Oracle® Real-Time Decisions

統合ガイド

ORACLE

リリース 2.2　　2007 年 4 月

Oracle Real-Time Decisions 統合ガイド, リリース 2.2

部品番号: E05256-01

原本名: Oracle Real-Time Decisions Integration with Oracle RTD, 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。

統合ガイド

はじめに

Oracle Real-Time Decisions（Oracle RTD）プラットフォームにより、ビジネス・プロセスの行動を分析し、提案をリアルタイムに作成するエンタープライズ・ソフトウェア・ソリューションを開発できます。これによって、発生した時点で即座に問題や機会を特定して対応できます。

このドキュメントについて

このドキュメントでは、Oracle RTD との統合について説明します。また、RTD スマート・クライアント、Web サービス、Real-Time Decision Server による直接メッセージ交換などの統合方法についても説明します。

対象読者

このドキュメントは、Java ベースの API、.NET コンポーネントまたは Web サービスを使用してエンタープライズ・アプリケーションと Oracle RTD インライン・サービスとを統合する開発担当者を対象にしています。統合開発担当者は、Java、.NET または Web サービスに関してすべての分野にわたる知識は必要ありませんが、アプリケーションとの統合に関して、統合で使用するプロトコルとその使用方法に精通している必要があります。また、統合開発担当者は、Decision Studio とインライン・サービスの使用方法にも精通している必要があります。

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「概要」では、Oracle RTD との統合方法の概要について説明します。

第2章「Java スマート・クライアントの使用」では、Java スマート・クライアントを使用してインライン・サービスと統合する方法について説明します。

第3章「Java スマート・クライアントの JSP タグの使用」では、JSP タグを使用して Java スマート・クライアントを統合する方法について説明します。

第4章「.NET スマート・クライアントの使用」では、.NET スマート・クライアントを使用してインライン・サービスと統合する方法について説明します。

第5章「ゼロ・クライアントによる統合」では、RTD のデシジョン・サービス用 SOAP インタフェースについて説明します。

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。

規則	説明
	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 概要

Oracle Real-Time Decisions には、企業の業務系システムとの統合方法として、次に示す強力で簡単に使用できる方法が用意されています。

- スマート・クライアント: Java 環境および.NET 環境では、スマート・クライアントのコンポーネントにより、Real-Time Decision Server における統合点との通信を管理します。
- ゼロ・クライアント: ゼロ・クライアントの手法として Web サービスを介して 統合点にアクセスします。

このドキュメントでは、Oracle RTD 上で実行するデプロイ済インライン・サービスとの統合に、これらの方法を使用する方法について概略を説明します。

Decision Studio を使用してインライン・サービスをデプロイする方法の詳細は、『Oracle Real-Time Decisions スタート・ガイド』を参照してください。統合 API の詳細は、Decision Studio のオンライン・ヘルプを参照してください。

1.1 最適な統合方法の選択

Oracle Real-Time Decisions には、複数の統合方法が用意されています。ご使用の環境に最適な方法を選択するには、使用するプラットフォームやパフォーマンス要件だけでなく、RTD スマート・クライアントが用意している統合方法固有の追加機能についても考慮する必要があります。

1.1.1 Java スマート・クライアントについて

Java 用の RTD スマート・クライアントはコンポーネントであり、これによって業務系システムとしてデプロイ済インライン・サービスとの統合が容易になり、統合に管理機能を実装できます。Java 環境を使用する場合は、Java スマート・クライアントが最適な統合方法です。Java スマート・クライアントにはその他の統合方法にはない重要な機能が 2 つ用意されています。1 つは、セッション・キーのマッピング（外部のロード・バランサによる HTTP セッション・アフィニティ管理が容易になる機能）で、もう 1 つはデフォルトのレスポンス処理です。

Java スマート・クライアント・インタフェースのファクトリ・メソッドには、クラスタ化されたサーバーとの通信の確立に必要な最低限の情報を表すパラメータがあります。コンポーネントのフル構成は、接続の確立後に、サーバーからダウンロードされます。この方法では、少数のパラメータ・セットのみがクライアント・アプリケーションで管理され、コンポーネントの大半の構成はサーバーの管理コンソールによって一元管理されます。

サーバーからクライアントに返される構成情報は、同じ Java 仮想マシン上に作成されているすべてのスマート・クライアント・インスタンスによって共有されます。クライアント・サイドには、この共有構成を管理するクラス（クライアント・サイド・ディスパッチャと呼ばれる）があります。また、セッションのアフィニティ情報もこのクラスによって管理され、リクエストのセッション・キーに基づいて適切なサーバーへリクエストをディスパッチするために使用されます。

Java スマート・クライアントはスレッドセーフですが、最適なパフォーマンスを実現するには、各スレッドに個別に Java スマート・クライアントを 1 つ作成します。Java スマート・クライアントの各インスタンスは情報と接続を共有するため、複数のインスタンスを持つことによるデメリットは実質的にありません。

Java スマート・クライアントの作成に使用できるファクトリ・メソッドは複数あります。大半のメソッドは、直接的または間接的に、ファイル・システムまたは Web サーバー上のプロパティ・ファイルを参照します。このプロパティ・ファイルには、単一クラスタ内の 1 台以上のサーバーに接続するためのアドレスに加えて、サーバーへの接続を構成するその他のプロパティがあります。また、ファクトリ・メソッドに、HTTP URL とポートを直接指定することも、デフォルト・アドレスを使用することも可能です。

クライアントのコンストラクタが特定のサーバーと通信して詳細な構成情報を受信すると、クライアント構成キャッシュと呼ばれるローカル・ファイルにその詳細構成情報が保存されます。これによって、サーバーが使用できない状態になったときにクライアントを再起動した場合でも、詳細構成情報にアクセスできます。クライアント構成キャッシュには、すべてのインライン・サービスのすべての統合点に対する、クライアントの一連のデフォルト・レスポンスなどに関する情報が格納されます。サーバーで構成が変更された場合、クライアント構成キャッシュは、クライアントによって自動的に更新されます。

サーバーからクライアントにダウンロードされる構成情報には、クライアントにおいてサーバーとの通信が切断された場合やサーバーが統合点リクエストに適時にレスポンスできない場合に使用されるデフォルト・レスポンスのセットがあります。これによって、個々のトランザクションの可用性とは関係なく、Real-Time Decision Server とクライアント・アプリケーションとの間におけるサービス・レベル合意（SLA）が維持されます。

これらのデフォルト・レスポンスは個々の統合点の粒度で構成され、各統合点は独自に特化したデフォルト・レスポンスに依存します。特定のデフォルト・レスポンスがサーバー上で再構成された場合は、その変更が通常の統合点レスポンスとともに、自動的にクライアントの帯域外データに伝播されます。

Java スマート・クライアントでは、Real-Time Decision Server の Web コンテナから返されたすべての HTTP Cookie が自動的に追跡管理されます。次回、同じインライン・サービス・キーをリクエストで使用すると、対応する Cookie が HTTP リクエストに挿入されます。これによって、外部のロード・バランサがそのインライン・サービス・キーを処理しているサーバー・インスタンスにリクエストをルーティングできるようになります。

他の統合方法を使用してクラスタリングを実現する場合は、アプリケーション自体がインライン・サービス・キーを追跡管理する必要があります。

1.1.2 .NET スマート・クライアント

.NET 環境では、.NET スマート・クライアントのコンポーネントを使用できます。このコンポーネントには、Java スマート・クライアントと同じインタフェースをコールする機能が用意されています。ただし、セッション・アフィニティの維持とデフォルト値関連の追加機能はありません。

1.1.3 Web サービス

クライアントは、Web サービスを介して Real-Time Decision Server にアクセスできます。この統合方法の利点は、クライアント上にコードが不要なことです。Web サービスの運用は WSDL ファイルに定義され、Web サービスの定義はスキーマ・ファイルで指定します。

1.2 CrossSell インライン・サービスについて

サンプルのインライン・サービスが Decision Studio に付属しています。その中の 1 つが抱合せ販売用のサンプルです。

CrossSell インライン・サービスでは、クレジット・カード会社のコンタクト・センターを想定した単純な実装がシミュレートされます。センターに電話すると、電話した顧客と連絡先チャネルに関する情報が取得されます。

この顧客に関する情報に基づいて、顧客に提案する抱合せ販売のオファーが選択されます。オファーの成功や失敗が追跡されてサーバーに追跡情報が返されます。これによって、基礎となる意思決定モデルに対して精度を高めるフィードバックが返され、抱合せ販売のオファーが改善されます。

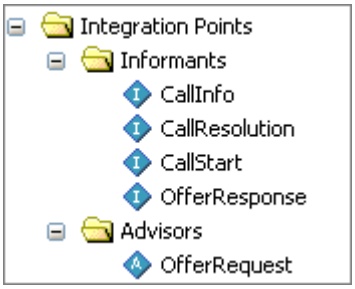
このドキュメントでは、CrossSell インライン・サービスを使用して、各種統合方法について説明します。

複数の統合点が CrossSell サンプルに用意されています。後述する記載に従って、これらの統合点について理解を深めてください。

インフォーマントでは、適切なパラメータが渡されると、サーバー上で実行されます。そして、アドバイザが実行してデータを返します。統合点へのコールに正しいパラメータを指定するには、最初にオブジェクト ID を特定する必要があります。

1.2.1 Decision Studio によるオブジェクト ID の特定

1. *RTD_HOME*¥eclipse¥eclipse.exe を実行し、Decision Studio を起動します。
2. 「**File**」→「**Import**」を選択し、CrossSell インライン・サービスを起動します。「**Import**」が表示されます。
3. 「**Existing Project into Workspace**」を選択し、「**Next**」をクリックします。*RTD_HOME*¥examples¥CrossSell にある CrossSell プロジェクトを参照します。「**OK**」を選択してから「**Finish**」をクリックし、CrossSell プロジェクトを起動します。
4. 「Inline Service Explorer」を使用して、「**Integration Points**」を開きます。その下に、「**Informants**」と「**Advisors**」が表示されます。それらの両方を開き、統合点を表示します。オブジェクト ID 切替えアイコン  を使用して、オブジェクト ID を「Inline Service Explorer」に表示します。この切替えアイコンを選択すると、オブジェクト ID が「Inline Service Explorer」に表示されます。選択を解除すると表示ラベルが表示されます。



統合点のオブジェクト ID は、ラベルと同じ場合と異なる場合があります。オブジェクト ID は、統合点をメソッド・コールにおいて参照するために使用されます。

1.2.2 アドバイザのレスポンスの決定

レスポンスを配信する統合点を、アドバイザと呼びます。Decision Studio では、アドバイザの「**Response**」タブを使用して、アドバイザによって暗黙的に起動されるパラメータ化 Decision オブジェクトを指定することでレスポンスが決まります。Decision オブジェクトの役割は、それに割り当てられている選択肢グループから最適な選択肢を選択することです。配信される選択肢属性は、選択肢グループの定義に設定されている構成によって決まります。

この例では、OfferRequest 統合点がアドバイザになります。OfferRequest が起動されると、単一の抱合せ販売オファーが返されます。

1. Decision Studio で、「**OfferRequest**」統合点を選択し、エディタを表示します。
2. 「**Response**」タブにある「**Decision**」で、OfferRequest がレスポンスの返信に使用するデシジョンを調べます。「**OfferDecision**」であることを確認します。
3. 「**Decisions**」にある「**OfferDecision**」をダブルクリックし、その詳細ペインを表示します。
4. 「**Selection Criteria**」タブにある「**Number of Choices to Select**」で、OfferRequest によって配信されるレスポンスの数を確認します。
5. 「**Selection Criteria**」タブにある「**Choice Group**」で、OfferRequest によって使用される選択肢グループを調べます。「**Offers**」であることを確認します。
6. 「**Choices**」にある「**Offers**」をダブルクリックし、この選択肢グループに関連付けられている選択肢属性を確認します。アドバイザがコールされたときに、これらの属性が返されます。

ヒント: Decision Studio では、「**Test**」ビューを使用して、アドバイザをコールし、返される内容を確認します。このビューには、返されるオファーと、それに付属する属性が表示されます。「**Test**」は、「**Problems**」以外のタブから使用できます。「**Execute Request**」  ボタンを使用して、リクエストをサーバーに送信します。

1.2.3 サーバーにレスポンスする方法の理解

選択肢の成功や失敗が追跡され、モデルにおいてその情報に基づいて自己学習が行われるにつれて、インライン・サービスはより強力になります。サーバーの自己学習に必要なフィードバックを確認するには、選択肢イベント・モデルを調べる必要があります。

1. Decision Studio で、「**Offer Acceptance**」選択肢イベント・モデルをダブルクリックします。エディタが右側に表示されます。

2 「**Choice**」タブにある「**Positive Outcome Events**」で、サーバーの自己学習に影響を及ぼすイベントを確認します。それらは次のとおりです。

- Interested
- Purchased

これらの結果がインライン・サービスからサーバーに報告されることで、適切なフィードバックがモデルに返されます。

3 **OfferResponse** 統合点には、この情報報告の役割があります。

1.2.4 セッション・キーおよび引数の特定

統合点を起動するには、統合点に対応するセッション・キーおよび引数の値を指定する必要があります。リクエストでは、Decision Studio によって定義されているオブジェクト ID を統合点のセッション・キーおよび引数に対して使用する必要があります。キー名は、Decision Studio に定義されている統合点用セッション・キー名のいずれかと一致する必要があります。

1 「**CallStart**」統合点を選択します。対応する統合点エディタの「**Request**」タブにある「**Session Keys**」リストに、セッション・キーへのパス（`session` で始まる）が表示されます。このパス名の最後はセッション・キーのオブジェクト ID です。

> 注意: セッション・キーがオブジェクト形式で表示されない場合は、**オブジェクト ID 切替えアイコン**を使用して表示設定を変更します。最後のオブジェクト ID のみセッション・キーで必要です。たとえば、この例では最後の文字列の `customerId` が使用されます。

2 統合点の引数を特定するには、詳細ペインを使用して、「**Request**」タブの「**Incoming Attribute**」列を表示します。「**CallStart**」の入力引数は **channel** です。

2　Java スマート・クライアントの使用

この章では、統合に Java スマート・クライアントを使用する方法について説明します。サンプルが Oracle RTD インストールに付属しています。

2.1　作業前の準備

Java スマート・クライアントのサンプルで作業を行う前に、次の作業を実行する必要があります。

1. Java Development Kit（JDK）をインストールし、*JAVA_HOME* 環境変数にそのインストール場所を設定します。JDK の入手方法の詳細は、Sun 社の Web サイト http://java.sun.com/products/ を参照してください。

2. Oracle RTD ファイルをインストールし、Oracle RTD をアプリケーション・サーバーにデプロイします。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。

3. Java スマート・クライアントのサンプルは、サンプルの CrossSell インライン・サービスと連携して動作します。そのため、最初に Oracle RTD Database に CrossSell のサンプル・データを移入してから、Decision Studio を使用して CrossSell インライン・サービスをデプロイする必要があります。Oracle RTD Database に CrossSell のサンプル・データを移入する方法の詳細は、『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。インライン・サービスのデプロイ方法の詳細は、『Oracle Real-Time Decisions Decision Studio リファレンス・ガイド』を参照してください。

4. Real-Time Decision Server を起動します。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。

2.2　Java スマート・クライアントによるインライン・サービスとの統合

一般的に、Java スマート・クライアントを使用した統合は、次の手順に従って実行します。

1. プロパティ・ファイルを準備します。

2. インライン・サービスへの接続を作成します。

3. 統合点による出力の決定に必要な様々な情報（接続先統合点やセッション識別用パラメータなど）を識別するリクエストを作成します。

4. リクエストを起動します。

5. レスポンス情報をアドバイザから収集して分析します。

6. 接続を切断します。

2.2.1 Java スマート・クライアント API のリファレンス

Java スマート・クライアント API の詳細は、Decision Studio のオンライン・ヘルプを参照してください。

2.2.2 Java スマート・クライアントのサンプルの準備

この項では、簡単なコマンドライン・アプリケーションに統合された CrossSell インライン・サービスを使用して、統合に Java スマート・クライアントを使用する方法について説明します。

次の手順に従って、スマート・クライアントのサンプルを準備します。

1. *RTD_HOME*¥client¥Client Examples¥Java Client Example¥lib¥sdbootstrap.properties ファイルのある場所に移動してファイルを開き、編集します。次に示すように、client=true 以外のすべてのプロパティをコメント化します。

```
client=true

#StudioStaticFilesLocation=shared_ui/studio

#WebServerLocation=http://localhost:8080

#WorkbenchServlet=/ui/workbench
```

   ファイルを保存して閉じます。

2. Decision Studio を起動し、「**File**」→「**Import**」→「**Existing Projects into Workspace**」を選択し、「**Next**」をクリックします。
3. 「**Select root directory**」で、*RTD_HOME*¥client¥Client Examples¥Java Client Example を参照し、「**OK**」をクリックします。「**Finish**」をクリックします。
4. メニュー・バーから、「**Window**」→「**Open Perspective**」→「**Java**」を選択します。コンソール・ビューが表示されない場合は、「**Window**」→「**Show View**」→「**Console**」を選択します。
5. メニュー・バーから、「**Run**」→「**Run**」を選択します。
6. 「Create, manage, and run configurations」画面で、「**Java Application**」を選択し、「**New**」をクリックします。
7. 「**Project**」フィールドの横にある「**Browse**」をクリックし、「**JavaSmartClientExample**」を選択して「**OK**」をクリックします。
8. 「**Main class**」フィールドの横にある「**Search**」をクリックし、「**Example**」を選択して「**OK**」をクリックします。
9. 「**Apply**」→「**Run**」をクリックします。コンソール・ビューに、次のテキストが表示されます。

10 カーソルをコロンの後に移動し、顧客 ID（たとえば、5）を入力し、[Enter]を押します。次のようなレスポンスが表示されます。

11 カーソルを最後のコロンの後に移動し、「1」を入力してオファーを選択します。サーバーによって、最後のメッセージがレスポンスされます。

12 プロセスが繰り返されます。プログラムを終了する場合は、1000 よりも大きな顧客 ID を入力します。

このサンプルのソース・コードは、*RTD_HOME*¥client¥Client Examples¥Java Client Example¥src¥com¥sigmadynamics¥client¥example¥Example.java ファイルにあります。次の項で、このサンプルについて説明します。

2.2.3 Java スマート・クライアントのプロパティについて

クライアント・アプリケーションで Java スマート・クライアントが作成されると、コンポーネントのエンドポイント構成となる Java スマート・クライアント・ファクトリに、プロパティのセットが渡されます。このファイルには、クライアントがサーバー・エンドポイントに接続するために必要な情報が記載されています。デフォルトの構成値を使用するファクトリ・メソッドが別に用意されていますが、明示的にプロパティを指定することをお薦めします。デフォルトのプロパティ・ファイルについては、次の項で説明します。

ファクトリ・メソッドでは、このプロパティを使用してサーバーに接続します。ファクトリがサーバーに接続すると、詳細な構成情報がクライアントにダウンロードされます。これには、サーバーが使用できない状態になったときに、クライアントで実行が必要な一連のデフォルト・レスポンスなどが含まれます。詳細なクライアント構成は、Java スマート・クライアント構成キャッシュと呼ばれるローカル・ファイルに保存され、サーバーの構成が変更されると自動的に更新されます。

2.2.4 プロパティ・ファイルの作成

1 *RTD_HOME*¥client¥Client Examples¥Java Client Example¥lib¥sdclient.properties ファイルのある場所に移動してファイルを開き、編集します。ファイルの内容は次のとおりです。

```
UseEndpointsInOrder = HTTP1

appsCacheDirectory = ${rootDir}/etc

timeout = 0

HTTP1.type = http

HTTP1.url = http://localhost:8080/
```

2 使用するサーバー構成に一致するように、ファイルの内容を変更します。次の表に、このファイルの各要素に関する説明を示します。特に、有効なキャッシュ・ディレクトリが存在することと、エンドポイントの URL がローカルの Real-Time Decision Server の URL とポートになっていることを確認してください。デフォルトでは、この URL は `http://localhost:8080` です。

要素	説明
`UseEndpointsInOrder`	エンドポイント名のカンマ区切りのリスト。スマート・クライアントの初期化時に、このリストに指定された順番でサーバー・クラスタへの初期接続の確立が試行されます。初期化後は、サーバーによってエンドポイント・リストが更新されるため、このリストは無効になります。 このリストにあるエンドポイント名はこのプロパティ・ファイル内の定義にのみ関連し、この名前が他で使用されることはありません。
`appsCacheDirectory`	クライアント・コンポーネントがサーバーから取得した構成情報を保存する書込み可能ディレクトリを指定するファイル URL。このキャッシュによって、アプリケーションによるクライアント・コンポーネントの初期化時に、Real-Time Decision Server が使用できない状態になった場合に使用する情報が用意されます。`sdclient.properties` では、キャッシュ・ディレクトリとして既存のディレクトリを指定する必要があります。存在しない場合は、Java 仮想マシンの一時ディレクトリが使用されます。
`timeout`	クライアント・コンポーネントの初期化時に、サーバーへの初期接続の試行に適用されるタイムアウト時間（ミリ秒単位）。サーバーに接続すると、サーバーのタイムアウト時間である `EntryPointRequestTimeout`（JMX MBean プロパティによって構成）が使用されます。
`<endpoint_name>.type`	指定されたエンドポイントのタイプ。現時点では、`HTTP` のみがサポートされています。
`<endpointName>.url`	サーバーの HTTP エンドポイントの HTTP ホストとポートを指定する URL。デフォルトのエンドポイントは、http://localhost:8080 です。

2.2.5 Java スマート・クライアントの作成

1 サンプル・アプリケーションのソース・ファイル（*RTD_HOME*¥client¥Client Examples¥Java Client Example¥src¥com¥sigmadynamics¥client¥example¥Example.java）を開きます。

ヒント: このサンプルのソース・コードを、各自の Java スマート・クライアント実装のテンプレートとして使用できます。

2 Oracle RTD 統合をサポートするために、次のインポートが使用されます。

```
import com.sigmadynamics.client.IntegrationPointRequestInterface;
import
com.sigmadynamics.client.IntegrationPointResponseInterface;
import com.sigmadynamics.client.ResponseItemInterface;
import com.sigmadynamics.client.SDClientException;
import com.sigmadynamics.client.SDClientFactory;
import com.sigmadynamics.client.SDClientInterface;
```

3 サンプル・アプリケーションのメイン・メソッドには、サンプル・アプリケーションに指定された引数に基づいて、SDClientFactory を使用して SDClientInterface 実装を作成するコードが記述されています。

これらの引数は getClient に渡され、そこで適切なファクトリ・メソッドが識別されます。

```
SDClientInterface client = getClient(args);
```

Java スマート・クライアントの作成に使用されるファクトリ・メソッドは複数あります。getClient のコードを調べることで、様々な方法を確認します。

```
private static SDClientInterface getClient(String[] args ){
    try{
      if ( args.length == 0 )
        return getClientWithDefaultPropertiesFile();
```

create(java.lang.String) を使用してデフォルトのプロパティ・ファイルに基づいて Java スマート・クライアントが作成されます。デフォルトのプロパティ・ファイルの詳細は、前述の項を参照してください。

```
      if ( "-h".equals(args[0])){
        if ( args.length < 2 )
          return getClientWithDefaultHttpAddress();
```

デフォルトのHTTPアドレスのhttp://localhost:8080を使用して Java スマート・クライアントが作成されます。これは、Real-Time Decision Server のデフォルトのインストール URL およびポートです。createHttp(java.lang.String, int, boolean)が使用されます。

```
      return getClientWithHttpAddress( args[1]);
    }
```

指定された HTTP アドレスを使用して Java スマート・クライアントが作成されます。これは、デフォルト・アドレスにない Real-Time Decision Server のアドレスおよびポートです。createHttp(String)が使用されます。

```
    if ( "-u".equals(args[0])){
      if ( args.length < 2 )
      {
        System.out.println("Missing properties file URL
        argument" );
        System.exit(-1);
      }
      return getClientWithPropertiesFileURL( args[1] );
    }
```

指定されたアドレスにあるプロパティ・ファイルに記述されている情報を使用して Java スマート・クライアントが作成されます。createFromProperties が使用されます。

```
    if ( "-f".equals(args[0])){
      if ( args.length < 2 )
      {
        System.out.println("Missing properties filename
argument" );
        System.exit(-1);
      }
      return getClientWithPropertiesFileName( args[1] );
    }
```

プロパティ・ファイルに記述されている情報を使用して Java スマート・クライアントが作成されます。createFromPropertiesURL が使用されます。

```
    System.out.println("Unrecognized argument");
  }catch (SDClientException e ){
    e.printStackTrace();
```

```
    }
    System.exit(-1);
    return null;
  }
```

これらのメソッドの概要は、Decision Studio のオンライン・ヘルプの Java スマート・クライアント API に関する項を参照してください。

2.2.6 リクエストの作成

1 次に、クライアント・アプリケーションでは、Real-Time Decision Server に送信するリクエストが作成されます。

リクエスト・オブジェクトの作成には、`SDClientInterface` の **`createRequest`**`(String appName, String integrationPointName)` が使用されます。

> `appName` パラメータは、Decision Studio にデプロイされたサーバー常駐アプリケーションの名前です。
>
> `integrationPointName` パラメータは、アプリケーションのインフォーマントまたはアドバイザの名前で、リクエストの受信先です。
>
> これらの値を特定する方法の詳細は、第1.2.1項「Decision Studio によるオブジェクト ID の特定」を参照してください。

このサンプルでは、リクエストは次のコードで作成されます。

```
IntegrationPointRequestInterface request  =
client.createRequest(INLINE_SERVICE_NAME, "CallStart");
```

2 リクエスト・オブジェクトには、単一のセッション・キーを設定するメソッドがあります。それぞれのキーごとに、このメソッドをコールします。

```
void setSessionKey(String keyName, String keyValue);
```

> たとえば、統合点のセッション・キーが Decision Studio に **session.customer.customerId** として指定されている場合は、**customerId** をキー名として `setSessionKey` に渡します。
>
> これらの値を特定する方法の詳細は、第1.2.4項「セッション・キーおよび引数の特定」を参照してください。

サンプル・アプリケーションでは、リクエストのセッション・キーは次のコードによって移入されます。

```
request.setSessionKey( SESSION_KEY, sCustID );
```

前述の SESSION_KEY は、次の文により設定されます。

```
static final String SESSION_KEY = "customerId";
```

sCustID は、コマンドライン入力から取得されます。

3 リクエスト・オブジェクトには、単一の引数を設定するメソッドが 2 つあります。それぞれの引数ごとに、適切なメソッドをコールします。最初のメソッドでは、単一の文字列値を持つ引数を受け取ります。2 番目のメソッドでは、文字列値の配列を持つ引数を受け取ります。

```
void setArg(String argName, String argValue);

void setArg(String argName, String[] argValue);
```

> 引数名は、統合点用として Decision Studio に指定されている入力名の 1 つと一致する必要があります。
>
> これらの値を特定する方法の詳細は、第1.2.4項「セッション・キーおよび引数の特定」を参照してください。この引数の値は、アプリケーションの設計に基づいて決めます。

サンプル・アプリケーションでは、リクエストは次のコードによって移入されます。

```
request.setArg( "channel", "Call");
```

2.2.7 リクエストの起動

1 リクエストが移入されると、クライアント･アプリケーションでは、SDClientInterface の invoke メソッドをコールしてリクエストをサーバーに送信し、サーバーによって計算処理された選択肢の配列を表す IntegrationPointResponseInterface を受信します。

```
IntegrationPointResponseInterface
invoke(IntegrationPointRequestInterface request);
```

サンプル･アプリケーションでは、次のようにコールされます。

```
client.invoke(request);
```

> **注意**: レスポンスが不要なリクエストやメッセージ配信順序が重要でないリクエストをクライアント･アプリケーションで送信する場合、invoke メソッドではなく invokeAsync メソッドを使用できます。
>
> invokeAsync をコールして送信されるリクエストは、それ以降 invokeAsync や invoke をコールして送信されるリクエストよりも前に、サーバーに到着するとはかぎりません。メッセージ配信順序が重要な場合、レスポンスが不要でも、invokeAsync メソッドではなく invoke メソッドを使用してください。

2 CallStart 統合点へのリクエストが起動されると、新しいリクエストが準備され、CallInfo に対して起動されます。

```
// Supply some additional information about the telephone call.

// Apparently the CrossSell service expects very little here --

// just the channel again, which it already knows. Hence this message
// could be left out with no consequences.
request = client.createRequest(INLINE_SERVICE_NAME, "CallInfo");
request.setSessionKey( SESSION_KEY, sCustID );
request.setArg( "channel", "Call");
client.invoke(request);
```

2.2.8 レスポンスの確認

アドバイザが起動されると、特定数のレスポンス・アイテム（選択肢とも呼ばれる）が返されます。使用するアプリケーションには、返される数だけのレスポンス・アイテムを処理するコードが必要です。詳細は、第1.2.2項「アドバイザのレスポンスの決定」を参照してください。

クライアント・アプリケーションでは、invoke メソッドによって返された IntegrationPointResponseInterface を介して、それらの選択肢にアクセスできます。IntegrationPointResponseInterface により、レスポンス・アイテムのオブジェクト配列を表す ResponseItemInterface にアクセスできます。ResponseItemInterface の各レスポンス・アイテムは、アドバイザの意思決定によって選択された選択肢オブジェクトに対応します。

com.sigmadynamics.client パッケージは、各選択肢を値文字列のコレクションとして抽出し、名前文字列によって区別します。

サンプル・アプリケーションでは、次のようにしてレスポンスを確認します。

1 リクエストをアドバイザの統合点に対して起動するとき、レスポンスの受信を準備します。

```
// Based on what the server knows about this customer, ask for some
// product recommendations.
request = client.createRequest(INLINE_SERVICE_NAME,
"OfferRequest");
request.setSessionKey( SESSION_KEY, sCustID );
IntegrationPointResponseInterface response =
client.invoke(request);
```

2 返されるレスポンスの数が判明している場合は、処理を正確に記述できます。レスポンスは配列から読み取られ、顧客に表示されます。

```
if ( response.size() > 0 ){
// Since I know that CrossSell's OfferDecision returns only
// one Choice, I could get that choice from the response with
// response.get(0); Instead, I'll pretend that
```

```
// multiple offers could be returned instead of just one.
        System.out.println();
        System.out.println("Here are the deals we've got for
        you:");
        ResponseItemInterface[] items =
        response.getResponseItems();
        for ( int i = 0; i < items.length; i++ ){
          System.out.println(" " + (i+1) + ": " + items[i].getId());
          String message = items[i].getValue("message");
          if ( message != null )
            System.out.println("   " + message );
        }
        System.out.println();
        System.out.println("Enter the line number of the offer
        that catches your interest, or zero if none do: " );
```

2.2.9 ループのクローズ

多くのインライン・サービスは自己学習が行われるように設計されています。CrossSell インライン・サービスでは、OfferResponse インフォーマントにより、抱合せ販売オファーに対する関心度が選択肢イベント・モデルにフィードバックされます。

```
// Tell the server the good news.
request = client.createRequest(INLINE_SERVICE_NAME, "OfferResponse");
request.setSessionKey( SESSION_KEY, sCustID );
request.setArg( "choiceName", prodName );

// "Interested" is one of the Choice Events defined for the choice group,
Offers.
```

> 選択肢イベント・モデルと選択肢を特定するには、第 1.2.3項「サーバーにレスポンスする方法の理解」を参照してください。

```
request.setArg( "choiceOutcome", "Interested" );
client.invoke(request);
```

最後に、CallResolution インフォーマント（サンプルの CrossSell でセッションの終了用に設計されている）をサーバーにおいて起動することでセッションがクローズします。

```
// Close the server's session.
request = client.createRequest(INLINE_SERVICE_NAME, "CallResolution");
request.setSessionKey( SESSION_KEY, sCustID );
client.invoke(request);
```

2.2.10 クライアントのクローズ

クライアント・アプリケーションで SDClientInterface の使用を終了して再利用しない場合は、コンポーネントの close メソッドをコールし、インスタンス固有の情報を解放します。

```
client.close();
```

3　Java スマート・クライアントの JSP タグの使用

Web アプリケーションをデプロイ済インライン・サービスと統合する便利な方法として、JSP クライアント統合タグを使用する方法があります。JSP を使用すると、埋込みの Java を使用する対話型 Web ページを作成できます。用意されている JSP タグは、前の項で説明した Java スマート・クライアントがベースになっています。

JSP タグを使用するときのオーバーヘッドは無視できます。また、JSP タグには、スマート・クライアントを同一セッションで自動的に再利用する機能があるためパフォーマンスが向上します。JSP タグを使用して Java スマート・クライアントが作成されると、同じ名前とプロパティを持つクライアントが存在してクローズされていないかどうかがチェックされます。存在する場合は、そのクライアントが自動的に再利用されます。存在しない場合は、新しいクライアントが作成されます。

3.1　作業前の準備

JSP クライアント統合タグで作業を行う前に、次の作業を実行する必要があります。

1. Java Development Kit（JDK）をインストールし、*JAVA_HOME* 環境変数にそのインストール場所を設定します。JDK の入手方法の詳細は、Sun 社の Web サイト http://java.sun.com/products/ を参照してください。

2. Oracle RTD ファイルをインストールし、Oracle RTD をアプリケーション・サーバーにデプロイします。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。

3. Java スマート・クライアントのサンプルは、サンプルの CrossSell インライン・サービスと連携して動作します。そのため、最初に Oracle RTD Database に CrossSell のサンプル・データを移入してから、Decision Studio を使用して CrossSell インライン・サービスをデプロイする必要があります。Oracle RTD Database に CrossSell のサンプル・データを移入する方法の詳細は、『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。インライン・サービスのデプロイ方法の詳細は、『Oracle Real-Time Decisions Decision Studio リファレンス・ガイド』を参照してください。

4. Real-Time Decision Server を起動します。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。

3.2　Java スマート・クライアントの JSP タグによるインライン・サービスとの統合

一般的に、Java スマート・クライアントを使用した統合は、次の手順に従って実行します。

1. プロパティ・ファイルを準備します。

2. Invoke タグまたは AsyncInvoke タグを使用して、サーバーへのリクエストを作成します。

3. レスポンス情報をアドバイザから収集して分析します。

4. 接続を切断します。

3.2.1 スマート・クライアントの JSP タグのリファレンス

スマート・クライアントの JSP タグの詳細は、Decision Studio のオンライン･ヘルプを参照してください。

3.2.2 サンプルの JSP コード

スマート・クライアントの JSP タグを使用した統合サンプル・コードが、*RTD_HOME*¥client¥Client Examples¥JSP Client Example¥example.jsp に用意されています。

3.2.3 JSP スマート・クライアント・サンプルのデプロイ

この項では、簡単なコマンドライン・アプリケーションに統合された CrossSell インライン・サービスを使用して、統合にスマート・クライアントを使用する方法について説明します。JSP スマート・クライアント・サンプルをアプリケーション・サーバーにデプロイするには、次の各項の手順に従う必要があります。

3.2.3.1 OC4J への JSP スマート・クライアント・サンプルのデプロイ

1 Application Server Control に oc4jadmin ユーザーとしてログインします。Application Server Control には、http://*oc4j_host*:*port*/em でアクセスできます。スタンドアロン版の OC4J では通常、ポート番号は 8888 です。

2 OC4J のホーム・ページで、「**アプリケーション**」タブをクリックします。

注意: OC4J を Oracle Application Server の一部として使用している場合は、最初に「グループ」の下の「**ホーム**」をクリックしてから、「**アプリケーション**」タブに進みます。

3 「**デプロイ**」をクリックします。「デプロイ: アーカイブの選択」ページの「アーカイブ」で、アーカイブのある場所を参照し *RTD_HOME*/client/Client Examples/JSP Client Example/sdclient-test.war を指定します。「**次へ**」をクリックします。

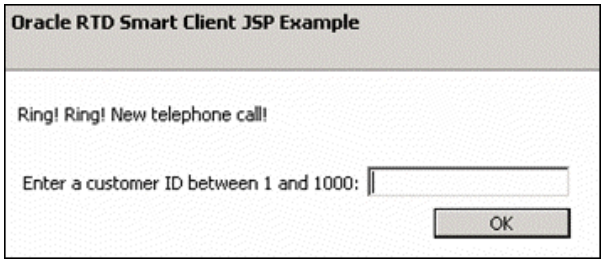
4 「デプロイ: アプリケーション属性」ページで、「**アプリケーション名**」に「JSPClientExample」と入力し、「**サイトへの Web モジュールのバインド**」に「rtd-web-site」を選択します。「**次へ**」をクリックします。

5 「デプロイ: デプロイ設定」ページで、「**デプロイ**」をクリックします。

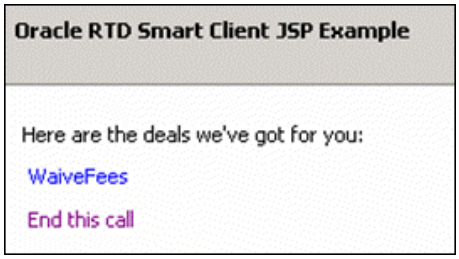
6 アプリケーションにアクセスするには、Web ブラウザを起動して、次の URL に移動します。

http://*ocj4_host*:8080/sdclient-test/example.jsp

サービスのコールをシミュレートする Web ページが表示されます。次に例を示します。



7 顧客 ID（たとえば、5）を入力し、「**OK**」をクリックします。レスポンス・ページが表示され、コールを終了するオプションとオファーが表示されます。



8 オファーのリンクをクリックするか、「**End this call**」をクリックします。

3.2.3.2 WebSphere への JSP スマート・クライアント・サンプルのデプロイ

1 `http://`*`websphere_host`*`:`*`port`*`/ibm/console` の URL で、Integrated Solutions Console にアクセスします。ログイン・プロンプトで、管理者のユーザー名とパスワードを入力します。Windows では、「**スタート**」→「**プログラム**」から Integrated Solutions Console にアクセスすることもできます。

2 左側のツリーで、「**Applications**」を開き、「**Enterprise Applications**」を選択します。

3 「**Install**」をクリックします。

4 新しいアプリケーション・セクションの「Path」に「*`RTD_HOME`*`/client/Client Examples/JSP Client Example/sdclient-test.war`」と入力するか、入力パスを参照します。

5 「**Context root**」に、「`sdclient-test`」と入力します。

6 「**Next**」→「**Next**」→「**Next**」をクリックします。

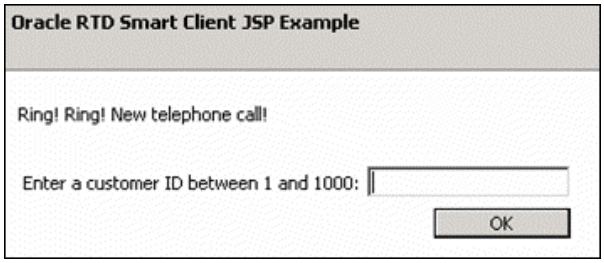
7 「**Finish**」→「**Save**」をクリックします。

8 「Enterprise Applications」ページで、「**sdclient-test**」アプリケーションを選択し、「**Start**」をクリックします。

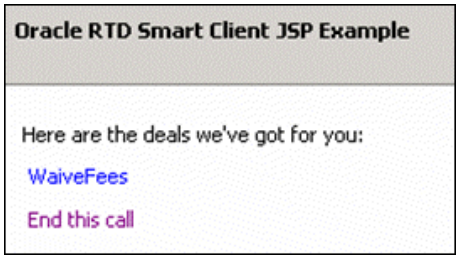
9 アプリケーションにアクセスするには、Web ブラウザを起動して、次の URL に移動します。

`http://`*`websphere_host`*`:8080/sdclient-test/example.jsp`

サービスのコールをシミュレートする Web ページが表示されます。次に例を示します。



10 顧客 ID（たとえば、5）を入力し、「**OK**」をクリックします。レスポンス・ページが表示され、コールを終了するオプションとオファーが表示されます。



11 オファーのリンクをクリックするか、「**End this call**」をクリックします。

3.2.3.3 WebLogic への JSP スマート・クライアント・サンプルのデプロイ

1 `http://`*`weblogic_host:port`*`/console` の URL から、Oracle RTD ドメインの WebLogic Server Administration Console にアクセスします。ログイン・プロンプトで、管理者のユーザー名とパスワードを入力します。Windows では、「**スタート**」→「**プログラム**」→「**BEA Products**」→「**User Projects**」→「***domain_name***」→「**Admin Server Console**」から WebLogic Server Administration Console にアクセスすることもできます。

2 左側のツリーで、「**Deployments**」をクリックします。

3 「**Install**」をクリックします。「**Install**」ボタンを有効にするには、最初に「**Lock & Edit**」をクリックする必要があります。

4 *`RTD_HOME`*`/client/Client Examples` に移動して「**JSP Client Example**」を選択し、「**Next**」をクリックします。

5 「**Install this deployment as an application**」を選択して、「**Next**」をクリックします。

6 「Optional Settings」ページで、「**Name**」に「`JSPClientExample`」と入力します。「**Next**」をクリックします。

7 設定を確認して「**Finish**」をクリックします。

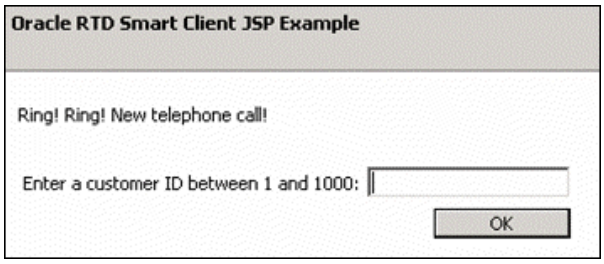
8 「**Save**」→「**Activate Changes**」をクリックします。

9 「**Deployments**」テーブルで「**JSPClientExample**」アプリケーションを選択し、「**Start**」→「**Servicing all Requests**」をクリックして、アプリケーションを起動します。メッセージが表示されたら、「**Yes**」をクリックします。アプリケーションが実行中になります。

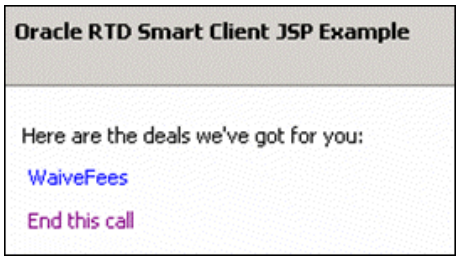
10 アプリケーションにアクセスするには、Web ブラウザを起動して、次の URL に移動します。

```
http://weblogic_host:8080/JSP %20Client %20Example/example.jsp
```

サービスのコールをシミュレートする Web ページが表示されます。次に例を示します。



11 顧客 ID（たとえば、5）を入力し、「**OK**」をクリックします。レスポンス・ページが表示され、コールを終了するオプションとオファーが表示されます。



12 オファーのリンクをクリックするか、「**End this call**」をクリックします。

4 .NET スマート・クライアントの使用

.NET スマート・クライアントにより、Java API とほぼ同等のクライアントを実現し、アプリケーションからコールします。.NET スマート・クライアントの現行の実装では、Java スマート・クライアントが持つ高度な機能の一部（セッション・アフィニティ管理やデフォルトのレスポンス処理など）がサポートされていません。

.NET スマート・クライアントは、*RTD_HOME*¥client¥Client Examples¥Dot Net Client Example¥sdclient.dll にあります。このファイルをアクセス可能にするには、アプリケーションと同じ場所に配置する必要があります。

4.1 作業前の準備

.Net スマート・クライアントで作業を行う前に、次の作業を実行する必要があります。

1. Java Development Kit（JDK）をインストールし、*JAVA_HOME* 環境変数にそのインストール場所を設定します。JDK の入手方法の詳細は、Sun 社の Web サイト http://java.sun.com/products/ を参照してください。

2. Oracle RTD ファイルをインストールし、Oracle RTD をアプリケーション・サーバーにデプロイします。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。

3. .NET スマート・クライアントのサンプルは、サンプルの CrossSell インライン・サービスと連携して動作します。そのため、最初に Oracle RTD Database に CrossSell のサンプル・データを移入してから、Decision Studio を使用して CrossSell インライン・サービスをデプロイする必要があります。Oracle RTD Database に CrossSell のサンプル・データを移入する方法の詳細は、『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。インライン・サービスのデプロイ方法の詳細は、『Oracle Real-Time Decisions Decision Studio リファレンス・ガイド』を参照してください。

4. Real-Time Decision Server を起動します。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 .NET スマート・クライアントによるインライン・サービスとの統合

この項では、.NET スマート・クライアントを使用して、Oracle RTD 上のデプロイ済インライン・サービスと統合する方法について概略を説明します。

一般的な統合手順は次のとおりです。

1. RTD スマート・クライアントをアプリケーション・コード内に作成します。

2. インライン・サービスと統合点に対して送信するリクエストを作成します。

3. リクエストにおいて引数とセッション・キーを移入します。

4. スマート・クライアントを使用してリクエストを起動します。

5. リクエストがアドバイザに対して起動された場合、レスポンスを確認します。

6. 終了したらスマート・クライアントをクローズします。

4.2.1 .NET API のリファレンス

.NET スマート・クライアント API の詳細は、Decision Studio のオンライン･ヘルプを参照してください。

4.2.2 .NET 統合のサンプル

.NET 統合クライアントのサンプルは、*RTD_HOME*¥client¥Client Examples¥Dot Net Client Example¥DotNetSmartClientExample.sln にあります。このサンプルは、Microsoft Visual Studio を使用して開いて実行やデバッグができます。

このサンプルでは、インフォーマントとアドバイザの統合点が CrossSell インライン・サービスに対して起動されます。このインライン・サービスの理解を深めるには、第1.2項「CrossSell インライン・サービスについて」を参照してください。サンプルでは、統合点が起動され、アドバイザからの戻り値がコンソールに出力されます。

サンプルを Microsoft Visual Studio で実行する手順は次のとおりです。

1 **Microsoft Visual Studio .NET** を起動します。

2 メニュー・バーから、「**File**」→「**Open**」→「**Project**」を選択します。

3 「**File Name**」に「*RTD_HOME*¥client¥Client Examples¥Dot NET Client Example¥DotNetSmartClientExample.sln」を選択し、「**Open**」をクリックします。

4 Real-Time Decision Server が別のコンピュータで実行している場合は、右側の「Solution Explorer」ウィンドウで、「**DotNetSmartClientExample.cs**」をダブルクリックしてから、次の行を探します。

```
SDClient client = new SDClient("http://localhost:8080");
```

localhost を、Real-Time Decision Server が実行しているコンピュータの名前に変更します。ファイルを保存して閉じます。

5 メニュー・バーから、「Debug」→「Start」を選択します。コンソール・ウィンドウに、次のテキストが表示されます。

6 カーソルをコロンの後に移動し、顧客 ID（たとえば、5）を入力し、[Enter]を押します。次のようなレスポンスが表示されます。

7 カーソルを最後のコロンの後に移動し、「1」を入力してオファーを選択します。サーバーによって、最後のメッセージがレスポンスされます。

8 プロセスが繰り返されます。プログラムを終了するには、数字を入力せずに、顧客 ID プロンプトで[Enter]を押します。

5　ゼロ・クライアントによる統合

Real-Time Decision Server の統合点は、ゼロ・クライアントの手法でも使用できます。デプロイ済のインライン・サービスの統合点は、Web サービスの定義によって公開されます。

統合点の起動プロセスを理解するには、第2章「Java スマート・クライアントの使用」に説明されているチュートリアルを実行することをお薦めします。

5.1　Web サービスについて

デプロイ済の統合点を同期して起動したり非同期で起動する機能は、Real-Time Decision Server によって Web サービスとして公開されます。この操作の定義は、WSDL ファイルの *RTD_HOME*¥deploy¥DecisionService¥DecisionService.wsdl から入手できます。WSDL ファイルには、使用可能なすべての複合型と操作が定義されています。

リリース 2.2 では構造がわずかに変更されて、デシジョン・サービスが WS-I Basic レベルに準拠するよう改善されています。以前のバージョンの WSDL ファイルは、*RTD_HOME*¥deploy¥DecisionService¥DecisionServiceLegacy.wsdl という名前です。新しい WSDL を使用して新しいクライアントを開発することをお薦めしますが、DecisionServiceLegacy.wsdl に定義されているプロトコルにもサーバーはこれまでと同様に対応でき、既存クライアント機能が実現されます。