Artículo
Kurro Lopez · 16 ene, 2020 Lectura de 11 min

En la función $Sequence

En este artículo vamos a comparar las funciones $Increment y $Sequence.

En primer lugar, una nota para los lectores que nunca han oído hablar de $Increment. $Increment es una función Caché ObjectScript que realiza una operación atómica para incrementar su argumento en 1 y devolver el valor resultante. Solo puede pasar un nodo variable global o local como parámetro a $Increment, no una expresión arbitraria. $Increment se usa mucho al asignar ID secuenciales. En tales casos, el parámetro de $Increment suele ser un nodo global. $Increment garantiza que cada proceso que lo use obtenga una identificación única.

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

El problema con $Increment es que si muchos procesos agregan filas en paralelo, estos procesos pueden pasar tiempo esperando su turno para cambiar atómicamente el valor del nodo global que contiene la ID - ^Person en el ejemplo anterior

$Sequence es una nueva función que fue diseñada para manejar este problema. $Sequence está disponible desde Caché 2015.1. Al igual que $Increment, $Sequence incrementa atómicamente el valor de su parámetro. A diferencia de $Increment, $Sequence reservará algunos valores de contador posteriores para el proceso actual y durante la próxima llamada en el mismo proceso simplemente devolverá el siguiente valor del rango reservado. $Sequence calcula automáticamente cuántos valores reservar. Si más a menudo el proceso llama $Sequence, más valores $Sequence reserva:

```
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
```

Cuando $Sequence (^myseq) devolvió 9, los siguientes 8 valores (hasta 16) ya estaban reservados para el proceso actual. Si otro proceso llama a $Sequence, obtendría un valor de 17, no de 10.

$Sequence está diseñado para procesos que incrementan simultáneamente algún nodo global. Dado que los valores de reserva de $Sequence, los ID pueden tener huecos si el proceso no utiliza todos los valores reservados. El uso principal de $Sequence es la generación de ID secuenciales. En comparación con $Sequence, $Increment es una función más genérica.

Comparemos el rendimiento de $Increment y $Sequence:

```
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,!
}

}
```

El método run ejecuta 10 procesos, cada uno insertando 200.000 registros en el global ^Person. Esperar hasta que finalicen los procesos secundarios. El método run intenta obtener un bloqueo exclusivo en ^P. Cuando los procesos secundarios finalizan su trabajo y liberan el bloqueo compartido en ^P, la ejecución adquirirá un bloqueo exclusivo en ^P y continuará la ejecución. Justo después de esto, registramos el tiempo de la variable del sistema $zhorolog y calculamos cuánto tiempo llevó insertar estos registros. Mi portátil multinúcleo con disco duro lento tardó 40 segundos (como experimento, lo ejecuté varias veces antes, así que esta fue la quinta ejecución):

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

Es interesante profundizar en estos 40 segundos. Mediante la ejecución de ^%SYS.MONLBL podemos ver que se gastaron 100 segundos en obtener la identificación. 100 segundos / 10 procesos = cada proceso pasó 10 segundos para adquirir una nueva ID, 1.7 segundos para obtener el nombre y apellido, y 28.5 segundos para escribir datos en datos globales.

La primera columna en el informe %SYS.MONLBL a continuación es el número de línea, la segunda es cuántas veces se ejecutó esta línea y la tercera es cuántos segundos tardó en ejecutar esta línea.

```
 ; ** Source for Method '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"
 ; ** End of source for Method 'filling' **
 ;
 ; ** Source for Method '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,!
 ; ** End of source for Method 'run' **
```

El tiempo total (43,47 segundos) es 4 segundos más que durante la ejecución anterior debido a la creación de perfiles.

Reemplacemos una cosa en nuestro código de prueba, en el método de filling. Cambiaremos $Increment(^Person) a $Sequence(^Person) y volveremos a ejecutar la prueba:

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

Este resultado es sorprendente. Ok, $Sequence disminuyó el tiempo para obtener ID, pero ¿a dónde se fueron 28.5 segundos para almacenar datos en global? Verifiquemos ^%SYS.MONLBL:

```
 ; ** Source for Method '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"
 ; ** End of source for Method 'filling' **
 ;
 ; ** Source for Method '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,!
 ; ** End of source for Method 'run' **
```

Ahora, cada proceso gasta 0,2 segundos en lugar de 10 segundos para la adquisición de ID. Lo que no está claro es por qué el almacenamiento de datos lleva solo 3,23 segundos por proceso. La razón es que los nodos globales se almacenan en bloques de datos, y generalmente cada bloque tiene un tamaño de 8192 bytes. Antes de cambiar el valor del nodo global (como set ^Person(Id) = ...), el proceso bloquea todo el bloque. Si varios procesos intentan cambiar datos dentro del mismo bloque al mismo tiempo, solo un proceso podrá cambiar el bloque y otros tendrán que esperar a que termine.

Echemos un vistazo a global creado usando $Increment para generar nuevas ID. Los registros secuenciales casi nunca tendrían la misma ID de proceso (recuerde, almacenamos la ID de proceso como el primer elemento de la lista de datos):

```
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")
```

Los procesos concurrentes intentaban escribir datos en el mismo bloque y pasaban más tiempo esperando que en realidad cambiando los datos. Usando $Sequence, las ID se generan en fragmentos, por lo que los diferentes procesos probablemente usarían diferentes bloques:

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

Si esta muestra se parece a algo que está haciendo en sus proyectos, considere usar $Sequence en lugar de $Increment. Por supuesto, consulte en la documentación antes de reemplazar cada aparición de $Increment con $Sequence.

Y claro, no creas en las pruebas proporcionadas aquí: verifica esto tú mismo.

A partir de Caché 2015.2, puede configurar tablas para usar $Sequence en lugar de $Increment. No es una función del sistema $system.Sequence.SetDDLUseSequence para eso, y la misma opción está disponible en Configuración de SQL en el Portal de administración.

Además, hay un nuevo parámetro de almacenamiento en la definición de clase: IDFunction, que está configurado en "incremento" de forma predeterminada, eso significa que $Increment se usa para la generación de Id. Puede cambiarlo a "secuencia" (Inspector> Almacenamiento> Predeterminado> Función ID).

Bonus

Otra prueba rápida que realicé en mi computadora portátil: es una configuración ECP pequeña con el servidor DB ubicado en el sistema operativo host y el servidor de aplicaciones en la máquina virtual host en la misma computadora portátil. Mapeé ^Person a la base de datos remota. Es una prueba básica, por lo que no quiero hacer generalizaciones basadas en ella. Hay cosas a considerar cuando se usa $Increment y ECP. Dicho esto, aquí están los resultados:

With $Increment:

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

^%SYS.MONLBL:

```
 ; ** Source for Method '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"
 ; ** End of source for Method 'filling' **
 ;
 ; ** Source for Method '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,!
 ; ** End of source for Method 'run' **
```

$Sequence:

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

^%SYS.MONLBL

```
 ; ** Source for Method '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"
 ; ** End of source for Method 'filling' **
 ;
 ; ** Source for Method '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,!
 ; ** End of source for Method 'run' **
```

#ObjectScript #Caché

URL de fuente:https://es.community.intersystems.com/post/en-la-funci%C3%B3n-sequence