Net.Data

语言环境接口参考

Net.Data

语言环境接口参考

注意

在使用本信息及其所支持的产品之前，务必请阅读第57页的『附录D. 注意事项』中的信息。

第六版 **(1999 年 5 月)**

目录

前言

感谢您选择 Net.Data® - IBM® 的开发工具来创建动态的 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广泛种类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。

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、允许使用 ODBC 的数据库和可以通过 DRDA 访问的数据库)的数据来创建动态的 Web 页面，还可以使用各种程序设计语言(例如 Java、JavaScript、Perl、C、C++ 和 REXX)所编写的应用程序。Net.Data 系列产品在执行 Windows NT、AIX、OS/2、OS/390、OS/400、HP-UX、 Sun Solaris、Santa Cruz 操作系统 (SCO) 和 Linux 操作系统的机器上提供了类似的功能。

Net.Data 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、其它数据源、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 Navigator 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:

- 在使用 C、C++、RPG、COBOL、JAVA、Perl 或 REXX 程序设计语言(但不局限于这些语言)所编写的应用程序中封装商业逻辑
- 访问诸如 DB2 等数据库
- 访问其它数据源，例如平面文件

Net.Data 将这个 Web 页面传递到 Web 服务器，随后 Web 服务器通过网络转发这个页面，最后显示在浏览器上。

Net.Data 可以用在配置为使用诸如超文本传输协议 (HTTP) 和公共网关接口 (CGI) 等接口的服务器环境中。HTTP 是一个用于浏览器和 Web 服务器之间交互的工业标准接口， CGI 是一个用于类似 Net.Data 这样的网关应用程序的 Web 服务器调用的工业标准接口。这些接口允许您选择您所喜爱的浏览器或 Web 服务器与 Net.Data 一起使用。

为了改进性能，Net.Data 支持各种各样的 Web 服务应用程序设计接口 (API)。另外，Net.Data 也可作为 Java 小服务程序启动。

关于本书

本书描述 Net.Data 的语言环境接口 (LEI)，您可用它来开发自己所定制的 Net.Data 语言环境。

本书可能引用已经发布、但现在还未进入实用阶段的某些产品或功能。

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

希望扩展 Net.Data 的功能以满足企业特定需要的人可以使用本书来编写自己的 Net.Data 语言环境。

要理解本书中讨论的概念，您应当熟悉以下信息:

- C 程序设计语言
- *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程序设计语言和数据库接口。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。通过使用函数调用来调用语言环境，您可以使用这些语言环境为了与商业应用程序一起使用而提供的功能。例如，您可以直接访问 ODBC 数据库，使用 Perl 语言环境来执行 Perl 脚本，或者调用 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。

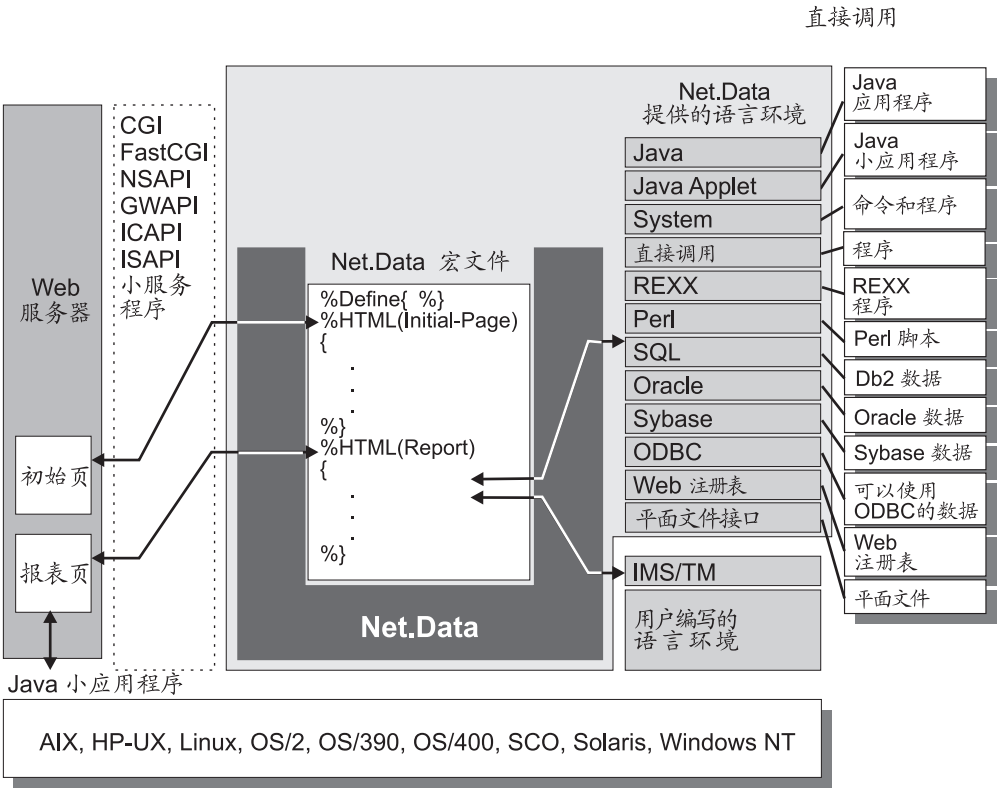

图 1. Net.Data 和语言环境

本书描述用于创建新的语言环境的 Net.Data 语言环境接口。*Net.Data 管理与程序设计指南*针对您的操作系统的语言环境一章中描述了 Net.Data 提供的语言环境。

第1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、共享程序库或依赖于操作系统环境的服务程序。在本文档中，使用术语*共享程序库*来泛指这些类型的文件。 Net.Data 提供了一系列语言环境，但当这些语言环境不能满足您的应用需求时，您还可以创建自己的语言环境并使用 Net.Data 语言环境的接口。

创建新的语言环境涉及到以下步骤:

- 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。
- 构建一个能够实现您希望提供的语言环境接口例程集的共享程序库。请参阅适用于您的编译器的文档，以便了解如何构建共享程序库。
- 使得所有的接口都可以从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。
- 将函数添加到使用新的语言环境的 Net.Data 宏中。

在您决定创建一个新的语言环境之前，请先确定 Net.Data 提供的语言环境是否满足您的需要。

本章将描述如何设计语言环境。

- 『创建共享库』
- 第3页的『语言环境通信结构』
- 第6页的『语言环境接口函数』
- 第8页的『设计语言环境语句』

要学习语言环境程序设计接口，请参阅第11页的『第2章 语言环境程序设计接口应用函数』。

创建共享库

构建一个语言环境时，您可以使用第39页的『附录B. 语言环境模板』中提供的模板，它们提供环境接口函数和通信结构，Net.Data 使用这些函数和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11页的『第2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2页的『我应该提供哪些语言环境?』
- 第2页的『处理输入参数』
- 第2页的『处理用户请求』
- 第3页的『处理输出参数』
- 第3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_t 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() 接口时以及您希望释放任何资源的情况下需要提供 dtw_cleanup() 接口。

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。如果 Net.Data 在 OS/400® 或 OS/390 操作系统上运行，则调用此接口。

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_t 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 Net.Data 初始化文件的 ENVIRONMENT 语句中为语言环境指定隐式参数。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_t 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39页的『附录B. 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_t 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_t 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_t 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_t 结构的 parm_data_array 字段中传递)。程序模板中的 processOutputParms() 例程(第39页的『附录B. 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_t 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39页的『附录B. 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_t
- dtw_parm_data_t

Net.Data 将一个语言环境接口结构(例如，dtw_lei_t) 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_t 结构

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

```
typedef struct dtw_lei_t {              /* 语言环境接口                 */
    char *function_name;               /* 函数块名称                   */
    int   flags;                       /* 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_t 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。

flags 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_t 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_t 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5页的『dtw_parm_data_t 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。

只有 IBM 可以使用 reserved 字段。

dtw_parm_data_t 结构

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

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_t 结构具有以下格式:

```
typedef struct dtw_parm_data_t {        /* 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_t 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 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7页的『dtw_initialize()』
- 第7页的『dtw_execute()』
- 第7页的『dtw_getNextRow()』
- 第8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。

对于每个宏，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 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”的方式处理表数据。如果是，则接口函数将在 dtw_lei_t 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，在 report 块阶段 dtw_execute 接口函数可以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_t 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_t 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。将此函数用于断开与数据库的连接或释放变量等任务。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_t 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_t 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 Net.Data 初始化文件中都有一个 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  ([specification parameter_list, ...)
```

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。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。

  重要: 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。如果使用 DTW 前缀，Net.Data 将无法装入您的语言环境 DLL。

- *library_name*

  对象的名称，包含 Net.Data 调用的语言环境接口。对于每个操作系统来说，文件扩展名是不同的:

  – 在 AIX® 中，共享程序库的名称是用扩展名 *.o* 指定的。
  – 在 HP/UX 中，共享程序库的名称是用扩展名 *.sl* 指定的。
  – 在 OS/2® 和 Windows NT 中，DLL 名称是用扩展名 *.dll* 指定的。
  – 在 OS/390® 中，指定 DLL 名称时不用扩展名 *.dll*。
  – 在 OS/400® 中，服务程序名称是用扩展名 *.SRVPGM* 指定的。
  在 SUN、SCO 和 LINUX 中，共享程序库的名称是用扩展名 *.so* 指定的。

  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。

- *specification*

  参数传递说明，指出 Net.Data 是否使用参数来输入、输出或者既输入又输出。可能的值有:

  IN 一个用于输入的参数

  OUT 一个用于输出的参数

  INOUT 一个可用于输入和输出的参数

- *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 FUNCION 块定义中指定的参数之后，它们也将在 *dtw_lei_t* 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 宏中将这些参数定义为变量。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。

以下示例显示了用于 Net.Data 所提供语言环境的 ENVIRONMENT 语句(使用 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)   () CLIETTE "DTW_JAVAPPS"
ENVIRONMENT (DTW_PERL)     DTWPERL   ()
ENVIRONMENT (DTW_REXX)     DTWREXX   ()
ENVIRONMENT (DTW_SYSTEM)   DTWSYS    ()
ENVIRONMENT (HWS_LE)       DTWHWS    ()
```

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2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39页的『附录B. 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12页的『用于表格操作的应用函数』
- 第12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 1. 内存管理应用函数

应用函数	说明
第16页的『dtw_malloc()』	使用 dtw_malloc() 从 Net.Data 的运行时间堆阵分配存储器。
第14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19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 2. 配置应用函数

应用函数	说明
第15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 3. 表格应用函数

应用函数	说明
第30页的『dtw_table_New()』	创建一个表格对象。
第22页的『dtw_table_Delete()』	删除一个表格对象。
第33页的『dtw_table_SetCols()』	设置表格的宽度并为列标题分配存储器。
第26页的『dtw_table_GetV()』	检索一个表格值。
第35页的『dtw_table_SetV()』	设置一个表格值。
第25页的『dtw_table_GetN()』	检索一个表格列标题。
第34页的『dtw_table_SetN()』	设置一个表格列标题。
第32页的『dtw_table_Rows()』	检索表格中当前的行号。
第21页的『dtw_table_Cols()』	检索表格中当前的列号。
第29页的『dtw_table_MaxRows()』	检索表格中允许的最大行数。
第31页的『dtw_table_QueryColnoNj()』	检索某一列的列号。
第20页的『dtw_table_AppendRow()』	在表格结尾添加一行或多行。
第28页的『dtw_table_InsertRow()』	在表格中插入一行或多行。
第24页的『dtw_table_DeleteRow()』	从表格中删除一行或多行。
第27页的『dtw_table_InsertCol()』	在表格中插入一列或多列。
第23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 4. 行应用函数

应用函数	说明
第17页的『dtw_row_SetCols()』	设置一行的宽度。
第18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

var_name	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

row	一个指向新创建的行的指针，这一行中尚未分配任何列。
cols	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

row	一个指向要修改的行的指针。
src	包含要设置的新值的字符串。
col	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

table	一个指向要追加行的表格的指针。
rows	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

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

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

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

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

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

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

table 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

table	新表格的名称。
rows	要在新表格中分配的初始行数。
cols	要在新表格中分配的初始列数。
row_lim	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

table	一个指向要查询的表格的指针。
name	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

table	一个指向没有分配列或行的新表的指针。
cols	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

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

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

例

```
void *myTable;
char *myTableValue = "newValue";

rc = dtw_table_SetV(myTable, myTableValue, 3, 5);
```

附录A. Net.Data 技术库

Net.Data 技术库可以从 Net.Data 的 Web 站点获得

`http://www.software.ibm.com/data/net.data/library.html`

文档	说明
• *Net.Data 管理与程序设计指南, OS/390 版* • *Net.Data 管理与程序设计指南, OS/2、Windows NT 和 UNIX 版* • *Net.Data 管理与程序设计指南, OS/400 版*	包含有关安装、配置和调用 Net.Data 的概念和任务信息。还描述了如何编写 Net.Data 宏、如何使用 Net.Data 性能技术、如何使用 Net.Data 语言环境、如何管理连接、以及如何使用 Net.Data 记录和跟踪来排除故障、调节性能。
Net.Data 参考	描述 Net.Data 宏语言、变量和内部函数。
Net.Data 语言环境接口参考	描述 Net.Data 语言环境接口。
Net.Data 消息和代码参考	列出 Net.Data 错误消息和返回码。

附录B. 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                                                                    */
/*                                                                           */
/* 更改活动性                                                                */
/*                                                                           */
/* 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C. 构建文件示例

以下示例可作为在各种不同的操作系统上构建 LE 的参考。因为特定的操作系统和环境都有其唯一的特性，这些示例只应用作参考。在构建自己的共享程序库时，请参考您的操作系统上所使用编译器的特定文档。

示例 OS/390 JCL

```
//BLDUSER  JOB ,'BLDUSER ',TIME=1,
// MSGCLASS=H,CLASS=A,
// USER=IBMUSER,MSGLEVEL=(1,1)
//*********************************************************************
//*  COMPILE STEP: C++ DLL     (USER-WRITTEN LE)                      *
//*                C/C++ FOR MVS/ESA(R) COMPILER V3 R2.0              *
//*                AD/CYCLE LE/370 V1 R7.0                            *
//*********************************************************************
//COMPILE EXEC PGM=CBC320PP,REGION=32M,
//          PARM=('/CXX SO,OPT,EXP,SE(''CEEV1R70.SCEEH.+'')',
//          'DEF(_XOPEN_SOURCE_EXTENDED)')
//STEPLIB   DD DSN=CEEV1R70.SCEERUN,DISP=SHR
//          DD DSN=CBCV3R20.SCBC3CMP,DISP=SHR
//SYSMSGS   DD DUMMY,DSN=CBCV3R20.SCBC3MSG(EDCMSGE),DISP=SHR
//SYSXMSGS  DD DUMMY,DSN=CBCV3R20.SCBC3MSG(CBCMSGE),DISP=SHR
//SYSIN     DD DSN=IBMUSER.NETDATA.USERLANG.C(USERLANG),DISP=SHR
//SYSLIB    DD DSN=CBCV3R20.SCLB3H.H,DISP=SHR
//USERLIB   DD DSN=IBMUSER.NETDATA.H,DISP=SHR
//SYSLIN    DD DSN=IBMUSER.NETDATA.USERLANG.OBJ(USERLANG),DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSCPRT   DD SYSOUT=*
//SYSTERM   DD DUMMY
//SYSUT1    DD DSN=&&SYSUT1;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT2    DD DSN=&&SYSUT2;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT3    DD DSN=&&SYSUT3;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4    DD DSN=&&SYSUT4;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5    DD DSN=&&SYSUT5;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT6    DD DSN=&&SYSUT6;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7    DD DSN=&&SYSUT7;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
```

```
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8    DD DSN=&&SYSUT8;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9    DD DSN=&&SYSUT9;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10   DD SYSOUT=*
//SYSUT14   DD DSN=&&SYSUT14;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15  DD  SYSOUT=*
//*
//*********************************************************************
//*  PRELINK STEP: C++ DLL     (USER-WRITTEN LE)                      *
//*                C/C++ FOR MVS/ESA COMPILER V3 R1.0                 *
//*                AD/CYCLE LE/370 V1 R7.0                            *
//*********************************************************************
//PLKED EXEC PGM=EDCPRLK,REGION=32M,COND=(0,NE,COMPILE),
//          PARM='MAP,UPCASE,MEMORY,DLLNAME(userdll)'
//STEPLIB   DD DSN=CEEV1R70.SCEERUN,DISP=SHR
//SYSMSGS   DD DSN=CEEV1R70.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB    DD DSN=CEEV1R70.SCEECPP,DISP=SHR
//SYSMOD    DD DSN=IBMUSER.NETDATA.USERLANG.DLLP(PRLKUSER),
//             DISP=SHR
//OBJECT    DD DSN=IBMUSER.NETDATA.USERLANG.OBJ,DISP=SHR
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSDEFSD  DD DSN=IBMUSER.NETDATA.USERLANG.DEFSD(USEREXP),
//             DISP=SHR
//SYSIN     DD DSN=IBMUSER.NETDATA.DEFSD(DTWLESHR),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(COMPLEX),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(APPSUPP),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(COLLECT),DISP=SHR
//          DD *
    INCLUDE OBJECT(USERLANG)
//*
//*********************************************************************
//*  LINK STEP: C++ DLL     (FFI LANGUAGE ENVIRONMENT)                 *
//*             C/C++ FOR MVS/ESA COMPILER V3 R2.0                     *
//*             AD/CYCLE LE/370 V1 R7.0                                *
//*********************************************************************
//LKED EXEC PGM=HEWL,REGION=2048K,COND=(4,LT,PLKED),
//          PARM='MAP,LIST,XREF,RENT,REUS,COMPAT=PM2'
//SYSLIB    DD DSN=CEEV1R70.SCEELKED,DISP=SHR
//SYSLIN    DD DSN=IBMUSER.NETDATA.USERLANG.DLLP(PRLKUSER),
//             DISP=SHR
//          DD *
            NAME USERDLL(R)
//SYSLMOD   DD DSN=IBMUSER.NETDATA.USERLANG.DLL,DISP=SHR
//SYSUT1    DD DSN=&&SYSUT1;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
```

```
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT  DD SYSOUT=*
//*
//*********************************************************************
//*  COPY STEP:  C++ DLL     (FFI LANGUAGE ENVIRONMENT)                *
//*              C/C++ FOR MVS/ESA COMPILER V3 R2.0                    *
//*              AD/CYCLE LE/370 V1 R7.0                               *
//*********************************************************************
//COPY EXEC PGM=IEWBLINK,REGION=500K,COND=(0,NE,LKED),
//          PARM='LIST,REUS,RENT,NCAL,LET,MAP,CASE=MIXED,COMPAT=PM2'
//SYSPRINT  DD SYSOUT=*
//INLIB     DD DSN=IBMUSER.NETDATA.USERLANG.DLL,DISP=SHR
//*
//SYSLMOD   DD PATH='/usr/lpp/netdata/cgi-bin',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=(SIRWXO,SIRWXG,SIRWXU)
//*
//SYSLIN    DD *
            INCLUDE INLIB(USERDLL)
            ENTRY CEESTART
            NAME userdll(R)
//*
```

示例 makefile (特定于 OS/390)

```
### Target
TARGET          = userlang

### C Compiler
CC              = c89
CFLAGS          = -D_XOPEN_SOURCE_EXTENDED=1 -O -I/u/USER01/netdata/include

### C++ Compiler
CCC             = c++
CCFLAGS         = -D_XOPEN_SOURCE_EXTENDED=1 -O -I/u/USER01/netdata/include

### Linker/Loader
LD              = c++
LDFLAGS         = $(CCFLAGS) -W l,dll

### Sources Headers and Objects
OBJS            = userlang.o
HDRS            = /u/USER01/netdata/include/dtwle.h userlang.h


######################
### 程序库           #
######################

LIBS = /u/USER01/netdata/defsd/dtwleshr.x

######################
### 附加目标         #
######################

all:        $(TARGET)

$(TARGET):  $(OBJS) $(LIBS)
            echo "Linking $(TARGET) ..."
```

```
            $(LD) $(LDFLAGS) -o $(TARGET) $(OBJS) $(LIBS)
            echo "done"

userlang.o: $(HDRS) userlang.c
            $(CCC) $(CCFLAGS) -c userlang.c

clean:
            rm -f $(OBJS) $(TARGET)
```

示例 OS/400 CL

假定有以下条件，使用以下步骤在 AS/400 上构建一个语言环境:

- SRC 是源文件(用 C 编写)。
- MYLE 包含可调出的过程 dtw_execute。
- 文件 QSRVSRC 成员 MYLEEXP 包含了调出过程 dtw_execute 的说明。

1. 创建模块:

   ```
   CRTCMOD MODULE(MYLIB/MYLE) SRCFILE(MYLIB/SRC)
   ```

2. 创建服务程序:

   ```
   CRTSRVPGM SRVPGM(MYLIB/MYLE) MODULE(MYLIB/MYLE)
                                SRCFILE(MYLIB/QSRVSRC) SRCMBR(MYLEEXP)
                                BNDSRVPGM(QTCP/QTMHLE)
   ```

QTCP/QTMHLE 是包含所有 Net.Data 可绑定 API 的服务程序。在版本 4 发行版 3 和后继发行版中，尽管可以使用 QTCP/QTMHLE，您还是应使用 QHTTPSVR/QTMJLE。

附录D. 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，包括但不局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
AS/400
CBIDO
CBPDO
CICS
CustomPac
DB2
DB2 Universal Database
DataJoiner
Distributed Relational Database Architecture
DRDA
IBM
IMS
语言环境
MVS/ESA
Net.Data
OpenEdition
Operating System/400
OS/2
OS/390
OS/400
RACF
SystemPac

以下术语是其它公司的商标:

Java 和所有基于 Java 的商标与标志都是 SUN 公司在美国和其它国家的商标。

Lotus 和 Domino Go Webserver 是 Lotus 公司在美国和其它国家的商标。

Microsoft、Windows、Windows NT 和 Windows 标志是 Microsoft 公司在美国和其它国家的商标或注册商标。

其它公司、产品和服务名称，以两个星号(**)表示，可能是其它公司的商标或服务标志。

词汇表

absolute path (绝对路径). 对象的全路径名。绝对路径名从最高一级开始，或者说从“根”目录(由斜杠 (/) 或反斜杠 (\) 字符标识)开始。

API. Application programming interface 的缩写，应用程序设计接口。Net.Data 支持 Web 服务器 API，以改进在 CGI 处理上的性能。

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 Navigator) 中可以运行小应用程序，它是在处理 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 支持以下这些用于 CGI 进程中经改进的性能的 Web 服务器 API: ICAPI、 GWAPI、ISAPI 和 NSAPI。

cache (高速缓存). 一部分包含最近访问过的数据的内存或磁盘空间，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. Net.Data 现场连接中一个长时间运行的进程，为来自 Web 服务器的请求提供服务。连接管理器负责调度 cliette 进程，使其为这些请求提供服务。

commitment control (确认控制). Net.Data 正在运行的进程内的边界创建，其中对资源的操作是工作单元的一部分。

Common Gateway Interface (CGI，公用网关接口). 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*

current working directory (当前工作目录). 进程的缺省目录，从该目录分辨所有的相对路径名。

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，并控制对其中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。

Domino *Go* Web server (Domino Go Web 服务器). Lotus 公司和 IBM 提供的 Web 服务器，提供正规连接和安全连接。ICAPI 和 GWAPI 是这个服务器提供的接口。

firewall (防火墙). 一台装有软件的计算机，用于防止外部未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。

GWAPI. *Go* Web 服务器 API 的缩写。

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。*请参阅*。

Internet. 国际公用 TCP/IP 计算机网络。

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。

Live Connection (现场连接). 一个 Net.Data 组件，由连接管理器和多个 cliette 组成。现场连接用于管理数据库和 Java 虚拟机连接的再使用。

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。它管理客户应用程序和服务器之间通过网络进行的交互。

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。

path name (路径名). 告诉系统如何找到一个对象。路径名的表示方法是：目录名，后面跟对象的名称。单独的目录和对象名之间用斜杠 (/) 或反斜杠 (\) 字符分隔。

Perl. 一种解释性程序设计语言。

persistence (持续性). 使指定的值在整个事务中得以保持的状态，这里的事务可以跨越多个 Net.Data 调用。只有变量是可以持久性的。另外，在完成一个显式的提交或撤消操作之前，或者在事务结束之前，对受确认控制影响的资源的操作将保持活动。

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。

registry (注册表). 一个可以存储和检索字符串的“仓库”。

relative path name (相对路径名). 不以最高级目录(或“根”目录)开始的路径名。系统假定路径名从进程的当前工作目录开始。

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

transaction (事务). 一个 Net.Data 调用。如果使用持久性的 Net.Data，则一个事务可能跨越多个 Net.Data 调用。

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`。

unit of work (工作单位). 作为一个原子操作的可恢复操作序列。工作单位内的所有操作都可以完成(提交)或取消(撤消)，就如同它们是一个操作。只有那些对受确认控制影响的资源进行的操作才可以提交或撤消。

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。

[Q]

[Y]

[Z]

D

E

F

P

Printed in China