IBM DB2 Information Integrator

包装器开发者指南

版本 8.2

S152-0845-00

IBM DB2 Information Integrator

包装器开发者指南

版本 8.2

S152-0845-00

在使用本资料及其支持的产品之前，请阅读第 125 页的『声明』中的一般信息。

本文档包含 IBM 的专利信息。它在许可协议下提供，并受版权法保护。本出版物包含的信息不包括任何产品保证，且本手册提供的任何声明不应作如此解释。

可以在线方式或通过您当地的 IBM 代表订购 IBM 出版物。

- 要以在线方式订购出版物，可访问“IBM 出版物中心”（IBM Publications Center），网址为 www.ibm.com/shop/publications/order
- 要查找您当地的 IBM 代表，可访问“IBM 全球联系人目录”（IBM Directory of Worldwide Contacts），网址为 www.ibm.com/planetwide

当您发送信息给 IBM 后，即授予 IBM 非专有权，IBM 可以它认为合适的任何方式使用或分发此信息，而无须对您承担任何责任。

目录

关于本书

本书帮助您开发包装器以访问定制数据源或 IBM 提供的标准包装器集合中未包括的数据源。

谁应该阅读本书?

要开发包装器以访问 IBM DB2 Information Integrator 的定制数据源的数据管理员、信息分析员、系统集成者、Web 集成者、数据资料管理员、数据体系结构设计者和应用程序开发者。

本书中使用的约定和术语

IBM DB2 Information Integrator 使用有关数据库、连接、“结构化查询语言”（SQL）和局域网（LAN）的概念的标准术语。在本书中使用的所有 DB2 Information Integrator 概念都在词汇表中作了定义。除非另外指定，否则假定下列意义:

数据 原始情况。它可以是结构化的、非结构化的或半结构化的。通常会组织数据以进行分析。数据还帮助您作出决策。

信息 可用格式的数据，通常以某种方式处理和解释。

第 1 部分 联合概念和开发包装器的概述

本书的此部分提供了在开发包装器之前需要熟悉的主要联合概念的概述并提供了整个包装器开发过程的概述。

第 1 章 联合概念的概述

下列主题提供了在开发包装器之前需要熟悉的主要联合概念的概述。

为什么要开发包装器?

问题: 没有一种容易的方法存取或集成不同种类的数据

企业在不同种类的关系数据管理系统（RDBMS）、非关系服务器和专业应用程序系统中存储数据。这种服务器的激增是在信息技术（IT）产业中的正常变化中产生的。这些变化包括 IT 企业的合并、产品和解决方案的逐步升级以及应用程序处理专门数据的需要。

当然，从不同的数据源合并数据并据此得到这些数据的新的观点是有益的。例如，可将联合服务器中的客户数据与存储在定制地理信息系统（GIS）中的信息连接在一起。这将使您能够映射具有特定喜好的客户的位置。或者，作为合并公司各部门的结果，您可能需要同时从两个部门检索存储在不同服务器中的数据。

遗憾的是，多种类型数据存储的一个结果是: 用户和应用程序接口被设计为存取有限的服务器子集。因此，如果大多数数据可通过一个接口存取，则仍将有一些关键数据是此接口不能存取的。于是您将需要第二个接口以存取这些关键数据。为此，您可能需要编写专门的应用程序以将两个数据集集成到同一结果集中。

解决方案: 联合系统

DB2® Information Integrator 的解决方案是使您能够设置称为*联合系统*的分布式计算系统。通过联合系统，客户机可以从单个接口查询所有种类的数据源。一个称为*联合服务器*的联合服务器实例管理该系统。有时数据源又称为*外部服务器*。

对于 DB2 UDB 客户机（最终用户和应用程序），联合系统中的数据源表现为单个集中数据库。事实上，带有 DB2 UDB 数据库的客户机接口称为*联合数据库*。联合数据库是由联合服务器管理的。要从数据源检索数据，客户机将 DB2 UDB 的 SQL 方言中的查询提交给联合数据库。此查询引用数据源内的表或其它数据存储。它可以表或视图的形式请求结果，这些结果会组合任意数目的这些数据源的输出。

示例: 存取和集成数据

假定联合系统中的两个非关系数据源包含生命科学信息。在其中一个数据源中，有一个表包含有关生化实验及其结果。在另一个数据源中，有一个数据集包含有关作为新药的候选项的分子的信息。对于表 EXP 和数据集 MOLECULES，联合服务器分别有自己的标识。这种标识（联合服务器通过其引用表、数据集和外部服务器中其它种类的数据存储的名称）称为*昵称*。

假定用户想要查找某种结构的分子，这种结构类似于在胃部实验中会产生特定结果的分子结构。为此，用户编写通过其昵称引用表和数据集的查询。第 4 页的图 1 显示此查询。

```
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
```

图 1. 通过查询来请求与在胃部实验中结果大于 0.8 的分子相似的分子的标识.

客户机将此查询提交给联合数据库。联合服务器查询其系统目录并发现 EXP 是第一个服务器中的表的昵称。它发现 MOLECULES 是另一个服务器中的数据集的昵称。联合服务器从该目录了解到指定为从第一个服务器检索数据的包装器称为 EXPERIMENTS。它还从该目录了解到指定为以从第二个服务器检索数据的包装器称为 MYMOL。

联合服务器和包装器共同为该查询开发一个执行方案。在此方案的基础上，联合服务器将该查询分解为几个分段。然后，包装器使用服务器各自的应用程序编程接口（API）将这些分段下推至服务器。服务器将返回结果。包装器从 API 检索结果并将它们传递至联合服务器。联合服务器合并这些结果并将它们返回至客户机。

完成基本联合查询

从较高级别看，联合查询中的基本步骤包括:

1. 用户或应用程序提交查询。
2. 联合服务器根据源分解查询。
3. 联合服务器和包装器共同制定查询方案。
4. 联合服务器通过查询执行实现该方案。
5. 包装器通过每个源的 API 获取源。
6. 源将数据返回给包装器。
7. 包装器将数据返回至联合服务器。
8. 联合服务器补偿数据源不能做的工作并组合不同源中的数据。
9. 联合服务器将数据返回至用户或应用程序。

在提交查询之后，联合服务器查询其系统目录。它会查找一些信息，例如，哪些表或其它数据存储包含要检索的信息以及哪些包装器被指定为启动该检索。

联合服务器设计出用于对该查询求值的备用策略，称为*存取方案*。这种方案可能要求查询的某些部分由数据源或联合服务器处理，或部分由源处理，部分由联合服务器处理。联合服务器主要根据成本在这些方案之间进行选择。

优化器生成用户的应用程序提交的原始查询的子段，称为*查询段*。联合服务器在一个*请求*中将每个查询段提交给包装器。包装器以*应答*响应。该应答让优化器知道包装器可以执行该特定查询段的哪些*子段*（例如，选择列表元素和谓词）。这一组子段称为*已接受分段*。在应答中，如果要求对已接受分段求值，包装器还会估计将生成的行的成本（及时）和数目。然后，优化器通过将数据源不能处理的这些子段添加至查询方案的联合服务器部分以对它们作出*补偿*。总之，包装器和联合服务器交互在查询方案期间的整个过程称为*请求应答补偿（RRC）*协议。

联合服务器分析个别分段的方案的组合以确定最佳的整体方案。

相关概念:

- 第 8 页的『联合系统的查询处理』
- 第 5 页的『包装器模块』

包装器模块

在 C++ 中，包装器模块是带有可存取一类数据源的特定入口点的共享库。

Java™ 包装器是一组 Java 类。

DB2® UDB 以动态方式按需装入 C++ 和 Java 包装器模块。

包装器模块是您将使用从随联合服务器提供的基本类派生的特定类进行编码的对象。它将包含特定构建块，该构建块允许它充当数据源和联合系统之间的转换程序。

图 2 演示包装器模块的各个部件。

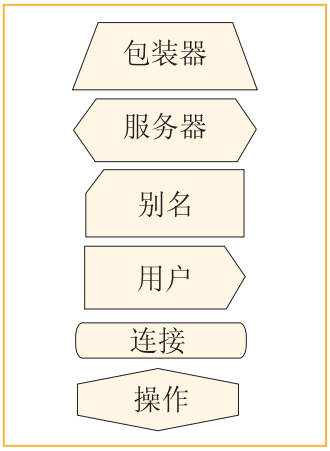

图 2. 包装器模块的各个部件

图 2 显示包装器模块包含下列构建块:

- 包装器
- 服务器
- 远程用户
- 昵称
- Remote_Connection
- Remote_Operation

所有构建块（远程连接和远程操作类除外）表示在联合服务器目录中注册和描述的持久实体。

表 1. 与包装器模块的基本构建块相关联的描述、服务和 DDL

构建块名称	描述	提供的服务	相关联的 DDL
包装器	代码模块。它是带有表示一类数据源的特定入口点的共享库。	• 引导和初始化 • 访问受包装器支持的服务器	CREATE WRAPPER

表 1. 与包装器模块的基本构建块相关联的描述、服务和 DDL (续)

构建块名称	描述	提供的服务	相关联的 DDL
服务器	表示受包装器支持的特定数据源。	• 存取一组昵称 • 使用“请求应答补偿”协议的查询规划	CREATE SERVER
昵称	表示服务器上的特定数据集合。	在昵称注册时或通过 DDL 描述联合服务器的昵称模式。	CREATE NICKNAME
用户	提供对特定服务器认证最终用户所需的信息。	将联合服务器用户信息映射至源信息。例如，用户标识和密码。	CREATE USER MAPPING
Remote_Connection	表示联合服务器与源的连接。 瞬态：没有持久目录信息。	• 从源连接／连接至源和从源断开连接／断开与源的连接 • 事务管理	不适用
Remote_Operation	表示源处的活动操作（例如，查询、插入、更新和删除）。 瞬态：没有持久目录信息。 可能正在执行多个操作。	• 远程查询：数据的只读查询 • Passthru：与数据源的直接会话（使用源的名称的源的语言）	不适用

相关概念:

- 第 8 页的『联合系统的查询处理』
- 『包装器和包装器模块』（*《联合系统指南》*）
- 第 3 页的『为什么要开发包装器？』

用户如何将数据源添加至联合系统

从用户或应用程序的透视图中，添加数据源涉及:

- 告诉联合服务器外部数据将如何表示关系模型中的行和列
- 为数据源配置包装器以便它可与源通信以检索其数据

通过最终用户或应用程序发出的一系列 DDL 语句完成注册过程。

当用户将 DDL 语句提交给联合服务器时，相关联的包装器代码将验证该语句的正确性。包装器可以使用数据源中的信息或通过硬编码的信息补充 DDL 信息。数据源可为包装器提供信息（例如，配置参数和统计信息）。还可将参数称为主变量、输入变量或输入数据。在包装器验证 DDL 语句中的信息的有效性之后（可能补充该信息），联合服务器会在相应的 DB2 目录中存储该信息。

DDL 语句上的选项允许您跨越同一数据源的几个不同变体使用包装器。

例如，平面文件的包装器可具有指定该平面文件的路径的选项。这样，一个包装器可以不同的路径处理这些平面文件。

用户或应用程序必须完成才能添加新数据源的步骤包括:

1. 使用 CREATE WRAPPER DDL 语句注册包装器
   a. 确定此包装器支持哪些包装器选项
   b. 确定包装器选项的值
   c. 发出 CREATE WRAPPER DDL 语句

      例如,

      ```
      CREATE WRAPPER Dctm_Wrapper LIBRARY 'libdb2lsdctm.a';
      ```
2. 注册服务器
   a. 确定此包装器支持哪些服务器选项
   b. 确定服务器选项的值
   c. 发出 CREATE SERVER DDL 语句

      例如,

      ```
      CREATE SERVER Dctm_Server1
      TYPE DCTM
      VERSION 3
      WRAPPER Dctm_Wrapper
      OPTIONS(NODE 'Dctm_Docbase',
      OS_TYPE 'AIX',
      RDBMS_TYPE 'ORACLE');
      ```
3. 如果需要，使用 CREATE USER MAPPING DDL 语句定义该服务器的远程用户

   例如,

   ```
   CREATE USER MAPPING FOR Chuck SERVER Dctm_Server1
   OPTIONS(REMOTE_AUTHID 'Charles',REMOTE_PASSWORD 'Charles_pw');
   ```
4. 注册该服务器的所有专门函数:
   a. 使用 CREATE FUNCTION... AS TEMPLATE DDL 语句创建本地模板函数

      例如,

      ```
      CREATE FUNCTION DCTM.ANY_EQ (CHAR(),CHAR())RETURNS INTEGER
      AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
      ```
5. 为数据集创建昵称
   a. 确定将在 CREATE NICKNAME 语句上指定哪些列名和列类型
   b. 确定此包装器支持哪些昵称选项和列选项
   c. 确定昵称选项和列选项的值
   d. 发出 CREATE NICKNAME DDL 语句

      例如,

      ```
      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. 通过使用标准 SQL 语句查询数据

   例如,

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

编写包装器代码时，由您负责在一组约束下定义 DDL 语句的形式。您还要负责确定要使用哪些现有选项和创建数据源所需的任何新选项。

相关概念:

- 第 3 页的『为什么要开发包装器？』

相关任务:

- 第 107 页的『将数据源添加至 DB2 控制中心』

联合系统的查询处理

以下是查询处理的两个阶段：查询规划和查询执行。查询规划阶段发生在编译时期间。查询执行阶段发生在运行时期间。第 9 页的图 3 显示编译时和运行时期间联合系统中的查询处理流。

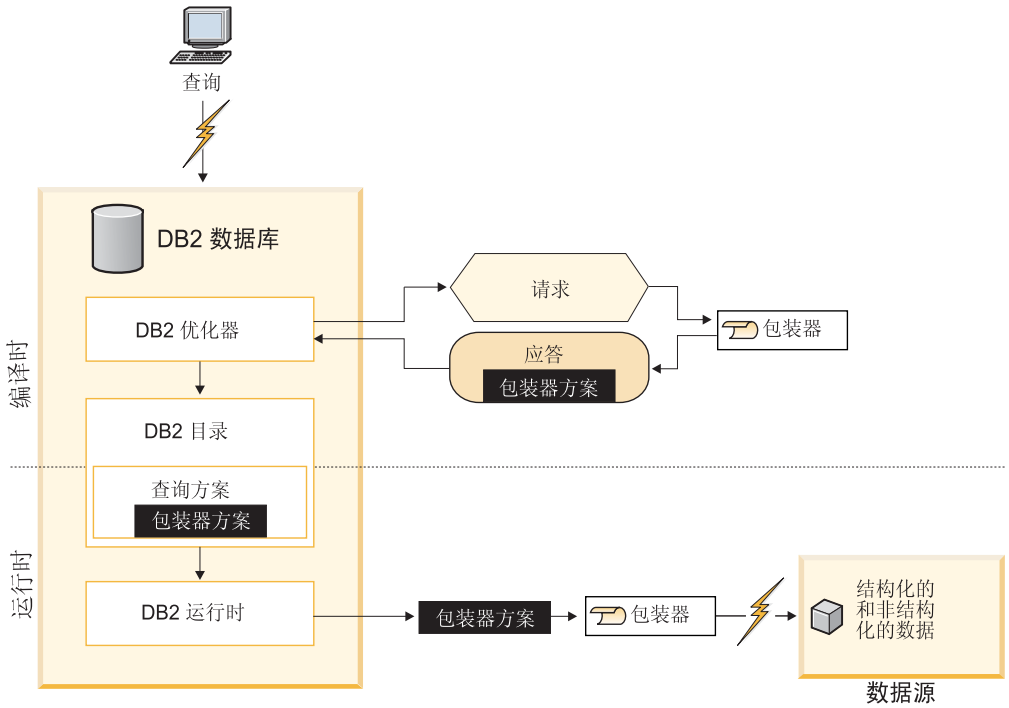

图 3. 联合查询处理流

相关概念:

- 第 14 页的『联合系统的查询执行』
- 第 5 页的『包装器模块』

请求应答补偿协议

在查询规划期间，优化器生成由用户的应用程序提交的原始查询的分段，称为*查询段*。查询段可以包含表、谓词和头表达式。头表达式是在查询的 SELECT 子句中找到的表达式。然后，优化器在一个*请求*中对包装器提交每个查询段。

根据定义，对分段求值所需的所有数据来自一个数据源。但是，此数据的处理可通过外部服务器和 / 或联合服务器完成。包装器指示它可对分段的哪个子段（称为*子段*）求值，并将此信息装入该请求的*应答*中。包装器必须接受或拒绝整个子段。

应答包含:

- 已接受的昵称、谓词和头表达式
- 已接受的分段的成本和基数估计
- 将返回的结果的排序属性（如果有的话）
- 包装器执行描述符（在后面的文本中描述）

对于此文档，“量词”和“昵称”是可交换的。

对于单个查询，优化器一般为每个包装器生成多个请求，每个请求表示原始查询的不同分段。对于每个这样的请求，包装器生成零个、一个或多个应答。每个应答表示不同的已接受分段。已接受分段是包装器或数据源可以自己求值的分段。每个应答包含已接受分段的关联成本和基数估计。

在查询规划结束时，优化器将做出基于成本的决策。它将提出一个方案：合并该包装器为响应请求而提供的一组已接受分段。最终将要求该包装器执行这一组特定已接受分段。

优化器将确保不属于它选择的方案中的已接受分段中的任何子段将由联合服务器进行求值。此问题的示例包括已超出所讨论的数据源的能力的复杂谓词或排序。这包括所有交叉源连接以及混合多个源中的数据的任何其它表达式（例如函数调用），原因是分段是单个源的。这称为*补偿*。

总之，在编译期间包装器和联合服务器交互的整个过程称为*请求应答补偿（RRC）*协议。

包装器可以自己确定要放入应答中的成本和基数估计，或者它可以重新使用由联合服务器提供的缺省成本模型。它还可以有选择地替换缺省成本模型的部分以改进其精确性而不需要从头开始构建新的模型。缺省模型使用包装器可通过数据计算得出的或可通过 DDL 提供的统计信息。

作为应答的一部分，包装器还必须提供*包装器执行描述符*。这是一个黑匣，其内容取决于包装器。包装器必须能够将应答表示的已接受分段提交至数据源。如果优化器在执行方案中使用它最终选择的已接受分段，则在要运行查询时，“包装器执行描述符”会被返回至包装器。在这期间，“包装器执行描述符”作为预编译查询的存取方案的一部分驻留在联合服务器目录中。

该数据源的包装器对“包装器执行描述符”中的远程查询的持久表示有完全的控制权。

使用句柄处理请求和应答

DB2 UDB 提供功能接口以处理请求和应答。

请求响应 SQL 查询，但是该查询不是以 SQL 表示的，这是因为数据源不懂 SQL。相反，使用整数句柄以标识和浏览查询表达式。谓词和头表达式都被表示为运算符树。提供函数来描述每种节点并浏览至任何节点的子节点。提供方法以将一些项从请求移至应答。

请求应答补偿协议的示例

例如，考虑第 11 页的图 4 中的示例。

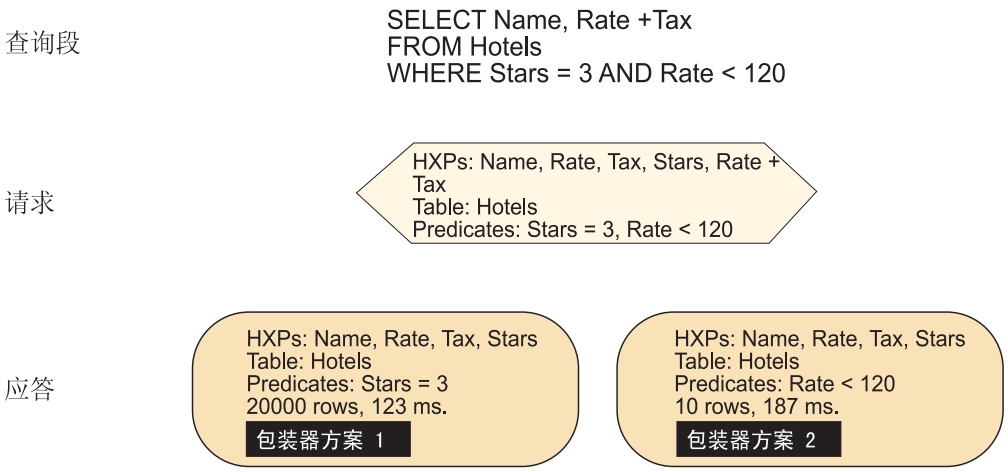

图 4. 请求应答补偿协议示例

该图显示正在使用的 RRC 协议。在此方案中，您想要存取作为 Web 站点的数据源，该站点仅允许您在对 Web 服务器的请求中指定一个谓词。

该图显示单个表存取查询的查询段。注意，SELECT 列表包含表达式：rate + tax。

在请求中，注意联合服务器可以请求在头表达式列表中分别包含在谓词和表达式中的列：rate、tax 和 stars。

现在存在带有不同的已接受谓词的两个应答（成本和基数）以及包装器方案（aka 包装器执行描述符）。两个方案都不接受头表达式 rate + tax。既然包装器和数据源都不能处理此头表达式，联合服务器将根据 rate 和 tax 的各自列值计算其值。

相关概念:

- 第 8 页的『联合系统的查询处理』
- 第 14 页的『联合系统的查询执行』
- 第 52 页的『查询执行的控制流』

联合查询的缺省成本模型

本节描述包装器如何使用成本模型将成本计算信息提供给联合服务器优化器。

查询成本和查询规划

通过“应答”类，包装器将信息提供给联合服务器。当确定处理特定查询的最合适方案时，联合服务器使用此信息。“应答”类提供计算以下四条信息的方法:

1. 应答表示的查询段的基数。这是查询段将返回的行数的估计。
2. 用来检索查询段所选的第一元组的估计时间（以毫秒计）。此估计是针对查询段的第一次调用而言的。它又称为分段的*第一元组成本*。
3. 用来检索查询段所选的整个回答集的估计时间（以毫秒计）。此估计是针对查询段的第一次调用而言的。
4. 用来检索查询段所选的整个回答集的估计时间（以毫秒计）。此估计是针对查询段的第二次或后续调用（可能绑定了新参数）而言的。它又称为分段的*重复执行成*

本。如果在包装器第一次将查询段提交至远程源时需要对包装器执行预处理，则重复执行成本将与第一元组成本不同。如果包装器提交同一分段，则将不需要重复预处理。

包装器可对“应答”类划分子类并为计算这些成本提供它自己的机制，或者它可以使用“应答”类提供的缺省成本模型。下面一节将描述缺省成本模型。

缺省成本模型

缺省“应答”类实现的成本模型使用一组缺省成本等式计算前面列示的四条信息，这些等式是由查询段中涉及的昵称的相关统计信息驱动的。使用缺省成本模型的包装器必须为每个昵称提供以下小节中列示的四条统计信息。

缺省成本模型的统计信息

缺省情况下，联合服务器将这些统计信息存储在系统目录中。通过覆盖相应的方法，包装器可以不同方式提供这些信息。包装器可以在覆盖这些方法的同时保留缺省成本模型的其余部分。

这四条统计信息包括:

1. 昵称的基数。它被定义为包含在昵称中的行数。联合服务器将个别昵称的基数存储在系统表 SYSCAT.TABLES 或 SYSSTAT.TABLES（任一表中的 “CARD” 列）中。如果昵称没有基数，则成本模型使用缺省值 1000 行。
2. 昵称的设置成本。设置成本表示包装器准备好查询段以提交至远程源时通常所花的时间（以毫秒计）。当包装器接收它在查询规划期间生成的包装器 “执行描述符” 时，设置开始，并且在包装器准备将相应操作提交给远程源时，设置结束。设置成本应仅包括当要求包装器再次执行同一查询段（可能带有不同的参数值）时它不需要重复的工作。例如，如果包装器将查询段以 URL 的形式提交给远程源，则设置成本包括从包装器存储在 “执行描述符” 中的信息生成该 URL 所需的时间。联合服务器将此统计信息存储在 SETUP_COST 昵称选项中。如果昵称没有该选项，则成本模型使用值 25 毫秒。
3. 昵称的提交成本。提交成本表示包装器将查询段提交至远程源通常所需的时间（以毫秒计）。提交在设置结束时（如上面定义的那样）开始，并且在包装器准备从源请求第一行或第一块结果数据时结束。提交成本应仅包括每次提交给定查询段时包装器必须重复的工作。例如，如果每次与远程源进行交互都需要新的 HTTP 连接，则提交成本应该包括创建此连接所需的时间。联合服务器将此统计信息存储在 SUBMISSION_COST 昵称选项中。如果昵称没有该选项，则成本模型使用值 2000 毫秒。
4. 昵称的高级成本。这是访存昵称的一行通常所需的时间（以毫秒计）。它是启动查询需要的任何时间之外的时间。联合服务器将此统计信息存储在 ADVANCE_COST 昵称选项中。如果昵称没有该选项，则成本模型使用的值是 50 毫秒。如果数据源以块而不是以行为单位返回数据，则通过将访存一块的典型成本除以每块中的典型行数来计算高级成本。

尽管可以通过检测包装器代码来获取必需的统计信息，但是通常来讲，通过针对所讨论的昵称的典型查询的运行时间的外部评估，获取它们会更加容易。例如，通过运行两个查询（每个查询返回的结果数目不一样）并将执行时间的差除以每个查询返回的行数差，就可以得出高级成本。这特别适用于这三个成本统计信息。数据源常常有确定基数的直接方法。

缺省成本等式

缺省成本模型提供一组四个的成本等式，它们派生优化器所需的四个参数。以下列表描述每个等式。

基数 分两个步骤计算查询段的基数。第一步，成本模型获取每个昵称的基数并使这些值加在一起。

在第二个步骤中，成本模型将第一个步骤中计算出来的总行数乘以查询段中的各个谓词的选择率。选择率是在 0.0 和 1.0 之间的数，它反映谓词将这些昵称中的行过滤出结果集的程度，较小的值说明过滤程度较大。缺省成本模型提供一种方法，它使用联合服务器提供的算法来估计谓词选择率。如果包装器能够提供一种更加精确的估计选择率的方法，则它可以覆盖此方法。选择率的主要错误原因是缺省模型完全不了解集合中的谓词之间的相关性。

例如，尽管谓词 `Make='Carmaker1_make'` 的选择率为 0.13，且 `Model='Carmaker1_model'` 的选择率为 0.05，但是 `Make='Car1_make' AND Model='Car1_model'` 的组合选择率还是 0.05，这是因为属性 `Make` 和 `Model` 的值不是独立的：每个汽车型号都来自同一汽车制造商。如果包装器意识到属性之间的相关性，则可以覆盖缺省选择率估计方法。还可在属性的值的分布很不对称时提供您自己的选择率估计方法。可在完成此操作的同时保留缺省成本模型的其余部分。此外还要注意，如果确实提供了定制选择率估计方法，则联合服务器优化器还将调用您的定制选择率方法来计算包装器在应答中不接受的所有谓词的选择率。正如前面所述，包装器可能意识到谓词之间的不对称性或相关性，从而提供更好的估计值，即使该源不能将这些谓词作为已接受查询段的一部分来求值。

第一元组成本

第一元组成本是三个值的和。第一个值是查询段中所有昵称的设置成本的平均值。第二个值是查询段中所有昵称的提交成本的平均值。第三个值是查询段中所有昵称的高级成本的平均值。

总成本 总（第一个回答集）成本是三个值的和。第一个值是查询段中所有昵称的设置成本的平均值。第二个值是提交成本的平均值。第三个值是高级成本的平均值乘以查询段的估计基数的乘积。

重复执行成本

查询段的重复执行成本是两个值的和。第一个值是查询段中的所有昵称的提交成本的平均值。第二个值是高级成本的平均值乘以查询段的估计基数的乘积。注意：在计算估计执行时间时，成本等式通常使用查询段中所有昵称的统计信息的平均值。如果这样不能得到数据源的合理结果，考虑通过对“应答”对象划分子类以覆盖该成本等式。

成本计算和查询规划总结

当要计算查询段的成本时，您有几个选项。第 14 页的图 5 演示了这些选项。

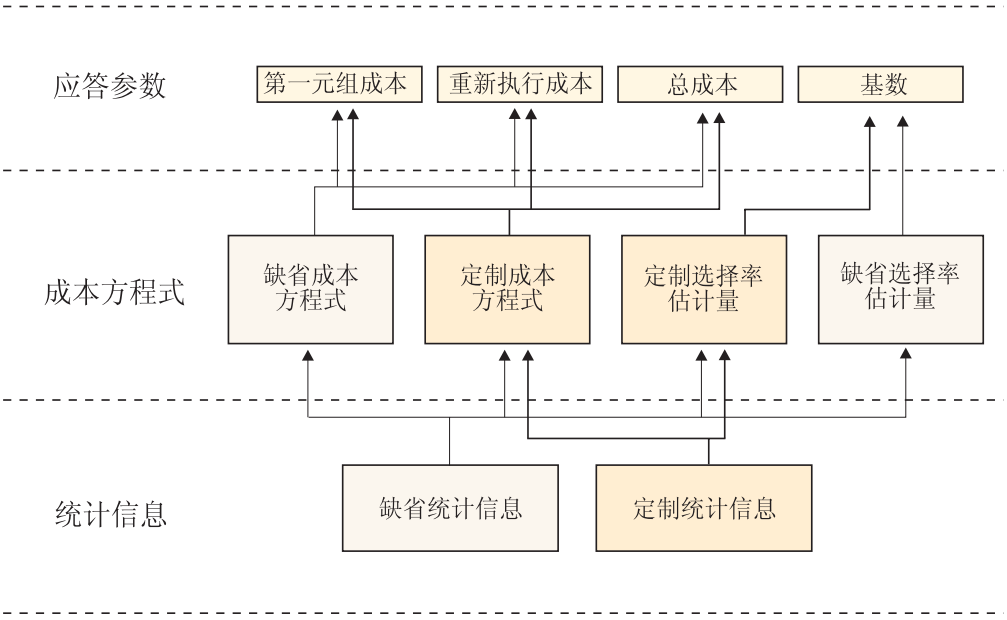

图 5. 成本计算是如何进行的

大致按复杂性、功能和灵活性的升序，包括:

1. 按原样接受缺省成本模型；或者
2. 覆盖计算昵称的四种统计信息（基数、平均设置成本、平均提交成本和平均高级成本）的方式；或者
3. 覆盖缺省选择率计算；或者
4. 通过再对“应答”对象分类来替换整个成本模型。

相关概念:

- 第 24 页的『针对数据源的查询的相对成本』
- 第 50 页的『查询规划的控制流』

联合系统的查询执行

一旦查询方案完成，联合服务器就可以运行查询。首先，联合服务器将指定给每个数据源的查询段分发至相应的包装器。同样，包装器将分段提交至数据源并检索其结果。联合服务器组合这些结果并对它们做进一步的处理。这些步骤都是典型的，并随特定源处理连接概念的方式（该源是通过查询语言还是通过一些其它 API 接受请求，以及源支持哪种结果集游标或迭代器（如果有的话）等等）而变化。第 15 页的图 6 引导您完成分发查询、执行查询以及检索其结果时通常采用的操作。

分发查询、执行查询以及返回其结果的典型过程:

1. 联合服务器将在 CREATE USER MAPPING 语句中指定的权限信息传递至包装器。
2. 联合服务器请求包装器建立与数据源的连接。
3. 包装器建立联合服务器请求的连接。
4. 包装器获取在此查询段的查询规划期间产生的“包装器执行描述符”。“包装器执行描述符”必须包含足够的信息才能将该查询提交给数据源。
5. 包装器将已转换的查询段提交至这些查询引用的数据源。然后，包装器获取该包装器要检索的结果集的迭代器。
6. 联合服务器从包装器请求一行结果。包装器有效地将该请求“转发”至相应的数据源。
7. 数据源执行查询段的一部分以返回请求的行。
8. 包装器检索请求的行。该包装器还将该行中的数据类型转换为联合服务器数据类型，并将已转换类型复制到 DB2 缓冲区中。
9. 联合服务器、数据源和包装器对每个连续的结果行重复步骤 7、8 和 9。
10. 包装器检索最后一行结果。在一些工作单元在不带引用的情况下运行后，联合服务器断开与数据源的连接。
11. 包装器断开与数据源的连接。

图 6. 分发查询、执行查询以及返回其结果的典型过程

相关概念:

- 第 8 页的『联合系统的查询处理』
- 第 5 页的『包装器模块』

将传递与包装器配合使用

传递允许应用程序将查询或其它请求直接提交给外部数据源。该查询或其它查询使用数据源本机查询语言。可使用传递象检索行和列一样检索数据。不能连接通过传递获取的结果或将其与来自其它源中的结果进行组合。传递的实现是可选的。它对调试连接性问题非常有用，并为应用程序提供了针对外部源执行管理命令的方法。

相关任务:

- 第 88 页的『远程 passthru 类』

第 2 章 开发包装器的概述

下列主题提供了包装器开发过程和用来开发包装器的包装器开发工具箱的概述。

包装器开发过程

作为特定数据源的包装器的编写者，您需要了解开发的基本流程才能创建包装器模块。

表 2. 包装器编写过程中的各个阶段

阶段	子任务
概念	了解联合概念
	了解一般包装器编写过程
设计	了解数据源
	开发数据源的关系模型
	决定每个联合构造的选项
	将联合构造映射至数据源
	决定数据源支持哪些类型的查询
	设计成本模型
	设计传递
	错误处理的设计
代码	编写包装器子类的代码
	为注册编码
	为初始化编码
	为查询规划编码
	为查询执行编码
	为传递编码
	考虑可移植性问题
文档	说明包装器
构建和打包	编译包装器
	链接包装器
	打包包装器
	安装包装器
测试	测试 SQL 语句
	调试和跟踪包装器
	测试包装器选项
	测试大范围查询

相关概念:

• 第 55 页的『开发包装器的典型过程』

- 第 57 页的『开发包装器的技巧』
- 第 18 页的『包装器开发工具箱』

包装器开发工具箱

DB2® Information Integrator 包括用于开发 C++ 和 Java™ 包装器的软件开发包（SDK）。

包装器开发工具箱包含:

- 样本 C++ 包装器
- 样本 Java 包装器
- 用于将包装器添加至 DB2 控制中心的工具和样本

缺省 Windows® 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

样本 C++ 包装器

表 3 显示每个平台的样本 C++ 包装器所在目录。

表 3. 样本 C++ 包装器的目录（按平台）

平台	包装器安装目录
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

样本 C++ 包装器包含:

- 显示包装器 API（包装器类声明）的头文件
- 允许包装器与联合服务器链接的文件
- 包装器公共库（提供的装入和调用定制包装器的库的存根库）
- 用来演示用于开发包装器的 C++ API 的用法的样本包装器源代码
- 用于构建样本包装器的样本 makefile。

样本 Java 包装器

表 4 显示每个平台的样本 Java 包装器所在目录。

表 4. 样本 Java 包装器的目录（按平台）

平台	包装器安装目录
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

样本 Java 包装器包含:

- 描述 Java API 类和方法的 Javadoc
- 用来演示用于开发包装器的 Java API 的用法的样本包装器源代码

用于将包装器添加至 DB2 控制中心的工具和样本

包装器工具箱包括帮助您将对定制包装器的支持添加至 DB2 控制中心的工具和样本文件:

- “开发 XML 配置文件”向导，创建用于将定制包装器添加至 DB2 控制中心中的选项的配置文件。表 5 显示每个平台上包含启动向导的文件的目录。

表 5. 用于启动“开发 XML 配置文件”向导的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%\bin\db2wrapperconfig.bat

- “开发 XML 配置文件”向导中的样本输出文件。表 6 显示每个平台上包含样本输出文件的目录。

表 6. “开发 XML 配置文件”向导中的样本输出文件的目录（按平台）

平台	包装器安装目录
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

- 基本发现工具，如果想要包装器支持 DB2 控制中心的发现功能部件，则可以使用此工具。该工具是一个简单的 Java 图形用户界面，显示已对包装器的数据源发现的任何内容。此工具也是随 DB2 控制中心提供的。表 7 显示每个平台上将该工具作为 Java .jar 文件提供的目录。

表 7. 基本发现工具的目录（按平台）

平台	包装器安装目录
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

- 此处提供的样本 Java 存储过程是内置发现如何帮助包装器编写者开发控制中心的插件的示例。表 8 显示哪个目录包含存储过程、用于编译存储过程的 makefile 和用于将标记文件安装到联合服务器中的脚本。

表 8. 样本 Java 存储过程的目录（按平台）

平台	包装器安装目录
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

相关概念:

- 第 17 页的『包装器开发过程』
- 第 55 页的『开发包装器的典型过程』

相关任务:

- 第 107 页的『将数据源添加至 DB2 控制中心』
- 『安装包装器开发工具箱』（*IBM DB2 Information Integrator 安装指南 Linux、UNIX 和 Windows 版*）

第 2 部分 设计数据源的包装器

本书的此部分指导您完成设计包装器所需的下列任务:

- 通过确定数据源特征设计包装器以有效地使用数据源
- 将数据源映射至联合构造
- 确定源可以接受的 SQL 构造
- 设计包装器的错误处理

第 3 章 确定数据源特征

在开始设计包装器之前，需要理解数据源。您需要回答下列问题:

- 数据源是否提供 API 选项？
- 每个接口支持哪些类型的操作？
- 数据源是否包含描述主要数据的模式或其它属性的元数据？
- 数据源可应答哪些类型的查询？
- 能否更有效地应答某些查询？
- 如果数据源有多个实例，则实例与实例之间有什么差异，又有什么始终相同？
- 数据源是否支持客户机与服务器的通信？
- 是否有可用于运行联合服务器的平台的数据源客户机？
- 数据源是否支持事务模型？如果支持的话，它又是否支持分布式（一阶段）落实协议？
- 数据源是如何认证用户的？
- 数据源是否支持大对象？

下列各节提供了上面每个问题的更多详细信息和示例。

数据源的 API 选择

当您研究数据源的 API 时，查找:

- API 支持的功能。例如，您能否提交需要通过此 API 提交的这类查询？API 是否允许包装器存取描述模式或由数据源管理的数据的统计属性的元数据？API 是否支持事务？尽管对非关系源的存取是只读的，但包装器还是可以参与在其中更新关系源的事务。
- API 需要的信息。例如，当包装器准备建立与数据源的连接时，API 需要什么信息？它需要什么信息才能使结果集的检索更加容易实现？

支持所选 API 的数据源客户机必须在想要运行 DB2 UDB 联合数据库服务器（它将存取该数据源）的每个平台上可用。

相关概念:

- 第 23 页的『数据源的接口支持的操作』
- 第 24 页的『数据源上的元数据』

数据源的接口支持的操作

使用下列问题以帮助您了解数据源的每个接口支持的操作:

- 数据源是否允许您有选择地根据数据值检索信息？数据源是否能应用谓词？
- 数据源是否允许您检索部分实体？数据源是否支持投影？

- 数据源是否能组合基于链接或与数据值匹配的多个连接集合中的数据？它是否能执行连接？
- 数据源是否支持不能用标准 SQL 表示的所有特殊搜索功能？对该源的搜索功能建模需要哪些映射函数？
- 数据源是否返回存储的数据值，或者它能否返回将这些值与其它值或常量组合起来的表达式的值？
- 谓词的数据源语义是否与联合服务器语义相匹配？

相关概念:

- 第 24 页的『数据源上的元数据』
- 第 24 页的『针对数据源的查询的相对成本』
- 第 27 页的『决定昵称和列选项』

相关任务:

- 第 23 页的『数据源的 API 选择』

数据源上的元数据

可从数据源获取的信息越多，注册源时数据库管理员需要完成的工作就越少。特别是，如果可从源获取该模式的详细信息，则可从 CREATE NICKNAME 语句中省略它们。同样地，有时可从包装器的成本模型（定制或缺省）使用的数据源统计元数据获取它们。

相关概念:

- 第 23 页的『数据源的接口支持的操作』
- 第 24 页的『数据源的多个实例』
- 第 25 页的『数据源的客户机／服务器通信』

针对数据源的查询的相对成本

了解数据源可应答的不同查询的相对成本非常重要。如果联合服务器中有一个查询段，通常有几种方法可执行数据源中此分段的全部或一部分。包装器必须能够为每个备用方法提供精确的成本估计。

相关概念:

- 第 23 页的『数据源的接口支持的操作』

相关任务:

- 第 23 页的『数据源的 API 选择』

数据源的多个实例

大部分数据源事实上并非同一类型。如果它们是同一类型，可能很难在包装器中编写连接至特定源及使用特定源所需的所有信息，包括模式。实际上，必须允许 DBA 为您的数据源的特定实例定制包装器。既然将通过 DDL 大量指定这类信息，包装器设计的一个重要部分就是确定在实例与实例之间有什么信息是不同的。可以使用此信息定义

一组将允许 DBA 指定需要什么内容的选项。如果模式在实例与实例之间是不同的，可以对其某些部分进行硬编码，然后通过 DDL 指定其余部分。

相关概念:

- 第 24 页的『数据源上的元数据』
- 第 25 页的『数据源的客户机 / 服务器通信』

数据源的客户机 / 服务器通信

包装器将与联合服务器在同一台计算机上运行，该服务器会将您的数据源合并到联合系统中。如果您希望存取的数据源在不同于联合服务器的另一计算机上运行，则必须使用数据源的客户机 / 服务器通信工具。包装器和整个联合服务器实质上是数据源的客户机。如果数据源不支持客户机 / 服务器通信，则必须在包装器本身中提供此功能或是另外提供。有时可通过在同一台计算机上运行数据源和联合服务器来解决该问题。

相关概念:

- 第 25 页的『数据源的事务模型和分布式落实协议』
- 第 25 页的『数据源中的用户认证』
- 第 26 页的『数据源中的大对象支持』

数据源的事务模型和分布式落实协议

事务是一种机制，用于使对数据库进行的一组更新原子化：要么整组更新同时进行，要么根本不进行任何更新。当事务涉及多个数据源时，这些源必须合作以确保跨越所有这些源的事务的可分性。DB2® Information Integrator 使用 XA 分布式落实协议以便在各个源之间协调事务管理。尽管 DB2 Information Integrator 的当前版本不支持对非关系数据源的更新，但如果数据源支持事务，则包装器必须参与事务管理。它必须参与事务管理以便在您的数据源中获取的锁定可在锁定它们的事务完成（无论成功还是失败）时删除。

相关概念:

- 第 25 页的『数据源的客户机 / 服务器通信』
- 第 25 页的『数据源中的用户认证』
- 第 26 页的『数据源中的大对象支持』

数据源中的用户认证

联合系统的客户机将使用任何受支持的认证方案对联合服务器进行认证。当联合服务器尝试以其自身名义存取数据源时，必须提供您的数据源识别并认为有效而接受的凭证。联合服务器将这些凭证作为“用户映射”存储在联合服务器目录中。设计包装器时，必须了解您的数据源期望什么凭证，例如，用户标识和密码。了解这一点后，可以定义允许这些凭证存储在联合服务器目录中的用户映射选项。

相关概念:

- 第 25 页的『数据源的客户机 / 服务器通信』

- 第 25 页的『数据源的事务模型和分布式落实协议』
- 第 26 页的『数据源中的大对象支持』

数据源中的大对象支持

大对象（LOB）数据类型被定义为字节序列，其大小可从 0 字节到大约 2G 字节。（LOB 数据类型包括 CLOB、BLOB 和 DBCLOB 数据类型）。如果数据源支持 LOB，则需要将您的包装器设计为支持它们并实现该包装器。

相关概念:

- 第 25 页的『数据源的客户机／服务器通信』
- 第 25 页的『数据源的事务模型和分布式落实协议』
- 第 25 页的『数据源中的用户认证』

相关参考:

- 第 83 页的『远程查询类』
- 『RemoteQuery class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Remote_Query 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

第 4 章 将数据源映射至联合构造

下列各节描述如何将数据源映射至关系模型。

对于每个联合概念，需要决定对数据源起作用的选项。

这些选项包括:

- 用于在联合服务器系统目录中存储特定于包装器的信息的类属（属性和值）对
- 为任何持久构建块（包装器、服务器、昵称和用户）指定。

  在昵称中，每个列也可具有选项。
- 由包装器编写者定义和介绍
- 在大多数情况下，联合服务器未解释

每个包装器都定义它自己的一组选项。

昵称的设计

不管源使用哪种数据模型来表示其数据，都必须将它映射至关系模型。然后，用户、数据库管理员或应用程序通过特定于数据源的 CREATE NICKNAME DDL 语句将此关系模型提供给联合服务器。

决定昵称和列选项

以下示例演示如何定义昵称和列选项。

考虑文件服务器的包装器。在包装器将查询段提交至服务器之前，它确定在该分段中引用其昵称的文件的路径和名称。为了可以实现此目的，将路径和名称的选项包括在定义昵称的同一语句中。

例如，假设具有位于目录结构 root/user/ 中的名为 EMPINFO 文件。可在 CREATE NICKNAME 语句上创建名为 PATH 的新选项。用户会将路径 `root/user/EMPINFO` 指定给 PATH 选项。

相关概念:

- 第 27 页的『将源数据的可查询集合映射至昵称』
- 第 28 页的『将分层数据结构映射至昵称』

将源数据的可查询集合映射至昵称

用户通过运行 CREATE NICKNAME 语句注册外部服务器数据（例如，表或视图）集合。该语句包括具有与联合服务器表的定义相同的特征的集合的定义。当注册外部服务器数据集合时，联合服务器在该模式中将其定义为联合服务器表。在语句中指定的昵称充当表的名称。在指定定义的其它部分（例如，列名和数据类型）之前，必须确

定与其相对应的集合的属性。什么属性与列相对应？哪种联合服务器数据类型与数据源属性的类型最相对应？回答这种问题时，可以在联合服务器表中将这些属性定义为它们的对应项。

假定 Oracle 数据库包含名为 EMPLOYEES 的表。在注册此表之前，需要确定它的哪些部分与联合服务器的表的各列相对应以及哪些部分与这些列的数据类型相对应。这是一项简单的任务，因为 Oracle 和联合服务器都遵循关系数据模型，很明显 Oracle 表的各列与联合服务器表的各列相对应并且 Oracle 列的数据类型与联合服务器数据类型相对应。

但是，如果数据源是文件服务器又将是什么情况？那么问题就是服务器中的文件是如何与联合服务器表相对应的？一种可能的方法是将每个文件看成是表的对应项、将每个文件记录看成是表行的对应项并将每个记录的字段看成是列的对应项。

相关概念:

- 第 28 页的『将分层数据结构映射至昵称』
- 第 27 页的『决定昵称和列选项』

将分层数据结构映射至昵称

许多数据源包含分层组织的数据集合。例如，一组“项目”包含在代表“部门”的实体内，而该实体可能是“分公司”的一组“部门”中的一个等等。但是，关系模型是平面的：表列本身不能是嵌套表。相反，必须为每种实体定义单独的表并将这些表连接起来才能浏览它们之间的分层关系。例如，要查找“部门”所属的“分公司”，或者要查找“部门”的所有“项目”。在将描述分层结构的数据的特征的源合并到 DB2® UDB 联合系统中时，将其分层集合建模为可连接在一起的一组相关昵称。

相关概念:

- 第 27 页的『将源数据的可查询集合映射至昵称』
- 第 27 页的『决定昵称和列选项』

将数据源中的数据类型映射至 DB2 通用数据库

“昵称”的各列必须具有基本 SQL 类型，象 INTEGER、VARCHAR() 和 DATE 等等。

DB2® Information Integrator 不支持用户定义的类型、单值类型和象昵称的各列的引用类型。而是必须使用其底层表示类型定义包含这种值的列。但是，可利用这些构造对昵称表示的数据定义视图。

对于外部资源支持的大部分数字类型，对相应 SQL 类型的映射是直接的。

但是，当决定表示数字值的最佳 SQL 类型时记住诸如精度、小数位和可允许值的范围之类的属性。将值的数据源表示法转换为该值的 DB2 UDB 表示法是包装器的职责，反之亦然。使用的不同类型越多，必须编写的转换代码越多。

对于具有字符数据类型的数据源属性，包装器的编写者必须选择是使用定长还是变长 SQL 类型表示该属性，并确定长度的相应上限。 字符数据类型的其它注意事项包括字符编码和代码页问题。一般地说，字符数据与联合数据库的指定代码页之间的转换由包装器负责。

要管理临时数据，将表示临时值的数据类型映射至下列 SQL 类型之一：TIME、DATE 或 TIMESTAMP。

DB2 UDB 提供运算符以便在多对 SQL 类型之间进行强制类型转型或转换。考虑通过使用基本类型（如 VARCHAR）表示数据源属性并定义视图以将此值强制类型转型或转换为具有更多语义的相应类型（如 FLOAT 或 DATE）来简化包装器。

相关概念:

- 第 27 页的『将源数据的可查询集合映射至昵称』
- 第 28 页的『将分层数据结构映射至昵称』

相关任务:

- 第 61 页的『将包装器的部件映射至类』

使用函数模板对数据源功能建模

您的数据源可能具有在 DB2 UDB 中没有的功能。在该情况下，您想要扩展 SQL 以对数据源的特殊功能建模。

例如，假定数据源是可以确定地区面积的“地理信息系统”。但 SQL 没有“面积”运算符，因此，您将此运算符建模为函数，并编写类似如下的查询:

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

由远程数据源执行的函数称为*定制函数*。在查询中使用映射函数的方法与“用户定义的函数”（UDF）是完全一样的。映射函数与 UDF 的不同之处在于 DB2 UDB 执行 UDF，而外部数据源执行定制函数。

定义定制函数只需要一个步骤。

1. 通过 DDL 创建函数模板：CREATE FUNCTION AREA(VARCHAR()) RETURNS INTEGER AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
   - 为了将您的定制函数与其它定制函数、UDF 或带有相同名称的 DB2 UDB 内置函数区分开来，考虑为映射函数定义单独的模式。

相关概念:

- 第 29 页的『使用伪列对数据源功能建模』

相关任务:

- 第 11 页的『联合查询的缺省成本模型』

使用伪列对数据源功能建模

通过关系接口显示数据源功能的另一种技术是通过伪列的使用。这些列被定义为用来控制查询的某些方面的昵称的部分。它们通常是由包装器而不是最终用户定义的。

示例将创建控制文本搜索模式的通配符的 `wildcard` 伪列。

另一个示例是 Blast 包装器的昵称的 MismatchPenalty 伪列。此伪列控制 Blast 搜索参数中的一个，但是其本身并不产生任何值。当该包装器调用 Blast 搜索时，格式为 MismatchPenalty = 5 的谓词成为命令行选项 -q5。

对于伪列，必须要记住以下几个问题:

- 如果用户引用选择列表中的伪列，则包装器必须返回该伪列的值，即使它是一个仅输入的数据。最好是 NULL。
- 并非所有关系运算符都适用于特定伪列。通常只有一个运算符（ “=”、 “<”或 “<=”）可用。与仅拒绝谓词相反，在查询规划期间，包装器必须检测无效运算符并拒绝整个查询。在稍后的示例中，联合服务器将尝试自己处理可能产生意外结果的谓词。
- 展示缺省行为的伪列可能会干扰具体查询表（MQT）。因为优化器不识别缺省行为，它可能会在不应使用 MQT 的情况下使用它，或者优化器可能会在应该使用 MQT 的时候未能使用它。在 MQT 定义中使用时，针对带有伪列的昵称的查询应指定所有谓词。

相关概念:

- 第 29 页的『使用函数模板对数据源功能建模』

相关任务:

- 第 11 页的『联合查询的缺省成本模型』

包装器的设计

包装器如何使用选项

包装器在下列几个方面会对选项信息起作用:

- 包装器验证在注册期间提交的 SQL 语句中指定的选项信息。
- 在包装器需要配置信息处理查询之前，联合服务器系统目录是包装器用来以选项值的形式存储此信息的固定位置。
- 它使用这种信息执行任务。例如，假定作为将错误消息从数据源路由至用户的先决条件，包装器需要知道是应路由所有这类消息还是仅路由严重错误的消息。可以创建用户可以对其指定表示这些备用项的值的选项。

在为包装器作好执行先前列示的操作的准备时，需要:

- 决定包装器是否需要应作为新选项的值存储的信息
- 确定包装器应如何验证 SQL 语句为联合服务器系统目录提供的选项信息

一定不能定义以 DB2_ 开头的选项名称。这些名称保留供 IBM® 使用。

将包装器需要的选项信息存储在联合服务器目录中

在为包装器创建选项时，需要决定如何获取该选项的值。您有三个选择:

- 指定在注册中使用的其中一个 SQL 语句的选项和值
- 在包装器中对选项和值进行硬编码，并将它们作为在这些语句中指定的信息的补充传递至联合服务器系统目录

- 从数据源本身获取某些选项值。这通常不适用于包装器选项，但也许对任何其它可能性适用。例如，服务器可能具有通用的 API，您可从该 API 获取有关其维护级别的信息。如果包装器将此记录为选项值，包装器也许能够使用它以避免某些已知会导致问题的查询构造。您也许还能从数据源获取昵称选项和列选项。

建议允许用户通过 SQL 语句指定选项值。如果用户不提供值，则使用硬编码的值或从数据源获取的值作为缺省值。如果用户从未打算更改值，则应该将其硬编码至包装器而不将其定义为选项。

相关概念:

- 第 31 页的『决定包装器选项』

相关任务:

- 第 31 页的『对数据源定义 CREATE WRAPPER 语句』

决定包装器选项

通常，包装器对象不需要选项。此级别的选项影响在包装器内定义的所有服务器。一个可能选项或一组可能选项可控制调试和诊断由包装器产生的输出。

相关概念:

- 第 30 页的『包装器如何使用选项』

相关任务:

- 第 31 页的『对数据源定义 CREATE WRAPPER 语句』

对数据源定义 CREATE WRAPPER 语句

映射包装器概念意味着了解系统在初始化包装器和数据源客户机库时需要的东西。

包装器通常不需要配置选项。但是，如果包装器需要使用配置选项，可以定义它们。当需要全局参数以初始化数据源客户机库时需要这样做（这种情况很少见，但应该会有）。例如，可能必须指定全局“本地语言支持”（NLS）语言环境或全局包装器调试级别。

在 Java 中，要为数据源定义 CREATE WRAPPER DDL 语句，至少需要知道不受防护包装器类的名称。

例如，如果包装器类名为 my.package.MyUnfencedWrapper，相应的 CREATE WRAPPER 语句可能为:

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

相关概念:

- 第 31 页的『决定包装器选项』
- 第 30 页的『包装器如何使用选项』

服务器的设计

决定服务器选项

以下示例演示如何决定要定义哪些服务器选项。

考虑包含电子表格的数据源的包装器。假定包装器通过 TCP/IP 与这些数据源通信。要连接至特定数据源，包装器需要知道主机名和数据源的 TCP/IP 端口的号码。相应地，将对应主机名和端口号的选项包括在定义数据源的服务器名的同一 SQL 语句中。

例如，假设数据源 A 具有主机名 Peter 且通过端口 40 可用。可在 CREATE SERVER 语句上创建两个新的选项 HOSTNAME 和 PORT。您的用户会将主机名 Peter 指定给 HOSTNAME 选项，并将号码 40 指定给 PORT 选项。

相关概念:

- 第 30 页的『包装器如何使用选项』

相关任务:

- 第 32 页的『对数据源定义 CREATE SERVER 语句』

对数据源定义 CREATE SERVER 语句

映射服务器意味着定义与数据源相匹配的 CREATE SERVER 语句。

需要确定映射至服务器概念的数据源构造。例如，考虑文档管理系统。在一个企业内，在不同位置可能有多个独立文档归档，每一个都包含几个文档集合。应用程序必须连接至特定归档才能搜索它包含的集合。对于此数据源，可在联合系统中将每个归档建模为服务器。

于是需要确定配置包装器以便它可连接至并使用该服务器所需的配置信息。这些都是 CREATE SERVER 语句上的选项。例如，在文档归档的情况下，NODE 选项可能是该归档所驻留的计算机的名称。

CREATE SERVER 语句上还有两个可用数据项。这些数据项属于服务器类型和版本。当某一组数据源具有大量常见行为而且变化很少时，服务器类型和版本可用来控制包装器的操作。这样就不需要对每个变体和版本设置不同的包装器。

例如，与数据源的先前信息相关联的 CREATE SERVER 语句可能为:

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

相关概念:

- 第 33 页的『决定用户映射选项』

相关任务:

- 第 31 页的『对数据源定义 CREATE WRAPPER 语句』
- 第 33 页的『对数据源定义 CREATE USER MAPPING 语句』

用户映射的设计

决定用户映射选项

数据源很少请求除 REMOTE_AUTHID 和 REMOTE_PASSWORD 以外的用户映射选项。

为了增强安全性，联合服务器会在将名为 REMOTE_PASSWORD 的任何用户映射选项的值存储在联合服务器系统的目录中之前对其进行编码。联合服务器在将其返回至包装器之前会再次对其进行译码。强烈建议对需要联合服务器以安全存储的认证信息使用此选项名。

您必须自己实现这些选项，这是因为“用户”子类的缺省版本不支持它们。

对于 Windows® 上的数据源，Windows 域对于网络连接和认证来说都是必需的。这是您应该支持的一个选项。例如，要记录域信息，包装器可定义名为 REMOTE_DOMAIN 的选项。

相关概念:

- 第 31 页的『决定包装器选项』
- 第 32 页的『决定服务器选项』
- 第 27 页的『决定昵称和列选项』

相关任务:

- 第 33 页的『对数据源定义 CREATE USER MAPPING 语句』

对数据源定义 CREATE USER MAPPING 语句

映射用户为数据源定义 CREATE USER MAPPING 语句。

例如，如果您正在为文件归档编写包装器，则您希望每个归档用户都有一个联合用户。您想要定义以下配置信息:

- REMOTE_AUTHID: 用户的用户标识
- REMOTE_PASSWORD: 用户的密码

生成的 CREATE USER MAPPING 语句可能类似如下:

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

相关概念:

- 第 33 页的『决定用户映射选项』

相关任务:

- 第 31 页的『对数据源定义 CREATE WRAPPER 语句』
- 第 32 页的『对数据源定义 CREATE SERVER 语句』

第 5 章 确定数据源可以接受的 SQL 构造

必须考虑数据源可对哪些 SQL 构造（包括映射函数）求值。必须考虑下列问题:

- 数据源可接受哪些类型的头表达式。头表达式是在 SELECT 列表中发现的表达式吗？
- 数据源可以接受哪些类型的谓词？
- 数据源可以接受哪些类型的连接？
- 数据源可以接受哪些函数？

下列各节更详细地讨论这些注意事项。

确定数据源可以接受的头表达式

包装器考察 SELECT 列表中联合服务器提供的查询段的每个表达式（也称为头表达式）。如果数据源可以处理该表达式，则包装器通过将该头表达式包括在应答中来指示这一点。如果数据源不能处理整个头表达式，则包装器通过从应答中排除该头表达式来拒绝它。

有一个局限性: DB2 Information Integrator V8.2 仅下推头表达式的列引用。

相关任务:

- 第 35 页的『确定数据源可以接受的谓词』
- 第 36 页的『确定数据源可以接受的连接』
- 第 36 页的『确定数据源可以接受的函数』

确定数据源可以接受的谓词

包装器考察联合服务器提供的查询段的每个谓词表达式。如果数据源可以处理该表达式，则包装器通过将该谓词表达式包括在应答中来指示这一点。如果数据源不能处理整个谓词表达式，则包装器通过从应答中排除该谓词表达式来拒绝它。

每个接受的谓词的数据源语义必须与联合服务器语义相匹配。包装器不能控制联合服务器是否选择下推某些谓词。任何谓词可由远程数据源或联合服务器处理。如果两者之间的谓词语义不同，根据是否下推谓词，最终结果也可能会不同。

在将查询段传递至包装器之前，联合服务器将谓词重写为称为合取范式的格式。将该谓词重新安排到由 AND 运算符连接的谓词表达式列表中。

例如:

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

Expr1、expr2 和 expr3 可能是包括 NOT、AND 和 OR 运算符的复杂谓词表达式。包装器必须接受或拒绝位于顶级的表达式。例如，如果第一个谓词 expr1 为 col1 = 0 OR col2 > 3，并且数据源可以对 col1 = 0 求值但不能对 col2 > 3 求值，则包装器拒绝整个 expr1 谓词。

相关任务:

- 第 35 页的『确定数据源可以接受的头表达式』
- 第 36 页的『确定数据源可以接受的连接』
- 第 36 页的『确定数据源可以接受的函数』

确定数据源可以接受的连接

连接请求包括两个或更多昵称以及将各个昵称关联在一起的一个或多个谓词。包装器检查连接请求并确定数据源是否支持所需的特定连接。

相关任务:

- 第 35 页的『确定数据源可以接受的头表达式』
- 第 35 页的『确定数据源可以接受的谓词』
- 第 36 页的『确定数据源可以接受的函数』

确定数据源可以接受的函数

定义谓词的表达式通常包含一个或多个函数调用。如果包装器接受包含 DB2 UDB 实现的函数的调用的谓词，则数据源运行的函数的语义必须与联合服务器的语义完全相同。

定义谓词的表达式通常包含一个或多个函数调用。谓词中的函数可能是 DB2 UDB 实现的函数或运算符（例如，>、LIKE、+ 和 CONCAT），或者它们可能是特定于您通过定义函数模板注册的包装器的定制函数。如果包装器接受包括 DB2 UDB 也可对其求值的函数的调用的谓词，则数据源运行的该函数的语义必须与 DB2 UDB 的语义相同。

数据源打算支持的任何 DB2 UDB 函数的语义必须与 DB2 UDB 本身的语义相同。这会有一些问题，比如，整理顺序和空值处理。

相关任务:

- 第 35 页的『确定数据源可以接受的头表达式』
- 第 35 页的『确定数据源可以接受的谓词』
- 第 36 页的『确定数据源可以接受的连接』

第 6 章 错误处理的设计

在准备编写包装器时，需要决定该包装器应该如何报告它遇到的错误。

如果从 DB2 UDB 支持函数或 DB2 UDB 实现的成员函数*接收*非零返回码，

1. 则恢复并复位返回码（如果可能的话）。
2. 否则，在本地清除并将相同代码返回给调用者。

如果*发现*“新的”错误，

1. 在 SQL Messages 参考指南中查找最接近的代码。
2. 确定可以从消息正文中使用的标记的数目。最大消息长度为 70 -（标记数目减 1）。
3. 报告错误。
4. 在本地清除并将代码返回至调用者。

Java API 使用 Java 异常来处理错误并且提供专门的类来抛出已转换为 DB2 错误消息的异常。例如:

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

表 9 列示错误的类别，每个类别后跟有关适合于它的代码和消息的信息。

表 9. 错误的类别和相关联的错误消息

事件	消息号	消息内容
认证失败	SQL1403	提供的用户名或密码不正确。应该在数据源支持认证时使用。
认证，用户映射，缺少	SQL1827	未定义从本地授权标识 *auth-ID* 至服务器 *server-name* 的用户映射。尽管此消息的描述文本专指 ALTER 和 DROP 语句，但是在需要用户映射而又没有用户映射时也会使用此消息。
空白截断	SQL1844	在远程数据源与联合服务器之间截断了列 *column-name* 的结尾空白。这是在将数据从远程数据源传送至服务器时截断了结尾空白时产生的警告消息。在由于代码页转换而发生截断的情况下，使用 SQL1580。
空白截断，代码页，转换，数据截断	SQL1580	在执行从代码页 *source-code-page* 转换为代码页 *target-code-page* 时截断了结尾空白。目标区域的最大大小为 *max-len*。源字符串长度为 *source-len* 并且它的十六进制表示为 *string*。在截断了空白时将使用的警告（Wrapper_Utilities::convert_codepage 返回 CP_CONV_BUFFER_SMALL 并且剩余字符为空白。）
代码页，转换，数据截断	SQL0334	在执行从代码页 *source* 转换为代码页 *target* 时发生了溢出。目标区域的最大大小为 *max-len*。源字符串长度为 *source-len* 并且它的十六进制表示为 *string*。当数据转换中发生截断 / 溢出时使用。当 Wrapper_Utilities::convert_codepage 返回 CP_CONV_BUFFER_SMALL 时应该发出此消息。

表 9. 错误的类别和相关联的错误消息 (续)

事件	消息号	消息内容
代码页，转换，无效数据	SQL0191	因为 MBCS 字符变成碎片，所以发生了错误。当 Wrapper_Utilities::convert_codepage 返回 CP_CONV_DBCS_TRUNCATE 时应该产生此消息。
代码页，转换，不受支持	SQL0332	
列，数据类型，本地，正在改变	SQL0270	不支持函数（原因码 = *reason-code*）。代码 65 表示“不允许将昵称本地类型从当前类型改变为指定类型”。可以对 ALTER NICKNAME ALTER COLUMN LOCAL TYPE 语句使用此消息。
列，数据类型，本地，无效长度精度小数位	SQL0604	列、单值类型、结构化类型、结构化类型的属性、函数或类型映射 *data-item* 的长度、精度或小数位属性无效。当验证昵称列规范（对于 CREATE 或 ALTER）时，如果长度、精度或小数位的其中一个不正确，则使用此消息。*Data-item* 应该为列名。
列，数据类型，本地，不受支持	SQL3324	列 *name* 的类型 *type* 不受支持。可用于昵称列的无效本地类型。
列，运算符，不允许	SQL1843	*nickname-name.column-name* 昵称列不支持 *operator-name* 运算符。如果包装器限制可应用于昵称列的运算符，则使用此消息。
列，远程，找不到	SQL0205	未在 *object-name* 中定义列或属性名 *name*。当选项引用本地列或远程列或属性但目标不存在时使用此消息。这比 SQL0204 更具体。
列，太多	SQL0680	对表、视图或表函数指定了太多列。如果数据源不支持和 DB2 UDB 一样多的列，则此消息很有用。
通信，失去连接，TCP/IP，一般错误	SQL30081	检测到通信错误。正在使用的通信协议: *protocol*。正在使用的通信 API: *interface*。检测到错误的位置: *location*。检测错误的通信功能: *function*。特定于协议的错误代码: *rc1*、*rc2* 和 *rc3*。对任何 TCP/IP 通信错误使用此消息，但 gethostbyname() 未发现解析名称（请参阅 SQL1336）除外。这包括指示连接断开的一些特殊情况。
通信，TCP/IP，找不到主机	SQL1336	找不到远程主机 *hostname*。当 TCP/IP gethostbyname() 无法返回主机时使用此消息。有关其它 TCP/IP 错误，请参阅 SQL30081。
转换，数字，超出范围	SQL0405	因为数字文字 *literal* 的值超出范围，所以无效。当从远程数据值转换为本地数据值时使用此消息。
转换，数字，溢出	SQL0413	在数字数据类型转换期间发生了溢出。在转换时发生溢出的情况下使用。
数据截断	SQL1844 和 SQL1845	在远程数据源与联合服务器之间截断了列列名的数据。这是与 SQL1844 警告相对应的错误消息。如果由于代码页转换而发生截断，则使用 SQL0334。
数据源错误	SQL1822	从数据源 *error-code* 接收到意外错误代码 *data-source-name*。相关联的文本和标记为 *tokens*。正如消息所述，这是由数据源报告的错误（对于此错误，没有相应的 DB2 UDB 消息。）不应该将它用于内部错误，也不应该用于由包装器本身检测到的错误。

表 9. 错误的类别和相关联的错误消息 (续)

事件	消息号	消息内容
日期时间，无效语法	SQL0180	日期时间值的字符串表示法不正确。当从远程源字符串转换为 DB2 Information Integrator 日期、时间或时间戳记值时使用此消息。
日期时间，超出范围	SQL0181	日期时间值的字符串表示法超出范围。此消息与 SQL0180 相同。
环境变量，缺少	SQL5182	尚未设置必需的环境变量 *variable-name*。当 db2dj.ini 中需要变量而但变量不存在时，应发出此消息。
名称，列，未定义	SQL0204	*name* 是未定义的名称。当具有选项（例如，REMOTE_TABLE）但引用的对象不存在时使用此消息。有关对列或属性的引用的信息，另请参阅 SQL0205。
名称，列，未定义	SQL0205	未在 *object-name* 中定义列或属性名 *name*。当选项引用本地列或远程列或属性但目标不存在时使用此消息。
选项，添加，无效	SQL1840	不能将 *option-type* 选项 *option-name* 添加至 *object-type* 对象。对包装器可能为供其自身使用而生成的选项使用此消息，或者对只能在 CREATE 时间而不是 ALTER 时间添加的选项使用此消息。
选项，冲突	SQL1846	*object-name-1* 对象的 *option-type-1* 选项 *option-name-1* 与 *object-name-2* 对象的 *option-type-2* 选项 *option-name-2* 相冲突。当两个（或更多）选项或选项值相冲突时使用此消息。注意，这可能是对象类型之间的冲突（例如，带有特定昵称选项的列选项无效。）
选项，删除，无效	SQL1837	类型为 *option-type* 的必需选项 *option-name* 不能用于不能删除的 *对象名*。当用户尝试删除（DROP）不能删除的选项时发出此消息。
选项，重复	SQl1884 和 SQL1885	对 *object-name* 多次指定了 *option-name*（*option-type* 选项）。某些选项（如某些列选项）只能指定一次。作为示例，XML 包装器昵称的 PRIMARY_KEY 列选项只能对一个列指定。在此上下文中，*object-name* 将是昵称。
选项，无效	SQL1881	*option-name* 对于 *object-name* 不是有效的 *option-type* 选项。选项无效。如果根本不识别该选项，或者如果该选项在当前上下文中无效（尽管 SQL1884 可以更好地处理后一种情况），则可以使用此消息。
选项，无效值	SQL1882 和 SQL1842	不能将 *option-type* 选项 *option-name* 设置为 *object-name* 的 *option-value*。选项值无效。如果该值与另一个选项值相冲突，则改为使用 SQL1846N。如果该值是对某个其它实体（例如，远程对象甚至是本地目录对象）的引用，则使用 SQL0204。
选项，缺少	SQL1883	*option-name* 是 *object-name* 的必需 *option-type* 选项。未指定必需的选项。将此消息用于 CREATE。对于 DROP，使用 SQL1837。
选项，设置服务器选项，无效	SQL1841	对于 *object-name* 对象，不能更改 *option-type* 选项 *option-name* 的值。与 SQL1840 相似，但用于 SET。

表 9. 错误的类别和相关联的错误消息 (续)

事件	消息号	消息内容
选项，未定义	SQL1886	*operation-type* 操作无效，因为尚未对 *object-name* 定义 *option-type* 选项 *option-name*。当用户尝试设置（SET）或删除（DROP）未定义的选项时，DDL 进程将检测到此消息。
服务器，类型，无效	SQL1816	包装器 *wrapper-name* 不能用来访问尝试对联合数据库定义的数据源（*server-type server-version*）的 *type-or-version*。当在 CREATE SERVER 和 ALTER SERVER 语句中验证服务器类型或版本时使用。组件框架可处理此消息。
服务器，版本，无效	SQL1817	CREATE SERVER 语句未标识想要对联合数据库定义的数据源的 *type-or-version*。当 CREATE SERVER 语句需要类型或版本但未指定时使用。组件基础结构可处理此消息。
SQL，不受支持	SQL0142	不支持该 SQL 语句。问题在于没有进一步的信息，这使得用户很难理解。带有其中一个预定义原因码或替代原因码的文本的 SQL30090N 更有用。
用户标识，缺少	SQL1027	

相关概念:

- 第 31 页的『决定包装器选项』
- 第 32 页的『决定服务器选项』
- 第 27 页的『决定昵称和列选项』
- 第 33 页的『决定用户映射选项』

第 3 部分 开发和说明包装器

本书的此部分指导您完成开发和说明包装器所需的下列任务:

- 根据设计编写包装器代码。
- 说明包装器以便用户可以快速地了解如何使用它。

第 7 章 数据流概述

联合查询处理和涉及的对象

下列各节提供典型查询流的详细概述以及有关查询处理涉及的各种对象的生命周期的注意事项。

联合查询的典型流程

以下列表概述典型的查询流程。子类名称中的 XX 表示特定包装器。

1. 客户机应用程序提交需要存取外部数据源的查询。
2. 联合服务器确定（根据查询中的昵称）哪些包装器与此查询有关。下列步骤 3 至 12 适用于这类包装器的每一个。
3. 联合服务器装入不受防护类属包装器库（C++）或类（Java）并调用引导入口点（挂钩函数）。引导程序函数仅在 C++ 中可用。
4. 联合服务器创建不受防护类属包装器类的包装器定制子类的实例。具体地说:

   C++: 联合服务器调用引导入口点（挂钩函数），然后该入口点将创建包装器的 Unfenced_XX_Wrapper 类的实例。

   Java™:
   联合服务器调用 UnfencedXXWrapper 类的构造函数，然后该构造函数创建该类的实例。
5. 联合服务器对新的包装器对象调用初始化方法。
6. 联合服务器确定查询中涉及哪些包装器的服务器（根据昵称）。下列步骤 7 至 12 适用于这类服务器的每一个。
7. 联合服务器对包装器对象调用服务器创建方法以创建包装器的服务器子类的实例。

   C++: Unfenced_XX_Server（Unfenced_Generic_Server 的子类）

   Java:
   UnfencedXXServer（UnfencedGenericServer 的子类）
8. 联合服务器对新的服务器对象调用初始化方法。
9. 联合服务器通过针对服务器对象的查询规划方法将包含查询段的“请求”对象提交至包装器。将在此过程中创建包装器的昵称子类的实例。
10. 包装器分析查询段并将一个或多个“应答”对象返回至联合服务器。每个“应答”对象都包含一个已接受查询段、成本估计和执行描述符。对联合服务器优化器生成的每个查询段重复步骤 9 至 12。
11. 联合服务器对服务器对象调用选择率估计方法。它传递包含查询段中不被包装器接受的所有谓词的列表。
12. 包装器使用定制选择率估计方法（如果提供了的话）或缺省方法（如果没有提供的话）估计谓词的组合选择率。
13. 联合服务器优化器为整个查询选择方案。

14. 重复步骤 2 至 8，用受防护的对象代替不受防护的对象。也就是说，创建并初始化所有必需的 Fenced_XX_Wrapper 和 Fenced_XX_Server 对象（在 Java 中为 FencedXXWrapper 和 FencedXXServer）。下列步骤 15 至 25 适用于每个服务器。
15. 联合服务器对服务器对象调用用户创建例程。它这么做是为了创建包装器的用户子类的实例以表示当前连接的用户。

    C++: Fenced_XX_User（Fenced_Generic_User 的子类）

    Java:

    FencedXXRemoteUser（FencedGenericRemoteUser 的子类）
16. 联合服务器对新的服务器对象调用初始化方法。
17. 联合服务器对服务器对象调用连接创建方法以创建包装器的远程连接子类的实例。

    C++: Remote_Connection（XX_Connection）

    Java:

    RemoteConnection（XXConnection）
18. 联合服务器对连接对象调用连接方法。包装器连接至远程源（如果必要的话）。
19. 联合服务器对连接对象调用查询创建方法以创建包装器的远程查询子类的实例。

    C++: Remote_Query（XX_Query）

    Java:

    RemoteQuery（XXQuery）
20. 联合服务器对查询对象调用查询初始化方法。包装器使用“应答”中的执行描述符（从查询对象获取）将查询提交至远程源。
21. 联合服务器对查询对象调用 fetch 方法以从包装器请求结果行。包装器将结果行返回至联合服务器.
22. 重复步骤 21 直到不再返回行。
23. 如果联合服务器想要重新执行该查询（可能使用已改变的参数值），则联合服务器重复步骤 20 至 22。
24. 联合服务器对查询对象调用查询清除方法。包装器清除游标或其它与查询相关的资源。
25. 联合服务器破坏查询对象。
26. 应用程序终止工作单元。
27. 联合服务器对连接对象调用落实／放弃方法。对每个服务器重复此步骤。
28. 应用程序与数据库断开连接。
29. 联合服务器对连接对象调用连接清除方法。联合服务器关闭与数据源的连接。
30. 联合服务器破坏连接对象。对每个服务器重复服务器 29 和 30。
31. 实例化的受防护／不受防护包装器、服务器、用户或昵称对象被破坏。

相关概念:

- 第 52 页的『查询执行的控制流』
- 第 53 页的『包装器与外部服务器之间的通信』
- 第 50 页的『查询规划的控制流』

相关任务:

- 第 45 页的『注册的控制流』

- 第 50 页的『初始化的控制流』

相关参考:

- 第 45 页的『联合查询中涉及的对象的生命周期』

联合查询中涉及的对象的生命周期

- 如果受防护包装器、服务器、用户或昵称对象已存在，则不会创建它们。即，如果为了规划或执行应用程序先前提交的查询已经创建了它们，则不需要再创建了。
- 当正在运行的工作单元没有使用与应用程序相关联的受防护或不受防护包装器、服务器、用户或昵称对象时，联合服务器会破坏它们。如果发生这种情况，联合服务器会在应用程序提交另一查询时根据需要创建对象。在应用程序仍然连接至数据库期间破坏这类对象的通常原因是应用程序发出了 ALTER 或 DROP DDL 语句。ALTER 或 DROP DDL 语句改变该对象表示的包装器、服务器、用户或昵称的属性。
- 当联合服务器完成工作单元时，它不会立即清除与远程数据源的连接或破坏 XX_Connection 对象。该连接将保持打开状态一段时间以防应用程序想要执行存取同一远程源的另一 UOW。在应用程序完成不引用该源的若干工作单元之后，或当应用程序与数据库断开连接时，联合服务器将启动清除。

相关概念:

- 第 53 页的『包装器与外部服务器之间的通信』
- 第 43 页的『联合查询的典型流程』

过程的控制流

下列各节描述下列过程的控制流:

- 注册
- 初始化
- 查询规划
- 查询执行

注册的控制流

本主题论述当注册数据源及其关联构造时对存储在联合服务器系统目录中的信息建模的类。它还概述注册过程和如何验证包装器的注册过程的一般控制流。

标准和选项信息

当联合服务器注册构造时，这两种信息中的一种或两种信息转入联合服务器系统目录。*标准信息*由给定的类型的大多数的构造所共有的规范组成。例如，当注册服务器时，对服务器提供用户指定的名称和将在 CREATE SERVER 语句上用来访问服务器的包装器的名称。因为每个服务器定义都需要这些信息，所以用户指定的名称和相关联的包装器的名称可被认为是服务器的标准信息。

除了联合服务器为构造的所有实例而保留的标准信息之外，联合服务器系统目录还包含其本质在不同类型的数据源之间有所不同的信息。例如，IBM 提供存取存储在某些类型的文件中的信息的包装器。要存取文件，包装器必须知道文件的名称。与文件名相似的信息是特定于这种数据源的。例如，在存取 Oracle 源时，它将没有任何意义。联合

服务器通过将值指定给称为选项的变量来将这种信息存储在目录中。使用 DDL 指定选项名称和值。因此，文件包装器支持昵称的选项变量 FILE_PATH，可使用 CREATE NICKNAME 语句指定它的值。

标准和选项信息的类

几种类对有关数据源和关联构造的所有信息建模：标准和选项。这些类被全体称为*构造信息类*。它们包括：

表 10. C++ 和 Java 构造信息类

C++	Java	描述
Wrapper_Info	WrapperInfo	关于包装器的模型信息
Server_Info	ServerInfo	关于数据源的模型信息
Nickname_Info	NicknameInfo	关于数据源中的数据集合（例如，表或视图）的模型信息
Column_Info	ColumnInfo	关于数据源的数据集合中的值列（或者这种列的等价物）的模型信息
User_Info	UserInfo	关于使用数据源的权限的模型信息

如何使用构造信息类的对象

联合服务器和包装器按下列方法使用构造信息类的对象：

- 它们在系统将信息从 SQL 语句和包装器传送至联合服务器系统目录时充当信息的容器。例如，包装器在 CREATE SERVER 语句中验证信息并且联合服务器对此信息及该包装器提供的任何补充信息进行编目。系统将已验证的信息和补充信息存储到联合服务器系统目录的 Server_Info 对象（在 Java 中为 ServerInfo）中。
- 它们在将信息从该联合服务器系统目录传送至表示构造的对象（例如，表示特定数据源的“服务器”对象）时充当信息的容器。当系统进行传送时，接收对象使用此信息初始化自身。

  每个包装器从联合服务器系统目录获取授权信息以连接至数据源。系统将此信息传递到包装器的 User_Info 对象（在 Java 中为 UserInfo）中。类似地，该包装器包含有关查询引用的昵称的已编目信息，以便该包装器可以标识数据源处的相应数据集合。系统将此信息传递到包装器的 Nickname_Info 对象（在 Java 中为 NicknameInfo）中。

CREATE WRAPPER、CREATE SERVER 和 CREATE USER MAPPING 的流

以下主题使用 C++ 类和方法名。如果要开发 Java 包装器，则替代为相应的 Java 方法名。

1. 应用程序提交 CREATE YY DDL 语句，其中 YY = WRAPPER、SERVER 或 USER MAPPING。
2. 联合服务器对 DDL 语句进行语法分析、检查基本语法是否正确、检查先前存在的实体是否具有相同的名称并确定哪个包装器负责要创建的实体。
3. 联合服务器创建并初始化创建此类型的新实体所需的所有父对象。要创建新的“用户映射”，必须先实例化相关的包装器和服务器对象。

表 11. C++ 和 Java 包装器和服务器对象

C++	Java
Unfenced_XX_Wrapper	UnfencedXXWrapper
Unfenced_XX_Server	UnfencedXXServer

4. 联合服务器创建相关包装器的服务器、用户或包装器子类的新实例。联合服务器是通过对父对象调用相应的创建方法实现这一点的。对于通过装入相应的库并调用钩子函数创建的 C++ 包装器，联合服务器未对父对象调用相应的创建方法。对于 Java 包装器，不受防护包装器子类是直接装入的。此对象将成为类 Unfenced_XX_YY 的实例，其中 XX 表示对包装器指定的名称，而 YY 表示对要创建的对象的类型指定的名称。例如:

   C++: Unfenced_File_Wrapper

   Java:

   UnfencedFileWrapper
5. 联合服务器创建与要创建的实体相对应的信息子类的实例。此对象将成为类 YY_Info 的实例并包含应用程序的 DDL 语句中的所有信息。
6. 联合服务器对 Unfenced_XX_YY 对象调用创建验证方法，将该信息对象传递至包装器以进行检查。
7. 该包装器通过检查 YY_Info 对象来检查 DDL 语句中的信息的一致性和正确性。特别是，该包装器检查选项值是否显示为强制的、有效的并且相互一致。
8. 包装器可选择构造另一个 YY_Info 对象，称为 delta-info 对象。包装器可以通过在 delta-info 对象中提供覆盖或补充通过 DDL 语句指定的信息的信息来对选项指定值或控制正在创建的实体的其它方面。
9. 联合服务器将 DDL 语句中的信息与 delta-info 对象（如果有的话）中的信息合并在一起，并将生成的配置存储在联合服务器系统目录中。
10. 联合服务器对 Unfenced_XX_YY 对象调用初始化方法。

CREATE NICKNAME 的流

以下主题使用 C++ 类和方法名。如果要开发 Java 包装器，则替代为相应的 Java 方法名。

CREATE NICKNAME 的注册流添加对昵称类使用创建验证方法的额外验证步骤。既然受防护类具有与数据源通信的方法，这就允许包装器连接至数据源以收集未在 DDL 中显式指定的配置信息。

1. 应用程序提交 CREATE NICKNAME DDL 语句。
2. 联合服务器创建适当的 Unfenced_XX_Wrapper 和 Unfenced_XX_Server 父对象以及表示新昵称的 Unfenced_XX_Nickname 对象。
3. 联合服务器创建 Fenced_XX_Wrapper、Fenced_XX_Server 和 Fenced_XX_Nickname 对象。
4. 联合服务器根据 DDL 语句上的信息创建 Nickname_Info 对象。
5. 联合服务器对 Fenced_XX_Nickname 对象调用创建验证方法，传递 Nickname_Info 对象以进行检查。
6. 包装器可选择连接至外部数据源并从数据源获取附加信息。此信息包括模式信息和／或统计信息。

7. 包装器可选择创建备份 Nickname_Info 对象，它包含补充或覆盖 DDL 的信息（可能从数据源获取）。
8. 联合服务器将 DDL 语句中的信息与备份 Nickname_Info 对象（如果有的话）中的信息合并在一起。
9. 由 Fenced_XX_Nickname 验证方法创建的已合并 Nickname_Info 成为 Unfenced_XX_Nickname 验证方法的输入，且余下的步骤与包装器、服务器和用户映射注册流的步骤 6 至步骤 10 相同。

ALTER WRAPPER、ALTER SERVER、ALTER NICKNAME 和 ALTER USER MAPPING 的流

以下主题使用 C++ 类和方法名。如果要开发 Java 包装器，则替代为相应的 Java 方法名。

ALTER 流与 CREATE 流不同，因为包装器必须确定通过 DDL 指定的新选项值是否与保持不变的当前值一致，以及相互之间是否一致。

1. 应用程序提交实体 YY 的先前列示的 DDL 语句中的一个（其中 YY 是“包装器”、“服务器”或“用户映射”中的一个）。
2. 联合服务器对 DDL 语句进行语法分析、检查基本语法是否正确、检查以确保具有指定名称的实体存在并确定哪个包装器负责要改变的实体。
3. 联合服务器创建并初始化实例化代表要改变的实体的对象所需的所有父对象。要实例化 Unfenced_XX_User 或 Unfenced_XX_Nickname，必须先实例化相关的 Unfenced_XX_Wrapper 和 Unfenced_XX_Server 对象。如果对象已经存在，则联合服务器将不会创建它们。
4. 联合服务器根据用户想要改变的实体创建相关的包装器对象 Unfenced_Generic_Wrapper、Unfenced_Generic_Server 或 Unfenced_Generic_User 子类的新实例。联合服务器是通过对父对象调用合适的创建方法来实现此目的的，但通过装入适当的库和调用钩子函数所创建的包装器的情况除外。此对象将是类 Unfenced_XX_YY 的一个实例，其中 XX 是包装器而 YY 是正创建的对象的类型（例如，Unfenced_File_Wrapper 和 Unfenced_Blast_Server 等等）。如果对象已经存在，则联合服务器将不会创建它们。在 Java 中，没有库也没有钩子函数。联合服务器反而会将相应的包装器子类实例化。
5. 联合服务器初始化新创建的 Unfenced_XX_YY 对象。
6. 联合服务器创建与用户想要改变的实体相对应的 Catalog_Info 子类的实例。此对象将是类 YY_Info 的一个实例（例如，Server_Info 和 Remote_User_Info）并包含应用程序的 DDL 语句中的所有信息。
7. 联合服务器对 Unfenced_XX_YY 对象调用改变验证方法，将 YY_Info 对象传递至包装器以进行检查。
8. 该包装器通过检查 YY_Info 对象来检查 DDL 语句中的信息的一致性和正确性。特别是，包装器检查以确保新的选项值与其值保留不变的现有选项值一致，并确保没有删除强制选项。
9. 包装器可选择构造另一个 YY_Info 对象，称为 delta-info 对象。包装器可以通过在 delta-info 对象中提供覆盖或补充通过 DDL 语句指定的信息的信息来对选项指定值或控制正在改变的实体的其它方面。
10. 联合服务器将 DDL 语句中的信息与 delta-info 对象（如果有的话）中的信息合并在一起，并将生成的配置存储在联合服务器系统目录中。

11. 联合服务器破坏 Unfenced_XX_YY 对象，该对象是使用不反映此 DDL 语句的旧目录信息初始化的。当再次需要该对象时，联合服务器重新创建它并使用已更新的信息将其初始化。

验证注册过程的一般步骤

验证包装器的注册过程遵循下列一般步骤:

CREATE DDL

以下主题使用 C++ 类和方法名。如果要开发 Java 包装器，则替代为相应的 Java 方法名。

仅当包装器开发者定义了特定选项时，对任何 CREATE DDL 的验证才应是必需的。下面是验证注册的一般过程。有关每个包装器构建块的特定验证过程，请参阅下列几节。

1. 对于 XX_Info 对象中的每个 Catalog_Option:
   a. 通过调用 is_reserved_xx_option()（Java API 在 CatalogOption 类上提供具有类似功能的方法 isReserved()）来确定它是不是保留的 DB2 UDB 选项。如果它是保留选项，则跳至下一步。
   b. 确定其它是否是受包装器支持的选项；如果不是，则发出 SQL1881N 错误消息。
   c. 确定选项值对该选项是否有效；如果无效，则发出 SQL1882N 错误消息。
2. 如果用户省略所有必需的选项，则发出 SQL1883N 错误消息。
3. 确定所有已提供的选项值相互之间是否一致（如果适用的话）。使用 SQL1882N 错误消息报告无效或不一致的值。
4. 如果必要，提供下列附加选项:
   a. 确定是否分配了 delta XX_Info 对象。如果需要，则分配该对象。
   b. 必要时使用 XX_Info 的 add_option 方法添加新选项。

ALTER DDL

与 CREATE 相似，仅当包装器开发者定义了特定于包装器的选项时，它才是必需的。ALTER 处理与 CEATE 处理有两个地方明显不同：首先，验证例程必须考虑包装器的当前状态。 XX_Info 对象中提供的信息将仅仅是有关哪一个包装器是新的或是哪一个包装器正在更改的信息。其次，验证例程必须注意每个选项的操作参数。最重要的是，如果用户尝试“删除”必需的选项，则验证例程一定会发出 SQL1881N 错误消息。此外，尝试检查正在“删除”的选项的值将导致诸如引擎崩溃且伴随有寻址错误之类的问题。

相关概念:

- 第 52 页的『查询执行的控制流』
- 第 53 页的『包装器与外部服务器之间的通信』
- 第 50 页的『查询规划的控制流』
- 第 43 页的『联合查询的典型流程』

相关任务:

- 第 37 页的第 6 章，『错误处理的设计』
- 第 50 页的『初始化的控制流』

- 『改变包装器』（《*联合系统指南*》）

相关参考:

- 『ALTER WRAPPER statement』（*SQL Reference, Volume 2*）

初始化的控制流

下面概述初始化过程的控制流。

此概述使用 C++ 对象名。如果要开发 Java 包装器，则替代为相应的 Java 对象名。

事实上此流很一般，适用于作为 Fenced_Generic 或 Unfenced_Generic 基本类（例如，Unfenced_Blast_Nickname 和 Fenced_File_User 等等）的子类的实例的所有对象。

1. 联合服务器对其父对象使用适当的创建方法（或对包装器使用钩子函数）来对包装器定义的子类“X”（例如 Fenced_Excel_Server）创建实例。
2. 联合服务器创建 YY_Info（例如 Server_Info）对象，该对象包含有关该对象表示的实体的所有已知信息，就象存储在相应的联合服务器系统目录中一样。所有 Catalog_Info 对象都包括 Catalog_Option 对象列表，这些对象提供对相应实体定义的所有选项的名称和值。Nickname_Info 对象包括 Column_Info 对象列表，每个对象对应昵称的每一列。
3. 联合服务器对在步骤 1 中创建的“X”对象调用初始化方法，传递在步骤 2 中创建的 YY_Info 对象。
4. 包装器可选择从 YY_Info 对象抽取信息（例如，选项值）并将其存储在“X”对象的成员变量中以便更有效地存取。联合服务器复制相应的 YY_Info 对象并将其连接至“X”对象。对于“昵称”对象（受防护的或不受防护的），联合服务器不复制 Nickname_Info 对象。

相关概念:

- 第 52 页的『查询执行的控制流』
- 第 53 页的『包装器与外部服务器之间的通信』
- 第 50 页的『查询规划的控制流』
- 第 43 页的『联合查询的典型流程』

相关任务:

- 第 45 页的『注册的控制流』

查询规划的控制流

下面概述查询规划过程流。

此概述使用 C++ 对象名。如果要使用 Java™ 开发包装器，则替代为相应的 Java 对象名。

该流程概述假定联合服务器正要求包装器使用一个或多个头表达式和谓词规划单个表查询段。联合服务器先规划单个表分段（后跟连接源中两个昵称的分段），然后进行三相连接等等。对于这些较复杂的分段，“请求”中将会有附加量词句柄并且“请求”中的一些谓词句柄将表示连接谓词。

1. 联合服务器创建描述查询段的“请求”对象，并对适当的 Unfenced_XX_Server 对象调用规划方法。

2. 包装器为此请求创建“应答”对象。如果包装器支持定制成本模型，则这将是包装器定义的“应答”子类的一个实例。如果包装器使用缺省成本模型，则该包装器实例化基本“应答”类。
3. 包装器获取表示查询段所覆盖的昵称的量词的句柄（Java API 使用实例而不是句柄）。
4. 通过使用句柄，包装器向联合服务器要求其 Unfenced_Generic_Nickname 子类 Unfenced_XX_Nickname 的实例，该子类与在查询段中找到的昵称相对应。
5. 联合服务器创建并初始化 Unfenced_XX_Nickname 对象。
6. 必要时包装器会从 Unfenced_XX_Nickname 对象获取有关昵称列和配置等等的信息。
7. 包装器将量词的句柄添加至“应答”。
8. 包装器为查询段中的其中一个头表达式获取句柄。
9. 包装器使用句柄以获取描述该头表达式的 Request_Exp 对象。
10. 包装器确定数据源能否计算 Request_Exp 表示的头表达式的值。包装器按降序对 Request_Exp 树中的每一项以递归方式重复步骤 12 直到该包装器作出决定为止。
11. 如果数据源可以计算整个头表达式，则包装器将头表达式的句柄添加至“应答”对象。
12. 对请求中的每个附加头表达式重复步骤 9 至 12。
13. 包装器为查询段中的其中一个谓词获取句柄。
14. 包装器使用句柄以获取描述该谓词的 Request_Exp 对象。
15. 包装器确定数据源是否可以处理 Request_Exp 表示的整个谓词。包装器将需要在 Request_Exp 树中递归向下以作出此决定。
16. 如果数据源可以处理该谓词，包装器会将该谓词的句柄添加至“应答”对象。
17. 对请求中的每个附加谓词重复步骤 15 至 17。
18. 如果数据源以与 DB2® Information Integrator 整理相匹配的特定顺序返回行，包装器会将顺序描述添加至“应答”。
19. 包装器检查“应答”表示的查询子段并生成持久表示法以存储在包装器执行描述符中。
20. 包装器将执行描述符存储在“应答”中。
21. 如果包装器希望为此查询段生成备用方案，则对每个备用方案重复步骤 9 至 23。
22. 包装器将“应答”对象列表返回至联合服务器.
23. 联合服务器对“应答”对象调用成本计算方法以计算相应的包装器方案的成本和基数。
24. 如果包装器使用定制成本模型，则包装器的成本计算方法可选择引用从 Unfenced_XX_Nickname 对象获取的定制统计信息来计算成本和基数。否则，缺省实现将通过使用缺省成本模型计算这些值。
25. 对每个附加“应答”对象（如果有的话）重复步骤 26 至 27。

相关概念:

- 第 52 页的『查询执行的控制流』
- 第 53 页的『包装器与外部服务器之间的通信』
- 第 43 页的『联合查询的典型流程』

相关任务:

- 第 45 页的『注册的控制流』
- 第 50 页的『初始化的控制流』

查询执行的控制流

下面概述查询执行过程。

此查询使用 C++ 对象名。如果要使用 Java™ 开发包装器，则替代为相应的 Java 对象名。

1. 联合服务器对包装器的 Remote_Query 子类 XX_Query 的实例调用 query-initiation 方法。
2. 包装器包含 XX_Query 对象中的包装器执行描述符。使用描述符中的信息，包装器准备在远程源处执行查询段。
3. 包装器从 XX_Query 对象获取输出 Runtime_Data_List 并确定从数据源返回的数据与包装器返回至联合服务器的数据之间的映射。
4. 如果查询段有参数，则包装器从输入 Runtime_Data_List（从 XX_Query 对象获取）的相应元素获取参数值。
5. 包装器将参数值（如果有的话）从 SQL（DB2 UDB）表示法转换为数据源期望的表示法。
6. 包装器在远程源处启动查询段的执行。
7. 联合服务器对 XX_Query 对象调用 fetch 方法。
8. 包装器从数据源检索至少一行数据。包装器缓冲附加行。
9. 包装器将结果值从数据源提供的表示法转换为联合服务器期望的 SQL 表示法。它通过对输出 Runtime_Data_List 的适当元素调用方法来将每个转换值复制到 DB2® 缓冲区中。
10. 重复步骤 7 至步骤 9，直到数据源没有要返回的行为止。
11. 联合服务器调用查询清除方法 XX_Query。状态标志指示 DB2 是否将查询段作为同一应用程序查询的一部分重新执行。
12. 如果重新执行查询，则联合服务器将对 XX_Query 方法调用查询重新执行方法。重复步骤 4 至步骤 11 直到联合服务器不再请求重新执行。

相关概念:

- 第 53 页的『包装器与外部服务器之间的通信』
- 第 50 页的『查询规划的控制流』
- 第 43 页的『联合查询的典型流程』

相关任务:

- 第 45 页的『注册的控制流』
- 第 50 页的『初始化的控制流』

包装器与外部服务器之间的通信

包装器参与查询处理的某些方法是将查询段转换为数据源理解的请求以将这些子查询提交至数据源并检索数据源返回的结果行。不受防护类属服务器子类的方法规划这些任务。远程查询子类的方法执行该执行操作；并且您在确定如何在方法间分布这些任务方面有相当大的决定权。为举例说明:

示例 1

当文件服务器的包装器开始处理引用列且不包含任何谓词的查询段中的 SELECT 语句时，下列示例开始。

1. 在查询规划期间，优化器检查查询段并根据该分段给包装器发送请求。包装器检查该请求并返回应答及包装器方案，该方案让优化器知道包装器可以对数据源执行的操作和不能对包装器执行的操作。优化器为包括包装器方案的执行创建查询方案。
2. 在查询执行期间，优化器将包装器方案发送给包装器。
3. 包装器对数据源执行该方案。
4. 根据包装器执行描述符提供的选项的值，包装器确定包含与昵称相对应的数据集的数据源文件的名称和位置。
5. 包装器检查包装器方案以确定将要检索的列的名称。
6. 根据包装器执行描述符提供的选项的值，包装器确定文件中的哪个顺序列与查询中的每个指定列相对应。
7. 根据包装器执行描述符提供的选项的值，包装器确定查询中指定的每列的终止定界符。
8. 包装器打开文件并开始读取数据。它使用先前确定的信息解析并格式化要返回至联合服务器的行。

示例 2

当文档服务器的包装器开始处理查询段中的 SELECT 语句时，下列示例开始。此分段引用三种数据源构造：文档的资源库、称为*属性*的构造和文本搜索函数。在联合服务器系统目录中，联合服务器将资源库定义为 DB2® UDB 表、DB2 UDB 将属性定义为这些表的各列并且函数映射至函数模板。

1. 在查询规划期间，优化器检查查询段并根据该分段给包装器发送请求。包装器检查该请求并返回应答及包装器方案，该方案让优化器知道包装器可以对数据源执行的操作和不能对包装器执行的操作。优化器为包括包装器方案的执行创建查询方案。
2. 在查询执行期间，优化器将包装器方案发送给包装器。
3. 包装器对数据源执行该方案。
4. 根据包装器执行描述符提供的选项的值，包装器确定文档的哪个资源库与昵称相对应。
5. 包装器检查包装器方案以确定将要检索的列的名称。
6. 根据包装器执行描述符提供的选项的值，包装器确定哪个属性与查询中指定的每个列相对应。
7. 包装器在包装器执行描述符中标识文本搜索函数。
8. 使用先前确定的信息（例如，文档的资源库的名称、属性的名称和文本搜索谓词），包装器以文档服务器的查询语言生成语句。

9. 包装器将该查询提交至文档服务器的 API，并开始读取要返回至联合服务器的数据。

相关概念:

- 第 52 页的『查询执行的控制流』
- 第 50 页的『查询规划的控制流』
- 第 43 页的『联合查询的典型流程』

相关任务:

- 第 45 页的『注册的控制流』
- 第 50 页的『初始化的控制流』

相关参考:

- 第 45 页的『联合查询中涉及的对象的生命周期』

第 8 章 使用包装器类进行开发

下列各节描述如何开发在开发包装器时需要使用的包装器和类。

开发包装器的典型过程

本主题描述开发包装器的典型过程。在开始开发包装器之前，首先:

- 熟悉在注册中使用的 SQL 语句、需要再细分的基本类以及数据源的 API。
- 确定如何将数据源处的数据集合建模为关系表。
- 确定如何在 DB2® UDB 目录中表示有关数据源处的数据集合的元数据。
- 必要时执行其它规划活动。

开发包装器时，通常执行下列步骤:

1. 实现并测试包装器子类:

   C++: Unfenced_Generic_Wrapper 和 Fenced_Generic_Wrapper

   Java™:
   UnfencedGenericWrapper 和 FencedGenericWrapper

   通过运行 CREATE WRAPPER 语句测试您的实现。检查在 SYSCAT.WRAPPERS 和 SYSCAT.WRAPOPTIONS 目录视图中得出的结果条目。
2. 实现并测试服务器类:

   C++: Unfenced_Generic_Server 和 Fenced_Generic_Server

   Java:
   UnfencedGenericServer 和 FencedGenericServer

   通过运行 CREATE SERVER 语句并检查在 SYSCAT.SERVERS 和 SYSCAT.SERVEROPTIONS 目录视图中得出的结果条目来测试您的实现。
3. （可选）如果包装器在昵称验证期间需要权限并与数据源联系，则实现并测试用户类。

   C++: Unfenced_Generic_User 和 Fenced_Generic_User

   Java:
   UnfencedGenericRemoteUser 和 FencedGenericRemoteUser

   通过运行 CREATE USER MAPPING 语句并检查在 SYSCAT.USEROPTIONS 目录视图中得出的结果条目测试您的实现。
4. （可选）如果包装器在昵称验证期间与数据源联系，则实现远程连接子类。

   C++: Remote_Connection

   Java:
   RemoteConnection
5. 如果想要数据源支持 passthru，则实现远程 passthru 类:

   C++: Remote_Passthru

Java:

RemotePassthru

通过运行引用数据源中的数据集合的 passthru 请求来测试远程 passthru 子类和远程连接类的实现。

6. 实现昵称子类:

   C++: Unfenced_Generic_Nickname 和 Fenced_Generic_Nickname

   Java:

   UnfencedGeneric 昵称和 FencedGeneric 昵称

   通过对几个外部服务器数据集合运行 CREATE NICKNAME 语句并检查在 SYSCAT.TABLES、SYSCAT.TABOPTIONS、SYSCAT.COLUMNS 和 SYSCAT.COLOPTIONS 目录视图以及 SYSTAT.TABLES 和 SYSTAT.COLUMNS 目录表中得出的结果条目测试实现。如果想要用户使用与 DB2 UDB 不相同的数据源函数，则创建与该函数相对应的函数模板。

7. （可选）如果包装器在昵称验证期间需要权限并且不连接至数据源，则实现并测试用户类。

   C++: Unfenced_Generic_User 和 Fenced_Generic_User

   Java:

   UnfencedGenericRemoteUser 和 FencedGenericRemoteUser

   通过运行 CREATE USER MAPPING 语句并检查在 SYSCAT.USEROPTIONS 目录视图中得出的结果条目测试您的实现。

8. 如果在步骤 4 中未进行，则实现远程连接子类。

   C++: Remote_Connection

   Java:

   RemoteConnection

9. 实现远程查询子类。

   C++: Remote_Query

   Java:

   RemoteQuery

   要测试此类和远程连接类的实现，提交引用数据源中的数据集合的联合查询。

10. 说明包装器。

相关概念:

- 第 57 页的『开发包装器的技巧』

相关任务:

- 第 58 页的『可信和设防方式进程环境』
- 第 63 页的『包装器类』
- 第 66 页的『服务器类』
- 第 72 页的『用户类』

子类和方法的实现

在熟悉包装器接口时，考虑下列内容:

- 要使用哪些基本类。您需要知道:
  - 表 12 显示需要创建您自己的子类实现的基本类。

表 12. 需要子类实现的基本类

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
Remote_Query	RemoteQuery

  - 如果正在实现您自己的成本模型，则需要下列子类:
    - 应答
  - 如果包装器支持传递，则需要子类:
    - Remote_Passthru（使用 Java 的 RemotePassthru）
  - 需要使用 IBM® 的某些类的实现。
- 在您创建的实现中要包括的内容。您需要知道:
  - 必须使用您自己的定制实现来覆盖某些方法的缺省实现。
  - 可以选择是使用方法的某些缺省实现还是使用您自己的定制实现覆盖它们。
  - 可以添加自己的新方法。注意，当您想这样做时，可以构建新方法来调用您正在实现的父类的方法。
  - 您可以添加新数据成员。

相关任务:

- 第 75 页的『应答类』
- 第 88 页的『远程 passthru 类』

开发包装器的技巧

编写包装器时，应知道:

- Wrapper_Utilities 类（在 Java™ 中为 WrapperUtilities）提供服务以帮助您工作。例如，如果遇到代码问题，可以使用类的 trace_data 方法。trace_data 方法生成一条记录，该记录可帮助您定位并标识该问题。
- 可以生成几个版本的包装器，第一个版本限于基本服务，而后续版本逐渐有更多的功能并且更有效。

相关概念:

- 第 57 页的『子类和方法的实现』
- 第 55 页的『开发包装器的典型过程』

可信和设防方式进程环境

在查询执行期间，联合服务器可通过创建多个子代理（线程或进程）以并行方式存取数据源来提高性能。可更容易地隔离正在子代理环境中执行的代码。此隔离使得在出故障的情况下不会导致失去系统的完整性。由于此原因，将查询执行中涉及的包装器部分与查询规划中涉及的部分分隔开来。

因此，包装器将分成两个部分:

- *不受防护*部分涉及查询规划。
- *受防护*部分涉及查询执行。

因此，必须将每个包装器构建块指定为包装器的受防护部分或不受防护部分。在某些情况下，必须在两者之间分割构建块。如果需要与外部源通信，则函数进入受防护部分。

处理环境会根据是用 C++ 还是用 Java 开发包装器而有细微差别。

C++ 处理环境

C++ 包装器是作为两个独立的库链接的，一个包含包装器的不受防护部分的类实现，另一个包含受防护部分的类实现。在称为*设防方式*的情况下，包装器的受防护部分装入到称为“设防方式”进程（FMP）的特殊隔离子代理中。包装器的不受防护部分装入到主联合服务器代理进程中。第 59 页的图 7 演示设防方式进程技术模型。

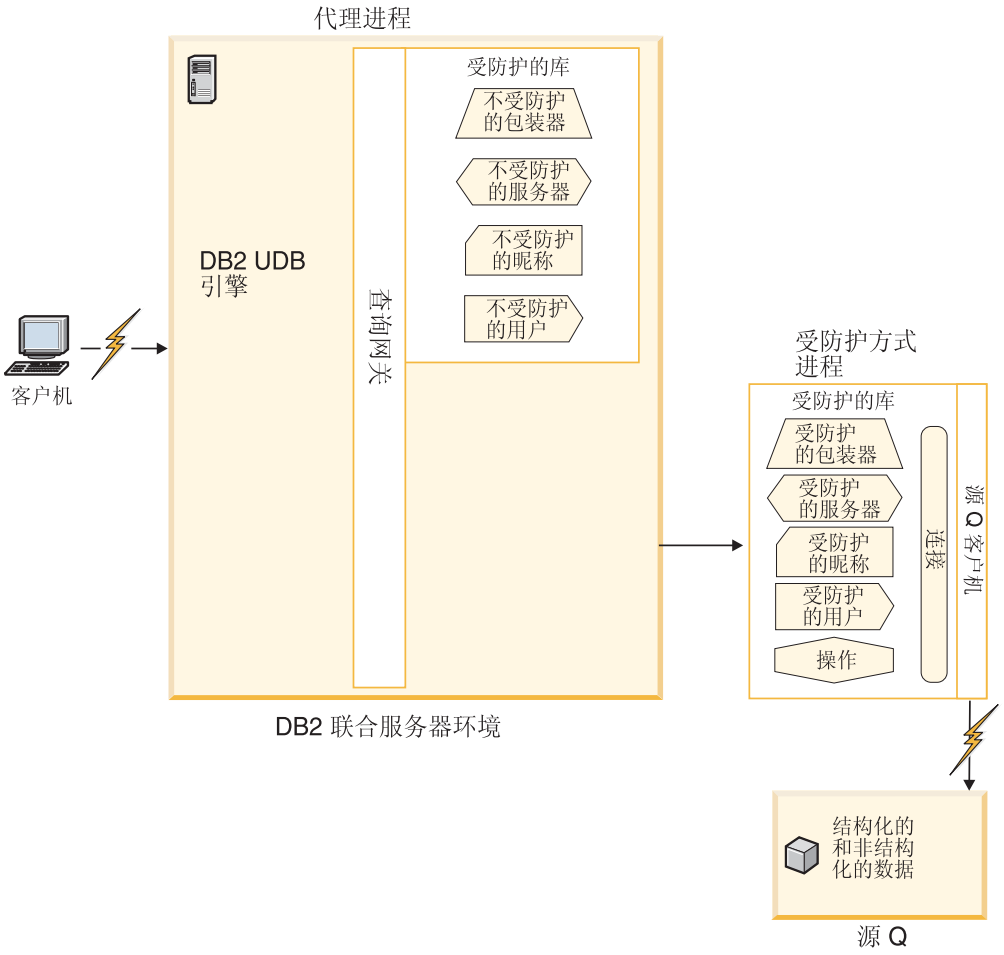

图 7. C++ 的设防方式进程技术模型

图 7显示包含包装器的受防护和不受防护部分的两个不同库和装入这两个库的不同区域；联合服务器环境中的不受防护部分和设防方式进程中的受防护部分。

当并行性不再是问题时，如果不使用子代理进程并且所有函数都在联合服务器环境中运行，性能将会更好。因为牺牲了隔离的安全性，所以此方式称为*可信方式*。第 60 页的图 8 演示可信方式进程技术模型。

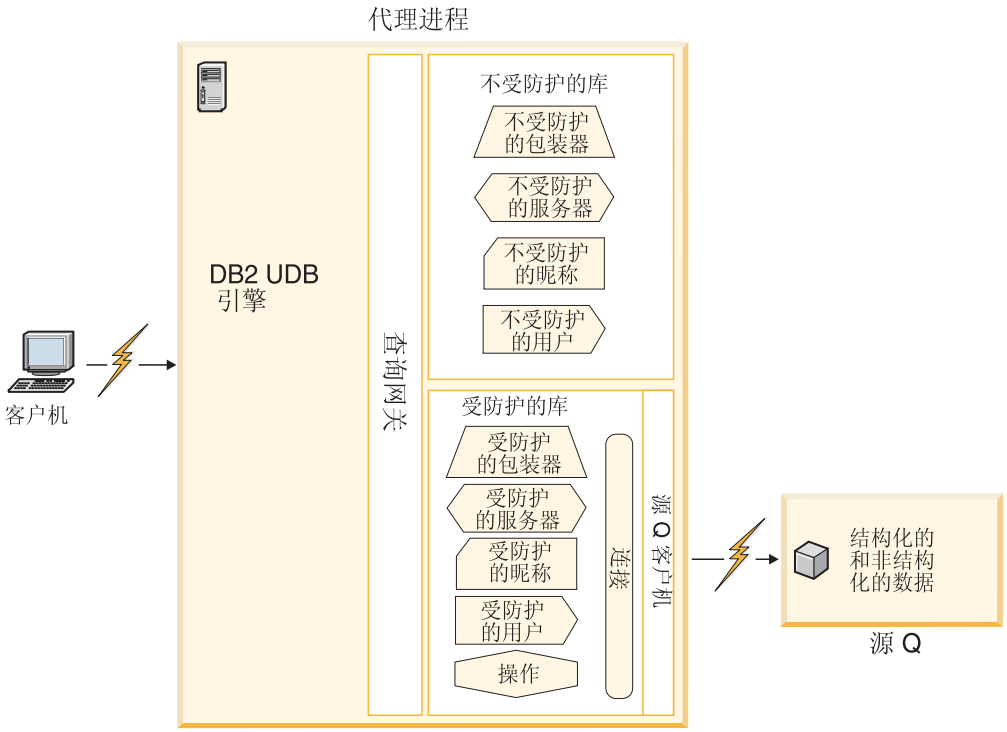

图 8. C++ 的可信方式进程技术模型

图 8显示在可信方式中，包装器的受防护部分和不受防护部分都将装入到联合服务器环境中。

由于 DB2 Information Integrator 仅支持 V8.1 中的可信方式执行，所以您必须设计包装器以支持可信方式和设防方式执行。

Java 处理环境

用 Java 开发的包装器总是整体在设防方式进程中运行。受防护和不受防护 Java 类在设防方式进程中运行。第 61 页的图 9 演示 Java 包装器的设防方式处理环境。

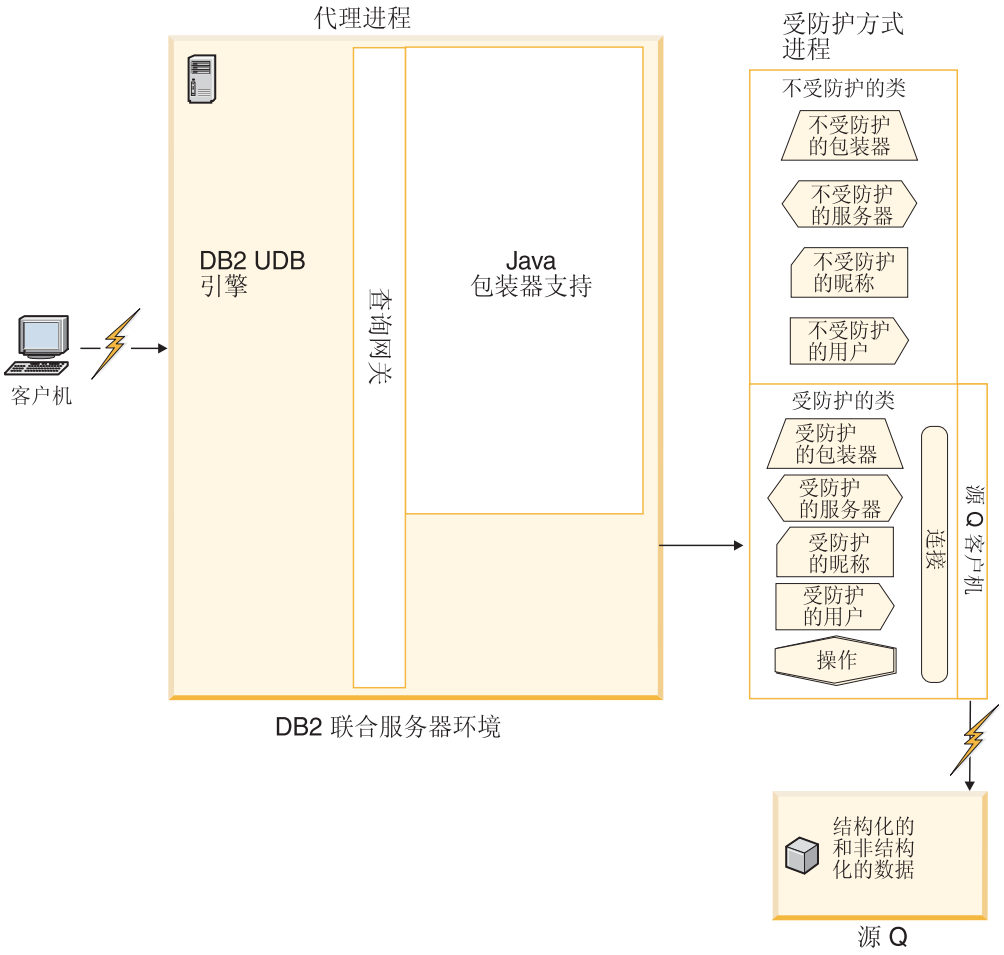

图 9. Java 的设防方式进程技术模型

图 9 显示在设防方式进程（而不是联合服务器环境）中运行的受防护类和不受防护类。

相关概念:

- 第 63 页的『包装器与数据源之间的通信的类』
- 第 57 页的『子类和方法的实现』
- 第 55 页的『开发包装器的典型过程』
- 第 57 页的『开发包装器的技巧』

相关任务:

- 第 61 页的『将包装器的部件映射至类』

将包装器的部件映射至类

许多构建块具有对它们来说是不受防护的“端”和受防护的“端”，您必须使用相应的类编写它们。

不受防护的端参与查询规划。当联合服务器处于设防方式时，查询执行阶段涉及受防护端以保护联合服务器免受数据源的环境影响。

表 13. C++ 基本包装器构建块的不受防护类和受防护类

构建块	C++ 不受防护类	C++ 受防护类
包装器	Unfenced_Generic_Wrapper	Fenced_Generic_Wrapper
服务器	Unfenced_Generic_Server	Fenced_Generic_Server
昵称	Unfenced_Generic_Nickname	Fenced_Generic_Nickname
用户	Unfenced_Generic_User	Fenced_Generic_User
连接	无	Remote_Connection
操作	无	Remote_Operation

表 14. Java 基本包装器构建块的不受防护类和受防护类

构建块	Java 不受防护类	Java 受防护类
包装器	UnfencedGenericWrapper	FencedGenericWrapper
服务器	UnfencedGenericServer	FencedGenericServer
昵称	UnfencedGenericNickname	FencedGenericNickname
用户	UnfencedGenericRemoteUser	FencedGenericRemoteUser
连接	无	RemoteConnection
操作	无	RemoteOperation

相关概念:

- 第 55 页的『开发包装器的典型过程』
- 第 57 页的『开发包装器的技巧』

相关任务:

- 第 58 页的『可信和设防方式进程环境』

第 9 章 用于编写包装器的类

本节描述用来编写包装器的类。

包装器与数据源之间的通信的类

表 15 显示对与数据源的连接建模的基本类和数据源执行的操作。

表 15. 对与数据源的连接建模的基本类

C++ 基本类	Java™ 基本类
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

与连接相似，数据源操作也是瞬态的。因此，联合服务器不会在联合服务器系统目录中存储它们的相关信息。

相关概念:

- 第 57 页的『子类和方法的实现』
- 第 55 页的『开发包装器的典型过程』
- 第 57 页的『开发包装器的技巧』

相关任务:

- 第 58 页的『可信和设防方式进程环境』
- 第 61 页的『将包装器的部件映射至类』
- 第 82 页的『远程连接类』
- 第 85 页的『运行时数据类』
- 第 87 页的『运行时数据描述类』
- 第 88 页的『远程 passthru 类』

相关参考:

- 第 83 页的『远程查询类』

包装器类

下列各节描述不受防护包装器和受防护类属包装器类。

Unfenced_Generic_Wrapper 类

C++: 在包含包装器代码的共享库中调用函数 UnfencedWrapper_Hook 会创建不受防护包装器子类的实例。当联合服务器第一次装入包装器的不受防护共享库时，该联合服务器会调用此函数。如果 DDL 操作更改与包装器有关的信息，则在其下次使用之前联合服务器会破坏并重新创建包装器对象。

Java: 当联合服务器装入不受防护包装器类时，将自动创建该类的实例。特定于包装器的不受防护包装器类的名称在 CREATE WRAPPER 语句中被指定为 UNFENCED_WRAPPER_CLASS 选项的值。

不受防护类属包装器基本类实现保留以下信息:

- 包装器名称。
- 包装器类型: "N" 表示非关系。
- 包装器版本: 表示在使用联合服务器注册包装器期间正在执行的包装器代码的版本的指定包装器版本号。可以将此值与当前正在执行的代码的版本作比较以确保兼容性。
- 包装器模块库的顶层文件名。
- 包含与此包装器有关的所有信息的包装器信息对象，该对象作为执行 CREATE WRAPPER 或 ALTER WRAPPER DDL 语句的结果存储在联合服务器系统目录中。

仅成员函数才应检查或改变此信息。

所有包装器必需的定制

不受防护类属包装器子类必须实现:

- 构造函数，它将调用相应的不受防护类属包装器基本类构造函数。

  C++: Unfenced_Generic_Wrapper

  Java:
  UnfencedGenericWrapper

- create server: 用来创建不受防护类属服务器子类的实例的方法。

  C++: Unfenced_Generic_Server

  Java:
  UnfencedGenericServer

附加定制

- 如果需要将附加信息存储在不受防护类属包装器子类的实例中，则覆盖 initialize_my_wrapper() 函数的缺省实现以从作为参数提供的包装器信息对象抽取此信息。在 C++ 中，不能保留指向此包装器信息对象的指针。如果选择以此格式引用该信息，则使用包含包装器信息对象副本的数据成员。在 Java 中，包装器可保留对 WrapperInfo 对象的引用，但没有任何复制功能。
- 如果定义包装器选项，则覆盖 verify_my_register_wrapper_info 函数和 verify_my_alter_wrapper_info 函数的缺省实现以验证在 DDL 上提供的选项和值是否有效。如果希望改变在 DDL 上提供的选项值或提供附加选项，则实现应该通过 "备份" 包装器信息对象提供覆盖 / 附加信息。在分配新的 "备份" 对象之前，检查该对象是否已经存在。如果 "备份" 对象已存在，则不要再分配一个。在 Java 中，"备

份”包装器信息对象实际上是 verifyMyRegisterWrapperInfo 和 verifyMyAlterWrapperInfo 方法的返回对象。此对象必须由包装器创建。

- 如果不受防护类属包装器子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 16. 虚函数

C++ 虚函数	Java 虚函数	缺省行为
initialize_my_wrapper	initializeMyWrapper	没有选项
create_server	createServer	错误
verify_my_register_wrapper_info	verifyMyRegisterWrapperInfo	验证是否未指定任何非 DB2 Information Integrator 选项
verify_my_alter_wrapper_info	verifyMyAlterWrapperInfo	调用 verify_my_register_wrapper_info 对象

表 17. 公用／受保护成员函数

C++ 公用／受保护成员函数	Java 公用／受保护成员函数	行为
is_reserved_wrapper_option	isReserved	如果选项由 DB2 UDB 处理，则返回 true

Fenced_Generic_Wrapper 类

C++: 在包含包装器代码的共享库中调用函数 FencedWrapper_Hook 会创建受防护包装器子类的实例。当联合服务器第一次装入包装器的受防护共享库时，该联合服务器会调用此函数。如果 DDL 操作更改与包装器有关的信息，则在其下次使用之前联合服务器会破坏并重新创建包装器对象。

Java: 当联合服务器装入受防护包装器类时，将自动创建该类的实例。特定于包装器的受防护包装器类的名称在 CREATE WRAPPER 语句中被指定为 FENCED_WRAPPER_CLASS 选项的值。

受防护类属包装器基本类实现保留以下信息:

- 包装器名称。
- 包装器类型: “N”表示非关系。
- 包装器版本: 表示在使用联合服务器注册包装器期间正在执行的包装器代码的版本的指定包装器版本号。此版本号是整数而不是浮点类型。可以将此值与当前正在执行的代码的版本作比较以确保兼容性。
- 包装器模块库的文件名。
- 包含与此包装器有关的信息的包装器信息对象，该对象作为执行 CREATE WRAPPER 或 ALTER WRAPPER DDL 语句的结果存储在数据库管理器目录中。

仅成员函数才应检查或改变此信息。

所有包装器必需的定制

受防护类属包装器子类必须实现:

- 构造函数，它将调用相应的受防护类属包装器基本类构造函数。

  C++: Fenced_Generic_Wrapper

Java:

FencedGenericWrapper

- create server: 用来创建受防护类属服务器子类的实例的方法。

  C++: Fenced_Generic_Server

  Java:

  FencedGenericServer

附加定制

- 如果需要将附加信息存储在受防护类属包装器子类的实例中，则覆盖 initialize_my_wrapper 函数的缺省实现以从作为参数提供的包装器信息对象抽取此信息。在 C++ 中，不能保留指向此包装器信息对象的指针。如果选择以此格式引用该信息，则使用包含包装器信息对象副本的数据成员。在 Java 中，包装器可保留对 WrapperInfo 对象的引用，但没有任何复制功能。
- 如果您的 Unfenced_Generic_Wrapper 子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 18. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
initialize_my_wrapper	initializeMyWrapper	是	没有选项
create_server	createServer	是	错误

相关任务:

- 第 66 页的『服务器类』
- 第 70 页的『昵称类』
- 第 72 页的『用户类』
- 『改变包装器』（*《联合系统指南》*）

相关参考:

- 『ALTER WRAPPER statement』（*SQL Reference, Volume 2*）
- 『Wrapper classes for the Java API』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Wrapper class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『C++ API 的包装器类』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）

服务器类

以下各节描述 Unfenced_Generic_Server 和 Fenced_Generic_Server 类。

Unfenced_Generic_Server 类

对不受防护包装器子类的实例调用 create server 方法将创建不受防护服务器子类的实例。联合服务器会在应用程序第一次使用相关服务器之前调用此方法。如果 DDL 操作更改与服务器有关的信息，则在其下次使用之前联合服务器会破坏并重新创建服务器对象。

除非定制，否则 Unfenced_Generic_Server 基本类实现保留以下信息:

- 服务器名称。
- 指向相应的包装器对象的指针。
- 包含与服务器有关的信息的服务器信息对象，作为执行 DDL 语句的结果存储在联合服务器系统目录中。

可以直接作为数据成员存取此信息中的一些信息。仅数据成员函数可以检查或改变此信息的剩余部分。

所有包装器必需的定制

Unfenced_Generic_Server 子类必须实现:

- 子类的构造函数，用来调用相应的 Unfenced_Generic_Server 构造函数。
- create nickname: 用来创建不受防护昵称子类的实例的方法。
- plan request: 用来分析包含在“请求”对象中的查询段并返回一个或多个“应答”对象（每个对象标识可由数据源执行的查询段及其相应的执行成本）的方法。

附加定制

- 如果需要将附加信息存储在 Unfenced_Generic_Server 子类的实例中，则覆盖 initialize_my_server 成员函数的缺省实现以从作为参数提供的服务器信息对象抽取此信息。在 C++ 中，不能保留指向此服务器信息对象的指针。如果选择以此格式存储该信息，则使用包含此信息对象副本的数据成员。在 Java 中，服务器可保留对 ServerInfo 对象的引用，但没有任何复制功能。
- 如果除了已经对 Unfenced_Generic_Server 类定义的选项之外还要定义服务器选项、服务器类型和服务器版本，则覆盖 verify_my_register_server_info 对象和 verify_my_alter_server_info 对象的缺省实现以验证在 DDL 上提供的选项和值的有效性。不需要验证标准 Unfenced_Generic_Server 选项; Unfenced_Generic_Server 实现将执行此操作。如果希望改变在 DDL 上提供的选项值或提供附加选项，则实现应该通过“备份”服务器信息对象提供覆盖／附加信息。在 C++ 中，在分配新的“备份”对象之前，检查该对象是否已经存在。如果该对象存在，则不要再分配一个。在 Java 中，“备份”包装器信息对象实际上是 verifyMyRegisterWrapperInfo 和 verifyMyAlterWrapperInfo 方法的返回对象。此对象必须由包装器创建。
- 如果缺省 Unfenced_Generic_User 类对包装器来说不够，则覆盖 create remote user 方法的实现以创建 Unfenced_Generic_User 子类的实例。
- 如果缺省成本模型对基数或执行时间生成不准确的估计值，则覆盖 create reply 方法的实现以创建为支持定制成本模型而定制的“应答”子类的实例。
- 如果缺省选择率求值程序对涉及数据源中的数据的谓词生成不准确的值，则使用产生更精确的结果的方法来覆盖 get selectivity 方法的缺省实现。
- 如果您的 Unfenced_Generic_Server 子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 19. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
verify_my_register_server_info	verifyMyRegisterServerInfo	是	针对预定义列表验证各个选项（根据 C++ is_reserved_server_option() 函数和 Java CatalogOption 类中的 isReserved() 函数）。任何其它选项都产生错误。
verify_my_alter_server_info	verifyMyAlterServerInfo	是	针对预定义列表验证各个选项（根据 C++ is_reserved_server_option() 函数和 Java CatalogOption 类中的 isReserved() 函数）。任何其它选项都产生错误。
initialize_my_server	initializeMyServer	是	没有选项
create_nickname	createNickname	是	错误
create_remote_user	createRemoteUser	是	创建不受防护类属用户
create_reply	createReply	是	创建支持缺省成本模型的“应答”
get_selectivity	getSelectivity	是	使用缺省模型计算选择率
get_type	getType	是	从信息中获取类型
get_version	getVersion	是	从信息中获取版本
plan_request	planRequest	否	返回相应的应答对象

Fenced_Generic_Server 类

对受防护包装器子类的实例调用 create server 方法将创建受防护服务器子类的实例。联合服务器会在应用程序第一次使用相关服务器之前调用此方法。如果 DDL 操作更改与服务器有关的信息，则在其下次使用之前联合服务器会破坏并重新创建服务器对象。

除非定制，否则受防护类属服务器基本类实现保留下列信息:

- 服务器名称。
- 指向相应的包装器对象的指针。
- 包含与此服务器有关的信息的服务器信息对象，作为执行 DDL 语句的结果存储在联合服务器系统目录中。
- 此服务器的实例化远程连接对象的表（当前实现将远程连接的数目限制为一）。
- 如果存在与数据源的连接，则为指向表示已连接用户的用户映射的受防护类属用户对象的指针。

可以直接作为数据成员存取此信息中的一些信息。仅数据成员函数可以检查或改变此信息的剩余部分。

所有包装器必需的定制

受防护类属服务器子类必须实现:

- 子类的构造函数，用来调用相应的受防护类属服务器构造函数。

- create nickname: 用来创建受防护昵称子类的实例的方法。

  C++: Fenced_Nickname

  Java:
  FencedNickname
- create remote connection: 用来创建远程连接子类的实例的方法。

  C++: Remote_Connection

  Java:
  RemoteConnection

附加定制

- 如果需要将附加信息存储在受防护类属服务器子类的实例中，则覆盖 initialize_my_server 函数的缺省实现以从作为参数提供的服务器信息对象抽取此信息。在 C++ 中，如果选择以此格式存储该信息，则不能保留指向此服务器信息对象的指针，而是使用包含此对象副本的数据成员。在 Java 中，服务器可保留对 ServerInfo 对象的引用，但没有任何复制功能。
- 如果缺省受防护类属用户类对包装器来说不够，则覆盖 create remote user 方法的实现以创建受防护类属用户子类的实例。

  C++: Fenced_Generic_User

  Java:
  FencedGenericRemoteUser
- 如果受防护类属服务器子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 20. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
initialize_my_server	initializeMyServer	是	没有选项
create_remote_user	createRemoteUser	是	创建不受防护类属用户类
create_remote_connection	createRemoteConnection	是	错误
create_nickname	createNickname	是	错误
get_type	getType	是	从信息中获取类型
get_version	getVersion	是	从信息中获取版本

表 21. 公用 / 受保护成员函数

C++ 公用 / 受保护成员函数	Java 公用 / 受保护成员函数	行为
find_current_remote_user	findRemoteUser	查找当前授权标识的远程用户对象
find_connection	findConnection	获取单个已连接远程用户的连接对象
find_nickname	findNickname	获取给定本地模式名的不受防护昵称对象

相关任务:

- 第 63 页的『包装器类』
- 第 70 页的『昵称类』

- 第 72 页的『用户类』

相关参考:

- 『Server classes for the Java API』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『C++ API 的服务器类』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

昵称类

下列各节描述不受防护类属昵称和受防护类属昵称类。

Unfenced_Generic_Nickname 类

对受防护服务器子类的实例调用 create nickname 方法将创建不受防护昵称子类的实例。通常会因为调用不受防护类属服务器方法 find_nickname（在 Java 中为 findNickname）而调用此方法。如果 DDL 操作更改与昵称有关的信息，则在其下次使用之前联合服务器会破坏并重新创建昵称对象。

除非定制，否则不受防护类属昵称基本类实现保留下列信息:

- 正对其定义昵称的远程数据集的本地名称。
- 正对其定义昵称的远程数据集的本地模式。
- 指向相应的服务器对象的指针。
- 缺省成本模型使用的四条统计信息的估计值:
  1. 昵称的基数。它被定义为包含在昵称中的行数。联合服务器将个别昵称的基数存储在系统表 SYSCAT.TABLES 或 SYSSTAT.TABLES（任一表中的 “CARD” 列）中。如果昵称没有基数，则成本模型使用缺省值 1000 行。
  2. 昵称的设置成本。设置成本表示包装器准备好查询段以提交至远程源时通常所花的时间（以毫秒计）。当包装器接收它在查询规划期间生成的包装器 “执行描述符” 时，设置开始，并且在包装器准备将相应操作提交给远程源时，设置结束。设置成本应仅包括当要求包装器再次执行同一查询段（可能带有不同的参数值）时它不需要重复的工作。例如，如果包装器将查询段以 URL 的形式提交给远程源，则设置成本包括从包装器存储在 “执行描述符” 中的信息生成该 URL 所需的时间。联合服务器将此统计信息存储在 SETUP_COST 昵称选项中。如果昵称没有该选项，则成本模型使用值 2000 毫秒。
  3. 昵称的提交成本。提交成本表示包装器将查询段提交至远程源通常所需的时间（以毫秒计）。提交在设置结束时（如上面定义的那样）开始，并且在包装器准备从源请求第一行或第一块结果数据时结束。提交成本应仅包括每次提交给定查询段时包装器必须重复的工作。例如，如果每次与远程源进行交互都需要新的 HTTP 连接，则提交成本应该包括创建此连接所需的时间。联合服务器将此统计信息存储在 SUBMISSION_COST 昵称选项中。如果昵称没有该选项，则成本模型使用值 25 毫秒。
  4. 昵称的高级成本。这是访存昵称的一行通常所需的时间（以毫秒计）。它是启动查询需要的任何时间之外的时间。联合服务器将此统计信息存储在 ADVANCE_COST 昵称选项中。如果昵称没有该选项，则成本模型使用的值是 50 毫秒。如果数据源以块而不是以行为单位返回数据，则通过将访存一块的典型成本除以每块中的典型行数来计算高级成本。

所有包装器必需的定制

Unfenced_Generic_Nickname 子类必须实现:

- 子类的构造函数，它仅应调用相应的 Unfenced_Generic_Nickname 构造函数。

附加定制

- 如果需要将附加信息存储在 Unfenced_Generic_Nickname 子类的实例中，则覆盖 initialize_my_nickname 函数的缺省实现以从作为参数提供的昵称信息对象抽取此信息。在 C++ 中，不能保留指向此昵称信息对象的指针。在 Java 中，可保留指向昵称信息对象的引用。
- 如果定义昵称选项或列选项，则覆盖 verify_my_register_nickname_info 和 verify_my_alter_nickname_info（在 Java 中为 verifyMyRegisterNicknameInfo 和 verifyMyAlterNicknameInfo）的缺省实现以验证在 DDL 上提供的选项和值的有效性。如果希望改变在 DDL 上提供的选项值或提供附加选项，则实现应该通过“备份”昵称信息对象提供覆盖／附加信息。在分配新的“备份”对象之前，检查该对象是否已经存在。如果是的话，则使用它而不再分配另一个。在 Java 中，“备份”对象实际上是返回对象，必须由包装器创建。
- 如果已分配 Unfenced_Generic_Nickname 子类指向的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 22. 虚函数

C++ 虚函数	Java 虚函数	缺省行为
initialize_my_nickname	initializeMyNickname	没有选项
verify_my_register_nickname_info	verifyMyRegisterNicknameInfo	针对预定义的列表验证各个选项（根据 C++ is_reserved_nickname_option () 函数和 Java CatalogOption.isReserved() 方法）。任何其它选项都产生错误。
verify_my_alter_nickname_info	verifyMyAlterNicknameInfo	针对预定义的列表验证各个选项（根据 C++ is_reserved_nickname_option () 函数和 Java CatalogOption.isReserved() 方法）。任何其它选项都产生错误。

Fenced_Generic_Nickname 类

对受防护服务器子类的实例调用 create nickname 方法将创建受防护昵称子类的实例。通常会因为执行 CREATE NICKNAME 语句而调用此方法。如果 DDL 操作更改与昵称有关的信息，则在其下次使用之前联合服务器会破坏并重新创建昵称对象。

除非定制，否则 Fenced_Generic_Nickname 基本类实现保留以下信息:

- 正对其定义昵称的远程数据集的本地名称。
- 正对其定义昵称的远程数据集的本地模式。
- 指向相应的服务器对象的指针。

所有包装器必需的定制

Fenced_Generic_Nickname 子类必须实现:

- 子类的构造函数，用来调用相应的 Fenced_Generic_Nickname 构造函数。

附加定制

- 如果需要将附加信息存储在 Fenced_Generic_Nickname 子类的实例中，则覆盖 initialize_my_nickname 函数的缺省实现以从作为参数提供的昵称信息对象抽取此信息。在 C++ 中，不能保留指向此昵称信息对象的指针。在 Java 中，可保留指向昵称信息对象的引用。
- 如果定义昵称选项，则覆盖 verify_my_register_nickname_info 函数的缺省实现以验证在 DDL 上提供的选项和值的有效性。如果希望改变在 DDL 上提供的选项值、提供附加选项或提供从数据源而不是从 DDL 获取的信息和选项，实现应使用“备份”昵称信息对象提供覆盖或附加信息。在分配新的“备份”对象之前，检查该对象是否已经存在。如果是的话，则使用它而不再分配另一个。在 Java 中，“备份”对象实际上是返回对象，必须由包装器创建。
- 如果已分配 Unfenced_Generic_Nickname 子类指向的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 23. 虚函数

C++ 虚函数	Java 虚函数	缺省行为
initialize_my_nickname	initializeMyNickname	没有选项
verify_my_register_nickname_info	verifyMyRegisterNicknameInfo	针对预定义的列表验证各个选项（根据 C++ is_reserved_nickname_option () 和 Java CatalogOption.isReserved() 方法）。任何其它选项都产生错误。

表 24. 公用／受保护成员函数

C++ 公用／受保护成员函数	Java 公用／受保护成员函数	行为
get_card	getCard	存取方法
get_setup_cost	不适用	存取方法
get_advance_cost	不适用	存取方法
get_submission_cost	不适用	存取方法

相关任务:

- 第 63 页的『包装器类』
- 第 66 页的『服务器类』
- 第 72 页的『用户类』

相关参考:

- 『Nickname classes for the Java API』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Nickname class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『C++ API 的昵称类』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

用户类

下列各节描述 Unfenced_Generic_User 和 Fenced_Generic_User 类。

Unfenced_Generic_User 类

对不受防护服务器子类的实例调用 create remote user 方法将创建不受防护用户子类的实例。联合服务器在处理 CREATE USER MAPPING 或 ALTER USER MAPPING 语句时调用此方法。如果 DDL 操作更改与远程用户有关的信息，则在其下次使用之前联合服务器会破坏并重新创建该远程用户对象。

除非定制，否则 Unfenced_Generic_User 基本类实现保留以下信息:

- 用户的本地授权标识。
- 包含与此（服务器和用户）对有关的信息的用户信息对象，该对象作为执行 DDL 语句的结果存储在联合服务器系统目录中。
- 指向相应的服务器对象的指针。

所有包装器必需的定制

如果数据源不需要认证信息，则不受防护类属用户基本类的定制不是必需的。

如果创建 Unfenced_Generic_User 子类，则它必须实现:

- 子类的构造函数，它仅应调用相应的 Unfenced_Generic_User 构造函数。

附加定制

- 如果需要将附加信息存储在 Unfenced_Generic_User 子类的实例中，则覆盖 initialize_my_user 函数的缺省实现以从作为参数提供的用户信息对象抽取此信息。在 C++ 中，如果选择以此格式存储该信息，则不能保留指向此用户信息对象的指针，而是使用包含此对象副本的数据成员。在 Java 中，可保留对 UserInfo 对象的引用，但没有任何复制功能。
- 如果定义用户选项，则覆盖 verify_my_register_user_info 和 verify_my_alter_user_info 函数的缺省实现以验证在 DDL 上提供的选项和值的有效性。如果希望改变在 DDL 上提供的选项值或提供附加选项，则实现应该通过“备份”用户信息对象提供覆盖／附加信息。在分配新的“备份”对象之前，检查该对象是否已经存在。如果是的话，则使用它而不再分配另一个。在 Java 中，“备份”对象实际上是验证方法返回的对象，需要由包装器创建。
- 如果您的 Unfenced_Generic_User 子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 25. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
verify_my_register_user_info	verifyMyRegisterUserInfo	是	验证是否仅指定了 DB2 UDB 选项。
initialize_my_user	initializeMyUser	是	没有选项
verify_my_alter_user_info	verifyMyAlterUserInfo	是	验证是否仅指定了 DB2 UDB 选项。

表 26. 公用／受保护成员函数

C++ 公用／受保护成员函数	Java 公用／受保护成员函数	行为
get_local_name	getLocalName	存取方法

Fenced_Generic_User 类

对受防护服务器子类的实例调用 create remote user 方法将创建受防护用户子类的实例。在应用程序使用远程连接对象第一次请求连接至服务器之前，联合服务器会调用此方法。如果 DDL 操作更改与远程用户有关的信息，则在其下次使用之前联合服务器会破坏并重新创建该远程用户对象。

除非定制，否则受防护类属用户基本类实现保留下列信息:

- 用户的本地授权标识。
- 包含与此服务器 / 用户对有关的信息的用户信息对象，作为执行 DDL 语句的结果存储在联合服务器系统目录中。
- 指向相应的服务器对象的指针。

所有包装器必需的定制

如果数据源不需要认证信息，则 Fenced_Generic_User 基本类的定制不是必需的。

如果创建 Fenced_Generic_User 子类，则它必须实现:

- 子类的构造函数，它仅应调用相应的 Fenced_Generic_User 构造函数。

附加定制

- 如果需要将附加信息存储在受防护类属用户子类的实例中，则覆盖 initialize_my_user 函数的缺省实现以从作为参数提供的用户信息对象抽取此信息。在 C++ 中，如果选择以此格式存储该信息，则不能保留指向此用户信息对象的指针，而是使用包含此对象副本的数据成员。在 Java 中，可保留对 UserInfo 对象的引用，但没有任何 “复制” 功能。
- 如果您的 Fenced_Generic_User 子类指向已分配的任何外联存储器，则必须对释放此存储器的子类实现析构函数。

表 27. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
initialize_my_user	initializeMyUser	是	没有选项

表 28. 公用 / 受保护成员函数

C++ 公用 / 受保护成员函数	Java 公用 / 受保护成员函数	行为
get_local_name	getLocalName	存取方法

相关任务:

- 第 63 页的『包装器类』
- 第 66 页的『服务器类』
- 第 70 页的『昵称类』

相关参考:

- 『User classes for the Java API』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『C++ API 的用户类』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

请求类

此类是在查询规划期间作为“请求应答补偿”协议的部分使用的。联合服务器优化器生成此类的实例以描述可能要求数据源对其求值的每个查询段。

方法

表 29. 方法

C++ 名称	Java 名称	描述
get_number_of_quantifiers	getNumberOfQuantifiers	检索 FROM 子句中的元素数目。
get_number_of_predicates	getNumberOfPredicates	检索 WHERE 子句中的元素数目。
get_number_of_head_exp	getNumberOfHeadExp	检索 SELECT 子句中的元素数目。
get_quantifier_handle	getQuantifier.getHandle	检索 FROM 子句中指定位置处的元素的句柄。
get_predicate_handle	getPredicate.getHandle	检索 WHERE 子句中指定位置处的元素的句柄。
get_head_exp_handle	getHeadExp.getHandle	检索 SELECT 子句中指定位置处的元素的句柄。
get_nickname	getNickname	检索 FROM 子句中句柄指定的量词的“昵称”对象。
get_head_exp	getHeadExp	检索 SELECT 子句中句柄指定的头表达式的 Request Exp 对象。
get_predicate	getPredicate	检索 WHERE 子句中句柄指定的谓词的 Request Exp 对象。
get_distinct	getDistinct	测试 DISTINCT 指示符。

相关参考:

- 『Request class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『C++ API 的请求类』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

应答类

包装器生成此类的实例以表示请求中数据源可以运行的查询段的一部分。包装器可以对单个“请求”生成几个应答。

“应答”类将方法和数据成员添加至:

- 通过向类添加条目以填充应答: add_to_CCCC
- 存储执行描述符指针和大小
- 当包装器对同一请求返回多个方案时链接应答
- 存储顺序信息: 优化器可能用来构造更优方案的已返回数据的感兴趣的排序

高级定制

可以使用更精确描述该源的执行模型的成本方法来覆盖成本方法的缺省实现。覆盖缺省成本取决于查询段中谓词的缺省选择率估计。调用

Unfenced_Generic_Server::get_selectivity 方法（在 Java 中为 UnfencedGenericServer.getSelectivity 方法）将获取选择率估计，您也可以覆盖它。

方法

表 30. 方法

C++ 名称	Java 名称	是否提供了缺省值?	描述
add_head_exp	addHeadExp	是	在下一个空闲位置添加头表达式。
add_predicate	addPredicate	是	在下一个空闲位置添加谓词句柄。
add_quantifier	addQuantifier	是	在下一个空闲位置添加量词句柄。
add_order_entry	addOrderEntry	是	添加排序条目。排序条目由方向（升序或降序）和排序过的头表达式的整数索引组成。
get_parameter_order	不适用	是	返回表示需要参数的查询段的应答的参数顺序。它执行 WHERE 和 SELECT 子句中的所有表达式的预排序遍历并产生一个顺序。返回参数句柄列表。
cardinality	cardinality	是	返回通过执行由应答表示的查询段返回的结果的基数。cardinality() 的缺省版本返回谓词的选择率乘以所有昵称的基数的乘积。
total_cost	totalCost	是	执行由应答表示的查询段所需的总执行成本（以毫秒计）。缺省版本调用 all_costs 并返回总成本值。
re_exec_cost	reExecCost	是	重新执行由应答表示的查询段所需的时间。缺省情况下调用 all_costs() 并返回重新执行成本值。 **注**: 开发者覆盖 all_costs() 比覆盖此例程要好。
first_tuple_cost	firstTupleCost	是	获取由应答表示的查询段的第一结果元组所需的时间。缺省情况下调用 all_costs() 并返回第一元组成本值。 **注**: 开发者覆盖 all_costs() 比覆盖此例程要好。
set_next_reply	setNextReply	是	当包装器对一个请求返回多个应答时链接应答。

表 30. 方法 (续)

C++ 名称	Java 名称	是否提供了缺省值?	描述
all_costs	不适用	否; 必须实现	计算三个成本值，总成本、第一元组成本和重新执行成本。 **第一元组成本** = average-setup-cost + average-submission-cost + average-advance-cost。 **重新执行成本** = average-submission-cost + (average-advance-cost * estimated-cardinality) **总成本** = average-setup-cost + average-submission-cost + (average-advance-cost * estimated-cardinality) 其中: **average-setup-cost** 查询段中所有昵称的设置成本的平均值。 **average-submission-cost** 查询段中所有昵称的提交成本的平均值。 **average-advance-cost** 查询段中所有昵称的高级成本的平均值。 **estimated-cardinality** 查询段的估计基数（由 cardinality() 方法返回）。
get_number_of_quantifiers	getNumberOfQuantifiers	是	检索 FROM 子句中的元素数目。
get_number_of_predicates	getNumberOfPredicates	否; 必须实现	检索 WHERE 子句中的元素数目。
get_number_of_head_exp	getNumberOfHeadExp	否; 必须实现	检索 SELECT 子句中的元素数目。
get_quantifier_handle	getQuantifier.getHandle	否; 必须实现	检索 FROM 子句中某个位置处的句柄。
get_predicate_handle	getPredicate.getHandle	否; 必须实现	检索 WHERE 子句中某个位置处的句柄。
get_head_exp_handle	getHeadExp.getHandle	否; 必须实现	检索 SELECT 中某个位置处的句柄。
get_nickname	getNickname	否; 必须实现	检索 FROM 子句中给定句柄处的量词的类“昵称”对象。
get_head_exp	getHeadExp	否; 必须实现	检索 SELECT 子句中给定句柄处的头表达式的类 Request_Exp 对象。
get_predicate	getPredicate	否	检索 WHERE 子句中给定句柄处的谓词的类 Request_Exp 对象。
set_distinct	setDistinct	否; 必须实现	检索 DISTINCT 指示符（SELECT DISTINCT 子句）。
get_distinct	getDistinct	否; 必须实现	测试 DISTINCT 指示符。
get_exec_desc	getExecDistinct	否; 必须实现	检索此查询的执行描述符的指针和长度。
set_exec_desc	setExecDesc	否; 必须实现	将执行描述符存储在“应答”中。

相关参考:

- 『Reply class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Reply 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

谓词列表类

谓词列表类的实例是选择率估计方法 Unfenced_Generic_Server::getSelectivity()（在 Java 中为 UnfencedGenericServer.getSelectivity）的输入。这两个谓词列表包括:

- 一组对其请求选择率的谓词 P
- 一组已应用的谓词 AP

结果选择率是在给定 AP 情况下的 P 的*条件*选择率，即，在已应用集合 AP 中的谓词的情况下集合 P 中的谓词的选择率。

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

如果需要无条件选择率，则 AP 可能为 NULL。该谓词列表与“请求”中的谓词列表相似，是使用类似方法处理的。

方法

表 31. 方法

C++ 名称	Java 名称	描述
create_empty_predicate_list	不适用	创建一个空的谓词列表，它的大小与请求中的谓词列表大小相同。生成的列表是空的。
create_and_copy_predicate_list	不适用	创建一个空的谓词列表并从应答中复制该谓词列表。
operator new	不适用	在 QG 堆上分配谓词列表。
destructor	不适用	由该包装器创建 Predicate_List 对象且必须由该包装器破坏它。
get_number_of_predicates	getNumberOfPredicates	检索谓词列表的长度。
get_predicate	getPredicate	返回在句柄处描述谓词的 Request_Exp 对象。
get_predicate_handle	getPredicateHandle	返回某个位置的谓词的句柄。
get_number_of_applied_predicates	getNumberOfAppliedPredicates	返回已应用谓词列表的长度。
get_applied_predicate	getAppliedPredicate	返回在句柄处描述已应用谓词的 Request_Exp 对象。
get_applied_predicate_handle	getAppliedPredicateHandle	返回某个位置的已应用谓词的句柄。
add_predicate	addPredicate	将谓词句柄添加至谓词列表中的下一个空闲位置。
add_applied_predicate	addAppliedPredicate	将已应用谓词句柄添加至已应用谓词列表中的下一个空闲位置。

相关参考:

- 『PredicateList class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Predicate_List 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

请求表达式类

此类定义用于研究请求表达式的接口。表达式可以是头表达式或谓词。所有这些表达式都表示为已解析运算符树。由于表达式是递归定义的，所以每个树节点本身就是一个表达式。树的内部节点（子表达式）表示运算符，而叶表示常量、列或参数。根据表达式的类型的不同，包装器编写者可使用的信息也不同。对于每种表达式，包装器可使用的信息也不同。除了返回种类的方法外，所有表达式都具有返回表达式的数据类型、其父表达式（对于根表达式为 NULL）及其兄弟表达式（如果有的话，则为该节点右边的相邻节点，否则为 NULL）的方法。表达式的个别种类具有返回附加信息的方法，如下表所述:

表 32. 有关每种类型的表达式的附加信息

表达式的类型	信息
运算符	• 子节点的数目（操作数） • 第一个操作数 • 标记（如通过查询解析） • 特征符 - 包括操作数数据类型的已解析函数 / 运算符名称
列	• 列名 • 列所属的量词
常量	值缓冲区、长度和类型
参数	用于标识参数数组中参数位置的参数句柄，该数组用于将参数从引擎传递至包装器。

方法

表 33. C++ 请求表达式类方法

C++ 名称	描述
get_kind	返回表达式的种类（Request_Exp::oper、Request_Exp::column、Request_Exp::unbound 和 Request_Exp::constant）。仅当从优化器调用函数时，才需要函数的最后一个自变量。
get_data_type	返回描述表达式的结果数据类型的 Request_Exp_Type 实例。
get_parent	返回表达式的父节点。
get_next_child	返回兄弟节点。
get_column_name	返回带有列名的字符串。
get_column_quantifier_handle	返回此列所属的量词。
get_value	返回表示表达式中值和常量类型的 Request_Constant 实例。

表 33. C++ 请求表达式类方法 (续)

C++ 名称	描述
get_parameter_handle	返回唯一标识参数的整数。
get_number_of_children	返回运算符表达式的操作数的数目。
get_first_child	返回第一个操作数。
get_token	将标记值作为字符串返回。
get_signature	返回运算符的特征符。

表 34. Java 请求表达式类方法

Java 名称	描述
getKind	返回表达式的种类（RequestExp.OPERATOR、RequestExp.CONSTANT、RequestExp.UNBOUND 和 RequestExp.COLUMN）。
getDataType	返回描述表达式的结果数据类型的 RequestExpType 实例。
getParent	返回表达式的父节点。
getNextChild	返回兄弟节点。
getColumnName	返回带有列名的字符串。
getQuantifier	返回此列所属的量词。
getValue	返回表示表达式中的常量的值的 RequestConstant 实例。
getNumberOfChildren	返回运算符表达式的操作数的数目。
getFirstChild	返回第一个操作数。
getToken	将标记值作为字符串返回。
getSignature	返回运算符的特征符。
getHandle	返回表达式的句柄。

相关参考:

- 『RequestExp class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Request_Exp 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

请求常量类

使用此类描述查询表达式中的常量。Request_Exp::get_value()（在 Java 中为 RequestExp getValue）函数在应用于 Request_Exp::constant 种类（在 Java 中为 RequestExp.CONSTANT）的节点时返回请求常量类的实例。

方法

表 35. 方法

C++ 名称	Java 名称	描述
get_actual_length	getActualLength	获取缓冲区中的数据长度。
get_data	getData	获取指向数据缓冲区的指针。

表 35. 方法 (续)

C++ 名称	Java 名称	描述
get_data_type	getDataType	返回该列的 DB2 UDB 类型标识
is_data_null	isDataNull	语义或友好算术是否为空？
get_precision	getPrecision	返回数字常数值的精度。
get_scale	getScale	返回数字常数值的小数位。
get_codepage	getCodepage	返回字符值的代码页。

相关参考:

- 『RequestConstant class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Request_Constant 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）

请求表达式类型类

此类的实例描述查询表达式的数据类型。输入每个查询表达式（包括子表达式）。函数 Request_Exp::get_data_type()（在 Java 中为 RequestExpType getDataType）返回请求表达式类型实例。

方法

表 36.

C++ 名称	Java 名称	描述
get_for_bit_data	getForBitData	列是否包含二进制数据？有效值为 Y 和 N，其中 Y 表示“是”而 N 表示“否”。Java 方法 getForBitData 返回布尔值（不是 Y 和 N）。
get_null_indicator	getNullIndicator	指定哪一列是空指示符
get_data_type	getDataType	在 C++ 中，列（在 sql.h 中）的 SQL 类型。在 Java 中，类型标识常量是在数据类中定义的。
get_maximum_length	getMaximumLength	列（SQL_TYP_DECIMAL 除外）的最大长度
get_precision	getPrecision	精度（仅适用于 SQL_TYP_DECIMAL）
get_scale	getScale	小数位（仅适用于 SQL_TYPE_DECIMAL）
get_codepage	getCodepage	列的代码页
get_name	getName	列名（如果有的话）
get_name_length	getNameLength	列名的长度

相关参考:

- 『RequestExpType class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）

- 『Request_Exp_Type 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）

远程连接类

要创建远程连接实例，对相应受防护服务器子类的实例调用方法 create_remote_connection（在 Java 中为 createRemoteConnection）。联合服务器在相关服务器上处理第一次远程操作之前调用此方法。在应用程序在没有使用服务器的情况下执行了规定数目的事务之后，联合服务器破坏远程连接实例。

除非定制，否则远程连接基本类实现保留下列信息:

- 与数据源的连接的状态（已连接或已断开连接）。
- 指向相应的服务器对象的指针。
- 指向相应的用户对象的指针。
- 对此连接实例化的远程操作对象的列表。
- 要用于此连接的代码页。这是已连接至联合服务器的客户机应用程序的代码页。
- 连接种类: 无阶段、一阶段和两阶段（版本 8.2 不支持两阶段事务）。

所有包装器必需的定制

远程连接子类必须实现该子类的构造函数，以用来调用相应的远程连接构造函数。

远程连接类还需要实现其它方法，这要视包装器的需要而定。以下列表描述构造函数及其使用:

- connect: 建立与数据源的连接的方法。即使数据源不需要连接，也必须实现此方法。
- disconnect: 关闭与数据源已建立的连接的方法。即使数据源不需要连接，也必须实现此方法。

  在建立或关闭连接后，就不必调用 mark_disconnected 方法（在 Java 中为 markDisconnected 方法）。
- create remote query 方法: 创建远程查询子类的实例。

  C++: create_remote_query

  Java:

  createRemoteQuery
- commit: 用来在数据源处落实当前事务的方法。如果正将连接用于 passthru 会话并且数据源支持事务，则实现应在数据源处落实当前事务。所有非 passthru 事务在版本 8.1 中是只读的，但数据源仍需要事务结束通知以释放资源和删除读锁定等等。即使数据源不支持事务，也必须实现此方法。
- rollback: 用来在数据源处回滚当前事务的方法。如果正将连接用于 passthru 会话且数据源支持事务，则实现应在数据源处回滚当前事务。所有非 passthru 事务在版本 8.1 中是只读的，但数据源仍需要事务结束通知以释放资源和删除读锁定等等。即使数据源不支持事务，也必须实现此方法。

此外，如果实现的任何部分检测到与数据源的连接已丢失，则会调用 mark disconnected 方法并返回错误。mark disconnection 方法将通知数据库管理器连接已丢失并启动相应的清除。

附加定制

- 如果需要存储有关与数据源的连接的信息（例如，连接句柄），则将适当的数据成员添加至子类定义并在构造函数中初始化它们。
- 如果分配了远程连接子类指向的任何外联存储器，则必须对释放此存储器的子类实现析构函数。析构函数不需要关闭与数据源的连接；如果存在与数据源的连接，联合服务器将总是在删除远程连接对象之前调用断开连接方法。
- 如果包装器支持 passthru，则必须覆盖 create remote passthru 方法的缺省实现。实现应创建远程 passthru 子类的实例。

表 37. 虚函数

C++ 虚函数	Java 虚函数	是否提供了缺省值？	缺省行为
连接	连接	否；必须实现	
断开连接	断开连接	否；必须实现	
create_remote_query	createRemoteQuery	是	错误
create_remote_passthru	createRemotePassthru	是	错误
落实	落实	否；必须实现	
回滚	回滚	否；必须实现	

表 38. 公用／受保护成员函数

C++ 公用／受保护成员函数	Java 公用／受保护成员函数	行为
is_connected	isConnected	是否已连接至数据源？
mark_connected	markConnected	已连接至数据源
mark_disconnected	markDisconnected	未连接至数据源
get_kind	getKind	存取方法
get_codepage	getCodepage	存取方法

相关参考:

- 『Remote_Connection 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）
- 『RemoteConnection class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）

远程查询类

对适当的远程连接子类的实例调用方法 create remote query class 将创建远程查询子类的实例。在提交远程查询的应用程序对查询的结果集关闭其游标时，联合服务器将破坏该远程查询对象。

对于某些数据源，包装器将不能控制数据源返回数据值的顺序。包装器可能获取后跟 LOB 数据的非 LOB 数据，该 LOB 数据后跟有更多非 LOB 数据等等。该包装器只能按从数据源接收数据的顺序来处理数据。

必须实现该子类的构造函数，以用来调用相应的远程查询构造函数。输出运行时数据列表是由远程操作基本类构造函数创建的。远程查询子类必须实现下列方法。下列解释使用 C++ 名称；如果要开发 Java 包装器，则替代为相应的 Java 方法名。

fetch()

fetch() 方法负责将非 LOB 数据检索到由运行时数据列表标识的联合服务器缓冲区中。

fetch_lob()

fetch_lob() 方法负责将 LOB 数据检索到联合服务器提供的特殊 LOB 缓冲区中（一次一块）。fetch_lob() 方法可能需要若干调用来处理整个 LOB，原因是缓冲区的大小限制了每次调用传送的量。

fetch() 和 fetch_lob()

fetch() 和 fetch_lob() 方法以协同例程的方式运行以按 LOB 和非 LOB 列从远程源到达的顺序来处理它们。远程查询对象提供共享状态，因此包装器可记录它在数据流中的位置。在从 fetch() 或 fetch_lob() 返回之前，包装器指示网关从联合服务器的下一次调用应该是 fetch() 还是 fetch_lob()。

open_input_lob()

open_input_lob() 方法允许包装器初始化包含输入 LOB 参数的远程查询。如果包装器支持输入 LOB 参数，则必须在特定于包装器的远程查询类中实现此方法。如果 DB2 Information Integrator 发现输入 LOB 主变量，它将调用此方法。包装器必须返回接收下一个 LOB 段的主变量的索引并调用 set_row_status() 以指示存在输入 LOB 参数。

当包装器指示还有更多 LOB 输入参数要处理时，DB2 Information Integrator 将使用另一 LOB 段调用此方法。当前 LOB 段的大小（以字节计）将传递至包装器，而包装器必须使用该信息进至下一个输入变量或者指示已读取完整输入值。

reopen_input_lob()

reopen_input_lob() 方法复位先前打开的结果流并准备好数据源以返回更多结果集（可能根据带有不同输入 LOB 参数的查询的不同参数绑定）。如果包装器支持输入 LOB 参数，则必须在特定于包装器的远程查询类中实现此方法。除非该查询先前是以结束查询状态（close 方法）关闭的，否则将不调用此方法。如果发现输入 LOB 主变量，DB2 Information Integrator 将调用 reopen_input_lob() 方法。包装器必须返回接收下一个 LOB 段的主变量的索引并调用 set_row_status() 以指示存在输入 LOB 参数。

当包装器指示还有更多 LOB 输入参数要处理时，DB2 Information Integrator 将使用另一 LOB 段调用此方法。LOB 输入值和当前 LOB 段的大小（以字节计）将传递至包装器，而包装器必须使用这些值进至下一个输入变量或者指示已读取完整输入值。

set_row_status()

get_row_status() 方法设置当前行状态。此方法已经实现，包装器不能重载。

get_row_status()

set_row_status() 方法检索当前行状态。此方法已经实现，包装器不能重载。

您可以决定想要如何在此处的方法间分发函数。

表 39. 函数

C++ 虚函数	Java 虚函数	是否提供了缺省值?	缺省行为
fetch	fetch	是	错误
fetch_lob	fetchLob	是	错误
open	open	是	错误
open_input_lob	openInputLob	是	错误
reopen	reopen	是	错误
reopen_input_lob	reopenInputLob	是	错误
close	close	是	错误
set_row_status	setRowStatus	是	错误
get_row_status	getRowStatus	是	错误

相关参考:

- 『RemoteQuery class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Remote_Query 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

运行时数据类

下列各节描述运行时数据和运行时数据列表类。

表 40. C++ 和 Java 运行时数据列表类

C++	Java
Runtime_Data	RuntimeData
Runtime_Data_List	RuntimeDataList

运行时数据类

联合服务器创建运行时数据类的实例以表示系统用来在服务器与包装器之间传送列值的每个缓冲区。联合服务器为在执行期间将值传送至包装器的每个查询参数创建缓冲区。联合服务器还会为包装器返回的结果行的每一列（例如，为查询段中的每个头表达式）创建缓冲区。

下列状态将会保留:

- 列号
- 指向描述该值的 Runtime_Data_Desc 对象（在 Java 中为 RuntimeDataDesc）的指针。
- 指向数据缓冲区的指针
- 缓冲区中数据的长度
- 对于表示参数值的列，指示参数值是否为不变量的标志。除非通过“reopen”方法的“action”参数通知包装器，否则不变量参数的值将不会更改。

联合服务器提供列号、数据描述和数据缓冲区。如果要使用运行时数据对象将参数从联合服务器传递至包装器，则将由联合服务器提供数据长度和缓冲区内容。缓冲区中的值将具有由附带描述指定的 SQL 类型。包装器必须将值转换为数据源期望的任何类型。如果要使用运行时数据对象从包装器将结果返回至联合服务器，则该包装器将提供数据长度和缓冲区内容。缓冲区的最大（已分配）长度和期望的数据类型可从附带描述中获取。包装器必须将从数据源获取的值转换为由描述指定的 SQL 类型。

表 41. 公用成员函数

C++ 公用成员函数	Java 公用成员函数	行为
check_friendly_div_by_0	checkFriendlyDivBy0	是否除以零？
check_friendly_exception	checkFriendlyException	算术是否异常？
get_actual_length	getActualLength	获取缓冲区中的数据长度
get_data	getData	获取指向数据缓冲区的指针
get_data_index	getDataIndex	获取列号
get_invariant	getInvariant	输入值是否为不变量？
is_data_null	isDataNull	语义或“友好算术”是否为空？
is_semantic_null	isSemanticNull	语义是否为空？
set_actual_length	setActualLength	设置缓冲区中的数据长度
set_data	setXX	将数据复制至缓冲区
set_data_null	setDataNull	设置空指示符
set_friendly_div_by_0	setFriendlyDivBy0	指示零除异常，先设置为空！
set_friendly_exception	setFriendlyException	指示算术异常，先设置为空！

运行时数据列表类

基本类构造函数为远程查询创建此类的实例。为输出行和输入参数创建独立的列表。

表 42. 公用成员函数

C++ 公用成员函数	Java 公用成员函数	行为
get_number_of_values	getNumberOfValues	获取列的数目
get_ith_value	getValue	获取指向 i-th 列的 Runtime_Data 的指针
operator[]	不适用	获取指向 i-th 列的 Runtime_Data 的指针

相关参考:

- 『Runtime_Data 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）
- 『Runtime_Data_List 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）
- 『RuntimeDataList class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『RuntimeData class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）

运行时数据描述类

下列各节描述运行时数据描述和运行时描述列表类。

表 43. C++ 和 Java 运行时数据描述列表类

C++	Java
Runtime_Data_Desc	RuntimeDataDesc
Runtime_Data_Desc_List	RuntimeDataDescList

运行时数据描述类

联合服务器创建此类的实例以表示系统用来将列值传送至包装器的每个缓冲区。联合服务器为在执行期间将值传送至包装器的每个查询参数创建缓冲区。联合服务器还会为包装器返回的结果行的每一列（例如，为查询段中的每个头表达式）创建缓冲区。对于远程查询和远程 passthru 对象，联合服务器将为输入数据列表中表示参数值的每个运行时数据对象创建运行时数据描述。联合服务器提供列描述。对于远程查询对象，联合服务器还将为输出数据列表中表示结果行中的值的每个运行时数据对象创建运行时数据描述。联合服务器还会提供这些列描述。

表 44. 公用成员函数

C++ 公用成员函数	Java 公用成员函数	行为
get_for_bit_data	getForBitData	列是否包含二进制数据？
get_null_indicator	getNullIndicator	列能否为空？
get_data_type	getDataType	列（在 sql.h 中）的 SQL 类型
get_maximum_length	getMaximumLength	列（SQL_TYP_DECIMAL 除外）的最大长度
get_precision	getPrecision	精度（仅适用于 SQL_TYP_DECIMAL）
get_scale	getScale	小数位（仅适用于 SQL_TYPE_DECIMAL）
get_codepage	getCodepage	列的代码页
get_name	getName	列名（如果有的话）
get_name_length	getNameLength	列名的长度

运行时数据描述列表类

表 45. 公用成员函数

C++ 公用成员函数	Java 公用成员函数	行为
get_number_of_values	getNumberOfValues	获取列的数目
get_ith_value	getValue	获取指向 i-th 列的 Runtime_Data_Desc 的指针
set_ith_value	setValue	获取指向 i-th 列的 Runtime_Data_Desc 的指针
operator[]	不适用	获取指向 i-th 列的 Runtime_Data_Desc 的指针

相关参考:

- 『Runtime_Data_Desc 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）
- 『Runtime_Data_Desc_List 类（C++）』（*《IBM DB2 Information Integrator 开发包装器的 C++ API 参考》*）
- 『RuntimeDataDesc class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『RuntimeDataDescList class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）

远程 passthru 类

对适当的远程连接子类的实例调用方法 create remote passthru 将创建远程 passthru 子类的实例。如果数据源不支持 passthru，将不实现远程 passthru 子类且不覆盖此方法的缺省实现。

所有包装器必需的定制

远程 passthru 子类必须实现:

- 子类的构造函数，它应调用相应的远程 passthru 构造函数。

  C++: Remote_Passthru

  Java:
  RemotePassthru
- prepare: 允许将 passthru 字符串提交至数据源以确定将组成结果的每一行的列的数目和类型的方法。
- describe: 填充描述构成结果的每一行的列的数目和类型的远程数据描述列表的方法。
- open: 允许数据源准备返回查询的第一个结果行的方法。
- fetch: 将单个结果行复制到输出运行时数据列表中的方法。
- close: 允许在执行查询之后清除数据源的方法。

就象远程查询一样，包装器编写者对于上面列示的方法之间的函数分布有相当大的决定权。例如，运行时数据描述列表的填充可能发生在 prepare 或 describe 方法中，将 passthru 字符串提交至数据源可能发生在 open 或 fetch 方法中。

附加定制

如果数据源支持返回结果代码但不返回任何行的 passthru 字符串，则覆盖执行方法的缺省实现。实现应将该字符串提交至数据源并返回结果代码。如果客户机应用程序通过 EXECUTE 或 EXECUTE IMMEDIATE 语言构造提交 passthru 字符串，则将调用执行方法，并在 passthru 字符串不代表适当操作（即它代表返回行的操作）的情况下时报告错误。

表 46. 虚函数

C++ 和 Java 虚函数	是否为缺省值?	缺省行为
prepare	是	错误

表 46. 虚函数 (续)

C++ 和 Java 虚函数	是否为缺省值?	缺省行为
describe	是	错误
execute	是	错误
open	是	错误
close	是	错误

相关参考:

- 『RemotePassthru class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Remote_Passthru 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

包装器实用程序类

不应实例化此类。此类的存在只是为了将静态方法的集合组合到一起，这些静态方法允许包装器存取由联合服务器提供的各种服务。

包装器实用程序类的静态方法提供下列服务:

- 内存分配和释放。除非在使用“新建”实例化或使用“删除”破坏从 Sqlqg_Base_Class 派生的 C++ 对象，否则实现必须使用这些方法以分配和释放内存。如果类不是 Sqlqg_Base_Class 的后代，或者您希望覆盖“新建”或“删除”的 Sqlqg_Base_Class 实现，则必须提供使用这些方法分配和释放内存的这些运算符的实现。
- 错误报告。包装器会将数据源返回的错误映射至最适合的 DB2 UDB 错误或 SQL 代码。每个 SQL 代码都存在错误消息字符串，该字符串通常由特定于错误发生环境的标记参数化。例如，用于报告缺少选项的错误消息字符串由包含该类选项（包装器和服务器等等）、实体名（包装器名和服务器名等等）和所缺选项的名称的标记参数化。包装器应在报告错误时提供这些标记的值。要查找最适当的 SQL 代码和关联标记的内容，请参阅《*消息参考*》。

  在标识最适当的 SQL 代码后，查找相应的符号定义。表 47 显示每个平台的该目录。

表 47. 符号定义的目录（按平台）

平台	包装器安装目录
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

  缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

  对于不容易映射至任何其它 SQL 代码的错误，使用 SQL_RC_E1822（SQL 代码 = -1822）。
- 适当的代码页大小写转换。
- 存取当前 DB2 UDB 数据库的单字节和双字节代码页。
- 包装器跟踪设施中的包装器控制流信息。

表 48. 包装器实用程序类的公用静态成员函数行为

C++ 公用静态成员函数	Java 公用静态成员函数	行为
allocate	不适用	分配内存块。
deallocate	不适用	释放内存块。
report_error	WrapperException 类（Java 使用异常）	使用 SQL 代码、消息和标记报告错误。
report_warning	reportWarning	报告警告。
convert_to_upper	不适用	适当的代码页大小写转换。
convert_to_lower	不适用	适当的代码页大小写转换。
get_sb_DB_codepage	getSingleByteDBCodepage	获取单字节数据库代码页。
get_db_DB_codepage	getDoubleByteDBCodepage	获取双字节数据库代码页。
string_to_tokens	不适用	标记化字符串。
trace_data	不适用	将跟踪信息写至跟踪设施。
get_db2_install_path	getDB2InstallPath	检索指向安装的路径。
get_db2_instance_path	getDB2InstancePath	检索指向联合服务器的路径。
get_db2_release	getDB2Release	检索包装器正在运行的 DB2 UDB（包括修订包）的版本。
fnc_entry	traceFunctionEntry	记录包装器跟踪设施的进入函数操作。
fnc_exit	traceFunctionExit	记录包装器跟踪设施的退出函数操作。
fnc_data	traceFunctionData	通过使用包装器跟踪设施记录跟踪数据。
fnc_data2	traceFunctionData	使用包装器跟踪设施记录跟踪数据，包括探测点和两个数据元素。
fnc_data3	traceFunctionData	使用包装器跟踪设施记录跟踪数据，包括探测点和三个数据元素。
trace_error	traceException 或 traceError	使用包装器跟踪设施记录错误跟踪数据，包括错误代码和探测点。
convert_codepage	不适用	将输入数据从源代码页转换为目标代码页。
get_expected_conv_len	不适用	返回从原始代码页转换为新代码页的字符串的字节数。
get_env_lang	不适用	返回操作系统的语言设置。
change_endian2	不适用	更改双字节字符串的字节存储顺序。

相关参考:

- 『WrapperUtilities class (Java)』（*IBM DB2 Information Integrator Java API Reference for Developing Wrappers*）
- 『Wrapper_Utilities 类（C++）』（《*IBM DB2 Information Integrator 开发包装器的 C++ API 参考*》）

第 10 章 确保包装器与环境共存

本节描述确保包装器与环境共存需要考虑的内容。它包括下列主题:

- 使用系统服务时需要知道哪些内容
- 如何使环境变量可供包装器访问
- 包装器可移植性

将系统服务与包装器配合使用

因为包装器必须与其它联合服务器进程共存，所以在使用操作系统服务时必须小心。

联合服务器使用所有的服务是不可能的；因此最好假定联合服务器使用任意系统服务和方案。这一需求也适用于包装器可能包括的所有第三方软件。如果不能保证正常运行，则必须将某些功能隔离在它自己的进程空间中。

第一步是尽可能少地使用操作系统服务。第二步是使用通过包装器实用程序提供的那些系统服务例程；使用这些实用程序来完成内存管理尤其重要。

最后，包装器必须确保使用正确更改系统状态的服务。例如，如果包装器使用信号处理程序，则它必须在每次调用包装器例程时安装和除去它们；当该包装器将控制权返回给联合服务器时，其信号处理程序不能再保持活动状态。另一个示例是警报的使用。无论何时返回至联合服务器，包装器都必须复原警报状态。

系统服务的已知限制:

- 必须通过提供的实用程序函数和类完成内存管理。
- stdout、stderr、stdin、cin、cout 和 cerr 的输入／输出（I/O）不起作用。
- Windows: 包装器必须使用 Win32 例程 GetEnvironmentVariable() 而不是 getenv()。
- 包装器一定不能将 Unix 和 Windows 系统服务 `strtok` 和 `strtok_r` 用于标记化。包装器接口为标记化提供替代系统服务。

内存管理（仅适用于 C++）

所有内存管理必须通过包装器实用程序分配和释放方法来完成。已经提供了一个基本类 Sqlqg_Base_Class，它具有使用这些方法的新建和删除操作程序，所以自 Sqlqg_Base_Class 派生的类可以正常工作。

标记化服务（仅适用于 C++）

Unix 和 Windows 服务 `strtok` 和 `strtok_r` 的用途是将字符串分为基于分隔符的几个部分。要想使用比这些服务更好的办法，包装器编写者必须使用包装器实用程序类的 `string_to_token()` 方法。此方法的工作方式与 `strtok_r` 完全一样，并且是线程安全的。

以下代码段演示 `Wrapper_Utilities::string_to_token` 方法的使用。

```
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);
}
```

相关概念:

- 第 92 页的『使环境变量对包装器可用』

相关任务:

- 第 93 页的『包装器可移植性』

使环境变量对包装器可用

联合服务器控制包装器可使用哪些环境变量。为了使包装器可存取某个环境变量，必须在 db2dj.ini 配置文件中指定它的值。表 49 按平台显示配置文件驻留在联合服务器上的目录。

表 49. 联合服务器上的配置文件目录

平台	包装器安装目录
AIX®	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows®	%DB2PATH%/cfg

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2® Information Integrator 在 Windows 上的安装目录路径的环境变量。

联合服务器在启动时会读取该环境变量。要更改环境变量的值，必须停止联合服务器并重新启动它。

db2dj.ini 中的条目具有下列格式:

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

注意，[white space] 是可选的。变量名和变量值都不能包含空格或行结束符。配置文件中的每一行都需要行结束符，即使是该文件的最后一行也是如此。

包装器可使用 getenv()（在 Windows 平台上，使用 GetEnvironmentVariable()）访问环境变量。

C++ 编码注意事项

包装器使用某些 C++ 工具（例如，异常和“运行时类型标识”（RTTI））的方式有一些限制。

包装器可以使用 C++ 异常，但是该包装器必须确保异常将不会传播至联合服务器。未能捕获所有异常并在包装器接口处将其转换为返回码将导致联合服务器代理进程（或者可能是整个联合服务器实例）异常终止。

包装器接口不支持“运行时类型标识”（RTTI）。这意味着一些其它 C++ 功能部件（例如，动态强制类型转型）不可用，因为许多编译器需要 RTTI 才能实现这些功能部件。

相关任务:

- 第 91 页的『将系统服务与包装器配合使用』
- 第 93 页的『包装器可移植性』

包装器可移植性

对于 Windows，包装器必须使用 Win32 例程 GetEnvironmentVariable() 或 GetEnvironmentStrings() 例程而不是使用 Unix getenv() 例程访问环境变量。

相关概念:

- 第 92 页的『使环境变量对包装器可用』

相关任务:

- 第 91 页的『将系统服务与包装器配合使用』

第 11 章 说明包装器

因为在很大程度上是由包装器开发者确定要对注册而提交的 SQL 语句指定的内容，所以建议开发者让用户知道如何编码这些语句。因为开发者赋予包装器的能力在很大程度上确定用户可期望从数据源得到的内容，建议开发者让用户知道是如何使用数据源的。为用户提供信息的一种方法是在手册或联机文件中编译它。本主题建议在这种编译中将包括哪些项并提供有关如何处理这些项的提示。

在说明包装器时，考虑讨论下列项:

数据源处提供的信息

这样用户可以知道在他们的查询中可请求哪类信息，您可以描述:

- 用户可从数据源处的数据集合（例如，表、数据集或电子表格）中检索的信息的性质
- 数据源处提供的数据集合的类型
- 数据源处可从已存储的信息派生的函数的可检索信息的性质

注册 在论述注册时，考虑提供下列信息:

- 需要注册哪些构造
- 将使用其启动注册的 SQL 语句的语法
- 在这些语句中，可以对哪些选项进行编码
- 选项的允许值有哪些
- 语句的参数的含义
- 参数的允许值有哪些
- 数据源处的数据集合是如何映射至 CREATE NICKNAME 语句的参数的
- 函数模板表示数据源的哪些能力

报告 可以包括报告的小段有用信息。有几个种类: 需求、限制、建议、提示、技巧和提醒。以下列表中的示例来自 IBM® 关于表结构文件服务器的包装器的文档。

需求的示例

- “列定界符在整个文件中必须一致。”
- “必须通过更新 SYSSTAT 视图手工更新表结构文件的昵称的统计信息。”

限制的示例

- “Passthru 会话不允许使用包装器。”
- “将文件限制为每行一个记录。”

提示的示例

- “同未排序文件相比，系统可以更有效地搜索已排序数据文件。”
- “对于已排序文件，可以通过指定键列的值或范围来改进性能。”

错误和警告

还应考虑说明包装器通过其报告可能问题的错误或警告消息。这样用户就可查看这些消息的解释，包括关联 SQLCODE、返回码（如果有的话）和 SQLSTATE 值（如果有的话）。

用户还会发现示例和逐步指示信息很有帮助。

有关包装器文档的示例，请参阅《*DB2® Information Integrator 数据源配置指南*》。

相关任务:

- 第 99 页的『编译包装器（C++）』
- 第 103 页的第 13 章，『链接包装器（仅适用于 C++）』
- 第 105 页的第 14 章，『安装包装器』
- 第 113 页的『测试包装器的有效选项和无效选项』
- 第 100 页的『编译包装器（Java）』

第 4 部分 构建、测试和跟踪包装器

本书的此部分指导您完成构建、测试和跟踪包装器所需的下列任务:

- 构建和打包包装器以进行部署。
- 测试包装器以确保它按设计的方式工作。

第 12 章 编译包装器

编译包装器（C++）

当编译包装器代码时，编译器必须能够访问随包装器开发工具箱安装的包装器接口头文件。这些文件与其它必需的头文件一起驻留在联合服务器的 include 子目录中。

在 AIX 上编译

使用 xlC_r7 程序以在 AIX 上编译包装器代码。以下示例显示用来从样本包装器编译一个文件的命令:

```
/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
```

如果正在使用 VisualAge 6.0（或更新版本）编译器，则使用 `-qnamemangling=v5` 和 `-qlonglong` 选项。

有关更多详细信息，请参阅包装器开发工具箱中提供的 makefile。表 50 按平台显示存储包装器开发工具箱的子目录。

表 50. 包装器开发工具箱的目录（按平台）

平台	包装器安装目录
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

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

在 Windows 上编译

使用 CL 程序在 Windows 上编译包装器代码。以下示例显示用来从样本包装器编译一个文件的命令:

```
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
```

相关任务:

- 第 105 页的第 14 章, 『安装包装器』
- 第 113 页的『测试包装器的有效选项和无效选项』
- 第 100 页的『编译包装器（Java）』

编译包装器（Java）

为编译包装器执行的步骤取决于开发该包装器所用的语言。本主题说明如何编译用 Java 开发的包装器。

先决条件:

必须具有 Java Development Kit（JDK）V1.3 或更新版本。特别是，必须具有 Java 编译器，它包括在 JDK 中。在 Windows 上，Java 编译器为 javac.exe，在 UNIX 上，Java 编译器为 javac。

类路径中必须有 db2qgjava.jar。db2qgjava.jar 是包含包装器 API 类的 JAR。可将它添加至系统 CLASSPATH 或在编译期间使用 `-classpath` 选项。表 51 按平台显示 db2qgjava.jar 文件所在的目录。

表 51. Java 配置文件的目录（按平台）

平台	包装器安装目录
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

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

过程:

要编译 Java 包装器:

1. 确保 db2qgjava.jar 在类路径中。
2. 像编译任何其它 Java 类一样编译定制包装器类。

   ```
   javac @wrapperfiles
   ```

   javac 编译器允许您指定各种选项，包括:

   - 要编译的 Java 源文件。因为 Java 源文件具有它们所包含的类定义的名称，所以包装器的源文件通常包括诸如 UnfencedXXWrapper.java、UnfencedXXServer.java、FencedXXWrapper.java、FencedXXServer.java 和 FencedXXUser.java（其中 XX 是包装器的名称）之类的文件。
   - 要编译的 Java 源文件的位置。如果 Java 源文件不在工作目录中也不在类路径中，可在 javac 命令中使用 -sourcepath 选项来指定它们的位置。
   - 对 CLASSPATH 的补充。如果未将 db2qgjava.jar 包括在 CLASSPATH 环境变量中，则使用 -classpath 选项将它包括在 javac 命令中。
   - 已编译 .class 文件的目标目录。

   例如，可以从 DB2 CLP 窗口在工作目录中执行以下命令来将 db2qgjava.jar 文件添加至类路径、编译样本 Java 包装器并安装 %DB2PATH%\function 中的已编译类:

   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
```

相关任务:

- 第 99 页的『编译包装器（C++）』
- 第 105 页的第 14 章, 『安装包装器』
- 第 113 页的『测试包装器的有效选项和无效选项』

第 13 章 链接包装器（仅适用于 C++）

包装器由三个独立的代码库组成；它们分别是顶层库、受防护库和不受防护库。顶层库的共享库是作为包装器开发工具箱的一部分提供的。余下的两个库必须通过包装器代码构建。

包装器的三个库必须遵守可执行代码的共享库的平台约定，并且它们还必须遵守与这三个库相关联的命名约定。下列这些示例说明了这一点。

在 AIX 上，样本包装器顶层库命名为 “libsample.a”。 “lib” 前缀和 “.a” 后缀是共享库的平台约定。不受防护包装器库命名为 “libsampleU.a”。注意，这与顶层库相同，但不受防护包装器库只是在后缀之前多了大写字母 “U”。受防护包装器库命名为 “libsampleF.a”。

在 Windows 上，样本包装器库为 “db2sample.dll” 、 “db2sampleU.dll” 和 “db2sampleF.dll”。

下表列示每个平台的共享库后缀:

表 52. 平台和受支持的共享库后缀

平台	库后缀
AIX	.a
HP-UX	.sl
Linux	.so
Sun	.so
Windows	.dll

不受防护包装器库必须包含不受防护包装器挂机例程、UnfencedWrapper_Hook() 和下列各项的子类的所有代码:

- Unfenced_Generic_Wrapper
- Unfenced_Generic_Server
- Unfenced_Generic_User
- Unfenced_Generic_Nickname

受防护包装器库必须包含受防护包装器挂机例程、FencedWrapper_Hook() 和下列各项的子类的所有代码:

- Fenced_Generic_Wrapper
- Fenced_Generic_Server
- Fenced_Generic_User
- Fenced_Generic_Nickname
- Remote_Connection
- Remote_Query
- Remote_Passthru

如果不使用某些类（例如，如果包装器使用 Fenced_Generic_User 类的缺省实现），则这些类没有要链接到库中的代码。

在 AIX 上链接

在 DB2 Information Integrator 安装的 /usr/opt/db2_08_01/lib 目录中提供了顶级库并将其命名为 libdb2sqqgtop.a。已链接此库，只需对包装器复制此库并对其重命名。

受防护库和不受防护库必须与已导出符号的文件链接在一起；当在执行时将该库装入到联合服务器环境中时，此文件允许链接程序解析可用的符号。已导出符号文件在联合服务器的 /usr/opt/db2_08_01/lib 目录中，且被称为 udbwrapper.exp。

要在 AIX 上链接受防护库和不受防护库，使用 makeC++SharedLib_r 工具。下面是使用此工具链接不受防护包装器库的一个示例:

```
/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
```

包装器开发工具箱提供了具有详细信息的 makefile。makefile 位于 /usr/opt/db2_08_01/samples/wrapper_sdk 子目录中。

在 Windows 上链接

包装器开发工具箱的 %DB2PATH%\bin 目录中提供了顶级库并将其命名为 db2sqqgtop.dll。已链接此库，只需对包装器复制此库并对其重命名。

受防护库和不受防护库必须与已导出符号的文件链接在一起；当在执行时将该库装入到联合服务器环境中时，此文件允许链接程序解析可用的符号。已导出符号文件可在联合服务器的 %DB2PATH%\bin 目录中找到。有两个文件 db2qgstp.lib 和 db2qg.lib。可信端库应链接至 db2qg.lib，而受防护端应链接至 db2qgstp.lib。

要在 Windows 上链接受防护库和不受防护库，使用 “link” 工具。下面是使用此工具链接不受防护包装器库的一个示例:

```
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
```

包装器开发工具箱提供了具有详细信息的 makefile。makefile 位于 %DB2PATH%\samples\wrapper_sdk 目录中。

相关任务:

- 第 99 页的『编译包装器（C++）』
- 第 105 页的第 14 章，『安装包装器』
- 第 100 页的『编译包装器（Java）』

第 14 章 安装包装器

必须先在联合服务器上安装定制包装器才能使用它。安装包装器的详细信息取决于开发该包装器所用的语言。

安装 C++ 包装器

要安装 C++ 包装器:

1. 停止联合服务器。在安装、替换或删除 C++ 包装器库之前，必须停止 DB2 通用数据库联合服务器。在 DB2 UDB 运行期间管理这些库可能会导致 DB2 UDB 崩溃。
2. 通过链接包装器来查找正在创建的三个 C++ 包装器库: 共享顶级库（它包括在包装器工具箱中）、受防护库和不受防护库。
3. 将这些库安装到 DB2 UDB 安装的相应目录中。表 53 显示对每个平台应用的目录。

表 53. C++ 包装器安装的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/lib
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib
Windows	%DB2PATH%\bin

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

安装 Java 包装器

可以两种格式中的任一种传递 Java 包装器:

- 一组 .class 文件，通过编译 Java 包装器创建它们。通常，开发者只在开发和测试期间才使用 .class 文件安装包装器。
- 单个 .jar 文件，通过使用 JAR 工具（包括在 Java Development Kit 中）从 .class 文件创建它们。通常，开发者使用 .jar 文件传递 Java 应用程序，这是因为 .jar 文件更容易安装。安装单个 .jar 文件还会减少丢失文件的机会。

安装 .class 文件

要将包装器作为一组 .class 文件安装:

1. 将开发系统中的所有包装器的 .class 文件复制至联合服务器。在安装过程中，如果更新了 CLASSPATH 或者如果替换了现有 .jar 和 .class 文件，则在使用新安装的包装器文件之前可能需要重新启动 DB2 UDB。为便于安装，根据平台将文件安装到以下目录中。表 54 显示对每个平台应用的目录。

表 54. Java 包装器安装的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/lib
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib
Windows	%DB2PATH%\bin

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

2. 将包含包装器的 .class 文件的目录包括在系统 CLASSPATH 环境变量中。CLASSPATH 环境变量通常已经包含 SQLLIB\function 目录。
3. 将包装器使用的所有 Java 类包括在系统 CLASSPATH 环境变量中。

安装 .jar 文件

要将包装器作为 .jar 文件安装:

1. 将开发系统中的包装器 .jar 文件复制至联合服务器。
2. 配置系统以便联合服务器可以找到包装器 .jar 文件。有两种方法可用来完成此任务:
   - 将包装器 .jar 文件的路径和名称包括在系统 CLASSPATH 环境变量中，或者
   - 使用 DB2 通用数据库存储过程 SQLJ.install_jar 向联合服务器注册包装器 .jar 文件。当使用 SQLJ.install_jar 时，DB2 通用数据库制作 .jar 文件的副本。如果更新 .jar 文件，则必须指示 DB2 通用数据库替换它的 .jar 文件副本。有关其它信息，请参阅 DB2 通用数据库文档。
3. 将包装器使用的所有 Java 类包括在系统 CLASSPATH 环境变量中。

确保 Java 内存大小设置足够

通过修改 DB2 通用数据库的 JAVA_HEAP_SZ DBM 环境变量来确保 Java 内存大小设置足够。对于简单包装器（例如，包装器工具箱中提供的样本包装器），建议的最小大小是 1024。JAVA_HEAP_SZ 变量的量度单位是 4 KB。最佳值取决于包装器: 装入了多少个类、创建了多少个对象以及使用 Java 包装器的并发连接数。装入 500 到 1000 个类的包装器可能需要高达 2048 的设置。

有关设置 JAVA_HEAP_SZ 变量的其它信息，请参阅 DB2 通用数据库文档。

相关任务:

- 第 108 页的『安装开发 XML 配置文件向导』
- 第 110 页的『安装 XML 配置文件』
- 第 107 页的『将数据源添加至 DB2 控制中心』

第 15 章 将数据源添加至控制中心

将数据源添加至 DB2 控制中心

可以通过在 DB2 控制中心中将定制包装器作为数据源添加来使包装器可供用户使用。然后，用户可以选择数据源（包装器）并指定选项。

XML 配置文件用来对 DB2 控制中心描述数据源及其选项。当启动 DB2 控制中心时，它将装入存在 XML 配置文件的所有数据源。当用户选择数据源时，控制中心将显示在 XML 配置文件中为该数据源定义的选项。

包装器开发工具箱包括用来帮助您创建 XML 配置文件的工具。

先决条件:

必须具有包装器开发工具箱。

过程:

要将定制数据源添加至 DB2 控制中心:

1. 使用“开发 XML 配置文件”向导来为包装器创建 XML 配置文件和相关联的属性文件。
2. 安装 XML 配置文件。表 55 显示在联合服务器上安装该文件的目录（按平台）。

表 55. 安装 XML 配置文件的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows	%DB2PATH%/cfg

表 56 显示相关联的属性文件所在的位置。

表 56. 安装 XML 配置文件的目录（按平台）

平台	包装器安装目录
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

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

3. 如果想要支持 DB2 控制中心的发现功能部件以便用户可以发现有关定制数据源的昵称、视图和服务器发现信息，则在联合服务器上安装必需的发现支持。如果选择使用内置发现工具，则只需要安装定制发现存储过程。（已随 DB2 控制中心安装了图形发现工具。）
4. 重新启动 DB2 控制中心。

相关概念:

- 第 6 页的『用户如何将数据源添加至联合系统』

相关任务:

- 第 108 页的『安装开发 XML 配置文件向导』
- 第 108 页的『创建 XML 配置文件』
- 第 110 页的『安装 XML 配置文件』

安装开发 XML 配置文件向导

“开发 XML 配置文件”向导用来创建将包装器（数据源）添加至 DB2 控制中心所需的配置文件。

该向导是与 DB2 Information Integrator 包装器开发工具箱一起安装的。表 57 显示确定系统上是否安装了该向导的位置。向导的安装文件是 db2qgjava.jar。

表 57. XML 配置文件向导的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%\bin\db2wrapperconfig.bat

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

相关任务:

- 第 107 页的『将数据源添加至 DB2 控制中心』
- 『安装包装器开发工具箱』（*IBM DB2 Information Integrator 安装指南 Linux、UNIX 和 Windows 版*）

创建 XML 配置文件

控制中心读取 XML 配置文件以确定要提供给用户的数据源（包装器）和选项。要将定制包装器添加至 DB2 控制中心，必须创建用于该包装器的 XML 配置文件。要创建 XML 配置文件，使用“开发 XML 配置文件”向导。

先决条件:

必须已安装包装器开发工具箱。

必须已安装“Java 运行时环境”（JRE）。（该向导是 Java 应用程序。）

过程:

要创建 XML 配置文件:

1. 启动“开发 XML 配置文件”向导。表 58 显示从中启动向导的目录。

表 58. 从中启动向导的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/lib/db2wrapperconfig

表 58. 从中启动向导的目录（按平台） (续)

平台	包装器安装目录
HP/Sun/Linux	/opt/IBM/db2/V8.1/lib/db2wrapperconfig
Windows	%DB2PATH%\bin\db2wrapperconfig.bat

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

2. 选择是想要创建新配置文件还是修改现有配置文件。
3. 遵循向导所提供的步骤。向导将提示您输入:
   - 文件信息，包括在 DB2 控制中心中用来标识包装器的名称、DTD 路径和 XML 文件路径的名称。
   - 包装器信息，包括在 DB2 控制中心中用来标识包装器的数据源的名称和描述。包装器信息还包括受支持操作系统和用于每个操作系统的包装器库或类名。
   - 用户必须提供给包装器的包装器选项。对于每个选项，都可以指定允许的值、缺省值和描述。还可以指定该选项是否是必需的、在创建包装器期间用户是否可以编辑它以及在创建包装器之后用户是否可以改变它。还可以指定 SQL 需求，例如，CREATE SERVER 语句是否需要用户标识。
   - 用户必须提供给包装器的服务器选项。
   - 用户必须提供给包装器用户映射选项。
   - 用户必须提供给包装器的昵称选项。还可以指定用户是否可以在 CREATE NICKNAME 语句中指定列定义。
   - 用户必须提供给包装器的列选项。
   - 包装器支持的 DB2 数据类型。可以对每个服务器指定不同的数据类型。
   - DB2 控制中心是否支持对此包装器使用发现功能。如果选择支持发现功能，则还必须执行其它步骤。
   - 用户必须提供的环境变量并设置它们的位置。
   - 此包装器必需的函数模板、用户定义的函数和函数映射。

   所有页面上都提供了帮助和弹出信息。

结果:

该向导创建两个文件:

- XML 配置文件，包含在该向导中指定的信息。XML 配置文件位于在该向导的第二页上指定的目录中。文件名是基于包装器名称的。例如，如果包装器名称为 GeoDataSource，则 XML 文件名为 GeoDataSource.xml。
- 属性文件，包含将在 DB2 控制中心中显示的对应包装器及其选项的文字文本字符串。将文本字符串外部化为属性文件使得更容易支持多种语言。可以将一个配置文件与多个属性文件配合使用。XML 配置文件包含替代文本字符串的符号名称；控制中心在与包装器名称和用户的语言相对应的属性文件中查找这些符号名称。属性文件是在创建 XML 配置文件的同一目录中创建的并且具有基于包装器名称的名称。例如，如果包装器名称为 GeoDataSource，则属性文件名为 GeoDataSource.properties。

表 59 显示 cc_plugin 目录中提供这两个文件的样本的位置。

表 59. 样本的目录（按平台）

平台	包装器安装目录
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

相关概念:

- 第 31 页的『决定包装器选项』
- 第 32 页的『决定服务器选项』
- 第 27 页的『决定昵称和列选项』
- 第 33 页的『决定用户映射选项』
- 第 6 页的『用户如何将数据源添加至联合系统』

相关任务:

- 第 35 页的『确定数据源可以接受的头表达式』
- 第 35 页的『确定数据源可以接受的谓词』
- 第 36 页的『确定数据源可以接受的连接』
- 第 36 页的『确定数据源可以接受的函数』
- 第 107 页的『将数据源添加至 DB2 控制中心』

安装 XML 配置文件

DB2 控制中心通过读取 XML 配置文件来确定将哪些包装器（数据源）和选项呈现给用户。在创建用于包装器的 XML 配置文件之后，必须在联合服务器上安装那些可从 DB2 控制中心装入的文件。要将数据源添加至 DB2 控制中心，必须在联合服务器上安装该数据源的 XML 配置文件。

先决条件:

必须具有对包装器的 XML 配置和属性文件集的访问权。

必须具有对联合服务器的访问权。

过程:

要安装用于包装器的 XML 配置文件:

1. 将包装器 XML 配置文件和相关联的属性文件传送至联合服务器。
2. 移动 XML 配置文件（wrappername.xml）。表 60 显示将该文件移至的目录。

表 60. 将 XML 配置文件移至的目录（按平台）

平台	包装器安装目录
AIX	/usr/opt/db2_08_01/cfg
HP/Sun/Linux	/opt/IBM/db2/V8.1/cfg
Windows	%DB2PATH%\cfg

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

3. 将每个属性文件（wrappername.properties）移至（wrapper_cfg）目录。例如，如果提供对 DB2 控制中心的特定于包装器的添加项的美国英语支持，则移动包装器的美国英语属性文件。表 61 显示将该文件移至的目录。

表 61. 将属性文件移至的目录（按平台）

平台	包装器安装目录
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. 重新启动 DB2 控制中心。

当启动 DB2 控制中心时，它将装入为其安装 XML 配置文件和属性文件的所有数据源的选项。数据源将显示为选项并且伴随有标准 IBM 数据源。

相关概念:

- 第 6 页的『用户如何将数据源添加至联合系统』

相关任务:

- 第 105 页的第 14 章，『安装包装器』
- 第 108 页的『创建 XML 配置文件』

在 DB2 控制中心中支持发现

DB2 控制中心可以发现数据源的功能部件，例如，服务器、视图和昵称。当开发定制包装器时，可选择对 DB2 控制中心的发现功能部件提供支持。当提供发现支持时，用户可对定制数据源以可对标准数据源使用的方式发现昵称、视图和服务器信息。

先决条件:

在开发系统上，必须具有包装器开发者工具箱。

在联合服务器上，必须具有“Java 运行时环境”（JRE）。

过程:

为了支持对定制包装器（数据源）使用发现:

1. 创建指定发现功能的 XML 配置文件和用于包装器的选项:
   a. 启动“开发 XML 配置文件”向导。
   b. 如果用于该包装器的 XML 配置文件存在，则选择修改该文件。如果 XML 配置文件不存在，则选择创建新的 XML 配置文件。遵循以下内容中的指示信息
   c. 在向导的**指定发现功能需求**页上，选择两个选项中的一个以支持发现功能:
      - **支持使用内置具体 Java 类的发现功能**使用随 DB2 控制中心和包装器开发工具箱安装的简单 Java 发现工具。（在 DB2 控制中心安装中，该工具是作为 Java 文件 db2WrapperDiscovery.jar 提供的。在包装器开发工具箱中，它被称为 db2WrapperDiscoverySDK.jar。这两个文件的内容完全相同。）
      - **支持使用定制 Java 类的发现功能**使用提供的定制 Java 类。

d. 在同一页上，为发现类添加任何定制或用户接口选项。

e. 完成向导。

2. 如果选择了内置具体 Java 类，则必须创建用来返回想要从包装器的数据源发现的信息的存储过程。表 62 显示包装器开发工具箱的目录，该工具箱包括可用作示例的 Java 存储过程。

表 62. Java 存储过程的目录（按平台）

平台	包装器安装目录
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

缺省 Windows 目录路径为 C:\Program Files\IBM\SQLLIB。%DB2PATH% 是用来指定 DB2 Information Integrator 在 Windows 上的安装目录路径的环境变量。

样本存储过程是有关内置发现如何工作的 Java 示例。可以使用 C 存储过程、SQL 存储过程或任何其它存储过程来实现发现功能。要根据 cc_plugin 中提供的样本创建定制存储过程:

a. 通过将 sample.java 用作指南，创建将返回想要对数据源发现的信息的存储过程。

b. 更新样本 makefile 以便它引用 Java 存储过程。

c. 使用 makefile 来编译 Java 存储过程。

d. 更新 sample.db2 脚本以便它将存储过程装入到充当联合服务器的 DB2 通用数据库实例中。

e. 运行更新的 sample.db2 脚本以在联合服务器中安装存储过程。

3. 如果选择了定制 Java 类，则必须提供它需要的类和任何支持存储过程。在联合服务器上安装定制 Java 类和任何其它必需的组件。

4. 在联合服务器上安装 XML 配置文件。

5. 重新启动 DB2 命令中心。

结果:

在安装包装器、包装器的 XML 配置文件和存储过程之后，DB2 控制中心用户会将定制数据源视作选项并将能够对该数据源执行发现。DB2 控制中心读取 XML 配置文件以确定它是否应支持对此数据源使用发现，如果支持的话，要对发现工具使用哪个 Java 类。

如果选择发现的内置具体 Java 类，则不需要安装任何其它发现工具。DB2 控制中心将它的内置 Java 工具与定制存储过程配合使用来执行发现。

相关任务:

- 第 108 页的『安装开发 XML 配置文件向导』
- 第 108 页的『创建 XML 配置文件』
- 第 110 页的『安装 XML 配置文件』
- 第 107 页的『将数据源添加至 DB2 控制中心』

第 16 章 测试包装器

下列各节描述如何测试包装器。

使用注册 DLL 语句测试包装器

可使用 SQL 语句（例如，CREATE SERVER 和 ALTER SERVER）测试包装器。例如，要测试包装器基本类的实现，可以运行 CREATE WRAPPER。要测试“服务器”基本类的实现，可以运行 CREATE SERVER。

由您自己确定在 SQL 语句中指定用于进行测试和注册的信息。要作出这样的决定，需要了解这些语句的参数。要了解这一点，请参阅有关 DB2® 通用数据库版本 8 的 *SQL Reference*。

相关任务:

- 第 113 页的『测试包装器的有效选项和无效选项』
- 第 116 页的『从包装器创建跟踪信息』

测试包装器的有效选项和无效选项

建议测试下列情况下的选项信息:

- 指定选项应对包含该选项的 SQL 语句有效。否则，包装器应返回 SQLCODE SQL1881N。
- 已将选项设置的值应对此选项有效。否则，包装器应返回 SQLCODE SQL1882N。

  在某些情况下，选项值无效，原因是它与另一选项值不一致。例如，假设在用户将称为 ENCRYPT 的选项设置为 Y（是，加密此结果集）并将称为 ENCRYPTION_KEY 的选项设置为加密程序所需的符号字符串时，数据源将对查询结果进行编码。很明显，在将 ENCRYPT 设置为 N（否，不加密此结果集）时，符号字符串（或此情况下的任何值）将对 ENCRYPTION_KEY 无效。
- SQL 语句上需要的选项应在该语句中指定。如果缺少所需的选项，则包装器应返回 SQLCODE SQL1883N。
- 一个选项在同一 SQL 语句上应仅指定一次。如果指定多次，则 DB2 返回 SQLCODE SQL1884N。
- 如果用户使用 ALTER [CONSTRUCT] 语句（例如，ALTER SERVER）以添加已定义的选项，则 DB2 返回 SQLCODE SQL1885N。
- 仅当已对该选项设置值时，用户才能使用 ALTER [CONSTRUCT] 语句（例如，ALTER SERVER）以更新或删除选项的值。如果还没有对选项设置值，则 DB2 返回 SQLCODE SQL1886N。

相关概念:

- 第 113 页的『使用注册 DLL 语句测试包装器』

相关任务:

- 第 116 页的『从包装器创建跟踪信息』

第 17 章 跟踪包装器

包装器跟踪设施记录来自包装器的控制流和故障诊断信息。下列各节描述包装器跟踪设施以及如何从包装器创建跟踪信息。

包装器跟踪设施

包装器跟踪设施记录您开发的包装器中的控制流信息（例如，函数入口点和函数出口点）。可以使用此信息来标识开发包装器时的潜在问题并解决部署包装器之后可能遇到的问题。

要获取此故障诊断信息，调用在包装器中指定的区域中的跟踪成员函数。然后，可以使用 **db2trc** 命令来启动并控制跟踪操作。

包装器实用程序类提供了一组成员函数，可以在包装器代码中使用这些函数来检索函数入口点和出口点以及错误跟踪中的数据。下表描述了这些成员函数的用途。

表 63. 包装器实用程序类的跟踪成员函数

C++ 成员函数	Java™ 成员函数	用途
fnc_entry	traceFunctionEntry	记录进入函数操作。
fnc_exit	traceFunctionExit	记录退出函数操作。
fnc_data	traceFunctionData	记录数据跟踪，包括探测点和单个数据元素。
fnc_data2	traceFunctionData	记录数据跟踪，包探测点和两个数据元素（如果需要的话）。
fnc_data3	traceFunctionData	记录数据跟踪，包括探测点和三个数据元素（如果需要的话）。
trace_error	traceError or traceException	记录错误跟踪，包括错误代码和探测点。

在进入包装器代码中的跟踪成员函数之后，可以使用 **db2trc** 命令来启动跟踪设施。跟踪设施收集包装器控制流信息。要从包装器收集此数据，必须发出带有跟踪记录组件标识为 135 的 **db2trc** 命令。组件标识 135 是特定于定制包装器的。

注意: 外部包装器的组件标识可能会更改。有关最新信息，请参阅《*DB2 Information Integrator 发行说明*》。

相关任务:

- 第 116 页的『从包装器创建跟踪信息』

相关参考:

- 第 117 页的『包装器跟踪设施的示例』

从包装器创建跟踪信息

可以使用跟踪设施来记录控制流信息并从包装器获取故障诊断信息。包装器实用程序类提供从包装器收集控制流记录的成员函数。使用 **db2trc** 命令来抽取这些控制流记录并将此信息格式化为可阅读文本。

过程:

要从包装器创建跟踪信息:

1. 发出带有足够缓冲区大小的 **db2trc** 命令。

   例如:

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

2. 在运行跟踪设施一段时间之后，会将跟踪缓冲区的当前内容写至文件。

   例如:

   ```
   db2trc dump trc.dmp
   ```

   此命令将跟踪信息写入当前目录中称为 trc.dmp 的输出文件中。

3. 通过发出以下命令来关闭跟踪设施:

   ```
   db2trc off
   ```

4. 将跟踪输出的内容写至包含来自包装器的错误代码和跟踪流程信息的文件。

   例如:

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

   必须指定掩码选项 `*.*.135.*.*` 以便仅检索与外部包装器相对应的跟踪记录。值 135 是外部包装器的组件标识。

   注意: 外部包装器的组件标识可能会更改。有关最新信息，请参阅《*DB2 Information Integrator 发行说明*》。

5. 将跟踪输出的内容按年月日顺序写至包含来自包装器的跟踪数据的文件。

   例如:

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

相关概念:

- 第 115 页的『包装器跟踪设施』

相关参考:

- 『db2trc - Trace Command』（*Command Reference*）
- 第 117 页的『包装器跟踪设施的示例』

包装器跟踪设施的示例

本主题提供显示如何通过使用包装器代码中的包装器实用程序类来调用跟踪成员函数的示例。本主题还提供了包含来自包装器代码中的跟踪成员函数的控制流信息的 trc.flw 和 trc.fmt 文件。通过发出 **db2trc** 命令来生成跟踪文件。

此示例使用 C++ 方法名。如果要开发 Java 包装器，则替代为相应的 Java 方法名。

以下示例显示使用包装器实用程序类来调用跟踪成员函数的 UnfencedWrapper_Hook 函数。此示例获取有关 fnc_data2 成员函数指示的两个数据元素的跟踪信息。

```
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;
}
```

图 10. 带有跟踪成员函数的包装器代码

然后，可以发出 **db2trc** 命令来启动跟踪设施。

下列示例显示运行跟踪设施之后 trc.flw 和 trc.fmt 文件的内容。

trc.flw 文件的示例:

trc.flw 文件包含包装器跟踪错误代码、函数入口点、函数出口点和数据跟踪点。

`probe 0` 指示对调用跟踪设施的函数名的引用。在 trc.flw 文件中未指定函数名。

```
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
```

图 11. trc.flw 文件

trc.fmt 文件的示例:

trc.fmt 文件包含 trc.flw 文件中的 probe 0 指示的实际函数名。

以下 trc.fmt 文件中的第 1 行和第 6 行分别指示函数入口点和出口点。第 2 行、第 3 行和第 5 行指示对从包装器代码调用跟踪设施的 UnfencedWrapper_Hook 函数的引用。第 4 行显示数据跟踪和数据元素。

```
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
```

图 12. trc.fmt 文件

相关概念:

- 第 115 页的『包装器跟踪设施』

相关任务:

- 第 116 页的『从包装器创建跟踪信息』

相关参考:

- 『db2trc - Trace Command』（*Command Reference*）

词汇表

〔B〕

半结构化数据（semistructured data）： 可包括电子表格、地址簿、配置参数、金融事务或技术图样的数据。“结构化查询语言”（SQL）可以很好地配合结构化数据使用。

包装器（wrapper）： 在联合系统中，这是联合服务器用来与数据源通信并从中检索数据的机制。要实现包装器，联合服务器使用存储在称为包装器模块的库中的例程。这些例程允许联合服务器执行一些操作，如连接至数据源和从数据源重复检索数据。DB2 通用数据库联合实例所有者使用 CREATE WRAPPER 语句来向将包括在联合系统中的每个数据源注册包装器。

不受防护（unfenced）： 与定义在数据库管理器进程中运行的过程、用户定义函数或联合包装器的一种类型或特征有关。当此类型的对象运行时（使用不受防护子句），该对象将不能更改数据库管理器。

不同种类的系统（heterogeneous systems）： 具有各种计算资源的不同系统的集合，这些系统彼此可以是本地的或分布在不同的地区。

〔C〕

超文本传输协议（hypertext transfer protocol（http））： 超文本传输协议（HTTP）是用于分布式合作超媒体信息系统的应用程序级协议。

〔F〕

方案（scenario）： 使用情况的实例。即，方案是在严格指定的假设下对使用情况的执行。方案是在特定系统中通过协作实现的。

非结构化数据（unstructured data）： 以未组织方式存储的任何数据，很有可能存储在传统数据库外部。这种数据的一些示例有文本、声音、影像、传真、图像或图片。

分解（decomposition）： 又称拆分。这是在 DB2 中存储 XML 文档的过程。将 XML 文档分成小块（或拆分）并将各元素作为字段存储在一个或多个 DB2 表中。

分解（decomposition）： 又称拆分。这是在 DB2 中存储 XML 文档的过程。将 XML 文档分成小块（或拆分）并将各元素作为字段存储在一个或多个 DB2 表中。

〔H〕

合成（composition）： 这是从 DB2 表中的数据创建（或合成）XML 文档的过程。生成的 XML 文档中的元素是通过一个或多个 DB2 表中的字段创建的。并且，该 XML 文档可存储在 DB2 中或 DB2 之外，或是存储在文件系统和 MQSeries 消息队列中。

〔J〕

结构化数据（structured data）： 可包括电子表格、地址簿、配置参数、金融事务或技术图样的数据。“结构化查询语言”（SQL）可以很好地配合结构化数据使用。

旧数据（legacy data）： 已拥有和使用的从数十年的信息收集和数据分析得出的数据。此数据通常是当前使用的系统上现有数据库中的记录。旧数据一般是在本地或专有数据库中存在的信息，安全地收藏在数据管理系统中，通常不能用于企业系统。

〔K〕

扩展企业应用程序（extended enterprise applications）： 集成了多个组织中的程序交互作用的应用程序。又称商家到商家应用程序。

〔M〕

名称空间表（namespace table）： “名称空间表”（NST）资源定义从 DB2 XML Extender DTDID 至“XML 模式”（XSD）名称空间和位置的映射。名称空间使您能够在一个 XML 文档中混合多个 XML 词汇表中的元素（有些时候是属性）名。

〔N〕

昵称（nickname）： （1）在联合系统中，在查询中用来引用数据源中的对象的标识。昵称标识的对象被称为数据源对象。数据源对象的示例包括表、视图、同义词、表结构文件和搜索算法。（2）DB2 Information Integrator 中定义的名称，用来表示非 DB2 关系数据库中的物理数据库对象（例如，表或存储过程）。

〔S〕

使用情况（use case）： 特定系统中可标识且可外部观察的行为。它是由使用者或用户启动的使用模式，执行或用于执行一些有用的工作。使用情况表示使用者与系统之间的对话。例如，“从支出帐户中取出资金”就是一种使用情况。

受防护（fenced）： 与定义在不同于数据库管理器的进程中运行的过程、用户定义函数或联合包装器的一种类型或特征有关。当此类型的对象运行时（使用受防护子句），该对象将不能修改数据库管理器。

〔X〕

协作（collaboration）： 实现使用情况方案的一系列操作。又称用户对用户。它通常涉及两个或多个组件之间的协作。作为一个示例，考虑在客户机服务器系统中更新客户详细信息的方案。它们是这样的一系列操作：其中图形用户界面（GUI）组件显示窗口、使用对数据的请求调用数据服务器组件、显示客户详细信息（并改正它们）以及调用数据服务器以执行更新。组件之间的组件操作和交换的整个模式就是“实现”该方案的协作。

信息聚集应用程序（information aggregation applications）： 其中工具从其它数据源抽取信息的应用程序。又称用户对数据。

信息性数据（informational data）： 从可操作数据抽取出来然后进行变换以用于决策的数据。另见可操作数据。

信息性数据（informational data）： 从可操作数据抽取出来然后进行变换以用于决策的数据。另见可操作数据。

〔Z〕

自助应用程序（self-service applications）： 又称“用户对企业”应用程序，其中用户与企业事务和数据进行交互

D

DB2 管理员（DB2 Administrator）： 负责诸如存取授权和内容管理之类管理任务的人。管理员还可以对用户授予各种级别的权限。

E

EJB（enterprise java bean）： EJB 是可与其它企业 bean 及其它 Java 组件组合以创建分布式应用程序的 Java 组件。有两种类型的企业 bean：实体 bean 和会话 bean。

J

Java 2 Platform, Enterprise Edition（Java 2 Platform, Enterprise Edition（J2EE））： 一个平台，提供多层分布式应用程序模型、重用组件的能力、集成基于“可扩展标记语言”（XML）的数据交换、统一安全性模型以及灵活的事务控制。

W

WORF： Web 对象运行时框架。支持 DB2 Web 服务提供程序的运行时引擎。

X

XSD： “可扩展标记语言”（XML）模式定义。用于描述包含模式的 XML 文件的语言。

辅助功能

辅助功能部件可帮助那些身体有某些缺陷（如活动不方便或视力不太好）的用户成功地使用软件产品。以下列表指定 DB2® V8 产品中的主要辅助功能部件:

- 所有 DB2 功能可使用键盘（而不是鼠标）导航来实现。有关更多信息，请参阅『键盘输入和导航』。
- 可定制 DB2 用户界面上的字体大小和颜色。有关更多信息，请参阅『界面显示的辅助功能』。
- DB2 产品支持使用 Java™ Accessibility API 的辅助功能应用程序。有关更多信息，请参阅第 124 页的『与辅助技术的兼容性』。
- DB2 文档是以易使用格式提供的。有关更多信息，请参阅第 124 页的『文档的辅助功能』。

键盘输入和导航

键盘输入

只使用键盘就可以操作 DB2 工具。使用键或键组合就可以执行使用鼠标所能完成的操作。标准操作系统击键用于标准操作系统操作。

有关使用键或键组合执行操作的更多信息，请参阅 键盘快捷方式和加速键: 公共 GUI 帮助。

键盘导航

可使用键或键组合来导航 DB2 工具用户界面。

有关使用键或键组合来导航 DB2 工具的更多信息，请参阅 键盘快捷方式和加速键: 公共 GUI 帮助。

键盘焦点

在 UNIX® 操作系统中，击键操作起作用的活动窗口的区域将突出显示。

界面显示的辅助功能

DB2 工具所具有的功能部件使视力不太好的用户更易使用。这些辅助功能方面的增强包括了对可定制字体属性的支持。

字体设置

可使用“工具设置”笔记本来选择菜单和对话框窗口中文本的颜色、大小和字体。

有关指定字体设置的更多信息，请参阅 更改菜单和文本的字体: 公共 GUI 帮助。

不依赖于颜色

不需要分辨颜色就可以使用此产品中的任何功能。

与辅助技术的兼容性

DB2 工具界面支持 Java Accessibility API，它使您能够将屏幕阅读器和其它辅助技术与 DB2 产品配合使用。

文档的辅助功能

DB2 的相关文档是以 XHTML 1.0 格式提供的，它在大部分 Web 浏览器中是可查看的。XHTML 允许您根据浏览器中设置的显示首选项来查看文档。还允许您使用屏幕阅读器和其它辅助技术。

语法图是以点分十进制格式提供的。仅当使用屏幕阅读器访问联机文档时，此格式才可用。

相关概念:

- *基础结构主题（DB2 公共文件）*中的『点分十进制语法图』

相关任务:

- 『键盘快捷方式和加速键: 公共 GUI 帮助』
- 『更改菜单和文本的字体: 公共 GUI 帮助』

声明

此信息是为在美国提供的产品和服务编写的。IBM 可能在所有国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation 以 “按现状” 的基础提供本出版物，不附有任何形式的（无论是明示的还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证。因此，本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息，而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Corporation
J46A/G4
555 Bailey Avenue

San Jose, CA 95141-1003
U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本资料中包含用于日常业务运作的数据和报表的示例。为了尽可能完整地说明问题，这些示例可能包含个人、公司、品牌和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有雷同，纯属巧合。

版权许可证:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。用户如果是为按照 IBM 的应用程序编程接口开发、使用、经销或分发应用程序，则可以任何形式复制、修改和分发这些样本程序，而无须向 IBM 付费。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（*贵公司的名称*）（*年*）。此部分代码是根据 IBM 公司的样本程序衍生出来的。
© Copyright IBM Corp. （*输入年份*）。All rights reserved.

商标

下列各项是国际商业机器公司在美国和／或其他国家或地区的商标:

IBM
AIX
DB2
Java
Windows

下列各项是其他公司的商标或注册商标:

Java 和所有基于 Java 的商标和徽标是 Sun Microsystems, Inc. 在美国和／或其他国家或地区的商标或注册商标。

Microsoft、Windows、Windows NT 和 Windows 徽标是 Microsoft Corporation 在美国和／或其他国家或地区的商标。

Intel、Intel Inside（徽标）、MMX 和 Pentium 是 Intel Corporation 在美国和／或其他国家或地区的商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

[A]

[B]

[C]

[D]

[F]

[G]

[H]

[J]

[K]

[T]

[W]

[X]

[Y]

[Z]

B

C

D

F

J

L

N

P

R

S

U

W

X

与 IBM 联系

在中国，请致电下列其中一个号码以与 IBM 联系:

- 800-810-1818 或 (010) 84981188 分机 5151，可获得售前客户服务;
- 800-810-1818 或 (010) 84981188 分机 5200，可获得售后客户服务;
- 800-810-1818 或 (010) 84981188 分机 5017，可获得市场营销与销售的信息;

要查找您所在国家或地区的 IBM 营业处，可在网上查看 IBM 全球联系人目录（Directory of Worldwide Contacts），网址为: www.ibm.com/planetwide。

产品信息

关于 DB2 Information Integrator 的信息可通过万维网获取，网址为: http://www-900.ibm.com/cn/software/db2/。

此站点包含有关 DB2 产品家族、DB2 解决方案、技术前沿与趋势、DB2 服务、成功案例、市场活动、培训与认证、DB2 开发者园地、合作伙伴、下载中心、资料库、第三方分析报告、殊荣与奖项、DB2 新闻以及如何购买 DB2 的最新信息。

要查找您所在国家或地区的 IBM 营业处，可在网上查看 IBM 全球联系人目录（Directory of Worldwide Contacts），网址为: www.ibm.com/planetwide。

对文档的意见

您的反馈有助于 IBM 提供高质量的信息。请发送您对本书或其它 DB2 Information Integrator 文档的任何意见。可以使用下列任何一种方法提出意见:

- 使用 www.ibm.com/software/data/rcf 上的在线读者意见表发送您的意见。
- 通过电子邮件（e-mail）将您的意见发送至 ctscrcf@cn.ibm.com。确保包括产品的名称、产品的版本号和书籍的名称及部件号（如果适用的话）。如果您对特定文本有意见，请包括此文本的位置（例如，标题、表号或页码）。

中国印刷

S152-0845-00

Spine information:

IBM DB2 Information Integrator

包装器开发者指南

版本 8.2