PEX：外部言語によるプロダクションの開発

Version 2023.1
2024-01-02

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

PEX : 外部言語によるプロダクションの開発
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

1

PEX の概要

Production EXtension (PEX) フレームワークでは、Java や Python などの外部言語を選択し、その言語を使用して相互運用プロダクションを開発できます。相互運用プロダクションにより、メッセージ形式や通信プロトコルが異なる複数のシステムを統合できます。相互運用プロダクションに精通していない場合は、“プロダクションの概要” を参照してください。

PEX は、外部言語で実装されているビジネス・サービス、ビジネス・プロセス、およびビジネス・オペレーションの間の接続に柔軟性をもたらします。そのうえ、PEX 対応言語は、受信アダプタおよび送信アダプタの開発にも使用できます。PEX フレームワークでは、プロダクション全体を外部言語で作成することも、外部言語のコンポーネントと ObjectScript コンポーネントを混合して使用するプロダクションを作成することもできます。統合されると、外部言語で記述されたプロダクション・コンポーネントは実行時に呼び出され、PEX フレームワークを使用して、プロダクション内の別のコンポーネントにメッセージを送信します。

PEX フレームワークを使用する理由は以下のとおりです。

- 外部言語で記述された利用可能なライブラリを使用してプロトコルの新規アダプタを作成する。
- 利用可能な外部言語ライブラリを使用して複雑な分析や計算を実行する。
- アプリケーションのテクノロジ・スタックで ObjectScript を使用することなく、永続的なメッセージングおよび長期実行型のビジネス・プロセスを実装する。

PEX 対応言語を使用して、以下のプロダクション・コンポーネントを作成できます。

- 受信アダプタ
- 送信アダプタ
- ビジネス・サービス
- ビジネス・プロセス
- ビジネス・オペレーション

プロダクション・コンポーネントのコードの開発に加えて、通常はプロダクションの定義、構成、および監視も管理ポータルを使用して行います。例えば、管理ポータルを使用して、プロダクションの作成、ビジネス・サービス、ビジネス・プロセス、およびビジネス・オペレーションの構成、プロダクションの開始と停止、プロダクションで実行される永続メッセージのトレースを行います。

注釈　PEX は、Java ビジネス・ホスト機能に置き換わるものです。PEX を使用するために既存の Java ビジネス・ホスト・プロダクションを移行する処理の詳細は、Community の記事 ″Migrate from Java Business Host to PEX″ を参照してください。Java ビジネス・ホストはリリース 2019.3 で非推奨となり、この製品には含まれていません。

2
はじめに

PEX フレームワークを使用して外部言語コンポーネントを相互運用プロダクションに組み込む処理は、以下の手順で構成されます。

1. 好みの IDE で外部言語を使用してビジネス・ホストまたはアダプタを記述し、そのコードをコンパイルします。
2. 管理ポータルで新しい PEX コンポーネントを登録します。その PEX コンポーネントに対して、ObjectScript プロキシ・クラスが自動的に作成されます。
3. PEX コンポーネントがビジネス・サービス、ビジネス・オペレーション、またはビジネス・プロセスの場合は、標準のウィザードを使用してビジネス・ホストをプロダクションに追加し、ObjectScript プロキシ・クラスをビジネス・ホストのクラスとして指定します。PEX コンポーネントがアダプタの場合は、そのアダプタのプロキシ・クラスを参照するようにビジネス・サービスまたはビジネス・オペレーションを変更します。

2.1 PEX ライブラリ

各外部言語には、プロダクション・コンポーネントのタイプごとにスーパークラスを格納する PEX ライブラリが含まれています。使用可能な PEX ライブラリは以下のとおりです。

外部言語	PEX ライブラリ
Java	`com.intersystems.enslib.pex`
.NET	`InterSystems.EnsLib.PEX`
Python	`iris.pex`

2.2 PEX コンポーネントを使用した作業

PEX コンポーネントは、外部言語で記述されたリモート・クラスと、そのリモート・クラスを操作するためにネイティブ・プロダクションで使用される ObjectScript プロキシ・クラスで構成されます。カスタムの PEX アダプタまたはビジネス・ホストをプロダクションに追加するには、PEX コンポーネントとして登録しておく必要があります。登録すると、プロダクションを構築するユーザが、そのコンポーネントの詳細を利用できるようになります。PEX コンポーネントを表示および登録するには、管理ポータルを使用して、**[相互運用性]** > **[構成]** > [Production EXtensions Components] に移動します。

2.3 環境上の考慮事項

InterSystems IRIS Interoperability は、特定の Web アプリケーションが属している相互運用対応ネームスペース内でしか使用できません。クラスの作成では、予約パッケージ名を使用しないようにする必要があります。以降の項で詳しく説明します。

2.3.1 プロダクション対応ネームスペース

相互運用対応ネームスペースは、プロダクションをサポートするクラス、データ、およびメニューをネームスペースで使用可能にするグローバル・マッピング、ルーチン・マッピング、およびパッケージ・マッピングで構成されたネームスペースです。マッピングの詳細は、“ネームスペースの構成”を参照してください (このセクションの情報を使用して、相互運用対応ネームスペースで行われる実際のマッピングを確認できます。詳細はリリースにより異なる場合がありますが、ユーザ側で特に作業する必要はありません)。

InterSystems IRIS のインストール時に作成されるシステム提供のネームスペースは、Community Edition で USER ネームスペースが相互運用対応ネームスペースである場合を除き、相互運用対応ではありません。ユーザが作成する新しいネームスペースはすべて、デフォルトで相互運用対応になります。ネームスペースを作成する際に **[ネームスペースを相互運用プロダクション対応にする]** チェック・ボックスのチェックを外すと、InterSystems IRIS は相互運用に対応しないネームスペースを作成します。

重要　システム提供のネームスペースはすべて、再インストールまたはアップグレードの際に上書きされます。このため、常に、ユーザが作成した新規ネームスペースで作業することをお勧めします。新規ネームスペースの作成の詳細は、“データの構成”を参照してください。

2.3.2 Web アプリケーションの要件

また、プロダクションは、/csp/namespace (この場合の namespace はネームスペースの名前) と名付けられた Web アプリケーションが関連付けられている場合のみ、そのネームスペースで使用できます。(これは、ネームスペースに対するデフォルトの Web アプリケーション名です。)Web アプリケーションの定義の詳細は、"アプリケーション" を参照してください。

2.3.3 予約パッケージ名

相互運用対応ネームスペースでは、Ens、EnsLib、EnsPortal、または CSPX をパッケージ名として使用しないでください。これらのパッケージは、アップグレード・プロセス中に完全に置換されます。これらのパッケージ内でクラスを定義した場合は、アップグレード前にそれらのクラスをエクスポートして、アップグレード後にインポートする必要があります。

また、先頭に Ens (大文字と小文字の区別あり) が付くパッケージ名を使用しないことをお勧めします。これには次の 2 つの理由があります。

- 先頭に Ens が付く名前のパッケージにクラスをコンパイルすると、コンパイラは、生成されたルーチンを ENSLIB システム・データベースに書き込みます (コンパイラがそうするのは、名前の先頭に Ens が付くルーチンはすべて、そのデータベースにマッピングされるからです)。つまり、インスタンスをアップグレードすると、ENSLIB データベースが置換され、生成されたルーチンはアップグレードによって削除され、クラス定義のみが残ります。この時点で、クラスを使用するために、クラスのリコンパイルが必要になります。

  これに対し、名前の先頭が Ens ではないパッケージ内のクラスは、インスタンスのアップグレード時にリコンパイルする必要はありません。

- 名前の先頭に Ens が付くパッケージ内のクラスを定義すると、それらのクラスは相互運用対応のすべてのネームスペースで使用可能になりますが、このことは望ましい場合も、望ましくない場合もあります。1 つ言えることとして、パッ

ケージの名前の先頭に Ens が付いている場合、名前が同じでコンテンツが異なる 2 つのクラスを、異なる相互運用対応ネームスペースで使用することはできなくなります。

3

ビジネス・ホストおよびアダプタの概要

ここでは、外部言語で記述されたすべてのビジネス・ホストとアダプタに適用される情報を提供します。

3.1 実行時変数の作成

PEX フレームワークでは、管理ポータルを使用してリモート・クラスの変数の実行時の値を指定できるので、1 つの PEX コンポーネントをさまざまなプロダクションで再利用できます。リモート・クラスで宣言された変数は、管理ポータルには対応するビジネス・ホストの設定として表示されます。例えば、ビジネス・サービスの Python クラスで `Min = int(0)` を宣言した場合、プロダクションのビジネス・サービスには [**最小**] の設定があります。実行時に、この変数は管理ポータルでの設定の値に設定されます。

管理ポータルに表示したくない変数をリモート・クラスで宣言する場合は、メタデータを追加してその変数を非表示にできます。

3.1.1 変数のメタデータ

変数に対して作成された管理ポータルの設定を制御するメタデータをリモート・クラスに追加できます。このメタデータの構文は、外部言語によって異なります。Java ではアノテーション、.NET では属性、Python では *`variableName`*`_info` という名前の特殊なメソッドをそれぞれ使用します。例えば、以下のコードは、設定 `MyVariable` が必須であることを指定し、その設定のヒントとして表示される説明を追加します。

Java

```
@FieldMetadata(IsRequired=true,Description="Name of the company")
public String MyVariable;
```

.NET

```
[FieldMetadata(IsRequired=true,Description="Name of the company")]
public string MyVariable;
```

Python

```
MyVariable = int(0)

@classmethod
def MyVariable_info(self) -> \
 {
    'Description': "Maximum value",
    'IsRequired': True
  }:
  pass
```

この構文を使用して、管理ポータルの設定の動作を制御する以下の要素をメタデータに追加できます。

メタデータの要素	説明
Description	管理ポータルにヒントとして表示される、設定の説明。
Category	カテゴリ見出しで設定をグループ化します。例えば、Category="Basic" を Java クラスに追加すると、その設定は、設定の [基本] セクションにグループ化されます。
DataType	変数のデータ型を上書きし、設定を新しいデータ型に関連付けます (例 : Ens.DataType.ConfigName)。
IsRequired	true の場合、設定の値を指定する必要があります。既定値は false です。
ExcludeFromSettings	true の場合、変数は管理ポータルに設定として表示されなくなります。既定値は false です。

3.2 PEX コンポーネントのメタデータ

リモート・クラスの個々の変数にメタデータを追加できるのと同様に、PEX コンポーネント全体に適用されるメタデータを追加できます。このメタデータを使用して、管理ポータルでコンポーネントに関する情報を提供します。この情報は、管理ポータルの [情報を提供する設定] の [Production EXtensions] ページおよび [プロダクション構成] ページに表示されます。

メタデータの要素	説明
Description	PEX コンポーネントの説明。
Info URL	PEX コンポーネントに関連付けられている URL。例えば、コンポーネントの詳細を提供する Web ページ。

このクラス・メタデータを組み込むための構文は、外部言語によって異なります。Java ではアノテーション、.NET では属性を使用します。Python では、docstring と特殊変数を使用します。以下のコードは、これらのメタデータ要素を PEX コンポーネントのリモート・クラスに追加します。

Java

```
@ClassMetadata(Description="Custom Java Business Service",InfoURL="http://www.mycompany.com")
public class MyBusinessService extends com.intersystems.enslib.pex.BusinessService {
```

.NET

```
[ClassMetadata(Description ="Custom .NET Business Service", InfoURL="www.mycompany.com")]
public class MyBusinessService : InterSystems.EnsLib.PEX.BusinessService
```

Python

```
class MyBusinessService(iris.pex.BusinessService):
  """ Custom Business Service in Python """

  INFO_URL = "http://www.mycompany.com"
```

4

PEX メッセージング

PEX フレームワーク内では、ビジネス・ホスト間で送信されるメッセージのほとんどは、以下のいずれかのサブクラスからインスタンス化されたオブジェクトです。

言語	メッセージ・クラス
Java	`com.intersystems.enslib.pex.Message`
.NET	`InterSystems.EnsLib.PEX.Message`
Python	`iris.pex.Message`

必要な操作は、サブクラスにプロパティを追加し、そのサブクラスのインスタンス化されたオブジェクトを、`SendRequestAsync()` や `SendRequestSync()` などのメソッドを使用して渡すだけです。InterSystems IRIS 内では、PEX 対応言語で記述されたメッセージ・オブジェクトは、メッセージを永続的および動的にする、`EnsLib.PEX.Message` クラスのオブジェクトに対応します。`EnsLib.PEX.Message` タイプのオブジェクトを操作することにより、外部 PEX オブジェクト内の任意のプロパティを参照できます。内部的には、PEX オブジェクトはビジネス・ホスト間で渡される際に JSON として表されるため、管理ポータルでメッセージを表示すると JSON 形式で表示されます。

他のメッセージ・オブジェクトを使用することもできますが、これらがビジネス・ホスト間で渡される場合は、やはり永続的である必要があります。組み込みの ObjectScript コンポーネントにオブジェクトを渡すには、IRISObject タイプを使用して、これをその ObjectScript コンポーネントで必要な永続オブジェクトにマッピングします。例えば、EnsLib.PubSub.PubSub-Operation にメッセージを送信する場合は、IRISObject を EnsLib.PubSub.Request にマッピングします。ビジネス・ホスト間で非永続オブジェクトをメッセージとして渡そうとすると、実行時エラーが発生します。

受信アダプタからビジネス・サービスに送信されるオブジェクトは任意で、永続的である必要はありません。

注釈　　送信するメッセージは InterSystems IRIS の最大文字列長に制限されます。

5

PEX 受信アダプタ

ビジネス・サービスは受信アダプタを使用して特定のタイプの入力データを受信します。ObjectScript ビジネス・サービスで使用されるカスタムの受信アダプタを記述できます。PEX 対応言語で記述されたビジネス・サービスで、これを使用することもできます。外部言語で記述されたすべてのプロダクション・コンポーネントに関する一般情報は、"ビジネス・ホストおよびアダプタの概要" を参照してください。

5.1 カスタム・アダプタの開発

外部言語でカスタム受信アダプタを記述する処理を開始するには、以下のいずれかのクラスを拡張します。

言語	クラス
Java	`com.intersystems.enslib.pex.InboundAdapter`
.NET	`InterSystems.EnsLib.PEX.InboundAdapter`
Python	`iris.pex.InboundAdapter`

5.1.1 抽象メソッドの実装

通常、受信アダプタの `OnTask()` メソッドが、そのアダプタの主要機能を実行します。実行時、`OnTask()` メソッドは、アダプタを使用しているビジネス・ホストの設定で指定された間隔で呼び出されます。`OnTask()` から `BusinessHost.ProcessInput()` を呼び出し、関連付けられたビジネス・サービスの `ProcessInput` メソッドにオブジェクトをディスパッチします。例えば、簡単なアダプタには、以下のようなコードを含めます。

Java

```
public void OnTask() throws Exception {
  SimpleObject request = new SimpleObject("message #"+(++runningCount));
  // send object to business service's ProcessInput() method
  String response = (String) BusinessHost.ProcessInput(request);
  return;
}
```

.NET

```
public override void OnTask()
{
  SimpleObject request = new SimpleObject("message #" + (++runningCount));
  // send object to business service's ProcessInput() method
  string response = (string)BusinessHost.ProcessInput(request);
  return;
}
```

Python

```
def OnTask(self):
  msg = "this is message # %d" %self.runningCount
  request = demo.SimpleObject(msg)
  # send object to business service's ProcessInput() method
  response = self.BusinessHost.ProcessInput(request)
  return
```

アダプタの `BusinessHost.ProcessInput` 呼び出しからビジネス・サービスの `ProcessInput` メソッドに送信されるオブジェクトは任意であり、InterSystems IRIS 内で永続的である必要はありません。このことは、ビジネス・サービスの `ProcessInput` メソッドからアダプタに返されるオブジェクトにも当てはまります。

既定では、アダプタからビジネス・サービスに送信されるオブジェクトは、JSON にシリアル化され、ビジネス・サービスはそれを IRISObject タイプとして受け取ります。ただし、アダプタとビジネス・サービスが接続を共有している場合は、ビジネス・サービスは同じオブジェクト・タイプを受信して返すことができます。これは有利な点です。詳細は、"接続の共有" を参照してください。

少なくとも、アダプタにはスーパークラスの `OnInit` メソッド、`OnTearDown` メソッド、および `OnTask` メソッドを実装する必要があります。受信アダプタのこれらのメソッドと他のメソッドの詳細は、"PEX API リファレンス" を参照してください。

5.2 アダプタの登録

PEX アダプタのコードの記述が完了したら、次はその登録です。アダプタを登録することにより、ObjectScript プロキシ・クラスが生成され、ビジネス・サービスはこのクラスを使用してアダプタを識別できるようになります。また、プロダクションがアダプタへの接続に使用する外部言語サーバが定義されます。アダプタの登録の詳細は、"PEX コンポーネントの登録" を参照してください。

5.3 ビジネス・サービスへのアダプタの追加

PEX アダプタは、PEX ビジネス・サービスまたはネイティブ ObjectScript ビジネス・サービスで使用できます。PEX アダプタを使用するようビジネス・サービスを構成する手順は、ビジネス・サービスのタイプによって異なります。どちらのシナリオでも、アダプタを PEX コンポーネントとして登録する必要があります。

PEX ビジネス・サービスで PEX アダプタを使用する場合、そのビジネス・サービスのリモート・クラスはメソッドを使用してそのアダプタを識別します。詳細は、"受信アダプタの使用法" を参照してください。

すべてのネイティブ ObjectScript ビジネス・サービスと同様に、PEX アダプタを使用するネイティブ・ビジネス・サービスも ADAPTER パラメータを使用してアダプタを識別します。この場合、ADAPTER パラメータは、PEX アダプタの ObjectScript プロキシ・クラスの名前に設定されます。既定では、このプロキシ・クラスの名前はアダプタのリモート・クラスの名前と同じですが、アダプタの登録時にカスタムのプロキシ名が定義されている場合があります。

6

PEX 送信アダプタ

ビジネス・オペレーションは、送信アダプタを使用して、プロダクションから特定のタイプのデータを送信します。ObjectScript ビジネス・オペレーションで使用されるカスタムの送信アダプタを記述できます。PEX 対応言語で記述されたビジネス・オペレーションで、これを使用することもできます。外部言語で記述されたすべてのプロダクション・コンポーネントに関する一般情報は、"ビジネス・ホストおよびアダプタの概要" を参照してください。

6.1 カスタム・アダプタの開発

外部言語で送信アダプタを記述するには、以下のいずれかのクラスを拡張します。

言語	クラス
Java	`com.intersystems.enslib.pex.OutboundAdapter`
.NET	`InterSystems.EnsLib.PEX.OutboundAdapter`
Python	`iris.pex.OutboundAdapter`

アウトバウンド・アダプタ内で、プロダクションからデータを正常に送信するために必要なすべてのメソッドを作成できます。これらの各メソッドは、このアダプタに関連付けられたビジネス・オペレーションから呼び出すことができます。ビジネス・オペレーションは、任意のオブジェクトやリテラルの引数を指定してメソッドを呼び出すことができます。ビジネス・オペレーションがアダプタ・メソッドを呼び出すしくみの詳細は、"アダプタ・メソッドの呼び出し" を参照してください。

6.1.1 抽象メソッドの実装

プロダクションからデータを送信するメソッドを作成することに加えて、リモート送信アダプタ・クラスには、いくつかの抽象メソッドを実装する必要があります。これらのメソッドの詳細は、"PEX API リファレンス" を参照してください。

6.2 アダプタの登録

PEX アダプタのコードの記述が完了したら、次はその登録です。アダプタを登録することにより、ObjectScript プロキシ・クラスが生成され、ビジネス・オペレーションはこのクラスを使用してアダプタを識別できるようになります。また、プロダクションがアダプタへの接続に使用する外部言語サーバが定義されます。アダプタの登録の詳細は、"PEX コンポーネントの登録" を参照してください。

6.3 ビジネス・オペレーションへのアダプタの追加

PEX アダプタは、PEX ビジネス・オペレーションまたはネイティブ ObjectScript ビジネス・オペレーションによって使用できます。PEX アダプタを使用するようビジネス・オペレーションを構成する手順は、ビジネス・オペレーションのタイプによって異なります。どちらのシナリオでも、アダプタを PEX コンポーネントとして登録する必要があります。

PEX ビジネス・オペレーションで PEX アダプタを使用する場合、そのビジネス・オペレーションのリモート・クラスはメソッドを使用してそのアダプタを識別します。詳細は、"送信アダプタの使用法" を参照してください。

すべてのネイティブ ObjectScript ビジネス・オペレーションと同様に、PEX アダプタを使用するネイティブ・ビジネス・オペレーションも ADAPTER パラメータを使用してアダプタを識別します。この場合、ADAPTER パラメータは、PEX アダプタの ObjectScript プロキシ・クラスの名前に設定されます。既定では、このプロキシ・クラスの名前はアダプタのリモート・クラスの名前と同じですが、アダプタの登録時にカスタムのプロキシ名が定義されている場合があります。

7

PEX ビジネス・サービス

ビジネス・サービスは外部システムと接続し、そこから受信アダプタを経由してメッセージを受信します。外部言語で記述されたすべてのプロダクション・コンポーネントに関する一般情報は、"ビジネス・ホストおよびアダプタの概要" を参照してください。

7.1 ビジネス・サービスの開発

外部言語でビジネス・サービスを記述するには、以下のいずれかのクラスを拡張します。

言語	クラス
Java	`com.intersystems.enslib.pex.BusinessService`
.NET	`InterSystems.EnsLib.PEX.BusinessService`
Python	`iris.pex.BusinessService`

ビジネス・サービスを実装する方法は、次の 3 通りあります。

1. アダプタを使用してビジネス・サービスをポーリングする ー プロダクション・フレームワークがアダプタの OnTask() メソッドを定期的に呼び出します。このメソッドは、受信データをビジネス・サービスの ProcessInput() メソッドに送信し、それを受けて ProcessInput () メソッドはビジネス・サービス・コードが設定された OnProcessInput メソッドを呼び出します。アダプタからのデータを処理するには、カスタム・コードに ProcessInput ではなく OnProcessInput メソッドを実装する必要があります。
2. 既定のアダプタを使用するビジネス・サービスをポーリングするー この場合、プロダクション・フレームワークは、データを指定せずに既定のアダプタの OnTask メソッドを呼び出します。次に、OnProcessInput() メソッドがアダプタの役割を実行し、外部システムにアクセスしてデータを受信します。
3. ビジネス・サービスをポーリングしないープロダクション・フレームワークは、ビジネス・サービスを開始しません。代わりに、長時間実行されるプロセスか、定期的に開始されるプロセスのカスタム・コードが、Director.CreateBusinessService() メソッドを呼び出すことで、ビジネス・サービスを開始します。詳細は、"Director" を参照してください。

7.1.1 抽象メソッドの実装

PEX クラスを拡張した後に、ビジネス・サービス用に抽象メソッドを実装する必要があります。

アダプタを使用したポーリング・ビジネス・サービスを開発する場合、`OnProcessInput()` はアダプタから任意のオブジェクトを取り、任意のオブジェクトを返します。これらの任意のオブジェクトは、永続的である必要はありません。

実装する必要のある抽象メソッドの詳細は、"PEX API リファレンス" を参照してください。

7.2 受信アダプタの使用法

プロダクション内では、ビジネス・サービスは、受信アダプタを使用してプロダクションの外部のシステムと通信します。PEX ビジネス・サービスの開発時に、リモート・クラスに特殊なメソッドを組み込んで、ビジネス・サービスでどの受信アダプタを使用するかを定義できます。この受信アダプタとしては、PEX アダプタまたはネイティブ ObjectScript アダプタのいずれかを使用できます。

PEX ビジネス・サービスの受信アダプタを指定するために使用するメソッドは、`getAdapterType()` です。例えば、PEX ビジネス・サービスでカスタム PEX 受信アダプタを使用する場合、リモート・クラスに以下のようなコードを含めます。

Java

```
public String getAdapterType() {
  return "com.demo.pex.MyInboundAdapter";
}
```

.NET

```
public override string getAdapterType() {
  return "Demo.PEX.MyInboundAdapter";
}
```

Python

```
def getAdapterType():
  return "demo.PEX.MyInboundAdapter"
```

PEX アダプタを使用する場合、`getAdapterType` メソッドで、アダプタの登録時に指定された ObjectScript プロキシ・クラスの名前を返す必要があります。既定では、このプロキシ名はリモート・クラスの名前と同じですが、カスタム名が定義されている場合があります。

リモート・クラスに `getAdapterType` を含めない場合は、標準の `Ens.InboundAdapter` アダプタがビジネス・サービスで使用されます。ビジネス・サービスでアダプタを使用しない場合、空の文字列が返されます。

7.3 ビジネス・サービスの使用法

PEX ビジネス・サービスのリモート・クラスの開発が完了したら、以下の手順に従って、ビジネス・サービスを相互運用プロダクションに統合できます。

1. **[相互運用性]** > **[構成]** > [Production EXtensions Components] に移動して、PEX ビジネス・サービスを登録します。詳細は、"PEX コンポーネントの登録" を参照してください。
2. プロダクションを開き、標準のウィザードを使用して、ビジネス・サービスを追加します。**[サービスクラス]** フィールドで、PEX コンポーネントの ObjectScript プロキシ・クラスを選択します。既定では、このプロキシ・クラスの名前はリモート・クラスの名前と同じですが、コンポーネントの登録時にカスタム名が定義されている場合があります。

8

PEX ビジネス・プロセス

ビジネス・プロセスでは、ルーティングやメッセージ変換を含むビジネス・ロジックを定義できます。ビジネス・プロセスは、プロダクション内の他のビジネス・ホストからメッセージを受信して処理します。

外部言語で記述されたすべてのプロダクション・コンポーネントに関する一般情報は、"ビジネス・ホストおよびアダプタの概要" を参照してください。

8.1 ビジネス・プロセスの開発

外部言語でビジネス・プロセスを記述するには、以下のいずれかのクラスを拡張します。

言語	クラス
Java	`com.intersystems.enslib.pex.BusinessProcess`
.NET	`InterSystems.EnsLib.PEX.BusinessProcess`
Python	`iris.pex.BusinessProcess`

8.1.1 抽象メソッドの実装

ビジネス・プロセス・クラスを拡張した後に、抽象メソッドをいくつか実装する必要があります。これらのメソッドの詳細は、"PEX API リファレンス" を参照してください。

8.1.2 永続プロパティ

InterSystems IRIS において、ネイティブのビジネス・プロセスは永続オブジェクトです。ビジネス・プロセスの存続期間中、プロパティは保存され、各コールバックの間アクセス可能です。既定では、PEX ビジネス・プロセスにはこの特性がありません。各メソッドは個別のインスタンスで呼び出され、変数の値は保持されません。ただし、PEX ビジネス・プロセスのプロパティを永続的にすることで、ネイティブのビジネス・プロセスの永続性に似せることができます。PEX ビジネス・プロセスのプロパティを永続化するには、以下の構文を使用します。

Java

```
// Use annotation to create a persistent property
// Strings, primitive types, and their boxed types can be persisted

@Persistent
public integer runningTotal = 0;
```

.NET

```
// Use attribute to create a persistent property
// Strings, primitive types, and their boxed types can be persisted

[Persistent]
public int runningTotal = 0;
```

Python

```
#Set class variable to create persistent property
#Only variables of types str, int, float, bool and bytes can be persisted

PERSISTENT_PROPERTY_LIST=["myVariable1","myVariable2"]
```

InterSystems IRIS において、永続プロパティは、ビジネス・プロセスの対応するインスタンスに保存されます。永続プロパティは、各コールバックの前にリストアされ、各コールバックの後に保存されます。

8.2 ビジネス・プロセスの使用法

PEX ビジネス・プロセスのリモート・クラスの開発が完了したら、以下の手順に従って、ビジネス・プロセスを相互運用プロダクションに統合できます。

1. **[相互運用性] > [構成] > [Production EXtensions Components]** に移動して、PEX ビジネス・プロセスを登録します。詳細は、"PEX コンポーネントの登録" を参照してください。
2. プロダクションを開き、標準のウィザードを使用して、ビジネス・プロセスを追加します。**[ビジネス・プロセス・クラス]** フィールドで、PEX コンポーネントの ObjectScript プロキシ・クラスを選択します。既定では、このプロキシ・クラスの名前はリモート・クラスの名前と同じですが、コンポーネントの登録時にカスタム名が定義されている場合があります。

9

PEX ビジネス・オペレーション

ビジネス・オペレーションは、外部システムと接続し、送信アダプタを経由してこれらにメッセージを送信します、

PEX 対応言語で記述されたすべてのプロダクション・コンポーネントに関する一般情報は、"ビジネス・ホストおよびアダプタの概要" を参照してください。

9.1 ビジネス・オペレーションの開発

外部言語でビジネス・オペレーションを記述するには、以下のいずれかのクラスを拡張します。

言語	クラス
Java	`com.intersystems.enslib.BusinessOperation`
.NET	`InterSystems.EnsLib.PEX.BusinessOperation`
Python	`iris.pex.BusinessOperation`

9.1.1 抽象メソッドの実装

実行時、ビジネス・オペレーションが別のビジネス・ホストからのメッセージを受信すると、`OnMessage()` メソッドが呼び出されます。ビジネス・オペレーションは、このメソッドから、このビジネス・オペレーションに関連付けられた送信アダプタで定義された任意のメソッドを呼び出すことができます。ビジネス・オペレーションから送信アダプタへの呼び出しのパラメータは、永続的である必要はありません。

実装する必要のある他の抽象メソッドの詳細は、"PEX API リファレンス" を参照してください。

9.2 送信アダプタの使用法

プロダクション内では、ビジネス・オペレーションは、送信アダプタを使用してプロダクションの外部のシステムと通信します。PEX ビジネス・オペレーションの開発時に、リモート・クラスに特殊なメソッドを組み込んで、ビジネス・オペレーションでどの送信アダプタを使用するかを定義できます。この送信アダプタとしては、PEX アダプタまたはネイティブ ObjectScript アダプタのいずれかを使用できます。

PEX ビジネス・オペレーションの送信アダプタを指定するために使用するメソッドは、`getAdapterType()` です。例えば、PEX ビジネス・オペレーションでカスタム PEX 送信アダプタを使用する場合、リモート・クラスに以下のようなコードを含めます。

Java

```
public String getAdapterType() {
  return "com.demo.pex.MyOutboundAdapter";
}
```

.NET

```
public override string getAdapterType() {
  return "Demo.PEX.MyOutboundAdapter";
}
```

Python

```
def getAdapterType():
  return "demo.PEX.MyOutboundAdapter"
```

PEX アダプタを使用する場合、`getAdapterType` メソッドで、アダプタの登録時に指定された ObjectScript プロキシ・クラスの名前を返す必要があります。既定では、このプロキシ名はリモート・クラスの名前と同じですが、カスタム名が定義されている場合があります。

9.2.1 アダプタ・メソッドの呼び出し

ビジネス・オペレーションは、そのアダプタのコードに定義されているメソッドを呼び出して、送信アダプタを使用します。このメソッドを呼び出すための構文は、ビジネス・オペレーションが PEX コンポーネントであるか、それともネイティブ ObjectScript クラスであるかによって異なります。ネイティブのビジネス・オペレーションから PEX アダプタ・メソッドを呼び出す方法の詳細は、"ビジネス・ホストからのプロパティとメソッドへのアクセス" を参照してください。

ビジネス・オペレーションが PEX コンポーネントの場合は、`Adapter.invoke()` を使用してアダプタのメソッドを呼び出します。そのシグニチャは、以下のとおりです。

`Adapter.invoke("`*`methodName`*`", `*`arguments`*`)`

説明：

- methodName は、送信アダプタに定義されている実行対象のメソッドの名前を指定します。
- arguments には、指定したメソッドの引数が含まれます。

例えば、アダプタの `printString` メソッドを呼び出すには、ビジネス・オペレーションに以下のコードを追加します。

Java

```
public Object OnMessage(Object request) throws Exception {
  MyRequest myReq = (MyRequest)request;
  Adapter.invoke("printString", myReq.requestString);
}
```

.NET

```
public override object OnMessage(object request)
{
  MyRequest myReq = (MyRequest)request;
  Adapter.invoke("printString", myReq.RequestString);
}
```

Python

```
def OnMessage(self, messageInput):
  self.Adapter.invoke("printString", messageInput.requestString)
  return
```

ビジネス・オペレーションがプリミティブをアダプタに渡した場合、アダプタは同じプリミティブを受け取ります。ただし、既定では、ビジネス・オペレーションがオブジェクトをアダプタに渡した場合、そのオブジェクトは JSON にシリアル化され、アダプタはそれを IRISObject タイプとして受け取ります。この動作を変更して、アダプタが同じオブジェクト・タイプを受け取って返すようにするには、"接続の共有" を参照してください。

9.3 ビジネス・オペレーションの使用法

PEX ビジネス・オペレーションのリモート・クラスの開発が完了したら、以下の手順に従って、ビジネス・オペレーションを相互運用プロダクションに統合できます。

1. **[相互運用性]** > **[構成]** > [Production EXtensions Components] に移動して、PEX ビジネス・オペレーションを登録します。詳細は、"PEX コンポーネントの登録" を参照してください。
2. プロダクションを開き、標準のウィザードを使用して、ビジネス・オペレーションを追加します。**[オペレーションクラス]** フィールドで、PEX コンポーネントの ObjectScript プロキシ・クラスを選択します。既定では、このプロキシ・クラスの名前はリモート・クラスの名前と同じですが、コンポーネントの登録時にカスタム名が定義されている場合があります。

10

PEX コンポーネントの登録

外部言語を使用して PEX コンポーネントのリモート・クラスを作成したら、管理ポータルを使用してそのクラスを登録する必要があります。この登録により、PEX コンポーネントに関する重要な情報が定義されます。例えば、リモート・クラスを接続する外部言語サーバや、このコンポーネント用に作成される ObjectScript プロキシ・クラスの名前などです。登録プロセスは、PEX コンポーネントがアダプタであってもビジネス・ホストであっても同じです。

Tipヒント　リモート・クラスを接続する外部言語サーバは、PEX コンポーネントを登録する前に、完全に機能する状態にしておく必要があります。Java を使用する場合は、ユーティリティ jar ファイルをサーバのクラスパスに追加する必要があります。この jar ファイルの詳細は、"外部言語サーバとの接続" を参照してください。

PEX コンポーネントを登録する手順は、以下のとおりです。

1. 管理ポータルを開いて、**[相互運用性]** > **[構成]** > [Production EXtensions Components] に移動します。
2. [Register New Component] を選択します。
3. **[リモートクラス名]** フィールドを使用して、外部言語で記述されたクラスを、そのパッケージも含めて指定します。
4. 必要に応じて、[Proxy Name] フィールドを使用し、PEX コンポーネント用に作成される ObjectScript プロキシ・クラスのカスタム名を指定します。既定値は、リモート・クラスの名前です。
5. [External Language Server] フィールドを使用して、プロダクションがリモート・クラスへの接続に使用する外部言語サーバを選択します。外部言語サーバは、現在は動作していなくても、完全に機能する状態である必要があります。
6. [Gateway Extra CLASSPATHS] フィールドを使用して、リモート・クラスが含まれる実行可能ファイルを指定します。リモート・クラスに必要な他のバイナリ、ファイル、およびディレクトリを追加することもできます。

注釈　**[登録]** を選択した後、[更新] アイコンを選択しない限り、新しい PEX コンポーネントはリストに表示されません。

11

外部言語サーバとの接続

外部言語で記述された PEX コンポーネントを登録する際に、プロダクションでリモート・クラスとの通信に使用する外部言語サーバを指定します。この外部言語サーバは一般的にゲートウェイと呼ばれます。

ほとんどの言語では、PEX プロダクションの実行に必要な機能は PEX 言語用の組み込みの外部言語サーバにすべて含まれており、カスタム・サーバを作成する必要はありません。ただし、Java を使用する場合は、外部言語サーバのクラスパスに特殊なユーティリティ jar ファイルを追加する必要があります。したがって、既定またはカスタムの Java サーバのクラスパスに *install-dir*/dev/java/lib/1.8/intersystems-utils-*version*.jar を追加する必要があります。install-dir はインターシステムズ製品のインストール先のディレクトリです。このファイルの場所を手動で入力する場合は、キーワード $$IRISHOME を使用して、インストール・ディレクトリを特定します。

11.1 接続の共有

場合によっては、PEX アダプタで、別個の外部言語サーバ接続を使用するのではなく、関連する PEX ビジネス・サービスまたはビジネス・オペレーションの接続を使用できます。各 PEX コンポーネントで別個の接続を使用する場合、コンポーネント間で渡されるオブジェクトを JSON 文字列にシリアル化し、コンポーネント間でオブジェクトを直接渡さないようにする必要があります。1 つの接続を共有することで、アダプタとビジネス・ホストは、オブジェクトを直接渡し、同じオブジェクトを受け取ることができます。接続を共有するには、PEX アダプタおよび関連する PEX ビジネス・ホストを同じ外部言語で記述する必要があります。

PEX アダプタがそのビジネス・サービスまたはビジネス・オペレーションと接続を共有するように指定するには、管理ポータルを使用して、ビジネス・ホストの [Alternative Adapter Connection] > [Use Host Connection] 設定を選択します。この設定を有効にすると、PEX フレームワークは、PEX アダプタの登録時に指定した外部言語サーバ設定を無視するようになります。

以下の例は、アダプタとその関連するビジネス・ホストとの間で接続を共有することの利点を示しています。

ビジネス・サービスと受信アダプタ

受信アダプタがビジネス・サービスと接続を共有する場合、ビジネス・サービスは、送信アダプタが送信したオブジェクトと同じオブジェクトを受け取ります。以下に例を示します。

Java

```
// Code from the inbound adapter:
public void OnTask() throws Exception {
  SimpleObject request = new SimpleObject("message #1");
  String response = (String) BusinessHost.ProcessInput(request);
}

// Code from the business service:
public Object OnProcessInput(Object messageInput) throws Exception {
  SimpleObject obj = (SimpleObject)messageInput;
  System.out.print("\r\n[Java] Object received: " + obj.value);
  return "...Service received " + obj.value;
}
```

一方、以下のコードは、ビジネス・サービスと受信アダプタが接続を共有しない場合、受信アダプタからビジネス・サービスに送信されたオブジェクトをどのように処理する必要があるのかを示しています。

Java

```
// Code from the inbound adapter:
public void OnTask() throws Exception {
  SimpleObject request = new SimpleObject("message #1");
  String response = (String) BusinessHost.ProcessInput(request);
}

// Code from the business service:
public Object OnProcessInput(Object messageInput) throws Exception {
  com.intersystems.jdbc.IRISObject obj = (com.intersystems.jdbc.IRISObject)messageInput;
  System.out.print("\r\n[Java] Object received: " + obj.get("value"));
  return "...Service received" + obj.get("value");
}
```

ビジネス・オペレーションと送信アダプタ

送信アダプタがビジネス・オペレーションと接続を共有する場合、アダプタは、ビジネス・オペレーションが送信したオブジェクトと同じオブジェクトを受け取ります。以下に例を示します。

Java

```
// Code from the business operation:
public Object OnMessage(Object request) throws Exception {
  SimpleObject myObj = new SimpleObject("my string");
  Adapter.invoke("passObj", myObj);
  return null;
}

// Code from the associated adapter:
public Object passObj(SimpleObject obj) {
  System.out.print("\r\n[Java]Object received: " + obj.value);
  return null;
}
```

一方、以下のコードは、ビジネス・オペレーションと送信アダプタが接続を共有しない場合、ビジネス・オペレーションからアダプタに送信されたオブジェクトをどのように処理する必要があるのかを示しています。

Java

```
// Code from the business operation:
public Object OnMessage(Object request) throws Exception {
  SimpleObject myObj = new SimpleObject("my string");
  Adapter.invoke("passObj", myObj);
  return null;
}

// Code from the associated adapter:
public Object passObj(com.intersystems.jdbc.IRISObject obj) {
  System.out.print("\r\n[Java]Object received: " + obj.get(“value”));
  return null;
}
```

A

PEX API リファレンス

このリファレンスは、PEX フレームワークでサポートされるいずれかの外部言語で記述された PEX コンポーネント用のメソッドを示しています。リモート・クラスに対して生成される ObjectScript プロキシ・クラスは、EnsLib.PEX パッケージ内の ObjectScript クラスから継承します。

A.1 メソッドに関する一般的な注意事項

メソッドをオーバーライドしてプロダクション・コンポーネントを実装する際は、以下の点に留意してください。

- 各プロダクション・コンポーネントは、すべての抽象メソッドをオーバーライドする必要があります。
- ネイティブの相互運用メソッドは 1 つの入力引数と 1 つの出力引数を使用してステータスを返しますが、対応する PEX メソッドは 1 つの入力引数を取り、戻り値として出力引数を返します。
- 例外により PEX メソッドのすべてのエラー処理が行われます。
- 入力および出力引数として永続オブジェクトを必要としないネイティブの相互運用メソッドでは、対応する PEX メソッドも引数として任意のオブジェクトを使用して値を返すことができます。PEX は、外部言語サーバのフォワード・プロキシとリバース・プロキシを使用して、任意のオブジェクトを適切にマッピングします。
- 引数として永続オブジェクトを必要とするネイティブの相互運用メソッド (他のプロセスにメッセージを送信するメソッドなど) では、対応する PEX メソッドで引数として PEX メッセージを使用して値を返すことができます。このようなメソッドの例には、SendRequestSync、SendRequestAsync、OnRequest、OnResponse、OnMessage があります。
- コールバック・メソッドをオーバーライドする際は、メッセージ・クラスをカスタマイズした場合でも、そのメソッドの仮仕様を変更しないでください。引数タイプはオブジェクトとして残ります。

A.2 ビジネス・オペレーション

ビジネス・オペレーションでは、オプションでアダプタを使用して送信メッセージを処理できます。ビジネス・オペレーションにアダプタが指定されている場合、オペレーションはそのアダプタを使用して外部システムにメッセージを送信します。アダプタとしては、PEX アダプタまたは ObjectScript アダプタのいずれかを使用できます。

A.2.1 OnMessage() メソッド

OnMessage() メッセージは、ビジネス・オペレーションが別のプロダクション・コンポーネントからメッセージを受信したときに呼び出されます。通常、そのオペレーションは、受信したメッセージを外部システムに送信するか、ビジネス・プロセスまたは別のビジネス・オペレーションに転送します。オペレーションにアダプタが指定されている場合、そのオペレーションは Adapter.invoke() メソッドを使用して、外部システムにメッセージを送信するメソッドをアダプタで呼び出します。オペレーションが、メッセージを別のプロダクション・コンポーネントに転送するよう指定されている場合、そのオペレーションは、SendRequestAsync() メソッドまたは SendRequestSync() メソッドを使用します。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：(request)

- request ー Object 型。ビジネス・オペレーション宛ての受信メッセージが含まれます。

OnMessage メソッドの実装は、Object 型の単一パラメータを使用してユーザが行う必要があります。このメソッド内で、呼び出し元から渡される実際の型にこのパラメータをキャストできます。

戻り値：オブジェクト

A.2.2 OnInit() メソッド

OnInit() は、コンポーネントが起動されるときに呼び出されます。OnInit() を使用すると、そのコンポーネントで必要な構造を初期化できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.2.3 OnTearDown() メソッド

OnTearDown() メソッドは、コンポーネントが終了される前に呼び出されます。OnTeardown() を使用すると、構造を解放できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.2.4 SendRequestAsync() メソッド

SendRequestAsync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに非同期に送信します。

パラメータ：(target, request [, description])

- target ー要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポートされているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。

- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

戻り値 : void

A.2.5 SendRequestSync() メソッド

SendRequestSync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに同期的に送信します。

パラメータ : (target, request [,timeout [, description]])

- target ー要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポートされているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。
- timeout ー送信要求をエラーとして扱うまで待機する時間を秒数で指定するオプションの整数。既定値は -1 です。これは永久に待機することを意味します。
- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

A.2.6 LOGINFO() メソッド、LOGALERT() メソッド、LOGWARNING() メソッド、LOGERROR() メソッド、および LOGASSERT() メソッド

これらのログ・メソッドは、プロダクション・ログに書き込みます。プロダクション・ログは、管理ポータルでコンポーネントの [ログ] タブに表示できます。これらのメソッドは同じパラメータを使用し、ログ・メッセージの種類のみが異なります。

- LOGINFO() のメッセージの種類は情報です。
- LOGALERT() のメッセージの種類はアラートです。
- LOGWARNING() のメッセージの種類は警告です。
- LOGERROR() のメッセージの種類はエラーです。
- LOGASSERT() のメッセージの種類はアサートです。

パラメータ : (message)

- message ー ログに書き込まれる文字列。

A.3 ビジネス・プロセス

ビジネス・プロセス・クラスには通常、プロダクションのほとんどのロジックが含まれます。ビジネス・プロセスは、ビジネス・サービス、別のビジネス・プロセス、またはビジネス・オペレーションからメッセージを受信できます。また、受信したメッセージに変更を加えるか、メッセージを別の形式に変換するか、またはメッセージの内容に基づいてメッセージをルーティングすることができます。ビジネス・プロセスは、メッセージをビジネス・オペレーションまたは別のビジネス・プロセスにルーティングできます。ビジネス・プロセスのプロパティを永続的にする方法の詳細は、"永続プロパティ" を参照してください。

A.3.1 OnRequest() メソッド

OnRequest() メソッドは、ビジネス・プロセスに送信された要求を処理します。プロダクションは、特定のビジネス・プロセスに関する最初の要求が適切なキューに到着して、実行すべきジョブが割り当てられるたびに、このメソッドを呼び出します。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：(request)

- request ー ビジネス・プロセスに送信された要求メッセージが含まれるオブジェクト。

A.3.2 OnResponse() メソッド

OnResponse() メソッドは、ビジネス・プロセスからターゲットに送信されたメッセージのレスポンスとしてビジネス・プロセスに送信されたレスポンスを処理します。プロダクションは、特定のビジネス・プロセスに関する応答が適切なキューに到着して、実行すべきジョブが割り当てられるたびに、このメソッドを呼び出します。通常、これは、要求の responseRequired パラメータの値が true の場合に、ビジネス・プロセスによって生成された非同期の要求に対するレスポンスです。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：(request, response, callRequest, callResponse, completionKey)

- request ー ビジネス・プロセスに送信された最初の要求メッセージが含まれるオブジェクト。
- response ー 初期メッセージを送信したプロダクション・コンポーネントにこのビジネス・プロセスが返すことができるレスポンス・メッセージが含まれるオブジェクト。
- callRequest ー ビジネス・プロセスがターゲットに送信した要求が含まれるオブジェクト。
- callResponse ー 受信レスポンスが含まれるオブジェクト。
- completionKey ー 送信処理を行う SendAsync() メソッドの completionKey パラメータに指定された completionKey が含まれる文字列。

A.3.3 OnComplete() メソッド

OnComplete() メソッドは、ビジネス・プロセスがターゲットに送信した要求に対するすべてのレスポンスを受信して処理した後に呼び出されます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：(request, response)

- request ー ビジネス・プロセスに送信された最初の要求メッセージが含まれるオブジェクト。
- response ー 初期メッセージを送信したプロダクション・コンポーネントにこのビジネス・プロセスが返すことができるレスポンス・メッセージが含まれるオブジェクト。

A.3.4 OnInit() メソッド

OnInit() メソッドは、コンポーネントが起動されるときに呼び出されます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.3.5 OnTearDown() メソッド

OnTearDown() メソッドは、コンポーネントが終了される前に呼び出されます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.3.6 Reply() メソッド

Reply() メソッドは、ビジネス・プロセスに最初の要求を送信したプロダクション・コンポーネントに、指定されたレスポンスを送信します。

パラメータ：(response)

- response ー レスポンス・メッセージが含まれるオブジェクト。

A.3.7 SendRequestAsync() メソッド

SendRequestAsync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに非同期に送信します。

パラメータ：(target, request [, responseRequired [, completionKey [, description]]])

- target ー 要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポートされているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。
- responseRequired ー ターゲットがレスポンス・メッセージを送信する必要があるかどうかを指定するブーリアン値。
- completionKey ー レスポンス・メッセージと共に送信される文字列。
- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

A.3.8 SendRequestSync() メソッド

SendRequestSync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに同期的に送信します。

パラメータ：(target, request [,timeout [, description]])

- target ー 要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポート

されているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。

- timeout ー送信要求をエラーとして扱うまで待機する時間を秒数で指定するオプションの整数。既定値は -1 です。これは永久に待機することを意味します。
- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

A.3.9 SetTimer() メソッド

SetTimer() メソッドは、ビジネス・プロセスがすべてのレスポンスを待機する最大時間を指定します。

パラメータ：(timeout [, completionKey])

- timeout ー 秒数を指定する整数または期間を指定する文字列。例えば、“PT15S” は、15 秒のプロセッサ時間を表します。
- completionKey ー 最大時間を超過した場合にレスポンスと共に返される文字列。

A.3.10 LOGINFO() メソッド、LOGALERT() メソッド、LOGWARNING() メソッド、LOGERROR() メソッド、および LOGASSERT() メソッド

これらのログ・メソッドは、プロダクション・ログに書き込みます。プロダクション・ログは、管理ポータルでコンポーネントの [ログ] タブに表示できます。これらのメソッドは同じパラメータを使用し、ログ・メッセージの種類のみが異なります。

- LOGINFO() のメッセージの種類は情報です。
- LOGALERT() のメッセージの種類はアラートです。
- LOGWARNING() のメッセージの種類は警告です。
- LOGERROR() のメッセージの種類はエラーです。
- LOGASSERT() のメッセージの種類はアサートです。

パラメータ：(message)

- message ー ログに書き込まれる文字列。

A.4 ビジネス・サービス

ビジネス・サービス・クラスは、外部システムからデータを受信して、プロダクション内のビジネス・プロセスまたはビジネス・オペレーションにそのデータを送信します。ビジネス・サービスでは、アダプタを使用して外部システムにアクセスできます。

A.4.1 OnProcessInput() メソッド

OnProcessInput() メソッドは、受信アダプタからアダプタの ProcessInput() メソッド経由でメッセージを受信し、それをターゲットのビジネス・プロセスまたはビジネス・オペレーションに転送します。ビジネス・サービスでアダプタを指定しない場合、既定のアダプタがメッセージを指定せずに OnProcessInput() メソッドを呼び出し、ビジネス・サービスは外部システムからデータを受信して、それを検証します。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：(message)

- message ー 受信アダプタがビジネス・サービスに送信したデータが含まれるオブジェクト。message には、受信アダプタおよびビジネス・サービスで受け入れられるものであればどのような構造を使用してもかまいません。message は、Message のサブクラスまたは IRISObject である必要はなく、通常はデータベース内に永続化されません。

A.4.2 OnInit() メソッド

OnInit() メソッドは、コンポーネントが起動されるときに呼び出されます。OnInit() メソッドを使用すると、そのコンポーネントで必要な構造を初期化できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.4.3 OnTearDown() メソッド

OnTearDown() メソッドは、ビジネス・コンポーネントが終了される前に呼び出されます。OnTeardown() メソッドを使用すると、構造を解放できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.4.4 ProcessInput() メソッド

受信アダプタがビジネス・サービスの ProcessInput0 メソッドを呼び出し、それを受けて、ProcessInput() メソッドは、カスタム・コードである OnProcessInput() メソッドを呼び出します。

パラメータ：(input)

- input ー 受信アダプタに受け入れられる任意の構造を持つオブジェクト。ProcessInput() メソッドに渡されるパラメータは、永続オブジェクトである必要はありません。

戻り値：オブジェクト

A.4.5 SendRequestAsync() メソッド

SendRequestAsync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに非同期に送信します。

パラメータ：(target, request [, description])

- target ー 要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポートされているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。
- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

A.4.6 SendRequestSync() メソッド

SendRequestSync() メソッドは、指定されたメッセージをターゲットのビジネス・プロセスまたはビジネス・オペレーションに同期的に送信します。

パラメータ：(target, request [,timeout [, description]])

- target ー 要求を受信するビジネス・プロセスまたはビジネス・オペレーションの名前を指定する文字列。target は、コンポーネントのクラス名ではなく、プロダクション定義の Item Name プロパティに指定されているコンポーネント名です。
- request ー ターゲットに送信するメッセージを指定します。request には、Message クラスのサブクラスであるクラスまたは IRISObject クラスを使用します。ターゲットが組み込みの ObjectScript コンポーネントである場合、IRISObject クラスを使用する必要があります。IRISObject クラスを使用すると、PEX フレームワークにより、ターゲットでサポートされているクラスにメッセージを変換できます。ターゲットが組み込みの ObjectScript コンポーネントでない場合は、Message クラスのサブクラスを使用できます。
- timeout ー 送信要求をエラーとして扱うまで待機する時間を秒数で指定するオプションの整数。既定値は -1 です。これは永久に待機することを意味します。
- description ー メッセージ・ヘッダに説明のプロパティを設定するオプションの文字列パラメータ。

A.4.7 LOGINFO() メソッド、LOGALERT() メソッド、LOGWARNING() メソッド、LOGERROR() メソッド、および LOGASSERT() メソッド

これらのログ・メソッドは、プロダクション・ログに書き込みます。プロダクション・ログは、管理ポータルでコンポーネントの [ログ] タブに表示できます。これらのメソッドは同じパラメータを使用し、ログ・メッセージの種類のみが異なります。

- LOGINFO() のメッセージの種類は情報です。
- LOGALERT() のメッセージの種類はアラートです。
- LOGWARNING() のメッセージの種類は警告です。
- LOGERROR() のメッセージの種類はエラーです。
- LOGASSERT() のメッセージの種類はアサートです。

パラメータ：(message)

- message ー ログに書き込まれる文字列。

A.5 Director

Director クラスは、ポーリングされないビジネス・サービスの場合に使用します。つまり、呼び出し間隔でプロダクション・フレームワークによって (受信アダプタ経由で) 自動的に呼び出されることはないビジネス・サービスの場合に使用します。これらのビジネス・サービスは、Director.CreateBusinessService() メソッドを呼び出すことで、カスタム・アプリケーションによって作成されます。

A.5.1 CreateBusinessService() メソッド

CreateBusinessService() メソッドは、指定されたビジネス・サービスを開始します。

パラメータ：(connection, target)

- connection – 使用している言語の外部言語サーバへの接続を指定する IRISConnection オブジェクト。
- target – プロダクション定義でビジネス・サービスの名前を指定する文字列。

戻り値 : businessService – 戻り値には、作成されたビジネス・サービス・インスタンスが含まれます。

A.6 受信アダプタ

InboundAdapter は、ProcessInput() メソッドを呼び出すことで、外部システムからデータを受信し、そのデータを検証して、ビジネス・サービスに送信します。

A.6.1 OnTask() メソッド

OnTask() メソッドは、ビジネス・サービスの CallInterval プロパティで指定された間隔で、プロダクション・フレームワークによって呼び出されます。OnTask() メソッドは、外部システムからデータを受信し、そのデータを検証して、メッセージに含めてビジネス・サービスの OnProcessInput() メソッドに送信します。message には、受信アダプタおよびビジネス・サービスで受け入れられるものであればどのような構造を使用してもかまいません。

抽象メソッド : ユーザが実装する必要があります。

パラメータ : なし

A.6.2 OnInit() メソッド

OnInit() メソッドは、コンポーネントが起動されるときに呼び出されます。OnInit() メソッドを使用すると、そのコンポーネントで必要な構造を初期化できます。

抽象メソッド : ユーザが実装する必要があります。

パラメータ : なし

戻り値 : void

A.6.3 OnTearDown() メソッド

OnTearDown() メソッドは、ビジネス・コンポーネントが終了される前に呼び出されます。OnTeardown() メソッドを使用すると、構造を解放できます。

抽象メソッド : ユーザが実装する必要があります。

パラメータ : なし

戻り値 : void

A.6.4 LOGINFO() メソッド、LOGALERT() メソッド、LOGWARNING() メソッド、LOGERROR() メソッド、および LOGASSERT() メソッド

これらのログ・メソッドは、プロダクション・ログに書き込みます。プロダクション・ログは、管理ポータルでコンポーネントの [ログ] タブに表示できます。これらのメソッドは同じパラメータを使用し、ログ・メッセージの種類のみが異なります。

- LOGINFO() のメッセージの種類は情報です。
- LOGALERT() のメッセージの種類はアラートです。
- LOGWARNING() のメッセージの種類は警告です。

- LOGERROR() のメッセージの種類はエラーです。
- LOGASSERT() のメッセージの種類はアサートです。

パラメータ：(message)

- message ー ログに書き込まれる文字列。

A.7 送信アダプタ

送信アダプタ・クラスは、外部システムにデータを送信します。

A.7.1 OnInit() メソッド

OnInit() メソッドは、コンポーネントが起動されるときに呼び出されます。OnInit() メソッドを使用すると、そのコンポーネントで必要な構造を初期化できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.7.2 OnTearDown() メソッド

OnTearDown() メソッドは、ビジネス・コンポーネントが終了される前に呼び出されます。OnTeardown() メソッドを使用すると、構造を解放できます。

抽象メソッド：ユーザが実装する必要があります。

パラメータ：なし

戻り値：void

A.7.3 Invoke() メソッド

Invoke() メソッドを使用すると、BusinessOperation で、OutboundAdapter に定義されているあらゆるパブリック・メソッドを実行できます。

パラメータ：(methodname, arguments)

- methodname ー アウトバウンド・アダプタに定義されている実行対象のメソッドの名前を指定します。
- arguments ー 指定したメソッドの引数が含まれます。

A.7.4 LOGINFO() メソッド、LOGALERT() メソッド、LOGWARNING() メソッド、LOGERROR() メソッド、および LOGASSERT() メソッド

これらのログ・メソッドは、プロダクション・ログに書き込みます。プロダクション・ログは、管理ポータルでコンポーネントの [ログ] タブに表示できます。これらのメソッドは同じパラメータを使用し、ログ・メッセージの種類のみが異なります。

- LOGINFO() のメッセージの種類は情報です。
- LOGALERT() のメッセージの種類はアラートです。

- LOGWARNING() のメッセージの種類は警告です。
- LOGERROR() のメッセージの種類はエラーです。
- LOGASSERT() のメッセージの種類はアサートです。

パラメータ：(message)

- message ー ログに書き込まれる文字列。

A.8 Message

Message クラスは、コンポーネント間で送信される永続メッセージのスーパークラス・クラスである抽象クラスです。Message クラスには、プロパティもメソッドもありません。プロパティを追加するには、ユーザが Message クラスのサブクラスを作成します。PEX フレームワークは、Message クラスから派生したオブジェクトに永続性を提供します。