IBM WebSphere Information Integrator
OmniFind Edition

エンタープライズ・サーチ
プログラミング・ガイドおよび API リファレンス

バージョン 8.3

SD88-6375-02
(英文原典：SC18-9284-02)

IBM WebSphere Information Integrator
OmniFind Edition

エンタープライズ・サーチ
プログラミング・ガイドおよび API リファレンス

バージョン 8.3

SD88-6375-02
(英文原典：SC18-9284-02)

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

本書には、IBM の専有情報が含まれています。その情報は、使用許諾条件に基づき提供され、著作権により保護されています。本書に記載される情報には、いかなる製品の保証も含まれていません。また、本書で提供されるいかなる記述も、製品保証として解釈すべきではありません。

IBM の資料は、オンラインまたは最寄りの事業所の IBM 担当員を通じて注文できます。

- オンラインで資料を注文する場合は、IBM Publications Center (www.ibm.com/shop/publications/order) にアクセスしてください。
- 最寄りの IBM 担当員をお探しになる場合は、IBM Directory of Worldwide Contacts (www.ibm.com/planetwide) にアクセスしてください。

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

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

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

原　典： SC18–9284–02
IBM WebSphere Information Integrator
OmniFind Edition
Programming Guide and API Reference for Enterprise Search
Version 8.3

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.11

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

第 1 章 エンタープライズ・サーチ API

IBM® WebSphere® Information Integrator OmniFind™ Edition (WebSphere II OmniFind Edition) では、エンタープライズ・サーチ用の Java™ アプリケーション・プログラミング・インターフェース (API) が豊富に用意されているので、検索アプリケーションをカスタマイズしたり、クロールする文書を変更することができます。

IBM 検索および索引 API

検索および索引アプリケーション・プログラミング・インターフェースを使用して、カスタム検索アプリケーションを作成します。検索および索引 API のエンタープライズ・サーチのインプリメンテーションにより、検索サーバーにリモートでアクセスすることができます。検索サーバーは、エンタープライズ・サーチ・システムのコレクション・データを保管します。ユーザーは、これらの API を使用して、検索要求のサブミット、検索結果の処理、分類法ツリーのブラウズを行うアプリケーションを作成することができます。

また、検索および索引 API を使用して、コレクションを管理したり、索引を検索可能にする管理アプリケーションを作成することもできます。管理アプリケーションを使用して、チーム・ルームやポートレットのコレクションを管理できます。また、索引付け製品をマイグレーションして、コレクションの管理、文書の追加、エンタープライズ・サーチ・コレクションの検索を行うこともできます。

サンプル検索アプリケーション ESSearchApplication

検索および索引 API を使用してカスタマイズ検索アプリケーションを作成することができますが、 WebSphere II OmniFind Edition で提供されているサンプル検索アプリケーションを使用することもできます。ESSearchApplication 検索アプリケーションを使用すると、検索するコレクションの選択、そのコレクションの照会、照会結果の構成などのエンタープライズ検索コレクションに関する基本的な検索タスクを実行することができます。

クローラー・プラグイン API

文書のクロール後、文書を解析して索引付けを行い検索可能にする前に文書を変更する場合は、クローラー・プラグイン API を使用して、文書または文書のメタデータの情報を追加、変更、削除できます。また、文書を無視 (スキップ) して索引付けしないように指定することもできます。

クローラー・プラグイン API は、検索および索引 API (SIAPI) の一部ではありません。

データ・リスナー API

現在、データ・リスナー API の使用は推奨されていません。検索および索引管理 API を使用して、コレクションへの追加またはコレクションからの削除を行ってください。

データ・リスナーは、クライアント・アプリケーションからの要求を受け入れる、エンタープライズ・サーチ・コンポーネントです。要求を受けて、コレクションにデータを追加したり、コレクションからデータを除去します。データ・リスナー・クライアント・アプリケーションを使用して、コレクションにページを追加したり、データ・ソースのクロールを待たずにコレクションから Uniform Resource Identifier (URI) を除去したり、コレクションの Web クローラーに対して Uniform Resource Locator (URL) へのアクセスまたは再アクセスを指示します。

データ・リスナー API は、検索および索引 API (SIAPI) の一部ではありません。

検索および索引 API のセキュリティー

検索および索引 API は、WebSphere II OmniFind Edition がインストールされている場合に、各 WebSphere 検索ノードにインストールされている ESSearchServer エンタープライズ・アプリケーションと HTTP を介してリモートで通信します。

WebSphere Application Server のグローバル・セキュリティーを使用可能に設定して、遠隔通信を保護する必要があります。アプリケーションが機密保護機能のあるリモート検索および索引 API 要求を発行する場合は、WebSphere が認証用に使用するエンタープライズ・ユーザー・レジストリーに保管されている正当なユーザー名で、サービス・クラスにユーザー名とパスワードを設定する必要があります。正当なユーザー名とパスワードが含まれていない要求はリジェクトされます。WebSphere 管理コンソールで、ユーザー名とパスワードを追加することができます。

エンタープライズ・サーチ・アプリケーションでは、 Properties オブジェクトが getSearchService メソッドまたは getBrowseService メソッドの呼び出しで渡されます。 Properties オブジェクトは、WebSphere 用に username と password と呼ばれるプロパティー名を指定します。

エンタープライズ・サーチは、HTTP BASIC 認証をサポートします。HTTPS (SSL v2 または v3) はサポートされていません。

アプリケーションには ID が必要です。また、コレクションにも ID が必要です。特定のコレクションにアクセスする必要があるアプリケーションの場合、コレクション ID をアプリケーション ID と関連付けておかなければなりません。エンタープライズ・サーチ管理コンソールで、特定のコレクションにアクセスする権限をアプリケーションに付与することができます。

検索アプリケーションおよびデータ・リスナー・アプリケーションのサンプルのコンパイル

エンタープライズ・サーチの ESSearchApplication サンプル、データ・リスナー・サンプル、検索および索引 API コードは、IBM Software Developer's Kit 1.4.x を使用してコンパイルする必要があります。IBM Software Developer's Kit 1.5 はサポートされていません。

Java アプリケーションを作成する前に、Java ベースのビルド・ツールである Apache ANT をインストールして構成する必要があります。Apache ANT のインストールおよび構成方法について詳しくは、 http://ant.apache.org/ を参照してください。

ES_INSTALL_ROOT/samples ディレクトリーにある ESSearchApplication は、JRE バージョン 1.4 環境で実行する必要があります。WebSphere Application Server バージョン 5.1 と WebSphere Portal バージョン 5.1 は、両方とも JRE バージョン 1.4 を提供しています。管理アプリケーションをコンパイルするには、2 ページの『検索アプリケーションおよびデータ・リスナー・アプリケーションのサンプルのコンパイル』を参照してください。

検索アプリケーションまたはデータ・リスナー・アプリケーションをコンパイルするには、次のようにします。

1. コマンド行で、以下のいずれかのディレクトリーに変更します。

   サンプル検索アプリケーション

   ES_INSTALL_ROOT/samples/siapi:

   - AIX®、Linux®、 Solaris: /opt/IBM/es/samples/siapi
   - Windows®: C:¥Program Files¥IBM¥es¥samples¥siapi

   ESSearchApplication 検索アプリケーション

   ES_INSTALL_ROOT/samples/ESSearchApplication:

   - AIX、Linux、 Solaris: opt/IBM/es/samples/ESSearchApplication
   - Windows: C:¥Program Files¥IBM¥es¥samples¥datalistener

   データ・リスナー・クライアント・アプリケーション

   ES_INSTALL_ROOT/samples/datalistener:

   - AIX、Linux、 Solaris: /opt/IBM/es/samples/datalistener
   - Windows: C:¥Program Files¥IBM¥es¥samples¥ESSearchApplication

   いずれのディレクトリーにも、ANT がファイルのビルド時に使用する `build.xml` ファイルが含まれています。
2. `ant` と入力して Enter キーを押します。

Java ソース・コードのコンパイル後、以下のメッセージが表示されます。

```
BUILD SUCCESSFUL
Total time: xx seconds
```

関連タスク

103 ページの『サンプル管理アプリケーションのコンパイル』
サンプル管理アプリケーションは、Ant スクリプトを実行してコンパイルします。

4 ページの『管理 API 用のクライアント・ツールキットのインストール』
管理アプリケーションを作成するには、アーカイブ・ファイル es.siapi.toolkit.jar をインストールする必要があります。このファイルには、必要な Java パッケージ、 サンプル・アプリケーション、ビルド・スクリプト、Javadoc 情報が含まれています。

関連資料

91 ページの『サンプル検索アプリケーションのコンパイル』
サンプル検索アプリケーションは、Ant スクリプトを実行してコンパイルします。

管理 API 用のクライアント・ツールキットのインストール

管理アプリケーションを作成するには、アーカイブ・ファイル es.siapi.toolkit.jar をインストールする必要があります。このファイルには、必要な Java パッケージ、サンプル・アプリケーション、ビルド・スクリプト、Javadoc 情報が含まれています。

クライアント・ツールキットをインストールするには、次のようにします。

1. ご使用のオペレーティング・システム用のクライアント・ツールキット JAR ファイルを検索するか、または検索管理者に問い合わせて、その JAR ファイルを入手します。
   - AIX、Linux、Solaris: *ES_INSTALL_ROOT*/lib/es.siapi.toolkit.jar
   - Windows: *ES_INSTALL_ROOT*¥lib¥es.siapi.toolkit.jar
2. 以下のコマンドを実行して、JAR ファイルを抽出します。

   ```
   jar -xvf es.siapi.toolkit.jar
   ```
3. クライアント構成ファイル es.client.cfg を開発環境に保管します。
   - AIX、Linux、Solaris: *ES_NODE_ROOT*/nodeinfo/es.client.cfg
   - Windows: *ES_NODE_ROOT*¥nodeinfo¥es.client.cfg

   これは、API がエンタープライズ・サーチ・サーバーにアクセスする際に必要な構成ファイルです。構成ファイルの絶対パスをシステム引数として Java 実行可能ファイルに渡します。

関連タスク

2 ページの『検索アプリケーションおよびデータ・リスナー・アプリケーションのサンプルのコンパイル』
エンタープライズ・サーチの ESSearchApplication サンプル、データ・リスナー・サンプル、検索および索引 API コードは、IBM Software Developer's Kit 1.4.x を使用してコンパイルする必要があります。IBM Software Developer's Kit 1.5 はサポートされていません。

検索および索引 API Javadoc 文書

WebSphere II OmniFind Edition に添付されている検索および索引 API Javadoc 文書は、ご使用のエンタープライズ・サーチ・ソリューションにデプロイするカスタム検索および管理アプリケーションを作成する際に役立ちます。

検索アプリケーションを作成する際に使用できるアプリケーション・プログラミング・インターフェース (API) のリストは、Javadoc 文書を参照してください。この文書には、データ・リスナー・クライアント・アプリケーション用の API は含まれていません。

Javadoc 文書には、クローラー・プラグイン用の API は含まれていません。各クローラー・プラグイン API について詳しくは、このガイドの該当するトピックまたはインフォメーション・センターを参照してください。

現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、管理アプリケーションを作成してください。

Javadoc 文書は、 *ES_INSTALL_ROOT*/docs/api/siapi ディレクトリーにあります。

関連概念

7 ページの『第 2 章 検索および索引 API の概要』
IBM 検索および索引 API は、ユーザーによるコレクションおよび分類法の検索、ブラウズ、管理を可能にするプログラミング・インターフェースです。

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

関連資料

10 ページの『検索アプリケーションの構造』
検索アプリケーションは、エンタープライズ・サーチ・コレクションにアクセスし、照会を実行し、 照会結果を処理します。

第 2 章 検索および索引 API の概要

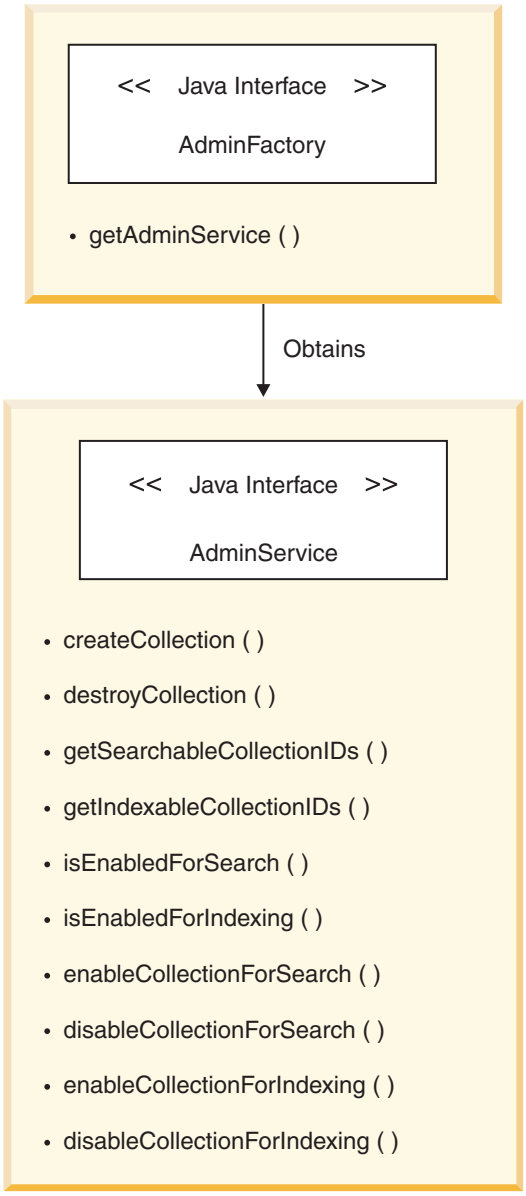
IBM 検索および索引 API は、ユーザーによるコレクションおよび分類法の検索、ブラウズ、管理を可能にするプログラミング・インターフェースです。

検索および索引 API は、検索エンジンのさまざまなインプリメンテーションを可能にするファクトリー・ベースのインターフェースです。検索および索引 API を使用することにより、ご使用の検索および索引 API アプリケーションを変更せずに、ユーザーの検索アプリケーションで IBM より提供されるさまざまな検索エンジンを使用することができます。例えば、ポータル検索エンジンを使用する WebSphere Portal で検索および索引 API アプリケーションを作成する場合、ご使用の検索アプリケーションを変更せずに、WebSphere II OmniFind Edition エンタープライズ検索エンジンを使用できます。

検索および索引 API は、以下のタイプの検索および管理タスクをサポートします。

- 検索アプリケーション・タスク:
  - 索引の検索
  - 検索結果セットに戻される情報のカスタマイズ
  - 分類法の検索および参照
  - 複数のコレクションを 1 つのコレクションであるかのように検索 (サーチ・フェデレーション)
  - URI のクリックによる結果の表示およびスコア情報 (ランキング) の表示
  - 広範囲のエンタープライズ・データ・ソース (WebSphere Information Integrator Content Edition リポジトリーや Lotus Notes® データベースなど) からの文書の検索および取得
- 管理アプリケーション・タスク:
  - ユーザー・アプリケーション ID の管理
  - コレクションの作成または破棄
  - 検索および索引に対するコレクションの使用可能化または使用不可化
  - コレクションの作成および再編成
  - コレクションへの文書の追加またはコレクションからの文書の除去

以下の図は、検索および索引 API 間の関係を示しています。最初の図は、管理 API を示しています。



次の図は、索引 API を示しています。

次の図は、検索 API を示しています。

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

4 ページの『検索および索引 API Javadoc 文書』
WebSphere II OmniFind Edition に添付されている検索および索引 API Javadoc 文書は、ご使用のエンタープライズ・サーチ・ソリューションにデプロイするカスタム検索および管理アプリケーションを作成する際に役立ちます。

検索アプリケーションの構造

検索アプリケーションは、エンタープライズ・サーチ・コレクションにアクセスし、照会を実行し、照会結果を処理します。

検索および索引 API を使用して検索アプリケーションを作成するには、次の汎用ステップに従います。

1. SearchFactory オブジェクトのインプリメンテーションをインスタンス化します。

これにより、SearchFactory を使用して、SearchService オブジェクトを取得できるようになります。

2. SearchService オブジェクトを取得します。

   SearchService オブジェクトは、検索エンジンとのやりとりに必要な接続情報を使用して構成されます。

3. 検索可能オブジェクトを取得します。

   SearchService オブジェクトを取得後、そのオブジェクトを使用して 1 つ以上の検索可能オブジェクトを取得することができます。各検索および索引 API 検索可能オブジェクトは、1 つのエンタープライズ・サーチ・コレクションに関連付けられます。 SearchService オブジェクトを使用して、federator オブジェクトを取得することもできます。 federator オブジェクトは、特殊な種類の Searchable オブジェクトで、複数の Searchable オブジェクト (コレクション) にわたって同時に単一の照会を実行することができます。

4. 照会を実行します。

   検索アプリケーションは、検索サーバー上の検索ランタイムに検索照会を渡します。

5. 照会結果を処理します。

   ResultSet インターフェース・オブジェクトおよび Result インターフェース・オブジェクトを使用して照会を処理します。検索および索引 API には、ResultSet インターフェースや個々の Result インターフェース・オブジェクトと対話するためのさまざまなメソッドがあります。

検索および索引 API は、ファクトリー・ベースの Java API です。検索アプリケーションで使用するすべてのオブジェクトは、検索および索引 API オブジェクト・ファクトリー・メソッドを呼び出して作成されるか、またはファクトリー生成オブジェクトのメソッドを呼び出すことによって戻されます。異なるファクトリーをロードすることによって、検索および索引 API インプリメンテーション間を容易にスイッチすることができます。

WebSphere II OmniFind Edition の検索および索引 API インプリメンテーションは、com.ibm.es.api.search.RemoteSearchFactory クラスにより提供されます。

以下の検索および索引 API パッケージを使用して、検索アプリケーションを作成します。

com.ibm.siapi
: ルート・パッケージ

com.ibm.siapi.browse
: 分類法ブラウズ・インターフェースが含まれています

com.ibm.siapi.common
: 共通 SIAPI インターフェース

com.ibm.siapi.search
: コレクション検索用のインターフェース

SearchFactory オブジェクトの取得

検索および索引 API アプリケーションを作成するには、以下の例のように、SearchFactory オブジェクトのインプリメンテーションを取得します。

```
SearchFactory factory =
SiapiSearchImpl.createSearchFactory
    ("com.ibm.es.api.search.RemoteSearchFactory");
```

SearchService オブジェクトの取得

SearchService オブジェクトを取得するには、 SearchFactory オブジェクトを使用します。 SearchService オブジェクトを使用して、検索可能なコレクションにアクセスすることができます。

検索サーバーのホスト名、ポート、および WebSphere グローバル・セキュリティーが使用可能な場合は有効な WebSphere ユーザー名とパスワードを指定して、SearchService を構成します。

構成パラメーターは java.util.Properties オブジェクトに設定されます。パラメーターは、 SearchService オブジェクトを生成する getSearchService ファクトリー・メソッドに渡されます。以下の例は、SearchService オブジェクトを取得する方法を示しています。

```
Properties configuration = new Properties();
configuration.setProperty("hostname", "es.mycompany.com");
configuration.setProperty("port", "80");
config.setProperty("username", "websphereUser");
config.setProperty("password", "webspherePassword");
SearchService searchService =
 factory.getSearchService(config);
```

Searchable オブジェクトの取得

Searchable オブジェクトを取得するには、 SearchService オブジェクトを使用します。 Searchable オブジェクトは、検索可能なコレクションと関連しています。Searchable オブジェクトを使用して、照会を実行し、関連したコレクションに関する情報を取得することができます。各エンタープライズ・サーチ・コレクションには ID があります。

ユーザーが、 Searchable オブジェクトを要求する際には、アプリケーション ID を使用して、アプリケーションを識別する必要があります。適切なアプリケーション ID については、エンタープライズ・サーチ管理者に問い合わせてください。

以下の例は、Searchable オブジェクトを取得する方法を示しています。

```
ApplicationInfo appInfo = factory.createApplicationInfo
("my_application_id","my_password");
Searchable searchable =
 searchService.getSearchable(appInfo, "some_collection_id");
```

アプリケーションで使用可能なすべての Searchable オブジェクトを取得するには、getAvailableSearchables メソッドを呼び出します。

```
Searchable[] searchables =
 searchService.getAvailableSearchables(appInfo);
```

照会の実行

Searchable オブジェクトを取得したら、その Searchable オブジェクトに対して照会を実行します。 Searchable オブジェクトに照会を実行するには、次のようにします。

1. Query オブジェクトを作成します。
2. Query オブジェクトをカスタマイズします。
3. Searchable オブジェクトに対して Query オブジェクトを実行します。
4. ResultSet オブジェクトに示されている照会結果を取得します。

以下の例は、照会を実行する方法を示しています。

```
String queryString = "big apple";
Query query = factory.createQuery(queryString);
query.setRequestedResultRange(0, 10);
ResultSet resultSet = searchable.search(query);
```

照会結果の処理

以下の例のように、ResultSet インターフェースおよび Result インターフェースを使用して照会結果を処理することができます。

```
Result[] results = resultSet.getResults();
for ( int i = 0 ; i < results.length ; i++ ) {
 System.out.println
( "Result " + i + ": " + results[i].getDocumentID()
  + " - " + results[i].getTitle() );
}
```

関連概念

『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

4 ページの『検索および索引 API Javadoc 文書』
WebSphere II OmniFind Edition に添付されている検索および索引 API Javadoc 文書は、ご使用のエンタープライズ・サーチ・ソリューションにデプロイするカスタム検索および管理アプリケーションを作成する際に役立ちます。

管理アプリケーションの構造

検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

共通の検索および索引 API は、内部のエンタープライズ・サーチ API と連動します。検索および索引 API 管理アプリケーションは、以下のタスクを行います。

- アプリケーション ID の登録
- アプリケーション ID のインスタンスの作成
- コレクションの管理
  - コレクションの作成または破棄
  - コレクションへの文書の追加またはコレクションからの文書の除去
- 索引のリフレッシュまたは再編成
- 索引または検索に対するコレクションの使用可能化または使用不可化

- アプリケーション ID の登録抹消

管理アプリケーションは、サービス・クラスから索引オブジェクトを検索します。管理アプリケーションは、検索および索引 API の管理パッケージを使用して、コレクションを作成または破棄したり、コレクションを索引あるいは検索に対して使用可能または使用不可に設定します。次に、管理アプリケーションは、検索および索引 API の索引パッケージを使用して、 1 つ以上の索引オブジェクトを取得します。ユーザーは、その索引オブジェクトで、文書の追加または削除、あるいは追加文書による索引の作成や再編成などの操作を行うことができます。

索引を作成して検索に使用できるように設定したら、検索アプリケーションで索引を検索できます。

コレクションを作成または破棄したり、コレクションを索引および検索に対して使用可能または使用不可に設定するには、com.ibm.siapi.admin パッケージを使用します。

コレクションに文書を追加したり、コレクションから文書を削除したり、索引をリフレッシュまたは再編成するには、com.ibm.siapi.index パッケージを使用します。

関連概念

7 ページの『第 2 章 検索および索引 API の概要』
IBM 検索および索引 API は、ユーザーによるコレクションおよび分類法の検索、ブラウズ、管理を可能にするプログラミング・インターフェースです。

4 ページの『検索および索引 API Javadoc 文書』
WebSphere II OmniFind Edition に添付されている検索および索引 API Javadoc 文書は、ご使用のエンタープライズ・サーチ・ソリューションにデプロイするカスタム検索および管理アプリケーションを作成する際に役立ちます。

関連タスク

16 ページの『アプリケーション ID のインスタンスの作成』
各クライアント・アプリケーションは、アプリケーション ID によって識別されます。 AdminFactory ファクトリーを使用して、アプリケーション ID を作成します。

17 ページの『コレクションの作成または破棄』
コレクションは、初期アプリケーション ID を使用して作成されます。このアプリケーション ID のみが、指定されたコレクションの削除を許可されます。コレクションは、 addCollectionToApplication メソッドを使用して、他のアプリケーション ID と関連付けることができます。

18 ページの『コレクションへの文書の追加』
コレクションに文書を追加するには、IndexFactory オブジェクトを使用します。

20 ページの『索引の再編成およびリフレッシュ』
索引に文書を追加したり、索引から文書を削除した後は、索引をリフレッシュおよび 再編成する必要があります。

21 ページの『索引を検索で使用可能にする』
索引に文書を設定したら、索引を検索で使用できるようにします。これにより、検索アプリケーションが索引付き文書を検索することができます。

関連資料

10 ページの『検索アプリケーションの構造』
検索アプリケーションは、エンタープライズ・サーチ・コレクションにアクセスし、照会を実行し、 照会結果を処理します。

アプリケーション ID の登録

アプリケーション ID のインスタンスを作成する前に、アプリケーション ID をエンタープライズ・サーチ・サーバーに登録します。

アプリケーション ID を登録するには、次のようにします。

1. エンタープライズ・サーチ・サーバーでアプリケーションを作成します。
2. clientAppInfo インスタンスを作成します。
3. サーバーにアプリケーションを登録します。

```
import java.util.Properties;

import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.es.siapi.common.ApplicationInfoImpl;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
* Creates an application on the enterprise search server.
* This sample also describes the way to unregister the application.
*
*/
public class RegisterApplicationID {

public static void main(String[] args) {
try {
AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);
if(factory != null){
AdminService service = factory.getAdminService(null);
if(service != null){

// create the client application info instance
ApplicationInfo clientAppInfo = factory.createApplicationInfo
("SIAPI-client", "password");

// currently privileges are not honored for custom applications
int priv = 0;
// register an application ID on the server
service.registerApplication(null, clientAppInfo, priv);

// to unregister an application, destroy the application info on the server
service.unregisterApplication(clientAppInfo, clientAppInfo.getId());

}
}
} catch (SiapiException e) {
e.printStackTrace();
System.out.println(e.getLocalizedMessage());
}
}
}
```

アプリケーション ID の登録抹消

アプリケーション ID の登録を抹消するには、エンタープライズ・サーチ・サーバー上のアプリケーション情報を破棄します。

アプリケーション ID の登録を抹消するには、次のようにします。

アプリケーション情報 (clientAppInfo オブジェクト) を破棄します。

```
// to unregister an application, destroy the application
// information on the server
service.unregisterApplication(clientAppInfo, clientAppInfo.getId())
```

アプリケーション ID のインスタンスの作成

各クライアント・アプリケーションは、アプリケーション ID によって識別されます。 AdminFactory ファクトリーを使用して、アプリケーション ID を作成します。

まず、アプリケーション ID をエンタープライズ・サーチ・サーバーに登録する必要があります。アプリケーション ID の登録後、アプリケーション ID のインスタンスを検索できます。

認証のために、ユーザー・アプリケーション ID にパスワードを関連付ける必要がありますが、別のクライアント・アプリケーションが同じエンタープライズ・サーチ・アプリケーション ID を使用することができます。エンタープライズ・サーチ・アプリケーション ID は、アプリケーションがアクセス権限を持つコレクションのみを指定します。ユーザーは、エンタープライズ・サーチ・アプリケーション ID によって定義および制御されている同じコレクション・セットへのアクセス権限を付与されている 別のアプリケーションが 2 つ (例えば、財務アプリケーションと人事アプリケーション) 存在することがあります。

コレクション ID を取得する際には、検索管理者に問い合わせてください。

アプリケーション ID を作成するには、次のようにします。

AdminFactory ファクトリーを使用して、アプリケーション ID を作成します。アプリケーション ID はコレクション ID と関連付けることができるので、関連付けたコレクションを管理できます。

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

AdminFactory factory = SiapiAdminImpl.createAdminFactory
( "com.ibm.es.siapi.admin.AdminFactoryImpl");

if(factory != null){
  String appID = "SIAPI-App";
  String appPw = "password";
 ApplicationInfo appInfo = factory.createApplicationInfo(appID,appPw);
 if(appInfo != null){
     System.out.println("Application ID called " + appInfo.getId() + "
      was created successfully." );
 } else {
     System.out.println("Application ID called " + appInfo.getId() + "
```

```
      was not created!!" );
 }
}
. . .
```

エンタープライズ・サーチ管理コンソールで、アプリケーション ID をコレクション ID に関連付けてください。

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

コレクションの作成または破棄

コレクションは、初期アプリケーション ID を使用して作成されます。このアプリケーション ID のみが、指定されたコレクションの削除を許可されます。コレクションは、 addCollectionToApplication メソッドを使用して、他のアプリケーション ID と関連付けることができます。

コレクションを作成または破棄するには、まず、以下の例に示すように、AdminService クラスを呼び出す必要があります。この例は、コレクションを作成する AdminService クラスを示しています。

```
class AdminService {
       void createCollection(ApplicationInfo appInfo,
      java.lang.String collectionID,
      java.lang.String collectionLabel,
      int optimizationMode,
      java.lang.String defaultLanguage,
      java.util.Properties config)
```

java.util.Properties インスタンスを使用して、一連のオプションのプロパティーを渡すこともできます。 ApplicationInfo オブジェクトを使用してコレクションを作成するには、まず、AdminService オブジェクトを作成する必要があります。以下の例では、すべてのオプション・プロパティーを示します。

```
AdminFactory factory = SiapiAdminImpl.createAdminFactory
 (IAdminConstants.ADMIN_FACTORY_IMPL);
  if(factory != null){
   AdminService service = factory.getAdminService(null);
   if(service != null){
    ApplicationInfoImpl appInfo = new ApplicationInfoImpl();
    appInfo.setId("SIAPI-App");
    appInfo.setPassword("search");

    Properties config = new Properties();
    config.setProperty(IAdminConstants.KEY_INDEX_LOCATION,
          "/home/esadmin/siapidata");
    config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_NGRAM,
          "true");
    config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_SECURITY,
          "true");
    config.setProperty(IAdminConstants.KEY_MAX_DOCS_IN_INDEX, "10000");

    // if collection ID can be set to "", system generates a
    // unique id
    String colID = "col_123";
    String colLabel = "SIAPI Client";
    // if language is "", a default language of "en" is associated
    String colLangauge = "en";
```

```
      // identified the ranking model for the collection.
            // DATE_BASED, LINK_BASED and NO_STATIC_RANK are the supported models
      int optimizationMode = IAdminConstants.DATE_BASED;
      adminService.createCollection(appInfo,
        colID,
        colLabel,
        optimizationMode,
        colLangauge,
        config);
     }
    }
....
```

config.setProperty メソッドでは、索引のロケーションを指定します。

コレクションを作成して、そのコレクションを索引に対して使用可能に設定したら、文書を追加することができます。文書の追加後、索引をリフレッシュまたは再編成して、文書を検索に使用できるようにします。indexRef.build API は索引をリフレッシュし、indexRef.reorganize API は索引を再編成するか、またはリフレッシュされた使用可能な索引をメイン索引にマージします。この後、索引は検索に使用できるようになります。

以下の例では、コレクションの破棄方法を示します。コレクションを破棄することにより、ディスク上の索引付き文書もすべて削除されます。

```
class AdminService {
        void destroyCollection(ApplicationInfo appInfo,
 java.lang.String collectionID);

...
AdminService service = factory.getAdminService(null);
if(service != null){
 ApplicationInfoImpl appInfo = new ApplicationInfoImpl();
 appInfo.setId("SIAPI-App");
 appInfo.setPassword("search");
 String colID = "col_123";
 adminService.destroyCollection(appInfo, colID);
}
    }
. . .
```

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

コレクションへの文書の追加

コレクションに文書を追加するには、IndexFactory オブジェクトを使用します。

コレクションに文書を追加するには、次のようにします。

1. 以下の例のように、adminService.isEnabledForIndexing メソッドを使用して、コレクションを索引に対して使用可能に設定します。

```
. . .
boolean enabled = adminService.isEnabledForIndexing(appInfo, colID);
        if(!enabled){
                // enable for indexing
```

```
                System.out.println("Enabling collection for indexing");
                adminService.enableCollectionForIndexing(appInfo, colID, null);
        }
. . .
```

コレクションに文書を追加するには、コレクションを索引に対して使用可能にする必要があります。この機能は、コレクションに関連したパーサー・ドライバーを開始します。

2. データ・ソースから文書を作成します。以下に示す API を使用して、文書を以下のものとして追加します。
   - コンテンツのバイト配列
   - コンテンツの文字配列
   - コンテンツのストリング

   以下の例では、文書の作成方法を示します。

```
class IndexFactory{
        // Create a raw document by its byte array content.
  Document createDocument(java.lang.String documentID, byte[] content,
  java.lang.String documentType, java.lang.String documentSource);

        // Create a raw document by its char array content.
     Document createDocument(java.lang.String documentID, char[] content,
  java.lang.String documentType, java.lang.String documentSource) ;

        // Create a raw document by its string content.
     Document createDocument(java.lang.String documentID,
  java.lang.String content, java.lang.String documentType,
  java.lang.String documentSource) ;


...
IndexFactory indexFactory = SiapiIndexImpl.createIndexFactory
("com.ibm.es.siapi.index.IndexFactoryImpl");
. . .

String documentContent = readDocumentFromDB2
("docid:db2://sample//EmployeePicture");
Document document = indexFactory.createDocument
("docid:db2://sample//EmployeePicture",
documentContent, "DB2 type", "Sample");
```

3. 以下のいずれかの API を使用して、コレクションに文書を追加します。

```
class Index{
        // Add raw document to index
        void addDocument(Document doc);
        // Add raw document with meta data to the index
        void addDocument(Document doc, java.util.HashMap fieldMapping);

IndexService indexService = indexFactory.getIndexService(null);
Index indexRef = indexService.getIndex(appInfo, collectionID);
// add raw document
indexRef.addDocument(document);
```

4. オプション: 索引に文書のメタデータを追加します。メタデータは、照会用語の一部として索引に構成することができる文書内のフィールドのリストです。以下のいずれかの API を使用して、さまざまなコンストラクターでフィールドを作成できます。

```
class IndexFactory{
        Field createField(java.lang.String fieldName, boolean value);
        Field createField(java.lang.String fieldName, boolean[] value);
```

```
        Field createField(java.lang.String fieldName, byte[] value);
        Field createField(java.lang.String fieldName, java.util.Date value);
        Field createField(java.lang.String fieldName, double value);
        Field createField(java.lang.String fieldName, double[] value);
        Field createField(java.lang.String fieldName, int value);
        Field createField(java.lang.String fieldName, int[] value);
        Field createField(java.lang.String fieldName, java.lang.String text);
        Field createField(java.lang.String fieldName, java.lang.String[] value);

// create a searchable/parametric/returnable field
Field myField = indexFactory.createField("empid", "00140");
myField.setFieldSearchable(true);
myField.setParametric(true);
myField.setReturnable (true);

// create a searchable and returnable field
Field myField2 = indexFactory.createField("Format", "image/gif");
myField2.setFieldSearchable(true);
myField2.setParametric(false);
myField2.setReturnable (true);


HashMap fieldMapping= new HashMap();
fieldMapping.put (myField.getID(), myField);
fieldMapping.put (myField2.getID(),myField2);
// add raw document with meta data
indexRef.addDocument(document, fieldMapping);
```

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

索引の再編成およびリフレッシュ

索引に文書を追加したり、索引から文書を削除した後は、索引をリフレッシュおよび再編成する必要があります。

索引を最初に作成する場合には、メイン索引を作成する必要があります。最初の索引の作成後、その索引を定期的にリフレッシュします。索引を何度もリフレッシュしたら、索引のパフォーマンスおよびスペース割り振りを改善するために、索引を再編成します。索引の再編成とは、リフレッシュされた小さい索引を大きいメイン索引にマージすることを意味します。索引の再編成は、索引のリフレッシュよりも、より多くの時間とリソースを必要とします。

以下の API を使用して、索引をリフレッシュおよび再編成します。

```
Class Index {
        // Causes the index to reflect all pending index operations
       // in the index data structures or storage, so that
       // added documents can be searched, and deleted
       // documents cannot be searched.
        void build();
        // Instructs the index to reorganize its persistent storage,
        // usually to remove unused space, and so on.
        void reorganize();


indexRef.build(); or indexRef.reorganize()
```

コレクションの作成後、管理パッケージを使用して、コレクションを検索で使用できるようにします。

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

索引を検索で使用可能にする

索引に文書を設定したら、索引を検索で使用できるようにします。これにより、検索アプリケーションが索引付き文書を検索することができます。

以下の API を使用して、索引を検索で使用できるようにします。

```
class AdminService {
        // Make a collection available for search.
        void enableCollectionForSearch(ApplicationInfo appInfo,
 java.lang.String collectionID,
         java.util.Properties config) ;
        // Make a collection unavailable for search.
        void disableCollectionForSearch(ApplicationInfo appInfo,
 java.lang.String collectionID,
         java.util.Properties config) ;

adminService. enableCollectionForSearch(appInfo, collectionID, null);
```

関連概念

13 ページの『管理アプリケーションの構造』
検索および索引 API 管理アプリケーションは、照会を索引サーバーに送信し、コレクションを管理します。

照会動作の制御

Query インターフェースに属するメソッドを使用して、照会の処理方法やそれぞれの結果に戻されるメタデータの内容など、あらゆる分野の照会動作を制御することができます。

各メソッドの詳細は、Javadoc 文書を参照してください。

setProperty(String name, String value) メソッド

照会プロパティーの値を設定します。このメソッドには、以下のモードがあります。

- HighlightingMode: 検索結果詳細のいくつかの領域で、照会用語を強調表示可能にします。値は、次のとおりです。
  - DefaultHighlighting: サマリーのみ、照会用語を強調表示します。これは、検索アプリケーションで HighlightingMode プロパティーが設定されていない場合のデフォルトです。
  - ExtendedHighlighting: 照会用語の強調表示を検索結果の他の領域に拡張します。例えば、タイトル、URL、およびその他のフィールドなどです。
- FuzzyNGramSearch: ファジー検索で N-gram コレクションにおける非制限検索の実行を可能にします。このプロパティーはブールで、値は次のとおりです。

  - false: 明確な検索が行われます。これは、検索アプリケーションで FuzzyNGramSearch プロパティーが設定されていない場合のデフォルトです。
  - true: ファジー検索が行われます。
- ProximityWindowSize: 2 つの照会用語がこのサイズのウィンドウにある場合、文書スコアは隣接するランキング調整に調整されます。このプロパティーの値は、符号なしの整数です。デフォルト値は 5 です。
- AllowStopwordRemoval: 照会構文解析時にストップワードを除去するかどうかを判別します。このプロパティーが設定されていない場合、エンジンは、検索エンジンのポリシーに従って、ストップワードを除去したり除去しなかったりします。このプロパティーはブールで、値は次のとおりです。
  - false: ストップワードは、照会構文解析時に除去されません。
  - true: ストップワードは、照会構文解析時に除去されます。

ランキング情報の公開

検索アプリケーションが値が true の Query オブジェクトのブール・プロパティー ExposeRankInfo と関連付けを行うと、エンタープライズ・サーチは、ResultSet オブジェクトに関連したプロパティーとしてランキング情報を戻します。 ExposeRankInfo プロパティーを検索アプリケーションの照会オブジェクトに追加することができます。

以下の 3 つのプロパティーは ResultSet オブジェクトと関連しています。これらのプロパティーには、 ResultSet.getProperty(property-name) オブジェクトを介してアクセスできます。

- TextScoreWeightInFinalScore プロパティー: この値 (a ともいいます) は、最終文書スコアのテキスト・スコア係数です。文書の最終スコアは、a*(text score) + (1-a)*(static score) として表されます。
- BoostsVector プロパティー: この値は、スペースで区切られた 16 個の 10 進数の整数です。これらの数値は、16 個のランキング調整クラスのランキング調整値です。この値は、検索管理者によって文書のフィールドおよびトークン属性に関連付けられています。 16 の値は、コンテンツ・クラス A、...、コンテンツ・クラス H、メタデータ・クラス A、...メタデータ・クラス H という順序でランキング調整クラスにマッチングします。
- QueryWordBoosts プロパティー: この値は、照会ワードから成るストリングです。各ワードの後にランキング調整が続きます。ランキング調整値は 10 進数の整数です。ワードおよび整数は、スペースで区切られています。照会ワードのランキング調整は、辞書によって決定されます。

以下のプロパティーは Result オブジェクトと関連しています。これらのプロパティーには、 Result.getProperty(property-name) オブジェクトを介してアクセスできます。

- TermHitsPerBoostClass プロパティー: このプロパティーには、スペースで区切られた 16 個の 10 進数の整数値があります。i-th 整数は、ランキング調整クラス i に関連したトークン属性のトークンとして、文書の照会用語オカレンスの数を指定します。 16 の値は、コンテンツ・クラス

A、...、コンテンツ・クラス H、メタデータ・クラス A、...メタデータ・クラス H という順序でランキング調整クラスにマッチングします。

- StaticScore プロパティー: このプロパティー (値は整数) は、文書の静的スコアを表すストリングです。
- NormalizedStaticScore プロパティー: このプロパティー (値は倍精度の数値 (Double)) は、正規化された静的スコア (0 から 1 の数値) を表します。
- TextScore プロパティー: このプロパティー (値は倍精度の数値 (Double)) は、文書のテキスト・スコアを表します。
- KeywordCount プロパティー: このプロパティー (値は 10 進数の整数) は、文書のワード数を表します。
- UniqueKeywordCount プロパティー: このプロパティー (値は 10 進数の整数) は、文書の固有 (特殊) ワードの数を表します。

setLinguisticMode(int mode) メソッド

この照会の言語処理モードを設定します。以下のいずれかのモードを設定できます。

- LINGUISTIC_MODE_ENGINE_DEFINED: エンジンの最善のポリシーに従って、変更されていない用語が突き合わされます。これが、デフォルト・モードです。
- LINGUISTIC_MODE_EXACT_MATCH: 言語処理を行わずに、入力されたままの変更されていない用語が突き合わされます。
- LINGUISTIC_MODE_BASEFORM_MATCH: 言語処理を行った後、変更されていない用語の基本型によって突き合わされます。例えば、照会用語 *jumping* は、*jump*、*jumped*、*jumps* などが含まれている文書にマッチングします。
- LINGUISTIC_MODE_EXACT_AND_BASEFORM: 言語処理を行った後、変更されていない用語の基本型および完全型によって突き合わされます。例えば、照会用語 *jumping* は、*jump*、*jumped*、*jumps* などが含まれている文書にマッチングします。 LINGUISTIC_MODE_BASEFORM_MATCH モードとの違いは、言語基本型によるマッチングが結果文書の指定言語に一致する照会言語に依存するのに対して、LINGUISTIC_MODE_EXACT_AND_BASEFORM モードは、指定言語に関係なく、*jumping* の完全型が含まれている文書が戻されることが保証される点です。

setQueryLanguage(java.lang.String lang) メソッド

検索サーバーで使用する、コレクション・デフォルト言語以外の言語を指定します。例えば、英語の場合、照会言語パラメーターは `en` です。中国語では、中国語 (簡体字) の場合は `zh-CN`、中国語 (繁体字) の場合は `zh-TW` を使用します。

setRequestedResultRange(int fromResult, int numberOfResult) メソッド

戻される結果の範囲を制御します。

fromResult 値は、ランク付けされた文書のうち、どの文書から結果セットを開始するかを制御します。例えば、値 0 は、照会結果の最初の文書を要求していることを意味します。

numberOfResults 値は、結果の現行ページにいくつの結果を戻すかを制御します。最大値は 500 です。

setReturnedAttribute(int attributeType, boolean isReturned) メソッド

各 Result オブジェクトで戻される定義済み結果属性値を使用可能/使用不可に設定します。

デフォルトでは、エンタープライズ・サーチは、メタデータ・フィールド属性 RETURN_RESULT_FIELDS を除く、すべての定義済み結果属性値を戻します。

attributeType オブジェクトに有効な値は、以下のとおりです。

- RETURN_RESULT_TITLE: Result.getTitle() オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_DESCRIPTION: Result.getDescription() オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_FIELDS: Result.getFields() オブジェクトおよび Result.getFields(String) オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_CATEGORIES: Result.getCategories() オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_TYPE: Result.getDocumentType() オブジェクトは、isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_SOURCE: Result.getDocumentSource() オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_LANGUAGE: Result.getLanguage() オブジェクトは、isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_DATE: Result.getDate() オブジェクトは、 isReturned オブジェクトが false に設定されるとヌルを戻します。
- RETURN_RESULT_SCORE: Result.getScore() オブジェクトは、 isReturned オブジェクトが false に設定されると 0.0 を戻します。
- RETURN_RESULT_URI: Result.getDocumentURI() オブジェクトは、isReturned オブジェクトが false に設定されるとヌルを戻します。

setReturnedFields(String[] fieldNames) メソッド

Result オブジェクトに戻されるメタデータ・フィールドを制御します。

デフォルトでは、エンタープライズ・サーチはメタデータ・フィールドを戻しません。

setSiteCollapsingEnabled(boolean value) メソッド

上位の結果に、同じ Web サイトまたはデータ・ソースからの結果を 3 個以上含めるかどうかを指定します。

例えば、特定の照会で http://www.ibm.com から 100 個の結果が戻された場合、サイト縮小が使用可能に設定されていると、ResultSet オブジェクトには、上位の結果に 100 個の結果のうち 2 個の結果しか含まれません。そのサイトの他の結果は、他のサイトの結果がリストされた後に表示されます。

同じサイトからより多くの結果を検索するには、 samegroupas:*result URL* 照会構文を使用するか、または照会ストリングに http://www.ibm.com サイトを追加して、同じ照会を再実行します。詳しくは、 26 ページの『照会構文』を参照してください。

setSortKey(java.lang.String sortKey) メソッド
ソート・キーを指定します。以下の事前定義ソート・キー値が、com.ibm.siapi.search.BaseQuery に定義されています。

- SORT_KEY_NONE
- SORT_KEY_DATE
- SORT_KEY_RELEVANCE

上記のほかに、検索するコレクションの有効な数値フィールド名をソート・キーとして指定できます。デフォルトのソート・キーは SORT_KEY_RELEVANCE です。

setSortOrder(int sortOrder) メソッド
SORT_ORDER_ASCENDING または SORT_ORDER_DESCENDING のいずれかのソート順序を指定します。

ソート・キーが SORT_KEY_RELEVANCE または SORT_KEY_NONE の場合には、ソート順序は無視されます。

setSortPoolSize(int sortPoolSize) メソッド
上位関連結果のいくつをソートして、結果セットに戻すかを制御します。値は 1 から 500 までです (デフォルトのソート・プール・サイズは 500 です)。それ以外の値を指定すると、検索サーバーは SiapiException オブジェクトをスローします。

sortKey が SORT_KEY_RELEVANCE または SORT_KEY_NONE の場合には、ソート・プール・サイズは無視されます。

setPredefinedResultsEnabled (boolean value) メソッド
照会結果で、通常の結果のほかに、定義済みリンクを含めるかどうかを指定します。デフォルトでは、定義済みリンクが使用可能になります。

setSpellCorrectionEnabled(boolean enable) メソッド
照会結果に照会のスペル修正の提案を含めるかどうかを指定します。デフォルトでは、スペル修正は使用不可になります。

setResultCategoriesDetailLevel (int detailLevel) メソッド
照会結果に要求されるカテゴリー詳細レベルを指定します。このメソッドは、カテゴリー属性 RETURN_RESULT_CATEGORIES が使用可能な場合に使用されます。デフォルト値は、RESULT_CATEGORIES_ALL です。

- RESULT_CATEGORIES_ALL: 各結果カテゴリーは、完全なパス (ルート・パスから始まる) 情報と共に戻されます。
- RESULT_CATEGORIES_NO_PATH_TO_ROOT: 各結果カテゴリーは、絶対パス情報を伴わずに戻されます。つまり、ResultCategory.getPathFromRoot() はヌルを戻します。結果カテゴリーの検索を完全に停止する場合は、setReturnedAttribute(RETURN_RESULT_CATEGORIES, false) 属性を使用します。

setSynonymExpansionMode (int mode) メソッド

照会に対して同義語の拡張モードを設定します。以下のいずれかのモードを設定できます。

- SYNONYM_EXPANSION_OFF: この定数を setSynonymExpansionMode メソッドに渡して、照会に同義語演算子が含まれていても、同義語が拡張されないようにします。
- SYNONYM_EXPANSION_MANUAL: この定数を setSynonymExpansionMode メソッドに渡して、同義語演算子による影響を受ける照会用語についてのみ同義語を拡張します。
- SYNONYM_EXPANSION_AUTOMATIC: この定数を setSynonymExpansionMode メソッドに渡して、最大限の努力をして、該当するすべての照会用語を拡張します。

setACLConstraints(java.lang.String aclConstraints)

セキュアな検索を行うために、この照会にアクセス制御に関する制約を設定します。

関連概念

『照会構文』
照会で特定の文字を使用することにより、検索結果を絞り込むことができます。

関連資料

38 ページの『照会構文の構造』
検索および索引 API 照会は、特定の構造に従って、スペースで区切られた照会用語のリストです。

照会構文

照会で特定の文字を使用することにより、検索結果を絞り込むことができます。

単純照会構文文字

以下に、照会結果を絞り込むために検索アプリケーションで使用できる文字を示します。

フリー・スタイル照会構文

フリー・スタイル照会構文は、明示的な変換処理が無い照会を記述する際に使用します。この照会には、通常、正符号 (+)、曲折アクセント記号 (^)、負符号 (-) が付いておらず、デフォルトの動作が定義されていない用語が含まれます。

照会: `computer software`

結果: この照会は、 *computer* または *software* という用語、その両方、あるいはインプリメンテーションのセマンティクスによってはそれ以外の用語を含む文書を戻します。

~ (接頭部)

用語の前に波形記号 (~) を付けると、その用語またはその用語のシノニム (同義語) が含まれている文書を検索することを示します。

照会: `~fort`

照会: この照会は、 `fort` という用語またはそのシノニム (`garrison` や `stronghold` など) が含まれている文書を検索します。

~ (接尾部)

用語の後に波形記号 (~) を付けると、言語学上の基本型が照会用語と同じである用語 (見出し語または語幹ともいいます) が含まれている文書を検索することを示します。

照会: `apples~`

結果: この照会は、*apples* または *apple* という用語が含まれている文書を検索します (apple は apples の基本型であるため)。

+

用語の前に正符号 (+) を付けると、その用語に一致する用語が含まれている文書を検索することを示します。

照会: `+computer +software`

結果: この照会は、 *computer* と *software* という用語が含まれている文書を戻します。

–

用語の前に負符号 (-) を付けると、その用語が含まれていない文書を検索することを示します。

照会: `computer -hardware`

結果: この照会は、 *computer* という用語が含まれており、*hardware* という用語が含まれていない文書を戻します。

=

用語の前に等号 (=) を付けると、その用語に完全一致する用語が含まれている文書を検索することを示します。 (見出し語処理は使用できません。)

照会: `=apples`

結果: この照会は、複数形 `apples` が含まれている文書のみを戻します。

^

用語の前に曲折アクセント記号 (^) を付けると、曲折アクセント記号 (^) を付けて指定した用語と、曲折アクセント記号を付けずに指定した用語が少なくとも 1 つ以上含まれている文書を検索することを示します。

照会: `^computer science software`

結果: この照会は、 `computer` と *science* という用語、または `computer` と *software* という用語が含まれている文書を戻します。

照会: `cats dogs ^$language::en ^$doctype::html`

結果: この照会は、 *cats* および *dogs* という用語のうち、少なくとも 1 つが含まれている英語の HTML 文書を戻します。

*****

用語またはフィールドの途中または前後にワイルドカード文字 (*) を付けると、起こりうるあらゆる組み合わせに一致する用語が含まれている文書を検索することを示します。ワイルドカード文字が含まれている用語は、その該当するすべての拡張の OR と同等に解釈されます。ワイルドカード・サポートでは、以下のルールが適用されます。

- 拡張セットには、拡張の最大構成数が含まれます。索引内に最大数を超える拡張がある場合、それらの拡張は無視されます。ワイルドカードを含む用語の一部の拡張が無視されると、照会結果にそのことが示されます。

- 拡張セットには、ワイルドカード文字を任意の文字シーケンスに置き換えてできる、索引内のすべての用語が含まれます。
- ワイルドカード文字サポートが、フィールド・セットに限られている場合、セットにはそのいずれかのフィールドにある用語のみが含まれます。用語は、索引内の少なくとも 1 つの文書のいずれか 1 つのフィールドにのみある必要があります。
- 用語がフィールド化された用語である場合、ワイルドカード文字はフィールド指定子の後にのみ指定できます。ワイルドカード・サポートが、フィールド・セットに限られている場合、ワイルドカードを含む用語のフィールド名は、これらのフィールドのいずれかでなければなりません。それ以外の場合、用語は拡張されません。
- ワイルドカードを含む用語が、コレクションの構成済みワイルドカード・フレーバーでサポートされていない場合、拡張セットは空です。ただし、検索ランタイムは、最善の方法で照会結果を戻すために、以下を行います。
  - ワイルドカード文字がコレクションでサポートされていない場合、ワイルドカードを含む用語はワイルドカード文字を除いた用語として解釈されます。例えば、 `to*y` を照会した場合、`tony` ではなく、 `toy` に一致するすべての結果が戻されます。
  - 末尾ワイルドカード文字のみがサポートされているのに、ワイルドカードを含む用語の途中にワイルドカード文字が含まれている場合、この用語は最初のワイルドカード文字の後続文字を除いた用語として解釈されます。例えば、 `to*y` を照会した場合、`to*` という照会で戻されるすべての結果が戻されます。
- ワイルドカード文字は、プレーン・テキスト用語に対してのみサポートされます。ワイルドカード文字は、XML エレメント名、属性名、属性値にはサポートされません。
- ワイルドカードのみが単独で含まれている用語は、サポートされません。
- ワイルドカード文字は、句 (フレーズ) 内でサポートされます。
- ワイルドカードを含む用語の拡張数が拡張の最大構成数を超えると、その最大数を超える拡張は照会評価で無視されます。その場合、ResultSet オブジェクトは、以下のことを示します。
  - ResultSet オブジェクトの isEvaluationTruncated() メソッドは true を戻します。これは、最大数を超える状態のみを示しているわけではありません。このメソッドは、評価がタイムアウトによって早期終了した場合にも true を戻します。
  - ResultSet オブジェクトの特定のプロパティーが true に設定されます。これは ResultSet.getProperty(″WildCardTruncation″) オブジェクトによって抽出でき、true が戻されます。タイムアウトによる切り捨ての場合は、isEvaluationTruncated() オブジェクトで true が戻され、ResultSet.getProperty(″TimeOutTruncation″) オブジェクトで true が戻されます。

照会: `app*`

結果: この照会は、 *apple*、*apples*、*application* などの用語が含まれている文書を検索します (これらの用語 は *app* で始まるため)。

照会: `DB2 info*`

結果: この照会は、*DB2* の後に *info* で始まるワードが続く用語が含まれている文書をすべて検索します。

照会: `title:tech*`

結果: この照会では、*title* というフィールドに *technology* という用語があれば、その用語が検索されます。

要確認: エンタープライズ・サーチ管理コンソールで、ワイルドカード・サポートを使用可能に設定する必要があります。

`" "`

二重引用符 (") を使用すると、二重引用符内に指定した句とまったく同じ句が含まれている文書を検索することを示します。句の中の用語には、見出し語処理は行われません。

句の中にも、ワイルドカード文字 (*) を追加できます。ワイルドカード文字 (*) は、文字または用語に隣接して指定します。ワイルドカード文字は単独ではサポートされません。エンタープライズ・サーチ管理コンソールで、ワイルドカード文字サポートを使用可能に設定する必要があります。

照会: `"computer software programming"`

結果: この照会は、`computer software programming` とまったく同じ句が含まれている文書を検索します。

デフォルトでは、句は必要条件になります。したがって、`building "new york"` および `building +"new york"` という 2 つの照会は同じになります。句は、禁止条件 (-) および必要不十分条件 (^) にすることもできます。

照会: `"app* pea*"`

結果: この照会は、 *apples pears*、*appears peaceful*、*appreciate peas* などの用語が含まれている文書を検索します (これらの用語は、*app* および *pea* で始まるため)。この照会では、*apples and pears* などのような組み合わせが含まれる文書は検出されません。句の中の単独のワイルドカード文字 (*) は無視され、`"apples * pears"` という照会は、 *apples pears* という照会と同じ結果になります。

`"apples * pears"` のような照会は、*apples and pears* や *apples or pears* には一致せず、 *apples pears* に一致します。

`( )` 括弧 ( ) を使用すると、括弧内の用語のうち 1 つ以上が含まれる文書を検索することを示します。

括弧内には、正符号 (+)、負符号 (-)、曲折アクセント記号 (^) を使用しないでください。

括弧内の用語は、OR や垂直バー (|) で区切ります。

照会: `+computer (hardware OR software)`

照会: `+computer (hardware | software)`

結果: 上記の照会は、両方とも、*computer* という用語と、 *hardware* または *software* という用語が少なくとも 1 つ以上含まれている文書を検索します。

OR 条件は、必要条件 (+) または必要不十分条件 (^) にすることができますが、禁止条件 (-) にすることはできません。これによって、照会言語機能が制限されることはありません。 `-(dogs OR cats)` は、`-dogs -cats` と表すことができます。

OR 条件は、デフォルトでは必要条件 (+) になります。よって、前述の照会は、 `+computer +(hardware | software)` と同じになります。

ネストされた OR ステートメントはサポートされません。

site:*text*

Web コンテンツが含まれているコレクションを検索する場合に、 `site` キーワードを使用して、特定のドメインを検索します。例えば、特定の Web サイトの全ページを戻すことができます。

サイト照会では、接頭部に `http://` は付けないでください。

照会: `+laptop site:www.ibm.com`

結果: この照会は、`laptop` という用語が含まれている、`www.ibm.com` ドメイン上のすべての文書を検索します。

url:*text*

Web コンテンツが含まれているコレクションを検索する場合に、 `url` キーワードを使用して、URL の一部に特定の語が含まれている文書を検索します。

照会: `url:support`

結果: この照会は、 `support` という値が含まれている URL (`http://www.ibm.com/support/fr/` など) の文書を検索します。

link:*text*

Web コンテンツが含まれているコレクションを検索する場合に、 `link` キーワードを使用して、特定の Web ページへのハイパーテキスト・リンクが少なくとも 1 つ以上含まれている文書を検索します。

照会: `link:http://www.ibm.com/us`

結果: この照会は、ページ `http://www.ibm.com/us` へのリンクが 1 つ以上含まれている文書をすべて検索します。

*field***:***text*

コレクション内の文書にフィールド (または列) が含まれており、コレクション管理者がこれらのフィールドをフィールド名によって検索可能にしている場合に、コレクション内の特定のフィールドを照会できます。

照会: `lastname:smith div:software`

結果: この照会は、ソフトウェア部門 (`div:software`) で働いており、ラストネームが Smith (`lastname:smith`) である従業員に関する文書をすべて検索します。

docid:*documentid*

特定の URI (または文書 ID) の文書を検索するには、docid キーワードを使用します。通常、コレクション内で特定の URI に一致する文書は多くても 1 つしかありません。

照会: (docid:http://www.ibm.com/solutions/us/ OR docid:http://www.ibm.com/products/us/)

結果: この照会は、URI が http://www.ibm.com/solutions/us/ または http://www.ibm.com/products/us/ のすべての文書を検索します。

samegroupas:*URI*

デフォルトで、エンタープライズ・サーチは、ホスト名が同じ URL は同じグループに属しているとみなします。また、同じスレッドのニュース記事は同じグループに属しているとみなします。その他すべてのデータ・ソースの URI の場合、各 URI は独自のグループを形成します。ただし、エンタープライズ・サーチを使用して、特定の接頭部に一致する URI をグループに編成することができます。例えば、以下のグループ定義を構成することができます。

```
http://mycompany.server1.com/hr/          hr
http://mycompany.server2.com/hr/          hr
http://mycompany.server3.com/hr/          hr
http://mycompany.server1.com/finance/     finance

file:///myfileserver1.com/db2/sales/      sale
file:///myfileserver1.com/websphere/sales/  sale
file:///myfileserver2.com/db2/sales/      sale
file:///myfileserver2.com/websphere/sales/  sale
```

この例では、接頭部が http://mycompany.server1.com/hr/ または http://mycompany.server2.com/hr/ または http://mycompany.server3.com/hr/ の URI は、すべて *hr* という 1 つのグループに属することになります。接頭部が http://mycompany.server1.com/finance/ の URI は、すべて別のグループ *finance* に属することになります。さらに、接頭部が file:///myfileserver1.com/db2/sales/ または file:///myfileserver1.com/websphere/sales/ または file:///myfileserver2.com/db2/sales/ または file:///myfileserver2.com/websphere/sales/ の URI は、すべて *sale* という別のグループに属することになります。

file:///myfileserver2.com/websphere/sales/mypath/mydoc.txt がコレクション内の URI である場合、以下の検索項目を指定した照会では、検索が *sale* グループ内の URI に限定されます。

```
samegroupas:file:///myfileserver2.com/websphere/sales/mypath/mydoc.txt
```

この照会のすべての結果には、以下のいずれかの接頭部が付きます。

```
file:///myfileserver1.com/db2/sales/
file:///myfileserver1.com/websphere/sales/
file:///myfileserver2.com/db2/sales/
file:///myfileserver2.com/websphere/sales/
```

照会: samegroupas:http://www.ibm.com/solutions/us/

結果: この照会は、`http://www.ibm.com/solutions/us/` と同じグループに属する URI (この場合は URL) の文書をすべて検索します。

taxonomy_ID`::`*category_ID*

エンタープライズ・サーチ用に作成されたカテゴリーが含まれているコレクションを検索する場合に、特定の分類法の特定のカテゴリーに属する文書を検索することができます。エンタープライズ・サーチは 2 つのタイプのカテゴライザーをサポートしており、各カテゴライザーには分類法 ID (ルール・ベースまたはモデル・ベース) があります。

カテゴリー ID を検索するには、*ES_NODE_ROOT*/*col_xyz*.parserdriver/CategoryTree.xml ディレクトリーに移ります。CategoryTree.xml ファイルにカテゴリー ID が含まれています。

分類法 ID

rulebased

この分類法 ID を使用して、文書のカテゴリー化に文書コンテンツ規則または文書 URI 規則を使用しているカテゴリーに属する文書を検索します。ルール・ベースのカテゴリーの構成については、「*DB2 Information Integrator OmniFind Edition エンタープライズ・サーチの管理*」を参照してください。

照会: `rulebased::c1`

結果: この照会は、 c1 というルール・ベースのカテゴリー ID に属している文書を戻します。

modelbased

この分類法 ID を使用して、WebSphere Portal 分類法に定義されているカテゴリーに属する文書を検索します。

照会: `modelbased::c3`

結果: この照会は、 c3 というモデル・ベースのカテゴリー ID に属している文書を戻します。

taxonomy_ID`::`*category_ID* 照会用語は、指定された *category_ID* またはそのサブカテゴリーに属する文書を検索します。これは、分類法 ID の前に波形記号 (~) を付けることによって、明示的に指定できます。

照会で、指定されたカテゴリーに属する文書を戻し、そのサブカテゴリーに属する文書を戻さない場合は、 *taxonomy_ID::category_ID* の前に等号 (=) をつけて、 `=rulebased::c1` のように指定します。

scopes::*scope_name*

有効範囲 (索引内の URI の範囲) に属する文書を検索するには、有効範囲名を使用します。有効範囲の構成については、「*DB2 Information Integrator OmniFind Edition エンタープライズ・サーチの管理*」を参照してください。

照会: `scopes::research "computer science"`

結果: この照会は、 `computer science` という句が含まれている research という有効範囲に属する文書を戻します。

$source::*source_type*

特定のデータ・ソース・タイプの文書を検索するには、$source キーワードを使用します。ソース照会は、複数のソースの文書が含まれているコレクションにおいて便利です。

コレクションに有効なソース・タイプのリストを入手するには、そのコレクションの検索可能オブジェクトの getAvailableAttributeValues(Searchable ATTRIBUTE_SOURCE) メソッドを呼び出します。

照会: `$source::DB2 "computer science"`

結果: この照会は、computer science という句が含まれている、DB2 クローラーによってコレクションに追加された文書を検索します。

$language::*language_id*

特定の言語で書かれている文書を検索するには、$language キーワードを使用します。

コレクションに有効な言語 ID のリストを入手するには、そのコレクションの検索可能オブジェクトの getAvailableAttributeValues(Searchable.ATTRIBUTE_LANGUAGE) メソッドを呼び出します。

照会: `$language::en "computer science"`

結果: この照会は、computer science という句が含まれている英語の文書を検索します。

$doctype::*document_type*

特定の文書フォーマットまたは MIME タイプの文書を検索するには、$doctype キーワードを使用します。

コレクションに有効な文書タイプのリストを入手するには、そのコレクションの検索可能オブジェクトの getAvailableAttributeValues(Searchable.ATTRIBUTE_DOCTYPE) メソッドを呼び出します。

照会: `$doctype::application/pdf "computer science"`

結果: この照会は、computer science という句が含まれている Portable Document Format (PDF) 文書を検索します。

#*field***::=***value*

指定した数値に等しい値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::=1700 laptop`

結果: この照会は、laptop および price フィールドの値が 1700 である文書を検索します。

#*field***::>***value*

指定した数値より大きい値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>1700 laptop`

結果: この照会は、laptop および price フィールドの値が 1700 より大きい文書を検索します。

#*field*::<*value*

指定した数値より小さい値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::<1700 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 より小さい文書を検索します。

#*field*::>=*value*

指定した数値より大きいか等しい値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>=1700 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 以上である文書を検索します。

#*field*::<=*value*

指定した数値より小さいか等しい値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::<=1700 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 以下である文書を検索します。

#*field*::>*value1*<*value2*

指定した数値の範囲内の値 (指定した数値を含まない) が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>1700<3900 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 より大きく 3900 より小さい文書を検索します。

#*field*::>=*value1*<=*value2*

指定した数値の範囲内の値 (指定した数値を含む) が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>=1700<=3900 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 以上 3900 以下の文書を検索します。

#*field*::>*value1*<=*value2*

指定した数値の範囲内の基準に一致する値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>1700<=3900 laptop`

結果: この照会は、`laptop` および `price` フィールドの値が 1700 より大きく 3900 以下の文書を検索します。

#*field*::>=*value1*<*value2*

指定した数値の範囲内の基準に一致する値が数値フィールドにある文書を検索するには、パラメトリック制約構文を使用します。

照会: `#price::>=1700<3900 laptop`

結果: この照会は、laptop および price フィールドの値が 1700 以上で 3900 より小さい文書を検索します。

ACL constraints: (*security_tokens*)
セキュリティーに関して、照会ストリングにアクセス制御制約を指定することはできません。照会のアクセス制御制約を指定するには、Query インターフェースの setACLConstraints(String aclConstraints) メソッドを使用します。括弧、正符号 (+)、負符号 (-)、曲折アクセント記号 (^)、 XML セキュリティー・コンテキスト・ストリングを ACL 制約ストリング (@SecurityContext::'securityContext') に指定することができます。securityContext ストリング構文については、setACLContstraints メソッドについて記載されている Javadoc 文書を参照してください。各シンボルの意味は、前述の構文記述に示されているものと同じです。

setACLConstraints メソッドにおける ACL 制約ストリング (michelle_c | dev_group)

setACLConstraints メソッドにおける ACL 制約ストリング: michelle_c @SecurityContext::'securityContext'

照会: thinkpad

結果: この照会は、最初のケースでは thinkpad という用語とセキュリティー・トークン michelle_c または dev_group が含まれている文書を、2 番目のケースでは thinkpad という用語と michelle_c および指定されたセキュリティー・コンテキスト制約が含まれている文書を検索します。

不透明条件の照会構文文字

エンタープライズ・サーチを使用して、2 つのタイプの不透明条件に関する照会構文を作成することもできます。不透明条件を使用すると、照会の一部を XML フラグメントおよび XPath などの他の言語で表すことができます。 XML フラグメントおよび XPath は、XML 照会言語の 2 つのタイプです。XML フラグメントは、UIMA 構造の照会にも使用できます。不透明条件の記号は、*@xmlf2::* (XML フラグメント) または *@xmlxp::* (XPath 照会) で表されます。 XML フラグメントまたは XPath 照会は、単一引用符 (' ') で囲みます。

xmlf2 表記は XML フラグメントに、*xmlxp* 表記は XPath 条件に使用されます。不透明条件の構文は @syntax_name::'value' のようになります。式は @ 記号で始まり、構文名 (xmlf2 または xmlxp)、2 つのコロン (::)、および単一引用符 (' ') で囲まれた値の順になります。'value' パラメーターの前に +、–、^ が付くことがあります。式の値セクションで単一引用符を使用する必要がある場合には、円記号 (¥) を使用して単一引用符を拡張します。例えば、¥' のようにします。

@xmlf2::'<tag1> text1 </tag1>'
フラグメント照会を 新規検索および索引 API 不透明条件として表すには、@xmlf2:: 接頭部を使用して、照会を単一引用符で囲みます。

照会: @xmlf2::'<title>"Data Structures"</title>'

結果: この照会は、 *title* という索引付き注釈のスパン内に *Data Structures* という句が含まれている文書を検索します。

@xmlf2::<tag1><.depth value=″$number″><tag2> ... </tag2></.depth></tag1>
@xmlf2::<tag1><.depth value='$number'><tag2> ... </tag2></.depth></tag1>

最初の照会では二重引用符を使用しています。2 番目の照会では単一引用符を使用しています。ただし、各照会では同じ結果が戻されます。この照会構文は、tag1 から $number レベル下の tag2 のオカレンスを探します。depth は、照会において、語 w またはタグ t と、w または t に最も近いタグ t' との間のレベル数を表します。

$number は正の整数です。数値を単一引用符 (' ') または二重引用符 (" ") で囲むことができます。この照会構文は Unstructured Information Management Architecture (UIMA) には適用されません。

照会: (この照会は 1 行で指定してください。)

```
@xmlf2::'<author>Albert Camus<.depth value='1'>
<publisher>Carey Press</publisher></.depth></author>'
```

結果: この照会は、author より 1 レベル下の publisher の文書を検索します。以下のような XML エレメントを持つ文書があるとします。

```
<author>Albert Camus<ISBN>002-12345</ISBN><country>USA
<publisher>Carey Press</publisher></country></author>
```

この文書は、上記の照会例では戻されません。 publisher (<publisher>) エレメントが author (<author>) エレメントより 2 レベル下にあるからです。

@xmlf2::'<tag1><@tag1> ... </@tag1></tag1>'

エレメントと属性を区別することができます。属性は、エレメント内に明示的に指定するか、または @ 記号の後にサブエレメントとして指定します。@ 記号によって、同じ名前である可能性があるエレメントと属性を区別することができます。

属性内に語句を定義することができます。これは、照会の通常の用語と同じです。ただし、タグを使用せず、語句のみで式を作成できます。これらの語または句は、通常の照会での用語と同じ機能をサポートします。

照会: `@xmlf2::'<author country="USA"></author>'`

照会: `@xmlf2::'<author><@country>USA</@country></author>'`

結果: この照会は、author が USA 出身である文書を検索します。

照会:

```
@xmlf2::'<author><@country>USA</@country>
<firstName>Michelle</firstName>
<lastName>Ropelatto</lastName></author>'
```

結果: この照会は、USA 出身の author 名 Michelle Ropelatto が含まれている文書を検索します。

@xmlf2::'+text1 ... +text2 -text3 ... -text4 text5'

語または句の前に、正符号 (+) または負符号 (-) を使用します (通常、引用符 (" ") で囲みます)。各照会レベルで、「+」はその用語が含まれていなければならないことを意味し、「-」はその用語が含まれていてはならないことを意味します。それ以外の用語はオプションであり、ランキング情報にのみ関与します。「+」が付いた用語が無い場合は、少なくとも 1 つ以上のオプションの用語がなければなりません。エレメントの下にあるデータは、新たにネストされた照会レベルを作成します。

照会: `@xmlf2::'+"Graph Theory" -network'`

結果: この照会は、`Graph Theory` という句が含まれており、network という用語が含まれていない文書を検索します。

@xmlf2::'<tag1> <.or> ... </.or> <.and> ... </.and> </tag1>'
フラグメント照会で、AND (<.and>) および OR (<.or>) スコープを表すために絶対ブールを使用します。

照会: `@xmlf2::'<book><.or><author>Sylvia Plath</author><title>XML -Microsoft</title></.or></book>'`

結果: この照会は、book の author が Sylvia Plath であるか、または book の title に XML という語が含まれており Microsoft® という語が含まれていない文書を検索します。

@xmlf2::'<annotation1+annotation2> ... </annotation1+annotation2>'
フラグメント照会で、エレメントの開始タグと終了タグの間に正符号 (+) を使用することで、連続する注釈の連結を表すことができます。連続する注釈は、少なくとも 1 語はオーバーラップしていなければなりません (交差していなければなりません)。複数の重複した注釈の連結は、個々の注釈によってカバーされるテキスト全体をカバーする新規の仮想注釈になります。

照会: `@xmlf2::'<Report+HoldsDuring> +Pakistan +March +Reuters</Report+HoldsDuring>'`

結果: この照会は、『Report』 と 『HoldsDuring』 によって形成される連結注釈に含まれている、 3 月 (March) にパキスタン (Pakistan) で発生した事件に関するロイター (Reuters) の文書を検索します。

@xmlf2::'<annotation1*annotation2> ... </annotation1*annotation2>'
エレメントの開始タグと終了タグの間にアスタリスク記号 (*) を使用することで、フラグメント照会において注釈の交差を表すことができます。複数の重複した注釈の交差は、重複した注釈の交差によってカバーされるテキストのみをカバーする新規の仮想注釈になります。

照会: `@xmlf2::'<Inhibits* Activates>Aspirin</Inhibits*Activates>'`

結果: この照会は、「Inhibits」と「Activates」の両方の注釈に Aspirin という用語がある文書を検索します。

@xmlxp::'/tag1/@tag1'
エレメント (XML 開始および終了タグ) と属性を区別することができます。属性を @ 記号の後に明示的に指定します。@ 記号によって、同じ名前である可能性があるエレメントと属性を区別することができます。連結および論理積は UIMA 文書にのみ適用でき、純正の XML 文書には適用できません。XML 文書では定義によってスパンが交差しないためです。

照会: `@xmlxp::'/author[@country="USA"]'`

結果: この照会は、*author* に関連した属性 *country* の値である文字ストリングに *USA*が含まれている文書を検索します。

@xmlxp::'/tag1[tag2 or tag3 and tag4]'
XPath 照会で AND および OR スコープを表すには、絶対ブールを使用します。

照会: `@xmlxp::'book[author ftcontains("Jose Perez") or title ftcontains("XML -Microsoft")]'`

結果: この照会は、book の author が Jose Perez であるか、または book の title に XML という語が含まれており Microsoft という語が含まれていない文書を検索します。

@xmlxp::'tag1//tag2/tag3'

下層ノード (//) と下位ノード (/) を区別することができます。

照会: `@xmlxp::'/books//book/name'`

結果: この照会は、book エレメントが books エレメントの子孫で、 name エレメントが book エレメントの直接の子である文書を検索します。

@xmlxp::'tag1/.../tagn'

XPath 照会を検索および索引 API 不透明条件として表すには、@xmlxp:: 接頭部を使用して、照会を単一引用符で囲みます。

照会: `@xmlxp::'books[booktitle ftcontains("Data Structures")]'`

結果: この照会は、「title」という索引付き注釈のスパン内に「Data Structures」という句が含まれている文書を検索します。

関連概念

21 ページの『照会動作の制御』

Query インターフェースに属するメソッドを使用して、照会の処理方法やそれぞれの結果に戻されるメタデータの内容など、あらゆる分野の照会動作を制御することができます。

照会構文の構造

検索および索引 API 照会は、特定の構造に従って、スペースで区切られた照会用語のリストです。

照会は、以下のような構造になっています。

```
<space>*{Query_term <space>+}* Query_term
```

照会用語は、以下のいずれかのタイプになります。

- Word
- Phrase
- AttributeConstraint
- CategoryConstraint
- RangeConstraint
- OrTerm
- OpaqueTerm

外観修飾子

外観修飾子は、照会用語が以下のいずれであるか制御します。

- 必須: 結果文書内になければなりません。
- 禁止: 結果文書内にあってはなりません。

- 必須であるが不十分: 不十分な用語のみが含まれている文書は、検索結果として適格ではありません。

セマンティクスは、検索および索引 API の BaseQueryTerm オブジェクトの定数に対応します : `Appearance_Modifier = { + | -| ^ }`

- `+` は、必須用語を表します。
- `–` は、禁止用語を表します。
- `^` は、文書を検索結果として適格とするには不十分である必須用語を表します。

OR 用語を禁止として表すことはできません: `OR_Appearance_Modifier = { + | ^ }`

突き合わせタイプ修飾子

前方突き合わせタイプ修飾子は、修飾する語の直前に指定します: `PreMatch_Type = { = | ~ }`

- `=` は、語がそのままの形 (語幹処理や見出し語処理を行わない) で一致しなければならないことを表します。また、語のシノニムが含まれるように検索を拡張しないことを示します。
- `~` は、語のシノニムが含まれるように検索を拡張することを表します。

後方突き合せ修飾子は、修飾する語の直後に指定します: `PostMatch_Type = { * | ~ }`

- `*` は、指定した接頭部を持つ語に一致します
- `~` は、同じ基本型を共有する語に一致します。例えば、指定した語の語幹や見出し語がある語などです。

デフォルトで、外観修飾子により明示的に修飾され、突き合わせタイプ修飾子により修飾されていない語は、完全一致突き合わせ (“as is”) セマンティクスを使用します。

フィールド検索の表記

フィールド検索の表記、つまりフィールド名 (トークン) は、直後にコロンを付けます。フィールド名とコロンの間にスペースは不要です : `Field = field_name:`

OR 条件

OR 条件は、OR 可能用語を順に指定したものです。スペースと OR-SIGN で区切り、括弧で囲みます。「OrTerm」と「OpaqueTerm」以外のすべての照会用語タイプが OR 可能用語です。 OR 式は括弧で囲みます。括弧内の用語は、以下の 3 つの必須シーケンスで区切ります。

- 1 つ以上のスペース
- 垂直バー ‘|’ または大文字の‘OR’のいずれか (どちらの書き方をすることもできます)
- 1 つ以上のスペース

意味的に、論理和演算された用語が少なくとも 1 つはある文書が検索結果として戻されます。

```
ORable_term = Query_term ¥ { OrTerm, OpaqueTerm }

OR-SIGN = | | OR

ORable_query = <space> * { ORable_term <space> + OR-SIGN <space> +}*
ORable_term

OrTerm = OR_Appearance_Modifier? ( ORable_query )
```

OR_Appearance_Modifier が指定されていない場合、a + が暗黙的に想定されます。ORable_query の個々の用語に固有の外観修飾子を指定することはできません。

語句

以下において、

```
Word = PreMatch_Type? Appearance_Modifier? Field? value  |
Appearance_Modifier? Field? value PostMatch?
```

- 語 (またはフィールド名) は、単一の突き合わせタイプ標識 (先頭または末尾) および外観修飾子が指定されている場合もあります。
- 突き合わせタイプ標識も接頭部標識も指定されていない場合、検索されるフォームはインプリメンテーションによって異なります。
- 外観修飾子が指定されていない場合、結果に用語が含まれていなければならないかどうかは、インプリメンテーションによって異なります。
- 値の任意の場所にワイルドカード記号「*」を含むことができますが、ワイルドカード以外の文字が少なくとも 1 つは値になければなりません。

以下の句は 1 行で表す必要があります。

```
Phrase = Appearance_Modifier? Field?"
<space>*{value<space>+}* value <space>*"
```

- 引用符内には、空ではない値をスペースで区切って順に指定します。フィールドおよび外観修飾子は、両方ともオプションです。
- 外観修飾子が指定されていない場合、a + が暗黙的に想定されます。
- 句内の各値は現状のまま検索されます。つまり、完全一致突き合わせセマンティクスが使用されます。

属性、カテゴリー、範囲の制約

```
Attribute_Name: { language | source |  doctype }
AttributeConstraint = $Attribute_Name::value
```

$ 記号の後に属性名を指定します。その後に 2 つのコロンと値を指定します。Appearance_Modifier が指定されていない場合、a + が暗黙的に想定されます。

以下のカテゴリー制約は 1 行で表す必要があります。

```
CategoryConstraint =
PreMatch_Type?Appearance_Modifier?taxonomy_id::category_id
```

分類法 ID の後には、 2 つのコロンとカテゴリー ID を指定します。

- Match_Type = は、文書を指定されたカテゴリーのメンバーに制限しますが、 ~ は、文書が指定されたカテゴリーの下層 (サブカテゴリー) に属していても構わないことを表します。
- Match_Type が指定されていない場合、~ が暗黙的に想定されます。

- Appearance_Modifier が指定されていない場合、a + が暗黙的に想定されます。

パラメトリック・フィールドが 2 倍値より大きくなければならない (2 番目のケースでは以上) 場合:

```
Grelation = > double_value | >= double_value
```

パラメトリック・フィールドが 2 倍値より小さくなければならない (2 番目のケースでは以下) 場合:

```
Lrelation = < double_value | <= double_value
```

文字の後にフィールド名、2 つのコロン、および少なくとも 1 つの関係 (または =) を指定します。外観修飾子は + または ^ です (- は使用できません)。Appearance_Modifier が指定されていない場合、a + が暗黙的に想定されます。

```
RangeConstraint = Appearance_Modifier?# field :: Grelation Lrelation? |
              Appearance_Modifier?# field :: Grelation? Lrelation  |
              Appearance_Modifier?# field :: =double_value
```

不透明条件

@ 記号の後に、いくつかの構文名、2 つのコロン、および単一引用符で囲まれた値を指定します。不透明条件は、外観修飾子の後に指定できます。値として単一引用符を使用する場合は、¥'のように、単一引用符の前に ¥ を付ける必要があります。

```
OpaqueTerm = Appearance_Modifier?@ syntax_name :: ' value '
```

不透明条件のセマンティクスについて、検索および索引 API は以下のように対処します。

- 単一引用符内の値を解析しません。値のストリングは syntax_name に対応するパーサーにそのまま渡されます。
- インプリメンテーションによってサポートされる外部照会言語を定義しません。
- 照会内の不透明条件数および条件間の相互作用を定義しません。これらはすべてインプリメンテーションで定義されます。ほとんどの場合、照会は不透明条件単独であるか、または不透明条件が全く含まれていないかのどちらかです。

照会におけるトークン、フィールド名、値

トークン、フィールド名、および値には、以下の規則があります。

- 特殊文字が含まれていない文字シーケンスをトークンといいます。
- 文字 = (" は、前にスペースがある場合、あるいは照会ストリングの最初にある場合は特殊な意味を持ちます。つまり、これらの文字をトークン内で使用することはできますが、トークンの先頭では使用できません。
- 前述の項目の例外規則として、'(' は OrTerm の中ではトークンの開始に使用できます。 OrTerm はネストできないため、'(' が特殊な意味で使用されることはないからです。
- 文字 + - ^ は、前にスペース、= ~ (のいずれかがある場合、あるいは照会ストリングの最初にある場合にのみ、特殊な意味を持ちます。
- コロンは、フィールド/制約タイプと値の間にある場合は、区切り記号としての意味を持ちます。コロンは、それ以外のすべての場合は通常の文字とみなされます。

- 文字) は、OrTerm の内部にあって OrTerm 内の句の外部にある場合にのみ、特殊な意味を持ちます。この場合、文字) は OrTerm を終了させます。それ以外の場合、文字) は通常の文字とみなされます。
- 文字 * は、値である場合にのみ特殊な意味を持ちます。つまり、ワイルドカード文字はフィールド名には適用されません。
- シーケンス <、<=、>、>= は、制約範囲内でのみ特殊な意味を持ちます。
- " 以外のすべての特殊文字は、句の中では通常の文字とみなされます。これらの特殊文字は、句の中では特殊機能を持ちません。" は句を終了させます。この規則は、前述のすべての規則に優先します。
- ワイルドカード文字は、句の中で使用できます。

一般的に、特殊文字のいずれかが特定の設定で意味を持たない場合は、その文字はトークン/値の一部とみなされます。

照会パーサーの動作は、非準拠ストリングに対して定義されていません。終了されない句を終了させるなど、パーサーが暗黙的に問題を解決する場合もありますが、パーサーがこのような問題を解決しない場合もあります。

ACL 式の構文

ACL 式の構文は、完全照会構文のサブセットです。ACL 式は、基本的に、語、多数の語に関する OR 式、および不透明条件で構成されます。

```
ACL_Expression = <space>* {ACL_term<space>+}* ACL_term

ACL_term = {  ACL_Value  |
              ACL_OrTerm   |
              Security_OpaqueTerm    }

ACL_Value = Appearance_Modifier? value
```

- 値は、外観修飾子が指定されている場合もあります。
- 外観修飾子が指定されていない場合、^ とみなされます。

ACL_OrTerm は 1 行で表す必要があります:

```
ACL_OrTerm = {+|^}? (<space>*{value <space>+
OR-SIGN<space>+}* value <space>*)
```

値のシーケンスは、スペースおよび OR-SIGN で区切られ、括弧で囲みます。OrTerm には、オプションとして、外観修飾子、+ または ^ (- は使用できません) を使用できます。Appearance_Modifier が指定されていない場合、^ が暗黙的に想定されます。OrTerm 内の個々の値に固有の外観修飾子を指定することはできません。

セキュリティー不透明条件

```
Security_OpaqueTerm = Appearance_Modifier?@ syntax_name :: ' value '
```

@ 記号の後に、いくつかの構文名、2 つのコロン、および単一引用符で囲まれた値を指定します。不透明条件は、外観修飾子の後に指定できます。値として単一引用符を使用する場合は、¥'のように、¥ で拡張する必要があります。

照会ストリングにおける不透明条件に関する意味的な特記事項が、同様に適用されます。

ACL 式パーサーの動作は、非準拠ストリングに対して定義されていません。終了されない OR 用語を終了させるなど、パーサーが暗黙的に問題を解決する場合もありますが、パーサーがこのような問題を解決しない場合もあります。

デフォルトで、すべての用語は'^'で修飾されるように、必須であるが不十分とみなされます。 ACL_term を a + で修飾できますが、これは ACL をフィルターとして使用する場合は適していないように思われます。

この構文の表現は、簡単な文書レベルのセキュリティーよりも広範囲なものです。まず、禁止トークン (例えば、『do not return documents that are viewable with this ACL』 など) を使用できます。また、OrTerm を多数組み込むことができるので、この構文で複数レベルのセキュリティーをサポートできます。

```
(server_ACL1 | server_ACL2) (group_ACL1 | group_ACL2)
(user_ACL1 | user_ACL2)
```

関連概念

21 ページの『照会動作の制御』
Query インターフェースに属するメソッドを使用して、照会の処理方法やそれぞれの結果に戻されるメタデータの内容など、あらゆる分野の照会動作を制御することができます。

ターゲット XML エレメントの検索

記号を前に付けることにより、文書と共に結果フィールドが戻されるように指定できます。この結果フィールドには、XML 照会用語に指定した Unstructured Information Management Architecture (UIMA) 注釈のすべてのオカレンスが列挙されます。列挙されているこれらの注釈オカレンスは、戻された文書内にあるもので、それぞれが文書内の XML 照会用語全体のオカレンスの一部となります。

意味検索を指定する不透明な条件で、ポンド記号またはハッシュ記号 (#) を、xmlf2 照会用語の 1 つの XML エレメント (または注釈) の前に付加することができます。 XML エレメントは、オカレンスが列挙されるターゲット XML エレメントとして指定されます。意味検索が XPath によって表される場合、XPath の定義により、大括弧内の句 [..] および述部内にない最下位レベルのエレメントがターゲット・エレメントになります。

例えば、<book language=en> <#author> </#author> </book> という照会または同等の <book language=en> <#author/> </book> という照会では、属性 language=en を持っており、スパン内に注釈 author のオカレンスが含まれている注釈 book のオカレンスが少なくとも 1 つは含まれている文書が戻されます。また、この照会では、<book> タグのオカレンス内にあり、language=en 属性を持っている <author> タグのすべてのオカレンスの列挙も戻されます。

各オカレンスは、固有の ID で列挙されます。 UIMA アノテーターは、固有の ID を、アノテーターが生成した各注釈に割り当てます。 UIMA アノテーターによって生成された注釈ではなく、ロー文書の一部である XML エレメントは固有の ID を持たないので、その結果フィールドに列挙されません。検索された文書のサマリー・フィールドに、列挙されるオカレンスによって文書内でカバーされているテキストが含まれている場合、そのテキストは強調表示されます。

検索された文書の <author> タグの以下のオカレンスは列挙されません。

- <journal> タグのスパン内の <author> タグのオカレンス
- language=ge 属性を持っている <book> タグのスパン内の <author> タグのオカレンス
- language 属性を持っていない <book> タグのスパン内の <author> タグのオカレンス
- XML 文書の一部である <author> タグのオカレンス (つまり、<author> タグが、生成された注釈ではなくロー文書の一部である場合)

検索アプリケーションは、Result オブジェクトの TargetElement プロパティーを介して、ターゲット・エレメントのオカレンスの列挙にアクセスできます。例えば、Result.getProperty("TargetElement") のように指定します。そのプロパティーの戻り値は、スペースで区切られた整数のストリングです。各ストリングは、単一のターゲット・エレメントのオカレンス ID です。

検索および索引 API フェデレーター

フェデレーターを使用して、異機種間で検索可能なコレクションにフェデレーテッド・サーチ要求を発行したり、統一された文書結果セットを入手します。

検索フェデレーターは、サービスのリクエスターとそのサービスを実行するエージェントとの間に介在する仲介コンポーネントです。また、単一の要求から生成される多数の検索を管理する調整リソースでもあります。

エンタープライズ・サーチは、以下のタイプの検索および索引 API フェデレーターを提供します。

- Local Federator
- Remote Federator

検索フェデレーターは、検索および索引 API 検索可能オブジェクトです。複数レベルのフェデレーションが可能ですが、フェデレーションのレベルが多すぎると検索のパフォーマンスが低下します。

Local Federator および Remote Federator は、 WebSphere II OmniFind Edition を使用して作成されたコレクションまたは他の製品を使用して作成されたコレクションを統合します。 WebSphere II OmniFind Edition 以外を使用して作成されたコレクションについては、そのコレクションが Lightweight Directory Access Protocol (LDAP) または Java データベース接続 (JDBC) を使用する場合は統合することができます。

LDAP または JDBC 検索可能オブジェクトを作成するには、完全修飾 LDAP または JDBC AdminService オブジェクト・クラス・パスを渡して、アプリケーションで AdminService オブジェクトを作成します。 LDAP または JDBC コレクションを作成するには、createCollection メソッドを使用します。 LDAP または JDBC 構成情報は、XML 構成ファイルを介して渡されます。 LDAP または JDBC コレクションの作成後、 Service インターフェースを介して検索可能オブジェクトを検索し、検索可能オブジェクトを直接あるいは Local Federator や Remote Federator を介して使用できます。

Local Federator

Local Federator は、クライアントから検索可能なオブジェクトを統合します。

Local Federator は、SIAPI SearchFactory クラスの createLocalFederator メソッドを使用して作成します。照会対象となる一連の検索可能コレクションは、フェデレーターの作成時に指定します。検索呼び出し時に、検索可能オブジェクトのサブセットを指定することもできます。

Local Federator を作成する前に、検索および索引 API SearchFactory を使用して、検索可能オブジェクトを作成または取得する必要があります。 Local Federator に渡される検索可能オブジェクトは、追加情報を使用せずに検索できる状態でなければなりません。 Local Federator は、検索可能オブジェクトを使用して、フェデレーテッド・サーチ要求を発行します。この要求を完了するには、さまざまな検索可能オブジェクトを使用するために必要なすべてのソフトウェア・コンポーネントが Local Federator 環境になければなりません。

以下のコード・サンプルは、

LocalFederator オブジェクトの作成方法および検索要求の発行方法を示しています。

```
Searchable[] finalSearchables;

// create searchables

// create a query and set query options
Query query = searchFactory.createQuery(queryString);
query.setRequestedResultRange(0, 100),
query.setQueryLanguage("en_US");
query.setSpellCorrectionEnabled(true);
query.setPredefinedResultsEnabled(true);

// create the local federator and call search
LocalFederator federator =
 factory.createLocalFederator(finalSearchables);
ResultSet rs = federator.search(query);
```

Remote Federator

Remote Federator は、サーバーから検索可能なオブジェクトを統合します。

Remote Federator は、サーバー上で稼動し、サーバーのリソースを消費します。 Remote Federator では、入力コレクション ID を一致する検索可能オブジェクトにマップする際に、追加のステップが必要です。

Remote Federator は、検索および索引 API AdminService インターフェースを使用して作成されます。 RemoteFederator の構成時に、コレクション ID のセットを渡す必要があります。コレクション ID は、RemoteFederator によって SIAPI 検索可能オブジェクトに内部的にマップされます。 Remote Federator 環境では、Remote Federator をアクセス可能にする小規模なプロキシー以外に、検索可能な関連ソフトウェア・コンポーネントは必要ありません。

各検索アプリケーションには、固有のフェデレーターがあるため、フェデレーター ID は ApplicationInfo ID の値と同じ値です。

以下のコード・サンプルは、RemotelFederator オブジェクトの作成方法および検索要求の発行方法を示しています。

```
// get collection IDs
String[] collectionIDs;

// create a query object
Query query = searchFactory.createQuery(queryString);
query.setRequestedResultRange(0, 100),
query.setQueryLanguage("en_US");
query.setSpellCorrectionEnabled(true);
query.setPredefinedResultsEnabled(true);

// create a remote federator
RemoteFederator federator = getFederator(appinfo, appinfo.getID());
// search
ResultSet rs = federator.search(query);
```

第 3 章 クローラー・プラグイン

クローラー・プラグインは、クロールされた文書のコンテンツまたはメタデータを変更する際に使用する Java アプリケーション・プログラミング・インターフェース (API) です。クローラー・プラグインには、非 Web ソース用と Web ソース用の 2 つのタイプがあります。

非 Web ソース用のクローラー・プラグイン

ビジネスおよびセキュリティー・ルールを適用して、文書レベルのセキュリティーを施行し、エンタープライズ・サーチ索引内の文書に関連したクロール・メタデータを追加、更新、削除することができます。クローラー・プラグイン API は、以下のエンタープライズ・サーチ・クローラーに使用できます。

- Content Manager
- DB2®
- Exchange Server
- Notes®
- NNTP
- UNIX® ファイル・システム
- Windows ファイル・システム
- Content Edition (以前の名称は VeniceBridge)

Web ソース用のクローラー・プラグイン

文書を要求するために起点サーバーに送信される HTTP 要求ヘッダーにフィールドを追加することができます。また、文書のダウンロード後、文書のコンテンツ、セキュリティー・トークン、メタデータを表示することもできます。任意のフィールドを追加、削除、置換することができ、文書の解析を停止することができます。

Web クローラー・プラグインは、プリフェッチと事後解析の 2 つの種類のフィルター操作をサポートします。 Web クローラー・プラグインとして指定できるのは 1 つの Java クラスのみですが、プリフェッチ・プラグインと事後解析プラグインの動作は 2 つの異なる Java インターフェースで定義されており、 Java クラスはインターフェースをいくつでもインプリメントできるので、 Web クローラー・プラグイン・クラスはどちらか一方または両方の動作をインプリメントすることができます。

Web クローラー・プラグインには、2 つの固有のプラグイン・タイプがあります。

プリフェッチ・プラグイン
: プリフェッチ・プラグインは、クローラーが文書をダウンロードする前に呼び出されます。プラグインには、文書の URL、フェッチ・メソッド、HTTP バージョン、および HTTP 要求ヘッダーが渡されます。プラグインは、これらのエレメントを使用して、要求ヘッダーを変更する (例えば、Cookie を追加する) かどうか、またはダウンロードを取り消すかどうかなどを決定します。

事後解析プラグイン

事後解析プラグインは、ダウンロード試行後に呼び出されます。ダウンロードでは、コンテンツを作成したり、コンテンツを解析する必要はありません。プラグインには、文書の URL、クローラーがさまざまなソースから抽出したメタデータ、および文書のコンテンツが渡されます。プラグインは、文書内のこれらの項目を変更するかどうか、文書のコンテンツが解析される前に保管するかどうかを決定します。

Web 以外のデータ・ソース用のクローラー・プラグインの作成

Java クラスを作成して、セキュリティー・トークンの値やエンタープライズ・サーチ索引内の文書に関連したメタデータをプログラマチックに検出することができます。

制約事項

バージョン 8.2 クローラーを使用する文書クローラー・プラグインを使用するには、新規コレクションを作成する必要があります。以前に作成したコレクションで新規プラグインを使用することはできません。

DB2 II OmniFind Edition バージョン 8.2 を使用してクローラー用のセキュリティー・トークン・プラグインを作成している場合、セキュリティー・トークン・プラグインは、そのまま引き続き使用できます。ただし、セキュリティー・トークン・プラグインを文書クローラー・プラグインと一緒に使用することはできません。(API が異なります。) 文書クローラー・プラグインは、セキュリティー・トークン・プラグインのすべての関数と追加メタデータ関数をサポートするので、文書クローラー・プラグインを使用してください。

このタスクについて

以下のエンタープライズ・サーチ・クローラーに対してクローラー・プラグインを構成することができます。

- Content Manager
- DB2
- Exchange Server
- Notes
- NNTP
- UNIX ファイル・システム
- Windows ファイル・システム
- WebSphere Information Integrator Content Edition (以前の名称は VeniceBridge)

Web 文書を変更するには、Web クローラー・プラグインを使用します。

クローラー・プラグインを作成および構成することにより、独自のビジネスおよびセキュリティー・ルールを適用して、文書レベルのセキュリティーを施行し、エンタープライズ・サーチ索引内の文書に関連したクロール・メタデータを追加、更新、削除することができます。

エンタープライズ・サーチ・クローラーを構成して、セキュリティー・トークンおよびメタデータを、クロールする文書に関連付ける場合は、クローラーと共に使用するカスタム Java クラスの名前を指定します。この Java クラスをクローラー・プラグインとして指定すると、クローラーは、クロールする各文書ごとにクラスを呼び出します。

クローラーは、文書ごとに、管理者が指定した文書 ID、セキュリティー・トークン、メタデータを Java クラスに渡します。 Java クラスは、新規または変更されたセキュリティー・トークンおよびメタデータを戻します。

手順

Java クラスを作成し、エンタープライズ・サーチでクローラー・プラグインとして使用するには、次のようにします。

1. com.ibm.es.crawler.plugin.CrawlerPlugin インターフェースを継承し、以下のメソッドをインプリメントします。

   ```
   method init()
   isMetadataUsed()
   term()
   updateDocument()
   ```

   ネーム解決には、以下のいずれかのファイルを使用します。

   - AIX、Linux、Solaris: *ES_INSTALL_ROOT*/lib/dscrawler.jar
   - Windows: *ES_INSTALL_ROOT*¥lib¥dscrawler.jar

2. インプリメントされたコードをコンパイルして、そのコード用の JAR ファイルを作成します。コンパイル時に、クラスパスに dscrawler.jar ファイルを追加します。
3. エンタープライズ・サーチ管理コンソールで、以下を行います。
   a. 該当するコレクションを編集します。
   b. 「クロール」ページを選択して、カスタム Java クラスを使用するクローラーのクローラー・プロパティーを編集します。
   c. 以下の項目を指定します。
      - インプリメントされた Java クラスの完全修飾クラス名
      - JAR ファイルの完全修飾クラスパス
      - Java クラスに必要なすべてのファイルがあるディレクトリー
   d. 編集したクローラーのセッションを停止して再始動します。その後、フル・クロールを開始します。

   管理コンソールでは、セキュリティー・トークン・プラグインのプラグイン・クラスパスの値は無視されます。バージョン 8.2 で提供されていたセキュリティー・トークン・プラグインを使用する場合、プラグインは必要なすべてのライブラリー (JAR ファイル) をコピーする必要があります。

クローラーがプラグインのロード中に停止した場合は、ログ・ファイルを調べて、以下を確認してください。

- クローラーのプロパティー・ページで指定したクラス名とクラスパスが正しいか。
- 必要なすべてのライブラリーがプラグイン・クラスパスに指定されているか。

- クローラー・プラグインが CrawlerPluginException エラーをスローしていないか。

Web クローラー・プラグイン

Web クローラー・プラグインは、プリフェッチ・プラグインと事後解析プラグインの 2 つのタイプのプラグインを提供します。

プリフェッチ・プラグインでは、Java API を使用して、文書を要求するために起点サーバーに送信される HTTP 要求ヘッダーにフィールドを追加することができます。

事後解析プラグインでは、文書が解析およびトークン化される前に、Java API を使用して、文書のコンテンツ、セキュリティー・トークン、メタデータを表示することができます。フィールドを追加、削除、置換したり、文書のパーサーへの送信を停止することができます。

ご使用のプラグインで、プラグインのほかに Java クラスや非 Java ライブラリーやその他のファイルを必要とする場合には、その要件を処理するようにプラグインを作成する必要があります。例えば、プラグインでクラス・ローダーを呼び出して、より多くの Java クラスやロード・ライブラリーを取り込んだり、ネットワーク接続やデータベース接続を作成したり、その他必要なことを行うことができます。

プラグインは、クローラー JVM プロセスの一部として稼働します。例外やエラーは検出されますが、クローラーのパフォーマンスは、プラグイン実行の影響を受けます。最小限の処理を行い、予測されるすべての例外を検出するように、プラグインを作成する必要があります。プラグイン・コードは、マルチスレッド・セーフにしなければなりません。200 の並行ダウンロードがある場合は、プラグインに対して 200 の並行呼び出しが発生することになります。

Web クローラーのプリフェッチ・プラグインの作成

プリフェッチ・プラグインを作成するには、 com.ibm.es.wc.pi.PrefetchPlugin インターフェースをインプリメントする Java クラスを作成します。

プリフェッチ・プラグインを作成するには、次のようにします。

1. com.ibm.es.wc.pi.PrefetchPlugin インターフェースを継承し、以下のメソッドをインプリメントします。

```
public class MyPrefetchPlugin implements com.ibm.es.wc.pi.PrefetchPlugin {
    public MyPrefetchPlugin() { ... }
    public boolean init() { ... }
    public boolean processDocument(PrefetchPluginArg[] args) { ... }
    public boolean release() { ... }
 }
```

   init メソッドは、プラグインのインスタンス化を行う際に 1 度だけ呼び出されます。プラグイン・クラスがあることを指定した場合、クローラーは、開始時にそのクラスをロードし、単一のクラス・インスタンスを作成します。プラグイン・クラスには、引数以外のコンストラクターがなければなりません。クローラーは、クラスのインスタンスを 1 つだけ作成します。クラスのインスタンスの作成後、クローラーは、そのインスタンスを最初に使用する前に、init メソッド

を呼び出します。このメソッドは、クラスのインスタンスがメモリーに入るまで行えない、必要なセットアップ・タスクを行います。

プラグインを使用するべきでない場合、または他のエラーが発生した場合、 init メソッドは false を戻します。クローラーは、このインスタンスをプリフェッチ・プラグインのリストから除去します。init メソッドが true を戻した場合、プラグインは使用可能です。init メソッドは例外をスローできません。

processDocument メソッドは、ダウンロードされる文書ごとに、単一のプラグイン・インスタンスで呼び出されます。クローラーは 1 つから多数に到る非同期に実行されるダウンロード・スレッドを使用するので、このメソッドは複数のスレッドから並行して呼び出されます。

release メソッドは、プラグイン・オブジェクトがシステム・リソースをリリースするか、またはキューに入れられたオブジェクトをフラッシュするために、クローラーが停止する際に 1 度だけ呼び出されます。このメソッドは例外をスローできません。結果が true の場合は、正常終了を意味します。結果が false の場合は、ログに記録されます。

ネーム解決には、以下のいずれかのファイルを使用します。

- AIX、Linux、Solaris: *ES_INSTALL_ROOT*/lib/URLFetcher.jar
- Windows: *ES_INSTALL_ROOT*¥lib¥URLFetcher.jar

2. インプリメントされたコードをコンパイルして、そのコードの JAR ファイルを作成します。コンパイル時に、クラスパスに URLFetcher.jar ファイルを追加します。
3. エンタープライズ・サーチ管理コンソールで、以下を行います。
   a. 該当するコレクションを編集します。
   b. 「クロール」ページを選択して、カスタム Java クラスを使用するクローラーのクローラー・プロパティーを編集します。
   c. エンタープライズ・サーチ管理コンソールで、以下の項目を指定します。
      - インプリメントされた Java クラスの完全修飾クラス名
      - 必要なすべての JAR ファイルを含む、プラグインのクラスパス
   d. 編集したクローラーのセッションを停止して再始動します。その後、フル・クロールを開始します。

エラーが発生した場合、および Web クローラーがプラグインのロード中に停止した場合は、ログ・ファイルを調べて、以下を確認してください。

- クローラーのプロパティー・ページで指定したクラス名とクラスパスが正しいか。
- 必要なすべての JAR ファイルがプラグイン・クラスパスに指定されているか。
- クローラー・プラグインが CrawlerPluginException または他の予期しない例外をスローしていないか。また、プラグインで致命的エラーが発生していないか。

このメソッドはマルチスレッド・セーフになるよう作成する必要があります。これは、同期化ブロックでコンテンツ全体をラッピングすることにより行えますが、そ

うすることにより一度に 1 つのスレッドしかメソッドを実行できなくなるため、クローラーがプラグイン操作時に単一スレッドになり、パフォーマンスのボトルネックになります。

メソッドをマルチスレッド・セーフにするためにより良い方法は、すべての状態にローカル (スタック) 変数を使用する方法です。これにより、グローバル・データの量が最小化され、スレッド間で共有されているオブジェクトにアクセスする間のみ同期化されます。このメソッドは例外をスローできません。文書の処理が正常に行われた場合は true を、問題が発生した場合は false を戻します。false 戻り値は、クローラーによって URL と共にログに記録されます。

プリフェッチ・プラグインの例

プリフェッチ・プラグインを使用して、文書がダウンロードされる前に、Cookie を HTTP 要求ヘッダーに追加します。

```
package com.mycompany.ofpi;
// OmniFind plug-ins

 public class MyPrefetchPlugin implements com.ibm.es.wc.pi.PrefetchPlugin {
    public MyPrefetchPlugin() { }
    public boolean init() { return true; }
    public boolean release() { return true; }
    public boolean processDocument(PrefetchPluginArg[] args) {
       PrefetchPluginArg1 arg = (PrefetchPluginArg1)args[0];
              if (arg.getURL().startsWith("http://mysite.com/backups"))
          return false;
       if (arg.getURL().startsWith("http://special.mysite.com/")) {
          String httpHeader = arg.getHTTPHeader();
          arg.setHTTPHeader( httpHeader + "Cookie: SPECIAL=cookievalue¥r¥n");
       }
       return true;
    }
 }
```

この例では、以下のことを示しています。

- プラグインに渡される引数配列内の最初のエレメント ([0]) は、PrefetchPluginArg1 タイプのオブジェクトで、 PrefetchPluginArg インターフェースを拡張するインターフェースです。これは、プリフェッチ・プラグインに渡される唯一の引数であり、唯一の引数タイプです。これに安全にキャストすることができます。完全に保護するには、キャストを try/catch ブロックで囲み、ClassCastException オブジェクトを探すか、または最初に「instanceof」テストを行います。
- 引数を設定したら、PrefetchPluginArg1 インターフェースの任意のメソッドを呼び出すことができます。 getURL メソッドは、クローラーがダウンロードする文書の URL (ストリング形式) を戻します。この URL を使用して、文書が Cookie などの要求ヘッダー内の追加情報を必要とするかどうかを決定します。
- getHTTPHeader メソッドは、クローラーが文書をダウンロードするために送信する HTTP 要求ヘッダーのすべてのコンテンツが含まれているストリングを戻します。プラグインは、必要に応じて、このヘッダーを検査および変更することができます。例えば、HTTP 要求ヘッダーに有効な、単一の Cookie をヘッダーやその他の情報に追加できます。また、この情報の一部を除去することもできます。ヘッダーを変更する場合は、 HTTP プロトコル要件に準拠しなければなりませ

ん。例えば、各行は CRLF シーケンスで終了しなければならず、ヘッダーは ISO-8859-1 エンコード方式を使用しなければなりません。

- processDocument メソッドは、クローラーがダウンロードする各文書ごとに 1 回呼び出されます。processDocument メソッドが false を戻すと、その結果は無視されます。 true を戻すと、クローラーは行った処理を検査します。ダウンロードを停止するために、プリフェッチ・プラグインは setFetch(false) メソッドを呼び出します。

プリフェッチ・プラグインのデプロイ

クローラーに対してプラグイン・クラスを識別するには、JAR ファイルにクラスを書き込み、エンタープライズ・サーチ管理コンソールのクローラー・ウィンドウで、プラグイン・クラスの名前と JAR ファイルのロケーションを入力します。プラグイン・クラスの完全修飾名と、JAR ファイルの絶対パス名を入力する必要があります。

プリフェッチ・プラグインをデプロイするには、次のようにします。

1. JAVA ファイルをコンパイルして、それに対応する JAR ファイルを作成します。JAR ファイルには、サポートするクラスやリソースも含まれるので、ofplugins.jar のような名前にします。
2. この JAR ファイルをエンタープライズ・サーチ・インストール済み環境で Web クローラーが稼動するコンピューターにコピーします。プラグインを使用可能にする際に、管理コンソールのクローラー・ウィンドウで JAR ファイルの絶対パスを入力します。
3. エンタープライズ・サーチ管理コンソールで、以下の項目を指定します。
   - インプリメントされた Java クラスの完全修飾クラス名。例えば、com.mycompany.ofpi.MyPrefetchPlugin など。
   - JAR ファイルの完全修飾クラスパス

   クローラーが開始され、プラグイン JAR ファイルとクラス名を検出したら、クローラーは、引数以外のコンストラクターを使用して、JAR をロードし、クラスをインスタンス化します。その後、クローラーは、init メソッドを呼び出して、インスタンスを初期化します。 init メソッドから true が戻されると、プリフェッチ・プラグインのリストにプラグインが追加されます。

Web クローラーの事後解析プラグインの作成

事後解析プラグインを使用して、文書のダウンロード後に、文書のコンテンツ、セキュリティー・トークン、およびメタデータを表示することができます。これにより、フィールドを追加、削除、置換したり、文書のパーサーへの送信を停止することができます。

事後解析プラグインを作成するには、 com.ibm.es.wc.pi.PostparsePlugin インターフェースをインプリメントする Java クラスを作成します。例えば以下のようにします。

```
public class MyPostparsePlugin implements
com.ibm.es.wc.pi.PostparsePlugin {
   public MyPostparsePlugin () { ... }
   public boolean init() { ... }
   public boolean processDocument(PostparsePluginArg[] args) { ... }
   public boolean release() { ... }
}
```

プラグイン・クラスは両方のインターフェースをインプリメントできますが、必要な init メソッドおよび release メソッドはそれぞれ 1 つだけです。クラスがプリフェッチ処理と事後解析処理の両方を行う場合は、両方のタスクについて、リソースを初期化およびリリースする必要があります。init メソッドと release メソッドは、両方とも 1 度だけ呼び出されます。

processDocument メソッドは、ダウンロードされる URL ごとに、単一のプラグイン・インスタンスで呼び出されます。すべてのダウンロードでコンテンツが戻されるわけではありません。呼び出し時に何を行うかを判別するために、HTTP 戻りコード(200、302、404 など) がプラグインによって使用されます。コンテンツが獲得され、コンテンツが HTML 構文解析に適している場合、コンテンツはパーサーに渡され、プラグイン呼び出し時に構文解析結果が使用可能になります。

事後解析プラグインの例

以下の例は、特定のサイトからダウンロードされた文書と共にクローラーが送信するメタデータにセキュリティー ACL を追加する方法を示しています。クローラーが文書をパーサーの入力バッファーに書き込む直前に、事後解析プラグインを使用して、これらの ACL を追加することができます。

```
package com.mycompany.ofpi;  // OmniFind plug-ins

import com.ibm.es.wc.pi.*;

 public class MyPostparsePlugin implements PostparsePlugin {
    public MyPostparsePlugin() { }
    public boolean init() { return true; }
    public boolean release() { return true; }
    public boolean processDocument(PostparsePluginArg[] args) {
              try {
           PostparsePluginArg1 arg = (PostparsePluginArg1)args[0];
                  if (arg.getURL().startsWith("http://mysite.com/users/")) {
                     // Extract user name from URL; look up appropriate tokens.
              String acls = // Create a comma-separated list of the
                            // additional ones.
              arg.addSecurityACLs(acls);
           }
           return true;
               } catch (Exception e) {
                  return false;  // disregard returned results
               }
    }
 }
```

事後解析プラグインを使用して、クロールされた文書に新規のメタデータ・フィールドを追加することもできます。例えば、一部の文書にキーワードが含まれている場合、クローラーがさまざまな「検索可能」属性で実行されている際に、検索したいストリングが含まれている検索レコードに「MyUserSpecificMetadata」というメタデータ・フィールドを追加することができます。以下の例は、メタデータ・フィールドの追加方法を示しています。

```
package com.mycompany.ofpi;  // OmniFind plug-ins

import com.ibm.es.wc.pi.*;   // Plug-in API

public class MyPostparsePlugin implements PostparsePlugin {

   public MyPostparsePlugin() { }
   public boolean init() { return true; }
```

```
   public boolean release() { return true; }
   public boolean processDocument(PostparsePluginArg[] args) {
      try {
         PostparsePluginArg1 arg = (PostparsePluginArg1)args[0];
         if (arg.getContent() != null && arg.getContent().length > 0) {
            String content = new String( arg.getContent(), arg.getEncoding() );
            if (content != null && content.indexOf(keyword) > 0) {
               final String userdata = null; // look up string by keyword.
               FieldMetadata mf = new FieldMetadata(
                  "MyUserSpecificMetadata" // field name
                  userdata,                // field value
                  false,                   // searchable?
                  true,                    // field-searchable?
                  false,                   // parametric-searchable?
                  true,                    // can be extracted by search?
                  "MetadataPreferred",     // metadata value rather
                                           // than content
                  false);                  // show in summary?
               addMetadataField(mf);    // Add it to the list.
               return true;             // Use results.
            }
         }
         return false;  // ignore results
      } catch (Exception e) {
         return false;  // disregard returned results
      }
   }
}
```

文書コンテンツは、プラグイン引数 (arg.getContent) から入手できます。クローラーが検出したエンコードが有効になります。コンテンツとエンコードを使用して、ストリングを作成できます。キーワード (content.indexOf(...)) を検索し、新規データ (userdata = ...) を関連付け、そのデータを新規フィールドのコンテンツとして挿入します。

新規メタデータ・フィールドを定義するには、FieldMetadata オブジェクトのインスタンスを作成し、そのフィールド値を設定します。

第 4 章 データ・リスナー

現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

データ・リスナーは、クライアント・アプリケーションからの要求を受け入れる、エンタープライズ・サーチ・コンポーネントです。要求を受けて、コレクションにデータを追加したり、コレクションからデータを除去します。通常、1 つのコレクションに対して 1 つ以上のクローラーを作成します。これらのクローラーは、Web や DB2 Content Manager データベースなどの特定のデータ・ソースからデータを検索します。データ・リスナー・クライアント・アプリケーションを使用すると、クローラーを作成せずに、コレクションにページを追加することができます。また、コレクションから Uniform Resource Identifier (URI) を除去したり、コレクションの Web クローラーに対して Uniform Resource Locator (URL) へのアクセスまたは再アクセスを指示することもできます。

図 1 に、データ・リスナー・クライアント・アプリケーションを使用しない場合のエンタープライズ・サーチ・システムの検索動作の概要を示します。ユーザーは検索コンポーネントに照会を実行し、索引は定期的に検索コンポーネント内のデータをリフレッシュします。 (これらのコンポーネントは、1 つのサーバー上にあっても、複数の接続されたサーバー上にあってもかまいません。このトピックの説明では、複数のサーバー・システムを使用していることを想定しています。)

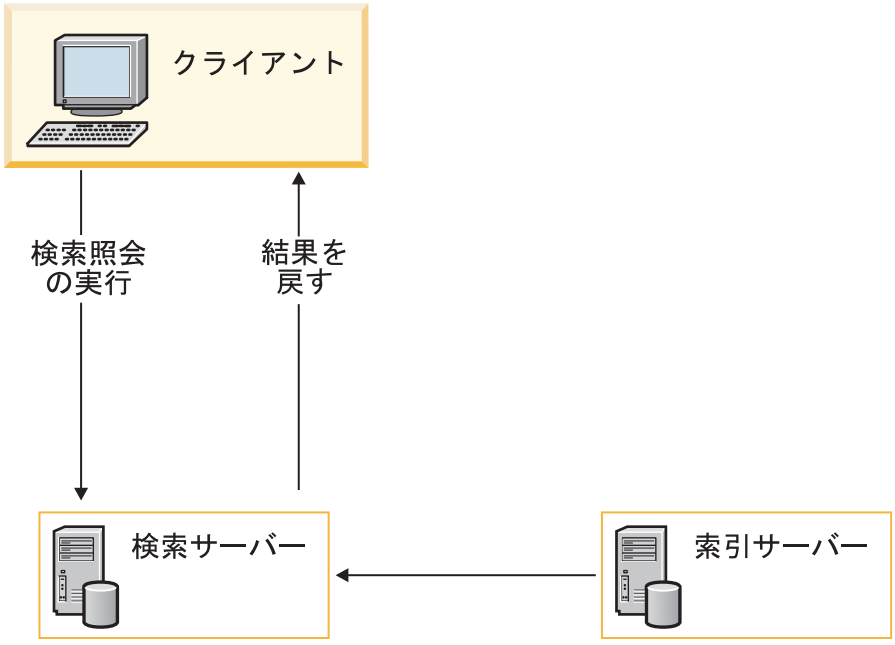


図 1. 照会はエンタープライズ・システムにどのように送信されるか

データ・リスナー・クライアント・アプリケーションを使用して、データ・リスナーに要求をサブミットすることができます。要求のタイプに応じて、データ・リスナーは、後続処理のために、その要求を他のエンタープライズ・サーチ・コンポーネントに送信します。

クライアント・アプリケーションがデータ・リスナーに接続すると、データ・リスナーは、指定されたパスワードがクライアントID およびパスワードに対して有効であるか検査します。また、データ・リスナーは、クライアント・アプリケーションが、指定されたコレクションに対してデータの追加または除去を許可されているかについても検査します。

図 2 に、データ・リスナー・クライアント・アプリケーションがエンタープライズ・サーチ・システムに要求を送信する方法を示します。

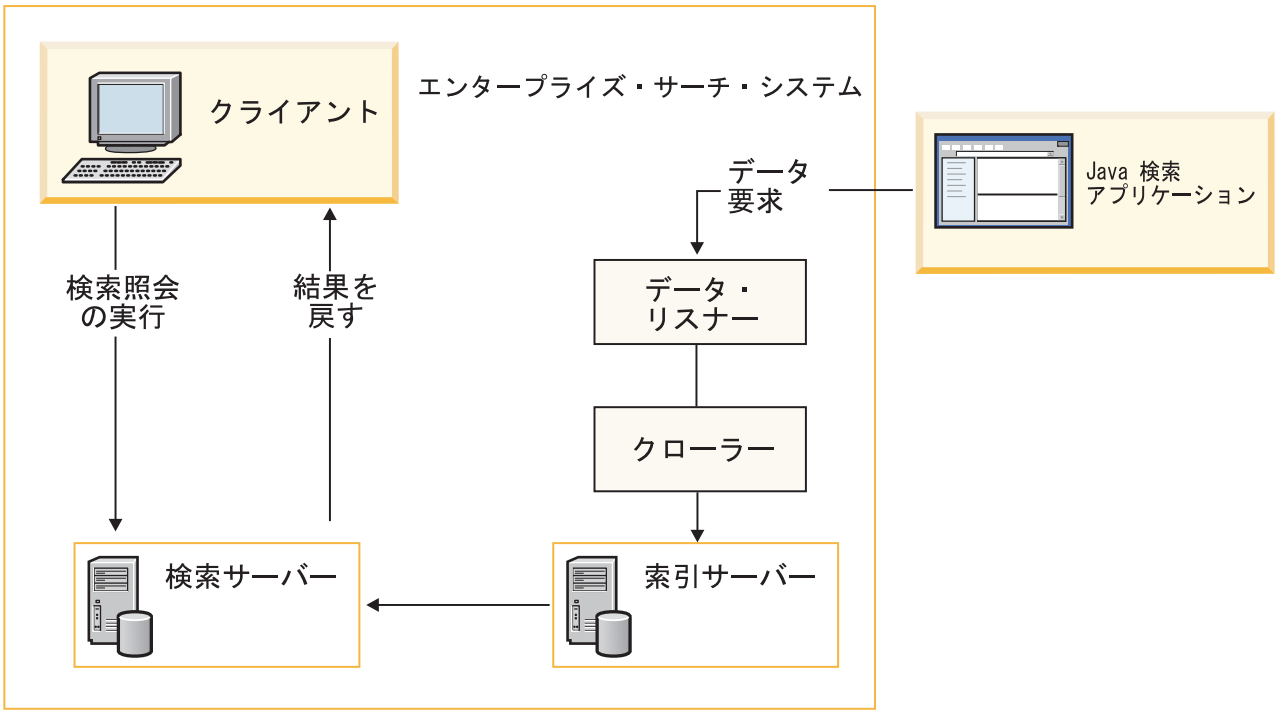


図 2. データ・リスナー API はエンタープライズ・サーチ・システムとどのように連動するか

データ・リスナー API は、以下の JAR ファイルにパッケージされています。

- es.dl.client.jar
- es.oss.jar

これらの JAR ファイルは、ご使用のオペレーティング・システムによって、以下のディレクトリーにあります。

- AIX、Linux、 Solaris: *ES_INSTALL_ROOT*/IBM/es/lib
- Windows: *ES_INSTALL_ROOT*¥IBM¥es¥lib

JAR ファイルをデフォルトのインストール・ディレクトリーにインストールしている場合は、以下の値を CLASSPATH 変数に追加してください。

- AIX、Linux、Solaris: opt/IBM/es/lib/es.oss.jar:/opt/IBM/es/lib/es.dl.client.jar

- Windows: c:/Program Files/IBM/es/lib/es.oss.jar;c:/Program Files/IBM/es/lib/es.dl.client.jar

JAR ファイルをデフォルトのディレクトリーにインストールしていない場合は、適切なディレクトリーを CLASSPATH に追加してください。

データ・リスナー API によるデータの除去

データ・リスナーを使用して、特定の URI または URI のパターンを除去することによって、文書をコレクションから除去することができます。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

特定の URI を除去すると、除去した URI は即時に検索コンポーネントからなくなります。 URI パターンを除去すると、次回の索引再編成時に、除去したパターンに一致する文書が除去されます。

検索コンポーネントおよび索引から URI を除去する

データ・リスナー API を使用して、特定の URI を除去することができます。図 3 で示しているように、要求がデータ・リスナーに送信され、URI が索引サーバーおよび検索サーバーから除去されます。

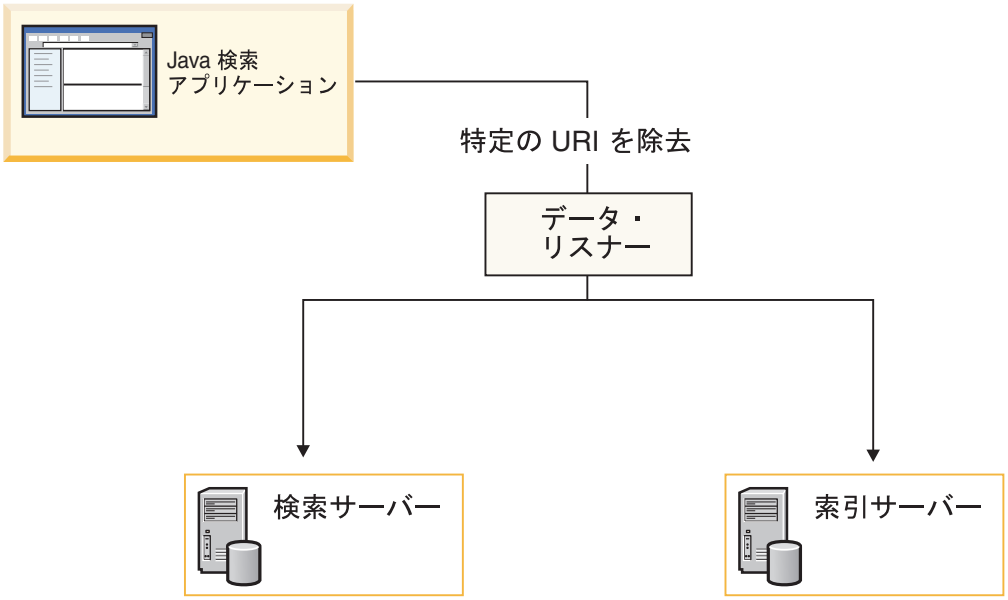


図 3. 特定の URI の除去

removeURI メソッドを使用して、指定されたコレクションからデータを削除します。

索引から URI パターンを除去する

データ・リスナー API を使用して、 URI パターンを除去することができます。例えば、URI として http://www.ibm.com/*.html を指定して URI パターンの除去要求をサブミットすると、索引サーバーは以下の URL を除去します。

- http://www.ibm.com/home.html
- http://www.ibm.com/family.html
- http://www.ibm.com/pics.html

重要: URI パターンの除去要求は、注意して使用してください。除去されたコンテンツはすぐにリカバリーできません。クローラーまたはデータ・リスナー・クライアント・アプリケーションによって、コンテンツを索引に再度、追加する必要があります。

データ・リスナー API によるデータの追加

データ・リスナーを使用して、コンテンツと共に URI を追加したり、URL にアクセスまたは再アクセスすることにより、検索コレクションにデータを追加あるいはプッシュすることができます。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

コンテンツと共に URI を追加する

データ・リスナー API を使用して、 pushData メソッドに content パラメーターを組み込むことによって、URI とそのコンテンツを追加することができます。このメソッドは、指定されたデータを指定されたコレクションに送信します。

追加された URI とそのコンテンツは、索引の更新または再編成後に検索で使用可能になります。

URL にアクセスまたは再アクセスする

データ・リスナー API を使用して、revisitURLs メソッドによって、特定の URL または URL パターンをコレクションの Web クローラーに追加することができます。このメソッドは、Web コンテンツに対してのみ有効です。クローラーは、URL をクロールし、そのデータ (URL、メタデータ、およびコンテンツ) をコレクションに追加します。URL は、クローラーにとって新規のもの (アクセス) でも、クローラーが既に検出しているもの (再アクセス) でもかまいません。 URL パターンを指定すると、指定されたパターンに一致する、クローラーが既に検出済みの URL に再アクセスすることを要求します。

データ・リスナー・クライアント・アプリケーションの作成

データ・リスナーに要求を送信するには、クライアント・アプリケーションを作成して、コレクションに関する具体的なプロパティー (クライアント・アプリケーションがターゲット・コレクションのコレクションID への要求に指定しなければならない) と認証データを指定します。認証データには、クライアント ID とパスワードが含まれます。クライアント ID は、コレクションを更新する権限を持っている必要があります。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

データ・リスナー・アプリケーションを作成するには、以下のようにします。

1. 変更するコレクションを決定します。

2. そのコレクションの ID を入手します。コレクション ID は、エンタープライズ・サーチ管理コンソールでコレクションを作成した際に作成されます。コレクション ID は「コレクション設定」ページで調べることができます。エンタープライズ・サーチ管理コンソールで「コレクション」ビューの「一般」ページから「**コレクション設定の表示**」をクリックします。エンタープライズ・サーチ管理コンソールにログインするか、またはこの ID をエンタープライズ・サーチの管理者に問い合わせる必要があります。
3. コレクションのクライアント ID とパスワードを指定します。クライアントとは、コレクションにアクセスするデータ・リスナー・アプリケーションです。アクセスするコレクションに対するクライアント ID とパスワードを作成して、コレクションにアクセスする許可をデータ・リスナー・アプリケーションに与える必要があります。クライアント ID とパスワードを指定するには、エンタープライズ・サーチの管理者に依頼するか、または以下のステップに従ってください。
   a. エンタープライズ・サーチ管理コンソールにログインします。
   b. 「**システム**」 → 「**データ・リスナー**」 → 「**データ・リスナーの構成**」の順にクリックします。
   c. 「**データ・リスナー・クライアント ID の追加**」をクリックします。
   d. データ・リスナー・クライアント ID、パスワードを入力して、コレクションを選択します。「**OK**」をクリックします。

      さらに、他のコレクションに対するクライアント ID とパスワードを追加する場合は、「**データ・リスナー・クライアント ID の追加**」 をクリックして、同様の操作を行います。

第 5 章 クローラー・プラグイン API リファレンス

非 Web 文書用のクローラー・プラグイン API

クローラーが開始されると、プラグイン・プロセスが fork されます。CrawlerPlugin オブジェクトがデフォルトのコンストラクターを使用してインスタンス化され、init メソッドと isMetadataUsed メソッドが 1 度ずつ呼び出されます。クローラーが停止すると、 term メソッドが呼び出され、オブジェクトが破棄されます。

com.ibm.es.crawler.plugin.CrawlerPlugin インターフェース

プラグインを作成するには、com.ibm.es.crawler.plugin.CrawlerPlugin インターフェースを使用します。

init メソッド

クローラー・セッションの開始時に 1 度だけ呼び出されます。

構文

```
public void init() throws CrawlerPluginException
```

スロー

クローラーを停止する場合は `CrawlerPluginException`。

isMetadataUsed メソッド

このメソッドは、クローラー・セッションが開始されると、init メソッドの後に 1 回呼び出されます。メタデータを使用または変更する場合は、値 true を戻します。メタデータを使用しない場合は、値 false を戻します。値が false の場合、CrawledData.getMetadataList メソッドは常時 NULL 値を戻すため、パフォーマンスが向上します。

構文

```
public boolean isMetadataUsed()
```

戻り

メタデータを使用する場合は `true`。それ以外の場合、メソッドは `false` を戻します。

term メソッド

クローラー・セッションの停止時に 1 度だけ呼び出されます。

構文

```
public void term() throws CrawlerPluginException
```

スロー

クローラーを停止する場合は `CrawlerPluginException`。

updateDocument メソッド

クローラーによって呼び出されます。クローラーがマルチスレッド処理を行う場合、並行して updateDocument メソッドが呼び出されます。文書の索引付けを行わない場合は、NULL 値を戻します。

構文

```
public com.ibm.es.cawler.plugin.CrawledData updateDocument
(CrawledData crawledData) throws CrawlerPluginException
```

戻り

CrawledData オブジェクト

スロー

クローラーを停止する場合は CrawlerPluginException。

com.ibm.es.crawler.plugin.CrawledData クラス

メタデータの URI、セキュリティー・トークン、リストを取得します。この情報を更新する場合は、このオブジェクトを操作します。このクラスは、CrawlerPlugin クラスの updateDocument メソッドでパラメーターおよび戻り値として使用されます。

getURI メソッド

クロールされる文書の URI を戻します。

構文

```
public String getURI()
```

戻り

クロールされる文書の URI。

getMetadataList メソッド

クロールされる文書のメタデータが含まれている FieldMetadata オブジェクトのリストを戻します。isMetadataUsed メソッドの戻り値が false の場合、このメソッドは常時ヌルを戻します。メタデータを追加、更新、削除する場合は、 List オブジェクトの FieldMetadata オブジェクトを直接操作します。

構文

```
public java.util.List getMetadataList()
```

戻り

FieldMetadata オブジェクトのリスト。

getSecurityTokens メソッド

オリジナル・セキュリティー・トークンを戻します。戻り値がヌルの場合、クロールされる文書は全ユーザーに公開されるように指定されます。

構文

```
public String getSecurityTokens()
```

戻り

オリジナル・セキュリティー・トークン (コンマで区切られたストリング)

setSecurityTokens メソッド

セキュリティー・トークンをパラメーターによって置き換えます。このメソッドが呼び出されない場合は、エンタープライズ・サーチ管理者が設定したセキュリティー・トークンが使用されます。トークン・パラメーターがヌルの場合、クロールされる文書は全ユーザーに公開されるように指定されます。

構文

```
public void setSecurityTokens(String tokens)
```

パラメーター

tokens
新規セキュリティー・トークン (コンマで区切られたストリング)

com.ibm.es.crawler.plugin.FieldMetadata クラス

メタデータ情報を取得します。

構文

```
public class FieldMetadata {
    public FieldMetadata(String fieldName, String value,
                         boolean searchable, boolean fieldSearchable,
                         boolean parametricSearchable,
                         boolean asMetadata, String resolveConflict,
                         boolean content);
    public String getFieldName();
    public void setFieldName(String fieldName);
    public boolean isSearchable();
    public void setSearchable(boolean searchable);
    public boolean isFieldSearchable();
    public void setFieldSearchable(boolean fieldSearchable);
    public boolean isParametricSearchable();
    public void setParametricSearchable(boolean parametricSearchable);
    public boolean isAsMetadata();
    public void setAsMetadata(boolean asMetadata);
    public String getResolveConflict();
    public void setResolveConflict(String resolveConflict);
    public boolean isContent();
    public void setContent(boolean content);
}
```

METADATAPREFERRED 定数

構文

```
public static final String METADATAPREFERRED
```

CONTENTPREFERRED 定数

構文

```
public static final String CONTENTPREFERRED
```

COEXIST 定数

構文

```
public static final String COEXIST
```

FieldMetadata コンストラクター

フィールド・メタデータ・オブジェクトを作成します。

構文

```
public FieldMetadata(String fieldName, String value)
public FieldMetadata(String fieldName,
                     String value,
                     boolean isSearchable,
                     boolean isFieldSearchable,
                     boolean isParametricSearchable,
                     boolean isAsMetadata,
                     String  resolveConflict,
                     boolean isContent)
```

パラメーター

fieldName
検索フィールド名

value
フィールドの値。

isSearchable
このフィールドが検索可能な場合は true。

isFieldSearchable
このフィールドがフィールド検索可能な場合は true。

isParametricSearchable
このフィールドがパラメトリック検索可能な場合は true。

isAsMetadata
検索結果のメタデータを使用する場合は true。

resolveConflict
使用するメタデータ・ソース。メタデータが、添付文書のメタデータ・フィールドと同じ名前である場合に、使用するメタデータ・ソースを指定します。このパラメーターは METADATAPREFERRED、CONTENTPREFERRED、COEXIST のいずれかでなければなりません。

isContent
このフィールドを検索結果のサマリーとして表示できる場合は true。

スロー

パラメーターが無効な場合、IllegalArgumentException。

getInternalFieldName メソッド

データ・ソース・フィールドの内部またはオリジナル・フィールド名を戻します。

構文

```
getInternalFieldName
```

戻り

データ・ソース・フィールドのオリジナル (ネイティブ) フィールド名。

getFieldName メソッド

フィールドの値を戻します。

構文

```
public String getValue();
```

戻り

フィールドの値。

isSearchable メソッド

フィールドが検索可能かどうかを指定します。

構文

```
public boolean isSearchable()
```

戻り

このフィールドが検索可能な場合は true。

isFieldSearchable メソッド

構文

```
public boolean isFieldSearchable()
```

戻り

このフィールドがフィールド名で検索できる場合は true。

isParametricSearchable メソッド

構文

```
public boolean isParametricSearchable()
```

戻り

このフィールドがフィールド名で検索できる場合は true。

isAsMetadata メソッド

フィールドを検索アプリケーションから抽出できるどうかを指定します。

構文

```
public boolean isAsMetadata()
```

戻り

このフィールドを検索アプリケーションから抽出できる場合は true。

getResolveConflict メソッド

メタデータが、添付文書のメタデータ・フィールドと同じ名前である場合に、使用するメタデータ・ソースを指定します。

構文

```
public String getResolveConflict()
```

戻り

使用するメタデータ・ソース。メタデータが添付文書のメタデータ・フィールドと同じ名前である場合に、メソッドは使用するメタデータ・ソースを指定します。パラメーターは METADATAPREFERRED、CONTENTPREFERRED、COEXIST のいずれかでなければなりません。

isContent メソッド

フィールドを検索結果のサマリーとして表示できるかどうかを指定します。

構文

```
public boolean isContent()
```

戻り

このフィールドを検索結果のサマリーとして表示できる場合は `true`。

setFieldName メソッド

フィールド名を設定します。

構文

```
public void setFieldName(String fieldname)
```

パラメーター

fieldname
フィールド名。

setValue メソッド

フィールドの値を設定します。

構文

```
public void setValue(String value)
```

パラメーター

value
フィールドの値。

setSearchable メソッド

フィールドが検索可能かどうかを指定します。

構文

```
public void setSearchable(boolean b)
```

パラメーター

b このフィールドが検索可能な場合は true。

setFieldSearchable メソッド

フィールドがフィールド名で検索可能かどうかを指定します。

構文

```
public void setFieldSearchable(boolean b)
```

パラメーター

b このフィールドがフィールド名で検索できる場合は true。

setParametricSearchable メソッド

フィールドがパラメトリック検索可能かどうかを指定します。

構文

```
public void setParametricSearchable(boolean b)
```

パラメーター

b このフィールドがパラメトリック検索可能な場合は true。

setAsMetadata メソッド

フィールドを検索アプリケーションから抽出できるどうかを指定します。

構文

```
public void setAsMetadata(boolean b)
```

パラメーター

b このフィールドを検索アプリケーションから抽出できる場合は true。

setResolveConflict メソッド

メタデータが、添付文書のメタデータ・フィールドと同じ名前である場合に、使用するメタデータ・ソースを指定します。

構文

```
public void setResolveConflict(String resolveConflict)
```

パラメーター

resolveConflict
使用するメタデータ・ソース。メタデータが、添付文書のメタデータ・フィールドと同じ名前である場合に、使用するメタデータ・ソースを指定します。パラメーターは METADATAPREFERRED、CONTENTPREFERRED、COEXIST のいずれかでなければなりません。

スロー

パラメーターが無効な場合、IllegalArgumentException。

setContent メソッド

フィールドを検索結果のサマリーとして表示できるかどうかを指定します。

構文

```
public void setContent(boolean b)
```

パラメーター

b このフィールドを検索結果のサマリーとして表示できる場合は `true`。

第 6 章 PrefetchPlugin インターフェース

プリフェッチ・プラグインは、PrefetchPlugin インターフェースをインプリメントする必要があります。 processDocument メソッドは、PrefetchPluginArg インターフェースをインプリメントする引数を指定して呼び出されます。ユーザーのコードは、PrefetchPluginArg インターフェースを使用しますが、インプリメントしません。

PrefetchPlugin インターフェース

```
package com.ibm.es.wc.pi;
public interface PrefetchPlugin {
    public boolean init();
    public boolean processDocument(PrefetchPluginArg[] args);
    public boolean release();
}
```

init メソッド

プラグインのインスタンス化を行う際に 1 度だけ呼び出されます。プラグインは、操作を継続する場合は、値 true を戻す必要があります。オペレーションからプラグイン自体を除去する場合は、プラグインで値 false を戻します。戻り値はログに記録されます。

構文

```
public boolean init()
```

戻り

ブール値: true または false。

processDocument メソッド

配列への参照をプラグインに渡します。配列には、PrefetchPluginArg1 をインプリメントする単一のオブジェクトが含まれています。

引数には、プラグインで変更できる項目を定義するデータが含まれています。プラグインで、これらの値を検査して、変更したり、置き換えたりすることができます。プラグインは、処理が正常に終了した場合は値 true を戻し、結果が無効になるような問題があった場合は値 false を戻します。プラグインが値 true を戻し、戻された引数内のデータが有効な場合、オリジナル・データに代わって、そのデータが要求ヘッダーの構成に使用されます。プラグインが値 false を戻した場合、またはデータが無効であったり、データが戻されなかった場合は、オリジナル・ヘッダーのコンテンツが使用されます。

構文

```
public boolean processDocument(PrefetchPluginArg[] args)
```

release メソッド

クローラーの停止時に 1 度だけ呼び出されます。

構文

```
public boolean release()
```

戻り

リリース値。この値はログに記録されます。

第 7 章 PrefetchPluginArg1 インターフェース

PrefetchPluginArg1 インターフェースは、PrefetchPluginArg インターフェースの拡張版です。エンタープライズ・サーチ用のプラグインは、PrefetchPluginArg1 インターフェースをインプリメントする 1 つの引数を受け取ります。この引数の値を変更することができます。変更は、プラグインの processDocument メソッドの戻り時に有効になります。

構文

```
public interface PrefetchPluginArg1 extends PrefetchPluginArg {
    public String getURL();
    public void setURL(String u);
    public String getHTTPHeader();
    public void setHTTPHeader(String h);
    public boolean doFetch();
    public void setFetch(boolean yesno);
}
```

getURL メソッド

このプラグイン引数が参照する URL (ストリング) を戻します。これは、ダウンロードされる文書です。

構文

```
public String getURL()
```

戻り

URL (ストリング)。

setURL メソッド

プラグイン引数が参照する URL を設定します。このメソッドを変更すると、ダウンロード URL が変更されます。

構文

```
public void setURL(String u)
```

getHTTPHeader

HTTP 要求ヘッダー全体をストリングとして戻します。このヘッダーには、復帰改行 (CRLF) が含まれていますが、最後にある 2 番目の (終了) CRLF は戻されません。

構文

```
public String getHTTPHeader()
```

戻り

HTTP 要求ヘッダーをストリングとして戻します。

setHTTPheader

HTTP ヘッダー・ストリングを設定します。戻されるストリングは、各行の終わりと最終行に復帰改行 (CRLF) があり、連続した CRLF が含まれていない、HTTP 要求ヘッダーに適したフォーマットでなければなりません。

構文

```
public void setHTTPHeader(String h)
```

doFetch メソッド

フェッチャーが問題になっている文書をダウンロードする (true) か否か (false) を戻します。

構文

```
public boolean doFetch()
```

戻り

ブール値: true または false。

setFetch メソッド

フェッチャーが文書をダウンロードするかどうかを設定します。このメソッドを false に設定すると、クローラーは指定された文書をダウンロードしません。

構文

```
public void setFetch(boolean yesno)
```

第 8 章 PostparsePlugin インターフェース

事後解析プラグインは、PostparsePlugin インターフェースをインプリメントする必要があります。

構文

```
package com.ibm.es.wc.pi;
public interface PostparsePlugin {
    public boolean init();
    public boolean processDocument(PostparsePluginArg[] args);
    public boolean release();
}
```

init メソッド

プラグインのインスタンス化を行う際に 1 度だけ呼び出されます。プラグインは、操作を継続する場合は、値 true を戻す必要があります。オペレーションからプラグイン自体を除去する場合は、プラグインで値 false を戻します。戻り値はログに記録されます。

構文

```
public boolean init()
```

戻り

ブール値: true または false。

processDocument メソッド

配列への参照をプラグインに渡します。配列には、PostparsePluginArg1 インターフェースをインプリメントする単一のオブジェクトが含まれています。

引数には、プラグインで表示および変更することができる項目を定義するデータが含まれています。プラグインで、これらの値を検査して、変更したり、置き換えたりすることができます。プラグインは、処理が正常に終了した場合は値 true を戻し、結果が無効になるような問題があった場合は値 false を戻します。プラグインが値 true を戻し、戻された引数内のデータが有効な場合は、そのデータがクロールされた文書の処理に使用されます。プラグインが値 false を戻した場合、またはデータが無効であった場合は、プラグインの変更は無視されます。

構文

```
public boolean processDocument(PostparsePluginArg[] args);
```

release メソッド

クローラーの停止時に 1 度だけ呼び出されます。

構文

```
public boolean release()
```

戻り

リリース値。この値はログに記録されます。

第 9 章 PostparsePluginArg1 インターフェース

processDocument メソッドは、PostparsePluginArg1 インターフェースをインプリメントする単一の引数を指定して呼び出されます。ユーザーのプラグインは、そのインターフェースのメソッドを使用して、引数オブジェクトの値を取得および設定します。変更は、プラグインの processDocument メソッドの戻り時に有効になります。

構文

```
package com.ibm.es.wc.pi;
public interface PostparsePluginArg1 extends PostparsePluginArg {
    public String getURL();
    public String getSecurityACLs();
    public void addSecurityACLs(String acls); // comma-separated list
    public void setSecurityACLs(String acls); // comma-separated list
    public void addMetadataField(FieldMetadata md);
    public java.util.ArrayList getMetadataFields();
    public void setMetadataFields(java.util.ArrayList md);
    public byte[] getContent();
    public void setContent(byte[] b);
    public String getContentType();
    public void setContentType(String contentType);
    public String getEncoding();
    public void setEncoding(String e);
    public String getHTTPHeader();
    public String getLanguage();
    public void setLanguage(String language);
    public boolean doSave();
    public void setSave (boolean yesno);
    public void addLink(String url);
   public java.util.ArrayList getLinks();
   public void setLinks(java.util.ArrayList urls);
}
```

PostparsePluginArg1 インターフェースのメタデータ関連メソッド (add、get、set) は、 以下の API に示すように、FieldMetadata タイプの引数を取ります。

```
package com.ibm.es.wc.pi;
public class FieldMetadata {
    public FieldMetadata(String fieldName, String value,
           boolean searchable, boolean fieldSearchable,
           boolean parametricSearchable, boolean asMetadata,
           String resolveConflict, boolean content);
    public String getFieldName();
    public void setFieldName(String fieldName);
    public boolean isSearchable();
    public void setSearchable(boolean searchable);
    public boolean isFieldSearchable();
    public void setFieldSearchable(boolean fieldSearchable);
    public boolean isParametricSearchable();
    public void setParametricSearchable(boolean parametricSearchable);
    public boolean isAsMetadata();
    public void setAsMetadata(boolean asMetadata);
    public String getResolveConflict();
    public void setResolveConflict(String resolveConflict);
    public boolean isContent();
    public void setContent(boolean content);
}
```

プラグインは、processDocument メソッドに渡されるメタデータ仕様の値を検査および変更することができます。また、プラグインは、メタデータ・フィールドを削除あるいは追加することもできます。変更は、processDocument メソッドの戻り時に有効になります。

getURL メソッド

このプラグイン引数が参照する URL (ISO-8859-1 ストリング) を戻します。これは、ダウンロードされた文書です。

構文

```
public String getURL()
```

戻り

URL (ISO-8859-1 ストリング) を戻します。

getSecurityACLs メソッド

現行の文書のメタデータの一部として書き込まれるセキュリティー ACL のコンマで区切られたリストが含まれている UTF-8 ストリングを戻します。

構文

```
public String getSecurityACLs()
```

戻り

UTF-8 ストリングを戻します。

addSecurityACLs メソッド

クローラーがすでに保有している ACL に、追加セキュリティー ACL のコンマで区切られたリストを追加します。

構文

```
    public void addSecurityACLs(String acls)
```

setSecurityACLs メソッド

現行の文書に関するクローラーのセキュリティー ACL を、現行の文書のメタデータの一部として書き込まれるセキュリティー ACL のコンマで区切られたリスト (UTF-8 ストリング) に置き換えます。

構文

```
public void setSecurityACLs(String acls)
```

addMetadataField メソッド

ユーザー定義フィールドをこの文書のメタデータに追加します。メタデータ値は、FieldMetadata クラスのインスタンスを作成し、値を設定することにより定義されます。

構文

```
public void addMetadataField(FieldMetadata md)
```

getMetadataFields メソッド

FieldMetadata オブジェクトが含まれている ArrayList オブジェクトを現行の文書のすべてのメタデータ値と共に戻します。プラグインは、このリストの値を検査および変更することができます。

構文

```
public java.util.ArrayList getMetadataFields()
```

戻り

ArrayList オブジェクト。

setMetadataFields メソッド

現行の文書のすべてのメタデータ値が含まれている FieldMetadata オブジェクトを有する ArrayList オブジェクトを受け入れます。

構文

```
public void setMetadataFields(java.util.ArrayList md)
```

getContent メソッド

文書のダウンロードされたコンテンツが含まれているバイト配列を戻します。これには、解体および転送デコード後の HTTP ヘッダー以降のすべての情報が含まれています。

構文

```
public byte[] getContent()
```

戻り

バイト配列を戻します。

setContent メソッド

コンテンツのバイト配列を設定します。これを変更することにより、パーサーに送信されるものおよびエンタープライズ・サーチのダウンストリーム・コンポーネントが変更されます。

構文

```
public void setContent(byte[] b)
```

getContentType メソッド

起点となるサーバーから戻されたコンテンツ・タイプがある場合、そのコンテンツ・タイプを戻します。

構文

```
public String getContentType()
```

戻り

該当する場合は、コンテンツ・タイプを戻します。

setContentType メソッド

この文書のメタデータを使用して記録されるコンテンツ・タイプを設定します。

構文

```
public void setContentType(String contentType)
```

getEncoding メソッド

クローラーが現行の文書のエンコードであると判別した文字セットのストリング名を戻します (コンテンツ・バイト配列をストリングに変換する際に必要になります)。

構文

```
public String getEncoding()
```

戻り

ストリング名を戻します。

setEncoding メソッド

プラグインは、このメソッドを呼び出して、クローラーがダウンストリーム処理のためにこの文書に関連付けられているメタデータに記録するエンコードを変更することができます。

構文

```
public void setEncoding(String e)
```

getHTTPHeader

HTTP 要求ヘッダー全体をストリングとして戻します。このヘッダーには、復帰改行 (CRLF) が含まれていますが、最後にある 2 番目の (終了) CRLF は戻されません。

構文

```
public String getHTTPHeader()
```

戻り

HTTP 要求ヘッダー全体をストリングとして戻します。

getLanguage メソッド

現行の文書について、起点となるサーバーから戻された言語のコード、またはパーサー (使用している場合) によって検出された言語のコードを戻します。

構文

```
public String getLanguage()
```

戻り

言語コードを戻します。

setLanguage メソッド

現行の文書の言語コードと、言語が推測されるのではなく「既知の」言語であることを示すフラグを設定します。この値は、文書のダウンストリーム処理に影響します。

構文

```
public void setLanguage(String language)
```

doSave メソッド

クローラーが文書を保管するかどうかを指定します。

構文

```
public boolean doSave()
```

戻り

ブール値: true または false。

setSave メソッド

クローラーが文書を保管するかどうかを設定します。

構文

```
public void setSave (boolean yesno)
```

addLink メソッド

現行の文書ですでに検出されているリンクのリストに アウトリンク (クロールする別の URL) を追加できます。

重複は除去されます。相対 URL (例えば、/page/foo/mypage.html など) が追加されます。相対 URL は、ページの BASE URL (定義されている場合) またはページのロケーションを使用して解決されます。絶対 URL も追加できます。リンクはクロール・スペースになければなりません。それ以外の場所にある場合は廃棄されます。

構文

```
public void addLink(String url)
```

getLinks メソッド

クローラーがすでに認識している現行ページの URL を (ストリングのリストとして) 戻します。

構文

```
public java.util.ArrayList getLinks()
```

戻り

URL ストリングのリストを戻します。

setLinks メソッド

ページで検出されたリンクを破棄して、その代わりに、ユーザーが渡すリストを使用するようにクローラーに指示します。 ArrayList URL は、String タイプの ArrayList です。各ストリングには URL 指定が含まれています。

重複は除去されます。相対 URL (例えば、/page/foo/mypage.html など) が追加されます。相対 URL は、ページの BASE URL (定義されている場合) またはページのロケーションを使用して解決されます。絶対 URL も追加できます。リンクはクロール・スペースになければなりません。それ以外の場所にある場合は廃棄されます。

構文

```
public void setLinks(java.util.ArrayList urls)
```

第 10 章 FieldMetadata クラス

メタデータ情報を取得します。

構文

```
public class FieldMetadata {
    public FieldMetadata(String fieldName, String value,
                         boolean searchable, boolean fieldSearchable,
                         boolean parametricSearchable,
                         boolean asMetadata, String resolveConflict,
                         boolean content);
    public String getFieldName();
    public void setFieldName(String fieldName);
    public boolean isSearchable();
    public void setSearchable(boolean searchable);
    public boolean isFieldSearchable();
    public void setFieldSearchable(boolean fieldSearchable);
    public boolean isParametricSearchable();
    public void setParametricSearchable(boolean parametricSearchable);
    public boolean isAsMetadata();
    public void setAsMetadata(boolean asMetadata);
    public String getResolveConflict();
    public void setResolveConflict(String resolveConflict);
    public boolean isContent();
    public void setContent(boolean content);
}
```

fieldMetadata クラスのメソッドは、非 Web クローラー・プラグインのメソッドと同じです。詳しくは、65 ページの『com.ibm.es.crawler.plugin.FieldMetadata クラス』を参照してください。

プラグインは、processDocument メソッドに渡されるメタデータ仕様の値を検査および変更することができます。また、メタデータ・フィールドを削除あるいは追加することもできます。変更は、processDocument メソッドの戻り時に有効になります。

第 11 章 データ・リスナー API リファレンス

DLResponse クラス

クライアント・アプリケーションがデータ・リスナーから受け取るオブジェクトを定義します。

クライアント・アプリケーションがデータ・リスナーに要求を送信すると、データ・リスナーは DLResponse オブジェクトを戻します。オブジェクトには、要求の状況に関する情報が含まれています。また、このクラスは、オブジェクトに関して起こりうる戻りコードも定義します。

タイプ	値
Class	public class DLResponse

DLResponse クラスは、データ・リスナーから以下のタイプの状況を戻す可能性があります。

値	説明
public final static int SUCCESS = 0;	要求された操作が正常に行われました。
public final static int COMMUNICATION_ERROR = 1;	通信エラーが発生しました。
public final static int BAD_REQUEST = 2;	データ・リスナーは要求を理解しませんでした。
public final static int PERMISSION_DENIED = 3;	クライアント ID はコレクションを更新する権限を持っていません。
public final static int FAIL_TO_SAVE = 4;	データ・リスナーは要求の保管に失敗しました。
public final static int UNSUPPORTED_REQUEST_VERSION = 5;	WebSphere II OmniFind Edition のクライアントのバージョンが、サーバーのバージョンと一致していません。
public final static int UNSUPPORTED_RESPONSE_VERSION = 6;	WebSphere II OmniFind Edition のクライアントのバージョンが、サーバーのバージョンと一致していません。
public final static int INVALID_PASSWD = 7;	指定したパスワードは指定したクライアント ID に対して無効です。
public final static int UNKNOWN_COLLECTION = 8;	指定したコレクション ID が無効です。
public final static int FAIL_TO_ADD_TO_CRAWLER = 9;	指定した URL を保管できませんでした。
public final static int FAIL_TO_REMOVE_FROM_COLLECTION = 10;	データ削除要求が失敗しました。
public final static int UNKNOWN_URICODE = 11;	データ・リスナーは要求された操作を実行できませんでした。

getCode メソッド

データ・リスナーから状況コードを戻します。

構文

```
public int getCode()
```

戻り

データ・リスナーから戻りコードに対応する数値を戻します。

getCodeName メソッド

戻りコードに応じてメッセージ・ストリングを戻します。

構文

```
public String getCodeName()
```

戻り

データ・リスナーのいずれかの戻りコードからストリングを戻します。

DLDataPusher クラス

このクラスは、コレクションに対してデータを追加する API およびコレクションから URI を除去するAPI を提供します。また、コレクションの Web クローラーに URL へのアクセスまたは再アクセスを要求する API も提供します。

共通クラス `DLDataPusher`

これは、データ・リスナー API の main クラスです。

これらの API を呼び出す場合は、データ・リスナーのホスト名とポートを指定し、認証用のクライアント ID とパスワードを指定する必要があります。

removeURIs メソッド

指定されたコレクションから URI を除去します。

コレクションから除去する URI を別々に指定するか、またはコレクションから除去する URI パターン を指定することができます。このメソッドを使用して個別に URI を除去すると、 URI は即時に検索で使用できなくなります。しかし、このメソッドを使用して URI パターンを除去すると、パターンに一致する URI が、索引の次回再編成時に除去されます。索引が再編成されるまで、 URI は検索可能です。

構文

```
public static DLResponse removeURIs(String hostname,
                                    int port,
                                    String id,
                                    String pw,
                                    String uris,
                                    String col)
```

パラメーター

hostname
データ・リスナーが稼動しているサーバーのホスト名。

port
データ・リスナーのポート番号。

ID ユーザーまたはエンタープライズ・サーチの管理者が、エンタープライズ・サーチ管理コンソールで特定のコレクションに対して指定したクライアント ID。

pw クライアント ID のパスワード。

uris
コレクションから除去する特定の URI、複数の URI、または URI パターン。各 URI または URI パターンを空白で区切る必要があります。URI パターンを指定する場合は、 URI の最後または途中にワイルドカード文字 (*) を使用します。例えば、 cm://enterprise/finance/* または cm://enterprise/*/sales のように指定します。

col 変更するコレクションの ID。コレクション ID は、エンタープライズ・サーチ管理コンソールで見ることができます。

revisitURLs メソッド

指定されたコレクションの Web クローラーに、URL を追加する、または URL に再アクセスするように指示します。

このメソッドで URL パターンを指定すると、Web クローラーは、パターンに一致する検出済みの URL に即時に再アクセスしようとします。 URL を個別に指定すると、Web クローラーは、指定された URL に即時にアクセスします (または、クローラーが既にそれらの URL を検出済みの場合は再アクセスします)。

revisitURLs メソッドは、Web データ・ソース (URL) に対してのみ使用できます。

構文

```
public static DLResponse revisitURLs(String hostname,
                                      int port,
                                      String id,
                                      String pw,
                                      String urls,
                                      String col)
```

パラメーター

hostname
データ・リスナー・サーバーのホスト名。

port
データ・リスナーのポート番号。

id クライアント ID。

pw クライアントのパスワード。

urls
空白文字で区切られた URL または URL パターン。URL パターンを指定する

場合は、 URL の最後または途中にワイルドカード文字 (*) を使用します。例えば、 http://www.ibm.com/* または http://www.ibm.com/*/software のように指定します。

col コレクションの ID。

pushData メソッド

データ・リスナーにデータをプッシュします。

構文

```
public static DLResponse pushData(String hostname,
                                  int port,
                                  String id,
                                  String pw,
                                  String uri,
                                  String col,
                                  DataSourceMetadata metadata,
                                  byte[] content)
```

パラメーター

hostname
データ・リスナー・サーバーのホスト名。

port
データ・リスナーのポート番号。

id クライアント ID。

pw クライアントのパスワード。

uri データの URI。

col コレクションの ID。

metadata
文書のメタデータを記述するオブジェクト。

content
文書のコンテンツ。

メタデータ・オブジェクトの作成:

データ・プッシュ要求を送信する前に、メタデータ・オブジェクトを作成する必要があります。メタデータには、文書に関する情報が含まれており、作成者、変更日付、作成日付などの情報を組み込むことができます。

事前定義されたフィールドを使用してメタデータ・オブジェクトを作成するには、createDataSourceMetadata API を使用します。データ・ソースに固有のその他のフィールドを追加するには、addMetaField API を使用します。

createDataSourceMetadata メソッド:

メタデータ・オブジェクトを作成します。

構文

```
public static DataSourceMetadata createDataSourceMetadata(String ds,
String cid,
                                                   String dsName,
```

```
                                        int score,
                                        Date dt,
                                        String language,
                                        String securityACLs,
                                        String contentType,
                                        String charSet,
                                        byte[] content)
```

パラメーター

ds この文書の元となるデータ・ソース。

cid データ・リスナー・アプリケーション・クライアント ID。

dsName
データ・ソース名。DB2 や Notes など (ストリング)。

score
このパラメーターは使用されません。

dt この文書の現行性を示す日付オブジェクト。この文書のランキングを左右するために使用されます。

language
文書の言語。「en」、「en_US」、「zh」など。. エンタープライズ・サーチは、文書の言語の判別に失敗した場合に、このパラメーターを使用します。

securityACLs
コンマで区切られたセキュリティー・トークンのストリング。値がヌルの場合、文書にはセキュリティーに関する制限がなく、誰でもアクセスすることができます。例えば、文書にトークン A、B、および C がある場合、セキュリティー・トークン A、B、または C を持っているユーザーのみが文書にアクセスできます。

contentType
この文書の MIME タイプ。

charSet
文書の文字セット。UTF-8 など。

content
文書のコンテンツのバイト配列。

addMetaField メソッド:

メタデータ・オブジェクトにエレメントを追加します。

構文

```
public static void addMetaField(DataSourceMetadata metadata,
                                String fieldName,
                                String fieldValue,
                                boolean searchable,
                                boolean partOfResult,
                                boolean fieldSearchable,
                                boolean parametricSearchable
              boolean isContent)
```

パラメーター

metadata
新規フィールドの追加先となるメタデータ・オブジェクト。

fieldName
フィールドの名前。

fieldValue
フィールドの値。

searchable
このフィールドが検索可能かどうかを示します。

partOfResult
このフィールドが検索結果の一部であるかどうかを示します。

fieldSearchable
このフィールドに対してフィールド検索がサポートされるかどうかを示します。

parametricSearchable
このフィールドに対してパラメトリック検索がサポートされるかどうかを示します。

isContent
このフィールドがコンテンツの一部とみなされるかどうかを示します。

setKnownLanguageFor メソッド:

文書の言語を指定します。

このメソッドを使用して言語を指定しないと、エンタープライズ・サーチは、文書の言語を判別しようとします。

構文

```
static public void setKnownLanguageFor(DataSourceMetadata dsmd,
String knownLanguage);
```

パラメーター

dsmd
メタデータ・オブジェクト。

knownLanguage
言語 ID の名前。「en」、「en_US」、「zh」など。

第 12 章 検索サンプル・アプリケーション

サンプル検索アプリケーション

検索および索引 API には、単純検索アプリケーションや拡張検索アプリケーションの作成方法を示すいくつかのサンプル・アプリケーションが含まれています。

以下のサンプル・アプリケーションでは、さまざまな検索タスクの実行方法を示します。

単純検索

SearchExample クラスは、検索サーバーに検索をサブミットするのに必要な最小必要要件の簡単な例を提供します。

基本的な分類法ブラウズおよびナビゲーション

BrowseExample クラスは、コレクションの分類法ツリーにアクセスし、基本的なナビゲーション・プロパティーをいくつか表示する例を提供します。

すべての検索結果の検索

このサンプル・アプリケーション (コードの断片) は、ソートされていない結果を戻し、照会結果をループする照会の設定方法を示します。

フェデレーテッド・サーチ

FederatedSearchExample クラスは、検索サーバーにフェデレーテッド・サーチをサブミットするのに最低限必要なタスクの簡単な例を提供します。

拡張検索

AdvancedSearchExample クラスは、拡張照会設定および結果処理オプションの使用例を提供します。

サンプル検索アプリケーションのコンパイル

サンプル検索アプリケーションは、Ant スクリプトを実行してコンパイルします。

サンプル検索アプリケーションをコンパイルして実行するには、次のようにします。

1. 以下のディレクトリーに変更します。(これらは、デフォルトのインストール・ディレクトリーです。)
   - AIX、Linux、Solaris: /opt/IBM/es/samples/siapi
   - Windows: C:¥Program Files¥IBM¥es¥samples¥siapi
2. Ant スクリプトを実行します。
3. 以下のコマンドを実行して、アプリケーションを実行します。

   AIX、Linux、 Solaris:

   ```
   java -classpath ES_INSTALL_ROOT/lib/esapi.jar:ES_INSTALL_ROOT/lib/siapi.jar:.
   SearchExample
   ```

   Windows:

```
java -classpath "ES_INSTALL_ROOT¥lib¥esapi.jar;ES_INSTALL_ROOT¥lib/siapi.jar;."
SearchExample
```

コンパイルする他のサンプル検索アプリケーションのいずれかに、SearchExample を置き換えてください。

関連タスク

2 ページの『検索アプリケーションおよびデータ・リスナー・アプリケーションのサンプルのコンパイル』
エンタープライズ・サーチの ESSearchApplication サンプル、データ・リスナー・サンプル、検索および索引 API コードは、IBM Software Developer's Kit 1.4.x を使用してコンパイルする必要があります。IBM Software Developer's Kit 1.5 はサポートされていません。

単純サンプル検索アプリケーションと拡張サンプル検索アプリケーション

SearchExample クラスは、検索サーバーに検索照会をサブミットするのに最低限必要なタスクの簡単な例を提供します。 AdvancedSearchExample クラスは、簡単な例と同じタスクを示していますが、 ResultSet オブジェクトの一部の値だけでなく、すべての値を印刷します。

単純サンプル・アプリケーションでは、以下の方法を示します。

- サービスへのアクセス
- コレクションの指定
- アプリケーションの指定
- 照会の実行
- 戻される結果の処理

拡張サンプル・アプリケーションは、単純サンプルと同じタスクを実行しますが、戻された結果の処理が単純サンプルとは異なります。

単純サンプル・アプリケーション (SearchExample.java) と拡張サンプル・アプリケーション (AdvancedSearchExample.java) は、以下のディレクトリーにあります。

- AIX、Linux、Solaris: /opt/IBM/es/samples/siapi
- Windows: C:¥Program Files¥IBM¥es¥samples¥siapi

ブラウズおよびナビゲーション・サンプル・アプリケーション

BrowseExample クラスは、コレクションの分類ツリーにアクセスし、一部の基本ナビゲーション・プロパティーを表示するサンプル・アプリケーションを提供します。

このサンプルでは、以下の方法を示します。

- ブラウズ・ファクトリーの取得
- ブラウズ・サービスの取得
- ブラウザー参照の取得
- ルート・カテゴリーの検出および表示
- ルートの最初の子カテゴリーの検出

- 子カテゴリーおよびそのルートからのパスの表示

サンプル BrowseExample.java アプリケーションは、以下のディレクトリーにあります。

- AIX、Linux、Solaris: /opt/IBM/es/samples/siapi
- Windows: C:¥Program Files¥IBM¥es¥samples¥siapi

すべての検索結果取得のサンプル

このサンプル・コードは、ソートされていない結果を戻し、照会結果をループする照会の設定方法を示します。照会に対して、ソートされた結果は最大500 個のみ得ることができます。ただし、ソートされていない結果はすべて得ることができます。

以下のサンプル・コードでは、次の方法を示します。

- SearchFactory および Searchable オブジェクトの取得
- 新規照会オブジェクトの作成
- ソートされていない結果を戻す照会の設定
- 検索の実行

SearchFactory および Searchable オブジェクトの取得

92 ページの『単純サンプル検索アプリケーションと拡張サンプル検索アプリケーション』 の例で示すように、SearchFactory オブジェクトおよび Searchable オブジェクトを取得します。

```
SearchFactory factory;
Searchable searchable;

... // obtain a SearchFactory and Searchable object
```

新規照会オブジェクトの作成

```
Query q = factory.createQuery("big apple");
```

ソートされていない結果を戻す照会の設定

```
q.setSortKey(Query.SORT_KEY_NONE);
```

検索の実行

照会をループして実行し、一度に 1 ページ分の結果を取得します。エンタープライズ・サーチで許可される最大結果ページ・サイズは 100 です。

結果ページを受け取る際に、ソートされた照会結果の場合とは異なる方法で、getAvailableNumberOfResults メソッドと getEstimatedNumberOfResults メソッドを解釈する必要があります。

- エンタープライズ・サーチは、ソートされていない結果に対する結果数の見積もりを提供しないため、 getEstimatedNumberOfResults メソッドは常に 0 を戻します。
- getAvailableNumberOfResults メソッドは、これが最終結果ページである場合は 0、まだ結果が続く場合は 1 を戻します。

- getResults メソッドによって戻される配列の長さで、この結果ページ内にいくつの結果があるかがわかります。

```
int fromResult = 0;
int pageSize = 100;
boolean moreResults = true;

// loop over query results, pageSize results at a time
while (moreResults) {

 // set the result range for the next page of results
 q.setRequestedResultRange(fromResult, pageSize);

 // execute the search
 ResultSet resultPage = s.search(q);

 // loop over the results from the ResultSet
 Result[] results = resultPage.getResults();
 for (int i=0;i<results.length;i++) {
... // process result
 }

 // check if there are more available results
 moreResults = (resultPage.getAvailableNumberOfResults() == 1);

 // modify the range for getting the next page of results
 fromResult += pageSize;
}
```

フェデレーテッド・サーチ・サンプル・アプリケーション

FederatedSearchExample クラスは、検索サーバーにフェデレーテッド・サーチをサブミットするのに最低限必要なタスクの簡単な例を提供します。

FederatedSearchExample アプリケーションは、以下の方法を示します。

- フェデレーター ID を使用した RemoteFederator オブジェクトの取得。この ID は、ApplicationInfo オブジェクト ID と同じです。
- 新規照会オブジェクトの作成
- 結果範囲の設定
- RemoteFederator オブジェクトのデフォルト search メソッドの呼び出しによる検索の実行

FederatedSearchExample.java ファイルは、以下のディレクトリーにあります。

- AIX、Linux、Solaris: /opt/IBM/es/samples/siapi
- Windows: C:¥Program Files¥IBM¥es¥samples¥siapi

管理サンプル・アプリケーション

サンプル管理アプリケーション

サンプル・アプリケーションを使用して、アプリケーション ID の作成、コレクションの作成または破棄、索引に対するコレクションの使用可能化または使用不可化、検索に対するコレクションの使用可能化または使用不可化、文書の追加または削除、索引の作成を行うことができます。

サンプル: アプリケーション ID を作成する

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Creates a application ID on the omnifind server.
 * An authenticated application ID is required to
 * administer Omnifind server.
 *
 */
public class CreateApplicationID {
public static void main(String[] args) {
try {
 // instiantiate the admin factory
AdminFactory factory = SiapiAdminImpl.createAdminFactory
("com.ibm.es.siapi.admin.AdminFactoryImpl");
if(factory != null){
String appID = "SIAPI-App";
String appPw = "password";
ApplicationInfo appInfo = factory.createApplicationInfo(appID,appPw);
if(appInfo != null){
System.out.println("Application ID called " +
appInfo.getId() + " was created successfully." );
} else {
System.out.println("Application ID called " +
appInfo.getId() + " was not created!!" );
}

// change the password
appInfo.setPassword("search");
System.out.println("New password was successfully set." );
}
} catch (SiapiException e) {
e.printStackTrace();
System.out.println(e.getMessage());
}

}
}
```

サンプル: コレクションを作成する

```
import java.util.Properties;

import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.es.siapi.common.ApplicationInfoImpl;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Creates a sample collection on the omnifind server
 * and associates the collection
 * to a already created application ID.
 *
 * @see CreateApplicationID
  */
public class CreateCollection {

public static void main(String[] args) {
  try {
```

```
    AdminFactory factory =
SiapiAdminImpl.createAdminFactory
("com.ibm.es.siapi.admin.AdminFactoryImpl");
      if (factory != null) {
        AdminService service = factory.getAdminService(null);
        if (service != null) {
          // get handle to a created application ID
          ApplicationInfo appInfo = factory.createApplicationInfo
          ("SIAPI-App","password");
           Properties config = new Properties();
           // specify custom data directory
          config.setProperty(IAdminConstants.KEY_INDEX_LOCATION,
          "/home/esadmin/siapidata");
           // specify optional n-gram option
          config.setProperty(IAdminConstants.KEY_ENABLE_COLLECTION_NGRAM,
          "true");
           // specify optional security option
          config.setProperty
          (IAdminConstants.KEY_ENABLE_COLLECTION_SECURITY, "true");
           // specify optional max index in the collection
          config.setProperty(IAdminConstants.KEY_MAX_DOCS_IN_INDEX, "10000");

          String colID = "col_123";
          String colLabel = "SIAPI Collection";
          String colLangauge = "en";

         service.createCollection(appInfo,
  colID,
  colLabel,
  0,
  colLangauge,
  config);
       }
    }
  } catch (SiapiException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}
}
```

サンプル: コレクションを破棄する

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
  * Deletes the sample collection created by CreateCollection.
  *
  * @see CreateCollection
  */
public class DestroyCollection {

public static void main(String[] args) {
  try {
    AdminFactory factory =
SiapiAdminImpl.createAdminFactory("com.ibm.es.siapi.admin.AdminFactoryImpl");
if (factory != null) {
  ApplicationInfo appInfo = factory.createApplicationInfo
  ("SIAPI-App","password");
      AdminService service = factory.getAdminService(null);
      if (service != null) {
        // collection ID is required to destroy the collection
```

```
        String colID = "col_123";
        service.destroyCollection(appInfo, colID);
      }
    }
  } catch (SiapiException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}
}
```

サンプル: コレクションを索引で使用可能にする

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Enables a collection for indexing.
 * The parser driver and the data listener sessions
 * are checked and started.
 *
*/
public class EnableCollectionForIndexing {
public static void listIndexableCollectionIDs
(AdminService service, ApplicationInfo appInfo)
throws SiapiException{

  String [] indexables = service.getIndexableCollectionIDs(appInfo);
  if (indexables != null && indexables.length > 0) {
    System.out.println("Following are indexable collection ids:");
    for (int i=0; i<indexables.length; ++i) {
      System.out.println(indexables[i]);
    }
  }
}

  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
     (IAdminConstants.ADMIN_FACTORY_IMPL);
    if (factory != null) {
      ApplicationInfo appInfo = factory.createApplicationInfo
      ("SIAPI-App","password");
      AdminService service = factory.getAdminService(null);
      if (service != null) {
        String colID = "col_123";
        listIndexableCollectionIDs(service, appInfo);
        boolean enabled = service.isEnabledForIndexing(appInfo, colID);
        if (!enabled) {
          // enable for indexing
          System.out.println("Enabling collection for indexing");
          service.enableCollectionForIndexing(appInfo, colID, null);
        }
        //check if enabled for indexing
        enabled = service.isEnabledForIndexing(appInfo, colID);
        if (enabled) {
          System.out.println("Collection is enabled for indexing");
        } else {
          System.out.println("Collection is not enabled for indexing");
        }
    listIndexableCollectionIDs(service, appInfo);
    }
   }
```

```
  } catch (SiapiException e) {
    e.printStackTrace();
  }
 }
}
```

サンプル: コレクションを索引で使用不可にする

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Disables a collection for indexing.
 *
  */
public class DisableCollectionForindexing {

  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo = factory.createApplicationInfo
        ("SIAPI-App","password");
        AdminService service = factory.getAdminService(null);
        if (service != null) {

          String colID = "col_123";

          boolean enabled = service.isEnabledForIndexing(appInfo, colID);
          if (enabled) {
            // enable for indexing
            System.out.println("Disabling collection for indexing");
            service.disableCollectionForIndexing(appInfo, colID, null);
          }
          //check if collection is disabled for indexing
          enabled = service.isEnabledForIndexing(appInfo, colID);
          if (enabled) {
            System.out.println("Collection is enabled for indexing");
          } else {
            System.out.println("Collection is not enabled for indexing");
          }
         }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
    }
  }
}
```

サンプル: コレクションを検索で使用可能にする

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 * Enables a collection for searching.
 * The search runtimes for the specified collection are started
 *
```

```
 */
public class EnableCollectionForSearch {

  public static void listSearchableCollectionIDs
(AdminService service, ApplicationInfo appInfo) throws SiapiException{
    String [] searchables = service.getSearchableCollectionIDs(appInfo);
    if (searchables != null && searchables.length > 0) {
      System.out.println("Following are searchable collection ids:");
      for (int i=0; i<searchables.length; ++i) {
        System.out.println(searchables[i]);
      }
    }
  }


  public static void main(String[] args) {
    try {
      AdminFactory factory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo = factory.createApplicationInfo
        ("SIAPI-App","password");
        AdminService service = factory.getAdminService(null);
        if (service != null) {
          String colID = "col_123";
          listSearchableCollectionIDs(service, appInfo);
          boolean enabled = service.isEnabledForSearch(appInfo, colID, null);
          if (!enabled) {
            // enable for searching
            System.out.println("Enabling collection for searching");
            service.enableCollectionForSearch(appInfo, colID, null);
          }
          //check if enabled for searching
          enabled = service.isEnabledForSearch(appInfo, colID, null);
          if (enabled) {
            System.out.println("Collection is enabled for searching");
          } else {
            System.out.println("Collection is not enabled for searching");
          }

          listSearchableCollectionIDs(service, appInfo);
        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
    }
  }
}
```

サンプル: コレクションを検索で使用不可にする

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;

/**
 *
 * Disables a collection for search.
 *  The search runtimes for the specified collection
 *  are stopped.
 */
public class DisableCollectionForSearch {

  public static void main(String[] args) {
```

```
        try {
            AdminFactory factory = SiapiAdminImpl.createAdminFactory
            (IAdminConstants.ADMIN_FACTORY_IMPL);
            if (factory != null) {
              ApplicationInfo appInfo = factory.createApplicationInfo
              ("SIAPI-App","password");
              AdminService service = factory.getAdminService(null);
              if (service != null) {

                String colID = "col_123";

                boolean enabled = service.isEnabledForSearch
                (appInfo, colID, null);
                if (enabled) {
                  // disable for searching
                  System.out.println("Disabling collection for indexing");
                  service.disableCollectionForSearch(appInfo, colID, null);
                }
                //check if enabled for searching
                enabled = service.isEnabledForSearch(appInfo, colID, null);
                if (enabled) {
                  System.out.println("Collection is enabled for searching");
                } else {
                  System.out.println("Collection is not enabled for searching");
                }
              }
            }
        } catch (SiapiException e) {
          e.printStackTrace();
        }
      }
    }
```

サンプル: 文書を索引に追加する

```
import java.util.Date;

import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.admin.AdminFactory;
import com.ibm.siapi.admin.AdminService;
import com.ibm.siapi.admin.SiapiAdminImpl;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Document;
import com.ibm.siapi.index.Field;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Add documents to a index.
 *
 * Type comment
 */
public class AddDocumentsAndFields {

  public static void main(String[] args) {
    try {
      IndexFactory iFactory = SiapiIndexImpl.createIndexFactory
(IAdminConstants.INDEX_FACTORY_IMPL);
      AdminFactory aFactory = SiapiAdminImpl.createAdminFactory
(IAdminConstants.ADMIN_FACTORY_IMPL);

      if (iFactory != null) {
        ApplicationInfo appInfo = iFactory.createApplicationInfo
```

```
("SIAPI-App","password");
if (aFactory != null) {
// enable collection for indexing
AdminService aService = aFactory.getAdminService(null);
if (aService != null) {
  aService.enableCollectionForIndexing(appInfo, "col_123", null);
}
if (!aService.isEnabledForIndexing(appInfo, "col_123")) {
  System.out.println("Sorry, can't add document to the index.
                      The parser driver can not be started");
  System.exit(0);
}
}

IndexService service = iFactory.getIndexService(null);
if (service != null) {
  Index index = service.getIndex(appInfo, "col_123");
  if (index == null) {
    System.out.println("Index instance could not be
                        instantiated.. return now!!");
    System.exit(0);
  }

  for (int i=0;i<3; ++i) {
    Document doc = iFactory.createDocument( "docid" + i,
                                    "My document id="+i,
                                    "custom" + i,
                                    "custom" + i);


    doc.setDate(new Date());
    doc.setLanguage("en");
    doc.setRawContentFormat("text/plain");

    // set security tokens
    /*
    String[] nativeACL = new String [1];

    if(i == 0){
      nativeACL[0] = "<NativeACL><Groups><Group>Admins
      </Group><Group>Staff</Group></Groups></NativeACL";
    } else if (i== 1){
      nativeACL[0] = "Admin,staff";
    } else {
      nativeACL[0] = "";
    }
    doc.setACL(nativeACL);
    */

    // create fields
    Field field1 = iFactory.createField("Title", "Mr. Srinivas");
    field1.setContentSearchable(true);
    field1.setFieldSearchable(true);
    field1.setParametric(false);
    field1.setReturnable(true);
    field1.setConflictResolutionPolicy
    (Field.CONFLICT_FIELD_OVERRIDES);


    Field field2 = iFactory.createField("Address",
                   "200 Gibralter Drive, USA");
    field2.setContentSearchable(true);
    field2.setFieldSearchable(true);
    field2.setParametric(false);
    field2.setReturnable(true);
    field2.setConflictResolutionPolicy
    (Field.CONFLICT_CONTENT_OVERRIDES);
```

```
        Field field3 = iFactory.createField("Street number", 200);
        field3.setContentSearchable(true);
        field3.setFieldSearchable(true);
        field3.setParametric(true);
        field3.setReturnable(true);
        field3.setConflictResolutionPolicy(Field.CONFLICT_COEXIST);

        doc.addField(field1);
        doc.addField(field2);
        doc.addField(field3);

        // add document to the index
        index.addDocument(doc);
      }
      // get statistics of documents in the store
      IndexStats stats = index.getStatistics();
      if (stats != null && stats.getNumPendingUpdates() > 0) {
        // build main index
        index.reorganize();
      }
     }

    }
```

サンプル: 文書を索引から除去する

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Remove specified documents from the index.
 *
 * Type comment
 */
public class RemoveDocument {

  public static void main(String[] args) {
    try {
      IndexFactory factory =
SiapiIndexImpl.createIndexFactory(IAdminConstants.INDEX_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo =
        factory.createApplicationInfo("SIAPI-App","password");
        IndexService service = factory.getIndexService(null);
        if (service != null) {
          Index index = service.getIndex(appInfo, "col_123");
          if (index == null) {
            System.out.println("Index instance could not be
                                instantiated.. return now!!");
            System.exit(0);
          }

          index.removeDocument("docid0");

        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
```

```
      System.out.println(e.getLocalizedMessage());
    }
  }
}
```

サンプル: 索引を作成する

```
import com.ibm.es.siapi.client.IAdminConstants;
import com.ibm.siapi.SiapiException;
import com.ibm.siapi.common.ApplicationInfo;
import com.ibm.siapi.index.Index;
import com.ibm.siapi.index.IndexFactory;
import com.ibm.siapi.index.IndexService;
import com.ibm.siapi.index.IndexStats;
import com.ibm.siapi.index.SiapiIndexImpl;

/**
 * Build index with documents.
 * The
 * Type comment
 */
public class BuildIndex {

  public static void main(String[] args) {
    try {
      IndexFactory factory =
SiapiIndexImpl.createIndexFactory(IAdminConstants.INDEX_FACTORY_IMPL);
      if (factory != null) {
        ApplicationInfo appInfo =
  factory.createApplicationInfo("SIAPI-App","password");

        IndexService service = factory.getIndexService(null);
        if (service != null) {
          Index index = service.getIndex(appInfo, "col_123");
          if (index == null) {
            System.out.println("Index instance could not be
                                instantiated.. return now!!");
            System.exit(0);
          }

          // set fragmentation count to 2
          // after every 2 delta build, the index
          // will be reorganized
          index.setProperty
          (IAdminConstants.BUILD_FRAGMENTATION_COUNT, "2");

          // get statistics of documents in the store
          IndexStats stats = index.getStatistics();
          if (stats != null && stats.getNumPendingUpdates() > 0) {
            // build index
            index.build();
          }
        }
      }
    } catch (SiapiException e) {
      e.printStackTrace();
      System.out.println(e.getLocalizedMessage());
    }
  }
}
```

サンプル管理アプリケーションのコンパイル

サンプル管理アプリケーションは、Ant スクリプトを実行してコンパイルします。

管理 API 用のクライアント・ツールキットがインストールされていることを確認してください。

サンプル管理アプリケーションをコンパイルして実行するには、次のようにします。

1. 以下のディレクトリーに変更します。(これらは、デフォルトのインストール・ディレクトリーです。)

   AIX、Linux、Solaris: /home/user/siapi/samples

   Windows: C:¥Program Files¥IBM¥es¥siapi¥samples
2. Ant スクリプトを実行します。
3. 以下のコマンドを実行して (1 行で指定する)、アプリケーションを実行します。

   AIX、Linux、Solaris:

   ```
   runSample.sh "absolute path of es.client.cfg"
   SampleName
   ```

   Windows:

   ```
   runSample "absolute path of es.client.cfg"
   SampleName
   ```

Sample_Name をコンパイルする他のサンプル管理アプリケーションに置き換えてください。

関連タスク

2 ページの『検索アプリケーションおよびデータ・リスナー・アプリケーションのサンプルのコンパイル』

エンタープライズ・サーチの ESSearchApplication サンプル、データ・リスナー・サンプル、検索および索引 API コードは、IBM Software Developer's Kit 1.4.x を使用してコンパイルする必要があります。IBM Software Developer's Kit 1.5 はサポートされていません。

第 13 章 非 Web クローラーのサンプル・プラグイン・アプリケーション

サンプル・プラグイン・アプリケーションは、クロールされた文書のコンテンツおよびメタデータの変更方法を示します。

```
package sample;

import java.util.List;

import com.ibm.es.crawler.plugin.CrawledData;
import com.ibm.es.crawler.plugin.CrawlerPlugin;
import com.ibm.es.crawler.plugin.CrawlerPluginException;
import com.ibm.es.crawler.plugin.FieldMetadata;

public class CrawlerPluginImpl implements CrawlerPlugin {


   /* init() is called when the crawler session starts.*/
   public void init() throws CrawlerPluginException {
      // Your code goes here.
   }


   /* term() is called when the crawler session stops.*/
   public void term() throws CrawlerPluginException {
      // Your code goes here.
   }


   /* isMetadataUsed() specifies whether updateDocument() uses
      crawled metadata or not.
    * Return true to obtain crawled metadata at updateDocument().
    * Return false to disable crawled metadata transfer from the crawler
      to the crawler plug-in. */
   public boolean isMetadataUsed() {
      // Your code goes here.

      return true;
   }


   /* updateDocument() is called for each crawled document.
    *
    *  crawledData argument is composed of document URI, security
       tokens and
    * list of crawled metadata (if isMetadataUsed() returns true.)
    *  Return CrawledData to overwrite security tokens and
    *  metadata by the crawler.
    *  Return null not to index current document. */
   public CrawledData updateDocument(CrawledData crawledData)
   throws CrawlerPluginException {
      // Your code goes here.

      // Extract document URI.
      String uri = crawledData.getURI();

      // Extract security tokens string.
      String securityTokens = crawledData.getSecurityTokens();
      // Set new security tokens.
      crawledData.setSecurityTokes(securityTokens+",new_security_tokens");
```

```
                // Extract metadata list. getMetadataList() returns null
                   if isMetadataUsed() returns false.
                List metadataList = crawledData.getMetadataList();
                // Append additional metadata.
                FieldMetadata newFieldMetadata1 = new FieldMetadata(
                      "new_search_field_name1",
                      // Search field name
                      "new_field_value1",
                     // Field value
                      true,
                     // True if this field is searchable.
                      true,
                     // True if this field is field-searchable.
                      false,
                     // True if this field is parametric-searchable.
                      true,
                     // True if this field can be extracted from search application.
                      FieldMetadata.METADATAPREFERRED,
                     // Metadata source priority between crawled metadata and content.
                     //   This argument should be METADATAPREFERRED, CONTENTPREFERRED,
                     //   or COEXIST.
                      true);
                    // True if this field can be shown as summary of search result.
                FieldMetadata newFieldMetadata2 = new FieldMetadata(
                      "new_search_field_name2", "new_field_value2", true, false,
                       false, false, FieldMetadata.CONTENTPREFERRED, false);
                FieldMetadata newFieldMetadata3 = new FieldMetadata(
                      "new_search_field_name3", "new_field_value3", false, true,
                       false, true, FieldMetadata.COEXIST, true);
                metadataList.add(newFieldMetadata1);
                metadataList.add(newFieldMetadata2);
                metadataList.add(newFieldMetadata3);

                // Return modified data.
                return crawledData;
            }
        }
```

第 14 章 データ・リスナー・サンプル・アプリケーション

データ・リスナー・クライアント・サンプル・アプリケーション

WebSphere II OmniFind Edition は、データ・リスナー API を使用して、エンタープライズ・サーチ・コレクションにデータを追加する方法またはエンタープライズ・サーチ・コレクションからデータを除去する方法を示すサンプル・クライアント・アプリケーションを提供します。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

以下のサンプル・クライアント・アプリケーションは、次のタスクを行う方法を示します。

DLRemoveURIs.java
コレクションから URI を除去します。

DLPushData.java
コレクションに URI およびコンテンツを追加します。

DLRevisitURLs.java
Web クローラーに URL への再アクセスを指示します。

DLSampleClient.java
すべてのタスクを URI の除去、URI の追加、および URL の再アクセスを行う 1 つのアプリケーションに結合します。

データ・リスナー・クライアント・サンプル・アプリケーション: コレクションからの URI の除去

DLRemoveURIs クラスには、コレクションの URI または URI パターンを除去するクライアント・アプリケーション API の例が含まれています。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

DLRemoveURIs クラスは、main と removeURIsExample の 2 つの静的メソッドを定義します。

main メソッドは、まず、ホスト名、ポート、クライアント ID、パスワード、およびコレクション ID を準備します。次に、 main メソッドは、これらのパラメーターを指定して removeURIsExample メソッドを呼び出します。

removeURIsExample メソッドは、パラメーターとしてホスト名、ポート、クライアント ID、パスワード、およびコレクション ID を受け取って、 URI または URI パターン・ストリングを準備します。

URI または URI パターンを準備したら、 removeURIsExample は、DLDataPusher クラスの removeURIs メソッドを呼び出します。その後、removeURIsExample メソッドは、データ・リスナーからの応答を検査して印刷します。

```
public class DLRemoveURIs {

    /**
     *  To remove URIs or URI patterns from a collection, you
     *  need to specify the host name and the port of the
     *  data listener server. You also need to provide
     *  the client ID and password for authentication.
     *  You need to specify which collection the data is
     *  applied to by providing the collection ID.
     *
     *  This function shows how to use the data listener
     *  APIs to remove URIs from a collection. In enterprise
     *  search, you can remove specific URIs and URIs that
     *  match patterns. If you remove specific URIs, those
     *  URIs are removed from the index. If you remove URI patterns,
     *  those URIs that match those patterns are not removed until
     *  the next index reorganization, and those URIs are still
     *  searchable. After the URIs are removed from a collection, those
     *  URIs will not be searchable until the they are re-crawled by
     *  the crawler or added through the data listener later.
     */
    static void removeURIsExample(String hostname, int port,
    String clientID, String passwd, String collectionID) {

        // In this example, you will remove some URIs from
        // one collection.
        // URIs can be either specific URIs or URIs that match patterns.
        StringBuffer sb = new StringBuffer();
        sb.append("uri1");
        sb.append("¥n");
        sb.append("uri*pattern1");
        sb.append("¥n");

        sb.append("uri2");
        sb.append("¥n");
        sb.append("uri*pattern2");

        String uris = sb.toString();

        // Now, you call the data listener API to remove URIs
        // from the collection.
        // Note that you can specify multiple URIs and URI patterns
        // in a single call. The URIs or URI patterns are seperated
        // by a newline character. In your application, you need
        // to replace the URIs and URI patterns from the sample
        // application with the real URIs that make sense for a
        // real collection.
        DLResponse dlRes = DLDataPusher.removeURIs(hostname, port,
        clientID, passwd, uris, collectionID);

        // Check the result.
        // The DLResponse object contains a result code and
        // a message that indicates the result of the operation.
        // See the documentation for the DLResponse class for the
        // code values and their meanings.
        if (dlRes != null) System.out.println(dlRes.toString());
    }

    public static void main (String [] argv) {
        if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
            System.out.println("usage: java DLRemoveURIs [hostname [port]]");
            return;
        }

        // First, you get the host name and the port number
        // for the data listener.
        // You need to ensure that the data listener is running on
```

```
        // the host server and listening to the port.
        // Otherwise, you might get a connection refused exception.
        String hostname = "localhost";
        if (argv.length > 0) hostname = argv[0];
        int port = 6668;
        if (argv.length > 1) {
            try {
                port = Integer.parseInt(argv[1]);
            }
            catch (Exception e) {
               port = 6668;
            }
        }

        // Assume that client_id_1 is a valid client id with password_1.
        // If not, you will get INVALID_PASSWD as the result code

        // Assume that client_id_1 has the authorization to update
        // collection with collection ID collection_id_1.
        // If not, you will get PERMISSION_DENIED as the result code.
        //
        // Assume that the collection  with collection ID collection_id_1
        // is a valid collection. Otherwise, you will get
        // UNKNOWN_COLLECTION as the result code.
        //
        // You need to contact the enterprise search administrator
        // to find out what client ID, password, and collection ID
        // are valid and use those valid values here.
        //
        removeURIsExample(hostname, port, "client_id_1", "password_1",
        "collection_id_1");
    }
}
```

データ・リスナー・クライアント・サンプル・アプリケーション: コレクションへの URI およびコンテンツの追加

DLPushData クラスには、URI および各 URI のコンテンツをコレクションに追加またはプッシュするクライアント・アプリケーション API の例が含まれています。現在、データ・リスナー API の使用は推奨されていません。検索および索引 API を使用して、コレクションを管理してください。

DLPushData クラスは、main と dataPushExample の 2 つの静的メソッドを定義します。

main メソッドは、まず、ホスト名、ポート、クライアント ID、パスワード、およびコレクション ID を準備します。次に、 main メソッドは、これらのパラメーターを指定して dataPushExample メソッドを呼び出します。

URI およびそのコンテンツをコレクションに追加するには、文書 URI、コンテンツ、およびメタデータを準備します。URI およびコンテンツを準備するには、シンプル・テキスト・ストリングを追加します。文書のメタデータを準備するには、DLDataPusher クラスから DataSourceMetadata オブジェクトを作成します。その後、DataSourceMetadata オブジェクトにフィールドを追加します。

pushData メソッドを呼び出して、データ・リスナーからの応答を確認します。

```
public class DLPushData {

    /**
```

```
     *  To push data to a data listener, you need to specify the host name
     *  and the port of the data listener server.
     *  You also need to provide the client ID and password for authentication.
     *  You need to specify which collection the data is applied to by providing the
     *  collection ID.
     *
     *  This function shows how to use the data listener APIs to push a document
     *  to an enterprise search collection for indexing. A document consists of
     *  three parts: the URI, the metadata and the content.
     *  The content is the raw data of the document.
     *  The metadata contains the attributes values of
     *  the document.
     */
    static void dataPushExample(String hostname, int port, String clientID,
    String passwd, String collectionID) {

        // Prepare the data.
        // Suppose this is a very simple document with "almost empty" content
        String content = "almost empty";
        // Suppose the URI for the document is "myURI"
        String uri = "myURI";

        // Use data listener client APIs to prepare metadata.
        //
        // First, create a DataSourceMetadata object.
        //
        // The first argument indicates the data source type
        // of this document. The second argument specifies the client ID.
        // The third argument is a string that specifies the data source name.
        // The fourth argument is a number that indicates the quality of this
        // this document. This number is not currently used.
        // The fifth argument is a date that indicates
        // the currency (how up to date it is) of this document. It can be
        // used to influence the ranking of this document in the search results.
        // The sixth argument indicates language of the document.
        // Normally, enterprise search will detect the language ID of a
        // given document.
        // However, if the detection fails, the enterprise search system
        // will assume the document is in this specified language.
        // The seventh argument specifies security tokens.
        // If it is null, then this document is assumed to be available
        // to anyone. Otherwise, the document is accessible only by
        // a user with a security token that is specified here.
        // The eigthth argument specifies the MIME type of the document.
        // The ninth argument specifies the character set of the content.
        // Finally, the tenth argument is the content.
        DataSourceMetadata md =
    DLDataPusher.createDataSourceMetadata("CustomerDataSource",
     clientID,
                          "CustomerDataSourceName",
                          90,
                          new Date(),
                          "en",
                          "securityToken",
                          "text/plain",
                          "iso-8859-1",
                          content.getBytes());

        // Second, you add more fields to the metadata.
        // Each field is a name/value pair.
        // The fourth argument specifies whether the field value is searchable.
        // The fifth argument specifies whether the field value will
        // be part of the search result.
        // The sixth argument specifies whether to support field search.
        // The seventh argument specifies whether the field is
        // parametric searchable.
        // The eight argument specifies whether the field is part of the content.
```

```
            DLDataPusher.addMetaField(md,
                                      "fieldName1", "fieldValue1",
                                      true, false, true, false, true);
            DLDataPusher.addMetaField(md,
                                      "fieldName2", "fieldValue2",
                                      true, false, true, false, true);
            System.out.println("metadata:¥n" +  md.generateXML().toString());

            // Call the pushData method.
            DLResponse dlRes = DLDataPusher.pushData(hostname, port, clientID,
            passwd, uri, collectionID, md, content.getBytes());

            // Check the result from the data listener.
            if (dlRes != null) System.out.println(dlRes.toString());
        }

        public static void main (String [] argv) {
            if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
                System.out.println("usage: java DLPushData [hostname [port]]");
                return;
            }

            // First, you get the host name and the port number for the data listener.
            // You need to ensure that the data listener is running on the host server
            // and listening to the port.
            // Otherwise, you might get a connection refused exception.
            String hostname = "localhost";
            if (argv.length > 0) hostname = argv[0];
            int port = 6668;
            if (argv.length > 1) {
                try {
                    port = Integer.parseInt(argv[1]);
                }
                catch (Exception e) {
                   port = 6668;
                }
            }

            // Assume that client_id_1 is a valid client ID with password_1.
            // Assume that client_id_1 has the authorization to update the
            // collection with collection ID collection_id_1.
            //
            // You need to contact the enterprise search administrator to find out what
            // client ID, password, and collection ID are valid and
            // use those valid values here.
            //
            dataPushExample(hostname, port, "client_id_1", "password_1",
                            "collection_id_1");
        }
    }
```

データ・リスナー・クライアント・サンプル・アプリケーション: URL への再アクセス

DLRevisitURLs クラスには、Web クローラーに特定の URL または URL パターンへの再アクセスを指示するクライアント・アプリケーション API の例が含まれています。

DLRevisitURLs クラスは、main と revisitURLsExample の 2 つの静的メソッドを定義します。

main メソッドは、まず、ホスト名、ポート、クライアント ID、パスワード、およびコレクション ID を準備します。次に、 main メソッドは、これらのパラメーターを指定して revisitURLsExample メソッドを呼び出します。

revisitURLsExample メソッドは、パラメーターとしてホスト名、ポート、クライアント ID、パスワード、およびコレクション ID を受け取って、 URL または URL パターン・ストリングを準備します。

URL または URL パターンを準備したら、 revisitURLsExample メソッドは、DLDataPusher クラスの revisitURLs メソッドを呼び出します。その後、revisitURLsExample メソッドは、データ・リスナーからの応答を検査して印刷します。

```
public class DLRevisitURLs {

    /**
     *  To revisit URLs, you need to specify the host name
     *  and the port of the data listener server.
     *  You also need to provide the client ID and password for
     *  authentication.
     *  You need to specify which collection the data is applied
     *  to by providing the collection ID.
     *
     *  This function shows you how to use the data listener APIs
     *  to instruct the Web crawler of a collection to revisit
     *  URLs or URLs that match patterns.
     */
    static void revisitURLsExample(String hostname, int port,
    String clientID, String passwd, String collectionID) {

        // You will revisit some URLs from one collection.
        // URLs can be either individual URLs or URLs that
        // match patterns.
        StringBuffer sb = new StringBuffer();
        sb.append("uri1");
        sb.append("¥n");
        sb.append("uri*pattern1");
        sb.append("¥n");

        sb.append("uri2");
        sb.append("¥n");
        sb.append("uri*pattern2");

        String uris = sb.toString();

        // Now, you call the data listener API to revisit URLs.
        // Note that you can specify multiple URLs and URL
        // patterns in a single call. The URLs (URL patterns)
        // are seperated by a newline character.
        DLResponse dlRes = DLDataPusher.revisitURLs(hostname, port,
        clientID, passwd, uris, collectionID);

        // Check the result from the data listener.
        // The DLResponse object contains a result code and a
        // message that indicates the result of the operation.
        // See the documentation for the DLResponse class for
        // the code values and their meanings.
        if (dlRes != null) System.out.println(dlRes.toString());
    }

    public static void main (String [] argv) {
        if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
            System.out.println("usage: java DLRevisitURLs [hostname [port]]");
            return;
```

```
        }

        // First, you get the host name and the port number for
        // the data listener. You need to ensure that the data
        // listener is running on the host server and listening
        // to the port.
        // Otherwise, you might get a connection refused exception.
        String hostname = "localhost";
        if (argv.length > 0) hostname = argv[0];
        int port = 6668;
        if (argv.length > 1) {
            try {
                port = Integer.parseInt(argv[1]);
            }
            catch (Exception e) {
               port = 6668;
            }
        }

        // Assume that client_id_1 is a valid client ID with password_1.
        // If not, you will get INVALID_PASSWD as the result code.

        // Assume that client_id_1 has the authorization to update
        // collection with collection ID collection_id_1.
        // If not, you will get PERMISSION_DENIED as the result code
        //
        // Assume that the collection  with collection id collection_id_1
        // is a valid collection. Otherwise, you will get
        // UNKNOWN_COLLECTION as the result code.
        //
        // You need to contact the enterprise search administrator
        // to find out what  client ID, password, and collection ID
        // are valid and use those valid values here.
        //
        revisitURLsExample(hostname, port, "client_id_1", "password_1",
        "collection_id_1");
    }
}
```

データ・リスナー・クライアント・サンプル・アプリケーション: コレクションに対するデータの追加、除去、および再アクセス

DLSampleClient クラスは、1 つ以上のコレクションに対してデータを追加、除去、および再アクセスする方法を示すサンプル・コードを提供します。

DLSampleClient は、URI の追加、URL の除去、および URL の再アクセス用のサンプル・クライアント・アプリケーションを結合しています。

```
import java.io.*;
import java.net.*;
import java.util.Date;

import com.ibm.es.datalistener.client.*;
import com.ibm.es.datalistener.common.*;
import com.ibm.es.util.DataSourceMetadata;

/**
 * This class is sample code that shows you how to use the data listener APIs
 * to update collections.
 */
public class DLSampleClient {

    /**
     *  This example shows how to use data listener APIs to revisit URLs of a
     *  collection. You need to specify the host name
```

```
 *  and the port of the data listener server.
 *  You also need to provide the client ID and password for authentication.
 *  You need to specify which collection this operation is applied to
 *  by providing the collection ID.
 */
static void dataPushExample_1(String hostname, int port, String clientID,
String passwd, String collectionID) {

    // You will visit (add) and revisit several URLs of one collection.
    // You can revisit URLs or URLs that match patterns. URLs and
    // URL patterns are separated by a newline character. A URL pattern
    // is a string with a wildcard character (*).
    //
    StringBuffer sb = new StringBuffer();
    sb.append("url1");
    sb.append("¥n");
    sb.append("url*pattern1");
    sb.append("¥n");

    sb.append("url2");
    sb.append("¥n");
    sb.append("url*pattern2");

    String urls = sb.toString();

    // Now, you call the revisitURLs method of DLDataPusher class.
    DLResponse dlRes = DLDataPusher.revisitURLs(hostname, port, clientID,
    passwd, urls, collectionID);

    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());
}

/**
 *  This example shows how to use data listener APIs to remove URIs of a
 *  collection. You need to specify the host name
 *  and the port of the data listener server.
 *  You also need to provide the client ID and password for authentication.
 *  You need to specify which collection this operation is applied to
 *  by providing the collection ID.
 */
static void dataPushExample_2(String hostname, int port, String clientID,
String passwd, String collectionID) {

    // You now remove some URIs from a collection.
    // URIs are separated by a  newline characters.
    // Those URIs will be removed immediately so that they do
    // not appear in the search result.
    StringBuffer sb = new StringBuffer();
    sb.append("url1");
    sb.append("¥n");
    sb.append("url2");
    String urls = sb.toString();

    // Now, you call the removeURIs method of DLDataPusher class.
    DLResponse dlRes = DLDataPusher.removeURIs(hostname, port, clientID,
    passwd, urls, collectionID);
    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());



    // You can also remove URIs that match patterns.
    // URI patterns are separated by newline characters.
    // Those URIs that match these patterns will be removed
    // during the next index reorganization. Note that
    // these results might still appear in the search result until
```

```
        // the next index reorganization.
        sb = new StringBuffer();
        sb.append("url*pattern1");
        sb.append("¥n");
        sb.append("url*pattern2");
        String url_patterns = sb.toString();

        // Now, you call the removeURIs method of DLDataPusher class.
        dlRes = DLDataPusher.removeURIs(hostname, port, clientID, passwd,
        url_patterns, collectionID);
        // Check the response from the data listener.
        if (dlRes != null) System.out.println(dlRes.toString());

        // You can even remove both individual URLs and URLs that match
        // patterns in the same request.
        sb = new StringBuffer();
        sb.append("url3");
        sb.append("¥n");
        sb.append("url*pattern4");
        sb.append("¥n");
        sb.append("url5");

        // Now, you call the removeURIs method of DLDataPusher class.
        dlRes = DLDataPusher.removeURIs(hostname, port, clientID, passwd,
        sb.toString(), collectionID);
        // Check the response from the data listener.
        if (dlRes != null) System.out.println(dlRes.toString());

    }

    /**
     *  This example shows how to use data listener APIs to push
     *  documents to a collection. You need to specify the host name
     *  and the port of the the data listener server.
     *  You also need to provide the client ID and password for authentication
     *  You need to specify which collection the data is applied to
     *  by providing the collection ID.
     */
    static void dataPushExample_3(String hostname, int port, String clientID,
    String passwd, String collectionID) {

        // Prepare the content.
        String content = "Almost empty";

        // Prepare the URI.
        String uri = "myURI2";

        // Prepare the metadata.
        //
        // First, create a DataSourceMetadata object
        DataSourceMetadata md =
DLDataPusher.createDataSourceMetadata("CustomerDataSource",
 clientID,
                        "CustomerDataSourceName",
                        90,
                        new Date(),
                        "en",
                        "securityToken",
                        "text/plain",
                        "iso-8859-1",
                        content.getBytes());

        // Second, add more fields to the metadata.
        // Each field is a name/value pair.
        // The fourth argument specifies whether the field value is
        // searchable.
        // The fifth argument specifies whether the field value will be part
```

```
    // of the search result.
    // The sixth argument specifies whether to support field search.
    // The seventh argument specifies whether the field is parametric
    // searchable.
    // The eight argument specifies whether the field is part of the
    // content.
    DLDataPusher.addMetaField(md,
                              "fieldName1", "fieldValue1",
                              true, false, true, false, false);
    DLDataPusher.addMetaField(md,
                              "fieldName2", "fieldValue2",
                              true, false, true, false, false);
    System.out.println("metadata:¥n" +  md.generateXML().toString());

    // Call the pushData method
    DLResponse dlRes = DLDataPusher.pushData(hostname, port, clientID,
    passwd, uri, collectionID, md, content.getBytes());

    // Check the response from the data listener.
    if (dlRes != null) System.out.println(dlRes.toString());

    // Push the same result again. This one will overwrite the previous one.
    dlRes =  DLDataPusher.pushData(hostname, port, clientID, passwd, uri,
    collectionID, md, content.getBytes());
    if (dlRes != null) System.out.println(dlRes.toString());
}

public static void main (String [] argv) {
    if (argv.length > 0 && argv[0].equalsIgnoreCase("-h")) {
        System.out.println("usage: java DLSampleClient [hostname [port]]");
        return;
    }

    // First, you obtain the host name and the port number for the data
    // listener.
    // You need to ensure that the data listener is running on the
    // host server and listening to the port.
    // Otherwise, you might get a connection refused exception.
    String hostname = "localhost";
    if (argv.length > 0) hostname = argv[0];
    int port = 6668;
    if (argv.length > 1) {
        try {
            port = Integer.parseInt(argv[1]);
        }
        catch (Exception e) {
           port = 6668;
        }
    }

    // Assume that client_id_1 is a valid client ID with password_1.
    // Assume that client_id_1 has the authorization to update the
    // collection with collection ID collection_id_1.
    //
    // You need to contact the enterprise search administrator to find
    // out what client ID, password, and collection ID are valid and
    // use those valid values here.
    //
    dataPushExample_1(hostname, port, "client_id_1", "password_1",
    "collection_id_1");

    dataPushExample_2(hostname, port, "client_id_1", "password_1",
    "collection_id_1");


    // Assume that client_id_2 is a valid client ID with password_2.
    // Assume that client_id_2 has the authorization to update the
```

```
        // collection with collection ID collection_id_2.
        //
        dataPushExample_2(hostname, port, "client_id_2", "password_2",
        "collection_id_2");

        dataPushExample_3(hostname, port, "client_id_1", "password_1",
        "collection_id_1");
    }
}
```

WebSphere Information Integration に関する情報の入手

WebSphere Information Integration 製品に関する情報は、Web で入手できます。

WebSphere Information Integration に関する情報は、 Web の www.ibm.com/software/data/integration/db2ii/ にあります。このサイトには、次の最新情報が入っています。

- 製品資料
- 製品ダウンロード
- フィックスパック
- リリース情報とその他のサポート文書
- WebSphere Information Integration に関する新情報
- ホワイト・ペーパーや IBM Redbooks™ などの Web リソースへのリンク
- ニュースグループやユーザー・グループへのリンク
- WebSphere Information Integration 製品のオンライン・インフォメーション・センターへのリンク
- 資料の注文方法

製品資料を入手するには、以下のようにします。

1. Web の www.ibm.com/software/data/integration/db2ii/ にアクセスします。
2. ドロップダウン・リストから製品 (例えば、WebSphere Information Integrator OmniFind Edition) を選択します。
3. ページ左側の「Support」リンクをクリックします。
4. 「Learn」セクションで必要なリンクを選択します。選択した製品のインフォメーション・センターがある場合は、インフォメーション・センターのリンクを選択できます。 120 ページの図 4 の例を参照してください。

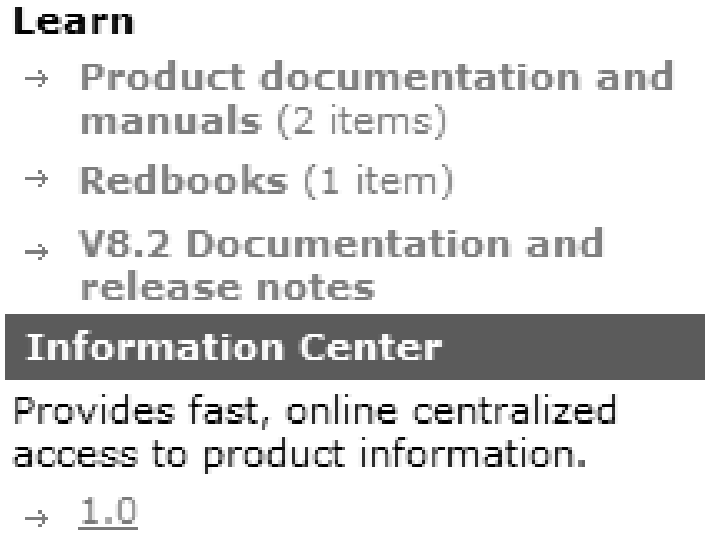


図 4. WebSphere Information Integration Support Web サイトにおける製品資料へのリンクの例

IBM と連絡を取る

お客様の国または地域で IBM に連絡する方法については、Web の www.ibm.com/planetwide にある「IBM Directory of Worldwide Contacts」にアクセスしてください。

商標

ここでは、IBM の商標と、特定の IBM 以外の商標をリストします。

IBM の商標について詳しくは、http://www.ibm.com/legal/copytrade.shtml を参照してください。

以下は、それぞれ各社の商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

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

Intel、Intel Inside (ロゴ)、および Pentium は、Intel Corporation の米国およびその他の国における商標です。

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

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

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

WebSphere II OmniFind Edition アクセシビリティ

IBM WebSphere Information Integrator OmniFind Edition のユーザー・インターフェースおよび資料はアクセス可能です。

インストール・プログラム

キーボード・ショートカットを使用して、 WebSphere II OmniFind Edition インストール・プログラム全体を移動することができます。以下のテーブルは、キーボード・ショートカットを説明しています。

表 1. インストール・プログラム用キーボード・ショートカット

アクション	ショートカット
ラジオ・ボタンの強調表示	矢印キー
ラジオ・ボタンの選択	タブ・キー
プッシュボタンの強調表示	タブ・キー
プッシュボタンの選択	Enter キー
次のウィンドウまたは前のウィンドウへ移動、またはキャンセル	タブ・キーを押してプッシュボタンを強調表示し、Enter キーを押す
アクティブ・ウィンドウを非アクティブにする	Ctrl + Alt + Esc

エンタープライズ・サーチ管理コンソールおよびインフォメーション・センター

管理コンソールおよびインフォメーション・センターは、 Microsoft Internet Explorer または Mozilla FireFox で表示することのできるブラウザー・ベースのインターフェースです。ブラウザーのキーボード・ショートカットのリストおよび他のアクセシビリティ機能については、Internet Explorer または FireFox のオンライン・ヘルプを参照してください。

PDF 資料

エンタープライズ・サーチ資料のすべてを PDF で表示できます。PDF 文書は、Adobe Acrobat Version 6.0 によって利用できます。PDF 文書は、ほとんどのスクリーン・リーダー用に構造化され、それによって読むことができるはずです。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、米国以外の国においては本書で述べる製品、サービス、またはプログラムを提供しない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。 〒106-0032 東京都港区六本木3-2-31
IBM World Trade Asia Corporation Licensing

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

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

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

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

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

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

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

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

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

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

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

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

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、 IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

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

Outside In (®) Viewer Technology, ©1992-2005 Stellent, Chicago, IL., Inc. All Rights Reserved.

索引

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

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

A

B

C

D

F

G

I

J

Printed in Japan

SD88-6375-02

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12