IBM solidDB
IBM solidDB Universal Cache

版本 6.5

共享内存访问和链接库访问用户指南

Information Management software

S151-1292-00

IBM solidDB
IBM solidDB Universal Cache

版本 6.5

共享内存访问和链接库访问用户指南

Information Management software

S151-1292-00

声明

在使用本资料及其支持的产品之前，请阅读第 87 页的『声明』中的信息。

此版本应用于IBM solidDB V6R5（产品编号 5724-V17）和IBM solidDB Universal Cache V6R5（产品编号 5724-W91）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

图

表

关于本手册

IBM® solidDB® 共享内存访问（SMA）和链接库访问（LLA）使应用程序可以直接链接至 solidDB 服务器，而无需通过网络协议（如 TCP/IP）进行通信。SMA 允许链接至多个应用程序，而 LLA 允许链接至一个应用程序。通过将网络连接替换为本地函数调用，性能将得到极大的提高。

本指南包含特定于 SMA 和 LLA 的信息。本指南补充了*《IBM solidDB 管理员指南》*中包含的信息，后者包含有关管理和维护 solidDB 的详细信息。

本指南假定您了解使用 C 编程语言所需的知识、一般的 DBMS 知识、熟悉 SQL 并拥有 solidDB 数据管理产品（如 solidDB 内存数据库或 solidDB 基于磁盘的引擎）的知识。如果要将 SMA 或 LLA 与 Java™ 结合使用，那么此手册也假定您拥有使用 Java 所需的知识。

印刷约定

solidDB 文档使用下列印刷约定:

表 1. 印刷约定

格式	适用于
数据库表	此字体用于所有普通文本。
NOT NULL	采用此字体的大写字母指示 SQL 关键字和宏名称。
`solid.ini`	这些字体指示文件名和路径表达式。
`SET SYNC MASTER YES;` `COMMIT WORK;`	此字体用于程序代码和程序输出。示例 SQL 语句也使用此字体。
run.sh	此字体用于样本命令行。
TRIG_COUNT()	此字体用于函数名。
java.sql.Connection	此字体用于接口名称。
LockHashSize	此字体用于参数名、函数自变量和 Windows® 注册表条目。
argument	此类强调词指示用户或应用程序必须提供的信息。
管理员指南	这种样式用于引用其他文档或者同一文档中的章节。新术语和强调的问题也按此样式书写。
文件路径表示	除非另有声明，否则文件路径按 UNIX® 格式表示。斜杠（/）字符表示安装根目录。

表 1. 印刷约定 (续)

格式	适用于
操作系统	如果文档包含有关操作系统之间的差别的内容，那么首先提到的是 UNIX 格式。Microsoft® Windows 格式位于 UNIX 格式之后并括在括号中。其他操作系统将单独列出。对于不同的操作系统还可能有不同的章节进行描述。

语法表示法约定

solidDB 文档使用下列语法表示法约定:

表 2. 语法表示法约定

格式	适用于
`INSERT INTO` *`table_name`*	语法描述采用此字体。可替换部分采用*此*字体。
`solid.ini`	此字体指示文件名和路径表达式。
[ ]	方括号指示可选项；如果是粗体文本，那么必须将方括号包含在语法中。
|	竖线，用于将语法行中的两个互斥选项分隔开。
{ }	大括号用于对语法行中的一组互斥选项进行定界；如果是粗体文本，那么必须将大括号包括在语法中。
...	省略号指示可以多次重复使用自变量。
. . .	由三个点组成的一列表示这是先前代码行的延续。

1 共享内存访问和链接库访问的概述

solidDB *共享访问库*（SMA）和*链接库访问*（LLA）允许应用程序直接链接至 solidDB 服务器，而无需通过影响性能的网络协议（如 TCP/IP）进行通信。SMA 允许链接至多个应用程序，而 LLA 允许链接至一个应用程序。

SMA 和 LLA 都作为库文件进行实施，它们包含库格式的 solidDB 服务器的完整副本。

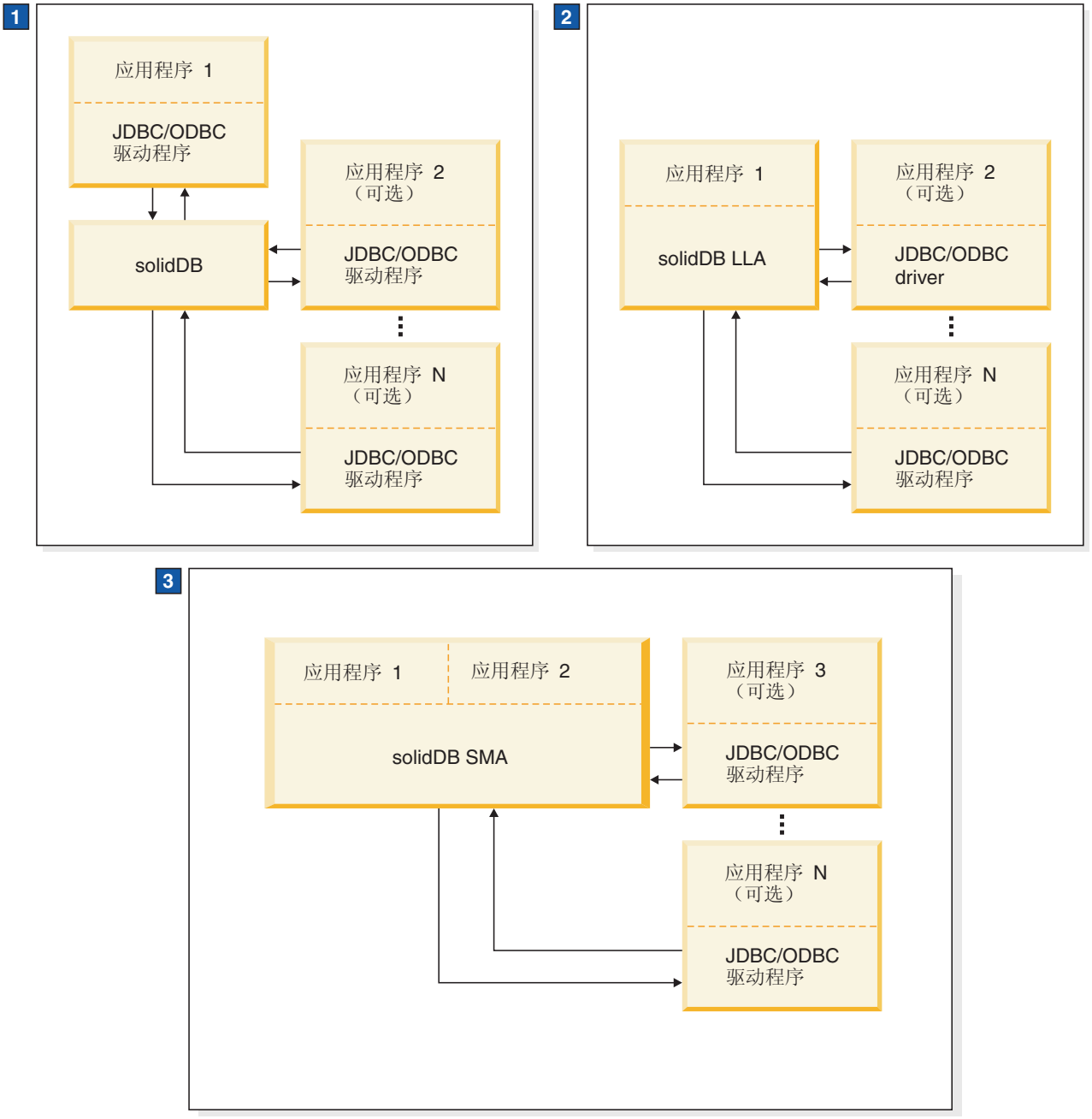
- 如果是 SMA，应用程序链接到的库可视为驱动程序。启动链接的应用程序之前，solidDB 服务器以 SMA 方式启动，它会动态装入 SMA 驱动程序并分配和初始化应用程序用于访问数据库的共享内存段。
- 如果是 LLA，应用程序将链接至 LLA 库，并且应用程序和服务器将构建到单个可执行文件中。

应用程序不必经过重新编写以使用 SMA 或 LLA。应用程序使用 ODBC 或 JDBC 调用或 solidDB 专用 SA API 来与 solidDB 服务器通信。

SMA 和 LLA 服务器还可以处理通过通信协议（如 TCP/IP）连接到服务器的远程应用程序的请求。远程应用程序认为 SMA 或 LLA 服务器与任何其他 solidDB 服务器类似，但是本地 SMA 和 LLA 应用程序认为它们是速度更快并且可更加准确地进行控制的 solidDB 服务器。

同样，与基于网络的服务器类似，多个 SMA 和 LLA 服务器可以在同一个节点上运行。

与 SMA 和 LLA 配合使用的 solidDB 服务器可以是基于磁盘的服务器或无盘服务器。还支持内存表和基于磁盘的表。



1. 在标准 solidDB 数据库配置中，应用程序和服务器是独立的程序。
2. 在 LLA 配置中，LLA 是链接至应用程序的库。其他应用程序也可以与 LLA 服务器进行通信。
3. 在 SMA 配置中，SMA 是多个应用程序可链接至的驱动程序库。基于其他网络连接的应用程序也可以与 SMA 服务器进行通信。

图 1. 基于 solidDB 服务器使用 SMA、LLA 和网络连接的配置

共享内存访问（SMA）

使用共享内存访问（SMA），多个应用程序可链接至包含完整数据库服务器功能的动态驱动程序库。这意味着应用程序 ODBC 或 JDBC 请求几乎完全在应用程序进程空间中进行处理，而无需在进程之间进行上下文切换。为促进对公共数据库的处理，驱动程序访问服务器启动的共享内存段。

使用 SMA 驱动程序链接至 solidDB 服务器的应用程序称为 *SMA 应用程序*；服务器称为 *SMA 服务器*。

SMA 服务器

使用 SMA 启动第一个应用程序之前，通过启动动态装入 SMA 驱动程序库的小存根应用程序（`solidsma.exe` 或 `solidsma.bin`）来在 SMA 方式中启动 solidDB 服务器。SMA 服务器存根应用程序在内部启动服务器代码，它分配并初始化应用程序用于访问数据库的共享内存段。

SMA 服务器具有网络服务器的完整功能:

- SMA 服务器进程管理所有独立于客户机的任务: 启动和恢复、检查点和记录和进行备份等。
- 可以使用 solidDB 配置参数、admin 命令和命令行参数。
- 可相同地访问内存表和基于磁盘的表。
- SMA 服务器可用于 solidDB 通用高速缓存中，并且具有高可用性、高级复制和 CDC 复制配置。
-
- SMA 服务器也可用作基于常规网络连接的服务器。

服务器以 SMA 方式启动时，它会在常规侦听端口处接受来自 SMA 驱动程序的连接请求。通过将不同的端口号分配给不同的 SMA 服务器，可以在单个系统上同时运行多个 SMA 服务器。

关闭服务器或抛出所有用户时，应用程序在下次请求时会收到 `Connection lost` 错误。如果应用程序在强制关闭期间等待响应，那么它会收到关闭通知。

基于磁盘的服务器和无盘服务器

SMA 服务器可以是基于磁盘的服务器或无盘服务器。无盘服务器在没有硬盘的嵌入式系统（例如，网络路由器或交换机中的线卡）中很有用。

SMA 和 solidDB 工具

solidDB 数据管理工具可与 SMA 服务器基于网络的连接配合使用。

SMA 应用程序

现有的 ODBC 或 JDBC 应用程序不需要为了使用 SMA 而进行修改，数据源名称或连接字符串例外。例如，在 ODBC 应用程序中，可以使用常规 ODBC SQLConnect() 调用来请求连接。

现有的 LLA 应用程序可以更改为 SMA 应用程序，也可进行反向更改。应用程序也可从 SMA 应用程序更改为基于网络的应用程序。

SMA 驱动程序

SMA 驱动程序是包含采用库格式的 solidDB 服务器完整副本的动态库。应用程序可以直接或使用驱动程序管理器链接到 SMA 驱动程序。

驱动程序的二进制文件的占用量对应于完整的 solidDB服务器，其大小根据平台为 3-6 MB。但是，由于所有的应用程序都链接到同一个驱动程序，因此在添加其他应用程序时，内存占用量不会相乘。整个应用程序系统（应用程序、驱动程序和服务器）的总内存占用量相对于某个客户机/服务器模型。

SMA 连接

只要 SMA 服务器正在运行，应用程序就可以建立 SMA 或网络连接。对于 SMA 连接，应用程序必须位于与服务器相同的节点上。连接类型在连接字符串中进行定义。对于 ODBC 应用程序，如果使用了驱动程序管理器，那么配置 SMA 数据源的方式与配置 ODBC 驱动程序的方式相同。

连接请求使用任何本地可用的协议（tcpip、命名管道和 unix 管道）通过网络连接（握手连接）进行发送。在连接握手期间，共享内存段句柄将传递到驱动程序，以便它可以访问服务器的共享内存段。

SMA 的系统需求

SMA 在 64 位平台以及 32 位的 Linux® 上可用。对于使用 Java 的 SMA，需要 Java 运行时环境（JRE）1.4.2 或 Java Development Kit (JDK) 1.4.2 或更新版本。

SMA 支持的平台

- AIX®
- HP-UX
- Linux 64 位
- Linux 32 位
- Solaris
- Windows 64 位

有关受支持平台的详细信息，请参阅位于 http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html的 solidDB Web 站点上的 solidDB系统需求。

SMA 组件和包

SMA 驱动程序库和 SMA 服务器应用程序都包括在 solidDB 软件包中。为了将 SMA 与 Java 配合使用，需要 solidDB JDBC 驱动程序。

SMA 驱动程序（库）

用于最常用平台的 SMA 驱动程序库在下表中显示:

表 3. SMA 驱动程序（库）

平台	SMA 驱动程序库	缺省安装位置
Windows	ssolidsmaxx.dll **注**: 如果要直接链接到 SMA驱动程序，那么链接到 solidsma.lib 导入库文件，通过该文件可访问实际的 .dll 库文件。	库: <solidDB 安装目录>\bin 导入库: <solidDB 安装目录>\lib
Linux	ssolidsmaxx.so	<solidDB 安装目录>/bin
Solaris	ssolidsmaxx.so	<solidDB 安装目录>/bin
HP-UX	ssolidsmaxx.sl	<solidDB 安装目录>/bin
AIX	ssolidsmaxx.so	<solidDB 安装目录>/bin
xx 是驱动程序库的版本号，例如，ssolidsma65.so。		

所有平台的 SMA 驱动程序库包含下列各项:

- 完整的 solidDB 服务器功能
- 三个 API 的函数
  - solidDB ODBC 驱动程序函数，此函数允许与服务器库直接通信而不需要通过网络。
  - solidDB 控制 API（SSC API）库，此库包含用于控制 SMA 服务器的启动和关闭的函数。
  - solidDB SA API 库，此库对于其他使用链接库访问的功能可能是必需的。例如，此库允许从表中插入、删除和选择记录。

  由于应用程序链接到的库包含三个 API，所以应用程序可以调用这些 API 的任意组合中的函数。有关每个 API 的详细信息，请参阅第 8 页的『用于 SMA 和 LLA 的 solidDB API 和驱动程序』。

为了将 SMA 与 Java 配合使用，需要 solidDB JDBC 驱动程序；在 solidDB 服务器安装期间会将 solidDB JDBC 驱动程序（SolidDriver2.0.jar）安装到 solidDB 安装目录中的 jdbc 目录。

SMA 服务器应用程序

表 4. SMA 服务器应用程序

平台	SMA 应用程序
Windows	solidsma.exe
Linux	solidsma
Solaris	solidsma
HP-UX	solidsma
AIX	solidsma

链接库访问（LLA）

通过链接库访问（LLA），应用程序链接至一个包含完整数据库服务器功能的静态库或动态库。这就意味着 solidDB 与应用程序在同一个可执行文件中运行，因此不需要通过网络传输数据。

使用 LLA 库链接至 solidDB 服务器的应用程序称为 *LLA 应用程序*；服务器称为 *LLA 服务器*。

链接至 LLA 库的应用程序还可以使用 ODBC API、SA API 和 JDBC API 建立多个连接。所有这些 API 都是可重入的，允许同时建立来自不同线程的连接。

直接链接至 LLA 库的应用程序还可以创建与其他数据库服务器的远程连接。连接类型（本地或远程）在传递到 ODBC API 或 SA API 连接函数的连接字符串或 JDBC 连接属性中定义。

操作原则

当您启动应用程序时，只有此应用程序中的代码会自动开始运行。服务器代码在很大程度上与您的应用程序代码无关，您必须通过调用函数来显式启动服务器。在大多数实现中，运行服务器的线程与应用程序使用的线程不同。通过调用函数来启动服务器将执行服务器代码所需的任何初始化步骤，必要时创建相应的其他线程并启动在这些线程上运行的服务器。

基于磁盘的服务器和无盘服务器

与 LLA 配合使用的 solidDB 服务器可以是基于磁盘的服务器或无盘服务器。LLA 库包含两个不同的函数调用来启动服务器。SSCStartServer 函数调用启动常规的基于磁盘的服务器，而 SSCStartDisklessServer 启动不使用磁盘驱动器的服务器。

LLA 的系统需求

LLA 可用于 solidDB 支持的所有平台。对于使用 Java 的 LLA，需要 Java 运行时环境（JRE）1.4.2 或 Java Development Kit (JDK) 1.4.2 或更新版本。

有关受支持平台的列表，请参阅位于 http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html 的 solidDB Web 站点上的 solidDB 系统需求。

LLA 组件和包装

LLA 库包含在 solidDB 软件包中。对于使用 Java 的 LLA，JDBC 驱动程序和 solidDB 专用控制类嵌入在 solidDB JDBC 驱动程序中。

最常用平台的 LLA 库在下表中显示:

表 5. LLA 库

平台	静态 LLA 库	动态 LLA 库	缺省位置
Windows	solidimpac.lib 这是让您可以访问实际库文件 ssolidacxx.dll 的导入库文件。	ssolidacxx.dll	导入库: <solidDB 安装目录>\lib 库: <solidDB 安装目录>\bin
Linux	solidac.a	ssolidacxx.so	<solidDB 安装目录>/bin
Solaris	solidac.a	ssolidacxx.so	<solidDB 安装目录>/bin
HP-UX	solidac.a	ssolidacxx.sl	<solidDB 安装目录>/bin
xx 是动态库的版本号，例如，ssolidac65.so。			

所有平台的 LLA 库包含下列各项:

- 完整的 solidDB 服务器功能
- 三个单独 API 的功能
  - solidDB 控制 API (SSC API) 库，此库包含用于控制任务调度的函数。
  - solidDB ODBC Driver 函数，此函数允许与服务器库直接通信而不需要通过网络。
  - solidDB SA API 库，此库对于其他使用链接库访问的功能可能是必需的。例如，此库允许从表中插入、删除和选择记录。

  由于应用程序链接到的库包含这三个 API（SSC、SA 和 ODBC），所以应用程序可以调用这些 API 的任意组合中的函数。有关每个 API 的详细信息，请参阅第 8 页的『用于 SMA 和 LLA 的 solidDB API 和驱动程序』。

  注: 远程应用程序也可以访问这三个 API（SSC、SA 和 ODBC）。但是，对于远程应用程序来说，这三个 API 的函数并不是全部在同一个文件中。有关远程应用程序和具有双重角色的应用程序的详细信息，请阅读第 11 页的『使用本地和远程应用程序类型的配置』。有关用于远程应用程序的 API 文件的信息，请阅读第 8 页的『用于 SMA 和 LLA 的 solidDB API 和驱动程序』。

对于与 Java 配合使用的 LLA，需要 solidDB JDBC 驱动程序; solidDB JDBC 驱动程序 jar 文件（SolidDriver2.0.jar）包含以下软件包:

- solid.jdbc.* solidDB JDBC 驱动程序类
- solid.ssc.* solidDB 服务器控制类（Java 接口的 SSC API）

LLA 的静态与动态链接库

solidDB 同时提供了链接库访问库的*静态* 和*动态* 版本。

静态和动态库文件中都包含采用库格式的 solidDB 服务器的完整副本。当您使用静态库文件（例如，lib/solidac.a）时，将程序直接链接至此文件，代码和库代码都将写入生成的可执行文件中。如果您链接至动态库文件，那么此库中的代码不会包含在输出文件中，但是输出文件中包含可执行程序。当运行您的程序时，将单独从此动态链接库中装入代码。

除了更改可执行文件的大小之外，链接至静态库文件与链接至动态库文件具有很小的区别。如果您在计算机上执行单个客户机和单个服务器，那么内存中的代码总量在任何时候都相似。它们的性能也差不多，只不过当您使用动态库时稍微会增加一点开销。

使用动态链接库文件的主要优势是在同一台计算机上执行服务器的多个副本时可以节省内存。例如，如果您正在单台计算机上执行开发工作，并且您希望此计算机上同时具有高级复制主控服务器和副本服务器，或者您希望同时具有 HotStandby 主服务器和 HotStandby 辅助服务器，那么您可能更愿意使用动态库，以便不会同时在内存中拥有 LLA 的多个副本。

在 Microsoft Windows 上，solidDB 链接库访问包含其他文件 `lib/solidimpac.lib`。在 Microsoft Windows 上，如果您想使用动态链接库，请不要直接链接至动态链接库 `ssolidacxx.dll` 本身，而是链接至 `solidimpac.lib`（这是一个导入库）。这只会将少量代码链接至客户机可执行文件。当实际执行客户机程序时，Microsoft Windows 操作系统将自动装入 `ssolidacxx.dll` 文件，并且客户机可以调用此 `.dll` 文件中的常用链接库访问函数。运行引用了 itl 的程序时，`.dll` 文件必须位于装入路径中。

注: 使用动态链接库文件并不意味着您可以将多个 LLA 应用程序客户机链接至单个 solidDB 服务器。即使使用动态库方法，您仍然只能具有单个本地客户机；所有其他客户机必须是远程客户机，这意味着它们将通过使用 TCP 或者其他网络协议与 solidDB 服务器进行通信，而不是通过可供本地客户机使用的直接函数调用来与服务器进行通信。要使多个本地应用程序可访问 solidDB，请使用共享内存访问（SMA）来设计您的环境。

用于 SMA 和 LLA 的 solidDB API 和驱动程序

SMA 和 LLA 应用程序请求通常通过 ODBC API 直接函数调用或 JDBC 调用进行处理。solidDB 专用的 solidDB API 也可用。solidDB 服务器控制 API（SSC API 和用于 Java 的 SSC API）包括在 LLA 库中，以处理用于控制 solidDB 后台进程和客户机任务的本地请求。SMA 包括对 SSC API 的有限支持；仅包括用于启动或停止 SMA 服务器的调用。

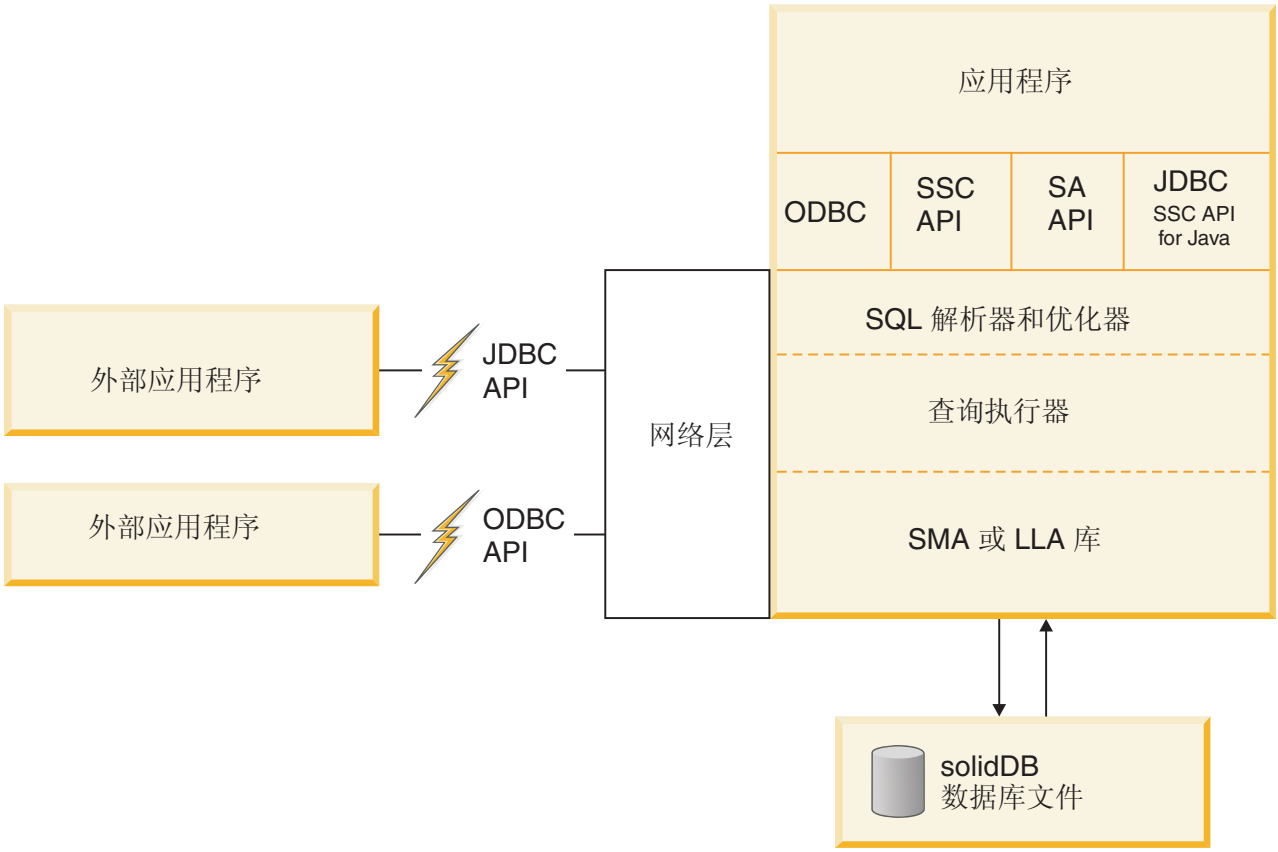


图 2. 示例: 用于 C/C++ 程序的 SMA 和 LLA API

solidDB SA API

solidDB SA API 是用于 solidDB 数据管理服务的低级专用 C 语言 API。SA API 使用 SA API 函数调用对本地应用程序提供支持。

SA API 库用于 solidDB 产品内部，通过此库可以访问 solidDB 数据库表中的数据。此库包含大约 90 个函数，它们提供了用于连接数据库和运行基于游标的操作的低级别机制。solidDB SA API 可以显著提高性能。例如，可以使用 SA API 来优化批处理插入操作的性能。

有关 solidDB SA API 的详细信息，请参阅 *IBM solidDB Programmer Guide*。

远程应用程序也可以使用 SA API 函数调用。但是，您必须链接至单独的 SA API 库文件（例如，在 Windows 平台上为 `solidimpsa.lib`）。

solidDB ODBC API

solidDB ODBC API 提供了一种符合标准的方法并通过 SQL 来访问本地或远程 solidDB 数据库中的数据。它提供了一些函数来控制数据库连接、执行 SQL 语句、检索结果集、落实事务以及其他数据管理功能。

solidDB ODBC API 是 solidDB 数据库的调用级接口（CLI）。它与 ANSI X3H2 SQL CLI 兼容。

SMA 和 LLA 支持 ODBC 3.51 标准。 SMA 和 LLA 库包含 solidDB ODBC 3.x，它支持需要对服务器进行直接函数调用的本地应用程序。

有关 solidDB ODBC API 的详细信息，请参阅 *IBM solidDB Programmer Guide*。

solidDB JDBC API

JDBC API 定义 Java 类来表示数据库连接、SQL 语句、结果集和数据库元数据等。它允许您发出 SQL 语句并处理结果。JDBC 是用于 Java 数据库访问的主要 API。

SMA 和 LLA 支持 JDBC 1.x 和 2.x。

JDBC 接口和 solidDB JDBC 驱动程序（包括特定于 solidDB 增强的描述）都记录在 *IBM solidDB Programmer Guide* 中。

solidDB 服务器控制 API（SSC API）

solidDB 服务器控制 API（SSC API）是一个用 C 语言编写的线程安全接口，用来控制 solidDB 服务器的行为。

SSC API 函数包括在 SMA 驱动程序和 LLA 库文件中。但是，对于 SMA，仅支持用于启动和停止服务器的函数。

LLA 通过使用 SSC API 函数调用来支持本地应用程序，还提供了一个仅用于远程应用程序的独立库。

如果 LLA 应用程序远程运行并且包含 SSC API 函数调用，那么您必须链接至 SSC API 存根库（例如，对于 Windows 是 `solidctrlstub.lib`）。实际上，此库不会向您提供服务器的远程应用程序控制；它仅允许您将应用程序作为远程应用程序进行编译和连接，且不会从具有 LLA 的 solidDB 收到链接时错误。

用于 Java 的 solidDB服务器控制 API（SSC API）

用于 Java 的 solidDB 服务器控制 API（SSC API）是专用 API，根据 SolidServerControl 类进行命名。用于 Java 调用的 SSC API 用于启动和停止 LLA 服务器。实际的数据库连接使用常规 solidDB JDBC API 来完成。用于 Java 类的 SSC API 和 solidDB JDBC 驱动程序类包括在 solidDB JDBC 驱动程序中（`SolidDriver2.0.jar`）。

用于访问 solidDB 服务器的 SolidServerControl 类已嵌入在 `solid.ssc` 包中的 solidDB JDBC 驱动程序文件中。solidDB JDBC 驱动程序 JAR 文件（`SolidDriver2.0.jar`）中包含下列程序包:

- `solid.jdbc.*` solidDB JDBC 驱动程序类
- `solid.ssc.*` solidDB 服务器控制类（专用 API 接口）

solidDB 服务器控制（`solid.ssc`）程序包中的类为:

- SolidServerControl（用于从 Java 启动并停止 LLA 服务器）
- SolidServerControlInitializationError（用于报告错误）

有关更多详细信息，请参阅 第 79 页的附录 E, 『SolidServerControl 类接口』。

使用本地和远程应用程序类型的配置

使用 SMA 和 LLA 时，应用程序总是会连接到本地 solidDB 服务器（SMA 服务器或 LLA 服务器）；应用程序和 solidDB 服务器位于同一个节点上。除了处理来自此本地 SMA 或 LLA 应用程序的请求，SMA 或 LLA 服务器还可以处理通过通信协议（如 TCP/IP）连接到服务器的远程应用程序的请求。还可以编写双方式应用程序；该应用程序根据其编译和链接的方式，在本地和远程方式之间进行切换。

SMA 或 LLA 应用程序是*本地应用程序*；服务器和应用程序位于同一个节点上。例如，对 ODBC 函数的调用将直接访问服务器，而不是通过 ODBC 驱动程序和通信协议（如 TCP/IP）来访问服务器。

远程应用程序 将不会链接到 SMA 驱动程序或 LLA 库。它是一个独立的可执行文件，必须使用网络连接（如 TCP/IP）或其他连接来与服务器通信。尽管运行远程应用程序的节点通常与运行服务器的节点不同，但是，如果应用程序使用网络通信协议来与服务器通信，那么也会认为此应用程序是远程应用程序。单个节点可以运行本地 SMA 或 LLA 应用程序，同时将一个或多个远程应用程序作为独立进程来运行。

远程应用程序认为 SMA 和 LLA 服务器与任何其他 solidDB 服务器类似，但是本地应用程序认为它们是速度更快并且可更加准确地进行控制的 solidDB 服务器。

大多数应用程序是本地（链接到 SMA 驱动程序或 LLA 库）或远程（从不链接）的。但是，也可以编写使用本地连接和基于网络的连接的*双方式应用程序*。例如，应用程序可以在本地或远程方式中使用相同的 C 语言应用程序代码，但是它在本地方式和远程方式中链接到的库不同。

例如，在以下情况中，双方式应用程序可能会比较有用:

- 将本地应用程序与 SMA 或 LLA 库相链接之前，您要先测试该本地应用程序。
- 您希望本地或远程的所有用户/进程拥有相同的应用程序逻辑。

远程应用程序可以是 C 程序和 Java 程序的混合体。编写本地客户机时使用的语言并不会限制可以用来编写远程客户机的语言。例如，如果将 LLA 与 Java 配合使用，那么远程客户机程序可以使用 C 和/或 Java。

用于远程应用程序的 SSC API 和 SA API 库

包含 SSC API 或 SA API 函数调用的远程应用程序必须链接到不同的库。

表 6. 用于远程应用程序的 SSC API 和 SA API 库

平台	SSC API 存根库	SA API 库	缺省位置
Windows	solidctrlstub.lib	solidimpsa.lib	<solidDB 安装目录>\lib
其他平台	solidctrlstub.a	solidimpsa.a	<solidDB 安装目录>/bin

远程应用程序需要 SSC API 存根库，因为在 SMA 和 LLA 库中包含的 SSC API 函数不能用于远程应用程序。例如，假定您有已链接至标准 ODBC 库的本地应用程序（包含 SSC API 函数）。您希望远程运行相同的应用程序。通过链接到 SSC API 存根库，就可以避免必须从代码中除去 SSC API 函数调用。这样就很容易将 LLA 或 SMA 应用程序变为一个正常的远程客户机应用程序。

注: SSC API 存根库包含“不执行任何操作”函数；如果您在远程应用程序中调用这些函数，那么它们对服务器没有影响。这意味着实际上，SSC API 存根库不会向您提供服务器的远程应用程序控制权；它仅仅是允许您将应用程序作为远程应用程序进行编译和链接，且不会从具有 LLA 或 SMA 的 solidDB 收到链接时错误。

2 创建并运行 SMA 应用程序

要创建 SMA 应用程序，请根据需要配置 solidDB、将应用程序链接至 SMA 驱动程序、启动 SMA 服务器并在应用程序和服务器之间建立本地连接。在创建应用程序之后，您可以使用 solidDB 提供的监视功能部件来监视 SMA 性能。

要点：创建并运行 SMA 应用程序的指示信息提供了与没有 SMA 的 solidDB 相比较时，特定于 SMA 的增强、补充和使用差异。

有关 solidDB SQL、solidDB 数据管理工具、一般 solidDB 管理和维护以及数据库错误代码的信息，请参阅《*IBM solidDB 管理员指南*》。

有关 API 和 solidDB JDBC 以及 ODBC 驱动程序的详细信息，请参阅 *IBM solidDB Programmer Guide*。

创建 SMA 应用程序 - 概述

要创建使用 SMA 的应用程序，您必须准备系统以供 SMA 使用、配置 solidDB、将应用程序设置为使用 SMA 驱动程序、启动 SMA 服务器并使应用程序与其相连接。

关于此任务

此过程提供了如何创建 SMA 应用程序的概述。用于 C/ODBC 环境的 SMA 应用程序的创建方式与不使用 SMA 的应用程序的创建方式相似。

注：开发应用程序时，建议您使用基于网络的连接。应用程序就绪后，请改为使用 SMA 连接。

有关用 C 语言编写的 SMA 应用程序的示例，请参阅 solidDB 安装目录中 `samples/sma` 目录下的 SMA 示例。

过程

1. **检查环境中共享内存使用的系统设置。**

   环境中共享内存使用的缺省值对于使用 SMA 可能不足。有关查看并设置系统上共享内存系统参数的详细信息，请参阅第 14 页的『修改共享内存内核参数 - 概述』。

2. **通过创建工作目录、solidDB 数据库和用户帐户来设置数据库环境。**

   有关指示信息，请参阅《*IBM solidDB 管理员指南*》中的『*创建新的数据库*』一节。

3. **配置 solidDB 以满足您的环境、性能和操作需求。**

   使用 `solid.ini` 配置文件来定义基本的配置设置，如数据库文件名和位置以及数据库块大小等。

   - 在常规设置中，不必修改 `solid.ini` 文件的 `[SharedMemoryAccess]` 部分中特定于 SMA 的参数；出厂值适用于大多数情况。
   - **Srv.ProcessMemoryLimit** 参数对于 SMA 服务器无效。

   如果没有配置文件，那么将使用缺省值。

4. **准备应用程序以供 SMA 使用。**

   可以设置应用程序来使用带或不带驱动程序管理器的 SMA。

   - 第 19 页的『准备应用程序以供带驱动程序管理器的 SMA 使用』
   - 第 20 页的『准备应用程序以供不带驱动程序管理器的 SMA 使用』

5. **启动 SMA 服务器。**

   有关指示信息，请参阅第 22 页的『启动 SMA 服务器』。

6. **启动应用程序。**

修改共享内存内核参数 - 概述

共享内存以段为单位进行分配。共享内存系统参数控制系统上允许的段的最大大小和数量。

通常 solidDB 使用 8 MB 大小。

共享内存参数及其管理机制取决于系统。在 Linux 和 UNIX 环境中，您可能需要处理下面描述的内核参数的类型。

要点: 本节和下面的各节仅讨论 solidDB 设置的需求。在同一系统上运行的其他进程可能需要更高的限制值。

- 共享内存段的最大大小

  通常，您不需要修改缺省系统设置。这是因为 solidDB 段大小为 8 MB 是比较小的。

- 系统/进程中共享内存段的最大数
  - 由于 solidDB 分配的段大部分都为 8 MB，您需要的段可能比系统缺省情况下允许的段要多，特别是使用大型数据库时。

    共享内存段的最大数应该至少为可以由 8 除的 solidDB 进程的大小（以 MB 为单位）。

    例如，对于 1 GB（1024 MB）的进程大小，需要至少 128 个段。
  - 您应总是明确地将最大段数设置为比数据库大小所需的值更高的值；较高的值没有副作用。
  - solidDB 仅使用一个进程；如果您的环境要求对进程和系统分别设置最大段数，那么您可以对这两者使用相同的值。
- 所有共享内存段的最大总大小

  所有共享内存段的组合总大小取决于数据库的大小和磁盘空间的可用性。建议您不要将此参数设置为比机器上可用物理内存更高的值。

AIX 上 SMA 的共享内存内核参数

在 AIX 系统上，不需要修改共享内存内核参数。已为 AIX IPC 机制定义了上限，因此不可对它们进行配置。共享内存限制根据需要进行动态分配和取消分配，因此内存需求总是取决于当前的系统使用。

有关一般 AIX 共享内存缺省限制和 IPC 机制的详细信息，请参阅 IBM Systems 信息中心（http://publib.boulder.ibm.com/infocenter/systems/index.jsp）内的进程间通信（IPC）限制一节。

修改 HP-UX 上 SMA 的共享内存内核参数

HP-UX 上共享内存内核参数的缺省值可能不足以运行 SMA 应用程序。可以使用 kctune 命令来动态更改内核参数值。

开始之前

您必须具有 root 用户的权限才能修改共享内存内核参数。

关于此任务

以下步骤显示如何在 HP-UX 上设置共享内存内核参数。根据需求显示的最小值由 solidDB 进行设置。在同一系统上运行的其他进程可能需要更高的限制值。

在 HP-UX 环境中，您可能需要修改以下共享内存内核参数:

- **shmmni** – 系统上共享内存段的最大数
- **shmseg** – 与进程相连接的共享内存段的最大数
- **shmmax** – 系统上所有共享内存段的最大大小（以字节为单位）

过程

1. **查看共享内存内核参数以确定系统是否需要进行任何必要的更改。**

   查看 **shmmni** 参数:

```
kctune -v shmmni
Tunable             shmmni
Description         Maximum number of shared memory segments on the system
Module              vm_asi
Current Value       400 [Default]
Value at Next Boot  400 [Default]
Value at Last Boot  400
Default Value       400
Constraints         shmmni >= 3
                    shmmni <= 32768
                    shmmni >= shmseg
Can Change          Immediately or at Next Boot
```

   查看 **shmseg** 参数:

```
kctune -v shmseg
Tunable             shmseg
Description         Maximum number of shared memory segments attached to a process
Module              vm_asi
Current Value       300 [Default]
Value at Next Boot  300 [Default]
Value at Last Boot  300
Default Value       300
Constraints         shmseg >= 1
                    shmseg <= shmmni
Can Change          Immediately or at Next Boot
```

   查看 **shmmax** 参数:

```
kctune -v shmmax
Tunable             shmmax
Description         Maximum size of a shared memory segment (bytes)
Module              vm_asi
Current Value       1073741824 [Default]
Value at Next Boot  1073741824 [Default]
```

```
Value at Last Boot  1073741824
Default Value       1073741824
Constraints         shmmax >= 2048
                    shmmax <= 4398046511104
Can Change          Immediately or at Next Boot
```

表 7. SMA（HP-UX）的共享内存内核参数的最低需求

参数	描述	修改时间	注释
shmmni	系统上共享内存段的最大数	该值小于由 8 除的 solidDB 进程大小（以 MB 为单位）时 例如，对于 1 GB（1024 MB）的进程大小，需要至少 128 个段。	您应总是明确将此参数设置为比数据库大小所需的值更高的值；较高的值没有副作用。
shmseg	与进程相连接的共享内存段的最大数	该值小于由 8 除的 solidDB 进程大小（以 MB 为单位）时 例如，对于 1 GB（1024 MB）的进程大小，需要至少 128 个段。	由于 solidDB 仅使用一个进程，所以 **shmmni** 和 **shmseg** 的值可以相同。
shmmax	共享内存段的最大大小（以字节为单位）	值小于 32768 KB（32 MB）时	将此参数设置为没有副作用的较高值。

2. **要修改这些参数，请使用 kctune 命令。**

   例如，要将共享内存段的最大数设置为 1024，请使用以下命令:

   ```
   kctune shmmni=1024
   ```

   更改的参数值将立即生效，并且在重新引导后仍然有效。

下一步做什么

如果您在收到 `out of memory` 错误后修改了共享内存参数，那么可能需要使用 `ipcrm` 命令来清除正在暂挂的共享内存段。有关更多详细信息，请参阅 第 23 页的『对 SMA 进行故障诊断』。

修改 Linux 上 SMA 的共享内存内核参数

Linux 上共享内存内核参数的缺省值可能不足以运行 SMA 应用程序。要修改 Linux 上的共享内存内核参数，请编辑 `/etc/sysctl.conf` 文件。

开始之前

您必须具有 root 用户的权限才能修改内核参数。

关于此任务

以下步骤显示如何使用 solidDB 设置的共享内存需求来更新 Red Hat 和 SUSE Linux 上的内核参数。在同一系统上运行的其他进程可能需要更高的限制值。

在 Linux 环境中，您可能需要修改以下共享内存参数:

- **SHMMNI** – 系统上共享内存段的最大数
- **SHMMAX** – 系统上单个共享内存段的最大大小
- **SHMALL** – 系统上分配的最大共享内存页数

过程

1. **运行 ipcs -l 命令。**

   例如:

   注: 在 // 后添加了注释以显示参数名称。

   ```
   # ipcs -l

   ------ Shared Memory Limits --------
   max number of segments = 4096                // SHMMNI
   max seg size (kbytes) = 32768                // SHMMAX
   max total shared memory (kbytes) = 8388608  // SHMALL
   ```

2. **分析输出以确定系统是否需要进行任何必要的更改。**

表 8. SMA（Linux）的共享内存内核参数的最低需求

内核参数	描述	修改时间	注释
SHMMNI	系统上共享内存段的最大数	该值小于由 8 除的 solidDB 进程大小（以 MB 为单位）时 例如，对于 1 GB（1024 MB）的进程大小，需要至少 128 个段。	您应总是明确将此参数设置为比数据库大小所需的值更高的值；较高的值没有副作用。
SHMMAX	系统上单个共享内存段的最大大小	值小于 32768 KB（32 MB）时	将此参数设置为没有副作用的较高值。 **注:** ipcs 输出已转换为 SHMMAX 千字节。内核需要以字节表示的 SHMMAX 值。
SHMALL	系统上分配的最大共享内存页数	值小于由 4 除的进程的最大大小时（以 KB 为单位）	此参数可增加到计算机物理内存的大约 90%。 例如，如果您有一个 16 GB 内存的计算机系统主要用于 SMA，那么 SHMALL 应设置为 3774873（16 GB 的 90% 是 14.4 GB；然后用 4 KB 基本页面大小来除 14.4 GB）。 **注:** ipcs 输出已将 SHMALL 转换为千字节。内核要求 SHMALL 值为页数。

3. **要修改这些内核参数，请编辑 /etc/sysctl.conf 文件。**

   如果该文件不存在，请创建该文件。

   以下行是应该放入文件中的示例:

   ```
   #Example shmmni for a 1 GB database
   kernel.shmmni=400
   #Example shmmax for a 64-bit system
   kernel.shmmax=1073741824
   #Example shmall for 90 percent of 16 GB memory
   kernel.shmall=3774873
   ```

4. **运行带 -p 参数的 sysctl 以便从缺省文件 /etc/sysctl.conf 中装入 sysctl 设置。**

   ```
   sysctl -p
   ```

5. **每次更改后都进行重新引导以使更改生效。**
   - 在 SUSE Linux 中: 激活 boot.sysctl。
   - 在 Red Hat Linux 中: rc.sysinit 初始化脚本将自动读取 /etc/sysctl.conf 文件。

下一步做什么

如果您在收到 `out of memory` 错误后修改了共享内存参数，那么可能需要使用 `ipcrm` 命令来清除正在暂挂的共享内存段。有关更多详细信息，请参阅 第 23 页的『对 SMA 进行故障诊断』。

修改 Solaris 上 SMA 的共享内存内核参数

Solaris 10 上共享内存内核参数的缺省值可能不足以运行 SMA 应用程序。在 Solaris 10 上，共享内存内核参数值可随 Solaris 资源控制设施动态更改。

开始之前

您必须具有 root 用户的权限才能修改共享内存参数。

关于此任务

以下步骤显示如何在 Solaris 10 上设置共享内存内核参数。根据需求显示的最小值由 solidDB 进行设置。在同一系统上运行的其他进程可能需要更高的限制值。

在 Solaris 环境中，您可能需要修改以下共享内存内核参数:

- **max-shm-ids** – 系统上共享内存段的最大数
- **max-shm-memory** – 系统上所有共享内存段的最大大小（以 MB 为单位）

在以下示例中，将使用操作系统的缺省项目。

过程

1. **查看共享内存参数以确定系统是否需要进行任何必要的更改。**

   查看 **project.max-shm-ids** 参数:

```
prctl -n project.max-shm-ids -i project default
project: 3: default
NAME    PRIVILEGE       VALUE    FLAG   ACTION          RECIPIENT
project.max-shm-ids
        privileged        128       -   deny                    -
        system          16.8M     max   deny                    -
```

   查看 **project.max-shm-memory** 参数:

```
prctl -n project.max-shm-memory -i project default
project: 3: default
NAME    PRIVILEGE       VALUE    FLAG   ACTION          RECIPIENT
project.max-shm-memory
        privileged      62.7GB       -   deny                    -
        system          16.0EB     max   deny
```

表 9. SMA（Solaris）的共享内存内核参数的最低需求

参数	描述	修改时间	注释
max-shm-ids	系统上共享内存段的最大数	该值小于由 8 除的 solidDB 进程大小（以 MB 为单位）时 例如，对于 1 GB（1024 MB）的进程大小，需要至少 128 个段。	您应总是明确将此参数设置为比数据库大小所需的值更高的值；较高的值没有副作用。
max-shm-memory	系统上所有共享内存段的最大大小	值小于进程大小的最大大小时	将此参数设置为没有副作用的较高值。

2. **要修改这些参数，请使用 prctl 命令。**

   例如，要将共享内存段的最大数设置为 1024，请使用以下命令:

   ```
   prctl -n project.max-shm-ids -r -v 1024 -i project default
   ```

3. **每次更改后都进行重新引导以使更改生效。**

   ```
   /usr/sbin/projmod -sK "project.max-shm-ids=(privileged,1024,deny)" default
   ```

下一步做什么

如果您在收到 out of memory 错误后修改了共享内存参数，那么可能需要使用 ipcrm 命令来清除正在暂挂的共享内存段。有关更多详细信息，请参阅 第 23 页的『对 SMA 进行故障诊断』。

准备应用程序以供带驱动程序管理器的 SMA 使用

使用带驱动程序管理器的 SMA 时，将连接到 SMA 数据源，其方式与连接到 solidDB ODBC 数据源相似。

关于此任务

在安装 solidDB 时安装 SMA 驱动程序库文件。下表列出了最常用平台的文件名及其缺省安装位置。

表 10. SMA 驱动程序（库）

平台	SMA 驱动程序库	缺省安装位置
Windows	ssolidsmaxx.dll **注**: 如果要直接链接到 SMA驱动程序，那么链接到 solidsma.lib 导入库文件，通过该文件可访问实际的 .dll 库文件。	库: <solidDB 安装目录>\bin 导入库: <solidDB 安装目录>\lib
Linux	ssolidsmaxx.so	<solidDB 安装目录>/bin
Solaris	ssolidsmaxx.so	<solidDB 安装目录>/bin
HP-UX	ssolidsmaxx.sl	<solidDB 安装目录>/bin
AIX	ssolidsmaxx.so	<solidDB 安装目录>/bin
xx 是驱动程序库的版本号，例如，ssolidsma65.so。		

过程

1. **连接到 SMA 数据源。**

   定义数据源连接信息时，请使用特定于 SMA 的连接字符串。

   SMA 连接的连接字符串语法为:

   ```
   sma <协议名称> <端口号或管道名称>
   ```

   例如:

   ```
   sma tcp 2315
   ```

2. **禁用信号处理程序。**

信号处理程序用来向应用程序报告发生了意外事件。不要在应用程序中设置信号处理程序，因为它们会覆盖 SMA 设置的信号处理程序。

例如，如果应用程序为浮点异常设置了信号处理程序，那么此设置将覆盖 SMA 设置的处理程序。因此服务器无法捕获诸如“除数为 0”等异常情况。

准备应用程序以供不带驱动程序管理器的 SMA 使用

使用不带驱动程序管理器的 SMA 时，将应用程序直接链接到 SMA 驱动程序库，其方式与直接链接到 solidDB ODBC 驱动程序库类似。

过程

1. **将应用程序链接到 SMA 驱动程序库。**

   在安装 solidDB 时安装 SMA 驱动程序库文件。下表列出了最常用平台的文件名及其缺省安装位置。

   表 11. SMA 驱动程序（库）

平台	SMA 驱动程序库	缺省安装位置
Windows	ssolidsmaxx.dll **注**: 如果要直接链接到 SMA驱动程序，那么链接到 solidsma.lib 导入库文件，通过该文件可访问实际的 .dll 库文件。	库: <solidDB 安装目录>\bin 导入库: <solidDB 安装目录>\lib
Linux	ssolidsmaxx.so	<solidDB 安装目录>/bin
Solaris	ssolidsmaxx.so	<solidDB 安装目录>/bin
HP-UX	ssolidsmaxx.sl	<solidDB 安装目录>/bin
AIX	ssolidsmaxx.so	<solidDB 安装目录>/bin
xx 是驱动程序库的版本号，例如，ssolidsma65.so。		

2. **将连接字符串更改为本地 SMA 服务器名称。**

   SMA 连接的连接字符串语法为:

   ```
   sma <协议名称> <端口号或管道名称>
   ```

   例如:

   ```
   sma tcp 2315
   ```

   有关使用 ODBC API 或 SA API 时连接字符串的示例，请参阅第 21 页的『建立 SMA 的本地连接』。

3. **禁用信号处理程序。**

   信号处理程序用来向应用程序报告发生了意外事件。不要在应用程序中设置信号处理程序，因为它们会覆盖 SMA 设置的信号处理程序。

   例如，如果应用程序为浮点异常设置了信号处理程序，那么此设置将覆盖 SMA 设置的处理程序。因此服务器无法捕获诸如“除数为 0”等异常情况。

建立 SMA 的本地连接

为了供 SMA 使用，应用程序需要建立与 SMA 服务器的本地 SMA 连接。连接类型使用特定于 SMA 的连接字符串进行定义。

对于 SMA，连接请求使用任何本地可用的协议（tcpip、命名管道和 unix 管道）通过网络连接（握手连接）进行发送。在连接握手期间，共享内存段句柄将传递到驱动程序，以便它可以访问服务器的共享内存。

SMA 连接的连接字符串语法为:

```
sma <协议名称> <端口号或管道名称>
```

例如:

```
sma tcp 2315
```

如果向 SMA 服务器以外的服务器发出了 SMA 连接请求，那么将返回连接错误。

要点: 单个应用程序只能连接到一个 SMA 服务器。但是，SMA 应用程序可以对任何本地或远程服务器定期发出基于网络的连接。

ODBC API

使用 ODBC API 时，在 SQLConnect 函数调用中定义特定于 SMA 的连接字符串。

示例

以下 ODBC API 代码示例将使用用户名 dba 和密码 dba 直接连接到本地 SMA solidDB 服务器:

```
rc = SQLConnect(hdbc, "sma tcp 1315", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

或者

```
rc = SQLConnect(hdbc, "sma upipe SOLID", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

SA API

使用 SA API 时，在 SaConnect 函数调用中定义特定于 SMA 的连接字符串。

以下代码示例将使用用户名 dba 和密码 dba 直接连接到本地 SMA solidDB 服务器:

```
SaConnectT* sc = SaConnect("sma tcp 1315", "dba", "dba");
```

或者

```
SaConnectT* sc = SaConnect("sma upipe SOLID", "dba", "dba");
```

驱动程序管理器

使用驱动程序管理器时，在 SMA 数据源中定义特定于 SMA 的连接字符串。

启动和关闭 SMA 服务器

启动、重新启动和关闭 SMA 服务器与基于常规网络协议的 solidDB 服务器的方式相似。

启动 SMA 服务器

SMA 服务器使用命令提示符启动，其方式与基于常规网络协议的 solidDB 服务器相同。SMA 服务器启动时，它会检查数据库是否已存在。服务器首先查找 solid.ini 配置文件并读取 FileSpec 参数的值。如果使用在 FileSpec 参数中指定的名称和路径找到了数据库文件，那么此数据库将自动打开。如果未找到数据库，那么服务器将提示您创建新的数据库。

过程

要启动 SMA 服务器:

表 12. 启动 SMA 服务器

操作系统	要启动 SMA 服务器:
Linux 和 UNIX	在命令提示符上输入命令 solidsma。 首次启动服务器时，请在命令提示符上输入命令 solidsma -f 以强制服务器在前台中运行。
Windows	在命令提示符上输入命令 solidsma。

结果

服务器以 SMA 方式启动时，它会动态装入 SMA 驱动程序库，在常规侦听端口处接受来自 SMA 驱动程序的连接请求。通过将不同的端口号分配给不同的 SMA 服务器，可以同时在单个系统上运行多个 SMA 服务器。

提示: 也可以通过让应用程序调用 SSC API 函数 SSCStartSMAServer 来以 SMA 方式启动 solidDB 服务器。但是，在这样的设置中，只有一个应用程序可以启动（并停止）SMA 服务器。有关 SSC API 调用的详细信息，请参阅 第 75 页的『SSCStartSMAServer』。

关闭 SMA 服务器

SMA 服务器使用 solidDB ADMIN COMMAND 关闭。

过程

1. 为了防止与 solidDB 建立新的连接，请通过输入以下命令来关闭数据库:

   ```
   ADMIN COMMAND 'close'
   ```

2. 通过输入以下命令来退出所有 solidDB 用户:

   ```
   ADMIN COMMAND 'throwout all'
   ```

3. 通过输入以下命令来停止 solidDB:

   ```
   ADMIN COMMAND 'shutdown'
   ```

结果

所有关闭机制都将启动同一例程，此例程会将所有已缓冲的数据写入数据库文件中，释放高速缓存存储器，最终将终止服务器程序。关闭服务器可能要花一点时间，这是因为服务器必须将所有已缓冲的数据从主内存写入磁盘中。

提示: 也可以通过让应用程序调用 SSC API 函数 SSCStopServer() 来停止处于 SMA 方式的 solidDB 服务器。只有一个应用程序可以启动和停止 SMA 服务器。启动 SMA 服务器的同一应用程序必须也执行关闭操作。有关详细信息，请参阅 第 77 页的『SSCStopServer』。

监视 SMA

solidDB 包括了用于监视和收集有关 SMA 连接的类型和数量以及 SMA 内存段大小的数据的方法。

- 使用 ADMIN COMMAND 'userlist' 来打印用户连接类型的列表（网络客户机或 SMA 客户机）。
- 使用 ADMIN COMMAND 'report' 来按类型打印连接列表。
- 检查 solmsg.out 文件登录条目以获取连接类型。
- 使用性能计数器 pmon xxx 来收集有关 SMA 连接数量的数据。
- 使用性能计数器 pmon yyyy 来收集有关 SMA 内存段大小的数据。

有关使用 ADMIN COMMAND、solmsg.out 和性能计数器的详细信息，请参阅《*IBM solidDB 管理员指南*》。

对 SMA 进行故障诊断

本节提供了配置或使用 SMA 时，如何阻止常见问题并对其进行故障诊断的指示信息和准则。

错误: 服务器无法根据“id -1”来分配共享内存段

症状

尝试启动 SMA 服务器时，将显示以下类型的错误并且无法启动 SMA。

```
IBM solidDB 进程遇到内部错误且无法正常继续。请将以下信息报告给技术支持人员。
SOLID Fatal error: Out of central memory when allocating buffer memory (size = 33554432)
Date: 2009-08-24 15:39:44
Product: IBM solidDB
Version: 99.99.0.0 Build 0096

[rd@bench12]~ ./solidsma -f -c .
Server could not allocate shared memory segment by id -1
```

原因

由于没有内存可用，所以 SMA 服务器启动失败。出现此情境的原因可能是由于出现了以下情况:

- SMA 应用程序或 solidDB 崩溃时，它们可能留下了共享内存。即使您关闭了所有 SMA 进程，仍有可能会保留共享内存。
- 您分配给 SMA 使用的内存太少。

这样将导致出现所有内存都被使用并且您无法再启动 SMA 服务器的情境。

解决此问题

在 Linux 和 UNIX 环境中，使用 ipcrm 命令清除暂挂共享内存段。

例如，在 Linux 环境中，请使用以下脚本来标识并除去未使用的共享内存段。

```
#!/bin/sh

if [ $# -ne 1 ]
then
   echo "$0 user"
   exit 1
fi

for shm_id in $(ipcs -m|grep $1|awk -v owner=$1 ' { if ( owner == $3 ) {print $2} }')
do
  ipcrm -m $shm_id
done
```

有关 ipcrm 命令的更多详细信息，请参阅您的操作系统文档。

3 创建并运行使用 Java 的 SMA 应用程序

Java 应用程序将链接到 SMA 驱动程序库。实际的数据库连接使用常规 JDBC API 来完成。

将 SMA 与 Java 配合使用的概述

创建使用 SMA 的 Java 应用程序与创建使用常规 solidDB 服务器的应用程序的方式相同，区别在于前者您是启动 SMA 服务器，而后者您是启动常规 solidDB 服务器。 Java 应用程序连接至 SMA 服务器并使用 solidDB 服务器通过标准 JDBC API 提供的服务。通过链接至动态库，此应用程序避免了通过网络进行 RPC（远程过程调用）所产生的开销。

要使用 SMA 链接至 SMA 驱动程序库（`ssolidsmaxx`）的 Java/JDBC 程序。此库包含整个 solidDB 服务器，只不过它是采用可调用库形式，而不是采用独立的可执行程序形式。与 Java/JDBC 配合使用的库和与 C/C++ 应用程序配合使用的库相同；没有单独用于 Java 的版本。

将 SMA 与 Java/JDBC 配合使用时，请将以下各项链接到单个可执行进程中:

- SMA 驱动程序库
- Java 语言客户机程序，以及
- JVM。

可执行进程中的层从上到下依次为:

- 本地 Java/JDBC 客户机应用程序
- JVM（Java 虚拟机）
- SMA 驱动程序库

客户机中的 Java 命令由 JVM 执行。如果命令是 JDBC 函数调用，那么 JVM 在 SMA 驱动程序库中调用相应的函数。函数调用是直接进行的，不通过网络（通过 RPC）来调用。调用使用 Java 本机接口（JNI）来执行。您不需要自己编写任何 JNI 代码；您只需要调用远程客户机程序将调用的 JDBC 函数。

使用 SMA 的每个应用程序都遵循相同的四步骤基本模式:

1. 配置 solidDB 服务器和连接设置。
2. 启动 SMA 服务器。
3. 使用常规 JDBC API 来访问数据库。
4. 完成数据库处理之后，停止 SMA 服务器。

为将 SMA 与 Java 配合使用而配置环境

将 SMA 与 Java 配合使用时，您的 PATH（Windows）或 LD_LIBRARY_PATH（Linux 和 UNIX）环境变量必须包括 SMA 驱动程序库的位置。

开始之前

假设您拥有 solidDB JDBC 驱动程序的有效安装。

关于此任务

SMA 驱动程序库文件在安装 solidDB 时安装。下表列出了最常用平台的文件名及其缺省安装位置。

表 13. SMA 驱动程序（库）

平台	**SMA 驱动程序库**	**缺省安装位置**
Windows	ssolidsmaxx.dll **注**: 如果要直接链接到 SMA驱动程序，那么链接到 solidsma.lib 导入库文件，通过该文件可访问实际的 .dll 库文件。	库: <solidDB 安装目录>\bin 导入库: <solidDB 安装目录>\lib
Linux	ssolidsmaxx.so	<solidDB 安装目录>/bin
Solaris	ssolidsmaxx.so	<solidDB 安装目录>/bin
HP-UX	ssolidsmaxx.sl	<solidDB 安装目录>/bin
AIX	ssolidsmaxx.so	<solidDB 安装目录>/bin
xx 是驱动程序库的版本号，例如，ssolidsma65.so。		

过程

1. **将 SMA 驱动程序库的位置添加到 PATH（Windows）或 LD_LIBRARY_PATH（Linux 和 UNIX）环境变量。**
   - 在 Windows 环境中，使用以下语法:

     ```
     set PATH=<path to SMA library>=;%PATH%
     ```
   - 在 Linux 和 UNIX 环境中，使用以下语法:

     ```
     export LD_LIBRARY_PATH=<path to SMA library>:$LD_LIBRARY_PATH
     ```
2. **通过创建工作目录、solidDB 数据库和用户帐户来设置数据库环境。**

   有关指示信息，请参阅《*IBM solidDB 管理员指南*》中的『*创建新的数据库*』一节。
3. **配置 solidDB 以满足您的环境、性能和操作需求。**

   使用 solid.ini 配置文件来定义基本的配置设置，如数据库文件名和位置以及数据库块大小等。
   - 在常规设置中，不必修改 solid.ini 文件的 [SharedMemoryAccess] 部分中特定于 SMA 的参数；出厂值适用于大多数情况。
   - **Srv.ProcessMemoryLimit** 参数对于 SMA 服务器无效。

   如果没有配置文件，那么将使用缺省值。

启动和关闭 SMA 服务器

启动、重新启动和关闭 SMA 服务器与基于常规网络协议的 solidDB 服务器的方式相似。

启动 SMA 服务器

SMA 服务器使用命令提示符启动，其方式与基于常规网络协议的 solidDB 服务器相同。SMA 服务器启动时，它会检查数据库是否已存在。服务器首先查找 solid.ini 配置文件并读取 FileSpec 参数的值。如果使用在 FileSpec 参数中指定的名称和路径找到了数据库文件，那么此数据库将自动打开。如果未找到数据库，那么服务器将提示您创建新的数据库。

过程

要启动 SMA 服务器:

表 14. 启动 SMA 服务器

操作系统	要启动 SMA 服务器:
Linux 和 UNIX	在命令提示符上输入命令 solidsma。 首次启动服务器时，请在命令提示符上输入命令 solidsma -f 以强制服务器在前台中运行。
Windows	在命令提示符上输入命令 solidsma。

关闭 SMA 服务器

SMA 服务器使用 solidDB ADMIN COMMAND 关闭。

过程

1. 为了防止与 solidDB 建立新的连接，请通过输入以下命令来关闭数据库:

   ```
   ADMIN COMMAND 'close'
   ```

2. 通过输入以下命令来退出所有 solidDB 用户:

   ```
   ADMIN COMMAND 'throwout all'
   ```

3. 通过输入以下命令来停止 solidDB:

   ```
   ADMIN COMMAND 'shutdown'
   ```

为 SMA 建立 JDBC 连接

为了进行与 SMA 服务器的本地（非基于 RPC）的 JDBC 连接，您需要定义自己正使用 solidDB 特定的连接属性 solid_shared_memory 连接到 SMA 服务器，且正在给定的端口使用本地服务器。

连接驱动程序管理器

在代码中包括非标准的连接属性 solid_shared_memory。

例如:

```
Properties props = new Properties();
// enable the direct access property
props.put("solid_shared_memory", "yes");
// get connection
Connection c = DriverManager.getConnection
("jdbc:solid://localhost:1315", props);
```

在连接字符串中定义连接属性

在连接字符串中包括连接属性。

例如:

```
Connection c = DriverManager.getConnection
("jdbc:solid://localhost:1315?solid_shared_memory=yes");
```

注: 除了 DriverManager 类，类似的语法还可用于 SolidDataSource 和 SolidConnectionPoolDataSource 类。

4 创建并运行 LLA 应用程序

创建 LLA 应用程序包括将应用程序链接到库，启动服务器并在应用程序和服务器之间建立本地连接。您可以使用 SSC API、ODBC API 和 SA API 来启动和停止服务器。

特定于 LLA 的指示信息提供了与没有 LLA 的 solidDB 相比较时的增强、补充和使用差异。

有关 solidDB SQL、solidDB 数据管理工具、一般 solidDB 管理和维护以及数据库错误代码的信息，请参阅《*IBM solidDB 管理员指南*》。

有关 API 和 solidDB JDBC 以及 ODBC 驱动程序的详细信息，请参阅 *IBM solidDB Programmer Guide*。

配置环境以供 LLA 使用

使用 LLA 时，您必须将应用程序链接至 LLA 库文件。

过程

1. **将应用程序链接至特定于操作系统的 LLA 库文件。**

表 15. 链接库访问（LLA）系统库

平台	LLA 库
Windows	`solidimpac.lib` 这是让您可以访问实际库文件 `ssolidacxx.dll` 的导入库文件。
Solaris	`solidac.a`
HP-UX	`solidac.a`
Linux	`solidac.a`

示例: 在 Windows 中提供 LLA 库名的 Makefile

在以下 makefile 示例中，使用了 `solidimpac.lib` 库。

```
# compiler
CC        = cl
# compiler flags
CFLAGS        = -I. -DSS_WINDOWS -DSS_WINNT
# linker flags and directives
SYSLIBS = libcmt.lib kernel32.lib advapi32.lib netapi32.lib wsock32.lib
user32.lib oldnames.lib gdi32.lib
LFLAGS  = ..\solidimpac.lib
OUTFILE  = -Fe

# MyApp building
all: myapp
```

```
myapp: myapp.c
  $(CC) $(CFLAGS)  $(OUTFILE)myapp myapp.c /link$(LFLAGS)
/NODEFAULTLIB:libc.lib
```

2. **如果不打算使用隐式启动方法来使用 SSC API 启动 solidDB 服务器，请将 ImplicitStart 参数设置为“no”。**

   在 solid.ini 配置文件的 [Accelerator] 节中，缺省情况下，参数 **ImplicitStart** 设置为 Yes。当您使用任何 ODBC 连接必需的 SQLConnect 函数时，此缺省设置将自动启动服务器。SaConnect 函数的行为相同。首次调用 SQLConnect 或 SaConnect 时，将隐式启动服务器。

3. **禁用信号处理程序。**

   信号处理程序用来向应用程序报告发生了意外事件（例如，除数为 0）。不能在用户应用程序中设置信号处理程序，这是因为它们将覆盖由链接库访问设置的信号处理程序。例如，如果用户应用程序为浮点异常设置了信号处理程序，那么该设置将覆盖链接库访问设置的处理程序。因此服务器无法捕获诸如“除数为 0”等异常情况。

建立 LLA 的本地连接

一旦应用程序链接至链接库访问库，它就可以使用 ODBC API 或者 SA API 直接与本地服务器建立本地或远程连接。应用程序还可以与其他 solidDB 服务器（包括其他使用链接库访问的服务器）建立远程连接。

在 ODBC API 中，要建立与本地服务器（该服务器链接到应用程序）的连接，用户应用程序使用文字串“localserver”调用 SQLConnect 函数。请注意，对于本地服务器连接，也可以指定一个空源名“”。您也可以指定本地服务器名称，但是这将导致链接库访问使用“远程”连接（即，通过网络来建立连接，而不是通过直接对链接库访问库进行函数调用来建立连接）。

以下 ODBC API 代码示例使用用户名 dba 和密码 dba 直接连接至本地 solidDB 服务器:

```
rc = SQLConnect(hdbc, "localserver", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

或者

```
rc = SQLConnect(hdbc, "", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

在 SA API 中，要建立连接，用户应用程序将使用文字串“localserver”（而不是服务器名称）来调用 SaConnect 函数。请注意，对于本地服务器连接，也可以指定一个空源名“”。您也可以指定本地服务器名称，但是这将导致链接库访问使用“远程”连接（即，通过网络来建立连接，而不是通过直接对链接库访问库进行函数调用来建立连接）。

以下 SA API 示例代码使用用户名 dba 和密码 dba 直接连接至 solidDB 服务器:

```
SaConnectT* sc = SaConnect("localserver", "dba", "dba");
```

或者

```
SaConnectT* sc = SaConnect("", "dba", "dba");
```

启动和关闭 LLA 服务器

您可以使用 SSC API、ODBC API 或 SA API 函数调用来启动、重新启动和关闭 LLA 服务器。仅当已存在数据库时，才可以使用 ODBC API 和 SA API 函数调用来启动服务器。可使用 SSC API 在启动时创建新的数据库。

在服务器启动时，在将控制权归还给应用程序之前，如果需要，可执行恢复。因此，如果服务器已成功启动，那么它就可以开始处理应用程序请求。在应用程序进程的持续时间内，可以根据需要来启动或停止服务器。

使用 SSC API 来显式启动和关闭

使用 SSC API 来显式启动和关闭 LLA 服务器。应用程序调用 SSC API 函数 SSCStartServer 来启动 solidDB，并调用 SSCStopServer 函数来将其关闭。

当您启动没有数据库的新 LLA 服务器时，您必须通过提供以下参数以及 SSCStartServer() 函数来显式指定创建新的数据库:

```
-Uusername
-Ppassword
-Ccatalogname（缺省数据库目录名）
```

注: 如果要启动无盘服务器，那么必须使用 SSC API 函数 SSCStartDisklessServer 来启动该服务器。
有关详细信息，请参阅『使用 SSC API 函数 SSCStartServer 显式启动』。

使用 ODBC API 和 SA API 进行隐式启动和关闭

ODBC API 和 SA API 只能用于隐式启动和关闭 LLA 服务器。应用程序首次本地连接到 LLA 服务器时，它会调用 ODBC API 函数 SQLConnect 或 SA API 函数 SaConnect。在这种情况下，服务器将在通过调用 SQLDisconnect 或 SaDisconnect 函数使最后一个本地连接与 solidDB 断开连接时停止。

当 LLA 服务器从应用程序隐式启动时，它会检查数据库是否已存在于 solidDB工作目录中。如果找到了数据库文件，那么 solidDB 将自动打开该数据库。如果未找到数据库文件，那么 solidDB 将报告错误。

solidDB 将不会在隐式启动期间创建新的数据库。要创建新的数据库，您必须使用显式启动函数（如 SSCStartServer 以及相应的参数）或像对待常规 solidDB 服务器那样创建数据库。

有关详细信息，请参阅第 33 页的『使用 ODBC API 函数调用 SQLConnect 来隐式启动』和第 34 页的『使用 SA API 函数调用 SaConnect 来隐式启动』。

有关在常规 solidDB设置中如何创建数据库的指示信息，请参阅*《IBM solidDB 管理员指南》*中的『*创建新的数据库*』一节。

使用 SSC API 函数 SSCStartServer 显式启动

要显式启动 solidDB，用户应用程序应调用以下控制 API 函数:

要显式启动 solidDB，用户应用程序应调用以下 SSC API 函数:

```
SSCStartServer (int argc, char* argv [ ],
SscServerT* h, SscStateT runflags)
```

其中，参数是:

表 16. SSCStartServer 参数

参数	描述
argc	命令行参数的数目。
argv	在执行函数调用期间使用的命令行参数组成的数组。argv[0] 参数是仅为用户应用程序的路径和文件名保留的，并且必须提供此参数。有关有效选项，请参阅下面的 SSCStartServer 选项。
h	每个服务器都有一个“句柄”（就是一个指向数据结构的指针）用于标识此服务器并指示有关此服务器的信息的存储位置。当使用其他控制 API 函数来引用此服务器时，此句柄是必需的。将在调用 SSCStartServer 函数时为您提供此服务器的句柄。 要获取服务器的句柄，请创建一个类型为 pointer-to-server-handle 的变量（即创建一个 SSCServerT *，它是一个指向句柄的指针，实质上是一个指向指针的指针），并在调用 SSCStartServer 时传递此变量。如果成功创建了服务器，那么 SSCStartServer 函数会将新服务器的句柄（指针）写入您为其传递了地址的变量中。
runflags	此参数的选项为 SSC_STATE_OPEN（允许建立远程连接）和 SSC_STATE_PREFETCH（需要时服务器将执行预取）。预取涉及到内存和/或磁盘高速缓存，磁盘高速缓存提供了对表内容的预读功能。请参阅下面的 runflags 参数条目: `runflags = SSC_STATE_OPEN | SSC_STATE_PREFETCH;`

在没有现存的数据库的情况下启动 LLA 服务器

当您首次启动服务器时，仅当您已经指定了数据库管理员的用户名和密码以及缺省数据库目录名时，solidDB 才会创建新的数据库。

例如:

```
SscServerT h; char* argv[4];
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-UDBA"; /* user name */
argv[2] = "-PDBA"; /* user's password */
argv[3] = "-CDBA"; /* catalog name */
/* Start the server */
rc = SSCStartServer(argc, argv, &h, run_flags);
```

如果您启动的服务器上没有现存的数据库并且未指定数据库目录名，那么 solidDB 将返回一个错误并指出找不到数据库。

缺省情况下，将在 solidDB 工作目录中将数据库创建为一个文件（可以使用缺省名称 `solid.db`，也可以使用您在 `solid.ini` 文件中指定的名称）。一个只包含系统表和视图的空数据库将占用大约 850 KB 磁盘空间。创建数据库所花的时间取决于您使用的硬件平台。

创建数据库之后，solidDB 将开始侦听网络，以了解是否有远程客户机连接请求。

使用现有数据库来启动 LLA 服务器

如果您已经有一个现存的数据库，那么不需要在 SSCStartServer 函数调用中指定用户名和密码或者目录名。

使用 ODBC API 函数调用 SQLConnect 来隐式启动

首次调用 SQLConnect 函数时，将隐式启动服务器。当用户应用程序调用 SQLDisconnect 函数并且这是最后一个打开的本地连接时，服务器将隐式关闭。请注意，无论当前是否具有远程连接，服务器都将关闭。

注: 当您首次启动服务器时，必须使用 SSCStartServer() 函数并且指定缺省数据库目录以及管理员的用户名和密码来创建 solidDB 数据库。有关描述和示例，请阅读第 31 页的『使用 SSC API 函数 SSCStartServer 显式启动』。

下面是一个使用 SQLConnect 和 SQLDisconnect 进行隐式启动和关闭的示例:

```
/* Connection #1 */
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); //Server Started Here
... odbc calls

/* Disconnect #1 */
SQLDisconnect (hdbc1); //Server Shut Down Here

/* Connection #2 */
rc = SQLConnect (hdbc2, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); //Server Started Here
... odbc calls

/* Disconnect #2 * /
SQLDisconnect (hdbc2); //Server Shut Down Here
```

或者

```
/* Connection #1*/
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba",
SQL_NTS, "dba", SQL_NTS); // Server Started Here

/* Connection #2*/
rc = SQLConnect (hdbc2, "", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);

... odbc calls

/* Disconnect #1 */
SQLDisconnect (hdbc1);
/* Disconnect #2 * /
SQLDisconnect (hdbc2); // Server Shut Down Here
```

注: 如果服务器是通过调用 SSCStartServer 函数启动的，那么 SQLDisconnect 不会隐式关闭。必须通过 SSCStopServer、ADMIN COMMAND 'shutdown' 或其他显式关闭方法来显式关闭服务器。

```
SscStateT runflags = SSC_STATE_OPEN;
SscServerT server;
SQLHDBC hdbc;
SQLHENV henv;
SQLHSTMT hstmt;

/* Start the server */
SSCStartServer (argc, argv, &server, runflags); // Server Started Here

/* Alloc environment */
```

```
rc = SQLAllocEnv (&henv);

/* Connect to the database */
rc = SQLAllocConnect (henv, &hdbc);
rc = SQLConnect (hdbc, "", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);

/* Delete all the rows from table foo */
rc = SQLAllocStmt (hdbc, &hstmt):
rc = SQLExecDirect (hsmt, (SQLCHAR *) "DELETE FROM FOO", SQL_NTS);

/* Commit */
rc = SQLTransact (henv, hdbc, SQL_COMMIT);
rc = SQLFreeStmt (hstmt, SQL_DROP);

/* Disconnect */
SQLDisconnect (hdbc);
SQLFreeConnect (hdbc);

/* Free the environment */
SQLFreeEnv(henv);

/* Stop the server */
SSCStopServer (server, TRUE);  // Server Shut Down Here
```

使用 SA API 函数调用 SaConnect 来隐式启动

首次调用 SaConnect 函数时，将隐式启动服务器。当用户应用程序调用 SaDisconnect 函数并且没有后续连接时，服务器将隐式关闭。

注: 当您首次启动服务器时，必须使用 SSCStartServer() 函数并且指定缺省数据库目录以及用户名和密码来创建 solidDB 数据库。有关描述和示例，请阅读第 31 页的『使用 SSC API 函数 SSCStartServer 显式启动』。

下面是一个使用 SaConnect 和 SaDisconnect 进行隐式启动和关闭的示例:

```
/* Open Connection */
SaConnect(...);

Server Started Here
... sa calls

/* Close Connection */
SaDisconnect(...);

Server Shut Down Here
```

注: 如果服务器是使用 SSCStartServer 函数调用启动的，那么只能通过 SSCStopServer 函数调用来关闭此服务器。

关闭 LLA 服务器

只要您有 SYS_ADMIN_ROLE 特权，就可以从 solidDB 客户机界面，甚至从另一远程 solidDB 连接来关闭 solidDB 服务器。

您可以程序化地从 solidDB SQL 编辑器（solsql）或 solidDB 远程控制台之类的应用程序来执行关闭[1].

为此，请执行下列步骤:

1. 对步骤 1 至 3 使用 solidDB 远程控制时，输入命令名称时不能使用引号（例如，close）。

1. 为了防止与 solidDB 建立新的连接，请通过输入以下命令来关闭数据库:

   ADMIN COMMAND 'close'

2. 通过输入以下命令来退出所有 solidDB 用户:

   ADMIN COMMAND 'throwout all'

3. 通过输入以下命令来停止 solidDB:

   ADMIN COMMAND 'shutdown'

所有关闭机制都将启动同一例程，此例程会将所有已缓冲的数据写入数据库文件中，释放高速缓存存储器，最终将终止服务器程序。关闭服务器可能要花一点时间，这是因为服务器必须将所有已缓冲的数据从主内存写入磁盘中。

注: 您可以使用显式方法 SSCStopServer 来关闭使用隐式方法（SQLConnect）启动的服务器。反向操作则不行；例如，不能使用 SQLDisconnect 来停止先前使用 SSCStartServer 启动的服务器。

使用 SSCStopServer 关闭 LLA 服务器

如果服务器是通过 SSCStartServer 启动的，那么必须通过在嵌入式应用程序中执行以下函数调用来关闭此服务器:

SSCStopServer()

例如:

```
/* Stop the server * /
SSCStopServer (h, TRUE);
```

用于 LLA 的样本 C 应用程序

solidDB 程序包包括用 C 语言编写的 LLA 应用程序的样本，这些应用程序使用 ODBC API 函数连接至 solidDB 服务器。

样本位于 solidDB 安装目录中的以下目录中:

- `samples/aclib`: 使用单个 solidDB 的 LLA 应用程序的样本
- `samples/aclib_control_api`: 使用 SSC API 函数的 LLA 应用程序的样本。
- `samples/aclib_diskless`: 使用无盘 solidDB 服务器的 LLA 应用程序的样本
- `samples/aclib_replication`: 结合了 LLA 和高级复制的 LLA 应用程序的样本

  请参阅『将 LLA 用于高级复制的样本』以获取有关如何使用复制样本的信息。

每个目录都包含一个 `readme.txt` 文件，该文件提供了有关如何设置系统并运行样本的指示信息。

将 LLA 用于高级复制的样本

如果您对 solidDB 数据同步不熟悉，*IBM solidDB Advanced Replication User Guide* 中包含的信息将描述如何使用 solidDB 随附的样本脚本。

在运行样本 C 应用程序 acsnet.c（在 samples/aclib_replication 目录下）之前，建议您通过执行至少以下一项操作来熟悉 solidDB 功能:

- 使用 solidDB（没有 SMA 或 LLA）来运行 *IBM solidDB Advanced Replication User Guide* 中包含的 SQL 脚本。可在 samples/replication 中找到以下脚本。
- 使用 solidDB SMA 或 LLA 在本地运行 SQL 脚本。您必须根据此文档中的指示信息设置应用程序以启动服务器。

注: 不能使用 SA API 来运行同步命令。

- 运行实现样本文件 aclibstandalone.c（它带有链接库访问库）时将模拟正常的服务器。此样本文件位于 samples/aclib 目录下。

在使用以上任何一种方法之后，都可以使用 solidDB SQL 编辑器（solsql）来运行 *Getting Started with Data Synchronization* 的 *IBM solidDB Advanced Replication User Guide* 一章中的所有步骤。

使用高级复制样本脚本来设置 ODBC 应用程序

可以构建一个与样本 C 应用程序 acsNet.c 相似的 ODBC 应用程序，用来执行在设置、配置和运行同步环境时所需要的所有语句。在 samples/aclib_replication 目录下可找到 acsNet.c。

要设置样本数据库以与 ODBC 客户机应用程序配合使用，可以执行样本脚本 replica3.sql、replica4.sql、replica5.sql 和 replica6.sql，所有这些脚本都可在 samples/replication/eval_setup 目录中找到。这些样本脚本中包含一些 SQL 语句，可用来将新数据写入副本以及控制同步消息的执行。可以通过 solidDB SQL 编辑器（solsql）来独立运行每个脚本。

或者，您可以将 SQL 语句嵌入 C/ODBC 应用程序中，编译此应用程序，然后将它直接链接至链接库访问库。链接至链接库访问库之后，使用样本脚本时就能获得链接库访问体系结构的性能优势。

samples/odbc 目录中的样本程序 embed.c 说明了如何通过使用链接库访问来设置具有 ODBC 客户机应用程序的数据库。您可以将样本脚本中的 SQL 命令（例如，replica3.sql）插入 embed.c 应用程序中。

5 创建并运行使用 Java 的 LLA 应用程序

Java 应用程序将链接到 LLA 库，它们使用 solidDB SSC API 调用来启动并停止 solidDB。实际的数据库连接使用常规 JDBC API 来完成。 SolidServerControl API 调用和 JDBC 驱动程序包括在 solidDB JDBC Driver .jar 文件（SolidDriver2.0.jar）中。

将 LLA 与 Java 配合使用的概述

LLA 使 Java 应用程序可以启动本地 solidDB 服务器，该服务器将从动态库装入到 Java 虚拟机上下文中。然后，Java 应用程序将能够连接至 solidDB 服务器并使用 solidDB 服务器通过标准 JDBC API 提供的服务。通过链接至动态库，此应用程序避免了通过网络进行 RPC（远程过程调用）所产生的开销。

要使用 LLA 链接至 LLA 库（ssolidacxx）的 Java/JDBC 程序。此库包含整个 solidDB 服务器，只不过它是采用可调用库形式，而不是采用独立的可执行程序形式。与 Java/JDBC 配合使用的库和与 C/C++ 应用程序配合使用的库相同；没有单独用于 Java 的版本。

将 LLA 与 Java/JDBC 配合使用时，请将以下各项链接到单个可执行进程中:

- LLA 库,
- Java 语言客户机程序，以及
- JVM。

可执行进程中的层从上到下依次为:

- 本地 Java/JDBC 客户机应用程序
- JVM（Java 虚拟机）
- LLA 库

客户机中的 Java 命令由 JVM 执行。如果命令是 JDBC 函数调用，那么 JVM 将在 ssolidacxx 中调用相应的函数。函数调用是直接进行的，不通过网络（通过 RPC）来调用。调用使用 Java 本机接口（JNI）来执行。您不需要自己编写任何 JNI 代码；您只需要调用远程客户机程序将调用的 JDBC 函数。

使用 LLA 访问 solidDB 数据库与通过 RPC 访问 solidDB 数据库相同，只有一处例外，为了访问数据库服务，使用 LLA 的应用程序必须首先启动 LLA 服务器。使用名为用于 Java 的 solidDB 服务器控制（SSC）API 的专用 API（根据 SolidServerControl 类命名）来完成此操作。实际的数据库连接使用常规 solidDB JDBC API 来完成。用于 Java 的 SSC API 和 solidDB JDBC 驱动程序都可在名为 SolidDriver2.0.jar 的 .jar 文件中找到。

启动本地 solidDB 服务器之后，它会从动态库装入到 Java 虚拟机上下文中。然后，Java 应用程序将能够连接到 solidDB 服务器并使用该服务器通过标准 JDBC API 提供的服务。

使用 SMA 或 LLA 的每个应用程序都遵循相同的四步骤基本模式:

1. 配置 solidDB 和连接设置。
2. 使用 SolidServerControl 类来启动 LLA 服务器。
3. 使用常规 JDBC API 来访问数据库。
4. 完成数据库处理之后，再次使用 SolidServerControl 类来停止 LLA 服务器。

局限性

- 将 LLA 与 Java 配合使用时，所有的 solidDB“admin commands”都不能使用。
- 如果在 VM 上下文外部（例如，在本机方法调用中）发生了故障，那么 Java 的行为不一致。如果在 solidDB 服务器本机代码中应当对某个对象作出断言（甚至使它崩溃），那么 Java 将异常退出或彻底挂起。在后面这种情况下，您可能必须手动终止悬挂的 Java 进程。
- 要将内存消耗降至最低，您应显式丢弃所有的分配语句；即，必须通过调用 close() 方法来显式释放所有分配的 JDBC 语句对象。这一点很重要，特别是如果您的设置正在使用透明连接（TC）。

为将 LLA 与 Java 配合使用而配置环境

将 LLA 与 Java 配合使用时，您的 PATH（Windows）或 LD_LIBRARY_PATH（Linux 和 UNIX）系统变量必须包括 LLA 库的位置。

开始之前

假设您已经安装并注册了 solidDB JDBC 驱动程序。

过程

1. **将 LLA 库的位置添加到 PATH（Windows）或 LD_LIBRARY_PATH（Linux 和 UNIX）系统变量。**
   - 在 Windows 环境中，使用以下语法:

     ```
     set PATH=<path to LLA library>=;%PATH%
     ```

   - 在 Linux 和 UNIX 环境中，使用以下语法:

     ```
     export LD_LIBRARY_PATH=<path to LLA library>:$LD_LIBRARY_PATH
     ```

2. **通过创建工作目录、solidDB 数据库和用户帐户来设置数据库环境。**

   有关指示信息，请参阅《*IBM solidDB 管理员指南*》中的『*创建新的数据库*』一节。

通过用于 Java 的 SSC API 来启动并停止 LLA 服务器

要从 Java 应用程序启动 solidDB 服务器，必须在应用程序的开头将 SolidServerControl 类实例化并使用正确的参数来调用 `ssc.startServer` 方法。启动服务器后，您就可以与服务器建立 JDBC 连接。类似地，可以使用 `ssc.stopServer` 调用来停止服务器。

过程

1. **启动服务器**
   - **LLA 服务器**: `ssc.startServer`

   启动服务器时，必须向 solidDB 服务器传递至少以下参数:

```
-c<solidDB working directory containing license file>
-U<username>
-P<password>
-C<catalog>
```

请注意，同时使用了大写字母“C”和小写字母“c”，它们代表不同的含义。

2. **停止服务器**
   - **LLA 服务器**: `ssc.stopServer`

为 LLA 建立 JDBC 连接

solidDB Java 加速器支持本地数据库连接以及基于 RPC 的连接。

为了建立本地 JDBC 连接（而不是基于 RPC 的 JDBC 连接），需要指定您要在端口 0 使用“localserver”的 JDBC 驱动程序。因此，如果您要通过使用诸如 JDBC 类 DriverManager 来建立数据库连接，请使用以下语句来进行连接（在非常靠后的示例代码 SJASample 中也提供了此语句）：

```
DriverManager.getConnection("jdbc:solid://localserver:0", myLogin, myPwd);
```

正如您所知道的，DriverManager 使用 URL *"jdbc:solid://localserver:0"* 与本地服务器建立连接。如果为 getConnection 子例程提供了另一个 URL，那么驱动程序可能将尝试使用 RPC 建立连接。

因此，当您建立 Java 加速器连接时，请记住 URL –

```
jdbc:solid://localserver:0
```

建立 Java 加速器连接。

注:

如果您要使用多个将访问 Java 应用程序中的 solidDB 链接库访问服务器的线程（`java.lang.Thread` 对象），那么在使用某个线程来启动与 JDBC 相关的任何活动之前，必须向 solidDB 链接库访问服务器单独注册此线程。线程的注册是通过在此线程的上下文中调用 SolidServerControl API 的 registerThread 方法来完成的。必须为使用 solidDB 的 JDBC 驱动程序的每个用户线程（主线程除外）显式完成线程注册。

用户还必须显式注销已向 solidDB 链接库访问服务器注册的每个线程。要注销线程，请调用 SolidServerControl API 的 unregisterThread 函数。

编译并运行样本 LLA 程序

关于此任务

本过程中的示例针对 Windows 命令提示符提供。

过程

1. 设置路径。

   ```
   set PATH=<path to your ssolidacxx DLL>;%PATH%
   ```

   请确保您提供的路径中也有一个包含 solidDB 通信库的目录。

2. 将 PATH 环境变量设置为包含 JDK 的 HOTSPOT 运行时环境（SJA 仅在热点 JRE 中进行了测试）。

   例如:

   ```
   set PATH=<your JDK directory>\jre\bin\hotspot;%PATH%
   ```

3. 使用以下命令编译样本 SJASample.java 文件（位于 samples/aclib_java 目录中）:

   ```
   javac -classpath <IBM solidDB JDBC driver directory>/
   SolidDriver2.0.jar;. \
   SJASample.java
   ```

4. 使用与以下命令行类似的命令行来运行样本应用程序:

   ```
   java -Djava.library.path=<path to ssolidacxx DLL> \ -classpath
   <IBM solidDB JDBC driver directory>/SolidDriver2.0.jar;. \ SJAsample
   ```

   例如，如果您将服务器安装到 C:\soliddb 并且要运行 SJASample 程序，那么命令行将类似于:

   ```
   java -Djava.library.path=C:\soliddb\bin
   -classpath C:\soliddb\jdbc\SolidDriver2.0.jar;. SJASample
   ```

   在 Windows 上，ssolidacxx.dll 动态库位于 solidDB 根安装目录的 bin 子目录中。

   与在示例类 SJASample 中一样，必须使用 SolidServerControl 的 startServer 方法为 solidDB 服务器至少传递下列参数:

   ```
   -c<directory containing solidDB license file>
   -U<username>
   -P<password>
   -C<catalog>
   ```

   请注意，同时使用了大写字母“C”和小写字母“c”，它们代表不同的含义。

5. 如果在当前工作目录中拥有所有必需文件（ssolidacxx 库、通信库、JDBC 驱动程序和许可证文件），那么可以使用类似以下的命令行来启动 SJASample:

   ```
   java -Djava.library.path=. -classpath SolidDriver2.0.jar;. SJAsample
   ```

结果

LLA 服务器已启动且正在运行。

6 使用无盘功能

SMA 和 LLA 服务器可用于创建在没有任何磁盘存储空间的情况下运行的数据库引擎。这在没有硬盘的嵌入式系统（例如，网络路由器或交换机中的线卡）中很有用。

可以通过两种方式来运行无盘服务器：作为单个服务器（独立运行）或作为高制复制系统中的副本服务器运行。在两种情况下，您都需要通过使用 SSC API 或用于 Java 函数调用的 SSC API 来启动服务器。

SSC API

- 使用 SSCStartDisklessSMAServer 来启动无盘 SMA 服务器。
- 使用 SSCStartDisklessServer 来启动无盘 LLA 服务器。

用于 Java 的 SSC API

- 使用 startDisklessServer 来启动无盘 LLA 服务器。

无盘服务器单独运行

如果独立运行无盘服务器，那么它在启动时无法读取数据，在关闭时无法写入数据。这就意味着此服务器每次启动时都没有先前的任何数据。

由于服务器无法将数据写入磁盘，因此如果它由于电源故障之类的问题而导致异常关闭，那么此服务器中的所有数据都将丢失并且无法恢复。可以通过使用 solidDB HotStandby 组件创建一个“热备用”机器（它包含数据的副本）以降低丢失数据的风险。有关热备用功能的更多信息，请参阅《*IBM solidDB 高可用性用户指南*》。

作为高级复制系统一部分的无盘服务器

无盘服务器可以是高级复制系统中的一个副本服务器。在这种情况下，副本服务器可以将数据发送至主控服务器，并且可以从该主控服务器中下载数据。因此，尽管副本服务器没有自己的磁盘存储器或者其他永久存储器，它也可以使它的某些数据或所有数据持久保存在高级复制系统中。

7 创建并运行远程或双方式应用程序

远程应用程序可访问 SMA 和 LLA 服务器。如果远程应用程序包括 SSC API 或 SA API 函数调用，那么您必须链接至不同的 SSC API 和 SA API 库，因为远程应用程序无法访问 SMA 和 LLA 库中包含的函数。如果您的远程应用程序仅使用 ODBC 或 JDBC，那么可使用 ODBC 和 JDBC 接口来像往常一样构建应用程序。远程连接类型在连接字符串中定义。

示例: 使用 ODBC 和 SSC API 函数调用来创建双方式 LLA 应用程序

如果您的应用程序是使用 SSC API 和 ODBC 函数调用的双方式应用程序，那么需要两个不同的可执行文件，一个本地运行，另一个远程运行。

过程

1. 创建将在本地方式中运行的应用程序版本。
   a. 将应用程序链接到 LLA 库（例如，对于 Windows 是 `solidimpac.lib`）。

      LLA 库支持 ODBC 函数和 SSC API 函数。
   b. 修改连接字符串以使用本地连接。
2. 创建将在远程方式中运行的应用程序版本。
   a. 将应用程序链接到 solidDB ODBC 驱动程序和 SSC API 存根库（例如，对于 Windows 是 `solidctrlstub.lib`）。

      实际上，存根库不会让您的远程应用程序对服务器进行任何控制；它仅允许您编译和链接程序，且不会收到有关“未解析符号”的错误。
   b. 修改连接字符串以使用远程连接。

建立远程连接

建立远程连接时，应用程序将通过网络来调用服务器，而不是使用对 SMA 或 LLA 库的直接函数调用。

ODBC API

使用 ODBC API 时，要建立远程连接，应用程序将使用远程服务器的名称来调用 SQLConnect 函数。

示例

以下 ODBC API 代码示例使用用户名 dba 和密码 dba 连接至远程 solidDB 服务器。在此示例中，客户机和服务器使用的网络协议为“tcp”（TCP/IP）。服务器名为“remote_server1”，它侦听的端口为 1313。

```
rc = SQLConnect(hdbc, "tcp remote_server1 1313",
(SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

SA API

使用 SA API 时，要建立远程连接，应用程序将使用远程服务器的名称来调用 SaConnect 函数。

示例

在此示例中，客户机和服务器使用的网络协议为“tcp”（TCP/IP）。服务器名为“remote_server1”，它侦听的端口为 1313。

```
SaConnectT* sc = SaConnect("tcp remote_server1 1313", "dba", "dba");
```

JDBC

使用 JDBC 时，要建立远程连接，远程服务器的名称在连接字符串中定义。

```
jdbc:solid://<hostname>:<port>
```

附录 A. 共享内存访问参数

SMA 参数显示在 solid.ini 配置文件的 [SharedMemoryAccess] 部分中。

表 17. 共享内存访问参数

[SharedMemoryAccess]	描述	出厂值	启动
MaxSharedMemorySize	此参数设置共享内存区域大小的最大值。如果 SMA 服务器尝试分配更多内存，那么将出现“内存不足”错误。如果使用值“0”，那么最大值将自动设置为计算机物理内存的大小（特定于平台）。	0（自动）	RW/启动
SharedMemoryAccessRights	此参数设置用户访问共享内存区域时的验证上下文。它根据传统的文件验证掩模进行建模，具有组件“用户”（仅仅是与启动 SMA 服务器相同的用户）、“组”（任何属于同一个组的用户）和“全部”（所有用户）。	组	RW/启动

附录 B. 链接库访问参数

链接库访问参数显示在 solid.ini 配置文件的 [Accelerator] 节中。

表 18. 加速器参数

[Accelerator]	描述	出厂值
ImplicitStart	如果将此参数设置为 yes，那么一旦在用户应用程序中调用了 ODBC API 函数 SQLConnect，此参数就会自动启动 solidDB。如果设置为 no，那么必须通过调用 SSC API 函数 SSCStartServer 来显式启动 solidDB。	yes

附录 C. 无盘服务器的配置参数

本节描述用于实现和维护无盘服务器的参数设置。

在无盘服务器中使用的参数

solid.ini 配置文件的以下部分包含一些参数，这些参数对于无盘服务器具有特定设置。

IndexFile 部分

在 IndexFile 部分中，**FileSpec** 和 **CacheSize** 参数具有无盘服务器的特定设置。

Filespec_[1...n]

FileSpec 参数描述数据库文件的名称和最大大小。要定义主内存引擎的最大大小（按字节计），FileSpec 参数接受下列自变量:

- 数据库文件名 – 由于无盘服务器不会创建物理数据库文件，因此未使用此参数；但是，必须为此自变量提供一个哑元值。
- 最大文件大小 – 此设置是必需的。您需要指定足够大的文件大小（按字节计），以存储无盘服务器中的所有数据。请注意，最大文件大小必须小于高速缓存大小，高速缓存大小使用 **CacheSize** 参数进行设置。

FileSpec 参数的缺省值是 solid.db，2147483647 字节（2GB -1）。例如:

```
FileSpec_1=solid.db 2147483647
```

注: 如果指定了多个文件，那么最大文件大小设置必须是所有 **FileSpec** 参数设置的总和。

最大大小受到可用物理内存的限制；这是因为无盘机器没有磁盘用作虚拟内存的交换空间。

注: 在某些平台上，可用于应用程序的物理内存量可能小于机器中的物理内存量。

例如，在某些版本的 32 位 Linux 系统中，可用于应用程序的内存量只有理论地址空间（4GB）的一半或者四分之一，这是因为 Linux 会将地址的一个或两个最高有效位保留给它自己的内存管理器使用。

如果内存中的数据量超过了最大文件大小，那么会显示错误消息 11003:

```
文件写入失败，超过了配置大小
```

CacheSize

CacheSize 参数定义服务器为缓冲区高速缓存分配的主内存量（按字节计）。例如:

```
CacheSize=10000000
```

此值的设置取决于无盘服务器的下列条件:

- 对基于磁盘的表，高速缓存大小（按字节计）至少应该比使用 FileSpec 参数设置的最大文件大小（即，数据量）大 20%，原因是此数据保存在缓冲区高速缓存中。20% 的开销是一个估计值，可能会根据数据库的使用情况而变化。例如:

```
[IndexFile]
FileSpec_1=solid.db 10MB
CacheSize=12MB
```

- 即使未使用基于磁盘的表（数据库是使用内存表创建的），也需要使用高速缓存来存放系统表。在这种情况下，高速缓存大小最少应为 1 到 2 MB。系统表占用的空间取决于数据库对象的数目和复杂程度以及是否使用了高级复制。
- 高速缓存大小必须小于可用于运行无盘服务器的物理内存量。

  可以按如下所示估计无盘服务器使用的内存总量。（请注意，所有这些内存的总量必须小于可用的物理内存总量，这就意味着高速缓存大小必须远远小于可用于服务器的物理内存总量: ）

  ```
  CacheSize + 5MB
  + (100K * 用户数 * 每个用户具有的活动语句数)
  + 内存表空间
  + (要发送至辅助服务器的 HSB 操作数) [1][2]
  ```

  [1] 此等式中的条件仅适用于 HotStandby 用户。HSB 主服务器需要一些内存用来存储要发送至辅助服务器的 HotStandby 操作。在主服务器与无盘辅助服务器之间临时发生网络故障期间，主服务器可以继续接受来自应用程序的事务。当服务器之间恢复了网络连接之后，就会将主服务器中的更新发送至辅助服务器。（HotStandby 使用事务日志来存储这些操作。无盘服务器无法将事务日志写入磁盘；信息必须存储在内存中。）此内存与高速缓存是分开的。

  [2] 对于此等式中的条件，最大限制目前是 1 MB 或者 512 项操作，以这两者中的较小者为准。与基于磁盘的服务器不同，并不允许事务日志一直增大到用尽可用空间为止。

  所需内存的准确数量还取决于其他因素（例如，对服务器执行的查询的性质）。由于操作系统等将占用一些物理内存，因此可用于服务器的内存量通常小于物理内存总量。

Com 节

如果您将无盘服务器用作高级复制副本服务器，那么 Com 部分中的 **Listen** 参数将影响主服务器和无盘副本服务器之间的通信。

Listen

Listen 参数定义无盘服务器在开始侦听网络时使用的协议和名称。

例如:

```
[Com]
Listen=tcpip 2315
```

Listen 参数的缺省值是 `tcp 1964`。

有关网络名和协议的更多信息，请参阅《*IBM solidDB 管理员指南*》中的『*管理网络连接*』一节。

不适用于无盘引擎的配置参数

对于无盘服务器，下列配置文件参数（按节分组）已被禁用或者不起作用。这些参数将影响不适用于无盘引擎的行为。

表 19. 配置参数不适用于无盘引擎

参数	描述
[General] 节	
CheckpointInterval	此参数已被禁用，这是因为检查点不应用于无盘服务器。
[IndexFile] 节	
ReadAhead	因为不会从数据库文件中物理读取，因此该参数不起作用
PreFlushPercent	因为不会物理写入数据库文件，因此该参数不起作用
[Logging] 节	
LogEnabled	此参数已被禁用，这是因为始终将对无盘服务器禁用事务记录。 **注**：无盘方式仅支持事务回滚。通常在某些故障中断了已部分完成的事务时使用事务回滚。无盘方式不支持进行前滚恢复。

附录 D. solidDB 服务器控制 API（SSC API）

solidDB 服务器控制 API（SSC API）是一组函数，它们提供了简单而有效的方法来控制 solidDB 的任务系统。

注: 有关函数的某些信息也适用于用于 Java 的 SSC API。有关用于 Java 的 SSC API 的更多信息，请参阅 第 79 页的附录 E, 『SolidServerControl 类接口』一节。

使用 SSC API

检索任务信息

要检索所有活动任务的列表，请使用 SSCGetActiveTaskClass 函数。要检索所有暂挂任务的列表，请使用 SSCGetSuspendedTaskClass 函数。要获取任务类的优先级，请使用 SSCGetTaskClassPrio 函数。

特殊事件的通知函数

链接库访问对优先级任务做出了精确调整。可以使用 SSCSetNotifier() 函数来确定一旦发生了特殊事件，solidDB 就会调用用户定义的指定函数。此函数将检测的特殊事件为:

- solidDB 服务器关闭
- 从索引到存储器树的 Bonsai 合并
- Bonsai 合并最大时间间隔
- 备份或检查点请求
- 空闲服务器状态
- 从主服务器中接收到 netcopy 请求，这是一个将主数据库的网络副本发送至辅助服务器的请求。
- netcopy 请求的完成，当使用通过网络复制（netcopy）接收到的新数据库来启动服务器时就会完成 netcopy 请求。

获取 solidDB 状态和服务器信息

可以使用 SSCGetStatusNum 函数来查看 solidDB 数据库服务器的当前状态信息。将显示以下信息:

- 未从 Bonsai 树合并到存储器树的行数

如果 solidDB 服务器正在运行，那么 SSCGetServerHandle 函数将返回此服务器的句柄。

还可以使用 SSCIsRunning 函数来验证服务器是否正在运行，使用 SSCIsThisLocalServer 函数来验证应用程序是已链接至本地链接库访问服务器库（例如，在 Windows 平台上为 `ssolidacxx.dll`）还是“伪”服务器库（例如，在 Windows 平台上为 `solidctrlstub.lib`），这两个库用来测试要使用控制 API 的远程应用程序。

SSC API 和等价的 ADMIN COMMAND

solidDB 服务器控制 API（SSC API）函数具有等价的 solidDB SQL 扩展 ADMIN COMMAND。在远程站点和本地站点中都可以通过 solidDB 工具（例如，solidDB 远程控制（solcon）和 solidDB SQL 编辑器（solsql））来执行这些命令。

有关控制 API 等价 ADMIN COMMAND 的详细信息，请参阅第 47 页的附录 B，『链接库访问参数』。

SSC API 函数总结

以下是 solidDB 服务器控制 API（SSC API）函数以及在『SSC API 函数参考』一节中对此函数进行描述的位置的简要总结。

表 20. 控制 API 函数的总结

函数	描述	支持	要了解更多详细信息，请参阅:
SSCSetCipher	设置由应用程序提供的加密库。**要点**：不推荐。请改为使用 SSCSetDataCipher	LLA 和 SSC API 存根库	请参阅 第 60 页的『SSCSetCipher（不推荐）』。
SSCSetDataCipher	设置由应用程序提供的加密库。	LLA 和 SSC API 存根库	请参阅 第 62 页的『SSCSetDataCipher』。
SSCSetDefaultCipher	设置由应用程序提供的加密库。	LLA 和 SSC API 存根库	请参阅 第 64 页的『SSCSetDefaultCipher』。
SSCStartSmaServer	启动 SMA 服务器。	SMA	请参阅 第 75 页的『SSCStartSMAServer』。
SSCStartDisklessSmaServer	启动无磁盘 SMA 服务器。	SMA	请参阅 第 74 页的『SSCStartDisklessSMAServer』。
SSCStartServer	启动 LLA 服务器。	LLA 和 SSC API 存根库	请参阅 第 71 页的『SSCStartServer』。
SSCStartDisklessServer	启动无磁盘 LLA 服务器。	LLA 和 SSC API 存根库	请参阅 第 70 页的『SSCStartDisklessServer』。
SSCSetState	设置 solidDB 服务器的状态（例如，SSC_STATE_OPEN 指示是否允许建立后续连接）。将状态设置为 ~SSC_STATE_OPEN 将同时阻塞本地连接和远程连接。	LLA 和 SSC API 存根库	请参阅 第 69 页的『SSCSetState』。
SSCRegisterThread	为服务器注册一个链接库访问应用程序线程。需要在用户应用程序的每个线程中注册之后，才能调用任何 LLA API 函数。	LLA 和 SSC API 存根库	请参阅 第 59 页的『SSCRegisterThread』。
SSCUnregisterThread	为服务器注销一个 LLA 应用程序线程。需要在已注册的每个线程中解除注册之后才能终止服务器。	LLA 和 SSC API 存根库	请参阅 第 78 页的『SSCUnregisterThread』。
SSCStopServer	停止 SMA 或 LLA 服务器。	SMA、LLA 和 SSC API 存根库	请参阅 第 77 页的『SSCStopServer』。

表 20. 控制 API 函数的总结 (续)

函数	描述	支持	要了解更多详细信息，请参阅:
SSCSetNotifier	指定用户定义的函数，当发生所指定的事件（例如，合并、备份和关闭等）时，solidDB 就会调用此函数。	LLA 和 SSC API 存根库	请参阅第 66 页的『SSCSetNotifier』。
SSCIsRunning	如果服务器正在运行，那么将返回一个非零值。	LLA 和 SSC API 存根库	请参阅 第 58 页的『SSCIsRunning』。
SSCIsThisLocalServer	指示应用程序是已链接到具有 LLA 的 solidDB服务器还是“伪”库（solidctrlstub），以使用 SSC API 来测试 solidDB远程应用程序。	LLA 和 SSC API 存根库	请参阅 第 59 页的『SSCIsThisLocalServer』。
SSCGetServerHandle	如果服务器正在运行，那么将返回 solidDB 服务器句柄。	LLA 和 SSC API 存根库	请参阅 第 57 页的『SSCGetServerHandle』。
SSCGetStatusNum	获取 solidDB 状态信息。	LLA 和 SSC API 存根库	请参阅 第 58 页的『SSCGetStatusNum』。

SSC API 参考

SSC API 参考按字母顺序描述了每个 SSC API 函数。每个描述都包含用途、摘要、参数、返回值和注释。

- 『函数摘要』
- 『参数』
- 第 56 页的『返回值』
- 第 57 页的『SSC API 错误代码和消息』

函数摘要

此函数的声明摘要为:

```
ReturnType SSC_CALL function(modifier parameter[,...]);
```

ReturnType 会发生变化，但通常是一个指示调用成功或失败的值。在本节的后面部分更详细地描述了返回值。

为了提供可移植性，SSC_CALL 是必需的。SSC_CALL 指定函数的调用约定。在 sscapi.h 文件中为每个平台都适当地定义了调用约定。

参数以斜体表示。

参数

在每个函数描述中，都是按表格式来描述各个参数的。表中包含的是参数的一般用法类型（下面进行了描述）和特定函数中参数变量的用途。

参数用法类型

下表显示了 SSC API 参数可能采用的用法类型。请注意，如果一个参数用作指针，那么它将包含另一个用法类别，以指定参数变量在调用之后的所有权。

表 21. SSC API 参数用法类型

用法类型	含义
in	指示此参数是输入。
output	指示此参数是输出。
in out	指示此参数是输入/输出。
use	仅适用于 pointer 参数。这意味着只有在进行函数调用期间才使用此参数。在进行函数调用之后，调用者可以使用此参数来执行它希望的操作。这是最常见的参数传递类型。
take	仅适用于 pointer 参数。这意味着函数采用了此参数值。在进行函数调用之后，调用者不能引用此参数。当不再需要此参数时，此函数或者在此函数中创建的对象负责释放此参数。

hold	仅适用于 pointer 参数。这意味着即使在进行函数调用之后，函数仍将保留参数值。在进行函数调用之后，调用者可以继续引用参数值并且负责释放参数。
	
	警告:
	
	因为用户和服务器共享此参数，所以在服务器结束使用此参数之前请不要将它释放。通常，对于一个被挂起的对象，可以在释放挂起它的对象之后再释放此对象。例如:
	`conn = SaConnect("", "dba", "dba");`
	`/* Connection is held until cursor is freed */`
	`scur = SaCursorCreate(conn, "mytable");`
	`...`
	`SaCursorFree(scur);`
	`/* After we free the cursor, it is safe to free */`
	`/* the connection (or, as in this case, call a */`
	`/* server function that frees the connection). */`
	`SaDisconnect(conn);`

返回值

每个函数描述都指示此函数是否返回了值以及所返回值的类型。

SscTaskSetT

当函数返回一个类型为 SscTaskSetT 的值时，此定义用作一个位掩码。SScTaskSetT 是在 `sscapi.h` 中定义的，它可以为下列值:

```
SSC_TASK_NONE
SSC_TASK_CHECKPOINT
SSC_TASK_BACKUP
SSC_TASK_MERGE
SSC_TASK_LOCALUSERS
SSC_TASK_REMOTEUSERS
SSC_TASK_SYNC_HISTCLEAN
```

```
SSC_TASK_SYNC_MESSAGE
SSC_TASK_HOTSTANDBY
SSC_TASK_HOTSTANDBY_CATCHUP
SSC_TASK_ALL（上述所有任务）
```

请注意，HotStandby“netcopy”和 HotStandby“copy”操作是由“SSC_TASK_BACKUP”任务执行的；没有单独的“SSC_TASK_NETCOPY”任务。

SSC API 错误代码和消息

SSC API 函数可能会返回下表中列出的错误代码和消息。

这些常量在 `sscapi.h` 文件中定义。

表 22. SSC API 函数的错误代码和消息

错误代码/消息	描述
SSC_SUCCESS	操作成功。
SSC_ERROR	一般错误。
SSC_ABORT	操作异常终止。
SSC_FINISHED	如果已经执行了所有任务，那么 SSCAdvanceTasks 将返回此消息。
SSC_CONT	如果还需要执行其他任务，那么 SSCAdvanceTasks 将返回此消息。
SSC_CONNECTIONS_EXIST	存在开放式连接。
SSC_UNFINISHED_TASKS	存在未完成的任务。
SSC_INFO_SERVER_RUNNING	服务器已经在运行。
SSC_INVALID_HANDLE	给定了无效的本地服务器句柄。此服务器与通过 SSCStartServer 启动的服务器不匹配。
SSC_INVALID_LICENSE	找不到许可证或者找到了无效许可证文件。
SSC_NODATABASEFILE	找不到数据库文件。
SSC_SERVER_NOTRUNNING	服务器未运行。
SSC_SERVER_INNETCOPYMODE	服务器采用 netcopy 方式（仅适用于高可用性/HotStandby）。

SSCGetServerHandle

如果服务器正在运行，那么 SSCGetServerHandle 将返回 solidDB 服务器句柄。

摘要

```
SscServerT SSC_CALL SSCGetServerHandle(void)
```

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

返回值

- 如果服务器未运行，那么将返回 NULL。
- 如果服务器正在运行，那么将返回服务器句柄。

SSCGetStatusNum

SSCGetStatusNum 将获取 solidDB 的状态信息。

摘要

```
SscRetT SSC_CALL SSCGetStatusNum(SscServerT h, SscStatusT stat,
      long * num)
```

SSCGetStatusNum 函数接受下列参数:

表 23. SSCGetStatusNum 参数

参数	用法类型	描述
h	in, use	服务器的句柄。
stat	in	指定用于检索的状态标识:
num	out	如果成功调用了此函数，那么当它返回时，此参数的值将设置为未合并的写入数，或者设置为服务器线程数，这取决于所请求的信息。

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

如果您调用 SSCGetStatusNum 并为 stat 参数传递一个不识别的值，那么此函数将返回 SSC_SUCCESS。

返回值

- SSC_SUCCESS - 操作成功。如果您为 stat 参数传递了一个无效值，那么也会返回此值。
- SSC_ERROR - 操作失败。
- SSC_SERVER_INNETCOPYMODE - 服务器采用 netcopy 方式（仅适用于 HotStandby）。
- SSC_SERVER_NOTRUNNING - 服务器未运行。

SSCIsRunning

如果服务器正在运行，那么 SSCIsRunning 将返回非零值。

摘要

```
int SSC_CALL SSCIsRunning(SscServerT h)
```

SSCIsRunning 函数接受下列参数:

表 24. SSCIsRunning 参数

参数	用法类型	描述
h	in, use	服务器的句柄

返回值

- 0 – 服务器未运行。
- 非零值 – 服务器正在运行。

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

SSCIsThisLocalServer

SSCIsThisLocalServer 指示应用程序是已链接至 solidDB 服务器还是“伪”库（solidctrlstub）。solidctrlstub 库使开发者可以使用控制 API 来测试 solidDB 远程应用程序，而不需要与链接库访问库进行链接和修改源代码。

摘要

```
int SSC_CALL SSCIsThisLocalServer(void)
```

返回值

- 0 – 应用程序并未链接至 solidDB 服务器。
- 1 – 应用程序已链接至 solidDB 服务器。

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

SSCRegisterThread

SSCRegisterThread 用于为服务器注册 solidDB 应用程序线程。每个使用控制 API、ODBC API 或 SA API 的线程都必须注册。必须在线程调用 SSCRegisterThread 函数之后，才能使用任何其他链接库访问 API 函数。

如果应用程序只有一个线程（主线程），也就是说，应用程序本身未创建任何线程，那么不需要注册。

在线程终止之前，它必须通过调用 SSCUnregisterThread 函数将它自己注销。

摘要

```
SscRetT SSC_CALL SSCRegisterThread(SscServerT h)
```

SCCRegisterThread 函数接受下列参数:

表 25. SCCRegisterThread 参数

参数	用法类型	描述
h	In, Use	服务器的句柄

返回值

- SSC_SUCCESS
- SSC_INVALID_HANDLE

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

另请参阅

SSCUnregisterThread

SSCSetCipher（不推荐）

要点: 不推荐使用此函数。请改为使用 第 62 页的『SSCSetDataCipher』。

SSCSetCipher 函数设置由应用程序提供的密码以及加密/解密函数。当 SSCStartServer 中使用了相关的数据库加密命令行参数时，就会自动使用所提供的密码。

摘要

```
void SSC_CALL SSCSetCipher(
    void* cipher,
    char* (SSC_CALL *encrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize),
    int (SSC_CALL *decrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize));
```

表 26. SSCSetCipher 参数

参数	用法类型	描述
cipher	in	指向特定于应用程序的密码对象（例如，加密密码）的指针。solidDB 服务器不使用此指针或相反解释此指针。它只是将它传递给由应用程序提供的加密/解密函数。

表 26. SSCSetCipher 参数 (续)

参数	用法类型	描述
encrypt	in	指向由应用程序提供的加密函数的指针。当服务器必须对数据库文件或日志文件页进行加密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 指向由应用程序提供的密码对象的指针。 • page_no – 由服务器提供的页号。加密算法可以安全地忽略它，或者将它用作加密密钥的一部分。 • n – 要加密的页数。 • pagesize – 要加密的页的大小。 • page – 指向要加密的数据缓冲区的指针。数据缓冲区的大小为: *n*pagesize* 函数必须返回指向要写入文件的已加密数据缓冲区（大小为 *n*pagesize*）的指针。 服务器不会释放指向已加密数据缓冲区的指针。 加密函数可以采用任何方式来覆盖或处理作为“page”参数传递给函数的数据缓冲区。例如，加密函数可以“在原地”对数据进行加密并返回“页”指针。
decrypt	in	指向由应用程序提供的解密函数的指针。当服务器已读取已加密的数据库或日志文件的一部分并且需要对它进行解密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 指向由应用程序提供的密码对象的指针。 • page_no – 由服务器提供的页号。解密算法可以安全地忽略它，或者将它用作解密密钥的一部分。 • n – 要解密的页数。 • pagesize – 要解密的页的大小。 • page – 指向要解密的数据缓冲区的指针。数据缓冲区的大小为: *n*pagesize*

返回值

SSCSetCipher 函数不会返回任何值。应当在使用 SSCStartServer 函数来启动链接库访问服务器之前调用此函数。

注释

如果解密函数已成功地对页进行解密，那么它应当返回一个非零值；如果由于某个原因造成解密失败，那么它应当返回 0。在后面这种情况下，服务器将紧急关闭，因为它无法继续运行。此函数应当将已加密的数据返回在由“page”参数指定的同一缓冲区中。

使用链接库访问加密 API

以下代码说明了 AcceleratorLib 加密 API 的用法。采用的加密方法很普通，也就是常用的 XOR 迷惑。

```
char* SS_CALLBACK encrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;

        for (i=0; i<n; i++) {
   page[i] ^= (i+*key);
        }
        return page;
}


bool SS_CALLBACK decrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;

        for (i=0; i<n; i++) {
   page[i] ^= (i+*key);
        }

        return TRUE;
}
...

int main(int argc, char** argv)
{
    int key = 17;
...
        SSCSetCipher(&key, encrypt, decrypt);
        sscret = SSCStartServer(argsc, args, &h, SSC_STATE_OPEN);
        ....
        SSCStopServer(h, FALSE);
...

}
```

SSCSetDataCipher

SSCSetDataCipher 函数设置了应用程序提供的密码以及加密/解密函数。当 SSCSetDataCipher 中使用了相关的数据库加密命令行参数时，就会自动使用提供的密码。

摘要

```
void SSC_CALL SSCSetDataCipher(
    void* cipher,
    char* (SSC_CALL *encrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize),
    int (SSC_CALL *decrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize));
```

表 27. SSCSetDataCipher 参数

参数	用法类型	描述
cipher	in	指向特定于应用程序的密码对象（例如，加密密码）的指针。solidDB 服务器不使用此指针或相反解释此指针。它只是将它传递给由应用程序提供的加密/解密函数。
encrypt	in	指向由应用程序提供的加密函数的指针。当服务器必须对数据库文件或日志文件页进行加密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 指向由应用程序提供的密码对象的指针。 • page_no – 由服务器提供的页号。加密算法可以安全地忽略它，或者将它用作加密密钥的一部分。 • n – 要加密的页数。 • pagesize – 要加密的页的大小。 • page – 指向要加密的数据缓冲区的指针。数据缓冲区的大小为: `n*pagesize` 函数必须返回指向要写入文件的已加密数据缓冲区（大小为 *n*pagesize*）的指针。 服务器不会释放指向已加密数据缓冲区的指针。 加密函数可以采用任何方式来覆盖或处理作为“page”参数传递给函数的数据缓冲区。例如，加密函数可以“在原地”对数据进行加密并返回“页”指针。
decrypt	in	指向由应用程序提供的解密函数的指针。当服务器已读取已加密的数据库或日志文件的一部分并且需要对它进行解密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 指向由应用程序提供的密码对象的指针。 • page_no – 由服务器提供的页号。解密算法可以安全地忽略它，或者将它用作解密密钥的一部分。 • n – 要解密的页数。 • pagesize – 要解密的页的大小。 • page – 指向要解密的数据缓冲区的指针。数据缓冲区的大小为: `n*pagesize`

返回值

SSCSetDataCipher 函数不会返回任何值。在使用 SSCStartServer 函数启动链接库访问服务器之前，需要调用此函数。

注释

如果解密函数已成功地对页进行解密，那么它应当返回一个非零值；如果由于某个原因造成解密失败，那么它应当返回 0。在后面这种情况下，服务器将紧急关闭，因为它无法继续运行。此函数应当将已加密的数据返回在由“page”参数指定的同一缓冲区中。

使用链接库访问加密 API

以下代码说明了 AcceleratorLib 加密 API 的用法。加密方法为 XOR 迷惑。

```
char* SS_CALLBACK encrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;

        for (i=0; i<n; i++) {
   page[i] ^= (i+*key);
        }
        return page;
}


bool SS_CALLBACK decrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;

        for (i=0; i<n; i++) {
   page[i] ^= (i+*key);
        }

        return TRUE;
}
...

int main(int argc, char** argv)
{
    int key = 17;
...
        SSCSetDataCipher(&key, encrypt, decrypt);
        sscret = SSCStartServer(argsc, args, &h, SSC_STATE_OPEN);
        ....
        SSCStopServer(h, FALSE);
...

}
```

SSCSetDefaultCipher

SSCSetDefaultCipher 函数设置应用程序提供的密码以及加密/解密函数。当 SSCSetDefaultCipher 中使用了相关的数据库加密命令行参数时，就会自动使用所提供的密码。

摘要

```
int SSC_CALL SSCSetDefaultCipher(
            void* cipher,
            void (SSC_CALL *encrypt)(void* cipher, char* data, int datalen),
            void (SSC_CALL *decrypt)(void* cipher, char* data, int datalen));
```

表 28. SSCSetDefaultCipher 参数

参数	用法类型	描述
cipher	in	指向特定于应用程序的密码对象（例如，加密密码）的指针。solidDB 服务器不使用此指针或相反解释此指针。它只是将它传递给由应用程序提供的加密/解密函数。
encrypt	in	指向由应用程序提供的加密函数的指针。当服务器必须对密码、数据库文件或日志文件页面进行加密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 密码指的是应用程序提供的安全上下文（密码对象），如加密密码。此参数也会传递回应用程序提供的加密/解密函数。 • encrypt – 加密函数 • data – 要由应用程序函数加密的数据。此参数用于已加密数据的输入和输出。 • datalen – 正在加密/解密的数据的长度。
decrypt	in	指向由应用程序提供的解密函数的指针。当服务器已读取已加密的密码、数据库或日志文件的一部分并且需要对其进行解密时，就会从服务器中调用此函数。此函数的参数为: • cipher – 密码指的是应用程序提供的安全上下文（密码对象），如加密密码。此参数也会传递回应用程序提供的加密/解密函数。 • decrypt – 解密函数 • data – 要由应用程序函数解密的数据。此参数用于已解密数据的输入和输出。 • datalen – 正在解密的数据的长度。

返回值

SSCSetDefaultCipher 函数不会返回任何值。在使用 SSCStartServer 或 SSCStartSMAServer 函数启动链接库访问服务器之前，需要调用此函数。

注释

如果解密函数已成功地对页进行解密，那么它应当返回一个非零值；如果由于某个原因造成解密失败，那么它应当返回 0。在后面这种情况下，服务器将紧急关闭，因为它无法继续运行。此函数应当将已加密的数据返回在由“page”参数指定的同一缓冲区中。

使用加密 API

以下代码说明了定制加密 API 的用法。加密方法为 XOR 迷惑。

```
char* SS_CALLBACK encrypt(void *cipher, char *data, datalen)

{
        size_t n = np*pagesize;
        size_t i;
```

```
        for (i=0; i<n; i++) {
   data[i] ^= (i+*key);
        }
        return page;
}


bool SS_CALLBACK decrypt(void *cipher, char *data, datalen)
{
        size_t n = np*datalen;
        size_t i;

        for (i=0; i<n; i++) {
   data[i] ^= (i+*key);
        }

        return TRUE;
}
...

int main(int argc, char** argv)
{
    int key = 17;
...
        SSCSetDefaultCipher(&key, encrypt, decrypt);
        sscret = SSCStartServer(argsc, args, &h, SSC_STATE_OPEN);
        ....
        SSCStopServer(h, FALSE);
...

}
```

SSCSetNotifier

SSCSetNotifier 设置链接库访问服务器在启动和停止时将调用的回调函数。此函数没有相应的 ADMIN COMMAND。

摘要

```
SscRetT SSC_CALL SSCSetNotifier(SscServerT h, SscNotFunT what,
    notify_fun handler, void* userdata
)
```

SSCSetNotifier 函数接受下列参数:

表 29. SSCSetNotifier 参数

参数	用法类型	描述
h	in	服务器的句柄。

表 29. *SSCSetNotifier 参数 (续)*

参数	用法类型	描述
what	in	指定要通知的事件。选项为: • SSC_NOTIFY_EMERGENCY_EXIT 在使用 SSCStartServer() 激活了服务器之后，如果此服务器崩溃，就会调用此函数。必须在发出 SSCStartServer() 之前发出通知函数调用 SSCSetNotifier() • SSC_NOTIFY_SHUTDOWN 服务器关闭时调用此函数。 • SSC_NOTIFY_SHUTDOWN_REQUEST 当服务器接收到关闭请求时调用此函数，如果用户定义的函数接受请求，那么服务器可能会关闭。可以通过从被通知的函数返回 SSC_ABORT 或者为此请求返回 SSC_CONT 来拒绝关闭服务器。 • SSC_NOTIFY_ROWSTOMERGE 当 bonsai 索引树中有需要合并到存储服务器的数据时，就要调用此函数。 • SSC_NOTIFY_MERGE_REQUEST 当超过了 `solid.ini` 配置文件中的 MergeInterval 参数设置并且必须启动合并时，就要调用此函数。 • SSC_NOTIFY_BACKUP_REQUEST 当请求了备份时调用此函数。可以通过从被通知的函数返回 SSC_ABORT 来拒绝备份。 • SSC_NOTIFY_CHECKPOINT_REQUEST 当请求了执行检查点操作时调用此函数。可以通过从被通知的函数返回 SSC_ABORT 来拒绝执行检查点操作。 • SSC_NOTIFY_IDLE 当服务器切换到空闲状态时调用此函数。 • SSC_NOTIFY_NETCOPY_REQUEST 此回调函数仅适用于 HotStandby 组件。当从主服务器中接收到 netcopy 请求时调用此函数，netcopy 请求是要求将主数据库的网络副本发送至辅助服务器的请求。有关 netcopy 命令的详细信息，请参阅 *《IBM solidDB 高可用性用户指南》*。 • SSC_NOTIFY_NETCOPY_FINISHED 此回调函数仅适用于 HotStandby 组件。当完成了 netcopy 请求时调用此函数。完成了 netcopy 请求时，将使用通过网络复制（netcopy）接收到的新数据库来启动服务器，并且调用 SSC_NOTIFY_FINISHED 以便让应用程序知道服务器再次可用。
notify_fun_handler	in, hold	要调用的用户函数。
userdata	in, hold	要传递给 notify 函数的用户数据。 请阅读第 55 页的『参数』中有关发布用法类型为 *hold* 的参数的警告。

返回值

- SSC_SUCCESS – 接受了来自服务器的请求。

  仅适用于 HotStandby:

  如果 SSC_NOTIFY_NETCOPY_FINISHED 返回 SSC_SUCCESS，那么所有其他应用程序连接都将终止并且将服务器设置为“netcopy 侦听方式”。在此方式下，服务器将接受来自主服务器的连接，并且辅助服务器唯一可以执行的操作就是接收来自 HotStandby netcopy 命令的数据。有关“netcopy 侦听方式”的更多详细信息，请阅读 *《IBM solidDB 高可用性用户指南》*。（请注意，“netcopy 侦听方式”在以前又称为“备份侦听方式”。）
- SSC_ABORT – 拒绝了来自服务器的请求。

  仅适用于 HotStandby:

  如果 SSC_NOTIFY_NETCOPY_REQUEST 返回了 SSC_ABORT，那么表示 netcopy 未启动并且会将错误代码（SRV_ERR_OPERATIONREFUSED）返回到主服务器中。
- SSC_INNETCOPYMODE – 服务器采用 netcopy 方式（仅适用于 HotStandby）。

  SSC_SERVER_NOTRUNNING – 服务器未运行。

注释

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

释放一个用法类型为 *hold* 的参数时应谨慎。请阅读有关 hold 第 55 页的『参数』的警告。

用户定义的 notifier 函数不应调用任何 SA、SSC 或 ODBC 函数。

创建用户定义的 notifier 函数时，必须符合以下原型:

```
int SSC_CALL mynotifyfun(SscServerT h, SscNotFunT what ,void* userdata);
```

一旦您使用了 SSC_CALL 来显式定义用户函数的约定，请使用 SSCSetNotifier 函数来注册此函数，以便在发生所指定的事件时调用此函数；例如:

```
SscRetT SSCSetNotifier(h, SSC_NOTIFY_IDLE, mynotifyfun, NULL);
```

示例

关闭时调用函数

假定用户将创建 user_own_shutdownrequest 函数，每次请求关闭时就会调用此函数:

```
int SSC_CALL user_own_shutdownrequest(SscServerT h, SscNotFunT what, void
    *userdata);
{
  if (shutdown not needed) {
    return SSC_ABORT;
  }
  return SSC_CONT; /*Proceed with shutdown*/
}
```

然后，可以按如下所示调用 SSCSetNotifier 函数，以指定在服务器关闭之前调用 user_own_shutdownrequest。

```
SSCSetNotifier(handle, SSC_NOTIFY_SHUTDOWN, user_own_shutdownrequest, NULL);
```

注:

如果 user_own_shutdownrequest 函数返回 SSC_ABORT，那么不允许关闭服务器；如果该函数返回 SSC_CONT，那么可以关闭服务器。

SSCSetState

SSCSetState 设置链接库访问服务器的状态。这使您可以控制服务器是否接受后续连接以及服务器是否使用预取。

如果服务器设置为“打开”，那么服务器将接受连接。如果服务器设置为“关闭”，那么它将不接受任何后续连接（无论是本地连接还是远程连接）；但是，允许已经建立的所有连接都继续运行。

打开预取就会告知服务器要进行“预读”，以访存可能马上就要引用的数据。进行预取将需要更多内存或磁盘高速缓存空间。打开预取时，性能通常会更高。关闭预取时，需要的内存更少。如果您的查询需要对服务器进行大量顺序扫描，那么打开预取就非常有用。例如，如果您使用报告或聚集函数来获取整个数据库（或者它的很大一部分）的值，那么预取可能对您有帮助。如果您的所有查询只涉及到一条或多条记录，那么预取通常就没太大用处。由于预取将耗用内存，因此在可用内存很少的系统中，预取实际上可能会降低性能。

下列准则可以帮助您决定何时使用预取。

如果您有大量的可用内存（或者磁盘高速缓存空间），并且您的查询需要执行大量顺序扫描，那么就可以使用预取。

如果您的可用内存很少，并且您的查询通常一次只读取一条不相关的记录，请不要使用预取。

摘要

```
SscRetT SSC_CALL SSCSetState(SscServerT h,SscStateT runflags)
```

SSCSetState 函数接受下列参数:

表 30. SSCSetState 参数

参数	用法类型	描述
h	in, use	服务器的句柄。
runflags	in	选项可以由 SSC_STATE_OPEN 标志（它意味着允许建立新的远程连接）和 SSC_STATE_PREFETCH 标志（它意味着用户允许服务器在必要时执行预取）组合而成。下面是可能存在的组合的一个示例: • 服务器设置为“打开”: state = state | SSC_STATE_OPEN; • 服务器设置为“关闭”: state = state & ~SSC_STATE_OPEN; • 打开预取: state = state | SSC_STATE_PREFETCH; • 关闭预取: state = state & ~SSC_STATE_PREFETCH;

返回值

- SSC_SUCCESS – 操作成功。
- SSC_ERROR – 操作失败。
- SSC_SERVER_INNETCOPYMODE – 服务器采用 netcopy 方式（仅适用于 HotStandby）。
- SSC_SERVER_NOTRUNNING – 服务器未运行。

注释

此函数具有相应的 solidDB SQL 扩展 ADMIN COMMAND。此命令为:

ADMIN COMMAND 'close';

SSCStartDisklessServer

SSCStartDisklessServer 将启动一个使用链接库访问的无盘服务器。

摘要

```
SscRetT SSC_CALL SSCStartDisklessServer (int argc, char* argv[ ],
    SscServerT * h, SscStateT runflags, char* lic_string, char* ini_string);
```

SSCStartDisklessServer 函数接受下列参数:

表 31. SSCStartDisklessServer 参数

参数	用法类型	描述
argc	in	命令行参数的数目。
argv	in, use	在执行函数调用期间使用的命令行参数组成的数组。参数 argv[0] 是仅为用户应用程序的路径和文件名保留的，它必须存在。 有关可用参数的列表，请参阅《*IBM solidDB 管理员指南*》中的『*solidDB 命令行选项*』一节。
h	out	将返回已启动的服务器的句柄。当通过其他控制 API 函数来引用服务器时需要此句柄。
runflags	in	此参数的唯一选项是: SSC_STATE_OPEN – 允许建立远程连接。 runflags = SSC_STATE_OPEN
lic_string	in	指定包含 solidDB 许可证文件的字符串。
ini_string	in	指定包含 solidDB 配置文件的字符串。

返回值

- SSC_SUCCESS – 服务器已启动。
- SSC_ERROR – 服务器未能启动。
- SSC_SERVER_INNETCOPYMODE – 服务器是 netcopy 方式（仅适用于 HotStandby）。
- SSC_INFO_SERVER_RUNNING – 服务器已在运行。
- SSC_INVALID_HANDLE – 提供了无效的本地服务器句柄。
- SSC_INVALID_LICENSE – 未找到许可证或找到无效的许可证文件。

注释

缺省情况下，状态设置为 SSC_STATE_OPEN。

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

示例

SSCStartDisklessServer

```
SscStateT runflags = SSC_STATE_OPEN;
SscServerT h;
char* argv[4]; /* pointers to four parameter strings */
int argc = 4;
char* lic = get_lic(); /* get the license */
char* ini = get_ini(); /* get the solid.ini */
SscRetT rc;
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-Udba"; /* user name */
argv[2] = "-Pdba"; /* user's password */
argv[3] = "-Cdba"; /* catalog name */
/* Start the diskless server */
rc = SSCStartDisklessServer(argc, argv, &h, runflags, lic, ini);
```

注:

在此示例中，get_ini() 和 get_lic() 是用户必须编写的函数。每个函数必须返回一个包含 `solid.ini` 文件文本或者 `solid.lic` 许可证文件的字符串。

如果您未指定目录名，那么 solidDB 将返回错误。

另请参阅

SSCStopServer

另请参阅第 41 页的6 章，『使用无盘功能』。

SSCStartServer

SSCStartServer 将启动链接库访问。在多线程环境中，服务器与客户机运行于不同线程中。在应用程序的持续时间内，应用程序可以根据需要来启动或停止服务器子例程。

请注意，第三个参数是一个“out”参数。如果已成功启动服务器，那么 SSCStartServer 例程会将此参数设置为指向此服务器的句柄。

注:

如果您要启动无盘服务器，那么必须使用控制 API 函数 SSCStartDisklessServer 来启动此服务器。请阅读 第 70 页的『SSCStartDisklessServer』。

摘要

```
SscRetT SSC_CALL SSCStartServer(int argc, char* argv[], SscServerT* h
    SscStateT runflags)
```

SSCStartServer 函数接受下列参数:

表 32. SSCStartServer 参数

参数	用法类型	描述
argc	in	命令行参数的数目。
argv	in, use	命令行参数的数组。 有关可用参数的列表，请参阅《*IBM solidDB 管理员指南*》中的『*solidDB 命令行选项*』一节。
h	out	将返回已启动的服务器的句柄。当通过其他控制 API 函数来引用服务器时需要此句柄。
runflags	in	选项可以是下列其中一项或者两项: • SSC_STATE_OPEN – 允许建立远程连接。 • SSC_STATE_PREFETCH – 服务器将在必要时执行预取。 例如: `runflags = SSC_STATE_OPEN & SSC_STATE_PREFETCH`

示例: 启动 SSCStartServer

使用服务器名称、目录名以及管理员的用户名和密码来启动 SSCStartServer:

```
SscStateT runflags = SSC_STATE_OPEN; SscServerT h; char* argv[5];
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-nsolid1"; argv[2] = "-UDBA" argv[3] = "-PDBA";
argv[4] = "-CDBA"; /* Start the server */ rc =
SSCStartServer(argc, argv, &h, run_flags);
```

注: 如果您已经有一个现存的数据库，那么不需要指定用户名和密码或者目录名。

返回值

- SSC_SUCCESS – 服务器已启动。
- SSC_ERROR – 服务器未能启动。
- SSC_ABORT
- SSC_BROKENNETCOPY – 由于未完成 netcopy 而导致数据库被破坏。
- SSC_FINISHED
- SSC_CONT

- SSC_CONNECTIONS_EXIST
- SSC_UNFINISHED_TASKS
- SSC_INVALID_HANDLE – 提供了无效的本地服务器句柄。
- SSC_INVALID_LICENSE – 未找到许可证或找到无效的许可证文件。
- SSC_NODATABASEFILE – 找不到数据库文件。
- SSC_SERVER_NOTRUNNING
- SSC_INFO_SERVER_RUNNING – 服务器已在运行。
- SSC_SERVER_INNETCOPYMODE – 服务器采用 netcopy 方式（仅适用于 HotStandby）。
- SSC_DBOPENFAIL – 打开数据库失败。
- SSC_DBCONNFAIL – 连接至数据库失败。
- SSC_DBTESTFAIL – 测试数据库失败。
- SSC_DBFIXFAIL – 修复数据库失败。
- SSC_MUSTCONVERT – 必须转换数据库。
- SSC_DBEXIST – 数据库已存在。
- SSC_DBNOTCREATED – 未创建数据库。
- SSC_DBCREATEFAIL – 创建数据库失败。
- SSC_COMINITFAIL – 通信初始化失败。
- SSC_COMLISTENFAIL – 通信侦听失败。
- SSC_SERVICEFAIL – 服务操作失败。
- SSC_ILLARGUMENT – 命令行参数非法。
- SSC_CHDIRFAIL – 更改目录失败。
- SSC_INFILEOPENFAIL – 打开输入文件失败。
- SSC_OUTFILEOPENFAIL – 打开输出文件失败。
- SSC_SRVCONNFAIL – 服务器连接失败。
- SSC_INITERROR – 操作初始化失败。
- SSC_CORRUPTED_DBFILE – 断言或其他致命错误。
- SSC_CORRUPTED_LOGFILE – 断言或其他致命错误。

注释

缺省情况下，状态设置为 SSC_STATE_OPEN。

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

当您启动新的 solidDB 服务器时，必须显式指定 solidDB 会通过将 SSCStartServer() 函数与 -U *username* -P *password* -C *catalogname*（缺省数据库目录名）参数配合使用来创建新的数据库。有关详细信息，请阅读第 31 页的『使用 SSC API 函数 SSCStartServer 显式启动』。

如果您要重新启动数据库服务器（即，一个数据库已经存在于目录中），那么 SSCStartServer 将使用现有数据库。

SSCStartServer 函数可能会衍生多个线程来运行服务器任务。服务器任务包括处理本地和远程客户机请求以及运行各种后台任务（例如，执行检查点操作和合并等等）。

另请参阅

SSCStopServer

SSCStartDisklessSMAServer

SSCStartDisklessSMAServer 使用 SMA 启动无盘服务器。

摘要

```
SscRetT SSC_CALL SSCStartDisklessSMAServer (int argc, char* argv[ ],
    SscServerT * h, SscStateT runflags, char* lic_string, char* ini_string);
```

SSCStartDisklessSMAServer 函数接受以下参数:

表 33. SSCStartDisklessSMAServer 参数

参数	用法类型	描述
argc	in	命令行参数的数目。
argv	in, use	在执行函数调用期间使用的命令行参数组成的数组。参数 argv[0] 是仅为用户应用程序的路径和文件名保留的，它必须存在。 有关可用参数的列表，请参阅《*IBM solidDB 管理员指南*》中的『*solidDB 命令行选项*』一节。
h	out	将返回已启动的服务器的句柄。当通过其他控制 API 函数来引用服务器时需要此句柄。
runflags	in	此参数的唯一选项是: SSC_STATE_OPEN – 允许建立远程连接。 runflags = SSC_STATE_OPEN
lic_string	in	指定包含 solidDB 许可证文件的字符串。
ini_string	in	指定包含 solidDB 配置文件的字符串。

返回值

- SSC_SUCCESS – 服务器已启动。
- SSC_ERROR – 服务器未能启动。
- SSC_SERVER_INNETCOPYMODE – 服务器是 netcopy 方式（仅适用于 HotStandby）。
- SSC_INFO_SERVER_RUNNING – 服务器已在运行。
- SSC_INVALID_HANDLE – 提供了无效的本地服务器句柄。
- SSC_INVALID_LICENSE – 未找到许可证或找到无效的许可证文件。

注释

缺省情况下，状态设置为 SSC_STATE_OPEN。

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

示例

```
TBA
```

另请参阅

SSCStopServer

SSCStartSMAServer

SSCStartSMAServer 使用 SMA 启动服务器。

摘要

```
SscRetT SSC_CALL SSCStartSMAServer (int argc, char* argv[ ],
    SscServerT * h, SscStateT runflags, char* lic_string, char* ini_string);
```

SSCStartSMAServer 函数接受以下参数:

表 34. SSCStartSMAServer 参数

参数	用法类型	描述
argc	in	命令行参数的数目。
argv	in, use	在执行函数调用期间使用的命令行参数组成的数组。参数 argv[0] 是仅为用户应用程序的路径和文件名保留的，它必须存在。 有关可用参数的列表，请参阅《*IBM solidDB 管理员指南*》中的『*solidDB 命令行选项*』一节。
h	out	将返回已启动的服务器的句柄。当通过其他控制 API 函数来引用服务器时需要此句柄。
runflags	in	此参数的唯一选项是: SSC_STATE_OPEN – 允许建立远程连接。 runflags = SSC_STATE_OPEN

返回值

- SSC_SUCCESS – 服务器已启动。
- SSC_ERROR – 服务器未能启动。
- SSC_ABORT
- SSC_BROKENNETCOPY – 由于未完成 netcopy 而导致数据库被破坏。
- SSC_FINISHED
- SSC_CONT

- SSC_CONNECTIONS_EXIST
- SSC_UNFINISHED_TASKS
- SSC_INVALID_HANDLE – 提供了无效的本地服务器句柄。
- SSC_INVALID_LICENSE – 未找到许可证或找到无效的许可证文件。
- SSC_NODATABASEFILE – 找不到数据库文件。
- SSC_SERVER_NOTRUNNING
- SSC_INFO_SERVER_RUNNING – 服务器已在运行。
- SSC_SERVER_INNETCOPYMODE – 服务器采用 netcopy 方式（仅适用于 HotStandby）。
- SSC_DBOPENFAIL – 打开数据库失败。
- SSC_DBCONNFAIL – 连接至数据库失败。
- SSC_DBTESTFAIL – 测试数据库失败。
- SSC_DBFIXFAIL – 修复数据库失败。
- SSC_MUSTCONVERT – 必须转换数据库。
- SSC_DBEXIST – 数据库已存在。
- SSC_DBNOTCREATED – 未创建数据库。
- SSC_DBCREATEFAIL – 创建数据库失败。
- SSC_COMINITFAIL – 通信初始化失败。
- SSC_COMLISTENFAIL – 通信侦听失败。
- SSC_SERVICEFAIL – 服务操作失败。
- SSC_ILLARGUMENT – 命令行参数非法。
- SSC_CHDIRFAIL – 更改目录失败。
- SSC_INFILEOPENFAIL – 打开输入文件失败。
- SSC_OUTFILEOPENFAIL – 打开输出文件失败。
- SSC_SRVCONNFAIL – 服务器连接失败。
- SSC_INITERROR – 操作初始化失败。
- SSC_CORRUPTED_DBFILE – 断言或其他致命错误。
- SSC_CORRUPTED_LOGFILE – 断言或其他致命错误。

注释

缺省情况下，状态设置为 SSC_STATE_OPEN。

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

启动新的 solidDB服务器时，您必须显式指定 solidDB 使用函数 SSCStartSMAServer() 以及 -U *用户名* -P *密码* -C *目录名*（缺省数据库目录名）参数来创建新的数据库。有关详细信息，请阅读第 31 页的『使用 SSC API 函数 SSCStartServer 显式启动』。

如果您要重新启动数据库服务器（目录中已有数据库），那么 SSCStartSMAServer 将使用现有数据库。

SSCStartSMAServer 函数可能会衍生多个线程来运行服务器任务。服务器任务包括处理本地和远程客户机请求以及运行各种后台任务（例如，执行检查点和合并等等）。

示例

TBA

另请参阅

SSCStopSMAServer

SSCStopServer

SSCStopServer 将停止链接库访问服务器。

请注意，可以使用显式方法（例如，SSCStopServer）来关闭先前使用隐式方法（例如，SQLConnect）启动的服务器。反向操作则不行；例如，不能使用 SQLDisconnect 来停止先前使用 SSCStartServer 启动的服务器。

每当应用程序运行时，并不限制应用程序只能启动和停止服务器一次。在服务器已停止之后，应用程序可以使用 SSCStartServer 来重新启动此服务器。

摘要

```
SscRetT SSC_CALL SSCStopServer(SscServerT h, bool force)
```

SSCStopServer 函数接受下列参数:

表 35. SSCStopServer 参数

参数	用法类型	描述
h	in, use	服务器的句柄
force	in	选项为: • TRUE – 在所有情况下都停止服务器。 • FALSE – 如果没有开放式连接就停止服务器。否则，就无法停止服务器。

返回值

- SSC_SUCCESS – 服务器已停止。
- SSC_CONNECTIONS_EXIT – 存在开放式连接。
- SSC_UNFINISHED_TASKS – 正在执行的任务。
- SSC_ABORT
- SSC_ERROR

注释

远程用户可以使用 ADMIN COMMAND 'shutdown' 来停止 solidDB。有关详细信息，请参阅第 47 页的附录 B, 『链接库访问参数』。

如果与数据库或现有用户建立了开放式连接，那么 FALSE 选项将不允许关闭。此选项等价于 solidDB SQL 扩展 ADMIN COMMAND 'shutdown'。

带有 &~SSC_STATE_OPEN 选项的 SSCSetState() 将阻止与 solidDB 建立新连接。

另请参阅

SSCStartServer

SSCSetState

SSCUnregisterThread

SSCUnregisterThread 将注销服务器的 solidDB 应用程序线程。必须由已注册的每个线程自己通过 SSCRegisterThread 函数来调用 SSCUnregisterThread 函数。将在线程终止之前调用此函数。

摘要

```
SscRetT SSC_CALL SSCUnregisterThread(SscServerT h)
```

SCCUnregisterThread 函数接受下列参数:

表 36. SCCUnregisterThread 参数

参数	用法类型	描述
h	in, use	服务器的句柄

返回值

- SSC_SUCCESS
- SSC_INVALID_HANDLE

注释

SSC_CALL 需要显式定义用户函数的调用约定。它是在每个平台的相应 `sscapi.h` 文件中定义的。

此函数没有相应的 solidDB SQL 扩展 ADMIN COMMAND。

另请参阅

SSCRegisterThread

附录 E. SolidServerControl 类接口

下面描述了 SolidServerControl 类的完整公共接口。

有关参数和相应的 ADMIN COMMAND 的详细信息包括在 第 55 页的『SSC API 参考』一节中的 SSC 函数描述中。

有关在此类中使用一些方法的 LLA 程序的示例，请参阅 samples/aclib_java 目录中用于 Java 的 LLA 样本。

返回值常量

```
        public final static int SSC_SUCCESS = 0;
        public final static int SSC_ERROR = 1;
        public final static int SSC_ABORT = 2;
        public final static int SSC_FINISHED = 3;
        public final static int SSC_CONT = 4;
        public final static int SSC_CONNECTIONS_EXIST = 5;
        public final static int SSC_UNFINISHED_TASKS = 6;
        public final static int SSC_INVALID_HANDLE = 7;
        public final static int SSC_INVALID_LICENSE = 8;
        public final static int SSC_NODATABASEFILE = 9;
        public final static int SSC_SERVER_NOTRUNNING = 10;
        public final static int SSC_INFO_SERVER_RUNNING = 11;
        public final static int SSC_SERVER_INNETCOPYMODE = 12;
```

```
/**
 * Initiates a SolidServerControl class. Output is not directed to any
 * PrintStream.
 *
 * @return        SolidServerControl instance
 */
public static  SolidServerControl instance()
     throws SolidServerInitializationError;


/**
 * Initiates a SolidServerControl class. Output is being directed
 * to a PrintStream object given in parameter 'os'.
 *
 * @param os      the PrintStream for output
 * @return        SolidServerControl instance
 *
 */
public static SolidServerControl instance( PrintStream os )
     throws SolidServerInitializationError;

/**
 * setOutStream method sets the output to the given PrintStream
 *
 * @param os      the PrintStream for output
 */
public void setOutStream( PrintStream os );


/**
 * getOutStream returns the stream used for output in class
 * SolidServerControl
 *
 * @return        returns the outputstream of this object
 */
```

```
public PrintStream getOutStream();

/**
* startDisklessServer starts the solidDB Linked Library Access server
* in a diskless mode
*
* @param argv       parameter vector for the accelerator server.
*
* @param runflags   Options for this parameter are SSC_STATE_OPEN (remote
*                   connections are allowed) and SSC_STATE_PREFETCH (server
*                   will do a "prefetch" if needed). Prefetch refers to the
*                   memory cache that provides read-ahead
*                   capability for table content. Following is a runflags
*                   parameter entry:
*
* @param lic_file   The contents of the solidDB license file as a string,
*                   since the diskless version cannot read the
*                   information from the disk
*
* @param ini_file   The contents of the solid.ini configuration file as a string,
*                   since the diskless version cannot read the information
*                   from the disk
*
* @return   the return value from the server :
*              SSC_SUCCESS
*              SSC_ERROR
*    SSC_INVALID_LICENSE - No license or license file found.
*              SSC_NODATABASEFILE  - No database file found.
*/
public static long startDisklessServer( String[] argv, long runflags,
String lic_string, String ini_string )


/**
 * startServer starts the solidDB Linked Library Access server
 *
 * @param argv       parameter vector for the LLA server
 *                   Note! You must give the working directory containing
 *                   solidDB license file (f.ex. -c\tmp) first, in front
 *                   of other parameters.
 *
 * @param runflags   Options for this parameter are SSC_STATE_OPEN
 *                   (remote connections are allowed) and
 *                   SSC_STATE_PREFETCH (server will do a "prefetch"
 *                   if needed). Prefetch refers to the memory
 *                   and/or disk cache that provides read-ahead
 *                   capability for table content. Following is
 *                   a runflags parameter entry:
 *
 *                    runflags |= SSC_STATE_OPEN & SSC_STATE_PREFETCH
 *
 * @return           the return value from the server:
 *              SSC_SUCCESS
 *              SSC_ERROR
 *              SSC_INVALID_LICENSE - No license or invalid license file found.
 *              SSC_NODATABASEFILE  - No database file found.
 */
public long startServer( String[] argv, long runflags );


/**
* stopServer stops the LLA server
 *
 * @param runflags   Runflags for stopping LLA server.
 * See section SSCStopServer for more
 *                   details.
 *
```

```
 * @return           the return value from the server
 *              SSC_SUCCESS if server is stopped.
 *              SSC_CONNECTIONS_EXIT if there are open connections.
 *              SSC_UNFINISHED_TASKS if there are still tasks that are
 *                                   executing.
 *              SSC_SERVER_NOTRUNNING if the server is not running.
 */
public long stopServer( int runflags );


/**
 * returns the state of the server, i.e. is the server running or not
 *
 * @return SSC_STATE_OPEN if server is up and running
 */
public int getState();




/**
* registerThread registers this user thread to solidDB Linked Library Access server
*
*
* @return           the return value from the server
*                    SSC_SUCCESS            Registration succeeded.
*                    SSC_ERROR              Registration failed.
*                    SSC_INVALID_HANDLE     Invalid local server handle given.
*                    SSC_SERVER_NOTRUNNING  Server is not running.
*/
public long registerThread();



/**
* unregisterThread unregisters this user thread from the
* solidDB Linked Library Access server
*
*
* @return           the return value from the server
*                    SSC_SUCCESS            Registration succeeded.
*                    SSC_ERROR              Registration failed.
*                    SSC_INVALID_HANDLE     Invalid local server handle given.
*                    SSC_SERVER_NOTRUNNING  Server is not running.
*/
public long unregisterThread();
```

索引

声明

本产品受美国专利 6144941、7136912、6970876、7139775、6978396、7266702、7406489 和 7502796 的保护。

为此产品指定的美国出口管制分类编号是 ECCN=5D992b。

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。有关您当前所在区域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，则由用户自行负责。

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（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:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区：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、适销和适用于某种特定用途的保证。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。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/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）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

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。

本资料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 国际软件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。

本信息包含在日常业务操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。所有这些名字都是虚构的，若现实生活中实际业务企业使用的名字和地址与此相似，纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。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。这些示例并未在所有条件下作全面测试。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。

凡这些实例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:

©（贵公司的名称）（年）。此部分代码是根据 IBM Corp. 公司的样本程序衍生出来的。

商标

IBM、IBM 徽标、ibm.com®、Solid、solidDB、InfoSphere™、DB2®、Informix® 和 WebSphere® 是 International Business Machines Corporation 在美国和/或其他国家或地区的商标或注册商标。如果这些商标和其他 IBM 注册商标在本资料中第一次出现时标有商标符号（® 或 ™），那么这些符号表示它们是发布本资料时归 IBM 所有的经过美国政府注册的商标或普通法商标。这些商标也可能是在其他国家或地区的注册商标或普通法商标。在 Web 上的版权和商标信息（www.ibm.com/legal/copytrade.shtml）处提供了 IBM 商标的最新列表。

Java 和所有基于 Java 的商标和徽标是 Sun Microsystems, Inc. 在美国和/或其他国家或地区的商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的注册商标。

Microsoft 和 Windows 是 Microsoft Corporation 在美国和/或其他国家或地区的注册商标。

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

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

Printed in China

S151-1292-00