Nimbus業務フロー仕様書

2016年9月6日

目次

[1. 概要 1](#_Toc462157087)

[2. インタフェース設計 3](#_Toc462157088)

[2.1. BeanFlowInvoker 3](#_Toc462157089)

[2.2. BeanFlowInvokerFactory 6](#_Toc462157090)

[2.3. BeanFlowInvokerServer 7](#_Toc462157091)

[3. 実装設計 8](#_Toc462157092)

[3.1. BeanFlowInvoker 8](#_Toc462157093)

[3.1.1. jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl 8](#_Toc462157094)

[3.1.2. jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl2 13](#_Toc462157095)

[3.2. BeanFlowInvokerFactory 43](#_Toc462157096)

[3.2.1. jp.ossc.nimbus.service.beancontrol.DefaultBeanFlowInvokerFactoryService 43](#_Toc462157097)

[3.2.2. jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerFactoryGroupService 45](#_Toc462157098)

[3.2.3. jp.ossc.nimbus.service.beancontrol.ClientBeanFlowInvokerFactoryService 46](#_Toc462157099)

[3.3. BeanFlowInvokerServer 48](#_Toc462157100)

[3.3.1. jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerServerService 48](#_Toc462157101)

[3.4. Compiler 50](#_Toc462157102)

# 概要

業務フローとは、業務ロジックの再利用性及び保守性を高めるために、アプリケーションを細分化して、細分化した小さなアプリケーションをつなぎ合わせて、業務処理の単位に組み上げる処理フローのことである。

業務処理１

業務ロジックa

業務ロジックe

業務ロジックb

業務ロジックc

業務ロジックd

業務処理2

業務ロジックf

業務ロジックg

業務フローを組み上げるために必要な機能には、以下のようなものがある。

* 業務ロジックへのリソース、パラメータ注入
* 子処理の同期呼び出し
* 子処理の非同期呼び出し及び待ち合わせ
* 処理結果による条件分岐
* 繰り返し処理
* トランザクション制御
* 流量制御
* 例外処理

これらを実現する機能が、Nimbus 業務フローである。Nimbus 業務フローでは、Javaで開発した業務ロジックを使って、XMLファイルで業務フローを定義することができる。

**業務フロー１XML**

業務処理１

Java

業務ロジックa

**業務フロー２XML**

Java

業務ロジックb

Java

業務ロジックc

Java

業務ロジックd

業務処理2

Java

業務ロジックf

Java

業務ロジックe

Java

業務ロジックg

XMLファイル読み込み

BeanFlowInvoker

業務処理1

BeanFlowInvokerFactory

invokeFlow (引数)

createFlow(業務フロー名)

BeanFlowInvoker

業務処理2

JavaDocは、以下を参照。

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/beancontrol/package-summary.html>

サンプルコードは、以下のzipファイル内のbeancontrolを参照。

<https://osdn.jp/projects/nimbus/downloads/64236/nimbus-sample-1.2.3.zip>

# インタフェース設計

## BeanFlowInvoker

業務処理の呼び出し機能を提供するインタフェースである。

invokeFlow(input)

**BeanFlowInvoker**

invokeFlow(input)呼び出す事で、業務処理に必要な入力オブジェクトを引数として渡し、業務処理の実行結果を出力オブジェクトとして、戻り値で受け取る同期処理実行ができる。

createMonitor()

**BeanFlowInvoker**

任意の他のスレッド

BeanFlowMonitor

invokeFlow(input, monitor)

suspend()

resume()

BeanFlowInvokerからcreateMonitor()でBeanFlowMonitorを生成し、invokeFlow()の引数として渡しておくことで、invokeFlow()を呼び出して業務フローを実行しているスレッドとは、別のスレッドから業務フローの一時停止や再開、中止といった制御を行ったり、実行中の処理を特定したりといった、状態監視及び制御が可能となる。

invokeAsynchFlow(input, monitor, false, maxAsynchWait)

**BeanFlowInvoker**

invokeAsynchFlow()を呼び出す事で、処理要求をキューに投入して、呼び出しスレッドとは別のスレッドで業務処理を実行する非同期実行ができる。第３引数のfalseは、戻り値を必要としない非同期呼び出しである事を示す。第４引数のmaxAsynchWaitは、非同期処理を実行するスレッドは、一定数に限られており、処理要求がキューに滞留する可能性があるため、滞留が多い時は処理要求を諦めるように指定するためのもので、指定された値以上の滞留があるときは、UnavailableFlowExceptionがthrowされ、処理要求が拒否される。

invokeAsynchFlow(input, monitor, true, maxAsynchWait)

**BeanFlowInvoker**

getAsynchReply (context, monitor, true, timeout, isCancel)

invokeAsynchFlow()の第３引数をtrueで呼び出す事で、処理要求をキューに投入して、呼び出しスレッドとは別のスレッドで業務処理を実行して、処理終了後に応答を取得する、非同期待ち合わせ処理が実行できる。invokeAsynchFlow()を呼び出すと戻り値で、非同期実行処理を識別するコンテキストが受け取れる。非同期処理の応答を待ち合わせるgetAsynchReply()の第一引数でコンテキストを指定する事で、要求した非同期処理を特定して待ち合わせる事が可能になる。第４引数で、待ち合わせのタイムアウトを指定できる。タイムアウトした場合は、BeanFlowAsynchTimeoutExceptionがthrowされる。また、タイムアウトした場合に、要求していた処理を取り消すかどうかを第５引数で指定できる。但し、取り消し要求が受理されるのは、非同期処理要求が、要求受付キューから引き抜かれる前までである。

invokeAsynchFlow(input, monitor, callback, maxAsynchWait)

**BeanFlowInvoker**

BeanFlowAsynchInvokeCallback

reply(output, throwable)

BeanFlowAsynchInvokeCallbackを引数に持つinvokeAsynchFlow()を呼び出す事で、処理要求をキューに投入して、呼び出しスレッドとは別のスレッドで業務処理を実行して、処理終了後に呼び戻す、非同期コールバック処理が実行できる。

以下に、ソースコードの例を示す。

import jp.ossc.nimbus.core.\*;

import jp.ossc.nimbus.service.beancontrol.\*;

import jp.ossc.nimbus.service.beancontrol.interfaces.\*;

BeanFlowInvokerFactory factory = (BeanFlowInvokerFactory)ServiceManagerFactory.getServiceObject(“BeanFlowInvokerFactory”);

BeanFlowInvoker invoker = factory.createFlow(“業務処理1”);

Object output = invoker.invokeFlow(input);

以下に、BeanFlowInvokerインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl | レガシーな業務フロー定義XMLを解釈する実装クラスである。 |
| 2 | jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl2 | 新しい業務フロー定義XMLを解釈するデフォルト実装クラスである。 |

## BeanFlowInvokerFactory

業務処理の実行単位であるBeanFlowInvokerを生成するファクトリインタフェースである。

createFlow(業務フロー名)

**BeanFlowInvokerFactory**

BeanFlowInvoker

業務処理を特定する単位である業務フローに名前を付けて、業務フロー名指定で、BeanFlowInvokerを生成する。

以下に、BeanFlowInvokerFactoryインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.beancontrol.DefaultBeanFlowInvokerFactoryService | 業務フローを生成するデフォルト実装クラスである。 |
| 2 | jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerFactoryGroupService | 複数のBeanFlowInvokerFactoryをグルーピングして１つに見せるラッパークラスである。 |
| 3 | jp.ossc.nimbus.service.beancontrol.ClientBeanFlowInvokerFactoryService | リモートに分散したBeanFlowInvokerServerを呼び出し、業務フローをリモート分散実行する実装クラスである。 |

## BeanFlowInvokerServer

リモートから業務フローを実行するサーバ側のサービスが実装するRMIインタフェースである。

クライアント側

createFlow(flowName)

BeanFlowInvokerFactory

invokeFlow(input)

BeanFlowInvoker

RMI通信

サーバ側

**BeanFlowInvokerServer**

BeanFlowInvoker

BeanFlowInvokerFactory

以下に、BeanFlowInvokerServerインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerServerService | デフォルト実装クラスである。 |

# 実装設計

インタフェース設計で示した各インタフェースのうち、機能性のあるインタフェースの実装クラスの機能を説明する。

また、インタフェース設計に現れなかった、クラスの機能も説明する。

## BeanFlowInvoker

### jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl

レガシーな業務フローXML定義ファイルを解釈するBeanFlowInvokerインタフェースの実装クラスである。但し、レガシーな実装であるため、非同期呼び出し機能は、サポートしない。

業務フロー

XML定義ファイル

XMLファイル読み込み

DefaultBeanFlowInvokerFactoryService

fillInstance(DOM)

業務フローDOM

setBeanFlowInvokerAccessClass(Class)

**BeanFlowInvokerAccessImpl**

後述するBeanFlowInvokerFactoryインタフェースの実装クラスDefaultBeanFlowInvokerFactoryServiceにBeanFlowInvokerインタフェースの実装クラス名と、業務フローをXMLで記述した定義ファイルのパスを設定しておき、サービスを起動すると、業務フローXML定義ファイルを読み込み、BeanFlowInvokerAccessImpl のfillInstance(dom)メソッドで、そのDOM（JavaのXML文書を扱うJAXP規格の一部である**D**ata **O**bject **M**odel）の読み込みが行われる。これにより、XMLに記述した業務フローの内容がBeanFlowInvokerAccessImplに取り込まれ、BeanFlowInvokerFactoryのcreateFlow(flowName)の戻り値のBeanFlowInvokerとして提供される。

業務ロジックBean

・リソース

・設定

・処理結果

・実行()

業務ロジックBean

・リソース

・設定

・処理結果

・実行()

Javaで開発

XMLで開発

業務フローXML定義ファイル

業務ロジックBean 1

業務ロジックBean 2

業務ロジックBean 3

基本的な業務フローの開発スタイルは、リソースや設定値を渡すsetterを持った業務ロジックBeanに、引数なし、戻り値なしの処理実行メソッドと、処理結果を取得するgetterを持たせた業務ロジックをJavaで開発する。この業務ロジックBeanを直列に並べて処理を実行する業務フロー定義ＸＭＬファイルを記述する。

以下に、業務フロー定義ＸＭＬファイルの仕様を定義する。

子要素の”+”は、１個以上。”\*”は、０個以上現れる事を意味する。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 要素名 | 子要素 | 説明 | |
| 1 | flows | flow+ | ルート要素。 | |
| 2 | flow | step+  alias\*  resource\* | 業務フローを表す。 | |
| 属性名 | 必須/任意 | 説明 |
| name | 必須 | 業務フロー名を定義する。 |
| 3 | step | attribute\* | 業務フロー内の処理単位であるステップを表す。 | |
| 属性名 | 必須/任意 | 説明 |
| name | 必須 | ステップ名を定義する。 |
| className | 必須 | このステップで対象となる業務ロジックBeanのクラス名を定義する。 |
| methodName | 必須 | 業務ロジックBeanの処理実行メソッドのメソッド名を定義する。 |
| type | 任意 | このステップで対象となる業務ロジックBeanの取得方法を定義する。  ・new：classNameで指定した業務ロジックBeanを生成する。  ・input：業務フローの引数を参照する  ・step.ステップ名：指定したステップの対象となる業務ロジックBeanを参照する。  指定なしの場合は、new |
| getterName | 任意 | ステップの戻り値を取得するgetterのメソッド名を定義する。  最後に呼び出されたgetterの戻り値がフローの戻り値となる。 |
| 4 | alias | なし | 業務フロー名の別名を定義する。 | |
| 属性名 | 必須/任意 | 説明 |
| name | 必須 | 別名を定義する。 |
| 5 | resource | なし | 業務フローで使用するリソースを定義する。 | |
| 属性名 | 必須/任意 | 説明 |
| name | 必須 | リソース名を定義する。 |
| service | 必須 | リソースのサービス名を定義する。  ここで指定するサービスは、jp.ossc.nimbuse.service.resource.ResourceFactoryインタフェースを実装していなければならない。 |
| key | 必須 | リソースを取得する際に使用するキーを定義する。  ResourceFactory#makeResource(key)の引数として、使用する。 |
| trancontrol | 任意 | リソースのトランザクション制御を行うかどうかを定義する。  制御を行う場合は、true。  指定しない場合は、false。 |
| tranclose | 任意 | リソースの終了処理を行うかどうかを定義する。  行わない場合は、false。  指定しない場合は、true。 |
| 6 | attribute | なし | 業務ロジックBeanに設定する属性を定義する。 | |
| 属性名 | 必須/任意 | 説明 |
| name | 必須 | 属性名を定義する。  Java Beanのsetterのプロパティ名に該当するものを指定する。 |
| type | 必須 | 設定する属性の値を、取得する方法を定義する。  ・value：即値  ・service：サービスを取得する。  ・step：ステップから取得する。  ・resource：リソースを取得する。  ・input：引数を取得する。 |
| value | 任意 | 業務ロジックBeanの属性に、設定する値を定義する。  要素の内容に指定する事もできる。  type属性の値が、service、step、resourceの場合は、必須。  type属性の値によって、設定すべきものが変わる。  ・value：setterの引数に与える値を文字列で指定する。setterの引数の型に応じて、java.beans.PropertyEditorで変換を行う。  ・service：サービス名  ・step：ステップ名#this。または、ステップ名#getterメソッド名。  ・resource：リソース名  ・input：メソッド名、またはMapのキー。指定しない場合は、引数そのもの。 |

以下に、業務フロー定義XMLファイルの例を示す。

<?xml version="1.0" encoding="UTF-8"?>

<flows>

<flow name=”BeanFlow1”>

<alias name=”flow1”/>

<resource name=”ConnectionResource”

service=”Nimbus#ConnectionResourceFactory”

key=””/>

<step name=”step1”

className=”sample.LogicBean1”

methodName=”load”>

<attribute name=”Connection”

type=”resource”

value=” ConnectionResource”/>

<attribute name=”TableName”

type=”value”

value=”USER\_MST”/>

</step>

<step name=”step2”

className=”sample.LogicBean1”

type=”step.step1”

methodName=”update”

getterName=”getResult”>

<attribute name=”Connection”

type=”resource”

value=” ConnectionResource”/>

</step>

</flow>

</flows>

### jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerAccessImpl2

新しい業務フローXML定義ファイルを解釈するBeanFlowInvokerインタフェースの実装クラスである。

業務フロー

XML定義ファイル

XMLファイル読み込み

DefaultBeanFlowInvokerFactoryService

fillInstance(DOM)

業務フローDOM

**BeanFlowInvokerAccessImpl2**

後述するBeanFlowInvokerFactoryインタフェースの実装クラスDefaultBeanFlowInvokerFactoryServiceのBeanFlowInvokerインタフェース実装クラスのデフォルト値となっている。DefaultBeanFlowInvokerFactoryServiceに、業務フローをXMLで記述した定義ファイルのパスを設定しておき、サービスを起動すると、業務フローXML定義ファイルを読み込み、BeanFlowInvokerAccessImpl2 のfillInstance(dom)メソッドで、そのDOM（JavaのXML文書を扱うJAXP規格の一部である**D**ata **O**bject **M**odel）の読み込みが行われる。これにより、XMLに記述した業務フローの内容がBeanFlowInvokerAccessImpl2に取り込まれ、BeanFlowInvokerFactoryのcreateFlow(flowName)の戻り値のBeanFlowInvokerとして提供される。

業務ロジックBean

・リソース

・設定

・任意のメソッド()

業務ロジックBean

・リソース

・設定

・処理結果

・実行()

Javaで開発

XMLで開発

業務フローXML定義ファイル

条件分岐、繰り返し、

子フロー呼び出しなど

多彩なフロー制御

業務ロジックBean 1

業務ロジックBean 2

業務ロジックBean 3

基本的な業務フローの開発スタイルは、リソースや設定値を渡すsetterを持った業務ロジックBeanに、任意のメソッドを持たせた業務ロジックをJavaで開発する。この業務ロジックBeanを、条件分岐、繰り返し、子フロー呼び出しなどで組み立てて、業務フロー定義ＸＭＬファイルに記述する。

レガシーな実装に比べると、業務ロジックBeanの設計に対する制約がなく、直列なフロー以外に分岐が行えたり、子フロー呼び出しが可能な事でフローの粒度設計が可能になったりしている。また、非同期処理もサポートしているため、並列処理も可能になる。

以下に、業務フロー定義ＸＭＬファイルの仕様を定義する。

子要素の”+”は、１個以上。”\*”は、０個以上。”?”は0個か１個、現れる事を意味する。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | 要素名 | 親要素 | 子要素 | 説明 | |
| 1 | flows | なし | flow\* | ルート要素。 | |
| 2 | flow | flows | step\*  alias\*  override\*  resource\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  catch\*  finally?  return? | 業務フローを表す。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | 業務フロー名を定義する。 |
| maxRunThreads | | 任意 | 同時実行数を定義する。  指定しない場合は、制限しない。 |
| maxWaitThreads | | 任意 | 同時実行待ち数を定義する。  同時実行数が限界まで来ている時に、同時実行数に空きが出来るまで待機させられる。  その際に、待機している数が多い場合は、長い時間待たされる可能性がある。  そのため、その待機数が多い場合は、待たずに諦めたいことがある。  この属性は、その待機数の最大値を定義する。  待機せずに諦めた時は、UnavailableFlowExceptionをthrowする。  この属性を定義しない場合は、同時実行待ち数を制限しない。  また、maxRunThreads属性が指定されていない場合は、無効。 |
| timeout | | 任意 | 同時実行待ち時間[ms]を定義する。  同時実行数が限界まで来ている時に、同時実行数に空きが出来るまで待機させられる。  その際に、諦めずに待機する時間を定義する。  指定された時間だけ待機して諦めた時は、UnavailableFlowExceptionをthrowする。  この属性を定義しない場合は、無限に待ち続ける。  また、maxRunThreads属性が指定されていない場合は、無効。 |
| forceFreeTimeout | | 任意 | 業務フローの同時実行数制御タイムアウト[ms]を定義する。  同時実行数制御を行っている場合で、実行時間がこの時間を過ぎた場合には、そのスレッドが確保している同時実行数を開放するためのもの。  この属性を定義しない場合は、実行時間は無制限である。  また、maxRunThreads属性が指定されていない場合は、無効。 |
| transaction | | 任意 | トランザクション属性を定義する。  Supports以外の値を指定した場合は、JTAを使用してトランザクション制御を行うため、JTA実装を持ったJ2EEコンテナ上で動作させる必要がある。  指定可能な値は、以下。  ・Required：トランザクションがなければ開始、あれば参加する。  ・RequiresNew：トランザクションを開始する。  ・Supports：トランザクションがあれば参加、なければ何もしない。  ・Mandatory：トランザクションがなければ、BeanControlUncheckedExceptionをthrowする。あれば、参加する。  ・Never：トランザクションがあれば、BeanControlUncheckedExceptionをthrowする。  ・NotSupported：トランザクションに参加しない。  指定しない場合は、Supports |
| trantimeout | | 任意 | トランザクションタイムアウト[s]を定義する。  transaction属性がRequiredまたはRequiresNewで、トランザクションがそこで開始される場合のみ有効。 |
| journal | | 任意 | ジャーナルを出力するかどうかをtrue、falseで定義する。  デフォルトは、true。  この要素の子要素全てに適用される。 |
| 3 | alias | flow | なし | 業務フローの別名を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | 業務フローの別名を定義する |
| 4 | override | flow  callflow | なし | callflow要素で定義する子フロー呼び出しの上書き子フロー名を定義する。  上書き子フロー名には、"${スレッドコンテキストプロパティ名}"を含む事が可能で、予めスレッドコンテキスト上に、上書き子フロー名の一部となる名前を設定しておけば、呼び出すフロー名を動的に変える事ができる。  上書き子フロー名のフローが存在しない場合は、name属性で指定した子フロー名が適用される。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | 上書き子フロー名を定義する |
| 5 | resource | flow | なし | 業務フローで使用するリソースを定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | リソース名を定義する。 |
| service | | 必須 | リソースのサービス名を定義する。  ここで指定するサービスは、jp.ossc.nimbuse.service.resource.ResourceFactoryインタフェースを実装していなければならない。 |
| key | | 必須 | リソースを取得する際に使用するキーを定義する。  ResourceFactory#makeResource(key)の引数として、使用する。 |
| trancontrol | | 任意 | リソースのトランザクション制御を行うかどうかを定義する。  制御を行う場合は、true。  指定しない場合は、false。 |
| Tranclose | | 任意 | リソースの終了処理を行うかどうかを定義する。  行わない場合は、false。  指定しない場合は、true。 |
| 6 | step | flow  case  default  if  for  while  catch  finally | target?  field\*  attribute\*  invoke\*  static-invoke\*  catch\*  interpreter?  result?  finally? | 特定の対象に対する、ひとかたまりの処理を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| Name | | 任意 | ステップの名前を定義する。  step-ref要素で他ステップから参照する場合に、ステップを一意に識別するために名前をつける。  同じ名前のステップがある場合は、後勝ちで上書かれていく。  また、他ステップから参照しない場合は、名前を付ける必要はない。 |
| maxRunThreads | | 任意 | ステップの同時実行数を定義する。  flow要素のmaxRunThreads属性と同じ。 |
| maxWaitThreads | | 任意 | ステップの同時実行待ち数を定義する。  flow要素のmaxWaitThreads属性と同じ。 |
| timeout | | 任意 | ステップの同時実行待ち時間を定義する。  flow要素のtimeout属性と同じ。 |
| forceFreeTimeout | | 任意 | ステップの同時実行数制御タイムアウトを定義する。  flow要素のforceFreeTimeout属性と同じ。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| suspend | | 任意 | 実行の一時中断を許容するかどうかを定義する。  デフォルトは、true |
| Stop | | 任意 | 実行の停止を許容するかどうかを定義する。  デフォルトは、true |
| 7 | target | step  for | input  object  service-ref  resource-ref  step-ref  static-field-ref  static-invoke  var  expression | 処理の対象となるオブジェクトを定義する。  step要素の子要素として定義する場合は、そのステップで各インジェクションを行う対象のオブジェクトを定義する。  for要素の子要素として定義する場合は、繰り返し対象のオブジェクトを定義する。  繰り返し対象のオブジェクトには、以下のクラスのオブジェクトを指定できる。  ・任意のクラスの配列  ・java.util.Collectionインタフェースを実装したクラス  ・java.sql.ResultSetインタフェースを実装したクラス  ・jp.ossc.nimbus.recset.RecordSetクラス | |
| 属性名 | | 必須/任意 | 説明 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 8 | result | step | attribute  field  invoke  this  input  static-invoke  static-field-ref  object  expression  step-ref  var | ステップの結果を定義する。  ステップに結果が必要な場合に定義する。  この要素によって定義されたステップ結果は、step-ref要素で参照する事ができる。  但し、step要素にname属性を指定しなければならない。  また、内容に文字列を指定して、それを結果にすることもできる。 | |
| 属性名 | | 必須/任意 | 説明 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 9 | callflow | flow  case  default  if  for  while  catch  finally | override\*  argument\*  catch\*  finally?  callback? | 子フロー呼び出しを定義する。  子フローの呼び出し結果は、ステップ結果の参照と同様で、step-ref要素で行う。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | 呼び出す子フロー名を定義する。 |
| stepname | | 任意 | ステップ名を定義する。  定義しない場合は、name属性で指定した子フロー名がそのままステップ名になる。 |
| transaction | | 任意 | トランザクション属性を定義する。  Supports以外の値を指定した場合は、JTAを使用してトランザクション制御を行うため、JTA実装を持ったJ2EEコンテナ上で動作させる必要がある。  指定可能な値は、以下。  ・Required：トランザクションがなければ開始、あれば参加する。  ・RequiresNew：トランザクションを開始する。  ・Supports：トランザクションがあれば参加、なければ何もしない。  ・Mandatory：トランザクションがなければ、BeanControlUncheckedExceptionをthrowする。あれば、参加する。  ・Never：トランザクションがあれば、BeanControlUncheckedExceptionをthrowする。  ・NotSupported：トランザクションに参加しない。  指定しない場合は、Supports。  flow要素のtransaction属性よりも、こちらの指定が優先される。 |
| trantimeout | | 任意 | トランザクションタイムアウト[s]を定義する。  transaction属性が指定されていない場合は、無効。 |
| factory | | 任意 | BeanFlowInvokerFactoryサービスのサービス名を定義する。  factory属性が指定されていない場合は、この業務フローを生成したBeanFlowInvokerFactoryサービスを使用する。 |
| override | | 任意 | 呼び出された側でオーバーライドを判断するかどうかを定義する。  デフォルトは、true。  子要素にoverride要素がある場合は、暗黙的にfalseになる。 |
| asynch | | 任意 | 非同期で実行するかどうかを定義する。  デフォルトは、false。 |
| reply | | 任意 | 非同期呼び出しを行った場合の応答を必要とするかどうかを定義する。  デフォルトは、false。  asynch属性がfalseの場合は、無効。 |
| maxAsynchWait | | 任意 | 非同期呼び出しを行った場合の非同期実行待機数の最大値を定義する。  非同期実行待機数が多い時は、非同期実行スレッドに空きが出来るまで待機させられる。  その際に、待機している数が多い場合は、長い時間待たされる可能性がある。  そのため、その待機数が多い場合は、待たずに諦めたいことがある。  この属性は、その待機数の最大値を定義する。  待機せずに諦めた時は、UnavailableFlowExceptionをthrowする。  この属性を定義しない場合は、非同期実行待機数を制限しない。  また、asynch属性がfalseの場合は、無効。 |
| journal | | 任意 | ジャーナルを出力するかどうかをtrue、falseで定義する。  デフォルトは、true。  この要素の子要素全てに適用される。 |
| 10 | argument | callflow  constructor  static-invoke  invoke | input  object  service-ref  resource-ref  step-ref  static-field-ref  static-invoke  var  this  expression | 引数を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| type | | 任意 | 引数の型を定義する。  argument要素の内容または子要素で定義する引数値の型が、メソッドの引数の型と異なる場合に、メソッドの引数の型を明示するために使用する。  argument要素の内容または子要素で定義する引数値の型が、メソッドの引数の型と等しい場合には、指定する必要はない。  ここで指定する型は、完全修飾クラス名でなければならない。  また、配列クラスは、完全修飾クラス名の後ろに[]を付与する事で表現できる。 |
| valueType | | 任意 | 値の型を定義する。  メソッドの引数の型と、argument要素の内容で定義する値の型が代入互換の関係にあるが異なる型である時に、argument要素の内容で定義する値の型を明示するために使用する。  メソッドの引数の型と、argument要素の内容で定義する値の型が等しい場合には、指定する必要はない。  ここで指定する型は、完全修飾クラス名でなければならない。  また、配列クラスは、完全修飾クラス名の後ろに[]を付与する事で表現できる。 |
| nullValue | | 任意 | 引数がnullである事を定義する。  デフォルトは、false。 |
| narrowCast | | 任意 | 数値型のダウンキャストを許可するかどうかを定義する。  デフォルトは、falseで許可しない。 |
| 11 | callback | callflow | override\*  attribute\* | 非同期子フロー呼び出しのコールバックフローを定義する。  コールバックされたフローには、引数として、AsynchCallbackContextが渡る。  子要素のattributeを指定した値は、AsynchCallbackContext#getContext(key)で取得できる。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | コールバックするフロー名を定義する。 |
| 12 | reply | flow  case  default  if  for  while  catch  finally | なし | 非同期子フロー呼び出しの応答の取得を定義する。  子フローの呼び出し結果は、ステップ結果の参照と同様で、step-ref要素で行う。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | ステップ名を定義する。 |
| stepname | | 任意 | 取得する非同期子フロー呼び出しであるcallflow要素のステップ名を定義する。  定義しない場合は、name属性で指定したステップ名がそのままcallflow要素のステップ名になります。 |
| timeout | | 任意 | 応答待ちタイムアウト[ms]を定義する。 |
| cancel | | 任意 | 応答待ちでタイムアウトした場合に非同期呼び出しをキャンセルするかどうかを定義する。  timeout属性を指定していない場合は、無効。 |
| 13 | if | flow  case  default  if  for  while  catch  finally | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | 条件を定義する。  この条件に一致する場合のみ、子要素が実行される。 | |
| 属性名 | | 必須/任意 | 説明 |
| test | | 必須 | 条件を定義する。  この属性の値には、@で囲んだキーとそのキー値に対する条件式を記述する。  キーでは、以下のものが参照できる。  (1)ステップ  ステップの参照では、ステップの対象と結果を参照する事ができる。  ・ステップの対象 : @ステップ名.target@  ・ステップの結果 : @ステップ名.result@ または @ステップ名@  この要素の親要素ステップの対象：@this@  また、ステップの対象及び結果のプロパティを参照する事もできる。  その場合は、上記の表現の後に、プロパティを表現する文字列を指定する。ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。Java Beansのような単純なプロパティに加え、java.util.Mapなどのマッププロパティやjava.util.List、配列などのインデックスプロパティ、またそれらがネストしたプロパティなどもサポートする。  例 : @ステップ名.hoge[0].fuga@  (2)フローの入力  フローの入力を参照する場合には、@input@と指定する。  ステップの参照と同様にフローの入力のプロパティも指定できる。  (3)繰り返し変数  繰り返し変数を参照する場合には、@var(変数名)@と指定する。  ステップの参照と同様に繰り返し変数のプロパティも指定できる。  条件式は、The Apache Jakarta Projectの Commons Jexl(http://jakarta.apache.org/commons/jexl/)を使用する。  また、条件式の結果は必ずbooleanでなければならない。  例 : test='@ステップ名.hoge[0].fuga@ == "HOGE"' |
| nullCheck | | 任意 | キーのプロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのキーはnullとみなされる。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 14 | switch | flow  case  default  if  for  while  catch  finally | case+  default? | フロー中で条件選択を定義する。  各条件は、子要素のcase要素に指定し、その条件に該当する場合の処理は、case要素の子要素に定義する。  また、全てのcase要素の条件に一致しない場合の処理を定義したい場合は、default要素を定義して、その子要素に定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 15 | case | switch | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | switch要素下の1条件を定義する。  この条件に一致する場合のみ、子要素が実行される。 | |
| 属性名 | | 必須/任意 | 説明 |
| test | | 必須 | if要素のtest属性と同じ |
| nullCheck | | 任意 | if要素のnullCheck属性と同じ |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 16 | default | switch | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | switch要素下の最終条件を定義する。全てのcase条件に一致しない場合のみ、子要素が実行される。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | if要素のnullCheck属性と同じ |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 17 | for | flow  case  default  if  for  while  catch  finally | target?  step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | 繰り返し処理を定義する。  繰り返し対象のオブジェクトを子要素のtarget要素で指定する。  繰り返し対象のオブジェクトには、以下のクラスのオブジェクトを指定できる。  ・任意のクラスの配列  ・java.util.Collectionインタフェースを実装したクラス  ・java.util.Enumerationインタフェースを実装したクラス  ・java.sql.ResultSetインタフェースを実装したクラス  ・jp.ossc.nimbus.recset.RecordSetインタフェースを実装したクラス  繰り返し変数名をvar属性で必ず指定する。  繰り返し変数は、var要素で参照する事ができる。    繰り返し対象のオブジェクトを指定しない場合は、end属性で指定した回数分、子要素のステップを実行する。 | |
| 属性名 | | 必須/任意 | 説明 |
| var | | 任意 | 繰り返し変数の変数名を定義する。  ここで定義した変数名で、var要素を使って参照する事ができる。 |
| index | | 任意 | 現在の繰り返し回数の変数名を定義する。  ここで定義した変数名で、var要素を使って参照する事ができる。 |
| begin | | 任意 | 繰り返し開始番号を定義する。  この属性を指定しない場合は、0番目から繰り返す。  数値を直接記述する以外にも、expression要素と同じ式表現で数値を表現する事もできる。 |
| end | | 任意 | 繰り返し終了番号を定義する。  この属性を指定しない場合は、繰り返し対象の長さ分だけ繰り返す。  この属性を指定した場合で、繰り返し対象の長さよりも大きな数値を指定した場合には、繰り返し対象の長さ分だけ繰り返す。  数値を直接記述する以外にも、expression要素と同じ式表現で数値を表現する事もできる。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。この要素の子要素全てに適用される。 |
| 18 | while | flow  case  default  if  for  while  catch  finally | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | 一定条件下での繰り返し処理を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| test | | 必須 | if要素のtest属性と同じ |
| nullCheck | | 任意 | if要素のnullCheck属性と同じ |
| do | | 任意 | 条件判定より先に子要素のステップを実行するかどうかを定義する。  デフォルトは、false。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。この要素の子要素全てに適用される。 |
| 19 | catch | flow  step  callflow | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | 例外処理を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| var | | 任意 | 例外の変数名を定義する。 |
| exception | | 任意 | 捕捉する例外を定義する。  指定しない場合は、java.lang.Exception |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。この要素の子要素全てに適用される。 |
| 20 | finally | flow  step  callflow | step\*  callflow\*  reply\*  switch\*  if\*  for\*  while\*  break?  continue?  return?  throw? | 最終処理を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 21 | throw | case  default  if  for  while  catch  finally | object?  var?  step-ref? | 例外をthrowする事を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| var | | 任意 | throwする例外の変数名を定義する。 |
| 22 | return | flow  case  default  if  for  while  catch  finally | input  object  service-ref  static-invoke  static-field-ref  resource-ref  step-ref  var  expression | 処理の中断を定義する。  定義可能な子要素以外に、内容に文字列を指定する事で、文字列を返す事もできる。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullValue | | 任意 | 戻り値がnullである事を定義する。 |
| 23 | break | case  default  if  for  while  catch  finally | なし | 繰り返し処理を中断する。 | |
| 24 | continue | case  default  if  for  while  catch  finally | なし | 繰り返し処理の開始ステップへ戻る。 | |
| 25 | object | target  result  field  attribute  argument  service-ref  return  throw | constructor?,  field\*  attribute\*  invoke\* | オブジェクトの生成を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| code | | 必須 | 生成するオブジェクトのクラス名を定義する。  クラス名は、完全修飾クラス名で定義する。  また、配列クラスは、完全修飾クラス名の後ろに[]を付与する事で表現できる。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 26 | constructor | object | argument+  static-field-ref  static-invoke | コンストラクタインジェクションを定義する。  引数ありのコンストラクタを使用してインスタンスを生成したい場合に、使用する。  この要素を指定しない場合には、デフォルトコンストラクタでインスタンス生成が行われる。 | |
| 27 | field | step  result  object | input  object  service-ref  resource-ref  step-ref  static-field-ref  static-invoke  var  this  expression | フィールドインジェクション及びフィールド参照を定義する。  object要素、step要素の子要素として定義する場合は、フィールドインジェクションを行う。内容に文字列を指定した場合、その文字列をフィールドの型に変換してインジェクションを行う。  result要素の子要素として定義する場合は、フィールド参照を行う。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | フィールドのフィールド名を定義する。  ここで指定する名前のフィールドは、publicなクラスのpublicなフィールドでなければならない。また、フィールド名は大文字小文字を区別する。 |
| type | | 任意 | フィールドの値の型を定義する。  ここで指定する型は、完全修飾クラス名でなければならない。  但し、配列クラスは、完全修飾クラス名の後ろに[]を付与する事で表現できる。 |
| nullValue | | 任意 | 値がnullである事を定義する。  フィールドインジェクションの場合のみ有効。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 28 | attribute | step  result  callback  object | input  object  service-ref  resource-ref  step-ref  static-field-ref  static-invoke  var  this  expression | プロパティインジェクション及びプロパティ参照を定義する。  object要素、step要素の子要素として定義する場合は、プロパティインジェクションを行います。  result要素の子要素として定義する場合は、プロパティ参照を行います。  callback要素の子要素として定義する場合のみ特殊で、コールバックされるフローの引数であるAsynchCallbackContextのsetContext(key,value)に設定する。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | プロパティ名を定義する。  ここで指定する名前のプロパティは、publicなクラスのpublicなプロパティでなければならない。また、プロパティ名は最初の1文字以外は大文字小文字を区別する。  プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。 |
| type | | 任意 | プロパティ値の型を定義する。  ここで指定する型は、完全修飾クラス名でなければならない。  但し、配列クラスは、完全修飾クラス名の後ろに[]を付与する事で表現できる。 |
| nullValue | | 任意 | 値がnullである事を定義する。  プロパティインジェクションの場合のみ有効。  デフォルトは、false。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 29 | invoke | step  result  object | argument\* | メソッドインジェクション及びメソッド呼び出し結果参照を定義する。  object要素、step要素の子要素として定義する場合は、メソッドインジェクションを行う。  result要素の子要素として定義する場合は、メソッド呼び出し結果参照を行う。 | |
| 属性名 | | 必須/任意 | 説明 |
| name | | 必須 | メソッド名を定義する。  ここで指定する名前のメソッドは、publicなクラスのpublicなメソッドでなければならない。また、メソッド名は大文字小文字を区別する。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 30 | static-invoke | step  target  field  attribute  result  constructor  argument  service-ref  return | argument\* | staticメソッドの呼び出しを定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| code | | 必須 | staticメソッドが宣言されているクラス名を定義する。 |
| name | | 必須 | staticメソッド名を定義する。 |
| journal | | 任意 | ジャーナルを出力するかどうかを定義する。  この要素の子要素全てに適用される。 |
| 31 | static-field-ref | target  field  attribute  result  constructor  argument  service-ref  return | なし | staticフィールド参照を定義する。 | |
| 属性名 | | 必須/任意 | 説明 |
| code | | 必須 | staticフィールドが宣言されているクラス名を定義する。 |
| Name | | 必須 | staticフィールド名を定義する。 |
| 32 | var | target  field  attribute  result  argument  service-ref | なし | 繰り返し変数を参照する。  この要素の内容に、繰り返し変数名を指定する。  また、繰り返し変数のプロパティを参照する事もできる。  その場合は、上記の表現の後に、プロパティを表現する文字列を指定する。  ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | 内容で、プロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのプロパティはnullとみなされる。 |
| 33 | this | field  attribute  result  argument | なし | 現在のステップの対象を参照する。  現在のステップ対象のプロパティを参照する場合は、内容にプロパティ名を指定する。  ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | 内容で、プロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのプロパティはnullとみなされる。 |
| 34 | input | target  field  attribute  result  argument  service-ref | なし | フローの入力を参照する。  フローの入力のプロパティを参照する場合は、内容にプロパティ名を指定する。  ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | 内容で、プロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのプロパティはnullとみなされる。 |
| 35 | service-ref | target  field  attribute  argument  return | input  object  static-invoke  static-field-ref  step-ref  var  expression | サービスを参照する。  内容にサービス名を指定する。  サービス名は、マネージャ名#サービス名の形式で指定する。 | |
| 36 | resource-ref | target  field  attribute  argument  return | なし | リソースを参照する。  参照したいリソースを定義したresource要素のname属性の値を、この要素の内容に指定する。 | |
| 属性名 | | 必須/任意 | 説明 |
| raw | | 任意 | resource-ref要素で参照するTransactionResourceをそのまま参照するかどうかを定義する。  trueを指定した場合は、TransactionResourceそのものを参照する。  デフォルトはfalseで、TransactionResource#getObject()で取得されるオブジェクトを参照する。 |
| 37 | step-ref | target  field  attribute  result  argument  service-ref  return  throw | なし | ステップを参照する。  ステップの参照では、ステップの対象と結果を参照する事ができる。  ・ステップの対象 : ステップ名.target  ・ステップの結果 : ステップ名.result または ステップ名  また、ステップの対象及び結果のプロパティを参照する事もできる。  その場合は、上記の表現の後に、プロパティを表現する文字列を指定する。  ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | 内容で、プロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのプロパティはnullとみなされる。 |
| 38 | expression | target  field  attribute  result  argument  service-ref  return | なし | 式を表す。  この要素の内容には、式を記述する事ができ、式中の可変パラメータとして@で囲んだキーを指定する事ができる。  キーでは、以下のものが参照できる。  (1)ステップ  ステップの参照では、ステップの対象と結果を参照する事ができる。  ・ステップの対象 : @ステップ名.target@  ・ステップの結果 : @ステップ名.result@ または @ステップ名@  ・この要素の親要素ステップの対象：@this@  また、ステップの対象及び結果のプロパティを参照する事もできる。  その場合は、上記の表現の後に、プロパティを表現する文字列を指定する。  ここで言う、プロパティの概念は、Java Beansのプロパティの概念より広く、jp.ossc.nimbus.beans.PropertyFactoryの規約に従う。  (2)フローの入力  フローの入力を参照する場合には、@input@と指定する。  ステップの参照と同様にフローの入力のプロパティも指定できる。  (3)繰り返し変数  繰り返し変数を参照する場合には、@var(変数名)@と指定する。  ステップの参照と同様に繰り返し変数のプロパティも指定できる。  式は、The Apache Jakarta Projectの Commons Jexl(http://jakarta.apache.org/commons/jexl/)を使用する。  例 : @ステップ名.hoge[0].fuga@ \* 100 | |
| 属性名 | | 必須/任意 | 説明 |
| nullCheck | | 任意 | 内容で、プロパティ表現を指定した場合に、ネストされたプロパティがnullかどうかをチェックするかどうかを設定する。  trueを設定した場合は、ネストされたプロパティがnullの場合、jp.ossc.nimbus.beans.NoSuchPropertyExceptionが発生する。  デフォルトは、falseで、ネストされたプロパティがnullの場合、そのプロパティはnullとみなされる。 |
| 39 | interpreter | step | なし | インタープリタを使って処理を表す。  この要素の内容には、jp.ossc.nimbus.service.interpreter.Interpreterが解釈するソースコードを記述する。  ソースコード内では、以下の変数が暗黙的に使用可能。  (1)ステップ  ステップ名で参照する。  ・ステップの対象 : ステップ名.target  ・ステップの結果 : ステップ名.result  ・この要素の親要素ステップの対象：target  但し、指定したステップを通っていない場合は、ステップ名で参照した値はnullになるので、注意が必要。  (2)フローの入力  フローの入力を参照する場合には、inputと指定する。  (3)繰り返し変数  繰り返し変数を参照する場合には、var.get("変数名")と指定する。  但し、変数が宣言されていない場合は、varがnullになるので、注意が必要。  (4)リソース  リソースを参照する場合には、resource.getResource("リソース名")と指定する。  ソースコード内で戻り値を返すと、このステップのresultになる。  但し、result要素を記述した場合は、そちらが優先される。  BeanFlowInvokerFactoryサービスに使用するjp.ossc.nimbus.service.interpreter.Interpreterを設定しておく必要がある。 | |

上記の表に該当する、DTDファイルが以下のURLからダウンロード可能である。

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/beancontrol/beanflow_1_0.dtd>

以下に、業務フロー定義XMLファイルの例を示す。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE flows PUBLIC

"-//Nimbus//DTD Nimbus Bean Flow 1.0//JA"

"http://nimbus.sourceforge.jp/dtd/beanflow\_1\_0.dtd">

<flows>

<flow name=”BeanFlow1”>

<alias name=”flow1”/>

<resource name=”ConnectionResource”

service=”Nimbus#ConnectionResourceFactory” />

<step name=”step1”>

<target><object code=” sample.LogicBean1”></target>

<attribute name=”Connection”>

<resource-ref> ConnectionResource</resource-ref>

</attribute>

<attribute name=”TableName”> USER\_MST</attribute>

<invoke name=”load”/>

</step>

<step name=”step2”>

<target><step-ref>step1</step-ref></target>

<attribute name=”Connection”>

<resource-ref> ConnectionResource</resource-ref>

</attribute>

<invoke name=”update”/>

<result><attribute name=”Result”/></result>

</step>

</flow>

</flows>

## BeanFlowInvokerFactory

### jp.ossc.nimbus.service.beancontrol.DefaultBeanFlowInvokerFactoryService

XMLファイルで定義した業務フローを実行するBeanFlowInvokerを生成するBeanFlowInvokerFactoryのデフォルト実装クラスである。

業務フロー

定義XML

createFlow(業務フロー名)

**DefaultBeanFlowInvokerFactoryService**

BeanFlowInvoker

TransactionManagerFactory

invokeFlow(input)

InterceptorChainFactory

Journal

Interpreter

ThreadContextService

QueueHandlerContainer

XMLファイルの解釈は、BeanFlowInvokerの実装クラスに依存する。デフォルトでは、BeanFlowInvokerAccessImplである。

このサービスには、業務フローの実行に必要な様々なサービスが注入されている。

トランザクション制御を行うTransactionManagerを取得するTransactionManagerFactory、業務フローの実行に横断的な付加処理を与えるInterceptorChainを取得するInterceptorChainFactory、実行記録を残すJournal、インタプリタ実行を行うInterpreter、スレッド単位でデータを格納しコンテキスト情報として利用するThreadContextService、非同期処理のスレッドプールを持つQueueHandlerContainerなどがある。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ResourceManagerFactoryServiceName | ServiceName | jp.ossc.nimbus.service.beancontrol.resource.ResourceManagerFactoryサービスのサービス名を設定する。  設定しない場合は、デフォルトで、jp.ossc.nimbus.service.beancontrol.resource.ResourceManagerFactoryServiceが生成される。 |
| 2 | ThreadContextServiceName | ServiceName | jp.ossc.nimbus.service.context.Contextサービスのサービス名を設定する。 |
| 3 | LogServiceName | ServiceName | jp.ossc.nimbus.service.log.Loggerサービスのサービス名を設定する。 |
| 4 | JournalServiceName | ServiceName | jp.ossc.nimbus.service.journal.Journalサービスのサービス名を設定する。 |
| 5 | EditorFinderServiceName | ServiceName | jp.ossc.nimbus.service.journal.editorfinder.EditorFinderサービスのサービス名を設定する。 |
| 6 | InterpreterServiceName | ServiceName | jp.ossc.nimbus.service.interpreter.Interpreterサービスのサービス名を設定する。 |
| 7 | ManageExecBeanFlow | boolean | 実行中のBeanFlowを管理するかどうかを設定する。  デフォルトは、true。 |
| 8 | BeanFlowInvokerAccessClass | Class | BeanFlowInvokerAccessインタフェースの実装クラスを設定する。  デフォルトは、BeanFlowInvokerAccessImpl。 |
| 9 | Validate | boolean | フロー定義XMLを、DTDで検証するかどうかを設定する。  デフォルトは、false。 |
| 10 | InterceptorChainFactoryServiceName | ServiceName | jp.ossc.nimbus.service.aop.InterceptorChainFactoryサービスのサービス名を設定する。 |
| 11 | TransactionManagerFactoryServiceName | ServiceName | jp.ossc.nimbus.service.transaction.TransactionManagerFactoryサービスのサービス名を設定する。  設定しない場合は、jp.ossc.nimbus.service.transaction.JndiTransactionManagerFactoryServiceが適用される。 |
| 12 | AsynchInvokeQueueHandlerContainerServiceName | ServiceName | jp.ossc.nimbus.service.queue.QueueHandlerContainerサービスのサービス名を設定する。  設定しない場合は、非同期実行をサポートしない。 |
| 13 | DirPaths | String[] | 業務フロー定義XMLの存在するディレクトリを設定する。 |
| 14 | Paths | String[] | 業務フロー定義XMLのパスを設定する。 |

### jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerFactoryGroupService

BeanFlowInvokerFactoryをグルーピングするBeanFlowInvokerFactoryインタフェースの実装クラスである。

**BeanFlowInvokerFactoryGroupService**

BeanFlowInvoker

createFlow(業務フロー名)

createFlow(業務フロー名)

BeanFlowInvokerFactory

BeanFlowInvokerFactory

BeanFlowInvokerFactoryを内部に複数持ち、createFlow(String)が呼び出されると、内部のBeanFlowInvokerFactoryから順次、指定された業務フローを探し、その業務フローを持っているBeanFlowInvokerFactoryを特定して、createFlow(String)を実行する。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | BeanFlowInvokerFactoryServiceNames | ServiceName[] | グルーピングするBeanFlowInvokerFactoryのサービス名を設定する。 |

### jp.ossc.nimbus.service.beancontrol.ClientBeanFlowInvokerFactoryService

業務フローをリモート実行するBeanFlowInvokerFactoryインタフェースの実装クラスである。

クライアント

createFlow(業務フロー名)

**ClientBeanFlowInvokerFactoryService**

invokeFlow(input)

BeanFlowInvoker

ClusterService

BeanFlowInvokerServer

（リモートオブジェクト）

BeanFlowInvokerServer

（リモートオブジェクト）

サーバ

サーバ

ClusterService

BeanFlowInvokerServer

（サーバオブジェクト）

BeanFlowInvokerServer

（リモートオブジェクト）

createFlow(業務フロー名)

BeanFlowInvoker

BeanFlowInvokerFactory

業務フローを実行するサーバ側では、BeanFlowInvokerServerインタフェースを実装したサービスが、ClusterServiceに自分自身をRMIで呼び出すリモートオブジェクトを登録し、リモートからの呼び出しに備える。一方、クライアント側では、ClusterServiceを使って、クラスタに参加しているリモートのBeanFlowInvokerServerを認識する。createFlow(業務フロー名)を呼び出されると、認識しているBeanFlowInvokerServerの中から、呼び出し対象の業務フローを持ち、リソース利用量が少ないBeanFlowInvokerServerを選択し、サーバ側にもBeanFlowInvokerを生成する。invokeFlow(input)が呼び出されると、サーバ側のBeanFlowInvokerを呼び出し、リモート上で業務フローの実行を行う。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClusterServiceName | ServiceName | jp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 2 | ContextServiceName | ServiceName | リモートの業務フローに引き渡すコンテキスト情報を持つjp.ossc.nimbus.service.context.Contextのサービス名を設定する。 |
| 3 | ContextKeys | String[] | リモートの業務フローに引き渡すコンテキスト情報のキーを設定する。 |
| 4 | AsynchInvokeQueueHandlerContainerServiceName | ServiceName | 業務フローの非同期実行を、クライアント側で担保するためのQueueHandlerContainerサービスのサービス名を設定する。 |

## BeanFlowInvokerServer

### jp.ossc.nimbus.service.beancontrol.BeanFlowInvokerServerService

RMIでリモートから業務フローを実行するサーバサイドの呼び出し口となるBeanFlowInvokerServerのデフォルト実装クラスである。

クライアント

createFlow(業務フロー名)

ClientBeanFlowInvokerFactoryService

invokeFlow(input)

BeanFlowInvoker

ClusterService

BeanFlowInvokerServer

（リモートオブジェクト）

BeanFlowInvokerServer

（リモートオブジェクト）

サーバ

サーバ

**BeanFlowInvokerServerService**

ClusterService

BeanFlowInvokerServer

（サーバオブジェクト）

BeanFlowInvokerServer

（リモートオブジェクト）

createFlow(業務フロー名)

ResourceUsage

BeanFlowInvokerFactory

BeanFlowInvoker

クライアント側からisAlive()で生存状態を問い合わせられると、呼びさし先であるBeanFlowInvokerFactoryサービスの状態をチェックして、サービスが開始されていれば、trueを返し、生存を応答する。また、getResourceUsage()で、リソースの使用量を問い合わせられると、ResourceUsageサービスに委譲して、結果を返す。但し、ResourceUsageサービスが設定されていない場合は、実行中の業務フローの数を応答する。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | BeanFlowInvokerFactoryServiceName | ServiceName | BeanFlowInvokerFactoryサービスのサービス名を設定する。 |
| 2 | InterceptorChainFactoryServiceName | ServiceName | 業務フローの実行に挟み込むjp.ossc.nimbus.service.aop.InterceptorChainを生成するjp.ossc.nimbus.service.aop.InterceptorChainFactoryサービスのサービス名を設定する。 |
| 3 | ContextServiceName | ServiceName | クライアントから業務フローに引き渡すコンテキスト情報を乗せるjp.ossc.nimbus.service.context.Contextのサービス名を設定する。 |
| 4 | ResourceUsageServiceName | ServiceName | jp.ossc.nimbus.service.performance.ResourceUsageサービスのサービス名を設定する。 |
| 5 | RMIPort | int | てRMI呼び出しをする時のポート番号を設定する。  デフォルトは、0で任意のポートが使用される。 |
| 6 | ClusterServiceName | ServiceName | jp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 7 | RMIClientSocketFactoryServiceName | ServiceName | java.rmi.server.RMIClientSocketFactoryサービスのサービス名を設定する。 |
| 8 | RMIServerSocketFactoryServiceName | ServiceName | java.rmi.server.RMIServerSocketFactoryサービスのサービス名を設定する。 |

## Compiler

業務フロー定義XMLファイルをDefaultBeanFlowInvokerFactoryServiceで読み込んだ場合に、正常に読み込めるかを事前に検証するコンパイラを提供する。

コマンド使用方法：

java jp.ossc.nimbus.service.beancontrol.Compiler [options] [beanflow files]

[options]

[-v]

実行の詳細を表示します。

[-class]

jp.ossc.nimbus.service.beancontrol.interfaces.BeanFlowInvokerの実装クラス。

[-help]

ヘルプを表示します。

[beanflow files]

コンパイルする業務フロー定義ファイル、またはディレクトリを指定します。

スペース区切りで複数指定可能です。

使用例 :

java -classpath classes;lib/nimbus.jar jp.ossc.nimbus.service.beancontrol.Compiler beanflow-def.xml