



Net.Data

语言环境参考



Net.Data

语言环境参考

注意

在使用本信息及其所支持的产品之前，务必请阅读第77页的『附录B. 注意事项』中的信息。

目录

前言	v
关于 Net.Data	v
关于本书	v
谁应当阅读本书	vi
有关本书中的例子	vi
关于 Net.Data 语言环境	vii
第1部分 Net.Data 提供的语言环境	1
第1章 Net.Data 所提供语言环境的概述	3
第2章 使用 Net.Data 提供的语言环境	5
平面文件接口语言环境	5
IMS Web 语言环境	5
Java Applet 语言环境	6
创建 Java 小应用程序	6
生成小应用程序标记	6
Java 小应用程序的例子	9
使用 Net.Data Java 小应用程序接口	10
Java 应用程序语言环境	12
Java 语言环境文件结构	12
创建 Java 函数	13
定义 Java 语言环境 Cliette	13
为 Java 语言环境配置 Net.Data	14
创建并运行宏文件	14
ODBC 语言环境	15
Oracle 语言环境	15
Perl 语言环境	18
REXX 语言环境	19
SQL 语言环境	20
Sybase 语言环境	20
System 语言环境	22
Web 注册表语言环境	22
配置 Net.Data 语言环境	23
第2部分 非 IBM 的语言环境	25
第3章 创建一个新的语言环境	27
设计一个 DLL 或共享程序库	27
我应该提供哪些语言环境?	28
处理输入参数	28
处理用户请求	28
处理输出参数	29
通信错误条件	29
语言环境通信结构	29
dtw_lei 结构	29
dtw_parm_data 结构	31
语言环境接口函数	32

dtw_initialize()	33
dtw_execute()	33
dtw_getNextRow()	33
dtw_cleanup()	34
设计语言环境语句	34
ENVIRONMENT 语句语法	35
ENVIRONMENT 语句例子	35
第4章 语言环境程序设计接口应用函数	37
语言环境应用函数	37
用于管理内存的应用函数	37
用于管理配置变量的应用函数	37
用于表格操作的应用函数	38
用于行操作的应用函数	38
应用函数语法参考	39
dtw_free()	40
dtw_getvar()	41
dtw_malloc()	42
dtw_row_SetCols()	43
dtw_row_SetV()	44
dtw_strdup()	45
dtw_table_AppendRow()	46
dtw_table_Cols()	47
dtw_table_Delete()	48
dtw_table_DeleteCol()	49
dtw_table_DeleteRow()	50
dtw_table_GetN()	51
dtw_table_GetV()	52
dtw_table_InsertCol()	53
dtw_table_InsertRow()	54
dtw_table_MaxRows()	55
dtw_table_New()	56
dtw_table_QueryColnoNj()	57
dtw_table_Rows()	58
dtw_table_SetCols()	59
dtw_table_SetN()	60
dtw_table_SetV()	61
附录A. 语言环境模板	63
附录B. 注意事项	77
商标	78
词汇表	79
索引	81

前言

感谢您选择 Net.Data 版本 2 - IBM 的开发工具来创建动态 Web 页面! 使用 Net.Data 之后, 您就可以迅速地开发具有动态内容的 Web 页面, 这只要通过结合来自各种不同数据源的数据并使用您已知的程序设计语言的功能即可实现。

Net.Data 版本 2 提供了显著改进的性能和一些新的功能, 这些新功能将赋予您构建与采纳自己的 Internet 商业方案的能力。

关于 Net.Data

采用 IBM 的 Net.Data 产品之后, 您就可以使用来自关系型或非关系型数据库管理系统 (DBMS, 包括 DB2、IMS 和 允许使用 ODBC 的数据库)的数据来创建动态的 Web 页面; 还可以使用各种编程语言(例如 Java、JavaScript、Perl、C、C++ 和 REXX) 所编写的应用程序。

您可以将 Net.Data 看作是一个宏处理器, 在 Web 服务器上作为中件执行。您可以编写 Net.Data 应用程序(称之为宏), Net.Data 将对它进行解释以便使用根据用户输入、数据库当前状态、现有商业逻辑以及您在宏中所设计的其它因素而定制的内容来创建动态的 Web 页面。

一个 URL (统一资源定位器)形式的请求, 从浏览器(例如 Netscape 或 Internet Explorer)流动到将请求转发给 Net.Data 进行执行的 Web 服务器, Net.Data 找出这个宏加以执行, 并构建一个根据您所编写的函数定制的 Web 页面。这些函数能够:

- 在 Perl 脚本、C 与 C++ 应用程序或 REXX 程序中封装商业逻辑
- 访问诸如 DB2 等数据库

Net.Data 支持诸如超文本传输协议 (HTTP) 和公共网关接口 (CGI) 等工业标准接口。HTTP 用在浏览器和 Web 服务器之间, 而 CGI 用在 Web 服务器和 Net.Data 之间。这样您就可以选择您所喜爱的浏览器或 Web 服务器与 Net.Data 一起使用。Net.Data 还支持多个操作系统上的 FastCGI 和主要的 Web 服务器 API。

关于本书

本书讨论 Net.Data 的语言环境, 它们在您从 Net.Data 宏文件调用程序或函数时使用, 或在调用诸如 DB2、Oracle 或 Sybase 数据库时使用。书中描述了 Net.Data 提供的每个语言环境, 还描述了您可用于设计与构建自己的语言环境的语言环境接口。

本书可能引用已经发布、但现在还未进入实用的某些产品或功能。

包括示例 Net.Data 宏、演示程序以及本书最新副本在内的更多信息, 可以从以下 World Wide Web 站点获得:

- <http://www.software.ibm.com/data/net.data>
- <http://www.as400.ibm.com/netdata>

谁应当阅读本书

编写 *Net.Data* 宏的人可以使用此信息来了解有关 *Net.Data* 所提供的语言环境的功能。本书还为希望编写自己的 *Net.Data* 语言环境的人提供了信息。

要理解本书中讨论的概念，您应当熟悉 C 编程语言以及*Net.Data* 管理与程序设计指南和*Net.Data* 参考中的信息。

有关本书中的例子

本书中出现的例子相对较为简单，目的是演示特定的概念，而没有考虑各种可能的情况。某些例子只是一些片段，不能单独运行。

关于 Net.Data 语言环境

Net.Data 被设计成允许按照可接插方式来添加新的编程语言和数据库接口。这些接口称为语言环境，它们是作为 DLL 或共享程序库被访问的。语言环境提供对支持动态 Web 页面的应用程序和数据库的访问。通过使用函数调用以及 SQL 语句来调用语言环境，您可以访问这些语言环境为您的商业应用所提供的功能与实用程序。例如，您可以直接访问您的 ODBC 数据库来运行 Java 小应用程序，也可以使用 Perl 语言环境调用 Perl 脚本来运行 Java 小应用程序，还可以调用 Java Applet 语言环境来运行 Java 小应用程序。

Net.Data 初始化文件将每个语言环境的名称与一个 DLL 或共享程序库关联起来。每种语言环境必须支持由 Net.Data 定义的一套标准接口。在首次遇到对指定该语言环境的 FUNCTION 块的函数调用时，Net.Data 将装入初始化文件中指定的 DLL 或共享程序库。

Net.Data 分析 Net.Data 宏，维护 Net.Data 变量，与语言环境通信，并根据 REPORT 和 MESSAGE 块说明来格式化输出。语言环境支持定义给 Net.Data 的接口，使得语言处理器能以某种独立于语言的方式访问 Net.Data 参数，调用语言解释程序，并以某种独立于语言的方式接收从语言解释程序返回的变量。

图1 演示了 Net.Data 与语言环境之间的交互。

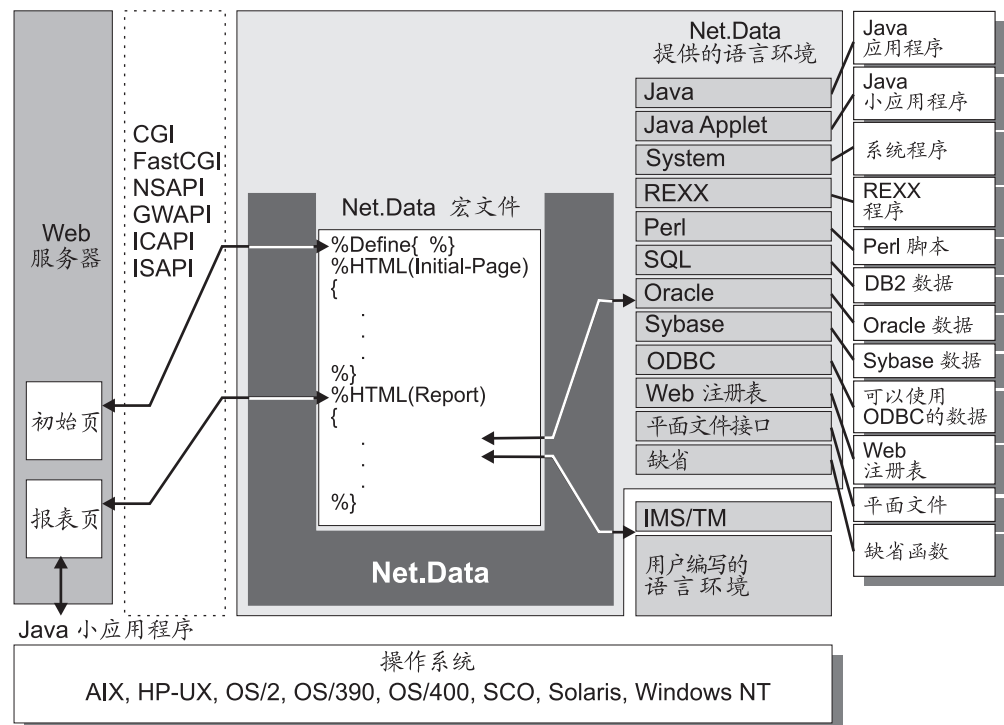


图 1. Net.Data 和语言环境

在 Net.Data 应用程序中使用语言环境涉及两类任务。

- 使用 Net.Data 提供的语言环境来开发 Net.Data 应用程序。
- 在开发 Net.Data 应用程序时为其他用户开发可以使用的新的语言环境或数据库环境。

本书的组织有助于您完成这两个任务:

- 第1页的『第1部分 Net.Data 提供的语言环境』描述了可以与您的 Net.Data 应用程序一起使用的 Net.Data 所提供的语言环境。
- 第25页的『第2部分 非 IBM 的语言环境』描述了如何创建新的语言环境和数据库环境。

第1部分 Net.Data 提供的语言环境

Net.Data 提供了多种语言环境，可以让您在数据源间来回传送信息。例如，用于 SQL 的语言环境可让您将本机的 SQL 查询传送到一个数据库。同样，REXX 语言环境可让您调用 REXX 程序。

这一部分将描述每一种语言环境以及如何在 Net.Data 初始化文件中配置语言环境。

第3页的『第1章 Net.Data 所提供语言环境的概述』

第5页的『平面文件接口语言环境』

第5页的『IMS Web 语言环境』

第6页的『Java Applet 语言环境』

第12页的『Java 应用程序语言环境』

第15页的『ODBC 语言环境』

第15页的『Oracle 语言环境』

第18页的『Perl 语言环境』

第19页的『REXX 语言环境』

第20页的『SQL 语言环境』

第20页的『Sybase 语言环境』

第22页的『System 语言环境』

第22页的『Web 注册表语言环境』

第23页的『配置 Net.Data 语言环境』

第1章 Net.Data 所提供语言环境的概述

Net.Data 提供了许多语言环境，尽管有些操作系统并不支持所有的环境。表1列出了 IBM 提供的语言环境。要确定您的操作系统上是否支持某个语言环境，请参阅 *Net.Data* 参考中的操作系统引用附录。关于您的操作系统所使用的语言环境语句的细节，请参阅 *Net.Data* 自述文件或“程序目录”。

表 1. *Net.Data* 语言环境

语言环境	环境语句	说明
平面文件接口	DTW_FILE	平面文件接口 (FFI) 提供了支持文本文件作为数据源的函数。
IMS Web	HWS_LE	IMS Web 语言环境可让您使用 IMS Web 来提交 IMS 事务，并从 Web 浏览器接收该事务的输出。
Java Applet	DTW_APPLET	Java applet 语言环境可让您在自己的 Net.Data 应用程序中使用 Java 小应用程序。要生成一个小应用程序标记，您必须提供该小应用程序标记的限定符以及小应用程序的参数表。
Java 应用程序	DTW_JAVAPPS	Net.Data 支持带有 Java 语言环境的现有 Java 应用程序。
ODBC	DTW_ODBC	ODBC 语言环境通过一个 ODBC 接口来执行 SQL 语句，以便访问多个数据库管理系统。
Oracle	DTW_ORA	Oracle 语言环境可以让您直接访问您的 Oracle 数据。
Perl	DTW_PERL	Perl 语言环境解释 Net.Data 宏中的 FUNCTION 块内指定的内部 Perl 脚本，或执行存储在单独文件中的外部 Perl 脚本。
REXX	DTW_REXX	REXX 语言环境解释 Net.Data 宏中 FUNCTION 块内指定的内部 REXX 程序，或可以执行存储在一个单独文件中的 REXX 程序。
SQL	DTW_SQL	SQL 语言环境通过 DB2 执行 SQL 语句。SQL 语句的结果可以在表格变量中返回。
Sybase	DTW_SYB	Sybase 语言环境可以让您直接访问您的 Sybase 数据。
System	DTW_SYSTEM	System 语言环境支持对某些外部程序的调用，这些外部程序在 FUNCTION 块内的一条 EXEC 语句中标识。System 语言环境通过将程序名及其参数传递到执行程序操作系统来解释 EXEC 程序。
Web 注册表	DTW_WEBREG	Web 注册表语言环境为应用程序相关数据的永久性存储器提供函数。

每个语言环境都需要在初始化文件中有一个 ENVIRONMENT 语句，在服务器的 /lib 或 /d11 目录中有一个共享程序库或 DLL 文件。请参阅 *Net.Data* 管理与程序设计指南中配置一章，以获取更多信息。

建议：在进行更改之前先为您的初始化文件制作一个备份。

第2章 使用 Net.Data 提供的语言环境

以下几节将描述 Net.Data 提供的语言环境以及设置和使用这些语言环境的步骤。

平面文件接口语言环境

如果您选择使用平面文件(或明文文件)作为数据源,则使用平面文件接口 (FFI) 及其关联的函数来打开、关闭、读取、写入或删除 Web 服务器上的文件。您必须在初始化文件中为 FFI_PATH 变量指定一个路径。

文件语言支持根据 Web 客户的请求,通过浏览器使用 FFI 函数来读取或写入 Web 服务器上的文件。FFI 将文件看作记录文件,每个记录都等价于 Net.Data 宏表格变量中的一行,而记录中的每个值则等价于 Net.Data 宏表格变量中的一个字段值。FFI 从文件中将记录读至一个 Net.Data 宏表格的行中,并将行从表格写至记录中。

请参阅 *Net.Data* 参考一书,以获取有关 FFI 内部函数的描述和语法。

IMS Web 语言环境

IMS Web 语言环境是在 World Wide Web 环境中运行 IMS 事务的一个完整的端对端解决方案的一部分。IMS Web 语言环境提供:

- 一个带有以下内容的 Net.Data 宏:
 - 用于输入事务输入数据的 HTML
 - 一个调用 IMS Web 语言环境的 Net.Data FUNCTION 块
 - 显示事务输出的 HTML
- IMS Web 语言环境所调用的事务 DLL 或共享程序库

限制: 只有当 Net.Data 作为 CGI 应用程序运行时,才支持 Net.Data 的 IMS Web 语言环境。使用 ICAPI 的 Net.Data 不支持该语言环境。

IMS Web Studio 工具从事务的(报文格式服务) MFS 源为 DLL 和宏生成代码,并为构建 DLL 或共享程序库生成一个 MAK 文件。在构建了 DLL 的可执行形式之后, DLL 和宏文件将被移动到运行 Net.Data 的 Web 服务器上。此时,事务已经可以在 Web 环境中运行了。

要使用 IMS Web 语言环境:

1. 在运行 Net.Data 的 Web 服务器上安装 IMS Web Runtime 组件。有关 IMS Web Runtime 组件的信息,请参阅 *IMS Web 用户指南*:
<http://www.software.ibm.com/data/ims/about/imsweb/document/index.html>
2. 创建事务 DLL 文件。
 - a. 使用 IMS Web Studio 工具为您的事务生成 C++、MAK 和 Net.Data 宏文件。
 - b. 如果您在不同于 IMS Web Studio 工具所运行的操作系统上运行 Net.Data,那么您需要将 DLL 源文件移动到运行目标操作系统的 IMS Web 开发机器上。例如,如果您在 Windows NT 上运行 IMS Web Studio 工具,而目标平台是 AIX 或 OS/390,则需要将事务 DLL 的源文件分别移动到在 AIX 或 OS/390 下运行的 IMS Web 开发机器上。

- c. 使用生成的 MAK 文件来构建事务 DLL 的可执行形式。
3. 将事务 DLL 文件 (DTWproj.dll) 和 Net.Data 宏文件 (DTWproj.d2w) 复制到您的 Web 服务器。
 - a. 将该宏放置到一个 Net.Data 从中检索宏的目录中。(请参阅 *Net.Data 管理与程序设计指南*中配置一章中的 MACRO_PATH 语句。)
 - b. 将事务 DLL 或共享程序库放在 Web 服务器从中检索 DLL 或共享程序库的目录中。
4. 使用 IMS Web Studio 工具所生成的示例文件中的链接 DTWproj.htm 来修改 Web 服务器的 HTML 树中的 HTML 文件。然后, 您可以使用这个链来调用 Net.Data 并在 Web 浏览器上显示对这个事务所输入的 HTML 格式。填充事务输入, 然后选择表格中的 SUBMIT 按钮来运行该事务并在 Web 浏览器上接收其输出。

IMS Web 使用 IMS TCP/IP Open Transaction Manager Access (OTMA) Connection 在 Web 服务器和 IMS 环境之间通信。请参阅 IMS Web 的主页以获取更多信息:

<http://www.software.ibm.com/data/ims/about/imsweb>

Java Applet 语言环境

Java applet 语言环境可让您在自己的 Net.Data 应用程序中方便地为 Java 小应用程序生成 HTML 标记。当您调用 Java applet 语言环境时, 需要指定小应用程序的名称并传送小应用程序所需的全部参数。语言环境将处理宏并生成 HTML 小应用程序标记, Web 浏览器使用这些标记来运行该小应用程序。

另外, Net.Data 提供了一系列接口, 您的小应用程序可以用它们来访问表格参数。这些接口包含在 DTW_Applet.class 类中。

以下章节将描述如何使用 Java applet 语言环境来运行您的 Java 小应用程序。

创建 Java 小应用程序

使用 Net.Data Java applet 语言环境之前, 您需要确定您计划使用哪些小应用程序或者需要编写哪些小应用程序。请参阅您的 Java 文档以获取更多有关创建小应用程序的信息。

生成小应用程序标记

您使用 Net.Data 函数调用来指定对 applet 语言环境的调用。对于这个函数调用不需要任何说明。函数调用的语法如下:

@DTWA_AppletName(parm1, parm2, ..., parmN)

- DTWA_ 用于标识对 applet 语言环境的函数调用。
- AppletName 是为其生成标记的小应用程序名。
- parm1 到 parmN 是用于生成 PARAM 标记的参数。

要编写一个生成小应用程序标记的宏文件:

1. 在宏文件的 DEFINE 部分定义小应用程序所需的全部参数。这些参数包括小应用程序标记属性、Net.Data 变量、以及您需要作为小应用程序输入的 Net.Data 表格参数。
例如:


```
%define{
DATABASE = "celdial"                                <=Net.Data variable: name of the database
MyGraph.codebase = "/netdata-java/" <=Required applet parameter
MyGraph.height = "200"                               <=Required applet parameter
MyGraph.width = "400"                                <=Required applet parameter
MyTitle = "This is my Title"                         <=Net.Data variable: name of the Web page
MyTable = %TABLE(all)                                <=Table to store query results
%}
```

2. 可选项: 指定一个对数据库的查询, 以便生成一个作为小应用程序输入的结果集。在您使用一个生成图表或表格的小应用程序时, 这是相当有用的。例如:

```
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
```

3. 在 `Net.Data` 宏中指定函数调用来调用 Java applet 语言环境以及调用该小应用程序。函数调用中要指定小应用程序的名称和您希望传送给语言环境的参数。这些参数包括 `Net.Data` 变量、以及您需要作为小应用程序输入的 `Net.Data` 表格参数或列参数。

例如:

```
%HTML(report){                                     <=The start of the HTML block
@mySQL(MyTable)                                     <=A call to the SQL function
                                                    mySQL
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable) <=Applet function call
%}
```

4. 调用 `Net.Data` 并运行宏文件。请参阅 *Net.Data 管理与程序设计指南*, 以了解如何调用 `Net.Data`。

小应用程序标记属性

您可以在 `Net.Data` 宏中的任何地方为小应用程序标记指定属性。 `Net.Data` 将所有形如 `AppletName.attribute` 的变量替代到小应用程序标记中, 作为属性。对小应用程序标记定义属性的语法是这样的:

```
%define AppletName.attribute = "value"
```

这些属性是所有小应用程序都需要的:

- *codebase*: 小应用程序的位置, 由 URL 来标识。
- *height*: 小应用程序的高度(以像素为单位)。
- *width*: 小应用程序的宽度(以像素为单位)。

例如, 如果您的小应用程序名为 `MyGraph`, 那么您可以定义这些必需的属性:

```
%DEFINE{
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
%}
```

实际的赋值不需要在 `DEFINE` 部分完成。您可以使用 `DTW_ASSIGN` 函数来设置值。如果您没有为 `AppletName.code` 变量定义一个变量, 则 `Net.Data` 将在小应用程序标记中添加一个缺省的 `code` 参数。 `code` 参数的值为 `AppletName.class`, 其中 `AppletName` 是您的小应用程序名。

小应用程序标记参数

您定义一个参数列表，它们在函数调用中传递给 Java applet 语言环境。您可以传递的参数包括：

- Net.Data 变量(包括 LIST 变量)
- Net.Data 表格
- Net.Data 表格中的列

传递参数时，Net.Data 将使用您为该参数指定的名称和值在 HTML 输出中创建一个 Java 小应用程序 PARAM 标记。不能传递字符串文字或函数调用的结果。

Net.Data 变量参数： 您可以将 Net.Data 变量用作参数。如果您在宏的 DEFINE 块中定义一个变量并在 DTWA_AppletName 函数调用中传送该变量，那么 Net.Data 将生成一个名称与值都与该变量相同的 PARAM 标记。例如，给出以下宏语句：

```
%define{  
  
...  
  
MyTitle = "This is my Title"  
%}  
  
%HTML (report){  
@DTWA_MyGraph( MyTitle, ...)  
%}
```

Net.Data 产生以下小应用程序的 PARAM 标记：

```
<param name = 'MyTitle' value = "This is my Title" >
```

Net.Data 表格参数：

每次调用 Java applet 语言环境时，Net.Data 都将使用名称 DTW_NUMBER_OF_TABLES 来自动生成一个 PARAM 标记，从而指定该函数调用是否传送表格变量。它的值是 Net.Data 在函数中使用的表格变量的个数。如果在函数调用中没有指定表格变量，则将生成以下标记：

```
<param name = "DTW_NUMBER_OF_TABLES" value = "0" >
```

在函数调用时，您可以将一个或多个 Net.Data 表格变量作为参数来传递。如果您在 DTWA_AppletName 函数调用中指定了一个 Net.Data 表格变量，则 Net.Data 将生成以下 PARAM 标记：

表名参数标记

此标记指定要传递的表格的名称。它具有以下语法：

```
<param name = 'DTW_TABLE_i_NAME' value = "tname" >
```

其中 i 是表格的个数(根据函数调用的次序)，tname 是表格的名称。

行和列说明参数标记：

生成 PARAM 标记是为了指定特定表格中的行数或列数。它具有以下语法：

```
<param name = 'DTW_tname_NUMBER_OF_ROWS' value = "rows" >  
<param name = 'DTW_tname_NUMBER_OF_COLUMNS' value = "cols" >
```

其中表格的名称为 *tname*, *rows* 是表格中的行数, *cols* 是表格中的列数。对于函数调用中指定的每个唯一的表格都将生成这样一个标记对。

列值参数标记:

PARAM 标记指定一个特定列的列名。它具有以下语法:

```
<param name = 'DTW_tname_COLUMN_NAME_j' value = "cname" >
```

其中表格的名称为 *tname*, *j* 是列数, *cname* 则是表格中列的名称。

行值参数标记:

PARAM 标记指定一个特定行与列中的值。它具有以下语法:

```
<param name = 'DTW_tname_cname_VALUE_k' value = "val" >
```

其中表格名称为 *tname*, *cname* 是列名, *k* 是行数, *val* 是与相应行和列中的值匹配的值。

表格列参数: 您可以在函数调用时传递一个表格列(将它作为参数), 从而为特定的列生成标记。Net.Data 仅对于指定的列生成相应的小应用程序标记。表格列参数使用以下语法:

```
@DTWA_AppletName(DTW_COLUMN( x )Table)
```

其中 *x* 是表格中的列名或列号。

表格列参数使用为表格参数定义的小应用程序标记。

不支持 Java 功能的浏览器上小应用程序标记的替换文本

变量 DTW_APPLET_ALTTEXT 指定了显示在不支持 Java 的浏览器或关闭了 Java 支持的浏览器上的文本。例如, 以下变量定义:

```
%define DTW_APPLET_ALTTEXT = "<P>Sorry, your browser is not Java-enabled."
```

将产生以下 HTML 标记和文本:

```
<P>Sorry, your browser is not Java-enabled.<BR>
```

如果没有定义这个变量, 则不显示任何替换文本。

Java 小应用程序的例子

下面的例子演示了调用 Java applet 语言环境的 Net.Data 宏文件以及该语言环境生成的结果小应用程序标记。

Net.Data 宏文件中包含以下对 Java applet 语言环境的函数调用:

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<P>Sorry, your browser is not Java-enabled."
DTW_DEFAULT_REPORT = "no"
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
MyTitle = "This is my Title"
%}
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
```

```
%HTML (report){
@mySQL(MyTable)
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
%}
```

DEFINE 部分的 Net.Data 宏定义行指定了小应用程序标记的第一行:

```
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
```

语言环境使用以下限定符生成了一个小应用程序标记:

```
<applet code = 'MyGraph.class'
codebase = '/netdata-java/' width = '400'
height = '200' >
```

Net.Data 从 Net.Data 宏文件的 SQL 部分返回 SQL 查询的结果, 结果放在输出表 MyTable 中。此表格在 DEFINE 部分指定:

```
MyTable = %TABLE(all)
```

宏当中对小应用程序的调用是在 HTML 部分指定的:

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

根据函数调用中的参数, Net.Data 生成完整的小应用程序标记, 其中包含有关结果表格的信息, 例如: 列数、返回的行数以及结果行。 Net.Data 为结果表格中的每个单元生成一个参数标记, 如下面这个例子所示:

```
param name = 'DTW_MyTable_ages_VALUE_1' value = "35">
```

参数名称 *DTW_MyTable_ages_VALUE_1* 指定了表格 MyTable 中的表格单元(行 1, 列 ages), 其值为 4。在对小应用程序的函数调用中的关键字 DTW_COLUMN 指定了您只对结果表格 MyTable 中的列 age 感兴趣, 如这里所示:

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

以下输出显示了 Net.Data 为上述例子生成的完整的小应用程序标记。

```
<applet code = 'MyGraph.class'
codebase = '/netdata-java/' width = '400' height = '200' >
<param name = 'MyTitle' value = "This is my Title" >
<param name = 'DTW_NUMBER_OF_TABLES' value = "1" >
<param name = 'DTW_TABLE_1_NAME' value = "MyTable" >
<param name = 'DTW_MyTable_NUMBER_OF_ROWS' value = "5" >
<param name = 'DTW_MyTable_NUMBER_OF_COLUMNS' value = "1" >
<param name = 'DTW_MyTable_COLUMN_NAME_1' value = "ages" >
<param name = 'DTW_MyTable_ages_VALUE_1' value = "35">
<param name = 'DTW_MyTable_ages_VALUE_2' value = "32">
<param name = 'DTW_MyTable_ages_VALUE_3' value = "31" >
<param name = 'DTW_MyTable_ages_VALUE_4' value = "28" >
<param name = 'DTW_MyTable_ages_VALUE_5' value = "40" >
<P>Sorry, your browser is not Java-enabled.<BR>
</applet>
```

使用 Net.Data Java 小应用程序接口

Net.Data 在一个名为 DTW_Applet.class 的类中提供了一系列接口, 它们可以和您的 Java 小应用程序一起使用, 帮助处理为表格变量生成的 PARAM 标记。您可以创建一个扩充此接口的小应用程序, 用于从您的小应用程序调用例程。

Net.Data 提供了这些接口:

- **int GetNumberOfTables()** 返回在小应用程序标记中找到的表格个数。
- **String [] GetTableNames()** 返回一个在小应用程序标记中找到的表名的列表。
- **int GetNumberOfColumns(String table_name)** 返回表格 table_name 中的列数。
- **int GetNumberOfRows(String table_name)** 返回表格 table_name 中的行数。
- **String[] GetColumnNames(String table_name)** 返回表格 table_name 中的列名。
- **String[][] GetTable(String table_name)** 返回一个二维的字符串数组, 这些字符串中包含表格的行列值。

要访问接口, 请在您的小应用程序代码中使用 EXTENDS 关键字来把您的小应用程序从 DTW_APPLET 类中归为子类, 如下面的例子所示:

```
import java.io.*;
import java.applet.Applet;

public class myDriver extends DTW_Applet
{
    public void init()
    {
        super.init();

        if (GetNumberOfTables() > 0)
        {
            String [] tables = GetTableNames();
            printTables(tables);
        }
    }

    private void printTables(String[] tables)
    {
        String table_name;

        for (int i = 0; i < tables.length; i++)
        {
            table_name = tables[i];
            printTable(table_name);
        }
    }

    private void printTable(String table_name)
    {
        int nrows = GetNumberOfRows(table_name);
        int ncols = GetNumberOfColumns(table_name);

        System.out.println("Table: " + table_name + " has " + ncols + " columns and "
            + nrows + " rows.");

        String [] col_names = GetColumnNames(table_name);

        System.out.println("-----");

        for (int i = 0; i < ncols; i++)
            System.out.print("    " + col_names[i] + "    ");
        System.out.println("\n-----");

        String [][] mytable = GetTable(table_name);

        for (int j = 0; j < nrows; j++)
        {
            for (int i = 0; i < ncols; i++)
                System.out.print("    " + mytable[i][j] + "    ");
        }
    }
}
```

```

        System.out.println("\n");
    }
}

```

Java 应用程序语言环境

Net.Data 支持带有 Java 语言环境的现有 Java 应用程序。采用对 Java 小应用程序和 Java 方法(或应用程序)的支持后,您可以通过 Java 数据库连接 (JDBC**) API 来访问 DB2。

有关 JDBC 的详细信息,可从这些站点获得:

- 具有 JDK 1.1 或更高版本的 IBM 软件,在使用 JDBC 和 Net.Data 时需要:
<http://www.software.ibm.com/data/db2/jdbc/>
- JavaSoft 具有额外的 JDBC 驱动程序、JDBC API 文档和 JDBC 的最新更新:
<http://splash.javasoft.com/jdbc/>

Java 语言环境提供了一个类似于远程过程调用 (RPC) 的接口。您可以从 Net.Data 宏文件发出 Java 函数调用(将 Net.Data 字符串作为参数),您所调用的 Java 函数将返回一个字符串。当您使用 Java 语言环境时,必须使用 Net.Data 现场连接(请参阅 *Net.Data 管理与程序设计指南*中的性能一章,以获取有关现场连接的更多信息)。要使用 Java 语言环境,必须完成以下步骤。这些步骤将在后面的章节中详细描述。

1. 编写您的 Java 函数。
2. 为所有的 Java 函数创建一个 Net.Data cliette (Net.Data cliette 启动您运行 Java 函数的 Java 虚拟机)。
3. 在现场连接配置文件中定义一个 cliette 语句。
4. 启动连接管理器。
5. 运行调用 Java 语言环境的 Net.Data 宏文件。

每当您引入新的 Java 函数时,都必须重新创建 Java cliette。

Java 语言环境文件结构

Net.Data 在安装期间创建了几个目录。这些目录中包括您使用 Java 语言环境来创建 Java 函数、定义 cliette 和运行宏所需的文件:

- 一个名为 UserFunctions.java 的示例 Java 函数。
- 一个名为 makeClas 的示例文件。在运行时,这个文件将为您的 Java 函数创建一个 Net.Data cliette。
- 一个名为 launchjv 的示例文件,Net.Data cliette 用它来启动 Java 虚拟机并运行 Java 函数。

表2 描述了您操作系统上那些文件的目录与文件名。

表 2. 创建 Java 函数时所使用的文件

操作系统	文件名	目录
OS/2	UserFunctions.java	javaapps
	launchjv.com	connect

表 2. 创建 Java 函数时所使用的文件 (续)

操作系统	文件名	目录
Windows NT	UserFunctions.java	javaclas
	makeClas.bat	javaclas
	launchjv.bat	connect
UNIX	UserFunctions.java	javaapps
	launchjv	javaapps

创建 Java 函数

修改 Java 函数示例文件 UserFunctions.java, 或者以下面的示例文件为模型来创建一个新的文件 myfile.java:

```
=====myfile.java=====
import mypackage.*                                <=contain your functions
public String myfctcall(...parameters from macro file...)
{
    return ( mypackage.mymethod(...parameters...));    <=high-level call to your functions
}

public String lowlevelcall(...parameters...)
{
    string result;
    .....code using many functions of your package...
    return(result)
}
```

定义 Java 语言环境 Cliette

修改示例文件 makeClas.bat, 或者为所有的 Java 函数创建一个新的 .bat 文件来生成一个名为 dtw_samp.class 的 Net.Data cliette 类。下面的例子显示批处理文件 CreateServer 如何处理三个 Java 函数:

```
rem Batch file to create dtw_samp for Net.Data
java CreateServer dtw_samp.java UserFunctions.java myfile.java
javac dtw_samp.java
```

批处理文件处理以下文件和 Net.Data 提供的名为 Stub.java 的 stub 文件, 用于处理 dtw_samp.class。

- dtw_samp.java
- UserFunctions.java
- myfile.java

编写一个 JDBC 应用程序或小应用程序与编写一个使用 DB2 CLI 或 ODBC 来访问数据库的 C 应用程序是非常类似的。应用程序和小应用程序之间最主要的区别是: 应用程序可能需要特殊的软件来与 DB2 通信, 例如 DB2 Client Application Enabler。而小应用程序则取决于允许 Java 功能的 Web 浏览器, 但不需要安装在客户上的任何 DB2 代码。

在使用 JDBC 之前, 您的系统需要进行一些配置。在 DB2 JDBC 应用程序和小应用程序支持 Web 站点中讨论了这些需要考虑的事项:

<http://www.software.ibm.com/data/db2/jdbc/db2java.html>

为 Java 语言环境配置 Net.Data

要使用 Java 语言环境，您必须配置 Net.Data。使用以下步骤来完成配置：

1. 创建一个启动 Java 应用程序的批处理文件，因为 Net.Data 无法直接启动 Java 应用程序。Net.Data 使用这个文件来启动运行 Java 函数的 Java 虚拟机。批处理文件中必须包括 `java-classpath` 语句，以便确保可以找到必需的 Java 包(标准包与特定于应用程序的包)。例如，批处理文件 `launchjv.bat` 中包含以下 `java-classpath`：

```
java -classpath %CLASSPATH%;C:\DB2WWW\Javaclas dtw_samp %1 %2 %3 %4 %5 %6
```

2. 在现场连接配置文件 `dtwcm.cnf` 中定义一个 `cliette` 来与 Java 语言环境一起工作。为这个 `cliette` 指定一个唯一的端口号码，并用 `EXEC_NAME` 配置变量来指定相关的批处理文件。在下面的例子中，Java `cliette` 名称被定义为 `DTW_JAVAPPS`，`EXEC_NAME` 配置变量被设置为批处理文件的名称 `launchjv.bat`：

```
CLIETTE DTW_JAVAPPS{
MIN_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
MAX_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
START_PRIVATE_PORT=5100      <= Must be a unique port number
START_PUBLIC_PORT=5300       <= Must be a unique port number
EXEC_NAME=launchjv.bat       <= The name of the batch file that includes the
                               classpath statements
}
```

启动 Net.Data 连接管理器时，Net.Data 将启动配置文件中指定的 Java `cliette`。`cliette` 变得可用之后就您的 Net.Data 宏应用程序处理 Java 语言环境请求。

3. 通过在 Net.Data 初始化文件 `db2www.ini` 中的 `DTW_JAVAPPS ENVIRONMENT` 路径语句中添加每个 `cliette` 的名称来更新该语句。例如：

```
ENVIRONMENT DTW_JAVAPPS ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"
```

创建并运行宏文件

在创建了 Java 函数、定义了 `cliette` 类并配置了 Net.Data 之后，您就可以运行包含对 Java 函数引用的宏文件。

1. 创建一个调用您的 Java 函数的宏文件。例如，函数调用 `myfctcall` 使用 `cliette DTW_JAVAPPS` 来调用 Net.Data 提供的示例函数。

```
%function (DTW_JAVAPPS) myfctcall( ....parameters from macro file ....)
```

```
%{ to call the sample provided with Net.Data %}
```

```
%function (DTW_JAVAPPS) reverse_line1(str);
```

```
%HTML (report){
```

```
您应当看到反转显示的字符串 "Hello World"。
```

```
@reverse_line("Hello World")
```

```
您应当具有函数调用的结果。
```

```
@myfctcall( ... ....)
```

```
%}
```

2. 启动连接管理器。请参阅 *Net.Data 管理与程序设计指南* 中的性能一章，以获取有关连接管理器的更多信息。
3. 使用一个 HTML 链、HTML 格式或 URL 语句来启动宏并调用 Net.Data。例如，使用以下 URL 语句来调用 Net.Data 宏文件 `mymacro.d2w`：

ODBC 语言环境

开放数据库连接 (ODBC) 语言环境通过一个 ODBC 接口执行 SQL 语句。ODBC 是基于 X/Open SQL CAE 规格说明的，它允许单一的应用访问多个数据库管理系统。

要使用 ODBC 语言环境，您必须有一个 ODBC 驱动程序和一个驱动程序管理器。您的 ODBC 文档描述了如何安装和配置 ODBC 环境。

在 ODBC 环境在发送 SQL 语句类似于其它的 Net.Data 函数。下面是 Net.Data 宏的一个例子，这个宏将多个 SQL 语句发送到作为您的 ODBC 数据源的数据库中。使用 DATABASE 变量的操作系统必须指定与 ODBC.INI 文件中的数据源相同的数据库。

```
%DEFINE{
    DATABASE="qesql"
    SHOWSQL="YES"
    table="int_null"
    LOGIN="netdata1"
    PASSWORD="ibmdb2"
}%

%function(dtw_odbc) sql1() {
create table int_null (int1 int, int2 int)
}%

%function(dtw_odbc) sql2() {
insert into $(table) (int1) values (111)
}%

%function(dtw_odbc) sql3() {
insert into $(table) (int2) values (222)
}%

%function(dtw_odbc) sql4() {
select * from $(table)
}%

%function(dtw_odbc) sql5() {
drop table $(table)
}%

%HTML(REPORT){
@sql1()
@sql2()
@sql3()
@sql4()
}%
```

Oracle 语言环境

Oracle 语言环境提供了对 Oracle 数据的本机访问。您可以从在 CGI、FastCGI、NSAPI、ISAPI 或 GWAPI 方式下运行的 Net.Data 中访问 Oracle 表格。本语言环境支持 Oracle 7.2、7.3 和 8.0。

限制:

- 在这个语言环境中不支持存储过程。
- DATABASE 变量不用于访问 Oracle 数据库。

- LOGIN 变量中必须包含 Oracle 数据库的实例名称。例如, *ora73* 是在以下 LOGIN 变量中定义的实例名称:

```
LOGON=admin@ora73
```

要从 *Net.Data* 访问 Oracle

1. 验证 *Net.Data* 初始化文件中的 ENVIRONMENT 语句对于 Oracle 语言环境是正确的。请参阅 *Net.Data* 管理与程序设计指南中的配置一章, 以获取有关步骤和例子的更多信息。
2. 确保已经安装了适当的 Oracle 组件, 并能如下工作:
 - a. 在安装 *Net.Data* 的机器上安装 SQL*Net (如果尚未安装的话)。有关的更多信息, 请参阅以下 URL:

```
http://www.oracle.com/products/networking/html/stdn_sqlnet.html
```

- b. 验证使用 Oracle *tnsping* 函数时可以采用与 Web 服务器所使用的相同的安全性权限。要验证这一点, 可用您的用户 ID 注册到 Web 服务器, 然后键入:

```
tnsping oracle-instance-name
```

其中 *oracle-instance-name* 是您的 *Net.Data* 宏所访问的 Oracle 系统的名称。

如果您的 Web 服务器在系统权限下运行, 那么您可能无法在 Windows NT 上验证 *tnsping* 函数。如果是这样的话, 则跳过这一步。

- c. 验证 Oracle 表格可以采用与 Web 服务器所使用的相同的安全性权限来访问。为了验证这一点, 请使用 SQL*Plus 行命令工具, 输入一个 SQL SELECT 语句, 从而通过 SQL SELECT 语句使用您的 Web 服务器的权限来访问 Oracle 表格。例如:

```
SELECT * FROM tablename
```

如果您的 Web 服务器在系统权限下运行, 那么您可能无法在 Windows NT 上进行验证。如果是这样的话, 则跳过这一步。

疑难问题解决: 如果上述步骤失败, 请不要继续。如果有一步失败, 请检查您的 Oracle 配置。

3. 确保在您的 Web 服务器处理中已经正确设置了 Oracle 环境变量。

- 对于 AIX, 请在 */etc/environment* 文件中加入以下几行:

```
ORACLE_SID=oracle-instance-name
ORACLE_HOME=oracle-runtime-library-directory
```

- 对于 Windows NT, 请使用“系统属性控制”屏面来添加以下环境变量:

```
ORACLE_SID=oracle-instance-name
ORACLE_HOME=oracle-runtime-library-directory
```

提示: 对于其它的 Oracle 环境变量, 您可能还需要添加一些行, 这取决于您计划使用的 Oracle 程序, 例如: 国家语言支持和两阶段确认。有关这些环境变量的更多信息, 请参考 Oracle 管理文档。

4. 从 *Net.Data* 测试与 Oracle 的连接。在您的 *Net.Data* 宏文件中, 请在 LOGIN 和 PASSWORD 变量中指定适当的值。访问 Oracle 数据库时, 不要定义 DATABASE 变量。下面是宏文件中连接语句的一个例子:

```
%DEFINE LOGIN=user_ID@remote-oracle-instance-name
%DEFINE PASSWORD=password
```

本地 Oracle 实例:

如果您只访问本地的 Oracle 实例，则不要指定远程 Oracle 实例的名称作为注册用户 ID 的一部分，如下面的例子所示：

```
%DEFINE LOGIN=user_ID
%DEFINE PASSWORD=password
```

现场连接:

如果您使用现场连接，那么您可以在现场连接配置文件中指定 LOGIN 和 PASSWORD，尽管出于安全性的目的，我们建议您不要这么做。例如：

```
LOGIN=user_ID
PASSWORD=password
```

提示： 不要对 Oracle 指定 DATABASE 变量。

5. 通过运行 CGI 外壳脚本来测试您的配置，从而确保可以从您的 Web 服务器访问 Oracle 实例，如下面的例子所示：

```
#!/bin/sh
echo "content-type; text/html"
echo
echo "< html>< pre>"
set
echo "</pre>< p>< pre>"
tnsping oracle-instance-name
echo
```

同样，您可以从 Net.Data 宏直接执行 *tnsping*，如下面的例子所示：

```
%DEFINE testora = %exec "tnsping oracle-instance-name"
%HTML (report){
< P>About to test Oracle access with tnsping.
< hr>
$(testora)
< hr>
< P>The Oracle test is complete.
%}
```

疑难问题解决:

如果验证步骤失败，则请检查前面所有的步骤都是成功的，这可以通过验证以下项来实现：

- 检查您的 Oracle 配置。
- 验证 Oracle 环境变量的语法正确，并且没有丢失变量。
- 检查 Oracle 连接，确保您输入的用户 ID 与口令是正确的。

如果验证步骤仍然失败，请与 IBM 的服务部门联系。

例子:

在完成了访问验证步骤之后，您可以使用宏文件中的函数来调用 Oracle 语言环境，如下面的例子所示：

```
%FUNCTION(DTW_ORA) STL1() {
insert into $(tablename) (int1,int2) values (111,NULL)
%}
```

Perl 语言环境

Perl 语言环境能够解释在 Net.Data 宏的一个 FUNCTION 块中指定的在线 Perl 脚本，或者它能处理存储在服务器上单独文件中的外部 Perl 脚本。对外部 Perl 脚本的调用在 FUNCTION 块中是由一个 EXEC 语句来标识的，例如：

```
%EXEC{ perl-script-name [optional parameters] %}
```

Perl 语言环境不能直接传递或检索 Net.Data 变量，因此 Perl 脚本以这种方式来使用它们：

- Net.Data 将输入参数作为环境变量传递给 Perl 脚本。Perl 脚本可以通过读数 Perl 相关数组来检索参数。
- Perl 脚本将输出参数传回该语言环境的方法是：将结果写入一个 Net.Data 在环境变量 DTWPIPE 中传递其名称的已命名管道中。使用 DEFINE 语句语法来将数据写入已命名管道：

```
name = value
```

对于复合数据项，可以用一个新行字符或空字符分开每个项。

如果一个变量名称与输出参数的名称相同且使用上述语法，则新的值将替换当前值。如果变量名不对应于输出参数，Net.Data 将把它忽略。

以下示例显示了 Net.Data 如何从一个宏文件传递变量。

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    $date = 'date';
    chop $date;
    open(DTW, "> $ENV{DTWPIPE}") || die "Could not open: $!";
    print DTW "result = \"\$date\"\n";
}%
%HTML(INPUT) {
    @today()
}%
```

如果 Perl 脚本位于一个名为 today.pr1 的外部文件中，那么这个函数可以按下一个例子来编写：

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    %EXEC { today.pr1 %}
}%
```

Perl 语言环境程序通过 Net.Data 名称来访问表格参数的值。表格 T 的列标题为 T_N_i，字段值为 T_V_i_j。表格 T 中的行数与列数分别是 T_ROWS 和 T_COLS。

REPORT 和 MESSAGE 块在所有 FUNCTION 部分都是允许的。它们由 Net.Data 来处理，而不是由语言环境来处理。当然，Perl 程序可以将文本写入标准输出流，并直接处理输出的 HTML 格式。

权限技巧：请确保执行 Net.Data 的用户 ID 对该语言环境所引用的任何外部的可执行文件都具有访问权限，包括 Perl 解释程序正确的版本号。请参阅 *Net.Data 管理和程序设计指南* 的配置一章中关于指定 Web 服务器对 Net.Data 文件的访问权限这一部分，以获取更多信息。

REXX 语言环境

REXX 语言环境可以解释 `Net.Data` 宏中 `FUNCTION` 块内指定的在线 REXX 程序，或可以执行存储在单独文件中的 REXX 程序。对外部 REXX 程序的调用在 `FUNCTION` 块中是由一条语句来标识的，例如：

```
%EXEC{ REXX-program-file-name [optional parameters] %}
```

REXX 语言环境使用 `RexxStart()` API 来让 REXX 解释程序执行指定的文件，然后在文件名之后将参数传递给程序，就如同它们是在命令行中输入的一样。对于 REXX 程序来说，所有的参数都是作为 `ARG[1]` 接收的。

权限技巧：请确保执行 `Net.Data` 的用户 ID 对语言环境所引用的任何外部的可执行文件都具有访问权限。请参阅 *Net.Data 管理和程序设计指南* 的配置一章中关于指定 Web 服务器对 `Net.Data` 文件的访问权限这一部分，以获取更多信息。

变量替换：

之后在 `FUNCTION` 块的可执行语句部分来执行变量替换。当然，无论 REXX 程序是在一个 `FUNCTION` 块中内部定义的，还是在一个单独文件中外部定义的，它都可以访问参数。REXX 语言环境使用 REXX 语言环境的 `RexxVariablePool()` 函数来与 REXX 程序共享 `Net.Data` 变量。这使得 REXX 程序能够直接处理参数列表中标识的 `Net.Data` 变量。

REXX 程序访问表格参数的值，把它作为 REXX 词干变量。对于 REXX 程序来说，表格 `T` 的列标题为 `T_N.i`，字段值为 `T_V.i.j`。表格 `T` 中的行数与列数分别为 `T_ROWS` 和 `T_COLS`。

为 AIX 操作系统提高性能：

如果在 AIX 系统中有许多个对 REXX 语言环境的调用，则可以考虑将 `RXQUEUE_OWNER_PID` 环境变量设置为 0。而调用此 REXX 语言环境的宏可以很方便地调用许多进程、调用系统资源。

您可以用以下三种方式来设置环境变量：

- 在宏文件中使用 `DTW_SETENV` 内部函数：

```
@DTW_rSETENV("RXQUEUE_OWNER_PID", "0")
```

- 在 AIX 系统环境文件中插入以下语句：

```
/etc/environment: RXQUEUE_OWNER_PID = 0
```

此方法将影响整个机器上 REXX 的功能。

- 在 HTTP Web 服务器环境文件中；例如，对于 Domino Go Webserver 插入以下语句：

```
InheritEnv RXQUEUE_OWNER_PID = 0
```

此方法将影响 Web 服务器上 REXX 的功能。

SQL 语言环境

SQL 语言环境通过 DB2 执行 SQL 语句。SQL 语句的结果可以 Net.Data 的缺省表格或您指定的表格中返回。

Net.Data 支持所有您授权的 SQL 语句。在调用 Net.Data (作为 CGI 应用程序)时, 您可以为每个 HTML 部分连接到一个数据库, 并且必须使用 DATABASE 变量(除了 OS/390) 来指定数据库的名称。如果您的 DB2 数据库和 Web 服务器在同一台机器上, 则不需要额外的设置。否则, 根据您所使用的操作系统, 您可以通过使用“客户应用程序允许程序”(CAE) 来访问远程数据库, 或者使用“数据库连接服务”(DDCS) 来获取所有支持 DB2 支持的事务。您也许还可以使用 DataJoiner 来访问其它数据库。使用 DataJoiner 可以让您对支持的数据库使用两阶段确认。

Sybase 语言环境

Sybase 语言环境提供了对 Sybase 数据的本机访问。您可以从在 CGI、FastCGI、NSAPI、ISAPI 或 GWAPI 方式下运行的 Net.Data 中访问 Sybase 表格。

限制:

- Sybase 语言环境不支持诸如图象和音频等大型对象。只有对于那些没有 SELECT 语句的过程才支持存储过程。
- Sybase 语言环境需要现场连接来使用 FastCGI。

要从 Net.Data 访问 Sybase

1. 验证 Net.Data 初始化文件中的 ENVIRONMENT 语句对于 Sybase 语言环境是正确的。请参阅 Net.Data 管理与程序设计指南中的配置一章, 以获取有关步骤和例子的更多信息。
2. 确保已经安装了适当的 Sybase 组件, 并能如下工作:
 - a. 在安装 Net.Data 的机器上安装 Sybase 的 Open Client (如果尚未安装的话)。有关的更多信息, 请参阅 Sybase Open Client 文档。
 - b. 验证使用 Sybase ping 函数时可以采用与 Web 服务器所使用的相同的安全性权限。要验证这一点, 可用您的用户 ID 注册到 Web 服务器, 然后键入:

```
ping sybase-instance-name
```

其中 *sybase-instance-name* 是您的 Net.Data 宏所访问的 Sybase 系统的名称。

如果您的 Web 服务器在系统权限下运行, 那么您可能无法在 Windows NT 上验证 ping 函数。如果是这样的话, 则跳过这一步。

- c. 验证 Sybase 表格可以采用与 Web 服务器所使用的相同的安全性权限来访问。为了验证这一点, 请使用 ISQL 行命令工具, 输入一个 SQL SELECT 语句, 从而使用您的 Web 服务器的权限来访问 Sybase 表格。例如:

```
SELECT * FROM tablename
```

如果您的 Web 服务器在系统权限下运行, 那么您可能无法在 Windows NT 上进行验证。如果是这样的话, 则跳过这一步。

疑难问题解决: 如果上述步骤失败, 请不要继续。如果有一步失败, 请检查您的 Sybase 配置。

3. 确保在您的 Web 服务器处理中已经正确设置了 Sybase 环境变量。

- 对于 AIX, 请在 `/etc/environment` 文件中加入以下几行:

```
DSQUERY=sybase-instance-name
SYBASE=sybase-runtime-library-directory
```

- 对于 Windows NT, 请使用“系统属性控制”屏面来添加以下环境变量:

```
DSQUERY=sybase-instance-name
SYBASE=sybase-runtime-library-directory
```

提示: 对于其它的 Sybase 环境变量, 您可能还需要添加一些行, 这取决于您计划使用的 Sybase 程序, 例如: 国家语言支持和两阶段确认。有关这些环境变量的更多信息, 请参考 Sybase 管理文档。

4. 从 Net.Data 测试与 Sybase 的连接。在您的 Net.Data 宏文件中, 请在 LOGIN、PASSWORD 和 DATABASE 变量中指定适当的值。下面是宏文件中连接语句的一个例子:

```
%DEFINE DATABASE=database-name
%DEFINE LOGIN=user_ID@remote-sybase-instance-name
%DEFINE PASSWORD=password
```

现场连接: 如果您使用现场连接, 那么您可以在现场连接配置文件中指定 LOGIN 和 PASSWORD, 尽管出于安全性的目的, 我们建议您不要这么做。例如:

```
DATABASE=database-name
LOGIN=user_ID
PASSWORD=password
```

5. 通过运行 CGI 外壳脚本来测试您的配置, 从而确保可以从您的 Web 服务器访问 Sybase 实例, 如下面的例子所示:

```
#!/bin/sh
echo "content-type; text/html"
echo
echo "< html>< pre>"
set
echo "</pre>< p>< pre>"
isql -u user_ID -p password << EOFF
SELECT * FROM tablename
EOFF
echo
```

疑难问题解决:

如果验证步骤失败, 则请检查前面所有的步骤都是成功的, 这可以通过验证以下项来实现:

- 检查您的 Sybase 配置。
- 验证 Sybase 环境变量的语法正确, 并且没有丢失变量。
- 检查 Sybase 连接, 确保您输入的用户 ID 与口令是正确的。

如果验证步骤仍然失败, 请与 IBM 的服务部门联系。

例子:

一旦完成了访问验证步骤, 您可以使用宏文件中的函数来调用 Sybase 语言环境, 如下面的例子所示:


```
%function(DTW_SYB) STL1() {
insert into $(tablename) (int1,int2) values (111,NULL)
%}
```

System 语言环境

System 语言环境是 Net.Data 所定义的环境变量，它支持对某些外部程序的调用，这些外部程序在 FUNCTION 块内的一条 EXEC 语句中标识。

System 语言环境通过将程序名和参数传递到操作系统（由 C 语言的 system() 函数调用运行），来处理 EXEC 程序。这个方法不允许外部的程序直接与 Net.Data 传递变量，就象 REXX 语言环境那样，因此 Net.Data 采用以下方式来处理这些变量：

- Net.Data 将输入参数作为环境变量传递给外部程序，外部程序则对它们进行检索：
 - UNIX CSHELL 脚本通过在环境变量名前添加一个美元符号 (\$) 来引用此环境变量，例如 \$x。
 - Perl 语言脚本则通过引用关联数组 ENV 来引用此环境变量，例如 %ENV{'x'}。
 - DOS 批处理文件引用括在百分号 (%) 中的变量，例如 %x%。
- 大部分操作系统将输出参数传回该语言环境的方法是：将结果写入一个 Net.Data 在环境变量 DTWPIPE 中传递其名称的已命名管道中。(Net.Data for OS/400 使用环境变量将参数传回语言环境。)使用 DEFINE 语句语法来将数据写入已命名管道：

```
name = value
```

对于复合数据项，可以用一个新行字符或空字符分开每个项。

如果变量名与输出参数匹配，则新的值将替换当前值。Net.Data 将忽略那些与任何输出参数都不匹配的变量名。

system 语言环境程序通过 Net.Data 名称来访问表格参数的值。表格 T 的列标题为 T_N_i，字段值为 T_V_i_j。表格 T 中的行数与列数分别是 T_ROWS 和 T_COLS。

权限技巧：请确保执行 Net.Data 的用户 ID 对从 System 语言环境调用的任何外部的可执行文件都具有访问权限。请参阅 *Net.Data 管理和程序设计指南* 的配置一章中关于指定 Web 服务器对 Net.Data 文件的访问权限这一部分，以获取更多信息。

Web 注册表语言环境

Net.Data Web 注册表为与应用程序相关的数据提供了持久性的存储器。Web 注册表可用于存储配置信息和其它能够被基于 Web 的应用程序在运行时动态访问的数据。您只能这样来访问 Web 注册表：从为此目的而编写的 CGI 程序出发，通过 Net.Data 宏并使用 Net.Data 和 Web 注册表内部支持。Web 注册表在操作系统的子集上也是可用的。请参阅 *Net.Data 参考* 中的 Net.Data 操作系统参考附录

标准 Web 页面的开发需要直接将 URL 放在该页面的 HTML 源代码中。这就使更改链接变得困难。静态的特性还限制了那些可以方便地放入 Web 页面的链接的类型。使用一个 Web 注册表来存储与应用程序相关的数据，例如：URL 可以帮助您创建具有动态设置链的 HTML 页面。

应用程序开发者和对注册表具有写入权限的 Web 管理员可以将信息存储在注册表中并对其进行维护。应用程序在运行时从与它们关联的注册表中检索信息。这就方便了具有灵活性的应用程序的设计，并且允许应用程序和服务器的移植。您可以使用 Net.Data 宏来创建使用动态设置链的 HTML 页面。

信息以注册表条目的形式存储在 Web 注册表中。每个注册表条目都由一对字符串组成：一个 RegistryVariable 字符串和一个相应的 RegistryData 字符串。任何可以由一对字符串来表示的信息都可以作为注册表的条目存储。 Net.Data 将变量字符串作为搜索关键字，在注册表中定位和检索特定的条目。

您可以在表3 中看到 Web 注册表的示例内容。

表 3. 示例 Web 注册表

CompanyName	WorldConnect
Server	ftp.einet.net
JohnDoe/foreground	Green
CompanyURL/IBM Corp.	http://www.ibm.com
CompanyURL/Sun Microsystems Corp.	http://www.sun.com
CompanyURL/Digital Equipment Corp.	http://www.dec.com
JaneDoe/Home_page	http://jane.info.net

这里是考虑使用 Web 注册表的一些原因：

- 您可以使用一个 Web 注册表来存储服务器和 URL 的别名，从而简化了应用程序和服务器的重定位。
- 应用程序开发者可以将他们基于 Web 的应用程序和数据一起发行，例如注册表中预定义的 URL。最终用户可以修改注册表数据，从而更改应用程序的功能。
- Web 注册表可以根据产品名、国家语言、制造商等来执行 URL 搜索。

Web 注册表中的索引项的 RegistryVariable 字符串都有一个附加的索引字符串，使用以下语法：

RegistryVariable/Index

用户在一个内部函数单独的参数中提供了索引字符串的值，这个内部函数被设计为与索引项一起作用。多个索引的注册表条目可以具有相同的 RegistryVariable 字符串值，但它们可以通过具有不同的索引字符串值来维持它们的唯一性。

表 4. 示例索引 Web 注册表

Smith/Company_URL	http://www.ibm.link.ibm.com
Smith/Home_page	http://www.advantis.com

甚至上述两个索引项具有相同的 RegistryVariable 字符串值 Smith，在各种情况下索引字符串都是不同的。 Web 注册表函数将它们作为两个不同的条目来对待。

配置 Net.Data 语言环境

在使用语言环境和 Net.Data 之前，您必须先配置 Net.Data 初始化文件，如果您使用现场连接，则还需要配置现场连接配置文件。

以下是这些任务的概述。请参阅 *Net.Data 管理与程序设计指南* 中的配置一章，以获取有关如何配置 Net.Data 语言环境的详细信息。另外，请检查先前的语言环境部分，以获取特殊的配置指导。

- 验证或更新 Net.Data 初始化文件 `db2www.ini` 中的 ENVIRONMENT 语句。这些语句在 *Net.Data 管理与程序设计指南* 中的『配置 Net.Data』中有所描述。
- 在现场连接文件 `dtwcm.cnf` 中为数据库或 Java 应用程序语言环境定义 cliette。定义 cliette 在 *Net.Data 管理与程序设计指南* 的配置一章中有所描述。

第2部分 非 IBM 的语言环境

除了提供语言环境之外，Net.Data 还允许您添加自己的语言环境和数据库环境。Net.Data 访问您的语言环境时将它们作为动态链接库或共享程序库，这与 Net.Data 可执行文件的访问是分开的。每种语言环境都必须支持由 Net.Data 定义的一套标准接口。以下章节将讨论如何创建一种新的语言环境，并描述语言环境的程序设计接口和环境。

第27页的『第3章 创建一个新的语言环境』

第34页的『设计语言环境语句』

第37页的『第4章 语言环境程序设计接口应用函数』

第3章 创建一个新的语言环境

Net.Data 将语言环境作为可插入的编程语言和数据库接口来使用，作为 DLL 文件或共享程序库来访问。Net.Data 提供了一系列语言环境，但是在当它们不能满足您的应用需求时，您还可以创建新的语言环境。在您决定创建一个新的语言环境之前，请先确定 IBM 在提供 Net.Data 时一起提供的语言环境是否满足您的需要。

当您决定要创建一个新的语言环境时，请完成以下步骤：

- 确定您必须为该语言环境提供哪些接口与函数。dtw_execute() 接口必须提供，并且提供的所有接口都必须与 dtwle.h C 语言首部中所定义的原型完全匹配。
- 创建一个 makefile 或 JCL 来构建 DLL 或共享程序库。
- 构建一个能够实现您希望提供的语言环境接口例程集的 DLL 或共享程序库。请参阅适用于您的操作系统的 C 或 C++ 文档，以便理解如何创建 makefile，如何构建 DLL 或共享程序库。
- 使得所有的接口都可以从 DLL 或共享程序库的外部获得，这样 Net.Data 就调用它们了。
- 确定您的 ENVIRONMENT 配置语句，然后将它添加到 Net.Data 初始化文件中。请参阅第34页的『设计语言环境语句』，以获取细节。
- 将函数添加到使用新的语言环境的 Net.Data 宏文件中。

本章将描述如何设计语言环境。

- 『设计一个 DLL 或共享程序库』
- 第29页的『语言环境通信结构』
- 第32页的『语言环境接口函数』
- 第34页的『设计语言环境语句』

设计一个 DLL 或共享程序库

构建一个语言环境时，您将更新第63页的『附录A. 语言环境模板』中提供的模板来包括环境接口函数和通信结构，Net.Data 使用这些通信结构来与您的语言环境通信并传递参数。

以下章节将描述函数与结构的概念与设计问题。第37页的『第4章 语言环境程序设计接口应用函数』中描述了语言环境接口中所提供的实用程序。

- 第28页的『我应该提供哪些语言环境?』
- 第28页的『处理输入参数』
- 第28页的『处理用户请求』
- 第29页的『处理输出参数』
- 第29页的『通信错误条件』

我应该提供哪些语言环境？

编写语言环境时，必须确定提供哪些接口。您的选项取决于您希望语言环境做什么。例如，如果您希望语言环境访问数据库，那么您所作出的选项将和把它用于脚本语言时有很大的不同。下面这一节将描述 `Net.Data` 语言环境接口。

`dtw_execute()`

您必须提供 `dtw_execute()` 接口来从宏文件传递输入参数；它是每个语言环境中唯一必需的接口。`Net.Data` 通过语言环境通信结构 `dtw_lei` 将所有的输入参数传递给 `dtw_execute()`。

`dtw_initialize()`

提供 `dtw_initialize()` 接口是为了分配或初始化数据。在对您的语言环境进行首次函数调用之前，`Net.Data` 仅为每个宏调用调用一次此接口。如果没有对您的语言环境的函数调用，则 `Net.Data` 将不调用 `dtw_initialize()` 接口。

`dtw_cleanup()`

在您提供 `dtw_initialize()` 接口时以及您希望在宏异常终止时允许错误处理的情况下需要提供 `dtw_cleanup()` 接口。对于每个宏调用，`Net.Data` 仅调用此接口一次。

`dtw_getNextRow()`

`dtw_getNextRow()` 接口是作为数据库语言环境或能以“每次一行”方式处理数据的语言环境的一部分提供的。只有当 `Net.Data` 在 OS/400 或 OS/390 操作系统上运行时才调用此接口。

处理输入参数

`Net.Data` 语言环境使用 `dtw_execute()` 接口来接收和处理参数。`dtw_execute()` 接口与 `dtw_lei` 结构一起作用，`Net.Data` 使用该结构来与语言环境通信。在编写您的语言环境时，请对输入参数处理使用以下建议。

- 在初始化文件中指定任何隐式的参数。当 `Net.Data` 传递了宏编写者在要执行的 `FUNCTION` 块中指定的参数之后，它将在所有对语言环境的函数调用中传递这里所指定的参数。
- 接收那些给 `dtw_execute()` 接口的输入参数，把它们作为 `dtw_lei` 结构的一部分。宏编写者在 `Net.Data` 宏定义的 `FUNCTION` 块中指定参数时便确定了 `Net.Data` 传递这些参数的顺序。

程序模板中的 `processInputParms()` 例程(第63页的『附录A. 语言环境模板』中)，显示了一个处理输入参数的方法。

处理用户请求

语言环境如何处理用户请求取决于该语言环境如何接收请求。`Net.Data` 提供了多种能够使您与您的语言环境就某个请求进行通信的不同方法。

- 通过 `FUNCTION` 块上指定的函数名。在每次函数调用时，`Net.Data` 在 `dtw_lei` 结构的 `function_name` 字段中将函数名传递给语言环境。
- 通过 `FUNCTION` 块参数表。您可以指定参数表中的某个参数能够表示一个用户请求。在每次函数调用时，`Net.Data` 在 `dtw_lei` 结构的 `parm_data_array` 字段中将参数传递给语言环境。

- 通过 FUNCTION 块的可执行语句部分。在每次函数调用时，Net.Data 在 dtw_lei 结构的 exec_statement 字段中将 FUNCTION 块中指定的所有可执行语句传递给语言环境。

处理输出参数

您用于处理输出参数的方法完全取决于您的语言环境以及它如何处理用户请求。但是，一旦语言环境具有了它需要返回给 Net.Data 宏的数据，您就可以设置语言环境来修改参数的值(这些参数在 dtw_lei 结构的 parm_data_array 字段中传递)。程序模板中的 processOutputParms() 例程(第63页的『附录A. 语言环境模板』中)，显示了一个处理输出参数的可能的方法，同时还提供了如何设置字符串与表格参数值的例子。

通信错误条件

函数调用的成功或失败可以过隐式的 Net.Data 宏变量 RETURN_CODE 来传递。此变量是在从 dtw_execute() 接口的调用返回之后由 Net.Data 设置的。它的值被设置为 dtw_execute() 调用本身的返回值。然后，Net.Data 使用这个值来处理 Net.Data 宏中的 MESSAGE 块(如果在此函数调用中指定了的话)。

如果您不指定 MESSAGE 块，或者在指定的 MESSAGE 块中没有某个条目可以处理来自 dtw_execute() 的返回吗，那么 Net.Data 将显示 dtw_lei 结构中 default_error_message 字段的内容。任何时刻，语言环境都可以在 dtw_execute() 例程中设置这个字段。程序模板中的 setErrorMessage() 例程(第63页的『附录A. 语言环境模板』中)，显示了如何设置 default_error_message 字段的例子。

语言环境通信结构

Net.Data 使用两个结构来与您的语言环境通信。您的语言环境必须与这些结构一起工作，并在结构内部设置与传递信息。

- dtw_lei
- dtw_parm_data

Net.Data 将一个语言环境接口结构(例如，dtw_lei) 传递到它调用的语言环境函数中。这个结构中除了其它内容外，还包含了一个参数数据数组，该数组中包含要传递给语言环境函数的参数列表。Net.Data 调用的语言环境函数处理请求、更新参数数据数组中的参数(如果适用的话)并返回给 Net.Data。

然后，Net.Data 转向参数数据数组，更新其中的参数副本以便反映语言环境函数设置的新值，然后继续处理 Net.Data 宏。

dtw_lei 结构

每个语言环境的接口函数都接收一个指向 dtw_lei 结构的指针。dtw_lei 结构具有以下格式：

```
typedef struct dtw_lei {
    char *function_name; /* 语言环境接口          */
    int  flags;          /* 函数块名称          */
                                /* 语言环境接口标志    */

    char *exec_statement; /* 语言环境语句        */
}
```

```

    dtw_parm_data_t *parm_data_array; /* 参数数组 */
    char *default_error_message; /* 缺省消息 */
    void *le_opaque_data; /* 语言环境特定的数据 */

    void *row; /* 用于“每次一行”处理 */

    char reserved[64]; /* 保留 */
} dtw_lei_t;

```

dtw_lei 结构中的字段:

function_name

function_name 字段中包含一个指向字符串的指针，该字符串中包含函数块的名称。这在语言环境所显示的错误消息中指定 FUNCTION 块的名称时有用。

标志 Net.Data 使用标志字段来与语言环境通信。它通过执行一个使用以下常量的“或”操作来设置该标志字段指针:

- Net.Data 设置 DTW_STMT_EXEC 是为了告诉 dtw_execute() 接口函数: exec_statement 字段中包含来自 EXEC 语句的文件名和参数。
- DTW_END_ABNORMAL 是由 Net.Data 设置的，用于告诉 dtw_cleanup() 接口函数出现了不正常或非期望的情况，并且语言环境需要在 Net.Data 终止之前执行必需的清理(即，释放占有的资源)。
- DTW_LE_FATAL_ERROR 是由一个语言环境接口函数设置的，用于告诉 Net.Data 语言环境中发生了致命错误。如果设置了此标志，Net.Data 将停止对 Net.Data 宏的处理，调用所有活动的语言环境中标志设置为 DTW_END_ABNORMAL 的 dtw_cleanup() 接口函数，打印出缺省的消息，然后退出。只有对语言环境调用返回非零值时才检查此标志。
- DTW_LE_MSG_KEEP 是由一个语言环境接口函数设置的，用于告诉 Net.Data 不应释放 default_error_message 所指向的存储器。如果没有设置这个常量，Net.Data 将试图释放该存储器。
- DTW_LE_CONTINUE 是由 dtw_execute() 接口函数设置的，用于告诉 Net.Data 去调用 dtw_getNextRow() 接口函数。只有在设置了此标志并且 dtw_execute() 接口函数的返回值为零时，Net.Data 才调用 dtw_getNextRow()。

exec_statement

exec_statement 字段中包含下面的一个指针:

- 指向一个字符串的指针，该字符串中包含来自 FUNCTION 块的可执行语句(在变量替代之后)
- 指向文件名与参数的指针，它们来自 EXEC 语句

parm_data_array

parm_data_array 字段包含一个指向 dtw_parm_data 结构的数组的指针。该数组以一个包含零的 parm_data 结构结束。Net.Data 使用 dtw_parm_data 结构来将变量和相关的值传递给一个语言环境，并检索语言环境可能对变量值作的任何更改。请参阅第31页的『dtw_parm_data 结构』，以获取对该结构的描述。

default_error_message

default_error_message 字段是由语言环境设置的，它被设置为一个描述错误条件的字符串。如果对语言环境接口函数调用的返回值非零，且返回值与 MESSAGE 块中的消息值不匹配，则将显示缺省的消息。否则，Net.Data 将显示从 MESSAGE 块中选择的消息。

le_opaque_data

le_opaque_data 字段是由语言环境中的任一接口函数设置的，用于将参数从一个接口函数传递到另一个接口函数。Net.Data 保存指针并将它传递到 Net.Data 调用的另一个接口函数中。在处理完 Net.Data 宏之后、返回 Net.Data 之前，Net.Data 将定义指向 NULL 的指针。因为这个字段是特定于线程的，所有语言环境可以存储特定于线程的数据。只有当您具有一个 dtw_cleanup() 接口函数时才使用此字段，这样函数就可以释放与 le_opaque_data 字段关联的存储器。

row row 字段是由 Net.Data 设置在调用一个语言环境的 dtw_getNextRow() 接口函数时为行对象设置的。dtw_getNextRow() 函数使用 Net.Data 行实用程序接口函数在对象中插入一行表格数据。然后，Net.Data 处理该行并调用 dtw_getNextRow()，直至没有需要处理的行为止。

保留字段是为 IBM 今后的使用而保留的。

dtw_parm_data 结构

Net.Data 使用 dtw_parm_data 结构来向语言环境传递参数。这些参数有三个来源：

- 在 FUNCTION 块定义中明确指定的参数
- 在 Net.Data 初始化文件的 ENVIRONMENT 配置语句中指定的参数
- 在 FUNCTION 块定义的 RETURNS 关键字中指定的返回变量

Net.Data 首先传递明确的参数，然后传递 ENVIRONMENT 语句中指定的参数，接着再是返回变量。

dtw_parm_data 结构具有以下格式：

```
typedef struct dtw_parm_data {          /* 参数数据          */
    int  parm_descriptor;               /* 参数描述符          */
    char *parm_name;                   /* 参数名              */
    char *parm_value;                  /* 参数值              */
    void *res1;                        /* 保留                */
    void *res2;                        /* 保留                */
} dtw_parm_data_t;
```

dtw_parm_data 结构中的字段：

parm_descriptor

parm_descriptor 字段描述要传递给语言环境的参数的类型和用法。Net.Data 通过执行一个使用以下常量的“或”操作来设置该字段：

- DTW_IN 表示某个参数是一个仅为输入的参数。
- DTW_OUT 表示某个参数是一个仅为输出的参数。
- DTW_INOUT 表示某个参数是一个输入输出参数。
- DTW_STRING 表示参数值是一个指向字符串的指针。
- DTW_TABLE 表示参数值是一个指向表格的指针。

Net.Data 总是将 parm_descriptor 字段设置为 DTW_IN、DTW_OUT 或 DTW_INOUT，并在它们与 DTW_STRING 和 DTW_TABLE 之间使用一个逻辑“或”。

parm_name

parm_name 字段是一个指向字符串的指针，该字符串中包含参数的名称。如果参数是文字串，则 Net.Data 将此指针设置为 NULL。

parm_value

parm_value 字段是一个指向对象的指针，该对象中包含参数的值。如果参数是一个尚未定义的变量，则 Net.Data 将此指针设置为 NULL。

res1 和 res2 字段是保留字段。

parm_name 和 parm_value 都指向一个从 Net.Data 运行时间堆阵分配的对象(所谓堆阵，就是 Net.Data 用于动态存储器分配的存储器区域)。如果用其它字符串来替换 parm_name 或 parm_value，那么原始的字符串必须被释放并用一个指向从 Net.Data 堆阵分配的字符串的指针来代替。使用 dtw_malloc() 和 dtw_free() 应用函数来释放原始的字符串。

语言环境接口函数

Net.Data 在使用语言环境时还使用了四个接口函数：您提供了这些函数中的一个或多个。其中三个函数是可选的，但是每个语言环境都必须有一个 dtw_execute() 接口函数。如果 Net.Data 宏引用一个没有 dtw_execute() 接口函数的语言环境，那么 Net.Data 将返回一个错误消息并停止对 Net.Data 宏的处理。

要调用一个语言环境，需要在 Net.Data 宏的 FUNCTION 块中引用它。语言环境接口函数必须按以下顺序调用：

1. dtw_initialize()
2. dtw_execute()
3. dtw_getNextRow()
4. dtw_cleanup()

dtw_execute() 函数是您在语言环境中必须提供的唯一的接口函数。

当 Net.Data 遇到对使用语言环境的函数的调用时，它将使用以下步骤来调用语言环境：

1. 如果已经为语言环境定义了 dtw_initialize()，Net.Data 将调用该函数。这个函数执行语言环境所需的任何初始化任务，例如连接至数据库或分配变量。
2. Net.Data 调用 dtw_execute() 来处理包含语言环境必须处理的语句的宏文件的 FUNCTION 块。
3. 如果 Net.Data 对 dtw_getNextRow() 的调用成功返回，则 dtw_execute() 表示应当调用 dtw_getNextRow()。
4. Net.Data 宏处理完成之后，如果已经为该语言环境定义了此函数，则 Net.Data 将调用 dtw_cleanup() 来清理环境(例如，断开与数据库的连接或释放变量)，然后返回 Web 服务器。

以下章节将描述接口函数：

- 第33页的『 dtw_initialize() 』
- 第33页的『 dtw_execute() 』
- 第33页的『 dtw_getNextRow() 』
- 第34页的『 dtw_cleanup() 』

dtw_initialize()

dtw_initialize() 接口函数执行语言环境所需的任何特殊的初始准备，例如连接至数据库或分配变量。此接口函数只调用一次，并且是可选的。

Net.Data 只调用语言环境的 dtw_initialize() 接口函数一次，也就是在 Net.Data 首次调用引用该语言环境的 FUNCTION 块时。对于该语言环境的后继引用将会绕过对 dtw_initialize() 接口函数的调用。

此接口函数不影响对 message 块的处理。返回码为正数或为零表示处理正在继续；而返回码为负则表示处理不会继续。如果返回码非零，并且在 default_error_message 字段中定义了一个缺省消息，则发出该缺省的消息；如果没有缺省消息，Net.Data 将发出错误消息。

dtw_execute()

dtw_execute() 接口函数用于处理宏文件 FUNCTION 块，其中包含必须由语言环境处理的语句。例如，一个引用数据库语言环境的 FUNCTION 块中包含了该语言环境用于查询数据库的 SQL 语句。

每当 Net.Data 宏处理一个引用语言环境的 FUNCTION 块时，都将调用 dtw_execute() 接口函数。dtw_execute() 接口函数完成之后所出现的情况取决于该语言环境是否在以“每次一行”的方式处理表格数据。如果是，则接口函数将在 dtw_lei 结构中设置 DTW_LE_CONTINUE 标志，从而告诉 Net.Data 需要调用 dtw_getNextRow() 接口函数。请参阅『dtw_getNextRow()』，以获取有关 dtw_getNextRow() 接口函数及其处理步骤的更多信息。

您可以通过让 dtw_execute() 接口函数执行所有产生 report 块处理的输入所必需的处理来优化性能。例如，SQL 语言环境的 dtw_execute 接口函数将在 report 块阶段生成整个有待处理的表格。

dtw_getNextRow()

dtw_getNextRow() 接口函数用于检索 Net.Data 表格“每次一行”处理的输入。每当设置了 DTW_LE_CONTINUE 标志时都将调用这个函数，表示表格中有另一行需要处理的数据。对于数据库语言环境使用 dtw_getNextRow()。

限制：只有当 Net.Data 在 OS/400 或 OS/390 操作系统上运行时才可调用此接口函数。

Net.Data 在满足以下情况时将调用 dtw_getNextRow()：

- 对语言环境的 dtw_execute() 调用已经成功完成(返回值为零)时
- dtw_execute() 接口函数已经设置了 dtw_lei 结构中的 DTW_LE_CONTINUE 标志时。

当 dtw_execute() 函数将 DTW_LE_CONTINUE 标志设置为“on”时，Net.Data 将执行以下步骤：

1. 为 dtw_execute() 接口函数的返回值处理 message 块。
2. 调用语言环境的接口函数 dtw_getNextRow()，然后开始“每次一行”处理。
3. 处理 report 块。
4. 为 dtw_getNextRow() 接口函数的返回值处理 message 块。

5. 确定 `dtw_getNextRow()` 是否将 `DTW_LE_CONTINUE` 标志设置为“on”：

- 如果是，继续第 2 步中对 `dtw_getNextRow()` 接口函数的处理。
- 如果否，则“每次一行”处理终止，`Net.Data` 继续处理 `Net.Data` 宏。

在调用 `dtw_getNextRow()` 时，`dtw_lei` 结构中的 `row` 字段将被设置为指向行对象。要处理行对象，可以使用 `Net.Data` 应用函数 `dtw_row_SetCols()` 和 `dtw_row_SetV()`。`Net.Data` 假定在首次调用 `dtw_getNextRow()` 接口函数之后，行对象中将包含表格的列标题。后继的调用中包含实际的表数据。

只要设置了 `DTW_LE_CONTINUE` 标志，`dtw_getNextRow()` 函数就继续被调用(除非 `message` 块的处理中指出其它的情况)。

dtw_cleanup()

如果您使用 `dtw_initialize()` 来初始化语言环境，则使用 `dtw_cleanup()` 接口函数来清理语言环境。例如，断开与数据库的连接或释放变量。这个接口函数是可选的。

在处理 `Net.Data` 请求时，`Net.Data` 将在处理结束或某个错误停止宏文件处理时调用一次语言环境的 `dtw_cleanup()` 接口函数。

如果清理过程异常，`Net.Data` 将把 `dtw_lei` 结构中的标志字段设置为 `DTW_END_ABNORMAL`。以下的异常条件提供了一些何时使用 `dtw_cleanup()` 的例子：

- 通过在 `dtw_lei` 结构的标志字段中设置 `DTW_LE_FATAL_ERROR` 位，语言环境接口函数可以以此来指出发生了致命错误。
- `Net.Data` 遇到了一个不可恢复的错误。
- `Net.Data` 宏 `message` 块处理出口中的结果。

如果语言环境的接口函数用一个要在接口函数之间传递的参数设置了 `le_opaque_data` 字段，则可以在处理结束时使用 `dtw_cleanup()` 来释放该字段。

此接口函数不影响对 `message` 块的处理。如果返回值非零，则发出缺省消息；如果不存在缺省消息，则宏处理器将发出警告消息。

设计语言环境语句

每个语言环境在初始化文件 `DB2WWW.INI` 中都有一个 `ENVIRONMENT` 语句，其中包含特定于该语言环境的信息。创建一个新的语言环境时，您需要为初始化文件设计一个环境语句，并用一个文档来说明用户应如何将它添加到初始化文件中。

`ENVIRONMENT` 语句指定有关 `Net.Data` 需要调用的语言环境的信息，并装入该语言环境 `DLL` 或共享程序库，例如语言环境名、`DLL` 或共享程序库名、以及在每个函数调用中要传递给语言环境的参数列表。

在调用 `Net.Data` 时，它将读取配置信息，但直到宏文件中调用指定该语言环境的 `FUNCTION` 块时，`Net.Data` 才装入语言环境 `DLL` 或共享程序库。`DLL` 将一直装在内存中，直到 `Net.Data` 结束。

以下章节将提供有关语法、参数描述的信息以及您可以在您的文档中使用的例子。

ENVIRONMENT 语句语法

ENVIRONMENT 语句具有以下格式:

```
ENVIRONMENT(type) library-name ([usage parameter, ...])
```

每个 ENVIRONMENT 语句必须在单独一行上。

下面是您必须为每个语言环境指定的参数:

- *type*

与此语言环境相关联的名称, 在 Net.Data 宏中具有 FUNCTION 块。您还必须在 FUNCTION 块定义中指定语言环境的类型, 以便告诉 Net.Data 哪些语言环境处理函数调用。名称不能以前缀 DTW_ 开头。这个前缀是为那些与 Net.Data 一起发行的语言环境保留的。请参阅 *Net.Data* 参考中的“Function 块”一章, 以获取有关 FUNCTION 块的更多信息。

- *library_name*

对象的名称, 包含 Net.Data 调用的语言环境接口。在 Windows NT 和 OS/2 中, 指定 DLL 的名称时是没有扩展名 *.dll* 的。在 AIX 中, 共享对象的名称是使用扩展名 *.o* 来指定的, 而在 OS/400 中, 服务程序的名称是用扩展名 *.SRVPGM* 指定的。OS/390 中, DLL 文件没有扩展名。您可以查看适用于您的操作系统的 Net.Data 所附带的初始化文件, 从而了解如何指定这个名称。可以考虑使用一个全限定路径名称, 这样可以确保 Net.Data 能够找到 DLL 或共享程序库。

- *parameter_list*

在每个函数调用中传递给语言环境的参数列表(除了那些在 FUNCTION 块定义中指定的参数)。在传递了 FUNCTION 块定义中指定的参数之后, 它们也将在 *dtw_lei* 结构的 *parm_data_array* 字段中传递。在函数调用之前, 您必须先在自己的 Net.Data 宏中将参数定义为变量。如果一个函数修改了这些参数的值, 那么在该函数结束处理时, 这些参数仍将保留被修改了的值。

ENVIRONMENT 语句例子

下面的例子显示了 Net.Data 所提供的用于语言环境的 ENVIRONMENT 语句。这些例子说明了如何指定参数。您在 ENVIRONMENT 语句中包含的变量是您允许 Net.Data 宏编写者在他们的宏中定义或覆盖的变量。请在 *Net.Data* 参考的附录或您的 Net.Data 自述文件中参阅特定于操作系统的信息, 在程序目录中参阅附加的例子。

以下例子显示了 OS/2、AIX 和 Windows NT 中的语法。

```
ENVIRONMENT (DTW_SQL)      DTWSQL      ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_SYB)      DTWSYB      ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_ORA)      DTWORA      ( IN LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_ODBC)     DTWODBC     ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, SHOWSQL, ALIGN, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_APPLET)   DTWJAVA     (  
ENVIRONMENT (DTW_JAVAPPS)  ( OUT RETURN_CODE ) CLLETTE "DTW_JAVAPPS"  
ENVIRONMENT (DTW_PERL)     DTWPERL     ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_REXX)     DTWREXX     ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_SYSTEM)   DTWSYS      ( OUT RETURN_CODE )  
ENVIRONMENT (HWS_LE)       DTWHWS      ( OUT RETURN_CODE )
```

ENVIRONMENT 语句在各种操作系统中可能不同；例如 OS/390 中对于 SQL 和 ODBC 访问就略有不同：

```
ENVIRONMENT (DTW_SQL)      DTWSQL      ( IN LOCATION, DB2SSID, DB2PLAN,  
TRANSACTION_SCOPE)
```

```
ENVIRONMENT (DTW_ODBC)     DTWODBC     ( IN LOCATION, TRANSACTION_SCOPE)
```

第4章 语言环境程序设计接口应用函数

Net.Data 为您提供了一个程序设计接口，在设计新的语言环境时使用。语言环境接口具有访问 Net.Data 服务(管理内存、配置变量)的应用函数，并提供了表格和行操作功能。第63页的『附录A. 语言环境模板』提供了一个模板，您可以在设计自己的语言环境时用作模型。

下面这一节将对 Net.Data 语言环境接口应用函数进行说明。

语言环境应用函数

语言环境使用应用函数来访问 Net.Data 服务。这些函数分成四类：

- 『用于管理内存的应用函数』
- 『用于管理配置变量的应用函数』
- 第38页的『用于表格操作的应用函数』
- 第38页的『用于行操作的应用函数』

用于管理内存的应用函数

语言环境使用内存管理应用函数来分配 Net.Data 所拥有的存储器，并释放使用 Net.Data 运行时程序库分配的存储器。

以下例子说明了为何需要这些应用函数。假定 Net.Data 是使用编译程序 A 以及相应的运行时程序库编写的。程序员编写一种新的语言环境，但却使用具有不同运行时程序库的编译程序 B。由于两个运行时程序库之间潜在的不兼容性，这个语言环境将无法释放 Net.Data 分配的存储区，而 Net.Data 也无法释放该语言环境所分配的存储区。

表 5. 内存管理应用函数

应用函数	说明
第42页的『dtw_malloc()』	使用 dtw_malloc() 从 Net.Data 的运行时间堆阵分配存储器。
第40页的『dtw_free()』	使用 dtw_malloc() 从 Net.Data 的运行时间堆阵释放已经分配的存储器。
第45页的『dtw_strdup()』	使用 dtw_malloc() 从 Net.Data 的运行时间堆阵分配存储器，并将指定的字符串复制到分配的存储器中。

用于管理配置变量的应用函数

用于配置变量的管理应用函数让语言环境访问存储在 Net.Data 初始化文件中的配置信息。通过使用这些函数，所有的语言环境就可以共享 Net.Data 初始化文件并使用其中的信息来配置语言环境。

表 6. 配置应用函数

应用函数	说明
第41页的『dtw_getvar()』	从 Net.Data 初始化文件中检索配置变量的值。

用于表格操作的应用函数

使用表格函数来处理传送给语言环境的 Net.Data 宏表格变量。

行号与列号以 (1) 开头。

表 7. 表格应用函数

应用函数	说明
第56页的 『 dtw_table_New() 』	创建一个表格对象。
第48页的 『 dtw_table_Delete() 』	删除一个表格对象。
第59页的 『 dtw_table_SetCols() 』	设置表格的宽度并为列标题分配存储器。
第52页的 『 dtw_table_GetV() 』	检索一个表格值。
第61页的 『 dtw_table_SetV() 』	设置一个表格值。
第51页的 『 dtw_table_GetN() 』	检索一个表格列标题。
第60页的 『 dtw_table_SetN() 』	设置一个表格列标题。
第58页的 『 dtw_table_Rows() 』	检索表格中当前的行号。
第47页的 『 dtw_table_Cols() 』	检索表格中当前的列号。
第55页的 『 dtw_table_MaxRows() 』	检索表格中允许的最大行数。
第57页的 『 dtw_table_QueryColnoNj() 』	检索某一列的列号。
第46页的 『 dtw_table_AppendRow() 』	在表格结尾添加一行或多行。
第54页的 『 dtw_table_InsertRow() 』	在表格中插入一行或多行。
第50页的 『 dtw_table_DeleteRow() 』	从表格中删除一行或多行。
第53页的 『 dtw_table_InsertCol() 』	在表格中插入一列或多列。
第49页的 『 dtw_table_DeleteCol() 』	从表格中删除一列或多列。

用于行操作的应用函数

在“每次一行”的处理过程中，行应用函数将处理传送给语言环境的 dtw_getNextRow() 接口函数的行对象。

行号以 (1) 开头。

表 8. 行应用函数

应用函数	说明
第43页的 『 dtw_row_SetCols() 』	设置一行的宽度。
第44页的 『 dtw_row_SetV() 』	设置一个表格值。

应用函数语法参考

本节将描述每个应用函数及其格式、用法、参数，并将提供一个简单的示例。

dtw_free()

使用法

使用 dtw_malloc() 从 Net.Data 的运行时间堆阵释放已经分配的存储器。缓冲区指向要释放的已分配存储器。

格式

```
void dtw_free(void *buffer)
```

参数

buffer 一个指向要释放的已分配存储器的指针。

例

```
char *myBuf;  
long  nbytes = 8192;  
  
myBuf = (char *)dtw_malloc(nbytes);  
  
dtw_free((void *)myBuf);
```

dtw_getvar()

使用法

从 `Net.Data` 初始化文件中检索 `var_name` 指定的配置变量的值。`Net.Data` 拥有 `dtw_getvar()` 所返回的内存；不作修改也不释放。

格式

```
char *dtw_getvar(char *var_name)
```

参数

<i>var_name</i>	要检索的配置变量的名称。
-----------------	--------------

例

```
char *myBindFile;  
  
myBindFile = dtw_getvar("BIND_FILE");
```

dtw_malloc()

使用法

返回一个指向使用 `dtw_malloc()` 从 `Net.Data` 的运行时间堆阵分配的存储器的指针。存储器有 n 个字节长。如果 `Net.Data` 无法返回请求的存储器，则将返回一个空指针。

格式

```
void *dtw_malloc(long nbytes)
```

参数

nbytes 要分配的字节数。

例

```
char *myBuf;  
long  nbytes = 8192;  
  
myBuf = (char *)dtw_malloc(nbytes);
```

dtw_row_SetCols()

使用法

指定行宽并为列标题分配存储器。您可以对每一行使用一次 dtw_row_SetCols() 应用函数。

格式

```
int dtw_row_SetCols(void *row, int cols)
```

参数

<i>row</i>	一个指向新创建的行的指针，这一行中尚未分配任何列。
<i>cols</i>	要在新行中分配的初始列数。

例

```
void *myRow;  
rc = dtw_row_SetCols(myRow, 5);
```

dtw_row_SetV()

使用法

指定一个表格值。dtw_row_SetV() 应用函数的调用程序保留 *src* 所指向的内存的所有权。要删除当前的表格值, 可将这个值指定为 NULL。

格式

```
int dtw_row_SetV(void *row, char *src, int col)
```

参数

<i>row</i>	一个指向要修改的行的指针。
<i>src</i>	包含要设置的新值的字符串。
<i>col</i>	要设置值的列号。

例

```
void *myTable;  
char *myFieldValue = "newValue";  
  
rc = dtw_row_SetV(myRow, myFieldValue, 3);
```

dtw_strdup()

使用法

使用 `dtw_malloc()` 从 `Net.Data` 的运行时间堆阵中分配存储器，并将 *string* 指定的字符串复制到已分配的存储器中。如果 `Net.Data` 无法返回请求的存储器，则将返回一个空指针。

格式

```
char *dtw_strdup(char *string)
```

参数

string 一个指向要复制到分配的存储器中去的字符串值的指针。

例

```
char *myString = "This string will be duplicated.";
char *myDupString;

myDupString = dtw_strdup(myString);
```

dtw_table_AppendRow()

使用法

在表格结尾添加一行或多行。向表格中追加行之后，使用 dtw_table_SetV() 实用程序来指定新行的值。

格式

```
int dtw_table_AppendRow(void *table, int rows)
```

参数

<i>table</i>	一个指向要追加行的表格的指针。
<i>rows</i>	要追加的行数。

例

```
void *myTable;  
  
rc = dtw_table_AppendRow(myTable, 10);
```


dtw_table_Cols()

使用法

返回表格中当前的列数。

格式

```
int dtw_table_Cols(void *table)
```

参数

table 一个指向返回当前列数的表格的指针。

例

```
void *myTable;  
int currentColumns;  
  
currentColumns = dtw_table_Cols(myTable);
```

dtw_table_Delete()

使用法

删除所有的列标题、表格值以及表格对象。

格式

```
int dtw_table_Delete(void *table)
```

参数

table 一个指向要删除的表格的指针。

例

```
void *myTable;  
rc = dtw_table_Delete(myTable);
```

dtw_table_DeleteCol()

使用法

删除一个或多个以 *start_col* 中指定的列开头的列。要删除一个表格中所有的行与列，可用应用函数 `dtw_table_Cols()` 来替换 *cols* 参数。

```
dtw_table_DeleteCol(table, 1, dtw_table_Cols());
```

格式

```
int dtw_table_DeleteCol(void *table, int start_col, int cols)
```

参数

<i>table</i>	一个指向要修改的表格的指针。
<i>start_col</i>	要删除的第一列的列号。
<i>rows</i>	要删除的列数。

例

```
void *myTable;  
  
rc = dtw_table_DeleteCol(myTable, 1, 10);
```

dtw_table_DeleteRow()

使用法

删除一个或多个以 *start_row* 中指定的行开头的行。

格式

```
int dtw_table_DeleteRow(void *table, int start_row, int rows)
```

参数

<i>table</i>	一个指向要修改的表格的指针。
<i>start_row</i>	要删除的第一行的行号。
<i>rows</i>	要删除的行数。

例

```
void *myTable;  
  
rc = dtw_table_DeleteRow(myTable, 3, 10);
```

dtw_table_GetN()

使用法

检索列标题。Net.Data 拥有 *dest* 所指向的内存；不作修改也不释放。

格式

```
int dtw_table_GetN(void *table, char **dest, int col)
```

参数

<i>table</i>	一个指向从中检索到列标题的表格的指针。
<i>dest</i>	一个指向要包含列标题的字符串的指针。
<i>col</i>	列标题的列号。

例

```
void *myTable;  
char *myColumnHeading;  
  
rc = dtw_table_GetN(myTable, &myColumnHeading, 5);
```

dtw_table_GetV()

使用法

从表格中检索值。Net.Data 拥有 *dest* 所指向的内存；不作修改也不释放。

格式

```
int dtw_table_GetV(void *table, char **dest, int row, int col)
```

参数

<i>table</i>	一个指向从中检索到值的表格的指针。
<i>dest</i>	一个指向要包含该值的字符串的指针。
<i>row</i>	要检索的值的行号。
<i>col</i>	要检索的值的列号。

例

```
void *myTable;  
char *myTableValue;  
  
rc = dtw_table_GetV(myTable, &myTableValue, 3, 5);
```

dtw_table_InsertCol()

使用法

在指定的列后插入一列或多列。

格式

```
int dtw_table_InsertCol(void *table, int after_col, int cols)
```

参数

<i>table</i>	一个指向要修改的表格的指针。
<i>after_col</i>	新列插入之后的列号。要在表格的开头插入列，请指定 0。
<i>cols</i>	要插入的列数。

例

```
void *myTable;  
  
rc = dtw_table_InsertCol(myTable, 3, 10);
```

dtw_table_InsertRow()

使用法

在指定的行后插入一行或多行。

格式

```
int dtw_table_InsertRow(void *table, int after_row, int rows)
```

参数

<i>table</i>	一个指向要修改的表格的指针。
<i>after_row</i>	新行插入之后的行号。要在表格的开头插入行，请指定 0。
<i>rows</i>	要插入的行数。

例

```
void *myTable;  
  
rc = dtw_table_InsertRow(myTable, 3, 10);
```


dtw_table_MaxRows()

使用法

返回 `Net.Data` 表格所允许的最大行数，如同 `dtw_table_New()` 应用函数的参数 `row_lim` 所定义的那样。

格式

```
int dtw_table_MaxRows(void *table)
```

参数

<i>table</i>	一个指向返回最大行数的表格的指针。
--------------	-------------------

例

```
void *myTable;
int maximumRows;
```

```
maximumRows = dtw_table_MaxRows(myTable);
```

dtw_table_New()

使用法

创建一个 Net.Data 表格对象并将所有的列标题和字段值初始化为 NULL。调用程序指定行、列的初始数目以及最大行数。如果行、列的初始数目为 0，则必须在任何表格函数调用之前使用 dtw_table_SetCols() 函数来指定一行中的字段数目。

格式

```
int dtw_table_New(void **table, int rows, int cols, int row_lim)
```

参数

<i>table</i>	新表格的名称。
<i>rows</i>	要在新表格中分配的初始行数。
<i>cols</i>	要在新表格中分配的初始列数。
<i>row_lim</i>	此表能够包含的最大行数。

例

```
void *myTable;  
  
rc = dtw_table_New(&myTable, 20, 5, 100);
```

dtw_table_QueryColnoNj()

使用法

返回与列标题相关联的列号。

格式

```
int dtw_table_QueryColnoNj(void *table, char *name)
```

参数

<i>table</i>	一个指向要查询的表格的指针。
<i>name</i>	一个指定列标题的字符串(对于这个列标题返回了列号)。如果表格中不存在列标题, 则返回 0。

例

```
void *myTable;  
int columnNumber;  
  
columnNumber = dtw_table_QueryColnoNj(myTable, "column 1");
```

dtw_table_Rows()

使用法

返回表格中当前的行数。

格式

```
int dtw_table_Rows(void *table)
```

参数

table 一个指向返回当前行数的表格的指针。

例

```
void *myTable;  
int currentRows;  
  
currentRows = dtw_table_Rows(myTable);
```

dtw_table_SetCols()

使用法

设置表格的列数并为列标题分配存储器。请在创建表格时指定列标题；否则，就必须在使用任何其它表格函数之前调用这个应用函数来指定列标题。对于每个表格，您只能使用 dtw_table_SetCols() 应用函数一次。之后，可以使用 dtw_table_DeleteCol() 或 dtw_table_InsertCol() 应用函数。

格式

```
int dtw_table_SetCols(void *table, int cols)
```

参数

<i>table</i>	一个指向没有分配列或行的新表的指针。
<i>cols</i>	要在新表格中分配的初始列数。

例

```
void *myTable;  
  
rc = dtw_table_SetCols(myTable, 5);
```

dtw_table_SetN()

使用法

给列标题指定一个名称。dtw_table_SetN() 应用函数的调用程序保留 *src* 参数所指向的内存的所有权。要删除列标题，可将列标题指定为 NULL。

格式

```
int dtw_table_SetN(void *table, char *src, int col)
```

参数

<i>table</i>	一个指向已指定了列标题的表格的指针。
<i>src</i>	一个要分配给新的列标题的字符串。
<i>col</i>	列号。

例

```
void *myTable;  
char *myColumnHeading = "newColumnHeading";  
  
rc = dtw_table_SetN(myTable, myColumnHeading, 5);
```

dtw_table_SetV()

使用法

在表格中指定一个值。dtw_table_SetV() 应用函数的调用程序保留 *src* 参数所指向的内存的所有权。要删除表格值，可将该值指定为 NULL。

格式

```
int dtw_table_SetV(void *table, char *src, int row, int col)
```

参数

<i>table</i>	一个指向要为其分配值的表格的指针。
<i>src</i>	一个指定给新值的字符串。
<i>row</i>	新值的行号。
<i>col</i>	新值的列号。

例

```
void *myTable;  
char *myTableValue = "newValue";  
  
rc = dtw_table_SetV(myTable, myTableValue, 3, 5);
```


附录A. 语言环境模板

使用这个模板来创建您自己的语言环境。

```

/*****
/*
/* 文件名
/*
/* 说明
/*
/* 功能
/*
/* 入口点
/*
/* 更改活性
/*
/* 标志      原因      日期      开发者      说明
/* -----
/*
*****/

/*-----*/
/* 包含文件
/*-----*/
#include "dtwle.h"
```

图 2. 语言环境模板 (1/14)

```

#ifdef __MVS__
#pragma export(dtw_initialize)
#pragma export(dtw_execute)
#pragma export(dtw_getNextRow)
#pragma export(dtw_cleanup)
#endif

#ifdef __AIX__
/*-----*/
/* 函数 */
/*    dtw_getFp */
/* */
/* 目的 */
/*    设置函数的指针，使之指向此语言环境所提供的所有语言环境接口 */
/*    例程。如果结构中的某个例程没有提供，则将该字段设置为 NULL。 */
/* */
/* 格式 */
/*    int dtw_getFp(dtw_fp_t *func_pointer) */
/* */
/* 参数 */
/*    func_pointer    一个指向结构的指针，该结构中包含此语言环境 */
/*                   为所有函数提供的函数指针。 */
/* */
/* 返回 */
/*    成功 ..... 0 */
/*    失败 ..... -1 */
/*-----*/
int dtw_getFp(dtw_fp_t *func_pointer)
{
    func_pointer->dtw_initialize_fp = dtw_initialize;
    func_pointer->dtw_execute_fp = dtw_execute;
    func_pointer->dtw_getNextRow_fp = dtw_getNextRow;
    func_pointer->dtw_cleanup_fp = dtw_cleanup;
    return 0;
}
#endif

```

图 2. 语言环境模板 (2/14)

```

/*-----*/
/*
/* 函数
/* dtw_initialize
/*
/* 目的
/*
/* 格式
/* int dtw_initialize(dtw_lei_t *le_interface)
/*
/* 参数
/* le_interface 一个指向结构的指针，该结构中包含以下字段:
/*
/* function_name
/* flags
/* exec_statement
/* parm_data_array
/* default_error_message
/* le_opaque_data
/* row
/*
/* 返回
/* 成功 ..... 0
/* 失败 ..... 0
/*-----*/
int dtw_initialize(dtw_lei_t *le_interface)
{
    return rc;
}

```

图 2. 语言环境模板 (3/14)

```

/*-----*/
/*
/* 函数
/* dtw_execute
/*
/* 目的
/*
/* 格式
/* int dtw_execute(dtw_lei_t *le_interface)
/*
/* 参数
/* le_interface 一个指向结构的指针，该结构中包含以下字段:
/*
/* function_name
/* flags
/* exec_statement
/* parm_data_array
/* default_error_message
/* le_opaque_data
/* row
/*
/* 返回
/* 成功 ..... 0
/* 失败 ..... 0
/*-----*/
int dtw_execute(dtw_lei_t *le_interface)
{
    /*-----*/
    /* 确定是否指定了 %exec 语句。
    /*-----*/
    if (le_interface->flags & DTW_STMT_EXEC) {
        /*-----*/
        /* 分析 %exec 语句
        /*-----*/
        rc = processExecStmt(le_interface->exec_statement);
        if (rc)
        {
        }
    }
    else {
        /*-----*/
        /* 分析在线数据
        /*-----*/
        rc = processInlineData(le_interface->exec_statement);
        if (rc)
        {
        }
    }
}

```

图2. 语言环境模板 (4/14)

```

/*-----*/
/* 分析输入参数 */
/*-----*/
rc = processInputParms(le_interface->parm_data_array);
if (rc)
{
}
/*-----*/
/* 处理请求 */
/*-----*/
rc = processRequest();
if (rc)
{
}
/*-----*/
/* 处理输出数据 */
/*-----*/
rc = processOutputParms(le_interface->parm_data_array);
if (rc)
{
}
/*-----*/
/* 处理返回码与缺省的错误消息 */
/*-----*/
if (rc)
{
    setErrorMessage(rc, &(le_interface->default_error_message));
}
/*-----*/
/* 清理并退出程序。 */
/*-----*/
return rc;
}

```

图 2. 语言环境模板 (5/14)

```

/*-----*/
/*
/* 函数
/* dtw_getNextRow
/*
/* 目的
/*
/* 格式
/* int dtw_getNextRow(dtw_lei_t *le_interface)
/*
/* 参数
/* le_interface 一个指向结构的指针，该结构中包含以下字段:
/*
/*     function_name
/*     flags
/*     exec_statement
/*     parm_data_array
/*     default_error_message
/*     le_opaque_data
/*     row
/*
/* 返回
/* 成功 ..... 0
/* 失败 ..... 0
/*-----*/
int dtw_getNextRow(dtw_lei_t *le_interface)
{
    return rc;
}

```

图 2. 语言环境模板 (6/14)

```

/*-----*/
/*
/* 函数
/* dtw_cleanup
/*
/* 目的
/*
/* 格式
/* int dtw_cleanup(dtw_lei_t *le_interface)
/*
/* 参数
/* le_interface 一个指向结构的指针，该结构中包含以下字段:
/*
/* function_name
/* flags
/* exec_statement
/* parm_data_array
/* default_error_message
/* le_opaque_data
/* row
/*
/* 返回
/* 成功 ..... 0
/* 失败 ..... 0
/*-----*/
int dtw_cleanup(dtw_lei_t *le_interface)
{
    /*-----*/
    /* 确定这是正常终止还是异常终止。
    /*-----*/
    if (le_interface->flags & DTW_END_ABNORMAL) {
        /*-----*/
        /* 执行异常终止清理。
        /*-----*/
    }
    else {
        /*-----*/
        /* 执行正常终止清理。
        /*-----*/
    }

    return rc;
}

```

图 2. 语言环境模板 (7/14)

```

/*-----*/
/*
/* 函数
/* processInputParms
/*
/* 目的
/*
/* 格式
/* unsigned long processInputParms(dtw_parm_data_t *parm_data)
/*
/* 参数
/* dtw_parm_data_t *parm_data
/*
/* 返回
/* 成功 ..... 0
/* 失败 .....
/*
/*-----*/
unsigned long processInputParms(dtw_parm_data_t *parm_data)
{
    /*-----*/
    /* 在参数数据数组中的所有变量之间循环。
    /* 数组以一个空条目终止，意为 parm_name 字段设置为 NULL，
    /* parm_value 字段设置为 NULL，parm_descriptor 字段设置为 0。
    /* 但是，对于参数数据数组结尾唯一的有效检查是检查是否
    /* parm_descriptor == 0，这是因为在将文字串传送进来的时候
    /* parm_name 字段为 NULL，而在将未声明的变量传送进来时
    /* parm_value 字段被设置为 NULL。
    /*-----*/
    for (; parm_data->parm_descriptor != 0; ++parm_data) {

```

图 2. 语言环境模板 (8/14)


```

/*-----*/
/* 确定每个输入参数的用法。 */
/*-----*/
switch(parm_data->parm_descriptor & DTW_USAGE) {

    case(DTW_IN):
        /*-----*/
        /* 确定每个输入参数的类型。 */
        /*-----*/
        switch (parm_data->parm_descriptor & DTW_TYPE) {
            case DTW_STRING:
                break;
            case DTW_TABLE:
                break;
            default:
                /*-----*/
                /* 内部错误 - 未知的数据类型 */
                /*-----*/
                break;
        }
        break;

    case(DTW_OUT):
        break;

    case(DTW_INOUT):
        break;

        default:
            /*-----*/
            /* 内部错误 - 未知的用法 */
            /*-----*/
            break;
    }
}
return rc;
}

```

图 2. 语言环境模板 (9/14)

```

/*-----*/
/*                                          */
/* 函数                                          */
/*    processOutputParms()                      */
/*                                          */
/* 目的                                          */
/*                                          */
/* 格式                                          */
/*    unsigned long processOutputParms(dtw_parm_data_t *parm_data) */
/*                                          */
/* 参数                                          */
/*    dtw_parm_data_t *parm_data                */
/*                                          */
/* 返回                                          */
/*    成功 ..... 0                            */
/*    失败 ..... -1                          */
/*                                          */
/*-----*/
unsigned long processOutputParms(dtw_parm_data_t *parm_data) {
    /*-----*/
    /* 以某种特定于语言环境的方式获取输出数据。          */
    /* 这完全取决于和哪个语言环境进行连接以及 LE 选择如何与之连接。 */
    /*-----*/
}

```

图 2. 语言环境模板 (10/14)

```

/  /*-----*/
/* 在参数数据数组中的所有变量之间循环，查找输出参数。 */
/*-----*/
for (; parm_data->parm_descriptor != 0; ++parm_data) {

    /*-----*/
    /* 确定每个参数的用法。 */
    /*-----*/
    if (pd_i->parm_descriptor & DTW_OUT) {
        /*-----*/
        /* 确定每个输入参数的类型。 */
        /*-----*/
        switch (pd_i->parm_descriptor & DTW_TYPE) {
            case DTW_STRING:
                /*-----*/
                /* 给字符串参数一个新的值。如果目前参数值不为 */
                /* NULL，则必须使用 LE 接口应用函数来释放存储 */
                /* 器(如果它是由 Net.Data 分配的)。 */
                /*-----*/
                if (parm_data->parm_value != NULL)
                    dtw_free(parm_data->parm_value);
                parm_data->parm_value = dtw_strdup(newValue);
            break;
            case DTW_TABLE:
                /*-----*/
                /* 更改表格参数的大小。使用 LE 接口应用函数来 */
                /* 修改表格对象。 */
                /*-----*/
                /*-----*/
                /* 首先获取指向表格对象的指针。 */
                /*-----*/
                void *myTable = (void *) parm_data->parm_value;

```

图 2. 语言环境模板 (11/14)

```

/*-----*/
/* 接下去获取表格当前的大小。 */
/*-----*/
cols = dtw_table_Cols(myTable);
rows = dtw_table_Rows(myTable);
/*-----*/
/* 现在来设置新的大小(假定新的大小值有效)。 */
/*-----*/

/*-----*/
/* 先设置列。 */
/*-----*/
if (cols > newColValue)
{
    dtw_table_DeleteCol(myTable,
                        newColValue + 1,
                        cols - newColValue);
}
else if (cols < new_col_value)
{
    dtw_table_InsertCol(myTable,
                       cols,
                       newColValue - cols);
}

/*-----*/
/* 现在设置行。 */
/*-----*/
if (newColValue > 0) {
    if (rows > newRowValue)
    {
        dtw_table_DeleteRow(myTable,
                            newRowValue + 1,
                            rows - newRowValue);
    }
    else if (rows < new_row_value)
    {
        dtw_table_InsertRow(myTable,
                            rows,
                            newRowValue - rows);
    }
}
}

```

图 2. 语言环境模板 (12/14)

```

/*-----*/
/* 现在获取最后一行/列的值。 */
/*-----*/
dtw_table_GetV(myTable,
               &myValue;,
               newRowValue,
               newColValue);

/*-----*/
/* 删除最后一行/列的值。 */
/*-----*/
dtw_table_SetV(myTable,
               NULL,
               newRowValue,
               newColValue);

/*-----*/
/* 设置最后一行/列的值。 */
/*-----*/
dtw_table_SetV(myTable,
               dtw_strdup(myNewValue),
               newRowValue,
               newColValue);

break;
default:
/*-----*/
/* 内部错误 - 未知的数据类型 */
/*-----*/
break;
}
}
return 0;
}

```

图 2. 语言环境模板 (13/14)

```

/*-----*/
/*
/* 函数
/*  setErrorMessage()
/*
/* 目的
/*
/* 格式
/*  unsigned long setErrorMessage(int returnCode,
/*                                char **defaultErrorMessage)
/*
/* 参数
/*  int    returnCode
/*  char **defaultErrorMessage
/*
/* 返回
/*  成功 ..... 0
/*  失败 ..... -1
/*
/*-----*/
unsigned long setErrorMessage(int returnCode,
                             char **defaultErrorMessage)
{
    /*-----*/
    /* 根据返回码设置缺省的错误消息。
    /*-----*/
    switch(returnCode) {
        case LE_SUCCESS:
            break;
        case LE_RC1:
            *defaultErrorMessage = dtw_strdup(LE_RC1_MESSAGE_TEXT);
            break;
        case LE_RC2:
            *defaultErrorMessage = dtw_strdup(LE_RC2_MESSAGE_TEXT);
            break;
        case LE_RC3:
            *defaultErrorMessage = dtw_strdup(LE_RC3_MESSAGE_TEXT);
            break;
        case LE_RC4:
            *defaultErrorMessage = dtw_strdup(LE_RC4_MESSAGE_TEXT);
            rc = LE_RC1INTERNAL;
            break;
    }
    return 0;
}

```

图2. 语言环境模板 (14/14)

附录B. 注意事项

本信息是为在美国提供的产品和服务而开发的。IBM 在其它国家也许没有提供本文档中所讨论的产品、服务或功能部件。关于您所在区域目前可用的产品及服务的信息，请向当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并不说明或暗示只能使用 IBM 的产品、程序或服务。凡是同等功能的产品、程序或服务，只要不侵犯 IBM 的知识产权，都可以用来替代 IBM 产品、程序或服务。当然，评估和验证非 IBM 产品、程序或服务均由用户自行负责。

本文档的议题可能涉及 IBM 的某些专利或正在申请中的专利的应用。提供此文档并不给予您使用这些专利的任何许可。您可以将许可证查询以书面形式发送给：

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

关于双字节 (DBCS) 许可证查询的信息，请与您所在国家的 IBM 知识产权部分联系，或通过写信将查询邮寄至：

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

以下段落不适用于英国与其它当地法律不允许这种供应方式的国家：国际商用机器公司『按现在的样子』出版此书，不做任何明确或暗示的担保，包括但不限于可销售性或适用于特殊目的暗示担保。一些地区在某些事务中不允许放弃明确或暗示的担保，因此本条款可能不适合您。

本信息中可能有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；这些信息将包含在本书新的版本中。IBM 可以在任何时间对本书中说明的产品或程序进行改进，而不必通知您。

已经获得这个程序许可证的用户，如果希望得到有关的更多信息，以允许：(i) 在独立创建的程序和其它程序(包括本程序)之间交换信息，以及 (ii) 相互使用已经交换的信息，请联络：

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

这些信息可以通过遵循相应的条款和条件来获取，在某些情况下，需支付一定的费用。

这些信息中描述的特许程序及其所有可用的特许资料，按 IBM 客户协议 (IBM Customer Agreement) 或任何等价的协议中的条款，由 IBM 提供。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版宣布或其它公众可用源得到。IBM 未测试这些产品，因此不能确认性能、兼容性或其它关于非 IBM 产品承诺等的准确度。有关非 IBM 产品功能方面的问题可向它们的供应商提出。

版权许可证:

本信息中包含用源语言编写的示例应用程序，它们说明了各种不同的操作平台上的程序设计技术。您可以为了开发、使用、市场营销或分发应用程序(这些应用程序遵守编写这些示例程序的操作平台的应用程序接口)的目的，以任何形式复制、修改和分发这些示例程序，不用向 IBM 付费。这些例子未经所有条件下的完整测试。因此，IBM 不能保证或暗示其可靠性、可用性或这些程序的功能。您可以为了开发、使用、市场营销或分发应用程序(这些应用程序遵守编写这些示例程序的操作平台的应用程序接口)的目的，以任何形式复制、修改和分发这些示例程序，不用向 IBM 付费。

商标

以下术语是 IBM 公司在美国或其他国家的注册商标:

AIX	Lotus
DataJoiner	MVS
DB2	Net.Data
Domino	OS/2
IBM	OS/390
IMS	OS/400

以下术语是其它公司的商标:

Java 和 HotJava 是 SUN 公司的商标。

Microsoft、Windows、Windows NT® 和 Windows 95 标志都是 Microsoft 公司的注册商标。

UNIX 是在美国和其它国家的注册商标，许可权属于 X/Open 有限公司专有。

其它公司、产品和服务名称可能是其它公司的商标或服务标志。

词汇表

API. Application programming interface 的缩写，应用程序设计接口。

applet (小应用程序). 包含在 HTML 页中的一段 Java 程序。在支持 Java 的浏览器(例如 Netscape) 中可以运行小应用程序，它是在装入 HTML 页时装入的。

application programming interface (API, 应用程序设计接口). 由操作系统或可单独订购的特许应用程序提供的一个功能接口，它允许以高级语言编写的应用程序可以使用特定于操作系统或特许程序的数据。Net.Data 支持以下这些专门的 Web 服务器 API，这些 API 用于 CGI 进程中的改进性能：ICAPI、GWAPI、ISAPI 和 NSAPI。

BLOB. Binary large object 的缩写，二进制大型对象。

cache (高速缓存). 一类包含最近访问过的数据的内存，是为了加快对相同数据的后继访问而设计的。高速缓存经常是用来对网络中可以访问的、频繁使用的数据保留一个本地副本。

caching (高速缓存). 将频繁使用的结果(来自对 Web 服务器的请求)存储在本地以备快速检索的过程，直到刷新信息时为止。

Cache Manager (高速缓存管理器). 为一台机器管理高速缓存的程序。它可以管理多个高速缓存。

CGI. Common Gateway Interface 的缩写，公共网关接口。

cliette. 为来自 Web 服务器的请求提供服务的一个长时间运行的进程。连接管理器负责调度 cliette 进程，使其为这些请求提供服务。

CLOB. Character large object 的缩写，字符大型对象。

Common Gateway Interface (公共网关接口). Web 服务器将控制传递给一个应用程序以及接收回数据的一种标准方法。

Connection Manager (连接管理器). Net.Data 中的一个可执行文件 dtwcm，用于支持“现场连接”。

cookie. 一个信息包，由 HTTP 服务器发送给 Web 浏览器，然后在浏览器每次访问该服务器时发回。Cookies 中可以包含服务器所选择的任意信息，用于维持否则将没有状态的 HTTP 事务之间的状态。*计算的自由联机字典*

database (数据库). 表格的一个集合，或表格空间和索引空间的一个集合。

database management system (DBMS, 数据库管理系统). 用于控制创建、组织和修改一个数据库，并控制对其存储的数据进行访问的一个软件系统。

data type (数据类型). 列和字面量的属性。

DBMS. Database management system 的缩写，数据库管理系统。

firewall (防火墙). 一台装有软件的计算机，用于防止外部未经授权计算机侵入内部网络。

flat file interface (平面文件接口). 一系列 Net.Data 内部函数，可让您在明文文件中读写数据。

HTML. Hypertext markup language 的缩写，超文本标记语言。

HTTP. Hypertext transfer protocol 的缩写，超文本传送协议。

hypertext markup language (超文本标记语言). 一种用于编写 Web 文档的标记语言。

hypertext transfer protocol (超文本传送协议). 一种在 Web 服务器和浏览器之间使用的通信协议。

ICAPI. Internet Connection API 的缩写。

ICS. Internet Connection Server 的缩写。

ICSS. Internet Connection Secure Server 的缩写。

Internet. 国际公用 TCP/IP 计算机网络。

Internet Connection Server. IBM 公司的公开 Web 服务器。

Internet Connection Secure Server. IBM 公司的安全 Web 服务器。

Intranet. 在公司防火墙内部的 TCP/IP 网络。

ISAPI. Microsoft 公司的 Internet Server API。

Java. 一种独立于操作系统的面向对象的程序设计语言，特别适用于 Internet 应用程序。

language environment (语言环境). 一个模块，提供从 Net.Data 宏到外部数据源(例如 DB2 或诸如 Perl 等程序设计语言)的访问。有一些语言环境是与 Net.Data 一起提供的，例如 REXX、Perl 和 Oracle。您还可以创建自己的语言环境。

Live Connection (现场连接). 一种 Net.Data 配置, 与连接管理器和 Web 服务器 API 一起使用。现场连接使得数据库连接成为可重用的。

LOB. Large object 的缩写, 大型对象。

middleware (中件). 一种介于应用程序与网络之间的软件。它用于管理多机种计算操作系统中那些完全不同的应用程序之间的交互。*计算的自由联机字典*

NSAPI. Netscape API 的缩写。

null (空值). 表示信息异常的一个特殊值。

path (路径). 用于查找文件的搜索路径。

Perl. 一种解释性编程语言。

port (端口). 一个 16 位数, 用于在 TCP/IP 和高级协议或应用程序之间进行通信。

TCP/IP. Transmission Control Protocol / Internet Protocol 的缩写, 传输控制协议/网际协议。

Transmission Control Protocol / Internet Protocol (传输控制协议/网际协议). 一组通信协议, 同时支持局域网和广域网中的点对点连接功能。

URL. Uniform resource locator 的缩写, 统一资源定位器。

uniform resource locator (统一资源定位器). 一个用于命名 HTTP 服务器和(可选的)目录及文件名的地址, 例如: <http://www.software.ibm.com/data/net.data/index.html>。

Web server (Web 服务器). 一台运行 http 服务器软件(例如 Internet Connection) 的计算机。

索引

本索引按汉语拼音, 数字, 英文字母和特殊字符顺序排列。

[B]

变量

传递 30

释放 34

表格

操作应用函数 38

创建新表 56

删除 48

追加行 46

表格值

检索 52

删除 48, 52, 61

指定 44, 61

[C]

参数

传递 30, 31

命名 31

指定 31

parm_name 31

初始化任务, 语言环境 32, 33

传递

变量 30

参数 30

创建表格 56

词汇表 78

存储器

分配 42, 43, 45, 59

释放 30, 32, 40

dtw_lei 标志 30

错误条件 29

错误条件消息 30

[D]

调用小应用程序 6

动态存储器分配 32

堆阵, Net.Data 运行时间 32

[H]

行

插入 54

返回 31, 32, 33

行 (续)

返回允许的最大行数 55

检索当前数目 58

删除 49, 50

指定宽度 43

追加 46

dtw_getNextRow() 接口函数 31

行操作应用函数 38

[J]

结构, 语言环境

dtw_lei 29

dtw_parm_data 31

接口函数

处理顺序 32

语言环境描述 32

dtw_cleanup() 34

dtw_execute() 33

dtw_getNextRow() 33

dtw_initialize() 33

[K]

可执行语句, dtw_lei 标志 30

[L]

列

插入 53

确定表格中的总列数 47

删除 49

指定表格中的列号 59

列标题

返回列号 57

分配存储器 43, 59

检索 51

删除 48, 51, 60

指定名称 60

[M]

模板, 语言环境 63

[N]

内存管理应用函数 37

[P]

配置变量

检索变量值 41

配置变量 (续)
 用于管理的应用函数 37
配置环境 34
平面文件接口
 语言环境 5

[Q]

清理
 处理之后 32, 34
 用于异常条件的标志 30, 34
 dtw_lei 标志 30, 34

[Y]

异常条件
 错误消息 30
 dtw_lei 标志 30, 34
应用函数
 表格操作 38
 行操作 38
 内存管理 37
 配置变量 37
 语言环境 37
 dtw_free() 40
 dtw_getvar() 41
 dtw_malloc() 42
 dtw_row_SetCols() 43
 dtw_row_SetV() 44
 dtw_strdup() 45
 dtw_table_AppendRow() 46
 dtw_table_Cols() 47
 dtw_table_DeleteCol() 49
 dtw_table_DeleteRow() 50
 dtw_table_Delete() 48
 dtw_table_GetN() 51
 dtw_table_GetV() 52
 dtw_table_InsertCol() 53
 dtw_table_InsertRow() 54
 dtw_table_MaxRows() 55
 dtw_table_New() 56
 dtw_table_QueryColnoNj() 57
 dtw_table_Rows() 58
 dtw_table_SetCols() 59
 dtw_table_SetN() 60
 dtw_table_SetV() 61
语言环境
 初始化 32
 处理之后清理 32, 34
 创建 27
 结构 29
 接口函数 32

语言环境 (续)
 接口模板 63
 介绍 37
 配置 34
 平面文件接口 5
 应用函数 37
 语句, 执行 32
 摘要 3
 IMS Web 5
 Java 小应用程序 6
 Java 应用程序 12
 ODBC 15
 Oracle 15
 Perl 18
 REXX 18
 SQL 20
 Sybase 20
 system 22
 Web 注册表 22

[Z]

指向存储器 42
执行语言环境语句 32, 33
致命错误, dtw_lei 标志 30, 34
注意事项 77
最大行数 55
“每次一行”处理
 dtw_getNextRow() 32, 33
 dtw_lei 标志 30
 DTW_LE_CONTINUE 30

D

DB2, SQL 语言环境 20
dtw_ 结构 29
dtw_ 接口函数 32
dtw_ 实用程序 37
DTW_APPLET 6
DTW_FFI 5
DTW_IMS 5
DTW_JAVAPPS 12
dtw_lei
 结构 29
 字段
 标志 30
 函数名 30
 default_error_messages 30
 exec_statement 30
 le_opaque_data 30
 parm_data_array 30
 row 31

DTW_LE_CONTINUE 33
DTW_ODBC 15
DTW_ORA 15
dtw_parm_data
 结构 31
 字段
 parm_descriptor 31
 parm_name 31
 parm_value 31
DTW_PERL 18
DTW_REXX 18
DTW_SQL 20
DTW_SYB 20
DTW_SYSTEM 22
DTW_WEBREG 22

E

ENVIRONMENT 语句
 例子 36
 用于新的语言环境 34
 语法 34

F

FUNCTION 块
 名称 30
 执行语句 32, 33

I

IMS Web
 语言环境 5
 Studio 工具 5

J

Java 小应用程序
 创建 6
 调用 6
 类 10
 例子 9
 生成标记 6
 语言环境 6
Java 应用程序
 创建函数 13
 创建 cliettes 13
 调用 14
 特殊配置 14
 语言环境 12

O

ODBC
 宏文件中的 SQL 语句 15
 语言环境 15
Oracle
 存储过程 15
 访问 16
 特殊配置 16
 语言环境 15

P

parm_data_array 结构, 指定名称 30
Perl
 脚本中的 Net.Data 变量 18
 语言环境 18

R

REXX
 变量替换 19
 语言环境 18

S

SQL
 使用 DataJoiner 20
 语言环境 20
 支持的语句 20
Sybase
 大型对象 20
 访问 20
 特殊配置 20
 语言环境 20
System
 传递变量 22
 语言环境 22

W

Web 注册表
 说明 23
 语言环境 22



Printed in China