SqlAccessorチュートリアル

Tanaka Hiroshi [FAMILY Given]

SqlAccessorチュートリアル

Tanaka Hiroshi [FAMILY Given]

目次

表の一覧

第1章 SqlAccessorとは

SqlAccessorは、データベース(DB)からクラス単位でデータを抽出/追加/更新/削除するライブラリです。 このクラスをレコードと呼んでいます。レコードにはいくつかの決まりごとを除いて、 ユーザが自由に定義したクラスを使うことができます。DBから取得したデータをレコードというひと固まりのデータとして扱えるので、アプリケーションの作成が容易になります。

1.1. SqlAccessorの機能

他にも以下の機能があります。

・ 悲観的ロックを用いた排他制御

DBのデータをロックして、複数のトランザクションから同時に抽出/更新/削除されないように制御します。　DBが提供するロックとは異なり、ロック中のデータであっても参照することが可能です。そのため、ロック中のデータを画面に表示することもできます

・ DBとアプリケーションの間のデータ型変換

DBから取得した個々のデータは、レコードのプロパティに格納されますが、取得したデータはプロパティの データ型に自動的に型変換します。データ型変換についてユーザがコードを記述したり、設定をする必要はありません。

・ エラー処理

トランザクション実行中にエラーが発生したときは、トランザクションを自動的にロールバックして原子性を保証します。 エラー処理についてユーザがコードを記述したり、設定をする必要はありません。

1.2. SqlAccessorの利点

SqlAccessorはアプリケーションとDBの間に位置し、DBを扱う上で考慮すべき事柄を上記の機能によってアプリーケーション から隠蔽します。そのため、アプリーケーションでは以下の利点が得られます。

・ アプリーケーションの複雑性の低減

業務ロジック等のアプリーケーションが本来目的とするコードに、排他制御、データ型変換、エラー処理などの機能を 実現するコードが混ざり合うと、アプリーケーションのコードから本来目的とする機能や意図が掴みにくくなり、　複雑性が増加します。これらの機能はSqlAccessorが自動的に処理するため、アプリーケーションの複雑性を低減させることができます。

・ DBスキーマへの依存性の低減

DBとのデータのやり取りは全てレコードを用いるので、アプリケーションのDBスキーマへの依存を最小限にします。 DBスキーマが変更されても、レコードやそれを使用するアプリケーションへの変更を最小限に抑えられます。

・ 特定製品のDMBSからの独立

アプリーケーションが特定製品のDMBSに依存する主な理由は、アプリーケーション内にSQL文を埋め込んでしまうためです。　SqlAccessorで使用するSQL文は、SqlPodと呼ぶXMLファイルに

記述してアプリーケーションとSQL文を分離します。 その結果、DMBSが変更された場合でもアプリーケーションを変更する必要はありません。

第2章 チュートリアル

2.1. 準備

2.1.1. 動作環境

SqlAccessorを使用するのに必要なソフトウェアは以下の通りです。

- .NET Framework 2.0以上
- 操作対象DBのADO.NETドライバ

2.1.2. ファイルの配置

SqlAccessor構成ファイルを以下のような構成にして配置してください。配置場所はどこでも構いません。

- SqlAccessor.dll
- Foundation.dll
- MiniSqlParser.dll
- Antlr4.Runtime.Standard.dll
- PropertyTypes/CommonTypes.dll
- SqlPods/LockData.sql

2.1.3. アプリケーションの用意

SqlAccessorは単なるライブラリなので、これを利用するプログラムが無いと動作させることはできません。本マニュアルでは、SqlAccessorを利用するプログラムをアプリーケーションと呼びます。Visual Studio等でアプリーケーションを作成するためのプロジェクトを作成してください。

2.1.4. 参照設定

以下のDLLファイルをプロジェクトの参照設定に追加してください。

- SqlAccessor.dll
- Foundation.dll

2.1.5. 名前空間の指定

アプリーケーションのコードファイルには、以下の名前空間を指定してください。

```
using SqlAccessor;
```

以上で準備は完了です。

2.2. レコードを取得する

2.2.1. レコードの作成

DBとのデータのやり取りの単位となるレコードをアプリーケーションのプロジェクトに作成します。

```
```

```
public class Person : IRecord ❶ ❷

  public int      Id       { get; set; } ❸
  public string   Name     { get; set; }
  public DateTime BirthDay { get; set; }

  public int Age { ❹
    get {
      int age = DateTime.Now.Year - this.BirthDay.Year;
      return this.BirthDay > DateTime.Now.AddYears(-age) ? --age : age;
    }
  }
}
```

❶ レコードは必ず引数無しでnewできるように定義してください。
❷ レコードは必ずIRecordインタフェースを継承してください。
❸ DBの項目の値を格納するプロパティを定義します。プロパティのアクセサはpublicにします。またgetとsetの両方を定義して下さい。プロパティのデータ型は初期設定では、int, int?、long, long? decimal, decimal?、string、bool?, DateTimeが使えます。これら以外のデータ型を使用したい場合はPropertyType と呼ぶデータ型変換を行うDLLを用意します。
❹ DBの項目の値を格納しないプロパティ、メソッド、メンバ変数などは制限なく定義できます。

これでレコードの定義は完了です。

2.2.2. SqlPodの作成

DBに格納されているデータをレコードに格納するために、そのデータをDBからどのように取得するかを定義する必要があります。 この定義はSqlPodと呼ぶXMLファイルに1つのSELECT文を記述して定義します。SELECT句のAS別名をプロパティ名と同じ名称にすることで、 そのSELECT句の値をプロパティに格納するように定義します。

```
<?xml version="1.0" encoding="UTF-8"?>
<sqlPod>  ❶
  <Find><![CDATA[  ❷ ❸
  SELECT
    PersonId                    AS Id       ❹
   ,FirstName || ' ' || LastName AS Name     ❺
   ,BirthDay                    AS BirthDay
  FROM
      Persons
  ]]></Find>
</sqlPod>
```

❶ XMLのルート要素はsqlPodタグです。

❷ SELECT文はFindタグ内に記述します。
❷ WHERE句はSqlAccessorが自動的に付加するので、必要ありません。
❸ SQL文に"<"や">"を記述できるようにCDATAで囲むことをお勧めします。
❹ PersonIdにIdというAS別名を与えることで、PersonsテーブルのPersonId列の値を レコードのIdプロパティに格納するように定義しています。
❹ AS別名とプロパティ名は文字の大小も含めて一致させてください。
❺ SELECT句で演算した結果でもプロパティに対応させることができます。

上記のXMLファイルを作成し、SqlPodsディレクトリに配置します。XMLファイルの名称は、 [レコード名] + .sqlとします。

これでDBからレコードを取得する準備が完了しました。

2.2.3. Dbオブジェクトの生成と破棄

SqlAccessorを使用するには必ずDbオブジェクトを生成する必要があります。Dbオブジェクトの生成は以下のように 記述します。

```
var db = new Db([DBMS種別], [接続文字列]);
```

Dbオブジェクトの破棄は以下のように記述します。

```
db.Dispose();
```

Dbオブジェクトは、SqlAccessorの管理情報やDBのメタ情報を保持するオブジェクトです。使用されるに従ってそれらの 情報を収集していき効率的に動作するようになります。従って、Dbオブジェクトを生成した後は、全ての操作でDbオブジェクトを共有してください。 大抵の場合は、アプリケーションの開始時にDbオブジェクトを生成し、アプリケーションの終了時に破棄することになるはずです。

2.2.4. レコードの取得

DBからレコードを抽出するための抽出条件の指定方法には、レコードオブジェクトを指定する方法と、Queryオブジェクト を指定する2つの方法があります。

2.2.4.1. レコードオブジェクトを指定する方法

レコードオブジェクトを抽出条件として指定します。レコードオブジェクトの各プロパティとその値を、「プロパティ = 値」という一致条件でAND連結した条件として解釈します。

```
var db = new Db([DBMS種別], [接続文字列]);

// 抽出条件の作成
```

```
var criteria = new Person();
criteria.Name = "足利 尊氏";

// DBからPersonレコードを抽出する(型推論を利用)
var reader = db.Find(criteria);

// 抽出したPersonレコードを表示する
foreach(var person In reader) {
  WriteLine("抽出結果 >> " + person.Id.ToString());
  WriteLine("            " + person.Name);
  WriteLine("            " + person.BirthDay.ToString());
  WriteLine("            " + person.Age.ToString());
}

db.Dispose();
```

DBからレコードを抽出するにはFind()を使用します。引数には抽出条件であるPersonレコードを指定します。　Find()の戻り値は、全ての抽出結果を保持するReaderオブジェクトです。抽出したレコードはこのReaderオブジェクトから　1件づつ取り出します。Readerオブジェクトはレコードの型をジェネリックパラメータに持つ型で、上記の例では　Reader<Person>型となります。型推論を用いればFind()の戻り値の型を気にせずに済むのでお勧めです。

2.2.4.2. Queryオブジェクトを指定する方法

Queryオブジェクトとは、抽出条件を保持するためのオブジェクトです。Queryオブジェクトには一致条件以外の条件も格納 することができます。

```
var db = new Db([DBMS種別], [接続文字列]);

// 抽出条件の作成
var criteria = new Query<Person>();
criteria.And(val.of("Name").Like("足利%") &&
             val.of("BirthDay") < Datetime.Now
            );

// DBからPersonレコードを抽出する
var reader = db.Find(criteria);

// 抽出したPersonレコードを表示する
foreach(var person In reader) {
  WriteLine("抽出結果 >> " + person.Id.ToString());
  WriteLine("            " + person.Name);
  WriteLine("            " + person.BirthDay.ToString());
  WriteLine("            " + person.Age.ToString());
}
```

```
db.Dispose();
```

Queryオブジェクトの宣言時には、抽出対象となるレコードの型をジェネリックパラメータとして指定する必要があります。抽出条件を格納するにはAnd()を使用します。引数には抽出条件を指定しますが、「プロパティ名 + 演算子 + 値」という論理式で指定します。[1] プロパティ名は必ず、val.of("")で囲んでください。演算子には、=、!=、<、<=、=>、>が使えます。　またBETWEEN、LIKE、IN、IS NULLを指定する場合は、「val.of("Id").Between(A, B)」のように記述してください。[1]

作成したQueryオブジェクトは、Find()の引数に指定します。

2.3. レコードの件数を取得する

「1.　DBからレコードを取得する」の準備ができていれば、他の準備は不要です。　SqlAccessorはSqlPodに定義されたSELECT文から、件数を取得するためのSELECT　COUNT(*)文を自動生成します。　カウント対象のレコードの指定には、レコードの取得と同様にレコードオブジェクトを指定する方法と、Queryオブジェクトを指定する方法の二つがあります。

レコードオブジェクトを指定して、件数を取得するには以下の様に記述します。

```
var db = new Db([DBMS種別], [接続文字列]);

// 抽出条件の作成
var criteria = new Person();
criteria.Name = "足利 尊氏";

// 抽出条件に一致するレコードの件数を取得する
var i = db.Count(criteria);

// 件数を表示する
WriteLine("件数 : " + i.ToString());

db.Dispose();
```

Queryオブジェクトを指定する方法もレコードを取得する場合と同じ方法で指定できます。

2.4. レコードを格納する

DBにレコードを格納するには、SqlPodに更新系SQL文を定義します。

2.4.1. SqlPodへの追加

SqlPodにSaveタグを追加で定義し、その中には後で説明するSaveメソッドの実行時に発行されるSQL文を記述します。Saveメソッドは引数で渡されたレコードが既にDBに存在すれば更新処理を、

[1]実際にはAnd()の実行前に論理式が評価されるので、And()の引数には値が入ります。

存在しなければ新規追加処理を行うように用意されたメソッドです。そのためSaveタグ内には更新処理と新規追加処理の両方の場合に対応するSQL文を記述する必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<sqlPod>
  <Find><![CDATA[
  SELECT
    PersonId                      AS Id
   ,FirstName || ' ' || LastName AS Name
   ,BirthDay                      AS BirthDay
  FROM
      Persons
  ]]></Find>

  <Save><![CDATA[  ❶
  UPDATE Persons SET  ❷
    PersonId  = @Id  ❸
   ,FirstName = SUBSTRING(@Name FROM 0 FOR LENGTH(@Name) - POSITION(@Name, ' '))  ❹
   ,LastName  = SUBSTRING(@Name FROM POSITION(@Name, ' ') + 1)
   ,BirthDay  = @BirthDay
  ;  ❺

  IF @LAST_AFFECTED_ROWS = 0 THEN ❻
    INSERT INTO Persons(
      PersonId
     ,FirstName
     ,LastName
     ,BirthDay
    )VALUES(
      @Id
     ,SUBSTRING(@Name FROM 0 FOR LENGTH(@Name) - LOCATE(@Name, ' '))
     ,SUBSTRING(@Name FROM LOCATE(@Name, ' ') + 1)
     ,@BirthDay
    )
    ;
  END IF;
  ]]></Save>
</sqlPod>
```

❶ 追加/更新処理はSaveタグ内に記述します。

❷ WHERE句はSqlAccessorが自動的に付加するので、必要ありません。

❸ 追加/更新の値となるレコードのプロパティ値は、@プロパティ名で指定します。追加/更新先の列の型にリテラル表記を合わせるので　(列の型が文字列の場合自動的に''で囲む等)、ユーザは列のデータ型を気にする必要はありません。

❹ Nameプロパティは、SELECT文の定義でFirstName列とLastName列の値を文字列結合した値です。　そのため、UPDATE/INSERT文でFirstName列とLastName列に値を指定するには、Nameプロパティの 値を’’の前後で分割した値をそれぞれの列に指定する必要があります。

❺ Find以外のタグ内では";"でSQL文を区切ることで複数のSQL文を記述できます。

❻ Find以外のタグ内ではIF文の記述が可能です。条件式の@LAST_AFFECTED_ROWSは直前に実行したSQL文の更新対象行数が入ります。 上記の例では格納するレコードと同じIDを持つレコードが既にDBに存在するときはUPDATE文を、存在しない場合はINSERT文を発行します。

2.4.2. レコードの格納

DBにレコードを追加/更新するには、追加/更新対象のレコードを指定します。レコードのプロパティ値のうちテーブルのキー列に 紐付く値を、「キー列 = プロパティ値」という一致条件でAND連結した条件として解釈します。それ以外のプロパティ値を 追加/更新値として解釈します。

```
var db = new Db([DBMS種別], [接続文字列]);

// 格納レコードの作成
var newPerson = new Person();
newPerson.Id       = 1;
newPerson.Name     = "足利 尊氏";
newPerson.Birthday = new DateTime(1305, 7, 27);

// DBにPersonレコードを格納する
db.Save(newPerson);  ❶

db.Dispose();
```

❶ Save()を実行することでSqlPodのSaveタグに定義したSQL文が実行されます。引数には格納するPersonレコードを指定します。

2.5. レコードを削除する

DBからレコードを削除するには、SqlPodに削除のためのSQL文を追加します。

2.5.1. SqlPodへの追加

DBからレコードを削除するには、SqlPodにDeleteタグを追加で定義してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<sqlPod>
  <Find><![CDATA[
  SELECT
    PersonId                    AS Id
```

```
   ,FirstName || ' ' || LastName AS Name
   ,BirthDay                     AS BirthDay
  FROM
      Persons
  ]]></Find>

  <Save><![CDATA[
  UPDATE Persons SET
    PersonId  = @Id
   ,FirstName = SUBSTRING(@Name FROM 0 FOR LENGTH(@Name) - POSITION(@Name, ' '))
   ,LastName  = SUBSTRING(@Name FROM POSITION(@Name, ' ') + 1)
   ,BirthDay  = @BirthDay
  ;

  IF @LAST_AFFECTED_ROWS = 0 THEN
    INSERT INTO Persons(
      PersonId
     ,FirstName
     ,LastName
     ,BirthDay
    )VALUES(
      @Id
     ,SUBSTRING(@Name FROM 0 FOR LENGTH(@Name) - LOCATE(@Name, ' '))
     ,SUBSTRING(@Name FROM LOCATE(@Name, ' ') + 1)
     ,@BirthDay
    )
     ;
  END IF;
  ]]></Save>

  <Delete><![CDATA[ ❶
    DELETE FROM Persons  ❷
  ]]></Delete>
</sqlPod>
```

❶ Deleteタグ内にSQL文を記述します。SQL文は";"で区切ることで複数記述可能です。

❷ WHERE句はSqlAccessorが自動的に付加するので、必要ありません。

2.5.2. レコードの削除

DBからレコードを削除するには、削除するレコードの抽出条件を指定します。レコードの抽出条件の指定は、 レコードの取得と同様にレコードオブジェクトを指定する方法と、Queryオブジェクトを指定する2つの方法があります。

2.5.2.1. レコードオブジェクトを指定する方法

レコードオブジェクトを抽出条件として指定します。

```
var db = new Db([DBMS種別], [接続文字列]);

// 格納レコードの作成
var deadPerson = new Person();
deadPerson.Id       = 1;
deadPerson.Name     = "足利 尊氏";

// DBからPersonレコードを削除する
db.Delete(deadPerson); ❶

db.Dispose();
```

❶ DBからレコードを削除するにはDelete()を使用します。引数には抽出条件であるPersonレコードを指定します。

2.5.2.2. Queryオブジェクトを指定する方法

Queryオブジェクトで抽出条件を指定します。

```
var db = new Db([DBMS種別], [接続文字列]);

// 抽出条件の作成
var criteria = new Query<Person>
criteria.And(val.of("Id") = 1 &&
             val.of("Name").Like("足利%"
            ); ❶

// 抽出対象のレコードを更新する
db.Delete<Person>(criteria);

db.Dispose();
```

❶ Queryオブジェクトの指定方法はFindメソッドで指定する場合と同じです。

第3章 FAQ

問： レコードのプロパティに指定できるデータ型には何がありますか?

答： int、int?、long、long?、decimal、decimal?、bool?、string、DateTimeが　指定できます。もちろん、DBから取得した値を格納しないプロパティにはどのようなデータ型を　指定しても構いません。

問： プロパティのデータ型に、独自に定義したデータ型を指定したい

答： 独自に作成したクラスを型としてプロパティのデータ型に指定するには、PropertyTypeと呼ぶ　DBのデータ型とクラスの型の変換を定義したDLLファイルを所定のディレクトリに配置します。 PropertyTypeの作成方法は、チュートリアルの域を超えるのでここでは詳細は控えます。

問： NULLはどのような値で返しますか?

答： DBに格納されている値がレコードのプロパティに格納される時、SqlAccessorはデータ型変換を行います。　どのようにデータ型変換を行うかは、プロパティの型ごとに定義されています。NULLもまたデータ型変換の処理を経て　プロパティに格納されるため、NULLをどのような値に変換するかは格納先プロパティの型ごとに定義されています。変換後の値はホスト言語においてはNULLと扱うので、NULL表現値と呼んでいます。以下に初期設定で使用可能な データ型とそれに対応するNULL表現値を示します。

表3.1 NULL表現値

データ型	NULL表現値	NULL表現値(定数表現)
int	-2147483648	int.MinValue
int?	null	null
long	-9223372036854775808	long.MinValue
long?	null	null
decimal	-79228162514...3543950335	decimal.MinValue
decimal?	null	null
bool?	null	null
string	""	string.Empty
DateTime	0001/01/01 00:00:00.000	DateTime.MinValue

問： NULL表現値を変更したい

答： NULL表現値を本来の値の意味で使用したい場合など、NULL表現値を変更したい場合が考えられます。　int、int?、long、long?、decimal、decimal?、bool?、string、DateTimeのNULL表現値を変更するには、 これらのデータ型変換処理を定義したCommonTypes.dllを変更する必要があります。この方法を採用する前に以下の方法も検討してみてください。

- データ型の全ての値をNULL表現値に当てられない場合、Nullable型を使用する方法があります。

例えば、int型の代わりにint?を使用して、Nullable.HasValue = Falseな値をNULL表現値とします。

・ 値型の代わりに参照型を使用して、nullをNULL表現値にします。

例えば、intを金額の意味で使用する場合は、intの代わりに独自に定義したMoney型を使用します。 Money型は参照型なので、この型のプロパティのNULL表現値にnullを利用できます。[1]

問: レコードオブジェクトを抽出条件とするとき、条件に含めたくないプロパティにはどのような値を格納しますか?

答: 一致条件として扱われたくないプロパティには、NULL表現値を格納してください。 なお、コンストラクタでプロパティ値にNULL表現値を格納するようにしておけば、 抽出条件用のレコードオブジェクトの作成が容易になります。

問: どんなSQL文でも記述できますか?

答: 以下のSQL文をSqlPodに記述できます。

・ SELECT文
・ INSERT文
・ UPDATE文
・ DELETE文
・ IF文

以下のSQL文は記述可能ですが、対象となるテーブル行はロックできません。

・ INSERT-SELECT文
・ CALL文
・ Viewを含んだSQL文

上記以外のSQL文は記述できません。

問: IS NULLを抽出条件にするには?

答: Nameプロパティが NULLであるという条件は以下のように記述します。

```
query.And(val("Name").IsNull);
```

問: 抽出条件にサブクエリを指定するには?

答: プレースホルダで指定してください。

```
SELECT PersonId, Name
```

[1] 一般的に値の取りうる範囲ではなく、値の意味で型を定義するとコードの可読性を向上させるうえ、 コンパイラの型チェックを有効に利用できます。

```
FROM   Persons
WHERE  @Predicate
```

と定義したSELECT文に対して、

```
@Predicate = "EXISTS (SELECT * FROM Persons P
                      WHERE P.Birthday = Persons.Birthday)"
```

というプレースホルダ値を指定します。[2]

問: プレースホルダとは?

答: SqlPodに定義したSQL文の一部を任意の値や式に置換するための目印です。 以下のSQL文の@Predicateがプレースホルダです。プレースホルダは全て@で始まる名称で指定します。

```
SELECT PersonId, Name
FROM   Persons
WHERE  @Predicate
```

プレースホルダを置き換える値や式をプレースホルダ値と呼びます。 プレースホルダ値は、以下の様にQueryオブジェクトに格納します。

```
// プレースホルダ値を設定する
var criteria = new Query<Person>();
criteria.SetPlaceHolder("Predicate", "1=1");

// DBからPersonレコードを抽出する
var reader = db.Find(criteria);
```

問: プレースホルダに初期値を設定するには?

答: プレースホルダ値に初期値を設定すると、抽出時に値が設定されなかった場合にその初期値が使用されます。初期値はSQL文のコメント内で定義します。

```
/** @Predicate = "1=1" */
```

[2]この方法はアプリーケーションがSQL文を扱うことになるので、アプリーケーションがDBスキーマ、及び特定DBMSのSQL方言に依存する危険性があります。そのため、将来のバージョンでは、Queryオブジェクトにサブクエリを指定できるようにする予定です。

```
SELECT PersonId, Name
FROM   Persons
WHERE  @Predicate
```

初期値は、/** @[プレースホルダ名] = [プレースホルダ値] */ の形式で定義します。

問: テーブルにタイムスタンプを格納するには?

答: DBサーバの日付と時刻を参照する場合は、CURRENT_DATE()等のSQL関数を使用します。 SqlAccessorが稼働しているサーバの日付と時刻を参照する場合は@CURRENT_DATE、@CURRENT_DATETIMEを使用します。 これらのプレースホルダはSqlAccessorが事前に値を格納するプレースホルダで、定義済みプレースホルダと呼んでいます。

表3.2 定義済みプレースホルダ

プレースホルダ	値	例
@CURRENT_DATE	現在日付	20180401
@CURRENT_DATETIME	現在日付時刻	20180401000000

問: トランザクションを使うには?

答: 1つのトランザクションに複数の操作を含めるには、以下の様に記述します。

```
var db = Db([DBMS種別], [接続文字列]);

// レコードの作成
var takauji = new Person();
takauji.Id = 1;
takauji.Name = "足利 尊氏";
takauji.Birthday = New DateTime(1305, 7, 27);

var yoshiakira = new Person();
yoshiakira.Id = 2;
yoshiakira.Name = "足利 義詮";
yoshiakira.Birthday = New DateTime(1330, 7, 4);

// レコードを格納する
using(var tran = db.CreateTran()) {  ❶❷
  tran.Save(takauji);  ❸
  tran.Save(yoshiakira);
}

db.Dispose();
```

❶ tranオブジェクトをdbオブジェクトから取得します。
❷ tranオブジェクトは必ずDispose()で終了させてください。上記コードでは、例外が送出された 場合でも確実にDispose()を呼ぶために、using句を使用しています。
❸ tranオブジェクトに対して、Find()、FindOne()、Save()、Delete()などの操作を行えます。同じtranオブジェクトに対する操作は、1つのトランザクションに含まれます。

問: トランザクションの終了を忘れたときは?

答: tranオブジェクトがガーベージコレクトされる時に、そのtranオブジェクトのDispose()が呼ばれていなければ、 ガーベージコレクタが代わりにDispose()を呼び出します。ただし、SqlAccessorはこの時にトランザクションを 終了することを保証しません。SqlAccessorは可能な限りトランザクションを終了させようとしますが、 必ずしも成功するとは限らないためです。また、ガーベージコレクタの起動は不定期なため、 かなり後にならないと終了しない場合もあります。トランザクションはロック情報を保持し続けるので、 終了するまでの間ずっとロックを掛け続けます。これらの理由から、トランザクションは必ず終了するようにしてください。

問: 同時に2つ以上のトランザクションを扱えますか?

答: 可能です。1つのtranオブジェクトが1つのトランザクションを意味しています。 複数のトランザクションを同時に扱いたい場合は、複数のtranオブジェクトを生成するだけです。

問: 一つのトランザクション内で、RecordReaderからレコードを取得しつつ、そのレコードを格納することはできますか?

答: できません。RecordReaderを生成してからDispose()するまでは、同一トランザクションに対するいかなる操作もできません。 この制限はADO.NETに由来するものです。以下のコードは例外InvalidOperationExceptionが送出されます。

```
// DBからPersonレコードを抽出する
var reader = tran.Find<Person>(criteria);

foreach(var person in reader) {
  tran.Delete(person);  // 例外InvalidOperationExceptionが送出される
}
```

このようなコードを可能とするためには、おそらく分散トランザクションを実現する必要があるでしょう。

問: 複数のDBやトランザクションをまたがるトランザクション(分散トランザクション)を扱えますか?

答: 扱えません。ただし、要望が強ければ検討することはあり得ます。

問: トランザクション内でロックしたテーブル行のうち一部の行だけをロック解除できますか?

答: できません。ロックを解除する手段は唯一Tran.Dispose()だけです。 この要望を実現するには、セーブポイントを利用したトランザクションの部分ロールバックか、入れ子トランザクション

の　どちらかを実装する必要があります。この要望はときどき挙げられるので必要性を感じていますが、実装に手間がかかると予想しているので当面はこのままです。

問：　レコードをINSERTしてCOMMITするまでの間に、他トランザクションで同じキーのレコードをINSERTすると送出する例外は?

答：　以下のコードを実行したとき、送出される例外はDuplicateKeyExceptionです。

```
var yoshimitsu = new Person();
yoshimitsu.Id = 3;
yoshimitsu = "足利 義満";
yoshimitsu = new DateTime(1358, 9, 25);

var tran1 = db.CreateTran();
var tran2 = db.CreateTran();

tran1.Insert(yoshimitsu);
tran1.Dispose();

tran2.Insert(yoshimitsu);
tran2.Dispose();
```

しかし、以下のコードを実行したとき、送出される例外はWriteToLockedRecordExceptionです。

```
var yoshimitsu = new Person();
yoshimitsu.Id = 3;
yoshimitsu = "足利 義満";
yoshimitsu = new DateTime(1358, 9, 25);

var tran1 = db.CreateTran();
var tran2 = db.CreateTran();

tran1.Insert(yoshimitsu);
tran2.Insert(yoshimitsu);

tran1.Dispose();
tran2.Dispose();
```

前者のコードは、tran1.Dispose()によってレコードの新規追加が確定した後に、同じキー値のレコードを格納しようと　しています。その時に送出される例外は、キー重複によってレコードが格納出来なかったことを示すDuplicateKeyExceptionです。　それに対して後者のコードは、tran1のトランザクションが確定する前に、tran2が同じキー値のレコードを格納しよう　とし

ています。従って、レコードyoshimitsuの格納が確定していないので、キー重複によってレコードを格納できない という事実もまた確定できないのです。もし、DuplicateKeyExceptionを送出した後にtran1がロールバックすれば、 キー重複という事実は誤った情報になります。この時点で確定できることは、tran1がレコードyoshimitsuをロックしている ためにtran2はレコードを格納できないということだけです。

問: DBから切断した後もトランザクションを維持できますか?

答: できます。Dbトランザクションが破棄されない限り、DBから切断したあとでもトランザクションを維持することが可能です。 この機能は、ASP.NETフレームワークでWebページを跨いでトランザクションを維持したい場合や、 トランザクションがDBが扱える期間を超えるぐらいの長期間に及ぶ場合などに有用です。

トランザクションを維持しつつDBから切断するには、Tran.Suspend()を呼び出すだけです。この操作で行われる処理を、 トランザクションの中断と呼んでいます。中断したトランザクションを再開するには、Find()等の操作を行うだけです。 SqlAccessorは、中断中のトランザクションに対してこの様な操作が行われたとき、自動的にトランザクションを再開します。 また、Tranオブジェクトをシリアライズしたときには自動的に中断されます。

問: Find()/FindOne()で抽出するレコードをロックするには?

答: 以下の様に、引数にReadWriteを指定すれば抽出したレコードをロックします。 ただし、Find()の処理内ではロックはしません。 Find()が返したRecordReaderからレコードを取得するときにそのレコードをロックします。

```
var reader = tran.Find(criteria, Tran.ReadWrite);

foreach(var person in reader) {  // ここで、aPersonオブジェクトがロックされます。
  WriteLine("抽出結果 >> " + aPerson.Name);
}
```

問: レコードがロックされているか否かを判定するには?

答: レコードのロックの有無は、RecordReaderのWritableプロパティで判定します。 Writableプロパティは RecordReaderが列挙中のレコードのロックの有無を返します。 Trueならロック中で、Falseならロックされていません。

```
var reader = tran.Find(criteria, Tran.ReadWrite);

foreach(var person in reader) {  // ここで、aPersonオブジェクトがロックされます。
  WriteLine("抽出結果 >> " + aPerson.Name);
  if(!reader.Writable) {
    WriteLine("このレコードはロックされています");
  }
}
```

なお、Find()の引数にReadOnlyを指定した場合(指定しない場合も含みます)、Writableプロパティは常にFalseを返します。 Save()などでロック中のレコードを更新しようとしたときは、例外WriteToLockedRecordExceptionを送出します。

問: Save()/Delete()が余分にテーブル行をロックします

答: Save()/Delete()では、INSERT文のVALUES句、UPDATE/DELETE文ではWHERE句から対象とするテーブル行を判別し、 そのテーブル行をロックします。判別の精度が不足しているため、実際にINSERT/UPDATE/DELETEが対象とするテーブル行と SqlAccessorが判別したテーブル行とは必ずしも一致しません。例えば、列に対する単純な一致条件であれば認識できますが、 サブクエリを使用したような条件であれば認識できません。その結果、更新対象ではないテーブル行がロックされることが あり得ます。幸いにも、SqlAccessorが判別するテーブル行は、INSERT/UPDATE/DELETEが実際に対象とするテーブル行よりも 常に大きい範囲なので、更新対象となるテーブル行は必ずロックします。

問: Find()/FindOne()で述語ロックは使えないですか?

答: UPDATE/DELETE文の実行時は、WHERE句に記述された条件でロック対象となるテーブル行を特定します。 この様に、ロック対象を条件で特定してロックする手法を、述語ロックと呼びます。 述語ロックを使用すると、ファントムリードが防げます。また、ロック対象としている条件に一致するテーブル行が、 INSERTされることも防げます。これらの特徴は殆どのユーザにとって利点になるはずです。

残念ながら、Find()/FindOne()は述語ロックではなく値ロック方式なので、Find()/FindOne()でロックしても これらの利点はありません。その代わり、Find()/FindOne()はテーブル行を余分にロックすることはありません。 この様な仕様にした理由は、UPDATE/DELETE文に比べてSELECT文の方が発行される頻度が高いのと、 UPDATE/DELETE文に比べてSELECT文の条件は複雑になることが多いため、テーブル行を余分にロックする頻度も 高いと考えたためです。 ただし、Find()/FindOne()で述語ロックの利点を得ることが、原理的に不可能とまでは考えていないので、将来のバージョンでは改善される可能性もあります。

問: DBに接続出来るか判定するには?

答: 以下の様に、Db.IsConnectable([接続文字列])で判定できます。

```
if(db.IsConnectable([接続文字列])) {
  WriteLine("接続OK");
} else {
  WriteLine("接続失敗");
}
```

問: DBへ発行したSQLを確認するには?

答: Debug版のSqlAccessorは標準出力に発行したSQL文を出力します。 大抵はVisial Studioの出力ウインドウで主力内容を確認します。

エラーリファレンス

目次

名前

BadFormatSqlPodException

説明

SqlPodファイルが正常に読込めなかった場合に送出します。

原因

SqlPodファイルの記述が誤っている

名前

DbAccessException

説明

DBへのアクセスでエラーが発生した場合に送出します。

原因

- DBへの接続/切断ができない。
- DBトランザクションの開始/終了ができない。
- SQL文を発行したらDBがエラーを返した。

名前

DuplicateKeyException

説明

レコードを新規追加しようとして一意性制約エラーが発生した時に送出します。

原因

既に同じキーのレコードがDBに存在している。

名前

InvalidColumnToPropertyCastException

説明

テーブルのカラムからレコードのプロパティへのデータ変換に失敗した時に送出します。

原因

- テーブルのカラムの値を、int, int?, long, long?, char, char?, string、bool?, Datetime以外のデータ型のプロパティに格納しようとした。
- 独自に定義したデータ型に対する変換処理を作成したが、その処理が誤っている。

名前

InvalidPropertyToColumnCastException

説明

レコードのプロパティからテーブルのカラムへのデータ変換に失敗した時に送出します。

原因

プログラムのバグ

名前

InvalidOperationException

説明

オブジェクトの状態が仕様と異なる遷移をした時に送出します。

原因

プログラムのバグ

名前

MoreThanTwoRecordsException

説明

FindOne()で2件以上のレコードを抽出した時に送出します。

原因

FindOne()の引数に渡した抽出条件では1件のレコードに特定できない。

名前

SqlSyntaxErrorException

説明

SqlAccessorで受け付けられないSQL文を発行しようとした時に送出します。

原因

SQL文の文法が誤っている

名前

WriteToLockedRecordException

説明

ロックされたレコードを更新しようとした時に送出します。

原因

同上

索引