文章
姚 鑫 · 六月 27, 2022 阅读大约需 7 分钟

第十二章 信号（二）- 生产者消费者示例

第十二章 信号（二）- 生产者消费者示例

下面是一系列使用信号量实现生产者/消费者场景的类。“ 主 ”进程初始化信号量并等待用户指示活动已全部完成。生产者在循环中随机增加一个信号量值，更新之间的延迟可变。消费者尝试在随机时间从信号量中删除随机数量，也是在循环中。该示例由 5 个类组成：
- Main – 初始化环境并等待信号量上的活动完成的类。
- Counter – 实现信号量本身的类。它记录它的创建以及由于信号量在等待列表中而发生的任何回调。
- Producer – 一个类，其主要方法增加信号量值。增量是一个随机选择的小整数。完成增量后，该方法会在下一个增量之前延迟一小段随机数秒。
- Consumer 消费者——这是对生产者的补充。此类的主要方法尝试将信号量减少一个随机选择的小整数。它将递减请求添加到其等待列表中，等待时间也是随机选择的秒数。
- Util - 这个类有几个方法被示例的其他类使用。几种方法解决了为所有活动维护公共日志的问题；其他人解决了多个消费者和多个生产者的命名问题。

注意：组成这些类的代码特意写得简单。尽可能地，每个语句只完成一个动作。这应该使用户更容易和更直接地修改示例。

Class: Semaphore.Main

此类建立演示环境。它调用实用程序类来初始化日志和名称索引工具。然后它用初始值 0
初始化公共信号量，并等待用户输入一个字符（通常是 ENTER 键），表明实验已经完成。

一旦它接收到用户输入，它就会报告信号量的当前值，尝试删除它，并终止执行。

```
Class Semaphore.Main Extends %RegisteredObject
{

/// ????????
Parameter ME = "Main";

/// ??????????
ClassMethod Run()
{
    // ??????????
    d ##class(Semaphore.Util).InitLog()
    d ##class(Semaphore.Util).InitIndex()

    s msg = ..#ME _ " ??"
    d ..Log(msg)

    // ?????????
    s inventory = ##class(Semaphore.Counter).%New()
    if ('$isobject(inventory)) {
        s msg = "%New() of MySem failed"
        d ..Log(msg)
        q
```

```
    }

    s msg = "??????: " _ inventory.Init(0)
    d ..Log(msg)

    // ??????
    s msg = "????????? Run ??"
    d ..Log(msg)

    r *x

    // ?????????????
    s msg = "?? = " _ inventory.GetValue()
    d ..Log(msg)
    s msg = "???????: " _ inventory.Delete()
    d ..Log(msg)
    s msg = ..#ME _ " ??"
    d ..Log(msg)

    q
}

/// ?????????????
ClassMethod Log(msg As %String) [ Private ]
{
    d ##class(Semaphore.Util).Logger(..#ME, msg)
    q
}

}
```

Class: Semaphore.Counter

此类实现示例中使用的信号量。根据需要，它是 %SYSTEM.Semaphore 的子类，并提供方法 WaitCompleted 的实现。为了简单起见，初始化信号量的代码也包含在这个类中。还有一个类方法提供此信号量的名称，以允许设置、生产者和消费者类获取它。

```
Class Semaphore.Counter Extends %SYSTEM.Semaphore
{

ClassMethod Name() As %String
{
    Quit "Counter"
}

/// ??????????
Method Log(Msg As %String) [ Private ]
{
    d ##class(Semaphore.Util).Logger(..Name(), Msg)
    q
}

/// ???? id ???????????
Method MyId() As %String
{
   q ("0x" _ $zhex(..SemID))
```

```
}

/// ???????
Method %OnNew() As %Status
{
    s msg = "???"
    d ..Log(msg)
    q $$$OK
}

Method Init(initvalue = 0) As %Status
{
    try {
        if (..Create(..Name(), initvalue)) {
            s msg = "?? """ _ ..Name()
                    _ """; ?? = " _ initvalue
                    _ "; Id = 0x" _ ..MyId()
            d ..Log(msg)
            ret 1
        } else {
            s msg = "??????: Name = """ _ ..Name() _ """"
            d ..Log(msg)
            ret 0
        }
    } catch {
        s msg = "???????"
        d ..Log(msg)
        ret 0
    }
}

Method %OnClose() As %Status [ Private ]
{
    s msg = "?????: Id = " _ ..MyId()
    d ..Log(msg)
    q $$$OK
}

/// ???? WaitMany() ????????????????????
/// ????????????????????????
///
/// ??????????????????
/// ?????? AddToWaitMany ???????????
Method WaitCompleted(amt As %Integer)
{
    // ???????
    s msg = "WaitCompleted: " _ ..MyId() _ "; Amt = " _ amt
    d ..Log(msg)
    q
}

}
```

Class: Semaphore.Producer

此类负责获取公共信号量的 OREF。一旦它拥有了OREF，它就会尝试将信号量重复增加一个随机选择的小整数，并在每次增量之间暂停一个小的随机选择间隔。每次增加信号量的尝试都会输入到日志中。

```
Class Semaphore.Producer Extends %RegisteredObject
{

/// ??
Parameter MeBase = "Producer";

/// ????????????
ClassMethod Run() As %Status
{
    // ?????????
    s ME = ##class(Semaphore.Util).IndexName(..#MeBase)
    s msg = ME _ " ??"
    d ..Logger(ME, msg)

    s cell = ##class(Semaphore.Counter).%New()
    d cell.Open(##class(Semaphore.Counter).Name())

    s msg = "open Id = " _ cell.MyId()
    d ..Logger(ME, msg)

    // ?????????????
    for addcnt = 1 : 1 : 8 {
        s incamt = $random(5) + 1
        s waitsec = $random(10) + 1
        s msg = "increment " _ cell.MyId()
                _ " = " _ cell.GetValue()
                _ " by " _ incamt
                _ " wait " _ waitsec _ " sec"
        d cell.Increment(incamt)
        d ..Logger(ME, msg)
        h waitsec
    }

    // ??
    s msg = ME _ " ??"
    d ..Logger(ME, msg)

    q $$$OK
}

/// ??????????
ClassMethod Logger(id As %String, msg As %String) [ Private ]
{
    d ##class(Semaphore.Util).Logger(id, msg)
    q
}

}
```

Class: Semaphore.Consumer

这个类是对 Semaphore.Producer 的补充。它也获得了公共信号量的 OREF，并以与 Producer 类似的方式尝试将信号量重复减少随机选择的数量，并在每次尝试之间随机选择暂停。每次尝试的成功或失败都会写入日志。

```
Class Semaphore.Consumer Extends %RegisteredObject
{

/// ??
Parameter MeBase = "Consumer";

/// ?????????????
ClassMethod Run() As %Status
{
    // ??????????
    s ME = ##class(Semaphore.Util).IndexName(..#MeBase)
    s msg = ME _ " ??"
    d ..Logger(ME, msg)

    s cell = ##class(Semaphore.Counter).%New()
    d cell.Open(##class(Semaphore.Counter).Name())
    s msg = "Consumer: Open Id = " _ cell.MyId()
    d ..Logger(ME, msg)

    // ????????????????????
    for deccnt = 1 : 1 : 15 {
        s decamt = $RANDOM(5) + 1
        s waitsec = $RANDOM(10) + 1
        s msg = "Decrement " _ cell.MyId()
                _ " = " _ cell.GetValue()
                _ " by " _ decamt
                _ " wait " _ waitsec _ " sec"
        // ????????????????????????????????????? 200?
        d cell.AddToWaitMany(decamt)
        d ..Logger(ME, msg)
        s result = ##class(%SYSTEM.Semaphore).WaitMany(waitsec)
        s msg = $select((result > 0) : "??", 1 : "??")
        d ..Logger(ME, msg)

    }

    // ??
    s msg = ME _ " ??"
    d ..Logger(ME, msg)

    q $$$OK
}

/// ???????????
ClassMethod Logger(id As %String, msg As %String) [ Private ]
{
    d ##class(Semaphore.Util).Logger(id, msg)
    q
}

}
```

Class: Semaphore.Util

此类包含解决与此示例相关的两个问题的方法。第一个是保存记录消息所需的结构的初始化，以及归档提交到日志的消息及其后续显示的方法。

第二组方法处理生成编号序列的名称以识别生产者和消费者。这不是严格需要的，因为 $JOB 命令提供的进程 ID 也这样做，但使用更易于阅读的标签更容易。

```
Class Semaphore.Util Extends %RegisteredObject
{

/// ????????
Parameter ME = "Util";

/// ???????
ClassMethod InitLog()
{
    // ?????????
    k ^SemaphoreLog
    s ^SemaphoreLog = 0
    q
}

/// ??????????????????
///
ClassMethod Logger(sender As %String, msg As %String)
{
    s inx = $i(^SemaphoreLog)
    s ^SemaphoreLog(inx, 0) = $job
    s ^SemaphoreLog(inx, 1) = sender
    s ^SemaphoreLog(inx, 2) = msg
    w "(", ^SemaphoreLog, ") ", msg, !
    q
}

/// ????????
ClassMethod ShowLog()
{
    s msgcnt = $g(^SemaphoreLog, 0)
    w "??????? = ", msgcnt, !, !
    w "#", ?5, "$Job", ?12, "Sender", ?25, "Message", !

    for i = 1 : 1 : msgcnt {
        s job = ^SemaphoreLog(i, 0)
        s sender = ^SemaphoreLog(i, 1)
        s msg = ^SemaphoreLog(i, 2)
        w i, ")", ?5, job, ?12, sender, ":", ?25, msg, !
    }
    q
}

/// ???????
ClassMethod InitIndex()
{
    k ^SemaphoreNames
    q
}

/// ???????
ClassMethod IndexName(name As %String) As %String
{
    if ($d(^SemaphoreNames(name)) = 0) {
        s ^SemaphoreNames(name) = 0
```

```
    }
    s index =  $i(^SemaphoreNames(name))
    q (name _ "." _ index)
}

}
```

#SQL #Caché

源URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E5%8D%81%E4%BA%8C%E7%AB%A0-%E4%BF%A1%E5%8F%B7%EF%BC%88%E4%BA%8C%EF%BC%89-%E7%94%9F%E4%BA%A7%E8%80%85%E6%B6%88%E8%B4%B9%E8%80%85%E7%A4%BA%E4%BE%8B