

Xcrypt Manual

E-Science Group, Nakashima Laboratory, Kyoto University

January 18, 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	n_section_method	13
4.5	dry	13
5	Job Definition Hash	15
5.1	id	15
5.2	exe	15
5.3	arg <i>i</i>	15
5.4	linkedfile <i>i</i>	15
5.5	copiedfile <i>i</i>	15
5.6	copieddir <i>i</i>	16
5.7	stdofile	16
5.8	stdefile	16
5.9	queue	16
5.10	cpu	16
5.11	proc	16

5.12	option	16
6	Built-in Function	17
6.1	prepare	17
6.2	submit	20
6.3	sync	21
6.4	addkeys	21
6.5	addperiodic	21
6.6	prepare_submit	22
6.7	submit_sync	22
6.8	prepare_submit_sync	22
A	ジョブクラス拡張モジュール実装の手引	23
A.1	はじめに	23
A.2	拡張モジュールの定義と利用	23
A.3	拡張モジュールスクリプトの構成	23
A.4	特別な意味を持つメソッド	25
A.4.1	new メソッド	25
A.4.2	before_isready メソッド	25
A.4.3	before メソッド	26
A.4.4	start メソッド	26
A.4.5	after_isready メソッド	26
A.4.6	after メソッド	26
A.4.7	before_isready , before , start , after_isready , after の並行性 . . .	26

Memo Xcrypt requires

- Marc Lehmann’s AnyEvent, common::sense, Coro, EV, and Guard,
- Joshua Nathaniel Pritikin’s Event,
- Daniel Muey’s Recursive.

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

Xcrypt requires a superset of Bourne or C shell, Perl 5.10.0 or any later version, and Perl/Tk 8.4 for GUI.

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(my module 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 = '-';

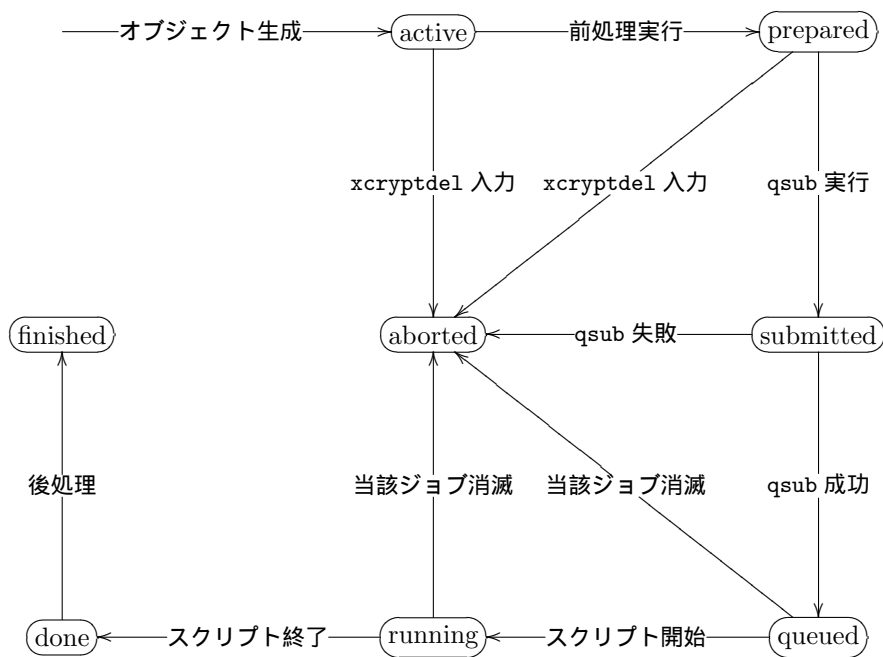
%myjob = (
    'id' => 'myjob',
    'exe' => './myexe',
    'arg0' => '100',
    'arg1' => 'myinput',
    'linkedfile0' => 'myexe',
    'copiedfile0' => 'myinput',
    'stdofile' => 'myout',
    'stdefile' => 'myerr',
    'queue' => 'myqueue',
    'option' => 'myoption'
);
&prepare_submit_sync(%myjob,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

¹NQS 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 `stderrfile` 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.

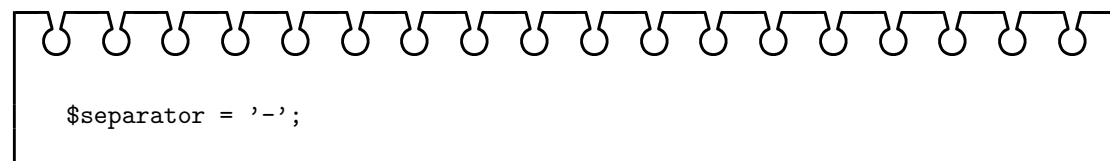
4.1 core

This module is the Xcrypt core module, and required to be read in order to use anything particular to Xcrypt (e.g., a job definition 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_noccheck to be available in Xcrypt scripts. When the value of \$separator_noccheck 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

This module provides how to describe dependency of jobs declaratively. For example, in order to define jobs of the name `%x`, `%y`, describe:

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

using the key `successors` in the job definition hash.

4.4 n_section_method

This module provides *n*-section method, a root-finding algorithm of the only difference from bisection method¹ is the number of sections.

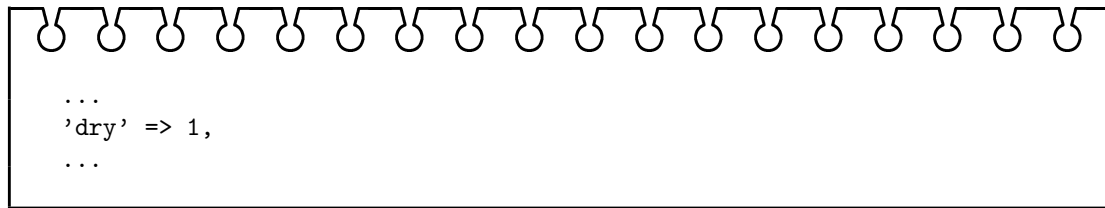
The values `partition` and `epsilon`, denote a partition number and an error, respectively. An interval is expressed by `x_left` and `x_right`. The values `y_left` and `y_right` are values on `x_left` and `x_right`. Typically, we can call the function `n_section_method` with these keys, e.g.,

```
&n_section_method::n_section_method(%job,  
  'partition' => 12, 'epsilon' => 0.01,  
  'x_left'    => -1,  'x_right' => 10,  
  'y_left'    => 0.5,  'y_right' => -5  
);
```

4.5 dry

This module provides job processing in dry mode (skipping any command execution). Description in a job definition hash

¹http://en.wikipedia.org/wiki/Bisection_method/



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

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

Chapter 5

Job Definition Hash

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

5.1 id

Its value is a word for identifying a job. Any word of ASCII printable characters except

@_ "\$%&'/:;<=>?[\] '{|}

is available.

5.2 exe

Its value is a command for a job. The command is executed as follows,

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

with arg_i as explained below.

5.3 arg_i

Its values are arguments of a command. The command is executed as follows,

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

5.4 linkedfile*i*

A soft link of the file of the name its value is created in the job working directory.

5.5 copiedfile*i*

A file of the name its value is copied in the job working directory.

5.6 `copieddir`

All files in the directory of the name its value are copied in the job working directory.

5.7 `stdofile`

The standard output is stored in a file of the name its value. The default is `stdout`.

5.8 `stdefile`

The standard error is stored in a file of the name its value. The default is `stderr`.

5.9 `queue`

Its value is a name of a queue.

5.10 `cpu`

Its value is the number of cores used exclusively.

5.11 `proc`

Its value is the number of processes used exclusively.

5.12 `option`

Its value is an option of a job scheduler.

Chapter 6

Built-in Function

In this chapter, we introduce built-in functions.

6.1 prepare

This function takes a job definition hash and parameters of references¹, and returns an array of job references. The `ids` of job objects are generated by `RANGEi` as described later.

Format

```
prepare(%myjob
[, 'RANGE0' => \@myparam0s]...[, 'RANGE255' => \@myparam255s]
[, 'key@' => \@myparams]...);
```

where *key*@ denotes the one whose postfix is the character @ (e.g., `arg0@`).

Any word of ASCII printable characters except

`@_"$%&'/:;<=>?[\] '{|}`

is available for `RANGEi`'s values.

Example

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

This is almost the same as

¹In this manual, references do not denote type globs.

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

Declarative description is as follows,

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

Description for a job definition hash can be separated from parameters:

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

Various job objects can be generated by using RANGE0. For example,

```
@jobs = prepare(%template, 'RANGE0' => [0..99], 'arg0@' => '2 * $R0');
```

is the same as a sequence of `prepare(%template, 'arg0' => 0)`, `prepare(%template, 'arg0' => 2)`, ..., `prepare(%template, 'arg0' => 198)`.

Advanced

Multiple parameters can be available. For example,

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

is almost the same as

```
%template = (
    'id' => 'myjob',
    'exe' => './myexe'
);
@jobs = ();
@job0 = prepare(%template, 'arg0' => '0', 'arg1' => '2');
push(@jobs, $job0[0]);
@job1 = prepare(%template, 'arg0' => '1', 'arg1' => '3');
push(@jobs, $job1[0]);
```

To use `RANGE`is makes it to generate job products by multiple parameters, e.g.,

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

is almost the same as

```

%template = (
    'id' => 'myjob',
    'exe' => './myexe'
);
@jobs = ();
@job0 = prepare(%template, 'arg0' => '0', 'arg1' => '2');
push(@jobs, $job0[0]);
@job1 = prepare(%template, 'arg0' => '0', 'arg1' => '4');
push(@jobs, $job1[0]);
@job2 = prepare(%template, 'arg0' => '1', 'arg1' => '2');
push(@jobs, $job2[0]);
@job3 = prepare(%template, 'arg0' => '1', 'arg1' => '4');
push(@jobs, $job3[0]);

```

6.2 submit

This function takes an array of job references, passes jobs to a job scheduler, and returns the array of job references.

Format

```

submit(@myjobs);

```

Example

Typically, this function takes a return value of `prepare`.

```

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

```

It is possible to describe job references without using `prepare` (though not to be recommended).

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

6.3 sync

This function takes an array of job references, syncs the jobs, and returns the array of job references.

Format

```
sync(@myjobs);
```

Example

Typically, this function takes `prepare` (the same as `submit`).

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

6.4 addkeys

This function takes an array of words and makes it available as keys in job definition hashes.

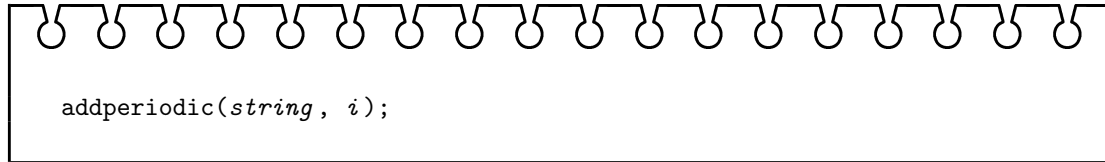
Format

```
addkeys(@words);
```

6.5 addperiodic

This function takes a Xcrypt's code and an integer i , and evaluates it each the i seconds.

Format



```
addperiodic(string, i);
```

6.6 prepare_submit

This function is an abbreviation of `prepare` and `submit`. Its format follows `prepare`.

6.7 submit_sync

This function is an abbreviation of `submit` and `sync`. Its format follows `submit`.

6.8 prepare_submit_sync

This function is an abbreviation of `prepare_submit` and `sync`. Its format follows `prepare`.

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_member}` 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_member=>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、その戻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戻り値として `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。

メソッドの戻り値は破棄される。

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した後、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。

メソッドの戻り値は破棄される。

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。