

使用DaoDataModel模块处理数据库

傅利民(phoolimin@gmail.com)

September 19, 2009

DaoDataModel 是一个可以将道类映射到数据库表 并对数据库进行操作的模块（目前此模块只支持MYSQL数据库）。通过建立道类到数据库表的对应关系，数据库操作变得很简单。

例如，如果有以下类，

```
load DaoDataModel;
# class that can be mapped to a database table
class Gene
{
  my id : INT_PRIMARY_KEY_AUTO_INCREMENT;
  my name : VARCHAR100;
  my seq : TEXT;
}
```

这里类型INT_PRIMARY_KEY_AUTO_INCREMENT,VARCHAR100,TEXT 由DaoDataModel模块定义，以便处理道语言类型到数据库储存类型之间的 数据转换。这些类型均被定义为道语言内置类型的别名：

- **int** 整型作为整数域：

```
INT
TINYINT
SMALLINT
MEDIUMINT
INT_PRIMARY_KEY
INT_PRIMARY_KEY_AUTO_INCREMENT
```

- **string** 字符串类型作为字符域：

```
CHAR10
CHAR20
CHAR50
```

```
CHAR100
CHAR200
VARCHAR10
VARCHAR20
VARCHAR50
VARCHAR100
VARCHAR200
```

- **string** 字符串类型作为文本或二进制数据域:

```
TEXT
MEDIUMTEXT
LONGTEXT
BLOB
MEDIUMBLOB
LONGBLOB
```

1 连接数据库

对应MySQL数据库，可以按下面的方式连接，

```
# connect database
model = DataModel( '数据库名', '主机名', '用户名', '密码' );
```

此函数的原型是，

```
DataModel( name : string, host='', user='', pwd='' )=>DataModel
```

2 创建数据表

如果更类Gene相关联的数据表还不存在，可以按以下方式创建，

```
# create a table for class Gene
model.CreateTable( Gene );
```

一个名叫Gene将被创建。

此方法的原型是，

```
DataModel.CreateTable( klass )
```

如果类含有一个名为`__TABLE_NAME__`的特殊的字符串常量，那么与此类对应的数据库表的名字将是字符串`__TABLE_NAME__`的值。类还可以含有另一个特殊的字符串常量，名为`__TABLE_PROPERTY__`，它可以包含数据库表的一些额外信息，如某些域的限定（如UNIQUE等）。

3 插入记录

如要在Gene表中插入一个或多个记录，只需将Gene的实例直接插入到模型中即可，

```
gene = Gene{ 0, 'RAS', '...AATCCG...' };  
# insert a record into the table  
model.Insert( gene );
```

此方法知道应该在那个数据表里插入记录。当类实例被成功插入时，如果此类包含一个类型为`INT PRIMARY KEY AUTO INCREMENT`的成员，那么那些被插入的实例的此成员值将被设为相应记录的序号，在上面的例子里，`gene.id`将被设为与实例`gene`对应的记录的序号。

要一次插入多个记录，只需要将一个类实例列表作为参数传递给`Insert()`方法即可。此方法将返回一个数据操作手柄，使用此手柄，可继续插入多个记录，同时避免反复编译同样的SQL语言。

类似的，还有`Delete()`方法，

```
DataModel.Insert( object ) => Handler  
DataModel.Delete( object ) => Handler
```

4 数据库查询

查找名为RAS的基因，

```
# SELECT * FROM Gene WHERE name = 'RAS';  
hd = model.Select( Gene ).Where().EQ( 'name', 'RAS' );  
# query and store the result into 'gene' object:  
hd.QueryOnce( gene );
```

此模块使用一系列函数调用来构造SQL查询语句。首先需要针对查询类型使用

以下函数之一，

```
DataModel.Select( object, ... ) => Handler  
DataModel.Update( object, ... ) => Handler
```

这两个函数均以一个或多个道语言类为参数，那么查询将在这些类所对应的数据库表中进行。在它们的参数列表里，每个类后面还可跟一个整数参数N，表示查询仅限于该类/表的前面N个域。例如

```
hd = model.Select( Gene, 2 ).Where().EQ( 'name', 'RAS' );
```

这将生成如下SQL语句，

```
# SELECT id,name FROM Gene WHERE name = 'RAS';
```

然后，前面生成的手柄可用来准备查询的SQL语句，

```
# WHERE  
Handler.Where( ) => Handler  
# SET field=value, or, SET field=?  
Handler.Set( field : string, value=nil ) => Handler  
# SET field=field+value, or, SET field=field+?  
Handler.Add( field : string, value=nil ) => Handler  
# field=value, or, field=?  
Handler.EQ( field : string, value=nil ) => Handler  
# field!=value, or, field!=?  
Handler.NE( field : string, value=nil ) => Handler  
Handler.GT( field : string, value=nil ) => Handler  
Handler.GE( field : string, value=nil ) => Handler  
Handler.LT( field : string, value=nil ) => Handler  
Handler.LE( field : string, value=nil ) => Handler  
# SET table.field=value, or, SET table.field=?  
Handler.Set( table, field : string, value=nil ) => Handler  
Handler.Add( table, field : string, value=nil ) => Handler  
Handler.EQ( table, field : string, value=nil ) => Handler  
Handler.NE( table, field : string, value=nil ) => Handler  
Handler.GT( table, field : string, value=nil ) => Handler  
Handler.GE( table, field : string, value=nil ) => Handler  
Handler.LT( table, field : string, value=nil ) => Handler  
Handler.LE( table, field : string, value=nil ) => Handler  
# field IN ( values ), or, field IN ?  
Handler.In( field : string, values={} ) => Handler  
Handler.In( table, field : string, values={} ) => Handler  
# OR  
Handler.Or( ) => Handler  
Handler.And( ) => Handler  
Handler.Not( ) => Handler
```

```

# (
Handler.LBrace( ) => Handler
# )
Handler.RBrace( ) => Handler
# table1.field1=table2.field2
Handler.Match( table1, table2, field1='', field2='' ) => Handler
# ORDER BY field ASC/DESC
Handler.Sort( field : string, desc=0 ) => Handler
# ORDER BY table.field ASC/DESC
Handler.Sort( table, field : string, desc=0 ) => Handler
# LIMIT limit, or, LIMIT limit OFFSET offset
Handler.Range( limit : int, offset=0 ) => Handler

```

对于有可选参数value的函数，如果此参数被省略，那么一个占位变量（名称?）将被用在SQL语句的相应位置。在查询前，需要使用下面的函数将查询用的数据捆绑到占位变量上，

```

Handler.Bind( value, index=0 ) => Handler

```

调用此函数时，可使用参数index来指定数据应该被绑定到哪个占位变量。如果不用此参数，数据将被按顺序绑定到相应的占位变量。

最后查询时，需使用下面的方法之一，

```

Handler.Query( ... ) => int
Handler.QueryOnce( ... ) => int

```

此二方法以类实例为参数，当查询成功时，结果数据将被存在类实例的成员变量里，并返回1，否则返回0。如果一个查询命中多个记录，可反复调用Query()获取结果数据。调用Query()之后还应调用Handler.Done()重置模型，如果调用的是QueryOnce()则不必再调用Handler.Done()。

5 Other Methods

```

DataModel.Query( sql : string ) => int

```

执行任意查询，返回查询状况。

```

Handler.sqlstring( ) => string

```

返回手柄的构造的SQL语句。

```
Handler.Insert( object ) => int
```

如果手柄是DataModel.Insert()调用生成的，此手柄可被用来插入更多的记录