
記事

[Hiroshi Sato](#) · 2020年8月18日 8m read

[Open Exchange](#)

Caché .Net Binding **アプリケーションをIRISの.Net Native APIを利用して書き換える方法（その１）**

IRISでは.Net Bindingは非推奨機能となりました。

.Net Bindingを使ったアプリケーションは、IRISで提供されている.Net Native APIを利用して書き換えることができます。

ここでは、実際に書き換えをおこなったサンプルコードを示しながら、具体的な方法を説明していきます。

CacheDirect(VisM)エミュレーター

OpenExchangeに登録しているVisMエミュレーターは、元々Cachéの.Net Bindingを使用して作成されました。

それをIRISの標準機能で動作可能にするために、.Net Native APIを使用して書き換えをおこないました。

以下にどのように書き換えを行ったかを順を追って説明します。

参照の変更

まず以前の参照を削除します。

Visual Studioのソリューションエクスプローラーの所で参照をクリックします。

表示されるInterSystems.Data.CacheClientを削除します。
(右クリックして削除を選ぶ)

次にプロジェクトメニューから参照の追加をクリックして、以下の２つのファイルを選択します。

（プロジェクトの.Net Frameworkバージョンに合わせて、それに対応するファイルを選択する
以下の例は、v4.5を選択）

c:\InterSystems\IRIS\dev\dotnet\bin\v4.5

InterSystems.Data.IRISClient.dll

using句の変更

先頭のusing句の変更が必要になります。

```
using InterSystems.Data.CacheClient;  
using InterSystems.Data.CacheTypes;
```

上記を以下の様に書き換えます。
using InterSystems.Data.IRISClient;
using InterSystems.Data.IRISClient.ADO;

connection情報

connectionオブジェクトをCachéからIRISに変更する必要があります。

```
CacheConnection conn;
```

```
public IRISConnection conn = new IRISConnection();
```

Proxyクラスの削除

.Net Native APIではプロキシクラスは必要なくなるので、その参照を削除します。

（プロジェクトからもUser.CacheDirect.csを削除します。）

```
public User.CacheDirect cd;
```

代わりにIRISオブジェクトを宣言します。

```
public IRISObject cd;
```

続いてプロキシクラスが保持していたプロパティ機能を実装するために、以下の宣言を追加します。
（すべてのプロパティに対してプライベート変数とそれにパブリックアクセスするためのアクセッサメソッド）

```
private string p0;  
private string p1;  
private string p2;  
private string p3;  
private string p4;  
private string p5;  
private string p6;  
private string p7;  
private string p8;  
private string p9;  
private string plist;  
private string pdelim;  
private string value;  
private string code;  
private long execflag;  
private string errorname;  
private long error;
```

```
public string P0  
{  
    set { this.p0 = value; }  
    get { return this.p0; }  
}  
public string P1  
{  
    set { this.p1 = value; }  
    get { return this.p1; }  
}  
public string P2  
{  
    set { this.p2 = value; }  
    get { return this.p2; }  
}  
public string P3  
{  
    set { this.p3 = value; }  
    get { return this.p3; }  
}  
public string P4  
{  
    set { this.p4 = value; }  
    get { return this.p4; }  
}  
public string P5  
{  
    set { this.p5 = value; }  
    get { return this.p5; }  
}  
public string P6  
{  
    set { this.p6 = value; }  
    get { return this.p6; }  
}  
public string P7  
{  
    set { this.p7 = value; }  
    get { return this.p7; }  
}
```

```
}  
public string P8  
{  
    set { this.p8 = value; }  
    get { return this.p8; }  
}  
public string P9  
{  
    set { this.p9 = value; }  
    get { return this.p9; }  
}  
public string PLIST  
{  
    set { this.plist = value; }  
    get { return this.plist; }  
}  
public string PDELIM  
{  
    set { this.pdelim = value; }  
    get { return this.pdelim; }  
}  
public string Value  
{  
    set { this.value = value; }  
    get { return this.value; }  
}  
public string Code  
{  
    set { this.code = value; }  
    get { return this.code; }  
}  
public long ExecFlag  
{  
    set {  
        this.execflag = value;  
        if (value == 1) {  
            this.Execute(this.code);  
        }  
    }  
    get { return this.execflag; }  
}  
public string ErrorName  
{  
    get { return this.errorname; }  
}  
public string Error  
{  
    get { return this.error.ToString(); }  
}
```

サーバー接続処理

コンストラクターの処理の所で、サーバー接続処理をIRIS用に変更します。

```
conn = new CacheConnection();  
conn.ConnectionString = constr;  
conn.Open();
```

```
cd = new User.CacheDirect(conn);
```

IRISでは、コネクションオブジェクトを作成した後、プロキシークラスのインスタンスを生成する代わりにIRISクラスのインスタンスを生成し、サーバーのCacheDirect.Emulatorクラスの%Newクラスメソッドを呼び出して、IRISObjectクラスのインスタンスを生成しています。

（.Net Binding版のクラスUser.CacheDirectから名前も変更）このインスタンスが従来のプロキシークラスのインスタンスと同様の機能を提供します。

```
conn.ConnectionString = constr;  
conn.Open();  
IRIS iris = IRIS.CreateIRIS(conn);  
cd = (IRISObject)iris.ClassMethodObject("CacheDirect.Emulator", "%New");
```

プロキシークラスでの実装と異なり、.Net Native APIではプロパティに値を設定するにはIRISObjectクラスのSetメソッドを使って、明示的に値を設定する必要があります。

```
public long Execute(string command)  
{  
    long status;  
  
    cd.Set("P0", p0);  
    cd.Set("P1", p1);  
    cd.Set("P2", p2);  
    cd.Set("P3", p3);  
    cd.Set("P4", p4);  
    cd.Set("P5", p5);  
    cd.Set("P6", p6);  
    cd.Set("P7", p7);  
    cd.Set("P8", p8);  
    cd.Set("P9", p9);  
    cd.Set("PLIST", plist);  
    cd.Set("PDELIM", pdelim)
```

サーバーのインスタンスメソッド（Execute）を呼び出すためには、IRISObjectクラスのInvokeメソッドを呼び出します。

```
status = (long)cd.Invoke("Execute", command);
```

サーバー側のExecuteメソッド実行後に変更された可能性のあるプロパティの値（P0-P9,PLIST,Valueなど）をクライアントの
プロパティに反映させるためにIRISObjectクラスのGetメソッドを呼び出します。

ここでは、サーバー側のプロパティのタイプに関わらず、戻り値によってタイプが動的に変化する可能性があるために戻り値の型をチェックして
適切に処理する必要があります。

```
if (cd.Get("P0") is string)  
{
```

```
p0 = (string)cd.Get("P0");
}
else {
    if (cd.Get("P0") is null)
    {
    }
    else
    {
        p0 = cd.Get("P0").ToString();
    }
}
```

ErrorMessageとErrorもサーバー側のプロパティからGetメソッドを使用して取得します。

```
errorname =(string) cd.Get("ErrorName");
error = (long)cd.Get("Error");
```

PLISTの処理用に追加したメソッドも同様にサーバー側のPLISTプロパティをGetメソッドで取得します。
PLISTプロパティに値を設定するためにSetメソッドを使用します。

```
string[] PLISTArray = cd.Get("PLIST").ToString().Split(cd.Get("PDELIM").ToString().ToCharArray());
cd.Set("PLIST", string.Join(cd.Get("PDELIM").ToString(), PLISTArray));
```

#.NET #InterSystems IRIS

[InterSystems Open Exchangeで関連アプリケーションを確認してください](#)

ソースURL:<https://jp.community.intersystems.com/post/cach%C3%A9-net-binding%E3%82%A2%E3%83%97%E3%83%AA%E3%82%B1%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3%E3%82%92iris%E3%81%AEnet-native-api%E3%82%92%E5%88%A9%E7%94%A8%E3%81%97%E3%81%A6%E6%9B%B8%E3%81%8D%E6%8F%9B%E3%81%88%E3%82%8B%E6%96%B9%E6%B3%95%EF%BC%88%E3%81%9D%E3%81%AF%EF%BC%91%EF%BC%89>