機能紹介：XEP による .NET オブジェクト永続性

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

機能紹介 : XEP による .NET オブジェクト永続性
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください:

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

機能紹介：XEP による .NET オブジェクト永続性

この機能紹介では、InterSystems IRIS® データ・プラットフォームでのきわめて迅速な .NET オブジェクトの格納と取得のサポートを提供する XEP API を紹介します。.NET オブジェクト永続性を実現する XEP アプローチの概要を示し、API の主要機能の実例を示す単純なシナリオを紹介します。

これらのアクティビティは、既定の設定と機能のみを使用する設計になっているので、ユーザはこの概要の範囲を超える詳細を扱うことなく、XEP の基本部分を十分に理解することができます。XEP の完全なドキュメントは、"InterSystems XEP による .NET オブジェクトの永続化" を参照してください。

InterSystems IRIS の無料の評価版インスタンスで実施できるものも含めて、すべての機能紹介を確認するには、"インターシステムズの機能紹介" を参照してください。

1 迅速なオブジェクトの格納と取得

オブジェクト指向プログラミングは .NET Framework の中心であるため、.NET アプリケーションは当然、データをオブジェクトとしてモデル化します。ただし、そのデータをデータベース内に保存する必要があるアプリケーションの場合、これによって問題が発生する可能性があります。ADO.NET を使用してオブジェクトの格納と取得を行う場合、オブジェクト・データを一連のリレーショナル・テーブルに変換し、クエリ結果セットをオブジェクトに戻すときに、問題に直面します。この問題に対する有益な解決策は、Entity Framework などのオブジェクト・リレーショナル・マッピング (ORM) フレームワークを使用して、プロセスを自動化することです。InterSystems IRIS は Entity Framework インタフェースを提供しているため、大規模で複雑なオブジェクト階層には、これを使用することをお勧めします。一方で、リアルタイムのデータ取得などのタスクを実行するアプリケーションにとって、主な問題はデータの複雑さではなく、速度です。Entity Framework は (適切に最適化されていれば) 決して低速ではありませんが、単純なデータや適度に複雑なデータをきわめて迅速に格納および取得する必要があるタスクの場合、XEP の方が優れた選択肢です。ほとんどの場合、XEP は Entity Framework または ADO.NET のいずれよりもかなり迅速です。

2 XEP の仕組み

XEP は軽量の .NET API で、.NET オブジェクト・データを永続イベントとして投影します。永続イベントは、.NET オブジェクト内のデータ・フィールドのコピーを含む InterSystems IRIS クラス (通常は %Persistent のサブクラス) のインスタンスです。他のこうしたインスタンスと同様に、オブジェクト・アクセス、SQL クエリ、直接グローバル・アクセスによって取得できます。

永続イベントを作成して格納するには、事前に XEP が対応する .NET クラスを分析して、データベースにスキーマをインポートする必要があります。スキーマは、.NET オブジェクトの格納に使用する永続イベント・クラスの構造を定義します。スキーマをインポートすることで、永続イベント・クラスに対応する ObjectScript スキーマが自動的に作成されます (まだ存在していない場合)。単純なスキーマのインポートで XEP が必要とするものは、この分析からの情報ですべてまかなわれる場合もあります。もっと複雑な構造では、XEP がインデックスを生成して、フィールドのインポートに関する既定のルールを上書きできるように、追加情報を提供できます。

クラスにスキーマを作成した後は、さまざまな XEP メソッドを使用して、イベントの格納、更新、削除、SQL クエリの実行、およびクエリ結果セットでの反復を行うことができます。

3 体験 : XEP の実践

それではご自身で実際に XEP を試してみましょう。この XepSimple デモは、きわめて小さなプログラムですが、XEP の主要機能のほとんどについて例を提供しているので、XEP API を使用する方法の概要を学ぶことができます。

3.1 開始の前に

この手順を使用するには、.NET フレームワークおよび Visual Studio がインストールされた、操作するための Windows システム、および接続先として、稼働している InterSystems IRIS インスタンスが必要です。InterSystems IRIS の選択肢としては、いくつかのタイプのラインセンス付与されたインスタンスおよび無料の評価版インスタンスがあります。操作しているシステムでインスタンスをホストする必要はありません (ただし、相互のネットワーク・アクセスが必要です)。操作を実行するインスタンスをまだ用意できていない場合にインスタンスのタイプ別の導入方法の詳細を確認するには、"InterSystems IRIS の基礎 : IDE の接続" の "InterSystems IRIS の導入" を参照してください。同じドキュメントの "InterSystems IRIS 接続情報" および ".NET IDE" の情報を使用して、Visual Studio を InterSystems IRIS インスタンスに接続します。

3.1.1 Visual Studio プロジェクトの構成

まず、Visual Studio を開き、[Visual C#] オプションおよび [**コンソール アプリ** (.NET Framework)] オプションを選択して、新しいコンソール・アプリケーション・プロジェクトを作成します。[**名前**] フィールドに、`netxep` と入力します。

3.1.2 アセンブリ参照の追加

XEP API は、**InterSystems.Data.XEP.dll** ライブラリにパッケージ化されています。これは、ローカル・システムにインストールする必要があります。最新バージョンのライブラリは、"InterSystems IRIS Driver Packages" ページからダウンロードできます。InterSystems IRIS がローカル・マシンまたはアクセスできる別のシステムにインストールされている場合、このファイルは、install-dir**¥dev¥dotnet¥bin¥**version (install-dir はインスタンスのインストール・ディレクトリで、version はライブラリの .NET バージョン) にあります。

プロジェクトに **InterSystems.Data.XEP.dll** へのアセンブリ参照を追加するには、以下の手順を実行します。

1. Visual Studio のメイン・メニューで、[**プロジェクト**]→[**参照の追加**] を選択します。
2. 表示されるウィンドウで [**参照...**] をクリックします。
3. **InterSystems.Data.XEP.dll** ファイルの場所を参照します。
4. このファイルを選択し、[**追加**] をクリックします。
5. [OK] をクリックします。

Visual Studio のソリューション・エクスプローラでは、[**参照**] の下に **InterSystems.Data.XEP.dll** アセンブリが表示されます。

3.2 サンプル・クラスの追加

プログラムのソース・ファイルを作成する前に、メイン・プログラムがサンプル・オブジェクトを生成して格納するためにアクセスするクラスを追加します。Visual Studio で、[**プロジェクト**]→[**クラスの追加**] を使用して、C# クラス・ファイルを **netxep** プロジェクトに追加します。クラス・ファイルの既定の内容を削除し、以下のコードをコピーして貼り付けます。

```
using System;

namespace xep.samples
{
    public class SingleStringSample {
        public string name;
        public SingleStringSample() { }
        internal SingleStringSample(string str) {
```

```
            name = str;
        }

        public static SingleStringSample[] generateSampleData(int objectCount) {
            SingleStringSample[] data = new SingleStringSample[objectCount];
            for (int i = 0; i < objectCount; i++)
            {
                data[i] = new SingleStringSample("single string test");
            }
            return data;
        }
    }
}
```

このクラスは非常に単純なので、XEP では、ここから有効なスキーマを生成するためにアノテーションは必要ありません。現実のアプリケーションのより複雑なクラスでは、さまざまなオプションを使用してインポートを最適化できます。

3.3 XEP デモ・プログラムの作成

XEPSimple デモ・プログラムを作成する準備ができました。Visual Studio で、プロジェクトの作成時に作成した既定の **program.cs** ファイルを見つけます。ファイルの既定のコンテンツを削除し、以下のコードを貼り付けて、`host`、`port`、`irisnamespace`、`username`、および `password` の各変数の値を該当する InterSystems IRIS インスタンスの接続情報に置き換えます。示されているとおりに **USER** ネームスペースを指定することも、インストールしたインスタンスで作成した別のネームスペースを選択することもできます。

```
using System;
using InterSystems.XEP;
using xep.samples;

namespace XepSimpleNamespace
{
    public class XepSimple
    {
        public static void Main(string[] args)
        {
            // Generate 12 SingleStringSample objects for use as test data
            SingleStringSample[] sampleArray = SingleStringSample.generateSampleData(12);

            // EventPersister
            EventPersister xepPersister = PersisterFactory.CreatePersister();

            String host = "127.0.0.1"; // InterSystems IRIS host
            int port = 51774; // InterSystems IRIS Superserver port
            String irisnamespace = "USER"; // InterSystems IRIS namespace
            String username = "_system"; // Credentials for InterSystems IRIS
            String password = "SYS"; //Credentials for InterSystems IRIS

            xepPersister.Connect(host,port,irisnamespace,username,password); // connect to localhost
            xepPersister.DeleteExtent("xep.samples.SingleStringSample");   // remove old test data
            xepPersister.ImportSchema("xep.samples.SingleStringSample");   // import flat schema

            // Event
            Event xepEvent = xepPersister.GetEvent("xep.samples.SingleStringSample");

            long[] itemIdList = xepEvent.Store(sampleArray);
            int itemCount = 0;
            for (int i = 0; i < itemIdList.Length; i++)
            {
                if (itemIdList[i] > 0) itemCount++;
            }
            Console.WriteLine("Stored " + itemCount + " of " + sampleArray.Length + " events");


            // EventQuery
            EventQuery<SingleStringSample> xepQuery = null;
            String sqlQuery = "SELECT * FROM xep_samples.SingleStringSample WHERE %ID BETWEEN ? AND
?";
            xepQuery = xepEvent.CreateQuery<SingleStringSample>(sqlQuery);
            xepQuery.AddParameter(3);    // assign value 3 to first SQL parameter
            xepQuery.AddParameter(12);   // assign value 12 to second SQL parameter
            xepQuery.Execute();            // get resultset for IDs between 3 and 12


            xepQuery.Close();
            xepEvent.Close();
            xepPersister.Close();
```

```
        } // end main()
    } // end class xepSimple
}
```

デモ・プログラムは 3 つのセクションに分かれており、それぞれが 3 つのメイン XEP クラス EventPersister、Event、および EventQuery のいずれかを紹介しています。XepSimple デモ・プログラムを実行すると、以下のタスクが実行されます。

- まず、サンプルのデータ・クラス xep.samples.SingleStringSample のメソッドを呼び出すことで、サンプルのオブジェクトがいくつか生成されます。
- EventPersister は、XEP の主なエントリ・ポイントで、データベースへの接続を提供します。

  このクラスは、xepPersister という名前のインスタンスを作成します。このインスタンスは、データベース内の User ネームスペースへの TCP/IP 接続を確立して、既存のサンプル・データをすべて削除します。次に、このクラスは、ImportSchema() を呼び出して、サンプル・クラスを分析し、スキーマをデータベースに送信します。その結果、SingleStringSample 永続オブジェクトを保持する、対応する ObjectScript スキーマが作成されます。
- Event は、.NET オブジェクトと対応するデータベース・オブジェクトとの間のインタフェースをカプセル化します。

  スキーマが生成されると、xepPersister は、サンプル・クラスに xepEvent という名前の Event オブジェクトを作成できます。Store() メソッドは、.NET オブジェクトの配列を永続イベントとして格納します。
- EventQuery は、SQL クエリの限定されたサブセットの準備と実行に使用されます。

  xepEvent オブジェクトの CreateQuery() メソッドにクエリ文字列を渡すことによって、xepQuery という名前の EventQuery<SingleStringSample> オブジェクトが作成されます。この文字列によって、2 つのパラメータ (? 文字) を受け取る SQL クエリが定義されます。AddParameter() の呼び出しによってパラメータ値が定義され、Execute() の呼び出しによってクエリの結果セットがフェッチされます。
- 処理が完了したら、XEP オブジェクトに対して close() メソッドを呼び出すことによって、クリーン・アップが行われます。

3.4 XEPSimple プログラムの実行

XEPSimple デモ・プログラムをビルドして実行する準備ができました。Ctrl-F5 を押してプログラムを実行し、プログラムの終了時にコマンド・プロンプトが開いたままになるようにします。

3.5 次の手順

XsepSimple デモは、詳細にとらわれずに XEP の基本機能を体験していただくことを目的としており、実際のプロダクション・コードのモデルではありません。例外の確認さえ行いませんでした。簡略化されている中でも最も重要な部分がサンプル・データです。スキーマの生成や最適化に役立つアノテーションやその他のメカニズムを必要としない、いくつかの小さな .NET オブジェクトをクラスから永続化しました。実際のアプリケーションでは通常、アノテーションやその他のオプションが必要になりますが、これらのオプションはわかりやすく簡単に使用できます。API も同様に単純で簡単に使用できます。

XEP のすべての機能に関する包括的な説明は、"InterSystems XEP による .NET オブジェクトの永続化" を参照してください。

4 .NET サポートの詳細

InterSystems IRIS は、SQL テーブル、オブジェクト、および多次元ストレージを使用した簡単なデータベース・アクセスを実現する .NET API を提供します。各アクセス・タイプの詳細は、以下のドキュメントを参照してください。

- .NET アプリケーションからの InterSystems IRIS グローバルへのアクセスについては、"Native SDK for .NET の使用法"。
- SQL テーブル・アクセスについては、"ADO.NET Managed Provider クラスの使用法"。ADO.NET Managed Provider 汎用クラスの完全準拠バージョンを使用して .NET プロジェクトから InterSystems IRIS データベースにアクセスできるようにします。
- SQL テーブル・アクセスについては、"InterSystems ODBC ドライバの使用法"。InterSystems ODBC ドライバにより、InterSystems IRIS は外部アプリケーションへの ODBC 接続を確立し、SQL 経由で外部データ・ソースにアクセスできるようになります。
- オブジェクトへのアクセスについては、"InterSystems XEP による .NET オブジェクトの永続化"。単純なものから中程度の複雑さのオブジェクト階層を操作して非常に高速のオブジェクト・データ永続性と取得が必要なトランザクション処理アプリケーションに対して XEP は最適化されています。
- オブジェクトのリレーショナル・マッピングについては、"Entity Framework Provider の使用法"。Entity Framework テクノロジを使用した InterSystems IRIS データベースへのアクセスに関する情報を提供します。