文章
姚 鑫 · 五月 7, 2021 阅读大约需 7 分钟

第三章 使用多维存储(全局变量)（三）

第三章 使用多维存储(全局变量)（三）

在全局变量中复制数据

若要将全局变量(全部或部分)的内容复制到另一个全局变量(或局部数组)中，请使用ObjectScript Merge命令。

下面的示例演示如何使用Merge命令将OldData全局变量的全部内容复制到NewData全局变量中：

```
Merge ^NewData = ^OldData
```

如果合并命令的source参数有下标，则复制该节点及其后代中的所有数据。如果Destination参数有下标，则使用目标地址作为顶级节点复制数据。例如，以下代码：

```
Merge ^NewData(1,2) = ^OldData(5,6,7)
```

将^OldData(5，6，7)及其下的所有数据复制到^NewData(1，2)。

维护全局变量内的共享计数器

大规模事务处理应用程序的一个主要并发瓶颈可能是创建唯一标识符值。例如，考虑一个订单处理应用程序，在该应用程序中，必须为每一张新发票指定一个唯一的标识号。传统的方法是维护某种计数器表。每个创建新发票的进程都会等待获取此计数器上的锁，递增其值，然后将其解锁。这可能会导致对此单个记录的激烈资源争用。

为了解决此问题，InterSystems IRIS提供了ObjectScript $INCREMENT函数。$INCREMENT自动递增全局节点的值(如果该节点没有值，则设置为1)。$INCREMENT的原子性意味着不需要锁；该函数保证返回一个新的增量值，不会受到任何其他进程的干扰。

可以使用$INCREMENT，如下所示。首先，必须决定在其中存放计数器的全局节点。接下来，无论何时需要新的计数器值，只需调用$INCREMENT：

```
SET counter = $INCREMENT(^MyCounter)
```

InterSystems IRIS对象和SQL使用的默认存储结构使用$INCREMENT来分配唯一的对象(行)标识符值。

对全局变量中的数据进行排序

存储在全局变量中的数据会根据下标的值自动排序。例如，下面的ObjectScript代码定义了一组全局变量(按随机顺序)，然后遍历它们以演示全局节点按下标自动排序：

```
/// w ##class(PHA.TEST.Global).GlobalSort()
ClassMethod GlobalSort()
{
    Kill ^Data

    Set ^Data("Cambridge") = ""
    Set ^Data("New York") = ""
    Set ^Data("Boston") = ""
    Set ^Data("London") = ""
    Set ^Data("Athens") = ""


    Set key = $Order(^Data(""))
    While (key '= "") {
        Write key,!
        Set key = $Order(^Data(key))
    }
    q ""
}


DHC-APP> w ##class(PHA.TEST.Global).GlobalSort()
Athens
Boston
Cambridge
London
New York
```

应用程序可以利用全局函数提供的自动排序来执行排序操作或维护对某些值的有序、交叉引用的索引。
InterSystems SQL和ObjectScript使用全局变量自动执行这些任务。

全局变量节点排序规则

全局变量节点的排序顺序(称为排序)在两个级别上进行控制:全局变量本身内部和使用全局变量的应用程序。

在应用程序级别，可以通过对用作下标的值执行数据转换来控制全局节点的排序方式(InterSystems SQL和对象通过用户指定的排序函数来执行此操作)。
例如，如果创建一个按字母顺序排序但忽略大小写的名称列表，那么通常你会使用名称的大写版本作为下标:

```
/// w ##class(PHA.TEST.Global).GlobalSortAlpha()
ClassMethod GlobalSortAlpha()
{
    Kill ^Data

    For name = "Cobra","jackal","zebra","AARDVark" {
        Set ^Data($ZCONVERT(name,"U")) = name
    }


    Set key = $Order(^Data(""))
    While (key '= "") {
        Write ^Data(key),!
        Set key = $Order(^Data(key))
    }
```

```
    q ""
}


DHC-APP>w ##class(PHA.TEST.Global).GlobalSortAlpha()
AARDVark
Cobra
jackal
zebra
```

此示例将每个名称转换为大写(使用$ZCONVERT函数)，以便对下标进行排序，而不考虑大小写。每个节点都包含未转换的值，以便可以显示原始值。

数值和字符串值下标

数字值在字符串值之前进行排序；也就是说，值1在值“ a ”之前。如果对给定的下标同时使用数值和字符串值，则需要注意这一点。如果将全局变量用于索引(即根据值对数据进行排序)，则最常见的是将值排序为数字(如薪水salaries)或字符串(如邮政编码postal codes)。

对于按数字排序的节点，典型的解决方案是使用一元+运算符将下标值强制为数字值。例如，如果要构建按年龄对id值进行排序的索引，则可以强制年龄始终为数字：

```
 Set ^Data(+age,id) = ""
```

如果希望将值排序为字符串(如“ 0022 ”、“ 0342 ”、“ 1584 ”)，则可以通过添加空格(“ ”)字符来强制下标值始终为字符串。例如，如果正在构建一个按邮政编码对id值进行排序的索引，则可以强制zipcode始终为字符串：

```
 Set ^Data(" "_zipcode,id) = ""
```

这确保带有前导零的值(如“ 0022 ”)始终被视为字符串。

$SORTBEGIN和$SORTEND函数

通常，不必担心在InterSystems IRIS中对数据进行排序。无论使用SQL还是直接全局访问，排序都是自动处理的。

然而，在某些情况下，可以更有效地进行排序。
具体来说，在以下情况下(1)需要设置大量随机(即未排序)的全局节点，(2)生成的全局节点的总大小接近InterSystems IRIS缓冲池的很大一部分，那么性能可能会受到不利影响-
因为很多SET操作涉及到磁盘操作(因为数据不适合缓存)。
这种情况通常出现在涉及创建索引全局函数的情况下，例如批量数据加载、索引填充或对临时全局函数中的未索引值进行排序

为了有效地处理这些情况，ObjectScript提供了$SORTBEGIN和$SORTEND函数。
$SORTBEGIN函数为全局变量(或其中的一部分)启动了一种特殊模式，在这种模式中，进入全局变量的数据集被写入一个特殊的临时缓冲区，并在内存(或临时磁盘存储)中进行排序。
当在操作结束时调用$SORTEND函数时，数据将按顺序写入实际的全局存储中。
总体操作效率更高，因为实际的写操作是按照要求更少磁盘操作的顺序完成的。

$SORTBEGIN函数很容易使用;
在开始排序操作之前，用你想要排序的全局变量的名称调用它，并在操作完成时调用$SORTEND:

```
/// w ##class(PHA.TEST.Global).GlobalSortBeginEnd()
ClassMethod GlobalSortBeginEnd()
{

    Kill ^Data

    // ?^Data?????????
    Set ret = $SortBegin(^Data)

    For i = 1:1:10000 {
        Set ^Data($Random(1000000)) = ""
    }

    Set ret = $SortEnd(^Data)

    // ^Data?????????

    Set start = $ZH
    // ???????(???)
    Set key = $Order(^Data(""))
    While (key '= "") {
        Write key,!
        Set key = $Order(^Data(key))
    }

    Set elap = $ZH - start
    Write "Time (seconds): ",elap
    q ""
}
```

$SORTBEGIN函数是为全局变量创建的特殊情况而设计的，在使用时必须小心。
特别地，在$SORTBEGIN模式下，不能从正在写入的全局变量中读取数据;
由于数据没有写入，读取将是不正确的。

InterSystems SQL自动使用这些函数创建临时全局索引(例如对未索引的字段进行排序)。

在全局变量中使用间接

通过间接方式，ObjectScript提供了一种在运行时创建全局变量引用的方法。
这对于在程序编译时不知道全局变量结构或名称的应用程序非常有用。

间接操作符@支持间接操作，它解除了对包含表达式的字符串的引用。
根据@操作符的使用方式，有几种间接类型。

下面的代码提供了一个名称间接引用的示例，在这个示例中，使用@操作符对包含全局引用的字符串进行解引用:

```
/// w ##class(PHA.TEST.Global).GlobalIndirect()
ClassMethod GlobalIndirect()
{
    Kill ^Data
    Set var = "^Data(100)"
    // ????????^Data(100)
    Set @var = "This data was set indirectly."
    // ???????:
    Write "Value: ",^Data(100)
```

```
    q ""
}


DHC-APP> w ##class(PHA.TEST.Global).GlobalIndirect()
Value: This data was set indirectly.
```

也可以使用下标间接在间接语句中混合表达式(变量或文字值):

```
/// w ##class(PHA.TEST.Global).GlobalIndirect1()
ClassMethod GlobalIndirect1()
{

    Kill ^Data
    Set glvn = "^Data"
    For i = 1:1:10 {
        Set @glvn@(i) = "This data was set indirectly."
    }

    Set key = $Order(^Data(""))
    While (key '= "") {
        Write "Value ",key, ": ", ^Data(key),!
        Set key = $Order(^Data(key))
    }
    q ""
}


DHC-APP>w ##class(PHA.TEST.Global).GlobalIndirect1()
Value 1: This data was set indirectly.
Value 2: This data was set indirectly.
Value 3: This data was set indirectly.
Value 4: This data was set indirectly.
Value 5: This data was set indirectly.
Value 6: This data was set indirectly.
Value 7: This data was set indirectly.
Value 8: This data was set indirectly.
Value 9: This data was set indirectly.
Value 10: This data was set indirectly.
```

间接是ObjectScript的一个基本特性;
它并不局限于全局引用。

#SQL #Caché #InterSystems IRIS #InterSystems IRIS for Health

源URL:
https://cn.community.intersystems.com/post/%E7%AC%AC%E4%B8%89%E7%AB%A0-%E4%BD%BF%E7%94%A8%E5%A4%9A%E7%BB%B4%E5%AD%98%E5%82%A8%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%EF%BC%88%E4%B8%89%EF%BC%89