IBM DB2 Information Integrator

ラッパー開発者向けガイド

バージョン 8.2

SC88-9923-00
(英文原典：SC18-9174-00)

IBM DB2 Information Integrator

ラッパー開発者向けガイド

バージョン 8.2

SC88-9923-00
(英文原典：SC18-9174-00)

ご注意！

本書および本書で紹介する製品をご使用になる前に、 161 ページの『特記事項』に記載されている情報をお読みください。

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：	SC18-9174-00 IBM DB2 Information Integrator Wrapper Developer's Guide Version 8.2
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 2004.8

この文書では、平成明朝体™W3、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™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

本書について

本書は、カスタム・データ・ソースや、IBM が提供する標準のラッパー・セットでは対象としていないデータ・ソースにアクセスするためのラッパーを作成する際に役立ちます。

本書の対象読者

IBM DB2 Information Integrator 用のカスタム・データ・ソースにアクセスするためのラッパーを作成するデータ管理者、情報分析者、システム統合担当者、Web 統合担当者、データ・ライブラリアン、データ設計者、およびアプリケーション開発者。

本書の表記規則および用語

IBM DB2 Information Integrator では、データベース、接続、構造化照会言語 (SQL)、およびローカル・エリア・ネットワーク (LAN) についての概念を表す標準用語が使われています。本書で用いられている DB2 Information Integrator の概念はすべて、用語集に定義されています。特に断りがない限り、以下のような意味を想定しています。

データ 生の事実。これは、構造化、非構造化、または半構造化することができます。通常、データは分析できるよう編成されます。データはまた、決定をくだす際にも有用です。

情報 使用可能フォームのデータ。通常はいずれかの方法で処理または解釈されます。

第 1 部 フェデレーテッドの概念およびラッパーの作成の概要

本書のこの部では、ラッパーを作成するときにあらかじめ精通しておく必要のある主なフェデレーテッドの概念の概要と、全般的なラッパーの作成プロセスの概要を示します。

第 1 章 フェデレーテッドの概念の概要

以下のトピックでは、ラッパーを作成するときにあらかじめ精通しておく必要のある主なフェデレーテッドの概念について概説します。

ラッパーを作成する理由

問題: 異種のデータをアクセスまたは統合する簡単な方法がない

企業は、さまざまなブランドのリレーショナル・データ管理システム (RDBMS)、非リレーショナル・サーバー、および特殊なアプリケーション・システムに、データを保管します。このサーバーの急増は、情報技術 (IT) 業界における当然の変化によって生じました。これらの変化には、IT ショップの併合、製品およびソリューションの漸進的なアップグレード、および特化されたデータを処理するアプリケーションのニーズが含まれます。

もちろん、種々のデータ・ソースからのデータを統合すれば、このデータの新しいビューを提供するのに役立ちます。たとえば、フェデレーテッド・サーバーにある顧客データを、カスタマイズした地理情報システム (GIS) に保管されている情報と結合することができます。これによって、特定の好みを持つ顧客が住んでいる場所をマップできるようになります。または、会社の部門の併合の結果として、別々のサーバーに保管されているデータを両方の部門から取り出す必要があるかもしれません。

残念ながら、データ・ストレージの機種が異なるため、ユーザーおよびアプリケーションのインターフェースが、限られたサーバーのサブセットにアクセスするように設計されることになります。それで、ご使用のデータの多くが 1 つのインターフェースを通してアクセス可能である場合、このインターフェースがアクセスできないいくらかの重要なデータが依然としてあることになります。このため、重要なデータにアクセスするには、2 番目のインターフェースが必要になります。これを行うには、特別のアプリケーションを書いて、両方のデータ集合を同じ結果セットに統合する必要があります。

解決策: フェデレーテッド・システム

DB2® Information Integrator のソリューションは、フェデレーテッド・システム と呼ばれる、分散コンピューター・システムのセットアップを可能にします。フェデレーテッド・システムによって、クライアントは単一インターフェースからあらゆる種類のデータ・ソースを照会できます。フェデレーテッド・サーバー と呼ばれるフェデレーテッド・サーバー・インスタンスがシステムを管理します。データ・ソースは、外部サーバー と呼ばれることもあります。

DB2 UDB クライアント (エンド・ユーザーおよびアプリケーション) にとって、フェデレーテッド・システム内のデータ・ソースは、単一の集合データベースに見えます。実際には、それはフェデレーテッド・データベース と呼ばれる、DB2 UDB データベースとのクライアント・インターフェースです。フェデレーテッド・データベースはフェデレーテッド・サーバーにより管理されます。データ・ソースから

データを検索するために、クライアントは DB2 UDBの SQL ダイアレクト内の照会をフェデレーテッド・データベースにサブミットします。これは、データ・ソース内にある参照表または他のデータ・ストアを照会します。任意の数のこれらのデータ・ソースからの出力を結合する表またはビューという形で、結果を要求することができます。

例: データのアクセスおよび統合

フェデレーテッド・システム内の 2 つの非リレーショナル・データ・ソースに生命科学情報が含まれているとします。 1 つのソースでは、表に、生化学的な実験とその結果に関するデータが入っています。別のソースでは、データ・セットに、新薬の候補になる分子についての情報が入っています。フェデレーテッド・サーバーには、表およびデータ・セット (EXP および MOLECULES) に対して、それぞれの ID があります。このような ID (フェデレーテッド・サーバーが外部サーバー内で表、データ・セット、および他の種類のデータ・ストアを呼ぶ名前) は、ニックネーム と呼ばれます。

ユーザーが、ある腹部の実験で特定の結果をもたらす分子構造に類似した構造を持つ分子を検索したい、と考えているとします。これを行うために、ユーザーは、ニックネームで表およびデータ・セットを参照する照会を書きます。 図 1 はこの照会を示しています。

```
SELECT M.ID, E.MOLECULE_ID, E.RESULTS
FROM MOLECULES M, EXP E
WHERE E.EXP_TYPE = "STOMACH"
AND E.RESULTS > 0.8
AND SIMILAR_TO(E.MOLE_ID, M.ID) > .85
```

図 1. 腹部の実験で 0.8 を超える結果を持つ分子に類似した分子の ID を要求する照会 :

クライアントはこの照会をフェデレーテッド・データベースにサブミットします。フェデレーテッド・サーバーは、そのシステム・カタログを調べて、EXP が最初のサーバーにある表のニックネームであることを検出します。 MOLECULES がもう 1 つのサーバーにあるデータ・セットのニックネームであることを検出します。フェデレーテッド・サーバーは、カタログから、最初のサーバーからデータを取り出すよう指定されたラッパーが EXPERIMENTS であることを学びます。また、カタログから、2 番目のサーバーからデータを検索するよう指定されたラッパーが MYMOL であることを学びます。

フェデレーテッド・サーバーとラッパーは、照会用の実行プランを共同開発します。このプランに基づいて、フェデレーテッド・サーバーは照会をフラグメントに分解します。それから、ラッパーは、サーバーのそれぞれのアプリケーション・プログラミング・インターフェース (API) を使用して、そのフラグメントをサーバーにプッシュします。サーバーは結果を戻します。ラッパーは API からの結果を取り出して、それをフェデレーテッド・サーバーに渡します。フェデレーテッド・サーバーは結果をフェデレーテッドして、それをクライアントに戻します。

基本的なフェデレーテッド照会の概観

フェデレーテッド照会の基本的なステップは、大まかに以下のとおりです。

1. ユーザーまたはアプリケーションがある照会をサブミットします。

2. フェデレーテッド・サーバーがその照会をソースで分解します。
3. フェデレーテッド・サーバーおよびラッパーは照会プラン上でコラボレーションします。
4. フェデレーテッド・サーバーは照会実行を通してそのプランをインプリメントします。
5. ラッパーは各ソースの API を通してソースを受け取ります。
6. ソースはデータをラッパーに戻します。
7. ラッパーはそのデータをフェデレーテッド・サーバーに戻します。
8. フェデレーテッド・サーバーは、データ・ソースが処理できないデータ・ソースを補正して、異なるソースからのデータを結合します。
9. フェデレーテッド・サーバーは、そのデータをユーザーまたはアプリケーションに戻します。

照会がサブミットされると、フェデレーテッド・サーバーはそのシステム・カタログを調べます。検索する情報がどの表またはデータ・ストアに含まれているか、また検索を開始するよう指定されたのはどのラッパーか、といった情報を探します。

フェデレーテッド・サーバーは、アクセス・プランと呼ばれる、照会を評価するための代替ストラテジーを作成します。このようなプランは、データ・ソース、フェデレーテッド・サーバー、あるいは、一部はソース、一部はフェデレーテッド・サーバーで処理される照会のパーツを必要とする場合があります。フェデレーテッド・サーバーは主にコストに基づいて、そのプランの中から 1 つを選択します。

オプティマイザーは、*照会フラグメント* と呼ばれる、ユーザーのアプリケーションによってサブミットされた、オリジナルの照会のサブピースを生成します。フェデレーテッド・サーバーは、それぞれの照会フラグメントを*要求* としてラッパーにサブミットします。そのラッパーは、*応答*を使用して応答します。その応答は、ラッパーで実行できるのが特定の照会フラグメントのどのサブフラグメント (選択リスト・エレメントおよび述部) であるかをオプティマイザーに知らせます。このサブフラグメントのセットは、*受け入れ済みフラグメント* と呼ばれます。この応答では、ラッパーは受け入れ済みフラグメントを評価するよう依頼された場合、コスト (時間) または作成される行数の見積もりも行います。それから、オプティマイザーは、データ・ソースが処理できないそれらのサブフラグメントを、フェデレーテッド・サーバーの照会プラン部分に追加することによって*補正* します。全体として、ラッパーおよびフェデレーテッド・サーバーが照会プラン中に対話するプロセス全体は、*Request-Reply-Compensate (RRC)* プロトコルと呼ばれます。

フェデレーテッド・サーバーは、個々のフラグメントごとにプランの組み合わせを分析し、最も良い全体的なプランを判別します。

関連概念:

- 10 ページの『フェデレーテッド・システムの照会処理』
- 6 ページの『ラッパー・モジュール』

ラッパー・モジュール

C++ では、ラッパー・モジュールは、データ・ソースのクラスへのアクセスを提供する、特定のエントリー・ポイントを持つ共用ライブラリーです。

Java™ のラッパーは、Java クラスのセットです。

DB2® UDB は、C++ と Java の両方のラッパー・モジュールをオンデマンドで動的にロードします。

ラッパー・モジュールは、フェデレーテッド・サーバーで提供される基本クラスから得られた特定のクラスを使用して、ユーザーがコーディングするものです。これには、ユーザーのデータ・ソースとフェデレーテッド・システム間で変換プログラムとして行動するのを可能にする、特定の構築ブロックが含まれます。

図 2 は、ラッパー・モジュールの部品を示しています。

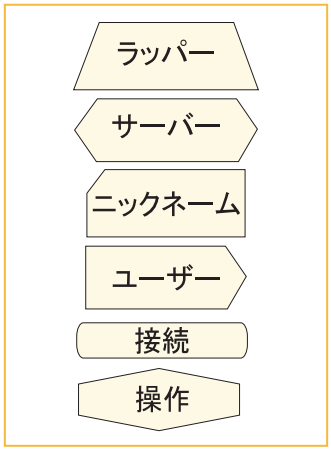

図 2. ラッパー・モジュールの部品

図 2 は、ラッパー・モジュールに以下の構築ブロックが含まれることを示しています。

- ラッパー
- サーバー
- リモート・ユーザー
- ニックネーム
- Remote_Connection
- Remote_Operation

リモート接続クラスおよびリモート操作クラスを除くすべての構築ブロックは、フェデレーテッド・サーバー・カタログに登録および記述されている永続エンティティーを表します。

表 1. ラッパー・モジュールの基本構築ブロックに関連した説明、サービス、および DDL

構築ブロック名	説明	提供されるサービス	関連した DDL
ラッパー	コード・モジュール。これは、データ・ソースのクラスを表す特定のエントリー・ポイントを持つ、共用ライブラリーです。	• ブートストラッピングおよび初期化 • ラッパーがサポートするサーバーへのアクセス	CREATE WRAPPER
サーバー	ラッパーがサポートする特定のデータ・ソースを表します。	• ニックネームのセットへのアクセス • Request-Reply-Compensate プロトコルを使用した照会プラン	CREATE SERVER
ニックネーム	サーバーに特定のデータのコレクションを表します。	ニックネーム・スキーマを、ニックネーム登録で、または DDL を経由してフェデレーテッド・サーバーに記述する。	CREATE NICKNAME
ユーザー	特定のサーバーに対してエンド・ユーザーを認証するのに必要な情報を提供します。	フェデレーテッド・サーバー・ユーザー情報をソース情報にマップします。たとえば、ユーザー ID およびパスワードなどです。	CREATE USER MAPPING
Remote_Connection	ソースへのフェデレーテッド・サーバーの接続を表します。 一時的: 永続的カタログ情報はありません。	• ソースへ接続およびソースから切断する。 • トランザクション管理。	なし
Remote_Operation	ソースでのアクティブな操作 (たとえば、照会、挿入、更新、削除) を表します。 一時的: 永続的カタログ情報はありません。 複数の操作が進行中の可能性があります。	• Remote query: データの読み取り専用照会 • Passthru: データ・ソースとの直接セッション (ソースの名前を使用するソースの言語)	なし

関連概念:

- 10 ページの『フェデレーテッド・システムの照会処理』
- 「フェデレーテッド・システム・ガイド」の『ラッパーおよびラッパー・モジュール』
- 3 ページの『ラッパーを作成する理由』

フェデレーテッド・システムにデータ・ソースを追加する方法

ユーザーまたはアプリケーションの側から見ると、データ・ソースの追加には以下のことが関係しています。

- 外部データをリレーショナル・モデル内で行および列として表す方法をフェデレーテッド・サーバーに伝える。
- データを検索するソースと通信するために、データ・ソースに対してラッパーを構成する。

ご使用のエンド・ユーザーまたはアプリケーション・プログラムが発行する、一連の DDL ステートメントを通して登録処理を行う。

ユーザーが DDL ステートメントをフェデレーテッド・サーバーにサブミットする場合、関連したラッパー・コードはそのステートメントの正確さを検査します。ラッパーは、データ・ソースからの情報を使用して、またはハードコーディングされた情報を通して、DDL 情報を増補することができます。データ・ソースは、構成パラメーターおよび統計といった情報をラッパーに提供することができます。パラメーターは、ホスト変数、入力変数、または入力データと呼ばれることもあります。ラッパーが DDL ステートメントからの情報が有効であることを検証して、おそらく情報量を増やした後、フェデレーテッド・サーバーはそれを該当するカタログに保管します。

各 DDL ステートメントに関するオプションを使用すると、ラッパーを同じデータ・ソースのいくつかの異なるバリエーションで使用できるようになります。

たとえば、フラット・ファイル用のラッパーには、フラット・ファイルへのパスを指定したオプションがあります。そのようにして、1 つのラッパーがさまざまなパスでフラット・ファイルを処理できるようになります。

ユーザーまたはアプリケーションが、新規データ・ソースを追加するために完了しなければならないステップは、以下のとおりです。

1. CREATE WRAPPER DDL ステートメントを使用してラッパーを登録する。
   a. このラッパーがどんなラッパー・オプションをサポートするかを判別する。
   b. ラッパー・オプションの値を決定する。
   c. CREATE WRAPPER DDL ステートメントを発行する。

      以下に例を示します。

      ```
      CREATE WRAPPER Dctm_Wrapper LIBRARY 'libdb2lsdctm.a';
      ```
2. サーバーを登録します。
   a. このラッパーがどんなサーバー・オプションをサポートするかを判別する。
   b. サーバー・オプションの値を決定する。
   c. CREATE SERVER DDL ステートメントを発行する。

以下に例を示します。

```
CREATE SERVER Dctm_Server1
TYPE DCTM
VERSION 3
WRAPPER Dctm_Wrapper
OPTIONS(NODE 'Dctm_Docbase',
OS_TYPE 'AIX',
RDBMS_TYPE 'ORACLE');
```

3. 必要であれば、CREATE USER MAPPING DDL ステートメントを使用して、サーバーに対してリモート・ユーザーを定義します。

   以下に例を示します。

```
CREATE USER MAPPING FOR Chuck SERVER Dctm_Server1
OPTIONS(REMOTE_AUTHID 'Charles',REMOTE_PASSWORD 'Charles_pw');
```

4. そのサーバーに特化された関数があれば登録します。
   a. CREATE FUNCTION... AS TEMPLATE DDL ステートメントを使用して、ローカルのテンプレート関数を作成します。

      以下に例を示します。

```
CREATE FUNCTION DCTM.ANY_EQ (CHAR(),CHAR())RETURNS INTEGER
AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
```

5. データ・セットのニックネームを作成する。
   a. CREATE NICKNAME ステートメントで、どんな列名および列タイプを指定するかを決定する。
   b. このラッパーがどんなニックネーム・オプションおよび列オプションをサポートするかを判別する。
   c. ニックネーム・オプションおよび列オプションの値を決定する。
   d. CREATE NICKNAME DDL ステートメントを発行する。

      以下に例を示します。

```
CREATE NICKNAME std_doc (
object_name varchar(255)not null,
object_id char(16)not null OPTIONS(REMOTE_NAME 'r_object_id'),
object_type varchar(32)not null OPTIONS(REMOTE_NAME 'r_object_type'),
title varchar(255)not null,
subject varchar(128)not null,
author varchar(32)OPTIONS(REMOTE_NAME 'authors',IS_REPEATING 'Y'),
keyword varchar(32)OPTIONS(REMOTE_NAME 'keywords',IS_REPEATING 'Y'),
creation_date timestamp OPTIONS(REMOTE_NAME 'r_creation_date'),
modifed_date timestamp OPTIONS(REMOTE_NAME 'r_modify_date'),
status varchar(16)not null OPTIONS(REMOTE_NAME 'a_status'),
content_type varchar(32)not null OPTIONS(REMOTE_NAME 'a_content_type'),
content_size integer not null OPTIONS(REMOTE_NAME 'r_content_size'),
owner_name varchar(32))
FOR SERVER Dctm_Server2 OPTIONS (REMOTE_OBJECT 'dm_document',
            IS_REG_TABLE 'N')
```

6. 標準 SQL ステートメントを使用してデータを照会する。

   以下に例を示します。

```
SELECT object_name
FROM std_doc
WHERE DCTM.ANY_EQ(author,'Joe Doe')=1
```

ラッパー・コードを書く際には、ご自分の責任で、これらの DDL ステートメントが一連の制約の中でどんなものになるかを定義する必要があります。ユーザーは、既存のどのオプションを使用するかを決定し、使用するデータ・ソースに必要な新規のオプションを作成する責任を負っています。

関連概念:

- 3 ページの『ラッパーを作成する理由』

関連タスク:

- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』

フェデレーテッド・システムの照会処理

照会処理には 2 つのフェーズがあります。照会の計画と照会の実行です。照会の計画フェーズはコンパイル時に発生し、照会の実行フェーズは実行時に発生します。図 3 は、フェデレーテッド・システムにおけるコンパイル時と実行時の照会処理の流れを示しています。

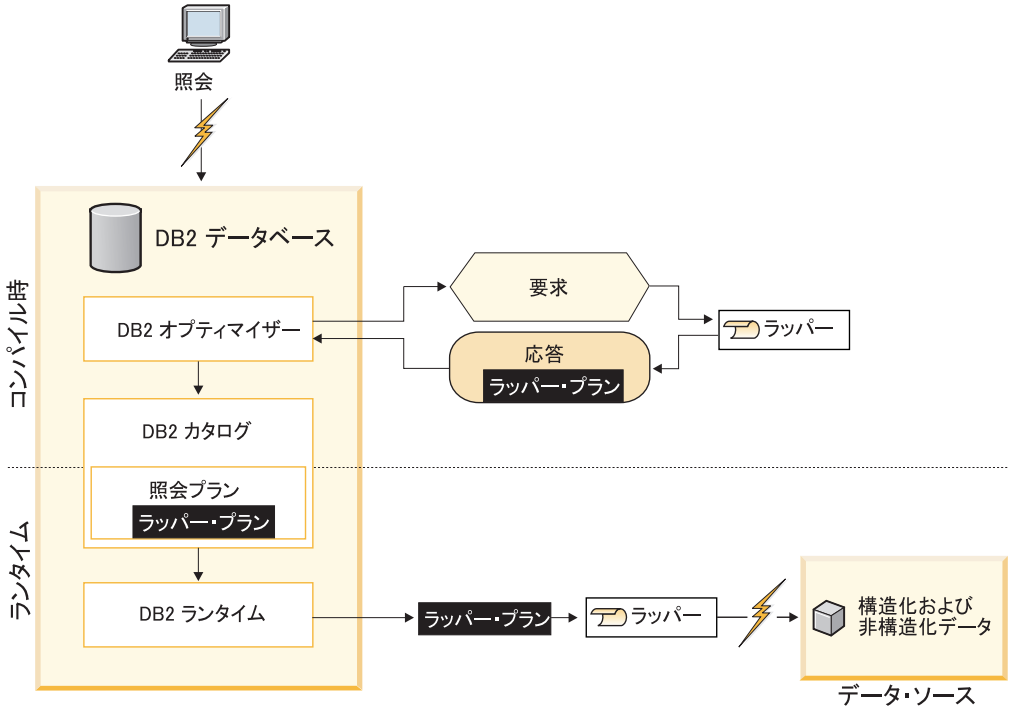

図 3. フェデレーテッド照会処理の流れ

関連概念:

- 17 ページの『フェデレーテッド・システムの照会の実行』
- 6 ページの『ラッパー・モジュール』

Request-Reply-Compensate プロトコル

照会の計画中にオプティマイザーは、*照会フラグメント*と呼ばれる、ユーザーのアプリケーションによってサブミットされた、オリジナルの照会のサブピースを生成します。照会フラグメントには、表、述部、およびヘッド式を含めることができます。ヘッド式は、照会の SELECT 文節にある式のことです。それから、オプティマイザーは、それぞれの照会フラグメントを*要求* 内のラッパーにサブミットします。

定義によると、フラグメントの評価に必要なデータはすべて、1 つのデータ・ソースに由来します。しかし、このデータは、外部サーバーまたはフェデレーテッド・サーバーで処理可能であり、その一部はどちらででも処理可能です。ラッパーは、*サブフラグメント* と呼ばれるフラグメントのどのサブピースを評価できるかを示し、*要求への応答* にこの情報を書き込みます。ラッパーは、サブフラグメント全体を受け入れるかリジェクトするかしなければなりません。

応答には以下のものが含まれます。

- 受け入れられるニックネーム、述部、およびヘッド式。
- 受け入れられるフラグメントのコストとカーディナリティーの見積もり。
- もしあれば、戻される予定の結果の順序付けプロパティー。
- ラッパー実行記述子 (以下のテキストで説明されている)。

この資料では、「数量詞」と「ニックネーム」は交換可能です。

単一照会の場合、オプティマイザーは、通常それぞれのラッパー、オリジナルの照会の異なるフラグメントを表すそれぞれの要求ごとに、多くの要求を生成します。このような要求ごとに、ラッパーは、ゼロ、1 つ、または複数の応答を生成します。それぞれの応答は、異なる、受け入れられたフラグメントを表します。受け入れられるフラグメントとは、ラッパーまたはデータ・ソースがそれ自身を評価できるフラグメントのことです。それぞれの応答には、受け入れられるフラグメントの関連コストとカーディナリティーの見積もりが含まれます。

照会の計画の終わりまでに、オプティマイザーはコストに基づいた決定をします。要求に応じて、ラッパーによって提案された、受け入れられたフラグメントのいくつかのセットを組み込んだプランを提示します。ラッパーが最終的に実行するよう求められるのは、この特定のセットの受け入れられたフラグメントです。

オプティマイザーは、選択したプランにおいて受け入れられるフラグメントの一部ではない、任意のサブフラグメントをフェデレーテッド・サーバーが評価するようにします。この例としては、調査対象のデータ・ソースに、能力を超えた複雑な述部やソートが含まれる場合があります。フラグメントは単一ソースであるため、これには、ソース間のすべての結合に加えて、複数のソースからのデータを混合する他のすべての式 (関数呼び出しなど) が含まれます。これは、*適合* と呼ばれます。

総合的に、ラッパーおよびフェデレーテッド・サーバーがコンパイル中に対話するプロセス全体は、*Request-Reply-Compensate (RRC)* プロトコルと呼ばれます。

ラッパーは、コストおよびカーディナリティー見積もりを決定して、自身の応答内に書き込むか、またはフェデレーテッド・サーバーが提供するデフォルトのコスト・モデルを再使用することができます。また、新規モデルを初めから作成しなく

ても、デフォルトのコスト・モデルの一部を選択的に置き換えて、その精度を向上させることができます。デフォルトのモデルは、ラッパーがデータから計算できる、または DDL を経由して提供される統計を使用します。

応答の一環として、ラッパーは *ラッパー実行記述子* を提供する必要もあります。これはブラック・ボックスであって、その内容はラッパー次第です。ラッパーは、データ・ソースに対して応答が示す、受け入れられたフラグメントをサブミットできなければなりません。オプティマイザーが最終的に選択する実行プランの中の受け入れられたフラグメントを使用する場合、照会を実行する時間になると、ラッパー実行記述子がラッパーに戻されます。その間、ラッパー実行記述子は、プリコンパイルされた照会のためのアクセス・プランの一部として、フェデレーテッド・サーバー・カタログに常駐します。

ソース用のラッパーは、ラッパー実行記述子にあるリモート照会の永続表現を完全に制御します。

ハンドルによる要求および応答の操作

DB2 UDB は、要求および応答を操作するための機能インターフェースを提供します。

要求は SQL 照会に対応しますが、データ・ソースは SQL を理解できないため、この照会は SQL では表されません。その代わりに、整数ハンドルを使用して照会式を識別およびナビゲートします。述部およびヘッド式は、演算子ツリーとして表されます。ノードの各種類について記述し、任意のノードの子をナビゲートするために、関数が提供されます。用語を要求から応答へ移動するために、メソッドが提供されます。

Request-Reply-Compensate プロトコルの例

たとえば、図 4 にある例を考慮してください。

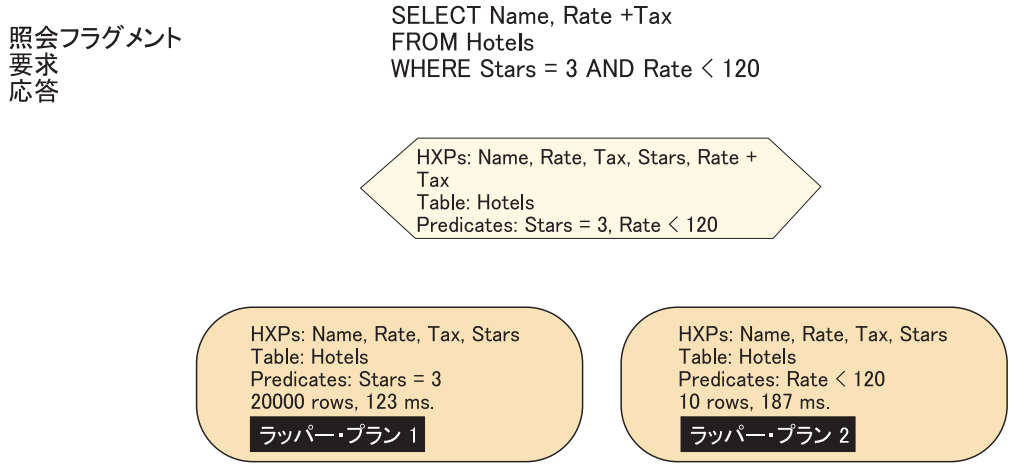

図 4. Request-Reply-Compensate プロトコルの例

この図は RRC プロトコルが実行中であることを示しています。このシナリオでは、ユーザーは、Web サーバーへの要求に対して 1 つの述部のみを指定できる Web サイトであるデータ・ソースにアクセスしたいと考えています。

この図は、単一テーブル・アクセス照会の照会フラグメントを示しています。SELECT リストに rate + tax という式が含まれることに注意してください。

この要求では、フェデレーテッド・サーバーは、ヘッド式 rate、tax、stars のリストの述部および式に個々に含まれている列を要求することができます。

異なる述部が受け入れられた 2 つの応答、コストおよびカーディナリティー、およびラッパー・プラン (ラッパー実行記述子とも呼ばれる) があります。どのプランもヘッド式 rate + tax を受け入れません。ラッパーおよびデータ・ソースはこのヘッド式を処理できないため、フェデレーテッド・サーバーはレートおよび税を求める際に、その値を個々の列値から計算します。

関連概念:

- 10 ページの『フェデレーテッド・システムの照会処理』
- 17 ページの『フェデレーテッド・システムの照会の実行』
- 65 ページの『照会の実行のための制御フロー』

フェデレーテッド照会のデフォルト・コスト・モデル

このセクションでは、ラッパーがコスト・モデルを使用して、フェデレーテッド・サーバー・オプティマイザーにコスト情報を提供する方法を説明します。

照会コストおよび照会プラン

Reply クラスを通して、ラッパーはフェデレーテッド・サーバーに情報を提供します。フェデレーテッド・サーバーは、特定の照会を処理するために最も適切なプランを決定する際にこの情報を使用します。 Reply クラスは以下の 4 つの情報を計算するメソッドを提供します。

1. 応答によって表される照会フラグメントのカーディナリティー。これは、照会フラグメントが戻す行数の見積もりです。
2. 照会フラグメントに選択された最初のタプルを検索する時間の見積もり (ミリ秒)。この見積もりは、照会フラグメントの最初の呼び出しのためのものです。これは、フラグメントの*第 1 タプル・コスト* と呼ばれます。
3. 照会フラグメントに選択された全体の応答セットを検索する時間の見積もり (ミリ秒)。この見積もりは、照会フラグメントの最初の呼び出しのためのものです。
4. 照会フラグメントに選択された全体の応答セットを検索する時間の見積もり (ミリ秒)。この見積もりは、おそらく新規パラメーターがバインドされた、照会フラグメントの 2 番目またはそれ以降の呼び出しのためのものです。これは、フラグメントの*再実行コスト*と呼ばれます。ラッパーが、照会フラグメントを初めてリモート・ソースにサブミットする際に行わなければならないプリプロセスがある場合、この再実行コストは第 1 タプル・コストとは異なります。ラッパーが同じフラグメントをサブミットする場合、プリプロセスを繰り返す必要はありません。

ラッパーは、Reply クラスをサブクラス化して、そのコストを計算するための自身のメカニズムを提供することができます。または、Reply クラスが提供するデフォ

ルトのコスト・モデルを使用することができます。以下のセクションでは、デフォルトのコスト・モデルについて説明します。

デフォルトのコスト・モデル

デフォルトの Reply クラスがインプリメントするコスト・モデルは、照会フラグメントに含まれるそれぞれのニックネームごとに、統計情報により駆動されたデフォルト設定のコスト式を使用して、以前にリストされた 4 つの情報を計算します。デフォルトのコスト・モデルを使用するラッパーは、それぞれのニックネームごとに、以下のセクションでリストされている 4 つの統計を提供する必要があります。

デフォルトのコスト・モデルの統計

デフォルトでは、フェデレーテッド・サーバーはこれらの統計をシステム・カタログに保管します。該当するメソッドをオーバーライドすることによって、ラッパーはそれを異なる方法で提供することができます。ラッパーはこれらのメソッドをオーバーライドする一方、残りのデフォルトのコスト・モデルはそのままにすることができます。

4 つの統計は以下のとおりです。

1. ニックネームのカーディナリティー。これはニックネームに含まれる行数として定義されます。フェデレーテッド・サーバーは個々のニックネームのカーディナリティーを、システム表 SYSCAT.TABLES または SYSSTAT.TABLES (のいずれかの表の「CARD」列) に格納します。カーディナリティーをニックネームに使用できない場合、コスト・モデルはデフォルト値の 1000 行を使用します。
2. ニックネームのセットアップ・コスト。セットアップ・コストは、ラッパーが照会フラグメントをリモート・ソースにサブミットできる状態にするまでにかかる標準的な時間 (ミリ秒単位) を表します。セットアップは、ラッパーが、照会の計画時に生成したラッパー実行記述子を受け取った時点から始まり、ラッパーが対応する操作をリモート・ソースにサブミットできるようになった時点で終わります。セットアップ・コストには、ラッパーが同じ照会フラグメントを (おそらくは異なるパラメーター値を使用して) 再度実行するよう要求されても繰り返す必要がない作業のみが含まれているべきです。たとえば、ラッパーが照会フラグメントを URL の形式のリモート・ソースにサブミットする場合、セットアップ・コストには、ラッパーが実行記述子に格納した情報から URL を生成するのに必要な時間が含まれます。フェデレーテッド・サーバーはこの統計を SETUP_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 25 ミリ秒の値を使用します。
3. ニックネームのサブミット・コスト。サブミット・コストは、ラッパーが照会フラグメントをリモート・ソースにサブミットするのにかかる標準的な時間 (ミリ秒単位) を表します。サブミットは、上で定義されているセットアップが終了した時点から始まり、ラッパーがソースに対して最初の行または結果データのブロックを要求できるようになった時点で終わります。サブミット・コストには、特定の照会フラグメントがサブミットされるたびにラッパーが繰り返す必要がある作業のみが含まれているべきです。たとえば、リモート・ソースとの各対話用に新しい HTTP 接続が必要な場合、サブミット・コストには、この接続を作成するのに必要な時間が含まれていなければなりません。フェデレーテッド・サーバ

ーはこの統計を SUBMISSION_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 2000 ミリ秒の値を使用します。

4. ニックネームの拡張コスト。これは、ニックネーム用の単一行を取り出すのにかかる標準的な時間 (ミリ秒単位) です。これには、照会を開始するのに必要な時間は含まれません。フェデレーテッド・サーバーはこの統計を ADVANCE_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 50 ミリ秒の値を使用します。データ・ソースがデータを行単位ではなくブロック単位で戻す場合は、ブロックを取り出すための標準コストを、ブロック当たりの標準行数で除算することによって、拡張コストを計算します。

ラッパー・コードを装備することによって必要な統計を入手できますが、通常は、疑問視されるニックネームに対する一般的な照会の実行時間を外部から測定するほうが、より簡単に統計を入手できます。たとえば、それぞれの照会が戻す結果の数が異なる 2 つの照会を実行することによって、また実行時間の差をそれぞれの照会が戻す行数の差で除算することによって、アドバンス・コストを見出すことができます。これは、特に 3 コスト統計に適用されます。多くの場合、データ・ソースにはカーディナリティーを決定するための直接法があります。

デフォルトのコスト式

デフォルトのコスト・モデルは、オプティマイザーが必要とする 4 つのパラメーターを導出する 4 つのコスト式のセットを提供します。以下のリストはそれぞれの式について説明しています。

カーディナリティー

照会フラグメントのカーディナリティーは 2 つのステップで計算されます。最初に、コスト・モデルはそれぞれのニックネームごとにカーディナリティーを入手して、その値を乗算します。

2 番目のステップでは、そのコスト・モデルが、照会フラグメント内の述部の選択度によって、最初のステップで計算された全体の行数を乗算します。この選択度は、0.0 から 1.0 までの数です。これは、結果セットから取り出されるニックネームからの、述部フィルター行の反映の度合いを示すもので、値が小さいほど、より多くフィルターがかかっていることを示しています。デフォルトのコスト・モデルは、フェデレーテッド・サーバーが述部選択度を見積もるために提供するアルゴリズムを使用するメソッドを提供します。ラッパーは、より正確な選択度の見積もりを提供できる場合、このメソッドをオーバーライドすることができます。選択度におけるエラーの主要な要因は、あるセット内の述部の相関であり、デフォルトのモデルはこれをまったく理解しません。

たとえば、述部 `Make='Carmaker1_make'` の選択度が 0.13 で、`Model='Carmaker1_model'` の選択度が 0.05、`Make='Car1_make' AND Model='Car1_model'` の結合した選択度も 0.05 であるとしても、属性 `Make` および `Model` の値が独立していないため、どの自動車モデルも同じ自動車メーカーのものになります。ご使用のラッパーが属性間の相関を理解している場合、デフォルトの選択度見積もりメソッドをオーバーライドすることができます。また、ある属性に対する値の分散が極めて偏っている場合も、選択度見積もりメソッドを提供することができます。残りのデフォルトのコス

ト・モデルを保持する一方で、これを行うことができます。さらに、カスタム選択度見積もりメソッドを提供する場合、フェデレーテッド・サーバー・オプティマイザーもこのカスタム選択度見積もりメソッドを呼び出して、ラッパーが応答内で受け入れなかった任意の述部の選択度を計算します。前に述べたとおり、ラッパーは、述部がソースによって受け入れられた照会フラグメントの一部として評価されない場合でも、述部間のスキューまたは相関を理解して、さらに適した見積もりを提供することができます。

第 1 タプル・コスト

第 1 タプル・コストは、3 つの値の合計です。最初の値は、照会フラグメントにあるすべてのニックネームのセットアップ・コストの平均です。 2 番目の値は、照会フラグメントにあるすべてのニックネームのサブミット・コストの平均です。 3 番目の値は、照会フラグメントにあるすべてのニックネームのアドバンス・コストの平均です。

合計コスト

合計 (最初の応答セット) コストは、3 つの値の合計です。最初の値は、照会フラグメントにあるすべてのニックネームのセットアップ・コストの平均です。 2 番目の値は、サブミット・コストの平均です。 3 番目の値は、アドバンス・コストの平均を、照会フラグメントの見積もりカーディナリティーで乗算した積です。

再実行コスト

照会フラグメントの再実行コストは、2 つの値を合計したものです。最初の値は、照会フラグメントにあるすべてのニックネームのサブミット・コストの平均です。 2 番目の値は、アドバンス・コストの平均を、照会フラグメントの見積もりカーディナリティーで乗算した積です。実行時間の見積もりを計算する際、コスト式は通常、照会フラグメントにあるすべてのニックネームにおける統計の平均値を使用します。これによって、ご使用のデータ・ソースにとって妥当な結果が出ないように思われる場合、Reply オブジェクトをサブクラス化することによってコスト式をオーバーライドすることを検討してください。

コストおよび照会プランのサマリー

照会フラグメントのコストの計算に関しては、いくつかのオプションがあります。17 ページの図 5 はこれらのオプションを示しています。

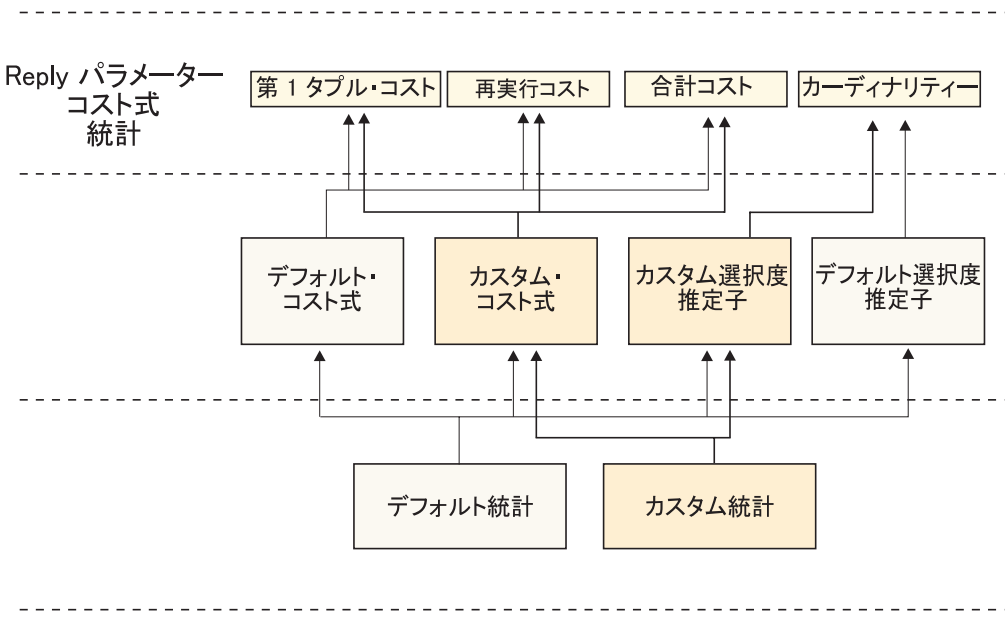

図 5. コストが機能する仕組み

複雑さを増大する、また能力および柔軟性を増大するおおよその順序は以下のとおりです。

1. デフォルトのコスト・モデルをそのまま受け入れる。
2. ニックネームの 4 つの統計 (カーディナリティー、平均セットアップ・コスト、平均サブミット・コストおよび平均アドバンス・コスト) を計算する方法をオーバーライドする。
3. デフォルトの選択度計算をオーバーライドする。
4. Reply オブジェクトをサブクラス化することで全体のコスト・モデルを置き換える。

関連概念:

- 27 ページの『データ・ソースの照会の相対コスト』
- 63 ページの『照会の計画のための制御フロー』

フェデレーテッド・システムの照会の実行

照会の計画が完了したら、フェデレーテッド・サーバーで照会を実行できます。まずフェデレーテッド・サーバーは、個々のデータ・ソースに割り当てられた照会フラグメントを、対応するラッパーに配布します。次に、ラッパーがフラグメントをデータ・ソースにサブミットし、その結果を取り出します。フェデレーテッド・サーバーは結果を結合し、さらに処理を行います。このステップは代表的なもので、特定のソースが接続の概念を取り扱う方法、ソースが照会言語やその他の API を介して要求を受け入れるかどうか、ソースでサポートされている結果セット・カーソルまたはイテレーター (ある場合) の種類などに応じて変わります。 18 ページの図 6 で、照会を配布して実行し、その結果を取り出す場合に通常取られる処置を概観できます。

照会を配布して実行し、その結果を取り出す代表的なプロセス

1. フェデレーテッド・サーバーは、CREATE USER MAPPING ステートメント中に指定された許可情報をラッパーに渡します。
2. フェデレーテッド・サーバーは、データ・ソースに対する接続を確立するようにラッパーに要求します。
3. ラッパーは、フェデレーテッド・サーバーが要求した接続を確立します。
4. ラッパーは、この照会のフラグメントの計画時に作成されたラッパー実行記述子を取得します。ラッパー実行記述子には、照会をデータ・ソースにサブミットできるだけの情報が含まれていなければなりません。
5. ラッパーは、変換された照会フラグメントを、照会が参照するデータ・ソースにサブミットします。次にラッパーは、取り出そうとしている結果セットのイテレーターを入手します。
6. フェデレーテッド・サーバーは、ラッパーから結果の行を要求します。その結果、ラッパーは該当するデータ・ソースに要求を『転送』します。
7. データ・ソースは、照会フラグメントを実行して、要求された行を戻します。
8. ラッパーは、要求された行を取り出します。ラッパーはさらに行のデータのタイプをフェデレーテッド・サーバーのデータ・タイプに変換し、変換されたタイプをバッファーの中にコピーします。
9. フェデレーテッド・サーバー、データ・ソース、およびラッパーは、結果の後続行ごとにステップ 7、8、および 9 を繰り返します。
10. ラッパーは、結果の最後の行を取り出します。多数の作業単位が参照なしで実行されてから、フェデレーテッド・サーバーはデータ・ソースから切断します。
11. ラッパーは、データ・ソースから切断します。

図 6. 照会を配布して実行し、その結果を取り出す代表的なプロセス

関連概念:

- 10 ページの『フェデレーテッド・システムの照会処理』
- 6 ページの『ラッパー・モジュール』

ラッパーでのパススルーの使用

アプリケーションでパススルーを使用すると、照会やその他の要求を外部データ・ソースに直接サブミットできます。照会やその他の要求は、データ・ソース固有の照会言語を使用します。パススルーにより、データを行および列として取り出せます。パススルーによって入手した結果は、他のソースからの結果と結合したり、他の方法で結合したりできません。パススルーのインプリメンテーションはオプションです。パススルーは、接続の問題をデバッグするのに役立ちますし、アプリケーションで外部ソースに対して管理コマンドを実行できるようになります。

関連タスク:

- 112 ページの『リモート passthru クラス』

第 2 章 ラッパーの作成の概要

以下のトピックでは、ラッパー作成プロセスと、ラッパーを作成するために使用するラッパー開発キットについての概要を示します。

ラッパーの作成プロセス

特定のデータ・ソース用のラッパーの作成者は、ラッパー・モジュールを作成する際の基本的な開発の流れを知っている必要があります。

表 2. ラッパー作成プロセスの段階

段階	サブタスク
概念	フェデレーテッドの概念について理解する
	一般的なラッパー作成プロセスについて理解する
設計	データ・ソースについて理解する
	データ・ソースのリレーショナル・モデルを開発する
	フェデレーテッド構成ごとにオプションを決める
	フェデレーテッド構成をご使用のデータ・ソースにマップする
	ご使用のデータ・ソースでサポートする照会の種類を決める
	コスト・モデルを設計する
	パススルー用に設計する
	エラー処理用に設計する
コーディング	ラッパー・サブクラスをコーディングする
	登録用にコーディングする
	初期化用にコーディングする
	照会プラン用にコーディングする
	照会実行用にコーディングする
	パススルー用にコーディングする
	移植の問題を考慮する
文書化	ラッパーを文書化する
ビルドとパッケージ	ご使用のラッパーをコンパイルする
	ご使用のラッパーをリンクする
	ご使用のラッパーをパッケージする
	ご使用のラッパーをインストールする
テスト	ご使用の SQL ステートメントをテストする
	ご使用のラッパーのデバッグとトレースを行う
	ご使用のラッパー・オプションをテストする
	広範囲の照会をテストする

関連概念:

- 69 ページの『ラッパーを開発するための一般的な手順』
- 72 ページの『ラッパーの作成のためのヒント』
- 20 ページの『ラッパー開発キット』

ラッパー開発キット

DB2® Information Integrator には、C++ および Java™ でラッパーを作成するための Software Development Kit (SDK) が組み込まれています。

ラッパー開発キットには、以下のものが含まれています。

- サンプル C++ ラッパー
- サンプル Java ラッパー
- DB2 コントロール・センターにラッパーを追加するためのツールおよびサンプル

デフォルトの Windows® のディレクトリー・パスは、C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

サンプル C++ ラッパー

表 3 は、各プラットフォームごとの、サンプル C++ ラッパーが置かれるディレクトリーを示しています。

表 3. プラットフォーム別のサンプル C++ ラッパーのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX®	/usr/opt/db2_08_01/samples/wrapper_sdk
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk
Windows	%DB2PATH%¥samples¥wrapper_sdk

サンプル C++ ラッパーには、以下のものが含まれています。

- ラッパー API を表示するヘッダー・ファイル (ラッパー・クラス宣言)
- ラッパーをフェデレーテッド・サーバーにリンクできるファイル
- ラッパー共通ライブラリー (カスタム・ラッパーのライブラリーをロードして呼び出すスタブ・ライブラリー)
- ラッパー開発用の C++ API の使用法を説明するために使われるサンプル・ラッパーのソース・コード
- サンプル・ラッパーを構築するためのサンプル Makefile

サンプル Java ラッパー

21 ページの表 4 は、各プラットフォームごとの、サンプル Java ラッパーが置かれるディレクトリーを示しています。

表 4. プラットフォーム別のサンプル Java ラッパーのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk_java
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk_java
Windows	%DB2PATH%¥samples¥wrapper_sdk_java

サンプル Java ラッパーには、以下のものが含まれています。

- Java API のクラスおよびメソッドについて説明する Javadoc
- ラッパー開発用の Java API の使用法を説明するために使われるサンプル・ラッパーのソース・コード

DB2 コントロール・センターにラッパーを追加するためのツールおよびサンプル

ラッパー開発キットには、DB2 コントロール・センターにカスタム・ラッパーのサポートを追加するのに役立つ、以下のツールおよびサンプル・ファイルが組み込まれています。

- DB2 コントロール・センターのオプションにカスタム・ラッパーを追加するための構成ファイルを作成する「XML 構成ファイルの作成 (Develop XML Configuration File)」ウィザード。表 5 は、各プラットフォームごとの、このウィザードを開始するファイルが含まれるディレクトリーを示しています。

表 5. プラットフォーム別の「XML 構成ファイルの作成 (Develop XML Configuration File)」ウィザードを開始するためのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%¥bin¥db2wrapperconfig.bat

- 「XML 構成ファイルの作成 (Develop XML Configuration File)」ウィザードからのサンプルの出力ファイル。表 6 は、各プラットフォームごとの、サンプルの出力ファイルが含まれるディレクトリーを示しています。

表 6. プラットフォーム別の「XML 構成ファイルの作成 (Develop XML Configuration File)」ウィザードからのサンプルの出力ファイルのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk/cc_plugin
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk/cc_plugin
Windows	%DB2PATH%¥samples¥wrapper_sdk¥cc_plugin

- ラッパーで DB2 コントロール・センターのディスカバリー・フィーチャーをサポートする場合に使用できる、基本ディスカバリー・ツール。このツールは、ラッパーのデータ・ソースについて発見したものをすべて表示する、単純な Java GUI です。このツールは、DB2 コントロール・センターにも組み込まれています。 22 ページの表 7 は、各プラットフォームごとの、Java .jar ファイルとしてこのツールを備えているディレクトリーを示しています。

表 7. プラットフォーム別の基本ディスカバリー・ツールのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/tools/db2WrapperDiscoverySDK.jar
HP/Sun/Linux	/opt/IBM/db2/V8.1/tools/db2WrapperDiscoverySDK.jar
Windows	%DB2PATH%¥tools¥db2WrapperDiscoverySDK.jar

- ここに準備されているサンプル Java ストアード・プロシージャーは、ラッパーの作成者がコントロール・センターに対するプラグインを作成するために、組み込みディスカバリーをどのように役立てることができるかを示す例です。 表 8 は、ストアード・プロシージャー、ストアード・プロシージャーをコンパイルするための Makefile、およびフェデレーテッド・サーバーにマークアップ・ファイルをインストールするためのスクリプトが含まれているディレクトリーを示しています。

表 8. プラットフォーム別のサンプル Java ストアード・プロシージャーのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk¥cc_plugin
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk¥cc_plugin
Windows	%DB2PATH%¥samples¥wrapper_sdk¥cc_plugin

関連概念:

- 19 ページの『ラッパーの作成プロセス』
- 69 ページの『ラッパーを開発するための一般的な手順』

関連タスク:

- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』
- 「*IBM DB2 Information Integrator インストール・ガイド*」の『ラッパー開発キットのインストール』

第 2 部 データ・ソース用のラッパーの設計

本書のこの部では、ラッパーの設計に必要な次のようなタスクを順に説明します。

- データ・ソースの特性を判別することにより、データ・ソースで効果的に機能するラッパーを設計する
- データ・ソースをフェデレーテッド構成にマップする
- ソースが受け入れることのできる SQL 構成を判別する
- ラッパーのエラー処理を設計する

第 3 章 データ・ソースの特性の判別

ラッパーの設計を始める前に、ご使用のデータ・ソースについて理解しておく必要があります。以下の質問に答える必要があります。

- データ・ソースは API の選択を可能にしているか?
- インターフェースはどんな種類の操作をサポートしているか?
- データ・ソースには、スキーマまたは基本データの他のプロパティーを記述するメタデータが含まれるか?
- データ・ソースはどんな種類の照会に応答できるか?
- 他の照会よりも効果的に応答できる照会があるか?
- データ・ソースに複数のインスタンスがある場合、インスタンスを異ならせるものは何か、また常に同じものは何か?
- データ・ソースはクライアント/サーバー通信をサポートするか?
- フェデレーテッド・サーバーが実行するプラットフォームに対して使用可能なデータ・ソース・クライアントがあるか?
- データ・ソースはトランザクション・モデルをサポートするか? その場合、分散 (1 フェーズ) コミット・プロトコルをサポートするか?
- データ・ソースはユーザーをどのように認証するか?
- データ・ソースはラージ・オブジェクトをサポートするか?

以下のセクションでは、前述の質問のそれぞれについての詳細情報および例を説明します。

データ・ソース用の API の選択

データ・ソースの API について学習する際には、以下の点を調べてください。

- API がサポートする機能。たとえば、この API を介して、サブミットする必要のある照会をサブミットできるか ? ラッパーがこの API を介して、データ・ソースによって管理されるデータのスキーマまたは統計のプロパティーを記述するメタデータにアクセスできるか ? この API はトランザクションをサポートしているか ? 非リレーショナル・ソースへのアクセスが読み取り専用の場合でも、ラッパーは、リレーショナル・ソースが更新されるトランザクションに参加できます。
- API に必要な情報。たとえば、ラッパーがデータ・ソースとの接続を確立しようとしている際に、API に必要な情報は何か ? 結果セットを取り出すのに必要な情報は何か ?

選択した API をサポートするデータ・ソース・クライアントは、データ・ソースにアクセスする DB2 UDB フェデレーテッド・データベース・サーバーを実行する個々のプラットフォーム上で使用できなければなりません。

関連概念:

- 26 ページの『データ・ソースのインターフェースによってサポートされる操作』

- 26 ページの『データ・ソースのメタデータ』

データ・ソースのインターフェースによってサポートされる操作

以下の質問を利用すると、データ・ソースの個々のインターフェースによってサポートされる操作について理解するのに役立ちます。

- データ・ソースを使用して、データ値を基にした情報を選択して取り出せるか ? データ・ソースは述部を適用できるか ?
- データ・ソースを使用して、部分エンティティーを取り出せるか ? データ・ソースは射影をサポートするか?
- データ・ソースは、リンクまたは一致データ値を基にした複数のコレクション中のデータを結合できるか ? 結合は行えるか ?
- データ・ソースは、標準 SQL で表せない特殊な検索機能をサポートするか ? ソースの検索機能をモデル化するのに必要なマップ関数は何か ?
- データ・ソースは保管データ値だけを戻すのか、それとも保管データ値と他の値や定数を結合した式の値を戻すのか ?
- 述部に関するデータ・ソースの意味構造は、フェデレーテッド・サーバーの意味構造と一致しているか ?

関連概念:

- 26 ページの『データ・ソースのメタデータ』
- 27 ページの『データ・ソースの照会の相対コスト』
- 31 ページの『ニックネームおよび列オプションの決定』

関連タスク:

- 25 ページの『データ・ソース用の API の選択』

データ・ソースのメタデータ

直接データ・ソースから入手できる情報が増えるほど、データ・ソースを登録する際にデータベース管理者が行う必要のある作業は少なくなります。特に、スキーマの詳細情報をデータ・ソースから入手できる場合、その情報を CREATE NICKNAME ステートメントから省略できます。同様に、ラッパーのコスト・モデル (カスタムまたはデフォルト) で使用されるデータ・ソース統計メタデータから情報を入手できる場合もあります。

関連概念:

- 26 ページの『データ・ソースのインターフェースによってサポートされる操作』
- 27 ページの『データ・ソースの複数のインスタンス』
- 27 ページの『データ・ソースのクライアント/サーバー通信』

データ・ソースの照会の相対コスト

データ・ソースが応答できるさまざまな照会の相対的コストについて理解することは重要です。フェデレーテッド・サーバーからの照会フラグメントの場合、たいていの場合はデータ・ソースでこのフラグメントの全部または一部を実行する方法が複数あります。ラッパーは、選択肢ごとの正確なコスト見積もりを提供できなければなりません。

関連概念:

- 26 ページの『データ・ソースのインターフェースによってサポートされる操作』

関連タスク:

- 25 ページの『データ・ソース用の API の選択』

データ・ソースの複数のインスタンス

ほとんどのデータ・ソースは、実際には 1 つの種類ではありません。同じ種類だとすると、特定のソースに接続して使用するのに必要なすべての情報 (スキーマを含む) をラッパー中にハードコーディングできることになります。実際には、DBA がこの情報を使用してラッパーをデータ・ソースの特定のインスタンス用にカスタマイズできるようにする必要があります。この情報の指定内容は DDL を使用するため大きくなるので、ラッパーを設計する際には、どの情報がインスタンス間で異なるか判別することが重要になります。この情報を使用して、DBA が必要な情報を指定できるようにするオプションを設定できます。スキーマがインスタンス間で異なる場合は、その一部をハードコーディングし、残りを DDL を使用して指定できます。

関連概念:

- 26 ページの『データ・ソースのメタデータ』
- 27 ページの『データ・ソースのクライアント/サーバー通信』

データ・ソースのクライアント/サーバー通信

ラッパーは、フェデレーテッド・サーバーと同じコンピューター上で実行します。このサーバーは、データ・ソースをフェデレーテッド・システム中に組み込みます。アクセスしたいデータ・ソースをフェデレーテッド・サーバーとは異なるコンピューター上で実行する場合は、データ・ソースのクライアント/サーバー通信機能を使用しなければなりません。ラッパーとフェデレーテッド・サーバー全体は、実質的にデータ・ソースのクライアントになります。データ・ソースがクライアント/サーバー通信をサポートしない場合は、ラッパー自体の一部として、またはラッパー自体に追加して、この機能を設ける必要があります。データ・ソースとフェデレーテッド・サーバーを同じコンピューター上で実行すると、問題を解決できる場合があります。

関連概念:

- 28 ページの『データ・ソースのトランザクション・モデルと分散コミット・プロトコル』

- 28 ページの『データ・ソースからのユーザー認証』
- 29 ページの『データ・ソースからのラージ・オブジェクト・サポート』

データ・ソースのトランザクション・モデルと分散コミット・プロトコル

トランザクションとは、更新のグループをデータベース・アトミックに行う機構のことです。更新のグループ全体が同時に行われるか、または更新が全く行われないかのいずれかです。トランザクションに複数のデータ・ソースが関係している場合は、これらのソースを連携して、すべてのソース間でトランザクションの原子性を確実に保持しなければなりません。 DB2® Information Integrator は XA 分散コミット・プロトコルを使用して、ソース間のトランザクション管理を調整します。現行バージョンの DB2 Information Integrator は非リレーショナル・データ・ソースに対する更新をサポートしていませんが、データ・ソースがトランザクションをサポートする場合は、ラッパーがトランザクション管理に参加しなければなりません。ラッパーがトランザクション管理に参加しなければならないのは、ロックをかけたトランザクションが (正常であれ、異常であれ) 完了したときに、データ・ソースで取得したロックをドロップできるようにするためです。

関連概念:

- 27 ページの『データ・ソースのクライアント/サーバー通信』
- 28 ページの『データ・ソースからのユーザー認証』
- 29 ページの『データ・ソースからのラージ・オブジェクト・サポート』

データ・ソースからのユーザー認証

フェデレーテッド・システムのクライアントは、サポートされる認証方式のいずれかを使用して、フェデレーテッド・サーバーに対する認証を行います。これらのクライアントに代わってフェデレーテッド・サーバーがデータ・ソースへのアクセスを試行する際には、そのデータ・ソースが有効であると認識し、受諾している証明書があることが必要です。フェデレーテッド・サーバーはこれらの証明書をフェデレーテッド・サーバー・カタログ中にユーザー・マッピングとして保管します。ラッパーを設計する際には、データ・ソースが予期している証明書 (ユーザー ID やパスワードなど) が分かっていなければなりません。それが分かっていれば、その証明書をフェデレーテッド・サーバー・カタログ中に保管できるユーザー・マッピング・オプションを定義できます。

関連概念:

- 27 ページの『データ・ソースのクライアント/サーバー通信』
- 28 ページの『データ・ソースのトランザクション・モデルと分散コミット・プロトコル』
- 29 ページの『データ・ソースからのラージ・オブジェクト・サポート』

データ・ソースからのラージ・オブジェクト・サポート

ラージ・オブジェクト (LOB) データ・タイプは、0 バイトから約 2 ギガバイトまでの範囲のサイズを持つバイトのシーケンスとして定義されています。 (LOB データ・タイプには、CLOB、BLOB、および DBCLOB データ・タイプが含まれます。) データ・ソースが LOB をサポートする場合は、ラッパーもそれらをサポートするように設計およびインプリメントする必要があります。

関連概念:

- 27 ページの『データ・ソースのクライアント/サーバー通信』
- 28 ページの『データ・ソースのトランザクション・モデルと分散コミット・プロトコル』
- 28 ページの『データ・ソースからのユーザー認証』

関連資料:

- 106 ページの『リモート照会クラス』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RemoteQuery クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Remote_Query クラス (C++)』

第 4 章 データ・ソースのフェデレーテッド構成へのマッピング

以下のセクションでは、リレーショナル・モデルにデータ・ソースをマップする方法について説明します。

それぞれのフェデレーテッド概念ごとに、データ・ソースに有効なオプションを決定する必要があります。

以下のオプションがあります。

- ラッパー固有の情報をフェデレーテッド・サーバーのシステム・カタログに保管するための汎用 (属性、値の) ペア。
- 持続する構築ブロック (ラッパー、サーバー、ニックネーム、ユーザー) に指定します。

  ニックネーム内では、各列がオプションを持つこともできます。
- ラッパー書き込み機能によって定義および導入されます。
- ほとんどのケースで、フェデレーテッド・サーバーによってアンインタープリットされます。

各ラッパーは、自身のオプションのセットを定義します。

ニックネームのための設計

データを提供するためにソースが使用するデータ・モデルが何であるかにかかわらず、それをリレーショナル・モデルにマップする必要があります。リレーショナル・モデルは、ユーザー、データベース管理者、またはアプリケーションによって (ご使用のデータ・ソースに固有の CREATE NICKNAME DDL ステートメントを通して)、フェデレーテッド・サーバーに提供されます。

ニックネームおよび列オプションの決定

以下の例は、ニックネームと列のオプションを定義する方法を示しています。

ファイル・サーバー用のラッパーについて考慮します。ラッパーは、照会フラグメントをサーバーにサブミットする前に、フラグメント中でニックネームが参照されるファイルのパスと名前を判別します。ラッパーが判別できるように、ニックネームを定義した同じステートメントにパスと名前のオプションを組み込みます。

たとえば、EMPINFO という名前のファイルが、ディレクトリー構造 root/user/ 中にあるとします。 CREATE NICKNAME ステートメントに、PATH という新しいオプションを作成できます。ユーザーは、パス `root/user/EMPINFO` を PATH オプションに割り当てます。

関連概念:

- 32 ページの『照会可能なソース・データの集合のニックネームへのマッピング』
- 32 ページの『階層データ構造のニックネームへのマッピング』

照会可能なソース・データの集合のニックネームへのマッピング

ユーザーは、CREATE NICKNAME ステートメントを実行して、外部サーバー・データの集合 (表やビューなど) を登録します。このステートメントは、フェデレーテッド・サーバー表の定義と同じ特性の集合の定義を組み込みます。外部サーバー・データの集合を登録すると、フェデレーテッド・サーバーはその集合をスキーマの中でフェデレーテッド・サーバー表として定義します。このステートメント中で指定されているニックネームは、表の名前として使用されます。列名やデータ・タイプなどの、定義の他の部分を指定する前に、それらの部分に対応する集合の属性を判別しなければなりません。「列に対応する属性は何か ? データ・ソース属性のタイプに最も近似に対応するフェデレーテッド・サーバーのデータ・タイプは何か ?」このような質問に答えると、属性をフェデレーテッド・サーバー表内で対応するものとして定義できます。

Oracle データベースに EMPLOYEES という名前の表があるとします。この表を登録する前に、フェデレーテッド・サーバー表の列に対応する部分と、フェデレーテッド・サーバー表の列のデータ・タイプに対応する部分を判別する必要があります。この作業は簡単です。Oracle とフェデレーテッド・サーバーは両方ともリレーショナル・データ・モデルに従っているので、Oracle 表の列はフェデレーテッド・サーバー表の列に対応しており、Oracle 列のデータ・タイプはフェデレーテッド・サーバーのデータ・タイプに対応していることは明白だからです。

しかし、データ・ソースがファイル・サーバーの場合はどうでしょうか ? この場合の質問は、「サーバー中のファイルはどのようにフェデレーテッド・サーバー表に対応しているか ?」ということです。 1 つの方法として、個々のファイルを表に対して対応するものと見なし、個々のファイル・レコードを表の行に対して対応するものと見なし、個々のレコードのフィールドを列に対して対応するものと見なすことができます。

関連概念:

- 32 ページの『階層データ構造のニックネームへのマッピング』
- 31 ページの『ニックネームおよび列オプションの決定』

階層データ構造のニックネームへのマッピング

多くのデータ・ソースには、階層編成のデータの集合が含まれます。たとえば、課を表すエンティティー中にプロジェクトの集合が含まれ、この課は部門中の課の集合の 1 つであるなどです。しかし、リレーショナル・モデルはフラットです。ネストされている表に列を使用することはできません。代わりに、エンティティーの種類ごとに別の表を定義し、これらの表を結合して、これらの表の間の階層関係をナビゲートしなければなりません。たとえば、課が属する部門を検出したり、課の中のすべてのプロジェクトを検出したりします。階層構造のデータを使用しているソースを DB2® UDB フェデレーテッド・システムに取り込む際には、個々の階層の集合を、結合可能な関連のあるニックネームとしてモデル化してください。

関連概念:

- 32 ページの『照会可能なソース・データの集合のニックネームへのマッピング』
- 31 ページの『ニックネームおよび列オプションの決定』

データ・ソースから DB2 Universal Database へのデータ・タイプのマッピング

ニックネームの列には、INTEGER、VARCHAR()、DATE などの基本的な SQL タイプが必要です。

DB2® Information Integrator はニックネームの列として、ユーザー定義タイプ、特殊タイプ、および参照タイプをサポートしていません。この種の値を含む列は、代わりに基礎的な表記タイプを使用して定義しなければなりません。しかし、それらの構成を利用するニックネームが表すデータを使ってビューを定義することができます。

外部ソースでサポートされるほとんどの数値タイプの場合、対応する SQL タイプへのマッピングは直接的に行えます。

しかし、数値を表す SQL タイプとして最善のものを判別する際には、精度、桁数、許容値の範囲などの属性を覚えておいてください。データ・ソースの値の表記を DB2 UDB のその値の表記に変換したり、その逆の変換を行ったりするのは、ラッパーです。使用するタイプの種類が多いほど、作成する必要のある変換コードの数が多くなります。

文字データ・タイプのデータ・ソース属性の場合、ラッパーの作成者は、固定長か可変長のどちらの SQL タイプを使用して属性を表すかを選択し、長さの適切な上限を決めなければなりません。 文字データ・タイプに関するその他の考慮事項として、文字のエンコード方式とコード・ページの問題があります。一般的には、文字データとフェデレーテッド・データベースの指定済みのコード・ページとの間の変換は、ラッパーが行います。

時を表すデータの管理の場合、時の値を表すデータ・タイプを、SQL タイプの TIME、DATE、または TIMESTAMP のいずれかにマップしてください。

DB2 UDB は、多数の SQL タイプ間でキャストしたり変換したりする演算子を備えています。データ・ソース属性を VARCHAR などの基本タイプで表し、この値を FLOAT や DATE などのより意味のある適切なタイプにキャスト/変換するビューを定義して、ラッパーを単純化することを考慮してください。

関連概念:

- 32 ページの『照会可能なソース・データの集合のニックネームへのマッピング』
- 32 ページの『階層データ構造のニックネームへのマッピング』

関連タスク:

- 76 ページの『ラッパーの部品のクラスへのマッピング』

関数テンプレートによるデータ・ソース機能のモデル化

DB2® UDB 中にない機能をデータ・ソースに設けることができます。この場合、SQL を拡張して、データ・ソースの特殊機能をモデル化できます。

たとえば、データ・ソースが地理情報システムで、特定の地域を判別できるとします。 SQL には「Area」演算子はないので、この演算子を関数としてモデル化し、以下のように照会を作成します。

```
SELECT id, name
FROM Regions
WHERE Area(id) > 200
```

リモート・データ・ソースによって実行される関数のことを、*カスタム関数* といいます。マップ関数は、ユーザー定義関数 (UDF) とまったく同じ方法で照会中で使用されます。マップ関数と UDF の違いは、UDF は DB2 UDB で実行され、カスタム関数は外部データ・ソースで実行される点です。

カスタム関数の定義は、1 つのステップから成るプロセスです。

1. DDL を使用して以下の関数テンプレートを作成します。 CREATE FUNCTION AREA(VARCHAR()) RETURNS INTEGER AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
   - ご使用のカスタム関数を同じ名前の他のマップ関数、UDF、または DB2 UDB 組み込み関数と区別するには、マッピングされた関数に別のスキーマを定義することを考慮してください。

関連概念:

- 34 ページの『疑似列を使用したデータ・ソース機能のモデル化』

関連タスク:

- 13 ページの『フェデレーテッド照会のデフォルト・コスト・モデル』

疑似列を使用したデータ・ソース機能のモデル化

リレーショナル・インターフェースを介してデータ・ソース機能を公開するもう 1 つの手法として、疑似列を使用することがあります。疑似列は、ニックネームの一部として定義され、照会の特定の局面を制御するために使用できます。疑似列は、普通はエンド・ユーザーではなくラッパーによって定義されます。

一例として、テキスト検索パターン用のワイルドカード文字を制御する `wildcard` 疑似列を作成できます。

別の例として、Blast ラッパーのニックネームの `MismatchPenalty` 疑似列があります。この疑似列は、Blast 検索パラメーターの 1 つを制御しますが、この列自体は値を作成しません。ラッパーが Blast 検索を呼び出す際に、`MismatchPenalty = 5` という書式の述部は、コマンド行オプション `-q5` になります。

疑似列に関する以下の問題を覚えておく必要があります。

- ユーザーが選択リスト中の疑似列を参照する場合、ラッパーはその疑似列が入力専用データであってもその値を戻さなければならない。 NULL を選択することをお勧めします。
- 特定の疑似列に、すべてのリレーショナル演算子が適しているわけではない。多くの場合、1 つの演算子 (「=」、「<」、または「<=」) だけ使用できます。照会の計画中に、ラッパーは無効な演算子を検出し、述部だけをリジェクトするの

ではなく照会全体をリジェクトしなければなりません。述部だけをリジェクトすると、フェデレーテッド・サーバーがその述部自体を処理しようとして、予期しない結果になる可能性があります。

- デフォルトの動作を示す疑似列は、マテリアライズ照会表 (MQT) を妨げる可能性がある。オプティマイザーは、デフォルトの動作を認識しないため、MQT を使用すべきでないときに使用したり、MQT を使用すべきときに使用しなかったりすることがあります。疑似列を含むニックネームに対する照会は、MQT 定義の中で使用される場合、すべての述部を指定している必要があります。

関連概念:

- 33 ページの『関数テンプレートによるデータ・ソース機能のモデル化』

関連タスク:

- 13 ページの『フェデレーテッド照会のデフォルト・コスト・モデル』

ラッパーのための設計

ラッパーによるオプションの処理

ラッパーは、以下の複数の方法でオプション情報に関する作業を行います。

- ラッパーは、登録時にサブミットされる SQL ステートメント中に指定されているオプション情報を妥当性検査する。
- ラッパーは、照会を処理する必要が生じるまで、フェデレーテッド・サーバーのシステム・カタログという保管場所を使用して、構成情報をオプション値の形式で保管する。
- この種の情報を使用してタスクを実行できる。たとえば、データ・ソースからユーザーへエラー・メッセージを経路指定する際の前提条件として、この種のメッセージをすべて経路指定するのか、それとも重大エラーだけ経路指定するのかをラッパーが認識する必要があるとします。この場合、ユーザーがこれらの選択内容を示す値を割り当てられるオプションを作成できます。

上記のアクションを行うようラッパーを準備する際には、以下の作業を行う必要があります。

- ラッパーが新しいオプションの値として保管する必要のある情報があるかどうかを決める。
- SQL ステートメントからフェデレーテッド・サーバーのシステム・カタログに提供されるオプション情報を、ラッパーが妥当性検査する方法を決める。

先頭が DB2_ のオプション名は定義できません。これらの名前は IBM® によって予約されています。

ラッパーが必要とするオプション情報をフェデレーテッド・サーバー・カタログに保管する

ラッパーのオプションを作成する際には、そのオプションの値を入手する方法を決める必要があります。以下の 3 つから選択します。

- 登録で使用する SQL ステートメントの 1 つにオプションと値を指定する。

- ラッパー中にオプションと値をハードコーディングし、ステートメント中で指定されている情報の補足としてフェデレーテッド・サーバーのシステム・カタログに渡す。
- データ・ソース自体から特定のオプション値を入手する。この方法は通常はラッパー・オプションに適用されませんが、他の場合に適用できることがあります。たとえば、サーバーに総称 API があり、この API から保守レベルに関する情報を入手できる場合があります。ラッパーがこの情報をオプション値として記録すると、ラッパーはこの情報を使用して、問題が生じることが知られている特定の照会構成を避けることができます。データ・ソースからニックネーム・オプションや列オプションを入手することもできます。

ユーザーが SQL ステートメントを使用してオプション値を指定できるようにすることをお勧めします。ユーザーが値を指定しない場合は、ハードコーディングされた値か、デフォルトとしてデータ・ソースから入手した値を使用してください。ユーザーが値を変更することがまったくない場合は、オプションとして定義せずに、ラッパー中でハードコーディングする必要があります。

関連概念:

- 36 ページの『ラッパー・オプションの決定』

関連タスク:

- 36 ページの『データ・ソース用の CREATE WRAPPER ステートメントの定義』

ラッパー・オプションの決定

通常、ラッパー・オブジェクトにはオプションは必要ありません。このレベルのオプションは、ラッパー中に定義されているすべてのサーバーに影響を与えます。 1 つのオプションやオプションの集合によって、ラッパーによって作成されるデバッグと診断の出力を制御できます。

関連概念:

- 35 ページの『ラッパーによるオプションの処理』

関連タスク:

- 36 ページの『データ・ソース用の CREATE WRAPPER ステートメントの定義』

データ・ソース用の CREATE WRAPPER ステートメントの定義

ラッパー概念のマッピングとは、ラッパーとデータ・ソース・クライアントのライブラリーを初期化するのにシステムに必要なものを知ることです。

普通ラッパーには構成オプションは必要ありません。しかし、ラッパーで構成オプションを使用できる場合は、構成オプションを定義できます。構成オプションが必要になることはまれですが、データ・ソース・クライアント・ライブラリーを初期化するのにグローバル・パラメーターが必要な場合には、構成オプションが必要になることが予想されます。たとえば、グローバル National Language Support (NLS) ロケールやグローバル・ラッパー・デバッグ・レベルを指定する必要が生じることがあります。

Java でデータ・ソース用の CREATE WRAPPER DDL ステートメントを定義するには、少なくとも unfenced ラッパー・クラスの名前が分かっている必要があります。

たとえば、ラッパー・クラスの名前が my.package.MyUnfencedWrapper の場合は、対応する CREATE WRAPPER ステートメントは以下のようになります。

```
CREATE WRAPPER MyWrapper LIBRARY 'db2qgjava.dll' options
                                                 (UNFENCED_WRAPPER_CLASS
                                                  'my.package.MyUnfencedWrapper')
```

関連概念:

- 36 ページの『ラッパー・オプションの決定』
- 35 ページの『ラッパーによるオプションの処理』

サーバーのための設計

サーバー・オプションの決定

以下の例は、どのサーバー・オプションを定義するかを決定する方法を示しています。

スプレッドシートを含むデータ・ソース用のラッパーについて考慮します。ラッパーが TCP/IP を介してこれらのデータ・ソースと通信するとします。特定のデータ・ソースに接続するには、ラッパーがホスト名とこのデータ・ソースの TCP/IP ポートの番号を認識している必要があります。したがって、データ・ソースのサーバー名を定義している SQL ステートメントに、ホスト名とポート番号のオプションを組み込みます。

たとえば、データ・ソース A のホスト名が `Peter` で、ポート `40` を使用できるとします。 CREATE SERVER ステートメントに、HOSTNAME および PORT という 2 つの新しいオプションを作成できます。ユーザーはホスト名 `Peter` を HOSTNAME オプションに割り当て、番号 `40` を PORT オプションに割り当てます。

関連概念:

- 35 ページの『ラッパーによるオプションの処理』

関連タスク:

- 37 ページの『データ・ソース用の CREATE SERVER ステートメントの定義』

データ・ソース用の CREATE SERVER ステートメントの定義

サーバーのマッピングとは、データ・ソースと一致する CREATE SERVER ステートメントを定義することです。

サーバー概念にマップするデータ・ソース構成を判別する必要があります。たとえば、文書管理システムについて考慮します。 1 つの企業で、複数の独立した文書アーカイブがさまざまな場所にあり、個々のアーカイブに複数の文書の集合が含まれる場合があります。アプリケーションがアーカイブに含まれる集合を検索するに

は、特定のアーカイブに接続しなければなりません。このデータ・ソースの場合、個々のアーカイブをフェデレーテッド・システム中のサーバーとしてモデル化できます。

次に、ラッパーがサーバーに接続して使用できるように、ラッパーを構成するのに必要な構成情報を判別する必要があります。 CREATE SERVER ステートメント上のオプションが必要になります。たとえば、文書アーカイブの場合、NODE オプションを、アーカイブのあるコンピューターの名前にすることができます。

CREATE SERVER ステートメントには、他に 2 つの使用可能データ項目があります。それらのデータ項目とは、サーバー・タイプとバージョンです。多くの共通動作があるが、いくらかの微小なバリエーションもあるデータ・ソースの複数の集合がある場合は、サーバー・タイプとバージョンを使用してラッパーの操作を制御することができます。このようにすれば、それぞれのバリエーションおよびバージョンごとに別個のラッパーを持つ必要はなくなります。

たとえば、CREATE SERVER ステートメントをデータ・ソースに関する以前の情報に関連付けると、以下のようになります。

```
CREATE SERVER Server1 TYPE TYPEA WRAPPER MyWrapper OPTIONS(NODE 'Mname_Orion')
```

関連概念:

- 38 ページの『ユーザー・マッピング・オプションの決定』

関連タスク:

- 36 ページの『データ・ソース用の CREATE WRAPPER ステートメントの定義』
- 39 ページの『データ・ソース用の CREATE USER MAPPING ステートメントの定義』

ユーザー・マッピングのための設計

ユーザー・マッピング・オプションの決定

データ・ソースに REMOTE_AUTHID および REMOTE_PASSWORD 以外のユーザー・マッピング・オプションが必要になることがまれにあります。

フェデレーテッド・サーバーは、セキュリティーを向上させるために、REMOTE_PASSWORD という名前のユーザー・マッピング・オプションの値をエンコードしてからフェデレーテッド・サーバー・システムのカタログに保管します。フェデレーテッド・サーバーはこの値をデコードしてからラッパーに戻します。フェデレーテッド・サーバーが安全に保管する必要のある認証情報に、このオプション名を使用することを強くお勧めします。

デフォルト・バージョンのユーザー・サブクラスがこれらのオプションをサポートしていない場合は、自分でこれらのオプションをインプリメントしなければなりません。

Windows® 上のデータ・ソースの場合、ネットワーク接続と認証には Windows ドメインが必要です。これはサポートする必要のあるオプションの 1 つです。たとえば、ドメイン情報を記録するには、ラッパーは REMOTE_DOMAIN という名前のオプションを定義できます。

関連概念:

- 36 ページの『ラッパー・オプションの決定』
- 37 ページの『サーバー・オプションの決定』
- 31 ページの『ニックネームおよび列オプションの決定』

関連タスク:

- 39 ページの『データ・ソース用の CREATE USER MAPPING ステートメントの定義』

データ・ソース用の CREATE USER MAPPING ステートメントの定義

ユーザーのマッピングとは、データ・ソース用に CREATE USER MAPPING ステートメントを定義することです。

たとえば、ファイル・アーカイブ用のラッパーを作成する場合に、フェデレーテッド・ユーザーをアーカイブ・ユーザー当たり 1 人ずつにしたいとします。この場合、以下の構成情報を定義したいと思います。

- REMOTE_AUTHID: ユーザーのユーザー ID
- REMOTE_PASSWORD: ユーザーのパスワード

その結果、CREATE USER MAPPING ステートメントは、次のようになります。

```
CREATE USER MAPPING FOR simon SERVER myserver
OPTIONS (REMOTE_AUTHID 'joy',
   REMOTE_PASSWORD 'open1up')
```

関連概念:

- 38 ページの『ユーザー・マッピング・オプションの決定』

関連タスク:

- 36 ページの『データ・ソース用の CREATE WRAPPER ステートメントの定義』
- 37 ページの『データ・ソース用の CREATE SERVER ステートメントの定義』

第 5 章 データ・ソースが受け入れることのできる SQL 構成の判別

ご使用のデータ・ソースが、マップされた関数を含め、どの SQL 構成を評価できるか考慮する必要があります。以下の質問をご検討ください。

- ご使用のデータ・ソースがどんな種類のヘッド式を受け入れることができるか。ヘッド式は、SELECT リストにある式か。
- ご使用のデータ・ソースがどのような種類の述部を受け入れることができるか。
- ご使用のデータ・ソースがどのような種類の結合を受け入れることができるか。
- ご使用のデータ・ソースがどのような関数を受け入れることができるか。

以下のセクションでは、これらの考慮事項について詳しく説明します。

データ・ソースが受け入れ可能なヘッド式の判別

ラッパーは、フェデレーテッド・サーバーが提供する照会フラグメントの SELECT リスト中の個々の式を参照します。この式のことを、ヘッド式ともいいます。データ・ソースがこの式を処理できる場合は、ラッパーは応答中にヘッド式を組み込んでそのことを示します。データ・ソースがヘッド式全体を処理できない場合は、ラッパーは応答からヘッド式を除外してヘッド式をリジェクトします。

1 つの制約事項として、DB2 Information Integrator v8.2 はヘッド式の列参照しかプッシュダウンできません。

関連タスク:

- 41 ページの『データ・ソースが受け入れ可能な述部の判別』
- 42 ページの『データ・ソースが受け入れ可能な結合の判別』
- 42 ページの『データ・ソースが受け入れ可能な関数の判別』

データ・ソースが受け入れ可能な述部の判別

ラッパーは、フェデレーテッド・サーバーが提供する照会フラグメントの個々の述部式を参照します。データ・ソースがこの式を処理できる場合は、ラッパーは応答中に述部式を組み込んでそのことを示します。データ・ソースが述部式全体を処理できない場合は、ラッパーは応答から述部式を除外して述部式をリジェクトします。

受け入れる述部のデータ・ソースの意味構造は、フェデレーテッド・サーバーの意味構造と一致している必要があります。ラッパーは、フェデレーテッド・サーバーが特定の述部をプッシュダウンするかどうかを制御できません。述部は、リモート・データ・ソースかフェデレーテッド・サーバーのいずれかで処理することができます。 2 つの述部の間で意味構造が異なる場合は、述部がプッシュダウンされるかどうかによって、最終結果が異なる可能性があります。

フェデレーテッド・サーバーは、照会フラグメントをラッパーに引き渡す前に、接合正規形として知られている形式に述部を書き直します。述部は、AND 演算子で接続された述部式のリストに配置し直されます。

たとえば、以下のようになります。

```
WHERE expr1 AND expr2 AND expr3 ...
```

expr1、expr2、および expr3 は、NOT、AND、および OR 演算子を含む複雑な述部式にすることができます。ラッパーは、トップレベルで式の受諾やリジェクトを行わなければなりません。たとえば、1 つ目の述部 expr1 が col1 = 0 OR col2 > 3 で、データ・ソースが col1 = 0 を評価して col2 > 3 を評価しない場合、ラッパーは expr1 述部全体をリジェクトします。

関連タスク:

- 41 ページの『データ・ソースが受け入れ可能なヘッド式の判別』
- 42 ページの『データ・ソースが受け入れ可能な結合の判別』
- 42 ページの『データ・ソースが受け入れ可能な関数の判別』

データ・ソースが受け入れ可能な結合の判別

結合要求には、複数のニックネームと、それらのニックネームを関連付ける 1 つ以上の述部が組み込まれます。ラッパーは、結合要求を検査し、必要な特定の結合をデータ・ソースがサポートできるかどうかを判別します。

関連タスク:

- 41 ページの『データ・ソースが受け入れ可能なヘッド式の判別』
- 41 ページの『データ・ソースが受け入れ可能な述部の判別』
- 42 ページの『データ・ソースが受け入れ可能な関数の判別』

データ・ソースが受け入れ可能な関数の判別

通常は、述部を定義する式には 1 つ以上の関数呼び出しが含まれます。 DB2 UDB がインプリメントする関数の呼び出しを含む述部をラッパーが受け入れる場合は、データ・ソースが実行する関数の意味構造と、フェデレーテッド・サーバーの関数の意味構造が同一でなければなりません。

通常は、述部を定義する式には 1 つ以上の関数呼び出しが含まれます。述部の関数は、DB2 UDB がインプリメントする関数や演算子 (>、LIKE、+、CONCAT など) か、関数テンプレートを定義する際に登録したラッパーごとに固有のカスタム関数のどちらでもかまいません。 DB2 UDB が評価できる関数の呼び出しを含む述部をラッパーが受け入れる場合は、データ・ソースが実行する関数の意味構造と、DB2 UDB の関数の意味構造が同一でなければなりません。

データ・ソースでサポートしようとしている DB2 UDB 関数の意味構造と、DB2 UDB 自体の関数の意味構造は同一でなければなりません。これには、照合シーケンスや NULL の扱い方の問題も含まれます。

関連タスク:

- 41 ページの『データ・ソースが受け入れ可能なヘッド式の判別』
- 41 ページの『データ・ソースが受け入れ可能な述部の判別』
- 42 ページの『データ・ソースが受け入れ可能な結合の判別』

第 6 章 エラー処理に関する設計

ラッパー作成の準備中に、ラッパーが検出したエラーを報告する方法を決める必要があります。

DB2 UDB サポート関数または DB2 UDB によってインプリメントされたメンバー関数からゼロ以外の戻りコードを受け取る 場合、次のようにします。

1. 可能であれば、リカバリーし、戻りコードをリセットします。
2. 可能でない場合は、ローカルに終結処理して、同じコードを呼び出し側に戻します。

「新規の」エラーを見つけた 場合は、以下のようにします。

1. SQL リファレンスで、最も近いコードを見つけます。
2. メッセージ・テキストから、使用できるトークンの数を判別します。最大メッセージ長は、70 - (nTokens - 1) です。
3. エラーを報告します。
4. ローカルに終結処理して、コードを呼び出し側に戻します。

Java API は、Java 例外を使用してエラーを処理し、さらに、特殊クラスを提供して、DB2 エラー・メッセージに変換された例外をスローします。たとえば、次のとおりです。

```
throw new WrapperException(-1822, "FQftc", new String[]
                           { Integer.toString(remoteCode),
                           serverName, remoteMsg} );
```

表 9 には、エラーのカテゴリーをリストし、それに該当するコードとメッセージに関する情報を記載しています。

表 9. エラーのカテゴリーと関連エラー・メッセージ

イベント	メッセージ番号	メッセージ内容
認証障害	SQL1403	提供されたユーザー名またはパスワードが正しくありませんでした。これは、データ・ソースが認証をサポートしている場合に使用するべきです。
認証、ユーザー・マッピング、欠落	SQL1827	ローカル許可 ID *auth-ID* からサーバー *server-name* へのユーザー・マッピングが定義されていません。このメッセージの説明テキストは、ALTER および DROP ステートメントに特に言及していますが、このメッセージは、ユーザー・マッピングが必要で、それが存在しない場合にも使用されます。
ブランクの切り捨て	SQL1844	リモート・データ・ソースとフェデレーテッド・サーバーの間で、列 *column-name* の末尾ブランクが切り捨てられました。これは、リモート・データ・ソースからサーバーへのデータの転送中に、末尾ブランクが切り捨てられた場合に生成される警告メッセージです。切り捨てがコード・ページの変換によるものである場合は、SQL1580 を使用します。

表 9. エラーのカテゴリーと関連エラー・メッセージ (続き)

イベント	メッセージ番号	メッセージ内容
ブランクの切り捨て、コード・ページ、変換、データ切り捨て	SQL1580	コード・ページ *source-code-page* からコード・ページ *target-code-page* への変換の実行中に、末尾ブランクが切り捨てられました。ターゲット・エリアの最大サイズは、*max-len* でした。ソース・ストリング長は *source-len* で、その16 進数表記は *string* でした。これは、ブランクが切り捨てられるときに使用される警告です (Wrapper_Utilities::convert_codepage は CP_CONV_BUFFER_SMALL を戻し、残りの文字はブランクとなります)。
コード・ページ、変換、データ切り捨て	SQL0334	コード・ページ *source* からコード・ページ *target* への変換の実行中にオーバーフローが発生しました。ターゲット・エリアの最大サイズは、*max-len* でした。ソース・ストリング長は *source-len* で、その16 進数表記は *string* でした。これは、データ変換中に切り捨て/オーバーフローが発生した場合に使用されます。これは、Wrapper_Utilities::convert_codepage が CP_CONV_BUFFER_SMALL を戻す場合に発行されます。
コード・ページ、変換、無効なデータ	SQL0191	フラグメント化された MBCS 文字のためにエラーが発生しました。このメッセージは、Wrapper_Utilities::convert_codepage が CP_CONV_DBCS_TRUNCATE を戻した場合に生成されます。
コード・ページ、変換、サポートされていない	SQL0332	
列、データ・タイプ、ローカル、変更	SQL0270	関数がサポートされていません (理由コード = *reason-code*)。コード 65 は、「現行タイプから指定のタイプへのニックネーム・ローカル・タイプの変更が許可されていない」ことを示します。これは、ALTER NICKNAME ALTER COLUMN LOCAL TYPE ステートメントに対して使用できます。
列、データ・タイプ、ローカル、無効な長さ、精度、桁数	SQL0604	列、特殊タイプ、構造化タイプ、構造化タイプの属性、関数またはタイプ・マッピング *data-item* の長さ、精度、または桁数の属性が無効です。ニックネーム列の指定 (CREATE または ALTER の) の検証の際に、長さ、精度、または桁数のいずれかが無効な場合は、このメッセージを使用します。 *Data-item* は列名になります。
列、データ・タイプ、ローカル、サポートされていない	SQL3324	列 *name* に、サポートされていないタイプ *type* があります。これは、ニックネーム列の無効なローカル・タイプに対して使用できます。
列、演算子、許可されていない	SQL1843	*operator-name* 演算子は、*nickname-name.column-name* ニックネーム列ではサポートされていません。これは、ラッパーによって、ニックネーム列に適用できる演算子が制限されている場合に使用します。

表 9. エラーのカテゴリーと関連エラー・メッセージ (続き)

イベント	メッセージ番号	メッセージ内容
列、リモート、検出されない	SQL0205	列名または属性名 *name* が、*object-name* 内で定義されていません。これは、ローカル列またはリモート列または属性のいずれかを参照するオプションを使用しており、ターゲットが存在しない場合に使用します。これは、SQL0204 よりも詳細です。
列、多すぎる	SQL0680	表、ビュー、または表関数に対して、過度に多くの列が指定されました。これは、データ・ソースがサポートする列の数が DB2 UDB より少ない場合に役立ちます。
通信、失われた接続、TCP/IP、一般エラー	SQL30081	通信エラーが検出されました。使用されている通信プロトコル: *protocol*。 使用されている通信 API: *interface*。 エラーが検出された場所: *location*。エラーを検出した通信関数: *function*。プロトコルに固有のエラー・コード: *rc1*、*rc2*、*rc3*。これは、名前が検出および解決されない gethostbyname() (SQL1336 を参照) 以外の何らかの TCP/IP 通信エラーに対して使用します。これには、接続がドロップされたことを示す何らかの特殊な状態が含まれます。
通信、TCP/IP、ホストが検出されない	SQL1336	リモート・ホスト *hostname* が検出されませんでした。これは、TCP/IP gethostbyname() がホストを戻せない場合に使用します。他の TCP/IP エラーについては、SQL30081 を参照してください。
変換、数値、範囲外	SQL0405	数値リテラル *literal* は、その値が範囲外であるため無効です。このメッセージは、リモート・データ値からローカル・データ値への変換時に使用します。
変換、数値、オーバーフロー	SQL0413	数値データ・タイプの変換中にオーバーフローが発生しました。これは、変換中にオーバーフローが発生した場合に使用します。
データ切り捨て	SQL1844、SQL1845	リモート・データ・ソースとフェデレーテッド・サーバーの間で列 column-name のデータが切り捨てられました。これは、SQL1844 の警告と対応するエラー・メッセージです。切り捨てがコード・ページの変換によるものである場合は、SQL0334 を使用します。
データ・ソース・エラー	SQL1822	データ・ソース *data-source-name* から、予期しないエラー・コード *error-code* を受信しました。関連したテキストとトークンは *tokens* です。メッセージが示すように、これはデータ・ソースによって報告されるエラーです (これに対応する DB2 UDB メッセージはありません)。これは、内部エラーや、ラッパーそのものによって検出されたエラーには使用すべきではありません。
日時、無効な構文	SQL0180	日時値のストリング表記が誤りです。これは、リモート・ソース・ストリングから DB2 Information Integrator の日付、時刻、またはタイム・スタンプ値への変換時に使用します。
日時、範囲外	SQL0181	日時値のストリング表記が範囲外です。このメッセージは、SQL0180 と同じです。

表 9. エラーのカテゴリーと関連エラー・メッセージ (続き)

イベント	メッセージ番号	メッセージ内容
環境変数、欠落	SQL5182	必須の環境変数 *variable-name* が設定されていません。このメッセージは、ある変数が db2dj.ini 内になければならないときに、その変数が存在しない場合に発行されます。
名前、列、未定義	SQL0204	*name* は未定義の名前です。これは、オプション (REMOTE_TABLE など) を使用しているときに、参照されているオブジェクトが存在しない場合に使用します。列または属性については、SQL0205 も参照してください。
名前、列、未定義	SQL0205	列名または属性名 *name* が、*object-name* 内で定義されていません。これは、ローカル列またはリモート列または属性のいずれかを参照するオプションを使用しており、ターゲットが存在しない場合に使用します。
オプション、追加、無効	SQL1840	*option-type* オプション *option-name* を、*object-type* オブジェクトに追加できません。これは、ラッパーが独自に使用するために生成することのあるオプションや、CREATE の実行時にのみ追加できて ALTER の実行時には追加できないオプションに使用します。
オプション、競合	SQL1846	*object-name-1* オブジェクトに対する *option-type-1* オプション *option-name-1* が、*object-name-2* オブジェクトに対する *option-type-2* オプション *option-name-2* と競合しています。これは、2 つ (以上) のオプションまたはオプション値が競合している場合に使用します。これは、オブジェクト・タイプ間の競合 (特定のニックネーム・オプションを指定すると無効になる列オプションなど) の場合もある点にご注意ください。
オプション、ドロップ、無効	SQL1837	*object-name* に対する *option-type* タイプの必須オプション *option-name* をドロップできません。これは、ドロップできないオプションの DROP を試行したときに発行されます。
オプション、重複	SQl1884、SQL1885	*object-name* に対して *option-name* (*option-type* オプション) を複数回指定しました。特定の列オプションなどのいくつかのオプションは、1 回しか指定できません。例として、XML ラッパー・ニックネームに対する PRIMARY_KEY 列オプションは、1 つの列にしか指定できません。このコンテキストでは、*object-name* がニックネームになります。
オプション、無効	SQL1881	*option-name* は、*object-name* に対する有効な *option-type* オプションではありません。無効なオプション。これは、オプションがまったく認識されない場合か、または、オプションが現行のコンテキストで有効でない場合に使用できます。ただし、後者の状況の方が SQL1884 によってよりよく処理される可能性があります。

表 9. エラーのカテゴリーと関連エラー・メッセージ (続き)

イベント	メッセージ番号	メッセージ内容
オプション、無効な値	SQL1882、SQL1842	*object-name* に対する *option-type* オプション *option-name* に、*option-value* を設定できません。無効なオプション値。値が他のオプション値と競合している場合は、代わりに SQL1846N を使用します。値が何らかの他のエンティティー (リモート・オブジェクトや、ローカル・カタログ・オブジェクトなど) の参照である場合は、SQL0204 を使用します。
オプション、欠落	SQL1883	*option-name* は、*object-name* に対する必須指定の *option-type* オプションです。必須指定のオプションが指定されていません。これは、CREATE に対して使用します。 DROP に対しては、SQL1837 を使用します。
オプション、サーバー・オプションの指定、無効	SQL1841	*object-name* オブジェクトに対する *option-type* オプション *option-name* の値を変更できません。これは SQL1840 に似ていますが、SET に対するものです。
オプション、未定義	SQL1886	*object-name* に対して *option-type* オプション *option-name* が指定されていないため、*operation-type* 操作は無効です。定義されていないオプションの SET または DROP が試行されると、DDL プロセスがこれを検出します。
サーバー、タイプ、無効	SQL1816	ラッパー *wrapper-name* を使用して、フェデレーテッド・データベースに定義しようとしているデータ・ソースの *type-or-version* (*server-type server-version*) にアクセスすることができません。これは、CREATE SERVER および ALTER SERVER ステートメントでサーバーのタイプまたはバージョンを検証する際に使用されます。これは、コンポーネント・フレームワークによって自動的に処理できます。
サーバー、バージョン、無効	SQL1817	CREATE SERVER ステートメントは、フェデレーテッド・データベースに定義しようとしているデータ・ソースの *type-or-version* を識別できません。これは、CREATE SERVER ステートメントにタイプまたはバージョンを指定する必要があるが、指定されていない場合に使用されます。これは、コンポーネント・インフラストラクチャーが自動的に処理できます。
SQL、サポートされない	SQL0142	SQL ステートメントがサポートされていません。これには、さらに詳しい情報がないため、ユーザーにとって非常に分かりにくいという問題があります。事前定義理由コードの 1 つ、または理由コードに代わるテキストが付随した SQL30090N のほうが役立ちます。
ユーザー ID、欠落	SQL1027	

関連概念:

- 36 ページの『ラッパー・オプションの決定』
- 37 ページの『サーバー・オプションの決定』
- 31 ページの『ニックネームおよび列オプションの決定』

• 38 ページの『ユーザー・マッピング・オプションの決定』

第 3 部 ラッパーの作成と文書化

本書のこの部では、ラッパーの作成と文書化に必要な次のようなタスクを順に説明します。

- その設計に基づいたラッパーのコーディング
- ユーザーが速やかに使用法を習得できるようにするためのラッパーの文書化

第 7 章 データ・フローの概要

フェデレーテッド照会の処理と、関係するオブジェクト

以下のセクションでは、標準的な照会の流れの詳細な全体像と、照会処理に関係するさまざまなオブジェクトのライフ・サイクルについての注を示します。

標準的なフェデレーテッド照会の流れ

以下のリストには、標準的な照会の流れが概略されています。サブクラス名の XX は、特定のラッパーを表します。

1. クライアント・アプリケーションは、外部データ・ソースにアクセスする必要のある照会をサブミットします。
2. フェデレーテッド・サーバーは、(照会中のニックネームに基づいて) この照会に関係するラッパーを判別します。以下のステップ 3 から 12 は、これらの個々のラッパーに適用されます。
3. フェデレーテッド・サーバーは、unfenced 汎用ラッパー・ライブラリー (C++) またはクラス (Java) をロードし、ブートストラッピング・エントリー・ポイント (フック関数) を呼び出します。ブートストラップ関数は、C++ でのみ使用可能です。
4. フェデレーテッド・サーバーは、ラッパーの unfenced 汎用ラッパー・クラスのカスタマイズ・サブクラスのインスタンスを作成します。具体的には、次のとおりです。

   C++の場合
   フェデレーテッド・サーバーは、ブートストラッピング・エントリー・ポイント (フック関数) を呼び出し、その結果、ラッパーの Unfenced_XX_Wrapper クラスのインスタンスが作成されます。

   Java™ の場合
   フェデレーテッド・サーバーが、UnfencedXXWrapper クラスのコンストラクターを呼び出し、その結果、クラスのインスタンスが作成されます。
5. フェデレーテッド・サーバーは、新しいラッパー・オブジェクト上で初期化メソッドを呼び出します。
6. フェデレーテッド・サーバーは、(ニックネームに基づいて) 照会に関係するラッパーのサーバーを判別します。以下のステップ 7 から 12 は、これらの個々のサーバーに適用されます。
7. フェデレーテッド・サーバーは、ラッパー・オブジェクト上でサーバー作成メソッドを呼び出し、ラッパーのサーバー・サブクラスのインスタンスを作成します。

   C++の場合
   Unfenced_XX_Server (Unfenced_Generic_Server のサブクラス)

Java の場合

UnfencedXXServer (UnfencedGenericServer のサブクラス)

8. フェデレーテッド・サーバーは、新しいサーバー・オブジェクト上で初期化メソッドを呼び出します。
9. フェデレーテッド・サーバーは、サーバー・オブジェクト上の照会プラン・メソッドを介して、照会のフラグメントを含む Request オブジェクトをラッパーにサブミットします。ラッパーのニックネーム・サブクラスのインスタンスがプロセス中に作成されます。
10. ラッパーは照会フラグメントを分析して、1 つ以上の Reply オブジェクトをフェデレーテッド・サーバーに戻します。個々の Reply オブジェクトには、受諾された照会フラグメント、コストの見積もり、および実行記述子が含まれます。フェデレーテッド・サーバー・オプティマイザーによって生成された照会フラグメントごとに、ステップ 9 から 12 を繰り返します。
11. フェデレーテッド・サーバーは、サーバー・オブジェクト上で選択度見積もりメソッドを呼び出します。ラッパーに受諾されなかった照会フラグメントから、すべての述部を含むリストを渡します。
12. ラッパーは、カスタム選択度見積もりメソッドが備えられている場合はこのメソッドを使用し、備えられていない場合はデフォルトのメソッドを使用して、述部の結合された選択度を見積もります。
13. フェデレーテッド・サーバー・オプティマイザーは、照会全体の計画を選択します。
14. unfenced を fenced に置き換えて、ステップ 2 から 8 を繰り返します。つまり、必要な Fenced_XX_Wrapper オブジェクトと Fenced_XX_Server オブジェクト (Java の場合は FencedXXWrapper と FencedXXServer) をすべて作成して初期化します。以下のステップ 15 から 25 は、個々のサーバーに適用されます。
15. フェデレーテッド・サーバーは、サーバー・オブジェクト上でユーザー作成ルーチンを呼び出します。これによって、ラッパーのユーザー・サブクラスのインスタンスが作成され、現在接続しているユーザーが表されます。

    C++の場合

    Fenced_XX_User (Fenced_Generic_User のサブクラス)

    Java の場合

    FencedXXRemoteUser (FencedGenericRemoteUser のサブクラス)
16. フェデレーテッド・サーバーは、新しいユーザー・オブジェクト上で初期化メソッドを呼び出します。
17. フェデレーテッド・サーバーは、サーバー・オブジェクト上で接続作成メソッドを呼び出し、ラッパーのリモート接続サブクラスのインスタンスを作成します。

    C++の場合

    Remote_Connection (XX_Connection)

    Java の場合

    RemoteConnection (XXConnection)
18. フェデレーテッド・サーバーは、接続オブジェクト上で接続メソッドを呼び出します。ラッパーは、(必要に応じて) リモート・ソースに接続します。

19. フェデレーテッド・サーバーは、接続オブジェクト上で照会作成メソッドを呼び出し、ラッパーのリモート照会サブクラスのインスタンスを作成します。

    C++の場合

    Remote_Query (XX_Query)

    Java の場合

    RemoteQuery (XXQuery)

20. フェデレーテッド・サーバーは、照会オブジェクト上で照会開始メソッドを呼び出します。ラッパーは、Reply (照会オブジェクトから入手する) 中の実行記述子を使用して、照会をリモート・ソースにサブミットします。
21. フェデレーテッド・サーバーは、照会オブジェクト上でフェッチ・メソッドを呼び出して、ラッパーからの結果行を要求します。ラッパーは結果行をフェデレーテッド・サーバーに戻します。
22. 行が戻されなくなるまでステップ 21 を繰り返します。
23. フェデレーテッド・サーバーが照会を (おそらくパラメーター値を変更して) 再実行する場合は、フェデレーテッド・サーバーはステップ 20 から 22 を繰り返します。
24. フェデレーテッド・サーバーは、照会オブジェクト上で照会終結処理メソッドを呼び出します。ラッパーは、カーソルかその他の照会関連リソースを終結処理します。
25. フェデレーテッド・サーバーは、照会オブジェクトを破棄します。
26. アプリケーションは作業単位を終了します。
27. フェデレーテッド・サーバーは、接続オブジェクト上でコミット/アボート・メソッドを呼び出します。サーバーごとにこのステップを繰り返します。
28. アプリケーションがデータベースから切断します。
29. フェデレーテッド・サーバーは、接続オブジェクト上で接続終結処理メソッドを呼び出します。フェデレーテッド・サーバーはデータ・ソースとの接続をクローズします。
30. フェデレーテッド・サーバーは、接続オブジェクトを破棄します。サーバーごとにステップ 29 および 30 を繰り返します。
31. インスタンス化された Fenced または Unfenced のラッパー・オブジェクト、サーバー・オブジェクト、ユーザー・オブジェクト、またはニックネーム・オブジェクトがすべて破棄されます。

関連概念:

- 65 ページの『照会の実行のための制御フロー』
- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 63 ページの『照会の計画のための制御フロー』

関連タスク:

- 56 ページの『登録の制御の流れ』
- 62 ページの『初期設定の制御フロー』

関連資料:

- 56 ページの『フェデレーテッド照会に関係するオブジェクトのライフ・サイクル』

フェデレーテッド照会に関係するオブジェクトのライフ・サイクル

- Fenced または Unfenced のラッパー・オブジェクト、サーバー・オブジェクト、ユーザー・オブジェクトまたはニックネーム・オブジェクトは、すでに存在している場合は作成されません。つまり、以前にアプリケーションによってサブミットされた照会を計画するか実行するために、すでに作成されている場合は、作成されません。
- フェデレーテッド・サーバーは、アプリケーションに関連した Fenced または Unfenced のラッパー・オブジェクト、サーバー・オブジェクト、ユーザー・オブジェクト、またはニックネーム・オブジェクトが、進行中の作業単位で使用されていない場合は、それらを破棄します。破棄した場合、アプリケーションが別の照会をサブミットする際に、フェデレーテッド・サーバーは必要に応じてオブジェクトを作成します。アプリケーションがデータベースに接続し続けているのにこの種のオブジェクトが破棄されるのは、通常はアプリケーションが ALTER または DROP DDL ステートメントを発行した場合です。 ALTER または DROP DDL ステートメントは、オブジェクトが表すラッパー、サーバー、ユーザー、またはニックネームのプロパティーを変更します。
- フェデレーテッド・サーバーは、作業単位を完了した時点で、即時にリモート・データ・ソースとの接続を終結処理したり、XX_Connection オブジェクトを破棄したりしません。アプリケーションが同じリモート・ソースにアクセスする別の UOW を実行する場合、接続はしばらくの間、オープンしたままになります。アプリケーションがソースを参照しない作業単位をいくつか完了した後か、またはアプリケーションがデータベースから切断した時点で、フェデレーテッド・サーバーは終結処理を開始します。

関連概念:

- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

プロセスの制御フロー

以下のセクションでは、次のプロセスの制御フローについて説明します。

- 登録
- 初期化
- 照会の計画
- 照会の実行

登録の制御の流れ

この章では、データ・ソースおよび関連した構成を登録する際に、フェデレーテッド・サーバーのシステム・カタログ中に保管される情報をモデル化したクラスについて紹介します。また、登録プロセスの一般的な制御の流れと、ラッパーの登録プロセスを妥当性検査する方法について概説します。

標準情報とオプション情報

フェデレーテッド・サーバーが構成を登録する際には、2 種類の情報の一方または両方がフェデレーテッド・サーバーのシステム・カタログに入れられます。*標準情*

報 は、特定タイプのほとんどの構成に共通な仕様から成ります。たとえば、サーバーを登録する際には、サーバーのユーザー割り当て名と、サーバーへのアクセスに使用するラッパーの名前を、CREATE SERVER ステートメントに指定します。これらの情報部分はすべてのサーバー定義に必要なので、ユーザー割り当て名と関連ラッパーの名前を、サーバーに関する標準情報と見なすことができます。

構成のすべてのインスタンスに関する標準情報をフェデレーテッド・サーバーが保守することに加えて、フェデレーテッド・サーバーのシステム・カタログにはデータ・ソースのタイプによって性質が異なる情報が含まれます。たとえば、IBM は、特定のタイプのファイルに保管されている情報にアクセスするラッパーを提供しています。ファイルにアクセスするには、ラッパーがファイルの名前を認識していなければなりません。ファイル名のような情報は、この種類のデータ・ソースに固有のものです。たとえば、Oracle のソースにアクセスする際には、この情報には意味がありません。フェデレーテッド・サーバーは、オプションと呼ばれる変数に値を割り当てて、カタログ中に情報を保管します。ユーザーは DDL を使用してオプションの名前と値を指定します。したがって、ファイル・ラッパーは、ニックネームのオプション変数 FILE_PATH をサポートしています。CREATE NICKNAME ステートメントを使用してこの変数の値を指定できます。

標準情報とオプション情報のクラス

データ・ソースおよび関連した構成に関するすべての情報 (標準およびオプションの両方) は、複数のクラスによりモデル化されています。これらのクラスのことをまとめて*構成情報クラス* といいます。以下のクラスがあります。

表 10. C++ および Java の構成情報クラス

C++	Java	説明
Wrapper_Info	WrapperInfo	ラッパーに関する情報のモデルです。
Server_Info	ServerInfo	データ・ソースに関する情報のモデルです。
Nickname_Info	NicknameInfo	データ・ソース中のデータの集合 (表やビューなど) に関する情報のモデルです。
Column_Info	ColumnInfo	データ・ソースのデータの集合中の値の列 (または列と同等のもの) に関する情報のモデルです。
User_Info	UserInfo	データ・ソースの使用許可に関する情報のモデルです。

構成情報クラスのオブジェクトの使用法

フェデレーテッド・サーバーとラッパーは、以下の方法で構成情報クラスのオブジェクトを使用します。

- システムが SQL ステートメントおよびラッパーからフェデレーテッド・サーバーのシステム・カタログに転送する情報のコンテナーとして機能します。たとえば、ラッパーは CREATE SERVER ステートメント中の情報を妥当性検査し、フェデレーテッド・サーバーはこの情報と、ラッパーによって備えられた補足情報

をカタログします。システムは、妥当性検査済みの情報と補足情報の両方を、Server_Info オブジェクト (Java の場合は ServerInfo) の、フェデレーテッド・サーバーのシステム・カタログに保管します。

- これらは、フェデレーテッド・サーバーのシステム・カタログから、構成を表すオブジェクト (たとえば特定のデータ・ソースを表す Server オブジェクト) に情報が転送される際の、情報のコンテナーとして機能します。システムが転送を実行すると、受信側のオブジェクトはこの情報を使用して自己初期化します。

個々のラッパーは、データ・ソースに接続するために、フェデレーテッド・サーバーのシステム・カタログから許可情報を入手します。システムはこの情報を、User_Info オブジェクト (Java の場合は UserInfo) でラッパーに渡します。同様に、ラッパーは、照会が参照するニックネームに関するカタログ情報を入手することにより、データ・ソース中の対応するデータ集合を識別できます。システムはこの情報を、Nickname_Info オブジェクト (Java の場合は NicknameInfo) でラッパーに渡します。

CREATE WRAPPER、CREATE SERVER、および CREATE USER MAPPING の流れ

以下のトピックでは、C++ のクラス名とメソッド名を使用しています。 Java でラッパーを作成する場合は、対応する Java メソッド名を代わりに使用してください。

1. アプリケーションは CREATE YY DDL ステートメント (YY = WRAPPER、SERVER、または USER MAPPING) をサブミットします。
2. フェデレーテッド・サーバーは DDL ステートメントを構文解析し、基本構文の正確さをチェックし、同じ名前のエンティティーが既存かをチェックし、エンティティーを作成するラッパーを判別します。
3. フェデレーテッド・サーバーは、このタイプの新しいエンティティーを作成するために必要な、親オブジェクトを作成して初期化します。新規のユーザー・マッピングを作成するには、関係のあるラッパー・オブジェクトおよびサーバー・オブジェクトを最初にインスタンス化しなければなりません。

表 11. C++ および Java のラッパー・オブジェクトとサーバー・オブジェクト

C++	Java
Unfenced_XX_Wrapper	UnfencedXXWrapper
Unfenced_XX_Server	UnfencedXXServer

4. フェデレーテッド・サーバーは、関係のあるラッパーのサーバー・サブクラス、ユーザー・サブクラス、またはラッパー・サブクラスの新しいインスタンスを作成します。フェデレーテッド・サーバーは、親オブジェクト上で該当する作成メソッドを呼び出すことによってこの処理を行います。フェデレーテッド・サーバーは、該当するライブラリーをロードし、フック関数を呼び出すことによって作成された C++ ラッパーの親オブジェクト上では、該当する作成メソッドを呼び出しません。 Java ラッパーの場合、unfenced ラッパー・サブクラスは直接ロードされます。このオブジェクトはクラス Unfenced_XX_YY (XX は指定したラッパーの名前であり、YY は指定した作成されるオブジェクトのタイプの名前) のインスタンスになります。たとえば、以下のようになります。

   C++: Unfenced_File_Wrapper

Java:

UnfencedFileWrapper

5. フェデレーテッド・サーバーは、作成されるエンティティーに対応する情報サブクラスのインスタンスを作成します。このオブジェクトはクラス YY_Info のインスタンスになり、アプリケーションの DDL ステートメント中のすべての情報が含まれます。
6. フェデレーテッド・サーバーは、Unfenced_XX_YY オブジェクト上で creation-validation メソッドを呼び出し、情報オブジェクトを検査するためにラッパーに引き渡します。
7. ラッパーは、YY_Info オブジェクトをチェックして、DDL ステートメント中の情報の整合性と正確さをチェックします。特にラッパーは、オプション値が必要なところに存在し、有効であり、相互に整合しているかをチェックします。
8. オプションでラッパーは別の YY_Info オブジェクト (デルタ情報オブジェクトという) を構成します。ラッパーは、値をオプションに割り当てることもできますし、DDL ステートメントに指定された情報をデルタ情報オブジェクト中の情報でオーバーライドしたり増補したりして、作成されたエンティティーの他の局面を制御することもできます。
9. フェデレーテッド・サーバーは、DDL ステートメント中の情報とデルタ情報オブジェクト (存在する場合) 中の情報を組み合わせ、結果の構成をフェデレーテッド・サーバーのシステム・カタログ中に保管します。
10. フェデレーテッド・サーバーは、Unfenced_XX_YY オブジェクトに対して初期化メソッドを呼び出します。

CREATE NICKNAME の流れ

以下のトピックでは、C++ のクラス名とメソッド名を使用しています。 Java でラッパーを作成する場合は、対応する Java メソッド名を代わりに使用してください。

CREATE NICKNAME の場合の登録の流れには、ニックネーム・クラス上で creation-validation メソッドを使用する妥当性検査ステップが追加されます。 fenced クラスにはデータ・ソースに連絡する機構があるので、ラッパーはデータ・ソースに接続して、DDL で明示的に指定されていない構成情報を収集できます。

1. アプリケーションは CREATE NICKNAME DDL ステートメントをサブミットします。
2. フェデレーテッド・サーバーは該当する Unfenced_XX_Wrapper および Unfenced_XX_Server 親オブジェクトと、新しいニックネームを表す Unfenced_XX_Nickname オブジェクトを作成します。
3. フェデレーテッド・サーバーは Fenced_XX_Wrapper、Fenced_XX_Server、および Fenced_XX_Nickname オブジェクトを作成します。
4. フェデレーテッド・サーバーは、DDL ステートメント上の情報から、Nickname_Info オブジェクトを作成します。
5. フェデレーテッド・サーバーは、Fenced_XX_Nickname オブジェクト上で creation-validation メソッドを呼び出し、Nickname_Info オブジェクトを検査するためにラッパーに引き渡します。

6. オプションで、ラッパーは外部ソースに接続し、データ・ソースから追加情報を入手します。この情報には、スキーマ情報または統計情報、あるいはその両方が含まれます。
7. オプションで、ラッパーはデルタ Nickname_Info オブジェクトを作成します。このオブジェクトには、DDL をオーバーライドする情報 (おそらくデータ・ソースから入手したもの) が含まれています。
8. フェデレーテッド・サーバーは、DDL ステートメント中の情報と、デルタ Nickname_Info オブジェクト (存在する場合) 中の情報を組み合わせます。
9. Fenced_XX_Nickname 妥当性検査メソッドによって作成され、マージされた Nickname_Info は、Unfenced_XX_Nickname 妥当性検査メソッドに対する入力になります。残りのステップはラッパー、サーバー、およびユーザー・マッピング登録の流れのステップ 6 ～ 10 と同じです。

ALTER WRAPPER、ALTER SERVER、ALTER NICKNAME、ALTER USER MAPPING の流れ

以下のトピックでは、C++ のクラス名とメソッド名を使用しています。 Java でラッパーを作成する場合は、対応する Java メソッド名を代わりに使用してください。

ALTER の流れは、CREATE の流れとは違って、ラッパーは DDL で指定された新しいオプション値が未変更のままの現行値と整合しているか、および相互に整合しているか判別しなければなりません。

1. アプリケーションは、前述のエンティティー YY (YY はラッパー、サーバー、またはユーザー・マッピングのいずれか) の DDL ステートメントの 1 つをサブミットします。
2. フェデレーテッド・サーバーは DDL ステートメントを構文解析し、基本構文の正確さをチェックし、指定された名前のエンティティーが存在することを確認し、どのラッパーがエンティティーを変更するかを判別します。
3. フェデレーテッド・サーバーは、変更されるエンティティーを表すオブジェクトをインスタンス化するために必要な親オブジェクトを作成して初期化します。 Unfenced_XX_User または Unfenced_XX_Nickname をインスタンス化するには、関係のある Unfenced_XX_Wrapper および Unfenced_XX_Server オブジェクトを最初にインスタンス化しなければなりません。フェデレーテッド・サーバーは、オブジェクトが既存の場合は作成しません。
4. フェデレーテッド・サーバーは、ユーザーが変更を加えたいエンティティーに応じて、関係のあるラッパー・オブジェクト Unfenced_Generic_Wrapper、Unfenced_Generic_Server、または Unfenced_Generic_User サブクラスの新しいインスタンスを作成します。フェデレーテッド・サーバーは親オブジェクト上で該当する作成メソッドを呼び出してこの処理を行います。ただし、該当するライブラリーをロードし、フック関数を呼び出してラッパーが作成された場合は例外です。このオブジェクトはクラス Unfenced_XX_YY (XX はラッパー、YY は Unfenced_File_Wrapper や Unfenced_Blast_Server などの作成されるオブジェクトのタイプ) のインスタンスになります。フェデレーテッド・サーバーは、オブジェクトが既存の場合は作成しません。Java では、ライブラリーとフック関数はありません。その代わり、フェデレーテッド・サーバーが適切なラッパー・サブクラスをインスタンス化します。

5. フェデレーテッド・サーバーは、新しく作成された Unfenced_XX_YY オブジェクトを初期化します。
6. フェデレーテッド・サーバーは、ユーザーが変更を加えたいエンティティーに対応する Catalog_Info サブクラスのインスタンスを作成します。このオブジェクトはクラス YY_Info (Server_Info や Remote_User_Info など) のインスタンスになり、アプリケーションの DDL ステートメント中のすべての情報が含まれます。
7. フェデレーテッド・サーバーは、Unfenced_XX_YY オブジェクト上で alter-validation メソッドを呼び出し、YY_Info オブジェクトを検査するためにラッパーに引き渡します。
8. ラッパーは、YY_Info オブジェクトをチェックして、DDL ステートメント中の情報の整合性と正確さをチェックします。特にラッパーは、新しいオプション値が未変更のままの既存オプション値と整合しているか、また、必須オプションがドロップしていないか確認します。
9. オプションでラッパーは別の YY_Info オブジェクト (デルタ情報オブジェクトという) を構成します。ラッパーは、値をオプションに割り当てることもできますし、DDL ステートメントに指定された情報をデルタ情報オブジェクト中の情報でオーバーライドしたり増補したりして、変更されたエンティティーの他の局面を制御することもできます。
10. フェデレーテッド・サーバーは、DDL ステートメント中の情報とデルタ情報オブジェクト (存在する場合) 中の情報を組み合わせ、結果の構成をフェデレーテッド・サーバーのシステム・カタログ中に保管します。
11. フェデレーテッド・サーバーは Unfenced_XX_YY オブジェクトを破棄します。このオブジェクトは、この DDL ステートメントを反映していない古いカタログ情報で初期化されています。フェデレーテッド・サーバーは、次回このオブジェクトが必要になる時点で再作成し、更新された情報で初期化します。

登録プロセスの妥当性検査に関する一般的なステップ

ラッパーの登録プロセスの妥当性検査は、以下の一般的なステップに従います。

CREATE DDL

以下のトピックでは、C++ のクラス名とメソッド名を使用しています。 Java でラッパーを作成する場合は、対応する Java メソッド名を代わりに使用してください。

CREATE DDL の妥当性検査は、ラッパーの開発者が特定のオプションを定義した場合に限り必要になります。登録の妥当性検査に関する一般的な手順を以下に示します。ラッパー作成ブロックごとに固有の妥当性検査の手順については、これ以降のセクションを参照してください。

1. XX_Info オブジェクト中の Catalog_Option ごとに、以下のようにします。
   a. is_reserved_xx_option() を呼び出すことによって、これが予約済みの DB2 UDB オプションかどうかを判別します (Java API は、同様の機能を持つ CatalogOption クラス上のメソッド isReserved() を提供しています)。予約済みの場合は、次のステップにスキップします。
   b. ラッパーでサポートされているオプションかどうか判別します。サポートされていない場合は、SQL1881N エラー・メッセージを発行します。

c. このオプションのオプション値が有効かどうか判別します。有効でない場合は、SQL1882N エラー・メッセージを発行します。

2. ユーザーが必須のオプションを省略した場合は、SQL1883N エラー・メッセージを発行します。
3. 指定されているすべてのオプション値が相互に整合しているか判別します (該当する場合)。 SQL1882N エラー・メッセージを使用して、無効値や不整合値を報告します。
4. 必要な場合、追加のオプションを用意します。
   a. デルタ XX_Info オブジェクトが割り振られているか判断し、必要に応じて割り振ります。
   b. 必要に応じて、XX_Info の add_option メソッドを使用して、新しいオプションを追加します。

ALTER DDL

この妥当性検査は、CREATE と同様に、ラッパーの開発者がラッパー固有のオプションを定義した場合に限り必要です。 ALTER の処理は、2 つの重要な点で CREATE の処理とは違います。1 つ目は、妥当性検査ルーチンがラッパーの現行の状態を考慮に入れなければならない点です。 XX_Info オブジェクトで提供される情報は、新しい情報か変更情報のみです。 2 つ目は、妥当性検査ルーチンがオプションごとにアクション・パラメーターに注意を払わなければならない点です。最重要な点として、ユーザーが必須オプションの DROP を試行した場合に、妥当性検査ルーチンが SQL1881N エラー・メッセージを発行しなければなりません。さらに、オプション値の DROP を実行すると、アドレッシング・エラーによるエンジン破損などの問題が生じるかどうか、チェックを試行します。

関連概念:

- 65 ページの『照会の実行のための制御フロー』
- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 63 ページの『照会の計画のための制御フロー』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

関連タスク:

- 45 ページの『第 6 章 エラー処理に関する設計』
- 62 ページの『初期設定の制御フロー』
- 「*フェデレーテッド・システム・ガイド*」の『ラッパーの変更』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER WRAPPER ステートメント』

初期設定の制御フロー

以下は、初期設定プロセスの制御フローの要約です。

この要約では、C++ のオブジェクト名を使用しています。 Java でラッパーを作成する場合は、対応する Java オブジェクト名を代わりに使用してください。

この流れは、Fenced_Generic または Unfenced_Generic 基本クラスのサブクラスのインスタンスであるすべてのオブジェクト (たとえば、Unfenced_Blast_Nickname、Fenced_File_User など) に当てはまる、ごく一般的なものです。

1. フェデレーテッド・サーバーは、その親オブジェクト上の適切な作成メソッド (あるいは、ラッパーの場合はフック関数) を使用して、ラッパーで定義されたサブクラス「X」(たとえば Fenced_Excel_Server) のインスタンスを作成します。
2. フェデレーテッド・サーバーは、YY_Info (たとえば Server_Info) オブジェクト (適切なフェデレーテッド・サーバーのシステム・カタログに格納されている、そのオブジェクトが表すエンティティーに関する既知のすべての情報を含む) を作成します。すべての Catalog_Info オブジェクトには、対応するエンティティーに対して定義されているオプションの名前と値を指定する、Catalog_Option オブジェクトのリストが含まれます。 Nickname_Info オブジェクトには、Column_Info オブジェクトが、ニックネームの各列に 1 つずつ入っています。
3. フェデレーテッド・サーバーはステップ 1 で作成した「X」オブジェクト上で初期設定メソッドを呼び出し、ステップ 2 で作成した YY_Info オブジェクトを渡します。
4. アクセスの効率を上げるため、ラッパーはオプションで YY_Info オブジェクトから情報 (たとえばオプション値) を抽出し、それを「X」オブジェクトのメンバー変数に格納します。フェデレーテッド・サーバーは対応する YY_Info オブジェクトをコピーして「X」オブジェクトに添付します。 Nickname オブジェクト (fenced または unfenced) の場合、フェデレーテッド・サーバーは Nickname_Info オブジェクトはコピーしません。

関連概念:

- 65 ページの『照会の実行のための制御フロー』
- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 63 ページの『照会の計画のための制御フロー』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

関連タスク:

- 56 ページの『登録の制御の流れ』

照会の計画のための制御フロー

以下は、照会の計画プロセスの制御フローの要約です。

この要約では、C++ のオブジェクト名を使用しています。 Java™ でラッパーを作成する場合は、対応する Java オブジェクト名を代わりに使用してください。

このフロー要約では、フェデレーテッド・サーバーがラッパーに、1 つ以上のヘッド式と述部を持つ単一表の照会フラグメントを計画するように求めていると想定しています。フェデレーテッド・サーバーは、まず単一表のフラグメントを計画し、その次にソースの 2 つのニックネームを結合するフラグメントを計画し、その後で三者間結合およびその他を計画します。複雑なフラグメントの場合は、Request の中に追加の数量詞ハンドルが含まれ、Request 中の述部ハンドルのいくつかは結合述部を表します。

1. フェデレーテッド・サーバーが、照会フラグメントについて記述する Request オブジェクトを作成し、適切な Unfenced_XX_Server オブジェクト上で計画メソッドを呼び出します。
2. ラッパーが、この要求に対する Reply オブジェクトを作成します。ラッパーがカスタム・コスト・モデルをサポートしている場合、これはラッパーで定義されている Reply サブクラスのインスタンスになります。ラッパーがデフォルトのコスト・モデルを使用している場合、ラッパーは基本の Reply クラスをインスタンス化します。
3. ラッパーが、照会フラグメントの範囲を示すニックネームを表す数量詞のハンドルを取得します (Java API はハンドルではなく、インスタンスを使用します)。
4. ラッパーが、ハンドルを使用して、フェデレーテッド・サーバーに対して、照会フラグメントで見つかるニックネームに応じたその Unfenced_Generic_Nickname サブクラスのインスタンスである Unfenced_XX_Nickname を要求します。
5. フェデレーテッド・サーバーが Unfenced_XX_Nickname オブジェクトを作成し、初期設定します。
6. 必要に応じて、ラッパーがニックネームの列や構成などに関する情報を、Unfenced_XX_Nickname オブジェクトから取得します。
7. ラッパーが数量詞のハンドルを Reply に追加します。
8. ラッパーが照会フラグメントの中の 1 つのヘッド式のハンドルを取得します。
9. ラッパーがハンドルを使用して、ヘッド式について説明する Request_Exp オブジェクトを取得します。
10. ラッパーが、Request_Exp で表されたヘッド式の値をデータ・ソースが計算できるかどうかを判別します。ラッパーは Request_Exp ツリーの末端に向かって、ステップ 12 を再帰的に繰り返して決定を下します。
11. データ・ソースがヘッド式全体を計算できたら、ラッパーがそのヘッド式のハンドルを Reply オブジェクトに追加します。
12. 要求内の追加のヘッド式ごとにステップ 9 ～ 12 を繰り返します。
13. ラッパーが照会フラグメントの中の 1 つの述部のハンドルを取得します。
14. ラッパーがハンドルを使用して、述部について説明する Request_Exp オブジェクトを取得します。
15. ラッパーが、Request_Exp によって表される述部全体をデータ・ソースが処理できるかどうかを判別します。ラッパーは、Request_Exp ツリーの末端に向かって再帰的にこの決定を下す必要があります。
16. データ・ソースが述部を計算できたら、ラッパーがその述部のハンドルを Reply オブジェクトに追加します。
17. 要求の中の追加の述部ごとにステップ 15 ～ 17 を繰り返します。
18. データ・ソースが、DB2® Information Integrator の照合との互換性のある特定の順序で行を戻す場合、ラッパーは Reply に順序記述を追加します。
19. ラッパーが Reply によって表された照会のサブフラグメントを検査し、ラッパーの実行記述子に格納する永続表記を生成します。
20. ラッパーが実行記述子を Reply に格納します。

21. ラッパーがこの照会フラグメント用の代替計画を生成する場合は、代替プランごとにステップ 9 から 23 を繰り返します。
22. ラッパーが Reply オブジェクトのリストをフェデレーテッド・サーバーに戻します。
23. フェデレーテッド・サーバーが Reply オブジェクト上でコスト・メソッドを呼び出し、対応するラッパー・プランのコストとカーディナリティーを計算します。
24. ラッパーがカスタム・コスト・モデルを使用している場合、そのラッパーのコスト・メソッドは、オプションで Unfenced_XX_Nickname オブジェクトから取得されるカスタム統計を参照しながら、コストとカーディナリティーを計算します。それ以外の場合は、デフォルトのインプリメンテーションにより、デフォルトのコスト・モデルを使用してこれらの値が計算されます。
25. 追加の Reply オブジェクトがあれば、追加の Reply オブジェクトごとにステップ 26 から 27 を繰り返します。

関連概念:

- 65 ページの『照会の実行のための制御フロー』
- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

関連タスク:

- 56 ページの『登録の制御の流れ』
- 62 ページの『初期設定の制御フロー』

照会の実行のための制御フロー

以下は、照会の実行プロセスの要約です。

この照会は、C++ のオブジェクト名を使用しています。 Java™ でラッパーを作成する場合は、対応する Java オブジェクト名を代わりに使用してください。

1. フェデレーテッド・サーバーが、ラッパーの Remote_Query サブクラス、XX_Query のインスタンス上で照会初期設定メソッドを呼び出します。
2. ラッパーが XX_Query オブジェクトから、ラッパー実行記述子を取得します。ラッパーが、記述子の情報を使用して、リモート・ソースでの照会フラグメントの実行の準備をします。
3. ラッパーが、XX_Query オブジェクトから出力 Runtime_Data_List を取得し、データ・ソースから戻されるデータとラッパーがフェデレーテッド・サーバーに戻すデータの間のマッピングを判別します。
4. 照会フラグメントにパラメーターが含まれている場合、ラッパーは XX_Query オブジェクトから取得した入力 Runtime_Data_List の対応するエレメントからパラメーター値を取得します。
5. ラッパーが、パラメーター値 (存在する場合) を、SQL (DB2 UDB) 表記からデータ・ソースで期待される表記へと変換します。
6. ラッパーが、リモート・ソースでの照会フラグメントの実行を開始します。
7. フェデレーテッド・サーバーが XX_Query オブジェクト上で取り出しメソッドを呼び出します。

8. ラッパーが、少なくとも 1 行分のデータをデータ・ソースから検索します。ラッパーは追加の行はバッファーに入れます。
9. ラッパーが、データ・ソースで規定されている表記から、フェデレーテッド・サーバーで期待される SQL 表記へと値を変換します。ラッパーは、出力 Runtime_Data_List の適切なエレメント上でメソッドを呼び出すことによって、変換した各値を DB2® バッファーにコピーします。
10. データ・ソースが戻す行がなくなるまで、ステップ 7 ～ 9 を繰り返します。
11. フェデレーテッド・サーバーが照会終結処理メソッド XX_Query を呼び出します。状況フラグで、DB2 が同じアプリケーション照会の一部として照会フラグメントを再実行するかどうかが示されます。
12. 照会が再実行される場合は、フェデレーテッド・サーバーが XX_Query で照会再実行メソッドを呼び出します。フェデレーテッド・サーバーによって再実行が要求されなくなるまで、ステップ 4 ～ 11 が繰り返されます。

関連概念:

- 66 ページの『ラッパーと外部サーバーとの間の通信』
- 63 ページの『照会の計画のための制御フロー』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

関連タスク:

- 56 ページの『登録の制御の流れ』
- 62 ページの『初期設定の制御フロー』

ラッパーと外部サーバーとの間の通信

ラッパーが照会処理に参加する方法のいくつかは、照会フラグメントを、データ・ソースが理解する「要求」に変換し、それらの副照会をデータ・ソースにサブミットして、データ・ソースが戻す結果の行を検索するという方法です。 unfenced 汎用サーバー・サブクラスのメソッドは、これらのタスクを計画しています。リモート照会サブクラスのメソッドが実行を行います。メソッド間のタスクの分散方法は、ユーザーが自分の裁量でかなりの程度決定できます。これについて、以下で例を使って説明します。

例 1 次の例は、ファイル・サーバー用のラッパーが、列を参照する、述部を含まない照会フラグメントの中の SELECT ステートメントの処理を開始するところから始まります。

1. 照会の計画時に、オプティマイザーが照会フラグメントを検査して、そのフラグメントに基づく要求をラッパーに送信します。ラッパーが要求を検査し、応答と共に、ラッパーがデータ・ソースにできることとできないことをオプティマイザーに通知するラッパー・プランを戻します。オプティマイザーが、ラッパー・プランを含んだ、実行する照会の計画を作成します。
2. 照会の実行中、オプティマイザーがラッパーにラッパー・プランを送信します。
3. ラッパーがデータ・ソースに対して計画を実行します。

4. ラッパーが、ラッパー実行記述子で指定されたオプションの値を基にして、ニックネームに対応するデータ・セットを含んだデータ・ソース・ファイルの名前とロケーションを判別します。
5. ラッパーがラッパー・プランを検査して、検索する列の名前を判別します。
6. ラッパーが、ラッパー実行記述子で指定されたオプションの値を基にして、ファイル内のどの序数列が、照会の中の指定された各列に対応するかを判別します。
7. ラッパーが、ラッパー実行記述子で指定されたオプションの値を基にして、照会の中で指定された各列の終端区切り文字を判別します。
8. ラッパーがファイルを開いて、データの読み取りを開始します。ラッパーはそれまでに判別した情報を使用して、フェデレーテッド・サーバーに戻す行を解析してフォーマットします。

例 2 次の例は、文書サーバー用のラッパーが、照会フラグメントの中の SELECT ステートメントの処理を開始するところから始まります。このフラグメントは、文書のリポジトリー、*属性* と呼ばれる構造、およびテキスト検索関数という 3 種類のデータ・ソース構成を参照します。フェデレーテッド・サーバーのシステム・カタログにおいて、フェデレーテッド・サーバーはリポジトリーを DB2® UDB 表として定義し、DB2 UDB は属性をこれらの表の列として定義し、関数は関数テンプレートにマップしています。

1. 照会の計画時に、オプティマイザーが照会フラグメントを検査して、そのフラグメントに基づく要求をラッパーに送信します。ラッパーが要求を検査し、応答と共に、ラッパーがデータ・ソースにできることとできないことをオプティマイザーに通知するラッパー・プランを戻します。オプティマイザーが、ラッパー・プランを含んだ、実行する照会の計画を作成します。
2. 照会の実行中、オプティマイザーがラッパーにラッパー・プランを送信します。
3. ラッパーがデータ・ソースに対して計画を実行します。
4. ラッパーが、ラッパー実行記述子で指定されたオプションの値を基にして、文書のどのリポジトリーがニックネームに対応するかを判別します。
5. ラッパーがラッパー・プランを検査して、検索する列の名前を判別します。
6. ラッパーが、ラッパー実行記述子で指定されたオプションの値を基にして、照会の中で指定されている各列に対応する属性を判別します。
7. ラッパーが、ラッパー実行記述子の中のテキスト検索関数を識別します。
8. ラッパーが、これまでに判別した情報 (たとえば、文書のリポジトリーの名前、属性の名前、およびテキスト検索述部) を使用して、文書サーバーの照会言語でステートメントを生成します。
9. ラッパーが文書サーバーの API に照会をサブミットして、フェデレーテッド・サーバーに戻すデータの読み取りを開始します。

関連概念:

- 65 ページの『照会の実行のための制御フロー』
- 63 ページの『照会の計画のための制御フロー』
- 53 ページの『標準的なフェデレーテッド照会の流れ』

関連タスク:

- 56 ページの『登録の制御の流れ』
- 62 ページの『初期設定の制御フロー』

関連資料:

- 56 ページの『フェデレーテッド照会に関係するオブジェクトのライフ・サイクル』

第 8 章 ラッパー・クラスを使用した作成

以下のセクションでは、ラッパーの作成方法と、ラッパーを作成する際に使用する必要のあるクラスについて説明します。

ラッパーを開発するための一般的な手順

このトピックでは、ラッパーを作成するための標準的な手順について説明します。ラッパーの作成を始める前に、まず以下を実行します。

- 登録に使用する SQL ステートメント、サブクラスに必要な基本クラス、データ・ソースの API に精通します。
- データ・ソースのデータの集合をリレーショナル表としてモデル化する方法を決定します。
- DB2® UDB カタログ中で、データ・ソースのデータの集合に関するメタデータを表す方法を決定します。
- 必要に応じて、その他の計画アクティビティーを実行します。

ラッパーを作成する場合は、通常は以下のステップを実行します。

1. 以下のラッパー・サブクラスをインプリメントし、テストします。

   C++の場合
   Unfenced_Generic_Wrapper および Fenced_Generic_Wrapper

   Java™ の場合
   UnfencedGenericWrapper および FencedGenericWrapper

   CREATE WRAPPER ステートメントを実行して、インプリメンテーションをテストします。 SYSCAT.WRAPPERS および SYSCAT.WRAPOPTIONS カタログ・ビュー中に作成される結果項目を検査します。
2. 以下のサーバー・サブクラスをインプリメントし、テストします。

   C++の場合
   Unfenced_Generic_Server および Fenced_Generic_Server

   Java の場合
   UnfencedGenericServer および FencedGenericServer

   CREATE SERVER ステートメントを実行してインプリメンテーションをテストし、SYSCAT.SERVERS および SYSCAT.SERVEROPTIONS カタログ・ビュー中に作成される結果項目を検査します。
3. (オプション) ラッパーが許可を必要とし、ニックネーム検証の際にデータ・ソースにアクセスする場合は、ユーザー・クラスをインプリメントしてテストします。

   C++の場合
   Unfenced_Generic_User および Fenced_Generic_User

Java の場合

UnfencedGenericRemoteUser および FencedGenericRemoteUser

CREATE USER MAPPING ステートメントを実行してインプリメンテーションをテストし、SYSCAT.USEROPTIONS カタログ・ビュー中に作成される結果項目を検査します。

4. (オプション) ラッパーがニックネーム検証の際にデータ・ソースにアクセスする場合は、リモート接続サブクラスをインプリメントします。

   C++の場合

   Remote_Connection

   Java の場合

   RemoteConnection

5. データ・ソースで passthru をサポートしたい場合は、リモート passthru クラスをインプリメントします。

   C++の場合

   Remote_Passthru

   Java の場合

   RemotePassthru

   データ・ソース中のデータの集合を参照する passthru 要求を実行して、リモート passthru サブクラスとリモート接続クラスの両方のインプリメンテーションをテストします。

6. ニックネーム・サブクラスをインプリメントします。

   C++の場合

   Unfenced_Generic_Nickname および Fenced_Generic_Nickname

   Java の場合

   UnfencedGeneric Nickname および FencedGeneric Nickname

   複数の外部サーバー・データの集合に CREATE NICKNAME ステートメントを実行し、SYSCAT.COLOPTIONS カタログ・ビュー、SYSTAT.TABLES カタログ表、および SYSTAT.COLUMNS カタログ表中の結果項目を検査して、インプリメンテーションをテストします。DB2 UDB に同等の関数のないデータ・ソース関数をユーザーが使用できるようにしたい場合は、この関数に対応する関数テンプレートを作成してください。

7. (オプション) ラッパーが許可を必要とし、ニックネーム検証の際にデータ・ソースに接続しない場合は、ユーザー・クラスをインプリメントしてテストします。

   C++の場合

   Unfenced_Generic_User および Fenced_Generic_User

   Java の場合

   UnfencedGenericRemoteUser および FencedGenericRemoteUser

   CREATE USER MAPPING ステートメントを実行してインプリメンテーションをテストし、SYSCAT.USEROPTIONS カタログ・ビュー中に作成される結果項目を検査します。

8. ステップ 4 を実行しなかった場合は、リモート接続サブクラスをインプリメントします。

   C++の場合
   Remote_Connection

   Java の場合
   RemoteConnection

9. リモート照会サブクラスをインプリメントします。

   C++の場合
   Remote_Query

   Java の場合
   RemoteQuery

   このクラスとリモート接続クラスの両方のインプリメンテーションをテストするには、データ・ソース中のデータの集合を参照するフェデレーテッド照会をサブミットします。

10. ラッパーを文書化します。

関連概念:

- 72 ページの『ラッパーの作成のためのヒント』

関連タスク:

- 73 ページの『トラステッドおよび fenced モードのプロセス環境』
- 80 ページの『ラッパー・クラス』
- 83 ページの『サーバー・クラス』
- 91 ページの『ユーザー・クラス』

サブクラスおよびメソッドのインプリメンテーション

ラッパー・インターフェースに精通するには、以下の点を考慮してください。

- 使用する基本クラス。以下の点に注意してください。
  - 表 12 は、独自のサブクラスのインプリメンテーションを作成する必要がある基本クラスを示しています。

表 12. サブクラスのインプリメンテーションが必要な基本クラス

C++	Java™
Unfenced_Generic_Wrapper	UnfencedGenericWrapper
Fenced_Generic_Wrapper	FencedGenericWrapper
Unfenced_Generic_Server	UnfencedGenericServer
Fenced_Generic_Server	FencedGenericServer
Unfenced_Generic_Nickname	UnfencedGenericNickname
Fenced_Generic_Nickname	FencedGenericNickname
Unfenced_Generic_User	UnfencedGenericRemoteUser
Fenced_Generic_User	FencedGenericRemoteUser
Remote_Connection	RemoteConnection

表 12. サブクラスのインプリメンテーションが必要な基本クラス (続き)

C++	Java™
Remote_Query	RemoteQuery

  - 独自のコスト・モデルをインプリメントする場合は、以下のサブクラスが必要になる。
    - Reply
  - ラッパーがパススルーをサポートする場合は、以下のサブクラスが必要になる。
    - Remote_Passthru (Java の場合は RemotePassthru)
  - 特定のクラスの IBM® のインプリメンテーションを使用する必要がある。
- 作成するインプリメンテーションに組み込むもの。以下の点に注意してください。
  - 特定のメソッドのデフォルトのインプリメンテーションを、独自のカスタマイズ済みのインプリメンテーションでオーバーライドしなければならない。
  - 特定のメソッドのデフォルトのインプリメンテーションを使用するか、独自のカスタマイズ済みのインプリメンテーションでオーバーライドするか選択しなければならない。
  - 独自の新しいメソッドを追加できる。追加したい場合は、インプリメントする親クラスのメソッドを呼び出す新しいメソッドを作成できることに注意してください。
  - 新しいデータ・メンバーを追加できる。

関連タスク:

- 95 ページの『Reply クラス』
- 112 ページの『リモート passthru クラス』

ラッパーの作成のためのヒント

ラッパーを作成する際には、以下の点に注意してください。

- Wrapper_Utilities クラス (Java™ の場合は WrapperUtilities) には、作業に役立つサービスが備えられています。たとえば、コードに関する問題が生じた場合は、クラスの trace_data メソッドを使用できます。 trace_data メソッドは、問題を見つけて識別するのに役立つレコードを生成します。
- 複数のバージョンのラッパーを作成して、最初のバージョンは基本サービスに限定し、それ以降のバージョンで徐々に機能や効率性を増していくことができます。

関連概念:

- 71 ページの『サブクラスおよびメソッドのインプリメンテーション』
- 69 ページの『ラッパーを開発するための一般的な手順』

トラステッドおよび fenced モードのプロセス環境

複数のサブエージェント (スレッドまたはプロセス) を作成してデータ・ソースに並列アクセスすることにより、照会の実行時に、フェデレーテッド・サーバーのパフォーマンスを向上させることができます。サブエージェント環境で実行するコードをさらに簡単に分離できます。このような分離により、システムの保全性が失われる可能性のある障害を発生しにくくすることができます。そのため、ラッパーのうち照会の実行に関係する部分と、照会の計画に関係する部分が分離されます。

結果として、ラッパーは次の 2 つの部分に分割されます。

- *unfenced* の部分は、照会の計画に関係します。
- *fenced* の部分は、照会の実行に関係します。

したがって、個々のラッパー構築ブロックを、ラッパーの fenced 部分か unfenced 部分のどちらかに割り当てなければなりません。構築ブロックを両者の間で分割しなければならない場合もあります。外部ソースとの通信が必要な場合は、関数は fenced 部分に入れられます。

処理環境は、ラッパーが C++ と Java のどちらで作成されているかによっていくらか異なります。

C++ の処理環境

C++ のラッパーは 2 つの独立したライブラリーとしてリンクされます。その 1 つにはラッパーの unfenced 部分のクラス・インプリメンテーションが含まれ、もう 1 つには fenced 部分のクラス・インプリメンテーションが含まれます。 *fenced* モード というモードで、ラッパーの fenced 部分は、fenced モード・プロセス (FMP) という分離された特殊なサブエージェント中にロードされます。ラッパーの unfenced 部分は、メインのフェデレーテッド・サーバー・エージェントにロードされます。 74 ページの図 7 では、fenced モード・プロセスのモデルを図示しています。

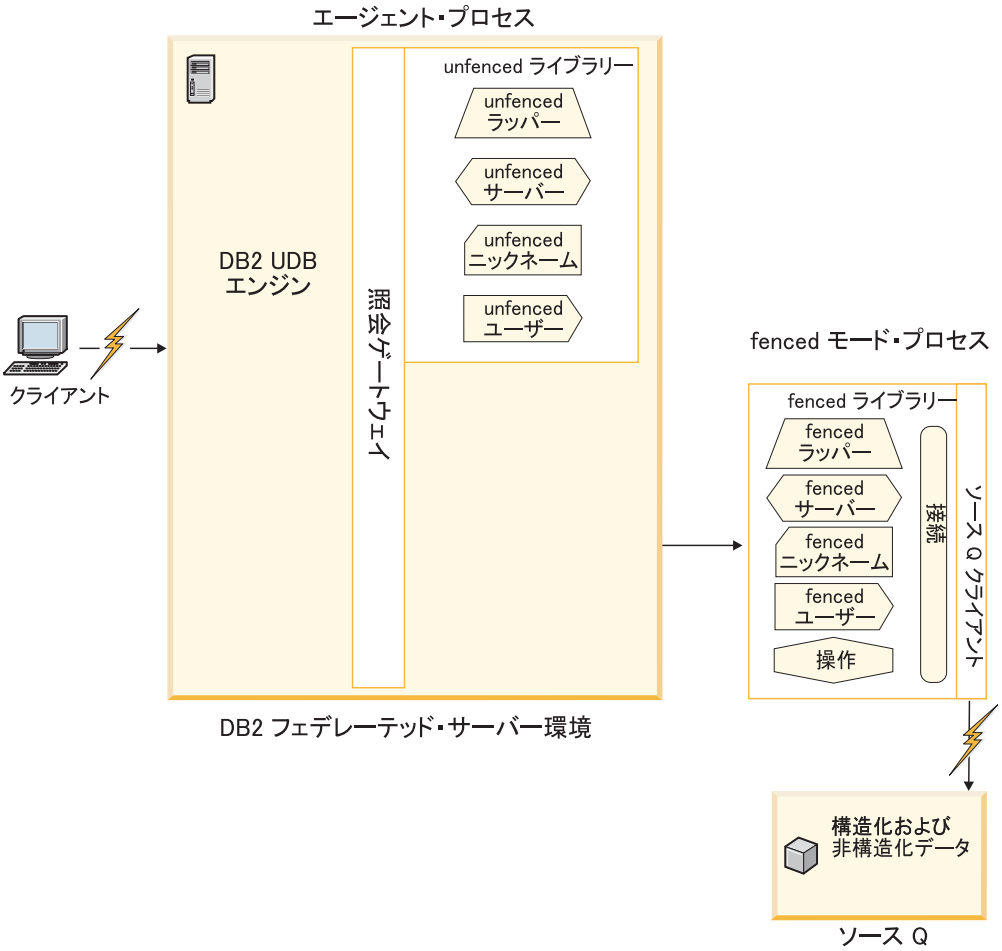

図 7. C++ の fenced モード・プロセスのモデル

図 7 は、ラッパーの fenced の部分と unfenced の部分が含まれた 2 つの別個のライブラリーと、それらがロードされる別個のエリア (フェデレーテッド・サーバー環境の unfenced の部分と fenced モード・プロセスの fenced の部分) を示しています。

並列処理が問題にならない場合は、サブエージェントを使用せず、フェデレーテッド・サーバー環境ですべての関数を実行する方がパフォーマンスは向上します。分離による安全性を犠牲にするので、このモードのことをトラステッド・モード といいます。 75 ページの図 8 では、トラステッド・モード・プロセスのモデルを図示しています。

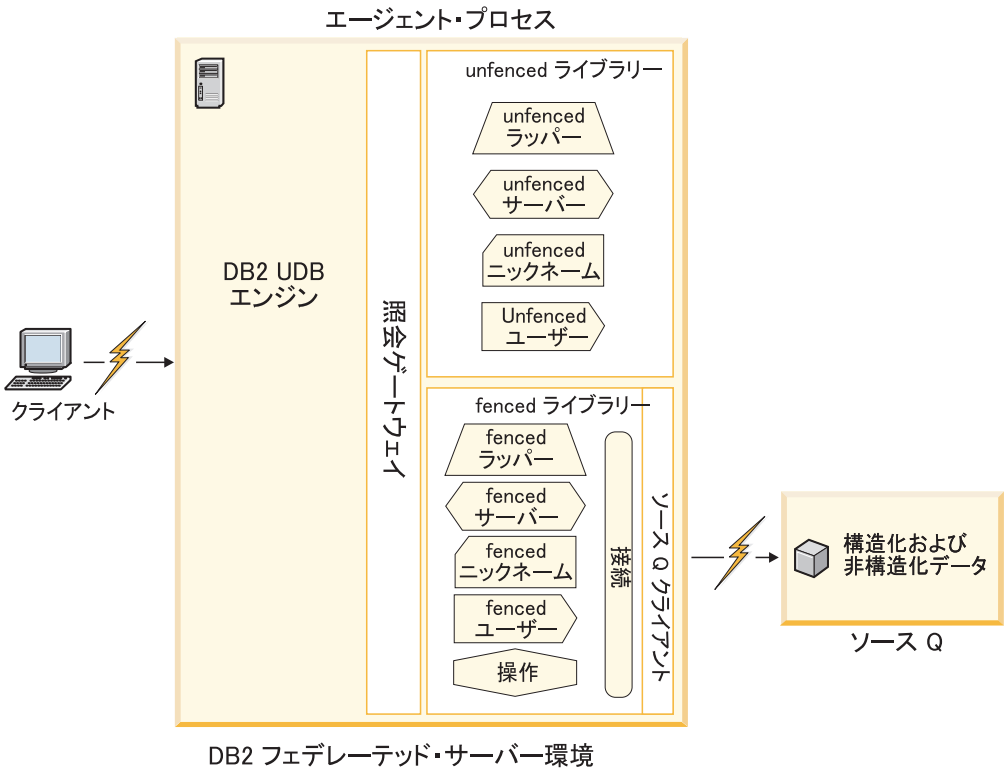

図 8. C++ のトラステッド・モード・プロセスのモデル

図 8 は、トラステッド・モードでは、ラッパーの fenced の部分と unfenced の部分が両方ともフェデレーテッド・サーバー環境内にロードされることを示しています。

DB2 Information Integrator V8.1 ではトラステッド・モードの実行のみサポートしていますが、トラステッド・モードと fenced モードの両方の実行をサポートするようにラッパーを設計しなければなりません。

Java の処理環境

Java で作成されたラッパーは、必ず fenced モード・プロセスのみで実行します。fenced と unfenced の両方の Java クラスが fenced モード・プロセスで実行します。 76 ページの図 9 は、Java ラッパーの fenced モード処理環境を示しています。

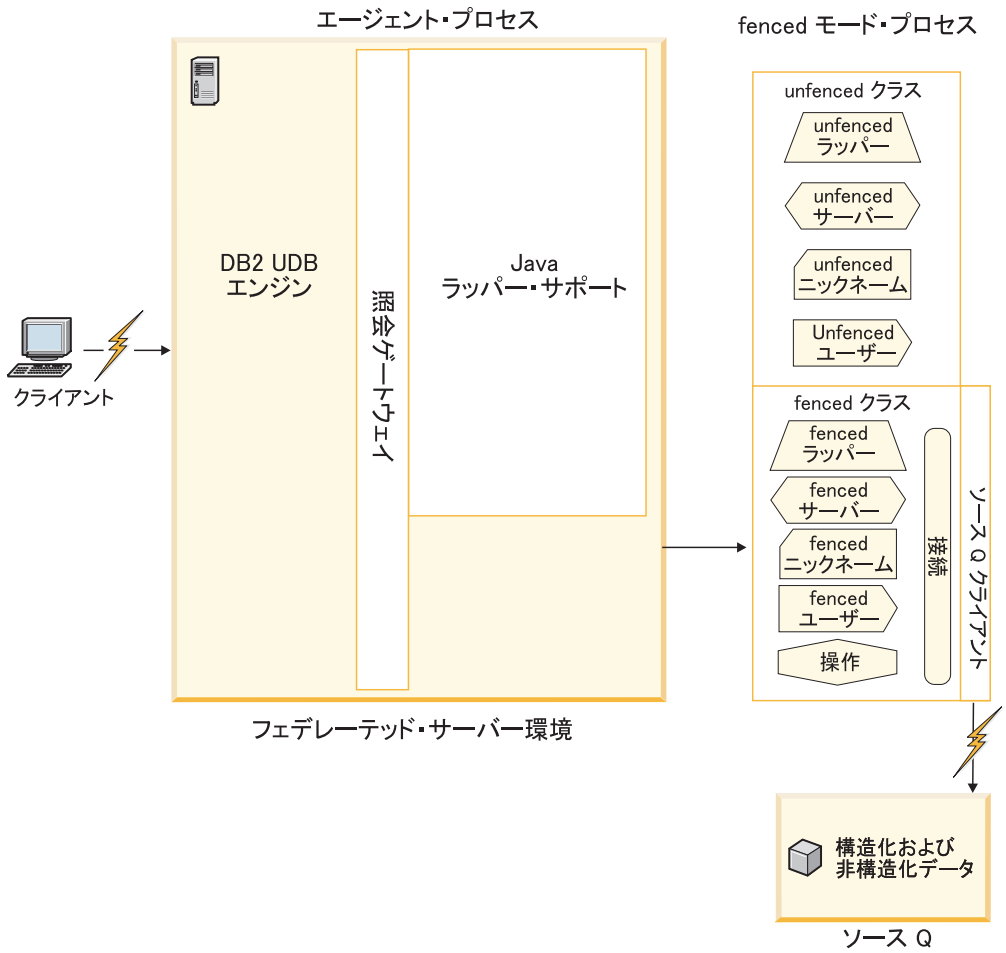

図 9. Java の fenced モード・プロセスのモデル

図 9 は、フェデレーテッド・サーバー環境内ではなく、fenced モード・プロセス内で実行する fenced クラスと unfenced クラスを示しています。

関連概念:

- 79 ページの『ラッパーとデータ・ソース間での通信のためのクラス』
- 71 ページの『サブクラスおよびメソッドのインプリメンテーション』
- 69 ページの『ラッパーを開発するための一般的な手順』
- 72 ページの『ラッパーの作成のためのヒント』

関連タスク:

- 76 ページの『ラッパーの部品のクラスへのマッピング』

ラッパーの部品のクラスへのマッピング

構築ブロックの多くには unfenced と fenced の両面があるので、それぞれのクラスを使用してコーディングしなければなりません。

unfenced 側は、照会の計画に関与します。 fenced 側は、照会の実行段階に関与し、フェデレーテッド・サーバーが fenced モードの場合にデータ・ソースの環境からフェデレーテッド・サーバーを保護します。

表 13. C++ の基本的なラッパー構築ブロックの unfenced クラスと fenced クラス

構築ブロック	C++ unfenced クラス	C++ fenced クラス
ラッパー	Unfenced_Generic_Wrapper	Fenced_Generic_Wrapper
サーバー	Unfenced_Generic_Server	Fenced_Generic_Server
ニックネーム	Unfenced_Generic_Nickname	Fenced_Generic_Nickname
ユーザー	Unfenced_Generic_User	Fenced_Generic_User
接続	なし	Remote_Connection
操作	なし	Remote_Operation

表 14. Java の基本的なラッパー構築ブロックの unfenced クラスと fenced クラス

構築ブロック	Java unfenced クラス	Java fenced クラス
ラッパー	UnfencedGenericWrapper	FencedGenericWrapper
サーバー	UnfencedGenericServer	FencedGenericServer
ニックネーム	UnfencedGenericNickname	FencedGenericNickname
ユーザー	UnfencedGenericRemoteUser	FencedGenericRemoteUser
接続	なし	RemoteConnection
操作	なし	RemoteOperation

関連概念:

- 69 ページの『ラッパーを開発するための一般的な手順』
- 72 ページの『ラッパーの作成のためのヒント』

関連タスク:

- 73 ページの『トラステッドおよび fenced モードのプロセス環境』

第 9 章 ラッパーをコーディングするためのクラス

このセクションでは、ラッパーをコーディングするために使用するクラスについて説明します。

ラッパーとデータ・ソース間での通信のためのクラス

表 15 は、データ・ソースへの接続とデータ・ソースが実行する操作のモデルである基本クラスを示しています。

表 15. データ・ソースへの接続のモデルを構成する基本クラス

C++ の基本クラス	Java™ の基本クラス
Remote_Query	RemoteQuery
Remote_Connection	RemoteConnection
Remote_Operation	RemoteOperation
Runtime_Data	RuntimeData
Runtime_Data_List	RuntimeDataList
Runtime_Data_Desc	RuntimeDataDesc
Runtime_Data_Desc_List	RuntimeDataDescList
Remote_Passthru	RemotePassthru

接続と同様に、データ・ソースの操作は一時的なものです。したがって、フェデレーテッド・サーバーはそれらに関する情報をフェデレーテッド・サーバーのシステム・カタログに保管しません。

関連概念:

- 71 ページの『サブクラスおよびメソッドのインプリメンテーション』
- 69 ページの『ラッパーを開発するための一般的な手順』
- 72 ページの『ラッパーの作成のためのヒント』

関連タスク:

- 73 ページの『トラステッドおよび fenced モードのプロセス環境』
- 76 ページの『ラッパーの部品のクラスへのマッピング』
- 104 ページの『リモート接続クラス』
- 109 ページの『ランタイム・データ・クラス』
- 111 ページの『ランタイム・データ記述クラス』
- 112 ページの『リモート passthru クラス』

関連資料:

- 106 ページの『リモート照会クラス』

ラッパー・クラス

以下のセクションでは、unfenced 汎用ラッパー・クラスおよび fenced 汎用ラッパー・クラスについて説明します。

Unfenced_Generic_Wrapper クラス

C++: ラッパー・コードを含む共有ライブラリーの中の関数 UnfencedWrapper_Hook を呼び出すと、unfenced ラッパー・サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、初めてラッパーの unfenced 共有ライブラリーをロードするときに、この関数を呼び出します。 DDL 操作でラッパーに関係した情報が変更されると、フェデレーテッド・サーバーはそのラッパー・オブジェクトが次に使用される前に、それを破棄して再作成します。

Java: フェデレーテッド・サーバーが unfenced ラッパー・クラスをロードするときに、そのクラスのインスタンスが自動的に作成されます。ラッパー固有の unfenced ラッパー・クラスの名前は、CREATE WRAPPER ステートメントの UNFENCED_WRAPPER_CLASS オプションの値として指定されます。

unfenced 汎用ラッパー基本クラスのインプリメンテーションは、以下の情報を保守します。

- ラッパー名
- ラッパー・タイプ: 非リレーショナルの場合は「N」。
- ラッパー・バージョン: ラッパーがフェデレーテッド・サーバーに登録されたときに実行していたラッパー・コードのバージョンを表す、ラッパー指定のバージョン番号。この値を現在実行しているコードのバージョンと比較すると、互換性を確認することができます。
- ラッパー・モジュール・ライブラリーの最上位ファイル名。
- CREATE WRAPPER または ALTER WRAPPER DDL ステートメントを実行した結果として、フェデレーテッド・サーバーのシステム・カタログに格納された、このラッパーに関連するすべての情報を含むラッパー情報オブジェクト。

メンバー関数だけが、この関数の検査や変更を行えます。

すべてのラッパーに必要なカスタマイズ

unfenced 汎用ラッパー・サブクラスは以下をインプリメントする必要があります。

- コンストラクター。これは対応する unfenced 汎用ラッパー基本クラスのコンストラクターを呼び出します。

  C++: Unfenced_Generic_Wrapper

  Java:
  UnfencedGenericWrapper
- サーバーの作成: unfenced 汎用サーバー・サブクラスのインスタンスを作成するためのメソッド。

  C++: Unfenced_Generic_Server

  Java:
  UnfencedGenericServer

その他のカスタマイズ

- unfenced 汎用ラッパー・サブクラスのインスタンスに追加情報を格納する必要がある場合は、initialize_my_wrapper() 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているラッパー情報オブジェクトからこの情報を抽出します。 C++ では、このラッパー情報オブジェクトへのポインターを保存することはできません。このフォームの情報を参照する場合は、ラッパー情報オブジェクトのコピーを含むデータ・メンバーを使用します。 Java では、ラッパーは WrapperInfo オブジェクトへの参照を保存できますが、コピー機能はありません。
- ラッパー・オプションを定義する場合は、verify_my_register_wrapper_info 関数および verify_my_alter_wrapper_info 関数のデフォルト・インプリメンテーションをオーバーライドして、DDL で指定されているオプションと値が有効であることを検査します。 DDL で指定されているオプション値を変更したり、追加のオプションを指定したりする場合は、インプリメンテーションで「差分」ラッパー情報オブジェクトによってオーバーライド/追加情報が指定されていなければなりません。新しい「差分」オブジェクトを割り振る前に、「差分」オブジェクトがすでに存在していないかどうかチェックしてください。「差分」オブジェクトが存在している場合は、別の「差分」オブジェクトを割り振らないでください。 Java では、「差分」ラッパー情報オブジェクトは実際には、verifyMyRegisterWrapperInfo および verifyMyAlterWrapperInfo メソッドから戻されるオブジェクトです。このオブジェクトは、ラッパーにより作成される必要があります。
- unfenced 汎用ラッパー・サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 16. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトの動作
initialize_my_wrapper	initializeMyWrapper	ノーオペレーション
create_server	createServer	エラー
verify_my_register_wrapper_info	verifyMyRegisterWrapperInfo	非 DB2 Information Integrator オプションが指定されていないことを検査する
verify_my_alter_wrapper_info	verifyMyAlterWrapperInfo	マイ・レジスター・ラッパー情報の検査 (verify my register wrapper information) オブジェクトを呼び出す

表 17. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
is_reserved_wrapper_option	isReserved	DB2 UDB がオプションを処理する場合は、true が戻される。

Fenced_Generic_Wrapper クラス

C++: ラッパー・コードを含む共有ライブラリーの中の関数 FencedWrapper_Hook を呼び出すと、fenced ラッパー・サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、初めてラッパーの fenced 共有ライブラリーをロードす

るときに、この関数を呼び出します。 DDL 操作でラッパーに関係した情報が変更されると、フェデレーテッド・サーバーはそのラッパー・オブジェクトが次に使用される前に、それを破棄して再作成します。

Java: フェデレーテッド・サーバーが fenced ラッパー・クラスをロードするときに、そのクラスのインスタンスが自動的に作成されます。ラッパー固有の fenced ラッパー・クラスの名前は、CREATE WRAPPER ステートメントの FENCED_WRAPPER_CLASS オプションの値として指定されます。

fenced 汎用ラッパー基本クラスのインプリメンテーションは、以下の情報を保守します。

- ラッパー名
- ラッパー・タイプ: 非リレーショナルの場合は「N」。
- ラッパー・バージョン: ラッパーがフェデレーテッド・サーバーに登録されたときに実行していたラッパー・コードのバージョンを表す、ラッパー指定のバージョン番号。このバージョン番号は integer で、float ではありません。この値を現在実行しているコードのバージョンと比較すると、互換性を確認することができます。
- ラッパー・モジュール・ライブラリーのファイル名。
- CREATE WRAPPER または ALTER WRAPPER DDL ステートメントを実行した結果データベース・マネージャー・カタログに格納された、このラッパーに関連する情報を含むラッパー情報オブジェクト。

メンバー関数だけが、この関数の検査や変更を行えます。

すべてのラッパーに必要なカスタマイズ

fenced 汎用ラッパー・サブクラスは以下をインプリメントする必要があります。

- コンストラクター。これは対応する fenced 汎用ラッパー基本クラスのコンストラクターを呼び出します。

  C++: Fenced_Generic_Wrapper

  Java:
  FencedGenericWrapper
- サーバーの作成: fenced 汎用サーバー・サブクラスのインスタンスを作成するためのメソッド。

  C++: Fenced_Generic_Server

  Java:
  FencedGenericServer

その他のカスタマイズ

- fenced 汎用ラッパー・サブクラスのインスタンスに追加情報を格納する必要がある場合は、initialize_my_wrapper 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているラッパー情報オブジェクトからこの情報を抽出します。 C++ では、このラッパー情報オブジェクトへのポインターを保存することはできません。このフォームの情報を参照する場合

は、ラッパー情報オブジェクトのコピーを含むデータ・メンバーを使用します。Java では、ラッパーは WrapperInfo オブジェクトへの参照を保存できますが、コピー機能はありません。

- Unfenced_Generic_Wrapper サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 18. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
initialize_my_wrapper	initializeMyWrapper	はい	ノーオペレーション
create_server	createServer	はい	エラー

関連タスク:

- 83 ページの『サーバー・クラス』
- 88 ページの『ニックネーム・クラス』
- 91 ページの『ユーザー・クラス』
- 「*フェデレーテッド・システム・ガイド*」の『ラッパーの変更』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER WRAPPER ステートメント』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Java API のラッパー・クラス』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Wrapper クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『C++ API のラッパー・クラス』

サーバー・クラス

以下のセクションでは、Unfenced_Generic_Server クラスと fenced 汎用サーバー・クラスについて説明します。

Unfenced_Generic_Server クラス

unfenced ラッパー・サブクラスのインスタンス上でサーバーの作成メソッドを呼び出すと、unfenced サーバー・サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、アプリケーションが関係するサーバーを最初に使用する前に、このメソッドを呼び出します。 DDL 操作でサーバーに関係した情報が変更されると、フェデレーテッド・サーバーはそのサーバー・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、Unfenced_Generic_Server 基本クラスのインプリメンテーションは、以下の情報を保守します。

- サーバー名。
- 適切なラッパー・オブジェクトへのポインター。

- DDL ステートメントを実行した結果フェデレーテッド・サーバーのシステム・カタログに格納された、このサーバーに関連する情報を含むサーバー情報オブジェクト。

この情報の一部には、データ・メンバーとして直接アクセスできます。この情報のその他の部分は、データ・メンバー関数でのみ検査または変更できます。

すべてのラッパーに必要なカスタマイズ

Unfenced_Generic_Server サブクラスは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは、対応する Unfenced_Generic_Server コンストラクターを呼び出します。
- ニックネームの作成: unfenced ニックネーム・サブクラスのインスタンスを作成するためのメソッド。
- 要求のプラン: Request オブジェクトに含まれている照会フラグメントを分析し、1 つ以上の Reply オブジェクト (それぞれが、データ・ソースが実行できる照会フラグメントとそれに対応する実行コストを識別する) を戻すメソッド。

その他のカスタマイズ

- Unfenced_Generic_Server サブクラスのインスタンスに追加情報を格納する必要がある場合は、initialize_my_server メンバー関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているサーバー情報オブジェクトからこの情報を抽出します。 C++ では、このサーバー情報オブジェクトへのポインターを保存することはできません。このフォームの情報を格納する場合は、この情報オブジェクトのコピーを含むデータ・メンバーを使用します。 Java では、サーバーは ServerInfo オブジェクトへの参照を保存できますが、コピー機能はありません。
- Unfenced_Generic_Server クラスにすでに定義されているものに加えて、サーバー・オプション、サーバー・タイプ、およびサーバー・バージョンを定義する場合は、verify_my_register_server_info オブジェクトおよび verify_my_alter_server_info オブジェクトのデフォルト・インプリメンテーションをオーバーライドして、DDL で指定されているオプションと値の有効性を検査します。標準の Unfenced_Generic_Server オプションを検査する必要はありません。これは、Unfenced_Generic_Server インプリメンテーションが行います。 DDL で指定されているオプション値を変更したり、追加のオプションを指定したりする場合は、インプリメンテーションで「差分」サーバー情報オブジェクトによってオーバーライド/追加情報が指定されていなければなりません。 C++ では、新しい「差分」オブジェクトを割り振る前に、「差分」オブジェクトがすでに存在していないかどうかチェックしてください。存在している場合は、別の「差分」オブジェクトを割り振らないでください。 Java では、「差分」ラッパー情報オブジェクトは実際には、verifyMyRegisterWrapperInfo および verifyMyAlterWrapperInfo メソッドから戻されるオブジェクトです。このオブジェクトは、ラッパーにより作成される必要があります。
- デフォルトの Unfenced_Generic_User クラスが、ラッパーに対して十分でない場合は、リモート・ユーザーの作成メソッドのインプリメンテーションをオーバーライドして、Unfenced_Generic_User サブクラスのインスタンスを作成します。
- デフォルト・コスト・モデルが生成する、カーディナリティーまたは実行時間の見積もりが正確でない場合は、応答の作成メソッドのインプリメンテーションを

オーバーライドして、カスタムのコスト・モデルをサポートするようカスタマイズした Reply サブクラスのインスタンスを作成します。

- デフォルト選択度推定子が生成する、データ・ソースからのデータを含む述部の値が正確でない場合は、選択度の取得メソッドのデフォルトのインプリメンテーションを、より正確な結果を生成するメソッドでオーバーライドします。
- Unfenced_Generic_Server サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 19. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
verify_my_register_server_info	verifyMyRegisterServerInfo	はい	(C++ では is_reserved_server_option() 関数、Java では CatalogOption クラス の isReserved() 関数に基づく) 事前定義のリストと比較してオプションを検査します。他のオプションはエラーを生成します。
verify_my_alter_server_info	verifyMyAlterServerInfo	はい	(C++ では is_reserved_server_option() 関数、Java では CatalogOption クラス の isReserved() 関数に基づく) 事前定義のリストと比較してオプションを検査します。他のオプションはエラーを生成します。
initialize_my_server	initializeMyServer	はい	ノーオペレーション
create_nickname	createNickname	はい	エラー
create_remote_user	createRemoteUser	はい	unfenced 汎用ユーザーを作成します
create_reply	createReply	はい	デフォルト・コスト・モデルをサポートする Reply を作成します。
get_selectivity	getSelectivity	はい	デフォルト・モデルの選択度を計算します。
get_type	getType	はい	情報からタイプを取得します。
get_version	getVersion	はい	情報からバージョンを取得します。
plan_request	planRequest	いいえ	適切な応答オブジェクトを戻します

Fenced_Generic_Server クラス

fenced ラッパー・サブクラスのインスタンス上でサーバーの作成メソッドを呼び出すと、fenced サーバー・サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、アプリケーションが関係するサーバーを最初に使用する前に、このメソッドを呼び出します。 DDL 操作でサーバーに関係した情報が変更されると、フェデレーテッド・サーバーはそのサーバー・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、fenced 汎用サーバー基本クラスのインプリメンテーションは、以下の情報を保守します。

- サーバー名。
- 適切なラッパー・オブジェクトへのポインター。
- DDL ステートメントを実行した結果、フェデレーテッド・サーバーのシステム・カタログに格納された、このサーバーに関連する情報を含むサーバー情報オブジェクト。
- このサーバーに対する、インスタンス化されたリモート接続オブジェクトの表 (現行のインプリメンテーションでは、リモート接続の数が 1 つに制限されています)。
- データ・ソースへの接続が存在する場合は、接続されているユーザーのユーザー・マッピングを表す fenced 汎用ユーザー・オブジェクトへのポインター。

この情報の一部には、データ・メンバーとして直接アクセスできます。この情報のその他の部分は、データ・メンバー関数でのみ検査または変更できます。

すべてのラッパーに必要なカスタマイズ

fenced 汎用サーバー・サブクラスは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは、対応する fenced 汎用サーバーのコンストラクターを呼び出します。
- ニックネームの作成: fenced ニックネーム・サブクラスのインスタンスを作成するためのメソッド。

  C++: Fenced_Nickname

  Java:
  FencedNickname
- リモート接続の作成: リモート接続サブクラスのインスタンスを作成するためのメソッド。

  C++: Remote_Connection

  Java:
  RemoteConnection

その他のカスタマイズ

- fenced 汎用サーバー・サブクラスのインスタンスに追加情報を格納する必要がある場合は、initialize_my_server 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているサーバー情報オブジェクトからこの情報を抽出します。 C++ では、このサーバー情報オブジェクトへのポインターを保存することはできません。その代わり、このフォームに情報を格

納する場合は、このオブジェクトのコピーを含むデータ・メンバーを使用します。 Java では、サーバーは ServerInfo オブジェクトへの参照を保存できますが、コピー機能はありません。

- デフォルトの fenced 汎用ユーザー・クラスが、ラッパーに対して十分でない場合は、リモート・ユーザーの作成メソッドのデフォルト・インプリメンテーションをオーバーライドして、fenced 汎用ユーザー・サブクラスのインスタンスを作成します。

  C++: Fenced_Generic_User

  Java:
  FencedGenericRemoteUser

- fenced 汎用サーバー・サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 20. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
initialize_my_server	initializeMyServer	はい	ノーオペレーション
create_remote_user	createRemoteUser	はい	unfenced 汎用ユーザー・クラスを作成します
create_remote_connection	createRemoteConnection	はい	エラー
create_nickname	createNickname	はい	エラー
get_type	getType	はい	情報からタイプを取得します。
get_version	getVersion	はい	情報からバージョンを取得します。

表 21. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
find_current_remote_user	findRemoteUser	現行の作成者 ID に対するリモート・ユーザー・オブジェクトを検索します
find_connection	findConnection	単一接続のリモート・ユーザーに対する接続オブジェクトを取得します
find_nickname	findNickname	ローカル・スキーマ名が与えられている unfenced ニックネーム・オブジェクトを取得します

関連タスク:

- 80 ページの『ラッパー・クラス』
- 88 ページの『ニックネーム・クラス』
- 91 ページの『ユーザー・クラス』

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Java API のサーバー・クラス』

- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『C++ API のサーバー・クラス』

ニックネーム・クラス

以下のセクションでは、unfenced 汎用ニックネーム・クラスおよび fenced 汎用ニックネーム・クラスについて説明します。

Unfenced_Generic_Nickname クラス

fenced サーバー・サブクラスのインスタンス上でニックネームの作成メソッドを呼び出すと、unfenced ニックネーム・サブクラスのインスタンスが作成されます。このメソッドは通常、unfenced 汎用サーバー・メソッドの find_nickname (Java では findNickname) を呼び出した結果として呼び出されます。 DDL 操作でニックネームに関係した情報が変更されると、フェデレーテッド・サーバーはそのニックネーム・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、unfenced 汎用ニックネーム基本クラスのインプリメンテーションは、以下の情報を保守します。

- ニックネームを定義するリモート・データ・セットのローカル名。
- ニックネームを定義するリモート・データ・セットのローカル・スキーマ。
- 該当するサーバー・オブジェクトを指すポインター。
- デフォルト・コスト・モデルが使用する 4 つの統計の見積もり値。
  1. ニックネームのカーディナリティー。これはニックネームに含まれる行数として定義されます。フェデレーテッド・サーバーは個々のニックネームのカーディナリティーを、システム表 SYSCAT.TABLES または SYSSTAT.TABLES (のいずれかの表の「CARD」列) に格納します。カーディナリティーをニックネームに使用できない場合、コスト・モデルはデフォルト値の 1000 行を使用します。
  2. ニックネームのセットアップ・コスト。セットアップ・コストは、ラッパーが照会フラグメントをリモート・ソースにサブミットできる状態にするまでにかかる標準的な時間 (ミリ秒単位) を表します。セットアップは、ラッパーが、照会の計画時に生成したラッパー実行記述子を受け取った時点から始まり、ラッパーが対応する操作をリモート・ソースにサブミットできるようになった時点で終わります。セットアップ・コストには、ラッパーが同じ照会フラグメントを (おそらくは異なるパラメーター値を使用して) 再度実行するよう要求されても繰り返す必要がない作業のみが含まれているべきです。たとえば、ラッパーが照会フラグメントを URL の形式のリモート・ソースにサブミットする場合、セットアップ・コストには、ラッパーが実行記述子に格納した情報から URL を生成するのに必要な時間が含まれます。フェデレーテッド・サーバーはこの統計を SETUP_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 2000 ミリ秒の値を使用します。
  3. ニックネームのサブミット・コスト。サブミット・コストは、ラッパーが照会フラグメントをリモート・ソースにサブミットするのにかかる標準的な時間 (ミリ秒単位) を表します。サブミットは、上で定義されているセットアップが終了した時点から始まり、ラッパーがソースに対して最初の行または結果デー

タのブロックを要求できるようになった時点で終わります。サブミット・コストには、特定の照会フラグメントがサブミットされるたびにラッパーが繰り返す必要がある作業のみが含まれているべきです。たとえば、リモート・ソースとの各対話用に新しい HTTP 接続が必要な場合、サブミット・コストには、この接続を作成するのに必要な時間が含まれていなければなりません。フェデレーテッド・サーバーはこの統計を SUBMISSION_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 25 ミリ秒の値を使用します。

4. ニックネームの拡張コスト。これは、ニックネーム用の単一行を取り出すのにかかる標準的な時間 (ミリ秒単位) です。これには、照会を開始するのに必要な時間は含まれません。フェデレーテッド・サーバーはこの統計を ADVANCE_COST ニックネーム・オプションに格納します。ニックネームにそのオプションがない場合、コスト・モデルは 50 ミリ秒の値を使用します。データ・ソースがデータを行単位ではなくブロック単位で戻す場合は、ブロックを取り出すための標準コストを、ブロック当たりの標準行数で除算することによって、拡張コストを計算します。

すべてのラッパーに必要なカスタマイズ

Unfenced_Generic_Nickname サブクラスは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは対応する Unfenced_Generic_Nickname コンストラクターを呼び出す必要があります。

その他のカスタマイズ

- Unfenced_Generic_Nickname サブクラスのインスタンスに追加の情報を格納する必要がある場合は、initialize_my_nickname 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているニックネーム情報オブジェクトからこの情報を抽出します。 C++ では、このニックネーム情報オブジェクトへのポインターを保存することはできません。 Java では、ニックネーム情報オブジェクトへの参照を保存することができます。
- ニックネーム・オプションや列オプションを定義する場合は、verify_my_register_nickname_info および verify_my_alter_nickname_info (Java では verifyMyRegisterNicknameInfo および verifyMyAlterNicknameInfo) のデフォルト・インプリメンテーションをオーバーライドして、DDL で指定されているオプションと値の有効性を検査します。 DDL で指定されているオプション値を変更したり、追加のオプションを指定したりする場合は、インプリメンテーションで「差分」ニックネーム情報オブジェクトによってオーバーライド/追加情報が指定されていなければなりません。新しい「差分」オブジェクトを割り振る前に、「差分」オブジェクトがすでに存在していないかどうかチェックしてください。存在している場合は、それを使用し、他の差分オブジェクトを割り振ることはしないでください。 Java では、「差分」オブジェクトは実際には戻されるオブジェクトで、ラッパーにより作成される必要があります。
- Unfenced_Generic_Nickname サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラスのデストラクターをインプリメントする必要があります。

表 22. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトの動作
initialize_my_nickname	initializeMyNickname	ノーオペレーション
verify_my_register_nickname_info	verifyMyRegisterNicknameInfo	(C++ では is_reserved_nickname_option() 関数、Java では CatalogOption.isReserved() メソッドに基づく) 事前定義のリストと比較してオプションを検査します。他のオプションはエラーを生成します。
verify_my_alter_nickname_info	verifyMyAlterNicknameInfo	(C++ では is_reserved_nickname_option() 関数、Java では CatalogOption.isReserved() メソッドに基づく) 事前定義のリストと比較してオプションを検査します。他のオプションはエラーを生成します。

Fenced_Generic_Nickname クラス

fenced サーバー・サブクラスのインスタンス上でニックネームの作成メソッドを呼び出すと、fenced ニックネーム・サブクラスのインスタンスが作成されます。このメソッドは通常、CREATE NICKNAME ステートメントを実行した結果として呼び出されます。 DDL 操作でニックネームに関係した情報が変更されると、フェデレーテッド・サーバーはそのニックネーム・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、Fenced_Generic_Nickname 基本クラスのインプリメンテーションは、以下の情報を保守します。

- ニックネームを定義するリモート・データ・セットのローカル名。
- ニックネームを定義するリモート・データ・セットのローカル・スキーマ。
- 該当するサーバー・オブジェクトを指すポインター。

すべてのラッパーに必要なカスタマイズ

Fenced_Generic_Nickname サブクラスは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは、対応する Fenced_Generic_Nickname コンストラクターを呼び出します。

その他のカスタマイズ

- Fenced_Generic_Nickname サブクラスのインスタンスに追加の情報を格納する必要がある場合は、initialize_my_nickname 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているニックネーム情報オブジェクトからこの情報を抽出します。 C++ では、このニックネーム情報オブジェクトへのポインターを保存することはできません。 Java では、ニックネーム情報オブジェクトへの参照を保存することができます。
- ニックネーム・オプションを定義する場合は、verify_my_register_nickname_info 関数のデフォルト・インプリメンテーションをオーバーライドして、DDL で指定されているオプションと値の有効性を検査します。 DDL で指定されているオプションと値を変更したり、追加のオプションを指定したり、DDL ではなくデータ・

ソースから取得した情報とオプションを指定したりする場合は、インプリメンテーションで「差分」ニックネーム情報オブジェクトを使用してオーバーライド情報または追加情報が指定されていなければなりません。新しい「差分」オブジェクトを割り振る前に、「差分」オブジェクトがすでに存在していないかどうかチェックしてください。存在している場合は、それを使用し、他の差分オブジェクトを割り振ることはしないでください。 Java では、「差分」オブジェクトは実際には戻されるオブジェクトで、ラッパーにより作成される必要があります。

- Unfenced_Generic_Nickname サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラスのデストラクターをインプリメントする必要があります。

表 23. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトの動作
initialize_my_nickname	initializeMyNickname	ノーオペレーション
verify_my_register_nickname_info	verifyMyRegisterNicknameInfo	(C++ では is_reserved_nickname_option()、Java では CatalogOption.isReserved() メソッドに基づく) 事前定義のリストと比較してオプションを検査します。他のオプションはエラーを生成します。

表 24. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
get_card	getCard	アクセサー
get_setup_cost	なし	アクセサー
get_advance_cost	なし	アクセサー
get_submission_cost	なし	アクセサー

関連タスク:

- 80 ページの『ラッパー・クラス』
- 83 ページの『サーバー・クラス』
- 91 ページの『ユーザー・クラス』

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Java API のニックネーム・クラス』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Nickname クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『C++ API のニックネーム・クラス』

ユーザー・クラス

以下のセクションでは、Unfenced_Generic_User クラスと fenced 汎用ユーザー・クラスについて説明します。

Unfenced_Generic_User クラス

unfenced サーバー・サブクラスのインスタンス上でリモート・ユーザーの作成メソッドを呼び出すと、unfenced ユーザー・サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、CREATE USER MAPPING または ALTER USER MAPPING ステートメントを処理するときに、このメソッドを呼び出します。 DDL 操作でリモート・ユーザーに関係した情報が変更されると、フェデレーテッド・サーバーはそのリモート・ユーザー・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、Unfenced_Generic_User 基本クラスのインプリメンテーションは、以下の情報を保守します。

- ユーザーのローカル authid。
- DDL ステートメントを実行した結果、フェデレーテッド・サーバーのシステム・カタログに格納された、この (サーバーとユーザーの) ペアに関連する情報を含むユーザー情報オブジェクト。
- 該当するサーバー・オブジェクトを指すポインター。

すべてのラッパーに必要なカスタマイズ

データ・ソースが認証情報を要求しない場合は、unfenced 汎用ユーザー基本クラスをカスタマイズする必要はありません。

Unfenced_Generic_User サブクラスを作成する場合、それは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは対応する Unfenced_Generic_User コンストラクターを呼び出す必要があります。

その他のカスタマイズ

- Unfenced_Generic_User サブクラスのインスタンスに追加の情報を格納する必要がある場合は、initialize_my_user 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているユーザー情報オブジェクトからこの情報を抽出します。 C++ では、このユーザー情報オブジェクトへのポインターを保存することはできません。その代わり、このフォームに情報を格納する場合は、このオブジェクトのコピーを含むデータ・メンバーを使用します。 Java では、UserInfo オブジェクトへの参照を保存できますが、コピー機能はありません。
- ユーザー・オプションを定義する場合は、verify_my_register_user_info および verify_my_alter_user_info 関数のデフォルト・インプリメンテーションをオーバーライドして、DDL で指定されているオプションと値の有効性を検査します。DDL で指定されているオプション値を変更したり、追加のオプションを指定したりする場合は、インプリメンテーションで「差分」ユーザー情報オブジェクトによってオーバーライド/追加情報が指定されていなければなりません。新しい「差分」オブジェクトを割り振る前に、「差分」オブジェクトがすでに存在していないかどうかチェックしてください。存在している場合は、それを使用し、他の差分オブジェクトを割り振ることはしないでください。 Java では、「差分」オブジェクトは実際には検査メソッドにより戻されるオブジェクトで、ラッパーにより作成される必要があります。

- Unfenced_Generic_User サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 25. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
verify_my_register_user_info	verifyMyRegisterUserInfo	はい	DB2 UDB オプションだけが指定されているかどうかを検査します。
initialize_my_user	initializeMyUser	はい	ノーオペレーション
verify_my_alter_user_info	verifyMyAlterUserInfo	はい	DB2 UDB オプションだけが指定されているかどうかを検査します。

表 26. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
get_local_name	getLocalName	アクセサー

Fenced_Generic_User クラス

fenced サーバー・サブクラスのインスタンス上でリモート・ユーザーの作成メソッドを呼び出すと、fenced ユーザー・サブクラスのインスタンスが作成されます。このメソッドは、アプリケーションが、リモート接続オブジェクトを使用して、サーバーへの接続を最初に要求する前に、フェデレーテッド・サーバーによって呼び出されます。 DDL 操作でリモート・ユーザーに関係した情報が変更されると、フェデレーテッド・サーバーはそのリモート・ユーザー・オブジェクトが次に使用される前に、それを破棄して再作成します。

特にカスタマイズされていない場合、fenced 汎用ユーザー基本クラスのインプリメンテーションは、以下の情報を保守します。

- ユーザーのローカル authid。
- DDL ステートメントを実行した結果、フェデレーテッド・サーバーのシステム・カタログに格納された、このサーバーとユーザーのペアに関連する情報を含むユーザー情報オブジェクト。
- 該当するサーバー・オブジェクトを指すポインター。

すべてのラッパーに必要なカスタマイズ

データ・ソースが認証情報を要求しない場合は、Fenced_Generic_User 基本クラスをカスタマイズする必要はありません。

Fenced_Generic_User サブクラスを作成する場合、それは以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは対応する Fenced_Generic_User コンストラクターを呼び出す必要があります。

その他のカスタマイズ

- fenced 汎用ユーザー・サブクラスのインスタンスに追加の情報を格納する必要がある場合は、initialize_my_user 関数のデフォルト・インプリメンテーションをオーバーライドして、パラメーターとして提供されているユーザー情報オブジェクトからこの情報を抽出します。 C++ では、このユーザー情報オブジェクトへのポインターを保存することはできません。その代わり、このフォームに情報を格納する場合は、このオブジェクトのコピーを含むデータ・メンバーを使用します。 Java では、UserInfo オブジェクトへの参照を保存できますが、「コピー」機能はありません。
- Fenced_Generic_User サブクラスが、ユーザーが割り振った不適切なストレージを指している場合は、このストレージを解放する、サブクラス用のデストラクターをインプリメントする必要があります。

表 27. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
initialize_my_user	initializeMyUser	はい	ノーオペレーション

表 28. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
get_local_name	getLocalName	アクセサー

関連タスク:

- 80 ページの『ラッパー・クラス』
- 83 ページの『サーバー・クラス』
- 88 ページの『ニックネーム・クラス』

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Java API のユーザー・クラス』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『C++ API のユーザー・クラス』

Request クラス

このクラスは、照会の計画時に Request-Reply-Compensate プロトコルの一部として使用されます。フェデレーテッド・サーバー・オプティマイザーはこのクラスのインスタンスを生成して、データ・ソースが評価することを求められる可能性がある各照会フラグメントを記述します。

メソッド

表 29. メソッド

C++ での名前	Java での名前	説明
get_number_of_quantifiers	getNumberOfQuantifiers	FROM 文節内のエレメントの数を検索します。
get_number_of_predicates	getNumberOfPredicates	WHERE 文節内のエレメントの数を検索します。
get_number_of_head_exp	getNumberOfHeadExp	SELECT 文節内のエレメントの数を検索します。
get_quantifier_handle	getQuantifier.getHandle	FROM 文節内の指定位置のエレメントに対応するハンドルを検索します。
get_predicate_handle	getPredicate.getHandle	WHERE 文節内の指定位置のエレメントに対応するハンドルを検索します。
get_head_exp_handle	getHeadExp.getHandle	SELECT 文節内の指定位置のエレメントに対応するハンドルを検索します。
get_nickname	getNickname	FROM 文節内のハンドルで指定された数量詞に対応する Nickname オブジェクトを検索します。
get_head_exp	getHeadExp	SELECT 文節内のハンドルで指定されたヘッド式に対応する Request Exp オブジェクトを検索します。
get_predicate	getPredicate	WHERE 文節内のハンドルで指定された述部に対応する Request Exp オブジェクトを検索します。
get_distinct	getDistinct	DISTINCT 標識をテストします。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Request クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『C++ API の要求クラス』

Reply クラス

ラッパーは、このクラスのインスタンスを生成して、データ・ソースが実行できる要求内の照会フラグメントの一部を表します。ラッパーは、単一の Request に対していくつかの応答を生成できます。

Reply クラスは、以下を行うためにメソッドとデータ・メンバーを追加します。

- クラスに項目 (add_to_CCCC) を追加することによって、応答にデータを追加する。
- 実行記述子のポインターとサイズを格納する。
- ラッパーが同一の要求に対して複数の計画を戻した場合に応答をチェーニングする。

- 順序情報 (オプティマイザーが最適な計画を立てるために使用できる、戻りデータの興味に基づく順序付け) を格納する。

拡張カスタマイズ

コスト・メソッドのデフォルト・インプリメンテーションは、ソースの実行モデルについてより正確に記述するコスト・メソッドでオーバーライドできます。デフォルト・コストのオーバーライドは、照会フラグメント内の述部のデフォルトの選択度見積もりを基にします。 Unfenced_Generic_Server::get_selectivity メソッド (Java では UnfencedGenericServer.getSelectivity メソッド) を呼び出すと、オーバーライドも可能な選択度見積もりを取得できます。

メソッド

表 30. メソッド

C++ での名前	Java での名前	デフォルトがあるか?	説明
add_head_exp	addHeadExp	はい	次の空き位置にヘッド式ハンドルを追加します。
add_predicate	addPredicate	はい	次の空き位置に述部ハンドルを戻します。
add_quantifier	addQuantifier	はい	次の空き位置に数量詞ハンドルを戻します。
add_order_entry	addOrderEntry	はい	順序項目を追加します。順序項目は、方向 (ASC または DSC) と、順番に並んだヘッド式の整数索引で構成されます。
get_parameter_order	なし	はい	パラメーターの必要な照会フラグメントを表す応答のパラメーターの順序を戻します。これは、WHERE 文節と SELECT 文節内のすべての式の先行順巡回を実行して、順序を生成します。パラメーター・ハンドルのリストを戻します。
Cardinality	Cardinality	はい	応答で示された照会フラグメントの実行により戻された結果のカーディナリティーを戻します。 cardinality() のデフォルト・バージョンは、すべてのニックネームのカーディナリティーに、述部の選択度を乗算した積を戻します。
total_cost	totalCost	はい	応答で示された照会フラグメントを実行するために必要な合計実行コスト (ミリ秒単位)。デフォルト・バージョンは、all_costs を呼び出して、合計コスト値を戻します。

表 30. メソッド (続き)

C++ での名前	Java での名前	デフォルトがあるか?	説明
re_exec_cost	reExecCost	はい	応答で示された照会フラグメントを再実行するために必要な時間。デフォルトは、all_costs() を呼び出して、再実行コスト値を戻します。 **注:** 開発者は、このルーチンをオーバーライドするよりは、all_costs() をオーバーライドすることをお勧めします。
first_tuple_cost	firstTupleCost	はい	応答で示された照会フラグメントの最初の結果タプルを取得するために必要な時間。デフォルトは、all_costs() を呼び出して、最初のタプルのコスト値を戻します。 **注:** 開発者は、このルーチンをオーバーライドするよりは、all_costs() をオーバーライドすることをお勧めします。
set_next_reply	setNextReply	はい	ラッパーが要求に対して複数の応答を戻した場合に、応答をチェーニングします。

表 30. メソッド (続き)

C++ での名前	Java での名前	デフォルトがあるか?	説明
all_costs	なし	いいえ。インプリメントする必要あり	合計コスト、最初のタプルのコスト、および再実行コストの 3 つのコスト値を計算します。 **最初のタプルのコスト** = 平均セットアップ・コスト + 平均サブミット・コスト + 平均拡張コスト **再実行コスト** = 平均サブミット・コスト + (平均拡張コスト * 見積もりカーディナリティー) **合計コスト** = 平均セットアップ・コスト + 平均サブミット・コスト + (平均拡張コスト * 見積もりカーディナリティー) ここで、 **平均セットアップ・コスト** 照会フラグメントの中のすべてのニックネームのセットアップ・コストの平均。 **平均サブミット・コスト** 照会フラグメントの中のすべてのニックネームのサブミット・コストの平均。 **平均拡張コスト** 照会フラグメントの中のすべてのニックネームの拡張コストの平均。 **見積もりカーディナリティー** 照会フラグメントの (cardinality() メソッドによって戻された) 見積もりカーディナリティー。
get_number_of_quantifiers	getNumberOfQuantifiers	はい	FROM 文節内のエレメントの数を検索します。
get_number_of_predicates	getNumberOfPredicates	いいえ。インプリメントする必要あり	WHERE 文節内のエレメントの数を検索します。
get_number_of_head_exp	getNumberOfHeadExp	いいえ。インプリメントする必要あり	SELECT 文節内のエレメントの数を検索します。
get_quantifier_handle	getQuantifier.getHandle	いいえ。インプリメントする必要あり	FROM 文節内のある位置のハンドルを検索します。

表 30. メソッド (続き)

C++ での名前	Java での名前	デフォルトがあるか?	説明
get_predicate_handle	getPredicate.getHandle	いいえ。インプリメントする必要あり	WHERE 文節内のある位置のハンドルを検索します。
get_head_exp_handle	getHeadExp.getHandle	いいえ。インプリメントする必要あり	SELECT 文節内のある位置のハンドルを検索します。
get_nickname	getNickname	いいえ。インプリメントする必要あり	FROM 文節内の特定のハンドルにある数量詞に対応する、クラス Nickname のオブジェクトを検索します。
get_head_exp	getHeadExp	いいえ。インプリメントする必要あり	SELECT 文節内の特定のハンドルにあるヘッド式に対応する、クラス Request_Exp のオブジェクトを検索します。
get_predicate	getPredicate	いいえ	WHERE 文節内の特定のハンドルにある述部に対応する、クラス Request_Exp のオブジェクトを検索します。
set_distinct	setDistinct	いいえ。インプリメントする必要あり	DISTINCT 標識 (SELECT DISTINCT 文節) を検索します。
get_distinct	getDistinct	いいえ。インプリメントする必要あり	DISTINCT 標識をテストします。
get_exec_desc	getExecDistinct	いいえ。インプリメントする必要あり	この照会用の実行記述子のポインターと長さを検索します。
set_exec_desc	setExecDesc	いいえ。インプリメントする必要あり	実行記述子を Reply に格納します。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『Reply クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Reply クラス (C++)』

述部リスト・クラス

述部リスト・クラスのインスタンスは、選択度見積もりメソッド、Unfenced_Generic_Server::getSelectivity() (Java では UnfencedGenericServer.getSelectivity) への入力です。 2 つの述部リストは以下のとおりです。

- 選択度の送信を請求された、述部のセット P。
- すでに適用済みの述部のセット、AP。

結果の選択度は、AP が選択されている場合の P の*条件付き* 選択度、つまり、セット AP の中の述部がすでに適用済みである場合のセット P の中の述部の選択度になります。

```
selectivity(P/AP)
```

無条件の選択度が必要な場合は、AP を NULL にすることができます。述部のリストは、Request の中の述部のリストと類似しており、類似のメソッドで操作されます。

メソッド

表 31. メソッド

C++ での名前	Java での名前	説明
create_empty_predicate_list	なし	要求の中の述部リストと同じサイズの、空の述部リストを作成します。この結果として作成されたリストは空です。
create_and_copy_predicate_list	なし	空の述部リストを作成し、応答の述部リストをコピーします。
operator new	なし	QG ヒープに述部リストを割り振ります。
destructor	なし	Predicate_List オブジェクトはラッパーによって作成され、ラッパーによって破棄される必要があります。
get_number_of_predicates	getNumberOfPredicates	述部リストの長さを取り出します。
get_predicate	getPredicate	ハンドルにある述部について記述する Request_Exp オブジェクトを戻します。
get_predicate_handle	getPredicateHandle	ある位置の述部のハンドルを戻します。
get_number_of_applied_predicates	getNumberOfAppliedPredicates	適用済みの述部リストの長さを戻します。
get_applied_predicate	getAppliedPredicate	ハンドルにある適用済みの述部について記述する Request_Exp オブジェクトを戻します。
get_applied_predicate_handle	getAppliedPredicateHandle	ある位置の適用済みの述部のハンドルを戻します。
add_predicate	addPredicate	述部リスト内の次の空き位置に述部ハンドルを追加します。
add_applied_predicate	addAppliedPredicate	適用済みの述部リスト内の次の空き位置に、適用済みの述部ハンドルを追加します。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『PredicateList クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Predicate_List クラス (C++)』

式の要求クラス

このクラスは、式の要求を探索するためのインターフェースを定義します。式は、ヘッド式か述部になります。これらの式はすべて、解析される演算子のツリーとして表されます。式は再帰的に定義されるので、各ツリー・ノードそのものが式になります。ツリーの内部ノード (副次式) は演算子を表し、リーフは定数、列、またはパラメーターを表します。ラッパーの作成者が使用できる情報は、式のタイプに応じて異なります。ラッパーが使用できる情報は、式の種類ごとに異なります。すべての式には、種類を戻すメソッドに加えて、式のデータ・タイプ、その親式 (ルート式の場合は NULL)、およびその兄弟式 (存在する場合は右隣りの子で、存在しない場合は NULL) を戻すメソッドがあります。個々の式の種類ごとに、以下の表に示されているような追加情報を戻すメソッドが存在します。

表 32. 各式のタイプ別の追加情報

式のタイプ	情報
演算子	• 子 (オペランド) の数 • 最初のオペランド • (照会から解析された) トークン • シグニチャー - オペランドのデータ・タイプを含む、解決された関数/演算子名
列	• 列名 • 列が所属する数量詞
定数	値バッファー、長さ、およびタイプ
パラメーター	エンジンからラッパーにパラメーターを渡すために使用されるパラメーター配列中のパラメーターの位置を識別するために使用される、パラメーター・ハンドル。

メソッド

表 33. C++ の式の要求クラスのメソッド

C++ での名前	説明
get_kind	式の種類 (Request_Exp::oper、Request_Exp::column, Request_Exp::unbound、Request_Exp::constant) を戻します。関数に対する最後の引き数は、オプティマイザーから呼び出されるときにのみ必要です。
get_data_type	式の結果データ・タイプについて記述する Request_Exp_Type インスタンスを戻します。
get_parent	式の親ノードを戻します。
get_next_child	兄弟ノードを戻します。

表 33. C++ の式の要求クラスのメソッド (続き)

C++ での名前	説明
get_column_name	列名を含むストリングを戻します。
get_column_quantifier_handle	この列が属している数量詞を戻します。
get_value	式の中の定数の値とタイプを表す Request_Constant インスタンスを戻します。
get_parameter_handle	パラメーターを一意的に識別する整数を戻します。
get_number_of_children	演算式のオペランドの数を戻します。
get_first_child	最初のオペランドを戻します。
get_token	トークン値をストリングとして戻します。
get_signature	演算子のシグニチャーを戻します。

表 34. Java の式の要求クラスのメソッド

Java での名前	説明
getKind	式の種類 (RequestExp.OPERATOR、 RequestExp.CONSTANT、RequestExp.UNBOUND、 RequestExp.COLUMN) を戻します。
getDataType	式の結果データ・タイプについて記述する RequestExpType インスタンスを戻します。
getParent	式の親ノードを戻します。
getNextChild	兄弟ノードを戻します。
getColumnName	列名を含むストリングを戻します。
getQuantifier	この列が属している数量詞を戻します。
getValue	式の中の定数の値を表す RequestConstant インスタンスを戻します。
getNumberOfChildren	演算式のオペランドの数を戻します。
getFirstChild	最初のオペランドを戻します。
getToken	トークン値をストリングとして戻します。
getSignature	演算子のシグニチャーを戻します。
getHandle	式のハンドルを戻します。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RequestExp クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Request_Exp クラス (C++)』

定数の要求クラス

このクラスは、照会式の中の定数について記述するために使用します。Request_Exp::get_value() (Java では RequestExp getValue) 関数は、Request_Exp::constant の種類 (Java では RequestExp.CONSTANT) のノードに適用されると、定数の要求クラスのインスタンスを戻します。

メソッド

表 35. メソッド

C++ での名前	Java での名前	説明
get_actual_length	getActualLength	バッファー内のデータの長さを取得します。
get_data	getData	データ・バッファーへのポインターを取得します。
get_data_type	getDataType	列の DB2 UDB タイプ ID を戻します。
is_data_null	isDataNull	意図的、または算術計算エラー時の NULL かどうか?
get_precision	getPrecision	数値定数値の精度を戻します。
get_scale	getScale	数値定数値の位取りを戻します。
get_codepage	getCodepage	文字値のコード・ページを戻します。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RequestConstant クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Request_Constant クラス (C++)』

式タイプの要求クラス

このクラスのインスタンスは、照会式のデータ・タイプについて記述します。各照会式 (副次式も含む) は、タイプ分けされています。関数 Request_Exp::get_data_type() (Java では RequestExpType getDataType) は、式タイプの要求インスタンスを戻します。

メソッド

表 36.

C++ での名前	Java での名前	説明
get_for_bit_data	getForBitData	列にバイナリー・データが含まれるか? 有効値は、Y と N です。Y は Yes、N は No を表します。 Java メソッド getForBitData は、ブール値 (Y と N ではない) を戻します。
get_null_indicator	getNullIndicator	どの列が NULL 標識であるかを指定します
get_data_type	getDataType	C++ では、列の SQL タイプ (sql.h より)。 Java では、タイプ ID 定数は Data クラスで定義されます。

表 36. (続き)

C++ での名前	Java での名前	説明
get_maximum_length	getMaximumLength	列の最大長 (SQL_TYP_DECIMAL を除く)
get_precision	getPrecision	精度 (SQL_TYP_DECIMAL のみ)
get_scale	getScale	位取り (SQL_TYPE_DECIMAL のみ)
get_codepage	getCodepage	列のコード・ページ
get_name	getName	列名 (あれば)
get_name_length	getNameLength	列名の長さ

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RequestExpType クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Request_Exp_Type クラス (C++)』

リモート接続クラス

リモート接続インスタンスを作成するには、該当する fenced サーバー・サブクラスのインスタンス上でメソッド create_remote_connection (Java では createRemoteConnection) を呼び出します。フェデレーテッド・サーバーは、関係のあるサーバーで最初のリモート操作を処理する前に、このメソッドを呼び出します。フェデレーテッド・サーバーは、アプリケーションがサーバーを使用せずに指示された数のトランザクションを実行した後、リモート接続インスタンスを破棄します。

特にカスタマイズされていない場合、リモート接続基本クラスのインプリメンテーションは、以下の情報を保守します。

- データ・ソースへの接続の状態 (接続または切断)。
- 該当するサーバー・オブジェクトを指すポインター。
- 適切なユーザー・オブジェクトへのポインター。
- この接続のためにインスタンス化されるリモート操作オブジェクトのリスト。
- この接続に使用するコード・ページ。これは、フェデレーテッド・サーバーに接続されているクライアント・アプリケーションのコード・ページです。
- 接続の種類: フェーズなし、1 フェーズ、2 フェーズ (バージョン 8.2 では、2 フェーズ・トランザクションはサポートされていません)。

すべてのラッパーに必要なカスタマイズ

リモート接続サブクラスは、サブクラスのコンストラクターをインプリメントする必要があります。これは、対応するリモート接続コンストラクターを呼び出します。

リモート接続クラスは、ラッパーの必要に応じて他のメソッドもインプリメントすることが必要な場合があります。以下のリストは、コンストラクターとその使用法について説明しています。

- connect: データ・ソースへの接続を確立するメソッド。データ・ソースで接続が必要がない場合でも、このメソッドをインプリメントする必要があります。
- disconnect: データ・ソースへの確立済みの接続をクローズするメソッド。データ・ソースで接続が必要がない場合でも、このメソッドをインプリメントする必要があります。

  接続を確立または閉じた後は、mark_disconnected メソッド (Java では markDisconnected メソッド) への呼び出しは必要ありません。
- リモート照会の作成メソッド: リモート照会サブクラスのインスタンスを作成します。

  C++: create_remote_query

  Java:
  createRemoteQuery
- commit: データ・ソースで現在のトランザクションをコミットするメソッド。passthru セッションに接続が使用されていて、データ・ソースがトランザクションをサポートする場合、インプリメンテーションでは、データ・ソースで現在のトランザクションをコミットする必要があります。 passthru 以外のトランザクションはすべてバージョン 8.1 では読み取り専用になりますが、データ・ソースでは、リソースを解放し、読み取りロックをドロップするなどのために、トランザクション終了通知が必要です。データ・ソースがトランザクションをサポートしない場合でも、このメソッドをインプリメントする必要があります。
- rollback: データ・ソースで現在のトランザクションをロールバックするメソッド。 passthru セッションに接続が使用されていて、データ・ソースがトランザクションをサポートする場合、インプリメンテーションでは、データ・ソースで現在のトランザクションをロールバックする必要があります。 passthru 以外のトランザクションはすべてバージョン 8.1 では読み取り専用になりますが、データ・ソースでは、リソースを解放し、読み取りロックをドロップするなどのために、トランザクション終了通知が必要です。データ・ソースがトランザクションをサポートしない場合でも、このメソッドをインプリメントする必要があります。

さらに、インプリメンテーションのいずれかの部分で、データ・ソースへの接続の消失が検出される場合、切断のマーク・メソッドが呼び出され、エラーが戻されます。切断のマーク・メソッドは、接続が消失し、適切なクリーンアップが開始することをデータベース・マネージャーに通知します。

その他のカスタマイズ

- データ・ソースに接続ハンドルなどの接続についての情報を保管する必要がある場合、サブクラス定義に適切なデータ・メンバーを追加して、コンストラクターで初期設定します。
- リモート接続サブクラスによって指示された不適切なストレージを割り振った場合、このストレージを解放する、サブクラスのデストラクターをインプリメントする必要があります。デストラクターは、データ・ソースへの接続をクローズする必要はありません。データ・ソースへの接続が存在する可能性がある場合、フェデレーテッド・サーバーは、リモート接続オブジェクトを削除する前に、必ず disconnect メソッドを呼び出します。

- ラッパーが passthru をサポートする場合、デフォルト・インプリメンテーションのリモート passthru の作成メソッドをオーバーライドする必要があります。インプリメンテーションでは、リモート passthru サブクラスのインスタンスを作成する必要があります。

表 37. 仮想関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
connect	connect	いいえ。インプリメントする必要あり	
disconnect	disconnect	いいえ。インプリメントする必要あり	
create_remote_query	createRemoteQuery	はい	エラー
create_remote_passthru	createRemotePassthru	はい	エラー
commit	commit	いいえ。インプリメントする必要あり	
rollback	rollback	いいえ。インプリメントする必要あり	

表 38. 公開/保護メンバー関数

C++ の公開/保護メンバー関数	Java の公開/保護メンバー関数	動作
is_connected	isConnected	データ・ソースへ接続しているか?
mark_connected	markConnected	データ・ソースへ接続している
mark_disconnected	markDisconnected	データ・ソースへ接続していない
get_kind	getKind	アクセサー
get_codepage	getCodepage	アクセサー

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Remote_Connection クラス (C++)』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RemoteConnection クラス (Java)』

リモート照会クラス

適切なリモート接続サブクラスのインスタンスでリモート照会クラスの作成メソッドを呼び出すと、リモート照会サブクラスのインスタンスが作成されます。フェデレーテッド・サーバーは、照会をサブミットしたアプリケーションが照会の結果セットでカーソルをクローズすると、リモート照会オブジェクトを破棄します。

データ・ソースの種類によっては、データ・ソースがデータ値を戻す順序をラッパーが制御できない場合もあります。ラッパーが LOB でないデータを取得した後で

LOB を取得し、その後で LOB でないデータを取得するということが続く場合があります。ラッパーは選択の余地がなく、データ・ソースから受け取った順序でデータを処理する必要があります。

サブクラスのコンストラクターをインプリメントする必要があります。これは、対応するリモート照会コンストラクターを呼び出します。出力ランタイム・データ・リストは、リモート操作基本クラス・コンストラクターにより作成されます。リモート照会サブクラスは、以下のメソッドをインプリメントする必要があります。以下の説明では、C++ での名前を使用しています。Java でラッパーを作成する場合には、対応する Java のメソッド名を代わりに使用してください。

fetch()

fetch() メソッドは、LOB でないデータを取り出して、ランタイム・データ・リストによって識別されるフェデレーテッド・サーバー・バッファーに入れます。

fetch_lob()

fetch_lob() メソッドは、LOB データを、一度に 1 つのチャンクずつ取り出し、フェデレーテッド・サーバーが提供する特殊な LOB バッファーに入れます。 LOB 全体を処理するには、fetch_lob メソッドは複数回呼び出しを行う必要が生じる場合があります。個々の呼び出しで転送される量は、バッファーのサイズによって制限されるからです。

fetch() および fetch_lob()

fetch() および fetch_lob() メソッドは、調整された方式で作動し、リモート・ソースから着信した順序で LOB 列と LOB でない列を処理します。リモート照会オブジェクトは共有状態を提供するので、ラッパーはデータ・ストリーム中での位置を追跡できます。 fetch() または fetch_lob() から戻る前に、ラッパーはフェデレーテッド・サーバーからの次の呼び出しが fetch() または fetch_lob() のどちらになるかをゲートウェイに指示します。

open_input_lob()

open_input_lob() メソッドによって、ラッパーは入力 LOB パラメーターを含むリモート照会を初期化することができます。ラッパーが入力 LOB パラメーターをサポートする場合には、このメソッドをラッパー固有のリモート照会クラスにインプリメントする必要があります。 DB2 Information Integrator は、入力 LOB ホスト変数を検出した場合にこのメソッドを呼び出します。入力 LOB パラメーターがあることを示すために、ラッパーは次の LOB フラグメントを受け取るホスト変数の索引を戻し、set_row_status() を呼び出す必要があります。

ラッパーは、さらに処理する LOB 入力パラメーターがあることを知らせますが、DB2 Information Integrator はこのメソッドを別の LOB フラグメントで呼び出します。現在の LOB フラグメントのサイズ (バイト単位) はラッパーに渡され、ラッパーはその情報を使用して、次の入力変数に進むか、または入力値全体が読み取られたことを通知します。

reopen_input_lob()

reopen_input_lob() メソッドは、以前にオープンした結果ストリームをリセットし、データ・ソースが、入力 LOB パラメーターを持つ照会のための別のパラメーター・バインディングに基づくさらに多くの結果セットを戻せるように準備します。ラッパーが入力 LOB パラメーターをサポートする場合

には、このメソッドをラッパー固有のリモート照会クラスにインプリメントする必要があります。このメソッドは、照会が以前に照会の終了状況でクローズ (クローズ・メソッド) されていなければ、呼び出されません。入力 LOB ホスト変数が検出された場合には、DB2 Information Integrator は、reopen_input_lob() メソッドを呼び出します。入力 LOB パラメーターがあることを示すために、ラッパーは次の LOB フラグメントを受け取るホスト変数の索引を戻し、set_row_status() を呼び出す必要があります。

ラッパーは、さらに処理する LOB 入力パラメーターがあることを知らせますが、DB2 Information Integrator はこのメソッドを別の LOB フラグメントで呼び出します。 LOB 入力値と現在の LOB フラグメントのサイズ (バイト単位) はラッパーに渡され、ラッパーはそれらの値を使用して、次の入力変数に進むか、または入力値全体が読み取られたことを通知します。

set_row_status()

get_row_status() メソッドは、現行行の状況を設定します。このメソッドはすでにインプリメントされており、ラッパーによって多重定義することはできません。

get_row_status()

set_row_status() メソッドは、現行行の状況を取り出します。このメソッドはすでにインプリメントされており、ラッパーによって多重定義することはできません。

ここでのメソッド間で関数を配布する方法については、ユーザーの裁量にかなりの程度ゆだねられています。

表 39. 関数

C++ の仮想関数	Java の仮想関数	デフォルトがあるか?	デフォルトの動作
fetch	fetch	はい	エラー
fetch_lob	fetchLob	はい	エラー
open	open	はい	エラー
open_input_lob	openInputLob	はい	エラー
reopen	reopen	はい	エラー
reopen_input_lob	reopenInputLob	はい	エラー
close	close	はい	エラー
set_row_status	setRowStatus	はい	エラー
get_row_status	getRowStatus	はい	エラー

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RemoteQuery クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Remote_Query クラス (C++)』

ランタイム・データ・クラス

以下のセクションでは、ランタイム・データ・クラスおよびランタイム・データ・リスト・クラスについて説明します。

表 40. C++ および Java のランタイム・データ・リスト・クラス

C++	Java
Runtime_Data	RuntimeData
Runtime_Data_List	RuntimeDataList

ランタイム・データ・クラス

フェデレーテッド・サーバーは、サーバーとラッパーとの間で列値を転送するときにシステムが使用する各バッファーを表すために、ランタイム・データ・クラスのインスタンスを作成します。フェデレーテッド・サーバーは、その実行時にラッパーに値が転送される照会パラメーターごとにバッファーを作成します。フェデレーテッド・サーバーはさらに、ラッパーにより戻される結果行の列ごと (たとえば、照会フラグメントのヘッド式ごと) にも、1 つのバッファーを作成します。

以下の状態は維持されます。

- 列番号
- 値を記述する Runtime_Data_Desc (Java では RuntimeDataDesc) オブジェクトへのポインター
- データ・バッファーへのポインター
- バッファー内のデータの長さ
- パラメーター値を表す列の場合、そのパラメーター値が不変かどうかを示すフラグ。不変パラメーターの値は、「reopen」メソッドの「action」パラメーターを使用してラッパーに通知されない限り、変更されることはありません。

フェデレーテッド・サーバーは、列番号、データ記述、およびデータ・バッファーを提供します。パラメーターをフェデレーテッド・サーバーからラッパーへ渡すためにランタイム・データ・オブジェクトが使用されている場合、データ長およびバッファーの内容はフェデレーテッド・サーバーによって提供されます。バッファーの値には、付加された記述で指定された SQL タイプが入ります。ラッパーは、この値をデータ・ソースで受け入れられる任意のタイプに変換する必要があります。結果をラッパーからフェデレーテッド・サーバーへ戻すためにランタイム・データ・オブジェクトが使用されている場合、ラッパーは、データ長とバッファーの内容を提供します。バッファーの最大 (割り振り) 長と予想データ・タイプは、付加された記述から知ることができます。ラッパーは、データ・ソースから入手した値を、記述で指定された SQL タイプに変換する必要があります。

表 41. パブリック・メンバー関数

C++ のパブリック・メンバー関数	Java のパブリック・メンバー関数	動作
check_friendly_div_by_0	checkFriendlyDivBy0	ゼロによる除算?
check_friendly_exception	checkFriendlyException	演算例外?

表 41. パブリック・メンバー関数 (続き)

C++ のパブリック・メンバー関数	Java のパブリック・メンバー関数	動作
get_actual_length	getActualLength	バッファー内のデータの長さの入手
get_data	getData	データ・バッファーへのポインターの入手
get_data_index	getDataIndex	列番号の入手
get_invariant	getInvariant	入力値が不変かどうか?
is_data_null	isDataNull	セマンティック、または「算術計算エラー時」が NULL かどうか?
is_semantic_null	isSemanticNull	セマンティックが NULL かどうか?
set_actual_length	setActualLength	バッファー内のデータの長さの設定
set_data	setXX	バッファーへのデータのコピー
set_data_null	setDataNull	NULL 標識の設定
set_friendly_div_by_0	setFriendlyDivBy0	ゼロによる除算例外の通知。まず、NULL を設定する。
set_friendly_exception	setFriendlyException	演算例外の通知。まず、NULL を設定する。

ランタイム・データ・リスト・クラス

基本クラス・コンストラクターは、リモート照会のために、このクラスのインスタンスを作成します。出力行および入力パラメーターのために、別個のリストが作成されます。

表 42. パブリック・メンバー関数

C++ のパブリック・メンバー関数	Java のパブリック・メンバー関数	動作
get_number_of_values	getNumberOfValues	列数の入手
get_ith_value	getValue	i 番目の列のための Runtime_Data へのポインターの入手
operator[]	なし	i 番目の列のための Runtime_Data へのポインターの入手

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Runtime_Data クラス (C++)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Runtime_Data_List クラス (C++)』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RuntimeDataList クラス (Java)』

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RuntimeData クラス (Java)』

ランタイム・データ記述クラス

以下のセクションでは、ランタイム・データ記述クラスおよびランタイム・データ記述リスト・クラスについて説明します。

表 43. C++ および Java のランタイム・データ記述リスト・クラス

C++	Java
Runtime_Data_Desc	RuntimeDataDesc
Runtime_Data_Desc_List	RuntimeDataDescList

ランタイム・データ記述クラス

フェデレーテッド・サーバーは、ラッパーに列値を転送するときにシステムが使用する各バッファーを表すために、このクラスのインスタンスを作成します。フェデレーテッド・サーバーは、その実行時にラッパーに値が転送される照会パラメーターごとにバッファーを作成します。フェデレーテッド・サーバーはさらに、ラッパーにより戻される結果行の列ごと (たとえば、照会フラグメントのヘッド式ごと) にも、1 つのバッファーを作成します。リモート照会オブジェクトとリモート passthru オブジェクトでは、フェデレーテッド・サーバーは、入力データ・リスト内のランタイム・データ・オブジェクトごとに、パラメーター値を表すランタイム・データ記述を作成します。フェデレーテッド・サーバーは、列記述を提供します。リモート照会オブジェクトでは、フェデレーテッド・サーバーはさらに、出力データ・リスト内のランタイム・データ・オブジェクトごとに、結果行の値を表すランタイム・データ記述も作成します。フェデレーテッド・サーバーは、これらの列記述も提供します。

表 44. パブリック・メンバー関数

C++ のパブリック・メンバー関数	Java のパブリック・メンバー関数	動作
get_for_bit_data	getForBitData	列にバイナリー・データが含まれるか?
get_null_indicator	getNullIndicator	列は NULL 可能か ?
get_data_type	getDataType	列の SQL タイプ (sql.h より)
get_maximum_length	getMaximumLength	列の最大長 (SQL_TYP_DECIMAL を除く)
get_precision	getPrecision	精度 (SQL_TYP_DECIMAL のみ)
get_scale	getScale	位取り (SQL_TYPE_DECIMAL のみ)
get_codepage	getCodepage	列のコード・ページ
get_name	getName	列名 (あれば)
get_name_length	getNameLength	列名の長さ

ランタイム・データ記述リスト・クラス

表 45. パブリック・メンバー関数

C++ のパブリック・メンバー関数	Java のパブリック・メンバー関数	動作
get_number_of_values	getNumberOfValues	列数の入手
get_ith_value	getValue	i 番目の列のための Runtime_Data_Desc へのポインターの入手
set_ith_value	setValue	i 番目の列のための Runtime_Data_Desc へのポインターの設定
operator[]	なし	i 番目の列のための Runtime_Data_Desc へのポインターの入手

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Runtime_Data_Desc クラス (C++)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Runtime_Data_Desc_List クラス (C++)』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RuntimeDataDesc クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RuntimeDataDescList クラス (Java)』

リモート passthru クラス

適切なリモート接続サブクラスのインスタンスでリモート passthru クラスの作成メソッドを呼び出すと、リモート passthru サブクラスのインスタンスが作成されます。データ・ソースが passthru をサポートしていない場合、リモート passthru サブクラスをインプリメントしたり、このメソッドのデフォルトのインプリメンテーションをオーバーライドしないでください。

すべてのラッパーに必要なカスタマイズ

リモート passthru サブクラスでは、以下をインプリメントする必要があります。

- サブクラスのコンストラクター。これは、対応するリモート passthru コンストラクターを呼び出すものです。

  C++: Remote_Passthru

  Java:
  RemotePassthru
- prepare: データ・ソースに対して passthru ストリングをサブミットし、結果の各行を構成する列の数とタイプを判別できるようにするメソッド。
- describe: 結果の各行を構成する列の数とタイプを記述する、ランタイム・データ記述リストを取り込むメソッド。

- open: データ・ソースが照会の最初の結果行を戻す準備をできるようにするメソッド。
- fetch: 1 つの結果行を出力ランタイム・データ・リストへコピーするメソッド。
- close: データ・ソースが照会の実行後にクリーンアップできるようにするメソッド。

リモート照会の場合のように、前述のメソッド間で関数を配布することは、ラッパー作成者の裁量にかなりの程度ゆだねられています。たとえば、ランタイム・データ記述リストの取り込みは、prepare または describe メソッドのいずれかで発生する可能性があり、データ・ソースに対する passthru ストリングのサブミットは、open または fetch メソッドで発生する可能性があります。

その他のカスタマイズ

データ・ソースが、結果コードを戻すが行を戻さない passthru ストリングをサポートする場合、execute メソッドのデフォルト・インプリメンテーションをオーバーライドしてください。そのインプリメンテーションは、データ・ソースに対してストリングをサブミットし、結果コードを戻します。クライアント・アプリケーションが EXECUTE または EXECUTE IMMEDIATE 言語構造体で passthru ストリングをサブミットする場合、execute メソッドが呼び出され、passthru ストリングが適切な操作を表示しない場合 (つまり、操作が表示されても行が戻されない場合) に、エラーを報告することになります。

表 46. 仮想関数

C++ および Java の仮想関数	デフォルトの有無	デフォルトの動作
prepare	はい	エラー
describe	はい	エラー
execute	はい	エラー
open	はい	エラー
close	はい	エラー

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『RemotePassthru クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Remote_Passthru クラス (C++)』

ラッパー・ユーティリティー・クラス

このクラスは、インスタンス化してはなりません。このクラスは、ラッパーがフェデレーテッド・サーバーに備えられている各種サービスへアクセスできるようにする、静的メソッドの集合をグループ化するためだけに存在しています。

ラッパー・ユーティリティー・クラスの静的メソッドには、以下のサービスがあります。

- メモリーの割り振りと割り振り解除。インプリメンテーションでは、このメソッドを使用してメモリーの割り振りおよび割り振り解除を行う必要があります。ただし、Sqlqg_Base_Class から派生した C++ オブジェクトをインスタンス化するために「new」を使用している場合、または破棄するために「delete」を使用している場合は除きます。クラスが Sqlqg_Base_Class の下位でないか、「new」または「delete」の Sqlqg_Base_Class インプリメンテーションをオーバーライドする場合、これらのメソッドを使用してメモリーを割り振りおよび割り振り解除する演算子のインプリメンテーションを提供する必要があります。
- エラー報告。データ・ソースによって戻されたエラーを最も適切な DB2 UDB エラーまたは SQL コードに割り当てるのは、ラッパーの役割です。それぞれの SQL コードに、エラー・メッセージ・ストリングがありますが、大抵は、エラーが発生したときの環境に特有のトークンでパラメーター化されています。たとえば、オプションの欠落を報告するエラー・メッセージ・ストリングは、オプションの種類 (ラッパー、サーバーなど)、エンティティー名 (ラッパー名、サーバー名など)、および欠落オプションの名前を含むトークンでパラメーター化されます。ラッパーは、エラーを報告するときには、これらのトークンの値を提示する必要があります。最も適切な SQL コード、および関連付けられたトークンの内容を見つけるには、メッセージ・リファレンス を参照してください。

  最も適切な SQL コードを見つけた後、対応するシンボルの定義を見つけてください。 表 47 は、プラットフォーム別のディレクトリーを示しています。

表 47. プラットフォーム別のシンボルの定義のディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/include/sqlcodes.h
HP/Sun/Linux	/opt/IBM/db2/V8.1/include/sqlcodes.h
Windows	%DB2PATH%/include/sqlcodes.h

  デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

  他の SQL コードへすぐに割り当てることのできないエラーの場合、SQL_RC_E1822 (SQL コード = -1822) を使用します。
- コード・ページに適した大文字小文字に変換する。
- 現在の DB2 UDB データベースの単一バイトおよび 2 バイト・コード・ページへのアクセス。
- ラッパー・トレース機能からのラッパー制御フロー情報。

表 48. ラッパー・ユーティリティー・クラスのパブリック静的メンバー関数の動作

C++ のパブリック静的メンバー関数	Java のパブリック静的メンバー関数	動作
allocate	なし	メモリー・ブロックを割り振る。
deallocate	なし	メモリー・ブロックを解放する。

表 48. ラッパー・ユーティリティー・クラスのパブリック静的メンバー関数の動作 (続き)

C++ のパブリック静的メンバー関数	Java のパブリック静的メンバー関数	動作
report_error	WrapperException クラス (Java は例外を使用します)	SQL コード、メッセージ、およびトークンでエラーを報告する。
report_warning	reportWarning	警告を報告する。
convert_to_upper	なし	コード・ページに適した大文字小文字に変換する。
convert_to_lower	なし	コード・ページに適した大文字小文字に変換する。
get_sb_DB_codepage	getSingleByteDBCodepage	単一バイト・データベース・コード・ページを取得する。
get_db_DB_codepage	getDoubleByteDBCodepage	2 バイト・データベース・コード・ページを取得する。
string_to_tokens	なし	ストリングをトークン化する。
trace_data	なし	トレース機能へトレース情報を書き込む。
get_db2_install_path	getDB2InstallPath	インストールへのパスを検索する。
get_db2_instance_path	getDB2InstancePath	フェデレーテッド・サーバーへのパスを検索する。

表 48. ラッパー・ユーティリティー・クラスのパブリック静的メンバー関数の動作 (続き)

C++ のパブリック静的メンバー関数	Java のパブリック静的メンバー関数	動作
get_db2_release	getDB2Release	ラッパーが現在稼働している DB2 UDB のバージョン (フィックスパックを含む) を検索する。
fnc_entry	traceFunctionEntry	ラッパー・トレース機能用の関数への入り口を記録する。
fnc_exit	traceFunctionExit	ラッパー・トレース機能用の関数からの出口を記録する。
fnc_data	traceFunctionData	ラッパー・トレース機能を使用することにより、トレース・データを記録する。
fnc_data2	traceFunctionData	ラッパー・トレース機能を使用することにより、プローブ・ポイントと 2 つのデータ・エレメントを含むトレース・データを記録する。

表 48. ラッパー・ユーティリティー・クラスのパブリック静的メンバー関数の動作 (続き)

C++ のパブリック静的メンバー関数	Java のパブリック静的メンバー関数	動作
fnc_data3	traceFunctionData	ラッパー・トレース機能を使用することにより、プローブ・ポイントと 3 つのデータ・エレメントを含むトレース・データを記録する。
trace_error	traceException または traceError	ラッパー・トレース機能を使用することにより、エラー・コードとプローブ・ポイントを含むエラー・トレース・データを記録する。
convert_codepage	なし	入力データをソースのコード・ページからターゲットのコード・ページに変換する。
get_expected_conv_len	なし	元のコード・ページから新規のコード・ページに変換されるストリングのバイト数を戻す。
get_env_lang	なし	オペレーティング・システムの言語設定を戻す。

表 48. ラッパー・ユーティリティー・クラスのパブリック静的メンバー関数の動作 (続き)

C++ のパブリック静的メンバー関数	Java のパブリック静的メンバー関数	動作
change_endian2	なし	2 バイト文字ストリングのエンディアン・バイト・オーダーを変更する。

関連資料:

- 「*IBM DB2 Information Integrator ラッパー開発における Java API リファレンス*」の『WrapperUtilities クラス (Java)』
- 「*IBM DB2 Information Integrator ラッパー開発における C++ API リファレンス*」の『Wrapper_Utilities クラス (C++)』

第 10 章 ラッパーを環境と共存させるための確認

このセクションでは、ラッパーを確実に他の環境と共存させるために考慮する必要がある点を説明します。これには以下のトピックが含まれています。

- システム・サービスの処理中に注意する必要のある点
- 環境変数をラッパーにアクセスできるようにする方法
- ラッパーの移植性

ラッパーにおけるシステム・サービスの使用

ラッパーは他のフェデレーテッド・サーバー・プロセスと共存しなければならないので、オペレーティング・システム・サービスを使用する際には注意が必要です。

フェデレーテッド・サーバーが使用するサービスの完全なインベントリーは不可能です。フェデレーテッド・サーバーがどのシステム・サービスでも使用すると想定し、それに従って計画を立てるほうが賢明です。この要件は、ラッパーに含めることのできる任意の第三者ソフトウェアにも適用されます。正常な動作を保証できない場合は、一部の機能をその独自のプロセス・スペース内に分離する必要があります。

最初のステップは、可能な限りオペレーティング・システム・サービスの使用を最小限にとどめることです。 2 番目のステップは、ラッパー・ユーティリティーを通して提供されるシステム・サービス・ルーチンを使用することです。メモリー管理をこれらのユーティリティーを使用して実行することは特に重要です。

最後に、ラッパーは、システムの状態を正しく変更するサービスを使用するようにする必要があります。たとえば、ラッパーが信号ハンドラーを使用する場合は、ラッパー・ルーチンを呼び出すたびにそれらの信号ハンドラーをインストールして除去する必要があります。ラッパーがフェデレーテッド・サーバーに制御を戻すときに、その信号ハンドラーがアクティブのままであってはなりません。もう 1 つの例は、アラームの使用です。ラッパーは、フェデレーテッド・サーバーに戻るたびにアラーム状態をリストアしなければなりません。

システム・サービスに関する既知の制約事項は次のとおりです。

- メモリー管理は、提供されているユーティリティー関数とクラスを通して実行しなければなりません。
- stdout、stderr、stdin、cin、cout、cerr への入出力 (I/O) は機能しません。
- Windows の場合: ラッパーは getenv() ではなく Win32 ルーチンの GetEnvironmentVariable() を使用しなければなりません。
- ラッパーはトークン化、`strtok`、および `strtok_r` のための Unix および Windows のシステム・サービスを使用してはなりません。ラッパー・インターフェースには、トークン化のための代わりのシステム・サービスが備えられています。

メモリー管理 (C++ のみ)

すべてのメモリー管理は、ラッパー・ユーティリティーの allocate と deallocate メソッドによって実行する必要があります。これらのメソッドを使用する新規演算子と削除演算子を持つ基本のクラス Sqlqg_Base_Class が備えられており、Sqlqg_Base_Class から派生するクラスは正しく動作します。

トークン化サービス (C++ のみ)

Unix および Windows のサービス `strtok` および `strtok_r` は、区切り文字を基にしてストリングをいくつかの部品に分解するためのものです。ラッパー作成者は、これらを使用するよりも、ラッパー・ユーティリティー・クラスの `string_to_token()` メソッドを使用する必要があります。このメソッドは `strtok_r` とまったく同様に機能し、スレッド・セーフです。

以下のコード断片は、`Wrapper_Utilities::string_to_token` メソッドの使用法を示しています。

```
char*   string_to_scan = "this is a test string";
char*   scan_state = NULL;
char*   cur_token = NULL;

// Scan for the first token
   cur_token = Wrapper_Utilities::string_to_token (
     string_to_scan, " ", &scan_state);
while (cur_token != NULL)
{
   // Do something useful here with the token we found

   // Get the next token; note that we pass NULL for the string
   cur_token = Wrapper_Utilities::string_to_token (
      NULL, " ", &scan_state);
}
```

関連概念:

- 120 ページの『ラッパーの環境変数へのアクセス可能化』

関連タスク:

- 121 ページの『ラッパーの移植性』

ラッパーの環境変数へのアクセス可能化

フェデレーテッド・サーバーは、ラッパーが使用できる環境変数を制御します。ラッパーが環境変数にアクセスできるようにするためには、db2dj.ini 構成ファイルにその環境変数の値を指定する必要があります。 表 49 は、フェデレーテッド・サーバー上の構成ファイルがあるディレクトリーをプラットフォーム別に示しています。

表 49. フェデレーテッド・サーバー上の構成ファイルのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX®	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows®	%DB2PATH%/cfg

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2® Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

フェデレーテッド・サーバーは、始動時にそれを読み取ります。環境変数の値を変更する場合は、フェデレーテッド・サーバーを停止してから再始動する必要があります。

db2dj.ini の中の項目は、次の形式になります。

```
[white space]Variable[white space]=[white space]Value[white space][eoln]
```

`[white space]` はオプションです。変数名と変数値には、空白文字や行の終わり文字を含めることはできません。ファイルの最終行を含め、構成ファイル内の各行には、行の終わり文字が必要です。

ラッパーは、getenv() (Windows プラットフォームの場合は GetEnvironmentVariable()) を使用して環境変数にアクセスできます。

C++ コーディングに関する考慮事項

ラッパーによる、例外や Run Time Type Identification (RTTI) などの一部の C++ 機能の使用法に関連した制約事項があります。

ラッパーは C++ 例外を使用することができますが、例外がフェデレーテッド・サーバーに伝搬しないようにする必要があります。すべての例外をキャッチして、ラッパー・インターフェースでそれらを戻りコードに変えることができないと、フェデレーテッド・サーバー・エージェント・プロセス (あるいは場合によってはフェデレーテッド・サーバー・インスタンス全体) が異常終了します。

ラッパー・インターフェースは Run Time Type Identification (RTTI) をサポートしていません。このことは、動的キャストなどの他のいくつかの C++ 機能を使用するために、多くのコンパイラーが RTTI を必要としていても、それらの機能は使用できないことを意味します。

関連タスク:

- 119 ページの『ラッパーにおけるシステム・サービスの使用』
- 121 ページの『ラッパーの移植性』

ラッパーの移植性

Windows の場合、ラッパーは、Unix getenv() ルーチンではなく、Win32 GetEnvironmentVariable() ルーチンか GetEnvironmentStrings() ルーチンを使用して、環境変数にアクセスする必要があります。

関連概念:

- 120 ページの『ラッパーの環境変数へのアクセス可能化』

関連タスク:

- 119 ページの『ラッパーにおけるシステム・サービスの使用』

第 11 章 ラッパーの文書化

登録のためにサブミットする SQL ステートメントに何を指定するかは、大部分をラッパー開発者が決定するため、開発者には、これらのステートメントのコーディング方法をユーザーに知らせておくことをお勧めします。データ・ソースに関してユーザーは何を予想できるかについては、大部分は開発者がラッパーに付与する機能によって決定されるため、開発者には、データ・ソースを処理する方法をユーザーに知らせておくことをお勧めします。ユーザーに情報を提供する 1 つの方法は、ブックレットかオンライン・ファイルでコンパイルすることです。このトピックでは、そのようなコンパイルに含めるべき項目を提案し、これらの項目を扱う方法に関するヒントを与えます。

ラッパーを文書化するときには、以下の項目への対応を考慮してください。

データ・ソースで使用可能な情報

ユーザーが照会でどのような種類の情報を要求できるか理解できるように、以下の点を記述できます。

- ユーザーがデータ・ソースのデータの集合 (たとえば、表、データ・セット、またはスプレッドシート) から取り出せる情報の性質
- データ・ソースで使用可能なデータ集合のタイプ
- データ・ソースの関数が保管情報から派生させられる、取り出し可能な情報の性質

登録

登録について説明する際には、以下の情報を提供することを考慮してください。

- 登録する必要のある構成
- 登録を開始するときの SQL ステートメントの構文
- ステートメントにコーディングできるオプション
- オプションの許容値
- ステートメントのパラメーターの意味
- パラメーターの許容値
- データ・ソースのデータの集合と CREATE NICKNAME ステートメントのパラメーターの対応の仕組み
- 関数テンプレートによって表されるデータ・ソースの機能

助言

助言 (役に立つ簡単な情報) も含めることができます。要件、制限、推奨、ヒント、覚え書きなど、さまざまな種類があります。次のリストの例は、表構造ファイル・サーバーのラッパーに関する、IBM® の資料から取られたものです。

要件の例

- 『列区切り文字はファイル全体で一貫している必要がある』
- 『表構造ファイルのニックネームの統計は、SYSSTAT ビューを更新することにより手動で更新する必要がある』

制限の例

- 『ラッパーではパススルー・セッションは許可されていない』
- 『ファイルは 1 行で 1 つのレコードに制限されている』

ヒントの例

- 『システムは、ソートされていないファイルよりも、ソートされたデータ・ファイルの方がはるかに効率的に検索できる』
- 『ソートされたファイルの場合、かぎとなる列の値か範囲を指定することにより、パフォーマンスを向上させることができる』

エラーおよび警告

ラッパーが生じる可能性のある問題に対してエラーまたは警告を報告するときのメッセージについて説明してください。ユーザーがこれらのメッセージの説明を探せるように、関連付けられた SQLCODE、戻りコード (あれば)、および SQLSTATE 値 (あれば) を含めてください。

ユーザーには、実例やステップ・バイ・ステップの説明も大いに役立ちます。

ラッパー説明の例については、「*DB2® Information Integrator データ・ソース構成ガイド*」を参照してください。

関連タスク:

- 127 ページの『ラッパーのコンパイル (C++)』
- 131 ページの『第 13 章 ラッパーのリンク (C++ のみ)』
- 135 ページの『第 14 章 ラッパーのインストール』
- 147 ページの『有効なオプションと無効なオプションを使用したラッパーのテスト』
- 128 ページの『ラッパーのコンパイル (Java)』

第 4 部 ラッパーの作成、テスト、およびトレース

本書のこの部では、ラッパーを作成、テスト、およびトレースするために必要な次のようなタスクを順に説明します。

- デプロイメントを目的としたラッパーの構築およびパッケージ化
- 設計どおりの作動の確認のためのラッパーのテスト

第 12 章 ラッパーのコンパイル

ラッパーのコンパイル (C++)

ラッパー・コードをコンパイルするときには、コンパイラーは、ラッパー開発キットでインストールされるラッパー・インターフェース・ヘッダー・ファイルにアクセスできなければなりません。これらのファイルは、他の必要なヘッダー・ファイルと共に、フェデレーテッド・サーバーの include サブディレクトリーにあります。

AIX でのコンパイル

AIX でラッパー・コードをコンパイルするには、xlC_r7 プログラムを使用します。以下の例は、サンプルのラッパーから 1 つのファイルをコンパイルするときに使用するコマンドを示しています。

```
/usr/ibmcxx/bin/xlC_r7 -qlanglvl=ansi -qflag=i:i -qmaxmem=-1
-M -qnoansialias -qnotempinc -DSQLUNIX -g -qnamemangling=v5 -qlonglong
-I/home/inst/sqllib/include
-c sample_wrapper.C -o sample_wrapper.o
```

VisualAge 6.0 (あるいはそれ以降) のコンパイラーを使用している場合には、`-qnamemangling=v5` および `-qlonglong` オプションを使用してください。

詳細は、ラッパー開発キットに備えられている Makefile を参照してください。 表 50 は、ラッパー開発キットが格納されているサブディレクトリーをプラットフォーム別に示しています。

表 50. プラットフォーム別のラッパー開発キットのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk
Windows	%DB2PATH%¥samples¥wrapper_sdk

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

Windows でのコンパイル

Windows 上でラッパー・コードをコンパイルするには、CL プログラムを使用します。以下の例は、サンプルのラッパーから 1 つのファイルをコンパイルするときに使用するコマンドを示しています。

```
C:¥VC98¥bin¥CL -c /nologo -Zi -GZ -W3 -DNULL=0
-DWIN32 -D_X86_=1 -D_CRTAPI1=__cdecl
-D_CRTAPI2=__cdecl -D_MT -D_DLL -J -Od
-IC:¥VC98¥include
-IC:¥sqllib¥include
sample_wrapper.C /Tp -Fosample_wrapper.obj
```

関連タスク:

- 135 ページの『第 14 章 ラッパーのインストール』
- 147 ページの『有効なオプションと無効なオプションを使用したラッパーのテスト』
- 128 ページの『ラッパーのコンパイル (Java)』

ラッパーのコンパイル (Java)

ラッパーのコンパイルを行うステップは、ラッパーが作成される言語により異なります。このトピックは、Java で作成されるラッパーをコンパイルする方法を説明します。

前提条件:

Java 開発キット (JDK) バージョン 1.3 以降が必要です。特に JDK に組み込まれている Java コンパイラーが必要です。 Java コンパイラーは Windows では javac.exe、UNIX では javac です。

クラス・パスに db2qgjava.jar を含める必要があります。 db2qgjava.jar は、ラッパー API クラスを含む Java アーカイブです。それをシステムの CLASSPATH に追加するか、コンパイル時に `-classpath` オプションを使用します。 表 51 は、db2qgjava.jar ファイルが置かれているディレクトリーをプラットフォーム別に示しています。

表 51. プラットフォーム別の Java 構成ファイルのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/java/db2qgjava.jar
HP/Sun/Linux	/opt/IBM/db2/V8.1/java/db2qgjava.jar
Windows	%DB2PATH%¥java¥db2qgjava.jar

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

手順:

Java ラッパーをコンパイルするには、以下のようにします。

1. クラス・パスに db2qgjava.jar があることを確認します。
2. カスタム・ラッパー・クラスを、他の Java クラスと同じ方法でコンパイルします。

   ```
   javac @wrapperfiles
   ```

   javac コンパイラーでは、以下のようなさまざまなオプションを指定することができます。

   - コンパイルする Java ソース・ファイル。 Java ソース・ファイルには、それらが含むクラス定義の名前が含まれているため、ラッパー用のソース・ファイルには通常、UnfencedXXWrapper.java、 UnfencedXXServer.java、

FencedXXWrapper.java、 FencedXXServer.java、 FencedXXUser.java などのファイルが組み込まれています。ここで、XX はラッパーの名前です。

- コンパイルする Java ソース・ファイルのロケーション。 Java ソース・ファイルが作業ディレクトリーまたはクラス・パスにない場合には、-sourcepath オプションを使用して javac コマンドでそのロケーションを指定できます。
- CLASSPATH への追加。 db2qgjava.jar を CLASSPATH 環境変数に組み込まなかった場合には、-classpath オプションを使用して javac コマンドにそれを組み込んでください。
- コンパイルされた .class ファイルの宛先ディレクトリー。

たとえば、DB2 CLP ウィンドウから以下のコマンドを作業ディレクトリー内で実行することにより、db2qgjava.jar ファイルをクラス・パスに追加したり、サンプル Java ラッパーをコンパイルしたり、コンパイルしたクラスを %DB2PATH%¥function にインストールしたりできます。

Windows

```
javac -classpath %CLASSPATH%;%DB2PATH%¥java¥db2qgjava.jar
-d %DB2PATH%¥function UnfencedFileWrapper.java
UnfencedFileServer.java UnfencedFileNickname.java
FencedFileWrapper.java FencedFileServer.java FencedFileNickname.java
FileConnection.java FileQuery.java FileExecDesc.java
```

UNIX

```
javac -classpath $CLASSPATH:$DB2PATH/java/db2qgjava.jar
-d %DB2PATH%/function UnfencedFileWrapper.java
UnfencedFileServer.java UnfencedFileNickname.java
FencedFileWrapper.java FencedFileServer.java FencedFileNickname.java
FileConnection.java FileQuery.java FileExecDesc.java
```

関連タスク:

- 127 ページの『ラッパーのコンパイル (C++)』
- 135 ページの『第 14 章 ラッパーのインストール』
- 147 ページの『有効なオプションと無効なオプションを使用したラッパーのテスト』

第 13 章 ラッパーのリンク (C++ のみ)

ラッパーは、3 つの独立したコード・ライブラリーで構成されています。最上位のライブラリー、fenced ライブラリー、および unfenced ライブラリーです。最上位ライブラリー用の共有ライブラリーは、ラッパー開発キットの一部として提供されます。残りの 2 つのライブラリーは、ラッパー・コードから作成する必要があります。

ラッパーの 3 つのライブラリーは、実行可能コードの共有ライブラリーのプラットフォーム規則に準拠し、3 つのライブラリーを関連付ける命名規則に準拠していなければなりません。以下の例は、このことを示しています。

AIX では、サンプル・ラッパーの最上位ライブラリーは、「libsample.a」と名付けられます。「lib」接頭部と「.a」接尾部は、共有ライブラリーのプラットフォーム規則です。 unfenced ラッパー・ライブラリーは、「libsampleU.a」と名付けられます。これは、最上位ライブラリーと同じですが、接尾部の直前に大文字の「U」が追加されています。 fenced ラッパー・ライブラリーは、「libsampleF.a」と名付けられます。

Windows では、サンプル・ラッパー・ライブラリーは、「db2sample.dll」、「db2sampleU.dll」、および「db2sampleF.dll」です。

次の表は、各プラットフォームの共有ライブラリーの接尾部をリストしています。

表 52. プラットフォームおよびサポートされる共有ライブラリーの接尾部

プラットフォーム	ライブラリーの接尾部
AIX	.a
HP-UX	.sl
Linux	.so
Sun	.so
Windows	.dll

unfenced ラッパー・ライブラリーには、unfenced ラッパー・フック・ルーチン UnfencedWrapper_Hook() と、以下のサブクラスのコードすべてが含まれている必要があります。

- Unfenced_Generic_Wrapper
- Unfenced_Generic_Server
- Unfenced_Generic_User
- Unfenced_Generic_Nickname

fenced ラッパー・ライブラリーには、fenced ラッパー・フック・ルーチン FencedWrapper_Hook() と、以下のサブクラスのコードすべてが含まれている必要があります。

- Fenced_Generic_Wrapper
- Fenced_Generic_Server

- Fenced_Generic_User
- Fenced_Generic_Nickname
- Remote_Connection
- Remote_Query
- Remote_Passthru

特定のクラスを使用しない場合 (たとえば、ラッパーが Fenced_Generic_User クラスのデフォルト・インプリメンテーションを使用する場合)、それらのクラスをライブラリーにリンクするためのコードはありません。

AIX でのリンク

最上位ライブラリーは、DB2 Information Integrator インストール・システムの /usr/opt/db2_08_01/lib ディレクトリーにあり、`libdb2sqqgtop.a` という名前が付けられています。このライブラリーはすでにリンクされているため、コピーしてラッパー用に名前変更するだけです。

fenced および unfenced ライブラリーは、エクスポートされたシンボルのファイルとリンクさせる必要があります。このファイルを使用することにより、リンク・プログラムは、ライブラリーが実行時にフェデレーテッド・サーバー環境へロードされるときに、使用可能なシンボルを解決できるようになります。エクスポートされたシンボル・ファイルは、フェデレーテッド・サーバーの /usr/opt/db2_08_01/lib ディレクトリーにあり、udbwrapper.exp という名前が付けられています。

AIX で fenced および unfenced ライブラリーをリンクする場合、makeC++SharedLib_r ツールを使用します。次に示すのは、このツールを使用して unfenced ラッパー・ライブラリーをリンクする一例です。

```
/usr/lpp/xlC/bin/makeC++SharedLib_r -p 2048
-I /home/inst/sqllib/lib/udbwrapper.exp
-n UnfencedWrapper_Hook
-lpthreads -lc -lc_r
sample_wrapper.o sample_server.o sample_nickname.o
-o libsampleU.a
```

ラッパー開発キットには、詳細な Makefile が備えられています。 Makefile は /usr/opt/db2_08_01/samples/wrapper_sdk サブディレクトリーにあります。

Windows でのリンク

最上位ライブラリーは、ラッパー開発キットの %DB2PATH%¥bin ディレクトリーにあり、`db2sqqgtop.dll` という名前が付けられています。このライブラリーはすでにリンクされているため、コピーしてラッパー用に名前変更するだけです。

fenced および unfenced ライブラリーは、エクスポートされたシンボルのファイルとリンクさせる必要があります。このファイルを使用することにより、リンク・プログラムは、ライブラリーが実行時にフェデレーテッド・サーバー環境へロードされるときに、使用可能なシンボルを解決できるようになります。エクスポートされたシンボル・ファイルは、フェデレーテッド・サーバーの %DB2PATH%¥bin ディレクトリーにあります。 2 つのファイル db2qgstp.lib および db2qg.lib があります。トラステッド側のライブラリーは db2qg.lib にリンクされ、fenced 側のライブラリーは db2qgstp.lib にリンクされる必要があります。

Windows で fenced および unfenced ライブラリーをリンクする場合、「link」ツールを使用します。次に示すのは、このツールを使用して unfenced ラッパー・ライブラリーをリンクする一例です。

```
C:¥VC98¥bin¥link /OUT:db2sampleU.dll
C:¥sqllib¥lib¥db2qg.lib
/DLL /NODEFAULTLIB /INCREMENTAL:NO
/MACHINE:i386 /SUBSYSTEM:CONSOLE
sample_wrapper.obj sample_server.obj sample_nickname.obj
```

ラッパー開発キットには、詳細な Makefile が備えられています。 Makefile は %DB2PATH%¥samples¥wrapper_sdk ディレクトリーにあります。

関連タスク:

- 127 ページの『ラッパーのコンパイル (C++)』
- 135 ページの『第 14 章 ラッパーのインストール』
- 128 ページの『ラッパーのコンパイル (Java)』

第 14 章 ラッパーのインストール

カスタム・ラッパーを使用する前に、それをフェデレーテッド・サーバーにインストールする必要があります。ラッパーのインストールの詳細は、ラッパーが開発された言語により異なります。

C++ ラッパーのインストール

C++ ラッパーをインストールするには、以下のようにします。

1. フェデレーテッド・サーバーを停止します。 C++ ラッパー・ライブラリーをインストール、置換、または削除する前に、DB2 Universal Database フェデレーテッド・サーバー・インスタンスを停止する必要があります。 DB2 UDB の稼働中にこれらのライブラリーを管理すると、DB2 UDB が破損する可能性があります。
2. ラッパーをリンクすることにより作成された 3 つの C++ ラッパー・ライブラリー、つまり共有最上位ライブラリー (ラッパー開発キットに含まれる)、fenced ライブラリー、および unfenced ライブラリーを見つけます。
3. これらのライブラリーを DB2 UDB インストール・システムの適切なディレクトリーにインストールします。 表 53 は、それぞれのプラットフォームにどのディレクトリーが適用されるかを示しています。

表 53. プラットフォーム別の C++ ラッパーのインストール・ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/lib
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib
Windows	%DB2PATH%¥bin

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

Java ラッパーのインストール

Java ラッパーは以下の 2 つの形式のいずれかで配信できます。

- .class ファイルのセット。これは Java ラッパーをコンパイルすることにより作成します。一般に開発者は、開発とテストの時にのみ .class ファイルを使用してラッパーをインストールします。
- 単一の .jar ファイル。これは Java アーカイブ・ツール (Java 開発キットに組み込まれている) を使用して .class ファイルから作成します。 .jar ファイルはインストールが容易であるため、一般に開発者は、.jar ファイルを使用して Java アプリケーションを配信します。さらに単一の .jar ファイルのインストールは、欠落ファイルの可能性を減らします。

.class ファイルのインストール

ラッパーを .class ファイルのセットとしてインストールするには、以下のようにします。

1. ラッパーのすべての .class ファイルを、開発システムからフェデレーテッド・サーバーにコピーします。インストール・プロセス時に CLASSPATH を更新するか、既存の .jar および .class ファイルを置換した場合、新しくインストールしたラッパー・ファイルを使用する前に DB2 UDB を再始動することが必要な場合もあります。インストールを容易にするために、プラットフォームに応じてファイルを以下のディレクトリーにインストールします。 表 54 は、それぞれのプラットフォームにどのディレクトリーが適用されるかを示しています。

表 54. プラットフォーム別の Java ラッパーのインストール・ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/lib
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib
Windows	%DB2PATH%¥bin

   デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

2. ラッパーの .class ファイルがあるディレクトリーを、システムの CLASSPATH 環境変数に組み込みます。 CLASSPATH 環境変数には、通常 SQLLIB¥function ディレクトリーがすでに含まれています。
3. ラッパーが使用するすべての Java クラスを、システムの CLASSPATH 環境変数に組み込みます。

.jar ファイルのインストール

ラッパーを .jar ファイルとしてインストールするには、以下のようにします。

1. ラッパーの .jar ファイルを、開発システムからフェデレーテッド・サーバーにコピーします。
2. フェデレーテッド・サーバーがラッパーの .jar ファイルを見つけることができるようにシステムを構成します。これを行うには、2 つの方法があります。
   - ラッパーの .jar ファイルのパスと名前を、システムの CLASSPATH 環境変数に組み込む。
   - DB2 Universal Database のストアード・プロシージャー `SQLJ.install_jar` を使用して、ラッパーの .jar ファイルをフェデレーテッド・サーバーに登録する。 `SQLJ.install_jar` を使用する場合、DB2 Universal Database は .jar ファイルのコピーを作成します。 .jar ファイルを更新する場合には、DB2 Universal Database がその .jar ファイルのコピーを置き換えるようにしなければなりません。追加情報については、DB2 Universal Database 資料を参照してください。
3. ラッパーが使用するすべての Java クラスを、システムの CLASSPATH 環境変数に組み込みます。

Java のメモリー・サイズ設定が十分であることの確認

DB2 Universal Database 用の JAVA_HEAP_SZ DBM 環境変数を変更することにより、Java のメモリー・サイズ設定が十分であるようにします。推奨される最低のサイズは、ラッパー開発キットで提供されているサンプル・ラッパーなどの単純なラッパーでは、1024 です。 JAVA_HEAP_SZ 変数のメジャー単位は 4 KB です。最適の値はラッパーに応じて異なります。つまり、ロードされるクラスの数、作成されるオブジェクトの数、および Java ラッパーを使用する同時接続の数に左右されます。 500 から 1000 のクラスをロードするラッパーでは、2048 の大きさの設定が必要となるかもしれません。

JAVA_HEAP_SZ 変数の設定の追加情報については、DB2 Universal Database 資料を参照してください。

関連タスク:

- 140 ページの『「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードのインストール』
- 143 ページの『XML 構成ファイルのインストール』
- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』

第 15 章 コントロール・センターへのデータ・ソースの追加

DB2 コントロール・センターへのデータ・ソースの追加

カスタム・ラッパーを DB2 コントロール・センターにデータ・ソースとして追加すると、ユーザーはそれを使用できるようになります。その後ユーザーはそのデータ・ソース (ラッパー) を選択してオプションを指定できるようになります。

XML 構成ファイルは、データ・ソースとそれらのオプションを DB2 コントロール・センターに記述するために使用されます。 DB2 コントロール・センターが始動する時、XML 構成ファイルが存在するすべてのデータ・ソースをロードします。ユーザーがデータ・ソースを選択するとき、コントロール・センターは、そのデータ・ソースの XML 構成ファイルで定義されているオプションを表示します。

ラッパー開発キットには、XML 構成ファイルの作成を支援するツールが含まれています。

前提条件:

ラッパー開発キットが必要です。

手順:

カスタム・データ・ソースを DB2 コントロール・センターに追加するには、以下のようにします。

1. ラッパーの XML 構成ファイルおよび関連プロパティー・ファイルを作成するには、「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードを使用します。
2. XML 構成ファイルをインストールします。 表 55 は、フェデレーテッド・サーバー上のどのディレクトリーにそのファイルをインストールするかをプラットフォーム別に示しています。

表 55. プラットフォーム別の XML 構成ファイルのインストール・ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows	%DB2PATH%/cfg

表 56 は、関連プロパティー・ファイルがどこにあるかを示しています。

表 56. プラットフォーム別の XML 構成ファイルのインストール・ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/tools/en_US/wrapper_cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/tools/en_US/wrapper_cfg
Windows	%DB2PATH%¥tools¥en_US¥wrapper_cfg

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

3. ユーザーがカスタム・データ・ソースのニックネーム、ビュー、およびサーバー・ディスカバリー情報をディスカバーできるように、DB2 コントロール・センターのディスカバリー機能をサポートしたい場合には、フェデレーテッド・サーバーに必要なディスカバリー・サポートをインストールしてください。組み込みディスカバリー・ツールを使用することを選択する場合には、カスタム・ディスカバリー・ストアード・プロシージャーのみをインストールする必要があります。 (グラフィカル・ディスカバリー・ツールは、すでに DB2 コントロール・センターでインストールされています。)
4. DB2 コントロール・センターを再始動します。

関連概念:

- 8 ページの『フェデレーテッド・システムにデータ・ソースを追加する方法』

関連タスク:

- 140 ページの『「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードのインストール』
- 141 ページの『XML 構成ファイルの作成』
- 143 ページの『XML 構成ファイルのインストール』

「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードのインストール

「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードは、ラッパー (データ・ソース) を DB2 コントロール・センターに追加するために必要な構成ファイルを作成するために使用されます。

このウィザードは、DB2 Information Integrator ラッパー開発キットでインストールされます。 表 57 は、このウィザードがご使用のシステムにインストール済みでかどうかを判別するために、どこを調べればよいかを示しています。このウィザードのインストール・ファイルは db2qgjava.jar です。

表 57. プラットフォーム別の XML 構成ファイル・ウィザードのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%¥bin¥db2wrapperconfig.bat

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

関連タスク:

- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』

- 「*IBM DB2 Information Integrator インストール・ガイド*」の『ラッパー開発キットのインストール』

XML 構成ファイルの作成

コントロール・センターは、XML 構成ファイルを読み取り、ユーザーに提供するデータ・ソース (ラッパー) およびオプションを判別します。カスタム・ラッパーを DB2 コントロール・センターに追加するには、ラッパーの XML 構成ファイルを作成する必要があります。 XML 構成ファイルを作成するには、「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードを使用します。

前提条件:

ラッパー開発キットがインストール済みであることが必要です。

Java ランタイム環境 (JRE) がインストール済みであることが必要です。 (このウィザードは Java アプリケーションです。)

手順:

XML 構成ファイルを作成するには、以下のようにします。

1. 「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードを開始します。 表 58 は、どのディレクトリーからウィザードを開始するかを示しています。

表 58. プラットフォーム別のウィザードを開始するディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%¥bin¥db2wrapperconfig.bat

   デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

2. 新しい構成ファイルを作成するか、それとも既存のものを変更するかを選択します。
3. ウィザードにより規定されたステップに従います。以下を入力するようにウィザードからプロンプトが出されます。
   - ファイル情報。これには、ラッパー、DTD パス、および XML ファイル・パスを識別するために DB2 コントロール・センターで使用される名前が含まれます。
   - ラッパー情報。これには、ラッパーのデータ・ソースを識別するために DB2 コントロール・センターで使用される名前と記述が含まれます。ラッパー情報には、サポートされるオペレーティング・システム、および各オペレーティング・システムで使用するためのラッパー・ライブラリー名またはクラス名も含まれます。

- ユーザーがラッパーに供給する必要があるラッパー・オプション。オプションごとに、許可される値、デフォルト値、および記述を指定できます。オプションが必要か、ラッパーの作成中にユーザーがそれを編集できるか、およびラッパーの作成後にユーザーがそれを変更できるかについても指定できます。さらに、CREATE SERVER ステートメントにユーザー ID が必要であるかなどの SQL 要件も指定できます。
- ユーザーがラッパーに供給する必要があるサーバー・オプション。
- ユーザーがラッパーに供給する必要があるユーザー・マッピング・オプション。
- ユーザーがラッパーに供給する必要があるニックネーム・オプション。さらに、ユーザーが CREATE NICKNAME ステートメントで列定義を指定できるかどうかも指定できます。
- ユーザーがラッパーに供給する必要がある列オプション。
- ラッパーがサポートする DB2 データ・タイプ。データ・タイプはサーバーごとに指定できます。
- DB2 コントロール・センターがこのラッパーにおけるディスカバリー機能をサポートするかどうか。ディスカバリー機能のサポートを選択する場合には、実行すべき追加のステップがあります。
- ユーザーが供給する必要がある環境変数およびそれらが設定されるロケーション。
- 関数テンプレート、ユーザー定義関数、およびこのラッパーが必要な関数マッピング。

ヘルプと infopop がすべてのページで使用可能です。

結果:

ウィザードでは、以下の 2 つのファイルが作成されます。

- XML 構成ファイル。これは、ウィザードで指定した情報を含んでいます。 XML 構成ファイルは、ウィザードの 2 番目のページで指定したディレクトリーに置かれます。ファイル名はラッパー名を基にしています。たとえば、ラッパー名が `GeoDataSource` の場合には、XML ファイル名は GeoDataSource.xml となります。
- プロパティー・ファイル。これは、ラッパーおよびそのオプションのために DB2 コントロール・センターで表示されるリテラル・テキスト・ストリングを含んでいます。テキスト・ストリングをプロパティー・ファイルに外部化すると、複数の言語を容易にサポートできるようになります。複数のプロパティー・ファイルに対して 1 つの構成ファイルを使用できます。 XML 構成ファイルは、テキスト・ストリングの代わりにシンボル名を含んでいます。コントロール・センターは、ラッパー名とユーザーの言語に対応するこれらのシンボル名をプロパティー・ファイル内で検索します。プロパティー・ファイルは、XML 構成ファイルと同じディレクトリーで作成され、ラッパー名を基にした名前を持っています。たとえば、ラッパー名が `GeoDataSource` の場合には、プロパティー・ファイル名は GeoDataSource.properties となります。

143 ページの表 59 は、両方のファイルのサンプルが cc_plugin ディレクトリーのどこにあるかを示しています。

表 59. プラットフォーム別のサンプルのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk/cc_plugin
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk/cc_plugin
Windows	%DB2PATH%¥samples¥wrapper_sdk¥cc_plugin

関連概念:

- 36 ページの『ラッパー・オプションの決定』
- 37 ページの『サーバー・オプションの決定』
- 31 ページの『ニックネームおよび列オプションの決定』
- 38 ページの『ユーザー・マッピング・オプションの決定』
- 8 ページの『フェデレーテッド・システムにデータ・ソースを追加する方法』

関連タスク:

- 41 ページの『データ・ソースが受け入れ可能なヘッド式の判別』
- 41 ページの『データ・ソースが受け入れ可能な述部の判別』
- 42 ページの『データ・ソースが受け入れ可能な結合の判別』
- 42 ページの『データ・ソースが受け入れ可能な関数の判別』
- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』

XML 構成ファイルのインストール

DB2 コントロール・センターは、XML 構成ファイルを読み取ることにより、どのラッパー (データ・ソース) およびオプションをユーザーに提示するかを決定します。ラッパー用の XML 構成ファイルの作成後に、フェデレーテッド・サーバー上にそれらのファイルをインストールする必要があります。DB2 コントロール・センターはそれらのファイルをそこからロードします。データ・ソースを DB2 コントロール・センターに追加するには、データ・ソースの XML 構成ファイルをフェデレーテッド・サーバーにインストールする必要があります。

前提条件:

ラッパーの XML 構成ファイルおよびプロパティー・ファイルのセットにアクセスできることが必要です。

フェデレーテッド・サーバーにアクセスできることが必要です。

手順:

ラッパーの XML 構成ファイルをインストールするには、以下のようにします。

1. ラッパー XML 構成ファイルおよび関連したプロパティー・ファイルをフェデレーテッド・サーバーに転送します。
2. XML 構成ファイル (wrappername.xml) を移動させます。 144 ページの表 60 は、ファイルの移動先ディレクトリーを示しています。

表 60. プラットフォーム別の XML 構成ファイルの移動先ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows	%DB2PATH%¥cfg

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

3. 各プロパティー・ファイル (wrappername.properties) を (wrapper_cfg) ディレクトリーに移動させます。たとえば、ラッパー固有の追加機能の米国英語でのサポートを DB2 コントロール・センターに提供する場合には、そのラッパーの米国英語プロパティー・ファイルを移動させてください。 表 61 は、ファイルの移動先ディレクトリーを示しています。

表 61. プラットフォーム別のプロパティー・ファイルの移動先ディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/tools/en_US/wrapper_cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/tools/en_US/wrapper_cfg
Windows	%DB2PATH%¥Tools¥en_US¥wrapper_cfg

4. DB2 コントロール・センターを再始動します。

DB2 コントロール・センターが始動する時、XML 構成ファイルとプロパティー・ファイルがインストールされているすべてのデータ・ソースのオプションをロードします。データ・ソースは、標準の IBM データ・ソースと共にオプションとして提示されます。

関連概念:

- 8 ページの『フェデレーテッド・システムにデータ・ソースを追加する方法』

関連タスク:

- 135 ページの『第 14 章 ラッパーのインストール』
- 141 ページの『XML 構成ファイルの作成』

DB2 コントロール・センターでのディスカバリーのサポート

DB2 コントロール・センターは、サーバー、ビュー、ニックネームなどのデータ・ソースの機能を発見できます。カスタム・ラッパーを開発するときに、DB2 コントロール・センターのディスカバリー機能のサポートを提供するように選択することができます。ディスカバリーのサポートを提供する場合、ユーザーは、カスタム・データ・ソース用のニックネーム、ビュー、およびサーバー情報を、標準データ・ソースの場合と同様の方法でディスカバーできます。

前提条件:

開発システムでは、ラッパー開発キットが必要です。

フェデレーテッド・サーバーでは、Java ランタイム環境 (JRE) が必要です。

手順:

カスタム・ラッパー (データ・ソース) のディスカバリーをサポートするには、以下のようにします。

1. ラッパーで使用するためのディスカバリー機能とオプションを指定する XML 構成ファイルを作成します。
   a. 「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードを開始します。
   b. そのラッパー用の XML 構成ファイルが存在する場合には、そのファイルを変更します。 XML 構成ファイルが存在しない場合には、新規に作成してください。
   c. ウィザードの**「ディスカバリー機能要件の指定 (Specify the discover function requirements)」**ページで、発見機能をサポートするための 2 つのオプションのいずれかを選択します。
      - **「組み込み Java コンクリート・クラスを使用してディスカバリー機能をサポートする (Support the discover function using a built-in concrete Java class)」**を選択すると、DB2 コントロール・センターおよびラッパー開発キットでインストールされる単純な Java ディスカバリー・ツールを使用します。 (DB2 コントロール・センターのインストール・システムでは、そのツールは Java ファイル db2WrapperDiscovery.jar として提供されます。ラッパー開発キットでは、それは db2WrapperDiscoverySDK.jar という名前です。これら 2 つのファイルの内容は同一です。)
      - **「カスタム Java クラスを使用してディスカバリー機能をサポートする (Support the discover function using a custom Java class)」**を選択すると、ユーザー自身が作成したカスタム Java クラスを使用します。
   d. 同じページで、ディスカバリー・クラス用の任意のカスタム・オプションまたはユーザー・インターフェース・オプションを追加します。
   e. ウィザードを終了します。
2. 組み込み Java コンクリート・クラスを選択した場合には、ラッパーのデータ・ソースから発見したい情報を戻すストアード・プロシージャーを作成する必要があります。 表 62 は、例として使用できる Java ストアード・プロシージャーが組み込まれているラッパー開発キットのディレクトリーを示しています。

表 62. プラットフォーム別の Java ストアード・プロシージャーのディレクトリー

プラットフォーム	ラッパーのインストール・ディレクトリー
AIX	/usr/opt/db2_08_01/samples/wrapper_sdk/cc_plugin/sample.java
HP/Sun/Linux	/opt/IBM/db2/V8.1/samples/wrapper_sdk/cc_plugin/sample.java
Windows	%DB2PATH%¥samples¥wrapper_sdk¥cc_plugin¥sample.java

デフォルトの Windows ディレクトリー・パスは C:¥Program Files¥IBM¥SQLLIB です。 %DB2PATH% は、Windows 上で DB2 Information Integrator がインストールされているディレクトリー・パスを指定するために使用される環境変数です。

サンプルのストアード・プロシージャーは、組み込みディスカバリーが Java でどのように機能するかを示す例です。 C ストアード・プロシージャー、SQL ストアード・プロシージャー、またはその他の任意のストアード・プロシージャーを使用してディスカバリー機能をインプリメントできます。 cc_plugin で提供されているサンプルからカスタム・ストアード・プロシージャーを作成するには、以下のようにします。

a. sample.java をガイドとして使用して、データ・ソースで発見したい情報を戻すストアード・プロシージャーを作成します。
b. サンプルの Makefile を更新し、それがユーザーの Java ストアード・プロシージャーを参照するようにします。
c. Makefile を使用して Java ストアード・プロシージャーをコンパイルします。
d. sample.db2 スクリプトを更新して、それがストアード・プロシージャーを、フェデレーテッド・サーバーである DB2 Universal Database インスタンスにロードするようにします。
e. 更新した sample.db2 スクリプトを実行して、ストアード・プロシージャーをフェデレーテッド・サーバーにインストールします。

3. カスタム Java クラスを選択した場合には、そのクラスとそれが必要とするサポートするストアード・プロシージャーを供給する必要があります。カスタム Java クラスとその他の必要なコンポーネントをフェデレーテッド・サーバーにインストールします。
4. XML 構成ファイルをフェデレーテッド・サーバーにインストールします。
5. DB2 コマンド・センターを再始動します。

結果:

ラッパー、ラッパーの XML 構成ファイル、およびストアード・プロシージャーのインストール終了後、DB2 コントロール・センターのユーザーは、カスタム・データ・ソースをオプションとして扱うようになり、そのデータ・ソースに対してディスカバリーを実行できるようになります。 DB2 コントロール・センターは XML 構成ファイルを読み取り、このデータ・ソースに対するディスカバリーをサポートするかを判別します。サポートする場合には、ディスカバリー・ツールとしてどの Java クラスを使用するかを判別します。

組み込み Java コンクリート・クラスをディスカバリーのために選択する場合には、他のディスカバリー・ツールをインストールする必要はありません。 DB2 コントロール・センターは、ディスカバリーを実行するために、組み込み Java ツールをカスタム・ストアード・プロシージャーと組み合わせて使用します。

関連タスク:

- 140 ページの『「XML 構成ファイルの開発 (Develop XML Configuration File)」ウィザードのインストール』
- 141 ページの『XML 構成ファイルの作成』
- 143 ページの『XML 構成ファイルのインストール』
- 139 ページの『DB2 コントロール・センターへのデータ・ソースの追加』

第 16 章 ラッパーのテスト

以下のセクションでは、ラッパーをテストする方法を説明します。

登録 DLL ステートメントを使用したラッパーのテスト

ラッパーをテストするときには、CREATE SERVER および ALTER SERVER などの SQL ステートメントを使用できます。たとえば、ラッパー基本クラスのインプリメンテーションをテストするには、CREATE WRAPPER を実行できます。 Server 基本クラスのインプリメンテーションをテストするには、CREATE SERVER を実行できます。

テストおよび登録の目的で SQL ステートメントに指定する情報の決定は、ユーザーに任されています。そのような決定をする場合、これらのステートメントのパラメーターを理解している必要があります。このことについて理解するには、DB2® Universal Database バージョン 8 の「*SQL リファレンス* 」を参照してください。

関連タスク:

- 147 ページの『有効なオプションと無効なオプションを使用したラッパーのテスト』
- 150 ページの『ラッパーからのトレース情報の作成』

有効なオプションと無効なオプションを使用したラッパーのテスト

以下の条件で、オプション情報をテストすることをお勧めします。

- 指定したオプションは、このオプションを含む SQL ステートメントで有効でなければなりません。そうでない場合、ラッパーは SQLCODE SQL1881N を戻さなければなりません。
- オプションに設定する値はこのオプションに有効でなければなりません。そうでない場合、ラッパーは SQLCODE SQL1882N を戻さなければなりません。

  場合によっては、他のオプション値と矛盾するために、あるオプション値が無効になることがあります。たとえば、データ・ソースは照会結果をエンコードしますが、このときに、ユーザーは ENCRYPT というオプションに Y (はい、この結果セットを暗号化します) を設定し、ENCRYPTION_KEY というオプションに暗号化プログラムで必要なシンボルのストリングを設定する場合を考えてください。 ENCRYPT に N (いいえ、この結果セットを暗号化しません) を設定すると、シンボル (またはその問題に関係するすべての値) のストリングは、ENCRYPTION_KEY では無効になることは明らかです。
- SQL ステートメントで必須のオプションは、そのステートメントに指定する必要があります。必須オプションが欠落している場合、ラッパーは SQLCODE SQL1883N を戻す必要があります。
- オプションは同じ SQL ステートメントに 1 度だけ指定するようにします。 2 回以上指定する場合、DB2 は SQLCODE SQL1884N を戻します。

- ユーザーが ALTER [CONSTRUCT] ステートメント (たとえば、ALTER SERVER) を使用して、すでに定義されたオプションを追加する場合、DB2 は SQLCODE SQL1885N を戻します。
- ユーザーは、オプションに特定の値がすでに設定されている場合にのみ、ALTER [CONSTRUCT] ステートメント (たとえば、ALTER SERVER) を使用して、そのオプションの値を更新または削除できます。そのオプションに値が設定されていない場合、DB2 は SQLCODE SQL1886N を戻します。

関連概念:

- 147 ページの『登録 DLL ステートメントを使用したラッパーのテスト』

関連タスク:

- 150 ページの『ラッパーからのトレース情報の作成』

第 17 章 ラッパーのトレース

ラッパー・トレース機能は、ラッパーからの制御フローおよびトラブルシューティング情報をログに記録します。以下のセクションでは、ラッパー・トレース機能およびラッパーからのトレース情報の作成方法について説明します。

ラッパー・トレース機能

ラッパー・トレース機能は、関数のエントリー・ポイントや関数のエグジット・ポイントなど、開発するラッパーからの制御フロー情報を記録します。この情報を使用して、ラッパーの開発をしているときに問題となりそうな点を識別したり、ラッパーをデプロイした後に起こる問題を解決したりすることができます。

このトラブルシューティング情報を取得するには、ラッパーで指定する領域からトレース・メンバー関数を呼び出します。その後、**db2trc** コマンドを使用してトレース操作を開始および制御することができます。

ラッパー・ユーティリティー・クラスは、関数の項目ポイント、エグジット・ポイント、およびエラー・トレースからのデータを検索するためにラッパー・コードで使用できる、メンバー関数のセットを備えています。以下の表は、これらのメンバー関数の目的を説明しています。

表 63. ラッパー・ユーティリティー・クラス用のトレース・メンバー関数

C++ のメンバー関数	Java™ のメンバー関数	目的
fnc_entry	traceFunctionEntry	関数への入り口を記録する。
fnc_exit	traceFunctionExit	関数からの出口を記録する。
fnc_data	traceFunctionData	プローブ・ポイントと単一のデータ・エレメントを含むデータ・トレースを記録する。
fnc_data2	traceFunctionData	プローブ・ポイントと、必要ならば 2 つのデータ・エレメントを含むデータ・トレースを記録する。
fnc_data3	traceFunctionData	プローブ・ポイントと、必要ならば 3 つのデータ・エレメントを含むデータ・トレースを記録する。
trace_error	traceError または traceException	エラー・コードとプローブ・ポイントを含むエラー・トレースを記録する。

ラッパー・コードにトレース・メンバー関数を入力した後、**db2trc** コマンドを使用してトレース機能を開始します。トレース機能は、ラッパー制御フロー情報を収集します。このデータをラッパーから収集するには、**db2trc** コマンドを 135 のトレ

ース・レコード・コンポーネント ID で発行する必要があります。 135 のコンポーネント ID はカスタム・ラッパーに固有のものです。

注: 外部ラッパーのコンポーネント ID は変更される場合があります。最新の情報については、「*DB2 Information Integrator リリース・ノート* 」を参照してください。

関連タスク:

- 150 ページの『ラッパーからのトレース情報の作成』

関連資料:

- 151 ページの『ラッパー・トレース機能の例』

ラッパーからのトレース情報の作成

トレース機能を使用すると、ラッパーからの制御フロー情報をログに記録し、トラブルシューティング情報を取得できます。ラッパー・ユーティリティー・クラスは、ラッパーから制御フロー・レコードを収集するメンバー関数を備えています。**db2trc** コマンドを使用して、これらの制御フロー・レコードを抽出し、この情報を読み取り可能なテキストにフォーマットします。

手順:

ラッパーからのトレース情報を作成するには、以下のようにします。

1. **db2trc** コマンドを、適切なバッファー・サイズを指定して発行します。

   たとえば、以下のようになります。

   ```
   db2trc on -l 8M
   ```

2. 適当な長さの時間トレース機能を実行した後、トレース・バッファーの現在の内容をファイルに書き込みます。

   たとえば、以下のようになります。

   ```
   db2trc dump trc.dmp
   ```

   このコマンドは、トレース情報を現行ディレクトリーの trc.dmp という名前の出力ファイルに書き込みます。

3. 次のコマンドを発行してトレース機能をオフにします。

   ```
   db2trc off
   ```

4. トレース出力の内容を、ラッパーからのエラー・コードおよびトレース・フロー情報を含むファイルに書き込みます。

   たとえば、以下のようになります。

   ```
   db2trc flw  -m *.*.135.*.* trc.dmp trc.flw
   ```

   `*.*.135.*.*` のマスク・オプションを指定して、外部ラッパーに対応するトレース・レコードのみを検索する必要があります。 135 の値は、外部ラッパーのコンポーネント ID です。

注: 外部ラッパーのコンポーネント ID は変更される場合があります。最新の情報については、「*DB2 Information Integrator リリース・ノート* 」を参照してください。

5. トレース出力の内容を、ラッパーからのトレース・データを含むファイルに発生順に書き込みます。

   たとえば、以下のようになります。

```
db2trc flw  -m *.*.135.*.* trc.dmp trc.fmt
```

関連概念:

- 149 ページの『ラッパー・トレース機能』

関連資料:

- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』
- 151 ページの『ラッパー・トレース機能の例』

ラッパー・トレース機能の例

このトピックでは、ラッパー・コード内でラッパー・ユーティリティー・クラスを使用することによりトレース・メンバー関数を呼び出す方法を例示します。このトピックでは、ラッパー・コード内のトレース・メンバー関数からの制御フロー情報を含む trc.flw および trc.fmt ファイルについても例示します。トレース・ファイルは、**db2trc** コマンドを発行して生成します。

この例では C++ メソッド名を使用します。 Java でラッパーを作成する場合は、対応する Java メソッド名を代わりに使用してください。

以下の例は、トレース・メンバー関数を呼び出すラッパー・ユーティリティー・クラスを使用する UnfencedWrapper_Hook 関数を示しています。この例では、fnc_data2 メンバー関数により示される 2 つのデータ・エレメントのトレース情報を取得します。

```
extern "C" UnfencedWrapper* UnfencedWrapper_Hook()
{

    #define FUNC_ID 1
    UnfencedWrapper* wrapper=NULL;
    sqlint32 rc=0;
    const char* fName = "UnfencedWrapper_Hook";
    Wrapper_Utilities::fnc_entry(FUNC_ID, fName);
    Wrapper_Utilities::fnc_data2(FUNC_ID, fName, 10,
         strlen("First Function"), "First Function", sizeof(rc), &rc);

    wrapper = new(&rc) Sample_Wrapper(&rc);

    if( (rc) || (wrapper == NULL) )
    {
      Wrapper_Utilities::trace_error(FUNC_ID, fName,
                           30, sizeof(rc), &rc);

      if (wrapper != NULL )
      {
          delete wrapper;
          wrapper = NULL;
      }

    }

    Wrapper_Utilities::fnc_exit(FUNC_ID, fName, rc);
    return wrapper;
}
```

図 10. トレース・メンバー関数を持つラッパー・コード

次いで、**db2trc** コマンドを発行してトレース機能を開始します。

以下の例は、トレース機能を実行した後の trc.flw および trc.fmt ファイルの内容を示しています。

trc.flw ファイルの例を以下に示します。

trc.flw ファイルは、ラッパー・トレース・エラー・コード、関数のエントリー・ポイント、関数のエグジット・ポイント、およびデータ・トレース・ポイントを含んでいます。

probe 0 は、トレース機能を呼び出す関数名への参照を示しています。関数名は trc.flw ファイルでは指定されません。

```
pid = 2316 tid = 2292 node = 0

1           Func~1 entry
2           Func~1 data [probe 0]
3           Func~1 data [probe 0]
4           Func~1 data [probe 10]
5           Func~1 data [probe 0]
6           Func~1 exit
```

図 11. trc.flw ファイル

trc.fmt ファイルの例を以下に示します。

trc.fmt ファイルは、trc.flw ファイルで probe 0 により示される実際の関数名を含んでいます。

以下の trc.fmt ファイルの 1 行目と 6 行目は、それぞれ関数の項目ポイントとエグジット・ポイントを示しています。 2、3、および 5 行目は、ラッパー・コードからトレース機能を呼び出す UnfencedWrapper_Hook 関数への参照を示しています。 4 行目は、データ・トレースおよびデータ・エレメントを示しています。

```
1 entry DB2 External Wrappers Func~1 fnc (1.3.135.1.0)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 0

2 data DB2 External Wrappers Func~1 fnc (3.3.135.1.0.0)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 527 probe 0
  bytes 28
  Data1  (PD_TYPE_HEXDUMP,20) Hexdump:
  556E 6665 6E63 6564 5772 6170 7065 725F 486F 6F6B    UnfencedWrapper_Hook

3 data DB2 External Wrappers Func~1 fnc (3.3.135.1.0.0)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 639 probe 0
  bytes 28
  Data1  (PD_TYPE_HEXDUMP,20) Hexdump:
  556E 6665 6E63 6564 5772 6170 7065 725F 486F 6F6B    UnfencedWrapper_Hook

4 data DB2 External Wrappers Func~1 fnc (3.3.135.1.0.10)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 696 probe 10
  bytes 34

  Data1  (PD_TYPE_HEXDUMP,14) Hexdump:
  4669 7273 7420 4675 6E63 7469 6F6E         First Function

  Data2  (PD_TYPE_HEXDUMP,4) Hexdump:
  0000 0000                                  ....

5 data DB2 External Wrappers Func~1 fnc (3.3.135.1.0.0)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 2644 probe 0
  bytes 28

  Data1  (PD_TYPE_HEXDUMP,20) Hexdump:
  556E 6665 6E63 6564 5772 6170 7065 725F 486F 6F6B    UnfencedWrapper_Hook

6 exit DB2 External Wrappers Func~1 fnc (2.3.135.1.0)
  pid 2316 tid 2292 cpid -1 node 0 sec 0 nsec 2737 rc = 0
```

図 12. trc.fmt ファイル

関連概念:

- 149 ページの『ラッパー・トレース機能』

関連タスク:

- 150 ページの『ラッパーからのトレース情報の作成』

関連資料:

- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』

用語集

［ア行］

異種システム (heterogeneous systems). 異なるコンピューティング・リソースの範囲を指定して、同種ではないシステムを集めたもの。相互にローカルであるか地理的に分散している場合がある。

［カ行］

拡張エンタープライズ・アプリケーション (extended enterprise applications). 組織間のプログラム的な対話を統合するアプリケーション。企業間取引アプリケーションとも呼ばれる。

構成 (composition). これは、DB2 表のデータから、XML 文書を作成する (または組み立てる) プロセスである。生成された XML 文書のエレメントは、1 つ以上の DB2 表のフィールドから作成される。そして、XML 文書は、DB2 内外のファイル・システムおよび MQSeries メッセージ・キューに保管できる。

構造化データ (structured data). スプレッドシート、アドレス帳、構成パラメーター、金融トランザクション、または技術図面を含められるデータ。構造化照会言語 (SQL) と構造化データは相性が良い。

コラボレーション (collaboration). 使用事例シナリオを実現する一連の操作が発生すること。ユーザー間 (user-to-user) とも呼ばれる。一般に、2 つかそれ以上のコンポーネント間のコラボレーションが関係する。一例として、顧客の詳細をクライアント/サーバー・システムで更新するシナリオが考えられる。グラフィカル・ユーザー・インターフェース (GUI) コンポーネントが、ウィンドウを表示する、データ要求でデータ・サーバー・コンポーネントを呼び出す、顧客の詳細を表示する (そして、修正する)、データ・サーバーを呼び出して更新を実行するときの一連の操作がある。このようなコンポーネント間でのコンポーネント操作および交換のパターン全体が、シナリオを「実現する」コラボレーションである。

［サ行］

シナリオ (scenario). 使用事例のインスタンス。つまり、シナリオとは、十分に満たされた前提事項の下で使用事例を実行すること。シナリオは、コラボレーションによって特定のシステムで実現する。

使用事例 (use case). 識別可能で外部から観察が可能な、特定のシステム内での動作。これは、行為者またはユーザーによって開始され、有益な作業を実行する (またはしようとする) 使用法のパターンである。使用事例は、行為者とシステムとの間の会話を表す。たとえば、「当座預金から資金を引き出す」は使用事例である。

情報集約アプリケーション (information aggregation applications). ツールで他のデータ・ソースから情報を取り出すアプリケーション。 user-to-data とも呼ばれる。

情報データ (informational data). オペレーショナル・データから抜き出され、意思決定用に変換されたデータ。「オペレーショナル・データ」も参照。

情報データ (informational data). オペレーショナル・データから抜き出され、意思決定用に変換されたデータ。「オペレーショナル・データ」も参照。

セルフサービス・アプリケーション (self-service applications). ユーザーが企業のトランザクションおよびデータとやり取りする、User-to-Business アプリケーションとも呼ばれる。

［ナ行］

ニックネーム (nickname). (1) フェデレーテッド・システムでは、データ・ソースにあるオブジェクトを参照するために照会で使用される ID。ニックネームが識別するオブジェクトは、データ・ソース・オブジェクトと呼ばれる。データ・ソース・オブジェクトの例としては、表、ビュー、同義語、表構造のファイル、および検索アルゴリズムがある。 (2) 非 DB2 リレーショナル・データベースにおいて物理データベース・オブジェクト (表やストアード・プロシージャーなど) を表すために、DB2 Information Integrator で定義されている名前。

ネーム・スペース表 (namespace table). ネーム・スペース表 (NST) リソースは、DB2 XML エクステンダー DTDID から XML スキーマ (XSD) ネーム・スペースおよびロケーションへのマッピングを定義する。ネーム・スペースを使用すると、エレメント (属性の場合もあり) 名を、複数の XML ボキャブラリーから 1 つの XML 文書にまとめることができる。

［ハ行］

半構造化データ (semistructured data). スプレッドシート、アドレス帳、構成パラメーター、金融トランザクション、または技術図面を含められるデータ。構造化照会言語 (SQL) と構造化データは相性が良い。

非構造化データ (unstructured data). おそらくは従来のデータベースの外側で、組織されずに保管されているすべてのデータ。このデータの例としては、テキスト、オーディオ、ビデオ、FAX、イメージ、またはグラフィックスがある。

分解 (decomposition). 分解 (shredding) とも呼ばれる。これは、XML 文書を DB2 に保管するプロセスである。 XML 文書は部品に分けられ (つまり分解され)、エレメントが 1 つ以上の DB2 表にフィールドとして保管される。

分解 (decomposition). 分解 (shredding) とも呼ばれる。これは、XML 文書を DB2 に保管するプロセスである。 XML 文書は部品に分けられ (つまり分解され)、エレメントが 1 つ以上の DB2 表にフィールドとして保管される。

［ラ行］

ラッパー (wrapper). フェデレーテッド・システムでは、フェデレーテッド・サーバーが、データ・ソースとの通信およびデータ・ソースからのデータの検索を行うために使用するメカニズム。ラッパーをインプリメントするために、フェデレーテッド・サーバーは、ライブラリーに保管されているラッパー・モジュールという名前のルーチンを使用する。これらのルーチンは、データ・ソースに接続し、そこからデータを繰り返し検索するための接続といった操作を、フェデレーテッド・サーバーが実行できるようにする。 DB2 Universal Database フェデレーテッド・インスタンスの所有者は、フェデレーテッド・システムに組み込む各データ・ソースのラッパーを登録するために、CREATE WRAPPER ステートメントを使用する。

レガシー・データ (legacy data). 数十回の情報収集およびデータ分析によって生成されるデータで、すでに所有し使用しているもの。大抵は、現在使用中のシステム上にある既存のデータベースのレコードという形をとる。これは一般には、ローカルまたは所有データベースに存在する情報で、データ管理システムに安全にしまわれているので、通常はエンタープライズ・システム側から使用することはできない。

D

DB2 管理者 (DB2 Administrator). アクセス許可やコンテンツ管理などの管理作業を担当するユーザー。管理者は、ユーザーに権限のレベルを付与することもできる。

E

Enterprise Java Bean. Enterprise Java Bean は、分散アプリケーションを作成するために他の Enterprise Bean や他の Java コンポーネントと結合させることができる Java コンポーネントである。 Enterprise Bean には、Entity Bean と Session Bean という 2 つのタイプがある。

F

fenced. データベース・マネージャーとは別個のプロセスで実行するように定義された、プロシージャー、ユーザー定義関数、またはフェデレーテッド・ラッパーのタイプまたは特性に関係している。このタイプのオブジェクトが (fenced 文節を使用して) 実行されると、データベース・マネージャーは変更されないようにそのオブジェクトにより保護される。

H

Hypertext Transfer Protocol (HTTP). Hypertext Transfer Protocol (HTTP) は、分散、コラボレーション、ハイパーメディア情報システムのための、アプリケーション・レベルのプロトコルである。

J

Java 2 Platform、Enterprise Edition (J2EE). 複数層での分散アプリケーション・モデル、コンポーネントを再利用する機能、統合 Extensible Markup Language (XML) ベースのデータ交換、統一されたセキュリティー・モデル、および柔軟なトランザクション制御を提供するプラットフォーム。

U

unfenced. データベース・マネージャーのプロセスで実行するために定義された、プロシージャー、ユーザー定義関数、またはフェデレーテッド・ラッパーのタイプまたは特性に関係している。このタイプのオブジェクトが (not fenced 文節を使用して) 実行されると、このオブジェクトによって加えられる変更からデータベース・マネージャーは保護されない。

W

WORF. Web Object Runtime Framework。 DB2 Web サービス・プロバイダーをサポートするランタイム・エンジン。

X

XSD. Extensible Markup Language (XML) スキーマ定義。スキーマを含む XML ファイルを記述するための言語。

アクセス支援

アクセス支援機能は、身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーがソフトウェア製品を十分活用できるように支援します。 DB2® バージョン 8 製品に備わっている主なアクセス支援機能は、以下のとおりです。

- すべての DB2 機能は、マウスの代わりにキーボードを使ってナビゲーションできます。詳細については、『キーボードによる入力およびナビゲーション』を参照してください。
- DB2 ユーザー・インターフェースのフォント・サイズおよび色をカスタマイズすることができます。詳細については、 160 ページの『アクセスしやすい表示』を参照してください。
- DB2 製品は、Java™ Accessibility API を使用するアクセス支援アプリケーションをサポートします。詳細については、 160 ページの『支援テクノロジーとの互換性』を参照してください。
- DB2 資料は、アクセスしやすい形式で提供されています。詳細については、 160 ページの『アクセスしやすい資料』を参照してください。

キーボードによる入力およびナビゲーション

キーボード入力

キーボードだけを使用して DB2 ツールを操作できます。マウスを使って実行できる操作は、キーまたはキーの組み合わせによっても実行できます。標準のオペレーティング・システム・キー・ストロークを使用して、標準のオペレーティング・システム操作を実行できます。

キーまたはキーの組み合わせによって操作を実行する方法について、詳しくは キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・ナビゲーション

キーまたはキーの組み合わせを使用して、 DB2 ツールのユーザー・インターフェースをナビゲートできます。

キーまたはキーの組み合わせによって DB2 ツールをナビゲートする方法の詳細については、キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・フォーカス

UNIX® オペレーティング・システムでは、アクティブ・ウィンドウの中で、キー・ストロークによって操作できる領域が強調表示されます。

アクセスしやすい表示

DB2 ツールには、視力の弱いユーザー、その他の視力障害をもつユーザーのためにアクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

「ツール設定」ノートブックを使用して、メニューおよびダイアログ・ウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

フォント設定に関する詳細情報は、メニューおよびテキストのフォントを変更する: Common GUI help を参照してください。

色に依存しない

本製品のすべての機能を使用するために、ユーザーは必ずしも色を識別する必要はありません。

支援テクノロジーとの互換性

DB2 ツールのインターフェースは、Java Accessibility API をサポートします。これによって、スクリーン・リーダーその他の支援テクノロジーを DB2 製品で利用できるようになります。

アクセスしやすい資料

DB2 形式は、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で提供されています。 XHTML により、ご使用のブラウザーに設定されている表示設定に従って資料を表示できます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

シンタックス・ダイアグラムはドット 10 進形式で提供されます。この形式は、スクリーン・リーダーを使用してオンライン・ドキュメンテーションにアクセスする場合にのみ使用できます。

関連概念:

- 「*Infrastructure Topics (DB2 Common Files)*」の『ドット 10 進シンタックス・ダイアグラム』

関連タスク:

- 『キーボード・ショートカットおよびアクセラレーター: Common GUI help』
- 『メニューおよびテキストのフォントを変更する: Common GUI help』

特記事項

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

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

IBM Director of Licensing
〒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 Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

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。

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。

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の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。

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

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。

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]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

B

C

D

F

J

L

P

R

T

U

W

X

IBM と連絡を取る

技術上の問題がある場合は、お客様サポートにご連絡ください。

製品情報

DB2 Information Integrator についての情報は、電話または Web から入手することができます。

米国にお住まいの場合は、以下のいずれかの番号にお問い合わせください。

- 製品の注文または一般情報の入手: 1-800-IBM-CALL (1-800-426-2255)
- 資料の注文: 1-800-879-2755

Web 上で http://www.ibm.com/software/data/integration/db2ii/support.html にアクセスします。このサイトには、最新のテクニカル・ライブラリーに関する情報、資料の注文、クライアントのダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースのリンクが含まれています。

お住まいの国や地域の IBM 事務所の所在地を調べる場合は、Web 上で IBM Directory of Worldwide Contacts (www.ibm.com/planetwide) を参照してください。

資料についてのコメント

お客様のフィードバックは IBM が良質な情報を提供する助けになります。この資料や他の DB2 Information Integrator の資料についてのコメントをお送りください。コメントの送付には、以下のいずれかの方法を利用することができます。

- www.ibm.com/software/data/rcf で、オンラインの読者コメント・フォームを使用して送信する。
- 電子メール (E メール) で comments@us.ibm.com に送信する。お送りいただく情報には、製品の名前、製品のバージョン番号、および資料の名前と部品番号 (該当する場合) を必ず含めてください。特定の本文についてコメントする場合は、本文の位置 (たとえば、タイトル、表の番号、またはページ番号) を含めてください。

Printed in Japan

SC88-9923-00

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