

ADON Version 3.2

ADD-ON ユーティリティー・プログラム・セット

ユーティリティー・プログラム 機能概要

日本アイ・ビー・エム システムズ・エンジニアリング株式会社

目 次

1 . 高速ソート・プログラム	1.1
主な機能	1.1
高速ソート・プログラムの処理	1.2
パフォーマンス・データ(1)	1.3
パフォーマンス・データ(2)	1.4
2 . 高速ソート拡張機能ライブラリ	2.1
主な機能	2.1
指定できるキー・タイプの種類	2.2
条件指定による入力レコードの抽出	2.3
可変長フィールドのサポート	2.4
数値フィールドのサポート	2.5
3 . ジョブ制御プログラム	3.1
主な機能	3.1
ジョブ制御プログラムの概要	3.1
分散環境下でのバッチ処理-スケジューリング	3.2
分散環境下でのバッチ処理-実行	3.3
サーバー・プロセスの監視	3.4
定義ファイル例	3.5
4 . 自動バックアップ・プログラム	4.1
主な機能	4.1
多彩なバックアップ・タイプ	4.1
GUI による簡単操作	4.2
バックアップ情報の記録	4.3

5 . コンソール・ロギング機能	5.1
主な機能	5.1
機能の概要	5.1
コマンドの自動実行	5.2
6 . FTP プログラミング・キット	6.1
API ライブラリー	6.1
ユーティリティー・コマンド	6.2
7 . コード変換プログラム	7.1
コード変換	7.1
変換モード	7.1
自動変換	7.1
フォーマット指定変換	7.1
ユーザー指定の外字変換表	7.1

高速ソート・プログラム

UNIX[®]標準 sort コマンドの機能・性能拡張版。大規模ファイル(100MB 以上)を複数の異なるタイプのキーでソートできるようにします。ソートの中間ファイルを異なる物理ボリュームに分散させることによって、処理速度の大幅な向上を実現しました。また、バイナリー・データを扱う事も可能です。

【 主な機能 】

UNIX 標準の sort コマンドを拡張

テキスト・ファイル以外にバイナリー・ファイル(固定長レコード)のソートができます。

大規模ファイルの高速ソート

中間ファイルを異なる物理ディスクに配置する事で I/O を分散して処理効率をアップします。

使用するメモリー・サイズを指定する事で、他のプロセスへの影響を考慮してシステム全体のパフォーマンス管理が可能です。

従来 の 3 分の 1 程度のソート例が実測されています。

柔軟なキー指定

ソートのタイプと順序を、各キーに対して指定できます。

- 最大 128 個のソート・キー指定
- 各ソート・キーは最大 256 バイトまで指定可能
- 各ソート・キーに対して、キー・タイプと昇順/降順の指定が可能

容易な使用方法

プログラム起動時に各種オプションを記述した制御ファイル名を与えるだけで、ソートが実行できます。

ソート・パラメーターの指定は、制御ファイルとコマンド・ラインの二通りがあり、使い方に応じてパラメーターの指定方法が選べます。

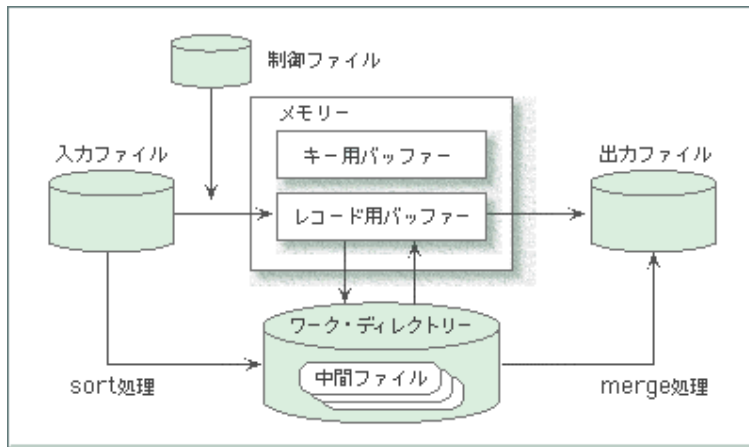
AIX4.2 以降の環境では、2GB を越えるラージ・ファイルのアクセスがサポートされます。

「ADON-EX/高速ソート拡張機能ライブラリ」と共に使用する事で、次の機能が使えるようになります。

- 出力レコード形式の変更
- キーが重複するレコードの処理
- サマリー・ソート
- 条件指定による入力レコードの抽出
- 統計情報出力

【 高速ソート・プログラムの処理 】

高速ソート・プログラムの処理は、ソート処理とマージ処理から構成されます。



1. コマンド実行の後に、ソートのオプションをコマンド・ラインまたは制御ファイルから取り出します。
2. 入力ファイルからレコードをメモリーのバッファに読みこみます。
3. バッファが一杯になった時点でソートを行ないます。
 - 入力ファイルの全レコードをバッファに読みこめた場合には、ソートの結果を出力ファイルに書き出して処理を終了します。
 - 入力ファイルの一部をソートが読みこんだ場合には、ソートの結果を中間ファイルに書きだし、入力ファイルの続きをソートします。
4. 入力ファイルのソート終了後、中間ファイルをマージして最終的に出力ファイルに書き出します。

ソート内部のロジックには、クイック・ソートとインサート・ソートの組み合わせを使用し、ハイパフォーマンスなソートのアルゴリズムを採用しています。また、メモリー上でのソート処理は、レコード全体をソートするのではなく、キーだけを抜き出した制御ブロックを作成し、イン・コアでのデータ移動も最小になるようにしています。

【 パフォーマンス・データ(1) 】

環境

モデル RS6000 44P 170
 POWER3- 400MHz × 1
 メモリー 512MB
 ディスク 内蔵：9.1GB × 1
 OS AIX 5.2.0
 測定方法 time コマンド

測定結果

レコード長	80 バイト	200 バイト	80 バイト	80 バイト		128 バイト
総件数	25,000 件	500,000 件	250,000 件	500,000 件		500,000 件
総容量	2MB	100MB	20MB	40MB		64MB
バッファ・サイズ	4MB	16MB	4MB	4MB	40MB	4MB
キー（数/長さ）	5 個/20 バイト	2 個/2 バイト	5 個/20 バイト	5 個/20 バイト		1 個/4 バイト
高速ソート	0.28 秒	10.00 秒	3.15 秒	8.01 秒	4.56 秒	11.04 秒
sort コマンド	6.65 秒	N/A	1 分 45.76 秒	3 分 34.02 秒	N/A	5 分 52.25 秒
cp コマンド	0.03 秒	1.15 秒	0.25 秒	0.47 秒	N/A	0.76 秒

【 パフォーマンス・データ(2) 】

環境

モデル	RS/6000 44P 170 POWER3- 400MHz × 1
メモリー	512MB
ディスク	内蔵：9.1GB × 1
OS	AIX 5.2.0
測定方法	time コマンド

測定結果

レコード長	226 バイト		
総件数	約 200 万件		
総容量	約 450MB		
バッファ・サイズ	20MB	20MB	200MB
ワーク・ディレクトリー数	8	14	8
キー (数/長さ)	1 個/8 バイト		
実行時間	4 分 21 秒	3 分 43 秒	2 分 36 秒

約 200 万件、450MB のデータをソートした場合のパフォーマンスです。バッファの大きさや、ワーク・ディレクトリーの数で実行時間に差が出ます。バッファがあまり大きくない場合は、ワーク・ディレクトリーの数が多い方がマージ処理の効率が良くなるので、全体の処理時間が短くなります。バッファが大きい場合は、一度にソートできるレコードが多くなり、早く処理できます。

ADON-EX/高速ソート拡張機能ライブラリ

ADON-EX/高速ソート拡張機能ライブラリは、ADON/高速ソート・プログラムの導入された環境に、追加導入する事で次の機能を ADON/高速ソート・プログラムから使用できるようになります。

【 主な機能 】

出力形式の指定

ソートの出力時に各フィールドの順番の入れ換え/削除、定数の挿入などを定義して、レコード形式を変更して出力ファイルに書き出すことができます。

重複レコードの処理

キーが同じレコードについて、その範囲の最初あるいは最後のレコードだけを出力させることができます。

サマリー・ソート

キーが同じレコードについて、指定したフィールドの値の総合計を計算します。計算された合計値は、キーが同じ範囲の最初あるいは最後のフィールドと置き換えられて出力されます。合計値を計算する対象のフィールド・タイプは 2 バイト又は 4 バイトの符号付き二進整数あるいは符号無し二進整数、または数値フィールドです。合計値の計算中にオーバーフローが発生した場合は、オーバーフローが発生する直前のレコードの合計対象フィールドを中間値と置き換えて出力し、後続レコードの合計値を改めて計算して処理を継続します。

統計情報出力

指定されたファイルに次の情報を書きだします。

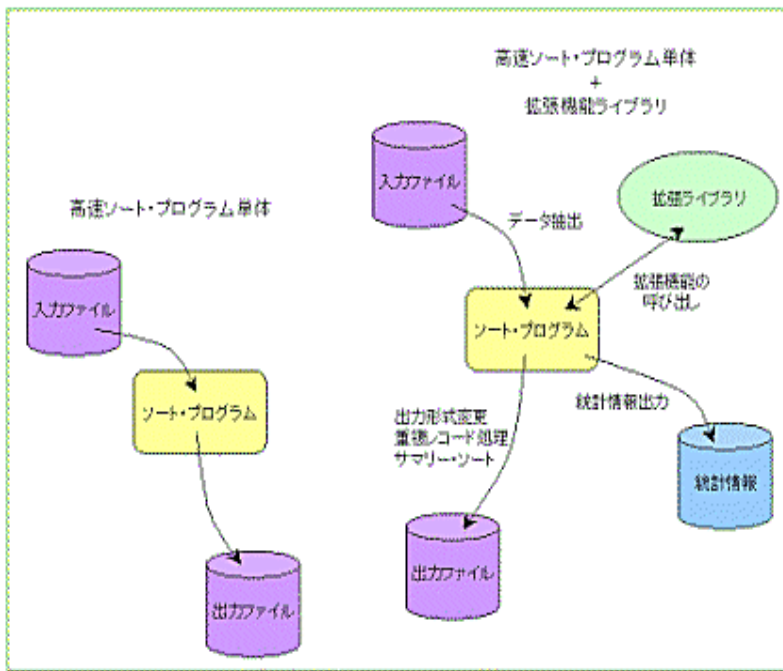
- 各キー・フィールドの開始位置、長さ、種類、昇順/降順
- 同じキー毎のレコード件数とキーの値
- 同じキーの最大/最少レコード件数と、それぞれのキーの値
- キーの数
- キーあたりの平均レコード数

データ抽出

条件指定により入力レコードを選択してソートを行う事ができます。

可変長レコードの処理

任意のフィールド区切り文字を指定する事で、CSV の様な可変長フィールドのレコードを処理することができます。



【 指定できるキー・タイプの種類 】

キーのタイプは次のものが指定できます。

- キャラクター
- EBCDIC 順キャラクター
- 二進整数(2 バイト又は 4 バイト)
- パック十進数
- 数値フィールド

EBCDIC 順のソートでは、ASCII-EBCDIC の変換テーブルを指定できます。この変換テーブルでは、全 256 文字について変換するコードを指定できるので、単なる ASCII-EBCDIC の変換以外に、独自の重み付をした並び替えルールとして利用する事も可能です。例えば、辞書順(AaBbCc...YyZz)や電話番号順(ABC...YZabc...yz)というルールでソートする事ができます。

数値フィールドにより、ASCII 文字で表現された数字を、文字ではなく数値として大小関係を比較する事ができます。数値フィールドは、 $n(x,y)$ の形式で定義され、全体で x 桁、少数点以下 y 桁を意味します。有効な桁数は、符号と少数点を含めて最大 20 桁、そのうち数字部分は 18 桁です。

【 条件指定による入力レコードの抽出 】

条件式を与えて、入力レコードのうち条件に一致するものを抽出又は除外してソートすることができます。条件式は

<フィールド定義>,<比較演算子>,<フィールド定義又は定数>

の形式で指定でき、複数の条件式を「AND(論理積)」「OR(論理和)」で結合する事も可能です。また、括弧による条件式の優先順位も指定できます。括弧のネスト・レベルにプログラム上の制限はありません。

フィールド定義でサポートされるフィールド・タイプは次のものです。

- 文字
- 符号付き二進整数(2 バイト又は 4 バイト)
- 符号無し二進整数(2 バイト又は 4 バイト)
- バック十進数
- EBCDIC 順の文字
- バイト・ストリーム
- 数値フィールド

定数の種類は次のものが使えます。

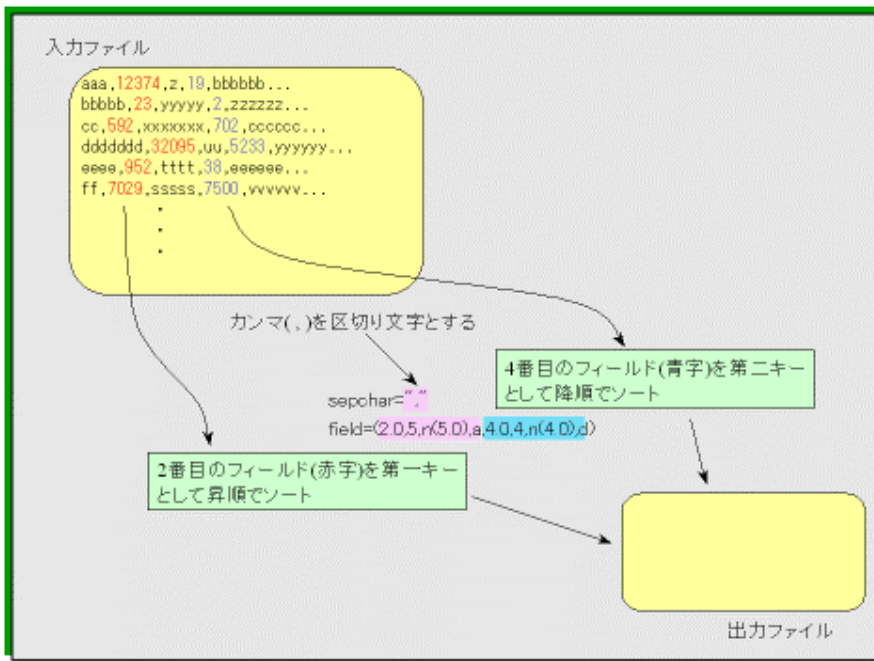
- 文字列
- 16 進数
- 十進数
- 数値

比較演算子は次の種類がサポートされます。

- EQ (等しい)
- NE (等しくない)
- GT (...より大きい)
- GE (...以上)
- LT (...より小さい)
- LE (...以下)

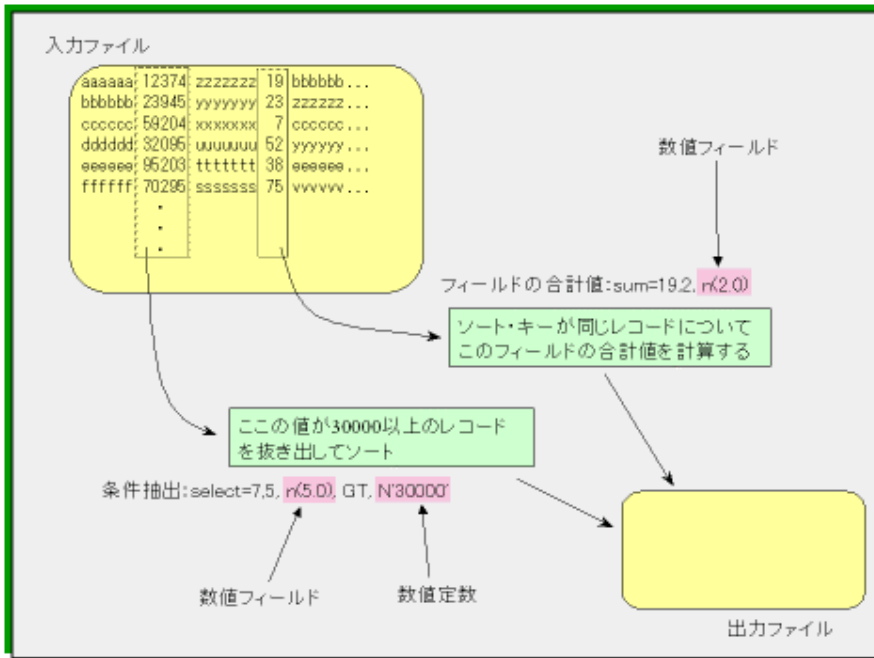
【 可変長フィールドのサポート 】

フィールド区切り文字を指定することで、特定の文字で区切られた可変長フィールドを扱うことができるようになります。例えば、各フィールドがカンマ「,」で区切られた CSV ファイルを入力ファイルとして、ソート処理を行うことができます。



【 数値フィールドのサポート 】

数値フィールドとは、ASCII 文字で表現された数字を「文字」ではなく「数値」として大小関係を判断するフィールド・タイプで、符号や少数点を含んだ数字に対してサマリー・ソートやデータの抽出などの拡張機能が使えます。



数値フィールドは $n(x,y)$ の形式で定義され、全体で x 桁、少数点以下 y 桁を意味します。数値フィールドは、符号と少数点を含めて最大 20 桁で、そのうち数字部分は最大 18 桁となります。

ジョブ制御プログラム

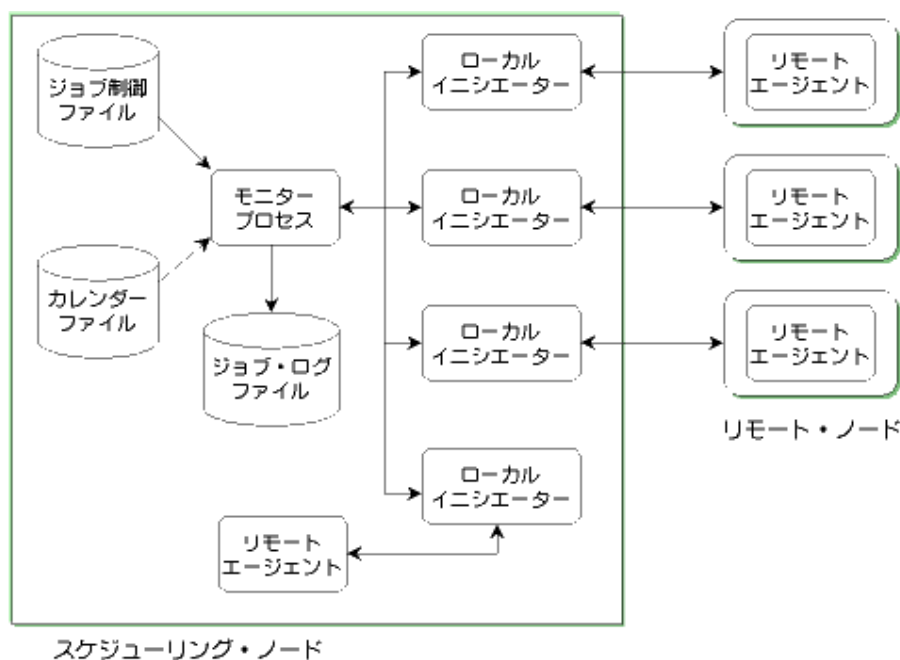
ジョブ制御プログラムは、メインフレームと同様のジョブ・ネットワーク定義によるジョブ・スケジューリングを実現します。

【 主な機能 】

- 前ジョブの実行結果により後続ジョブの実行を制御します。
- 指定時刻(日時)にジョブを実行します。
- 指定時間を越えて実行しているジョブをキャンセルします。
- ジョブが異常終了した場合には、システム・コンソール通知します。
- ジョブの実行結果をログに記録します。
- ウォーム・スタート/コールド・スタートをサポートします。
- ジョブ・クラスをサポートします。
- リモート・ホストでのジョブの実行をサポートします。

【 ジョブ制御プログラムの概要 】

ジョブ制御プログラムは、モニター・プロセス、ローカル・イニシエーター、リモート・エージェントの3種類のプロセスで稼働します。



モニター・プロセスは、ジョブ制御プログラムの中心となるプロセスで、ジョブ制御ファイルを読みこんでジョブ・ネットワークに基づいたスケジューリングを行ないます。ジョブの実行状況は、モニター・プロセスを通じてジョブ・ログファイルに書き込まれます。

モニター・プロセスはスタートするとジョブ制御ファイルの指定に従ってローカル・イニシエーターを複数スタートさせます。各ローカル・イニシエーターはリモート・ノード定義に従って、リモート・ノードのリモート・エージェントと通信を開始。スケジュールさ

れたジョブにより実際に実行されるプログラムは、ローカル・イニシエーター経由でリモート・エージェントに指示され、リモート・ノード上で実行されます。

ジョブの依存関係

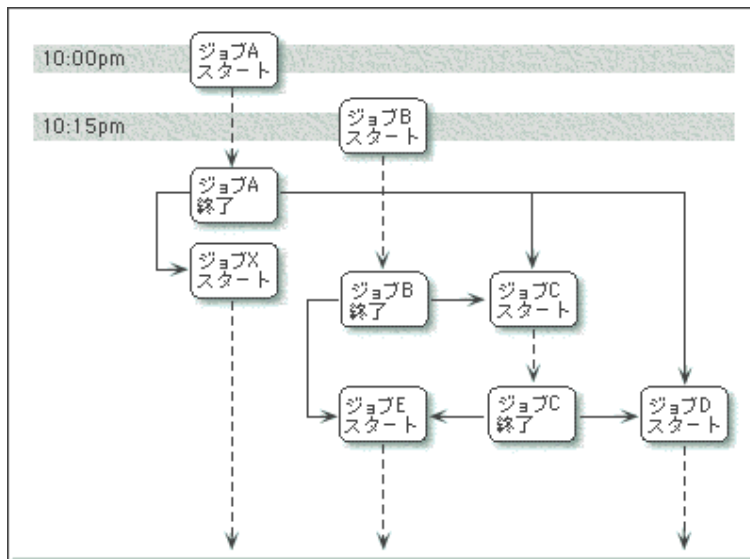
ジョブ制御プログラムに定義される各ジョブは単独での定義も可能ですが、多くの場合はジョブ・ネットワークに従った依存関係を持ちます。

ジョブの実行クラス

各ジョブは1つの実行クラスが定義されます。各イニシエーターは、最大4つの実行クラスが定義され、それぞれに定義された実行クラスと一致するクラスのジョブを順次実行します。このことにより、複数のイニシエーターに定義されたクラスに属するジョブは実行される確立が高まり、結果的には他のジョブより優先処理されます。

【 分散環境下でのバッチ処理 - スケジューリング 】

ジョブ制御プログラムでバッチ処理を行なう場合の例



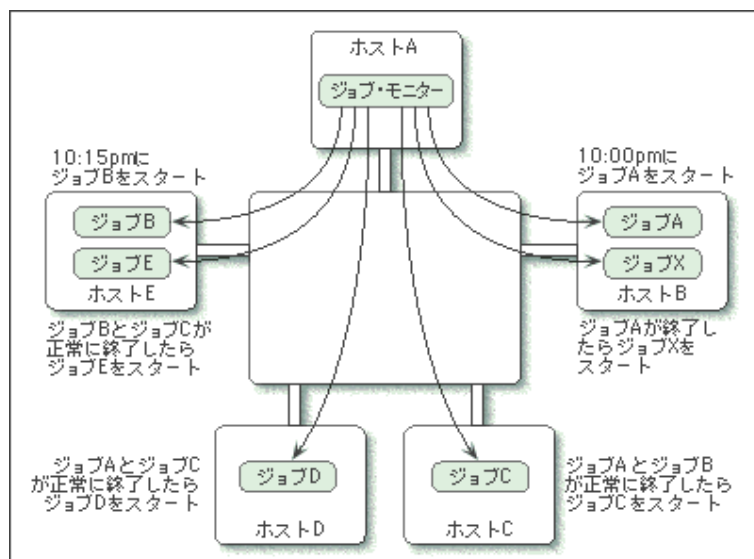
1. バッチ処理のスケジュールのサンプルを考えます。
2. 時刻指定で動くジョブが2つあります。
ジョブ A は、10:00pm にスタート。もう1つの時刻指定ジョブであるジョブ B は、10:15pm にスタート。後続ジョブが前提ジョブの終了に従って実行されます。
3. ジョブ X は、ジョブ A の終了後直ちに実行開始。
4. ジョブ C はジョブ A とジョブ B の終了を待って実行されます。
5. ジョブ D はジョブ A とジョブ C の終了後に、ジョブ E はジョブ B とジョブ C の両方が終了後に実行されるものとして。

このようなスケジューリングをシェル・スクリプトだけで組むのは容易ではありません。ネットワーク上の複数の AIX で分散させて実行する事はさらに困難です。例えば、ジョブ A とジョブ B がともに CPU バウンダリーのジョブだった場合、同じマシン上で実行すると

お互いに CPU を取り合い、その結果エラー・タイムが単独で実行した時よりも延びてしまいます。これを別々の AIX 上で実行させれば、本来の処理時間でそれぞれのジョブを終了させることができます。

【 分散環境下でのバッチ処理 - 実行 】

ジョブ・スケジュールを実際に分散環境下で実行した場合

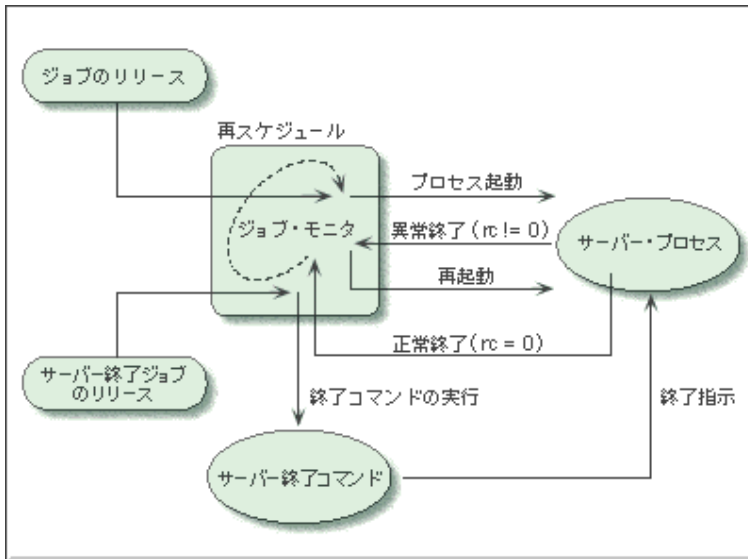


ジョブ・モニターはホストAでジョブのスケジューリングを行ないます。

1. 10:00pm にジョブ A をホスト B で実行。
2. 10:15pm にジョブ B をホスト E で実行。
3. ジョブ A が終了すると後続ジョブで実行条件のそろっているジョブ X が直ちに実行されます。上記の場合は、ジョブ A と同じくホスト B で実行されています。
4. ジョブ B が終了すると、ジョブ A とジョブ B の終了を待っていたジョブ C はジョブ X と平行処理される目的でホスト C にスケジュールされ、ジョブ C の終了を待ってジョブ D がホスト D で、ジョブ E がホスト E でそれぞれ実行されます。これらのジョブもジョブ X と独立して平行処理できるので、それぞれ異なるホストにスケジュールします。

【 サーバー・プロセスの監視 】

システム稼働中は常駐すべきサーバー・プロセスを、繰返しスケジュールされるジョブとして定義することでプロセスの監視・再起動が可能です。



リピート定義（再スケジュールリングの定義）されたジョブは、ジョブ・モニターの起動時にホールド・ジョブとしてスケジュールされます。ホールド状態を解きサーバー・プロセスを起動するには、`jobcmd` コマンドでジョブをリリースします。途中、プロセスが異常終了 (`rc != 0`) した場合は、リターン値が 0 意外の時の後続ジョブとして、再度サーバーのジョブをリリースする `jobcmd` を定義することで自動的なサーバー・プロセスの再起動を行なう事ができます。

また、直接サーバー終了コマンドをオペレーターが実行しなくても、サーバー終了ジョブとして定義する事で該当のジョブをリリースすればサーバーを正常に停止することができます。

【 定義ファイル例 】

ジョブ制御プログラムには定義ファイルにてジョブの起動日時や起動条件、リモート・ノードの設定、休日の指定を定義することができます。

ジョブ制御ファイル

ジョブ制御ファイルは、時刻や実行コマンド、クラスなどが「項目 = 値」で分かりやすく記述できます。

```
#      JOB Shedule Table
#
maxprocs      = 4  # MAX No. of tasks to run concurrently
maxjobs       = 10 # MAX No. of jobs to be scheduled
textarea      = 3  # "program", "message" text area
maxdependency = 8  # MAX No. of dependencies
init_class    = AB,FY,C,AZ # class
jobxxx:
daily         = 14:35
program       = '/home/toshiaki/jobctl/dev4/xxx'
class         = A
succesxxx:
jobxxx        = 0
program       = 'ksh -c "echo `date` && banner jobxxx ok"'
class         = F
failxxx:
jobxxx        > 0
program       = 'ksh -c "echo `date` && banner jobxxx ng"'
class         = Y
tar:
succesxxx:    = 0
weekly        = friday
clock         = 22:00
program       = '/usr/bin/tar cvf /dev/rmt0 /usr/lib/X11/fonts'
time          = 300
class         = C
alart         = console
message       = 'JOB Name [tar] abend...'
succestar:
tar           = 0
program       = 'ksh -c "echo `date` && banner tar success"'
class         = Z
failtar:
tar           > 0
program       = 'ksh -c "echo `date` && banner tar failed"'
class         = Z
```

リモート・ノード・ファイル/カレンダー・ファイル

リモート・ノード・ファイルはジョブを実行したいリモート・ノードのホスト名を記述します。上から順番にイニシエーター番号が割り当てられます。

カレンダー・ファイルは、休日制御を行ないたい場合に使用します。休日は、曜日、月日、月と曜日等の形式で指定できます。

リモート・ノード・ファイル

```
oscar  
whale  
ois1  
inoki
```

カレンダー・ファイル

```
saturday  
sunday  
1/1  
1/2  
1/3  
1/monday(2)  
2/11  
4/29  
5/3  
5/4  
...
```

自動バックアップ・プログラム

自動バックアップ・プログラムはユーザーの指定したスケジュールに従って、データのバックアップを自動的に行なうユーティリティです。GUI に従って、バックアップのタイプや実行時間、バックアップするファイルなどを指定するだけで、UNIX のコマンドについての知識がなくても定期的にバックアップを取ることができます。

【 主な機能 】

多彩なバックアップ・スケジュール

マンスリー・バックアップ(日付指定)、ウィークリー・バックアップ(単曜日指定)、デイリー・バックアップ(複数曜日指定)と、異なるバックアップのタイプをスケジュールすることができます。また、特定の日に例外的なバックアップの中止・実行を行なうように指定できるので、お使いになる方の環境に合わせて年間を通じたバックアップのスケジュール管理が可能です。

GUI による簡単操作

操作はすべてマウスで行なえるため、お使いになる方は、UNIX のコマンドに精通している必要はありません。また、CDE の環境では、ヘルプ・ビューアーを使用したヘルプが各画面で用意され、操作に迷うことはありません。

バックアップ情報の記録

バックアップ先のメディアには、テープ、ディスケット、ファイルが指定できます。バックアップの際には、バックアップのタイプ、保存期間、バックアップを行なった日付を同時に記録するため、誤って上書きする心配もありません。

【 多彩なバックアップ・タイプ 】

自動バックアップ・プログラムは 3 種類のバックアップ・タイプをサポートします。

月次バックアップ

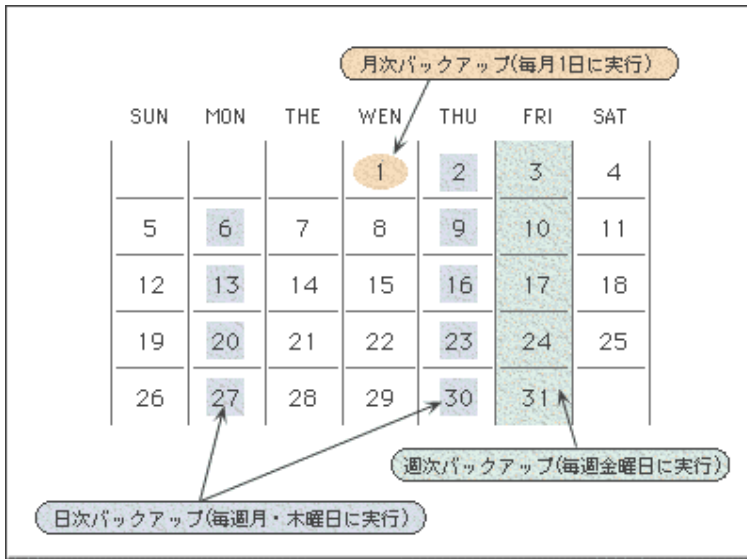
月次バックアップは、毎月決まった日にバックアップを実行します。

週次バックアップ

週次バックアップは、毎週決まった曜日にバックアップを実行します。週次バックアップで指定できる曜日は 1 つです。

日次バックアップ

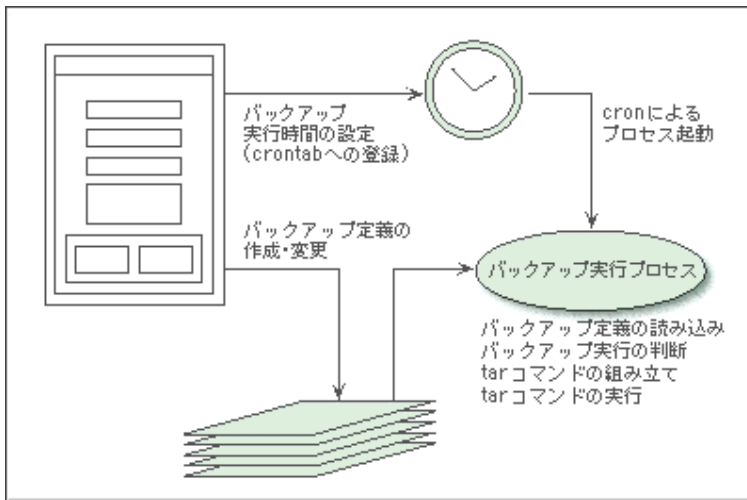
日次バックアップは、毎週決まった複数の曜日にバックアップを実行します。



さらに例外処理として、特定日バックアップ・タイプがあります。特定日バックアップ・タイプは、月次、週次、日次のいずれのバックアップも指定されていない日に、例外的にバックアップを行なう、また逆にバックアップがスケジュールされている日に例外的にバックアップを中止するように指定するものです。

【 GUI による簡単操作 】

バックアップ・スケジュールやバックアップの実行時間の指定は GUI により、ほとんどの操作がマウスだけで行なえます。CDE 環境では、ヘルプ・ビューアーによるヘルプ表示もサポートされます。



バックアップ・スケジュールはユーザー毎に定義ファイルに記録されバックアップ実行プロセスがこの定義ファイルを読みこんで実際のバックアップを行ないます。バックアップの実行時刻を指定すると、バックアップ実行プロセスが crontab に登録され指定された時間に cron からバックアップ実行プロセスが起動されます。

【 バックアップ情報の記録 】

バックアップ・プログラムは、バックアップ・タイプや保存期間、バックアップの実行日を自動的にバックアップ・メディアに記録します。さらにバックアップを取ったメディアに再度バックアップを行なう場合、自動バックアップ・プログラムはまず記録されているバックアップ情報を検査して、保存期間内のメディアに対してはバックアップを実行しません。

このことにより、重要なデータに誤って上書きバックアップすることを防止します。

コンソール・ロギング機能

多数のワークステーションが存在するクライアント/サーバー環境や、大規模データベース・サーバーが稼働している環境で威力を発揮します。

【 主な機能 】

コンソール・メッセージのディスクへの記録

`/dev/console` に書き込まれるコンソール・メッセージを監視し、ディスクに記録します。同時に、指定された端末へのメッセージ表示も行なうので、スムーズにユーザーへのオンタイム表示ができます。

他のワークステーションへのコンソール・メッセージの転送

コンソール・メッセージを、TCP/IP ネットワークを通じて、他のワークステーションで稼働しているコンソール・ロギング機能へ転送することができます。これによって、コンソール・メッセージを一台のワークステーションで集中表示させることが可能です。

過去のコンソール・メッセージの再表示

端末から消えてしまった過去のコンソール・メッセージを再表示させることができます。重要なメッセージがスクロール・アウトされても、後から確認できます。

自動コマンド実行機能

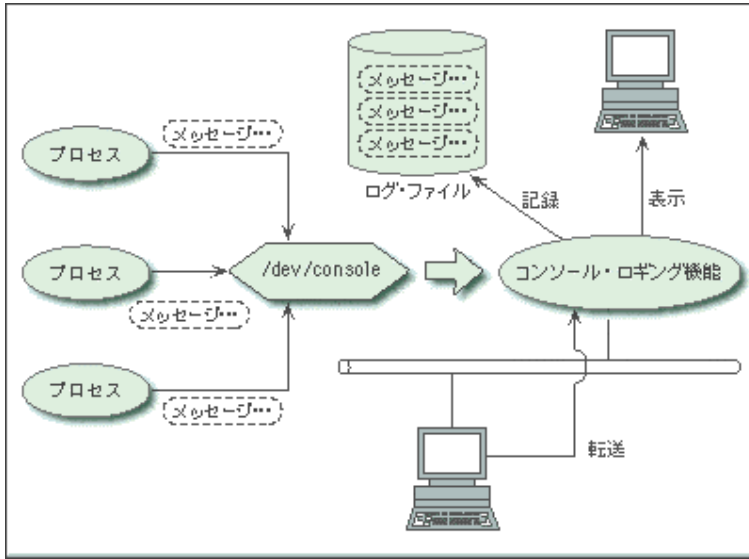
あらかじめ登録したコンソール・メッセージをトリガーとして、特定のコマンドを自動的に実行させることができます。コマンドは UNIX 標準のコマンドに限らず、ユーザーの作成したプログラムやシェル・スクリプトでも OK。また設定により、トリガーとなったメッセージをパラメーターとしてコマンドに渡す事ができ、メッセージの内容に応じてプログラムで処理を変えることも可能です。

ログのプリント機能

`prtcons` コマンドにより、現在までのコンソール・ログをファイルとして印刷できます。また、`prtcons` コマンドのオプションにより、現在のログ・ファイルをクリアすることも可能です。

【 機能の概要 】

ADON/コンソール・ロギング機能は、`/dev/console` に書き込まれたメッセージを拾い、ログ・ファイルに記録すると同時に端末に表示します。コンソール・ロギング機能によって記録されたメッセージには 1/100 秒単位でのタイムスタンプがつくので、単にコンソールをファイルにリダイレクトした場合とは異なり、どのメッセージがいつ出力されたのかが明確になります。また、コンソールに書き込みを行なったプロセスは、ユーザー作成のプログラム、UNIX のコマンド、あるいは UNIX カーネル自身であるかは問いません。このため、アプリケーションのログ・ファイルとして `/dev/console` をつかうことで、システム全体の活動ログを作成する事が可能になります。

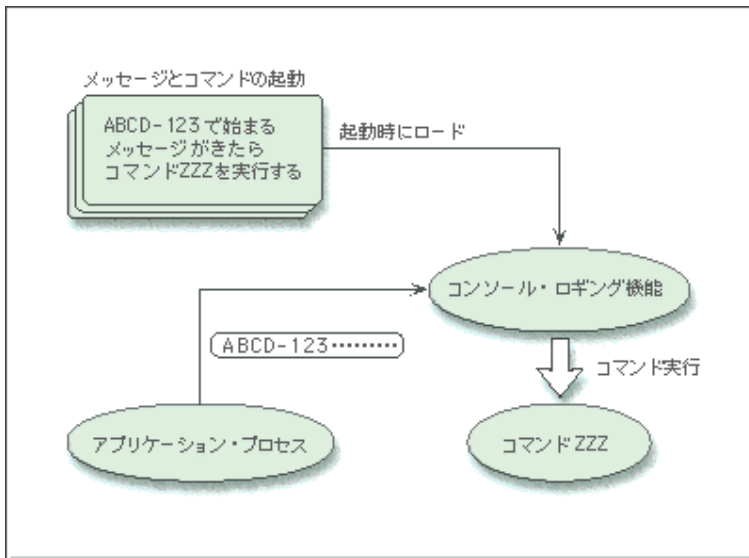


複数プロセスで構成されるアプリケーション・システムを作成した場合、各プロセスが独自のログ・ファイルに記録するとプロセス間の関わりが分かりにくくなります。このような場合でもコンソール・ロギング機能を利用する事で、各プロセスの出力したメッセージが時系列に一括管理されるので、プロセス間の関わりが分かりやすくなります。

さらに、メッセージ転送で集中管理することにより、分散環境下でのクライアントとサーバー・プロセスの関係についても明確にできます。

【 コマンドの自動実行 】

あらかじめ特定のメッセージとコマンドを登録しておくことで、該当するメッセージがコンソールに書き込まれたタイミングで、登録済みのコマンドを自動的に実行させることができます。自動実行に登録されるコマンドは、UNIX 標準のコマンドに限らず、シェル・スクリプトやユーザー作成のプログラムなど、実行可能であれば限定されません。



メッセージの登録は、比較文字列、比較開始オフセット、比較文字列長で登録するので、トリガー対象のメッセージに応じて柔軟な指定が可能です。さらに、コンソール・メッセージがネットワークで転送されることに対応して、登録メッセージ毎に対象ホストを、自分自身のみ、特定のリモート・ホストのみ、全ホスト、と3通りに設定することができます。

応用例

~ アプリケーションのエラー発生事に snmp トラップを送信する ~

プログラムから直接 snmp トラップを送信するには、ソケット関係のシステムコールを使いこなすだけでなく、snmpに関する知識も必要となります。これらの処理を逐一アプリケーション・プログラムに組みこんでいくと、その開発量は無視できない大きさになります。

これをアプリケーションに直接コーディングするのではなく、snmp トラップを送信する汎用的なプログラムを作成し、コンソール・ロギング機能からこのプログラムを起動する形にすると、アプリケーション・プログラムは、重大なエラーが生じた事を示すメッセージをコンソールに書き出すだけで snmp トラップを送信する事ができるようになります。同時に、そのメッセージがディスク上にログとして記録されるのでアプリケーション・ログに改めて書き込む必要もありません。

AIX の環境では、システム管理ツールとして SystemMonitor/6000 を導入すると、snmp トラップ送信用のコマンドが SystemMonitor/6000 から提供されるので、snmp トラップ送信に関する開発はほとんど必要ありません。

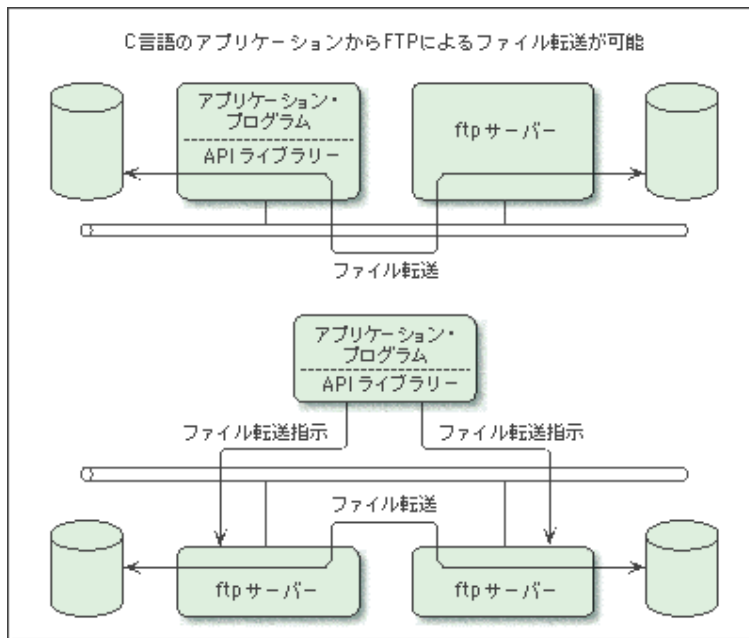
FTP プログラミング・キット

TCP/IP の FTP を利用したファイル転送を C 言語で書かれたプログラムや、シェル・スクリプトから簡単に実行するためのツールです。

ftp サーバーと直接通信するので、TCP/IP の FTP をサポートする OS であれば UNIX に限らず、あらゆるプラットフォームのサーバーとのファイル転送が可能です。

【 API ライブラリー 】

FTP でファイル転送を行う際に必要となるプロトコルの処理はライブラリー内で行なわれるので、プログラマーは FTP に関わる RFC の知識を必要とせずにファイル転送のプログラム開発ができます。また、用意されたサブルーチン群は基本的に ftp コマンドのサブコマンドに対応しているので、新しい API を覚える煩わしさもほとんどありません。



ライブラリーは、ftp コマンドの get、put というようなサブコマンドの機能がサブルーチンとして用意されています。これらのサブルーチンを使用することで、C 言語で書かれたアプリケーションから簡単に FTP によるファイル転送を行うことができますようになります。

ファイル転送には RFC で定められた標準プロトコル(FTP)を使用するので、ftp サーバーの実装された OS であればファイル転送の相手は UNIX に限らずホストや PC でも可能です。さらに、二つの ftp サーバー間での第三者代理ファイル転送機能もあるので、直接自分とのファイル転送だけでなく、異なる TCP/IP ホスト間でのファイル転送の指示もできます。

GUI を持ったファイル転送アプリケーション

API ライブラリーを使用しない場合

ftp サーバーと通信するためには、ソケットによる TCP/IP のプロセス間通信をコーディングする必要があります。このため、プログラマーは、socket()、

send()、recv()などのソケット関連システムコールについて熟知する必要があり、さらに ftp サーバーとの通信のために RFC で定められたファイル転送プロトコルの知識も必要です。

API ライブラリーを使用した場合

例えばファイルをリモートからローカルへ転送する場合、ftpget()サブルーチン 1 つで目的が果たせます。プログラマーは、RFC に関する知識を必要としないだけでなく、ソケット関連のシステムコールについてもこだわらずに本来のアプリケーション開発に専念する事ができます。

【 ユーティリティー・コマンド 】

シェル・スクリプトで利用するコマンド群です。ftp コマンドのサブコマンドに相当する機能が独立したコマンドとして使用できるので、FTP によるファイル転送のシェル・スクリプト化が簡単に行なえます。

ユーティリティー・コマンドは、FTP の各機能毎に用意されたフロント・エンド・コマンド群と、実際に ftp サーバーと通信をしてファイル転送を行うデーモン・プロセスの 2 つがあります。

フロント・エンド・コマンドとデーモン・プロセスは、メッセージ・キュー(msg_q)を通じて要求と処理結果を通信します。このことで、ftpkconnect コマンドでデーモン・プロセスが起動されてから ftpdisconn コマンドでデーモン・プロセスが終了するまで ftp サーバーとの通信は維持されます。そのため、フロント・エンド・コマンドを実行する度に ftp サーバーと TCP/IP のコネクションを設定しなおす事はありません。

シェル・スクリプトによるバッチ・ファイル転送

ADON/FTP プログラミング・キットを使わずにシェル・スクリプトでファイル転送した場合、ftp コマンドへのリダイレクトを行うなど、.netrc ファイルにマクロを生成するスクリプトを組み上げてから ftp コマンドを実行するというような手間のかかる方法をとる必要があります。また、このような方法では、スクリプト内で得られるファイル転送のリターン・コードは ftp コマンド自身のリターン・コードとなるため、多くの場合「正常終了」となりファイル転送自身の結果を得る事は困難です。

ホスト名「artemis」から「file1」を「file2」として取ってくる例

```
#!/bin/ksh
:
:
ftpconnect artemis

ftpget artemis file1 file2
RC=$?
if (( $RC != 0 )); then
echo "ERROR!! . . . ."
fi

:
:
ftpdiscn
:
:
```

ユーティリティー・コマンドを
使用した場合

```
#!/bin/ksh
:
:
USERID=$1
PASSWD=$2
ftp srtemis << EOD
$USERID
$PASSWD
get file1 file2
bye
EOD

:
:
:
```

ユーティリティー・コマンドを
使用しない場合

これを ADON/FTP プログラミング・キットのユーティリティー・コマンドを利用する事で、ftpconnect コマンドでデーモン・プロセスを起動し、ftpget コマンドでリモートのファイルをローカルに転送するだけで実現できます。

コード変換プログラム

コード変換プログラムは、NEC**および富士通**のホスト漢字コードとシフト JIS との相互変換を行なうソフトウェアです。

【 コード変換 】

NEC と富士通ホストの漢字コードとシフト JIS との相互変換

- JIPS-E シフト JIS
- JEF シフト JIS
- 変換コマンドと、C 言語用の API ライブラリを提供

【 変換モード 】

コマンドでのコード変換には「自動変換」と「フォーマット指定変換」の 2 種類のモードが用意され、変換元のファイルの形に応じて柔軟に対応できます。

【 自動変換 】

SI/SO コードを判別して SBCS/DBCS をそれぞれ自動で変換します。

【 フォーマット指定変換 】

バイト単位でレコード・フィールドの変換属性を指定します。フォーマット指定変換の場合は、ホスト漢字コードからシフト JIS に変換する場合、パック十進数、ゾーン十進数の変換も可能です。

【 ユーザー指定の外字変換表 】

外字変換テーブルをユーザーが指定できるので、外字の変換時についてもお客様毎に対応が可能です。また、外字変換テーブルに用いるファイルはパラメーターとして指定できるので、変換作業毎に異なる外字テーブルを利用する事も可能です。