文章
姚 鑫 · 六月 15, 2022 阅读大约需 5 分钟

第一章 锁定和并发控制制（一）

第一章 锁定和并发控制（一）

任何多进程系统的一个重要特征是并发控制，即防止不同进程同时更改特定数据元素的能力，从而导致损坏。
提供了一个锁管理系统。本文提供了一个概述。

此外，%Persistent 类提供了一种控制对象并发访问的方法，即 %OpenId()
的并发参数和该类的其他方法。这些方法最终使用本文讨论的 ObjectScript LOCK
命令。所有持久对象都继承这些方法。同样，系统会自动对 INSERT、UPDATE 和 DELETE
操作执行锁定（除非指定 %NOLOCK 关键字）。

%Persistent 类还提供方法 %GetLock()、%ReleaseLock()、%LockId()、%UnlockId()、%LockExtent() 和
%UnlockExtent()。

介绍

基本的锁定机制是 LOCK
命令。此命令的目的是延迟一个进程中的活动，直到另一个进程发出可以继续进行的信号。

锁本身并不能阻止活动行为。锁定仅按约定起作用：它要求相互竞争的进程都使用相同的锁定名称实现锁定。例如，
下面描述了一个常见的场景：

1. 进程 A 发出 LOCK 命令， 创建一个锁（默认情况下，一个独占锁）。通常，进程 A
   然后对global中的节点进行更改。详细信息是特定于应用程序的。
2. 进程 B 发出具有相同锁名称的 LOCK 命令。因为存在一个现有的排他锁，所以进程 B
   暂停。具体来说，LOCK 命令不返回，并且不能执行连续的代码行。
3. 当进程A释放锁时，进程B中的LOCK命令最终返回，进程B继续。通常，进程 B
   然后对同一global中的节点进行更改。

锁名称

LOCK 命令的参数之一是锁名称。锁名称是任意的，但按照通用约定，程序员使用与要锁定的项目名称相同的锁名称
。通常要锁定的项目是global或global的一个节点。因此锁名称通常看起来像global名称的名称或全局节点的名称。
（本文只讨论以脱字符开头的锁名称，因为这些是最常见的；有关名称不以脱字符(^)开头的锁的详细信息，请参阅
ObjectScript 参考中的 “ LOCK ” 。）

形式上，锁名称遵循与局部变量和全局变量相同的命名约定，如使用 ObjectScript 中的 “ 变量 ” 一章所述。与变量一样
，锁名称区分大小写并且可以有下标。不要使用进程私有的global名称作为锁名称（无论如何都不需要这样的锁，因
为根据定义，只有一个进程可以访问这样的全局）。

提示：由于锁定按约定工作并且锁定名称是任意的，因此无需在创建具有相同名称的锁定之前定义给定变量。

由于分配和管理内存的方式，锁名称的形式会影响性能。锁定针对使用下标的锁定名称进行了优化。一个例子是
^sample.person(id)。

相反并未针对锁名称进行优化，例如 ^nameconcatenatedidentifier。非下标锁名称也可能导致与 ECP
相关的性能问题。

锁表

维护系统范围的内存表，记录所有当前锁和拥有它们的进程。此表（锁定表）可通过管理门户访问，可以在其中查看锁定并（在极少数情况下，如果需要）删除它们。请注意，任何给定的进程都可以拥有多个具有不同锁名称的锁（甚至可以拥有多个具有相同锁名称的锁）。

当一个进程结束时，系统会自动释放该进程拥有的所有锁。因此，通常不需要通过管理门户移除锁，除非出现应用程序错误。

锁定表不能超过固定大小，可以使用 locksiz
设置指定该大小。因此，锁表可能会被填满，这样就不可能再有锁了。如果发生这种情况，会将以下消息写入 messages.log 文件：

LOCK TABLE FULL

填充锁表一般不认为是应用程序错误；IRIS 还提供了一个锁队列，进程等待直到有空间将它们的锁添加到锁表中。（但是，死锁被认为是应用程序编程错误。请参阅本文后面的“ 避免死锁 ”。）

锁和阵列

锁定阵列时，可以锁定整个阵列或阵列中的一个或多个节点。锁定阵列节点时，会阻止其他进程锁定从属于该节点的任何节点。其他进程也被阻止锁定锁定节点的直接祖先。

下图显示了一个示例：

隐式锁不包含在锁表中，因此不会影响锁表的大小。

锁排队算法按接收到的顺序将相同锁名的所有锁排队，即使没有直接的资源争用。

使用 LOCK 命令

本节讨论如何使用 LOCK 命令添加和删除锁。

添加增量锁

要添加锁，请使用 LOCK 命令，如下所示：

```
LOCK +lockname
```

其中 lockname 是文字锁名称。加号（+）创建增量锁，这是常见的场景；

该命令执行以下操作：

1. 尝试将给定的锁添加到锁表中。也就是说，这个条目被添加到锁队列中。
2. 暂停执行，直到可以获取锁为止。

有不同类型的锁，它们的行为不同。要添加非默认锁类型的锁，请使用以下变体：

```
LOCK +lockname#locktype
```

其中 locktype 是用双引号括起来的一串锁类型代码；

注意一个给定的进程可以添加多个同名的增量锁；这些锁可以是不同的类型，也可以是相同的类型。

添加具有超时的增量锁

如果使用不当，增量锁可能会导致称为死锁的不良情况，稍后将在“ 避免死锁 ”中讨论。避免死锁的一种方法是在创建锁时指定超时时间。为此，请按如下方式使用 LOCK 命令：

```
LOCK +lockname#locktype :timeout
```

其中 timeout 是以秒为单位的超时时间。冒号前的空格是可选的。如果将超时指定为 0，
会尝试添加锁（但请参阅下面的注释）。

该命令执行以下操作：

1. 尝试将给定的锁添加到锁表中。也就是说，这个条目被添加到锁队列中。
2. 暂停执行，直到可以获取锁或超时期限结束，以先到者为准。
3. 设置 $TEST 特殊变量的值。如果获得锁，将 $TEST 设置为 1。否则，将$TEST 设置为 0。

这意味着如果使用 timeout 参数，代码接下来应该检查 $TEST
特殊变量的值并使用该值来选择是否继续。下面显示了一个示例：

```
Lock +^ROUTINE(routinename):0
If '$TEST {  Return $$$ERROR("Cannot lock the routine: ",routinename)}
```

#SQL #Caché

源 URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E4%B8%80%E7%AB%A0-%E9%94%81%E5%AE%9A%E5%92%8C%E5%B9%B6%E5%8F%91%E6%8E%A7%E5%88%B6%E5%88%B6%EF%BC%88%E4%B8%80%EF%BC%89