Xcrypt 言語仕様

E-Science Group, Nakashima Laboratory, ACCMS, Kyoto University

平成 23 年 12 月 21 日

目 次

1 Overview

In using a high-performance computer, we usually commit job processing to a job scheduler. At this time, we often go through the following procedures:

- to create a script in its writing style depending on the job scheduler,
- to pass the script to the job scheduler, and
- to extract data from its result, create another script from the data, and pass it to the job scheduler.

However, such procedures require manual intervention cost. It therefore seems better to remove manual intervention in mid-processing by using an appropriate script language. Xcrypt is a script language for job parallelization. We can deal with jobs as objects (called *job objects*) in Xcrypt and manipulate the jobs as well as objects in an object-oriented language. Xcrypt provides some functions and modules for facilitating job generation, submission, synchronization, etc. Xcrypt makes it easy to write scripts to process job, and supports users to process jobs easily.

2 用語

バッチスケジューラ スーパーコンピュータ等大規模計算システムにおける計算資源の利用状況を管理し，ユーザからシステム上で行いたい処理（ジョブ）の要求があれば，資源の空き状況などに基づいて適切にジョブに計算資源を割当て，実行させるソフトウェア．NQS，SGE，Torque など．ユーザは実行したい処理や実行に必要とする計算資源量などを記述したスクリプト（ジョブスクリプト）を入力としてジョブを投入すると，バッチスケジューラはそれを自身が管理する「ジョブキュー」にいったん追加し，ジョブの実行が可能と判断されたタイミングでキューから取り除き，ジョブへの計算資源の割当て・実行指示を行う．

request ID バッチスケジューラが，投入されたジョブを識別するためのユニークな ID．

ジョブオブジェクト Xcrypt 上において，これから投入しようとする，あるいは投入したジョブを表現するオブジェクト．下記のジョブ ID の文字列などをメンバに持つ．

ジョブ ID Xcrypt 上でジョブオブジェクトを識別するための文字列．基本的に request ID と 1 対 1 対応である．[1]

[1] 一度投入したジョブが失敗・ユーザによるキャンセルなどにより再投入された場合には複数の request ID が 1 つのジョブ ID に対応することになるが，その場合でも最新の request ID とジョブ ID との対応は一対一になる．

3 動作環境

- Bourne shell 互換のシェルスクリプト
- Perl Version 5.8.5 以上．なお，以下の CPAN パッケージを利用する（Xcrypt の配布パッケージに含まれる）．
  - Sherzod Ruzmetov's Config-Simple,
  - Marc Lehmann's Coro (where conftest.c is not contained), EV,
  - Gurusamy Sarathy's Data-Dumper,
  - Graham Barr's Error,
  - Joshua Nathaniel Pritikin's Event,
  - Salvador Fandiño's Net-OpenSSH,
  - Daniel Muey's Recursive,
  - H.Merijn Brand and Jochen Wiedmann's Text-CSV_XS, and
  - Marc Lehmann's AnyEvent, common::sense, and Guard.
- バッチスケジューラがインストールされた環境では，そのバッチスケジューラ用の設定ファイルを記述することにより，Xcrypt からそのバッチスケジューラのジョブを投入することができる．バッチスケジューラは，以下の要件をみたす必要がある．
  - ジョブの投入をバッチスケジューラに依頼し，投入されたジョブの request ID を含むメッセージを標準出力に出力するジョブ投入コマンド（`qsub` 等）を提供すること．
  - ジョブ投入コマンドにより投入され実行待ちとなっているジョブおよび実行中のジョブの request ID の一覧を含むメッセージを標準出力に出力するジョブ確認コマンド（`qstat` 等）を提供すること．
  - 実行待ちあるいは実行中のジョブの request ID をパラメータとして与えると，そのジョブの実行を取り消すジョブ中止コマンド（`qdel` 等）を提供すること．
  - ジョブの実行が行われるノードと，ジョブ投入・確認・中止コマンドを実行するノードで（NFS 等により）共有されるファイルシステムが存在すること．

4 Xcrypt スクリプト

4.1 スクリプトの構成

Xcrypt スクリプトは以下の構成を持つテキストファイルであり，".xcr" を拡張子とするファイル名で保存されなければならない．

```
use qw([module-name_1 ... module-name_n] core);
script-body
```

ここで，*module-name*$_k$($1 \leq k \leq n$) はこの Xcrypt スクリプトが使用するモジュール名である．

script-body は Xcrypt スクリプト本体であり，追加の `use` 文を含むほぼ通常の Perl プログラムを書くことができる．ただし，以下の点において Perl とは異なる．

- デフォルトのネームスペース名は `main` ではなく `user` である．
- 4.2 節で述べる Xcrypt のコア機能のほか，*module-name*$_1$ ... *module-name*$_n$ のモジュールの機能を利用することができる．また，3 節で列挙した CPAN モジュールが暗黙のうちに読み込まれている．

4.2 標準 API

4.2.1 builtin::prepare

引数

- %*template*: ジョブテンプレート

返り値

- スカラコンテキストにおいては生成したジョブオブジェクトの数
- リストコンテキストにおいては生成した全てのジョブオブジェクトを要素に持つ配列

解説 引数として与えられたハッシュオブジェクト（ジョブテンプレート）の情報をもとに，0 個以上のジョブオブジェクトを生成し，返り値として返す．ジョブテンプレートは Perl のハッシュオブジェクトであり，以下の名前のメンバを持つことができる．これ以外の名前のメンバが含まれていた場合は，警告メッセージを表示し，そのメンバは無視される．また，`id`, `id@`いずれかの指定はジョブ ID の設定のために必須であり，指定なしに呼び出された場合はエラーを発生する．

1. `RANGE`n（n は自然数），`RANGES`．ただし，この両者を同時に使用してはならない．
2. `id`
3. `exe`n（n は自然数）
4. `arg`n`_`m（n, m は自然数）

5. exe, initially, finally, env, transfer_variable, transfer_reference_level, not_transfer_info, initially, before, before_to_job, before_return, before_bkup, before_in_job, before_in_xcrypt, before_in_xcrypt_return, finally, after, after_to_job, after_return, after_bkup, after_in_job, after_in_xcrypt, after_in_xcrypt_return, cmd_before_exe, cmd_after_exe, header
6. JS_で始まるメンバ名
7. :で始まるメンバ名
8. buildin::add_key 関数によって登録されたメンバ名
9. buildin::add_prefix_of_key 関数によって登録された文字列から始まるメンバ名
10. 2.–9. のメンバ名の最後に@を追加したメンバ名．ただし，同じメンバ名に対して@を追加した名前と追加していない名前のメンバを同時に指定してはならない．

prepare は呼び出されるとまず，以下の通りジョブオブジェクト（列）を生成し，メンバの設定を行う．

- ジョブテンプレートが RANGEn，RANGES をメンバに持たない場合
  単一のジョブオブジェクトを生成し，ジョブテンプレートが持つ全てのメンバと同じ名前・値のメンバを設定する．
- ジョブテンプレートが RANGE0, ..., RANGEn をメンバに持つ場合
  RANGE0, ..., RANGEn の値は，それぞれ Perl の配列 @$A_0, \ldots,$ @A_n への参照でなければならず，ジョブオブジェクトは

$$R = \{(i_0, \ldots, i_n) \mid i_0 \leq \$\#A_0, \ldots, i_n \leq \$\#A_n\} \quad (i_0, \ldots, i_n \text{は自然数})$$

  の各要素に対応してそれぞれ生成する．R の要素 $(i_0, \ldots, i_n)$ に対応するジョブオブジェクトを $J(i_0, \ldots, i_n)$ とすると，ジョブテンプレートが “$name$” または “$name$@” の名前のメンバを v_{name} を値として持つような $name$それぞれに対して（前述の通り，ジョブテンプレートが “$name$” と “$name$@” を同時にメンバとして持つことはない.），$J(i_0, \ldots, i_n)$ のメンバを以下のように設定する．RANGE0, ..., RANGEn および RANGES に対しては以下の設定は行わない．また，以下によるもの以外に，メンバ VALUE が $A[$$i_0$], ..., $A[$$i_n$] を値とする配列への参照を値として設定される．

  1. ジョブテンプレートが “$name$” をメンバとして持つ場合
     (a) $name$ = id の場合
     $J(i_0, \ldots, i_n)$ のメンバ id に
     $v_{name}.sep.i_0. \ldots .sep.i_n$
     を設定する．ただし，sep は buidin::set_separator 関数で設定されたセパレータ文字列である（デフォルトは ‘_’）．

(b) $name \neq$ id の場合
$J(i_0, \ldots, i_n)$ のメンバ"$name$" に v_{name} を値として設定する．

2. ジョブテンプレートが "$name$@" をメンバとして持つ場合

(a) v_{name} が配列 @V への参照の場合
$J(i_0, \ldots, i_n)$ のメンバ"$name$" の値として，$V[$$count$$]$ を設定する．ここで $count$は，今回の prepare 関数の呼び出しで生成される全てのジョブオブジェクトを直列に並べたときの通し番号であり，以下の式で定まる．

$$count = i_n \times B_{n-1} + \ldots + i_1 \times B_0 + i_0$$
$$\text{where} \quad B_k = \prod_{j=0}^{k} (\$\#A_j + 1)$$

(b) v_{name} が関数への参照の場合
参照先の関数 v_{name} が以下のように呼び出され，その返り値を $J(i_0, \ldots, i_n)$ のメンバ"$name$" の値として設定する．
- ジョブテンプレートへの参照が第 1 引数として渡される
- $A[$$i_0$$]$, ..., $A[$$i_n$$]$ の値がそれぞれ第 2 引数–第 $(n+2)$ 引数として渡される
- 関数の実行中，変数$user::self を用いて生成途中のジョブオブジェクト $J(i_0, \ldots, i_n)$ への参照にアクセスできる．このジョブオブジェクトには少なくとも上記 1. により決定されるメンバが設定済みである．
- 関数の実行中，変数@user::VALUE を用いて 配列 ($A[$$i_0$$]$, ..., $A[$$i_n$$]$) にアクセスできる．

(c) v_{name} がスカラ値への参照の場合
$J(i_0, \ldots, i_n)$ のメンバ"$name$" の値として，v_{name} の参照先のスカラ値を設定する．

(d) それ以外の場合
警告を発生し，$J(i_0, \ldots, i_n)$ のメンバ"$name$" の値として任意の値を設定する．

- ジョブテンプレートが RANGES をメンバに持つ場合
RANGES の値は，配列への参照 $R_0, \ldots, R_n$ からなる配列への参照

$$[R_0, \ldots, R_n]$$

でなければならず，

$$\text{RANGE0} = R_0, \ldots, \text{RANGE}n = R_n$$

が設定された場合と同様にジョブオブジェクト列の生成およびメンバの設定を行う．

- ジョブテンプレートが RANGEn，RANGES をメンバに持つかどうかによらず，ユーザ設定ファイル（**??**節）の template セクションに記述されている変数名 N と対応する値 V の組それぞれに対して，N を名前とするメンバが上記の操作によって設定されなかった場合，当該メンバを値を V として追加する．

ジョブオブジェクト（列）の生成とメンバ設定の終了後，各ジョブオブジェクトに対して，クラスメソッド $module\text{-}name_1$::new, ... $module\text{-}name_n$::new, core::new のうち定義されている最初のものを呼び出す．new メソッドには第 1 引数としてモジュール名の文字列$*class*，第 2 引数として対応するジョブオブジェクトへの参照$*self* が渡される．core::new 以外の各 new メソッドでは，モジュールの機能を達成するためにジョブオブジェクト生成時に必要となる処理が行われる．また，これらのメソッドは，

$*self* = $*class*->NEXT::new($*class*, $*self*);

に相当する処理を 1 度だけ実行することにより，次の new メソッドを呼び出す（すなわち，core::new が必ず 1 度呼び出される）ことを保証するように実装されている．

なお，core::new は Xcrypt システムが提供するものであり，必ず定義されている．core::new メソッドは，ジョブオブジェクトに対し，以下の処理を行う．ただし，以下において id は当該ジョブオブジェクトのメンバ id の値として設定された文字列，$sched$ はデフォルトスケジューラ名（??節）である．

- メンバ workdir が未設定であれば，値を文字列'.' として設定する．
- メンバ env が未設定であれば，値をデフォルトの環境オブジェクト（??節）への参照として設定する．
- メンバ JS_stdout が未設定であれば，値を文字列"id_stdout"として設定する．
- メンバ JS_stderr が未設定であれば，値を文字列"id_stderr"として設定する．
- メンバ jobscript_header，jobscript_body が未設定であれば，値を空の配列への参照として設定する．
- メンバ jobscript_file が未設定であれば，値を文字列"id_env_stdout"として設定する．
- メンバ before_in_job_file が未設定であれば，値を文字列"id_before_in_job.pl"として設定する．
- メンバ exe_in_job_file が未設定であれば，値を文字列"id_exe_in_job.pl"として設定する．
- メンバ after_in_job_file が未設定であれば，値を文字列"id_after_in_job.pl"として設定する．
- メンバ qsub_options が未設定であれば，値を空の配列への参照として設定する．
- メンバ not_transfer_info が未設定であれば，値を'dumped_environment', 'before_in_job_script', 'exe_in_job_script', 'after_in_job_script' の 4 つの文字列を要素として持つ配列への参照として設定する．not_transfer_info が定義済みであり，かつその値が配列への参照であれば，これら 4 つの文字列をその配列に追加する．それ以外の場合，警告を表示し，メンバの値をこれら 4 つ文字列を持つ配列への参照に上書きする．

- メンバ cmd_before_exe，cmd_after_exe が未設定であれば，値を空の配列への参照として設定する．
- メンバ header が未設定であれば，値を空の配列への参照として設定する．
- ジョブの状態（??節）を initialized に遷移させ，直後に prepared に遷移させる．[2]

new メソッドの呼び出し完了後，以下の通り値を返す．

- prepare 関数がスカラコンテキストで呼び出されていた場合
  生成したジョブオブジェクトの数
- prepare 関数がリストコンテキストで呼び出されていた場合
  生成した全てのジョブオブジェクトの参照からなる配列．配列中の参照の順序は任意である．

4.2.2 submit

引数

- @*jobs*: ジョブオブジェクトへの参照の配列

返り値

- スカラコンテキスト：渡されたジョブオブジェクトの数
- リストコンテキスト：渡されたジョブオブジェクトの配列自身

解説 @*jobs*に含まれるそれぞれジョブオブジェクトの参照（以下，この参照を持つ変数を$*self*とする）に対して，以下の処理を行う．

1. $*self*->make_dumped_environment メソッド（??節） を呼び出して，Xcrypt の環境をシリアライズした文字列を保存する．
2. 以下の処理を非同期に実行するスレッド[3]（ジョブスレッド）を生成し，そのスレッドを$*self*->{thread}に代入する．
   (a) ジョブの状態に応じて，以下の処理を行う．
   - ジョブがシグナル（??節）を受けていない場合
     ジョブの前回の終了状態（??節）が finished の場合，ジョブの状態を finished に遷移させ，ジョブスレッドを終了する．前回の終了状態がそれ以外の場合，何も行わない．

[2] 現在の Xcrypt においては，initialized と prepared の 2 つの状態を区別する必要がないが，後方互換性のために残している．
[3] ここでいう「スレッド」は CPAN の Coro モジュールにおけるスレッドであり，非同期に動作するが，複数のスレッドが同時に進行することはない．

- ジョブが abort シグナルを受けている場合
  シグナルを解除する．前回の終了状態が finished の場合，ジョブの状態を finished に遷移させ，ジョブスレッドを終了する．前回の終了状態がそれ以外の場合，前回の終了状態の設定を削除する．
- ジョブが cancel シグナルを受けている場合
  シグナルの解除，および前回の終了状態の設定の削除を行う．
- ジョブが invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させ，ジョブスレッドを終了する．

(b) `$`*self*`->initially`, *module-name*$_1$`::initially`, ... *module-name*$_n$`::initially`, `core::initially` のうち，定義されているものをこの順に全て呼び出す

(c) `$`*self*`->{before_in_xcrypt}`が定義されていれば呼び出す．

(d) ジョブの状態に応じて，以下の処理を行う．

- ジョブがシグナルを受けていない場合
  ジョブの前回の終了状態が設定されていないか initialized, prepared, aborted のいずれかであれば，*module-name*$_1$`::before`, ... *module-name*$_n$`::before` のうち，定義されているものをこの順に全て呼び出す．`$`*self*`->{before_to_job}` が偽であれば，さらに`$`*self*`->before` を呼び出す．ジョブの前回の終了状態が上記以外であれば何も行わない．
- ジョブが abort シグナル，あるいは cancel シグナルを受けている場合
  ジョブの状態を aborted に遷移させる．
- ジョブが invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させる．

(e) ジョブの状態に応じて，以下の処理を行う．

- ジョブがシグナルを受けていない場合
  ジョブの前回の終了状態が submitted, queued, running, done, finished のいずれかであれば何もしない．前回の終了状態が設定されていないか initialized, prepared, submitted, aborted のいずれかであれば，*module-name*$_1$`::start`, ... *module-name*$_n$`::start`, `core::start` のうち定義されている最初のものを呼び出す．`core::start` 以外の `start` メソッドでは，

  `$`*self*`->NEXT::start();`

  に相当する呼び出しにより，次の `start` メソッドを呼び出すことができるが，必須ではない．したがって，`core::start` が呼び出されることは保証されない．ただし，`core::start` が呼び出されない場合でも，以下のいずれかが保証される．
  - `start` メソッドの処理によってジョブの状態が submitted, queued にこの順に遷移させられること．さらに，queued に遷移した後，`start` メソッド

の処理またはこのジョブに対応するジョブスレッド以外のスレッドによって，ジョブの状態が running, done にこの順に有限時間内に遷移させられること．

– `start` メソッドの処理またはこのジョブに対応するジョブスレッド以外のスレッドによって，ジョブの状態が aborted に有限時間内に遷移させられること．

`core::start` が呼び出された場合，ジョブの情報や環境設定などに基づいて，ジョブスクリプトの生成やバッチスケジューラへのジョブの投入が実行される．（詳細な動作は**??**に記述する．）

- ジョブが abort シグナル，あるいは cancel シグナルを受けている場合
  ジョブの状態を aborted に遷移させる．
- ジョブが invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させる．

(f) ジョブの状態に応じて，以下の処理を行う．

- ジョブがシグナルを受けていない場合
  ジョブの前回の終了状態が設定されていないか initialized, prepared, submitted, queued, running, aborted のいずれかであれば，ジョブが done, finished, aborted のいずれかになるのを待ち合わせる．前回の終了状態が上記以外であれば何もしない．
- ジョブが abort シグナル，あるいは cancel シグナルを受けている場合
  ジョブの状態を aborted に遷移させる．
- ジョブが invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させる．

(g) ジョブの状態に応じて，以下の処理を行う．

- ジョブがシグナルを受けていない場合
  ジョブの前回の終了状態が設定されていないか initialized, prepared, submitted, queued, running, done, aborted のいずれかであれば，$*self*`->{after_to_job}` の真偽を確認し，偽であれば$*self*`->after`, を呼び出す．さらに，*module-name*$_n$`::after`, ... *module-name*$_1$`::after` のうち，定義されているものをこの順に全て呼び出す．ジョブの前回の終了状態が finished であれば何も行わない．
- ジョブが abort シグナル，あるいは cancel シグナルを受けている場合
  ジョブの状態を aborted に遷移させる．
- ジョブが invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させる．

(h) $*self*`->{after_in_xcrypt}`が定義されていれば呼び出す．

(i) `core::finally`, *module-name*$_n$`::finally`, ... *module-name*$_1$`::finally`, $*self*`->finally` のうち，定義されているものをこの順に全て呼び出す．

(j) ジョブの状態に応じて，以下の処理を行う．

- ジョブがシグナルを受けていないか，invalidate シグナルを受けている場合
  ジョブの状態を finished に遷移させる．
- ジョブが abort シグナル，あるいは cancel シグナルを受けている場合
  ジョブの状態を aborted に遷移させる．

なお，上記における `initially`, `before_in_xcrypt`, `before`, `after`, `after_in_xcrypt`, `finally` の呼び出し時の引数は以下の通りである．

- 第 1 引数：ジョブオブジェクトへの参照$*self*
- 第 2 引数以降：`@{$`*self*`->{VALUE}}`（設定されている場合のみ）

また，`start` 呼び出し時には第 1 引数としてジョブオブジェクトへの参照のみが渡される．

以上の処理の後，`submit` がリストコンテキストで呼び出されていた場合は`@`*jobs*自身を，スカラコンテキストで呼び出されていた場合は`@`*jobs*の要素数を返す．

4.2.3 builtin::sync

引数

- `@`*jobs*: ジョブオブジェクトへの参照の配列

返り値

- スカラコンテキスト：渡されたジョブオブジェクトの数
- リストコンテキスト：渡されたジョブオブジェクトの配列自身

解説 `@`*jobs*の要素数が 1 以上の場合，この配列の要素から参照されている全てのジョブオブジェクトのジョブスレッドの終了を待ち合わせる．`submit` 関数適用前のジョブオブジェクトに対して `sync` が適用されたときの動作は未定義である．

`@`*jobs*の要素数が 0，即ち無引数で呼び出された場合は，現在所属する最内の join スコープ（4.2.8 節）において生成された全てのジョブスレッドの終了を待ち合わせる．

`sync` の返り値は，この関数がリストコンテキストで呼び出されていた場合は`@`*jobs*自身，スカラコンテキストで呼び出されていた場合は`@`*jobs*の要素数である．

4.2.4 builtin::prepare_submit

引数

- `%`*template*: ジョブテンプレート

返り値

- スカラコンテキストにおいては生成したジョブオブジェクトの数
- リストコンテキストにおいては生成した全てのジョブオブジェクトを要素に持つ配列

解説　builtin::prepare(%*template*) の呼び出しと同様にジョブオブジェクト（列）の生成およびメンバの設定を行った後，生成された各ジョブオブジェクトに対して new メソッドおよび builtin::submit 関数の適用を行う．

それぞれのジョブオブジェクトに対する new メソッドおよび builtin::submit 関数の適用の順序は，1 つのジョブオブジェクトに対して new と builtin::submit がこの順序で適用されている限り任意である．

prepare_submit の返り値は，生成されたジョブオブジェクト列への参照の配列（リストコンテキスト）あるいは生成されたジョブオブジェクトの数（スカラコンテキスト）である．

4.2.5　builtin::submit_sync

引数

- @*jobs*: ジョブオブジェクトへの参照の配列

返り値

- スカラコンテキスト：渡されたジョブオブジェクトの数
- リストコンテキスト：渡されたジョブオブジェクトの配列自身

解説　以下の呼び出しと等価である．

```
sync (submit (@jobs))
```

4.2.6　builtin::prepare_submit_sync

引数

- %*template*: ジョブテンプレート

返り値

- スカラコンテキストにおいては生成したジョブオブジェクトの数
- リストコンテキストにおいては生成した全てのジョブオブジェクトを要素に持つ配列

解説　以下の呼び出しと等価である．

```
sync (prepare_submit (%template))
```

4.2.7　builtin::spawn

書式　`builtin::spawn {`$code_{exe}$`}`[`_initially_{`$code_{initially}$`}`]
[`_before_in_xcrypt_{`$code_{before\_in\_xcrypt}$`}`] [`_before_{`$code_{before}$`}`] [`_after_{`$code_{after}$`}`]
[`_after_in_xcrypt_{`$code_{after\_in\_xcrypt}$`}`] [`_finally_{`$code_{finally}$`}`] [`(%`*template*`)`];

解説　以下の呼び出し文と等価である．

```
prepare_submit(
  id => id,
  exe => sub{code_exe},
  initially => sub{code_initially},
  before_in_xcrypt => sub{code_before_in_xcrypt},
  before => sub{code_before},
  after => sub{code_after},
  after_in_xcrypt=> sub{code_after_in_xcrypt},
  finally => sub{code_finally},
  %template_id);
```

ただし，%*template*$_{id}$ は%*template* からメンバ `id` を除いたハッシュオブジェクトであり，*id* は以下のように定められる文字列である．

- %*template*にメンバ `id` が含まれていればその値．
- %*template*にメンバ `id` が含まれていなければ，これまで `prepare` で生成されたジョブオブジェクトのメンバ `id` の値および前回の実行履歴（**??**節）に登録されているジョブ ID のいずれの文字列とも重複しない，かつ “`spawned`” で始まる文字列．

4.2.8　builtin::join

書式　`builtin::join {`$code_{body}$`};`

解説　新たな「join スコープ」を構成し，そのスコープ内において $code_{body}$ を実行する．

join スコープは動的な生存期間を持つ．すなわち，$code_{body}$ の実行開始から実行終了までは，構文上 `join` ブロック外に記述されたコードの実行もこの join スコープに属しているとみなされる．また，`join` で構成される全ての join スコープの外側にグローバルな join スコープが存在し，どの非グローバルな join スコープにも属さないコード実行は，このグローバル join スコープに属しているとみなされる．