CM-04 データコピー機能

■ 概要

本機能は、あるクラスの各プロパティの値を別のクラスの対応するプロパティに自動的に型変換を行い、コピーする機能を提供する。コピー処理は、DataCopyManager クラスが提供するコピーメソッドを呼び出すことで実施可能である。

なお、「CL-03 イベント処理実行機能」における「画面データ」と DTO 間のコピー処理や、「CL-02 画面遷移機能」における遷移元画面と遷移先画面の「画面データ」間のコピー処理については、各フレームワーク機能が自動的に DataCopyManager クラスのコピーメソッドを実行するため、開発者が明示的にメソッドを呼び出す必要はない。

また、ユーザが定義したクラス(POCO)間だけではなく、ADO.NET で使用する DataSet や EntityObject とのコピーも可能であるため、POCO-DataSet 間や POCO-EntityObject 間のコピー処理を自動化することで DB 周りの開発作業を効率化できる。

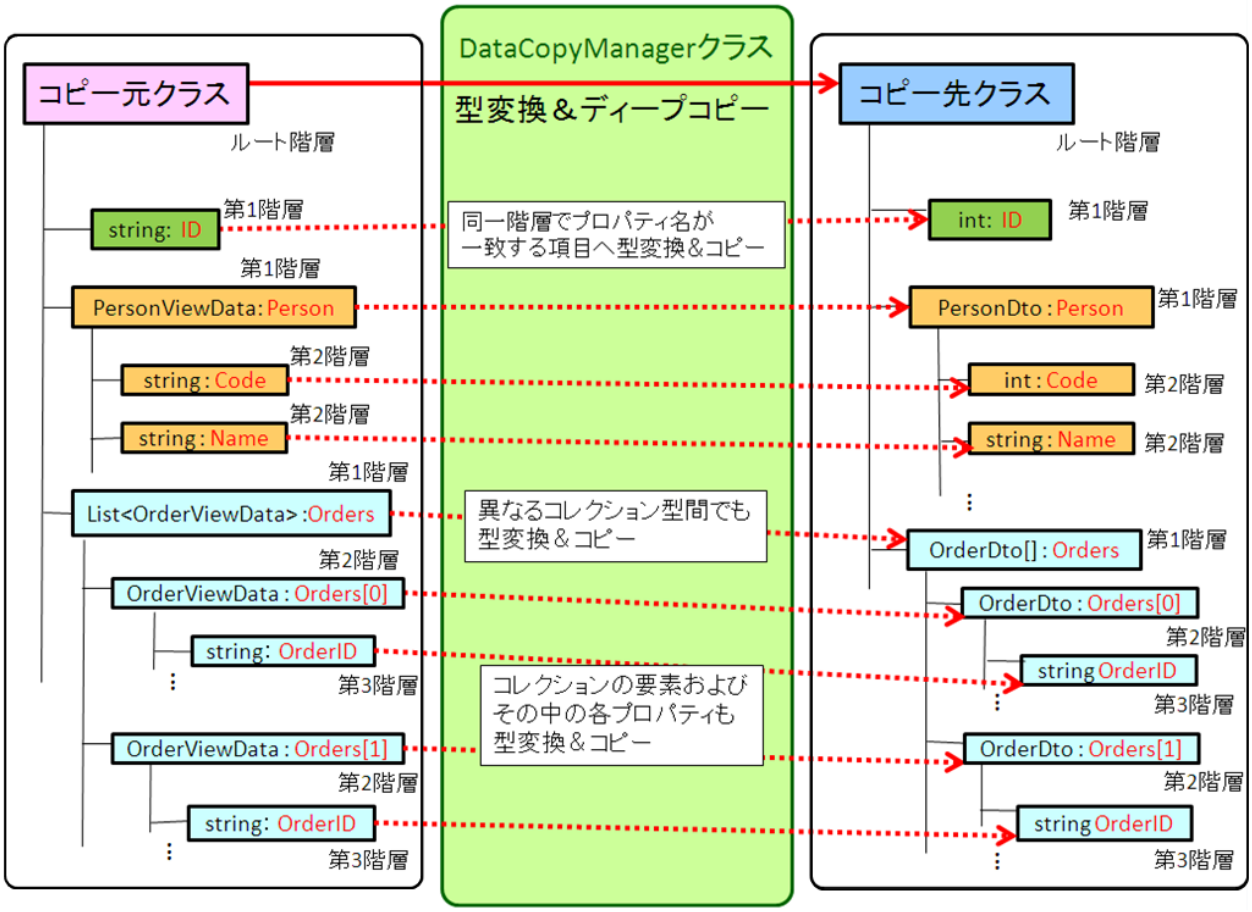


図 1 データコピー機能の動作イメージ

図 1 にあるように、コピー元クラスとコピー先クラスが異なるクラス型であっても、クラスの階層構造を辿り、一定の変換ルールに基づいてディープコピーすることができる。また、コピー先クラスの対象プロパティのインスタンスが生成されていなかったとしても（null の場合でも）、自動的にインタンスを生成しコピーすることができる。

標準提供されるデータコピー機能の主なルールは、以下の通りである。

- コピー元、コピー先のクラス階層構造において、同一階層上にプロパティ名が一致するものが存在すれば、必要に応じて型変換を行い、コピー先のプロパティへコピーする。
- コレクションの場合は、コピー元の要素およびその中のプロパティをコピー先へコピーする。また List と配列といった異なるコレクション間でも型変換＆コピーができる。
- 名称や階層が異なるプロパティ同士でも、マッピング定義を追加すれば型変換&コピーが可能である。

このように、フレームワークが一定のルールをもとに広範囲にわたって自動的にマッピング処理を実施することで、データコピー処理にかかる開発者の負担を大幅に軽減することができる。

■ 使用方法

◆ FW 構成ファイル(TerasolunaFramework.config)の設定

本機能を利用するためには、FW 構成ファイル（TerasolunaFramework.config）の /configuration/unity/containers/container/extensions/add タグで、以下の UnityContainerExtension 継承クラスを記述する。

- 本機能をクライアント AP で利用する場合
  - Terasoluna.Windows.ViewModel.Validation.ValidatableViewDataExtension クラス
    - 本機能を利用するための Extension クラス。
    - クライアント AP 用に「画面データ」のコピー処理を考慮した実装になっている。

- 本機能をサーバ AP で利用する場合
  - Terasoluna.DataCopy.DataCopyExtension クラス
    - 本機能を利用するためのデフォルトの Extension クラス。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <unity>
    ・・・
    <containers>
      <container>
        <types>
          ・・・
        </types>
        <extensions>
          <!-- データ変換機能の設定-->
          <add
          type="Terasoluna.Windows.ViewModel.Validation.DataCopy.ValidatableDataCopyExtension,
                Terasoluna.Windows.ViewModel.Validation" />
          ・・・
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 1 クライアント AP における TerasolunaFramework.config の設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <unity>
    ・・・
    <containers>
      <container>
        <types>
          ・・・
        </types>
        <extensions>
          <!-- データ変換機能の設定-->
          <add type="Terasoluna.DataCopy.DataCopyExtension, Terasoluna" />
          ・・・
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 2 サーバ AP における TerasolunaFramework.config の設定例

◆ データコピーの実行

(1) データコピーメソッド

開発者がビジネスロジッククラス等で明示的にデータコピー処理を呼び出して実装したい場合は、DataCopyManager のコピーメソッドを呼び出す。

なお、「CL-03 イベント処理実行機能」、「CL-02 画面遷移機能」では、開発者が DataCopyManager のコピーメソッドを呼び出す必要はない。

表 1 DataCopyManager のコピーメソッドの一覧

項番	メソッドの種類	型パラメータ	第 1 引数	第 2 引数	第3引数
1	public static void Copy(object source, object destination)	-	コピー元のオブジェクト	コピー先のオブジェクト	-
2	public static void Copy<TSource, TDestination>(TSource source, TDestination destination, IMappingInfo mapping)	TSource: コピー元の型 TDestination: コピー先の型	コピー元のオブジェクト	コピー先のオブジェクト	マッピング情報
3	public static TDestination CreateCopy<TSource, TDestination>(TSource source, IMappingInfo mapping)	TSource: コピー元の型 TDestination: コピー先の型	コピー元のオブジェクト	マッピング情報	
4	public static object CreateCopy<TSource>(TSource source, Type destinationType, IMappingInfo mapping)	TSource: コピー元の型 TDestination: コピー先の型	コピー元のオブジェクト	コピー先の型	マッピング情報

項番 1,2 の Copy メソッドは、コピー先のインスタンスが必要であるが、項番 3,4 の CreateCopy メソッドは、指定した型情報を元にコピー先のインスタンスも生成する。

引数として、マッピング情報があるメソッドは、本機能の標準マッピングルールでは自動コピーされないコピールールの指定や自動コピーしてほしくない項目の指定などを設定する場合に使用する。

(2) マッピング情報

マッピング情報は、IMappingInfo インタフェースの実装クラスである MappingInfo クラスを作成し、以下のプロパティに個別のマッピングルールを IList.Add メソッドで追加していく。もしくは、MappingInfo クラスのコンストラクタにより、各プロパティの値を一度に設定する。

表 2 MappingInfo クラスのプロパティ

項番	プロパティ	説明
1	IList<MappingItem> Mappings { get; }	コピー元のプロパティパスとコピー先のプロパティパスの対応関係を表す MappingItem を保持するリスト。

2	IList<string> SourceTargetPropertyPaths { get; }	コピー対象とする、コピー元のプロパティパスを保持するリスト。本プロパティに要素を追加した場合、追加されたプロパティのみコピー対象となる。
3	IList<string> SourceIgnorePropertyPaths { get; }	コピー対象から外す、コピー元のプロパティパスを保持するリスト。
4	IList<string> DestinationTargetPropertyPaths { get; }	コピー対象とする、コピー先のプロパティパスを保持するリスト。本プロパティに要素を追加した場合、追加されたプロパティのみコピー対象となる。
5	List<string> DestinationIgnorePropertyPaths { get; }	コピー対象から外す、コピー元のプロパティパスを保持するリスト。

なお、Mappings プロパティに関しては、以下のメソッドを使用しても追加可能である。

表 3　MappingInfo クラスのメソッド

項番	メソッド	第 1 引数	第 2 引数	説明
1	public void AddMapping(string sourcePropertyPath, string destinationPropertyPath)	コピー元のプロパティパス	コピー先のプロパティパス	第1引数で指定したプロパティパスから第2引数で指定したプロパティパスへコピーするようマッピング定義を追加する。

Mappings プロパティに設定する「プロパティパス」は、半角ピリオド「.」を使ってクラスの階層構造を表すことができる。例えば、あるクラスが PersonDto 型の Person プロパティを持ち、PersonDto 型には Name というプロパティを持っている場合、プロパティパスは「Person.Name」になる。プロパティパスの記述方法など、マッピング情報を用いたコピー動作の詳細な仕様については、後述する「標準のデータコピールール」を参照のこと。

(3) 実装例

以下に、DataCopyManager クラスを使用した実装例を示す。
この例では、DestData クラスのインスタンスを生成後、SourceData クラスのインスタンスの各プロパティ値を、DestData インスタンスの対応するプロパティに自動コピーしている。
コピー元である SourceData クラスのコードとクラス図は以下の通り。

```
public class SourceData
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string PersonCode { get; set; }
    public List<OrderViewData> Orders { get; set; }

    public void Write()
    {
        Console.WriteLine("【コピー元クラス】");
        Console.WriteLine("ID:" + ID);
        Console.WriteLine("PersonCode:" + PersonCode);
        Console.WriteLine("Name:" + Name);
        for (int i = 0; i < Orders.Count; i++)
        {
            Console.WriteLine("Orders[" + i + "].OrderID:" + Orders[i].OrderID);
        }
    }
}

public class OrderViewData
{
    public string OrderID { get; set; }
}
```

リスト 3 コピー元のクラス(SourceData クラス)

図 2 コピー元のクラス(SourceData クラス)のクラス図

コピー先である DestData クラスのコードとクラス図は以下の通り。

```
public class DestData
{
      public int ID { get; set; }
      public PersonDto Person { get; set; }
      public OrderDto[] Orders { get; set; }

      public void Write()
      {
            Console.WriteLine("【コピー先クラス】");
            Console.WriteLine("ID:" + ID);
            Console.WriteLine("Person.Code:" + Person.Code);
            Console.WriteLine("Person.Name:" + Person.Name);
            for (int i = 0; i < Orders.Length; i++)
            {
                  Console.WriteLine("Orders[" + i + "].OrderID:" + Orders[i].OrderID);
            }
      }
 }

public class PersonDto
{
      public int Code { get; set; }
      public string Name { get; set; }
}

public class OrderDto
{
      public int OrderID { get; set; }
}
```

リスト 4 コピー先のクラス(DestData クラス)

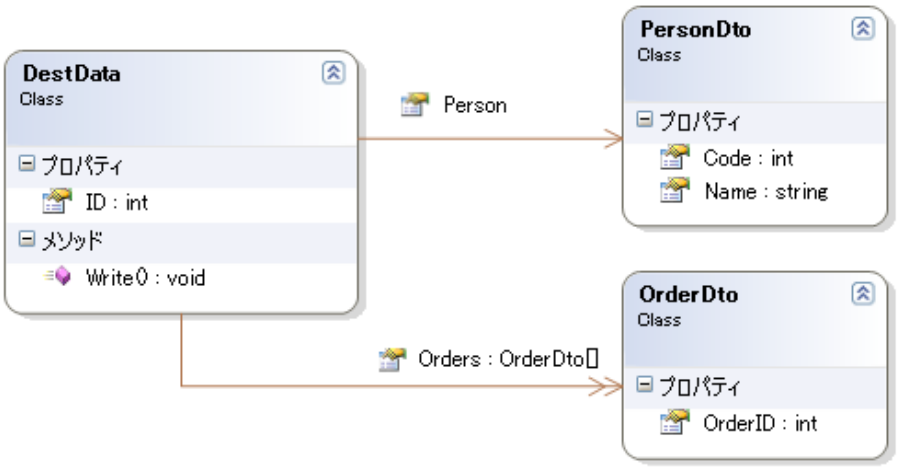


図 3 コピー先のクラス(DestData クラス)のクラス図

このサンプルプログラムでは、コピー元のクラスとコピー先のクラスは、以下のようにコピーすることとする。なお、表中の「コピー要否」は、該当プロパティのコピー要件(コピーしたいかどうか)を表し、「自動マッピング」は、デフォルト動作において、自動コピーが実行されるかどうかを表す。つまり、

「コピー要否」の業務要件を満たすために、デフォルト動作である「自動マッピング」で足りないものを「MappingInfo の追加ルール」でルール追加することにより充足する、という意味である。

表 4 サンプルプログラムのデータコピーの仕様

項番	SourceData クラス	DestData クラス		コピー要否	自動マッピング	MappingInfo の追加ルール
1	string：ID	int：ID		×	○	「“ID”をコピーしない」
2	string：PersonCode	PersonDto: Person	int: Code	×	×	-
3	string：Name		string: Name	○	×	Name⇒ Person.Name
4	List<OrderViewData>: Orders	OrderDto[]：Orders		○	○	-
	Order[0]	Order[0]		○	○	-
	Order[1]	Order[1]		○	○	-
	Order[2]	Order[2]				

を上記例の記述例を以下に示す。[1]

[1]実際に下記サンプルコードの Test メソッドを実行するためには、Test メソッドの実行前後で、「CM-01 アプリケーション起動・終了機能」が提供する TerasolunaFramework クラスによるフレームワークの起動・終了処理が必要であるが省略している。

```
public class DataCopyManagerTest
{
    public static void Test()
    {
        ///コピー元データの作成
        SourceData src = CreateSourceData();
        ///コピー先データの作成
        DestData dest = new DestData();

        ///MappingInfoの作成
        MappingInfo mappingInfo = new MappingInfo();
        ///DestinationIgnorePropertyPathsプロパティで
        ///コピー先のIDプロパティへ自動コピーしないように設定（要素を追加）
        mappingInfo.DestinationIgnorePropertyPaths.Add("ID");
        ///AddMappingメソッドで
        ///コピー元のプロパティパスとコピー先のプロパティパスをマッピング定義
        mappingInfo.AddMapping("Name", "Person.Name");
        ///データコピー
        DataCopyManager.Copy(src, dest, mappingInfo);

        ///結果出力
        src.Write();
        dest.Write();
    }

    ///コピー元のデータ作成
    private static SourceData CreateSourceData()
    {
        return new SourceData()
        {
            ID = "1234",
            Name = "Taro",
            PersonCode = "5678",
            Orders = new List<OrderViewData>()
            {
                new OrderViewData()
                {
                    OrderID = "1001"
                },
                new OrderViewData()
                {
                    OrderID = "1002"
                },
                new OrderViewData()
                {
                    OrderID = "1003"
                }
            }
        };
    }
}
```

リスト 5 データコピー機能の実行例

サンプルプログラムの実行結果を以下に示す。

SourceData クラスの値が、DestData クラスの適切なプロパティへ型変換されてコピーされていることが分かる。また、MappingInfo で指定したマッピングルール(「"ID"プロパティをコピーしない」、「Name プロパティを Person.Name プロパティにコピー」)が適用されていることが分かる。

```
【コピー元クラス】
ID:1234
PersonCode:5678
Name:Taro
Orders[0].OrderID:1001
Orders[1].OrderID:1002
Orders[2].OrderID:1003
【コピー先クラス】
ID:0
Person.Code:0
Person.Name:Taro
Orders[0].OrderID:1001
Orders[1].OrderID:1002
Orders[2].OrderID:1003
```

リスト 6 サンプルプログラムの実行結果

◆ イベント処理実行機能や画面遷移機能におけるマッピングルールの設定

「CL-03 イベント処理実行機能」や「CL-02 画面遷移機能」から本機能を利用する場合は、MappingInfo クラスのプロパティやメソッドを使って個別のマッピングルールを「コーディング」する必要はなく、Visual Studio の「デザイナ」で同等の設定をすることができる。両機能からの本機能の利用方法の詳細は、「CL-03 イベント処理実行機能」及び「CL-02 画面遷移機能」を参照のこと。

■ 標準のデータコピールール

◆ データタイプ

本機能では、同一のクラス階層上でプロパティ名が一致するデータ項目間や、個別にマッピングルールを定義したデータ項目間について、コピー処理を実施する。しかし、マッピングされたデータ間であっても、データの構造が異なる場合コピーはできない。データ項目はその構造の違いから、「シンプルタイプ」「コンプレックスタイプ」「コレクションタイプ」の 3 つに分類される。

表 5 3 つのデータタイプ

項番	データタイプ名	説明
1	シンプルタイプ	単一値をもつデータ型。string 型や int 型の項目が該当する。
2	コンプレックスタイプ	構造をもつデータ型。「画面データ」や通信データが該当する。
3	コレクションタイプ	シンプルタイプ及びコンプレックスタイプの集合。List や配列が該当する。

本機能は、同じ「データタイプ」間でのみコピー可能である。
よって、シンプルタイプ-コンプレックスタイプ間、コンプレックスタイプ-コレクションタイプ間といった異なる「データタイプ」間ではコピーできない。

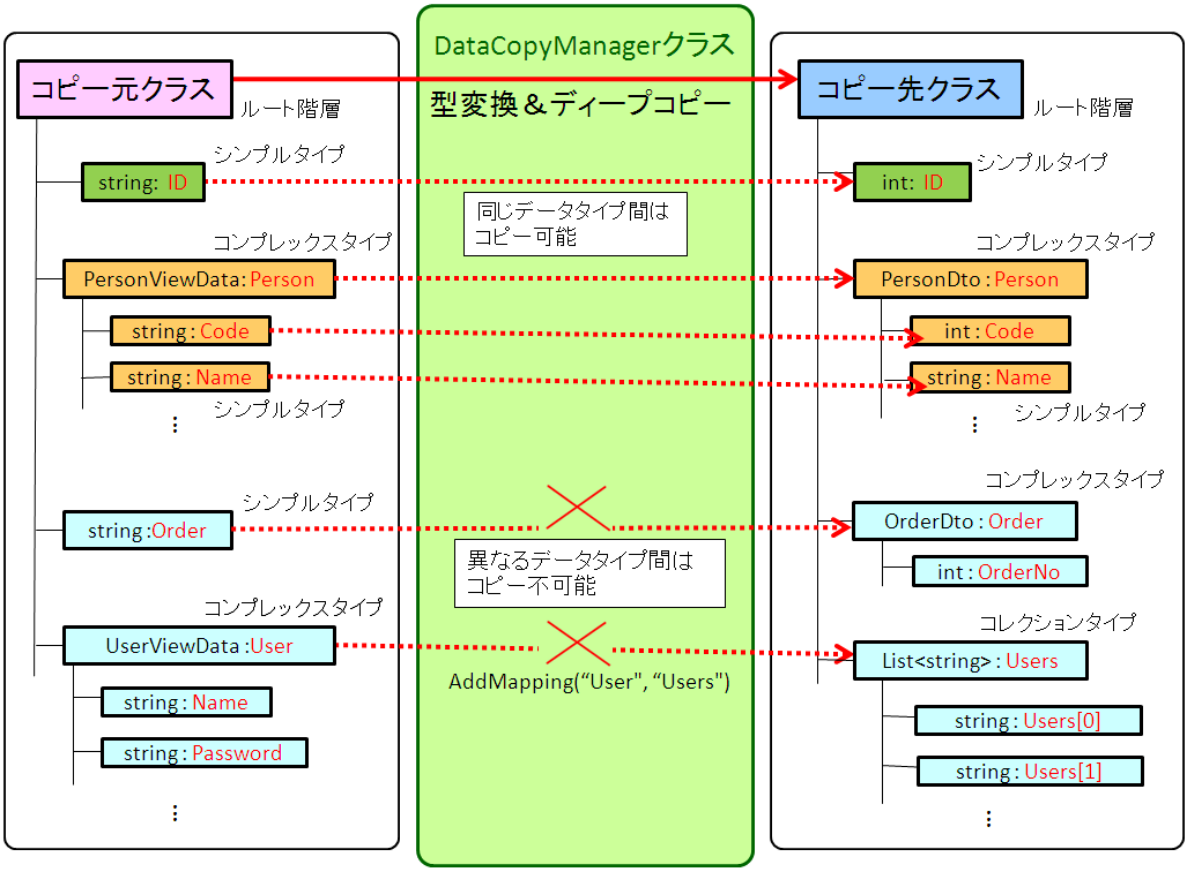


図 4 3 つのデータタイプとコピールール

◆ 型変換

コピー元とコピー先のプロパティの型が異なる場合（string 型と int 型など）、自動的に型変換してからコピーする。型変換には System.ComponentModel.TypeConverter クラスを利用する。
型変換に失敗した場合は例外をスローし、データコピー処理は失敗する。
TypeConverter クラスについては、MSDN のドキュメント[2]を参照のこと。

◆ コピー先クラスの実装

本機能による「インスタンスの自動生成」の機能を利用する場合、コピー先クラスには「既定のコンストラクタ」が必須であり、既定のコンストラクタがないクラスに対してコピー処理を実行すると、例外が発生する。
既定のコンストラクタとは、引数をとらないインスタンスコンストラクタである。既定のコンストラクタは、コンストラクタが1つも宣言されていない場合にはコンパイル時に自動生成されるが、1 つでもコンストラクタが存在する場合は自動生成されない。コピー先クラスの定義に引数をとるコンストラクタが宣言されている場合、既定のコンストラクタも必ず明示的に宣言すること。既定のコンストラクタについては言語仕様や MSDN ドキュメント[3]を参照のこと。
なお、常に「既存のインスタンス」が利用される場合など、インスタンス自動生成の必要がない場合は、この限りではない。

◆ コピー先クラスのインスタンス生成ルール

本機能の Copy メソッドでは、コピー先クラスのコピー対象プロパティのインスタンスが生成されていなかったとしても、自動的にインタンスを生成しコピーを実行することが可能である。このコピー先プロパティのインスタンスは、基本的にはインスタンスの有無や型によって以下のルールに従って扱われる。

- コピー先のプロパティに既に生成されたインスタンスがセットされている場合は、そのインスタンスをそのまま利用する。
- コピー先のプロパティが null の場合、コピー先プロパティの型のインスタンスを生成しセットする。なお、コピー先のインスタンスそのものが null の場合は、インスタンスを自動生成することはできないので注意すること。
- コピー先のプロパティが null で、かつインスタンス生成できない型（インタフェース、抽象クラス）で定義されている場合、コピー元のプロパティの型がその継承クラスであれば、コピー元の型でインスタンスを生成しセットする。
- コピー先のプロパティが null でかつ、コピー先プロパティの型が IList 型ならば List<T>型

---

[2] TypeConverter クラス(MSDN):
http://msdn.microsoft.com/ja-jp/library/system.componentmodel.typeconverter.aspx
[3] 既定のコンストラクタ(MSDN):
http://msdn.microsoft.com/ja-jp/library/aa645608(VS.71).aspx

のインスタンスを生成する。

その他、プロパティに付与された属性や型のプロパティによって、上記の基本ルールに加えて細かいコピールールが存在する。データコピーの詳細なルールについては後述の「コピールール（詳細）コピールール（詳細）」を参照のこと。

◆ マッピング情報の設定によるマッピングルール

本機能では、マッピング情報(IMappingInfo インタフェース)を設定することで、標準のルールでは自動的にコピーされないプロパティ間のマッピングや、コピー対象となるプロパティの設定などが可能となる。マッピング情報は、MappingInfo クラスのプロパティとメソッドを使用して設定する。

表 6 MappingInfo のプロパティ(再掲)

項番	プロパティ	説明
1	IList<MappingItem> Mappings { get; }	コピー元のプロパティパスとコピー先のプロパティパスの対応関係を表す MappingItem を保持するリスト。
2	IList<string> SourceTargetPropertyPaths { get; }	コピー対象とする、コピー元のプロパティパスを保持するリスト。本プロパティに要素を追加した場合、追加されたプロパティのみコピー対象となる。
3	IList<string> SourceIgnorePropertyPaths { get; }	コピー対象から外す、コピー元のプロパティパスを保持するリスト。
4	IList<string> DestinationTargetPropertyPaths { get; }	コピー対象とする、コピー先のプロパティパスを保持するリスト。本プロパティに要素を追加した場合、追加されたプロパティのみコピー対象となる。
5	List<string> DestinationIgnorePropertyPaths { get; }	コピー対象から外す、コピー元のプロパティパスを保持するリスト。

表 7 MappingInfo のメソッド(再掲)

項番	メソッド名	第 1 引数	第 2 引数	説明
1	public void AddMapping(string sourcePropertyPath, string destinationPropertyPath)	コピー元のプロパティパス	コピー先のプロパティパス	第1引数で指定したプロパティから第2引数で指定したプロパティへコピーするようマッピング定義を追加する。

マッピング情報のそれぞれの設定内容は、相互に関連し合ってコピー処理の動作を決定する。また、設定の中でコレクションタイプのプロパティを扱う場合には、一定の記述ルールが存在する。ここでは、これら 2 点について解説する。

(1) マッピング情報を利用したコピー処理

本機能のコピー処理は、マッピング情報の設定に基づき、以下のステップで実行される。

1. ルート階層からの自動コピーが実行される。
2. SourceTargetPropertyPaths と、DestinationTargetPropertyPaths に 1 つでも要素が追加されている場合は、追加されたプロパティのみがコピーされ、その他のプロパティは自動コピーされなくなる(デフォルト動作でコピーされるはずのプロパティも、対象外となる)。
3. SourceIgnorePropertyPaths と、DestinationIgnorePropertyPaths に追加されたプロパティは、デフォルト動作でコピー対象となるプロパティであっても、コピーされない。
4. Mappings に指定されたマッピング定義に基づく自動コピーが実行される。このコピーはルート

階層からの自動コピー(上記「1」)の結果を上書きする形で実行される。なお、マッピング定義の階層構造下においても、Target/Ignore の設定は同様に評価される。

5. Mappings に指定されたマッピング定義が複数ある場合、上から順に実行され、上書き処理が繰り返される。

以上のルールをサンプルプログラムで確認する。サンプルのクラスとして用いる、コピー元の SourceData クラスとコピー先の DestData クラスのコードとクラス図は以下の通り。

```
public class SourceData
{
    public string ID { get; set; }
    public string Password { get; set; }
    public PersonViewData Person { get; set; }
    ///コンストラクタで初期値設定
    public SourceData()
    {
        ID = "1234";
        Password = "****";
        Person = new PersonViewData()
        {
            Code = "5678" ,
            Name = "Taro"
        };
    }
}

public class PersonViewData
{
    public string Code { get; set; }
    public string Name { get; set; }
}
```

リスト 7 コピー元のクラス(SourceData クラス)

図 5 コピー元のクラス(SourceData クラス)のクラス図

```
public class DestData
{
    public string ID { get; set; }
    public string Pass { get; set; }
    public PersonDto Person { get; set; }
}

public class PersonDto
{
    public int Code { get; set; }
    public string Name { get; set; }
    public int SectionCode { get; set; }
}
```

リスト 8 コピー先のクラス(DestData クラス)

図 6 コピー先のクラス(DestData クラス)のクラス図

以下に、マッピング情報の設定に対して具体例を用いてマッピングルールの仕様を説明する。MappingInfo の設定例を表 8 に示す。また、MappingInfo の設定に対してデータコピー処理実行結果としてコピー先 DestData の状態を表 9 に示す。

表 8 MappingInfo の設定例(各プロパティ名末尾の「PropertyPaths」は省略)

項番	AddMapping メソッド (Mappings)		Source Target	Source Ignore	Destination Target	Destination Ignore
	第 1 引数	第 2 引数				
1						
2			ID			
3			AAA			
4			ID	ID		
5				AAA		
6					Person	
7			Person		ID	
8			Person			Person.Code
9	Password	Pass				
10	Password	AAA				
11	AAA	Pass				
12	ID	Person.SectionCode				
13	Password	Person.SectionCode				
14	ID	Person.Code				Person.Code
15	ID	Person.Code		ID		

16	ID	Person.Code	Person			
17	Password	Pass				
	ID	Pass				
18	Password	Person.SectionCode				
	ID	Person.SectionCode				

表 9 データコピー実行後の DestData の値

項番	DestData オブジェクトのプロパティパス						データコピー仕様の解説
	ID	Pass	Person	Person.Code	Person.Name	Person.SectionCode	
1	1234	null		5678	Taro	0	標準の自動コピー
2	1234	null	null				SourceTarget に指定したもののみコピーされる
3	0	null	nul				Target に存在しないプロパティパスを指定した結果、何もコピーされない
4	0	null	null				Ignore に指定したものは、Target に含まれていてもコピーされない
5	1234	null		5678	Taro	0	Ignore に存在しないプロパティパスを指定しても、コピーには影響しない
6	0	null			Taro	0	DestinationTarget に指定したもののみコピーされる
7	0	null	null				SourceTarget と DestinationTarget で共通項がないので、何もコピーされない
8	0	null		0	Taro	0	コンプレックスタイプの一部のプロパティのみ指定することもできる
9	1234	****		5678	Taro	0	自動コピーの後に、Mappings で指定したコピーが実行される
10	例外発生						Mappings に「存在しないプロパティパス」を指定した場合、例外が発生する
11	例外発生						
12	1234	null		5678	Taro	1234	異なる型のプロパティ同士でも、型変換できる場合はコピーされる
13	例外発生						異なる型のプロパティ同士で、型変換できない場合は例外が発生する
14	1234	null		0	Taro	0	Mappings に指定していても、Ignore に指定されているのでコピーされない
15	0	null		5678	Taro	0	
16	0	null		5678	Taro	0	Mappings に指定していても、Target に含まれていないのでコピーされない
17	1234	1234		5678	Taro	0	複数の Mappings による上書き処理の結果、より後に定義されたものが残る
18	例外発生						Mappings 定義の中に 1 つでも不正な定義が含まれていると、例外が発生する

(2) コレクションを扱う場合のマッピング情報の設定

コレクションタイプを扱う場合、マッピング情報のプロパティパスには、要素の全てを意味する半角角括弧「[]」を使用する。半角角括弧を用いたプロパティパスの記述ルールを以下に示す。

- 「[]」でコレクション要素の全てを意味する。
- 「[0]」のようにインデックスを指定した記述はできない。
- 「Persons[].Name」のように、コンプレックスタイプのコレクションの場合は、要素となるデータ型のプロパティ単位で指定することができる。
- Mappings プロパティ（AddMapping メソッド）に指定する場合、コピー元とコピー先に含まれる「[]」の数は同じであり、かつ 1 つまでとする（必ず 1 つずつ）。

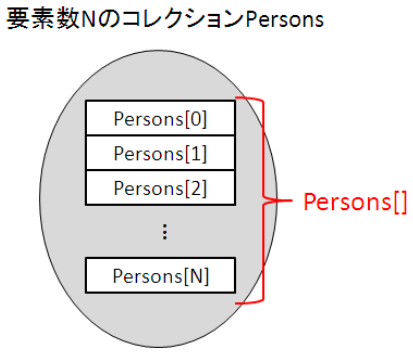


図 7 コレクションの全要素を表すプロパティパス

以上のルールをサンプルプログラムで確認する。サンプルのクラスとして用いる、コピー元の SourceData クラスとコピー先の DestData クラスのコードとクラス図は以下の通り。

```
public class SourceData
{
    public List<string> IDs { get; set; }
    public List<PersonViewData> Persons { get; set; }
    ///コンストラクタで初期値設定
    public SourceData()
    {
        IDs = new List<string>()
        {
            "1001",
            "1002",
            "1003"
        };
        Persons = new List<PersonViewData>()
        {
            new PersonViewData()
            {
                Code = "2001",
                Name = "Taro"
            },
            new PersonViewData()
            {
                Code = "2002",
                Name = "Hanako"
            },
            new PersonViewData()
            {
                Code = "2003",
                Name = "TJiro"
            }
        };
    }
}

public class PersonViewData
{
    public string Code { get; set; }
    public string Name { get; set; }
}
```

リスト 9 コピー元のクラス(SourceData クラス)

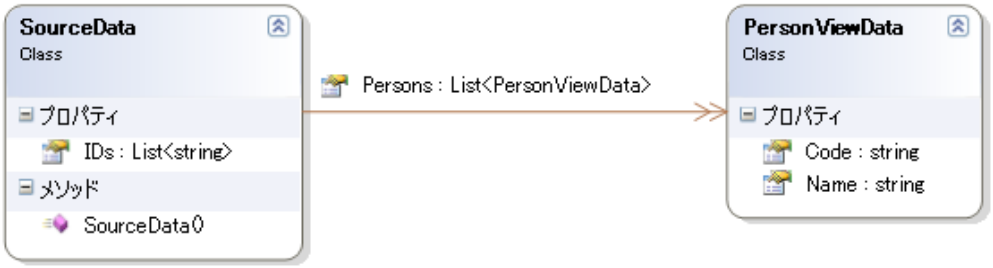


図 8 コピー元のクラス(SourceData クラス)のクラス図

```
public class DestData
{
    public string[] IDs { get; set; }
    public PersonDto[] Persons { get; set; }
}

public class PersonDto
{
    public string Code { get; set; }
    public string Name { get; set; }
    public int SectionCode { get; set; }
}
```

リスト 10 コピー先のクラス(DestData クラス)

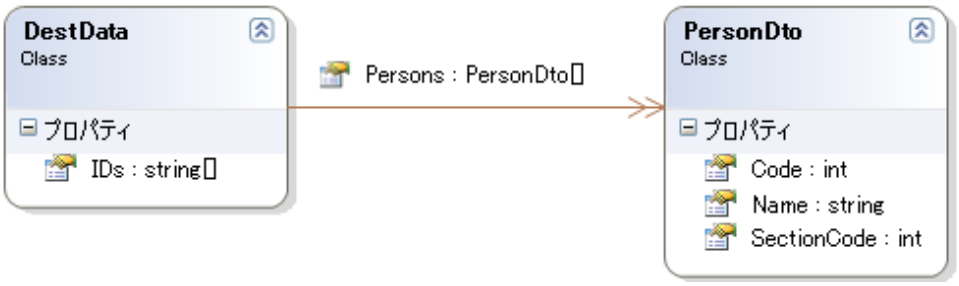


図 9 コピー先のクラス(DestData クラス)のクラス図

以下に、マッピング情報の設定に対して具体例を用いてマッピングルールの仕様を説明する。MappingInfo の設定例を表 10 に示す。また、MappingInfo の設定に対してデータコピー処理実行結果としてコピー先 DestData の状態を表 9 に示す。

表 10 MappingInfo の設定例(各プロパティ名末尾の「PropertyPaths」は省略)

項番	AddMapping メソッド (Mappings)		Source Target	Source Ignore	Destination Target	Destination Ignore
	第 1 引数	第 2 引数				
1						
2			IDs			
3				IDs		
4					Persons	Persons
5			IDs[]			
6			IDs[0]			
7			Persons[].Code			
8			Persons[].Code Persons[].Name			
9			Persons[].Code Persons[].Name			Persons[].Code
10			Persons[].Code Persons[].Name			Persons[0].Code
11	IDs[]	Persons[].SectionCode				

12	IDs[]	Persons.SectionCode				
13	IDs[] IDs[]	Persons[].SectionCode Persons[].Code				
14	IDs[].A[]	Persons[].B[]				

表 11 データコピー実行後の DestData の値

項番	プロパティパス					解説
	IDs[]	Persons	Persons[].Code	Persons[].Name	Persons[].SectionCode	
1	[0]:1001 [1]:1002 [2]:1003		[0]:2001 [1]:2002 [2]:2003	[0]:Taro [1]:Hanako [2]:Jiro	[0]:0 [1]:0 [2]:0	標準の自動コピー
2	[0]:1001 [1]:1002 [2]:1003	null				SourceTarget に指定したもののみコピーされる(コレクションも同様)
3	null		[0]:2001 [1]:2002 [2]:2003	[0]:Taro [1]:Hanako [2]:Jiro	[0]:0 [1]:0 [2]:0	Ignore に指定したものはコピーされない(コレクションも同様)
4	null	null				Ignore に指定したものは、Target に含まれていてもコピーされない
5	[0]:1001 [1]:1002 [2]:1003	null				「[]」で、指定したコレクションの要素の全てをコピー
6	例外発生					インデックスを指定した記述は、記述ルール違反で例外が発生する
7	null		[0]:2001 [1]:2002 [2]:2003	[0]:null [1]:null [2]:null	[0]:0 [1]:0 [2]:0	コンプレックスタイプのコレクションでは、型のプロパティ単位でコピーできる
8	null		[0]:2001 [1]:2002 [2]:2003	[0]:Taro [1]:Hanako [2]:Jiro	[0]:0 [1]:0 [2]:0	
9	null		[0]:0 [1]:0 [2]:0	[0]:Taro [1]:Hanako [2]:Jiro	[0]:0 [1]:0 [2]:0	
10	例外発生					1 つでも記述ルール違反が含まれていると例外が発生する
11	[0]:1001 [1]:1002 [2]:1003		[0]:0 [1]:0 [2]:0	[0]:null [1]:null [2]:null	[0]:1001 [1]:1002 [2]:1003	コンプレックスタイプのプロパティがシンプルタイプなので、シンプルタイプのコレクション同士のコピーとなる
12	例外発生					Mappings で指定するパス中の「[]」の数が異なる場合、例外が発生する

13	[0]:1001 [1]:1002 [2]:1003		[0]:1001 [1]:1002 [2]:1003	[0]:null [1]:null [2]:null	[0]:1001 [1]:1002 [2]:1003	自動コピーの後に Mappings で指定したコピーが実行されるので、自動コピーの結果を上書き(コレクションも同様)
14	例外発生					Mappings で指定できる「[]」の数は、それぞれ 1 つまで

この他にも、書き込み不可の定義があるプロパティや PK をもつ DataTable 型のプロパティにおいては基本ルール以外に細かいコピールールが存在する。データコピーの詳細なルールについては、後述の「コピールール(詳細)」を参照のこと。

■ TIPS

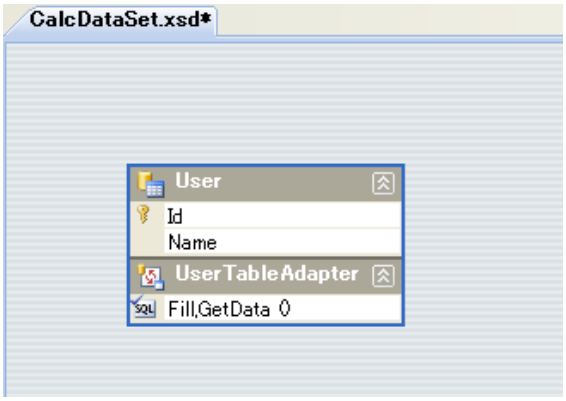
以下では、本機能を利用する際、開発者がよく直面する問題について対処方法をまとめている。開発時に実装方法に困った場合に参考にするとよい。

◆ POCO と DataSet 間のコピー

コピー処理対象のオブジェクトは、ユーザが定義したクラス(POCO)間だけではなく、ADO.NET で使用する DataSet や EntityObject とのコピーも可能であるため、POCO-DataSet 間や POCO-EntityObject 間のコピー処理を自動化することで DB 周りの開発作業を効率化できる。

```
public class SampleDto
{
    public List<UserDto> User { get; set; }
}
public class UserDto
{
    public string Id { get; set; }
    public string Name { get; set; }
}
```

リスト 11 POCO の例



リスト 12 型付き DataSet の例

```
using (TransactionScope scope = new TransactionScope())
{
        UserTableAdapter ta = new UserTableAdapter();
        CalcDataSet ds = new CalcDataSet();
        ta.Fill(ds.User);
        //POCO と DataSet 間のコピーの例
        SampleDto copiedData = DataCopyManager.CreateCopy<SampleDto>(ds);
}
```

リスト 13 POCO と DataSet 間のデータコピー処理の実装例

◆ クライアント DTO で生成される「*Specified Flags」[4]の対処

.NET(WCF)クライアント-Java(JAX-WS)サーバ間の通信の場合、Java 側の Web サービスメソッドの DTO に、Integer 型、Date 型の要素が含まれていると、WSDL より生成された.NET 側の DTO の要素として、対象プロパティ名に、接尾語として「Specified」が付与された bool 型のプロパティが併せて生成される。
これは「*Specified Flags」と呼ばれるもので、.NET の値型（ ValueType ）における null 表現である。例えば Java 側の Integer 型が null 値の代入を許容する型であるのに対して、対応する.NET 側の int(Int32)型は値型であるため null 値を代入できない。このため、.NET 側で対象プロパティの「*Specified Flags」を false に設定することで、Java 側の Integer 型への null 値代入処理を表現できるようになっている。

```
public DateTime DateTimeValue {get; set;}
public bool DateTimeValueSpecified {get; set;}
```

リスト 14　*Specified Flags の例

ここで、「*Specified Flags」が生成された DTO の対象プロパティでは、「*Specified Flags」を true に設定しないと、サーバ側にデータが送信されなくなる（null 設定される）ので注意が必要である。
「*Specified Flags」はデフォルトでは false に設定されており、そのままデータを送信すると、上述したとおり、サーバ側の対象プロパティには null 値が設定されてしまう。
つまり、対象プロパティに値が入力されている場合には「*Specified Flags」を true に、対象プロパティの値が未入力の場合は、「*Specified Flags」を false にそれぞれ設定する必要がある。
そこで、本機能によるデータコピー処理の際、対象プロパティに「*Specified Flags」が存在する場合において、値が入力されている場合にはフラグを true に、未入力の場合にはフラグを false に自動的に設定する。
これにより、開発者による「*Specified Flags」の設定は不要となり、設定漏れによるサーバへのデータ送信漏れを防止することができる。

---

[4] Schema Files
http://msdn.microsoft.com/en-us/library/cc500289.aspx

■ 内部構成

◆ コピールール（詳細）

本機能のデータコピールールについて、図 10 のマインドマップに示す。なお、通常の利用方法ではこれれの細かいルールを意識する必要はない。インタフェースで型を定義したり、自動生成のルールを細かく規定したい場合や、アーキテクトが内部の細かい仕様を押さえたい場合などに、適宜参照する。
データコピーは、以下の3つのルールに分けて、整理している。

- インスタンス化ルール
  - コピー先オブジェクトのインスタンス生成ルール。コピー先オブジェクトやそのオブジェクトプロパティにインスタンスがない場合、図 10 に記述した条件に基づきインスタンス生成する。なお、Copy メソッドでコピー先オブジェクトそのものが null の場合、インスタンス化はできない（コピー先オブジェクトのオブジェクトプロパティはインスタンス生成可能）。オブジェクト自体を生成したい場合は、CreateCopy メソッドを利用する。
- コピールール
  - コピー元オブジェクトのプロパティからコピー先オブジェクトのプロパティへコピーする際のルール。図 10 に記述した条件に基づき、型変換処理を実施し値をコピーする。
- マッピングルール
  - コピー元オブジェクトとコピー先オブジェクト間で、どのプロパティ同士をコピー対象とするかどうかのマッピングルール。図 10 に記述した条件に基づき、コピー対象を決定する。

なお、図 10 の「インスタンス化ルール」の中で「コピー先にインスタンスがない」場合に、「プロパティパスにコピー先が登録されている」場合には「登録されている型でインスタンス生成」するといったルールが存在する。これにより、コピー先のインスタンスがなく、かつ対象プロパティのデータ型がインタフェースであるため本来インスタンス生成＆コピーできないケースであっても、プロパティパスに対してインスタンス生成する型を登録しておくことで、データコピーが可能となる。
「プロパティパスにコピー先を登録」するには、以下のように実装する。

- DataCopyManager.CreateCopyContext メソッドで IDataCopyContext オブジェクトを取得する。
- IDataCopyContext オブジェクトの「Source.EntryManager.RegisterRootEntry」メソッドでコピー元オブジェクトのデータ型を指定する。コピー元はすでにインスタンスが存在するため特に型を特定する必要はなく、object 型でよい。
- IDataCopyContext オブジェクトの「Destination.EntryManager.RegisterRootEntry」メソッドで、コピー先オブジェクトのデータ型を指定する。指定した型に基づきインスタンス生成するため、コピーしたい型を明示する。
- IDataCopyContext オブジェクトの「Destination.RegisterInstantiationType」メソッドで、指定したプロパティパス（第1引数）に対して、インスタンス生成する型（第2引数）を登録する。

以下に、実装例を示す。

```
class Class1
{
        public void Copy()
        {
            // コピー元オブジェクト
            object source = new SomeClass()
            {
                Prop1 = new SomeClass2()
                {
                    Prop21 = "src21",
                }
            };
            // コピー先オブジェクト
            object destination = new SomeClass();

            // IDataCopyContextオブジェクトを生成する
            IDataCopyContext context = DataCopyManager.CreateCopyContext();

            //コピー元、コピー先の型を指定
            Type sourceType = typeof(object);
            Type destinationType = typeof(SomeClass);
            context.Source.EntryManager.RegisterRootEntry(sourceType, source);
            context.Destination.EntryManager.RegisterRootEntry(destinationType, destination);

            // 対象プロパティパスをインスタンス化する型を登録する
            context.Destination.RegisterInstantiationType("Prop1", typeof(SomeClass2));
            // コピーする
            context.Copy();
        }
    }

    public class SomeClass
    {
        // プロパティがインタフェースで定義されているため、
        // コピー先のインスタンスがないと本来コピーできない
        public ISomeClass2 Prop1 { get; set; }
    }

    public interface ISomeClass2
    {
        string Prop21 { get; set; }
    }

    public class SomeClass2 : ISomeClass2
    {
        public string Prop21 { get; set; }
    }
}
```

リスト 15 インスタンス化する型を登録して実行するデータコピーの例

図 10 データコピールール

◆ 構成クラス

以下に、本機能を構成するクラスを示す。

表 12 構成クラス一覧

項番	クラス名	説明
Terasoluna.DataCopy 名前空間		
1	DataCopyManager	本機能のエントリクラス。
2	IDataAccessor	各種データにアクセスする機能を持つインタフェース。
3	IDataAccessorFactory	IDataAccessor オブジェクトを生成する機能を提供するインタフェース。
4	IDataContext	コピー元、または、コピー先データに関する情報を保持するインタフェース。
5	DataContextType	IDataContext がコピー元データかコピー先データであるかを表す種別を定義するクラス。
6	IDataCopyContext	データコピー実行時に必要な情報を提供するインタフェース。
7	IDataCopyFlowController	データコピーの機能を提供するインタフェース。
8	IDataCopyStrategy	コピー方法に関する定義するインタフェース。
9	DataEntry	データアクセスオブジェクト（コピーで扱うデータのオブジェクトツリーの構成要素）についての情報を保持するクラス。
10	DataEntrySet	コピー元とコピー先の DataEntry オブジェクトを保持するクラス
11	IDataEntryManager	DataEntry を管理するインタフェース。
12	IDataInstantiationStrategy	データコピー時のインスタンス生成に関する機能を定義するインタフェース。
13	IDataTypeConversionStrategy	値の型変換を行う機能を提供するインタフェース。
14	IDataTypeManager	データ型に関する定義をするインタフェース。
15	IMappingInfo	マッピング情報にアクセスするためのインタフェース。
16	DataAccessDataType	コピー処理対象のデータ型の種別を表します
17	DataAccessorBase	IDataAccessor の基底クラス。
18	ArrayDataAccessor	配列に関する IDataAccessor 実装クラス。
19	BindingListDataAccessor	System.ComponentModel.IBindingList に関する IDataAccessor 実装クラス。
20	DataRowDataAccessor	System.Data.DataRow に関する IDataAccessor 実装クラス。
21	DataTableDataAccessor	System.Data.DataTable に関する IDataAccessor 実装クラス。
22	DataSetDataAccessor	System.Data.DataSet に関する IDataAccessor 実装クラス。

23	FixedDataRowDataAccessor	カラムの変更を許容しない System.Data.DataRow に関する IDataAccessor 実装クラス。
24	FixedDataSetDataAccessor	テーブルの変更を許容しない System.Data.DataSet に関する IDataAccessor 実装クラス。
25	ListDataAccessor	System.Collections.Generic.List<T> に関する IDataAccessor 実装クラス。
26	NoTargetDataAccessor	対象データがない場合に関する IDataAccessor 実装クラス。
27	NullableElementListDataAccessor	System.Nullable<T> を保持する System.Collections.Generic.List<T> に関する IDataAccessor 実装クラス。
28	ObjectDataAccessor	任意のオブジェクトデータにアクセスする IDataAccessor 実装クラス。
29	RootDataAccessor.	ルート値にアクセスする IDataAccessor 実装クラス。
30	WcfDataObjectDataAccessor	WCF の通信データクラスにアクセス IDataAccessor 実装クラス。
31	DataAccessKeyType	データコピーのキーの種類を表す Enum。
32	PrimaryKeyValue	プライマリキーとその値を保持するクラス。
33	PrimaryKeyValues	リスト要素のプライマリキー情報の値を保持するクラス。
34	DataAccessorFactory	IDataAccessorFactory のデフォルト実装クラス。
35	DataContext	IDataContext のデフォルト実装クラス。
36	DataCopyContext	IDataCopyContext のデフォルト実装クラス。
37	DataCopyFlowController	IDataCopyFlowController のデフォルト実装クラス。
38	DataCopyStrategyBase	IDataCopyStrategy の基底クラス
39	SimpleDataCopyStrategy	単純なデータコピーを提供する IDataCopyStrategy 実装クラス
40	MappingDataCopyStrategy	IDataCopyStrategy のデフォルト実装クラス。マッピング定義があるデータコピーにも対応する。
41	DataInstantiationStrategy	IDataInstantiationStrategy のデフォルト実装クラス。
42	DataTypeConversionStrategy	IDataTypeConversionStrategy のデフォルト実装クラス。
43	DataEntryManager	IDataEntryManager のデフォルト実装クラス
44	DataTypeManager	IDataTypeManager のデフォルト実装クラス。
45	MappingInfo	コードによりデータコピー時に使用する IMappingInfo 実装クラス。

46	DesignTimeMappingInfo	Visual Studio のデザイナでマッピング情報を設定する場合に利用する IMappingInfo 実装クラス。
47	DesignTimeMappingInfoTypeConverter	Visual Studio のデザイナで DesignTimeMappingInfo をシリアライズ可能にする TypeConveter クラス。
48	MappingItem	コピー元とコピー先の対応関係を表すマッピング情報のを保持するクラス。
49	MappingItemTypeConverter	Visual Studio のデザイナで MappingItem をシリアライズ可能にする TypeConveter クラス。
50	DataCopyMappingInfo	データコピーの際のマッピング情報を保持するクラス。
51	SimpleDataCopyMappingInfo	単純なマッピング情報を保持するクラス。
52	DataCopyPropertyPathInfo	マッピングに利用するプロパティパスを解析するクラス。
53	DataEntryUtility	DataEntry オブジェクトに関するユーティリティクラス。
54	DataCopyErrorInfo	データコピーで発生したエラー情報を保持するクラス。
55	DataCopyMessageBuilder	データコピー発生したエラーなどの情報を文字列として組み立てるクラス。
56	DataCopyExtension	本機能の標準機能を提供する UnityContainerExtension 継承クラス。
Terasoluna.Windows.ViewModel.Validation.DataCopy 名前空間		
57	ViewDataAccessor	「画面データ」に関する IDataAccessor 実装クラス。
58	ValidatableDataAccessorFactory	DataAccessorFactory クラスの機能に加えて、ViewDataAccessor を考慮した IDataAccessorFactory 実装クラス。
59	ValidatableDataInstantiationStrategy	DataInstantiationStrategy クラスの機能に加えて ViewDataAccessor を考慮した IDataInstantiationStrategy 実装クラス。
60	ValidatableDataCopyExtension	DataCopyExtension クラスの機能に加えて ViewDataAccessor を考慮した UnityContainerExtension 継承クラス。

■ 拡張ポイント

本機能は、データコピー処理を構成するコンポーネントについてそれぞれインタフェースが提供されている。独自のデータコピールールを追加したい場合などは、開発プロジェクトの特色に合わせて各インタフェースを実装した拡張クラスを実装し、標準のデータコピー機能を容易に差し替えることができる。

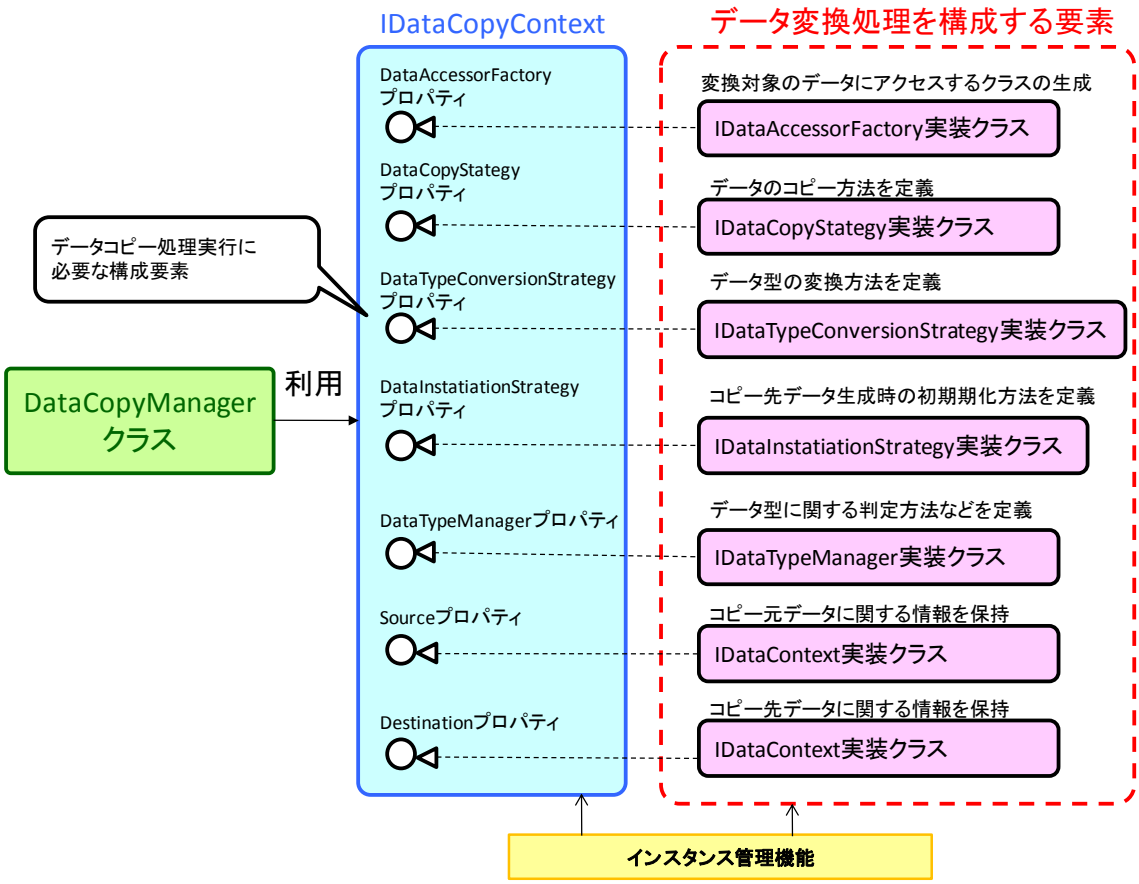


図 11 データコピー機能の内部構造

例えば、新たに型変換ルールを追加したい場合、標準の型変換機能を実装するクラスである、DataTypeConversionStrategy クラスを継承し、指定した型に対して優先的に利用する TypeConverter を登録することができる。

```
public class SampleDataTypeConversionStrategy : DataTypeConversionStrategy
{
    public SampleDataTypeConversionStrategy()
        : base()
    {
        ///指定した型に対して優先的に利用するTypeConveterを追加登録
        RegisterTypeConverter(typeof(SampleClass), new SampleTypeConverter());
    }
}

public class SampleTypeConverter : TypeConverter
{
    ///TypeConveterのメソッドをオーバライドして型変換処理を実装
}
```

リスト 16 DataTypeConversionStragegy の拡張

デフォルトのデータコピー機能の差し替えは DataCopyExtension クラスを参考に、UnityContainerExtension の継承クラスを作成して、UnityContainer.RegisterType メソッドで、機能拡張した実装クラスを DI する。
以下に、Extension クラスの実装例を示す。
この例では、デフォルトの型変換処理を実施する DataTypeConversionStrategy クラスの代わりに、先ほど作成した SampleDataTypeConversionStrategy クラスを DI している。

```
public class SampleDataCopyExtension : UnityContainerExtension
{
    protected override void Initialize()
    {
        Container.RegisterType<IDataAccessorFactory, DataAccessorFactory>
                                        (new ContainerControlledLifetimeManager());
        Container.RegisterType<IDataCopyStrategy, MappingDataCopyStrategy>
                                        (new ContainerControlledLifetimeManager());
        Container.RegisterType<IDataInstantiationStrategy, DataInstantiationStrategy>
                                        (new ContainerControlledLifetimeManager());
        Container.RegisterType<IDataTypeManager, DataTypeManager>
                                        (new ContainerControlledLifetimeManager());
        ///デフォルトの型変換機能を機能拡張したクラスと切り替え
        Container.RegisterType<IDataTypeConversionStrategy, SampleDataTypeConversionStrategy>
                                        (new ContainerControlledLifetimeManager());

        Container.RegisterType<IDataContext, DataContext>();
        Container.RegisterType<IDataCopyContext, DataCopyContext>();
    }
}
```

リスト 17 UnityContainerExtension 継承クラスの作成

FW 構成ファイル(TerasolunaFramework.config)に DataCopyExtension クラスの代わりに、作成した Extension クラスを設定する。
以下に、TerasolunaFramework.config の設定例を示す。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
・・・・
  <unity>
    ・・・
    <containers>
      <container>
        <types>
          ・・・
        </types>
        <extensions>
          <!-- 拡張したデータコピー機能の設定　-->
          <add type="Terasoluna.Laboratories.DataCopy.SampleDataCopyExtension ,
                                Terasoluna.Laboratories" />
          ・・・
        </extensions>
      </container>
    </containers>
  </unity>
</configuration>
```

リスト 18 TerasolunaFramework.config の設定例

■ 関連機能

- CM-02 インスタンス管理機能
- CL-02 画面遷移機能
- CL-03 イベント処理実行機能