法

NQS スクリプト (nqs.sh) を作成し, それを NQS に渡し

```
$ qsub nqs.sh
```

NQS が返す結果からまた別の NQS スクリプトを作成し, それを NQS に渡し....., という一連の操作を,
実行方法

環境変数 XCRYPT を \$HOME/e-science/xcrypt とかにする .
移動する .

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

xcrypt スクリプト (例えば hoge.xcr) を作成する .
実行する .

```
$XCRYPT/xcrypt hoge.xcr
```

perl スクリプト user.pl を作成し, それを xcrypt に渡す

```
$ xcrypt sample.pl
```

4.2 ジョブ処理後に作成されるもの

4.2.1 ディレクトリ

4.2.2 `nqs.sh`

4.2.3 `request_id`

4.2.4 `stdout`

4.2.5 `stderr`

第 II 部

各論

第 5 章

モジュール

コア扱い？モジュール扱い？

5.1 limit

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

5.2 function

5.3 successor

5.4 predecessor

5.5 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．また，

```
$ xcrypt -d sample.xcr
```

とオプションつきで実行することでドライモードで動作させることもできる．

1 にセットすることで Xcrypt をドライモード（実行コマンドなしで）で動作させる．

5.6 jobsched

ジョブスケジューラとのインターフェイスである .

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

```
$jobsched::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 .

```
$ xcrypt -s sample.xcr
```

とオプションつきで実行することと同義である .

5.7 UI

組込み関数を提供する .

Xcrypt は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

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 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．

6.8 queue

実行するジョブを投入するキューの名前を記述する．

6.9 cpu

6.10 proc

6.11 option

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、ジョブリファレンスの配列を返す。

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],
    '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

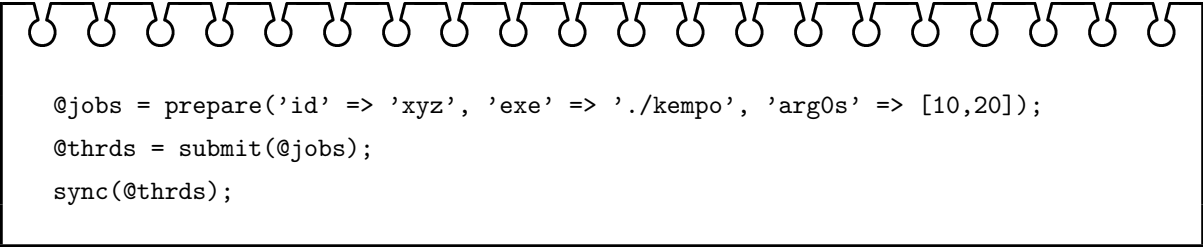
ジョブスレッドの配列を受け取り，ジョブ処理後のジョブリファレンスの配列を返す．

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', 'args' => [10,20]);  
@thrs = submit(@jobs);  
sync(@thrs);
```

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．