

IBM DB2 Information Integrator



# 包装器开发者 API 参考

版本 8



IBM DB2 Information Integrator



# 包装器开发者 API 参考

版本 8

在使用本资料及其支持的产品之前，请阅读第 219 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](http://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](http://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。

© Copyright International Business Machines Corporation 2003. All rights reserved.

# 目录

关于本书 . . . . .	v	Runtime_Data_Desc_List 类 . . . . .	184
本书的读者 . . . . .	v	Runtime_Data 类 . . . . .	187
本书中使用的约定和术语 . . . . .	v	Runtime_Data_List 类 . . . . .	195
<b>C++ 类和方法 . . . . .</b>	<b>1</b>	Wrapper_Uilities 类 . . . . .	197
C++ API 的目录类 . . . . .	1	<b>DB2 Information Integrator 技术文档 . . . . .</b>	<b>207</b>
Catalog_Option 类 . . . . .	2	访问书籍和发行信息 . . . . .	207
Wrapper_Info 类 . . . . .	4	DB2 Information Integrator 书籍 . . . . .	207
Server_Info 类 . . . . .	12	发行说明和安装需求 . . . . .	210
User_Info 类 . . . . .	22	DB2 Information Integrator 文档的修订包 . . . . .	210
Column_Info 类 . . . . .	28	使用 DB2 Information Integrator 信息中心或	
Nickname_Info 类 . . . . .	48	DB2 HTML 文档 CD 访问主题 . . . . .	210
C++ API 的包装器类 . . . . .	62	DB2 Information Integrator 信息中心的功能 . . . . .	211
Unfenced_Generic_Wrapper 类 . . . . .	62	查找 DB2 Information Integrator 信息中心	
Fenced_Generic_Wrapper 类 . . . . .	69	中的主题 . . . . .	211
C++ API 的服务器类 . . . . .	75	使用 DB2 HTML 文档 . . . . .	212
Unfenced_Generic_Server 类 . . . . .	75	搜索 DB2 文档 . . . . .	214
Fenced_Generic_Server 类 . . . . .	85	对于使用 Netscape 4.x 搜索 DB2 文档进	
C++ API 的用户类 . . . . .	92	行故障诊断 . . . . .	215
Unfenced_Generic_User 类 . . . . .	92	<b>易使用性 . . . . .</b>	<b>217</b>
Fenced_Generic_User 类 . . . . .	98	键盘输入和导航 . . . . .	217
C++ API 的昵称类 . . . . .	101	易使用的界面显示 . . . . .	217
Unfenced_Generic_Nickname 类 . . . . .	101	字体设置 . . . . .	217
Fenced_Generic_Nickname 类 . . . . .	109	与颜色的不相关性 . . . . .	217
Remote_Connection 类 . . . . .	114	备用的警告提示 . . . . .	217
C++ API 的操作类 . . . . .	121	与辅助技术的兼容性 . . . . .	218
Remote_Query 类 . . . . .	121	可访问文档 . . . . .	218
Remote_Passthru 类 . . . . .	130	<b>声明 . . . . .</b>	<b>219</b>
C++ API 的请求类 . . . . .	137	商标 . . . . .	221
Request 类 . . . . .	137	<b>索引 . . . . .</b>	<b>223</b>
Reply 类 . . . . .	143	<b>与 IBM 联系 . . . . .</b>	<b>225</b>
Request_Exp 类 . . . . .	159	产品信息 . . . . .	225
Request_Exp_Type 类 . . . . .	166	关于文档的意见 . . . . .	225
Request_Constant 类 . . . . .	169		
Predicate_List 类 . . . . .	173		
C++ API 的数据类 . . . . .	179		
Runtime_Data_Desc 类 . . . . .	179		



---

## 关于本书

本书提供了有关您在为数据源开发包装器时可以使用的 API 类的参考信息。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。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<sup>®</sup> 提供的 DB2<sup>®</sup> 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 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	封装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	封装 CREATE USER MAPPING 和 ALTER USER MAPPING 语句中的用户映射的目录信息的类。
Column_Info	封装昵称列的目录信息的类。此类包括列统计信息。
Nickname_Info	封装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』
- 第 4 页的『Wrapper\_Info 类』
- 第 12 页的『Server\_Info 类』
- 第 22 页的『User\_Info 类』
- 第 28 页的『Column\_Info 类』
- 第 48 页的『Nickname\_Info 类』

---

## Catalog\_Option 类

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 类表示单值或多值的已命名目录选项。子类将区分这些选项。

Catalog\_Option 类是 C++ API 的目录类之一。

**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。

### 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** 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。

## Catalog\_Option

### 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

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 的目录类之一。

**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)
```

## Wrapper\_Info

### 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 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
<code>a_wrapper_type</code>	<code>sqluint8</code>	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
<code>a_wrapper_type</code>	<code>sqluint8*</code>	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
<code>a_wrapper_version</code>	<code>sqlint32</code>	包装器代码版本。

## Wrapper\_Info

输出自变量

无。

返回值 无。

### get\_version 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）。
a_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束）。
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

## Wrapper\_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
<code>a_option</code>	<code>Catalog_Option*</code>	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
<code>a_opt_name</code>	<code>sqluint8*</code>	选项名（不是以 <code>null</code> 结束）。
<code>a_name_len</code>	<code>sqlint32</code>	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
<code>a_option</code>	<code>Catalog_Option**</code>	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

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` 类封装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 的目录类之一。

**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函数。

表 18. `Server_Info` 类的构造函数

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

表 19. `Server_Info` 类的成员函数

成员函数	描述
<code>add_option</code>	将单值选项添加至目录信息。
<code>drop_option</code>	从 <code>Catalog_Info</code> 类中删除选项。
<code>get_option</code>	按名称检索目录选项。
<code>get_first_option</code>	检索指向选项链中第一个选项的指针。
<code>get_next_option</code>	检索选项链中的下一个选项。
<code>get_basic_info</code>	从 <code>Server_Info</code> 对象中检索基本信息。
<code>get_server_name</code>	从 <code>Server_Info</code> 中检索服务器名（如果该名称有效的话）。
<code>set_server_type</code>	设置服务器的服务器类型。

表 19. *Server\_Info* 类的成员函数 (续)

成员函数	描述
<code>get_server_type</code>	从 <i>Server_Info</i> 中检索服务器类型 (如果该类型有效的话)。
<code>set_server_version</code>	设置服务器的服务器版本。
<code>get_server_version</code>	从 <i>Server_Info</i> 中检索服务器版本 (如果该版本有效的话)。
<code>get_wrapper_name</code>	从 <i>Server_Info</i> 中检索包装器名 (如果该名称有效的话)。
<code>merge</code>	将增量 <i>Server_Info</i> 对象合并到当前对象中。
<code>copy</code>	复制 <i>Server_Info</i> 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）。
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）。
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）。
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）。
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；0 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）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束）。
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）。
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）。
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。

## Server\_Info

**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）。
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)
```

## 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

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 类封装 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 的目录类之一。

**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）。

表 42. `add_option` 成员函数的输入自变量 (续)

名称	数据类型	描述
<code>a_opt_name_len</code>	<code>sqlint32</code>	选项名的长度。
<code>a_opt_value</code>	<code>sqluint8*</code>	选项值 (不是以 <code>null</code> 结束)。
<code>a_opt_value_len</code>	<code>sqlint32</code>	选项值的长度。
<code>a_act</code>	<code>Catalog_Option::Action</code>	此选项的操作 (ADD、SET、DROP 或无)。
<code>a_opt_type</code>	<code>char*</code>	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 <code>sqlqg_misc.h</code> 头文件中定义的 <code>SQLQG_USER_OPTION</code> 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
<code>a_option</code>	<code>Catalog_Option*</code>	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）。
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

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** 类封装昵称列的目录信息。此类包括列统计信息。

**Column\_Info** 类是 C++ API 的目录类之一。

**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函数。

表 47. `Column_Info` 类的构造函数

构造函数	描述
<code>Column_Info</code>	构造缺省（空） <code>Column_Info</code> 对象。

表 48. `Column_Info` 类的成员函数

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

表 48. *Column\_Info* 类的成员函数 (续)

成员函数	描述
<code>get_avg_length</code>	从 <code>Column_Info</code> 中检索平均列长度 (如果该值有效的话)。
<code>set_high2key</code>	设置列的次高值。
<code>get_high2key</code>	从 <code>Column_Info</code> 中检索次高值 (如果该值有效的话)。
<code>set_low2key</code>	设置列的次低值。
<code>get_low2key</code>	从 <code>Column_Info</code> 中检索次低值 (如果该值有效的话)。
<code>get_default</code>	从 <code>Column_Info</code> 中检索缺省值 (如果该值有效的话)。
<code>set_colcard</code>	设置列的基数。
<code>get_colcard</code>	从 <code>Column_Info</code> 中检索基数 (如果该值有效的话)。
<code>set_codepage1</code>	设置列的代码页。
<code>get_codepage1</code>	从 <code>Column_Info</code> 中检索代码页 (如果该值有效的话)。
<code>set_codepage2</code>	设置列的代码页。
<code>get_codepage2</code>	从 <code>Column_Info</code> 中检索代码页 (如果该值有效的话)。
<code>merge</code>	将增量 <code>Column_Info</code> 对象合并到当前对象中。
<code>copy</code>	复制 <code>Column_Info</code> 对象。
<code>add_option</code>	将单值选项添加至目录信息。
<code>drop_option</code>	从 <code>Catalog_Info</code> 类中删除选项。
<code>get_option</code>	按名称检索目录选项。
<code>get_first_option</code>	检索指向选项链中第一个选项的指针。
<code>get_next_option</code>	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）。
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
<code>a_column_name</code>	<code>sqluint8**</code>	指向以 <code>null</code> 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
<code>a_new_col_name</code>	<code>sqluint8**</code>	指向新的以 <code>null</code> 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）。
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 Server\_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。

## Column\_Info

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（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）。
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）。
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。

## Column\_Info

### 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）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束）。
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）。

表 79. *add\_option* 成员函数的输入自变量 (续)

名称	数据类型	描述
<code>a_opt_type</code>	<code>char*</code>	在 <code>SQLN1884</code> 错误消息中使用的标记 (如果这是重复选项的话)。使用在 <code>sqlqg_misc.h</code> 头文件中定义的 <code>SQLQG_COLUMN_OPTION</code> 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
<code>a_option</code>	<code>Catalog_Option*</code>	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）。
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

## Column\_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

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 类封装目录中的昵称定义并包括列定义。

Nickname\_Info 类是 C++ API 的目录类之一。

**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	检索昵称的本地名称。
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

### 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** 检索昵称的本地名称。

**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)
```

## Nickname\_Info

### 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 值。

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)
```

## Nickname\_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）。
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）。
a_opt_name_len	sqlint32	选项名的长度。
a_opt_value	sqluint8*	选项值（不是以 null 结束）。
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）。
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 62 页的『Unfenced\_Generic\_Wrapper 类』
- 第 69 页的『Fenced\_Generic\_Wrapper 类』

---

### Unfenced\_Generic\_Wrapper 类

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。

表 110. *Unfenced\_Generic\_Wrapper* 类的成员数据 (续)

名称	数据类型	描述
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、析构函数和函数。

表 111. *Unfenced\_Generic\_Wrapper* 类的构造函数

构造函数	描述
Unfenced_Generic_Wrapper	构造 <i>Unfenced_Generic_Wrapper</i> 类的实例。

表 112. *Unfenced\_Generic\_Wrapper* 类的析构函数

析构函数	描述
~Unfenced_Generic_Wrapper	<i>Unfenced_Generic_Wrapper</i> 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。
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。

## Unfenced\_Generic\_Wrapper

### 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 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 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 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 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 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 函数**

**定义位置**

包装器

**用途** 返回指向包装器目录信息的指针。

**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。

## Unfenced\_Generic\_Wrapper

### initialize\_my\_wrapper 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 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*	服务器的以 null 结束的名称。

## 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`。

## 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** 启用 DB2 联合服务器来实例化 `Unfenced_Generic_Wrapper` 的特定于包装器的子类的 `Hook` 函数。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" UnfencedWrapper* UnfencedWrapper_Hook()
```

### 输入自变量

无。

### 输出自变量

无。

**返回值** 如果产生了错误，则将返回 `Unfenced_Generic_Wrapper` 的已实例化的子类或者是 `null`。

### 相关参考:

- 第 62 页的『C++ API 的包装器类』

---

## Fenced\_Generic\_Wrapper 类

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。

## Fenced\_Generic\_Wrapper

**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
<code>m_info</code>	<code>Wrapper_Info*</code>	包含包装器的目录信息。
<code>m_name</code>	<code>sqluint8*</code>	指向包含包装器名称的字符串的指针。
<code>m_corelib</code>	<code>sqluint8*</code>	指向包含包装器库名称的字符串的指针。
<code>m_version</code>	<code>sqlint32</code>	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、析构函数和函数。

表 124. `Fenced_Generic_Wrapper` 类的构造函数

构造函数	描述
<code>FencedGeneric_Wrapper</code>	构造 <code>Fenced_Generic_Wrapper</code> 类的实例。

表 125. `Fenced_Generic_Wrapper` 类的析构函数

析构函数	描述
<code>~FencedGeneric_Wrapper</code>	<code>Fenced_Generic_Wrapper</code> 类的析构函数。

表 126. `Fenced_Generic_Wrapper` 类的成员函数

成员函数	描述
<code>get_name</code>	返回指向包含包装器名称的以 <code>null</code> 结束的字符串的指针。
<code>get_corelib</code>	返回指向包含包装器库名称的以 <code>null</code> 结束的字符串的指针。
<code>get_version</code>	返回包装器代码的版本。
<code>get_info</code>	返回指向包装器目录信息的指针。
<code>initialize_my_wrapper</code>	从目录信息对象初始化包装器对象状态。

表 126. *Fenced\_Generic\_Wrapper* 类的成员函数 (续)

成员函数	描述
<code>create_server</code>	实例化包装器的 <code>Server</code> 的适当子类。

**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
<code>a_wrapper_version</code>	<code>sqlint32</code>	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
<code>a_rc</code>	<code>sqlint32*</code>	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。

## Fenced\_Generic\_Wrapper

### **get\_name** 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** 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** 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 函数

定义位置

包装器

用途 返回指向包装器目录信息的指针。

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 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。

## Fenced\_Generic\_Wrapper

### 输出自变量

无。

**返回值** 返回码。如果值为 0，则指示成功。

### create\_server 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*	服务器的以 null 结束的名称。

### 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。

## 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** 启用 DB2 联合服务器来实例化 Fenced\_Generic\_Wrapper 的特定于包装器的子类的 Hook 函数。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" FencedWrapper* FencedWrapper_Hook()
```

## 输入自变量

无。

## 输出自变量

无。

**返回值** 如果产生了错误，则将返回 `Fenced_Generic_Wrapper` 的已实例化的子类或者是 `null`。

## 相关参考:

- 第 62 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
<code>Unfenced_Generic_Server</code>	<code>Unfenced_Generic_Server</code> 类是 <code>Unfenced_Server</code> 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。 <code>Unfenced_Generic_Server</code> 类将负责验证 <code>CREATE SERVER</code> 和 <code>ALTER SERVER</code> 语句中的目录信息。
<code>Fenced_Generic_Server</code>	<code>Fenced_Generic_Server</code> 类是 <code>Server</code> 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。 <code>Fenced_Generic_Server</code> 类将负责创建远程连接和昵称。

## 相关参考:

- 第 75 页的『`Unfenced_Generic_Server` 类』
- 第 85 页的『`Fenced_Generic_Server` 类』

## Unfenced\_Generic\_Server 类

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 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。

表 135. Unfenced\_Generic\_Server 类的成员函数 (续)

成员函数	描述
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 <i>plan_request</i> 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 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** 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。

## Unfenced\_Generic\_Server

### get\_info 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保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。

#### 相关参考:

- 第 75 页的『C++ API 的服务器类』

---

## Fenced\_Generic\_Server 类

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。

## Fenced\_Generic\_Server

### 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。

表 156. *Fenced\_Generic\_Server* 类的成员函数 (续)

成员函数	描述
<code>get_version</code>	检索在 <code>CREATE SERVER</code> 语句中指定的以 <code>null</code> 结束的服务器版本。
<code>get_info</code>	返回指向存储的目录信息对象的指针。
<code>initialize_my_server</code>	从有效的目录信息中初始化服务器实例。
<code>create_remote_user</code>	实例化此包装器的 <code>Remote_User</code> 的适当子类。
<code>create_nickname</code>	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
<code>a_server_name</code>	<code>sqluint8*</code>	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 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 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 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

## Fenced\_Generic\_Server

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保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。

## Fenced\_Generic\_Server

### 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。

### 相关参考:

- 第 75 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 92 页的『Unfenced\_Generic\_User 类』
- 第 98 页的『Fenced\_Generic\_User 类』

---

## Unfenced\_Generic\_User 类

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。

## Unfenced\_Generic\_User

表 169. *Unfenced\_Generic\_User* 类的成员函数 (续)

成员函数	描述
<code>verify_my_register_user_info</code>	验证 <code>CREATE USER MAPPING</code> 语句中的信息。
<code>verify_my_alter_user_info</code>	验证 <code>ALTER USER MAPPING</code> 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
<code>a_user_name</code>	<code>sqluint8*</code>	以 <code>null</code> 结束的 (本地) 用户名。
<code>a_server</code>	<code>UnfencedServer*</code>	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
<code>a_rc</code>	<code>sqlint32*</code>	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 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 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 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 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 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。

## 相关参考:

- 第 92 页的『C++ API 的用户类』

---

### Fenced\_Generic\_User 类

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
<code>local_name</code>	<code>sqluint8*</code>	以 <code>null</code> 结束的（本地）用户名。
<code>server</code>	<code>Server*</code>	指向拥有此用户的服务器对象的指针。
<code>info</code>	<code>User_Info*</code>	目录信息的本地副本。只有在执行了 <code>initialize_my_user</code> 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、析构函数和函数。

表 179. `Fenced_Generic_User` 类的构造函数

构造函数	描述
<code>Fenced_Generic_User</code>	构造 <code>Fenced_Generic_User</code> 的实例。

表 180. `Fenced_Generic_User` 类的析构函数

析构函数	描述
<code>~Fenced_Generic_User</code>	破坏 <code>Fenced_Generic_User</code> 的实例。

表 181. *Fenced\_Generic\_User* 类的成员函数

成员函数	描述
<code>get_info</code>	返回指向 <code>User_Info</code> 的本地副本的指针。
<code>initialize_my_user</code>	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
<code>a_local_user_name</code>	<code>sqluint8*</code>	以 <code>null</code> 结束的（本地）用户名。
<code>a_server</code>	<code>FencedServer*</code>	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
<code>a_rc</code>	<code>sqlint32*</code>	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 ()
```

## Fenced\_Generic\_User

输入自变量

无。

输出自变量

无。

返回值 无。

### get\_info 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 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。

相关参考:

- 第 92 页的『C++ API 的用户类』

## C++ API 的昵称类

下表描述 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 101 页的『Unfenced\_Generic\_Nickname 类』
- 第 109 页的『Fenced\_Generic\_Nickname 类』

## Unfenced\_Generic\_Nickname 类

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 的昵称类之一。

**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\_nickname.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。

表 186. *Unfenced\_Generic\_Nickname* 类的数据成员 (续)

名称	数据类型	描述
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	取装行的成本 (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)。
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函数。

表 187. *Unfenced\_Generic\_Nickname* 类的构造函数

构造函数	描述
Unfenced_Generic_Nickname	构造 <i>Unfenced_Generic_Nickname</i> 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。

表 188. *Unfenced\_Generic\_Nickname* 类的成员函数 (续)

成员函数	描述
<code>get_advance_cost</code>	检索在缺省成本模型中使用的 <code>ADVANCE_COST</code> 昵称选项的值。
<code>get_setup_cost</code>	检索在缺省成本模型中使用的 <code>SETUP_COST</code> 昵称选项的值。
<code>get_submission_cost</code>	检索在缺省成本模型中使用的 <code>SUBMISSION_COST</code> 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
<code>a_schema</code>	<code>sqluint8*</code>	以 <code>null</code> 结束的模式名。
<code>a_nickname_name</code>	<code>sqluint8*</code>	以 <code>null</code> 结束的 (本地) 昵称名。
<code>a_nickname_server</code>	<code>Unfenced_Generic_Server*</code>	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
<code>a_rc</code>	<code>sqlint32*</code>	指向返回码的指针; 0 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。

## Unfenced\_Generic\_Nickname

### 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)
```

## Unfenced\_Generic\_Nickname

### 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保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* 函数是可信的进程空间的一部分，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）。

**相关参考:**

- 第 101 页的『C++ API 的昵称类』

**Fenced\_Generic\_Nickname 类**

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 的昵称类之一。

**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**

## Fenced\_Generic\_Nickname

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函数。

表 200. *Fenced\_Generic\_Nickname* 类的构造函数

构造函数	描述
<code>Fenced_Generic_Nickname</code>	构造 <code>Fenced_Generic_Nickname</code> 的实例。

表 201. *Fenced\_Generic\_Nickname* 类的成员函数

成员函数	描述
<code>get_local_name</code>	返回指向以 null 结束的本地昵称名的指针。
<code>get_local_schema</code>	返回指向以 null 结束的本地模式名的指针。
<code>get_server</code>	返回指向包含服务器对象的指针。
<code>is_reserved_nickname_option</code>	确定特定选项名是否为内置 DB2 昵称选项之一。
<code>is_reserved_column_option</code>	确定特定选项名是否为内置 DB2 列选项之一。
<code>initialize_my_nickname</code>	使用目录中的有效信息初始化昵称。
<code>verify_my_register_nickname_info</code>	验证 <code>CREATE NICKNAME</code> 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；0 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。

## Fenced\_Generic\_Nickname

### 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。

## Fenced\_Generic\_Nickname

**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
<code>a_nick_info</code>	<code>Nickname_Info*</code>	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
<code>a_delta_info</code>	<code>Nickname_Info**</code>	包装器添加的信息。

**返回值** 返回码。如果值为 0，则指示成功。

### 相关参考:

- 第 101 页的『C++ API 的昵称类』

---

## Remote\_Connection 类

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	<code>connection_kind</code>	连接的种类（1 阶段或 2 阶段）。
服务器	<code>FencedServer*</code>	指向拥有此连接的服务器的指针。
用户	<code>FencedRemote_User*</code>	与连接相关联的用户。
连接	<code>unsigned short</code>	指示操作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函数。

表 210. `Remote_Connection` 类的构造函数

构造函数	描述
<code>Remote_Connection</code>	构造 <code>Remote_Connection</code> 对象。

表 211. `Remote_Connection` 类的成员函数

成员函数	描述
<code>connect</code>	建立与远程数据源的连接。
<code>disconnect</code>	破坏与远程数据源的连接。
<code>is_connected</code>	指示连接是活动的还是不活动的。
<code>create_remote_query</code>	创建 <code>Remote_Query</code> 的适当的子类。
<code>create_remote_passthru</code>	创建 <code>Remote_Passthru</code> 的适当的子类。
<code>get_server</code>	返回指向包含服务器的指针。
<code>get_user</code>	返回指向相关联的 <code>Remote_User</code> 对象的指针。
<code>connect</code>	指示事务成功完成于是远程数据源落实该事务。
<code>rollback</code>	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; 0 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。

## Remote\_Connection

**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。
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。

表 215. `create_remote_passthru` 成员函数的输入自变量 (续)

名称	数据类型	描述
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。

## Remote\_Connection

### 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。

#### 相关参考:

- 第 75 页的『C++ API 的服务器类』



## C++ API 的操作类

下表描述 C++ API 的每个操作类。

表 216. 操作类

类名	描述
Remote_Query	此类封装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以实现该操作。
Remote_Passthru	此类表示远程数据源处的通过操作。包装器必须创建 Remote_Passthru 类的子类才能支持通过操作。

### 相关参考:

- 第 121 页的『Remote\_Query 类』
- 第 130 页的『Remote\_Passthru 类』

## Remote\_Query 类

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 类封装了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。在描述这些表之后更详细地描述了构造函数和函数。

## Remote\_Query

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	取装单个行。
close	关闭查询。
report_eof	报告取装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，而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 数据传送到缓冲区中。

### 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。

表 219. Remote\_Query 构造函数的输入自变量 (续)

名称	数据类型	描述
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。

## Remote\_Query

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** DB2 Information Integrator 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** DB2 Information Integrator 调用此成员函数以使用新的参数值来启动查询。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 reopen (sqlint16 a_status)
```

## Remote\_Query

### 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** 取装单个行。

**用法** DB2 Information Integrator 调用此成员函数以便从远程查询中取装一个非 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_EQQ)。SQLQG_CLOSE_EOS 指示包装器脱离了某种状态，以便可以调用 <code>reopen()</code> 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_EQQ。

## 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** 报告取装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取装了最后一行。

## 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

## Remote\_Query

```
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	先前对 <i>fetch_lob</i> 函数的调用为当前列写入的字节总数。

## 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，而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** 指示包装器已将一部分 LOB 数据传送到缓冲区中。

### 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
<i>a_col_number</i>	sqlint32	包装器要从其中取出数据并将数据写入 LOB 缓冲区的列。
<i>a_bytes_ready</i>	sqluint32	包装器已复制到 LOB 缓冲区中的字节数。
<i>a_status</i>	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。
<i>a_next</i>	sqlqg_lob_intent	用来指示包装器接下来是调用 <code>fetch</code> 函数还是调用 <code>fetch_lob</code> 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。

### 相关参考:

- 第 121 页的『C++ API 的操作类』

---

## Remote\_Passthru 类

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通过操作。包装器必须创建 `Remote_Passthru` 类的子类才能支持通过操作。

`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函数。

表 227. `Remote_Passthru` 类的构造函数

构造函数	描述
<code>Remote_Passthru</code>	构造 <code>Remote_Passthru</code> 对象。

表 228. `Remote_Passthru` 类的成员函数

成员函数	描述
<code>get_connection</code>	检索指向拥有此操作的连接的指针。
<code>get_server</code>	检索指向拥有此操作的服务器的指针。
<code>get_input_data</code>	检索指向操作的输入值列表的指针。
<code>get_output_data</code>	检索指向输出数据缓冲区列表的指针。
<code>get_SQL_statement</code>	检索用于通过执行的语句。
<code>report_eof</code>	报告取装期间的文件结束符条件。
<code>prepare</code>	准备远程通过操作。
<code>describe</code>	描述远程通过操作。
<code>execute</code>	运行远程通过操作。
<code>open</code>	打开远程通过操作的游标。
<code>fetch</code>	在远程通过操作中取装一行。
<code>close</code>	关闭远程通过操作的游标。

## 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)
```

## Remote\_Passthru

### 输入自变量

表 229. *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 230. *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。

## Remote\_Passthru

### **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 报告取装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 准备远程通过操作。

#### 语法

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

#### 输入自变量

无。

## 输出自变量

表 231. *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*	描述通过操作的结果的 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** 描述远程通过操作。

**语法**

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

**输入自变量**

无。

**输出自变量**

表 232. *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*	描述通过操作的结果的 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** 运行远程通过操作。

**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** 打开远程通过操作的游标。

**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** 在远程通过操作中取装一行。

**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** 关闭远程通过操作的游标。

**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。



**相关参考:**

- 第 121 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 233. 请求类

类名	描述
Request	此类封装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	此类封装谓词列表，并供 RRC 协议使用。

**相关参考:**

- 第 159 页的『Request\_Exp 类』
- 第 137 页的『Request 类』
- 第 169 页的『Request\_Constant 类』
- 第 166 页的『Request\_Exp\_Type 类』
- 第 143 页的『Reply 类』
- 第 173 页的『Predicate\_List 类』

**Request 类**

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 封装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。

## Request

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 234. `Request` 类的成员函数

成员函数	描述
<code>get_number_of_quantifiers</code>	检索此查询的量词数目。
<code>get_number_of_predicates</code>	检索此查询的谓词数目。
<code>get_number_of_head_exp</code>	检索此查询的标题（选择列表）表达式的数目。
<code>get_quantifier_handle</code>	检索此查询的第 $n$ 个量词的句柄。
<code>get_predicate_handle</code>	检索此查询的第 $n$ 个谓词的句柄。
<code>get_head_exp_handle</code>	检索此查询的第 $n$ 个标题表达式的句柄。
<code>get_nickname</code>	检索与指定的句柄相关联的昵称对象。
<code>get_head_exp</code>	检索与指定的句柄相关联的标题表达式。
<code>get_predicate</code>	检索与指定的句柄相关联的谓词表达式。
<code>get_distinct</code>	检索查询的 <code>DISTINCT</code> 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 235. *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 236. *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 237. *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 238. *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 239. *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 240. *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 241. *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 242. *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 243. *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 244. *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 245. *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 246. *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。

### 相关参考:

- 第 137 页的『C++ API 的请求类』

---

## Reply 类

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函数。

表 247. 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 248. 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 248. 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	检索取装第一个元组的成本。
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 249. 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 250. *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 251. *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 252. *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 253. *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 254. *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 255. *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 256. *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 257. *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 258. *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 259. *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 260. *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 261. *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 262. *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 263. *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 264. *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 265. *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 266. *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 267. *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 268. *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 269. *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 270. *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\_Uilities::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 271. 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 272. 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 273. *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 274. *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 275. *re\_exec\_cost* 成员函数的输出自变量

名称	数据类型	描述
<i>a_re_exec_cost</i>	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 检索取装第一个元组的成本。

语法

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

输入自变量

无。

输出自变量

表 276. *first\_tuple\_cost* 成员函数的输出自变量

名称	数据类型	描述
<i>a_first_tuple_cost</i>	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 277. *all\_costs* 成员函数的输出自变量

名称	数据类型	描述
<i>a_total_cost</i>	float*	总成本。
<i>a_first_tuple_cost</i>	float*	第一个元组成本。
<i>a_re_exec_cost</i>	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 278. *set\_next\_reply* 成员函数的输入自变量

名称	数据类型	描述
<i>a_next_reply</i>	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。

相关参考:

- 第 137 页的『C++ API 的请求类』

## Request\_Exp 类

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 279. 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 280. 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 281. *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 282. *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。如果这是一个顶层节点，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 283. *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 284. *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 285. *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 286. *get\_value* 成员函数的输出自变量

名称	数据类型	描述
<code>a_constant_value</code>	<code>Request_Constant**</code>	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 287. *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 <code>null</code> 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 288. *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 <code>null</code> 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:**

- 第 137 页的『C++ API 的请求类』

---

### Request\_Exp\_Type 类

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 289. 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。
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，它是 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。

## Request\_Exp\_Type

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:**

- 第 137 页的『C++ API 的请求类』

**Request\_Constant 类**

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 290. 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。

表 290. *Request\_Constant* 类的成员函数 (续)

成员函数	描述
<code>get_codepage</code>	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，它是 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。

## Request\_Constant

### 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:**

- 第 137 页的『C++ API 的请求类』

**Predicate\_List 类**

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 类封装谓词列表，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 291. *Predicate\_List* 类的成员函数

成员函数	描述
<code>create_empty_predicate_list</code>	实例化空的谓词列表。
<code>create_and_copy_predicate_list</code>	创建一个包含特定应答的所有谓词的谓词列表。
<code>get_number_of_predicates</code>	检索列表中的谓词数目。
<code>get_predicate_handle</code>	检索列表中的特定位置的谓词句柄。
<code>get_predicate</code>	检索表达式句柄的谓词表达式。
<code>get_number_of_applied_predicates</code>	获取已经应用的谓词数目。
<code>get_applied_predicate_handle</code>	检索列表中的指定位置已应用的谓词句柄。
<code>get_applied_predicate</code>	检索表达式句柄的已应用的谓词表达式。
<code>add_predicate</code>	将谓词添加到列表中。
<code>add_applied_predicate</code>	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 292. *create\_empty\_predicate\_list* 成员函数的输入自变量

名称	数据类型	描述
<code>a_reply</code>	<code>Reply*</code>	拥有谓词的应答。

输出自变量

表 293. *create\_empty\_predicate\_list* 成员函数的输出自变量

名称	数据类型	描述
<code>a_pred_list</code>	<code>Predicate_List**</code>	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 294. *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 295. *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 296. *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 297. *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 298. *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 299. *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 300. *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 301. *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 302. *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 303. *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 304. *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 305. *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。

## 相关参考:

- 第 137 页的『C++ API 的请求类』

---

## C++ API 的数据类

下表描述 C++ API 的每个数据类。

表 306. 数据类

类名	描述
Runtime_Data_Desc	封装数据项描述的类。
Runtime_Data_Desc_List	Runtime_Data_Desc 的列表。
Runtime_Data	封装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 179 页的『Runtime\_Data\_Desc 类』
- 第 184 页的『Runtime\_Data\_Desc\_List 类』
- 第 187 页的『Runtime\_Data 类』
- 第 195 页的『Runtime\_Data\_List 类』

---

## Runtime\_Data\_Desc 类

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 类封装数据项的描述。

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，并且从不由包装器创建它的子类。此类可以在准备处理通过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函数。

表 307. 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 308. 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。当响应通过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 309. *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 310. *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；0 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。

## Runtime\_Data\_Desc

返回值 代码页。

相关参考:

- 第 179 页的『C++ API 的数据类』

---

## Runtime\_Data\_Desc\_List 类

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 的数据类之一。

用法 此类由 DB2 查询网关实例化，但从不由包装器创建它的子类。

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 311. 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 312. *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 313. *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 314. 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 315. 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 179 页的『C++ API 的数据类』

## Runtime\_Data 类

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 类封装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，并且从不由包装器创建它的子类。

**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 316. 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。

表 316. *Runtime\_Data* 类的成员函数 (续)

成员函数	描述
<code>get_maximum_length</code>	检索最大长度。
<code>get_precision</code>	获取数字类型的精度。
<code>get_scale</code>	获取数字类型的小数位。
<code>get_codepage</code>	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 317. *set\_actual\_length* 成员函数的输入自变量

名称	数据类型	描述
<code>a_length</code>	<code>int</code>	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 318. *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。

**相关参考:**

- 第 179 页的『C++ API 的数据类』

**Runtime\_Data\_List 类**

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 数据类之一。

**用法** 此类由 DB2 查询网关实例化，并且从不由包装器创建它的子类。

**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 319. 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>i</i> 个数据描述符。
operator[]	通过使用下标表示法检索列表中第 <i>i</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 ()
```

## Runtime\_Data\_List

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 320. *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 321. *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 179 页的『C++ API 的数据类』

---

## Wrapper\_Uilities 类

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

### 概述

Wrapper\_Uilities 类是几个静态实用程序函数的容器。不要实例化 Wrapper\_Uilities 类或者为该类创建子类。

Wrapper\_Uilities 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\_Uilities 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 322. Wrapper\_Uilities 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。
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。

表 322. *Wrapper\_Utilities* 类的成员函数 (续)

成员函数	描述
<code>get_db2_instance_path</code>	返回一个以 <code>null</code> 结束的字符串, 该字符串显示 DB2 Information Integrator 实例的绝对路径名。
<code>trace_data</code>	将一组信息写入 DB2 跟踪设施。

### **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 323. *convert\_to\_upper* 成员函数的输入自变量

名称	数据类型	描述
<code>a_string</code>	<code>char*</code>	要转换的字符串 (不是以 <code>null</code> 结束)。
<code>a_length</code>	<code>size_t</code>	字符串的长度。
<code>a_codepage</code>	<code>unsigned int</code>	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, 则指示未定义代码页, 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 324. *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）。
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，则指示未定义代码页，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 325. *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。
a_number_of_tokens	int	消息的替代标记数。

表 325. *report\_error* 成员函数的输入自变量 (续)

名称	数据类型	描述
...	(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 326. *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。



表 326. *report\_warning* 成员函数的输入自变量 (续)

名称	数据类型	描述
...	(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给 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 327. *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 328. *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 329. *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()` 函数的安全替代函数。

**语法**

```
char* string_to_tokens (char*      a_string,
                      const char* a_sep,
                      char**     a_last)
```

**输入自变量**表 330. *string\_to\_tokens* 成员函数的输入自变量

名称	数据类型	描述
<code>a_string</code>	<code>char*</code>	要扫描的字符串。
<code>a_sep</code>	<code>const char*</code>	要用作标记之间的分隔符的字符串。
<code>a_last</code>	<code>char*</code>	用来保留对 <code>string_to_tokens</code> 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 331. *get\_db2\_install\_path* 成员函数的输入自变量

名称	数据类型	描述
<code>a_path</code>	<code>char*</code>	指向包含路径名的缓冲区的指针。
<code>a_path_size</code>	<code>sqlint32</code>	缓冲区的长度（包括 <code>NULL</code> 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 Information Integrator 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 332. *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 跟踪设施。

**用法** 有关 DB2 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 333. *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。

表 333. *trace\_data* 成员函数的输入自变量 (续)

名称	数据类型	描述
<i>a_data</i>	void*	指向要跟踪的数据的指针。
<i>a_data_size</i>	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。

**相关参考:**

- 第 1 页的『C++ API 的目录类』
- 第 179 页的『C++ API 的数据类』



---

## DB2 Information Integrator 技术文档

以下主题描述了如何:

- 访问书籍和发行信息, 包括打印和订购书籍
- 通过使用 DB2 Information Integrator 信息中心或 DB2 HTML 文档 CD 来访问主题

---

### 访问书籍和发行信息

DB2 Information Integrator 技术信息有以下格式可用:

- 书籍 (PDF 以及印刷格式)。DB2 Information Integrator 资料库中每本书的描述都可从“IBM 出版物中心”(IBM Publications Center) 获得, 网址为: [www.ibm.com/shop/publications/order](http://www.ibm.com/shop/publications/order)。
- 信息中心 (HTML 格式)。
- DB2 数据库工具的帮助 (HTML 格式)。

### DB2 Information Integrator 书籍

DB2 Information Integrator PDF 文档 CD 中包含了 DB2 Information Integrator 资料库和 DB2 通用数据库资料库中书籍的 PDF 文件。DB2 Information Integrator PDF 文档 CD 的结构如下:

- 在 Windows 操作系统上: `x:\doc\%L`
- 在 UNIX 操作系统上: `/cdrom/doc/%L/`

其中:

- `x` 表示 CD-ROM 盘符 (Windows)
- `cdrom` 表示 CD-ROM 的安装点 (UNIX)
- `%L` 是要使用的文档的语言环境, 例如, `en_US`

语言	语言环境	标识符	语言	语言环境	标识符
阿拉伯语	ar_AA	w	日语	ja_JP	j
巴西葡萄牙语	pt_BR	b	韩国语	ko_KR	k
保加利亚语	bg_BG	u	挪威语	no_NO	n
克罗地亚语	hr_HR	9	波兰语	pl_PL	p
捷克语	cs_CZ	x	葡萄牙语	pt_PT	v
丹麦语	da_DK	d	罗马尼亚语	ro_RO	8
荷兰语	nl_NL	q	俄语	ru_RU	r

英语	en_US	e	简体中文	zh_CN	c
芬兰语	fi_FI	y	斯洛伐克语	sk_SK	7
法语	fr_FR	f	斯洛文尼亚语	sl_SI	l
德语	de_DE	g	西班牙语	es_ES	z
希腊语	el_GR	a	瑞典语	sv_SE	s
匈牙利语	hu_HU	h	繁体中文	zh_TW	t
意大利语	it_IT	i	土耳其语	tr_TR	m

每个 PDF 文件名中第 6 个位置的字符表示该书的语言版本（参见下表）。例如，文件名 `iiyige80` 表示英文版本的《*IBM DB2 Information Integrator 安装指南*》，而文件名 `iiyigg80` 则表示同一本书的德语版本。

下表中的书可用于 DB2 Information Integrator。

表 334. *DB2 Information Integrator* 文档

书名	书号	安装类别	PDF 文件名
《 <i>IBM DB2 Information Integrator 解决方案指南</i> 》	S152-0551	getting_started	iiyisc80
《 <i>IBM DB2 Information Integrator 安装指南</i> 》	G152-0550	getting_started	iiyigc80
《 <i>IBM DB2 Information Integrator 迁移指南</i> 》	S152-0603	getting_started	iiymgc80
《 <i>IBM DB2 Information Integrator 联合系统指南</i> 》	S152-0600	admin	iiyipc80
《 <i>IBM DB2 Information Integrator 数据源配置指南</i> 》	只有在线时可用	optional	iiylsc80
《 <i>IBM DB2 Information Integrator 开发者指南</i> 》	S152-0601	ad	iiypsc80

## 从 PDF 文件打印书籍

您可以从 DB2 Information Integrator PDF 文档 CD 上的 PDF 文件打印 DB2 Information Integrator 书籍。可以使用 Adobe Acrobat Reader 打印整本书、某个范围内的页或特定页。



## 先决条件:

确保具有 Adobe Acrobat Reader。可从 Adobe Web 站点获得, 网址为 [www.adobe.com](http://www.adobe.com)。

## 过程:

要从 PDF 文件打印 DB2 Information Integrator 书籍:

1. 插入 DB2 Information Integrator PDF 文档 CD。在 UNIX 操作系统中, 装上 CD。
2. 启动 Adobe Acrobat Reader。
3. 从下列位置之一打开 PDF 文件:
  - 在 Windows 操作系统上: `x:\doc\%L`
  - 在 UNIX 操作系统上: `/cdrom/doc/%L/`其中:
  - `x` 表示 CD-ROM 盘符 (Windows)
  - `cdrom` 表示 CD-ROM 的安装点 (UNIX)
  - `%L` 是要打印的文档的语言环境, 例如, `en_US`
4. 单击文件 → 打印。
5. 在“打印”窗口中, 指定是打印所有页、当前页还是某个范围的页。
6. 单击确定。

## 订购印刷书籍

可通过从 IBM 分销商处订购 DB2 Information Integrator 产品的文档包来获取印刷的手册。该文档包是 DB2 Information Integrator 资料库中手册的一部分。这些文档包旨在帮助您开始使用所购买的 DB2 Information Integrator 产品。

文档包中的手册与随 DB2 Information Integrator 产品提供的 DB2 Information Integrator PDF 文档 CD 上的手册一样。

还可以使用以下方法之一来订购单独的书籍:

- 与 IBM 销售代表或授权经销商联系。要查找您当地的 IBM 代表, 请查看“IBM 全球联系人目录”(IBM Worldwide Directory of Contacts), 网址为 [www.ibm.com/planetwide](http://www.ibm.com/planetwide)。
- 在美国可致电 1-800-879-2755, 在加拿大则可致电 1-800-IBM-4YOU。
- 访问“IBM 出版物中心”, 网址为 [www.ibm.com/shop/publications/order](http://www.ibm.com/shop/publications/order)。

## 发行说明和安装需求

本发行说明和安装需求提供特定于产品发行版和修订包级别的信息。本发行说明还提供了并入到每个发行版和修订包中的文档更新的总结。

本发行说明和安装需求可在产品 CD-ROM 上以文本和 HTML 格式获得:

- 在 Windows 操作系统上: `x:\doc\%L`
- 在 UNIX 操作系统上: `/cdrom/doc/%L/`

其中:

- `x` 表示 CD-ROM 盘符 (Windows)
- `cdrom` 表示 CD-ROM 的安装点 (UNIX)
- `%L` 是要使用的文档的语言环境; 例如, `en_US`

表 335. 发行说明

书名	文件名	位置
<i>DB2 Information Integrator</i> 发行说明	<i>ReleaseNotes</i>	<ul style="list-style-type: none"><li>• 产品 CD-ROM</li><li>• DB2 Information Integrator 信息中心</li><li>• DB2 Information Integration 安装启动板</li></ul>
<i>DB2 Information Integrator</i> 安装需求	<i>Prereqs</i>	<ul style="list-style-type: none"><li>• 产品 CD-ROM</li><li>• DB2 Information Integration 安装启动板</li></ul>

## DB2 Information Integrator 文档的修订包

IBM 可能会定期提供文档修订包。在新信息可用时, 可使用文档修订包来更新从 DB2 HTML 文档 CD 安装的信息。

文档修订包是累积修订包。例如, 如果您安装了用于 V8.1 的文档然后应用了 V8.1.2, 您将获得对修订包 1 和 V8.1.2 的文档更新。

安装文档修订包后, 您的 HTML 文档将比针对所用产品的印刷 PDF 手册或在线 PDF 手册更新。

---

## 使用 DB2 Information Integrator 信息中心或 DB2 HTML 文档 CD 访问主题

DB2 Information Integrator 信息中心使您可访问在您的业务中利用 DB2 Information Integrator 所需要的信息。

## DB2 Information Integrator 信息中心的功能

DB2 Information Integrator 信息中心具有以下功能:

### 集成的导航树

从单个导航树找出 DB2 Information Integrator 资料库中的任何主题。

**搜索** 通过单击导航工具栏中的**搜索**来搜索工作站上的所有主题。

**主索引** 可从一个主索引访问主题和工具帮助中的信息。索引包括整个 DB2 Information Integrator 资料库中的条目。

### 主词汇表

主词汇表定义在 DB2 Information Integrator 资料库中使用的术语。

### 定期更新的文档

通过下载更新的 HTML 主题以保持您的主题是最新的。

## 查找 DB2 Information Integrator 信息中心中的主题

DB2 Information Integrator 信息中心由以下主要元素组成:

**导航树** 导航树位于浏览器窗口左边的框架中。该树可展开和折叠以显示和隐藏 DB2 Information Integrator 信息中心中的主题链接、词汇表和主索引。

### 导航工具栏

导航工具栏位于浏览器窗口的右上边框架中。使用导航工具栏中的按钮可搜索 DB2 Information Integrator 信息中心、隐藏导航树以及查找导航树中当前显示的主题。

### 内容框架

内容框架位于浏览器窗口的右下边框架中。在单击导航树中的链接时、单击搜索结果时或跟随另一主题中或主索引中的链接时, 内容框架中会显示适当的主题。

### 先决条件:

要从浏览器访问 DB2 Information Integrator 信息中心, 必须使用以下浏览器之一:

- Microsoft Explorer V5 或更高版本
- Netscape Navigator V6.1 或更高版本

### 限制:

DB2 Information Integrator 信息中心仅包含从 DB2 HTML 文档 CD 中安装的主题集。如果在跟随到某个主题的链接时, 您的 Web 浏览器返回了文件未找到的错误, 应从 DB2 HTML 文档 CD 另外安装一个或多个主题集。

### 过程:

要通过搜索关键字来查找主题:

1. 在导航工具栏中, 单击**搜索**。
2. 在“搜索”窗口最上面的文本输入字段中, 输入两个或多个与您感兴趣的方面相关的术语, 然后单击**搜索**。**结果**字段显示主题列表。与您搜索的字符串最相匹配的主题位于该列表的顶部。

输入的术语越多, 查询的准确度就越高, 从查询返回的主题数就越少。

3. 在**结果**字段中, 单击想要阅读的主题的标题。该主题显示在内容框架中。

要查找导航树中的主题:

1. 在导航树中, 单击您感兴趣的主题类别旁的书籍图标。将会有有一个子类别列表显示在该图标下面。
2. 继续单击书籍图标, 直到找到包含您感兴趣的主题的类别为止。链接至主题的类别在您将光标移到该类别标题上时, 将类别标题显示为链接。在导航树中使用页图标识主题。
3. 单击主题链接。该主题显示在内容框架中。

要通过使用主索引来查找主题:

1. 在导航树中, 单击**索引**。该索引展开, 并显示按字母顺序排列的链接列表。
2. 在导航树中, 单击您正在查找的主题的第一个字符。具有该首字符的条目列表将会显示在内容框架中。如果显示书籍图标, 则该主题具有多个索引条目。
3. 单击与您感兴趣的主题相对应的书籍图标。一个主题列表将显示在您单击的术语下面。
4. 单击符合需要的主题的标题。该主题显示在内容框架中。

## 使用 DB2 HTML 文档

此主题描述如何安装、查看和复制 DB2 HTML 文档 CD 中的文档以及在安装之后如何更新文档。

### 安装 DB2 HTML 文档

DB2 HTML 文档 CD 的安装目录对于各个信息类别来说是不同的:

*htmlcdpath/doc/htmlcd/%L/category*

*htmlcdpath*

DB2 HTML 文档 CD 的安装目录

*%L* 要使用的文档的语言环境, 例如, en\_US。

*类别* 类别标识符, 例如, *getting\_started* 表示安装信息。

## 直接从 DB2 HTML 文档 CD 来查看技术文档

可从 DB2 HTML 文档 CD 中查看所有的 HTML 主题。

### 限制:

必须安装 DB2 产品才能查看联机帮助。

### 过程:

从 DB2 HTML 文档 CD 查看 HTML 文档:

1. 插入 DB2 HTML 文档 CD。
2. 启动 Web 浏览器并打开以下文件:
  - 对于 Windows 操作系统: `x:\Program Files\IBM\sqllib\doc\htmlcd\%L\index.*`
  - 对于 UNIX 操作系统: `/cdrom/Program Files/IBM/sqllib/doc/htmlcd/%L/index.htm`

其中:

- *x* 表示 CD-ROM 盘符 (Windows)
- *cdrom* 表示 CD-ROM 的安装点 (UNIX)
- *%L* 是要使用的文档的语言环境, 例如, `en_US`

## 将文件从 DB2 HTML 文档 CD 复制到 Web 服务器上

在 DB2 HTML 文档 CD 上提供了整个 DB2 资料库, 因此您可以将该资料库复制到 Web 服务器上, 这样更容易访问。

### 过程:

将文件从 DB2 HTML 文档 CD 复制到 Web 服务器上的适当路径 (缺省路径如下所示):

- 对于 Windows 操作系统, 该路径为: `x:\Program Files\IBM\sqllib\doc\htmlcd\%L\*.*`
- 对于 UNIX 操作系统, 该路径为: `/cdrom/Program Files/IBM/sqllib/doc/htmlcd/%L`

其中:

- *x* 表示 CD-ROM 盘符 (Windows)
- *cdrom* 表示 CD-ROM 的安装点 (UNIX)
- *%L* 是要使用的文档的语言环境, 例如, `en_US`

## 更新计算机上的 HTML 文档

当 IBM 提供更新时, 可通过以下任何一种方式更新您从 DB2 HTML 文档 CD 安装的 HTML 文件:

- 使用“信息中心”(如果安装了 DB2 管理图形用户界面工具的话)

- 下载并应用 DB2 HTML 文档修订包

此过程将不会更新 DB2 代码。

#### 先决条件:

如果需要的话，确保您的计算机具有因特网访问权，因为更新程序将从 IBM 服务器下载最新的文档修订包。要连接到因特网，您可能需要提供代理信息。

#### 过程:

要使用信息中心来更新本地 HTML 文档:

1. 启动 DB2 信息中心:
  - 从图形管理工具中，单击工具栏中的**信息中心**图标。
  - 在命令行中输入 db2ic。
2. 单击**信息中心** -> **更新本地文档**，以开始更新。  
如果有文档更新可用，可下载并应用它。

要手工下载并应用文档更新:

1. 在 Web 浏览器中打开 DB2 支持页面，网址为：  
[www.ibm.com/software/data/db2/udb/winos2unix/support](http://www.ibm.com/software/data/db2/udb/winos2unix/support)。
2. 单击 **DB2 V8** 并查找您的操作系统的文档修订包链接。
3. 通过将文档修订包级别与已安装的文档级别进行比较，从而确定本地 DB2 文档是否已过时。
4. 如果有更新的文档版本，则下载适用于您的操作系统的修订包。对于所有的 Windows 操作系统有一个修订包，对于所有的 UNIX 操作系统有一个修订包。
5. 应用修订包:
  - 对于 Windows 操作系统: 该文档修订包是自解压压缩文件。将该文档修订包下载到一个空目录然后解压缩。在该目录中运行 **setup** 命令来安装文档修订包。
  - 对于 UNIX 操作系统: 文档修订包是压缩的 tar.Z 文件。解压并解取 (untar) 文件以创建名为 `delta_install` 的目录。在该目录下运行脚本 `installdocfix` 以安装文档修订包。

## 搜索 DB2 文档

要搜索 DB2 文档，应使用 Netscape V6.1 (或更高版本) 或 Microsoft Internet Explorer V5 (或更高版本)。确保启用了浏览器的 Java 支持。

单击浏览器中的“DB2 Information Integrator 信息中心”的导航工具栏中的搜索图标时，将有一个搜索窗口打开。如果是第一次使用搜索功能，可能要花 1 分钟左右的时间来装入搜索窗口。

#### 限制:

当搜索文档时，将存在下列限制:

- 不支持布尔搜索。在搜索中，将忽略布尔搜索限定符 *and* 和 *or*。例如，下列搜索将产生相同的结果:
  - *servlets and beans*
  - *servlets or beans*
- 不支持通配符搜索。对 *java\** 的搜索将查找文字串 *java\**，而不会查找例如 *javadoc* 之类的字符串。

通常，搜索短语比搜索单个词可获得更好的搜索结果。

#### 过程:

要搜索 DB2 文档:

1. 在导航工具栏中，单击**搜索**。
2. 在“搜索”窗口最上面的文本输入字段中，输入两个或多个与您感兴趣的方面相关的术语，然后单击**搜索**。**结果**字段会显示一个主题列表，按准确性排序。  
输入的术语越多，查询的准确度就越高，由查询返回的主题数就越少。
3. 在**结果**字段中，单击想要阅读的主题的标题。该主题将会显示在内容框架中。

执行搜索后，第一个结果将自动装入到浏览器框架中。要查看其它搜索结果的内容，单击结果列表中的结果。

### 对于使用 Netscape 4.x 搜索 DB2 文档进行故障诊断

大多数搜索问题都与 Web 浏览器提供的 Java 支持有关。本主题提供可能的解决方案。

#### 过程:

一个 Netscape 4.x 常见问题是丢失和设置安全性类。尝试以下解决方案，尤其是当您在浏览器的 Java 控制台中看到以下行时更应尝试此方法:

找不到类 java/security/InvalidParameterException

将以下文件从 DB2 HTML 文档 CD 复制到您的 Netscape 浏览器的安装目录的 `java\classes\java\security\` 目录中。可能需要创建 `java\security\` 子目录结构。

- 在 Windows 操作系统上, 该文件为:

```
x:\Program Files\IBM\sqllib\doc\htmlcd\%L\InvalidParameterException.class
```

- 在 UNIX 操作系统上, 该文件为:

```
/cdrom/Program Files/IBM/sqllib/doc/htmlcd/%L  
/InvalidParameterException.class
```

其中:

- *x* 表示 CD-ROM 盘符 (Windows)
- *cdrom* 表示 CD-ROM 的安装点 (UNIX)
- *%L* 是要使用的文档的语言环境, 例如, `en_US`

如果 Netscape 浏览器仍无法显示搜索输入窗口, 则尝试下列操作:

- 停止 Netscape 浏览器的所有实例以确保计算机上无任何 Netscape 代码运行。然后, 打开 Netscape 浏览器的新实例并再次启动搜索。
- 清除浏览器的高速缓存。
- 尝试另一版本的 Netscape 或另一浏览器。



---

## 易使用性

身体有某些缺陷（如活动不方便或视力不太好）的用户可通过易使用性功能来成功使用软件产品。以下是 DB2 Information Integrator V8 中主要的易使用性功能：

- 通过键盘即可对所有功能进行操作，而不必使用鼠标。
- 您可以定制字体的大小和颜色。
- 您可以接收可视或声音警告提示。
- DB2 支持使用 Java™ Accessibility API 的易使用性应用程序。
- B2 文档以易使用的格式提供。

---

### 键盘输入和导航

只使用键盘就可对 DB2 数据库工具（如控制中心、数据仓库中心和复制中心）进行操作。使用键或组合键就可执行大多数操作，而不必使用鼠标。

在基于 UNIX 的系统中，键盘焦点的位置是突出显示的。此突出显示指示窗口的哪个区域处于活动状态且击键对何处生效。

---

### 易使用的界面显示

DB2 数据库工具中的功能增强了用户界面，使视力不太好的用户更易使用。这些易使用性方面的增强包括了对可定制字体属性的支持。

#### 字体设置

对于 DB2 数据库工具，可以使用“工具设置”笔记本来选择菜单和窗口中文本的颜色、大小和字体。

#### 与颜色的不相关性

不需要分辨颜色就可以使用此产品中的任何功能。

---

### 备用的警告提示

可使用“工具设置”笔记本来指定是否想要通过声音提示或可视提示接收警告。

---

## 与辅助技术的兼容性

DB2 Information Integrator 图形界面支持 Java Accessibility API，它支持使用屏幕阅读器以及其它由有某些缺陷的用户使用的辅助性技术。

---

## 可访问文档

DB2 产品系列的文档提供了 HTML 格式的版本。您可以根据浏览器中设置的显示首选项来查看文档。可使用屏幕阅读器和其它辅助性技术。

---

## 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证。您可以用书面方式将许可证查询寄往：

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

有关双字节（DBCS）信息的许可证查询，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, 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：**国际商业机器公司以“按现状”的基础提供本出版物，不附有任何形式的（无论是明示的，还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证；因此，本条款可能不适用于您。

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

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

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

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

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

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

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可证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，在其它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、其出版说明或其它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、兼容性或任何其它关于非 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。

本信息可能包含在日常业务操作中使用的数据和报告的示例。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：

本信息可能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、经销或分发的目的，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。

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 / 或其它国家或地区的商标：

IBM  
DB2

下列各项是其它公司的商标或注册商标：

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

Microsoft 和 Windows 是 Microsoft Corporation 在美国和 / 或其它国家或地区的商标。

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

其它公司、产品或服务名称可能是其它公司的商标或服务标记。



# 索引

## [ B ]

### 包装器类

- 列表 62
- Fenced\_Generic\_Wrapper 类 69
- Unfenced\_Generic\_Wrapper 62

## [ C ]

### 操作类

- 列表 121
- Remote\_Passthru 130
- Remote\_Query 121

### 成员函数

- 请求类 137
- 应答类 143
- Catalog\_Option 类 2
- Column\_Info 类 28
- Fenced\_Generic\_Nickname 类 109
- Fenced\_Generic\_Wrapper 类 69
- Fenced\_Generic\_Server 类 85
- Fenced\_Generic\_User 类 98
- Nickname\_Info 类 48
- Predicate\_List 类 173
- Remote\_Connection 类 114
- Remote\_Passthru 类 130
- Remote\_Query 类 121
- Request\_Constant 类 169
- Request\_Exp 类 159
- Request\_Exp\_Type 类 166
- Runtime\_Data 类 187
- Runtime\_Data\_Desc 类 179
- Runtime\_Data\_Desc\_List 类 184
- Runtime\_Data\_List 类 195
- Server\_Info 类 12
- Unfenced\_Generic\_Nickname 类 101
- Unfenced\_Generic\_Wrapper 类 62
- Unfenced\_Generic\_Server 类 75

### 成员函数 (续)

- Unfenced\_Generic\_User 类 92
- User\_Info 类 22
- Wrapper\_Info 类 4
- Wrapper\_Uilities 类 197

## [ F ]

### 服务器类

- 列表 75
- Fenced\_Generic\_Server 85
- Unfenced\_Generic\_Server 75

## [ G ]

### 构造函数

- 应答类 143
- Column\_Info 类 28
- Fenced\_Generic\_Nickname 类 109
- Fenced\_Generic\_Wrapper 类 69
- Fenced\_Generic\_Server 类 85
- Fenced\_Generic\_User 类 98
- Nickname\_Info 类 48
- Remote\_Connection 类 114
- Remote\_Passthru 类 130
- Remote\_Query 类 121
- Runtime\_Data\_Desc 类 179
- Server\_Info 类 12
- Unfenced\_Generic\_Nickname 类 101
- Unfenced\_Generic\_Wrapper 类 62
- Unfenced\_Generic\_Server 类 75
- Unfenced\_Generic\_User 类 92
- User\_Info 类 22
- Wrapper\_Info 类 4

## [ M ]

### 目录类

- 列表 1
- Catalog\_Option 2
- Column\_Info 28
- Nickname\_Info 48
- Server\_Info 12
- User\_Info 22
- Wrapper\_Info 4

## [ N ]

### 昵称类

- 列表 101
- Fenced\_Generic\_Nickname 109
- Unfenced\_Generic\_Nickname 101

## [ Q ]

### 请求类 137

- 列表 137
- 请求 137
- 应答 143
- Predicate\_List 173
- Request\_Constant 169
- Request\_Exp 159
- Request\_Exp\_Type 166

## [ S ]

### 数据成员

- Fenced\_Generic\_Nickname 类 109
- Fenced\_Generic\_Server 类 85
- Fenced\_Generic\_User 类 98
- Remote\_Connection 类 114
- Unfenced\_Generic\_Nickname 类 101
- Unfenced\_Generic\_Wrapper 类 62

## 数据成员 (续)

- Unfenced\_Generic\_Server 类 75
- Unfenced\_Generic\_User 类 92

## 数据类

- 列表 179
- Runtime\_Data 187
- Runtime\_Data\_Desc 179
- Runtime\_Data\_Desc\_List 184
- Runtime\_Data\_List 195

## [ X ]

### 析构函数

- Fenced\_Generic\_Wrapper 类 69
- Fenced\_Generic\_User 类 98
- Unfenced\_Generic\_Wrapper 类 62
- Unfenced\_Generic\_User 类 92

## [ Y ]

应答类 143

### 用户类

- 列表 92
- Fenced\_Generic\_User 98
- Unfenced\_Generic\_User 92

## C

Catalog\_Option 类 2

Column\_Info 类 28

C++ 类和方法 1

### C++ API

#### 包装器类

- 列表 62
- Fenced\_Generic\_Wrapper 69
- Unfenced\_Generic\_Wrapper 62

#### 操作类

- 列表 121
- Remote\_Passthru 130
- Remote\_Query 121

#### 服务器类

- 列表 75
- Fenced\_Generic\_Server 85
- Unfenced\_Generic\_Server 75

## C++ API (续)

### 目录类

- 列表 1
- Catalog\_Option 2
- Column\_Info 28
- Nickname\_Info 48
- Server\_Info 12
- User\_Info 22
- Wrapper\_Info 4

### 昵称类

- 列表 101
- Fenced\_Generic\_Nickname 109
- Unfenced\_Generic\_Nickname 101

### 请求类

- 列表 137
- 请求 137
- 应答 143
- Predicate\_List 173
- Request\_Constant 169
- Request\_Exp 159
- Request\_Exp\_Type 166

### 数据类

- 列表 179
- Runtime\_Data 187
- Runtime\_Data\_Desc 179
- Runtime\_Data\_Desc\_List 184
- Runtime\_Data\_List 195

### 用户类

- 列表 92
- Fenced\_Generic\_User 98
- Unfenced\_Generic\_User 92

## F

Fenced\_Generic\_Nickname 类 109

Fenced\_Generic\_Server 类 85

Fenced\_Generic\_User 类 98

Fenced\_Generic\_Wrapper 类 69

## N

Nickname\_Info 类 48

## P

Predicate\_List 类 173

## R

Remote\_Connection 类 114

Remote\_Passthru 类 130

Remote\_Query 类 121

Request\_Constant 类 169

Request\_Exp 类 159

Request\_Exp\_Type 类 166

Runtime\_Data 类 187

Runtime\_Data\_Desc 类 179

Runtime\_Data\_Desc\_List 类 184

Runtime\_Data\_List 类 195

## S

Server\_Info 类 12

## U

Unfenced\_Generic\_Nickname 类 101

Unfenced\_Generic\_Server 类 75

Unfenced\_Generic\_User 类 92

Unfenced\_Generic\_Wrapper 类 62

User\_Info 类 22

## W

Wrapper\_Info 类 4

Wrapper\_Uilities 类 197



---

## 与 IBM 联系

要在美国或加拿大联系 IBM，请致电以下号码之一：

- 客户服务中心：1-800-IBM-SERV (1-800-426-7378)
- DB2 市场营销和销售中心：1-800-IBM-4YOU (1-800-426-4968)

要了解所提供的服务项目，请致电以下号码之一：

- 在美国：1-888-426-4343
- 在加拿大：1-800-465-9600

要查找您所在国家或地区的 IBM 分部，可查看“IBM 全球联系人目录”（IBM Directory of Worldwide Contacts），网址为 [www.ibm.com/planetwide](http://www.ibm.com/planetwide)。

---

## 产品信息

还可通过电话或 Web 获取关于 DB2 Information Integrator 的信息。

您如果住在美国，请致电以下号码之一：

- 订购产品或获取一般信息，请致电：1-800-IBM-CALL (1-800-426-2255)
- 订购出版物，请致电：1-800-879-2755

在 Web 上，访问 [www.ibm.com/software/data/integration](http://www.ibm.com/software/data/integration)。此站点包含有关技术库、订购书籍、客户机下载、新闻组、修订包、新闻和 Web 资源链接的最新信息。

要查找您所在国家或地区的 IBM 分部，可查看“IBM 全球联系人目录”（IBM Directory of Worldwide Contacts），网址为 [www.ibm.com/planetwide](http://www.ibm.com/planetwide)。

---

## 关于文档的意见

您的反馈将有助于 IBM 提供质量信息。请发送您对该书或其它 DB2 Information Integrator 文档的任何意见。可使用以下任一方法提供意见：

- 使用在线读者意见表发送意见，网址为 [www.ibm.com/software/data/rcf](http://www.ibm.com/software/data/rcf)。
- 将您的意见以电子邮件方式发送至 [comments@us.ibm.com](mailto:comments@us.ibm.com)。确定包括产品名称、产品版本号以及书籍的名称和部件号（如果可应用的话）。如果您的意见针对特定文本，请包括该文本的位置（例如，标题、表号或页码）。







中国印刷

Spine information:



**IBM DB2 Information  
Integrator**

**包装器开发者 API 参考**

版本 8