IBM DB2 Information Integrator

开发包装器的 C++ API 参考

版本 8.2

S152-0844-00

IBM DB2 Information Integrator

开发包装器的 C++ API 参考

版本 8.2

S152-0844-00

在使用本资料及其支持的产品之前，请阅读第 183 页的『声明』中的一般信息。

本文档包含 IBM 的专利信息。它在许可证协议下提供，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

本书提供了有关 C++ API 类的参考信息，在为数据源开发包装器时您可以使用这些类。每个类的概述中都包含一般描述和用法信息。然后列示了每个 C++ 类的成员函数（以及构造函数和析构函数，如果适用的话）及相应的用途、语法、返回值、必需的输入自变量和输出自变量。为数据源开发包装器之后，就可以在联合数据库系统中使用该数据源了。

本书的读者

本书是为将 API 与 IBM® 提供的 DB2® Information Integrator 配合使用的 DBA 和包装器开发者编写的。

本书中使用的约定和术语

突出显示约定:

本书使用了下列突出显示约定:

粗体

指示命令和图形用户界面控件（例如，字段名、按钮名和菜单选项）。粗体用来指定注释、限制、先决条件和建议。

等宽字体

指示您输入的文本、文件名和代码示例。等宽字体还用于 SQL 语句或 DB2 命令参数名。

斜体

指示 SQL 语句或者将用适当值来替换的 DB2 命令参数值。SQL 语句或 DB2 命令示例将使用斜体来表示样本参数值。斜体还用来强调词语、标识新术语和指示文档标题。

大写类型

指示 DB2 命令和 SQL 语句的名称以及它们的关键字。数据类型名称、选项和只取首字母的缩写词也使用大写。

C++ 类和方法

下列各节描述了可以与 C++ API 配合使用的类。这些类包括:

- 目录类
- 包装器类
- 服务器类
- 用户类
- 昵称类
- 远程连接类
- 操作类
- 请求类
- 数据类
- 包装器实用程序类

每个类的概述中都包含一般描述和用法信息。然后列示了每个类的成员函数（以及构造函数和析构函数，如果有的话）的相应用途、语法、返回值和必需的输入自变量和输出自变量。

C++ API 的目录类

下表描述了 C++ API 的每个目录类。

表 1. 目录类

类名	描述
Catalog_Option	该类表示单值或多值的已命名目录选项。子类将区分这些选项。不要实例化此类或者创建此类的子类。
Wrapper_Info	包含包装器的目录信息的 Catalog_Info 的子类。
Server_Info	包括服务器对象的目录信息（CREATE SERVER 和 ALTER SERVER 语句中的数据）的类。
User_Info	包括 CREATE USER MAPPING 和 ALTER USER MAPPING 语句中的用户映射的目录信息的类。
Column_Info	包括昵称列的目录信息的类。此类包括列统计信息。
Nickname_Info	包括目录中的昵称定义并包括列和索引定义的类。

相关参考:

- 第 2 页的『Catalog_Option 类（C++）』
- 第 4 页的『Wrapper_Info 类（C++）』
- 第 10 页的『Server_Info 类（C++）』
- 第 18 页的『User_Info 类（C++）』
- 第 23 页的『Column_Info 类（C++）』
- 第 39 页的『Nickname_Info 类（C++）』

Catalog_Option 类（C++）

本主题描述 Catalog_Option 类并提供每个成员函数的详细信息。

概述

Catalog_Option 类表示命名目录选项。目录选项可以是单值也可以是多值，并且按子类区分。

Catalog_Option 类是 C++ API 的其中一个目录类。

用法 DB2 联合服务器实例化在目录中或在“数据定义语言”（DDL）语句上指定的每个选项的已命名目录选项。包装器可以实例化这些已命名目录选项（通过 Catalog_Info 子类的 add_option() 方法）以添加或更改选项值。

文件 sqlqg_catalog.h

数据成员

无。

类型 Catalog_Option::action

类型 enum

值

值	描述
sqlqg_None	未对此选项进行操作
sqlqg_Add	将此选项添加至目录对象
sqlqg_Set	为目录对象更改此选项的值
sqlqg_Drop	从目录对象中删除此选项
sqlqg_SetSO	未被非相关包装器所用

成员函数

下表描述 Catalog_Option 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 2. Catalog_Option 类的成员函数

成员函数	描述
get_name	检索选项名。
get_action	检索此选项的操作（ADD、SET、DROP 或无）。
get_value	检索选项值。
get_value	检索选项值和长度。

get_name 成员函数

用途 检索选项名。

语法

```
sqluint8* get_name ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的选项名。

get_action 函数

用途 检索此选项的操作（ADD、SET、DROP 或无）。

语法

```
Catalog_Option::Action get_action ()
```

输入自变量

无。

输出自变量

无。

返回值 操作。

get_value 函数

用途 检索选项值。

语法

```
virtual sqluint8* get_value ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的选项值。

get_value 函数

用途 检索选项值和长度。

语法

```
virtual sqluint8* get_value (sqlint32* a_length)
```

输入自变量

无。

输出自变量

表 3. get_value 成员函数的输出自变量

名称	数据类型	描述
a_length	sqlint32*	选项值的长度。

返回值 以 null 结束的选项值。

相关参考:

- 第 1 页的『C++ API 的目录类』

Wrapper_Info 类（C++）

本主题描述 Wrapper_Info 类并提供构造函数和成员函数的详细信息。

概述

Wrapper_Info 类是 Catalog_Info 的子类并包含包装器的目录信息。

Wrapper_Info 类是 C++ API 的其中一个目录类。

用法 此类由 DB2 联合服务器实例化以包含 CREATE WRAPPER 语句或 DB2 Information Integrator 目录中的信息。当在 CREATE WRAPPER 或 ALTER WRAPPER 操作期间添加信息时，此类由包装器实例化。

文件 sqlqg_catalog.h

数据成员

无。

构造函数和成员函数

下列各表描述 Wrapper_Info 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 4. Wrapper_Info 类的构造函数

构造函数	描述
Wrapper_Info	构造 Wrapper_Info 的实例。

表 5. Wrapper_Info 类的成员函数

成员函数	描述
get_wrapper_name	检索包装器名。
get_corelib	检索包装器库的名称。
set_type	设置包装器类型。
get_type	检索包装器类型。
set_version	设置包装器代码版本。
get_version	检索包装器版本，如果存在的话。
copy	复制 Wrapper_Info 对象。
add_option	将选项添加至目录信息。
drop_option	从 Catalog_Info 类中删除选项。
get_option	按名称检索目录选项。
get_first_option	检索指向选项链中第一个选项的指针。
get_next_option	检索选项链中的下一个选项。

Wrapper_Info 构造函数

用途 构造 Wrapper_Info 的实例。

语法

```
Wrapper_Info ()
```

输入自变量

无。

输出自变量

无。

get_wrapper_name 函数

用途 检索包装器名。

语法

```
sqlint32 get_wrapper_name (sqluint8** a_name)
```

输入自变量

无。

输出自变量

表 6. get_wrapper_name 成员函数的输出自变量

名称	数据类型	描述
a_name	sqluint8**	指向以 null 结束的包装器名称字符串的指针。

返回值 返回码。如果显示名称，则值为 0。如果不显示名称，则值为 SQLQG_NOVALUE。

get_corelib 成员函数

用途 检索包装器库的名称。此名称是 CREATE WRAPPER 语句中的包装器库的基本名。

语法

```
sqlint32 get_corelib (sqluint8** a_lib_name)
```

输入自变量

无。

输出自变量

表 7. get_corelib 成员函数的输出自变量

名称	数据类型	描述
a_lib_name	sqluint8**	指向以 null 结束的库名字符串的指针。

返回值 返回码。如果显示名称，则值为 0。如果不显示名称，则值为 SQLQG_NOVALUE。

set_type 函数

用途 设置包装器类型。

语法

```
void set_type (sqluint8 a_wrapper_type)
```

输入自变量

表 8. set_type 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_type	sqluint8	包装器类型（必须为 N）。

输出自变量

无。

返回值 无。

get_type 函数

用途 检索包装器类型。

语法

```
sqlint32 get_type (sqluint8* a_wrapper_type)
```

输入自变量

无。

输出自变量

表 9. get_type 成员函数的输出自变量

名称	数据类型	描述
a_wrapper_type	sqluint8*	包装器类型。

返回值 返回码。如果显示类型，则值为 0。如果不显示类型，则值为 SQLQG_NOVALUE。

set_version 函数

用途 设置包装器代码版本。

语法

```
void set_version (sqlint32 a_wrapper_version)
```

输入自变量

表 10. set_version 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_version	sqlint32	包装器代码版本。

输出自变量

无。

返回值 无。

get_version 成员函数

用途 检索包装器版本，如果存在的话。

语法

```
sqlint32 get_version (sqlint32* a_wrapper_version)
```

输入自变量

无。

输出自变量

表 11. get_version 成员函数的输出自变量

名称	数据类型	描述
a_wrapper_version	sqlint32*	包装器代码版本。

返回值 返回码。如果显示版本，则值为 0。如果不显示版本，则值为 SQLQG_NOVALUE。

copy 函数

用途 复制 Wrapper_Info 对象。

语法

```
sqlint32 copy (Wrapper_Info** a_new_wrapper_info)
```

输入自变量

无。

输出自变量

表 12. copy 成员函数的输出自变量

名称	数据类型	描述
a_new_wrapper_info	Wrapper_Info**	指向新分配的 Wrapper_Info 的指针。

返回值 返回码。如果值为 0，则指示成功。

add_option 函数

定义位置

Catalog_Info

用途 将选项添加至目录信息。

用法 当在 CREATE WRAPPER 和 ALTER WRAPPER 语句处理期间将包装器生成的选项添加至目录时，包装器可以调用此成员函数。

语法

```
sqlint32 add_option (sqluint8*  a_opt_name,
                     sqlint32   a_name_len,
                     sqluint8*  a_opt_value,
                     sqlint32   a_value_len,
                     Catalog_Option::Action a_action
                     = Catalog_Option::sqlqg_None,
                       char*    a_option_type = "")
```

输入自变量

表 13. add_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束的）。
a_value_len	sqlint32	选项值的长度。
a_action	Catalog_Option::Action	此选项的操作（ADD、SET、DROP 或无）。
a_option_type	char*	在 SQLN1884 错误消息中使用的标记（如果这是重复选项的话）。对于包装器选项，使用在 sqlqg_misc.h 头文件中定义的 SQLQG_WRAPPER_OPTION 常量。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

drop_option 函数

定义位置

Catalog_Info

用途 从 Catalog_Info 类中删除选项。此成员函数不会从目录中删除选项。在将带有 Catalog_Option::sqlqg_Drop 操作的选项添加至增量 Catalog_Info 对象时，会从目录中删除该选项。

语法

```
sqlint32 drop_option (Catalog_Option* a_option)
```

输入自变量

表 14. drop_option 成员函数的输入自变量

名称	数据类型	描述
a_option	Catalog_Option*	要删除的选项。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_option 函数

定义位置

Catalog_Info

用途 按名称检索目录选项。

用法 如果未找到该选项，则输出自变量 option 为空。

语法

```
sqlint32 get_option (sqluint8*        a_opt_name,
                     sqlint32         a_name_len,
                     Catalog_Option** a_option)
```

输入自变量

表 15. get_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。

输出自变量

表 16. get_option 成员函数的输出自变量

名称	数据类型	描述
a_option	Catalog_Option**	找到选项。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到该选项。

get_first_option 函数

定义位置

Catalog_Info

用途 检索指向选项链中第一个选项的指针。

语法

```
Catalog_Option* get_first_option ()
```

输入自变量

无。

输出自变量

无。

返回值 指向选项链中第一个选项的指针。如果链是空的，则该值为空。

get_next_option 函数

定义位置

Catalog_Info

用途 检索选项链中的下一个选项。

语法

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

输入自变量

表 17. get_next_option 成员函数的输入自变量

名称	数据类型	描述
a_current_option	Catalog_Option*	当前选项。

输出自变量

无。

返回值 指向选项链中下一个选项的指针。如果在结尾，则该值为空。

相关参考:

- 第 1 页的『C++ API 的目录类』

Server_Info 类（C++）

本主题描述 Server_Info 类并提供构造函数和成员函数的详细信息。

概述

Server_Info 类包括服务器对象的目录信息（CREATE SERVER 和 ALTER SERVER 语句中的数据）。

Server_Info 类是 C++ API 的其中一个目录类。

用法 此类由 DB2 联合服务器实例化以包含 CREATE SERVER 或 ALTER SERVER 语句中的信息或包含 DB2 Information Integrator 目录中的信息。当在 CREATE SERVER 或 ALTER SERVER 语句操作期间添加信息时，此类还会由包装器实例化。

文件 sqlqg_catalog.h

数据成员

无。

构造函数和成员函数

下列各表描述 Server_Info 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 18. Server_Info 类的构造函数

构造函数	描述
Server_Info	构造空的 Server_Info 对象。
Server_Info	构造具有特定参数的 Server_Info 对象。

表 19. Server_Info 类的成员函数

成员函数	描述
add_option	将单值选项添加至目录信息。
drop_option	从 Catalog_Info 类中删除选项。
get_option	按名称检索目录选项。

表 19. Server_Info 类的成员函数 (续)

成员函数	描述
get_first_option	检索指向选项链中第一个选项的指针。
get_next_option	检索选项链中的下一个选项。
get_basic_info	从 Server_Info 对象中检索基本信息。
get_server_name	从 Server_Info 中检索服务器名（如果该名称有效的话）。
set_server_type	设置服务器的服务器类型。
get_server_type	从 Server_Info 中检索服务器类型（如果该类型有效的话）。
set_server_version	设置服务器的服务器版本。
get_server_version	从 Server_Info 中检索服务器版本（如果该版本有效的话）。
get_wrapper_name	从 Server_Info 中检索包装器名（如果该名称有效的话）。
merge	将增量 Server_Info 对象合并到当前对象中。
copy	复制 Server_Info 对象。

Server_Info 构造函数

用途 构造空的 Server_Info 对象。

语法

```
Server_Info ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

Server_Info 构造函数

用途 构造具有特定参数的 Server_Info 对象。

语法

```
Server_Info (sqlint32*   a_rc,
             sqluint8*   a_server_name,
             sqlint32    a_name_len,
             sqluint8*   a_server_type,
             sqlint32    a_type_len,
             sqluint8*   a_server_version,
             sqlint32    a_ver_len,
             sqluint8*   a_server_wrapper,
             sqlint32    a_wrap_len);
```

输入自变量

表 20. Server_Info 构造函数的输入自变量

名称	数据类型	描述
a_server_name	sqluint8*	服务器名（不是以 null 结束的）。
a_name_len	sqlint32	服务器名的长度。
a_server_type	sqluint8*	服务器类型（不是以 null 结束的）。
a_type_len	sqlint32	服务器类型的长度。
a_server_version	sqluint8*	服务器版本（不是以 null 结束的）。
a_ver_len	sqlint32	服务器版本的长度。
a_server_wrapper	sqluint8*	包装器的名称（不是以 null 结束的）。
a_wrap_len	sqlint32	包装器名称的长度。

输出自变量

表 21. Server_info 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

add_option 函数

定义位置

Catalog_Info

用途 将单值选项添加至目录信息。

用法 当在 CREATE SERVER 或 ALTER SERVER 语句处理期间将包装器生成的选项添加至目录时，包装器可以调用此成员函数。

语法

```
sqlint32 add_option (sqluint8*  a_opt_name,
                     sqlint32   a_opt_name_len,
                     sqluint8*  a_opt_value,
                     sqlint32   a_opt_value_len,
                     Catalog_Option::Action     a_act
                   = Catalog_Option::sqlqg_None,
                     char*      a_opt_type = "")
```

输入自变量

表 22. add_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_opt_name_len	sqlint32	选项名的长度。

表 22. add_option 成员函数的输入自变量 (续)

名称	数据类型	描述
a_opt_value	sqluint8*	选项值（不是以 null 结束的）。
a_opt_value_len	sqlint32	选项值的长度。
a_act	Catalog_Option::Action	此选项的操作（ADD、SET、DROP 或无）。
a_opt_type	char*	在 SQLN1884 错误消息中使用的标记（如果这是重复选项的话）。使用在 sqlqg_misc.h 头文件中定义的 SQLQG_SERVER_OPTION 常量。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

drop_option 函数

定义位置

Catalog_Info

用途 从 Catalog_Info 类中删除选项。此成员函数不会从目录中删除选项。在将带有 Catalog_Option::sqlqg_Drop 操作的选项添加至增量 Catalog_Info 对象时，将删除该选项。

语法

```
sqlint32 drop_option (Catalog_Option* a_option)
```

输入自变量

表 23. drop_option 成员函数的输入自变量

名称	数据类型	描述
a_option	Catalog_Option*	要删除的选项。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_option 函数

定义位置

Catalog_Info

用途 按名称检索目录选项。

用法 如果未找到该选项，则输出自变量 `option` 为空。

语法

```
sqlint32 get_option (sqluint8*        a_opt_name,
                     sqlint32         a_name_len,
                     Catalog_Option** a_option)
```

输入自变量

表 24. get_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。

输出自变量

表 25. get_option 成员函数的输出自变量

名称	数据类型	描述
a_option	Catalog_Option**	找到选项。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到该选项。

get_first_option 函数

定义位置

Catalog_Info

用途 检索指向选项链中第一个选项的指针。

语法

```
Catalog_Option* get_first_option ()
```

输入自变量

无。

输出自变量

无。

返回值 指向选项链中第一个选项的指针。如果链是空的，则该值为空。

get_next_option 函数

定义位置

Catalog_Info

用途 检索选项链中的下一个选项。

语法

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

输入自变量

表 26. get_next_option 成员函数的输入自变量

名称	数据类型	描述
a_current_option	Catalog_Option*	当前选项。

输出自变量

无。

返回值 指向链中下一个选项的指针。如果在结尾，则该值为空。

get_basic_info 函数

用途 从 Server_Info 对象中检索基本信息。与其它 get_XXX 成员例程不同，get_basic_info 成员函数返回 SQLQG_ERROR 并记录错误（如果有任何无效的数据项的话）。

语法

```
sqlint32 get_basic_info (sqluint8** a_server_name,
                         sqluint8** a_server_type,
                         sqluint8** a_server_version,
                         sqluint8** a_server_wrapper)
```

输入自变量

无。

输出自变量

表 27. get_basic_info 成员函数的输出自变量

名称	数据类型	描述
a_server_name	sqluint8**	指向以 null 结束的服务器名的指针。
a_server_type	sqluint8**	指向以 null 结束的服务器类型的指针。
a_server_version	sqluint8**	指向以 null 结束的服务器版本的指针。
a_server_wrapper	sqluint8**	指向以 null 结束的包装器名的指针。

返回值 返回码。如果值为 0，则指示成功。

get_server_name 函数

用途 从 Server_Info 中检索服务器名（如果该名称有效的话）。

语法

```
sqlint32 get_server_name (sqluint8** a_server_name)
```

输入自变量

无。

输出自变量

表 28. get_server_name 成员函数的输出自变量

名称	数据类型	描述
a_server_name	sqluint8**	指向以 null 结束的服务器名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置服务器名。

set_server_type 函数

用途 设置服务器的服务器类型。

用法 在 CREATE SERVER 或 ALTER SERVER 语句处理期间，包装器可以调用此成员函数以在初始语句中未显示缺省值时提供缺省值。

语法

```
sqlint32 set_server_type (sqluint8* a_server_type,
                          sqlint32  a_server_type_len)
```

输入自变量

表 29. set_server_type 成员函数的输入自变量

名称	数据类型	描述
a_server_type	sqluint8*	服务器类型（不是以 null 结束的）。
a_server_type_len	sqlint32	服务器类型的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_server_type 函数

用途 从 Server_Info 中检索服务器类型（如果该类型有效的话）。

语法

```
sqlint32 get_server_type (sqluint8** a_server_type)
```

输入自变量

无。

输出自变量

表 30. get_server_type 成员函数的输出自变量

名称	数据类型	描述
a_server_type	sqluint8**	指向以 null 结束的服务器类型的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置服务器类型。

set_server_version 函数

用途 设置服务器的服务器版本。

用法 在 CREATE SERVER 或 ALTER SERVER 语句处理期间，包装器可以调用此成员函数以在未在 DDL 中指定缺省服务器版本时提供缺省服务器版本。

语法

```
sqlint32 set_server_version (sqluint8* a_server_version,
                             sqlint32  a_server_version_len)
```

输入自变量

表 31. set_server_version 成员函数的输入自变量

名称	数据类型	描述
a_server_version	sqluint8*	服务器版本（不是以 null 结束的）。
a_server_version_len	sqlint32	服务器版本的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_server_version 函数

用途 从 Server_Info 中检索服务器版本（如果该版本有效的话）。

语法

```
sqlint32 get_server_version (sqluint8** a_server_version)
```

输入自变量

无。

输出自变量

表 32. get_server_version 成员函数的输出自变量

名称	数据类型	描述
a_server_version	sqluint8**	指向以 null 结束的服务器版本的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置服务器版本。

get_wrapper_name 函数

用途 从 Server_Info 中检索包装器名（如果该名称有效的话）。

语法

```
sqlint32 get_wrapper_name (sqluint8** a_wrapper_name)
```

输入自变量

无。

输出自变量

表 33. get_wrapper_name 成员函数的输出自变量

名称	数据类型	描述
a_wrapper_name	sqluint8**	指向以 null 结束的包装器名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置包装器名。

merge 函数

用途 将增量 Server_Info 对象合并到当前对象中。

语法

```
sqlint32 merge (Server_Info* a_delta_info)
```

输入自变量

表 34. merge 成员函数的输入自变量

名称	数据类型	描述
a_delta_info	Server_Info*	增量对象。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

copy 函数

用途 复制 Server_Info 对象。

语法

```
sqlint32 copy (Server_Info** a_new_server_info)
```

输入自变量

无。

输出自变量

表 35. copy 成员函数的输出自变量

名称	数据类型	描述
a_new_server_info	Server_Info**	指向新的 Server_Info 对象的指针。

返回值 返回码。如果值为 0，则指示成功。

相关参考:

- 第 1 页的『C++ API 的目录类』

User_Info 类（C++）

本主题描述 User_Info 类并提供构造函数和成员函数的详细信息。

概述

User_Info 类包括 CREATE USER MAPPING 和 ALTER USER MAPPING 语句中的用户映射的目录信息。

User_Info 类是 C++ API 的其中一个目录类。

用法 此类由 DB2 联合服务器实例化以包含 CREATE USER MAPPING 或 ALTER USER MAPPING 语句中的信息或包含 DB2 Information Integrator 目录中的信息。当在 CREATE USER MAPPING 或 ALTER USER MAPPING 语句处理期间添加信息时，此类由包装器实例化。

文件 sqlqg_catalog.h

数据成员

无。

构造函数和成员函数

下列各表描述 User_Info 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 36. User_Info 类的构造函数

构造函数	描述
User_Info	构造缺省（空）User_Info 对象。

表 37. User_Info 类的成员函数

成员函数	描述
get_server_name	检索此用户映射的服务器名。
get_authid	检索此用户映射的授权标识。
merge	将增量 User_Info 对象合并到当前对象中。
copy	创建当前 User_Info 对象的副本。
add_option	将单值选项添加至目录信息。
drop_option	从 Catalog_Info 类中删除选项。
get_option	按名称检索目录选项。
get_first_option	检索指向选项链中第一个选项的指针。
get_next_option	检索选项链中的下一个选项。

User_Info 构造函数

用途 构造缺省（空）User_Info 对象。

语法

```
User_Info ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

get_server_name 函数

用途 检索此用户映射的服务器名。

语法

```
sqlint32 get_server_name (sqluint8** a_server_name)
```

输入自变量

无。

输出自变量

表 38. get_server_name 成员函数的输出自变量

名称	数据类型	描述
a_server_name	sqluint8**	指向以 null 结束的服务器名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示服务器名无效或未设置。

get_authid 函数

用途 检索此用户映射的授权标识。

语法

```
sqlint32 get_authid (sqluint8** a_authid)
```

输入自变量

无。

输出自变量

表 39. get_authid 成员函数的输出自变量

名称	数据类型	描述
a_authid	sqluint8**	指向以 null 结束的授权标识的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示标识无效或未设置。

merge 函数

用途 将增量 User_Info 对象合并到当前对象中。

语法

```
sqlint32 merge (User_Info* a_delta_info)
```

输入自变量

表 40. merge 成员函数的输入自变量

名称	数据类型	描述
a_delta_info	User_Info*	要合并的增量对象。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

copy 函数

用途 创建当前 User_Info 对象的副本。

语法

```
sqlint32 copy (User_Info** a_new_user_info)
```

输入自变量

无。

输出自变量

表 41. copy 成员函数的输出自变量

名称	数据类型	描述
a_new_user_info	User_Info**	指向复制对象的指针。

返回值 返回码。如果值为 0，则指示成功。

add_option 函数

用途 将单值选项添加至目录信息。

用法 包装器可以在 CREATE USER MAPPING 或 ALTER USER MAPPING 语句处理期间调用此成员函数以添加包装器生成的选项。

语法

```
sqlint32 add_option (sqluint8*  a_opt_name,
                     sqlint32   a_opt_name_len,
                     sqluint8*  a_opt_value,
                     sqlint32   a_opt_value_len,
                     Catalog_Option::Action     a_act
                   = Catalog_Option::sqlqg_None,
                     char*      option_type = "")
```

输入自变量

表 42. add_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束的）。
a_opt_value_len	sqlint32	选项值的长度。
a_act	Catalog_Option::Action	此选项的操作（ADD、SET、DROP 或无）。
a_opt_type	char*	在 SQLN1884 错误消息中使用的标记（如果这是重复选项的话）。使用在 sqlqg_misc.h 头文件中定义的 SQLQG_USER_OPTION 常量。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

drop_option 函数

定义位置

Catalog_Info

用途 从 Catalog_Info 类中删除选项。此成员函数不会从目录中删除选项。在将带有 Catalog_Option::sqlqg_Drop 操作的选项添加至增量 Catalog_Info 对象时，将删除该选项。

语法

```
sqlint32 drop_option (Catalog_Option* a_option)
```

输入自变量

表 43. drop_option 成员函数的输入自变量

名称	数据类型	描述
a_option	Catalog_Option*	要删除的选项。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_option 函数

定义位置

Catalog_Info

用途 按名称检索目录选项。

用法 如果未找到该选项，则输出自变量 `option` 为空。

语法

```
sqlint32 get_option (sqluint8*        a_opt_name,
                     sqlint32         a_name_len,
                     Catalog_Option** a_option)
```

输入自变量

表 44. get_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。

输出自变量

表 45. get_option 成员函数的输出自变量

名称	数据类型	描述
a_option	Catalog_Option**	找到选项。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到该选项。

get_first_option 函数

定义位置

Catalog_Info

用途 检索指向选项链中第一个选项的指针。

语法

```
Catalog_Option* get_first_option ()
```

输入自变量

无。

输出自变量

无。

返回值 指向选项链中第一个选项的指针。如果链是空的，则该值为空。

get_next_option 函数

定义位置

Catalog_Info

用途 检索选项链中的下一个选项。

语法

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

输入自变量

表 46. get_next_option 成员函数的输入自变量

名称	数据类型	描述
a_current_option	Catalog_Option*	当前选项。

输出自变量

无。

返回值 指向链中下一个选项的指针。如果在结尾，则该值为空。

相关参考:

- 第 1 页的『C++ API 的目录类』

Column_Info 类（C++）

本主题描述 Column_Info 类并提供构造函数和成员函数的详细信息。

概述

Column_Info 类包括昵称列的目录信息。此类包括列统计信息。

Column_Info 类是 C++ API 的其中一个目录类。

用法 此类由 DB2 联合服务器实例化以包含 CREATE NICKNAME 或 ALTER NICKNAME 语句中的信息或包含 DB2 Information Integrator 目录中的信息。当在 CREATE NICKNAME 或 ALTER NICKNAME 语句操作期间添加信息时，此类由包装器实例化。

文件 sqlqg_catalog.h

数据成员

无。

构造函数和成员函数

下列各表描述 Column_Info 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 47. Column_Info 类的构造函数

构造函数	描述
Column_Info	构造缺省（空）Column_Info 对象。

表 48. Column_Info 类的成员函数

成员函数	描述
set_column_name	设置列的名称。
get_column_name	检索此列的名称。
get_new_column_name	检索在包括 ALTER（或 SET）COLUMN 子句以重命名该列的 ALTER COLUMN 语句中指定的新列名。
set_column_type	设置列的类型。
get_column_type	从 Column_Info 中检索本地列类型（如果该类型有效的话）。
set_for_bit_data	设置列的 FOR BIT DATA 标志。
get_for_bit_data	检索列的 FOR BIT DATA 标志（如果该标志有效的话）。
set_column_ID	设置列的标识（位置）。
get_column_ID	检索列的标识（位置）（如果该标识有效的话）。
set_org_length	设置列的最大长度。
get_org_length	从 Column_Info 中检索最大列长度（如果该值有效的话）。
set_org_scale	设置列的数字小数位。
get_org_scale	从 Column_Info 中检索数字小数位（如果该值有效的话）。
set_nulls	设置列的允许空值标志。
get_nulls	从 Column_Info 中检索允许空值标志（如果该值有效的话）。
set_avg_length	设置列的平均长度。
get_avg_length	从 Column_Info 中检索平均列长度（如果该值有效的话）。
set_high2key	设置列的次高值。

表 48. Column_Info 类的成员函数 (续)

成员函数	描述
get_high2key	从 Column_Info 中检索次高值（如果该值有效的话）。
set_low2key	设置列的次低值。
get_low2key	从 Column_Info 中检索次低值（如果该值有效的话）。
get_default	从 Column_Info 中检索缺省值（如果该值有效的话）。
set_colcard	设置列的基数。
get_colcard	从 Column_Info 中检索基数（如果该值有效的话）。
set_codepage1	设置列的代码页。
get_codepage1	从 Column_Info 中检索代码页（如果该值有效的话）。
set_codepage2	设置列的代码页。
get_codepage2	从 Column_Info 中检索代码页（如果该值有效的话）。
merge	将增量 Column_Info 对象合并到当前对象中。
copy	复制 Column_Info 对象。
add_option	将单值选项添加至目录信息。
drop_option	从 Catalog_Info 类中删除选项。
get_option	按名称检索目录选项。
get_first_option	检索指向选项链中第一个选项的指针。
get_next_option	检索选项链中的下一个选项。

Column_Info 构造函数

用途 构造缺省（空）Column_Info 对象。

语法

```
Column_Info ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

set_column_name 函数

用途 设置列的名称。

用法 不要使用此成员函数更改列名。

语法

```
sqlint32 set_column_name (sqluint8* a_column_name,
                          sqlint32  a_column_name_len)
```

输入自变量

表 49. set_column_name 成员函数的输入自变量

名称	数据类型	描述
a_column_name	sqluint8*	列名（不是以 null 结束的）。
a_column_name_len	sqlint32	列名的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_column_name 函数

用途 检索此列的名称。

语法

```
sqlint32 get_column_name (sqluint8** a_column_name)
```

输入自变量

无。

输出自变量

表 50. get_column_name 成员函数的输出自变量

名称	数据类型	描述
a_column_name	sqluint8**	指向以 null 结束的列名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示列名无效或未设置。

get_new_column_name 函数

用途 检索在包括 ALTER（或 SET）COLUMN 子句以重命名该列的 ALTER COLUMN 语句中指定的新列名。

语法

```
sqlint32 get_new_column_name (sqluint8** a_new_col_name)
```

输入自变量

无。

输出自变量

表 51. get_new_column_name 成员函数的输出自变量

名称	数据类型	描述
a_new_col_name	sqluint8**	指向新的以 null 结束的列名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置新列名。

set_column_type 函数

用途 设置列的类型。

语法

```
sqlint32 set_column_type (sqluint8* a_column_type,
                          sqlint32  a_column_type_len)
```

输入自变量

表 52. set_column_type 成员函数的输入自变量

名称	数据类型	描述
a_column_type	sqluint8*	列类型（不是以 null 结束的）。
a_column_type_len	sqlint32	列类型的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_column_type 函数

用途 从 Column_Info 中检索本地列类型（如果该类型有效的话）。

语法

```
sqlint32 get_column_type (sqluint8** a_column_type)
```

输入自变量

无。

输出自变量

表 53. get_column_type 成员函数的输出自变量

名称	数据类型	描述
a_column_type	sqluint8**	指向以 null 结束的列类型的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置类型。

set_for_bit_data 函数

用途 设置列的 FOR BIT DATA 标志。

语法

```
void set_for_bit_data (sqluint8 a_for_bit_data)
```

输入自变量

表 54. set_for_bit_data 成员函数的输入自变量

名称	数据类型	描述
a_for_bit_data	sqluint8	FOR BIT DATA 标志（标志必须为“Y”或“N”）。

输出自变量

无。

返回值 无。

get_for_bit_data 函数

用途 检索列的 FOR BIT DATA 标志（如果该标志有效的话）。

语法

```
sqlint32 get_for_bit_data (sqluint8* a_for_bit_data)
```

输入自变量

无。

输出自变量

表 55. get_for_bit_data 成员函数的输出自变量

名称	数据类型	描述
a_for_bit_data	sqluint8*	一个标志，它必须为“Y”或“N”。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置标志。

set_column_ID 函数

用途 设置列的标识（位置）。

用法 不要使用此成员函数更改昵称中列的顺序。

语法

```
sqlint32 set_column_ID (sqlint16 a_column_ID)
```

输入自变量

表 56. set_column_ID 成员函数的输入自变量

名称	数据类型	描述
a_column_ID	sqlint16	列标识（位置）。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_column_ID 函数

用途 检索列的标识（位置）（如果该标识有效的话）。

语法

```
sqlint32 get_column_ID (sqluint16* a_column_ID)
```

输入自变量

无。

输出自变量

表 57. get_column_ID 成员函数的输出自变量

名称	数据类型	描述
a_column_ID	sqluint16*	列标识。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置标识。

set_org_length 函数

用途 设置列的最大长度（以字节计）。

语法

```
void set_org_length (sqlint32 a_org_length)
```

输入自变量

表 58. set_org_length 成员函数的输入自变量

名称	数据类型	描述
a_org_length	sqlint32	列的最大长度。

输出自变量

无。

返回值 无。

get_org_length 函数

用途 从 Column_Info 中检索最大列长度（以字节计，如果该值有效的话）。

语法

```
sqlint32 get_org_length (sqlint32* a_org_length)
```

输入自变量

无。

输出自变量

表 59. get_org_length 成员函数的输出自变量

名称	数据类型	描述
a_org_length	sqlint32*	列的最大长度。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_org_scale 函数

用途 设置列的数字小数位。

语法

```
void set_org_scale (sqlint16 a_org_scale)
```

输入自变量

表 60. set_org_scale 成员函数的输入自变量

名称	数据类型	描述
a_org_scale	sqlint16	小数位。

输出自变量

无。

返回值 无。

get_org_scale 函数

用途 从 Column_Info 中检索数字小数位（如果该值有效的话）。

语法

```
sqlint32 get_org_scale (sqlint16* a_org_scale)
```

输入自变量

无。

输出自变量

表 61. get_org_scale 成员函数的输出自变量

名称	数据类型	描述
a_org_scale	sqlint16*	数字小数位。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_nulls 函数

用途 设置列的允许空值标志。

语法

```
void set_nulls (sqluint8 a_nulls)
```

输入自变量

表 62. set_nulls 成员函数的输入自变量

名称	数据类型	描述
a_nulls	sqluint8	允许空值标志。该标志必须为“Y”或“N”。

输出自变量

无。

返回值 无。

get_nulls 函数

用途 从 Column_Info 中检索允许空值标志（如果该值有效的话）。

语法

```
sqlint32 get_nulls (sqluint8* a_nulls)
```

输入自变量

无。

输出自变量

表 63. get_nulls 成员函数的输出自变量

名称	数据类型	描述
a_nulls	sqluint8*	列的允许空值标志。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_avg_length 函数

用途 设置列的平均长度（以字节计）。

用法 包装器在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间设置列的平均长度。当 DB2 优化器开发查询优化方案时，该优化器使用此平均长度信息。

语法

```
void set_avg_len (sqlint32 a_avg_len)
```

输入自变量

表 64. set_avg_length 成员函数的输入自变量

名称	数据类型	描述
a_avg_len	sqlint32	列的平均长度。

输出自变量

无。

返回值 无。

get_avg_length 函数

用途 从 Column_Info 中检索平均列长度（以字节计）（如果该值有效的话）。

语法

```
sqlint32 get_avg_length (sqlint32* a_avg_len)
```

输入自变量

无。

输出自变量

表 65. get_avg_length 成员函数的输出自变量

名称	数据类型	描述
a_avg_len	sqlint32*	列的平均长度。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_high2key 函数

用途 设置列的次高值。

用法 包装器可以在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间设置列的次高值。当 DB2 优化器开发查询优化方案时，该优化器可以使用此次高值或最高值。

语法

```
sqlint32 set_high2key (sqluint8* a_high2key,
                       sqlint32  a_high2key_len)
```

输入自变量

表 66. set_high2key 成员函数的输入自变量

名称	数据类型	描述
a_high2key	sqluint8*	次高字符串值（不是以 null 结束的）。
a_high2key_len	sqlint32	值的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_high2key 函数

用途 从 Column_Info 中检索次高值（如果该值有效的话）。

语法

```
sqlint32 get_high2key (sqluint8** a_high2key)
```

输入自变量

无。

输出自变量

表 67. get_high2key 成员函数的输出自变量

名称	数据类型	描述
a_high2key	sqluint8**	列的次高值。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_low2key 函数

用途 设置列的次低值。

用法 包装器可以在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间设置列的次低值。当 DB2 优化器开发查询优化方案时，该优化器可以使用此次低值或最低值。

语法

```
sqlint32 set_low2key (sqluint8* a_low2key,
                      sqlint32  a_low2key_len)
```

输入自变量

表 68. set_low2key 成员函数的输入自变量

名称	数据类型	描述
a_low2key	sqluint8*	次低字符串值（不是以 null 结束的）。
a_low2key_len	sqlint32	值的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_low2key 函数

用途 从 Column_Info 中检索次低值（如果该值有效的话）。

语法

```
sqlint32 get_low2key (sqluint8** a_low2key)
```

输入自变量

无。

输出自变量

表 69. get_low2key 成员函数的输出自变量

名称	数据类型	描述
a_low2key	sqluint8**	列的次低值。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

get_default 函数

用途 从 Column_Info 中检索缺省值（如果该值有效的话）。

语法

```
sqlint32 get_default (sqluint8** a_default)
```

输入自变量

无。

输出自变量

表 70. get_default 成员函数的输出自变量

名称	数据类型	描述
a_default	sqluint8**	列的缺省值。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_colcard 函数

用途 设置列的基数。

用法 包装器在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间设置列基数（如果已知的话）。DB2 优化器在生成优化性能方案时使用此信息。对于具有特异值（没有重复）的列，列基数必须与昵称基数相同。如果列基数大于昵称基数，则 DB2 优化器将生成错误。

语法

```
void set_colcard (sqlint64 a_colcard)
```

输入自变量

表 71. set_colcard 成员函数的输入自变量

名称	数据类型	描述
a_colcard	sqlint64	列的基数。

输出自变量

无。

返回值 无。

get_colcard 函数

用途 从 Column_Info 中检索基数（如果该值有效的话）。

语法

```
sqlint32 get_colcard (sqlint64* a_colcard)
```

输入自变量

无。

输出自变量

表 72. get_colcard 成员函数的输出自变量

名称	数据类型	描述
a_colcard	sqlint64*	列的基数。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_codepage1 函数

用途 设置列的代码页。

语法

```
void set_codepage1 (sqlint16 a_codepage1)
```

输入自变量

表 73. set_codepage1 成员函数的输入自变量

名称	数据类型	描述
a_codepage1	sqlint16	列的代码页。

输出自变量

无。

返回值 无。

get_codepage1 函数

用途 从 Column_Info 中检索代码页（如果该值有效的话）。

语法

```
sqlint32 get_codepage1 (sqlint16* a_codepage1)
```

输入自变量

无。

输出自变量

表 74. get_codepage1 成员函数的输出自变量

名称	数据类型	描述
a_codepage1	sqlint16*	列的代码页。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_codepage2 函数

用途 设置列的代码页。

语法

```
void set_codepage2 (sqlint16 a_codepage2)
```

输入自变量

表 75. set_codepage2 成员函数的输入自变量

名称	数据类型	描述
a_codepage2	sqlint16	列的代码页。

输出自变量

无。

返回值 无。

get_codepage2 函数

用途 从 Column_Info 中检索代码页（如果该值有效的话）。

语法

```
sqlint32 get_codepage2 (sqlint16* a_codepage2)
```

输入自变量

无。

输出自变量

表 76. get_codepage2 成员函数的输出自变量

名称	数据类型	描述
a_codepage2	sqlint16*	列的代码页。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

merge 函数

用途 将增量 Column_Info 对象合并到当前对象中。

语法

```
sqlint32 merge (Column_Info* a_delta_info)
```

输入自变量

表 77. merge 成员函数的输入自变量

名称	数据类型	描述
a_delta_info	Column_Info*	要合并的数据。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

copy 函数

用途 复制 Column_Info 对象。

语法

```
sqlint32 copy (Column_Info** a_new_column_info)
```

输入自变量

无。

输出自变量

表 78. copy 成员函数的输出自变量

名称	数据类型	描述
a_new_column_info	Column_Info**	指向复制对象的指针。

返回值 返回码。如果值为 0，则指示成功。

add_option 函数

定义位置

Catalog_Info

用途 将单值选项添加至目录信息。

用法 当在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间将包装器生成的选项添加至目录时，包装器将调用此成员函数。

语法

```
sqlint32 add_option (sqluint8*  a_opt_name,
                     sqlint32   a_opt_name_len,
                     sqluint8*  a_opt_value,
                     sqlint32   a_opt_value_len,
                     Catalog_Option::Action     a_act
                   = Catalog_Option::sqlqg_None,
                     char*      a_opt_type = "")
```

输入自变量

表 79. add_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束的）。
a_opt_value_len	sqlint32	选项值的长度。
a_act	Catalog_Option::Action	此选项的操作（ADD、SET、DROP 或无）。
a_opt_type	char*	在 SQLN1884 错误消息中使用的标记（如果这是重复选项的话）。使用在 sqlqg_misc.h 头文件中定义的 SQLQG_COLUMN_OPTION 常量。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

drop_option 函数

定义位置

Catalog_Info

用途 从 Column_Info 类中删除选项。此成员函数不会从目录中删除选项。在将带有 Catalog_Option::sqlqg_Drop 操作的选项添加至增量 Catalog_Info 对象时，将删除该选项。

语法

```
sqlint32 drop_option (Catalog_Option* a_option)
```

输入自变量

表 80. drop_option 成员函数的输入自变量

名称	数据类型	描述
a_option	Catalog_Option*	要删除的选项。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_option 函数

定义位置

Catalog_Info

用途 按名称检索目录选项。

用法 如果未找到该选项，则输出自变量 a_option 为空。

语法

```
sqlint32 get_option (sqluint8*        a_opt_name,
                     sqlint32         a_name_len,
                     Catalog_Option** a_option)
```

输入自变量

表 81. get_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。

输出自变量

表 82. get_option 成员函数的输出自变量

名称	数据类型	描述
a_option	Catalog_Option**	找到选项。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到该选项。

get_first_option 函数

定义位置

Catalog_Info

用途 检索指向选项链中第一个选项的指针。

语法

```
Catalog_Option* get_first_option ()
```

输入自变量

无。

输出自变量

无。

返回值 指向链中第一个选项的指针。如果链是空的，则该值为空。

get_next_option 函数

定义位置

Catalog_Info

用途 检索选项链中的下一个选项。

语法

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

输入自变量

表 83. get_next_option 成员函数的输入自变量

名称	数据类型	描述
a_current_option	Catalog_Option*	当前选项。

输出自变量

无。

返回值 指向链中下一个选项的指针。如果在结尾，则该值为空。

相关参考:

- 第 1 页的『C++ API 的目录类』

Nickname_Info 类（C++）

本主题描述 Nickname_Info 类并提供构造函数和成员函数的详细信息。

概述

Nickname_Info 类包括目录中的昵称定义并包括列定义。

Nickname_Info 类是 C++ API 的其中一个目录类。

用法 此类由 DB2 联合服务器实例化以包含 CREATE NICKNAME 或 ALTER NICKNAME 语句中的信息或包含 DB2 Information Integrator 目录中的信息。当在 CREATE NICKNAME 或 ALTER NICKNAME 语句操作期间添加信息时，此类由包装器实例化。

文件 sqlqg_catalog.h

数据成员

无。

构造函数和成员函数

下列各表描述 Nickname_Info 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 84. Nickname_Info 类的构造函数

构造函数	描述
Nickname_Info	构造缺省（空）Nickname_Info 对象。

表 85. Nickname_Info 类的成员函数

成员函数	描述
get_local_schema	检索昵称的本地模式名。
get_nickname	检索昵称的局部名。
get_server_name	检索昵称的服务器名。
set_card	设置昵称的基数。
get_card	检索昵称基数（如果该值有效的话）。
set_npages	设置昵称的 npages 统计信息。
get_npages	检索昵称 npages 统计信息（如果该值有效的话）。
set_fpages	设置昵称的 fpages 统计信息。
get_fpages	检索昵称 fpages 统计信息（如果该值有效的话）。
set_overflow	设置昵称的溢出统计信息。
get_overflow	检索昵称溢出统计信息（如果该值有效的话）。
insert_column	将列定义添加至昵称信息。
replace_column	在昵称信息中替换列定义。
get_number_columns	检索昵称的列的数目。
get_first_column	检索指向昵称的第一个 Column_Info 对象的指针。
get_next_column	检索指向下一个 Column_Info 对象的指针。
get_column	按（本地）名称检索列描述。
merge	将增量 Nickname_Info 对象合并到当前对象中。
copy	复制 Nickname_Info 对象。
add_option	将单值选项添加至目录信息。
drop_option	从 Catalog_Info 类中删除选项。
get_option	按名称检索目录选项。
get_first_option	检索指向选项链中第一个选项的指针。
get_next_option	检索选项链中的下一个选项。

Nickname_Info 构造函数

用途 构造缺省（空）Nickname_Info 对象。

语法

```
Nickname_Info ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

get_local_schema 函数

用途 检索昵称的本地模式名。

语法

```
sqlint32 get_local_schema (sqluint8** a_local_schema)
```

输入自变量

无。

输出自变量

表 86. get_local_schema 成员函数的输出自变量

名称	数据类型	描述
a_local_schema	sqluint8**	指向以 null 结束的本地模式名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

get_nickname 函数

用途 检索昵称的局部名。

语法

```
sqlint32 get_nickname (sqluint8** a_nickname)
```

输入自变量

无。

输出自变量

表 87. get_nickname 成员函数的输出自变量

名称	数据类型	描述
a_nickname	sqluint8**	指向以 null 结束的本地名称的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

get_server_name 函数

用途 检索昵称的服务器名。

语法

```
sqlint32 get_server_name (sqluint8** a_server_name)
```

输入自变量

无。

输出自变量

表 88. get_server_name 成员函数的输出自变量

名称	数据类型	描述
a_server_name	sqluint8**	指向以 null 结束的服务器名的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_card 函数

用途 设置昵称的基数。

用法 包装器可以在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间调用此成员函数以指定昵称的初始基数。可以使用 Column_Info 对象的 set_colcard 方法来设置个别列基数。对于具有唯一值的列，列基数必须等于昵称基数。列基数不能大于昵称基数。

语法

```
void set_card (sqlint64 a_card)
```

输入自变量

表 89. set_card 成员函数的输入自变量

名称	数据类型	描述
a_card	sqlint64	昵称的基数。

输出自变量

无。

返回值 无。

get_card 函数

用途 检索昵称基数（如果该值有效的话）。

语法

```
sqlint32 get_card (sqlint64* a_card)
```

输入自变量

无。

输出自变量

表 90. get_card 成员函数的输出自变量

名称	数据类型	描述
a_card	sqlint64*	昵称基数。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_npages 函数

用途 设置昵称的 npages 统计信息。

语法

```
void set_npages (sqlint32 a_npages)
```

输入自变量

表 91. set_npages 成员函数的输入自变量

名称	数据类型	描述
a_npages	sqlint32	昵称的 Npages 值。

输出自变量

无。

返回值 无。

get_npages 函数

用途 检索昵称 npages 统计信息（如果该值有效的话）。

语法

```
sqlint32 get_npages (sqlint32* a_npages)
```

输入自变量

无。

输出自变量

表 92. get_npages 成员函数的输出自变量

名称	数据类型	描述
a_npages	sqlint32*	昵称 npages 统计信息。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_fpages 函数

用途 设置昵称的 fpages 统计信息。

语法

```
void set_fpages (sqlint32 a_fpages)
```

输入自变量

表 93. set_fpages 成员函数的输入自变量

名称	数据类型	描述
a_fpages	sqlint32	昵称的 fpages 统计信息。

输出自变量

无。

返回值 无。

get_fpages 函数

用途 检索昵称 fpages 统计信息（如果该值有效的话）。

语法

```
sqlint32 get_fpages (sqlint32* a_fpages)
```

输入自变量

无。

输出自变量

表 94. get_fpages 成员函数的输出自变量

名称	数据类型	描述
a_fpages	sqlint32*	昵称 fpages 统计信息。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

set_overflow 函数

用途 设置昵称的溢出统计信息。

语法

```
void set_overflow (sqlint32 a_overflow)
```

输入自变量

表 95. set_overflow 成员函数的输入自变量

名称	数据类型	描述
a_overflow	sqlint32	昵称的溢出统计信息。

输出自变量

无。

返回值 无。

get_overflow 函数

用途 检索昵称溢出统计信息（如果该值有效的话）。

语法

```
sqlint32 get_overflow (sqlint32* a_overflow)
```

输入自变量

无。

输出自变量

表 96. get_overflow 成员函数的输出自变量

名称	数据类型	描述
a_overflow	sqlint32*	昵称溢出统计信息。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未设置值。

insert_column 函数

用途 将列定义添加至昵称信息。

用法 必须在堆上分配 Column_Info 对象（使用新的）。此例程控制指针。如果设置了 a_new_col_info 自变量的列标识，则将发生错误。

语法

```
sqlint32 insert_column (Column_Info* a_new_col_info)
```

输入自变量

表 97. insert_column 成员函数的输入自变量

名称	数据类型	描述
a_new_col_info	Column_Info*	要添加的 Column_Info 对象。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

replace_column 函数

用途 在昵称信息中替换列定义。如果两列定义使用相同的标识，则认为它们是相同的。

用法 如果未在 a_new_col_info 自变量中设置列标识，则将发生错误。

语法

```
sqlint32 replace_column (Column_Info* a_new_col_info)
```

输入自变量

表 98. replace_column 成员函数的输入自变量。

名称	数据类型	描述
a_new_col_info	Column_Info*	要替换的 Column_Info。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_number_columns 函数

用途 检索昵称的列的数目。

语法

```
sqlint16 get_number_columns ()
```

输入自变量

无。

输出自变量

无。

返回值 列的数目。

get_first_column 函数

用途 检索指向昵称的第一个 Column_Info 对象的指针。

用法 按标识顺序维护列。

语法

```
Column_Info* get_first_column ()
```

输入自变量

无。

输出自变量

无。

返回值 指向第一个 Column_Info 对象的指针。如果列表是空的，则该值为空。

get_next_column 函数

用途 检索指向下一个 Column_Info 对象的指针。

用法 按标识顺序维护列。

语法

```
Column_Info* get_next_column (Column_Info* a_cur_col_info)
```

输入自变量

表 99. get_next_column 成员函数的输入自变量

名称	数据类型	描述
a_cur_col_info	Column_Info*	当前列。

输出自变量

无。

返回值 指向下一个 Column_Info 的指针。如果在列表结尾，则该值为空。

get_column 函数

用途 按（本地）名称检索列描述。

语法

```
sqlint32 get_column (sqluint8*     a_col_name,
                     sqlint32      a_col_name_len,
                     Column_Info** a_col_info)
```

输入自变量

表 100. get_column 成员函数的输入自变量

名称	数据类型	描述
a_col_name	sqluint8*	列名（不是以 null 结束的）。
a_col_name_len	sqlint32	列名的长度。

输出自变量

表 101. get_column 成员函数的输出自变量

名称	数据类型	描述
a_col_info	Column_Info**	指向 Column_Info 对象的指针。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到列名。

merge 函数

用途 将增量 Nickname_Info 对象合并到当前对象中。

语法

```
sqlint32 merge (Nickname_Info* a_delta_info)
```

输入自变量

表 102. merge 成员函数的输入自变量

名称	数据类型	描述
a_delta_info	Nickname_Info*	要合并的数据。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

copy 函数

用途 复制 Nickname_Info 对象。

语法

```
sqlint32 copy (Nickname_Info** a_new_idx_info)
```

输入自变量

无。

输出自变量

表 103. copy 成员函数的输出自变量

名称	数据类型	描述
a_new_idx_info	Nickname_Info**	指向复制对象的指针。

返回值 返回码。如果值为 0，则指示成功。

add_option 函数

定义位置

Catalog_Info

用途 将单值选项添加至目录信息。

用法 当在 CREATE NICKNAME 或 ALTER NICKNAME 语句处理期间将包装器生成的选项添加至目录时，包装器可以调用此成员函数。

语法

```
sqlint32 add_option (sqluint8*  a_opt_name,
                     sqlint32   a_opt_name_len,
                     sqluint8*  a_opt_value,
                     sqlint32   a_opt_value_len,
                     Catalog_Option::Action     a_act
                   = Catalog_Option::sqlqg_None,
                     char*      a_opt_type = "")
```

输入自变量

表 104. add_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束的）。
a_opt_value_len	sqlint32	选项值的长度。
a_act	Catalog_Option::Action	此选项的操作（ADD、SET、DROP 或无）。
a_opt_type	char*	要在 SQLN1884 错误消息中使用的标记（如果这是重复选项的话）。使用在 sqlqg_misc.h 头文件中定义的 SQLQG_NICKNAME_OPTION 常量。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

drop_option 函数

定义位置

Catalog_Info

用途 从 Catalog_Info 类中删除选项。

用法 此成员函数不会从目录中删除选项。在将带有 Catalog_Option::sqlqg_Drop 操作的选项添加至增量 Catalog_Info 对象时，将从目录中删除该选项。

语法

```
sqlint32 drop_option (Catalog_Option* a_option)
```

输入自变量

表 105. drop_option 成员函数的输入自变量

名称	数据类型	描述
a_option	Catalog_Option*	要删除的选项。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_option 函数

定义位置

Catalog_Info

用途 按名称检索目录选项。

用法 如果未找到该选项，则输出自变量 `a_option` 为空。

语法

```
sqlint32 get_option (sqluint8*        a_opt_name,
                     sqlint32         a_name_len,
                     Catalog_Option** a_option)
```

输入自变量

表 106. get_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	选项名（不是以 null 结束的）。
a_name_len	sqlint32	选项名的长度。

输出自变量

表 107. get_option 成员函数的输出自变量

名称	数据类型	描述
a_option	Catalog_Option**	找到选项。

返回值 返回码。如果值为 0，则指示成功。SQLQG_NOVALUE 指示未找到该选项。

get_first_option 函数

定义位置

Catalog_Info

用途 检索指向选项链中第一个选项的指针。

语法

```
Catalog_Option* get_first_option ()
```

输入自变量

无。

输出自变量

无。

返回值 指向链中第一个选项的指针。如果链是空的，则该值为空。

get_next_option 函数

定义位置

Catalog_Info

用途 检索选项链中的下一个选项。

语法

```
Catalog_Option* get_next_option (Catalog_Option* a_current_option)
```

输入自变量

表 108. get_next_option 成员函数的输入自变量

名称	数据类型	描述
a_current_option	Catalog_Option*	当前选项。

输出自变量

无。

返回值 指向链中下一个选项的指针。如果在结尾，则该值为空。

相关参考:

- 第 1 页的『C++ API 的目录类』

C++ API 的包装器类

下表描述了 C++ API 的每个包装器类。

表 109. 包装器类

类名	描述
Unfenced_Generic_Wrapper	此类表示不受防护（可信）的进程空间上的包装器，并且该类负责验证目录数据。
Fenced_Generic_Wrapper	此类表示受防护的进程空间上的包装器。

相关参考:

- 第 50 页的『Unfenced_Generic_Wrapper 类（C++）』
- 第 58 页的『Fenced_Generic_Wrapper 类（C++）』

Unfenced_Generic_Wrapper 类（C++）

本主题描述 Unfenced_Generic_Wrapper 类并提供有关构造函数、析构函数和成员函数的详细信息。

概述

Unfenced_Generic_Wrapper 类表示不受防护（可信）的进程空间上的包装器，并且验证目录数据。

Unfenced_Generic_Wrapper 类是 C++ API 的其中一个包装器类。

用法 包装器必须实现 Unfenced_Generic_Wrapper 类的子类。此类是通过特定于包装器的 UnfencedWrapper_Hook 函数来实例化的。

文件 sqlqg_unfenced_generic_wrapper.h

数据成员

下表列示了可以与 Unfenced_Generic_Wrapper 类配合使用的数据成员。

表 110. Unfenced_Generic_Wrapper 类的数据成员

名称	数据类型	描述
m_info	Wrapper_Info*	包含包装器的目录信息。
m_name	sqluint8*	指向包含包装器名称的以 null 结束的字符串的指针。
m_corelib	sqluint8*	指向包含包装器库名称的以 null 结束的字符串的指针。此名称是在 CREATE WRAPPER 语句中使用的包装器库。此名称并不对应于在处理语句期间所使用的库。
m_version	sqlint32	包装器代码的版本。

构造函数、析构函数和成员函数

下列各表描述了 Unfenced_Generic_Wrapper 类的构造函数、析构函数和成员函数。在这些表的后面更详细地描述了构造函数、析构函数和成员函数。

表 111. Unfenced_Generic_Wrapper 类的构造函数

构造函数	描述
Unfenced_Generic_Wrapper	构造 Unfenced_Generic_Wrapper 类的实例。

表 112. Unfenced_Generic_Wrapper 类的析构函数

析构函数	描述
~Unfenced_Generic_Wrapper	Unfenced_Generic_Wrapper 类的析构函数。

表 113. Unfenced_Generic_Wrapper 类的成员函数

成员函数	描述
verify_my_register_wrapper_info	当提交 CREATE WRAPPER 语句时验证目录信息。
verify_my_alter_wrapper_info	当提交 ALTER WRAPPER 语句时验证目录信息。
get_name	返回指向包含包装器名称的以 null 结束的字符串的指针。

表 113. Unfenced_Generic_Wrapper 类的成员函数 (续)

成员函数	描述
get_corelib	返回指向包含包装器库名称的以 null 结束的字符串的指针。
get_version	返回包装器代码的版本。
get_info	返回指向包装器目录信息对象的指针。
initialize_my_wrapper	从目录信息对象初始化包装器对象状态。
create_server	实例化包装器的 Server 的适当子类。
is_creating_threads	指示包装器创建进程线程。
is_ready_for_fenced_mode	指示包装器可以用设防方式运行。
is_safe_in_thread_mode	指示包装器是线程安全的。

Unfenced_Generic_Wrapper 构造函数

用途 构造 Unfenced_Generic_Wrapper 的实例。

语法

```
Unfenced_Generic_Wrapper (sqlint32* a_rc,
                          sqlint32  a_wrapper_version = 0)
```

输入自变量

表 114. Unfenced_Generic_Wrapper 构造函数的输入自变量

名称	数据类型	描述
a_wrapper_version	sqlint32	包装器代码的版本。

输出自变量

表 115. Unfenced_Generic_Wrapper 构造函数的输出自变量。

名称	数据类型	描述
a_rc	sqlint32*	指向返回值的指针。该值必须为 0 才表示成功。

返回值 无。

~Unfenced_Generic_Wrapper 析构函数

用途 Unfenced_Generic_Wrapper 类的析构函数。

语法

```
~Unfenced_Generic_Wrapper ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

verify_my_register_wrapper_info 成员函数

用途 当提交 CREATE WRAPPER 语句时验证目录信息。

用法 如果特定于包装器的包装器选项受支持，则此成员函数可以由 Unfenced_Generic_Wrapper 的特定于包装器的子类中的包装器实现。

包装器将检查在它自己分配一个增量对象之前是否已经分配了一个增量对象。

语法

```
virtual sqlint32 verify_my_register_wrapper_info
                 (Wrapper_Info*  a_wrapper_info,
                  Wrapper_Info** a_delta_info)
```

输入自变量

表 116. verify_my_register_wrapper_info 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_info	Wrapper_Info*	来自 CREATE WRAPPER 语句的信息。

输出自变量

表 117. verify_my_register_wrapper_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Wrapper_Info**	由 verify_my 添加的信息。

返回值 返回码。如果值为 0，则指示成功。

verify_my_alter_wrapper_info 成员函数

用途 当提交 ALTER WRAPPER 语句时验证目录信息。

用法 如果特定于包装器的包装器选项受支持，则此成员函数可以由 Unfenced_Generic_Wrapper 的特定于包装器的子类中的包装器实现。

包装器将检查在它自己分配一个增量对象之前是否已经分配了一个增量对象。

语法

```
virtual sqlint32 verify_my_alter_wrapper_info
                 (Wrapper_Info*  a_wrapper_info,
                  Wrapper_Info** a_delta_info)
```

输入自变量

表 118. verify_my_alter_wrapper_info 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_info	Wrapper_Info*	来自 ALTER WRAPPER 语句的信息。

输出自变量

表 119. verify_my_alter_wrapper_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Wrapper_Info**	由 verify_my 添加的信息。

返回值 返回码。如果值为 0，则指示成功。

get_name 成员函数

定义位置

包装器

用途 返回指向包含包装器名称的以 null 结束的字符串的指针。

语法

```
sqluint8* get_name()
```

输入自变量

无。

输出自变量

无。

返回值 指向字符串的指针。

get_corelib 成员函数

定义位置

包装器

用途 返回指向包含包装器库名称的以 null 结束的字符串的指针。此名称是在 CREATE WRAPPER 语句中使用的包装器库。此名称并不对应于在处理语句期间所使用的库。

语法

```
sqluint8* get_corelib()
```

输入自变量

无。

输出自变量

无。

返回值 指向字符串的指针。

get_version 成员函数

定义位置

包装器

用途 返回包装器代码的版本。

语法

```
sqlint32 get_version()
```

输入自变量

无。

输出自变量

无。

返回值 包装器版本。

get_info 成员函数

定义位置

包装器

用途 返回指向包装器目录信息对象的指针。

语法

```
Wrapper_Info* get_info()
```

输入自变量

无。

输出自变量

无。

返回值 指向目录信息对象的指针。

initialize_my_wrapper 成员函数

定义位置

包装器

用途 从目录信息对象初始化包装器对象状态。缺省版本不执行任何操作。

用法 如果特定于包装器的包装器选项受支持，则此成员函数可以在 Unfenced_Generic_Wrapper 的特定于包装器的子类中实现。

语法

```
virtual sqlint32 initialize_my_wrapper (Wrapper_Info* a_wrapper_info)
```

输入自变量

表 120. initialize_my_wrapper 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_info	Wrapper_Info*	目录信息对象。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_server 成员函数

定义位置

包装器

用途 实例化包装器的 Server 的适当子类。

用法 此成员函数必须由 Unfenced_Generic_Wrapper 的特定于包装器的子类中的包装器实现。

语法

```
virtual Server* create_server (sqluint8* a_server_name,
                               sqlint32* a_rc)
```

输入自变量

表 121. create_server 成员函数的输入自变量

名称	数据类型	描述
a_server_name	sqluint8*	以 null 结束的数据源服务器名称。

输出自变量

表 122. create_server 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向新创建的服务器对象的指针或 null。

is_creating_threads 成员函数

定义位置

包装器

用途 向联合服务器指明包装器创建进程线程。

用法 如果包装器创建线程，则包装器重设此函数。

语法

```
virtual bool is_creating_threads (void) const
```

输入自变量

无。

输出自变量

无。

返回值 如果包装器创建线程，则返回值 true。缺省实现返回值 false。

is_ready_for_fenced_mode 成员函数

定义位置

Unfenced_Generic_Wrapper 类

用途 向联合服务器指明包装器可以用设防方式运行。

用法 如果包装器无法以设防方式运行，则包装器必须重设此函数。

语法

```
virtual sqlint16 is_ready_for_fenced_mode (void)
```

输入自变量

无。

输出自变量

无。

返回值 缺省情况下返回值 true，这表示包装器以设防方式运行。如果包装器无法以设防方式运行，则返回值 false。

is_safe_in_thread_mode 成员函数

定义位置

包装器

用途 指示包装器是线程安全的。

用法 如果包装器不是线程安全的，则包装器必须重设此函数。

语法

```
virtual bool is_safe_in_thread_mode (void) const
```

输入自变量

无。

输出自变量

无。

返回值 缺省实现返回值 true。如果包装器不是线程安全的，则返回值 false。

相关联的函数

Unfenced_Generic_Wrapper 类并不使用 UnfencedWrapper_Hook 函数，但是该函数直接与此类相关联。

UnfencedWrapper_Hook 函数

用途 一种钩子函数，用来启用联合服务器以实例化 Unfenced_Generic_Wrapper 类的特定于包装器的子类。此函数必须由包装器实现，并且必须从不受防护的包装器模块中导出。

语法

```
extern "C" UnfencedWraper* UnfencedWrapper_Hook()
```

输入自变量

无。

输出自变量

无。

返回值 如果出现错误，则将返回 Unfenced_Generic_Wrapper 的已实例化的子类或者是 null。

相关任务:

- 『包装器类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 50 页的『C++ API 的包装器类』

Fenced_Generic_Wrapper 类（C++）

本主题描述 Fenced_Generic_Wrapper 类并提供有关构造函数、析构函数和成员函数的详细信息。

概述

Fenced_Generic_Wrapper 类表示受防护的进程空间上的包装器。

Fenced_Generic_Wrapper 类是 C++ API 的其中一个包装器类。

用法 包装器必须实现 Fenced_Generic_Wrapper 的子类。此类是由特定于包装器的 FencedWrapper_Hook 函数实例化的。

文件 sqlqg_fenced_generic_wrapper.h

数据成员

下表列示了可以与 Fenced_Generic_Wrapper 类配合使用的数据成员。

表 123. Fenced_Generic_Wrapper 类的数据成员

名称	数据类型	描述
m_info	Wrapper_Info*	包含包装器的目录信息。
m_name	sqluint8*	指向包含包装器名称的字符串的指针。
m_corelib	sqluint8*	指向包含包装器库名称的字符串的指针。
m_version	sqlint32	包装器代码的版本。

构造函数、析构函数和成员函数

下列各表描述了 Fenced_Generic_Wrapper 类的构造函数、析构函数和成员函数。在这些表的后面更详细地描述了构造函数、析构函数和成员函数。

表 124. Fenced_Generic_Wrapper 类的构造函数

构造函数	描述
FencedGeneric_Wrapper	构造 Fenced_Generic_Wrapper 类的实例。

表 125. Fenced_Generic_Wrapper 类的析构函数

析构函数	描述
~FencedGeneric_Wrapper	Fenced_Generic_Wrapper 类的析构函数。

表 126. Fenced_Generic_Wrapper 类的成员函数

成员函数	描述
get_name	返回指向包含包装器名称的以 null 结束的字符串的指针。
get_corelib	返回指向包含包装器库名称的以 null 结束的字符串的指针。
get_version	返回包装器代码的版本。
get_info	返回指向包装器目录信息对象的指针。
initialize_my_wrapper	从目录信息对象初始化包装器对象状态。
create_server	实例化包装器的 Server 的适当子类。
is_creating_threads	指示包装器创建进程线程。
is_ready_for_fenced_mode	指示包装器可以用设防方式运行。
is_safe_in_thread_mode	指示包装器是线程安全的。

FencedGeneric_Wrapper 构造函数

用途 构造 Fenced_Generic_Wrapper 类的实例。

语法

```
FencedGeneric_Wrapper (sqlint32* a_rc,
                       sqlint32  a_wrapper_version = 0)
```

输入自变量

表 127. FencedGeneric_Wrapper 构造函数的输入自变量

名称	数据类型	描述
a_wrapper_version	sqlint32	包装器代码的版本。

输出自变量

表 128. FencedGeneric_Wrapper 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针。此值必须为 0 才表示成功。

返回值 无。

~FencedGeneric_Wrapper 析构函数

用途 Fenced_Generic_Wrapper 类的析构函数。

语法

```
~FencedGeneric_Wrapper ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

get_name 成员函数

定义位置

包装器

用途 返回指向包含包装器名称的以 null 结束的字符串的指针。

语法

```
sqluint8* get_name()
```

输入自变量

无。

输出自变量

无。

返回值 指向字符串的指针。

get_corelib 成员函数

定义位置

包装器

用途 返回指向包含包装器库名称的以 null 结束的字符串的指针。

语法

```
sqluint8* get_corelib()
```

输入自变量

无。

输出自变量

无。

返回值 指向字符串的指针。

get_version 成员函数

定义位置

包装器

用途 返回包装器代码的版本。

语法

```
sqlint32 get_version()
```

输入自变量

无。

输出自变量

无。

返回值 包装器版本。

get_info 成员函数

定义位置

包装器

用途 返回指向包装器目录信息对象的指针。

语法

```
Wrapper_Info* get_info()
```

输入自变量

无。

输出自变量

无。

返回值 指向目录信息对象的指针。

initialize_my_wrapper 成员函数

定义位置

包装器

用途 从目录信息对象初始化包装器对象状态。缺省版本不执行任何操作。

用法 如果特定于包装器的包装器选项受支持，则包装器可以实现 Fenced_Generic_Wrapper 的特定于包装器的子类。

语法

```
virtual sqlint32 initialize_my_wrapper (Wrapper_Info* a_wrapper_info)
```

输入自变量

表 129. initialize_my_wrapper 成员函数的输入自变量

名称	数据类型	描述
a_wrapper_info	Wrapper_Info*	目录信息对象。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_server 成员函数

定义位置

包装器

用途 实例化包装器的 Server 的适当子类。

用法 在 Fenced_Generic_Wrapper 的特定于包装器的子类中，包装器必须覆盖此成员函数

语法

```
virtual Server* create_server (sqluint8* a_server_name,
                               sqlint32* a_rc)
```

输入自变量

表 130. create_server 成员函数的输入自变量

名称	数据类型	描述
a_server_name	sqluint8*	以 null 结束的数据源服务器名称。

输出自变量

表 131. create_server 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向新创建的服务器对象的指针或 null。

is_creating_threads 成员函数

定义位置

包装器

用途 向联合服务器指明包装器创建进程线程。

用法 如果包装器创建线程，则包装器重设此函数。

语法

```
virtual bool is_creating_threads (void) const
```

输入自变量

无。

输出自变量

无。

返回值 如果包装器创建线程，则返回值 true。缺省实现返回值 false。

is_ready_for_fenced_mode 成员函数

定义位置

Fenced_Generic_Wrapper 类

用途 向联合服务器指明包装器可以用设防方式运行。

用法 如果包装器无法以设防方式运行，则包装器必须重设此函数。

语法

```
virtual sqlint16 is_ready_for_fenced_mode (void)
```

输入自变量

无。

输出自变量

无。

返回值 缺省情况下返回值 true，这表示包装器以设防方式运行。如果包装器无法以设防方式运行，则返回值 false。

is_safe_in_thread_mode 成员函数

定义位置

包装器

用途 指示包装器是线程安全的。

用法 如果包装器不是线程安全的，则包装器必须重设此函数。

语法

```
virtual bool is_safe_in_thread_mode (void) const
```

输入自变量

无。

输出自变量

无。

返回值 缺省实现返回值 true。如果包装器不是线程安全的，则返回值 false。

相关联的函数

Fenced_Generic_Wrapper 类并不使用 FencedWrapper_Hook 函数，但是该函数直接与此类相关联。

FencedWrapper_Hook 函数

用途 一种钩子函数，用来启用联合服务器以实例化 Fenced_Generic_Wrapper 类的特定于包装器的子类。此函数必须由包装器实现，并且必须从受防护的包装器模块中导出。

语法

```
extern "C" FencedWraper* FencedWrapper_Hook()
```

输入自变量

无。

输出自变量

无。

返回值 如果出现错误，则将返回 Fenced_Generic_Wrapper 的已实例化的子类或者是 null。

相关任务:

- 『包装器类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 50 页的『C++ API 的包装器类』

C++ API 的服务器类

下表描述了 C++ API 的每个服务器类。

表 132. 服务器类

类名	描述
Unfenced_Generic_Server	Unfenced_Generic_Server 类是 Unfenced_Server 类的一个子类，并且是在不受防护（可信）的进程空间中运行的所有服务器功能的抽象基本类。Unfenced_Generic_Server 类将负责验证 CREATE SERVER 和 ALTER SERVER 语句中的目录信息。
Fenced_Generic_Server	Fenced_Generic_Server 类是 Server 类的一个子类，并且是在受防护（不可信）的进程空间中运行的所有服务器功能的抽象基本类。Fenced_Generic_Server 类将负责创建远程连接和昵称。

相关参考:

- 第 64 页的『Unfenced_Generic_Server 类（C++）』
- 第 72 页的『Fenced_Generic_Server 类（C++）』

Unfenced_Generic_Server 类（C++）

本主题描述 Unfenced_Generic_Server 类并提供有关构造函数和成员函数的详细信息。

概述

Unfenced_Generic_Server 类是 Unfenced_Server 类的一个子类，并且是在不受防护（可信）的进程空间中运行的所有服务器功能的抽象基本类。此类将负责验证 CREATE SERVER 和 ALTER SERVER 语句中的目录信息。

Unfenced_Generic_Server 类是 C++ API 的其中一个服务器类。

用法 包装器必须实现 Unfenced_Generic_Server 的子类。Unfenced_Generic_Server 类是通过 Unfenced_Generic_Wrapper 类的特定于包装器的子类的 create_server() 方法中的包装器来实例化的。

文件 sqlqg_unfenced_generic_server.h

数据成员

下表列示了可以与 Unfenced_Generic_Server 类配合使用的数据成员。

表 133. Unfenced_Generic_Server 类的数据成员

名称	数据类型	描述
info	Server_Info*	服务器的目录信息。
name	sqluint8*	包含服务器名称的以 null 结束的字符串。
kind	server_kind	服务器的类型，必须是 Server::generic_kind。
wrapper	Wrapper*	指向拥有此服务器的相关联包装器对象的指针。

构造函数和成员函数

下列各表描述了 Unfenced_Generic_Server 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 134. Unfenced_Generic_Server 类的构造函数

构造函数	描述
Unfenced_Generic_server	构造 Unfenced_Generic_Server 类的实例

表 135. Unfenced_Generic_Server 类的成员函数

成员函数	描述
create_remote_user	实例化 Remote_User 的适当子类。
plan_request	分析提出的方案，并确定可以将哪些部分（如果有的话）下推至数据源。
get_selectivity	计算谓词列表的选择性。
create_reply	实例化应答对象。
get_name	返回指向以 null 结束的服务器名称的指针。
get_type	从目录信息中检索在 CREATE SERVER 语句中指定的以 null 结束的服务器类型。
get_version	检索在 CREATE SERVER 语句中指定的以 null 结束的服务器版本。
get_info	返回指向存储的目录信息对象的指针。
initialize_my_server	从有效的目录信息中初始化服务器实例。
create_nickname	实例化此服务器的昵称的适当子类。
is_reserved_server_option	标识受 DB2 支持但是被包装器忽略了的选项。
verify_my_register_server_info	验证 CREATE SERVER 语句中提供的信息。
verify_my_alter_server_info	验证 ALTER SERVER 语句中提供的信息。

Unfenced_Generic_server 构造函数

用途 构造 Unfenced_Generic_Server 类的实例

语法

```
Unfenced_Generic_server (sqluint8*        a_server_name,
                         UnfencedWrapper* a_wrapper,
                         sqlint32*        a_rc)
```

输入自变量

表 136. Unfenced_Generic_server 构造函数的输入自变量

名称	数据类型	描述
a_server_name	sqluint8*	包含服务器名称的以 null 结束的字符串。
a_wrapper	UnfencedWrapper*	指向拥有此服务器对象的包装器对象的指针。

输出自变量

表 137. Unfenced_Generic_server 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

create_remote_user 函数

用途 实例化 Remote_User 的适当子类。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现。如果实现了 Unfenced_Generic_User 类的特定于包装器的子类，则必须实现此成员函数。

语法

```
virtual Remote_User* create_remote_user (sqluint8* a_user_name,
                                         sqlint32* a_rc)
```

输入自变量

表 138. create_remote_user 成员函数的输入自变量

名称	数据类型	描述
a_user_name	sqluint8*	包含用户名称的以 null 结束的字符串。

输出自变量

表 139. create_remote_user 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向 Remote_User 对象的指针。

plan_request 函数

用途 分析提出的方案，并确定可以将哪些部分（如果有的话）下推至数据源。

用法 此成员函数必须由特定于包装器的不受防护的服务器子类中的包装器实现。

语法

```
virtual sqlint32 plan_request (Request* a_req,
                               Reply**  a_rpl) = 0
```

输入自变量

表 140. plan_request 成员函数的输入自变量

名称	数据类型	描述
a_req	Request*	指向用来描述提出的方案的请求对象的指针。

输出自变量

表 141. plan_request 成员函数的输出自变量

名称	数据类型	描述
a_rpl	Reply**	指向由 plan_request 构造的应答对象链的指针。

返回值 返回码。如果值为 0，则指示成功。

get_selectivity 函数

用途 计算谓词列表的选择性。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现。缺省版本使用内置的 DB2 Information Integrator 公式来计算选择性。包装器使用这些公式和谓词表达式之间的已知相关性。

语法

```
virtual sqlint32 get_selectivity (Predicate_List* a_pl,
                                  float*          a_selectivity)
```

输入自变量

表 142. get_selectivity 成员函数的输入自变量

名称	数据类型	描述
a_pl	Predicate_List*	谓词列表。

输出自变量

表 143. get_selectivity 成员函数的输出自变量

名称	数据类型	描述
a_selectivity	float*	用 0.0 到 1.0 之间的值来表示的选择性。

返回值 返回码。如果值为 0，则指示成功。

create_reply 函数

用途 实例化应答对象。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现，如果使用特定于包装器的 Reply 子类，则必须实现此成员函数。如果包装器通过为应答创建子类来实现它自己的成本模型，则此方法将被覆盖。

语法

```
virtual sqlint32 create_reply (Request* a_req,
                               Reply**  a_rpl)
```

输入自变量

表 144. create_reply 成员函数的输入自变量

名称	数据类型	描述
a_req	Request*	指向请求对象的指针。

输出自变量

表 145. create_reply 成员函数的输出自变量

名称	数据类型	描述
a_rpl	Reply**	指向应答对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_name 成员函数

定义位置

服务器

用途 返回指向以 null 结束的服务器名称的指针。

语法

```
sqluint8* get_name()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器名称的指针。

get_type 函数

定义位置

服务器

用途 从目录信息中检索在 CREATE SERVER 语句中指定的以 null 结束的服务器类型。

用法 只有在初始化服务器对象之后，此方法才有效。如果在目录中未指定服务器类型，则此方法将记录一个错误。但是，不会生成用户错误。

语法

```
sqluint8* get_type()
```

输入自变量

无。

输出自变量
无。

返回值 指向服务器类型的指针。

get_version 成员函数

定义位置
服务器

用途 检索在 CREATE SERVER 语句中指定的以 null 结束的服务器版本。

用法 只有在初始化服务器对象之后，此方法才有效。如果在目录中未指定服务器类型，则此方法将记录一个错误。但是，不会生成用户错误。

语法

```
sqluint8* get_version()
```

输入自变量
无。

输出自变量
无。

返回值 指向服务器版本的指针。

get_info 成员函数

定义位置
服务器

用途 返回指向存储的目录信息对象的指针。

用法 只有在初始化服务器之后，此方法才有效。

语法

```
Server_Info* get_info()
```

输入自变量
无。

输出自变量
无。

返回值 指向目录信息对象的指针。

initialize_my_server 函数

定义位置
服务器

用途 从有效的目录信息中初始化服务器实例。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现，如果特定于包装器的服务器选项受支持，则必须实现此成员函数。

语法

```
virtual sqlint32 initialize_my_server (Server_Info* a_server_info)
```

输入自变量

表 146. initialize_my_server 成员函数的输入自变量

名称	数据类型	描述
a_server_info	Server_Info*	已验证的目录信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_nickname 函数

定义位置

服务器

用法 此成员函数必须由特定于包装器的不受防护的服务器子类中的包装器实现。

用途 实例化此服务器的昵称的适当子类。

语法

```
virtual Nickname* create_nickname (sqluint8* a_schema_name,
                                   sqluint8* a_nickname_name,
                                   sqlint32* a_rc)
```

输入自变量

表 147. create_nickname 成员函数的输入自变量

名称	数据类型	描述
a_schema_name	sqluint8*	包含昵称的本地（DB2）模式名称的以 Null 结束的字符串。
a_nickname_name	sqluint8*	包含本地（DB2）昵称名称的以 Null 结束的字符串。

输出自变量

表 148. create_nickname 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向已实例化的昵称对象的指针。

is_reserved_server_option 函数

定义位置

不受防护的服务器

用途 标识受 DB2 支持但是被包装器忽略了的选项。

语法

```
virtual sqlint32 is_reserved_server_option (sqluint8* a_op_name)
```

输入自变量

表 149. is_reserved_server_option 成员函数的输入自变量

名称	数据类型	描述
a_op_name	sqluint8*	包含选项名称的以 null 结束的字符串。

输出自变量

无。

返回值 如果值为 0，则指示选项不是由 DB2 定义的服务器选项。如果值不为 0，则指示选项是由 DB2 定义的服务器选项。

verify_my_register_server_info 函数

定义位置

不受防护的服务器

用途 验证 CREATE SERVER 语句中提供的信息。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现，如果特定于包装器的服务器选项受支持，则必须实现此成员函数。包装器在分配它自己的 a_delta_info 对象之前必须检查是否已经分配了一个 a_delta_info 对象。

语法

```
virtual sqlint32 verify_my_register_server_info
        (Server_Info*  a_server_info,
         Server_Info** a_delta_info)
```

输入自变量

表 150. verify_my_register_server_info 成员函数的输入自变量

名称	数据类型	描述
a_server_info	Server_Info*	来自 CREATE SERVER 语句的信息。

输出自变量

表 151. verify_my_register_server_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Server_Info**	由包装器提供的附加信息。

返回值 返回码。如果值为 0，则指示成功。

verify_my_alter_server_info 函数

定义位置

不受防护的服务器

用途 验证 ALTER SERVER 语句中提供的信息。

用法 此成员函数可以由特定于包装器的不受防护的服务器子类中的包装器实现，如果特定于包装器的服务器选项受支持，则必须实现此成员函数。包装器在分配它自己的 a_delta_info 对象之前必须已经分配了一个 a_delta_info 对象。

语法

```
virtual sqlint32 verify_my_alter_server_info
        (Server_Info*  a_server_info,
         Server_Info** a_delta_info)
```

输入自变量

表 152. verify_my_alter_server_info 成员函数的输入自变量

名称	数据类型	描述
a_server_info	Server_Info*	来自 ALTER SERVER 语句的信息。

输出自变量

表 153. verify_my_alter_server_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Server_Info**	由包装器提供的附加信息。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『服务器类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 64 页的『C++ API 的服务器类』

Fenced_Generic_Server 类（C++）

本主题描述 Fenced_Generic_Server 类并提供有关构造函数和成员函数的详细信息。

概述

Fenced_Generic_Server 类是 Server 类的一个子类，并且是在受防护（不可信）的进程空间中运行的所有服务器功能的抽象基本类。此类将负责创建远程连接和昵称。

Fenced_Generic_Server 类是 C++ API 的其中一个服务器类。

用法 包装器必须实现 Fenced_Generic_Server 的子类。此类是通过 Fenced_Generic_Wrapper 的特定于包装器的子类的 create_server() 方法中的包装器来实例化的。

文件 sqlqg_fenced_genserver.h

数据成员

下表列示了可以与 Fenced_Generic_Server 类配合使用的数据成员。

表 154. Fenced_Generic_Server 类的数据成员

名称	数据类型	描述
info	Server_Info*	服务器的目录信息。
name	sqluint8*	包含服务器名称的以 null 结束的字符串。
kind	server_kind	服务器的类型，必须是 Server::generic_kind。
wrapper	Wrapper*	指向拥有此服务器的相关联包装器对象的指针。

构造函数和成员函数

下列各表描述了 Fenced_Generic_Server 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 155. Fenced_Generic_Server 类的构造函数

构造函数	描述
Fenced_Generic_Server	此类的构造函数。

表 156. Fenced_Generic_Server 类的成员函数

成员函数	描述
create_remote_connection	实例化 Remote_Connection 的适当子类。
get_name	返回指向以 null 结束的服务器名称的指针。
get_type	从目录信息中检索在 CREATE SERVER 语句中指定的以 null 结束的服务器类型。
get_version	检索在 CREATE SERVER 语句中指定的以 null 结束的服务器版本。
get_info	返回指向存储的目录信息对象的指针。
initialize_my_server	从有效的目录信息中初始化服务器实例。
create_remote_user	实例化此包装器的 Remote_User 的适当子类。
create_nickname	实例化此服务器的昵称的适当子类。

Fenced_Generic_Server 构造函数

用途 此类的构造函数。

语法

```
Fenced_Generic_Server (sqluint8* a_server_name,
                       FencedWrapper* wrapper,
                       sqlint32* rc)
```

输入自变量

表 157. Fenced_Generic_Server 构造函数的输入自变量

名称	数据类型	描述
a_server_name	sqluint8*	服务器的名称。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_remote_connection 函数

定义位置

受防护的服务器

用途 实例化 Remote_Connection 的适当子类。

用法 此成员函数必须由特定于包装器的受防护的服务器子类中的包装器实现。

语法

```
virtual sqlint32 create_remote_connection
                 (FencedRemote_User*  a_user,
                  Remote_connection** a_connection)
```

输入自变量

表 158. create_remote_connection 成员函数的输入自变量

名称	数据类型	描述
a_user	FencedRemote_User*	指向表示用于连接的用户的对象的指针。

输出自变量

表 159. create_remote_connection 成员函数的输出自变量

名称	数据类型	描述
a_connection	Remote_Connection**	指向远程连接对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_name 成员函数

定义位置

服务器

用途 返回指向以 null 结束的服务器名称的指针。

语法

```
sqluint8* get_name()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器名称的指针。

get_type 函数

定义位置

服务器

用途 从目录信息中检索在 CREATE SERVER 语句中指定的以 null 结束的服务器类型。

用法 只有在初始化服务器对象之后（即，运行 initialize_server 之后），此方法才有效。如果在目录中未指定服务器类型，则调用此方法时将记录一个错误。但是，不会生成用户错误。

语法

```
sqluint8* get_type()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器类型的指针。

get_version 成员函数

定义位置

服务器

用途 检索在 CREATE SERVER 语句中指定的以 null 结束的服务器版本。

用法 只有在初始化服务器对象之后（即，运行 initialize_server 之后），此方法才有效。如果在目录中未指定服务器类型，则调用此方法时将记录一个错误。但是，不会生成用户错误。

语法

```
sqluint8* get_version()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器版本的指针。

get_info 成员函数

定义位置

服务器

用途 返回指向存储的目录信息对象的指针。

用法 只有在初始化服务器之后，此方法才有效。

语法

```
Server_Info* get_info()
```

输入自变量

无。

输出自变量

无。

返回值 指向目录信息对象的指针。

initialize_my_server 函数

定义位置

服务器

用途 从有效的目录信息中初始化服务器实例。

用法 此成员函数可以由特定于包装器的受防护的服务器子类中的包装器实现。如果特定于包装器的服务器选项受支持，则必须实现此成员函数。

语法

```
virtual sqlint32 initialize_my_server (Server_Info* a_server_info)
```

输入自变量

表 160. initialize_my_server 成员函数的输入自变量

名称	数据类型	描述
a_server_info	Server_Info*	已验证的目录信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_remote_user 函数

定义位置

服务器

用途 实例化此包装器的 Remote_User 的适当子类。

用法 此成员函数可以由特定于包装器的受防护的服务器子类中的包装器实现。如果使用了特定于包装器的 Fenced_Generic_User 子类，则必须实现此成员函数。

语法

```
virtual Remote_User* create_remote_user (sqluint8* a_user_name,
                                         sqlint32* a_rc)
```

输入自变量

表 161. create_remote_user 成员函数的输入自变量

名称	数据类型	描述
a_user_name	sqluint8*	远程用户的名称。

输出自变量

表 162. create_remote_user 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向已实例化的 Remote_User 的指针。

create_nickname 函数

定义位置

服务器

用途 实例化此服务器的昵称的适当子类。

用法 此成员函数必须由特定于包装器的受防护的服务器子类中的包装器实现。

语法

```
virtual Nickname* create_nickname (sqluint8* a_schema_name,
                                   sqluint8* a_nickname_name,
                                   sqlint32* a_rc)
```

输入自变量

表 163. create_nickname 成员函数的输入自变量

名称	数据类型	描述
a_schema_name	sqluint8*	包含昵称的本地（DB2）模式名称的以 Null 结束的字符串。
a_nickname_name	sqluint8*	包含本地（DB2）昵称名称的以 Null 结束的字符串。

输出自变量

表 164. create_nickname 成员函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 指向已实例化的昵称对象的指针。

相关任务:

- 『服务器类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 64 页的『C++ API 的服务器类』

C++ API 的用户类

下表描述了 C++ API 的每个用户类。

表 165. 用户类

类名	描述
Unfenced_Generic_User	此类表示不受防护（可信）的进程空间中的用户映射。此类将负责验证 CREATE USER MAPPING 和 ALTER USER MAPPING 语句中的信息。
Fenced_Generic_User	此类表示受防护（不可信）的进程空间中的用户映射。

相关参考:

- 第 78 页的『Unfenced_Generic_User 类（C++）』
- 第 82 页的『Fenced_Generic_User 类（C++）』

Unfenced_Generic_User 类（C++）

本主题描述 Unfenced_Generic_User 类并提供有关构造函数、析构函数和成员函数的详细信息。

概述

Unfenced_Generic_User 类表示不受防护（可信）的进程空间中的用户映射。此类将负责验证 CREATE USER MAPPING 和 ALTER USER MAPPING 语句中的信息。

Unfenced_Generic_User 类是 C++ API 的其中一个用户类。

用法 如果对 CREATE USER MAPPING 或 ALTER USER MAPPING 语句使用了特定于包装器的选项，则包装器必须实现 Unfenced_Generic_User 的子类。此类仅用于选项验证，并且是通过 Unfenced_Generic_Server 的特定于包装器的子类的 create_remote_user() 方法中的包装器来实例化的。

文件 sqlqg_unfenced_generic_user.h

数据成员

下表列示了可以与 Unfenced_Generic_User 类配合使用的数据成员。

表 166. Unfenced_Generic_User 类的数据成员

名称	数据类型	描述
local_name	sqluint8*	以 null 结束的（本地）用户名。
server	Server*	指向拥有此用户的服务器对象的指针。
info	User_Info*	目录信息的本地副本。只有在执行了 initialize_my_user 之后，此数据成员才有效。

构造函数、析构函数和成员函数

下列各表描述了 Unfenced_Generic_User 类的构造函数、析构函数和成员函数。在这些表的后面更详细地描述了构造函数、析构函数和成员函数。

表 167. Unfenced_Generic_User 类的构造函数

构造函数	描述
Unfenced_Generic_User	构造 Unfenced_Generic_User 的实例。

表 168. Unfenced_Generic_User 类的析构函数

析构函数	描述
~Unfenced_Generic_User	破坏 Unfenced_Generic_User 的实例。

表 169. Unfenced_Generic_User 类的成员函数

成员函数	描述
get_info	返回指向 User_Info 的本地副本的指针。
is_reserved_user_option	验证指定的目录选项是否是 DB2 为用户映射保留的选项。
initialize_my_user	根据有效的目录信息来初始化对象的状态。
verify_my_register_user_info	验证 CREATE USER MAPPING 语句中的信息。
verify_my_alter_user_info	验证 ALTER USER MAPPING 语句中的信息。

Unfenced_Generic_User 构造函数

用途 构造 Unfenced_Generic_User 的实例。

语法

```
Unfenced_Generic_User (sqluint8*       a_user_name,
                       UnfencedServer* a_server,
                       sqlint32*       a_rc)
```

输入自变量

表 170. Unfenced_Generic_User 构造函数的输入自变量

名称	数据类型	描述
a_user_name	sqluint8*	以 null 结束的（本地）用户名。
a_server	UnfencedServer*	指向拥有此用户的服务器对象的指针。

输出自变量

表 171. Unfenced_Generic_User 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向拥有此用户的服务器对象的指针。

返回值 无。

~Unfenced_Generic_User 析构函数

用途 破坏 Unfenced_Generic_User 的实例。

语法

```
virtual ~Unfenced_Generic_User ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

get_info 成员函数

用途 返回指向 User_Info 的本地副本的指针。

语法

```
User_Info* get_info ()
```

输入自变量

无。

输出自变量

无。

返回值 指向 User_Info 对象的指针。

is_reserved_user_option 成员函数

用途 验证指定的目录选项是否是 DB2 为用户映射保留的选项。

语法

```
virtual sqluint32 is_reserved_user_option (sqluint8* a_opt_name)
```

输入自变量

表 172. is_reserved_user_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	要检查的以 null 结束的选项名称。

输出自变量

无。

返回值 如果值为 0，则指示选项不是 DB2 保留的选项。

initialize_my_user 成员函数

用途 根据有效的目录信息来初始化对象的状态。

用法 此成员函数可以由特定于包装器的不受防护的用户子类中的包装器实现。

语法

```
virtual sqlint32 initialize_my_user (User_Info* a_user_info)
```

输入自变量

表 173. initialize_my_user 成员函数的输入自变量

名称	数据类型	描述
a_user_info	User_Info*	已验证的目录信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

verify_my_register_user_info 成员函数

用途 验证 CREATE USER MAPPING 语句中的信息。

用法 此成员函数可以由特定于包装器的不受防护的用户子类中的包装器实现。如果特定于包装器的用户映射选项受支持，则必须实现此成员函数。

包装器将验证在它自己分配一个 a_delta_info 对象之前是否已经分配了一个 a_delta_info 对象。

语法

```
virtual sqlint32 verify_my_register_user_info (User_Info*  a_user_info,
                                               User_Info** a_delta_info)
```

输入自变量

表 174. verify_my_register_user_info 成员函数的输入自变量

名称	数据类型	描述
a_user_info	User_Info*	来自 CREATE USER MAPPING 语句的信息。

输出自变量

表 175. verify_my_register_user_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	User_Info**	由包装器提供的附加信息。

返回值 返回码。如果值为 0，则指示成功。

verify_my_alter_user_info 成员函数

用途 验证 ALTER USER MAPPING 语句中的信息。

用法 此成员函数可以由特定于包装器的不受防护的用户子类中的包装器实现。如果特定于包装器的用户映射选项受支持，则必须实现此成员函数。

包装器将验证在它自己分配一个 a_delta_info 对象之前是否已经分配了一个 a_delta_info 对象。

语法

```
virtual sqlint32 verify_my_alter_user_info (User_Info*  a_user_info,
                                            User_Info** a_delta_info)
```

输入自变量

表 176. verify_my_alter_user_info 成员函数的输入自变量

名称	数据类型	描述
a_user_info	User_Info*	来自 ALTER USER MAPPING 语句的信息。

输出自变量

表 177. verify_my_alter_user_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	User_Info**	由包装器提供的附加信息。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『用户类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 77 页的『C++ API 的用户类』

Fenced_Generic_User 类（C++）

本主题描述 Fenced_Generic_User 类并提供有关构造函数、析构函数和成员函数的详细信息。

概述

Fenced_Generic_User 类表示受防护（不可信）的进程空间中的用户映射。

Fenced_Generic_User 类是 C++ API 的其中一个用户类。

用法 此类是通过 Fenced_Generic_Server 的特定于包装器的子类的 create_remote_user() 方法中的包装器来实例化的。如果特定于包装器的用户映射选项受支持，则包装器将实现 Fenced_Generic_User 的子类。

文件 sqlqg_fenced_generic_user.h

数据成员

下表列示了可以与 Fenced_Generic_User 类配合使用的数据成员。

表 178. Fenced_Generic_User 类的数据成员

名称	数据类型	描述
local_name	sqluint8*	以 null 结束的（本地）用户名。
server	Server*	指向拥有此用户的服务器对象的指针。
info	User_Info*	目录信息的本地副本。只有在执行了 initialize_my_user 之后，此数据成员才有效。

构造函数、析构函数和成员函数

下列各表描述了 Fenced_Generic_User 类的构造函数、析构函数和成员函数。在这些表的后面更详细地描述了构造函数、析构函数和成员函数。

表 179. Fenced_Generic_User 类的构造函数

构造函数	描述
Fenced_Generic_User	构造 Fenced_Generic_User 的实例。

表 180. Fenced_Generic_User 类的析构函数

析构函数	描述
~Fenced_Generic_User	破坏 Fenced_Generic_User 的实例。

表 181. Fenced_Generic_User 类的成员函数

成员函数	描述
get_info	返回指向 User_Info 的本地副本的指针。
initialize_my_user	根据有效的目录信息来初始化对象的状态。

Fenced_Generic_User 构造函数

用途 构造 Fenced_Generic_User 的实例。

语法

```
Fenced_Generic_User (sqluint8*     a_local_user_name,
                     FencedServer* a_server,
                     sqlint32*     a_rc)
```

输入自变量

表 182. Fenced_Generic_User 构造函数的输入自变量

名称	数据类型	描述
a_local_user_name	sqluint8*	以 null 结束的（本地）用户名。
a_server	FencedServer*	指向拥有此用户的服务器对象的指针。

输出自变量

表 183. Fenced_Generic_User 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

~Fenced_Generic_User 析构函数

用途 破坏 Fenced_Generic_User 的实例。

语法

```
virtual ~Fenced_Generic_User ()
```

输入自变量

无。

输出自变量

无。

返回值 无。

get_info 成员函数

用途 返回指向 User_Info 的本地副本的指针。

语法

```
User_Info* get_info ()
```

输入自变量

无。

输出自变量

无。

返回值 指向 User_Info 对象的指针。

initialize_my_user 成员函数

用途 根据有效的目录信息来初始化对象的状态。

用法 此成员函数可以由特定于包装器的受防护的用户子类中的包装器实现，如果特定于包装器的用户映射选项受支持，则必须实现此成员函数。

语法

```
virtual sqlint32 initialize_my_user (User_Info* a_user_info)
```

输入自变量

表 184. initialize_my_user 成员函数的输入自变量

名称	数据类型	描述
a_user_info	User_Info*	已验证的目录信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『用户类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 77 页的『C++ API 的用户类』

C++ API 的昵称类

下表描述了 C++ API 的每个昵称类。

表 185. 昵称类

类名	描述
Unfenced_Generic_Nickname	此类表示不受防护（可信）进程空间中的昵称。此类负责验证 CREATE NICKNAME 和 ALTER NICKNAME 语句中的信息。
Fenced_Generic_Nickname	此类表示受防护的（不可信）进程空间中的昵称。此类负责验证 CREATE NICKNAME 语句中的信息。

相关参考:

- 第 85 页的『Unfenced_Generic_Nickname 类（C++）』
- 第 91 页的『Fenced_Generic_Nickname 类（C++）』

Unfenced_Generic_Nickname 类（C++）

本主题描述 Unfenced_Generic_Nickname 类并提供构造函数和成员函数的详细信息。

概述

Unfenced_Generic_Nickname 类表示不受防护的（可信）进程空间中的昵称。此类负责验证 CREATE NICKNAME 和 ALTER NICKNAME 语句中的信息。

Unfenced_Generic_Nickname 类是 C++ API 的其中一个昵称类。

用法 必须由包装器来创建此类的子类。此类由包装器在 Unfenced_Generic_Server 的特定于包装器的子类的 create_nickname() 方法中实例化。

文件 sqlqg_unfenced_generic_nickhame.h

数据成员

下表列示可以与 Unfenced_Generic_Nickname 类配合使用的数据成员。

表 186. Unfenced_Generic_Nickname 类的数据成员

名称	数据类型	描述
名称	sqluint8*	包含昵称的（本地）名称的以 null 结束的字符串。
模式	sqluint8*	包含模式的（本地）名称的以 null 结束的字符串。
服务器	Server*	指向拥有此昵称的服务器对象的指针。
m_cardinality	sqlint64	基数。
m_advance_cost	sqlint32	访存行的成本（时间，以毫秒计）。
m_setup_cost	sqlint32	执行查询的一次设置的成本（时间，以毫秒计）。

表 186. Unfenced_Generic_Nickname 类的数据成员 (续)

名称	数据类型	描述
m_submission_cost	sqlint32	提交查询的成本（不同于一次设置成本）（时间，以毫秒计）。

构造函数和成员函数

下列各表描述 Unfenced_Generic_Nickname 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 187. Unfenced_Generic_Nickname 类的构造函数

构造函数	描述
Unfenced_Generic_Nickname	构造 Unfenced_Generic_Nickname 的实例。

表 188. Unfenced_Generic_Nickname 类的成员函数

成员函数	描述
get_local_name	返回指向以 null 结束的本地昵称名的指针。
get_local_schema	返回指向以 null 结束的本地模式名的指针。
get_server	返回指向包含服务器对象的指针。
is_reserved_nickname_option	确定特定选项名是否为内置 DB2 昵称选项之一。
is_reserved_column_option	确定特定选项名是否为内置 DB2 列选项之一。
initialize_my_nickname	使用目录中的有效信息初始化昵称。
verify_my_register_nickname_info	验证 CREATE NICKNAME 语句中的信息。
verify_my_alter_nickname_info	验证 ALTER NICKNAME 语句中的信息。
get_card	检索昵称的基数。
get_advance_cost	检索在缺省成本模型中使用的 ADVANCE_COST 昵称选项的值。
get_setup_cost	检索在缺省成本模型中使用的 SETUP_COST 昵称选项的值。
get_submission_cost	检索在缺省成本模型中使用的 SUBMISSION_COST 昵称选项的值。

Unfenced_Generic_Nickname 构造函数

用途 构造 Unfenced_Generic_Nickname 的实例。

语法

```
Unfenced_Generic_Nickname (sqluint8* a_schema,
          sqluint8*                a_nickname_name,
          Unfenced_Generic_Server* a_nickname_server,
          sqlint32*                a_rc)
```

输入自变量

表 189. Unfenced_Generic_Nickname 构造函数的输入自变量

名称	数据类型	描述
a_schema	sqluint8*	以 null 结束的模式名。
a_nickname_name	sqluint8*	以 null 结束的（本地）昵称名。
a_nickname_server	Unfenced_Generic_Server*	指向包含服务器对象的指针。

输出自变量

表 190. Unfenced_Generic_Nickname 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

get_local_name 函数

用途 返回指向以 null 结束的本地昵称名的指针。

语法

```
sqluint8* get_local_name ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的昵称名。

get_local_schema 函数

用途 返回指向以 null 结束的本地模式名的指针。

语法

```
sqluint8* get_local_schema ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的模式名。

get_server 函数

用途 返回指向包含服务器对象的指针。

语法

```
Server* get_server ()
```

输入自变量

无。

输出自变量

无。

返回值 指向包含服务器对象的指针。

is_reserved_nickname_option 函数

用途 确定特定选项名是否为内置 DB2 昵称选项之一。

语法

```
virtual sqluint32 is_reserved_nickname_option (sqluint8* a_opt_name)
```

输入自变量

表 191. is_reserved_nickname_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	以 null 结束的选项名字符串。

输出自变量

无。

返回值 值 0 指示选项不是保留昵称选项。非零返回值指示选项是保留昵称选项。

is_reserved_column_option 函数

用途 确定特定选项名是否为内置 DB2 列选项之一。

语法

```
virtual sqluint32 is_reserved_column_option (sqluint8* a_opt_name)
```

输入自变量

表 192. is_reserved_column_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	以 null 结束的选项名字符串。

输出自变量

无。

返回值 值 0 指示选项不是保留选项。非零返回值指示选项是保留列选项。

initialize_my_nickname 函数

用途 使用目录中的有效信息初始化昵称。

用法 此成员函数可以由包装器在特定于包装器的不受防护的昵称子类中实现。

语法

```
virtual sqlint32 initialize_my_nickname (Nickname_Info*  a_cat_info)
```

输入自变量

表 193. initialize_my_nickname 成员函数的输入自变量

名称	数据类型	描述
a_cat_info	Nickname_Info*	目录中的信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

verify_my_register_nickname_info 函数

用途 验证 CREATE NICKNAME 语句中的信息。

用法 此成员函数可以由包装器在特定于包装器的受防护的昵称子类中实现。如果支持特定于包装器的昵称或列选项，则必须实现此方法或者与受防护类相同的方法。因为 verify_my_register_nickname_info 函数是可信进程空间的一部分，所以此成员函数不能与远程数据源交互作用。如果验证昵称信息需要交互作用，则必须实现 Fenced_Generic_Nickname 类的 verify_my_register_nickname_info 成员函数。

包装器在分配它自己的 a_delta_info 对象之前必须检查是否已经分配了一个 a_delta_info 对象。在调用 Fenced_Generic_Nickname::verify_my_register_nickname_info 后，调用此方法。

语法

```
virtual sqlint32 verify_my_register_nickname_info
       (Nickname_Info*   a_nick_info,
        Nickname_Info**  a_delta_info)
```

输入自变量

表 194. verify_my_register_nickname_info 成员函数的输入自变量

名称	数据类型	描述
a_nick_info	Nickname_Info*	CREATE NICKNAME 语句中的信息。

输出自变量

表 195. verify_my_register_nickname_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Nickname_Info**	包装器添加的信息。

返回值 返回码。如果值为 0，则指示成功。

verify_my_alter_nickname_info 函数

用途 验证 ALTER NICKNAME 语句中的信息。

用法 此成员函数可以由包装器在特定于包装器的昵称子类中实现。如果支持特定于包装器的昵称或列选项，则必须实现此成员函数。因为 verify_my_alter_nickname_info 函数是可信进程空间的一部分，所以此成员函数不能与远程数据源交互作用。

包装器在分配它自己的 a_delta_info 对象之前必须检查是否已经分配了一个 a_delta_info 对象。

语法

```
virtual sqlint32 verify_my_alter_nickname_info
       (Nickname_Info*  a_nick_info,
        Nickname_Info** a_delta_info)
```

输入自变量

表 196. verify_my_alter_nickname_info 成员函数的输入自变量

名称	数据类型	描述
a_nick_info	Nickname_Info*	ALTER NICKNAME 语句中的信息。

输出自变量

表 197. verify_my_alter_nickname_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Nickname_Info**	包装器添加的信息。

返回值 返回码。如果值为 0，则指示成功。

get_card 函数

用途 检索昵称的基数。基数存储在 DB2 Information Integrator 系统目录中。

语法

```
void  get_card (sqlint64* a_cardinality) const
```

输入自变量

无。

输出自变量

表 198. get_card 成员函数的输出自变量

名称	数据类型	描述
a_cardinality	sqlint64*	基数。

返回值 无。

get_advance_cost 函数

用途 检索在缺省成本模型中使用的 ADVANCE_COST 昵称选项的值。

语法

```
sqlint32 get_advance_cost (void) const
```

输入自变量

无。

输出自变量

无。

返回值 ADVANCE_COST 昵称选项的值或缺省值（如果未对昵称指定该选项的话）。

get_setup_cost 函数

用途 检索在缺省成本模型中使用的 SETUP_COST 昵称选项的值。

语法

```
sqlint32 get_setup_cost (void) const
```

输入自变量

无。

输出自变量

无。

返回值 SETUP_COST 昵称选项的值或缺省值（如果未对昵称指定该选项的话）。

get_submission_cost 函数

用途 检索在缺省成本模型中使用的 SUBMISSION_COST 昵称选项的值。

语法

```
sqlint32 get_submission_cost (void) const
```

输入自变量

无。

输出自变量

无。

返回值 SUBMISSION_COST 昵称选项的值或缺省值（如果未对昵称指定该选项的话）。

相关任务:

- 『昵称类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 85 页的『C++ API 的昵称类』

Fenced_Generic_Nickname 类（C++）

本主题描述 Fenced_Generic_Nickname 类并提供构造函数和成员函数的详细信息。

概述

Fenced_Generic_Nickname 类表示受防护的（不可信）进程空间中的昵称。此类负责验证 CREATE NICKNAME 语句中的信息。

Fenced_Generic_Nickname 类是 C++ API 的其中一个昵称类。

用法 必须由包装器来创建此类的子类，并由包装器在 Fenced_Generic_Server 的特定于包装器的子类的 create_nickname() 方法中实例化此类。

文件 sqlqg_fenced_generic_nickname.h

数据成员

下表列示可以与 Fenced_Generic_Nickname 类配合使用的数据成员。

表 199. Fenced_Generic_Nickname 类的数据成员

名称	数据类型	描述
名称	sqluint8*	包含昵称的（本地）名称的以 null 结束的字符串。
模式	sqluint8*	包含模式的（本地）名称的以 null 结束的字符串。
服务器	Server*	指向拥有此昵称的服务器对象的指针。

构造函数和成员函数

下列各表描述 Fenced_Generic_Nickname 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 200. Fenced_Generic_Nickname 类的构造函数

构造函数	描述
Fenced_Generic_Nickname	构造 Fenced_Generic_Nickname 的实例。

表 201. Fenced_Generic_Nickname 类的成员函数

成员函数	描述
get_local_name	返回指向以 null 结束的本地昵称名的指针。
get_local_schema	返回指向以 null 结束的本地模式名的指针。
get_server	返回指向包含服务器对象的指针。
is_reserved_nickname_option	确定特定选项名是否为内置 DB2 昵称选项之一。
is_reserved_column_option	确定特定选项名是否为内置 DB2 列选项之一。
initialize_my_nickname	使用目录中的有效信息初始化昵称。
verify_my_register_nickname_info	验证 CREATE NICKNAME 语句中的信息。

Fenced_Generic_Nickname 构造函数

用途 构造 Fenced_Generic_Nickname 的实例。

语法

```
Fenced_Generic_Nickname (sqluint8*              a_schema,
                         sqluint8*              a_name,
                         Fenced_Generic_Server* a_server,
                         sqlint32*              a_rc)
```

输入自变量

表 202. Fenced_Generic_Nickname 构造函数的输入自变量

名称	数据类型	描述
a_schema	sqluint8*	以 null 结束的模式名。
a_name	sqluint8*	以 null 结束的（本地）昵称名。
a_server	Fenced_Generic_Server*	指向包含服务器对象的指针。

输出自变量

表 203. Fenced_Generic_Nickname 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

get_local_name 函数

用途 返回指向以 null 结束的本地昵称名的指针。

语法

```
sqluint8* get_local_name ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的昵称名。

get_local_schema 函数

用途 返回指向以 null 结束的本地模式名的指针。

语法

```
sqluint8* get_local_schema ()
```

输入自变量

无。

输出自变量

无。

返回值 以 null 结束的模式名。

get_server 函数

用途 返回指向包含服务器对象的指针。

语法

```
Server* get_server ()
```

输入自变量

无。

输出自变量

无。

返回值 指向包含服务器对象的指针。

is_reserved_nickname_option 函数

用途 确定特定选项名是否为内置 DB2 昵称选项之一。

语法

```
virtual sqluint32 is_reserved_nickname_option (sqluint8* a_opt_name)
```

输入自变量

表 204. is_reserved_nickname_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	以 null 结束的选项名字符串。

输出自变量

无。

返回值 值 0 指示选项不是保留选项。非零返回值指示选项是保留昵称选项。

is_reserved_column_option 函数

用途 确定特定选项名是否为内置 DB2 列选项之一。

语法

```
virtual sqluint32 is_reserved_column_option (sqluint8* a_opt_name)
```

输入自变量

表 205. is_reserved_column_option 成员函数的输入自变量

名称	数据类型	描述
a_opt_name	sqluint8*	以 null 结束的选项名字符串。

输出自变量

无。

返回值 值 0 指示选项不是保留选项。非零返回值指示选项是保留列选项。

initialize_my_nickname 函数

用途 使用目录中的有效信息初始化昵称。

用法 包装器可以在特定于包装器的受防护昵称子类中实现此成员函数。

语法

```
virtual sqlint32 initialize_my_nickname (Nickname_Info*  a_nick_info)
```

输入自变量

表 206. initialize_my_nickname 成员函数的输入自变量

名称	数据类型	描述
a_nick_info	Nickname_Info*	目录中的信息。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

verify_my_register_nickname_info 函数

用途 验证 CREATE NICKNAME 语句中的信息。

用法 包装器可以在特定于包装器的受防护昵称子类中实现此成员函数。如果支持特定于包装器的昵称或列选项，则必须实现此方法或者与不受防护类相同的方法。如果在验证昵称信息时包装器必须与远程数据源交互作用，则实现此方法。

包装器在分配它自己的 a_delta_info 对象之前必须检查是否已经分配了一个 a_delta_info 对象。在调用 Unfenced_Generic_Nickname::verify_my_register_nickname_info 前调用此方法。

语法

```
virtual sqlint32 verify_my_register_nickname_info
       (Nickname_Info*  a_nick_info,
        Nickname_Info** a_delta_info)
```

输入自变量

表 207. verify_my_register_nickname_info 成员函数的输入自变量

名称	数据类型	描述
a_nick_info	Nickname_Info*	CREATE NICKNAME 语句中的信息。

输出自变量

表 208. verify_my_register_nickname_info 成员函数的输出自变量

名称	数据类型	描述
a_delta_info	Nickname_Info**	包装器添加的信息。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『昵称类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 85 页的『C++ API 的昵称类』

Remote_Connection 类（C++）

本主题描述 Remote_Connection 类并提供构造函数和成员函数的详细信息。

概述

Remote_Connection 类表示 DB2 与远程数据源之间的连接。此类管理连接、创建远程操作对象并维护远程操作对象。

Remote_Connection 类是 C++ API 的连接类。

用法 包装器可以创建此类的子类，以创建特定于包装器的连接子类。此类由特定于包装器的 Fenced_Generic_Server 子类的 create_remote_connection() 方法实例化。

文件 sqlqg_connection.h

数据成员

下表列示可以与 Remote_Connection 类配合使用的数据成员。

表 209. Remote_Connection 类的数据成员

名称	数据类型	描述
种类	connection_kind	连接的种类（1 阶段或 2 阶段）。
服务器	FencedServer*	指向拥有此连接的服务器的指针。
用户	FencedRemote_User*	与连接相关联的用户。
连接	unsigned short	指示连接处于活动状态的标志。

构造函数和成员函数

下列各表描述 Remote_Connection 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 210. Remote_Connection 类的构造函数

构造函数	描述
Remote_Connection	构造 Remote_Connection 对象。

表 211. Remote_Connection 类的成员函数

成员函数	描述
connect	建立与远程数据源的连接。
disconnect	破坏与远程数据源的连接。
is_connected	指示连接是活动的还是不活动的。
create_remote_query	创建 Remote_Query 的适当的子类。
create_remote_passthru	创建 Remote_Passthru 的适当的子类。
get_server	返回指向包含服务器的指针。
get_user	返回指向相关联的 Remote_User 对象的指针。

表 211. Remote_Connection 类的成员函数 (续)

成员函数	描述
connect	指示事务成功完成于是远程数据源落实该事务。
rollback	指示事务未成功完成于是远程数据源回滚该事务。

Remote_Connection 构造函数

用途 构造 Remote_Connection 对象。

语法

```
Remote_Connection (FencedServer*      remote_server,
                   FencedRemote_User* remote_user,
                   connection_kind    k=one_phase_kind,
                   sqlint32*          rc = 0)
```

输入自变量

表 212. Remote_Connection 构造函数的输入自变量

名称	数据类型	描述
FencedServer*	remote_server	拥有此连接的服务器。
remote_user	FencedRemote_User*	与连接相关联的用户。
k	connection_kind	连接的种类。

输出自变量

表 213. Remote_Connection 构造函数的输出自变量

名称	数据类型	描述
rc	sqlint32*	指向返回码的指针；如果值为 0，则指示成功。

返回值 无。

connect 函数

用途 建立与远程数据源的连接。

用法 DB2 Information Integrator 调用此成员函数以打开与远程数据源的连接。此成员函数必须由包装器在特定于包装器的 Remote_Connection 子类中实现。缺省行为将报告错误。

语法

```
virtual sqlint32 connect ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

disconnect 函数

用途 破坏与远程数据源的连接。

用法 DB2 Information Integrator 调用此成员函数以关闭与远程数据源的连接。此成员函数必须由包装器在特定于包装器的 Remote_Connection 子类中实现。缺省行为将报告错误。

语法

```
virtual sqlint32 disconnect ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

is_connected 函数

用途 指示连接是活动的还是不活动的。

语法

```
unsigned short is_connected ()
```

输入自变量

无。

输出自变量

无。

返回值 如果连接是活动的，则值为 TRUE。否则，返回值为 FALSE。

create_remote_query 函数

用途 创建 Remote_Query 的适当的子类。

用法 DB2 Information Integrator 调用此成员函数以实例化特定于包装器的 Remote_Query 子类。包装器必须在特定于包装器的 Remote_Connection 子类中实现此成员函数。

语法

```
virtual sqlint32 create_remote_query
                (Runtime_Operation* runtime_query,
                 Remote_Query**     query)
```

输入自变量

表 214. create_remote_query 成员函数的输入自变量

名称	数据类型	描述
runtime_query	Runtime_Operation*	描述该操作的 DB2 内部结构。

表 214. create_remote_query 成员函数的输入自变量 (续)

名称	数据类型	描述
query	Remote_Query**	指向已分配的 Remote_Query 对象的指针。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

create_remote_passthru 函数

用途 创建 Remote_Passthru 的适当的子类。

用法 DB2 Information Integrator 调用此成员函数以实例化特定于包装器的 Remote_Passthru 子类。包装器可以在特定于包装器的 Remote_Connection 子类中实现此成员函数。仅当包装器支持 PASSTHRU 方式时，此成员函数才是必需的。

语法

```
virtual sqlint32 create_remote_passthru
                 (Runtime_Operation*   runtime_passthru,
                  Remote_Passthru**    query)
```

输入自变量

表 215. create_remote_passthru 成员函数的输入自变量

名称	数据类型	描述
runtime_passthru	Runtime_Operation*	描述该操作的 DB2 内部结构。
query	Runtime_Passthru**	指向已分配的 Remote_Passthru 对象的指针。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_server 函数

用途 返回指向包含服务器的指针。

语法

```
FencedServer* get_server()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器的指针。

get_user 函数

用途 返回指向相关联的 Remote_User 对象的指针。

语法

```
FencedRemote_User* get_user ()
```

输入自变量

无。

输出自变量

无。

返回值 指向远程用户对象的指针。

commit 函数

用途 指示事务成功完成于是远程数据源落实该事务。

用法 DB2 Information Integrator 调用此成员函数以指示事务成功完成。此成员函数必须由包装器在特定于包装器的 Remote_Connection 子类中实现。缺省行为将报告错误。

语法

```
sqlint32 commit ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

rollback 函数

用途 指示事务未成功完成于是远程数据源回滚该事务。

用法 此成员函数由 DB2 Information Integrator 调用以指示事务未成功完成且远程数据源回滚该事务。此成员函数可以由包装器在特定于包装器的 Remote_Connection 子类中实现。缺省行为将报告错误。

语法

```
sqlint32  rollback ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『Remote connection 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 64 页的『C++ API 的服务器类』

C++ API 的操作类

下表描述了 C++ API 的每个操作类。

表 216. 操作类

类名	描述
Remote_Query	此类包括对远程数据源运行查询所需的信息。包装器必须创建此类的子类才能实现该操作。
Remote_Passthru	此类表示远程数据源处的传递（Pass-Through）操作。包装器必须创建 Remote_Passthru 类的子类才能支持传递（Pass-Through）操作。

相关参考:

- 第 101 页的『Remote_Query 类（C++）』
- 第 111 页的『Remote_Passthru 类（C++）』

Remote_Query 类（C++）

本主题描述 Remote_Query 类并提供有关构造函数和成员函数的详细信息。

概述

Remote_Query 类包括对远程数据源运行查询所需要的信息。包装器必须创建此类的子类才能实现该操作。

Remote_Query 类是 C++ API 的其中一个操作类。

用法 此类是通过特定于包装器的 Remote_Connection 子类的 create_remote_query() 方法中的包装器来实例化的。

文件 sqlqg_operation.h

父类 Remote_Operation。

数据成员

无。

构造函数和成员函数

下列各表描述了 Remote_Query 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 217. Remote_Query 类的构造函数

构造函数	描述
Remote_Query	构造 Remote_Query 对象。

表 218. Remote_Query 类的成员函数

成员函数	描述
get_connection	检索指向拥有此操作的连接的指针。
get_server	检索指向拥有此操作的服务器的指针。
get_input_data	检索指向操作的输入值列表的指针。
get_output_data	检索指向输出数据缓冲区列表的指针。
get_exec_desc	检索此查询的执行描述符的指针和长度。
open	打开查询。
reopen	重新打开查询。
fetch	访存单个行。
close	关闭查询。
report_eof	报告访存期间的文件结束符条件。
get_status	检索当前查询状态。
set_status	设置当前查询状态。
fetch_lob	将 LOB 数据检索到由 DB2 提供的特殊 LOB 缓冲区中。
set_lob_next	暂挂非 LOB 列的处理并开始处理一个或多个 LOB 列。
lob_data_ready	指示包装器已将一部分 LOB 数据传送到缓冲区。
open_input_lob	打开具有 LOB 参数的查询。
reopen_input_lob	重新打开先前已打开的结果流，并可能根据不同的参数绑定来准备数据源，以便为包含 LOB 参数的查询返回更多结果集。
set_row_status	设置当前行状态。

Remote_Query 构造函数

用途 构造 Remote_Query 对象。

语法

```
Remote_Query (Remote_Connection* a_active_connection,
              Runtime_Operation* a_runtime_info,
              sqlint32*          a_rc)
```

输入自变量

表 219. Remote_Query 构造函数的输入自变量

名称	数据类型	描述
a_active_connection	Remote_Connection*	此查询的连接。
a_runtime_info	Runtime_Operation*	此查询的内部数据。

输出自变量

表 220. Remote_Query 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	返回码；如果值为 0，则指示成功。

返回值 无。

get_connection 函数

定义位置

Remote_Operation

用途 检索指向拥有此操作的连接的指针。

语法

```
Remote_Connection* get_connection ()
```

输入自变量

无。

输出自变量

无。

返回值 指向 Remote_Connection 对象的指针。

get_server 函数

定义位置

Remote_Operation

用途 检索指向拥有此操作的服务器的指针。

语法

```
Fenced_Generic_Server* get_server ()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器对象的指针。

get_input_data 函数

定义位置

Remote_Operation

用途 检索指向操作的输入值列表的指针。

语法

```
Runtime_Data_List* get_input_data ()
```

输入自变量

无。

输出自变量

无。

返回值 指向数据值列表的指针。

get_output_data 函数

定义位置

Remote_Operation

用途 检索指向输出数据缓冲区列表的指针。

语法

```
Runtime_Data_List* get_output_data ()
```

输入自变量

无。

输出自变量

无。

返回值 指向数据缓冲区列表的指针。

get_exec_desc 函数

定义位置

Remote_Operation

用途 检索此查询的执行描述符的指针和长度。

语法

```
void get_exec_desc (void** a_exec_desc,
                    int*   a_exec_desc_len)
```

输入自变量

无。

输出自变量

表 221. get_exec_desc 成员函数的输出自变量

名称	数据类型	描述
a_exec_desc	void**	指向执行描述符的指针。
a_exec_desc_len	int*	描述符的长度。

返回值 无。

open 函数

用途 打开查询。

用法 联合服务器调用此成员函数以启动查询。此成员函数必须由特定于包装器的 Remote_Query 子类中的包装器实现。

语法

```
sqlint32 open ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

reopen 函数

用途 重新打开查询

用法 联合服务器调用此成员函数以启动包含新参数值的查询。此成员函数必须由特定于包装器的 Remote_Query 子类中的包装器实现。

语法

```
sqlint32 reopen (sqlint16 a_status)
```

输入自变量

表 222. reopen 成员函数的输入自变量

名称	数据类型	描述
a_status	sqlint16	未使用。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

fetch 函数

用途 访存单个行。

用法 DB2 Information Integrator 调用此成员函数以便从远程查询中访存一个非 LOB 数据行。此成员函数必须由特定于包装器的 Remote_Query 子类中的包装器实现。

语法

```
sqlint32 fetch ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

close 函数

用途 关闭查询。

用法 DB2 Information Integrator 调用此成员函数以关闭查询游标。此成员函数必须由特定于包装器的 Remote_Query 子类中的包装器实现。

语法

```
sqlint32 close (sqlint16 a_status)
```

输入自变量

表 223. close 成员函数的输入自变量

名称	数据类型	描述
a_status	sqlint16	状态（SQLQG_CLOSE_EOS、SQLQG_CLOSE_EOA 或 SQLQG_CLOSE_EOQ）。SQLQG_CLOSE_EOS 指示包装器脱离了某种状态，以便可以调用 reopen() 函数。SQLQG_CLOSE_EOA 指示包装器可以完成所有必需的处理。未使用 SQLQG_CLOSE_EOQ。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

report_eof 函数

用途 报告访存期间的文件结束符条件。

用法 在执行 fetch() 方法期间，特定于包装器的 Remote_Query 类调用此成员函数来指示已访存了最后一行。

语法

```
sqlint32 report_eof ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。此代码是从 fetch() 返回的。

get_status 函数

用途 检索当前查询状态。

语法

```
sqluint8 get_status ()
```

输入自变量

无。

输出自变量

无。

返回值 状态（SQLQG_UNREADY、 SQLQG_READY、 SQLQG_OPEN 或 SQLQG_EOF）。

set_status 函数

用途 设置当前查询状态。

语法

```
void set_status (sqluint8 a_new_status)
```

输入自变量

表 224. set_status 成员函数的输入自变量

名称	数据类型	描述
a_new_status	sqluint8	状态（SQLQG_UNREADY、SQLQG_READY、SQLQG_OPEN 或 SQLQG_EOF）。

输出自变量

无。

返回值 无。

fetch_lob 函数

用途 将 LOB 数据检索到由 DB2 提供的特殊 LOB 缓冲区中。

语法

```
virtual sqlint32 fetch_lob (unsigned char* a_buffer,
                            sqluint32      a_buffer_size,
                            sqluint32      a_bytes_written)
```

输入自变量

表 225. fetch_lob 成员函数的输入自变量

名称	数据类型	描述
a_buffer	unsigned char*	由 DB2 提供的 LOB 数据缓冲区的地址。
a_buffer_size	sqluint32	缓冲区的大小（以字节计）。
a_bytes_written	sqluint32	由先前对 fetch_lob 函数的调用为当前列写入的总字节数。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

set_lob_next 函数

用途 暂挂非 LOB 列的处理并开始处理一个或多个 LOB 列。此函数是从 fetch 函数中调用的。

语法

```
void set_lob_next (void)
```

输入自变量

无。

输出自变量

无。

返回值 无。

lob_data_ready 函数

用途 指示包装器将一部分 LOB 数据传送到缓冲区。

语法

```
sqlint32 lob_data_ready (sqlint32          a_col_number,
                         sqluint32         a_bytes_ready,
                         sqlqg_lob_status  a_status,
                         sqlqg_lob_intent  a_next)
```

输入自变量

表 226. lob_data_ready 成员函数的输入自变量

名称	数据类型	描述
a_col_number	sqlint32	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。
a_bytes_ready	sqluint32	包装器已复制到 LOB 缓冲区中的字节数。
a_status	sqlqg_lob_status	用来指示状态的枚举数据类型。如果包装器正在处理一列的最后部分，则该值为 LOB_LAST。否则，该值为 LOB_MORE。
a_next	sqlqg_lob_intent	用来指示包装器接下来是调用 fetch 函数还是调用 fetch_lob 函数的枚举数据类型。有效值为 LOB_NEXT、NON_LOB_NEXT 和 ROW_DONE。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

open_input_lob 函数

用途 准备远程数据源以返回包含 LOB 参数的查询的第一个结果行。

用法 联合服务器调用此成员函数以启动包含 LOB 参数的查询。如果包装器支持 LOB 参数，则该包装器必须在特定于包装器的 Remote_Query 子类中实现此成员函数。如果查询具有 LOB 参数，则调用 open_input_lob 成员函数而不是 open 成员函数。

语法

```
virtual  sqlint32 open_input_lob (sqlint32*       a_column_number,
                                  sqluint32       a_mat_size,
                                  sqluint32       a_xfer_bytes,
                                  unsigned char*  a_buffer)
```

输入自变量

表 227. 为 open_input_lob 成员函数输入自变量

名称	数据类型	描述
a_column_number	sqlint32*	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。
a_mat_size	sqluint32	LOB 输入的总大小（以字节计）。
a_xfer_bytes	sqluint32	当前 LOB 片段的大小（以字节计）。
a_buffer	unsigned char*	当前 LOB 片段。对于 CLOB 变量而言，对象是一个未终结字符数组，对于 BLOB 变量而言，对象是一个字节数组。

输出自变量

表 228. open_input_lob 成员函数的输出自变量

名称	数据类型	描述
a_column_number	sqlint32*	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。

返回值 如果值为 0，则指示成功。其它任何返回码指示失败。

reopen_input_lob 函数

用途 重新打开先前已打开的结果流，并可能根据不同的参数绑定来准备数据源，以便为包含 LOB 参数的查询返回更多结果集。

用法 除非事先在查询结束状态下关闭了查询，否则不调用 reopen_input_lob 函数。

如果找到输入 LOB 主变量，则联合服务器调用 reopen_input_lob 函数而不是 reopen 函数。

如果包装器支持 LOB 参数，则该包装器必须在特定于包装器的 Remote_Query 子类中实现 reopen_input_lob 函数。

语法

```
virtual  sqlint32 reopen_input_lob (sqlint16       a_status,
                                    sqlint32*      a_column_number,
                                    sqluint32      a_mat_size,
                                    sqluint32      a_xfer_bytes,
                                    unsigned char* a_buffer)
```

输入自变量

表 229. reopen_input_lob 成员函数的输入自变量

名称	数据类型	描述
a_status	sqlint16	未使用。
a_column_number	sqlint32*	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。
a_mat_size	sqluint32	LOB 输入的总大小（以字节计）。
a_xfer_bytes	sqluint32	当前 LOB 片段的大小（以字节计）。
a_buffer	unsigned char*	当前 LOB 片段。对于 CLOB 变量而言，对象是一个未终结字符数组，对于 BLOB 变量而言，对象是一个字节数组。

输出自变量

表 230. reopen_input_lob 成员函数的输出自变量

名称	数据类型	描述
a_column_number	sqlint32*	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。

返回值 如果值为 0，则指示成功。其它任何返回码指示失败。

set_row_status 函数

用途 设置当前行状态。

语法

```
virtual void set_row_status (sqluint32 a_row_status)
```

输入自变量

表 231. set_row_status 成员函数的输入自变量

名称	数据类型	描述
a_row_status	sqluint32	状态（SQLQG_LOB_INIT、SQLQG_LOB_MORE 或 SQLQG_LOB_LAST）。

输出自变量

无。

返回值 无。

相关参考:

- 『Remote query 类』（《*IBM DB2 Information Integrator 包装器开发者指南*》）
- 第 101 页的『C++ API 的操作类』

Remote_Passthru 类（C++）

本主题描述 Remote_Passthru 类并提供有关构造函数和成员函数的详细信息。

概述

Remote_Passthru 类表示远程数据源中的传递（Pass-Through）操作。包装器必须创建 Remote_Passthru 类的子类才能支持传递（Pass-Through）操作。

Remote_Passthru 类是 C++ API 的其中一个操作类。

用法 此类是通过特定于包装器的 Remote_Connection 子类的 create_remote_passthru() 方法来实例化的。

文件 sqlqg_operation.h

父类 Remote_Operation

数据成员

无。

构造函数和成员函数

下列各表描述了 Remote_Passthru 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 232. Remote_Passthru 类的构造函数

构造函数	描述
Remote_Passthru	构造 Remote_Passthru 对象。

表 233. Remote_Passthru 类的成员函数

成员函数	描述
get_connection	检索指向拥有此操作的连接的指针。
get_server	检索指向拥有此操作的服务器的指针。
get_input_data	检索指向操作的输入值列表的指针。
get_output_data	检索指向输出数据缓冲区列表的指针。
get_SQL_statement	检索用于通过执行的语句。
report_eof	报告访存期间的文件结束符条件。
prepare	准备远程传递（Pass-Through）操作。
describe	描述远程传递（Pass-Through）操作。
execute	运行远程传递（Pass-Through）操作。
open	打开远程传递（Pass-Through）操作的游标。
fetch	在远程传递（Pass-Through）操作中访存一行。
close	关闭远程传递（Pass-Through）操作的游标。

Remote_Passthru 构造函数

用途 构造 Remote_Passthru 对象。

语法

```
Remote_Passthru (Remote_Connection* a_active_connection,
                 Runtime_Operation* a_runtime_passthru,
                 sqlint32*          a_rc)
```

输入自变量

表 234. Remote_Passthru 构造函数的输入自变量

名称	数据类型	描述
a_active_connection	Remote_Connection*	此查询的连接。
a_runtime_info	Runtime_Operation*	查询的内部数据。

输出自变量

表 235. Remote_Passthru 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	返回码；如果值为 0，则指示成功。

返回值 无。

get_connection 函数

定义位置

Remote_Operation

用途 检索指向拥有此操作的连接的指针。

语法

```
Remote_Connection* get_connection ()
```

输入自变量

无。

输出自变量

无。

返回值 指向 Remote_Connection 对象的指针。

get_server 函数

定义位置

Remote_Operation

用途 检索指向拥有此操作的服务器的指针。

语法

```
Fenced_Generic_Server* get_server ()
```

输入自变量
无。

输出自变量
无。

返回值 指向服务器对象的指针。

get_input_data 函数

定义位置
Remote_Operation

用途 检索指向操作的输入值列表的指针。

语法

```
Runtime_Data_List* get_input_data ()
```

输入自变量
无。

输出自变量
无。

返回值 指向数据值列表的指针。

get_output_data 函数

定义位置
Remote_Operation

用途 检索指向输出数据缓冲区列表的指针。

语法

```
Runtime_Data_List* get_output_data ()
```

输入自变量
无。

输出自变量
无。

返回值 指向数据缓冲区列表的指针。

get_SQL_statement 函数

定义位置
Remote_Operation

用途 检索用于通过执行的语句。

语法

```
char* get_SQL_statement ()
```

输入自变量
无。

输出自变量

无。

返回值 语句（以 null 结束）。

report_eof 函数

定义位置

Remote_Operation

用途 报告访存期间的文件结束符条件。

语法

```
sqlint32 report_eof ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。此代码是从 fetch() 返回的。

prepare 函数

用途 准备远程传递（Pass-Through）操作。

语法

```
sqlint32 prepare (Runtime_Data_Desc_List* a_data_description_list)
```

输入自变量

无。

输出自变量

表 236. prepare 成员函数的输出自变量

名称	数据类型	描述
a_data_description_list	Runtime_Data_Desc_List*	描述传递（Pass-Through）操作的结果的 Runtime_Data_Desc 的列表。

返回值 返回码。如果值为 0，则指示成功。

describe 函数

用途 描述远程传递（Pass-Through）操作。

语法

```
sqlint32 describe (Runtime_Data_Desc_List* a_data_description_list)
```

输入自变量

无。

输出自变量

表 237. describe 成员函数的输出自变量

名称	数据类型	描述
a_data_description_list	Runtime_Data_Desc_List*	描述传递（Pass-Through）操作的结果的 Runtime_Data_Desc 的列表。

返回值 返回码。如果值为 0，则指示成功。

execute 函数

用途 运行远程传递（Pass-Through）操作。

语法

```
sqlint32 execute ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

open 函数

用途 打开远程传递（Pass-Through）操作的游标。

语法

```
sqlint32 open ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

fetch 函数

用途 在远程传递（Pass-Through）操作中访存一行。

语法

```
sqlint32 fetch ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

close 函数

用途 关闭远程传递（Pass-Through）操作的游标。

语法

```
sqlint32 close ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『Remote passthru 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 101 页的『C++ API 的操作类』

C++ API 的请求类

下表描述了 C++ API 的每个请求类。

表 238. 请求类

类名	描述
Request	此类包括要由包装器分析和处理的查询片段。
Reply	此类表示可以由包装器处理的查询的一部分。如果包装器使用的成本模型不是缺省成本模型，则可以创建此类的子类。
Request_Exp	此类描述 SQL 表达式节点。SQL 表达式节点可以是一个标题表达式（选择列表）或者是谓词的一部分。
Request_Exp_Type	此类表示 Request_Exp 节点的数据类型描述符。
Request_Constant	此类表示类型为 Request_Exp::constant 的 Request_Exp 节点的数据值。
Predicate_List	此类包括谓词列表，并供 RRC 协议使用。

相关参考:

- 第 135 页的『Request_Exp 类（C++）』
- 第 117 页的『Request 类（C++）』
- 第 143 页的『Request_Constant 类（C++）』
- 第 140 页的『Request_Exp_Type 类（C++）』
- 第 122 页的『Reply 类（C++）』
- 第 146 页的『Predicate_List 类（C++）』

Request 类（C++）

本主题描述 Request 类并提供有关每个成员函数的详细信息。

概述

Request 类包括要由包装器分析和处理的查询片段。

Request 类是 C++ API 的其中一个请求类。

用法 此类从不会被包装器实例化。

文件 sqlqg_request.h

父类 Parsed_Query_Fragment

数据成员

无。

成员函数

下表描述了 Request 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 239. Request 类的成员函数

成员函数	描述
get_number_of_quantifiers	检索此查询的量词数目。
get_number_of_predicates	检索此查询的谓词数目。
get_number_of_head_exp	检索此查询的标题（选择列表）表达式的数目。
get_quantifier_handle	检索此查询的第 *n* 个量词的句柄。
get_predicate_handle	检索此查询的第 *n* 个谓词的句柄。
get_head_exp_handle	检索此查询的第 *n* 个标题表达式的句柄。
get_nickname	检索与指定的句柄相关联的昵称对象。
get_head_exp	检索与指定的句柄相关联的标题表达式。
get_predicate	检索与指定的句柄相关联的谓词表达式。
get_distinct	检索查询的 DISTINCT 标志。

get_number_of_quantifiers 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的量词（昵称和表表达式）的数目。

语法

```
int get_number_of_quantifiers ()
```

输入自变量

无。

输出自变量

无。

返回值 量词的数目。

get_number_of_predicates 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的谓词数目。

语法

```
int get_number_of_predicates ()
```

输入自变量

无。

输出自变量

无。

返回值 谓词的数目。

get_number_of_head_exp 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的标题（选择列表）表达式的数目。

语法

```
int get_number_of_head_exp ()
```

输入自变量

无。

输出自变量

无。

返回值 标题表达式的数目。

get_quantifier_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个量词的句柄。

语法

```
sqlint32 get_quantifier_handle (int  a_quant_pos,
                                int* a_quant_handle)
```

输入自变量

表 240. get_quantifier_handle 成员函数的输入自变量

名称	数据类型	描述
a_quant_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 241. get_quantifier_handle 成员函数的输出自变量

名称	数据类型	描述
a_quant_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_predicate_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个谓词的句柄。

语法

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle )
```

输入自变量

表 242. get_predicate_handle 成员函数的输入自变量

名称	数据类型	描述
a_pred_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 243. get_predicate_handle 成员函数的输出自变量

名称	数据类型	描述
a_pred_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_head_exp_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个标题表达式的句柄。

语法

```
sqlint32 get_head_exp_handle (int  a_head_exp_pos,
                              int* a_head_exp_handle)
```

输入自变量

表 244. get_head_exp_handle 成员函数的输入自变量

名称	数据类型	描述
a_head_exp_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 245. get_head_exp_handle 成员函数的输出自变量

名称	数据类型	描述
a_head_exp_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_nickname 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的昵称对象。

语法

```
sqlint32 get_nickname (int                          a_quant_handle,
                       Unfenced_Generic_Nickname** a_nickname)
```

输入自变量

表 246. get_nickname 成员函数的输入自变量

名称	数据类型	描述
a_quant_handle	int	昵称的句柄。

输出自变量

表 247. get_nickname 成员函数的输出自变量

名称	数据类型	描述
a_nickname	Unfenced_Generic_Nickname**	指向昵称对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_head_exp 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的标题表达式。

语法

```
sqlint32 get_head_exp (int           a_head_exp_handle,
                       Request_Exp** a_head_exp)
```

输入自变量

表 248. get_head_exp 成员函数的输入自变量

名称	数据类型	描述
a_head_exp_handle	int	表达式的句柄。

输出自变量

表 249. get_head_exp 成员函数的输出自变量

名称	数据类型	描述
a_head_exp	Request_Exp**	指向表达式对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_predicate 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的谓词表达式。

语法

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

输入自变量

表 250. get_predicate 成员函数的输入自变量

名称	数据类型	描述
a_pred_handle	int	表达式的句柄。

输出自变量

表 251. get_predicate 成员函数的输出自变量

名称	数据类型	描述
a_pred_exp	Request_Exp**	指向表达式对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_distinct 函数

定义位置

Parsed_Query_Fragment

用途 检索查询的 DISTINCT 标志。

语法

```
int get_distinct ()
```

输入自变量

无。

输出自变量

无。

返回值 Distinct 标志。

相关任务:

- 『Request 类』（《*IBM DB2 Information Integrator 包装器开发者指南*》）

相关参考:

- 第 116 页的『C++ API 的请求类』

Reply 类（C++）

本主题描述 Reply 类并提供有关构造函数和成员函数的详细信息。

概述

Reply 类表示包装器可以处理的查询部分。如果包装器使用的成本模型不是缺省模型，则可以创建此类的子类。

Reply 类是 C++ API 的其中一个请求类。

用法 此类是由 Unfenced_Generic_Server 类的 create_reply() 方法来实例化的。如果包装器实现 Reply 类的子类，则 create_reply() 方法在 Unfenced_Generic_Server 的特定于包装器的子类中会被覆盖。

文件 sqlqg_request.h

父类 Parsed_Query_Fragment

数据成员

无。

构造函数和成员函数

下列各表描述了 Reply 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 252. Reply 类的构造函数

构造函数	描述
Reply	构造一个空的应答对象。

表 253. Reply 类的成员函数

成员函数	描述
get_number_of_quantifiers	检索此查询的量词数目。
get_number_of_predicates	检索此查询的谓词数目。
get_number_of_head_exp	检索此查询的标题（选择列表）表达式的数目。
get_quantifier_handle	检索此查询的第 *n* 个量词的句柄。
get_predicate_handle	检索此查询的第 *n* 个谓词的句柄。
get_head_exp_handle	检索此查询的第 *n* 个标题表达式的句柄。
get_nickname	检索与指定的句柄相关联的昵称对象。
get_head_exp	检索与指定的句柄相关联的标题表达式。
get_predicate	检索与指定的句柄相关联的谓词表达式。
get_distinct	检索查询的 DISTINCT 标志。
set_distinct	为查询设置 DISTINCT 标志。

表 253. Reply 类的成员函数 (续)

成员函数	描述
add_head_exp	将标题表达式添加至应答。
add_predicate	将谓词表达式添加至应答。
add_quantifier	将量词添加至应答。
add_order_entry	将 ORDER BY 规范添加至应答。
get_number_of_order_entries	检索应答的 ORDER BY 条目数。
get_order_entry	检索特定 ORDER BY 条目。
get_exec_desc	检索与应答相关联的执行描述符。
set_exec_desc	将执行描述符存储在应答中。
get_parameter_order	检索参数句柄的列表。
基数	检索查询片段的基数。
total_cost	检索查询片段的总成本。
re_exec_cost	再次运行查询片段的成本。
first_tuple_cost	检索访存第一个元组的成本。
all_costs	一次检索所有成本值。
next	检索指向一系列应答中的下一个应答的指针。
set_next_reply	将新应答链接至当前应答。
get_server	检索指向拥有此应答的服务器的指针。

Reply 构造函数

用途 构造一个空的应答对象。

语法

```
Reply (Request*                 a_rq,
       Unfenced_Generic_Server* a_server,
       sqlint32*                a_rc)
```

输入自变量

表 254. Reply 构造函数的输入自变量

名称	数据类型	描述
a_rq	Request*	派生出此应答的请求。
a_server	Unfenced_Generic_Server*	用于请求或应答协议的服务器。

输出自变量

表 255. Reply 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	返回码；如果值为 0，则指示成功。

返回值 无。

get_number_of_quantifiers 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的量词（昵称和表表达式）的数目。

语法

```
int get_number_of_quantifiers ()
```

输入自变量

无。

输出自变量

无。

返回值 量词的数目。

get_number_of_predicates 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的谓词数目。

语法

```
int get_number_of_predicates ()
```

输入自变量

无。

输出自变量

无。

返回值 谓词的数目。

get_number_of_head_exp 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的标题（选择列表）表达式的数目。

语法

```
int get_number_of_head_exp ()
```

输入自变量

无。

输出自变量

无。

返回值 标题表达式的数目。

get_quantifier_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个量词的句柄。

语法

```
sqlint32 get_quantifier_handle (int  a_quant_pos,
                                 int* a_quant_handle)
```

输入自变量

表 256. get_quantifier_handle 成员函数的输入自变量

名称	数据类型	描述
a_quant_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 257. get_quantifier_handle 成员函数的输出自变量

名称	数据类型	描述
a_quant_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_predicate_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个谓词的句柄。

语法

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle)
```

输入自变量

表 258. get_predicate_handle 成员函数的输入自变量

名称	数据类型	描述
a_pred_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 259. get_predicate_handle 成员函数的输出自变量

名称	数据类型	描述
a_pred_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_head_exp_handle 函数

定义位置

Parsed_Query_Fragment

用途 检索此查询的第 *n* 个标题表达式的句柄。

语法

```
sqlint32 get_head_exp_handle (int  a_head_exp_pos,
                              int* a_head_exp_handle)
```

输入自变量

表 260. get_head_exp_handle 成员函数的输入自变量

名称	数据类型	描述
a_head_exp_pos	int	列表中句柄的位置（从 1 开始）。

输出自变量

表 261. get_head_exp_handle 成员函数的输出自变量

名称	数据类型	描述
a_head_exp_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_nickname 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的昵称对象。

语法

```
sqlint32 get_nickname (int                          a_quant_handle,
                       Unfenced_Generic_Nickname** a_nickname)
```

输入自变量

表 262. get_nickname 成员函数的输入自变量

名称	数据类型	描述
a_quant_handle	int	昵称的句柄。

输出自变量

表 263. get_nickname 成员函数的输出自变量

名称	数据类型	描述
a_nickname	Unfenced_Generic_Nickname**	指向昵称对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_head_exp 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的标题表达式。

语法

```
sqlint32 get_head_exp (int a          a_head_exp_handle,
                       Request_Exp** a_head_exp)
```

输入自变量

表 264. get_head_exp 成员函数的输入自变量

名称	数据类型	描述
a_head_exp_handle	int	表达式的句柄。

输出自变量

表 265. get_head_exp 成员函数的输出自变量

名称	数据类型	描述
a_head_exp	Request_Exp**	指向表达式对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_predicate 函数

定义位置

Parsed_Query_Fragment

用途 检索与指定的句柄相关联的谓词表达式。

语法

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

输入自变量

表 266. get_predicate 成员函数的输入自变量

名称	数据类型	描述
a_pred_handle	int	表达式的句柄。

输出自变量

表 267. get_predicate 成员函数的输出自变量

名称	数据类型	描述
a_pred_exp	Request_Exp**	指向表达式对象的指针。

返回值 返回码。如果值为 0，则指示成功。

get_distinct 函数

定义位置

Parsed_Query_Fragment

用途 检索查询的 DISTINCT 标志。

语法

```
int get_distinct ()
```

输入自变量

无。

输出自变量

无。

返回值 Distinct 标志。

set_distinct 函数

定义位置

Parsed_Query_Fragment

用途 为查询设置 DISTINCT 标志。

语法

```
void set_distinct (int a_distinct)
```

输入自变量

表 268. set_distinct 成员函数的输入自变量

名称	数据类型	描述
a_distinct	int	Distinct 标志（1 表示 DISTINCT；0 表示不是 DISTINCT）。

输出自变量

无。

返回值 无。

add_head_exp 函数

用途 将标题表达式添加至应答。

语法

```
sqlint32 add_head_exp (int a_head_exp_handle)
```

输入自变量

表 269. add_head_exp 成员函数的输入自变量

名称	数据类型	描述
a_head_exp_handle	int	表达式的句柄。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

add_predicate 函数

用途 将谓词表达式添加至应答。

语法

```
sqlint32 add_predicate_exp (int a_pred_handle)
```

输入自变量

表 270. add_predicate 成员函数的输入自变量

名称	数据类型	描述
a_pred_handle	int	表达式的句柄。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

add_quantifier 函数

用途 将量词添加至应答。

语法

```
sqlint32 add_quantifier (int a_quant_handle)
```

输入自变量

表 271. add_quantifier 成员函数的输入自变量

名称	数据类型	描述
a_quant_handle	int	量词的句柄。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

add_order_entry 函数

用途 将 ORDER BY 规范添加至应答。

语法

```
sqlint32 add_order_entry (int                    a_gindex,
                          Reply::order_direction a_direction)
```

输入自变量

表 272. add_order_entry 成员函数的输入自变量

名称	数据类型	描述
a_gindex	int	用于排序的标题表达式索引（不是句柄）。
a_direction	Reply_order_direction	排序的方向（升序或降序）。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_number_of_order_entries 函数

用途 检索应答的 ORDER BY 条目数。

语法

```
int get_number_of_order_entries ()
```

输入自变量

无。

输出自变量

无。

返回值 ORDER BY 条目的数目。

get_order_entry 函数

用途 检索特定 ORDER BY 条目。

语法

```
get_order_entry (int                     a_pos,
                 int*                    a_gindexP,
                 Reply::order_direction* a_direction)
```

输入自变量

表 273. get_order_entry 成员函数的输入自变量

名称	数据类型	描述
a_pos	int	顺序条目的位置（从 1 开始）。

输出自变量

表 274. get_order_entry 成员函数的输出自变量

名称	数据类型	描述
a_gindexP	int*	ORDER BY 表达式的标题表达式索引。
a_direction	Reply::order_direction*	排序的方向。

返回值 返回码。如果值为 0，则指示成功。

get_exec_desc 函数

用途 检索与应答相关联的执行描述符。

语法

```
void get_exec_desc (void** a_exec_desc,
                    int*   a_exec_desc_size)
```

输入自变量

无。

输出自变量

表 275. get_exec_desc 成员函数的输出自变量

名称	数据类型	描述
a_exec_desc	void**	指向执行描述符的指针。
a_exec_desc_size	int*	执行描述符的大小。

返回值 无。

set_exec_desc 函数

用途 将执行描述符存储在应答中。

用法 必须使用 Wrapper_Utillities::allocate 来分配用于执行描述符的存储器。

语法

```
void set_exec_desc (void** a_exec_desc,
                    int*   a_exec_desc_size)
```

输入自变量

表 276. set_exec_desc 成员函数的输入自变量

名称	数据类型	描述
a_exec_desc	void**	指向执行描述符的指针。
a_exec_desc_size	int*	执行描述符的大小。

输出自变量

无。

返回值 无。

get_parameter_order 函数

用途 检索参数句柄的列表。列表的顺序对应于 Remote_Operation 对象中的参数值的顺序。

语法

```
sqlint32 get_parameter_order (int*  a_number_of_params,
                              int** a_param_handle_array)
```

输入自变量

无。

输出自变量

表 277. get_parameter_order 成员函数的输出自变量

名称	数据类型	描述
a_number_of_params	int*	参数句柄的数目。
a_param_handle_array	int**	参数句柄的数组。

返回值 返回码。如果值为 0，则指示成功。

基数函数

用途 检索查询片段的基数。

语法

```
sqlint32 cardinality (float* a_cardinality)
```

输入自变量

无。

输出自变量

表 278. cardinality 成员函数的输出自变量

名称	数据类型	描述
a_cardinality	float*	基数。

返回值 返回码。如果值为 0，则指示成功。

total_cost 函数

用途 检索查询片段的总成本。

语法

```
sqlint32 total_cost (float* a_total_cost)
```

输入自变量

无。

输出自变量

表 279. total_cost 成员函数的输出自变量

名称	数据类型	描述
a_total_cost	float*	成本。

返回值 返回码。如果值为 0，则指示成功。

re_exec_cost 函数

用途 再次运行查询片段的成本。

语法

```
sqlint32 re_exec_cost (float* a_re_exec_cost)
```

输入自变量

无。

输出自变量

表 280. re_exec_cost 成员函数的输出自变量

名称	数据类型	描述
a_re_exec_cost	float*	成本。

返回值 返回码。如果值为 0，则指示成功。

first_tuple_cost 函数

用途 检索访存第一个元组的成本。

语法

```
sqlint32 first_tuple_cost (float* a_first_tuple_cost)
```

输入自变量

无。

输出自变量

表 281. first_tuple_cost 成员函数的输出自变量

名称	数据类型	描述
a_first_tuple_cost	float*	成本。

返回值 返回码。如果值为 0，则指示成功。

all_costs 函数

用途 一次检索所有成本值。

语法

```
sqlint32 all_costs (float* a_total_cost,
                    float* a_first_tuple_cost,
                    float* a_re_exec_cost)
```

输入自变量

无。

输出自变量

表 282. all_costs 成员函数的输出自变量

名称	数据类型	描述
a_total_cost	float*	总成本。
a_first_tuple_cost	float*	第一个元组成本。
a_re_exec_cost	float*	重新执行成本。

返回值 返回码。如果值为 0，则指示成功。

next 函数

用途 检索指向一系列应答中的下一个应答的指针。

语法

```
Reply* next ()
```

输入自变量

无。

输出自变量

无。

返回值 指向下一个应答的指针。如果是在应答链的末尾，则该值为空。

set_next_reply 函数

用途 将新应答链接至当前应答。

用法 包装器验证它是否将应答添加至应答链的末尾。否则，可能会发生内存泄漏。

语法

```
void set_next_reply (Reply* a_next_reply)
```

输入自变量

表 283. set_next_reply 成员函数的输入自变量

名称	数据类型	描述
a_next_reply	Reply*	要添加至应答链的应答。

输出自变量

无。

返回值 无。

get_server 函数

用途 检索指向拥有此应答的服务器的指针。

语法

```
Unfenced_Generic_server*  get_server ()
```

输入自变量

无。

输出自变量

无。

返回值 指向服务器对象的指针。

相关任务:

- 『Reply 类』（《*IBM DB2 Information Integrator 包装器开发者指南*》）

相关参考:

- 第 116 页的『C++ API 的请求类』

Request_Exp 类（C++）

本主题描述 Request_Exp 类并提供有关每个成员函数的详细信息。

概述

Request_Exp 类表示表达式树中的一个节点。此节点可以是列引用、常量值、主机参数或运算符。

Request_Exp 类是 C++ API 的其中一个请求类。

用法 此类从不会被包装器实例化。

文件 sqlqg_request.h

数据成员

无。

类型 Request_Exp::kind

类型 enum

值 badkind、column、unbound、constant 和 oper

成员函数

下表描述了 Request_Exp 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 284. Request_Exp 类的成员函数

成员函数	描述
get_kind	检索表达式节点的类型。
get_data_type	检索与表达式相关联的数据类型。
get_parent	检索指向当前节点的父代表达式节点的指针。
get_next_child	检索指向同一父代的下一个子代的指针。
get_handle	检索表达式节点的句柄。
get_column_name	检索列表达式节点的列名。
get_column_quantifier_handle	检索列量词的句柄。
get_value	检索常量表达式节点的值。
get_first_child	检索运算符表达式节点的第一个子代。
get_number_of_children	检索运算符表达式节点的子代数。
get_token	检索运算符表达式节点的标记。
get_signature	检索运算符表达式节点的所有特征符。

get_kind 函数

用途 检索表达式节点的类型。

用法 此成员函数的附加语法是 sqlint32 get_kind()。当 Request 或 Reply 不可用时，在 Unfenced_Server::get_selectivity() 表达式中使用此语法。get_kind() 语法不区分列引用和不受限制的引用。

语法

```
sqlint32 get_kind (Parsed_Query_Fragment* a_query_fragment,
                   kind*                  a_kind )
```

输入自变量

表 285. get_kind 成员函数的输入自变量

名称	数据类型	描述
a_query_fragment	Parsed_Query_Fragment*	此表达式所属于的 Request 或 Reply。

输出自变量

表 286. get_kind 成员函数的输出自变量

名称	数据类型	描述
a_kind	kind*	表达式节点的类型。

返回值 返回码。如果值为 0，则指示成功。

get_data_type 函数

用途 检索与表达式相关联的数据类型。

语法

```
sqlint32 get_data_type (Request_Exp_Type** a_new_type)
```

输入自变量

无。

输出自变量

表 287. get_data_type 成员函数的输出自变量

名称	数据类型	描述
a_new_type	Request_Exp_Type**	指向描述符的类型的指针。

返回值 返回码。如果值为 0，则指示成功。

get_parent 函数

用途 检索指向当前节点的父代表达式节点的指针。

语法

```
Request_Exp* get_parent ()
```

输入自变量

无。

输出自变量

无。

返回值 指向父节点的指针。如果这是一个顶级节点，则该值为空。

get_next_child 函数

用途 检索指向同一父代的下一个子代（即，当前节点的下一个兄弟节点）的指针。

语法

```
Request_Exp* get_next_child ()
```

输入自变量

无。

输出自变量

无。

返回值 指向下一个子代的指针。如果没有兄弟节点，则该值为空。

get_handle 函数

用途 检索表达式节点的句柄。

语法

```
sqlint32 get_handle (int* a_handle)
```

输入自变量

无。

输出自变量

表 288. get_handle 成员函数的输出自变量

名称	数据类型	描述
a_handle	int*	句柄。

返回值 返回码。如果值为 0，则指示成功。

get_column_name 函数

用途 检索列表达式节点的列名。

用法 仅当表达式类型为 Request_Exp::column 时，此方法才有效。

语法

```
sqlint32 get_column_name (char** a_col_name,
                          int*   a_name_length)
```

输入自变量

无。

输出自变量

表 289. get_column_name 成员函数的输出自变量

名称	数据类型	描述
a_col_name	char**	指向列名（不是以 null 结束）的指针。
a_name_length	int*	列名的长度。

返回值 返回码。如果值为 0，则指示成功。

get_column_quantifier_handle 函数

用途 检索列量词（表或表表达式）的句柄。

用法 仅当表达式类型为 Request_Exp::column 时，此方法才有效。

语法

```
sqlint32 get_column_quantifier_handle (int* a_quant_handle)
```

输入自变量

无。

输出自变量

表 290. get_column_quantifier_handle 成员函数的输出自变量

名称	数据类型	描述
a_quant_handle	int*	量词句柄。

返回值 返回码。如果值为 0，则指示成功。

get_value 函数

用途 检索常量表达式节点的值。

用法 仅当表达式类型为 Request_Exp::constant 时此方法才有效。

语法

```
sqlint32 get_value (Request_Constant** a_constant_value)
```

输入自变量

无。

输出自变量

表 291. get_value 成员函数的输出自变量

名称	数据类型	描述
a_constant_value	Request_Constant**	指向常量值的指针。

返回值 返回码。如果值为 0，则指示成功。

get_first_child 函数

用途 检索运算符表达式节点的第一个子代。

用法 仅当表达式类型为 Request_Exp::oper 时此方法才有效。

语法

```
Request_Exp* get_first_child ()
```

输入自变量

无。

输出自变量

无。

返回值 指向第一个子代的指针。如果没有子代，则该值为空。

get_number_of_children 函数

用途 检索运算符表达式节点的子代数。

用法 仅当表达式类型为 Request_Exp::oper 时此方法才有效。

语法

```
int get_number_of_children ()
```

输入自变量

无。

输出自变量

无。

返回值 子代的数目。

get_token 函数

用途 检索运算符表达式节点的标记（函数名）。

用法 仅当表达式类型为 Request_Exp::oper 时此方法才有效。

语法

```
sqlint32 get_token (char** a_operator_token,
                    int*   a_token_length )
```

输入自变量

无。

输出自变量

表 292. get_token 成员函数的输出自变量

名称	数据类型	描述
a_operator_token	char**	指向标记（不是以 null 结束）的指针。
a_token_length	int*	标记的长度。

返回值 返回码。如果值为 0，则指示成功。

get_signature 函数

用途 检索运算符表达式节点的所有特征符。

用法 仅当表达式类型为 Request_Exp::oper 时此方法才有效。

语法

```
sqlint32 get_signature (char** a_signature)
```

输入自变量

无。

输出自变量

表 293. get_signature 成员函数的输出自变量

名称	数据类型	描述
a_signature	char**	指向以 null 结束的特征符的指针。

返回值 返回码。如果值为 0，则指示成功。

相关参考:

- 『Request expression 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）
- 第 116 页的『C++ API 的请求类』

Request_Exp_Type 类（C++）

本主题描述 Request_Exp_Type 类并提供有关每个成员函数的详细信息。

概述

Request_Exp_Type 类表示 Request_Exp 节点的数据类型描述符。

Request_Exp_Type 类是 C++ API 的其中一个请求类。

用法 此类从不会被包装器实例化。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

成员函数

下表描述了 Request_Exp_Type 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 294. Request_Exp_Type 类的成员函数

成员函数	描述
get_for_bit_data	检索 FOR BIT DATA 标志。
get_null_indicator	检索可空指示符。
get_data_type	检索数据类型。

表 294. Request_Exp_Type 类的成员函数 (续)

成员函数	描述
get_maximum_length	检索类型的最大长度。
get_precision	获取数字类型的精度。
get_scale	获取数字类型的小数位。
get_codepage	检索字符类型的代码页。

get_for_bit_data 函数

用途 检索数据的 FOR BIT DATA 标志。

语法

```
unsigned char get_for_bit_data ()
```

输入自变量

无。

输出自变量

无。

返回值 FOR BIT DATA 标志（Y 或 N）。

get_null_indicator 函数

用途 检索可空指示符。

语法

```
short get_null_indicator ()
```

输入自变量

无。

输出自变量

无。

返回值 空指示符标志，即 SQL_NULLABLE 或 SQL_NO_NULLS。

get_data_type 函数

用途 检索数据类型。

语法

```
short get_data_type ()
```

输入自变量

无。

输出自变量

无。

返回值 数据类型（SQL_TYP_xxx）。

get_maximum_length 函数

用途 检索类型的最大长度。

语法

```
int get_maximum_length ()
```

输入自变量

无。

输出自变量

无。

返回值 类型的最大长度。

get_precision 函数

用途 获取数字类型的精度。

语法

```
unsigned char get_precision ()
```

输入自变量

无。

输出自变量

无。

返回值 数字精度。

get_scale 函数

用途 获取数字类型的小数位。

语法

```
unsigned char get_scale ()
```

输入自变量

无。

输出自变量

无。

返回值 数字小数位。

get_codepage 函数

用途 检索字符类型的代码页。

语法

```
unsigned short get_codepage ()
```

输入自变量

无。

输出自变量

无。

返回值 代码页。

相关参考:

- 『Request expression type 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）
- 第 116 页的『C++ API 的请求类』

Request_Constant 类（C++）

本主题描述 Request_Constant 类并提供有关每个成员函数的详细信息。

概述

Request_Constant 类表示类型为 Request_Exp::constant 的 Request_Exp 节点的数据值。

Request_Constant 类是 C++ API 的其中一个请求类。

用法 此类从不会被包装器实例化。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

成员函数

下表描述了 Request_Constant 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 295. Request_Constant 类的成员函数

成员函数	描述
get_actual_length	检索数据值的实际长度。
get_data	检索指向数据值的指针。
is_data_null	指示数据值是否是空的。
get_for_bit_data	检索 FOR BIT DATA 标志。
get_null_indicator	检索可空指示符。
get_data_type	检索数据类型。
get_maximum_length	检索类型的最大长度。
get_precision	获取数字类型的精度。
get_scale	获取数字类型的小数位。
get_codepage	检索字符类型的代码页。

get_actual_length 函数

用途 检索数据值的实际长度。

语法

```
int get_actual_length ()
```

输入自变量
无。

输出自变量
无。

返回值 实际数据长度。

get_data 函数

用途 检索指向数据值的指针。

语法

```
unsigned char* get_data ()
```

输入自变量
无。

输出自变量
无。

返回值 指向数据值的指针。

is_data_null 函数

用途 指示数据值是否是空的。

语法

```
sqlint32 is_data_null ()
```

输入自变量
无。

输出自变量
无。

返回值 空指示（TRUE 或 FALSE）。

get_for_bit_data 函数

用途 检索数据的 FOR BIT DATA 标志。

语法

```
unsigned char get_for_bit_data ()
```

输入自变量
无。

输出自变量
无。

返回值 FOR BIT DATA 标志（Y 或 N）。

get_null_indicator 函数

用途 检索可空指示符。

语法

```
short get_null_indicator ()
```

输入自变量

无。

输出自变量

无。

返回值 空指示符标志，即 SQL_NULLABLE 或 SQL_NO_NULLS。

get_data_type 函数

用途 检索数据类型。

语法

```
short get_data_type ()
```

输入自变量

无。

输出自变量

无。

返回值 数据类型（SQL_TYP_xxx）。有关 SQL_TYP_xxx 符号，请参阅 sql.h 头文件。

get_maximum_length 函数

用途 检索类型的最大长度。

语法

```
int get_maximum_length ()
```

输入自变量

无。

输出自变量

无。

返回值 类型的最大长度。

get_precision 函数

用途 获取数字类型的精度。

语法

```
unsigned char get_precision ()
```

输入自变量

无。

输出自变量
无。

返回值 数字精度。

get_scale 函数

用途 获取数字类型的小数位。

语法

```
unsigned char get_scale ()
```

输入自变量
无。

输出自变量
无。

返回值 数字小数位。

get_codepage 函数

用途 检索字符类型的代码页。

语法

```
unsigned short get_codepage ()
```

输入自变量
无。

输出自变量
无。

返回值 代码页。

相关参考:

- 『Request constant 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）
- 第 116 页的『C++ API 的请求类』

Predicate_List 类（C++）

本主题描述 Predicate_List 类并提供有关每个成员函数的详细信息。

概述

Predicate_List 类包括谓词列表，并供 RRC 协议使用。

Predicate_List 类是 C++ API 的其中一个请求类。

用法 包装器不会实例化此类，除非定制成本模型的代码为特定的谓词列表调用 get_selectivity() 方法。

文件 sqlqg_request.h

数据成员

无。

成员函数

下表描述了 Predicate_List 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 296. Predicate_List 类的成员函数

成员函数	描述
create_empty_predicate_list	实例化空的谓词列表。
create_and_copy_predicate_list	创建一个包含特定应答的所有谓词的谓词列表。
get_number_of_predicates	检索列表中的谓词数目。
get_predicate_handle	检索列表中的特定位置的谓词句柄。
get_predicate	检索表达式句柄的谓词表达式。
get_number_of_applied_predicates	获取已经应用的谓词数目。
get_applied_predicate_handle	检索列表中的指定位置已应用的谓词句柄。
get_applied_predicate	检索表达式句柄的已应用的谓词表达式。
add_predicate	将谓词添加到列表中。
add_applied_predicate	将谓词表达式添加到已应用的谓词列表中。

create_empty_predicate_list 函数

用途 实例化空的谓词列表。

语法

```
static sqlint32 create_empty_predicate_list
   (Reply*           a_reply,
    Predicate_List** a_pred_list )
```

输入自变量

表 297. create_empty_predicate_list 成员函数的输入自变量

名称	数据类型	描述
a_reply	Reply*	拥有谓词的应答。

输出自变量

表 298. create_empty_predicate_list 成员函数的输出自变量

名称	数据类型	描述
a_pred_list	Predicate_List**	指向新的谓词列表的指针。

返回值 返回码。如果值为 0，则指示成功。

create_and_copy_predicate_list 函数

用途 创建一个包含特定应答的所有谓词的谓词列表。

语法

```
static sqlint32 create_and_copy_predicate_list
   (Reply*           a_reply,
    Predicate_List** a_pred_list)
```

输入自变量

表 299. create_and_copy_predicate_list 成员函数的输入自变量

名称	数据类型	描述
a_reply	Reply*	拥有谓词的应答。

输出自变量

表 300. create_and_copy_predicate_list 成员函数的输出自变量

名称	数据类型	描述
a_pred_list	Predicate_List**	指向新的谓词列表的指针。

返回值 返回码。如果值为 0，则指示成功。

get_number_of_predicates 函数

用途 检索列表中的谓词数目。

语法

```
int get_number_of_predicates ()
```

输入自变量

无。

输出自变量

无。

返回值 谓词的数目。

get_predicate_handle 函数

用途 检索列表中的特定位置的谓词句柄。

语法

```
sqlint32 get_predicate_handle (int  a_pred_pos,
                               int* a_pred_handle)
```

输入自变量

表 301. get_predicate_handle 成员函数的输入自变量

名称	数据类型	描述
a_pred_pos	int	列表中谓词的位置（从 1 开始）。

输出自变量

表 302. get_predicate_handle 成员函数的输出自变量

名称	数据类型	描述
a_pred_handle	int*	谓词表达式句柄。

返回值 返回码。如果值为 0，则指示成功。

get_predicate 函数

用途 检索表达式句柄的谓词表达式。

语法

```
sqlint32 get_predicate (int           a_pred_handle,
                        Request_Exp** a_pred_exp)
```

输入自变量

表 303. get_predicate 成员函数的输入自变量

名称	数据类型	描述
a_pred_handle	int	谓词表达式句柄。

输出自变量

表 304. get_predicate 成员函数的输出自变量

名称	数据类型	描述
a_pred_exp	Request_Exp**	指向表达式节点的指针。

返回值 返回码。如果值为 0，则指示成功。

get_number_of_applied_predicates 函数

用途 获取已经应用的谓词数目。

语法

```
int get_number_of_applied_predicates ()
```

输入自变量

无。

输出自变量

无。

返回值 已应用的谓词数目。

get_applied_predicate_handle 函数

用途 检索列表中的指定位置已应用的谓词句柄。

语法

```
sqlint32 get_applied_predicate_handle (int  a_applied_pred_pos,
                                       int* a_applied_pred_handle)
```

输入自变量

表 305. get_applied_predicate_handle 成员函数的输入自变量

名称	数据类型	描述
a_applied_pred_pos	int	列表中谓词的位置（从 1 开始）。

输出自变量

表 306. get_applied_predicate_handle 成员函数的输出自变量

名称	数据类型	描述
a_applied_pred_handle	int*	已应用的谓词表达式句柄。

返回值 返回码。如果值为 0，则指示成功。

get_applied_predicate 函数

用途 检索表达式句柄的已应用的谓词表达式。

语法

```
sqlint32 get_applied_predicate (int           a_applied_pred_handle,
                                Request_Exp** a_applied_pred_exp)
```

输入自变量

表 307. get_applied_predicate 成员函数的输入自变量

名称	数据类型	描述
a_applied_pred_handle	int	已应用的谓词表达式句柄。

输出自变量

表 308. get_applied_predicate 成员函数的输出自变量

名称	数据类型	描述
a_applied_pred_exp	Request_Exp**	指向表达式节点的指针。

返回值 返回码。如果值为 0，则指示成功。

add_predicate 函数

用途 将谓词添加到列表中。

语法

```
sqlint32 add_predicate (int a_pred_handle)
```

输入自变量

表 309. add_predicate 成员函数的输入自变量

名称	数据类型	描述
a_pred_handle	int	谓词表达式的句柄。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

add_applied_predicate 函数

用途 将谓词表达式添加到已应用的谓词列表中。

语法

```
sqlint32 add_applied_predicate (int a_applied_pred_handle)
```

输入自变量

表 310. add_applied_predicate 成员函数的输入自变量

名称	数据类型	描述
a_applied_pred_handle	int	已应用的谓词表达式的句柄。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

相关任务:

- 『Predicate list 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 116 页的『C++ API 的请求类』

C++ API 的数据类

下表描述了 C++ API 的每个数据类。

表 311. 数据类

类名	描述
Runtime_Data_Desc	包括数据项描述的类。
Runtime_Data_Desc_List	Runtime_Data_Desc 的列表。
Runtime_Data	包括在包装器和查询网关接口之间传递的数据值的类。此数据值可以为列数据，它在结果或在查询参数中。
Runtime_Data_List	指向 Runtime_Data 对象的指针的列表。

相关参考:

- 第 152 页的『Runtime_Data_Desc 类（C++）』

- 第 155 页的『Runtime_Data_Desc_List 类（C++）』
- 第 158 页的『Runtime_Data 类（C++）』
- 第 164 页的『Runtime_Data_List 类（C++）』

Runtime_Data_Desc 类（C++）

本主题描述 Runtime_Data_Desc 类并提供每个成员函数的详细信息。

概述

Runtime_Data_Desc 类包括数据项的描述。

Runtime_Data_Desc 类是 C++ API 的数据类之一。

用法 此类由 DB2 查询网关实例化，并且从不由包装器划分其子类。此类可以在准备处理传递（Pass-Through）操作期间由包装器实例化。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

构造函数和成员函数

下列各表描述 Runtime_Data_Desc 类的构造函数和成员函数。在这些表的后面更详细地描述了构造函数和成员函数。

表 312. Runtime_Data_Desc 类的构造函数

构造函数	描述
Runtime_Data_Desc	构造 Runtime_Data_Desc 的实例。

表 313. Runtime_Data_Desc 类的成员函数

成员函数	描述
get_for_bit_data	检索数据的 FOR BIT DATA 标志。
get_null_indicator	检索可空指示符。
get_data_type	检索数据类型。
get_maximum_length	检索类型的最大长度。
get_precision	获取数字类型的精度。
get_scale	获取数字类型的小数位。
get_codepage	检索字符类型的代码页。

Runtime_Data_Desc 构造函数

用途 构造 Runtime_Data_Desc 的实例。当响应传递（Pass-Through）会话的准备或描述操作而实例化 Runtime_Data_Desc 类时，使用此构造函数。

语法

```
Runtime_Data_Desc (sqlint32*      a_rc,
                   short          a_type,
                   int            a_max_length,
                   short          a_codepage,
```

```
                                short          a_null_ind,
                                unsigned char  a_data_precision=0,
                                unsigned char  a_data_scale=0,
                                sqluint8*      a_data_name=NULL,
                                short          a_name_length=0,
                                short          a_remote_type=0)
```

输入自变量

表 314. Runtime_Data_Desc 构造函数的输入自变量

名称	数据类型	描述
a_type	short	数据类型。使用在 sql.h 头文件中定义的 SQL_TYP_xxx 值。
a_max_length	int	数据的最大长度。
a_codepage	short	字符数据的代码页。
a_null_ind	short	空指示符。
a_data_precision	unsigned char	数字数据的精度。
a_data_scale	unsigned char	数字数据类型的小数位。
a_data_name	sqluint8*	数据项的名称。
a_name_length	short	名称的长度。
a_remote_type	short	远程类型代码。

输出自变量

表 315. Runtime_Data_Desc 构造函数的输出自变量

名称	数据类型	描述
a_rc	sqlint32*	指向返回码值的指针；如果值为 0，则指示成功。

get_for_bit_data 函数

用途 检索数据的 FOR BIT DATA 标志。

语法

```
unsigned char get_for_bit_data ()
```

输入自变量

无。

输出自变量

无。

返回值 FOR BIT DATA 标志（Y 或 N）。

get_null_indicator 函数

用途 检索可空指示符。

语法

```
short get_null_indicator ()
```

输入自变量
无。

输出自变量
无。

返回值 空指示符标志，即 SQL_NULLABLE 或 SQL_NO_NULLS。

get_data_type 函数

用途 检索数据类型。

语法

```
short get_data_type ()
```

输入自变量
无。

输出自变量
无。

返回值 数据类型（SQL_TYP_xxx）。请参阅 SQL_TYP_xxx 符号的 sql.h 头文件。

get_maximum_length 函数

用途 检索类型的最大长度。

语法

```
int get_maximum_length ()
```

输入自变量
无。

输出自变量
无。

返回值 类型的最大长度。

get_precision 函数

用途 获取数字类型的精度。

语法

```
unsigned char get_precision ()
```

输入自变量
无。

输出自变量
无。

返回值 数字精度。

get_scale 函数

用途 获取数字类型的小数位。

语法

```
unsigned char get_scale ()
```

输入自变量

无。

输出自变量

无。

返回值 数字小数位。

get_codepage 函数

用途 检索字符类型的代码页。

语法

```
unsigned short get_codepage ()
```

输入自变量

无。

输出自变量

无。

返回值 代码页。

相关任务:

- 『运行时数据描述类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 第 151 页的『C++ API 的数据类』

Runtime_Data_Desc_List 类（C++）

本主题描述 Runtime_Data_Desc_List 类并提供每个成员函数的详细信息。

概述

Runtime_Data_Desc_List 类提供 Runtime_Data_Desc 类的列表。

Runtime_Data_Desc_List 类是 C++ API 的其中一个数据类。

用法 此类由 DB2 查询网关实例化，但从不由包装器划分其子类。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

成员函数

下表描述 Runtime_Data_Desc_List 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 316. Runtime_Data_Desc_List 类的成员函数

成员函数	描述
get_number_of_values	检索值的数目。
set_number_of_values	设置列表中值的数目。
get_ith_value	检索第 *i* 个数据描述符。
set_ith_value	在列表的第 *i* 个位置存储 Runtime_Data_Desc 指针。
operator[]	通过使用下标表示法检索列表中第 *i* 个条目。

get_number_of_values 函数

用途 检索值的数目。

语法

```
int get_number_of_values ()
```

输入自变量

无。

输出自变量

无。

返回值 值的数目。

set_number_of_values 函数

用途 设置列表中值的数目。此成员函数可以通过将数目设置为 0 来清空列表。此成员函数还可以加长或缩短列表。

语法

```
sqlint32 set_number_of_values (int a_count)
```

输入自变量

表 317. set_number_of_values 成员函数的输入自变量

名称	数据类型	描述
a_count	int	值的数目。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_ith_value 函数

用途 检索第 *i* 个数据描述符。

语法

```
Runtime_Data_Desc* get_ith_value (int a_index)
```

输入自变量

表 318. get_ith_value 成员函数的输入自变量

名称	数据类型	描述
a_index	int	在列表中建立索引（从 0 开始）。

输出自变量

无。

返回值 指向第 *i* 个值（或空值）的指针。

set_ith_value 函数

用途 在列表的第 *i* 个位置存储 Runtime_Data_Desc 指针。

语法

```
sqlint32 set_ith_value (Runtime_Data_Desc* a_desc,
                        int                a_index)
```

输入自变量

表 319. set_ith_value 成员函数的输入自变量

名称	数据类型	描述
a_desc	Runtime_Data_Desc*	描述符指针。
a_index	int	索引（从 0 开始）。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

operator[] 函数

用途 通过使用下标表示法检索列表中第 *i* 个条目。

语法

```
Runtime_Data_Desc* operator[] (int a_index)
```

输入自变量

表 320. operator[] 成员函数的输入自变量

名称	数据类型	描述
a_index	int	在列表中建立索引（从 0 开始）。

输出自变量

无。

返回值 第 *i* 个条目。

相关参考:

- 第 151 页的『C++ API 的数据类』

Runtime_Data 类（C++）

本主题描述 Runtime_Data 类并提供每个成员函数的详细信息。

概述

Runtime_Data 类包括在包装器和查询网关接口之间传递的数据值。此数据值可以为列数据，它在结果或在查询参数中。

Runtime_Data 类是 C++ API 的其中一个数据类。

用法 此类由 DB2 查询网关实例化，并且从不由包装器划分其子类。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

成员函数

下表描述 Runtime_Data 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 321. Runtime_Data 类的成员函数

成员函数	描述
get_actual_length	检索数据值的实际长度。
set_actual_length	设置数据值的实际长度。
get_data	检索指向数据值的指针。
set_data	将信息复制到数据值中。
is_data_null	指示数据值是否为空。
set_data_null	将数据值标记为空。
clear_null_indicator	复位数据值的空指示符。
set_friendly_div_by_0	指示值为空的原因是发生了除以零错误。
set_friendly_exception	指示值为空的原因是数字异常。
is_data_nullable	检索数据值是否可空的指示。
is_semantic_null	返回值在语义上为空的指示。
check_friendly_div_by_0	确定空指示的原因是否为除以零错误。
check_friendly_exception	确定空指示的原因是否为数字异常。
get_for_bit_data	检索数据的 FOR BIT DATA 标志。
get_null_indicator	检索可空指示符。
get_data_type	检索数据类型。
get_maximum_length	检索最大长度。
get_precision	获取数字类型的精度。
get_scale	获取数字类型的小数位。
get_codepage	检索字符类型的代码页。

get_actual_length 函数

用途 检索数据值的实际长度。

语法

```
int get_actual_length ()
```

输入自变量

无。

输出自变量

无。

返回值 实际数据长度。

set_actual_length 函数

用途 设置数据值的实际长度。

语法

```
void set_actual_length (int a_length)
```

输入自变量

表 322. set_actual_length 成员函数的输入自变量

名称	数据类型	描述
a_length	int	长度。

输出自变量

无。

返回值 无。

get_data 函数

用途 检索指向数据值的指针。

语法

```
unsigned char* get_data ()
```

输入自变量

无。

输出自变量

无。

返回值 指向数据值的指针。

set_data 函数

用途 将信息复制到数据值中。

语法

```
sqlint32 set_data (unsigned char* a_data_ptr,
                   int            a_copy_len)
```

输入自变量

表 323. set_data 成员函数的输入自变量

名称	数据类型	描述
a_data_ptr	unsigned char*	数据值。
a_copy_len	int	值的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

is_data_null 函数

用途 指示数据值是否为空。

语法

```
sqlint32 is_data_null ()
```

输入自变量

无。

输出自变量

无。

返回值 空指示。

set_data_null 函数

用途 将数据值标记为空。

语法

```
sqlint32 set_data_null ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

clear_null_indicator 函数

用途 复位数据值的空指示符。

语法

```
sqlint32 clear_null_indicator ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

set_friendly_div_by_0 函数

用途 指示值为空的原因是发生了除以零错误。

语法

```
sqlint32 set_friendly_div_by_0 ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

set_friendly_exception 函数

用途 指示值为空的原因是数字异常。

语法

```
sqlint32 set_friendly_exception ()
```

输入自变量

无。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

is_data_nullable 函数

用途 检索数据值是否可空的指示。

语法

```
sqlint32 is_data_nullable ()
```

输入自变量

无。

输出自变量

无。

返回值 可空指示。如果可空的话，则值为 TRUE。

is_semantic_null 函数

用途 返回值在语义上为空的指示。

语法

```
sqlint32 is_semantic_null ()
```

输入自变量

无。

输出自变量

无。

返回值 可空指示。如果可空的话，则值为 TRUE。

check_friendly_div_by_0 函数

用途 确定空指示的原因是否为除以零错误。

语法

```
sqlint32 check_friendly_div_by_0
```

输入自变量

无。

输出自变量

无。

返回值 空指示。如果发生了除以零的情况，则该值为 TRUE。

check_friendly_exception 函数

用途 确定空指示的原因是否为数字异常。

语法

```
sqlint32 check_friendly_exception ()
```

输入自变量

无。

输出自变量

无。

返回值 空指示。如果发生异常，则该值为 TRUE。

get_for_bit_data 函数

用途 检索数据的 FOR BIT DATA 标志。

语法

```
unsigned char get_for_bit_data ()
```

输入自变量

无。

输出自变量

无。

返回值 FOR BIT DATA 标志（Y 或 N）。

get_null_indicator 函数

用途 检索可空指示符。

语法

```
short get_null_indicator ()
```

输入自变量

无。

输出自变量

无。

返回值 空指示符标志，即 SQL_NULLABLE 或 SQL_NO_NULLS。

get_data_type 函数

用途 检索数据类型。

语法

```
short get_data_type ()
```

输入自变量

无。

输出自变量

无。

返回值 数据类型（SQL_TYP_xxx）。请参阅 SQL_TYP_xxx 符号的 sql.h 头文件。

get_maximum_length 函数

用途 检索类型的最大长度。

语法

```
int get_maximum_length ()
```

输入自变量

无。

输出自变量

无。

返回值 类型的最大长度。

get_precision 函数

用途 获取数字类型的精度。

语法

```
unsigned char get_precision ()
```

输入自变量

无。

输出自变量
无。

返回值 数字精度。

get_scale 函数

用途 获取数字类型的小数位。

语法

```
unsigned char get_scale ()
```

输入自变量
无。

输出自变量
无。

返回值 数字小数位。

get_codepage 函数

用途 检索字符类型的代码页。

语法

```
unsigned short get_codepage ()
```

输入自变量
无。

输出自变量
无。

返回值 代码页。

相关任务:

- 『运行时数据类』（《*IBM DB2 Information Integrator 包装器开发者指南*》）

相关参考:

- 第 151 页的『C++ API 的数据类』

Runtime_Data_List 类（C++）

本主题描述 Runtime_Data_List 类并提供每个成员函数的详细信息。

概述

Runtime_Data_List 类提供指向 Runtime_Data 对象的指针的列表。

Runtime_Data_List 类是 C++ API 的其中一个数据类。

用法 此类由 DB2 查询网关实例化，并且从不由包装器划分其子类。

文件 sqlqg_runtime_data_operation.h

数据成员

无。

成员函数

下表描述 Runtime_Data_List 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 324. Runtime_Data_List 类的成员函数

成员函数	描述
get_number_of_values	检索值的数目。
get_ith_value	检索第 *i* 个数据描述符。
operator[]	通过使用下标表示法检索列表中第 *i* 个条目。

get_number_of_values 函数

用途 检索值的数目。

语法

```
int get_number_of_values ()
```

输入自变量

无。

输出自变量

无。

返回值 值的数目。

get_ith_value 函数

用途 检索第 *i* 个数据值。

语法

```
Runtime_Data* get_ith_value (int a_index)
```

输入自变量

表 325. get_ith_value 成员函数的输入自变量

名称	数据类型	描述
a_index	int	在列表中建立索引（从 0 开始）。

输出自变量

无。

返回值 指向第 *i* 个值（或空值）的指针。

operator[] 函数

用途 通过使用下标表示法检索列表中第 *i* 个条目。

语法

```
Runtime_Data* operator[] (int a_index)
```

输入自变量

表 326. operator[] 成员函数的输入自变量

名称	数据类型	描述
a_index	int	在列表中建立索引（从 0 开始）。

输出自变量

无。

返回值 第 *i* 个条目。

相关任务:

- 『运行时数据类』（《*IBM DB2 Information Integrator 包装器开发者指南*》）

相关参考:

- 第 151 页的『C++ API 的数据类』

Wrapper_Utilities 类（C++）

本主题描述 Wrapper_Utilities 类并提供有关每个成员函数的详细信息。

概述

Wrapper_Utilities 类是几个静态实用程序函数的容器。不要实例化 Wrapper_Utilities 类或者为该类创建子类。

Wrapper_Utilities 类是 C++ API 的实用程序类。

文件 sqlqg_utils.h

数据成员

无。

成员函数

下表描述了 Wrapper_Utilities 类的每个成员函数。在该表的后面更详细地描述了每个函数。

表 327. Wrapper_Utilities 类的成员函数

成员函数	描述
convert_to_upper	使用指定的代码页将字符串转换为大写。
convert_to_lower	使用指定的代码页将字符串转换为小写。
report_error	生成要向用户报告的错误。
report_warning	生成要向用户报告的警告。
allocate	分配一定数量的内存。
deallocate	释放由 allocate() 分配的一定数量的内存。
get_sb_DB_codepage	返回当前数据库的单字节的代码页。
get_db_DB_codepage	返回当前数据库的双字节的代码页。

表 327. Wrapper_Utilities 类的成员函数 (续)

成员函数	描述
string_to_tokens	扫描一个字符串，并将该字符串划分为连续的标记。
get_db2_install_path	返回一个以 null 结束的字符串，该字符串显示 DB2 Information Integrator 安装目录的绝对路径名。
get_db2_instance_path	返回一个以 null 结束的字符串，该字符串显示 DB2 通用数据库实例的绝对路径名。
trace_data	将一组信息写入 DB2 通用数据库跟踪设施。
get_db2_release	返回包装器当前运行的 DB2 通用数据库的版本（包括修订包）。
convert_codepage	将输入数据从源代码页转换至目标代码页。
get_expected_conv_len	从转换的字符串返回预期的字节数。
get_env_lang	从操作系统返回语言设置。
change_endian2	更改在计算机中共享双字节字符串序列的顺序。
fnc_entry	将条目记录到函数中。
fnc_exit	记录函数的出口。
fnc_data	记录包括探测点和单个数据元素的数据跟踪。
fnc_data2	记录包括探测点和两个数据元素的数据跟踪。
fnc_data3	记录包括探测点和三个数据元素的数据跟踪。
trace_error	记录包括错误代码和探测点的错误跟踪。

convert_to_upper 函数

用途 使用指定的代码页将字符串转换为大写。

用法 使用 get_sb_DB_codepage() 或 get_db_DB_codepage() 函数来获取当前数据库代码页。

语法

```
int convert_to_upper (char*        a_string,
                      size_t       a_length,
                      unsigned int a_codepage)
```

输入自变量

表 328. convert_to_upper 成员函数的输入自变量

名称	数据类型	描述
a_string	char*	要转换的字符串（不是以 null 结束的）。
a_length	size_t	字符串的长度。
a_codepage	unsigned int	用于转换的代码页。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。如果值为 -1，则指示代码页未定义，并且要转换的字符串包含扩展字符。将跳过这些扩展字符。

convert_to_lower 函数

用途 使用指定的代码页将字符串转换为小写。

用法 使用 get_sb_DB_codepage() 或 get_db_DB_codepage() 函数来获取当前数据库代码页。

语法

```
int convert_to_lower (char*        a_string,
                      size_t       a_length,
                      unsigned int a_codepage)
```

输入自变量

表 329. convert_to_lower 成员函数的输入自变量

名称	数据类型	描述
a_string	char*	要转换的字符串（不是以 null 结束的）。
a_length	size_t	字符串的长度。
a_codepage	unsigned int	用于转换的代码页。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。如果值为 -1，则指示代码页未定义，并且要转换的字符串包含扩展字符。将跳过这些扩展字符。

report_error 函数

用途 生成要向用户报告的错误。

语法

```
sqlint32 report_error (const char* a_function_name,
                       int         a_sql_code,
                         int          a_number_of_tokens,
                         ...)
```

输入自变量

表 330. report_error 成员函数的输入自变量

名称	数据类型	描述
a_function_name	const char*	用来标识生成错误的函数的以 null 结束的字符串。该字符串不能超过五个字符。客户机程序可通过 SQLCA 的 sqlerrp 字段来存取此字符串值，此字符串值以大写字母显示，并且具有前缀 SQL。
a_sql_code	int	为错误预定义的 SQL 代码。请参阅 sqlcodes.h 文件。

表 330. report_error 成员函数的输入自变量 (续)

名称	数据类型	描述
a_number_of_tokens	int	消息的替代标记数。
...	(int, const char*)	每个标记的一对值。每对值都由一个整数长度和指向不是以 null 结束的字符串的一个指针组成。

输出自变量

无。

返回值 返回码。此函数将报告的错误代码返回给调用者，调用者又将此错误代码返回给 DB2。

report_warning 函数

用途 生成要向用户报告的警告。

语法

```
sqlint32 report_warning (const char*  a_function_name,
                         int          a_warning_index,
                         char         a_warning_flag,
                         int          a_number_of_tokens,
                           ...)
```

输入自变量

表 331. report_warning 成员函数的输入自变量

名称	数据类型	描述
a_function_name	const char*	用来标识生成警告的函数的以 null 结束的字符串。该字符串不能超过五个字符。客户机程序可通过 SQLCA 的 sqlerrp 字段来存取此字符串值，此字符串值以大写字母显示，并且具有前缀 SQL。
a_warning_index	int	用来标识警告类型的值。使用在 sql.h 头文件中定义的 SQL_WARN_xxx 常量。
a_warning_flag	char	警告标志。使用在 sql.h 头文件中定义的 SQL_WARNING 常量。
a_number_of_tokens	int	要通过 SQLCA 的 sqlerrm 字段提供的标记（长度与字符串对）的数目，并且客户机程序可以存取该数目。
...	(int, const char*)	每个标记的一对值。每对值都由一个整数长度和指向不是以 null 结束的字符串的一个指针组成。

输出自变量

无。

返回值 如果值为 0，则指示成功地生成了警告。如果返回一个非零值，则指示生成警告时发生了问题。必须将非零返回码返回到 DB2。

allocate 函数

用途 分配一定数量的内存。

语法

```
int allocate (size_t a_size,
              void** a_block)
```

输入自变量

表 332. allocate 成员函数的输入自变量

名称	数据类型	描述
a_size	size_t	要分配的块大小。

输出自变量

表 333. allocate 成员函数的输出自变量

名称	数据类型	描述
a_block	void**	指向已分配的块的指针。

返回值 返回码。如果值为 0，则指示成功。

deallocate 函数

用途 释放由 allocate() 分配的一定数量的内存。

语法

```
void deallocate (void* a_block)
```

输入自变量

表 334. deallocate 成员函数的输入自变量

名称	数据类型	描述
a_block	void*	要释放的内存块。

输出自变量

无。

返回值 无。

get_sb_DB_codepage 函数

用途 返回当前数据库的单字节的代码页。

语法

```
int get_sb_DB_codepage ()
```

输入自变量

无。

输出自变量

无。

返回值 代码页。

get_db_DB_codepage 函数

用途 返回当前数据库的双字节的代码页。

语法

```
int get_db_DB_codepage ()
```

输入自变量

无。

输出自变量

无。

返回值 代码页。

string_to_tokens 函数

用途 扫描一个字符串，并将该字符串划分为连续的标记。此成员函数是 strtok() 和 strtok_r() 函数的安全备用函数。

语法

```
char* string_to_tokens (char*       a_string,
                        const char* a_sep,
                        char**      a_last)
```

输入自变量

表 335. string_to_tokens 成员函数的输入自变量

名称	数据类型	描述
a_string	char*	要扫描的字符串。
a_sep	const char*	要用作标记之间的分隔符的字符串。
a_last	char*	用来保留对 string_to_tokens 的各个调用之间的状态的值。

输出自变量

无。

返回值 指向字符串中的下一个标记的指针或 null。

get_db2_install_path 函数

用途 返回一个以 null 结束的字符串，该字符串显示 DB2 Information Integrator 安装目录的绝对路径名。

语法

```
sqlint32 get_db2_install_path (char*    a_path,
                               sqlint32 a_path_size)
```

输入自变量

表 336. get_db2_install_path 成员函数的输入自变量

名称	数据类型	描述
a_path	char*	指向包含路径名的缓冲区的指针。
a_path_size	sqlint32	缓冲区的长度（包括 NULL 终止符的空间）。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_db2_instance_path 函数

用途 返回一个以 null 结束的字符串，该字符串显示 DB2 通用数据库实例的绝对路径名。

语法

```
sqlint32 get_db2_instance_path (char*    a_path,
                                sqlint32 a_path_size)
```

输入自变量

表 337. get_db2_instance_path 成员函数的输入自变量

名称	数据类型	描述
a_path	char*	指向包含路径名的缓冲区的指针。
a_path_size	sqlint32	缓冲区的长度（包括 NULL 终止符的空间）。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

trace_data 函数

用途 将一组信息写入 DB2 通用数据库跟踪设施。

用法 有关 DB2 通用数据库跟踪设施的更多信息，请参阅 *DB2 Command Reference*。

语法

```
void trace_data (int    a_probe,
                 void*  a_data,
                 int    a_data_size)
```

输入自变量

表 338. trace_data 成员函数的输入自变量

名称	数据类型	描述
a_probe	int	用来标识跟踪点的数目。
a_data	void*	指向要跟踪的数据的指针。
a_data_size	int	要跟踪的数据的长度。

输出自变量

无。

返回值 返回码。如果值为 0，则指示成功。

get_db2_release 函数

用途 返回包装器当前运行的 DB2 通用数据库的版本（包括修订包）。在 sql.h 头文件中定义返回值。

语法

```
int get_db2_release (void)
```

输入自变量

无。

输出自变量

无。

返回值 当前 DB2 通用数据库版本，包括修订包。例如，SQL_REL8103 表示包装器正在 DB2 通用数据库版本 8.1，修订包 3 中运行。

convert_codepage 函数

用途 将输入数据从源代码页转换至目标代码页。

语法

```
static sqlint32 convert_codepage (sqluint8**     a_input_data,
                                  sqluint32      a_input_len,
                                  sqluint32      a_source_CP,
                                  sqluint32      a_target_CP,
                                  sqluint8*      a_output_data,
                                  sqluint32*     a_output_len,
                                  bool*          a_EBCDIC_is_DBCS,
                                  sqluint32*     a_substitute )
```

输入自变量

表 339. convert_codepage 成员函数的输入自变量

名称	数据类型	描述
a_input_data	sqluint8**	指向输入数据字符串的指针。
a_input_len	sqluint32	输入数据字符串中的字节数。
a_source_CP	sqluint32	输入数据字符串的代码页。

表 339. convert_codepage 成员函数的输入自变量 (续)

名称	数据类型	描述
a_target_CP	sqluint32	输入数据字符串转换至的代码页。
a_output_data	sqluint8*	在其中复制转换的数据的输出缓冲区。调用者分配输出缓冲区。
a_output_len	sqluint32*	以字节计的 a_output_data 缓冲区的大小
a_EBCDIC_is_DBCS	bool*	指示 EBCDIC 字符是否为双字节字符集（TRUE 或 FALSE）。

输出自变量

表 340. convert_codepage 成员函数的输出自变量

名称	数据类型	描述
a_output_data	sqluint8*	转换的数据。
a_output_len	sqluint32*	转换的数据的字节数。
a_EBCDIC_is_DBCS	bool*	指示 EBCDIC 字符是否为双字节字符集（TRUE 或 FALSE）。
a_input_data	sqluint8**	指向需要转换的数据字符串中下一个字节的指针。
a_substitute	sqluint32*	指示输出字符串是否包含替换代码（TRUE 或 FALSE）。

返回值 CP_CONV_OK 指示成功。CP_CONV_BUFFER_SMALL 是一个警告，指示用于转换的数据的输出缓冲区太小。CP_CONV_CP_SAME 是一个警告，指示源代码页和目标代码页相同。CP_CONV_DBCS_TRUNCATE 是一个警告，指示输入数据字符串的最后一个字符被截断。CP_CONV_NOT_SUPPORTED 是一个错误，指示转换表不存在；CP_CONV_ERROR 指示发生另一个错误。

get_expected_conv_len 函数

用途 为转换的字符串返回预期字节数。

语法

```
static sqluint32 get_expected_conv_len (sqluint32 a_input_len,
                                        sqluint32 a_source_CP,
                                        sqluint32 a_target_CP)
```

输入自变量

表 341. get_expected_conv_len 成员函数的输入自变量

名称	数据类型	描述
a_input_len	sqluint32	初始字符串的字节数。
a_source_CP	sqluint32	初始字符串的代码页。
a_target_CP	sqluint32	初始字符串转换至的代码页。

输出自变量

无。

返回值 数据类型为 sqluint32 的转换字符串的预期字节数。

get_env_lang 函数

用途 从操作系统返回语言设置。

语法

```
static sqluint32 get_env_lang (sqlint8* a_buffer)
```

输入自变量

表 342. get_env_lang 成员函数的输入自变量

名称	数据类型	描述
a_buffer	sqlint8*	指向长度为 256 个字节的缓冲区的指针。

输出自变量

表 343. get_env_lang 成员函数的输出自变量

名称	数据类型	描述
a_buffer	sqlint8*	以 null 结束的语言设置（对于 UNIX 操作系统，此值为 LANG；对于 Microsoft Windows 操作系统，此值为系统缺省语言设置）。

返回值 如果值为 0，则指示成功。非零值表示发生错误。

change_endian2 函数

用途 更改在计算机中存储双字节字符串序列的顺序。将大尾数表示法更改为小尾数表示法；将小尾数表示法更改为大尾数表示法。

语法

```
static void change_endian2 (sqlint8*  a_source,
                            sqluint32 a_source_len)
```

输入自变量

表 344. change_endian2 成员函数的输入自变量

名称	数据类型	描述
a_source	sqlint8*	要更改的双字节字符串。
a_source_len	sqluint32	输入字符串中的字节数。

输出自变量

表 345. change_endian2 成员函数的输出自变量

名称	数据类型	描述
a_source	sqlint8*	转换的字符串。

返回值 无。

fnc_entry 函数

用途 当包装器跟踪设施运行时，将条目记录到函数中。

语法

```
static void fnc_entry (sqlint16     a_function_id,
                       const char*  a_function_name)
```

输入自变量

表 346. fnc_entry 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。

输出自变量

无。

返回值 无。

fnc_exit 函数

用途 当包装器跟踪设施运行时，记录函数中的出口。

语法

```
static void fnc_exit (sqlint16     a_function_id,
                      const char*  a_function_name,
                      sqlint32     a_rc)
```

输入自变量

表 347. fnc_exit 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。
a_rc	sqlint32	指向返回码的指针；如果值为 0，则指示成功。

输出自变量

无。

返回值 无。

fnc_data 函数

用途 记录包装器跟踪设施中的数据跟踪，此数据跟踪包括探测点和单个数据元素。

语法

```
static void fnc_data (sqlint16     a_function_id,
                      const char*  a_function_name,
                      sqluint32    a_probe,
                      sqluint32    a_data1_size,
                      const void*  a_data1)
```

输入自变量

表 348. fnc_data 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。
a_probe	sqluint32	探测点或跟踪点。此数字标识正在生成跟踪数据的代码中的行。
a_data1_size	sqluint32	以字节计的 a_data1 自变量的大小。
a_data1	const void*	指向要跟踪的数据的指针。

输出自变量

无。

返回值 无。

fnc_data2 函数

用途 记录包装器跟踪设施中的数据跟踪，此数据跟踪包括探测点和两个数据元素。

语法

```
static void fnc_data2 (sqlint16     a_function_id,
                       const char*  a_function_name,
                       sqluint32    a_probe,
                       sqluint32    a_data1_size,
                       const void*  a_data1,
                       sqluint32    a_data2_size,
                       const void*  a_data2)
```

输入自变量

表 349. fnc_data2 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。
a_probe	sqluint32	探测点或跟踪点。此数字标识正在生成跟踪数据的代码中的行。
a_data1_size	sqluint32	以字节计的 a_data1 自变量的大小。

表 349. fnc_data2 成员函数的输入自变量 (续)

名称	数据类型	描述
a_data1	const void*	指向要跟踪的数据的指针。
a_data2_size	sqluint32	以字节计的 a_data2 自变量的大小。
a_data2	const void*	指向要跟踪的数据的指针。

输出自变量

无。

返回值 无。

fnc_data3 函数

用途 记录包装器跟踪设施中的数据跟踪，此跟踪数据包括探测点和三个数据元素。

语法

```
static void fnc_data3 (sqlint16      a_function_id,
                       const char*   a_function_name,
                       sqluint32     a_probe,
                       sqluint32     a_data1_size,
                       const void*   a_data1,
                       sqluint32     a_data2_size,
                       const void*   a_data2,
                       sqluint32     a_data3_size,
                       const void*   a_data3)
```

输入自变量

表 350. fnc_data3 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。
a_probe	sqluint32	探测点或跟踪点。此数字标识正在生成跟踪数据的代码中的行。
a_data1_size	sqluint32	以字节计的 a_data1 自变量的大小。
a_data1	const void*	指向要跟踪的数据的指针。
a_data2_size	sqluint32	以字节计的 a_data2 自变量的大小。
a_data2	const void*	指向要跟踪的数据的指针
a_data3_size	sqluint32	以字节计的 a_data3 自变量的大小。
a_data3	const void*	指向要跟踪的数据的指针。

输出自变量

无。

返回值 无。

trace_error 函数

用途 记录包装器跟踪设施中的错误跟踪，此错误跟踪包括错误代码、错误数据和探测点。

语法

```
static void trace_error (sqlint16     a_function_id,
                         const char*  a_function_name,
                         sqluint32    a_probe,
                         sqluint32    a_data_size,
                         const void*  a_data
```

输入自变量

表 351. trace_error 成员函数的输入自变量

名称	数据类型	描述
a_function_id	sqlint16	函数标识。
a_function_name	const char*	调用跟踪成员函数的函数。
a_probe	sqluint32	探测点或跟踪点。此数字可用于标识生成跟踪数据的代码中的行。
a_data_size	sqluint32	以字节计的错误跟踪的大小。
a_data	const void*	错误跟踪。

输出自变量

无。

返回值 无。

相关概念:

- 『包装器跟踪设施』（*《IBM DB2 Information Integrator 包装器开发者指南》*）

相关参考:

- 『Wrapper utilities 类』（*《IBM DB2 Information Integrator 包装器开发者指南》*）
- 第 1 页的『C++ API 的目录类』
- 第 151 页的『C++ API 的数据类』

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 182 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 182 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:

- 『点分十进制语法图』（*基础结构主题（DB2 公共文件）*）

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

本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 “按现状” 提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。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 国际软件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。

本资料中可能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这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

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。如果是为按照 IBM 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。

凡这些实例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
DB2

下列各项是其他公司的商标或注册商标:

Microsoft 和 Windows 是 Microsoft 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。

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。

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

F

P

R

S

U

W

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 上的联机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

S152-0844-00

Spine information:

IBM DB2 Information Integrator

开发包装器的 C++ API 参考

版本 8.2