IBM® Net.Data®
OS/2® 版、Windows NT®版和 UNIX® 版

管理与编程指南

版本 7

IBM® Net.Data®
OS/2® 版、Windows NT®版和 UNIX® 版

管理与编程指南

版本 7

注意

在使用本资料及其所支持的产品之前，务必阅读第227页的『声明』中的信息。

2001 年 6 月版

这一版适用于 IBM Net.Data OS/2 版、Windows NT 版和 UNIX 版（ “DB2 通用数据库” 版本 7.2 的一个功能部件）以及所有后续的发行版和修订版，直到在新版本中另有声明为止。

通过您当地的 IBM 代表或 IBM 分部可订购出版物，或者，通过致电 1-800-879-2755（在美国）或 1-800-IBM-4YOU（在加拿大）来订购出版物。

目录

前言

感谢您选择 Net.Data®，IBM™ 的开发工具来创建动态的 Web 页面！使用 Net.Data 之后，您就可以迅速地开发具有动态内容的 Web 页面，这只要通过结合来自广泛种类数据源的数据并使用您已知的编程语言的功能即可实现。

关于 Net.Data

借助 Net.Data，您可同时使用关系和非关系数据库管理系统 (DBMS)（包括 DB2、IMS 和启用 ODBC 的数据库以及用各种编程语言（如 Java、JavaScript、Perl、C、C++ 和 REXX）编写的应用程序来创建动态 Web 页面。

Net.Data 是一个宏处理器，它在 Web 服务器上作为中间件执行。您可以编写称之为宏的 Net.Data 应用程序， 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、其他数据源、现有商业逻辑以及您在宏中所设计的其他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 Perl 脚本、C、C++ 或 REXX 程序中封装商业逻辑。
- 访问诸如 DB2 等数据库
- 访问其他数据源，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。为了改进性能，Net.Data 还支持各种 “Web 服务器应用程序编程接口”(API)。Net.Data 系列产品在 OS/400、OS/390、Windows NT、AIX、OS/2、HP-UX、Sun Solaris、Linux 和 Dynix/PTX 操作系统上提供了类似的功能。 Net.Data 还支持多个操作系统上的 FastCGI 和主要的 Web 服务器应用程序编程接口 (API)。

一个图形管理工具可以帮助您管理 AIX、Windows NT 和 OS/2 操作系统上的 Net.Data 配置设置。管理工具还可以辅助您为使用“现场连接”的数据库连接指定安全性。

为了帮助您方便地从数据库访问数据，Net.Data 提供了各种各样的工具，包括 NetObjects Fusion 插件和基于 Java 开发的向导。这些工具在 Java 环境中与 Net.Data Java 小应用程序一起使用，允许您创建可以在操作系统间移植的应用程序。NetObjects Fusion 插件允许您使用 NetObjects Fusion Web 开发工具来构建使用来自关系数据源的动态数据的复杂应用程序。Net.Data 向导提供了一个图形工具，它将指导您创建基本的 Net.Data 宏。

版本 7.2 中的新增内容

Net.Data 版本 7.2 除提供了 Net.Data 先前发行版的所有功能之外，还提供了许多其他功能！Net.Data OS/2 版、Net.Data Windows NT 版以及 Net.Data UNIX 版在版本 7.2 中提供了以下附加的功能部件:

- 从其他 SQL 函数的 REPORT 和 ROW 块调用 SQL 函数的能力，如在使用“现场连接”时调用嵌套式 SQL 函数。
- 通过利用 DB2 高速缓存语句改进了性能。现在可将参数标记放入宏的 SQL 语句中，这使您能够有效地利用语句高速缓存。
- 新增 Net.Data 内置函数: DTWF_COPY()、DTWF_EXISTS()、DTWF_WRITEFILE()。
- 直接调用 Java 函数而不需要在 AIX、Sun、Windows NT 和 Linux 上使用“现场连接”的能力。
- 支持 Linux S/390
- 支持 Oracle 存储过程
- 支持 Net.Data 跟踪

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 宏语言或 Java 小服务程序来开发宏。您将了解如何使用 Net.Data 提供的用来访问 DB2 数据库的语言环境、IMS 事务，以及使用 Java、REXX、Perl、 和其他编程语言来访问数据。

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详情，可以从以下 World Wide Web 站点获得:

http://www.ibm.com/software/data/net.data/

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 表标记）。

Net.Data Reference 中描述了 Net.Data 宏语言、变量、内置函数以及操作系统的区别。

关于本书中的示例

本书中出现的示例相对较为简单，目的是演示特定的概念，而不说明 Net.Data 构造元的所有用法。某些示例只是一些片段，需要其他代码才能运行。

如何发表意见

您的反馈意见有助于 IBM 提供高质量的信息。欢迎您发表有关本书或其他 DB2 文档的任何意见。可使用下列任何一种方法来提供意见:

- 将您的意见用电子邮件发送至 db2pubs@vnet.ibm.com，并在电子邮件中注明产品名称、产品的版本号和书号。如果对特定文本有意见，请列出文本的位置（例如，章节标识、页码或帮助主题标题）。
- 从 Web 发送您的意见。访问以下 Web 站点:

  http://www.ibm.com/software/db2os390

  该 Web 站点上具有可用来发送意见的反馈信息页。
- 完成本书背后的读者意见表，并用电子邮件、传真（800-426-7773，用于美国和加拿大）返回或交给 IBM 代表。
- 邮寄打印和使用下一页上的读者意见表。要打印该表，从**服务**下拉菜单选择**打印**或**复制**。输入*意见*作为要打印或复制的主题。将完成的表寄至:

  IBM Corporation, Department W92/H3
  P.O. Box 49023
  San Jose, CA 95161-9023
  U.S.A.

  如果是从美国以外的国家发送该表，则将该表交给您当地的 IBM 分部或 IBM 代表邮寄。

- 传真打印和使用本书末尾的读者意见表，并将其传真至以下美国号码: 800-426-7773 或 (408) 463-4393。要打印该表，遵循“邮件”中的指示信息。

第1章 介绍

Net.Data 是一种服务器端脚本语言，它通过允许动态生成使用来自各种数据源的数据的 Web 页来扩展 Web 服务器。数据源可能包括关系和非关系数据库管理系统，如 DB2、启用 DRDA 的数据库和平面文件数据。可使用 Net.Data 的简单但功能强大的脚本语言来快速地构建应用程序。通过支持调用使用各种编程语言（包括 Java、C/C++、REXX 和其他语言）编写的应用程序，Net.Data 允许重复使用现存的商业逻辑。

本章描述 Net.Data 以及选择将其用于 Web 应用程序的原因。

- 『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 宏您可以执行编程逻辑、访问和处理变量、调用函数、使用报告生成工具。宏是包含 Net.Data 语言构造的文本文件，这些构造用来构建可包含 HTML、XML、Javascript 和语言环境语句（如 SQL 和 Perl）的应用程序。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。

第2页的图1说明了 Net.Data、Web 服务器以及支持的数据和编程语言环境之间的关系。

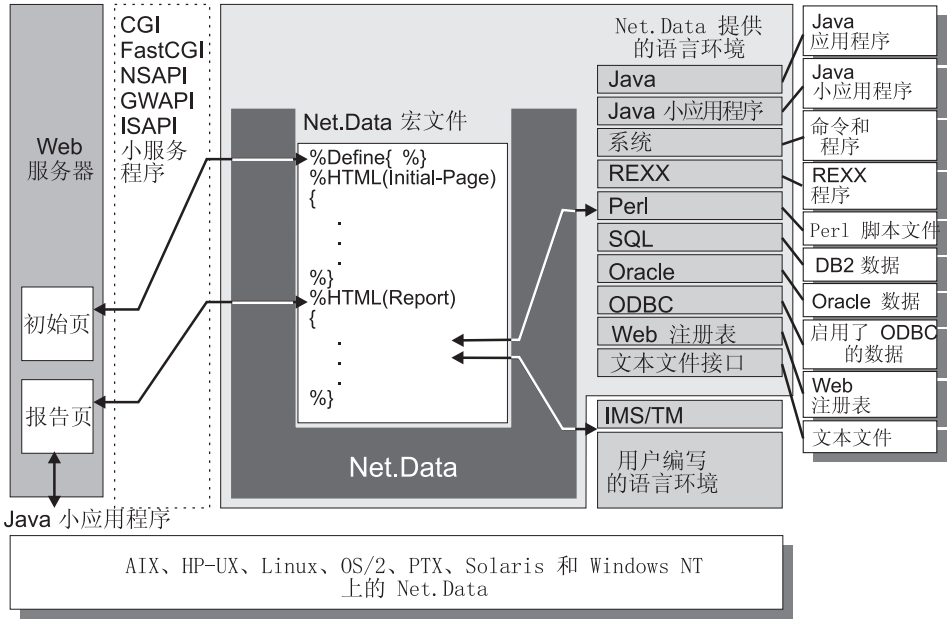


图 1. Net.Data、Web 服务器和支持的数据以及程序源之间的关系

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 DLL 或共享程序库调用，从而将它作为 CGI、FastCGI 或 Web 服务器应用程序编程接口 (API) 来调用。此 URL 中含有特定于 Net.Data 的信息，包括要处理的宏或者要直接调用的 SQL 语句或程序。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、Perl、REXX 和 JavaScript。Net.Data 还提供了访问 DB2 数据库、执行使用 IMS Web 的 IMS 事务或使用 REXX、Perl 和其它用于您的应用程序的语言的语言环境。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，允许您快速开发因特网和内部网应用程序。

- 允许 Web 应用程序中数据生成逻辑和呈示逻辑的分离。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示技术来方便地更改数据的呈示方式。
- 通过提供与 C、C++、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术和商业逻辑来生成 Web 页面。
- 通过使用简单的宏语言，提供了快速开发复杂的因特网应用程序的能力。
- 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迁移。

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，那么在下一次指定执行该宏的 URL 时将可以立即看到您所作的任何更改。不需要重新编译。

直接请求

需要执行简单的 SQL 语句、DB2 存储过程、REXX 程序、C 或 C++ 程序、Perl 脚本的简单请求，不需要创建宏。这些请求可以在从浏览器流向 Web 服务器的 URL 内直接指定。

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 将所有的数据都看作字符串。Net.Data 使用内置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93页的『Net.Data 宏变量』中详细讨论。

内置函数

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置函数。其它内置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，带有说明的出错信息将会返回给客户。您可以在出错信息返回到用户之前在浏览器中定制它们。参见第13页的『配置变量语句』和 *Net.Data Reference* 以获取详情。

第2章 配置 Net.Data

您可以通过使用产品附带的自述文件中的指示信息来为操作系统安装 Net.Data。大部分的配置步骤在安装过程中完成；这是根据操作系统而有所不同的。

在为您的操作系统安装 Net.Data 之后，修改 Web 服务器和 Net.Data 配置。配置任务包括下列一项或多项任务:

- 定制 Net.Data 初始化 (INI) 文件
- 为 CGI、FastCGI、其中一个受支持的 Web 服务器 API（可选）或者 Net.Data 小服务程序配置 Net.Data。
- 定制 Web 服务器配置和环境变量文件
- 配置高速缓存管理器（可选）
- 配置“现场连接”（可选）
- 设置 Net.Data 语言环境
- 指定访问权

使用以下工具来配置 Net.Data:

- 一个文本编辑器

  在所有的操作系统上，使用一个文本编辑器来编辑初始化文件和“现场连接”以及高速缓存管理器配置文件。您还要使用文本编辑器来更新所有的 Web 服务器配置文件。在更改这些文件之前进行备份是个不错的主意。

- Net.Data 管理工具

  管理工具提供了一个图形界面，用于定制初始化文件和“现场连接”配置文件。您可以使用管理工具在 OS/2、Windows NT 和 AIX 操作系统上配置 Net.Data。

您所使用的方法取决于需要配置的组件以及运行 Net.Data 的操作系统，如表1中的描述。如果您使用一个特定的方法来启动一个配置任务，那么您应继续使用这种方法，以获取最佳结果。

表 1. 配置方法(包括任务和操作系统)的比较。 **A** - *可以使用管理工具进行配置，或人工配置。* **M** - *只可以人工配置。*

任务	操作系统:			
	AIX	**NT**	**OS/2**	**HP|SUN|Linux**
配置 Net.Data INI 文件	A	A	A	M

*表 1. 配置方法(包括任务和操作系统)的比较。 **A** - 可以使用管理工具进行配置，或人工配置。**M** - 只可以人工配置。 (续)*

任务	操作系统:			
	AIX	**NT**	**OS/2**	**HP|SUN|Linux**
定义 cliette 端口	A	A	A	M
定义 cliette	A	A	A	M
启用 cliette 口令加密	A	A	N/A	M
启用错误记录	A	A	A	M
为 FastCGI、CGI 和 API 配置 Web 服务器*	M	M	M	M
定义高速缓存管理器端口	M	M	N/A	N/A
配置高速缓存管理器	M	M	N/A	N/A

***提示**: 许多 Web 服务器都提供了管理工具，您可以使用这些工具来配置 Web 服务器。

本章将描述如何配置 Net.Data 以及如何修改 Web 服务器的配置以便与 Net.Data 一起使用。另外，还将描述如何配置可选的组件。

- 第12页的『定制 Net.Data 初始化文件』
- 第28页的『设置 Net.Data 语言环境』
- 第32页的『配置“现场连接”』
- 第36页的『为使用 CGI 而配置 Web 服务器』
- 第38页的『为 FastCGI 配置 Net.Data』
- 第40页的『为配合 Java 小服务程序使用而配置 Net.Data』
- 第41页的『为使用 Web 服务器 API 而配置 Net.Data』
- 第44页的『使用 Net.Data 管理工具来配置 Net.Data』
- 第57页的『对 Net.Data 访问的文件授予访问权限』

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:

- 把字符数据的编码方式指定为 Unicode
- 字符串和单词函数是否支持 MBCS
- 用于访问数据库数据的 DB2 实例的名称
- Net.Data 如何连接到语言环境、数据库、连接管理和高速缓存并与它们通信

- 是否激活了错误记录

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、REXX 程序和 Perl 脚本)的目录路径。

Net.Data 初始化文件 db2www.ini 位于 Web 服务器的文档目录中。参见操作系统的自述文件以了解详情。

权限提示: 确保 Web 服务器执行所使用的用户标识具有读取此文件的权限。参见第57页的『对 Net.Data 访问的文件授予访问权限』，以获取详情。

关于可选组件的 Net.Data 配置文件

以下章节将讨论 Net.Data 可选组件的配置文件。

『“现场连接”配置文件』

『高速缓存管理器配置文件』

第8页的『Net.Data 初始化、控制和宏文件的公用节』

“现场连接”配置文件

“现场连接”在 Windows NT、OS/2、AIX、Linux 和 Sun Solaris 操作系统上提供了连接管理，从而通过消除启动开销来改进性能。“Net.Data 现场连接”配置文件中包含有关一个或多个已经命名的 cliette 的信息。cliette 是一个维护数据库连接的长时间运行的进程，或者是一个在多用户调用 Net.Data 宏期间持续存在的 Java 应用程序。cliette 启动之后，它就将一直存在，直至“Net.Data Live 现场连接”终止。多个 cliette 可以连接至单个数据库。

作为配置文件中 cliette 信息的一部分，您需要指定 cliette 名称，以及进程的最小个数和最大个数。对于数据库 cliette，您还可以为每个 cliette 条目指定数据库名称、登录以及口令。

权限提示: 确保启动“连接管理器”的用户标识具有读取此文件的权限。参见第57页的『对 Net.Data 访问的文件授予访问权限』，以获取详情。

高速缓存管理器配置文件

高速缓存管理器配置文件中包含对高速缓存管理器和每个高速缓存的定义。Net.Data 高速缓存在 第172页的『Net.Data 高速缓存』中描述。配置高速缓存管理器在第177页的『配置高速缓存管理器和 Net.Data 高速缓存』中描述。文件的结构是由一系列的段或节组成的:

高速缓存管理器节

这一节定义高速缓存管理器本身的参数，包括网络信息、记录状态以及跟踪状态。它需要而且必须标记为高速缓存管理器。

高速缓存定义节

这些节定义了每个高速缓存的参数；对于高速缓存管理器所管理的每个高速缓，在配置文件中都存在一个高速缓存定义节；这一部分包含网络信息、内存以及空间需求、记录状态和统计状态。对于高速缓存管理器所管理的每个高速缓存，都需要这个高速缓存定义节。

高速缓存管理器配置文件不是由管理工具来管理的，并且可以使用任何文本编辑器来进行修改。参见 第172页的『Net.Data 高速缓存』，以学习如何定义这个文件。

权限提示: 请确保启动高速缓存管理器的用户标识对这个文件具有访问权。参见第57页的『对 Net.Data 访问的文件授予访问权限』以获取详情。

Net.Data 初始化、控制和宏文件的公用节

为了能够使 Net.Data 的所有组件能够作为一个整体来工作，Net.Data 初始化、配置和宏文件中的某些部分必须一致。下表概述了这些文件中必须匹配的区域。

表 2. 对于 Net.Data 配置文件和宏的一致性需求

文件	公用节	注释
Net.Data INI 文件	环境语句	使用“现场连接”的语言环境必须在环境语句中指定数据库 cliette 名称
	“现场连接”配置变量	使用“Net.Data 现场连接”时，需要指定“现场连接”端口 DTW_CM_PORT。这个变量值必须与“现场连接”配置文件中的 MAIN_PORT 值相匹配。
	高速缓存配置变量	使用 Net.Data 高速缓存时，可以选择包含端口号和机器名变量。这些值必须和高速缓存管理器配置文件中所使用的那些值相匹配(如果使用的话)。
“现场连接”配置文件	Cliette 定义	每个 cliette 定义都必须与 INI 文件中相应的定义匹配。另外，MAIN_PORT 值必须与 INI 文件中的 DTW_CM_PORT 变量值匹配。

表 2. 对于 Net.Data 配置文件和宏的一致性需求 (续)

文件	公用节	注释
高速缓存管理器配置文件	高速缓存管理器配置变量	使用 Net.Data 高速缓存时，可以选择包含端口号和机器名变量。这些值必须和 INI 文件中所使用的那些值相匹配(如果使用的话)。

以下几段说明了宏、Net.Data 初始化文件和“现场连接”配置文件之间的关系。两个 cliette 是由宏使用的 (DTW_SQL:SAMPLE、DTW_SQL:CELDIAL)，它们访问两个名为 SAMPLE 和 CELDIAL 的数据库。“现场连接”配置文件中包含 cliette 的名称和定义。Net.Data 初始化文件中的 ENVIRONMENT 语句指 cliette 的名称。LOGIN 和 PASSWORD 的值是在“现场连接”配置文件中指定的。

图2显示了包含 @DTW_ASSIGN 语句的宏段，该语句定义在访问数据库时使用哪个 cliette。

```
<3**********************************************>
<3** This is an HTML comment                    **>
<3** Access the SAMPLE database using          **>
<3** cliette DTW_SQL:SAMPLE                    **>
<3**********************************************>
@DTW_ASSIGN (DATABASE, ″ SAMPLE ″)
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)


<3**********************************************>
<3** This is an HTML comment                    **>
<3** Process the CELDIAL database using         **>
<3** the cliette DTW_SQL:CELDIAL **>
<3**********************************************>
@DTW_ASSIGN (DATABASE, ″ CELDIAL ″)
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)
```

图 2. Net.Data 宏段

请注意，DATABASE 配置变量将替换初始化文件的 ENVIRONMENT 语句，从而生成 cliette 的名称。这允许您从同一个宏访问多个数据库。

图3显示了包含 ENVIRONMENT 语句和相关 cliette 类型的 Net.Data 初始化文件段。对于初始化文件中的每个 cliette 类型，都有一个 ENVIRONMENT 语句。对于每个数据库 cliette 类型， ENVIRONMENT 语句指定一个 cliette 名称。这个名称是由 cliette 的类型和一个变量引用 $(DATABASE) 组成的，该变量引用是在运行时解析的。每个使用 “现场连接” 的语言环境都必须在 ENVIRONMENT 语句中有一个 cliette 的定义。

```
ENVIRONMENT (DTW_SQL)
    (IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,
    ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
    CLIETTE "DTW_SQL:$(DATABASE)"
```

图 3. Net.Data 初始化文件段

第11页的图4显示了一段 “现场连接” 配置文件，其中包含对 DTW_SQL:CELDIAL 和 DTW_JAVAPPS 的 cliette 定义。

```
    CONNECTION_MANAGER{
    MAIN_PORT=7128
    ENCRYPTION=key
}

    ##################################################################
    # This is a comment in a Live Connection configuration file.
    # Comments start with a pound (hash) character.
    # Comments terminate at the end of the line and do not continue to
    # the next line unless another pound (hash) character is specified.
    # You can include comments at the end of lines containing Live
    # Connection keywords except on password lines.
    # You cannot include comments anywhere on lines containing the
    # password keyword.
    # You cannot include spaces and pound (hash) characters within any
    # name, such as cliette name or in database cliette passwords.
   ##################################################################
    CLIETTE  DTW_SQL:CELDIAL{
    MIN_PROCESS=1
    MAX_PROCESS=5
    EXEC_NAME=./dtwcdb2
    DATABASE=CELDIAL
    LOGIN=marshall
    PASSWORD=encrypted_password
    }

    CLIETTE  DTW_JAVAPPS{
    MIN_PROCESS=1
    MAX_PROCESS=5
    EXEC_NAME=./launchjv
    }
```

图 4. “现场连接”配置文件段

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13页的『配置变量语句』
- 第21页的『路径配置语句』
- 第26页的『环境配置语句』

图5中所示的示例初始化文件包含这些语句的示例，且对于 OS/2 和 Windows NT 是有效的。

每个独立配置语句的文本都必须在同一行中。请确保对于您从宏中调用的每个语言环境，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。

```
1  DTW_CM_PORT 7128
2  DTW_INST_DIR c:\db2www
3  DTW_LOG_DIR c:\db2www\logs
4  DB2INSTANCE DB2
5  DTW_DIRECT_REQUEST NO
6  DTW_SHOWSQL NO
7  DTW_UNICODE NO
8  DTW_MBMODE NO
9  MACRO_PATH c:\DB2WWW\Macro
10 HTML_PATH  c:\www\html
11 INCLUDE_PATH c:\db2www\Macro
12 EXEC_PATH c:\db2www\Macro
13 FFI_PATH c:\pub\ffi;pub\ffi\data
14 ENVIRONMENT (DTW_SQL)      [DLL path]   [Parameter list]
15 ENVIRONMENT (DTW_ORA)      [DLL path]   [Parameter list]
16 ENVIRONMENT (DTW_ODBC)     [DLL path]   [Parameter list]
17 ENVIRONMENT (DTW_DEFAULT)  [DLL path]   [Parameter list]
18 ENVIRONMENT (DTW_APPLET)   [DLL path]   [Parameter list]
19 ENVIRONMENT (DTW_REXX)     [DLL path]   [Parameter list]
20 ENVIRONMENT (DTW_PERL)     [DLL path]   [Parameter list]
21 ENVIRONMENT (DTW_SYSTEM)   [DLL path]   [Parameter list]
22 ENVIRONMENT (DTW_FILE)     [DLL path]   [Parameter list]
23 ENVIRONMENT (DTW_WEBREG)   [DLL path]   [Parameter list]
24 ENVIRONMENT (DTW_JAVAPPS)  [DLL path]   [Parameter list]
25 ENVIRONMENT (HWS_LE)       [DLL path]   [Parameter list]
```

- 第 1 - 8 行定义配置变量
- 第 9 - 13 行定义处理宏所需的文件的路径
- 第 14 - 25 行定义可用的环境语句。

图 5. Net.Data 初始化文件. 有关“DLL 路径”和“参数列表”的完整描述，请参考 db2www.ini 文件本身和第26页的『环境配置语句』。

以下章节将描述如何在初始化文件中定制配置语句。

- 第13页的『配置变量语句』
- 第21页的『路径配置语句』

需要进行以下 ENVIRONMENT 语句的更改:

- 从任何出现 RETURN_CODE 变量的 ENVIRONMENT 语句的参数列表中除去该变量。
- 除去 DTW_DEFAULT、DTW_FILE 和 DTW_APPLET ENVIRONMENT 语句。

因为某些配置缺省值已更改，所以应该考虑下列更改:

- 如果应用程序需要使用变量 SHOWSQL，则将 DTW_SHOWSQL 配置变量更改为 YES。参见第18页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』中的语法和示例。
- 如果应用程序需要使用直接请求调用，则将 DTW_DIRECT_REQUEST 配置变量更改为 YES。参见第16页的『DTW_DIRECT_REQUEST: 启用直接请求变量』中的语法和示例。
- 如果不想将 Net.Data 错误日志或 Net.Data 跟踪存储在 /usr/lpp/netdata/logs/ 中，则将 DTW_ERROR_LOG_DIR 或 DTW_TRACE_LOG_DIR 配置变量更改为适当的目录。

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。其他变量控制要构造的 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和数据库，以及其他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:

- 第14页的『高速缓存管理器配置变量』
- 第15页的『DB2INSTANCE: DB2 实例变量』
- 第15页的『DTW_CM_PORT: “现场连接”端口号变量』
- 第16页的『DTW_DEFAULT_ERROR_MESSAGE: 指定类属出错信息』
- 第16页的『DTW_DIRECT_REQUEST: 启用直接请求变量』
- 第16页的『DTW_INST_DIR: Net.Data 安装目录变量』

- 第16页的『HTML_PATH』
- 第17页的『DTW_LOG_DIR 和 DTW_LOG_LEVEL: 错误日志变量』
- 第17页的『DTW_LOG_LEVEL: 错误日志级别变量』
- 第18页的『DTW_MBMODE: 本机的语言支持变量』
- 第18页的『DTW_REMOVE_WS: 用于除去额外空格的变量』
- 第18页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』
- 第19页的『DTW_SMTP_SERVER: 电子邮件 SMTP 服务器变量』
- 第19页的『DTW_UNICODE: Unicode 变量』
- 第20页的『DTW_UPLOAD_DIR』
- 第21页的『DTW_USE_DB2_PREPARE_CACHE』
- 第21页的『DTW_VARIABLE_SCOPE: 变量作用域变量』

高速缓存管理器配置变量

如果高速缓存管理器不是在 Net.Data 宏所运行的机器上运行，则将使用两个可选的配置变量:

- DTW_CACHE_PORT 指定 Net.Data 使用哪个端口号连接至高速缓存管理器。
- DTW_CACHE_HOST 指定本地或远程机器的 TCP/IP 主机名。

如果高速缓存管理器在本地机上运行，那么 UNIX 域套接字或已命名管道将用于通信，并且不需要进行配置。

高速缓存管理器只在 AIX 和 Windows NT 机器上运行。 参见 第172页的『Net.Data 高速缓存』，以了解 Net.Data 高速缓存。

DTW_CACHE_PORT: 高速缓存管理器端口变量

指定高速缓存管理器正在监听的 TCP/IP 端口。此端口号必须与高速缓存管理器配置文件中指定的端口号相匹配，这样 Net.Data 就可以与高速缓存管理器通信。如果没有指定，高速缓存管理器将使用缺省的端口 7175。

语法:

```
DTW_CACHE_PORT [=] port_number
```

参数:

port_number

为高速缓存管理器分配的唯一的端口号，用于服务高速缓存请求。缺省值为 7175。

表3描述了为这些变量指定机器标识和端口号的选项。

表 3. 高速缓存管理器配置变量: 配置选项

缺省连接管理器值	如果指定了高速缓存机器 ...	如果没有指定高速缓存机器 ...
如果指定了高速缓存端口...	Net.Data 使用指定的端口在指定的机器上连接至高速缓存管理器。	Net.Data 使用指定的端口在本地机上连接至高速缓存管理器。
如果没有指定高速缓存端口 ...	Net.Data 使用缺省的端口 7175 在指定的机器上连接至高速缓存管理器。	Net.Data 使用缺省的端口 7175 在本地机上连接至高速缓存管理器。

DTW_CACHE_HOST: 高速缓存管理器机器标识变量

指定高速缓存管理器所驻留的机器。如果没有指定, Net.Data 将假定它就是本地机。

语法:

```
DTW_CACHE_HOST [=] host_name
```

参数:

host_name

运行高速缓存管理器的本地或远程机器的限定 TCP/IP 主机名。缺省值是本地机的主机名。

DB2INSTANCE: DB2 实例变量

指定 SQL 语言环境所使用的 DB2 实例。当 Net.Data 连接到在 Windows NT、OS/2 和 UNIX 操作系统上运行的 DB2 时需要这个变量值。

OS/2、Windows NT 和 UNIX 操作系统上的 DB2 需要将 DB2INSTANCE 定义为一个环境变量。如果 Net.Data 检测到 DB2INSTANCE 没有定义为环境变量, 那么它将把 DB2INSTANCE 环境变量设置为试图连接到 DB2 之前在初始化文件中找到的 DB2INSTANCE 的值。

语法:

```
DB2INSTANCE [=] instance_name
```

DTW_CM_PORT: “现场连接”端口号变量

指定 Net.Data 用于“现场连接”的唯一的端口号。

语法:

```
DTW_CM_PORT [=] port_number
```

其中 *port_number* 指定了用于“现场连接” 的唯一的端口号。

DTW_DEFAULT_ERROR_MESSAGE: 指定类属出错信息

使用 DTW_DEFAULT_ERROR_MESSAGE 配置变量来为处于生产状态的应用程序指定类属出错信息。此变量为任何 MESSAGE 块中未捕捉的错误状态提供了一个类属消息。

如果您仍希望查看 Net.Data 生成的实际出错信息，可使用出错信息日志来捕捉这些消息。参见 第197页的『第8章 Net.Data 记录』，以了解如何使用错误日志。

如果未指定配置变量，则 Net.Data 会对错误状态显示它自己提供的消息。

语法:

```
DTW_DEFAULT_ERROR_MESSAGE [=] "message"
```

示例: 指定类属消息

```
DTW_DEFAULT_ERROR_MESSAGE "This site is temporarily unavailable."
```

DTW_DIRECT_REQUEST: 启用直接请求变量

启用或禁用 Net.Data 直接请求调用。在缺省情况下，直接请求被禁用。

调用 Net.Data 的直接请求方式允许用户指定 SQL 语句的执行，或直接在 URL 中指定 Perl、REXX 或 C 程序。禁用直接请求的时侯，用户必须使用宏请求方式来调用 Net.Data，允许用户只执行那些 SQL 语句和已经定义的或在宏中调用的函数。参见第62页的『使用 Net.Data 机制』，以获取使用 DTW_DIRECT_REQUEST 时与安全性相关的建议。

语法:

```
DTW_DIRECT_REQUEST [=] YES|NO
```

其中:

YES 启用 Net.Data 直接请求。

NO 禁用 Net.Data 直接请求。NO 是缺省值。

DTW_INST_DIR: Net.Data 安装目录变量

在 Net.Data 执行过程中定位某些文件。您可以在安装时设置这个变量来指定主目录 <*inst_dir*> (Net.Data 安装在这个目录中)。安装之后不要更改这个值。

HTML_PATH

指定 Net.Data 将大对象 (LOB) 写入哪个目录。

安装期间，Net.Data 将创建一个名为 tmplobs 的目录，该目录在 HTML_PATH 路径配置变量指定的目录下。Net.Data 将所有 LOB 文件存储在这个目录中。如果更改了 HTML_PATH 的值，则将在新的目录下创建一个新的子目录。

语法:

```
HTML_PATH [=] path
```

示例: 以下示例显示了初始化文件中的 HTML_PATH 配置变量。

```
HTML_PATH /db2/lobs
```

当查询返回一个 LOB 时，Net.Data 将把它保存在 HTML_PATH 配置变量中指定的目录中。

提示: 当使用 LOB 时应考虑系统限制，因为它们会很快地消耗资源。参见第135页的『使用大对象』以获取详情。

DTW_LOG_DIR 和 DTW_LOG_LEVEL: 错误日志变量

DTW_LOG_DIR 指定存储错误日志的目录。除非同时设置了此变量和 DTW_LOG_LEVEL 变量，否则将不记日志。

参见第197页的『记录 Net.Data 出错信息』，以了解关于这些变量和记录 Net.Data 的出错信息日志的详情。

语法:

```
DTW_LOG_DIR [=] \inst_dir\path
```

示例: 初始化文件配置

```
DTW_LOG_DIR \inst_dir\mylogfiles\
```

DTW_LOG_LEVEL: 错误日志级别变量

DTW_LOG_LEVEL 指定要记录在错误日志中的错误的级别。除非同时设置了此变量和 DTW_LOG_DIR 变量，否则将不记日志。

参见第197页的『记录 Net.Data 出错信息』，以了解关于这些变量和记 Net.Data 的出错信息日志的详情。

语法:

```
DTW_LOG_LEVEL [=] off|warning|error
```

示例: 初始化文件配置

```
DTW_LOG_LEVEL error
```

DTW_MBMODE: 本机的语言支持变量

对字处理和字符串函数激活国家语言支持。当这个变量的值为 YES 时，所有的字符串函数和字处理函数都将通过把字符串作为混合数据（即，作为可能同时包含来自单字节字符集和双字节字符集的字符的字符串）来正确地处理 MBCS 字符。缺省值为 NO。您可以通过在 Net.Data 宏中设置 DTW_MBMODE 变量来覆盖初始化文件中值的设置。

此配置变量与 DTW_UNICODE 配置变量一起使用。如果 DTW_UNICODE 使用缺省值 NO，则将使用 DTW_MBMODE 的值。如果对 DTW_UNICODE 所设置的值不是 NO，则使用它自己的值。表4说明了这两个变量的设置如何确定内置函数处理字符串的方式:

表 4. DTW_UNICODE 和 DTW_MBMODE 设置之间的关系

如果 DTW_UNICODE 设置为	如果 DTW_MBMODE=YES	如果 DTW_MBMODE=NO
NO	支持 MBCS 与 SBCS 混合使用	仅支持 SBCS
UTF8	支持 UTF-8	支持 UTF-8

语法:

```
DTW_MBMODE [=] NO|YES
```

DTW_REMOVE_WS: 用于除去额外空格的变量

如果此变量设置为 YES，则 Net.Data 会除去 HTML 输出中的多余空白。通过压缩空白，这个变量压缩了要发送给 Web 浏览器的数据量，从而改进了性能。缺省值为 NO。

可以在宏中使用 DEFINE 语句来覆盖这个变量。

语法:

```
DTW_REMOVE_WS [=] YES|NO
```

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5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 5. 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_SERVER: 电子邮件 SMTP 服务器变量

指定使用 DTW_SENDMAIL 内置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。

性能提示: 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。

示例:

```
DTW_SMTP_SERVER us.ibm.com
```

DTW_UNICODE: Unicode 变量

指定 Net.Data 在以下事物中是否支持 Unicode:

- 宏
- 表数据
- 从 DB2 数据库中检索得到的数据
- Net.Data 内置函数处理的字符串

Net.Data 在宏、表数据以及内置函数中支持 UTF-8 Unicode 格式，并且输出通常是 UTF-8。Net.Data 可以访问包含 UCS-2 数据的数据库，并将这些数据转换为 UTF-8 格式。

当设置为 UTF8 时，DTW_UNICODE 会告诉 Net.Data 在 Unicode 环境中运行。然后 Net.Data 生成 UTF-8 格式的页面，并期望任何输入数据都是 UTF-8 格式的（或者，如果是 DB2 数据库数据的话，UCS-2 也可以）。输入数据包括宏文件的内容、从浏览器发送的表单数据，以及所有其他来自外部数据源的数据。

DB2 Unicode 数据库需求：除了设置 DTW_UNICODE 变量之外，还需在 Net.Data 的运行环境中将 DB2 特定环境变量 DB2CODEPAGE 设置为 1208。例如，对于 Apache Web 服务器，向 HTTPD.CONF 文件添加下面这一行:

```
SetEnv DB2CODEPAGE 1208
```

参见 Web 服务器文档，以确定如何为 CGI 脚本、Web 服务器 API、Fast-CGI 程序或小服务程序设置环境变量。

当在 Unicode 环境中运行时，Net.Data 使用的是英文信息目录。

DTW_UNICODE 配置变量与 DTW_MBMODE 配置变量一起使用。在处理单词或字符串内置函数时，DTW_UNICODE 配置变量的值将覆盖 DTW_MBMODE 变量的设置。但是，如果 DTW_UNICODE 被设置为 NO 或未设置，则使用 DTW_MBMODE 的值。第18页的表4说明了这两个变量的设置如何确定内置函数处理字符串的方式:

语法:

```
DTW_UNICODE [=] NO|UTF8
```

其中:

NO 指定遵从 DTW_MBMODE 变量的值。第18页的表4描述了根据 DTW_MBMODE 的值对 Net.Data 的支持

UTF8 指定支持 UTF-8 代码页并忽略 DTW_MBMODE 配置变量的值。UTF-8 由可变的字节数来表示字符，对 ASCII 字符没有影响。

DTW_UPLOAD_DIR

指定 Net.Data 将客户机上载的文件存储在哪个目录中。未设置此变量时，Net.Data 将不接受文件上载。

语法:

```
DTW_UPLOAD_DIR [=] path
```

示例:

```
DTW_UPLOAD_DIR /tmp/uploads
```

DTW_USE_DB2_PREPARE_CACHE

指定 Net.Data 应该利用 DB2 准备高速缓存，而不应在宏文件的 SQL 语句中显式地使用参数标记。当您想要所有宏都利用此功能部件时，在 Net.Data 初始化文件中将 DTW_USE_DB2_PREPARE_CACHE 配置变量设置为 “是”。要只为特定宏中的语句激活此功能部件，可以使用 DTW_USE_DB2_PREPARE_CACHE 宏变量。有关详情，参见 *Net.Data Reference*。

语法:

```
DTW_USE_DB2_PREPARE_CACHE [=] YES|NO
```

其中:

是 指定 Net.Data 将修改所有 SQL 语句以利用准备高速缓存。可以通过使用 %DEFINE 或 @DTW_ASSIGN() 将宏变量设置为 “否” 来对特定 SQL 语句禁用此功能部件。

否 指定 Net.Data 让 SQL 语句保持不变。这是缺省值。

DTW_VARIABLE_SCOPE: 变量作用域变量

指定 Net.Data 如何对待局部变量的作用域: 局部变量是否仍为局部变量，或者局部变量是否可在创建它们的函数块外部使用。此变量的作用是提供与 Net.Data 先前版本的向后兼容性，但不可用于 Net.Data 的 OS/390 或 OS/400 版本。

语法:

```
DTW_VARIABLE_SCOPE [=] LOCAL|GLOBAL
```

其中:

LOCAL

指定局部变量仍然为局部变量。这种特性是在 Net.Data 版本 2.0 中出现的，是缺省情况。

GLOBAL

指定局部变量可在创建该变量的函数块之外使用。它的作用是提供与 Net.Data 先前版本的向后兼容性; LOCAL 是建议的设置值。

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:

- 第22页的『 DTW_ATTACHMENT_PATH 』
- 第22页的『 EXEC_PATH 』
- 第23页的『 FFI_PATH 』

- 第24页的『INCLUDE_PATH』
- 第25页的『MACRO_PATH』

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LOB 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。每个路径语句的描述中都注明了例外情况。

- 用分号 (;) 分隔路径语句中指定的每个目录。
- 每个路径语句都可以指定多个路径，但是 HTML_PATH 除外，它只能有一个路径语句。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。这些示例可能和您的应用程序不同，这取决于操作系统和配置。

DTW_ATTACHMENT_PATH

此路径配置语句指定用来定位要使用 DTW_SENDMAIL 来发送的附件的路径。

语法:

```
DTW_ATTACHMENT_PATH [=] path
```

示例:

```
DTW_ATTACHMENT_PATH /usr/lpp/internet/server_root/pub/upload
```

EXEC_PATH

此路径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。目录在路径语句中的顺序确定了 Net.Data 搜索目录的顺序。如果找到程序，则将外部程序名附加到路径规范后，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
```

示例: 以下示例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 /u/user1/prgms;/usr/lpp/netdata/prgms;
```

Net.Data 宏:

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

如果在 /usr/lpp/netdata/prgms 目录中找到文件 myFunction.cmd，则程序的限定名为 /usr/lpp/netdata/prgms/myFunction.cmd。

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，如果提交了以下 URL:

  ```
  http://myserver/cgi-bin/db2www/usr/user1/prgms/myFunction.cmd
  ```

  Net.Data 将在 /u/user1/prgms/myFunction.cmd 目录路径中搜索文件。
- 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，如果提交了以下 URL:

  ```
  http://myserver/cgi-bin/db2www/myFunction.cmd/report
  ```

  并且在 EXEC_PATH 指定的所有目录中都没有找到文件 myFunction.cmd，那么 Net.Data 将试图在当前工作目录中查找该文件。

FFI_PATH

此路径配置语句标识了一个或多个 Net.Data 搜索的目录，以它们指定的顺序进行搜索，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
```

示例: 以下示例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，因此对于没有找到的 FFI 文件提供了特殊措施。参见 *Net.Data 参考*中有关 FFI 内置函数一节。

INCLUDE_PATH

此路径配置语句标识了一个或多个 Net.Data 搜索的目录，以它们指定的顺序进行搜索，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规范后面，以便产生限定的包含文件名。

语法:

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

例 1: 以下示例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: 以下示例显示了 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，如果提交了以下 URL:

  ```
  http://myserver/cgi-bin/db2www/u/user1/includes/oeheader.inc
  ```

  Net.Data 将在 /u/user1/includes/oeheader.inc 目录路径中搜索文件。

- 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，如果提交了以下 URL:

  ```
  http://myserver/cgi-bin/db2www/my.cmd/report
  ```

  并且在 INCLUDE_PATH 指定的所有目录中都没有找到文件 myFunction.cmd，那么 Net.Data 将试图在当前工作目录中查找该文件。

MACRO_PATH

此路径配置语句标识了 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.dtw 的 Net.Data 宏:

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

语法:

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

等号 (=) 是可选的，由方括号指出。

Net.Data 将路径 /macro/sqlm.d2w/report 追加至 MACRO_PATH 配置语句中的路径中，从左至右，直至 Net.Data 找到宏为止。如果找不到宏，则 Net.Data 将执行为 DTW_DEFAULT_MACRO 配置变量定义的宏，否则它将打印错误。参见第69页的『第4章 调用 Net.Data』以获取有关调用 Net.Data 宏的信息。

示例: 以下示例显示了初始化文件中的 MACRO_PATH 语句以及调用 Net.Data 的相关链接。

Net.Data 初始化文件:

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

HTML 链接:

```
<a href="http://server/cgi-bin/db2www/query.dtw/input">Submit another query.</a>
```

如果在目录 /u/user1/macros 中找到文件 *query.dtw*，则全限定路径为 /u/user1/macros/query.dtw。

如果在 MACRO_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，如果提交了以下 URL:

  ```
  http://server/cgi-bin/db2www/u/user1/macros/myfile.txt/report
  ```

  Net.Data 将在 /u/user1/macros/myfile.txt 目录路径中搜索文件。

- 如果指定的路径是相对路径，那么 Net.Data 将从根 (/) 目录开始，在所有的目录中搜索文件。例如，如果提交了以下 URL:

  ```
  http://server/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`，则 Net.Data 将试图在根 (/) 目录中查找文件: `/myfile.txt`

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127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。

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送到语言环境。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。

要点: 如果在宏中定义了一个变量，但未在 ENVIRONMENT 语句中加以指定，则该宏变量不会被传送至语言环境。

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 的值必须传送到 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。要将变量传送到语言环境，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, ...) [CLIETTE "cliette_name"]
```

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*

  DLL 或共享程序库的名称中包含 Net.Data 调用的语言环境接口。

  – 在 AIX 中，共享程序库的名称是用扩展名 *.o* 指定的。
  – 在 HP-UX 中，共享程序库的名称是用扩展名 *.sl* 指定的
  – 在 SUN 和 LINUX 中，共享库的名称是使用 *.so* 扩展名指定的
  – 在 OS/2 和 Windows NT 中，共享库名是使用 *.dll* 扩展名指定的。

- *parameter_list*

  在每个函数调用中传送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送变量，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。以下示例指定 ENVIRONMENT 语句中的变量:

```
ENVIRONMENT(DTW_SQL) C:\WINNT\System32\nddb2.dll(IN
 DATABASE,TRANSACTION_SCOPE,USERID,PASSWORD)
```

  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。

- *cliette_name*

  cliette 的名称。*cliette_ name* 可以指 Java 应用程序语言环境 cliette，也可以是数据库 cliette。 *cliette_name* 参数是和 CLIETTE 关键字一起使用的，它们都只和“现场连接”一起使用。CLIETTE 和 *cliette_name* 是可选的，只能对数据库和 Java 应用程序语言环境指定。

  Java 应用程序 cliette

  这个 cliette 名称指定 Java 应用程序语言环境。

  语法:

```
CLIETTE "DTW_JAVAPPS"
```

  数据库 cliette

  这个 cliette 名称指定一个与数据库相关联的 cliette。

  语法:

```
CLIETTE "type:db_name"
```

  参数:

type 与 cliette 关联的数据库语言环境。参见第54页，以获取有效类型的列表。

db_name

数据库 cliette 名称。这个名称通常与 cliette 所关联的数据库相同，例如 MYDBASE，但也可以是另一个名称。在使用 Oracle 语言环境时，*db_name* 是可选的。

Net.Data 处理初始化文件时，它不会装入语言环境 DLL 或共享程序库。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 DLL 或共享程序库。然后，只要 Net.Data 是装入的， DLL 或共享程序库将保持装入状态。

示例: 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，请向 ENVIRONMENT 语句中添加需要从初始化文件传送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。

```
ENVIRONMENT (DTW_SQL)  /net.data/lib/dtwsql.so  ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
  CLIETTE "DTW_SQL:MYDBASE"
ENVIRONMENT (DTW_ORA)  /net.data/lib/dtwora.so  ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ODBC) /net.data/lib/dtwodbc.so ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, ALIGN, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_APPLET)  /net.data/lib/dtwjava.so    ( )
ENVIRONMENT (DTW_JAVAPPS) /net.data/lib/dtwjavapps.so ( OUT RETURN_CODE )
  CLIETTE "DTW_JAVAPPS"
ENVIRONMENT (DTW_PERL)    /net.data/lib/dtwperl.so    ( OUT RETURN_CODE )
ENVIRONMENT (DTW_REXX)    /net.data/lib/dtwrexx.so    ( OUT RETURN_CODE )
ENVIRONMENT (DTW_SYSTEM)   dtwsys.so     ( OUT RETURN_CODE )
ENVIRONMENT (HWS_LE)       dtwhws.so     ( OUT RETURN_CODE )
```

必需: 每个 ENVIRONMENT 语句必须在单独一行上。

设置 Net.Data 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:

- 『设置带有 Cliette 的 Java 语言环境』
- 第29页的『设置 Oracle 语言环境』

设置带有 Cliette 的 Java 语言环境

如果“现场连接”用来管理与“Java 虚拟机”的连接，则在可以从宏调用函数之前，需要对 Java 语言环境进行一些附加设置:

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

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

2. 在“现场连接”配置文件 `dtwcm.cnf` 中定义一个 cliette 来与 Java 语言环境一起工作。对带有 EXEC_NAME 配置变量的 cliette 指定批处理文件名。在下面的示例中，Java cliette 名称被定义为 DTW_JAVAPPS，EXEC_NAME 配置变量被设置为批处理文件的名称 `launchjv.bat`:

   ```
   CLIETTE DTW_JAVAPPS{
   MIN_PROCESS=1

   MAX_PROCESS=1

   EXEC_NAME=launchjv.bat

   }
   ```

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

3. 通过将 cliette 名称添加到语句中，来更新 Net.Data 初始化文件 `db2www.ini` 中的 DTW_JAVAPPS ENVIRONMENT 语句。例如:

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

设置 Oracle 语言环境

使用以下步骤从 Net.Data 宏访问 Oracle 数据库:

1. 确保已经安装了适当的 Oracle 组件，并能如下工作:
   a. 在安装 Net.Data 的机器上安装 SQL*Net (如果尚未安装的话)。有关的详情，参见以下 URL:

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

   b. 验证使用 Oracle *tnsping* 函数时可以采用与 Web 服务器所使用的相同的安全性权限。要验证这一点，可用您的用户标识登录到 Web 服务器，然后输入:

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

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

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

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

```
SELECT * FROM tablename
```

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

故障诊断: 如果上述步骤失败，请不要继续。如果有一步失败，请检查您的 Oracle 配置。

2. 确保在您的 Web 服务器处理中已经正确设置了 Oracle 环境变量。
   - 对于 AIX，在 /etc/environment 文件中，或在运行 Web 服务器所用的用户标识的 .profile 文件中放置下列各行:

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

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

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

   提示: 对于其他的 Oracle 环境变量，您可能还需要添加一些行，这取决于您计划使用的 Oracle 程序，例如: 国家语言支持和两阶段提交。有关这些环境变量的详情，请参考 Oracle 管理文档。

3. 从 Net.Data 测试与 Oracle 的连接。在 Net.Data 宏中，请在 LOGIN 和 PASSWORD 变量中指定适当的值。在访问 Oracle 数据库时，不要定义 Net.Data DATABASE 变量。下面是宏中连接语句的一个示例:

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

   本地 Oracle 实例:

   如果您只访问本地的 Oracle 实例，则不要指定远程 Oracle 实例的名称作为登录用户标识的一部分，如下面的示例所示:

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

   现场连接:

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

```
CLIETTE DTW_ORA:{
MIN_PROCESS=1
MAX_PROCESS=3
EXEC_NAME=./dtwora8
DATABASE=not_used
LOGIN=userid@remote_oracle_instance_name
PASSWORD=password
}
```

提示: 不要对 Oracle 指定 DATABASE 变量。

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

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

同样，您可以从 Net.Data 宏直接执行 *tnsping*，如下面的示例所示:

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

故障诊断:

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

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

如果验证步骤仍然失败，请与 IBM 的服务部门联系。

示例:

完成了访问验证步骤之后，可以使用宏中的函数来调用 Oracle 语言环境，如下例所示:

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

配置“现场连接”

“现场连接”管理数据库和 Java 应用程序连接，从而改进 Net.Data 在 Windows NT、 OS/2、AIX 和 Sun Solaris 操作系统上的性能。通过使用“连接管理器”和维护开放连接的 cliette、进程， “现场连接”消除了连接到数据库或启动 Java 虚拟机的启动开销。

“现场连接”使用配置文件 dtwcm.cnf 来确定需要启动哪几个 cliette。它包含管理信息和对“现场连接”所使用的每个 cliette 的定义。参见第168页的『管理连接』来更多地了解“现场连接”。

图6中所示的示例配置文件包含了以下类型的信息:

- “连接管理器”端口信息
- 用于 DB2 连接的 SQL cliette 信息
- Java 应用程序 cliette 信息

```
 1  CONNECTION_MANAGER{
 2  MAIN_PORT=7100
 3  }
 4
 5  CLIETTE DTW_SQL:CELDIAL{
 6  MIN_PROCESS=1
 7  MAX_PROCESS=5
 8  EXEC_NAME=./dtwcdb2
 9  DATABASE=CELDIAL
10  LOGIN=marshall
11  PASSWORD=stlpwd
12  }
13
14  CLIETTE DTW_JAVAPPS{
15  MIN_PROCESS=1
16  MAX_PROCESS=5
17  EXEC_NAME=./javaapp
18  }
```

- 1 - 3 行是配置文件所必需的，定义与“现场连接”一起使用的唯一的端口号。
- 5 - 12 行定义所有的数据库 cliette，标识了 cliette 名称、要运行的进程个数、数据库名称以及 cliette 可执行文件。您可以包含一些附加信息，例如连接到 DB2 数据库的用户标识和口令。
- 14 - 18 行定义所有用于 Java 应用程序的 cliette，标识了 cliette 名称、要运行的进程个数、唯一的端口号以及 cliette 可执行文件。

图 6. “现场连接”配置文件

开始之前: 在定制“现场连接”配置文件之前，请先阅读执行步骤后面的提示和技巧部分。

配置“现场连接”端口:

您对 MAIN_PORT 选择的值就是将要首先使用的端口号。“现场连接”可使用的端口号可以通过 MAIN_PORT 的设置和每个 cliette 的 MAX_PROCESSES 计算出来。在装入时，“现场连接”从 MAIN_PORT 中指定的编号开始分配端口，并逐渐递增，直到达到累积的 MIN_PROCESSES 为止。根据需要，它将继续装入端口，直到达到 MAX_PROCESSES 为止。使用的最大端口号是 MAX_PROCESSES 设置的和。

例如，在第32页的图6中的配置中，分配的端口号将是 7100、7101 和 7102，然后根据需要增加至 7110。

要点:

- 请与系统管理员一起进行检查，以确保您计划使用的端口号是可用的。
- 确保 MAIN_PORT 的值应与 Net.Data 初始化文件中 DTW_CM_PORT 的值相匹配。

***配置数据库 cliette**:*

1. 输入 cliette 环境语句。

   ```
   CLIETTE type:db_name
   ```

   参数:

   type 使语言环境和 cliette 关联的名称。参见第54页，以获取有效类型的列表。

   db_name

   数据库 cliette 的名称，通常与 cliette 关联的数据库同名，例如 MYDBASE；当然，*db_name* 也可以是另一个名称。在使用 Oracle 语言环境时，*db_name* 是可选的。

2. 确定 MIN_PROCESS 和 MAX_PROCESS 的值。MIN_PROCESS 指定了启动“连接管理器”时要启动的进程个数。随后，如果同时到达其他的请求，则“连接管理器”将启动更多的 cliette，根据需要添加 cliette，直至到达为 MAX_PROCESS 指定的值。

   输入 MIN_PROCESS 和 MAX_PROCESS 语句:

   ```
   MIN_PROCESS=min_num
   MAX_PROCESS=max_num
   ```

   参数:

   min_num

   在启动“连接管理器”时要启动的 cliette 进程的个数。对于这个数量的 cliette，您必须有足够多可用的、唯一的端口号。

max_num

可以同时运行的 cliette 的最多个数。对于这个数量的 cliette，您必须有足够多可用的、唯一的端口号。

3. 指定 cliette 可执行文件的名称。此文件名的指定如下:

```
EXEC_NAME=./dtwcdbtypeid
```

其中，*dbtypeid* 是数据库类型标识符。参见表6，以获取有效的可执行文件名:

表 6. Cliette 可执行文件名

Cliette 描述	Cliette 类型	名称		平台有效性					
		UNIX	Windows NT 或 OS/2	AIX	NT	OS/2	HP	SUN	PTX
DB2 进程 cliette	DTW_SQL	dtwcdb2	dtwcdb2.exe	是	是	是	是	是	否
ODBC 进程 cliette	DTW_ODBC	dtwcodbc	dtwcodbc.exe	是	是	否	否	否	否
Oracle 进程 cliette	DTW_ORA	dtwcora	dtwcora.exe	是	是	否	否	否	否

4. 指定 cliette 所关联的数据库的名称:

```
DATABASE=db_name
```

其中 *db_name* 是 cliette 所关联的数据库的名称；例如 MYDBASE。

5. 可选: 将 LOGIN 和 PASSWORD 变量的缺省值更改为 *USE_DEFAULT，这样，Net.Data 就可以使用启动 “连接管理器” 时使用的用户标识连接到 DB2 数据库。通过指定这些缺省值，您可以避免将这一信息放在配置文件中。例如，用以下几行来代替第32页的图6中示例配置文件中的 14 和 15 行:

```
 LOGIN=*USE_DEFAULT
PASSWORD=*USE_DEFAULT
```

提示: 如果您在配置文件中定义多个 cliette 条目，则您可以对某个特定的数据库指定不同的数据库登录和口令。

配置 Java 应用程序 cliette:

1. 输入 cliette 环境语句:

```
CLIETTE DTW_JAVAPPS
```

2. 确定 MIN_PROCESS 和 MAX_PROCESS 的值。MIN_PROCESS 指定了启动“连接管理器”时要启动的进程个数。随后，如果有同时的请求到达，则“连接管理器”将启动更多的 cliette，根据需要添加 cliette，直至到达为 MAX_PROCESS 指定的值。

   输入 MIN_PROCESS 和 MAX_PROCESS 语句。

   ```
   MIN_PROCESS=min_num
   MAX_PROCESS=max_num
   ```

   参数:

   min_num

   在启动“连接管理器”时启动的 cliette 进程的个数。对于这个数量的 cliette，您必须有足够多可用的、唯一的端口号。

   max_num

   可以同时运行的附加 cliette 的最多个数。对于这个数量的 cliette，您必须有足够多可用的、唯一的端口号。

配置“现场连接”的提示和技巧:

- “连接管理器”使用 cliette 名称来唯一地标识一系列 cliette。
- 对于数据库 cliette，您必须确保对每个计划访问的数据库都有一个已命名的 cliette 集合。对于很少访问的数据库，您可以将 cliette 的 MIN 和 MAX 个数设置为 1。同样，您还可以将 MIN 设置为 0，这意味着在对 cliette 进行 Net.Data 请求之前不会启动进程。
- cliette 的 NAME 必须与初始化文件中用于 cliette 类型的 ENVIRONMENT 语句所引用的 cliette 名称一致。cliette 名称中可以包含变量，如果是数据库 cliette，那么它应该包含变量引用 $(DATABASE)。ENVIRONMENT 语句中用于 cliette 名称的缺省值是 DTW_SQL:$(DATABASE)。您可以在初始化文件中使用变量引用，但不能在“现场连接”配置文件中使用变量引用。

  DATABASE 变量在 Net.Data 宏中定义。当宏中遇到 SQL 语句时，Net.Data 初始化文件中的 $(DATABASE) 变量引用将被 DATABASE 的当前值代替。

  您可以使用这个方法来访问多个数据库。如果您想要在 Net.Data 宏中访问三个数据库(例如，D1、D2 和 D3)，并且初始化文件中有标准的 CLIETTE ″DTW_SQL:$(DATABASE)″ 行，那么您在“现场连接”配置文件中需要这样的三个部分:

  ```
  CLIETTE DTW_SQL:D1{ ...}
  CLIETTE DTW_SQL:D2{....}
  CLIETTE DTW_SQL:D3{....}
  ```

- 进程被启动，但不停止。如果您将最大进程个数设置为 M，并且任意时刻 M 个进程是同时使用的，在关闭“连接管理器”之前它们将保持活动状态，这样您就不希望 MAX_PROCESS 的值这么高，为了启动很少使用的进程而用尽了所有的系统资源。

  建议：请尝试对 MIN_PROCESS 和 MAX_PROCESS 使用不同的值，看看哪一个能在您的系统上发挥最佳性能。如果“连接管理器”接收到的请求超过了指定的最大值，最后一个请求将被排队，直至某个 cliette 完成了它的处理。当有一个 cliette 可用时，排队的那个请求将被处理。这种将请求排队的过程对于应用程序用户来说是透明的。
- 您可以对不同名称的部分使用同一个 cliette。例如，配置文件中所有的 DB2 数据库部分都使用相同的 cliette 类型。但是两个部分不能有相同的名称。

如果您使用 CGI，并且只希望几个数据库使用“现场连接”，则只要在配置文件中列出您所希望的数据库即可。如果 Net.Data 在处理 Net.Data 宏时遇到了 SQL 函数，它将向“连接管理器”询问某个特定的 cliette。如果“连接管理器”没有该类型的 cliette，它将以一个 NO_CLIETTE_AVAIL 信息作为应答。然后，Net.Data 用一个 DLL 版本来处理该请求。

确认“连接管理器”服务自动启动:

在 Windows NT 上，您可以指定将“连接管理器”作为 Windows NT 的服务启动，而不是从命令行启动。将“连接管理器”作为 Windows NT 的服务运行可以使“连接管理器”在每次启动机器时自动启动。

要点：在将“连接管理器”设置为自动启动之前，先从命令行启动，以确保“现场连接”配置文件是正确的。

- 从 Windows NT 任务栏，选择**开始 -> 设置 -> 控制面板 -> 服务**。
- 选择 **Net.Data 连接管理器**，然后单击**启动**按钮。
- 选择**自动启动类型**，然后单击**确定**。

为使用 CGI 而配置 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。

将 Net.Data 配置为一次仅使用一个接口。例如，若将 Web 服务器配置为使用 CGI 执行 Net.Data，就不要还将 Web 服务器配置为使用另一接口执行 Net.Data。若您想稍后运行使用另一接口（如 FastCGI）的 Net.Data，则单独针对新接口重新配置 Web 服务器。

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。

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。

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。

一般 Web 服务器参数设置

将 Web 服务器配置为设置和/或传送 Net.Dat 所需的环境变量。各语言环境和操作系统在处理环境变量的方式上有所不同。请参见 Web 服务器文档以了解用于您的环境和平台的精确语法。

libpath LIBPATH 环境变量应包含至（包含 Net.Data 共享库或出现在 Net.Data 初始化文件中的 ENVIRONMENT 语句中的 DLL 的）目录的路径。若访问 DB2，则 LIBPATH 变量应包含至 DB2 库目录的路径。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv LIBPATH /u/mydir/myserver/lib:/u/mydir/myserver:
  /usr/lpp/db2_07_01/lib:/usr/lib
```

oracle_home

在使用 Oracle 时所必需。Oracle 数据库可执行文件的路径和目录。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv ORACLE_HOME /home.native/oracle/product/8.1.5
```

oracle_sid

在使用 Oracle 时所必需。Oracle 数据库的实例。您必须对 Oracle 使用“现场连接”。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv ORACLE_SID mvpdb2
```

db2instance

在使用 DB2 时所必需。DB2 数据库的实例。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv DB2INSTANCE wwwinst
```

REXX_owner_pid

在 AIX 上使用 REXX 时所必需。这个性能变量是在 AIX 操作系统上和 FastCGI 以及 REXX 一起使用的。缺省值为 0。对于其他的产品和操作系统，则在 Net.Data 宏中说明此变量。参见第209页的『附录B. Net.Data for AIX』，以获取有关此变量的详情。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv RXQUEUE_OWNER_PID 0
```

lang UNIX 语言环境变量。对于美国英语使用 En_US。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv LANG En_US
```

NLSPATH

指定信息目录的目录位置。

对于 Apache 和 IBM HTTP Web 服务器:

```
SetEnv NLSPATH /usr/lib/nls/msg/%L/%N
```

为 FastCGI 配置 Net.Data

FastCGI 接口是一个业界标准接口，它允许应用程序以与 CGI 应用程序类似的方式执行，而进程在各请求之间保持为活动状态。它向其他具有 CGI 应用程序隔离的 Web API 程序提供了类似的性能。Net.Data 可在 Apache Web 服务器和 IBM HTTP 服务器上作为 FastCGI 进程来执行。AIX 和 Sun Solaris 操作系统上支持 FastCGI。

将 Net.Data 配置为一次仅使用一个接口。例如，若将 Web 服务器配置为使用 FastCGI 执行 Net.Data，就不要还将 Web 服务器配置为使用另一接口执行 Net.Data。若您想稍后运行使用另一接口的 Net.Data，则单独针对新接口重新配置 Web 服务器。

在使用 FastCGI 之前，必须安装:

- Apache Web 服务器 1.2.0
- IBM HTTP 服务器 1.3.12.0 或更新版本
- HTTP Fast-CGI 模块

要为 FastCGI 配置 Net.Data:

1. 为您的操作系统配置 Web 服务器和 FastCGI 配置文件:

   对 Apache Web 服务器:

   更新 httpd.conf 文件。

   - 声明新的应用程序:

     ```
     AppClass inst_dir
     -processes proc_num
     -initial-env LIBPATH=libpath
     -initial-env ORACLE_HOME=oracle_path
     -initial-env ORACLE_SID=oracle_instance
     -initial-env DB2INSTANCE=db2_instance
     -initial-env RXQUEUE_OWNER_PID=REXX_perf_var
     -initial-env LANG=locale
     ```

   - 声明 FastCGI 模块:

     ```
     <Location /fcgi-bin>
     SetHandler fastcgi-script
     </Location>
     ```

   对于 IBM HTTP Web 服务器:

   启用 FastCGI 模块和 Net.Data FastCGI 模块:

   - 在 httpd.conf 文件中，使用下列条目:
     - ```
       #Load FCGI Module
       LoadModule fastcgi_module   libex/mod_fastcgi.so
       ```
     - ```
       # Add FCGI Module
       AddModule mod_fastgi.c
       ```
     -
       ```
       FastCgiServer  /usr/HTTPServer/db2www/fcgi-bin/fcgi-bin/db2www
       -appConnTimeout 0 -idle-timeout 30 -init-start-delay 1
       -listen-queue-depth 100 -processes 3
       -restart-delay 5 -port 7125

       <Location /fcgi-bin>
       SetHandler fastcgi-script

       <Location>
       ```

   参数:

   inst_dir

   Net.Data 的可执行文件所使用的路径和目录名。

   对于 Apache Web 服务器:

   ```
   AppClass /u/mydir/apache/fcgi-bin/db2www
   ```

   对于 IBM HTTP Web 服务器:

```
SetEnv /u/mydir/apache/fcgi-bin/db2www
```

proc_num

可以同时处理的请求个数。缺省值为 1，但应增加该值来改进性能，这取决于应用程序需求。参见第167页的『使用 FastCGI』以获取调节信息。

对 Apache Web 服务器:

```
-processes 7
```

IBM HTTP Web 服务器:

```
NumProcesses 7
```

MAXREQUEST

指定 Web 服务器在重复利用 Net.Data Fast-CGI 进程和启动新进程之前，一个 Net.Data Fast-CGI 进程将要服务的请求数。

对于 Apache Web 服务器:

```
SetEnv MAXREQUEST 5000
```

对于 IBM HTTP Web 服务器:

```
SetEnv MAXREQUEST 5000
```

2. **对于 Apache:** 将 fcgi-bin 目录作为新脚本别名添加至 srm.conf 文件中:
   ```
   ScriptAlias /fcgi-bin/ /u/mydir/apache/fcgi-bin
   ```
3. 将所有静态或动态生成的 Web 页面从 CGI-BIN 迁移至 FCGI-BIN。例如:
   ```
   <a href="http://server/fcgi-bin/db2www/filename.ext/block
   [?name=val&...]">any text</a>
   ```
4. 用 FCGI-BIN 代替 CGI-BIN 来修改用于 Net.Data 的 URL 调用的最终用户文档。例如:
   ```
   http://server/fcgi-bin/db2www/filename.ext/block[?name=val&...]
   ```

为配合 Java 小服务程序使用而配置 Net.Data

参见您的 Web 服务器文档以获取有关登记和使用小服务程序的指示信息。Net.Data 小服务程序包含在 NetDataServlets.jar 文件中。Web 服务器需要您在 CLASSPATH 中添加 *inst_dir*/servlet-lib/NetDataServlets.jar 和 *inst_dir*/servlet-lib。

注: 务必解压 NetDataServlets.jar 文件。某些 Web 服务器要求在使用所有 jar 文件之前先将它们解压。

有关安装 Web 服务器和 Web 服务器配置文件伪指令的详情，参见您的 Web 服务器文档。

为使用 Web 服务器 API 而配置 Net.Data

使用 Web 服务器应用程序编程接口 (API) 而不是 CGI 可以很大地改进 Net.Data 的性能。Net.Data 支持以下服务器 API:

- Apache API (APAPI)
- Microsoft Internet Server API (ISAPI)
- Netscape API (NSAPI)

有关每个 API 的详情，参见第167页的『使用 FastCGI』以及您所使用的 Net.Data 版本的自述文件。

要求: 要以 ISAPI、或 NSAPI 方式运行 Net.Data，必须重新配置 Web 服务器来将 Net.Data DLL 或共享库用作其服务伪指令。重新配置之后，您必须重新启动 Web 服务器以使 Net.Data 初始化文件的更改生效。缺省情况下，Net.Data 以 CGI 方式运行。

将 Net.Data 配置为一次仅使用一个接口。例如，若将 Web 服务器配置为使用 FastCGI 执行 Net.Data，就不要还将 Web 服务器配置为使用 ISAPI 或另一接口来执行 Net.Data。若您想稍后运行使用另一接口（如 NSAPI）的 Net.Data，则单独针对新接口重新配置 Web 服务器。

以下章节描述了如何配置 Net.Data 和 Web 服务器以运行 Web 服务器 API 方式。其中提供了一般步骤和示例，但它们可能和您的操作系统有所不同。参见针对您所使用的操作系统的 Net.Data 的自述文件，以获取特定的指示信息。

要配置 Apache API:

1. 停止 Web 服务器。
2. 确保 `libmod_db2www.so` 在 `/opt/netdata/lib` 目录中。
3. 仅对于 Linux：将 `libmod_db2www.so` (Linux) 或 mod_db2www.dll (WinNT) 复制至 `/apache/modules` 目录。
4. 将服务语句添加至 Web 服务器的配置文件 (`httpd.conf`) 以调用 API。

   例如:

   ```
   LoadModule db2www_module /pot/netdata/lib/libmod_db2www.so AddHandler
        db2 www_handler .db2www
   ```

   参见针对您所使用的操作系统的 Net.Data 的自述文件，以获取特定的文件和目录名称。
5. 重新启动 Web 服务器。

APAPI 仅在 Windows NT、Linux 和 Linux s/390 上受支持。

要配置 ISAPI:

1. 停止 Web 服务器。
2. 将 Net.Data 所附带的、用于 ISAPI 的 DLL 复制到服务器的子目录中。例如:

   ```
   /inetsrv/scripts/dtwisapi.filetype
   ```

   其中的 *filetype*对于 Windows NT 和 OS/2 来说是 `.dll`，对于 UNIX 操作系统来说则是 `.o`。

   参见针对您所使用的操作系统的 Net.Data 的自述文件，以获取特定的文件和目录名称。
3. 因为 ISAPI 绕过了 CGI 处理，因此在表和链接中不需要 URL 的 `cgi-bin/db2www/` 部分。相反，需要使用 `dtwisapi.filetype`。例如，如果以下 URL 将 Net.Data 作为 CGI 程序调用:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/test1.dtw/report
   ```

   那么您应使用以下 URL 将 Net.Data 作为 ISAPI 插件来调用:

   ```
   http://server1.stl.ibm.com/scripts/dtwisapi.dll/test1.dtw/report
   ```
4. 若您在 MACRO_PATH 中指定的其中一个目录或 Web 服务器的当前目录下的子目录 `/order/` 中存储了宏 `test1.dtw`，则使用以下 URL 来以 CGI 方式调用 Net.Data:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.dtw/report
   ```

   那么，在 ISAPI 方式中调用 Net.Data 的等价的 URL 是:

   ```
   http://server1.stl.ibm.com/scripts/dtwisapi.dll/orders/test1.dtw/report
   ```
5. 重新启动 Web 服务器。

要配置 NSAPI:

1. 停止 Web 服务器。
2. 将 Net.Data 所附带的、用于 NSAPI 的 DLL 复制到服务器目录中。例如:

   ```
   /netscape/server/bin/httpd/dtwnsapi.filetype
   ```

   其中的 *filetype*对于 Windows NT 和 OS/2 来说是 `.dll`，对于 UNIX 操作系统来说则是 `.o`。

   参见针对您所使用的操作系统的 Net.Data 的自述文件，以获取特定的文件和目录名称。
3. 使用下面列出的更改来修改您的服务器配置文件。参见针对您所使用的操作系统的 Net.Data 的自述文件或程序目录，以了解操作系统之间的区别。

obj.conf　　添加到文件开头:

```
Init fn="load-modules" shlib="<path>dtwnsapi.dll" funcs=dtw_nsapi
```

obj.conf　　添加到服务命令:

```
Service fn="dtw_nsapi" method=(GET|HEAD|POST)
  type="magnus-internal/d2w"
```

mime.types　　添加此类型，其中 *d2w* 是宏的缺省扩展名。您可以指定任意的三个字符的组合。

```
type=magnus-internal/d2w exts=d2w
```

4. 将 Net.Data 宏从 `netdata/macro` 目录移动到服务器的根文档目录:

   ```
   /netscape/server/docs/
   ```

5. 将服务器的根文档目录添加到初始化文件的 MACRO_PATH 语句中。这个更改告诉 Net.Data 在哪里查找宏。
6. 因为 NSAPI 绕过了 CGI 处理，因此在表和链接中不需要 URL 的 `cgi-bin/db2www/` 部分。服务器知道具有 `d2w` 文件类型的文件是 Net.Data 宏，因为您在更改 Netscape 配置文件时对它进行了定义。例如，以下 URL 将 Net.Data 作为 CGI 程序调用:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/test1.dtw/report
   ```

   而以下 URL 将 Net.Data 作为 NSAPI 程序调用:

   ```
   http://server1.stl.ibm.com/test1.dtw/report
   ```

7. 重新启动 Web 服务器。

如果将 Net.Data 宏放在几个目录中，那么最后的三个步骤改为:

1. 将目录和其中包含的 Net.Data 宏移动到服务器的根文档目录。
2. 更新初始化文件中的 MACRO_PATH 变量来包括宏所在的所有目录和子目录。
3. 修改指向这些 Net.Data 宏的链接和表，保留它们的目录名。例如，当以 CGI 方式运行时，以下 URL 调用存储在 `/orders/` 目录中的 Net.Data 宏。

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.dtw/report
   ```

   经过修改后，在 NSAPI 方式中调用 Net.Data 的 URL 要短一些，但仍保留了目录名:

   ```
   http://server1.stl.ibm.com/orders/test1.dtw/report
   ```

使用 Net.Data 管理工具来配置 Net.Data

Net.Data 管理工具帮助您在 Windows NT、AIX 和 OS/2 操作系统上配置和管理 Net.Data 初始化文件 (DB2WWW.INI) 以及用于“现场连接”的初始化文件 (dtwcm.cnf)。使用这一工具之后，您可以完成以下任务:

- 『启动管理工具』
- 第45页的『配置路径语句』
- 第46页的『配置端口』
- 第47页的『配置 Cliette』
- 第52页的『配置语言环境』
- 第55页的『定义配置变量』

参见『开始之前』以学习如何设置管理工具以及如何确保您具有正确的软件先决条件。

开始之前

1. 计划对 Net.Data 语言环境、数据库、cliette、端口和配置变量的配置。
2. 从 CD-ROM 安装 Net.Data。
3. 安装 Java 运行时程序库(各操作系统的 JDK 1.1 及后继版本)。请查看针对您所使用的操作系统的 Net.Data 的自述文件，以获取详情。

   请确保安装 JDK 之后在您的 CLASSPATH 中有`classes.zip`。
4. 如果已经安装了与 DB2 Universal Database 封装在一起的 IBM JDBC 驱动程序，则将该驱动程序目录添加到 Java CLASSPATH 语句中，以便启用 DB2 登录测试。
5. 更改到存储 Net.Data 管理工具程序的目录:

   对于 OS/2 和 Windows NT:

   inst_dir`\connect\`*admin_directory*，其中 *inst_dir* 是安装期间为 Net.Data 指定的目录，而 *admin_directory* 是管理工具文件所在的目录。

   对于 AIX:

   `/usr/lpp/internet/db2www/db2.v2/`*admin_directory*，其中 *admin_directory* 是管理工具文件所在的目录

启动管理工具

您所使用的操作系统决定了您应如何启动管理工具。

对于 OS/2 和 Windows NT:

从 IBM Net.Data 文件夹，选择 **Net.Data 管理工具**图标。

对于 AIX:

更改到 Net.Data 的安装目录 (inst_dir)。从命令行输入 `ndadmin` 来启动该工具。

管理工具被启动，并显示 Net.Data 管理笔记本。

配置路径语句

使用**路径**页面来添加、修改或删除用于定位处理 Net.Data 宏所需文件的路径语句。这些语句在第21页的『路径配置语句』中有所描述。 图7显示了**路径 (Path)** 页面。

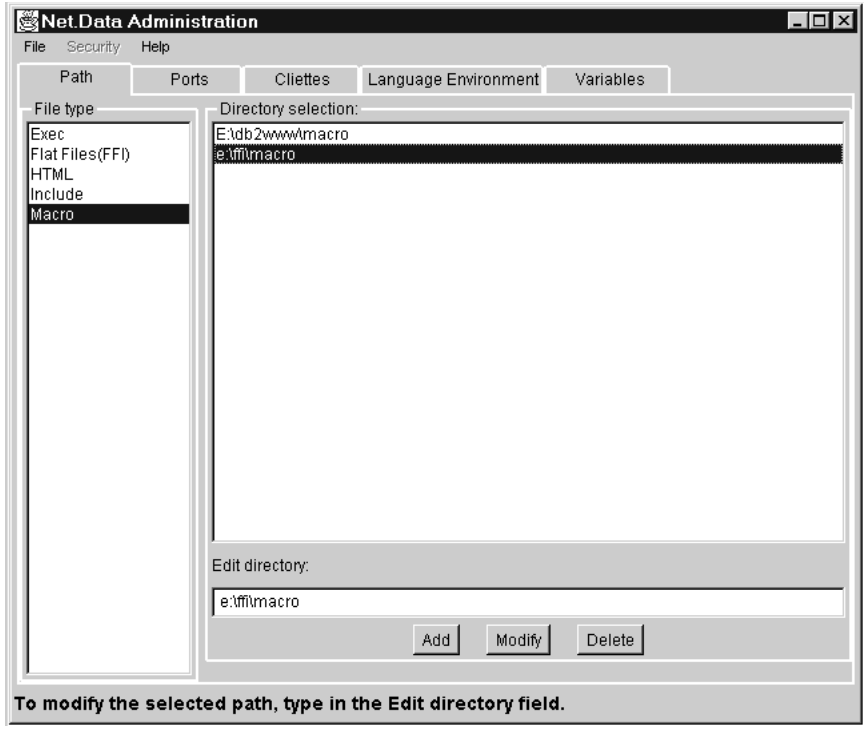


图 7. Net.Data 管理工具的路径页面. 使用这个页面来添加、修改或删除路径语句。

配置提示: HTML 文件类型只有一个路径。

要添加一个路径语句:

1. 启动管理工具。

2. 在**路径 (Path)** 页面中，从**文件类型 (File type)** 中选择一个文件类型，例如，选择 Exec。
3. 在**编辑目录 (Edit directory)** 字段中，输入新的路径并单击**添加 (Add)** 按钮。

   如果指定的路径不存在，则将显示一个警告窗口。如果没有选择目录，则新的目录将作为最后一项添加至列表中。
4. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要修改一个路径语句:

1. 启动管理工具。
2. 在**路径 (Path)** 页面中，从**文件类型 (File type)** 列表中选择您想要更改的文件类型。
3. 在**目录选择 (Directory selection)** 列表中选择您想要修改的路径。选定的路径将在**编辑目录 (Edit directory)** 字段中打开。
4. 修改**编辑目录 (Edit directory)** 字段中的路径并单击**修改 (Modify)** 按钮。如果输入的路径不存在，则将显示一个警告窗口。
5. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要删除一个路径语句:

1. 启动管理工具。
2. 在**路径 (Path)** 页面中，从**文件类型 (File type)** 列表中选择您想要删除的文件类型。
3. 在**目录选择 (Directory selection)** 字段中，选择您想要删除的路径。选定的路径将在**编辑目录 (Edit directory)** 字段中打开。
4. 单击**删除 (Delete)** 按钮。
5. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

配置端口

使用**端口 (Port)** 页面来指定 Net.Data 所使用的 TCP/IP 端口号。第47页的图8显示了**端口 (Port)** 页面。

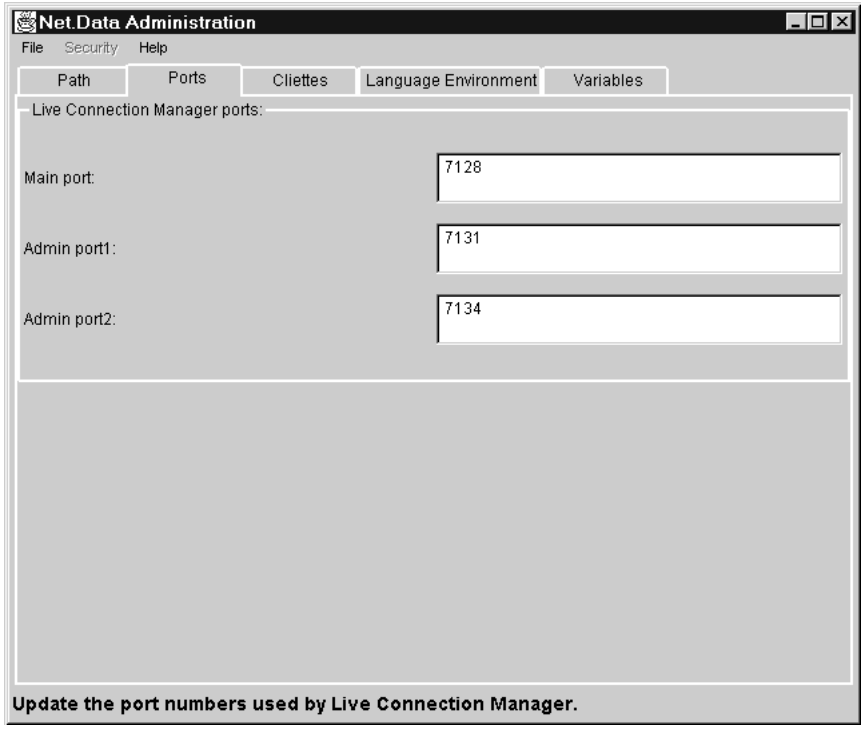


图 8. Net.Data 管理工具的端口页面. 使用这个页面来指定端口。

要指定 TCP/IP 端口号:

1. 启动管理工具。
2. 在**端口 (Port)** 页面中，在每个端口字段中输入唯一的端口号。在您将光标移动到下一个字段时，管理工具将验证您在每个字段中输入的端口号。
3. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

配置 Cliette

使用 **Cliette** 页面来添加、修改或删除“现场连接”数据库 cliette，您还可以管理数据库和用于 cliette 的管理员用户标识和口令。 第168页的『管理连接』中提供了有关 cliette 的详情。第48页的图9显示了 **Cliette** 页面。

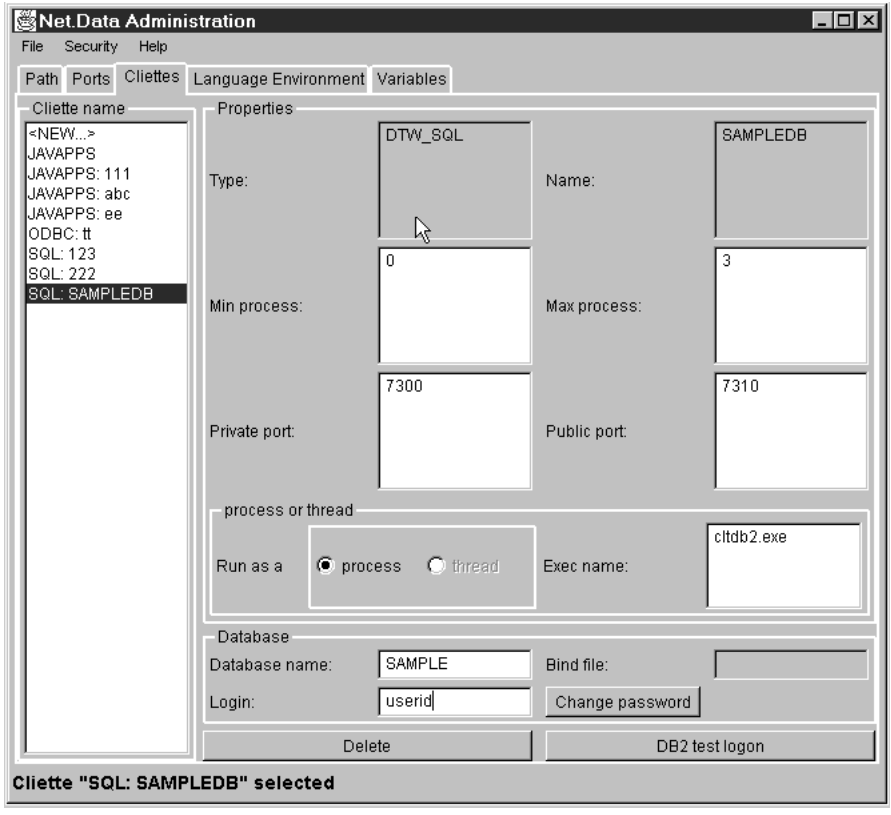


图 9. Net.Data 管理工具的 Cliette 页面. 使用这个页面来添加、修改和删除 cliette。

要添加一个 cliette:

1. 启动管理工具。
2. 在 **Cliette** 页面中，从 **Cliette 名称 (Cliette name)** 列表中选择**<新建 (new) ...>**。将打开**添加 cliette (Add a cliette)** 窗口。

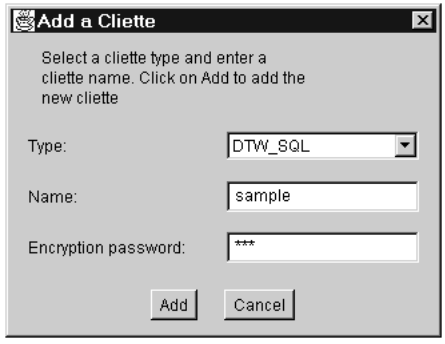


图 10. Net.Data 管理工具的 "添加 Cliette" (Add a Cliette) 窗口. 使用此页面来添加 cliette。

如果启用了加密，那么在您第一次创建或修改 cliette 时将提示您输入加密口令。这个口令将被保存，以后就不需要再输入了。

3. 从**类型 (Type)** 列表中选择一个 cliette 类型。
4. 在**名称 (Name)** 字段中为新的 cliette 输入一个名称。这个名称可以是数据库的名称，也可以是另一个唯一的 cliette 名称。例如: MYCLIETTE。
5. 如果启用了**加密口令 (Encryption password)** 字段，则输入加密口令。您不需要再次输入这个口令，因为管理工具将为您保存该口令。
6. 单击**添加 (Add)** 按钮。

   新的 cliette 将被创建并添加到 cliette 列表底部。另外，新的名称是突出显示的，cliette 的缺省特性将显示在**特性 (Properties)** 组框中。您可以更改这些值来与您的配置相匹配。
7. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要修改一个 cliette:

1. 启动管理工具。
2. 在 **Cliette** 页面中，从 **Cliette 名称 (Cliette name)** 列表中选择您想要更改的 cliette 名称。cliette 的特性显示在**特性 (Properties)** 组框中。
3. 根据需要在**特性 (Properties)** 组框中修改特性。
   a. **类型 (Type)** 字段显示了要定义的 cliette 的类型，且对应于一个语言环境类型名称。Net.Data 在您添加新的 cliette 时填充此字段，选项是在 "添加一个 Cliette" 窗口中的 **Cliette 类型 (Cliette type)** 中定义的。
   b. **名称 (Name)** 字段显示了 cliette 名称，这通常是数据库的名称。Net.Data 在您添加新的 cliette 时填充此字段。

c. 在**最少进程 (Min process)** 字段中输入启动时可以启动的 cliette 进程的个数。对于每个进程，您都需要一个唯一的端口地址。参见第32页的『配置“现场连接”』，以获取有关“最少进程”值的详情。

d. 在**最多进程 (Max process)** 字段输入可以同时运行的 cliette 进程的个数（启动时启动的进程除外）。对于每个进程，您都需要一个唯一的端口地址。参见第32页的『配置“现场连接”』，以获取有关“最多进程”值的详情。

e. 在**专用端口 (Private port)** 字段中输入唯一的端口号，以指定与“连接管理器”同时启动的 cliette 进程一起使用时的启动端口号。对于**最少进程 (Min Process)** 值所指定的每个进程，都将使用一个附加端口号。例如如果您对**专用端口 (Private port)** 指定端口号 7012，对**最少进程 (Min Process)** 指定值 5，则将使用端口号 7012-7016，并且绝对不能与系统中其他的端口分配发生冲突。

f. 在**公用端口 (Public port)** 字段中输入唯一的端口号，以指定与附加进程同时启动的 cliette 进程一起使用时的启动端口号，最多为**最多进程 (Max process)** 字段中指定的个数。对于每个进程都使用了一个附加端口号。例如，如果您对**公用端口 (Public port)** 指定端口号 7020，对**最多进程 (Max process)** 指定值 5，则将使用端口号 7020-7024，并且绝对不能与系统中其他的端口分配发生冲突。

g. **可执行程序名 (Exec name)** 字段显示了 cliette 可执行文件的名称。

4. 如果 cliette 要与数据库一起使用，则根据需要修改**数据库 (Database)** 组框的值:

   a. 指定 cliette 所关联的数据库的名称（在**数据库名称 (Database name)** 字段中），例如，MYDBASE。

   b. **绑定文件 (Bind file)** 字段包含了用于您所使用的 cliette 类型的绑定文件的名称和路径。

   c. **登录 (Login)** 字段指定了用于连接到数据库的登录用户标识。

   d. **更改口令 (Change password)** 按钮将打开“更改数据库口令”(Change Database Password) 窗口。输入加密口令和新的口令（两次）。您可以通过使用**安全性 (Security)** 下拉菜单中指定的加密功能来加密数据库口令。

5. 选择**文件 (File)**，然后选择**保存 (Save)** 来保存您所作的更改。
6. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要测试 DB2 数据库登录和连接:

1. 在管理工具的 **Cliette** 页面上单击 **DB2 测试登录 (DB2 test logon)** 按钮。测试完成之后将打开一个确认窗口，显示连接测试的状态。
2. 关闭此窗口继续配置，或关闭管理工具。

要删除一个 cliette:

1. 启动管理工具。
2. 在 **Cliette** 页面中，从 **Cliette 名称 (Cliette name)** 列表中选择您想要删除的 cliette 名称。
3. 单击**删除 (Delete)** 按钮。
4. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要启用对 cliette 用户标识和口令的加密:

加密为使用 cliette 的数据库连接提供了安全性。启用加密的时候，“现场连接”配置文件中的所有数据库口令均被加密，并且在访问和解密的时候需要一个加密口令。

要求: 必须使用“Net.Data 版本 2 现场连接”配置文件才能使用加密。

1. **要点**: 对“现场连接”配置文件 <path>dtwcm.cnf进行备份。如果您丢失了加密口令，或者想要解密数据库口令并恢复口令，就需要这个文件。
2. 从管理工具的 **Cliette** 页面，选择**安全性 (Security) -> 启用加密 (Turn encryption on)** 下拉菜单选项。将打开“启用加密”确认窗口。
3. 单击**是 (Yes)** 继续。将打开“加密口令”(Encryption Password) 窗口。
4. 对于使用具有加密口令的 cliette 的权限输入口令（两次）。
5. 单击**确定 (OK)** 定义新的口令并对 cliette 加密所有的数据库口令。

要关闭对 cliette 用户标识和口令的加密:

1. 从管理工具的 **Cliette** 页面，选择**安全性 (Security) -> 关闭加密 (Turn encryption off)** 下拉菜单选项。将打开“关闭加密”(Turn encryption off) 确认窗口。
2. 单击**是 (Yes)** 继续。出于安全性的原因，所有的口令都设置为 *USE_DEFAULT。您可以从“现场连接”文件的备份 <path>dtwcm.cnf 中恢复口令。

要更改加密口令:

1. 从管理工具的 **Cliette** 页面，选择**安全性 (Security) -> 更改加密口令 (Change Encryption Password)** 下拉菜单选项。将打开“更改加密口令”(Change Encryption Password) 确认窗口。
2. 单击**是 (Yes)** 继续。将打开“更改加密口令”(Change Encryption Password) 窗口。
3. 输入原来的口令一次，新的口令两次。
4. 单击**确定 (OK)** 更改加密口令。

要更改数据库口令:

1. 在管理工具的 **Cliette** 页面中，单击**更改口令 (Change Password)** 按钮。将打开“更改数据库口令”(Change Database Password) 窗口。
2. 输入加密口令一次，新的数据库口令两次。
3. 单击**确定 (OK)** 更改口令并关闭窗口。如果您启用了加密，则更改后的数据库口令将被加密。

配置语言环境

使用**语言环境 (Language Environment)** 页面来添加、修改或删除 Net.Data 语言环境。语言环境在第26页的『环境配置语句』中讨论。图11显示了**语言环境 (Language Environment)** 页面。

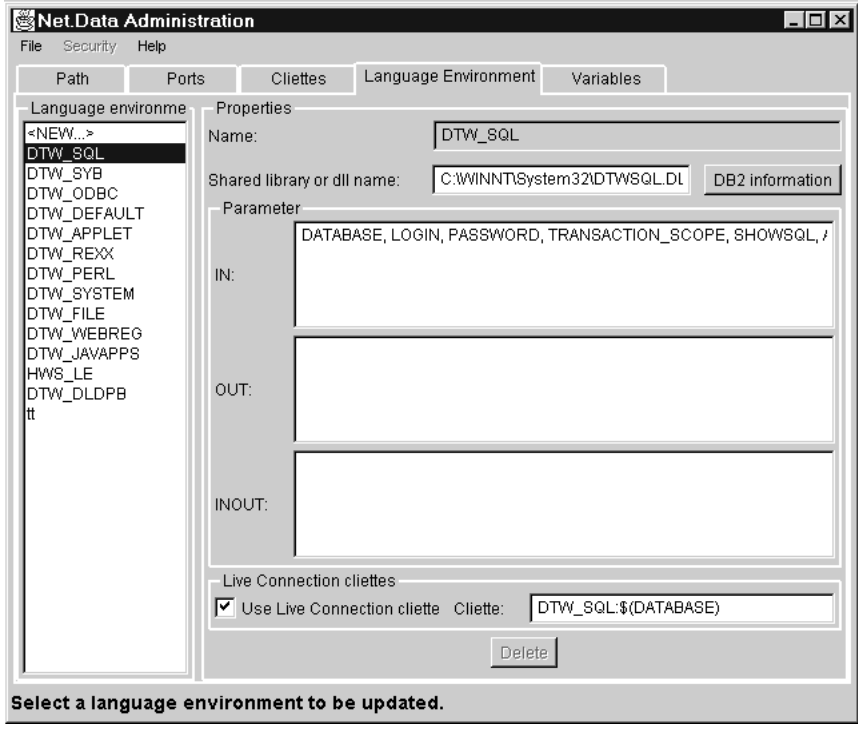


图 11. Net.Data 管理工具的“语言环境”(Language Environment) 页面. 使用这个页面来指定语言环境。

要添加一个语言环境:

1. 启动管理工具。

2. 在**语言环境 (Language Environment)** 页面中，从**语言环境 (Language Environment)** 列表中选择**<新建 (new)...>**。将打开**添加新的语言环境 (Add a new language environment)** 窗口。
3. 在该字段中输入语言环境的名称并单击**添加 (Add)** 按钮。将打开 “添加语言环境” (Add a Language Environment) 窗口。

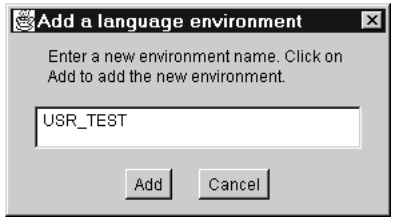


图 12. Net.Data 管理工具的 “添加一个语言环境” 窗口. 使用此页面来指定一个新的语言环境。

新的语言环境将被创建，并且这个名称将被添加到语言环境列表的底部。另外，新的名称是突出显示的，语言环境的缺省特性将显示在**特性 (Properties)** 组框中。您可以更改这些值来与您的配置相匹配。

4. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要修改一个语言环境:

1. 启动管理工具。
2. 在**语言环境 (Language Environment)** 页面中，从**语言环境 (Language Environment)** 列表中选择您想要更改的语言环境的名称。cliette 的特性显示在**特性 (Properties)** 组框中。
3. 根据需要在**特性 (Properties)** 组框中修改特性，如图12中所示:
   a. 在**名称 (Name)** 字段中指定语言环境的名称; 这个名称对应于定义 cliette 时所使用的语言环境类型。要更改此值，可在**语言环境 (Language environment)** 列表中双击另一个名称。参见第26页的『环境配置语句』以获取有关语言环境类型的详情。
   b. 在**共享程序库或 dll 名称 (Shared library or dll name)** 字段中指定共享程序库或 DLL 程序的名称以及语言环境的路径。
   c. 选择 **DB2 信息 (DB2 information)** 按钮来显示 “DB2 信息” (DB2 Information) 窗口，如第54页的图13中所示。

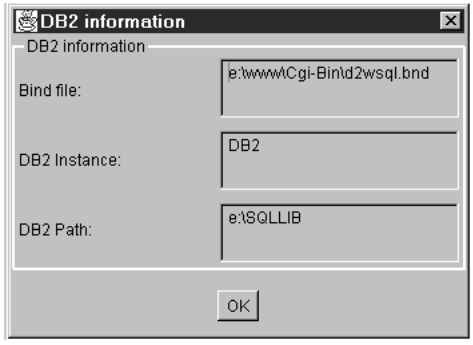


图 13. Net.Data 管理工具的 "DB2 信息" (DB2 Information) 窗口. 使用此页面来指定专用于 DB2 数据库的信息。

指定 DB2 环境变量的值:

1) 在**绑定文件 (Bind file)** 字段中输入绑定文件的路径和文件名。
2) 在 **DB2 实例 (DB2 Instance)** 字段中为使用 SQL 语言环境时所关联的数据库指定 DB2INSTANCE 值。
3) 在 **DB2 路径 (DB2 Path)** 字段中指定 DB2 产品可执行文件的路径目录名，通常是 \SQLLIB。
4) 单击**确定 (OK)** 保存更改并关闭窗口。

d. 在**参数 (Parameters)** 组框中指定输入和输出参数，它们在每次调用语言环境时传送给语言环境或从语言环境传回。

提示: 除非您正在定义自己的语言环境，否则不要更新这些字段。

e. 在**现场连接 cliette (Live Connection cliettes)** 组框中指定是否使用 cliette 以及哪些 cliette 应和语言环境关联。

1) 通过选择**使用现场连接 cliette (Use Live Connection cliette)** 复选框来指定用于语言环境的 cliette 是否活动。如果希望在调用语言环境时使用 **Cliette** 字段中指定的 cliette，则选中这个复选框。
2) 在 **Cliette** 字段中指定和正在定义的语言环境一起运行的 cliette 的名称。该名称的语法取决于您是在配置数据库还是 Java 应用程序语言环境。缺省为 `DTW_SQL:$(DATABASE)`。

用于数据库的语法:

type`:`*name*

其中:

type cliette 的语言环境类型。它可以是以下的一个值:

对于 **Windows NT**:

DTW_ODBC、DTW_ORA、DTW_SQL、DTW_JAVAPPS

对于 **OS/2**:

DTW_SQL、DTW_JAVAPPS

对于 **AIX**:

DTW_ODBC、DTW_ORA、DTW_SQL、DTW_JAVAPPS

name 在 **Cliette** 页面中定义的 cliette 名称。缺省为 $(DATABASE)。

Java 应用程序的语法:

```
DTW_JAVAPPS
```

4. 选择**文件 (File)**，然后选择**保存 (Save)** 来保存您所作的更改
5. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要删除一个语言环境:

限制: 您只能删除用户创建的语言环境，但不能删除 Net.Data 原来所附带的语言环境。

1. 启动管理工具。
2. 在**语言环境 (Language Environment)** 页面中，从**语言环境 (Language Environment)** 列表中选择您想要删除的语言环境的名称。
3. 单击**删除 (Delete)** 按钮。
4. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

定义配置变量

使用**变量 (Variables)** 页面来指定 Net.Data 的主目录并选择出错信息记录的级别。第56页的图14显示了**变量 (Variables)** 页面。

图 14. Net.Data 管理工具的“变量”(Variables) 页面. 使用这个页面来指定初始化变量。

要指定 Net.Data 的主目录:

这个变量也就是我们所知的安装目录变量。

1. 启动管理工具。
2. 在**变量 (Variables)** 页面中，在**安装目录 (Installation directory)** 字段中输入要存储日志文件的目录路径。缺省为 `\inst_dir\logs\`。例如: `e:\db2www`。
3. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

要指定 Net.Data 的出错信息记录级别:

1. 启动管理工具。
2. 在**变量 (Variables)** 页面中，从**错误记录 (Error logging)** 组框中选择一个错误记录级别。
   - **关闭 (off)**
   - **仅记录错误 (errors only)**
   - **记录警告和错误 (both warnings and errors)**
3. 关闭管理工具，或单击另一个选项卡来完成另外的配置任务。

对 Net.Data 访问的文件授予访问权限

使用 Net.Data 之前，需要确保执行 Net.Data 的用户标识对于 Net.Data 宏中引用的文件以及 URL 引用的宏具有适当的访问权。这就意味着这些文件必须位于 Web 服务器可以连接的库或目录中，或者这些用户标识对于它们的目录具有显式访问权。

尤为特别的，请确保执行 Net.Data 的用户标识具有以下权限:

- 要读取 Net.Data 初始化文件 `db2www.ini`
- 要执行 Net.Data 可执行文件和 DLL ，并要在可执行文件和 DLL 的路径中搜索目录
- 要读取适当的 Net.Data 宏并搜索 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
- 要读取“现场连接”配置文件 dtwcm.cnf
- 要读取高速缓存管理器配置文件 CACHEMGR.CNF
- 要读取语言环境引用的外部 Perl 和 REXX 可执行文件

对这些文件授予访问权的方法取决于运行 Net.Data 的操作系统。

第3章 保障您资产的安全性

因特网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一起提供的。

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61页的『在网络上加密数据』
- 第62页的『使用权限审批』
- 第62页的『使用权限』
- 第62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内部网与公用因特网隔开的防火墙。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 服务器与安全内部网以及公用因特网隔开的子网。防火墙软件用于在 Web 服务器和公用因特网之间创建一个防火墙，

并在 Web 服务器和受保护的内部网（其中包含 DB2 服务器）之间创建另一个防火墙。这一配置由图15显示。

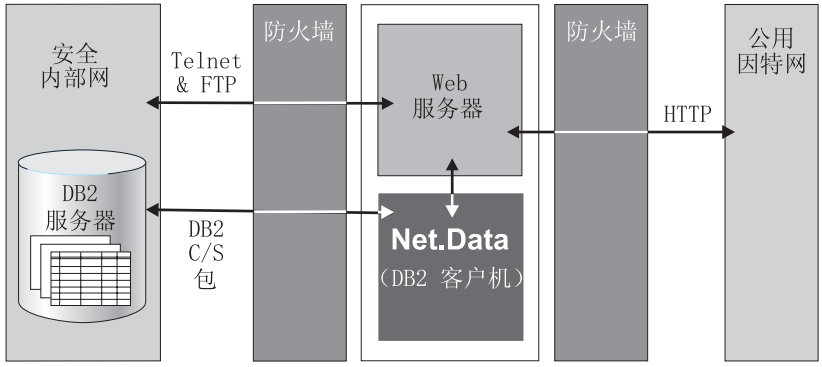


图 15. 高安全性配置

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内部网内部访问 DB2 服务器:

  - 在 Web 服务器设备上安装 Client Application Enabler (CAE)。
  - 配置防火墙，以允许 DB2 通信量通过防火墙。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 服务器和安全内部网之间的 FTP 和 Telnet 访问。一个方法是在 Web 服务器设备上安装一个套接字服务器。

– 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，指定出局 TCP 应答信息包可以从 Web 服务器进入公用因特网上的任何主机。

- **中等安全性配置**

  在此配置中，防火墙软件将受保护的内部网和 DB2 服务器与公用因特网隔开。Net.Data 和 Web 服务器位于防火墙外部的工作站平台上。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。第61页的图16显示了这一配置。

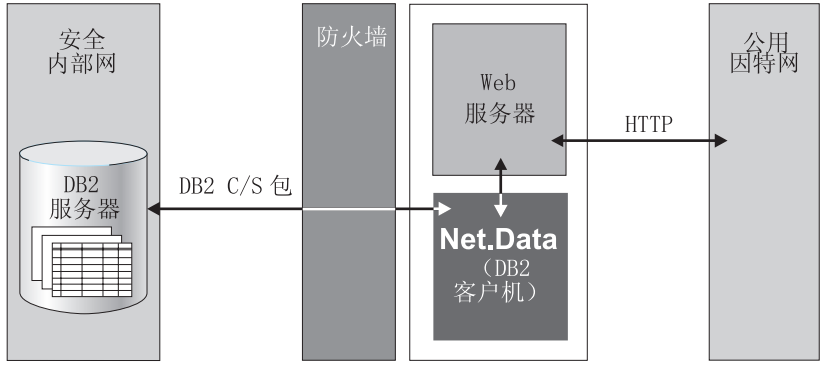


图 16. 中等安全性配置:

您必须在 Web 服务器上安装 CAE 以使 Net.Data 能够与 DB2 服务器通信。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 安装在防火墙和受保护的内部网的外部。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17显示了这一配置。

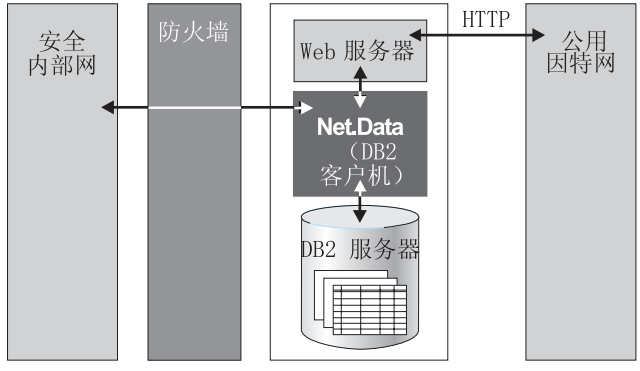


图 17. 低安全性配置:

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登录标识、口令、通过 HTML 表单从客户系统传输到 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。大多数 Web 服务器支持 SSL。

使用权限审批

*权限审批*用于确保进行 Net.Data 请求的用户标识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标识和口令的过程，用以确认请求来自有效的用户标识。Web 服务器将用户标识和它所处理的每个 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标识已授权的资源。

您可以使用两种类型的权限审批：一种是保护您服务器上的某些目录，一种是保护您的数据库。

- 大部分 Web 服务器允许您指定服务器上要保护的目录。您还可以使系统要求那些访问您指定目录的人给出用户标识和口令。参见 Web 服务器的*管理员指南*，以确定系统的能力。
- DB2 对于能够限制某些用户访问表和列的数据库访问有一个权限审批系统。您可以使用 Net.Data 的特殊变量（例如 LOGIN 和 PASSWORD）来链接到 DB2 权限审批例行程序。

提示: 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 的用户标识对宏具有访问权限。有关授予访问权限的详情，参见第57页的『对 Net.Data 访问的文件授予访问权限』。

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 Net.Data 请求相关联的用户标识已得到正确认证，如『使用权限审批』中所述。然后，根据已授权用户标识所具有的权限，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、

可执行文件、包含文件或其他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。参见第5页的『第2章 配置 Net.Data』以获取有关路径语句的附加详细信息。

您还应使用权限检查（如第62页的『使用权限』中所述）并验证 INCLUDE 语句中的文件名是否不能更改（如第64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 Net.Data 版本 2.0.7 或更高版本时，在 Net.Data 初始化文件中使用 DTW_SHOWSQL 配置变量来覆盖在 Net.Data 宏中设置 SHOWSQL 而产生的效果。参见第18页的『DTW_SHOWSQL: 启用或禁用 SHOWSQL 配置变量』中的语法和附加信息。
- Net.Data 版本 2.0.5 或更早版本的用户可以使用 DTW_ASSIGN() 函数，如第64页的『宏开发技术』中所述。

参见 *Net.Data Reference* 的变量一章中有关 SHOWSQL 的内容，以获取 SHOWSQL Net.Data 变量的语法和示例。

考虑对于生产环境启用直接请求是否合适

调用 Net.Data 的直接请求方式允许用户指定 SQL 语句的执行，或直接从 URL 指定 Perl、 REXX 或 C 程序。宏请求方式只允许用户执行那些 SQL 语句和已经定义或在宏中调用的函数。

您应当仔细考虑是否要允许直接请求的使用，因为它可能给您的用户以执行多种函数的能力。在启用这种调用方式的时侯，请确保处理 Net.Data 请求的用户标识具有适当的权限级别。

可以使用 DTW_DIRECT_REQUEST 配置变量来禁用直接请求。参见第16页的『DTW_DIRECT_REQUEST: 启用直接请求变量』中的语法和附加信息。

口令加密

如果在“现场连接”配置文件 (dtwcm.cnf) 和／或 Net.Data 宏中指定 LOGIN 和 PASSWORD，则应通过加密口令来保护它。

要启用加密，如将纯文本口令替换为加密口令:

- 对于 dtwcm.cnf 文件:
  1. 设置 ENCRYPTION=<key>。

2. 使用 'dtwcm -p' 来生成加密口令条目。
3. 剪切此加密字符串并将其粘贴至 dtwcm.cnf 中的 PASSWORD 条目。

- 对于 macros:
  1. 在 db2www.ini 文件中设置 ENCRYPTION=<key>。此密钥必须与 dtwcm.cnf 中的密钥相同。
  2. 使用 'dtwcm -p' 来生成加密口令条目。
  3. 剪切此加密字符串并将其粘贴至 Net.Data 宏中的 PASSWORD 条目。

要禁用加密:

- 对于 dtwcm.cnf 文件，除去带有 ENCRYPTION 的一行。
- 对于宏，除去在 db2www.ini 中带有 ENCRYPTION 的一行。

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。

示例: 不使用:

```
%define START_ROW_NUM = "1"
```

使用:

```
@DTW_ASSIGN(START_ROW_NUM, "1")
```

以此方式指定变量可避免查询字符串赋值（如 "START_ROW_NUM=10"）覆盖宏设置。

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，客户标识必须以字母开头，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。

示例 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 的值是一个购物者标识。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标识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。

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，可使用以下逻辑来确保单引号未被用来修改 SQL 语句:

```
@DTW_ADDQUOTE(shlogid, shlogid)
  @query1()
```

查询变为:

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

示例 2: 使用 DTW_TRANSLATE()

假定应用程序需要确认输入变量 num_orders 中所提供的值是一个整数。一种方法是创建一个事务表 trans_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(num_orders, "x", trans_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。

示例：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 配置语句规范所期望的。

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，以下逻辑可用来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。

示例：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标识。

例如，如果 shlogid 是包含与订单相关联的客户标识的列，SESSION_ID 是一个包含浏览器前用户的授权标识的 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 浏览器查看您的 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。参见第100页的『隐藏变量』，以获取有关隐藏变量的详情。

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 表单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。

有关保护资产的详情，参见以下 Web 站点中有关 “经常询问的问题” (FAQ) 的因特网安全性列表:

```
http://www.w3.org/Security/Faq
```

第4章 调用 Net.Data

本章描述如何使用各种不同的 Web 服务器接口来调用 Net.Data。在使用这些调用方法之前，必须先对指定的接口配置 Net.Data。可以配置 Net.Data 以使用以下 Web 服务器接口:

- 公用网关接口 (CGI)
- FastCGI
- Apache API (APAPI)
- IBM HTTP Server API
- Netscape Server (NSAPI)
- Microsoft Internet Server (ISAPI)
- Java 小服务程序

参见第5页的『第2章 配置 Net.Data』以了解有关对这些接口配置 Net.Data 的详情。缺省情况下，Web 服务器将 Net.Data 作为 CGI 程序调用，每个 Net.Data 请求都在一个新的单独的进程中运行。您在配置 Web 服务器时确定如何调用 Net.Data。

以下章节描述 Net.Data 能够接受的请求类型以及使用各种 API 和小服务程序调用 Net.Data 的方式。

- 『调用请求的类型』
- 第80页的『通过 Web 服务器 API 调用 Net.Data』

调用请求的类型

无论您用何种方式调用 Net.Data，都可以指定两种类型的请求。

宏请求 指定 Net.Data 应执行指定的宏。

直接请求
指定 Net.Data 应执行 SQL 语句、存储过程或函数。

想要编写单个 SQL 查询或调用单个函数（例如 DB2 存储过程、REXX 程序或 Perl 函数）的 Web 开发者可以向数据库发出直接请求了。直接请求中没有任何需要 Net.Data 宏的复杂的 Net.Data 应用逻辑，因此可以绕过 Net.Data 宏处理器。为了改进性能，直接请求参数被传递到适当的语言环境进行处理。

图18说明了宏请求和直接请求之间的区别。宏请求总是在请求的 URL 中指定一个宏，还可以使用表单数据。直接请求则不在 URL 中指定宏文件，但仍然可以使用表单数据。

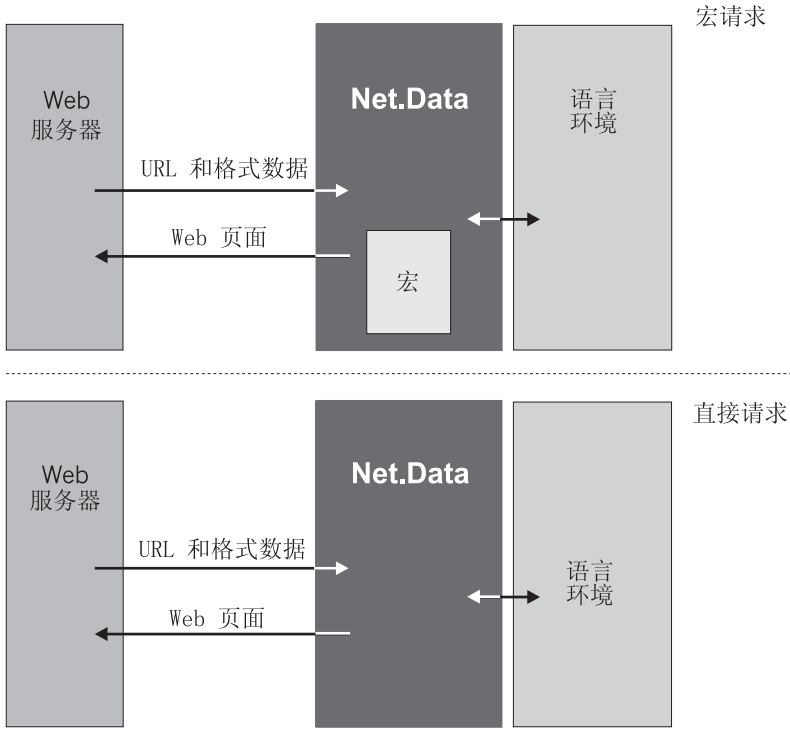


图 18. 宏请求与直接请求

调用 Net.Data 的语法取决于 Net.Data 的配置以及请求的类型。 对于宏和直接请求，都是使用 URL 来调用 Net.Data 的。URL 可以由用户直接输入，也可以作为 HTML 链接或 HTML 表编码到 HTML 页面中。 Web 服务器使用 CGI、FastCGI 或其中一个 Web 服务器 API 来调用 Net.Data。

对于宏请求，在 URL 中指定 Net.Data 宏的名称以及要在 Net.Data 宏内部执行的 HTML 块的名称。对于直接请求，在 URL 中指定 Net.Data 语言环境的名称、SQL 语句或函数的名称、以及任何附加的必需参数值。您可以使用 Net.Data 定义的语法来指定这些值。

如果正从 CGI 迁移至 APAPI 或 FastCGI，可能需要注意某些 RESS 语言环境问题。参见第158页的『REXX 语言环境』以获取详情。

以下章节更为详细地描述了这些调用请求:

- 『使用宏（宏请求）调用 Net.Data』
- 第75页的『不使用宏对 Net.Data 进行调用（直接请求）』

尽管这些示例指出了使用 CGI 调用 Net.Data 时要使用的语法，但这些概念适用于所有那些用于调用 Net.Data 的接口。对于各种接口类型所必需的精确语法，请参考针对该内容的章节。

- 第80页的『通过 Web 服务器 API 调用 Net.Data』

使用宏（宏请求）调用 Net.Data

客户机浏览器通过发送 URL 格式的请求来调用 Net.Data。本节将告诉您如何通过在 URL 请求中指定一个宏来调用 Net.Data。

发送给 Net.Data 的请求具有以下格式。

```
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、小服务程序类、DLL 或共享程序库的路径和文件名。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。参见第25页的『MACRO_PATH』以获取详情。

block 在引用的 Net.Data 宏中指定 HTML 块的名称。

?*name***=***val***&...**
指定一个或多个传送给 Net.Data 的可选参数。

您直接在浏览器中指定此 URL。也可以在 HTML 链接中指定它，或使用表单来构建它，如下所示:

- HTML 链接:

```
<a href="URL">any text</a>
```

- HTML 表单:

```
<form method="method" ACTION="URL">any text</form>
```

参数:

method 指定与表单配合使用的 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。

示例

以下示例演示了调用 Net.Data 的不同方式。

示例 1: 使用 HTML 链接调用 Net.Data:

```
<a href="http://server/cgi-bin/db2www/myMacro.dtw/report">
.
.
.
</a>
```

示例 2: 使用表单来调用 Net.Data

```
<form method="post"
 action="http://server/cgi-bin/db2www/myMacro.dtw/report">
.
.
.
</form>
```

以下章节描述了 HTML 链接和表单，以及有关如何使用链接和表单来调用 Net.Data 的详情:

- 『HTML 链接』
- 第73页的『HTML 表单』

HTML 链接

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

要创建一个 HTML 链接，可使用 HTML <a> 标记。确定希望用作指向 Net.Data 宏的超链接的文本或图形，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。

以下示例显示了这样一个链接: 当用户在 Web 页面上选择文本 ″List all monitors″ 时，这个链接将使得一个 SQL 查询开始执行。

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

单击调用称为 listA.dtw 的宏的链接，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='$(hdware)'
%REPORT{
<h3>Here is the list you requested</h3>
%ROW{
<hr />
$(N1): $(V1), $(N2): $(V2)
<p>$(N3): $(V3)</p>
%}
%}
%}

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

查询将返回一个包含型号、成本和 EQPTABLE 表中对每个监控器描述的描述信息的表。SQL 语句中的值 hdware 是从 URL 输入中获得的。有关 ROW 块中所使用的变量的详细描述，参见 *Net.Data Reference*。

HTML 表单

您可以使用 HTML 表单来动态地定制 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。

以下示例构建在第72页的『HTML 链接』中监控器列表的示例上，它使得浏览器前的用户可以使用一个简单的 HTML 表单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/listA.dtw/report">
<p>What type of hardware do you want to see?</p>
<ul>
<li><input type="radio" name="hdware" value="mon" checked /> Monitors</li>
<li><input type="radio" name="hdware" value="pnt" /> Pointing devices</li>
<li><input type="radio" name="hdware" value="prt" /> Printers</li>
<li><input type="radio" name="hdware" value="scn" /> Scanners</li>
</ul>

<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 执行宏 listA.dtw，它具有一个称为 ″report″ 的 HTML 块，如上所示。

```
%DEFINE DATABASE="MNS97"

 %FUNCTION(DTW_SQL) myQuery(){
```

```
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='$(hdware)'
%REPORT{
<h3>Here is the list you requested</h3>
%ROW{
<hr />
$(N1): $(V1), $(N2): $(V2)
<p>$(N3): $(V3)</p>
%}
%}
%}

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

在上述示例中，SQL 语句中 hdware 的值是从 HTML 表单输入中获得的。有关 ROW 块中所使用的变量的详细描述，参见 *Net.Data Reference*。

FILE 输入类型是经过 Net.Data 的特殊处理的一种输入类型。借助此输入类型，用户可以将文件上载至服务器，以便由服务器上的 Net.Data 或任何其他应用程序进行进一步的处理。

Net.Data 不对已上载的文件执行任何转换，该文件被视为二进制数据。 上载的文件存储在由 DTW_UPLOAD_DIR 指定的目录中，并被赋予唯一的名称，此名称是使用下列规则确定的:

语法:

MacroFileName + '.' + *FormVarName* + '.' + *UniqueIdentifier* + '.' + *FormFileName*

MacroFileName
处理请求的宏的名称（表单中调用的宏）。仅使用文件名，不使用整个路径。

FormVarName
用来在表单中标识该文件的变量的名称。

UniqueIdentifier
用来确保唯一性的字符串。

示例:

首先，在 Net.Data 初始化文件中设置 DTW_UPLOAD_DIR:

```
DTW_UPLOAD_DIR /tmp/uploads
```

然后构造一个调用宏的表单，并且至少使用一个类型为*文件*的输入标记。

```
<form method="post" enctype="multipart/form-data"
      action="/netdatadev/form.dtw/report">
Name:  <input type="text" name="name" /><br />
Zip code:  <input type="text" name="zipno" /><br />
Resume:  <input type="file" name="resume" /><br />
  <input type="submit" />
</form>
```

如果用户要提交该表单，并指定文件 myresume.txt，则在服务器上生成的文件将具有类似如下的名称:

```
/tmp/uploads/form.dtw.resume.20010108112341275-6245-021.myresume.txt
```

不使用宏对 Net.Data 进行调用（直接请求）

本节将告诉您如何使用*直接请求*来调用 Net.Data。在使用直接请求时，不要在 URL 中指定宏的名称。相反，可以使用 Net.Data 定义的语法来指定 Net.Data 语言环境、要执行的 SQL 语句或程序、以及 URL 中所需的任何附加参数值。参见第16页的『DTW_DIRECT_REQUEST: 启用直接请求变量』，以学习如何启用和禁用直接请求。

SQL 语句或程序以及其他指定的参数都被直接传递到指定的语言环境进行处理。直接请求可以改进性能，因为 Net.Data 不需要读取和处理宏。SQL、ODBC、Oracle、Java、 System、Perl 和 REXX 这些 Net.Data 提供的语言环境支持直接请求，您可以使用 URL、HTML 表或链接来调用 Net.Data。

直接请求通过传递 URL 或表单数据的查询字符串中的参数来调用 Net.Data。以下示例说明了可以指定直接请求的上下文。

```
<a href="http://server/cgi-bin/db2www/?direct_request">any text</a>
```

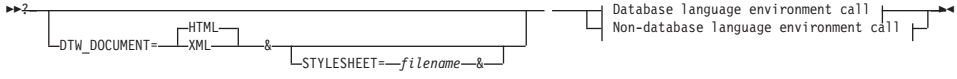
其中 *direct_request* 代表直接请求的语法。例如，以下 HTML 链接中包含直接请求:

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
  any text</a>
```

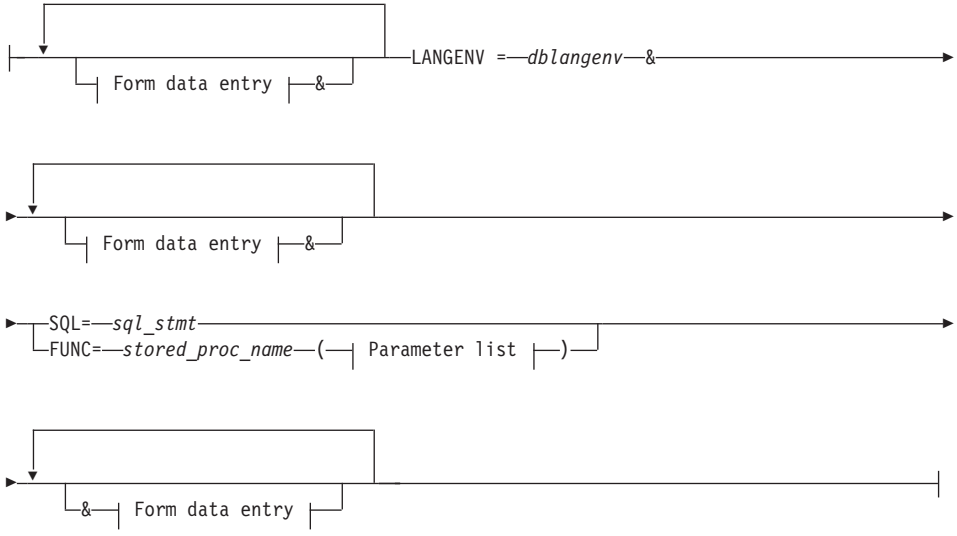
直接请求的语法

使用直接请求调用 Net.Data 的语法中可以包含一个对数据库语言环境或非数据库语言环境的调用。

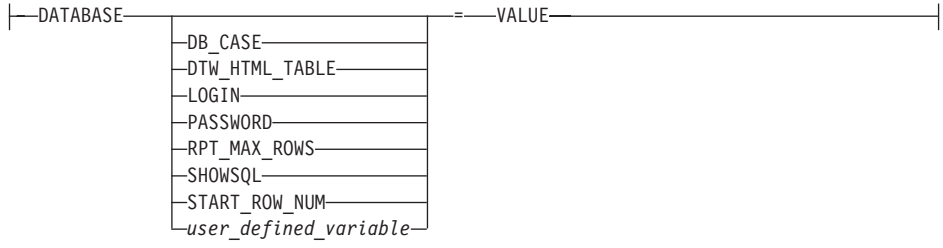
语法



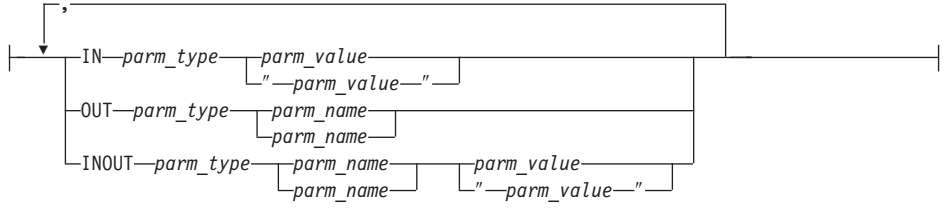
Database language environment call:



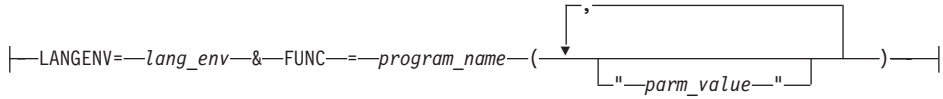
Form data entry:



Parameter list:



Non-database language environment call:



参数

DTW_DOCUMENT

指定 Net.Data 应作为输出返回的文档类型。允许的值是 XML 或 HTML。此参数是可选的，若未指定，则假设为 HTML。

DTW_STYLESHEET

指定显示 XML 时，Net.Data 应使用的样式表。此参数是可选的，且只当 DTW_DOCUMENT=XML 时才有意义。

stylesheet

指定服务器上样式表的文件名。

Database language environment call

向 Net.Data 指定一个调用数据库语言环境的直接请求。

Form data entry

允许您指定 SQL 变量设置或请求简单的 HTML 格式的参数。参见 *Net.Data 参考*以了解这些变量的详情。

DATABASE

指定 Net.Data 应将 SQL 请求传递到哪个数据库。此参数是必需的。

DB_CASE

指定 SQL 语句的大小写情况(大写或小写)。

DTW_HTML_TABLE

指定 Net.Data 应返回一个 HTML 表还是应返回一个预格式化的文本表。

DTW_DOCUMENT

指定 Net.Data 应将结果显示成 XML 还是 HTML。允许的值是 XML 或 HTML。未指定此关键字时，HTML 是缺省值。

LOGIN

指定数据库用户标识。

PASSWORD

指定数据库口令。

RPT_MAX_ROWS

指定一个函数应当在它的报告中返回的最大行数。

SHOWSQL
指定 Net.Data 是应当隐藏还是显示要执行的 SQL 语句。

START_ROW_NUM
指定一个函数应该在哪一行开始它的报告。

user_defined_variable
传递给 Net.Data 的变量，并提供必需的信息或实现 Net.Data 的行为。用户定义的变量是您为应用程序定义的。

VALUE
指定 Net.Data 变量的值。

LANGENV
为 SQL 语句或存储过程调用指定目标语言环境。如果该语言环境是数据库语言环境，则必须指定数据库名称。

dblangenv
数据库语言环境的名称:

- DTW_SQL
- DTW_ODBC
- DTW_ORA

SQL
表示直接请求指定了在线 SQL 语句的执行。

sql_stmt
指定一个字符串，其中包含任何可以使用动态 SQL 来执行的有效的 SQL 语句。

FUNC
表示直接请求指定了一个存储过程的执行。

stored_proc_name
指定任何有效的 DB2 存储过程名。

parm_type
为 DB2 存储过程指定任何有效的参数类型。

parm_name
指定任何有效的参数名。

parm_value
为 DB2 存储过程指定任何有效的参数值。

IN 指定 Net.Data 应当使用该参数将输入数据传递到存储过程。

INOUT

指定 Net.Data 应当使用该参数将输入数据传递到存储过程并返回来自语言环境的输出数据。

OUT

指定语言环境应当使用该参数返回来自存储过程的输出数据。

非数据库语言环境调用

向 Net.Data 指定一个调用非数据库语言环境的直接请求。

LANGENV

为函数的执行指定目标语言环境。

lang_env

指定非数据库语言环境的名称:

- DTW_PERL
- DTW_REXX
- DTW_SYSTEM

FUNC

表示直接请求指定了一个程序的执行。

program_name

指定程序，其中包含要执行的函数。

parm_value

为函数指定任何有效的参数值。

直接请求示例

以下示例显示了可以在使用直接请求方法时调用 Net.Data 的不同方式。

HTML 链接: 以下示例使用直接请求通过链接来调用 Net.Data。

示例 1: 一个调用 Perl 语言环境并调用 Net.Data 初始化文件中 EXEC 路径语句内的 Perl 脚本的链接。

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
  any text</a>
```

示例 2: 一个调用 Perl 语言环境的链接，同前例，但它传递的字符串中具有双引号和空格字符的 URL 编码值。

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_PERL&FUNC=my_perl
     (%22Hello+World%22)">any text</a>
```

例 3一个将执行 SQL 查询(使用 SQL 语言环境)的 URL

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_SQL&DATABASE=CELDIAL
 &SQL=select+*+from+customer">any text</a>
```

示例 4: 一个调用 REXX 语言环境、调用 REXX 程序并将参数传递到程序的 URL

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_REXX&FUNC=myexec.cmd(parm1,parm2)"
     >any text</a>
```

示例 5: 一个调用存储过程并将参数传递到 SQL 语言环境的 URL

```
<a href="http://server/cgi-bin/db2www/?LANGENV=DTW_SQL&FUNC=MY_STORED_PROC
   (IN+CHAR(30)+Salaries)&DATABASE=CELDIAL">any text</a>
```

提示: 您必须在 URL 内对某些字符编码，例如空格和双引号。在此例中，参数值中的双引号字符和空格必须分别编码成 %22 和 + 字符。如果此链接是从宏生成的，可以使用内置函数 DTW_URLESCSEQ 来对 URL 中必须编码的文本进行编码。有关 DTW_URLESCSEQ 函数的详情，参见 *Net.Data 参考*中的描述。

HTML 表: 以下示例使用直接请求通过表来调用 Net.Data。

示例: 一个将执行 SQL 查询(使用 SQL 语言环境)的 HTML 表单，它还连接到 CELDIAL 数据库并查询一个表

```
<form method="post"
 action="http://server/cgi-bin/db2www/">
<input type=hidden name="LANGENV" value="dtw_sql" />
<input type=hidden name="database" value="celdial" />
 <input type=hidden name="SQL"
       value="select * from table1 where col1=$(inputname)" />
Enter Customer name:
<input type=text name="inputname" value="john" />
<input type=submit />
</form>
```

通过 Web 服务器 API 调用 Net.Data

Net.Data 支持以下列表中的 Web API，这取决于您的操作系统:

APAPI 插件
Apache API 插件

IBM HTTP Server API 插件
IBM HTTP Server API 插件

ISAPI 插件
Microsoft Internet Server API 插件

NSAPI 插件
Netscape Server API 插件

参见 *Net.Data 参考*中的操作系统参考附录，以确定对于您的操作系统支持哪个 Web 服务器 API。参见第41页的『为使用 Web 服务器 API 而配置 Net.Data』，以学习如何配置 Net.Data 和 Web 服务器以使它们与 API 一起使用。

要求:

- 如果以 APAPI、ISAPI 或 NSAPI 方式运行 Net.Data，则应重新启动 Web 服务器，这样 Web 服务器就可以重新装入 Net.Data 并将它作为进程运行。
- 如果您在 Web 服务器以 API 方式调用 Net.Data 之后对初始化文件进行更改，那么您必须重新启动 Web 服务器。否则，对于 Net.Data 初始化文件 (db2www.ini) 的任何更改都没有作用。在 API 方式中，Net.Data 只读取初始化文件一次，从而减少对性能的系统开销。
- 在 API 方式中运行时，Oracle 和 ODBC 语言环境需要“现场连接”。

要调用 Web 服务器 API:

对于 APAPI:

语法:

```
http://server/.db2www/macro_name/block[?name=val&...
```

参数:

server
服务器的名称。

macro_name
宏在 MACRO_PATH 指定的目录下的相对路径名。

block
宏中要被处理的 HTML 或 XML 块的名称。

?*name***=***val***&...**
指定一个或多个传递给 Net.Data 的可选参数。

示例:

```
http://myserver/CGI-BIN/.db2www/mymacro.dtw/report
```

对于 ISAPI:

语法:

```
http://server/server_HTML_root_directory/dll_name/macro_name/
  block[?name=val&...]
```

参数:

server_name
服务器的名称。

server_HTML_root_directory
Web 服务器 HTML 根目录的名称。

dll_name
Net.Data 的 ISAPI .dll 文件名，`dtwisapi.dll`。

macro_name
宏在 MACRO_PATH 指定的目录下的相对路径名。

block
宏中要被处理的 HTML 或 XML 块的名称。

?*name*=*val*&...
指定一个或多个传递给 Net.Data 的可选参数。

示例:

```
http://myserver/scripts/dtwisapi.dll/mymacro.dtw/report
```

对于 **NSAPI**:

语法:

```
http://server/macro_name/block[?name=val&...]
```

参数:

server
服务器的名称。

macro_name
宏在 MACRO_PATH 指定的目录下的相对路径名。宏的扩展名，例如 `.dtw`，必须在 Web 服务器配置文件中定义。参见第41页的『为使用 Web 服务器 API 而配置 Net.Data』，以获取详情。

block
宏中要被处理的 HTML 或 XML 块的名称。

?*name*=*val*&...
指定一个或多个传递给 Net.Data 的可选参数。

示例:

```
http://myserver/mymacro.dtw/report
```

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 页的布局
- 定义变量和函数
- 调用 Net.Data 的内置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19所示，Net.Data 宏包括两个组织部分: 说明部分和显示部分。

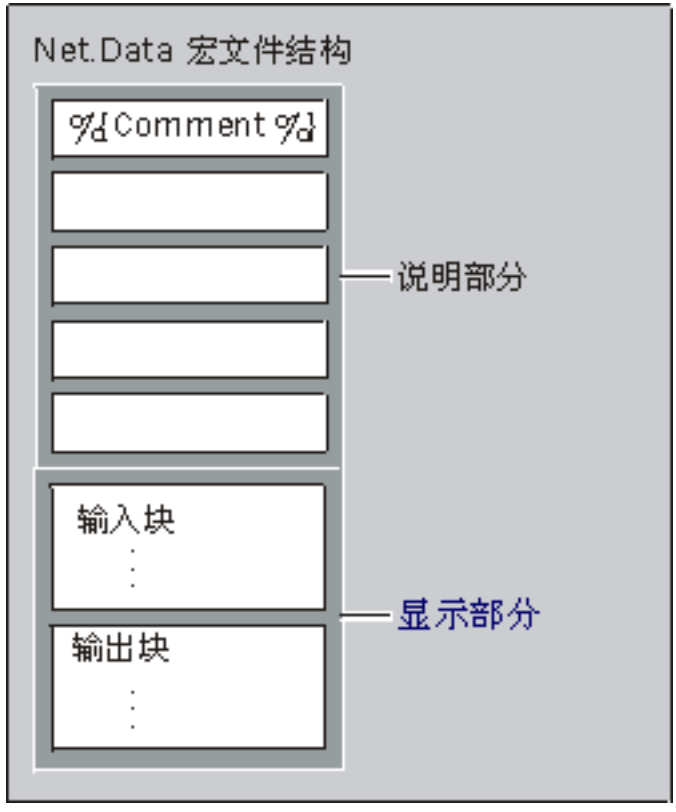


图 19. 宏结构

- *说明部分*包含宏中变量和函数的定义。

- *显示部分*包含指定 Web 页布局的 HTML 或 XML 块。HTML 或 XML 块由 Web 浏览器支持的文本显示语句组成，如 HTML、JavaScript 和架构良好的 XML。

您可以以任何顺序多次使用这些部分。参见 *Net.Data Reference* 以了解有关宏的部分和结构的语法。

权限提示: 确保执行 Net.Data 所使用的用户标识有权读取此文件。参见第57页的『对 Net.Data 访问的文件授予访问权限』，以获取详情。

本章检查组成 Net.Data 宏的不同块以及用于写宏的方法。

Net.Data 宏的剖析

宏由两部分组成:

- 说明部分，包含了要在显示部分中使用的定义。说明部分使用两个主要的可选块:
  - DEFINE 块
  - FUNCTION 块

  说明部分还可以包含其他语言结构和语句，例如 EXEC 语句、IF 块、INCLUDE 语句和 MESSAGE 块。关于语言结构的详情，参见 *Net.Data Reference* 中关于语言结构的章节。

  权限提示: 确保执行 Net.Data 所使用的用户标识有权读取和执行 EXEC 语句所引用的文件，且有权读取 INCLUDE 语句所引用的文件。参见第57页的『对 Net.Data 访问的文件授予访问权限』，以获取详情。
- 显示部分定义 Web 页面的布局、引用变量，并使用 HTML 或 XML 块作为宏的入口和出口点来调用函数。在调用 Net.Data 时，指定一个块名作为处理宏的入口点。HTML 或 XML 块在第87页的『HTML 块』和第89页的『XML 块』中作了描述。

在本小节中，用一个 Net.Data 宏说明了宏语言的元素。此示例宏呈示一个表单，该表单提示要向 REXX 程序传送的信息。宏将此信息传送至名为 ompsamp.cmd 的外部 REXX 程序，该程序回送用户输入的数据。然后在第二个 Web 页面上显示结果。

首先看整个宏，然后详细看每块:

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

%{ ********************** FUNCTION Definition block ************************%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
{
  %EXEC{ompsamp.cmd %}
%}

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

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

<form method="post" action="output">
Type some data to pass to a REXX program:
<
input name="input_data" type="text" size="30" />
<p>
<input type="submit" value="enter" />
</p>
</form>

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

%{ **********************    HTML Block: Output     ************************%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rexx1(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 等文本显示语句，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、Web 服务器 API 或 Java 小服务程序 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 Block        ************************%}
%DEFINE{
   page_title="Net.Data Macro Template"
%}
```

第一行是一个注释。注释是 %{ 和 %}中的任何文本。注释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 Block ***********************************%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- This function accepts
                                                         one parameter and returns the
                                                         variable 'result', which is
```

```
                                                    assigned by the external REXX
                                                    program
     %EXEC{ompsamp.cmd %}  <-- The function executes an external REXX program
                               called "ompsamp.cmd"
%}

%FUNCTION(DTW_REXX) today () RETURNS(result) {
      result = date()  <-- The single source statement for this function is
                           contained inline.
%}
```

第一函数块 rexx1 是一个 REXX 函数说明，它依次运行一个名为 ompsamp.cmd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.cmd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。可通过用普通标记样式标记（如 <b> or ）将请求 @rexx1() 函数调用括起来，以便以您想要的任何方式来格式化结果文本。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 函数都允许内联程序，因为它们是解释语言，可以动态语法分析和执行。内联程序的优点是简明，不需要一个程序文件来管理。第一个 REXX 函数也可以内联处理。

HTML 块

HTML 块定义 Web 页面的布局，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单的 HTML。

```
%{ **********************      HTML Block: Input    ************************%}
%HTML (INPUT) {               <--- Identifies the name of this HTML block.
<html>
<head>
<title>$(page_title)</title>  <--- Note the variable substitution.
</head><body>
<h1>Input Form</h1>
Today is @today()             <--- This line contains a call to a function.

<form method="post" action="output">  <--- When this form is submitted,
```

```
                                        the "OUTPUT" HTML block is called.<p>
Type some data to pass to a REXX program:
<input name="input_data"      <--- "input_data" is defined when the form
TYPE="text" SIZE="30" />           is submitted and can be referenced elsewhere in
                                   this macro. It is initialized to whatever the
                                   user types into the input field.
</p>
<input type="submit" value="enter" />

< hr>
<p>
[
<a href="/">Home page</a>]</p>
 </body><html>
%}                            <--- Closes the HTML block.
```

整个块由 HTML 块标识符 %HTML (INPUT) {...%} 括起来。INPUT 标识了该块的名称。名称可包含下划线、句点和字母数字字符；Net.Data 是不分大小写的。HTML <title> 标记包含了变量替换的示例。变量 page_title 的值替换了表单的标题。

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 块之间或在宏之间浏览的示例。表单提交时， ACTION 参数中对另一块名称的引用将访问此块。HTML 表单中的任何输入数据都作为隐式变量传送给块。此表单上定义的单个输入字段，就是这样的。当表单提交时，在此表单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.dtw/main" 访问在宏 othermacro.dtw 中称为主程序的 HTML 块。表单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.dtw/main?input_data=value">Next macro</a>
```

您可以通过引用表单中指定的变量名来访问或处理宏中的表单数据。

示例中的下一个 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 Block: Output     ************************%}
%HTML (OUTPUT) {
<html>
```

```
<head>
<title>$(page_title)</title>  <--- More substitution.

</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)  <--- This line contains a call to function rexx1
                           passing the argument "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 rexx1 并将变量 input_data 的内容传送给它，该变量接收自 Input 块中定义的表单。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。

XML 块

当您想要将 XML 传递至另一处理应用程序或客户机浏览器时，可使用 XML 块结构来传递 XML 内容。

XML 块的工作方式与 HTML 块相同；它是宏的入口点。在这个块中，可以直接输入 XML 标记，引用变量和进行函数调用。

因此，您可以定制生成的 XML 文档以满足您的需要，XML 块不生成 prolog 标记。输入您的企业特定的 prolog 信息，并包括您选择的样式表。Net.Data 附带提供了三个您可以使用的 XSL 样式表。这些样式表包含用于 Net.Data 生成的所有 XML 元素的变换。但是这些样式表只是一些示例，我们鼓励您扩展这些样式表，或创建您自己的样式表。

```
%DEFINE SHOWSQL = "yes"

%FUNCTION(DTW_SQL) NewManager(){
select * from staff where job = 'Mgr' and years <= 5
%}

%XML(report) {
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="/netdata-xml/ndTable.xsl" ?>

<XMLBlock>
  <h1>List of New Managers</h1>
  @NewManager()
</XMLBlock>
%}
```

图 20. 包含 XML 报告块的宏

在调用 SQL 函数以返回缺省报告时，Net.Data 生成使用一小组 XML 元素的结果集，如以下样本“文档类型描述”(DTD) 中所述。

```
<!-------------------------------------------------------->
<!-- The root element of the document.                  -->
<!-------------------------------------------------------->
<!ELEMENT XMLBlock (RowSet|ShowSQL|Message)*>
<!ATTLIST XMLBlock name   CDATA #IMPLIED>

<!-------------------------------------------------------->
<!-- The default presentation format for tables uses    -->
<!-- the RowSet, Row, and Column elements.              -->
<!-------------------------------------------------------->
<!ELEMENT RowSet   (Row)*>
<!ATTLIST RowSet   name   CDATA #IMPLIED>
<!ELEMENT Row      (Column)*>
<!ATTLIST Row      name   CDATA #IMPLIED
                   number CDATA #IMPLIED>
<!ELEMENT Column   (#PCDATA)>

<!-------------------------------------------------------->
<!-- SQL statements resulting from setting the SHOWSQL  -->
<!-- variable are presented with the ShowSQL element.   -->
<!-------------------------------------------------------->
<!ELEMENT ShowSQL  (#PCDATA)>

<!-------------------------------------------------------->
<!-- Messages are presented with the Message element.   -->
<!-------------------------------------------------------->
<!ELEMENT Message  (#PCDATA)>
```

这些元素定义如下:

XMLBlock

文档的根元素。此标记必须人工输入。

RowSet

包含结果集中的行。RowSet 的名称属性按如下方式确定:

- 对于从调用执行 SQL 查询的函数返回的结果集，使用该函数的名称。
- 对于从调用运行存储过程返回的结果集，使用结果集的名称。如果该结果集未命名，则使用函数名。

Row 包含一行中的列，并作了编号以便于标识。

Column

包含特定行和作为该行命名依据的列的数据值。

ShowSQL

包含用于当前查询的 SQL 语句。

Message

包含 Net.Dta 或 DB2 生成的任何出错信息。

通过使用以上元素，Net.Data 将从第90页的图20中列示的宏生成以下输出。

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="/netdata-xml/ndTable.xsl" ?>
<XMLBlock>
  <h1>List of New Managers</h1>
  <ShowSQL>select * from staff where job = 'Mgr' and years <= 5</ShowSQL>
  <RowSet name="NewManager">
    <Row number="1">
      <Column name="ID">30</Column>
      <Column name="NAME">Marenghi</Column>
      <Column name="DEPT">38</Column>
      <Column name="JOB">Mgr</Column>
      <Column name="YEARS">5</Column>
      <Column name="SALARY">17506.75</Column>
      <Column name="COMM"></Column>
    </Row>
    <Row number="2">
      <Column name="ID">240</Column>
      <Column name="NAME">Daniels</Column>
      <Column name="DEPT">10</Column>
      <Column name="JOB">Mgr</Column>
      <Column name="YEARS">5</Column>
      <Column name="SALARY">19260.25</Column>
      <Column name="COMM"></Column>
    </Row>
  </RowSet>
</XMLBlock>
```

第92页的图21和第93页的图22显示了当使用 Net.Data 附带提供的两个样式表（ndTable.xsl 和 ndRecord.xsl）中的每一个时，上述数据在浏览器中的外观。

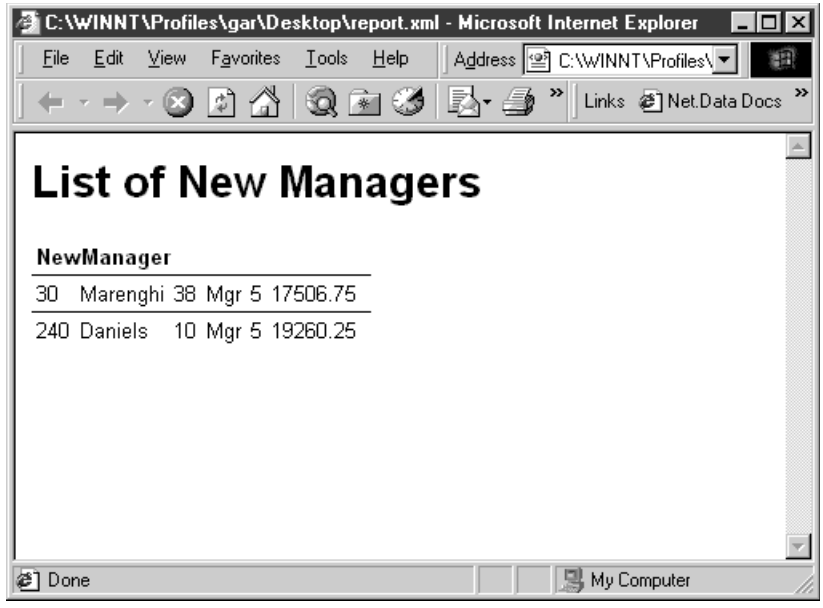


图 21. 使用 ndTable.xsl 样式表显示的 XML

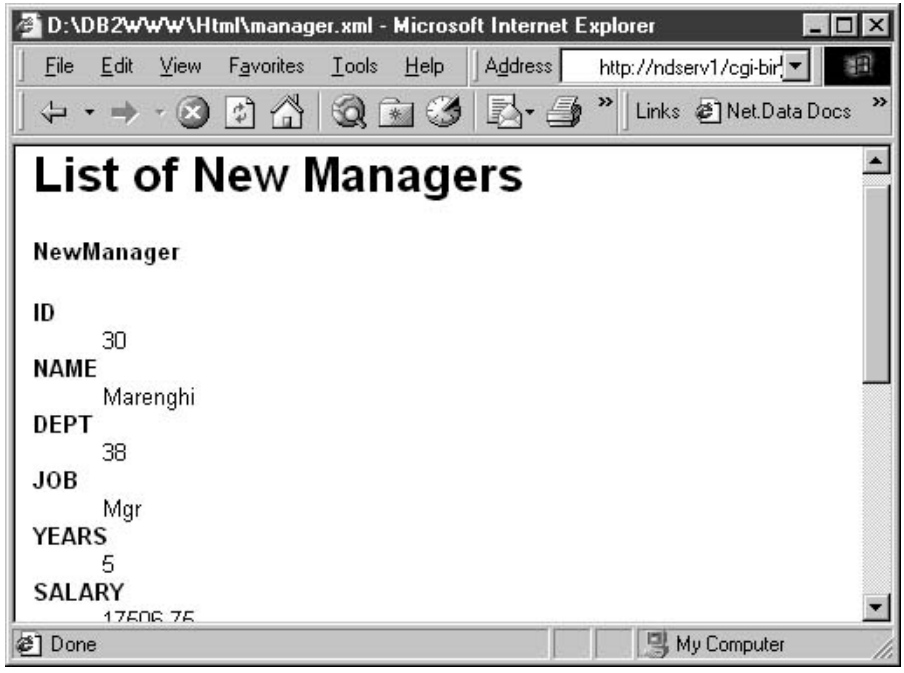


图 22. 使用 ndRecord.xsl 样式表显示的 XML

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。传送的变量名、值和文字串称为记号。Net.Data 对记号的大小没有限制，只要系统内存能处理，就可以传送任何记号。但是个别语言环境对记号大小可能会有所限制。

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 例示并对应于表单数据或查询字符串数据的那些变量。

– 与一个 Net.Data 表相关联并只可以在 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95页的『定义变量』
- 第97页的『引用变量』
- 第98页的『变量类型』

标识符作用域

如果标识符具有全局作用域，则在执行单个请求期间可在宏中的任何位置引用它。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 内置函数
  – 表单数据
  – 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。(此定义不包括 DEFINE 块。) 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说明之后的项引用。
- 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列表中说明。

    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列表中的标识符。
  – 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列表中说明，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、 ...、N*n*), 则它具有报告块作用域。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告块作用域。例示的任何其他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。例示的任何其他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 表单标记
- 查询字符串数据

从表单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 (#)。所有变量名都是区分大小写的, 但 V_*columnName* 除外, 它是一个表变量。

  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 表单标记: 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 表单标记来定义 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，而变量值是根据表单中接收的输入来确定的。参见第73页的『HTML 表单』以获取关于如何在 Net.Data 宏中使用此类变量定义的示例。

- **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.dtw/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，如同来自表单数据一样。

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或输入数据。例如，可以生成 HTML 表单元素列表，这些表单元素是根据 SQL 查询所返回的记录生成的。

要将变量作为文本显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，如果一个变量引用包含了无效字符，如惊叹号 (!)，则该引用被解析为空字符串。

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）。

示例 1: 链接中的变量引用

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。 **示例 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 表单:

```
...
  %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单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。

示例 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

可在宏中使用下列类型的变量。

- 第99页的『条件变量』
- 第99页的『环境变量』
- 第99页的『可执行变量』
- 第100页的『隐藏变量』
- 第101页的『列表变量』
- 第102页的『表变量』
- 第103页的『杂项变量』
- 第103页的『表处理变量』
- 第104页的『报告变量』
- 第105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。

参见 *Net.Data Reference* 以获取有关每种类型的变量的语法和示例。

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
```

参见第101页的『列表变量』以获取使用条件变量与列表变量的示例。

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)
```

输出类似于:

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

参见 *Net.Data Reference* 以了解有关 ENVVAR 语句的详情。

可执行变量

您可以用可执行变量来从变量引用中调用其他函数。

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 语言环境元素的详情，参见 *Net.Data Reference* 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。以下示例演示了这个方法。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"
```

无论 $(date) 出现在 Net.Data 宏中的何处，Net.Data 都会搜索可执行程序 date，并在定位它时显示:

```
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。参见第22页的『EXEC_PATH』，以获取详细信息。
2. 如果 Net.Data 找不到此程序，系统将搜索系统 PATH 环境变量或库列表所定义的目录。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。在先前的示例中，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)"
```

隐藏变量

您可以使用隐藏变量，对用他们的 Web 浏览器查看您的 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，将 $(X) 替换为 $$(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 表单时，$$(name) 和 $$(addr) 分别被替换以 $(name) 和 $(addr)，所以实际的表和列名肯定不出现在 HTML 表单上。应用程序用户无法区分实际变量名是隐藏的。当用户提交该表单时，调用 HTML(REPORT) 块。当 @mySelect() 调用 FUNCTION 块时，$(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.dtw/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 表单中，如果没有选择任何框，则 conditions 是空的，因此查询中的 whereClause 也是空的。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'
```

表变量

表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:

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

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

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

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

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值。当然，语言环境脚本或程序将逐格定义表值。参见第127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变量的详情。

您可以通过引用表变量名来传送一个表。可以在一个函数的 REPORT 块中引用表中的个别元素，也可以使用 Net.Data 表函数来实现。参见『表处理变量』以了解如何在 REPORT 块中访问表内的个别元素，参阅第114页的『表函数』以了解如何使用表函数来访问表的个别元素。在 SQL 函数中表变量通常是填充了值的，然后，表变量在 SQL 函数或另一个函数中在作为参数传送给该函数之后，被用作对报告的输入。您可以将表变量作为 IN、OUT 或 INOUT 参数传送给任何非 SQL 函数。表只能作为 OUT 参数传递给 SQL 函数。

如果引用了一个表变量，则将显示表的内容，并根据 DTW_HTML_TABLE 变量的设置对其进行格式化。在下面的示例中，将显示 myTable 的内容:

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

表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"
```

表处理变量

Net.Data 定义表处理变量供 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)
```

一些表处理变量受其他变量或内置函数的影响。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)
```

报告变量

Net.Data 以缺省报告格式显示宏生成的 Web 页面输出。在 HTML 块中，缺省报告格式使用 <pre> </pre> 标记或使用 HTML 表标记显示表。在 XML 块中，使用 <RowSet>、<Row> 和 <Column> 标记。通过用显示输出的说明来定义 REPORT 块，或通过使用报告变量之一来防止生成缺省报告，可以覆盖缺省报告。

报告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告和 Net.Data 表一起使用。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。

报告变量指定间隔、覆盖缺省报告格式、指定是应当使用 HTML 还是使用固定宽度字符来显示表输出，并指定其他显示特征。例如，可将 DTW_HTML_TABLE 设置为“是”，而 Net.Data 将生成使用 HTML 表标记而不是纯文本格式的表来生成缺省报告。

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

START_ROW_NUM 报告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 表而不是文本格式的表。

```
%DEFINE DTW_HTML_TABLE="YES"

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

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、为 Java 小应用程序提供替换文本、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)
%}
```

下一个示例显示怎样定义要访问哪个数据库。

```
%DEFINE DATABASE="CELDIAL"
```

Net.Data 函数

Net.Data 提供了在应用程序中使用的内置函数，例如字处理函数、字符串处理函数或检索和设置表变量函数的函数。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 内置函数

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变量的函数。

这些章节将描述以下主题:

- 『定义函数』
- 第111页的『调用函数』
- 第111页的『调用 Net.Data 内置函数』

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 块可包含 HTML 块或 XML 块所允许的任何语句。

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, ...)  {
  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。参见第111页的『调用函数』，以获取详细信息。

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列表。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列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。参见第127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。参数被传送至语言环境，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。 *executable-statements* 包含在 HTML 块中允许的那些可执行语句。

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详情，参见第99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。参见第117页的『报告块』。

message-block

定义 MESSAGE 块，它处理 FUNCTION 块因错误状态返回的任何消息。有关如何捕捉错误状态的详情，参见第109页的『信息块』。

在 Net.Data 宏调用函数之前，在其他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，它们可能被作为 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 快的开头来解释。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，或使用任何其他 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，而不是将代码内联。
- 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});
  }
  ...
%}
```

信息块

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、消息文本和一个要进行的操作。*Net.Data Reference* 一书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。然而，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（Net.Data 内置函数是一个例外，其错误由全局消息块处理）。

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

1. 检查局部 MESSAGE 块中的 RETURN_CODE 或 SQL_STATE 值的精确匹配; 根据指定来退出或继续。
2. 如果值不是 0，则检查局部 MESSAGE 块中的 `+default` 或 `-default`; 根据值的符号，根据指定来退出或继续。
3. 如果值不是 0，则检查局部 MESSAGE 块中的 `default`; 根据指定来退出或继续。
4. 检查全局 MESSAGE 块中的 RETURN_CODE 或 SQL_STATE 的精确匹配; 根据指定来退出或继续。

5. 如果值不是 0，则检查全局 MESSAGE 块中的 +default 或 -default；根据值的符号，根据指定来退出或继续。
6. 如果值不是 0，则检查全局 MESSAGE 块中的 default；根据指定来退出或继续。
7. 如果值不是 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:

```
%{ global message block %}
%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.cmd %}
  %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 块中检查精确匹配。
2. 在局部 MESSAGE 块中检查 +default。
3. 在局部 MESSAGE 块中检查 default。
4. 在局部 MESSAGE 块中检查精确匹配。
5. 在全局 MESSAGE 块中检查 +default。

当 Net.Data 找到一个匹配时，它向 Web 浏览器发送消息文本，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，然后才打印消息文本。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函数和内置函数。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。除非是内置函数，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。

示例 1: 用文本字符串参数进行函数调用

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

示例 2: 用变量和函数调用参数进行函数调用

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

示例 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 内置函数

Net.Data 提供了大量的内置函数来简化 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。您可以象调用其他函数一样调用这些函数。

第112页的图23显示了 Net.Data 内置函数和宏是如何相互作用的。

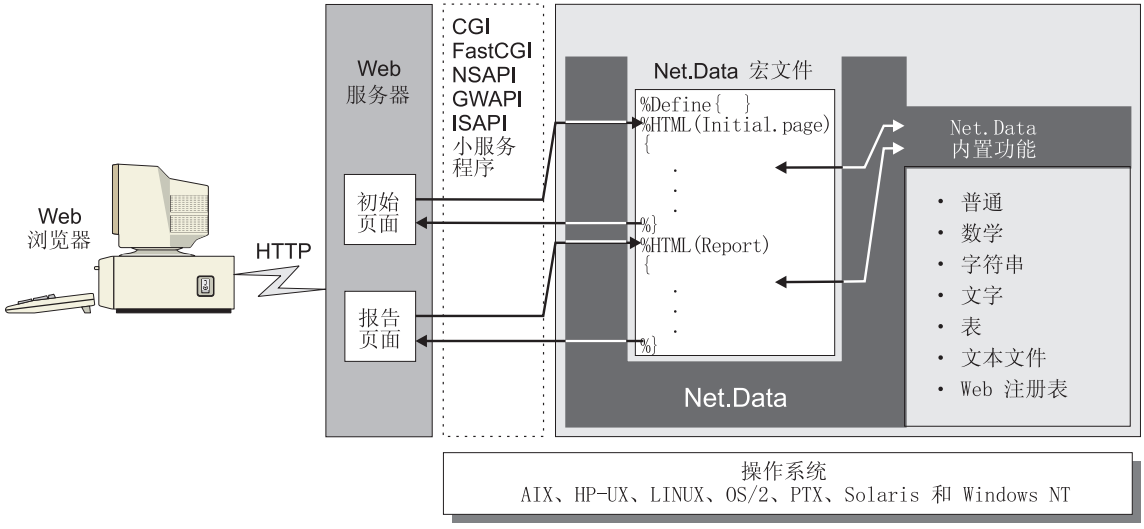


图 23. Net.Data 内置函数

根据前缀的不同，内置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 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。

有些内置函数并不具有每一类型。要确定某个特定内置函数具有的类型，参见 *Net.Data Reference* 中的 Net.Data 内置函数章节。

以下章节提供了 Net.Data 内置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、字处理或表处理功能。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。参见 *Net.Data 参考*以获取每个函数及其语法和示例的描述。

- 第113页的『通用函数』
- 第113页的『数学函数』
- 第114页的『字符串函数』
- 第114页的『字处理函数』
- 第114页的『表函数』

- 第115页的『平面文件函数』
- 第115页的『Java 小应用程序函数』
- 第115页的『Web 注册表函数』

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，并从系统中获取其他有用信息。

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(sort_page) {

<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"`。

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，如下面的示例中所示:

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

DTW_POWER 在变量 result 中返回 ".125"

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、增加其他有用的函数。

例如，可使用 DTW_ASSIGN 来指定值或更改变量的值。还可使用此函数来对变量指定值或更改变量的值。在下面的示例中，变量 RC 被赋值为 0。

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

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

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_DELW0RD 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"。

其他字处理函数包括 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)。

表函数

您可以使用这些函数来生成报告或表(这些报告或表使用 Net.Data 表变量中的数据)。您还可以使用这些函数来创建 Net.Data 表，处理和检索那些表中的值。表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 在表后追加一行。在下面的示例中，Net.Data 在表 myTable 后面追加了十行:

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

另外，DTW_TB_DUMPH 返回一个宏表变量的内容（包括在 <pre></pre> 标记中），表中的每一行显示在不同的行中。而 DTW_TB_CHECKBOX 从宏表变量返回一个多个 HTML 复选框输入标记。

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，这样其他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，从而允许其他请求访问该文件。

Java 小应用程序函数

使用 “Java 小应用程序” 函数可对基于 Net.Data 变量的 web 页很容易地生成 <applet> 和 <parm> 标记。

例如，如果将小应用程序命名为 myApplet，且您想要将某些参数传送至该小应用程序（包括表变量），则可进行以下操作:

```
%define REMOTE_USER = %ENVVAR
 %define myTable = %TABLE(all)
 ...
%HTML (report){
 ...
 @DTWA_myApplet(REMOTE_USER, myTable)
 ...
 %}
```

这将让 Net.Data 生成 <applet> 标记，并对表中的每个值和 REMOTE_USER 环境变量的值生成 <parm> 标记。

此外，可传送表的单列。例如:

```
@DTWA_myApplet(REMOTE_USER, DTW_COLUMN(mycol)myTable)
```

此示例传送 Net.Data 表变量 myTable 的 mycol 列。

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 维护此文件的键，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_DELENTRY 删除条目。DTWR_LISTSUB 在一个 OUT 表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。

生成文档标记

Net.Data 动态生成要由客户机应用程序（如 Web 浏览器）使用的 HTML 或 XML 文档。下列各节描述了各种结构，您可用来通过 Net.Data 宏对文档进行格式化。有关每种结构的特定语法信息，参见 *Net.Data Reference* 中的语言结构章节。

HTML 和 XML 块

客户机应用程序通过指定宏名和其中一个宏的入口点的名称来调用 Net.Data。宏的入口点可以任意为 HTML 块或 XML 块。这些块包含生成结果页的 Net.Data 语言语句和文本表示语句。

因为入口点块驱动宏的执行，所以宏中必须至少存在一个入口点。可以有多个 HTML 块或 XML 块，但每个客户机请求只执行一个入口点块。并且，对于每个请求，将单一文档返回至客户机。要创建由许多客户机文档组成的应用程序，可以使用标准导航技术（如链接和表单）多次调用 Net.Data 来处理各种宏中的各种 HTML 或 XML 块。

HTML 或 XML 块中可以出现任何文本表示语句，只要这些语句对客户机有效。例如，HTML 块可以包含 HTML 或 JavaScript。Net.Data 不执行 JavaScript，但与输出的其余部分一起被发送至客户机，以便执行和显示。在 HTML 或 XML 块中，还可包括函数调用、变量引用和 INCLUDE 语句。下例显示 Net.Data 宏中的 HTML 块的常见用法:

```
%HTML(input){
<h1>Hardware Query Form</h1>
<hr/>
<form method="post" action="report">
<dl>
</dt>What hardware do you want to list?
<dd><input type="radio" name="hdware" value="MON" checked />Monitors</ddl>
<dd><input type="radio" name="hdware" value="PNT" />Pointing devices</ddl>
<dd><input type="radio" name="hdware" value="PRT" />Printers</ddl>
<dd><input type="radio" name="hdware" value="SCN" />Scanners</ddl>
</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()
%}
```

可以从 HTML 链接调用 Net.Data 宏。

```
<a href="http://www.ibm.com/cgi-bin/db2www/equiplst.dtw/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，在此情况下是输入，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“提交”按钮后，客户机请求在 HTML 表单的操作属性中指定的操作。此操作指定调用宏的输出 HTML 块。然后，Net.Data 象处理输入 HTML 块那样处理输出 HTML 块。

Net.Data 然后处理 `myQuery()` 函数调用，该函数调用又调用“SQL 语言环境” FUNCTION 块。用在输入表中返回的值代替 SQL 语句中引用的 `$(hdware)` 变量之后，Net.Data 运行查询。在此点，Net.Data 继续处理报告，根据 REPORT 块中指定的文本显示语句来显示查询的结果。

Net.Data 完成 REPORT 块处理后，它返回至输出 HTML 块并完成处理。

报告块

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。除了 SQL 和 ODBC 语言环境以外，如果没有指定表名称，Net.Data 将使用 FUNCTION 参数列表中第一个输出表的表数据。

REPORT 块具有三部分，每部分都是可选的:

- 标题信息，包含在表行数据之前显示一次的文本。
- ROW 块，包含在结果表的每行上显示一次的文本和表变量。
- 注脚信息，包含在表行数据之后显示一次的文本。

示例:

```
%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.dtw/details?name=$(V1)&loc=$(V2)">$(V1)</a>
    </td>
    <td>$(V2)</td>
  </tr>
  %}
</table>
%}
```

REPORT 块准则

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 宏结果表中的数据。第103页的『表处理变量』中描述了这些变量。有关的附加详细信息，参见 *Net.Data Reference* 中的“报告变量”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示语句对待，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告，就不要在宏中包含 REPORT 块。以下示例显示在 HTML 块中调用函数时的缺省报告格式:

```
 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。
- 要禁用缺省报告的打印，可将 DTW_DEFAULT_REPORT 设置为 NO，或指定一个空的 REPORT 块。例如:

```
%REPORT{%}
```

例: 定制报告

下例显示如何使用特殊变量和 HTML 标记来定制报告格式。它显示来自表 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 浏览器中的结果报告如下所示:

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生成报告是通过:

1. 在报告的开头打印一次 *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. 在报告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告。

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(参见第138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告。

要使用多个 REPORT 块，可以在对每个结果集的存储过程 CALL 中放置一个结果集名称。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 内置函数 DTW_DEFAULT_REPORT = ″MULTIPLE″，则将为每个不与报告块关联的表生成缺省报告。如果没有指定报告块，并且 DTW_DEFAULT_REPORT = ″YES″，则仅生成一个缺省报告。请注意对于 SQL 语言环境来说，DTW_DEFAULT_REPORT 值为 ″YES″ 等价于值 ″MULTIPLE″。

示例: 以下示例演示了使用多个报告块的方式。

要使用缺省的报告格式来显示多个报告:

示例 1: DTW_SQL 语言环境

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

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告，首先显示 table1，然后显示 table2。

示例 2: MACRO_FUNCTION 块。在此例中，两个表被传送到 MACRO_FUNCTION 块中。在指定 DTW_DEFAULT_REPORT=″MULTIPLE″ 的情况下，Net.Data 将对这两个表生成报告。

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

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

示例 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 仅对第一个表显示缺省报告。

要通过对显示处理指定 REPORT 块来显示多个报告:

示例 1: 已命名的 REPORT 块

```
%FUNCTION(dtw_sql) myStoredProc  () {
   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都指定了 REPORT 块。这些表是以它们在 REPORT 块中指定的顺序显示的，先是 table2，然后是 table1。通过在 REPORT 块中指定表名，您可以控制报告显示的顺序。

示例 2: 未命名的 REPORT 块

```
%FUNCTION(dtw_sql) myStoredProc  () {
    CALL myproc

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

在此示例中，对从 myproc 返回的两个结果集指定了 REPORT 块。因为 REPORT 块上未指定任何表名，所以将根据从存储过程返回结果集的次序对头两个结果集执行 REPORT 块。

要使用缺省报告和 REPORT 块的组合来显示多个报告:

示例: 缺省报告和 REPORT 块的组合

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%FUNCTION(dtw_system) editTables (INOUT table1, table2, table3) {
  %REPORT(table2) {
    ...
    %ROW {  ....   %}
    ...
  %}
%}
```

在此例中，仅指定了一个 REPORT 块。因为该块指定 table2，而 table2 是 CALL 语句上所列出的第二个结果集，因此第二个结果集将被用于显示报告。因为指定的 REPORT 块比存储过程返回的结果集个数少，因此对剩余的表显示缺省报告: 先是第一个结果集 table1 的缺省报告，然后是第三个结果集 table3 的缺省报告。指定了一个输出表 table1，它可以被用于今后在宏中进行处理。

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:

- 可对每个结果集指定一个 REPORT 块。
- 以您希望为多个表进行处理的顺序来为它们指定 REPORT 块。
- 当没有为表指定 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 块的表显示了报告之后，将为表显示缺省报告。
- 要防止 Net.Data 显示不具有 REPORT 块的表，必须设置 DTW_DEFAULT_REPORT = ″NO″。
- 当 DTW_SAVE_TABLE_IN 变量与返回多个结果集的函数一起使用时，从函数返回的第一个结果集被指定给 DTW_SAVE_TABLE_IN 表。
- 多个报告块可以与任何语言环境一起使用。

限制:

- 函数中所有报告变量（例如 START_R_N 和 RPT_M_R）的值都适用于该函数中所有 REPORT 块。 您不能修改单个 REPORT 块的报告变量。
- 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。
- 如果第一个报告块指定了一个表名称，那么所有的报告块都必须指定表名称。
- 如果第一个报告块没有指定表名称，那么其他报告块也不能指定表名称。

- 单个存储过程的表的最大数目是 32。

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125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 Reference* 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 宏说明部分中的任何其他块之外指定 IF 块，则在 IF 语句中只允许那些在任何其他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，而后者在说明部分中任何其他块的外部，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。

有效整数字符串的示例:

```
+1234567890
-47
000812
92000
```

无效整数字符串的示例:

```
-  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 块的时侯求解 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: 一个在其他块中包含 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 Reference* 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。

示例: 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>Description
    %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，必须作为动态链接库 (DLL) 或共享程序库实现。有关 Net.Data 提供的语言环境以及如何创建用户编写的语言环境的完整详细信息，参见 *Net.Data 语言环境接口参考*。

图24显示了 Web 服务器、Net.Data 以及 Net.Data 语言环境之间的关系。

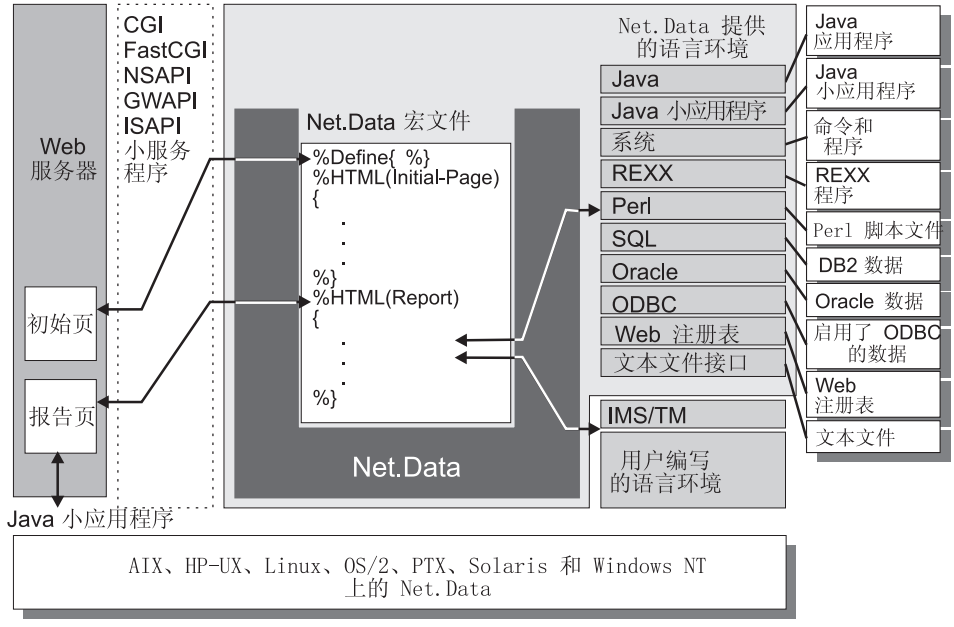


图 24. Net.Data 语言环境

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128页的『Net.Data 提供的语言环境概述』
- 第129页的『调用语言环境』

- 第130页的『关系数据库语言环境』
- 第151页的『编程语言环境』

有关 Net.Data 提供的语言环境的配置信息，参见第28页的『设置 Net.Data 语言环境』。

有关在使用语言环境时改进性能的信息，参见第194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。

Net.Data 提供了两种类型的语言环境:

- 第130页的『关系数据库语言环境』
- 第151页的『编程语言环境』

表7对每个语言环境作了一个简短的描述。参见 *Net.Data Reference* 的操作系统附录，以了解各种操作系统上分别支持哪些语言环境。

表 7. Net.Data 语言环境

语言环境	环境名称	说明
IMS Web	HWS_LE	IMS Web 语言环境可让您使用 IMS Web 来提交 IMS 事务，并从 Web 浏览器接收该事务的输出。
Java 应用程序	DTW_JAVAPPS	Net.Data 支持带有 Java 语言环境的现有 Java 应用程序。
ODBC	DTW_ODBC	ODBC 语言环境通过一个 ODBC 接口来执行 SQL 语句，以便访问多个数据库管理系统。ODBC 语句的结果可在表变量中返回。
Oracle	DTW_ORA	Oracle 语言环境可以让您直接访问您的 Oracle 数据。
Perl	DTW_PERL	Perl 语言环境解释 Net.Data 宏中的 FUNCTION 块内指定的内部 Perl 脚本，或执行存储在单独文件中的外部 Perl 脚本。
REXX	DTW_REXX	REXX 语言环境解释 Net.Data 宏中 FUNCTION 块内指定的内部 REXX 程序，或可以执行存储在一个单独文件中的 REXX 程序。
SQL	DTW_SQL	SQL 语言环境通过 DB2 执行 SQL 语句。SQL 语句的结果可以在表变量中返回。ODBC 语句的结果可在表变量中返回。
System	DTW_SYSTEM	System 语言环境支持执行命令和调用外部程序。

表 7. Net.Data 语言环境 (续)

语言环境	环境名称	说明
Web 注册表	DTW_WEBREG	Web 注册表语言环境为应用程序相关数据的永久性存储器提供函数。

调用语言环境

要调用一个语言环境:

- 通过提供语言语句或 %EXEC 语句，使用 FUNCTION 语句来定义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 CUSTNAME, CUSTNO from ibmuser.customer
 %}
 ...
%HTML(REPORT){
 @custinfo()
 %}
```

处理错误状态的准则

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状态:

- 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（如 SQL 语言环境）将 RETURN_CODE 变量设置为由数据库返回的 SQLCODE，并将 SQL_STATE 变量设置为由数据库返回的 SQLSTATE。参见针对您的 DBMS 的消息和代码文档，以进一步了解您的 DBMS 所使用的 SQLCODE 和 SQLSTATE。

安全性

请确保运行 Net.Data 的用户标识有适当的权限访问那些语言环境语句可引用的任何对象。例如，SQL 语言环境执行 SQL 语句，所以 Net.Data 执行时使用的用户标识必须有权访问数据库资源，才能成功执行。

关系数据库语言环境

Net.Data 提供了关系数据库语言环境，帮助您访问关系数据资源。 您提供的用来访问关系数据的 SQL 语句，将作为动态 SQL 执行。有关动态 SQL 的详情，参见数据库文档。

以下章节描述语言环境以及如何使用这些语言环境:

- 『ODBC 语言环境』
- 第131页的『Oracle 语言环境』
- 第131页的『SQL 语言环境』
- 第132页的『使用 DB2 参数标记』
- 第134页的『管理 Net.Data 应用程序中的事务』
- 第135页的『使用大对象』
- 第138页的『存储过程』
- 第145页的『在结果集中编码 DataLink URL』
- 第146页的『关系数据库语言环境示例』
- 第149页的『Web 注册表语言环境』

ODBC 语言环境

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

要使用 ODBC 语言环境:

要使用 ODBC 语言环境，首先应获取并安装 ODBC 驱动程序和驱动程序管理器。您的 ODBC 驱动程序文档描述了如何安装和配置 ODBC 环境。

验证 Net.Data 初始化文件中是否有类似如下的配置语句，并且在一行上。

注: 以下示例中的路径可能因操作系统的不同而有所变化。

```
ENVIRONMENT (DTW_ODBC) d:/net.data/lib/dtwodbc.dll ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

限制:

- 仅当与 DB2 连接时，ODBC 语言环境才支持存储过程。
- 指定 DATABASE 变量时，必须指定与 ODBC 初始化文件中相同的数据库作为数据源。

- 内联语句块中的 SQL 语句最长可达 64 KB。 DB2 Universal Database 具有以下限制:
  - 版本 6 或更高版本: 64 KB
  - 版本 5 发行版 2 或更低版本: 32 KB

  您使用的数据库可能有不同的限制; 请参考您的数据库文档以确定对该数据库的限制是否有所不同。

Oracle 语言环境

Oracle 语言环境提供了对 Oracle 数据的本机访问。 可以在使用 CGI、FastCGI、NSAPI、 ISAPI 或 APAPI 时, 从 Net.Data 访问 Oracle 数据库。此语言环境支持 Oracle 8.1.5。

要使用 Oracle 语言环境, 验证初始化文件中是否有以下配置语句, 并且是在一行上。

注: 此配置语句中的路径可能因操作系统或设置而有所不同。

```
ENVIRONMENT (DTW_ORA) /net.data/lib/dtwora.so ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

参见第29页的『设置 Oracle 语言环境』以了解如何进一步设置 Oracle 语言环境。

限制:

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

  ```
  LOGIN=admin@ora73
  ```

- 在使用除 CGI 以外的其它接口时, 必须使用 “现场连接”。
- 长型数据类型就象普通字符串一样打包, 且一定***不能***超过 32 KB。
- Net.Data 不支持 Oracle 中返回结果集的存储过程。

SQL 语言环境

SQL 语言环境提供了对 DB2 数据库的访问。在访问 DB2 时, 使用此语言环境以获得最佳的性能。

要使用 SQL 语言环境, 验证初始化文件中是否有以下配置语句, 并且是在一行上。

注: 此配置语句中的路径可能因操作系统或设置而有所不同。

```
ENVIRONMENT (DTW_SQL) d:/net.data/lib/dtwsql.dll (IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

嵌套的 SQL 语句

可以调用在另一个 SQL 函数中的 SQL 函数。如果传送了表，则应确保在每个函数中使用唯一的表名；否则，可能会出现意外的结果。

示例: 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) %}
```

限制:

- Linux S/390 不支持嵌套式 SQL。
- 内联语句块中的 SQL 语句最长可达 64 KB。 DB2 Universal Database 具有以下限制:
- 版本 6 或更高版本: 64 KB
- 版本 5 发行版 2 或更低版本: 32 KB

  您使用的数据库可能有不同的限制；请参考您的数据库文档以确定对该 DBMS 的限制是否有所不同。
- 在嵌套 SQL 语句时，任何给定时间可处理的结果集的最大数目为 32。例如，可嵌套三层，每层返回 10 个结果集。或嵌套 32 层，每层返回一个结果集。

使用 DB2 参数标记

如果使用正确，参数标记可通过允许 DB2 利用其高速缓存来提高查询的性能。在 SQL 语句中，参数标记是一个问号 (?)，指示在执行该语句时替代应用程序提供的值的位置。该值是从 Net.Data SQL 函数定义的参数列表上的 Net.Data 变量获得的。获取这些值的方式依据使用参数标记的方式而定。

可以两种方式来使用参数标记:

- 显式
- 隐式

显式使用参数标记:

在创建 SQL 语句时，可将参数标记人工添加至查询。

例如:

```
%FUNCTION (DTW_SQL) select_staff(in id, in dept){
     select * from staff
     where id = ? and dept = ?
     and salary = 35,000%}
```

对于每个参数标记，DTW_SQL 函数中都会有一个对应的 IN 参数。对于 SQL 和函数参数列表，映射次序都是从左至右。**未**与 SQL 参数标记相关联的函数参数可放在函数参数列表的末尾。

隐式使用参数标记:

通过在初始化文件或宏中设置以下标志: DTW_USE_DB2_PREPARE_CACHE = YES，以便隐式使用参数标记。如果“DB2 准备高速缓存”配置变量被设置为 YES，Net.Data 会将 SQL 语句中的每个变量替换为参数标记。数据被绑定至每个参数标记，且不会从 Net.Data 参数列表传送出来（就好象显式使用参数标记一样）。

例如:

```
%FUNCTION (DTW_SQL) select_staff() {
     select * from staff
     where id = $(ID) and dept = $(dept)
     and salary = 35,000%}
```

限制:

- 参数标记仅在 DB2 上可用。
- 对于显式参数标记，DTW_USE_DB2_PREPARE_CACHE 中的配置变量标志在初始化文件或宏中必须设置为 NO。
- 对于隐式参数标记，SQL 语句中的所有 Net.Data 变量都必须处于标记位置。如果没有这样的话，将发生语法错误，因为 Net.Data 不知道哪个变量是合法标记。

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 的提交语句，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回滚语句，撤销上一次提交之后所作的全部修改。

Net.Data 发送提交和可能的回滚语句的方式取决于 TRANSACTION_SCOPE 的设置以及宏中是否显式地指定了提交语句。TRANSACTION_SCOPE 的值可以是 MULTIPLE 和 SINGLE。缺省值为 MULTIPLE。要将 TRANSACTION_SCOPE 设置为 SINGLE，使用 %DEFINE 语句或者调用 @DTW_ASSIGN(), 并对适当的 LE 传送 ENVIRONMENT 语句上的变量。有关详情，参见本书第二章中的『定制 Net.Data 初始化文件』。

SINGLE

指定 Net.Data 在每个成功完成的 SQL 语句后都发出一个提交语句。如果 SQL 语句返回错误，则发出一个回滚语句。SINGLE 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，今后就不可能使用回滚语句来撤销所做的修改。

MULTIPLE

指定 Net.Data 将在发出提交语句之前执行所有的 SQL 语句。Net.Data 在请求的最后发送提交，如果每个 SQL 语句都已成功发出，提交将使数据库中所有的修改都变为永久性的修改。如果任何一个语句返回错误，则 Net.Data 就会在产生错误时发出回滚语句，该语句将把数据库设置回先前的状态。

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提交语句，这样，您这个应用程序开发者就可以对应用程序中的提交和回滚行为进行完全的控制。例如，在每次对宏进行更新之后发出提交语句将有助于确保数据的完整性。

要发出 SQL 提交语句，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()
  ...
%}
```

限制:

在连接数据库之后，便不能更改 TRANSACTION_SCOPE 的设置。因此，宏中的所有 SQL 事务都属于同一处理过程。

如果使用 Net.Data 作为 Net.Commerce 的一部分，则需注意，Net.Commerce 有它自己的事务处理，并禁用了 Net.Data 的事务处理。

使用大对象

可以将大对象文件 (LOB) 存储在 DB2 数据库中，并使用 Net.Data SQL 或 ODBC 语言环境 来将它们合并成动态的 Web 页面。

当语言环境执行 SQL SELECT 语句或者返回 LOB 的存储过程时，它不会为 V(*n*) 表处理变量或 Net.Data 表字段指定对象。然而，它将 LOB 存储在 一个由 Net.Data 创建的文件中，并且只在 V(*n*) 表处理变量或 Net.Data 表字段中返回该文件的名称。在 Net.Data 宏中，可以使用名称来引用 LOB 文件；例如，可以通过超文本引用来创建 HTML 锚元素或者包含该文件的 URL 的映象元素。Net.Data 将包含 LOB 的文件放在由 HTML_PATH 配置变量指定的目录中，这些配置变量位于 Net.Data 初始化文件 (db2www.ini) 中。对 LOB 文件的写访问权仅限于与检索 LOB 的 Net.Data 请求相关联的用户标识。

LOB 的文件名是动态构造的，并且具有以下格式:

name[.*extension*]

其中:

name 是动态生成的、用来标识大对象的唯一字符串

extension

是用来标识对象的类型的字符串。对于 CLOB 和 DBCLOB，扩展名为 .txt。对于 BLOB，SQL 语言环境通过在 LOB 文件的最前面的一些字节中查找签名来确定扩展名。表8显示 SQL 语言环境所使用的 LOB 扩展名:

表 8. 在 SQL 语言环境中使用的 LOB 扩展名

扩展名	对象类型
.bmp	位图图像
.gif	图形图像格式
.jpg	联合图像专家组 (JPEG) 图像
.tif	带有标记的图像文件格式
.ps	PostScript
.mid	乐器数字界面 (midi) 音频
.aif	AIFF 音频
.avi	音频视频交错音频
.au	基本音频

表 8. 在 SQL 语言环境中使用的 LOB 扩展名 (续)

扩展名	对象类型
.ra	实际音频
.wav	windows 音频视频
.pdf	可移植文档格式
.rmi	midi 序列

如果不能识别 BLOB 的对象类型，则不会为文件名添加扩展名。

当 Net.Data 返回包含 LOB 的文件名时，它将使用以下语法来为文件名添加前缀 /tmplobs/:

```
/tmplobs/name.[extension]
```

此前缀允许您在不是 Web 服务器的文档根目录的目录中查找 LOB 目录。

要确保正确地解析对 LOB 文件的引用，应将以下 Pass 伪指令添加到 Web 服务器的配置文件中:

```
Pass    /tmplobs/*     <full_path>/tmplobs/*
```

<full_path> 是在 Net.Data 初始化文件中为 HTML_PATH 配置变量指定的值。

计划提示: 返回 LOB 的每次查询都会导致在由 HTML_PATH 路径配置变量指定的目录中创建文件。在使用 LOB 时请考虑系统限度，因为它们会很块地消耗资源。您可能想定期清理目录，或者执行 dtwclean 守护程序。有关详情，参见第137页的『管理临时 LOBS』。建议您使用 DataLinks，它不需要由 SQL 语言环境来将文件存储在目录中，从而获得更好的性能并且使用更少的系统资源。

示例: 以下应用程序使用 MPEG 音频 (.mpa) 文件。因为 SQL 语言环境不识别此文件类型，所以使用 EXEC 变量来将 .mpa 扩展名追加到文件名中。此应用程序的用户必须单击文件名才能调用 MPEG 音频文件查看器。

```
%DEFINE{
lobdir="/u/IBMUSER/tmplobs""
myFile=%EXEC "rename $(lobdir)$(filename) $(lobdir)$(filename).mpa"
%}
%{ where rename is the command on your operating system to rename files %}
%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
  %REPORT{
    <p>Here is the information you selected:</p>
    %ROW{
      @DTW_ASSIGN(filename, @DTW_rSUBSTR(V3, @DTW_rLASTPOS("/", V3)))
      $(myFile)
      $(V1) <img src="$(V2)">
            <a href="$(V3).mpa">Voice sample</a><p>
    %}
```

```
  %}
%}

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

如果 RepProfile 表中包含关于 Kinson Yamamoto 和 Merilee Lau 的信息，则执行 REPORT 块时将把以下 HTML 添加到所生成的 Web 页面中:

```
<p>Here is the information 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>
```

先前示例中的 REPORT 块使用隐式表变量 V1、V2 和 V3。

- V1 的值是人名，它是字符数据。
- V2 的值是包含人的照片的 GIF 文件的名称。显示的图像直接插入在所生成的 Web 页面中。
- V3 的值是包含人的声音样本的 MPA 文件的名称。因为 Net.Data 不识别 MPA 文件格式，所以当它为由 HTML_PATH 指定的目录中的 LOB 创建文件时，不会为文件名添加扩展名。此示例说明了如何使用 EXEC 变量来为文件名添加 .mpa 扩展名。当用户单击文本“声音样本”（它是超链接文本）时，将播放声音样本。

对 LOB 的访问权:

在所提供的 Net.Data 初始化文件中，LOB 的缺省 tmplobs 目录在由 HTML_PATH 指定的目录下面。任何用户标识都可以访问它。如果更改了 HTML_PATH 值，则应确保运行 Web 服务器的用户标识对于由 HTML_PATH 指定的目录具有写访问权（有关详情，参见第16页的『HTML_PATH』）。 **管理临时 LOB**:

Net.Data 将临时 LOB 存储在一个称为 tmplobs 的子目录中，该子目录在 HTML_PATH 路径配置变量指定的目录下面。这些文件可能很大，应该定期清除，以便维持满意的性能。

Net.Data 提供了一个称为 dtwclean 的守护程序，它可以帮助您定期管理 tmplobs 目录。dtwclean 使用端口 7127。

要运行 dtwclean 守护程序: 在命令行窗口中输入以下命令:

```
dtwclean [-t xx] [-d|-l]
```

其中:

-t 是一个标志，它指定 dtwclean 清理目录所采用的时间间隔

xx 是在 dtwclean 删除文件之前，该文件保留在目录中的时间间隔（以秒计）。对此值没有限制。缺省值为 3600 秒。

-d 是一个用来指定调试方式的标志；跟踪消息显示在命令窗口中。

-l 是一个用来指定日志记录方式的标志；跟踪消息被打印至日志文件中。

存储过程

存储过程就是存储在数据库中、可以执行 SQL 语句的已编译程序。在 Net.Data 中，存储过程是使用 CALL 语句来从 Net.Data 函数中调用的。存储过程参数是从 Net.Data 函数的参数列表中传送来的。可以使用存储过程来改进性能和完整性，即，通过对数据库服务器保持已编译的 SQL 语句来实现。Net.Data 通过 SQL 和 ODBC 语言环境来支持将存储过程与 DB2 配合使用。Oracle 存储过程是通过 Oracle 语言环境来支持的。对于特定的 DB2，Net.Data 支持存储过程返回一个或多个结果集。

本节描述下列主题:

- 『存储过程语法』
- 第139页的『调用存储过程』
- 第141页的『传送参数』
- 仅对于 DB2: 第141页的『处理来自 DB2 存储过程的结果集』

存储过程语法

用于存储过程的语法包括 FUNCTION 语句、CALL 语句以及可选的 REPORT 块。

```
%FUNCTION (DTW_SQL) function_name ([IN datatype arg1, INOUT datatype arg2,
    OUT resultsetname, ...]) {
  CALL stored_procedure [(resultsetname, ...)]
[%REPORT [(resultsetname)] { %}]
...
[%REPORT [(resultsetname)] { %}]
[%MESSAGE %}]

%}
```

其中:

function_name
是启动调用存储过程的 Net.Data 函数的名称

stored_procedure
是存储过程的名称

datatype

是 Net.Data 所支持的其中一种数据库数据类型，如表9 和 表10中所示。在参数列表中指定的数据类型必须与存储过程中的数据类型相匹配。有关这些数据类型的详情，参见数据库文档。

仅对于 DB2: *tablename*

是用来存储结果集的 Net.Data 表的名称（仅当结果集要存储在 Net.Data 表中时才使用）。如果指定了该名称，则此参数名必须与 *resultsetname* 的相关联的参数名相匹配。

仅对于 DB2: *resultsetname*

是用来将存储过程中返回的结果集与 REPORT 块和函数 parm 列表上的表名（或者这两者）关联起来的名称。REPORT 块上的 *resultsetname* 必须与CALL 语句上的结果集相匹配。

表 9. 用于 DB2 的受支持的存储过程数据类型

BIGINT	DOUBLEPRECISION	SMALLINT
CHAR	FLOAT	TIME
CLOB[1]	INTEGER	TIMESTAMP
DATE	GRAPHIC	VARCHAR
DECIMAL	LONGVARCHAR	VARGRAPHIC
DOUBLE	LONGVARGRAPHIC	

[1] CLOB 只能用作 OUT 和 INOUT 参数，并且 Net.Data 会解释大小（以字节计）。例如，如果将变量指定为 OUT CLOB(20000)，则将把大小为 20K 的 CLOB 用作输出参数。

表 10. 用于 Oracle 的受支持的存储过程数据类型

BIGINT	LONG
CHAR	LONG RAW
DATE	NUMBER
DECIMAL	RAW
FLOAT	VARCHAR / VACHAR2
INTEGER	

要点: 当 Windows 或 Unix 上的 Net.Data 调用位于 OS/390 和 OS/400 上的 DB2 中的存储过程时，当从 DB2 数据库中检索 DECIMAL 数据时，这些操作系统上的存储过程必须使用主变量类型 DOUBLE 或 FLOAT。使用主变量类型 DOUBLE 或 FLOAT 将确保所返回的数据采用可读格式。

调用存储过程

1. 定义用来启动调用存储过程的函数。

```
%FUNCTION (DTW_SQL) function_name()
```

2. （可选）为存储过程指定任何 IN、INOUT 或 OUT 参数。对于 DB2 存储过程，参数可以包括用来将结果集存储在 Net.Data 表中的表变量名（如果您想要结果集存储在 Net.Data 表中，则只需要指定 Net.Data 表）。

```
%FUNCTION (DTW_SQL) function_name (IN datatype
arg1, INOUT datatype arg2,
    OUT resultsetname...)
```

3. 使用 CALL 语句来标识存储过程名。

```
 CALL stored_procedure
```

4. 对于 DB2: 如果 DB2 存储将生成一个结果集，则可以选择指定 REPORT 块，以便定义 Net.Data 如何显示结果集。

```
%REPORT [(resultsetname)] {
...
%}
```

   示例:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1)  {
    CALL myproc
 %REPORT (mytable){
  ...
  %ROW {  ...   %}
  ...
 %}
%}
```

5. 如果存储过程将生成多个结果集:
   - 在 CALL 语句上指定结果集名。

     ```
      CALL stored_procedure[ (resultsetname1[, resultsetname2, ...]) ]
     ```

   - 可以选择指定一个或多个 REPORT 块来定义 Net.Data 如何显示结果集。

     ```
     %REPORT[(resultsetname1)] {
     ...
     %}
     ```

     示例:

     ```
     %FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1, OUT table1) {
        CALL myproc (table1, table2)
       %REPORT(table2) {
            ...
            %ROW {  ...   %}
            ...
      %}
     %}
     ```

6. 除了存储过程之外，Oracle 还提供了存储函数。要调用 Oracle 存储函数，必须使用 Net.Data 变量 DTWORA_RESULT。在执行之后，DTWORA_RESULT 中包含存储函数的返回值。

示例:

```
%FUNCTION (DTW_ORA) orastp (IN datatype arg1, OUT datatype arg2,...)
   returns (DTWORA_RESULT) {
   CALL stored_oracle_function
   %}
```

传送参数

可以将参数传送给存储过程，并且可以使存储过程更新参数值，以便将新值传送回 Net.Data 宏。函数参数列表上的参数数目和类型必须与为存储过程定义的数目和类型相匹配。例如，如果在参数列表上为存储过程定义的参数是 INOUT，则函数参数列表上相应的参数必须是 INOUT。如果在列表上为存储过程定义的参数的类型为 CHAR(30)，则函数参数列表上相应的参数也必须是 CHAR(30)。

示例 1: 将参数值传送给存储过程

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) valuein)  {
    CALL myproc
...
```

示例 2: 从存储过程中返回值

```
%FUNCTION (DTW_SQL) mystoredproc (OUT VARCHAR(9) retvalue)  {
    CALL myproc
...
```

处理来自 DB2 存储过程的结果集

使用 SQL 或 ODBC 语言环境可以从存储过程中返回一个或多个结果集。可以将结果集存储在 Net.Data 表中，以便在宏中进一步处理或者使用 REPORT 块来处理。如果存储过程生成了多个结果集，则必须将名称与存储过程生成的每个结果集相关联。这是通过在 CALL 语句上指定参数来实现的。于是，您为结果集指定的名称可以与 REPORT 块或 Net.Data 表相关联，并允许您确定如何由 Net.Data 来处理每个结果集。您可以:

- 通过不为结果集定义报告块来以 Net.Data 的缺省报告样式来处理结果。
- 将结果集与 REPORT 块相关联，以便应用您自己的报告样式。在 REPORT 块中，可以使用 Net.Data 变量、文本处理语句（例如 HTML 或 JavaScript）或者其他函数来指定在浏览器中如何显示报告数据。
- 当您想要 Net.Data 稍后在宏中使用数据时，将结果集存储在 Net.Data 表中。例如，可以将 Net.Data 表传送给另一个函数，以便它可以将数据用于计算，并根据那些计算来显示结果。

有关使用多个报告块时的指南和限制，参见第122页的『多个 REPORT 块的准则和限制』。

要返回单个结果集并使用缺省报告:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name () {
    CALL stored_procedure
%}
```

例如:

```
 %FUNCTION (DTW_SQL) mystoredproc() {
    CALL myproc
%}
```

要返回单个结果集并指定 REPORT 块:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name () {
     CALL stored_procedure  [(resultsetname)]
 %REPORT [(resultsetname)] {
  ...
 %}
%}
```

示例 1:

```
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
 %REPORT {
      ...
      %ROW {  ...   %}
      ...
 %}
%}
```

示例 2:

```
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc (mytable1)
 %REPORT (mytable1) {
  ...
  %ROW {  ...   %}
  ...
 %}
%}
```

要将单个结果集存储在 Net.Data 表中以进行进一步处理:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename) {
   CALL stored_procedure [(resultsetname)]
%}
```

例如:

```
%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {
    CALL myproc
%}
```

注意，DTW_DEFAULT_REPORT 被设置为否，以便不为结果集生成缺省报告。

要返回多个结果集并使用缺省报告格式化来显示它们:

使用以下语法:

```
%FUNCTION (DTW_SQL) function_name () {
    CALL stored_procedure [(resultsetname1, resultsetname2, ...)]
%}
```

此处未指定报告块。

例如:

```
 %DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
%}
```

要返回多个结果集，并将结果集存储在 Net.Data 表中以便进行进一步处理:

使用以下语法:

```
 %FUNCTION (DTW_SQL) function_name (OUT (resultsetname1, resultsetname2, ...) {
    CALL stored_procedure (resultsetname1, resultsetname2, ...)
%}
```

例如:

```
%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
     CALL myproc (mytable1, mytable2)
%}
```

注意，DTW_DEFAULT_REPORT 被设置为否，以便不为结果集生成缺省报告。

要返回多个结果集并为显示处理指定 REPORT 块:

每个结果集都与它的一个或多个 REPORT 块相关联。使用以下语法:

```
%FUNCTION (DTW_SQL) function_name (, ...) {
   CALL stored_procedure (resultsetname1, resultsetname2, ...)
 %REPORT (resultsetname1)
   ...
   %ROW {  ...   %}
   ...
```

```
 %}
 %REPORT (resultsetname2)
   ...
   %ROW {  ...   %}
   ...
 %}

...
%}
```

例如:

```
%FUNCTION (DTW_SQL) mystoredproc () {
   CALL myproc (mytable1, mytable2)

 %REPORT(mytable1) {
  ...
  %ROW {  ...   %}
  ...
 %}

 %REPORT(mytable2) {
  ...
  %ROW {  ...   %}
  ...
 %}
%}
```

要返回多个结果集并为每个结果集指定不同的显示或处理选项:

可以为使用唯一的参数名的每个结果集指定不同的处理选项。**示例 1**:

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable2) {
    CALL myproc (mytable1, mytable2, mytable3)

 %REPORT(mytable1) {
 ...
 %ROW {  ...   %}
 ...
 %}
%}
```

结果集 mytable1 是由相应的 REPORT 块来处理的，并且是由宏的编写者指定时才显示。结果集 mytable2 被存储在 Net.Data 表 mytable2 中，并且现在可用于进一步的处理，例如，传送给另一个函数。结果集 mytable3 是使用 Net.Data 的缺省报告格式来显示的，因为没有为它指定 REPORT 块。

示例 2:

```
%FUNCTION(DTW_SQL) mystoredproc(OUT mytable4, OUT mytable3) {
    CALL myproc (mytable1, mytable2, mytable3, mytable4)
    %REPORT(mytable2) {
    ...
```

```
    %ROW { ... %}
    ...
    %}
    %REPORT(mytable1) {
    ...
    %ROW { ... %}
    ...
    %}
    %REPORT(mytable4) {
    ...
    %ROW { ... %}
    ...
    %}
 %}
```

结果集 *mytable2*、*mytable1* 和 *mytable4* 是由它们的相应 REPORT 块按照这种次序来处理的，并且根据指定来显示。结果集 *mytable4* 和 *mytable3* 被存储在表变量中，以便进行进一步处理。在处理完三个 REPORT 块之后，结果集 *mytable3* 也将使用 Net.Data 的缺省报告格式在显示。

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 文件管理器的详情，参见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 内置函数 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。

示例 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 列中指定的文件就可以在任何 Web 浏览器上引用。

示例 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。

关系数据库语言环境示例

以下示例显示如何从宏调用关系数据库语言环境:

ODBC

以下示例为 ODBC 语言环境定义和调用多个函数。

```
%DEFINE{
   DATABASE="qesq1"
   SHOWSQL="YES"
   table="int_null"
   LOGIN="netdata1"
   PASSWORD="ibmdb2"%}

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

%function(dtw_odbc) sql2() {
```

```
insert into $(table) (int1) values (111)
%}

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

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

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

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

Oracle

以下示例显示了一个具有 DTW_ORA 函数定义的宏，它查询 Oracle 数据库 udatabase，使用一个变量引用来确定要查询的数据库表。FUNCTION 块也包含一个处理错误状态的 MESSAGE 块。当 Net.Data 处理宏时，它将在浏览器上显示一个缺省的报告。

```
%DEFINE{
   LOGIN="ulogin"
   PASSWORD="upassword"
   DATABASE=""
   table= "utable"
%}


%FUNCTION(DTW_ORA) myQuery(){
select ename,job,empno,hiredate,sal,deptno from $(table) order by ename
%}
%MESSAGE{
100 : "<b>WARNING</b>: No employee were found that met your search criteria.<p>"
    : continue
%}

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

SQL

以下示例显示了具有 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 语言环境根据参

数的数据类型将每个参数传递给存储过程。当存储过程完成处理之后，将返回输出参数，Net.Data 也将相应地更新变量。

```
%{************************************************************
                    DEFINE BLOCK
************************************************************%}
%DEFINE{
 MACRO_NAME     = "TEST ALL TYPES"
  DTW_HTML_TABLE      = "YES"
 parm1          = "1"              %{SMALLINT      %}
 parm2          = "11"             %{INT         %}
 parm3          = "1.1"            %{DECIMAL (2,1)  %}
 %}

 %FUNCTION(DTW_SQL)   myProc
  (INOUT SMALLINT     parm1,
   INOUT INT            parm2,
   INOUT DECIMAL(2,1)    parm3){
CALL TESTTYPE
%}
%HTML (report){
 <head>
<title>Net.Data : SQL Stored Procedure: Example '$(MACRO_NAME)'. </title>
 </head>
<body bgcolor="#bbffff" text="#000000" link="#000000">
<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:
</h2>
<b>parm1  (SMALLINT)<p></b>
$(parm1)<br />
<b>parm2  (INT)</b>
$(parm2)<br />
<b>parm3  (DECIMAL)</b>
$(parm3)
<hr/>
<h2>
Calling the function that executes the stored procedure.
</h2>
<p>
 @myProc(parm1,parm2,parm3)
</p><hr/>
<h2>
Values of the INOUT parameters after
calling the stored procedure:<p>
</h2>
<p><b>parm1  (SMALLINT)</b><br />
$(parm1)<br />
<b>parm2  (INT)</b>
$(parm2)<br />
```

```
<b>parm3  (DECIMAL)</b>
$(parm3)
</p></body>
%}
```

Web 注册表语言环境

Net.Data Web 注册表为与应用程序相关的数据提供了持久性的存储器。Web 注册表可用于存储配置信息和其它能够被基于 Web 的应用程序在运行时动态访问的数据。您只能这样来访问 Web 注册表: 从为此目的而编写的 CGI 程序出发，通过 Net.Data 宏并使用 Net.Data 和 Web 注册表内置支持。Web 注册表在操作系统的子集上也是可用的。参见 *Net.Data 参考*以获取对 Web 注册表内置函数的描述、语法以及支持该语言环境的操作系统列表。

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

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

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

表11显示了一个示例 Web 注册表:

表 11. 示例 Web 注册表

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

使用 Web 注册表时要考虑的一些因素:

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

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

```
RegistryVariable/Index
```

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

表 12. 示例索引 Web 注册表

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

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

配置 Web 注册表语言环境

验证初始化文件中是否有以下配置语句，并且是在一行上:

```
ENVIRONMENT (DTW_WEBREG)   DTWWEB    ( OUT RETURN_CODE )
```

参见第26页的『环境配置语句』，以进一步了解 Net.Data 初始化文件和语言环境 ENVIRONMENT 语句。

调用 Web 注册表内置函数

与调用其它函数一样调用 Web 注册表函数。使用 DEFINE 语句来定义与您想要传递的参数类似的变量。例如:

```
%DEFINE{
    name = "smith"
 %}
```

然后用一个函数调用语句来调用该函数; 例如:

```
@DTWR_ADDENTRY("URLLIST", name, "http://www.ibm.com/software/",
    "WORK_URL"
```

示例

以下示例将创建一个 Web 注册表并在其中添加条目。然后，它将显示一个包含这些条目的报告。

```
%DEFINE{
RegTable = %TABLE(ALL)
%}

%MESSAGE {
    default:"<p>Function Error: Return code: $(RETURN_CODE)." :continue
%}

%FUNCTION(DTW_WEBREG) ListTable(INOUT RegTable) {
%}

%HTML (report){
@DTWR_CREATEREG("MYREG")
@DTWR_ADDENTRY("MYREG", "Dept. 1", "Payroll")
@DTWR_ADDENTRY("MYREG", "Dept. 2", "Technical Support")
@DTWR_ADDENTRY("MYREG", "Dept. 3", "Research")
@DTWR_LISTREG("MYREG", RegTable)

<p>Report:<br />
@ListTable(RegTable)

%}
```

编程语言环境

在调用外部程序时，Net.Data 提供以下语言环境供您使用:

- 『Java 应用程序语言环境』
- 第155页的『Perl 语言环境』
- 第158页的『REXX 语言环境』
- 第164页的『System 语言环境』

访问权限: 请确保执行 Net.Data 的用户标识有权执行程序，并对该程序可能访问的任何对象具有访问权。参见第57页的『对 Net.Data 访问的文件授予访问权限』，以获取详情。

Java 应用程序语言环境

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

配置 Java 语言环境

要使用 Java 语言环境，您需要验证 Net.Data 的初始化设置并设置语言环境。

验证初始化文件中是否有类似如下配置语句的语句，并且是在一行上:

```
ENVIRONMENT (DTW_JAVAPPS) /opt/IBMNetData/lib/libdtwjava.so ( OUT RETURN_CODE )
   {% CLIETTE "DTW_JAVAPPS" %}
```

在此示例中，ENVIRONMENT 条目末尾的 CLIETTE 字符串被注释掉。如果 CLIETTE 字符串未被注释掉，Net.Data 会通过 “Net.Data 现场连接” 尝试调用 Java 应用程序。在使用不同于 CGI 和 FCGI 的接口运行 Net.Data 时，使用 “Net.Data 现场连接” 是必需的。如果 CLIETTE 字符串**未**被注释掉，且 “Net.Data 现场连接” **未**在运行，Net.Data 将尝试直接执行 Java 应用程序。但是，如果使用不同于 CGI 和 FCGI 的接口运行 Net.Data，可能会导致不可预见的行为。

参见第26页的『环境配置语句』，以进一步了解 Net.Data 初始化文件和语言环境 ENVIRONMENT 语句。

要点: 如果想要通过 “现场连接” 调用 Java 应用程序，则参见第28页的『设置带有 Cliette 的 Java 语言环境』以了解如何设置 Java 语言环境。

调用 Java 函数

借助 Java 语言环境，可从 Net.Data 宏发出 Java 函数调用，并将 Net.Data 字符串用作参数。这些调用的 Java 函数调用会返回字符串。

Net.Data 提供了两个方法来调用 Java 函数:

1. Net.Data 可通过将 Java 语言环境装入 Net.Data 进程直接启动 Java 应用程序。

   在设置和配置方面，这是最简单的方法。如果通过 CGI 或 FCGI 运行 Net.Data，建议使用此方法。此方法在 Windows、AIX、Linux 和 Solaris 上都可用。

2. Net.Data 通过 “Net.Data 现场连接” 连接至 Java 语言环境。

   如果将 Net.Data 作为 Web 服务器 API 运行，则建议使用此方法。Java 应用程序通过 “现场连接” 作为单独的进程运行，且对于正在处理的其他 Java 的潜在干涉是 “屏蔽” 的。此方法在 Windows、OS/2 和 AIX 上可用。

设置 Net.Data 调用 Java 函数的过程与您使用的方法无关。

通过装入 Java 应用程序来调用 Java 函数: 要通过装入 Java 应用程序来调用 Java 函数:

1. 编写 Java 函数并将源代码放入 Net.Data 提供的 Java 函数样本文件 `UserFunctions.java` 中。

2. 将 gnu.regexp-1.0.8.jar 文件添加至 CLASSPATH 设置。例如，在 AIX 中，添加路径:

   ```
   /opt/IBMNetData/bkends/javaapps/gnu.regexp-1.0.8.jar
   ```

   gnu.regexp-1.0.8.jar 文件是与 Net.Data 一起交付的。
3. 创建 Java 类文件:
   a. 在包含文件 UserFunctions.java 的目录中执行批处理文件 rebuild。
   b. 成功完成时，rebuild 会生成两个 Java 类文件 dtw_getsignature.class 和 dtw_userfunction.class。
4. 将 Java 类文件 dtw_getsignature.class 和 dtw_userfunction.class 放入 Net.Data Java 语言环境可找到的目录。通常，这是在 CLASSPATH 设置中指定的目录。
5. 运行调用 Java 语言环境的 Net.Data 宏。确保 Net.Data 配置文件中的 “Java 应用程序” 语言环境条目没有 cliette 字符串 CLIETTE "DTW_JAVAPPS"，或此 cliette 字符串已被注释掉。

注： 每当 Java 函数源代码发生更改时，遵循这些步骤。此方法对 Windows、AIX、Linux 和 Solaris 是可用的。

通过 “Net.Data 现场连接” 调用 Java 函数: 要通过 “现场连接” 调用 Java 函数:

1. 编写您的 Java 函数。
2. 为所有 Java 函数创建 Net.Data cliette。

   Net.Data cliette 启动运行 Java 函数的 “Java 虚拟机”。
3. 在 “现场连接” 配置文件的 Java ENVIRONMENT 语句中定义一个 cliette。

   每当您引入新的 Java 函数时，都必须重新创建 Java cliette。
4. 启动 “连接管理器”。
5. 运行调用 Java 语言环境的 Net.Data 宏。

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

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

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

- dtw_samp.java

- UserFunctions.java
- myfile.java

创建 Java 函数: 使用您自己的 Java 函数来修改 Java 函数样本文件 UserFunctions.java:

```
====================UserFuctions.java====================
import mypackage.*
 public String myfctcall(...parameters from macro...)
{
 return ( mypackage.mymethod(...parameters...));
 }

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

Java 语言环境文件结构

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

- 一个名为 UserFunctions.java 的示例 Java 函数。
- 一个名为 makeClas 的示例文件。在运行时，这个文件将为您的 Java 函数创建一个 Net.Data cliette。
- 一个名为 launchjv 的示例文件，Net.Data cliette 用它来启动 Java 虚拟机并运行 Java 函数。
- Net.Data 通过装入 Java 语言环境来使用样本文件 rebuild 构建 Java 函数以直接执行。

表13 描述了您操作系统上那些文件的目录与文件名。

表 13. 创建 Java 函数时所使用的文件

操作系统	文件名	目录
OS/2	UserFunctions.java	javaapps
	launchjv.com	connect
Windows NT	UserFunctions.java	javaclas
	makeClas.bat	javaclas
	launchjv.bat	connect

表 13. 创建 Java 函数时所使用的文件 (续)

操作系统	文件名	目录
AIX	UserFunctions.java	javaapps
	launchjv	javaapps
	rebuild	javaapps
	gnu.regexp-1.0.8.jar	javaapps
Linux & Solaris	用户 Functions.java	javaapps
	rebuild	javaapps
	gnu.regexp-1.0.8.jar	javaapps

Java 语言环境示例

在创建了 Java 函数、定义了 cliette 类或运行了 rebuild 并配置了 Net.Data 之后，您就可以运行包含对 Java 函数引用的宏。

以下示例宏演示函数定义和 Java 应用程序函数 reverse_line() 的函数调用。

```
%{ to call the sample }
%FUNCTION (DTW_JAVAPPS) reverse_line(str);

%HTML (report){
you should see the string "Hello World" in reverse.
@reverse_line("Hello World")
You should have the result of your function call.
%}
```

Perl 语言环境

Perl 语言环境能够解释在 Net.Data 宏的一个 FUNCTION 块中指定的在线 Perl 脚本，或者它能处理存储在服务器上单独文件中的外部 Perl 脚本。

配置 Perl 语言环境

验证 Net.Data 初始化文件中有以下配置语句，并且是在一行上:

```
ENVIRONMENT (DTW_PERL)     DTWPERL   ( OUT RETURN_CODE )
```

参见第26页的『环境配置语句』，以进一步了解 Net.Data 初始化文件和语言环境 ENVIRONMENT 语句。

日文用户: 日文 SJIS 字符集的某些字符可能会被 Perl 错误地解释成控制字符。有一个名为 jperl 的开放源代码软件包可解决此问题。下载并安装该软件包，然后在 Perl 脚本的标题中包括语句`use I18N::Japanese.pm`。

调用外部的 Perl 脚本

对外部 Perl 脚本的调用在 FUNCTION 块中是由一个 EXEC 语句来标识的:

```
%EXEC{ perl_script_name [optional parameters] %}
```

必需: 请确保 Perl 脚本名称 *perl_script_name* 列在 Net.Data 初始化文件中为 EXEC_PATH 配置变量指定的路径中。

```
%FUNCTION(DTW_PERL) perl1() {
%EXEC{ MyPerl.pl %}
%}
```

传递参数

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

直接 在调用 Perl 脚本时直接传递参数。例如:

```
%DEFINE INPARM1 = "SWITCH1"

%FUNCTION(DTW_PERL) sys1() {
 %EXEC{
    MyPerl.pl $(INPARM1) "literal string"
 %}
%}
```

Net.Data 变量 INPARM1 被引用并被传递到 Perl 脚本。参数传递到 Perl 脚本的方式与从命令行调用 Perl 脚本时参数传递到 Perl 脚本的方式相同。使用此方法传送至 Perl 脚本的参数被视作输入类型参数。使用此方法传送至 Perl 脚本的参数被视作输入参数，而对这些值的任何修改都不会反映给 Net.Data。

间接

使用下列其中一种方法在调用 Perl 脚本时直接传送参数:

- 让 Net.Data 将输入参数作为环境变量传递给 Perl 脚本。然后， Perl 脚本就可以通过环境变量检索参数。
- 通过将数据写至 Net.Data 已将其名称指定给环境变量 DTWPIPE 的文件，以使 Perl 脚本将输出参数传送回语言环境。perl 脚本传送至 Net.Data 的数据应具有以下语法:

  ```
  name="value"
  ```

  对于复合数据项，可以用一个新行字符或空字符分开每个项。

  如果一个变量名称与 OUT 或 INOUT 参数的名称相同且使用上述语法，则新的值将替换当前值。如果变量名未对应于 OUT 或 INOUT 参数，Net.Data 将忽略它。

以下示例显示了 Net.Data 如何从一个宏传递变量。

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

如果 Perl 脚本位于一个名为 today.pl 的外部文件中，那么这个函数可以按下一个示例来编写:

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

可以 Net.Data 表传递给由 Perl 语言环境调用的 Perl 脚本。Perl 脚本根据 Net.Data 宏表参数的 Net.Data 名称来访问它们的值。列标题和字段值都包含在用表名和列号标识的变量中。例如，在表 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。表的行数与列数是 myTable_ROWS 和 myTable_COLS。

FUNCTION 块中的 REPORT 和 MESSAGE 块

REPORT 和 MESSAGE 块在所有 FUNCTION 部分都是允许的。它们由 Net.Data 来处理，而不是由语言环境来处理。当然，Perl 脚本可以将文本写至标准输出流，作为 Web 页面的一部分。

Perl 语言环境示例

以下示例显示了 Net.Data 如何通过执行外部的 Perl 脚本来生成一个表:

```
%define {
 c = %TABLE(20)
 rows = "5"
 columns = "5" %}
 %function(DTW_PERL) genTable(in rows, in columns, out table) {
 open(D2W,"> $ENV{DTWPIPE}");
 print "genTable begins ... ";

 $r = $ENV{ROWS};
 $c = $ENV{COLUMNS};
 print D2W "table_ROWS=\"$r\" ";
 print D2W "table_COLS=\"$c\" ";
 print "rows: $r ";

 print "columns: $c";
 for ($j=1; $j<=$c; $j++)
 {
 print D2W "table_N_$j=\"COL$j\" ";
```

```
 }
 for ($i=1; $i<=$r; $i++)
 {
 for ($j=1; $j<=$c; $j++)
 {
 print D2W "table_V_$i","_","$j=\"" $i $j "\" ";
 }
 }
 close(D2W);  %}

 %message{
 default: "genTable: Unexpected Error"
 %}
 %}

%HTML(REPORT){
 @genTable(rows, columns, c)
 return code is $(RETURN_CODE)
 %}
```

结果: genTable 生成:

```
rows: 5 columns: 5
  COL1  |  COL2   |  COL3   |  COL4   |  COL5   |
-------------------------------------------------
[ 1 1 ] | [ 1 2 ] | [ 1 3 ] | [ 1 4 ] | [ 1 5 ] |
-------------------------------------------------
[ 2 1 ] | [ 2 2 ] | [ 2 3 ] | [ 2 4 ] | [ 2 5 ] |
-------------------------------------------------
[ 3 1 ] | [ 3 2 ] | [ 3 3 ] | [ 3 4 ] | [ 3 5 ] |
-------------------------------------------------
[ 4 1 ] | [ 4 2 ] | [ 4 3 ] | [ 4 4 ] | [ 4 5 ] |
-------------------------------------------------
[ 5 1 ] | [ 5 2 ] | [ 5 3 ] | [ 5 4 ] | [ 5 5 ] |
-------------------------------------------------
return code is 0
```

REXX 语言环境

REXX 语言环境允许您运行已编写的要在 DTW_REXX 环境中运行的 REXX 程序。 “Net.Data REXX 语言环境”提供了允许 REXX 程序容易返回大量数据的控件。

Net.Data 还对将输出引导至浏览器的 REXX SAY 语句提供了支持，而不管用于 Net.Data 的是哪种 Web 服务器环境。如果您使用 Web 服务器 FastCGI、GWAPI 或小服务程序配置来运行本机 REXX，则来自 REXX SAY 语句的输出会按路径发送至 Web 服务器日志文件而不是浏览器。对于已编写的要在 DTW_REXX 环境中运行的 REXX 程序来说，不会出现这种情况。

对变量的支持: 为了允许 REXX 程序容易返回大量数据，Net.Data 会自动将代码添加到 REXX 程序的开头，以及将代码追加到 REXX 程序末尾。此代码是用来处理在 DTW_REXX 函数语句上所提供的变量的。

对 REXX SAY 语句（FastCGI、GWAPI 和 SERVLET 环境）的支持: 在执行 REXX 程序之前，REXX SAY 语句会被 Net.Data 自动转换成 REXX 赋值语句。Net.Data 会自动将代码追加到用来将输出从原始的 REXX SAY 语句引导至浏览器的 REXX 程序。使用 REXX 子例程和函数: 由于 Net.Data 将代码添加到 REXX 程序前面，并将代码追加到 REXX 程序的末尾，因此，主要的 REXX 例程必须以 REXX 程序的最后一个语句结束。如果使用 REXX 子例程或函数，则必须确保 REXX 程序的最后一个语句与主要的 REXX 例程相关联。以下是在已编写的要在 DTW_REXX 环境中运行的 REXX 程序中使用子例程和函数的示例:

```
%function(DTW_REXX) genData(out s1,s2) {
         call subrtn1
         s2=funrtn1()
        signal rexxEnd  /* Go to end of Program */
      subrtn1: PROCEDURE EXPOSE s1
         string1 = "data for s1"
         return 0
      funrtn1: PROCEDURE
         retvar = "data for s2"
         return retvar
      rexxEnd:          /* End of Main Program */
         return 0
    %}
    %HTML (Report) {


    @genData(a,c)

  Value for s1: $(a)

  Value for s2: $(c)
   %}
```

使用 REXX EXIT 和 RETURN 语句: Net.Data 自动将代码追加到为输出变量提供值的 REXX 程序中，并将输出从 SAY 语句引导至浏览器。如果 REXX 程序从主例程发出 RETURN 或者从任何地方发出 EXIT 语句，但是不是 REXX 程序的最后一个语句，则将不执行由 Net.Data 追加到 REXX 程序的代码。这将导致丢失 SAY 语句中的输出变量和输出。如果必须在到达最后一个语句之前退出 REXX 程序，则应该分支到正常退出的 REXX 程序中的最后一个语句。如果使用 RETURN 或 EXIT 语句来结束主要的 REXX 程序，则该语句必须是 REXX 程序中的最后一个语句。这包括 REXX 注释语句。例如:

```
%function(DTW_REXX) genData(out s1,s2) {
......
If S2 < 0 Then signal rexxEnd
......
```

```
......
rexxEnd:
/* This comment must be before the following
RETURN statement */
return 0
%}
%HTML (Report) {
@genData(a,c)
......
%}
```

从 DTW_REXX 函数中调用外部的 REXX 程序: 可以使用 Net.Data %EXEC 语句来从 DTW_REXX 函数中调用 REXX 程序，或者使用由 REXX 提供的方法来从 REXX 程序中调用 REXX 程序。

当使用 Net.Data %EXEC 语句来调用外部的 REXX 程序时，Net.Data 会自动将代码添加到 REXX 程序前面，并将代码追加到 REXX 程序的末尾，以便处理 Output 变量并将输出从 REXX SAY 语句引导至浏览器。

当使用由 REXX 提供的方法来调用 REXX 程序时，Net.Data 不会接收控件，并且不会将代码添加至 REXX 程序。被调用的 REXX 程序必须将输出传送回使用标准 REXX 约定的调用 REXX 程序。当在 GWAPI 或 SERVLET 环境中运行时，来自 REXX SAY 语句的输出将被发送到 Web 服务器日志文件中。

配置 REXX 语言环境

要使用 REXX 语言环境，需要验证 Net.Data 的初始化设置，并设置语言环境。

验证初始化文件中有以下配置语句，并且是在一行上:

```
ENVIRONMENT (DTW_REXX)     DTWREXX   ( OUT RETURN_CODE )
```

参见 *Net.Data 管理与编程指南*，以进一步了解 Net.Data 初始化文件和语言环境 ENVIRONMENT 语句。

执行 REXX 程序

利用 REXX 语言环境，可以执行直接插入 REXX 程序或外部 REXX 程序。直接插入 REXX 程序是一种在宏中具有 REXX 程序的源的 REXX 程序。外部 REXX 程序在外部文件中具有 REXX 程序的源。

要执行直接插入 REXX 程序:

定义一个使用 REXX (DTW_REXX) 语言环境的函数，并在该函数的可执行的语言环境部分中包含 REXX 代码。

示例: 包含直接插入 REXX 程序的函数

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

要运行外部 REXX 程序:

定义一个使用 REXX (DTW_REXX) 语言环境的函数，它包含一个路径，该路径指向要在 EXEC 语句中运行的 REXX 程序。

示例: 包含指向外部程序的 EXEC 语句的函数

```
%function(DTW_REXX) externalHelloWorld() {
%EXEC{ helloworld.cmd%}
%}
```

必需条件: 确保 REXX 文件名列示在 Net.Data 初始化文件中为 EXEC_PATH 配置变量指定的路径中。参见第22页的『EXEC_PATH』，以学习如何定义 EXEC_PATH 配置变量。

将参数传送给 REXX 程序

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

直接 使用 %EXEC 语句来将参数直接传送到外部的 REXX 程序。例如:

```
%FUNCTION(DTW_REXX) rexx1() {
  %EXEC{CALL1.CMD $(INPARM) "literal string"   %}
%}
```

Net.Data 变量 INPARM1 被引用并被传送到外部的 REXX 程序。通过使用 REXX PARSE ARG 指令，REXX 程序可以引用变量。使用这种方法来传送给 REXX 程序的参数被认为是输入参数，并且对值所作的任何修改不会反映给 Net.Data（传送给程序的参数可以由程序使用和处理，但对参数的更改不会反映给 Net.Data）。

间接

利用 REXX 程序*变量池*来间接传送参数。当启动了 REXX 程序时，会创建一个包含了关于所有变量的信息的空间，并由 REXX 解释器来维护。此空间被称为变量池。

当调用 REXX 语言环境 (DTW_REXX) 函数时，REXX 语言环境将在执行 REXX 程序之前把任何输入 (IN) 或输入／输出 (INOUT) 的函数参数存储在该空间中。当调用 REXX 程序时，它可以直接访问这些变量。当成功完成 REXX 程序时，DTW_REXX 语言环境就会确定是否有任何输出 (OUT) 或 INOUT 函数参数。如果有，语言环境将从变量池中检索对应于

函数参数的值，并使用新值来更新函数参数值。当 Net.Data 接收控件时，它便使用从 REXX 语言环境中获得的新值来更新所有 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)
%}
```

在以上示例中，调用 *@double_func* 传送了两个参数 *a* 和 *b*。REXX 函数 *double_func* 将第一个参数加倍，然后将结果存储在第二个参数中。当 Net.Data 调用宏时，*b* 的值为 6。

可以将 Net.Data 表传送至 REXX 程序。REXX 程序作为 REXX stem 变量来访问 Net.Data 宏表参数的值。对于 REXX 程序，列标题和字段值都包含在用表名和列号标识的变量中。例如，在表 myTable 中，列标题为 myTable_V.*j*，字段值为 myTable_V.*i*.*j*，其中 *i* 是行号，*j* 是列号。表中的行数为 myTable_*ROWS*，表中的列数为 myTable_*COLS*。

改进 AIX 操作系统的性能:

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

您可以用以下三种方式来设置环境变量:

- 在宏中，通过使用 DTW_SETENV 内置函数:

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

- 在 AIX 系统环境文件中，通过插入下列语句:

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

  此方法将影响整个机器上 REXX 的功能。

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

  ```
  InheritEnv RXQUEUE_OWNER_PID = 0
  ```

  此方法将影响 Web 服务器上 REXX 的功能。

REXX 语言环境示例

以下示例显示了一个宏调用 REXX 函数来生成具有两列、三行的 Net.Data 表。在调用 REXX 函数之后，调用内置函数 DTW_TB_TABLE() 来生成发送回浏览器的 HTML 表。

```
%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:

```
<table>
  <tr>
<th>COL1</th>
<th>COL2</th>
</tr>
<tr>
<td>[1 1]</td>
<td>[1 2],</td>
</tr>
<tr>
<td>[2 1]</td>
<td>[2 2],</td>
</tr>
<tr>
<td>[3 1]</td>
<td>[3 2],</td>
</tr>
</table>
```

System 语言环境

System 语言环境支持执行命令和调用外部程序。

配置 System 语言环境

在初始化文件中添加以下配置语句，并且是在一行上:

```
ENVIRONMENT (DTW_SYSTEM)   DTWSYS    ( OUT RETURN_CODE )
```

参见 *Net.Data 管理与编程指南*，以进一步了解 Net.Data 初始化文件和语言环境 ENVIRONMENT 语句。

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 { ADDLIBLE.CMD %}
 %}
```

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。参见第22页的『EXEC_PATH』，以学习如何定义 EXEC_PATH 配置变量。

示例 1: 调用程序

```
%FUNCTION(DTW_SYSTEM) sys3() {
      %EXEC {MYPGM.EXE %}
 %
```

将参数传送到程序

有两种方式将信息传送到 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送参数。例如:

```
%DEFINE INPARM1 = "SWITCH1"

%FUNCTION(DTW_SYSTEM) sys1() {
 %EXEC{
  CALL1.CMD  $(INPARM1) "literal string"
 %}
%}
```

Net.Data 变量 INPARM1 被引用并被传送到程序。参数传送到程序的方式与从命令行调用程序时参数传送到程序的方式相同。使用此方法传送给程序的参数被认为是输入参数，并且对值所作的任何修改都不会反映给 Net.Data（传送给程序的参数可以由程序使用和处理，但对参数的更改不会反映给 Net.Data)。

间接 System 语言和不能直接传送或检索 Net.Data 变量，因此这些变量以如下方式使程序可以使用它们:

- Net.Data 将输入参数作为环境变量传送到程序。然后，程序就可以通过环境变量检索参数。
- 程序通过将结果写入一个已命名的管道来将输出参数传回语言环境 (Net.Data 在环境变量 DTWPIPE 中传送该管道的名称)。使用以下语法将数据写至已命名的管道:

  name="value"

  对于复合数据项，可以用一个新行字符或空字符分开每个项。

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

以下示例显示了 Net.Data 如何从一个宏传送变量。

```
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
  %EXEC {
     UPDPGM
  %}
%}
```

您可以通过 System 语言环境将 Net.Data 表传送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 宏表参数的值。列标题和字段值都包含在用表名和列号标识的变量中。例如，在表 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。表的行数与列数是 myTable_ROWS 和 myTable_COLS。

System 语言环境示例

以下示例显示了一个包含函数定义的宏，其中有三个参数 P1、P2 和 P3。P1 是一个输入 (IN) 参数，P2 和 P3 是输出 (OUT) 参数。函数调用程序 UPDPGM，它用 P1 的值更新参数 P2，并将 P3 设置为字符串。在处理 %EXEC 块中的语句之前，DTW_SYSTEM 语言环境在环境空间中存储 P1 和相应的值。

```
%DEFINE{
   MYPARM2  = "ValueOfParm2"
   MYPARM3  = "ValueOfParm3"
%}
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
  %EXEC {
     UPDPGM
  %}
%}

%HTML(upd1) {
<p>
Passing data to a program.  The current value
```

```
 of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is
 "$(MYPARM3)".  Now we invoke the Web macro function.
<p>
 @sys1("ValueOfParm1", MYPARM2, MYPARM3)

<p>
 After the function call, the value of MYPARM2 is "$(MYPARM2)",
 and the value of MYPARM3 is "$(MYPARM3)".
<p>
 %}
```

第7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。将涉及以下主题:

- 使用 Web 服务器 API
- 第168页的『管理连接』
- 第172页的『Net.Data 高速缓存』
- 第193页的『设置出错日志的级别』
- 第194页的『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有多快，Web 服务器的性能对响应时间都有直接影响。

使用 Web 服务器 API

您可以用一个 Web 服务器 API（例如 APAPI），而不是 CGI 来调用 Net.Data。当使用一个 Web 服务器 API 执行 Net.Data 时，它将作为 Web 服务器进程中的一个线程来执行。Web 服务器的进程是多线程的，它创建了一个可处理多个 Net.Data 请求的环境。这些请求是在同一地址空间内并行处理的，从而避免了将 Net.Data 作为 CGI 进程调用的开销。

注意事项: 使用 Web 服务器 API 使得性能有所改进，而且没有隔离应用程序。因为 Net.Data 是在多线程环境中运行的，所以 Web 服务器发生的问题有可能会使其关闭。例如，用户编写的语言环境错误，不正确的调用，甚至是数据库故障。在决定是否使用某个 Web 服务器 API 时，应确定应用程序性能和应用程序隔离哪一个具有较高的优先级。

使用 FastCGI

FastCGI 提供了改进的性能和 CGI 的应用程序隔离。您可以在所有支持 FastCGI 的 Web 服务器上同时使用 Net.Data 和 FastCGI。有关配置 FastCGI 的更多信息，参见第168页的『管理连接』。

可调整 FastCGI 来运行适量的进程，以便处理一定数量的入局请求。例如，如果要每秒处理 100 个请求，每个请求花半秒时间进行处理，则应在 FastCGI 配置文件中将 NumProcessess 伪指令设置为 50。

FastCGI 在所有 LE 中都受支持；但是 Oracle 和 ODBC 需要“现场连接”。

要调节同步进程的个数:

1. 打开定义进程配置参数的配置文件。

   对于 Apache 和 IBM HTTP，使用 `httpd.conf` 文件。

2. 更改用来指定进程个数的配置参数值:
   - 对于 Apache: `Process=`*num*。
   - 对于 ISC: `NumProcess=`*num*。

其中 *num* 是进程的个数。

管理连接

Net.Data 提供了一个称为“现场连接”的组件来管理数据库和 Java 虚拟机连接。“现场连接”维护持久连接，以改进性能。有些 Net.Data 的操作需要很长的启动时间。例如，一个进程在发出数据库查询之前，必须先向 DBMS 标识它自己并连接到数据库。这段时间通常在访问数据库的 Net.Data 宏所需的进程时间中占很大部分。由于 CGI 程序操作的方式，在每次请求 Web 服务器时都需要耗费这些启动时间。Net.Data 对 OS/2、Windows NT、AIX、Solaris 和 Linux 操作系统提供了“现场连接”并维护持久连接。

以下章节描述了“现场连接”。

- 『关于“现场连接”』
- 第169页的『“现场连接”的优点』
- 第170页的『我应当使用“现场连接”吗？』
- 第170页的『启动“连接管理器”』
- 第171页的『Net.Data 和“现场连接”进程流』

关于“现场连接”

“现场连接”通过避免启动时的系统开销极大地改进了性能。这些节省来自于连续运行一个或多个执行启动功能的进程。然后，这些进程将等待服务请求。如果将 Net.Data 用作 CGI 或 FastCGI 程序，或者用作 Web 服务器 API 插件，那么您可以运行“现场连接”。

“现场连接”由“连接管理器”和 cliette 组成。*Cliette* 是“连接管理器”所启动的一些进程，它们在服务器运行期间保持活动状态。Cliette 处理数据并与您在初始化文件中用 CLIETTE 关键字指定的 Net.Data 语言环境通信。每种类型的 cliette 处理一个特定的语言环境函数，例如 DB2 cliette，它连接到 DB2 数据库并建立在 Net.Data 处理任何 Net.Data 宏之前执行 SQL 调用的操作。这个可执行文件是在

“现场连接”配置文件中命名的，称为 dtwcm.cnf。图25显示了“现场连接”、宏以及语言环境之间的交互作用。

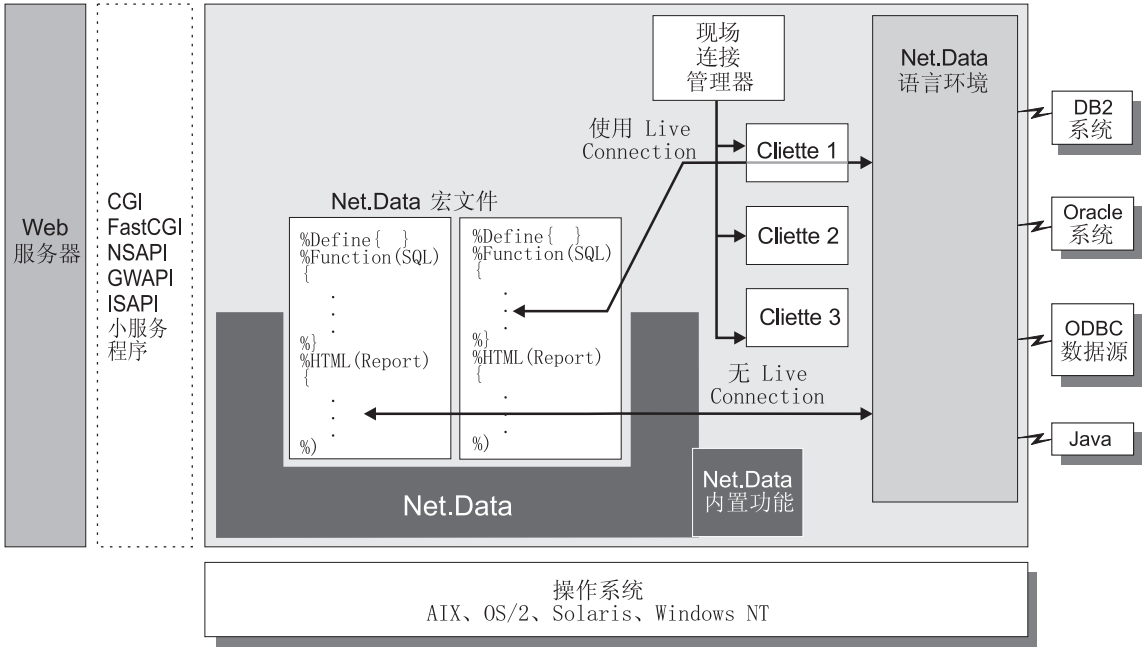


图 25. 使用 Cliette 的“现场连接”

以下章节更详细地描述了“现场连接”。要学习如何配置“现场连接”，参见第32页的『配置“现场连接”』。

“现场连接”的优点

使用“现场连接”的主要优点是:

- **改进性能**

  重新使用连接要比建立新的连接有效得多。通常，如果您请求小的 SQL 语句（例如，对少于 100000 行的数据库的简单查询），或者如果您的数据库连接很困难（例如，远程服务器），那么连接的时间会占很大比重。

- **多数据库访问**

  “现场连接”允许一个 Net.Data 宏同时连接到多个数据库。这一点是可能的，因为每个数据库都有唯一的 cliette，因此 Net.Data 只需和多个 cliette 通信即可。

我应当使用“现场连接”吗?

可通过 CGI、FastCGI 或 API 方式使用“现场连接”，以与数据库或“Java 虚拟机”通信。另外，如果应用程序需要来自多个数据库的数据，您还可以从“现场连接”受益。

“现场连接”和 API 插件一起使用，可以改进许多系统的性能（取决于系统的负载和配置）。您应当在自己的系统上试验一下，确定哪一个配置是最佳的。

使用 ACTIVATE DATABASE 命令可节省建立数据库连接的时间，因此许多应用程序不必使用“现场连接”就可改进性能。参见数据库的文档，以获取有关您的数据库所使用的命令的详细信息。还请检查一下您的操作系统文档，查看是否有其他能够帮助改进性能的步骤。

要求: 在 FastCGI 和 API 方式中运行时，ODBC 和 Oracle 语言环境需要“现场连接”。

启动“连接管理器”

“连接管理器”是与 Net.Data 一起交付的一个独立的可执行文件，名为 `dtwcm`。在启动 Web 服务器时启动“连接管理器”。

启动“连接管理器”时，它将读取配置文件并启动一组进程。在每个进程中，“连接管理器”都开始执行一个特定的 cliette。要学习如何配置“现场连接”，参见第32页的『配置“现场连接”』。

要在 Windows NT 和 OS/2 中启动“连接管理器”:

1. 从命令行中，切换到 <*inst_dir*>\connect\ 目录。
2. 输入 `dtwcm`。

其中 <*inst_dir*> 是 Net.Data 的安装目录。

要在 AIX 中启动“连接管理器”:

1. 从命令行中，切换到 `/usr/lpp/internet/db2www/db2/` 目录。
2. 输入 `dtwcm`。

要启动带消息选项的“连接管理器”:

缺省情况下，“连接管理器”的消息是关闭的。如果您希望显示“连接管理器”的消息，可以在启动“连接管理器”时使用 -d 选项。

从命令行输入: `dtwcm -d`

使用 -d 选项之后，要想再次关闭消息，就必须重新启动“连接管理器”。

要自动将“连接管理器”作为 Windows NT 服务启动:

在 Windows NT 上，您可以指定将“连接管理器”作为 Windows NT 的服务启动，而不是从命令行启动。将“连接管理器”作为 Windows NT 服务运行可以使“连接管理器”在每次启动机器时自动启动。

提示: 在将“连接管理器”设置为自动启动之前，先从命令行启动，以确保“现场连接”配置文件是正确的。

1. 从 Windows NT 任务栏，选择**开始->设置->控制面板->服务**。
2. 选择 **Net.Data 现场连接**，然后单击**启动**按钮。
3. 选择**自动启动类型**，然后单击**确定**。

注: 缺省情况下，记录是关闭的。要打开记录，从“服务”窗口选择“Net.Data 现场连接”，并在“启动参数”框中输入 -l。

Net.Data 和“现场连接”进程流

在配置并启动了数据库、Web 服务器以及“连接管理器”之后，Net.Data 处理一般将在启用了“现场连接”的情况下涉及以下步骤:

1. Web 服务器接收请求并启动一个 FastCGI、CGI 或 API 进程来运行 Net.Data。
2. Net.Data 开始处理 Net.Data 宏。
3. 当 Net.Data 遇到使用“现场连接”的函数调用时，它将确定需要初始化文件中哪些类型的 cliette。对于 DB2，cliette 类型通常是根据 DB2 数据库名所取的一个名称，例如 DTW_SQL:CELDIAL。
4. Net.Data 向“连接管理器”要求该类型的 cliette。
5. “连接管理器”查找可用的该类 cliette。如果没有可用的 cliette，“连接管理器”将把请求放入队列，当有正确的 cliette 类型可用时再进行处理。
6. 当有一个 cliette 可用时，“连接管理器”将告诉 Net.Data 如何与该 cliette 通信。
7. Net.Data 要求 cliette 处理函数。
8. 处理从步骤 3 开始重复，直至 Net.Data 宏处理完成为止。
9. 释放所有的 cliette。

如果初始化文件中指定了一个 cliette，但“连接管理器”没有运行，Net.Data 将装入 DLL 并处理该宏。如果使用 API，那么很可能会接收到错误，这时就应启动“连接管理器”。

Net.Data 高速缓存

高速缓存有助于您提高应用程序用户的响应时间。Net.Data 将来自对 Web 服务器的请求存储在本地以备快速检索，直到刷新信息时为止。本章描述 Net.Data 高速缓存的概念、任务和限制。

- 『关于 Web 页面高速缓存』
- 第173页的『关于 Net.Data 高速缓存』
- 第173页的『Net.Data 高速缓存术语』
- 第174页的『Net.Data 高速缓存概念』
- 第175页的『Net.Data 高速缓存的限制』
- 第175页的『Net.Data 高速缓存接口』
- 第176页的『高速缓存管理器的规划』
- 第177页的『高速缓存标识符』
- 第177页的『配置高速缓存管理器和 Net.Data 高速缓存』
- 第185页的『启动和停止高速缓存管理器』
- 第186页的『高速缓存 Web 页』
- 第189页的『CACHEADM 命令』
- 第191页的『高速缓存日志』

关于 Web 页面高速缓存

许多软件组件都为 Web 应用程序执行高速缓存。这里是高速缓存应用程序的一些示例:

- Web 浏览器在内存或磁盘中本地保存 Web 页面和相关对象（例如图象和音频文件以及 Java 小程序），当在用户重复访问相同页面时节省网络时间。
- Web 代理服务器高速缓存在靠近一组用户的本地服务器上保存 Web 页面和相关对象，以减少对远程 Web 服务器的网络访问时间，例如减少 Web 服务器检索所请求的项目的次数。Web 代理服务器高速缓存还允许多个用户之间有效共享公共访问页面。
- Web 服务器在内存中高速缓存频繁检索的页和相关对象，以在用户重复检索相同页时节省磁盘存取时间。
- 数据库管理系统在内存中高速缓存通常保留在磁盘上的数据项，以在重复检索相同数据项时节省磁盘访问时间。

所有这些组件都各自完成它们的高速缓存过程，但是整个结果为用户改进了响应时间。为了确定何时刷新高速缓存的项目，Web 组件（浏览器、代理服务器和 Web 服务器）通常考虑以下不同选项，包括:

- 浏览器和服务器配置选项
- 从 Web 服务器中与 Web 页面和相关项目一起返回的 HTTP 题头的内容，特别是到期日期信息

关于 Net.Data 高速缓存

Net.Data 本身为 Net.Data 宏生成的频繁访问页面和相关数据项提供自己的高速缓存功能。通过从 Net.Data 高速缓存中传送页面，可以节省为了创建页面而运行 Net.Data 宏和访问数据库的时间。

对于每个服务器，可以使用一个高速缓存管理器。**建议**： 为 Net.Data 的许多实例使用一个高速缓存管理器，每个高速缓存管理器对应多个高速缓存。

图26显示 Net.Data 使用高速缓存管理器来管理来自一个宏的 HTML 输出的高速缓存过程。此输出可能包含来自数据库的数据。

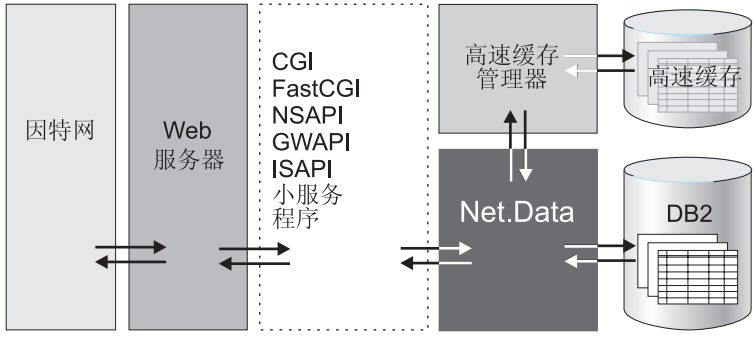


图 26. Net.Data 高速缓存

Net.Data 高速缓存术语

Net.Data 文档使用以下项目来描述 Net.Data 高速缓存。

高速缓存

一类包含最近访问过的数据的内存，是为了加快对相同数据的后继访问而设计的。高速缓存经常是用来对网络中可以访问的、频繁使用的数据保留一个本地副本。在 Net.Data 中，指这样的本地内存：它包含 Net.Data 所生成的 HTML Web 页面，以让 Net.Data 宏重新使用。在高速缓存中存储了页之后，Net.Data 就不必重新生成高速缓存中的信息。每个高速缓存都是由高速缓存管理器来管理的，高速缓存管理器负责管理多个高速缓存，并可以服务于 Net.Data 的多个实例。

高速缓存标识符

一个标识特定高速缓存的字符串。

高速缓存管理器

为一台机器管理高速缓存的程序。它可以管理多个高速缓存。

高速缓存管理器配置文件

此文件包含一些设置，Net.Data 使用这些设置来确定对记录、跟踪、高速缓存大小和其他选项的设置。它包含对高速缓存管理器和特定高速缓存管理器所管理的所有高速缓存文件的设置。在 Net.Data 中封装时，文件名是 `cachemgr.cnf`。

Net.Data 高速缓存概念

根据系统上具有多少 HTTP 服务器以及每个 HTTP 服务器是否运行 Net.Data 的自身副本（使用各自的 Net.Data 配置文件），您可以使 Net.Data 的所有副本与一个或多个高速缓存管理器相关联。一个高速缓存管理器可以支持许多内存中的高速缓存，每个高速缓存具有一个*高速缓存标识符*。图27显示一个高速缓存管理器，它处理多个宏并管理两个高速缓存。

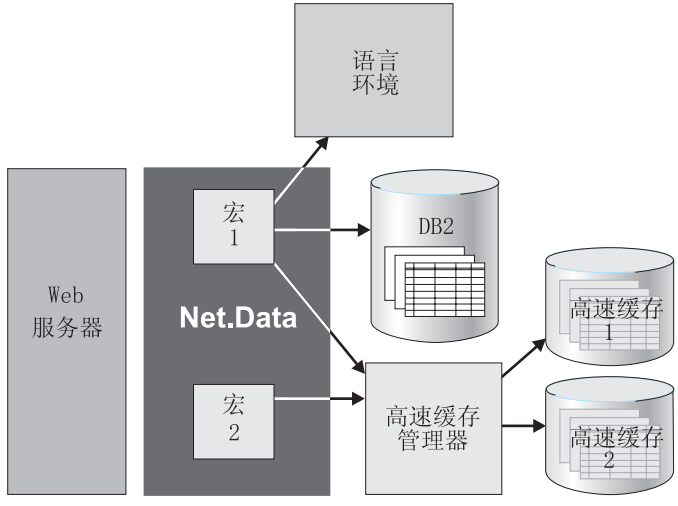


图 27. 高速缓存管理器处理多个宏和高速缓存

任何数目的项目（称为*高速缓存的页*）都可以放在一个高速缓存中。每个高速缓存的页具有唯一的标识符，例如一个统一资源定位器 (URL)。一个页面是一个完整的 HTML 页面或它的一个分段。

当 Net.Data 接收一个高速缓存化的数据的请求（例如，来自内置函数 DTW_CACHE_PAGE）时，将发生以下步骤:

1. Net.Data 连接至高速缓存管理器。
2. Net.Data 检查数据是否已经高速缓存。
   - 如果数据已高速缓存并且未到期，Net.Data 将从高速缓存管理器请求页面，将它发送到浏览器并停止执行宏。
   - 如果数据未高速缓存，Net.Data 将继续处理宏，然后将生成的 HTML 页发送到 Web 浏览器和高速缓存管理器，在高速缓存管理器中进行高速缓存。
3. Net.Data 断开与高速缓存管理器的连接

当宏成功完成处理之后，高速缓存管理器高速缓存 HTML 输出，确保只高速缓存成功生成的 Web 页。数据直到发送到浏览器才被高速缓存，用户看到的数据与高速缓存的数据相同。

当 Net.Data 遇到一个错误并从宏中退出时，高速缓存管理器:

- 不接受部分或有故障的页
- 在高速缓存中保存现存页

Net.Data 高速缓存的限制

Net.Data 高速缓存具有以下限制:

安全性 高速缓存管理器不提供安全性。例如，一个数据库用户运行某个宏且将高速缓存某数据库结果页。而另一个数据库用户可以检索该高速缓存的页。

直接请求

Net.Data 的直接请求调用不能使用 Net.Data 高速缓存。

Net.Data 高速缓存接口

Net.Data 提供了一组灵活的接口，供您在为应用程序配置和设置高速缓存时使用。表14描述了使用 Net.Data 高速缓存功能的各种选项以及这些选项描述的地方。

表 14. Net.Data 高速缓存接口

接口	描述	转至 ...
高速缓存管理器配置选项	您可以在高速缓存管理器配置文件的高速缓存管理器节中为高速缓存管理器指定许多选项，例如记录和跟踪。	第177页的『定义高速缓存管理器』

表 14. Net.Data 高速缓存接口 (续)

接口	描述	转至 ...
高速缓存配置选项	在 Net.Data 的高速缓存管理器的单个实例中，您可以定义许多高速缓存来保存高速缓存项。每个高速缓存具有自己的特性集（例如大小和位置）和高速缓存标识符。在高速缓存管理器配置文件中的高速缓存节中定义了这些特性。每个节是由高速缓存标识符来标识的。	第180页的『定义高速缓存』
Net.Data 初始化选项	如果 Net.Data 和相应的高速缓存管理器在各自系统上运行，则必须在 Net.Data 初始化文件中指定高速缓存管理器系统和端口号。	第14页的『高速缓存管理器配置变量』
Net.Data 高速缓存内置函数	可以使用 Net.Data 内置函数来处理 Net.Data 高速缓存的内容。在适当的宏函数中指定高速缓存标识符来选取具有最合适特性的高速缓存。	参见*Net.Data 参考*的内置函数一章

高速缓存管理器的规划

在规划 Net.Data 高速缓存功能的使用时，必须考虑:

- 高速缓存什么页有好处，以及将获取的性能改进
- 何时高速缓存项目
- 何时刷新高速缓存中的项目，以及要使用的刷新方式

要使用 Net.Data 高速缓存，必须完成以下步骤，即需要了解您希望如何使用高速缓存。

建议: 在着手于使用高速缓存的较大应用程序之前，必须先规划和设计应用程序的原型，然后才将它投入生产。

- 安装包含了高速缓存功能的 Net.Data。
- 配置高速缓存管理器。参见第177页的『配置高速缓存管理器和 Net.Data 高速缓存』。
- 确定您将如何把 Net.Data 应用程序投入生产。

- 检查各种 Net.Data 高速缓存日志，以确定是否应当改进高速缓存的使用和配置的方式。

高速缓存错误

当 Net.Data 遇到内部错误，使它在完成处理之前退出宏，则高速缓存管理器不高速缓存 Web 页。高速缓存管理器不高速缓存不完整的或包含 Net.Data 错误的页面。这些错误类型包含宏语法错误和 SQL 错误。

有错误的页在以下条件下可被高速缓存:

- Net.Data 遇到一个错误而 Net.Data 由于在信息块中有 CONINUE 配置变量而继续执行宏并正常终止。
- 错误发生在 Net.Data 的错误确定作用域之外，例如数据库滚回。

高速缓存标识符

在设计应用程序的高速缓存时，需要规划两种类型的标识符。

- **高速缓存的标识符**: 此标识符是高速缓存标识符，并在定义高速缓存的配置文件节中指定名称。可以使用许多方法来分类和命名高速缓存。例如，通过应用程序来命名高速缓存。对于每个 Net.Data 应用程序，都可以具有一个高速缓存，给予每个高速缓存一个名称，该名称推导自它服务的 Net.Data 宏。
- **高速缓存的页面的标识符**: 此标识符是高速缓存的页的标识符，并指定要高速缓存的页面的名称。高速缓存的页的标识符可以是任何字符串，例如 URl 地址。用 DTW_CACHE_PAGE() 内置函数可以指定标识符。参见 *Net.Data 参考*的内置函数一章以获取语法和示例。

配置高速缓存管理器和 Net.Data 高速缓存

高速缓存管理器管理系统中一个或多个高速缓存。这些高速缓存中的每一个都包含动态生成的 HTML 页的内容。要配置高速缓存管理器和每个高速缓存，必须更新高速缓存管理器配置文件 `cachemgr.cnf` 中的关键字值。

高速缓存管理器配置文件中包含两种类型的节: 高速缓存管理器节和高速缓存定义节。下列步骤描述如何为应用程序定制这两个类型的节。

定义高速缓存管理器

通过指定允许的关键字的值，来定义高速缓存管理器节。所有关键字都是可选的; 除非您不希望接受缺省值，否则不必指定它们。

要定义高速缓存管理器:

1. 指定高速缓存管理器日志文件的名称。日志显示所有高速缓存的所有事务的动作，并供调试和问题分析来使用。

   缺省情况是将信息写至控制台。

语法:

```
log=path
```

其中，*path* 是高速缓存文件的路径与文件名称。

提示: 要指定每个高速缓存的日志文件，可在高速缓存定义节中使用 **tran-log** 关键字。

2. 指定高速缓存管理器用于接受请求的 TCP/IP 端口号。只对于从远程机上联系高速缓存管理器时，才使用该端口号码。

   此值必须与 DTW_CACHE_PORT 配置变量在 Net.Data 初始化文件中指定的端口号匹配。用以下方法确定缺省值:

   a. 高速缓存管理器在路径 */etc/services* 中检查与名称 `ibm-cachemgrd` 关联的值。如果找到此值，高速缓存管理器将使用此值。如果找不到，则使用下一种方法。

   b. 高速缓存管理器使用缺省端口 7175。

   语法:

   ```
   port=port_number
   ```

   其中 *port_number* 是唯一的 TCP/IP 端口号。

3. 以秒为单位指定一个最大时间长度，在此时间内，高速缓存管理器应当允许暂挂的读取操作保持活动。如果超过此时间，高速缓存管理器删除连接。

   缺省值是 30 秒。

   语法:

   ```
   connection-timeout=seconds
   ```

   其中 *seconds* 是秒数，用于暂挂的读取操作应当活动的时间长度。

4. 指定是否记录信息。

   缺省值是 **no** 或 **off**。

   语法:

   ```
   logging=yes|on|no|off
   ```

   其中:

   yes|on
   指示需要进行记录。

   no|off 指出不应执行记录。

5. 指定是否环绕日志。

缺省值是 **no**。如果指定为 **yes**，则在达到最大大小时当前日志将关闭（参见下面的 **log-size**），文件具有文件类型 .old，新的日志被打开。只维护一代日志文件（现存的 .old 文件被覆盖）。

语法:

```
wrap-log=yes|no
```

其中:

yes 指定环绕日志。

no 指定不环绕日志。

6. 指定最大大小，日志最多可为此大小（如果指定了环绕日志的话）。

   缺省值是 64000。

   语法:

   ```
   log-size=bytes
   ```

   其中 *bytes* 是最大大小的字节数。

7. 指定要写入日志的信息级别。当这些值包含在 *trace_flag_definitions* 列表中时，则已被设置，不再需要进行任何设置。

   缺省值是只记录高速缓存管理器启动和关闭信息。

   语法:

   ```
   trace-flags=trace_flag_definitions
   ```

   其中:

   D_ALL
   启用所有跟踪标志。

   D_NONE
   禁用所有跟踪标志

示例: 启用指定所有跟踪标志的跟踪标志:

```
trace=flags=D_ALL
```

节的示例: 一个有效的配置管理器节:

```
cache-manager {
port = 7175
connection-timeout = 60
logging = off
log = /local/netdata/cachemgr/logs/cachemgr.log
wrap-log = yes
log-size = 32KB
}
```

定义高速缓存

通过指定允许的关键字的值，来定义“高速缓存定义”节。大部分关键字都是可选的，除非您不希望使用缺省值，否则不必指定它。

要定义一个高速缓存:

1. 指定要保持高速缓存页的路径和目录名。就启动来说，包含此目录的文件系统必须至少象 **fssize**（参见下面）的值一样大；否则高速缓存将不能启动。可将此值指定为一个绝对路径名或对应于高速缓存管理器启动的路径的相对路径名。

   必需的。

   语法:

   ```
   root=path_name
   ```

   其中:

   path_name
   是高速缓存页所存储的绝对或相对的目录和路径名。

2. 指定在高速缓存管理器启动时是否激活当前高速缓存。

   不是必需的；缺省值是 **yes**。如果设置为 **no**，则高速缓存被定义在高速缓存管理器中但是不激活。以后您可以用 **cacheadm** 命令来激活它。

   语法:

   ```
   caching=yes|no
   ```

   其中:

   yes 指出当高速缓存管理器启动时将激活高速缓存。

   no 指出当高速缓存管理器启动时将不激活高速缓存。

3. 指定由当前高速缓存中的页在文件系统中所使用的最大空间。超过最大空间数时，高速缓存管理器从最旧的页开始删除足够的页，以将高速缓存所占有的总空间限制在一个极限之内。您可以通过将该值设置为一个很大的数目来有效禁用对条目的自动清除；但是如果超过物理文件系统空间，则试图将新的页面添加到高速缓存将会失败。

   不是必需的；缺省值是 0（没有高速缓存至磁盘）。

   语法:

   ```
   fssize=nnB|nnKB|nnM
   ```

   其中:

   ***nn*B** 是字节数；例如 5000B。

*nn***KB** 是千字节数；例如 640KB。

*nn***MB** 是兆字节数；例如 30MB。

4. 指定由该高速缓存中的所有页面使用的最大内存数。超过最大内存数时，高速缓存管理器从最旧的页开始删除足够的页，以将高速缓存所占有的总内存限制在一个范围之内。您可以通过将该值设置为一个很大的数目来有效禁用对页面的自动清除；但是如果 **cachemgrd** 过程消耗太多内存，操作系统可能会终止它。

   不是必需的；缺省值是 1MB。

   语法:

   ```
   mem-size=nnB|nnKB|nnMB
   ```

   其中:

   *nn***B** 是字节数；例如 5000B。

   *nn***KB** 是千字节数；例如 640KB。

   *nn***MB** 是兆字节数；例如 30MB。

5. 指定了一个页面可在高速缓存中保持的最大时间长度。超过此值时，高速缓存管理器将页面标记为期满，但是除非到达 **fssize**（如果高速缓存在磁盘上）或 **memsize**（如果高速缓存在内存中）极限，否则不删除该页面。如果到达 **memsize** 或 **fssize** 极限，则高速缓存管理器在所有其他页面之前删除标记为期满的页。可以使用 **check_expiration** 关键字来禁用 **lifetime** 检查。

   必需的: 否；缺省值是 5 分钟。

   语法:

   ```
   lifetime=time_length
   ```

   其中:

   *nn***S** 秒数；例如 600S。

   *nn***M** 分钟数；例如 20M。

   *nn***H** 小时数；例如 30H。

6. 指定是否将高速缓存页标记为期满并执行寿命检查。

   不是必需的；缺省值是 **yes**，缺省寿命长度是 60 秒。还可将此值设置为一个时间长度，以指出 **yes** 值并说明一个项目可在高速缓存中保持的最大时间长度。设置为 **no** 时，高速缓存页不标记为期满，并且不执行寿命检查。

   语法:

   ```
   check-expiration=yes|nnS|nnM|nnH|no
   ```

其中:

yes 指出高速缓存管理器执行寿命检查，并且高速缓存页标记为期满。

*nn***S** 秒数；例如 600S。

*nn***M** 分钟数；例如 20M。

*nn***H** 小时数；例如 30H。

no 指出高速缓存管理器不执行寿命检查，并且高速缓存页不标记为期满。

7. 指定一个高速缓存的页面可在内存高速缓存中占用的最大空间量。如果对于内存来说页面太大，则选择文件高速缓存。如果存在足够的空间，高速缓存管理器将在文件高速缓存中擦除高速缓存页面。如果页面不匹配文件高速缓存，则高速缓存的试图将失败。如果页面小于 datum_memory_limit 值 (cacheobj-memory-limit)，但是如果高速缓存没有足够的空间，则将从内存高速缓存中删除最早的高速缓存页面以容纳新的页面。

   不是必需的；缺省值是 1KB。

   语法:

   ```
   datum-memory-limit (cacheobj-memory-limit)=nnB|nnKB|nnMB
   ```

   其中:

   *nn***B** 是字节数；例如 5000B。

   *nn***KB** 是千字节数；例如 640KB。

   *nn***MB** 是兆字节数；例如 30MB。

8. 指定一个高速缓存页面可在文件高速缓存中占用的最大空间量。如果一个页面小于 datum_disk_limit，但是在文件高速缓存中没有余留空间，则将从文件高速缓存中删除最早的高速缓存页面以容纳新的页面。

   不是必需的；缺省值是 1KB。

   语法:

   ```
   datum-disk-limit (cacheobj-space-limit)=nnB|nnKB|nnMB
   ```

   其中:

   *nn***B** 是字节数；例如 5000B。

   *nn***KB** 是千字节数；例如 640KB。

   *nn***MB** 是兆字节数；例如 30MB。

9. 指定了创建统计记录的时间。如果设置为 0，则不写下统计记录。

   不是必需的；缺省值是 0（无任何统计信息）。

语法:

```
stat-interval = nnS|nnM|nnH
```

其中:

***nn*S** 秒数; 例如 600S。

***nn*M** 是分钟数; 例如 1M。

***nn*H** 是小时数; 例如 3H。

10. 指定一个文件的路径和名称, 该文件用于记录当前高速缓存的统计值。

    当 **stat-interval** 的值大于 0 时是必需的。

    语法:

    ```
    stat-files=filename
    ```

    其中 *filename* 是记录统计文件的路径和名称。

11. 指定每次写日志文件时, 统计计数器是否应当复位。

    不是必需的; 缺省值是 **yes**。

    语法:

    ```
    reset-stat-counters=yes|no
    ```

    其中:

    yes 复位统计计数器。

    no 不复位统计计数器。

12. 定义存放每个高速缓存的事务日志的路径与文件名称。高速缓存事务日志文件独立与高速缓存管理器日志文件, 后者用于记录整个高速缓存管理器的活动。

    必需的; 如果不指定, 就不创建高速缓存的事务日志。

    语法:

    ```
    tran-log=filename
    ```

    其中 *filename* 是每个高速缓存的事务日志的路径和文件名。

13. 指定是否在高速缓存管理器首次启动时打开高速缓存的事务日志。除非已通过 tran-log 参数指定了一个有效事务日志文件, 否则忽略此参数。如果在高速缓存管理器配置文件中已经指定 tran-log 值, 则当高速缓存管理器守护程序正在运行时, 可使用 **cacheadm** 命令来激活事务日志。

    不是必需的; 缺省值是 **no**。

    语法:

    ```
    tran-logging=yes|on|no|off
    ```

其中:

yes|on
指示需要进行记录。

no|off 指出不应执行记录。

14. 指定是否应当环绕事务登记。

不是必需的; 缺省值是 **yes**。如果指定为 **yes**, 则在达到最大大小时当前日志将关闭（参见下面的 **tran-log-size**）, 具有文件类型 .old, 新的日志被打开。只维护一代日志（现存的 .old 文件被覆盖）。

语法:

```
wrap-tran-log=yes|no
```

其中:

yes 指定环绕日志。

no 指出不环绕日志。

15. 以字节为单位指定最大大小, 如果指定了 **wrap-tran-log**, 则允许事物日志多至这般大小。

不是必需的; 缺省值是 64000。

语法:

```
tran-log-size=bytes
```

其中 *bytes* 是最大大小的字节数。

节的示例: 高速缓存的一个有效的高速缓存定义节:

```
cache0
{
root = /locale/netdata/cachemgr/caches/cache0
caching = on
mem-size = 10MB
fs-size = 1MB
datum-memory-limit = 200KB
datum-disk-limit = 1MB
lifetime = 6000000
check-expiration = 999999
tran-logging = no
tran-log-size = 10000
wrap-tran-log = yes
tran-log = /ocale/netdata/cachemgr/logs/tran.log
}
```

启动和停止高速缓存管理器

以下章节描述了如何启动和停止高速缓存管理器。

- 『启动高速缓存管理器』
- 『停止高速缓存管理器』

启动高速缓存管理器

使用 cachemgrd 命令来启动高速缓存管理器守护程序。

语法:

参数:

cachemgrd

命令关键字。

config_file

指定一个文件名称，在该文件中定义高速缓存管理器以及高速缓存管理器所管理的每个高速缓存。Net.Data 装运的配置文件是 cachemgr.cnf。

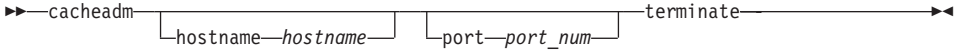
示例:

```
cachemgrd -c myconfig.cfg
```

停止高速缓存管理器

使用 cacheadm 来停止高速缓存管理器。

语法:



参数:

cacheadm

命令关键字。

hostname

指定高速缓存所运行的机器的名称，如果该机器不同于发出 cacheadm 命令的机器的话。

port_num

指定高速缓存端口号，如果该号码不同于缺省值 (7175) 的话。

terminate

指定要停止高速缓存管理器。

示例:

```
cacheadm hostname host1 port 7178 terminate
```

高速缓存 Web 页

利用使用 DTW_CACHE_PAGE 内置函数来高速缓存一个 Web 页。当 Net.Data 在宏中发现 DTW_CACHE_PAGE 函数时，它与高速缓存管理器联系并开始在内存中保存宏的 HTML 输出。在 Net.Data 成功地处理了一个宏之后，HTML 输出就发送给浏览器，并且高速缓存管理器如图28中所示地在一个事物中将输出进行高速缓存。

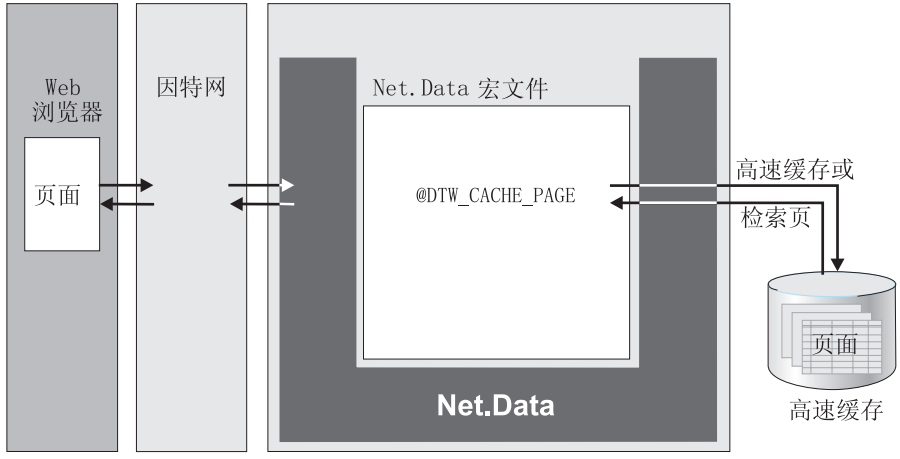


图 28. DTW_CACHE_PAGE 函数初启高速缓存

高速缓存一个页面

使用 Net.Data DTW_CACHE_PAGE() 内置函数来指定要写至高速缓存的 Net.Data 生成的页面。

一旦确定在高速缓存中已经不存在页面或页面已经过期，DTW_CACHE_PAGE() 函数将高速缓存函数语句之后的宏的所有输出。如果页面不存在于高速缓存中或超过指定的年龄，Net.Data 将把输出页面发送回浏览器，从宏执行中生成新的输出页面，并将页面存储在高速缓存中。

如果高速缓存管理器找到高速缓存页面并且该页面仍然是当前的，则它显示高速缓存的内容，并且 Net.Data 退出宏。此行为保证了在从高速缓存中检索了 Web 页面之后，不再作不需要的处理。

性能提示: 把 DTW_CACHE_PAGE() 放在最先，或作为宏中的第一条语句，以将执行宏的代价降到最低。

要高速缓存一个页面:

1. 在宏的 HTML 块或 XML 块中，在进行 HTML 编码之前，插入以下函数语句:

   ```
   @DTW_CACHE_PAGE("cache_id", cached_page_id, "age", status)
   ```

   使用该函数来指出 Net.Data 将对跟随此语句之后的宏中的所有 HTML 输出进行高速缓存。如果您希望高速缓存所有 HTML 输出，则将该语句放在宏的最前面。

   参数:

   cache_id
   标志放置此页的高速缓存的字符串。您可以将高速缓存标识符与宏或宏组相关联。

   cached_page_id
   一个包含了一个标识符的字符串，该标识符用于标识后继 @DTW_CACHE_PAGE 高速缓存请求中的高速缓存中的页，例如页面的 URL。

   age 包含时间长度（以秒计）的字符串变量，是在认为页面过期时指定的。如果请求的页在高速缓存中停留的时间长于值 *age*，Net.Data 将执行页面，并高速缓存所生成的页面，代替过期页面。如果请求的页在高速缓存中停留的时间等于或小于 age 的值，Net.Data 将在高速缓存中检索页面并将它发送到浏览器。在此情况下，Net.Data 立即结束宏执行。

   status 由 Net.Data 返回的一个字符串变量，它指出页面是否已经正确高速缓存。

示例:

```
%HTML(cache_example) {
 %IF (customer == "Joe Smith")
 @DTW_CACHE_PAGE("mymacro.dtw", "http://www.mypage.org", "-1", status)
%ENDIF
 ...
<html>
```

```
 <head>
 <:title>This is the page title</title>
 </head>

 <body>
 <center>
 <h3>This is the Main Heading</h3>
 <p>It is $(time). Have a nice day!
 </body>

</html>
 %}
```

高级高速缓存: 动态确定是否高速缓存

DTW_CACHE_PAGE() 从宏中的位置初启高速缓存。您通常将函数放在宏的开头，以获取最佳性能并确保所有 HTML 都已高速缓存。

对于高级的高速缓存应用程序，当您需要在处理期间决定在某个特定点作高速缓存时，可以把 DTW_CACHE_PAGE() 函数放在 HTML 输出段，而不是在宏的开始。例如，您可能需要根据从查询或函数调用返回的行数来作高速缓存决定。

示例: 因为进行高速缓存的决定取决于 HTML 输出的期望大小，所以把函数放在 HTML 块或 XML 块中

```
 % DEFINE { ...%}

...

%FUNCTION(DTW_SQL) count_rows(){
  select count(*) from customer
%REPORT{
%ROW{
  @DTW_ASSIGN(ALL_ROWS, V1)
%}
%}
%}

%FUNCTION(DTW_SQL) all_customers(){
  select * from customer
%}

%HTML(OUTPUT) {
<html>
 <head>
 <title>This is the customer list
 </head>
 <body>

@count_rows()

 %IF (ALL_ROWS > "100")
 @DTW_CACHE_PAGE("mymacro.dtw", "http://www.mypage.org", "-1", status)
```

```
%ENDIF

@all_customers()

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

在此例中，此页根据 HTML 输出的期望大小作高速缓存或检索。只有当数据库表包含多于 100 行时，才认为 HTML 输出页是值得作高速缓存的。Net.Data 总是在执行这个宏之后，把 OUTPUT 块中的文本 (`This is the customer list`) 发送给浏览器；此文本永不作高速缓存。跟在函数调用后的行 (`@count_rows()`) 在满足 IF 块的条件时作高速缓存或检索。两部分共同形成一个完整的 Net.Data 输出页。

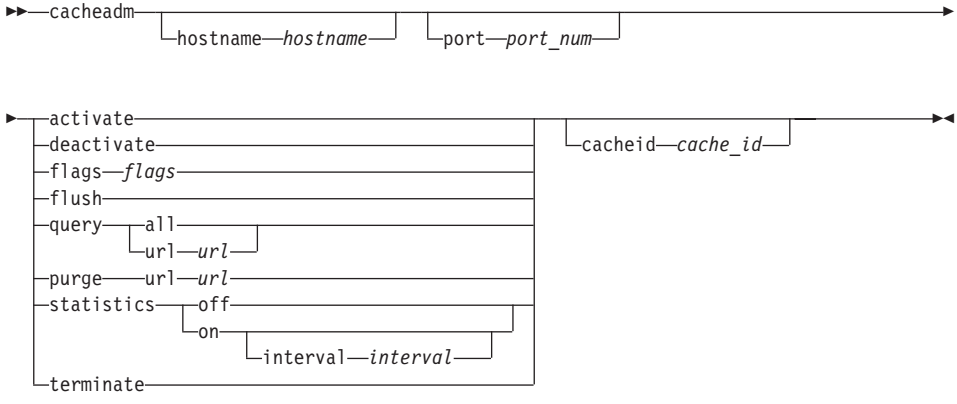
CACHEADM 命令

对于以下任务，使用 CACHEADM 命令:

- 停止高速缓存管理器
- 闪断特定的高速缓存
- 查询特定的高速缓存
- 启用或禁用记录
- 记录标志
- 启动和停止统计值的收集

所有参数都可以缩写为最小唯一字符集。

语法:



参数:

activate

激活指定的高速缓存。如果高速缓存已是活动的，高速缓存管理器就不做任何事。

cache_id

一个字符串变量，识别页面所位于的高速缓存。例如: `cache1`。

deactivate

释放指定的高速缓存。如果高速缓存已是不活动的，高速缓存管理器就不做任何事。完成所有暂挂操作并且不接受新的。最后一个操作完成时，高速缓存管理器将高速缓存标记为不活动。

flags

指定应当打开还是关闭列出的标志。

D_ALL

打开所有跟踪标志。

D_NONE

关闭所有跟踪标志。

flush

闪断一个由 *cache_id* 参数指定的高速缓存，它是该参数必需的。此参数从指定的高速缓存中无条件删除所有项目。

hostname

指定高速缓存所运行的机器的名称，如果该机器不同于发出 `cacheadm` 命令的机器的话。例如: `myhost`。

port_num

指定高速缓存端口号，如果该号码不同于缺省值 (7175) 的话。在系统中，此数目必须是唯一的。

purge

指定要从高速缓存中清除的特定页。如果已指定 *url*，高速缓存管理器将用匹配 *url*的关键字来清除页面。如果已指定从属关系，高速缓存管理器将清除具有关联从属关系的所有项目，并将其关键字写入标准输出流 `stdout`。

query

根据指定的参数来返回高速缓存数据:

- 如果只指定高速缓存标识符，则返回关于此高速缓存的信息。
- 如果指定了 *url*，则返回关于特定的高速缓存的页的信息。
- 如果指定了 `all`，则返回关于所有页的信息。

  其他程序使用 `all` 选项来格式化或解释结果。每行包含以下信息:

  – 页面关键字

– 页面年龄
– 页面长度
– 页面建立日期
– 页面到期日期
– 页面最后引用的日期

所有日期都使用标准的 UNIX 整数时间格式。

性能提示: 选项高速缓存查询全部可能会影响性能，应小心使用。

statistics
启用或禁用对为特定高速缓存收集的统计值的记录，并需要 *cache_id* 参数。如果当 `statistics` 参数设置为 `on` 时指定了一个间隔，则 Net.Data 将更新间隔设置或复位为特定的秒数。

terminate
指定要停止高速缓存管理器。

tranlogging
启用或禁用特定高速缓存的事务处理登记，并需要 *cache_id* 参数。仅当用 `tran-log` 参数在高速缓存管理器配置文件中指定了一个有效的高速缓存事务日志之后，此参数才起作用。

url 全球相对位置 (URL) 地址指定 Web 服务器上文件的位置。例如 `http:www.ibm.com/mydir/page1`。

高速缓存日志

根据内部操作，可保持几种类型的统计值，并可选择写入高速缓存日志。可以为每个高速缓存各自维护一个独立的日志，也可以将所有统计值写入同一日志。本章讨论以下高速缓存日志主题:

- 『配置日志』
- 第192页的『高速缓存日志的格式』

配置日志

要记录统计值，必须配置高速缓存管理器配置文件。

要配置日志:

在高速缓存管理器配置文件中的高速缓存节中指定 `stat-files` 和 `stat-interval` 关键字。

您可以修改统计值设置，而不必停止、重新配置和重新启动高速缓存管理器。

要修改统计值收集设置:

指定 cacheadm statistics 命令。但是请注意，在重新启动高速缓存管理器时，用 cacheadm statistics 命令作的更改不保存。

高速缓存日志的格式

统计日志是一个普通 ASCII 文件，可以通过电子表或数据库程序来处理或导入它。可写入三种类型记录:

- 初始化记录记录了为特定高速缓存收集的统计值的启动。这些记录具有以下格式:

  mm/dd/yy hh:mm:ss id Initialization: interval *n* seconds

  其中:

mm/dd/yy	是收集的统计值启动时的月、日、年
hh:mm:ss	是收集的统计值启动时的时、分、秒
id	是与记录关联的高速缓存的名称
n	是集合间隔

- 终止记录记录了为特定高速缓存收集的统计值的终止。这些记录具有以下格式:

  mm/dd/yy hh:mm:ss id Termination

  其中:

mm/dd/yy	是收集的统计值停止时的月、日、年
hh:mm:ss	是收集的统计值停止时的时、分、秒
id	是与记录关联的高速缓存的名称

- 统计记录是以空格定界的数目集，显示高速缓存中的活动。这些记录具有以下格式:

  mm/dd/yy hh:mm:ss id statistics

  其中:

mm/dd/yy	是收集的统计值创建时的月、日、年
hh:mm:ss	是收集的统计值创建时的小时、分和秒
id	是与记录关联的高速缓存的名称
<*statistics***>**	如 第193页的表15 所示，是一个以空格定界的统计值列表，为该高速缓存而收集:

表 15. 统计值列表

字段号	内容	描述	计数器重新初始化为零
1	读取	基于高速缓存执行的读出操作的数目	是
2	写入	基于高速缓存执行的写操作的数目	是
3	关闭	在高速缓存中的对象上执行的关闭操作的数目	是
4	打开读取	在高速缓存中的对象上执行的打开读取操作的数目	是
5	打开写入	在高速缓存中的对象上执行的打开写入操作的数目	是
6	打开写入查询	在高速缓存中的对象上执行的打开写入查询操作的数目	是
7	读取命中	在高速缓存的对象上的读取命中的数目	是
8	写入命中	在高速缓存的对象上的写入命中的数目	是
9	写入查询命中	在高速缓存的对象上的写入查询命中的数目	是
10	初始化	与该高速缓存确定的新会话的数目	是
11	终止	与该高速缓存终止的会话的数目	是
12	清除	从该高速缓存中删除的对象的数目	否
13	使用的内存	此高速缓存的内存部分中的对象所使用的内存数	否
14	使用的磁盘	此高速缓存的磁盘部分中的对象所使用的磁盘空间数	否
15	可用内存	供此高速缓存的内存部分中的对象所使用的有效内存数	否
16	可用磁盘	供此高速缓存的磁盘部分中的对象所使用的有效磁盘空间数	否
17	内存对象数目	此高速缓存的内存部分中的对象数目	否
18	文件对象数目	此高速缓存的磁盘部分中的对象数目	否
19	会话数目	基于高速缓存的当前活动的会话数目	否

设置出错日志的级别

Net.Data 提供了一个出错日志，这样您就可以监视 Net.Data 系统中的错误或性能问题。

在使用 Net.Data 出错日志时，您可能会注意到如果有许多消息要写入出错日志，会对系统的性能产生影响。例如，每当用户访问 Net.Data 找不到的宏时，Net.Data 把该消息作为输出发送到出错日志中。

为了减少对性能的影响，请检查 Net.Data 宏中用 DTW_LOG_LEVEL 关键字设置的出错日志的记录级别。如果该级别设置为 WARNING，可以考虑将级别降低为 ERROR 以稍微改进性能，或者将其设置为 OFF 以获取较大的性能增益。

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』
- 『SQL 语言环境』
- 第196页的『System 和 Perl 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。程序较少、较大时提供的性能比小程序提供的性能要好，因为每次在宏中调用 REXX 语言环境函数时都要初始化 REXX 解释器。
- 将 REXX 程序存储在一个外部文件中，而不是把 REXX 程序内联包括在 Net.Data 宏中。
- 对于外部 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 变量以及引用变量，可以将仅用于输入的参数直接传送给 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。
- 为避免启动 REXX 解释器的开销，应考虑使用 MACRO_FUNCTION 块来代替 REXX 程序。

SQL 语言环境

在本节中，将描述一些有关数据库和 SQL 语言环境的性能技术。要了解有关 DB2 的性能注意事项，请访问以下 Web 站点: *http://review.ibm.com/software/data/db2/performance*。

数据库技术

以下总结概述了一些可以改进数据库访问的最简单的数据库技术:

- 激活数据库。通过发出命令db2 activate database *databaseName*，连接到数据库的时间将显著缩短。有关 DB2 activate database 命令的更多信息，参见*DB2 Administration Guide*。
- 避免数值转换。在比较列值和文字值时，尝试指定相同的数据类型和属性。如果文字值的精度高于列值的精度，DB2 对于已命名的列将不使用索引。如果要比较的两项具有不同的数据类型，DB2 就必须转换其中一个值，但这样产生的结果将不够精确（由于机器精度的限制）。

  例如，EDUCLVL 是一个半字整数值 (SMALLINT)。指定:

  ```
  ... WHERE EDUCLVL < 11 AND EDUCLVL >= 2
  ```

  来取代:

  ```
  ... WHERE EDUCLVL < 1.1E1 AND EDUCLVL > 1.3
  ```
- 避免字符串填空。在比较定长字符串列值和文字值时，尝试使用相同的数据长度。如果文字值超过列的长度，DB2 将不使用索引。

  例如，EMPNO 是 CHAR(6)， DEPTNO 是 CHAR(3)。指定:

  ```
  ... WHERE EMPNO > '000300' AND  DEPTNO < 'E20'
  ```

  来取代:

  ```
  ... WHERE EMPNO > '000300 ' AND DEPTNO < 'E20 '
  ```
- 避免使用以 % 或 _ 开头的 LIKE 模式。百分号 (%) 和下划线 (_) 用在 LIKE 谓词的模式中时，请指定一个字符串，类似于您想选择的行的列值。在用于表示字符串中间或结尾的字符时，LIKE 模式可以利用索引。例如:

  ```
  ... WHERE LASTNAME LIKE 'J%SON%'
  ```

  当然，用于字符串开头时，LIKE 模式可以防止 DB2 使用任何可能在 LASTNAME 列上定义的索引，从而限制扫描的行数。例如:

  ```
  ... WHERE LASTNAME LIKE '%SON'
  ```

  避免在字符串的开始处使用这些符号，特别是在访问非常大的表时。

SQL 语言环境技术

- 如果结果集中有大量的行，可以使用 START_ROW_NUM 和 RPT_MAX_ROWS 来指定返回给浏览器的结果集的子集。START_ROW_NUM 指定返回的子集应从哪一行开始，而 RPT_MAX_ROWS 指定要返回至页面的最大行数。然后，可以在链接中使用 START_ROW_NUM 来显示下一页结果。

  注意，因为在多个请求中游标的位置并非保持不变，所以 Net.Data 要对每页重新发出查询。

- 考虑使用存储过程来处理复杂的数据库任务。在理解结果集结构的基础上使用嵌入式 SQL 将减少 Net.Data 用来动态描述结果的开销。有关在使用存储过程时的性能折衷的更多信息，参见 *DB2 管理指南*。
- 如果 SQL 语句中唯一更改的信息就是 WHERE 子句中的输入值，应考虑利用 Net.Data 的 DTW_USE_DB2_PREPARE_CACHE 功能。如果不想要全局应用该功能，可在初始化文件或个别宏中将此值设置为 ″YES″。此设置让 Net.Data 对输入值使用主变量，以帮助 DB2 准备语句更快地运行。

System 和 Perl 语言环境

将仅用于输入的参数直接传送给 System 或 Perl 语言环境正在调用的程序。这可以通过定义并引用全局的 Net.Data 变量来实现。对于外部的程序和 Perl 脚本，可以在 %EXEC 语句中从命令行引用这些变量。对于内联的 Perl 脚本，可以在 Perl 源代码中直接引用这些变量。而且，为避免启动 perl 解释器的开销，应考虑使用 MACRO_FUNCTION 块来代替 Perl 脚本。

第8章 Net.Data 记录

在监视 Net.Data 的性能时和解决错误时，Net.Data 提供了好几个日志供您使用。Net.Data 日志包括:

Net.Data 出错信息日志
包含了所有的 Net.Data 出错信息的日志。

“现场连接”日志
包含了“现场连接”出错信息以及 Net.Data、“现场连接”和 DB2 数据库之间的通信的日志。对于 DB2 数据库来说，记录对于 DTW_SQL 和 DTW_ODBC 语言环境是可用的。

Net.Data 跟踪日志
包含所有 Net.Data 跟踪消息的日志。

以下章节描述 Net.Data 记录:

- 『记录 Net.Data 出错信息』
- 第200页的『记录“现场连接 Cliette”和出错信息』
- 第204页的『Net.Data 跟踪日志』

记录 Net.Data 出错信息

Net.Data 将出错信息写入到 Net.Data 错误日志文件 `netdata.log` 中。错误日志的最大大小被 Net.Data 固定为 500 KB，大约 3000 个日志条目。

您可以浏览错误日志文件或归档副本，定期地确定 Net.Data 系统中是否存在问题。

要激活 Net.Data 错误日志:

- 设置 Net.Data 记录配置变量 DTW_LOG_DIR:

  ```
  DTW_LOG_DIR path
  ```

  其中，*path* 是您希望存储错误日志文件的目录。
- 在宏中，设置 Net.Data 记录变量 DTW_LOG_LEVEL:

  ```
  @DTW_ASSIGN(DTW_LOG_LEVEL, "level")
  ```

  其中，*level* 是记录的级别。它可以是以下值:

  off Net.Data 不记录错误。这是缺省。

error Net.Data 记录出错信息。

本节将讨论以下记录主题:

- 『规划 Net.Data 错误日志』
- 『控制 Net.Data 记录级别』
- 第199页的『Net.Data 出错信息不被记录的类型』
- 第199页的『Net.Data 日志文件的大小和循环』
- 第199页的『Net.Data 错误记录格式』

规划 Net.Data 错误日志

在记录错误时，需要计划以下问题:

- 确定磁盘空间:

  如果您计划使用错误记录，则必须允许对错误日志使用额外的磁盘空间。

- 配置 Net.Data:

  如果您计划控制整个 Net.Data 系统的错误记录，则应在 Net.Data 初始化文件中设置一个配置变量: DTW_LOG_DIR

  这个变量是错误记录必需的，即使已经在宏中将 DTW_LOG_LEVEL 变量设置为 error 或 warning。参见第17页的『DTW_LOG_DIR 和 DTW_LOG_LEVEL: 错误日志变量』，以学习如何更新初始化文件。

- 编写 Net.Data 宏:

  在宏中用 DTW_LOG_LEVEL 关键字来设置记录的级别。

- 运行 Net.Data:

  如果您在使用错误记录，那么您可以检查 Net.Data 系统中错误的错误日志和档案文件。

- 调节:

  请注意，记录将会影响性能。参见第193页的『设置出错日志的级别』，以获取有关性能问题的信息。

控制 Net.Data 记录级别

您可以使用 DTW_LOG_LEVEL 变量来指定记录的级别。在 Net.Data 宏中定义此关键字。此变量有三个设置:

off Net.Data 不记录错误。这是缺省值。

error Net.Data 记录出错信息。

warning
Net.Data 记录警告和出错信息。

Net.Data 出错信息不被记录的类型

Net.Data 不记录由宏中的 MESSAGE 部分显式处理的错误。

Net.Data 日志文件的大小和循环

日志文件的最大大小可达 500 KB。在这种情况下，大约可以装入 3000 个日志条目。

当日志文件到达最大大小时，该文件将被归档到 netdata.log*MMMDDYYYY_nn*

其中:

MMM 月份 (Jan-Dec)

DD 日期

YYYY 年

nn 一个从 01 到 99 的数字，用于唯一地标识某一天的每个档案文件。

在原来的文件中继续记录。

Net.Data 错误记录格式

日志文件项具有以下格式:

[*DD/MMM/YYYY:HH:MM:SS*] [*MACRO*] [*BLOCK*] [*PID#*] [*TID#*] *error_message*

参数:

DD 日期

MMM 月份 (Jan-Dec)

YYYY 年

HH 小时 (00-23)

MM 分钟 (00-59)

SS 秒 (00-59)

MACRO
生成出错信息的宏

BLOCK
生成出错信息的 HTML 块的名称。

PID# 生成出错信息的进程的进程标识号。此标识是必需的，因为多个 Net.Data 进程都可以写入这个日志文件。

TID# 生成出错信息的线程的线程标识号。此标识是必需的，因为来自同一个 Net.Data 进程的多个线程都可以写入这个日志文件。

error_message
出错信息的文本

记录“现场连接 Cliette”和出错信息

“现场连接”在“现场连接”日志文件中记录信息以及“现场连接”、Net.Data 和 DB2 数据库之间的通信。日志的最大大小被 Net.Data 固定为 1 MB，大约 1200 个日志条目。

您可以浏览日志文件或副本，定期确定 cliette 或 DB2 数据库是否存在问题。

要激活“现场连接”日志:

用 -l 属性启动“连接管理器”：

```
dtwcm -l [level]
```

其中，*level* 是记录的级别。它可以是以下值:

normal
“现场连接”记录所有的 cliette 活动、相关的 DB2 SQL 语句和状态信息和“现场连接”出错信息

minimal
“现场连接”仅记录重要信息，例如数据库查询和结果集中的行数。

本节将讨论以下记录主题:

- 『规划“现场连接”日志』
- 第201页的『控制“现场连接”记录级别』
- 第201页的『不会记录的“现场连接”信息的类型』
- 第201页的『“现场连接”日志文件名』
- 第202页的『“现场连接”日志文件的大小和循环』
- 第203页的『“现场连接”日志格式』

规划“现场连接”日志

在记录信息时，需要规划以下问题:

- 确定磁盘空间:
  如果您计划使用错误记录，则必须允许对日志文件使用额外的磁盘空间。
- 运行“连接管理器”：

您可以通过在 dtwcm 命令中输入一个属性来激活记录。参见第200页的『记录“现场连接 Cliette”和出错信息』以了解有关语法。

如果使用记录，那么您可以通过检查这些记录和档案文件来了解 cliette 的错误。

- 调节:

  请注意，记录将会影响性能。参见第193页的『设置出错日志的级别』，以获取有关性能问题和『控制“现场连接”记录级别』的信息

控制“现场连接”记录级别

您可以在调用“连接管理器”时在 dtwcm 命令中指定记录的级别。dtwcm 命令的 -l 属性具有两种设置:

normal

“现场连接”记录所有的 cliette 活动、相关的 DB2 SQL 语句和状态信息和“现场连接”出错信息

minimal

“现场连接”仅记录重要信息。这个选项在记录中提供的信息要少一些。

不会记录的“现场连接”信息的类型

“现场连接”不记录由宏中的 MESSAGE 部分显式处理的 Net.Data 错误。

“现场连接”日志文件名

“现场连接”为“连接管理器”和每个 cliette 创建一个日志文件。下面的列表描述了名称的格式:

“连接管理器”文件

格式:

conman-*process_id*-*DDMMMYYYYHHMMSS*.log

参数:

process_id
: “连接管理器”进程的标识符

DD
: 日期

MMM
: 月份 (Jan-Dec)

YYYY
: 年

HH
小时，24 小时时钟

MM
分钟

SS 秒

示例:

```
conman-513-01Feb1999095639.log
```

Cliette 文件

格式:

```
cliett-process_id-DDMMMYYYYHHMMSS.log
```

参数:

process_id
cliette 进程的标识符

DD
日期

MMM
月份 (Jan-Dec)

YYYY
年

HH
小时，24 小时时钟

MM
分钟

SS 秒

示例:

```
cliett-592-01Feb1999095647.log
```

“现场连接”日志文件的大小和循环

日志文件的最大大小可达 1 MB。在这种情况下，大约可以装入 6000 个日志条目。当日志文件达到最大大小时，该进程将关闭原始的日志文件，创建新的日志文件，然后继续在新文件中进行记录。

日志文件位于与 `dtwcm` 和 `dtwcdb2` 所在的相同的目录中

“现场连接”日志格式

日志文件项具有以下格式:

```
--process_type-DD/MMM/YYYY:HH:MM:SS-PID#--
message_text
```

参数:

process_type
dtwcm 或 cliet，取决于“连接管理器”或 cliette 是否记录了信息。

DD 日期

MMM 月份 (Jan-Dec)

YYYY 年

HH 小时 (00-23)

MM 分钟 (00-59)

SS 秒 (00-59)

PID# 生成信息的进程的进程标识号。

message_text
信息的文本。

示例 1: 一个“连接管理器”日志条目。

```
--dtwcm-02/Mar/1999:13:43:07-330--
Creating connection manager ...successfully
Reading configuration info ...
Completing initialization ...
Initializing cm server ... successfully
Initializing NLS environment ... successfully
Detecting cliette ./dtwcdb2 for DTW_SQL:CELDIAL:
    Min process(es) = 1,
    Priv Port = 7100.
Starting 1 cliettes for DTW_SQL:CELDIAL.
Started: ./dtwcdb2 7128 7100 7200 DTW_SQL:CELDIAL  LOG_MAX , pid: 213
1 cliettes for DTW_SQL:CELDIAL started.
...
```

示例 2: 一个 cliette 日志条目。

```
--cliet-02/Mar/1999:13:43:08-335--
Cliette starting ...
Cliette: DTW_SQL:SAMPLE, database: SAMPLE, user: *USE_DEFAULT
Making a new connection to database: SAMPLE, user: *USE_DEFAULT.
Calling SQLAllocHandle for environment ...
Calling SQLAllocHandle for connection ...
Calling SQLSetConnectAttr ...
Calling SQLConnect ...
```

```
Connecting to database: SAMPLE sucessfully.
Telling CM the cliette is ready ...
Ready and waiting for command from CM ...
```

Net.Data 跟踪日志

Net.Data 提供了一些跟踪数据，关于跟踪日志中所记录的宏的执行情况。您可以指定在哪里存储跟踪日志，以及记录何种级别的跟踪。使用 IBM 跟踪消息以在与 IBM 服务代表一起工作时提供信息。有关 Net.Data 跟踪消息的列表，参见 *Net.Data 消息和代码参考*。

配置 Net.Data，以进行跟踪

要针对跟踪配置 Net.Data，需要设置配置变量，以指定在哪里存储跟踪日志，以及 Net.Data 需要捕捉何种级别的跟踪数据。

- 『设置跟踪日志目录』
- 『设置跟踪记录的级别』

设置跟踪日志目录

跟踪日志的名称为 `netdata.trace.pid`，其中 `pid` 是处理该请求的进程的标识。使用 DTW_TRACE_LOG_DIR 配置变量来指定存储跟踪文件的目录。

注: Net.Data 将日志文件的大小限制为 50 MB。如果日志文件大小达到 50 MB，文件被归档为 `net.data.trace.pid.`*`ϕ`*。

语法:

```
DTW_TRACE_LOG_DIR [=] full_directory_path
```

示例:

```
DTW_TRACE_LOG_DIR /usr/lpp/internet/server_root/logs
```

设置跟踪记录的级别

通过设置 DTW_TRACE_LOG_LEVEL 配置变量的值来确定 Net.Data 记录的跟踪的级别。

语法:

```
DTW_TRACE_LOG_LEVEL [=] OFF|APPLICATION|SERVICE
```

其中:

OFF 指定不在跟踪日志中捕捉跟踪数据。这是缺省值。

APPLICATION
Net.Data 将应用程序级的跟踪消息写至跟踪日志。

SERVICE

Net.Data 将*所有*跟踪消息写至跟踪日志。

示例:

```
DTW_TRACE_LOG_LEVEL SERVICE
```

跟踪日志格式

跟踪日志条目的格式是:

[DD/MMM/YYYY:HH:MM:SS][macro][PID#][TID#][UID] trace_message

其中:

DD/MMM/YYYY:HH:MM:SS

是一个时间戳记，它指示跟踪条目的创建时间。

macro 是生成宏信息的宏的名称。

PID# 是生成跟踪消息的进程的进程标识。

TID# 是生成跟踪消息的线程的标识。

UID 是生成跟踪消息的用户的标识。

trace_message

是跟踪消息的文本。

访问权

要成功地将跟踪消息写至跟踪日志文件， Net.Data 执行时使用的用户标识必须:

- 对 DTW_TRACE_LOG_DIR 配置变量中指定的日志目录具有写入权限。
- 对路径中的所有目录（包括日志目录）具有执行权限。

附录A. 书目提要

Net.Data 技术资料库

Net.Data 技术资料库可以从 Net.Data 的以下 Web 站点访问:

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

文档	描述
• *Net.Data 管理与编程指南, OS/390 版* • *Net.Data 管理与编程指南, OS/2、Windows NT 和 UNIX 版* • *Net.Data 管理与编程指南, OS/400 版*	包含有关安装、配置和调用 Net.Data 的概念和任务信息。还描述了如何编写 Net.Data 宏、如何使用 Net.Data 性能技术、如何使用 Net.Data 语言环境、如何管理连接、以及如何使用 Net.Data 日志记录和跟踪来进行故障诊断、调节性能。
Net.Data Reference	描述 Net.Data 宏语言、变量和内置函数。
Net.Data 语言环境接口参考	描述 Net.Data 语言环境接口。
Net.Data 消息和代码参考	列出 Net.Data 出错信息和返回码。

附录B. Net.Data for AIX

AIX 版本的详细信息已经包含在与 Net.Data 一起提供的“自述文件”中。自述文件中包含以下信息:

- 需求
- 安装
- 配置
- 解除安装

为语言环境装入共享程序库

在 AIX 平台上创建语言环境时，需要装入共享程序库。在 AIX 中，当提供一个由 Net.Data 调用的例程时，或返回语言环境例程（例如 dtw_initialize() 和 dtw_execute()）的地址时，需要提供语言环境。

Net.Data 使用 dtw_fp 结构从 AIX 的语言环境中检索指向该语言环境接口例程的指针，其格式如下:

```
typedef struct dtw_fp {
    int (* dtw_initialize_fp)();   /* dtw_initialize 函数指针         */
    int (* dtw_execute_fp)();      /* dtw_execute 函数指针            */
    int (* dtw_cleanup_fp)();      /* dtw_cleanup 函数指针            */
} dtw_fp_t;
```

当装入共享库时，这个结构由 Net.Data 作为 dtw_getFp() 例程中的一个参数被传递到语言环境。

作为仅有的参数来传递 dtw_fp 结构。此结构包含一个用于每个所支持接口的字段，并且由语言环境来设置这些字段。如果语言环境提供指定的接口，则它将该字段设置成指向指定接口的函数指针。 如果语言环境没有提供指定的接口，则它将该字段设置成 NULL。程序模板中的 dtw_getFp() 例程显示了此例程的一个正确实现。

为了在装入共享库时 Net.Data 能够获得指向这个例程的指针，dtw_getFp 例程必须是在共享库的导出文件中指定的第一个入口点。以下是 dtwsampshr.o 库的导出文件的一个示例，这个库支持所有可用的语言环境接口例程。

```
#!dtwsampshr.o
dtw_getFp
dtw_initialize
dtw_execute
dtw_cleanup
```

改进 REXX 环境的性能

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

您可以用以下三种方式来设置环境变量:

- 在宏中，通过使用 DTW_SETENV 内置函数:

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

- 在 AIX 系统环境文件中:

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

  此方法将影响整个机器上 REXX 的功能。

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

  ```
  InheritEnv RXQUEUE_OWNER_PID = 0
  ```

  此方法将影响 Web 服务器上 REXX 的功能。

NLS 考虑

Net.Data 使用与 Web 服务器相同的代码页运行。为了使 Net.Data 使用针对您的“语言环境”的正确的代码页， Web 服务器必须使用正确的代码页。例如，然后您使用 IBM Internet Connection Server 并希望使用一个韩国语代码页，则应停止服务器并用韩国语的语言环境重新启动:

```
stopsrc httpd
startsrc -s httpd -e "LC_ALL=ko_KR"
```

如果宏中包含 UTF-8 字符，或者您连接到一个包含 Unicode 数据的 DB2 数据库，请在初始化文件中将 DTW_UNICODE 配置变量设置为 UTF8。目前，Net.Data 在宏中支持 UTF-8 字符，但不支持 UTF-16。 当然，Net.Data 可以处理 UTF-8 或 UCS-2 编码的 DB2 数据库数据。Net.Data 的输出总是 UTF-8。

性能提示: 如果在双字节的语言环境中运行，而您的字符串或工作内置函数通常处理的是单字节字符串，请将 DTW_MBMODE 设置为 NO 以避免不必要的转换。

附录C. Net.Data 向导

Net.Data 向导是为了给您提供一种创建定制 Net.Data 应用程序的便捷方式而设计的。只要简单地选择一个向导，回答几个问题，Net.Data 就会为您创建一个定制应用程序。

Net.Data 为您提供了以下向导，您可以在学习如何创建宏和使用 Net.Data 特征的时候使用这些向导:

下访 此向导能够取得您现有的数据库表并创建一个支持 Web 功能的下访应用程序，这个下访应用程序能使您访问不同详细程度的数据。还可以选择将下访向导连接到数据库，以便收集数据库信息。您最多可以定制 5 个 Web 报告。生成的宏使用存储在您数据库中的主键和外键信息来自动链接您的 Web 报告。

存储过程
此向导将连接至您的数据库，并获取一张所有登记在数据库中的存储过程的列表。选择一个存储过程，向导就会为您生成一个调用存储过程的 Net.Data 宏。然后，您可以修改生成的宏，或者将它集成到现有的 Net.Data 应用程序中。

联络 此向导生成一个支持 Web 功能的通讯录，可用于存储姓名、地址、电话号码和其它重要的联络信息。它带有一个搜索功能，可以让您快速访问您的联络人。生成的联络应用程序中可以包括一个简短报告或一个详细报告。另外，您也可以包括一个定制报告。

请查看 Net.Data 的 Web 站点: `http://www.ibm.com/software/data/net.data`, 以获取 Net.Data 向导软件包的最新版本。

本附录将讨论以下主题:

- 『开始之前』
- 第212页的『运行向导』

开始之前

要运行此向导并生成 Net.Data 宏，必须先安装以下软件:

- Java Development Kit (JDK) 或 Java Runtime Environment (JRE) 1.1.x
- Net.Data 版本 2 或更高版本
- IBM Universal Database (UDB) 5.0 或更高版本

- REXX（下访向导所必需）

运行向导

Net.Data 向导是从命令行启动的，它们包含在文件 NetDataSmartGuides.jar 中。

要使用 Java Development Kit (JDK) 来启动向导:

1. 将以下行添加到 CLASSPATH 环境变量中。

   *[Path]*NetDataSmartGuides.jar

   其中 *[Path]* 是到 NetDataSmartGuides.jar 文件的可选路径。

2. 如果您想要使用下访和存储过程向导的数据库连接功能，请在 CLASSPATH 环境变量中添加 UDB JDBC 驱动器。

   对于 Windows NT 和 OS/2 操作系统，在您的 CLASSPATH 环境变量中添加以下这一行:

   *[Path]*NetDataSmartGuides.jar;*[UDBInstallationPath]*\java\db2java.zip

   对于 UNIX 操作系统，请向您的 CLASSPATH 环境变量中添加以下行:

   *[Path]*NetDataSmartGuides.jar:*[UDBInstallationPath]*\java\db2java.zip

   其中 *[Path]* 是到 NetDataSmartGuides.jar 文件的可选路径，*[UDBInstallationPath]* 是到 UDB 安装的路径，例如 C:\SQLLIB。

3. 启动向导。
   - 对于 UNIX 操作系统，输入以下命令:

     java TaskGuide LaunchPad.class

   - 对于 Windows NT 和 OS/2 操作系统，运行以下文件:

     SmartGuides.cmd

   将打开一个具有可用向导列表的启动板，如第213页的图29中所示。

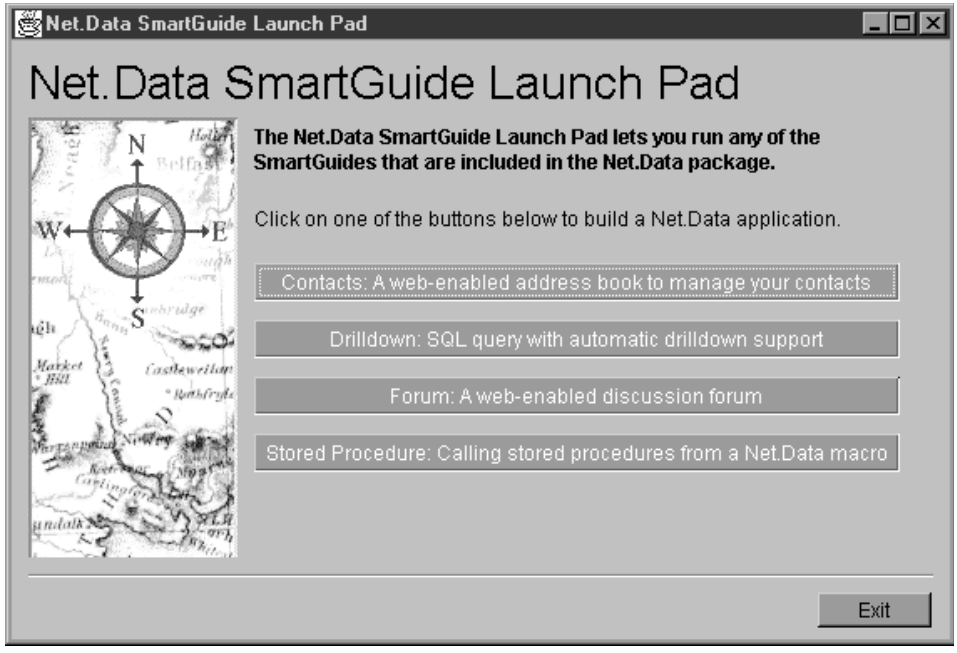


图 29. 向导启动板

4. 单击您想要运行的向导的名称。

要使用 Java Runtime Environment (JRE) 来启动向导:

1. 对于 Windows NT 操作系统，选择**开始->程序->Net.Data->向导**，这样将运行一个名为 SMARTGUIDES.BAT 的批处理文件。对于其他操作系统，输入以下命令来运行向导:

   ```
   jre -cp [Path]NetDataSmartGuides.jar TaskGuide LaunchPad.class
   ```

   其中 *[Path]* 是到 NetDataSmartGuides.jar 文件的可选路径。

   将打开一个具有可用向导列表的启动板，如图29中所示。
2. 如果您想要使用下访和存储过程向导的数据库连接功能，请输入以下命令:

   对于 Windows NT 和 OS/2 操作系统:

   ```
   jre -cp [Path]NetDataSmartGuides.jar;[UDBInstallationPath]
     \java\db2java.zip TaskGuide LaunchPad.class
   ```

   对于 UNIX 操作系统:

   ```
   jre -cp [Path]NetDataSmartGuides.jar:[UDBInstallationPath]
     \java\db2java.zip TaskGuide LaunchPad.class
   ```

其中 *[Path]* 是到 NetDataSmartGuides.jar 文件的可选路径，*[UDBInstallationPath]* 是到 UDB 安装的路径，例如 C:\SQLLIB。

3. 单击您想要运行的向导的名称。

附录D. 用 Net.Data SQL 辅助来构建 SQL 语句

Net.Data SQL 辅助是一个基于 Java 的 SQL 语句编制器，它提供了一个易于使用的 GUI，指导您完成构建 SQL 语句的过程。通过使用 Net.Data SQL 辅助，您能够:

- 构建 SELECT、INSERT、UPDATE 和 DELETE 语句(包括 SELECT DISTINCT)
- 使用值查找、AND 或 OR、输入关键输入字段来构建复合条件
- 定义表 JOINS 操作(内部、右外、左外)
- 选择要查看的列
- 选择排序顺序
- 输入在条件、值和排序中要使用的用户定义的变量

构建 SQL 语句之后，您能够:

- 将 SQL 语句另存为一个文件
- 生成并保存包含 SQL 语句的宏
- 将 SQL 语句或宏复制到剪贴板

本附录将讨论以下主题:

- 『开始之前』
- 第216页的『运行 Net.Data SQL 辅助』

开始之前

要运行 Net.Data SQL 辅助，必须先安装以下软件:

- Java Development Kit (JDK) 或 Java Runtime Environment (JRE) 1.1.x
- 一个支持使用 JDBC 的数据库

参见您的数据库文档，以获取有关通过 JDBC 访问数据源的更多详细信息以及其它可能需要的服务器启动步骤。例如，在远程访问 DB2 UDB 版本 5.0 的数据源时，数据库服务器必须运行 JDBC 服务器 (db2jstrt)。

运行 Net.Data SQL 辅助

Net.Data SQL 辅助是从命令行启动的，它包含在文件 {*inst_dir*}/assist/NetDataAssist.jar 中。

要使用 Java Development Kit (JDK) 来启动 Net.Data 辅助:

输入以下命令来启动 Net.Data 辅助:

```
java -classpath %CLASSPATH%;{inst_dir}/assist/NetDataAssist.jar NetDataAssist
```

要使用 Java Runtime Environment (JRE) 来启动 Net.Data 辅助:

输入此命令来启动 Net.Data 辅助:

```
jre -cp %CLASSPATH%;{inst_dir}/assist/NetDataAssist.jar NetDataAssist
```

单击**下一步**按钮，然后逐个经过用于登录、构造 SQL 语句和生成 Net.Data 宏的窗口。

附录E. 使用 NetObjects Fusion NOF 插件和 Net.Data 小服务程序

Net.Data 为 Net.Data 小服务程序提供了一个 NetObjects Fusion 插件。

您可以使用 NetObjects Fusion (NOF) 来更好地集成现有的 Net.Data 宏和 Web 站点管理，它还提供了一个便于使用的图形用户界面。

本附录将讨论以下主题:

有关 NetObjects Fusion 插件

NetDataServlet.NFX 插件与 Net.Data 小服务程序一起使用。这个 NOF 插件支持对现有 Net.Data 宏或单个 Net.Data 函数的调用。该插件生成 HTML，将 Net.Data 作为小服务程序或服务器方包含文件 (SSI) 来调用。当 Web 服务器调用 Net.Data 时，Net.Data 宏或函数将运行。使用 Net.Data 小服务程序插件将宏和函数小服务程序插入 NOF 管理的 Web 站点，如第218页的图30所描述。该插件提供了系统基本宏或函数参数，以及为自动构建宏而选择的缺省项。要使用这个插件，必须安装并配置 NOF 和插件文件。

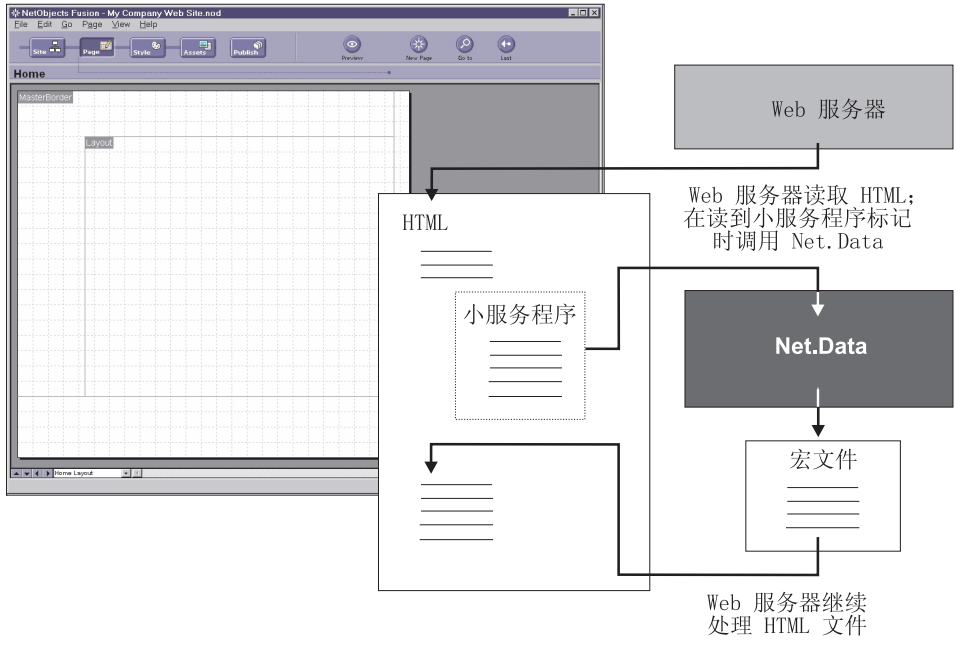


图 30. Net.Data 小服务程序插件

安装 NetObjects Fusion 插件

软件与硬件需求:

NOF 插件需要 NetObjects Fusion 版本 2.0 或更高版本。

要安装: 将 <inst_dir>\fx\NetDataServlet.nfx 和 <inst_dir>\fx\NetDataServlet.gif 复制到您的 <NetObjects Fusion>\components\ 目录。

为 NetObjects Fusion 设置 Net.Data 插件

您可以使用 NOF 来更改正在使用的小服务程序的特性。

1. 打开 NetObjects Fusion。
2. 从 NetObjects Fusion (NOF) 的 **Tools** 组件栏选择 **NetObjects Components** 按钮:  插件按钮显示在 **Tools** 组件栏的底部。
3. 从这六个 **Tools** 组件栏按钮中选择 **NetObjects Components** 按钮: 
4. 在 NOF 画面上，标记您想要放置所选插件的区域。这是显示小服务程序结果的地方。将打开显示插件列表的“已安装组件”窗口，您可以从中进行选择。如

果小服务程序插件不在列表中，可以使用路径以及文件名字段来指定插件的文件名 NetDataServlet.NFX，以便与宏小服务程序以及函数小服务程序一起使用

5. 从列表中选择小服务程序插件并单击**确定**按钮。

   该插件将变为 NOF 画布上的一个对象。

修改插件的特性

您可以使用 Net.Data 小服务程序插件来修改宏和函数小服务程序。

要使用 NOF 来修改 Net.Data 小服务程序:

1. 在 NOF 画面上，标记您想要放置所选插件的区域。这是显示小服务程序结果的地方。将打开显示插件列表的“已安装组件”窗口，您可以从中进行选择。如果小服务程序插件不在列表中，可使用路径及文件名字段来指定插件的文件名 NetDataServlet.NFX，以便与宏小服务程序以及函数小服务程序一起使用。
2. 从列表中选择 Net.Data 小服务程序插件并单击**确定**按钮。

   该插件将变为 NOF 画布上的一个对象。
3. 选择并定制 Net.Data 小服务程序插件的特性:
   a. 在 NOF 画面上选择 Net.Data 小服务程序插件。将打开 NOF **特性**组件栏，显示插件的特性，如第220页的图31中所示。

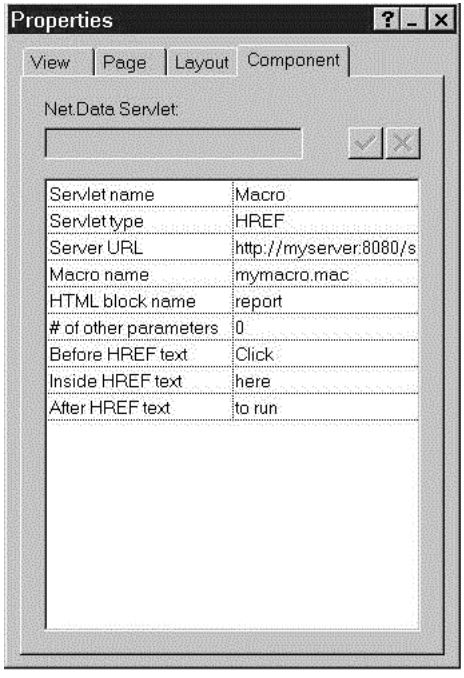


图 31. Net.Data 小服务程序 的 Properties 组件栏

Net.Data 小服务程序的特性:

您可以定制以下特性:

Servlet name

选择您想要调用的小服务程序的名称: Function 或 Macro。根据您所选择的小服务程序名称，将显示不同的特性。

Servlet type

选择想要的小服务程序类型: SSI、HREF 或 FORM 提交按钮。根据您所想要的小服务程序类型，将显示不同的特性。

Submit label

如果选择 FORM 提交按钮类型，则请指定提交标号的文本。否则，将不显示此特性。

Server URL

如果选择 HREF 小服务程序类型，则需要向支持小服务程序功能的 Web 服务器指定服务器 URL。如果选择 SSI，将不显示此特性。

Macro name

如果选择宏小服务程序的名称，则指定要执行的现有 Net.Data 宏的名称。否则，将不显示此特性。

HTML block name

如果选择宏小服务程序的名称，则在要运行的 Net.Data 宏中指定 HTML 块的名称。否则，将不显示此特性。

Function type

如果选择函数小服务程序的名称，则选择要执行的函数类型：Function 或 SQL。否则，将不显示此特性。

Language env

如果选择函数小服务程序的名称，则指定要使用的 Net.Data 语言环境。否则，将不显示此特性。

Statement

如果选择函数小服务程序的名称，则指定要执行的语句。否则，将不显示此特性。

Database

如果选择函数小服务程序名，则指定要使用的数据库的名称。否则，将不显示此特性。

# of other parameters

指定要传递给 Net.Data 的其他参数个数（最大数目：25）。对于每个参数，输入参数的名称及其值（可选）。

Before HREF text

如果选择 HREF 小服务程序类型，则在要显示在 `<a href` HTML 标记前的文本前面指定要显示的文本。否则，将不显示此特性（可选）。

Inside HREF text

如果选择 HREF 小服务程序类型，则在 `<a href` HTML 标记内指定要显示的文本。否则，将不显示此特性（可选）。

After HREF text

如果选择 HREF 小服务程序类型，则在要显示在 `<a href` HTML 标记后的文本后面指定要显示的文本。否则，将不显示此特性（可选）。

SQL Reminder!

如果选择 HREF 小服务程序类型并指定了一个 SQL 函数类型，将

显示带有提示的消息，告诉您 HREF SQL 语句应对任何空格字符 ( ) 使用加号字符 (+)。在页面发行之后，文本既不能更改，也不能显示。否则，将不显示此特性。

b. 在定义页面的特性以后，您就可以单击**发布**按钮来用 Net.Data 小服务程序 NOF 插件构建并发布 Web 页面。

注: 如果选择了 SSI 小服务程序类型，Web 页面文件的扩展名就应当是 `.shtml`。您可以在 NOF **特性**组件栏中将此设置为页面的缺省值: 选择**页面**笔记本选项卡，单击**定制**按钮并在**扩展名类型**字段中输入 `.shtml`。

使用 NOF 插件发布小服务程序

在设置页面的特性以后，您就可以单击**发布**按钮来用插件构建并发行 Web 页面。

附录F. 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。

此宏使用一个包含文件，其中包含用于该宏的 DEFINE 块。

第224页的图32显示了示例宏。第226页的图33显示了包含文件。

```
%{************************ Sample Macro ****************************
*   FileName = sqlsamp1.dtw                                          *
*   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 EMPLOYEE
%MESSAGE {
    -204:  {<p><b>ERROR -204: Table EMPLOYEE not found. </b> </p>
           %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}

       %REPORT {
<select name="emp_name">
%ROW{
<option>$(V1)</option>
%}
</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 EMPLOYEE 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 32. 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. </p>
<hr />
<form method="post" action="report">
@queryDB()
<input type="submit" value="Select Employee" />
</form>
<hr />
 </body>
</html>
%}
```

图 32. 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></p>
<hr /><a href="input">Return to previous page</a>
 </body>
</html>
%}

%{     End of Net.Data macro 1 %}
```

图 32. 示例宏 (3/3)

```
=========================================================================
%{************************** Include File *********************************
*   FileName = sqlsamp1.hti                                               *
*   Description:                                                          *
*     This include file provides global DEFINEs for the sqlsamp1.dtw      *
*     Net.Data macro.                                                     *
*************************************************************************%}
%define {
    emp_name   =""
    reposition = sign
    exampleTitle = "Sample Macro"
    %}

%{     End of include file  %}
```

图 33. 包含文件

声明

本资料是为在美国提供的产品和服务而开发的。IBM 可能未在其他国家提供本文档中讨论的产品、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。提供本文档，并不表示允许您使用这些专利。您可以将许可证查询以书面形式寄往:

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 知识产权部门联系，将查询以书面形式寄往:

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条款的国家: 国际商业机器公司“按原样”出版此书，不做任何明确或暗示的担保，包括但不限于有关非伪造、商业性或符合特殊目的的隐含保证。一些地区在某些事务中不允许否认拒绝明确或暗示的担保，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 可以随时对本书中说明的产品和/或程序进行改进和/或改动，而不必通知您。

本出版物中对非 IBM Web 站点的任何引用仅是为了方便起见，而不以任何方式为那些 Web 站点作保证。那些 Web 站点的资料并非此 IBM 产品资料的一部分，使用那些 Web 站点的风险由您自己承担。

为了以下目的: (1) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换 (2) 允许对已经交换的信息进行相互使用，而希望获取本程序有关信息的合法用户请与下列地址联系:

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

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

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、其发行公告或其它公众可用源得到。IBM 未测试这些产品，因此不能确认性能的精确度、兼容性或其它对非 IBM 产品的索赔赔偿要求等。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随时更改或撤消，而不作任何通知，并且仅代表发展目标。

本资料仅用于计划目的。在讨论的产品可用之前，此处的资料可能会作出更改。

此信息包含了用于日常商业处理的数据和报表的示例。为了尽可能完整地说明问题，这些示例中包含了个人、公司、品牌和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址相似，纯属巧合。

注册商标

以下各项是 IBM 公司在美国和／或其他国家的商标:

AIX
AS/400
DB2
DB2 Universal Database
DRDA
DataJoiner
IBM
IMS
Language Environment
MVS/ESA
Net.Data
OS/2OS/390
OS/400
OpenEdition

以下各项是其他公司的商标:

Java 和所有基于 Java 的商标和徽标是 Sun 公司在美国和 / 或其他国家的商标。

UNIX 是在美国和 / 或经 X/Open 有限公司唯一许可的其他国家的注册商标。

Lotus 和 Domino Go Webserver 是莲花软件有限公司在美国和 / 或其他国家的商标。

Microsoft、Windows、Windows NT 和 Windows 徽标是微软公司在美国和 / 或其他国家的商标或注册商标。

以双星号 (**) 标出的其他公司、产品和服务的名称，可能是其他公司的商标或服务标志。

索引

[D]

[F]

[G]

[H]

[J]

[K]

[L]

[P]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

[特别字符]

A

B

C

N

O

P

R

S

T

U

W

Printed in China