```
Artigo
```

Andre Larsen Barbosa · Ago. 9, 2021 11min de leitura

Sobre a função \$Sequence

Neste artigo, vamos comparar as funções \$Increment e \$Sequence.

Em primeiro lugar, uma nota para os leitores que nunca ouviram falar de <u>\$Increment</u>. \$Increment é uma função Caché ObjectScript que realiza uma operação atômica para incrementar seu argumento em 1 e retornar o valor resultante. Você só pode passar um nó de variável global ou local como um parâmetro para \$Increment, não uma expressão arbitrária. \$Increment é muito usado ao atribuir IDs sequenciais. Em tais casos, o parâmetro de \$Increment é geralmente um nó global. \$Increment garante que cada processo que o utiliza obtenha um ID exclusivo.

```
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)
}`
```

O problema com \$Increment é que se muitos processos estão adicionando linhas em paralelo, esses processos podem perder tempo esperando sua vez para alterar atomicamente o valor do nó global que contém o ID - ^ Person na amostra acima

[\$ Sequence] (http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...) é uma nova função que foi projetada para lidar com esse problema. \$ Sequence está disponível desde o Caché 2015.1. Assim como \$ Increment, \$ Sequence incrementa atomicamente o valor de seu parâmetro. Ao contrário de \$ Increment, \$ Sequence reservará alguns valores de contador subsequentes para o processo atual e, durante a próxima chamada no mesmo processo, simplesmente retornará o próximo valor do intervalo reservado. \$ Sequence calcula automaticamente quantos valores reservar. Mais frequentemente, processar chamadas \$ Sequence, mais valores \$ Sequence reservas:

```
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
```

Quando \$Sequence (^ myseq) retornou 9, os próximos 8 valores (até 16) já estavam reservados para o processo atual. Se outro processo chamar \$Sequence, ele obterá o valor 17, não 10.

\$Sequence é projetado para processos que incrementam simultaneamente algum nó global. Como os valores de reserva de \$Sequence, os IDs podem ter lacunas se o processo não usar todos os valores que foram reservados. O principal uso de \$Sequence é a geração de IDs sequenciais. Comparado com \$Sequence, \$Increment é uma função mais genérica.

Vamos comparar a performance de \$Increment e \$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,!
   }
   }
```

O método run jobs em 10 processos, cada um inserindo 200.000 registros em ^ Person global. Para esperar até que os processos filhos terminem. o método run tenta obter um bloqueio exclusivo em ^ P. Quando os processos filhos concluem seu trabalho e liberam o bloqueio compartilhado em ^ P, a execução adquirirá um bloqueio exclusivo em ^ P e continuará a execução. Em seguida, registramos o tempo da variável de sistema \$zhorolog e calculamos quanto tempo levou para inserir esses registros. Meu notebook multi-core com HDD lento levou 40 segundos (para ciência, eu o executei várias vezes antes, então esta foi a 5ª execução):

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

É interessante detalhar esses 40 segundos. Ao executar [& # 94;% SYS.MONLBL]

(http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...), podemos ver que um total de 100 segundos foram gastos obtendo ID. 100 segundos / 10 processos = cada processo gastou 10 segundos para adquirir um novo ID, 1,7 segundo para obter o nome e o sobrenome e 28,5 segundos para gravar dados nos dados globais.

A primeira coluna no relatório% SYS.MONLBL abaixo é o número da linha, a segunda é quantas vezes essa linha foi executada e a terceira é quantos segundos levou para executar esta linha.

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

O tempo total (43,47 segundos) é 4 segundos a mais do que durante a execução anterior devido ao perfil.

Vamos substituir algo em nosso código de teste, no método fill. Vamos mudar \$ Increment (^Person) para \$Sequence (^Person) e executar o teste novamente:

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

Este resultado é surpreendente. Ok, \$Sequence diminuiu o tempo para obter a ID, mas para onde foram 28,5 segundos para armazenar dados no global? Vamos verificar ^% SYS.MONLBL:

```
** Source for Method 'filling' **
                                  lock +^P:"S"
   1
                 10
                       .001181
   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, nam
```

```
e)
    8
             1951569
                          1.0243
                                      lock -^P:"S"
    9
                         .000219
                  10
     ;
       ** End of source for Method 'filling' **
     ;
       ** Source for Method 'run' **
     ;
                         .006514
                                     kill ^Person
    1
                   1
                   1
                         .000002
                                     set z1 = $zhorolog
    2
    3
                   1
                         .000002
                                     for i=1:1:10 {
                         .385055
                                          job ..filling()
    4
                  10
    5
                   0
                                0
    6
                        6.558119
                                      lock ^P
                   1
    7
                   1
                         .000011
                                      set z2 = \$zhorolog - z1
    8
                   1
                         .000008
                                      lock
    9
                   1
                         .000025
                                      write "done: ", z2,!
          End of source for Method 'run' **
```

Agora, cada processo gasta 0,2 segundos em vez de 10 segundos para aquisição de ID. O que não está claro é por que o armazenamento de dados leva apenas 3,23 segundos por processo? O motivo é que os nós globais são armazenados em blocos de dados e, geralmente, cada bloco tem um tamanho de 8.192 bytes. Antes de alterar o valor do nó global (como set ^Person (Id) =...), o processo bloqueia todo o bloco. Se vários processos estiverem tentando alterar dados dentro de um mesmo bloco ao mesmo tempo, apenas um processo terá permissão para alterar o bloco e os outros terão que aquardar sua conclusão.

Vejamos o global criado usando \$Increment para gerar novos IDs. Os registros sequenciais quase nunca teriam o mesmo ID do processo (lembre-se - armazenamos o ID do processo como o primeiro elemento da lista de dados):

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

Os processos simultâneos estavam tentando gravar dados no mesmo bloco e gastando mais tempo esperando do que realmente alterando os dados. Usando \$ Sequence, os IDs são gerados em blocos, portanto, processos diferentes provavelmente usariam blocos diferentes:

```
354: ^Person(100353) = $lb("12967", "Roentgen", "Barb")
```

Se este exemplo parece algo que você está fazendo em seus projetos, considere o uso de \$Sequence em vez de \$Increment. Claro, consulte a [documentação]

(http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...) antes de substituir cada ocorrência de \$Increment por \$Sequence.

E, claro, não acredite nos testes fornecidos aqui - verifique você mesmo.

A partir do Caché 2015.2, você pode configurar tabelas para usar \$Sequence em vez de \$Increment. Existe uma função de sistema [\$system.Sequence.SetDDLUseSequence] (http://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?P...) para isso, e a mesma opção está disponível em Configurações de SQL no Portal de gerenciamento.

Além disso, há um novo parâmetro de armazenamento na definição de classe - [IDFunction] (http://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?P... #PROPERTYIdFunction), que é definido como "incremento" por padrão, o que significa que \$ Increment é usado para geração de Id. Você pode alterá-lo para "sequência" (Inspetor> Armazenamento> Padrão> IDFunction).

Bônus

Outro teste rápido que realizei no meu notebook: é uma pequena configuração ECP com DB Server localizado no sistema operacional host e Application Server na VM convidada no mesmo notebook. Mapeei ^ Pessoa para banco de dados remoto. É um teste básico, então não quero fazer generalizações com base nele. Existem [coisas a serem consideradas] (http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...) ao usar \$ Increment e ECP. Dito isso, aqui estão os resultados:

```
### Com $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
                                    for i=1:1:200000 {
    3
                  10
                        .000044
    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
                        .000279
                                    lock -^P:"S"
                  10
          End of source for Method 'filling' **
       ** Source for Method 'run' **
                        .000045
                                    kill ^Person
    1
                   1
                   1
                        .000001
                                    set z1 = \$zhorolog
    2
    3
                   1
                        .000007
                                    for i=1:1:10 {
                                        job ..filling()
    4
                  10
                        .059868
    5
                   0
                              0
    6
                   1 170.342459
                                    lock ^P
    7
                        .000005
                                    set z2 = \$zhorolog - z1
```

```
8
                  1
                        .000013
                                   lock
    9
                       .000018
                                   write "done: ", z2,!
                  1
     ; ** End of source for Method 'run' **
### $Sequence:
    USER>do ##class(DC.IncSeq.Test).run()
    done:13.826716
^%SYS.MONLBL
     ; ** Source for Method 'filling' **
                 10
                       .000434
                                   lock +^P:"S"
    1
                       .000014
                                   set job = $job
    2
                 10
                                   for i=1:1:200000 {
    3
                 10
                       .000033
    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
                                   }
                       .000255
                                   lock -^P:"S"
    9
                 10
     ; ** End of source for Method 'filling' **
     ;
       ** Source for Method 'run' **
                       .000046
                                  kill ^Person
    1
                  1
    2
                  1
                       .000002
                                   set z1 = $zhorolog
    3
                       .000004
                                   for i=1:1:10 {
                  1
                 10
                        .037271
                                       job ..filling()
    5
                  0
                              0
    6
                  1
                     14.620781
                                  lock ^P
    7
                       .000005
                                  set z2 = \$zhorolog - z1
                  1
    8
                        .000013
                  1
                                   lock
                       .000016
                  1
                                   write "done: ", z2,!
     ; ** End of source for Method 'run' **
```

#ObjectScript #Caché

URL de origem: https://pt.community.intersystems.com/post/sobre-fun%C3%A7%C3%A3o-sequence