

Oracle9iAS Personalization

プログラマーズ・ガイド

リリース 9.0.2

2002 年 9 月

部品番号 : J06021-02

ORACLE®

Oracle9iAS Personalization プログラマーズ・ガイド, リリース 9.0.2

部品番号: J06021-02

原本名: Oracle9iAS Personalization Programmer's Guide, Release 2 (v9.0.2)

原本部品番号: A95245-02

Copyright © 2001, 2002 Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル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に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されております。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation（米国オラクル）または日本オラクル株式会社（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、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を用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

目次

| | |
|------------|------|
| はじめに | vii |
| 対象読者 | viii |
| 構成 | viii |
| 関連文書 | ix |
| 表記規則 | x |

1 Oracle9i/AS Personalization プログラミング

| | |
|------------------------|-----|
| OP API の構成 | 1-2 |
| OP プログラムの実行 | 1-2 |
| OP API の Javadoc | 1-2 |

第 I 部 リコメンデーション・エンジン API

2 REAPI 概要

| | |
|-------------------------------------|-----|
| REAPI 前提条件 | 2-2 |
| REAPI の定義と概念 | 2-2 |
| REAPI エンド・ユーザー（顧客と匿名ユーザー） | 2-2 |
| Web アプリケーションとセッション | 2-2 |
| セッション対応型の REAPI Web アプリケーション | 2-3 |
| 非セッション対応型の REAPI Web アプリケーション | 2-3 |
| REAPI データ収集 | 2-3 |
| REAPI によるリコメンデーション | 2-4 |
| REAPI ホット・ピックス | 2-4 |
| REAPI を使用する前に | 2-4 |

| | |
|---------------------------------|-----|
| REAPI Demo プログラム | 2-5 |
| REProxyRT オブジェクトの作成 | 2-5 |
| REAPI セッションの開始 | 2-6 |
| REAPI サポート・クラスのインスタンスの作成 | 2-6 |
| REAPI によるリコメンデーション用のデータ収集 | 2-7 |
| OP データ・キャッシング | 2-7 |
| REAPI のリコメンデーションの取得 | 2-7 |
| REAPI でのリコメンデーションの作成方法 | 2-8 |
| 匿名ユーザーのスコアリング | 2-8 |
| 顧客のスコアリング | 2-8 |
| REAPI によるリコメンデーションの作成 | 2-8 |
| REAPI セッションの終了 | 2-9 |
| REProxyRT オブジェクトの削除 | 2-9 |

3 REAPI サポート・クラス

| | |
|---|------|
| OP のレーティング | 3-2 |
| REAPI クラスの位置 | 3-2 |
| REAPI EnumType インタフェース | 3-2 |
| REAPI CategoryMembership インタフェース | 3-3 |
| REAPI DataSource インタフェース | 3-4 |
| REAPI Filtering インタフェース | 3-4 |
| REAPI InterestDimension インタフェース | 3-5 |
| REAPI PersonalizationIndex インタフェース | 3-5 |
| REAPI ProfileDataBalance インタフェース | 3-6 |
| REAPI ProfileUsage インタフェース | 3-6 |
| REAPI RecommendationAttribute インタフェース | 3-7 |
| REAPI Sorting インタフェース | 3-8 |
| REAPI User インタフェース | 3-8 |
| その他の REAPI サポート・クラス | 3-9 |
| ContentItem クラス | 3-9 |
| DataItem クラス | 3-10 |
| FilteringSettings クラス | 3-10 |
| IdentificationData クラス | 3-11 |
| Item クラス | 3-12 |
| ItemDetailData クラス | 3-12 |

| | |
|---------------------------------|------|
| Recommendation クラス | 3-12 |
| RecommendationContent クラス | 3-13 |
| RecommendationList クラス | 3-13 |
| TuningSettings クラス | 3-13 |

4 REAPI の使用

| | |
|--------------------------|-----|
| リコメンデーション・プロキシ・クラス | 4-2 |
| RE プロキシ・クラスの位置 | 4-2 |
| RE プロキシの作成と管理 | 4-2 |
| RE データ収集 | 4-3 |
| REProxyManager クラス | 4-3 |
| プロキシ・メソッド | 4-3 |
| RE プロキシ・セッション管理 | 4-4 |
| RE プロキシ・データの収集と管理 | 4-4 |
| RE プロキシの顧客登録 | 4-4 |
| RE プロキシのリコメンデーション | 4-4 |
| OP のレーティング | 4-5 |
| リコメンデーションの戻り値の意味 | 4-5 |
| 規則とリコメンデーション | 4-6 |
| RE プロキシ・メソッドの使用法 | 4-6 |
| セッションの作成 | 4-6 |
| データ収集 | 4-6 |
| アイテムの追加 | 4-6 |
| アイテムの削除 | 4-7 |
| プロキシの作成 | 4-7 |
| キャッシュ・サイズ | 4-7 |
| 時間隔 | 4-8 |
| 抱合せ販売メソッド | 4-8 |
| プロキシの破棄 | 4-9 |

5 REAPI の例と使用方法

| | |
|-------------------------|-----|
| REAPI Demo | 5-2 |
| REAPI の基本的な使用方法 | 5-2 |
| REProxy オブジェクトの作成 | 5-2 |
| プロキシの使用 | 5-3 |

| | |
|-----------------------------------|------|
| プロキシの破棄 | 5-3 |
| セッション対応型の Web アプリケーションの概要 | 5-4 |
| 非セッション対応型の Web アプリケーションの概要 | 5-5 |
| JVM と REProxyManager との相互作用 | 5-5 |
| スタンドアロン Java アプリケーション | 5-6 |
| Java サーバーサイド・モジュール | 5-6 |
| REProxy の複数インスタンスの使用 | 5-7 |
| 初期化のフェイル・セーフ | 5-7 |
| 中断されない REAPI サービス | 5-8 |
| ロード・バランシング | 5-9 |
| 個別のリコメンデーションの抽出 | 5-9 |
| 複数通貨の処理 | 5-9 |
| リコメンデーション・エンジンの使用方法 | 5-10 |
| 人口統計データの使用 | 5-11 |
| 時間ベースのアイテムの処理 | 5-11 |

第 II 部 リコメンデーション・エンジン・バッチ API

6 RE Batch API の概要

| | |
|-------------------------------------|-----|
| RE Batch API 前提条件 | 6-2 |
| RE Batch API の定義と概念 | 6-2 |
| RE Batch API エンド・ユーザー（顧客） | 6-2 |
| RE Batch API によるリコメンデーション | 6-2 |
| RE Batch API の使用 | 6-2 |
| RE Batch API 環境の設定 | 6-3 |
| 顧客プロフィール・データ | 6-3 |
| RE へのパッケージ配布 | 6-3 |
| RE Batch API の使用例 | 6-3 |
| REBatchProxy オブジェクトの作成 | 6-4 |
| RE Batch API オブジェクトのインスタンスの作成 | 6-4 |
| RE Batch API のデータ変換 | 6-4 |
| RE Batch API の顧客プロフィールの管理 | 6-4 |
| RE API バッチ・リコメンデーションの取得 | 6-4 |
| OP のレーティング | 6-5 |
| リコメンデーションの作成 | 6-5 |
| スコアリング： | 6-5 |

| | |
|------------------------------|-----|
| RE バッチ・リコメンデーションの作成 | 6-5 |
| REBatchProxy オブジェクトの削除 | 6-5 |

7 RE Batch API サポート・クラス

| | |
|---------------------------------------|------|
| OP のレーティング | 7-2 |
| RE Batch API クラスの位置 | 7-2 |
| RE Batch API の EnumType インタフェース | 7-2 |
| CategoryMembership インタフェース | 7-3 |
| DataSource インタフェース | 7-3 |
| InterestDimension インタフェース | 7-4 |
| PersonalizationIndex インタフェース | 7-5 |
| ProfileDataBalance インタフェース | 7-5 |
| ProfileUsage インタフェース | 7-6 |
| Sorting インタフェース | 7-7 |
| その他の RE Batch API サポート・クラス | 7-7 |
| DataItem クラス | 7-7 |
| FilteringSettings クラス | 7-8 |
| Item クラス | 7-9 |
| Location クラス | 7-9 |
| TuningSettings クラス | 7-10 |

8 リコメンデーション・エンジン・バッチ・プロキシの使用

| | |
|----------------------------------|-----|
| REProxyBatch の概要 | 8-2 |
| REProxyBatch クラスの位置 | 8-2 |
| REProxyBatch の作成と管理 | 8-2 |
| 顧客プロファイル管理 | 8-2 |
| REProxyBatch のリコメンデーション | 8-2 |
| OP のレーティング | 8-3 |
| リコメンデーションの戻り値の意味 | 8-3 |
| 抱合せ販売メソッドの使用方法 | 8-3 |
| リコメンデーション・メソッドの使用方法 | 8-4 |
| REProxyBatch の規則とリコメンデーション | 8-4 |

9 REProxyBatch API の例と使用方法

| | |
|----------------------------------|-----|
| REProxyBatch API の基本的な使用方法 | 9-2 |
|----------------------------------|-----|

| | |
|----------------------------------|-----|
| コード・サンプル:顧客情報によるリコメンデーション | 9-2 |
| コード・サンプル:抱合せ販売によるリコメンデーション | 9-3 |
| リコメンデーション・エンジンの使用方法 | 9-3 |
| 複数通貨の処理 | 9-4 |
| 人口統計データの使用 | 9-4 |
| 時間ベースのアイテムの処理 | 9-5 |

A REAPI サンプル・プログラム

B REProxyBatch サンプル・プログラム

| | |
|-----------------------------|-----|
| RE バッチ・サンプル・プログラムの概要 | B-2 |
| RE バッチ・サンプル・プログラムの出力 | B-2 |
| RE バッチ・サンプル・プログラムの実行 | B-2 |
| RE バッチ・サンプル・プログラム・コード | B-4 |
| batchtest.txt | B-4 |
| REBatchTest.java | B-7 |

索引

はじめに

このマニュアルでは、Java プログラマが Oracle9iAS Personalization (OP) リコメンデーション・エンジン API (REAPI) を使用してリアルタイムでデータを収集し、リコメンデーションを取得する方法について説明します。

対象読者

このマニュアルは、Oracle9iAS Personalization を使用した Web サイトを作成、維持する Java プログラマを対象としています。

構成

このマニュアルには、次の章や付録が含まれます。

| | |
|-------|--|
| 第 1 章 | OP API について説明します。 |
| 第 2 章 | REAPI の概要について説明します。 |
| 第 3 章 | REAPI サポート・クラスについて説明します。 |
| 第 4 章 | セッションの管理、データの管理、リコメンデーションの要求に使用する REAPI メソッドについて説明します。 |
| 第 5 章 | REAPI を使用した共通タスクの実行例を示します。 |
| 第 6 章 | RE Batch API の概要について説明します。 |
| 第 7 章 | RE Batch API サポート・クラスについて説明します。 |
| 第 8 章 | リコメンデーションの要求に使用する RE Batch API メソッドについて説明します。 |
| 第 9 章 | RE Batch API を使用した共通タスクの実行例を示します。 |
| 付録 A | REAPI の使用例を示します。 |
| 付録 B | RE Batch API の使用例を示します。 |

関連文書

このリリースの Oracle9iAS Personalization には、次のマニュアルがあります。

- 『Oracle9iAS Personalization リリース・ノート リリース 9.0.2』。製品に関する最新情報と、プラットフォームごとのインストール情報が記載されています。
- 『Oracle9iAS Personalization 管理者ガイド リリース 9.0.2』。すべてのプラットフォームに共通するインストール情報が記載されています。
- 『Oracle9iAS Personalization ユーザーズ・ガイド リリース 9.0.2』。
- 『Oracle9iAS Personalization プログラマーズ・ガイド リリース 9.0.2』 (本書)。リアルタイムでリコメンデーション・エンジンにアクセスし、大量のリコメンデーションを取得する方法についてプログラマ向けに解説しています。
- 『Oracle9iAS Containers for J2EE タグ・ライブラリおよびユーティリティ・リファレンス』。第 9 章で OP タグについて説明しています。
- REAPI クラスと REAPI メソッドの詳細は、ドキュメント CD に含まれている Oracle9i Application Server Documentation Library の OP セクションにある javadoc を参照してください。

関連マニュアル

- 『Oracle9i 管理者ガイド』。
- 『Oracle9i Application Server 概要』。
- 『Oracle9i Application Server インストレーション・ガイド』。使用しているオペレーティング・システムに対応したバージョンを参照します。

ドキュメントの形式

Oracle9iAS Personalization のドキュメントは、PDF 形式で提供します。

PDF ファイルを表示するには、次のビューアが必要です。

- Adobe Acrobat Reader 4.0 以降 (<http://www.adobe.co.jp> からダウンロード可能)

表記規則

このマニュアルでは、Windows98 および Windows NT オペレーティング・システムを総称して Windows と記載します。

Oracle9i との SQL インタフェースを SQL と記載します。このインタフェースは、一般に ANSI/ISO SQL 規格または SQL92 と呼ばれる SQL 規格 ANSI X3.135-1992, ISO 9075:1992 の Oracle9i 実装です。

例の中では、特に記載がない限り、各行末に暗黙の改行があります。入力行末では [Return] キーを押す必要があります。

次の表に、このマニュアルの表記規則とその意味を示します。

| 表記規則 | 意味 |
|-----------------|---|
| | 例の中の縦の省略記号は、その例に直接関係のない情報が省略されていることを示します。 文またはコマンド中の横の省略記号は、その例に直接関係のない文やコマンドの一部が省略されていることを示します。 |
| 太字 | 本文中の太字は、本文か用語集または両方に用語が定義されていることを示します。 |
| イタリックのテキスト | イタリックのテキストや構文は、ユーザー指定の名前やデータを示します。 |
| < > | 山カッコはユーザー指定の名前を囲みます。 |
| [] | 大カッコは、1 つを選択するか、選択しなくてもよい任意の句を囲みます。 |

Oracle9iAS Personalization プログラミング

Oracle9iAS Personalization は、次のような 2 つの Java アプリケーション・プログラム・インタフェース（API）を備えています。

- リコメンデーション・エンジン API（REAPI）
- リコメンデーション・エンジン・バッチ API（RE Batch API）

REAPI を使用すると、Java で記述された Web アプリケーションで、OP モデルの作成に使用したデータを収集、前処理し、リコメンデーションを要求できます。リコメンデーションは、リアルタイムで返されます。REAPI については、このマニュアルの [第 I 部](#) に記載されています。

RE Batch API を使用すると、Java で記述された Web アプリケーションはバルク・モードで Oracle9iAS Personalization スタイルのリコメンデーションを要求できます。リコメンデーションは表に書き込まれます。RE Batch API はリアルタイムで結果を返しません。REAPI Batch については、このマニュアルの [第 II 部](#) に記載されています。

OP API の構成

2 つの OP API には、同じコンポーネントがあります。

- サポート・クラス（マイニング操作の絞込み条件を設定するのに使用）
- プロキシ・クラス（リコメンデーションを取得するのに使用）

OP プログラムの実行

いずれかの OP API を使用してプログラムを実行する前に、『Oracle9iAS Personalization ユーザーズ・ガイド』の手順に従い、OP パッケージをビルド、配布する必要があります。

OP API の Javadoc

OP API の詳細は、このマニュアルには記載されていません。API コールは Javadoc に記載されています。Oracle9i Application Server Documentation Library の OP のセクションを参照してください。

第I部

リコメンデーション・エンジン API

第I部では、OP (Oracle9iAS Personalization) REAPI (リコメンデーション・エンジン Application Program Interface (API)) について説明します。REAPI を使用すると、セッション中に Web アプリケーションでターゲット・データを収集し、リコメンデーションを返すことができます。

第I部は、次の章で構成されます。

- 第2章「REAPI 概要」
- 第3章「REAPI サポート・クラス」
- 第4章「REAPI の使用」
- 第5章「REAPI の例と使用方法」

REAPI の使用に関する詳細な例は、[付録 A](#) を参照してください。

REAPI クラスの詳細は、Oracle9i Application Server Documentation Library の OP のセッションにある Javadoc を参照してください。

REAPI 概要

OP (Oracle9iAS Personalization) REAPI (リコメンデーション・エンジン Application Program Interface (API)) を使用すると、Java で記述された Web アプリケーションで、OP モデルの作成に使用したデータを収集、前処理し、リコメンデーションを要求できます。リコメンデーションは、リアルタイムで返されます。

また、OP には、リコメンデーションが一括して返されるリコメンデーション・エンジン・バッチ API も組み込まれています。

REAPI は、拡張性に優れ、API 関数を少なくし、均一性を保ち、引数の数が最小限になるよう設計されています。

[付録 A](#) に REAPI の詳しい使用例が記載されています。

OP には API メソッドの動作を理解できるようデモ・プログラムが含まれています。

REAPI クラスとメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

注意： REAPI および REAPI Demo は、Oracle9iAS がインストールされているシステムにインストールされます。

REAPI 前提条件

REAPI メソッドを使用する前に、OP をインストールし、適切な表を作成して、データを移入する必要があります。既存のデータを使用する場合、適切なスキーマを使用するにはデータを変換する必要があります。ホット・ピックスを使用する場合、ホット・ピックスとホット・ピックス・グループを指定する必要があります。1 つ以上の分類を使用する場合、適切に指定する必要があります。

リコメンデーションを要求する場合、その前に、OP パッケージをビルドし、配布する必要があります。これには、OP 管理 UI を使用します。

OP のインストール方法の詳細は、『Oracle9i Application Server インストレーション・ガイド』と『Oracle9iAS Personalization 管理者ガイド』を参照してください。パッケージの作成と配布の詳細は、『Oracle9iAS Personalization ユーザーズ・ガイド』と OP 管理 UI のオンライン・ヘルプを参照してください。

REAPI の定義と概念

この項では、REAPI を構成する各種のメソッドと、API を説明する上での概念と用語について説明します。

REAPI エンド・ユーザー（顧客と匿名ユーザー）

エンド・ユーザー（パーソナライズ・サービスの OP を利用する Web サイト・ユーザー）は顧客と匿名ユーザーという 2 つのグループに分類されます。顧客とは登録ユーザーで、Web アプリケーションにより割り当てられた一意の顧客 ID で識別できます。匿名ユーザーとは未登録ユーザーで、通常 Web アプリケーションにより匿名ユーザー ID が割り当てられます。匿名ユーザーは登録をすれば、顧客になることができます。エンド・ユーザーは、IdentificationData クラスを使用して指定されます。

Web アプリケーションとセッション

一部の Web アプリケーションはステートフルです。すなわち、これらのアプリケーションが、所定の時間クライアント・アクティビティの状態を維持することを意味します。他の Web アプリケーションはステートレスです。E-Commerce をサポートする多くの Web アプリケーションはステートフル（セッション対応型）です。クライアント・セッションは通常、ログインで始まり、明示的なログアウトか、セッション・タイムアウトによって終了します。Oracle9iAS Personalization は、アプリケーションがステートフルかステートレスかに関係なく、データ・マイニングを目的として独自のセッションを保持します。アプリケーションがステートフルである場合、OP が保持するセッションはアプリケーションのセッションとして完全にマッピングできます（E-Commerce アプリケーションの場合、ユーザーへのリコメンデーションはユーザー・アクティビティに基づきます）。アプリケーションがユーザー・セッションを保持しない場合、OP は独自にユーザー・セッションを管理します。この場合、OP セッションは、特定のユーザー ID が REAPI メソッド・コールで初めて発生

したときに始まり、セッション・タイムアウト、すなわちユーザーがあらかじめ設定した時間非アクティブになると終了します。

要約すると、REAPI を呼び出す Web アプリケーションは次のいずれかです。

- セッション対応型（ステートフル）：Web サイトを訪れた各ユーザーにセッションを作成します。
- 非セッション対応型（ステートレス）：セッションを作成しません。

OP は常にセッション対応型で、Web アプリケーションが作成しない場合でもセッションを作成します。

Web アプリケーションでは、OP セッション中にデータの収集やリコメンデーションの要求を行うことができます。

セッション対応型の REAPI Web アプリケーション

Web アプリケーションがセッション対応型である場合、OP はそのセッションをアプリケーション・セッションにマッピングします。セッション対応型アプリケーションを作成するには、次のメソッドのいずれかを使用します。

- `createCustomerSession`: 顧客（登録ユーザー）にセッションを作成します。
- `createVisitorSession`: 匿名ユーザー（未登録ユーザー）にセッションを作成します。

Web アプリケーションは、クラス `IdentificationData` の `createSessionful()` メソッドを使用して、セッション中に使用する識別データを作成します。

非セッション対応型の REAPI Web アプリケーション

Web アプリケーションが非セッション対応型である場合、リコメンデーション・エンジン (RE) は独自に OP セッションを保持します。OP セッションは、所定の `customerId` で最初の REAPI メソッド（データ収集か、リコメンデーション要求）が発行されたときに作成されます。RE は、セッションがタイムアウトになるまで、すなわちその `customerId` が指定の時間非アクティブになるまで、ユーザー・アクティビティを管理します。

Web アプリケーションは、クラス `IdentificationData` の `createSessionless()` メソッドを使用し、セッションのユーザー ID を作成します。

REAPI データ収集

OP では、人口統計データ、購入、レーティング、ナビゲーション・データの 4 種類のデータを収集します。収集されるデータの種類の、Web アプリケーションで決定されます。

注意： OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

匿名ユーザーと顧客のデータはどちらも、永続データ（データベースに格納）にすることも、永続データにしないでおくこともできます。データは RE に収集され、マイニング・テーブル・リポジトリ（MTR）データベースで永続化されます。データを永続化するかどうかは、構成パラメータで指定します。永続化するデータの種類とそのタイミングの詳細は、『Oracle9iAS Personalization 管理者ガイド』のデータ同期化の説明を参照してください。

データ収集により、現在のセッション中のユーザー・アクティビティだけでなく、履歴データに基づいて、リコメンデーションを生成できるようになります。

REAPI によるリコメンデーション

匿名ユーザーでも、顧客でも、リコメンデーションは次の 2 種類のデータに基づきます。

- 履歴データ（データベースに格納され、現在のセッションの開始時に検索されます）
- 現在のセッション中に収集されるデータ

REAPI ホット・ピックス

特定の E-Commerce サイトで、ベンダーは「ホット・ピックス」と呼ばれる特定の製品の販売を促進します。たとえば、今週の特売品などがホット・ピックスになります。ホット・ピックス・アイテムは、ホット・ピックス・グループに分類されます。ホット・ピックス・アイテムおよびホット・ピックス・グループは、OP 管理者がマイニング・テーブル・リポジトリ（MTR）に指定します。各ホット・ピックス・グループは（long）int 型の ID で識別します。

REAPI を使用する前に

REAPI を使用する前に、次の事項を確認する必要があります。

- 少なくとも 1 つのリコメンデーション・エンジンを含むリコメンデーション・エンジン・ファームがある。
- リコメンデーション・エンジン・ファームにパッケージが正しく配布されている。

これらの手順を実行する方法は、『Oracle9iAS Personalization ユーザーズ・ガイド』と OP 管理 UI のオンライン・ヘルプを参照してください。

一部の REAPI メソッドは Oracle9i Database に存在するリコメンデーション・エンジン (RE) のデータを収集し、それ以外のメソッドはリコメンデーションを取得します。

データを収集することも、リコメンデーションを取得することもできます。OP 管理 UI で作成した既存の配布パッケージがないと、リコメンデーションは取得できません。また、なんらかのデータがないとパッケージを作成できません。このデータは、REAPI を使用して収集するか、Web アプリケーションで収集し、Oracle データベースに格納している既存のデータから変換することもできます。

OP アプリケーションを設計する場合、複数の RE が必要か、必要な場合はどのように使用するかを決定する必要があります。設計上の考慮事項の説明は、[第 5 章の「リコメンデーション・エンジンの使用方法」](#)を参照してください。

たとえば、リコメンデーションで所得レベル（給料）を考慮する場合、リコメンデーション・アイテムはユーザーが購入可能なものになります。Web サイト・ユーザーが複数の国にわたる場合（たとえば、Web サイトが日本とインドで商品を販売している場合）は、[第 5 章の「複数通貨の処理」](#)を参照してください。

REAPI Demo プログラム

OP には、様々な REAPI メソッドの使用方法を説明する REAPI Demo が含まれています。このサンプル・プログラムを使用して、REAPI コールについて理解することも、OP が正しくインストールされていることを確認することもできます。

OP をインストールしたら、Netscape か Internet Explorer で次の URL を開いて、REAPI Demo を起動します。

```
http://<hostname>:<port>/OP/redemo
```

REAPI Demo のソース・コードを表示するには、「ソース・コード表示」をクリックします。

デモの実行方法の詳細は、『Oracle9iAS Personalization ユーザーズ・ガイド』を参照してください。また、このマニュアルの[第 5 章](#)には REAPI を使用した代表的なタスクの実行例が示され、[付録 A](#)にはすべての REAPI 機能を使用した詳細な例が示されています。

REProxyRT オブジェクトの作成

REAPI メソッドを使用してリコメンデーション要求またはデータ収集要求を処理する前に、指定の RE に接続する REProxyRT オブジェクトを 1 つ以上作成する必要があります。

Web アプリケーション環境では、初期化段階で必要なプロキシをすべて作成してください。アプリケーションを正しく初期化した後にリコメンデーション要求を処理する必要がないので、これは安全な方法です。

初期化中に必要なプロキシをすべて作成できない場合、最初のリコメンデーション要求を処理するときにこれらのプロキシを作成できます。この場合、多数の要求が同時に発生したときに、アプリケーション・コードで適切に競合条件を処理する必要があります。プロキシが作成される前に多数のリコメンデーション要求が発生すると、プロキシの作成は同期プロセス

スなので、1つの要求のみがプロキシ・オブジェクトを作成します。プロキシ・オブジェクトを作成するには数百ミリ秒かかるので、その間に多くの要求が停滞する場合があります。このため、競合状態が発生することがあります。REAPI はマルチスレッド対応であり、このような競合状態が REProxyRT に問題を起すことはありません。ただし、アプリケーションで例外が発生することがあります。

プロキシの詳細は、[第 5 章](#)を参照してください。

REAPI セッションの開始

Web アプリケーションがセッション対応型の場合、セッションを開始する必要があります。Web アプリケーションでは、各アプリケーション・セッションに一意のセッション ID を指定する必要があります。この手順の例は、[第 5 章](#)を参照してください。

Web アプリケーションが非セッション対応型の場合は、セッションを開始する必要がありません（この場合、Web アプリケーションが最初の REAPI コールを行うときに、OP が該当ユーザーの内部セッションを開始します）。

OP は、Web アプリケーションによるユーザー ID の定義に基づいて、各ユーザーのセッションを開始します。2 人が同時にサイトを使用し、双方が同じユーザー ID を使用している場合（およびアプリケーションが異なるセッションを区別しない場合）、OP はどちらのユーザーにも同じセッション ID を割り当てます。OP ではこれらをシングル・ユーザーとして取り扱います。OP セッションのタイムアウト後、このユーザーが再びログインすると、OP は新しいセッション ID を割り当てます。

セッション対応型および非セッション対応型アプリケーションでは、ユーザーのリコメンデーションを取得します。ユーザー ID は一意にする必要があります。

REAPI サポート・クラスのインスタンスの作成

REAPI を使用するには、API メソッド・シグネチャが使用するオブジェクトのインスタンスを作成する必要があります。このインスタンスを作成するには、[第 3 章](#)にある REAPI サポート・クラスを使用します。たとえば、IdentificationData オブジェクトは作成する必要があります。REAPI シグネチャでは、次のクラスを頻繁に使用します。

- IdentificationData
- FilteringSettings
- TuningSettings
- Item
- DataItem
- Recommendation
- Recommendation Content

[第 5 章](#)の例と、[付録 A](#)の詳細な例を参照してください。

REAPI によるリコメンデーション用のデータ収集

OP は、過去および現在（あるいはその両方）のユーザー行動を示すデータに基づいてリコメンデーションを生成します。

Web アプリケーションが Oracle 表に格納したユーザー・データを保持している場合、このデータは、リコメンデーションの生成に使用する前に、変換してマイニング・テーブル・リポジトリ（MTR）に格納する必要があります。

Web アプリケーションは、現在のセッション中にもデータを収集できます。このデータを使用して現在のセッション中にリコメンデーションを作成することも、格納して今後のセッションでリコメンデーションを作成することもできます。

現在のセッション中にデータを収集して、管理するには次のメソッドを使用します。

```
addItem();
addItems();
removeItem();
removeItems();
```

これらのメソッドは、OP リコメンデーション・エンジン（RE）と指定の OP 内部セッション ID のキャッシュに対し情報を追加または削除します。セッション ID は、REAPI メソッドで渡される IdentificationData に格納されます。

OP データ・キャッシング

OP データ収集メソッドの 1 つ（addItem() または addItems()）を呼び出すと、ユーザー・プロファイル・データが Application Server のバッファ（データ・コレクション・キャッシュ）にまず保存されます。データ・コレクション・キャッシュは、REProxy オブジェクトの初期化の一部として作成されます。データ・バッファのサイズはユーザーが構成でき、メソッド REProxyManager.createProxy() の入力パラメータ cacheSize で指定します。バッファに保存されたデータは、定期的にデータベースに保存（アーカイブ）されます。アーカイブの時間隔は同じメソッドの別の入力パラメータ interval で設定します。データ・コレクション・キャッシュは 2 つの同じバッファで構成されます。1 つのバッファをアーカイブしているときは、もう一方を収集データの保存用に使用します。このようにして、データ収集動作は中断なく実行されます。

REAPI のリコメンデーションの取得

リコメンデーションを取得するため、Web アプリケーションは次のリコメンデーション・メソッドの 1 つを呼び出します。

- crossSellForItemFromHotPicks()
- crossSellForItemsFromHotPicks()
- rateItem()

- `rateItems()`
- `recommendBottomItems()`
- `recommendCrossSellForItem()`
- `recommendCrossSellForItems()`
- `recommendFromHotPicks()`
- `recommendTopItems()`
- `selectFromHotPicks()`

これらのメソッドを使用すると、匿名ユーザーと顧客のどちらに対してもリコメンデーションが取得されます。

REAPI でのリコメンデーションの作成方法

OP は、RE キャッシュに格納された規則表を使用して、前述のメソッドで要求したリコメンデーションを計算します。使用する規則表は、作成した REAPI メソッドによって異なります。通常は、規則の先行条件をキャッシュのデータ（履歴および現在のセッション・データ）と一致させ、各種の結果の確率を算出します。これらの項目は確率の順番に並べられ、`numberOfItems`（API 引数）項目が返されます。

RE データベースに十分なメモリーがあれば、RE は、一番最近使用された規則だけでなく、MTR から RE に配布された特定パッケージに対応する規則をすべてキャッシュします。

匿名ユーザーのスコアリング 匿名ユーザーの場合、現在のセッション・データのみを使用します。通常、匿名ユーザーにはナビゲーション・データ（クリック・ストリーム）のみが永続化されますが、Web アプリケーションが匿名ユーザーに対して他の種類のデータを永続化する場合、そのデータもモデル作成に使用されます（OP はパッケージを作成するときにモデルを作成します）。このような異なるメソッドのスコアリングでは、`addItem()` メソッドによって RE キャッシュに格納されたデータのみが使用されます。

顧客のスコアリング 顧客のスコアリングも、匿名ユーザーの場合と同じです。ただし、顧客の場合、履歴データもスコアリングに使用できます。

OP マイニング・テーブル・リポジトリ（MTR）には、細目および集計形式で格納したレーティング履歴、トランザクション・データおよびナビゲーション・データが含まれます。また、MTR には人口統計データも含まれます。顧客をスコアリングするとき、RE は人口統計データと他のデータ・ソース・タイプの集計バージョンを検索します。

REAPI によるリコメンデーションの作成

リコメンデーションを作成する REAPI メソッドは、Web アプリケーションにそのリコメンデーションを返します。この後、Web アプリケーションがユーザーに渡すリコメンデーションを決定します。

REAPI セッションの終了

セッション対応型の Web アプリケーションでは、`closeSession()` を使用して OP セッションを終了する必要があります。明示的な `closeSession()` メソッドがない場合、OP セッションはタイムアウト時に自動的に終了します。

非セッション対応型の Web アプリケーションでは、タイムアウト時に OP セッションが終了します。

非セッション対応型またはセッション対応型の Web アプリケーションの場合、タイムアウト時間を構成パラメータとして指定します。

構成パラメータの詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

REProxyRT オブジェクトの削除

プロキシをプログラムによって破棄する場合、次のメソッドのいずれかを使用できます。

- `destroyProxy()`: 指定した 1 つのプロキシを破棄します。
- `destroyAllProxies()`: すべての既存プロキシを破棄します。

どちらのメソッドも、アクティブ・ステータスであっても、強制的にプロキシを削除します。各種の使用モデルの詳細は、[第 5 章](#)の説明を参照してください。

REAPI サポート・クラス

この章では、REProxy クラスのサポート・クラスについて説明します。これらのクラスは、[第 4 章](#)にあるメソッドで使用するオブジェクトのインスタンスを作成するときに使用します。これらのクラスの多くに 1 つのインスタンスを作成し、その 1 つのインスタンスを複数コールの引数として使用できます。

この章で説明するメソッドは、すべてパブリックです。

サポート・クラスは、次の 2 つのカテゴリに分類されます。

- EnumType インタフェース
- その他のサポート・クラス

この章では、クラスの詳細は説明しません。詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

OP のレーティング

OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

REAPI クラスの位置

oracle.dmt.op.re.base パッケージには次のクラスが含まれています。

- DataItem
- Enum
- FilteringSettings
- Item
- ItemList
- TuningSettings
- RecommendationContent (oracle.dmt.op.re.reapi.rt の 1 つのクラス)

Enum インタフェースを使用するには、Java プログラムに次の文を含める必要があります。

```
import oracle.dmt.op.re.base.Enum;
```

REAPI EnumType インタフェース

REAPI メソッドの多くは、限られた数の値を取る属性を参照します。このような列挙定数のベース・クラスを実装するには、インタフェース Enum を使用します。

Enum インタフェースには、次の一般メソッドを含むネストされた EnumType クラスがあります。

```
int getId()

String toString()

String getName()

boolean isEqual(EnumType)
```

次のインタフェースは、EnumType を拡張します。

- CategoryMembership
- DataSource
- Filtering

- InterestDimension
- PersonalizationIndex
- ProfileDataBalance
- ProfileUsage
- RecommendationAttribute
- Sorting
- User

REAPI CategoryMembership インタフェース

CategoryMembershipType は、次のように実装されます。

- CategoryMembershipType (EnumType を拡張するクラス)
- CategoryMembership (インタフェース)

クラス CategoryMembership には、次のメソッドがあります。

```
CategoryMembershipType getType(String name)
CategoryMembershipType getType(int)
```

CategoryMembership は、カテゴリー一覧のカテゴリにフィルタリングを適用する方法を指定します。たとえば、Enum.CategoryMembership.EXCLUDE_ITEMS は、カテゴリー一覧のカテゴリのアイテムをリコメンデーション一覧から除外することを指定します。詳細は、この章の「[FilteringSettings クラス](#)」を参照してください。

CategoryMembership は、次の値を取ります。

- Enum.CategoryMembership.EXCLUDE_ITEMS
- Enum.CategoryMembership.INCLUDE_ITEMS
- Enum.CategoryMembership.EXCLUDE_CATEGORIES
- Enum.CategoryMembership.INCLUDE_CATEGORIES
- Enum.CategoryMembership.LEVEL
- Enum.CategoryMembership.SUBTREE_ITEMS
- Enum.CategoryMembership.SUBTREE_CATEGORIES
- Enum.CategoryMembership.ALL_ITEMS
- Enum.CategoryMembership.ALL_CATEGORIES

次の文は、変数 myEnum に Enum.CategoryMembership.LEVEL を割り当てます。

```
CategoryMembershipType myEnum = Enum.CategoryMembership.LEVEL
```

REAPI DataSource インタフェース

DataSource は、次のように実装されます。

- DataSourceType (EnumType を拡張するクラス)
- DataSource (インタフェース)

クラス DataSourceType には、次のメソッドがあります。

```
DataSourceType getType(String name)
```

```
DataSourceType getType(int)
```

DataSource は、OP の特定のオペレーションに使用するデータのタイプを指定します。たとえば、Enum.DataSource.DEMOGRAPHIC は、人口統計データを使用することを指定します。この章の後半で説明する「[DataItem クラス](#)」は、DataSource を使用します。1 つのメソッドが DataSource のすべての値に対応するわけではありません。詳細は、[第 4 章](#) のメソッドの説明を参照してください。

DataSource は、次の値を取ります。

- Enum.DataSource.DEMOGRAPHIC
- Enum.DataSource.PURCHASING
- Enum.DataSource.RATING
- Enum.DataSource.NAVIGATION
- Enum.DataSource.ALL

次の文は、変数 myEnum に Enum.DataSource.ALL を割り当てます。

```
DataSourceType myEnum = Enum.DataSource.ALL;
```

REAPI Filtering インタフェース

Filtering は、次のように実装されます。

- FilteringType (EnumType を拡張するクラス)
- Filtering (インタフェース)

クラス FilteringType には、次のメソッドがあります。

```
FilteringType getType(String name)
```

```
FilteringType getType(int)
```

Filtering は、フィルタリングの使用を切り替えるのに使用します。詳細は、この章の「[FilteringSettings クラス](#)」の説明を参照してください。

Filtering は、次の値を取ります。

- Enum.Filtering.ON
- Enum.Filtering.OFF

次の文は、変数 myEnum に Enum.Filtering.OFF を割り当てます。

```
FilteringType myEnum = Enum.Filtering.OFF;
```

REAPI InterestDimension インタフェース

InterestDimension は、次のように実装されます。

- InterestDimensionType (EnumType を拡張するクラス)
- InterestDimension (インタフェース)

クラス InterestDimensionType には、次のメソッドがあります。

```
InterestDimensionType getType(String name)
```

```
InterestDimensionType getType(int)
```

InterestDimension は、特定のアイテムに対して Web サイト・ユーザーが抱く関心の種類を示します。NAVIGATION は、ユーザーがそのアイテムに関心があることを示します。PURCHASING は、ユーザーがそのアイテムを購入したことを示します。RATING は、ユーザーがそのアイテムを好むことを示します。詳細は、この章の「[RecommendationList クラス](#)」および「[TuningSettings クラス](#)」の説明を参照してください。

InterestDimension は、次の値を取ります。

- Enum.InterestDimension.NAVIGATION
- Enum.InterestDimension.PURCHASING
- Enum.InterestDimension.RATING

次の文は、変数 myEnum に Enum.InterestDimension.PURCHASING を割り当てます。

```
InterestDimension myEnum = Enum.InterestDimension.PURCHASING;
```

REAPI PersonalizationIndex インタフェース

PersonalizationIndex は、次のように実装されます。

- PersonalizationIndexType (EnumType を拡張するクラス)
- PersonalizationIndex (インタフェース)

クラス PersonalizationIndexType には、次のメソッドがあります。

```
PersonalizationIndexType getType(String name)
```

```
PersonalizationIndexType getType(int)
```

`PersonalizationIndex` は、返されるリコメンデーションがどれくらい「例外的」であるかを指定します。たとえば、`LOW` は例外的ではないことを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

`PersonalizationIndex` は、次の値を取ります。

- `Enum.PersonalizationIndex.LOW`
- `Enum.PersonalizationIndex.MEDIUM`
- `Enum.PersonalizationIndex.HIGH`

次の文は、変数 `myEnum` に `Enum.PersonalizationIndex.LOW` を割り当てます。

```
PersonalizationIndexType myEnum = Enum.PersonalizationIndex.LOW;
```

REAPI ProfileDataBalance インタフェース

`ProfileDataBalance` は、次のように実装されます。

- `ProfileDataBalanceType` (`EnumType` を拡張するクラス)
- `ProfileDataBalance` (インタフェース)

クラス `ProfileDataBalanceType` には、次のメソッドがあります。

```
ProfileDataBalanceType getType(String name)
```

```
ProfileDataBalanceType getType(int)
```

`ProfileDataBalance` は、リコメンデーションを作成するときに、現在のセッションからデータを取得するか、履歴から取得するか、現在のセッションと履歴の間でデータを均衡化するかを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

`ProfileDataBalance` は、次の値を取ります。

- `Enum.ProfileDataBalance.HISTORY`
- `Enum.ProfileDataBalance.BALANCED`
- `Enum.ProfileDataBalance.CURRENT`

次の文は、変数 `myEnum` に `Enum.ProfileDataBalance.BALANCED` を割り当てます。

```
ProfileDataBalanceType myEnum = Enum.ProfileDataBalance.BALANCED;
```

REAPI ProfileUsage インタフェース

`ProfileUsage` は、次のように実装されます。

- `ProfileUsageType` (`EnumType` を拡張するクラス)

- ProfileUsage (インタフェース)

クラス ProfileUsageType には、次のメソッドがあります。

```
ProfileUsageType getType(String name)
```

```
ProfileUsageType getType(int)
```

ProfileUsage は、リコメンデーション一覧に顧客のプロファイルにあるアイテムを含めるか、除外するかを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

ProfileUsage は、次の値を取ります。

- Enum.ProfileUsage.INCLUDE
- Enum.ProfileUsage.EXCLUDE

次の文は、変数 myEnum に Enum.ProfileUsage.INCLUDE を割り当てます。

```
ProfileUsageType myEnum = Enum.ProfileUsage.INCLUDE;
```

REAPI RecommendationAttribute インタフェース

RecommendationAttribute は、次のように実装されます。

- RecommendationAttributeType (EnumType を拡張するクラス)
- RecommendationAttribute (インタフェース)

クラス RecommendationAttributeType には、次のメソッドがあります。

```
RecommendationAttributeType getType(String name)
```

```
RecommendationAttributeType getType(int)
```

RecommendationAttribute は、返される内容に含める属性を示します。選択できるのは、型、ID および予測です。詳細は、この章の「[ContentItem クラス](#)」および「[RecommendationContent クラス](#)」の説明を参照してください。

RecommendationAttribute は、次の値を取ります。

- Enum.RecommendationAttribute.TYPE
- Enum.RecommendationAttribute.ID
- Enum.RecommendationAttribute.PREDICTION

次の文は、変数 myEnum に Enum.RecommendationAttribute.TYPE を割り当てます。

```
RecommendationAttributeType myEnum = Enum.RecommendationAttribute.TYPE;
```

REAPI Sorting インタフェース

Sorting は、次のように実装されます。

- `SortingType` (EnumType を拡張するクラス)
- `Sorting` (インタフェース)

クラス `SortingType` には、次のメソッドがあります。

```
SortingType getType(String name)

SortingType getType(int)
```

`Sorting` は、ソートを行うかどうか (`NONE` はソートなし)、ソートを行う場合はその方法 (昇順または降順) を示します。詳細は、この章の「[ContentItem クラス](#)」および「[RecommendationContent クラス](#)」の説明を参照してください。

`Sorting` は、次の値を取ります。

- `Enum.Sorting.NONE`
- `Enum.Sorting.DESCEENDING`
- `Enum.Sorting.ASCENDING`

次の文は、変数 `myEnum` に `Enum.Sorting.NONE` を割り当てます。

```
SortingType myEnum = Enum.Sorting.NONE;
```

REAPI User インタフェース

User は、次のように実装されます。

- `UserType` (EnumType を拡張するクラス)
- `User` (インタフェース)

クラス `UserType` には、次のメソッドがあります。

```
UserType getType(String name)

UserType getType(int)
```

`UserType` は、顧客 (呼び出し側 Web サイトの登録ユーザー) または匿名ユーザー (未登録ユーザー) です。詳細は、この章の「[IdentificationData クラス](#)」の説明を参照してください。

`UserType` は、次の値を取ります。

- `Enum.User.CUSTOMER`
- `Enum.User.VISITOR`

次の文は、変数 `myEnum` に `Enum.User.CUSTOMER` を割り当てます。

```
UserType myEnum = Enum.User.CUSTOMER;
```

その他の REAPI サポート・クラス

その他のサポート・クラスは次のとおりです。

- `ContentItem`
- `DataItem`
- `FilteringSettings`
- `IdentificationData`
- `Item`
- `ItemDetailData`
- `Recommendation`
- `RecommendationContent`
- `RecommendationList`
- `TuningSettings`

これらのクラスは、このドキュメントで簡単に説明します。詳細は、OP の Javadoc を参照してください。

ContentItem クラス

このクラスは、リコメンデーション要求が返すオブジェクトに含まれる情報をカプセル化します。これは、コールによって返されるリコメンデーション一覧に含まれる属性を示すだけでなく、一覧を属性の 1 つに基づいてソートするかどうかも指定します。この章で後に説明する [RecommendationContent クラス](#) は、`ContentItem` 型のアイテムの配列です。

「[RecommendationContent クラス](#)」では、複数の順序を指定したときにどのようにソート順が機能するかを説明します。

このクラスには、次のメソッドがあります。

- `getContentAttribute()`
- `getSorting()`

DatalItem クラス

このクラスは、クラス `Item` のサブクラスです。これはアイテムに関するデータをカプセル化します。このオブジェクトは、データ収集メソッド `addItem()` および `addItem()` の引数として使用します。

このクラスには、次の 2 種類のメソッドがあります。

- `DatalItem` のコンストラクタ
- 属性値を返すメソッド
 - `getDataSource()`
 - `getValue()`

FilteringSettings クラス

このクラスは、リコメンデーションを生成するときどのアイテムを含めるかまたは除外するかを指定するのに使用します。

このバージョンの OP では、カテゴリ・フィルタリングにのみ対応しています。

このクラスには、次の 3 種類のメソッドがあります。

- `FilteringSettings` のコンストラクタ
- 属性値を設定するメソッド
 - `setItemFiltering(int taxonomyID)`
 - `setItemFiltering(int taxonomyID, long[] categoryList)`
 - `setItemExclusion(int taxonomyID, long[] categoryList)`
 - `setItemSubTreeFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryExclusion(int taxonomyID, long[] categoryList)`
 - `setCategorySubTreeFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryLevelFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryFiltering(int taxonomyID)`
 - `setCategoryFiltering(int taxonomyID, long[] categoryList)`

- 属性値を返すメソッド
 - `getTaxonomyID()`
 - `getCategoryFiltering()`
 - `getCategoryList()`
 - `getCategoryMembership()`

どのメソッドにもすべてのフィルタリング設定を使用できるわけではありません。特に、次のフィルタリング設定は、抱合せ販売メソッドには使用できません。

- `setCategoryLevelFiltering`
- `setCategorySubtreeFiltering`
- `setCategoryExclusion`
- `setCategoryFiltering(int)`
- `setCategoryFiltering(int, long[])`

IdentificationData クラス

ユーザーまたはセッション（あるいはその両方）を識別します。

このクラスには、次の 2 種類のメソッドがあります。

- IdentificationData インスタンスを作成するメソッド
 - `createSessionful(String appSessionID, UserType userType)`
 - `createSessionless(String appSessionID, UserType userType)`
- 属性値を返すメソッド
 - `getUserID()`
 - `getAppSessionID()`
 - `getUserType()`

呼び出し側 Web アプリケーションは、すべての顧客（登録ユーザー）とすべての匿名ユーザーに `userID` を割り当てる必要があります。顧客の ID は、一意にする必要があります。匿名ユーザーの ID が一意でない場合、OP は特定の匿名ユーザー固有のリコメンデーションを作成できません。そのかわりに、その ID を持った匿名ユーザー全員に同じリコメンデーションが作成されます。

Item クラス

このクラスは、推奨可能なアイテムや、データ収集が可能なアイテムを示すのに使用します。アイテムは、`type` と ID の組合せで一意に示されます。アイテム ID は、特定の `type` で一意であることが必要ですが、異なる `type` で同じ ID を持つことはできます。

このクラスには、次の 3 種類のメソッドがあります。

- Item インスタンスを作成するコンストラクタ
- 属性値を返すメソッド
 - `getType()`
 - `getID()`
- デバッグを補助するメソッド

ItemDetailData クラス

このクラスは、リコメンデーション要求の結果の一部として、OP が内部で作成します。呼び出し側 Web アプリケーションは、属性を調べ、どの属性や値が含まれているかを確認する必要があります。詳細は、この章の「[Recommendation クラス](#)」の説明を参照してください。

次の 3 つのメソッドがあります。

- `getAttribute()`
- `getValue()`
- `toString()`

Recommendation クラス

このクラスは、単一のリコメンデーション・アイテムに関する情報をカプセル化します。アイテムに関する情報は、`attributes` 配列に格納されます。

次の 2 つのメソッドがあります。

- `getAttributes()`
- `toString()`

RecommendationContent クラス

このクラスは、リコメンデーション要求が返す情報の種類を指定します。

このクラスには、次の 2 種類のメソッドがあります。

- ソート方法に応じて、RecommendationContent インスタンスを作成する 2 つのコンストラクタ
- コンテンツ・アイテムを返すメソッド

配列の複数インスタンスをソートする場合、ソート順は配列インデックス順に従います。すなわち、ソートを指定した最初の配列エントリの属性、次にソートを指定した 2 番目のエントリの属性という順で、結果がソートされます。

RecommendationList クラス

リコメンデーション一覧は、特定の InterestDimension のリコメンデーションの集まりです。RecommendationList は、リコメンデーションを返すすべての REAPI メソッドが返すクラスです。

このクラスのメソッドによって、呼び出し側 Web アプリケーションは、インタレスト・ディメンションの種類や返される実際のリコメンデーション数を決定し、個別のリコメンデーションを取得できるようになります。

TuningSettings クラス

このクラスは、リコメンデーションを算出するときに適用する設定を指定します。このクラスのインスタンスは、すべてのリコメンデーション要求に渡されます。

このクラスには、次の 3 種類のメソッドがあります。

- TuningSettings インスタンスを作成するコンストラクタ
- 属性値を設定するメソッド
- 属性値を返すメソッド

次のメソッドは属性値を設定します。

- setDataSourceType()
- setInterestDimension()
- setPersonalizationIndex()
- setProfileDataBalance()
- setProfileUsage()

次のメソッドは属性値を返します。

- `getDataSourceType()`
- `getInterestDimension()`
- `getPersonalizationIndex()`
- `getProfileDataBalance()`
- `getProfileUsage()`

REAPI の使用

この章では、リコメンデーション・エンジン・プロキシの管理、データの収集およびリコメンデーションの取得に使用するメソッドの概要と、一部のメソッドの使用方法について説明します。これらのメソッドのサポート・クラスは、[第3章](#)に記載されています。

これらのメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

クラスやメソッドの使用例は、[第5章](#)と[付録 A](#)の詳細な例を参照してください。

メソッドの戻り値はすべてリアルタイムで出力されます。通常、メソッドはシングル・ユーザーのリコメンデーションを返します。

この章で説明するメソッドは、すべてパブリックです。

リコメンデーション・プロキシ・クラス

リアルタイムのリコメンデーション・プロキシ (REProxyRT) メソッドは、次のように機能ごとに分類できます。

- プロキシの作成と管理 (プロキシ・マネージャと関連メソッド)
- セッション管理 (作成と終了)
- データ収集 (収集、前処理、リコメンデーション・エンジン (RE) 表へのデータの格納)
- リコメンデーション (各種のリコメンデーションの取得)

RE プロキシ・クラスの位置

REProxyRT (とその例外) を使用するには、Java プログラムに次の文を含める必要があります。

```
import oracle.dmt.op.re.reapi.rt.*;

import oracle.dmt.op.re.reexception.*;
```

これらのクラスはすべて、Oracle9iAS がインストールされているシステムに配置されます。

RE プロキシの作成と管理

REProxyManager は、REProxyRT インスタンスのプールを処理します。Oracle9i Application Server などの Web サーバーで複数の REProxyRT インスタンスを使用すると、次のような利点があります。

- フォルト・トレランス (1 つのインスタンスが失敗しても、別のインスタンスを使用可能)
- ロードの分散 (ロードをすべてのプロキシ・インスタンスに分散可能)
- ドメイン依存のリコメンデーション (各プロキシ・インスタンスと特定 RE との関連付け)

複数のプロキシ・インスタンスでは、次の問題が発生することがあります。

- プロキシ・インスタンスが失敗し、アプリケーションが別のインスタンスに移行したときに、収集したデータが失われることがあります。
- セッション中のすべてのトランザクションで、顧客を同じ RE に接続する必要があります。

また、REProxyManager クラスには、リコメンデーション・エンジンのデータ収集をサポートするキャッシング・メカニズムも含まれます。

RE データ収集

REProxyRT クラスには、RE のデータ収集をサポートする DataCollection キャッシュが含まれています。REProxyRT オブジェクトを作成するたびに、プロキシ・オブジェクトのサブコンポーネントとしてキャッシュが作成されます。REAPI コール addItem() および addItem() を使用してデータを収集すると、そのデータはキャッシュ（メモリー）に格納され、定期的に RE スキーマにフラッシュされます。このバッチ保存によって、RE パフォーマンスが向上します。キャッシュは、新しい REProxyRT オブジェクトを作成するときに作成されます。リフレッシュ・レートは、REProxyManager.createProxy() への入力パラメータで定義します。

現在は、クラス DataItem および IdentificationData のアイテムとユーザー ID のみがキャッシュされます。これらは現在のセッション・データとしてキャッシュされます。

REProxyManager クラス

REProxyManager は単独実装です。すなわち、REProxyManager クラスの 1 つのインスタンスのみが特定の JVM インスタンスに作成され、そのクラスが自動的にロードされます。

REProxyManager クラスを使用して、REProxyRT のインスタンスを作成し、管理します。REProxyManager には、スタティックなパブリック・メソッドのみがあります。REProxyManager にはパブリック・コンストラクタがないので、ユーザーは作成できません。REProxyManager は REProxyRT プールを保持し、プロキシ名を使用して個別の REProxyRT オブジェクトを参照します。

次のメソッドは、REProxyRT オブジェクトを管理します。

- createProxy
- getProxy
- destroyAllProxies
- destroyProxy

プロキシ・マネージャの使用例は、[第 5 章](#)と[付録 A](#)の詳細な例を参照してください。

プロキシ・メソッド

リコメンデーション要求は、すべてクラス REProxyRT によって処理されます。セッション対応型アプリケーションのセッション処理、顧客プロファイル・データの収集およびリコメンデーションの取得など、リコメンデーションの関連タスクを実行する前に、createProxy または getProxy を使用して、REProxyRT オブジェクトを取得します。

RE プロキシ・セッション管理

次のメソッドはセッションを管理します。

- `createCustomerSession`
- `createVisitorSession`
- `closeSession`

RE プロキシ・データの収集と管理

次のメソッドは、データの収集、前処理を行い、データを RE 表に格納します。収集されたデータは、適切な構成パラメータを設定すると永続化できます。

- `addItem`
- `addItems`
- `removeItem`
- `removeItems`

RE プロキシの顧客登録

次のメソッドを使用すると、匿名ユーザーを顧客（登録ユーザー）に変更できます。

- `setVisitorToCustomer`

このメソッドは、セッション対応型アプリケーションにも、非セッション対応型アプリケーションにも使用できます。

RE プロキシのリコメンデーション

次のメソッドは、リコメンデーションを取得して管理します。

- `rateItem`
- `rateItems`
- `recommendTopItems`
- `recommendBottomItems`
- `recommendFromHotPicks`
- `recommendCrossSellForItem`
- `recommendCrossSellForItems`
- `crossSellForItemFromHotPicks`

- `crossSellForItemsFromHotPicks`
- `selectFromHotPicks`

呼び出し側 Web アプリケーションでは、返されたリコメンデーションをエンド・ユーザーに伝達します。また、呼び出し側 Web アプリケーションは、ユーザーに渡すリコメンデーションを決定する必要があります。たとえば、Web アプリケーションはそのアイテムの在庫を確認してからアイテムを推奨する場合があります。

リコメンデーションを返すメソッドがアイテム一覧を返すとは限りません。`FilteringSettings.CategoryMembership` を次のいずれかの値に設定した場合、リコメンデーション・メソッド (`recommendTopItems` など) は、カテゴリを返します。

- `Enum.CategoryMembership.EXCLUDE_CATEGORIES`
- `Enum.CategoryMembership.INCLUDE_CATEGORIES`
- `Enum.CategoryMembership.SUBTREE_CATEGORIES`
- `Enum.CategoryMembership.ALL_CATEGORIES`

カテゴリは分類 (TAXONOMY) の構成要素です。分類は、マイニング・テーブル・リポジトリ (MTR) の次の表に定義します。

- `MTR_TAXONOMY`
- `MTR_TAXONOMY_CATEGORY`
- `MTR_TAXONOMY_CATEGORY_ITEM`
- `MTR_CATERGORY`

OP アプリケーションの設計には、適切な分類が不可欠です。分類の作成方法の詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

OP のレーティング

OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

リコメンデーションの戻り値の意味

`ItemDetailData.attribute` が `Enum.RecommendationAttribute.PREDICTION` と等しい場合、リコメンデーション・インスタンスに返された値の意味は、次のように `interestDimension` の値によって異なります。

- `InterestDimension.RATING` の場合、アイテムの予測されるレーティング値が返されます。

- InterestDimension.PURCHASING や InterestDimension.NAVIGATION の場合、ランキングが返されます。見込みの高さに基づいて、最も見込みが高いアイテムに 1 の値が割り当てられ、その他のアイテムは予想されるランクに基づいて整数値が割り当てられます。

規則とリコメンデーション

OP は、RE に格納された規則表を使用して、リコメンデーション・メソッドが要求したリコメンデーションを生成します。規則表は、パッケージのビルド時に作成され、パッケージの配布時に RE に格納されます。使用する規則表は、実行した REAPI コールによって異なります。通常は、規則の先行条件をキャッシュのデータ（履歴および現在のセッション・データ）と一致させ、各種の結果の確率を算出します。これらの項目は確率の順番に並べられ、numberOfItems（API 引数）項目が返されます。

RE プロキシ・メソッドの使用方法

これらのメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある OP Javadoc を参照してください。この項では、メソッドの概要とその使用方法を説明します。

セッションの作成

createCustomerSession と createVisitorSession の場合、呼び出し側 Web アプリケーションは、現在のアクティブ・セッション間で一意のセッション ID を提供する必要があります。いずれかのメソッドを RE で現在アクティブなセッション ID で実行すると、例外が返されます。ただし、RE でセッション ID がアクティブでなければ、そのセッション ID を再利用できます。appSessionID は、OP によって MTR と同期化されます（データ同期化の詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください）。OP は、customerID と appSessionID が有効な値であるかどうかを判断できないため、呼び出し側 Web アプリケーションはこれらの値が有効であることを確認する必要があります。

データ収集

データを収集するには、addItem または addItemS を使用します。ローカル・キャッシュからデータを削除するには、removeItem または removeItems を使用します。

アイテムの追加

addItem と addItemS の場合はどちらも、まずアイテムがローカルでキャッシュされ、同期化によって RE に書き込まれます。書き込みの頻度は、OP のインストール時に構成パラメータで指定します。データ同期化の時間隔には、Web アプリケーションの要件に十分対応できる頻度を設定することが重要です。データ同期化の詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

アプリケーションが複数のアイテムを同時に追加する必要がある場合、複数の addItem コールか、1つの addItem コールのいずれかを使用できます。addItem を使用する場合、アプリケーションはコールを実行するまで、アイテムの詳細を保持する必要があります。すなわち、アプリケーションは状態を維持する必要があります。複数の addItem コールを発行するほうが簡単な場合もあります。

addItem と addItem は非同期なので、呼び出し側アプリケーションはコールがデータをデータベースに保存するまで待つ必要はありません。

RE に収集されたデータは、自動的に MTR に書き込まれます。

アイテムの削除

removeitem と removeItems は、MTR（永続ストレージ）に書き込まれていないアイテムを削除します。データが MTR に書き込まれると、これらのメソッドを使用してデータを削除することはできません。

プロキシの作成

createProxy には、キャッシュ・サイズと時間隔を指定する必要があります。この項では、これらの値を決定する方法を説明します。

適切なキャッシュ・サイズと対応する最適な時間隔を決定するには、テストを行います。

キャッシュ・サイズと時間隔を構成する適切な方法は、次のとおりです。

1. キャッシュ・サイズを約 3027 KB に設定します。
2. 予想されるデータ収集速度に基づいて時間隔を設定します。
3. テストを行います。
4. アーカイブ間隔を調整します。

キャッシュ・サイズ

キャッシュ・サイズとは、リコメンデーション・エンジンが使用する KB 単位のキャッシュの大きさです。

キャッシュ・サイズを決定する場合、いくつかの要素を考慮します。

1. システム・リソース: キャッシュはメモリー領域を使用するため、必要なメモリーが十分あることを確認する必要があります。
2. アーカイブ間隔: 時間隔が長くなると、キャッシュ・サイズも大きくなります。
3. 最大 VARRAY サイズ: アーカイブを実行する PL/SQL プロシージャは VARRAY を使用し、最大サイズは現在 5000 に設定されています。アーカイブは 5000 項目以上を処理できますが、パフォーマンスはかなり低下します。したがって、キャッシュ・バッファは 5000 を超えないようにしてください。キャッシュに格納される各データ項目は最大で約 340 バイトを必要とするため、最大 VARRAY サイズは 3.3 MB となります（キャッ

シュには2つのバッファがあるので、実際のキャッシュ・バッファ・サイズはその半分です)。

4. データ収集速度: 最も重要な要素です。データ収集速度が1秒あたり100項目以下で、アーカイブ間隔が20秒である場合、妥当なキャッシュ・サイズは $100 \times 340 \times 1.5 \times 20$ 、すなわち約2000KBとなります(この計算では、データが欠落しないよう安全係数1.5を想定しています)。

時間隔

時間隔は、収集データをアーカイブする(メモリーからREスキーマにフラッシュする)頻度を決定します。この設定では、次のような要素を考慮します。

1. データ収集量と速度: データ収集の頻度が高くなると、収集されるデータの量が大きくなるので、時間隔は短くする必要があります。
2. キャッシュ・サイズ: キャッシュが小さいほど、時間隔が短くなります。
3. 現在のセッション・データの使用: 現在のセッション・データを使用してリコメンデーションの精度を上げる場合、データをキャッシュに長時間保持しないでください。データ収集の量や速度が問題にならない場合は、時間隔は10～30秒が適当です。

抱合せ販売メソッド

この項の説明は、`crossSellForItemFromHotPicks`、`crossSellForItemsFromHotPicks`、`recommendCrossSellForItem` および `recommendCrossSellForItems` に適用します。

インタレスト・ディメンションは、入力項目のデータ・ソース・タイプと同じにする必要があります。

データ・ソース・タイプは、NAVIGATION または PURCHASING にする必要があります。その他のタイプはサポートされていません。

これらのメソッドには、次のフィルタリング設定を使用できません。

- `setCategoryLevelFiltering`
- `setCategorySubtreeFiltering`
- `setCategoryExclusion`
- `setCategoryFiltering(int)`
- `setCategoryFiltering(int, long[])`

プロキシの破棄

プロキシ・オブジェクトの破棄は、特に慎重に行ってください。REProxyRT オブジェクトは多くのクライアントが共有するため、プロキシの破棄がリコメンデーション・サービスを中断することがあります。プロキシ破棄メソッドは、特に慎重に使用する必要があります。Web アプリケーションの場合、REProxyRT オブジェクトは、サーバー・サービスの一部として処理する必要があるため、絶対に必要な場合でないかぎり、破棄しないでください。これらのオブジェクトは、他のサーバー・コンポーネントと同じように、メンテナンスのためにサーバーをシャットダウンするか、オフラインにするときにのみ破棄する必要があります。

`destroyProxy` を使用してプール内の特定プロキシを破棄することも、`destroyAllProxies` を使用してプール内の全プロキシを破棄することもできます。

REAPI の例と使用方法

この章では、REAPI の使用例を示します。必要に応じて、コード・スニペット (snippet) を提供することも、OP を使用して特定のタスクを実行する方法を説明することもあります。

REAPI Demo

OP には REAPI Demo が含まれています。これは、多くの REAPI メソッドの使用方法を説明するサンプル・プログラムです。このサンプル・プログラムを使用して、REAPI コールについて理解することも、OP が正しくインストールされていることを確認することもできます。

OP をインストールしたら、Netscape か Internet Explorer で次の URL を開いて、REAPI Demo を起動します。

`http://<hostname>:<port>/OP/redemo`

OP REAPI Demo のソース・コードを表示するには、「ソース・コード表示」をクリックします。

デモのインストールおよび実行方法の詳細は、『Oracle9iAS Personalization ユーザーズ・ガイド』を参照してください。

REAPI の基本的な使用方法

[第4章](#)に記載された REProxy メソッドを使用して、Web サイトを設定できます。REAPI コールを使用するには、次の手順を実行する必要があります。

1. REProxy オブジェクトを取得します。
2. 必要に応じて、REAPI コールでプロキシ・インスタンスを使用します。プログラムが従うアウトラインは、Web アプリケーションがセッション対応型か、非セッション対応型かによって異なります。
3. 使用しているプログラムで不要になった場合、プロキシ・オブジェクトを破棄します。

REProxy オブジェクトの作成

この項では、基本的な REProxy の使用方を説明します。REProxy の詳細とその他の使用方は、この章の「[JVM と REProxyManager との相互作用](#)」および「[REProxy の複数インスタンスの使用](#)」を参照してください。

次のコード部分は、オブジェクト `proxy` を作成します。このオブジェクトを使用して、REAPI コールを実行します。RE スキーマのユーザー名とパスワードを指定する必要があります。

```
final String proxyName = "RE1";
final String dbURL = "jdbc.oracle.thin:@DBServer.myshop.com:1521:DB1";
final String user = "myself";
final String passWd = "secret";
final int cacheSize = 2048;           // 2 mbytes
final int interval = 10000;          // 10 seconds
REProxy proxy;
...

try {
    proxy = REProxyManager.createProxy(proxyName,
                                       dbURL,
                                       user,
                                       passWd,
                                       cacheSize,
                                       interval);
    ...
} catch (Exception e) {
    // exception handling here
}
```

プロキシの使用

REProxy オブジェクトを作成し、そのインスタンスを取得したら、プロキシを使用して REAPI コールを指定します。次に例を示します。

```
proxy.closeSession();
```

コールのシーケンスは、アプリケーションがセッション対応型か非セッション対応型かによって異なります。詳細は、この章の「[セッション対応型の Web アプリケーションの概要](#)」または「[非セッション対応型の Web アプリケーションの概要](#)」を参照してください。

プロキシの破棄

プロキシ・オブジェクトの破棄は、特に慎重に行ってください。REProxyRT オブジェクトは多くのクライアントが共有するため、プロキシの破棄がリコメンデーション・サービスを中断することがあります。プロキシ破棄メソッドは、特に慎重に使用する必要があります。Web アプリケーションの場合、REProxyRT オブジェクトは、サーバー・サービスの一部として処理する必要があるため、絶対に必要な場合でないかぎり、破棄しないでください。これらのオブジェクトは、他のサーバー・コンポーネントと同じように、メンテナンスのためにサーバーをシャットダウンするか、オフラインにするときにのみ破棄する必要があります。

セッション対応型の Web アプリケーションの概要

セッション対応型の Web アプリケーション（各顧客にセッションを開始するアプリケーション）の必須手順を次に示します。

1. この章の「[REProxy オブジェクトの作成](#)」ですでに説明したように、REProxy オブジェクトを作成します。RE スキーマのユーザー名とパスワードが必要です。プロキシがすでに存在する場合、getProxy を呼び出します。

2. 顧客セッションか、匿名ユーザー・セッションを作成します。

```
proxy.createCustomerSession(userID, appSessionID); //customer session
proxy.createVisitorSession(userID, appSessionID); //visitor session
```

3. 識別データを取得します。

```
idData = IdentificationData.createSessionful(appSessionID);
```

4. REAPI メソッドを呼び出します。次に例を示します。

```
/*Set input parameters. */
int nRec=10;
TuningSettings tune = new TuningSettings(Enum.DataSourec.NAVIGATION,
                                         Enum.InterestDimension.NAVIGATION,
                                         Enum.PersonalizationIndex.HIGH,
                                         Enum.ProfileDataBalance.BALANCED,
                                         Enum.ProfileUsage.EXCLUDE);

long [] catList = {1, 2, 3, 4};
FilteringSettings filters = new FilteringSettings();
filters.setItemFiltering(1, catList);
RecommendationContent rContent = new RecommendationContent (
                                         Enum.Sorting.ASCENDING);

/*Get a recommendation. */
try {
    RecommendationList rList = proxy.recommendTopItems(idData,
                                                       nRec, tune, filters, rContent);
}
/* Parse the results and pass recommendations to the user*/
```

5. 必要に応じて、他の REAPI コールを実行します。

6. セッションを終了します。

```
proxy.closeSession();
```

非セッション対応型の Web アプリケーションの概要

非セッション対応型の Web アプリケーション（各顧客にセッションを開始しないアプリケーション）の必須手順を次に示します。非セッション対応型アプリケーションは、タイムアウト時に終了します。

1. この章の「[REProxy オブジェクトの作成](#)」ですでに説明したように、REProxy オブジェクトを作成します。RE スキーマのユーザー名とパスワードが必要です。プロキシがすでに存在する場合、getProxy を呼び出します。
2. 識別データを取得します。

```
idData = IdentificationData.createSessionless(customerID);
```

3. REAPI メソッドを呼び出します。次に例を示します。

```
/*Set input parameters.*/
int nRec=10;
TuningSettings tune = new TuningSettings(Enum.DataSourec.NAVIGATION,
    Enum.InterestDimension.NAVIGATION,
    Enum.PersonalizationIndex.HIGH,
    Enum.ProfileDataBalance.BALANCED,
    Enum.ProfileUsage.EXCLUDE);

long [] catList = {1, 2, 3, 4};
FilteringSettings filters = new FilteringSettings();
filters.setItemFiltering(1, catList);
RecommendationContent rContent = new RecommendationContent (
    Enum.Sortinh.ASCENDING);

/*Get a recommendation. */
try {
    RecommendationList rList = proxy.recommendTopItems(idData,
        nRec, tune, filters, rContent);
    /* Parse the results and pass recommendations to the user*/
```

4. 必要に応じて、他の REAPI コールを実行します。

JVM と REProxyManager との相互作用

REProxyManager は単独実装です。すなわち、REProxyManager クラスの 1 つのインスタンスのみが特定の JVM インスタンスに作成され、そのクラスが自動的に JVM インスタンスにロードされます。この動作は、プログラムの動作に関連があります。アプリケーションがスタンドアロン Java プログラムか、Java サーバーサイド・モジュールかに応じて、この動作は異なります。これはプロキシ管理にも当てはまりますが、この場合は異なる使用方法モデルを考慮する必要があります。

スタンドアロン Java アプリケーション

次のようなコマンドをコマンドラインから実行する REAPI コールを使用して、スタンドアロン Java アプリケーションを作成するとします。

```
java myapplication.class
```

このようなアプリケーションには次の特性があります。

- 個別の JVM インスタンスで実行されます。
- REProxyManager インスタンスは、自動的に JVM インスタンスにロードされます。
- アプリケーションが実行を終了すると、JVM インスタンスはなくなります。

プログラムが終了する前にプロキシを破棄しないと、REProxy オブジェクトがメモリに残ります。オブジェクトを作成した JVM インスタンスがすでに存在しないので、このオブジェクトにはアクセスできません。

メモリ・リークを避けるには、プログラムが終了する前にプロキシを破棄する必要があります。

Java サーバーサイド・モジュール

サーブレットや JavaServer Pages (JSP) などの Java サーバーサイド・モジュールから REAPI を呼び出した場合、モジュールが存在する Oracle9i Application Server に REProxyManager クラスがロードされます。

Java モジュールを所有し使用する Web アプリケーションは通常、サーバーの起動時に開始され、サーバーをシャットダウンするまでは終了しません。この場合、Web アプリケーションの開始時か、最初の RE 要求の処理時にプロキシを作成できますが、プロキシの破棄を考慮する必要はありません。Web アプリケーションが起動し、実行している間は、リコメンデーション要求の処理にそのプロキシが使用されます。

プロキシの作成には時間がかかります (Sun E450 サーバーで数百ミリ秒)。このため、システム管理者がメンテナンスのために Web アプリケーションを終了する場合などにサーバーをシャットダウンするまで、プロキシを破棄しないほうが効率的です。

プロキシを詳細に管理すること、すなわち、使用していないプロキシ・オブジェクトを削除することを選択した場合、破棄メソッドを呼び出して削除を実行できます。ただし、どちらの破棄メソッドも強制的にプロキシを削除して、そのプロキシを他のプロセスが使用しているかどうかは確認しないため、破棄メソッドを使用するときは特に注意が必要です。

REProxy の複数インスタンスの使用

REProxyManager は、1 つ以上のプロキシのプールを管理します。この項では、複数のプロキシを使用するいくつかの方法を説明します。

- 初期化のフェイル・セーフ
- 中断されない REAPI サーバーの実現
- ロード・バランシング

初期化のフェイル・セーフ

次のコード部分は、2 つの RE を使用して使用率の障害を防ぐ方法を示します。このコードでは、標準のリコメンデーション・サービスのスキーマが「RE」という名前で、「RE」が失敗した場合は、「RE_BACKUP」という名前のバックアップ RE スキーマを使用すると想定しています。

```
REProxy initProxy(...)
{
    REProxy proxy;

    // initialization
    try {
        if ((proxy = REProxyManager.getProxy("RE")) == null)
            proxy = REProxyManager.createProxy("RE",
                                                dbURL,
                                                username,
                                                passWd,
                                                cacheSize,
                                                interval);
    } catch (REProxyInitException rie) {
        proxy = REProxyManager.createProxy("RE_BACKUP",
                                            dbURL1,
                                            username1,
                                            passWd1,
                                            cacheSize,
                                            interval);
    }
    return proxy;
}
```

中断されない REAPI サービス

次のコード部分は、通常の RE サーバーで障害が発生したときに、リコメンデーション・サービスが失敗しないよう保証する方法を示します。この場合、コードはクラス `NeverFail` を実装します。

```
class NeverFail() {
    REProxy re1;
    REProxy re2;

    void initProxies() {
        try {
            if ((re1 = REProxyManager.getProxy("RE1")) == null)
                String dbURL1="jdbc:oracle:thin:@db1.mycorp.com:1521:orc1";
                re1 = REProxyManager.createProxy("RE1",
                                                    dbURL1,
                                                    "user1",
                                                    "pw1",
                                                    2048,
                                                    10000);
            if ((re2 = REProxyManager.getProxy("RE2")) == null)
                String dbURL2="jdbc:oracle:thin:@db2.mycorp.com:1521:orc2";
                re2 = REProxyManager.createProxy("RE2",
                                                    dbURL2,
                                                    "user2",
                                                    "pw2",
                                                    2048,
                                                    10000);
        } catch (REProxyInitException rie) {
            // exception handling
        }
    }

    RecommendationList getRecommendation() {
        RecommendationList rList;

        // initialize input
        ....
        try {
            rList = re1.recommendTopItems(...);
        } catch (Exception e) {
            rList = re2.recommendTopItems(...);
            return rList;
        }
        return rList;
    }
}
```

ロード・バランシング

次のコード部分は、すべての顧客が同じ RE で処理されないようロード・バランシングを行う簡単な方法を示します。この例では、奇数 ID の顧客が RE1 で処理され、偶数 ID の顧客が別の RE2 を使用して処理されると想定しています。このため、次のように、まず 2 つの異なるプロキシ RE1 と RE2 を作成し、続いて `getRecommendation()` を呼び出します。

```
RecommendationList getRecommendation() {
    RecommendationList rList;

    // initialize input
    ....
    try {
        if ((idData.getUserID() % 2) == 1)
            rList = re1.recommendTopItems(...);
        else
            rList = re2.recommendTopItems(...);
    } catch (Exception e) {
        // exception handling
        .....
    }
    return rList;
}
```

個別のリコメンデーションの抽出

配列から個別のリコメンデーションを抽出するのではなく、`Recommendation` クラスの `getAttributes` メソッドを使用します。

複数通貨の処理

OP は、通貨データを人口統計表（たとえば、個人の所得）に数字で格納します。すなわち、OP はどのようなラベルも格納しません。10 ドル（米国）も、10 ポンド（英国）も「10」として格納されます。

通貨データが正しく解釈されているかどうかを確認する方法はいくつかあり、アプリケーションに選択する対処法は、アプリケーションが通貨データをどのように使用するかによって異なります。

- 顧客の人口統計データに国別コードを含めます。
この対処法により、国を考慮できますが、値と国を密接に関連付けません。
- すべての通貨を、ユーロや US ドルなどの共通通貨に変換します。

この対処法では、個別の通貨値を有意義に比較できますが（10 ポンドは 10 US ドルより価値が高い）、米国における 30,000 US ドルの給料とブラジルにおける同じ 30,000 US ドルの給料という、データ間の違いは保持できません。このような情報は、たとえば、米国とブラジルの両方で高所得者にアイテムを推奨する場合に必要です。というのは高所得者の給料を US ドルで表した場合、国によってその額にはかなりの開きがあるからです。

この方法では、OP がリコメンデーションを作成する前に、OP 外でデータを前処理する必要があります。

- 平均に基づいて通貨値のビンニングを行い、国間で比較できる相対値を取得します。

この対処法では、たとえば、ある国の高所得者を求めることができますが、適切にビン範囲を決定し、管理する必要があります。

この方法では、OP がリコメンデーションを作成する前に、OP 外でデータを前処理する必要があります。

リコメンデーション・エンジンの使用方法

Oracle9iAS Personalization では、1 つ以上のリコメンデーション・エンジン・ファームに 1 つ以上のリコメンデーション・エンジン（RE）が必要です。一般に、複数の RE を使用して十分なリコメンデーション・パフォーマンスを得る必要があります。多くのアプリケーションでは、異なるマシンと異なるデータベース・インスタンスで複数の RE を使用します。このような対処法のコードを作成する方法の例は、この章の「[ロード・バランシング](#)」を参照してください。

一般に、アプリケーションではこれらの RE が同じ RE ファームに属します。物理システムに複数のプロセッサがあり、データベースがプロセッサを有効に利用できる場合、ユーザー数に対して必要な RE 数を場合によっては 1 つにまで減らせます。詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

アプリケーションで複数の RE を利用する場合、どの RE を使用するかを決定する必要があります。次に、可能な対処法を示します。

1. Web サイトのあるユーザー（匿名ユーザーまたは顧客）を必ず、同じ Oracle9iAS/OC4J インスタンスで処理し、その Oracle9iAS/OC4J インスタンスが常に 1 つの RE を使用するよう構成します。アプリケーションは、ユーザーに対応する Oracle9iAS/OC4J インスタンスにルーティングし、Oracle9iAS/OC4J インスタンスが特定の RE に接続されるよう構成する必要があります。REProxy クラスは、構成引数を使用して、どの RE に接続するかを指定します。アプリケーションは、Oracle9iAS/OC4J.properties ファイルからこれらの構成引数を取得するか、Web アプリケーションに明示的にコーディングするか、その他の方法で、構成引数の取得方法を決定する必要があります。
2. Oracle9iAS/OC4J インスタンスであらゆる顧客を処理できるようにします。これには、顧客を特定の RE にハッシュする必要があります。データは RE でユーザーのセッション

ンにキャッシュされるので、少なくとも1つのセッション内では、同じ顧客が同じ RE にルーティングされることが重要です。

3. アプリケーションにフェイルオーバー・メカニズムを備えて、ある顧客のプライマリ RE につながらなかったときに、別の RE に接続できるようにします。これは、前述の対処法 1 または 2 に追加して適用できます。この場合、アプリケーションは、プライマリ RE とバックアップ RE（または複数のバックアップ RE）を指定し、RE 間で切替えを行う論理を制御します。同じユーザー・セッションを必ずしも同じ RE にルーティングできなくても、ある程度のリコメンデーション機能は維持されます。ただし、適切な堅牢性を保持する環境では、必ずしもこのような対処法を実装する必要はありません。

人口統計データの使用

MTR_CUSTOMER 表のスキーマは、サイト・データベースの任意の列にマッピングできる 50 の汎用属性で構成されています。異なるデータ型をすべてサポートするため、属性はすべて VARCHAR 型です。したがって、マッピングされた列を文字列に変換する必要があります。OP のこのリリースでは、このようにマッピングされた列は、カテゴリまたは数値のみとして処理されます。マッピングされた列が DATA 属性である場合、TO_NUMBER 関数を使用して変換する必要があります。変換した値は、ビン範囲を指定すると、他の属性と同じようにビンングを行うことができます。

人口統計データのビンングがあります。ビンングを行う属性は boolean 型です。OP では、ビン番号は内部で整数として表されますが、実際の値は呼び出し側アプリケーションに返されます。つまり、Web アプリケーションは実数値を渡し、実数値を受け取ります。

時間ベースのアイテムの処理

航空券のようなアイテムの場合、アイテムを購入した時間によって価格が異なります。たとえば、ボストンからロンドンへの航空券には、搭乗日の 6 か月前に購入した場合の価格と、搭乗日の 2 日前に購入した場合の価格があります。

購入した時間には関係なく、Web アプリケーションが同じ搭乗便のすべての航空券に同じアイテム ID を割り当てる場合、そのアイテムは「6 か月前」や「2 日前」などの異なるアイテム・タイプを持つ必要があります。あるいは、アプリケーションがアイテムに分類を定義し、カテゴリにリコメンデーションを取得することもできます。

アプリケーションが、異なる時間に購入した同じ搭乗便の航空券に別々のアイテム ID を割り当てる場合（6 か月前に購入した航空券と 2 日前に購入した同じ搭乗便の航空券で ID が異なる場合）、すべての航空券が同じアイテム・タイプを取れます。この場合、リコメンデーション・アイテム ID が適切ではないことがあるため、アプリケーションが分類を定義し、カテゴリのリコメンデーションを要求する必要があります。

第II部

リコメンデーション・エンジン・バッチ API

第II部では、OP (Oracle9iAS Personalization) RE Batch API (リコメンデーション・エンジン・バッチ Application Program Interface (API)) を使用して、Java で記述された Web アプリケーションがバルク・モードで Oracle9iAS Personalization スタイルのリコメンデーションを要求できることを説明します。

第II部は、次の章で構成されます。

- [第6章「RE Batch API の概要」](#)
- [第7章「RE Batch API サポート・クラス」](#)
- [第8章「リコメンデーション・エンジン・バッチ・プロキシの使用」](#)
- [第9章「REProxyBatch API の例と使用方法」](#)

RE Batch API の使用に関する詳細な例は、[付録B](#)を参照してください。

RE Batch API クラスとメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。バッチのメソッドやクラスの多くは、REAPI メソッドおよび REAPI クラスです。

RE Batch API の概要

OP (Oracle9iAS Personalization) RE Batch API (リコメンデーション・エンジン・バッチ Application Program Interface (API)) を使用すると、Java で記述されたアプリケーションは、バルク・モードで Oracle9iAS Personalization スタイルのリコメンデーションを要求できます。

RE Batch API は、拡張性に優れ、API 関数を少なくし、均一性を保ち、引数の数が最小限になるよう設計されています。

第 9 章では、RE Batch API による共通タスクの実行例を示します。

付録 B では、RE Batch API の詳細な使用例を示します。

RE Batch API クラスとメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

注意： RE Batch API は、Oracle9iAS がインストールされているシステムにインストールされます。

RE Batch API 前提条件

RE Batch API メソッドを使用する前に、OP をインストールし、適切な表を作成して、移入する必要があります。データベース表は、OP スキーマに変換する必要があります。OP MTR に顧客プロフィールを移入することが重要です。また、リコメンデーションが必要な顧客 ID を含む表やビューを作成することも必要です。

1 つ以上の分類を使用する場合、適切に指定する必要があります。

1 つ以上の OP パッケージを作成し、配布する必要があります。これには、OP 管理 UI を使します。パッケージを作成し、配布する方法の例は、『Oracle9iAS Personalization ユーザーズ・ガイド』を参照してください。

注意： RE バッチ・コールの実行中にはパッケージを配布しないでください。配布の実行中には RE バッチ・コールを開始しないでください。このようなアクティビティを行うと例外が発生します。

RE Batch API の定義と概念

この項では、RE Batch API を構成する各種のメソッドと、API を説明する上での概念と用語について説明します。

RE Batch API エンド・ユーザー（顧客）

エンド・ユーザー（OP のリコメンデーションを利用する Web サイト・ユーザー）は、顧客と匿名ユーザーという 2 つのグループに分類されます。顧客とは登録ユーザーで、Web アプリケーションにより割り当てられた一意の顧客 ID で識別できます。RE Batch API は、顧客にのみリコメンデーションを作成します。

RE Batch API によるリコメンデーション

リコメンデーションは、履歴データに基づきます。履歴データはデータベースに格納され、顧客プロフィールをロードするときに検索されます。

RE Batch API の使用

RE バッチ・プログラムを実行する前に、次の手順を行う必要があります。

- OP 環境を設定します（RE の作成、OP パッケージの作成と配布）。
- RE バッチ・メソッドで使用する表を作成します。

RE Batch API 環境の設定

RE Batch API メソッドを実行する前に、次の事項を確認する必要があります。

- マイニング・テーブル・リポジトリ（MTR）で適切なフォーマットの顧客プロフィール・データが利用できる。
- 少なくとも 1 つのリコメンデーション・エンジン（RE）を含むリコメンデーション・エンジン・ファームがある。
- パッケージが正しくビルドされ、リコメンデーション・エンジン・ファームに配布されている。

これらの手順を実行する方法は、『Oracle9iAS Personalization 管理者ガイド』と OP の管理 UI のオンライン・ヘルプを参照してください。

顧客プロフィール・データ

顧客プロフィール・データは MTR に存在します。

RE へのパッケージ配布

OP の管理インタフェースで作成した既存の配布パッケージがない場合、リコメンデーションは取得できません。配布前に、パッケージをビルドする必要があります。なんらかのデータがないとパッケージをビルドできません。データは、Web アプリケーションで収集し、Oracle データベースに格納された既存のデータから変換されます。

OP アプリケーションを設計する場合、複数の RE が必要か、必要な場合はどのように使用するかを決定する必要があります。大量のリコメンデーションに使用する RE は、他の目的には使用しないでください。設計上の考慮事項の説明は、[第 9 章の「リコメンデーション・エンジンの使用方法」](#)を参照してください。

注意： バッチ・プログラムの実行中にパッケージを配布すると、配布は失敗します。

たとえば、リコメンデーションで所得レベル（給料）を考慮する場合、リコメンデーション・アイテムはユーザーが購入可能なものになります。リコメンデーション・アイテムが複数通貨の価格を持つ場合（たとえば、アイテムを日本とインドで販売する場合）は、[第 9 章の「複数通貨の処理」](#)を参照してください。

RE Batch API の使用例

OP には、様々な RE Batch API メソッドの使用方法を説明する Java プログラムが含まれています。このプログラムは、[付録 B](#)に記載されています。また、[第 9 章](#)にも一般的なタスクの実行例が示されています。

REBatchProxy オブジェクトの作成

RE Batch API メソッドを使用する前に、1 つ以上の REBatchProxy オブジェクトを作成する必要があります。詳細は、[第 9 章](#)を参照してください。オブジェクトは、特定のデータベースやスキーマとの JDBC 接続を確立します。この接続は、明示的に破棄するまで存続します。

RE Batch API オブジェクトのインスタンスの作成

API を使用するには、API メソッド・シグネチャで使用するオブジェクトのインスタンスを作成する必要があります。このインスタンスを作成するには、[第 8 章](#)にある RE Batch API サポート・クラスを使用します。たとえば、フィルタリング設定やチューニング・セッションを作成する必要があります。これらの例は、[第 9 章](#)を参照してください。

RE Batch API のデータ変換

OP は、過去のユーザー行動を示すデータに基づいてリコメンデーションを生成します。

Oracle 表に格納されたユーザー・データは、リコメンデーションの生成に使用する前に、変換してマイニング・テーブル・リポジトリ (MTR) に格納する必要があります。

RE Batch API の顧客プロファイルの管理

OP は、マイニング・テーブル・リポジトリ (MTR) に顧客プロファイルを格納します。使用するプロファイルは、リコメンデーション要求を行う前に、RE にロードする必要があります。次のメソッドは、RE の顧客プロファイルのロードとアンロードを管理します。

- `loadCustomerProfiles()`
- `purgeCustomerProfiles()`

顧客プロファイルをロードする前に、ロードするプロファイルを識別する顧客 ID 一覧、すなわち、リコメンデーションが必要な顧客 ID の一覧を含む表やビューを作成する必要があります。

RE API バッチ・リコメンデーションの取得

リコメンデーションを取得するため、アプリケーションは次のリコメンデーション・メソッドの 1 つを呼び出します。

- `crossSellForItem()`
- `rateItem()`
- `recommendTopItems()`

これらのメソッドは、顧客（登録ユーザー）のリコメンデーションを取得するのに使用します。

OP のレーティング

OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

リコメンデーションの作成

OP は、RE に格納された規則表を使用して、前述のメソッドで要求したリコメンデーションを計算します。使用する規則表は、RE Batch API メソッドによって異なります。通常は、規則の先行条件を履歴データと一致させ、各種の結果の確率を算出します。これらの項目は確率の順番に並べられ、numberOfItems (API 引数) 項目が返されます。リコメンデーションは、データベース表に書き込まれます。

RE データベースに十分なメモリーがあれば、RE は、一番最近の規則だけでなく、MTR から RE に配布された特定パッケージに対応する規則をすべてキャッシュします。

スコアリング: スコアリングには、利用可能な履歴データをすべて使用します。

OP マイニング・テーブル・リポジトリ (MTR) には、細目および集計形式で格納したレーティング履歴、トランザクション・データおよびナビゲーション・データが含まれます。また、MTR には人口統計データも含まれます。顧客をスコアリングするとき、RE は人口統計データと他のデータ・ソース・タイプの集計バージョンを検索します。

RE バッチ・リコメンデーションの作成

リコメンデーションを作成する RE Batch API メソッドは、リコメンデーションをデータベース表に書き込みます。この出力に使用するスキーマは、使用するメソッドによって異なります。リコメンデーションは、様々な方法で抽出できます。たとえば、適切な SQL クエリーを使用して、ユーザーにどのリコメンデーションを渡すかを決定します。

REBatchProxy オブジェクトの削除

アプリケーションを終了する前に、不要なプロキシ・オブジェクトを破棄する必要があります。

RE Batch API サポート・クラス

この章では、REProxyBatch クラスのサポート・クラスについて説明します。これらのクラスは、[第 8 章](#)にあるメソッドで使用するオブジェクトのインスタンスを作成するときに使用します。これらのクラスの多くに 1 つのインスタンスを作成し、その 1 つのインスタンスを複数コールの引数として使用できます。

注意： Location を除いて、これらのサポート・クラスは REAPI で使用するサポート・クラスと同じです（すべての REAPI クラスを RE Batch API で使用するわけではありません）。

[第 8 章](#)に記載されたリコメンデーション・メソッドのいずれかを発行する前に、適切な [FilteringSettings クラス](#)、[TuningSettings クラス](#) および [Location クラス](#) のインスタンスを生成する必要があります。

この章で説明するメソッドは、すべてパブリックです。

この章では、クラスの詳細は説明しません。詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

サポート・クラスは、次の 2 つのカテゴリに分類されます。

- EnumType インタフェース
- その他のサポート・クラス

OP のレーティング

OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

RE Batch API クラスの位置

頻繁に使用される次のクラスは、`oracle.dmt.re.base` サブディレクトリにあります。

- `DataItem`
- `Enum`
- `FilteringSettings`
- `TuningSettings`

たとえば、`Enum` インタフェースを使用するには、Java プログラムに次の文を含める必要があります。

```
import oracle.dmt.op.re.base.Enum;
```

RE Batch API の EnumType インタフェース

RE Batch API メソッドの多くは、限られた数の値を取る属性を参照します。このような列挙のベース・クラスを実装するには、インタフェース `Enum` を使用します。

`Enum` インタフェースには、次の一般メソッドを含むネストされた `EnumType` クラスがあります。

```
int getId()

String toString()

String getName()

boolean isEqual(EnumType)
```

次のインタフェースは、`EnumType` を拡張します。

- `CategoryMembership`
- `DataSource`
- `InterestDimension`
- `PersonalizationIndex`
- `ProfileDataBalance`

- ProfileUsage
- Sorting

CategoryMembership インタフェース

CategoryMembershipType は、次のように実装されます。

- CategoryMembershipType (EnumType を拡張するクラス)
- CategoryMembership (インタフェース)

クラス CategoryMembership には、次のメソッドがあります。

```
CategoryMembershipType getType(String name)
CategoryMembershipType getType(int)
```

CategoryMembership は、カテゴリー一覧のカテゴリにフィルタリングを適用する方法を指定します。たとえば、Enum.CategoryMembership.EXCLUDE_ITEMS は、カテゴリーのアイテムをカテゴリー一覧から除外することを指定します。詳細は、この章の「[FilteringSettings クラス](#)」を参照してください。

CategoryMembership は、次の値を取ります。

- Enum.CategoryMembership.EXCLUDE_ITEMS
- Enum.CategoryMembership.INCLUDE_ITEMS
- Enum.CategoryMembership.EXCLUDE_CATEGORIES
- Enum.CategoryMembership.INCLUDE_CATEGORIES
- Enum.CategoryMembership.LEVEL
- Enum.CategoryMembership.SUBTREE_ITEMS
- Enum.CategoryMembership.SUBTREE_CATEGORIES
- Enum.CategoryMembership.ALL_ITEMS
- Enum.CategoryMembership.ALL_CATEGORIES

次の文は、変数 myEnum に Enum.CategoryMembership.LEVEL を割り当てます。

```
CategoryMembershipType myEnum = Enum.CategoryMembership.LEVEL;
```

DataSource インタフェース

DataSource は、次のように実装されます。

- DataSourceType (EnumType を拡張するクラス)
- DataSource (インタフェース)

クラス DataSourceType には、次のメソッドがあります。

```
DataSourceType getType(String name)
```

```
DataSourceType getType(int)
```

DataSource は、OP の特定のオペレーションに使用するデータのタイプを指定します。たとえば、Enum.DataSource.DEMOGRAPHIC は人口統計データを指定します。この章の後半で説明する [DataItem クラス](#) は、DataSource を使用します。1 つのメソッドが DataSource のすべての値に対応するわけではありません。詳細は、OP に含まれる Javadoc のメソッドの説明を参照してください。

DataSource は、次の値を取ります。

- Enum.DataSource.DEMOGRAPHIC
- Enum.DataSource.PURCHASING
- Enum.DataSource.RATING
- Enum.DataSource.NAVIGATION
- Enum.DataSource.ALL

次の文は、変数 myEnum に Enum.DataSource.ALL を割り当てます。

```
DataSourceType myEnum = Enum.DataSource.ALL;
```

InterestDimension インタフェース

InterestDimension は、次のように実装されます。

- InterestDimensionType (EnumType を拡張するクラス)
- InterestDimension (インタフェース)

クラス InterestDimensionType には、次のメソッドがあります。

```
InterestDimensionType getType(String name)
```

```
InterestDimensionType getType(int)
```

InterestDimension は、特定のアイテムに対して Web サイト・ユーザーが抱く関心の種類を示します。NAVIGATION は、ユーザーがそのアイテムに関心があることを示します。PURCHASING は、ユーザーがそのアイテムの購入を希望していることを示します。RATING は、ユーザーがそのアイテムを好むことを示します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

InterestDimension は、次の値を取ります。

- Enum.InterestDimension.NAVIGATION
- Enum.InterestDimension.PURCHASING
- Enum.InterestDimension.RATING

次の文は、変数 myEnum に Enum.InterestDimension.PURCHASING を割り当てます。

```
InterestDimensionType myEnum = Enum.InterestDimension.PURCHASING;
```

PersonalizationIndex インタフェース

PersonalizationIndex は、次のように実装されます。

- PersonalizationIndexType (EnumType を拡張するクラス)
- PersonalizationIndex (インタフェース)

クラス PersonalizationIndexType には、次のメソッドがあります。

```
PersonalizationIndexType getType(String name)
```

```
PersonalizationIndexType getType(int)
```

PersonalizationIndex は、返されるリコメンデーションがどれくらい「例外的」であるかを指定します。たとえば、LOW は例外的ではないことを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

PersonalizationIndex は、次の値を取ります。

- Enum.PersonalizationIndex.LOW
- Enum.PersonalizationIndex.MEDIUM
- Enum.PersonalizationIndex.HIGH

次の文は、変数 myEnum に Enum.PersonalizationIndex.LOW を割り当てます。

```
PersonalizationIndexType myEnum = Enum.PersonalizationIndex.LOW;
```

ProfileDataBalance インタフェース

ProfileDataBalance は、次のように実装されます。

- ProfileDataBalanceType (EnumType を拡張するクラス)
- ProfileDataBalance (インタフェース)

クラス `ProfileDataBalanceType` には、次のメソッドがあります。

```
ProfileDataBalanceType getType(String name)
```

```
ProfileDataBalanceType getType(int)
```

`ProfileDataBalance` は、リコメンデーションを作成するときに、現在のセッションからデータを取得するか、履歴から取得するか、現在のセッションと履歴の間でデータを均衡化するかを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

`ProfileDataBalance` は、次の値を取ります。

- `Enum.ProfileDataBalance.HISTORY`

注意： 大量のリコメンデーションで有効な唯一のプロファイル・データ・バランスの値は、`Enum.ProfileDataBalance.HISTORY` です。この値を指定する必要があります（現在のセッション・データは有効ではありません）。

次の文は、変数 `myEnum` に `Enum.ProfileDataBalance.HISTORY` を割り当てます。

```
ProfileDataBalanceType myEnum = Enum.ProfileDataBalance.HISTORY;
```

ProfileUsage インタフェース

`ProfileUsage` は、次のように実装されます。

- `ProfileUsageType` (`EnumType` を拡張するクラス)
- `ProfileUsage` (インタフェース)

クラス `ProfileUsageType` には、次のメソッドがあります。

```
ProfileUsageType getType(String name)
```

```
ProfileUsageType getType(int)
```

`ProfileUsage` は、リコメンデーション一覧に顧客のプロファイルにあるアイテムを含めるか、除外するかを指定します。詳細は、この章の「[TuningSettings クラス](#)」の説明を参照してください。

`ProfileUsage` は、次の値を取ります。

- `Enum.ProfileUsage.INCLUDE`
- `Enum.ProfileUsage.EXCLUDE`

次の文は、変数 `myEnum` に `Enum.ProfileUsage.INCLUDE` を割り当てます。

```
ProfileUsageType myEnum = Enum.ProfileUsage.INCLUDE;
```

Sorting インタフェース

Sorting は、次のように実装されます。

- `SortingType` (`EnumType` を拡張するクラス)
- `Sorting` (インタフェース)

クラス `SortingType` には、次のメソッドがあります。

```
SortingType getType(String name)
```

```
SortingType getType(int)
```

`Sorting` は、ソートを行うかどうか (`NONE` はソートなし)、ソートを行う場合はその方法 (昇順または降順) を示します。詳細は、この章の「[DataItem クラス](#)」の説明を参照してください。

`Sorting` は、次の値を取ります。

- `Enum.Sorting.NONE`
- `Enum.Sorting.DESENDING`
- `Enum.Sorting.ASCENDING`

次の文は、変数 `myEnum` に `Enum.Sorting.NONE` を割り当てます。

```
SortingType myEnum = Enum.Sorting.NONE;
```

その他の RE Batch API サポート・クラス

その他のサポート・クラスは次のとおりです。

- `DataItem`
- `FilteringSettings`
- `Location`
- `TuningSettings`

DataItem クラス

このクラスは、クラス `Item` のサブクラスです。これはアイテムに関するデータをカプセル化します。

このクラスには、次の 2 種類のメソッドがあります。

- `DataItem` インスタンスを作成するコンストラクタ
- 属性値を返すメソッド

次のメソッドは属性値を返します。

- `getDataSource()`
- `getValue()`

FilteringSettings クラス

このクラスは、リコメンデーションを生成するときどのアイテムを含めるかまたは除外するかを指定するのに使用します。

OP のリリース 9.0.2 は、カテゴリ・フィルタリングにのみ対応しています。

このクラスには、次の3種類のメソッドがあります。

- `FilteringSettings` のコンストラクタ
- 属性値を設定するメソッド
 - `setItemFiltering(int taxonomyID)`
 - `setItemFiltering(int taxonomyID, long[] categoryList)`
 - `setItemExclusion(int taxonomyID, long[] categoryList)`
 - `setItemSubTreeFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryExclusion(int taxonomyID, long[] categoryList)`
 - `setCategorySubTreeFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryLevelFiltering(int taxonomyID, long[] categoryList)`
 - `setCategoryFiltering(int taxonomyID)`
 - `setCategoryFiltering(int taxonomyID, long[] categoryList)`
- 属性値を返すメソッド
 - `getTaxonomyID()`
 - `getCategoryFiltering()`
 - `getCategoryList()`
 - `getCategoryMembership()`

どのメソッドにもすべてのフィルタリング設定を使用できるわけではありません。特に、次のフィルタリング設定は、抱合せ販売メソッドには使用できません。

- `setCategoryLevelFiltering`
- `setCategorySubtreeFiltering`
- `setCategoryExclusion`
- `setCategoryFiltering(int)`
- `setCategoryFiltering(int, long[])`

Item クラス

このクラスは、推奨可能なアイテムや、データ収集が可能なアイテムを示すのに使用します。アイテムは、`type` と ID の組合せで一意に示されます。アイテム ID は、特定の `type` で一意であることが必要ですが、異なる `type` で同じ ID を持つことはできます。

このクラスには、次の 2 種類のメソッドがあります。

- Item インスタンスを作成するコンストラクタ
- 属性値を返すメソッド
 - `getType()`
 - `getID()`

Location クラス

このクラスは、入力表の位置や、REProxyBatch メソッドの結果を含む表の位置を指定します。表のスキーマは、実行するコールに応じて異なります。詳細は、Javadoc の各メソッドの説明を参照してください。

このクラスには、次の 3 種類のメソッドがあります。

- Location インスタンスを作成するコンストラクタ
- 属性値を設定するメソッド
 - `setDBUrl()`
 - `setPassword()`
 - `setSchemaName()`
 - `setTableName()`
 - `setUserName()`

- 属性値を返すメソッド
 - `getDBUrl()`
 - `getDBAlias()`
 - `getSchemaName()`
 - `getTableName()`
 - `getUserName()`
 - `getPassword()`

TuningSettings クラス

このクラスは、リコメンデーションを算出するときに適用する設定を指定します。このクラスのインスタンスは、すべてのリコメンデーション要求に渡されます。

このクラスには、次の3種類のメソッドがあります。

- `TuningSettings` インスタンスを作成するコンストラクタ
- 属性値を設定するメソッド
- 属性値を返すメソッド

次のメソッドは属性値を設定します。

- `setDataSourceType()`
- `setInterestDimension()`
- `setPersonalizationIndex()`
- `setProfileDataBalance()`
- `setProfileUsage()`

次のメソッドは属性値を返します。

- `getDataSourceType()`
- `getInterestDimension()`
- `getPersonalizationIndex()`
- `getProfileDataBalance()`
- `getProfileUsage()`

リコメンデーション・エンジン・バッチ・ プロキシの使用

この章では、顧客プロファイルの管理とリコメンデーションの取得に使用するクラスやメソッドの概要を説明します。これらのメソッドのサポート・クラスは、[第7章](#)に記載されています。

これらのメソッドの詳細は、Oracle9i Application Server Documentation Library の OP のセクションにある Javadoc を参照してください。

この章で説明するメソッドは、すべてパブリックです。

REProxyBatch の概要

リコメンデーション・プロキシ (REProxyBatch) メソッドは、次のように機能ごとに分類できます。

- 顧客プロファイル管理 (顧客プロファイルのロードとページ) を含む、プロキシの作成と管理
- リコメンデーション・メソッド (リコメンデーションの取得)

これらのクラスとメソッドの使用例は、[第 9 章](#)を参照してください。

REProxyBatch クラスの位置

REProxyBatch (とその例外) を使用するには、Java プログラムに次の文を含める必要があります。

```
import oracle.dmt.op.re.reapi.batch.*;

import oracle.dmt.op.re.reexception.*;
```

これらのクラスは、Oracle9iAS がインストールされているシステムにインストールされます。

REProxyBatch の作成と管理

REProxyBatch.java クラスは、メソッドを実行する RE スキーマへの JDBC 接続を確立します。接続は、destroy() メソッドで明示的に破棄するまで存続します。また、このクラスは顧客プロファイル管理メソッドも含みます。

顧客プロファイル管理

リコメンデーションを要求する前に、MTR から RE に顧客プロファイルをロードする必要があります。終了後は、ロードしたプロファイルを RE からページする必要があります。メソッドは次のとおりです。

- LoadCustomerProfiles()
- PurgeCustomerProfiles()

REProxyBatch のリコメンデーション

次のメソッドでリコメンデーションを取得します。

- crossSellForItems
- rateItem
- recommendTopItems

アプリケーションでは、返されたリコメンデーションをエンド・ユーザーに伝達します。リコメンデーションは出力表に書き込まれます。出力表のスキーマは、呼び出したメソッドに応じて異なります。詳細は、各メソッドの説明を参照してください。

OP のレーティング

OP のレーティングは「昇順の適合度」で示されます。これは、レーティングが高いほど、ユーザーがそのアイテムを好むことを意味します。レーティングが低いアイテムは、ユーザーが好まないアイテムです。OP アルゴリズムでこの仮定条件を使用するため、適合度の高い順番にレーティングを並べることが重要です。

リコメンデーションの戻り値の意味

`ItemDetailData.attribute` が `Enum.RecommendationAttribute.PREDICTION` と等しい場合、リコメンデーション・インスタンスに返された値の意味は、次のように `interestDimension` の値によって異なります。

- `InterestDimension.RATING` の場合、アイテムの予測されるレーティング値が返されます。
- `InterestDimension.PURCHASING` や `InterestDimension.NAVIGATION` の場合、評価された確率が返されます。最も見込みが高いアイテムとの比較に基づいて、最も見込みが高いアイテムに 1 の値が割り当てられ、その他のアイテムには 1 未満の比例する値が割り当てられます。

抱合せ販売メソッドの使用法

この項の説明は、`recommendCrossSellForItems` に適用します。

インタレスト・ディメンションは、入力項目のデータ・ソース・タイプと同じにする必要があります。

データ・ソース・タイプは、`NAVIGATION` または `PURCHASING` にする必要があります。その他のタイプはサポートされていません。

このメソッドには、次のフィルタリング設定を使用できません。

- `setCategoryLevelFiltering`
- `setCategorySubtreeFiltering`
- `setCategoryExclusion`
- `setCategoryFiltering(int)`
- `setCategoryFiltering(int, long[])`

リコメンデーション・メソッドの使用方法

`recommendTopItems` は、必ずしもアイテム一覧を返すわけではありません。`FilteringSettings.CategoryMembership` を次のいずれかの値に設定した場合、`recommendTopItems` は、カテゴリー一覧を返します。

- `Enum.CategoryMembership.EXCLUDE_CATEGORIES`
- `Enum.CategoryMembership.INCLUDE_CATEGORIES`
- `Enum.CategoryMembership.SUBTREE_CATEGORIES`
- `Enum.CategoryMembership.ALL_CATEGORIES`

カテゴリーは分類（TAXONOMY）の構成要素です。分類は、マイニング・テーブル・リポジトリ（MTR）の次の表に定義します。

- `MTR_TAXONOMY`
- `MTR_TAXONOMY_CATEGORY`
- `MTR_TAXONOMY_CATEGORY_ITEM`
- `MTR_CATEGORY`

OP アプリケーションの設計には、適切な分類が不可欠です。分類の作成方法の詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

REProxyBatch の規則とリコメンデーション

OP は、RE に格納された規則表を使用して、リコメンデーション・メソッドが要求したリコメンデーションを生成します。規則表は、パッケージのビルド時に作成され、パッケージの配布時に RE に格納されます。使用する規則表は、RE Batch API コールによって異なります。通常は、規則の先行条件をキャッシュのデータ（RE バッチは履歴データのみ）と一致させ、各種の結果の確率を算出します。これらの項目は確率の順番に並べられ、`numberOfItems`（API 引数）項目が返されます。

REProxyBatch API の例と使用方法

この章では、REProxyBatch API の使用例を示します。必要に応じて、コーディング・スケルトンを提供することも、OP を使用して特定の問題を解決する方法を説明することもあります。

REProxyBatch API の基本的な使用方法

第 8 章で説明した REBatchProxy メソッドを使用して、リコメンデーションを生成する Java プログラムを作成できます。

注意： RE Batch API クラスは、Oracle9iAS がインストールされているシステムにインストールされます。使用する表は別のシステム（Oracle9i がインストールされているシステム）にインストールされます。正しいシステムで次の手順を実行する必要があります。

REProxyBatch API コールを使用するには、次の手順を実行する必要があります。

1. パッケージを作成し、リコメンデーションで使用する RE に配布します。
2. REBatchProxy のインスタンスを作成します。
3. 必要な表を作成します（あるいは、プログラムを実行する前に、SQL を使用して表を作成することもできます）。
4. 顧客プロファイルをロードします。
5. 必要なリコメンデーション・メソッドを実行します。
6. 手順 4 でロードした顧客プロファイルをページします。
7. 手順 2 で作成したデータベース接続を破棄します。

これで、要求したリコメンデーションを含む表が作成されます。SQL を使用して、この表を確認できます。

コード・サンプル：顧客情報によるリコメンデーション

次のコード・サンプルは、リコメンデーションの取得を示します。

```
// Create an instance of REProxyBatch
// Create customer table
// Load customer profiles
// Execute recommend_top
// Purge customer profiles loaded above
// Destroy the database connection held by REProxyBatch
```

コード・サンプル：抱合せ販売によるリコメンデーション

次のコード・サンプルは、抱合せ販売のリコメンデーションの取得を示します。

```
// Create an instance of REProxyBatch
// Create Items table
// Execute cross sell for items
// Destroy the database connection held by REProxyBatch
```

リコメンデーション・エンジンの使用方法

REBatchProxy では、1 つ以上のリコメンデーション・エンジン・ファームに 1 つ以上のリコメンデーション・エンジン（RE）が必要です。

大量のリコメンデーションに使用する RE は、他の目的には使用しないでください。

注意： バッチ・プログラムの実行中にパッケージを配布すると、配布は失敗します。

一般に、複数の RE を使用して十分なリコメンデーション・パフォーマンスを得る必要があります。多くのアプリケーションでは、異なるマシンと異なるデータベース・インスタンスで複数の RE を使用します。

一般に、アプリケーションではこれらの RE が同じ RE ファームに属します。物理システムに複数のプロセッサがあり、データベースがプロセッサを有効に利用できる場合、ユーザー数に対して必要な RE 数を場合によっては 1 つにまで減らせます。詳細は、『Oracle9iAS Personalization 管理者ガイド』を参照してください。

アプリケーションで複数の RE を利用する場合、どの RE を使用するかを決定する必要があります。異なる顧客プロファイルを別々の RE にロードし、適切なリコメンデーションを生成して、必要に応じてリコメンデーション表をマージできます。

複数通貨の処理

OP は、通貨データを人口統計表（たとえば、個人の所得）に数字で格納します。すなわち、OP はどのようなラベルも格納しません。10 ドル（米国）も、10 ポンド（英国）も「10」として格納されます。

通貨データが正しく解釈されているかどうかを確認する方法はいくつかあり、アプリケーションに選択する対処法は、アプリケーションが通貨データをどのように使用するかによって異なります。

- 顧客の人口統計データに国別コードを含めます。

この対処法により、国を考慮できますが、値と国を密接に関連付けません。

- すべての通貨を、ユーロや US ドルなどの共通通貨に変換します。

この対処法では、個別の通貨値を有意義に比較できますが（10 ポンドは 10 US ドルより価値が高い）、米国における 30,000 US ドルの給料とブラジルにおける同じ 30,000 US ドルの給料という、データ間の違いは保持できません。このような情報は、たとえば、米国とブラジルの両方で高所得者にアイテムを推奨する場合に必要です。というのは高所得者の給料を US ドルで表した場合、国によってその額にはかなりの開きがあるからです。

この方法では、OP がリコメンデーションを作成する前に、OP 外でデータを前処理する必要があります。

- 平均に基づいて通貨値のビンニングを行い、国間で比較できる相対値を取得します。

この対処法では、たとえば、ある国の高所得者を求めることができますが、適切にビン範囲を決定し、管理する必要があります。

この方法では、OP がリコメンデーションを作成する前に、OP 外でデータを前処理する必要があります。

人口統計データの使用

MTR_CUSTOMER 表のスキーマは、サイト・データベースの任意の列にマッピングできる 50 の汎用属性で構成されています。異なるデータ型をすべてサポートするため、属性はすべて VARCHAR 型です。したがって、マッピングされた列を文字列に変換する必要があります。OP のこのリリースでは、このようにマッピングされた列は、カテゴリまたは数値のみとして処理されます。マッピングされた列が DATA 属性である場合、TO_NUMBER 関数を使用して変換する必要があります。変換した値は、ビン範囲を指定すると、他の属性と同じようにビンニングを行うことができます。

人口統計データのビンニングがあります。ビンニングを行う属性は boolean 型です。OP では、ビン番号は内部で整数として表されますが、実際の値は呼び出し側アプリケーションに返されます。つまり、Web アプリケーションは実数値を渡し、実数値を受け取ります。

時間ベースのアイテムの処理

航空券のようなアイテムの場合、アイテムを購入した時間によって価格が異なります。たとえば、ボストンからロンドンへの航空券には、搭乗日の 6 か月前に購入した場合の価格と、搭乗日の 2 日前に購入した場合の価格があります。

購入した時間には関係なく、Web アプリケーションが同じ搭乗便のすべての航空券に同じアイテム ID を割り当てる場合、そのアイテムは「6 か月前」や「2 日前」などの異なるアイテム・タイプを持つ必要があります。あるいは、アプリケーションがアイテムに分類を定義し、カテゴリにリコメンデーションを取得することもできます。

アプリケーションが、異なる時間に購入した同じ搭乗便の航空券に別々のアイテム ID を割り当てる場合（6 か月前に購入した航空券と 2 日前に購入した同じ搭乗便の航空券で ID が異なる場合）、すべての航空券が同じアイテム・タイプを取れます。この場合、リコメンデーション・アイテム ID が適切ではないことがあるため、アプリケーションが分類を定義し、カテゴリのリコメンデーションを要求する必要があります。

REAPI サンプル・プログラム

この付録には、REAPI で使用するサンプル Java プログラム (ProxyTest.java) が記載されています。

このプログラムを実行する前に、適切なモデルを作成し、RE に配布する必要があります。データが返されない場合、モデルが適切でない可能性があります。コードは、Oracle9iAS がインストールされているシステムの \${ORACLE_HOME}/dmt/reapi/rt/ にインストールされます。

注意： REAPI は、Oracle9iAS がインストールされているシステムにインストールされます。このプログラムは、そのシステムで実行するほうが簡単です。

```
// Copyright (c) 2001, 2002 Oracle Corp

////////////////////////////////////
//
// Test program exercises the functionality of
// REAPI.
//
// Step 1 creates a unique session ID
// Step 2 creates a proxy instance
// Step 3 creates a session
// Step 4 creates settings data (IdentificationData, TuningSettings,
//      FilteringSettings, hotPick list, item list)
// Step 5 gets recommendations and ratings
// Step 6 closes session
// Step 7 destroys the proxy and flushes data cache
////////////////////////////////////

import java.math.BigDecimal;
import java.lang.Long;
```

```

import java.sql.*;
import java.io.IOException;
import java.io.StringWriter;
import java.io.PrintWriter;
import oracle.jdeveloper.cm.CMException;
import oracle.dmt.op.re.reapi.rt.*;
import oracle.dmt.op.re.reapi.batch.*;
import oracle.dmt.op.re.reexception.*;
import oracle.dmt.op.re.base.*;
import oracle.dmt.oputil.exceptions.MessageLogException;
import oracle.dmt.oputil.exceptions.StringTooLongException;

/**
 * Class ProxyTest
 * <P>
 * @author Oracle Corporation
 */
public class ProxyTest
{
    static boolean bVerbose;
    static final String SESSIONEXISTS = "";
    /**
     * Constructor
     */
    public ProxyTest()
    {
    }

    /**
     * main
     * @param args
     */
    public static void main(String[] args) throws ClassNotFoundException
    {
        long lStart;
        long lTotal = 0;
        String sProxyName = "REP1";
        String sdbURL = "jdbc:oracle:thin:@server-name:1521:darw900"; // sdbURL must be
correct for your installation
        String sUser = "RE01";
        String sPass = "REPW";

        int cSize = 3000; // Kbytes
        int interval = 10000; // in millisec
        new ProxyTest();
        REProxyRT proxy;
        // Step 1: Create a unique Session ID.

```

```

String custID = "1";
java.util.Date tmp = new java.util.Date();
Long tmpInt = new Long(tmp.getTime());
String sessionID = tmpInt.toString();

String trace = null;

lStart = System.currentTimeMillis();
try
{
    // Step 2: Get a proxy instance.
    if ((proxy = REProxyManager.getProxy(sProxyName)) == null)
        proxy = REProxyManager.createProxy(sProxyName, sdbURL, sUser, sPass, cSize,
interval);

    // Step 3: create OP session
    proxy.createCustomerSession(custID, sessionID);

    // Step 4: create settings data
    IdentificationData idData =
        IdentificationData.createSessionful(
            sessionID,
            Enum.User.CUSTOMER);

    TuningSettings tunings = new TuningSettings(
        Enum.DataSource.NAVIGATION,
        Enum.InterestDimension.NAVIGATION,
        Enum.PersonalizationIndex.HIGH,
        Enum.ProfileDataBalance.BALANCED,
        Enum.ProfileUsage.EXCLUDE);

    long[] hotPickGroups = {1,2,3,4,5,6,7,10,11};

    long[] m_catList = {1,2,3,4,5};

    FilteringSettings filters =
        new FilteringSettings(1);
    int taxonomy=1;
    filters.setItemFiltering( taxonomy, m_catList);
    RecommendationContent recContent = new
RecommendationContent (Enum.Sorting.ASCENDING);

    try{

//Create list of items for testing
DataItem[] items = new DataItem[4];

```

```

        items[0] = new DataItem(
            "MOVIE",
            72,
            Enum.DataSource.RATING,
            "1.5");
    items[1] = new DataItem(
        "MOVIE",
        287,
        Enum.DataSource.RATING,
        "1.5");
    items[2] = new DataItem(
        "MOVIE",
        122,
        Enum.DataSource.RATING,
        "1.5");
    items[3] = new DataItem(
        "MOVIE",
        1300,
        Enum.DataSource.RATING,
        "1.5");
    int count = 1;

    // Step 5: Get recommendations and ratings and print them out.
    // Note use of toString.
    try{
    System.out.println("\n##### " + count++ + " #####");
    //Call recommendBottomItems
    RecommendationList es1 = proxy.recommendBottomItems(
        idData,
        10,
        tunings,
        filters,
        recContent);
    System.out.println("");
    es1.toString();
    } catch(ErrorExecutingRE e) {
        e.printStackTrace();
    }

    try{
    System.out.println("\n##### " + count++ + " #####");
    //Call rateItems
    RecommendationList es2 = proxy.rateItems(
        idData,
        items,
        1,
        tunings,

```

```

        recContent);
System.out.println("");
System.out.println(es2.toString());
} catch(ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("Yn##### " + count++ + " #####");
//call selectFromHotPicks
RecommendationList es3 = proxy.selectFromHotPicks(
    idData,
    10,
    hotPickGroups,
    tunings,
    filters,
    recContent);
System.out.println("");
System.out.println("");
System.out.println(es3.toString());
} catch(ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("Yn##### " + count++ + " #####");
//Call crossSellForItemFromHotPicks
RecommendationList es4 = proxy.crossSellForItemFromHotPicks(
    idData,
    items[0],
    10,
    hotPickGroups,
    tunings,
    filters,
    recContent);
System.out.println("");
System.out.println(es4.toString());
} catch(ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("Yn##### " + count++ + " #####");
//Call recommendCrossSellForItem
RecommendationList es5 = proxy.recommendCrossSellForItem(
    idData,

```

```

        items[0],
        10,
        tunings,
        filters,
        recContent);
System.out.println("");
System.out.println(es5.toString());
} catch (ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("\n##### " + count++ + " #####");
RecommendationList es6 = proxy.recommendCrossSellForItems(
    idData,
    items,
    10,
    tunings,
    filters,
    recContent);
System.out.println("");
System.out.println(es6.toString());
} catch (ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("\n##### " + count++ + " #####");
RecommendationList es7 = proxy.crossSellForItemsFromHotPicks(
    idData,
    items,
    10,
    hotPickGroups,
    tunings,
    filters,
    recContent);
System.out.println("");
System.out.println(es7.toString());
} catch (ErrorExecutingRE e) {
    e.printStackTrace();
}

try{
System.out.println("\n##### " + count++ + " #####");
float es9 = proxy.rateItem(
    idData,

```

```

        items[2],
        1,
        tunings,
        recContent
    );
    System.out.println("");
    System.out.println("Result for recomend item:  " + es9);
    } catch(ErrorExecutingRE e) {
        e.printStackTrace();
    }

    try{
        System.out.println("Yn##### " + count++ + " #####");
        RecommendationList es10 = proxy.recommendFromHotPicks(
            idData,
            10,
            hotPickGroups,
            tunings,
            filters,
            recContent);
        System.out.println("");
        System.out.println(es10.toString());
    } catch(ErrorExecutingRE e) {
        e.printStackTrace();
    }

    try{
        System.out.println("Yn##### " + count++ + " #####");
        RecommendationList es11 = proxy.recommendTopItems(
            idData,
            10,
            tunings,
            filters,
            recContent);
        System.out.println("");
        System.out.println(es11.toString());
    } catch(ErrorExecutingRE e) {
        e.printStackTrace();
    }

    } catch(BadDBConnectionException bdbe) {
        bdbe.printStackTrace();
    }catch (ClassNotFoundException exc) {
        exc.printStackTrace();
    }
}

```

```
// Step 6: Close session
proxy.closeSession(idData);

// Step 7: Call destroy proxy (will flush data cache)
REProxyManager.destroyProxy(sProxyName);

    } catch (ErrorExecutingRE se) {
        System.err.println(se);
    } catch (InvalidIDException iie) {
        System.err.println(iie);
    } catch (BadDBConnectionException bdbe) {
        bdbe.printStackTrace();
    } catch (Exception e) {
        System.err.println(e);
        e.printStackTrace();
    }
}

}
```

REProxyBatch サンプル・プログラム

REProxyBatch のサンプル・プログラムは、Java プログラムとプロパティ・ファイルで構成されます。サンプル・プログラム、プロパティ・ファイルおよびプログラムの実行に必要な表は、OP のインストール時にインストールされます。

RE バッチ・サンプル・プログラムの概要

Java プログラム REBatchTest.java とプロパティ・ファイル batchtest.txt は、OP をインストールしたシステムにあります。

REBatchTest.java の REProxyBatch を使用して、リコメンデーションのサブセットをバルク・モードで実行できます (REProxyRT は、一度に 1 人のユーザーまたは 1 つのアイテムを評価します)。REProxyBatch は、評価するアイテムや顧客の一覧を入力表から読み込み、その結果を新しい出力表に書き込みます。このプログラムは、プロパティ・ファイル batchtest.txt からその入力を読み込みます。

RE バッチ・サンプル・プログラムの出力

入力項目の細目 (rateItem と crossSellForItem) は、OP デモ・データから導出します。ただし、OP では、同じデータで作成したモデルが、実行するたびに常に同じ規則を生成することは保証されません。したがって、評価中の項目を現在の規則セットで評価できない場合があります。出力表は空 (ゼロ行) か、予想よりも少ないレコードが含まれます (アイテムの一部のみが有効な抱合せ販売候補である場合)。

RE バッチ・サンプル・プログラムの実行

サンプル・プログラムを実行する手順は、次のとおりです。

1. OP をインストールします。
2. サンプル・プログラムのコードとデータは、OP のインストール時に次のディレクトリにインストールされます。
 - 次のコードが \${ORACLE_HOME}/dmt/reapi/batch/ にインストールされます。
 - batchtest.txt
 - README.txt
 - REBatchTest.java
 - サンプル・プログラムで使用するデータに関連する次の項目は、\${ORACLE_HOME}/dmt/reapi/batch/sampleData にインストールされます。
 - create_batch_demo_input_tables.sql
 - customer_list_in.ct1
 - customer_list_in.txt
 - item_list_in.ct1
 - item_list_in.txt
 - load_batch_demo_data.sh

3. シェル・スクリプト `load_batch_demo_data.sh` を実行して、次の表をロードします。
 - `customer_list_in`: `loadCustomerProfile` に使用 (`loadCustomerProfile` の出力は、`recommendTopItems` と `rateItem` で使用)
 - `item_list_in`: `crossSellForItem` に使用
4. サンプル・コードをコンパイルします。CLASSPATH 変数は、次の zip/jar ファイルを含む必要があります。
 - `${ORACLE_HOME}/dmt/opreapi-batch.jar`
 - `${ORACLE_HOME}/dmt/oputil.jar`また、JDBC 関連の jar/zip ファイルも含む必要があります。
 - `${ORACLE_HOME}/jdbc/lib/classes12.zip`
5. プロパティ・ファイルが適切なエントリを示すよう変更します。プロパティ・ファイルとファイル `README.txt` のコメントは、実際の変更内容を示します。
6. プロパティ・ファイル名を入力パラメータとして、`REBatchTest` を実行します。

RE バッチ・サンプル・プログラム・コード

この項では、サンプル・プログラム・コードとそのプロパティ・ファイルを示します。

batchtest.txt

サンプル・プログラムのプロパティ・ファイルは次のとおりです。RE の詳細と入出力表の詳細を置き換え、インストール内容を反映する必要があります。

```
###
### Input file for REProxyBatch sample program
### Before Running, you will need to replace the following dummy strings with actual information:
### 1. RE* details ( Url,Username,Password) to point to the RE.
### 2. Input and Output (Result) table details for each of the calls.

#A unique name for proxy
ProxyName=REB_1

#Recommendation Engine details
REUrl=jdbc:oracle:thin:@myDBUrl
REUsername=REUser
REPassword=REPassword

#Input customer table location
Input.Url=jdbc:oracle:thin:@myDBUrl
Input.Alias=myDBAlias
Input.Schema=User1
Input.Table=customer_list_in
Input.Username=User1
Input.Password=Password1

#Customer profile table
# This table is created in RE by loadCustomerProfile. Once created
# it is used for recommendTopItems and rateItem
CustProfile=MY_CUSTOMER_PROFILE

#
# Details for recommendTopItems
#
# Number of items to be recommended per customer
TopN.NumberOfItems=10
#TuningSettings details
#valid DataSourceTypes are ALL, DEMOGRAPHIC, PURCHASING, RATING, NAVIGATION
TopN.DataSourceType=ALL
#valid InterestDimension: PURCHASING, RATING, NAVIGATION
TopN.InterestDimension=PURCHASING
#valid PersonalizationIndex: LOW, MEDIUM, HIGH
```

```
TopN.PersonalizationIndex=MEDIUM
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
#valid ProfileDataBalance: HISTORY, CURRENT, BALANCED
TopN.ProfileDataBalance=HISTORY
#valid ProfileUsage:INCLUDE, EXCLUDE
TopN.ProfileUsage=INCLUDE
# FilteringSettings details
TopN.Taxonomy=1
#Category list is a series of numbers separated by "-"
TopN.CategoryList=1-2-3-4-5
#Valid CategoryMembership: ExcludeItems, ExcludeCategories, IncludeItems, IncludeCategories,
#    level, SubTreeItems, SubTreeCategories, AllItems, AllCategories
TopN.CategoryMember=AllItems
# Result table details
TopNResult.Url=jdbc:oracle:thin:@myDBUrl
TopNResult.Alias=myDBAlias
TopNResult.Schema=User2
TopNResult.Table=TopN_RESULTS
TopNResult.Username=User2
TopNResult.Password=Password2

#
# Details for rateItem
#
#TuningSettings details
RateI.ItemID=417
RateI.ItemType=MOVIE
RateI.DataSourceType=RATING
RateI.InterestDimension=RATING
RateI.PersonalizationIndex=LOW
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
RateI.ProfileDataBalance=HISTORY
RateI.ProfileUsage=INCLUDE
RateI.Taxonomy=1
# Result table details
RateIResult.Url=jdbc:oracle:thin:@myDBUrl
RateIResult.Alias=myDBAlias
RateIResult.Schema=User3
RateIResult.Table=RATEITEM_RESULTS
RateIResult.Username=User3
RateIResult.Password=Password3

#
# Details for crossSellForItem
#
```

```
#Input items table details
ItemTable.Url=jdbc:oracle:thin:@myDBUrl
ItemTable.Alias=myDBAlias
ItemTable.Schema=User4
ItemTable.Table=item_list_in
ItemTable.Username=User4
ItemTable.Password=User4
# Number of items to be recommended per input item
XSell.NumberOfItems=10
#TuningSettings details
XSell.DataSourceType=NAVIGATION
XSell.InterestDimension=NAVIGATION
XSell.PersonalizationIndex=HIGH
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
XSell.ProfileDataBalance=HISTORY
XSell.ProfileUsage=EXCLUDE
#FilteringSettings details
XSell.Taxonomy=1
XSell.CategoryList=1-3-5-7-9
XSell.CategoryMember=AllItems
# Result table details
XSellResult.Url=jdbc:oracle:thin:@myDBUrl
XSellResult.Alias=myDBAlias
XSellResult.Schema=User4
XSellResult.Table=XSELL_RESULTS
XSellResult.Username=User5
XSellResult.Password=Password5
```


REBatchTest.java

サンプル・プログラムは次のとおりです。RE の詳細と入出力表の詳細を置き換え、インストール内容を反映する必要があります。

```
// Copyright (c) 2001 Oracle

import java.math.BigDecimal;
import java.lang.Long;
import java.sql.*;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Properties;
import java.util.Enumeration;
import java.util.StringTokenizer;

import oracle.dmt.op.re.base.Enum;
import oracle.dmt.op.re.base.FilteringSettings;
import oracle.dmt.op.re.base.Item;
import oracle.dmt.op.re.base.TuningSettings;

import oracle.dmt.op.re.reapi.batch.Location;
import oracle.dmt.op.re.reapi.batch.REProxyBatch;
import oracle.dmt.op.re.reexception.*;
import oracle.dmt.oputil.exceptions.StringTooLongException;

/**
 * Class REBatchTest
 * This class demonstrates the use of REProxyBatch API included with Oracle
 * Personalization. REProxyBatch allows you to execute a subset of recommendation
 * functions in bulk. (REProxyRT scores one user/item at a time.)
 *
 * REProxyBatch reads a list of items/customers to be scored from an input table
 * and writes the result to a new output table. This program reads its input
 * from a property file supplied as input. The tags used in the input file are
 * listed here as static constants in CAPITAL LETTERS.
 *
 * Before you execute the program, you must edit the property file batchtest.ini
 * to reflect your environment.
 *
 * @author Oracle Corporation
 */
public class REBatchTest
{
    static private Properties m_Props;

    static private boolean m_Verbose;
```

```
static private final String s_Title = "REBatchTest";
static final boolean debug = true;

// These tags appear in the input property file.
static String TOPN = "TopN";
static String XSELL = "XSell";
static String RITEM = "RateI";
static String DSTYPE = "DataSourceType";
static String INIDIMENSION = "InterestDimension";
static String PDBALANCE = "ProfileDataBalance";
static String PUSAGE = "ProfileUsage";
static String PINDEX = "PersonalizationIndex";
static String NUMOFITEMS = "NumberOfItems";
static String PROXYNAME = "ProxyName";
static String REURL = "REUrl";
static String REUSERNAME = "REUsername";
static String REPASSWORD = "REPassword";
static String URL = "Url";
static String ALIAS = "Alias";
static String USNAME = "Username";
static String PSWORD = "Password";
static String SCHEMA = "Schema";
static String TABLE = "Table";
static String INPUT = "Input";
static String CUSTPROFILE = "CustProfile";
static String ITEMID = "ItemID";
static String ITEMTYPE = "ItemType";
static String RESULT = "Result";
static String TAXONOMY = "Taxonomy";
static String ITEMTAB = "ItemTable";
static String CATEGORYS = "CategoryList";
static String CATNODE = "CategoryMember";
static String DOT = ".";

// These variables are initialized from the property file.
static private String m_proxyName; // Name of the REProxyBatch instance
// Recommendation Engine Details
static private String m_reUrl;
static private String m_reUserName;
static private String m_rePassword;

// Location of the customer table; the customer table must be created before
// this program can be run.
// Refer to the REProxyBatch document for the expected schema.
static private Location m_inCustTable;
// Location of the items table; the itemstable must be created before this
// program can be run.
```

```

// Refer to the REProxyBatch document for the expected schema.
static private Location m_inItemTable;

// Name of the customer profile table created in RE after loadCustomerProfile
// is invoked.
static private String m_custProfile;

/**
 * main Entry point
 * @param args
 */
public static void main(String[] args) throws ClassNotFoundException, FileNotFoundException
{
    long lStart;
    long lTotal = 0;
    REProxyBatch proxy;
    // Name of the property file should be passed as an input.
    if (args.length < 1) {
        System.out.println("Usage: REBatchTest FullpathToPropfile");
        System.exit(-1);
    }
    // Initialize class variables using the property file.
    try {
        new REBatchTest(args[0]);
    } catch (Exception ioe) {
        System.out.println("Error in main(): " + ioe.getMessage());
        ioe.printStackTrace();
        System.exit(-1);
    }
    // Start a timer.
    lStart = System.currentTimeMillis();
    try {
        // Initialize a REProxyBatch object
        proxy = new REProxyBatch(m_proxyName,
                                m_reUrl,
                                m_reUserName,
                                m_rePassword);

        // Invoke loadCustomerProfiles.
        System.out.println("Loading customer profile.");
        proxy.loadCustomerProfiles(
            m_inCustTable, // Name of the input customer IDs table created by the end user
            m_custProfile); // Name of the profile table created by loadCustomerProfiles in RE
        System.out.println("Completed loading customer profile, in the table: " + m_custProfile + " in
current R
E schema.");
    }
}

```

```
// Initialize inputs needed for recommendTopItems
// Number of items to recommend (per customer)
int nRec = getNumOfItems(TOPN);
// Initialize TuningSettings object from the property file
TuningSettings tuningT = getTuning(TOPN);
// Initialize FilteringSettings object from the property file
FilteringSettings filterT = getFilter(TOPN);
// Location of the result table
Location resLocT = getLoc(addStr(TOPN, RESULT));
System.out.println("Recommending top items. Customer profile table name:" + m_custProfile );

// Invoke recommendTopItems
proxy.recommendTopItems(
    m_custProfile, // Customer profile table name, created by loadCustomerProfiles
    nRec, // Number of recommendations per customer.
    tuningT, // TuningSettings to be used for recommendations
    filterT, // FilteringSettings to be used for recommendations
    resLocT); // Location of the result table
System.out.println("Completed recommendTopItems.");
System.out.println("Result table details: Schema: " + resLocT.getSchemaName() + ", Table Name: "
+ resLo
cT.getTableName() + ", Database: " + resLocT.getDBAlias());

// Initialize inputs needed for rateItem
// Read the item to be rated. This single item is rated for all the customers in
// the customer profiles table.
Item item = getItem(RITEM);
// Taxonomy to be used for rating
int nTaxID = Integer.parseInt(m_Props.getProperty(addStr(RITEM, DOT, TAXONOMY)));
// Initialize TuningSettings object from the property file
TuningSettings tuningR = getTuning(RITEM);
// Location of the result table,
Location resLocR = getLoc(addStr(RITEM, RESULT));
System.out.println("Rating item.");
System.out.println("Rating item: " + item.getID() + "," + item.getType() + ". Customer profile
table nam
e:" + m_custProfile );
// Invoke rateItem
proxy.rateItem(
    m_custProfile, // Customer profile table name, created by loadCustomerProfiles
    item, // Item to be rated
    nTaxID, // Taxonomy to be used for rating
    tuningR, // TuningSettings to be used for rating
    resLocR); // Location of the result table
System.out.println("Completed rateItem.");
System.out.println("Result table details: Schema: " + resLocR.getSchemaName() + ", Table Name: "
+ resLo
```

```

cR.getTableName() + ", Database: " + resLocR.getDBAlias());

    // Invoke purgeCustomerProfiles
    // Cross sell is an item based recommendation call. Hence it does not need
    // the loaded customer profile table.
    System.out.println("Purging customer profile: " + m_custProfile + " from current RE.");
    proxy.purgeCustomerProfiles(m_custProfile);

    // crossSellForItem
    // Initialize inputs needed for rateItem
    // Number of cross sell items to be recommended (per input item)
    nRec = getNumOfItems(XSELL);
    // Initialize TuningSettings object from the property file
    TuningSettings tuningX = getTuning(XSELL);
    // Initialize FilteringSettings object from the property file
    FilteringSettings filterX = getFilter(XSELL);
    // Location of the result table
    Location resLocX = getLoc(addStr(XSELL, RESULT));
    System.out.println("Recommending cross sell for item. Input table details: Schema: " + m_
inItemTable.get
SchemaName() + ", Table Name: " + m_inItemTable.getTableName() + ", Database: " + m_
inItemTable.getDBAlias());

    // Invoke crossSellForItem
    proxy.crossSellForItem(
        m_inItemTable, // Name of the input items table created by the end user
        nRec, // Number of cross sell items to be recommended per input item
        tuningX, // Tuning settings to be used for recommendations
        filterX, // Filtering settings to be used for recommendations
        resLocX); // Location of the result object
    System.out.println("Completed crossSellForItem.");
    System.out.println("Result table details: Schema: " + resLocX.getSchemaName() + ", Table Name: "
+ resLo
cX.getTableName() + ", Database: " + resLocX.getDBAlias());

    // Destroy the proxy object (to free up the database connections).
    System.out.println("Completed REProxyBatch operations. Destroying the proxy.");
    proxy.destroy();
    proxy = null;
} catch (NullPointerException npe) {
    System.out.println(npe.getMessage());
    npe.printStackTrace();
} catch (BadDBConnectionException bde) {
    System.out.println(bde.getMessage());
    bde.printStackTrace();
} catch (SQLException se) {
    System.out.println(se.getMessage());
}

```

```
        se.printStackTrace();
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

/**
 * Constructor
 * Read the property file.
 */
public REBatchTest(String sProp)
throws FileNotFoundException, IOException, StringTooLargeException
{
    getConfig(sProp);
}

/**
 ** All the remaining code in this file deals with reading different values
 ** from the property file and initializing class variables.
 **/

/*
 * A helper function returning appended string
 */
static private String addStr(String s1, String s2) {
    StringBuffer sb = new StringBuffer(s1);
    sb.append(s2);
    return (sb.toString());
}

/*
 * A helper function returning appended string
 */
static private String addStr(String s1, String s2, String s3) {
    StringBuffer sb = new StringBuffer(s1);
    sb.append(s2).append(s3);
    return (sb.toString());
}

/**
 ** Read recommendation engine details from the property file.
 **
 */
static private void getConfig(String propFile)
throws FileNotFoundException, SecurityException, IOException, StringTooLargeException {
    FileInputStream is = new FileInputStream(propFile);
    String RE = "RE";
```

```

try {
    m_Props = new Properties();
    m_Props.load(is);
    // RE achema access info
    m_proxyName = m_Props.getProperty(PROXYNAME);
    // Recommendation Engine database details
    m_reUrl = m_Props.getProperty(addStr(RE, URL));
    m_reUserName = m_Props.getProperty(addStr(RE, USERNAME));
    m_rePassword = m_Props.getProperty(addStr(RE, PSWORD));

    // Input customer table
    m_inCustTable = getLoc(INPUT);
    // Input items table
    m_inItemTable = getLoc(ITEMTAB);
    // Customer profile table name
    m_custProfile = m_Props.getProperty(CUSTPROFILE);
} catch (SecurityException se) {
    System.err.println("Can't read the property file: " + propFile + "!");
    m_Props = null;
}
}
/*
** Read number of items from the property file.
**
*/
static private int getNumOfItems(String sPriX) {
    return (Integer.parseInt(m_Props.getProperty(addStr(sPriX, DOT, NUMOFITEMS))));
}
/*
** Read location object from the property file.
**
*/
static private Location getLoc(String sPriX) throws StringTooLargeException{
    String sUrl = null;
    String sAlias = null;
    String sTable = null;
    String sSchema = null;
    String sPsword = null;
    String sUsname = null;
    Enumeration propNames = m_Props.propertyNames();
    boolean bFound = false;

    while (propNames.hasMoreElements()) {
        String name = (String)propNames.nextElement();
        if (name.startsWith(sPriX)) {
            // Url
            if (name.endsWith(URL)) {

```

```
        sUrl = m_Props.getProperty(name);
        bFound = true;
    }
    // Alias
    if (name.endsWith(ALIAS))
        sAlias = m_Props.getProperty(name);
    // Schema
    if (name.endsWith(SCHEMA))
        sSchema = m_Props.getProperty(name);
    // Table
    if (name.endsWith(TABLE))
        sTable = m_Props.getProperty(name);
    // Username
    if (name.endsWith(USNAME))
        sUsname = m_Props.getProperty(name);
    // Password
    if (name.endsWith(PSWORD))
        sPsword = m_Props.getProperty(name);
    }
}
return (bFound ? new Location(sAlias, sUrl, sSchema, sTable, sUsname, sPsword) : null);
}
/*
** Read item details from the property file.
**
*/
static private Item getItem(String sPriX) throws StringTooLargeException, InvalidIDException,
NullParameterE
xception{
    String sType = null;
    long lID = 0;
    Enumeration propNames = m_Props.propertyNames();
    boolean bFound = false;

    while (propNames.hasMoreElements()) {
        String name = (String)propNames.nextElement();
        if (name.startsWith(sPriX)) {
            // ID
            if (name.endsWith(ITEMID)) {
                lID = Long.parseLong(m_Props.getProperty(name));
                bFound = true;
            }
            // Type
            if (name.endsWith(ITEMTYPE))
                sType = m_Props.getProperty(name);
        }
    }
}
```



```

        return (bFound ? new Item(sType, IID) : null);
    }
    /*
    ** Read tuning settings from the property file.
    **
    */
    static private TuningSettings getTuning(String sPri) throws NullPointerException {
        Enumeration propName = m_Props.propertyNames();
        Enum.DataSourceType dsType = null;
        Enum.InterestDimensionType iDimension = null;
        Enum.PersonalizationIndexType pIndex = null;
        Enum.ProfileDataBalanceType pdBalance = null;
        Enum.ProfileUsageType pUsage = null;

        while (propName.hasMoreElements()) {
            String name = (String)propName.nextElement();
            if (name.startsWith(sPri)) {
                // datasource type
                if (name.endsWith(DSTYPE)) {
                    String sDsType = m_Props.getProperty(name);
                    dsType = Enum.DataSourceType.getType(sDsType);
                }
                // interest dimension
                if (name.endsWith(INTDIMENSION)) {
                    String sIntDim = m_Props.getProperty(name);
                    iDimension = Enum.InterestDimensionType.getType(sIntDim);
                }
                // personalization index
                if (name.endsWith(PINDEX)) {
                    String spIndex = m_Props.getProperty(name);
                    pIndex = Enum.PersonalizationIndexType.getType(spIndex);
                }
                // profiledata balance
                if (name.endsWith(PDBALANCE)) {
                    String sPdBalans = m_Props.getProperty(name);
                    pdBalance = Enum.ProfileDataBalanceType.getType(sPdBalans);
                }
                // profile usage
                if (name.endsWith(PUSAGE)) {
                    String spUse = m_Props.getProperty(name);
                    pUsage = Enum.ProfileUsageType.getType(spUse);
                }
            }
        }
        return (new TuningSettings(dsType, iDimension, pIndex, pdBalance, pUsage));
    }
    /*

```

```
** Read list of categories from the property file.
**
*/
static private long [] getCatList(String str) {
    StringTokenizer stz = new StringTokenizer(str, "-");
    int nE = stz.countTokens();
    if (nE < 1)
        return null;

    long [] lAry = new long [nE];
    int n = 0;

    while (stz.hasMoreTokens()) {
        String tmp = stz.nextToken();
        tmp.trim();
        lAry[n++] = Long.parseLong(tmp);
    }
    return lAry;
}
/*
** Read filtering settings from the property file.
**
*/
static private FilteringSettings getFilter(String sPriX)
throws Exception {
    boolean bFound = false;
    Enumeration propNames = m_Props.propertyNames();
    int nTaxoID = 0;
    long [] catList = null;
    Enum.CategoryMembershipType cmType = null;

    while (propNames.hasMoreElements()) {
        String name = (String)propNames.nextElement();
        if (name.startsWith(sPriX)) {
            // Taxonomy
            if (name.endsWith(TAXONOMY))
                nTaxoID = Integer.parseInt(m_Props.getProperty(name));
            if (name.endsWith(CATEGORYS)) {
                String sCList = m_Props.getProperty(name);
                catList = getCatList(sCList);
            }
            if (name.endsWith(CATNODE)) {
                bFound = true;
                String sCmType = m_Props.getProperty(name);
                cmType = Enum.CategoryMembershipType.getType(sCmType);
            }
        }
    }
}
```

```
    }  
    return bFound ? FilteringSettings.setFilteringSettings(nTaxoID, catList, cmType) : null;  
}  
  
protected static String getFunctionName(String trace) {  
    // now go through the string to find what you want  
    trace = trace.substring(trace.indexOf('¥n')).trim();  
    trace = trace.substring(trace.indexOf(' ') + 1);  
    //trace = trace.substring(0, trace.indexOf(''));  
    return (trace);  
}  
  
}
```

索引

A

API の構成, 1-2

C

CategoryMembership, 7-3

CategoryMembership インタフェース, 3-3

ContentItem クラス, 3-9

D

DataItem クラス, 3-10, 7-7

DataSource インタフェース, 3-4, 7-3

E

EnumType インタフェース

RE Batch, 7-2

REAPI, 3-2

Enum インタフェース, 3-2, 7-2

F

FilteringSettings クラス, 3-10, 7-8

Filtering インタフェース, 3-4

I

IdentificationData クラス, 3-11

InterestDimension インタフェース, 3-5, 7-4

ItemDetailData クラス, 3-12

Item クラス, 3-12, 7-9

J

Javadoc, 1-2

Java アプリケーション

スタンドアロン, 5-6

Java サーバーサイド・モジュール, 5-6

JVM

プロキシとの相互作用, 5-5

L

Location クラス, 7-9

O

OP データ・キャッシング, 2-7

OP プログラム

実行方法, 1-2

OP プログラムの実行, 1-2

OP レーティング, 3-2

P

PersonalizationIndex インタフェース, 3-5, 7-5

ProfileDataBalance

インタフェース, 7-5

ProfileDataBalance インタフェース, 3-6

ProfileUsage インタフェース, 3-6, 7-6

ProxyBatch, 9-1

R

RE Batch API, 6-1

環境, 6-3

顧客, 6-2

- 顧客プロフィール, 6-4
- 使用, 6-2
- データ, 6-4
- リコメンデーション, 6-2, 6-4
- リコメンデーションの取得, 6-4
- 例, 6-3
- RE Batch API 前提条件, 6-2
- RE Batch API の使用, 6-2
- RE Batch プロキシ, 8-2
- REAPI
 - 使用, 4-1
 - プロキシ・オブジェクト, 2-5
- REAPI Demo, 5-2
- REAPI EnumType インタフェース, 3-2
- REAPI エンド・ユーザー, 2-2
- REAPI 概要, 2-1
- REAPI クラス
 - 位置, 3-2
- REAPI クラスとメソッド, 4-1
- REAPI クラスの位置, 3-2
- REAPI サポート・クラス, 2-6, 3-1
- REAPI サンプル・プログラム, A-1
- REAPI セッション, 2-2
 - 開始, 2-6
 - 終了, 2-9
- REAPI 前提条件, 2-2, 2-4
- REAPI データ収集, 2-3
- REAPI によるリコメンデーション, 2-4, 2-7, 2-8
 - 作成, 2-8
 - 取得, 2-7
- REAPI の基本的な使用方法, 5-2
- REAPI の使用方法, 5-1
- REAPI の例, 5-1
- REAPI プログラムの実行, 2-2, 2-4
- REAPI ホット・ピックス, 2-4
- REBatchProxy
 - 作成, 6-4
- REBatchProxy オブジェクト
 - 削除, 6-5
- RecommendationAttribute インタフェース, 3-7
- RecommendationList クラス, 3-13
- Recommendation クラス, 3-12
- REProxy
 - 複数インスタンス, 5-7
- REProxyBatch
 - 規則, 8-4
 - コード・サンプル, 9-2

- 使用方法, 9-1
- 人口統計データ, 9-4
- リコメンデーション, 8-4
- 例, 9-1
- REProxyBatch クラス
 - 位置, 8-2
- REProxyBatch サンプル・プログラム, B-1
- REProxyBatch の概要, 8-2
- REProxyManager
 - JVM との相互作用, 5-5
- REProxyManager クラス, 4-3
- REProxyRT
 - クラスの位置, 4-2
- REProxyRT オブジェクト, 2-5
 - 削除, 2-9
- REProxy オブジェクト
 - 作成, 5-2
 - 使用, 5-3
- RE データ収集, 4-3
- RE の使用方法, 5-10
 - REProxyBatch, 9-3
- RE バッチ・リコメンデーション, 6-5
- RE プロキシ, 4-2
 - 使用方法, 4-6

S

- Sorting インタフェース, 3-8, 7-7

T

- TuningSettings クラス, 3-13, 7-10

U

- User インタフェース, 3-8

あ

- インタフェース
 - CategoryMembership, 3-3, 7-3
 - DataSource, 3-4, 7-3
 - Filtering, 3-4
 - InterestDimension, 3-5, 7-4
 - PersonalizationIndex, 3-5, 7-5
 - ProfileDataBalance, 3-6, 7-5
 - ProfileUsage, 3-6, 7-6

- RecommendationAttribute, 3-7
- Sorting, 3-8, 7-7
- User, 3-8
- エンド・ユーザー
 - REAPI, 2-2

か

- 規則, 4-6
 - REProxyBatch, 8-4
- キャッシュ
 - データ, 4-3
- キャッシング
 - データ, 2-7
- クラス
 - Contentitem, 3-9
 - DatalItem, 3-10, 7-7
 - FilteringSettings, 3-10, 7-8
 - IdentificationData, 3-11
 - Item, 3-12, 7-9
 - ItemDetailData, 3-12
 - Location, 7-9
 - REAPI, 4-1
 - Recommendation, 3-12
 - RecommendationList, 3-13
 - REProxyBatch, 8-2
 - REProxyManager, 4-3
 - TuningSettings, 3-13, 7-10
- コード・サンプル
 - REProxyBatch, 9-2
- 顧客, 2-2, 8-2
 - RE Batch API, 6-2, 6-4
- 顧客登録, 4-4
- 顧客プロフィール管理
 - REProxyBatch, 8-2
- 個別のリコメンデーション, 5-9

さ

- サーバーサイド・モジュール, 5-6
- サポート・クラス
 - RE Batch, 7-1
 - REAPI, 3-1
- サンプル・プログラム
 - REAPI, A-1
 - REProxyBatch, B-1
- 初期化

- RE プロキシ, 5-7
- 使用方法
 - REAPI, 5-1, 5-2
 - RE プロキシ, 4-6
- 時間ベースのアイテム, 5-11, 9-5
- 時間ベースのアイテムの処理, 5-11, 9-5
- 人口統計データ
 - 使用, 9-4
- スコアリング
 - RE Batch API, 6-5
 - 顧客, 2-8
 - 匿名ユーザー, 2-8
- スタンドアロン Java アプリケーション, 5-6
- ステートフル, 2-2
- ステートレス, 2-2
- セッション, 2-2
 - 管理, 4-4
 - 終了, 2-9
- セッション管理, 4-4
- セッション対応型, 2-2, 2-3, 5-4
- セッション対応型の Web アプリケーション, 5-4
- セッションの作成, 4-6
- 前提条件
 - RE Batch API, 6-2

た

- 抱合せ販売, 4-8
 - 使用方法, 8-3
- 中断されない REAPI サービス, 5-8
- チューニング, 3-13
- 通貨
 - 処理, 5-9, 9-4
- 通貨の処理, 5-9, 9-4
- データ
 - RE Batch API, 6-4
 - データ管理, 4-4
 - データ・キャッシング, 2-7, 4-3
 - データ収集, 2-3, 4-3, 4-4, 4-6
 - 匿名ユーザー, 2-2

は

- はじめに, vii
- 表記規則表, x
- バッチ
 - 使用, 9-1

- 例, 9-1
- バッチ・プロキシ, 6-1
- バッチ・リコメンデーション, 8-4
- 非セッション対応型, 2-2, 2-3, 5-5
- 非セッション対応型の Web アプリケーション, 5-5
- 表記規則
 - 表, x
- フィルタリング, 3-10, 7-8
- 複数インスタンス
 - REProxy, 5-7
- 複数通貨, 5-9, 9-4
- プロキシ, 8-2
 - 管理, 4-2, 8-2
 - 作成, 4-2, 8-2
 - 初期化, 5-7
 - 使用, 5-3
 - 破棄, 5-3
- プロキシ・オブジェクト
 - REAPI, 2-5
 - 削除, 2-9
- プロキシの管理, 8-2
- プロキシの作成, 4-7, 8-2
- プロキシの使用, 5-3
- プロキシの破棄, 4-9, 5-3
- ホット・ピックス
 - REAPI, 2-4

ら

- リコメンデーション, 3-12, 4-6
 - RE Batch API, 6-2, 6-5
 - REAPI, 2-4, 4-4
 - REProxyBatch, 8-2, 8-4
 - 作成, 2-8, 6-5
 - 使用方法, 8-4
 - 戻り値, 8-3
- リコメンデーション・エンジンの使用方法, 5-10
 - REProxyBatch, 9-3
- 例
 - REAPI, 5-1
- レーティング, 8-3
 - OP, 3-2
- ロード・バランシング
 - REAPI, 5-9