記事
Toshihiko Minamoto · 2020年10月28日 12m read

$Sequence 関数について

この記事では $Increment 関数と $Sequence 関数を比較します。

まずは、$Increment 関数を聞いたことがないという方のために、その概要を説明いたします。 $Increment は、CachéObjectScript の関数で、引数をアトミックに 1 ずつインクリメントし、結果の値を返します。 $Increment にパラメーターとして渡せるのはグローバル変数ノードとローカル変数ノードのみで、任意の式を渡すことはできません。 $Increment は連続する ID の割り当てに多用されます。 その場合、$Increment のパラメーターにはグローバルノードがよく使用されます。 $Increment を使用するプロセスには確実に任意の ID が割り当てられます。

```
for i=1:1:10000 {
     set Id = $Increment(^Person) ; ??? ID
     set surname = ##class(%PopulateUtils).LastName() ; ???????????
     set name = ##class(%PopulateUtils).FirstName()  ; ????????????
     set ^Person(Id) = $ListBuild(surname, name)
}
```

$Increment
の問題は、多数のプロセスが並行して行を追加していると、各プロセスはそれぞれの順番が回ってくるまで、ID を持つグローバルノードの値をアトミックに変更することができない場合があるという点です。上の例では、^Person がこれに該当します。

この問題を解決するためにデザインされたのが新しい関数$Sequence です。 $Sequence は Caché 2015.1 から導入されています。 $Increment と同様に、$Sequence もそのパラメーターの値をアトミックにインクリメントします。 $Increment と違う点として、$Sequence は前のプロセスから後続のカウンター値を現在のプロセス用にリザーブし、リザーブしておいた範囲の次の値を同じプロセス内の次の呼び出し中に返します。 $Sequence はリザーブしておく値の数を自動的に計算します。 プロセスが $Sequence を呼び出す頻度が増えると、$Sequence がリザーブする値の数も増えていきます。

```
USER>kill ^myseq

USER>for i=1:1:15 {write "increment:",$Seq(^myseq)," allocated:",^myseq,! }
increment:1 allocated:1
increment:2 allocated:2
increment:3 allocated:4
increment:4 allocated:4
increment:5 allocated:8
increment:6 allocated:8
increment:7 allocated:8
increment:8 allocated:8
increment:9 allocated:16
increment:10 allocated:16
increment:11 allocated:16
increment:12 allocated:16
increment:13 allocated:16
increment:14 allocated:16
increment:15 allocated:16
```

$Sequence(^myseq) が 9 を返した時点で、次の 8 個の値 (16 まで)
が既に現在のプロセス用にリザーブされています。 別のプロセスが $Sequence を呼び出すと、10 ではなく、17 という値が割り当てられます。

$Sequence は、複数のグローバルノードを同時にインクリメントするプロセスを考えてデザインされています。 $Sequence は値をリザーブするため、リザーブされたすべての値がプロセスで使用されない場合は、ID
の間にギャップが生じる場合があります。 $Sequence を使用する主な目的は、連続する ID
を生成することです。 $Increment は $Sequence よりもジェネリクス型の関数と言えます。

それでは、$Increment と $Sequence のパフォーマンスを比較してみましょう。

```
Class DC.IncSeq.Test
{

ClassMethod filling()
{
    lock +^P:"S"
    set job = $job
     for i=1:1:200000 {
         set Id = $Increment(^Person)
         set surname = ##class(%PopulateUtils).LastName()
         set name = ##class(%PopulateUtils).FirstName()
         set ^Person(Id) = $ListBuild(job, surname, name)
     }
     lock -^P:"S"
}

ClassMethod run()
{
    kill ^Person
    set z1 = $zhorolog
    for i=1:1:10 {
        job ..filling()
     }
     lock ^P
     set z2 = $zhorolog - z1
     lock
     write "done:",z2,!
}

}
```

メソッド run は 10 個のプロセスを実行し、それぞれが 200,000 件のレコードを ^Person の Global 変数ノードに挿入しています。 子プロセスの終了を待機するため、 メソッド run は ^P
に対して排他ロックを取得しようとします。 子プロセスがジョブを終了し、^P
の共有ロックを開放すると、メソッド run は ^P の排他ロックを取得し、実行を続けます。
この直後に、システム変数 $zhorolog
の時間を記録し、これらのレコードを挿入するのにかかった時間を計算します。 低速 HDD
を搭載したマルチコアノートブックでは 40
秒かかりました（科学実験として、これまで数回実行していますが、今回で 5 回目です):

```
USER>do ##class(DC.IncSeq.Test).run()
done:39.198488
```

この 40 秒間を分析すると興味深いことが分かりました。 ^%SYS.MONLBL を実行すると、ID の取得に合計 100 秒もかかっていることが分かります。10 個のプロセスに 100 秒かかっているということは、各プロセスが新しい ID を取得するのに 10 秒かかっているということです。ファーストネームとラストネームの取得には 1.7 秒、データグローバルにデータを書き込むのには 28.5 秒かかっています。

下に示す %SYS.MONLBL レポートの 1 列目は行番号、2 列目はこの行が実行された回数、3 列目はこの行を実行するのにかかった秒数を表しています。

```
 ; ** ???? 'filling' ???????**
1            10    .001143    lock +^P:"S"
2            10    .000055    set job = $JOB
3            10    .000118     for i=1:1:200000 {
4       1998499 100.356554         set Id = $Increment(^Person)
5       1993866  10.409804         set surname = ##class(%PopulateUtils).LastName()
6       1990461   6.347832         set name = ##class(%PopulateUtils).FirstName()
7       1999762  285.54603         set ^Person(Id) = $ListBuild(job, surname, name)
8       1999825   3.393706     }
9            10    .000259     lock -^P:"S"
 ; ** ???? 'filling' ?????????? **
 ;
 ; ** ???? 'run' ??????? **
1             1    .005503    kill ^Person
2             1    .000002    set z1 = $zhorolog
3             1    .000002    for i=1:1:10 {
4            10    .201327        job ..filling()
5             0          0     }
6             1  43.472692     lock ^P
7             1     .00003     set z2 = $zhorolog - z1
8             1     .00001     lock
9             1    .000053     write "done:",z2,!
 ; ** ???? 'run' ??????????**
```

プロファイリングが実行されたため、前回よりも 4 秒長くなり、合計で 43.47 秒かかりました。

テストコードの filling メソッドで、1 つだけ入れ替えたい部分があります。 $Increment(^Person) を $Sequence(^Person) に変更して、もう一度テストを実行しましょう。

```
USER>do ##class(DC.IncSeq.Test).run()
done:5.135189
```

意外な結果になりましたね。 $Sequence によって ID を取得する時間は削減されましたが、データをグローバルに保管するのにかかっていた 28.5 秒はどうなったのでしょう。 では、^%SYS.MONLBL を確認してみましょう。

```
 ; ** ???? 'filling' ??????? **
1            10    .001181    lock +^P:"S"
2            10    .000026    set job = $JOB
3            10    .000087     for i=1:1:200000 {
4       1802473   1.996279         set Id = $Sequence(^Person)
5       1784910   4.429576         set surname = ##class(%PopulateUtils).LastName()
6       1853508   3.829051         set name = ##class(%PopulateUtils).FirstName()
7       1838752  32.281624         set ^Person(Id) = $ListBuild(job, surname, name)
8       1951569     1.0243     }
9            10    .000219     lock -^P:"S"
```

```
 ; ** ???? 'filling' ?????????? **
 ;
 ; ** ???? 'run' ??????? **
1             1    .006514    kill ^Person
2             1    .000002    set z1 = $zhorolog
3             1    .000002    for i=1:1:10 {
4            10    .385055        job ..filling()
5             0          0     }
6             1   6.558119     lock ^P
7             1    .000011     set z2 = $zhorolog - z1
8             1    .000008     lock
9             1    .000025     write "done:",z2,!
 ; ** ???? 'run' ?????????? **
```

各プロセスは 10 秒ではなく、わずか 0.2 秒で ID を取得できるようになりました。
よく解らないのは、なぜ各プロセスがたった 3.23 秒でデータを保管できているのか、という点です。
それは、グローバルノードはデータブロックに保管されるほか、通常は各ブロックのサイズが 8192 バイトだからです。 プロセスは、(set ^Person(Id) = … のように)
グローバルノードの値を変更する前に、ブロック全体をロックします。
複数のプロセスが同時に同じブロック内のデータを変更しようとすると、それを変更できるのは 1 つのプロセスに限られるので、他のプロセスはそれが終了するのを待たなくてはいけません。

$Increment を使って新しい ID を生成する際に作成されたグローバルを見てみましょう。
連続するレコードに同じプロセス ID が割り当てられることはまずありません (プロセス ID はデータリストの最初の要素として保管していることをお忘れなく)。

```
1:    ^Person(100000)    =    $lb("12950","Kelvin","Lydia")
2:     ^Person(100001)    =    $lb("12943","Umansky","Agnes")
3:     ^Person(100002)    =    $lb("12945","Frost","Natasha")
4:     ^Person(100003)    =    $lb("12942","Loveluck","Terry")
5:     ^Person(100004)    =    $lb("12951","Russell","Debra")
6:     ^Person(100005)    =    $lb("12947","Wells","Chad")
7:     ^Person(100006)    =    $lb("12946","Geoffrion","Susan")
8:     ^Person(100007)    =    $lb("12945","Lennon","Roberta")
9:     ^Person(100008)    =    $lb("12944","Beatty","Mark")
10:     ^Person(100009)    =    $lb("12946","Kovalev","Nataliya")
11:     ^Person(100010)    =    $lb("12947","Klingman","Olga")
12:     ^Person(100011)    =    $lb("12942","Schultz","Alice")
13:     ^Person(100012)    =    $lb("12949","Young","Filomena")
14:     ^Person(100013)    =    $lb("12947","Klausner","James")
15:     ^Person(100014)    =    $lb("12945","Ximines","Christine")
16:     ^Person(100015)    =    $lb("12948","Quine","Mary")
17:     ^Person(100016)    =    $lb("12948","Rogers","Sally")
18:     ^Person(100017)    =    $lb("12950","Ueckert","Thelma")
19:     ^Person(100018)    =    $lb("12944","Xander","Kim")
20:     ^Person(100019)    =    $lb("12948","Ubertini","Juanita")
```

並行プロセスが同じブロックにデータを書き込もうとしていたため、データの変更にかかる時間よりも待機していた時間の方が長くなっていました。 $Sequence を使用すると、ID はチャンク生成されるので、異なるプロセスが異なるブロックを使用する可能性が高くなります。

```
1:     ^Person(100000)    =    $lb("12963","Yezek","Amanda")
// ??????? 12963 ?????? 351 ?
353:     ^Person(100352)    =    $lb("12963","Young","Lola")
354:     ^Person(100353)    =    $lb("12967","Roentgen","Barb")
```

実際のプロジェクトがこのサンプルのような状態に陥っているという方は、$Increment の代わりに $Sequence を使用することを検討してください。 当然ですが、$Increment をすべて $Sequence に入れ替えてしまう前に、まずはドキュメンテーション を参照してください。

また、このテストの内容をそのまま鵜呑みにはせず、ご自身の目で確かめてください。

Caché 2015.2 からは、$Increment の代わりに $Sequence を使ってテーブルを操作できるようになりました。そのために $system.Sequence.SetDDLUseSequence というシステム関数が用意されています。Management Portal の SQL 設定からも同じオプションを使用できます。

また、クラス定義には -- IDFunction という新しいストレージパラメーターがあります。「increment」にデフォルト設定されており、ID 生成に $Increment が使用されることを意味します。 (Inspector > Storage > Default > IDFunction と移動して)「sequence」に変更することができます。

おまけ

自分の notebook で他にも簡単なテストを行いました。DB サーバーをホストオペレーションシステムに、そしてアプリケーションサーバーを同じ notebook のゲスト VM にインストールした小さな ECP 構成を使っています。 ^Person はリモートのデータベースにマッピングしています。
いたってベーシックなテストなので、それを基に一般論を出すのは控えておきます。 $Increment と ECP を使用する際には 考慮すべき点 がいくつかあります。 では、結果をご覧ください。

$Increment を使った場合

```
USER>do ##class(DC.IncSeq.Test).run()
done:163.781288
```

^%SYS.MONLBL:

```
 ; ** ???? 'filling' ??????? **
1            10    .000503         --     lock +^P:"S"
2            10    .000016    set job = $job
3            10    .000044    for i=1:1:200000 {
4       1843745 1546.57015        set Id = $Increment(^Person)
5       1880231   6.818051        set surname = ##class(%PopulateUtils).LastName()
6       1944594   3.520858        set name = ##class(%PopulateUtils).FirstName()
7       1816896  16.576452        set ^Person(Id) = $ListBuild(job, surname, name)
8       1933736    .895912    }
9            10    .000279    lock -^P:"S"
 ; ** ???? 'filling' ?????????? **
 ;
 ; ** ???? 'run' ??????? **
1             1    .000045    kill ^Person
2             1    .000001    set z1 = $zhorolog
3             1    .000007    for i=1:1:10 {
4            10    .059868        job ..filling()
5             0          0    }
6             1 170.342459    lock ^P
7             1    .000005    set z2 = $zhorolog - z1
8             1    .000013    lock
9             1    .000018    write "done:",z2,!
```

```
 ; ** ???? 'run' ?????????? **
```

$Sequence を使った場合

```
USER>do ##class(DC.IncSeq.Test).run()
done:13.826716
```

^%SYS.MONLBL

```
 ; ** ???? 'filling' ??????? **
1            10    .000434     lock +^P:"S"
2            10    .000014    set job = $job
3            10    .000033    for i=1:1:200000 {
4       1838247  98.491738        set Id = $Sequence(^Person)
5       1712000   3.979588        set surname = ##class(%PopulateUtils).LastName()
6       1809643   3.522974        set name = ##class(%PopulateUtils).FirstName()
7       1787612  16.157567        set ^Person(Id) = $ListBuild(job, surname, name)
8       1862728    .825769    }
9            10    .000255    lock -^P:"S"
 ; ** ???? 'filling' ?????????? **
 ;
 ; ** ???? 'run' ??????? **
1             1    .000046    kill ^Person
2             1    .000002    set z1 = $zhorolog
3             1    .000004    for i=1:1:10 {
4            10    .037271        job ..filling()
5             0          0    }
6             1  14.620781    lock ^P
7             1    .000005    set z2 = $zhorolog - z1
8             1    .000013    lock
9             1    .000016    write "done:",z2,!
 ; ** ???? 'run' ?????????? **
```

#ObjectScript #Caché

ソースURL:
https://jp.community.intersystems.com/post/sequence-%E9%96%A2%E6%95%B0%E3%81%AB%E3%81%A4%E3%81%84%E3%81%A6