記事
Toshihiko Minamoto · 2021年3月9日 16m read

セマフォを使用した共有リソースへの同時アクセスの実装

これまで Caché
のリソースアクセスを制御する方法が存在するかどうかを疑問に思っていた方の悩みを解決しました。
バージョン 2014.2 では、開発者がセマフォを操作できるようにする特別なクラスが追加されました。

セマフォは基本的に負ではない整数値の変数であり、次の 2 種類の操作の影響を受ける可能性があります。

- セマフォの P 操作は、セマフォの値を 1 つ減らそうとするものです。 P
  操作を実行する前のセマフォの値が 1 より大きい場合、P 操作は遅滞なく実行されます。
  操作前のセマフォの値が 0 の場合、P 操作を実行するプロセスは値が 0
  より大きくなるまで待機状態に切り替わります。
- セマフォの V 操作は、セマフォの値を 1 つ増やすものです。 このセマフォでの P
  操作を実行中に遅延したプロセスがあった場合、これらのプロセスのいずれかが待機状態を終了し、P
  操作を実行できます。

セマフォには、バイナリとカウンティングの 2 種類があります。 前者の場合は変数が 2 つの値（0 と
1）のみに制限されており、後者の場合は負ではない任意の整数をとることができるという点が異なります。

セマフォは次のようないくつかの事例で役立ちます。

1. セマフォによる排他制御

バイナリセマフォ S は、2
つ以上のプロセスによる共有データの同時変更の防止などの排他制御を実装するために作成されます。
このセマフォの初期値は 1 です。 クリティカルセクション（同時に 1
つのプロセスでのみ実行できるセクション）は、P（S）（最初）と V（S）（最後）の括弧で囲まれています。
クリティカルセクションに入るプロセスは P（S）操作を実行し、セマフォを 0 に切り替えます。
クリティカルセクションに別のプロセスがあるためにセマフォの値がすでに 0
になっている場合、このプロセスは現在のプロセスが終了し、終了時に V（S）操作を実行するまで P
操作がブロックされます。

2. セマフォによる同期

初期値が 0 のバイナリセマフォ S は同期を行う目的で作成されます。 初期値 0
は、イベントがまだ発生していないことを意味します。 イベントについて通知するプロセスは、値を 1
に設定する V（S）操作を実行します。 イベントを待機しているプロセスは、P（S）操作を実行します。
その時点ですでにイベントが発生している場合は、待機中のプロセスが引き続き実行されます。
そうでない場合、プロセスはシグナル送信プロセスが V（S）操作を実行するまで待機状態に切り替わります。

複数のプロセスが同じイベントを待機している場合、P（S）操作を正常に実行したプロセスは即座に
V（S）操作を実行し、次のキューに格納されたプロセスに新たなイベント信号を送信する必要があります。

3. セマフォリソースカウンター

特定のリソースが N ユニットある場合、その割り当てを制御する目的で値が N の一般的な S
セマフォが作成されます。 リソースは P（S）コマンドで割り当てられ、V（S）コマンドで解放されます。

したがって、セマフォの値には空きリソース単位の数が反映されます。 セマフォの値が 0
の場合は使用可能なユニットがそれ以上存在しないことを意味し、リソースを使用するいずれかのプロセスが
V（S）操作を実行してそれを解放するまで、このリソースを要求するプロセスは
P（S）操作を待機する状態に切り替わります。

これらのオプションはすべて Caché で負ではない 64
ビットの整数値をカプセル化し、動作中のすべてのプロセスに対してその値を変更するメソッドを提供する
%SYSTEM.Semaphore クラスを使用して実装できます。

上記の 3 番目の事例でセマフォカウンターを使用する方法を見てみましょう。

大学内の国際的な科学論文データベースにアクセスできる 10 人分の枠があると仮定しましょう。
このように、アクセスを希望する学生間で共有させる必要のあるリソースがあるとします。 各学生にはデータベースにアクセスするためのログインとパスワードが個別に割り当てられていますが、システムを同時に操作できるのは大学の 10 人以下のユーザーだけです。
学生がデータベースにログインしようとするたびに、使用可能な枠の数を 1 つ減らす必要があります。
したがって、学生がアクセスを終了する際には 1 つの枠を全体のアクセスプールに戻す必要があります。

特定の学生にアクセスを許可するか、待機させるかを確認するため、初期値が 10
のセマフォカウンターを使用します。 このアクセス許可を付与する仕組みを確認するため、ログを記録します。
Main クラスは変数を初期化し、学生の操作を監視します。 別のクラスが %SYSTEM.Semaphore
から継承され、セマフォが作成されます。 また、さまざまなユーティリティ用に別のクラスを作成しましょう。
そして最後の 2 つのクラスは、データベースからの学生のログインとログアウトをエミュレートします。 サーバーはシステムへのログインとログアウトを行うユーザーの名前を「認識」しているため、ここではアクティブユーザー（^LoggedUsers）に関する情報を格納するこの「認識」をシミュレートするために個別のグローバルを作成します。 このグローバルを使用してログイン中の学生のログイン名を登録し、ログアウトしている学生にはランダムな名前を選択します。

まずは次の処理を行う Main クラスから始めましょう。

1. セマフォを作成します。
2. セマフォを初期値（科学論文データベースにアクセスできる 10 人分の空き枠に対応する
   10）に設定します。
3. プロセスを停止し、セマフォを削除します。
4.
   ログを表示します。

```
Class SemaphoreSample.Main Extends %RegisteredObject [ ProcedureBlock ]
{

  /// ?????????
  ClassMethod Run()
  {
      // ???????????????
      Do ##class(SemaphoreSample.Util).InitLog()
      Do ##class(SemaphoreSample.Util).InitUsers()

      Set msg = "Process start "
      Do ..Log(msg)

      // ???????????
      Set inventory = ##class(SemaphoreSample.Counter).%New()
      If ('($ISOBJECT(inventory))) {
          Set msg = "The SemaphoreSample.Counter %New() class method hasn’t wor
ked yet"
          Do ..Log(msg)
          Quit
```

```
        }

        // ??????????
        if 'inventory.Init(10) {
            Set msg = "There was a problem initializing the semaphore"
            Do ..Log(msg)
            Quit
        }

        // ??????????
        Set msg = "Press any key to block access..."
        Do ..Log(msg)

        Read *x

        //???????
        Set msg = "The semaphore has been removed with the following status:  " _
 inventory.Delete()
        Do ..Log(msg)
        Set msg = " Process end "
        Do ..Log(msg)

        do ##class(SemaphoreSample.Util).ShowLog()

        Quit
    }

    /// ??????????????????
    ClassMethod Log(msg As %String) [ Private ]
    {
        Do ##class(SemaphoreSample.Util).Logger($Horolog, "Main", msg)
        Quit
    }

}
```

次に作成するクラスは、さまざまなユーティリティを備えたクラスです。
このクラスはテストアプリケーションの動作に必要になります。
また、次の処理を行うクラスメソッドが含まれています。

1. グローバルにログを記録する準備をする（^SemaphoreLog）。
2. ログをグローバルに保存する。
3. ログを表示する。
4. アクティブユーザーの名前をグローバルに保存する（^LoggedUsers）。
5. アクティブユーザーのリストからランダムな名前を選択する。
6. インデックスを指定してグローバルからアクティブユーザーの名前を削除する。

```
Class SemaphoreSample.Util Extends %RegisteredObject [ ProcedureBlock ]
{

/// ??????
ClassMethod InitLog()
{
    // ?????????????
```

```
    Kill ^SemaphoreLog Set ^SemaphoreLog = 0

    Quit
}

/// ??????
ClassMethod InitUsers()
{
    //??????????????????
        if $data(^LoggedUsers) '= 0
    {
        Kill ^LoggedUsers
    }
    Set ^LoggedUsers = 0
}

/// ????????????
ClassMethod Logger(time As %DateTime, sender As %String, msg As %String)
{
    Set inx = $INCREMENT(^SemaphoreLog)
    Set ^SemaphoreLog(inx, 0) = time
    Set ^SemaphoreLog(inx, 1) = sender
    Set ^SemaphoreLog(inx, 2) = msg
    Write "(", ^SemaphoreLog, ") ", msg_" ? "_$ztime($PIECE(time,",",2), 1), !
    Quit
}

/// ???????????
ClassMethod ShowLog()
{
    Set msgcnt = $GET(^SemaphoreLog, 0)
    Write "Message log: number of records = ", msgcnt, !, !
    Write "#", ?5, "Time", ?12, "Sender", ?25, "Message", !

    For i = 1 : 1 : msgcnt {
        Set time = ^SemaphoreLog(i, 0)
        Set sender = ^SemaphoreLog(i, 1)
        Set msg = ^SemaphoreLog(i, 2)
        Write i, ")", ?5, $ztime($PIECE(time,",",2), 1), ?15, sender, ":", ?35,
 msg, !
    }
    Quit
}

/// ??????????????????????
ClassMethod AddUser(Name As %String)
{
    Set inx = $INCREMENT(^LoggedUsers)
    set ^LoggedUsers(inx) = Name
}

/// ??????????????????????
ClassMethod DeleteUser(inx As %Integer)
{
    kill ^LoggedUsers(inx)
}

/// ??????????????????????
ClassMethod ChooseUser(ByRef Name As %String) As %Integer
```

```
{
    // ????????????????????????????????
        if $data(^LoggedUsers) = 1
    {
        Set Name = ""
        Quit -1
    } else
    {
        Set Temp = ""
        Set Numb = $Random(10)+5
           For i = 1 : 1: Numb {
                Set Temp = $Order(^LoggedUsers(Temp))
                   // ??????????? 1 ??????????
                   // ????????????????????????
                if (Temp = "")
                {
                    set Temp = $Order(^LoggedUsers(""))
                }
            }
            set Name = ^LoggedUsers(Temp)
            Quit Temp
        }
    }
}
```

次のクラスはセマフォを実装しています。 また、%SYSTEM.Semaphore
システムクラスを拡張しており、次のメソッドが含まれています。

1. セマフォの一意の名前を返す。
2. イベントをログに保存する。
3. セマフォの作成と破棄のイベントをログに記録する（コールバックメソッド）。
4. セマフォを作成して初期化する。

```
Class SemaphoreSample.Counter Extends %SYSTEM.Semaphore
{

/// ??????????????????????
ClassMethod Name() As %String
{
    Quit "Counter"
}

/// ??????????????????????
Method Log(Msg As %String) [ Private ]
{
    Do ##class(SemaphoreSample.Util).Logger($Horolog, ..Name(), Msg)
    Quit
}

/// ????????????????????????????
Method %OnNew() As %Status
{
    Set msg = "Creating a semaphore "
    Do ..Log(msg)
```

```
        Quit $$$OK
    }

    /// ??????????? Method Init(initvalue = 0) As %Status
    {
        Try {
            If (..Create(..Name(), initvalue)) {
                Set msg = "Created: """ _ ..Name()
                        _ """; Initial value = " _ initvalue
                Do ..Log(msg)
                Return 1
            }
            Else {
                Set msg = "There was a problem creating a semaphore with the name =
 """ _ ..Name() _ """"
                Do ..Log(msg)
                Return 0
            }
        } Catch errobj {
            Set msg = "There was an error creating a semaphore: "_errobj.Data
            Do ..Log(msg)
            Return 0
        }
    }

    /// ??????????????????????????
    Method %OnClose() As %Status [ Private ]
    {
        Set msg = "Closing the semaphore "
        Do ..Log(msg)
        Quit $$$OK
    }

    }
```

セマフォが作成時にシステムに渡される名前で識別されることを認識しておいてください。
この名前はローカル/グローバル変数の要件に準拠し、一意でなければなりません。 セマフォは一般的に自身が作成されたデータベースインスタンスに保存されており、このインスタンスの他のプロセスからアクセスできます。
グローバル変数の命名要件を満たしているセマフォは、ECP
を含むすべてのアクティブなプロセスで使用できるようになります。

最後の 2 つのクラスは、システムにログインおよびログアウトするユーザーをシミュレートします。
説明を簡単にするため、ちょうど 25 人のユーザーがいると仮定しましょう。 プロセスを開始した後にキーが押されるまで新規ユーザーを作成し続けることもできますが、ここでは単純に有限ループを使用することにしました。
どちらのクラスも既存のセマフォに接続し、カウンターを減らしたり（ログイン）、増やしたり（ログアウト）します。 ここでは学生が自分の順番を無期限に待つことを想定しています（学生はなんとしても図書室に入りたいと思っている）。そのため、待機時間を無期限に設定できる Decrement 関数を使用しています。
それ以外の場合は、正確なタイムアウト時間をミリ秒単位で指定できます。 全ユーザーが同時ログインを試みることのないよう、ログインを試みる前に適当な一時停止処理を追加しましょう。

```
Class SemaphoreSample.LogIn Extends %RegisteredObject [ ProcedureBlock ]
{

/// ??????????????????????
ClassMethod Run() As %Status
{
```

```
    //????????????????????????
    Set cell = ##class(SemaphoreSample.Counter).%New()
    Do cell.Open(##class(SemaphoreSample.Counter).Name())

    // ????????????
    // ????? 25 ?????????????
    For deccnt = 1 : 1 : 25 {
        // ??????????
       Set Name = ##class(%Library.PopulateUtils).LastName()

        try
        {
           Set result =  cell.Decrement(1, -1)
        } catch
        {
           Set msg = "Access blocked"
           Do ..Logger(##class(SemaphoreSample.Counter).Name(), msg)
           Return
        }
        do ##class(SemaphoreSample.Util).AddUser(Name)
        Set msg = Name _ " entered the system "
        Do ..Logger(Name, msg)

        Set waitsec = $RANDOM(10) + 7
        Hang waitsec
    }
    Set msg = "There are no more users waiting to log in to the system"
    Do ..Logger(##class(SemaphoreSample.Counter).Name(), msg)
    Quit $$$OK
}

/// ?????????????????
ClassMethod Logger(id As %String, msg As %String) [ Private ]
{
    Do ##class(SemaphoreSample.Util).Logger($Horolog, id, msg)
    Quit
}

}
```

このモデルでは、サーバーから切断された際に接続中のユーザーが他にもいるかどうかもチェックする必要があります。 このため、最初にユーザーを含むグローバル（^LoggedUsers）の内容を確認し、空だった場合はしばらく待機してからログインできた人がいるかどうかをチェックしています。

```
Class SemaphoreSample.LogOut Extends %RegisteredObject [ ProcedureBlock ]
{

/// ???????????????
ClassMethod Run() As %Status
{
    Set cell = ##class(SemaphoreSample.Counter).%New()
    Do cell.Open(##class(SemaphoreSample.Counter).Name())

    // ????????????
    For addcnt = 1 : 1 : 25 {
        Set inx = ##class(SemaphoreSample.Util).ChooseUser(.Name)
```

```
            while inx = -1
            {
                Set waitsec = $RANDOM(10) + 1
                Hang waitsec
                Set inx = ##class(SemaphoreSample.Util).ChooseUser(.Name)
            }
            try
            {
                Do cell.Increment(1)
            } catch
            {
                Set msg = "Access blocked"
                Do ..Logger(##class(SemaphoreSample.Counter).Name(), msg)
                Return
            }


            Set waitsec = $RANDOM(15) + 2
            Hang waitsec
        }
        Set msg = "All users have logged out of the system"
        Do ..Logger(##class(SemaphoreSample.Counter).Name(), msg)
        Quit $$$OK
    }

    /// ????????????????
    ClassMethod Logger(id As %String, msg As %String) [ Private ]
    {
        Do ##class(SemaphoreSample.Util).Logger($Horolog, id, msg)
        Quit
    }

    }
```

プロジェクトの準備が整いました。 このプロジェクトをコンパイルし、起動して結果を確認してみましょう。
それぞれの作業を 3 つのターミナルウィンドウで別々に行います。

最初のウィンドウでは、必要に応じて目的のネームスペースに切り替えてください（私の場合は USER
ネームスペースでプロジェクトを作成しました）。

```
zn "USER"
```

その後、Main クラスから作成した「サーバー」を起動するメソッドを呼び出します。

```
do ##class(SemaphoreSample.Main).Run()
```

2 番目のウィンドウでは、システムにログインするユーザーを生成する Login クラスから Run
メソッドを呼び出します。

```
do ##class(SemaphoreSample.LogIn).Run()
```

最後のウィンドウでは、システムからログアウトするユーザーを生成する LogOut クラスから Run
メソッドを呼び出します。

```
do ##class(SemaphoreSample.LogOut).Run()
```

全員がログインおよびログアウトすると、3 つのウィンドウすべてにログが表示されます。
状況を把握しやすくするため、認証された最初の 10
人のユーザーがシステムにログインするのを待ち、セマフォの値が 0
未満にならないことを実証してから、LogOut のルーチンを開始することをお勧めします。

この例で使用されている Increment メソッドと Decrement
メソッドとは別に、待機リストと対応する次のメソッドを使用してセマフォを操作することができます。

- AddToWaitMany — セマフォ関連の操作をリストに追加する
- RmFromWaitMany — リストから操作を削除する
- WaitMany — セマフォを使用したすべての操作が終了するまで待機する

この場合、WaitMany はリスト上のすべてのタスクをループし、操作が正常に完了した時点で WaitCompleted
メソッド（開発者が自分で実装する必要があります）を呼び出します。 このメソッドはセマフォが操作にゼロ以
外の値を割り当てた場合、またはタイムアウトによって待機終了になった場合に呼び出されます。
カウンターから減らす数は、このメソッドの引数に返されます（タイムアウトの場合は 0）。
メソッドの動作が終了すると、セマフォが WaitMany タスクリストから削除されて次のタスクが実行されます。

Caché
のセマフォに関する詳細については、公式ドキュメント
（別のセマフォの例も掲載されています）とクラスの説明を参照してください。

このプロジェクトは GitHub で公開されています。

皆様からのコメントやご提案をお待ちしております。

#チュートリアル #ヒントとコツ #Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%BB%E3%83%9E%E3%83%95%E3%82%A9%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%9F%E5%85%B1%E6%9C%89%E3%83%AA%E3%82%BD%E3%83%BC%E3%82%B9%E3%81%B8%E3%81%AE%E5%90%8C%E6%99%82%E3%82%A2%E3%82%AF%E3%82%BB%E3%82%B9%E3%81%AE%E5%AE%9F%E8%A3%85