Xcrypt Manual

E-science Group, Nakashima Laboratory, Kyoto University

January 7, 2010

Contents

I General **3**

1 Introduction **4**
1.1 Overview 4
1.2 Environment 4

2 Description **5**
2.1 Module 5
2.2 Job Definition Hash 6
2.3 Job Processing 6
2.4 Example 6

3 Flow **8**
3.1 Model 8
3.2 Execution 8
3.3 Product 9

II Details **11**

4 Module **12**
4.1 `core` 12
4.2 `limit` 12
4.3 `successor` 13
4.4 `simple_convergence` 13
4.5 `dry` 13

5 Job Definition Hash **14**
5.1 `id` 14
5.2 `exe` 14
5.3 `arg`i 14
5.4 `linkedfile`i 14
5.5 `copiedfile`i 15
5.6 `copieddir`i 15
5.7 `stdofile` 15
5.8 `stdefile` 15
5.9 `queue` 15
5.10 `cpu` 15
5.11 `proc` 15

5.12 option 15

6 組込み関数 **16**
6.1 prepare 16
6.1.1 書式 16
6.1.2 記述例 17
6.1.3 発展 18
6.2 submit 19
6.2.1 書式 19
6.2.2 記述例 19
6.3 sync 19
6.3.1 書式 19
6.3.2 記述例 19
6.4 addkeys 20
6.4.1 書式 20
6.5 prepare_submit 20
6.6 submit_sync 20
6.7 prepare_submit_sync 20

A ジョブクラス拡張モジュール実装の手引 **21**
A.1 はじめに 21
A.2 拡張モジュールの定義と利用 21
A.3 拡張モジュールスクリプトの構成 21
A.4 特別な意味を持つメソッド 23
A.4.1 new メソッド 23
A.4.2 before_isready メソッド 23
A.4.3 before メソッド 24
A.4.4 start メソッド 24
A.4.5 after_isready メソッド 24
A.4.6 after メソッド 24
A.4.7 before_isready, before, start, after_isready, after の並行性 25

Memo 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.

？氏の Coro.pm を使用している．Perl ライセンスなので問題ないはず.

Part I

General

Chapter 1

Introduction

1.1 Overview

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 Environment

・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）

Chapter 2

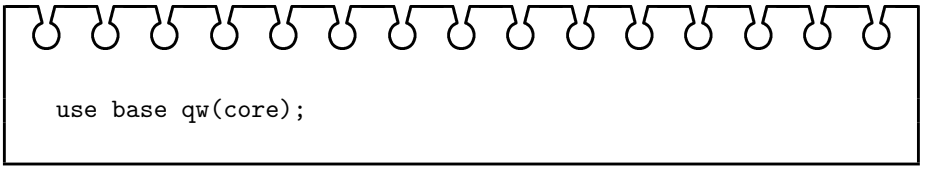
Description

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. Module
2. Job Definition Hash
3. Job Processing

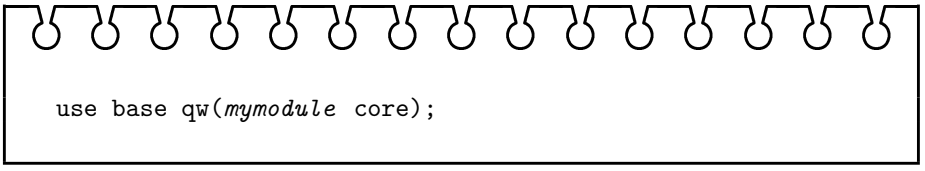
2.1 Module

モジュールは



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

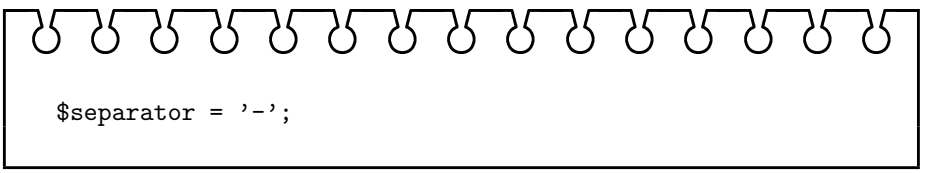
と記述して読み込む．複数読み込む場合は読み込みたい順に



```
use base qw(mymodule 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．

モジュールを読み込むことにより，利用可能になる変数（例えば`$separator`）は



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

と記述してセットする．

2.2 Job Definition Hash

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，

```
%myjob = (
    'id' => 'myjob',
    'exe' => './myexe',
    'arg0' => '100',
    'arg1' => 'myinput',
    'linkedfile0' => 'myexe',
    'copiedfile0' => 'myinput',
    'stdofile' => 'myout',
    'stdefile' => 'myerr',
    'queue' => 'myqueue',
    'option' => 'myoption'
);
```

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 5 章で述べる．

2.3 Job Processing

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 6 章で述べる．

2.4 Example

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．

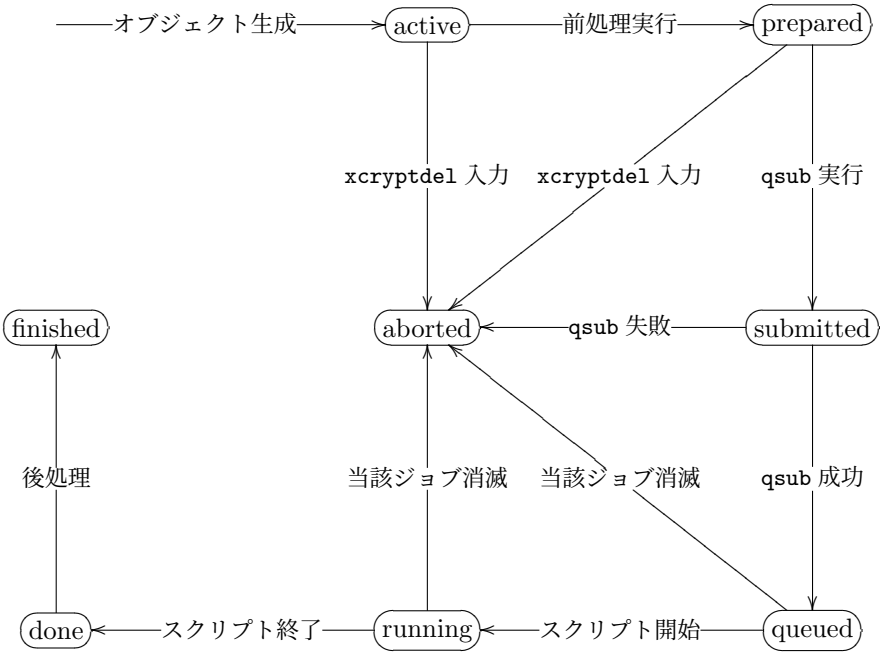
```
use base qw(limit core);

&limit::initialize(10);
$separator = '-';

%myjob = (
    'id' => 'myjob',
    'exe' => './myexe',
    'arg0' => '100',
    'arg1' => 'myinput',
    'linkedfile0' => 'myexe',
    'copiedfile0' => 'myinput',
    'stdofile' => 'myout',
    'stdefile' => 'myerr',
    'queue' => 'myqueue',
    'option' => 'myoption'
);
&prepare_submit_sync(%myjob, 'arg1@' => [2,4]);
```

Chapter 3

Flow

実際のジョブ処理の流れについて概観する.

3.1 Model



3.2 Execution

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．

環境変数 PERL5LIB に$XCRYPT/lib と$XCRYPT/lib/algo/lib とを追加する．

作業ディレクトリに移動する（ここでは $HOME/wd とする）．

```
$ cd $HOME/wd
```

Xcrypt スクリプト（2.4 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 Product

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 linkedfile$i$ 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，copiedfile$i$ 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．

[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．

`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．

Part II

Details

Chapter 4

Module

In this chapter, we introduce some modules available in Xcrypt scripts.

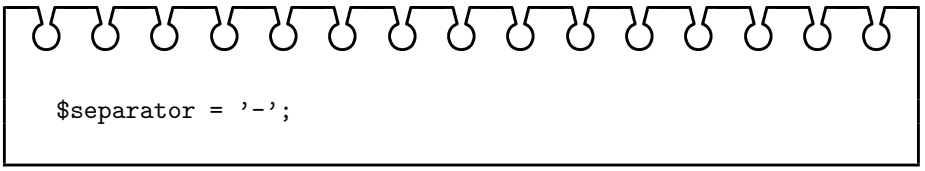
4.1 core

This module is the Xcrypt core module, and required to be read in order to use anything paricular to Xcrypt (e.g., a job definiton hash).

It creates a directory of the name

```
$myjob->{'id'} $separator $myjob->{'arg0'} $separator ...
```

under the job working directory, where `%myjob` is a job definition hash. The default of `$separator` is _. In order to redefine `$separator` to be `-`, for example, it is enough to describe as follows,



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

Any word of ASCII printable characters except

```
@␣"$%&'/:;<=>?[\]‘{|}
```

is available.

This module also makes the variable `$separator_nocheck` to be available in Xcrypt scripts. When the value of `$separator_nocheck` is 1, Xcrypt skips to check `$separator` for availability. The default is 0.

4.2 limit

This module limits the number of jobs to be submitted. In order to limit the number of jobs to 10, for example, it is enough to describe as follows,

```
&limit::initialize(10);
```

4.3 successor

ジョブを宣言的に定義できるようにする．ジョブ定義ハッシュに successors というキーが使えるようになるので，例えば，%x，%y というジョブを定義したければ，

```
...
'successor' => ['x', 'y'],
...
```

と記述して使う．

4.4 simple_convergence

投入したジョブの今回値と前回値の差が一定値以下になるまで繰り返す Plan-Do-Check-Action (PDCA) サイクルを容易に実現する関数を提供する．ジョブ定義ハッシュに initialvalue（初期値），initialvalue（初期値），isConvergent（収束条件），inputfile（namelist 形式の入力ファイル名），sweepname（入力ファイルから取得する値の名前），outputfile（出力ファイル名），extractrules（出力ファイルから値を取得するための Data_Extraction における規則群）というキーが使えるようになる．

4.5 dry

This module provides job processing in dry mode (skipping any command exectution).

Description in a job definition hash

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

makes any job (derived from this hash) to process in dry mode.

Chapter 5

Job Definition Hash

In this chapter, we introduce some keys available in job definition hashes by default.

5.1 id

実行されるジョブを識別する語を記述する．
Any word of ASCII printable characters except

@␣"$%&'/:;<=>?[\]`{|}

is available.

5.2 exe

実行されるジョブの実行コマンドを記述する．後述の arg$i$ とともに

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

といった形で実行される．

5.3 arg$i$

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．

5.4 linkedfile$i$

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．

5.5 copiedfile*i*

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.

5.6 copieddir*i*

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.

5.7 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.

5.8 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.

5.9 queue

実行するジョブを投入するキューの名前を記述する.

5.10 cpu

使用するコア数を指定する.

5.11 proc

使用するプロセス数を指定する.

5.12 option

ジョブスケジューラのオプションを記述する.

Chapter 6

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．

6.1 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 RANGEi により生成される．

6.1.1 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．

RANGEi 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．

6.1.2 記述例

```
@jobs = prepare('id' => 'myjob', 'exe' => './myexe', 'arg0@' => [10,20]);
```

これは以下と同義である.

```
@jobs = ();
push(@jobs, {'id' => 'myjob_0', 'exe' => './myexe', 'arg0' => '10'});
push(@jobs, {'id' => 'myjob_1', 'exe' => './myexe', 'arg0' => '20'});
```

宣言的に書くこともできる.

```
%abc = (
    'id' => 'myjob',
    'exe' => './myexe',
    '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' => 'myjob',
    'exe' => './myexe'
);
prepare(%abc, 'arg0@' => [10,20]);
```

RANGE0 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@' => '2 * $R0');
```

はprepare(%abc, 'arg0' => 0), prepare(%abc, 'arg0' => 2),..., prepare(%abc,

'arg0' => 198) を順番に行ったものと同義である.

6.1.3 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' => 'myjob',
    'exe' => './myexe'
);
@jobs = prepare(%abc, 'arg0@' => [0,1], 'arg1@' => [2,3]);
```

は以下と同義である.

```
@jobs = ();
push(@jobs, {'id' => 'myjob_0', 'exe' => './myexe', 'arg0' => '0', 'arg1' => '2'});
push(@jobs, {'id' => 'myjob_1', 'exe' => './myexe', 'arg0' => '1', 'arg1' => '3'});
```

RANGE0 等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' => 'myjob',
    'exe' => './myexe'
);
@jobs = prepare(%abc, 'RANGE0' => [0,1], 'RANGE1' => [2,4], 'arg0@' => '$R0 + $R1');
```

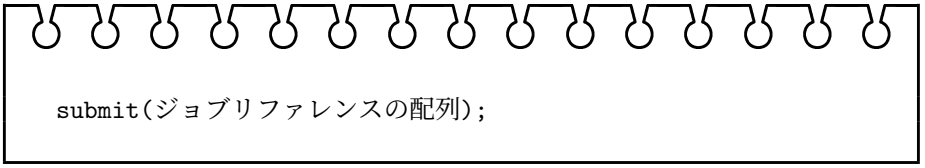
は以下と同義である.

```
@jobs = ();
push(@jobs, {'id' => 'myjob_0_2', 'exe' => './myexe', 'arg0' => '2'});
push(@jobs, {'id' => 'myjob_0_4', 'exe' => './myexe', 'arg0' => '4'});
push(@jobs, {'id' => 'myjob_1_2', 'exe' => './myexe', 'arg0' => '3'});
push(@jobs, {'id' => 'myjob_1_4', 'exe' => './myexe', 'arg0' => '5'});
```

6.2 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.

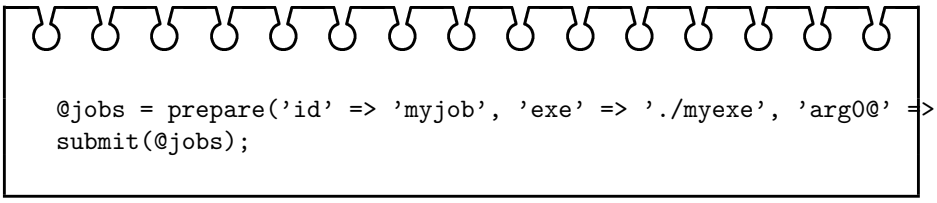
6.2.1 書式



```
submit(ジョブリファレンスの配列);
```

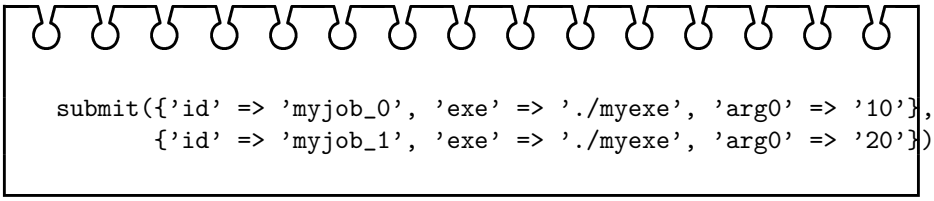
6.2.2 記述例

典型的には prepare の返り値を引数にとる.



```
@jobs = prepare('id' => 'myjob', 'exe' => './myexe', '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.

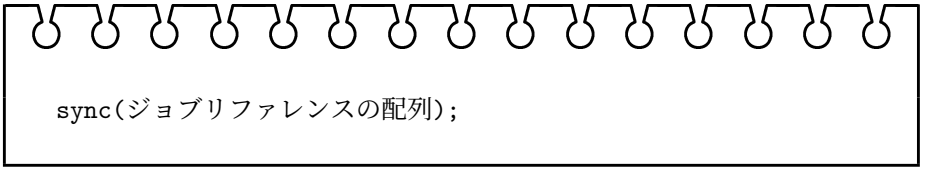


```
submit({'id' => 'myjob_0', 'exe' => './myexe', 'arg0' => '10'},
       {'id' => 'myjob_1', 'exe' => './myexe', 'arg0' => '20'});
```

6.3 sync

ジョブリファレンスの配列を受け取り，それらの同期をとり，ジョブリファレン
スの配列を返す.

6.3.1 書式



```
sync(ジョブリファレンスの配列);
```

6.3.2 記述例

典型的には submit の返り値を引数にとる.

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

6.4 addkeys

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

6.4.1 書式

```
addkeys(文字列の配列);
```

6.5 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．

6.6 submit_sync

submit，sync を順に行う．書式は submit に準ずる．

6.7 prepare_submit_sync

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

Appendix A

ジョブクラス拡張モジュール実装の手引

A.1 はじめに

Xcrypt の prepare 関数によって生成された全てのジョブオブジェクトは$XCRYPT/lib/core.pm で定義された core クラスに属する．Xcrypt のユーザや開発者は，拡張モジュールを定義して core クラスを拡張することで Xcrypt の機能を拡張することが可能である．本章では，この拡張モジュールを開発するために必要な情報を示す．

A.2 拡張モジュールの定義と利用

mymodule という名前の拡張モジュールを定義するためには，*mymodule*.pm という名前のファイルを$XCRYPT/lib/（または$PERL5LIB で指定されているどこかのディレクトリ）に置く．Xcrypt ユーザは，スクリプトの先頭で，

```
use base (... mymodule ... core);
```

のようにモジュール名を指定することで，当該モジュールの機能を使用できる．

A.3 拡張モジュールスクリプトの構成

典型的な拡張モジュールの定義スクリプトは以下のように記述される．

```
package  mymodule;


use strict;
use ...;

&addkeys('my_instance_member', ...);

my $my_class_member;

# special methods
sub new {
  my $class = shift;
  my $self = $class->NEXT::new(@_);
  ...
  return bless $self, $class;
}

sub before_isready { ... }
sub before { ... }

sub start
{
  my $self = shift;
  ...
  $self->NEXT::start();
  ...
}

sub after { ... }
sub after_isready { ... }

# general methods
sub another_method
{
  ...
}
```

以下，スクリプトの各構成要素について説明する．

1. モジュール名の定義：package で指定する．ここで指定する名前は，モジュールファイル名の拡張子（.pm）を除いた部分と同一でなければならない．
2. Perl モジュールの取り込み：通常の Perl プログラムと同様に，必要な Perl モジュールを use で取り込む．
3. 追加するインスタンス変数の定義：ジョブクラスに新たに追加したいインスタンス変数の名前を addkeys 関数により定義する．ここで定義したイン

スタンス変数は，ジョブオブジェクトの属性として，Xcrypt スクリプトやモジュール内のメソッドから`$job->{my_instance_menber}` などとしてアクセスできる．また，Xcrypt ユーザがジョブ定義ハッシュの記述時に，
`%template = { ..., my_instance_menber=>`*value*`, ...}`
のようにして値を設定することもできる．

4. クラス変数の定義：通常の Perl のオブジェクト指向プログラミングと同様，クラス変数はパッケージ内のグローバル変数として定義する．ここで定義した変数は，`$`*mymodule*`::my_class_member` としてアクセスできる

5. メソッドの定義：このモジュールにおいて追加，拡張するメソッドをパッケージ内のトップレベル関数として定義する．通常の Perl オブジェクト指向プログラミングと同様であるが，特定の名前を持つメソッドは特別な意味を持つので注意する．(次節で説明)

A.4 特別な意味を持つメソッド

A.4.1 `new` メソッド

コンストラクタに相当するクラスメソッドであり，Xcrypt の `prepare` 関数（6.1 節）の処理中に，最も specialized なクラス（Xcrypt スクリプトのヘッダで宣言されているモジュール列のうち最も左に書かれているもの）の `new` メソッドが呼び出される．

この new メソッドに渡される引数は以下の通りである．

1. Xcrypt スクリプトが属するパッケージ名（= `user`）

2. ジョブオブジェクトへの参照．オブジェクトのメンバには，`prepare` 関数に渡されたジョブ定義ハッシュに対応する値がセットされている．

ジョブ定義ハッシュの `RANGE` 等の指定により `prepare` 関数が複数のジョブオブジェクトを生成した場合は，その各オブジェクトに対して `new` が適用される．

メソッドの本体では，`$class->NEXT::new($self,$obj)`（`$class`，`$obj` はそれぞれ引数として渡されたクラス名およびオブジェクトへの参照）のようにして親クラスの `new` メソッドを呼び出すことができる．典型的には，各 `new` メソッドはまず親クラスの `new` メソッドを，与えられた 2 つの引数をそのまま引き渡して呼び出し，その返り値のオブジェクトにアクセスしつつ必要な処理を行った後，`bless` オブジェクトへの参照，渡されたクラス名の値をメソッドの返り値として `return` すべきである．

`core` モジュールでも `new` メソッドが定義されており，ここでは 3.3 節で説明したジョブの作業ディレクトリおよびその中へのファイルのコピー・シンボリックリンクの生成処理を行う．子クラスの `new` メソッドにより `core` モジュールの `new` メソッドが呼び出されないと，この処理が行われなくなるので注意すること．

A.4.2 `before_isready` メソッド

状態（3.1 節）が `prepared` になったジョブオブジェクトに対して適用される．一般には Xcrypt スクリプト中の `submit` 関数（6.2 節）の適用により，ジョブの状態が `prepared` になる．引数はジョブオブジェクトへの参照である．複数のモジュールで `before_isready` メソッドが定義されていた場合は，**子クラス→親クラスの順で全ての** `before_isready` メソッドが呼び出される．

各メソッドは，真偽値を返す．呼び出された before_isready メソッド列のうち，1 つでも偽を返したメソッドがあれば，しばらくの時間待ち合わせた後，もう一度 before_isready メソッド列の実行が行われる．

A.4.3 before メソッド

before_isready メソッド列が適用され，その全てが真を返したジョブオブジェクトに対して適用される．引数はジョブオブジェクトへの参照である．複数のモジュールで before メソッドが定義されていた場合は，**子クラス→親クラスの順で全ての** before メソッドが呼び出される．

メソッドの返り値は破棄される．

A.4.4 start メソッド

before メソッドの処理が終わったジョブオブジェクトに対して適用される．引数はジョブオブジェクトへの参照である．複数のモジュールで start メソッドが定義されていた場合は，最も specialized なクラスの start メソッドが呼び出される．

メソッドの本体では，$obj->NEXT::start()（$obj は引数として渡されたジョブオブジェクトへの参照）のようにして親クラスの start メソッドを呼び出すことができる．

core モジュールでも start メソッドが定義されており，ここではジョブスクリプトの生成およびバッチスケジューラへのジョブ投入処理を行う．子クラスの start メソッドにより core モジュールの start メソッドが呼び出されないと，この処理が行われなくなるので注意すること．

A.4.5 after_isready メソッド

状態（3.1 節）が done になったジョブオブジェクトに対して適用される．一般には，core::start メソッドによってバッチスケジューラに投入したジョブからのジョブ完了通知により，ジョブの状態が done になる．引数はジョブオブジェクトへの参照である．複数のモジュールで after_isready メソッドが定義されていた場合は，**親クラス→子クラスの順で全ての** after_isready メソッドが呼び出される．

各メソッドは，真偽値を返す．呼び出された after_isready メソッド列のうち，1 つでも偽を返したメソッドがあれば，しばらくの時間待ち合わせた後，もう一度 after_isready メソッド列の実行が行われる．

A.4.6 after メソッド

after_isready メソッド列が適用され，その全てが真を返したジョブオブジェクトに対して適用される．引数はジョブオブジェクトへの参照である．複数のモジュールで after メソッドが定義されていた場合は，**親クラス→子クラスの順で全ての** after メソッドが呼び出される．

メソッドの返り値は破棄される．

A.4.7 before_isready，before，start，after_isready，after の並行性

before, before_isready, start, after, after_isready の各メソッドは Xcrypt のユーザスレッドとは並行に実行される．ただし，各ジョブオブジェクト間についてのこれらのメソッドの実行は完全に並行に実行されるわけではない．Xcrypt 処理系が保証する並行性は以下の通りである．

- どのジョブオブジェクトの before_isready, before, start, after_isready, after メソッドも Xcrypt のユーザスレッドとは並行に実行される．
- どのジョブオブジェクトの before_isready，before，start メソッドも 2 つ以上並行に実行されることはない．
- どのジョブオブジェクトの after_isready，after，メソッドも 2 つ以上並行に実行されることはない．
- あるオブジェクトに対して before_isready メソッド列が全て真を返した際，それに続くそのオブジェクトへの before メソッド列の実行の前に，他のどのオブジェクトへの before_isready，before，start メソッド適用も行われることはない．
- あるオブジェクトへの before メソッド列の適用と start メソッドの適用の間に，他のどのオブジェクトへの before_isready，before，start メソッド適用も行われることはない．
- あるオブジェクトに対して after_isready メソッド列が全て真を返した際，それに続くそのオブジェクトへの after メソッド列の実行の前に，他のどのオブジェクトへの after_isready，after メソッド適用も行われることはない．