

Net.Data



管理与程序设计指南, OS/400 版

Net.Data



管理与程序设计指南, OS/400 版

注意

在使用本信息及其所支持的产品之前，务必请阅读第113页的『附录C. 注意事项』中的信息。

第六版 (1999 年 5 月)

此版本适用于：

- IBM Operating System/400 (程序 5763-SS1)，版本 3 发行版本 2 修订版 0
- IBM Operating System/400 (程序 5716-SS1)，版本 3 发行版本 7 修订版 0
- IBM TCP/IP Connectivity Utilities for AS/400 (程序 5763-TC1)，版本 3 发行版本 2 修订版 0
- IBM TCP/IP Connectivity Utilities for AS/400 (程序 5716-TC1)，版本 3 发行版本 7 修订版 0
- IBM HTTP Server for AS/400 (程序 5769-DG1)，版本 4 发行版本 3 修订版 0

以及所有后续版本、发行版本和修订版，直至在新版本中另行声明为止。

© Copyright International Business Machines Corporation 1997, 1999. All rights reserved.

目录

前言	vii
关于 Net.Data	vii
本发行版中有哪些新功能?	vii
关于本书	viii
谁应当阅读本书	viii
关于本书中的例子	viii
第1章 介绍	1
什么是 Net.Data?	1
为什么使用 Net.Data?	2
第2章 配置 Net.Data	5
将 Net.Data 程序对象复制到 CGI-BIN 库中	5
关于 Net.Data 初始化文件	6
定制 Net.Data 初始化文件	6
创建初始化文件	7
配置变量语句	8
路径配置语句	13
环境配置语句	16
设置语言环境	18
设置 Java 应用程序语言环境	18
设置 SQL 语言环境	18
配置 Web 服务器	19
对 Net.Data 访问的对象授予访问权限	20
第3章 保障您资产的安全性	23
使用防火墙	23
在网络上加密数据	25
使用权限审批	25
使用权限	26
使用 Net.Data 机制	26
Net.Data 配置变量	26
宏开发技术	27
第4章 调用 Net.Data	31
使用宏(宏请求)调用 Net.Data	31
HTML 链	32
HTML 表	33
调用持久性宏	34
持久性宏的语法	34
例子	35
第5章 开发 Net.Data 宏	37
Net.Data 宏的剖析	38
DEFINE 块	39
FUNCTION 块	39
HTML 块	40
Net.Data 宏变量	41
标识符作用域	42
定义变量	43

引用变量	44
变量类型	46
Net.Data 函数	52
定义函数	52
调用函数	56
调用 Net.Data 内部函数	57
在宏中生成 Web 页面	61
HTML 块	61
报表块	62
宏中的条件逻辑和循环	67
条件逻辑: IF 块	67
循环结构: WHILE 块	69
第6章 使用语言环境	71
Net.Data 提供的语言环境概述	72
调用语言环境	72
处理错误条件	72
安全性	72
直接调用语言环境	73
调用程序	73
将参数传送到程序	73
从程序返回值	75
直接调用语言环境示例	76
Java 应用程序语言环境	76
调用 Java 程序	76
将参数传送到 Java 程序	77
Java 应用程序 语言环境示例	77
REXX 语言环境	77
执行 REXX 程序	78
将参数传递到 REXX 程序	78
REXX 语言环境的例子	79
SQL 语言环境	80
执行 SQL 语句	80
数据类型考虑	82
管理 Net.Data 应用程序中的事务	85
管理多数据库连接	86
存储过程	86
SQL 语言环境示例	91
System 语言环境	92
发出命令和调用程序	92
将参数传送到程序	93
System 语言环境示例	94
第7章 具有持久性宏的事务管理	95
关于持久性的宏	95
定义事务	96
启动事务	96
在事务中指定 Macro HTML 块	97
结束事务	100
定义变量在事务中的作用域	100
指定事务中的 COMMIT 和 ROLLBACK	101
持久性宏的例子	101

	第8章 改进性能	105
	Net.Data 宏的高速缓存	105
	优化语言环境	105
	REXX 语言环境	105
	SQL 语言环境	106
	System 语言环境	106
	附录A. 书目提要	107
	Net.Data 技术库	107
	相关文档	107
	附录B. Net.Data 示例宏	109
	附录C. 注意事项	113
	商标	114
	词汇表	115
	索引	117

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,包括可以通过 DRDA 访问的 DB2 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。

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 或 REXX 等程序设计语言(但不局限于这些语言)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 还支持各种各样的 Web 服务器应用程序设计接口 (API)。Net.Data 系列产品在 OS/400、OS/390、Windows NT、AIX、OS/2、HP-UX、Sun Solaris、Linux 和 Santa Cruz Operating System (SCO) 操作系统上提供了类似的功能。

本发行版中有哪些新功能?

Net.Data for OS/400 在这个发行版中提供了以下新功能:

- 用对宏和包含文件进行高速缓存的功能来改进性能
- 语言环境的增强包括:
 - 两个新的语言环境:
 - Java 应用程序
 - 直接调用
 - 在 SQL 语言环境中支持大型对象 (LOB)

- 宏语言增强包括:
 - 使用 DTW_SENDMAIL 内部函数从宏生成和发送电子邮件消息的能力
 - 使用 DTW_SETCOOKIE 和 DTW_GETCOOKIE 内部函数获取和设置 HTTP cookie 的功能
 - 使用 DTW_REPLACE 函数替换字符串的能力
 - 在 DTW_TIME 函数中支持毫秒
 - 支持动态地构建变量引用
 - 在 DTW_SELECT() 的 OPTION 元素中设置 VALUE 属性的能力
 - 在 MACRO_FUNCTION 语言结构中支持 RETURNS 关键字
 - 在变量名中支持散列字符 (#)。
- 配置的增强包括:
 - 对于 Net.Data 提供的语言环境来说, Net.Data 初始化文件中不再需要 ENVIRONMENT 语句
 - 用 DTW_SHOWSQL 配置变量来禁用 SHOWSQL 变量的能力(缺省情况下被禁用)

关于本书

本书讨论 Net.Data 的管理和编程概念, 以及如何配置 Net.Data 和它的各个组件、如何计划安全性、如何改进性能。

根据您对于编程语言和数据库的知识, 您需要学习如何使用 Net.Data 宏语言来开发宏。您要学习如何使用 Net.Data 提供的访问 DB2 数据库的语言环境, 并使用 RPG、COBOL 和其它程序设计语言来访问您的数据。

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

本书的读者是规划和编写 Net.Data 应用程序的人员。要了解本书中讨论的概念, 您应当熟悉 Web 服务器如何工作, 了解简单的 SQL 语句, 并知道 HTML 标记(包括 HTML 表格标记)。

Net.Data 参考中描述了 Net.Data 宏语言、变量、内部函数以及操作系统的区别。

关于本书中的例子

本书中出现的例子相对较为简单, 目的是演示特定的概念, 而不说明 Net.Data 构造元的所有用法。某些例子只是一些片段, 需要其它代码才能运行。

第1章 介绍

Internet 上大部分 Web 页面是静态的 Web 页面；换句话说，除非您对这些页面进行编辑，否则它们不会更改。要想将“现场”数据和应用程序放到 Web 上(例如当前的销售统计)，Web 站点的开发者通常会编写一些在 Web 服务器上作为中件执行的程序，从而动态地构建 Web 页面。编写这些类型的程序并不容易。

Net.Data 通过宏简化了交互式 Web 应用程序的编写。

本章描述 Net.Data 以及将其用于 Web 应用程序的可能的原因。

- 『什么是 Net.Data?』
- 第2页的『为什么使用 Net.Data?』

什么是 Net.Data?

通过使用 Net.Data 宏您可以执行程序设计逻辑、访问和处理变量、调用函数、使用报表生成工具。宏是一个文本文件，包含 Net.Data 宏语言结构、HTML 标记、Javascript 以及语言环境语句，例如 SQL。Net.Data 对该宏进行处理以产生可由 Web 浏览器显示的输出。宏组合了 HTML 的简单性以及 Web 服务器程序的动态功能，从而使得向静态 Web 页面中添加现场数据变得简单。现场数据可以从本地或远程的数据库以及平面文件中抽取，也可以由应用程序和系统服务生成。

图1说明了 Net.Data for OS/400、Web 服务器以及支持的数据和程序设计语言环境之间的关系。

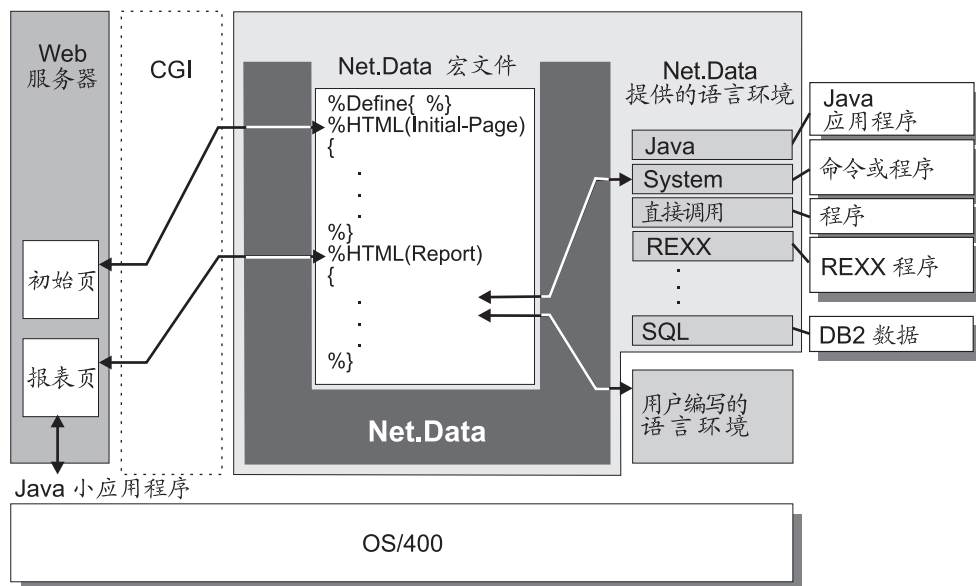


图 1. Net.Data for OS/400、Web 服务器和支持的数据以及程序源之间的关系

当 Web 服务器接收到一个请求 Net.Data 服务的 URL 时，它将 Net.Data 作为 CGI 应用程序调用。这个 URL 中含有特定于 Net.Data 的信息，包括要处理的宏。当

Net.Data 完成对请求的处理时，它将把结果的 Web 页面发送给 Web 服务器。服务器将它传递给 Web 客户，在那里它将通过浏览器显示。

为什么使用 Net.Data?

Net.Data 是创建动态 Web 页面时一个很好的选择，因为使用宏语言要比编写自己的 Web 服务器应用程序简单，并且 Net.Data 允许您使用已知的语言，例如 HTML、SQL、REXX 和 JavaScript。Net.Data 还提供了访问 DB2 数据库或使用 REXX、Perl 和其它用于您的应用程序的的语言的语言环境。另外，宏的更改结果可以立即在浏览器上看到。

Net.Data 通过为 Web 启用数据和相关的商业逻辑，对您的操作系统上已经存在的数据管理功能提供补充。更为重要地，Net.Data:

- 提供了一个简单但不失强大功能的宏语言，允许您快速开发 Internet 和 Intranet 应用程序。Net.Data Web 应用环境提供了以下功能:
- 允许 Web 应用程序中数据生成逻辑和呈现逻辑的分离。Net.Data 对于表示数据的方法(例如 HTML 或 Javascript)没有任何限制。这种分离使用户能够使用最新的呈现技术来方便地更改数据的呈现方式。
- 通过提供与 C、C++、RPG、COBOL、REXX、Java 或其它语言编写的程序实现接口的能力，允许您使用现有的技巧和商业逻辑来生成 Web 页面。
- 通过使用简单的宏语言，提供了快速开发复杂的 Internet 应用程序的能力。
- 提供对存储在 DB2 和任何远程的支持 DRDA 的数据库中的数据的高性能访问。
- 在 Net.Data 系列产品所支持的所有操作系统之间提供方便的宏移植。

解释宏语言

Net.Data 宏语言是一种解释语言。当调用 Net.Data 来处理宏时，Net.Data 将以一种顺序的方法直接解释每个语言语句，从文件的顶部开始。使用这种方法以后，如果您更改了一个宏，那么在下次指定执行该宏的 URL 时将可以立即看到您所作的任何更改。不需要重新编译。

自由格式

Net.Data 宏语言只有一些关于编程格式的规则。这种简单性为程序员提供了自由和灵活性。单条指令可以跨越多行，或者多条指令可以在一行中输入。指令可以从任何一列开始。空格或整个的空行都可以跳过。注释可以使用在任何地方。

无类型的变量

Net.Data 将所有数据都看作字符串。Net.Data 使用内部函数来对代表有效数值的字符串执行算术运算，包括那些指数格式的字符串。宏语言变量在第41页的『Net.Data 宏变量』中详细讨论。

内部函数

Net.Data 提供了对文本和数值执行各种不同的处理、搜索以及比较操作的内部函数。其它内部函数提供了格式化的功能和算术计算的能力。

错误处理

当 `Net.Data` 检测到一个错误时，带有说明的错误信息将会返回给客户。您可以在错误信息返回到用户之前在浏览器中定制它们。请参阅 *Net.Data* 参考以获取更多信息。

第2章 配置 Net.Data

Net.Data for OS/400 是作为以下产品的一个标准部分发行的:

- IBM TCP/IP Connectivity Utilities/400 V3R2、V3R7、V4R1 和 V4R2
- IBM HTTP Server for AS/400 版本 4 发行版 3 以及后继的发行版

不需要购买额外的软件; 也不需要下载和安装任何 Net.Data 软件。

您所需的 AS/400 TCP/IP 和 HTTP Server 软件是与标准 OS/400 一起提供的, 但在安装时可以作选择。对于以下版本的 OS/400 操作系统, 应在系统中安装以下可选软件:

- 对于 IBM OS/400 操作系统版本 3 发行版本 2、版本 3 发行版本 7 以及后继的版本和发行版本 (57xx-SS1):
 - IBM TCP/IP Connectivity Utilities/400 (57xx-TC1)
- 对于 IBM OS/400 操作系统版本 4 发行版本 3 以及后继的版本和发行版本 (57xx-SS1):
 - IBM HTTP Server for AS/400 (57xx-DG1)

安装 Net.Data 之后, 请完成以下章节中描述的步骤来配置 Net.Data for OS/400。这些步骤包括:

- 『将 Net.Data 程序对象复制到 CGI-BIN 库中』
- 第7页的『创建初始化文件』
- 第6页的『定制 Net.Data 初始化文件』
- 第18页的『设置语言环境』
- 第19页的『配置 Web 服务器』
- 第20页的『对 Net.Data 访问的对象授予访问权限』

将 Net.Data 程序对象复制到 CGI-BIN 库中

使用 Net.Data 之前, 必须将 Net.Data 程序对象复制到 CGI-BIN 库, 并提供对该对象的访问权限。

要复制 Net.Data 程序对象:

1. 使用“创建重复对象”(CRTDUPOBJ)命令, 将 Net.Data 程序对象 DB2WWW 从 QTCP 库复制到 CGI-BIN 库。

OS/400 V4R3 用户: 使用库 QHTTSPVR 中的程序对象; QTCP 库中的程序对象将 Net.Data 请求路由到 QHTTSPVR 库。

2. 更改 CGI-BIN 目录中的 DB2WWW 程序对象, 使得 CGI 程序所运行的用户简要表对此程序对象具有访问权。

缺省情况下, *PUBLIC 用户的 DB2WWW 程序对象权限被设置为 *EXCLUDE。为了提供对程序对象的访问权限, 可以将 *PUBLIC 用户的程序对象权限更改为 *USE, 或者特别给该用户对 DB2WWW 程序对象的概要访问。

对于不同的应用程序，您可以将 Net.Data 程序对象复制到多个库。这样就允许您有多个版本的 Net.Data 初始化文件或多个保护方案。请参阅『定制 Net.Data 初始化文件』以获取有关 Net.Data 初始化文件的更多信息；请参阅第25页的『使用权限审批』以获取有关权限的信息。

要将 **Net.Data** 程序对象复制到多个库:

1. 使用上面列出的步骤将 Net.Data 程序对象 DB2WWW 复制到一个库中。
2. 将 Net.Data 程序对象与每个库中的一个 CL 程序关联起来。
 - a. 创建一个调用位于步骤1中指定的库中的 Net.Data 程序对象的 CL 程序。
 - b. 将 CL 程序复制到每个库。

实际上，您所创建的 CL 程序成为了 Net.Data 程序对象。如果您不将程序对象与 CL 程序关联起来，并且将 Net.Data 程序对象 DB2WWW 复制到一个不同的库中，那么在使用 SQL 语言环境时，您将得到一个 -901 SQL 代码。

在以下章节中，如果您选择创建 CL 程序来调用 Net.Data，则应您所创建的 CL 程序应作为 Net.Data 程序对象对待。

关于 Net.Data 初始化文件

Net.Data 使用它的初始化文件来建立各种配置变量的设置，并配置语言环境和搜索路径。配置变量的设置值控制 Net.Data 操作的各种方面，如下：

- 指定一个用于发送电子邮件的 SMTP 服务器和字符集
- 启用 SQL 语言环境变量 SHOWSQL

语言环境语句定义了可用的 Net.Data 语言环境，并标识在语言环境之间来回流动的特殊的输入、输出参数值。语言环境允许 Net.Data 访问不同的数据源，例如 DB2 数据库和系统服务。路径语句指定了到 Net.Data 使用的文件(例如，宏和程序)的目录路径。

创建 Net.Data 初始化文件对于 Net.Data for OS/400 来说是可选的。通过使用初始化文件，您可以在 Net.Data 宏文件中使用缩短了 URL 和缩短了的对程序和宏文件的引用。但是，如果您要创建自己的语言环境，则必须有一个初始化文件。

如果没有创建初始化文件，Net.Data 在运行时将假定已经配置了一个初始化文件，其中只有受到支持的语言环境语句(请参阅第71页的『第6章 使用语言环境』来学习有关受支持的语言环境的内容)。在此情况下，所有宏、包含文件以及宏中的可执行引用都必须全是限定的。

定制 Net.Data 初始化文件

包含在初始化文件中的信息是使用三类配置语句指定的，如以下章节所述：

- 第8页的『配置变量语句』
- 第13页的『路径配置语句』
- 第16页的『环境配置语句』

请参阅第7页的『创建初始化文件』，以学习如何创建初始化文件。

第7页的图2中所示的示例初始化文件包含这些语句的例子，。

<pre> 1 DTW_SMTP_SERVER 9.5.5.78 2 MACRO_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE 3 INCLUDE_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE 4 EXEC_PATH /QSYS.LIB;/QSYS.LIB/WWW.LIB 5 ENVIRONMENT(MYLE1) /QSYS.LIB/LELIB.LIB/MYLE1.SRVPGM (IN VAR1, OUT VAR2) </pre>	<ul style="list-style-type: none"> • 第 1 行，设置配置变量的值 • 2 - 4 行，定义 Net.Data 需要访问的文件的路径 • 第 5 行指定了一个用户定义的 ENVIRONMENT 语句
---	--

图 2. Net.Data 初始化文件

每个独立配置语句的文本都必须在同一行中。（为了增加可读性，ENVIRONMENT 语句跨越了几行）。请确保对于您从宏中调用的每个用户定义的语言环境，初始化文件中都包含了一个 ENVIRONMENT 语句。如果全限定了宏中所有对文件的引用，就不需要指定任何路径配置语句了。

以下章节将描述如何在初始化文件中创建初始化文件和定制配置语句。

创建初始化文件

使用 Net.Data for OS/400 时，创建初始化文件是可选的。如果满足以下条件，就应创建初始化文件：

- 您希望设置任意 Net.Data 配置变量为非缺省值。
- 您希望为宏、包含文件以及可执行程序文件定义路径语句，以便缩短对这些文件的引用。
- 您在使用不是由 Net.Data 提供的语言环境。

要创建一个初始化文件：

1. 在 DB2WWW 程序对象所驻留的库中，使用“创建源物理文件 (CRTSRCPF)”命令来创建初始化文件。

文件名:	INI
成员名:	DB2WWW

建议您用记录长度 240 来创建初始化文件，因为配置语句的文本必须在同一行中。

2. 使用“源程序输入实用程序 (SEU)”或工作站编辑程序向文件中添加配置语句，如示例宏和以下章节中所示。

如果创建了一个初始化文件并对它进行更新，则不必结束或重新启动 Web 服务器就可以使更改生效。只在 HTTP 服务器作业初始调用 Net.Data 时，Net.Data 才读取初始化文件一次。然后保存配置文件数据，因此在后继的 Net.Data 调用中，Net.Data 不需要再读初始化文件。但是，如果初始化文件被更改，Net.Data 检测到对初始化文件的这种更改，则再次读取初始化文件。

权限技巧：请确保执行 Net.Data 的用户 ID 对这个文件具有适当的访问权。请参阅第 20 页的『对 Net.Data 访问的对象授予访问权限』，以获取更多信息。

配置变量语句

Net.Data 配置变量语句设置配置变量的值。配置变量用于各种不同的目的。有些变量是语言环境所必需的，以便使它们能够正确地工作，或者以可以替代的方式操作。其它变量控制要构造的 Web 页面的字符编码或内容。另外，您可以使用配置变量语句来定义特定于应用程序的变量。

您所使用的配置变量取决于您所使用的语言环境，以及其它特定于应用程序的因素。

要更新配置变量语句:

使用您的应用程序所需的配置变量来定制初始化文件。配置变量具有以下语法:

NAME[=]value-string

等号是可选的，由方括号指示。

以下细目描述了您可以在初始化文件中指定的配置变量语句:

- 『DTW_MACRO_CACHE_SIZE: 宏高速缓存尺寸变量』
- 第9页的『DTW_PAD_PGM_PARMS: 参数填充配置变量』
- 第9页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』
- 第10页的『DTW_SMTP_CC SID: 电子邮件 SMTP CCSID 变量』
- 第10页的『DTW_SMTP_CHARSET: 电子邮件 SMTP 字符集变量』
- 第11页的『DTW_SMTP_SERVER: 电子邮件 SMTP 服务器变量』
- 第11页的『DTW_SQL_ISOLATION: DB2 隔离变量』
- 第12页的『DTW_SQL_NAMING_MODE: SQL 表格命名变量』
- 第12页的『DTWR_CLOSE_REGISTRIES: 打开 Web 注册表变量』

DTW_MACRO_CACHE_SIZE: 宏高速缓存尺寸变量

以兆字节 (MB) 为单位指出 Net.Data 在高速缓存宏时应使用的内存大小。在超过高速缓存的大小时，Net.Data 将除去旧的被高速缓存的宏，以便为其它宏腾出地方。Net.Data 除去的是最近最少使用的宏。

语法:

DTW_MACRO_CACHE_SIZE [=] size

其中:

size 以兆字节 (MB) 的数量指定高速缓存存储器的大小。缺省值是 5 MB，通常是启用高速缓存的。如果 *size* 是 0，则不对任何宏进行高速缓存。如果 *size* 是 1 - 4，将使用缺省值 5。

例子: 指定高速缓存的大小为 16 MB。

DTW_MACRO_CACHE_SIZE 16

DTW_PAD_PGM_PARMS: 参数填充配置变量

向语言环境指出要传递给程序或存储过程的字符参数是否要用空格填充。字符参数具有数据类型 CHARACTER 或 CHAR。

对于 IN 或 INOUT 参数，如果参数值的长度小于指定的精度，则将在参数值的右侧插入空格，直至参数值的长度与精度相同。

对于 OUT 参数，参数值被设置为具有精确的空格。

在对程序或存储过程的调用之后，将从 OUT 和 INOUT 参数值中除去所有尾随的空格。

在 Net.Data 初始化文件中设置此变量，以便为所有的宏指定一个值。可以通过宏中的定义覆盖值。如果宏中没有定义 DTW_PAD_PGM_PARMS，它将使用初始化文件中的值。

“直接调用”和 SQL 语言环境支持 DTW_PAD_PGM_PARMS。

语法:

DTW_PAD_PGM_PARMS [=] YES|NO

其中:

YES 指定所有的 IN 和 INOUT 字符参数值都将留待调整，并在将参数传递到程序或存储过程之前根据定义的参数精度填充空格。在对程序或存储过程的调用之后，将除去尾随空格。

NO 指定在将参数传递到程序或存储过程时不对字符参数值(以 NULL 结束的值)添加补空。在调用程序或存储过程之后不除去尾随空格。这是缺省值。

DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量

覆盖 Net.Data 宏中 SHOWSQL 设置的效果。

语法:

DTW_SHOWSQL YES|NO

其中:

YES 在所有将 SHOWSQL 的值设置为 YES 的宏中启用 SHOWSQL。

NO 在宏中禁用 SHOWSQL，即使变量 SHOWSQL 被设置为 YES。NO 是缺省值。

表1描述 Net.Data 初始化文件和宏中的设置如何确定对于特定的宏是否要启用或禁用 SHOWSQL 变量。

表 1. Net.Data 初始化文件和宏中对 SHOWSQL 的设置之间的关系

DTW_SHOWSQL 的设置	设置 SHOWSQL	显示 SQL 语句
NO	NO	NO
NO	YES	NO
YES	NO	NO
YES	YES	YES

DTW SMTP CCSID: 电子邮件 SMTP CCSID 变量

指定 ASCII 编码字符集标识符 (CCSID)，它与 DTW SMTP CHARSET 中指定的多用途网际邮件扩展 (MIME) 字符串相关联。在把 DTW SENDMAIL 函数上指定的数据从 EBCDIC 转换为 ASCII 时，将使用 CCSID。

如果指定了 DTW SMTP CCSID，也必须指定 DTW SMTP CHARSET。指定 CCSID 时，请确保它对于 DTW SMTP CHARSET 中指定的 MIME 字符集是适当的，并且系统支持该 CCSID。表2列出了公用 MIME 字符集以及相关联的 ASCII CCSID。如果没有设置 DTW SMTP CCSID，Net.Data 将使用与 MIME 字符集 ISO-8859-1 相关联的 CCSID 819。

语法:

DTW SMTP CCSID [=] *ascii_ccsid*

其中 *ascii_ccsid* 是在从 EBCDIC 转换为 ASCII 时要使用的 ASCII CCSID (一个范围在 1-65534 内的数字)。

例子:

DTW SMTP CCSID 912

这个 ASCII CCSID 对应于 MIME 字符集 ISO-8859-2

DTW SMTP CHARSET: 电子邮件 SMTP 字符集变量

指定多用途网际邮件扩展 (MIME) 字符集将由 DTW SENDMAIL 函数在电子邮件消息中使用。如果指定了 DTW SMTP CHARSET，则必须指定 DTW SMTP CCSID。在指定 MIME 字符集的情况下，请确保该字符集有效，因为 Net.Data 不会对为此变量指定的值进行确认。如果没有设置 DTW SMTP CHARSET，Net.Data 将使用 MIME 字符集 ISO-8859-1，相关联的 CCSID 是 819。

表2列出了公用 MIME 字符集以及相关联的 ASCII CCSID。

表 2. Net.Data 支持的字符集

MIME 标准字符集	ASCII CCSID	说明
US-ASCII	367	美国英语
ISO-2022-JP	5052	日本 MBCS
ISO-8859-1	819	拉丁-1
ISO-8859-2	912	拉丁-2
ISO-8859-5	915	西里尔语
ISO-8859-6	1089	阿拉伯语
ISO-8859-7	813	希腊语
ISO-8859-8	916	希伯来语
ISO-8859-9	920	拉丁-5

语法:

DTW SMTP CHARSET *character_set*

其中，*character_set* 是要使用的 MIME 字符集。

例子:

```
DTW_SMTP_CHARSET iso-8859-2
```

这个 MIME 字符集对应 912 ASCII CCSID。

DTW_SMTP_SERVER: 电子邮件 SMTP 服务器变量

指定使用 DTW_SENDMAIL 内部函数用于发送电子邮件消息的 SMTP 服务器。这个变量的值可以是一个主机名，或是一个 IP 地址。如果没有设置这个变量，则 Net.Data 把本地主机用作 SMTP 服务器。

语法:

```
DTW_SMTP_SERVER server_name
```

其中 *server_name* 是要用于发送电子邮件消息的 SMTP 服务器的主机名或 IP 地址。

性能技巧: 对此值指定一个 IP 地址以防止 Net.Data 在检索指定的 SMTP 服务器的 IP 地址时连接到一个域名服务器。

例子:

```
DTW_SMTP_SERVER 9.5.34.5
```

DTW_SQL_ISOLATION: DB2 隔离变量

DTW_SQL 语言环境使用 DTW_SQL_ISOLATION 配置语句来确定 DTW_SQL 语言环境所执行的数据库操作和正在并发执行的进程之间的隔离程度是多少。

语法:

```
DTW_SQL_ISOLATION locking_method
```

其中 *locking_method* 是以下的一个值:

DTW_SQL_NO_COMMIT

指定不使用确认控制。对于 OS/400 操作系统，如果关系数据库是在关系数据库目录中指定的并且该关系数据库在一个非 OS/400 系统上的，那么不要指定这个值。

DTW_SQL_READ_UNCOMMITTED

对于 SQL ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON 和 REVOKE 语句中引用的对象指定锁定，并且对于已更新、删除和插入的行指定锁定。这些对象一直锁定到工作单元(事务)结束为止。 可以看到其它进程中未提交的更改。

DTW_SQL_READ_COMMITTED

对于 SQL ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON 和 REVOKE 语句中引用的对象指定锁定，并且对于已更新、删除和插入的行指定锁定。这些对象一直锁定到工作单元(事务)结束为止。 选定但未更新的行是锁定的，直到选择下一行为止。其它进程中未提交的更改是无法看到的。

DTW_SQL_REPEATABLE_READ

对于 SQL ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON

和 REVOKE 语句中引用的对象指定锁定，并且对于已选定、更新、删除和插入的行指定锁定。这些对象一直锁定到工作单元(事务)结束为止。其它进程中未提交的更改是无法看到的。

DTW_SQL_SERIALIZABLE

对于 SQL ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON 和 REVOKE 语句中引用的对象指定锁定，并且对于已选定、更新、删除和插入的行指定锁定。这些对象一直锁定到工作单元(事务)结束为止。其它进程中未提交的更改是无法看到的。所有在 SELECT、UPDATE、DELETE 和 INSERT 语句中引用的表格是锁定的，直到工作单元(事务)结束为止。

DTW_SQL_NAMING_MODE: SQL 表格命名变量

DTW_SQL_NAMING_MODE 配置语句指定了如何在 SQL 语句中指定一个表格的名称。

语法:

DTW_SQL_NAMING_MODE *mode*

其中 *mode* 是以下的一个值:

SQL_NAMING

指定集合名称以如下形式限定表格:

collection.table

其中 *collection* 是集合的名称，*table* 是表格的名称。缺省的限定符是运行那个执行 SQL 语句的进程的用户 ID，在没有明确限定表格名称且没有指定缺省集合名称的情况下使用这个限定符。SQL_NAMING 是缺省的表格名。

SYSTEM_NAMING

指定库名以如下形式限定文件:

library/file

其中 *library* 是库名，*file* 是表格名称。如果表格名(文件)没有明确受限定并且没有指定缺省的集合名称名称(库)，则缺省的搜索路径是用于非限定表格名称的库列表 (*LIBL)。

DTWR_CLOSE_REGISTRIES: 打开 Web 注册表变量

指定是关闭 Web 注册表还是保持打开状态。这个变量允许您保持 Web 注册表打开的状态，这样，访问相同注册表的后继 Net.Data 宏调用就不必重新打开注册表了。

语法:

DTWR_CLOSE_REGISTRIES YES|NO

其中:

YES 指定在处理完一个 Net.Data 宏之后关闭所有打开的 Web 注册表。

NO 指定在处理完一个 Net.Data 宏之后使所有打开的 Web 注册表保持打开状态。NO 是缺省值。

性能技巧: 您可以使用 DTWR_CLOSE_REGISTRIES 配置语句来改进访问 Web 注册表(使用 Web 注册表内部)时的性能，这可以通过最少化打开和关闭注册表的次数来实现。如果多个进程可同时访问注册表(如同有同时的浏览器请求的情况一样)，可将

DTWR_CLOSE_REGISTRIES 设置为 YES。

路径配置语句

Net.Data 从路径配置语句的设置中确定 Net.Data 宏所使用的文件和可执行程序的位置。路径语句有:

- 『MACRO_PATH』
- 第14页的『EXEC_PATH』
- 第15页的『INCLUDE_PATH』
- 第15页的『FFI_PATH』
- 第16页的『HTML_PATH』
- 第16页的『DTW_JAVA_CLASSPATH』

这些路径语句标识了一个或多个 Net.Data 在试图找到宏、文本文件 和包含文件时搜索的目录。您所需的路径语句取决于宏所使用的 Net.Data 的功能。

更新准则:

有些一般准则适用于路径语句。每个路径语句的说明中都注明了例外情况。

- 每个指定的目录都以一个分号 (;) 结束。
- 斜杠 (/) 和反斜杠 (\) 是同等对待的。
- 每个路径语句都可以指定多路径。路径是根据指定的顺序从左向右搜索的。这个多路径的功能可以让您在多个目录中组织文件。例如, 您可以把每个 Web 应用程序放在它们自己的目录中。
- 建议您使用绝对路径语句。

以下章节将描述每个路径语句的目的和语法, 并提供有效路径语句的示例。

MACRO_PATH

MACRO_PATH 配置语句标识了 Net.Data 搜索 Net.Data 宏的目录。例如, 指定以下 URL 将请求带有路径和文件名 /macro/sqlm.d2w:

```
http://server/cgi-bin/db2www/macro/sqlm.d2w/report
```

语法:

```
MACRO_PATH [=] path1;path2;...;pathn
```

等号 (=) 是可选的, 由方括号指出。

Net.Data 在 MACRO_PATH 配置语句中将路径 /macro/sqlm.d2w 附加到路径后面, 从左至右, 直至 Net.Data 找到宏或搜索完所有路径。请参阅第31页的『第4章 调用 Net.Data』以获取有关调用 Net.Data 宏的信息。

例子: 以下例子显示了初始化文件中的 MACRO PATH 语句以及调用 Net.Data 的相关链。

Net.Data 初始化文件:

```
MACRO_PATH /u/user1/macros;/usr/lpp/netdata/macros;
```


HTML 链:

```
<A HREF="http://server/cgi-bin/db2www/query.d2w/input">Submit another query.</A>
```

如果在目录 `/u/user1/macros` 中找到文件 `query.d2w`, 那么全限定路径就是 `/u/user1/macros/query.d2w`。

如果在 `MACRO_PATH` 语句指定的目录中没有找到文件, `Net.Data` 将在根 (`/`) 目录中搜索该文件。例如, 如果提交了以下 URL:

```
http://myserver/cgi-bin/db2www/myfile.txt/report
```

并且在 `MACRO_PATH` 指定的所有目录中都没有找到文件 `myfile.txt`, 那么 `Net.Data` 将试图在根 (`/`) 目录中查找文件:

```
/myfile.txt
```

EXEC_PATH

`EXEC_PATH` 配置语句标识了一个或多个目录, `Net.Data` 在其中搜索 `EXEC` 语句调用的外部程序或可执行变量。如果找到程序, 则将外部程序名附加到路径说明后, 形成一个传送到语言环境执行的全限定文件名。

语法:

```
EXEC_PATH [=] path1;path2;...;pathn
```

例子: 以下例子显示了初始化文件中的 `EXEC_PATH` 语句以及调用外部程序的宏中的 `EXEC` 语句。

`Net.Data` 初始化文件:

```
EXEC_PATH /qsys.lib/programs.lib;/qsys.lib/rexx.lib/rexxpgms.file;
```

`Net.Data` 宏:

```
%FUNCTION(DTW_REXX) myFunction() {  
  %EXEC{ myFunction.mbr %}  
%}
```

如果在 `/qsys.lib/rexx.lib/rexxpgms.file` 目录中找到文件 `myFunction.mbr`, 则程序的限定名为 `/qsys.lib/rexx.lib/rexxpgms.file/myFunction.mbr`。

如果在 `EXEC_PATH` 语句指定的目录中没有找到文件:

- 如果指定的路径是绝对路径, 那么 `Net.Data` 将在指定的路径中搜索该文件。例如, 如果指定了以下 `EXEC` 语句:

```
%EXEC{/qsys.lib/programs.lib/rpg1.pgm %}
```

`Net.Data` 将在 `/qsys.lib/programs.lib` 目录中搜索文件 `rpg1.pgm`。

- 如果指定的路径是绝对路径, 那么 `Net.Data` 将搜索当前工作目录。例如, 如果指定了以下 `EXEC` 语句:

```
%EXEC { rpg1.pgm %}
```

那么 `Net.Data` 将试图在当前工作目录中查找文件 `rpg1.pgm`。

INCLUDE_PATH

INCLUDE_PATH 配置语句标识了一个或多个 Net.Data 搜索的目录从而找到一个 Net.Data 宏中的 INCLUDE 语句所指定的文件。在找到这个文件之后，Net.Data 将把包含文件的名称附加到路径说明后面，以便产生限定的包含文件名。

语法:

```
INCLUDE_PATH [=] path1;path2;...;pathn
```

例 1: 以下例子显示了初始化文件中的 INCLUDE_PATH 语句和指定包含文件的 INCLUDE 语句。

Net.Data 初始化文件:

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data 宏:

```
%INCLUDE "myInclude.txt"
```

如果在 /u/user1/includes 目录中找到文件 *myInclude.txt*，则包含文件的全限定名称是 /u/user1/includes/myInclude.txt。

例 2: 以下例子显示了 INCLUDE_PATH 语句和带有子目录名称的 INCLUDE 文件。

Net.Data 初始化文件:

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data 宏:

```
%INCLUDE "OE/oeheader.inc"
```

包含文件是在目录 /u/user1/includes/OE 和 /usr/lpp/netdata/includes/OE 中搜索的。如果文件在 /usr/lpp/netdata/includes/OE 中找到，则包含文件的全限定名称就是 /usr/lpp/netdata/includes/OE/oeheader.inc。

如果在 INCLUDE_PATH 语句指定的目录中没有找到文件:

- 如果指定的路径是绝对路径，那么 Net.Data 将在指定的路径中搜索该文件。例如，如果指定了以下 INCLUDE 语句:

```
%INCLUDE "/u/user1/includes/oeheader.inc"
```

那么 Net.Data 将在 /u/user1/includes 目录中搜索文件 *oeheader.inc*。

- 如果指定的路径是绝对路径，那么 Net.Data 将搜索当前工作目录。例如，如果指定了以下 INCLUDE 语句:

```
%INCLUDE "oeheader.inc"
```

那么 Net.Data 将试图在当前工作目录中查找文件 *oeheader.inc*。

FFI_PATH

FFI_PATH 配置语句标识了一个或多个 Net.Data 搜索的目录从中搜索一个平面文件接口 (FFI) 函数引用的平面文件。

语法:

```
FFI_PATH [=] path1;path2;...;pathn
```

例子: 以下例子显示了初始化文件中的 FFI_PATH 语句。

Net.Data 初始化文件:

```
FFI_PATH /u/user1/ffi;/usr/lpp/netdata/ffi;
```

当调用 FFI 语言环境时, Net.Data 将查看 FFI_PATH 语句中指定的路径。

因为 FFI_PATH 语句用于为那些不在路径语句所含目录中的文件提供安全性, 因此对于没有找到 FFI 文件提供了特殊措施。请参阅 *Net.Data* 参考中有关 FFI 内部函数一节。

HTML_PATH

HTML_PATH 配置语句指定了 Net.Data 将大型对象 (LOB) 写入哪个目录。此路径语句只接受一个目录路径。

HTML_PATH 必须指定一个不在 QSYS.LIB 文件系统上的 IFS 目录。

语法:

```
HTML_PATH [=] path
```

例子: 以下例子显示了初始化文件中的 HTML_PATH 语句。

Net.Data 初始化文件:

```
HTML_PATH /db2/lobs
```

当查询返回一个 LOB 时, Net.Data 将把它保存在 HTML_PATH 配置语句指定的目录中。

性能技巧: 在使用 LOB 时请考虑系统限度, 因为它们会很快地消耗资源。请参阅第82页的『使用大型对象』以获取更多信息。

DTW_JAVA_CLASSPATH

DTW_JAVA_CLASSPATH 配置语句指定了用于定位 Java 类的路径。目录由冒号分开。

语法:

```
DTW_JAVA_CLASSPATH [=] path
```

例子: 以下示例显示了初始化文件中的 DTW_JAVA_CLASSPATH 语句。

Net.Data 初始化文件:

```
DTW_JAVA_CLASSPATH /directory1/directory2:/QIBM/ProdData/Java400
```

环境配置语句

ENVIRONMENT 语句配置一个语言环境。语言环境是 Net.Data 的一个组件, Net.Data 用它来访问诸如 DB2 数据库等数据源, 或者执行诸如 REXX 等语言编写的程序。

Net.Data 提供了一系列语言环境，还提供了允许您创建自己的语言环境的接口。第71页的『第6章 使用语言环境』中描述了这些语言环境，*Net.Data 语言环境接口参考*中描述了语言环境接口。

在调用某个特定的语言环境之前，Net.Data 要求用于该语言环境的 ENVIRONMENT 语句必须存在。

对于与 Net.Data 一起发行的语言环境，Net.Data for OS/400 不需要 ENVIRONMENT 语句。当然，如果遇到一个语言环境语句，它将覆盖 Net.Data 使用的缺省值。建议在 Net.Data 配置文件中不要添加用于 Net.Data 提供的语言环境的 ENVIRONMENT 语句。

可以通过将变量指定为 ENVIRONMENT 语句中的参数来将变量与语言环境相关联。Net.Data 将 ENVIRONMENT 语句中指定的参数作为宏变量隐式地传递到语言环境。要更改宏中 ENVIRONMENT 语句内指定的参数值，可以使用 DTW_ASSIGN() 函数为该变量赋一个值，也可以在 DEFINE 部分定义该变量。**重要事项：**如果宏中定义了一个宏变量，但 ENVIRONMENT 语句中没有加以指定，则该宏变量不会被传递到语言环境。

例如，宏可以定义一个 DATABASE 变量来指定一个数据库的名称，DTW_SQL 函数中的 SQL 语句将在此执行。DATABASE 的值必须传递到 SQL 语言环境 (DTW_SQL)，这样，SQL 语言环境就可以连接到指定的数据库。要将变量传递到语言环境，您必须向 DTW_SQL 的语言环境的参数列表中添加 DATABASE 变量。

示例 Net.Data 初始化文件对定制 Net.Data 语言环境配置语句的设置做了几个假设。这些假设对于您的环境来说可能是不正确的。请针对您的环境适当地修改这些语句。

要添加或更新 **ENVIRONMENT** 语句:

ENVIRONMENT 语句具有以下语法:

```
ENVIRONMENT(type) library_name (parameter_list, ...)
```

参数:

- *type*

Net.Data 将此语言环境和 Net.Data 宏中定义的 FUNCTION 块相关联的名称。您必须在 FUNCTION 块定义中指定语言环境的类型，从而标识 Net.Data 要执行函数所应使用的语言环境。

- *library_name*

服务程序的名称中包含 Net.Data 调用的语言环境接口。

服务程序名是用扩展名 *.SRVPGM* 指定的。

- *parameter_list*

在每个函数调用中传递给语言环境的参数列表(除了在 FUNCTION 块定义中指定的参数)。

要在参数列表中设置和传递变量，可以在宏中定义变量。

在执行将由语言环境处理的函数之前，您必须将这些参数定义为配置变量或宏中的变量。如果一个函数修改了它的输出参数，那么在函数完成之后这些参数仍将保留修改后的值。

Net.Data 处理初始化文件时，它不会装入语言环境服务程序。Net.Data 在它首次执行标识某个语言环境的函数时装入该语言环境的服务程序。然后，只要 Net.Data 是装入的，服务程序将保持装入状态。

例子: 用于 Net.Data 提供的语言环境的 ENVIRONMENT 语句

在为您的应用程序定制 ENVIRONMENT 语句时, 请在 ENVIRONMENT 语句中添加需要从初始化文件传递到语言环境的变量或 Net.Data 宏编写者需要在他们的宏中设置或覆盖的变量。

在 OS/400 上, Net.Data 语言环境不需要 ENVIRONMENT 语句, 不建议使用。当然, 这个示例显示了 Net.Data 使用的一些缺省的 ENVIRONMENT 语句。

```
1  MACRO_PATH      /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
2  INCLUDE_PATH    /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
3  EXEC_PATH       /QSYS.LIB;/QSYS.LIB/WWW.LIB

4  ENVIRONMENT(DTW_REXX) /QSYS.LIB//QTCP.LIB/QTMHREXX.SRVPGM ( )
5  ENVIRONMENT(DTW_SQL) /QSYS.LIB/QTCP.LIB/QTMSQL.SRVPGM (IN DATABASE,
    LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL, DB_CASE,
    RPT_MAX_ROWS, START_ROW_NUM, DTW_SET_TOTAL_ROWS,
    OUT DTWTABLE, SQL_CODE, TOTAL_ROWS)
6  ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ( )
```

必需: 每个 ENVIRONMENT 语句必须在单独一行上。

设置语言环境

在对 Net.Data 语言环境修改配置变量和 ENVIRONMENT 配置语句之后, 要想使以下语言环境能够正确工作, 则还需要一些额外的设置。以下章节描述设置语言环境所必需的步骤:

- 『设置 Java 应用程序语言环境』
- 『设置 SQL 语言环境』

设置 Java 应用程序语言环境

使用 Java 应用程序语言环境之前, 先完成以下步骤(最早在 OS/400 版本 4 发行版 4 中介绍):

1. 安装 『AS/400 Developer Kit for Java』 特许程序, 产品标识符 5769JV1。要在 AS/400 上运行 Java 应用程序, 就必须安装 『AS/400 Developer Kit for Java』。
2. 在 Net.Data 初始化文件中设置 DTW_JAVA_CLASSPATH 路径配置变量, 这样可以使 Java 找到 Java 应用程序类。有关此路径配置语句的更多信息, 请参阅第16页的 『DTW_JAVA_CLASSPATH』。

设置 Java 应用程序语言环境之后, 请参阅第76页的 『Java 应用程序语言环境』以了解如何使用 Java 应用程序语言环境。

设置 SQL 语言环境

使用 SQL 语言环境之前, 请先完成以下步骤:

1. 除了 SQL 语言环境需要访问的任何远程数据库以外, 再对关系数据库目录中的局部数据库创建一个目录项(具有远程位置 *LOCAL 的目录项)。
可以使用 “添加关系数据库目录项 (ADDRDBDIRE)” 命令来添加这个项。

如果访问远程数据库，请完成附加的配置步骤，例如设置本地系统与远程系统之间的通信。有关分布式数据库支持的更多信息，请参阅*OS/400 Distributed Database Programming*。

2. 如果使用 DataLink，请确保在所有使用的系统上都已配置了 TCP/IP，并且在所有将包含要链接的对象的系统上都已经启动并配置了 DataLink “文件管理器”。有关 DataLink 的更多信息，请参阅*DB2 for OS/400 SQL Programming*
3. 如果 SQL 语言环境将返回大型对象 (LOB)，请设置 HTML_PATH 配置变量。要进一步了解此配置变量，请参阅第16页的『HTML_PATH』。
4. 添加或更新配置变量。SQL 语言环境支持以下配置变量，它们可以在 Net.Data 初始化文件中指定：

DTW_SQL_ISOLATION

确定 SQL 语言环境所执行的数据库操作和正在并发执行的进程之间的隔离程度是多少

DTW_SQL_NAMING_MODE

确定如何在 SQL 语句中指定表格名

DTW_SHOWSQL

允许使用宏变量 SHOWSQL

要进一步了解 Net.Data 配置变量语句，请参阅第8页的『配置变量语句』。

设置 SQL 语言环境之后，请参阅第80页的『SQL 语言环境』以了解如何使用 SQL 语言环境。

配置 Web 服务器

公共网关接口 (CGI) 是一个允许 Web 服务器调用应用程序(如 Net.Data) 的工业标准接口。Net.Data 对 CGI 的支持使您可以将 Net.Data 和您所喜爱的 Web 服务器一起使用。

配置 Web 服务器来调用 Net.Data，这可以通过在 HTTP 配置文件中添加 Map、Exec 和 Pass 伪指令来实现对 Net.Data 的调用。

例如，假定 Net.Data 程序对象驻留在库 CGI 中，那么以下伪指令将把 Net.Data 重定向到 /QSYS.LIB/CGI.LIB/DB2WWW.PGM:

```
Map /cgi-bin/db2www/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Map /CGI-BIN/DB2WWW/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Exec /QSYS.LIB/CGI.LIB/*
```

建议：在 HTTP 配置文件中按以下顺序组织伪指令，以防止伪指令被忽略：Map、Exec、Pass。例如，如果以下 Pass 伪指令先于 Map 或 Exec 伪指令，则 Map 和 Exec 伪指令被忽略：

```
Pass /*
```

Map 伪指令

Map 伪指令将格式为 /cgi-bin/db2www/* 的条目映射到系统中 Net.Data 程序

所驻留的库中。(字符串尾部的星号(*)指跟在字符串后面的所有信息。)其中包括了大写和小写两种映射语句,因为伪指令是区分大小写的。在这个例子中,两条 Map 语句指向同一位置。

Exec 伪指令

Exec 伪指令启用 Web 服务器执行 CGI 库中的任何 CGI 程序。在伪指令中指定了程序所驻留(不是程序本身)的库。

Pass 伪指令

如果希望在 SQL 语言环境中使用大型对象(LOB),可以创建一个 Pass 伪指令来指定 SQL 语言环境存储 LOB 文件的目录。例如:

```
Pass /tmplobs/* /html_path/*
```

其中, *html_path* 是 HTML_PATH 配置变量中指定的目录名称,指定存储 LOB 的缺省目录。请参阅第16页的『HTML_PATH』,以获取更多信息。

Net.Data 中不使用 Pass 伪指令。如果希望简化 URL,则可以在 Net.Data 初始化文件中使用 MACRO_PATH 语句,这在第13页的『MACRO_PATH』中讨论。

对 Net.Data 访问的对象授予访问权限

使用 Net.Data 之前,需要确保执行 Net.Data 的用户 ID 对于 Net.Data 宏中引用的对象以及 URL 引用的宏具有适当的访问权。

尤为特别的,请确保执行 Net.Data 的用户 ID 具有以下权限:

- 要读取 Net.Data 初始化文件 INI.FILE/DB2WWW.MBR
- 要执行 Net.Data 可执行文件和服务程序,并要在可执行文件和服务程序的路径中搜索目录(库)
- 要读取适当的 Net.Data 宏文件并搜索 MACRO_PATH 路径配置语句中标识的适当的目录
- 要执行适当的文件并搜索 EXEC_PATH 路径配置语句中标识的适当的目录
- 要读取适当的文件并搜索 INCLUDE_PATH 路径配置语句中标识的适当的目录
- 要读取和写入适当的文件并搜索 FFI_PATH 路径配置语句中标识的适当的目录
- 要访问可能被语言环境语句的目标所引用的任何对象。例如, SQL 语言环境运行 SQL 语句,而 SQL 语句访问数据库文件,因此运行 Net.Data 的用户 ID 必须对数据库文件具有权限。

例子:

根据您选择用于存储 Net.Data 宏的文件系统的不同,需要把访问 Net.Data 宏所需的权限授予 Net.Data CGI 程序所运行的用户简要表。以下方法给了 QTMHHTTP1 用户简要表权限(在 V3R2 和 V3R7 中,Internet Connection for AS/400 只能在 QTMHHTTP1 用户简要表中运行 CGI 程序.):

- 在根文件系统中,使用“更改权限”(CHGAUT) CL 命令为用户简要表授权:

```
CHGAUT OBJ('/WWW') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTTP1) DTAAUT(*RX)
```

您需要为路径中所有对象授权。

- 在库文件系统 (QSYS.LIB) 中, 使用 “授予对象权限” (GRTOBJAUT) CL 命令为用户简要表授权:

```
GRTOBJAUT OBJ(WWW) OBJTYPE(*LIB) USER(QTMHHTP1) AUT(*USE)
GRTOBJAUT OBJ(WWW/MACRO) OBJTYPE(*FILE) USER(QTMHHTP1) AUT(*USE)
```

您只需要为库和源物理文件授权。

同样可以使用 CHGAUT CL 命令为 QSYS.LIB 文件系统中的对象授权, 如下所示:

```
CHGAUT OBJ('/QSYS.LIB/WWW.LIB') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/QSYS.LIB/WWW.LIB/MACRO.FILE') USER(QTMHHTP1) DTAAUT(*RX)
```

有关特定于语言环境的权限考虑, 请参阅第71页的『第6章 使用语言环境』节中每种语言环境的说明。

第3章 保障您资产的安全性

Internet 安全性是通过防火墙技术、操作系统功能、Web 服务器功能、Net.Data 机制和作为数据源一部分的访问控制机制的组合提供的。

您必须确定对于您的资产哪一级安全性是适当的。本章将描述可用于保障您资产安全性的方法，并提供对用于计划 Web 站点安全性的附加资源的参考。

以下章节包含了保护资产的准则。描述的安全性机制包括：

- 『使用防火墙』
- 第25页的『在网络上加密数据』
- 第25页的『使用权限审批』
- 第26页的『使用权限』
- 第26页的『使用 Net.Data 机制』

使用防火墙

防火墙是一些硬件、软件和策略的集合，是为在网络环境内限制对资源的访问而设计的。

防火墙：

- 保护内部网络不受侵入或窃密
- 保护内部网络不受内部用户带入的数据和程序的侵害
- 限制内部用户对外部数据的访问
- 在防火墙遭到破坏时限制可能造成的损坏

Net.Data 可以和在您的环境中执行的防火墙产品一起使用。

以下可能的配置为管理 Net.Data 应用程序的安全性提供了建议。这些配置提供了一些高级信息，并假定您已经配置了一个将您的安全 intranet 与公用 Internet 隔开的防火墙。请仔细考虑这些配置以及您所在组织的安全策略：

- **高安全性配置**

此配置创建了一个将 Net.Data 和 Web 服务器与安全 intranet 以及公用 Internet 隔开的子网。防火墙软件用于在 Web 服务器和公用 Internet 之间创建一个防火墙，并在 Web 服务器和安全 intranet (其中包含 DB2 服务器)之间创建另一个防火墙。这一配置由第24页的图3显示。

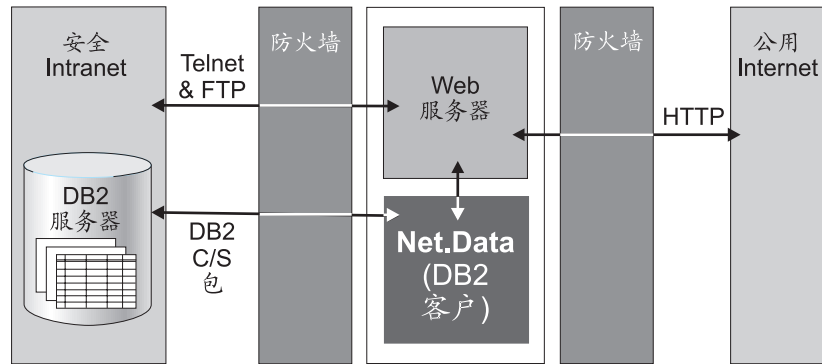


图 3. 高安全性配置

要设置此配置:

- 在 Web 服务器设备上安装 Net.Data, 并确保 Net.Data 可以通过以下途径在 intranet 内部访问 DB2 服务器配置防火墙以允许 DB2 通信量穿过防火墙。一个方法是添加信息包过滤规则, 从而允许来自 Net.Data 的 DB2 客户请求和从 DB2 服务器到 Net.Data 的应答信息包。
- 允许在 Web 服务器和安全 intranet 之间的 FTP 和 Telnet 访问。一个方法是在 Web 服务器设备上安装一个 socks 服务器。
- 在防火墙软件的信息包过滤配置文件中, 指定从标准 HTTP 端口进入的 TCP 信息包可以访问 Web 服务器。同样, 指定出去的 TCP 应答信息包可以从 Web 服务器进入公用 Internet 上的任何主机。

• 中等安全性配置

在此配置中, 防火墙软件将安全 intranet 和 DB2 服务器与公用 Internet 隔开。Net.Data 和 Web 服务器位于防火墙外部的 workstation 平台上。这种配置要比第一种简单一些, 但仍提供了数据库保护机制。图4显示了这一配置。

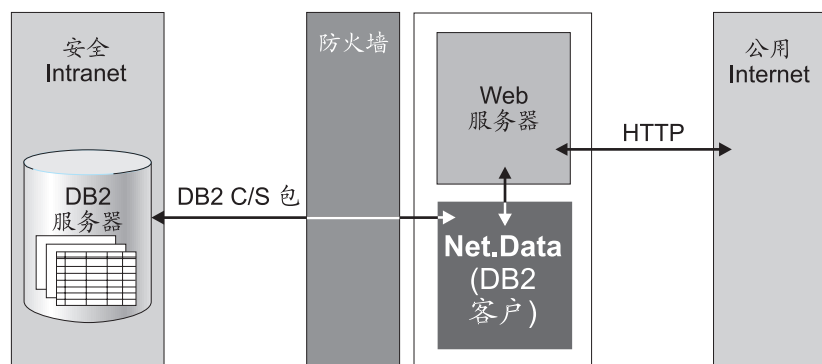
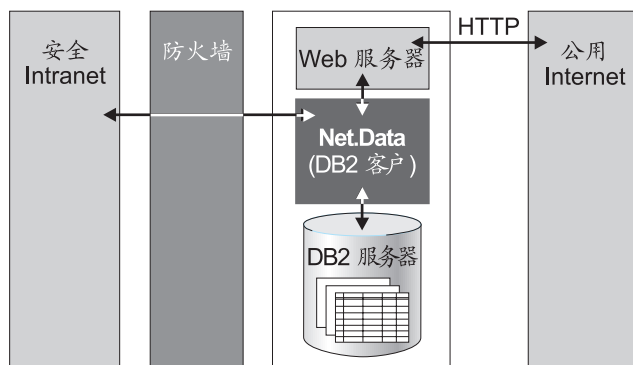


图 4. 中等安全性配置:

防火墙的配置必须能让 DB2 客户请求从 Net.Data 流到 DB2, 并能让应答信息包从 DB2 流到 Net.Data。

• 低安全性配置

在此配置中，DB2 服务器和 Net.Data 安装在防火墙和安全 intranet 的外部。在遇到外部攻击时，它们是不受保护的。对于这种配置，防火墙不需要任何信息包过滤规则。图5显示了这一配置。



在网络上加密数据

使用支持安全套接字层 (SSL) 的 Web 服务器时，您可以加密所有在客户系统和 Web 服务器之间发送的数据。这一安全度量支持对注册 ID、口令、通过 HTML 表从客户系统传输到 Web 服务器的所有数据以及从 Web 服务器发送到客户系统的所有数据进行加密。

使用权限审批

权限审批用于确保进行 Net.Data 请求的用户 ID 已被授权访问和更新应用程序内的数据。权限审批是一个匹配用户 ID 和口令的过程，用以确认请求来自有效的用户 ID。Web 服务器将用户 ID 和它所处理的每个 Net.Data 请求关联起来。然后，处理请求的进程或线程就可以访问该用户 ID 已授权的资源。

在一个 OS/400 环境中，用户 ID 能够以下列三种方式之一与处理 Net.Data 请求的线程或进程相关联：

基于客户的权限审批

在客户端提示用户输入本地 OS/400 用户 ID 和口令。然后 Web 服务器认证该用户。如果成功认证，则所提供的用户 ID 就与该请求相关联。特殊 Web 服务器 %%CLIENT%% 访问控制用户 ID 的使用，使得这种权限审批成为可能。

IBM 自 OS/400 V4R1 开始的 HTTP 服务器都支持基于客户的权限审批。

基于服务器的权限审批

Web 服务器的用户 ID 与每个请求相关联，不提示用户输入其用户标识符和口令。特殊 Web 服务器 %%SERVER%% 访问控制用户 ID 的使用，使得这种权限审批成为可能。

缺省地，IBM 的 HTTP 服务器使用 QTMHHTTP1 用户 ID (用户简要表) 来运行 CGI 程序。但是，如果 UserID 伪指令在起作用或者它在一个已指定 UserID 子伪指令的保护设置中，则使用指定的用户 ID 来执行该程序。

代理者权限审批

有权访问一些预定义资源集合的代理者用户 ID 是与客户请求相关联的。这种权限审批需要创建具有适合于一组用户或一类请求的访问权限的代理者用户 ID。代理者用户 ID 的权限审批通常使用验证列表对象(在 V4R1 中首先引入)。有关这方面的更多信息与例子, 请参阅 *OS/400 系统 API 参考*。

在配置 Web 服务器时指定 Web 服务器用来关联用户 ID 和客户请求的方法。有关访问控制用户 ID、安装 Web 服务器和使用 Protect、Protection、DefProt 和 UserId 伪指令来配置 Web 服务器的更多信息, 请参阅 HTTP 服务器文档。

技巧: 要保护 Net.Data 宏, 请执行以下操作:

1. 在 Net.Data 程序对象的 Web 服务器配置文件中添加保护命令。
2. 请确保将要运行 Net.Data 的用户 ID 对宏文件具有访问权限。有关授予访问权限的更多信息, 请参阅第20页的『对 Net.Data 访问的对象授予访问权限』。

使用权限

权限向用户提供对于一个对象、资源或函数的完整的或限定的访问权限。诸如 DB2 等数据源提供自己的权限机制来保护它们所管理的信息。这些权限机制假定与正在执行 Net.Data 请求的进程相关联的用户 ID 已经正确授权, 如第25页的『使用权限审批』中所说明的那样。然后, 根据已授权用户 ID 所具有的权限, 现有的访问控制机制将对这些数据源作出允许或拒绝访问的决定。

使用 Net.Data 机制

除了上面所说的方法, 还可以使用 Net.Data 配置变量或宏开发技术来限制最终用户的活动、隐蔽诸如数据库设计等共同资产、在产品环境内部验证用户提供的输入值。

Net.Data 配置变量

Net.Data 提供了一些可用于限制最终用户活动或隐蔽数据库设计的配置变量。

用路径语句控制文件访问

Net.Data 评估路径配置语句的设置来确定 Net.Data 宏所使用的文件和可执行程序的位置。这些路径语句标识了一个或多个 Net.Data 在试图找到宏文件、可执行文件、包含文件或其它平面文件时搜索的目录。通过在这些路径语句中有选择地包括目录, 您可以显式地控制用户可以从浏览器访问的文件。请参阅第5页的『第2章 配置 Net.Data』以获取有关路径语句的附加细节。

您还应使用权限检查(如『使用权限』中所述), 并验证 INCLUDE 语句中的文件名不能更改(如第27页的『宏开发技术』中所述)。

对生产系统禁用 SHOWSQL

SHOWSQL 变量允许用户指定让 Net.Data 在 Web 浏览器上显示在 Net.Data 函数内部指定的 SQL 语句。此变量主要用于在应用程序内部开发和测试 SQL, 一般不用于生产系统。

您可以使用以下的某种方法在生产环境中禁用 SQL 语句的显示:

- 使用支持 DTW_SHOWSQL 配置变量的 Net.Data 版本时，在 Net.Data 初始化文件中使用此变量来覆盖在 Net.Data 宏中设置 SHOWSQL 而产生的效果。请参阅第9页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』中的语法和附加信息。

- 使用 DTW_ASSIGN() 函数，如『宏开发技术』中所述。

请参阅*Net.Data* 参考的变量章节中有关 SHOWSQL 的内容，获取 SHOWSQL Net.Data 变量的语法和示例。

宏开发技术

Net.Data 提供了几个允许用户为输入变量赋值的机制。为了确保宏以预期方式执行，宏应确认这些输入变量。您的数据库和应用程序在设计时也应将用户对数据的访问限制在该用户被授权看到的范围内。

在编写 Net.Data 宏时，可以使用以下开发技术。这些技术将帮助您确保应用程序预期完成，并且对数据的访问仅限于正确授权的用户。

确保 Net.Data 变量在 URL 中不会被覆盖

用户在 URL 中对 Net.Data 变量的设置会覆盖宏中用于初始化变量的 DEFINE 语句所产生的作用。这可能会改变宏执行的方式。为了避免这种可能性，可以使用 DTW_ASSIGN() 函数来初始化 Net.Data 变量。

例子： 不使用 %DEFINE SHOWSQL="NO" 设置 Net.Data SHOWSQL 变量，而使用 @DTW_ASSIGN(SHOWSQL, "NO")。那么，诸如 SHOWSQL=YES 等查询字符串赋值就不会覆盖宏的设置了。

您可以使用以下的某种方法在生产环境中禁用 SQL 语句的显示：

- 使用支持 DTW_SHOWSQL 配置变量的 Net.Data 版本时，在 Net.Data 初始化文件中使用此变量来覆盖在 Net.Data 宏中设置 SHOWSQL 而产生的效果。请参阅第9页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』中的语法和附加信息。
- 如上例所述，使用 DTW_ASSIGN() 函数来指定 SHOWSQL 的值，以防该值被覆盖。

请参阅*Net.Data* 参考的变量章节中有关 SHOWSQL 的内容，获取 SHOWSQL Net.Data 变量的语法和示例。

也可以使用 DTW_ASSIGN 来确保其它 Net.Data 变量(例如 RPT_MAX_ROWS 或 START_ROW_NUM) 未被覆盖。请参阅*Net.Data* 参考中的变量章节以了解有关这些变量的更多信息。

请确认 SQL 语句不能以可能改变应用程序预期行为的方式进行修改

对宏中的 SQL 语句添加一个 Net.Data 变量可以允许用户在执行 SQL 语句之前动态地改变该语句。宏的编写者应负责确认用户提供的输入值并确保包含变量引用的 SQL 语句不能以非预期的方式进行修改。Net.Data 应用程序应确认用户从 URL 提供的输入值，这样，Net.Data 应用程序就可以拒绝无效的输入。确认设计过程应包含以下步骤：

1. 标识有效输入的语法；例如，客户 ID 必须以字母开头，并且只能包含字母数字字符。

2. 确定在允许不正确的输入、人为的有害输入、为了访问 Net.Data 应用程序的内部内容而输入某些信息的情况下可能存在哪些潜在的问题。
3. 在满足应用程序需要的宏中包括输入验证语句。这样的验证取决于输入的语法以及如何使用。在比较简单的情况中，它足以检查输入中的无效内容或调用 Net.Data 来验证输入类型。如果输入的语法更为复杂一些，宏的开发者可能就不得不对输入进行部分或完全的语法分析以验证它是否有效。

例 1: 使用 DTW_POS() 字符串函数来验证 SQL 语句

```
%FUNCTION(DTW_SQL) query1() {  
    select * from shopper where shlogid = '$(shlogid)'  
}%
```

shlogid 的值是一个购物者 ID。其作用是将 SELECT 语句返回的行限制在某些特定的行中，这些行中包含有关由购物者 ID 所标识的购物者的信息。当然，如果字符串 'smith' or shlogid<>'smith' 被作为变量 shlogid 的值进行传递，则查询将变为：

```
select * from shopper where shlogid = 'smith' or shlogid<>'smith'
```

原来的 SQL SELECT 语句经过用户如此修改之后将返回整个购物者的表格。

Net.Data 字符串函数可用于验证 SQL 语句未被用户以不恰当的方式修改。例如，以下逻辑可用于确保与 shlogid 变量相关联的输入值仅有一个购物者 ID：

```
@DTW_POS(" ", $(shlogid), result)  
%IF (result == "0")  
    @query1()  
%ELSE  
    %{ perform some sort of error processing %}  
%ENDIF
```

例 2: 使用 DTW_TRANSLATE()

假定应用程序需要确认输入变量 number_of_orders 中所提供的值是一个整数。一种方法是创建一个事务表格 input_translation_table，其中包含除数字字符 0-9 之外的所有键盘字符，并使用 DTW_TRANSLATE 和 DTW_POS 字符串函数来确认输入：

```
@DTW_TRANSLATE(number_of_orders, "x", input_translation_table, "x", string_out)  
  
    @DTW_POS("x", string_out, result)  
  
    %IF (result = "0")  
  
        %{ continue with normal processing %}  
  
%ELSE  
  
    %{ perform some sort of error processing %}  
  
%ENDIF
```

请注意，浏览器前的用户无法修改存储过程中的 SQL 语句，并且用户提供的输入参数值受到与输入参数相关联的 SQL 数据类型的约束。在使用 Net.Data 字符串函数确认用户输入值不可行的情况下，可以使用存储过程。

请确保 INCLUDE 语句中的文件名未以可能改变应用程序预期行为的方式进行修改

如果使用 Net.Data 变量对具有 INCLUDE 语句的文件名指定值，则在执行 INCLUDE 文件之前不会确定要包含的文件。如果您打算在宏中设置此变量的

值，但不允许浏览器前的用户覆盖该宏提供的值，则应使用 DTW_ASSIGN 而不是 DEFINE 来设置此变量的值。如果不打算让浏览器前的用户为文件名提供值，则您的宏应确认提供的值。

例子：诸如 filename=".././x" 的查询字符串赋值将导致从非 INCLUDE_PATH 配置语句正常指定的目录中包含文件。假定 Net.Data 初始化文件中包含以下路径配置语句：

```
INCLUDE_PATH /usr/lpp/netdata/include
```

而 Net.Data 宏中包含以下 INCLUDE 语句：

```
%INCLUDE "${filename}"
```

查询字符串赋值 filename=".././x" 将包含文件 /usr/lpp/x，而这不是 INCLUDE_PATH 配置语句说明所期望的。

Net.Data 字符串函数可用于验证所提供的文件名对于应用程序来说是恰当的。例如，以下逻辑可用于确保与文件名变量相关联的输入值不包含字符串 ".."：

```
@DTW_POS(".", ${filename}, result)
%IF (result > "0")
  %{ perform some sort of error processing %}
%ELSE
  %{ continue with normal processing %}
%ENDIF
```

设计数据库与查询，以便使用户请求无权访问有关其他用户的敏感数据

有些数据库设计为将敏感的用户数据收集在一个单独的表格中。除非 SQL SELECT 请求在某些方面受到限制，否则这种方法可能会使 Web 浏览器前的任何用户都能够看到敏感数据。

例子：以下 SQL 语句返回由变量 order_rn 标识的某个订单的订单信息：

```
select setsstatcode, setsfailtype, mestname
from   merchant, setstatus
where  merfnbr   = setsmenbr
and    setsornbr = $(order_rn)
```

这种方法允许浏览器前的用户随机指定订单号码，这样，他们就可能获得其他客户订单的敏感信息。要防止此类泄密的一个方法是进行以下更改：

- 在订单信息表格中添加一列，它标识与指定行中的订单信息相关联的客户。
- 修改 SQL SELECT 语句，以确保 SELECT 被限定为浏览器前的用户所提供的授权客户 ID。

例如，如果 shlogid 是包含与订单相关联的客户 ID 的列，SESSION_ID 是一个包含浏览器前用户的授权 ID 的 Net.Data 变量，则可以用以下语句来替换前面的 SELECT 语句：

```
select setsstatcode, setsfailtype, mestname
from   merchant, setstatus
where  merfnbr   = setsmenbr
and    setsornbr = $(order_rn)
and    shlogid   = $(SESSION_ID)
```

使用 Net.Data 隐藏变量

您可以使用 Net.Data 隐藏变量，对于那些用 Web 浏览器察看您的 HTML 源码的用户隐藏 Net.Data 宏的各种特性。例如，您可以隐藏数据库的内部结构。请参阅第48页的『隐藏变量』，以获取有关隐藏变量的更多信息。

来自用户的请求确认信息

您可以根据用户提供的输入创建自己的保护方案。例如，您可以通过 HTML 表请求来自用户的验证信息，并使用 `Net.Data` 宏从数据库中检索到的数据进行验证，也可以从 `Net.Data` 宏中所定义的函数中调用一个外部程序。

有关保护资产的更多信息，请参阅以下 Web 站点中有关“经常询问的问题” (FAQ) 的 Internet 安全性列表：

<http://www.w3.org/Security/Faq>

第4章 调用 Net.Data

Net.Data for OS/400 是使用公共网关接口 (CGI) 和一个宏来调用的。这种类型的调用方法称为宏请求。另外，您可以调用持久性宏或者包含指定事务边界的函数的宏。有关持久性宏的更多信息，请参阅第95页的『第7章 具有持久性宏的事务管理』

本章描述使用宏来调用 Net.Data。

- 『使用宏(宏请求)调用 Net.Data』
- 第34页的『调用持久性宏』

使用宏(宏请求)调用 Net.Data

本节将告诉您如何通过指定一个宏来调用 Net.Data。

以下语法语句显示了如何调用 Net.Data。

- URL:

`http://server/Net.Data_invocation_path/filename/block[?name=val&...]`

参数:

server 指定 Web 服务器的名称和路径。如果是本地服务器，则可以忽略服务器名称而使用相关的 URL。

Net.Data_invocation_path

Net.Data 可执行文件的路径和文件名。例如，`/cgi-bin/db2www/`。

filename

指定 Net.Data 宏文件的名称。Net.Data 搜索并试图用 MACRO_PATH 初始化路径变量中定义的路径语句来与这个文件名匹配。请参阅第13页的『MACRO_PATH』以获取更多信息。

block 在引用的 Net.Data 宏中指定 HTML 块的名称。

method 指定与表一起使用的 HTML 方法。

?name=val&...

指定一个或多个传递给 Net.Data 的可选参数。

然后，您可以在浏览器中直接指定 URL，也可以在 HTML 链或表中使用，如下所示：

- HTML 链:

`any text`

- HTML 表:

`<FORM METHOD=method ACTION="URL">any text</FORM>`

参数:

method 指定与表一起使用的 HTML 方法。

URL 指定用于运行 Net.Data 宏的 URL，其中的参数已经在上面描述过了。

例子

以下例子演示了调用 Net.Data 的不同方式。

例 1: 使用 HTML 链调用 Net.Data:

```
<A HREF="http://server/cgi-bin/db2www/myMacro.d2w/report">  
.  
.  
.  
</A>
```

例 2: 使用表来调用 Net.Data

```
<FORM METHOD=POST  
  ACTION="http://server/cgi-bin/db2www/myMacro.d2w/report">  
.  
.  
.  
</FORM>
```

例 3: 使用 HTML 链在 qsys.lib 文件系统中调用 Net.Data 宏:

```
<A HREF="http://server/cgi-bin/db2www/myMacro.mbr/report">  
.  
.  
.  
</A>
```

例 4: 使用表在 qsys.lib 文件系统中调用 Net.Data 宏:

```
<FORM METHOD=POST  
  ACTION="http://server/cgi-bin/db2www/  
  qsys.lib/mylib.lib/myfile.file/myMacro.mbr/report">  
.  
.  
.  
</FORM>
```

以下章节描述了 HTML 链和表, 以及有关如何使用链和表来调用 Net.Data 的更多信息:

- 『HTML 链』
- 第33页的『HTML 表』

HTML 链

如果要创作一个 Web 页面, 可以创建一个 HTML 链, 而这个链将执行一个 HTML 块。当浏览器前的用户单击被定义为 HTML 链的文本或图象时, Net.Data 就将执行宏中的 HTML 块。

要创建一个 HTML 链, 可使用 HTML <a> 标记。确定希望用作指向 Net.Data 宏的超链的文本或图形, 然后在两端加上 <a> 和 标记。在 <a> 标记的 HREF 属性中, 指定宏和 HTML 块。

以下例子显示了这样一个链: 当用户在 Web 页面上选择文本 "List all monitors" 时, 这个链将使得一个 SQL 查询开始执行。

```
<a href="http://server/cgi-bin/db2www/listA.d2w/report">
List all monitors</a>
```

单击调用宏 listA.d2w 的链接，它有一个名为 "report" 的 HTML 块，如下例所示：

```
%DEFINE DATABASE="MNS97"
```

```
%FUNCTION(DTW_SQL) myQuery() {
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='MONITOR'
%}
```

```
%HTML (report){
  @myQuery()
%}
```

查询将返回一个包含型号、成本和 EQPTABLE 表格中对每个监视器所描述信息的表格。此例通过生成一个缺省报表显示了查询的结果。请参阅第62页的『报表块』以获取有关如何使用 REPORT 块定制报表的信息。

HTML 表

您可以使用 HTML 表来动态地定制 Net.Data 宏的执行。这些表允许用户提供输入值，而这些值将影响宏的执行和 Net.Data 构建的 Web 页面的内容。

以下例子构建在第32页的『HTML 链』中监视器列表的例子，它使得浏览器前的用户可以使用一个简单的 HTML 表来选择要求显示信息的产品类型。

```
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD=POST ACTION="/cgi-bin/db2www/equip1st.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="SCN"> Scanners
</MENU>

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
```

当浏览器前的用户作出了他们的选择并单击“提交”按钮之后，Web 服务器将处理调用 Net.Data 的 FORM 标记的 ACTION 参数。然后，Net.Data 将执行 equip1st.d2w 宏中的 Report 块：

```
%DEFINE DATABASE="MNS97"

%FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='${hardware}'
%REPORT{
<H3>Here is the list you requested</H3>
%ROW{
<HR>
$(N1): $(V1), $(N2): $(V2)
<P>$(N3): $(V3)
%}
%}
%}
```

```
%HTML (report){
    @myQuery()
%}
```

在上述例子中，SQL 语句中 `TYPE=$(hardware)` 的值是从 HTML 表的输入中获得的。

请参阅 *Net.Data* 参考以获取对于 ROW 块中所使用变量的详细描述。

调用持久性宏

本节将告诉您如何调用持久性宏。这些宏包含用于事务处理的函数。调用这些宏类似于一般的宏请求，需要指定服务器、宏和 HTML 块。对于持久性宏，还要指定一个事务句柄，用来将 HTML 块标识为事务的一部分。

有关持久性宏和事务处理的更多信息，请参阅第95页的『第7章 具有持久性宏的事务管理』。

持久性宏的语法

使用以下语法来调用持久性宏：

- HTML 链：

```
<A HREF="http://server/Net.Data_invocation_path/transaction_handle/filename/
block/[?name=val&...]">any text</A>
```

- HTML 表：

```
<FORM METHOD=method ACTION="http://server/Net.Data_invocation_path/
transaction_handle/filename/block/
[?name=val&...]">any text</FORM>
```

- URL：

```
http://server/Net.Data_invocation_path/transaction_handle/filename/block/
[?name=val&...]
```

参数：

server 指定了 Web 服务器的名称。如果是本地服务器，则可以忽略服务器名称而使用相关的 URL。

Net.Data_invocation_path

Net.Data 可执行文件的路径和文件名。例如，`/cgi-bin/db2www/`。

transaction_handle

指定哪些 URL 是由 Net.Data 宏启动的事务中的一部分。此标识符是通过调用 `DTW_RTVHANDLE` 内部函数来得到的，并且必须跟在 *Net.Data_invocation_path* 之后。

filename

指定 Net.Data 宏文件的名称。Net.Data 搜索并试图用 `MACRO_PATH` 初始化路径变量中定义的路径语句来与这个文件名匹配。请参阅第13页的『`MACRO_PATH`』以获取更多信息。

block 在引用的 Net.Data 宏中指定 HTML 块的名称。

method 指定与表一起使用的 HTML 方法。

`?name=val&...`

指定一个或多个传递给 Net.Data 的可选参数。

例子

以下例子演示了如何调用持久性宏。

例 1: 宏中的 URL:

`http://www.mycompany.com/cgi-bin/db2www/$(handle)/mymacro.mac/report1`

例 2: 具有链接到同一事务内其它宏调用的链的典型 HTML 块

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html(report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
pcgil.mbr/quit">quit</a><br>
%}
```


第5章 开发 Net.Data 宏

Net.Data 宏是一个文本文件，它由一系列 Net.Data 宏语言结构组成：

- 指定 Web 页的版面设置
- 定义变量和函数
- 调用 Net.Data 的内部函数或宏中定义的函数
- 格式化处理输出，并把它返回给 Web 浏览器显示

正如图6所示，Net.Data 宏包括两个组织部分：声明部分和呈示部分。

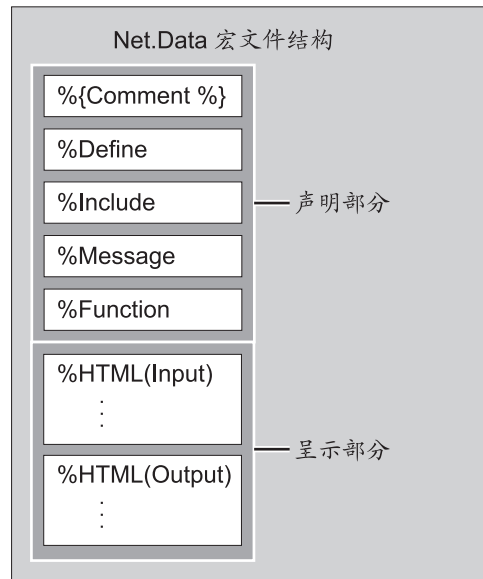


图 6. 宏结构

- 声明部分包含宏中变量和函数的定义。
- 呈示部分包含指定 Web 页版面设置的 HTML 块。HTML 块由受 Web 浏览器支持的文本呈示语句组成，例如 HTML 和 JavaScript。

您可以以任何顺序多次使用这些部分。请参阅 *Net.Data* 参考以获取关于宏各部分的语法和结构。

权限技巧： 确保此 Web 服务器具有对该文件的访问权。请参阅第20页的『对 Net.Data 访问的对象授予访问权限』，以获取更多信息。

本章检查组成 Net.Data 宏的不同块以及用于写宏的方法。

- 第38页的『Net.Data 宏的剖析』
- 第41页的『Net.Data 宏变量』
- 第52页的『Net.Data 函数』
- 第61页的『在宏中生成 Web 页面』
- 第67页的『宏中的条件逻辑和循环』

Net.Data 宏的剖析

宏由两部分组成:

- 声明部分, 包含了要在呈现部分中使用的定义。声明部分使用两个主要的可选块:
 - DEFINE 块
 - FUNCTION 块

声明部分还可以包含其它语言结构和语句, 例如 EXEC 语句、IF 块、INCLUDE 语句和 MESSAGE 块。关于语言结构的更多信息, 请参阅*Net.Data* 参考中关于语言结构的章节。

权限技巧: 确保此 Web 服务器具有对 EXEC 和 INCLUDE 语句引用的文件的访问权。请参阅第20页的『对 Net.Data 访问的对象授予访问权限』, 以获取更多信息。

- 呈现部分定义 Web 页面的布局、引用变量, 并使用 HTML 块作为宏的入口和出口点来调用函数。在调用 Net.Data 时, 指定一个 HTML 块名作为处理宏的入口点。第40页的『HTML 块』中描述了 HTML 块。

在本小节中, 用一个 Net.Data 宏说明了宏语言的元素。此例子宏表达了一种格式, 该格式提示要向 REXX 程序传送的信息。宏将此信息传送至名为 ompsamp.mbr 的外部 REXX 程序, 该程序回送用户输入的数据。然后在第二个 HTML 页面上显示结果。

首先看整个宏, 然后详细看每块:

```
%{ ***** DEFINE 块 *****%}
%DEFINE{
    page_title="Net.Data Macro Template"
}%

%{ ***** FUNCTION 定义块 *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
{
    %EXEC{ompsamp.mbr %}
}%

%FUNCTION(DTW_REXX) today () RETURNS(result)
{
    result = date()
}%

%{ ***** HTML 块: 输入 *****%}
%HTML(INPUT) {
<html>
  <head>
<title>$(page_title)</title>
</head><body>
<h1>Input Form</h1>
Today is @today()

<FORM METHOD="post" ACTION="OUTPUT">
输入一些数据传送给一个 REXX 程序:
<INPUT NAME="input_data" TYPE="text" SIZE="30">
<p>
<INPUT TYPE="submit" VALUE="Enter">

</form>

< hr>
<p>[<a href="/">Home page</a>]
</body></html>
}%
```



```
%{ ***** HTML 块: 输出 *****%}
%HTML(OUTPUT) {
<html>
  <head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rex1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
}%}
```

示例宏包含四个主要块: DEFINE、FUNCTION 和两个 HTML 块。在一个 Net.Data 宏中可以有多个 DEFINE、FUNCTION 和 HTML 块。

两个 HTML 块包含了诸如 HTML 等文本呈现语句, 这使 Web 宏更容易书写。如果您熟悉 HTML, 就知道构建一个宏只涉及添加要在服务器上动态处理的宏语句和要发送到数据库的 SQL 语句。

虽然宏看起来类似于 HTML 文档, 但是 Web 服务器是通过 Net.Data 使用 CGI 来访问它的。要调用宏, Net.Data 需要两个参数: 要处理的宏的名称和该宏中要显示的 HTML 块。

调用了宏之后, Net.Data 从头开始处理它。以下各章节着眼于当 Net.Data 处理文件时所发生的事情。

DEFINE 块

DEFINE 块包含了 DEFINE 语言结构以及以后要在 HTML 块中使用的变量定义。下例显示具有一个变量定义的 DEFINE 块:

```
%{ ***** DEFINE 块 *****%}
%DEFINE{
  page_title="Net.Data Macro Template"
}%}
```

第一行是一个注解。注解是 %{ 和 %} 中的任何文本。注解可处于宏中的任何地方。下一个语句起始于一个 DEFINE 块。可以在一个定义块中定义多个变量。在此例中, 只定义了一个变量 page_title。定义了该变量之后, 就可以使用语法 \$(page_title) 在宏中的任何地方引用它。使用变量便于以后对宏作全局变更。该块的最后一行 %} 标识了 DEFINE 块的结束。

FUNCTION 块

FUNCTION 块包含了对 HTML 块所调用的函数的说明。函数由语言环境处理, 可以执行程序、SQL 查询或存储过程。

以下示例显示了两个 FUNCTION 块。一个定义对外部 REXX 程序的调用, 另一个包含内联的 REXX 语句。

```
%{ ***** FUNCTION 块 *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- 该函数接受
                                                         一个参数并返回变量 "result",
它是
                                                         由外部 REXX 程序指定的
```

```
%EXEC{ompsamp.mbr %} <--- 该函数执行一个名为 "ompsamp.mbr" 的外部 REXX 程序
%}

%FUNCTION(DTW_REXX) today () RETURNS(result) {
    result = date() <--- 该函数的单独的源语句包含在内部。
%}
```

第一函数块 rexx1 是一个 REXX 函数说明，它依次运行一个名为 ompsamp.mbr 的外部 REXX 程序。输入变量 input 被该函数接受并自动传送至外部 REXX 命令。REXX 命令还返回变量 result。REXX 命令中的变量 result 的内容代替 OUTPUT 块中包含的调用的 @rexx1() 函数调用。REXX 程序可直接访问 input 和 result 变量，正如 ompsamp.mbr 源码中所显示的那样：

```
/* REXX */
result = 'The REXX program received "'input'" from the macro.'
```

该函数中的代码回送传送给它的数据。您可以通过用一般“标记”风格的标记(象 或)括起请求的 @rexx1() 函数来按您的需要格式化结果文本。与 result 变量相比，REXX 程序宁愿使用 REXX SAY 语句来将 HTML 标记写入标准输出。

第二个函数块也引用一个 REXX 程序 today。但是，在此情况下，整个 REXX 程序都包含在本身的函数说明中。不需要外部程序。REXX 和 Perl 函数都允许内部程序，因为它们解释语言，可以动态语法分析和执行。内部程序的优点是简明，不需要一个程序文件来管理。第一个 REXX 函数也可以内部处理。

HTML 块

HTML 块定义 Web 页面的版面设置，引用变量并调用函数。HTML 块被用作宏的入口点和出口点。HTML 块总是在 Net.Data 宏请求中指定的，并且每个宏必须至少有一个 HTML 块。

示例宏中的第一个 HTML 块名为 INPUT。HTML(INPUT) 块包含用于具有一个输入字段的简单表的 HTML。

```
%{ ***** HTML 块: 输入 *****%}
%HTML (INPUT) { <--- 标识该 HTML 块的名称。
<html>
<head>
<title>$(page_title)</title> <--- 加注变量替换。
</head><body>
<h1>Input Form</h1>
Today is @today() <--- 该行包含对一个函数的调用。

<FORM METHOD="post" ACTION="OUTPUT"> <--- 该表提交时，将调用 "OUTPUT" HTML 块。
输入一些数据传送给一个 REXX 程序:
<INPUT NAME="input_data" <--- 在表被提交时 "input_data" 被定义，
TYPE="text" SIZE="30"> 并且可以在该宏中的任何地方引用它。
                        它已被初始化为输入字段中的某种用户类型。

<p>
<INPUT TYPE="submit" VALUE="Enter">

< hr>
<p>
[
<a href="/">Home page</a>]
</body><html>
%} <--- 关闭 HTML 块。
```

整个块由 HTML 块标识符 %HTML (INPUT) {...%}括起。INPUT 标识了该块的名称。名称可以包含任何字母数字字符、下划线字符或句点。HTML <title> 标记包含了变量替换的例子。变量 page_title 的值替换了表的标题。

此块还具有一个函数调用。表达式 @today() 是对函数 today 的调用。该函数是在上面描述的 FUNCTION 块中定义的。Net.Data 将 today 函数的结果(即当前日期)插入 HTML 文本中与 @today() 表达式相同的位置。

FORM 语句的 ACTION 参数提供了一个在 HTML 块之间或在宏之间游历的例子。表提交时, ACTION 参数中对另一块名称的引用将访问此块。HTML 表中的任何输入数据都作为隐式变量传送给块。该表上定义的单个输入字段, 就是这样的。当表提交时, 在此表中输入的数据被传送到变量 input_data 中的 HTML(OUTPUT) 块。

如果另一个宏在相同的 Web 服务器上, 您就可以用交叉引用来访问该宏中的 HTML 块。例如, ACTION 参数 ACTION="../othermacro.d2w/main" 访问宏 othermacro.d2w 中 HTML 块调用的主程序。表中输入的任何数据类型再次在 input_data 中传送给该宏。

在调用 Net.Data 时, 您将变量作为 URL 的一部分传送。例如:

```
<a href="/cgi-bin/db2www/othermacro.d2w/main?input_data=value">Next macro</a>
```

您可以通过引用表中指定的变量名来访问或处理宏中的表数据。

例子中的下一个 HTML 块是 HTML(OUTPUT) 块。它包含 HTML 标记和 Net.Data 宏语句, 这些语句定义 HTML(INPUT) 请求所处理的输出。

```
%{ *****          HTML 块: 输出          *****%}
%HTML(OUTPUT) {
<html>
  <head>
<title>$(page_title)</title> <--- 更多替换。

</head><body>
<h1>Output Page</h1>
<p>@rex1(input_data) <--- 该行包含对函数 rex1 的调用, 并传送变元 "input_data"。
<p>
< hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>]
%}
```

与 HTML(INPUT) 块相似, 此块也是标准 HTML, 其中使用 Net.Data 宏语句来替换变量和函数调用。再次用 page_title 变量来替换标题语句。与上面相似, 此块也包含一个函数调用。在此情况下, 它调用函数 rex1 并将变量 input_data 的内容传送给它, 该变量接收自 Input 块中定义的表。您可以将任何数目的变量传送给函数, 或从函数中传送回任何数目的变量。函数定义指定传递的变量数目及其用法。

Net.Data 宏变量

Net.Data 允许您在 Net.Data 宏中定义和引用变量。另外, 也可以将这些变量从宏传送给语言环境, 反之亦然。Net.Data 记号, 例如变量名和值, 以及文字串都可以包含多达 256 KB 的数据。对于 OS/400, 最大记号大小是由操作系统确定的。个别语言环境可能对值大小提供其它限制。

可以根据变量类型和是否具有预定义值来定义 Net.Data 变量。可以根据变量的定义将这些变量分为以下类型:

- 在 DEFINE 块中用 DEFINE 语句显式定义的变量
- 预定义变量, 这些变量由 Net.Data 提供并设为一个值。此值通常无法更改。
- 隐式定义的变量, 有四类:
 - 未显式定义, 但是在首次被赋值时例示的那些变量
 - 参数变量, 它们是 FUNCTION 块定义的一部分, 只可以在 FUNCTION 中引用。
 - 由 Net.Data 例示并对应于表数据或查询字符串数据的那些变量。
 - 与一个 Net.Data 表格相关联并只可以在 ROW 或 REPORT 块中引用的那些变量。

以下章节将描述:

- 『标识符作用域』
- 第43页的『定义变量』
- 第44页的『引用变量』
- 第46页的『变量类型』

标识符作用域

一个标识符(它是一个变量或函数调用)成为可见, 意味着当它被说明或例示之后就可被引用。标识符可见的区域称为它的作用域。作用域有 5 种类型:

- 全局

如果您可以在一个宏中的任何地方引用一个标识符, 则该标识符就具有全局作用域。具有全局作用域的标识符有:

- Net.Data 内部函数
- 表数据
- 查询字符串数据
- 在一个 HTML 块中例示的变量

- 宏

如果一个标识符的说明出现在任何块的外面, 则它有此作用域。一个块以左括号 ({} 开始, 以百分号加右括号 (%) 结束。(注意, DEFINE 块是此定义的例外, 应当将它作为独立的 DEFINE 语句处理)。与具有全局作用域的标识符不同, 具有宏作用域的标识符只能由该宏中位于标识符声明之后的项引用。

- FUNCTION 块或 MACRO_FUNCTION 块

如果一个标识符满足以下条件, 则它具有函数块作用域:

- 标识符在函数定义的参数表中说明。
 - 如果一个标识符在函数定义的外面已经存在相同名称, 那么 Net.Data 将使用函数块中的参数表中的标识符。
- 标识符在函数块中实例化, 并且在函数调用之前没有说明或实例化。

如果一个标识符在函数外已被说明或初始化并且没有在函数参数表中说明, 则该标识符不具有函数块作用域。标识符在函数块中的值保持不变, 除非由函数进行更新。

- REPORT 块

如果一个标识符只可以在 REPORT 块中被引用(例如表列名 N1、N2、...、Nn)，则它具有报表块作用域。只有 Net.Data 隐式定义为表处理的一部分的那些变量才可以具有报表块作用域。例示的任何其它变量都具有函数块作用域。

- ROW 块

如果只可以从 ROW 块中调用一个标识符(例如表值名 V1、V2、...、Vn)，则该标识符具有行块作用域。只有 Net.Data 隐式定义为表处理的一部分的那些变量才可以具有行块作用域。例示的任何其它变量都具有函数块作用域。

定义变量

Net.Data 宏中有三种定义变量的方式:

- 定义语句或块
- HTML 表标记
- 查询字符串数据

从表或查询字符串数据接收到的变量值将覆盖 DEFINE 语句在 Net.Data 宏中设置的变量值。

- **DEFINE 语句或块**

定义一个变量以在 Net.Data 宏中使用的最简单方式是使用 DEFINE 语句。语法如下:

```
%DEFINE variable_name="variable value"

%DEFINE variable_name={ variable value on multiple
                        lines of text %}

%DEFINE{
    variable_name1="variable value 1"
    variable_name2="variable value 2"
%}
```

variable_name 是给予变量的名称。变量名必须以字母或下划线开头，可以包含任何字母数字字符、下划线字符、句点或散列字符 (#)。所有变量名都是区别大小写的，但是 *N_columnName* 和 *V_columnName* 除外，它们是表变量。

例如:

```
%DEFINE reply="hello"
```

变量 *reply* 具有值 *hello*。

单独的两个连续引号等于一个空串。例如:

```
%DEFINE empty=""
```

变量 *empty* 具有一个空字符串。

如果变量中包含特殊字符，例如行结束符，则在该值两侧使用块花括号:

```
%DEFINE introduction={
Hello,
My name is John.
%}
```

要在字符串中包含引号，可以使用两个连续的引号。

```
%DEFINE HI="say ""hello"""
```

还可以使用块花括号来避免使用引号:

```
%DEFINE HI={ say "hello" %}
```

要在一个 `DEFINE` 语句中定义几个变量, 可使用 `DEFINE` 块:

```
%DEFINE{
    variable1="value1"
    variable2="value2"
    variable3="value3"
    variable4="value4"
%}
```

- **HTML 表标记: `SELECT`, `INPUT`, 以及 `TEXTAREA`**

可以使用 `HTML FORM` 标记来为变量赋值, 这些标记有 `SELECT`、`INPUT` 和 `TEXTAREA` 标记。以下示例使用标准 `HTML` 表标记来定义 `Net.Data` 变量:

```
<INPUT NAME="variable_name" TYPE=...>
```

或

```
<SELECT NAME="variable_name">
  <OPTION>value one
  <OPTION>value two
</SELECT>
```

要指定跨多行或包含特殊字符(例如, 引号)的变量, `TEXTAREA` 标记可用于:

```
<TEXTAREA NAME="variable_name" ROWS="4">
Please type the multi-line value
of your variable here.
</TEXTAREA>
```

`variable_name` 是给予变量的名称, 而变量值是根据表中接收的输入来确定的。请参阅第33页的『`HTML` 表』以获取关于如何在 `Net.Data` 宏中使用此类变量定义的例子。

- **查询字符串数据**

可以通过查询字符串将变量传递给 `Net.Data`。例如:

```
http://www.ibm.com/cgi-bin/db2www/stdqry1.d2w/input?field=custno
```

在上例中, 变量名 `field` 和变量值 `custno` 指定 `Net.Data` 接收自查询字符串的附加数据。`Net.Data` 接收并处理数据, 如同来自表数据一样。

引用变量

您可以引用先前定义变量以返回它的值。要在 `Net.Data` 宏中引用一个变量, 可在 `$(` 和 `)` 中指定变量名。例如:

```
$(variableName)
$(homeURL)
```

当 `Net.Data` 发现一个变量引用时, 它用变量的值来替换变量引用。变量引用可以包含字符串、变量引用和函数调用。

可以动态生成变量名。如果列表中的个数无法预先确定, 则通过这种技术可以使用循环来为运行时构建的列表处理大小可变的表格或输入数据。例如, 可以生成 `HTML` 表元素列表, 这些表元素是根据 `SQL` 查询所返回的记录生成的。

要将变量作为文本呈示语句的一部分使用, 可在宏的 `HTML` 块中引用它们。

无效的变量引用: 无效的变量引用将被分辨为空字符串。例如，如果变量引用中包含无效字符(例如，感叹号 !)，该引用将被分辨为空字符串。

有效的变量名必须以字母数字字符或下划线开头，可以包含字母数字字符(包括句点、下划线以及散列标记)。

例 1: 链中的变量引用

如果定义了变量 *homeURL*:

```
%DEFINE homeURL="http://www.ibm.com/"
```

您可以指向主页为 *\$(homeURL)* 并创建一个链接:

```
<A href="$(homeURL)">Home page</A>
```

您可以在 *Net.Data* 宏中的许多部分引用变量; 请查看本章中的语言结构以便确定在宏中的哪些部分允许变量引用。如果变量在被引用时尚未被定义，*Net.Data* 将返回一个空字符串。单独的变量引用不定义变量。

例 2: 动态生成变量引用

假定您在运行一个具有任意个成分的 SQL SELECT 语句。可以使用以下 ROW 块创建具有输入字段的 HTML 表:

```
...
%ROW {
<input type=text name=@dtw_rconcat("I", ROW_NUM) size=10 maxlength=10>
%}
...
```

因为创建了 INPUT 字段，您可能希望访问用户在向宏提交表以备处理时输入的值。可以编写一段代码(循环)来检索变量长度列表中的值:

```
<PRE>
...
@dtw_assign(rowIndex, "1")
%while (rowIndex <= rowCount) {
The value entered for row $(rowIndex) is: $(I$(rowIndex))
@dtw_add(rowIndex, "1", rowIndex) %}
...
</PRE>
```

Net.Data 先使用 *I\$(rowIndex)* 引用生成变量名。例如，第一个变量名称是 *I1*。然后，*Net.Data* 就可以使用该值并分辨为变量的值。

例 3: 用嵌套的变量引用和函数调用进行变量引用

```
%define my = "my"
%define u = "lower"
%define myLOWERvar = "hey"

$( $(my)@dtw_ruppercase(u)var)
```

变量引用将返回 *hey* 的值。

变量类型

在宏在可以使用以下类型的变量。

- 『条件变量』
- 『环境变量』
- 第47页的『可执行变量』
- 第48页的『隐藏变量』
- 第49页的『列表变量』
- 第49页的『表格变量』
- 第50页的『杂项变量』
- 第51页的『表格处理变量』
- 第51页的『报表变量』
- 第52页的『语言环境变量』

如果您将字符串赋给变量，而变量由 `Net.Data` 定义为某种方式，例如 `ENVVAR`、`LIST`、条件列表变量，则变量不再表现为定义的方式。换句话说，变量成为一个包含字符串的简单变量。

请参阅 *Net.Data* 参考中每个变量的语法和例子。

条件变量

条件变量让您通过使用类似于 `IF`、`THEN` 结构的方法来为一个变量定义一个条件值。在定义条件变量时，可以指定两个可能的变量值。如果引用的第一个变量存在，条件变量将获取第一个值；否则获取第二个值。条件变量的语法是：

```
varA = varB ? "value_1" : "value_2"
```

如果 `varB` 已定义，则 `varA="value_1"`，否则 `varA="value_2"`。这是等价于使用 `IF` 块，如下例所示：

```
%IF ($(varB))
    varA = "value_1"
%ELSE
    varA = "value_2"
%ENDIF
```

请参阅第49页的『列表变量』以获取使用条件变量与列表变量的例子。

环境变量

您可以引用那些 `Web` 服务器使之对正在处理您的 `Net.Data` 请求的进程或线程可用的环境变量。当引用 `ENVVAR` 变量时，`Net.Data` 返回同名的环境变量的当前值。

定义环境变量的语法是：

```
%DEFINE var=%ENVVAR
```

其中 `var` 是要定义的环境变量名。

例如, 变量 `SERVER_NAME` 可被定义为环境变量:

```
%DEFINE SERVER_NAME=%ENVVAR
```

然后被引用:

```
The server is $(SERVER_NAME)
```

输出是这样的:

```
The server is www.software.ibm.com
```

请参阅*Net.Data* 参考以获取有关 `ENVVAR` 语句的更多信息。

可执行变量

您可以用可执行变量来从变量引用中调用其它函数。

使用 `DEFINE` 块中的 `EXEC` 语言结构来定义 `Net.Data` 宏中的可执行变量。有关 `EXEC` 语言环境元素的更多信息, 请参阅*Net.Data* 参考中有关语言结构的章节。在下例中, 定义了变量 `runit` 来执行可执行程序 `testProg`:

```
%DEFINE runit=%EXEC "testProg"
```

`runit` 成为可执行变量。

`Net.Data` 在 `Net.Data` 宏中遇到一个有效变量时运行可执行程序。例如, 当 `Net.Data` 宏中有一个有效变量引用建立成变量 `runit` 时, 即执行 `testProg` 程序。

一种简单的方法是从另一个变量定义中引用一个可执行变量。以下例子演示了这个方法。变量 `date` 定义成一个可执行变量, `dateRpt` 包含对可执行变量的引用。

```
%DEFINE date=%EXEC "date"  
%DEFINE dateRpt="Today is $(date)"
```

不管 `$(dateRpt)` 出现在 `Net.Data` 宏中的何处, `Net.Data` 都搜索可执行程序 `date`, 并在找出时返回:

```
Today is Tue 11-07-1999
```

当 `Net.Data` 在宏中遇到可执行变量时, 它将使用下列方法寻找被引用的可执行程序:

1. 它在 `Net.Data` 初始化文件由 `EXEC_PATH` 指定的目录中搜索。请参阅第14页的『`EXEC_PATH`』, 以获取细节。
2. 如果 `Net.Data` 找不到此程序, 系统将搜索系统 `PATH` 环境变量或库表所定义的目录。如果找到了此可执行程序, 则 `Net.Data` 运行它。

限制: 不要将可执行变量设置成它调用的可执行程序的输出值。在先前的例子中, 变量 `date` 的值为空 (`NULL`)。如果在 `DTW_ASSIGN` 函数调用中使用此变量来把它的值分配给另一个变量, 则赋值后新变量的值也是空 (`NULL`)。可执行变量的唯一目的是去调用它定义的程序。

也可以给要执行的程序, 通过在变量定义上指定此程序名, 将参数传送给它。在此例中, 距离和时间的值传送给程序 `calcMPH`。

```
%DEFINE mph=%EXEC "calcMPH $(distance) $(time)"
```

下一个例子将系统日期作为报表的一部分返回:

```
%DEFINE database="ce1dial"
%DEFINE tstamp=%EXEC "date"

%FUNCTION(DTW_SQL) myQuery() {
SELECT CUSTNO, CUSTNAME from dist1.customer
%REPORT{
%ROW{
<A HREF="/cgi-bin/db2www/exmp.d2w/report?value1=$(V1)&value2=$(V2)">
$(V1) $(V2) </A> <BR>
%}
%}
%}

%HTML (report){
<H1>Report made: $(tstamp) </H1>
@myQuery()
%}
```

每个报表都显示日期以便于跟踪。此例还将用户号和名称放在另一个 `Net.Data` 宏的链接中。单击报表中的任何用户将调用 `exmp.d2w Net.Data` 宏，即将用户号和名字传送给 `Net.Data` 宏。

隐藏变量

您可以使用隐藏变量，对用他们的 Web 浏览器察看您的 Web 页面源码的用户隐藏应用程序的实际变量名。要定义隐藏变量：

1. 在 HTML 块中变量的最后一个引用之后，为每个需要隐藏的字符串定义一个变量。如下例所示，变量在 HTML 块中使用之后，总是用 `DEFINE` 语言结构来定义。`$(variable)` 变量被引用然后被定义。
2. 在引用此变量的 HTML 块中，使用两个美元符号代替一个美元符号来引用变量。例如，`$(X)` 替换 `$(X)`。

```
%HTML(INPUT) {
<FORM ...>
<P>Select fields to view:
shanghai<SELECT NAME="Field">
<OPTION VALUE="$(name)"> Name
<OPTION VALUE="$(addr)"> Address
...
</FORM>
%}

%DEFINE{
name="customer.name"
addr="customer.address"
%}

%FUNCTION(DTW_SQL) mySelect() {
SELECT $(Field) FROM customer
%}

...
```

Web 浏览器显示 HTML 表时，`$(name)` 和 `$(addr)` 分别被替换以 `$(name)` 和 `$(addr)`，所以实际的表格和列名肯定不出现在 HTML 表上。应用程序用户无法区分实际变量名是隐藏的。当用户提交这个表时，调用 `HTML(REPORT)` 块。当 `@mySelect()` 调用 `FUNCTION` 块时，`$(Field)` 在 SQL 语句中用 SQL 查询的 `customer.name` 或 `customer.addr` 替换。

列表变量

使用列表变量来构建一个定界的值字符串。当要构建一个具有多个项目的 SQL 查询时（象某些 WHERE 或 HAVING 语句一样），它们特别有用。列表变量的语法是：

```
%LIST " value_separator " variable_name
```

建议：空格是必须的。在大多数情况下，在值分隔符之前和之后都插一个空格。大部分查询都为值分隔符使用布尔或数学运算符（例如，AND、OR 或 >）。下例说明条件、隐藏和列表变量的使用：

```
%HTML(INPUT) {
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/example2.d2w/report">
<H2>Select one or more cities:</H2>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$ (cond1)">Sao Paolo<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$ (cond2)">Seattle<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$ (cond3)">Shanghai<BR>
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
%}

%DEFINE{
DATABASE="custcity"
%LIST " OR " conditions
cond1="cond1='Sao Paolo'"
cond2="cond2='Seattle'"
cond3="cond3='Shanghai'"
whereClause= ? "WHERE $(conditions)" : ""
%}

%FUNCTION(DTW_SQL) mySelect() {
SELECT name, city FROM citylist
$(whereClause)
%}

%HTML(REPORT){
@mySelect()
%}
```

在 HTML 表中，如果没有选择任何校验框，则 conditions 为 NULL，因此查询中的 whereClause 也为 NULL。否则，whereClause 中包含了选定的值，值之间用 OR 分隔。例如，如果选择了所有这三个城市，则 SQL 查询为：

```
SELECT name, city FROM citylist
WHERE cond1='Sao Paolo' OR cond2='Seattle' OR cond3='Shanghai'
```

此例显示已选择 Seattle，这出现在该 SQL 的结果中。

```
SELECT name, city FROM citylist
WHERE cond1='Seattle'
```

表格变量

表格变量定义相关数据的集合。它包含一系列行和列，包括一行列标题。在 Net.Data 宏中如以下语句中所示地定义一个表格：

```
%DEFINE myTable=%TABLE(30)
```

%TABLE 后面的数目是对此表格变量可包含行数的限制。要指定不具有行数限制的表格，可如下例所示地使用缺省值或指定 ALL：

```
%DEFINE myTable2=%TABLE
%DEFINE myTable3=%TABLE(ALL)
```

在定义表格时，它具有零行和零列。用值填充表格的唯一方式是将其作为 OUT 或 INOUT 参数来传送给一个函数，或者使用 Net.Data 提供的内部表函数。DTW_SQL 语言环境自动将 SELECT 语句的结果放到表格中。

对于非数据库语言环境，例如 DTW_REXX 或 DTW_PERL，语言环境也负责设置表格值。当然，语言环境脚本或程序将逐格定义表格值。请参阅第71页的『第6章 使用语言环境』以获取有关语言环境如何使用表格变量的更多信息。

您可以通过引用表格变量名来传送一个表格。可以在一个函数的 REPORT 块中引用表格中的各个单独成份，也可以使用 Net.Data 表格函数来实现。请参阅第51页的『表格处理变量』以了解如何在 REPORT 块中访问表格内的单独成份，参阅第60页的『表格函数』以了解如何使用表格函数来访问表格的单独成分。在 SQL 函数中表格变量通常是填充了值的，然后，表格变量在 SQL 函数或另一个函数中在作为参数传送给该函数之后，被用作对报表的输入。您可以将表格变量作为 IN、OUT 或 INOUT 参数传送给任何非 SQL 函数。表格只能作为 OUT 参数传递给 SQL 函数。

如果引用了一个表格变量，则将显示表格的内容，并根据 DTW_HTML_TABLE 变量的设置对其进行格式化。在下面的例子中，将显示 myTable 的内容：

```
%HTML (output) {  
    $(myTable)  
}
```

表格中的列名和字段值被编址为起始地址为 1 的数组元素。

杂项变量

这些变量是 Net.Data 定义的变量，可用来：

- 影响 Net.Data 的处理
- 查找函数调用的状态
- 获取关于数据库查询结果集的信息
- 确定关于文件位置和日期的信息

杂项变量即可以具有 Net.Data 确定的预定义值，又可具有您设置的值。例如，Net.Data 根据正在处理的当前文件来确定 DTW_CURRENT_FILENAME 变量值，您可以在该文件中指定 Net.Data 是否除去由制表机和新行字符产生的额外空白。

预定义变量在宏中用作变量引用，并提供关于一个函数调用的状态、日期或文件的正确状态。例如，要检索当前文件的名称，可使用：

```
%REPORT {  
<p>This file is <i>$(DTW_CURRENT_FILENAME)</i>.</P>  
}
```

通常使用 DEFINE 语句或 @DTW_ASSIGN() 函数来设置可修改变量值，可修改变量值可让您影响 Net.Data 如何处理宏。例如，要指定是否除去空白，可以使用以下 DEFINE 语句：

```
%DEFINE DTW_REMOVE_WS="YES"
```

请参阅 *Net.Data* 参考中的变量章节，以获取带语法和示例的有效的其它变量列表。

表格处理变量

Net.Data 定义表格处理变量供 REPORT 和 ROW 块使用。使用这些变量从 SQL 查询和函数调用引用值。

表格处理变量具有 Net.Data 确定的预定义值。这些变量允许您引用由正在处理的行、列或字段所调用的函数或者 SQL 查询的结果集中的值。您还可以访问关于正在处理的行数的信息或所有列名列表。

例如，在 Net.Data 处理来自 SQL 查询的结果集时，它为每个当前列名指定变量值，即 N1 赋给第一列，N2 赋给第二列等等。您可以为 Web 页面输出引用当前列名。

在宏中使用处理变量作为变量引用。例如，要检索正在处理的当前列名，可使用：

```
%REPORT {  
<p>Column 1 is <i>$(N1)</i>.</P>  
}
```

表格处理变量还提供关于查询结果的信息。如下例所示，您可以在宏中引用变量 TOTAL_ROWS 来显示在 SQL 查询中返回多少行。

```
Names found: $(TOTAL_ROWS)
```

一些表格处理变量受其它变量或内部函数的影响。例如，TOTAL_ROWS 要求 DTW_SET_TOTAL_ROWS SQL 语言环境变量是激活的，因此在处理来自 SQL 查询或函数调用的结果时，Net.Data 指定 TOTAL_ROWS 的值，这如下例所示：

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"  
...
```

```
Names found: $(TOTAL_ROWS)
```

请参阅*Net.Data* 参考中的变量章节，以获取带语法和示例的有效表格处理变量列表。

报表变量

Net.Data 以缺省报表格式显示宏生成的 Web 页面输出。缺省报表格式使用 <PRE> </PRE> 标记以表格格式进行显示。通过用显示输出的说明来定义 REPORT 块，或通过使用报表变量之一来防止生成缺省报表，可以覆盖缺省报表。

报表变量有助于您定制如何显示 Web 页面输出以及如何与缺省报表和 Net.Data 表格一起使用。必须先定义这些变量，然后才可以在 DEFINE 语句或 @DTW_ASSIGN() 函数中使用它们。

报表变量指定空格、覆盖缺省报表格式、指定相对于缺省报表输出的 HTML 表格，并指定其它显示功能。例如，您可以使用 ALIGN 变量来控制表格处理变量的前导和尾随空格。下例使用 ALIGN 变量用一个空格来分割列表中每个行名，这是由一个查询来返回的。

```
%DEFINE ALIGN="YES"  
...  
<p>Your query was on these columns: $(NLIST)
```

START_ROW_NUM 报表变量让您确定从哪一行开始显示查询的结果。例如，以下变量值指定了 Net.Data 将在第三行开始显示查询的结果。

```
%DEFINE START_ROW_NUM = "3"
```

您还可以确定 `Net.Data` 是否对缺省格式使用 HTML 标记。当 `DTW_HTML_TABLE` 设置为 YES 时，将生成一个 HTML 表格而不是文本格式的表格。

```
%DEFINE DTW_HTML_TABLE="YES"

%FUNCTION(DTW_SQL){
SELECT NAME, ADDRESS FROM $(qTable)
%}
```

请参阅 *Net.Data* 参考中的变量章节，以获取带语法和示例的有效报表变量列表。

语言环境变量

这些变量与语言环境一起使用并影响语言环境处理请求的方式。

通过使用这些变量，您可以执行诸如这样的任务：建立与数据库的连接、启用 NLS 支持以及确定 SQL 语句的执行是否成功。

例如，您可以使用 `SQL_STATE` 变量来访问或者显示从数据库返回的 SQL 状态值。

```
%FUNCTION (DTW_SQL) val1() {
    select * from customer
    %REPORT {
    ...
    %ROW {
    ...
    %}
    SQLSTATE=$(SQL_STATE)
    %}
```

请参阅 *Net.Data* 参考中的变量章节，以获取带语法和示例的有效语言环境变量列表。

Net.Data 函数

`Net.Data` 提供了在应用程序中使用的内部函数，例如字处理函数、字符串处理函数或检索和设置表格变量函数的函数。还可以定义与应用程序一起使用的函数，例如调用外部程序或存储过程的函数。

用户定义的函数

那些为与应用程序一起使用而定义的函数，例如，调用一个外部程序或存储过程。

Net.Data 内部函数

`Net.Data` 为您应用程序中的使用而提供的函数，例如用于处理文字和字符串的函数以及获取和设置表格变量的函数。

这些章节将描述以下主题：

- 『定义函数』
- 第56页的『调用函数』
- 第57页的『调用 `Net.Data` 内部函数』

定义函数

要在宏中定义自己的函数，可使用 `FUNCTION` 块或 `MACRO_FUNCTION` 块：

FUNCTION 块

定义一个子例程，它调用自一个 Net.Data 宏，由语言环境来处理。FUNCTION 块必须包含语言语句或对外部程序的调用。

MACRO_FUNCTION 块

定义一个子例程，它调用自一个 Net.Data 宏，由 Net.Data 而非语言环境来处理。MACRO_FUNCTION 块中可以包含 HTML 块中允许的任何语句。

语法：使用以下语法来定义函数：

FUNCTION 块：

```
%FUNCTION(type) function-name([usage] [datatype] parameter, ...) [RETURNS(return-var)] {  
    executable-statements  
    [report-block]  
    ...  
    [report-block]  
    [message-block]  
%}
```

MACRO_FUNCTION 块：

```
%MACRO_FUNCTION function-name([usage] parameter, ...) [RETURNS(return-var)] {  
    executable-statements  
    report-block  
    ...  
    report-block  
%}
```

其中：

type 标识了初始化文件中配置的语言环境。语言环境调用一个专用语言处理器(处理可执行语句)并提供 Net.Data 和语言处理器之间的标准接口。

function-name

指定 FUNCTION 或 MACRO_FUNCTION 块的名称。函数调用指定 *function-name*，前导以 at (@) 符号。请参阅第56页的『调用函数』，以获取细节。

可以用同一个名称定义多个 FUNCTION 或 MACRO_FUNCTION 块，这样它们就可以被同时处理。每个块都必须具有相同的参数表。当 Net.Data 调用函数时，将以在 Net.Data 宏中定义的顺序执行具有相同名称的所有 FUNCTION 块或有相同名称的所有 MACRO_FUNCTION 块。

usage 指定参数是输入 (IN) 参数、输出 (OUT) 参数还是两种类型 (INOUT)。这个指定指出了是将传送至或接收自 FUNCTION 块、MACRO_FUNCTION 块(或这两者)。在被改为另一种用法类型之前，此用法类型适用于参数表中的所有后继参数。缺省类型是 IN。

datatype

参数的数据类型。有些语言环境期望获得被传递参数的数据类型。例如 SQL 语言环境在调用存储过程时期望了解这些数据类型，“直接调用”语言环境在调用程序时也是如此。请参阅第71页的『第6章 使用语言环境』，以进一步了解您正在使用的语言环境所支持的数据类型。

parameter

指具有局部作用域的变量名称，将用在函数调用上指定的相应变元的值来代替它。用参数的实际值来代替可执行语句或 REPORT 块中的参数引用，例如

`$(parm1)`。另外，参数传送至语言环境，并可以用该语言的语法或作为环境变量来被可执行语句访问。在 `FUNCTION` 或 `MACRO_FUNCTION` 块之外，参数变量引用无效。

return-var

在 `RETURNS` 关键字之后指定此参数来标识特殊的 `OUT` 参数。返回变量的值是在函数块中指定的，该值被返回到宏中调用函数的地方。例如，在句子 `<p>My name is @my_name().` 中，`@my_name()` 将由返回变量的值来替代。如果您没有指定 `RETURNS` 子句，则函数调用的值是：

- `NULL`，如果来自调用向语言环境的返回码是零
- 返回码的值，当回归码非零时。

executable-statements

语言语句的集合，在替换变量和处理函数之后，它们传送至特定语言环境进行处理。*executable-statements* 可包含 `Net.Data` 变量引用和 `Net.Data` 函数调用。

对于 `FUNCTION` 块，在可执行语句传送到语言环境之前，`Net.Data` 用变量值代替所有变量引用，执行所有函数调用并用结果值代替函数调用。每个语言环境处理语句的方式是不同的。关于指定可执行语句或调用可执行程序的更多信息，请参阅第47页的『可执行变量』。

对于 `MACRO_FUNCTION` 块，可执行语句是文本和 `Net.Data` 宏语言结构的组合。在此情况下将不涉及语言环境，因为 `Net.Data` 起语言处理器的作用并处理可执行语句。

report-block

定义一个或多个 `REPORT` 块，以便处理 `FUNCTION` 块或 `MACRO_FUNCTION` 块的输出。请参阅第62页的『报表块』。

message-block

定义 `MESSAGE` 块，它处理 `FUNCTION` 块返回的任何消息。请参阅第55页的『消息块』。

在 `Net.Data` 宏调用函数之前，在其它所有块的外部定义函数。

在函数中使用特殊字符

当匹配 `Net.Data` 语言结构语法的字符在函数块的语言结构节中作为一部分语法上有效的嵌入程序码(例如 `REXX` 或 `Perl`)使用时，它们可能作为 `Net.Data` 语言结构而被误解释，因而导致错误或宏中不可预测的结果。

例如，`Perl` 函数可能使用 `COMMENT` 块定界符 `%{`。运行宏时，`%{` 被作为 `COMMENT` 块的开头来解释。然后 `Net.Data` 查找 `COMMENT` 块的结尾，当它读到函数块结尾时就认为是找到了。`Net.Data` 然后继续查找函数块的结尾，但当找不到时，就发出一个错误。

使用以下方式之一来使用 `COMMENT` 块定界符字符，或使用任何其它 `Net.Data` 特殊字符作为嵌入程序代码的一部分，而不让它们被 `Net.Data` 作为特殊字符来解释：

- 使用 `EXEC` 语句来调用程序代码，而不是将代码放在内部。
- 使用一个变量引用来指定特殊字符。

例如，以下 Perl 函数包含表示一个 COMMENT 块定界符 `%{` 的字符作为 Perl 语言语句的一部分：

```
%FUNCTION(DTW_PERL) func() {  
    ...  
    for $num_words (sort bynumber keys %{ $Rtitles{$num} }) {  
        &make_links($Rtitles{$num}{$num_words});  
    }  
    ...  
}%
```

要保证 Net.Data 将 `%{` 字符作为 Perl 源码而不是作为 Net.Data COMMENT 块定界符，可以用以下方式之一重写函数：

- 使用 `%EXEC` 语句：

```
%FUNCTION(DTW_PERL) func() {  
    %EXEC{ func.prl %}  
}%
```

- 使用一个变量引用来指定 `%{` 字符：

```
%define percent_openbrace = "%{"  
  
%FUNCTION(DTW_PERL) func() {  
    ...  
    for $num_words (sort by number keys $(percent_openbrace) $Rtitles{$num} ) {  
        &make_links($Rtitles{$num}{$num_words});  
    }  
    ...  
}%
```

消息块

MESSAGE 块让您根据函数调用的成功或失败来确定在函数调用之后如何继续下去，并让您为函数的调用程序显示信息。在处理消息时，Net.Data 为每一个对 FUNCTION 块的函数调用设置语言环境变量 RETURN_CODE。在对 MACRO_FUNCTION 块的函数调用上不设置 RETURN_CODE。

一个 MESSAGE 块由一系列消息语句组成，每个消息语句指定一个返回码值、信息正文和一个要进行的操作。Net.Data 参考中语言结构章节中显示了 MESSAGE 块的语法。

MESSAGE 块可具有全局或局部作用域。如果 MESSAGE 块是在 FUNCTION 块中定义的，则它的作用域局部在该 FUNCTION 块中。如果它是在最外层指定的，则 MESSAGE 块是全局作用域，并且对于 Net.Data 宏中执行的所有函数调用都是活动的。如果您定义多个全局 MESSAGE 块，则最后定义的块是活动的。

Net.Data 使用这些规则来处理来自一个函数调用的 RETURN_CODE 变量的值：

1. 在局部 MESSAGE 块检查一个精确匹配；根据指定的来退出或继续。
2. 如果 RETURN_CODE 非 0，则检查局部 MESSAGE 块中的 `+default` 或 `-default`；根据 RETURN_CODE 的符号，按指定的来退出或继续。
3. 如果 RETURN_CODE 非 0，则检查局部 MESSAGE 块中的 `default`；按指定的来退出或继续。
4. 在全局 MESSAGE 块检查一个精确匹配；根据指定的来退出或继续。
5. 如果 RETURN_CODE 非 0，则检查全局 MESSAGE 块中的 `+default` 或 `-default`；根据 RETURN_CODE 的符号，按指定的来退出或继续。

6. 如果 RETURN_CODE 非 0, 则检查检查全局 MESSAGE 块中的 default; 按指定的来退出或继续。
7. 如果 RETURN_CODE 非 0, 则发出 Net.Data 内部缺省消息和出口。

下例显示 Net.Data 宏的一部分, 其中具有一个全局 MESSAGE 块和一个函数的 MESSAGE 块:

```
%{ 全局消息块 %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    +default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %} : continue
%}

%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
    %EXEC { my_command.mbr %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    -default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %} : exit
%}
```

如果 *my_function()* 返回 RETURN_CODE 值为 50, Net.Data 将以此顺序处理错误:

1. 在局部 MESSAGE 块中检查确切匹配。
2. 在局部 MESSAGE 块中检查 +default。
3. 在局部 MESSAGE 块中检查 default。
4. 在全局 MESSAGE 块中检查确切匹配。
5. 在全局 MESSAGE 块中检查 +default。

当 Net.Data 找到一个匹配时, 它向 Web 浏览器发送信息正文, 并检查请求的操作。

当您指定了 continue 之后, Net.Data 继续处理 Net.Data 宏, 然后才打印信息正文。例如, 一个宏调用 *my_functions()* 5 次并在用 MESSAGE 块处理期间发现错误 100, 则程序的输出是这样的:

```
.
.
.
11 May 1997                $245.45
13 May 1997                $623.23
19 May 1997                $ 83.02
return code 100 message
22 May 1997                $ 42.67

Total:                     $994.37
```

调用函数

使用一个 Net.Data 函数调用语句来调用用户定义的函数和内部函数。使用后面跟有函数名或宏函数名的 at (@) 字符:

```
@function_name([ argument,... ])
```

function_name

这是要调用的函数或宏函数的名称。除非是内部函数，否则必须在 Net.Data 宏中已定义函数。

argument

这是变量、引用字符串、变量引用或函数调用的名称。函数调用上的变元与函数或宏函数参数列表上的参数相匹配。在处理函数或宏函数时，每个参数都被赋予其相应变元的值。变元与对应的参数必须具有相同数目和类型。

作为变量的引用字符串可以包含变量引用和函数调用。

例 1: 用文本字符串参数进行函数调用

```
@myFunction("abc")
```

例 2: 用变量和函数调用参数进行函数调用

```
@myFunction(myvar, @DTW_rADD("2","3"))
```

例 3: 用包含变量引用和函数调用的文本字符串参数进行函数调用

```
@myFunction("abc$(myvar)def@DTW_rADD("2","3")ghi")
```

调用 Net.Data 内部函数

Net.Data 提供了大量的内部函数来简化 Web 页面的开发。这些函数已经由 Net.Data 定义好了，因此不需要再对它们进行定义。您可以象调用其它函数一样调用这些函数。

图7显示了 Net.Data 内部函数和宏是如何相互作用的。

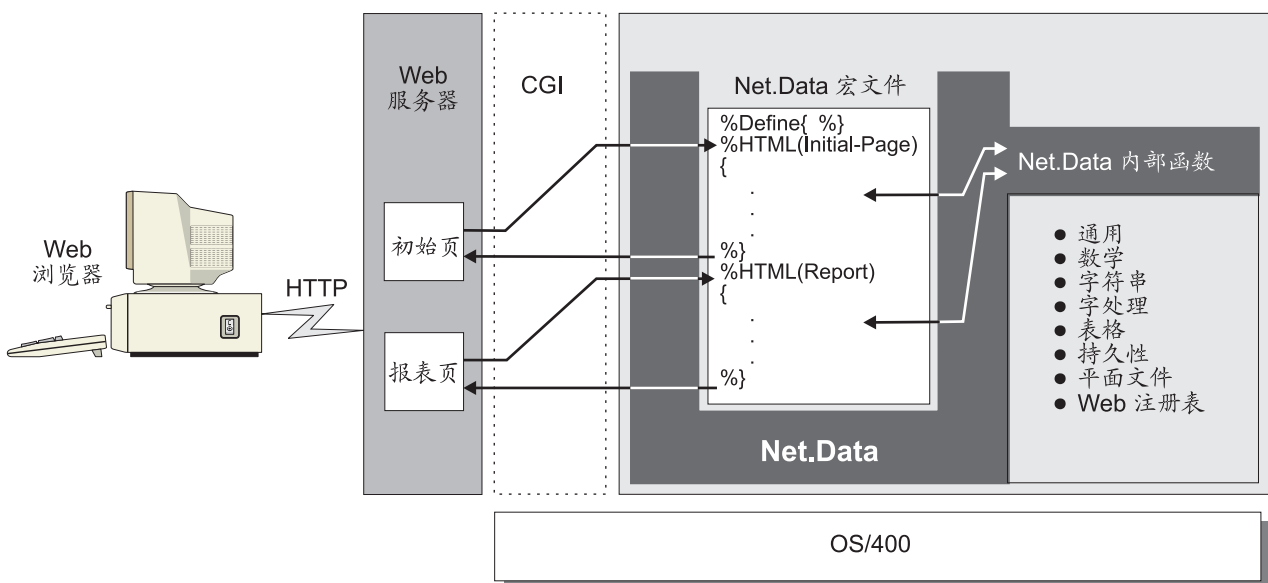


图 7. Net.Data 内部函数

根据前缀的不同，内部函数可以三种方式返回它们的结果：

- **DTW_**、**DTWF_** 和 **DTWR_**: 调用结果在一个输出参数中返回，或者不返回结果。(DTWF_ 是用于平面文件函数的前缀。DTWR_ 是用于 Web 注册表函数的前缀。)
- **DTW_r**、**DTWF_r** 和 **DTWR_r**: 函数调用的结果将替换宏中的函数调用，这就和指定了 RETURNS 关键字的用户自定义函数中用 RETURNS 关键字的值替换函数调用是相同的。
- **DTW_m**: 在传递给函数的每个参数中返回多个结果。

有些内部函数并不具有每一类型。要确定某个特定内部函数具有的类型，请参阅 *Net.Data* 参考中的 *Net.Data* 内部函数章节。

以下章节提供了 *Net.Data* 内部函数的一个高级概述。使用这些函数可以执行通用、数学、字符串、字处理或表格处理功能。另外，您可以将持久性函数用于事务处理。请参阅 *Net.Data* 参考以获取每个函数的语法说明和示例。这其中的某些函数需要变量在使用之前先进行设置，或者必须在特定的上下文中使用。并非所有的操作系统都支持每个内部函数。请参阅 *Net.Data* 参考以确定您的操作系统支持哪些函数。

- 『通用函数』
- 第59页的『数学函数』
- 第59页的『字符串函数』
- 第59页的『字处理函数』
- 第60页的『表格函数』
- 第60页的『平面文件函数』
- 第60页的『Web 注册表函数』
- 第60页的『持久性函数』

通用函数

这个函数集合通过改变数据或访问系统服务来帮助您开发 Web 页面。您可以用它们来发送邮件、处理 HTTP cookie、生成 HTML 转换代码，并从系统中获取其它有用信息。

例如，要指定 *Net.Data* 在发生某个特定的情况时应退出宏，而不处理剩下的宏，可以使用 **DTW_EXIT** 函数：

```
%HTML(cache_example) {

<html>
<head>
<title>This is the page title</title>
</head>
<body>
<center>
<h3>This is the Main Heading</h3>
<!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
<! Joe Smith sees a very short page                !>
<!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
%IF (customer == "Joe Smith")
</body>
</html>

@DTW_EXIT()

%ENDIF
```

```
...
</body>
</html>
%}
```

另一个有用的函数是 `DTW URLESCSEQ` 函数，它用转换值替换URL 中不允许的字符。例如，如果输入变量 `string1` 等于 "Guys & Dolls"，那么 `DTW URLESCSEQ` 将为输出变量赋值 "Guys%20%26%20Dolls"。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效通用函数列表。

数学函数

这些函数执行数学运算，使您能够计算或改变数字数据。除了标准的数学运算以外，您还可以执行按模除法、指定结果精度并使用科学记数法。

例如，函数 `DTW_POWER` 将它第一个参数的值提高为第二个参数的平方并返回结果，如下面的例子中所示：

```
@DTW_POWER("2", "-3", result)
```

`DTW_POWER` 在变量 `result` 中返回 ".125"

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效数学函数列表。

字符串函数

这些函数可以让您处理字符串中的字符。您可以更改字符串的大小写、插入或删除字符、给另一个变量指定字符串值、增加其它有用的函数。

例如，您可以使用 `DTW_ASSIGN` 将一个输入变量的值赋给输出变量。同样可以使用这个函数在宏中更改一个变量。在下面的例子中，变量 `RC` 被赋值为 0。

```
@DTW_ASSIGN(RC, "0")
```

其它字符串函数包括 `DTW_CONCAT` (用于连接字符串)、`DTW_INSERT` (在特定的位置插入字符串)以及许多其它字符串处理函数。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效字符串函数列表。

字处理函数

这些函数可以让您处理字符串中的单词。这些函数大部分和字符串函数以类似的方式作用，但它们是对整个单词进行作用。例如，它们可以让您计数一个字符串中的单词个数、删除单词、在字符串中搜索某个单词。

例如，使用 `DTW_DELWORD` 来从一个字符串中删除指定数目的单词：

```
@DTW_DELWORD("Now is the time", "2", "2", result)
```

`DTW_DELWORD` 返回字符串 "Now time"。

其它字处理函数包括 `DTW_WORDLENGTH` (返回单词中的字符个数)、`DTW_WORDPOS` (返回一个单词在字符串中的位置)。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效字处理函数列表。

表格函数

您可以使用这些函数来生成报表或表格(这些报表或表格使用 *Net.Data* 表格变量中的数据)。您还可以使用这些函数来创建 *Net.Data* 表格，处理和检索那些表格中的值。表格变量中包含了一系列值以及相关的列名。它们提供了一种便利的方式将一组值传递给一个函数。

例如，DTW_TB_APPENDROW 在表格后追加一行。在下面的例子中，*Net.Data* 在表格 myTable 后面追加了十行：

```
@DTW_TB_APPENDROW(myTable, "10")
```

另外，DTW_TB_DUMP 返回一个宏表格变量的内容(包括在 <PRE></PRE> 标记中)，表格中的每一行显示在不同的行中。而 DTW_TB_CHECKBOX 从宏表格变量返回一个多个 HTML 校验框输入标记。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效表格函数列表。

平面文件函数

使用平面文件接口 (FFI) 可以打开、读取和处理平面文件源(文本文件)中的数据，也可以将数据存储到平面文件中。

例如，DTWF_APPEND 将一个表格变量的内容写入文件末尾，而 DTWF_DELETE 从文件中删除记录。

另外，FFI 函数允许使用 DTWF_CLOSE 和 DTWF_OPEN 来进行文件锁定。DTWF_OPEN 锁定一个文件，这样其它请求就无法读取或更新该文件。DTWF_CLOSE 在 *Net.Data* 完成处理之后释放文件，从而允许其它请求访问该文件。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效 FFI 函数列表。

Web 注册表函数

使用 Web 注册表函数来维护注册表及其包含的条目。Web 注册表是一个文件，由 *Net.Data* 维护此文件的一个关键字，允许您方便地添加、检索和删除其中的条目。

例如，DTWR_ADDENTRY 添加条目，而 DTWR_DELENTY 删除条目。DTWR_LISTSUB 在一个 OUT 表格参数中返回有关注册表条目的信息，而 DTWR_UPDATEENTRY 用一个新值替换指定注册表条目的现有值。

请参阅*Net.Data* 参考中的内部函数章节，以获取带语法和示例的有效 Web 注册表函数列表。

持久性函数

持久性宏函数支持在 *Net.Data* 中进行事务处理，这是通过帮助您定义哪些宏块在单个事务中是持久性的来实现的。使用这些函数来定义一个事务的开头与结尾 (HTML 块在该事务中是持久性的)、变量在事务中的作用域以及在事务中提交还是撤消更改。

例如, DTW_ACCEPT 标识一个事务的事务句柄, 而 DTW_TERMINATE 标识该事务中最后一个 HTML 块。DTW_RTVHANDLE 为事务中的块生成一个唯一的事务句柄。在一个事务中, 您可以使用 DTW_COMMIT 和 DTW_ROLLBACK 来初启提交和撤消。

请参阅第95页的『第7章 具有持久性宏的事务管理』以获取更多信息。另外请参阅 *Net.Data* 参考中的内部函数章节, 以获取带语法和示例的有效的持久性函数列表。

在宏中生成 Web 页面

Net.Data 让您很容易地在应用程序用户浏览器中表示标准 Web 页。以下各节描述宏的 HTML 和 REPORT 块, 并显示如何在 Net.Data 宏中格式化 Web 页面。请参阅 *Net.Data* 参考中的语言结构章节, 以获取有关这些块的语法信息。

HTML 块

Net.Data 宏包含 HTML 块, 它在 Web 浏览器上生成文本呈现语句, 例如 HTML。在宏中, 必须指定至少一个 HTML 块, 多则不限。每个 HTML 块在浏览器上生成一个 Web 页面。Net.Data 在每次被调用时仅处理一个 HTML 块。要创建一个由许多 Web 页面组成的应用程序, 您可以多次调用 Net.Data, 以便处理那些使用标准的游历技术(例如, 链和表)的 HTML 块。

任何有效的正文呈现语句, 例如 HTML 或 JavaScript, 都可以出现在 HTML 块中。另外, 在 HTML 块中还可以使用 INCLUDE 语句、函数调用和变量引用。下例显示 Net.Data 宏中的 HTML 块的一般用法:

```
%DEFINE DATABASE="MNS96"

%HTML(INPUT) {
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/equip1st.d2w/report">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hardware" value="MON" checked>Monitors
<dd><input type="radio" name="hardware" value="PNT">Pointing devices
<dd><input type="radio" name="hardware" value="PRT">Printers
<dd><input type="radio" name="hardware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

%FUNCTION(DTW_SQL) myQuery() {
SELECT MODNO, COST, DESCRIP FROM EQPTABLE WHERE TYPE=$(hardware)
%REPORT{
<B>Here is the list you requested:</B><BR>
%ROW{
<HR>
$(N1): $(V1)      $(N2): $(V2)
<P>
$(V3)
%}
%}
%}
```



```
%HTML(REPORT){
  @myQuery()
%}
```

您可以如下例所示来从 HTML 链接中调用 Net.Data 宏:

```
<a href="http://www.ibm.com/cgi-bin/db2www/equiplst.d2w/input">
  List of hardware</a>
```

当应用程序用户单击此链接时, Web 浏览器调用 Net.Data, Net.Data 语法分析宏。当 Net.Data 开始处理在调用上指定的 HTML 块时(在此情况下为 HTML(INPUT) 块), 它开始处理其中的文本。Net.Data 对于不能识别为 Net.Data 宏语言结构的任何东西, 都发送到浏览器显示。

在用户作出选择并按了 Submit 按钮之后, Net.Data 运行 HTML FORM 元素的 ACTION 部分, 这指出了对 Net.Data 宏的 HTML(OUTPUT) 块的调用。然后, Net.Data 象处理 HTML(INPUT) 块那样处理 HTML(OUTPUT) 块。

Net.Data 然后处理 myQuery() 函数调用, 该函数调用依次调用 SQL FUNCTION 块。用在输入表中返回的值代替 SQL 语句中引用的 \$(hardware) 变量之后, Net.Data 运行查询。在此点, Net.Data 继续处理报表, 根据 REPORT 块中指定的文本呈现语句来显示查询的结果。

Net.Data 完成 REPORT 块的处理之后, 返回至 HTML(OUTPUT) 块, 并结束处理。

报表块

使用 REPORT 块语言结构来格式化并显示来自 FUNCTION 块的数据输出。这个输出通常是表格数据, 尽管可以指定文本、宏变量引用和函数调用的任何有效组合。通常可以任选地在 REPORT 块上指定表名。如果没有指定表格名称, Net.Data 将使用 FUNCTION 参数列表中第一个输出表格的表格数据。

REPORT 块具有三部分, 每部分都是可选的:

- 首部信息, 包含在表格行数据之前显示一次的文本。
- ROW 块, 包含在结果表格的每行上显示一次的文本和表格变量。
- 注脚信息, 包含在表格行数据之后显示一次的文本。

例子:

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR>
  <TD>Name</TD>
  <TD>Location</TD></TR>
%ROW{
<TR>
<TD>
  <a href="/cgi-bin/db2www/name.d2w/details?name=$(V1)&location;=$(V2)">$(V1)</a>
</TD>
<TD>$(V2)</TD>
</TR>
%}
</TABLE>
%}
```


REPORT 块的准则

在创建 REPORT 块时，请使用以下准则：

- 要避免显示来自 ROW 块的任何表格输出，可让 ROW 块为空或整个省略它。
- 可以在 REPORT 块中使用 Net.Data 提供的变量来访问 Net.Data 宏结果表格中的数据。第51页的『表格处理变量』中描述了这些变量。有关的附加细节，请参阅 *Net.Data* 参考中的“报表变量”一节。
- 要提供首部和注脚信息，必须在 ROW 块之前和之后提供文本。Net.Data 将它在 ROW 块之前发现的所有内容作为首部信息来对待。Net.Data 将它在 ROW 块之后发现的所有内容作为注脚信息来对待。象 HTML 块一样，Net.Data 将首部、ROW 和注脚块中未识别为宏语言结构的任何东西作为正文呈现语句对待，并将这些语句发送给浏览器。
- 可以在 REPORT 块中调用函数、引用变量。
- 要让 Net.Data 打印使用预格式化文本的缺省报表，就不要在宏中包含 REPORT 块。以下例子显示缺省报表格式：

```
SHIPDATE   | RECDATE    | SHIPNO    |
-----|-----|-----|
25/05/1997 | 30/05/1997 | 1495194B |
-----|-----|-----|
25/05/1997 | 28/05/1997 | 2942821G |
-----|-----|-----|
```

- 要使用 HTML 标记来代替预格式化文本，可将 DTW_HTML_TABLE 设置为 YES。
- 要禁用缺省报表的打印，可将 DTW_DEFAULT_REPORT 设置为 NO，或指定一个空的 REPORT 块。例如：

```
%REPORT{%}
```

例：定制报表

下例显示如何使用特殊变量和 HTML 标记来定制报表格式。它显示来自表格 CustomerTbl 的姓名、电话号码和传真号码：

```
%DEFINE SET_TOTAL_ROWS="YES"
...
%FUNCTION(DTW_SQL) custlist() {
    SELECT Name, Phone, Fax FROM CustomerTbl
%REPORT{
<I>Phone Query Results:</I>
<BR>
=====
<BR>
%ROW{
    Name: <B>$(V1)</B>
<BR>
    Phone: $(V2)
<BR>
    Fax: $(V3)
<BR>
-----
<BR>
    %}
    Total records retrieved: $(TOTAL_ROWS)
    %}
%}
```

Web 浏览器中的结果报表如下所示：

```

Phone Query Results:
=====
Name: Doen, David
Phone: 422-245-1293
Fax: 422-245-7383
-----
Name: Ramirez, Paolo
Phone: 955-768-3489
Fax: 955-768-3974
-----
Name: Wu, Jianli
Phone: 525-472-1234
Fax: 525-472-1234
-----
Total records retrieved: 3

```

Net.Data生成报表是通过:

1. 在报表的开头打印一次 *Phone Query Results:*。此文本和分隔线是 REPORT 块的页眉部分。
2. 对于检索到的每一行, 分别用 Name、Phone 和 Fax 的值替换变量 V1、V2 和 V3。
3. 在报表结尾打印一次字符串 *Total records retrieved:* 以及 TOTAL_ROWS 的值。(此文本是 REPORT 块的注脚部分。)

多个 REPORT 块

在一个 FUNCTION 或 MACRO FUNCTION 块中可以指定多个 REPORT 块, 从而用一次函数调用生成多个报表。

通常, 您将一起使用具有 DTW_SQL 语言环境的多个 REPORT 块和调用存储过程的函数, 该存储过程返回多个结果集(请参阅第86页的『存储过程』)。当然, 多个 REPORT 块可以与任何语言环境一起使用来生成多个报表。

要使用多个 REPORT 块, 可以在 FUNCTION 参数列表中放置一个 Net.Data 表格变量。如果存储过程返回的结果集比指定的 REPORT 块的个数多, 并且 Net.Data 内部函数 DTW_DEFAULT_REPORT = "MULTIPLE", 则将为每个不与报表块关联的表格生成缺省报表。如果没有指定报表块, 并且 DTW_DEFAULT_REPORT = "YES", 则仅生成一个缺省报表。请注意对于 SQL 语言环境来说, DTW_DEFAULT_REPORT 值为 "YES" 等价于值 "MULTIPLE"。

例子: 以下例子演示了使用多个报表块的方式。

要使用缺省的报表格式来显示多个报表:

例 1: DTW_SQL 语言环境

```

%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%FUNCTION (dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc %}

```

在此例中, 存储过程 myproc 返回两个结果集, 分别放在 table1 和 table2 中。因为没有指定 REPORT 块, 因此对于这两个表显示缺省报表, 首先显示 table1, 然后显示 table2。

例 2: MACRO_FUNCTION 块。在此例中, 两个表格被传送到 MACRO_FUNCTION 块中。在指定 DTW_DEFAULT_REPORT="MULTIPLE" 的情况下, Net.Data 将对这两个表格生成报表。

```

%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%MACRO_FUNCTION multReport (INOUT tablename1, tablename2) {
%}

```

在此例中，两个表格被传送到 `MACRO_FUNCTION multReport`。再一次，`Net.Data` 根据两个表格出现在 `MACRO_FUNCTION` 块参数列表中的顺序来显示它们的缺省报表，先是 `table1`，然后是 `table2`。

例 3: DTW_REXX 语言环境

```

%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (dtw_rexx) multReport (INOUT table1, table2) {
    SAY 'Generating multiple default reports...<BR>'
%}

```

在此例中，两个表格被传送到 `REXX` 函数 `multReport`。由于指定了 `DTW_DEFAULT_REPORT="YES"`，`Net.Data` 仅对第一个表格显示缺省报表。

要通过对显示处理指定 *REPORT* 块来显示多个报表:

例 1: 已命名的 REPORT 块

```

%FUNCTION(dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc (table1, table2)

    %REPORT(table2) {
        ...
        %ROW { .... }
        ...
    }

    %REPORT(table1) {
        ...
        %row { .... }
        ...
    }
%}

```

在此例中，对于在 `FUNCTION` 块参数列表中传递的两个表格都指定了 `REPORT` 块。这些表格是以它们在 `REPORT` 块中指定的顺序显示的，先是 `table2`，然后是 `table1`。通过在 `REPORT` 块中指定表格名，您可以控制报表显示的顺序。

例 2: 未命名的 REPORT 块

```

%FUNCTION(dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc

    %REPORT {
        ...
        %ROW { .... }
        ...
    }
    %REPORT {
        ...
        %ROW { .... }
        ...
    }
%}

```

在此例中，对于在 FUNCTION 块参数列表中传递的两个表格都指定了 REPORT 块。因为 REPORT 块中没有指定表格名称，因此将根据这两个表格从存储过程返回的顺序显示它们的报表。

要使用缺省报表和 REPORT 块的组合来显示多个报表：

例子： 缺省报表和 REPORT 块的组合

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%FUNCTION(dtw_system) editTables (INOUT table1, table2, table3) {
  %EXEC{ /qsys.lib/mylib.lib/mypgm.pgm %}
  %REPORT(table2) {
    ...
    %ROW { .... %}
    ...
  %}
%}
```

在此例中，只有指定了一个 REPORT 块，并且因为它指定了表格名 table2，因此使用这个表格来显示报表。因为指定的 REPORT 块比从存储过程返回的结果集个数少，因此对剩余的结果集显示缺省的报表：先是 table1 的缺省报表；然后是 table3 的缺省报表。

多个 REPORT 块的准则和限制： 在 FUNCTION 或 MACRO_FUNCTION 块中指定多个 REPORT 块时，请使用以下准则和限制。

准则：

- 对于每个结果集或表格名称，可以指定一个或多个 REPORT 块。对 REPORT 块指定的 name 必须与 FUNCTION 块参数列表中相应的结果集名称或表格名称参数相匹配。
- 以您希望为多个表格进行处理的顺序来为它们指定 REPORT 块。
- 当没有为表格指定 REPORT 块时，要指定缺省处理，可定义 DTW_DEFAULT_REPORT = "MULTIPLE"。当 Net.Data 构建 Web 页面时，当它为具有 REPORT 块的表格显示了报表之后，将为表格显示缺省报表。请注意设置 DTW_DEFAULT_REPORT = "YES" 将导致在没有指定 REPORT 块的情况下仅对一个表格生成缺省报表。异常情况出现在 SQL 语言环境中，其中值 YES 将形成与 MULTIPLE 相同的处理。
- 要防止 Net.Data 显示不具有 REPORT 块的表格，必须设置 DTW_DEFAULT_REPORT = "NO"。
- 当 DTW_SAVE_TABLE_IN 变量与返回多个表格的函数一起使用时，从函数返回的第一个表格被指定为 DTW_SAVE_TABLE_IN 表格。
- 多个报表块可以与任何语言环境一起使用。

限制：

- 函数中所有报表变量的值都适用于该函数中所有 REPORT 块。您不能修改单个 REPORT 块的报表变量。
- MESSAGE 块必须位于一个 REPORT 块列表的之前或之后，而不能在 REPORT 块的中间。
- 表格变量被传递到函数之前，必须先在 TABLE 语句中定义。
- 如果第一个报表块指定了一个表格名称，那么所有的报表块都必须指定表格名称。
- 如果第一个报表块没有指定表格名称，那么其它报表块也不能指定表格名称。

宏中的条件逻辑和循环

Net.Data 让您使用 IF 和 WHILE 块来在 Net.Data 宏中结合条件逻辑和循环。

IF 和 WHILE 块使用可以帮助您测试一个或多个条件的条件列表，然后根据条件测试的结果执行一个语句块。条件列表包含逻辑运算符(例如 = 和 <+) 和项，项是由引用字符串、变量、变量引用和函数调用组成的。引用字符串也可以包含变量引用和函数调用。可以嵌套条件列表。

以下章节描述条件逻辑和循环:

- 『条件逻辑: IF 块』
- 第69页的『循环结构: WHILE 块』

条件逻辑: IF 块

将 IF 块用于 Net.Data 宏中的条件处理。在大多数高级语言中 IF 块类似于 IF 语句，因为 IF 块提供测试一个或多个条件的能力，然后基于条件测试的结果执行一个语句块。

您可以在宏中的几乎任何地方指定 IF 块并可以嵌套它们。 *Net.Data* 参考中的语言结构章中显示了 IF 块的语法。

IF 块的规则: IF 块的语法规则是由该块在宏中的位置确定的。IF 块中语句的可执行块允许的元素取决于 IF 块自身的位置。

- 包含 IF 块的块中的任何有效元素在该 IF 块中也有效。 例如，如果您在一个 HTML 块中指定了一个 IF 块，则 HTML 块中允许的任何元素在 IF 块中也是允许的，例如 INCLUDE 语句和 WHILE 块。

```
%HTML 块
...
%IF 块
...
%INCLUDE
...
%WHILE
...
%ENDIF
%}
```

- 类似地，如果您在 Net.Data 宏声明部分中的任何其它块之外指定 IF 块，则在 IF 语句中只允许那些在任何其它块之外也被允许的元素(例如 DEFINE 块或 FUNCTION 块)。

```
%IF
...
%DEFINE
...
%FUNCTION
...
%ENDIF
```

- 如果 IF 块嵌套在一个 IF 块内，而后者在声明部分中任何其它块的外部，则它可以使用外部块可以使用的任何元素。如果 IF 块嵌套在另一个嵌套在某个 IF 块的块中，则它遵循它所处的那个块的语法规则。

例如，一个嵌套的 IF 块必须遵循在它处于一个 HTML 块中时使用的规则。

```

%IF
...
%HTML {
...
%IF
...
%ENDIF
%}
...
%ENDIF

```

例外：不要在 IF 块中指定 ROW 块。

IF 块字符串比较

Net.Data 根据组成条件的项目的内容，用两种方式中的一种来处理 IF 块条件列表。缺省操作是将所有项目作为字符串对待，并如条件中所指定的那样执行字符串比较。当然，如果比较是在两个代表整数的字符串之间进行的，那么这个比较也是数字的。如果字符串中仅包含数字，则 Net.Data 假定它是数字的，任选地前导以一个 '+' 或 '-' 字符。字符串不能包含任何非数字字符，'+' 或 '-' 除外。Net.Data 不支持非整数的数值比较。

有效整数字符串的例子：

```

+1234567890
-47
000812
92000

```

无效整数字符串的例子：

```

- 20      (包含空格字符)
234,000   (包含一个逗号)
57.987    (包含一个十进制小数点)

```

Net.Data 在执行 IF 块的时候求解 IF 条件，此时间可能与 Net.Data 初始读块的时间不同。例如，如果您在 REPORT 块中指定一个 IF 块，Net.Data 在读取包含 REPORT 块的 FUNCTION 块定义时不估计与 IF 块相关联的条件列表，而是在调用和执行它时进行。对于 IF 块的条件列表部分和要执行的语句块，都是这样的。

IF 块的示例： 一个在其它块中包含 IF 块的宏

```

%{ This macro is called from another macro, passing the operating system
   and version variables in the form data.
%}

%IF (platform == "AS400")
  %IF (version == "V3R2")
    %INCLUDE "as400v3r2_def.hti"
  %ELIF (version == "V3R7")
    %INCLUDE "as400v3r7_def.hti"
  %ELIF (version == "V4R1")
    %INCLUDE "as400v4r1_def.hti"
  %ENDIF
%ELSE
  %INCLUDE "default_def.hti"
%ENDIF

%MACRO_FUNCTION numericCompare(IN term1, term2, OUT result) {
%IF (term1 < term2)
  @dtw_assign(result, "-1")
%ELIF (term1 > term2)
  @dtw_assign(result, "1")
%ELSE

```

```

    @dtw_assign(result, "0")
%ENDIF
%}

%HTML (report){
    %WHILE (a < "10") {
        outer while loop #$(a)<BR>
        %IF (@dtw_rdivrem(a,"2") == "0")
            this is an even number loop<BR>
        %ENDIF
        @DTW_ADD(a, "1", a)
    %}
%}

```

循环结构: **WHILE** 块

在 *Net.Data* 宏中使用 **WHILE** 块来执行循环。类似于 **IF** 块, **WHILE** 块也提供测试一个或多个条件的能力, 然后基于条件测试的结果执行一个语句块。与 **IF** 不同的是, 基于条件测试的结果, 语句块可被执行多次。

可以在 **HTML** 块、**REPORT** 块、**ROW** 块 **MACRO_FUNCTION** 块和 **IF** 块中指定 **WHILE** 块, 并可以嵌套它们。 *Net.Data* 参考中语言结构章节中显示了 **WHILE** 块的语法。

Net.Data 处理 **WHILE** 块的方式与处理 **IF** 块的方式精确相同, 只是在每次执行该块之后重新计算条件。而且与任何条件循环结构相同, 如果条件编码不正确的话, 就有可能进入死循环。

例子: 具有 **WHILE** 块的宏

```

%DEFINE loopCounter = "1"

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
    %{ generate table tag and column headings %}
    %IF (loopCounter == "1")
        <TABLE BORDER>
        <TR>
        <TH>Item #
        <TH>说明
    %ENDIF

    %{ generate individual rows %}
    <TR>
    <TD>$(loopCounter)
    <TD>@getDescription(loopCounter)

    %{ generate end table tag %}
    %IF (loopCounter == "100")
%ENDIF

    %{ increment loop counter %}
    @DTW_ADD(loopCounter, "1", loopCounter)
}
}

```

```
%}  
%}
```


第6章 使用语言环境

Net.Data 提供了用于访问数据源以及执行包含商业逻辑的应用程序的语言环境。例如，SQL 语言环境可以让您将 SQL 语句传递到一个 DB2 数据库，REXX 语言环境可以让您调用 REXX 程序。您还可以使用 SYSTEM 语言环境来执行一个程序或发出一条命令。

使用了 Net.Data，您就能够以一种可插入的方式来添加用户编写的语言环境。每个用户编写的语言环境都必须支持 Net.Data 定义的一系列接口，必须作为服务程序实现。有关如何创建用户编写的语言环境的完整细节，请参阅 *Net.Data 语言环境接口参考*。

图8显示了 Web 服务器、Net.Data 以及 Net.Data 语言环境之间的关系。

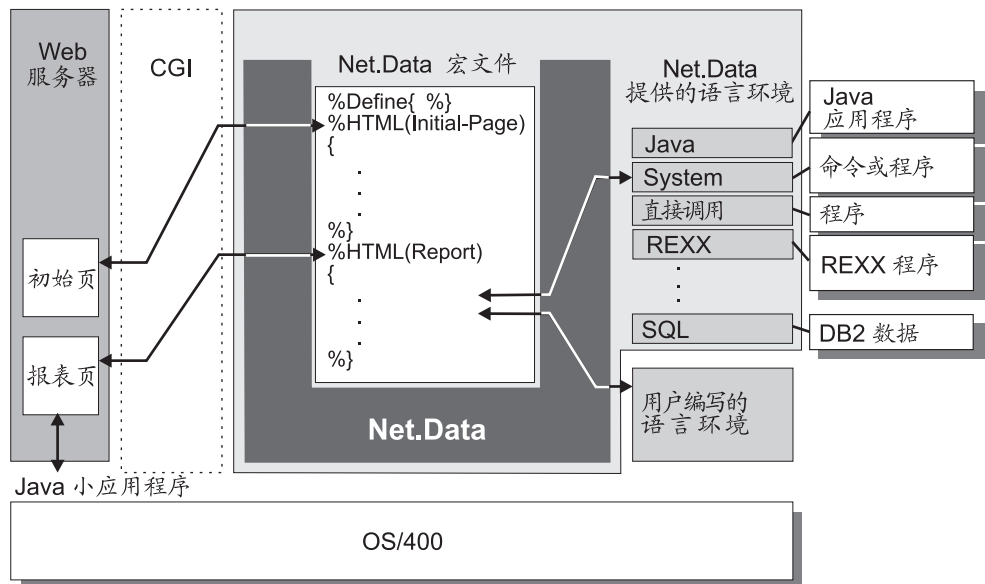


图 8. Net.Data 语言环境

以下章节描述 Net.Data 语言环境以及如何在宏中使用这些语言环境:

- 第72页的『Net.Data 提供的语言环境概述』
- 第72页的『调用语言环境』
- 第73页的『直接调用语言环境』
- 第76页的『Java 应用程序语言环境』
- 第77页的『REXX 语言环境』
- 第80页的『SQL 语言环境』
- 第92页的『System 语言环境』

有关在使用语言环境时改进性能的信息，请参阅第105页的『优化语言环境』。

Net.Data 提供的语言环境概述

Net.Data 提供了让您访问数据和应用程序编程资源的语言环境。

表3对每个语言环境作了一个简短的描述。

表 3. *Net.Data* 语言环境

语言环境	环境名称	说明
直接调用	DTW_DIRECTCALL	“直接调用”语言环境支持对那些用高级程序设计语言(例如 RPG、COBOL 和 C/C++) 编写的外部程序的调用。
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。
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。
System	DTW_SYSTEM	System 语言环境支持执行命令和调用外部程序。

调用语言环境

要调用一个语言环境:

- 使用 FUNCTION 语句来定义一个调用语言环境的函数。
- 使用一个对语言环境的函数调用。

例如:

```
%FUNCTION(DTW_SQL) custinfo() {
  select customer, custno from customer.data
  %}
...
%HTML(REPORT){
  @custinfo()
  %}
```

处理错误条件

当在语言环境函数中检测到错误时, 语言环境将用一个错误代码来设置 Net.Data RETURN_CODE 变量。

可以使用以下资源来处理错误条件:

- Net.Data 提供的语言环境返回*Net.Data* 消息和代码参考中所述的错误代码。
- 数据库语言环境(例如, SQL 语言环境)将 RETURN_CODE 设置为数据库管理系统 (DBMS) 返回的错误代码, 称为 SQLCODE。请参阅针对您的 DBMS 的消息和代码文档, 以进一步了解您的 DBMS 所使用的 SQLCODE。

安全性

请确保运行 Net.Data 的用户 ID 有适当的权限访问那些语言环境语句的目标可能引用的任何对象。例如, SQL 语言环境运行 SQL 语句, 而 SQL 语句访问数据库文件,

因此运行 Net.Data 的用户 ID 必须对数据库文件具有权限。

直接调用语言环境

“直接调用”语言环境允许您调用那些用高级程序设计语言(例如 C、RPG、COBOL 和 CL)编写的程序。参数可以被传递到程序,参数值也可以从程序接收,从而方便地实现了现有程序与 Net.Data 的集成,并允许用户使用现有的程序设计技巧来对复杂的商业逻辑编程。

调用程序

要调用一个程序,需要定义一个使用“直接调用”(DTW_DIRECTCALL)语言环境的函数,它包含一个路径,该路径指向要在 EXEC 语句中调用的程序。例如:

```
%function(DTW_DIRECTCALL) dc1() {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
%}
```

如果使用 EXEC_PATH 配置变量来定义至包含程序的目录的路径,则可以缩短至该程序的路径。请参阅第14页的『EXEC_PATH』,以学习如何定义 EXEC_PATH 配置变量。

受支持的语言环境变量

“直接调用”语言环境支持 DTW_PAD_PGM_PARMS 变量,它指出要传递给程序的参数是否用空格填充至指定的精度。请参阅*Net.Data* 参考,以获取对此变量的描述、此变量的语法和示例。请参阅『将参数传送到程序』,以获取有关向程序传递参数的更多信息。

将参数传送到程序

通过在函数定义中指定参数的数据类型以及要传递的参数是仅用于输入的参数 (IN)、仅用于输出的参数 (OUT) 还是输入/输出参数 (INOUT) 来将参数传递给程序。例如:

```
%function(DTW_DIRECTCALL) dc2(IN CHAR(3) p1,  
                                INOUT INTEGER p2,  
                                OUT DECIMAL(7,2) p3) {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
%}
```

在上面的例子中,“直接调用”语言环境向程序 MYPGM 传递了三个参数:一个字符变量、一个整数和一个压缩十进制变量。最多可以向被调用的程序传递 50 个参数。只有用数据类型指定的参数才会被传递到程序。“直接调用”语言环境将对应于该参数的字符串转换为数据类型的内部表示。然后,语言环境以函数定义中指定的顺序将指向变量内部表示的指针传递给被调用的程序。

因为指向变量的指针被传递给了程序,因此程序就可以更改该变量的值。当然,只有那些被程序更改过的 OUT 或 INOUT 变量才会反馈到调用语言环境的宏中。

受支持的数据类型

表4列出了“直接调用”语言环境所支持的数据类型。并非所有的数据类型都受到每一种高级语言的支持。

表 4. 直接调用数据类型

数据类型	用法注意
CHAR(<i>n</i>) CHARACTER(<i>n</i>) CHARACTER	一个字符串。如果指定 <i>n</i> ，则必须大于零。如果没有指定字符串，则将假定它是一个字符。因为从“直接调用”语言环境传递过来的所有字符串都是以 NULL 结束的，所以语言环境将分配 <i>n</i> +1 个字节(其中 1 个字节用于 NULL 结束符)。超过 <i>n</i> 的字符串都将被截断。
VARCHAR(<i>n</i>)	一个可变长度的字符串，其中 <i>n</i> 大于等于 0 且小于等于 32740。这个字符串是以 NULL 结束的，语言环境共分配 <i>n</i> +2+1 个字节 (2 个字节用于存储字符串长度，1 个字节用于 NULL 结束符)。超过 <i>n</i> 的字符串都将被截断。字符串的前两个字节包含了字符串的长度(二进制值)。如果该参数定义为 OUT (仅用于输出)，则在将变量传递给被调用程序之前将把字符串长度设置为零。
INTEGER INT	一个带符号的二进制整数，长度为 4 个字节。
SMALLINT	一个带符号的二进制整数，长度为 2 个字节
FLOAT(<i>p,s</i>)	单精度或双精度，浮点数。在单精度情况下， <i>p</i> 必须大于 0 且小于 25。在双精度情况下， <i>p</i> 必须大于 24 且小于 54。精度 (<i>p</i>) 和缩放程度 (<i>s</i>) 都只在将数据转换为可显示格式时使用；例如，在转换为字符串时使用。
REAL(<i>p,s</i>)	一个单精度浮点数。 <i>p</i> 必须大于 0 且小于 25。精度 (<i>p</i>) 和缩放程度 (<i>s</i>) 都只在将数据转换为可显示格式时使用；例如，在转换为字符串时使用。
DOUBLE (<i>p,s</i>) DOUBLEPRECISION(<i>p,s</i>)	一个双精度浮点数。 <i>p</i> 必须大于 0 且小于 53。精度 (<i>p</i>) 和缩放程度 (<i>s</i>) 都只在将数据转换为可显示格式时使用；例如，在转换为字符串时使用。
NUMERIC(<i>p,s</i>)	一个分区十进制数，精度为 <i>p</i> ，缩放程度为 <i>s</i> 。 <i>p</i> 的值必须大于 0 且小于 32。
DEC(<i>p,s</i>) DECIMAL(<i>p,s</i>)	一个压缩十进制数，精度为 <i>p</i> ，缩放程度为 <i>s</i> 。 <i>p</i> 的值必须大于 0 且小于 32。
DTWTABLE	一个特殊的数据类型，用于将 Net.Data 表格传递给被调用的程序。“直接调用”语言环境传递一个指向表格的指针，可以使用 Net.Data 语言环境接口表格函数对其进行处理。

被定义为数字的参数可以包括货币符号和三个数字的分隔符。“直接调用”语言环境在将数值变量传递给程序之前，先把它从字符串格式转换成其内部格式，并在此时除去货币符号和三个数字的分隔符。Net.Data 从运行 Net.Data 的进程的进程属性中检索货币符号、十进制格式和三位数字分隔符字符。

以 NULL 结束的字符串参数

如果配置文件或宏中的 DTW_PAD_PGM_PARMS 设置为 NO，则“直接调用”语言环境将使用一个 NULL 结束符(值 x'00') 把字符串值传递到您的程序。这需要您编写代码来处理字符串(除非使用 C 或 C++，它们使用的是以 NULL 结束的字符串)。

例如，如果定义的参数字段是 CHAR(10)，但传递的字符串值长度为 5 个字节，那么 Net.Data 将在第 5 个字节后加一个 NULL 结束符。将值 “12345” 作为一个字符串传递到 CHAR(10) 字段，结果是：

```
x'F1F2F3F4F500.....'
```

NULL 结束符之后的字节是未定义的(不能假定那些字节为 NULL 或为空)。

因为该字符串是以 NULL 结束的，并且在 NULL 结束符之后包含未初始化的字节，因此不能在 RPG 或 COBOL 程序中使用这个字符串。例如，如果将字符串用在比较操作中，该操作将不会产生有效的结果。程序不期望字符串包含 NULL 结束符，而期望用空格填充字符串的结尾。

可以在程序中使用字符串处理函数来抽取字符串值，也可以使用 VARCHAR 数据类型。这种方法在最前面的两个字节中给出字符串的长度。

如果配置文件或宏中的 DTW_PAD_PGM_PARMS 设置为 YES，“直接调用”语言环境将把用空格向右填充至精度长度的字符串值传递给程序。使用与上述相同的例子，但 DTW_PAD_PGM_PARMS 设置为 YES，那么将值 “12345” 作为一个字符串传递到 CHAR(10) 字段的结果是：

```
x'F1F2F3F4F5404040404000'
```

因为字符串的长度为 5，小于指定的精度，因此在该值之后插入了一些空格，直至达到精度所需的长度。现在，用 RPG 等语言编写的程序就可以使用该参数了，不必再处理以 NULL 结束的字符串。

传递参数时的公共错误

以下列表描述了使用“直接调用”语言环境调用程序或向程序传递参数时可能发生的错误。同时提供了避免这些错误的技巧。

参数不匹配错误

请确保参数的个数与顺序与它们出现在被调用程序参数列表中的个数与顺序相匹配。

数据类型错误

请确保为参数指定的数据类型与被调用程序所期望的数据类型相匹配。可能有些数据类型受“直接调用”语言环境的支持，但不受用于创建被调用程序的高级程序设计语言的支持。

长度错误

请确保为参数定义的长度正确，并与被调用程序中指定的长度相匹配。如果指定的长度比被调用程序所声明的长度短一些，可能破坏存储器中的内容，并使 Net.Data 不能正确工作。

从程序返回值

有些高层程序设计语言(例如 C)可以在程序调用时返回一个整数。这个整数可以通过在函数定义中指定 RETURNS 关键字来获得，例如：

```
%function(DTW_DIRECTCALL) dc3(IN CHAR(3) p1) RETURNS(retval) {  
  %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
  %}
```

当函数调用成功完成时，参数 *retval* 中将包含程序所返回的值。

直接调用语言环境示例

在此例中，宏调用一个程序并传递多个参数。程序的源代码在宏之后，是用 RPG 和 CL 编写的。被调用的程序接收两个整数参数。它将第一个参数(输入参数)复制到第二个参数(输出参数)。

宏:

```
%define ilepgm = "/QSYS.LIB/NETDATADEV.LIB/TDCCLI01.PGM"
%define out1 = "0"

%FUNCTION(DTW_DIRECTCALL) dcFunction(IN INT inp1,
                                     OUT INT outp2)

{ %EXEC { $(ilepgm) %} %}

%HTML(REPORT){
  @dcFunction("123", out1)
  The value of out1 is: "${out1}"
%}
```

ILE RPG 程序:

```
DINP1          S          10I00
DOUTP2         S          10I00
C*
C      *ENTRY      PLIST
C              PARM          INP1
C              PARM          OUTP2
C*
C              Z-ADD      INP1      OUTP2
C*
C              SETON                      LR
```

CL 程序:

```
PGM PARM(&INP1; &OUTP2;)

DCL VAR(&INP1;) TYPE(*CHAR) LEN(4)
DCL VAR(&OUTP2;) TYPE(*CHAR) LEN(4)

CHGVAR VAR(&OUTP2;) VALUE(&INP1;)
ENDPGM
```

Java 应用程序语言环境

Java 应用程序语言环境允许您调用 Java 程序，从而方便地实现了 Java 应用程序与 Net.Data 的集成。Java 应用程序语言环境最初是在 OS/400 版本 4 发行版 4 中介绍的。

要使用 Java 应用程序语言环境，请完成第18页的『设置 Java 应用程序语言环境』中所述的配置步骤。

调用 Java 程序

要调用一个 Java 程序，需要定义一个使用 Java 应用程序 (DTW_JAVAPPS) 语言环境的函数。指定一个代表 Java 程序类名的函数名称。

例子: 调用 Java 程序 helloWorld.java:

```
%function(DTW_JAVAPPS) helloWorld() { %}
```

Java 应用程序语言期望 Java 程序包含用于 'main' 的方式标识符, 这是运行在 Java 程序中的第一个方式。当语言环境调用一个应用程序时, 该应用程序对 stdin 和 stdout 有访问权。stdin 中没有表数据, 因为 Net.Data 已经读取了数据。

重要事项: 调用 Java 应用程序之前, 请先设置 DTW_JAVA_CLASSPATH 路径配置变量, 这样就可以找到 Java 类。请参阅第16页的『DTW_JAVA_CLASSPATH』中有关此变量的语法。

将参数传送到 Java 程序

通过在函数定义中指定要传递的参数来将参数传递给 Java 程序。仅指定“仅用于输入”(IN) 或“输入输出”(INOUT) 的字符串参数。

例子: 要将 IN 参数 p1 传递给函数调用

```
%function(DTW_JAVAPPS) jv1(IN p1) { %}
```

Java 应用程序语言环境不支持更新变量的 Java 程序, 因为它无法将更新后的值传回宏。

Java 应用程序语言环境示例

在此例中, Net.Data 宏调用了一个 Java 程序 echoString。该宏将两个字符串参数传递给 Java 语言环境。第一个字符串告诉 Java 程序在将第二个参数打印到标准输出 (stdout) 之前是否要使用斜体或粗体突出显示第二个参数(是一个文本字符串)。因为程序传递 "I", 表示斜体字, 因此 Web 服务器在浏览器上以斜体显示文本字符串 *Hello World*。Java 程序的源代码在宏之后。

宏:

```
%FUNCTION(DTW_JAVAPPS) echoString(textAttribute, text){ %}  
  
%HTML(runjava){  
@echoString("I","Hello World")  
%}
```

Java 程序:

```
class echoString {  
    public static void main (String args[]) {  
        if (args[0].equals("I"))  
            System.out.println("<I>" + args[1] + "</I>");  
        else  
            System.out.println("<B>" + args[1] + "</B>");  
    }  
}
```

REXX 语言环境

REXX 语言环境允许您运行 REXX 程序。

执行 REXX 程序

通过使用 REXX 语言环境，您可以执行内联的 REXX 程序或外部的 REXX 程序。内联的 REXX 程序是这样一种 REXX 程序，它在宏中具有 REXX 程序的源代码。外部的 REXX 程序在外部文件中具有 REXX 程序的源代码。

要执行内联的 REXX 程序:

定义一个使用 REXX (DTW_REXX) 语言环境的函数，并且该函数中包含 REXX 代码。

例子: 一个包含内联 REXX 程序的函数

```
%function(DTW_REXX) helloWorld() {  
    SAY 'Hello World'  
%}
```

要运行外部的 REXX 程序:

定义一个使用 REXX (DTW_REXX) 语言环境的函数，并且其中包含一个要在 EXEC 语句中运行的 REXX 程序的路径。

例子: 一个包含指向外部程序的 EXEC 语句的函数

```
%function(DTW_REXX) externalHelloWorld() {  
%EXEC{ /QSYS.LIB/REXX.LIB/REXXSRC.FILE/HELLOWORLD.MBR%}  
%}
```

如果使用 EXEC_PATH 配置变量来定义至包含程序的目录的路径，则可以缩短至该程序的路径。请参阅第14页的『EXEC_PATH』，以学习如何定义 EXEC_PATH 配置变量。

限制: 如果您运行的是 OS/400 V3R2 或 V3R7，并且 REXX 程序使用 SAY REXX 指令写至 stdout，则在字符串的开头插入 12 个空格。例如:

```
SAY '          STARTOFDATA'
```

这里的 12 个空格将被忽略，但是如果没有插入这些空格，则可能会出现不可预测的结果。

将参数传递到 REXX 程序

有两种方式将信息传递到 REXX (DTW_REXX) 语言环境调用的 REXX 程序，直接和间接。

直接 使用 %EXEC 语句将参数直接传递到外部 REXX 程序。例如:

```
%FUNCTION(DTW_REXX) rexx1() {  
    %EXEC{  
        /QSYS.LIB/NETDATA.LIB/QREXXSRC.FILE/CALL1.MBR ${INPARM1} %}  
%}
```

Net.Data 变量 INPARM1 被撤消引用并传递到外部的 REXX 程序。REXX 程序可以通过使用 REXX PARSE ARG 指令来引用变量。使用这种方式传递给程序的参数被认为是输入类型参数(传递给程序的参数可以由程序使用和处理，但对参数的更改并反映给 Net.Data)。

间接

通过 REXX 程序变量池的方式间接传递参数。启动 REXX 程序之后，REXX 解释程序将创建并维护一个含有所有变量信息的空间。这个空间就称为变量池。

调用一个 REXX 语言环境 (DTW_REXX) 函数时，REXX 语言环境将在执行 REXX 程序之前把所有输入 (IN) 或输入/输出 (INOUT) 函数参数存储在变量池中。调用 REXX 程序时，它就可以直接访问这些变量。REXX 程序成功完成时，DTW_REXX 语言环境将确定是否有输出 (OUT) 或 INOUT 函数参数。如果有，语言环境将从变量池中检索相应于函数参数的值，并使用新值更新函数参数值。当 Net.Data 接收控制时，它便使用从 REXX 语言环境中获得的新值更新所有的 OUT 或 INOUT 参数。例如：

```
%DEFINE a = "3"
%DEFINE b = "0"
%FUNCTION(DTW_REXX) double_func(IN inp1, OUT outp1){
  outp1 = 2*inp1
}%

%HTML(REPORT){
  Value of b is $(b), @double_func(a, b) Value of b is $(b)
}%
```

在上面的例子中，调用 `@double_func` 传递了两个参数，`a` 和 `b`。REXX 函数 `double_func` 将第一个参数的值乘以 2，并将结果存储在第二个参数中。当 Net.Data 调用宏时，`b` 的值为 6。

您可以将 Net.Data 表格传递给一个 REXX 程序。REXX 程序访问 Net.Data 宏表格参数的值，把它作为 REXX 词干变量。对于 REXX 程序，列标题和字段值都包含在用表格名和列号标识的变量中。例如，表格 `myTable` 中的列标题是 `myTable_N.j`，字段值是 `myTable_N.i.j`，其中 `i` 是行号，`j` 是列号。表格中的行数是 `myTable_ROWS`，列数是 `myTable_COLS`。

REXX 语言环境的例子

以下例子显示了调用 REXX 函数来生成具有两列三行的 Net.Data 表格的宏。在对 REXX 函数的调用之后，将调用一个内部函数 `DTW_TB_TABLE()` 来生成一个要返回给浏览器的 HTML 表格。

```
%DEFINE myTable = %TABLE
%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION(DTW_REXX) genTable(out out_table) {
  out_table_ROWS = 3
  out_table_COLS = 2

  /* Set Column Headings */
  do j=1 to out_table_COLS
    out_table_N.j = 'COL'j
  end

  /* Set the fields in the row */
  do i = 1 to out_table_ROWS
    do j = 1 to out_table_COLS
      out_table_V.i.j = '[' i j ']'
    end
  end
}%
```

```
%HTML(REPORT){
  @genTable(myTable)
  @DTW_TB_TABLE(myTable)
%}
```

结果:

COL1	COL2
[1 1]	[1 2]
[2 1]	[2 2]
[3 1]	[3 2]

SQL 语言环境

SQL 语言环境允许您通过将 SQL 语句发送给数据库管理系统 (DBMS) 来运行 SQL 语句。

要使用 SQL 语言环境，请确保遵循第18页的『设置语言环境』中所述的配置步骤

执行 SQL 语句

可以执行动态 SQL 支持的任何 SQL 语句。

要执行 SQL 语句，需要定义一个使用 SQL (DTW_SQL) 语言环境的函数，并在该函数的语言环境可执行部分包含 SQL 语句。

例子：一个运行 SQL SELECT 语句的 SQL 函数：

```
%function(DTW_SQL) getOrders() {
  SELECT cust, custid, custorder FROM mylibrary.customers
%}
```

确认控制

缺省情况下，SQL 语言环境在确认控制下运行，遵循调节确认控制的所有规则。

- 定期记录所有通过 DTW_SQL 访问的文件或表格，除非这个 SQL 语句是 SELECT 语句。
- 在 Net.Data 初始化文件中指定 DTW_SQL_ISOLATION 可以任意地更改确认级别。请参阅第11页的『DTW_SQL_ISOLATION: DB2 隔离变量』以获取有关 SQL 语言环境所支持的隔离级别的细节。

要获取有关事务管理的更多信息，请参阅第85页的『管理 Net.Data 应用程序中的事务』。

OUT 和 INOUT 表格

如果您在函数定义中指定 OUT 或 INOUT Net.Data 表格，并且 SQL 语句返回结果集，则 SQL 语言环境将在指定的表格中存储每个结果集。然后，您就可以在宏中使用该表格。如果没有指定 OUT 表格，SQL 语言环境将使用缺省的表格。

嵌套的 SQL 语句

您可以从一个 SQL 函数的 ROW 块中调用其它 SQL 函数。在每个 SQL 函数中使用唯一的 Net.Data 表格名，否则，可能出现无法预知的结果。

例子：从 ROW 块或另一个 SQL 函数调用 SQL 函数

```
%define mytable1 = %TABLE
%define mytable2 = %TABLE

%FUNCTION(DTW_SQL) sql2 (IN p1, OUT t2) {
    select * from NETDATA.STAFFINF where projno='$(p1)'
    %REPORT {
        %ROW { $(N1) is $(V1) %}
    %}
%}

%FUNCTION(DTW_SQL) sql1 (OUT t1) {
    select * from NETDATA.STAFFINF
    %REPORT {
        %ROW { @sql2(V1, mytable2) %}
    %}
%}

%HTML(netcall1) { @sql1(mytable1) %}
```

受支持的语言环境变量

SQL 语言环境支持那些为支持 DB2 而设计的变量。例如，DATABASE 变量指定了执行 SQL 语句时 SQL 语言环境所连接到的数据源。以下列表指定了对 SQL 语言环境所支持的变量。请参阅*Net.Data* 参考，以获取对这些变量的描述、变量的语法以及示例。

- DATABASE
- DB_CASE
- DTW_EDIT_CODES
- DTW_PAD_PGM_PARMS
- DTW_SET_TOTAL_ROWS
- LOGIN
- NULL_RPT_FIELD
- PASSWORD
- SHOWSQL
- SQL_STATE
- TRANSACTION_SCOPE

受支持的数据类型

SQL 语言环境支持表5中列出的数据类型

表 5. 数据类型

BLOB(1)	DOUBLE	SMALLINT
CHAR	DOUBLEPRECISION	TIME
CLOB(1)	FLOAT	TIMESTAMP
DATE	GRAPHIC	VARCHAR
DBCLOB(1)	INTEGER	VARGRAPHIC
DECIMAL	REAL	

(1) 这些数据类型不能作为参数传递到存储过程调用。要了解对于存储过程所支持的数据类型，请参阅第87页的『存储过程语法』

请参阅『数据类型考虑』，以了解对于 LOB 和 DATALINK 数据类型的特殊考虑事项。

SQL 语言环境限制

在计划您的环境时，请考虑以下限制：

- 如果以下某个条件存在，则不要使用 SQL 语言环境：
 - 创建了一个用户定义的语言环境，它使用数据库存取类库或 SQL 调用级接口 (CLI)，并且宏中引用了一个用户定义的语言环境
 - 一个使用 SQL CLI 的应用程序将在与 Net.Data 相同的进程中运行
- 内联语句块中的 SQL 语句的最长可达 32KB。
- 最多可使用 50 个本地或远程数据库连接。在使用多个连接时，请考虑以下限制：
 - Net.Data 不允许对同一个数据库的并发连接。
 - 访问一个远程数据库之后，如果 TRANSACTION_SCOPE=MULTIPLE，则不能更改注册 ID，这是缺省情况。请参阅第85页的『管理 Net.Data 应用程序中的事务』。

请参阅第86页的『管理多数据库连接』，以获取有关这些限制的更多信息。

数据类型考虑

SQL 语言环境支持的以下数据类型需要进行特殊考虑。

- 『使用大型对象』
- 第84页的『在结果集中编码 DataLink URL』

使用大型对象

可以将大型对象文件 (LOB) 存储在 DB2 数据库中，并使用针对您的 Web 应用程序的 SQL 语言环境来对它们进行访问。

SQL 语言环境不会将大型对象存储在处理变量的 Net.Data 表格中(例如 V1 或 V2)，当 SQL 查询在结果集中返回 LOB 时，也不将大型对象存储在 Net.Data 表格字段中。相反，当 Net.Data 遇到 LOB 时，它将 LOB 存储在 Net.Data 所创建的一个文件中。该文件位于 HTML_PATH 路径配置变量指定的目录中。Net.Data 表格字段和表格处理变量的值被设置为文件的路径。对该文件的访问被限于运行 Net.Data 的用户 ID。

存储 LOB 的文件名是动态构造的，具有如下格式：

name[.extension]

其中：

name 是一个标识大型对象的唯一的字符串

extension

是一个标识对象类型的字符串。对于 CLOB 和 DBCLOB 来说，扩展名是 'txt'。对于 BLOB，SQL 语言环境通过在 LOB 文件开始的几个字节中查找某个特征符来试图确定表示该 LOB 的扩展名。SQL 语言环境识别以下类型的数据(括号中是所使用的扩展名)：

- 位图图象 (.bmp)
- 图形图象格式 (.gif)
- JPEG 图象文件 (.jpg)
- TIFF 文件格式 (.tif)
- Postscript (.ps)
- 乐器数字接口 (MIDI) 音频文件 (.mid)
- AIFF 音频文件 (.aif)
- 视听交错音频文件 (.avi)
- 基本音频文件 (.au)
- 真实音频文件 (.ra)
- Windows 视听文件 (.wav)
- 可移植文档格式 (.pdf)
- Midi 序列文件 (.rmi)

如果无法识别 BLOB 的对象类型，则文件名上不添加任何扩展名。

当宏文件中引用 LOB 时，SQL 语言环境将返回文件名(在 LOB 文件名前添加字符串 /tmplobs/), 使用以下语法:

/tmplobs/name.[extension]

规划技巧: 对于每个返回 LOB 的查询，在 HTML_PATH 路径配置变量指定的目录中都将创建文件。在使用 LOB 时请考虑系统限度，因为它们会很快地消耗资源。您可能希望创建一个批处理程序来定期对该目录进行清除。建议您使用 DataLink，它将消除 SQL 语言环境在目录中存储文件的需要，从而提高性能，并大大减少对系统资源的使用。

例子: 应用程序用户必须单击文件名才能调用察看器，因为该应用程序使用 MPEG 音频文件 (.MPA)。SQL 语言环境不能识别这种文件类型，因此使用了一个 EXEC 变量来将扩展名附加到文件。

```
%DEFINE{
lobpath = "@DTW_RGETINIDATA("HTML_PATH")"
filename = "@DTW_RREPLACE($(V3), "/tmplobs/", "", "1", "F")"
myFile=%EXEC "REN '${lobpath}/${filename}' '${filename}.mpa'"
%}

%{ where rename is the rename command on your operating system %}
%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
%REPORT{
<P>Here is the information you selected:<P>
%ROW{
$(myFile)
$(V1) Voice sample <IMG SRC="$(V2)">
<A HREF="$(V3).mpa">Voice sample</A><P>
%}
%}
%}

%HTML(REPORT){
@queryData()
%}
```

queryData 函数返回以下 HTML 输出:

```
<P>Here are the images you selected:<P>
Kinson Yamamoto
<IMG SRC="/tmplobs/p2345n1.gif">
<A HREF="/tmplobs/p2345n2.mpa">Voice sample</A><P>
Merilee Lau
<IMG SRC="/tmplobs/p2345n3.gif">
<A HREF="/tmplobs/p2345n4.mpa">Voice sample</A><P>
```

先前例子中的 REPORT 块使用隐式表格变量 V1、V2 和 V3。

- V1 是人名, 是普通文本。
- V2 是 .GIF 文件中的人员照片。图象是内联的。SQL 语言环境自动包括前缀 /tmplobs/ 和 .GIF 扩展名。
- V3 是 .mpa 文件中人的语音的示例。当 SQL 语言环境遇到未被识别的格式(例如 .mpa 文件)时, 它将文件写入 HTML_PATH 配置变量指定的目录中, 不带扩展名。此例显示怎样通过用 EXEC 变量添加扩展名来处理此文件类型。当分辨出变量 \$(V3) 时, 它在文件名之前添加路径 /tmplobs/。例如, /tmplobs/sound2a。在此例中, EXEC 变量使用 REN 命令对文件重新命名, 为文件添加了扩展名 .mpa。在对文件名进行重新命名之前, 要从文件名中除去 /tmplobs/, 并使用 DTW_RGETINIDATA 函数检索要被重新命名的文件的完整路径, 以便检索 HTML_PATH 中指定的路径。当应用程序用户单击 Voice 示例时, 将播放语音。

对于 LOB 的访问权限: 请确保运行 Web 服务器的用户 ID 对于 HTML_PATH 指定的目录具有写的权限。

在结果集中编码 DataLink URL

DataLink 是一种基本构造块, 用于扩展可以存储在数据库文件中的数据类型。通过使用 DataLink, 存储在列中的实际数据便只是一个指向文件的指针了。这个文件可以是任何类型的文件; 图象文件、语音记录或文本文件。DataLink 存储 URL 以便分辨文件的位置。

DATALINK 数据类型需要使用 DataLink 文件管理器。有关 DataLink 文件管理器的更多信息, 请参阅针对您的操作系统的 DataLink 文档。使用 DATALINK 数据类型之前, 必须确保 Web 服务器对 DB2 文件管理器服务器所管理的文件系统具有访问权。

当 SQL 查询使用 DataLink 返回结果集时, 将用具有 READ PERMISSION DB DataLink 选项的 FILE LINK CONTROL 创建 DataLink 列, DataLink 列中的文件路径包含一个访问令牌。DB2 使用访问令牌来授予对文件的访问权。没有这个访问令牌, 所有对该文件的访问都将因权限违例而失败。当然, 访问令牌中可能包含要返回给浏览器的 URL 中不能使用的字符, 例如分号字符 (;)。例如:

```
/datalink/pics/UN1B;0YPVKGG346KEBE;baibien.jpg
```

URL 无效, 因为它包含分号 (;) 字符。要使该 URL 有效, 必须使用 Net.Data 内部函数 DTW_URLESCSEQ 对该分号进行编码。当然, 有些字符串处理必须在使用此函数之前执行, 因为这个函数也将对斜线 (/) 进行编码。

可以编写一个 Net.Data MACRO_FUNCTION 来自动进行字符串处理并使用 DTW_URLESCSEQ 函数。在每个从 DATALINK 数据类型列中检索数据的宏中使用此技术。

例 1: 一个使 DB2 UDB 返回的 URL 自动编码的 MACRO_FUNCTION

```
%{ TO DO: Apply DTW_URLESCSEQ to a DATALINK URL to make it a valid URL.
  IN: DATALINK URL from DB2 File Manager column.
  RETURN: The URL with token portion is URL encoded
%}
%MACRO_FUNCTION encodeDataLink(in DLURL) {
  @DTW_rCONCAT( @DTW_rDELSTR( DLURL,
    @DTW_rADD(@DTW_rLASTPOS("/", DLURL), "1" ) ),
    @DTW_rURLESCSEQ( @DTW_rSUBSTR(DLURL,
    @DTW_rADD( @DTW_rLASTPOS("/", DLURL), "1" ) ) ) )
%}
```

在使用此 MACRO_FUNCTION 之后, URL 即被正确编码, DATALINK 列中指定的文件可以在任何 Web 浏览器上引用。

例 2: 一个 Net.Data 宏, 指定返回 DATALINK URL 的 SQL 查询

```
%FUNCTION(DTW_SQL) myQuery() {
  select name, DLURLCOMPLETE(picture) from myTable where name like '%river%'
%REPORT{
%ROW{
  <p> $(V1) <br>
  Before Encoding: $(V2) <br>
  After Encoding: @encodeDataLink($(V2)) <br>
  Make HREF: <a href="@encodeDataLink($(V2))"> click here </a> <br> <p>
  %}
%}
%}
```

请注意, 这里使用了 DataLink “文件管理器” 函数。函数 dlurlcomplete 返回一个完整的 URL。

管理 Net.Data 应用程序中的事务

当使用 insert、delete 或 update 语句修改数据库的内容时, 只有当数据库接收到来自 Net.Data 的 commit 语句, 这些修改才会变为永久性的修改。如果发生错误, Net.Data 将向数据库发送一个 rollback 语句, 撤消上一次确认 (commit) 之后所作的全部修改。

Net.Data 发送 commit 和 rollback 的方式取决于 TRANSACTION_SCOPE 的设置以及宏中是否显式地指定了 commit。TRANSACTION_SCOPE 的值可以是 MULTIPLE 和 SINGLE。

MULTIPLE

指定 Net.Data 将在发出 commit 以及可能的 rollback 语句之前执行所有的 SQL 语句。Net.Data 在请求的最后发送 commit, 如果每个 SQL 语句都已成功发出, commit 命令将使数据库中的所有修改都变为永久性的修改。如果其中有任何一个语句返回错误, Net.Data 都将发出一个 rollback 语句, 该语句将把数据库设置回原来的状态。如果没有 TRANSACTION_SCOPE, 则缺省值是 MULTIPLE。

要激活这种 commit 方式, 可将 TRANSACTION_SCOPE 设置为 MULTIPLE。

例如:

```
@DTW_ASSIGN(TRANSACTION_SCOPE, "MULTIPLE")
```

SINGLE

指定 Net.Data 在每个成功完成的 SQL 语句后都发出一个 commit 语句。如果 SQL 语句返回错误, 则发出一个 rollback 语句。单次事务作用域确保了数据库修改的立即性; 但是使用此作用域之后, 今后就不可能使用 rollback 语句来撤销所做的修改。

要激活这种 commit 方式, 可将 TRANSACTION_SCOPE 设置为 SINGLE。例如:

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"SINGLE")
```

通过使用 COMMIT SQL 语句, 就可以在宏中每个 SQL 语句之后发出一个 commit 语句。保持 TRANSACTION_SCOPE 设置为 MULTIPLE 的状态并在您觉得可以作为一个事务来对待的每组语句的最后发出 commit 语句, 这样, 您这个应用程序开发者就可以对应用程序中的 commit 和 rollback 进行完全的控制。

要发出 SQL commit 语句, 可以定义一个能在 HTML 块中任意位置调用的函数:

```
%FUNCTION(DTW_SQL) user_commit() {  
    commit  
}%  
  
...  
  
%HTML {  
    ...  
    @user_commit()  
    ...  
}%
```

管理多数据库连接

一次最多可以连接 50 个本地或远程数据库。在正在运行 Net.Data 的 Web 服务器进程作业的整个生命期内, SQL 语言环境将使连接保持活动。在初始连接到数据库之后, 保持连接的活动可以提供快速的数据库访问。在考虑了以下事项之后可以防止错误:

- Net.Data 不允许对同一个数据库的并发连接。如果已经存在了使用一个用户 ID (LOGIN SQL 语言环境参数) 建立的、与远程数据库的一个连接, 此时又有另一个请求, 要求通过第二个用户 ID 连接到同一个远程数据库, 则 SQL 语言环境必须首先断开现存的连接, 作一次提交操作(如果使用了确认控制), 然后使用“新的”用户 ID 和口令重新建立连接。提交操作是必需的, 因为如果连接被断开, 则万一稍后在宏中发生了一个错误, 就没有办法执行复原操作。
- 访问一个远程数据库之后, 如果 TRANSACTION_SCOPE=SINGLE, 则可以更改注册 ID。SQL 语言环境断开现有的连接, 执行提交操作, 然后使用新的用户 ID 和口令重新建立连接。
- 访问一个远程数据库之后, 如果 TRANSACTION_SCOPE=MULTIPLE, 则不要更改注册 ID, 这是缺省情况。SQL 语言环境自动撤消并返回 SQL_CODE -752, 这意味着连接不能更改。

存储过程

存储过程是一个已编译的程序, 存储在可以执行 SQL 语句的 DB2 中。在 Net.Data 中, 使用 CALL 语句从 Net.Data 函数调用存储过程。存储过程的参数传送自 Net.Data 函数参数表。通过以数据库服务器保持编译过的 SQL 语句, 可使用存储过程来改进性能和完整性。Net.Data 支持 DB2 通过 SQL 和 ODBC 语言环境使用存储过程。

本节描述以下主题:

- 『存储过程语法』
- 『调用存储过程』
- 第88页的『传送参数』
- 第89页的『处理结果集』

存储过程语法

存储过程的语法使用 FUNCTION 语句、CALL 语句以及任选的 REPORT 块。

```
%FUNCTION function_name ([IN datatype arg1, INOUT datatype arg2,  
    OUT tablename, ...]) {  
    CALL stored_procedure  
    [%REPORT [(resultsetname)] { %}]  
    ...  
    [%REPORT [(resultsetname)] { %}]  
    [%MESSAGE %]}  
%}
```

其中:

function_name
是 Net.Data 函数的名称, 它初启存储过程的调用。

stored_procedure
是存储过程的名称

datatype
是 Net.Data 支持的数据库数据类型, 正如表6中所显示的那样。参数表中指定的数据类型必须匹配存储过程中的数据类型。请参阅数据库文档以获取关于这些数据类型的更多信息。

tablename
是 Net.Data 表格的名称, 结果集存储在表格中(仅当要在 Net.Data 表格中存储结果集时才使用)。如果指定的话, 此参数名称必须匹配 *resultsetname* 关联的参数名称。

resultsetname
是一个名称, 它将返回自存储过程的结果与函数参数列表上的 REPORT 块和一个表格名称 (或这两者)相关联。REPORT 块上的 *resultsetname* 必须与函数参数列表上的 *tablename*相匹配。

表 6. 存储过程的数据类型

CHAR	FLOAT	SMALLINT
DATE	GRAPHIC	TIME
DECIMAL	INTEGER	TIMESTAMP
DOUBLE	REAL	VARCHAR
DOUBLEPRECISION		VARGRAPHIC

调用存储过程

1. 定义一个函数, 让它初启对存储过程的调用。

```
%FUNCTION (DTW_SQL) function_name()
```

2. 任选地，为存储过程指定任何 IN、INOUT 或 OUT 参数，包括从存储过程返回的任何结果集的结果集名称。还可以从另一个存储过程将其指定为表格名称或结果集，指定为 IN 或 INOUT 参数。

```
%FUNCTION (DTW_SQL) function_name (IN datatype  
arg1, INOUT datatype arg2,  
OUT tablename...)
```

3. 使用 CALL 语句来标识存储过程名称。

```
CALL stored_procedure
```

4. 如果存储过程将要生成一个结果集，则任选地指定一个 REPORT 块来定义 Net.Data 如何显示结果集。

```
%REPORT {  
...  
%}
```

例子:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1 OUT mytable) {  
    CALL myproc  
    %REPORT {  
        ...  
    %ROW { ... %}  
        ...  
    %}  
%}
```

5. 如果存储过程将要生成多个结果集:

- 在 FUNCTION 语句中将结果集指定为 OUT 参数。结果集被保存为本地表格。

```
%FUNCTION (DTW_SQL) function_name (OUT tablename, ...)
```

- 任选地指定一个或多个 REPORT 块来定义 Net.Data 如何显示结果集。

```
%REPORT(resultsetname1) {  
...  
%}
```

例子:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1, OUT table1, table2) {  
    CALL myproc  
    %REPORT(table1) {  
        ...  
    %ROW { ... %}  
        ...  
    %}  
    %REPORT(table1) {  
        ...  
    %ROW { ... %}  
        ...  
    %}  
%}
```

传送参数

您可以将参数传送给一个存储过程，并可以让存储过程更新参数值，以使新值传送回 Net.Data 宏。函数参数列表上的参数个数与类型必须与为存储过程定义的个数与类型相匹配。例如，如果参数列表上为存储过程定义的一个参数是 INOUT，那么函数参数列表上相应的参数必须是 INOUT。如果列表上为存储过程定义的参数类型是 CHAR(30)，那么函数参数列表上相应的参数必须也是 CHAR(30)。

例子 1: 将参数值传送给存储过程

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) valuein) {  
    CALL myproc  
    ...  
}
```

例子 2: 从存储过程返回一个值

```
%FUNCTION (DTW_SQL) mystoredproc (OUT VARCHAR(9) retvalue) {  
    CALL myproc  
    ...  
}
```

处理结果集

可以从存储过程返回一个或多个结果集。结果集可存储在 `Net.Data` 表格中，以备在宏中进一步处理或使用 `REPORT` 块来处理。如果一个存储过程生成多个结果集，就必须使存储过程生成的每个结果集与一个名称相关联。这是通过在 `FUNCTION` 语句中指定参数来完成的。为结果集指定的名称就与一个 `REPORT` 块或 `Net.Data` 表格关联起来，允许您确定 `Net.Data` 如何处理每个结果集。您可以：

- 通过不为结果集定义报表块的方式来以 `Net.Data` 的缺省报表风格处理结果。
- 将结果集与 `REPORT` 块相关联，以便应用您自己的报表风格。在 `REPORT` 块中，可以使用 `Net.Data` 变量、HTML 或 JavaScript 等文本处理语句或其它函数来指定如何在浏览器中显示报表数据。

结果集通常存储在本地表格中，因此宏中的其它函数也可以访问其中的数据。例如，可以将一个 `Net.Data` 表格传送到另一个函数，以使它可使用数据进行计算并根据这些计算来显示结果。

请参阅第66页的『多个 `REPORT` 块的准则和限制』，以获取使用多个报表块时的准则和限制。

要返回单个结果集并使用缺省报表:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename) {  
    CALL stored_procedure  
}%
```

例如:

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {  
    CALL myproc  
}%
```

要返回单个结果集并指定一个 `REPORT` 块:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename) {  
    CALL stored_procedure [(resultsetname)]  
    %REPORT [(resultsetname)] {  
        ...  
    %}  
}%
```

例如:

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {  
    CALL myproc  
    %REPORT {  
        ...  
    }
```

```

%ROW { ... %}
...
%}
%}

```

同样可使用以下语法:

```

%FUNCTION (DTW_SQL) function_name () {
    CALL stored_procedure

    %REPORT () {
        ...
    %}
%}

```

例如:

```

%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
    %REPORT {
        ...
        %ROW { ... %}
        ...
    %}
%}

```

要返回多个结果集并用缺省报表格式显示它们:

使用以下语法:

```

%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2) {
    CALL stored_procedure
%}

```

其中不指定报表块。

例如:

```

%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
    CALL myproc
%}

```

要返回多个结果集并指定 *REPORT* 块以备显示处理:

每个结果集都与一个或多个REPORT 块相关联的。使用以下语法:

```

%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2, ...) {
    CALL stored_procedure
    %REPORT (tablename1)
    ...
    %ROW { ... %}
    ...
    %}
    %REPORT (tablename2)
    ...
    %ROW { ... %}
    ...
    %}

    ...
%}

```

例如:

```

%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
    CALL myproc

    %REPORT (mytable1) {
        ...
        %ROW { ... %}
        ...
    %}

    %REPORT(mytable2) {
        ...
        %ROW { ... %}
        ...
    %}
}

```

SQL 语言环境示例

以下例子显示了具有 DTW_SQL 函数定义的宏(该函数定义调用一个 SQL 存储过程)。它有三个不同数据类型的参数。DTW_SQL 语言环境将每个参数中的字符串值转换成正确的内部格式，并以“传地址”的方式将每个参数传递给 SQL 存储过程。当 SQL 存储过程完成处理之后，更新后内部表示被转换成一个字符串放置在相应的参数中。

```

%{*****}
*****%}
DEFINE {
    MACRO_NAME      = "TEST ALL TYPES"
    DTW_HTML_TABLE  = "YES"
    Procedure        = "NDLIB.TESTTYPE"
    parm1            = "1"                %{SMALLINT      %}
    parm2            = "11"               %{INT           %}
    parm3            = "1.1"              %{DECIMAL (2,1) %}
    %}

%FUNCTION(DTW_SQL) CRTPROC(){
    CREATE PROCEDURE $(Procedure)
    ( INOUT SMALLINT,
      INOUT INT,
      INOUT DECIMAL(2,1))
    EXTERNAL NAME $(Procedure) LANGUAGE C SIMPLE CALL
%MESSAGE{
    default : "$(DTW_DEFAULT_MESSAGE) : continuing.<br>": continue
    %}
%}

%FUNCTION(DTW_SQL) myProc
    (INOUT SMALLINT    parm1,
     INOUT INT         parm2,
     INOUT DECIMAL(2,1) parm3){
CALL $(Procedure)
%}

%HTML(REPORT){
<HEAD>
<TITLE>Net.Data : SQL Stored Procedure: Example '$(MACRO_NAME)'. <?TITLE>
</HEAD>
<BODY BGCOLOR="#BBFFFF" TEXT="#000000" LINK="#000000">
<p><p>
Calling the function to create the stored procedure.
<p><p>
    @CRTPROC()
< hr>
<h2>

```

```

Values of the INOUT parameters
prior to calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br>
$(parm1)<p>
<b>parm2 (INT)</b><br>
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br>
$(parm3)<p>
<p>
< hr>
<h2>
Calling the function that executes the stored procedure.
</h2>
<p><p>
    @myProc(parm1,parm2,parm3)
< hr>
<h2>
Values of the INOUT parameters after
calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br>
$(parm1)<p>
<b>parm2 (INT)</b><br>
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br>
$(parm3)<p>
</body>
%}

```

System 语言环境

System 语言环境支持执行命令和调用外部程序。

发出命令和调用程序

要发出一个命令，可以定义使用 System (DTW_SYSTEM) 语言环境的函数，它包含一个路径，该路径指向要在 EXEC 语句中发出的命令。例如：

```

%FUNCTION(DTW_SYSTEM) sys1() {
    %EXEC { /QSYS.LIB/ADDLIBLE.CMD LIB(MYLIBRARY) %}
%}

```

如果使用 EXEC_PATH 配置变量来定义至包含可执行对象(例如，命令和程序)的目录的路径，则可以缩短至该对象的路径。请参阅第14页的『EXEC_PATH』，以学习如何定义 EXEC_PATH 配置变量。

例 1：发出命令

```

%FUNCTION(DTW_SYSTEM) sys2() {
    %EXEC { /QSYS.LIB/CALL.CMD MYLIB/MYPGM %}
%}

```

例 2：调用程序

```

%FUNCTION(DTW_SYSTEM) sys3() {
    %EXEC { /QSYS.LIB/MYLIB.LIB/MYPGM.PGM %}
%}

```

技巧：调用程序时，请使用“直接调用”语言环境，因为它更为有效，并且易于使用。

将参数传送到程序

有两种方式将信息传递到 System (DTW_SYSTEM) 语言环境调用的程序，直接和间接。

直接 在调用程序时直接传递参数。例如:

```
%DEFINE INPARAM1 = "SWITCH1"

%FUNCTION(DTW_SYSTEM) sys1() {
  %EXEC{
    /QSYS.LIB/NETDATA.LIB/RPGCALL1.PGM ('${INPARAM1}' 'literalstring')
  %}
%}
```

Net.Data 变量 INPARAM1 被引用并被传递到程序。参数传递到程序的方式与从命令行调用程序时参数传递到程序的方式相同。使用这种方式传递给程序的参数被认为是输入类型参数(传递给程序的参数可以由程序使用和处理，但对参数的更改并反映给 Net.Data)。

间接

使用环境变量间接地传递参数。环境变量是形式为 "name=value" 的字符串，存储在程序以外的环境空间中。字符串存储在与进程关联的临时空间中。

当 Net.Data 调用一个 DTW_SYSTEM 语言环境函数时，语言环境将在执行 %EXEC 块中的语句之前把所有输入 (IN) 或输入/输出 (INOUT) 的函数参数存储到环境空间中。语句成功完成后，DTW_SYSTEM 语言环境将确定是否有输出 (OUT 或 INOUT) 函数参数。如果有，语言环境将从环境空间中检索相应于函数参数的值，并使用新值更新函数参数值。当 Net.Data 接收控制时，它便使用从 DTW_SYSTEM 语言环境中获得的新值更新所有的 OUT 或 INOUT 参数。

使用表7中所描述的 API 来设置与检索环境变量:

表 7. 环境变量 API

ILE 编程语言	要检索，请使用...	要设置，请使用...
C, C++	getenv()	putenv()
CL(1), RPG, COBOL	QtmhGetEnv()(2)	QtmhPutEnv()(3)

- 1. 对于 OS/400 V3R7 及后继版本，您还可以使用 CHGENVVAR 和 ADDENVVAR CL 命令来设置环境变量。
- 2. QtmhGetEnv() 是作为 IBM TCP/IP Connectivity Utilities/400 的一部分发行的。
- 3. QtmhPutEnv() 最初没有作为 IBM TCP/IP ConnectivityUtilities/400 V3R2 和 V3R7 的一部分发行。它是后来在周期中添加的，可以通过 V3R2 PTF 5763TC1-SF40953 或 V3R7 PTF 5716TC1-SF40954 获得。

您可以通过 System 语言环境将 Net.Data 表格传递给一个被调用的程序。该程序根据它们的 Net.Data 名来访问 Net.Data 宏表格参数的值。列标题和字段值都包含在用表格名和列号标识的变量中。例如，在表格 myTable 中，列标题是 myTable_N_j，字段值是 myTable_V_i_j，其中 i 是行号，j 是列号。表格的行数与列数是 myTable_ROWS 和 myTable_COLS。

不建议您传递有许多行的表格，因为进程环境变量的个数是有限的。

System 语言环境示例

以下示例显示了一个使用 System 语言环境向所有工作站消息队列发出 Send Break Message (SNDBRKMSG) 命令的宏。要发送的消息正文是从表数据构建的 (msgToSend)。

```
%FUNCTION(DTW_SYSTEM) sndbrkmsg () {  
  %EXEC { /QSYS.LIB/SNDBRKMSG.COMD MSG('$(msgToSend)') TOMSGQ(*ALLWS) %}  
  %}  
  %HTML(sndbrkmsg) {  
    @sndbrkmsg()  
  %}
```

第7章 具有持久性宏的事务管理

Net.Data 对具有持久性宏的事务处理提供了支持。持久性宏是一个包含内部函数的宏，这些内部函数允许宏作为 Web 服务器中持久性 CGI 进程的一部分来运行。这意味着一个宏的多个块或者多个宏可以作为一个逻辑事务的一部分运行。

对于非持久性宏，Net.Data 将每个宏调用都作为一个完整的事务。这意味着在每个应答发送给浏览器之后，将提交数据库，释放资源，一切都设置为初始状态。下一次调用同一个宏时，就要根据作为表数据传递到宏的信息或者宏本身的信息重新建立应用程序的状态。没有调用之间保存宏变量的功能，没有未明确撤消所做更改的情况下撤消数据库更改的功能，也没有将跨多个浏览器阶段作业的数据库更改作为一个完整事务对待的功能。

对于持久性宏，应用程序开发者就可以在事务级构建自己的应用程序，在维护一个持久性连接的同时调用一个或多个宏。这意味着变量数据在调用之间是持久性的，因此您不再需要在宏调用之间将信息(例如用户注册 ID)作为隐藏变量传递。这包括 Net.Data 表格变量，它们在非持久性宏中不能跨调用地传递。最重要的是，如果在一个事务进行的过程中用户决定取消，那么应用程序可以撤消所有的工作。

请参阅第34页的『调用持久性宏』以学习如何调用持久性的宏。

本章将描述以下主题：

- 『关于持久性的宏』
- 第96页的『定义事务』
- 第101页的『持久性宏的例子』

关于持久性的宏

使用持久性宏时，Net.Data 在 Web 服务器的一个特殊的持久性 CGI 进程中运行，通过标准输入接收输入和环境变量，并通过标准输出提供数据。当然，在输出返回给 Web 服务器之后，Web 服务器不必终止 Net.Data 进程。相反，进程仍保持活动状态，等待通过 Web 浏览器来自用户的响应。因为进程没有终止，因此 Net.Data 可以对宏维护状态信息，并使事务保持打开状态。

Net.Data 可以通过向 Web 服务器发送一个新的 HTTP 首部来告诉服务器它希望在一个持久性 CGI 进程中运行。在 AS/400 HTTP Server 版本 4 发行版本 3 (V4R3) 中添加了对新首部『Accept-HTSession』的支持。Net.Data 在它发送第一个输出时决定将哪些 HTTP 首部发送给服务器，因为首部必须先于输出。在开发一个持久性宏时，蕴含了以下含义：

- 在从宏生成第一个输出时，Net.Data 必须知道这是否将是一个持久性宏。
- 如果使用新的持久性宏内部函数，必须在生成任何输出之前将该宏指定为持久性宏。

在以后的文档中请注意这些限制。

持久性 Net.Data 进程的特性非常类似于那些标准 Net.Data 进程，除了以下一些例外：

- 它们在面向伪连接的环境中运行。Net.Data 和 Web 服务器之间的连接是持久性的，但浏览器和 Web 服务器之间仍然没有连接。
- 它们可以有长时间运行的事务。因为单个的 Net.Data 进程可能跨越多个浏览器请求，因此可以保留事务的打开状态，并根据后继的浏览器请求或错误条件执行提交或撤消。
- 持久性的 Net.Data 进程可能消耗更多的系统资源，因为它可能在一段很长的时间内保持活动。在管理那些资源的时候必须注意。
- 可移植性将会降低，因为 Web 服务器必须包含对持久性的支持。

定义事务

一个事务可以跨越一个 HTML 块、多个 HTML 块或多个宏。当您指定您希望宏在事务中为持久性宏时，需要定义该事务的开始和结尾，还要定义事务中包含了哪些 HTML 块。Net.Data 提供了帮助您完成以下持久性宏任务的内部函数：

- 『启动事务』
- 第97页的『在事务中指定 Macro HTML 块』
- 第100页的『结束事务』
- 第100页的『定义变量在事务中的作用域』
- 第101页的『指定事务中的 COMMIT 和 ROLLBACK』

启动事务

在向浏览器发送任何输出之前，您可以通过向 Net.Data 指出一个宏在您的宏中是持久性的来启动一个事务。然后，Net.Data 向 Web 服务器发送一个特殊的 HTTP 首部，告诉它该宏需要持久性的 CGI 支持。

要启动一个事务：

在任何输出发送到 Web 浏览器之前，请在宏中使用以下方法：

- 调用 DTW_STATIC() 内部函数。

DTW_STATIC() 函数告诉 Net.Data 当前的宏是持久性的。

语法： @DTW_STATIC (["*timeout*"])

其中 *timeout* 是一个可选的参数，用于指定 Web 服务器在结束事务前等待来自浏览器的响应的的时间(以秒计算)。

例子：

```
@DTW_STATIC("60")
%DEFINE{
  var1 = "val1"
  var2 = "val2"
}%
...

%HTML(input){
  ...
}%
```

```
%HTML (report){
...
%}
```

对这个事务指定了 60 秒的超时值。如果 60 秒之内没有收到来自浏览器的响应，Web 服务器将结束该事务。这不会影响浏览器上的当前页。当然，本来将成为这个事务一部分的下一页现在就成了新事务中的一部分。

- 用 STATIC 属性定义一个变量。

语法: %DEFINE(STATIC) var1 = "val1"

例子:

```
%DEFINE(STATIC) var1 = "val1"
%DEFINE var2 = "val2"
...
%HTML(input){
...
%}
%HTML (report){
...
%}
```

静态定义的变量在整个事务中保持其值，而一个事务可以跨几个 Net.Data 调用。

在事务中指定 Macro HTML 块

可以在调用 HTML 块的 URL 请求中使用一个称为事务句柄的标识符来定义哪些 HTML 块是您的事务的一部分。定义和使用事务句柄有三步：

1. 在宏中定义事务句柄。
2. 调用 DTW_ACCEPT 内部函数将句柄名传递给 Net.Data 和 Web 服务器。
3. 在 URL 请求中指定句柄来调用下一个 HTML 块。

要定义事务句柄:

1. 在 DEFINE 部分为事务句柄定义一个变量。例如:

```
%DEFINE handle=""
```

2. 通过在 DEFINE 部分中指定 DTW_RTVHANDLE() 内部函数来选择生成唯一的事务句柄。

语法: @DTW_RTVHANDLE(*handle_name*)

例子:

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)
```

事务句柄可以是任何有效的字符串。当然，DTW_RTVHANDLE() 函数通过生成唯一的事务句柄提供了一种安全性的测量，从而防止其他人调用要在您的事务中运行的宏。

要对 Net.Data 指定一个事务句柄:

用 DTW_ACCEPT() 内部函数对 Net.Data 指定事务句柄的值。因为这个句柄是包含在发送给服务器的 HTTP 首部信息中的一部分，因此必须在宏生成任何输出之前调用 DTW_ACCEPT() 函数。通常，它是 HTML 块中的第一个元素。

语法: @DTW_ACCEPT(*handle_name*, ["*timeout*"])

其中 *timeout* 是一个可选的参数，用于指定 Web 服务器在结束事务前等待来自浏览器的响应的时间(以秒计算)。

您可以在 HTML 块内部或任何 HTML 块外部调用 DTW_ACCEPT()。如果在任何 HTML 块的外部调用该函数，则事务句柄和可选的超时值适用于宏内部的所有 HTML 块。

例 1: 对后继的 URL 请求指定一个事务句柄以便在这个事务中运行

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)

%HTML(Block1){
@DTW_ACCEPT(handle)
...
%}
```

重要事项: 当您将 DTW_ACCEPT() 作为 HTML 块中的第一个元素调用时，请确保在指定 %HTML 语句的行和 DTW_ACCEPT() 调用本身之间没有空白。Net.Data 把这个空白看作要发送给浏览器的正文并发出错误，因为在将数据发送给浏览器之前没有找到 DTW_ACCEPT() 调用。

例 2: 在宏中指定一个适用于所有 HTML 块的事务句柄

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)

@DTW_ACCEPT(handle)

%HTML(Block1){
...
%}

%HTML(Block2){
...
%}
```

要在调用将在您的事务中运行的 HTML 块时指定句柄:

在生成一个事务句柄并调用 DTW_ACCEPT() 函数之后，只有那些具有事务句柄的 URL 才能够在您的事务中运行。在 URL 中，事务句柄必须紧随在 CGI 程序名之后。

注意: 在代码中输入语句时，URL 应在一行上且不含空格，这里为了显示上的需要而分作两行。

- HTML 链:

```
<A HREF="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</A>
```

- HTML 表:

```
<FORM METHOD=method
ACTION="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</FORM>
```

- URL:

```
http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]
```

参数:

server 指定了 Web 服务器的名称。如果是本地服务器, 则可以忽略服务器名称而使用相关的 URL。

Net.Data_invocation_path

Net.Data 可执行文件的路径和文件名。例如, /cgi-bin/db2www/。

transaction_handle

指定哪些 URL 是由 Net.Data 宏初启的事务中的一部分。此标识符是通过调用 DTW_RTVHANDLE 内部函数来得到的, 并且必须跟在 *Net.Data_invocation_path* 之后。

filename

指定 Net.Data 宏文件的名称。Net.Data 搜索并试图用 MACRO_PATH 初始化路径变量中定义的路径语句来与这个文件名匹配。请参阅第13页的『MACRO_PATH』以获取更多信息。

block 在引用的 Net.Data 宏中指定 HTML 块的名称。

method 指定与表一起使用的 HTML 方法。建议采用 METHOD=POST。

?name=val&...

指定一个或多个传递给 Net.Data 的可选参数。

通常, 您将对这些 URL 提供 HTML 链, 或者在宏中的表操作标记上指定 URL。

例 1: 具有链接到同一事务内其它宏调用的链的典型 HTML 块

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html(report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
  macros.file/pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
  macros.file/pcgil.mbr/quit">quit</a><br>
%}
```

例 2: 具有链接到另一个宏的 FORM ACTION 链的典型 HTML 块

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html(input) {
@DTW_ACCEPT(handle)
...
<form method=post action="/cgi-bin/db2www/${handle}/qsys.lib/
mylib.lib/macros.file/pcgil.mbr/report2">
<p>What type of hardware do you want to see?
<menu>
<li><input type="radio" name="hardware" value="MON" checked>Monitors
<li><input type="radio" name="hardware" value="PNT">Pointing devices
<li><input type="radio" name="hardware" value="PRT">Printers
<li><input type="radio" name="hardware" value="SCN">Scanners
```

```
</menu>
</form>
%}
```

结束事务

可以通过向 Net.Data 指出您不再希望您的宏为持久性宏来结束一个事务。

要结束事务:

使用 DTW_TERMINATE() 内部函数来指定一个事务的结尾。象 DTW_ACCEPT() 函数一样, 必须在宏生成任何输出之前调用这个函数, 并且它一般都被指定为 HTML 块中的第一个元素。DTW_TERMINATE 告诉 Net.Data 本次调用是当前事务中的最后一次调用。

语法: @DTW_TERMINATE()

此函数不接受任何参数。

例子:

```
%html(quit) {
@DTW_TERMINATE()
...
%}
```

定义变量在事务中的作用域

您可以通过把作用域指定为 %DEFINE 语句的属性来确定一个变量在事务中具有什么作用域。您可以指定

事务作用域

变量的作用域是整个事务。

单调用作用域

变量的作用域是单个 Net.Data 调用。

要对一个变量指定事务作用域:

指定属性 STATIC 来指出变量具有事务范围的作用域, 意味着变量的值在事务中的所有调用之间得以保存。STATIC 是持久性宏的缺省值。例如:

```
@dtw_static()
%define(static) var1 = "val1"
```

要对一个变量指定单调用作用域:

指定属性 TRANSIENT 来指出变量的作用域为单调调用, 意味着变量的值在每次调用时会被重新初始化。TRANSIENT 是非持久性宏的缺省值。例如:

```
@dtw_static()
%define(transient) var1 = "val1"
```

在持久性宏中:

- 如果 DTW_STATIC() 调用之后的所有变量都没有明确定义为 TRANSIENT, 那么它们都是 STATIC。

- 如果 DTW_STATIC() 调用之前的所有变量都没有明确定义为 STATIC，那么它们都是 TRANSIENT。

指定事务中的 COMMIT 和 ROLLBACK

在一个非持久性宏中，提交或撤消操作是由 Net.Data 在宏调用结束时根据调用的成功或失败隐式完成的。对于持久性宏，提交或撤消操作是在事务结束时完成的。当然，因为一个事务可能跨越许多宏调用，因此您可能希望在事务中多次提交或撤消更改。

要在事务进行期间提交暂挂的更改：

指定 DTW_COMMIT() 内部函数。

这个函数不带任何参数，它执行事务中所有暂挂的更改。

例如：

```
%html(report) {
@dtw_accept(handle)
...
%IF (action="Enter")
    @dtw_commit()
%ENDIF

%}
```

要撤消事务中的暂挂更改：

指定 DTW_ROLLBACK() 内部函数。

这个函数不带任何参数，它撤消事务中所有暂挂的更改。

例如：

```
%html(report) {
@dtw_accept(handle)
...
%IF (action="Cancel")
    @dtw_rollback()
%ENDIF

%}
```

持久性宏的例子

以下这个简单的宏包含了多个在一个单独事务中运行的 HTML 块：

```
@dtw_static()
%define a = "0"
%define(transient) b = "0"
%define handle = ""
@dtw_rtvhandle(handle)

%html(report) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2">
```

```

click here to continue</a><br>
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
click here to quit</a><br>
%}

%html(report2) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report3">
Click here to continue</a><br>
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(report3) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(quit) {
@dtw_terminate()
a = $(a)<br>
b = $(b)<br>
done
%}

```

假定第一个调用是调用 HTML 块 report，则 Net.Data:

1. 调用 DTW_STATIC() 函数，它表示此宏是持久性的。
2. 创建变量 a，作为 STATIC 变量，因为持久性宏的缺省值就是 STATIC。
3. 创建变量 b，作为 TRANSIENT 变量，因为它用 TRANSIENT 属性明确定义的。
4. 调用 DTW_RTVHANDLE()，它生成一个事务句柄并将该句柄放入变量 handle。
5. 开始处理 HTML 块 report 并调用 DTW_ACCEPT()，它告诉 Net.Data 这个事务的事务句柄是什么。
6. 找到要发送给浏览器的输出，这将使 Net.Data 向 Web 服务器发送 HTTP 首部，表示事务将要开始。
7. 显示 HTML 页面。变量 a 和 b 的值都为 0。

在将第一个页面输出发送给浏览器之后，用户可以选择继续该事务或者退出。如果选择继续，则 Web 服务器将调用 URL:

```
/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2
```

Web 服务器将事务句柄识别为 Net.Data 在 HTTP 首部所指定的句柄。它将 Net.Data 作为持久性 CGI 程序调用，这意味着宏调用是当前事务的一部分。

调用 HTML 块 report2 时，Net.Data:

1. 调用 DTW_STATIC() 函数，它表示此宏是持久性的。
2. 识别出变量 a 是一个 STATIC 变量，保留当前值而不是重新初始化为 0。
3. 识别出变量 b 是一个 TRANSIENT 变量，创建该变量的一个新实例并初始化为 0。

4. 调用 `DTW_RTVHANDLE()`, 它生成一个事务句柄并将该句柄放入变量 `handle`。
5. 开始处理 HTML 块 `report2` 并调用 `DTW_ACCEPT()`, 它告诉 `Net.Data` 这个事务的事务句柄是什么。
6. 找到要发送给浏览器的输出, 这将使 `Net.Data` 向服务器发送 HTTP 首部, 表示事务将继续。
7. 显示 HTML 页面。变量 `a` 的值将为 2, 而变量 `b` 的值将为 0。 `a` 的值在前一次调用后被保留, 因为它是一个静态变量。变量 `b` 的值被复位为 0。

在将第二个页面发送给浏览器之后, 用户可以选择继续该事务或者退出。如果选择退出, 则 Web 服务器将调用以下 URL:

```
/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit
```

Web 服务器将事务句柄识别为 `Net.Data` 在 HTTP 首部所指定的句柄, 并将 `Net.Data` 作为持久性 CGI 程序调用, 这意味着宏调用是当前事务的一部分。

调用 HTML 块退出时, `Net.Data`:

1. 调用 `DTW_STATIC()` 函数, 它表示此宏是持久性的。
2. 识别出变量 `a` 是一个 `STATIC` 变量, 保留当前值而不是重新初始化为 0。
3. 识别出变量 `b` 是一个 `TRANSIENT` 变量, 创建该变量的一个新实例并初始化为 0。
4. 调用 `DTW_RTVHANDLE()`, 它生成一个事务句柄并将该句柄放入变量 `handle`。
5. 开始处理 HTML 块 `quit` 并调用 `DTW_TERMINATE()`, 它告诉 `Net.Data` 这是该事务中最后一个调用。
6. 找到要发送给浏览器的输出, 这将使 `Net.Data` 向服务器发送 HTTP 首部, 表示事务将要结束。
7. 显示 HTML 页面。变量 `a` 的值为 4, 变量 `b` 的值为 0。
8. 清除所有变量和具有事务级作用域的资源(因为执行了 `DTW_TERMINATE()` 调用)。

第8章 改进性能

改进性能是调整系统时的一个重要部分。本章讨论改进 Net.Data 性能的策略。将讨论以下主题:

- 『Net.Data 宏的高速缓存』
- 『优化语言环境』

另外, 请确保已经正确地设置了 Web 服务器。无论 Net.Data 处理一个宏或直接请求的速度多快, Web 服务器的性能对响应时间都有直接影响。

Net.Data 宏的高速缓存

使用 Net.Data for OS/400, 将缺省地启用并使用宏高速缓存来提高吞吐率, 降低对 CPU 的利用。启用宏高速缓存之后, 经过预处理的宏在被首次调用之后便被高速缓存在内存中。然后, 这些预处理的版本可用于再使用, 从而消除了与从 HFS 读取宏并在该宏每次被请求时进行处理相关联的成本。对于那些对包含宏的文件具有读取权限的请求者来说, 可以使用宏被高速缓存的版本。

宏的预处理版本所使用的内存量大约是该宏文件本身大小的两倍。您可以通过使用高速缓存配置变量来控制用于宏高速缓存的内存量。要获取有关使用此变量的更多信息, 请参阅第8页的『DTW_MACRO_CACHE_SIZE: 宏高速缓存尺寸变量』。

优化语言环境

以下章节将说明在使用 Net.Data 提供的语言环境时可用于改进性能的技术。

- 『REXX 语言环境』
- 第106页的『SQL 语言环境』
- 第106页的『System 语言环境』

REXX 语言环境

使用以下技巧来改进 Net.Data 应用程序的性能:

- 在可能的地方将 REXX 程序组合起来。程序少一些、大一些, 这样所提供的性能比有许多小程序的情况下所提供的性能要好一些, 因为每次在宏中调用 REXX 语言环境函数时都要初始化 REXX 解释程序。
- 将 REXX 程序存储在一个外部文件中, 而不是把 REXX 程序内联在 Net.Data 宏中。
- 对于外部的 REXX 程序, 可以在 %EXEC 语句中从命令行引用全局变量。
- 通过定义全局 Net.Data 变量和引用变量, 可以将仅用于输入的参数直接传递给 REXX 程序。对于内联的 REXX 程序, 可以在 REXX 源代码中直接引用全局变量。

SQL 语言环境

要了解有关 DB2 的性能考虑，请参阅 *DB2 for OS/400 SQL Programming*。本出版物中有大量的信息，例如高效地使用 SQL 索引、改进连接查询的性能以及在从两个以上表格中选择数据时改进性能。

使用以下 SQL 语言环境技术来改进性能。

- 减少连接到数据库的用户 ID 数，以避免重新连接到数据库。SQL 语言环境将一个至任何远程连接的用户简要表和口令与它所建立的数据库关联起来。如果 LOGIN 和 PASSWORD 变量与一个打开的连接所关联的用户简要表和口令不匹配，则将关闭这个连接并重新建立连接，LOGIN 和 PASSWORD 值将与重新打开的连接相关联。
- 使用 START_ROW_NUM 和 RPT_MAX_ROWS Net.Data 变量来减少返回表格的大小。如果在执行 SELECT SQL 语句后的结果集中包含成百上千条记录，则可以向浏览器返回结果集的一个子集，使用 START_ROW_NUM 可以使它象一个可滚动的光标，而 RPT_MAX_ROWS 则可以限制返回记录的数量。您应意识到 Net.Data 在每次都重新发出查询，这是因为没有状态的概念。当然，您可以使用 Net.Data 对持久性宏的支持将结果集存储在一个 Net.Data 表格中(这个表格在事务的生命期中是持久性的)。请参阅第95页的『第7章 具有持久性宏的事务管理』来更多地了解持久性 Net.Data 宏。
- 请考虑调用一个使用静态 SQL 的存储过程。动态 SQL 是在运行时准备的，而静态 SQL 是在预编译阶段就准备的。SQL 语言环境使用动态 SQL，这允许它在程序运行期间运行 SQL 语句。因为准备语句需要额外的进程时间，因此静态 SQL 可能更为有效。

请注意，从 OS/400 V4R2 开始，SQL 引擎中增加了一个准备语句高速缓存。使用了高速缓存之后，SQL 引擎将存储有关准备语句的信息，并将这个信息保留在系统范围的存储器中。然后，当同一个语句再次执行时，即使是由不同的用户和不同的作业来执行，这个语句的运行速度将大大提高。系统范围的准备语句高速缓存是正常 SQL 处理的一部分，不需要用户操作来对它配置或启用。这个高速缓存可能会降低静态 SQL 可能超过动态 SQL 的性能优势。

System 语言环境

通过定义全局 Net.Data 变量和引用变量，可以将仅用于输入的参数直接传递给 System 语言环境正在调用的程序。

Net.Data for OS/400 中引入了一种新的语言环境，称为“直接调用”，它为调用程序提供了一个更为方便、更为有效的接口。使用 System 语言环境来发出命令；使用“直接调用”语言环境来调用程序

附录A. 书目提要

本节列出了本书中所参考的文档。

- 『Net.Data 技术库』
- 『相关文档』

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
<ul style="list-style-type: none">• <i>Net.Data 管理与程序设计指南, OS/390 版</i>• <i>Net.Data 管理与程序设计指南, OS/2、Windows NT 和 UNIX 版</i>• <i>Net.Data 管理与程序设计指南, OS/400 版</i>	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。
<i>Net.Data 参考</i>	描述 Net.Data 宏语言、变量和内部函数。
<i>Net.Data 语言环境接口参考</i>	描述 Net.Data 语言环境接口。
<i>Net.Data 消息和代码参考</i>	列出 Net.Data 错误消息和返回码。

相关文档

在使用 Net.Data 及其相关产品时, 以下文档可能会有用:

- *DB2 for OS/400 SQL Programming*, SC41-5611
- *OS/400 Distributed Database Programming*, SC41-5702

另外, 可以从以下 URL 获得 OS/400 的文档和红皮书(包括有关 DB2 的书籍):

<http://publib.boulder.ibm.com/html/as400/infocenter.html>

附录B. Net.Data 示例宏

这个示例宏应用程序显示了一张雇员列表，应用程序用户可以通过在列表中选择雇员的姓名来获取某个雇员的额外信息。此宏使用 SQL 语言环境来查询 EMPLOYEE 表，从中获取雇员姓名和某个特定雇员的有关信息。

此宏文件使用一个包含文件，其中包含用于该宏的 DEFINE 块。

第110页的图9显示了示例宏。第112页的图10显示了包含文件。

```

%{***** Sample Macro *****)
*   FileName = sqlsamp1.d2w
*   Description:
*       This Net.Data macro queries...
*       - The EMPLOYEE table to create a selection list of
*         employees for display at a browser
*       - The EMPLOYEE table to obtain additional information
*         about an individual employee
*
*****%}
%{*****
*   Include for global DEFINES -
*****%}
%INCLUDE "sqlsamp1.hti"
%{*****
*   Function: queryDB           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable and
*   creates a selection list from the result. The value of the variable
*   myTable is specified in the include file sqlsamp1.hti.
*****%}
%FUNCTION(DTW_SQL) queryDB() {
    SELECT FIRSTNME FROM $(myTable)
%MESSAGE {
    -204: {<p><b>ERROR -204: Table $(myTable) not found. </b>
        <p>Be sure the correct include file is being used.</b>
        %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}

    %REPORT {
<select name=emp_name>
%ROW{
<option>$(V1)
%}
</select>
%}
%}

%{*****
*   Function: fname           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable for
*   additional information about the employee identified by the
*   variable emp_name.
*****%}
%FUNCTION(DTW_SQL) fname(){
    SELECT FIRSTNME, PHONENO, JOB FROM $(myTable) WHERE FIRSTNME='$(emp_name)'
%MESSAGE {
    -204: "Error -204: Table not found "
    -104: "Error -104: Syntax error"
    100: "Warning 100: No records" : continue
    +default: "Warning $(RETURN_CODE)" : continue
    -default: "Unexpected SQL error" : exit
%}
%}

```

图 9. 示例宏 (1/3)


```
%{ *****
* HTML block: INPUT Title: Dynamic Query Selection *
*
* Description: Queries the EMPLOYEE table to create a selection list of
* the employees for display at the browser *
*****%}
%HTML(INPUT) {
<html>
<head>
<title>Generate Employee Selection List</title>
</head>
<body>
<h3>$(exampleTitle)</h3>
<p>This example queries a table and uses the result to create
a selection list using a <em>%REPORT</em> block.
< hr>
<form method="post" action="report">
@queryDB(<input type="submit" value="Select Employee">
</form>
< hr>
</body>
</html>
%}
```

图 9. 示例宏 (2/3)

```
%{ *****
* HTML block: REPORT *
* Description: Queries the EMPLOYEE table to obtain additional information *
* about an individual employee *
*****%}
%HTML(REPORT){
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<PRE>
@fname()
</PRE>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}

%{ End of Net.Data macro 1 %}
```

图 9. 示例宏 (3/3)

```

=====
%{***** Include File *****)
*   FileName = sqlsamp1.hti
*   Description:
*       This include file provides global DEFINES for the sqlsamp1.d2w
*       Net.Data macro.
*****%}
%define {
    emp_name    = ""
    reposition = sign
    exampleTitle = "Sample Macro"
    myTable = "EMPLOYEE"
    DATABASE = "sample"
%}

%{    End of include file  %}

```

图 10. 包含文件

附录C. 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。

本书中任何对非 IBM Web 站点的引用都只是为了提供方便，并不以任何形式作为对那些 Web 站点的保证。那些 Web 站点的资料并不是本 IBM 产品资料的一部分，使用那些 Web 站点的风险将由您自己承担。

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
W92/H3
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
_U.S.A.

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 国际程序许可证协议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。

所有关于 IBM 未来方向或意向的声明都可能更改或撤消，而不作任何通知，并且仅代表目标和结果。

此信息仅供用于计划。在描述的产品可用之前，此处的信息可能会更改。

此信息中包含了每日商业操作中所使用的数据和报表的例子。为了尽可能完整地进行说明，例子中包含了个人、公司、商标以及产品的名称。所有这些名称都是虚构的，如果与实际商业企业使用的名称和地址相似，则纯属巧合。

商标

以下术语是 IBM 公司在美国或其他国家的注册商标:

AIX	语言环境
AS/400	MVS/ESA
DB2	Net.Data
DB2 Universal Database	OS/2
DRDA	OS/390
DataJoiner	OS/400
IBM	OpenEdition
IMS	

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。

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。

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 和 GWAPI。

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

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

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

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。

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, 并控制对其存储的数据进行访问的一个软件系统。

DATALINK. 一种 DB2 数据类型, 允许从数据库中对存储在数据库外部的文件进行逻辑引用。

data type (数据类型). 列和字面量的属性。

DBCLOB. Double-byte character large object 的缩写, 双字节字符大型对象。

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。

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。

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。

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。

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 和更为高级的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。

[A]

安全性

- 防火墙 23
- 概述 23
- 权限 26
- 权限审批 25
- 网络加密 25
- 语言环境 72
- 指定访问权 20, 72
- Net.Data 机制 26

[B]

保护资产 23

报表

- 缺省 64
- 用一个函数调用生成多个 64

报表变量 51

报表格式, 定制 63

变量

- 报表 51
- 表格 49
- 表格处理 51
- 定义 43
- 动态生成的引用 44
- 动态生成名称 44
- 环境 46
- 记号大小 41
- 可执行 47
- 类型 41, 46
- 列表 49
- 配置, 语句
 - 初始化文件 8
 - 电子邮件 SMTP 服务器 (DTW_SMTP_SERVER) 11
 - 电子邮件 SMTP 字符集 (DTW_SMTP_CHARSET) 10
 - 电子邮件 SMTP CCSID (DTW_SMTP_CCSSID) 10
 - 宏高速缓存尺寸 (DTW_MACRO_CACHE_SIZE) 8
 - 禁用 SHOWSQL (DTW_SHOWSQL) 9
 - 启用 SHOWSQL (DTW_SHOWSQL) 9
 - 说明 8
 - 用空格填充参数 (DTW_PAD_PGM_PARMS) 9
 - SMTP 服务器 (DTW_SMTP_SERVER) 11
 - SMTP 字符集 (DTW_SMTP_CHARSET) 10

变量 (续)

配置, 语句 (续)

- SQL 隔离 (DTW_SQL_ISOLATION) 11
- SQL 命名方式 (DTW_SQL_NAMING_MODE) 12
- Web 注册表关闭 (DTWR_CLOSE_REGISTRIES) 12

说明 41

条件 46

隐藏 48

引用 44

语言环境 52

杂项 50

作用域 42

标识符作用域 42

表 31, 33

调用 Net.Data 31, 34, 98

在 Web 页面中调用 Net.Data 33

表格变量 49

表格处理变量 51

表格函数 60

[C]

持久性函数 60

持久性宏 95

初始化文件

创建 6, 7

格式 6

更新 6

路径语句 13

配置变量语句 8

说明 6

ENVIRONMENT 语句 16

处理结果集, 存储过程 89

传递参数 78, 88, 93

存储过程 88

直接调用语言环境 73

Java 应用程序语言环境 77

REXX 程序 78

System 语言环境 93

传递参数时的公共错误 75, 86

创建初始化文件 7

词汇表 114

从程序返回值 75

存储过程 86, 87, 88, 89, 90

步骤 87

处理结果集 89

传递参数 88

从宏调用 86

单个结果集 89

存储过程 (续)
 多个结果集 90
 缺省报表 89, 90
 有效的数据类型 87
 REPORT 块 89, 90
错误条件, 语言环境 72

[D]

打印, 禁用缺省报表 63
大型对象 (LOB) 82, 83
 说明 82
 有效格式 83
调用 77, 78, 86, 87, 92
 程序, 直接调用 73
 程序, System 92
 存储过程 86, 87
 函数 56
 语言环境 72
 Java 应用程序 76
 REXX 程序 77, 78
调用 Net.Data 31
 表 31, 34, 98
 概述 31
 链 31, 34, 98
 使用宏 31
 使用 CGI 31
 HTML 模块 61
 URL 31, 34, 98
定义变量
 查询字符串数据 44
 DEFINE 语句或块 43
 HTML 表 SELECT, INPUT, 以及 TEXTAREA 标记 44
动态生成变量名 44
多报表块 64

[F]

防火墙 23
访问权
 对于语言环境 72
 对于 Net.Data 文件 20
访问 DB2 80
复制 Net.Data 程序对象
 到多个库 6
 到 CGI-BIN 库 5

[G]

改进性能 105
高速缓存宏, 高速缓存尺寸 8

格式化数据输出 62

[H]

函数 86
 表格 60
 持久性的 60
 调用 56
 调用存储过程 86
 定义 52
 平面文件 60
 数学 59
 说明 52
 通用 58
 用户定义的 52
 字 59
 字符串 59
 FUNCTION 块语法 52
 MACRO_FUNCTION 块语法 53
 Web 注册表 60
函数调用
 内部 57
 语法 56
宏
 变量 41
 标识符作用域 42
 呈示部分 37
 持久性的 95
 函数 52
 开发 37
 块 39
 剖析 38
 生成 HTML 61
 声明部分 37
 示例 38
 说明 1
 条件逻辑 67
 循环 69
 在...中或在...之间游历 41
 DEFINE 块 39
 FUNCTION 块 39
 HTML 块 40
 IF 块 67
 WHILE 块 69
宏的一部分
 呈示 37
 声明 37
宏请求 31
 例子 31
 语法 31
环境变量 46

[J]

记号大小 41
加密, 网络 25
结果集 89, 90
 处理, 存储过程 89
 单个 89
 多个 90
 缺省报表 90
 准则和限制 66

[K]

可执行变量 47
块, 宏 39

[L]

类型, 变量 46
链 31, 32
 调用 Net.Data 31, 34, 98
 在 Web 页面中调用 Net.Data 32
列表变量 49
路径语句
 保护资产 26
 更新准则 13
 在初始化文件中进行配置 13
 DTW_JAVA_CLASSPATH 16
 EXEC_PATH 14
 FFI_PATH 15
 HTML_PATH 16
 INCLUDE_PATH 15
 MACRO_PATH 13

[P]

配置变量语句
 说明 8
 在初始化文件中进行配置 8
 DTWR_CLOSE_REGISTRIES 12
 DTW_MACRO_CACHE_SIZE 8
 DTW_PAD_PGM_PARMS 9
 DTW_SHOWSQL 9
 DTW_SMTP_CCSSID 10
 DTW_SMTP_CHARSET 10
 DTW_SMTP_SERVER 11
 DTW_SQL_ISOLATION 11
 DTW_SQL_NAMING_MODE 12
配置 Net.Data
 初始化文件
 创建 6
 更新 6
 路径语句 13

配置 Net.Data (续)
 初始化文件 (续)
 配置变量语句 8
 说明 6
 ENVIRONMENT 语句 16
 对 Net.Data 文件的访问权 20
 概述 5
 设置语言环境 18
平面文件函数 60

[Q]

启动 Net.Data 31
全局标识符作用域 42
权限
 安全性 26
 指定对 Net.Data 文件的访问权 20
权限审批, 安全性 25
缺省报表 89, 90
 打印 63
 为存储过程指定 89, 90

[S]

声明部分, 宏结构 37
示例宏 107
事务处理 95
首部信息, REPORT 块 63
数据类型 82, 84, 87
 用于存储过程 87
 直接调用 74
 DATALINK 84
 LOB 82
数学函数 59

[T]

条件
 变量 46
 逻辑, IF 块 67
通用函数 58

[W]

文件, 指定对 Net.Data 的访问权 20

[X]

性能 79
 优化语言环境 105
 REXX 环境 79
 REXX 语言环境 105

性能 (续)

SQL 语言环境 106

System 语言环境 106

循环, WHILE 块 69

[Y]

隐藏变量

保护资产 26

隐藏变量名 48

引用变量 44

用户定义的函数 52

用户定义的语言环境, ENVIRONMENT 语句 7

遍历, 在宏中和在宏之间 41

语言环境 77, 92

安全性 72

变量 52

处理错误条件 72

调用 72

例子 16

配置 ENVIRONMENT 语句 16

设置 18

在初始化文件中进行配置 16

支持 72

直接调用 73

Java 应用程序 76

REXX 77

SQL 80

System 92

[Z]

杂项变量 50

在结果集中编码 DataLink URL 84

执行命令 92

执行 SQL 语句 80

直接调用语言环境

传递参数 73

传递参数时的公共错误 75

从程序返回值 75

调用程序 73

概述 73

受支持的数据类型 74

注脚信息, REPORT 块 63

注意事项 113

字处理函数 59

字符串函数 59

作用域, 标识符

宏 42

全局 42

FUNCTION 块 42

REPORT 块 42

ROW 块 43

B

BLOB 82

C

CGI-BIN 库, 复制 Net.Data 程序对象 5

CLOB 82

D

DATALINK 数据类型 84

编码 URL 84

DataLink 文件管理器 84

DBCLOB 82

DEFINE 块

定义变量 43

说明 39

DTWR_CLOSE_REGISTRIES 12

DTW_DEFAULT_REPORT 64

DTW_DIRECTCALL 73

DTW_JAVAPPS 76

DTW_JAVA_CLASSPATH 16

DTW_MACRO_CACHE_SIZE 8

DTW_PAD_PGM_PARMS 9

DTW_REXX 77

DTW_SHOWSQL 9

DTW_SMTP_CCSID 10

DTW_SMTP_CHARSET 10

DTW_SMTP_SERVER 11

DTW_SQL 80

DTW_SQL_ISOLATION 11

DTW_SQL_NAMING_MODE 12

DTW_SYSTEM 92

E

ENVIRONMENT 语句

参数表 17

服务程序 17

例子 17

说明 16

用于用户定义的语言环境 7

语法 17

语言环境类型 17

在初始化文件中进行配置 16, 17

F

FFI_PATH 15

FUNCTION 块

标识符作用域 42

调用函数 56

FUNCTION 块 (续)

格式化输出 62

说明 39

H

HTML 31, 32, 33

表 31, 33

调用 Net.Data 31, 34, 98

关于 33

SELECT, INPUT, 和 TEXTAREA 标记, 定义变量

44

表格的标记 63

块

处理 62

调用 Net.Data 61

例子 61

说明 40

链 31, 32

调用 Net.Data 31, 34, 98

关于 32

未被识别的数据 62

在宏中生成 61

FORM Submit 按钮 62

HTML_PATH 16

I

IF 块 67

INCLUDE_PATH 15

J

Java 应用程序语言环境

传递参数 77

调用程序 76

概述 76

设置 18

L

LOB (大型对象) 82

受支持的类型 82

SQL 和 ODBC 语言环境 82

M

MACRO_FUNCTION 块

调用函数 56

语法 53

MACRO_PATH 13

MESSAGE 块

处理 55

MESSAGE 块 (续)

例子 56

说明 55

语法 55

作用域 55

N

Net.Data

安全性机制 26

调用 31

概述 1

宏, 开发 37

配置 5

文件, 访问权 20

Net.Data 程序对象

复制到多个库 6

复制到 CGI-BIN 库 5

Net.Data 宏。请参阅宏。 1

R

REPORT 块 89, 90

存储过程 89, 90

多个 64

多个的准则 66

格式化数据输出 62

例子 64

缺省报表 64

首部和注脚信息 63

说明 62

限制 66

作用域 42

RETURN_CODE 变量 55, 72

REXX 语言环境 77, 78, 79

传递参数 78

调用程序 78

概述 77

AIX 上的性能 79

ROW 块, 标识符作用域 43

S

SQL

隔离配置变量 11

命名方式配置变量 12

SQL 语句, 执行 80

SQL 语言环境

传递参数时的公共错误 86

概述 80

设置 18

执行 SQL 语句 80

SQLCODE 72
System 语言环境 92, 93
 传递参数 93
 调用程序 92
 发出命令 92
 概述 92

U

URL 31
 调用 Net.Data 31, 34, 98
 定义变量 44

W

Web 注册表函数 60
Web 注册表, 关闭变量 12
WHILE 块 69



Printed in China