文章
姚 鑫 · 五月 9, 2021 阅读大约需 分钟

第四章 多维存储的SQL和对象使用(一)

第四章 多维存储的SQL和对象使用(一)

本章介绍InterSystems IRIS®对象和SQL引擎如何利用多维存储(全局变量)来存储持久对象、关系表和索引。

尽管InterSystems
IRIS对象和SQL引擎会自动提供和管理数据存储结构，但了解其工作原理的详细信息还是很有用的。

数据的对象视图和关系视图使用的存储结构是相同的。为简单起见，本章仅从对象角度介绍存储。

数据

每个使用%Storage.Persistent存储类(默认)的持久化类都可以使用多维存储(全局变量)的一个或多个节点在InterSystems IRIS数据库中存储其自身的实例。

每个持久化类都有一个存储定义，用于定义其属性如何存储在全局变量节点中。这个存储定义(称为“默认结构”)由类编译器自动管理。

默认结构

用于存储持久对象的默认结构非常简单：
-
数据存储在名称以完整类名(包括包名)开头的全局变量中。附加“D”以形成全局数据的名称，而附加“I”作为全局索引。
- 每个实例的数据都存储在全局数据的单个节点中，所有非瞬态属性都放在$list结构中。
- 数据全局变量中的每个节点都以对象ID值作为下标。默认情况下，对象ID值是通过调用存储在全局变量数据根(没有下标)的计数器节点上的$Increment函数提供的整数。

例如，假设我们定义了一个简单的持久化类MyApp.Person，它有两个文本属性

```
Class MyApp.Person Extends %Persistent
{
Property Name As %String;
Property Age As %Integer;
}
```

如果我们创建并保存此类的两个实例，得到的全局变量结果将类似于：

```
^MyApp.PersonD = 2  // counter node
^MyApp.PersonD(1) = $LB("",530,"Abraham")
^MyApp.PersonD(2) = $LB("",680,"Philip")
```

注意，存储在每个节点中的$List结构的第一部分是空的；
这是为类名留的。
如果定义Person类的子类，则此槽包含子类名。
当多个对象存储在同一个区段内时，%OpenId方法(由%Persistent类提供)使用此信息动态地打开正确的对象类型。
此槽在类存储定义中显示为名为"%%CLASSNAME"的属性

IDKEY

IDKEY机制允许显式定义用作对象ID的值。为此，只需将IDKEY索引定义添加到类中，并指定将提供ID值的一个或多个属性。请注意，一旦保存对象，其对象ID值就不能更改。这意味着在保存使用IDKEY机制的对象后，不能再修改对象ID所基于的任何特性。

```
Class MyApp.Person Extends %Persistent
{
Index IDKEY On Name [ Idkey ];

Property Name As %String;
Property Age As %Integer;
}
```

如果我们创建并保存Person类的两个实例，得到的全局变量结果现在类似于：

```
 ^MyApp.PersonD("Abraham") = $LB("",530,"Abraham")
 ^MyApp.PersonD("Philip") = $LB("",680,"Philip")
```

请注意，不再定义任何计数器节点。还要注意，通过将对象ID基于Name属性，我们已经暗示了Name的值对于每个对象必须是唯一的。

如果IDKEY索引基于多个属性，则主数据节点具有多个下标。例如：

```
Class MyApp.Person Extends %Persistent
{
Index IDKEY On (Name,Age) [ Idkey ];

Property Name As %String;
Property Age As %Integer;
}
```

在这种情况下，生成的全局变量现在类似于：

```
 ^MyApp.PersonD("Abraham",530) = $LB("",530,"Abraham")
 ^MyApp.PersonD("Philip",680) = $LB("",680,"Philip")
```

重要提示：IDKEY索引使用的任何属性值中都不能有连续的一对竖线(||)，除非该属性值对持久类实例的有效引用。
这种限制是由InterSystems SQL机制的工作方式强加的。
在IDKey属性值使用||会导致不可预知的行为。

Subclasses

默认情况下，持久性对象的子类引入的任何字段都存储在附加节点中。
子类的名称用作附加的标值。

例如，假设我们定义了一个具有两个文本属性的简单持久MyApp.Person类：

```
Class MyApp.Person Extends %Persistent
{
Property Name As %String;

Property Age As %Integer;
}
```

现在，我们定义了一个持久子类MyApp.Students，它引入了两个额外的文本属性

```
Class MyApp.Student Extends Person
{
Property Major As %String;

Property GPA As %Double;
}
```

如果我们创建并保存此MyApp.Student类的两个实例，得到的全局结果将类似于：

```
^MyApp.PersonD = 2  // counter node
^MyApp.PersonD(1) = $LB("Student",19,"Jack")
^MyApp.PersonD(1,"Student") = $LB(3.2,"Physics")

^MyApp.PersonD(2) = $LB("Student",20,"Jill")
^MyApp.PersonD(2,"Student") = $LB(3.8,"Chemistry")
```

从Person类继承的属性存储在主节点中，而由Student类引入的属性存储在另一个子节点中。这种结构确保生数据可以作为人员数据互换使用。例如，列出所有Person对象名称的SQL查询正确地获取Person和Student数据。当属性被添加到超类或子类时，这种结构还使类编译器更容易维护数据兼容性

请 注 意，主节点的第一部分包含字符串“Student”-它标识包含学生数据的节点。

父子关系

在父子关系中，子对象的实例存储为它们所属的父对象的子节点。这种结构确保实例数据与父数据在物理上是集群的。

```
/// An Invoice class
Class MyApp.Invoice Extends %Persistent
{
Property CustomerName As %String;

/// an Invoice has CHILDREN that are LineItems
Relationship Items As LineItem  [inverse = TheInvoice, cardinality = CHILDREN];
}
```

和LineItem：

```
/// A LineItem class
Class MyApp.LineItem Extends %Persistent
{
Property Product As %String;
Property Quantity As %Integer;

/// a LineItem has a PARENT that is an Invoice
Relationship TheInvoice As Invoice [inverse = Items, cardinality = PARENT];
}
```

如果我们存储一个Invoice对象的实例，每个实例都有关联的LineItem对象，则得到的全局变量结果将类似于：

```
^MyApp.InvoiceD = 2  // invoice counter node
^MyApp.InvoiceD(1) = $LB("","Wiley Coyote")
^MyApp.InvoiceD(1,"Items",1) = $LB("","Rocket Roller Skates",2)
^MyApp.InvoiceD(1,"Items",2) = $LB("","Acme Magnet",1)

^MyApp.InvoiceD(2) = $LB("","Road Runner")
^MyApp.InvoiceD(2,"Items",1) = $LB("","Birdseed",30)
```

嵌入对象

存储嵌入对象的方法是先将它们转换为序列化状态(默认情况下是包含对象属性的List结构)，然后以与任何其他属性相同的方式存储此串行状态。

例如，假设我们定义了一个具有两个文字属性的简单串行(可嵌入)类：

```
Class MyApp.MyAddress Extends %SerialObject
{
Property City As %String;
Property State As %String;
}
```

现在，我们将前面的示例以添加嵌入的Home Address属性

```
Class MyApp.MyClass Extends %Persistent
{
Property Name As %String;
Property Age As %Integer;
Property Home As MyAddress;
}
```

如果我们创建并保存此类的两个实例，则生成的全局变量相当于：

```
 ^MyApp.MyClassD = 2  // counter node
 ^MyApp.MyClassD(1) = $LB(530,"Abraham",$LB("UR","Mesopotamia"))
 ^MyApp.MyClassD(2) = $LB(680,"Philip",$LB("Bethsaida","Israel"))
```

流

通过将全流数据拆分成系列块(每个块小于32K字节)并将这些块写入一系列顺序节点，全流被存储在全流中。文件流存储在外部文件中。

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

源 URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E5%9B%9B%E7%AB%A0-%E5%A4%9A%E7%BB%B4%E5%AD%98%E5%82%A8%E7%9A%84sql%E5%92%8C%E5%AF%B9%E8%B1%A1%E4%BD%BF%E7%94%A8%EF%BC%88%E4%B8%80%EF%BC%89