**.NET数据库编程-高端的实体EF**

.NET框架里面的数据库实体框架Entity Frammwork（EF）具有强大的数据库访问功能。EF通过创建数据库实体，简称实体，可以让用户像操作对象属性一样去查询、插入、删除、修改数据库中相对应的表的记录。

* **什么是实体？**

实体是EF通过映射机制将数据库中的真实物理数据表Table转换为客户端中的强类型对象。实际上，当我们创建这个强类型的对象的时候，EF已经为我们自动映射了该对象绑定的数据库中的Table的字段为该对象的属性。这样一来，我们便可以修改该对象的属性，然后提交修改操作来将修改后的操作保存到数据库中。同样，我们可以通过删除该对象，进而删除Table中对应的记录。

* **如何创建实体？**

Visual Stdio IDE具有强大的实体创建功能，我们可以通过以下步骤来创建实体。

**NOTE:**

**本教程的实体创建是基于现有的数据库表的，对于创建未有的数据库表的实体，可以采用.NET的领域先行编程，即先创建实体，再生成新的数据库。**

* 创建一个类库项目。

* 对于XmlSerializer，只有公有字段和被属性访问的私有字段（保存的是属性）才能被序列化。
* **使用BinaryFormatter序列化**
* 序列化

//导入命名空间System.Runtime.Serialization.Formatters.Binary

//引用程序集mscorlib.dll

//指定持久化对象的文件

FileInfo file = new FileInfo("user.dat");

if (!file.Exists)

{

file.Create();

}

//打开文件流

using (FileStream fstream = new FileStream("user.dat", FileMode.Open,

FileAccess.ReadWrite, FileShare.Read))

{

//以二进制格式序列化

BinaryFormatter binFormat = new BinaryFormatter();

//序列化

binFormat.Serialize(fstream, stu);

}

* 反序列化

FileInfo file = new FileInfo(@"user.dat");

if (file.Exists)

{

//打开流

using (FileStream fstream = file.Open(FileMode.Open,

FileAccess.Read, FileShare.ReadWrite))

{

BinaryFormatter binFormat = new BinaryFormatter();

//反序列化

//强制将Object类型转化为Student

stu = (Student)binFormat.Deserialize(fstream);

}

}

* **使用SoapFormatter序列化**
* 序列化

//导入命名空间System.Runtime.Serialization.Formatters.Soap

//引用程序集System.Runtime.Serialization.Formatters.Soap.dll

//指定持久化对象的文件

FileInfo file = new FileInfo("user.dat");

if (!file.Exists)

{

file.Create();

}

//打开文件流

using (FileStream fstream = new FileStream("user.dat", FileMode.Open,

FileAccess.ReadWrite, FileShare.Read))

{

//以基于网络传输的xml格式序列化

SoapFormatter soapFormat = new SoapFormatter();

//序列化

soapFormat.Serialize(fstream, stu);

}

* 反序列化

FileInfo file = new FileInfo(@"user.dat");

if (file.Exists)

{

//打开流

using (FileStream fstream = file.Open(FileMode.Open,

FileAccess.Read, FileShare.ReadWrite))

{

SoapFormatter soapFormat = new SoapFormatter();

//反序列化

//强制将Object类型转化为Student

stu = (Student)soapFormat.Deserialize(fstream);

}

}

* **使用XmlSerializer序列化**
* 序列化

//导入命名空间System.Xml.Serialization.XmlSerializer

//引用程序集System.Xml.dll

//指定持久化对象的文件

FileInfo file = new FileInfo("user.dat");

if (!file.Exists)

{

file.Create();

}

//打开文件流

using (FileStream fstream = new FileStream("user.dat", FileMode.Open,

FileAccess.ReadWrite, FileShare.Read))

{

//以纯xml格式序列化

XmlSerializer xmlFormat = new XmlSerializer(stu.GetType());

//序列化

XmlSerializer.Serialize(fstream, stu);

}

* 反序列化

FileInfo file = new FileInfo(@"user.dat");

if (file.Exists)

{

//打开流

using (FileStream fstream = file.Open(FileMode.Open,

FileAccess.Read, FileShare.ReadWrite))

{

XmlSerializer xmlFormat = new XmlSerializer(stu.GetType());

//反序列化

//强制将Object类型转化为Student

stu = (Student) xmlFormat.Deserialize(fstream);

}

}

* **如何控制生成的xml数据？**

这里的控制是说，将可序列化对象的字段保存在xml文件里面的形式，可以是xml中保存对象的子元素，也可以是对象的特性。默认情况下是以子元素形式保存。

<?xml version="1.0"?>

<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<sex>0</sex>

<Age>0</Age>

</Student>

* 指定字段以xml特性保存

[XmlAttribute]//以特性序列化在xml中

public char sex;

<?xml version="1.0"?>

<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" sex="0">

<Age>0</Age>

</Student>

* **如何保存多个可序列化对象？**

将多个对象保在容器如数组、ArrayList和List<T>中即可。

//声明容器

private List<Student> stus = new List<Student>();;

* 序列化

//导入命名空间System.Xml.Serialization.XmlSerializer

//引用程序集System.Xml.dll

stus.Add(new Student());

stus.Add(new Student());

//指定持久化对象的文件

FileInfo file = new FileInfo("user.dat");

if (!file.Exists)

{

file.Create();

}n

//打开文件流

using (FileStream fstream = new FileStream("user.dat", FileMode.Open,

FileAccess.ReadWrite, FileShare.Read))

{

//以纯xml格式序列化

XmlSerializer xmlFormat = new XmlSerializer(stus.GetType());

//序列化

XmlSerializer.Serialize(fstream, stus);

}

* 反序列化

FileInfo file = new FileInfo(@"user.dat");

if (file.Exists)

{

//打开流

using (FileStream fstream = file.Open(FileMode.Open,

FileAccess.Read, FileShare.ReadWrite))

{

XmlSerializer xmlFormat = new XmlSerializer(stus.GetType());

//反序列化

//强制将Object类型转化为List<Student>

stus = (List<Student>) xmlFormat.Deserialize(fstream);

}

}

<?xml version="1.0"?>

<ArrayOfStudent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Student sex="0">

<Age>0</Age>

</Student>

<Student sex="0">

<Age>0</Age>

</Student>

</ArrayOfStudent>

* **如何自定义BinaryFormatter、SoapFormatter序列化过程？**

我们可以自定义BinaryFormatter、SoapFormatter序列化过程，使得序列化对象的字段值在序列化过程中得到改变、或者增加其他数据。比如，在序列化时将字符串转化为大写，在反序列化时将字符串转化为小写。

* 使用ISerializable自定义序列化

该方法用于.NET2.0之前的版本自定义序列化。以下是自定义序列化步骤：

* 定义的自定义可序列化类型需要实现接口Iserializable

//定义一个可自定义序列化的类

//继承接口ISerializable

[Serializable]

public class Student : ISerializable

{

//

}

* 构造特殊的构造函数（反序列化过程）

//一定要构造默认构造函数

public Student(){}

//特殊的构造函数

//必须是protected

//SerializationInfo存放需要自定义序列化的字段信息

//自定义反序列化过程

protected Student(SerializationInfo info, StreamingContext ctx)

{

name = info.GetString("\_name").ToLower();

sex = info.GetChar("\_sex");

}

* 实现方法ISerializable.GetObjectData（序列化过程）

//自定义序列化过程

void ISerializable.GetObjectData(SerializationInfo info,

StreamingContext ctx)

{

//将字符串大写保存

info.AddValue("\_name", name.ToUpper());

info.AddValue("\_sex", sex);

}

* 使用特性自定义序列化
* OnSerializing：修饰的方法在序列化前调用
* OnSerialized：修饰的方法在完成序列化后调用
* OnDeserializing：修饰的方法在反序列化前调用
* OnDeserialized：修饰的方法在完成反序列化后调用

注意，以上被修饰的方法必须是void Fun(StreamingContext cxt)。

//定义一个可自定义序列化的类

[Serializable]

public class Student

{

//可序列化公有字段

public string name;

[XmlAttribute]//以特性序列化在xml中

public char sex;

[OnSerializing]//修饰的方法在序列化前调用

private void OnSerializing(StreamingContext ctx)

{

name = name.ToUpper();

}

[OnDeserialized]//修饰的方法在完成反序列化后调用

private void OnDeserialized(StreamingContext ctx)

{

name = name.ToLower();

}

}

自定义序列化过程对XmlSerializer序列化无效。