IBM InfoSphere Information Server
バージョン 11 リリース 3

接続ガイド: IBM WebSphere iLog JRules 編

SC43-0989-00
(英文原典：SC19-4338-00)

IBM InfoSphere Information Server
バージョン 11 リリース 3

接続ガイド: IBM WebSphere iLog JRules 編

SC43-0989-00
(英文原典：SC19-4338-00)

お願い

本書および本書で紹介する製品をご使用になる前に、 49 ページの『特記事項および商標』 に記載されている情報をお読みください。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC19-4338-00
IBM InfoSphere Information Server Version 11 Release 3
Connectivity Guide for IBM WebSphere
ILOG JRules

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

IBM WebSphere ILOG JRules

IBM® WebSphere® ILOG® JRules Connector を使用すると、ジョブの内部から ILOG JRules ルール・セットを呼び出すことができます。

注: IBM WebSphere ILOG JRules Business Rules Management System (BRMS) ファミリー内の製品は、IBM Operational Decision Manager としてブランドが変更されました。コネクターがアクセスするコンポーネントは、現在は IBM Decision Server と称しています。 InfoSphere® Information Server クライアントおよび資料は、この変更により更新されていません。

概説

ILOG JRules Business Rules Management System (BRMS) によって、カスタマーは、複雑なビジネス・ルールをアプリケーションの外に置くことができます。ILOG JRules ステージを使用して、ジョブのコンテキストの内部で複雑なビジネス・ルールを呼び出すことができます。

ステージによって、DataStage パラレル・ジョブで ILOG JRules エンジンにアクセスできます。JRules エンジンを使用して、ジョブ内のレコードに対して複雑なデータ変換を実行できます。

コネクターがコア・エンジン・モードで JRules にアクセスするように構成されている場合、コネクターは、IBM InfoSphere Information Server エンジン・ホストのルール・セット・アーカイブ・ファイルにあるルール・セットを呼び出します。コネクターがローカルに管理された (J2SE) モードまたはリモート側で管理された (J2EE) モードで JRules にアクセスするように構成されている場合、コネクターは、ルール・セット・リポジトリーに保持されているルール・セットを呼び出します。

ILOG JRules Connector は、デザイン時コンポーネントとランタイム・コンポーネントから構成されます。

デザイン時コンポーネントは、ユーザーからの入力を収集するステージ・エディターから構成されます。これはまた、指定されたルール・セットのパラメーター定義に基づいて、ステージ・プロパティーの構成およびコネクターのリンク・スキーマ定義の構成に使用される、構成ウィザードから構成されます。

コネクターのランタイム・コンポーネントは、DataStage ジョブのコンテキストの中でルール・セットの呼び出しを行います。ILOG JRules は、Java オブジェクトの形の入力データ、および ILOG JRules によって返される出力データ (これもまた Java オブジェクトから構成される) を受け入れます。ランタイム・コンポーネントの基本的な機能は、アップストリームのステージから送られた入力 DataStage レコードを読み取り、それを Java オブジェクトに変換することです。その後、ランタイム・コンポーネントはインスタンス化された Java オブジェクトを使用してルール・セットを呼び出し、ルール・セットによって返された Java オブジェクトを DataStage レコードに変換します。そして、これらのレコードは、ダウンストリームのステージに送られます。したがって、ILOG JRules ステージは要求/応答モードでのみ作動

でき、これはまた、ジョブの最初のステージでも最後のステージでもなく、中間のステージとして配置する必要があることも意味します。これは、例えば Transformer ステージの使用に類似しています。

JRules エンジン・モード

JRules エンジン・モードは、コネクターが、デザイン時にルール・セットを検出してステージを構成するために使用し、実行時に JRules セッションを確立してルール・セットを呼び出すために使用するインターフェースのタイプを指定します。JRules エンジン・モードとして、「コア・エンジン」、「J2SE RES XU」、または「J2EE RES XU」を選択できます。

コア・エンジン

このエンジン・モードでは、コネクターは、直接、エンジン JAR ファイルを通して JRules エンジンにアクセスします。コネクターは、実行されるルール・セットの定義を含む物理的なルール・セット・アーカイブ・ファイルへの JRules エンジンをポイントします。

このモードを使用するには、ルール・セット・アーカイブとしてルール・セットをエクスポートする必要があります。ルール・セットは、JAR ファイルとしてエクスポートされ、この JAR ファイルは、IBM InfoSphere Information Server エンジンが実行中のマシンでコピーされる必要があります。この JAR ファイルのパスは、「**ルール・セットの場所**」ステージ・プロパティーで指定されなければなりません。

J2SE RES XU モード

このエンジン・モードでは、JRules エンジンは、コネクターと同じ Java プロセスの中でローカルに管理される Rule Execution Server (RES) として実行されます。ルール・セットを呼び出すために、コネクターは、ルール・セット・リポジトリーの中のルール・セット定義への JRules エンジンをポイントします。

ルール・セット・リポジトリーは、ファイル・ベース・リポジトリーまたはデータベース・ベース・リポジトリーのどちらかにできます。リポジトリーがファイル・ベースである場合、リポジトリーは、IBM InfoSphere Information Server エンジンが実行中の同じマシン上になければなりません。ファイル・ベース・リポジトリーを使用するかデータベース・ベース・リポジトリーを使用するかについての構成は、`ra.xml` JRules 構成ファイルで指定されます。この構成ファイルの詳細は、JRules の資料を参照してください。Java 2 Standard Edition (J2SE) RES XU を使用している場合、「**ルール・セットの場所**」ステージ・プロパティーは、リポジトリー内のルール・セットのパスを指定します。

J2EE RES XU モード

このエンジン・モードでは、JRules エンジンは、Java 2 Enterprise Edition (J2EE) サーバー上のリモート側で管理される RES として実行されます。ルール・セットを呼び出すために、コネクターは、ルール・セット・リポジトリーの中のルール・セット定義への JRules エンジンをポイントします。ルール・セット・リポジトリーへの接続は、RES が J2EE サーバーにデプロイされるときに構成されます。ジョブの実行中に、コネクターはリモート EJB3 クライアントとして機能し、Rule Execution Server がデプロイされた J2EE サーバーにデプロイされる必要がある EJB3 ルール・セッション・コ

ンポーネントへの接続を確立します。 J2EE RES XU を使用している場合、「ルール・セットの場所」ステージ・プロパティーは、リポジトリー内のルール・セットのパスを指定します。

注: バッチ・タイプの処理の場合、「コア・エンジン」および「J2SE RES XU」のモードでは、コネクターは、コネクターと同じ Java 仮想マシンで動作する JRules エンジンで Java オブジェクトを交換するため、一般に、「J2EE RES XU」モードよりも高いパフォーマンスを提供します。「J2EE RES XU」エンジン・モードでは、コネクターは、リモート EJB インターフェースを通して J2EE サーバー上で実行される JRules エンジンと、交換には処理の一環としてかなりのオーバーヘッドを招くおそれのあるシリアライズとデシリアライズが必要な Java オブジェクトにアクセスします。

バッチ・モードおよびキー・モードの処理

JRules Connector ステージの各入力リンクは、そのステージ用に指定されたルール・セットの IN または IN_OUT パラメーターのどちらかに対応します。コネクターは、入力リンクで取得されたレコードを Java 値に変換し、対応する IN および IN_OUT ルール・セット・パラメーターの値として JRules に渡します。実行時機能に関係がある主要な概念は、ルール実行サイクルの機能です。各ルール実行サイクルで、コネクターは、入力リンクから 1 つ以上のレコードを読み取り、それを Java オブジェクトに変換して JRules に送信します。

各ルール実行サイクルの中で入力リンクから取得されるレコードの数は、バッチ・モードおよびキー・モードで制御されます。レコード処理に対して、バッチ・モードまたはキー・モードのどちらかを構成できます。最初の入力リンクからのレコードの処理にコネクターが使用する戦略は、残りの入力リンクからのレコードの処理に使用する戦略と異なります。コネクターは、内部で、入力リンクをマスター・リンクとセカンダリー・リンクに分類します。最初の入力リンクはマスター・リンクで、残りの入力リンクはセカンダリー・リンクです。

このトピックの中の情報は、次の条件を満たすシナリオに適用されます。

- バッファー・リンクとしてマークを付けられた入力リンクはない。バッファー・リンクでのバッチ・モードおよびキー・モードの使用の情報は、『バッファー・リンク』トピックを参照してください。
- ルール・セット・パラメーターの XOM タイプは Java XOM である。動的 XOM でのバッチ・モードおよびキー・モードの使用の情報は、『*実行オブジェクト・モデル・タイプ*』トピックを参照してください。

レコード処理の戦略

コネクターは、ルール・セットを呼び出す前に、ルール・セットのすべての IN および IN_OUT パラメーターの値を明示的に設定します。場合によっては、パラメーターに設定される値が NULL 値であることもあります。しかし、コネクターは、ルール・セットのすべての IN および IN_OUT パラメーターの値を事前に設定せずにルール・セットを呼び出すことはありません。

マスター・リンクで使用可能なレコードがない場合、コネクターは、どのセカンダリー・リンクでも使用可能なレコードがないことと、ジョブが正常に完了したことを確認します。しかし、あるセカンダリー・リンクで使用可能なレコードが余分に

ある場合、そのセカンダリー・リンクに、「**残存レコード・エラー**」オプションが選択されたリジェクト・リンクが定義されていれば、コネクターはそれらの余分なレコードをリジェクトします。そうでない場合は、コネクターはエラーをログに記録し、ジョブは失敗します。

セカンダリー・リンクでレコードが使用可能でない場合、コネクターが実行するアクションは、そのリンクに関連付けられたルール・セット・パラメーターが Java 配列タイプかどうかによって決まります。Java 配列タイプである場合は、コネクターは、空の配列を作成し、それをルール・セット・パラメーター値として設定します。 Java 配列タイプでない場合は、コネクターは状態をエラーとして扱い、現行のルール実行サイクルですべての入力リンク (マスター・リンクを含めて) で受け取ったレコードをリジェクトしますが、これは、これらの入力リンクに、「**残存レコード・エラー**」オプションが選択されたリジェクト・リンクが定義されている場合に限ります。そうでない場合は、コネクターはエラーをログに記録し、ジョブは失敗します。レコード処理戦略に適用される追加のルールは、ステージに構成された実行モードによって決まります。

バッチ・モード

バッチ・モードを使用可能にするには、「**バッチ処理を使用可能にする**」ステージ・プロパティーを「**はい**」に設定し、「**キーの処理を使用可能にする**」ステージ・プロパティーを「**いいえ**」に設定する必要があります。「**バッチ・サイズ**」プロパティーに、負でない整数値 *n* を指定する必要もあります。デフォルト値は 1 です。バッチ・モードの場合、各ルール実行サイクルで、コネクターは、入力リンクのそれぞれからすべての使用可能なレコードを読み取りますが、リンクあたり *n* レコードは超えません。ここで、*n* はステージに指定されたバッチ・サイズです。バッチ・サイズが 0 は、無制限を意味します。

コネクターはすべての入力リンクで必要な数のレコードを取り出すと、それらのレコードを Java オブジェクトに変換します。コネクターは、Java オブジェクトを後でルール・セット・パラメーター値として設定し、ルール・セットを呼び出します。ルール・セットの実行が成功すると、コネクターは、OUT および IN_OUT ルール・セット・パラメーター値に基づいて、出力リンク上にレコードを生成し、次のルール・セット実行処理を繰り返します。ルール・セットの実行が失敗すると、コネクターは、エラーをログに記録し、ジョブは失敗します。

それぞれのリンクに Integer タイプの 1 列のみが含まれる 2 つの入力リンクを持つ ILOG JRules ステージのジョブの例を考えます。入力リンクに受け取られたデータが次に指定されたとおりであるとします (右方のデータが最初に到着する)。

リンク 1

19 17 16 15 14 12 11 10

リンク 2

20 18 17 15 12 10 8

バッチ・モードを使用可能にし、バッチ・サイズが 2 であるとします。このデータには、4 つのルール実行サイクルがあります。各ルール実行サイクルに送られるデータは次のとおりです。

ルール実行サイクル 1

リンク 1: 11 10

リンク 2: 10 8

ルール実行サイクル 2

リンク 1: 14 12

リンク 2: 15 12

ルール実行サイクル 3

リンク 1: 16 15

リンク 2: 18 17

ルール実行サイクル 4

リンク 1:19 17

リンク 2: 20

キー・モード

キー・モードを使用可能にするには、「**キーの処理を使用可能にする**」ステージ・プロパティーを「**はい**」に設定する必要があります。「**キー列[*n*]**」プロパティーを使用して、入力リンク列がキー列として機能するように選択する必要があります。ここで、*n* は正整数で、キー列の索引を表します。最初に選択されたキー列の索引は 1 から始まり、列が追加されるたびに増分されます。キー列として使用される各キー列の名前は、対応する「**キー列[*n*]**」プロパティーの下の「**列名**」ステージ・プロパティーで指定する必要があります。少なくとも 1 つのキー列を指定する必要があります。これは、少なくとも 「**キー列[1]**」プロパティーがなければならず、その「**列名**」の子プロパティー値が設定されなければならないことを意味します。「**キー列[2]**」を追加するには、「**キー列[1]**」を右クリックして、「**プロパティー値の追加**」を選択します。さらに「**キー列[*n*]**」値を追加するには、この処理を繰り返します。特定の「**キー列[*n*]**」の「**列名**」ステージ・プロパティーを設定するには、「**列名**」の右側に表示されている「**選択**」ボタンをクリックします。選択のために提供される列は、すべての入力リンクに存在する列です。

選択されたそれぞれのキー列について、個々のキー列の値に関して、入力リンクのレコードのソート順を指定する必要もあります。これは、対応する「**キー列[*n*]**」の下の「**ソート順**」オプションで指定します。サポートされる値は、「**昇順**」および「**降順**」です。デフォルト値は「**昇順**」です。コネクターが、2 つのキー列の値を比較して相互の位置を決定する場合、これらの列の値から作成された Java 値を比較します。文字値を含むキー列 (例えば、VarChar 列) の場合、値の比較は大/小文字を区別して行われます。コネクターが NULL 値と非 NULL 値を比較する場合、NULL 値は非 NULL 値よりも小さいとされます。 NULL 可能のキー列で、ソート順が昇順の場合、そのキー列が NULL 値のレコードは、同じキー列が非 NULL 値のレコードより前に現れます。

キー・モードで実行している場合、コネクターは、ステージに指定されたキー列名とソート順に基づいて、実行時に、入力リンク上のレコードのソートを自動的に強制します。また、コネクター・ステージが、複数のノードで、キー・モードで並行して実行するように構成されていて、任意の入力リンクの「**パーティション**」タブの「**パーティション・タイプ**」フィールドがデフォルトの「**(自動)**」値に設定されている場合、コネクターは、その入力リンク上のレコードのハッシュ・パーティションを強制し、ステージで指定されたキー列をハッシュ・キーが使用することを強制します。これによって、一致するキー列値を持つレコードが、確実に同じステージ・インスタンス (同じノードで実行) によって処理されます。

「**バッチ・サイズ**」プロパティーで、ステージに指定されたバッチ・サイズは、キー・モードで重要な役割を果たします。「**キーの処理を使用可能にする**」が「**はい**」に設定されているときに「**バッチ処理を使用可能にする**」プロパティーが「**いいえ**」に設定されると、コネクターは、キー・モードで実行され、バッチ・サイズは 0 (無制限) とみなされます。コネクターがこのモードで入力リンクからレコードを読み取る場合、バッチ・サイズに加えて、レコードに示されたキー列の値も考慮します。それぞれのルール・セット実行で、コネクターは、マスター・リンクから最大 *n* レコードを読み取ります。ここで、*n* は、ステージに指定されたバッチ・サイズを表します。バッチ・サイズ *n* が 0 でなければ、それらのレコードの中のキー値にかかわらず、コネクターは、リンクから最大 *n* レコードを読み取ります。しかし、バッチ・サイズが 0 (無制限) であれば、コネクターは、キー列値が一致するマスター・リンクから、すべての使用可能なレコードを読み取ります。

マスター・リンクからレコードを読み取った後、コネクターは、残っているセカンダリー・リンクから、その同じルール実行サイクルのマスター・リンクで取り出されたいずれかのレコードのキー列値と一致するキー列値のレコードの読み取りを進めます。コネクターがセカンダリー・リンクから読み取るレコードの数は、そのリンクのルール・セット・パラメーターが Java 配列タイプに基づくかどうかによります。ルール・セット・パラメーターが配列タイプに基づく場合、コネクターは、使用可能なすべての一致するレコードを読み取り、それを使用してルール・セット・パラメーターの値を準備します。ルール・セット・パラメーターが配列タイプに基づかない場合、コネクターは、リンクから、一致するレコードを 1 つだけ読み取ります。その後、コネクターはルール・セットを呼び出します。

その後、コネクターは、配列タイプに基づいていないすべてのパラメーターについて、対応するセカンダリー入力リンク上に一致する別のレコードが存在するかどうか検査します。このようなレコードが存在する場合、コネクターは、それをルール・セット・パラメーター値に使用して、再度ルール・セットを呼び出します。再度ルール・セットを呼び出すときに、コネクターは、現在のルール・セット実行で一致する新規のレコードを何も読み取らなかった各ルール・セット・パラメーターについて、前のルール・セット実行からの値を再利用します。すべてのルール・セット・パラメーターについてそれ以上新規の一致するレコードが見つからなくなるまで、コネクターはこの処理を繰り返します。その後、次のルール・セット実行に備えて、マスター・リンクからの新規レコードの読み取りを進めます。

それぞれのリンクに 2 つの列が含まれる 2 つの入力リンクを持つ ILOG JRules ステージのジョブの例を考えます。リンク 1 で、1 番目の列は Integer タイプで、2 番目の列は VarChar(5) タイプです。リンク 2 で、1 番目の列は Integer タイプ

で、2 番目の列は Double タイプです。入力リンクに受け取られたデータが次のとおりであるとします (右方のデータが最初に到着する)。

リンク 1

(16, Sam) (16, Sam) (16, Sam) (12, John) (12, John) (10, Rick) (10, Rick) (10, Rick)

リンク 2

(16, 32.4) (12, 32.22) (12, 24.22) (12, 53.33) (12, 23.233) (10, 32.55) (10, 25.77)

キー・モードを使用可能にし、バッチ・サイズを 0 にするとします。1 番目の、タイプが Integer の列は、キー列であり、キーのソート順は「**昇順**」です。入力リンクに関連付けられたルール・セット・パラメーターが Java 配列タイプであるとします。そして、このデータには、3 つのルール実行サイクルがあります。各ルール実行サイクルに送られるデータは次のとおりです。

ルール実行サイクル 1

リンク 1: (10, Rick) (10, Rick) (10, Rick)

リンク 2: (10, 32.55) (10, 25.77)

ルール実行サイクル 2

リンク 1: (12, John) (12, John)

リンク 2: (12, 32.22) (12, 24.22) (12, 53.33) (12, 23.233)

ルール実行サイクル 3

リンク 1: (16,Sam) (16,Sam) (16,Sam)

リンク 2: (16, 32.4)

コネクターが、いくつかのセカンダリー・リンクで、マスター・レコードの中のどのキー列値にも一致しないキー列値を持つレコードがあることを検出し、かつ、一致するキー列値を持つレコードが、同じルール・セット実行の同じリンクで以前に取得されていなければ、コネクターは、そのリンクのルール・セット・パラメーター値に空の配列を指定します。ルール・セット・パラメーターが Java 配列タイプであり、かつ、列キーに指定されたソート順に基づいて、検出されたレコード内のキー列値が、コネクターがまだマスター・リンクで取得するレコード中のキー列値と一致する可能性があるとコネクターが判断した場合に限り、空の配列が指定されます。

そうでない場合は、コネクターは、キーの不一致のエラーを引き起こすレコードはリジェクトしようとします。コネクターは、現在のセカンダリー・リンクで検出したレコードのキー列値と、現在のルール・セット実行のマスター・リンクで既に収集したレコードのキー列値とを比較します。この比較を行うときに、コネクターは、キー列に指定されたソート順を考慮します。現在のセカンダリー・リンク上の次のレコードが、マスター・リンク上で既に取得されたいずれかのレコードのキー列値と一致するキー列値を持つ可能性がある場合、コネクターは、現在のセカンダ

リー・リンクで読み取ったレコードをリジェクトします。そうでない場合、コネクターは、マスター入力リンクからのレコードも含めて、残りの入力リンクで以前に取得したレコードをリジェクトします。

ILOG JRules ステージに 3 つの入力リンクが含まれるジョブの例を考えます。 3 つのリンク上のキー列が Integer タイプの C1 であるとします。3 つのリンク上の列 C1 の値が次のとおりであるとします (右方のデータが最初に到着する)。

リンク 1 (マスター): 5 5 5

リンク 2: 5 5

リンク 3: 6

バッチ・サイズを 0 に設定するとします。

さまざまな操作モードでのコネクターの動作は以下のとおりです。

1. すべての入力リンクのルール・セット・パラメーターは、配列タイプであり、列 C1 のソート順は、「**昇順**」です。コネクターはルール・セットを呼び出し、第 1 のルール・セット・パラメーターに対して C1 の値が 5 の 3 レコード、第 2 のルール・セット・パラメーターに対して C1 の値が 5 の 2 レコード、第 3 のルール・セット・パラメーターに対して 0 レコードを渡します。コネクターは、列 C1 のソート順に基づいて、マスター・リンク上の次のレコードは値が 6 の C1 を持っていて、現在のセカンダリー・リンク・レコードで一致が存在し、そのために、リンク 3 上のレコードはリジェクトされない可能性がまだあると判別します。
2. リンク 3 のルール・セット・パラメーターは Java 配列タイプでなく、ソート順は「**昇順**」です。この場合、値が 6 の C1 を含むレコードの後に第 3 の入力リンクに次に到着する可能性のあるレコードは、値が 6 以上の C1 のみを持つことができ、そのいずれも値が 5 の C1 と一致する可能性がないため、コネクターは、第 1 および第 2 の入力リンクで既に取得された、値が 5 の C1 を持つレコードをリジェクトします。
3. リンク 3 のルール・セット・パラメーターは Java 配列タイプでなく、ソート順は「**降順**」です。この場合、このリンク上の次のレコードが、値が 5 の C1 を含み、そのため、最初の 2 つの入力リンクで取得したレコードと一致する可能性があるため、コネクターは、第 3 入力リンクで取得した値が 6 の C1 を含むレコードをリジェクトします。

コネクターは、キーの不一致のためにレコードをリジェクトした後、レコードをリジェクトした第 1 入力リンクからレコードを読み取ることにより、現在のルール・セット実行のルール・セット・パラメーター値の準備を継続しようと試みます。これは、マスター・リンクまたは現在のセカンダリー・リンクです。マスター・リンクの場合、コネクターは、ルール・セット・パラメーター値の準備をゼロから開始します。現在のセカンダリー・リンクの場合、コネクターはそのリンク上でレコードの読み取りを進め、コネクターが既に現在のルール・セット実行においてマスター・リンク上で受け取ったレコードと一致するかどうか試します。

キーの不一致のためにレコードをリジェクトする必要がある場合、リジェクトされるレコードが、リジェクト・リンクが定義された入力リンクから来た場合に限り、また、そのリジェクト・リンクに「**キー・ミスマッチ・エラー**」オプションが選択

されている場合に限り、コネクターはそのレコードをリジェクトできます。そうでない場合は、コネクターはエラーをログに記録し、ジョブは失敗します。

バッチ・モードでは、入力リンクのいずれかのルール・セット・パラメーターが Java 配列タイプに基づかない場合、コネクターは、バッチ・サイズ値を 1 に強制します。キー・モードでは、マスター・リンクのルール・セット・パラメーターが Java 配列タイプに基づかない場合、コネクターは、バッチ・サイズ値を 1 に強制します。ユーザーが指定した「**バッチ・サイズ**」値を実行時にコネクターがオーバーライドし、値 1 に強制する必要がある場合、コネクターはジョブ・ログに、この変更についての通知メッセージを書き込みます。

レコードがトランザクション・ウェーブの中のステージの入力リンクに到着した場合、コネクターは、別のウェーブのレコードから独立して、各ウェーブの中のレコードを処理します。単一のルール・セット実行のルール・セット・パラメーター値を生成するために、あるウェーブからのレコードは、別のウェーブからのレコードと結合されません。

実行オブジェクト・モデル・タイプ

ILOG JRules Connector は、動的 Execution Object Model (動的 XOM) または Java Execution Object Model (Java XOM) のどちらかを使用できます。コネクターは、動的 XOM タイプと Java XOM タイプを組み合わせたルール・セット・パラメーターはサポートしません。ルール・セットのすべてのパラメーターは、同じ XOM タイプに基づいている必要があります。これを確実にするために、「**XOM タイプ**」プロパティーは、リンクではなくステージに関連付けられています。

Java XOM

Java XOM モードのステージを構成するには、「**XOM タイプ**」ステージ・プロパティーは、「**Java XOM**」に設定する必要があります。構成ウィザードを使用してルール・セット・パラメーターをインポートし、Java XOM に基づいてルール・セットのステージを構成すると、ウィザードは、ステージの「**XOM タイプ**」プロパティーを「**Java XOM**」に自動的に設定します。

「**Java XOM**」オプションを選択する場合は、「**ルール・セット・パラメーター・クラス**」リンク・プロパティー値を指定する必要があります。指定された値は、そのリンクに関連付けられたルール・セット・パラメーターの Java タイプを表します。指定されたタイプが、コネクターによってサポートされる基本 Java タイプの 1 つである場合、リンク上の単一列が全体のルール・セット・パラメーターに対応します。そうでない場合、入力リンク上の列は、ルール・セット・パラメーターの Java クラスのメソッドの引数に対応し、出力リンク上の列は、ルール・セット・パラメーターの Java クラスのメソッドの戻り値に対応します。コネクターの環境での基本 Java タイプの詳細は、『*基本 Java タイプおよび基本メソッド*』のトピックを参照してください。

動的 XOM

コネクターは、動的 XOM に基づくルール・セットの呼び出しをサポートします。このモードでは、ルール・セット・パラメーター値は XML 文書です。ルール・セットの XOM の定義に使用された XML 文書は、XML スキーマ文書に関して有効でなければなりません。

動的 XOM モードのステージを構成するには、「**XOM タイプ**」ステージ・プロパティーは、「**動的 XOM**」に設定する必要があります。構成ウィザードを使用してルール・セット・パラメーターをインポートし、動的 XOM に基づいてルール・セットのステージを構成する場合、ウィザードは自動的に、ステージ上の「**XOM タイプ**」プロパティーを「**動的 XOM**」に設定します。

ステージが動的 XOM 用に構成されている場合、コネクターは各入力リンクから 1 つのレコードを読み取り、これらのレコードから XML 文書を生成し、ルール・セットを呼び出して、生成した XML 文書をルール・セットの IN および IN_OUT パラメーター値として渡します。

コネクターが入力リンクからレコードを読み取るときに、コネクターは、レコードの中で、ルール・セット・パラメーターの XML 文書の値を作成する元の XML データを含むフィールドを検索します。全体の XML 文書がレコードの単一フィールドの中に保管される必要があります。レコードの残りのフィールドは、複数の入力リンクにわたってレコードを関連させるためのキー列値として使用できます。あるいは、コネクターが、入力リンクから出力リンクへ伝搬させるフィールドにすることもできます。

ルール・セット・パラメーターの XML 文書のために使用される入力リンク上の列は、ルール・セット・パラメーター値として使用される実際の XML データを含むそのリンク上の取得された各レコードの中のフィールドにコネクターがアクセスできるように、識別される必要があります。識別された列は、文字ベースの DataStage データ・タイプである Char、VarChar、LongVarChar、NChar、NVarChar、および LongNVarChar でなければなりません。複数のそのような列がある場合、その中の 1 つのみが、`CC_JRules();` 式を含む「**説明**」属性値を持つ必要があります。「**説明**」属性は、ステージの中のいずれかの入力リンクまたは出力リンクをクリックした後で表示される列タブの中に表示されます。これらの条件を検査した後、リンク上に適格な列が存在しないか、複数存在する場合は、ジョブはエラーで失敗します。

コネクターが、動的 XOM に基づくルール・セットを呼び出すように構成されている場合、バッチ・モードはサポートされません。このような場合には、1 のバッチ・サイズが暗黙に使用されます。コネクターは、単一のルール・セット・パラメーター値で複数の XML 文書を渡すことはサポートしません。

キー・モードは、動的 XOM に基づくルール・セットを呼び出すように構成されたコネクターによってサポートされます。この場合、指定されたキー列は、ステージの複数の入力リンクおよび複数の処理ノードにわたる入力レコードのソートおよびパーティショニングに使用され、コネクターが、ルール・セットのすべての IN パラメーターおよび IN_OUT パラメーターにわたって関連させられた XML 文書値で常にルール・セットを呼び出すことを保証することができます。

DataStage フィールドとルール・セット・パラメーターのマッピング

ILOG JRules connector は、レコードを、Java オブジェクトへの入力リンクで使用可能なレコードに変換します。コネクターは、変換を完了するために、コネクター・ステージ・フィールドとルール・セット・パラメーターの間のマッピングを識別する必要があります。

マッピングは、構成ウィザードを使用することによって自動的に識別することもできます。以下のセクションでは、どのように InfoSphere DataStage® フィールドがルール・セット・パラメーターにマップされるかを説明します。

Java XOM

Java XOM の場合、リンク上のフィールドと、そのリンクに関連付けられたルール・セット・パラメーターの間のマッピングは、「**列名**」および「**SQL タイプ**」のフィールド属性と、以下のいずれかに基づいて決定されます。

- ルール・セット・パラメーター・クラスの中に示されている方式
- ルール・セット・パラメーター名前およびデータ・タイプ (それが基本 Java タイプの場合)

基本 Java タイプの詳細は、『*基本 Java タイプおよび基本メソッド*』を参照してください。ルール・セット・パラメーターが、『*自動マッピング*』セクションに示された条件に従っている場合、マッピングは、フィールドの「**列名**」属性に基づいて自動的に行われます。そうでない場合、マッピングは、フィールドの「**説明**」属性に示されたマッピング仕様を使用して行われます。

自動マッピング

以下の条件のいずれかを満たす場合、ILOG JRules connector は、リンク・フィールドとルール・セット・パラメーターの間のマッピングを自動的に識別できます。

1. ルール・セット・パラメーターが、コネクターによってサポートされる基本 Java タイプの 1 つである場合、「**列名**」は、ルール・セット・パラメーターの名前と一致しなければなりません。フィールドの「**SQL タイプ**」は、ルール・セット・パラメーターの基本 Java タイプと互換でなければなりません。マッピングまたはデータ・タイプの互換性の詳細は、『*DataStage タイプから Java タイプへのマッピング*』および『*Java タイプから DataStage タイプへのマッピング*』を参照してください。この場合、フィールドは、ルール・セット・パラメーターに直接マップされます。
2. ルール・セット・パラメーターが基本 Java タイプでなく、リンクが入力リンクである場合、ルール・セット・パラメーター・クラスには、**setFieldName()** という名前のメソッドがなければなりません。ここで、**FieldName** は、フィールドの「**列名**」です。「**列名**」とメソッド名の比較で、大/小文字は区別されません。メソッドには戻りタイプがあってはならず、1 つの引数を持っている必要があります。また、入力リンク上のフィールドの「**SQL タイプ**」は、メソッドが求める引数のデータ・タイプと互換である必要があります。例えば、入力リンク上のタイプが **VarChar(10)** の「**firstName**」フィールドは、**void setFirstName(String name)** メソッドの最初の唯一の引数にマップします。この場合、入力フィールドは、メソッドが求める引数にマップされます。
3. ルール・セット・パラメーターが基本 Java タイプでなく、リンクが出力リンクである場合、ルール・セット・パラメーター・クラスには、**getFieldName()** という名前のメソッドがなければなりません。ここで、**FieldName** は、フィールドの「**列名**」です。「**列名**」とメソッド名の比較で、大/小文字は区別されません。特例は、メソッドの戻りタイプが **boolean** または **java.lang.Boolean** の場合です。この場合、メソッドは、**isFieldName()** という名前が予期されます。メソッ

ドからの戻り値は、出力リンクにあるフィールドの「**SQL タイプ**」と互換でなければなりません。この場合、フィールドは、メソッドの戻り値にマップされます。

第 2 の条件 (および「**説明**」属性が空) に従うルール・セット・パラメーター・クラスの中にメソッドがない入力リンクにフィールドがある場合、フィールド値は、ILOG JRules に渡されません。同様に、第 3 の条件 (および「**説明**」属性が空) に従うルール・セット・パラメーター・クラスの中にメソッドがない出力リンクにフィールドがある場合、フィールド値は、ILOG JRules connector によって設定されません。

「説明」仕様を使用したマッピング

ルール・セット・パラメーターが自動マッピングの条件に従わない場合、ILOG JRules connector は、InfoSphere DataStage の入力レコードおよび出力レコードの中の各フィールド値の設定および取得のために呼び出されるメソッドを指定するメカニズムを提供します。これは、ステージ・エディターの「**列**」タブにあるスキーマ・フィールドの「**説明**」属性の中に指定されています。リンク・スキーマのすべてのフィールドについて、「**説明**」には、リンクに関連付けられた Java クラスで呼び出されてフィールド値を設定または取得するメソッドに関する情報が入っています。どのフィールドの「**説明**」属性にも情報が入っていない場合、ルール・セット・パラメーターは、『*自動マッピング*』セクションに記載されている仕様に従っていると想定されます。

入力リンクの列の場合、その「**説明**」属性の値は、1 つ以上の連結された式から成ります。ここで、各式は、`CC_JRules(methodName, parameterIndex);` の形式です。`methodName` は、この列のルール・セット・パラメーター・オブジェクトに対して呼び出されるメソッドの名前を示します。`parameterIndex` 値は、1 以上の整数で、列のマップ先の (メソッド用の) パラメーターの索引を示します。

例えば、Java オブジェクトはメソッド `setFullName(String firstName, String lastName)` と、*firstName* および *lastName* の 2 つのフィールドを含む DataStage レコードを持ちます。この場合、*firstName* の「**説明**」属性は `CC_JRules (setFullName,1);` で、*lastName* では `CC_JRules(setFullName,2);` です。DataStage レコードの *firstName* フィールド値は、`setFullName` メソッドの最初の引数に使用され、同じレコードの *lastName* フィールド値は、2 番目の引数に使用されます。

methodName は、コンストラクターであることもできます。例の中の *firstName* 列がクラス `Customer` のコンストラクターの最初の引数にも使用される場合、*firstName* の「**説明**」属性は、`CC_JRules(setFullName,1);CC_JRules(Customer,1);` です。

出力リンクでは、メソッドは、Java オブジェクトからフィールド値を取得するために呼び出されます。したがって、フィールドは、どのようなパラメーターも予期しません。このため、出力リンクのフィールドにはパラメーター索引は必要ありません。出力リンクのフィールドの「**説明**」属性の構文は、`CC_JRules(methodName);` です。それぞれのフィールドについて、呼び出すことができるメソッドは、1 つのみである必要があります。複数のメソッドが存在する場合、ILOG JRules connector は、エラーをログに記録し、ジョブは失敗します。

リンク上の単一列が全体のルール・セット・パラメーター値を表す場合、その列の「**説明**」属性には CC_JRules(); 値があります。これは、ルール・セット・パラメーターが、コネクターによってサポートされる基本 Java タイプの 1 つである場合です。

動的 XOM

InfoSphere DataStage レコードのフィールドに、動的 XOM に基づくルール・セットに送信される XML 文書が含まれている場合、そのフィールドの「**説明**」属性に CC_JRules(); 値を含めることができます。 InfoSphere DataStage レコードに複数のフィールドがある場合、XML 文書が含まれるフィールドを示す値が必要です。

レコードに単一フィールドが含まれる場合、そのフィールドが、XML 文書を含むフィールドであると自動的に想定されるため、そのフィールドの「**説明**」属性は、空のままにすることもできます。XML 文書を含むフィールドは、例えば、Char、NChar、VarChar、NVarChar、LongVarChar、または LongNVarChar などの、InfoSphere DataStage 文字ベースのデータ・タイプのいずれかである必要があります。

フィールドの伝搬方式

ILOG JRules connector は、入力リンクから出力リンクまでの不一致のフィールドのパススルーをサポートします。不一致のフィールドは、コネクターによって、ルール・セットの呼び出しに使用されない入力リンク・フィールドです。不一致のフィールドの伝搬を使用可能にすると、不一致のフィールドは、入力リンクに関連付けられた出力リンクに伝搬されます。

コネクターは、不一致のフィールドを持つ各入力リンクと、これらの不一致のフィールドが伝搬される必要のある出力リンクとの間に、自動的に関連付けを行います。入力リンクのすべての不一致のフィールドと、その入力リンクに関連付けられたすべての出力リンクについて、コネクターは、出力リンクに不一致のフィールドと同じ名前のフィールドが含まれるかどうか検査します。

- 不一致のフィールドと同じ名前のフィールドが含まれる場合で、出力フィールドが XOM のどのフィールドにもマップされない場合、入力列のフィールド値は伝搬され、出力列のフィールド値として設定されます。
- 不一致のフィールドと同じ名前のフィールドが含まれない場合、コネクターは、その出力リンクの「**ランタイム列伝搬**」チェック・ボックスが選択されているかどうか検査します。選択されている場合は、コネクターは、入力フィールド定義をその出力リンク・スキーマに追加し、ジョブ実行時にフィールド値を伝搬します。チェック・ボックスが選択されていなければ、入力フィールド定義および値の伝搬は行われません。

不一致のフィールドの伝搬に、以下のいずれかの方式を使用できます。

レコードの順序に基づくマッチング

この方式が選択されると、コネクターは、最初の入力リンクから最初の出力リンクへ、2 番目の入力リンクから 2 番目の出力リンクへ、というように不一致のフィールドを伝搬します。出力リンクよりも入力リンクが多い場合、対応する出力リンクがない入力リンクについては、フィールドの伝搬は行われません。入力リンクは、1

つの出力リンクのみに関連付けられることができます。同様に、出力リンクは、1 つの入力リンクのみに関連付けられることができます。

この方式を使用する場合、ルール・セットの呼び出しに使用された入力リンクのレコードの数は、ルール・セットの実行後に、関連付けられた出力リンクで生成されたレコードの数と一致しなければなりません。数が一致しない場合、コネクターは、エラーをログに記録し、ジョブは失敗します。

伝搬キー列値に基づくマッチング

この方式を使用する場合、コネクターは、入力リンクから同じ伝搬キー列をもつ出力リンクに、不一致のフィールドを伝搬します。「**伝搬キー列**」リンク・プロパティーを通して、伝搬キー列は、それぞれの入力リンクに別々に指定されます。

コネクターは、それぞれの入力リンクをチェックして、伝搬キー列としてマークを付けられた列がないかどうか検査します。伝搬キー列としてマークを付けられた列を持つ入力リンクが検出され、また、そのリンクがルール・セット実行に使用されていないフィールドを持つ場合、これらのフィールドは、フィールド伝搬に適格です。

伝搬キー列は、同時にフィールド伝搬に適格である列であることはできません。このシナリオが存在する場合、ジョブはエラーで失敗します。コネクターは入力リンクを検査し、伝搬キー列および伝搬に適格の列を持つ各入力リンクに対して、以下のステップを実行します。

- 前の入力リンクのどれともまだ関連付けられていないすべての出力リンクを検査します。
- 入力リンクの伝搬キー列と同じ名前とデータ・タイプの列を含む出力リンクは、その入力リンクに関連付けられます。

この方式では、入力リンクは複数の出力リンクに関連付けられることができますが、出力リンクは、1 つの入力リンクのみにしか関連付けられることができません。

コネクターは、伝搬キー列の状態が使用可能になっている入力リンク上のレコードを読み取るとき、レコードの伝搬キー列値を検査します。値が、現在のルール・セット実行の同じリンク上で取得された別のレコードの値と一致した場合、ジョブは、エラーで失敗します。

出力リンクに送られるそれぞれのレコードのレコード・フィールドは、ルール・セットのそれぞれの OUT または IN_OUT パラメーターの値からデータが取り込まれます。そのとき、コネクターは、フィールド伝搬のために、出力リンクが入力リンクに関連付けられているかどうか検査します。関連付けられていれば、コネクターは、現在のルール・セット実行の入力リンクから取得し、出力レコードと同じ伝搬キー列値を持つ入力レコードを検索します。このようなレコードが存在する場合、コネクターは、伝搬されるフィールドの値を入力レコードから出力レコードにコピーします。このような入力レコードが存在しなければ、出力レコードの中の伝搬されたフィールドは、設定されないままにされます。

ILOG JRules ステージに 2 つの入力リンクと 2 つの出力リンクが含まれるジョブの例を考えます。

リンクには以下の列が含まれます。

入力リンク

リンク 1: (id、balance)

リンク 2: (id、name、address)

出力リンク

リンク 1: (name、cust_type、address)

リンク 2: (name、outstanding、address)

伝搬キー列方式を使用してフィールド伝搬が使用可能であり、かつ、「**name**」列がリンク 2 の伝搬キー列として指定されている場合、コネクターは、「**name**」列も 2 つの出力リンク上に定義されていることを検出します。「**address**」列が、入力リンクおよび出力リンクの両方のルール・セットにマップされていない場合、入力のリンク 2 からの「**address**」列のデータは、両方の出力リンクの「**address**」列に送られます。マッピングは、「**name**」フィールドに基づいて行われます。例えば、入力のリンク 2 が以下のデータを持っているとします。

1、"John"、"123, Las Vegas Dr, Las Vegas, NV"

2、"Paul"、"abc Drive, New York"

出力リンクのレコードは、次のようになります。

リンク 1

"John"、"Gold"、"123, Las Vegas Dr, Las Vegas, NV"

"Paul"、"Silver"、"abc Drive, New York"

リンク 2

"John"、"244.33"、"123, Las Vegas Dr, Las Vegas, NV"

"Paul"、"435.6445"、"abc Drive, New York"

"Manish"、4363.453"、""

入力「**address**」列からのデータは、「**name**」列の値を使用して、出力「**address**」列にマップされます。出力のリンク 2 の 3 番目のレコードの「**name**」フィールド値は、入力のリンク 2 のどのレコードの「**name**」フィールド値とも一致しません。そのため、出力レコードの「**address**」フィールド値は、設定されないままです。

Java オブジェクト ID に基づくマッチング

この方式を使用する場合、コネクターは、入力レコードから、同じ Java オブジェクトに対応する出力レコードに、不一致のフィールドを伝搬します。各入力リンクに対して、コネクターは、そのリンクに指定されたルール・セット・パラメーターの Java タイプを検査します。このタイプがコネクターによってサポートされる基本

Java タイプの 1 つでない場合、コネクターは、リンク上の、フィールド伝搬に適格である (これは、列がルール・セット・パラメーター値の生成に使用されていないことを意味する) すべての列をメモします。

伝搬に適格ないくつかの列 (フィールド) を持つそれぞれの入力リンクについて、コネクターは、内部で、次のように、その入力リンクと出力リンクの間の関連付けをします。

- 前の入力リンクのどれともまだ関連付けられていないすべての出力リンクを検査します。
- ルール・セット・パラメーターが入力リンクに指定されたものと互換の Java タイプの出力リンクは、その入力リンクに関連付けられます。この場合、2 つの Java タイプは同一であるか、一方が他方から派生されたものであれば、互換です。互換性の検査では、Java 配列タイプは、配列タイプではないとして扱われます。例えば、一方の Java タイプが ClassA で、他方の Java タイプが ClassB[] の場合、この 2 つの Java タイプは、ClassA と ClassB が同じであるか、一方が他方から派生されたものである場合、互換です。

このフィールドの伝搬方式が実行されると、1 つの入力リンクは複数の出力リンクに関連付けられることができますが、1 つの出力リンクは、1 つの入力リンクのみにしか関連付けられることができません。

コネクターが出力リンクに送るそれぞれのレコードについて、最初に、レコード・フィールドに、ルール・セットのそれぞれの OUT または IN_OUT パラメーターの値からデータを取り込み、次に、その出力リンクがフィールド伝搬目的のために入力リンクに関連付けられているかどうか検査します。関連付けられている場合、コネクターは、現在のルール・セット実行の入力リンクから取得されたレコードから、出力レコードと同じ Java オブジェクトと一致するレコードを探します。このようなレコードが存在する場合、コネクターは、伝搬されるフィールドの値を入力レコードから出力レコードにコピーします。このようなレコードが存在しなければ、出力レコードの中の伝搬されたフィールドは、設定されないままにされます。

Java クラスパス構成

ILOG JRules Connector は、IBM InfoSphere Information Server エンジン層ホスト上で実行されます。ILOG JRules connector は Java ベースのコネクターで、正しく機能するためには、さまざまな ILOG JRules リソース (ILOG JRules JAR ファイル、Java Execution Object Model (XOM) クラスなど)、および構成ファイルへのアクセスを必要とします。

コネクターの Java クラスパスの中に、コネクターが必要とするリソースを指定する必要があります。「ILOG JRules Connector ステージ」ページの「**クラスパス**」プロパティーに、必要なリソースを指定できます。

すべての必要なリソースをリストする InfoSphere DataStage プロジェクト環境変数を定義し、それを、「**クラスパス**」プロパティー値と指定されたジョブ・パラメーターとしてジョブに追加できます。リソースが「**クラスパス**」プロパティーの中に指定されると、ジョブ実行および構成ウィザード・セッションの中で、コネクターがそれらのリソースを使用できるようになります。

システムの Java クラスパスを使用して必要なリソースをリストすることは推奨されません。システム・クラスパスの中に指定すると、リソースは、同じマシン上で動作している他の Java プロセスを妨害する可能性があります。また、システム・クラスパスに追加された新しいエントリーの認識のために IBM InfoSphere DataStage エンジンを再始動する必要があるため、リソースをシステム・クラスパスにその都度追加していくのは不便でもあります。エンジンが Windows システムである場合は、リブートが必要です。システム・クラスパスを定期的に更新する必要がある例は、新しいルール・セットを呼び出すためにコネクターが使用する新規 Java XOM JAR ファイルを追加する場合です。

各 ILOG JRules ステージの「**クラスパス**」プロパティーに必要なリソースを追加するのが不便であれば、システム・クラスパス値を設定する代わりに、以下のいずれかを行うことができます。

- ジョブのランタイム実行については、IBM InfoSphere DataStage プロジェクト・レベルで CLASSPATH 環境変数を定義し、この環境変数のデフォルト値の中に、必要なリソースをリストします。そうすると、その IBM InfoSphere DataStage プロジェクトの中の各ジョブによって、指定された CLASSPATH 値が取り出されます。プロジェクト・レベルで指定されたデフォルト値をオーバーライドするために、ジョブは、この環境変数をジョブ・パラメーターとしてインポートすることができ、特定のジョブのコネクター・クラスパス値がその値を使用するように設定できます。
- 構成ウィザードについては、必要なリソースを *IS_HOME*/ASBNode/lib/java にコピーします。ここで、*IS_HOME* は、IBM InfoSphere Information Server エンジン層ホスト上の IBM InfoSphere Information Server インストール済み環境のホーム・ディレクトリーを表します。

  注: 構成ウィザード・セッションは ASB エージェント・サービスの制御下で実行されるため、構成ウィザード・セッションのシステム・クラスパスにリソースを追加することはできません。ASB エージェント・サービスは、システム・クラスパス値で指定されているリソースを含まない内部のカスタム・クラスパスで初期化されます。

選択したエンジン・モードに応じて、ジョブ実行の間、および構成ウィザードの使用のために、指定された JRules JAR ファイルがクラスパスの中になければなりません。

構成ウィザードは、IBM InfoSphere Information Server エンジン・ホスト上の ASB エージェント・サービス・プロセスの下で実行されます。ウィザードが、「**クラスパス**」プロパティーの中に指定されている Java XOM JAR ファイルから Java XOM クラスをロードするときに、ウィザード・セッションの間、その JAR ファイルはロックされた状態になります。ウィザードは、それぞれのセッションで XOM クラスを単独でロードします。したがって、必要であれば、一般に、ウィザード・セッションとウィザード・セッションの間で、XOM JAR を上書きすることは可能です。例えば、JAR の新しいバージョンを、古いバージョンの代わりに配置するなどです。しかし、プラットフォームによっては、クラスがロードされた ASB エージェント・プロセスがまだ実行中であるため、ウィザード・セッションが完了した後であっても、JAR ファイルはロックされたままの場合があります。そのような場合、ロックされた JAR を削除したり、別の JAR を移動してロックされた JAR を置換したりできない場合であっても、ロックされた JAR の上に別の JAR をコピー

することはできる場合があります。それもできない場合のもう 1 つのオプションは、新規 JAR を新しい場所に置いて、その場所を古い JAR の場所の代わりに「**クラスパス**」プロパティーに指定することです。この最後のオプションで、ASB エージェント・プロセスは再始動できます。

コア・エンジン・モード

ジョブの実行中、クラスパスの中に *JRULES_HOME*/executionserver/lib/jrules-engine.jar JAR ファイルがなければなりません。ここで、*JRULES_HOME* は、IBM InfoSphere Information Server エンジン・ホスト上の ILOG JRules 製品のホーム・ディレクトリーを表します。

この JAR ファイルは、構成ウィザード・セッションには必要ありません。

コネクターが構成されるルール・セットが Java XOM に基づく場合、ルール・セット・パラメーターの XOM クラスもクラスパスの中に含まれている必要があります。これは、ジョブ・ランタイムおよび構成ウィザード・セッションに当てはまります。

コネクターがコア・エンジン・モード用に構成されている場合、コネクターは構成ファイルへのアクセスを必要としません。

J2SE RES XU モード

ジョブ実行の間、および構成ウィザードの使用のために、以下の JRules JAR ファイルがクラスパスの中になければなりません。

JRULES_HOME/executionserver/lib/jrules-engine.jar

JRULES_HOME/executionserver/lib/jrules-res-session-java.jar

JRULES_HOME/executionserver/lib/jrules-res-execution.jar

JRULES_HOME/executionserver/lib/sam.jar

JRULES_HOME/executionserver/lib/log4j-1.2.8.jar

JRULES_HOME/executionserver/lib/j2ee_connector-1_5-fr.jar

ここで、*JRULES_HOME* は、IBM InfoSphere Information Server エンジン・ホスト上の ILOG JRules 製品のホーム・ディレクトリーを表します。

コネクターが構成されるルール・セットが Java XOM に基づく場合、ルール・セット・パラメーターの XOM クラスもクラスパスの中に含まれている必要があります。これは、ジョブ・ランタイムおよび構成ウィザード・セッションに当てはまります。

ルール・セット・リポジトリーが JDBC ベースである場合、クラスパスの中に、JRules エンジンがリポジトリーにアクセスするために必要な JDBC ドライバーの実装がなければなりません。例えば、ルール・セット・リポジトリーが DB2 データベースとして実装された場合、リポジトリーへのアクセスのために、DB2 製品のインストールで使用可能な db2jcc4.jar JDBC ドライバーが選択される可能性があります。

ルール・セット・リポジトリーのタイプとリポジトリーの接続情報は、ra.xml JRules 構成ファイルの中に保管されます。このファイルが含まれるディレクトリー

は、コネクターのクラスパスの中にリストされる必要があります。JRules インストールは、*JRULES_HOME*/executionserver/bin ディレクトリーにあるサンプルの ra.xml ファイルによって行い、これは、特定のルール・セット・リポジトリーにアクセスするようにカスタマイズできます。 ra.xml 構成ファイルの詳細は、ILOG JRules の製品資料を参照してください。

J2EE RES XU モード

ルール・セット・リポジトリーへの接続は、Rule Execution Server が J2EE サーバーにデプロイされるときに構成されます。

ジョブの実行中に、コネクターはリモート EJB3 クライアントとして機能し、Rule Execution Server がデプロイされた J2EE サーバーにデプロイされる必要がある EJB3 セッション・コンポーネントへの接続を確立します。このエンジン・モードでは、ILOG JRules Java EE アドオンが、JRules 環境に正しくインストールおよび構成されていることが必要です。以下の JRules JAR ファイルがジョブ実行のクラスパスになければなりません。ここで、JRULES_HOME は、IBM InfoSphere Information Server エンジン・ホスト上の ILOG JRules 製品のホーム・ディレクトリーを表します。

JRULES_HOME/executionserver/lib/jrules-engine.jar

JRULES_HOME/executionserver/lib/jrules-res-session-java.jar

JRULES_HOME/executionserver/lib/jrules-res-execution.jar

JRULES_HOME/executionserver/lib/sam.jar

JRULES_HOME/executionserver/lib/log4j-1.2.8.jar

JRULES_HOME/executionserver/lib/j2ee_connector-1_5-fr.jar

JRules JAR ファイルに加えて、J2EE サーバー上の EJB3 コンポーネントへの接続を確立するために必要な J2EE クライアント JAR ファイルがなければなりません。

WebSphere Application Server v7 上で JRules エンジンが実行中で、コネクターがそれに、エンジン層マシン上にインストールされた WebSphere Application Client v7 を通して接続する場合、以下の JAR ファイルをクラスパスに追加する必要があります。

- *WAS_HOME*/runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar
- *WAS_HOME*/runtimes/com.ibm.ws.webservices.thinclient_7.0.0.jar

ここで、*WAS_HOME* は、IBM InfoSphere Information Server エンジン層マシン上の WebSphere Application Client 7 のホーム・ディレクトリーです。

注: これらの 2 つの JAR ファイルをインストールに含めるために、WebSphere Application Client のインストール時に、「**スタンドアロンのシン・クライアントおよびリソース・アダプター**」オプションを選択する必要があります。

EJB3 スタブ JAR ファイルを作成し、クラスパスの中にリストする必要があります。このスタブ JAR ファイルは、WAS_HOME/bin ディレクトリーにある createEJBStubs スクリプトを実行することで生成されます。ここで、WAS_HOME は、Rule Execution Server がデプロイされた WebSphere Application Server v7 インストール済み環境のホーム・ディレクトリーです。このスクリプトのファイル拡張

子は、オペレーティング・システム間で異なります。例えば、Windows では .bat で、Linux では .sh です。スクリプトは、*JRULES_HOME*/executionserver/applicationservers/WebSphere7 ディレクトリーの下にある jrules-res-session-ejb3-WAS7.jar ファイルに対して、1 回実行する必要があります。オプション -newfile で、生成されるスタブ・ファイルの場所を指定します。例えば、以下のように指定します。

```
WAS_HOME/bin/createEJBStubs.sh JRULES_HOME/executionserver/
applicationservers/WebSphere7/jrules-res-session-ejb3-WAS7.jar -newfile
JRULES_HOME/executionserver/lib/jrules-res-session-ejb3-WAS7_stub.jar
```

生成された EJB3 スタブ・ファイルは、IBM InfoSphere Information Server エンジン・ホストにコピーし、ジョブ実行のクラスパスに含める必要があります。

コネクターが構成されるルール・セットが Java XOM に基づく場合、ルール・セット・パラメーターの XOM クラスもクラスパスの中に含まれている必要があります。これは、ジョブ・ランタイムおよび構成ウィザード・セッションに当てはまります。このエンジン・モードでは、Java XOM クラスは java.io.Serializable インターフェースを実装する必要があります。そうでない場合、ルール・セット・パラメーター値が EJB 接続を介して適切に転送されないため、ジョブは、CORBA マーシャル例外で失敗します。

コネクターはまた、jndi.properties 構成ファイルにアクセスする必要もあります。このファイルには、J2EE サーバー上の EJB3 ルール・セッション・コンポーネントへの接続を確立するための初期コンテキスト (java.naming.InitialContext オブジェクト) を作成するために必要なプロパティーが含まれます。サンプルの jndi.properties ファイルが *JRULES_HOME*/executionserver/samples/j2eerulesession/websphere7 ディレクトリーの下の JRules インストール済み環境にあります。このファイルをカスタマイズし、このファイルが含まれるディレクトリーをコネクターのクラスパスに追加できます。

コネクターが使用する WebSphere Application Client インストール済み環境から WebSphere Application Server への接続を確立するための追加情報が含まれた、sas.client.props、ssl.client.props、および soap.client.props の各ファイルへのコネクターをポイントする必要もあります。WebSphere Application Client インストール済み環境には、特定の WebSphere Application Server インスタンスに接続するようにカスタマイズして構成できるサンプル・ファイルが含まれています。これらのサンプル・ファイルは、*WAS_HOME*/properties ディレクトリーに保管されています。

ステージ・プロパティーの「**Java プロパティー**」セクションの下にある「**JVM 設定**」プロパティーの中で、以下のオプションを指定します。

```
-Dcom.ibm.CORBA.ConfigURL=file:WAS_HOME/properties/sas.client.props
-Dcom.ibm.SSL.ConfigURL=file:WAS_HOME/properties/ssl.client.props
-Dcom.ibm.SOAP.ConfigURL=file:WAS_HOME/properties/soap.client.props
-Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
-Djava.util.logging.configureByServer=true
```

ここで、*WAS_HOME* は、IBM InfoSphere Information Server エンジン層マシン上の WebSphere Application Client 7 のホーム・ディレクトリーです。

注: 構成ウィザードは、WebSphere Application Server にデプロイされた JRules Rule Execution Server にアクセスすることはありません。構成ウィザードが起動されるときに J2EE RES XU エンジン・モードが選択されているか、構成ウィザード・セッションでこのエンジン・モードが選択されている場合、ウィザードは、ローカルに管理された JRules Rule Execution Server として JRules エンジンにアクセスします。したがって、この場合の要件は、J2SE RES XU エンジン・モードの場合の要件と同じです。

構成ウィザードによる Java コードの生成

構成ウィザードを使用してステージ・リンクに定義されている列に基づいて Java コードを生成する場合、コネクターは、IBM InfoSphere Information Server によって、またはコネクターが実行される Java Runtime Environment (JRE) によって自動的に提供されるアクセスの他に、リソースへのアクセスを必要としません。

ILOG JRules Connector を使用した InfoSphere DataStage ジョブのデザイン

特定の JRules ルール・セットを呼び出すには、IBM InfoSphere DataStage ジョブを作成し、それを構成する必要があります。

手順

1. ジョブを作成します。
2. ILOG JRules Connector ステージを構成します。
3. 入力リンクおよび出力リンクのプロパティーを構成します。
4. (オプション) リジェクト・リンク・プロパティーを構成します。
5. (オプション) バッファー・リンクを構成します。
6. ジョブをコンパイルします。

ジョブの作成

特定の JRules ルール・セットを呼び出すには、ILOG JRules ステージを付けた IBM InfoSphere DataStage パラレル・ジョブを作成する必要があります。

手順

1. デザイナー・クライアントで、「**ファイル**」 > 「**新規**」を選択します。
2. 「**パラレル・ジョブ**」アイコンを選択して、「**OK**」をクリックします。
3. パラレル・ジョブ・キャンバスで、ソースおよびターゲットのデータ (ファイルまたはデータベース・データなど) にアクセスするためのステージを定義します。
4. デザイナー・クライアントのパレット領域で、「**リアルタイム**」をクリックします。
5. **ILOG JRules Connector** ステージのアイコンを選択し、開いているジョブまでステージをドラッグします。ソース・ステージとターゲット・ステージの間に、ステージを配置します。

6. 異なるステージをリンクします。
7. (オプション) リンクおよびステージの名前を変更します。
8. 「**ファイル**」 > 「**保存**」を選択してジョブを保存します。

ルール・セットを呼び出すステージの構成

ILOG JRules ステージ・ジョブを作成するときに、ステージ・プロパティーを構成する必要があります。

手順

1. パラレル・キャンバスで、ILOG JRules Connector ステージのアイコンをダブルクリックします。
2. 「**プロパティー**」タブで、エンジン・モード・フィールドの値を指定します。
3. 「**ルール・セットの場所**」フィールドの値を指定します。
4. バッチ処理を使用可能にするには、「**バッチ処理を使用可能にする**」フィールドの値として「**はい**」を選択します。
5. キー処理を使用可能にするには、「**キーの処理を使用可能にする**」フィールドの値として「**はい**」を選択します。
6. 「**XOM タイプ**」フィールドの値を指定します。
7. 不一致のフィールドを入力リンクから出力リンクに伝搬するには、「**アンマッチ・フィールドを伝搬**」オプションの値として「**はい**」を選択します。
8. 不一致の入力リンク・フィールドを伝搬するときに、コネクターがどのように入力リンクと出力リンクを関連付けるようにするかに応じて、「**フィールドの伝搬方式**」の値を指定します。
9. 「**Java プロパティー**」セクションで、「**クラスパス**」フィールド値を設定して、ステージに必要な Java クラス定義および構成ファイルを含んだディレクトリーおよびアーカイブを含めます。
10. (オプション) 「**ヒープ・サイズ**」フィールドの値を指定します。
11. (オプション) 「**JVM オプション**」フィールドの値を指定します。
12. 「**OK**」をクリックして保存します。

入力リンクおよび出力リンクのプロパティーの構成

選択された JRules ルール・セットを呼び出す ILOG JRules ステージには、入力リンクと出力リンクが含まれます。ILOG JRules ステージは、複数の入力リンクと出力リンクをサポートします。キー・モードでは、入力レコードは、共通キーによってソートされ、関連付けられる必要があります。1 つのマスター入力リンクと 1 つ以上のセカンダリー入力リンクがあります。

手順

1. ILOG JRules Connector ステージを開きます。
2. 入力リンクの「**プロパティー**」タブを開きます。
3. 「**ルール・セット・パラメーター名**」フィールドに、ルール・セット・シグニチャーに定義されたルール・セット・パラメーターの名前を指定します。
4. 「**ルール・セット・パラメーター・クラス**」フィールドに、リンクに関連付けられたルール・セット・パラメーターの Java XOM クラスを指定します。 Java

XOM クラスが配列の場合、パラメーター名は [] で終わる必要があります。Java XOM クラスが動的 XOM の場合、このフィールドは空のままでなければなりません。

5. ステージ・プロパティーの中で「**アンマッチ・フィールドを伝搬**」オプションが有効で、「**フィールドの伝搬方式**」オプションが「**レコードに存在するキー値を使用して突き合わせ**」に設定されている場合、「**伝搬キー列**」フィールドに、使用する伝搬キー列を指定します。
6. (オプション) バッファー・リンクとして入力リンクを変換するには、「**バッファー・リンク**」フィールドに「**はい**」を設定します。
7. 「**OK**」をクリックして保存します。

コネクター・ステージのリンクとルール・セット・パラメーターの関連

ILOG JRules ステージのリンクは、ルール・セット・パラメーター名にマップされる必要があります。ルール・セットが Java XOM に基づく場合、ルール・セット・パラメーターの完全修飾の Java クラスもリンクに指定される必要があります。パラメーターが配列タイプである場合、クラス名は [] で終わる必要があります。

ILOG JRules connector は入力リンクからレコードを読み取り、それらのレコードを、リンクのマップ先のクラスの Java オブジェクトに変換します。パラメーターが配列タイプである場合、同じ入力リンクから引き出されたすべてのオブジェクトは、配列に保管されます。全体の配列は、パラメーター値として設定されます。

パラメーターが配列タイプでない場合、個々の入力レコードは、Java オブジェクトに変換され、パラメーター値として設定されます。セカンダリー・リンクに関連付けられたパラメーターが配列でなく、かつ、セカンダリー・リンクからの複数の行がプライマリー・リンクの 1 つの行に結合する場合、セカンダリー・リンクからの行は、一度に 1 つのオブジェクトを ILOG JRules に送ります。マスター・リンクからの同じ行は、それぞれのルール・セット呼び出しで ILOG JRules に送られます。

出力リンクでレコードが生成されるようにするには、出力ルール・セット・パラメーターに関連付けられた Java 配列または変数に Java オブジェクトを追加するように、ルールをデザインします。入力リンクと出力リンクが異なるクラスにマップされる場合、出力オブジェクトは、ビジネス・ルールによって作成される必要があります。しかし、同じ Java クラスが入力リンクと出力リンクにマップされる場合、入力パラメーター・オブジェクトを出力パラメーターに渡すことができます。このシナリオでは、出力オブジェクトは、明示的にルールによって作成されることはありませんが、入力から渡されます。

バッファー・リンク

入力リンクがバッファー・リンクとしてマークを付けられている場合、コネクターは、このリンクからレコードを 1 回だけ読み取り、ジョブの間のルール・セット実行ごとに、それらのレコードを再使用します。ILOG JRules Connector ジョブの中の入力リンクを、バッファー・リンクとして構成できます。

バッファー・リンクが構成されると、バッファー・リンクで取り込まれたレコードを使用して、複数のルール・セット実行にわたって使用される一定のルール・セット・パラメーター値が生成されます。

レコードがトランザクション・ウェーブでステージに到着すると、コネクターは、バッファー・リンクからウェーブごとに別々にレコードを読み取り、その同じウェーブの中のルール・セット実行にわたって、それらのレコードを再使用します。

バッファー・リンクにレコードがなければ、バッファー・リンクに関連付けられたルール・セット・パラメーターのタイプに応じてアクションが行われます。ルール・セット・パラメーターが配列 Java タイプに基づく場合、コネクターは、空の配列を作成し、それを、ルール・セット実行にわたってルール・セット・パラメーター値に使用します。そうでない場合は、コネクターはエラーをログに記録し、ジョブは失敗します。

バッファー・リンクに関連付けられたルール・セット・パラメーターが配列 Java タイプに基づいていないか、ステージが、動的 XOM に基づくルール・セット用に構成されている場合、コネクターは、バッファー・リンク上に単一レコードがあり、それを、そのルール・セット・パラメーター値の生成に使用すると予期します。リンク上に複数の使用可能なレコードがある場合は、その入力リンクにリジェクト・リンクが定義されており、「**残存レコード・エラー**」オプションが使用可能であれば、コネクターは、最初のレコードの後のレコードをリジェクト・リンクに送ります。そうでない場合は、コネクターはエラー・メッセージをログに記録し、ジョブは失敗します。

セカンダリー・リンクがバッファー・リンクとして構成されている場合は、コネクターが実行されるモード、およびステージに指定されたバッチ・サイズとキー列にかかわらず、コネクターは、そのバッファー・リンク上のすべての使用可能なレコードを読み取ります。

マスター・リンクがバッファー・リンクとして構成されている場合、コネクターは、まず、マスター・リンク上で使用可能なすべてのレコードを読み取ります。その後、バッファー・リンクとして構成されていない残りのセカンダリー・リンクから、次のようにレコードを読み取ります。

- コネクターがバッチ・モードで実行中で、キー・モードが使用不可の場合、コネクターは、*n* レコード (*n* は、指定されたバッチ・サイズで、0 は無制限を表す) を超えない範囲で、セカンダリー・リンク上のすべての使用可能なレコードを読み取ります。ルール・セットが実行された後、コネクターは、次のルール・セット実行のために、このリンク上でのレコードの読み取りを続行します。
- コネクターがキー・モードで実行中の場合、バッチ・サイズの値は無視され、コネクターは、セカンダリー・リンク上のすべてのレコードを読み取り、そのすべてのレコードが、マスター・バッファー・リンクで取得されたいずれかのレコードの中のキー列値と一致するキー列値を持つと期待します。ジョブの中で、あるいは、レコードがトランザクション・ウェーブで到着する場合はウェーブの中で、単一のルール・セット実行が行われます。リジェクト・リンクがセカンダリー・リンクに定義されていて、そのリジェクト・リンクに「**キー・ミスマッチ・エラー**」オプションが選択されている場合、キー列値がマスター・レコードのどのキー列値とも一致しないレコードはリジェクトされます。そうでない場合は、コネクターはエラーをログに記録し、ジョブは失敗します。

NULL 値の処理

ILOG JRules connector には、入力リンクと出力リンク上のレコードのフィールド内の NULL 値、およびルール・セット・パラメーター値内の Java NULL 値を処理するメカニズムが用意されています。

入力リンク

IN または IN_OUT のルール・セット・パラメーターの値の生成に使用される入力レコードのフィールドが NULL 値の場合、マッピングができるのであれば、その NULL 値は、ルール・セット・パラメーター内の Java NULL 値にマップされます。リジェクト・リンクが構成されていない場合、マッピングができなければ、そのジョブはエラーで失敗します。入力リンクにリジェクト・リンクが構成されていて、「**NULL 入力エラー**」オプションが使用可能であれば、レコードはリジェクトされます。

以下のシナリオでは、NULL の入力値を、ルール・セット・パラメーター内の Java NULL 値にマップすることはできません。

- ルール・セットが動的 XOM に基づいている。この場合、ルール・セット・パラメーターの XML 文書値を表すフィールドは、NULL 設定できません。
- フィールドが、byte、short、int、long、float、double、boolean、char などの、プリミティブ Java タイプのルール・セット・パラメーターにマップしている。プリミティブ Java タイプに NULL 値は指定できません。
- フィールドがルール・セット・パラメーターの Java クラスのメソッドの引数にマップし、その引数がプリミティブ Java タイプである。この場合も、プリミティブ Java タイプに NULL 値は指定できません。

出力リンク

出力リンクに関連付けられた OUT または IN_OUT ルール・セット・パラメーターの値を検査しているときに、Java NULL 値が検出されると、マッピングが可能であれば、Java NULL 値は、対応する出力レコード・フィールドの NULL 値にマップされます。マッピングが可能でなければ、ジョブはエラーで失敗します。

以下のシナリオでは、Java NULL 値を、対応する出力レコード・フィールドの NULL 値にマップすることはできません。

- ルール・セット・パラメーター値が NULL で、これが、NULL 可能でない出力リンクの単一列にマップする。「**NULL 可能**」属性が「**いいえ**」に設定されている場合、列は NULL 可能ではありません。
- ルール・セット・パラメーター値が NULL で、これが、一部は NULL 可能で一部は NULL 可能ではない出力リンクの複数列にマップする。この場合、コネクターは、メソッドの戻り値に基づいてフィールド値を初期化するために、出力パラメーターにメソッドを呼び出すことができません。
- ルール・セット・パラメーター値は NULL ではないが、出力パラメーターの Java オブジェクトに呼び出されるメソッドが NULL を戻し、この戻り値がマップされる列が NULL 可能ではない。

リジェクト・リンクの構成

コネクターがエラーと判断した入力レコードを受け入れるために、ILOG JRules ステージにリジェクト・リンクを構成できます。コネクターがエラーのレコードをステージに定義されたリジェクト・リンクに送るとき、ジョブは、残りの入力レコードを処理するために実行を継続し、ユーザーは、リジェクトされたレコードを検査してエラーの原因を判別できます。コネクターがエラーのレコードを検出した入力リンクにリジェクト・リンクが構成されていない場合、コネクターは、エラーをログに記録し、ジョブは失敗します。

手順

1. ILOG JRules ステージの出力リンクを定義します。
2. 出力リンクを右クリックし、「**リジェクトに変換**」チェック・ボックスを選択して、出力リンクをリジェクト・リンクに変換します。
3. ステージ・エディターを開き、「**リジェクト**」タブを選択します。
4. 「**選択された条件に基づいて行をリジェクト**」リストの下で該当するオプションを選択して、リジェクト条件を設定します。
5. 「**リジェクト行に追加**」セクションで、「**ERRORCODE**」または「**ERRORTEXT**」、あるいは両方を選択して、レコードがリジェクトされた理由を識別するために、リジェクトされたレコードの中にエラー・コードとエラー・テキスト情報を含めなければならないことを示します。
6. 「**リンクからリジェクト**」フィールドで、入力リンクを選択します。
7. 「**異常終了条件**」フィールドに、条件のしきい値を指定します。
8. 「**リジェクト後に異常終了**」フィールドに、リジェクト・リンクの上限を指定します。
9. 「**OK**」をクリックして保存します。

ジョブのコンパイル

ジョブのデザインを完了したら、そのジョブをコンパイルします。

手順

1. デザイナー・クライアントで、コンパイルするジョブを開きます。
2. ツールバーで、「**コンパイル**」をクリックします。
3. エラー・メッセージが表示された場合は、ジョブを編集してエラーを解決します。エラーを解決した後、「**コンパイル**」をクリックして、ジョブをリコンパイルします。 ジョブを正常にコンパイルした後、それを、IBM InfoSphere DataStage and QualityStage® デザイナーまたは IBM InfoSphere DataStage and QualityStage ディレクターから実行できます。

構成ウィザードの使用

ILOG JRules connector には、構成ウィザードが含まれています。この構成ウィザードは、ジョブの実行時に JRules ルール・セットを呼び出すための、ステージ内のプロパティーおよびリンク・スキーマの構成を簡単にするために使用できます。構成ウィザードは、ステージに定義されたリンク・スキーマおよびプロパティー値に基づいて Java コードを自動的に生成するために使用することもできます。

ウィザードを使用したステージの構成

ILOG JRules connector には、ジョブの実行時に、選択された JRules ルール・セットを呼び出すステージ内のプロパティーおよびリンク・スキーマの構成に使用できるウィザードが含まれています。

このタスクについて

特定の JRules ルール・セットを呼び出すステージを構成する場合、入力リンクの数は、IN および IN_OUT ルール・セット・パラメーターの数と一致している必要があり、出力リンクの数は、OUT および IN_OUT パラメーターの数と一致している必要があります。ステージのリジェクト・リンクはこの方式から除外され、構成ウィザードによって無視されます。前もってルール・セット・パラメーターの数とタイプが分かっていれば、ウィザードを呼び出す前に、ステージに、正しい数の正しいタイプのリンクを定義できます。その情報がない場合でも、ウィザードを起動し、それを使用して目的のルール・セットを見つけて、それに定義されたパラメーターの数とタイプを知ることができます。しかし、欠落したリンクがある場合、ウィザードは自動的にリンクをステージに追加できないため、ウィザードをキャンセルし、手動で必要なリンクを追加して、ウィザードがステージ構成を完成できるようにする必要があります。

手順

1. ILOG JRules ステージをダブルクリックします。
2. 「**構成**」をクリックします。
3. 「**アクションとログ・ファイルの設定ウィザード**」ページで、「**ルール・セット・パラメーターをインポート**」を選択します。
4. (オプション) デバッグ情報を保管するために、「**ログ・ファイル・パス**」フィールドにログ・ファイルのパスを指定して、「**次へ**」をクリックします。
5. 「**エンジン・モードとルール・セット・フィルター**」ページで、エンジン・モードとルール・セット・フィルター式を指定して、「**次へ**」をクリックします。
6. 選択したエンジン・モードに応じて、その後の画面を構成します。
   - エンジン・モードとして「**コア・エンジン**」を選択した場合、「**ルール・セット・アーカイブの選択**」ページが表示されます。「**継承されたメソッドを含む**」チェック・ボックスを選択して、ルール・セット・パラメーター Java クラスが上位クラスから継承されたメソッドを処理するかどうかを指定します。ステージを構成するためのルール・セット・アーカイブ・ファイルを指定します。
   - エンジン・モードとして「**J2SE RES XU**」または「**J2EE RES XU**」を選択した場合、「**デプロイされたルール・セットの選択**」ページが表示されます。「**継承されたメソッドを含む**」チェック・ボックスを選択して、ルール・セット・パラメーター Java クラスが上位クラスから継承されたメソッドを処理するかどうかを指定します。ステージを構成するためのデプロイされたルール・セットを指定します。
7. 「**ルール・セット・パラメーターの選択**」ページで、インポート・モードとルール・セット・パラメーター、およびステージに定義されたリンクのスキーマ定義を構成するための対応する Java クラス・メソッドを構成します。

8. 「**リンクとルール・セット・パラメーターのスキーマ・マッピング**」ページで、ステージのルール・セット・パラメーターとリンクの間の関連を定義します。
9. 「**ルール・セット・パラメーターをインポート**」のプレビュー・ページで、期待される結果のレポートおよびサマリーを検討して、「**完了**」をクリックします。

ウィザードを使用した、ステージの Java コードの生成

ウィザードを使用して、ステージに定義されたリンクの列定義に基づいて、Java コードを生成できます。

手順

1. ILOG JRules Connector をダブルクリックします。
2. 「**構成**」をクリックします。
3. 「**アクションとログ・ファイルの設定ウィザード**」ページで、「**ウィザードのアクション**」フィールドに「**Java コードを生成**」値を選択します。
4. 「**Java コードの設定**」ページで、「**ソース・コード・ディレクトリー・パス**」フィールドに、エンジン・ホスト上の、生成された `.java` ファイルを書き込む完全修飾のディレクトリー・パスを指定します。
5. 「**出力フォーマット**」フィールドに、コードを生成する Java タイプのフォーマットを指定します。
6. 「**メンバー変数の接頭部**」フィールドに、生成された専用メンバー変数名に使用する接頭部を指定します。
7. 「**既存のファイルを置き換え**」チェック・ボックスを選択して、同じ名前と場所の `.java` ファイルを上書きするかどうか指定します。
8. 「**メソッドを個別に除外**」チェック・ボックスを選択して、選択したメソッドにコードが生成されないように、報告されたメソッドのリストから、手動で個別のメソッドの除外をするかどうか指定します。
9. 「**Java タイプのソース・コード・マッピング**」ページで、コードが生成される Java メソッドを検査します。前のステップで「**メソッドを個別に除外**」オプションを選択した場合は、コードを生成しないメソッドを選択します。「**次へ**」をクリックします。
10. 「**Java コードを生成**」プレビュー・ページで、ウィザードが実行するアクションを検討して、「**完了**」をクリックします。

タスクの結果

生成されたファイルは、Java タイプ名でのパッケージ指定によって決まる、指定されたソース・コード・ディレクトリーのサブディレクトリーに書き込まれます。一部のタイプにパッケージが指定されていなければ、そのタイプ用の `.java` ソース・コード・ファイルが指定されたソース・コード・ディレクトリー内に生成されます。例えば、ソース・コード・ディレクトリーが `/projects/javaxom` と指定され、Java タイプ名が `com.example.import.Test` である場合、`Test.java` ファイルが `/projects/javaxom/com/example/import` ディレクトリー内に作成されます。ディレクトリーが存在する場合、ウィザードはそれを再使用します。ディレクトリーを作成しているとき、または `.java` ファイルをディレクトリーに書き込んでいるときに問題が検出されると、エラー・メッセージが表示されます。

ウィザード構成ファイル

ウィザード構成ファイルは、構成ウィザードがコネクター・ステージから起動されるときに、そのウィザードによって表示されるデフォルト値を指定するオプションを提供します。

目的

場合によっては、構成ウィザード設定のデフォルト値が、現在の環境で不便なことがあります。例えば、ウィザードが、主として、リンク上の列定義の Java コードを生成するために起動される場合は、「**アクションとログ・ファイルの設定ウィザード**」ページの「**Java コードを生成**」の方が、「**ルール・セット・パラメーターをインポート**」よりもデフォルト・オプションとして適切です。

場合によっては、デフォルト値が、現在のシステム環境に固有であることもあります。例えば、ウィザードが、主として、InfoSphere Information Server エンジン・ホストの同じディレクトリーに順に常にデプロイされるルール・セット・アーカイブ・ファイルに基づくステージを構成するために使用される場合、そのディレクトリーへのパスは、「**ルール・セットのフィルター式**」設定のデフォルト値として使用するのに適していると考えられます。

ウィザード設定のデフォルト値は、ウィザード構成ファイル `ccjrules.wizard.properties` の中のプロパティーとして指定できます。このファイルは、デフォルトでは、コネクター・ステージ・タイプがインストールされるときに作成されません。ウィザード構成ファイルを作成して、InfoSphere Information Server エンジン・ホストの *IS_HOME*`/ASBNode/lib/java` ディレクトリーにコピーできます。ここで、*IS_HOME* は、InfoSphere Information Server ホーム・ディレクトリーです。あるいは、そのディレクトリーを、ウィザードが起動されるステージの「**クラスパス**」プロパティー値に含めることができます。

『サンプル』セクションに、`ccjrules.wizard.properties` ファイルの内容の例を示しています。サンプル内容には、デフォルト値を指定できるすべてのウィザード設定が含まれます。これらの設定のデフォルト値を割り当てる行は、デフォルトで、すべて使用不可にされています。特定の設定を使用可能にするには、行の初めの `#` 文字を削除する必要があります。

例えば、ディレクトリー `C:/temp/javadoc` を、ウィザードが `.java` ソース・コード・ファイルを生成するデフォルト・ディレクトリーとして指定するには、次の行から、先頭の `#` 文字を削除する必要があります。

```
# source_code_directory = C:/temp/javacode
```

ウィザード構成の内容が変更されるたびに、新しい値を使用するために、ウィザードを閉じて、再度起動する必要があります。

サンプル

次に示したコード・サンプルをコピーして、独自のウィザード構成ファイルを作成できます。

```
#-------------------------------------------------------------------------
# デフォルトのウィザード・アクション:
# import_ruleset - ルール・セット・パラメーターをインポートする
```

```
# generate_code - Java コードを生成する
#-------------------------------------------------------------------------------
# wizard_action = import_ruleset
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# ログ・ファイルの場所。Windows も含め、すべてのプラットフォームで、
# ファイル分離文字としてスラッシュを使用します。
#-------------------------------------------------------------------------------
# log_file_path = C:/temp/ccjrules.log
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# ログ・ファイルの上書きのフラグ。有効な値は以下のとおりです。
# 0 - 上書きしない。ファイルが存在する場合はログを追加し、
#     それ以外の場合はファイルを作成します。
# 1 - ファイルが存在する場合はそれを上書きし、それ以外の場合はファイルを作成する
#-------------------------------------------------------------------------------
# overwrite_log_file = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# デフォルトのエンジン・モード。ここで値を指定すると、ウィザードが起動された
# ステージの中の「エンジン・モード」プロパティー値よりも、
# その値が優先されます。有効な値は以下のとおりです。
# core_engine - コア・エンジン
# j2se_res_xu - J2SE RES XU
# j2ee_res_xu - J2EE RES XU
#-------------------------------------------------------------------------------
# engine_mode = core_engine
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# ルール・セットを見つけるためのデフォルトのフィルター式
# 指定されると、ウィザードが起動されたステージの「ルール・セットの場所」
# プロパティーに値が指定されていない限り、その値が使用されます。その場合、
# 「ルール・セットの場所」値は、ウィザードの中でデフォルトのフィルター式として
# 使用されます。Windows も含め、すべてのプラットフォームで、
# ファイル分離文字としてスラッシュを使用します。
#-------------------------------------------------------------------------------
# filter_expression = C:/temp
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# 「すべてのルール・アプリケーションとルール・セットのバージョンを表示」の
# デフォルト設定。有効な値は以下のとおりです。
# 0 - すべてのルール・アプリケーションとルール・セットのバージョンを表示する
# 1 - 最新のルール・アプリケーションとルール・セットのバージョンのみを表示する
#-------------------------------------------------------------------------------
# show_all_versions = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# 「ルール・セット・パラメーター・クラスの継承されたメソッドを表示」の
# デフォルト設定。有効な値は以下のとおりです。
# 0 - 継承されたメソッドを含めない
# 1 - 継承されたメソッドを含める
#-------------------------------------------------------------------------------
# include_inherited_methods = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# ルール・セット・パラメーターをインポートするときの、リンク上の既存の列の
# 保持についてのデフォルトのインポート・モード。有効な値は以下のとおりです。
# merge - マッピングに既存の列を使用する
# replace - 既存の列をすべて置換する
#-------------------------------------------------------------------------------
```

```
# import_mode = merge
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# .java ファイルの生成先のディレクトリー。Windows も含め、
# すべてのプラットフォームで、ファイル分離文字としてスラッシュを使用します。
#-------------------------------------------------------------------------------
# source_code_directory = C:/temp/javacode
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# 生成されるソース・コードの出力フォーマット。有効な値は以下のとおりです。
# class - Java クラス
# inteface - Java インターフェース
#-------------------------------------------------------------------------------
# output_format = class
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# 生成されるコードの中のメンバー変数の接頭部
#-------------------------------------------------------------------------------
# member_variable_prefix = m_
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# .java コードを生成するときの、既存の .java ファイルの置換についての
# デフォルト設定。
# 0 - 既存のファイルを置換しない
# 1 - 既存のファイルを置換する
#-------------------------------------------------------------------------------
# replace_existing_files = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# .java コードを生成するときの、メソッドを個別に除外することについての
# デフォルト設定。
# 0 - メソッドを個別に除外することを使用不可にする
# 1 - メソッドを個別に除外のために使用するチェック・ボックスを使用可能にする
#-------------------------------------------------------------------------------
# exclude_individual_methods = 0
#-------------------------------------------------------------------------------
```

基本 Java タイプおよび基本 Java メソッド

ILOG JRules connector では、基本 Java タイプとは、直接リンク列にマップできる Java タイプのことです。

基本 Java タイプ

基本 Java タイプは次のとおりです。

- プリミティブ・タイプ: int、short、long、double、float、boolean、byte、char
- プリミティブ・タイプのラッパー・クラス: java.lang.Integer、 java.lang.Short、 java.lang.Long、 java.lang.Double、 java.lang.Float、 java.lang.Boolean、 java.lang.Byte、 java.lang.Character
- ストリング・タイプ: java.lang.String
- 日時タイプ: java.util.Date、 java.util.Calendar、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp
- 数値タイプ: java.math.BigInteger、java.math.BigDecimal

ルール・セット・パラメーターが XML パラメーターであるか、基本 Java タイプの Java パラメーターである場合、パラメーターは、次のように、リンクの列に直接マップされます。

- パラメーターが入力リンク上の列にマップされる場合、そのパラメーターは、IN または IN_OUT パラメーターでなければなりません。列は、ルール・セットが呼び出されるときに、パラメーターに渡される値を表します。
- パラメーターが出力リンク上の列にマップされる場合、そのパラメーターは、OUT または IN_OUT パラメーターでなければなりません。列は、ルール・セットが呼び出されるときに、パラメーターのルール・セットによって返される値を表します。

ルール・セット・パラメーターがリンク列に直接マップされない場合、リンク列にマップされるルール・セット構成は、ルール・セット・パラメーターに関連付けられた Java クラスの基本メソッドの戻り値および引数です。

基本 Java メソッド

基本 Java メソッドは次のいずれかの条件を満たします。

- コンストラクター・メソッドであるか、値を返さないメソッド (void メソッド) であり、そのすべての引数は基本 Java タイプである。メソッド引数は、メソッドが定義されたクラスを持つルール・セット・パラメーターに関連付けられた入力リンク列にマップされます。ルール・セット・パラメーターは、IN または IN_OUT パラメーターでなければなりません。ルール・セットを呼び出して、それにパラメーターを渡す前に、それぞれのルール・セット・パラメーター・オブジェクトにメソッドを呼び出すときに、列値は、メソッド引数に渡されます。
- 引数を含まないメソッドであり、その戻り値が基本 Java タイプである。メソッドの戻り値は、メソッドが定義されたクラス・タイプを持つルール・セット・パラメーターに関連付けられた出力リンク上の列にマップされます。ルール・セット・パラメーターは、OUT または IN_OUT パラメーターでなければなりません。パラメーターを含んだルール・セットが呼び出され、メソッドがそのルール・セット・パラメーター・オブジェクトで呼び出された後、列は、メソッドから取得される戻り値を表します。

構成ウィザードは、Java 配列タイプに関連付けられたルール・セット・パラメーターを、対応する非配列タイプであるかのように処理します。ジョブが実行されるとき、ステージは、たいていの場合、ルール・セット・パラメーター値を準備するために、リンク上の複数のレコードを収集します。そのルール・セット・パラメーターに関連付けられた、リンク上のデータ・レコードは、全体としてルール・セット・パラメーター値と対応する配列に保管される Java オブジェクトと対応します。しかし、ルール・セット・パラメーターが char[] および byte[] の Java 配列タイプである場合、それは、次のように、リンク上の単一レコードからの単一フィールド値に対応します。

- 列が、VarChar または NVarChar などの文字データ・タイプである場合、テキスト・フィールド値の文字は、ルール・セット・パラメーターの char[] 配列の char エレメントにマップされます。
- 列が、VarBinary などのバイナリー・データ・タイプである場合、対応するバイナリー・フィールド値のバイトは、byte[] 配列の byte エレメントにマップされます。

DataStage タイプから Java タイプへのマッピング

ウィザードは、Java コードを生成するときに、リンクの中の InfoSphere DataStage 列タイプ定義を、専用メンバー変数、メソッドの戻り値、およびメソッドの引数に使用される Java タイプにマップします。

実行時に、コネクターは、必要に応じて、入力リンク上にある InfoSphere DataStage 列タイプ定義を、これらのリンクに関連付けられた IN および IN_OUT パラメーターの値を初期化するために、Java タイプにマップします。次の表は、どのように InfoSphere DataStage タイプ定義が Java タイプにマップされるかを説明しています。

表 1. InfoSphere DataStage タイプから Java タイプへのマッピング

DataStage SQL タイプ	**長さ**	**スケール**	**拡張**	**Java タイプ**
TinyInt	(n/a)	(n/a)	空	byte、java.lang.Byte
TinyInt	(n/a)	(n/a)	符号なし	short、java.lang.Short
SmallInt	(n/a)	(n/a)	空	short、java.lang.Short
SmallInt	(n/a)	(n/a)	符号なし	int、java.lang.Integer
Integer	(n/a)	(n/a)	空	int、java.lang.Integer
Integer	(n/a)	(n/a)	符号なし	long、java.lang.Long
BigInt	(n/a)	(n/a)	空	long、java.lang.Long
BigInt	(n/a)	(n/a)	符号なし	java.math.BigInteger
Bit	(n/a)	(n/a)	(n/a)	boolean、 java.lang.Boolean
Float	(n/a)	(n/a)	(n/a)	float、java.lang.Float
Double	(n/a)	(n/a)	(n/a)	double、java.lang.Double
Decimal	任意	任意	(n/a)	java.math.BigDecimal
Char、NChar	1	(n/a)	任意	char、java.lang.Character
Char、NChar	n > 1	(n/a)	任意	char[]
Char、NChar	空	(n/a)	任意	char[]
VarChar、NVarChar、LongVarChar、LongNVarChar	任意	(n/a)	任意	java.lang.String
Binary	1	(n/a)	(n/a)	byte、java.lang.Byte
Binary	n > 1	(n/a)	(n/a)	byte[]
Binary	空	(n/a)	(n/a)	byte[]
VarBinary	任意	(n/a)	(n/a)	byte[]
Time	(n/a)	(n/a)	空	java.sql.Time
Time	(n/a)	(n/a)	マイクロ秒	java.sql.Timestamp
Date	(n/a)	(n/a)	(n/a)	java.sql.Date
Timestamp	(n/a)	(n/a)	空	java.util.Date
Timestamp	(n/a)	(n/a)	マイクロ秒	java.sql.Timestamp

InfoSphere DataStage タイプ定義によっては、複数の Java タイプにマップできるものもあります。このようなシナリオでは、「**NULL 可能**」属性に指定された値によって、2 つの Java タイプのどちらにマップされるかが決まります。「**NULL 可能**」値が「**いいえ**」に設定されている場合、プリミティブ Java タイプが選択されます。「**NULL 可能**」値が「**はい**」または「**不明**」に設定されている場合、オブジェクト・ラッパー・タイプが選択されます。例えば、「**符号付き**」属性が「**はい**」に設定された InfoSphere DataStage TinyInt タイプのリンク列は、「**NULL 可能**」属性が「**いいえ**」に設定されている場合は Java プリミティブ・タイプの `byte` にマップされ、「**NULL 可能性**」属性が「**はい**」に設定されている場合は Java ラッパー・タイプの `java.lang.Byte` にマップされます。

InfoSphere DataStage の文字ベースのデータ・タイプの Char、NChar、VarChar、LongVarChar、NVarChar、および LongNVarChar も、`boolean` および `java.lang.Boolean` の Java タイプにマップできます。テキスト値の「true」(大/小文字の区別はない) は、boolean 値の「*true*」にマップされます。残りのテキスト値は、boolean 値の「*false*」にマップされます。

Java タイプから DataStage タイプへのマッピング

構成ウィザードは、ルール・セット・パラメーターが使用する Java タイプを、ステージのリンク上の対応する列定義にマップします。

実行時に、コネクターは、必要に応じて、出力リンクに関連付けられた OUT および IN_OUT ルール・セット・パラメーター内の Java タイプを、それらのリンク上の列の InfoSphere DataStage タイプ定義にマップします。

表 2. Java タイプから InfoSphere DataStage タイプへのマッピング

Java タイプ	DataStage SQL タイプ	長さ	スケール	拡張
short、java.lang.Short	SmallInt	(n/a)	(n/a)	空
int、java.lang.Integer	Integer	(n/a)	(n/a)	空
long、java.lang.Long	BigInt	(n/a)	(n/a)	空
float、java.lang.Float	Float	(n/a)	(n/a)	(n/a)
double、java.lang.Double	Double	(n/a)	(n/a)	(n/a)
boolean、java.lang.Boolean	Bit	(n/a)	(n/a)	(n/a)
byte、java.lang.Byte	TinyInt	(n/a)	(n/a)	空
byte[]、 java.lang.Byte[]	VarBinary	空	(n/a)	(n/a)
char、java.lang.Character	Char	1	(n/a)	Unicode
char[]、java.lang.Character[]	VarChar	空	(n/a)	Unicode
java.lang.String	VarChar	空	(n/a)	Unicode
java.util.Date、java.util.Calendar	Timestamp	(n/a)	(n/a)	マイクロ秒
java.sql.Date	Date	(n/a)	(n/a)	(n/a)
java.sql.Time	Time	(n/a)	(n/a)	マイクロ秒
java.sql.Timestamp	Timestamp	(n/a)	(n/a)	マイクロ秒

表 2. *Java タイプから InfoSphere DataStage タイプへのマッピング (続き)*

Java タイプ	**DataStage SQL タイプ**	**長さ**	**スケール**	**拡張**
java.math.BigInteger	BigInt	(n/a)	(n/a)	空
java.math.BigDecimal	Decimal	255	127	(n/a)

int、short、long、double、float、boolean、byte、および char の Java タイプに対応するリンク列は、「**NULL 可能**」属性が「**いいえ**」に設定されて構成されます。char[] および byte[] タイプも含めて、残りの Java タイプに対応する列は、「**NULL 可能**」属性が「**はい**」に設定されて構成されます。

`boolean` および `java.lang.Boolean` の Java タイプもまた、文字ベースの InfoSphere DataStage データ・タイプである、Char、 NChar、 VarChar、 LongVarChar、 NVarChar、および LongNVarChar にマップできます。 boolean 値の true は、テキスト値 *true* にマップされ、boolean 値の false は、テキスト値 *false* にマップされます。

付録 A. 製品のアクセシビリティー

IBM 製品のアクセシビリティー対応状況についての情報を入手できます。

IBM InfoSphere Information Server 製品のモジュールおよびユーザー・インターフェースは完全にはアクセシビリティー対応がなされていません。

IBM 製品のアクセシビリティー対応状況の詳細は、http://www.ibm.com/able/product_accessibility/index.html の IBM 製品のアクセシビリティー情報をご覧ください。

アクセシビリティー対応資料

IBM Knowledge Center には、製品のアクセシビリティー対応資料が用意されています。IBM Knowledge Center では、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で資料を提供しています。IBM Knowledge Center では XHTML を使用しているため、使用しているブラウザーに設定されている表示形式で資料を表示できます。さらに、スクリーン・リーダーやその他の支援技術を使用して、資料にアクセスすることもできます。

IBM Knowledge Center にある資料は、PDF ファイルでも提供されますが、こちらは完全にはアクセシビリティー対応がなされていません。

IBM のアクセシビリティーに対する取り組み

アクセシビリティーに関する IBM のコミットメントの詳細については、 IBM Human Ability and Accessibility Center を参照してください。

付録 B. コマンド・ライン構文の読み方

この資料では、特殊文字を使用してコマンド・ライン構文を定義しています。

次の特殊文字によってコマンド・ライン構文が定義されます。

[ ] オプションの引数を識別します。 大括弧で囲まれていない引数は必須です。

... 前の引数に複数の値を指定できることを示します。

| 同時には使用できない情報であることを示します。 区切り文字の左側の引数か、右側の引数のどちらか一方を使用できます。単一のコマンド使用で、両方の引数を使用することはできません。

{ } 同時には使用できない一連の引数を囲みます。この内の 1 つは必須です。引数がオプションの場合、引数は大括弧 ([ ]) で囲まれます。

注:

- 引数の最大文字数は 256 です。
- 埋め込みのスペースがある引数値は、単一引用符または二重引用符で囲みます。

例:

wsetsrc*[-S server] [-l label] [-n name] source*

source 引数は、**wsetsrc** コマンドで唯一必須の引数です。 他の引数は大括弧で囲まれています。これは、これらの引数がオプションであることを示します。

wlsac *[-l | -f format] [key...] profile*

この例で、-l および -f format 引数は、同時には使用できないもので、オプションでもあります。 *profile* 引数は必須です。 *key* 引数はオプションです。 *key* 引数のあとの省略符号 (...) は、複数の key 名を指定できることを示します。

wrb -import *{rule_pack | rule_set}...*

この例で、rule_pack および rule_set 引数は同時には使用できませんが、どちらか 1 つの引数は指定する必要があります。また、省略符号 (...) は、複数の rule_pack または rule_set を指定できることを示します。

付録 C. 構文図の見方

本書で使用される構文図には、以下の規則が適用されます。

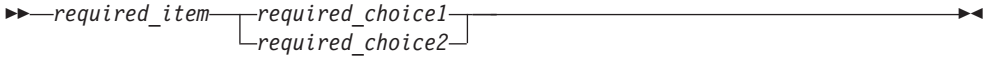
- 構文図は、左から右、上から下に、線に沿って読みます。以下の規則が使用されます。
  - >>--- 記号は、構文図の始まりを示します。
  - ---> 記号は、構文図が次の行に続くことを示します。
  - >--- 記号は、構文図が前の行から続いていることを示します。
  - --->< 記号は、構文図の終わりを示します。
- 必須項目は、水平線 (メインパス) 上にあります。

- オプション項目はメインパスの下に表示されます。

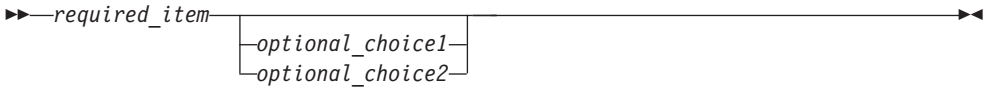
オプション項目がメインパスの上に表示される場合、その項目は構文要素の実行に影響せず、単に読みやすくするために使用されます。

- 複数の項目から選択できる場合は、それらの項目を縦に並べて (スタック) 示しています。

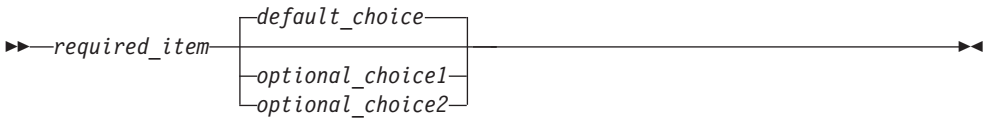
項目の 1 つを選択する必要がある場合は、スタックの 1 つの項目がメインパス上に示されています。



項目から 1 つをオプションで選択できる場合、スタック全体がメインパスよりも下に示されます。



項目の 1 つがデフォルトである場合は、その項目はメインパスの上に示され、残りの選択項目は下に示されます。



- メインラインの上に、左へ戻る矢印がある場合には、項目を繰り返して指定できることを示しています。

繰り返しの矢印にコンマが含まれている場合は、繰り返し項目をコンマで区切らなければなりません。

スタックの上の反復矢印は、スタック内の項目を反復できることを示します。

- 構文図が、複数のフラグメントに分かれている場合があります。構文フラグメントはメインの構文図とは別に示されますが、フラグメントの内容は、図のメインパス上にあるものとして読む必要があります。

fragment-name:

- キーワードは大文字で表示され、最小の省略形が存在する場合にはそれも大文字で表示されます。示されているとおりに入力する必要があります。
- 変数は、すべて小文字のイタリック体 (例えば、**column-name**) で表示されます。これらは、ユーザーが指定する名前または値を表します。
- 図の中に句読点がない場合は、キーワードおよびパラメーターを 1 つ以上のスペースで区切ります。
- 句読記号、括弧、算術演算子、およびその他の記号は、図に示されているとおりに入力してください。
- 脚注は、(1) のように、括弧の中に数字を入れた形で示されます。

付録 D. IBM の窓口

お客様サポート、ソフトウェア・サービス、製品情報、および全般情報について、IBM と連絡を取ることができます。 また、製品についてのフィードバックを行うことができます。

次の表に、お客様サポート、ソフトウェア・サービス、研修、製品およびソリューション情報に関するリソースをリストしています。

表 3. IBM リソース

リソース	説明と場所
IBM サポート・ポータル	サポート情報は、www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server で、製品と関心のあるトピックを選択してカスタマイズできます。
ソフトウェア・サービス	ソフトウェア、IT、およびビジネス・コンサルティング・サービスについての情報は、「ソリューション」サイト www.ibm.com/businesssolutions/jp/ja にアクセスしてください。
My IBM	www.ibm.com/account/jp/ja/ の「My IBM」サイトでアカウントを作成し、特定のテクニカル・サポートのニーズに合うように、IBM Web サイトおよび情報へのリンクを管理できます。
研修と認定	個人、法人、および公共団体向けに、IT 技術の習得、維持、最適化を目的としてデザインされた技術研修およびサービスについては、http://www.ibm.com/training にアクセスしてください。
IBM 担当員	ソリューションについて IBM 担当員と連絡を取るには、www.ibm.com/connect/ibm/us/en/ にアクセスしてください。

付録 E. 製品資料へのアクセス

資料は、オンラインの IBM Knowledge Center、オプションでローカルにインストールしたインフォメーション・センター、PDF のブックといったさまざまな形式で提供されます。製品クライアント・インターフェースから、オンラインまたはローカルにインストールしたヘルプに直接アクセスすることができます。

IBM Knowledge Center は、InfoSphere Information Server の最新情報を探すのに最適な場所です。 IBM Knowledge Center には、スイートのすべての製品モジュールの全資料のほか、ほとんどの製品インターフェースのヘルプも含まれています。IBM Knowledge Center は、インストール済み製品から開くことも、Web ブラウザーから開くこともできます。

IBM Knowledge Center へのアクセス

オンライン資料にアクセスするには、さまざまな方法があります。

- クライアント・インターフェースで、画面右上の「**ヘルプ**」リンクをクリックします。
- F1 キーを押します。 F1 キーを押すと、通常、クライアント・インターフェースの現行コンテキストを説明するトピックが開きます。

  注: F1 キーは、Web クライアントでは機能しません。
- 製品にログインしていないときなどに、Web ブラウザーにアドレスを入力します。

  すべてのバージョンの InfoSphere Information Server の資料にアクセスするには、以下のアドレスを入力します。

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ/
  ```

  特定のトピックにアクセスするには、製品 ID とバージョン番号、資料プラグイン名、および URL 内のトピック・パスを指定します。例えば、バージョン 11.3 用のこのトピックの URL は以下のとおりです。 (記号「⇒」は、行の継続を表します)

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/⇒
  com.ibm.swg.im.iis.common.doc/common/accessingiidoc.html
  ```

  ヒント:

  Knowledge Center には、以下の短縮 URL もあります。

  ```
  http://ibm.biz/knowctr
  ```

  特定の製品ページ、バージョン、またはトピックの短縮 URL を指定するには、短縮 URL と製品 ID の間にハッシュ文字 (#) を使用します。例えば、すべての InfoSphere Information Server 資料の短縮 URL は、以下のとおりです。

  ```
  http://ibm.biz/knowctr#SSZJPZ/
  ```

また、前述のトピックの URL を少し短くした短縮 URL は、以下のとおりです。(記号「⇒」は、行の継続を表します)

```
http://ibm.biz/knowctr#SSZJPZ_11.3.0/com.ibm.swg.im.iis.common.doc/⇒
common/accessingiidoc.html
```

ローカルにインストールした資料を参照するヘルプ・リンクの変更

IBM Knowledge Center には、最新版の資料が含まれています。一方、インフォメーション・センターとしてローカル版の資料をインストールして、それを指すようにヘルプ・リンクを構成することも可能です。ローカルのインフォメーション・センターは、お客様の企業でインターネットへのアクセスが提供されていない場合に便利です。

インフォメーション・センターのインストール・パッケージに付属するインストール手順を使用して、任意のコンピューターにそれをインストールします。インフォメーション・センターをインストールして開始した後、サービス層のコンピューターで **iisAdmin** コマンドを使用して、製品の F1 とヘルプ・リンクで参照する資料の場所を変更できます。 (記号「⇒」は、行の継続を表します)

Windows

```
IS_install_path¥ASBServer¥bin¥iisAdmin.bat -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

AIX® Linux

```
IS_install_path/ASBServer/bin/iisAdmin.sh -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

ここで、 `<host>` はインフォメーション・センターがインストールされたコンピューターの名前、 `<port>` はインフォメーション・センターのポート番号です。デフォルトのポート番号は 8888 です。例えば、デフォルト・ポートを使用するコンピューター `server1.example.com` 上の URL 値は、`http://server1.example.com:8888/help/topic/` になります。

PDF およびハードコピー資料の入手

- PDF ファイルのブックはオンラインで利用可能で、サポートの文書 https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1 からアクセスできます。
- IBM 資料は、オンラインでダウンロード、または IBM 担当員を通じてご注文いただけます。 資料をオンラインでダウンロードするには http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss の IBM Publications Center にアクセスしてください。

付録 F. 製品資料に関するフィードバックの提供

IBM の資料に関する貴重なフィードバックをご提供ください。

お客様からのご意見やご感想は、IBM が質の高い情報を提供するための参考にさせていただきます。ご意見をお寄せいただく場合は、次のいずれかの方法を使用することができます。

- IBM の Web サイトでホストしている IBM Knowledge Center 内のトピックについてコメントをお寄せいただくには、サインインし、トピックの下の「**コメントの追加**」ボタンをクリックしてコメントを追加してください。このようにして送信されたコメントは、一般に公開されます。
- IBM Knowledge Center 内のトピックに関するコメントを IBM に送信し、他の人からは閲覧できないようにするには、サインインし、IBM Knowledge Center の下の「**フィードバック**」リンクをクリックしてください。
- オンライン・リーダー用のコメント・フォーム (www.ibm.com/software/awdtools/rcf/) を使用して、コメントを送信します。
- コメントを E メールで comments@us.ibm.com に送付します。お送りいただく情報には、製品の名前、製品のバージョン番号、資料の名前と部品番号 (該当する場合) を含めてください。特定のテキストについてご意見がある場合は、そのテキストの位置 (例えば、タイトル、表番号、ページ番号など) を記載してください。

特記事項および商標

本書は米国 IBM が提供する製品およびサービスについて作成したものです。この資料は、IBM から他の言語でも提供されている可能性があります。ただし、ご利用にはその言語版の製品もしくは製品のコピーを所有していることが必要な場合があります。

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、さまざまなオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されて

います。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。© Copyright IBM Corp. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プライバシー・ポリシーに関する考慮事項

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項を確認ください。

このソフトウェア・オファリングは、展開される構成に応じて、セッションごとの Cookie または永続的な Cookie を使用する場合があります。製品またはコンポーネントがリストされていない場合、その製品またはコンポーネントは Cookie を使用しません。

表 4. InfoSphere Information Server 製品およびコンポーネントによる Cookie の使用

製品モジュール	コンポーネントまたは機能	使用される Cookie の種類	収集するデータ	データの目的	Cookie の無効化
すべて (InfoSphere Information Server インストール済み環境の部分)	InfoSphere Information Server Web コンソール	• セッション • 永続	ユーザー名	• セッション管理 • 認証	無効にできない

表 4. *InfoSphere Information Server* 製品およびコンポーネントによる *Cookie* の使用 (続き)

製品モジュール	コンポーネントまたは機能	使用される Cookie の種類	収集するデータ	データの目的	Cookie の無効化
すべて (InfoSphere Information Server インストール済み環境の部分)	InfoSphere Metadata Asset Manager	• セッション • 永続	個人情報でない	• セッション管理 • 認証 • 拡張されたユーザーのユーザビリティー • シングル・サインオン構成	無効にできない
InfoSphere DataStage	Big Data File ステージ	• セッション • 永続	• ユーザー名 • デジタル署名 • セッション ID	• セッション管理 • 認証 • シングル・サインオン構成	無効にできない
InfoSphere DataStage	XML ステージ	セッション	内部 ID	• セッション管理 • 認証	無効にできない
InfoSphere DataStage	IBM InfoSphere DataStage and QualityStage Operations Console	セッション	個人情報でない	• セッション管理 • 認証	無効にできない
InfoSphere Data Click	InfoSphere Information Server Web コンソール	• セッション • 永続	ユーザー名	• セッション管理 • 認証	無効にできない
InfoSphere Data Quality Console		セッション	個人情報でない	• セッション管理 • 認証 • シングル・サインオン構成	無効にできない
InfoSphere QualityStage Standardization Rules Designer	InfoSphere Information Server Web コンソール	• セッション • 永続	ユーザー名	• セッション管理 • 認証	無効にできない
InfoSphere Information Governance Catalog		• セッション • 永続	• ユーザー名 • 内部 ID • ツリーの状態	• セッション管理 • 認証 • シングル・サインオン構成	無効にできない
InfoSphere Information Analyzer	InfoSphere DataStage and QualityStage Designer クライアントの中の Data Rules ステージ	セッション	セッション ID	セッション管理	無効にできない

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライ

ン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含むさまざまなテクノロジーの使用の詳細については、IBM の『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』 (http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

Adobe は、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Intel、Itanium は、 Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

Microsoft、Windows および Windows NT は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Java™ およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ラ行]

B

D

I

J

N

W

X

Printed in Japan

SC43-0989-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21