Oracle® Real-Time Decisions

Decision Studio
リファレンス・ガイド

ORACLE

リリース 2.2　　2007 年 5 月

Oracle Real-Time Decisions Decision Studio リファレンス・ガイド, リリース 2.2

部品番号: E05257-01

原本名: Oracle Real-Time Decisions Decision Studio Reference Guide, Version 2.2

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Siebel は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

Decision Studio リファレンス・ガイド

はじめに

Decision Studio はインライン・サービスを定義および管理するためのツールです。Decision Studio にはインライン・サービスのすべての機能が公開されます。Decision Studio は、Java の基本的な知識があり、アプリケーション開発およびライフサイクルの問題を総合的に理解する IT プロフェッショナルを対象ユーザーとしています。Decision Studio の使用方法の詳細は、『Oracle Real-Time Decisions スタート・ガイド』を参照してください。

Decision Studio は、統合開発環境（IDE）パラダイムに準拠したリッチクライアント・アプリケーションです。Decision Studio の左側には「Inline Service Explorer」ビュー、右側にはエディタ・ビューがあります。このナビゲータ・ビューには、事前定義されたインライン・サービスのフォルダ構造が表示されます。各フォルダ内のアイテムは、インライン・サービスのメタデータ要素になります。メタデータ要素は、Decision Studio を使用して追加、編集および削除できます。メタデータ要素をダブルクリックすると、その要素の詳細情報がオブジェクト・エディタに表示されます。メタデータ要素のタイプごとに独自のエディタがあります。要素はまず XML メタデータとして示されます。その後、Java クラスが生成され、実行するインライン・サービスがここからコンパイルされます。

Decision Studio は、Eclipse IDE に基づいています。これにより、インライン・サービスの管理に固有の機能を、汎用的な Java 開発ツールを備え、ソフトウェア構成管理（SCM）システムを統合した Eclipse IDE の機能と組み合せることができます。

このガイドでは、Decision Studio を使用してインライン・サービスを開発するために必要な概念、コンポーネントおよび API について詳細に説明します。

このドキュメントについて

このドキュメントは、Decision Studio のリファレンスです。各インライン・サービスのプロパティおよび使用可能な API を含め、インライン・サービスの構成に使用する要素について個別に説明します。

対象読者

このドキュメントは、Decision Studio を使用してインライン・サービスを構成するテクニカル・ユーザーを対象としています。ユーザーには、Java およびソフトウェア開発ライフサイクルに関する基本的な知識が必要です。

このドキュメントの構成

このドキュメントの内容は次のとおりです。

第1章「Decision Studio について」では、プラットフォームの概要について説明しています。

第2章「Decision Studio の要素と API について」では、システムの各要素およびコール可能な API について説明しています。

第3章「Oracle RTD の一般的な API」では、Oracle RTD の一般的な API を示しています。

第4章「インライン・サービスのデプロイとテスト」では、テスト機能について概要を示し、Real-Time Decision Server へのデプロイおよび再デプロイについても説明しています。

第5章「インライン・サービスのトラブルシューティングとデバッグ」では、インライン・サービスをデバッグするときのヒントを示しています。

表記規則

規則	説明
固定幅 フォント	ソース・コードおよびプログラム出力を示します。
太字	ラベル、タブ、メニューなどのユーザー・インタフェースを示します。
	タスクの実行に役立つ追加情報を示します。
	その項目に関する追加情報を示します。
	データの損失またはエラーが生じる可能性のあるアクションを示します。

1 Decision Studio について

この章では、Decision Studio の概要について説明します。

この章の内容は次のとおりです。

- 1.1 インライン・サービスについて
- 1.2 Decision Studio と Eclipse
- 1.3 Decision Studio のプロジェクトの概要
- 1.4 インライン・サービスのディレクトリ構造
- 1.5 インライン・サービスの構成

1.1 インライン・サービスについて

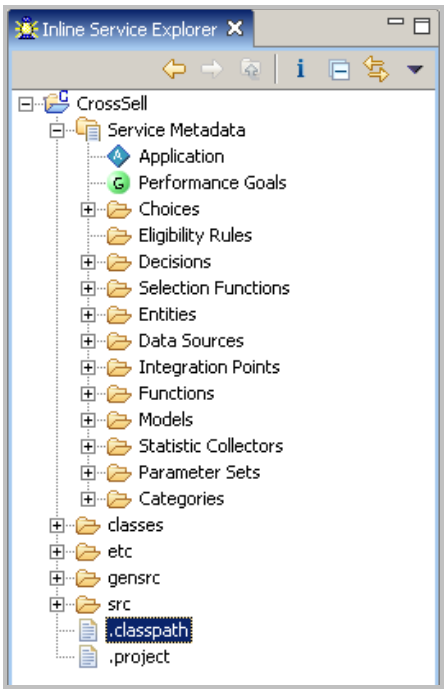
インライン・サービスは、企業全体のビジネス・プロセスにおける重要な局面での、リアルタイムかつ継続的な監視およびアドバイスを実行するデプロイ可能なアプリケーションです。インライン・サービスはビジネス・プロセス全体を追従するのではなく、プロセス内の特定の識別ポイントに焦点を当てます。インライン・サービスは、Decision Studio を使用して構成およびデプロイされ、Decision Center を使用して分析および調整されます。インライン・サービスの実行は Real-Time Decision Server で行われます。Oracle RTD は、これらすべてのコンポーネントを使用して構成されます。

1.2 Decision Studio と Eclipse

Decision Studio は、Eclipse Foundation 社により開発されたオープン・ソース Java IDE である Eclipse に基づいています。Decision Studio は Eclipse 環境に対する標準プラグインとして存在します。Eclipse を使用している場合は、この環境を利用して、追加の開発機能や高度な機能を使用できます。Eclipse に精通していない場合でも、Studio を完全に透過的に使用できます。

1.2.1 「Inline Service Explorer」について

「Inline Service Explorer」を使用すると、インライン・サービス・プロジェクトのすべての側面にアクセスできます。一般的なインライン・サービス・プロジェクトを次に示します。



各フォルダの内容は次のとおりです。

Service Metadata	インライン・サービスを形成するメタデータ。このファイル・タイプのデフォルトのエディタは各要素に固有です。メタデータはテキスト・エディタで編集することもできますが、この方法はお薦めしません。
classes	コンパイル処理で生成されたクラス。
etc	このディレクトリには、システム管理に使用する様々なスクリプトおよびファイルが含まれます。Load Generator スクリプトが構築されると、規定によりこのフォルダに保存されます。
gensrc	インライン・サービス用に生成されたソース・コード・ファイル。
src	カスタム Java コード。ユーザーが作成した任意の Java クラスが含まれる場合があります。これらのクラスの一部を使用して、生成済インライン・サービス Java クラスのデフォルト動作を上書きすることもできます。
lib	`lib` フォルダは、外側クラスを使用する場合にユーザーによってオプションで作成されます。たとえば、ある関数、ロジックまたは初期化ブロック内で `THashMap` というクラスにアクセスするとします。このクラスは `tcollections.jar` ファイル内にあります。このクラスを使用するには、プロジェクト・ワークスペースのプロジェクト・ディレクトリの下に `lib` フォルダを作成して、このフォルダに `tcollections.jar` ファイルを置きます。この jar のクラスを使用するには、説明の横にある「**Advanced**」ボタンを使用してインポートし、このクラスをコード内で使用します。

.classpath	プロジェクトの Java クラスパスを含むファイル。このファイルを編集する必要はありません。
.project	Eclipse プロジェクト・ファイル。

1.2.2 コード生成

通常、インライン・サービスの要素を構成すると、次の 4 つのファイルが生成されます。

- 構成をメタデータとして格納する.sda ファイル。
- メタデータから生成され、クラス・ファイルにコンパイルされる.java ファイル。
- 元の生成済ファイルを拡張して、特殊な環境で生成済ファイルの動作を上書きするために使用される.java ファイル。
- 最初は生成済ファイルからコンパイルされ、その後、任意の上書きファイルからコンパイルされるクラス・ファイル。

それぞれのファイルの名前は次のようになります。

- **メタデータ・ファイル**: *Object_ID*.sda
- **生成済ファイル**: GEN*Object_ID*.java
- **上書きファイル**: *Object_ID*.java
- **クラス・ファイル**: *Object_ID*.class
- **生成済クラス・ファイル**: GEN*Object_ID*.class

ヒント: オブジェクト ID は、オブジェクトに付けられた名前と同様に作成されます。オブジェクト ID は、作成されたオブジェクト名の下にあるテキスト・ボックスに表示されます。オブジェクト ID は Java 標準に準拠するため、ラベルと異なる場合があります。オブジェクト ID を調べるには、  ツールバー・ボタンを使用して、ラベルの表示とオブジェクト ID の表示を切り替えます。

たとえば、**Customer account** という名前の要素があるとします。このオブジェクト ID は、Java 命名規約に準拠して **CustomerAccount** という形式になります。

作成されるファイルは次のとおりです。

- **メタデータ・ファイル**: CustomerAccount.sda
- **生成済ファイル**: GENCustomerAccount.java
- **上書きファイル**: CustomerAccount.java
- **クラス・ファイル**: CustomerAccount.class
- **生成済クラス・ファイル**: GENCustomerAccount.class

1.2.3 Decision Studio のパースペクティブおよびビューについて

Decision Studio を使用すると、インライン・サービスを様々なパースペクティブから操作できます。パースペクティブは、そのパースペクティブのビューやエディタの初期設定およびレイアウトを定義します。パースペクティブごとに、特定のタイプのタスク達成を目的とした一連の機能があり、特定のタイプのリソースに対応します。パースペクティブによって、一部のメニューおよびツールバーの表示内容が制御されます。

デフォルトのインライン・サービス・パースペクティブには、次の 4 つのビューがあります。

- **「Inline Service Explorer」ビュー**: ツリー形式でプロジェクトおよび要素が表示されます。デフォルトでは、画面の左側に配置されます。
- **「Problem」ビュー**: プロジェクトに関するエラーおよび例外が表示されます。デフォルトでは画面の底部に配置され、このビューと「Test」ビューのいずれかがタブで選択されます。
- **「Test」ビュー**: インライン・サービスのテスト用の領域が提供されます。デフォルトでは画面の底部に配置され、このビューと「Problem」ビューのいずれかがタブで選択されます。
- **「Cheat Sheets」ビュー**: 一般的なタスクの手順が表示されます。デフォルトでは、画面の右側に配置されます。

インライン・サービス・パースペクティブの中央の領域がエディタ領域です。ここには、プロジェクト・ツリーで選択したノードに固有のエディタが表示されます。新しいエディタに変更するには、編集する要素をダブルクリックします。

Java ファイルを編集するには、Java パースペクティブに変更して、編集する Java ファイルをダブルクリックします。

インライン・サービス・パースペクティブはデフォルトのパースペクティブであり、インライン・サービスを構成およびデプロイするためのメインの作業領域となります。Oracle RTD には、インライン・サービス・メタデータを操作するための多くの機能があります。これらについては、後述の各項で説明します。ここに記載のない機能は、コアの Eclipse プラットフォームの一部になります。それらの機能の詳細は、Eclipse オンライン・ヘルプ・ドキュメント『Workbench User Guide』を参照してください。

インライン・サービス・パースペクティブには、次のメニューおよびツールバー・アイテムがあります。

「File」→「New Inline Service Project」	選択したワークスペースに新しいインライン・サービス・プロジェクトを作成します。
「Project」→「Download」	変更するために、デプロイ済のインライン・サービスを Real-Time Decision Server からダウンロードします。
「Project」→「Deploy」	インライン・サービスを Real-Time Decision Server にデプロイします。
「Window」→「Open Perspective」→「Inline Service Perspective」	インライン・サービス・パースペクティブを開きます。

「Window」→「Show View」→「Inline Service Explorer View」		現在のインライン・サービスのビューを表示します。
「Window」→「Dispay Object IDs」		ラベルとオブジェクト ID の表示を切り替えます。
「Help」→「About」		Decision Studio のバージョン情報を表示します。
	**「Deploy」**	インライン・サービスを Real-Time Decision Server にデプロイします。
	**「Download」**	変更するために、デプロイ済のインライン・サービスを Real-Time Decision Server からダウンロードします。
	**「Object ID」 トグル**	ラベルとオブジェクト ID の表示を切り替えます。

「Inline Service Explorer」ビューには、次のツールバー・アイテムがあります。

	**「Metadata」 トグル**	プロジェクト・ツリー全体とインライン・サービス・メタデータのみの表示を切り替えます。
	**「Collapse All」**	プロジェクト・ツリーを閉じます。
	**「Link with editor」**	要素の選択に応じてエディタが調整されるよう、選択した要素タイプとリンクに適切なエディタを探します。
	**「Menu」**	「**Link with editor**」および「**Always Show Object IDs**」へのアクセスを提供します。
	「Always Show Object IDs」	「Inline Service Explorer」とエディタで、要素のオブジェクト ID とラベルの両方を表示します。

Java パースペクティブでは、Java ソース・ファイルの編集時に通常使用するビューを組み合せます。一方、デバッグ・パースペクティブには、Java プログラムのデバッグ時に使用するビューが含まれます。

生成済 Java コードを直接操作するには、Java パースペクティブを使用します。Java コード・レベルでインライン・サービスをデバッグするには、デバッグ・パースペクティブを使用します。

1.2.4 ビューの配置とエディタのサイズ変更

エディタ領域のタブには、現在編集のために開かれているリソース名が表示されます。アスタリスク（*）は、変更がまだ保存されていないことを示します。ビューのタブにはビューの名前が表示され、ビューに固有の機能を提供するツールバーがあります。

パースペクティブのビューおよびエディタは、画面上の任意の領域にドラッグ・アンド・ドロップできます。ビューおよびエディタのサイズは、配置される領域に合せて自動的に変更されます。（主な作業場所である）エディタまたはビューの一部が、他のビューに覆われたり、使用するには不便な領域にサイズ変更される場合があります。エディタまたはビューのサイズを変更するには、開いている別のビューを閉じて残りが自動的にサイズ変更されるようにするか、エディタまたはビューを最大化します。

エディタとビューはどちらも、タブをダブルクリックするか、右クリックして表示されるメニュー・アイテムを使用して、最大化と最小化を切り替えることができます。パースペクティブ、エディタおよびビューの詳細は、Eclipse オンライン・ヘルプに含まれる『Workbench User Guide』のオンライン・ドキュメントを参照してください。

1.2.5 要素のロジックについて

Java コードは、Decision Studio 内の要素のロジック・パネルに追加されます。このコードは、`GEN`*`Object_ID`*`.java` ファイルの適切なメソッドに挿入されます。ロジックを要素に追加したり更新したりする場合は、要素を選択し、エディタを使用してロジック・パネルのコードを変更します。

生成済コード内に直接大きなコードの断片を挿入する方が便利な場合もあります。これらのファイルは、Decision Studio の Java パースペクティブを使用して直接編集できます。生成済コードは、特定の場所でしか手動で編集できないので注意してください。また、「**Project**」→「**Clean**」を選択すると、Decision Studio で生成済コードが再生成され、生成済 Java コードに直接行ったコード変更が上書きされます。

Decision Studio の Java パースペクティブを使用して編集可能な任意のメソッドは、Start および End マーカーで明示的に示されています。たとえば、Application オブジェクトにはインライン・サービスを初期化する `init()` というメソッドがあります。

このメソッドのコードは、Decision Studio インタフェースで、アプリケーション要素の「**Logic**」タブの「**Initialization Logic**」を使用して追加できます。

かわりに Eclipse を使用して初期化コードを直接 Application クラスに追加する場合は、次のようにマークされたメソッドにのみ追加します。

```
public void init() {

    // SDCUSTOMCODESTART.Application.InitBody.java.0

    // SDCUSTOMCODEEND.Application.InitBody.java.0

  }
```

コードは、メソッドの開始コメントと終了コメントの間に記述する必要があります。コメント化された領域外にコードを記述すると、上書きされるおそれがあります。生成済 Java ファイルに直接追加されたコードは、ファイルの再生成時に失われます。コードを保持するには、対応するメタデータ要素にそれをコピーする必要があります。

1.2.6 生成済コードの上書き

生成済クラス *Object_ID*.java は、クラス GEN*Object_ID*.java を拡張します。なんらかの理由で GEN*Object_ID*.java に含まれるコードを上書きする必要がある場合は、上書きするコードを *Object_ID*.java ファイルに追加します。このファイルは、gensrc ディレクトリから src ディレクトリに移動する必要があります。

1.3 Decision Studio のプロジェクトの概要

Studio のインライン・サービスは、Decision Studio のプロジェクトとして構築されます。

1.3.1 新規プロジェクトの開始

新しいインライン・サービスを開始するには、「**File**」→「**New Inline Service Project**」メニューを使用して、プロジェクトを開始します。リストからテンプレートを選択してプロジェクトの名前を指定し、「**Finish**」をクリックしてプロジェクトを作成します。

テンプレートのリストには、Oracle RTD インストールで提供されるテンプレートと任意のユーザー定義テンプレートが含まれます。

1.3.2 プロジェクトのインポート

既存のプロジェクトを開く場合は、「**File**」→「**Import**」メニューを使用してプロジェクトをインポートします。メタデータを以前のバージョンから更新する必要がある場合は、アップグレードするよう求められます。

1.3.3 ユーザー定義テンプレートの作成

インライン・サービスからテンプレートを作成するには、「**File**」→「**Export**」を使用してプロジェクトをテンプレートにエクスポートします。エクスポート・タイプは「**Inline Service Template**」を選択します。テンプレートは、インライン・サービスのプリファレンスで定義された場所に保存されます。プリファレンスにアクセスするには、「**Window**」→「**Preferences**」を使用して「**Inline Services**」を選択します。入力されたディレクトリは、ファイル・システム上でテンプレートが保存される場所になります。

1.3.4 デプロイされているインライン・サービスのダウンロード

デプロイされているインライン・サービスをダウンロードするには、「**Project**」→「**Download**」を使用します。また、ツールバーのダウンロード・アイコンを使用して Real-Time Decision Server からダウンロードすることもできます。デプロイされているインライン・サービスを変更する場合は、その変更と、ビジネス・ユーザーにより行われている可能性のある変更の両方が失われないよう、次の手順に従うことが重要です。次の方法を使用します。

1. デプロイされている目的のインライン・サービスを編集しているビジネス・ユーザーがいないことを確認します。
2. ダウンロードする場合は必ずインライン・サービスをロックして、編集中にビジネス・ユーザーが変更できないようにします。
3. Decision Studio で変更作業を行います。

4　インライン・サービスを再デプロイしてロックを解除します。

インライン・サービスがロックされている間、ビジネス・ユーザーはそのデプロイされているインライン・サービスを表示できますが、編集はできません。

1.3.5　デプロイの状態について

インライン・サービスを Decision Studio からデプロイする場合は、デプロイ・ダイアログでデプロイの状態を選択します。Decision Studio には、「Development」、「QA」および「Deployment」の 3 つのデプロイの状態があります。システム管理者は JConsole を使用して、さらにデプロイの状態を追加できます。

テスト・ビューでインライン・サービスをテストする場合は、最後のデプロイの状態がテストされます。

1.3.6　サンプル・プロジェクト

サンプル・プロジェクトは *RTD_HOME*¥examples ディレクトリにあり、インポートして使用できます。このディレクトリには、『Oracle Real-Time Decisions 統合ガイド』でサンプルとして使用されている Cross Sell インライン・サービスが含まれています。

Cross Sell インライン・サービスは、クレジット・カードのコンタクト・センターにおける簡単な実装をシミュレートします。センターで電話を受けると、顧客および連絡先チャネルに関する情報が入力されます。

既知の顧客情報に基づき、この顧客に対する抱合せ販売のオファーが提供されます。オファーの成否が追跡され、サーバーに返されます。これにより、基になる意思決定モデルがその機能の正確性の向上に役立つフィードバックを得て、より的確な抱合せ販売のオファーを実行できるようになります。

Cross Sell インライン・サービスでは、キー・パフォーマンス・インディケータ（KPI）による意思決定プロセスの推進、経費削減や収益増加などの競合する KPI の最適化、正しい意思決定を行うためのグラフィカルなルールベースのスコアリングの使用、最良の意思決定を予測するための分析的な自己学習モデルの使用などを含む、Oracle RTD の多くの機能を強調しています。

ただし、Cross Sell プロジェクトに表示されるいくつかの機能は、シミュレーション専用であるので注意してください。これらはサンプルの中で明示的に示しているので、本番のインライン・サービスでは使用しないでください。

Cross Sell サンプルは、プロジェクトをインポートすることで表示できます。このプロジェクトは、次の場所にあります。
RTD_HOME¥examples¥CrossSell

プロジェクトをインポートしたら、各要素をダブルクリックしてその要素のエディタを表示することで、次の機能を表示できます。

機能	要素名	説明
複数の KPI	Performance Goals	Cross Sell インライン・サービスは、組織の収益の最大化と経費削減の両方を最適化するように設計されています。
動的な顧客データ	Data Source/CustomerDataSource Entity/Customer	データソースとエンティティを組み合せると、顧客に提示するオファー・タイプの意思決定を支援する顧客データにアクセスできます。 エンティティは、1 つまたは複数のデータソースを、インライン・サービスの重要な単位を表すオブジェクトにマップする手段を提供するオブジェクトです。 エンティティを介してアクセスされるデータはセッション固有です。
抱合せ販売および顧客維持のためのオファー	Choices	拡張可能なオファーは「Choices」の下にまとめられています。これらのオファーには、抱合せ販売のオファーを意図したものも、顧客維持率の向上を意図したものもあります。各オファーの「Score」タブを開くと、オファーに割り当てられている評価のスコアが表示されます。スコアは、パフォーマンス目標（Revenue や Retention）ごとに用意されています。一部のオファー（たとえば、all Credit Cards）は、親の選択肢グループからスコアリングを引き継ぎます。これは、このグループ内のすべてのオファーが同様にスコアリングされることを示します。この場合、スコアは Profit Margin を受入れ確率で乗算して計算されます。 その他のオファー（Reduced Interest Rate など）は、ルールを使用して計算します。Reduced Interest Rate の収益目標には、組織の収益が赤字を示す場合、実際に負の値がスコアリングされます。

機能	要素名	説明
複数の KPI	Performance Goals	Cross Sell インライン・サービスは、組織の収益の最大化と経費削減の両方を最適化するように設計されています。
動的な顧客データ	Data Source/CustomerDataSource Entity/Customer	データソースとエンティティを組み合せると、顧客に提示するオファー・タイプの意思決定を支援する顧客データにアクセスできます。 エンティティは、1 つまたは複数のデータソースを、インライン・サービスの重要な単位を表すオブジェクトにマップする手段を提供するオブジェクトです。 エンティティを介してアクセスされるデータはセッション固有です。
スコアリング・ルール	Scoring Rules/Reduced Interest Rate	スコアリング・ルールは、顧客に関する情報など、セッション・データを使用してオファーを動的にスコアリングする方法です。
母集団のセグメント	Filtering Rules/Segment to Retain	フィルタリング・ルールを使用して母集団をセグメント化できます。このルールにより、顧客維持オファーに適するグループと、抱合せ販売の対象となるグループの 2 つのグループが出力されます。顧客が 6 回以上の放棄呼を受け、かつ 2 年を超えて顧客であり続けた場合、その顧客は維持オファーのグループにフィルタリングされます。
母集団のセグメントによる意思決定の重み付け	Decisions/OfferDecision	「Decision」要素を使用して、競合するパフォーマンス・メトリック全体で意思決定プロセスの重み付けを行うことができます。このサンプルでは、顧客維持プロファイルに一致する母集団のセグメントに対して、Customer Retention のスコアが高いオファーを重みの高い優先度に設定します。

機能	要素名	説明
複数の KPI	Performance Goals	Cross Sell インライン・サービスは、組織の収益の最大化と経費削減の両方を最適化するように設計されています。
動的な顧客データ	Data Source/CustomerDataSource Entity/Customer	データソースとエンティティを組み合せると、顧客に提示するオファー・タイプの意思決定を支援する顧客データにアクセスできます。 エンティティは、1 つまたは複数のデータソースを、インライン・サービスの重要な単位を表すオブジェクトにマップする手段を提供するオブジェクトです。 エンティティを介してアクセスされるデータはセッション固有です。
編成プロセスへの統合	Integration Points	統合点は、情報を収集したり（IVR から顧客に関する情報を収集する CallStart など）、外部システムに情報を提供したりすることによって（顧客に対する最高スコアのオファーを CRM システムに提供する OfferRequest など）、外部のシステムおよびプロセスと接するサイトです。 OfferResponse 統合点には `else` 分岐にコードがありますが、これはシミュレーション用なので注意してください。本番環境では、オファーが受け入れられたかどうかはサービス・センターのオペレータからのフィードバックとなります。

警告: インライン・サービスの負荷生成スクリプトが実行されているときの時間の経過をシミュレートするために、`Application.java` で `currentTimeMillis` メソッドが上書きされています。Cross Sell を本番のインライン・サービスのベースとして使用する場合は、次の上書きファイルを削除する必要があります。*RTD_HOME*¥examples¥CrossSell¥src¥com¥sigmadynamics¥sdo¥Application.java

生成済コードの上書きの詳細は、第 1.2.6 項「生成済コードの上書き」を参照してください。

Cross Sell インライン・サービスをデプロイしてデータをロードする準備ができました。インライン・サービスをデプロイしたら、*RTD_HOME*¥scripts¥loadgen.cmd を実行して Load Generator を開きます。そこで「**Open an existing Load Generator script**」を選択して、*RTD_HOME*¥examples¥CrossSell¥etc¥LoadGen.xml にブラウズします。最後に、スクリプトを実行します。

このスクリプトでは、シミュレーション上の顧客データを取得してインライン・サービスを実行します。データおよび検出された相関関係は、Decision Center で表示できます。

1.3.7 Decision Studio バージョン 1.2 のファイルのオープン

Decision Studio の以前のバージョンのインライン・サービスを開いても、プロジェクトとして作成されません。これをプロジェクトとして開くには、「**New Inline Service Project**」を開始し、「**Create project at external location**」を使用して、ファイル・システム上の対象ファイルを探します。

これにより、以前のバージョンのインライン・サービスが Decision Studio 2.2 プロジェクトに変換されます。

1.4 インライン・サービスのディレクトリ構造

インライン・サービスを作成する場合、プロジェクトはファイル・システム上の任意の場所に作成できます。使用しやすいように、すべてのプロジェクトは 1 つのディレクトリに保存することをお薦めします。デフォルトのワークスペースは、`C:¥Documents and Settings¥`*User_name*`¥Decision Studio` です。

インライン・サービスを保存すると、そのディレクトリ名はインライン・サービス名と同じになります。インライン・サービスには、次のディレクトリ構造が作成されます。

..¥*Inline_Service*¥classes	インライン・サービスのコンパイル済クラス
..¥*Inline_Service*¥dc	Decision Center のカスタム JSP 用フォルダ
..¥*Inline_Service*¥etc	Load Generator スクリプト
..¥*Inline_Service*¥gensrc	インライン・サービスの生成済ソース・コードの場所

..¥*Inline_Service*¥meta	インライン・サービスのメタデータ
..¥*Inline_Service*¥src	インライン・サービスの生成済コードを上書きするソース・コード

1.5 インライン・サービスの構成

インライン・サービスは、Decision Studio を使用して構成およびデプロイされます。要素がプロジェクトに追加され、機能ロジックを含む Java スクリプトレットが特定の要素に追加されます。インライン・サービスを保存すると、XML メタデータが作成されて Java コードが生成され、Real-Time Decision Server にデプロイされて、ここでインライン・サービスが実行されます。インライン・サービスの構成の詳細は、『Oracle Real-Time Decisions スタート・ガイド』を参照してください。

1.5.1 オブザーバ・インライン・サービス

監視では、インライン・サービスはビジネスに関するデータが収集される収集ポイントに焦点を当てます。このデータにおける傾向と相関関係の洞察および検出は、将来の動向を予測し、変化の結果を予想する自己学習モデルによって行われます。このタイプのインライン・サービスをオブザーバといいます。

これらの検出はシン・クライアントである Decision Center に公開されます。Decision Center では、ビジネス・ユーザーがこれらの洞察を使用して意思決定を行います。またビジネス・ユーザーは、Decision Center を使用してインライン・サービスの管理および最適化も実行します。

1.5.2 アドバイザ・インライン・サービス

ビジネス・プロセスのアドバイスの実行では、主要な意思決定ポイントおよび収集ポイントでインライン・サービス間が接続されます。意思決定ポイントとは、全体的なビジネス・プロセスにおいて、製品の推奨や顧客維持オファーの提示など、主要なビジネス上の意思決定される場所です。最初に収集ポイントでデータが集められ、自己学習モデルを使用して検出され、組織の実現目標としているパフォーマンス・メトリックに従って選択肢がスコアリングされます。インライン・サービスにより、ビジネス・プロセスの意思決定ポイントで、最高スコアの選択肢が提示されます。フィードバックを介してこれらの選択肢の成功レベルがインライン・サービスに返されると、より適切な選択肢を提示するようにモデルの機能が向上します。このタイプのインライン・サービスをアドバイザといいます。

2　Decision Studio の要素と API について

Decision Studio の要素は Decision Studio 内でグラフィカルに構成され、ロジックが Java スクリプトレットの形式で追加されます。次の各項では、各要素のプロパティおよび要素に含まれる Java スクリプトレット（存在する場合）について、例とともに説明します。

この章の内容は次のとおりです。

2.1　要素の表示ラベルとオブジェクト ID について

要素を作成する際、その要素の表示ラベルを入力します。表示ラベルを入力すると、オブジェクト ID が自動的に生成されます。オブジェクト ID は Java 命名規約に準拠するように自動的に作成されます。変数名は大文字と小文字が混在し、最初の文字は小文字になります。クラス名は大文字と小文字が混在し、最初の文字は大文字になります。ラベル名に空白が含まれる場合は、オブジェクト ID の生成時に空白が削除されます。オブジェクト ID を手動で入力すると、自動的に作成されたオブジェクト ID を上書きできます。

「Inline Explorer」のタスク・バーにある  アイコンを使用すると、オブジェクトの表示ラベルとそのオブジェクト ID 間で表示を切り替えることができます。

2.2 アプリケーション要素について

Decision Studio で新しいプロジェクトを開始すると、アプリケーション・オブジェクトが作成されます。アプリケーション・オブジェクトのプロパティでは、次の特性が定義されます。

- アプリケーション・パラメータ
- 制御グループ
- モデルのデフォルト値
- ロジック
- 権限

これらすべての値は、Decision Studio インタフェースを使用して定義されます。

2.2.1 アプリケーション・パラメータ

デバッグ・オプションの使用

本番データベースに対してデプロイされているインライン・サービスをテストする場合、モデル・データを混在させないようにするには、デバッグ・オプションを使用してデータが書き込まれないようにします。デバッグ・オプションは次のとおりです。

- 「Disable learning」: モデルの現在の状態を維持して、テストにより追加的な学習が導入されないようにします。
- 「Disable database writes」: データがデータベースに書き込まれないようにします。

Parameters	インライン・サービスに必要な任意のパラメータ。パラメータには、名前、データ型、デフォルト値があり、配列にすることもできます。

アプリケーション・パラメータの追加

アプリケーション・パラメータは、すべてのセッションについて定義および保存できるグローバルレベルのパラメータです。セッションは、アプリケーション・オブジェクトに指定するセッション・キーで定義されます。たとえば、CustomerId をセッション・キーとして指定すると、新しい CustomorId が識別されるたびに新しいセッションが作成されます。セッションは統合点によって明示的にクローズされるか、またはタイムアウトによってクローズされます。

「**Application Parameter**」タブで「**Add**」をクリックしてパラメータを追加します。パラメータを追加する際、「Name」、「Type」、「Array」および「Default Value」を入力します。

「**Remove**」をクリックすると、パラメータは削除されます。

2.2.2 アプリケーション API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

アプリケーション・パラメータ・メソッド

グローバル・パラメータが設定されると、コード内にゲッターとセッターが生成されます。アプリケーション・パラメータ（たとえば、`string` 型の `myApplicationParameter`）にアクセスするには、次を使用します。

```
String param =  Application.getApp().getMyApplicationParameter();
```

逆に、アプリケーション・パラメータを設定するには、次のように指定します。

```
Application.getApp().setMyApplicationParameter("my parameter");
```

2.2.3 制御グループの構成

制御グループは、予測モデルの結果を既存のビジネス・プロセスと比較可能にするベースラインとして機能します。制御グループのデシジョンは、Oracle RTD がインストールされていないときのデシジョンを正しく反映するように、正確に定義することが重要です。たとえば、コール・センター用の抱合せ販売インライン・サービスでは、Oracle RTD の導入前に抱合せ販売が行われていなかった場合は、選択肢を何も返さないように制御グループのデシジョンを定義する必要があります。

No Control Group	これを選択すると、制御グループは使用されず、「**Selection Value**」が 0 に設定され、「**Use value literally**」が選択されて無効になります。制御グループを有効にして設定を変更するには、「**No Control Group**」を選択解除します。
Selection value	属性値への参照。この値を使用して、リクエストのランダムな選択を制御グループに生成します。 「**Use value literally**」を選択解除する場合、制御グループの選択値は、セッション・キーまたは顧客を一意に識別する属性を参照する必要があります。制御グループの参加は、選択値の疑似乱数ハッシュを使用して決定されます。計算の結果は決定論的であり、制御グループの選択値と指定されたサイズにのみ依存します。制御グループの実際のサイズは、指定されたサイズと微妙に異なります。 「**Use value literally**」オプションを選択すると、選択値により制御グループの参加が直接決定されます。この場合の選択値には、ブール（制御グループの参加を true 値で示す）または整数（制御グループの参加を 0 以外の値で示す）を指定できます。
Use value literally	顧客の制御グループへの割当てを Oracle RTD 以外で行う場合に使用します。制御グループの選択値として使用する属性は、この割当てを示す必要があります。
Percent of Population	制御グループに割り当てる必要がある顧客の総数に対するパーセンテージ。

Use for analysis	制御グループの参加を解析により追跡するかどうかを制御します。
Name for Analysis	解析モデルが制御グループの参加の追跡に使用する名前。

2.2.4 モデルのデフォルト値の設定

モデルのデフォルト値は、使用するモデルおよびモデルの設定方法を制御します。ほとんどのモデルのデフォルト値は、Oracle サポート・サービスによる指示のないかぎり変更しないでください。

Study name	インライン・サービスで使用するスタディの名前。通常は、インライン・サービスごとに個別のスタディを持ちます。これは、スタディ名をインライン・サービス名と同じにすると実現できます。ただし、既存のスタディはインライン・サービスのテスト時に使用されることがあります。そのような場合は、学習を無効にして本番データが失われないようにする必要があります。
Persistence Interval	モデル・データをデータベースに保存する時間間隔。
Time Window Duration	デフォルト値は「Quarter」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
First Day of Week	デフォルト値はロケールにより異なります。米国では、デフォルト値は「Sunday」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
First Month of Year	デフォルト値は「January」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
Build when data changes by	デフォルト値は「20%」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
Significance threshold	デフォルト値は「25」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
Correlation threshold	デフォルト値は「0」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
Max Input Cardinality	デフォルト値は「500」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。
Max Input Buckets	デフォルト値は「200」です。この機能は、Oracle サポート・サービスによる指示のないかぎり調整しないでください。

2.2.5 アプリケーションの書込みロジック

アプリケーション要素の「**Logic**」タブにある「**Initialization Logic**」および「**Cleanup Logic**」パネルを使用して、インライン・サービスを初期化およびクリーンアップするスクリプトレットが Decision Studio を介して追加されます。

これらのスクリプトレットは、Application クラスの `init` および `cleanUp` メソッドに挿入されます。`init` メソッドは、インライン・サービスがロードされるときにコールされます。`cleanUp` メソッドは、インライン・サービスがアンロードされるときにコールされます。インライン・サービスのアクティブ・コピーがアンロードされ、サービスがアンデプロイおよび再デプロイされます。アプリケーションが再デプロイされると、`init` が再度コールされます。

2.2.6 インポートされた Java クラスの追加

ユーザーの設定した Java クラスを `init` または `cleanUp` メソッドで参照する場合、それらのクラスがインポートされている必要がある可能性があります。import 文を追加するには、説明の横にある「**Advanced**」をクリックします。

2.2.7 インライン・サービス権限の設定

ライフサイクル中にインライン・サービスを操作するユーザーおよびグループに対して、インライン・サービス権限を設定します。各インライン・サービスでは次の権限を使用できます。これらの権限は、インライン・サービスの開発者および Decision Center のビジネス・ユーザーに適しています。

Open Service for Reading	Decision Center でインライン・サービスを読取り専用モードで開くことを許可します。このモードは、Decision Center を使用してレポートを表示するビジネス・ユーザーに適しています。
Open Service	Decision Center ユーザーがインライン・サービスを編集して Real-Time Decision Server に再デプロイすることを許可します。 Decision Center でインライン・サービスを開くことを許可します。これにより、ユーザーに「Open Service for Reading」権限も付与されることになります。このモードは、Decision Center を使用してレポートを表示し、インライン・サービスを更新するビジネス・ユーザーに適しています。
Deploy Service From Studio	Studio ユーザーがインライン・サービスをデプロイすることを許可します。既存のインライン・サービスを再デプロイするユーザーには、既存のサービスおよび新しいサービスの両方に対するデプロイ権限が必要です。このモードは、Decision Studio を使用してインライン・サービスをデプロイするインライン・サービスの開発者に適しています。
Download Service	Decision Studio ユーザーが、サーバーから既存のデプロイ済インライン・サービスをダウンロードすることを許可します。このモードは、Decision Studio を使用してインライン・サービスをデプロイするインライン・サービスの開発者に適しています。

インライン・サービス権限は、サーバー側のクラスタ権限と連携して、権限のないユーザーによってインライン・サービスが変更または再デプロイされないように保護します。クラスタ権限の設定の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。さらに、Decision Center のパースペクティブでも権限を設定できます。詳細は、第2.19項「Decision Center のパースペクティブについて」を参照してください。

アプリケーション要素の「**Permissions**」タブを使用して、インライン・サービス権限を設定します。「**Add**」を使用して、ユーザーまたはグループをインライン・サービスに追加します。サーバーからユーザーおよびグループを取得するには、「**Get Names**」をクリックしてグループを取得し、「**Show Users**」を選択してすべてのユーザーを表示します。

ユーザーやグループはリストから選択することも、「**User or Group**」フィールドに名前を入力することもできます。ユーザーまたはグループを追加したら、「**Granted**」の下で適用する権限を選択します。

インライン・サービスからユーザーまたはグループを削除するには、ユーザーまたはグループを選択して「**Remove**」をクリックします。

注意: Windows 認証が有効な場合は、「**Get Names**」をクリックしてもユーザーおよびグループのリストを取得することはできません。かわりに、権限を割り当てるユーザーまたはグループの名前を「**User or Group**」フィールドに入力する必要があります。

2.3 データのアクセス

インライン・サービス内のデータにアクセスするには、エンティティ、データ ソース、およびセッション要素を使用します。

エンティティは、複数のソースからのデータをまとめて、インライン・サービス全体で使用されるオブジェクトを形成する抽象的な方法を提供します。エンティティは、そのコンテンツを記述する多数の属性で構成されます。また、エンティティは属性にアクセスするメソッドも備えています。

Customer などのエンティティは、様々なソースから着信するデータを組み合せます（IVR を介して入力されたアカウント番号や企業データベースから取得した顧客履歴など）。エンティティの属性にはキー（一意の識別子）もあります。

データ ソースは、データのサプライヤとして機能します。これは、リレーショナル・データベース・テーブルおよびストアド・プロシージャへの接続を管理する方法を提供します。データ ソースは、データベース・テーブルの列、またはストアド・プロシージャからの結果セットを属性として識別します。これらの属性は、エンティティ属性にマップされます。

セッション・オブジェクトは、インライン・サービスに対してメモリー内の使用可能な属性を識別する特別なエンティティです。これらの属性は、前述の Customer などのエンティティと、計算などの他のソースで設定された属性で構成できます。セッション・オブジェクトを使用して、ユーザー・セッションの情報を保存します。セッションに保存された属性はインライン・サービス全体で使用でき、セッションを閉じたときに破棄されます。

データにアクセスするには、通常、次の手順を実行します。

1. SQL テーブルまたはストアド・プロシージャに基づいてデータソースを作成します。
2. 1 つ以上のデータソースからの属性を持つエンティティを作成します。
3. キー値をエンティティに追加します。
4. エンティティを属性としてセッションに追加し、セッション・キーを割り当てます。
5. エンティティ属性をデータソース列または出力値にマップします。
6. エンティティ・キーをセッション・キーまたは関数にマップします。

2.3.1 Siebel Analytics データへのアクセス

Siebel Analytics Server は、OLTP および OLAP データベースに格納されたデータにアクセスするために、ODBC クライアント・インタフェースを公開しています。RTD Decision Service は、Java Runtime Environment（JRE）に含まれる JDBC-ODBC ブリッジを使用して、Siebel Analytics Server で提供される ODBC ドライバに接続します。

RTD Decision Service から見ると、Siebel Analytics Server は通常のデータベースと同様の SQL データソースです。Siebel Analytics Server のサブジェクト領域は、インライン・サービスによってデータベース・テーブルとして処理されます。列名は、プレゼンテーション・オブジェクト階層の 2 つのレベルを組み合せたものです。

Decision Studio データソースでアクセス可能な JDBC データソースの追加の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

2.3.2 データソースについて

データはインライン・サービス内でデータソース要素およびエンティティ要素を使用してアクセスします。データソースはデータの抽象プロバイダです。データソースは、インライン・サービスに対するデータのサプライヤとして機能します。

データソースは、Decision Studio 内ですべて作成されます。Java API を使用してデータソースにアクセスする必要はありません。データソースには、SQL データソースとストアド・プロシージャ・データソースの 2 種類があります。

2.3.3 SQL データソースの作成

SQL データソースでは、次の特性が定義されます。

Description	データソースの説明。
Data Source	JDBC データソースの JNDI 名。新しいデータソースを作成するには、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

Table Name	テーブルの名前。データベースで大文字と小文字が区別される場合であっても、この名前の大文字と小文字は常に区別されません。
Outputs Column Name	データソースから選択する列。
Outputs Type	出力列のデータ型。
Inputs Column Name	データソースに対するクエリーの WHERE 句に使用する列。これは、データソースからデータを選択する際に一致させる列です。
Inputs Type	入力列のデータ型。
Allow multiple results	複数行が返されることを許可します。このオプションを選択しない場合に複数行が返されると、最初の 1 行のみが使用されます。
Advanced	「Advanced」ボタンを使用すると、Decision Center に要素を表示し、その要素のラベルを変更することもできます。要素のラベルを変更しても、オブジェクト ID は変更されません。

データソースへの列の追加

「**Add**」または「**Remove**」をクリックすると、データソースに列を追加または削除できます。複数行が予想される場合は、「**Allow Multiple Results**」を選択します。このオプションを選択しない場合に複数行が返されると、最初の 1 行のみが使用されます。

データベース列名のインポート

「**Import**」をクリックして、データソースに直接接続します。指定されたデータソースのすべてのデータベース・テーブルが表示されます。データソースが指定されない場合は、デフォルトのデータソース SDDS が使用されます。

「**Include objects from all schemas**」を選択すると、データソース・スキーマに定義されていないテーブルが表示されます。すべてのアクセス可能なスキーマからのテーブルが表示されます。表スキーマは個別の列に表示されます。

インポートするテーブルを選択すると、それらの列の列名とデータ型がインポートされます。不要な列がある場合は、「**Remove**」を使用してそれらを削除します。

入力列の設定

「**Input column**」は、セッションに必要な行を取得するために、データベース・テーブルに対して一致させる列です。ほとんどの場合、これは単一レコードを返すための主キーまたは一意の索引列の値になります。大きな結果セットが必要な場合は、一意でない索引列にすることもできます。「**Add**」を使用して、一致させる属性を選択します。

2.3.4 ストアド・プロシージャ・データソースの作成

ストアド・プロシージャ・データソースでは、次の特性が定義されます。

Description	データソースの説明。
Data Source	JDBC データソースの JNDI 名。
Procedure Name	ストアド・プロシージャの名前。データベースで大文字と小文字が区別される場合であっても、この名前の大文字と小文字は常に区別されません。
Inputs and Outputs	ストアド・プロシージャの入出力パラメータ。各入出力パラメータには、Name、Type、および Direction があります。
Result Sets	ストアド・プロシージャからの結果セット。
Allow multiple results	複数の結果が返されることを許可します。このオプションを選択しない場合に複数の結果が返されると、最初の結果のみが使用されます。
Result Set Details	列名と予想される結果の型。
Advanced	「Advanced」ボタンを使用すると、Decision Center に要素を表示し、その要素のラベルを変更することもできます。要素のラベルを変更しても、オブジェクト ID は変更されません。

データソースへの属性の追加

「**Add**」または「**Remove**」をクリックすると、データソースに属性を追加または削除できます。複数の結果が予想される場合は、「**Allow Multiple Results**」を選択します。属性が「Input」、「Output」、または「Input/Output」かを選択します。

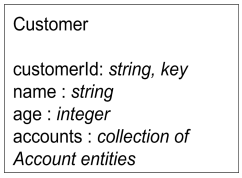
属性は順に並べる必要があります。「**Up**」または「**Down**」を使用して属性の順番を変更します。

データソースへの結果セットの追加

ストアド・プロシージャに結果セットがある場合、「Result Set」の「**Add**」ボタンを使用して結果セットを追加します。「Result Set Detail」の「**Add**」ボタンを使用すると、結果セットの列名および型を追加できます。結果セットの列は、ストアド・プロシージャから返される順番どおりに定義する必要があります。

2.4 エンティティの形成

エンティティは、名前付き属性とその属性にアクセスするためのメソッドのセットです。通常、1つの属性がエンティティのキーとして指定されます。たとえば、単純な顧客エンティティは次のようになります。



このエンティティでは、customerId がエンティティのキーで、name と age は単純な属性です。accounts は Account エンティティのコレクションです。

2.4.1 Session エンティティについて

特殊なエンティティの 1 つに Session エンティティがあります。

セッションの使用例として、クライアント Web アプリケーションについて考えます。ここでは、各リクエストはセッション・キーとして Web アプリケーションの HTTP セッション ID を備えます。新しい HTTP セッション ID を持つ最初のリクエストが到着すると、Real-Time Decision Server はそのセッション・キーが新しいことを認識し、新しい Session オブジェクトを作成して、リクエストを実行するときに統合点で使用できるようにします。これと同じ HTTP セッション ID を使用する後続のリクエストは、同じ Session オブジェクトにアクセスします。

Session エンティティは、インライン・サービスごとに自動的に作成されます。

セッション・キーについて

セッション・キーはリクエストに渡すフィールドの 1 つで、統合点で使用可能な、Real-Time Decision Server サーバーに存在する Session オブジェクトのインスタンスを識別します。統合点の処理により、後で呼び出される統合点で使用できるように、セッションの情報が明示的または暗示的に保存されます。

2.4.2 エンティティの作成

エンティティは、Decision Studio を使用して定義します。エンティティ名は、大文字で開始する必要があります。エンティティでは、次の特性が定義されます。

Description	Decision Studio に入力されたエンティティの説明。
Key	エンティティの一意の ID。「**Add Key**」をクリックすると、キーがエンティティに追加されます。
各エンティティの属性は、次のデータで定義されます。	
Description	Decision Studio に入力された属性の説明。

Type	属性タイプは、基本型、Java クラス、エンティティ・タイプ、選択肢または選択肢グループのいずれかです。
Array	単一値かコレクションかどうか。
Default value	デフォルト値。定数、関数、または属性への参照を指定できます。
Add attribute/key options	
Use for analysis	このオプションを選択して、予測モデル内の解析にこの属性を使用します。
Category	属性のカテゴリ。カテゴリは、Decision Center での属性の表示の編成に役立ちます。
Analysis options	Date 属性の追加の解析オプションが使用できます。解析に Date を使用するには、解析に使用するパターンを指定します。月、日、曜日、時刻の影響を個別に、または任意の組合せで解析できます。

2.4.3 属性およびキーのエンティティへの追加

「**Add Attribute**」または「**Add Key**」をクリックして、属性をエンティティに追加します。属性名は、小文字で開始する必要があります（例: `aSampleAttributeName`）。属性がコレクションの場合は、「**Array**」列を選択します。

警告: Key 属性を追加する場合、データ型は自動的に String になります。データソース列または出力パラメータのデータ型が String 以外の場合は、データソースの入力の設定時に変換関数を使用します。

2.4.4 データソースからの属性のインポート

データソースのすべての属性を自動的に追加するには、「**Import**」をクリックしてインポート元のデータソースを選択します。複数のデータソースからインポートする場合は、この手順を繰り返します。「**Remove**」をクリックすると、不要な属性を削除できます。

「**Import**」を使用する際、「**Build data mappings for selected data source**」を選択すると、属性がデータソースに自動的にマップされます。エンティティがネストされ（たとえば、1 対多関係）、属性が間接的にマップされる場合は、このオプションを選択しません。

2.4.5 解析の属性の使用

「**Use for Analysis**」を選択して、解析モデルに属性を追加します。

2.4.6 Decision Center の表示

「**Show in Decision Center**」オプションは、デフォルトで選択されます。Decision Center ユーザーに対して属性を表示しないようにする場合は、このオプションを選択解除します。「**Category**」を選択すると、Decision Center の属性の表示を制御できます。

2.4.7 セッション・キーの追加

使用するセッション・キー値がエンティティの属性である場合、最初にエンティティをセッションに追加します。そのためには、「**Add attribute**」をクリックして新しい属性を追加します。

たとえば、セッション・キーを Customer エンティティの `customerID` 属性にするとします。「**Add Attribute**」をクリックして、属性を `customer` というセッションに追加します。この属性の型は、エンティティ・タイプ（つまり、`Customer`）です。

エンティティ・タイプにアクセスするには、「**Type**」列のドロップダウン・リストを使用して「**Others**」を選択します。「**Type**」ウィンドウが表示されます。この属性の「**Entity Type**」を選択します。

セッション・キーを追加するには、「**Session Keys from Dependent Entities**」から「**Select**」をクリックします。セッションの属性であるエンティティからのキー値はすべて、セッションのキー値として選択できます。ベースとなるセッションのキーを選択します。この例では、`customerID` です。

注意: キャッシュされたエンティティのキーはセッション・キーとして使用できません。

独立したエンティティ以外のセッション・キーを追加するには、「**Add Key**」を使用します。キーを追加したら、「**Default value**」をクリックしてキーを属性、定数または関数コールにマップします。

2.4.8 セッションへの属性の追加

「**Add Attribute**」を使用して、セッション全体で使用可能にする属性を追加します。セッション属性には、他のエンティティ属性と同様にゲッターおよびセッターが生成されます。

2.4.9 属性のデータソースへのマップ

Decision Studio を使用してエンティティを作成したら、エンティティの属性を定数、計算済、データソースの属性への参照、またはセッション・キーのいずれかの値にマップします。

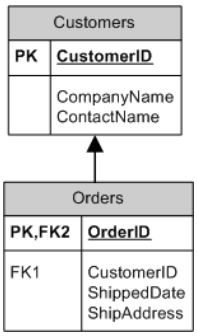
マッピングは、エンティティ・オブジェクトの「**Mapping**」タブで実行します。エンティティの属性をデータソースにマップするには、「**Source**」列を使用して、データソース列のパスまたはデータの属性を提供する出力パラメータを選択します。

キー値をマップするには、「**Data Source Input Values**」から「**Input Value**」をクリックします。ここで、属性をデータソース値にマップすると、キー値が表示されます。キーは、セッション・キー属性、別のエンティティ・キー値、または関数にマップできます。入力タイプは、String 型にする必要があります。String 型でない場合は、非文字列値を変換する関数を使用します。

2.4.10 1 対多関係

1 対多の外部キー関係のエンティティのデータにアクセスするには、関係するエンティティを最初のエンティティの属性にします。たとえば、次のような関係を考えます。

Customers テーブルにキー`CustomerID` があります。Customers には多数の Orders があり、`OrderID` と外部キー`CustomerID` で識別されます。



1. Decision Studio で、各テーブルのデータソースを定義します。
2. Customers および Orders のエンティティを作成します。
3. Customer をセッションに追加します。これは、次のデータ・レベルを取得する際のキーとなります。
4. セッション・キーとして `CustomerID` を選択します。
5. Orders と Customers の間の 1 対多関係を関連付けるには、`Orders` エンティティ・タイプの Orders という属性を Customer に追加します。1 つの Customer に対して多数の Orders があるので、これを配列にします。
6. Customer エンティティのマッピング・タブを使用してすべての属性値をマップできます。

2.4.11 インポートされた Java クラスの追加

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある「**Advanced**」をクリックします。

2.4.12 セッション・ロジック

セッション要素は、セッションの初期化および終了で実行されるスクリプトレットを受け入れます。クリーンアップ・スクリプトレットはセッションが強制終了されたとき、またはタイムアウトしたときに実行されます。

2.4.13 セッション API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

`session()` を使用して、インライン・サービスの別のエンティティにアクセスできます。たとえば、次のような場合があります。

```
session().getCustomer().getCustomerId();
```

ここで、Customer はエンティティ、customerId はそのエンティティの属性です。

session() を使用して、アプリケーション・セッションのインスタンスにアクセスします。セッションでは、次の API が使用できます。

```
public boolean isTemporary();
```

セッション・キーが渡されなかった場合、セッションは一時的であるとみなされます。

```
  public IntegrationPointRequestInterface getRequest();
```

統合点の外側の統合点リクエストへのアクセスに使用します。現在のリクエストを返します。

```
  boolean isClosed();
```

現在のセッションのインスタンスが閉じているかどうかを返します。

```
  Set getKeys();
```

既知のセッション・キーを返します。これは、コールされた場所により、完全なセットである場合もそうでない場合もあります。

```
  public void close();
```

現在のセッション・インスタンスを閉じます。

```
  public ApplicationInterface getApp();
```

現在のセッションのアプリケーション・オブジェクトを取得します。

2.4.14 エンティティ API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

2.4.15 エンティティ・クラスについて

生成される通常のクラスに加え、各エンティティには配列クラスも生成されます。生成されるクラスには、各属性のプロパティ、ゲッターおよびセッターがあります。したがって、Customer、Account、Call などのエンティティを定義すると、これらの名前を持つクラスとそのクラスのコレクションを示すもう 1 つのクラスが生成されます。

たとえば、Account エンティティの場合、次の 2 つのクラスが生成されます。

```
Account

SDAccountArray
```

2 番目のクラスは、Account のコレクションを表します。そのため、Customer エンティティに、名前が accounts でタイプが Account の属性がある場合（多重性に 1 超の値が設定されている場合）、Customer に次のゲッターが生成されます。

```
 Customer  {

 SDAccountArray getAccounts() {

   }

 void setAccounts(SDAccountArray accounts) {

   }

void addToAccounts(Account account) {

   }

 }
```

2.4.16 エンティティの作成

クラスはエンティティ・タイプごとに生成されるため、他の Java オブジェクトと同様に、new 演算子でエンティティを作成します。次に例を示します。

```
Customer cust = new Customer();
```

エンティティが別のエンティティの子である場合は、オプションで親エンティティの名前を渡します。セッションは、別のエンティティの親エンティティになることもあります。

```
Customer cust = new Customer(entityParent);
```

2.4.17 エンティティ・キーの追加

ほとんどのエンティティは、キー属性の値がなければ有用ではありません。キー属性は、他の属性と同様に、生成されたセッターを使用して設定されます。

```
Customer cust = new Customer();

cust.setCustomerId(newKey);
```

2.4.18 エンティティ属性へのアクセス

前述のように、ゲッターは属性ごとに生成されます。ゲッターの形式は、属性が 1 つの値を持つか複数の値を持つかによって異なります。サンプルの Customer エンティティは次のゲッターを持ちます。

```
String id = cust.getCustomerId();

String name = cust.getName();

double age = cust.getAge();
```

```
Collection accounts = cust.getAccounts();
```

対応するセッターも生成されます。customerId のセッターを見てきましたが、ここでは別の Customer の例を示します。

```
cust.setName("Fred Johnson");

cust.setAge(42);

cust.setAccounts(newCollection);
```

また、Accounts は複数の値を持つ属性であるので、コレクションに追加することもできます。

```
cust.addToAccounts(anotherAccountObject);
```

配列は、次を使用して追加できます。

```
cust.addAllAccounts(anotherAccountArray);
```

2.4.19 エンティティのリセットと入力

エンティティをリセットおよび入力するために、3 つの特別なメソッドが用意されています。

```
cust.reset();
```

セッション・キー以外のすべてのキーとすべての属性をリセットします。

```
cust.resetAttributes();
```

すべての属性をリセットしますが、キーはリセットしません。

```
cust.fill();
```

fill により、エンティティ・マッピングに従って属性の値が再帰的に入力されます。データ ソース、計算済または定数値のデータは強制的にリフレッシュされます。また、エンティティ・タイプの任意の属性も入力されます。

reset および fill は、キャッシュされたエンティティに対してはコールしないでください。

2.4.20 キャッシュされたエンティティについて

エンティティは、アクセスしやすいようにサーバー上にキャッシュできます。エンティティをキャッシュするには、エンティティの「**Cache**」タブをクリックします。

キャッシュの特徴は次のとおりです。

Enable caching for this entity type	このオプションを選択してキャッシュを有効にします。キャッシュされたエンティティはキャッシュされていないエンティティと同様に処理され、同じ API を持ちますが、キャッシュされたエンティティのキーをセッション・キーとして使用することはできません。

Max number of items to cache	キャッシュするアイテムの最大数。アイテムは先入れ先出し方式でフラッシュされます。
Caching strategy	
Use fixed lifetime	リフレッシュされるまで各オブジェクトをキャッシュに維持する秒数。
Use fixed period	キャッシュ全体がリフレッシュされるまでの秒数。
Never refresh cache	キャッシュされたアイテムは、最大数に達するまでキャッシュに維持されます。

エンティティがキャッシュにマークされている場合、次を使用して属性を設定します。エンティティを作成したら、キー値を設定して、キャッシュから属性値を取得します。キャッシュされたエンティティ属性（キー以外）には、セッターがありません。そのため、エンティティはキャッシュされたバージョンと常に同期します。

```
Customer cust = new Customer();

cust.setCustomerId(newKey);

cust.getCustomerId();

cust.getName();

cust.getAge();

cust.getAccounts();
```

2.5 意思決定プロセス

意思決定プロセスは、組織に関連する全体的なパフォーマンス目標、その目標を測定するパフォーマンス・メトリック、各選択肢のスコアリングに必要なアクション、および母集団のセグメントに基づくそのスコアの重み付けを考慮したフレームワークをベースとしています。

このフレームワークには、次のような要素があります。

1. パフォーマンス目標
2. 意思決定
3. 選択肢グループ
4. 選択肢
5. フィルタリング・ルール
6. スコアリング・ルール

7. 予測モデル

2.6 パフォーマンス目標

組織の意思決定プロセスの設計では、最初に組織の全体的なパフォーマンス目標を検討します。パフォーマンス目標は、組織が成功の度合いを測定する特定のメトリックで構成されます。一般的なパフォーマンス・メトリックには、次のようなものがあります。

- 収益
- コスト
- 顧客当たりの製品数
- 顧客満足度

パフォーマンス・メトリックは、最適化の方向（最大化または最小化）と正規化指数で構成されます。

パフォーマンス目標の特徴は次のとおりです。

Performance metric	組織全体の目標に関連する成功の度合いを測定するよう選択したメトリック。
Optimization	パフォーマンス・メトリックを最適化する方向を示す値。「Minimize」または「Maximize」。
Required	パフォーマンス・メトリックのスコアリングが必要な場合は選択します。メトリックが必要であるとマークされていない場合、データ不足によってスコアが使用できないと、Oracle RTD は別のスコアを調査してスコアを提供します。メトリックが必要であるとマークされている場合、全体的なスコアは提供されません。メトリックは使用不可とマークされ、スコアリング・プロセスから削除されます。
Normalization factor	このパフォーマンス・メトリックの組織の相対値。

2.6.1 パフォーマンス・メトリックの追加

「**Add**」をクリックして、パフォーマンス・メトリックを追加します。メトリック（例: 収益）、最適化の方向（最大化）、メトリックで意思決定を行うためのスコアを使用可能にする必要があるかどうかを追加します。

メトリックをすべて追加したら、正規化ファクタを決定する必要があります。

2.6.2 正規化ファクタの計算

正規化ファクタは組織への相対値を示します。多くの場合、パフォーマンス・メトリックは同じスケールで測定されません。たとえば、チャーンは乗り換えた顧客数で測定され、収益はドルで測定されます。チャーン・メトリックと収益メトリックが同じパフォーマンス目標の一部である場合、最初に平準化ファクタを探す必要があります。たとえば、顧客を 1 人失うと組織に$500 のコストがかかることがわかっている場合、正規化ファクタとして 500 を使用し、チャーン・メトリックを「**Minimize**」とマークします。

2.7 選択肢グループおよび選択肢

選択肢グループおよび選択肢の各要素は、多数の選択肢からの 1 つ以上の選択を可能にします。選択肢は、アドバイザから返されるようにデシジョンに返されるか、インフォーマントによってモデルに登録されます。選択肢の選択プロセスは、次の計算に分かれます。

1. 適格性: 意思決定に対して選択肢を選択可能にするかどうか決定するルールのセット。
2. スコアリング: パフォーマンス・メトリックで定義される各キー・パフォーマンス・インディケータ（KPI）に沿ったスコアの計算。選択すると、スコアは、特定のパフォーマンス・メトリックに対して選択肢が持つ期待効果に数値を割り当てます。
3. 正規化: 様々なパフォーマンス・メトリックに沿ってスコアを一般のスケールに移行して、スコアを比較できるようにします。
4. 総計化: 選択肢ごとに 1 つの数値を生成します。この数値を使用して、各選択肢の相対利益を比較できます。
5. 選択: 使用する選択肢を指定します。選択は通常、総計化で計算される総計に適用されます。

2.7.1 選択肢グループおよび選択肢について

選択肢グループと選択肢には次の特徴があります。

Attribute values	選択肢を構成する属性。これらは、親の選択肢グループから継承されるか、選択肢レベルで割り当てられます。
Scores	各選択肢は、「Scores」タブの定義に従ってスコアリングされます。選択肢は、パフォーマンス目標に含まれるすべてのパフォーマンス・メトリックに対してスコアリングされます。
Choice Events	選択肢イベントは、グループ・レベルでのみ記述されます。これらのイベントは、選択肢に関するライフサイクルについて定義されます。たとえば、実行される Cross Selling Offer には、Offered、Accepted、Product First Used などのイベントがあります。

Rules	ルールは、スコアリングおよび最終デシジョンで選択肢を考慮できるかどうかに影響します。ルールは、Decision Center ユーザーが変更できます。
Advanced	「Advanced」ボタンを使用すると、Decision Center に要素を表示し、その要素のラベルを変更することもできます。要素のラベルを変更しても、オブジェクト ID は変更されません。

2.7.2 選択肢属性について

選択肢属性の特徴は次のとおりです。

Name	属性の名前。
Category	属性が属するカテゴリ。カテゴリは、Category 要素で定義されます。
Type	属性のデータ型。
Array	属性がコレクションであるかどうか。
Inherited Value	存在する場合は、選択肢グループまたは選択肢の属性が親から継承した値。
Value	属性の値。この値は、常に継承した値を上書きします。
Show in Decision Center	このオプションを選択すると、ビジネス・ユーザーに対して属性が Decision Center で表示されるようになります。属性を内部的に使用する場合は、選択解除します。
Use for indexing	指定した属性で選択肢を検索できるようにする場合、このオプションを選択します。たとえば、`name` という選択肢属性があるとします。選択肢グループに、次のような静的メソッドが生成されます。 `getChoiceWithName(String name)` このメソッドは選択肢を 1 つ返します。
Send to client	この選択肢を返すインライン・サービスをコールする外側のクライアントに属性を送信する場合は、このオプションを選択します。

2.7.3 選択肢属性の追加

選択肢属性を追加または削除するには、「**Add**」または「**Remove**」をクリックします。選択肢属性を編集するには、属性を右クリックして、「**Properties**」を選択します。選択肢属性は、定義されている最上位レベルでのみ編集できます。

2.7.4 選択肢グループ属性について

選択肢グループ属性により、柔軟なグループ・レベルのルールを使用できます。グループ属性は選択肢グループ・レベルにのみ適用され、個々の選択肢には割り当てられません。

2.7.5 選択肢属性について

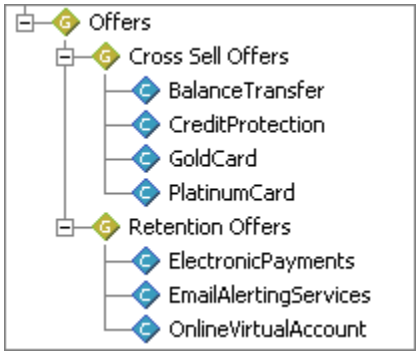
選択肢属性で選択肢を定義します。選択肢属性は、グループの各選択肢が同じ属性のセットを持つように選択肢グループ・レベルで設定した後、選択肢レベルで個別に定義します。選択肢属性にはデフォルト値を設定できます。これは、低位レベルで設定して上書きできます。

選択肢グループおよび選択肢は階層的に定義されます。階層は、選択肢の論理分類に準拠します。最上位レベルでは、サブツリー全体で一貫性のある選択肢属性の定義を考慮する必要があります。低位レベルでは、階層の形は通常、組織的な問題によって決定されます。

選択肢属性は、通常、階層の高位レベルで定義します。属性の中には上書き不可とマークされているデフォルト値を持つものもあります。これは、デフォルトで設定された値が使用される値になることを示します。これは一般的に計算が含まれる場合に設定されます。これはデプロイ後、ビジネス・ユーザーに属性を更新させないようにする場合に有用です。

選択肢の値には、定数、属性や変数、関数コールまたは予測を指定できます。予測属性は実行時にモデルによって計算され、デシジョンや他のロジックで正しい選択肢が選択されるようにスコアリング・デバイスとして使用されます。予測属性にはデフォルト値がありません。

例として、次の選択肢グループを考えます。



Offers レベルで設定される選択肢属性は次のとおりです。

属性	型	値
Message	String	デフォルト値なし。各選択肢の値は選択肢レベルで割り当てられます。
ShouldRespondPositively	Boolean	特定の選択肢に対して顧客が肯定的に反応するかどうかを示すブール値を返す `ShouldRespondPositively()` 関数。
likelihood	Predictive/Double	モデルによって割り当てられるプレースホルダ属性。選択肢が受け入れられる確率に使用します。実行時に計算されるため、デフォルト値はありません。
Profit Margin	Double	デフォルト値は 0.5。各選択肢の値は選択肢レベルで割り当てられます。

各選択肢は、Profit Margin および Message の値を、その選択肢を示す値で上書きします。ただし、有効な期間にサーバーが応答できない場合は、実行時にデフォルト値が使用されます。

選択肢は、`ShouldRespondPositively` 属性を上書きしません。これは、すべてで同じ関数を使用してその値を決定するためです。likelihood は、実行時に選択肢ごとにモデルで計算されます。

グループ・レベルには、もう 1 つの属性があります。それはグループ属性です。

属性	型	値
averageLikelihood	Predictive/Double	選択肢全体の likelihood の平均としてモデルで使用する属性。特定の選択肢の likelihood が使用できない場合の likelihood として使用します。実行時に計算されるため、デフォルト値はありません。

2.7.6 選択肢のスコアリングについて

選択肢グループおよび選択肢は、その親からスコアリングを継承します。選択肢または選択肢グループのスコアリングで、その選択肢に適用するパフォーマンス・メトリックを特定して、スコアリング・メソッドをそれに適用します。スコアリング・メソッドには、スコアリング・ルール、関数、定数、選択肢イベント・モデルでのイベントの発生率などを指定できます。

たとえば、前述の選択肢グループ構造を仮定すると、一部の選択肢は次のようなスコアリングを持ちます。

Mileage Plus Card	
Performance Metric	スコア
Increase Revenue	顧客がオファーを受け入れる可能性と、オファーの潜在収益を計算するためのカードの期待利益を使用する関数。可能性はモデルで計算されます。

Gold Card	
Performance Metric	スコア
Increase Revenue	選択肢グループ・レベルから継承された定数。

Credit analysis	
Performance Metric	スコア

Increase customer retention	顧客データを使用してスコアを割り当てるスコアリング・ルール。

スコアリングは double データ型を返す必要がありますが、パフォーマンス目標のパフォーマンス・メトリックに指定された正規化率と概念的に一致する必要があります。たとえば、平準化ファクタがドル間である場合、その関数はドルを表す値を返す必要があります。

2.7.7 適格性ルールについて

選択肢および選択肢グループには、意思決定の参加の適格性を判断するルールがあります。このルールは、目的の選択肢が、選択関数やデシジョンのルールまたは選択肢を選択するロジックへの参加に適格かどうかを判断します。グループレベルのルールは、選択肢グループ・エディタの「**Choice Eligibility**」および「**Group Eligibility**」タブにあります。選択肢レベルのルールは、選択肢エディタの「**Eligibility Rule**」タブにあります。

ルールは、選択肢グループまたは選択肢の先頭となるサブツリーが意思決定に適格かどうかを判断します。選択肢自身が適格であっても、そのすべての子孫が適格でないと、それらは適格となりません。

これらのルールのエディタの使用方法は、第2.10項「ルール・エディタの使用」を参照してください。

2.7.8 選択肢グループのルールおよび選択肢の適格性ルールの評価

選択肢グループおよび選択肢のルールは、継承され、追加されるようになっています。つまり、選択肢グループ（Group および Choice ルール）および選択肢レベルでルールがある場合、それは論理積が適用されるようにルールが拡張されます。継承されたルールは、「**Inherited eligibility conditions**」というラベルの付いたルールの最上位の展開可能なセクションに表示されます。  および  ボタンを使用して、セクションを開いたり閉じたりします。

次のような例を考えます。

$Group_1$ にルール $GroupRule_1$ と $ChoiceRule_1$ があります。

Group2 は、Group1 の子で、ルール GroupRule2 と ChoiceRule2 があります。

$Group_2$ には選択肢 $Choice_1$ があり、$Rule_1$ をルールとして持ちます。

$Choice_1$ のルールを評価する場合は、次のルールが適用されます。

GroupRule1 AND GroupRule2 AND ChoiceRule1 AND ChoiceRule2 AND Rule1

2.7.9 適格性の判断

選択肢の適格性を判断する場合は、最初に選択肢ルールが選択肢に対してテストされます。ここで、選択肢が適格であった場合、`super.isEligible()` を使用して親のルールがテストされます。`this.getParent().isEligible()` ではテストしないので注意してください。これでは、選択肢でなく親の適格がテストされてしまいます。

次の選択肢の適格を考えます。

Group$_1$ にはルール GroupRule$_1$ があります。

Group2 は、Group1 の子で、ルール GroupRule2 があります。

Group$_2$ には選択肢 Choice$_1$ があり、Rule$_1$ をルールとして持ちます。

これは、次のような方法で判断されます。

Choice$_1$ が Rule$_1$ で適格な場合、GroupRule$_2$ でテストします。

それが適格な場合、GroupRule$_1$ でテストします。

2.7.10 選択肢グループの API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

```
public Choice getChoice(String internalNameOfChoice);
```

選択肢グループから選択肢オブジェクトを返します。

```
public Choice getChoiceWithAttributeID(AttributeType val);
```

選択肢属性が索引にマークされている場合、このメソッドを使用して、索引付き属性で参照される選択肢を返します。

2.7.11 選択肢の API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

選択肢が含まれる選択肢グループを取得するには、次を使用します。

```
public ChoiceGroup getGroup();
```

API を追跡する選択肢イベントは、選択肢に定義された 2 つのメソッドで構成されます。

```
void recordEvent(String eventName);

void recordEvent(String eventName, String channel);
```

選択肢イベントを記録する統合点の一般的なコードは、次のようになります。

```
String choiceName = request.getChoiceName();

String choiceOutcome = request.getChoiceOutcome();

ChoiceGroup.getChoice(choiceName).recordEvent(choiceOutcome);
```

適格性ルールは過去に同じ顧客に受け入れられ展開された前のオファーに依存するので、展開され受け入れられたオファーは、多くのインライン・サービスで適格性ルールに対して追跡する必要があります。

選択肢イベント履歴により、特定の顧客に関する 2 つの質問に答えることができます。

- オファーが展開され受け入れられたのはどのくらい前か。
- 最近の特定の期間でオファーが展開され受け入れられた回数。

これらの質問の答えは、選択肢および選択肢グループに定義された次の API メソッドで提供されます。

```
int daysSinceLastEvent(String eventName);

int daysSinceLastEvent(String eventName, String channel);

int numberOfEventsDuringLastNDays(String eventName, int numberOfDays);

int numberOfEventsDuringLastNDays(String eventName, int numberOfDays,
String channel);
```

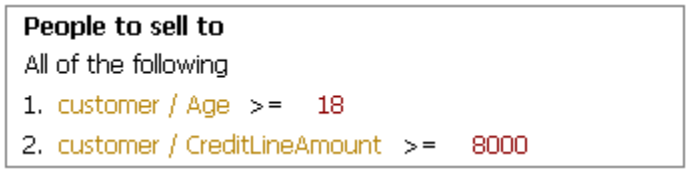
2.8 フィルタリング・ルール

フィルタリング・ルールは、スタンドアロン・ルールとして使用できます。この使用方法は、選択肢または選択肢グループに関する適格性ルールと同じです。その主な機能は、意思決定の対象となる母集団をセグメント化することです。適格性ルールの詳細は、第2.7.7項「適格性ルールについて」を参照してください。

ルールの編集の詳細は、第2.10項「ルール・エディタの使用」を参照してください。

2.8.1 母集団をセグメント化するフィルタリング・ルールの使用

フィルタリング・ルールは母集団のセグメント化に使用されます。さらにデシジョンで使用して、母集団のセグメントを差別化するパフォーマンス・メトリックを適用します。母集団のセグメント化に使用される一般的なルールは、次のとおりです。



このルールは、年齢が 18 歳以上で信用限度額が$8000 以上の顧客をターゲットとしています。フィルタリング・ルールはデシジョンで使用されます。

2.9 スコアリング・ルール

スコアリング・ルールは、スタンドアロン・ルールとして使用可能で、選択肢によるスコアの割当てに使用されます。スコアリング・ルールは、選択肢の適格性ルールの機能と似ています。ルール・エディタの使用の詳細は、第2.10項「ルール・エディタの使用」を参照してください。

フィルタリング・ルールの備える機能に加え、スコアリング・ルールは double 形式で数値スコアを評価します。スコアリング・ルールには、どのルール・セグメントも true に評価されない場合のデフォルト値があります。

値を追加するには、「**Value**」列の「**Then**」または「**The value is**」の下をクリックします。「**Edit Value**」が表示されます。ここで、他のルールの値と同様に、省略記号 ... で値を編集します。

たとえば、次のスコアリング・ルールでは、顧客の信用限度額に基づいてスコアが割り当てられます。いずれの信用限度額の範囲のカテゴリにも該当しない場合、スコアはデフォルトの 3 になります。

If All of the following 1. session / customer / CreditLineAmount > 0 2. session / customer / CreditLineAmount <= 50000	Then 7.0
If All of the following 1. session / customer / CreditLineAmount > 50000 2. session / customer / CreditLineAmount <= 60000	Then 6.0
If All of the following 1. session / customer / CreditLineAmount > 60000 2. session / customer / CreditLineAmount <= 70000	Then 5.0
If All of the following 1. session / customer / CreditLineAmount > 70000 2. session / customer / CreditLineAmount <= 80000	Then 4.0
Otherwise...	The value is: 3

Description	スコアリング・ルールは Decision Center ユーザーが調整できるため、スコアリング・ルールを適切に記述することが重要です。十分に調査したスコア範囲を指定することをお薦めします。
Advanced	「Advanced」ボタンを使用すると、Decision Center に要素を表示し、その要素のラベルを変更することもできます。要素のラベルを変更しても、オブジェクト ID は変更されません。

2.10 ルール・エディタの使用

ルールは、Decision Studio 内および Decision Center 内で、様々な目的に使用します。具体的には、母集団のセグメントをフィルタリングするためのスタンドアロンの再利用可能なルールとして、また選択肢をスコアリングするためのスタンドアロンの再利用可能なルールとして、選択肢グループや選択肢をデシジョンに含めることができるかどうかを判断するために使用します。

ルール・エディタのツールバーを使用すると、ルールの編集に使用する機能にアクセスできます。このツールバーは、実行しているタスクの状況によって異なります。右クリックして表示される状況依存メニューでも、これらの機能を使用できます。

機能は、左から右に次のようになっています。

- Edit rule properties
- Add Rule
- Add Ruleset
- Delete
- Invert
- Move up
- Move down
- Copy selection to clipboard
- Cut selection to clipboard
- Paste from clipboard

ルールの作成に使用するエディタは、それぞれよく似ています。次の項では、これらのエディタを使用してルールを作成する方法について説明します。

2.10.1 ルール・セグメントの追加

ルールを追加するには、「**Add Rule**」  ボタンを使用します。追加するルール・セグメントには、ルールとルール・セットの 2 種類があります。デフォルトのルールは、2 オペランド・ルールです。1 オペランド・ルールに変更するには、ルールの番号をクリックして選択します。オペランドの隅にある◪をクリックして、1 オペランドと 2 オペランドのいずれかを選択します。1 オペランドは、常にブールとして評価されます。

論理演算子について

一連のルールには、次の 4 つの論理演算子があります。

- All of the following（論理積、and）。論理式は、子の式がすべて満たされている場合に true になります。
- Any of the following（論理和、or）。論理式は、子の式のいずれかが true である場合に true になります。
- None of the following（否定論理積、nand）。論理式は、子の式がすべて false である場合に true になります。

- Not all of the following（否定論理和、nor）。論理式は、子の式のいずれかが false である場合に true になります。

論理演算子の値は、その演算子をクリックし、表示されるポップアップ・メニューで別の値を選択することによって変更できます。

ルールのプロパティの編集

フィルタリング・ルールにもスコアリング・ルールにもルールのプロパティがあり、そのプロパティは設定可能です。ルールのプロパティを編集するには、「**Rule properties**」  ボタンをクリックします。「**Edit rule properties**」が表示されます。

ルールのプロパティには、コール・テンプレートとネガティブ・コール・テンプレートがあります。コール・テンプレートを使用すると、別のルールからルールをコールする方法がわかりやすくなります。

コール・テンプレートを定義するには、「**Parameters**」の下の「**Add**」ボタンを使用して、ルールのパラメータの数を増やします。{0}や{1}などを引数として使用し、ルールを説明する表現を設定して、コール用のテンプレートを定義します。この表現はルール使用時に表示されるため、わかりやすい表現を使用することが重要です。

たとえば、あるユーザーから最近 x 日間に何回通話があったかを確認するルールは、次のように表現できます。

```
{0} calls in last {1} days
```

ネガティブ・コール・テンプレートは、ルールを逆にする場合に使用し、次のように逆の表現になります。

```
Not {0} calls in the last {1} days
```

ルールのプロパティを使用すると、ルールで使用する選択肢グループを割り当てることもできます。「**Use with choice group**」を選択することにより、パラメータで使用する選択肢属性を提供する選択肢グループまたはその中の選択肢を指定できます。これらの属性は、オペランドの値を編集するときに使用できます。

演算子の選択

演算子を選択するには、演算子  をクリックしてから、右下隅をクリックします。

使用可能な演算子は次のとおりです。

なし	オペランドが 1 個しかない単純な式
=	左が右と等しい
≠	左が右と等しくない
<	左が右より小さい

<=	左が右以下
>	左が右より大きい
>=	左が右以上
in	左側の値が右側のリストに含まれる
not in	左側の値が右側のリストに含まれない
includes all of	左側のリストに、右側のリストの値がすべて含まれる
excludes all of	左側のリストに、右側のリストの値が何も含まれない
includes any of	左側のリストに、右側のリストの値のいずれかが含まれる
does not include all of	左側のリストに、右側のリストの値の一部が含まれない

ルール要素の値の編集

ルールの要素を編集するには、左側をクリックしてから、省略記号...をクリックします。「**Edit Value**」が表示されます。定数、属性、関数コールのいずれかを選択できます。配列の値を指定するには、ページの最上部で「**Array**」を選択します。

「**Constant**」を選択した場合は、次の情報を指定します。

データ型	項目のデータ型。
値	定数の値。

「**Array**」を選択した場合は、必要な数の項目を配列に追加してから、データ型を選択し、それぞれの値を指定します。

「**Attribute**」を選択した場合は、次のいずれかを指定します。

グループ属性	ルールのプロパティで選択されている選択肢グループまたはその選択肢の一部である属性。
セッション属性	セッション・エンティティの一部である属性。
アプリケーション属性	アプリケーション要素のメンバーである属性。

「**Apply filter type**」を選択し、「**Data type**」を選択して、型別に属性をフィルタリングします（必須ではない）。「**Array**」を選択した場合は、必要な数の項目を配列に追加してから、それぞれに属性値を割り当てます。

「**Function call**」を選択した場合は、次のいずれかを指定します。

フィルタリング・ルール	インライン・サービス用に定義されているスタンドアロンのフィルタリング・ルール。
スコアリング・ルール	インライン・サービス用に定義されているスタンドアロンのスコアリング・ルール。
関数コール	インライン・サービス用に定義されているスタンドアロンの関数。

「**Apply filter type**」を選択し、「**Data type**」を選択して、型別に属性をフィルタリングします（必須ではない）。「**Array**」を選択した場合は、必要な数の項目を配列に追加してから、それぞれに関数またはルールを割り当てます。

ルール・セットの追加

「**Add Ruleset**」  ボタンをクリックします。ルールの新しいグループが式に表示されます。デフォルトでは、これは「**All of the following**」式です。式を変更するには、右隅をクリックします。

ルールの逆転

「**Invert**」  ボタンを使用すると、ルールの様々な要素を逆にできます。ルール・セグメントの番号を選択することにより、ルールの演算子を逆にできます。たとえば、ルールの演算子が=であった場合は、逆に<>となります。

ルールの論理演算子も逆にできます。論理演算子を選択して「**Invert**」を使用します。たとえば、「**All of the following**」は「**Not all of the following**」になります。

「**Invert**」の最後の使用方法は、ブール、つまり 1 オペランドのルールを逆にすることです。このタイプのルールを逆にすると、ルールを定義する関数のネガティブ・コール・テンプレートに変換されます。

2.11 デシジョン・プロセスについて

デシジョンは、関数またはルールに従って、選択肢をスコアリングするためにアドバイザによって呼び出され、1 つの選択肢グループから 1 つ以上の選択肢を返します。デシジョンの設定には、選択肢の選択元である 1 つ以上の選択肢グループと、選択肢をスコアリングするための関数またはルールが含まれている必要があります。実行時に、デシジョンでは、構成された選択肢グループのサブツリーにある適格な選択肢をすべて収集します。次に、選択肢がスコアリングされ、最終的に複数の選択肢が選択されます。

スコアの例は次のとおりです。

- 関心を持たれる可能性
- 予想されるビジネス上の金銭的な利益
- 予想される時間節約

あるいは、カスタム選択関数を作成して選択肢を選択することもできます。

デシジョンは、通常は適格な選択肢のグループから 1 つ以上の選択肢を選択するために使用します。最も一般的には、通常の処理用とコントロール・グループ用の 2 つのデシジョンを参照するアドバイザで使用されます。

選択条件	
Select Choice from	デシジョンでの検討対象となる選択肢グループ（複数可）の割当てに使用します。
Number of Choices to Select	デシジョンで選択される選択肢の数を示します。デフォルトであり、最も一般的に使用される数値は 1 です。これは最大値であり、実行時に返される選択肢の実際の数は、この数値以下になる場合もあります。
Select at random	選択肢グループからランダムに選択した選択肢を割り当てます。これは主として、Control Group デシジョンに使用します。
Target Segments	フィルタリング・ルールを使用してセグメント化されている母集団のセグメント。デフォルトのセグメントは全員（everyone）です。
Priorities	各セグメントに該当するパフォーマンス・メトリックに重みを付けることにより、様々なセグメントに優先度を設定するために使用します。

選択肢グループを追加するには、「**Select**」をクリックしてから、使用する選択肢グループ（複数可）を選択します。

2.11.1 母集団のセグメント化と目標の重み付け

デシジョンでは、母集団のセグメントをターゲットとし、セグメントごとにそのデシジョンにアタッチされているパフォーマンス・メトリックに重みを付けることもできます。セグメントを追加するには「**Add**」、削除するには「**Remove**」をクリックします。優先度にパフォーマンス・メトリックを追加するには「**Add**」をクリックします。

たとえば、Select Best Offer デシジョンに、「Customer Retention」と「Revenue」という 2 つのパフォーマンス目標があると想定します。母集団のセグメントも定義しました。これは、フィルタリング・ルールを使用して定義した、維持する人々です。デフォルトの残りは、抱合せ販売の対象となるセグメントです。

重み付けは、パフォーマンス目標ごと、かつセグメントごとに行います。

販売先	Customer Retention	20%
	Revenue	80%
デフォルト	Customer Retention	90%
	Revenue	10%

このデシジョンが呼び出されると、パフォーマンス・メトリックのスコアリング（whether 関数、スコアリング・ルール、関数など）が、適格な選択肢すべてに適用されます。スコアは、パフォーマンス・メトリックの正規化ファクタを使用して均一化されます。次に、デシジョンに含まれているパフォーマンス・メトリックの重み付けに従って、スコアに重みが付けられます。全体のスコアが得られ、スコアが最も高い選択肢が選択されます。

2.11.2 カスタム選択関数の使用

スコアリングでなくカスタム選択関数を使用する場合は、「**Custom Selection Function**」タブで「**Custom Selection Function**」オプションを選択します。リストから選択関数を選択し、関数に必要なパラメータを追加します。

2.11.3 選択前と選択後のロジック

「Pre and Post Selection」タブのスクリプトレットは、スコアリングが完了してデシジョンが作成される前または後に実行されます。これらのスクリプトレットでは通常、選択肢がなんらかの点で変更されるか、ファクトが記録されます。

選択前ロジックは、適格な選択肢をすべて収集してから選択が行われるまでの間に実行されます。選択後ロジックは、選択後、選択した選択肢が返されるまでの間に実行されます。選択後ロジックの方が一般的です。これを使用すると、作成されたデシジョンを記録したり、デシジョンをさらに処理できます。

このロジックでは、選択肢の計算用に定義されている変数を使用できます。たとえば、選択前は適格な選択肢、選択後は選択した選択肢を含む選択肢配列の名前が、「Pre/Post Selection」タブで設定されます（デフォルトは `choices`）。

デシジョンは `choiceArray` を返します。個々の要素にアクセスするには、配列のインデックスを使用します。次の例では、choiceArray を読み取って、ベース・イベント `Delivered` を選択肢イベント・モデルに記録します。メソッド `choice.recordEvent` は、モデル `recordEvent` をコールして、記録する選択肢に渡します。

```
    // SDCUSTOMCODESTART.<Classname>.PostSelectionBody.java.0

    for (int i = 0; i < outputChoiceArray.size(); i++) {

      Choice choice = outputChoiceArray.get(i);

      choice.recordEvent("Delivered");

    }

    session().addAllToPresentedOffers(outputChoiceArray); /* Store
presented offers for future reference */

    // SDCUSTOMCODEEND.<Classname>.PostSelectionBody.java.0
```

2.11.4 選択関数の API

重み付けパラメータの型は GoalValues です。GoalValues クラスには、デシジョンで定義されている目標ごとに、`getValue` メソッドがあります。たとえば、目標が「Customer Retention」と「Revenue」の場合、メソッドは次のようになります。

```
public double getValueForCustomerRetention();

public double getValueForRevenue();
```

2.11.5 インポートされた Java クラスの追加と Decision Center の表示の変更

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある「**Advanced**」をクリックします。Decision Center の表示ラベルを変更して、その要素を Decision Center Navigator に表示するかどうかを選択することもできます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.12 選択関数について

別の方法として、デシジョンでは選択関数を使用することができます。選択関数は、完全にユーザーが定義する関数と似ています。ただし、選択関数には明確な特性があります。選択肢配列を入力すると、選択肢配列が返されます。

選択関数の特徴は次のとおりです。

Description	選択関数の説明。
Primary Parameters	
Input Choice Array	選択関数に対する入力パラメータ。この変数のデータ型は `SDChoiceArray` です。
Output Choice Array	選択した選択肢を含む変数の名前を指定し、この選択関数のコール元に返す必要のある戻り変数。この戻り値は、この選択関数に渡される「Input Choice Array」、または「Logic」パネル内でローカルに定義した別の変数になります。この変数のデータ型は `SDChoiceArray` です。
Number of Choices Parameter	選択関数で返す必要のある選択肢の数を表す関数の引数の名前。このパラメータのデフォルト名は `numChoices` です。この引数のデータ型は `int` です。
Weights	この選択関数を使用するデシジョンに目標が定義されている場合は、その目標が、「Weights」で指定されているパラメータの下の選択関数に渡されます。この型は `GoalValue` です。`GoalValue` の詳細は、デシジョンに関する項を参照してください。
Extra Parameters	選択関数で必要なその他すべてのパラメータ。

選択関数のスクリプトレット

選択関数は、選択条件のカスタム関数として使用します。数多くの標準的な優先度関数が、テンプレートを使用して利用できます。優先度関数や選択関数は Java で定義します。これらのセットはテンプレートで事前に定義され、通常の場合、必要な箇所に記入されるか、拡張プロトタイプが用意されてそれを変更します。

「Input Choices Array」として渡されるリストから選択肢を実際に選択する Java コードは、「Logic」ペインに入力します。多くの場合、「Logic」セクションの Java コードは、他のクラスを参照します。Java コードと関数を正しくコンパイルするには、該当するクラスを関数にインポートする必要があります。

`execute` メソッドによって選択関数がコールされます。

選択関数の簡単な例を次に示します。

```
double maxL = -1.0;

Choice ch = null;

for (int i = 0; i < eligibleChoices.size(); i++) {

  Cross_Selling_OfferChoice cso =
(Cross_Selling_OfferChoice)eligibleChoices.get(i);

  double likelihood = cso.getLikelihood();

  if (ch == null || (!Double.isNaN(likelihood) && likelihood > maxL))
{

    maxL = likelihood;

    ch = cso;

  }

}

SDChoiceArray selectedChoices = new SDChoiceArray(1);

if (ch != null)

  selectedChoices.add(ch);
```

2.12.1 インポートされた Java クラスの追加と Decision Center の表示の変更

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある「**Advanced**」をクリックします。Decision Center の表示ラベルを変更して、その要素を Decision Center Navigator に表示するかどうかを選択することもできます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.13 分析モデルについて

分析モデルは、ビジネス・プロセス・パラメータのオンライン予測とデータのオフライン分析という 2 つの主な目標に対応します。

ビジネス・プロセス・パラメータのオンライン予測に際しては、モデルは選択肢にアタッチされ、特定のキー・パフォーマンス・インディケータ（KPI）に関して、選択肢を選択した場合の結果を予測する目的で使用します。

オフライン分析に際しては、モデルは、Decision Center のレポートに分析データを提供する選択肢ターゲットを分析する目的で使用します。

モデルの主なパーティション化属性は、通常は選択肢です。個々の状況や母集団を明確に区別するために、追加のパーティション化属性を使用できます。たとえば、顧客との連絡において対話に使用するチャネルや顧客の国を、パーティション化属性として使用できます。パーティション化属性は、モデルに必要なメモリー量が増大する原因となるため、慎重に使用してください。

2.13.1 モデルの種類

モデルには次の 3 種類があります。

選択肢モデル: 選択肢モデルは、相互排他的な一連のオプションのデータ分析に使用します。たとえば、コール・センターへの通話理由を分析する場合に選択肢モデルを使用します。

選択肢イベント・モデル: 選択肢イベント・モデルは、自己学習のために選択肢を追跡する目的専用に設計されています。選択肢イベントとは、インライン・サービスの内部または外部の特定の選択肢に関連するイベントです。

モデル: 汎用的で特異性のないモデル。このモデルは、特殊な目的にのみ使用してください。

2.13.2 モデルのアルゴリズム

「**Algorithm**」ドロップダウン・リストで、モデルで使用する予測アルゴリズムのタイプを定義します。母集団が大きい場合は「Regression」アルゴリズム、母集団が小さい場合は「Bayesian」アルゴリズムが適しています。

2.13.3 モデルの属性

選択肢モデルと選択肢イベント・モデルでは、「**Choice**」タブと「**Attributes**」タブが異なります。

選択肢モデルの「**Choice**」タブと「**Attributes**」タブの属性は次のとおりです。

Choice	
Choice Group	モデルが適用される選択肢グループの名前。
Label for Choice	選択肢グループのラベル。
Mutually Exclusive	このオプションは、選択肢が相互排他的な場合に選択します。

Attributes	
Partitioning attributes	各分析モデルは複数のサブモデルで構成されており、それぞれのサブモデルは、パーティション化属性の別々の値組合せに対応しています。 ある属性をパーティション化属性にするデシジョンは、問題の属性の値に応じて、予測される可能性の抜本的相違に基づく必要があります。予測結果に対する影響があまり大きくない属性は、パーティション化でなく通常の入力として扱ってください。モデルのサイズは、パーティション化属性のカーディナリティに大きく左右されます。それぞれのカーディナリティが 10 のパーティショニング属性が 2 つある場合、それぞれのカーディナリティが 2 の場合と比べ、各属性のモデルの大きさは 25 倍になります。
Excluded attributes	デフォルトでは、セッション属性とエンティティ属性はすべて、モデル入力として使用されます。ある属性をモデルが入力として使用しないようにするには、その属性を「Excluded Attributes」リストに追加できます。あるエンティティ属性をすべてのモデルから除外するには、その属性自体のプロパティで「**Use for analysis**」オプションを無効にすることが最も簡単な方法です。
Aggregate by	パーティション化属性の 1 つを選択すると、属性を学習するためのサブモデルを追加作成できます。パーティション化と同様に、ある属性に基づいて集計するデシジョンは、問題の属性の値に応じて、予測される可能性の抜本的相違に基づく必要があります。

選択肢イベント・モデルの「**Choice**」タブと「**Attributes**」タブの属性は次のとおりです。

Choice	
Choice Group	モデルが適用される選択肢グループの名前。
Label for Choice	選択肢グループのラベル。
Base Event	ベース・イベントは、分析のベースとして使用する選択肢イベントです。
Base Event Label	Decision Center に表示されるベース・イベントのラベル。
Positive Outcome events	ポジティブ結果イベントとは、予測が成功したことを示すイベントです。

Attributes	
Partitioning attributes	各分析モデルは複数のサブモデルで構成されており、それぞれのサブモデルは、パーティション化属性の別々の値組合せに対応しています。 ある属性をパーティション化属性にするデシジョンは、問題の属性の値に応じて、予測される可能性の抜本的相違に基づく必要があります。予測結果に対する影響があまり大きくない属性は、パーティション化でなく通常の入力として扱ってください。モデルのサイズは、パーティション化属性のカーディナリティに大きく左右されます。それぞれのカーディナリティが 10 のパーティショニング属性が 2 つある場合、それぞれのカーディナリティが 2 の場合と比べ、各属性のモデルの大きさは 25 倍になります。
Excluded attributes	デフォルトでは、セッション属性とエンティティ属性はすべて、モデル入力として使用されます。ある属性をモデルが入力として使用しないようにするには、その属性を「Excluded Attributes」リストに追加できます。あるエンティティ属性をすべてのモデルから除外するには、その属性自体のプロパティで「**Use for analysis**」オプションを無効にすることが最も簡単な方法です。

「Learn Location」タブと「Temporary Data Storage」タブは同じです。

Learn location	
Learn location	デフォルトでは、セッションの終了時に、すべてのモデルで学習が行われます。あるいは、個々のインフォーマントまたはアドバイザの統合点が呼び出されたときに学習を行うように指定することもできます。必要なデータをすべてセッションに格納し、セッション終了時にモデルに渡す方が、通常は効率的です。
Temporary data storage	かなりの時間を要するビジネス・プロセスでは、選択肢の結果がわかるのは、選択肢の作成から数週間後、場合によっては数か月後になることがあります。選択肢の結果が学習されたときに、エンティティ属性の元の値が確認できない場合があります。ただし、望ましいのは、すべての入力属性およびパーティション化属性で、選択肢の選択時とまったく同じ値を学習モデルに提供することです。 「**Use temporary data storage**」オプションが有効になっていると、モデルでは、ベース・イベントに対応するターゲット属性の値を受信するたびに、すべての入力属性およびパーティション化属性の値が格納されます。モデルでは、ほぼ同一のデータベース・レコードが、「Keys」リストにある各セッション・キーに 1 つずつ作成されます。 一時データ記憶域が有効になっていると、入力属性の値を気にせずに、選択肢の結果をモデルに提供できます。すべての入力モデル属性の値は、以前に格納されたレコードから取得されます。

Days to Keep	一時データをサーバーに格納しておく日数を指定します。
Keys	一時データ記憶域に格納されているデータは、データ格納域タブで定義されているキーのいずれか 1 つがあれば取得できます。
Advanced	「**Advanced**」ボタンを使用すると、Decision Center に要素を表示し、その要素のラベルを変更することもできます。要素のラベルを変更しても、オブジェクト ID は変更されません。

2.13.4 その他のモデル属性

「**Use explicit base**」オプションを指定すると、デシジョン（ベース・イベント）が明示的に通知されると予想し、個々のデシジョン結果に関する通知は受けないと想定するように、モデルに伝えられます。

選択肢イベント・モデルは常に、明示的なベース・イベントで定義されるため、「**Use explicit base**」オプションは汎用モデルにのみ表示されます。

ターゲット属性は予測の対象です。このモデルは、ターゲット属性の値が様々になる可能性があることを予測できます。モデルのパフォーマンス向上のために、ターゲット属性に考えられる値をすべて、「**Possible Values**」フィールドで指定することを強くお薦めします。

2.13.5 パーティション化と集計について

各分析モデルは複数のサブモデルで構成されており、それぞれのサブモデルは、パーティション化属性の別々の値組合せに対応しています。ある属性をパーティション化属性にするデシジョンは、問題の属性の値に応じて、予測される可能性の抜本的相違に基づく必要があります。予測結果に対する影響があまり大きくない属性は、パーティション化でなく通常の入力として扱ってください。モデルのサイズは、パーティション化属性のカーディナリティに大きく左右されます。それぞれのカーディナリティが 10 のパーティショニング属性が 2 つある場合、それぞれのカーディナリティが 2 の場合と比べ、各属性のモデルの大きさは 25 倍になります。

デフォルトでは、セッション属性とエンティティ属性はすべて、モデル入力として使用されます。ある属性をモデルが入力として使用しないようにするには、その属性を「**Excluded Attributes**」リストに追加できます。あるエンティティ属性をすべてのモデルから除外する必要がある場合は、その属性自体のプロパティで「**Use for Analysis**」オプションを無効にすることが最も簡単な方法です。

パーティション化属性の 1 つを、集計済としてマークできます。これにより、集計済の属性を除く追加のサブモデルが作成されます。たとえば、`choice` 属性を集計済とすると、そのモデルは、任意の選択肢に対するレスポンスが肯定的になる可能性を予測する目的で使用できます。

2.13.6 モデルの API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

モデルのクエリー

モデルは、次に示す getChoiceEventLikelihood メソッドのいずれかでクエリーすることができます。このクエリーでは、ある選択肢がモデルによって選択される可能性が返されます。

```
public static SDDoubleArray getChoiceEventLikelihoods(GENOffersChoice
choice);

public static SDDoubleArray getChoiceEventLikelihoods(GENOffers
choiceGroup) ;

public static double getChoiceEventLikelihoods(GENOffersChoice choice,
String eventName);

public static double getChoiceEventLikelihoods(GENOffers choiceGroup,
String eventName);
```

選択肢をモデルとともに記録

選択肢イベント・モデルの場合、選択肢のメソッド recordEvent がコールされたときに、モデルのメソッド recordEvent が実行されます。したがって、このメソッドをモデル上で直接起動する必要はありません。このメソッドは通常、選択肢がコール元のアプリケーションに拡張された統合点内からコールされます。

たとえば、アドバイザの統合点では、次のようになります。

```
if (choices.size() > 0) {

  Choice ch = choices.get(0);

  ch.recordEvent("Presented");

  session().setOfferExtended(ch.getSDOId());

}
```

この選択肢モデルに使用できる API は次のとおりです。

```
public static SDStringArray getChoice()

public static void setChoice(SDStringArray _v)

public static void addToChoice(String _a)

public static void addAllToChoice(SDStringArray _c)
```

インフォーマントは通常、選択肢をモデルとともに記録します。たとえば、通話理由コードの選択肢をモデル Reason Analysis とともに記録する場合は、次のようになります。

```
if (code == 17)

  ReasonAnalysis.addToChoice("BalanceInquiry");

else if (code == 18)

  ReasonAnalysis.addToChoice("MakePayment");

else if (code == 19)

  ReasonAnalysis.addToChoice("RateInquiry");

else

  ReasonAnalysis.addToChoice("Other");
```

選択肢が相互排他的とマークされていない場合は、選択肢を記録する前に、`getModelData()`に対するコールをこのコールに含める必要があります。

```
if (code == 17)

  ReasonAnalysis.getModelData().addToChoice("BalanceInquiry");

else if (code == 18)

  ReasonAnalysis.getModelData().addToChoice("MakePayment");

else if (code == 19)

  ReasonAnalysis.getModelData().addToChoice("RateInquiry");

else

  ReasonAnalysis.getModelData().addToChoice("Other");
```

選択肢配列を処理している場合は、まず空の文字列をモデルに送信する必要があります。

```
ReasonAnalysis.getModelData().addToChoice("");
```

2.14 統合点について

統合点は、Oracle RTD 内で、データとプロセスという 2 つの観点からの機能を果たします。

データの観点では、統合点はエンティティ属性に値を提供します。統合点の定義には、入力される値をセッション属性またはエンティティ属性に割り当てるためのマッピングが含まれています。

プロセスの観点では、統合点は、プロセスを実装する様々なシステムを経由して渡される際に、1つの単位に従うように定義されます。一般的には、単位を特定できる最も早い時点を特定することをお薦めします。プロセスのその時点で、インフォーマントがエンタープライズ運用システムによってコールされます。システムからインフォーマントにリクエストが送信され、それによってインライン・サービスではセッションの形成を開始でき、また必要に応じて、いずれかのデータソースの情報をプリフェッチできます。

次に、プロセスの中で、興味深い情報や測定が判明するその他の時点が特定され、それに対してインフォーマントが定義されます。

インライン・サービスは、観察のみを行うモードで実行される場合もあります。その場合は、アドバイザはなく、インフォーマントのみが存在します。このモードは、プロセスに関する情報の収集、および最適化されていないパフォーマンスの測定に便利です。この場合、インフォーマントは観察結果をモデルとともに記録し、モデル側からデータの相関関係および傾向を確認できます。

プロセス内で Inline Intelligence デシジョンが運用システムに提供される各時点に対して、アドバイザが定義されます。

外部システムおよび順序番号も、統合点ごとに定義されます。これらは、Decision Center で表示されるプロセス・マップの生成に使用します。システムによって、スイムレーンと位置の順序（左から右）が決定されます。この順序は、整数のみでなくどんな数値でも可能であり、既存の統合点を変更しないで新しい統合点を導入できます。

統合点にアクセスする運用システムの詳細は、『Oracle Real-Time Decisions 統合ガイド』を参照してください。

2.14.1 インフォーマントについて

インフォーマントは、そのモデルによって収集されたデータおよび分析からレポートを公開します。その分析のターゲットは、1 つ以上の選択肢グループにある選択肢です。インフォーマントには、処理およびモデルへの公開に必要なロジックが含まれています。エンティティが、意思決定および分析のためにデータをオブジェクトにまとめる役割を果たします。

2.14.2 インフォーマントの機能について

インフォーマントは、分析および分析モデルのターゲットとなる選択肢グループと協同して、分析を実行します。一般的に、インフォーマントをインライン・サービスに追加する手順は次のとおりです。

1. 外部システムを作成して、どのシステムが統合点にアクセスするかを特定します。
2. 分析のターゲットを表す選択肢グループを作成します。たとえば、サービス・センターへの通話理由を表す選択肢グループなどです。
3. セッション・キー情報を受信し、セッションに基づいてデータを収集および処理するインフォーマントを作成します。
4. データのリポジトリであり、データを分析する分析モデルを作成します。

インフォーマントの特徴は次のとおりです。

Description	インフォーマントの説明。
Request	
Session Keys	セッションの開始と終了を特定するために使用される 1 つ以上のセッション・キー。メッセージ内のどのセッション・キーでも、セッションを十分に特定できるため、このメッセージに関連する情報をすでに含むセッションが存在する場合、メッセージはそのセッションにディスパッチされます。
External System	インフォーマントにリクエストを送信する外部システムを特定します。インフォーマントと外部システムを関連付けると、そのインフォーマントを、Decision Center のプロセス・マップに、他のインフォーマントおよびアドバイザとともに表示できます。
Order	この番号は、Decision Center のプロセス・マップに表示される一連の統合点の中での、このインフォーマントの位置を特定します。別の統合点の順序より少ない順序の統合点は、他の統合点の前に表示されます。この順序は十進数です。たとえば、2.1 は 2.2.の前に表示されます。
Force Session Close	これを選択すると、このインフォーマントの非同期ロジックがすべて実行された後に、このインフォーマントのセッションが、インライン・サービスによって自動的に停止されます。インフォーマントの「Logic」タブのサブタブ内のどこかに Java 文 `session().close();` を配置しても、同じ効果が得られます。

セッション・キーの追加

「Request」タブで**「Select」**をクリックして、統合点のセッション・キーを選択します。これは、運用システムから統合点に提供される値の 1 つです。

外部システムおよび順序の特定

「Request」タブでドロップダウン・リストを使用して、統合点にアクセスする外部システムを選択します。このメニューは、外部システム要素を使用して外部システム識別子を作成することによって移入されます。

統合点にアクセスする順序は、**「Order」**に表されます。この番号と**「External System」**によって、Decision Center でのエンドツーエンド・プロセスの表示が決まります。

リクエスト・データの追加

「Request」タブで**「Add」**をクリックして、リクエスト・データを追加します。Assignment は、運用システムから統合点に提供される値です。Assignment の特徴は次のとおりです。

Incoming Parameter	インフォーマントに送信されるリクエスト内のフィールドの名前。このフィールドの値が、リクエストからセッション属性にコピーされます。この名前は、セッション属性と同じである必要はありませんが、普通は同じ名前が付けられます。 セッション・キーが作成されると、入力されたパラメータがセッション・キーの属性に割り当てられます。
Type	これはセッション属性のデータ型で、入力された引数はそこにコピーされます。有効な型は、integer、string、date または double です。 **注意:** リクエスト・フィールドの型とセッション・キーの属性が一致しない場合は、変換メソッドを使用する必要があります。
Array	型がコレクションの場合にマークされます。
Session Attribute	リクエストの入力パラメータのマップ先となるセッションの属性。

2.14.3 インポートされた Java クラスの追加と Decision Center の表示の変更

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある**「Advanced」**をクリックします。Decision Center の表示ラベルを変更して、その要素を Decision Center Navigator に表示するかどうかを選択することもできます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.14.4 インフォーマントの API

```
public String getSDOLabel();

public String getSDOId();
```

それぞれ、オブジェクトのラベルと ID を返します。

2.14.5 インフォーマントのロジック

ロジック

このスクリプトは、Request Data で宣言されたいずれかのリクエスト・データの実行後に実行されます。インフォーマントの主要な目的が、運用システムのリクエスト・フィールドからセッション・キーおよびリクエスト・データにデータを転送することである場合、この処理は「Request Data」タブでの宣言に従って自動的に行われるため、ロジックが不要な場合があります。

インフォーマントのロジックは通常、ログ・ファイルでメッセージ受信をトレースするため、またはインフォーマントのメッセージによってキーが提供されるエンティティを事前移入するために使用します。ロジックを使用すると、レスポンス時間がより重要なアドバイザで、後にこの処理を行う必要がなくなります。ロジックは、リクエスト・データの直後に実行されます。

インフォーマントのロジックは、選択肢を選択肢モデルとともに記録する場合にも使用できます。コールするメソッドについては、選択肢モデルの API を参照してください。

非同期ロジック

このスクリプトは、前述のロジックで定義したスクリプトの後に実行されます。ほかに必要な処理があったら、この領域に配置できます。非同期ロジックの実行順序は保証されません。

2.14.6 モデルとインフォーマントについて

インライン・サービスは、観察のみを行うモードで実行される場合もあります。その場合は、アドバイザはなく、インフォーマントのみが存在します。このモードは、プロセスに関する情報の収集、および最適化されていないパフォーマンスの測定に便利です。この場合、インフォーマントのロジックは一般的に、選択肢のメソッドを使用してイベントをモデルに記録します。

インフォーマントからのリクエスト・データへのアクセス

インフォーマントからリクエスト・データには、様々な方法でアクセスできます。入力されたパラメータがセッション属性にマップされる場合は、そのパラメータに `get` メソッドがあります。

```
request.get$()
```

この`$`は、先頭が大文字のパラメータ名です。

この属性がマップされない場合は、パラメータのフィールド名を使用して、同じ結果をもたらすメソッドがあります。

```
String request.getStringValue(fieldName)

SDStringArray request.getStringArrayValue(fieldName)

boolean request.isArgPresent(fieldName)
```

2.14.7 アドバイザについて

プロセス内で Inline Intelligence デシジョンが要求される各時点に対して、アドバイザが定義されます。通常は、アドバイザごとに、最適化されたデシジョンと Control Group 用の 2 つのデシジョン・オブジェクトが利用されます。

Control Group デシジョンは、最適化されたデシジョンの比較基準となるように、既存のビジネス・プロセスにできるだけ近いものにする必要があります。

デシジョンに加えて、デフォルトの選択肢をアドバイザに定義できます。これらの選択肢は、サーバーでの計算を時間内に完了できない場合や、クライアントとサーバーの通信が切断された場合に使用されます。

選択肢グループと選択肢にはルールがあります。このルールによって、選択肢または選択肢グループとその子をすべて、デシジョンの結果として、アドバイザのコール元に送信できる状況が決まります。これらのルールは、セッションまたはエンティティの値、または選択肢の設定を参照できます。たとえば、現在の顧客グループがエンティティ属性で、グループが選択肢属性の場合に、一定のグループの顧客のみが選択肢を使用できます。

2.14.8 アドバイザの意思決定プロセスについて

アドバイザは、デシジョンのターゲットとなる選択肢グループと協同します。関数、スコアリング・ルール、分析モデルまたは定数の形式で選択肢をスコアリングすると、どの選択肢が適切かを判断する際に役立ちます。次に、デシジョンでは、組織で定義されているパフォーマンス目標に従って、スコアが重み付けされます。結果としてスコアの最も高い選択肢が、レスポンスとしてアドバイザに渡されるデシジョンです。意思決定プロセスの詳細は、第2.11項「デシジョン・プロセスについて」を参照してください。

アドバイザの特徴は次のとおりです。

Description	アドバイザの説明。
Request	
Session Keys	セッションの開始と終了を特定するために使用される 1 つ以上のセッション・キー。メッセージ内のどのセッション・キーでも、セッションを十分に特定できるため、このメッセージに関連する情報をすでに含むセッションが存在する場合、メッセージはそのセッションにディスパッチされます。 アドバイザがコールされると、まずセッション・キーの作成が実行されます。
External System	アドバイザにリクエストを送信する外部システムを特定します。アドバイザと外部システムを関連付けると、そのアドバイザを、Decision Center のプロセス・マップに、他のインフォーマントおよびアドバイザとともに表示できます。
Order	この番号は、Decision Center のプロセス・マップに表示される一連の統合点の中での、このアドバイザの位置を特定します。別の統合点の順序より少ない順序の統合点は、他の統合点の前に表示されます。この順序は十進数です。たとえば、2.1 は 2.2.の前に表示されます。
Force Session Close	このオプションを選択すると、このアドバイザの非同期ロジックがすべて実行された後に、このアドバイザのセッションが、インライン・サービスによって自動的に停止されます。アドバイザの「Logic」タブのサブタブ内のどこかに Java 文 `session().close();` を配置しても、同じ効果が得られます。

2.14.9 インポートされた Java クラスの追加と Decision Center の表示の変更

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある「**Advanced**」をクリックします。Decision Center の表示ラベルを変更して、その要素を Decision Center Navigator に表示するかどうかを選択することもできます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.14.10 セッション・キーの追加

「Request」タブで**「Select」**をクリックして、統合点のセッション・キーを選択します。これは、運用システムから統合点に提供される値の 1 つです。

2.14.11 外部システムおよび順序の特定

「Request」タブでドロップダウン・リストを使用して、統合点にアクセスする外部システムを選択します。このリストは、外部システム要素を使用して外部システム識別子を作成することによって移入されます。

統合点にアクセスする順序は、**「Order」**に表されます。この番号と**「External System」**によって、Decision Center でのエンドツーエンド・プロセスの表示が決まります。

2.14.12 リクエスト・データの追加

「Request」タブで**「Add」**をクリックして、リクエスト・データを追加します。リクエスト・データは、運用システムから統合点に提供される値です。リクエスト・データの特徴は次のとおりです。

Request Data	
Incoming Parameter	アドバイザに送信されるリクエスト内のフィールドの名前。このフィールドの値が、リクエストからセッション属性にコピーされます。この名前は、セッション属性と同じである必要はありませんが、普通は同じ名前が付けられます。 セッション・キーが作成されると、入力されたパラメータがセッション属性に割り当てられます。
Type	これはセッション属性のデータ型で、入力された引数はそこにコピーされます。有効な型は、integer、string、date または double です。 **注意:** リクエスト・フィールドの型とセッション属性が一致しない場合は、変換メソッドを使用する必要があります。
Array	このオプションは、型がコレクションの場合に選択します。
Session Attribute	リクエストの入力パラメータのマップ先となるセッションの属性。

2.14.13 レスポンス・データの追加

「Response」タブで**「Add」**をクリックして、レスポンス・データを追加します。レスポンス・データは、リクエストを呼び出した後に、運用システムから統合点に返信される値です。レスポンス・データの特徴は次のとおりです。

Response	レスポンスには、選択した選択肢オブジェクトの配列が含まれ、各選択肢には、名前付きの属性値のコレクションが含まれています。選択肢の選択プロセスは、アドバイザで参照されている 2 つのデシジョン・オブジェクトのいずれか一方によって制御されます。1 つのデシジョンが、コール元のアプリケーションに与えられます。
Decision to Use	Control Group のセッションではなく、通常のセッションに使用するデシジョン・オブジェクトの名前。このデシジョンが、アドバイザからコール元システムへのレスポンスになります。
Control Group Decision to Use	Control Group デシジョンは、ベースラインを提供することによって他のデシジョンの効果を評価する方法として、少数のセッションのみに使用されます。Control Group デシジョンを使用するセッションの割合は、インライン・サービスのアプリケーション要素で指定します。Control Group デシジョンは、いつもどおりのロジックを使用して選択肢を選択するように設計する必要があります。つまり、インライン・サービスの導入前にエンタープライズで使用されていたルールです。Decision Center コンソールで、アドバイザの通常のデシジョン・オブジェクトのビジネス効果を Control Group デシジョンと比較するレポートを使用できます。
Parameters	デシジョンによって定義される入力パラメータ。「Name」列と「Type」列は説明のみで、デシジョン・オブジェクトからここに表示されます。
Default number of choices returned	デシジョンによって返されるデフォルトの選択肢数。これは、デシジョンで定義されている選択肢の数です。
Override default with	アドバイザでは、参照先のデシジョンで指定されている数値を上書きまたはそのまま使用できます。ここでは、アドバイザのレスポンスに含めることのできる選択肢の最大数を指定します。
Default Choices	コール元のクライアント・アプリケーションがアドバイザを起動しようとしたとき、サーバーの保証されているレスポンス時間内にアドバイザがレスポンスを提供できない場合に、コール元のクライアント・アプリケーションに返される選択肢のリスト。 デフォルトの選択肢は、アドバイザごとに指定する必要はありません。インライン・サービスでもデフォルトの選択肢を宣言することがありますが、これは独自の宣言を行わないアドバイザで使用されます。また、デフォルトの選択肢構成は、クライアント・アプリケーションに伝播され、Smart Client コンポーネントによってローカルのファイル・システムに格納されます。したがって、その後は、サーバーに接続できないクライアント・アプリケーションで使用できるようになります。

2.14.14 アドバイザのロジック

ロジック

このスクリプトは、「Request Data」タブで宣言されたいずれかのリクエスト・データが実行されてから、クライアントにレスポンスが返信されるまでの間に実行されます。

アドバイザのロジックは、一般的には不要です。これは、リクエストとともに着信するデータを事前処理するため、またはデバッグ目的に使用します。

非同期ロジック

このスクリプトは、クライアントにレスポンスを返信するサーバー側メカニズムにレスポンスが渡された後に実行されます。クライアントで使用されているエンドポイントのタイプによっては、このスクリプトが終了する前にクライアントが結果の処理を開始し、並列性を強化することによって効果的なレスポンス時間を改善できます。

2.14.15 アドバイザからリクエスト・データへのアクセス

アドバイザからリクエスト・データには、様々な方法でアクセスできます。入力されたパラメータがセッション属性にマップされる場合は、そのパラメータに `get` メソッドがあります。

```
request.get$()
```

この`$`は、先頭が大文字のパラメータ名です。

この属性がマップされない場合は、同じ結果をもたらすメソッドがいくつかあります。

```
String request.getStringValue(fieldName)

SDStringArray request.getStringArrayValue(fieldName)

boolean request.isArgPresent(fieldName)
```

2.15 外部システムについて

外部システムは、Decision Studio 内でのみ識別されます。外部システムは、インライン・サービスに統合する、エンタープライズ内の運用システムを表します。外部システムは、API 経由ではアクセスできません。外部システムは、統合点で、どの外部システムがその統合点にアクセスするかを識別するために使用します。外部システムは、Decision Center の統合マップでの表示に使用されます。

外部システムの特徴は次のとおりです。

Description	システムの説明。
Display Label	表示ラベルを変更できます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.16 カテゴリ・オブジェクトについて

カテゴリは、選択肢の整理に使用します。同じカテゴリにある選択肢はすべて、Decision Center に一緒に表示されます。カテゴリには、クラスは生成されません。カテゴリは、Decision Center で、選択肢のグループ分けおよび整理のみに使用します。

カテゴリの特徴は次のとおりです。

Name	Decision Studio に入力されるカテゴリの名前。
Description	Decision Studio に入力されるカテゴリの説明。
Display Label	表示ラベルを変更できます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

2.17 関数について

関数は、計算など、再利用可能にする処理に使用できます。関数は、Decision Studio を使用して定義します。関数定義の特徴は次のとおりです。

Description	関数の説明。
Return value	関数が値を返すかどうかを指定します。
Data Type	戻り値の型。
Array	このオプションは、戻り型が配列の場合に選択します。
Call Template	関数のコール方法の定義。`{0}`や`{1}`などを引数として使用し、関数を説明する表現を設定して、コール用のテンプレートを定義します。この表現は関数使用時に表示されるため、わかりやすい表現を使用することが重要です。たとえば、multiply（乗算）のコール・テンプレートは`{0} multiplied by {1}`です。
Parameters	関数のロジックで使用される名前付きパラメータ。この数値は、コール・テンプレートにある引数の数と一致する必要があります。たとえば、multiply には、`a, type Double; b, type Double` のようなパラメータがあります。
Logic	関数の Java コード。multiply のコードは次のとおりです。 `return a * b;`

2.17.1 インポートされた Java クラスの追加と Decision Center の表示ラベルの変更

インポートされた Java クラスをインライン・サービスに追加するには、説明の横にある「**Advanced**」をクリックします。Decision Center の表示ラベルを変更して、その要素を Decision Center Navigator に表示するかどうかを選択することもできます。表示ラベルを変更しても、オブジェクト ID には影響がありません。

関数は、コール・テンプレートを使用して、他の要素からコールされます。たとえば、前述の `multiply` 関数を使用する場合は、関数を「**Edit Value**」ダイアログから選択します。コール・テンプレート `{0} multiplied by {1}` を使用すると、引数の位置と数がエディタに与えられます。

2.18 統計コレクタについて

統計コレクタは、インライン・サービスのイベント統計のコレクションおよびライフサイクルを管理します。インライン・サービスごとに、選択肢イベントの統計コレクタが 1 つずつ作成されます。選択肢イベントの統計コレクタは、インライン・サービスで定義されているイベントの統計を自動的に収集します。統計コレクタのプロパティは次のとおりです。

Description	統計コレクタの説明。
Collect Statistics On	統計は、選択肢や選択肢グループなどのオブジェクトごとに個別に収集することも、同じタイプのすべてのオブジェクトに集計して収集することもできます。
Aggregation	個々のイベントを記録するか、または集計されたデータを記録します。個々のイベントを記録する場合は、トランザクションの多いシステムにパフォーマンス上の問題が発生する場合があるため、注意が必要です。
Aggregation Interval	データを記録する前の集計に要する時間（秒）。
Expiration	「**Keep forever**」または「**Purge old statistics**」を選択します。「**Keep forever**」を選択する場合は、データのサイズが問題になる場合があるため、注意が必要です。
Keep in database for	データを消去するまでの時間（日）。

パラメータはすべて、Decision Studio エディタを使用して構成できます。選択肢イベントの統計は、レポートとして Decision Center に表示されます。

2.18.1 カスタム統計コレクタの作成

Decision Studio を使用して、オブジェクトまたはクラスに関する追加の統計を記録するカスタム統計コレクタを作成できます。たとえば、選択肢に関する統計を記録する統計コレクタを作成できます。パラメータは、前項の説明に従って構成します。

インライン・サービスのコードで（たとえば、インフォーマントまたは関数コール経由で）、統計コレクタのファクトリを作成し、コレクタ名（String）または統計タイプ（String）を渡します。

```
StatisticCollectorFactory factory =
Application.getCollectorFactory(<stat collector name | statistic
type>);
```

このファクトリを使用して、コレクタを作成し、統計を収集するイベント名（String）または統計名（String）を渡します。

```
StatCollectorInterface collector = factory.getCollector(<event name |
statistic name> );
```

イベント名または統計名は、収集対象を表す任意の文字列です。

最後に、コレクタを使用して、object_type（String）、object_id（String）、イベント値（double）および追加データ（string）を渡してイベントを記録します。

```
Collector.recordEvent(<object_type>, <object_id>, event value, extra
data);
```

object_type には、選択肢、選択肢グループ、エンティティなど、有効なオブジェクト・タイプを指定する必要があります。object_id は、オブジェクトの内部名です。

2.19 Decision Center のパースペクティブについて

Decision Studio と同様に、Decision Center を使用すると、インライン・サービスを様々なパースペクティブから操作できます。パースペクティブは、Decision Center での初期レイアウトを定義します。パースペクティブごとに、特定のタイプのタスク達成を目的とした一連の機能があり、特定のタイプのリソースに対応します。パースペクティブによって、一部のメニューおよびツールバーの表示内容が制御されます。

Decision Center には、デフォルトのパースペクティブとして、「Explore」、「Design」、「At a Glance」の 3 つがあります。インライン・サービス・ナビゲータは、使用しているパースペクティブに従って変わります。システム管理者がパースペクティブを追加している場合もあります。

Decision Center のパースペクティブへのアクセスは、グループおよびユーザーに権限を割り当てることによって制御できます。グループおよびユーザーの管理の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

パースペクティブの権限を割り当てるには、Decision Studio のインライン・サービス・ナビゲータに進み、アクセスを設定するパースペクティブを右クリックします。「**Properties**」を選択して「**Add**」をクリックします。権限を割り当てるグループを選択して、「**Permissions**」の下の「**Use perspective**」を選択します。「**OK**」をクリックして終了します。

3　Oracle RTD の一般的な API

この章では、Oracle RTD の一般的な API について説明します。

この章の内容は次のとおりです。

3.1　com.sigmadynmics.util
クラス Null

Null のテスト用の、Null というユーティリティ・クラス。

---

isNull

public static boolean isNull(String val)

パラメータ:

val: null のテストを行う String 値

戻り値:

値が null の場合は true、null でない場合は false。

---

isNull

public static boolean isNull(boolean val)

パラメータ:

val: null のテストを行う boolean 値

戻り値:

値が null の場合は true、null でない場合は false。

---

isNull

public static boolean isNull(long val)

パラメータ:

val: null のテストを行う long 値

戻り値:

値が null の場合は true、null でない場合は false。

---

isNull

public static boolean isNull(float val)

パラメータ:

val: null のテストを行う float 値

戻り値:

値が null の場合は true、null でない場合は false。

---

isNull

public static boolean isNull(int val)

パラメータ:

val: null のテストを行う int 値

戻り値:

値が null の場合は true、null でない場合は false。

オブジェクト

3.2 com.sigmadynamics.support クラス SDOBase

ログに記録するメソッドは、SDO ベース・クラスで定義します。これらのメソッドは、どのインライン・サービスでも使用できます。

ロギングのレベルには、情報、警告、デバッグの 3 つがあります。インライン・サービスでは、デフォルトで情報レベルが有効および使用可能になっており、エラー・ログに出力されます。他のレベルを有効にするには、JConsole を使用します。JConsole の使用方法の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

---

logDebug

public static void logDebug(String msg)

String メッセージを、デバッグ・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

---

logDebug

public static void logDebug(String msg, Object[] args)

String メッセージおよび Object 配列を、デバッグ・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

---

logDebug

public static void logDebug(String msg, Throwable t)

String メッセージおよび例外を、デバッグ・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

t: ログに記録する例外。

---

logDebug

public static void logDebug(String msg, Object[] args, Throwable t)

String メッセージ、Object 配列および例外を、デバッグ・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

t: ログに記録する例外。

---

logInfo

public static void logInfo (String msg)

String メッセージを、情報レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

---

logInfo

public static void logInfo(String msg, Object[] args)

String メッセージおよび Object 配列を、情報レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

---

logInfo

public static void logInfo(String msg, Throwable t)

String メッセージおよび例外を、情報レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

t: ログに記録する例外。

---

logInfo

public static void logInfo(String msg, Object[] args, Throwable t)

String メッセージ、Object 配列および例外を、情報レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

t: ログに記録する例外。

---

logWarning

public static void logWarning(String msg)

String メッセージを、警告レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

---

logWarning

public static void logWarning(String msg, Object[] args)

String メッセージおよび Object 配列を、警告レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

---

logWarning

public static void logWarning(String msg, Throwable t)

String メッセージおよび例外を、警告レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

t: ログに記録する例外。

---

logWarning

public static void logDebug(String msg, Object[] args, Throwable t)

String メッセージ、Object 配列および例外を、警告レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

t: ログに記録する例外。

---

logError

public static void logError(String msg)

String メッセージを、エラー・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

---

logError

public static void logError(String msg, Object[] args)

String メッセージおよび Object 配列を、エラー・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

---

logError

public static void logError(String msg, Throwable t)

String メッセージおよび例外を、エラー・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

t: ログに記録する例外。

---

logError

public static void logError (String msg, Object[] args, Throwable t)

String メッセージ、Object 配列および例外を、エラー・レベルでログに記録します。

パラメータ:

msg: ログに記録する String 値。

args: ログに記録する Object 配列。Object 配列はまず、toString を使用して String 値に変換されます。

t: ログに記録する例外。

---

logError

public static void logError (Throwable t)

例外を、エラー・レベルでログに記録します。

パラメータ:

t: ログに記録する例外。

3.3 com.sigmadynamics.util クラス StringUtil

StringUtil クラスには、様々な型を読取り可能な文字列の形式で返すメソッドが含まれています。これはデバッグに便利です。

3.4 com.sigmadynamics.util クラス DateUtil

DateUtil クラスには、様々な形式に書式設定された日付を返すメソッドが含まれています。

---

getCalendar

public static Calendar **getCalendar**(long datetime)

1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）を Calendar オブジェクトに変換します。

パラメータ:

datetime: 1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）。

戻り値:

日付を表す Calendar オブジェクト。

関連項目: Calendar

---

formatDate

public static String formatDate(long date)

1970 年 1 月 1 日（UTC）を起点とした日付（ミリ秒）を文字列表現に変換します。

パラメータ:

date: 1970 年 1 月 1 日（UTC）を起点とした日付（ミリ秒）。

戻り値:

日付は短い文字列形式（米国ロケールでは「4/21/03」など）。

---

formatTime

public static String formatTime(long time)

1970 年 1 月 1 日（UTC）を起点とした時刻（ミリ秒）を文字列表現に変換します。

パラメータ:

time: 1970 年 1 月 1 日（UTC）を起点とした時刻（ミリ秒）。

戻り値:

時刻の値は短い文字列形式（米国ロケールでは「8:35 pm」など）。

---

formatDateTime

public static String formatDateTime(long datetime)

1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）を文字列表現に変換します。

パラメータ:

datetime: 1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）。

戻り値:

日時は短い文字列形式（米国ロケールでは「4/21/03 8:35 pm」など）。

---

format

public static String format(long datetime, DateFormat format)

用意されている書式仕様に従って、1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）を書式設定します。

パラメータ:

datetime: 1970 年 1 月 1 日（UTC）を起点とした日時（ミリ秒）。

format: 書式仕様。

戻り値: 書式設定された日時文字列。

関連項目: DateFormat

3.5 com.sigmadynamics.util クラス SDArray

配列クラスには、様々なものが用意されています。これらのベース・クラスが SDArray です。

パブリック抽象クラス SDArray

Object を拡張

Serializable を実装

この抽象クラスが、専用の SDArrayType クラスのベースになります。

定数フィールドの値

protected static final int ALLOC_UNIT

protected static final int DEFAULT_INITIAL_SIZE

protected int size

コンストラクタの詳細

SDArray

public SDArray()

デフォルトのコンストラクタ。

---

size

public final int size()

配列内の要素の数を返します。

戻り値:

要素の数。

---

isEmpty

public final boolean isEmpty()

配列に要素が含まれていない場合は true、含まれている場合は false を返します。

戻り値:

配列が空の場合は true、空でない場合は false。

---

capacity

public abstract int capacity()

戻り値:

バッファ容量。

---

setSize

public abstract void setSize(int size)

配列の最後で、要素を追加または削除します。

パラメータ:

size: 新しい要素数。

---

clear

public void clear()

配列からすべての要素を削除します。

---

trimToSize

public abstract void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

---

ensureCapacity

protected abstract void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

パラメータ:

capacity: 対応する要素数。

3.6 com.sigmadynamics.util クラス SDBooleanArray

boolean 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス SDBooleanArray

SDArray の拡張版

フィールドの詳細

buf

protected boolean[] buf

コンストラクタの詳細

SDBooleanArray

public SDBooleanArray()

空の配列を構成します。

SDBooleanArray

public SDBooleanArray(int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

---

capacity

public int capacity()

戻り値:

バッファ容量。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public boolean get(int index)

戻り値:

この配列内で指定された位置にある要素。

パラメータ:

index: 返す要素のインデックス。

---

set

public void set(int index, boolean val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

---

add

public void add(boolean element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDBooleanArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(boolean element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, boolean element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

---

contains

public boolean contains(boolean element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDBooleanArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDBooleanArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public boolean[] toArray()

boolean 型の要素をすべて含む配列を返します。

戻り値:

boolean 型の要素をすべて含む配列。

---

toArray

public void toArray(boolean[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。要素は、String.valueOf(boolean)によって文字列に変換されます。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(boolean element, boolean[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

3.7 com.sigmadynamics.util クラス SDDoubleArray

double 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス **SDDoubleArray**

SDArray の拡張版

フィールドの詳細

buf

protected double[] buf

コンストラクタの詳細

SDDoubleArray

public SDDoubleArray()

空の配列を構成します。

SDDoubleArray

public SDDoubleArray(int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

---

capacity

public int capacity()

バッファ容量を返します。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public double get(int index)

この配列内で指定された位置にある要素を返します。

パラメータ:

index: 返す要素のインデックス。

戻り値:

この配列内で指定された位置にある要素。

---

set

public void set(int index, double val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

---

add

public void add(double element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDDoubleArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(double element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, double element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

---

contains

public boolean contains(double element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDDoubleArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDDoubleArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public double[] toArray()

double 型の要素をすべて含む配列を返します。

戻り値:

double 型の要素をすべて含む配列。

---

toArray

public void toArray(double[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。要素は、String.valueOf(double)によって文字列に変換されます。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(double element, double[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

---

increment

public double increment(int index, double amount)

所定の数量で要素を増分します。

パラメータ:

index: 要素のインデックス。

amount: 要素に追加する値。

戻り値:

増分された値。

---

decrement

public double decrement(int index, double amount)

所定の数量で要素を減分します。

パラメータ:

index: 要素のインデックス。

amount: 要素から減じる値。

戻り値:

減分された値。

3.8 com.sigmadynamics.util クラス SDIntArray

integer 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス **SDIntArray**

SDArray の拡張版

フィールドの詳細

buf

protected double[] buf

コンストラクタの詳細

SDIntArray

public SDIntArray ()

空の配列を構成します。

SDIntArray

public SDIntArray (int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

SDIntArray

public SDIntArray (int[] elements)

このコンストラクタはパブリック API の一部ではないため、インライン・サービスのコードでは使用しないでください。

---

capacity

public int capacity()

バッファ容量を返します。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public double get(int index)

この配列内で指定された位置にある要素を返します。

パラメータ:

index: 返す要素のインデックス。

戻り値:

この配列内で指定された位置にある要素。

---

set

public void set(int index, int val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

val: 値。

---

add

public void add(int element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDIntArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(int element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, int element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

---

contains

public boolean contains(int element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDIntArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDIntArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public int[] toArray()

int 型の要素をすべて含む配列を返します。

戻り値:

int 型の要素をすべて含む配列。

---

toArray

public void toArray(int[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。要素は、String.valueOf(int)によって文字列に変換されます。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(int element, int[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

---

increment

public double increment(int index, int amount)

所定の数量で要素を増分します。

パラメータ:

index: 要素のインデックス。

amount: 要素に追加する値。

戻り値:

増分された値。

---

decrement

public double decrement(int index, int amount)

所定の数量で要素を減分します。

パラメータ:

index: 要素のインデックス。

amount: 要素から減じる値。

戻り値:

減分された値。

3.9 com.sigmadynamics.util クラス SDLongArray

long 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス **SDLongArray**

SDArray の拡張版

フィールドの詳細

buf

protected double[] buf

コンストラクタの詳細

SDLongArray

public SDLongArray ()

空の配列を構成します。

SDLongArray

public SDLongArray (int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

SDLongArray

public SDLongArray (int[] elements)

このコンストラクタはパブリック API の一部ではないため、インライン・サービスのコードでは使用しないでください。

---

capacity

public int capacity()

バッファ容量を返します。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public double get(int index)

この配列内で指定された位置にある要素を返します。

パラメータ:

index: 返す要素のインデックス。

戻り値:

この配列内で指定された位置にある要素。

---

set

public void set(int index, long val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

val: 値。

---

add

public void add(long element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDLongArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(long element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, long element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

---

contains

public boolean contains(long element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDLongArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDLongArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public long[] toArray()

long 型の要素をすべて含む配列を返します。

戻り値:

long 型の要素をすべて含む配列。

---

toArray

public void toArray(long[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。要素は、String.valueOf(long)によって文字列に変換されます。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(long element, long[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

3.10 com.sigmadynamics.util クラス SDStringArray

String 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス **SDStringArray**

SDArray の拡張版

フィールドの詳細

buf

protected double[] buf

コンストラクタの詳細

SDStringArray

public SDStringArray ()

空の配列を構成します。

SDStringArray

public SDStringArray (int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

SDLongArray

public SDLongArray (int[] elements)

このコンストラクタはパブリック API の一部ではないため、インライン・サービスのコードでは使用しないでください。

---

capacity

public int capacity()

バッファ容量を返します。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public double get(int index)

この配列内で指定された位置にある要素を返します。

パラメータ:

index: 返す要素のインデックス。

戻り値:

この配列内で指定された位置にある要素。

---

set

public void set(int index, String val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

val: 値。

---

add

public void add(String element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDStringArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(String element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, String element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

contains

public boolean contains(String element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDStringArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDStringArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public String[] toArray()

String 型の要素をすべて含む配列を返します。

戻り値:

String 型の要素をすべて含む配列。

---

toArray

public void toArray(String[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(String element, String[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

3.11 com.sigmadynamics.util クラス SDObjectArray

Object 型の要素を含むサイズ変更可能な配列の型保証実装。

パブリック・クラス **SDObjectArray**

SDArray の拡張版

フィールドの詳細

buf

protected double[] buf

コンストラクタの詳細

SDObjectArray

public SDObjectArray()

空の配列を構成します。

SDObjectArray

public SDObjectArray (int capacity)

指定した初期容量で、空の配列を構成します。

パラメータ:

capacity: 配列の初期容量。

---

capacity

public int capacity()

バッファ容量を返します。

指定を行うもの:

クラス SDArray 内の capacity

戻り値:

バッファ容量。

---

get

public double get(int index)

この配列内で指定された位置にある要素を返します。

パラメータ:

index: 返す要素のインデックス。

戻り値:

この配列内で指定された位置にある要素。

---

set

public void set(int index, Object val)

この配列内で指定された位置にある要素を、指定された要素に置き換えます。

パラメータ:

index: 置き換える要素のインデックス。

val: 値。

---

add

public void add(Object element)

指定された要素を、この配列の最後に追加します。

パラメータ:

element: この配列に追加される要素。

---

addAll

public void addAll(SDObjectArray array)

指定された配列にある要素をすべて、この配列の最後に追加します。

パラメータ:

array: この配列に追加される要素を含む配列。

---

fill

public void fill(Object element)

この配列にある要素をすべて、同じ値に置き換えます。

パラメータ:

element: この配列にある要素すべてを置き換える要素。

---

fill

public void fill(int fromIndex, int toIndex, Object element)

この配列にある連続要素のグループを、同じ値に置き換えます。置き換えられる要素のグループは、fromIndex と toIndex 直前の間にインデックスがある要素で構成されています。

パラメータ:

fromIndex: 置き換えられる最初の要素のインデックス。

toIndex: 置き換えられる最後の要素の後にあるインデックス。

element: fromIndex から toIndex までの要素を置き換える要素。

---

contains

public boolean contains(Object element)

指定した要素がこの配列に含まれている場合に true を返します。

パラメータ:

element: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素が存在している場合は true、存在していない場合は false。

---

containsAll

public boolean containsAll(SDObjectArray elements)

指定した配列の要素がすべて、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素がすべてこの配列に存在している場合は true、存在しない要素がある場合は false。

---

containsAny

public boolean containsAny(SDObjectArray elements)

指定した配列の要素のいずれかが、この配列に含まれている場合に true を返します。

パラメータ:

elements: この配列に存在しているかどうかがテストされる要素。

戻り値:

指定した要素のいずれかがこの配列に存在している場合は true、まったく存在していない場合は false。

---

sort

public void sort()

配列を昇順にソートします。

---

sort

public void sort(Comparator c)

指定したコンパレータで導かれた順序に従って、オブジェクトの配列をソートします。

パラメータ:

c: 配列の順序を決定するコンパレータ。この値を null にした場合は、要素の通常の順序が使用されます。

スローされるもの:

ClassCastException: 指定したコンパレータを使用して相互比較できない要素が配列に含まれている場合。

関連項目:

Comparator

---

equals

public boolean equals(Object anObject)

要素単位の比較。

パラメータ:

anObject: この配列と比較するオブジェクト。

戻り値:

配列が等しい場合は true、等しくない場合は false。

---

toArray

public Object[] toArray()

Object 型の要素をすべて含む配列を返します。

戻り値:

Object 型の要素をすべて含む配列。

---

toArray

public void toArray(Object[] array)

所定の配列に要素をコピーします。指定された配列が、この型の要素の数より短い場合は、指定された配列に適合する要素のみがコピーされます。指定された配列が、この型の要素の数より長い場合は、指定された配列の後部は変更されないままです。

パラメータ:

array: この型の要素がコピーされる配列。

スローされるもの:

ArrayStoreException: 指定された配列のランタイム・タイプが、この型のあらゆる要素のランタイム・タイプのスーパータイプでない場合。

---

setSize

public void setSize(int size)

配列の最後で、要素を追加または削除します。

指定を行うもの:

クラス SDArray 内の setSize

パラメータ:

size: 新しい要素数。

スローされるもの:

IllegalArgumentException: size が 0 未満の場合。

---

trimToSize

public void trimToSize()

実際の要素数を上回る余分のバッファ容量を削除します。

指定を行うもの:

クラス SDArray 内の trimToSize

---

toString

public String toString()

この配列の文字列表現を返します。文字列表現は、配列のうち、大カッコ（[]）で囲まれた要素で構成されます。隣接する要素は、文字列「, 」（カンマとスペース）で区切られています。要素は、String.valueOf(Object)によって文字列に変換されます。

戻り値:

この配列の文字列表現。

---

indexOf

public static int indexOf(Object element, Object[] array)

指定した要素が配列内で最初に出現した箇所のインデックスを返します。そのような要素が配列に含まれていない場合は-1 を返します。このメソッドでは、線形検索が使用されます。

---

ensureCapacity

protected void ensureCapacity(int capacity)

指定した要素数に対応するために、必要に応じて内部バッファを増やします。

指定を行うもの:

クラス SDArray 内の ensureCapacity

パラメータ:

capacity: 対応する要素数。

3.12 Studio でのデータ型

Studio で次のデータ型のメタデータを定義すると、次の Java クラスが生成されます。

メタデータの型	生成される Java タイプ
String	String
Double	double
Date	long。1970 年 1 月 1 日の午前 0 時を起点としたミリ秒で表現。

メタデータの型	**生成される Java タイプ**
ブール	ブール
Integer	int
Java クラス名	*Java クラス名*
<選択肢>（具体的）	<選択肢グループ ID>選択肢
<選択肢グループ>（具体的）	<*選択肢グループ ID*>
選択肢（汎用）	選択肢
選択肢グループ（汎用）	選択肢グループ
エンティティ ID	<*エンティティ ID*>

4　インライン・サービスのデプロイとテスト

この章では、インライン・サービスをデプロイおよびテストする方法について説明します。

この章の内容は次のとおりです。

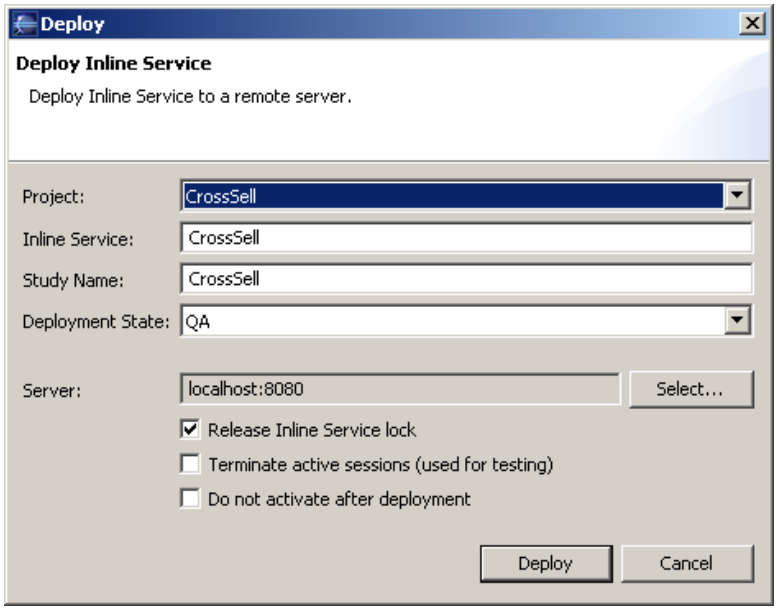
- 4.1 インライン・サービスのデプロイ
- 4.2 Real-Time Decision Server への接続
- 4.3 インライン・サービスの再デプロイ
- 4.4 インライン・サービスのテスト

4.1　インライン・サービスのデプロイ

構成の済んだインライン・サービスは、ローカルにデプロイするか、テストのためにテスト環境にデプロイします。インライン・サービスは、「QA」および「Production」の 2 つの状態にデプロイできます。

まず「QA」状態にデプロイし、その後テストが完了したら「Production」状態にデプロイします。「Production」状態にデプロイする際は、「**Release Inline Service locks**」を選択します。インライン・サービスをビジネス・ユーザー向けにデプロイすると、ビジネス・ユーザーもインライン・サービスを更新および再デプロイできるようになります。

インライン・サービスをデプロイするには、「**Project**」→「**Deploy**」のメニュー項目を使用するか、タスク・バーの「Deploy」  をクリックします。「Deploy」ダイアログには、次のいくつかのオプションがあります。



注意: インライン・サービスのデプロイには、サーバー・クラスタに対する適切な権限が必要です。クラスタ権限の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

Project	Real-Time Decision Server にデプロイするプロジェクトを選択します。
Inline Service	このプロジェクトに含まれているインライン・サービス。
Study Name	このインライン・サービスのスタディ名を入力します。各インライン・サービスの学習は、スタディ名に関連付けられています。インライン・サービスを再デプロイして学習を再開するには、新しいスタディ名を付けてデプロイします。使用するスタディ名は、「QA」と「Production」で別々にします。
Deployment State	デフォルトでは、インライン・サービスのデプロイの状態には、「Development」、「QA」、「Production」の 3 つがあります。他のユーザーにわかるように、「Deployment State」には、インライン・サービスの状態（開発、テストまたは本番）が示されています。 システム管理者が、カスタムのデプロイ状態を作成している場合があります。
Server	デプロイ先のサーバーおよびポートを入力するには、このオプションをクリックします。「Server」ダイアログ・ボックスで、デプロイ先のサーバー・クラスタに対してデプロイ権限を持つ有効なユーザー名およびパスワードを入力します。クラスタに対する認可は、管理者が JConsole を使用して付与します。
Release Application Lock	デプロイされたインライン・サービスは自動的にロックされ、再デプロイは、デプロイを行ったユーザーしかできません。開発とテストが完了したインライン・サービスを本番環境にデプロイする際、「**Release Application Lock**」を選択すると、Decision Center のユーザーがインライン・サービスを変更および再デプロイできるようになります。
Terminate Active Sessions	デプロイしているインライン・サービスが本番環境にある場合は、アクティブなセッションが存在している場合があります。アクティブなセッションが存在している状態で新しいバージョンのインライン・サービスをデプロイすると、古いバージョンが維持され、アクティブなセッションにサービスを提供し続けます。テスト中の場合は、**「Terminate Active Sessions」**を選択してアクティブなセッションを終了します。本番のインライン・サービスでは、このオプションを選択解除したままにして、アクティブなセッションが存在するとき、本番バージョンのインライン・サービスで実行され続けるようにします。新規のセッションは新しいバージョンにルーティングされ、古いバージョンはアクティブなセッションがすべて完了すると終了します。

Do not activate after deployment	このオプションは、インライン・サービスをサーバーにデプロイしてもプロセスを開始しないときに使用します。後日インライン・サービスをアクティブにする場合は、JConsole を使用します。JConsole の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

4.2 Real-Time Decision Server への接続

インライン・サービスをデプロイまたはダウンロードする際、またはデータソースをインポートする際には、Real-Time Decision Server に接続する必要があります。接続には、インストール時に作成したユーザー名およびパスワードを使用するか、管理者にユーザー名およびパスワードを問い合せてください。`https` を使用したセキュアな方法で接続するには、「**Secure connection**」を選択します。

4.3 インライン・サービスの再デプロイ

デプロイされているインライン・サービスを変更する場合は、その変更と、ビジネス・ユーザーにより行われている可能性のある変更の両方が失われないよう、次の手順に従うことが重要です。デプロイされているインライン・サービスを変更する場合は、ツールバーのダウンロード・アイコンを使用して Real-Time Decision Server からダウンロードできます。次の方法を使用します。

1. デプロイされている目的のインライン・サービスを編集しているビジネス・ユーザーがいないことを確認します。

2. ダウンロードする場合は必ずインライン・サービスをロックして、編集中にビジネス・ユーザーが変更できないようにします。

3. Decision Studio で変更作業を行います。

4. インライン・サービスを再デプロイしてロックを解除します。

インライン・サービスがロックされている間、ビジネス・ユーザーはそのデプロイされているインライン・サービスを表示できますが、編集はできません。

4.4 インライン・サービスのテスト

インライン・サービスのテスト用に、Load Generator を使用してデータを生成できます。Load Generator は、`RTD_HOME¥scripts¥loadgen.cmd` を実行して起動できます。Load Generator スクリプトのサンプルについては、Cross Sell サンプルの `etc` ディレクトリを参照してください。

4.4.1 Load Generator について

Load Generator は、ユーザーをシミュレートしてインライン・サービスのデバッグおよびベンチマークを行うためのツールです。Load Generator は、インライン・サービスのテストとパフォーマンスの特徴付けの両方に使用します。

Load Generator には、次の 4 つのタブがあります。

- **Run:** Load Generator のセッションを実行し、測定とグラフを使用してパフォーマンスのフィードバックを提供します。
- **General:** サーバーへのデータの送信速度やクライアントの構成ファイルの場所など、Load Generator の操作の一般的な設定を行います。
- **Variables:** スクリプト、メッセージおよびアクセス変数の作成に使用します。
- **Edit Script:** サーバーに送信する統合点リクエストを指定するスクリプトの設定に使用します。

Load Generator によるテスト

Load Generator は、サーバーに対して負荷を生成し、パフォーマンスおよびスケーラビリティをテストするために使用します。理にかなったランダムなメッセージがインライン・サービスに送信され、モデルが学習できます。Load Generator を十分な時間実行することによって、モデルの性能を測定できます。

Load Generator によるパフォーマンスの特徴付け

インライン・サービスを構成したら、Load Generator を使用し、特定の負荷がある状態で必要なサーバー数を判断するために、サービスの実行状況を評価します。Load Generator は多数のスレッドの実行を扱い、複数のスクリプトを同時に実行できるので、サーバーに負荷をかけるとき、通常は 1 台のクライアント・マシンで Load Generator を 1 インスタンス実行するのみで十分です。さらに負荷をかける必要があり、Microsoft のタスク マネージャに、Load Generator がすでにクライアントの処理能力の大部分を使用していると表示された場合、Load Generator のインスタンスのいくつかを何台かのクライアント・コンピュータで起動し、1 台のサーバーをポイントすることができます。理にかなったランダムなメッセージが生成され、セッションのコンテキストで送信されます。サーバーとともに、クライアントがパフォーマンス統計を測定します。

4.4.2 Load Generator のセッションの実行

セッションを開始するには、まずスクリプトを新規に作成するか、既存のスクリプトをロードします。次に、「**Run**」メニューから「**Run**」オプションを選択するか、ツールバーの「Run」アイコンをクリックします。「**General**」タブでは、データ・サンプル間の遅延を変更できます。

サーバーの負荷の測定

「**Run**」タブには、実行中のセッションのリアルタイム情報が表示されます。表示される情報は次のとおりです。

New Requests	前のデータ･サンプルが取得された後にクローズされたリクエストの数。
New Errors	前のデータ・サンプルが記録された後に、クライアント側とサーバー側いずれかで発生したエラーの数。
New Default Responses	最後のデータ・サンプル以後、そしてアドバイザのインライン・サービスでデフォルトのレスポンスが定義されて以後、（インフォーマントの統合点リクエストと相対する）アドバイザの統合点リクエストに発生したエラーの数。
Active Scripts	この Load Generator からサーバーに同時に接続しているシミュレートされたユーザーの数。
Peak Response Time	現在のデータ・サンプル中、最も古いリクエストをクローズするまでにかかった時間。
Total Requests	クローズされたリクエストの合計数。
Total Errors	エラーの合計数。
Total Default Responses	デフォルトのレスポンスの合計数。
Total Finished Scripts	シミュレートされたユーザーの合計数。
Average Script Duration	開始から終了までの、平均的なスクリプトの実行時間（ミリ秒）。

4.4.3 パフォーマンス・グラフの表示

デフォルトでは「Requests per Second」グラフが表示されます。「**View**」→「**Graphs**」を使用すると、グラフの表示と非表示を切り替えることができます。グラフのデータを消去するには、ツールバーの「Clear Graphs」  をクリックするか、「**View**」→「**Clear Graphs**」を使用します。

スクリプトを停止して再開した場合、記録されたデータはすべて消去されます。ただし、セッションを一時停止して再開した場合は、データは消去されません。使用可能なグラフは次のとおりです。

Average Response Time	直近 40 件の平均レスポンス時間を示すヒストグラム。

Errors	直近 12,000 件のデータ・サンプル内で発生したエラーの数を示す折れ線グラフ。
Peak Response Time	直近 12,000 件の各データ・サンプル内で発生したピーク・レスポンス時間（ミリ秒）を示す折れ線グラフ。
Requests Per Second	直近 12,000 件の各データ・サンプル内で発生した 1 秒当たりの平均リクエスト数を示す折れ線グラフ。
Requests Per Second distribution	直近 40 件の 1 秒当たりのリクエスト読取り数を示すヒストグラム。

4.4.4 「General」タブについて

「General」タブには、Load Generator の構成、タイミング、および、どのインライン・サービスが指定されているかに関する変数が含まれています。「General」タブの変数は次のとおりです。

Load Generator	
Client Configuration	サーバーとの通信に Load Generator が使用するエンドポイントを記述します。
Graphs Refresh Interval in Seconds	グラフおよびカウンタの更新間の遅延を設定します。スクリプトがすでに実行中のときに設定を有効にするには、「**Apply**」をクリックします。
Details	
Inline Service	このスクリプトからリクエストが送信されるインライン・サービスの名前。
Random Number Generator Seed	スクリプトにランダムな要素がある場合、これによってランダムな動作をある程度まで再現できます。同時に複数のスクリプトを実行している場合は、ランダムな状態を繰り返すことはできません（この表で後述する「**Number of Concurrent Scripts to Run**」を参照）。
Think Time	
Fixed Global Think Time	シミュレートされたすべてのユーザーがリクエスト間で待機する秒数。
Ranged Global Think Time	シミュレートされたユーザーがリクエスト間で待機する可変時間。この思考時間は、ランダムな数字、または設定された数字範囲から順次増分される数字によって変化します。
Minimum	ゼロ以外の最小待機秒数。
Maximum	ゼロ以外の最大待機秒数（最小待機秒数より大きい数値であることが必要）。

Access Type Sequential	アクセスのたびに 1 ずつ増分される思考時間で、最大待機秒数に達すると最小待機秒数にリセットされます。
Access Type Random	アクセスのたびに、最小待機秒数と最大待機秒数の間の値を選択します（最小待機秒数と最大待機秒数も選択可）。
Scripts	
Number of Concurrent Scripts to Run	シミュレート対象となる同時ユーザーの数。
Maximum Number of Scripts to Run	このフィールドの値が正の場合、その数のセッションの完了後、Load Generator の実行は停止します。ゼロは無制限を意味します。
Logging	
Enable Logging	このオプションを選択すると、Load Generator の統計データが、定期的にファイルに書き込まれます。
Append to Existing File	ロギングが有効になっているとき、このオプションを選択すると、ログ・ファイルがすでに存在している場合はその最後に新しい統計データが追加され、存在していない場合はファイルが新規に作成されます。
Log File	タブ区切りのログ・ファイルへのフルパスで、そのファイルの内容は後述します。
Logging Interval in Seconds	ログ・ファイルに値が書き込まれてから、次の値セットが書き込まれるまでに待機する秒数。

4.4.5 変数について

変数を使用すると、負荷をシミュレートする際に、様々なソースから情報を得ることができます。セッション変数は、セッションごとに 1 回生成されます。セッション変数は、その後アクセスした際にも同じ値が使用されます。メッセージ変数は、1 つのリクエスト内では変わりません。アクセス変数は、読込みのたびに変わる場合があります。変数は「Message Actions」で使用されます。

変数の使用

メッセージで変数をパラメータの値に使用する場合、ドロップダウン・リストから選択できます。ただし、より長い文字列値の一部として使用する場合は、変数名を中カッコで囲むことができます（たとえば`{customerNum}`）。

4.4.6 タイプ

変数には次の 5 種類があります。

Constant Value	定数の値。

Integer Range	範囲から整数を選択します。 例: Minimum: 0、Maximum: 50000、Access Type: Random Minimum: 0、Maximum: 1、Access Type: Sequential
String Array	指定した配列から文字列を選択します。 例: List: [A, B, C]、Access Type: Random List: [Male, Female]、Access Type: Sequential
Weighted String Array	指定した配列から、[0,1]のような可能性とともに文字列を選択します。 例: List: [[0.3, Interested]、 [0.3, Accepted]、 [0.4, Rejected]] List: [[0.999, Interested]、[0.001, Accepted]]
Text File	ファイルからテキストの行を選択します。 例: **c:/data.txt,** Access Type:Sequential -- C:ドライブ上のファイルへの絶対参照。 **inbox/data.txt,** Access Type:Random -- スクリプト・ファイルを含むディレクトリ下の inbox ディレクトリ内のファイルへの相対参照。

4.4.7 アクションについて

現実性のある負荷をサーバーに課す複数のクライアントを簡単にシミュレートできるように、メタデータに指定された、Load Generator が実行時に解釈するパターンからメッセージを生成できます。このパターンは、メッセージ間の遅延（思考時間）が固定またはランダムになっているメッセージ順序を指定します。また、メッセージ・フィールドに値を生成するパターンもあります。メッセージ・フィールドの値には、オプションでランダムな文字を埋め込んだリテラルな文字列を使用することも、フィールドに関連付けられた値セットからランダムに選択した値を使用することもできます。セッションをサポートしているため、セッション内のメッセージ全体で特定のフィールドを一定にできます。これは、（顧客 ID、コール ID、口座番号など）セッション・キーの表現に適しています。このパターンによって、メッセージの順序付けを柔軟なものにできます。たとえば、典型的なセッションで、特定のメッセージが他のメッセージより前にくるようにしたり、ある種類について事前に決定した数のメッセージが生じるようにすることができます。

アクションの種類

アクションには、メッセージとループの 2 種類があります。

メッセージの属性は次のとおりです。

Integration Point name	メッセージの送信先となる統合点の名前。

Session Keys and values	統合点リクエストに送信される値。セッション・キーは、サーバーがルーティングに使用するので、他のメッセージ・フィールドと区別する必要があります。

ループの属性は次のとおりです。

Number of times to execute	定数値または範囲値を指定できます。範囲値は、範囲内で順次またはランダムに実行されます。

4.4.8 Load Generator の CSV ログ・ファイルの内容

この項では、Load Generator の統計を含む、カンマ区切り値（CSV）ファイルのフィールドについて説明します。

Date/Time	カウンタの現在行がファイルに追加された時刻。サーバーのログ・ファイルに格納されているメッセージとの相関関係を高めるため、ミリ秒単位の精度が用意されています。
Thread Pool Size	スクリプトの実行のために使用中または使用可能なスレッド数。これはほとんどのユーザーには無関係な、実装に関する詳細です。
New Requests	前のデータ・サンプルが取得された後にクローズされたリクエストの数。
Total Requests	クローズされたリクエストの合計数。
New Errors	前のデータ・サンプルが記録された後に、クライアント側とサーバー側いずれかで発生したエラーの数。
Total Errors	エラーの合計数。
New Default Responses	最後のデータ・サンプル以後、そしてアドバイザのインライン・サービスでデフォルトのレスポンスが定義されて以後、（インフォーマントの統合点リクエストと相対する）アドバイザの統合点リクエストに発生したエラーの数。
Total Default Responses	デフォルトのレスポンスの合計数。
Active Scripts	この Load Generator からサーバーに同時に接続しているシミュレートされたユーザーの数。
Total Scripts	シミュレートされたユーザーの合計数。
Average Response Time (ms)	現在のデータ・サンプル中、最も古いリクエストをクローズするまでにかかった平均時間。

Max Response Time (ms)	現在のデータ・サンプル中、最も古いリクエストをクローズするまでにかかった最長時間。
Average Script Duration (ms)	開始から終了までの、平均的なスクリプトの実行時間（ミリ秒）。
Snapshot Period (ms)	現在のカウンタ値が累積された期間のミリ秒数。

4.4.9 XLS ファイルの内容

この項では、`log/loadgen.csv` に書き込まれる Load Generator のカウンタをレンダリングするための、インストールの `etc` ディレクトリに含まれる Microsoft Excel のファイル `lg_perf.xls` の内容について説明します。

最上部のセル A1 には、データソースとして `lg_perf.xls` をタブ区切りのカウンタ・ファイルにリンクする方法を記述した次のコメントがあります。

> Load Generator のパフォーマンス・ログへのパスを指定するには、セル A2 にカーソルを置き、「データ」メニューから「外部データの取り込み」→「テキスト ファイルのインポートの編集」を選択し、loadgen 構成で指定したパス（通常は{$install_directory}¥log¥loadgen.csv）にナビゲートします。要求されたらデフォルトの解析設定を使用します。データは、3 分おきに自動的にリフレッシュされます。間隔などの設定を変更するには、「データ」メニューから「外部データの取り込み」→「データ範囲プロパティ」を選択します。

行 2 には、各カウンタの名前が含まれるヘッダーが表示されます。前述の CSV ファイルのすべてのヘッダーがここに表示され、ヘッダーの下に値が表示されます。

5　インライン・サービスのトラブルシューティングとデバッグ

Oracle RTD には、インライン・サービスのトラブルシューティングおよびデバッグに役立つサービスが用意されています。インライン・サービスの開発時に、インタラクティブにエラーを検索および検証したり、インライン・サービスの組込みテスト・ベッドを使用するなど、いくつかの方法でデバッグできます。

デプロイの済んだ Real-Time Decision Server は、デバッグ・モードで実行して、インライン・サービスにブレーク・ポイントを設定できます。

この章の内容は次のとおりです。

5.1　「Problems」ビューについて

「Problems」ビューでは、インライン・サービスの構築時にコンパイル・エラーや検証エラーを特定できます。コンパイル・エラーをダブルクリックすると、Java パースペクティブが表示され、エラーが強調表示されます。

検証エラーをダブルクリックすると、インライン・サービス・パースペクティブが表示され、検証エラーのある要素に対する要素エディタも表示されます。

5.2　「Test」ビューの使用

Decision Studio には、個々の統合点をテストできる「Test」ビューがあります。「Test」ビューでは、統合点をコールする運用システムをシミュレートできます。「Test」ビューには、インライン・サービス内の統合点がすべて含まれるドロップダウン・リストがあります。統合点をテストするには、セッション・キーおよびリクエスト・データに値を挿入し、「Run」アイコンをクリックして実行します。統合点についての情報は、「Results」、「Trace」、「Log」の 3 つのサブタブに表示されます。

「Results」タブには、アドバイザの統合点をコールした結果が示されます。結果を返すのはアドバイザのみです。インフォーマントのテスト、およびアドバイザとインフォーマント両方の統合点のデバッグには、`logInfo()` を使用します。

`logInfo()` 文は、デバッグ手段として、コード内の様々な箇所で使用できます。この文は、アドバイザ、インフォーマント、デシジョン、関数などを、要素内で使用する際に役立ちます。この文は、要素の論理ペインに挿入し、様々な段階でログ・データに表示する手段として使用できます。

5.2.1 logInfo( )の使用

「Log」タブには、すべての logInfo()文が表示されます。

logInfo メソッドは、第3.2項「com.sigmadynamics.support クラス SDOBase」で説明するロギング API の一部です。このクラスには、情報、デバッグ、警告、エラーの各レベルでメッセージをログに記録するためのメソッドが含まれています。一般的に、これらのロギング・メソッドは、文字列などの引数をパラメータとして受け入れます。

logInfo の使用例を 2 つ、次に示します。

```
logInfo("Installation date = " + DateUtil.toString(session().
getCustomer().getInstallationDate());

logInfo("Customer age = " + session().getCustomer().getAge() );
```

5.2.2 着信リクエスト・データのテスト

統合点のテスト時に、次のメソッドを使用して着信リクエスト・データを確認できます。

入力されたパラメータがセッション属性にマップされる場合は、そのパラメータに get メソッドがあります。

```
request.get$()
```

この$は、先頭が大文字のパラメータ名です。

この属性がマップされない場合は、パラメータのフィールド名を使用して、同じ結果をもたらすメソッドがあります。

```
String request.getStringValue(fieldName)

SDStringArray request.getStringArrayValue(fieldName)

boolean request.isArgPresent(fieldName)
```

送信レスポンス・データは常に、SDChoiceArray に格納されます。

```
    SDChoiceArray choices = null;
```

デシジョンは統合点によって実行され、選択肢が格納されます。

```
    if (session().isControlGroup()) {

      choices = RandomChoice.execute();

    } else {

      choices = SelectOffer.execute();

    }
```

選択肢を調べるには、配列から取得し、getSDOId または getSDOLabel を使用します。

```
if (choices.size() > 0) {

  Choice ch = choices.get(0);

  ch.getSDOId();

}
```

これを行うために最適な場所が、デシジョンの選択後ロジック論理です。デシジョンの実行後に選択後ロジックが実行されます。

5.3 システム・ログの使用

Oracle RTD には、ログを記録する場所が 2 箇所あります。

ログ	デフォルトの場所
Real-Time Decision Server のログ: 「Window」メニューから「Error Log エラー ログ」ビューを使用して、Eclipse から表示できます。	*RTD_HOME*¥log¥server.log
Eclipse のログ	*RTD_HOME*¥eclipse¥workspace¥.metadata¥.log

5.3.1 ロギング・レベルの設定

Eclipse のロギング・レベルを設定するには、ファイル *RTD_HOME*¥eclipse¥plugins¥com.sigmadynamics.studio_2.2¥etc¥eclipse-log.properties を変更します。

ロギング・レベルを調整するには、次の値を true または false に設定します。デフォルトの設定は次のとおりです。

- debug=false
- info=true
- warn=true
- error=true
- fatal=true
- trace=false

Real-Time Decision Server のログのロギング・オプションを変更するには、JConsole を使用します。JConsole の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

5.4 パフォーマンス監視の使用

Oracle RTD には、インライン・サービスの動作を監視する堅牢なパフォーマンス監視システムがあります。パフォーマンス監視パラメータを設定すると、JConsole を通じて、いくつかの一般的なカウンタのスナップショット・ビューを監視できます。パフォーマンス監視を有効にすると、発生順のビューを取得できます。一度有効にすると、カンマ区切りの値（CSV）ファイルが作成され、これを使用して、時間の経過を追って動作を監視できます。

警告: このファイルは無制限に大きくなるので、有効にするのはアクティブなトラブルシューティングのみにしてください。

5.4.1 パフォーマンス監視パラメータの設定

パフォーマンス監視パラメータを設定するには、「**SDManagementCluster**」→「**Members**」→「**Properties**」→「**PerformanceMonitoring** MBean」を使用します。次の表では、パフォーマンス監視を制御するプロパティを説明します。

JConsole の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。

DSPerfCounterEnabled	DS パフォーマンス・カウンタの書込みを有効にします。 ファイルは無制限に大きくなるので、漠然と有効にはしないでください。
DSPerfCounterAppend	既存のファイルがある場合、true に設定すると、パフォーマンス・データがそのファイルに追加されます。false に設定すると、サーバーを再起動したときに既存のファイルは上書きされます。
DSPerfCounterLogFile	DS パフォーマンスのカウントが定期的に追加されるタブ区切りの CSV ファイル。MS Excel が使用できる場合、インストールの `etc` ディレクトリに用意された `ds_perf.xls` を見ると便利です。データソースとして `ds_perf.xls` を `ds_perf.csv` にリンクする手順については、`ds_perf.xls` の最初の行を参照してください。
DSPerfCounterLogInterval	DS パフォーマンスのカウントの更新間隔（ミリ秒）。

5.4.2 パフォーマンス監視スナップショットの一般的な値の表示

一部のパフォーマンス・カウンタのスナップショットを表示するには、「**SDManagementCluster**」→「**Members**」→「**Decision Service** MBean」を使用します。[F5]キーを使用すると、値がリフレッシュされます。

このビューを使用するために、パフォーマンス監視を有効にする必要はありません。

5.4.3 CSV ファイルの内容

この項では、パフォーマンス・カウンタを含む CSV ファイルのフィールドについて説明します。

<table>
<tr><td>Date/Time</td><td colspan="2">カウンタの現在行がファイルに追加された時刻。サーバーのログ・ファイルに格納されているメッセージとの相関関係を高めるため、ミリ秒単位の精度が用意されています。</td></tr>
<tr><td>Max Allowable Running Requests</td><td colspan="2">同時に実行できるインライン・サービス・リクエストの最大数。

この値は、構成設定から導き出されます。値を選択する際には、オペレーティング・システムのスレッドのスケジューリング・オーバーヘッドを最小限にし、それによって、ビジーなシステムのスループットが最大限になるようにしてください。

この値は手動で設定できます。そのためには、クラスタ全体の構成プロパティ（「SDManagementCluster」→「Properties」→「Misc」→「IntegrationPointMaxConcurrentJobs」）、またはサーバー固有のプロパティ（「SDManagementCluster」→「Members」→「Properties」→「Misc」→「IntegrationPointMaxConcurrentJobs」）に、ゼロ以外の値を設定します。

プロパティをゼロに設定すると、優先値が選択されます。その場合、次の計算式に従って値が計算されます。

<code>NumCPUs * Math.ceil(1/(1-DSRequestIOFactor)) + 5</code>

この計算式で使用されている用語は次のとおりです。</td></tr>
<tr><td></td><td>NumCPUs</td><td>サーバー固有の構成プロパティ（「SDManagementCluster」→「Members」→「Properties」→「Misc」→「NumCPUs」）。

マシンに搭載されている物理 CPU の数を使用します。</td></tr>
<tr><td></td><td>Math.ceil</td><td>「次に高い整数値への切上げ」を意味します。</td></tr>
<tr><td></td><td>DSRequestIOFactor</td><td>サーバー固有の構成プロパティ（「SDManagementCluster」→「Members」→「Properties」→「Misc」→「IntegrationPointRequestIOFactor」）。

統合点リクエストが入出力処理を行う時間の小数部分、または、この仮想マシンの外部にあるシステムに対する待機時間の小数部分。デフォルト値は 0.5 です。</td></tr>
<tr><td>Peak Requests Running</td><td colspan="2">サーバーの起動以降、同時に実行されたリクエストの最大数。</td></tr>
</table>

Max Requests Running	現在のロギング間隔中に、同時に実行されたリクエストの最大数。
Requests Running	現在実行中のインライン・サービス・リクエストの数。この値は必ず、「Max Allowable Running Requests」の値以下になります。
Request Queue Capacity	このサーバーで実行されるまで、同時に待機できるリクエストの構成済最大数。これは、クラスタ全体のプロパティ（「**SDManagement-Cluster**」→「**Properties**」→「**Misc**」→「**IntegrationPointQueueSize**」）、またはサーバー固有のプロパティ（「**SDManagement-Cluster**」→「**Members**」→「**Properties**」→「**Misc**」→「**IntegrationPointQueueSize**」）の値です。 リクエストが送信されてきたとき、リクエスト・キューが満杯の場合、そのリクエストは拒否され、サーバーのログに「Server Too Busy」というエラーが記録されます。 このプロパティは、Web サーバーがサポートする同時 HTTP リクエスト（スレッド）の数より少し低い値に設定してください。リクエストはまず Web サーバーに拒否されるため、リクエスト・キューが満杯になることはありません。
Peak Queue Length	サーバーの起動以降、このサーバーで実行されるまで、同時に待機したインライン・サービス・リクエストの最大数。これは必ず、「**Request Queue Capacity**」以下になります。
Max Queue Length	現在のロギング間隔中に、このサーバーで実行されるまで、同時に待機したインライン・サービス・リクエストの最大数。これは必ず、「**Request Queue Capacity**」以下になります。
Requests Waiting (Queue Length)	現在、実行待機中のインライン・サービス・リクエストの数。
Requests When Queue Full, Total	サーバーのリクエスト・キューが満杯の間に送信されてきたリクエストの合計数。これらのリクエストはそれぞれ拒否され、「Server Too Busy」というエラーが記録されました。
Requests Queued, Total	他のリクエストの実行が完了するまで、実行待機させられたインライン・サービス・リクエストの合計数。 すべてのリクエストがキューに入っている場合、そのシステムは非常にビジーです。
Requests Seen, Total	このサーバーに対するインライン・サービス・リクエストの合計数。
Requests In System	このサーバーで処理中のインライン・サービス・リクエストの現在数。この数には、実行待機中のリクエストも実行中のリクエストも含まれています。

Timed Out Requests, Total	クラスタ全体のプロパティ（「**SDManagementCluster**」→「**Properties**」→「**Misc**」→「**IntegrationPointGuaranteedRequestTimeout**」）で指定した、保証されているサービス・レベルのタイムアウト前に実行を終了できなかったリクエストの合計数。 このカウントには、サーバーの起動以降にタイムアウトしたリクエストがすべて含まれています。 この数が増えている一方で、キューに入っているリクエストの数が増えていない場合、リクエストを扱うインライン・サービスのロジックが遅すぎて、アイドル状態のシステムでさえ、レスポンス時間の保証に対応できていないことを表しています。1 つ以上の統合点リクエストを最適化するか、保証するレスポンス時間を増やす必要があります。
Timed Out Requests	保証されているサービス・レベルのタイムアウト前に実行を終了できなかったリクエストの数。
Timed Out While Running, Total	サーバーの起動以降、開始はしたが、保証されているレスポンス時間内に終了していないことが観察されたリクエストの合計数。 クライアントが最新のレスポンスを無視するため、このサーバーの処理能力が、これらのリクエストに消費される結果、大きく浪費されています。システムがビジーのとき、まだ実行待機中のリクエストがタイムアウトして、そのリクエストに対するリソースが浪費されない場合もあります。
Timed Out While Running	現在のロギング間隔の開始以降、開始はしたが、保証されているレスポンス時間内に終了していないことが観察されたリクエストの合計数。 クライアントが最新のレスポンスを無視するため、このサーバーの処理能力が、これらのリクエストに消費される結果、大きく浪費されています。システムがビジーのとき、まだ実行待機中のリクエストがタイムアウトして、そのリクエストに対するリソースが浪費されない場合もあります。
Timed Out Requests Still Running	実行が開始され、タイムアウトになったがまだ実行中のリクエストの数。値がゼロ以外の場合、1 つ以上の統合点にプログラミング上の問題があることを示します。
Request Run Time, Average (ms)	現在のロギング間隔中に、リクエストの実行に要した平均時間（ミリ秒）。待機時間があった場合は除外されます。
Request Run Time, Max (ms)	現在のロギング間隔中に、1 つのリクエストの実行に要した最長時間（ミリ秒）。待機時間があった場合は除外されます。

Run Times < [0.1 GRT]	現在のロギング間隔中に実行が終了し、かつその所要時間が、構成済の保証されているレスポンス時間の 10%未満となったリクエストの数。 類似の書式が設定された列が 9 個あり、保証されているレスポンス時間に対し、0.10、0.25、0.50、0.75、1.00、1.25、1.50 および 2.0 倍の実行時間分布を示します。
Run Times < N and >= M	現在のロギング間隔中に実行が終了し、かつその所要時間が N ミリ秒未満で M ミリ秒以上のリクエストの数。
Run Times >= [2.0 GRT]	現在のロギング間隔中に実行が終了し、かつその所要時間が、構成済の保証されているレスポンス時間の 2 倍以上となったリクエストの数。
Request Wait Time, Average (ms)	実行またはタイムアウトされる前に、リクエスト・キュー内でリクエストが待機した平均時間（ミリ秒）。 現在のロギング間隔中に、実行が終了したリクエストと実行前にタイムアウトしたリクエストのみが含まれます。
Request Wait Time, Max (ms)	現在のロギング間隔中に、1 つのリクエストがリクエスト・キュー内で待機した最長時間（ミリ秒）。 現在のロギング間隔中に、実行が終了したリクエストと実行前にタイムアウトしたリクエストのみが含まれます。
Wait Times < [0.1 GRT]	現在のロギング間隔中に実行が終了し、かつ実行前にリクエスト・キューに入れられ、その待機時間が、構成済の保証されているレスポンス時間の 10%未満となったリクエストの数。 類似の書式が設定された列が 9 個あり、保証されているレスポンス時間に対し、0.10、0.25、0.50、0.75、1.00、1.25、1.50 および 2.0 倍の待機時間分布を示します。
Wait Times < N and >= M	現在のロギング間隔中に実行が終了し、かつ実行前にリクエスト・キューで待機した時間が N ミリ秒未満で M ミリ秒以上のリクエストの数。
Wait Times >= [2.0 GRT]	現在のロギング間隔中に実行が終了し、かつタイムアウトまでの待機時間が、構成済の保証されているレスポンス時間の 2 倍以上となったリクエストの数。
Sessions, Current	このサーバーで開いたままの Decision Server セッションの数。
Sessions, Total	このサーバーで作成された Decision Server セッションの合計数。

Stale Sessions Closed Asynchronously	リクエスト・スレッドではなく、カーネル・ジョブによって閉じられた Decision Server セッションの合計数。 これは通常、重要ではありません。ビジーなシステムでは、失効しているセッションのほとんどはリクエスト・スレッドによって閉じられ、カーネル・ジョブは、システムがビジーでなくなったときにのみ使用されます。これは、カーネル・ジョブのアクティビティを監視している人にとっては重要です（「**Kernel Jobs Running, Current**」を参照）。
Stale Sessions Closed by Requests	タイムアウトし、リクエスト・スレッドによって閉じられた Decision Server セッションの合計数。ほとんどのセッションは、特にビジーなサーバーでは、このように閉じられます。 コール元のスレッドは、インライン・サービス・リクエストの処理後、コール元に戻る前に、失効しているセッションを多くても 1 つ閉じるように要求されます。
Requests Forwarded, Total	リクエスト内のセッション・キーが現在、別のサーバーでホストされているため、このサーバーが別のサーバーに転送したインライン・サービス・リクエストの合計数。 この数がゼロ以外の場合、アプリケーション・サーバーまたは外部のロード・バランサが、セッションのアフィニティを保証する方法でリクエストを正しくルーティングしていないことを示しています。これは問題ありませんが、アプリケーション・サーバーのセッションのアフィニティ・パラメータを調整したり、外部にロード・バランシング・システムを設置することによって、パフォーマンスを改善することができます。
Remote Session Keys, Current	このサーバーが、他のサーバーでホストされている参照セッションを認識しているセッション・キーの現在の数。これらのいずれかのキーとともにリクエストが送信されてきた場合、そのリクエストは他のサーバーに転送されます。
Remote Session Keys, Total	他のサーバーでホストされているセッションのセッション･キーが、このサーバーに登録された合計回数。これは「Remote Session Keys, Current」の集計です。
Kernel Jobs Running, Current	このサーバーで現在実行中のメンテナンス・アクティビティの数。メンテナンス・アクティビティには、モデルのメンテナンス、セッションの時間調整、タイムアウト・リクエストの処理などがあります。

Kernel Jobs Running, Peak	このサーバーで同時に実行されたメンテナンス・アクティビティの最大数。この値は必ず、クラスタ全体のプロパティ（「**SDManagement-Cluster**」→「**Properties**」→「**Misc**」→「**WorkerThreadPoolSize**」）、またはサーバー固有のプロパティ（**SDManagement-Cluster**」→「**Members**」→「**Properties**」→「**Misc**」→「**WorkerThreadPoolSize**」）以下になります。
Snapshot Period (ms)	カウンタのこの行をログに記録するまでにサーバーがデータを収集した期間（ミリ秒）。

5.4.4 XLS ファイルの内容

この項では、インストールの `etc` ディレクトリに含まれる Microsoft Excel のファイル `ds_perf.xls` の内容について説明します。

最上部のセル B1 には、データソースとして `ds_perf.xls` をタブ区切りのカウンタ・ファイルにリンクする方法を記述した次のコメントがあります。

> ds_perf.csv ファイルへのパスを指定するには、セル B2 にカーソルを置き、「データ」メニューから「外部データの取り込み」→「テキスト ファイルのインポートの編集」を選択し、{$install_directory}¥log¥フォルダにナビゲートして、ds_perf.csv ファイルを選択します。要求されたらデフォルトの解析設定を使用します。データは、3 分おきに自動的にリフレッシュされます。間隔などの設定を変更するには、「データ」メニューから「外部データの取り込み」→「データ範囲プロパティ」を選択します。

行 2 には、各カウンタの名前が含まれるヘッダーが表示されます。CSV ファイルのすべてのヘッダーがここに表示され、ヘッダーの下に値が表示されます。

CSV ファイルの値の後ろに次の列が表示され、CSV の値から計算された値を示す計算式が表示されます。

Gross Throuhput (req/sec)	現在のロギング間隔中に終了したリクエストの平均速度（1 秒当たりのリクエスト数）。 計算式: `RequestsFinished / SnapshotPeriod * 1000`
Net Throughput (req/sec)	タイムアウトしたリクエストを除き、現在のロギング間隔中に終了したリクエストの平均速度。 計算式: `(RequestsFinished - Timeouts) / SnapshotPeriod * 1000`
Utilization (%)	現在のロギング間隔中のサーバーの使用率。 計算式: `(RunTimeAverage * RequestsFinished) / (MaxAllowableRunningRequests * SnapshotPeriod) * 100` 前のロギング間隔中に実行が開始されたリクエストが終了したとき、この値は 100 を少し超える場合があります。

5.5 エラー・メッセージと例外

ログに次の例外が記述される場合があります。それぞれに対して行う必要のあるアクションを示します。これらのアクションを実行しても問題を解決できない場合は、Oracle サポート・サービスに連絡してください。

例外	アクション
Error reading file *file_name*	ファイルが存在し、読取り権限が設定されていることを確認します。
Error writing file *file_name*	ファイルが存在し、書込み権限が設定されていることを確認します。
Cannot load Inline Service with id *id*. No Decision Service is present.	Decision Service がデプロイされているサーバーが実行中であることを確認します。Decision Service のデプロイ先については、JConsole を使用して確認してください。JConsole の詳細は、『Oracle Real-Time Decisions インストレーションおよび管理ガイド』を参照してください。
Failure setting up Smart Client's default responses from file, *file_name*	Smart Client のプロパティ・ファイルでファイルの場所を確認し、存在していることを確かめます。
Error connecting to server	サーバーへの接続、およびネットワークが正常に動作していることを確認します。
There were compiler errors¥: *errors*	サーバーより新しいバージョンの Decision Studio にインライン・サービスを構成した場合、サーバーでコンパイル・エラーが発生する可能性があります。この問題を修正するには、サーバーのバージョンをアップグレードします。
Internal Error.	Oracle サポート・サービスに連絡してください。
No default choice is defined for Inline Service *Inline_Service_name*, or its Integration Point *Integration_Point_name*.	デフォルトの選択肢とは、サーバーが使用できないときにコール元のアプリケーションが使用する選択肢です。デフォルトの選択肢は統合点レベルで定義します。
Could not create a backup of {0} in {1}	バックアップの場所がネットワーク上の場合は、ネットワーク接続を確認してください。バックアップの場所がローカルの場合は、バックアップに十分なディスク領域があることを確認してください。
Could not find JMX Server *server_name*	JMX Server が使用できない場合、データベース接続を確認し、サーバーが実行中であることを確めます。

例外	アクション
Unknown Decision Service message type received by HTTP endpoint¥: *endpoint_name*	HTTP クエリーの例および仕様については、『Oracle Real-Time Decisions 統合ガイド』を参照してください。
Malformed SOAP query	SOAP クエリーの例および仕様については、『Oracle Real-Time Decisions 統合ガイド』を参照してください。
Session merging not implemented, but found two keys in same request referencing different sessions	セッション・マージがまだ実装されていません。インライン・サービスでは、セッション・キーを 1 つのみ使用してください。
Input column at location *table_or_stored_procedure* and name *column_name* has a null value which is not supported in where clauses	WHERE 句の「Input」列は、null にはできません。示されているデータ ソースに移動し、列に値を入力します。
The current server *server* has a newer version of the metadata so *Inline_Service* cannot be loaded	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
The current server *server_name* supports metadata versions up to *version_number,* but the metadata of *Inline_Server* is at version *version_number*	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
Generation of Inline Service "*Inline_Service_name*" failed	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
Unable to read a study definition created by a newer version of the software.	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
Unable to read a prediction model created by a newer version of the software.	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。

例外	アクション
Encountered a database record created by a newer version of the software.	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
Unable to read a learning model created by a newer version of the software.	このエラーは、使用している Decision Studio のバージョンが Real-Time Decision Server のバージョンより新しい場合に発生する可能性があります。このエラーを修正するには、サーバーをアップグレードします。
No result set found while getting result set of procedure "*stored_procedure_name*".	このエラーは、データソースに結果セットを定義したのに、ストアド・プロシージャに結果セットが存在しない場合に発生する可能性があります。データベースでストアド・プロシージャの定義を確認してください。
Generic Exception caught while setting blob for procedure "*stored_procedure_name*".	このエラーには様々な理由が考えられます。データベース接続を確認してください。データベース・サーバーが実行中であることを確認してください。
Exception during output of batched statements¥: database *insert/update/select/delete* operation for *table_or_stored_procedure* took *duration*. Batch size is *number_of_results*.	このエラーには様々な理由が考えられます。データベース接続を確認してください。データベース・サーバーが実行中であることを確認してください。
The stored procedure "*stored_procedure_name*" was not found in database.	このエラーには様々な理由が考えられます。データベース接続を確認してください。データベース・サーバーが実行中であることを確認してください。最後に、データソースで指定したストアド・プロシージャの名前が正しく、データベースに存在していることを確認してください。
Failed to find column "*column_name*" in table "*table_name*".	このエラーには様々な理由が考えられます。データベース接続を確認してください。データベース・サーバーが実行中であることを確認してください。最後に、データソースで指定したテーブルの名前が正しく、データベースに存在していることを確認してください。
Error setting up Smart Client properties.	このエラーは、Smart Client のプロパティ・ファイルの構成が不適切な場合に発生する可能性があります。Smart Client の使用方法については、『Oracle Real-Time Decisions 統合ガイド』を参照してください。

例外	アクション
Failed to load Inline Service¥: *Inline_Service_name*	このエラーは、Real-Time Decision Server の起動時に発生する可能性があります。起動しなかったインライン・サービスを再デプロイしてください。再度このエラーが発生した場合は、Oracle サポート・サービスに連絡してください。

開発時には、次のエラーが発生する場合があります。

エラー	説明
Internal error in code generator. See error log for details.	Decision Studio の「Problems」ペインでエラーを確認してください。見当たらない場合は、Oracle サポート・サービスに連絡してください。
Cannot get *Inline_Service* from the database. Will mark it invalid	データベース・サーバーが実行中であること、適切なドライバがあること、およびデータベース・サーバーに接続していることを確認してください。
Error loading Inline Service *Inline_Service_Name*. Will mark it as invalid in the database.	サーバーから、インライン・サービスのロードに関するエラーが示された場合、Application Initialization ロジックおよび Session Initialization ロジックにあるロジックが正しいことを確認してください。
Response for an asynchronous request to Advisor "*Advisor_name*" will not be sent	アドバイザからのレスポンスが必要な場合は、`invokeAsync()`ではなく、`invoke()`メソッドを使用する必要があります。
Invoke failed	Real-Time Decision Server に接続していることを確認してください。プロパティ・ファイルの構成が正しいことを確認してください。Invoke の使用方法の詳細は、『Oracle Real-Time Decisions 統合ガイド』を参照してください。
Internal error	Oracle サポート・サービスに連絡してください。
Internal error while generating code for *Inline_Service*.	Oracle サポート・サービスに連絡してください。
Error in Application Session Cleanup.	このエラーの原因としては、アプリケーション要素およびセッション要素のロジックが不適切であることが考えられます。アプリケーション・クリーンアップ・ロジックおよびセッション・クリーンアップ・ロジックにあるロジックが正しいかどうか確認してください。

エラー	説明
Failed to load study "*study_name*"	このエラーの原因としては、データベース接続の問題が考えられます。データベース・サーバーが実行中であること、適切なドライバがあること、およびデータベース・サーバーに接続していることを確認してください。
Failed to save study "*study_name*"	このエラーの原因としては、データベース接続の問題が考えられます。データベース・サーバーが実行中であること、適切なドライバがあること、およびデータベース・サーバーに接続していることを確認してください。
Failed to load prediction model "*model_name*"	このエラーの原因としては、データベース接続の問題が考えられます。データベース・サーバーが実行中であること、適切なドライバがあること、およびデータベース・サーバーに接続していることを確認してください。
Error delivering response message: *error_details*	統合点で `invoke()` がタイムアウトした可能性があります。プロパティ・ファイルでタイムアウトの設定を確認してください。`invoke()` の使用方法の詳細は、『Oracle Real-Time Decisions 統合ガイド』を参照してください。
Product sdstudio could not be found.	このエラー・メッセージは単に情報を伝えるもので、無視してかまいません。