IBM PowerHA SystemMirror for AIX
Standard Edition

V7.2

PowerHA SystemMirror 客户机应用程序编程

IBM PowerHA SystemMirror for AIX
Standard Edition

V7.2

PowerHA SystemMirror 客户机应用程序编程

注意

在使用本资料及其支持的产品之前，请阅读第 105 页的『声明』中的信息。

此版本适用于 IBM PowerHA SystemMirror 7.2 Standard Edition for AIX 以及所有后续发行版和修订版，直到在新版本中另有声明为止。

目录

关于本文档

本文档描述了 PowerHA® SystemMirror® 软件随附的集群信息程序 (Clinfo) 客户机应用程序编程接口 (API) 和管理信息库 (MIB)。

突出显示

本文档中使用了以下突出显示约定:

粗体	标识命令、子例程、关键字、文件、结构、目录和名称由系统预先定义的其他项。它也标识图形对象，例如用户选择的按钮、标签和图标。
斜体	标识将由用户提供其实际名称或值的参数。
等宽字体	标识特定数据值示例、与您所看到的显示的文本相类似的文本示例、与您（作为程序员）所写的程序代码相类似的部分程序代码示例、来自系统的消息或您应实际输入的信息。

AIX 区分大小写

AIX® 操作系统中的所有内容都是要区分大小写的，这意味着大写和小写是有区别的。例如，可以使用 **ls** 命令列出文件。如果您输入 LS，那么系统会响应该命令找不到。同样，**FILEA**、**FiLea** 和 **filea** 是三个不同的文件名，即使它们位于同一个目录中也是如此。为了避免执行不期望的操作，请始终确保使用正确的大小写。

ISO 9000

在本产品的开发和制造过程中，使用了 ISO 9000 注册质量体系。

相关信息

- PowerHA SystemMirror PDF 文档在以下主题提供: PowerHA SystemMirror 7.2 PDFs。
- PowerHA SystemMirror 发行说明在以下主题提供: PowerHA SystemMirror 7.2 release notes。

PowerHA SystemMirror 客户机应用程序编程

本信息描述了 PowerHA SystemMirror 软件随附的集群信息程序 (Clinfo) 客户机应用程序编程接口 (API) 和管理信息库 (MIB)。

应用程序可以访问存储在 PowerHA SystemMirror for AIX MIB 中的有关 PowerHA SystemMirror 集群的信息。应用程序可直接通过进行简单网络管理协议（SNMP）请求或间接通过使用 Clinfo C 或 C++ API 来访问此信息。

集群信息程序

这些主题概述了集群信息程序 (Clinfo)，并描述了 Clinfo 接收和维护的有关 PowerHA SystemMirror for AIX 集群的状态信息。

集群信息程序概述

PowerHA SystemMirror 集群的状态可随着时间的推移经历各种转变。例如，节点可以加入或脱离集群，或者应用程序可以故障转移到备份节点。其中每个更改都会影响集群的状态。因为集群是动态的，所以应用程序必须能够获取有关集群的最新准确信息，以便可以在发生变化时及时应对。Clinfo 提供了此服务。

PowerHA SystemMirror 集群软件通过 SNMP（一种业界标准网络协议）导出状态信息和集群事件。针对 SNMP API 编写程序会有一定难度，并且可能需要若干事务来获取与单个集群实体（例如节点或资源组）相关的所有信息。

Clinfo API 以及相关联的组件提供了访问例程库，这些例程通过使用明确的编程模型提供相同的集群信息。有关可以从 PowerHA SystemMirror 中获得的 SNMP 信息的更多信息或 Clinfo API 实现的详细信息，请参阅『实现细节』。

相关参考:

第 100 页的『实现细节』

有两个对 Clinfo 很重要的组件: **clinfo** 守护程序和 API 库。

入门

安装 PowerHA SystemMirror 时，将安装并配置 Clinfo API 及其相关联的组件。

在开始使用 Clinfo 之前，有一些需要考虑的事项:

1. 为了使用 Clinfo，必须指定在启动 PowerHA SystemMirror 集群服务时启动 Clinfo 代理程序。这是 `smitty clstart` 中的**启动集群信息守护程序**选项。
2. Clinfo 将识别并使用非公共名称的 SNMP 共用名。如果您的安装版本正在使用非公共名称，或者您要指定特定名称，请参阅『实现细节』。
3. Clinfo 将在安装了 PowerHA SystemMirror 的每个节点上运行。它也可以在其他机器上运行，前提条件是这些机器与其中一个 PowerHA SystemMirror 节点具有 TCP/IP 连接。有关此功能的其他信息，请参阅『实现细节』。

相关参考:

第 100 页的『实现细节』

有两个对 Clinfo 很重要的组件: **clinfo** 守护程序和 API 库。

Clinfo API

应用程序通过 Clinfo API 函数访问集群信息。开发者可以使用 Clinfo C API 或 Clinfo C++ API 来访问集群状态信息，然后，运行 Clinfo 程序的客户机可在本地获得此信息。

PowerHA SystemMirror for AIX 提供了两种版本的 Clinfo C 和 C++ API 库: 一种是用于单线程（无线程）应用程序的版本（**libcl.a** 和 **libclpp.a**），一种是用于多线程应用程序的版本（**libcl_r.a** 和 **libclpp_r.a**）。请确保与适合于应用程序的版本进行链接。**libcl_r.a** 是 **libcl.a** 的线程安全版本; **libclpp_r.a** 是 **libclpp.a** 的线程安全版本。

其中每个库都包含 32 位和 64 位对象，将根据 AIX 操作环境在运行时装入这些对象。

请参阅『Clinfo C API』和『Clinfo C++ API』，以了解这些 API 中例程的详细描述。

相关概念:

第 9 页的『Clinfo C API』
Clinfo C 应用程序编程接口 (API) 是高级接口，您可以在应用程序中使用该接口来获取有关 PowerHA SystemMirror 集群的状态信息。这些主题描述了 Clinfo C API 中可用的特定 C 语言例程和实用程序。

第 65 页的『Clinfo C++ API』
Clinfo C++ API 是面向对象的接口，您可以在 C++ 应用程序中使用该接口来获取有关 PowerHA SystemMirror for AIX 集群的状态信息。这些主题描述了 Clinfo C++ API 中可用的特定 C++ 语言对象和方法。

由 Clinfo 跟踪的事件

Clinfo 从集群管理器接收有关集群事件的状态信息。API 中的例程可以访问此信息。当集群经历各种状态时，Clinfo 将跟踪拓扑事件。

Clinfo 跟踪的某些状态包括:

- 集群状态为已启动或已关闭
- 集群子状态已变为稳定或不稳定
- 应用程序已联机，或者已故障转移到备份节点
- 网络已失败
- 节点正在加入集群
- 节点已完成加入集群的过程
- 节点正在脱离集群（即，节点已失败）
- 节点已脱离集群
- 已选定新的主集群管理器（可选事件）
- 集群站点（如果配置了该站点）的状态更改。

可以在本文档的后续章节或者已编译到应用程序中的 **clinfo.h** 包含文件中找到完整的事件列表。

Clinfo 接收动态重新配置事件，但不会跟踪这些事件; 即，应用程序无法注册为要接收动态重新配置事件的通知。Clinfo 在接收到动态重新配置事件时将集群子状态设置为 CLSS_RECONFIG。应用程序可以通过使用 **cl_getcluster** 例程获取此信息。动态重新配置所触发的事件（例如节点启动或节点关闭事件）是应用程序可以看到的事件。

由 Clinfo 跟踪的集群信息

某些集群信息由 Clinfo 进行维护。

集群

PowerHA SystemMirror 集群是一组处理器，这些处理器互相配合以提供高可用性环境。

可获得的集群信息

Clinfo 将维护有关已配置集群的以下信息:

集群名称

集群名称将唯一地标识集群。在配置集群时，管理员将指定该名称。

集群标识

集群标识将标识每个集群。集群标识是配置集群时由 PowerHA SystemMirror 分配的数字值（此值不是用户定义的值）。

集群状态

集群可以处于下列其中一种已定义状态:

项	描述
CLS_UP	集群中至少有一个节点已启动，并且定义了主节点。
CLS_DOWN	任何节点都未启动。
CLS_UNKNOWN	Clinfo 无法或者尚未与任何活动集群上的 SNMP 进程通信。
CLS_ NOTCONFIGURED	尚未配置集群。

集群子状态

集群可以处于若干已定义子状态之一:

项	描述
CLSS_ERROR	发生错误，需要进行手动干预。
CLSS_RECONFIG	正在对集群进行动态重新配置。
CLSS_STABLE	集群稳定（未在进行重新配置）。
CLSS_UNSTABLE	集群不稳定。事件历史记录将显示正在发生哪些事件。
CLSS_UNKNOWN	Clinfo 无法与集群节点上的 SNMP 进程通信
CLSS_ NOTCONFIGURED	尚未配置集群。
CLSS_NOTSYNCHED	某些基本集群配置信息存在，但是尚未对集群进行验证和同步。

主节点名称

指定主节点是先前发行版中不推荐使用的功能所遗留的概念。它当前不具有任何功能，并且不会影响行为。保留此概念仅为了支持二进制兼容性。

集群中的节点数

集群中定义的节点数。

集群中的网络数

集群中网络的总数。

集群中的资源组数

已配置的资源组的总数。

集群中的站点数

如果正在使用站点，那么此项指的是已配置的站点数。

节点

节点是组成集群的处理器之一。集群中的每个节点运行 **clstrmgr**、**clinfo** 和 **clsmuxpd** 守护程序。

可获得的节点信息

Clinfo 将维护有关节点的以下信息:

集群标识

此节点所属的集群的标识。

节点名

节点名是用户分配的字符串。节点名最多可以包含 32 个字符，不能以前导数字开头。

节点状态

节点可以处于下列其中一种已定义状态:

项	描述
CLS_UP	节点已启动并正在运行
CLS_DOWN	节点已关闭
CLS_JOINING	节点正在加入集群
CLS_LEAVING	节点正在脱离集群。

网络接口

已连接至节点的服务接口的数目和地址。

网络接口

网络接口是节点与网络之间的物理连接。

PowerHA SystemMirror 集群可以支持多个网络和点到点连接，并支持 RS232 串行线路点到点连接。每个网络将在每个集群节点上具有一个或多个网络接口。

可获得的网络接口信息

Clinfo 将维护有关网络接口的以下信息:

集群标识

此接口所属的集群的标识。

节点名

此接口所连接至的节点的名称。

活动节点标识

地址当前在其中处于活动状态的节点的标识。

接口名称

接口的名称与该接口在 **/etc/hosts** 文件中的名称（即，与主机的 IP 地址相关联的名称）相同。

接口标识

此接口所连接至的网络的网络标识。

接口地址

此接口的 IP 地址，在 **/etc/hosts**文件中进行定义。

接口状态

接口可以处于若干已定义状态之一。下列值描述了网络接口的状态:

项	描述
CLS_UP	接口已启动并正在运行。
CLS_DOWN	网络接口或网络已关闭。
CLS_INVALID	没有为此节点定义此接口。

接口角色

接口可以具有若干已定义角色之一。下列值描述了网络接口的角色:

项	描述
CL_INT_ROLE_INVALID	网络接口无效。
CL_INT_ROLE_SERVICE	网络接口已定义为服务接口。
CL_INT_ROLE_STANDBY	不推荐使用该网络接口角色。
CL_INT_ROLE_BOOT	网络接口已定义为服务接口。
CL_INT_ROLE_SH_SERVICE	不推荐使用该网络接口角色。

资源组

资源组包含与应用程序的实例相关联的所有资源，由 PowerHA SystemMirror 确保高可用性。

当节点加入集群时，将使资源组联机。发生故障时，将在集群节点之间移动资源组。可以通过 Clinfo 获得组的状态和位置。

可获得的资源组信息

Clinfo 将维护有关资源组的以下信息:

集群标识

此资源组所属的集群的标识。

组名

这是首次定义资源组时对其指定的名称。

组标识

与组相关联的数字标识。

组启动策略

用于此资源组的启动策略:

- 仅在主节点上联机
- 在第一个可用节点上联机
- 在所有可用节点上联机
- 使用分发策略联机

组故障转移策略

此资源组的故障转移策略:

- 故障转移到列表中的下一个优先级节点
- 使用动态节点优先级进行故障转移
- 脱机（仅在错误节点上）

组回退策略

此资源组的回退策略:

- 回退到列表中的较高优先级节点
- 从不回退

组站点策略

当组包含复制资源时，下列策略将用于资源组启动、故障转移和回退（如果它发生在两个站点之间）:

- 首选主站点
- 在任何一个站点上联机
- 在两个站点上都联机

节点数

参与资源组的节点数。

组节点标识

参与资源组的所有节点的节点标识。

组节点状态

资源组在每个节点上的状态。

资源组状态

资源组可以在一个或多个集群节点上处于下列其中一种状态:

项	描述
CL_RGNS_INVALID	节点不是此资源组的一部分。
CL_RGNS_ONLINE	组及其所有资源都已在节点上启动并可用。
CL_RGNS_OFFLINE	组在节点上当前未处于活动状态。
CL_RGNS_ACQUIRING	正在此节点上获取组。
CL_RGNS_RELEASING	正在从此节点释放组。
CL_RGNS_ERROR	尝试使组联机或脱机时发生错误。此组的资源未处于活动状态。需要进行手动干预。

注: 此列表并非包括了所有可能的资源组状态: 如果定义了站点，那么资源组的主要实例和次要实例可以处于联机状态、脱机状态、错误状态或未受管理状态。此处，资源组实例还可以处于正在获取或正在释放状态。此处未列示相应的资源组状态，但是提供了说明所采取的操作的描述性名称。

集群网络

PowerHA SystemMirror 集群可以同时包含 TCP/IP 网络和非基于 IP 的网络。通常，非基于 IP 的网络用于磁盘脉动流量。

可获得的网络信息

Clinfo 守护程序将提供有关集群中网络的下列信息:

网络名

与网络相关联的名称。您可以提供此名称，或者 PowerHA SystemMirror 可以生成此名称。

网络标识

由 PowerHA SystemMirror 生成的数字网络标识。

网络类型

网络的物理类型，例如以太网。

网络属性

基于 IP 的网络的网络属性可以设置为 Public 或 Private。将该属性设置为 Private 时，标识此网络供 Oracle 用作专用网络。缺省值为 Public。

网络节点信息

对于已连接至网络的每个集群节点，将提供下列信息:

项	描述
节点标识	每个节点的节点标识。
节点状态	网络在节点上的状态。此值可能与全局网络状态不相同。

网络状态

网络状态（包括全局状态和按节点的状态）可以是若干预定义值之一:

项	描述
CLS_UP	网络已启动。如果网络已在任何节点上启动，那么全局状态也会设置为已启动。
CLS_DOWN	网络未启动。网络可以在一个或多个节点上处于已关闭状态（即，按节点的状态为 **CLS_DOWN**），而仍然具有全局状态 **CLS_UP**。

集群站点

配置站点时，集群中的节点将按集群站点进行分组。将跟踪集群站点的状态，并且将生成集群事件。

可获得的集群站点信息

Clinfo 将维护有关集群站点的以下信息:

站点标识

PowerHA SystemMirror 为站点生成的标识。

站点名称

创建站点时指定的站点名称。

站点优先级

站点优先级确定站点之间数据镜像的优先顺序和方向。集群站点可以具有下列其中一个优先级:

项	描述
CL_SITE_PRIMARY	这是运行应用程序的站点。数据将从此站点镜像至备份站点。
CL_SITE_SECONDARY	此站点充当备份站点。
CL_SITE_TERTIARY	此站点正在对从辅助站点发送的数据进行镜像。保留此值供将来使用。

站点备份方法

这是站点的备份通信方法。如果配置了备份方法，那么它可以是下列其中一项:

项	描述
CL_SITE_BACKUP_DBFS	回拨失败安全
CL_SITE_BACKUP_SGN	串行全局网络
CL_SITE_BACKUP_NONE	未配置备份通信

站点状态

站点的全局状态:

项	描述
CLS_UP	站点中的一个或多个节点已启动
CLS_DOWN	站点中的所有节点都已关闭

站点中的节点数

此站点中节点的数目。

站点节点标识

参与此站点的节点标识的列表。

Clinfo C API

Clinfo C 应用程序编程接口 (API) 是高级接口，您可以在应用程序中使用该接口来获取有关 PowerHA SystemMirror 集群的状态信息。这些主题描述了 Clinfo C API 中可用的特定 C 语言例程和实用程序。

注: 请使用应用程序监视来代替 **cl_registerwithclsmuxpd()** 例程。请参阅《规划指南》中有关初始集群规划的部分。

在阅读本主题之前，您应该阅读『集群信息程序』，它描述了 Clinfo 所维护的有关 PowerHA SystemMirror 集群的信息的类型。

相关概念:

第 1 页的『集群信息程序』
这些主题概述了集群信息程序 (Clinfo)，并描述了 Clinfo 接收和维护的有关 PowerHA SystemMirror for AIX 集群的状态信息。

相关信息:

初始集群规划

在应用程序中使用 Clinfo C API

这些主题描述如何在应用程序中使用 Clinfo C API。

PowerHA SystemMirror for AIX 为多线程应用程序和单线程应用程序提供了不同库。请确保与适合于应用程序的库进行链接。

注: Clinfo 的 **cluster.es.client.lib** 库包含具有 32 位和 64 位对象的 **libcl.a**。必须在 64 位环境中重新编译/重新链接应用程序，才能获取使用 Clinfo API 的 64 位应用程序。

头文件

必须在每个使用 Clinfo C API 的源模块中指定必要的 **include** 伪指令。

这些伪指令包括:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <cluster/clinfo.h>
```

除了此 **include** 伪指令列表外，还请在每个使用 **cl_registereventnotify** 例程的源模块中指定以下 **include** 伪指令:

```
#include <signal.h>
```

针对使用客户机 API 的应用程序的编译器预处理器伪指令

编译使用客户机 API 的应用程序时，请使用编译器标志 —**D__HAES__**。

需要此标志是因为头文件中使用了 **#ifdef** 预处理器伪指令。这是由于先前支持另一个 PowerHA SystemMirror 版本 (HAS)。

链接 libcl.a 和 libcl_r.a 库

必须将必要的伪指令添加到使用 Clinfo C API 的*单线程*应用程序的对象装入命令:

这些伪指令包括:

```
-lcl
```

必须将以下伪指令添加到使用 Clinfo C API 的*多线程*应用程序的对象装入命令:

```
-lcl_r
```

libcl.a 和 **libcl_r.a** 库包含支持 Clinfo C API 的例程。

常量

Clinfo C API 例程使用 **clinfo.h** 文件中定义的常量。

这些常量包括:

项	描述
CL_MAXNAMELEN	用于对集群、节点或接口进行命名的字符串的最大长度 (256)。**CL_MAXNAMELEN** 的值与 **sys/param.h** 文件中定义的 **MAXHOSTNAMELEN** 相同。
CL_MAX_EN_REQS	允许的最大事件通知请求数 (10)。
CL_ERRMSG_LEN	最大错误消息长度（128 个字符）。
CL_MAXCLUSTERS	提供数据结构中数组的当前空间大小，该数组用于存放有关集群数的信息。
CL_MAXNODES	提供数据结构中数组的当前空间大小，该数组用于存放有关集群中节点数的信息。
CL_MAXNETIFS	提供数据结构中数组的当前空间大小，该数组用于存放有关已连接至节点的接口数的信息。
CL_MAXGROUPS	提供数据结构中数组的当前空间大小，该数组用于存放有关资源组数的信息。

数据类型和结构

Clinfo C API 使用 **clinfo.h** 文件中定义的不同数据类型和结构。

包含状态信息的枚举类型:

此枚举数据类型描述集群、节点、接口或事件通知的状态

```
enum cls_state {
    CLS_INVALID,
    CLS_VALID,
    CLS_UP,
    CLS_DOWN,
    CLS_UNKNOWN,
    CLS_GRACE,
    CLS_JOINING,
    CLS_LEAVING,
    CLS_IN_USE,
    CLS_PRIMARY
};
```

包含子状态信息的枚举类型:

此枚举数据类型描述集群的子状态。

```
enum cls_substate {
    CLSS_UNKNOWN,
    CLSS_STABLE,
    CLSS_UNSTABLE,
    CLSS_ERROR,
    CLSS_RECONFIG
    CLSS_NOT_CONFIGURED
};
```

包含资源组状态的枚举类型:

此枚举数据类型包含资源组在节点上可以处于的所有状态。

```
enum cl_resource_states{
    CL_RGNS_INVALID=1,
    CL_RGNS_ONLINE=2,
    CL_RGNS_OFFLINE=4,
    CL_RGNS_ACQUIRING=16,
    CL_RGNS_RELEASING=32,
    CL_RGNS_ERROR=64
};
```

包含资源组策略的枚举类型:

此枚举数据类型包含所有资源组策略（节点和站点）。

```
enum cl_rg_policies {
    CL_RGP_INVALID = 0,
    CL_RGP_ONLINE_ON_HOME_NODE = 1,
    CL_RGP_ONLINE_ONFIRST_AVAILBLE_NODE = 2,
    CL_RGP_ONLINE_USING_DISTRIBUTION_POLICY = 3,
    CL_RGP_ONLINE_ALL_NODES = 4,
    CL_RGP_FALLOVER_TO_PRIORITY_NODE = 5,
    CL_RGP_FALLOVER_USING_DNP = 6,
    CL_RGP_BRING_OFFLINE = 7,
    CL_RGP_FALLBACK_TO_HIGHER_PRIORITY_NODE = 8,
    CL_RGP_NEVER_FALLBACK = 9,
    CL_RGP_PREFER_PRIMARY_SITE = 10,
    CL_RGP_ONLINE_ON_EITHER_SITE = 11,
    CL_RGP_ONLINE_ON_BOTH_SITES = 12,
    CL_RGP_IGNORE_SITES = 13
};
```

包含接口角色的枚举类型:

此枚举类型包含接口角色。

```
enum cl_interface_role {
    CL_INT_ROLE_INVALID = 0,
    CL_INT_ROLE_SERVICE = 16,
    CL_INT_ROLE_STANDBY = 32,      /* deprecated */
    CL_INT_ROLE_BOOT = 64,
    CL_INT_ROLE_SH_SERVICE = 128,  /* deprecated */
};
```

包含网络属性的枚举类型:

此枚举数据类型指定网络的属性。

```
/*
* Enumeration of Network attributes. Note that the public/private
attribute is for use by Oracle only - PowerHA SystemMirror does not use this attribute.
```

```
*/

enum cl_network_attribute
{
    CL_NET_ATTR_INVALID = 0,   /* IP networks can be public or private */
    CL_NET_TYPE_PUBLIC = 1,
    CL_NET_TYPE_PRIVATE = 2,   /* non-IP (serial) networks are always */
    CL_NET_TYPE_SERIAL = 4
};
```

包含站点优先级的枚举类型:

此枚举数据类型描述站点的优先级。处理资源时，站点将定义为具有优先级。

```
/*
* Enumeration of Site Priorities
*/

enum cl_site_priority
{
    CL_SITE_PRI_NONE = 0,
    CL_SITE_PRI_PRIMARY = 1,
    CL_SITE_PRI_SECONDARY = 2,
    CL_SITE_PRI_TERTIARY = 4
};
```

包含站点备份通信方法的枚举类型:

此枚举数据类型包含可以为站点配置的可选备份方法。

```
/*
* Enumeration of Site Backup Communication Options
*/

enum cl_site_backup
{
    CL_SITE_BACKUP_NONE = 0,
    CL_SITE_BACKUP_DBFS = 1,
    CL_SITE_BACKUP_SGN = 2
};
```

表示资源组的数据结构:

此结构包含每个资源组的所有可获得的信息。

```
struct cl_group {
    int  clg_clusterid;
    int  clg_group_id;
    char clg_name[CL_MAXNAMELEN];
    enum cl_rg_policies clg_policy;   /* deprecated */
    enum cl_rg_policies clg_startup_policy;
    enum cl_rg_policies clg_fallover_policy;
    enum cl_rg_policies clg_fallback_policy;
    enum cl_rg_policies clg_site_policy;
    char clg_user_policy_name[CL_MAXNAMELEN];
    int  clg_num_nodes;
    int  clg_node_ids[MAXNODES];
        /* list of nodes (ids) in this group */
    enum cl_resource_states clg_node_states[MAXNODES];
        /* state on each */
    int  clg_num_resources;
    int  clg_resource_id[MAXRESOURCES];
        /* list of resources (id) group */
    enum cl_resource_states clg_res_state[MAXRESOURCES]; /* state
*/ int clg_vrmf;                        /* version of this client */
};
```

表示网络接口的数据结构:

此数据结构表示网络接口。

```
struct cl_netif {
    int cli_clusterid;                  /* Cluster Id */
    int cli_nodeid;                     /* Cluster node Id - used internally only */
    char cli_nodename[CL_MAXNAMELEN];   /* Cluster node name */
    int cli_interfaceid;                /* Cluster Node Interface Id */
    enum cls_state cli_state;           /* Cluster Node Interface State */
    char cli_name[CL_MAXNAMELEN];       /* Cluster Node Interface Name */
    int cli_active_nodeid;              /* Cluster node Id where addr is up */
    enum cl_interface_role cli_role;    /* Role of interface (boot/service)*/
    int cli_networkid;                  /* Cluster Network ID for this Interface */
    int cli_vrmf;
    struct sockaddr_storage cli_addr_v6; /*Cluster Node Interface IP Address */
};
#define cli_addr (*((struct sockaddr_in*)&cli_addr_v6)) /* For backward compatibility*/
```

表示节点的数据结构:

此数据结构表示集群节点。

```
struct cl_node {
    int cln_clusterid;                  /* Cluster Id */
    int cln_nodeid;                     /* Cluster node Id */
    char cln_nodename[CL_MAXNAMELEN];   /* Cluster node name */
    enum cls_state cln_state;           /* node state */
    int cln_nif;                        /* number of interfaces */
    struct cl_netif *cln_if;            /* interfaces */
    int cln_glidle;                     /* CPU.glidle */
    int cln_real_mem_free;              /* Paging space utilitization */
    int cln_disk_busy;                  /* disk busy */
    int cln_vrmf;                       /* version of this client */
};
```

表示集群的数据结构:

此数据结构表示集群。

```
struct cl_cluster{
    int  clc_clusterid;               /* cluster id*/
    enum cls_state clc_state;         /* Cluster State */
    enum cls_substate clc_substate;   /* Cluster Substate */
    char clc_primary[CL_MAXNAMELEN];  /* Cluster Primary Node */
    char clc_name[CL_MAXNAMELEN];     /* Cluster Name */
    int  clc_number_of_nodes;         /* number of cluster nodes */
    int  clc_number_of_groups;        /* number of resource groups */
    int  clc_number_of_networks;      /* number of networks */
    int  clc_number_of_sites;         /* number of sites */
    int clc_vrmf;                     /* version of this client */
};
```

表示事件通知注册请求的数据结构:

此数据结构表示事件通知注册请求。

```
struct cli_enr_req_t {
    int event_id;                   /* event id */
    int cluster_id;                 /* cluster id */
    int node_id;                    /* node id(internal use only)*/
    char node_name[CL_MAXNAMELEN];  /* node name */
```

```
    int net_id;                     /* network id */
    int signal_id;                  /* signal id */
    int vrmf;
};
```

表示事件通知消息的数据结构:

此数据结构表示事件通知消息。

```
struct cli_en_msg_t {
    int event_id;                 /* event id */
    int cluster_id;               /* cluster id */
    int node_id;                  /* node id(internal use only)*/
    char node_name[CL_MAXNAMELEN]; /* node name */
    int net_id;                   /* network id */
    int vrmf;<
};
```

表示集群网络的数据结构:

此数据结构包含每个集群网络的信息。

```
/*
* Structure containing information relating to a network.
*/
struct cl_net {
    int clnet_clusterid;                  /* Cluster Id */
    char clnet_name[CL_MAXNAMELEN];       /* Cluster network name */
    int clnet_id;                         /* Cluster Network Id */
    char clnet_type[CL_MAXNAMELEN];       /* ether, token, etc */
    enum cl_network_attribute clnet_attr; /* public/serial */
    enum cls_state clnet_state;           /* Cluster Network State */
      /* Note that this is the cluster wide or "global" network state */
      /* which may be different than the state of the network on any */
      /* particular node */
    int clnet_numnodes;
      /* Number of nodes connected to this Network */
    int clnet_node_ids[MAXNODES];
      /* Node ids connected to this Network */
    enum cls_state clnet_node_states[MAXNODES];
      /* Network State per Node */
    int clnet_vrmf;                          /* version of this client */
    enum cl_net_family clnet_type;          /* v4/v6 */
};
enum cl_net_family {
  CL_INET_INVALID = 0,
  CL_INET4 = 1,
  CL_INET6 = 2,
};
```

表示集群站点的数据结构:

此数据结构包含 PowerHA SystemMirror 集群中配置的每个站点的信息。注意，站点配置是可选的。

```
/*
* Structure containing information relating to a site
*/
struct cl_site {
    int clsite_clusterid;             /* Cluster ID */
    int clsite_id;
    char clsite_name[CL_MAXNAMELEN];
    enum cl_site_priority clsite_priority;
    enum cl_site_backup clsite_backup;
    enum cls_state clsite_state;      /* Cluster Site State */
```

```
    int clsite_numnodes;
    int clsite_nodeids[MAXNODES];     /* List of nodes (IDs) in this group */
    int clsite_vrmf;                  /* version of this client */
};
```

将应用程序从更低 Clinfo 发行版升级

在先前的 PowerHA SystemMirror 发行版中，将手动配置集群标识；现在会自动创建集群标识。

给定了集群名称或其他参数时，存在若干将返回集群标识的调用:

- **cl_getclusterid** - 给定了集群名称时，将返回集群标识
- **cl_getclusteridbyifaddr** - 给定了网络接口地址时，将返回集群标识
- **cl_getclusteridbyifname** - 给定了网络接口名称时，将返回集群标识。

如果应用程序使用需要集群标识的 API 调用，那么必须添加对这些返回集群标识的例程之一的调用。更低的 Clinfo 发行版使用整数（节点标识）而不是字符串（节点名）来标识集群节点。下列各节提供了一些示例，以说明如何转换应用程序中对先前使用了 *nodeid* 参数的各种 Clinfo C API 例程的调用。对于每个例程，在它与节点标识配合使用的示例后面都有一个使用节点名的示例。

相关参考:

第 31 页的『cl_getclusterid 例程』
cl_getclusterid 例程将返回具有所指定名称的集群的集群标识。

第 32 页的『cl_getclusteridbyifaddr 例程』
返回具有所指定网络接口地址的集群的集群标识。此例程只能处理 IPv4 地址。

第 33 页的『cl_getclusteridbyifname 例程』
返回具有所指定网络接口名称的集群的集群标识。

cl_getlocalid

这些是来自于更低发行版和更高版本的 **cl_getlocalid** 例程示例。

以下是来自于使用节点标识的更低发行版的 **cl_getlocalid** 例程示例:

```
int clusterid, nodeid, status;
status = cl_getlocalid (&clusterid, &nodeid);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
 printf ("This node is not a cluster member");
    } else {
 cl_perror (status, "Can't get local cluster ID");
 }
} else {
    printf ("member of cluster %d node %d",clusterid, nodeid);
}
```

在新版本的 C API **cl_getlocalid** 例程示例中，请注意声明语句中的更改。在先前已将所有变量声明为整数的情况下，现在必须将 *nodename* 声明为字符串，并对 **printf** 语句进行相应的更改。

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
status = cl_getlocalid (&clusterid, nodename);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
 printf ("This node is not a cluster member");
    } else {
 cl_perror (status, "Can't get local cluster ID");
```

```
    }
} else {
    printf ("member of cluster %d node %s",clusterid, nodename);
}
```

cl_getnodeidbyifname

这些是来自于更低发行版和更高版本的 **cl_getnodeidbyifname** 例程示例。

以下是来自于使用节点标识的更低发行版的 **cl_getnodeidbyifname** 例程示例:

```
int clusterid, nodeid;
char *interfacename;
strcpy (interfacename,"editserver");
clusterid = 1;
nodeid = cl_getnodeidbyifname (clusterid, interfacename);
if (nodeid < 0) {
    cl_perror(nodeid,"Can't get node ID");
} else {
    printf("ID of %s on cluster %d is %d", interfacename, clusterid, nodeid);
}
```

在新版本的 C API **cl_getnodeidbyifname** 例程示例中，请注意例程本身的名称已更改为 **cl_getnodenamebyifname**。您必须更改声明以包括字符串形式的 *nodename*，而不是包括整数形式的 *nodeid*; 然后对 **printf** 语句进行相应的更改。

```
int clusterid, status;
char nodename[MAXNAMELEN];
char *interfacename[MAXNAMELEN];
strcpy (interfacename,"editserver");
clusterid = 1;
status = cl_getnodenamebyifname (clusterid, interfacename, nodename);
if (status != CLE_OK)
    cl_perror(nodename,"Can't get node name");
} else {
    printf("name of %s on cluster %d is %s", interfacename, clusterid, nodename);
}
```

cl_getprimary

这些是来自于更低发行版和更高版本的 **cl_getprimary** 例程示例。

以下是来自于使用节点标识的更低发行版的 **cl_getprimary** 例程示例:

```
int clusterid, primary; /* clusterid is arbitrary.*/
clusterid = 1;
primary = cl_getprimary (clusterid);
if (primary < 0) {
    cl_perror (primary, "Can't get cluster primary");
} else {
    printf ("Primary node for cluster %d is %d", clusterid, primary);

}
```

在新版本的 C API **cl_getprimary** 例程示例中，请注意声明中的更改（字符串形式的 *nodename* 代替了整数形式的 *nodeid*）以及对 **printf** 语句的相应更改。此外，还必须更改 if 语句，这是因为先前版本依赖于期望的信息 (*nodeid*) 是整数这一事实；现在，此信息是字符串 (*nodename*)。

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
/* clusterid is arbitrary. */
clusterid = 1;
status = cl_getprimary (clusterid, nodename);
if (status != CLE_OK) {
```

```
    cl_perror (status, "Can't get cluster primary");
} else {
    printf ("Primary node for cluster %d is %s", clusterid, nodename);
}
```

cl_isaddravail

这些是来自于更低发行版和更高版本的 **cl_isaddravail** 例程示例。

以下是来自于使用节点标识的更低发行版的 **cl_isaddravail** 例程示例:

```
int clusterid, nodeid, status;
struct sockaddr_in addr;      /* clusterid, nodeid, and addr are arbitrary.*/
clusterid = nodeid = 1;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_isaddravail (clusterid, nodeid, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Interface not available");
} else {
    printf ("Interface address for %s is available", inet_ntoa
    (addr.sin_addr.s_addr));
}
```

在新版本的 C API **cl_isaddravail** 例程示例中，声明语句已更改为使用 *nodename* 而不是 *nodeid*。请参阅对 **printf** 语句的相应更改。赋值语句已更改为将 strcpy 用于 *nodename*。

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
struct sockaddr_in addr;    /* clusterid, nodename, and addr are arbitrary. */
clusterid = 1;
strcpy (nodename, "node1");
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_isaddravail (clusterid, nodename, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Interface not available");
} else {
    printf ("Interface address for %s is available", inet_ntoa
    (addr.sin_addr.s_addr));
}
```

内存分配例程

这些例程将分配或释放用来存储集群或节点映射信息的内存。

在调用对应的检索例程之前，必须调用适当的内存分配例程，然后在完成检索例程时，调用释放例程来释放存储器。

项	描述
cl_alloc_clustermap	为集群列表（4 个集群）分配存储器，每个集群具有 32 个节点，每个节点具有 32 个接口。
cl_alloc_groupmap	为资源组列表（64 个资源组）分配存储器。
cl_alloc_netmap	为网络列表分配存储器。
cl_alloc_nodemap	为节点列表（32 个节点）分配存储器，每个节点具有 32 个接口。
cl_alloc_sitemap	为站点列表分配存储器。
cl_free_clustermap	释放已通过调用 **cl_alloc_clustermap** 为集群列表分配的存储器。
cl_free_groupmap	释放已使用 **cl_alloc_groupmap** 为资源组列表分配的存储器。
cl_free_netmap	释放已通过调用 **cl_alloc_netmap** 为网络列表分配的存储器。
cl_free_nodemap	释放已通过调用 **cl_alloc_nodemap** 为节点列表分配的存储器。
cl_free_sitemap	释放已通过调用 **cl_alloc_sitemap** 为站点网络列表分配的存储器。

以下示例说明了如何使用这些例程。注意，请不要再使用 CL_MAXNODES 来分配存储器以存放返回的有关集群中节点的信息。有关其他示例，请参阅 **cl_getclusters** 和 **cl_getnodemap** 例程的参考页面。

```
int status;
struct cl_cluster *clustermap;

cl_alloc_clustermap (&clustermap);
status = cl_getclusters(clustermap);
if (status < 0) {
    cl_perror(status, "Can't get cluster information");
} else {
    printf("There are currently %d running clusters", status);
}
...
cl_free_clustermap (clustermap);
```

有一个额外的新 API **cl_node_free()**，它释放与单个 **cl_node** 结构相关联的存储器。请考虑以下示例:

```
struct cl_node nodebuf;
    cl_getnode(clusterid, "ppstest5", &nodebuf);
    printf("Node %s is id %d\n", nodebuf.cln_nodename, nodebuf.cln_nodeid);
    cl_node_free(&nodebuf);
);
```

在调用 **cl_getnode()** 之后，将使用与节点相关联的网络接口结构的列表来填写 cln_if 字段。此列表由 **cl_getnode()** 动态分配，必须释放此列表，以避免在长时间运行的程序中发生内存泄漏。

cl_node_free() 将释放 **cl_node** 结构的网络接口存储器字段。请参阅 **cl_node_free()** 的后续 API 描述以了解更多信息。

请求

Clinfo C API 具有若干类型的请求。

集群信息请求

集群信息请求将返回有关集群的信息。

项	描述
cl_getcluster	返回有关具有所指定集群标识的集群的所有已知信息。
cl_getclusterid	返回具有所指定集群名称的集群的集群标识。
cl_getclusteridbyifaddr	返回具有所指定网络接口地址的集群的集群标识。此例程只能处理 IPv4 地址。
cl_getclusteridbyifaddr6	返回具有所指定网络接口地址的集群的集群标识。此例程能够处理 IPv4 地址和 IPv6 地址。
cl_getclusteridbyifname	返回具有所指定网络接口名称的集群的集群标识。
cl_getclusters	返回有关所有工作集群的信息。
cl_isclusteravail	返回具有所指定集群标识的集群的状态。

节点信息请求

节点信息请求将返回有关集群中节点的信息。

项	描述
cl_bestroute	返回一对本地或远程 IP 地址，这一对地址指示从本地机器到所指定节点的最直接路由。
cl_bestroute6	返回一对本地或远程 IP 地址，这一对地址指示从本地机器到所指定节点的最直接路由。此例程能够处理 IPv4 地址和 IPv6 地址。
cl_getlocalid	返回发出请求的主机的集群标识和节点名。
cl_getnode	返回有关由一对集群标识或节点名指定的节点的信息。
cl_getnodeaddr	返回与一对指定的集群标识或节点名相关联的 IPv4 地址。
cl_getnodeaddr6	返回与一对指定的集群标识/节点名相关联的 IPv4 地址或 IPv6 地址。
cl_getnodenamebyifaddr	返回具有一对所指定集群标识或接口地址的节点的名称。此例程仅返回 IPv4 地址。
cl_getnodenamebyifaddr6	返回具有一对所指定集群标识或接口地址的节点的名称。此例程能够处理 IPv4 地址和 IPv6 地址。
cl_getnodenamebyifname	返回具有一对所指定集群标识/接口名称的节点的名称。
cl_getnodemap	返回有关所指定集群中所有节点的所有已知信息。
cl_getprimary	返回所指定集群的主集群管理器的节点名。
cl_isnodeavail	返回所指定节点的状态。

网络接口信息请求

网络接口信息请求将返回有关已连接至节点的接口的信息。

项	描述
cl_getifaddr	返回具有所指定集群标识和名称的接口的 IPv4 接口地址。
cl_getifaddr6	返回具有所指定集群标识和名称的接口的 IPv4 或 IPv6 接口地址。
cl_getifname	返回具有所指定集群标识和地址的 IPv4 接口的名称。
cl_getifname6	返回具有所指定集群标识和地址的 IPv4 或 IPv6 接口的名称。
cl_isaddravail	返回所指定网络接口的状态。此例程仅返回 IPv4 地址。
cl_isaddravail6	返回所指定网络接口的状态。此例程能够处理 IPv4 地址和 IPv6 地址。

网络信息请求

网络信息请求将返回有关集群中网络的信息。

项	描述
cl_getnetmap	返回有关集群中所有网络的所有已知信息
cl_getnet	返回有关特定网络的信息
cl_getnetbyname	返回有关已指定的特定网络的信息
cl_getnetsbytype	返回有关配置了所指定类型的任何网络的信息
cl_getnetsbyattr	返回有关配置了所指定属性的任何网络的信息
cl_getnetstatebynode	返回所指定网络在所指定节点上的状态

事件通知请求

事件通知例程将返回有关集群事件、节点事件或网络事件的信息。

项	描述
cl_getevent	在接收到事件信号时获取信息，并返回从 Clinfo 接收到的事件通知消息
cl_registereventnotify	向 Clinfo 注册事件通知请求列表，并在发生已注册的事件时返回一个信号到主调进程
cl_unregistereventnotify	向 Clinfo 注销事件通知请求列表

资源组信息请求

资源组信息请求将返回有关集群资源组的信息。

项	描述
cl_getgroupmap	返回有关所指定集群中所有资源组的所有已知信息
cl_getgroup	返回有关所指定集群中特定资源组的所有信息
cl_getgroupsbynode	返回所指定节点所隶属的所有资源组
cl_getgroupnodestate	返回所指定组在所指定节点上的状态

站点信息请求

站点信息请求将返回有关 PowerHA SystemMirror 集群中配置的站点的信息。

项	描述
cl_getsitemap	返回有关集群中所有站点的所有已知信息
cl_getsite	返回有关特定站点的信息
cl_getsitebyname	返回有关已指定的特定站点的信息
cl_getsitebypriority	返回有关配置了所指定优先级的任何站点的信息

实用程序

Clinfo C API 具有若干实用程序例程。

cl_initialize 例程

先前发行版中使用 **cl_initialize()** 例程来连接至共享内存。

cl_initialize() 例程不再具有任何功能，将始终返回 **CLE_OK**。提供 **cl_initialize()** 仅为了支持向后兼容性。请**不要**将此例程用于任何新程序。

cl_errmsg 例程

cl_errmsg 例程将获取由 Clinfo 返回的状态码，并返回表示该错误代码的文本。

语法

```
char *cl_errmsg (int status)
```

参数

项	描述
status	集群信息错误状态。

状态码

以 null 结束的错误字符串。

例如，字符串:

```
"Invalid status"
```

（如果 status 参数未描述有效的集群错误代码）。

示例

```
char *msg;
msg = cl_errmsg(CLE_BADARGS);
if (strcmp(msg, "Invalid status") != 0) {
    printf("CLE_BADARGS means %s", msg);
} else {
    printf("Can't lookup CLE_BADARGS");
}
```

cl_errmsg_r 例程

cl_errmsg_r 例程将获取由 Clinfo 返回的状态码，并返回表示该错误代码的文本。

这是 **cl_errmsg** 例程的线程安全版本。如果具有多线程应用程序，那么必须使用此例程。

语法

```
char *cl_errmsg_r (int status, char cbuf)
```

参数

项	描述
status	集群信息错误状态。
cbuf	用于存放所返回消息的存储器的长度必须至少为 **CL_ERRMSG_LEN**（足够存放 128 个字符）。

状态码

以 null 结束的错误字符串。

例如，字符串:

```
"Invalid status"
```

（如果 status 参数未描述有效的集群错误代码）。

示例

```
char *msg;
char cbuf[CL_ERRMSG_LEN];
msg = cl_errmsg_r(CLE_BADARGS, cbuf);
if (strcmp(msg, "Invalid status") != 0) {
printf("CLE_BADARGS means %s", msg);
} else {
printf("Can't lookup CLE_BADARGS");
}
```

cl_perror 例程

cl_perror 例程将一条描述所指定错误代码的消息写至标准错误。**cl_perror** 将提供的错误字符串置于错误消息前面，并将一个冒号置于错误消息后面。

语法

```
void cl_perror (int status, char *message)
```

例如，指定:

```
cl_perror(CLE_IVNODENAME, "Can't service this request");
```

会生成以下输出:

```
Can't service this request:Illeagle Node Name.
```

cl_perror 例程对于根据 Clinfo 请求所返回的状态码生成错误消息很有用。如果提供的状态不是有效的集群错误代码，那么 **cl_perror** 例程将写入以下字符串:

```
Error n
```

其中 n 是所指定状态码的值。

参数

项	描述
status	集群错误代码。
message	将位于状态描述前面的消息。

示例

```
struct cl_node nodebuf;
  int clusterid = 99999999;      /* invalid cluster id */
  int status;
  char nodename[CL_MAXNAMELEN];

  if ( (status = cl_getnode (clusterid, nodename, &nodebuf)) < 0 ) {
      cl_perror(status, "can't get node information");
  }
```

cl_alloc_clustermap 例程

cl_alloc_clustermap 例程将为集群列表分配存储器。在调用 **cl_getclusters** 例程之前，必须调用此例程。

在调用 **cl_getclusters** 例程之后，调用 **cl_free_clustermap** 例程以释放存储器。

语法

```
int cl_alloc_clustermap (struct cl_cluster **clustermap)
```

参数

项	描述
clustermap	集群映射的基指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对 clustermap 自变量指定了 NULL 指针。

示例

请参阅 cl_getclusters 例程的示例。

相关参考:

第 30 页的『cl_getcluster 例程』

cl_getcluster 将返回有关由集群标识指定的集群的信息。

cl_alloc_groupmap 例程

cl_alloc_groupmap 例程将为资源组描述符列表分配存储器。在调用 **cl_getgroups** 例程之前，必须调用此例程。

在调用 **cl_getgroups** 例程之后，请在完成时通过调用 **cl_free_groupmap** 例程释放存储器。

语法

```
int cl_alloc_groupmap (struct cl_group **groupmap);
```

参数

项	描述
groupmap	组映射的基指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对 groupmap 自变量指定了 NULL 指针。

示例

请参阅 cl_getgroupmap 例程的示例。

相关参考:

第 36 页的『cl_getgroupmap 例程』

cl_getgroupmap 例程将返回有关集群中资源组的信息。在调用此例程之前，应该调用 **cl_alloc_groupmap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_groupmap**。

cl_alloc_netmap 例程

分配用于存放一系列网络信息结构的存储器。

语法

```
int cl_alloc_netmap (struct cl_site **netmap)
```

参数

项	描述
netmap	指向结构 **cl_net** 指针的指针，将使用该结构指针来返回指向新分配的存储器的指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对 **netmap** 参数指定了 NULL 指针。

示例

请参阅 cl_getnetmap 例程的示例。

相关参考:

第 44 页的『cl_getnetmap 例程』
返回有关集群中所有网络的信息。

cl_alloc_netmap6 例程

分配用于存放一系列站点信息结构的存储器。

语法

```
int cl_alloc_netmap6(struct cl_net_v6** netmap)
```

参数

项	描述
netmap	指向结构 **cl_net_v6** 指针的指针，将使用该结构指针来返回指向新分配的存储器的指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。

cl_alloc_nodemap 例程

cl_alloc_nodemap 例程将为多个节点（或一组节点）以及与每个节点相关联的接口分配存储器。在调用 **cl_getnodemap** 例程之前，应该调用此例程。

在调用 **cl_getnodemap** 例程之后，请在完成时通过调用 **cl_free_nodemap** 例程释放存储器。

语法

```
int cl_alloc_nodemap (struct cl_node **nodemap)
```

参数

项	描述
nodemap	节点映射的基指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。

示例

请参阅 cl_getnodemap 例程的示例。

相关参考:

第 50 页的『cl_getnodemap 例程』

cl_getnodemap 例程将返回有关集群中节点的信息。在调用此例程之前，应该调用 **cl_alloc_nodemap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_nodemap**。

cl_alloc_nodemap6 例程

分配用于存放一系列站点信息结构的存储器。

语法

```
int cl_alloc_nodemap6(struct cl_node_v6** nodemap)
```

参数

项	描述
nodemap	指向结构 **cl_net_v6** 指针的指针，将使用该结构指针来返回指向新分配的存储器的指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。

cl_alloc_sitemap 例程

分配用于存放一系列站点信息结构的存储器。

语法

```
int cl_alloc_sitemap (struct cl_net **sitemap)
```

参数

项	描述
sitemap	指向结构 **cl_site** 指针的指针，将使用该结构指针来返回指向新分配的存储器的指针。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对 **sitemap** 参数指定了 NULL 指针。

示例

请参阅 cl_getsitemap 例程的示例。

相关参考:

第 57 页的『cl_getsitemap 例程』
返回有关集群中所有站点的所有已知信息。

cl_bestroute 例程

cl_bestroute 例程将返回一对本地/远程 IP 地址，这一对地址表示到所指定节点的最直接路由。此例程依赖于网络掩码值来与成对的网络接口匹配。此例程只能处理 IPv4 地址。

cl_bestroute 例程所返回的路由取决于发出请求的主机。Clinfo 首先构建一个由本地节点上的所有工作网络接口组成的列表，然后将此列表与所指定节点上的可用接口进行比较。该例程按网络对接口进行排序，并尝试将网络上的第一个工作接口与网络在远程节点上的第一个工作接口进行匹配。如果定义了专用网络，那么将先考虑这些网络，然后考虑非专用网络。

如果存在一对位于同一网络上的本地和远程接口，那么将在 **laddr** 和 **raddr** 参数中返回这些接口。否则，将选择所指定节点上的接口作为远程接口，并且将返回找到的第一个本地接口作为路由的本地端。

语法

```
int cl_bestroute (int clusterid, char *nodename,
struct sockaddr_in *laddr, struct sockaddr_in *raddr)
```

参数

项	描述
clusterid	期望节点的集群标识。
nodename	期望节点的名称。
laddr	用于存放所返回本地网络接口地址的存储器。
raddr	用于存放所返回远程网络接口地址的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。
CLE_NOROUTE	无可用路由。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
int clusterid = 1113325332;
  int status;
  char nodename[CL_MAXNAMELEN] = "node1";
  struct sockaddr_in laddr, raddr;

  status = cl_bestroute(clusterid, nodename, &laddr, &raddr);
  if (status != CLE_OK) {
      cl_perror(status, "can't get route");
  } else {
      printf("best route to node %s is from ", nodename);
      printf("%s to ", inet_ntoa(laddr.sin_addr));
      printf("%s\n", inet_ntoa(raddr.sin_addr));
  }
```

cl_bestroute6 例程

cl_bestroute6 例程将返回一对本地/远程 IP 地址，这一对地址表示到所指定节点的最直接路由。此例程依赖于网络掩码值来与成对的网络接口匹配。

cl_bestroute6 例程所返回的路由取决于发出请求的主机。Clinfo 首先构建一个由本地节点上的所有工作网络接口组成的列表，然后将此列表与所指定节点上的可用接口进行比较。该例程按网络对接口进行排序，并尝试将网络上的第一个工作接口与网络在远程节点上的第一个工作接口进行匹配。如果定义了专用网络，那么将先考虑这些网络，然后考虑非专用网络。

如果存在一对位于同一网络上的本地和远程接口，那么将在 **laddr** 和 **raddr** 参数中返回这些接口。否则，将选择所指定节点上的接口作为远程接口，并且将返回找到的第一个本地接口作为路由的本地端。

语法

```
int cl_bestroute6 (int clusterid, char *nodename,
struct sockaddr  *laddr, size_t size_of_laddr,
struct sockaddr *raddr, size_t size_of_raddr)
```

参数

项	描述
clusterid	期望节点的集群标识。
nodename	期望节点的名称。
laddr	用于存放所返回本地网络接口地址的存储器。
size_of_laddr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。
raddr	用于存放所返回远程网络接口地址的存储器。
size_of_raddr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。
CLE_NOROUTE	无可用路由。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

cl_free_clustermap 例程

cl_free_clustermap 例程将释放先前通过调用 **cl_alloc_clustermap** 为集群列表分配的存储器。

语法

```
void cl_free_clustermap (struct cl_cluster *clustermap)
```

参数

项	描述
clustermap	指向要释放的集群映射的指针。

示例

请参阅 cl_getclusters 例程的示例。

相关参考:

第 30 页的『cl_getcluster 例程』

cl_getcluster 将返回有关由集群标识指定的集群的信息。

cl_free_groupmap 例程

cl_free_groupmap 例程将释放先前通过调用 **cl_alloc_groupmap** 分配的存储器。

语法

```
void cl_free_groupmap (struct cl_group *groupmap);
```

参数

项	描述
groupmap	指向要释放的组映射的指针。

示例

请参阅 cl_getgroupmap 例程的示例。

相关参考:

第 36 页的『cl_getgroupmap 例程』

cl_getgroupmap 例程将返回有关集群中资源组的信息。在调用此例程之前，应该调用 **cl_alloc_groupmap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_groupmap**。

cl_free_netmap 例程

释放先前通过调用 **cl_alloc_netmap** 分配的存储器。

语法

```
void cl_free_netmap (struct cl_net *netmap)
```

参数

项	描述
netmap	指向要释放的网络映射的指针。

状态码

无。

示例

请参阅 cl_getnetmap 例程的示例。

相关参考:

第 44 页的『 cl_getnetmap 例程 』
返回有关集群中所有网络的信息。

cl_free_netmap6 例程

释放先前通过调用 **cl_alloc_netmap6** 分配的存储器。

语法

```
void cl_free_netmap6 (struct cl_net *netmap)
```

参数

项	描述
netmap	指向要释放的网络映射的指针。

状态码

无。

cl_free_nodemap 例程

cl_free_nodemap 例程将释放先前通过调用 **cl_alloc_nodemap** 例程分配的存储器。

语法

```
int cl_free_nodemap (struct cl_node *nodemap)
```

参数

项	描述
nodemap	指向要释放的节点映射的指针。

示例

请参阅 cl_getnodemap 例程的示例。

相关参考:

第 50 页的『cl_getnodemap 例程』

cl_getnodemap 例程将返回有关集群中节点的信息。在调用此例程之前，应该调用 **cl_alloc_nodemap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_nodemap**。

cl_free_sitemap 例程

释放先前通过调用 **cl_alloc_sitemap** 分配的存储器。

语法

```
void cl_free_sitemap (struct cl_site *sitemap)
```

参数

项	描述
sitemap	指向要释放的站点映射的指针。

状态码

无。

示例

请参阅 cl_getsitemap 例程的示例。

相关参考:

第 57 页的『cl_getsitemap 例程』

返回有关集群中所有站点的所有已知信息。

cl_getcluster 例程

cl_getcluster 将返回有关由集群标识指定的集群的信息。

语法

```
int cl_getcluster (int clusterid, struct cl_cluster *clusterbuf)
```

参数

项	描述
clusterid	期望集群的集群标识。
clusterbuf	用于存放所返回集群信息的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
  int clusterid = 1113325332;
  int status;
  struct cl_cluster cluster;

  status = cl_getcluster(clusterid, &cluster);
  if (status != CLE_OK) {
     cl_perror(status, "Can't get cluster information");
  } else {
printf("cluster id:
printf("cluster name:
printf("state= %d [%s]\n",
  cluster.clc_state,
  get_state(cluster.clc_state));
printf("substate=
printf("primary= <
printf("nodes= %d, sites= %d, groups= %d, networks= %d\n",
  cluster.clc_number_of_nodes,
  cluster.clc_number_of_sites,
  cluster.clc_number_of_groups,
  cluster.clc_number_of_networks);
  }
```

相关参考:

第 22 页的『 cl_alloc_clustermap 例程 』

cl_alloc_clustermap 例程将为集群列表分配存储器。在调用 **cl_getclusters** 例程之前，必须调用此例程。

第 28 页的『 cl_free_clustermap 例程 』

cl_free_clustermap 例程将释放先前通过调用 **cl_alloc_clustermap** 为集群列表分配的存储器。

cl_getclusterid 例程

cl_getclusterid 例程将返回具有所指定名称的集群的集群标识。

语法

```
int cl_getclusterid (char *clustername)
```

参数

项	描述
clustername	期望集群的名称。

状态码

非负数字（表示集群标识）指示成功。否则，将返回下列其中一个错误状态码:

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERNAME	该请求指定了无效集群名称。

示例

```
int clusterid = 1113325332;
  char clustername[CL_MAXNAMELEN] = "site1";

  clusterid = cl_getclusterid (clustername);
  if (clusterid < 0) {
      cl_perror (clusterid, "can't get cluster ID");
  } else {
      printf ("cluster %s has id %d\n", clustername, clusterid);
  }
```

cl_getclusteridbyifaddr 例程

返回具有所指定网络接口地址的集群的集群标识。此例程只能处理 IPv4 地址。

语法

```
int cl_getclusteridbyifaddr (struct sockaddr_in *addr)
```

参数

项	描述
addr	期望获得其集群标识的网络接口地址。

状态码

非负数字（表示集群标识）指示成功。否则，将返回下列其中一个错误状态码:

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

示例

```
char ifaddr[CL_MAXNAMELEN] = "9.57.28.23";
  int clusterid;
  struct sockaddr_in addr;

  /*
  * inet_addr converts addrs to
  * Internet numbers.
  */
```

```
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr (ifaddr);
clusterid = cl_getclusteridbyifaddr (&addr);
if (clusterid < 0) {
    cl_perror (clusterid,"can't get cluster ID");
} else {
    printf("cluster id w/ interface address %s is %d\n",
    inet_ntoa (addr.sin_addr.s_addr), clusterid);
}
```

cl_getclusteridbyifaddr6 例程

返回具有所指定网络接口地址的集群的集群标识。

语法

```
int cl_getclusteridbyifaddr6 (struct sockaddr *addr, size_t size_of_addr)
```

参数

项	描述
addr	期望获得其集群标识的网络接口地址。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

非负数字（表示集群标识）指示成功。否则，将返回下列其中一个错误状态码:

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

cl_getclusteridbyifname 例程

返回具有所指定网络接口名称的集群的集群标识。

语法

```
int cl_getclusteridbyifname (char *interfacename)
```

参数

项	描述
interfacename	期望获得其集群标识的网络接口名称。

状态码

非负数字（表示集群标识）指示成功。否则，将返回下列其中一个错误状态码:

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
int clusterid;
char interfacename[CL_MAXNAMELEN] = "geotest9";

clusterid = cl_getclusteridbyifname (interfacename);
if (clusterid < 0) {
    cl_perror (clusterid, "can't get cluster id");
} else {
    printf ("cluster id w/ interface named %s is %d\n",
    interfacename, clusterid);
}
```

cl_getclusters 例程

返回有关所有工作集群的信息。

语法

```
int cl_getclusters (struct cl_cluster *clustermap)
```

参数

项	描述
clustermap	返回的集群信息。在调用此例程之前，应该通过调用 **cl_alloc_clustermap** 分配用于存放此信息的存储器。然后在完成时，通过调用 **cl_free_clustermap** 释放此存储空间。

状态码

该例程将返回活动集群的数目。如果该例程未成功，那么它将返回下列其中一个错误状态码:

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。

示例

此示例使用 **cl_errmsg** 例程来说明正确的单线程应用程序编程。如果您的程序是多线程程序，那么必须使用 **cl_errmsg_r** 例程。

```
int i;
int numClusters = -1;
char cbuf[CL_ERRMSG_LEN];
struct cl_cluster *clustermap;

cl_alloc_clustermap (&clustermap);
numClusters = cl_getclusters(clustermap);
if(numClusters < 0)
{
    printf("cl_getclusters: (failed) %s\n",
    cl_errmsg(numClusters));
/*
** for threadsafe compilation use:
```

```
      cl_errmsg_r(numClusters,cbuf));
*/
  }
  else
      printf("there are currently %d running clusters\n", numClusters);
      for(i=0; i < numClusters; i++)
  {
printf("\n  cluster id:
printf("  cluster name:
printf("  state=
printf("  substate=
printf("  primary=
printf("  nodes= %d, sites= %d, groups= %d, networks= %d\n",
  clustermap[i].clc_number_of_nodes,
  clustermap[i].clc_number_of_sites,
  clustermap[i].clc_number_of_groups,
  clustermap[i].clc_number_of_networks);
  }
  }
  cl_free_clustermap(clustermap);
```

cl_getevent 例程

cl_getevent 例程将返回事件通知消息。调用程序只有在接收到先前 **cl_registereventnotify** 请求中指定的信号之后，才应该发出此请求。

语法

```
int cl_getevent (struct cli_en_msg_t * en_msg)
```

参数

项	描述
en_msg	事件通知消息缓冲区，其大小足以存放一条事件通知消息。成功返回时，此缓冲区包含接收到的事件、集群以及（如果适用）网络和节点标识。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。

示例

请参阅 cl_registereventnotify 例程的示例。

相关参考:

第 61 页的『cl_registereventnotify 例程』

cl_registereventnotify 例程将向 Clinfo 注册事件通知请求列表。

cl_getgroup 例程

cl_getgroup 例程将返回有关所指定集群中所指定资源组的信息。

语法

```
int cl_getgroup (int clusterid, char *groupname,
   struct cl_group *groupbufp);
```

参数

项	描述
clusterid	期望集群的集群标识。
groupname	资源组的名称。
groupbufp	指向 **cl_group** 结构的指针，将使用该结构来返回信息。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNODENAME	资源组名称无效。

示例

```
  int clusterid = 1113325332;
  int status, j;
  char* groupname = "rg01";
  struct cl_group group;

  status = cl_getgroup(clusterid, groupname, &group);
  if (status != CLE_OK){
      cl_perror(status, "can't get resource group information");
  } else {
      printf("resource group %s has %d nodes.\n",
      group.clg_name,
      group.clg_num_nodes);
      for(j=0; j < group.clg_num_nodes; j++){
  printf("node w/ id %d is in state %d [%s]\n",
  group.clg_node_ids[j],
  group.clg_node_states[j],
/* user defined function char* cvrt_rg_state(enum cl_resource_states state)
** to convert state id numbers to text
*/
 cvrt_rg_state(group.clg_node_states[j]));
      }
  }
```

cl_getgroupmap 例程

cl_getgroupmap 例程将返回有关集群中资源组的信息。在调用此例程之前，应该调用 **cl_alloc_groupmap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_groupmap**。

语法

```
int cl_getgroupmap (int clusterid, struct cl_group *groupmap)
```

参数

项	描述
clusterid	期望集群的集群标识。
groupmap	用于存放所返回资源组信息的存储器。

状态码

如果该请求返回非负数字（集群中的组数），那么该请求已成功完成。

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
  int i,j;
  int clusterid = 1113325332;
  int groups;
  char cbuf[CL_ERRMSG_LEN];
  struct cl_group *groupmap;

  cl_alloc_groupmap(&groupmap);
  if (groupmap==NULL){
      printf("unable to allocate storage: cl_alloc_groupmap = NULL\n");
      exit(-1);
  }
  groups = cl_getgroupmap(clusterid, groupmap);
  if(groups < 0) {
      cl_perror(groups, "can't get resource group map");
  } else {
      printf("cluster %d has %d resource groups\n", clusterid, groups);
      for(i=0; i < groups; i++){
  printf("resource group %d [%s] has %d nodes\n",
  groupmap[i].clg_group_id,
  groupmap[i].clg_name,
  groupmap[i].clg_num_nodes);
  printf("policies:\n");
  printf("\tstartup  %d \n",groupmap[i].clg_startup_policy);
  printf("\tfallover %d \n",groupmap[i].clg_fallover_policy);
  printf("\tfallback %d \n",groupmap[i].clg_fallback_policy);
  printf("\tsite     %d \n",groupmap[i].clg_site_policy);
  printf("\tuser     %d \n",groupmap[i].clg_user_policy_name);
  printf("node states:\n");
  for(j=0; j < groupmap[i].clg_num_nodes; j++){
 printf("\tnode %d is in state %d \n",
  groupmap[i].clg_node_ids[j],
  groupmap[i].clg_node_states[j]);
    }
  printf("resources
  for(j=0; j < groupmap[i].clg_num_resources; j++){
 printf("\tresource %d is in state %d\n",
  groupmap[i].clg_resource_id[j],
  groupmap[i].clg_res_state[j]);
  }
      }
  }
```

相关参考:

第 23 页的『 cl_alloc_groupmap 例程 』

cl_alloc_groupmap 例程将为资源组描述符列表分配存储器。在调用 **cl_getgroups** 例程之前，必须调用此例程。

第 28 页的『cl_free_groupmap 例程』
cl_free_groupmap 例程将释放先前通过调用 **cl_alloc_groupmap** 分配的存储器。

cl_getgroupnodestate 例程

cl_getgroupnodestate 例程将返回所指定组在所指定节点上的状态。

语法

```
enum cl_resource_states cl_getgroupnodestate (int clusterid,
  char *groupname, int nodeid);
```

参数

项	描述
clusterid	期望集群的集群标识。
groupname	资源组的名称。
nodeid	集群节点的标识。

状态码

返回 **cl_resource_states** 中的枚举值之一。此枚举值在包含资源组状态的枚举类型中进行描述。

示例

```
  enum cl_resource_states state;
  int clusterid = 1113325332;
  int nodeid = 1;
  char groupname[CL_MAXNAMELEN] = "rg01";

  state = cl_getgroupnodestate(clusterid, groupname, nodeid);
  if (state == CL_RGNS_INVALID){
  cl_perror(state, "can't get group node state");
  } else {
  printf("node w/ id %d is currently in state %d in group %s\n",
  nodeid, state, groupname);
  }
```

相关参考:

第 11 页的『包含资源组状态的枚举类型』
此枚举数据类型包含资源组在节点上可以处于的所有状态。

cl_getgroupsbynode 例程

cl_getgroupsbynode 例程将返回所指定节点所隶属的所有资源组的信息。注意，此例程将分配用于存放返回信息的存储器，调用程序必须释放此存储器。

语法

```
int cl_getgroupsbynode (int clusterid, int nodeid,
  struct cl_group **groupbufp, int *groupcountp);
```

参数

项	描述
clusterid	期望集群的集群标识。
nodeid	集群节点的标识。
groupbufp	指向 **cl_group** 结构的指针，将使用该结构来返回信息。
groupcountp	指向一个整数的指针，将使用该整数来存储返回的组数。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNODENAME	节点标识无效。

示例

```
  int clusterid = 1113325332;
  int nodeid = 1;
  int status;
  int groupcount;
  int j;
  struct cl_group *groups;

  status = cl_getgroupsbynode(clusterid, nodeid, &groups, &groupcount);
  if (status != CLE_OK){
      cl_perror(status,"failed to get resource group information");
  } else {
      printf("node %d is a member of %d groups:\n",nodeid, groupcount);
      for(j=0; j < groupcount; j++){
printf("node %d is in group %s\n",
 nodeid, groups[j].clg_name);}
      if (groupcount) free (groups);
  }
```

cl_getifaddr 例程

返回具有所指定集群标识和接口名称的接口的地址。此例程只能处理 IPv4 地址。

语法

```
int cl_getifaddr (int clusterid, char *interfacename,
struct sockaddr_in *addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
interfacename	期望网络接口的名称。
addr	用于存放所返回网络接口地址的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
int clusterid = 1113325332;
int status;
char* interfacename = "geotest9";
struct sockaddr_in addr;

status = cl_getifaddr (clusterid, interfacename, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Can't find interface address");
} else {
    printf ("interface address w/ name %s is %s\n",interfacename,
    inet_ntoa (addr.sin_addr.s_addr));
}
```

cl_getifaddr6 例程

返回具有所指定集群标识和接口名称的接口的地址。

语法

```
int cl_getifaddr6(int clusterid, char *interfacename,
struct sockaddr *addr, size_t size_of_addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
interfacename	期望网络接口的名称。
addr	用于存放所返回网络接口地址的存储器。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

cl_getifname 例程

返回具有所指定集群标识和 IP 地址的接口的名称。此例程只能处理 IPv4 地址。

语法

```
int cl_getifname (int clusterid, struct sockaddr_in *addr,

char *interfacename)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
addr	期望网络接口的 IP 地址。
interfacename	用于存放所返回网络接口名称的存储器。interfacename 参数的长度不应该超过 CL_MAXNAMELEN 个字符。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

示例

```
int clusterid = 1113325332;
int status;
struct sockaddr_in addr;
char interfacename[CL_MAXNAMELEN] = "9.57.28.23";

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr (interfacename);
status = cl_getifname (clusterid, &addr, interfacename);
if (status != CLE_OK) {
    cl_perror (status, "can't find interface name");
}
else {
    printf ("interface name w/ address %s is %s\n",
inet_ntoa (addr.sin_addr.s_addr), interfacename);
}
```

cl_getifname6 例程

返回具有所指定集群标识和 IP 地址的接口的名称。

语法

```
int cl_getifname6 (int clusterid, struct sockaddr *addr,  size_t size_of_addr,
char *interfacename)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
addr	期望网络接口的 IP 地址。
interfacename	用于存放所返回网络接口名称的存储器。interfacename 参数的长度不应该超过 CL_MAXNAMELEN 个字符。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

cl_getlocalid 例程

返回发出请求的节点的集群标识和节点名。此请求将返回集群中当前未处于活动状态的节点的错误状态码。

语法

```
int cl_getlocalid (int *clusterid, char *nodename)
```

参数

项	描述
clusterid	用于存放所返回集群标识的存储器。
nodename	用于存放所返回节点名的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVNODE	节点不是集群节点。

示例

```
int clusterid, status;
char nodename[CL_MAXNAMELEN] = "node1";

status = cl_getlocalid (&clusterid, nodename);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
printf("this node is not a cluster member\n");
    } else {
cl_perror(status, "can't get local cluster ID");
    }
```

```
  } else {
      printf ("this node [%s] is a member of cluster id %d\n",
      nodename,clusterid);
  }
```

cl_getnet 例程

返回有关所指定网络的信息。

语法

```
int cl_getnet (int clusterid, int netid, struct cl_net *netbuf)
```

参数

项	描述
clusterid	期望集群的集群标识。
netid	网络标识。
netbuf	指向 **cl_net** 结构的指针，将使用该结构来返回信息。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNETID	网络标识无效。

示例

```
  int clusterid = 1113325332;
  int netid = 1;
  int status, j;
  struct cl_net netmap;

  status = cl_getnet(clusterid, netid, &netmap);
  if (status == CLE_OK)
  {
 printf("information for cluster network %s (id %d):\n",
netmap.clnet_name, netmap.clnet_id);
 printf("network is type %s\n", netmap.clnet_type);
 printf("network attribute is %d\n", netmap.clnet_attr);
 printf("there are %d nodes on this network\n",
netmap.clnet_numnodes);
 for (j=0; j<netmap.clnet_numnodes; j++)
 {
      enum cls_state node_state;
      printf("  node id = %d, state = %d,",
      netmap.clnet_node_ids[j],
      netmap.clnet_node_states[j]);
      cl_getnetstatebynode( clusterid, netmap.clnet_id,
      netmap.clnet_node_ids[j], &node_state);
      printf(" state (cl_getnetstatebynode) = %d\n",
     node_state);
 }
  }
```

相关参考:

第 47 页的『cl_getnetstatebynode 例程』
返回所指定网络在所指定节点上的状态。

cl_getnetbyname 例程

返回有关已指定的特定网络的信息。

语法

```
int cl_getnetbyname (int clusterid, char *netname,
  struct cl_net *netbuf)
```

参数

项	描述
clusterid	期望集群的集群标识。
netname	网络的名称。
netbuf	指向 **cl_net** 结构的指针，将使用该结构来返回信息。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNETNAME	网络名无效。

示例

```
  char netname[CL_MAXNAMELEN];
  int clusterid = 1;
  int status;
  struct cl_net netmap;

  status = cl_getnetbyname(clusterid,netname,&netmap);
  if (status != CLE_OK) {
display_error(status,cmd);
  } else {
printf("network %s is type %s: connected to %d nodes\n",
netmap.clnet_name,
netmap.clnet_type,
netmap.clnet_numnodes);
  }
```

cl_getnetmap 例程

返回有关集群中所有网络的信息。

语法

```
int cl_getnetmap (int clusterid, struct cl_net *netmap)
```

参数

项	描述
clusterid	期望集群的集群标识。
netmap	用于存放所返回网络信息的存储器。

状态码

如果该请求返回非负数字（集群中的网络数），那么该请求已成功完成。

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
int clusterid = 1113325332;
int num_nets;
int i,j;
struct cl_net *nm;

cl_alloc_netmap(&nm);
num_nets = cl_getnetmap(clusterid, nm);
printf("\n\ndumping netmap with %d nets for cluster %d\n\n",
    num_nets, nm->clnet_clusterid);
for (i=0; i<num_nets; i++)
{
    printf("info for network: %s (%d)\n", nm[i].clnet_name,
   nm[i].clnet_id);
    printf(" type = %s\n", nm[i].clnet_type);
    printf(" attribute = %d\n", nm[i].clnet_attr);
    printf(" state = %d\n", nm[i].clnet_state);
    printf(" number of nodes = %d\n", nm[i].clnet_numnodes);
    for (j=0; j<nm[i].clnet_numnodes; j++)
    {
printf(" node id = %d , state = %d\n",
nm[i].clnet_node_ids[j],
nm[i].clnet_node_states[j]);
}
}
cl_free_netmap(nm);
```

相关参考:

第 23 页的『 cl_alloc_netmap 例程 』
分配用于存放一系列网络信息结构的存储器。

第 28 页的『 cl_free_netmap 例程 』
释放先前通过调用 **cl_alloc_netmap** 分配的存储器。

cl_getnetsbyattr 例程

返回有关配置了所指定属性（public、private 或 non-IP (serial)）的任何网络的信息。

语法

```
int cl_getnetsbyattr (int clusterid, int *netattr,
   struct cl_net **netbuf, int *netcount)
```

参数

项	描述
clusterid	期望集群的集群标识。
netattr	要检索的网络列表的网络属性。
netbuf	指向 **cl_net** 结构的指针，将使用该结构来返回信息。
netcount	指向一个整数的指针，将使用该整数来返回与所指定类型匹配的网络数。

状态码

项	描述
CLE_OK	成功。注意，即使未返回任何网络（即，具有所指定属性的网络不存在），也可能返回此代码。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNETATTR	网络属性无效。

示例

```
  int status,i;
  int netcount;
  int clusterid = 1;
  struct cl_net *netmap;
  enum cl_network_attribute attr;

  attr = CL_NET_TYPE_PRIVATE;
  status = cl_getnetsbyattr(clusterid, attr, &netmap,&netcount);
  if (status != CLE_OK) {
     cl_perror(status,"cl_getnetsbyattr() failed");
  } else {
     printf("there are %d private networks in this cluster.\n",netcount);
     for (i=0; i<netcount; i++)
     {
struct cl_net network;
 status = cl_getnetbyname(clusterid, netmap[i].clnet_name,
   &network);
 if (status != CLE_OK) {
      cl_perror(status,"cl_getnetbyname() failed");
} else {
      printf(" network %s is type %s: connected to %d nodes\n",
  network.clnet_name,
  network.clnet_type,
  network.clnet_numnodes);
 }
     }
  }
```

cl_getnetsbytype 例程

返回有关任何配置了所指定网络类型的网络的信息。

语法

```
int cl_getnetsbytype (int clusterid, char *nettype,
   struct cl_net **netbuf, int *netcount)
```

参数

项	描述
clusterid	期望集群的集群标识。
nettype	要检索的网络列表的网络类型。将从当前对 PowerHA SystemMirror 定义的网络接口模块 (NIM) 列表中检索网络类型。
netbuf	指向 **cl_net** 结构的指针，将使用该结构来返回信息。
netcount	指向一个整数的指针，将使用该整数来返回与所指定类型匹配的网络数。

状态码

项	描述
CLE_OK	成功。即使未返回任何网络（即，所指定类型的网络不存在），也可能返回此代码。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNETTYPE	网络类型无效。

示例

```
char net_type[CL_MAXNAMELEN] = "ether";
  int status,i,j;
  int netcount;
  int clusterid = 1113325332;
  struct cl_net *netmap;

  status = cl_getnetsbytype(clusterid, net_type, &netmap, &netcount);
  if (status != CLE_OK) exit(status);
  printf("there are %d networks of type:
  for (i=0; i<netcount; i++)
  {
    struct cl_net network;
    status = cl_getnetbyname(clusterid, netmap[i].clnet_name, &network);
    if (status != CLE_OK) exit(status);
    printf("network %s has attribute %d and is connected to %d nodes\n",
    network.clnet_name,
    network.clnet_attr,
    network.clnet_numnodes);
  }
```

cl_getnetstatebynode 例程

返回所指定网络在所指定节点上的状态。

语法

```
int cl_getnetstatebynode (int clusterid, int netid,
  int nodeid, enum cls_state *state)
```

参数

项	描述
clusterid	期望集群的集群标识。
netid	要检索的网络状态的网络标识。
nodeid	要检索的网络状态的节点标识。
netstate	指向一个整数的指针，将使用该整数来返回所指定网络在所指定节点上的状态。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVNETID	网络标识无效。
CLE_IVNODEID	节点标识无效。注意，此返回值可以指示仅标识无效（它不是此集群的有效节点标识），也可以指示节点未连接指定的网络（节点在该网络上没有接口）。

示例

请参阅 cl_getnet 例程的示例。

相关参考:

第 43 页的『cl_getnet 例程』

返回有关所指定网络的信息。

cl_getnode 例程

返回有关由一对集群标识/节点名指定的节点的信息。

语法

```
int cl_getnode (int clusterid, char *nodename,
  struct cl_node *nodebuf);
```

参数

项	描述
clusterid	期望节点的集群标识。
nodename	期望节点的节点名。
nodebuf	用于存放所返回节点信息的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
  int clusterid = 1113325332;
  int status;
  char* nodename = "node1";
  struct cl_node nodebuf;

  status = cl_getnode (clusterid, nodename, &nodebuf);
  if (status != CLE_OK) {
      cl_perror(status, "can't get node info");
  } else {
      printf("node named %s on cluster %d : id= %d, state= %d\n",
 nodename,
 clusterid,
 nodebuf.cln_nodeid,
 nodebuf.cln_state);
  }
  cl_node_free(&nodebuf);
```

cl_getnodeaddr 例程

返回与一对指定的集群标识/网络接口名称相关联的 IP 地址。此例程只能处理 IPv4 地址。

语法

```
int cl_getnodeaddr (int clusterid, char *interfacename,
struct sockaddr_in *addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
interfacename	期望网络接口的名称。
addr	用于存放所返回网络接口地址的存储器。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
  int clusterid = 1113325332;
  int status;
  char* interfacename = "geotest9";
```

```
struct sockaddr_in addr;

status = cl_getnodeaddr(clusterid, interfacename, &addr);
if (status != CLE_OK) {
    cl_perror(status, "can't get node addr");
} else {
    printf("address of interface %s on cluster %d is %s\n",
    interfacename, clusterid, inet_ntoa(addr.sin_addr));
}
```

cl_getnodeaddr6 例程

返回与一对指定的集群标识/网络接口名称相关联的 IP 地址。

语法

```
int cl_getnodeaddr6 (int clusterid, char *interfacename,
struct sockaddr *addr, size_t size_of_addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
interfacename	期望网络接口的名称。
addr	用于存放所返回网络接口地址的存储器。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

cl_getnodemap 例程

cl_getnodemap 例程将返回有关集群中节点的信息。在调用此例程之前，应该调用 **cl_alloc_nodemap** 以在内存中预留存储器。在调用此例程之后，应该调用 **cl_free_nodemap**。

语法

```
int cl_getnodemap (int clusterid, struct cl_node *nodemap)
```

参数

项	描述
clusterid	期望集群的集群标识。
nodemap	用于存放所返回节点信息的存储器。

状态码

如果该请求返回非负数字（集群中的节点数），那么该请求已成功完成。

项	描述
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
int clusterid = 1113325332;
int i;
int nodes;
struct cl_node *nodemap;

cl_alloc_nodemap (&nodemap);
if (nodemap==NULL){
    printf("unable to allocate storage: cl_alloc_nodemap = NULL\n");
    exit(-1);
}
nodes = cl_getnodemap(clusterid, nodemap);
if(nodes < 0)
{
    cl_perror(nodes,"can't get node map");
} else {
  printf("cluster %d has %d nodes:\n", clusterid, nodes);
  for(i=0; i < nodes; i++){
    printf("node %s in state %d has %d interfaces\n",
    nodemap[i].cln_nodename,
    nodemap[i].cln_state,
    nodemap[i].cln_nif);
    if(clusterid != nodemap[i].cln_clusterid){
printf("structure has invalid cluster ID: %d",
nodemap[i].cln_clusterid);
    }
  }
}
cl_free_nodemap(nodemap);
```

相关参考:

第 24 页的『cl_alloc_nodemap 例程』
cl_alloc_nodemap 例程将为多个节点（或一组节点）以及与每个节点相关联的接口分配存储器。在调用 **cl_getnodemap** 例程之前，应该调用此例程。

第 29 页的『cl_free_nodemap 例程』
cl_free_nodemap 例程将释放先前通过调用 **cl_alloc_nodemap** 例程分配的存储器。

cl_getnodenamebyifaddr 例程

返回具有所指定接口地址的节点的名称。Clinfo 将扫描集群中每个节点上的网络接口。如果找到了匹配项，那么将返回与该接口地址相关联的节点的节点名。

语法

```
int cl_getnodenamebyifaddr (int clusterid, struct sockaddr_in *addrp,
   char *nodename)
```

参数

项	描述
clusterid	期望节点的集群标识。
addr	期望节点的网络接口地址。
nodename	期望节点的名称。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

示例

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN];
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("9.57.28.23");
status = cl_getnodenamebyifaddr (clusterid, &addr, nodename);
if (status !=CLE_OK) {
     cl_perror(status,"can't get node name");
} else {
     printf("node name of interface w/ address %s on
             cluster %d is %s\n",
inet_ntoa(addr.sin_addr), clusterid, nodename);
}
```

cl_getnodenamebyifaddr6 例程

返回具有所指定接口地址的节点的名称。Clinfo 将扫描集群中每个节点上的网络接口。如果找到了匹配项，那么将返回与该接口地址相关联的节点的节点名。

语法

```
int cl_getnodenamebyifaddr6 (int clusterid, struct sockaddr *addrp, size_t size_of_addr, char *nodename)
```

参数

项	描述
clusterid	期望节点的集群标识。
addr	期望节点的网络接口地址。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。
nodename	期望节点的名称。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。

cl_getnodenamebyifname 例程

返回具有所指定接口名称的节点的节点名。Clinfo 将扫描集群中每个节点上的网络接口。如果找到了匹配项，那么将返回该节点的节点名。

语法

```
int cl_getnodenamebyifname (int clusterid, char *interfacename,
  char *nodename)
```

参数

项	描述
clusterid	期望节点的集群标识。
interfacename	期望节点的网络接口名称。
nodename	期望节点的名称。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
  int clusterid = 1113325332;
  int status;
  char nodename[CL_MAXNAMELEN];
  char interfacename[CL_MAXNAMELEN] = "geotest9";<

  status = cl_getnodenamebyifname (clusterid, interfacename, nodename);
  if (status != CLE_OK) {
      cl_perror(status,"can't get node name");
  } else {
```

```
      printf("interface=
      printf("node name of interface w/ name %s on cluster %d is %s\n",
interfacename,
clusterid,
nodename);
  }
```

cl_getprimary 例程

返回所指定集群的用户所指定主集群管理器的节点名。

语法

```
int cl_getprimary (int clusterid, char *nodename)
```

参数

项	描述
clusterid	期望获得其主节点名称的集群标识。
nodename	将在此自变量中返回已指定为主集群管理器节点的节点的名称。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_NOPRIMARY	未获得有关主集群管理器的信息。
CLE_IVCLUSTER	该集群不可用。

示例

```
  int clusterid = 1113325332;
  int status;
  char nodename[CL_MAXNAMELEN] = "node1";

  status = cl_getprimary (clusterid, nodename);
  if (status != CLE_OK) {
      cl_perror(status,"can't get cluster primary");
  } else {
      printf ("primary node for cluster %d is %s\n",
      clusterid, nodename);
  }
```

cl_getsite 例程

返回有关特定站点的信息。

语法

```
int cl_getsite(int clusterid, int siteid, struct cl_site *sitebuf)
```

参数

项	描述
clusterid	期望的集群。
siteid	集群的标识。
sitebuf	存放返回的 **cl_site** 结构。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVSITEID	站点标识无效。

示例

```
int clusterid = 1113325332;
int status;
int siteid = 1;
struct cl_site site;

status = cl_getsite(clusterid, siteid, &site);
if (status == CLE_OK) {
    printf("site %s (%d) has %d nodes and is priority %d\n",
site.clsite_name,
site.clsite_id,
site.clsite_numnodes,
site.clsite_priority);
} else {
    cl_perror(status,"can't get site information");
}
```

cl_getsitebyname 例程

返回有关已指定的特定站点的信息。

语法

```
int cl_getsitebyname (int clusterid, char *sitename, struct cl_site *sitebuf)
```

参数

项	描述
clusterid	期望的集群。
sitename	要返回的站点的名称。
sitebuf	存放返回的 **cl_site** 结构。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVSITENAME	站点标识无效。

示例

```
  int clusterid = 1113325332;
  int status;
  char sitename[CL_MAXNAMELEN] = "geo9";
  struct cl_site site;

  status = cl_getsitebyname(clusterid, sitename, &site);
  if (status == CLE_OK)
      printf("site %s (%d) has %d nodes and is priority %d\n",
site.clsite_name,
site.clsite_id,
site.clsite_numnodes,
site.clsite_priority);
```

cl_getsitebypriority 例程

返回有关配置了所指定优先级的任何站点的信息。

语法

```
int cl_getsitebypriority (int clusterid, enum cl_site_priority,
    priority, struct cl_site *sitebuf)
```

参数

项	描述
clusterid	期望的集群。
priority	枚举的站点优先级值之一。
site_buf	存放返回的站点信息。

状态码

项	描述
CLE_OK	成功。
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。
CLE_IVSITEPRIORITY	站点优先级无效。

示例

```
  enum cl_site_priority priority = CL_SITE_PRI_PRIMARY;
  int clusterid = 1113325332;
  int status;
  struct cl_site site;
```

```
  status = cl_getsitebypriority(clusterid, priority, &site);
  if (status == CLE_OK)
      printf("site %s (%d) has %d nodes and is priority %d\n",
site.clsite_name,
site.clsite_id,
site.clsite_numnodes,
site.clsite_priority);
```

cl_getsitemap 例程

返回有关集群中所有站点的所有已知信息。

语法

```
int cl_getsitemap (int clusterid, struct cl_site *sitemap)
```

参数

项	描述
clusterid	期望的集群。
sitemap	在调用 **cl_getsitemap** 之前，必须使用 **cl_alloc_sitemap(&sitemap)** 分配存储器。必须使用 **cl_free_sitemap(sitemap)** 释放存储器。

状态码

如果该请求返回非负数字（集群中的站点数），那么该请求已成功完成。

项	描述
CLE_BADARGS	缺少自变量或自变量无效。
CLE_SYSERR	系统错误。
CLE_NOCLINFO	未获得集群信息。
CLE_IVCLUSTERID	集群标识无效。

示例

```
  int clusterid = 1113325332;
  int status;
  int i,j;
  int nbr_sites;
  struct cl_site *sitemap;

  cl_alloc_sitemap(&sitemap);
  nbr_sites = cl_getsitemap(clusterid, sitemap);
  printf("dumping sitemap with %d sites for cluster %d\n\n",
      nbr_sites, sitemap->clsite_clusterid);
  for (i=0; i<nbr_sites; i++)
  {
      printf("info for site %s (%d)\n",
      sitemap[i].clsite_name, sitemap[i].clsite_id);
      printf(" priority = %d \n", sitemap[i].clsite_priority);
      printf(" backup = %d \n", sitemap[i].clsite_backup);
      printf(" state = %d \n", sitemap[i].clsite_state);
      printf(" number of nodes = %d\n", sitemap[i].clsite_numnodes);
      for (j=0; j<sitemap[i].clsite_numnodes; j++)
      {
  printf(" node id = %d\n",
  sitemap[i].clsite_nodeids[j]);
      }
  }
  cl_free_sitemap(sitemap);
```

相关参考:

第 25 页的『cl_alloc_sitemap 例程』
分配用于存放一系列站点信息结构的存储器。

第 30 页的『cl_free_sitemap 例程』
释放先前通过调用 **cl_alloc_sitemap** 分配的存储器。

cl_isaddravail 例程

返回所指定网络接口的状态。此例程只能处理 IPv4 地址。

语法

```
int cl_isaddravail (int clusterid, char *nodename,
struct sockaddr_in *addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
nodename	期望网络接口的节点名。
addr	期望网络接口的地址。

状态码

项	描述
CLE_OK	指定的地址在给定节点中可用。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVADDRESS	该请求指定了无效接口地址。
CLE_IVNETIF	该网络接口不可用。

示例

```
  int clusterid = 1113325332;
  int status;
  char ifaddr[CL_MAXNAMELEN] = "9.57.28.23";
  char nodename[CL_MAXNAMELEN] = "node1";
  struct sockaddr_in addr;

  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = inet_addr (ifaddr);
  status = cl_isaddravail (clusterid, nodename, &addr);
  if (status != CLE_OK) {
      cl_perror(status,"interface not available");
  } else {
      printf ("interface address for %s is available\n",
inet_ntoa (addr.sin_addr.s_addr));
  }
```

cl_isaddravail6 例程

返回所指定网络接口的状态。

语法

```
int cl_isaddravail6 (int clusterid, char *nodename,
struct sockaddr *addr, size_t size_of_addr)
```

参数

项	描述
clusterid	期望网络接口的集群标识。
nodename	期望网络接口的节点名。
addr	期望网络接口的地址。
size_of_addr	“addr”缓冲区的大小。它不得小于大小“struct sockaddr_in”（对于 IPv4 套接字）和“struct sockaddr6_in”（对于 IPv6 套接字）。

状态码

项	描述
CLE_OK	指定的地址在给定节点中可用。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVADDRESS	该请求指定了无效接口地址。
CLE_IVNETIF	该网络接口不可用。

cl_isclusteravail 例程

返回具有所指定集群标识的集群的状态。

语法

```
int cl_isclusteravail (int clusterid)
```

参数

项	描述
clusterid	期望集群的集群标识。

状态码

项	描述
CLE_OK	该集群可用。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_IVCLUSTER	指定的集群不可用。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
int clusterid = 1113325332;
int status;

status = cl_isclusteravail (clusterid);
if (status != CLE_OK) {
```

```
    cl_perror (status, "cluster is not available");
} else {
    printf ("cluster %d is available\n", clusterid);
}
```

cl_isnodeavail 例程

指示所指定节点是否处于活动状态。

语法

```
int cl_isnodeavail (int clusterid, char *nodename)
```

参数

项	描述
clusterid	期望节点的集群标识。
nodename	期望节点的节点名。

状态码

项	描述
CLE_OK	该节点在指定的集群中可用。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVNODE	该节点不可用。

示例

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN] = "node1";

status = cl_isnodeavail (clusterid, nodename);
if (status != CLE_OK) {
    cl_perror(status,"node is unavailable");
} else {
    printf ("node %s on cluster %d is available\n",
   nodename, clusterid);
}
```

cl_model_release 例程

先前发行版中使用 **cl_model_release()** 例程来拆离共享内存。

cl_model_release() 例程不再具有任何功能，将始终返回 **CLE_OK**。提供 **cl_model_release()** 仅为了支持向后兼容性。请**不要**将此例程用于任何新程序。

cl_node_free 例程

删除先前为某些网络接口分配的存储器，这些网络接口与先前通过调用 cl_getnode 例程所返回的节点相关联。成员 **cln_if** 将设置为空。

语法

```
int cl_node_free
```

参数

项	描述
nodebifp	节点缓冲区

状态码

项	描述
CLE_OK	请求已成功完成。

样本源代码

```
int clusterid;
struct cl_node nodebuf;
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("9.57.28.8");

clusterid = cl_getclusteridbyifaddr (&addr);

cl_getnode(clusterid, "ppstest5", &nodebuf);
printf ("node id: %d has %d interfaces \n",
     nodebuf.cln_nodeid, nodebuf.cln_nif);

/* call cl_node_free to release storage for the network */
/* interfaces (part of the cl_node struct) which was    */
/* allocated by cl_getnode */
cl_node_free(&nodebuf);

printf ("after cl_node_free, node id: %d has %d interfaces \n",
    nodebuf.cln_nodeid, nodebuf.cln_nif);
```

样本输出

```
node id: 1 has 5 interfaces
after cl_node_free, node id: 1 has 0 interfaces
```

cl_registereventnotify 例程

cl_registereventnotify 例程将向 Clinfo 注册事件通知请求列表。

每个请求都会指定事件标识、集群标识、信号标识以及（如果适用）节点名和/或网络标识。如果该例程成功，那么当发生符合所指定描述的事件时，Clinfo 会向主调进程发出信号。接收到此信号时，可以调用 **cl_getevent** 例程以获取有关刚发生的事件的信息。

在注册时可以将 -1 指定为事件标识，以便接收所有网络事件和集群事件的通知。如果您注册为要接收所有这种事件的通知，那么不能注销特定事件的通知；必须全部注销。

注意，不能对包括节点名标识的事件使用 -1 事件标识。请指定 NULL 字符串以注册所有节点名。当前，不能指定单个网络标识。请使用 -1 作为事件标识。

此例程在集群标识为 0 时不工作。请分配另一个数字作为集群标识。

语法

```
#include <signal.h>
int cl_registereventnotify (int num_reqs, struct cli_enr_req_t*enr_list)
```

参数

项	描述
num_reqs	要发出的事件通知注册请求数（不能超过 10）。
enr_list	事件通知注册请求列表。可以请求的事件包括: **CL_SWAPPING_ADAPTER**。正在交换指定的适配器。此事件需要集群标识、节点标识和网络标识。集群标识和网络标识的值为 -1 时，指示所有这种组件。请对所有节点名指定 NULL 字符串。此事件对应于 **hacmp.out** 中的 **swap_adapter** 事件。 **CL_SWAPPED_ADAPTER**。已交换指定的适配器。此事件需要集群标识、节点标识和网络标识。集群标识和网络标识的值为 -1 时，指示所有这种组件。请对所有节点名指定 NULL 字符串。此事件对应于 **hacmp.out** 中的 **swap_adapter_complete** 事件。 **CL_JOINING_NETWORK**。指定的网络处于加入过程。此事件需要集群标识和网络标识。其中任一标识的值为 -1 时，指示所有这种组件。此事件对应于 **hacmp.out** 中的 **network_up** 事件。 **CL_JOINED_NETWORK**。指定的网络已启动。此事件需要集群标识和网络标识。其中任一标识的值为 -1 时，指示所有这种组件。此事件对应于 **hacmp.out** 中的 **network_up_complete** 事件。 **CL_FAILING_NETWORK**。指定的网络正在关闭。此事件需要集群标识和网络标识。其中任一标识的值为 -1 时，指示所有这种组件。此事件对应于 **hacmp.out** 中的 **network_down** 事件。 **CL_FAILED_NETWORK**。指定的网络已关闭。此事件需要集群标识和网络标识。其中任一标识的值为 -1 时，指示所有这种组件。此事件对应于 **hacmp.out** 中的 **network_down_complete** 事件。
	CL_JOINING_NODE。指定的节点正在启动。此事件需要集群标识和节点标识。此事件对应于 **hacmp.out** 中的 **node_up** 事件。 CL_JOINED_NODE。指定的节点已启动。此事件需要集群标识和节点标识。此事件对应于 **hacmp.out** 中的 **node_up_complete** 事件。 **CL_FAILING_NODE**。指定的节点正在关闭。此事件需要集群标识和节点标识。此事件对应于 **hacmp.out** 中的 **node_down** 事件。 **CL_FAILED_NODE**。指定的节点已关闭。此事件需要集群标识和节点标识。此事件对应于 **hacmp.out** 中的 **node_down_complete** 事件。 **CL_NEW_PRIMARY**。指定的集群具有新的主节点。此事件需要集群标识。注意，主节点的概念是来自先前发行版的属性，可能不对应于与应用程序相关的节点的角色。 **CL_STABLE**。指定的集群已稳定。此事件需要集群标识。值为 -1 时，指示所有这种组件。 **CL_UNSTABLE**。指定的集群已不稳定。此事件需要集群标识。值为 -1 时，指示所有这种组件。
SIGNALS	您可以指定任何适当的 UNIX 信号；假定您熟悉信号。建议指定 SIGUSR1 和 SIGUSR2。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVREQNUM	指定了无效的事件通知注册请求数（大于 10）。
CLE_IVNETID	指定了无效网络标识。
CLE_IVEVENTID	指定了无效事件标识。

示例

以下示例说明了如何在简单测试程序中使用 **cl_registereventnotify**、**cl_unregistereventnotify** 和 **cl_getevent** 例程。

在此示例中，应用程序注册为要在出现以下情况时通过 SIGUSR1 信号得到通知: 已连接至集群 83 的节点 2 的任何网络发生足以导致 FAILING_NETWORK 事件的严重问题。在注册之后，应用程序等待该事件。发生该事件时，Clinfo 将 SIGUSR1 信号发送到应用程序；然后应用程序通过使用 **cl_getevent** 来获取有关该事件的信息，然后打印出所接收到的信息。最后，应用程序注销此事件的通知。

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <cluster/clinfo.h>
volatile int no_signal = 1;

/* Signal handler for catching event notification signal */
void
catch_sig(int sig)
{
no_signal = 0;
}

int
main(int argc, char *argv[])
{
  int ret_code, i;
  struct cli_enr_req_t en_req;  /*  Event notification request  */
  struct cli_en_msg_t  en_msg;  /*  Event notification message */
  en_req.event_id = CL_FAILING_NETWORK; /* specify a failing
    network  event  */
  en_req.cluster_id = 83;
  strcpy (en_req.node_name,"node2");
  en_req.net_id = 1;      /*  no net id necessary
    for this event  */
  en_req.signal_id = SIGUSR1;/*  Request to register for event
    notification */

  if(ret_code = cl_registereventnotify((int) 1, &en_req)!=CLE_OK )
  {
    printf("cl_en_test: cl_registereventnotify failed
    with error
  exit(1);
  }
/* Set up a signal handler to catch the event notification
   signal from Clinfo   */

  ret_code = signal(SIGUSR1, catch_sig);
  if ( ret_code < 0 ){
    perror("cl_en_test");
    exit(1);
  }
    /*  Wait for signal  */

  printf("cl_en_test: waiting for signal from Clinfo.");
  while (no_signal){
  pause();}
    /*  Execution will start here after catch_sig executes when
the signal is received.  Get the event notification message.  */

  if ( ret_code = cl_getevent(&en_msg) != CLE_OK )
      {
  printf("cl_en_test: cl_getevent failed with error %d.",
     ret_code);
  exit(1);
```

```
      }
/* Print out the event notification information received */
  printf("cl_en_test: Event notification message received
    from Clinfo:");
  printf("cl_en_test: Event id = %d", en_msg.event_id);
  printf("cl_en_test: Cluster id = %d", en_msg.cluster_id);
  printf("cl_en_test: Node name = %s", en_msg.node_name);
  printf("cl_en_test: Net id = %d", en_msg.net_id);
/*  Request to unregister the event notification  */
    if ( (ret_code = cl_unregistereventnotify((int) 1,&en_req))!=CLE_OK)
  {
  printf("cl_en_test: cl_unregistereventnotify failed with
  error %d.", ret_code);
  exit(1);
  }
}
```

相关参考:

第 35 页的『cl_getevent 例程』

cl_getevent 例程将返回事件通知消息。调用程序只有在接收到先前 **cl_registereventnotify** 请求中指定的信号之后，才应该发出此请求。

『cl_unregistereventnotify 例程』

cl_unregistereventnotify 例程将向 Clinfo 注销事件通知请求列表。每个请求都会指定事件标识、集群标识、信号标识以及（如果适用）节点和/或网络标识。如果该例程成功，那么 Clinfo 将注销所有符合所指定描述的事件。

cl_unregistereventnotify 例程

cl_unregistereventnotify 例程将向 Clinfo 注销事件通知请求列表。每个请求都会指定事件标识、集群标识、信号标识以及（如果适用）节点和/或网络标识。如果该例程成功，那么 Clinfo 将注销所有符合所指定描述的事件。

如果已通过将 -1 指定为事件标识来注册为要接收所有网络事件或集群事件的通知，那么必须全部注销。您不能全部注册，然后又注销各个事件。如果已注册特定事件，那么请分别注销那些事件。

语法

```
int cl_unregistereventnotify (int num_reqs,
  struct cli_enr_req_t *enr_list)
```

参数

请参阅 cl_registereventnotify 例程的参数。

状态码

请参阅 cl_registereventnotify 例程的状态码。

示例

请参阅 cl_registereventnotify 例程的示例。

相关参考:

第 61 页的『cl_registereventnotify 例程』

cl_registereventnotify 例程将向 Clinfo 注册事件通知请求列表。

Clinfo C++ API

Clinfo C++ API 是面向对象的接口，您可以在 C++ 应用程序中使用该接口来获取有关 PowerHA SystemMirror for AIX 集群的状态信息。这些主题描述了 Clinfo C++ API 中可用的特定 C++ 语言对象和方法。

除了本节之外，还请阅读『Clinfo C API』，它描述了 C API。C++ API 例程将调用 C API 例程。

请参阅 *IBM® C for AIX C/C++ Language Reference*，以了解有关 C++ 和 AIX XL C++ 编译器的更多信息。

注:

- Clinfo API 支持发行版之间的版本兼容性。如果已使用先前的 Clinfo 发行版编译 Clinfo 程序，那么该程序无需重新编译即可用于新的 Clinfo 发行版。但是，如果要使用此发行版中的新功能，那么将必须重新编译。
- 不推荐使用 **cl_registerwithclsmuxpd()** API。任何使用此 API 编译的 pre-script 或 post-script 都将无法装入。请使用应用程序监视来代替 **cl_registerwithclsmuxpd()** 例程。请参阅《规划指南》中有关初始集群规划的部分。

相关概念:

第 9 页的『Clinfo C API』

Clinfo C 应用程序编程接口 (API) 是高级接口，您可以在应用程序中使用该接口来获取有关 PowerHA SystemMirror 集群的状态信息。这些主题描述了 Clinfo C API 中可用的特定 C 语言例程和实用程序。

相关信息:

初始集群规划

Clinfo C++ 对象类

Clinfo C++ API 具有下列对象类: 集群、节点、网络接口和资源组。

每个网络接口都属于单个节点；每个节点和每个资源组都属于单个集群；这决定了类结构。集群类是最一般的类，接口类是最具体的类。

将函数按类进行分组取决于函数所使用的数据，并且会将函数划分到最一般的可能类。注意，在网络接口类和集群类中都存在名为 **CL_getclusterid** 的函数。这些函数是不同函数，虽然名称相同，但是类不相同。此命名约定是 C++ 中的标准。

所有值都将通过使用常规函数返回值传递到调用程序，而不是通过使用按引用传递的函数参数进行传递。当这些描述谈论到所传入的数据时，隐式说明了此数据来自函数所隶属的对象。

状态始终在 **CL_status** 自变量中返回，该自变量按引用传递，应该检查该自变量以获取错误情况。返回值始终是数据。这种情况的一个例外是 **CL_isavail()** 函数，其主要返回数据是状态，所以该函数不是使用 **CL_status** 自变量来返回状态。

Clinfo C++ API 包括两个不属于上述任何类的函数。**CL_getallinfo** 将处理集群数组而不是集群对象。它返回集群数组，并且数组指针按引用传递。**CL_getlocalid** 将处理本地节点而不是节点对象。本地主机返回它自己的名称。

在应用程序中使用 Clinfo C++ API

本节描述如何在应用程序中使用 Clinfo C++ API。

PowerHA SystemMirror for AIX 为多线程应用程序和单线程应用程序提供了不同库。请确保与适合于应用程序的库进行链接。

链接至 Clinfo C API

C++ 程序可以通过使用适当的链接伪指令来调用 Clinfo C API。

例如:

```
extern "C" int cl_getclusterid (char *);
```

请参阅 *AIX XL C++/6000 V1.1.1 Language Reference* 以了解更多信息。

允许 C++ 程序调用 Clinfo C API 的链接伪指令包含在 **clinfo.h** 中。但是，如果您需要更加面向对象的 C++ API，那么可以使用 Clinfo C++ API。

Clinfo C++ API 将定义集群、节点和网络接口的对象类。从这些对象检索数据的所有 Clinfo C++ 函数都是这些类的成员函数（有时称为方法）。

头文件

必须在每个使用 Clinfo C++ API 的源模块中指定 include 伪指令。

此 include 伪指令如下所示:

```
#include <cluster/clinfo.H>
```

libclpp.a 和 libclpp_r.a 库

必须将伪指令添加到使用 Clinfo C++ API 的*单线程*应用程序的对象装入命令

此伪指令如下所示:

```
-lclpp -lcl -lclstr
```

必须将以下伪指令添加到使用 Clinfo C API 的*多线程*应用程序的对象装入命令:

```
-lclpp_r -lcl_r  -lclstr_r
```

libclpp.a 和 **libclpp_r.a** 库存放支持 Clinfo C++ API 的例程；**libcl.a** 和 **libcl_r.a** 库包含支持 Clinfo C API 的例程。

要在 AIX 下编译 C++ 程序，请使用 xlC 编译器。

注: Clinfo 的 **cluster.es.client.lib** 库现在包含具有 32 位和 64 位对象的 **libcl.a**。必须在 64 位环境中重新编译/重新链接应用程序，才能获取使用 Clinfo API 的 64 位应用程序。

常量

Clinfo C++ API 例程使用 **clinfo.h** 文件中定义的常量。

项	描述
CL_MAXNAMELEN	用于对集群、节点或接口进行命名的字符串的最大长度 (256)。**CL_MAXNAMELEN** 的值与 **sys/param.h** 文件中定义的 **MAXHOSTNAMELEN** 相同。
CL_MAX_EN_REQS	允许的最大事件通知请求数 (10)。
CL_ERRMSG_LEN	最大错误消息长度（128 个字符）。
CLSMUXPD_SVC_PORT	该软件的先前版本中使用的常量 **CL_MAXCLUSTERS**、**CL_MAXNODES** 和 **CL_MAXNETIFS** 将替换为提供数据结构中各种数组的当前大小的宏。这些宏与它们所替换的常量同名。这些宏不能用作声明语句中的数组边界，这是因为已使用这些常量。
CL_MAXCLUSTERS	提供共享内存中数组的当前空间大小，该数组用于存放有关集群数的信息。
CL_MAXNODES	提供共享内存中数组的当前空间大小，该数组用于存放有关集群中节点数的信息。

项	描述
CL_MAXNETIFS	提供共享内存中数组的当前空间大小，该数组用于存放有关已连接至节点的接口数的信息。
CL_MAXGROUPS	提供共享内存中数组的当前空间大小，该数组用于存放有关资源组数的信息。

作为示例，以下代码片段说明了如何使用常量 **CL_MAXNODES** 来设置集群对象数组的大小。随后的示例说明如何将此常量替换为对同名例程的调用。

```
CL_cluster clusters[CL_MAXNODES];
CL_cluster *ret = &clusters[0];
ret = CL_getallinfo(ret, s);
if (s < 0)
cl_errmsg(s);
for (int i=0; i<CL_MAXNODES; i++) {
printf("[%d] cl %d", i, ret[i].clc_clusterid);
printf(" st %d", ret[i].clc_state);
printf(" su %d", ret[i].clc_substate);
printf(" pr %d", ret[i].clc_primary);
printf(" na %s", ret[i].clc_name.name);
}
```

请不要再使用常量 **CL_MAXNODES**。

```
CL_cluster clusters[8];
CL_cluster *ret = &clusters[0];
int numclus;
numclus = CL_getallinfo(ret, s);
if (s < 0)
cl_perror(s, progname);
printf("number of clusters found: %d", numclus);
for (int i=0; i<numclus; i++) {
printf("[%d] cl %d", i, ret[i].clc_clusterid);
printf(" st %d", ret[i].clc_state);
printf(" su %d", ret[i].clc_substate);
printf(" pr %s", ret[i].clc_primary);
printf(" na %s", ret[i].clc_name.name);
}
```

数据类型和结构

Clinfo C++ API 使用 **clinfo.H** 文件中定义的数据类型和结构。

clinfo.H 文件还包括 **sys/types.h**、**netinet/in.h** 和 **clinfo.h** 文件。

基本数据类型和类定义:

这些数据类型将 **clinfo.h** 文件中定义的 C 数据类型转换为 C++。

```
typedef int CL_clusterid;
typedef int CL_nodeid;
typedef int CL_ifid;
typedef struct sockaddr_in CL_ifaddr;
typedef enum cls_state CL_state;
typedef enum clss_substate CL_substate;
typedef int CL_status;
typedef int CL_groupid;
typedef enum cl_rg_policies CL_rg_policies;
typedef enum cl_resource_states CL_resource_states;
class CL_clustername {public: char name[CL_MAXNAMELEN]; };
class CL_nodename {public: char name[CL_MAXNAMELEN]; };
class CL_ifname {public: char name[CL_MAXNAMELEN]; };
class CL_route {
public:
CL_ifaddr localaddr;
CL_ifaddr remoteaddr;
```

```
};
class CL_groupname {public: char name[CL_MAXNAMELEN]; };
class CL_user_policy_name {public: char name[CL_MAXNAMELEN]; };
```

集群对象类:

集群类数据和成员函数:

```
class CL_cluster  {
public:
CL_clusterid clc_clusterid;// Cluster Id
CL_state clc_state;// Cluster State
CL_substate clc_substate;// Cluster Substate
CL_nodename clc_primary; // Cluster Primary Node
CL_clustername clc_name; // Cluster Name
CL_node *clc_node; // Pointer to child node array
CL_group *clc_group; // pointer to child resource group array

int CL_getallinfo(CL_node*, CL_status&);
int CL_getgroupinfo(CL_group*, CL_status&);
CL_clusterid CL_getclusterid(CL_status&);
CL_cluster CL_getinfo(CL_status&);
CL_status CL_getprimary(CL_status&, CL_nodename);
CL_status CL_isavail();
CL_cluster& operator=(const struct cl_cluster&);
};
```

网络接口对象类:

网络接口类数据和成员函数。

```
class CL_netif  {
 public:
 CL_clusterid cli_clusterid; // Cluster Id
 CL_nodeid cli_nodeid;  // Cluster node Id
 CL_nodename cli_nodename; // Cluster node name
 CL_ifid cli_interfaceid; // Cluster Node Interface Id
 CL_state cli_state;  // Cluster Node Interface State
 CL_ifname cli_name;  // Cluster Node Interface Name
 CL_ifaddr cli_addr;  // Cluster Node Interface IP Address
 CL_node *cli_pnode;  // pointer to parent Node object
 CL_ifaddr6 cli_addr6;  // Cluster Node Interface IP Address

 CL_clusterid CL_getclusterid(CL_status&);
 CL_clusterid CL_getclusterid6(CL_status&);
 CL_ifaddr CL_getifaddr(CL_status&);
 CL_ifaddr6 CL_getifaddr6(CL_status &s);
  CL_ifname CL_getifname(CL_status&);
 CL_ifname CL_getifname6(CL_status &);
 CL_ifaddr CL_getnodeaddr(CL_status&);
 CL_ifaddr6 CL_getnodeaddr6(CL_status&);
 CL_nodename CL_getnodenamebyif(CL_status&);
 CL_nodename CL_getnodenamebyif6(CL_status &);
 CL_status CL_isavail();
 CL_status CL_isavail6();
 CL_netif& operator=(const struct cl_netif&);
};
```

节点对象类:

节点类数据和成员函数。

```
class CL_node  {
 public:
 CL_clusterid cln_clusterid; // Cluster Id
 CL_nodeid cln_nodeid;         // Cluster node id - used internally
```

```
 CL_nodename cln_nodename; // Cluster node name
 CL_state cln_state;  // Cluster Node State
 int cln_nif;   // Cluster Node Number of Interfaces
 CL_netif *cln_if;         // Cluster Node interfaces
 CL_cluster *cln_pcluster; // pointer to parent cluster object

 CL_route CL_bestroute(CL_status&);
 CL_route6 CL_bestroute6(CL_status&);
 CL_node CL_getinfo(CL_status&);
 CL_status CL_isavail();
 CL_node& operator=(const struct cl_node&);
};
```

注: 提供了指向对象类数据中包括的父对象的指针，以免您要设置对象的树结构。Clinfo C++ API 不会填写这些指针。

资源组对象类:

资源组数据和成员函数

```
class CL_group  {
    public:
    CL_clusterid clg_clusterid;  // Cluster Id
    CL_groupid clg_group_id   // Resource Group Id
    CL_groupname clg_name;    // Resource group name

    /* The following field is deprecated in PowerHA SystemMirror 5.2 and will not be
     * used. The data field itself is not removed from the data
     * structures to maintain the backward compatibility */

    CL_rg_policies clg_policy;// Resource Group Policy
    CL_rg_policies clg_startup_policy;
    CL_rg_policies clg_fallover_policy;
    CL_rg_policies clg_fallback_policy;
    CL_rg_policies clg_site_policy;     // Resource Group site policy
    CL_user_policy_name clg_user_policy_name;
 // User defined policy

    int clg_num_nodes;
    int clg_node_ids[MAXNODES];  // Node ids in this group

    CL_resource_states clg_node_states[MAXNODES];
//and their state
    CL_cluster *cln_pcluster;
// pointer to parent cluster object
    CL_group CL_getinfo(CL_status&);
    CL_group& operator=(const struct cl_group&);
};
```

注: 提供了指向对象类数据中包括的父对象的指针，以免您要设置对象的树结构。Clinfo C++ API 不会填写这些指针。

未包括在任何对象类中的函数:

有一些函数未包括在任何对象类中。

```
int CL_getallinfo (CL_cluster *, CL_status&);
```

此函数不是类 **CL_cluster** 的成员函数，这是因为它将返回集群数以及有关所有集群而不是特定集群对象的信息。

```
CL_node CL_getlocalid(CL_status&);
```

此函数不是类 **CL_node** 的成员函数，这是因为它将返回有关本地主机的信息。

分配类 CL_netif 数据: cli_addr 和 cli_name

此代码示例说明了如何分配网络名和地址。

多个节点（或一组节点）以及与每个节点相关联的接口分配存储器。

给定了 char *addr = "1.2.3.4" 时，要分配 cli_addr，请输入以下内容:

```
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
```

要分配 cli_name，请输入以下内容:

```
strcpy(netif.cli_name.name, "node_name");
```

重载的赋值运算符

将重载类 **CL_cluster**、**CL_netif**、**CL_group** 和 **CL_node** 的 = 赋值运算符，以便将 C 结构 **cl_cluster**、**cl_netif**、**cl_group** 和 **cl_node** 分配给对应的 C++ CL_ 类。

使用 Clinfo C API 调用的函数

下列函数尚未从 C 转换为 C++。

必须使用 Clinfo C API 调用这些函数:

事件函数

```
int cl_registereventnotify(int, struct cli_enr_req_t *);
int cl_unregistereventnotify(int, struct cli_enr_req_t *);
int cl_getevent(cli_en_msg_t *);
```

实用程序函数

```
void cl_perror(int, char *);
char *cl_errmsg (int status);
    /*for single-threaded applications*/
char *cl_errmsg_r(int status, char cbuf);
    /*for multi-threaded applications*/
```

将应用程序从更低 Clinfo 发行版升级

更低的 Clinfo 发行版使用整数（节点标识）而不是字符串（节点名）来标识集群节点。

这些章节提供了一些示例，以说明如何转换应用程序中对先前使用了 *nodeid* 参数的各种 Clinfo C++ API 例程的调用。对于每个例程，在它与节点标识配合使用的示例后面都有一个使用节点名的示例。

CL_getlocalid:

这些是来自于更低发行版和更高版本的 **CL_getlocalid** 例程示例。

以下是来自于使用节点标识的更低发行版的 **CL_getlocalid** 例程示例。

```
CL_status s;
CL_node lnode;lnode = node.CL_getlocalid(s);
if (s < 0)
cl_errmsg(s);
printf("cluster id = %d, node id = %d", lnode.cln_clusterid,
lnode.cln_nodeid);
```

在新版本的 C++ API **CL_getlocalid** 例程示例中，请注意 **printf** 语句中的更改。

```
// This function is not a member of a class.

CL_status s;
CL_node lnode;
char cbuf[CL_ERRMSG_LEN];
lnode = CL_getlocalid(s);
if (s < 0)
cl_errmsg_r(s, cbuf);
printf("cluster id = %d, node name = %s", lnode.cln_clusterid,
lnode.cln_nodename.name);
```

***CL_isavail*:**

这些是来自于更低发行版和更高版本的 **CL_isavail** 例程示例。

以下是来自于使用节点标识的更低发行版的 **CL_isavail** 例程示例。

```
CL_status s;
CL_netif netif;
netif.cli_clusterid = 2;
netif.cli_nodeid = 2;
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
s = netif.CL_isavail();

printf("status = %d", s);
```

新版本的 C++ API **CL_isavail** 例程示例更改了赋值语句，这是因为先前版本使用 *cli_nodeid*；现在它是 *cln_name.name*。

```
CL_status s;
CL_netif netif;
netif.cli_clusterid = 2;
strcpy(netif.cln_name.name, "moby");
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
s = netif.CL_isavail();

printf("status = %d", s);
```

CL_getprimary

这些是来自于更低发行版和更高版本的 **CL_getprimary** 例程示例。

以下是来自于使用节点标识的更低发行版的 **CL_getprimary** 例程示例:

```
CL_cluster clus;
CL_status s;
CL_nodeid nid;
clus.clc_clusterid = 2;
nid = clus.CL_getprimary(s);
if (s < 0)
cl_errmsg(s);
printf("nodeid = %d", nid);
```

新版本的 C++ API **CL_getprimary** 例程示例已更改，这是因为主节点现在按其名称（字符串）而不是按整数进行识别。

```
CL_clusterid clusterid;
CL_cluster clus;
CL_status status;
CL_nodename name;
CL_node node;
char cbuf[CL_ERRMSG_LEN];
clus.clc_clusterid = 2;
```

```
status = clus.CL_getprimary(cluster.clc_clusterid);
if (status < 0)
cl_errmsg_r(status, cbuf);
printf( "Cluster %d's primary node is %s",
cluster.clc_clusterid, cluster.clc_primary.name);
```

请求

Clinfo C++ API 具有不同类型的请求。

集群信息请求

这些集群信息请求将返回有关集群的信息。

项	描述
CL_getallinfo	调用 C 函数 **cl_getnodemap**。此请求是与 **cl_cluster** 类相关联的成员函数。给定了集群标识时，此请求将返回有关集群中所有节点的信息。指向节点对象数组的指针将作为引用自变量传递到该函数。
CL_getallinfo	调用 C 函数 **cl_getclusters**。返回已找到的集群的数目。指向集群对象数组的指针将作为引用自变量传递到该函数。（调用 **CL_getallinfo** 以获取有关所有集群的信息时，该函数不是类 **CL_cluster** 的成员函数。）
CL_getclusterid	调用 C 函数 **cl_getclusterid**。给定了集群名称时，将返回集群标识。
CL_getinfo	调用 C 函数 **cl_getcluster**。返回有关具有所指定集群标识的集群的信息。
CL_getprimary	调用 C 函数 **cl_getprimary**。返回所指定集群中用户定义的主集群管理器的节点名。
CL_isavail	调用 C 函数 **cl_isclusteravail**。返回具有所指定集群标识的集群的状态。
CL_getgroupinfo	调用 C 函数 **cl_getgroupmap**。返回已找到的资源组的数目。指向资源组对象数组的指针将作为引用对象进行传递。

节点信息请求

这些节点信息请求将返回有关集群中节点的信息:

项	描述
CL_bestroute	调用 **cl_bestroute** C 函数。返回一对本地或远程 IPv4 地址，这一对地址提供了到所指定节点的最直接路由。
CL_bestroute6	调用 **cl_bestroute6** C 函数。返回一对本地/远程 IPv4 或 IPv6 地址，这一对地址提供了到所指定节点的最直接路由。
CL_getinfo	调用 **cl_getnode** C 函数。返回有关由一对集群标识或节点名指定的节点的信息。
CL_getlocalid	调用 **cl_getlocalid** C 函数。返回 **CL_node** 对象，该对象包含发出请求的主机的集群标识和节点名。（此函数不是 **CL_node** 类的成员函数）
CL_isavail	调用 **cl_isnodeavail** C 函数。给定了所指定节点的集群标识和节点名时，将返回该节点的状态。

网络接口信息请求

这些网络接口信息请求将返回有关已连接至节点的接口的信息。

项	描述
CL_getclusterid	调用 **cl_getclusteridbyifaddr** 或 **cl_getclusteridbyifname** C 函数。如果指定了 IPv4 地址，那么将返回集群的网络接口名称；如果指定了名称，那么将返回集群的 IPv4 网络接口地址。
CL_getclusterid6	调用 **cl_getclusteridbyifaddr6** 或 **cl_getclusteridbyifname** C 函数。如果指定了 IPv4 或 IPv6 地址，那么将返回集群的网络接口名称；如果指定了名称，那么将返回集群的 IPv4 或 IPv6 网络接口地址。
CL_getifaddr	调用 **cl_getifaddr** C 函数。返回具有所指定集群标识和网络接口名称的接口的 IPv4 网络接口地址。
CL_getifaddr6	调用 **cl_getifaddr6** C 函数。返回具有所指定集群标识和网络接口名称的接口的 IPv4 或 IPv6 网络接口地址。
CL_getifname	调用 **cl_getifname** C 函数。返回具有所指定集群标识和 IPv4 网络接口地址的接口的名称。
CL_getifname6	调用 **cl_getifname6** C 函数。返回具有所指定集群标识和 IPv4 或 IPv6 网络接口地址的接口的名称。
CL_getnodeaddr	调用 **cl_getnodeaddr** C 函数。返回与所指定集群标识和网络接口名称相关联的 IPv4 网络接口地址。
CL_getnodeaddr6	调用 **cl_getnodeaddr6** C 函数。返回与所指定集群标识和网络接口名称相关联的 IPv4 和 IPv6 网络接口地址。
CL_getnodenamebyif	调用 **cl_getnodenamebyifaddr** 或 **cl_getnodenamebyifname** C 函数。如果 IPv4 接口地址是已知的，那么将通过调用 **cl_getnodenamebyifaddr** 返回节点名，否则将通过调用 **cl_getnodenamebyifname** 返回节点名。
CL_getnodenamebyif6	调用 **cl_getnodenamebyifaddr6** 或 **cl_getnodenamebyifname** C 函数。如果 IPv4 或 IPv6 接口地址是已知的，那么将通过调用 **cl_getnodenamebyifaddr6** 返回节点名，否则将通过调用 **cl_getnodenamebyifname** 返回节点名。
CL_isavail	调用 **cl_isaddravail** C 函数。给定了所指定网络接口的集群标识、节点名和 IPv4 网络接口地址时，将返回该网络接口的状态。
CL_isavail6	调用 **cl_isaddravail6** C 函数。给定了所指定网络接口的集群标识、节点名和 IPv4 或 IPv6 网络接口地址时，将返回该网络接口的状态。

资源组信息请求

这些资源组信息请求将返回有关集群资源组的信息。

项	描述
CL_getinfo	返回有关所指定集群中特定资源组的所有信息。

事件通知请求

这些事件通知例程将返回有关集群事件、节点事件或网络事件的信息。必须从 Clinfo C API 中调用这些例程。

项	描述
cl_getevent	在接收到事件信号时获取信息，并返回从 Clinfo 接收到的事件通知消息。
cl_registereventnotify	向 Clinfo 注册事件通知请求列表，并在发生已注册的事件时返回一个信号到主调进程。
cl_unregistereventnotify	向 Clinfo 注销事件通知请求列表。

CL_cluster::CL_getallinfo 例程

返回有关集群中所有节点的信息。

语法

```
int *CL_cluster::CL_getallinfo (CL_node *nodes, CL_status s)
```

必需的输入对象数据

项	描述
CL_cluster::clc_clusterid	期望集群的集群标识。

返回值

项	描述
nodes	指向节点对象数组的指针将作为引用自变量传递到该函数。
int	集群中节点的数目。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
CL_cluster cluster;
CL_status status;
CL_node nodes[8];
CL_node *ret = &nodes[0];
int numnodes;

cluster.clc_clusterid = 1113325332;
numnodes = cluster.CL_getallinfo(ret, status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("number of nodes in cluster: %d\n", numnodes);
  for (int i=0; i<numnodes; i++) {
      printf("[%d] clusterid %d", i, ret[i].cln_clusterid);
      printf(" nodeid %d", ret[i].cln_nodeid);
      printf(" state %d", ret[i].cln_state);
      printf(" interfaces %d\n", ret[i].cln_nif);
  }
}
```

CL_getlocalid 例程

返回 **CL_node** 对象，该对象包含发出请求的主机的集群标识和节点名。此请求将返回集群中当前未处于活动状态的节点的错误状态。

语法

```
CL_node CL_getlocalid (CL_status s)
```

必需的输入对象数据

无。

返回值

项	描述
CL_node	具有本地集群和节点名的节点对象。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVNODE	节点不是集群节点。

示例

此示例使用 **cl_errmsg** 例程来说明正确的单线程应用程序编程。如果您的程序是多线程程序，那么必须使用 **cl_errmsg_r** 例程。

```
  CL_status status;
  CL_node lnode;

  lnode = CL_getlocalid(status);
  if (status < 0) {
      cl_errmsg(status);
  } else {
      printf("cluster id = %d, node name = %s\n", lnode.cln_clusterid,
lnode.cln_nodename.name);
  }
```

CL_cluster::CL_getallinfo 例程

返回有关集群中所有节点的信息。

语法

```
int *CL_cluster::CL_getallinfo (CL_node *nodes, CL_status s)
```

必需的输入对象数据

项	描述
CL_cluster::clc_clusterid	期望集群的集群标识。

返回值

项	描述
nodes	指向节点对象数组的指针将作为引用自变量传递到该函数。
int	集群中节点的数目。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
CL_cluster cluster;
CL_status status;
CL_node nodes[8];
CL_node *ret = &nodes[0];
int numnodes;

cluster.clc_clusterid = 1113325332;
numnodes = cluster.CL_getallinfo(ret, status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("number of nodes in cluster: %d\n", numnodes);
  for (int i=0; i<numnodes; i++) {
      printf("[%d] clusterid %d", i, ret[i].cln_clusterid);
      printf(" nodeid %d", ret[i].cln_nodeid);
      printf(" state %d", ret[i].cln_state);
      printf(" interfaces %d\n", ret[i].cln_nif);
  }
}
```

CL_cluster::CL_getclusterid 例程

CL_getclusterid 例程将返回具有所指定名称的集群的集群标识。

语法

```
CL_clusterid CL_cluster::CL_getclusterid(CL_status s)
```

必需的输入对象数据

项	描述
CL_cluster::clc_name	目标集群的名称。

返回值

项	描述
CL_clusterid	期望的集群标识。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERNAME	该请求指定了无效集群名称。

示例

```
    CL_cluster cluster;
    CL_status status;
    CL_clusterid clusterid;
    char cbuf[CL_ERRMSG_LEN];

    strcpy(cluster.clc_name.name, "site1");
    clusterid = cluster.CL_getclusterid(status);
    if (status < 0) {
  cl_errmsg(status);
    } else {
  printf("clusterid = %d\n", clusterid);
    }
}
```

CL_group::CL_getinfo 例程

给定了具有集群标识和组名的组对象时，将返回包含有关组的信息的组对象。

语法

```
CL_Group::CL_getinfo (CL_status s);
```

必需的输入对象数据

项	描述
CL_group::clg_clusterid	期望资源组的集群标识。
CL_group::clg_name.name	资源组的名称。

返回值

项	描述
CL_group	期望的资源组及其完整信息。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
CL_status status;
CL_group group;
CL_group ret;

group.clg_clusterid = 1113325332;
strcpy(group.clg_name.name, "rg01");
ret = group.CL_getinfo(status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("There are %d nodes in group %s\n",
  ret.clg_num_nodes, ret.clg_name.name);
}
```

CL_cluster::CL_getprimary 例程

返回所指定集群的用户所指定主集群管理器的节点名。

语法

```
CL_nodename CL_cluster::CL_getprimary (CL_status s)
```

必需的输入对象数据

项	描述
CL_cluster::clc_clusterid	期望获得其主集群管理器标识的集群标识。

返回值

项	描述
CL_nodename	主集群管理器的节点名。

状态值

项	描述
CL_status status	按引用传递的状态。返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_NOPRIMARY	未获得主集群管理器信息。此状态通常指示用户未指定主集群管理器。
CLE_IVCLUSTER	该集群不可用。

示例

```
CL_cluster cluster;
CL_status status;
CL_nodename name;

cluster.clc_clusterid = 1;
name = cluster.CL_getprimary(status);
if (status < 0) {
    cl_errmsg(status);
} else {
    printf( "cluster %d's primary node is %s\n",
cluster.clc_clusterid, cluster.clc_primary.name);
}
```

CL_cluster::CL_isavail 例程

如果指定的集群可用，那么将返回状态码 CLE_OK。

语法

```
CL_status CL_cluster::CL_isavail()
```

必需的输入对象数据

项	描述
CL_cluster::clc_clusterid	期望集群的集群标识。

返回值

项	描述
CL_status	所指定集群的状态。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_IVCLUSTER	该请求指定了无效集群。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
CL_status status;
CL_cluster cluster;

cluster.clc_clusterid = 1113325332;
status = cluster.CL_isavail();
printf("status = %d\n", status);
```

CL_cluster::CL_getgroupinfo 例程

返回有关集群中所有资源组的信息。

语法

```
int CL_cluster::CL_getgroupinfo (CL_group *groups, CL_status&);
```

必需的输入对象数据

项	描述
CL_cluster::clc_clusterid	期望集群的集群标识。

返回值

项	描述
groups	指向一组资源组对象的指针将作为引用自变量传递到该函数。
int	集群中资源组的数目

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。

示例

```
CL_status status;
CL_cluster cluster;
CL_group groups[MAXGROUPS];
int numgroups;

cluster.clc_clusterid = 1113325332;
numgroups = cluster.CL_getgroupinfo(&groups[0], status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("There are %d groups in cluster %d\n", numgroups,
  cluster.clc_clusterid);
  for (int i=0; i<numgroups; i++) {
    printf("Group %d is id %d\n", i, groups[i].clg_group_id);
    printf("Group %d is %s and has %d nodes\n",
    i, groups[i].clg_name.name, groups[i].clg_num_nodes);
  }
}
```

CL_group::CL_getinfo 例程

给定了具有集群标识和组名的组对象时，将返回包含有关组的信息的组对象。

语法

```
CL_Group::CL_getinfo (CL_status s);
```

必需的输入对象数据

项	描述
CL_group::clg_clusterid	期望资源组的集群标识。
CL_group::clg_name.name	资源组的名称。

返回值

项	描述
CL_group	期望的资源组及其完整信息。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
CL_status status;
CL_group group;
CL_group ret;

group.clg_clusterid = 1113325332;
strcpy(group.clg_name.name, "rg01");
ret = group.CL_getinfo(status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("There are %d nodes in group %s\n",
  ret.clg_num_nodes, ret.clg_name.name);
}
```

CL_netif::CL_getclusterid 例程

返回具有所指定网络接口地址的集群的名称。或者，给定了集群的网络接口名称时，将返回集群的网络接口地址。

语法

```
CL_clusterid CL_netif::CL_getclusterid (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_addr	期望获得其集群标识的网络接口地址。

或

项	描述
CL_netif::cli_name	期望获得其集群标识的网络接口名称。

返回值

项	描述
CL_clusterid	集群标识（非负整数）。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
// example using interface name

CL_status status;
CL_clusterid clid;
CL_netif netif;

strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr.sin_addr.s_addr = NULL;
clid = netif.CL_getclusterid(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("clusterid = %d\n", clid);
}

// example using interface address

  CL_status status;
  CL_clusterid clusterid;
  CL_netif netif;
  char *addr = "1.1.1.7";

  netif.cli_addr.sin_family = AF_INET;
  netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
  netif.cli_name.name[0] = NULL;
  clusterid = netif.CL_getclusterid(status);
  if (status < 0) {
cl_errmsg(status);
  } else {
printf("clusterid = %d\n", clusterid);
  }
```

CL_netif::CL_getclusterid6 例程

返回具有所指定网络接口地址的集群的名称。或者，给定了集群的网络接口名称时，将返回集群的网络接口地址。此例程能够处理 IPv4 地址和 IPv6 地址。

语法

```
CL_clusterid CL_netif::CL_getclusterid6(CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_addr6	期望获得其集群标识的网络接口地址。

或

项	描述
CL_netif::cli_name	期望获得其集群标识的网络接口名称。

返回值

项	描述
CL_clusterid	集群标识（非负整数）。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
CL_clusterid clid;
CL_netif netif;

strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr6.sin6_addr.s6_addr = NULL;
clid = netif.CL_getclusterid6(status);
if (status < 0)
{
cl_errmsg(status);
}
else
{
printf("clusterid = %d\n", clid);
}

// example using interface address

CL_status status;
CL_clusterid clusterid;
CL_netif netif;
char *addr = "fe80::1";

((struct sockaddr_in6)netif.cli_addr6).sin6_family = AF_INET6;
inet_pton (AF_INET6, addr,
 &(((struct sockaddr_in6 *)&netif.cli_addr6)->sin6_addr));
netif.cli_name.name[0] = NULL; clusterid = netif.CL_getclusterid6(status);
if (status < 0)
{
cl_errmsg(status);
}
```

```
else
{
printf("clusterid = %d\n", clusterid);
}
```

CL_netif::CL_getifaddr 例程

返回具有所指定集群标识和网络接口名称的接口的网络接口地址。此例程只能处理 IPv4 地址。

语法

```
CL_ifaddr CL_netif::CL_getifaddr (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_name	期望网络接口的名称。

返回值

项	描述
CL_ifaddr	期望的网络接口地址。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
CL_ifaddr ifaddr;
CL_netif netif;
char cbuf[CL_ERRMSG_LEN];
netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getifaddr(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("ifaddr = %s\n", inet_ntoa(ifaddr.sin_addr));
}
```

CL_netif::CL_getifaddr6 例程

返回具有所指定集群标识和网络接口名称的接口的网络接口地址。此例程能够处理 IPv4 地址和 IPv6 地址。

语法

```
CL_ifaddr6 CL_netif::CL_getifaddr6 (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_name	期望网络接口的名称。

返回值

项	描述
CL_ifaddr6	期望的网络接口地址。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
CL_ifaddr6 ifaddr;
CL_netif netif;
char cbuf[CL_ERRMSG_LEN];
char *addr;
netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getifaddr6(status);
if (status < 0)
{
cl_errmsg(status);
}
else
{
printf("ifaddr = %s\n", inet_ntop(AF_INET6,
&((struct sockaddr_in6 *)&ifaddr->sin6_addr), addr, INET6_ADDRSTRLEN);
}
```

CL_netif::CL_getifname 例程

给定了集群标识和网络接口地址时，将返回网络接口名称；或者给定了集群标识和节点名时，将返回网络接口名称。此例程只能处理 IPv4 地址。

如果该请求指定了 **cli_addr** 参数，那么 Clinfo 会检查地址的网络部分，然后在同一网络上寻找接口。如果找到了匹配项，那么 **CL_getifname** 例程将返回与该接口相关联的名称。

如果 **cli_addr** 为 NULL，那么 Clinfo 将在指定的节点上选择最容易从本地主机访问的接口，并且 **CL_getifname** 例程将返回与该接口相关联的名称。如果有两个接口从本地主机访问的容易程度相同，那么 Clinfo 会选择其中一个接口，然后返回与该接口相关联的名称。

在所有情况下，**CL_getifname** 例程都将采用以 NULL 结束的字符串的形式返回该名称。

语法

```
CL_ifname CL_netif::CL_getifname (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid 和 cli_addr	期望网络接口的集群标识和网络接口地址。

或

项	描述
CL_netif::cli_clusterid 和 cli_nodename	期望网络接口的集群标识和节点名。

返回值

项	描述
CL_ifname	网络接口名称。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
// CL_netif::CL_getifname get interfacename given clusterid and nodename

  CL_status status;
  char cbuf[CL_ERRMSG_LEN];
  CL_ifname ifname;
  CL_netif netif;

  netif.cli_addr.sin_addr.s_addr = NULL;
  netif.cli_clusterid = 1113325332;
  strcpy (netif.cli_nodename.name, "node1");
  ifname = netif.CL_getifname(status);
  if (status < 0) {
     cl_errmsg(status);
  } else {
     printf("ifname = %s\n", ifname.name);
  }
```

CL_netif::CL_getifname6 例程

给定了集群标识和网络接口地址时，将返回网络接口名称；或者给定了集群标识和节点名时，将返回网络接口名称。此例程能够处理 IPv4 地址和 IPv6 地址。

如果该请求指定了 **cli_addr6** 参数，那么 Clinfo 会检查地址的网络部分，然后在同一网络上寻找接口。如果找到了匹配项，那么 **CL_getifname6** 例程将返回与该接口相关联的名称。

如果 **cli_addr6** 为 NULL，那么 Clinfo 将在指定的节点上选择最容易从本地主机访问的接口，并且 **CL_getifname6** 例程将返回与该接口相关联的名称。如果有两个接口从本地主机访问的容易程度相同，那么 Clinfo 会选择其中一个接口，然后返回与该接口相关联的名称。

在所有情况下，**CL_getifname6** 例程都将采用以 NULL 结束的字符串的形式返回该名称。

语法

```
CL_ifname CL_netif::CL_getifname6 (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid 和 cli_addr6	期望网络接口的集群标识和网络接口地址。

或

项	描述
CL_netif::cli_clusterid 和 cli_nodename	期望网络接口的集群标识和节点名。

返回值

项	描述
CL_ifname	网络接口名称。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
// CL_netif::CL_getifname get interfacename given clusterid and nodename

CL_status status;
char cbuf[CL_ERRMSG_LEN];
CL_ifname ifname;
CL_netif netif;

netif.cli_addr6.sin6_addr.s6_addr = NULL;
netif.cli_clusterid = 1113325332;
strcpy (netif.cli_nodename.name, "node1");
ifname = netif.CL_getifname6(status);
if (status < 0)
{
```

```
cl_errmsg(status);
}
else
{
printf("ifname = %s\n", ifname.name);
}
```

CL_netif::CL_getnodeaddr 例程

返回与所指定集群标识和网络接口名称相关联的 IP 地址。此例程只能处理 IPv4 地址。

语法

```
CL_ifaddr CL_netif::CL_getnodeaddr(CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_name	期望网络接口的名称。

返回值

项	描述
CL_ifaddr	网络接口地址。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
CL_netif netif;
CL_ifaddr ifaddr;
char cbuf[CL_ERRMSG_LEN];

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getnodeaddr(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("ifaddr = %s\n", inet_ntoa(ifaddr.sin_addr));
}
```

CL_netif::CL_getnodeaddr6 例程

返回与所指定集群标识和网络接口名称相关联的 IP 地址。此例程能够处理 IPv4 地址和 IPv6 地址。

语法

```
CL_ifaddr6 CL_netif::CL_getnodeaddr6(CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_name	期望网络接口的名称。

返回值

项	描述
CL_ifaddr6	网络接口地址。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
CL_netif netif;
CL_ifaddr6 ifaddr;
char cbuf[CL_ERRMSG_LEN];
char *addr;

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getnodeaddr6(status);
if (status < 0)
{
cl_errmsg(status);
}
else
{
printf("ifaddr = %s\n", inet_ntop(AF_INET6, &((struct sockaddr_in6 *)
&ifaddr->sin6_addr), addr, INET6_ADDRSTRLEN);
}
```

CL_netif::CL_getnodenamebyif 例程

给定了集群标识和网络接口地址或集群标识和网络接口名称时，将返回节点名。

如果指定了网络接口地址，并且 **cli_name** 为 NULL，那么将返回 **cli_name**。相反，如果给定了 **cli_name**，并且 **cli_addr** 为 NULL，那么将返回 **cli_addr**。如果 **cli_name** 和 **cli_addr** 都不为 NULL，那么 **cli_addr** 优先。如果都为 NULL，那么将返回错误代码 CLE_BADARGS。

注：此例程替换先前发行版中可用的 **CL_getnodename** 例程。

语法

```
CL_nodename CL_netif::CL_getnodenamebyif (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid 和 cli_addr	期望节点的集群标识和该节点的网络接口地址。

或

项	描述
CL_netif::cli_clusterid 和 cli_name	期望节点的集群标识和网络接口的名称。

返回值

项	描述
CL_nodename	节点名。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
char cbuf[CL_ERRMSG_LEN];
CL_nodename nname;
CL_netif netif;
char *addr = "9.57.28.23";

netif.cli_clusterid = 1113325332;
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
netif.cli_name.name[0] = NULL;
nname = netif.CL_getnodenamebyif(status);
if (status < 0) {
   cl_errmsg(status);
} else {
   printf("node name = %s\n", nname.name);
}
```

CL_netif::CL_getnodenamebyif6 例程

给定了集群标识和网络接口地址或集群标识和网络接口名称时，将返回节点名。此例程能够处理 IPv4 地址和 IPv6 地址。

如果指定了网络接口地址，并且 **cli_name** 为 NULL，那么将返回 **cli_name**。相反，如果给定了 **cli_name**，并且 **cli_addr6** 为 NULL，那么将返回 **cli_addr6**。如果 **cli_name** 和 **cli_addr6** 都不为 NULL，那么 **cli_addr6** 优先。如果都为 NULL，那么将返回错误代码 CLE_BADARGS。

注: 此例程替换先前发行版中可用的 **CL_getnodename** 例程。

语法

```
CL_nodename CL_netif::CL_getnodenamebyif6 (CL_status s)
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid 和 cli_addr6	期望节点的集群标识和该节点的网络接口地址。

或

项	描述
CL_netif::cli_clusterid 和 cli_name	期望节点的集群标识和网络接口的名称。

返回值

项	描述
CL_nodename	节点名。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVADDRESS	该请求指定了无效网络接口地址。
CLE_IVNETIFNAME	该请求指定了无效网络接口名称。

示例

```
CL_status status;
char cbuf[CL_ERRMSG_LEN];
CL_nodename nname;
CL_netif netif;
char *addr = "fe80::1";

netif.cli_clusterid = 1113325332;
netif.cli_addr6.sin6_family = AF_INET6;
inet_pton (AF_INET6, addr,
 &(((struct sockaddr_in6 *)&netif.cli_addr6)->sin6_addr));
netif.cli_name.name[0] = NULL;
nname = netif.CL_getnodenamebyif6(status);
if (status < 0)
{
cl_errmsg(status);
}
```

```
else
{
printf("node name = %s\n", nname.name);
}
```

CL_netif::CL_isavail 例程

如果指定的网络接口可用，那么将返回状态码 CLE_OK。此例程只能处理 IPv4 地址。

语法

```
CL_status CL_netif::CL_isavail()
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_nodename	期望网络接口的节点名。
CL_netif::cli_addr	期望网络接口的地址。

返回值

项	描述
CL_status	所指定网络接口的状态。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVADDRESS	该请求指定了无效接口地址。
CLE_IVNETIF	该网络接口不可用。

示例

```
CL_status status;
CL_netif netif;
char *addr = "9.57.28.23";

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
strcpy(netif.cli_nodename.name, "node1");
status = netif.CL_isavail();
if (status < 0) {
     cl_perror(status,"netif.CL_isavail failed");
}
printf("status = %d\n", status);
```

CL_netif::CL_isavail6 例程

如果指定的网络接口可用，那么将返回 CLE_OK 状态码。此例程能够处理 IPv4 地址和 IPv6 地址。

语法

```
CL_status CL_netif::CL_isavail6()
```

必需的输入对象数据

项	描述
CL_netif::cli_clusterid	期望网络接口的集群标识。
CL_netif::cli_nodename	期望网络接口的节点名。
CL_netif::cli_addr6	期望网络接口的地址。

返回值

项	描述
CL_status	所指定网络接口的状态。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVADDRESS	该请求指定了无效接口地址。
CLE_IVNETIF	该网络接口不可用。

示例

```
CL_status status;
CL_netif netif;
char *addr = "fe80::1";

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr6.sin6_family = AF_INET6;
inet_pton (AF_INET6, addr,
 &(((struct sockaddr_in6 *)&netif.cli_addr6)->sin6_addr));
strcpy(netif.cli_nodename.name, "node1");
status = netif.CL_isavail6();
if (status < 0)
{
cl_perror(status,"netif.CL_isavail6 failed");
}
printf("status = %d\n", status);
```

CL_node::CL_bestroute 例程

CL_bestroute 例程将返回一对本地/远程 IP 地址，这一对地址表示到对象中指定的节点的最直接路由。此例程只能处理 IPv4 地址。

CL_bestroute 例程所返回的路由取决于发出请求的节点。Clinfo 首先构建一个由本地节点上的所有工作网络接口组成的列表，然后将此列表与所指定节点上的可用接口进行比较。该例程首先将 PowerHA SystemMirror 定

义的专用接口（例如串行光学通道）与本地接口进行比较。如果找不到匹配项，那么该例程将 PowerHA SystemMirror 定义的公共接口与本地接口进行比较。如果仍然没有匹配项，那么该例程将选择本地节点上最先定义的接口以及远程节点上最先定义的接口。

如果存在一对位于同一网络上的本地和远程接口，那么将在 **CL_route** 中返回这些接口。否则，将选择所指定节点上的接口作为远程接口，并且将返回主本地接口作为路由的本地端。

语法

```
CL_route CL_node::CL_bestroute(CL_status s)
```

必需的输入对象数据

项	描述
CL_node::cln_clusterid 和 cln_nodename	目标节点的集群标识和节点名。

返回值

项	描述
CL_route	一对本地/远程 IP 地址，这一对地址指示到所指定节点的最直接路由。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_NOROUTE	无可用路由。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
CL_status status;
CL_node node;
CL_route route;
char cbuf[CL_ERRMSG_LEN];

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
route = node.CL_bestroute(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  // don't call inet_ntoa twice in one printf!
  printf("local = %s  ", inet_ntoa(route.localaddr.sin_addr));
  printf("remote = %s\n", inet_ntoa(route.remoteaddr.sin_addr));
}
```

CL_node::CL_bestroute6 例程

CL_bestroute6 例程将返回一对本地或远程 IP 地址，这一对地址表示到对象中指定的节点的最直接路由。此例程能够处理 IPv4 地址和 IPv6 地址。

CL_bestroute6 例程所返回的路由取决于发出请求的节点。Clinfo 首先构建一个由本地节点上的所有工作网络接口组成的列表，然后将此列表与所指定节点上的可用接口进行比较。该例程首先将 PowerHA SystemMirror 定义的专用接口（例如串行光学通道）与本地接口进行比较。如果找不到匹配项，那么该例程将 PowerHA SystemMirror 定义的公共接口与本地接口进行比较。如果仍然没有匹配项，那么该例程将选择本地节点上最先定义的接口以及远程节点上最先定义的接口。

如果一对本地和远程接口存在于同一网络上，那么将在 **CL_route6** 中返回这些接口。否则，将选择所指定节点上的接口作为远程接口，并且将返回主本地接口作为路由的本地端。

语法

```
CL_route CL_node::CL_bestroute6(CL_status s)
```

必需的输入对象数据

项	描述
CL_node::cln_clusterid 和 cln_nodename	目标节点的集群标识和节点名。

返回值

项	描述
CL_route6	一对本地或远程 IP 地址，这一对地址指示到所指定节点的最直接路由。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_NOROUTE	无可用路由。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
CL_status status;
CL_node node;
CL_route6 route;
char cbuf[CL_ERRMSG_LEN];
char *addr;

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
route = node.CL_bestroute6(status);
if (status < 0)
{
 cl_errmsg(status);
}
else
{
 // don't call inet_ntop twice in one
 printf("local = %s ", inet_ntop(AF_INET6,
```

```
 &(((struct sockaddr_in6 *)&(route.localaddr))->sin6_addr), addr, INET6_ADDRSTRLEN);
 printf("remote = %s\n", inet_ntop(AF_INET6,
 &(((struct sockaddr_in6 *)&(route.remoteaddr))->sin6_addr), addr,  INET6_ADDRSTRLEN);
}
```

CL_node::CL_getinfo 例程

给定了具有集群标识和节点名的节点对象时，将返回包含有关节点的信息的节点对象。

语法

```
CL_node CL_node::CL_getinfo(CL_status s)
```

必需的输入对象数据

项	描述
CL_node::cln_clusterid	目标节点的集群标识。
CL_node::cln_nodename	目标节点的节点名。

返回值

项	描述
CL_node	期望的节点及其信息。

状态值

项	描述
CL_status s	按引用传递的状态。用于存放返回码的输出参数。
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_BADARGS	缺少参数或参数无效。此状态通常指示对输出参数地址指定了 NULL 指针。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。

示例

```
CL_status status;
CL_node node;
CL_node ret;
char cbuf[CL_ERRMSG_LEN];

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
ret = node.CL_getinfo(status);
if (status < 0) {
    cl_errmsg(status);
} else {
    printf("clusterid %d  ", ret.cln_clusterid);
    printf("nodename %s  ", ret.cln_nodename.name);
    printf("state %d  ", ret.cln_state);
    printf("nif %d\n", ret.cln_nif);
}
```

CL_node::CL_isavail 例程

如果指定的节点可用，那么将返回状态码 CLE_OK。

语法

```
CL_status CL_node::CL_isavail()
```

必需的输入对象数据

项	描述
CL_node::cln_clusterid	期望节点的集群标识。
CL_node::cln_nodename	期望节点的节点名。

返回值

项	描述
CL_status	所指定节点的状态。

状态码

项	描述
CLE_OK	请求已成功完成。
CLE_SYSERR	系统错误。请检查 AIX 全局变量 **errno** 以了解更多信息。
CLE_IVCLUSTERID	该请求指定了无效集群标识。
CLE_IVNODENAME	该请求指定了无效节点名。
CLE_IVNODE	该节点不可用。

示例

```
CL_status status;
CL_node node;

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
status = node.CL_isavail();
printf("status = %d\n", status);
```

样本 Clinfo 客户机程序

本节列示样本 **clinfo.rc** 脚本以及从该脚本调用的 C 程序的源代码。此程序将报告给定集群节点上每个服务网络接口的状态以及该节点本身的状态。

样本定制 clinfo.rc 脚本

此处显示了样本 Clinfo 客户机应用程序 **cl_status**，它位于典型的定制 **clinfo.rc** 脚本的上下文中。**clinfo.rc** 是由 Clinfo 运行的 PowerHA SystemMirror for AIX 脚本，它遵循下列集群拓扑更改。对脚本和程序进行了注释以说明其用法。

```
#!/bin/ksh
################################################################################
# Filename: /usr/sbin/cluster/etc/clinfo.rc
#
# Description: clinfo.rc is run by clinfo on clients following cluster
#      topology changes. This particular example demonstrates
#      user process management for a highly available database in a
#      two-node primary/standby configuration. Most database
#      client programs are state-dependent, and require
       #restart following a node failure. This example provides
#      user notification and application shutdown during
```

```
#     appropriate topology changes.
#
#############################################################################
#############################################################################
# Grab Parameters Passed
#############################################################################
EVENT=$1 # action, one of {join, fail, swap}
INTERFACE=$2 # target address label
CLUSTERNAME="cluster1" # cluster name
NODENAME="victor"# primary node name
WATCHIF="svc_en0"# interface to monitor

#############################################################################
#   Name:  _arp_flush
#   This function flushes the entire arp cache.
#   Arguments:  none
#   Return value:  none
#############################################################################
_arp_flush()
{
  for IPADDR in $(/etc/arp -a |/bin/sed -e 's/^.*(.*).*$//' -e /incomplete/d)
    do
 /etc/arp -d $IPADDR
    done
}

#############################################################################
#
#   Name:  _kill_user_procs
#
#   This function kills user processes associated with the specified
#   interface.
#
#   Arguments:  interface
#   Return value:  none
#############################################################################
_kill_user_procs()
{
    print _kill_user_procs
  # place commands appropriate to the database in use here

}
# The main if statement disregards status changes for all interfaces except
# WATCHIF, which in this example is svc_en0.
if [[ "$INTERFACE" = "WATCHIF" ]]
then
  case "$EVENT" in
    "join") # interface label $INTERFACE has joined the cluster
# perform necessary activity here, such as user notification, restoration of
# user access, and arp cache flushing.
      exit 0
    ;;

    "fail")# Use api calls in cl_status to determine if interface
    # failure is a result of node failure.

  CLSTAT_MSG=$(cl_status $CLUSTERNAME $NODENAME)
  CLSTAT_RETURN=$?  # return code from cl_status

  case "$CLSTAT_RETURN" in
    0)      # Node UP
 # Notify users of application availability
      wall "Primary database is now available."
  # flush arp cache
      _arp_flush
;;
```

```
    1)      # Node DOWN
      # Notify users of topology change and restart requirement
      touch /etc/nologin    # prevent new logins
      wall "Primary database node failure. Please login again
  2 minutes"
  sleep 10
      # Kill all processes attached to WATCHIF interface
      _kill_user_procs $WATCHIF
      # flush arp cache
_ arp_flush
  rm -f /etc/nologin     # enable logins
  ;;

    *)  # Indeterminate node state
 # flush arp cache
      _arp_flush
exit 1
;;

  esac  # case $CLSTAT_RETURN
  ;;
    "swap")# interface has been swapped
    # flush arp cache.
      _arp_flush
    ;;

  esac      # case $EVENT
else
    # event handling for other interfaces here, if desired
    /bin/true
fi
```

cl_status.c 样本程序

这是样本 c 程序。

```
/*
 * Program:  cl_status.c
 *
 * Purpose:  For systems running the clinfo daemon as a client, cl_status
 * will determine if the node for the network interface passed
 * to it is active in the cluster.
 *
 * Usage:    [path/]cl_status clustername nodename
 *
 * Returns:  0 = Node up
 * 1 = Node down
 * 2 = ERROR - Status Unavailable
 *
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cluster/clinfo.h>
#include <strings.h>
void usage()
{
  printf("usage: cl_status clustername nodename");
  printf("Returns status of node in PowerHA SystemMirror cluster.");
}
int main(int argc, char *argv[])
{
  int clusterid, node_status;
  char *clustername, *nodename;

  if(argc < 3)
```

```
  {
    /* incorrect syntax to cl_status call */
    usage();
    exit(2);
  }
  clustername = strdup(argv[1]);
  if (strlen(clustername) > CL_MAXNAMELEN)
  {
    printf("error: clustername exceeds maximum length of %i characters",
     CL_MAXNAMELEN);
    exit(2);
  }
  nodename = strdup(argv[2]);
  if (strlen(nodename) > CL_MAXNAMELEN)
  {
    printf("error: nodename exceeds maximum length of %i characters",
     CL_MAXNAMELEN);
    exit(2);
  }
  /* convert clustername (string) to clusterid (non-negative integer) */
  clusterid = cl_getclusterid(clustername);
  switch(clusterid)
  {
    case CLE_SYSERR: perror("system error");
      exit(5);
      break;

    case CLE_NOCLINFO: cl_perror(clusterid, "error");
      exit(5);
      break;

    case CLE_BADARGS:
    case CLE_IVCLUSTERNAME: /* typically a usage error */
 cl_perror(clusterid, "error");
 usage();
  exit(2);

    default:    /* valid clusterid returned */
     ;
  }
  node_status = cl_isnodeavail(clusterid, nodename);
  switch (node_status)
  {
    case CLE_OK: /* Node up */

    printf("node %s up", nodename);
exit(0);
break;

case CLE_IVNODENAME:  /* "Illegal node name" */
      cl_perror(node_status, "node unavailable");
      exit(2);
      break;

    default:
      cl_perror(node_status, "node unavailable");
      exit(1);
  }
}
```

实现细节

有两个对 Clinfo 很重要的组件: **clinfo** 守护程序和 API 库。

clinfo 守护程序是基于 SNMP 的监视器。SNMP 是一组用于监视和管理基于 TCP/IP 的网络的业界标准。SNMP 包括一个协议、一个数据库规范和多组数据对象。

多组数据对象构成了管理信息库 (MIB)。SNMP 提供了包括诸如 IP 地址信息和活动 TCP 连接数目信息的标准 MIB。实际 MIB 定义被编码到在系统上运行的代理程序。 AIX 中的标准 SNMP 代理程序是 SNMP 守护程序 **snmpd**。

程序员使用 SNMP 操作来实现将监视和管理网络的程序。这些程序可以从 **snmpd** 接收有关网络状态的信息，然后将此信息传递到客户机和应用程序。

可以使用 SNMP 多路复用 (SMUX) 协议将 SNMP 扩展为包括*特定于企业的* MIB，这些 MIB 包含与独立环境或应用程序有关的信息。管理代理程序（SMUX 同级守护程序）将检索和维护有关在其 MIB 中定义的对象的信息，并对专用的网络监视器或网络管理站提供此信息。

PowerHA SystemMirror 软件通过集群管理器守护程序提供此 SMUX 同级功能。**clinfo** 守护程序通过集群管理器从 PowerHA SystemMirror MIB 间接检索此信息。

Clinfo API 库（包括所有变体）与 **clinfo** 守护程序进行交互，以提供对集群信息的访问权。虽然可以通过 SNMP 直接获得相同的信息，但 Clinfo 库提供了一个非常简化的编程模型，以使客户机程序能够避免 SNMP API 的复杂性。Clinfo API 提供了例程来检索与诸如节点或资源组之类的集群实体相关的所有信息（SNMP 要求您逐个访存这些项），并提供了例程来注册特定集群事件（SNMP 需要陷阱来实现类似的功能）。库的变体（C、C++ 和线程安全等等）为各种运行时环境提供了一致的模型。

clinfo 守护程序和库可以在 PowerHA SystemMirror 集群节点或非集群节点上运行，前提条件是该守护程序可以通过 TCP/IP 访问集群节点上的 SNMP。

集群管理器和 Clinfo

集群管理器守护程序 **(clstrmgr)** 是 PowerHA SystemMirror 子系统，它监视集群并在必要时启动恢复操作。集群管理器将报告集群行为，以便其他程序可以确定集群中是否发生了更改，并在必要时对那些更改作出响应。

一旦**集群管理器**获取了集群信息，它就会在 PowerHA SystemMirror for AIX MIB 中维护集群的已更新拓扑，这是因为它将跟踪事件和产生的集群状态。在客户端机器或集群节点上运行的 Clinfo 将查询 **MIB** 以获取已更新的集群信息，并通过应用程序编程接口授予应用程序对 PowerHA SystemMirror for AIX MIB 信息的访问权。

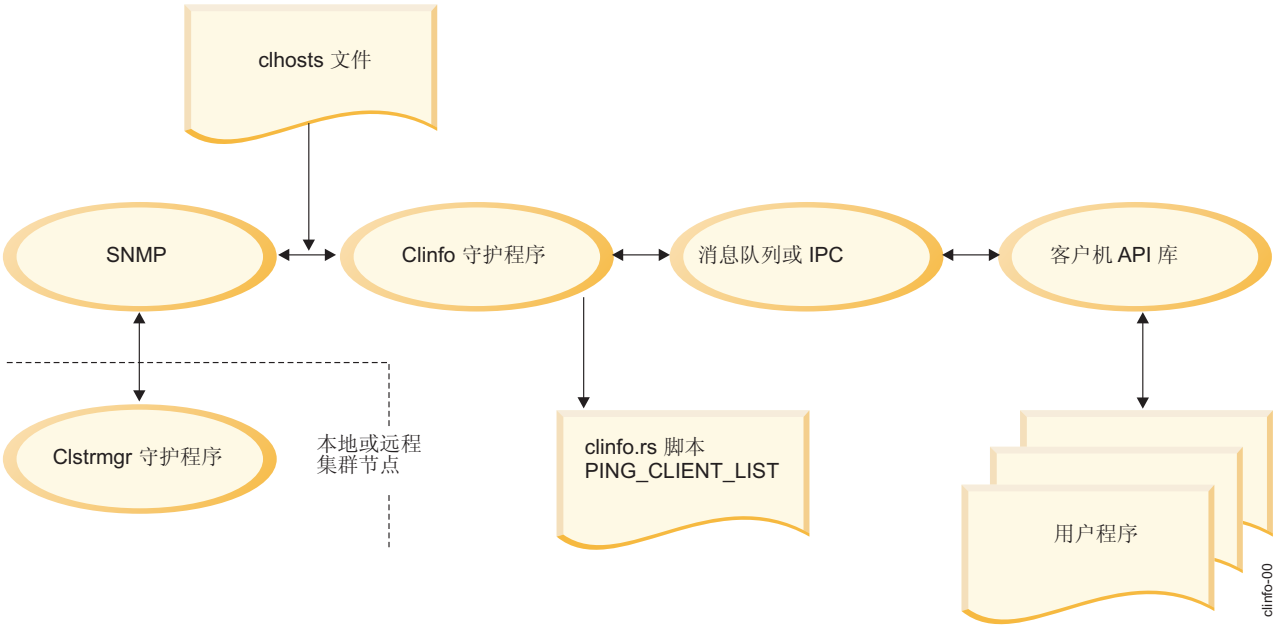
缺省情况下，Clinfo 将定期轮询 **SNMP 进程**以获取有关事件的已更新信息（每隔 15 秒轮询一次）。可以使用选项 (**-a**) 来启动 Clinfo，该选项使 Clinfo 能够在事件一发生时就接收到此信息。在这种情况下，**集群管理器**在接收到事件信息时会发送陷阱消息。然后 Clinfo 立即查询 **MIB** 以获取事件信息，而不会等待下一个轮询周期。

注: 使用 **-a** 选项启动 Clinfo 时，不能运行 NetView® for AIX 或任何其他预计会接收 SNMP 陷阱消息的应用程序。

当 Clinfo 启动时，它会读取 **/usr/sbin/cluster/etc/clhosts** 文件。此文件列示每个相关集群中所有可用节点的服务网络接口 IP 地址或 IP 标签。Clinfo 通过此文件搜索节点上的活动 **SNMP** 进程，将从 **clhosts** 文件中的第一个 IP 地址开始。一旦找到 **SNMP** 进程，Clinfo 就会从该 **SNMP** 进程接收到有关集群的拓扑和状态的信息。

如果此连接中断（例如，节点关闭），那么 Clinfo 会尝试与另一个节点的 SNMP 进程建立连接。一旦 Clinfo 从它首先与其建立通信的 **SNMP 进程**接收到集群信息，它就会将集群拓扑信息存放在内部（即，存放在本地节点上动态分配的数据结构中）。因此，它知道集群中其他节点的情况。

下图显示了集群管理器、Clinfo 以及集群节点之间的关系。



要使 Clinfo 按期望方式工作，**clhosts** 文件必须包含 Clinfo 可以与其通信的所有 PowerHA SystemMirror 服务器和客户机节点的 IP 地址。Clinfo 守护程序通过 SNMP 从 PowerHA SystemMirror 服务器节点（正在运行集群管理器守护程序 (**clstrmgr**) 的节点）检索信息。在启动期间，**clinfo** 守护程序按如下所示读入 **clhosts** 文件，以确定哪些节点可以通过 SNMP 进行通信:

- 对于与 **clstrmgr** 守护程序在同一服务器上运行的 **clinfo** 守护程序，它将读入基于本地服务器的 **/usr/es/sbin/cluster/etc/clhosts** 文件，该文件仅包含与回送地址相关联的 IP 地址。
- 对于在客户机节点（即，未在运行 **clstrmgr** 守护程序的节点）上运行的 **clinfo** 守护程序，为了实现最高可用性，基于客户机的 **/usr/es/sbin/cluster/etc/clhosts** 文件应该包含所有 PowerHA SystemMirror 服务器节点的 IP 地址。通过这种方式，如果 PowerHA SystemMirror 服务器节点不可用（例如，已关闭电源），那么客户机节点上的 **clinfo** 守护程序可以尝试通过 SNMP 连接至另一个 PowerHA SystemMirror 服务器节点。

如果 Clinfo 在启动时与本地 **SNMP** 进程通信未成功，那么它未获取集群映射，因此无法尝试连接至另一个 **SNMP** 进程。

相关信息:

概念和设施指南

SNMP 共用名和 Clinfo

/etc/snmpd.conf 文件的版本取决于您正在使用的 AIX 的版本。对于 AIX，PowerHA SystemMirror 中使用的缺省版本是 **snmpdv3.conf** 文件。

PowerHA SystemMirror 使用的简单网络管理协议 (SNMP) 共用名取决于系统上运行的 SNMP 版本。SNMP 共用名按如下方法确定:

- 如果系统正在运行 SNMP V1，那么共用名是 **lssrc -ls snmpd** 命令输出中找到的第一个非 `private` 或 `system` 的名称。
- 如果系统正在运行 SNMP V3，那么可在 **/etc/snmpdv3.conf** 文件的 `VACM_GROUP` 条目中找到共用名。

Clinfo 服务仍支持用于指定 SNMP 共用名的 **-c** 选项，但并非要求使用该选项。使用 **-c** 选项将视为具有安全风险，这是因为运行 **ps** 命令会找到 SNMP 共用名。

注: 使 SNMP 共用名在 Clinfo 中受到保护非常重要，请将 **/tmp/hacmp.out**、**/etc/snmpd.conf**、**/smit.log** 和 **/usr/tmp/snmpd.log** 的许可权更改为并非任何用户均可读取（例如 600）。

相关信息:

snmpd.conf 文件

用于网络管理的 SNMP

声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您所在区域当前可获得的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务的操作，由用户自行负责。

IBM 可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并不意味着授予用户使用这些专利的任何许可。您可以用书面形式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

有关双字节字符集 (DBCS) 信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

以下段落对于英国和与当地法律有不同规定的其他国家或地区均不适用: INTERNATIONAL BUSINESS MACHINES CORPORATION“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。 因此本条款可能不适用于您。

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

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

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

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

IBM Corporation
Dept. LRAS/Bldg. 903

11501 Burnet Road
Austin, TX 78758-3400
USA

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

本文档中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的 资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问 题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

所有 IBM 的价格均是 IBM 当前的建议零售价，可随时更改而不另行通知。经销商的价格可与此不同。

本信息仅用于规划目的。在所描述的产品上市之前，此处的信息会有更改。

本信息包括日常业务运作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称均是虚构的，如与实际的商业企业使用的名称和地址有任何相似之处，纯属巧合。

版权许可:

本信息包含源语言形式的样本应用程序，用以阐明在不同操作平台上的编程技术。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例尚未在所有条件下经过全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。这些实例程序“按现状”提供，不附有任何种类的保证。对于因使用样本程序所引起的任何损害，IBM 概不负责。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

此部分代码是根据 IBM Corp. 公司的样本程序衍生出来的。

隐私策略注意事项

IBM 软件产品 （“软件产品”，其中包括作为服务解决方案的软件）可能使用 cookie 或其他技术来收集产品使用信息，以帮助改进最终用户体验、定制与最终用户的交互或实现其他目的。在许多情况下，软件产品不会收集任何个人可标识信息。我们的某些软件产品可以帮助您收集个人可标识信息。如果此软件产品使用 cookie 来收集个人可标识信息，那么会在下面列出有关此产品使用 cookie 的特定信息

此软件产品不会使用 cookie 或其他技术来收集个人可标识信息。

如果为此软件产品部署的配置使您能够作为客户通过 cookie 和其他技术从最终用户收集个人可标识信息，那么您应该向您自己的法律顾问咨询有关适用于这种数据收集（其中包括对于通知和同意的任何需求）的任何法律。

有关为这些目的使用各种技术（其中包括 cookie）的更多信息，请参阅 “IBM 隐私策略”（网址为 http://www.ibm.com/privacy）和“IBM 在线隐私声明”（网址为 http://www.ibm.com/privacy/details）中标题为“cookie、Web 信标和其他技术”和“软件产品和 Software-as-a 服务”（网址为 http://www.ibm.com/software/info/product-privacy）的部分。

商标

IBM、IBM 徽标和 ibm.com 是 International Business Machines Corp. 在全世界许多管辖区域注册的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。当前最新的 IBM 商标列表在以下 Web 站点提供版权和商标信息 (www.ibm.com/legal/copytrade.shtml)。

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

索引

L

S

Printed in China