ロックと並行処理の制御

Version 2023.1
2024-01-02

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

ロックと並行処理の制御
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

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

目次

ロックと並行処理の制御

マルチプロセス・システムでの重要な機能の 1 つに並行処理の制御があります。これは、異なるプロセスがデータの特定の要素を同時に変更し、破損に至ることを防止する機能です。そのために、InterSystems IRIS にはロック管理システムが用意されています。このページでは、概要について説明します。

InterSystems SQL では、ロックを操作するコマンドも提供されています。詳細は、"InterSystems SQL リファレンス" を参照してください。

また、**%Persistent** クラスでは、オブジェクトへの同時アクセスを制御する手段も提供されています。すなわち、このクラスの %OpenId() およびその他のメソッドの同時処理引数です。これらのメソッドは、最終的には ObjectScript の LOCK コマンドを使用します。これについては、このページで後述します。すべての永続オブジェクトは、それらのメソッドを継承します。"クラスの定義と使用" の "オブジェクト同時処理" を参照してください。同様に、INSERT、UPDATE、および DELETE の各操作のロックが自動的に実行されます (%NOLOCK キーワードを指定していない場合)。

このクラス **%Persistent** では、%GetLock()、%ReleaseLock()、%LockId()、%UnlockId()、%LockExtent()、および %UnlockExtent() メソッドも提供されています。詳細は、**%Persistent** のクラスリファレンスを参照してください。

1 概要

基本のロック・メカニズムは、LOCK コマンドです。このコマンドの目的は、あるプロセス内のアクティビティを、別のプロセスが処理継続 OK の合図を送るまで遅延することです。

InterSystems IRIS® では、ロックはそれ自体ではアクティビティを防止しません。規約では、ロックが機能するには、互いに競合するプロセスがすべて同じロック名でロックを実装していることが要求されています。例えば、以下に一般的なシナリオについて説明します。

1. プロセス A が LOCK コマンドを発行し、InterSystems IRIS がロックを作成します (既定では、排他ロック)。

   通常、プロセス A はグローバルのノードを変更します。詳細はアプリケーション固有です。

2. プロセス B が同じロック名で LOCK コマンドを発行します。排他ロックが存在するため、プロセス B は一時停止します。具体的には、この LOCK コマンドは復帰しなくなり、コードの後続行が実行できなくなります。

3. プロセス A がロックを解放すると、プロセス B の LOCK コマンドが復帰してプロセス B を継続します。

   通常、プロセス B は同じグローバルのノードを変更します。

1.1 ロック名

LOCK コマンドの引数の 1 つにロック名があります。ロック名は、任意の名前ですが、共通の規則として、プログラマはロックする項目の名前と同じロック名を使用します。通常、ロックする項目は、グローバルまたはグローバルのノードになります。したがって、通常ロック名は、グローバル名の名前またはグローバルのノードの名前のようになります。(このページでは、最も一般的であるキャレットで始まるロック名のみについて説明します。キャレットで始まらない名前のロックの詳細は、"ObjectScript リファレンス" の "LOCK" を参照してください。)

形式的には、ロック名は、"ObjectScript の使用法" の "変数" の章で説明したローカル変数およびグローバル変数と同じ名前付け規約に従います。変数と同様に、ロック名は大文字と小文字を区別し、添え字を付けることができます。プロセス・プライベート・グローバル名をロック名として使用しないでください (定義によってそのようなグローバルにアクセスできるプロセスは 1 つしかないため、そのようなロックは必要ありません)。

Tipヒント 規約によってロックが機能し、ロック名が任意であるため、同じ名前のロックを作成する前に任意の変数を定義する必要はありません。

InterSystems IRIS によるメモリの割り当ておよび管理の方法により、ロック名の形式はパフォーマンスに影響します。ロックは、添え字を使用するロック名に対して最適化されます。例えば、`^sample.person(id)` です。

一方、InterSystems IRIS は、`^name_concatenated_identifier` のようなロック名に対しては最適化されません。添え字のないロック名を使用すると、ECP に関するパフォーマンスの問題の原因になる場合もあります。

1.2 ロック・テーブル

InterSystems IRIS は、現在のすべてのロックと、そのロックを所有しているプロセスを記録するために、システム規模のテーブルをメモリ内に保持しています。このテーブル (ロック・テーブル) は、管理ポータルからアクセスできます。管理ポータルでは、ロックの表示と、まれに必要になるロックの削除を実行できます。特定のプロセスが、ロック名が異なる複数のロックを所有している可能性がある点に注意してください (複数のロックが同じ名前の場合もあります)。

プロセスが終了すると、InterSystems IRIS は、そのプロセスが所有するすべてのロックを自動的に解放します。そのため、通常は、管理ポータルからロックを削除する必要はありません。ただし、アプリケーション・エラーが発生した場合を除きます。

ロック・テーブルは、固定サイズを超過することはありません (このサイズは locksiz 設定を使用して指定できます)。詳細は、″監視ガイド″ の “ロックの監視” を参照してください。そのため、ロック・テーブルが一杯になり、それ以上のロックが不可能になることがあります。この場合、InterSystems IRIS は、**messages.log** ファイルに以下のエラー・メッセージを書き込みます。

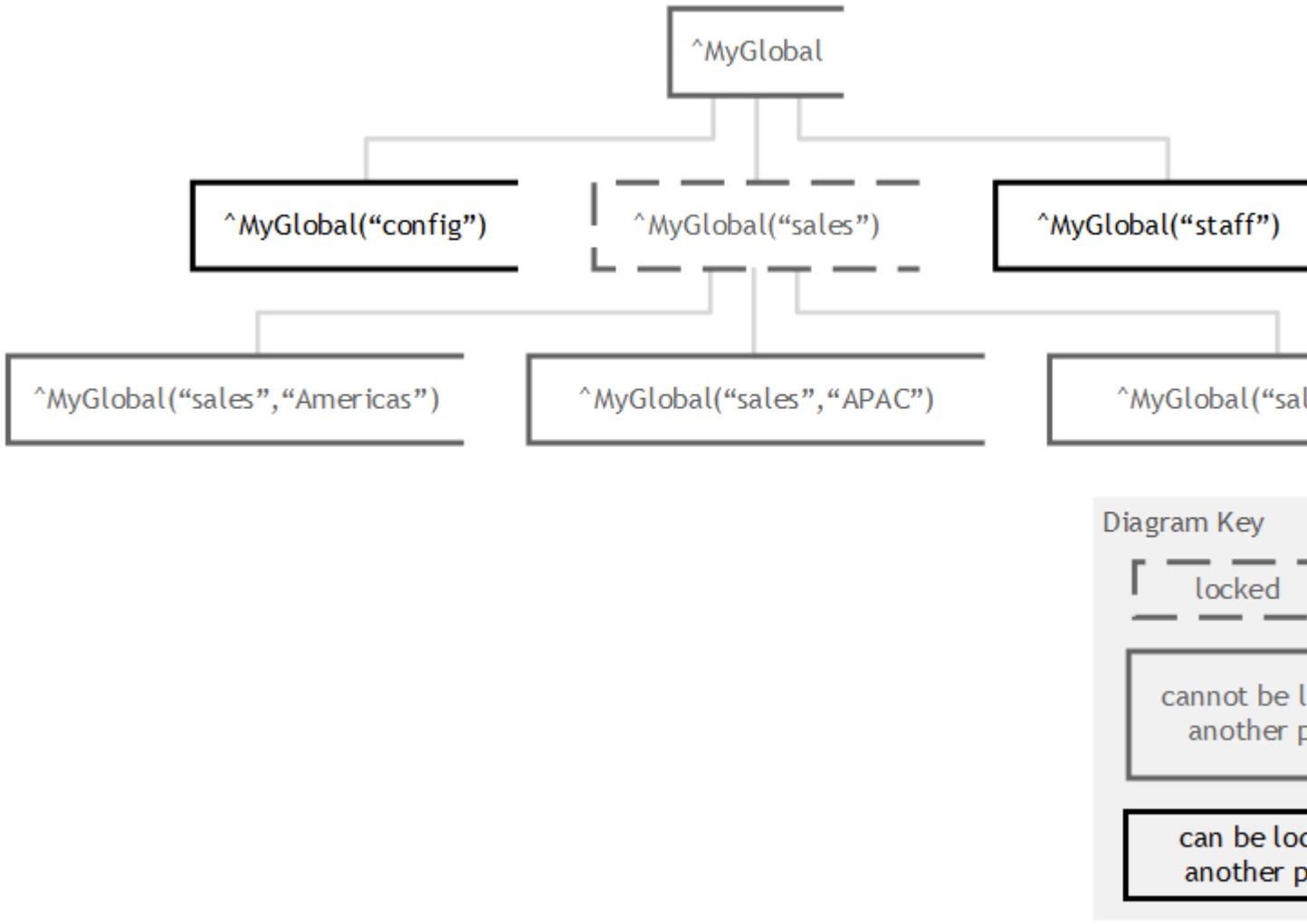
```
LOCK TABLE FULL
```

通常は、ロック・テーブルが一杯になっても、アプリケーション・エラーとは見なされません。InterSystems IRIS には、ロック・キューも用意されているため、プロセスはロック・テーブルにロックを追加する領域ができるまで待機します。(ただし、デッドロックは、アプリケーションのプログラミング・エラーです。このページで後述の “デッドロックの回避” を参照してください。)

2 ロックと配列

配列をロックする場合は、配列全体をロックすることも、配列内の 1 つ以上のノードをロックすることもできます。配列ノードをロックすると、他のプロセスは、そのノードに従属するどのノードもロックできなくなります。また、他のプロセスは、ロックされたノードの直接の祖先もロックできなくなります。

次の図に例を示します。



暗黙的なロックは、ロック・テーブルに含まれていないため、ロック・テーブルのサイズに影響しません。

InterSystems IRIS のロック・キュー・アルゴリズムでは、同じロック名のロックは、リソースが直接的に競合しない場合でも受信した順にキューに置かれます。詳細と使用例は、″ObjectScript の使用法″ の “ロック管理” の章にある “配列ノードに対するロックのキュー” を参照してください。

3 LOCK コマンドの使用法

ここでは、LOCK コマンドを使用してロックを追加および削除する方法について説明します。

3.1 増分ロックの追加

ロックを追加するには、以下のように LOCK コマンドを使用します。

```
LOCK +lockname
```

lockname はリテラルのロック名です。プラス記号 (+) によって増分ロックが作成されます。これは一般的なシナリオです。一般的ではない代替手段は、“単純ロックの作成” を参照してください。

このコマンドは、以下を実行します。

1. 指定のロックをロック・テーブルに追加しようとします。つまり、このエントリはロック・キューに追加されます。
2. ロックを取得できるまで実行を一時停止します。

異なる動作をするさまざまなロック・タイプがあります。既定でないロック・タイプのロックを追加するには、以下のバリエーションを使用します。

```
LOCK +lockname#locktype
```

locktype は、二重引用符で囲んだロック・タイプ・コードの文字列です。後述の“ロック・タイプ”を参照してください。

特定のプロセスが同じ名前の複数の増分ロックを追加できる点に注意してください。これらのロックは異なるタイプにもすべて同じタイプにもできます。

3.2 タイムアウト付き増分ロックの追加

正しくない方法で増分ロックを使用すると、デッドロックと呼ばれる好ましくない状態になる可能性があります。デッドロックについては、後述する“デッドロックの回避”で説明します。デッドロックを回避する 1 つの方法は、ロック作成時にタイムアウト時間を指定することです。これを実行するには、以下のように LOCK コマンドを使用します。

```
LOCK +lockname#locktype :timeout
```

timeout は、タイムアウト時間の秒数です。コロンの前のスペースはオプションです。timeout に 0 を指定すると、InterSystems IRIS はロックの追加を 1 回試行します (ただし、以下の注意を参照してください)。

このコマンドは、以下を実行します。

1. 指定のロックをロック・テーブルに追加しようとします。つまり、このエントリはロック・キューに追加されます。
2. ロックを取得できるまでまたはタイムアウト時間が経過するまで、実行を一時停止します。
3. $TEST 特殊変数の値を設定します。ロックが取得されると、InterSystems IRIS は $TEST を 1 に設定します。それ以外の場合、InterSystems IRIS は $TEST を 0 に設定します。

つまり、タイムアウト引数を使用している場合、コードで $TEST 特殊変数の値を調べ、その値を使用して続行するかどうかを選択する必要があります。以下に例を示します。

ObjectScript

```
Lock +^ROUTINE(routinename):0
If '$TEST {  Return $$$ERROR("Cannot lock the routine: ",routinename)}
```

3.2.1 ゼロ・タイムアウトに関する注意

前述のように、timeout に 0 を指定すると、InterSystems IRIS はロックの追加を 1 回試行します。ただし、ゼロのタイムアウトを使用して親ノードをロックしようとしたときに、既に子ノードをロックしている場合、ゼロのタイムアウトは無視され、代わりに内部の 1 秒のタイムアウトが使用されます。

3.3 ロックの削除

既定のタイプのロックを削除するには、以下のように LOCK コマンドを使用します。

```
LOCK -lockname
```

このコマンドを実行するプロセスが特定の名前の (既定のタイプの) ロックを所有している場合、このコマンドによってそのロックが削除されます。または、そのプロセスが複数の (既定のタイプの) ロックを所有している場合、このコマンドによってそれらのロックの 1 つが削除されます。

別のタイプのロックを削除するには、以下のようにします。

```
LOCK -lockname#locktype
```

locktype は、ロック・タイプ・コードの文字列です。後述の“ロック・タイプ”を参照してください。ロック・タイプ・コードは、ロック作成時と同じ順序でなくてもかまいません。

3.4 LOCK コマンドの他の基本的なバリエーション

完全を期すために、ここでは LOCK コマンドの他の基本的なバリエーションについて説明します。LOCK コマンドを使用した単純ロックの作成と LOCK コマンドを使用したすべてのロックの削除です。これらのバリエーションは実際には一般的ではありません。

3.4.1 単純ロックの作成

LOCK コマンドでは、+ 演算子を省略すると、LOCK コマンドは、まずこのプロセスが保持している既存のロックをすべて削除し、新しいロックを追加しようとします。この場合のロックは、増分ロックではなく単純ロックと呼ばれます。プロセスが以下のような構文で複数の単純ロックをすべて同時に作成する場合、プロセスで複数の単純ロックを所有することが可能です。

```
 LOCK (^MyVar1,^MyVar2,^MyVar3)
```

複数のロックを保持し、コード内の異なるステップでそれらを取得する必要があることが多いため、単純ロックは実際には一般的ではありません。したがって、増分ロックを使用するほうが実用的です。

ただし、単純ロックが適切な場合、単純ロックの作成時に locktype 引数と timeout 引数を指定できます。また、単純ロックを削除するには、マイナス記号 (-) を付けて LOCK コマンドを使用できます。

3.4.2 すべてのロックの削除

現在のプロセスが保持しているすべてのロックを削除するには、引数を付けずに LOCK コマンドを使用します。実際には、この方法でこのコマンドを使用することは、以下の 2 つの理由により一般的ではありません。

- できるだけ早く特定のロックを解放することが最良である。
- プロセスが終了すると、すべてのロックが自動的に解放される。

4 ロック・タイプ

locktype 引数は、追加または削除するロックのタイプを指定します。ロックを追加する場合、この引数を以下のように使用します。

```
LOCK +lockname#locktype
```

ロックを削除する場合には以下のようにします。

```
LOCK -lockname#locktype
```

どちらの場合でも、locktype は、二重引用符で囲まれた (任意の順序の) 1 つ以上のロック・タイプ・コードです。locktype 引数を指定する場合、シャープ文字 (#) を使用してロック名とロック・タイプを区切る必要があります。

以下のように 4 つのロック・タイプ・コードがあります。これらは大文字と小文字を区別しません。

- S — 共有ロックを追加します。“排他ロックと共有ロック”を参照してください。

- E ー エスカレート・ロックを追加します。“非エスカレート・ロックとエスカレート・ロック”を参照してください。
- I ー 即時アンロック・タイプのロックを追加します。
- D ー 遅延アンロック・タイプのロックを追加します。

  ロック・タイプ・コード D と I はトランザクションにおいて独特の動作をします。詳細は、″ObjectScript リファレンス″の ″LOCK″を参照してください。同じロック名に対してこれら 2 つのロック・タイプ・コードを同時に使用することはできません。

以下、数セクションにわたって、最も一般的なバリエーションについて説明し、最後のサブセクションですべてのロック・タイプについてまとめます。

4.1 排他ロックと共有ロック

あらゆるロックは、排他 (既定) または共有のどちらかになります。これらのタイプは、次に示す重要な意味を持ちます。

- あるプロセスが特定のロック名の付いた排他ロックを所有しているときに、その他のプロセスは、そのロック名の付いたロックを取得できません。
- あるプロセスが特定のロック名の付いた共有ロックを所有しているときに、その他のプロセスは、そのロック名の付いた共有ロックを取得できます。ただし、その他のプロセスはそのロック名の付いた排他ロックを取得できません。

排他ロックの一般的な目的は、値を変更する予定があることを示し、その他のプロセスがその値の読み取りまたは変更を試行してはいけないことを示すことです。共有ロックの一般的な目的は、値を読み取る予定があることを示し、その他のプロセスがその値の変更を試行してはいけないことを示すことです。ただし、その他のプロセスがその値を読み取ることは可能です。後述の“ロックの実際の使用方法”も参照してください。

4.2 非エスカレート・ロックとエスカレート・ロック

さらに、あらゆるロックは、非エスカレート (既定) またはエスカレートのどちらかになります。エスカレート・ロックの目的は、メモリを消費し、ロック・テーブルが埋め尽くされる可能性が大きくなる大量のロックの管理を簡単にすることです。

エスカレート・ロックは、同じ配列に含まれる複数のノードをロックする場合に使用します。エスカレート・ロックの場合、あるプロセスがある配列の並列ノードに特定の数 (既定では 1000) よりも多くのロックを作成すると、InterSystems IRIS は、個別のロック名を置き換えて、それらを新しい 1 つのロック (ロック・カウントを含むロック) に置き換えます (一方、InterSystems IRIS は非エスカレート・ロックに対してはこの動作を行いません)。詳細と使用例は、後述の“エスカレート・ロック”を参照してください。

注釈　添え字を含むロック名に対してのみエスカレート・ロックを作成できます。添え字のないロック名でエスカレート・ロックを作成しようとすると、InterSystems IRIS は <COMMAND> エラーを発行します。

4.3 ロック・タイプのまとめ

以下の表に可能なすべてのロック・タイプとその説明を示します。

	排他ロック	共有ロック (#"S" ロック)
非エスカレート・ロック	• locktype 省略 — 既定のロック・タイプ • #"I" — 即時アンロック・タイプの排他ロック • #"D" — 遅延アンロック・タイプの排他ロック	• #"S" — 共有ロック • #"SI" — 即時アンロック・タイプの共有ロック • #"SD" — 遅延アンロック・タイプの共有ロック
エスカレート・ロック (#"E" ロック)	• #"E" — 排他エスカレート・ロック • #"EI" — 即時アンロック・タイプの排他エスカレート・ロック • #"ED" — 遅延アンロック・タイプの排他エスカレート・ロック	• #"SE" — 共有エスカレート・ロック • #"SEI" — 即時アンロック・タイプの共有エスカレート・ロック • #"SED" — 遅延アンロック・タイプの共有エスカレート・ロック

複数のロック・コードを使用するロック・タイプの場合、ロック・コードの順序は任意にできます。例えば、ロック・タイプ #"SI" と #"IS" は同じです。

即時アンロックと遅延アンロックの詳細は、"ObjectScript リファレンス" の "LOCK" を参照してください。同じロック名に対してこれら 2 つのロック・タイプ・コードを同時に使用することはできません。

5 エスカレート・ロック

大量のロックを管理するためにエスカレート・ロックを使用します。これは配列のノードをロックする場合に関係します。特に同じ添え字レベルで複数のノードをロックする場合です。

あるプロセスが同じ配列の特定の添え字レベルで特定の数 (既定では 1000) よりも多くのエスカレート・ロックを作成すると、InterSystems IRIS は、個別のロック名をすべて削除し、それらを新しいロックに置き換えます。この新しいロックは親レベルになります。つまり、この配列の分岐全体が暗黙的にロックされます。(次に示す) 例でこのことを示します。

アプリケーションは、適切なタイミングが来たらすぐに特定の子ノードのロックを解放する必要があります (非エスカレート・ロックと同様)。ロックを解放すると、InterSystems IRIS は対応するロック・カウントをデクリメントします。アプリケーションが十分なロックを削除すると、InterSystems IRIS は親ノードのロックを削除します。2 番目のサブセクションに例を示します。

ロックしきい値 (既定値は 1000) の指定の詳細は、"構成パラメータ・ファイル・リファレンス" の “LockThreshold” を参照してください。

5.1 ロックのエスカレーションの例

^MyGlobal("sales",EU",salesdate) という形式で salesdate が日付を表しているロックが 1000 個あるとします。ロック・テーブルは以下のようになります。

Owner	ModeCount	Reference	Directory
1284	Exclusive	^%SYS("CSP","Daemon")	c:\intersystems\iris\mgr\
26324	Exclusive	^ISC.LMFMON("License Monitor")	c:\intersystems\iris\mgr\
23400	Exclusive	^ISC.Monitor.System	c:\intersystems\iris\mgr\
23180	Exclusive	^TASKMGR	c:\intersystems\iris\mgr\
23948	Exclusive	^%cspSession("vgMJ4iLMCL")	c:\intersystems\iris\mgr\irislocaldata\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-03")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-04")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-05")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-06")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-07")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-08")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-09")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-10")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-11")	c:\intersystems\iris\mgr\user\
19776	Exclusive_e	^MyGlobal("sales","EU","2015-07-12")	c:\intersystems\iris\mgr\user\

所有者 19776 のエントリに注目してください (これはロックを所有するプロセスです)。**[モードカウント]** 列は、これらが共有のエスカレート・ロックであることを示しています。

同じプロセスが同じ形式の別のロックを作成しようとすると、InterSystems IRIS はこれらをエスカレートします。これらのロックを削除し、`^MyGlobal("sales","EU")` という名前の単一のロックで置き換えます。ロック・テーブルは以下のようになります。

Owner	ModeCount	Reference	Directory
1284	Exclusive	^%SYS("CSP","Daemon")	c:\intersystems\iris\mgr\
26324	Exclusive	^ISC.LMFMON("License Monitor")	c:\intersystems\iris\mgr\
23400	Exclusive	^ISC.Monitor.System	c:\intersystems\iris\mgr\
23180	Exclusive	^TASKMGR	c:\intersystems\iris\mgr\
23948	Exclusive	^%cspSession("vgMJ4iLMCL")	c:\intersystems\iris\mgr\irislocaldata\
19776	Exclusive/1001E	^MyGlobal("sales","EU")	c:\intersystems\iris\mgr\user\

[モードカウント] 列は、これが共有のエスカレート・ロックで、そのカウントが 1001 であることを示しています。

以下の重要な点に注意してください。

- `^MyGlobal("sales","EU")` のすべての子ノードは、この時点で配列ロックの基本的な規則に従って暗黙的にロックされます。
- ロック・テーブルには、`^MyGlobal("sales","EU")` のどの子ノードが具体的にロックされたかに関する情報が含まれなくなります。これは、ロックを削除する際に重要な意味を持ちます。次のサブセクションを参照してください。

同じプロセスが `^MyGlobal("sales","EU",salesdate)` の形式のロック名をさらに追加すると、ロック・テーブルは、ロック名 `^MyGlobal("sales","EU")` のロック・カウントをインクリメントします。ロック・テーブルは以下のようになります。

Owner	ModeCount	Reference	Directory
1284	Exclusive	^%SYS("CSP","Daemon")	c:\intersystems\iris\mgr\
26324	Exclusive	^ISC.LMFMON("License Monitor")	c:\intersystems\iris\mgr\
23400	Exclusive	^ISC.Monitor.System	c:\intersystems\iris\mgr\
23180	Exclusive	^TASKMGR	c:\intersystems\iris\mgr\
23948	Exclusive	^%cspSession("vgMJ4iLMCL")	c:\intersystems\iris\mgr\irislocaldata\
19776	Exclusive/1026E	^MyGlobal("sales","EU")	c:\intersystems\iris\mgr\user\

[モードカウント] 列は、このロックのロック・カウントが 1026 になったことを示しています。

5.2 エスカレート・ロックの削除

非エスカレート・ロックと正確に同じ方法で、アプリケーションは、特定の子ノードのロックをできるだけ早く解放する必要があります。これを行うと、InterSystems IRIS は、エスカレートされたロックのロック・カウントをデクリメントします。例えば、コードが `^MyGlobal("sales","EU",salesdate)` のロックを削除したとします (salesdate は 2011 年の任意の日付に対応し、したがって 365 個のロックが削除されます)。ロック・テーブルは以下のようになります。

Owner	ModeCount	Reference	Directory
1284	Exclusive	^%SYS("CSP","Daemon")	c:\intersystems\iris\mgr\
26324	Exclusive	^ISC.LMFMON("License Monitor")	c:\intersystems\iris\mgr\
23400	Exclusive	^ISC.Monitor.System	c:\intersystems\iris\mgr\
23180	Exclusive	^TASKMGR	c:\intersystems\iris\mgr\
23948	Exclusive	^%cspSession("vgMJ4iLMCL")	c:\intersystems\iris\mgr\irislocaldata\
19776	Exclusive/660E	^MyGlobal("sales","EU")	c:\intersystems\iris\mgr\user\

ロックの数がしきい値 (1000) 未満になったにもかかわらず、ロック・テーブルには `^MyGlobal("sales","EU",salesdate)` のロックの個別のエントリが含まれていないことがわかります。

ノード `^MyGlobal("sales")` は、プロセスが `^MyGlobal("sales","EU",salesdate)` の形式のロックをさらに 661 個削除するまで明示的にロックされたままです。

重要

前記の説明に関して考慮する必要がある点があります。そもそもロックされていない配列ノードのロックをアプリケーションが 解放し、エスカレートされたロックのロック・カウントが不正確になり、エスカレートされたロックの解放が適切になる前にこれが“解放”されてしまう可能性があります。

例えば、2010 年から現在までの `^MyGlobal("sales","EU",salesdate)` のノードをプロセスがロックしたとします。1000 個を超えるロックが作成され、計画どおりにこのロックはエスカレートされます。アプリケーションのバグによって 1970 年のノードのロックが削除されたとします。InterSystems IRIS は、これらのノードが以前ロックされなかったにもかかわらずこの操作を許可します。InterSystems IRIS はロック・カウントを 365 デクリメントします。結果として得られるロック・カウントは、目的のロックのカウントとして正確ではなくなります。その後アプリケーションが別の年のロックを削除すると、エスカレートされたロックは、予期しない早さで削除される可能性があります。

6 ロック、グローバル、およびネームスペース

ロックは通常、グローバルへのアクセスを制御するために使用されます。グローバルは複数のネームスペースからアクセスできるため、InterSystems IRIS は、グローバルのロック・メカニズム用に、自動クロスネームスペース・サポートを提供しています。この動作は自動で、操作する必要はありませんが、参考のためにここで説明します。以下のような考慮すべきシナリオがいくつかあります。

- 永続クラスと追加グローバルのデータを格納する既定データベースがどのネームスペースにもあります。これが、そのネームスペースのグローバル・データベースです。ユーザがデータに (任意の方法で) アクセスする場合、他の考慮事項がない限り、InterSystems IRIS によってそのデータベースからそれが取得されます。特定のデータベースを複数のネームスペースのグローバル・データベースにすることも可能です。シナリオ 1 を参照してください。
- ネームスペースには、他のデータベースに格納されているグローバルへのアクセスを提供するマッピングを含めることができます。シナリオ 2 を参照してください。
- ネームスペースには、他のデータベースに一部格納されているグローバルへのアクセスを提供する添え字レベルのグローバル・マッピングを含めることができます。シナリオ 3 を参照してください。

- 1 つのネームスペースで実行されているコードは、拡張参照を使用して、拡張参照を使用しない場合にはこのネームスペースでは使用できないグローバルにアクセスできます。シナリオ 4 を参照してください。

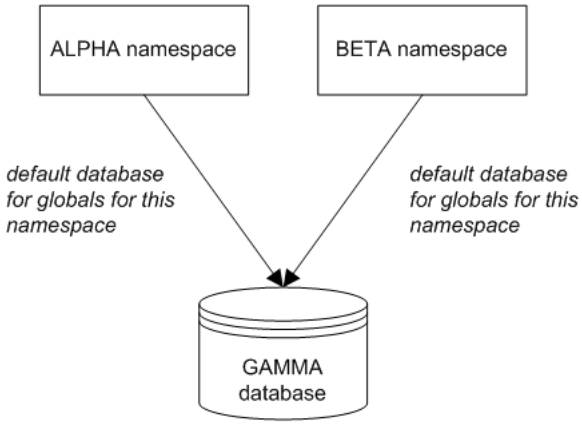
ロック名は本来任意ですが、キャレット (^) で始まるロック名を使用する場合、InterSystems IRIS は、これらのシナリオに適切な特別な動作を提供します。次のサブセクションで詳細を説明します。わかりやすくするために、排他ロックのみを説明します。論理は共有ロックでも同様です。

6.1 シナリオ 1 : 同じグローバル・データベースを持つ複数のネームスペース

前述のとおり、プロセス A が特定のロック名の付いた排他ロックを所有しているときに、その他のプロセスは、同じロック名の付いたロックを取得できません。

ロック名がキャレットで始まる場合、この規則は、同じグローバル・データベースを使用するすべてのネームスペースに適用されます。

例えば、ネームスペース ALPHA と BETA の両方がデータベース GAMMA をグローバル・データベースとして使用するように構成されているとします。以下は、部分的な例です。



次に以下のシナリオを考えてみます。

1. ネームスペース ALPHA では、プロセス A が ^MyGlobal(15) という名前の排他ロックを取得します。
2. ネームスペース BETA では、プロセス B が ^MyGlobal(15) という名前のロックを取得しようとします。この LOCK コマンドは復帰しません。このプロセスは、プロセス A がロックを解放するまでブロックされます。

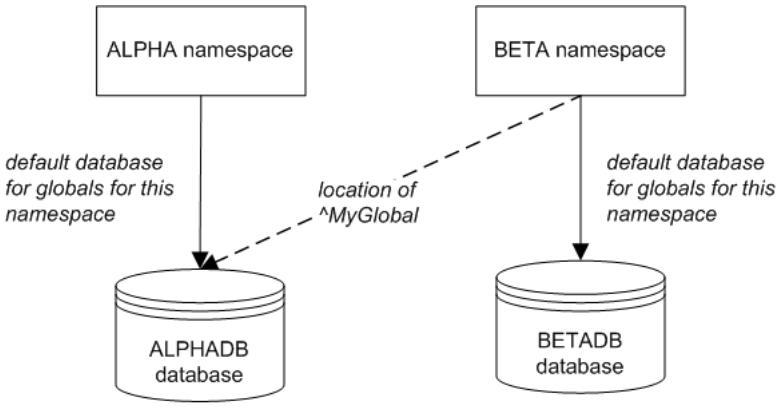
このシナリオでは、プロセス A が所有するロックのエントリのみがロック・テーブルに含まれています。ロック・テーブルを調べると、このロックの適用先の データベース を示していることがわかります。[ディレクトリ] 列を参照してください。例えば以下のようになります。

Owner	ModeCount	Reference	Directory
1284	Exclusive	^%SYS("CSP","Daemon")	c:\intersystems\iris\mgr\
26324	Exclusive	^ISC.LMFMON("License Monitor")	c:\intersystems\iris\mgr\
23400	Exclusive	^ISC.Monitor.System	c:\intersystems\iris\mgr\
23180	Exclusive	^TASKMGR	c:\intersystems\iris\mgr\
19776	Exclusive_e	^MyGlobal(15)	c:\intersystems\iris\mgr\gammadb\
23948	Exclusive	^%cspSession("vgMJ4iLMCL")	c:\intersystems\iris\mgr\irislocaldata\

6.2 シナリオ 2 : ネームスペースがマップされたグローバルを使用

1 つまたは複数のネームスペースにグローバル・マッピングが含まれている場合、システムは、該当のネームスペースにわたってロック・メカニズムを自動的に強制適用します。InterSystems IRIS は、既定以外のネームスペースでロックが取得されると、追加のロック・テーブル・エントリを自動的に作成します。

例えば、ネームスペース ALPHA がデータベース ALPHADB をグローバル・データベースとして使用するように構成されているとします。また、ネームスペース BETA が異なるデータベース (BETADB) をグローバル・データベースとして使用するように構成されているとします。ネームスペース BETA には、^MyGlobal が ALPHADB データベースに格納されることを指定するグローバル・マッピングも含まれています。以下は、部分的な例です。



次に以下のシナリオを考えてみます。

1. ネームスペース ALPHA では、プロセス A が ^MyGlobal(15) という名前の排他ロックを取得します。

   前述のシナリオと同様に、プロセス A が所有するロックのエントリのみがロック・テーブルに含まれています。このロックは ALPHADB データベースに適用されます。

```
19776   Exclusive_e   ^MyGlobal(15)                c:\intersystems\iris\mgr\alphadb\
```

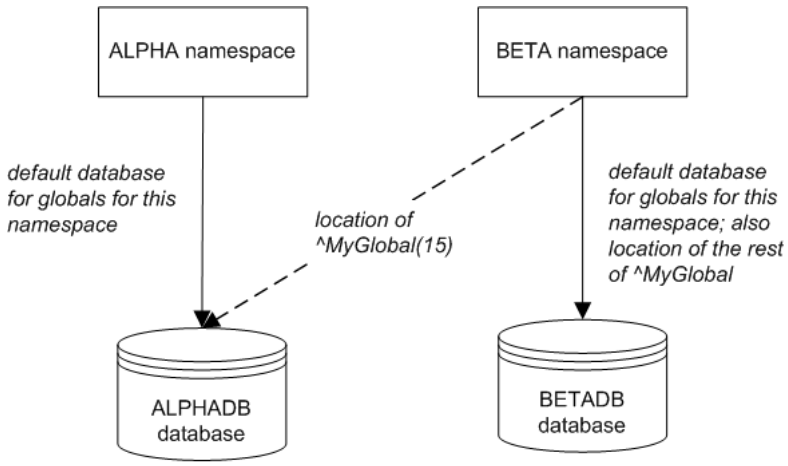
2. ネームスペース BETA では、プロセス B が ^MyGlobal(15) という名前のロックを取得しようとします。この LOCK コマンドは復帰しません。このプロセスは、プロセス A がロックを解放するまでブロックされます。

6.3 シナリオ 3 : ネームスペースがマップされたグローバル添え字を使用

添え字レベル・マッピングを使用するグローバル・マッピングが 1 つまたは複数のネームスペースに含まれている場合、システムは、該当のネームスペースにわたってロック・メカニズムを自動的に強制適用します。この場合、InterSystems IRIS は、既定以外のネームスペースでロックが取得されると、追加のロック・テーブル・エントリを自動的に作成します。

例えば、ネームスペース ALPHA がデータベース ALPHADB をグローバル・データベースとして使用するように構成されているとします。ネームスペース BETA は BETADB データベースをグローバル・データベースとして使用します。

また、ネームスペース BETA には、^MyGlobal(15) が ALPHADB データベースに格納される (一方、このグローバルの残りはネームスペースの既定の場所に格納される) ように添え字レベルのグローバル・マッピングも含まれているとします。以下は、部分的な例です。



次に以下のシナリオを考えてみます。

1. ネームスペース ALPHA では、プロセス A が ^MyGlobal(15) という名前の排他ロックを取得します。

   前述のシナリオと同様に、プロセス A が所有するロックのエントリのみがロック・テーブルに含まれています。このロックは ALPHADB データベース (c:\InterSystems\IRIS\mgr\alphadb など) に適用されます。

2. ネームスペース BETA では、プロセス B が ^MyGlobal(15) という名前のロックを取得しようとします。この LOCK コマンドは復帰しません。このプロセスは、プロセス A がロックを解放するまでブロックされます。

既定以外のネームスペースがロックを取得しても全体的な動作は同じですが、InterSystems IRIS による詳細の処理は若干異なります。ネームスペース BETA で、あるプロセスが ^MyGlobal(15) という名前のロックを取得するとします。この場合、ロック・テーブルには 2 つのエントリが含まれます。1 つは ALPHADB データベース用、もう 1 つは BETADB データベース用です。これらのロックは、両方ともネームスペース BETA のプロセスによって所有されます。

19776	Exclusive_e	^MyGlobal(15)	c:\intersystems\iris\mgr\alphadb\
19776	Exclusive_e	^MyGlobal(15)	c:\intersystems\iris\mgr\betadb\

このプロセスがロック名 ^MyGlobal(15) を解放すると、システムは両方のロックを自動的に削除します。

6.4 シナリオ 4 : 拡張グローバル参照

1 つのネームスペースで実行されているコードは、拡張参照を使用して、拡張参照を使用しない場合にはこのネームスペースでは使用できないグローバルにアクセスできます。この場合、InterSystems IRIS は、関連するデータベースに影響するロック・テーブルにエントリを追加します。このロックは、これを作成したプロセスに所有されます。例えば、以下のシナリオを考えてみます。わかりやすくするために、このシナリオにはグローバル・マッピングはありません。

1. プロセス A は ALPHA ネームスペースで実行されていて、このプロセスは以下のコマンドを使用して、BETA ネームスペースで使用できるグローバルでロックを取得します。

   ObjectScript

   ```
   lock ^["beta"]MyGlobal(15)
   ```

2. ここで、ロック・テーブルには以下のエントリが含まれています。

19776	Exclusive_e	^MyGlobal(15)	c:\intersystems\iris\mgr\betadb\

   これはグローバル名のみを示していることに注意してください (これにアクセスするために使用する参照ではありません)。また、このシナリオでは、BETADB は BETA ネームスペースの既定のデータベースです。

3. ネームスペース BETA では、プロセス B が ^MyGlobal(15) という名前のロックを取得しようとします。この LOCK コマンドは復帰しません。このプロセスは、プロセス A がロックを解放するまでブロックされます。

プロセス・プライベート・グローバルは技術的には拡張参照の一種ですが、InterSystems IRIS は、プロセス・プライベート・グローバル名をロック名として使用することをサポートしていません。定義によってそのようなグローバルにアクセスできるプロセスは 1 つしかないため、そのようなロックは必要ありません。

7 デッドロックの回避

増分ロックの使用は、潜在的な危険を伴います。これは、デッドロック と呼ばれる状況を引き起こす可能性があるからです。この状況は、既に他のプロセスによってロックされている変数に対して、2 つのプロセスがそれぞれ増分ロックをアサートするときに発生します。試行されているロックが増分ロックなので、既存のロックは解除されません。その結果、各プロセスは、他のプロセスが既存のロックを解除するまで待機する間、停止します。

以下に例を示します。

1. プロセス A が次のコマンドを発行します：`lock + ^MyGlobal(15)`
2. プロセス B が次のコマンドを発行します：`lock + ^MyOtherGlobal(15)`
3. プロセス A が次のコマンドを発行します：`lock + ^MyOtherGlobal(15)`

   この LOCK コマンドは復帰しません。このプロセスは、プロセス B がこのロックを解放するまでブロックされます。
4. プロセス B が次のコマンドを発行します：`lock + ^MyGlobal(15)`

   この LOCK コマンドは復帰しません。このプロセスは、プロセス A がこのロックを解放するまでブロックされます。しかし、プロセス A はブロックされ、ロックを解放できません。ここで、これらのプロセスは両方ともお互いの処理を待ちます。

デッドロックを防止するには以下のような方法があります。

- 常に timeout 引数を使用する。
- 増分 LOCK コマンドを発行する際にその順序に関して厳格なプロトコルに従う。すべてのプロセスが、ロック名に関して同じ順序に従っている限り、デッドロックが発生することはありません。単純なプロトコルは、照合順序でロックを追加するものです。
- 増分ロックではなく単純ロックを使用する（つまり、+ 演算子を使用しない）。前述のとおり、単純ロックでは、LOCK コマンドは最初に、プロセスによって以前から保持されていたすべてのロックを解放します(ただし、実際には単純ロックはあまり使用されません)。

デッドロックが発生した場合、管理ポータルまたは **^LOCKTAB** ルーチンを使用して解決できます。"監視ガイド" の "ロックの監視" を参照してください。

8 ロックの実際の使用方法

ここでは、実際にロックが使用される基本的な方法について説明します。

8.1 アプリケーション・データへのアクセスの制御

ロックは、グローバルに格納されているアプリケーション・データへのアクセスを制御するために使用されることが非常に多いです。アプリケーションは、このデータの特定の一部を読み取りまたは変更する必要があり、これを行う前に以下のように 1 つまたは複数のロックを作成する場合があります。

- アプリケーションが 1 つまたは複数のグローバル・ノードを読み取る必要があり、読み取り操作中に他のプロセスがこの値を変更しないようにする場合、これらのノードの共有ロックを作成します。
- アプリケーションが 1 つまたは複数のグローバル・ノードを変更する必要があり、変更中に他のプロセスがこれらのノードを読み取らないようにする場合、これらのノードの排他ロックを作成します。

その後、計画どおりに読み取りまたは変更を行います。完了したら、ロックを削除します。

ロック・メカニズムは純粋に規約によって動作することに注意してください。これらのノードの読み取りまたは変更を行う他のコードも、これらの操作を実行する前にロックを取得しようとする必要があります。

8.2 同時アクティビティの防止

ロックは、複数のプロセスが同じアクティビティを実行することを防止するためにも使用されます。このシナリオでは、グローバルも使用しますが、そのグローバルには、純粋なアプリケーション・データではなくアプリケーションの内部用データが含まれています。単純な例として、どのような場合でも複数のプロセスで実行されてはいけないルーチン (`^NightlyBatch`) があるとします。このルーチンは、処理のごく初期段階に以下のことを実行できます。

1. 特定のグローバル・ノードに排他ロックを作成します (例えば、`^AppStateData("NightlyBatch")`)。この操作にタイムアウトを指定します。
2. ロックが取得された場合、グローバルにノードを設定し、ルーチンが開始されたこと (および他の関連情報) を記録します。例えば以下のようになります。

   ObjectScript

   ```
   set ^AppStateData("NightlyBatch")=1
   set ^AppStateData("NightlyBatch","user")=$USERNAME
   ```

   タイムアウト時間内にロックが取得されなかった場合、このルーチンが既に開始されていることを示すエラー・メッセージで終了します。

処理の最後に、同じルーチンが該当のグローバル・ノードをクリアし、ロックを解放します。

以下の部分的な例はこの方法を示しています。これは、InterSystems IRIS が内部的に使用するコードから採用されています。

ObjectScript

```
lock ^AppStateData("NightlyBatch"):0
if '$TEST {
   write "You cannot run this routine right now."
   write !, "This routine is currently being run by user: "_^AppStateData("NightlyBatch","user")
   quit
}
set ^AppStateData("NightlyBatch")=1
set ^AppStateData("NightlyBatch","user")=$USERNAME
set ^AppStateData("NightlyBatch","starttime")=$h

//main routine activity omitted from example

kill ^AppStateData("NightlyBatch")
lock -^AppStateData("NightlyBatch")
```

9 詳細情報

ロックの詳細は、以下のリソースを参照してください。

- ″ObjectScript リファレンス″ の “LOCK”
- ″ObjectScript リファレンス″ の “^$LOCK”（^$LOCK はロックに関する情報を含む構造化システム変数です。）
- ″ObjectScript の使用法″ の “トランザクション処理”
- ″監視ガイド″ の ″ロックの監視″