作業キュー・マネージャの使用

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

作業キュー・マネージャの使用
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

作業キュー・マネージャの使用

作業キュー・マネージャは、プログラムによって作業を複数の同時プロセスに分散することによってパフォーマンスを向上させることができる、InterSystems IRIS の機能です。作業キュー・マネージャの登場以前は、JOB コマンドを使用してアプリケーション内で複数のプロセスを起動したり、カスタム・コードを使用してプロセス (およびその結果として生じる障害) を管理していたかもしれません。作業キュー・マネージャは、プロセス管理のオフロードを可能にする、効率的で簡単な API を提供します。

インターシステムズのコードでは、内部的に複数の場所で作業キュー・マネージャが使用されます。以下の各セクションで概要を説明するように、各自のニーズに合わせて作業キュー・マネージャを使用することができます。

詳細は、"クラス・リファレンス" で %SYSTEM.WorkMgr クラスと Config.WorkQueues クラスを参照してください。

1 背景

コンピュータ・ハードウェアの発展における最近のイノベーションは、高パフォーマンス、マルチプロセッサ、またはマルチコアのアーキテクチャに向かう傾向があります。その一方で、メモリとネットワーク・デバイスの速度はごく緩やかな向上にとどまっています。インターシステムズは、こうした傾向に対応して、以下の原則に従って作業キュー・マネージャを開発しました。

- ハードウェア・リソース (CPU と I/O を含む)、メモリ、およびネットワーキング・デバイスが固定されている。
- InterSystems IRIS でできるだけ効率的にハードウェア・リソースを使用し、実行するビジネス・タスクの速度を最大化する必要がある。
- 最大の効率性を実現するために、InterSystems ObjectScript コードの実行時に発生する可能性のある CPU の低使用率を作業キュー・マネージャで改善する必要がある。
- CPU の低使用率に対処するための方法にキューイングと優先順位付けが含まれる。

InterSystems IRIS データ・プラットフォームは全体的にシステム内のハードウェア・リソースをできるだけ効率的に利用するよう設計されていますが、このプラットフォームの作業キュー・マネージャ機能は、最新のハードウェア構成で利用可能な追加の CPU リソースを活用することに特化した設計になっています。作業キュー・マネージャは、次の 2 つの重要な役割を果たします。

- 大規模なプログラム・タスクを小さいチャンクに分割して複数の同時プロセスで実行できるようにするフレームワークを提供する。一度に複数の CPU を使用することで、ワークキュー・マネージャは大規模なワークロードの処理時間を大幅に短縮します。
- システム・タスクに対して一度にアクティブになる InterSystems IRIS ジョブの数を管理することによって、システム上の合計 CPU 負荷を制御する。アクティブなジョブの数は、カテゴリごとに制限されます。詳細は、"カテゴリの管理" を参照してください。

1.1 ObjectScript の CPU 使用率

一般的に、ObjectScript コードは単一のプロセスで実行され、1 つのプロセッサ・コアのみを使用します。トランザクション間で処理する命令とグローバル参照が比較的少ないトランザクション・データベース・アプリケーションでは、このアプローチは効果的です。実際、InterSystems IRIS データ・プラットフォームの重要な特徴は、トランザクション・ワークロードの大規模なスケーラビリティです。このプラットフォームでは、非常に多くのユーザによって比較的小さい作業単位が一度に大量に要求された場合、その処理が最適化されます。

より新しいタイプのワークロード (例えば分析ワークロード) の中には、InterSystems IRIS が構築当初に最適化対象としていたワークロードとは異なるものがあります。例えば、最新のワークロードには、数百万の行に対してさまざまな操作を実行する必要がある 1 つの SQL クエリの処理が含まれる場合があります。このようなワークロードの処理を高速化するため、インターシステムズは作業キュー・マネージャを開発しました。作業キュー・マネージャは、モノリシックなワークロードを小さなチャンクに分割し、そのチャンクを並列処理して各チャンクの結果を親プロセスに返し、その後、親プロセスが結果をユーザに戻します。言い換えると、作業キュー・マネージャはプロセス間キューに似たメカニズムです。このメカニズムにより、InterSystems IRIS でアプリケーションを構築する開発者は、大きいタスクを小さいタスクに分割して並列処理することができます。

1.2 作業キュー・マネージャの特徴

作業キュー・マネージャには、以下のように重要な特徴がいくつかあります。

- 低遅延および低オーバーヘッド
- スケーラビリティ
- オペレーティング・システムとの連携
- 柔軟性
- 高レベルの制御およびレポート

1.2.1 低遅延および低オーバーヘッド

作業キュー・マネージャは、低遅延と低オーバーヘッドを実現するように設計されています。例えば、ご使用のシステムで順次処理するのに 10 分かかるプログラム・タスクについて考えてみてください。システムにコアが 10 個ある場合、タスクを分割して、各コアで 1/10 の作業を並列で処理した方が効率的です。実際、タスクを分割してタスクの各部分をキューイングし、ワーカ・ジョブを開始し、タスクの各部分の完了通知を収集するのに伴うオーバーヘッドに追加の時間がかからなければ、10 倍速く結果に到達できます。作業キュー・マネージャは、オーバーヘッド・タスクが低遅延となるように設計されています。

1.2.2 スケーラビリティ

パフォーマンスを最大化するため、作業キュー・マネージャは、システム上のすべての CPU リソースを使用して 1 つのタスクを処理できます。実際には、作業キュー・マネージャは、特定のタイプのタスクが使用できるコアの数を制限して、システム上のワークロードすべてが効率的に処理されるようにします。

1.2.3 オペレーティング・システムとの連携

大規模なトランザクション・データベース・アプリケーション向けの従来の ObjectScript コードでは、オペレーティング・システムはプログラム・タスクの切り替え (コンテキスト・スイッチとも呼ばれる) にかなりの数のリソースを消費する場合があります。作業キュー・マネージャでは各コア上でキューイング・メカニズムを使用するので、コンテキスト・スイッチの必要性は大幅に少なくなります。オペレーティング・システムでコンテキスト・スイッチが必要になるのは、作業キュー・マネージャで管理されているアクティブなジョブの数が利用可能なコアの数を上回る場合だけです。このように、キューイング作業によってパフォーマンスが全般的に向上します。

1.2.4 柔軟性

作業単位は、一連の引数を取り、"作業単位について" で説明する要件を満たすクラス・メソッドまたはサブルーチンです。これらの制約内で表すことができるロジックはすべて作業キュー・マネージャで処理することができ、驚異的な柔軟性を提供します。

1.2.5 高レベルの制御およびレポート

作業キュー・マネージャでは、システムの CPU リソースをどのように使用するかを高いレベルで制御できます。例えば、作業のカテゴリを作成して、そのカテゴリに割り当てられるワーカ・ジョブの数を定義できます。さらに、作業キュー・マネージャはワークロードのメトリックも備えており、システムの負荷をリアルタイムで監視できます。

2 作業単位について

作業キュー・マネージャは、作業単位 (作業項目とも呼ばれる) を処理することで機能します。作業単位とは、以下の要件を満たす ObjectScript クラス・メソッドまたはサブルーチンです。

- クラス・メソッドまたはサブルーチンを独立して処理できる。例えば、1 つの作業単位が別の作業単位からの出力に依存することはできません。作業単位は任意の順序で処理される可能性があるため、独立性が必要です。ただし、必要に応じて、コールバックを使用して作業を順番に実行することは可能です。詳細は、"コールバックの使用法"を参照してください。
- クラス・メソッドまたはサブルーチンは、サイズが約数千行の ObjectScript コードである。この要件により、フレームワークのオーバーヘッドが重要な要因にならないようにします。

  さらに、非常に大きい作業単位を少数 (例えば 4 つ) 使用するよりも、小さい作業単位を多数 (例えば 100 個) 使用することをお勧めします。作業をこのように分散することで、システムでより多くの CPU コアを使用できるようになったときにスケールアップできます。
- WaitForComplete() メソッドが %Status 値を返して全体的な成功または失敗を通知できるように、コードが成功または失敗を示す %Status 値を返す。または、作業単位から例外をスローし、この例外をトラップして %Status 値に変換し、マスタ・プロセスで返すことができます。
- コードが別の作業単位と同じグローバルを変更する場合、何らかのロック方法を使用して、あるワーカ・ジョブがグローバルを読み取っている間は、別のワーカがそのグローバルを変更できないようにする必要がある。
- コードに排他的な新規作成、強制終了、またはアンロックが含まれていない。これらはフレームワークに干渉するためです。
- コードにデータを格納するためのプロセス・プライベート・グローバルが含まれている場合、これらのプロセス・プライベート・グローバルがマスタ・プロセスや他のチャンクからアクセスされない。この要件は、複数のジョブが各チャンクを処理するために必要になります。
- クラス・メソッドまたはサブルーチンの一部として呼び出されるロジックがすべて正しくクリーンアップされ、変数、ロック、プロセス・プライベート・グローバル、またはその他のアーティファクトがパーティション内に残らない。同じプロセスを使用してまったく別の作業項目を後で処理するため、この要件は重要です。

作業キュー・マネージャを使用するには、プログラムによる作業をある程度、作業単位に分割する必要があります。

3 ワーカ・ジョブについて

ワーカ・ジョブは、作業キュー・マネージャの作業単位を完了するプロセスです。%SYSTEM.Process クラスを使用することで、ワーカ・ジョブを他のプロセスと同じように表示、管理、および監視できます。特定のプロセスがワーカ・ジョブであるかどうかを判断する必要がある場合は、そのプロセス内から $system.WorkMgr.IsWorkerJob() を呼び出します。つまり、%SYSTEM.WorkMgr クラスの IsWorkerJob() メソッドを呼び出すことができます。

作業キュー・マネージャは、コントローラ・プロセスを使用してワーカ・ジョブに指示を行います。コントローラ・プロセスは、以下のような複数の操作を実行する専用プロセスです。

- ワーカ・ジョブの起動
- ワーカ・ジョブの数の管理
- 停止したワーカ・ジョブの検出およびレポート
- ワークロード・メトリックの記録
- 非アクティブな作業キューの検出
- 作業キューの削除

ワーカ・ジョブは、以下の状態のいずれかになります。

- 作業キューへのアタッチの待機中。
- 作業単位の待機中。ワーカ・ジョブは、ジョブが解放されるまでの短時間のみ、この状態になる可能性があります。
- アクティブ。ワーカ・ジョブは、作業単位の実行中にプロセスを進めているときにのみアクティブになります。
- 作業単位の処理中にロックまたはイベントによりブロックされている。ブロックされているワーカ・ジョブはアクティブではありません。ワーカがブロック状態になり、作業キューに追加の作業がある場合、作業キュー・マネージャはリタイア済みのワーカを有効化するか、新しいワーカを起動できます。ワーカ・ジョブのブロックが解除されると、アクティブなワーカの数が、作業キューに対して指定されたアクティブなワーカの最大数を超過する場合があります。これが発生した場合、コントローラ・プロセスは、作業単位を完了させる次のワーカをリタイアさせます。その結果、ワーカ・ジョブのアクティブな数が、特定の作業キューに対して指定されたワーカ・ジョブの最大数を上回る期間が短時間存在する場合があります。
- リタイア済みで、すぐに有効化することが可能。

未使用のワーカのジョブは、短時間、他の作業キュー・マネージャのキューによって使用できる状態のまま残ります。タイムアウト期間は変更されることがあるため、意図的に文書化されていません。タイムアウト期間の有効期限が過ぎると、ワーカは削除されます。

ワーカ・ジョブが、削除またはクリアされたキューの作業項目をアクティブに処理している場合、システムはごく短時間待機した後、EXTERNAL INTERRUPT エラーを発行します。ワーカ・ジョブがエラーの後も処理を続行すると、システムは DeleteTimeout プロパティで指定された秒数待機してから、ワーカを強制終了し、作業単位を処理するための新しいワーカを起動します。

ワーカ・ジョブを開始するのはスーパーサーバです。つまり、ジョブはスーパーサーバ・プロセスによって使用されるオペレーティング・システム・ユーザの名前で実行されます。このユーザ名は、現在ログインしているオペレーティング・システム・ユーザとは異なる場合があります。

4 基本ワークフロー

以下の手順を実行して、作業キュー・マネージャを使用できます。

1. ObjectScript コードを作業単位に分割します。作業単位は、特定の要件を満たすクラス・メソッドまたはサブルーチンです。詳細は、"作業単位について" を参照してください。
2. 作業キューを作成します。これは %SYSTEM.WorkMgr クラスのインスタンスです。そのためには、%SYSTEM.WorkMgr クラスの %New() メソッドを呼び出します。このメソッドは作業キューを返します。

   使用する並列ワーカ・ジョブの数を指定することも、既定値を使用することもできます。既定値はマシンとオペレーティング・システムによって異なります。さらに、カテゴリを作成済みの場合は、ジョブの取得元のカテゴリを指定できます。

   作業キューを作成すると、作業キュー・マネージャは以下のアーティファクトを作成します。

- 作業キューが実行されるネームスペース名など、作業キューについての情報を含むグローバル
- 作業キューが処理する必要のある、シリアライズされた作業単位の場所とイベント・キュー
- 作業キューが作業単位の処理を終了したときに作成される完了イベントの場所とイベント・キュー

3. 作業単位（作業項目とも呼ばれる）を作業キューに追加します。このためには、Queue() メソッドまたは QueueCallback() メソッドを呼び出すことができます。引数として、クラス・メソッド（またはサブルーチン）の名前と、対応する引数があれば渡します。

   キューに追加された項目に対してすぐに処理が開始されます。

   キューで利用可能なワーカ・ジョブよりもキュー内の項目が多い場合、キューを空にするためにジョブの競合が発生します。例えば、100 個の項目と 4 つのジョブがある場合、各ジョブはキューの先頭から項目を取り出して処理してから、キューの先頭に戻り、別の項目を取り出して処理します。キューが空になるまで、このパターンが続きます。

   作業キュー・マネージャは、作業項目の実行時に、呼び出し元のセキュリティ・コンテキストを使用します。

   作業項目をキューに入れると、作業キュー・マネージャは以下のタスクを実行します。

   - 作業単位を構成する引数、セキュリティ・コンテキスト、およびクラス・メソッドまたはサブルーチンをシリアライズしてから、シリアライズされたデータを、作業キューに関連付けられた作業単位をリストするグローバルに挿入する
   - 作業キュー上のイベントを通知する
   - 作業単位を処理するために追加のワーカ・ジョブが必要な場合に、ワーカ・ジョブが利用可能であれば、ワーカ・ジョブが作業キューにアタッチされるようにし、利用可能なワーカ・ジョブの数を減らす

4. 作業が完了するまで待機します。そのために、作業キューの WaitForComplete() メソッドを呼び出すことができます。

   次に、作業キュー・マネージャは以下のタスクを実行します。

   - 完了イベントを待機する
   - ワークロード・メトリックなどの出力をターミナルに表示する
   - 作業単位に関連するエラーがあれば収集する
   - QueueCallback() メソッドを使用して作業単位を作業キューに追加した場合は、コールバック・コードを実行する

5. アプリケーションに応じた処理を続行します。

以下の例では、これらの基本的な手順を示しています。

ObjectScript

```
Set queue=##class(%SYSTEM.WorkMgr).%New()
For i = 1:1:filelist.Count() {
    Set sc=queue.Queue("..Load",filelist.GetAt(i))
    If $$$ISERR(sc) {
        Return sc
    }
}
Set sc=queue.WaitForComplete()
If $$$ISERR(sc) {
    Return sc
}
```

コードで作業キュー・マネージャを初期化してから、ファイルのリストを繰り返し処理します。各ファイルに対して、ファイルを読み込む作業キュー項目を追加します。すべての作業キュー項目を追加したら、作業が完了するまでコードで待機します。

注釈　%SYSTEM.WorkMgr クラスは、このドキュメントで後述するように、メソッドでより複雑なワークフローをサポートします。

4.1 基本的なメソッド

前のセクションで説明した手順を完了するには、%SYSTEM.WorkMgr クラスの以下の 3 つのメソッドを使用できます。

%New()

```
classmethod %New(qspec As %String = "", numberjobs As %Integer, category) as WorkMgr
```

作業キュー（並列処理の実行に使用できる %SYSTEM.WorkMgr クラスのインスタンス）を作成し、初期化して返します。このメソッドは、以下の引数を受け入れます。

qspec

この作業キュー内で実行されるコードを制御するコンパイラ・フラグと修飾子の文字列。"クラスの定義と使用" の "クラスの定義とコンパイル" の章にある "クラス・コンパイラのフラグおよび修飾子の表示" を参照してください。

numberjobs

この作業キュー内で使用する並列ワーカ・ジョブの最大数です。既定値は、マシンとオペレーティング・システムの特性によって異なります。

category

この作業キュー内で使用するワーカ・ジョブを供給するカテゴリの名前です。詳細は、"カテゴリの管理" を参照してください。

作成時にキューにワーカ・ジョブが割り当てられることはありません。ワーカ・ジョブは、作業単位を作業キューに追加した後にのみ割り当てられます。

Queue()

```
method Queue(work As %String, args... As %String) as %Status
```

作業単位を作業キューに追加します。このメソッドは、以下の引数を受け入れます。

work

実行するコードです。一般的に、このコードは、成功または失敗を示す %Status 値を返します。

コードが %Status 値を返す場合、以下の構文を使用できます。

- クラス・メソッドには ##class(Classname).ClassMethod を使用します。Classname はクラスの完全修飾名、ClassMethod はメソッドの名前です。

  メソッドが同じクラスにある場合、例に示すように、構文 ..ClassMethod を使用できます。

- サブルーチンには $$entry^rtn を使用します。entry はサブルーチンの名前、rtn はルーチンの名前です。

コードが %Status 値を返さない場合は、代わりに以下の構文を使用します。

- クラス・メソッドには =##class(Classname).ClassMethod（または、メソッドが同じクラス内にある場合は =..ClassMethod）を使用します。
- サブルーチンには entry^rtn を使用します。

作業単位の要件に関する情報については、"作業単位について" を参照してください。

args

クラス・メソッドまたはサブルーチンの引数のコンマ区切りリストです。多次元配列を引数として渡すには、通常どおりその引数の前にピリオドを付けて、参照によって渡されるようにします。

このフレームワークを最大限に活用するには、これらの引数で渡されるデータのサイズを比較的小さくする必要があります。大量の情報を渡すには、引数の代わりにグローバルを使用します。

作業単位をキューに入れると、作業キューの作成時に指定した numberjobs の値または既定値まで、一度に 1 つずつジョブが割り当てられます。また、呼び出し元のセキュリティ・コンテキストが記録され、各作業項目はそのセキュリティ・コンテキスト内で実行されます。

WaitForComplete()

```
method WaitForComplete(qspec As %String, errorlog As %String) as %Status
```

作業キューがすべての項目を完了するまで待機してから、成功または失敗を示す %Status 値を返します。%Status 値には、作業項目によって返されるすべての %Status 値の情報が含まれます。このメソッドは、以下の引数を受け入れます。

qspec

コンパイラ・フラグと修飾子の文字列です。"クラスの定義と使用" の "クラスの定義とコンパイル" の章にある "クラス・コンパイラのフラグおよび修飾子の表示" を参照してください。

errorlog

エラー情報の文字列で、出力として返されます。

4.2 作業キューのプロパティ

各作業キュー (または %SYSTEM.WorkMgr のインスタンス) には、以下のプロパティがあります。

NumWorkers

作業キューに割り当てられたワーカ・ジョブの数です。

NumActiveWorkers

現在アクティブなワーカの数です。

また、作業キューが属するカテゴリのプロパティによって、作業キューの動作が決定されます。

5 カテゴリの管理

カテゴリは、ワーカ・ジョブの独立したプールです。ワーカ・ジョブのセットを初期化する際に、ワーカを供給するカテゴリを指定できます。そのセット内のいずれかのワーカ・ジョブが、作業項目の実行中に追加のワーカ・ジョブを要求すると、新しいワーカ・ジョブは同じカテゴリから取得されます。

例えば、システムで提供されている SQL カテゴリに、最大 8 つのワーカを割り当てるとします。次に、ビジネス・インテリジェンス・ダッシュボードの構築に関連するプロセスのカテゴリを作成し、このカテゴリに最大 4 つのワーカを割り当てるとします。特定の時間に、SQL プールのすべてのワーカが使用されている場合でも、ビジネス・インテリジェンス・カテゴリのワーカは、作業項目を即座に処理することができます。

システムには、ユーザが削除することのできない SQL および Default の 2 つのカテゴリが含まれています。SQL カテゴリは、クエリの並列処理を含め、システムによって実行される SQL 処理に使用します。カテゴリを指定せずにワーカ・ジョブのセットを初期化した場合は、Default カテゴリからワーカ・ジョブが供給されます。

各カテゴリには、カテゴリ内の各作業キューの動作に影響を与えるプロパティがあります。以下のプロパティがあります。

DefaultWorkers

このカテゴリの作業キューを作成して、ワーカ・ジョブ数を指定しなかった場合、これがその作業キューのワーカ・ジョブ数になります。このプロパティの既定値はコア数です。

MaxActiveWorkers

このカテゴリの要求を処理するジョブのプールに保持されている、アクティブなワーカ・ジョブの最大数。最大アクティブ・ジョブ数をこの限度付近に保つため、アイドル・ジョブが検出され、新規ジョブが自動的に起動されます。既定値はコア数の 2 倍です。

MaxWorkers

このカテゴリの作業キューの最大ワーカ・ジョブ数です。作業キューの作成時に、これより多くのワーカ・ジョブ数を指定した場合、代わりにこの限度値が使用されます。既定値はコア数の 2 倍です。

注釈　DefaultWorkers、MaxActiveWorkers、MaxWorkers の既定値の使用の詳細は、作業キューに関する CPF リファレンス・ページを参照してください。これらの値を変更するには、"アクティブな CPF の編集" を参照してください。

カテゴリを作成するには、**[システム管理]→[構成]→[システム構成]→[WQM Categories]** の順に移動して、カテゴリのプロパティを調整し、カスタム・カテゴリを削除します。カスタム・カテゴリの名前は大文字と小文字が区別され、文字、数字、アンダースコア、ダッシュ、ピリオドが使用可能です。

Config.WorkQueues API を使用してカテゴリを操作することもできます。

6 コールバックの使用

コールバックとは、作業キュー・マネージャが作業項目の完了後に実行する必要があるコードです。以下の 2 つの理由で、コールバックを使用できます。

- 作業項目の完了に依存する作業を実行する
- 作業項目を非同期に完了することを選択している場合に、キューイングされた作業がすべて完了したことを通知する

6.1 作業項目に対するコールバックの組み込み

コールバックを追加するには、作業項目を作業キューに追加する際に、Queue() メソッドの代わりに QueueCallback() メソッドを呼び出します。

```
method QueueCallback(work As %String, callback As %String, args... As %String) as %Status
```

work メソッドと args メソッドは、Queue() メソッドのものと同じです。ただし、以下の構文を使用して、実行するコールバック・コードを callback 引数で指定します。

- クラス・メソッドの場合は `##class(Classname).ClassMethod`
- サブルーチンの場合は `$$entry^rtn`

クラス・メソッドまたはサブルーチンは、メインの作業項目と同じ引数を同じ順序で受け付ける必要があります。マスタ・プロセスは、メインの作業項目とコールバック・コードに同じ引数を渡します。

コールバック・コードでは、以下のパブリック変数にアクセスできます。

- %job。実際に作業を実行したプロセスのジョブ ID を格納します。
- %status。作業単位によって返される %Status 値を格納します。
- %workqueue。作業キュー・インスタンスの OREF です。

これらのパブリック変数はコールバック内では利用できますが、作業項目内では利用できません。

6.2 コールバックを使用した完了の判断

WaitForComplete() メソッドを使用して、作業キュー内のキューイングされた作業がすべて完了するまで待ってからマスタ・プロセスに戻るのではなく、以下のように、作業キュー・マネージャをポーリングして完了を判断することができます。

- Queue() メソッドの代わりに QueueCallback() メソッドを使用して、前のセクションで説明されているように、作業項目を作業キューに追加します。
- すべての作業項目の作業が完了したら、コールバック・コードでパブリック変数 %exit を 1 に設定します。
- WaitForComplete() メソッドの代わりに Wait() メソッドを使用します。

```
method Wait(qspec As %String, byRef AtEnd As %Boolean) as %Status
```

Wait() メソッドは、終了して呼び出し元に戻るためにコールバックからのシグナルを待機します。具体的には、コールバック・コードでパブリック変数 %exit を 1 に設定するまで待機します。Wait() は参照によって AtEnd を返します。AtEnd が 1 の場合、作業はすべて完了しています。または、AtEnd が 0 の場合、1 つ以上の作業項目が完了していません。

7 現在のデバイスへの出力の制御

既定では、作業項目が現在のデバイスに対して出力 (**WRITE** 文) を生成する場合、作業キューは、WaitForComplete() または Wait() の終了まで、出力をバッファに保存します。それよりも早く出力を生成したい場合は、例えば以下のように、作業項目で %SYSTEM.Context.WorkMgr の Flush() クラス・メソッドを呼び出します。

```
set sc = $system.Context.WorkMgr().Flush()
```

作業項目がこのメソッドを呼び出すと、親作業キューは、その作業項目に対して保存されている出力をすべて書き込みます。

また、-d フラグを使用して現在のデバイスへの出力をすべて抑制できます。この場合、出力がないため、Flush() メソッドは何も行いません。

8 作業キューの一時停止と再開

%SYSTEM.WorkMgr クラスは、作業キュー内の作業を一時停止および再開するために使用できるメソッドを備えています。

Pause()

```
method Pause(timeout As %Integer, ByRef completed As %Boolean = 0) as %Status
```

この作業キューに関連付けられたワーカ・ジョブが、この作業キューから追加の項目を受け入れないようにします。Pause() メソッドは、進行中の作業項目も停止します。

timeout 引数は、進行中の作業項目を停止する前に、メソッドが待機する時間 (秒単位) を表します。タイムアウト期間が過ぎると、このメソッドは completed の値を返します。この値は、Pause() メソッドを呼び出したときに進行中であった作業項目が完了したかどうかを示します。したがって、0 の timeout 値を渡すと、ワーカ・ジョブが作業キュー内のすべての作業項目を完了したかどうかがすぐにわかります。

Resume()

```
method Resume() as %Status
```

この作業キュー内の作業が Pause() メソッドを使用して一時停止されている場合に作業を再開します。具体的には、このメソッドは、作業キュー内に追加の項目がある場合、作業キュー・プロセスがそれらを受け付けて開始できるようにします。

作業を完全に停止する方法の詳細は、"作業キューの停止と作業項目の削除" を参照してください。

9 作業キューのデタッチとアタッチ

通常は、ワーカ・ジョブのセットを初期化して作業項目をキューイングしたら、ワーカ・ジョブが作業項目を完了するまで待機します。しかし、ワーカ・ジョブが作業項目を完了するのに予想より長くかかっていたり、単一のプロセスを待機専用にできないといった状況が発生する場合があります。そのため、作業キュー・マネージャでは、プロセスから作業キューをデタッチし、その後作業キューを同じプロセスまたは異なるプロセスにアタッチすることが可能です。

例えば、queue が、ユーザが初期化した作業キューを参照しているとします。さらに、作業項目をいくつか作業キューに追加したと想定します。Wait() または WaitForComplete() を呼び出して処理中の作業のステータスを判別する前に、以下のメソッドを使用できます。

Detach()

```
method Detach(ByRef token As %String, timeout As %Integer=86400) as Status
```

作業キューを初期化したときに作成したオブジェクト参照から作業キュー・オブジェクトをデタッチします。Detach() メソッドは、進行中の作業を続行できるようにし、作業キューの現在の状態を保持します。

token 引数はセキュア・トークンを表します。セキュア・トークンを使用して、後で作業キューを別のプロセスにアタッチできます。timeout 引数はオプションです。デタッチされた作業キュー・オブジェクトをシステムが保持する時間 (秒単位) を指定します。タイムアウト時間が経過すると、作業キューに関連付けられたワーカ・ジョブと情報は削除されます。timeout の既定値は 1 日です。

Detach() メソッドを呼び出した後は、デタッチされたオブジェクト参照に対するほとんどの呼び出しはエラーを返します。ただし、NumActiveWorkers() メソッドおよび NumWorkers() メソッドは -1 を返します。

Attach()

```
Attach(token, ByRef sc As %Status) as WorkMgr
```

新しいオブジェクト参照を、以前にデタッチした作業キュー・オブジェクトにアタッチします (その作業キュー・オブジェクトがまだメモリ内にある場合)。Attach() メソッドは、作業キューに関連付けられた作業キュー・マネージャの新しいインスタンスを返します。その後、作業キュー上でメソッドを呼び出せます。例えば、timeout の値を 0 に設定して Wait() メソッドを呼び出して、キューがデタッチされる前に作業項目を完了したかどうかを判別できます。

token 引数は、以前に作業キュー上で呼び出した Detach() メソッドによって返されるセキュア・トークンを表します。

例えば、以下のように作業キューをデタッチしてから、再度アタッチできます。

```
Set sc=queue.Detach(.token,60)
If $$$ISERR(sc) {
        Return sc
    }
Set queue=$system.WorkMgr.Attach(token,.sc)
If $$$ISERR(sc) {
        Return sc
    }
```

10 作業キューの停止と作業項目の削除

作業キューを停止し、進行中の作業項目を中断して、キューに入っている作業項目を削除できます。そのためには、作業キューの Clear() メソッドを呼び出します。

```
method Clear(timeout As %Integer = 5) as %Status
```

タイムアウト期間 timeout (秒) を指定すると、このメソッドはワーカ・ジョブが現在のタスクを完了するまで待機してから、ジョブを強制終了します。システムによって作業キューが削除され、作業項目がアタッチされていない状態で再作成されます。その後、システムはすぐに Wait() または WaitForComplete() から戻ります。

11 セットアップおよびティアダウン処理の指定

通常、各作業キューには複数のワーカ・ジョブがあります。ワーカ・ジョブよりも多くの作業項目がある場合、ワーカ・ジョブは複数の作業項目を一度に 1 つずつ実行します。これらの作業項目が開始する前に必要なセットアップ手順を特定し、作業項目をキューに追加する前に該当ロジックをすべて呼び出すと、便利です。

%SYSTEM.WorkMgr クラスは Setup() メソッドと TearDown() メソッドを備えており、これらを使用して、ワーカ・ジョブのセットアップ・アクティビティとクリーンアップ・アクティビティを定義できます。例えば、Setup() を使って、ワーカ・ジョブ内で使用するためにパブリック変数を設定し、TearDown() を使用してこれらの変数を削除します。また、Setup() を使用してロックを取得したり、プロセス・プライベート・グローバルを設定したりできます。これらのロックを解放したりグローバルを削除したりするには、TearDown() を使用します。

どちらの場合も、Queue() または QueueCallback() を呼び出す前に、Setup()、TearDown()、またはその両方を呼び出す必要があります。Setup() メソッドと TearDown() メソッドは、作業キュー・マネージャによってのみ使用される内部グローバルに情報を保存します。いずれかのワーカ・ジョブがこのキューの最初の作業項目を開始すると、そのワーカ・ジョブは、最初に作業マネージャ・キューのグローバルをチェックして、セットアップ・ロジックがあるかどうかを確認します。ロジックがある場合、ワーカ・ジョブはそのロジックを実行してから作業項目を開始します。ワーカ・ジョブはそれ以上セットアップ・ロジックを実行しません。同様に、いずれかのワーカ・ジョブがキューの最後の作業項目を完了したら、そのワーカ・ジョブはティアダウン・ロジックがあるかどうかを確認します。ロジックがある場合、ワーカ・ジョブはそのロジックを実行します。

以下にこれらのメソッドの詳細を説明します。

Setup()

```
method Setup(work As %String, args... As %String) as %Status
```

キューの最初の項目を処理する前に呼び出すワーカ・プロセスのコードを指定します。このメソッドを使用する場合は、Queue() メソッドまたは QueueCallback メソッドを呼び出す前に呼び出す必要があります。Setup() は以下の引数を受け入れます。

work

実行するセットアップ・コードです。この引数に対してサポートされる構文は、Queue() メソッドの work 引数に対してサポートされている構文と同じです。この構文については、前のセクションで説明しています。

args

このコードの引数のコンマ区切りリストです。多次元配列を引数として渡すには、その引数の前にピリオドを付けて、参照によって渡されるようにします。

これらの引数で渡すデータのサイズは比較的小さく抑える必要があります。大量の情報を提供するには、引数を渡す代わりにグローバルを使用できます。

TearDown()

```
method TearDown(work As %String, args... As %String) as %Status
```

キューの最後の項目を処理した後にプロセスを前の状態に戻すために呼び出すワーカ・プロセスのコードを指定します。このメソッドを使用する場合は Queue() メソッドまたは QueueCallback メソッドを呼び出す前に呼び出す必要があります。

TearDown() は、Setup() メソッドと同じ引数を受け入れます。ただし、work 引数で、実行するティアダウン・コードを指定します。