IBM solidDB
IBM solidDB Universal Cache

バージョン 6.5

共有メモリーおよびリンク・ライブラリー・アクセス・ユーザー・ガイド

Information Management software

SC88-8166-00
(英文原典：SC23-9876-00)

IBM solidDB
IBM solidDB Universal Cache

バージョン 6.5

共有メモリーおよびリンク・ライブラリー・アクセス・ユーザー・ガイド

Information Management software

SC88-8166-00
(英文原典：SC23-9876-00)

注記

本書および本書で紹介する製品をご使用になる前に、 101 ページの『特記事項』に記載されている情報をお読みください。

本書は、バージョン 6 リリース 5 の IBM solidDB (製品番号 5724-V17) および IBM solidDB Universal Cache (製品番号 5724-W91)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC23-9876-00
IBM solidDB Universal Cache
Version 6.5
Shared Memory Access and Linked Library Access User Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2009.10

目次

図

表

本書について

IBM® solidDB® 共有メモリー・アクセス (SMA) およびリンク・ライブラリー・アクセス (LLA) を使用すると、アプリケーションから solidDB サーバーに直接リンクすることができるようになるため、TCP/IP などのネットワーク・プロトコルを介して通信する必要がなくなります。SMA では複数のアプリケーションのリンクが可能であり、LLA では 1 つのアプリケーションのリンクが可能です。ネットワーク接続をローカル関数呼び出しで置き換えることにより、パフォーマンスが大幅に向上します。

本書には、SMA および LLA に固有の情報が記載されています。本書は、solidDB の管理と保守について詳細に説明している「*IBM solidDB 管理者ガイド*」に含まれている情報を補足するものです。

本書は、C プログラミング言語に関する実用的な知識、一般的な DBMS の知識、SQL に精通していること、および solidDB インメモリー・データベースや solidDB ディスク・ベース・エンジンなどの solidDB データ管理製品に関する知識を前提としています。SMA または LLA を Java™ とともに使用する場合には、Java に関する実用的な知識も前提としています。

書体の規則

solidDB の資料では、以下の書体の規則を使用します。

表 1. 書体の規則

フォーマット	用途
データベース表	このフォントは、すべての通常テキストに使用します。
NOT NULL	このフォントの大文字は、SQL キーワードおよびマクロ名を示しています。
`solid.ini`	これらのフォントは、ファイル名とパス式を表しています。
`SET SYNC MASTER YES;` `COMMIT WORK;`	このフォントは、プログラム・コードとプログラム出力に使用します。SQL ステートメントの例にも、このフォントを使用します。
run.sh	このフォントは、サンプル・コマンド行に使用します。
TRIG_COUNT()	このフォントは、関数名に使用します。
java.sql.Connection	このフォントは、インターフェース名に使用します。
LockHashSize	このフォントは、パラメーター名、関数引数、および Windows® レジストリー項目に使用します。

表 1. 書体の規則 (続き)

フォーマット	用途
argument	このように強調されたワードは、ユーザーまたはアプリケーションが指定すべき情報を示しています。
管理者ガイド	このスタイルは、他の資料、または同じ資料内の他の章の参照に使用します。新しい用語や強調事項もこのように記述します。
ファイル・パス表示	特に明記していない場合、ファイル・パスは UNIX® フォーマットで示します。スラッシュ (/) 文字は、インストール・ルート・ディレクトリーを表します。
オペレーティング・システム	資料にオペレーティング・システムによる違いがある場合は、最初に UNIX フォーマットで記載します。UNIX フォーマットに続いて、小括弧内に Microsoft® Windows フォーマットで記載します。その他のオペレーティング・システムについては、別途記載します。異なるオペレーティング・システムに対して、別の章を設ける場合があります。

構文表記法の規則

solidDB の資料では、以下の構文表記法の規則を使用します。

表 2. 構文表記法の規則

フォーマット	用途
`INSERT INTO` *`table_name`*	構文の記述には、このフォントを使用します。置き換え可能セクションには、この フォントを使用します。
`solid.ini`	このフォントは、ファイル名とパス式を表しています。
[ ]	大括弧は、オプション項目を示します。太字テキストの場合には、大括弧は構文に組み込む必要があります。
|	垂直バーは、構文行で、互いに排他的な選択項目を分離します。
{ }	中括弧は、構文行で互いに排他的な選択項目を区切ります。太字テキストの場合には、中括弧は構文に組み込む必要があります。
...	省略符号は、引数が複数回繰り返し可能なことを示します。
. . .	3 つのドットの列は、直前のコード行が継続することを示します。

1 共有メモリー・アクセスおよびリンク・ライブラリー・アクセスの概要

solidDB *共有メモリー・アクセス* (SMA) および *リンク・ライブラリー・アクセス* (LLA) を使用すると、アプリケーションから solidDB サーバーに直接リンクすることができるようになるため、パフォーマンスを消費する TCP/IP などのネットワーク・プロトコルを介して通信する必要がなくなります。SMA では複数のアプリケーションのリンクが可能であり、LLA では 1 つのアプリケーションのリンクが可能です。

SMA および LLA は、solidDB サーバーの完全なコピーをライブラリー形式で含むライブラリー・ファイルとして実装されます。

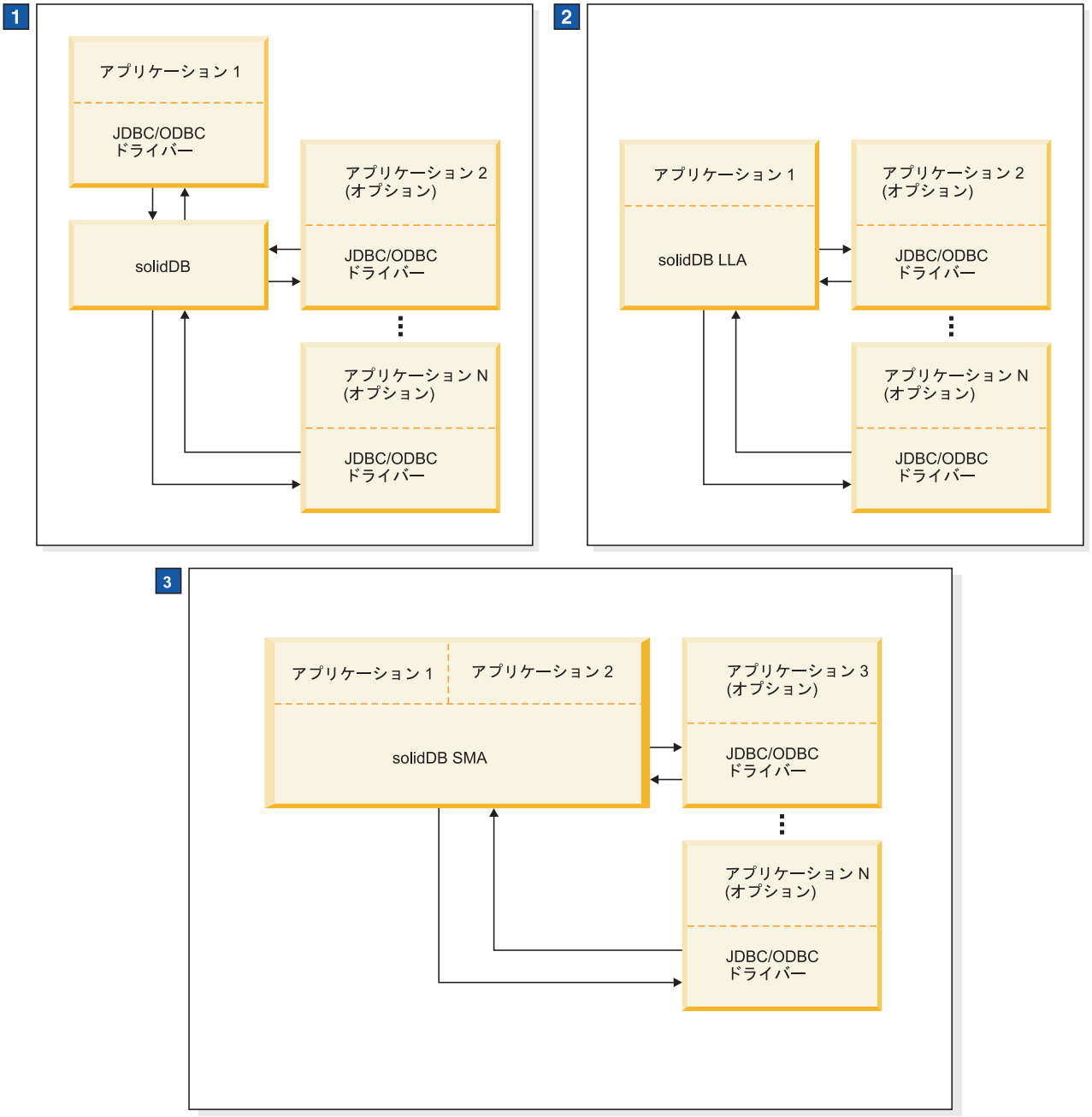
- SMA の場合、アプリケーションがリンクするライブラリーは、ドライバーとして使用されます。リンクするアプリケーションを開始する前に、solidDB サーバーを SMA モードで始動します。このサーバーは、SMA ドライバーを動的にロードし、アプリケーションがデータベースへのアクセスに使用する共有メモリー・セグメントの割り振りと初期化を行います。
- LLA の場合、アプリケーションは LLA ライブラリーにリンクし、アプリケーションとサーバーが単一の実行可能モジュールとしてビルドされます。

SMA または LLA を使用するためにアプリケーションを作成し直す必要はありません。アプリケーションは、ODBC または JDBC 呼び出し、あるいは、solidDB プロプラエタリー SA API を使用して、solidDB サーバーと通信します。

SMA および LLA サーバーは、TCP/IP などの通信プロトコルを介してサーバーに接続するリモート・アプリケーションからの要求も処理できます。リモート・アプリケーションから見た SMA または LLA サーバーは、他の solidDB サーバーとほぼ同じですが、ローカルの SMA および LLA アプリケーションから見ると、他の solidDB サーバーよりも高速で、詳細な制御が可能です。

また、ネットワーク・ベースのサーバーと同様、複数の SMA および LLA サーバーを同じノード上で実行することができます。

SMA および LLA で使用される solidDB サーバーは、ディスク・ベースまたはディスクレスです。インメモリー表およびディスク・ベース表もサポートされています。



1. 標準 solidDB データベース構成では、アプリケーションとサーバーは別々のプログラムです。
2. LLA 構成では、LLA はアプリケーションにリンクされるライブラリーです。その他のアプリケーションも LLA サーバーと通信できます。
3. SMA 構成では、SMA は複数のアプリケーションがリンクできるドライバー・ライブラリーです。その他のネットワーク接続ベースのアプリケーションも SMA サーバーと通信できます。

図 1. SMA、LLA、およびネットワーク接続ベースの solidDB サーバーでの構成

共有メモリー・アクセス (SMA)

共有メモリー・アクセス (SMA) を使用すると、複数のアプリケーションから、データベース・サーバーの全機能を備える動的ドライバー・ライブラリーにリンクできます。つまり、アプリケーションの ODBC または JDBC 要求が、アプリケーション・プロセス・スペースでほぼ完全に処理されるため、プロセス間のコンテキストの切り替えが不要になります。共通データベースの処理を簡単にするために、サーバーが初期化する共有メモリー・セグメントにドライバーからアクセスできます。

SMA ドライバーを使用して solidDB サーバーにリンクするアプリケーションは、*SMA アプリケーション* と呼ばれ、そのサーバーは *SMA サーバー* と呼ばれます。

SMA サーバー

最初のアプリケーションが SMA を使用して開始される前に、SMA ドライバー・ライブラリーを動的にロードする小さいスタブ・アプリケーション (`solidsma.exe` または `solidsma.bin`) を開始することで、solidDB サーバーが SMA モードで初期化されます。SMA サーバーのスタブ・アプリケーションは、サーバー・コードを内部で開始し、アプリケーションがデータベースへのアクセスに使用する共有メモリー・セグメントの割り振りと初期化を行います。

SMA サーバーは、以下に示すネットワーク・サーバーの全機能を備えます。

- SMA サーバー・プロセスは、始動とリカバリー、チェックポイントの指定とロギング、バックアップの作成などの、クライアントに依存しないすべてのタスクを処理します。
- solidDB 構成パラメーター、admin command、およびコマンド行パラメーターを使用できます。
- インメモリー表とディスク・ベース表の両方に同じようにアクセスできます。
- solidDB Universal Cache で、高可用性、拡張レプリケーション、CDC レプリケーションを構成した状態で SMA サーバーを使用できます。
-
- SMA サーバーは、通常のネットワーク接続ベースのサーバーとしても使用できます。

サーバーを SMA モードで始動すると、通常の listen ポートで SMA ドライバーからの接続要求を受け入れます。それぞれの SMA サーバーに異なるポート番号を割り当てることで、単一システムで複数の SMA サーバーを同時に稼働することが可能です。

サーバーをシャットダウンするか、すべてのユーザーが操作を終了した場合、アプリケーションは次の要求時に「`Connection lost`」エラーを受け取ります。強制シャットダウン時にアプリケーションが応答を待機していた場合、シャットダウン通知を受け取ります。

ディスク・ベース・サーバーおよびディスクレス・サーバー

SMA サーバーは、ディスク・ベースまたはディスクレスです。ディスクレス・サーバーは、ネットワーク・ルーターやスイッチ内のライン・カードなどの、ハード・ディスクを持たない組み込みシステムで役立ちます。

SMA および solidDB のツール

solidDB データ管理ツールは、SMA サーバーへのネットワーク・ベース接続で使用できます。

SMA アプリケーション

SMA を使用するために、データ・ソース名または接続ストリングを除いては、既存の ODBC または JDBC アプリケーションを変更する必要はありません。例えば、ODBC アプリケーションでは、接続は通常の ODBC SQLConnect() 呼び出しで要求されます。

既存の LLA アプリケーションを SMA アプリケーションに、あるいはその逆の変更が可能です。また、アプリケーションは、SMA アプリケーションからネットワーク・ベースのアプリケーションに変更することもできます。

SMA ドライバー

SMA ドライバーは、solidDB サーバーの完全なコピーをライブラリー形式で含む動的ライブラリーです。アプリケーションは、SMA ドライバーに直接リンクするか、ドライバー・マネージャーを使用することができます。

ドライバーのバイナリー・ファイルのフットプリントは、solidDB サーバーに完全に対応しており、サイズはプラットフォームに応じて 3 から 6 MB です。ただし、すべてのアプリケーションが同じドライバーにリンクするため、アプリケーションが追加されても、メモリー内のフットプリントは増加しません。アプリケーション・システム全体 (アプリケーション、ドライバー、およびサーバー) の総メモリー・フットプリントは、1 つのクライアント/サーバー・モデルと同等です。

SMA 接続

SMA サーバーを稼働すると、アプリケーションは SMA 接続またはネットワーク接続を確立できます。SMA 接続の場合、アプリケーションはサーバーと同じノードに配置する必要があります。接続タイプは、接続ストリング内で定義されます。ODBC アプリケーションでドライバー・マネージャーを使用する場合、SMA データ・ソースは、ODBC ドライバーと同じように構成できます。

接続要求はローカルで使用可能なプロトコル (TCPIP、名前付きパイプ、UNIX パイプ) を使用して、ネットワーク接続 (ハンドシェーク接続) 経由で送信されます。ハンドシェーク接続時に、共有メモリー・セグメント・ハンドルがドライバーに渡されるため、ドライバーはサーバーの共有メモリー・セグメントにアクセスできます。

SMA のシステム要件

SMA は、64 ビット・プラットフォームと同様、32 ビット Linux® でも使用できます。Java で SMA を使用する場合、Java ランタイム環境 (JRE) 1.4.2 または Java Development Kit (JDK) 1.4.2 以降が必要です。

SMA にサポートされるプラットフォーム

- AIX®
- HP-UX
- Linux 64 ビット
- Linux 32 ビット
- Solaris
- Windows 64 ビット

サポートされるプラットフォームについて詳しくは、solidDB Web サイト (http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html) にある solidDB システム要件を参照してください。

SMA コンポーネントおよびパッケージ化

SMA ドライバー・ライブラリーおよび SMA サーバー・アプリケーションは、solidDB ソフトウェア・パッケージに含まれています。Java を使用する SMA の場合、solidDB JDBC ドライバーが必要です。

SMA ドライバー (ライブラリー)

最も一般的なプラットフォームの SMA ドライバー・ライブラリーを以下の表に示します。

表 3. SMA ドライバー (ライブラリー)

プラットフォーム	SMA ドライバー・ライブラリー	デフォルトのインストール場所
Windows	ssolidsmaxx.dll **注:** SMA ドライバーに直接リンクする場合、実際の .dllライブラリー・ファイルへのアクセスを提供する solidsma.lib インポート・ライブラリー・ファイルにリンクします。	ライブラリー: <solidDB installation directory>¥bin インポート・ライブラリー: <solidDB installation directory>¥lib
Linux	ssolidsmaxx.so	<solidDB installation directory>/bin
Solaris	ssolidsmaxx.so	<solidDB installation directory>/bin
HP-UX	ssolidsmaxx.sl	<solidDB installation directory>/bin
AIX	ssolidsmaxx.so	<solidDB installation directory>/bin
xx は、ドライバー・ライブラリーのバージョン番号であり、例えば、ssolidsma65.so のようになります。		

全プラットフォームに対応する SMA ドライバー・ライブラリーには、以下が含まれています。

- solidDB サーバーの全機能
- 3 つの API 用の関数
  - ネットワークを介さずにサーバー・ライブラリーと直接通信できるようにする solidDB ODBC ドライバー関数

– SMA サーバーの始動と停止を制御する関数で構成される solidDB 制御 API (SSC API) ライブラリー
– リンク・ライブラリー・アクセスを使用するその他の機能に必要となる solidDB SA API ライブラリー。例えば、このライブラリーを使用して、表におけるレコードの挿入、削除、選択を実行できます。

アプリケーションがリンクするライブラリーは 3 つの API を含むため、アプリケーション・プログラムで、これらの 3 つの API を自由に組み合わせて関数を呼び出すことができます。それぞれの API の詳細については、 10 ページの『SMA および LLA の solidDB API およびドライバー』を参照してください。

Java で SMA を使用する場合は、solidDB JDBC ドライバーが必要です。solidDB JDBC ドライバー (`SolidDriver2.0.jar`) は、solidDB サーバーのインストール中に、solidDB インストール・ディレクトリーの `jdbc` ディレクトリーにインストールされます。

SMA サーバー・アプリケーション

表 4. SMA サーバー・アプリケーション

プラットフォーム	SMA アプリケーション
Windows	`solidsma.exe`
Linux	`solidsma`
Solaris	`solidsma`
HP-UX	`solidsma`
AIX	`solidsma`

リンク・ライブラリー・アクセス (LLA)

リンク・ライブラリー・アクセス (LLA) を使用する場合、アプリケーションは、完全なデータベース・サーバーの機能を備えた静的ライブラリーまたは動的ライブラリーにリンクします。つまり、solidDB がアプリケーションと同じ実行可能プログラムで実行されるので、ネットワークを介してデータを転送する必要がありません。

LLA ライブラリーを使用して solidDB サーバーにリンクするアプリケーションは、*LLA アプリケーション* と呼ばれ、そのサーバーは *LLA サーバー* と呼ばれます。

LLA ライブラリーにリンクしているアプリケーションは、ODBC API、SA API、および JDBC API を使用して、複数の接続を作成することも可能です。これらの API はすべて再入可能であるため、別々のスレッドからの同時接続が可能となります。

LLA ライブラリーに直接リンクしているアプリケーションでは、他のデータベース・サーバーとのリモート接続を作成することもできます。接続タイプ (ローカルまたはリモート) は、ODBC API または SA API の接続関数に渡される接続ストリングで定義されるか、あるいは、JDBC 接続プロパティーで定義されます。

操作の原理

アプリケーションを開始すると、アプリケーション内のコードのみが自動的に実行を開始します。サーバー・コードは大部分がアプリケーション・コードとは独立しており、関数を呼び出してサーバーを明示的に始動する必要があります。多くの実装では、サーバーはアプリケーションが使用するスレッドとは別のスレッドで稼働します。関数を呼び出してサーバーを始動すると、サーバー・コードに必要な初期化ステップが実行され、必要に応じて適切なスレッドが追加で作成され、そのスレッドでサーバーの稼働が開始されます。

ディスク・ベース・サーバーおよびディスクレス・サーバー

LLA で使用される solidDB サーバーは、ディスク・ベースまたはディスクレスです。LLA ライブラリーには、サーバーを始動する関数呼び出しが 2 種類あります。SSCStartServer 関数呼び出しでは通常のディスク・ベース・サーバーを始動し、SSCStartDisklessServer ではディスク・ドライブを使用しないサーバーを始動します。

LLA のシステム要件

LLA は、solidDB がサポートするすべてのプラットフォームで使用可能です。 Java で LLA を使用する場合、Java ランタイム環境 (JRE) 1.4.2 または Java Development Kit (JDK) 1.4.2 以降が必要です。

サポートされるプラットフォームのリストについては、solidDB の Web サイト (http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html) にある solidDB のシステム要件を参照してください。

LLA コンポーネントおよびパッケージ化

LLA ライブラリーは、solidDB ソフトウェア・パッケージに含まれています。Java を使用する LLA の場合、JDBC ドライバーおよび solidDB プロプラエタリー制御クラスが、solidDB JDBC ドライバーに組み込まれています。

最も一般的なプラットフォームの LLA ライブラリーを以下の表に示します。

表 5. LLA ライブラリー

プラットフォーム	静的 LLA ライブラリー	動的 LLA ライブラリー	デフォルトのロケーション
Windows	`solidimpac.lib` これはインポート・ライブラリー・ファイルで、実際のライブラリー・ファイルである `ssolidacxx.dll` へのアクセスを提供します。	`ssolidacxx.dll`	インポート・ライブラリー: `<solidDB installation directory>¥lib` ライブラリー: `<solidDB installation directory>¥bin`
Linux	`solidac.a`	`ssolidacxx.so`	`<solidDB installation directory>/bin`
Solaris	`solidac.a`	`ssolidacxx.so`	`<solidDB installation directory>/bin`

表 5. *LLA* ライブラリー *(続き)*

プラットフォーム	静的 LLA ライブラリー	動的 LLA ライブラリー	デフォルトのロケーション
HP-UX	solidac.a	ssolidacxx.sl	<solidDB installation directory>/bin
xx は、動的ライブラリーのバージョン番号であり、例えば、ssolidac65.so のようになります。			

全プラットフォームに対応する LLA ライブラリーには、以下が含まれています。

- solidDB サーバーの全機能
- 3 つの個別 API 用の関数
  - タスクのスケジューリングを制御する関数で構成される solidDB 制御 API (SSC API) ライブラリー
  - ネットワークを介さずにサーバー・ライブラリーと直接通信できるようにする solidDB ODBC ドライバー関数
  - リンク・ライブラリー・アクセスを使用するその他の機能に必要となる solidDB SA API ライブラリー。例えば、このライブラリーを使用して、表におけるレコードの挿入、削除、選択を実行できます。

  アプリケーションがリンクするライブラリーは、この 3 つの API (SSC、SA、および ODBC) すべてを含むため、アプリケーション・プログラムでこれらの API を任意に組み合わせて関数を呼び出すことができます。それぞれの API の詳細については、10 ページの『SMA および LLA の solidDB API およびドライバー』を参照してください。

  注: リモート・アプリケーションもこの 3 つの API (SSC、SA、および ODBC) にアクセスできます。ただし、リモート・アプリケーションの場合は、この 3 つの API の関数すべてを 1 つにまとめたファイルがありません。リモート・アプリケーションおよび二重の役割を持つアプリケーションについて詳しくは、12 ページの『ローカルおよびリモート・アプリケーション・タイプの構成』を参照してください。リモート・アプリケーション用の API ファイルについては、10 ページの『SMA および LLA の solidDB API およびドライバー』を参照してください。

Java で LLA を使用する場合、solidDB JDBC ドライバーが必要です。solidDB JDBC ドライバーの jar ファイル (SolidDriver2.0.jar) には、以下のパッケージが含まれています。

- solid.jdbc.* solidDB JDBC ドライバー・クラス
- solid.ssc.* solidDB サーバー制御クラス (プロプラエタリー SSC API for Java インターフェース)

LLA のスタティック・リンク・ライブラリーとダイナミック・リンク・ライブラリー

solidDB では、リンク・ライブラリー・アクセス・ライブラリーの*静的* バージョンと*動的* バージョンの両方が用意されています。

静的ライブラリー・ファイルと動的ライブラリー・ファイルの両方には、solidDB サーバーの完全なコピーがライブラリー形式で含まれています。静的ライブラリー・ファイル (lib/solidac.a) など) を使用する場合は、プログラムをそのファイルに直接リンクします。その結果、プログラム・コードとライブラリー・コードの両方が実行可能ファイルに書き込まれます。動的ライブラリー・ファイルにリンクする場合は、実行可能プログラムを含む出力ファイルにライブラリーのコードが書き込まれることはありません。代わりに、このコードは、プログラムの実行時にダイナミック・リンク・ライブラリーから別にロードされます。

静的ライブラリー・ファイルにリンクした場合と動的ライブラリー・ファイルにリンクした場合とでは、実行可能プログラムのサイズが変わる以外にほとんど違いはありません。メモリーに 1 回に読み込まれる全体的なコード量はほぼ同じです (コンピューター上でクライアントとサーバーを 1 つずつ実行している場合)。パフォーマンスもほぼ同じですが、動的ライブラリーを使用する場合は、わずかに余分なオーバーヘッドが生じます。

ダイナミック・リンク・ライブラリー・ファイルを使用する主な利点は、同じコンピューター内でサーバーの複数のコピーを実行する場合にメモリーを節約できることです。例えば、1 台のコンピューターで開発作業を行うときに、拡張レプリケーションのマスターとレプリカの両方を同時にそのコンピューターに配置する場合や、HotStandby の 1 次サーバーと 2 次サーバーを同時に配置する場合は、動的ライブラリーを使用することで、LLA の複数のコピーを同時にメモリーに格納する必要がなくなります。

Microsoft Windows では、solidDB リンク・ライブラリー・アクセスに lib/solidimpac.lib というファイルが追加されます。また、Microsoft Windows でダイナミック・リンク・ライブラリーを使用する場合は、ssolidacxx.dll ダイナミック・リンク・ライブラリー自体に直接リンクするのではなく、インポート・ライブラリーである solidimpac.lib にリンクします。これにより、少量のコードのみがクライアント実行可能プログラムにリンクされます。クライアント・プログラムが実際に実行されるときに、Microsoft Windows オペレーティング・システムによって ssolidacxx.dll ファイルが自動的にロードされ、クライアントがその .dll ファイルにある通常のリンク・ライブラリー・アクセス関数を呼び出せるようになります。.dll ファイルを参照するプログラムを実行するときには、その .dll ファイルをロード・パスに組み込む必要があります。

注: ダイナミック・リンク・ライブラリー・ファイルを使用しても、単一の solidDB サーバーに複数の LLA アプリケーション・クライアントをリンクできるわけではありません。動的ライブラリーを使用する場合でも、ローカル・クライアントの数は 1 つに制限されます。その他のクライアントはすべてリモート・クライアントでなければなりません。つまり、ローカル・クライアントのように関数を直接呼び出す方法ではなく、TCP などのネットワーク・プロトコルを使用して solidDB サーバーと通信します。複数のローカル・アプリケーションが solidDB にアクセスできるようにするには、共有メモリー・アクセス (SMA) を使用して環境を設計します。

SMA および LLA の solidDB API およびドライバー

SMA および LLA アプリケーション要求は、通常、ODBC API の直接関数呼び出し、あるいは JDBC 呼び出しによって処理されます。solidDB プロプラエタリー solidDB API も使用できます。 solidDB サーバー制御 API (SSC API および SSC API for Java) が LLA ライブラリーに含まれており、これを使用して、solidDB バックグラウンド・プロセスとクライアント・タスクを制御するためのローカル要求を処理することができます。 SMA の SSC API に対するサポートには制限があり、SMA サーバーの始動と停止の呼び出ししか含まれていません。

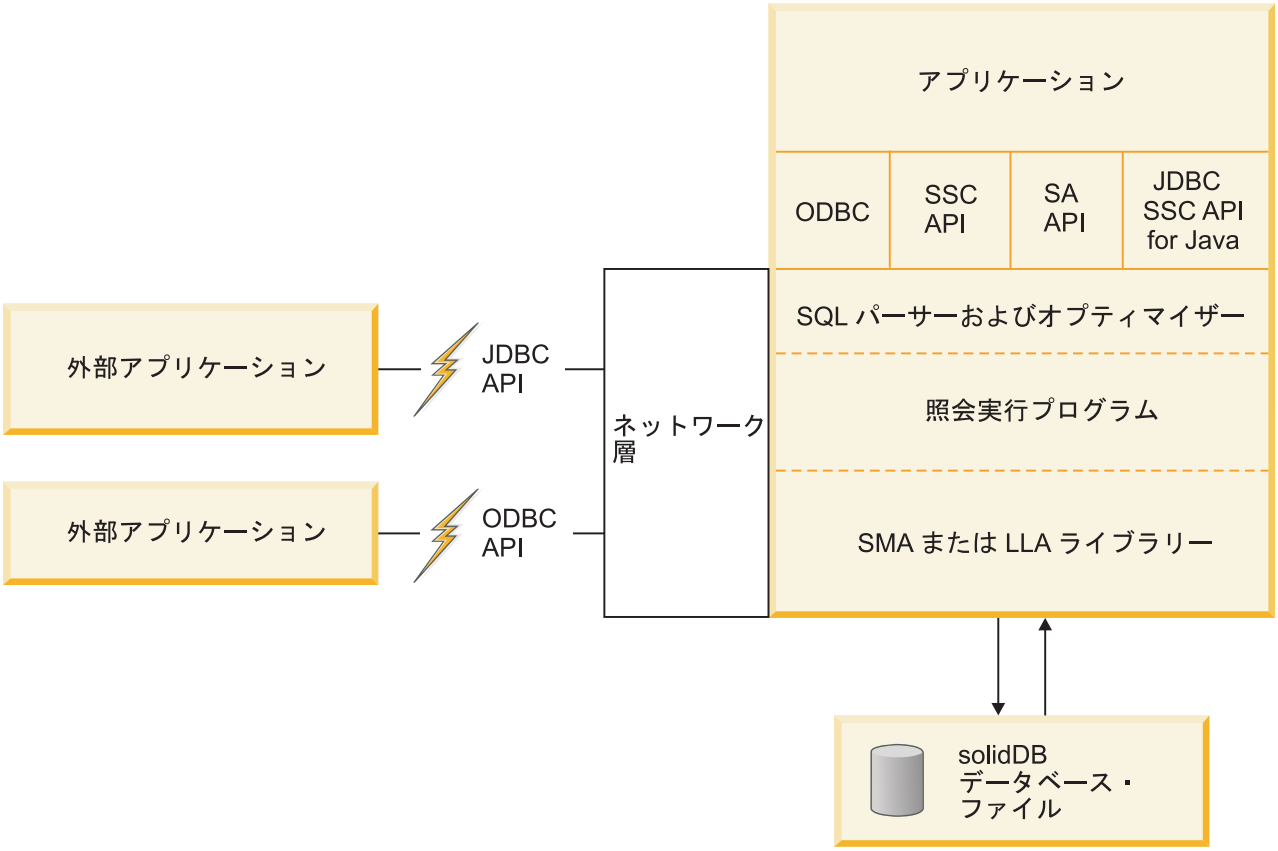


図 2. 例: C/C++ プログラムの SMA および LLA API

solidDB SA API

solidDB SA API は、solidDB データ管理サービスに対する低レベルのプロプラエタリー C 言語 API です。 SA API は、SA API 関数呼び出しを使用するローカル・アプリケーションをサポートします。

SA API ライブラリーは solidDB 製品で内部的に使用され、 solidDB データベース表のデータにアクセスできるようにします。このライブラリーに含まれる約 90 個の関数によって、データベースの接続およびカーソル・ベースの操作を実行するた

めの低レベルのメカニズムが構成されます。 solidDB SA API を使用することで、パフォーマンスは大幅に向上します。例えば、SA API を使用してバッチ挿入操作のパフォーマンスを最適化できます。

solidDB SA API の詳細については、「*IBM solidDB プログラマー・ガイド*」を参照してください。

リモート・アプリケーションでも SA API 関数呼び出しを使用できます。ただし、別の SA API ライブラリー・ファイル (例えば Windows での `solidimpsa.lib`) にリンクする必要があります。

solidDB ODBC API

solidDB ODBC API は、ローカルまたはリモートの solidDB データベースに SQL を使用してアクセスするための標準に準拠した手段です。この API が提供する関数では、データベース接続の制御、SQL ステートメントの実行、結果セットの取得、トランザクションのコミット、およびその他のデータ管理機能を実行できます。

solidDB ODBC API は、solidDB データベース向けのコール・レベル・インターフェース (CLI) です。これは、ANSI X3H2 SQL CLI に準拠しています。

SMA および LLA は ODBC 3.51 標準をサポートします。 SMA および LLA ライブラリーには、solidDB ODBC 3.x が組み込まれており、これによってサーバーへの直接関数呼び出しを必要とするローカル・アプリケーションがサポートされます。

solidDB ODBC API の詳細については、「*IBM solidDB プログラマー・ガイド*」を参照してください。

solidDB JDBC API

JDBC API は、データベース接続、SQL ステートメント、結果セット、データベース・メタデータなどを表す Java クラスを定義します。これを使用すると、SQL ステートメントを実行して、その結果を処理できます。JDBC は、Java でのデータベース・アクセスに使用される基本 API です。

SMA および LLA は、JDBC 1.x と 2.x の両方をサポートします。

solidDB 固有の機能拡張の説明を含む、JDBC インターフェースおよび solidDB JDBC ドライバーについては、「*IBM solidDB プログラマー・ガイド*」に記載されています。

solidDB サーバー制御 API (SSC API)

solidDB サーバー制御 API (SSC API) は、solidDB サーバーの動作を制御する C 言語のスレッド・セーフなインターフェースです。

SSC API 関数は、SMA ドライバーおよび LLA ライブラリー・ファイルに含まれています。ただし、SMA の場合、サーバーの始動および停止用の関数のみがサポートされます。

LLA は、SSC API 関数呼び出しを使用するローカル・アプリケーションをサポートし、リモート専用のアプリケーションには別のライブラリーを使用できます。

リモートで実行する LLA アプリケーションに SSC API 関数呼び出しが含まれている場合、SSC API スタブ・ライブラリー (例えば、Windows の場合、solidctrlstub.lib) にリンクする必要があります。このライブラリーは、実際には、使用するリモート・アプリケーションからのサーバー制御を可能にするものではなく、単に、LLA を使用する solidDB からリンク時エラーを出さずに、アプリケーションをリモート・アプリケーションとしてコンパイルおよびリンクできるようにするものです。

solidDB Server Control API (SSC API) for Java

solidDB Server Control API (SSC API) for Java はプロプラエタリー APIで、SolidServerControl クラスにちなんで命名されました。 SSC API for Java による呼び出しは、LLA サーバーを始動および停止するために使用されます。実際のデータベース接続は、標準の solidDB JDBC API を使用して行われます。SSC API for Java クラスおよび solidDB JDBC ドライバー・クラスは、いずれも solidDB JDBC ドライバー (SolidDriver2.0.jar) 内にあります。

solidDB サーバーにアクセスするための SolidServerControl クラスは、solid.ssc パッケージ内の solidDB JDBC ドライバー・ファイルの内部に組み込まれています。solidDB JDBC ドライバーの jar ファイル (SolidDriver2.0.jar) には、以下のパッケージが含まれています。

- solid.jdbc.* solidDB JDBC ドライバー・クラス
- solid.ssc.* solidDB サーバー制御クラス (プロプラエタリー API インターフェース)

solidDB サーバー制御 (solid.ssc) パッケージには以下のクラスが含まれています。

- SolidServerControl (Java から LLA サーバーを始動および停止する場合に使用)
- SolidServerControlInitializationError (エラーを報告する場合に使用)

詳しくは、 93 ページの『付録 E. SolidServerControl クラス・インターフェース』を参照してください。

ローカルおよびリモート・アプリケーション・タイプの構成

SMA および LLA を使用する場合、アプリケーションは常にローカル solidDB サーバー (SMA サーバーまたは LLA サーバー) に接続します。したがって、アプリケーションと solidDB サーバーは同じノードに配置されます。このローカル SMA または LLA アプリケーションからの要求の処理に加えて、SMA または LLA サーバーは、TCP/IP などの通信プロトコルを介してサーバーに接続するリモート・アプリケーションからの要求も処理できます。二重モード・アプリケーションを作成できるため、コンパイル方法とリンク方法に応じて、ローカル・モードとリモート・モードを切り替えることもできます。

SMA または LLA アプリケーションは、*ローカル・アプリケーション* です。したがって、サーバーとアプリケーションは同じノードに配置されます。例えば、ODBC 関数の呼び出しは、ODBC ドライバーと通信プロトコル (TCP/IP など) を経由せずにサーバーに直接送信されます。

リモート・アプリケーション は、SMA ドライバーまたは LLA ライブラリーにリンクされていません。このようなアプリケーションは、独立した実行可能プログラムであり、ネットワーク接続 (TCP/IP など) またはその他の接続を使用してサーバーと通信する必要があります。リモート・アプリケーションは、通常はサーバーが稼働するノードとは別のノードで実行されますが、ネットワーク通信プロトコルを使用してサーバーと通信する場合にも、アプリケーションはリモートであると見なされます。単一ノードで、SMA または LLA のローカル・アプリケーションを実行しながら、1 つ以上のリモート・アプリケーションを別のプロセスとして実行することができます。

リモート・アプリケーションから見た SMA および LLA サーバーは、他の solidDB サーバーとほぼ同じですが、ローカル・アプリケーションから見ると、他の solidDB サーバーよりも高速で、詳細な制御が可能です。

大多数のアプリケーションは、ローカル (SMA ドライバーまたは LLA ライブラリーにリンク) またはリモート (リンクなし) のいずれかです。ただし、ローカルとネットワーク・ベースの両方の接続を使用する二重モード・アプリケーション を作成することもできます。例えば、このアプリケーションでは、同じ C 言語アプリケーション・コードをローカル・モードまたはリモート・モードで使用できますが、それぞれのモードで異なるライブラリーにリンクされます。

二重モード・アプリケーションは、以下の場合などに便利です。

- 最初にローカル・アプリケーションをテストしてから、SMA または LLA ライブラリーにリンクする場合
- すべてのユーザー/プロセスで、ローカルまたはリモートに関係なく、同じアプリケーション・ロジックを使用する場合

リモート・アプリケーションには、C プログラムと Java プログラムを混在させることができます。ローカル・クライアントを作成した言語によって、リモート・クライアントの作成に使用できる言語が制限されることはありません。例えば、Java で LLA を使用する場合、リモート・クライアント・プログラムに C、Java、またはその両方を使用できます。

リモート・アプリケーションの SSC API および SA API ライブラリー

SSC API または SA API 関数呼び出しを含むリモート・アプリケーションは、個別ライブラリーにリンクする必要があります。

表 6. リモート・アプリケーションの SSC API および SA API ライブラリー

プラットフォーム	SSC API スタブ・ライブラリー	SA API ライブラリー	デフォルトのロケーション
Windows	solidctrlstub.lib	solidimpsa.lib	<solidDB installation directory>¥lib
その他のプラットフォーム	solidctrlstub.a	solidimpsa.a	<solidDB installation directory>/bin

リモート・アプリケーションには、SSC API スタブ・ライブラリーが必要です。これは、SMA および LLA ライブラリーに含まれる SSC API 関数が、リモート・ア

プリケーションで使用できないためです。例えば、標準の ODBC ライブラリーにローカル・アプリケーション (SSC API 関数を使用) がリンクする場合を考えます。このアプリケーションをリモートでも実行するとします。SSC API スタブ・ライブラリーにリンクすることで、コードから SSC API 関数呼び出しを削除する必要がなくなります。このように、LLA または SMA アプリケーションを、通常のリモート・クライアント・アプリケーションに簡単に変更できます。

注: SSC API スタブ・ライブラリーには、「何もしない」関数が含まれています。この関数をリモート・アプリケーションで呼び出しても、サーバーには影響しません。つまり、SSC API スタブ・ライブラリーは、実際には、使用するリモート・アプリケーションからのサーバー制御を可能にするものではなく、単に、LLA または SMA を使用する solidDB からリンク時エラーを出さずに、リモート・アプリケーションとしてアプリケーションをコンパイルおよびリンクできるようにするものです。

2 SMA アプリケーションの作成と実行

SMA アプリケーションを作成するには、必要に応じて solidDB を構成し、アプリケーションを SMA ドライバーにリンクし、SMA サーバーを始動して、アプリケーションとサーバー間のローカル接続を確立します。アプリケーションの作成が完了したら、solidDB が提供するモニター・フィーチャーを使用して、SMA のパフォーマンスをモニターできます。

重要: SMA アプリケーションの作成と実行に関する SMA 固有の追加事項、補足事項、および SMA を使用しない場合の solidDB と比較した使用法の違いを説明します。

solidDB SQL、solidDB データ管理ツール、solidDB の一般的な管理と保守、およびデータベース・エラー・コードについては、「*IBM solidDB 管理者ガイド*」を参照してください。

API および solidDB JDBC および ODBC ドライバーの詳細については、「*IBM solidDB プログラマー・ガイド*」を参照してください。

SMA アプリケーションの作成 - 概要

SMA を使用するアプリケーションを作成するには、SMA を使用するようにシステムを設定し、solidDB を構成して、SMA ドライバーを使用するようにアプリケーションを設定し、SMA サーバーを始動して、アプリケーションをこのサーバーに接続する必要があります。

このタスクについて

この手順では、SMA アプリケーションの作成方法の概要を説明します。C/ODBC 環境向けの SMA アプリケーションは、SMA を使用しないアプリケーションと同様の方法で作成します。

注: アプリケーションを開発する際には、ネットワーク・ベースの接続を使用することをお勧めします。アプリケーションの準備が整ったら、SMA 接続を使用するように切り替えます。

C で作成された SMA アプリケーションの例については、solidDB のインストール・ディレクトリー内の `samples/sma` ディレクトリーにある SMA サンプルを参照してください。

手順

1. **ご使用の環境での、共有メモリー使用量のシステム設定を確認します。**

   ご使用の環境での共有メモリー使用量のデフォルト値では、SMA を使用するのに不十分な場合があります。システムの共有メモリー・システム・パラメーターの表示および設定について詳しくは、16 ページの『共有メモリー・カーネル・パラメーターの変更 - 概要』を参照してください。

2. **作業ディレクトリー、solidDB データベース、およびユーザー・アカウントを作成して、データベース環境をセットアップします。**

   詳しくは、「*IBM solidDB 管理者ガイド*」の『データベースの新規作成』を参照してください。

3. **環境、パフォーマンス、および操作のニーズに合わせて、solidDB を構成します。**

   `solid.ini` 構成ファイルを使用して、データベース・ファイル名や場所、データベース・ブロック・サイズなどの基本構成設定を定義します。

   - 通常のセットアップでは、`solid.ini` ファイルの `[SharedMemoryAccess]` セクション内の SMA 固有パラメーターを変更する必要はありません。大部分のユース・ケースには、ファクトリー値を適用できます。
   - パラメーター **Srv.ProcessMemoryLimit** は、SMA サーバーでは無効です。

   構成ファイルが存在しなければ、ファクトリー値が使用されます。

4. **SMA 用のアプリケーションを準備します。**

   アプリケーションで SMA を使用する場合のドライバー・マネージャーの使用の有無をセットアップできます。

   - 23 ページの『ドライバー・マネージャーを使用する SMA 用のアプリケーションの準備』
   - 24 ページの『ドライバー・マネージャーを使用しない SMA 用のアプリケーションの準備』

5. **SMA サーバーを始動します。**

   詳しくは、26 ページの『SMA サーバーの始動』を参照してください。

6. **アプリケーションを開始します。**

共有メモリー・カーネル・パラメーターの変更 - 概要

共有メモリーは、セグメント単位で割り振られます。共有メモリー・システム・パラメーターは、システムに許可されるセグメントの最大サイズと最大数を制御します。

通常、solidDB は 8 MB のサイズを使用します。

共有メモリー・パラメーターおよびそれらの管理メカニズムは、システムによって異なります。 Linux および UNIX 環境では、以下に説明されているカーネル・パラメーターのタイプを処理する必要が生じることがあります。

重要: このセクションと、以下のセクションでは、solidDB によって設定される要件のみを説明します。同じシステム上で実行中のその他のプロセスでは、より高いしきい値が必要になる場合があります。

- 共有メモリー・セグメントの最大サイズ

  通常、デフォルトのシステム設定を変更する必要はありません。これは、solidDB のセグメント・サイズ 8 MB はかなり小さいためです。

- システム/プロセス上の共有メモリー・セグメントの最大数

– solidDB では、大部分のセグメントを 8 MB で割り振るため、特に大規模なデータベースを使用する場合は、デフォルトでシステムに許可されているセグメントより多くのセグメントが必要になる場合があります。

  共有メモリー・セグメントの最大数は、少なくとも、solidDB プロセス・サイズ (MB 単位) を 8 で割った値にする必要があります。

  例えば、プロセス・サイズが 1 GB (1024 MB) の場合、少なくとも 128 セグメントが必要です。
– セグメントの最大数を、使用するデータベース・サイズに必要な値より確実に大きい値に常に設定しておく必要があります。値を大きくしておくと、副次作用が生じません。
– solidDB は 1 つのプロセスしか使用しません。このため、ご使用の環境で、セグメントの最大数をプロセスとシステムに対して別々に設定する必要がある場合、両方に同じ値を使用することができます。

- すべての共有メモリー・セグメントの最大合計サイズ

  すべての共有メモリー・セグメントをまとめた合計サイズは、データベースのサイズと、使用可能なディスク・スペースによって異なります。このパラメーターを、マシンで使用可能な物理メモリーより大きい値に設定しないことをお勧めします。

AIX 上での SMA 用の共有メモリー・カーネル・パラメーター

AIX システムでは、共有メモリー・カーネル・パラメーターを変更する必要はありません。 AIX IPC メカニズム用に上限が定義されており、これは構成できません。共有メモリーの制限は、必要に応じて動的に割り振りまたは割り振り解除されるため、メモリー所要量は、常に現在のシステム使用量によって変わります。

AIX 共有メモリーのデフォルト制限、および一般的な IPC メカニズムについて詳しくは、IBM システム・インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/systems/index.jsp) のセクション『プロセス間通信 (IPC) 制限』を参照してください。

HP-UX 上での SMA 用の共有メモリー・カーネル・パラメーターの変更

HP-UX での、共有メモリー・カーネル・パラメーターのデフォルト値では、SMA アプリケーションを実行するのに不十分な場合があります。このカーネル・パラメーター値は、`kctune` コマンドを使用して動的に変更することができます。

始める前に

共有メモリー・カーネル・パラメーターを変更するには、root 権限を持っている必要があります。

このタスクについて

HP-UX 上に、共有メモリー・カーネル・パラメーターを設定する手順を以下に示します。示される最小値は、solidDB によって設定される要件に従っています。同じシステム上で実行中のその他のプロセスでは、より高いしきい値が必要になる場合があります。

HP-UX 環境では、以下の共有メモリー・カーネル・パラメーターを変更する必要がある場合があります。

- **shmmni** — システム上の共有メモリー・セグメントの最大数
- **shmseg** — プロセスに付加される共有メモリー・セグメントの最大数
- **shmmax** — システム上のすべての共有メモリー・セグメントの最大サイズ (バイト単位)

手順

1. **共有メモリー・カーネル・パラメーターを表示して、システムに対して何らかの変更を行う必要があるかどうかを判別します。**

   以下のようにして、**shmmni** パラメーターを表示します。

```
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
```

   以下のようにして、**shmseg** パラメーターを表示します。

```
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
```

   以下のようにして、**shmmax** パラメーターを表示します。

```
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 用の共有メモリー・カーネル・パラメーターの最小要件 (HP-UX)

パラメーター	説明	変更するタイミング	注意
shmmni	システム上の共有メモリー・セグメントの最大数	値が、solidDB プロセス・サイズ (MB) を 8 で割った値より小さい場合 例えば、プロセス・サイズが 1 GB (1024 MB) の場合、少なくとも 128 セグメントが必要です。	このパラメーターを、使用するデータベース・サイズに必要な値より確実に大きい値に常に設定しておく必要があります。値を大きくしておくと、副次作用が生じません。
shmseg	プロセスに付加される共有メモリー・セグメントの最大数	値が、solidDB プロセス・サイズ (MB) を 8 で割った値より小さい場合 例えば、プロセス・サイズが 1 GB (1024 MB) の場合、少なくとも 128 セグメントが必要です。	solidDB では 1 つのプロセスしか使用されないため、**shmmni** と **shmseg** の値を同じにすることができます。
shmmax	共有メモリー・セグメントの最大サイズ (バイト単位)	値が 32768 KB (32 MB) より小さい場合	このパラメーターの値を高く設定しておくと、副次作用が生じません。

2. **パラメーターを変更するには、`kctune` コマンドを使用します。**

   例えば、共有メモリー・セグメントの最大数を 1024 に設定するには、以下のコマンドを使用します。

   ```
   kctune shmmni=1024
   ```

   パラメーター値への変更はすぐに有効になり、リブートの後もそのまま有効になります。

次のタスク

「メモリー不足」エラーを受け取った後に共有メモリー・パラメーターを変更すると、`ipcrm` コマンドを使用して、停止している共有メモリー・セグメントをクリアする必要が生じる場合があります。詳しくは、28 ページの『SMA のトラブルシューティング』を参照してください。

Linux 上での SMA 用の共有メモリー・カーネル・パラメーターの変更

Linux での、共有メモリー・カーネル・パラメーターのデフォルト値では、SMA アプリケーションを実行するのに不十分な場合があります。Linux で共有メモリー・カーネル・パラメーターを変更するには、`/etc/sysctl.conf` ファイルを編集します。

始める前に

カーネル・パラメーターを変更するには、root 権限を持っている必要があります。

このタスクについて

solidDB で設定された共有メモリー所要量を使用して、Red Hat および SUSE Linux 上でカーネル・パラメーターを更新する手順を、以下に示します。同じシステム上で実行中のその他のプロセスでは、より高いしきい値が必要になる場合があります。

Linux 環境では、以下の共有メモリー・パラメーターを変更する必要がある場合があります。

- **SHMMNI** — システム上の共有メモリー・セグメントの最大数
- **SHMMAX** — システム上の単一共有メモリー・セグメントの最大サイズ
- **SHMALL** — システム上の共有メモリー・ページの最大割り振り数

手順

1. **ipcs -l コマンドを実行します。**

   以下に例を示します。

   注: // の後に追加されているコメントは、パラメーター名を示しています。

   ```
   # 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. **出力を分析して、システムに対して何らかの変更が必要かどうかを判別します。**

表 8. SMA 用の共有メモリー・カーネル・パラメーターの最小要件 (Linux)

カーネル・パラメーター	説明	変更するタイミング	注意
SHMMNI	システム上の共有メモリー・セグメントの最大数	値が、solidDB プロセス・サイズ (MB) を 8 で割った値より小さい場合 例えば、プロセス・サイズが 1 GB (1024 MB) の場合、少なくとも 128 セグメントが必要です。	このパラメーターを、使用するデータベース・サイズに必要な値より確実に大きい値に常に設定しておく必要があります。値を大きくしておくと、副次作用が生じません。
SHMMAX	システム上の単一共有メモリー・セグメントの最大サイズ	値が 32768 KB (32 MB) より小さい場合	このパラメーターの値を高く設定しておくと、副次作用が生じません。 **注:** ipcs 出力は、SHMMAX をキロバイトに変換します。カーネルでは、SHMMAX 値をバイト単位で示す必要があります。
SHMALL	システム上の共有メモリー・ページの最大割り振り数	値が、プロセスの最大サイズ (KB) を 4 で割った値より小さい場合	このパラメーターは、コンピューターの物理メモリーのおよそ 90% にまで上げることができます。 例えば、主に SMA に 16 GB のメモリーを使用するコンピューター・システムの場合、SHMALL は 3774873 (16 GB の 90% は 14.4 GB で、14.4 GB を基本ページ・サイズである 4 KB で割って得られる値) に設定する必要があります 。 **注:** ipcs 出力は、SHMALL をキロバイトに変換します。カーネルでは、SHMALL 値をページ数で示す必要があります。

3. **これらのカーネル・パラメーターを変更するには、/etc/sysctl.conf ファイルを編集します。**

   このファイルがない場合は、作成してください。

   以下の行は、ファイルに配置する必要のある行の例です。

   ```
   #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. **sysctl に -p パラメーターを設定して実行し、デフォルト・ファイル /etc/sysctl.conf から sysctl 設定でロードします。**

   ```
   sysctl -p
   ```

5. **リブートごとに、変更を有効にします。**
   - SUSE Linux の場合: boot.sysctl をアクティブにします。
   - Red Hat Linux の場合: rc.sysinit 初期化スクリプトによって /etc/sysctl.conf ファイルが自動的に読み取られます。

次のタスク

「メモリー不足」エラーを受け取った後に共有メモリー・パラメーターを変更すると、ipcrm コマンドを使用して、停止している共有メモリー・セグメントをクリアする必要が生じる場合があります。詳しくは、28 ページの『SMA のトラブルシューティング』を参照してください。

Solaris 上での SMA 用の共有メモリー・カーネル・パラメーターの変更

Solaris 10 での、共有メモリー・カーネル・パラメーターのデフォルト値では、SMA アプリケーションを実行するのに不十分な場合があります。Solaris 10 では、Solaris リソース制御機能を使用して、共有メモリー・カーネル・パラメーター値を動的に変更することができます。

始める前に

共有メモリー・パラメーターを変更するには、root 権限を持っている必要があります。

このタスクについて

Solaris 10 上に、共有メモリー・カーネル・パラメーターを設定する手順を以下に示します。示される最小値は、solidDB によって設定される要件に従っています。同じシステム上で実行中のその他のプロセスでは、より高いしきい値が必要になる場合があります。

Solaris 環境では、以下の共有メモリー・パラメーターを変更する必要がある場合があります。

- **max-shm-ids** — システム上の共有メモリー・セグメントの最大数

- **max-shm-memory** — システム上のすべての共有メモリー・セグメントの最大サイズ (MB)

以下の例では、オペレーティング・システムのデフォルト・プロジェクトが使用されています。

手順

1. **共有メモリー・パラメーターを表示して、システムに対して何らかの変更を行う必要があるかどうかを判別します。**

   以下のようにして、**project.max-shm-ids** パラメーターを表示します。

```
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                    -
```

   以下のようにして、**project.max-shm-memory** パラメーターを表示します。

```
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 用の共有メモリー・カーネル・パラメーターの最小要件 (Solaris)

パラメーター	説明	変更するタイミング	注意
max-shm-ids	システム上の共有メモリー・セグメントの最大数	値が、solidDB プロセス・サイズ (MB) を 8 で割った値より小さい場合 例えば、プロセス・サイズが 1 GB (1024 MB) の場合、少なくとも 128 セグメントが必要です。	このパラメーターを、使用するデータベース・サイズに必要な値より確実に大きい値に常に設定しておく必要があります。値を大きくしておくと、副次作用が生じません。
max-shm-memory	システム上のすべての共有メモリー・セグメントの最大サイズ	値が、プロセス・サイズの最大サイズより小さい場合	このパラメーターの値を高く設定しておくと、副次作用が生じません。

2. **パラメーターを変更するには、prctl コマンドを使用します。**

   例えば、共有メモリー・セグメントの最大数を 1024 に設定するには、以下のコマンドを使用します。

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

3. **リブートごとに、変更を有効にします。**

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

次のタスク

「メモリー不足」エラーを受け取った後に共有メモリー・パラメーターを変更すると、ipcrm コマンドを使用して、停止している共有メモリー・セグメントをクリアする必要が生じる場合があります。詳しくは、28 ページの『SMA のトラブルシューティング』を参照してください。

ドライバー・マネージャーを使用する SMA 用のアプリケーションの準備

SMA でドライバー・マネージャーを使用する場合、solidDB ODBC データ・ソースに接続するときと同様の方法で SMA データ・ソースに接続します。

このタスクについて

SMA ドライバー・ライブラリー・ファイルは、solidDB のインストール時にインストールされます。以下の表では、最も一般的なプラットフォームでのファイル名およびデフォルトのインストール・ロケーションをリストしています。

表 10. SMA ドライバー (ライブラリー)

プラットフォーム	SMA ドライバー・ライブラリー	デフォルトのインストール場所
Windows	ssolidsmaxx.dll **注:** SMA ドライバーに直接リンクする場合、実際の .dllライブラリー・ファイルへのアクセスを提供する solidsma.lib インポート・ライブラリー・ファイルにリンクします。	ライブラリー: <solidDB installation directory>¥bin インポート・ライブラリー: <solidDB installation directory>¥lib
Linux	ssolidsmaxx.so	<solidDB installation directory>/bin
Solaris	ssolidsmaxx.so	<solidDB installation directory>/bin
HP-UX	ssolidsmaxx.sl	<solidDB installation directory>/bin
AIX	ssolidsmaxx.so	<solidDB installation directory>/bin
xx は、ドライバー・ライブラリーのバージョン番号であり、例えば、ssolidsma65.so のようになります。		

手順

1. **SMA データ・ソースに接続します。**

   データ・ソースの接続情報を定義する場合、SMA 固有の接続ストリングを使用します。

   SMA 接続の接続ストリング構文は、以下のとおりです。

   ```
   sma <protocol name> <port number or pipe name>
   ```

   例えば、以下のように指定します。

   ```
   sma tcp 2315
   ```

2. **シグナル・ハンドラーを使用不可にします。**

   シグナル・ハンドラーは、例外イベントの発生をアプリケーションに報告するために使用されます。アプリケーションではシグナル・ハンドラーを設定しないでください。これは、SMA によって設定されたシグナル・ハンドラーがオーバーライドされてしまうためです。

例えば、アプリケーションで浮動小数点例外に対してシグナル・ハンドラーを設定すると、SMA によって設定されたハンドラーがオーバーライドされます。このため、サーバーは例えばゼロによる除算を catch できなくなります。

ドライバー・マネージャーを使用しない SMA 用のアプリケーションの準備

SMA でドライバー・マネージャーを使用しない場合、solidDB ODBC ドライバー・ライブラリーに直接リンクする場合と同様の方法で、アプリケーションを SMA ドライバー・ライブラリーに直接リンクします。

手順

1. **アプリケーションを SMA ドライバー・ライブラリーにリンクします。**

   SMA ドライバー・ライブラリー・ファイルは、solidDB のインストール時にインストールされます。以下の表では、最も一般的なプラットフォームでのファイル名およびデフォルトのインストール・ロケーションをリストしています。

   表 11. SMA ドライバー (ライブラリー)

プラットフォーム	SMA ドライバー・ライブラリー	デフォルトのインストール場所
Windows	ssolidsmaxx.dll **注:** SMA ドライバーに直接リンクする場合、実際の .dllライブラリー・ファイルへのアクセスを提供する solidsma.lib インポート・ライブラリー・ファイルにリンクします。	ライブラリー: <solidDB installation directory>¥bin インポート・ライブラリー: <solidDB installation directory>¥lib
Linux	ssolidsmaxx.so	<solidDB installation directory>/bin
Solaris	ssolidsmaxx.so	<solidDB installation directory>/bin
HP-UX	ssolidsmaxx.sl	<solidDB installation directory>/bin
AIX	ssolidsmaxx.so	<solidDB installation directory>/bin
xx は、ドライバー・ライブラリーのバージョン番号であり、例えば、ssolidsma65.so のようになります。		

2. **接続ストリングをローカルの SMA サーバー名に変更します。**

   SMA 接続の接続ストリング構文は、以下のとおりです。

   ```
   sma <protocol name> <port number or pipe name>
   ```

   例えば、以下のように指定します。

   ```
   sma tcp 2315
   ```

   ODBC API または SA API を使用する場合の接続ストリングの例については、25 ページの『SMA 用のローカル接続の確立』を参照してください。

3. **シグナル・ハンドラーを使用不可にします。**

シグナル・ハンドラーは、例外イベントの発生をアプリケーションに報告するために使用されます。アプリケーションではシグナル・ハンドラーを設定しないでください。これは、SMA によって設定されたシグナル・ハンドラーがオーバーライドされてしまうためです。

例えば、アプリケーションで浮動小数点例外に対してシグナル・ハンドラーを設定すると、SMA によって設定されたハンドラーがオーバーライドされます。このため、サーバーは例えばゼロによる除算を catch できなくなります。

SMA 用のローカル接続の確立

SMA を使用する場合、アプリケーションは SMA サーバーを使用してローカル SMA 接続を確立する必要があります。接続タイプは、SMA 固有の接続ストリングを使用して定義されます。

SMA の場合、接続要求はローカルで使用可能なプロトコル (TCPIP、名前付きパイプ、UNIX パイプ) を使用して、ネットワーク接続 (ハンドシェーク接続) 経由で送信されます。ハンドシェーク接続時に、共有メモリー・セグメント・ハンドルがドライバーに渡されるため、ドライバーはサーバーの共有メモリーにアクセスできます。

SMA 接続の接続ストリング構文は、以下のとおりです。

```
sma <protocol name> <port number or pipe name>
```

例えば、以下のように指定します。

```
sma tcp 2315
```

SMA 接続要求が、SMA サーバー以外のサーバーに実行された場合、接続エラーが返されます。

重要: 1 つのアプリケーションが接続できるのは、1 つの SMA サーバーのみです。ただし、SMA アプリケーションは、あらゆるローカル・サーバーまたはリモート・サーバーに対して、通常のネットワーク・ベースの接続を実行できます。

ODBC API

ODBC API を使用する場合は、SQLConnect 関数呼び出しに SMA 固有の接続ストリングを定義します。

例

以下の ODBC API コードの例では、ユーザー名 dba とパスワード dba を使用して、ローカルの SMA solidDB サーバーに直接接続します。

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

または

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

SA API

SA API を使用する場合は、SaConnect 関数呼び出しに SMA 固有の接続ストリングを定義します。

以下のコードの例では、ユーザー名 dba とパスワード dba を使用して、ローカルの SMA solidDB サーバーに直接接続します。

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

または

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

ドライバー・マネージャー

ドライバー・マネージャーを使用する場合は、SMA データ・ソースに SMA 固有の接続ストリングを定義します。

SMA サーバーの始動とシャットダウン

SMA サーバーは、通常のネットワーク・プロトコル・ベースの solidDB サーバーと同様の方法で始動、再始動、およびシャットダウンします。

SMA サーバーの始動

SMA サーバーは、通常のネットワーク・プロトコル・ベースの solidDB サーバーと同様にコマンド・プロンプトを使用して始動します。SMA サーバーを始動すると、データベースが既に存在しているかどうかが検査されます。サーバーは最初に `solid.ini` 構成ファイルを検索し、`FileSpec` パラメーターの値を読み取ります。`FileSpec` パラメーターで指定された名前とパスのデータベース・ファイルが検出されると、そのデータベースは自動的に開きます。データベースが検出されない場合は、サーバーは新しいデータベースを作成するためのプロンプトを出します。

手順

SMA サーバーを始動するには、以下のようにします。

表 12. SMA サーバーの始動

オペレーティング・システム	SMA サーバーを始動する操作
Linux および UNIX	コマンド・プロンプトに対して、コマンド `solidsma` を入力します。 初めてサーバーを始動する場合は、コマンド・プロンプトに対してコマンド `solidsma -f` を入力して、サーバーをフォアグラウンドで強制的に稼働します。
Windows	コマンド・プロンプトに対して、コマンド `solidsma` を入力します。

タスクの結果

サーバーを SMA モードで始動すると、サーバーは SMA ドライバー・ライブラリーを動的にロードし、通常の listen ポートで SMA ドライバーからの接続要求を受け入れます。それぞれの SMA サーバーに異なるポート番号を割り当てることで、単一システムで複数の SMA サーバーを同時に稼働することが可能です。

ヒント: アプリケーションで SSC API 関数 SSCStartSMAServer を呼び出して、SMA モードで solidDB サーバーを始動することもできます。ただし、このようなセットアップでは、SMA サーバーを始動 (および停止) できるアプリケーション

は、1 つのみです。SSC API 呼び出しについて詳しくは、 87 ページの『SSCStartSMAServer』を参照してください。

SMA サーバーのシャットダウン

SMA サーバーは、solidDB ADMIN COMMAND を使用してシャットダウンします。

手順

1. solidDB との新しい接続が作成されないように、以下のコマンドを入力してデータベースを閉じます。

   ```
   ADMIN COMMAND 'close'
   ```

2. 以下のコマンドを入力してすべての solidDB ユーザーを終了します。

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

3. 以下のコマンドを入力して、solidDB を停止します。

   ```
   ADMIN COMMAND 'shutdown'
   ```

タスクの結果

シャットダウン・メカニズムはいずれも同じルーチンを開始します。このルーチンは、バッファー内の全データをデータベース・ファイルに書き込み、キャッシュ・メモリーを解放し、最後にサーバー・プログラムを終了します。サーバーでバッファー内の全データをメイン・メモリーからディスクに書き込む必要があるため、サーバーのシャットダウンには時間がかかることがあります。

ヒント: また、アプリケーションで SSC API 関数 SSCStopServer() を呼び出すことでも、SMA モードで solidDB サーバーを停止することが可能です。 SMA サーバーを始動および停止できるアプリケーションは、1 つのみです。SMA サーバーを始動したものと同じアプリケーションでシャットダウンを実行する必要があります。詳しくは、『 89 ページの『SSCStopServer』』を参照してください。

SMA のモニター

solidDB には、SMA 接続のタイプと数、および SMA メモリー・セグメント・サイズに関するデータをモニターおよび収集するための手段が用意されています。

- `ADMIN COMMAND 'userlist'` を使用して、ユーザー接続のタイプのリストを印刷します (ネットワーク・クライアントまたは SMA クライアント)。
- `ADMIN COMMAND 'report'` を使用して、接続のリストをタイプ別に印刷します。
- `solmsg.out` ファイル・ログイン項目で、作成された接続のタイプを確認します。
- パフォーマンス・カウンター `pmon xxx` を使用して、SMA 接続数に関するデータを収集します。
- パフォーマンス・カウンター `pmon yyyy` を使用して、SMA メモリー・セグメント・サイズに関するデータを収集します。

ADMIN COMMAND、`solmsg.out`、およびパフォーマンス・カウンターの使用の詳細については、「*IBM solidDB 管理者ガイド*」を参照してください。

SMA のトラブルシューティング

このセクションでは、SMA を構成または使用する際の共通の問題を回避し、トラブルシューティングする方法について説明し、ガイドラインを示します。

エラー: サーバーが id -1 によって共有メモリー・セグメントを割り振ることができない

症状

SMA サーバーを始動しようとすると、以下のタイプのエラーが表示され、SMA を始動することができません。

```
IBM solidDB process has encountered an internal error and is unable to
continue normally. Please report the following information to technical support.
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

使用可能なメモリーがないため、SMA サーバーの始動が失敗しました。この状態は、以下の場合に生じることがあります。

- SMA アプリケーションまたは solidDB が異常終了して、共有メモリーが割り振られたままになっている可能性がある場合。すべての SMA プロセスをシャットダウンしても、共有メモリーは予約されたままになります。
- SMA の使用に対して割り振ったメモリーが少なすぎた場合。

これによって、すべてのメモリーが使用され、SMA サーバーを始動できなくなってしまう状況になります。

問題の解決

Linux および UNIX 環境では、`ipcrm` コマンドを使用して、停止している共有メモリー・セグメントをクリアします。

例えば Linux 環境では、以下のスクリプトを使用して、未使用の共有メモリー・セグメントを特定し、除去します。

```
#!/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
```

`ipcrm` コマンドについて詳しくは、ご使用のオペレーティング・システムの資料を参照してください。

3 Java による SMA アプリケーションの作成と実行

Java アプリケーションは、SMA ドライバー・ライブラリーにリンクされます。実際のデータベース接続は、標準の JDBC API を使用して行われます。

Java を使用する場合の SMA の使用の概要

SMA を使用する Java アプリケーションは、通常の solidDB サーバーではなく、SMA サーバーを始動するという点を除けば、通常の solidDB サーバーを使用するアプリケーションと同じ方法で作成されます。 Java アプリケーションは SMA サーバーに接続し、標準の JDBC API を介して、solidDB サーバーが提供するサービスを使用します。動的ライブラリーにリンクすることで、アプリケーションではネットワークを介した RPC (リモート・プロシージャー・コール) のオーバーヘッドを回避できます。

SMA を使用する Java/JDBC プログラムは、SMA ドライバー・ライブラリー (`ssolidsmaxx`) にリンクします。このライブラリーには solidDB サーバー全体が、スタンドアロン実行可能プログラムではなく呼び出し可能なライブラリーの形式で格納されています。Java/JDBC で使用されるライブラリーは、C/C++ アプリケーションで使用されるものと同じです。したがって、Java 用の個別バージョンは存在しません。

Java/JDBC で SMA を使用する場合は、以下の要素をリンクして、単一の実行可能プロセスにします。

- SMA ドライバー・ライブラリー
- Java 言語のクライアント・プログラム
- JVM

実行可能プロセスでのレイヤーは、上から順に以下のとおりです。

- ローカルの Java/JDBC クライアント・アプリケーション
- JVM (Java 仮想マシン)
- SMA ドライバー・ライブラリー

クライアントの Java コマンドは JVM によって実行されます。コマンドが JDBC 関数呼び出しである場合は、JVM によって、SMA ドライバー・ライブラリー内の適切な関数が呼び出されます。関数呼び出しは、ネットワーク (RPC) を介さずに直接実行されます。呼び出しには、Java ネイティブ・インターフェース (JNI) が使用されます。ユーザーは JNI コードを作成する必要はなく、リモート・クライアント・プログラムの場合と同じ JDBC 関数を呼び出すだけです。

SMA を使用するすべてのアプリケーションは、以下の 4 ステップと同じ基本パターンに準拠します。

1. solidDB サーバーおよび接続の設定を構成します。
2. SMA サーバーを始動します。
3. 標準の JDBC API を使用してデータベースにアクセスします。

4. データベースの処理が完了すると、SMA サーバーを停止します。

Java を使用する場合の SMA 用の環境の構成

SMA を Java とともに使用する場合、PATH (Windows の場合) または LD_LIBRARY_PATH (Linux および UNIX の場合) 環境変数に、SMA ドライバー・ライブラリーの場所を含める必要があります。

始める前に

solidDB JDBC ドライバーのインストール済み作業環境が用意されていることが前提となります。

このタスクについて

SMA ドライバー・ライブラリー・ファイルは、solidDB のインストール時にインストールされます。以下の表では、最も一般的なプラットフォームでのファイル名およびデフォルトのインストール・ロケーションをリストしています。

表 13. SMA ドライバー (ライブラリー)

プラットフォーム	SMA ドライバー・ライブラリー	デフォルトのインストール場所
Windows	ssolidsmaxx.dll **注:** SMA ドライバーに直接リンクする場合、実際の .dllライブラリー・ファイルへのアクセスを提供する solidsma.lib インポート・ライブラリー・ファイルにリンクします。	ライブラリー: <solidDB installation directory>¥bin インポート・ライブラリー: <solidDB installation directory>¥lib
Linux	ssolidsmaxx.so	<solidDB installation directory>/bin
Solaris	ssolidsmaxx.so	<solidDB installation directory>/bin
HP-UX	ssolidsmaxx.sl	<solidDB installation directory>/bin
AIX	ssolidsmaxx.so	<solidDB installation directory>/bin
xx は、ドライバー・ライブラリーのバージョン番号であり、例えば、ssolidsma65.so のようになります。		

手順

1. **SMA ドライバー・ライブラリーの場所を、PATH (Windows の場合) または LD_LIBRARY_PATH (Linux および UNIX の場合) 環境変数に追加します。**
   - Windows 環境では、以下の構文を使用します。

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

   - Linux および UNIX 環境では、以下の構文を使用します。

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

2. **作業ディレクトリー、solidDB データベース、およびユーザー・アカウントを作成して、データベース環境をセットアップします。**

   詳しくは、「*IBM solidDB 管理者ガイド*」の『*データベースの新規作成*』を参照してください。

3. **環境、パフォーマンス、および操作のニーズに合わせて、solidDB を構成します。**

   solid.ini 構成ファイルを使用して、データベース・ファイル名や場所、データベース・ブロック・サイズなどの基本構成設定を定義します。

   - 通常のセットアップでは、solid.ini ファイルの [SharedMemoryAccess] セクション内の SMA 固有パラメーターを変更する必要はありません。大部分のユース・ケースには、ファクトリー値を適用できます。
   - パラメーター **Srv.ProcessMemoryLimit** は、SMA サーバーでは無効です。

   構成ファイルが存在しなければ、ファクトリー値が使用されます。

SMA サーバーの始動とシャットダウン

SMA サーバーは、通常のネットワーク・プロトコル・ベースの solidDB サーバーと同様の方法で始動、再始動、およびシャットダウンします。

SMA サーバーの始動

SMA サーバーは、通常のネットワーク・プロトコル・ベースの solidDB サーバーと同様にコマンド・プロンプトを使用して始動します。SMA サーバーを始動すると、データベースが既に存在しているかどうかが検査されます。サーバーは最初に solid.ini 構成ファイルを検索し、FileSpec パラメーターの値を読み取ります。FileSpec パラメーターで指定された名前とパスのデータベース・ファイルが検出されると、そのデータベースは自動的に開きます。データベースが検出されない場合は、サーバーは新しいデータベースを作成するためのプロンプトを出します。

手順

SMA サーバーを始動するには、以下のようにします。

表 14. SMA サーバーの始動

オペレーティング・システム	**SMA サーバーを始動する操作**
Linux および UNIX	コマンド・プロンプトに対して、コマンド solidsma を入力します。 初めてサーバーを始動する場合は、コマンド・プロンプトに対してコマンド solidsma -f を入力して、サーバーをフォアグラウンドで強制的に稼働します。
Windows	コマンド・プロンプトに対して、コマンド solidsma を入力します。

SMA サーバーのシャットダウン

SMA サーバーは、solidDB ADMIN COMMAND を使用してシャットダウンします。

手順

1. solidDB との新しい接続が作成されないように、以下のコマンドを入力してデータベースを閉じます。

   ```
   ADMIN COMMAND 'close'
   ```

2. 以下のコマンドを入力してすべての solidDB ユーザーを終了します。

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

3. 以下のコマンドを入力して、solidDB を停止します。

```
ADMIN COMMAND 'shutdown'
```

SMA の JDBC 接続の作成

SMA サーバーに対するローカルの (RPC ベースではない) JDBC 接続では、solidDB 固有の接続プロパティーである solid_shared_memory を使用して、SMA サーバーに接続すること、および所定のポートでローカル・サーバーを使用することを定義する必要があります。

ドライバー・マネージャーによる接続

非標準の接続プロパティー solid_shared_memory をコードに組み込みます。

以下に例を示します。

```
Properties props = new Properties();
// 直接アクセス・プロパティーを使用可能にする
props.put("solid_shared_memory", "yes");
// 接続を取得する
Connection c = DriverManager.getConnection
("jdbc:solid://localhost:1315", props);
```

接続ストリングでの接続プロパティーの定義

接続プロパティーを接続ストリングに組み込みます。

以下に例を示します。

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

注: DriverManager クラスに加えて、SolidDataSource クラスおよび SolidConnectionPoolDataSource クラスでも同様の構文を使用できます。

4 LLA アプリケーションの作成と実行

LLA アプリケーションの作成作業には、ライブラリーへのアプリケーションのリンク、サーバーの始動、およびアプリケーションとサーバー間のローカル接続の確立などが含まれます。サーバーの始動と停止には、SSC API、ODBC API、および SA API を使用できます。

LLA 固有の追加事項、補足事項、および LLA を使用しない場合の solidDB と比較した使用法の違いを説明します。

solidDB SQL、solidDB データ管理ツール、solidDB の一般的な管理と保守、およびデータベース・エラー・コードについては、「*IBM solidDB 管理者ガイド*」を参照してください。

API および solidDB JDBC および ODBC ドライバーの詳細については、「*IBM solidDB プログラマー・ガイド*」を参照してください。

LLA 用の環境の構成

LLA を使用する場合、アプリケーションを LLA ライブラリー・ファイルにリンクする必要があります。

手順

1. **オペレーティング・システム固有の LLA ライブラリー・ファイルにアプリケーションをリンクします。**

表 15. リンク・ライブラリー・アクセス (LLA) システム・ライブラリー

プラットフォーム	LLA ライブラリー
Windows	`solidimpac.lib` これはインポート・ライブラリー・ファイルで、実際のライブラリー・ファイルである `ssolidacxx.dll` へのアクセスを提供します。
Solaris	`solidac.a`
HP-UX	`solidac.a`
Linux	`solidac.a`

例: Windows で LLA ライブラリー名を指定するための Make ファイル

以下の Make ファイルの例では、`solidimpac.lib` ライブラリーが使用されています。

```
# コンパイラー
CC        = cl
# コンパイラー・フラグ
```

```
CFLAGS        = -I. -DSS_WINDOWS -DSS_WINNT
# リンカー・フラグとディレクティブ
SYSLIBS = libcmt.lib kernel32.lib advapi32.lib netapi32.lib wsock32.lib
user32.lib oldnames.lib gdi32.lib
LFLAGS  = ..¥solidimpac.lib
OUTFILE  = -Fe

# MyApp のビルド
all: myapp

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

2. **SSC API を使用して solidDB サーバーを始動する場合に、暗黙的な start メソッドを使用する計画がない場合は、ImplicitStart パラメーターを「no」に設定します。**

   solid.ini 構成ファイルの [Accelerator] セクションでは、パラメーター **ImplicitStart** がデフォルトで Yes に設定されます。このデフォルト設定では、ODBC 接続に必要な SQLConnect 関数を使用すると、自動的にサーバーが始動されます。関数 SaConnect も同じように動作します。SQLConnect または SaConnect が初めて呼び出されるときに、サーバーが暗黙的に始動されます。

3. **シグナル・ハンドラーを使用不可にします。**

   シグナル・ハンドラーは、例外イベント (ゼロによる除算など) の発生をアプリケーションに報告するために使用されます。ユーザー・アプリケーションではシグナル・ハンドラーを設定しないようにする必要があります。これは、リンク・ライブラリー・アクセスによって設定されたシグナル・ハンドラーがオーバーライドされてしまうためです。例えば、ユーザー・アプリケーションで浮動小数点例外に対してシグナル・ハンドラーを設定すると、リンク・ライブラリー・アクセスによって設定されたハンドラーがオーバーライドされます。このため、サーバーは例えばゼロによる除算を catch できなくなります。

LLA 用のローカル接続の確立

リンク・ライブラリー・アクセス・ライブラリーにリンクされたアプリケーションでは、ODBC API または SAAPI を使用して、ローカル・サーバーとのローカル接続またはリモート接続を直接確立できます。また、リンク・ライブラリー・アクセスを使用するサーバーも含めた他の solidDB サーバーとのリモート接続を確立することもできます。

ODBC API でローカル・サーバー (アプリケーションにリンクされたサーバー) との接続を確立するには、ユーザー・アプリケーションでリテラル・ストリング ″localserver″ を指定して、SQLConnect 関数を呼び出します。ローカル・サーバー接続の場合は、空のソース名 ″″ を指定することもできます。ローカル・サーバー名を指定することもできますが、その場合はリンク・ライブラリー・アクセスで「リモート」接続が使用されます (リンク・ライブラリー・アクセス・ライブラリーへの直接関数呼び出しを使用するのではなくネットワーク経由で接続)。

以下の ODBC API コードの例では、ユーザー名 dba とパスワード dba を使用して、ローカルの solidDB サーバーに直接接続します。

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

または

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

SA API で接続を確立するには、ユーザー・アプリケーションでリテラル・ストリング ″localserver″ (サーバー名ではなく) を指定して SaConnect 関数を呼び出します。ローカル・サーバー接続の場合は、空のソース名 ″″ を指定することもできます。ローカル・サーバー名を指定することもできますが、その場合はリンク・ライブラリー・アクセスで「リモート」接続が使用されます (リンク・ライブラリー・アクセス・ライブラリーへの直接関数呼び出しを使用するのではなくネットワーク経由で接続)。

以下の SA API コードの例では、ユーザー名 dba とパスワード dba を使用して solidDB サーバーに直接接続します。

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

または

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

LLA サーバーの始動とシャットダウン

LLA サーバーを始動、再始動、およびシャットダウンするには、SSC API、ODBC API、または SA API 関数呼び出しを使用します。ODBC API および SA API 関数呼び出しは、データベースが既に存在する場合に限り、サーバーを始動するために使用できます。SSC API は、始動時に新規データベースを作成するために使用できます。

サーバー始動時に、アプリケーションに制御が戻る前に必要に応じてリカバリーが実行されます。したがって、サーバーが正常に始動された場合は、サーバーがアプリケーション要求を処理できる状態にあります。アプリケーション・プロセスが実行されている間は、サーバーを必要に応じて始動または停止できます。

SSC API による明示的な始動とシャットダウン

SSC API は、LLA サーバーを*明示的に* 始動およびシャットダウンするために使用します。アプリケーションは、solidDB を始動する場合には、SSC API 関数 SSCStartServer を呼び出し、停止する場合には、SSCStopServer を呼び出します。

まだデータベースが存在しない新しい LLA サーバーを始動する場合は、SSCStartServer() 関数に以下のパラメーターを指定して、新規データベースを作成することを明示的に指定する必要があります。

-Uusername
-Ppassword
-Ccatalogname (デフォルトのデータベース・カタログ名)

注: ディスクレス・サーバーを始動する場合は、SSC API 関数 SSCStartDisklessServer でサーバーを始動する必要があります。
詳しくは、『36 ページの『SSC API 関数 SSCStartServer による明示的な始動』』を参照してください。

ODBC API および SA API による暗黙的な始動とシャットダウン

ODBC API および SA API は、LLA サーバーを*暗黙的に* 始動およびシャットダウンする場合にのみ使用できます。アプリケーションは、初めてローカルで LLA サーバーに接続するとき、ODBC API 関数 SQLConnect または SA API 関数 SaConnect を呼び出します。この場合、solidDB との最後のローカル接続が関数 SQLDisconnect または SaDisconnect で切断されると、シャットダウンが行われます。

アプリケーションから LLA サーバーを暗黙的に始動するとき、アプリケーションは、データベースが solidDB 作業ディレクトリーに存在するかどうかを検査します。データベース・ファイルが見つかった場合は、solidDB が自動的にそのデータベースを開きます。データベース・ファイルが見つからなかった場合は、solidDB からエラーが通知されます

暗黙的な始動では、solidDB で新しいデータベースは作成されません。新しいデータベースを作成するには、適切なパラメーターを指定して SSCStartServer などの明示的な始動関数を使用するか、通常の solidDB サーバーの場合と同様にデータベースを作成する必要があります。

詳細については、 38 ページの『ODBC API 関数呼び出し SQLConnect による暗黙的な始動』および 39 ページの『SA API 関数呼び出し SaConnect による暗黙的な始動』を参照してください。

通常の solidDB セットアップでデータベースを作成する方法については、「*IBM solidDB 管理者ガイド*」の『*データベースの新規作成*』セクションを参照してください。.

SSC API 関数 SSCStartServer による明示的な始動

solidDB を明示的に始動するには、ユーザー・アプリケーションで以下の制御 API 関数を呼び出します。

solidDB を明示的に始動するには、ユーザー・アプリケーションで以下の SSC API 関数を呼び出します。

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

ここで使用されているパラメーターを以下で説明します。

表 16. SSCStartServer のパラメーター

パラメーター	説明
argc	コマンド行引数の数。
argv	関数呼び出しで使用されるコマンド行引数の配列。引数 argv[0] は、ユーザー・アプリケーションのパスおよびファイル名用として予約されており、必ず指定する必要があります。有効なオプションについては、以下の SSCStartServer オプションを参照してください。

表 16. SSCStartServer のパラメーター (続き)

パラメーター	説明
h	各サーバーには、そのサーバーを識別し、そのサーバーに関する情報の保管場所を示す「ハンドル」(データ構造へのポインター) があります。このハンドルは、他の制御 API 関数でサーバーを参照する際に必要となります。サーバーのハンドルは、SSCStartServer 関数を呼び出したときに提供されます。 サーバーのハンドルを取得するには、「サーバー・ハンドルへのポインター」型の変数、つまり、ハンドルへのポインター (本質的にはポインターへのポインター) である SSCServerT *) を作成し、SSCStartServer を呼び出すときにその変数を渡します。サーバーが正常に作成されると、SSCStartServer 関数によって新しいサーバーのハンドル (ポインター) が、アドレスを渡した変数に書き込まれます。
runflags	このパラメーターのオプションは、SSC_STATE_OPEN (リモート接続を許可) および SSC_STATE_PREFETCH (サーバーが必要に応じてプリフェッチを実行) です。プリフェッチでは、表内容の先読み機能を備えたメモリーおよびディスク・キャッシュ、またはそのいずれかが参照されます。runflags パラメーターは以下のように入力します。 `runflags = SSC_STATE_OPEN | SSC_STATE_PREFETCH;`

既存のデータベースを使用しない LLA サーバーの始動

サーバーを最初に始動するときに、データベース管理者のユーザー名、パスワード、およびデフォルト・データベース・カタログの名前を指定した場合に限り、solidDB によって新しいデータベースが作成されます。

以下に例を示します。

```
SscServerT h; char* argv[4];
argv[0] = "appname"; /* ユーザー・アプリケーションのパスとファイル名 */
argv[1] = "-UDBA"; /* ユーザー名 */
argv[2] = "-PDBA"; /* ユーザーのパスワード */
argv[3] = "-CDBA"; /* カタログ名 */
/* サーバーの始動 */
rc = SSCStartServer(argc, argv, &h, run_flags);
```

データベースが存在しない状態でデータベース・カタログ名を指定せずにサーバーを始動すると、データベースが見つからないことを通知するエラーが solidDB から返されます。

デフォルトでは、データベースが 1 つのファイルとして、デフォルト名の solid.db または solid.ini ファイルで指定した名前で、solidDB 作業ディレクトリーに作成されます。システム表とシステム・ビューのみで構成される空のデータベースは、約 850 KB のディスク・スペースを使用します。データベースの作成に要する時間は、使用しているハードウェア・プラットフォームによって異なります。

データベースが作成されると、solidDB がネットワークでリモート・クライアント接続要求の listen を開始します。

既存のデータベースを使用する LLA サーバーの始動

データベースが既に存在する場合は、SSCStartServer 関数呼び出しでユーザー名とパスワード、またはカタログ名を指定する必要はありません。

ODBC API 関数呼び出し SQLConnect による暗黙的な始動

関数 SQLConnect が初めて呼び出されるときに、サーバーが暗黙的に始動されます。ユーザー・アプリケーションが開いている最後のローカル接続に対して関数 SQLDisconnect を呼び出すと、サーバーが暗黙的にシャットダウンされます。その場合、リモート接続が存在していてもサーバーがシャットダウンされるので注意してください。

注: サーバーを初めて始動するときは、 solidDB データベースを作成する必要があります。そのためには、関数 SSCStartServer() を使用し、デフォルトのデータベース・カタログ、および管理者のユーザー名とパスワードを指定します。説明と例については、 36 ページの『SSC API 関数 SSCStartServer による明示的な始動』を参照してください。

SQLConnect および SQLDisconnect による暗黙的な始動とシャットダウンの例を以下に示します。

```
/* 接続 #1 */
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); // ここでサーバーが始動
... ODBC 呼び出し

/* 切断 #1 */
SQLDisconnect (hdbc1); // ここでサーバーがシャットダウン

/* 接続 #2 */
rc = SQLConnect (hdbc2, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); // ここでサーバーが始動
... ODBC 呼び出し

/* 切断 #2 * /
SQLDisconnect (hdbc2); // ここでサーバーがシャットダウン
```

または

```
/* 接続 #1*/
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba",
SQL_NTS, "dba", SQL_NTS); // ここでサーバーが始動

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

... ODBC 呼び出し

/* 切断 #1 */
SQLDisconnect (hdbc1);
/* 切断 #2 * /
SQLDisconnect (hdbc2); // ここでサーバーがシャットダウン
```

注: SSCStartServer 関数呼び出しでサーバーを始動した場合は、SQLDisconnect で暗黙シャットダウンが行われません。SSCStopServer、ADMIN COMMAND 'shutdown'、またはその他の明示的なシャットダウン・メソッドで、サーバーを明示的にシャットダウンする必要があります。

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

/* サーバーの始動 */
SSCStartServer (argc, argv, &server, runflags); // ここでサーバーが始動

/* 環境の割り振り */
rc = SQLAllocEnv (&henv);

/* データベースへの接続 */
rc = SQLAllocConnect (henv, &hdbc);
rc = SQLConnect (hdbc, "", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);

/* 表 foo からの全行削除 */
rc = SQLAllocStmt (hdbc, &hstmt):
rc = SQLExecDirect (hsmt, (SQLCHAR *) "DELETE FROM FOO", SQL_NTS);

/* コミット */
rc = SQLTransact (henv, hdbc, SQL_COMMIT);
rc = SQLFreeStmt (hstmt, SQL_DROP);

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

/* 環境の解放 */
SQLFreeEnv(henv);

/* サーバーの停止 */
SSCStopServer (server, TRUE);  // ここでサーバーがシャットダウン
```

SA API 関数呼び出し SaConnect による暗黙的な始動

関数 SaConnect が初めて呼び出されるときに、サーバーが暗黙的に始動されます。ユーザー・アプリケーションが関数 SaDisconnect を呼び出したときにそれ以上の接続が残っていない場合は、サーバーが暗黙的にシャットダウンされます。

注: サーバーを初めて始動するときは、 solidDB データベースを作成する必要があります。そのためには、関数 SSCStartServer() を使用し、デフォルトのデータベース・カタログ、およびユーザー名とパスワードを指定します。説明と例については、 36 ページの『SSC API 関数 SSCStartServer による明示的な始動』を参照してください。

SaConnect および SaDisconnect による暗黙的な始動とシャットダウンの例を以下に示します。

```
/* 接続を開く */
SaConnect(...);

ここでサーバーが始動
... sa 呼び出し

/* 接続を閉じる */
SaDisconnect(...);

ここでサーバーがシャットダウン
```

注: サーバーを SSCStartServer 関数呼び出しで始動した場合は、SSCStopServer 関数呼び出しでそのサーバーをシャットダウンする必要があります。

LLA サーバーのシャットダウン

SYS_ADMIN_ROLE 特権を所有している場合、solidDB クライアント・インターフェースから、および別のリモート solidDB 接続からでも、solidDB サーバーをシャットダウンできます。

プログラムで、solidDB SQL エディター (solsql) や solidDB リモート制御などのアプリケーションからシャットダウンを実行できます。[1].

そのためには、以下のステップを実行します。

1. solidDB との新しい接続が作成されないように、以下のコマンドを入力してデータベースを閉じます。

   ADMIN COMMAND 'close'
2. 以下のコマンドを入力してすべての solidDB ユーザーを終了します。

   ADMIN COMMAND 'throwout all'
3. 以下のコマンドを入力して、solidDB を停止します。

   ADMIN COMMAND 'shutdown'

シャットダウン・メカニズムはいずれも同じルーチンを開始します。このルーチンは、バッファー内の全データをデータベース・ファイルに書き込み、キャッシュ・メモリーを解放し、最後にサーバー・プログラムを終了します。サーバーでバッファー内の全データをメイン・メモリーからディスクに書き込む必要があるため、サーバーのシャットダウンには時間がかかることがあります。

注: 暗黙的な方法 (SQLConnect) で始動されたサーバーは、明示的な方法 (SSCStopServer) でシャットダウンできます。ただし、その逆は不可能です。例えば、SSCStartServer で始動されたサーバーを SQLDisconnect で停止することはできません。

SSCStopServer による LLA サーバーのシャットダウン

サーバーを SSCStartServer で始動した場合は、組み込みアプリケーションで以下の関数呼び出しを使用してそのサーバーをシャットダウンする必要があります。

SSCStopServer()

以下に例を示します。

```
/* サーバーの停止 * /
SSCStopServer (h, TRUE);
```

LLA のサンプル C アプリケーション

solidDB パッケージには、ODBC API 関数を使用して solidDB サーバーに接続する、C で作成された LLA アプリケーションのサンプルが含まれています。

1. ステップ 1 から 3 に solidDB リモート制御を使用する場合は、引用符で囲まずにコマンド名のみを入力します (例えば、close)。

これらのサンプルは、solidDB インストール・ディレクトリーの以下のディレクトリーにあります。

- samples/aclib: 単一の solidDB を使用する LLA アプリケーションのサンプル
- samples/aclib_control_api: SSC API 関数を使用する LLA アプリケーションのサンプル
- samples/aclib_diskless: ディスクレス solidDB サーバーを使用する LLA アプリケーションのサンプル
- samples/aclib_replication: LLA と拡張レプリケーションを組み合わせる LLA アプリケーションのサンプル

  レプリケーション・サンプルの使用方法については、『拡張レプリケーションで LLA を使用するためのサンプル』を参照してください。

各ディレクトリーには readme.txt ファイルが含まれており、このファイルにシステムのセットアップ方法とサンプルの実行方法が記載されています。

拡張レプリケーションで LLA を使用するためのサンプル

solidDB のデータ同期を初めて使用するユーザーのために、「*IBM solidDB 拡張レプリケーション・ユーザー・ガイド*」に、solidDB が提供するサンプル・スクリプトの使用方法に関する情報が記載されています。

サンプルの C アプリケーション acsnet.c (ディレクトリー samples/aclib_replication 内) を実行する前に、少なくとも以下のいずれかの作業を行い、solidDB の機能に対する理解を深めておくことをお勧めします。

- solidDB (SMA および LLA なし) を使用して、「*IBM solidDB 拡張レプリケーション・ユーザー・ガイド*」に記載されている SQL スクリプトを実行する。これらのスクリプトは、samples/replication にあります。
- solidDB SMA または LLA を使用して、SQL スクリプトをローカルで実行する。前提条件として、本書の指示に従って、サーバーを始動するようにアプリケーションをセットアップする必要があります。

注: SA API を使用して同期化コマンドを実行することはできません。

- リンク・ライブラリー・アクセス・ライブラリーを伴う実装サンプル・ファイル aclibstandalone.c を実行すると、標準のサーバーがエミュレートされます。このサンプル・ファイルは、ディレクトリー samples/aclib にあります。

上記のどの方法を使用した場合でも、「*IBM solidDB 拡張レプリケーション・ユーザー・ガイド*」の『*データ同期化の概要*』という章に記載されているすべてのステップを solidDB SQL エディター (solsql) で実行できるようになります。

拡張レプリケーション・サンプル・スクリプトによる ODBC アプリケーションのセットアップ

サンプルの C アプリケーション acsNet.c に似た ODBC アプリケーションをビルドして、同期環境をセットアップ、構成、および実行するために必要なすべてのステートメントを実行することができます。acsNet.c は、ディレクトリー samples/aclib_replication にあります。

ODBC クライアント・アプリケーションで使用するサンプル・データベースをセットアップするには、サンプル・スクリプト `replica3.sql`、`replica4.sql`、`replica5.sql`、および `replica6.sql` を実行します。これらのスクリプトはすべて samples/replication/eval_setup ディレクトリーにあります。これらのサンプル・スクリプトには、新しいデータをレプリカに書き込み、同期メッセージの実行を制御する SQL ステートメントが含まれています。スクリプトは、solidDB SQL エディター (solsql) から単独で実行できます。

あるいは、SQL ステートメントを C/ODBC アプリケーションに埋め込んでコンパイルし、リンク・ライブラリー・アクセス・ライブラリーに直接リンクできます。サンプル・スクリプトをリンク・ライブラリー・アクセスにリンクすることで、リンク・ライブラリー・アクセスのアーキテクチャー特有のパフォーマンス上の利益を得られます。

`samples/odbc` ディレクトリーにあるサンプル・プログラム `embed.c` には、リンク・ライブラリー・アクセスを使用して ODBC クライアント・アプリケーションでデータベースをセットアップする方法が示されています。`embed.c` アプリケーションには、`replica3.sql` などのサンプル・スクリプトから SQL コマンドを挿入できます。

5 Java による LLA アプリケーションの作成と実行

Java アプリケーションは、LLA ライブラリーにリンクされ、solidDB SSC API 呼び出しを使用して solidDB を始動および停止します。実際のデータベース接続は、標準の JDBC API を使用して行われます。SolidServerControl API 呼び出しおよび JDBC ドライバーは、いずれも solidDB JDBC ドライバーの .jar ファイル (SolidDriver2.0.jar) 内にあります。

Java を使用する場合の LLA の使用の概要

LLA は、Java アプリケーションがローカルの solidDB サーバーを始動できるようにします。このサーバーは、動的ライブラリーから Java 仮想マシンのコンテキストにロードされます。これで Java アプリケーションは、solidDB サーバーに接続できるようになり、標準の JDBC API を介して solidDB サーバーが提供するサービスを使用できるようになります。動的ライブラリーにリンクすることで、アプリケーションではネットワークを介した RPC (リモート・プロシージャー・コール) のオーバーヘッドを回避できます。

LLA を使用する Java/JDBC プログラムは、LLA ライブラリー (ssolidacxx) にリンクします。このライブラリーには solidDB サーバー全体が、スタンドアロン実行可能プログラムではなく呼び出し可能なライブラリーの形式で格納されています。Java/JDBC で使用されるライブラリーは、C/C++ アプリケーションで使用されるものと同じです。したがって、Java 用の個別バージョンは存在しません。

Java/JDBC で LLA を使用する場合は、以下の要素をリンクして単一の実行可能プロセスにします。

- LLA ライブラリー
- Java 言語のクライアント・プログラム
- JVM

実行可能プロセスでのレイヤーは、上から順に以下のとおりです。

- ローカルの Java/JDBC クライアント・アプリケーション
- JVM (Java 仮想マシン)
- LLA ライブラリー

クライアントの Java コマンドは JVM によって実行されます。コマンドが JDBC 関数呼び出しである場合は、JVM によって ssolidacxx 内の適切な関数が呼び出されます。関数呼び出しは、ネットワーク (RPC) を介さずに直接実行されます。呼び出しには、Java ネイティブ・インターフェース (JNI) が使用されます。ユーザーは JNI コードを作成する必要はなく、リモート・クライアント・プログラムの場合と同じ JDBC 関数を呼び出すだけです。

LLA から solidDB データベースにアクセスする操作は、RPC を介して solidDB データベースにアクセスする操作と同じですが、1 つの例外として、LLA を使用するアプリケーションでは、データベース・サービスにアクセスするために、まず LLA

サーバーを始動する必要があります。この処理には、solidDB Server Control (SSC) API for Java という名前のプロプラエタリー API を使用します (この名前は、SolidServerControl クラスから付けられました)。実際のデータベース接続は、標準の solidDB JDBC API を使用して行われます。SSC API for Java および solidDB JDBC ドライバーは、いずれも `SolidDriver2.0.jar` という .jar ファイル内にあります。

ローカルの solidDB サーバーを始動すると、このサーバーは動的ライブラリーから Java 仮想マシンのコンテキストにロードされます。これで Java アプリケーションは、solidDB サーバーに接続できるようになり、標準の JDBC API を介して、このサーバーが提供するサービスを使用できるようになります。

SMA または LLA を使用するすべてのアプリケーションは、以下の 4 ステップの同じ基本パターンに準拠します。

1. solidDB および接続の設定を構成します。
2. SolidServerControl クラスを使用して、LLA サーバーを始動します。
3. 標準の JDBC API を使用してデータベースにアクセスします。
4. データベースの処理が完了すると、再び SolidServerControl クラスで LLA サーバーを停止します。

制限事項

- すべての solidDB の「`admin command`」は、Java で LLA を使用する場合には使用できません。
- VM コンテキストの外部 (ネイティブ・メソッド呼び出しの内部など) で何らかの障害が発生すると、Java の動作が不安定になります。solidDB サーバーのネイティブ・コードでアサート (またはクラッシュ) が発生した場合、Java は異常終了するか、完全にハングアップします。ハングアップした場合は、付随する Java プロセスを手動で強制終了する必要があります。
- メモリー使用量を最小にするには、割り振られているすべてのステートメントをユーザーが明示的にドロップする必要があります。つまり、割り振られているすべての JDBC ステートメント・オブジェクトを明示的に解放するために、close() メソッドを呼び出す必要があります。これは特に、セットアップに透過接続 (TC) が使用されている場合に重要です。

Java を使用する場合の LLA 用の環境の構成

LLA を Java とともに使用する場合、PATH (Windows の場合) または LD_LIBRARY_PATH (Linux および UNIX の場合) システム変数に、LLA ライブラリーの場所を含める必要があります。

始める前に

solidDB JDBC ドライバーのインストールおよび登録を完了していることが前提となります。

手順

1. **LLA ライブラリーの場所を、PATH (Windows の場合) または LD_LIBRARY_PATH (Linux および UNIX の場合) システム変数に追加します。**
   - Windows 環境では、以下の構文を使用します。

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

   - Linux および UNIX 環境では、以下の構文を使用します。

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

2. **作業ディレクトリー、solidDB データベース、およびユーザー・アカウントを作成して、データベース環境をセットアップします。**

   詳しくは、「*IBM solidDB 管理者ガイド*」の『*データベースの新規作成*』を参照してください。

SSC API for Java による LLA サーバーの始動と停止

Java アプリケーションから solidDB サーバーを始動するには、アプリケーションの最初に SolidServerControl クラスをインスタンス化し、適切なパラメーターを指定して `ssc.startServer` メソッドを呼び出す必要があります。サーバーを始動したら、JDBC でサーバーに接続できるようになります。同様に、サーバーを停止するには、`ssc.stopServer` 呼び出しを使用します。

手順

1. **サーバーの始動**
   - **LLA サーバー:** `ssc.startServer`

   サーバーを始動するとき、solidDB サーバーに少なくとも以下のパラメーターを渡す必要があります。

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

   英大文字と英小文字の「C」があり、それぞれ別のものを指していることに注意してください。

2. **サーバーの停止**
   - **LLA サーバー:** `ssc.stopServer`

LLA の JDBC 接続の作成

solidDBJava アクセラレーターは、ローカル・データベース接続と RPC ベースの接続の両方をサポートします。

ローカルの (RPC ベースでない) JDBC 接続を作成するには、ポート 0 で「localserver」で使用する JDBC ドライバーを指定する必要があります。したがって、例えば JDBC クラス DriverManager を使用してデータベース接続を作成する場合は、以下のステートメントを使用して接続します (この後の SJASample サンプル・コードにも示されています)。

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

このように、DriverManager は URL *"jdbc:solid://localserver:0"* を使用して、ローカル・サーバーとの接続を作成します。getConnection サブルーチンに別の URL を指定すると、ドライバーはおそらく RPC を使用して接続しようとします。

したがって、

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

という URL を、Java アクセラレーター接続の作成時に忘れないようにしてください。

注:

Java アプリケーション内で solidDB リンク・ライブラリー・アクセス・サーバーにアクセスする複数のスレッド (`java.lang.Thread` オブジェクト) を使用する場合は、各スレッドを使用して JDBC 関連のアクティビティーを開始する前に、そのスレッドを個別に solidDB リンク・ライブラリー・アクセス・サーバーに登録する必要があります。スレッドを登録するには、SolidServerControl API の registerThread メソッドをスレッドのコンテキストで呼び出します。スレッドの登録は、ユーザー・スレッド (メイン・スレッドを除く) ごとに solidDB の JDBC ドライバーを使用して明示的に行う必要があります。

また、ユーザーは、solidDB リンク・ライブラリー・アクセス・サーバーに登録されている各スレッドを明示的に登録抹消する必要があります。スレッドの登録を抹消するには、SolidServerControl API の unregisterThread 関数を呼び出します。

サンプル LLA プログラムのコンパイルと実行

このタスクについて

この手順で示される例は、Windows コマンド・プロンプトで使用するためのものです。

手順

1. パスを設定します。

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

   このパスには、solidDB 通信ライブラリーを収容しているディレクトリーも含めるようにしてください。

2. PATH 環境変数に、JDK の HOTSPOT ランタイム環境を組み込みます (SJA は HotSpot JRE でのみテストされています)。

   例えば、以下のように指定します。

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

3. 以下のコマンドを使用して、サンプル `SJASample.java` ファイル (`samples/aclib_java` ディレクトリー内) をコンパイルします。

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

4. 以下のようなコマンド行でサンプル・アプリケーションを実行します。

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

   例えば、サーバーを C:¥soliddb にインストールした場合、SJASample プログラムを実行するには、以下のコマンド行を使用します。

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

   Windows では、ssolidacxx.dll 動的ライブラリーが、solidDB ルート・インストール・ディレクトリーの bin サブディレクトリーにあります。

   SJASample サンプル・クラスのように、SolidServerControl の startServer メソッドで、solidDB サーバーに少なくとも以下のパラメーターを渡す必要があります。

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

   英大文字と英小文字の「C」があり、それぞれ別のものを指していることに注意してください。

5. 必要なすべてのファイル (ssolidacxx ライブラリー、通信ライブラリー、JDBC ドライバー、およびライセンス・ファイル) が現行作業ディレクトリーに配置されていれば、以下のコマンド行で SJASample を開始できます。

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

タスクの結果

LLA サーバーが稼働中になります。

6 ディスクレス機能の使用

SMA および LLA サーバーを使用して、ディスク・ストレージ・スペースなしで稼働するデータベース・エンジンを作成できます。この機能は、ネットワーク・ルーターやスイッチ内のライン・カードのような、ハード・ディスクを持たない組み込みシステムで役立ちます。

ディスクレス・サーバーを稼働するには、単一サーバーとして (単独で) 稼働する方法と、拡張レプリケーション・システムのレプリカとして稼働する方法の 2 つの方法があります。いずれの場合も、SSC API または SSC API for Java の関数呼び出しを使用して、サーバーを始動する必要があります。

SSC API

- ディスクレス SMA サーバーを始動するには、SSCStartDisklessSMAServer を使用します。
- ディスクレス LLA サーバーを始動するには、SSCStartDisklessServer を使用します。

SSC API for Java

- ディスクレス LLA サーバーを始動するには、startDisklessServer を使用します。

ディスクレス・サーバーを単独で使用する場合

ディスクレス・サーバーを単独で稼働する場合、始動時にデータを読み取ることも、シャットダウン時にデータを書き込むこともできません。つまり、サーバーは以前のデータがない状態で毎回始動されます。

サーバーはディスクにデータを書き込むことができないため、サーバーが電源障害などにより正常にシャットダウンされなかった場合、サーバーのデータはすべて失われ、リカバリーできません。データ損失のリスクを軽減するには、solidDB の HotStandby コンポーネントを使用して、データのコピーを格納する「ホット・スタンバイ」マシンを作成します。ホット・スタンバイ機能の詳細については、「*IBM solidDB 高可用性ユーザー・ガイド*」を参照してください。

拡張レプリケーション・システムの一部としてディスクレス・サーバーを使用する場合

ディスクレス・サーバーは、拡張レプリケーション・システムのレプリカとして使用できます。この場合、レプリカはマスター・サーバーにデータを送信し、そのマスター・サーバーからデータをダウンロードできます。したがって、レプリカにディスク・ストレージやその他の専用永続ストレージがなくても、拡張レプリケーション・システム内でそのデータの一部または全部を永続化できます。

7 リモート・アプリケーションまたは二重モード・アプリケーションの作成と実行

SMA および LLA サーバーには、リモート・アプリケーションからアクセスできます。リモート・アプリケーションに SSC API または SA API 関数呼び出しが含まれる場合、個別の SSC API および SA API ライブラリーにリンクする必要があります。これは、SMA および LLA ライブラリーに含まれる関数が、リモート・アプリケーションからアクセスできないためです。リモート・アプリケーションが ODBC または JDBC のみを使用する場合は、ODBC および JDBC インターフェースを使用して、通常どおりにアプリケーションを作成できます。リモート接続タイプは、接続ストリングで定義されます。

例: ODBC および SSC API 関数呼び出しを使用する二重モード LLA アプリケーションの作成

アプリケーションが二重モード・アプリケーションで、かつ SSC API および ODBC 関数呼び出しを使用する場合は、ローカルおよびリモートでそれぞれ実行する 2 種類の実行可能プログラムが必要です。

手順

1. ローカル・モードで実行するバージョンのアプリケーションを作成します。
   a. アプリケーションを LLA ライブラリー (例えば、Windows の場合、`solidimpac.lib`) にリンクします。

      LLA ライブラリーは、ODBC 関数および SSC API 関数の両方をサポートします。
   b. ローカル接続を使用するように接続ストリングを変更します。
2. リモート・モードで実行するバージョンのアプリケーションを作成します。
   a. アプリケーションを solidDB ODBC ドライバーおよび SSC API スタブ・ライブラリー (例えば、Windows の場合、`solidctrlstub.lib`) の両方にリンクします。

      このスタブ・ライブラリーは、実際には、使用するリモート・アプリケーションからのサーバー制御を可能にするものではなく、単に、「未解決シンボル」に関するエラーを出さずにプログラムをコンパイルおよびリンクできるようにするものです。
   b. リモート接続を使用するように接続ストリングを変更します。

リモート接続の確立

リモート接続を確立すると、サーバーに対するアプリケーションの呼び出しは、SMA または LLA ライブラリーへの直接関数呼び出しではなく、ネットワーク経由により実行されます。

ODBC API

ODBC API でリモート接続を確立するには、アプリケーションでリモート・サーバーの名前を指定して SQLConnect 関数を呼び出します。

例

以下の ODBC API コードの例では、ユーザー名 dba とパスワード dba を使用して、リモートの solidDB サーバーに接続します。この例でクライアントとサーバーが使用するネットワーク・プロトコルは、「tcp」(TCP/IP) です。サーバーの名前は「remote_server1」、サーバーが listen するポートは 1313 です。

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

SA API

SA API でリモート接続を確立するには、アプリケーションでリモート・サーバーの名前を指定して SaConnect 関数を呼び出します。

例

この例でクライアントとサーバーが使用するネットワーク・プロトコルは、「tcp」(TCP/IP) です。サーバーの名前は「remote_server1」、サーバーが listen するポートは 1313 です。

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

JDBC

JDBC でリモート接続を確立するには、リモート・サーバーの名前を接続ストリングで定義します。

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

付録 A. 共有メモリー・アクセスのパラメーター

SMA のパラメーターは、solid.ini 構成ファイルの [SharedMemoryAccess] セクションで指定します。

表 17. 共有メモリー・アクセスのパラメーター

[SharedMemoryAccess]	説明	ファクトリー値	開始
MaxSharedMemorySize	このパラメーターは、共有メモリー領域のサイズの最大値を設定します。SMA サーバーがこれより大きなサイズを割り振ろうとすると、「メモリー不足」エラーが発生します。値を「0」にすると、最大値が自動的にコンピューターの物理メモリーのサイズに設定されます (プラットフォーム固有)。	0 (自動)	RW/Startup
SharedMemoryAccessRights	このパラメーターは、共有メモリー領域へのユーザー・アクセスに対する検証コンテキストを設定します。このコンテキストは、従来のファイル検証マスクをモデルにしており、「user」(SMA サーバーを始動したユーザーのみ)、「group」(同じグループに属しているすべてのユーザー)、および「all」(あらゆるユーザー) を構成要素に持ちます。	group	RW/Startup

付録 B. リンク・ライブラリー・アクセスのパラメーター

リンク・ライブラリー・アクセスのパラメーターは、solid.ini 構成ファイルの [Accelerator] セクションで指定します。

表 18. Accelerator パラメーター

[Accelerator]	説明	ファクトリー値
ImplicitStart	yes に設定した場合、このパラメーターは、ODBC API 関数 SQLConnect がユーザー・アプリケーションから呼び出されるとすぐに、solidDB を自動的に始動します。no に設定した場合は、SSC API 関数 SSCStartServer 呼び出しで、solidDB を明示的に始動しなくてはなりません。	yes

付録 C. ディスクレス・サーバー用の構成パラメーター

このセクションでは、ディスクレス・サーバーの実装と保守に関連するパラメーター設定について説明します。

ディスクレス・サーバーで使用されるパラメーター

以下に示す solid.ini 構成ファイルの各セクションには、ディスクレス・サーバー固有の設定があるパラメーターが含まれています。

IndexFile セクション

IndexFile セクションでは、**FileSpec** パラメーターおよび **CacheSize** パラメーターに、ディスクレス・サーバー用の固有の設定があります。

Filespec_[1...n]

FileSpec パラメーターは、データベース・ファイルの名前と最大サイズを指定します。メイン・メモリー・エンジンに対して最大サイズをバイト単位で定義する場合は、FileSpec パラメーターで以下の引数を指定します。

- データベース・ファイル名 - ディスクレス・サーバーでは物理データベース・ファイルが作成されないのでこのパラメーターは使用されませんが、この引数にダミー値を指定する必要があります。
- 最大ファイル・サイズ - この設定は必要です。ディスクレス・サーバー内の全データを格納できるサイズをバイト単位で指定する必要があります。最大ファイル・サイズは、**CacheSize** パラメーターで設定されるキャッシュ・サイズよりも小さくする必要があることに注意してください。

FileSpec パラメーターのデフォルト値は、solid.db、および 2147483647 バイト (2 GB -1) です。以下に例を示します。

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

注: 複数のファイルを指定する場合は、最大ファイル・サイズの設定値が、すべての **FileSpec** パラメーター設定の合計になる必要があります。

ディスクレス・マシンには仮想メモリー用のスワップ・スペースとして使用するディスクがないため、最大サイズは使用可能な物理メモリーによって制限されます。

注: 一部のプラットフォームでは、アプリケーションで使用可能な物理メモリーの量が、マシンの物理メモリーの量より少ない場合があります。

例えば、32 ビット・システムで稼働する Linux の一部のバージョンでは、アプリケーションで使用可能なメモリー量が、理論上のアドレス・スペース (4 GB) の 2 分の 1 または 4 分の 1 に制限されます。これは、アドレスの最上位ビットの 1 つまたは 2 つが Linux 自体のメモリー管理用として確保されるためです。

メモリー内のデータが最大ファイル・サイズを超えると、エラー・メッセージ 11003 が表示されます。

```
File write failed, configuration exceeded
```

CacheSize

CacheSize パラメーターでは、サーバーがバッファー・キャッシュに割り振るメイン・メモリーの量をバイト単位で定義します。例えば、以下のように指定します。

```
CacheSize=10000000
```

この値の設定は、ディスクレス・サーバーに関する以下の基準によって決まります。

- ディスク・ベース表の場合は、キャッシュ・サイズ (バイト単位) を FileSpec パラメーターで設定されている最大ファイル・サイズ (つまりデータの量) よりも少なくとも 20% 大きくする必要があります。これは、このデータがバッファー・キャッシュに保持されるためです。この 20% のオーバーヘッドは見積もりであり、データベースの使用状況によって変わる可能性があります。以下に例を示します。

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

- ディスク・ベース表が使用されない場合でも (インメモリー表を使用してデータベースが作成されている場合)、システム表を保持するためにキャッシュは必要です。その場合、最小キャッシュ・サイズは 1 から 2 MB です。システム表が占有するスペースは、データベース・オブジェクトの数と複雑度、および拡張レプリケーションが使用されているかどうかによって決まります。
- キャッシュ・サイズは、ディスクレス・サーバーの実行に使用できる物理メモリーよりも小さくする必要があります。

  ディスクレス・サーバーが使用する合計メモリー量は、以下のように試算できます (これらすべてを加算した合計は、使用可能な物理メモリー量の範囲内であることが必要です。つまり、キャッシュ・サイズはサーバーで使用可能な物理メモリー量よりも大幅に小さくしなければなりません)。

  ```
  CacheSize
  + 5 MB
  + (100 K * ユーザー数 * ユーザー 1 人あたりのアクティブなステートメント数)
  + インメモリー表のスペース
  + (2 次サーバーに送信される HSB 操作) [1][2]
  ```

  [1] 式のこの項は、HotStandby のユーザーにのみ適用されます。HSB 1 次サーバーは、2 次サーバーに送信される HotStandby 操作を保管するためにある程度のメモリーを必要とします。1 次サーバーと 2 次ディスクレス・サーバーの間で一時的なネットワーク障害が発生すると、1 次サーバーがアプリケーションからのトランザクションを継続して受け付けることがあります。サーバー間のネットワーク接続が復元すると、1 次サーバーから 2 次サーバーに更新情報が送信されます (HotStandby では、トランザクション・ログを使用してこれらの操作が保管されます。ディスクレス・サーバーではトランザクション・ログをディスクに書き込むことができないため、この情報をメモリーに保管する必要があります)。このメモリーはキャッシュとは別のものです。

[2] 式のこの項については、現時点で 1 MB または 512 件の操作 (どちらか小さい方) が上限となっています。ディスク・ベースのサーバーとは異なり、使用可能なスペースを使い切るまでトランザクション・ログが拡大することは許可されていません。

正確な必要量は、サーバーに対して実行される照会の性質をはじめとするその他の要因にも左右されます。物理メモリーはオペレーティング・システムなどによってある程度占有されるので、当然ながらサーバーで使用できるメモリー量は総物理メモリー量よりも少なくなります。

Com セクション

拡張レプリケーション・レプリカ・サーバーとしてディスクレス・サーバーを使用する場合、Com セクションの **Listen** パラメーターが、マスターとディスクレス・レプリカ・サーバー間の通信に影響を及ぼします。

Listen

Listen パラメーターは、ディスクレス・サーバーがネットワークの listen を開始するときに使用するプロトコルと名前を定義します。

以下に例を示します。

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

Listen パラメーターのデフォルト値は `tcp 1964` です。

ネットワーク名およびプロトコルについて詳しくは、「*IBM solidDB 管理者ガイド*」のセクション『*ネットワーク接続の管理*』を参照してください。

ディスクレス・エンジンに適用されない構成パラメーター

セクション別に分類された以下の構成ファイル・パラメーターは、ディスクレス・サーバーには無効であるか、または作用しません。これらのパラメーターは、ディスクレス・エンジンには適用されない動作に影響します。

表 19. ディスクレス・エンジンに適用されない構成パラメーター

パラメーター	説明
[General] セクション	
CheckpointInterval	ディスクレス・サーバーにはチェックポイントが適用されないため、このパラメーターは無効です。
[IndexFile] セクション	
ReadAhead	データベース・ファイルからの物理的な読み取りが行われないため、このパラメーターは作用しません。
PreFlushPercent	データベース・ファイルへの物理的な書き込みが行われないため、このパラメーターは作用しません。

表 19. ディスクレス・エンジンに適用されない構成パラメーター (続き)

パラメーター	説明
[Logging] セクション	
LogEnabled	ディスクレス・サーバーではトランザクション・ロギングが常に無効となるため、このパラメーターは無効です。 **注:** ディスクレス・モードでは、トランザクションのロールバックのみがサポートされます。トランザクション・ロールバックは、一般に何らかの障害によって、途中まで完了したトランザクションが中断された場合に使用されます。ディスクレス・モードでは、ロールフォワード・リカバリーをサポートしません。

付録 D. solidDB サーバー制御 API (SSC API)

solidDB サーバー制御 API (SSC API) は、solidDB のタスク処理システムを簡単かつ効率的に制御する一連の関数です。

注: 関数に関する一部の情報は、SSC API for Java にも当てはまります。SSC API for Java について詳しくは、93 ページの『付録 E. SolidServerControl クラス・インターフェース』のセクションを参照してください。

SSC API の使用

タスク情報の取得

すべてのアクティブ・タスクのリストを取得するには、SSCGetActiveTaskClass 関数を使用します。中断状態にある全タスクのリストを取得するには、SSCGetSuspendedTaskClass 関数を使用します。タスク・クラスの優先順位を取得するには、 SSCGetTaskClassPrio 関数を使用します。

特別なイベントの通知関数

リンク・ライブラリー・アクセスでは、優先タスクを微調整することができます。SSCSetNotifier() 関数を使用することで、特別なイベントが発生した場合に solidDB から指定のユーザー定義関数が呼び出されるように設定できます。この関数で検出される特別なイベントは以下のとおりです。

- solidDB サーバーのシャットダウン
- 索引からストレージ・ツリーへの Bonsai マージ
- Bonsai マージ間隔の最大値
- バックアップまたはチェックポイントの要求
- 活動停止状態のサーバー
- 1 次サーバーから受信したネットコピー要求 (1 次データベースのネットワーク・コピーを 2 次サーバーに送信する要求)
- ネットワーク・コピー (ネットコピー) を通じて受け取った新しいデータベースでサーバーを始動する際に発生したネットコピー要求の完了

solidDB の状況およびサーバー情報の取得

関数 SSCGetStatusNum を使用すると、solidDB データベース・サーバーの現在の状況情報が表示されます。表示される情報は以下のとおりです。

- Bonsai ツリーからストレージ・ツリーにマージされない行数

SSCGetServerHandle 関数は、solidDB サーバーが稼働している場合にそのサーバー・ハンドルを返します。

また、関数 SSCIsRunning を使用してサーバーが稼働しているかどうかを検証することや、関数 SSCIsThisLocalServer を使用して、アプリケーションのリンク先が、

制御 API を使用するリモート・アプリケーションのテストに使用する、ダミーのサーバー・ライブラリー (例えば Windows プラットフォームでの `solidctrlstub.lib`) であるのか、またはローカルのリンク・ライブラリー・アクセス・サーバー・ライブラリー (例えば Windows プラットフォームでの `ssolidacxx.dll`) であるのかを検証することもできます。

SSC API と対応する ADMIN COMMAND

solidDB サーバー制御 API (SSC API) 関数には、対応する solidDB SQL 拡張 ADMIN COMMAND があります。これらのコマンドは、リモート・サイトとローカル・サイトの両方から、solidDB リモート制御 (solcon) や solidDB SQL エディター (solsql) などの solidDB ツールを使用して実行できます。

制御 API に対応する ADMIN COMMAND について詳しくは、55 ページの『付録 B. リンク・ライブラリー・アクセスのパラメーター』を参照してください。

SSC API 関数の要約

solidDB サーバー制御 API (SSC API) 関数の簡単な要約、および『SSC API リファレンス』セクション内のその関数の参照先を以下に示します。

表 20. 制御 API 関数の要約

関数	説明	サポート条件	詳細の参照先
SSCSetCipher	アプリケーションが提供する暗号化ライブラリーを設定します。 **重要:** 推奨されません。代わりに、SSCSetDataCipher を使用してください。	LLA および SSC API スタブ・ライブラリー	69 ページの『SSCSetCipher (非推奨)』
SSCSetDataCipher	アプリケーションが提供する暗号化ライブラリーを設定します。	LLA および SSC API スタブ・ライブラリー	72 ページの『SSCSetDataCipher』
SSCSetDefaultCipher	アプリケーションが提供する暗号化ライブラリーを設定します。	LLA および SSC API スタブ・ライブラリー	75 ページの『SSCSetDefaultCipher』
SSCStartSmaServer	SMA サーバーを始動します。	SMA	87 ページの『SSCStartSMAServer』を参照してください。
SSCStartDisklessSmaServer	ディスクレス SMA サーバーを始動します。	SMA	86 ページの『SSCStartDisklessSMAServer』を参照してください。
SSCStartServer	LLA サーバーを始動します。	LLA および SSC API スタブ・ライブラリー	83 ページの『SSCStartServer』
SSCStartDisklessServer	ディスクレス LLA サーバーを始動します。	LLA および SSC API スタブ・ライブラリー	81 ページの『SSCStartDisklessServer』

表 20. 制御 API 関数の要約 (続き)

関数	説明	サポート条件	詳細の参照先
SSCSetState	solidDB サーバーの状態を設定します (例えば、SSC_STATE_OPEN は後続の接続を許可するかどうかを示します)。状態を ~SSC_STATE_OPEN に設定すると、ローカル接続とリモート接続がブロックされます。	LLA および SSC API スタブ・ライブラリー	80 ページの『SSCSetState』
SSCRegisterThread	リンク・ライブラリー・アクセス・アプリケーションのスレッドをサーバーに登録します。LLA API 関数を呼び出すには、事前にユーザー・アプリケーションのすべてのスレッドで登録を行う必要があります。	LLA および SSC API スタブ・ライブラリー	68 ページの『SSCRegisterThread』
SSCUnregisterThread	サーバーに対する LLA アプリケーション・スレッドの登録を抹消します。登録抹消は、登録されているすべてのスレッドで終了前に行う必要があります。	LLA および SSC API スタブ・ライブラリー	90 ページの『SSCUnregisterThread』
SSCStopServer	SMA または LLA サーバーを停止します。	SMA、LLA、および SSC API スタブ・ライブラリー	89 ページの『SSCStopServer』
SSCSetNotifier	指定されたイベント (マージ、バックアップ、シャットダウンなど) で solidDB が呼び出すユーザー定義関数を指定します。	LLA および SSC API スタブ・ライブラリー	77 ページの『SSCSetNotifier』
SSCIsRunning	サーバーが稼働していない場合はゼロ以外の値を返します。	LLA および SSC API スタブ・ライブラリー	68 ページの『SSCIsRunning』
SSCIsThisLocalServer	アプリケーションが、LLA を使用する solidDB サーバーにリンクされているかどうか、または SSC API を使用する solidDB リモート・アプリケーションをテストするための「ダミー」ライブラリー (`solidctrlstub`) にリンクされているかどうかを示します。	LLA および SSC API スタブ・ライブラリー	68 ページの『SSCIsThisLocalServer』
SSCGetServerHandle	solidDB サーバーが稼働している場合は、そのサーバー・ハンドルを返します。	LLA および SSC API スタブ・ライブラリー	67 ページの『SSCGetServerHandle』
SSCGetStatusNum	solidDB の状況情報を取得します。	LLA および SSC API スタブ・ライブラリー	67 ページの『SSCGetStatusNum』

SSC API リファレンス

SSC API リファレンスでは、各 SSC API 関数をアルファベット順に説明します。各説明では、目的、構文、パラメーター、戻り値、およびコメントが示されます。

- 『関数の構文』
- 『パラメーター』
- 65 ページの『戻り値』
- 66 ページの『SSC API のエラー・コードとメッセージ』

関数の構文

関数の宣言構文は以下のとおりです。

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

ReturnType は変動しますが、通常は呼び出しが成功したか失敗したかを示す値です。戻り値について詳しくは、このセクションで後述します。

SSC_CALL は移植性を確保するために必要です。SSC_CALL によって、関数の呼び出し規則を指定します。SSC_CALL は、`sscapi.h` ファイルで各プラットフォームに合わせて定義されます。

パラメーターは斜体で示されます。

パラメーター

各関数の説明では、パラメーターが表形式で説明されています。この表には、パラメーターの一般的な使用タイプ (以下で説明) および各関数でのパラメーター変数の用途が示されます。

パラメーターの使用タイプ

以下の表は、SSC API パラメーターの想定される使用タイプを示しています。パラメーターがポインターとして使用される場合は、呼び出し後のパラメーター変数の所属を指定する 2 番目の使用カテゴリーがパラメーターに追加されることに注意してください。

表 21. SSC API パラメーターの使用タイプ

使用タイプ	**意味**
in	パラメーターが入力であることを示します。
output	パラメーターが出力であることを示します。
in out	パラメーターが入出力であることを示します。

表 21. SSC API パラメーターの使用タイプ (続き)

使用タイプ	意味
use	ポインター・パラメーターにのみ適用されます。パラメーターが単に関数呼び出しで使用されることを意味します。呼び出し元は、関数呼び出し後にこのパラメーターを使用して任意の操作を実行できます。これは、パラメーター引き渡しのタイプとして最もよく使用されます。
take	ポインター・パラメーターにのみ適用されます。パラメーター値が関数で使用されることを意味します。呼び出し元は、関数呼び出し後にこのパラメーターを参照できません。パラメーターが不要となったときは、関数または関数で作成されたオブジェクトがそのパラメーターを解放します。

hold	ポインター・パラメーターにのみ適用されます。関数が関数呼び出し後もパラメーター値を保持することを意味します。呼び出し元は、関数呼び出し後もパラメーター値を参照でき、パラメーターの解放も担当します。
	
	重要:
	
	このパラメーターはユーザーとサーバーが共有するため、サーバーがパラメーターの処理を終えるまでユーザーはパラメーターを解放できません。一般に、保持されているオブジェクトを解放するには、保持しているオブジェクトを先に解放する必要があります。例えば、以下のように指定します。
	`conn = SaConnect("", "dba", "dba");`
	`/* 接続はカーソルが解放されるまで保持されます */`
	`scur = SaCursorCreate(conn, "mytable");`
	`...`
	`SaCursorFree(scur);`
	`/* カーソルを解放した後は、接続を安全に解放 */`
	`/* できます (あるいは、この例のように接続を */`
	`/* 解放するサーバー関数を呼び出します)。 */`
	`SaDisconnect(conn);`

戻り値

各関数の説明では、関数が値を返すかどうか、および返される値のタイプを示します。

SscTaskSetT

関数から SscTaskSetT タイプの値が返された場合、この定義はビット・マスクとして使用されます。SScTaskSetT は、sscapi.h で以下の値を使用して定義されます。

```
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 (上記の全タスク)
```

HotStandby の「ネットコピー」操作と「コピー」操作は、タスク SSC_TASK_BACKUP によって実行されることに注意してください。SSC_TASK_NETCOPY というタスクは存在しません。

SSC API のエラー・コードとメッセージ

SSC API 関数から、以下の表に記載されているエラー・コードとメッセージが返される場合があります。

これらの定数は、sscapi.h ファイルで定義されています。

表 22. SSC API 関数のエラー・コードとメッセージ

エラー・コード/メッセージ	説明
SSC_SUCCESS	操作が正常に終了しました。
SSC_ERROR	一般エラー。
SSC_ABORT	操作が異常終了しました。
SSC_FINISHED	すべてのタスクが実行されると、SSCAdvanceTasks からこのメッセージが返されます。
SSC_CONT	実行するタスクがまだ残っている場合に、SSCAdvanceTasks からこのメッセージが返されます。
SSC_CONNECTIONS_EXIST	開いている接続が存在します。
SSC_UNFINISHED_TASKS	未完了のタスクが存在します。
SSC_INFO_SERVER_RUNNING	サーバーは既に稼働しています。
SSC_INVALID_HANDLE	無効なローカル・サーバー・ハンドルが指定されました。このサーバーは、SSCStartServer で始動したサーバーと一致しません。
SSC_INVALID_LICENSE	ライセンスがないか、無効なライセンス・ファイルが見つかりました。
SSC_NODATABASEFILE	データベース・ファイルが見つかりません。
SSC_SERVER_NOTRUNNING	サーバーが稼働していません。
SSC_SERVER_INNETCOPYMODE	サーバーがネットコピー・モードです (高可用性/HotStandby を使用する場合のみ)。

SSCGetServerHandle

SSCGetServerHandle は、solidDB サーバーが稼働している場合に、そのサーバー・ハンドルを返します。

構文

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

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

戻り値

- サーバーが稼働していない場合は NULL。
- サーバーが稼働している場合はサーバー・ハンドル。

SSCGetStatusNum

SSCGetStatusNum は、solidDB の状況情報を取得します。

構文

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

SSCGetStatusNum 関数で使用されるパラメーターは以下のとおりです。

表 23. SSCGetStatusNum のパラメーター

パラメーター	使用タイプ	説明
h	in、use	サーバーへのハンドル。
stat	in	取得の対象となる状況 ID を指定します。
num	out	関数が正常に終了すると、関数が戻る際にこのパラメーターの値がマージされなかった書き込みの数またはサーバー・スレッドの数に設定されます。どちらに設定されるかは、要求された情報によって決まります。

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

SSCGetStatusNum を呼び出す際に stat パラメーターに認識不能な値を指定すると、この関数から SSC_SUCCESS が返されます。

戻り値

- SSC_SUCCESS - 操作が正常に終了しました。この値は、stat パラメーターに無効な値を指定したときにも返されます。
- SSC_ERROR - 操作が失敗しました。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。

- SSC_SERVER_NOTRUNNING - サーバーが稼働していません。

SSCIsRunning

SSCIsRunning は、サーバーが稼働している場合にゼロ以外の値を返します。

構文

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

SSCIsRunning 関数で使用されるパラメーターは以下のとおりです。

表 24. SSCIsRunning のパラメーター

パラメーター	使用タイプ	説明
h	in、use	サーバーへのハンドル

戻り値

- 0 - サーバーが稼働していません。
- ゼロ以外 - サーバーは稼働しています。

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

SSCIsThisLocalServer

SSCIsThisLocalServer は、アプリケーションが solidDB サーバーまたは「ダミー」ライブラリー (solidctrlstub) にリンクされているかどうかを示します。solidctrlstub ライブラリーを開発時に使用すると、制御 API を使用する solidDB リモート・アプリケーションを、リンク・ライブラリー・アクセス・ライブラリーにリンクすることなく、またソース・コードを変更することなくテストできます。

構文

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

戻り値

- 0 - アプリケーションは solidDB サーバーにリンクされていません。
- 1 - アプリケーションは solidDB サーバーにリンクされています。

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

SSCRegisterThread

SSCRegisterThread は、solidDB アプリケーション・スレッドをサーバーに登録します。制御 API、ODBC API、または SA API を使用するスレッドは、すべて登録する必要があります。SSCRegisterThread 関数は、他のリンク・ライブラリー・アクセス API 関数を使用する前にスレッドで呼び出す必要があります。

アプリケーションにスレッドが 1 つしかない場合 (メイン・スレッド)、つまりアプリケーション自体がスレッドを作成しない場合、登録は必要ありません。

スレッドは、終了前に SSCUnregisterThread 関数を呼び出して自身の登録を抹消する必要があります。

構文

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

SCCRegisterThread 関数で使用されるパラメーターは以下のとおりです。

表 25. SCCRegisterThread のパラメーター

パラメーター	使用タイプ	説明
h	In、Use	サーバーへのハンドル。

戻り値

- SSC_SUCCESS
- SSC_INVALID_HANDLE

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

関連項目

SSCUnregisterThread

SSCSetCipher (非推奨)

重要: この関数は推奨されません。代わりに、 72 ページの『SSCSetDataCipher』を使用してください。

SSCSetCipher 関数は、アプリケーションが提供する暗号および暗号化/暗号化解除関数を設定します。提供された暗号は、SSCStartServer で関連するデータベース暗号化コマンド行引数が使用されるときに自動的に使用されます。

構文

```
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* のパラメーター

パラメーター	使用タイプ	説明
cipher	in	アプリケーション固有の暗号オブジェクト (暗号化パスワードなど) へのポインター。solidDB サーバーがこのポインターを使用したり解釈したりすることはありません。アプリケーションが提供する暗号化/暗号化解除関数にこのポインターを渡すだけです。
encrypt	in	アプリケーションが提供する暗号化関数へのポインター。この関数は、サーバーがデータベース・ファイルまたはログ・ファイルのページを暗号化する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • cipher - アプリケーションが提供する暗号オブジェクトへのポインター。 • page_no - サーバーが提供するページ番号。暗号化アルゴリズムでは、この情報を無視するか、暗号鍵の一部として使用できます。 • n - 暗号化するページ数。 • pagesize - 暗号化するページのサイズ。 • page - 暗号化されるデータ・バッファーへのポインター。データ・バッファーのサイズは以下のとおりです。 `n*pagesize` この関数は、*n*pagesize* サイズのファイルに書き込まれる暗号化されたデータ・バッファーへのポインターを返す必要があります。 暗号化されたデータ・バッファーへのポインターをサーバーが解放することはありません。 関数に page パラメーターとして渡されたデータ・バッファーは、暗号化関数で自由に上書きまたは操作することができます。例えば、暗号化関数でデータを適切に暗号化し、page ポインターを返すことができます。

表 26. *SSCSetCipher* のパラメーター (続き)

パラメーター	使用タイプ	説明
decrypt	in	アプリケーションが提供する暗号化解除関数へのポインター。この関数は、サーバーが暗号化されたデータベースまたはログ・ファイルの一部を読み取って暗号化解除する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • cipher - アプリケーションが提供する暗号オブジェクトへのポインター。 • page_no - サーバーが提供するページ番号。暗号化解除アルゴリズムでは、この情報を無視するか、暗号化解除鍵の一部として使用できます。 • n - 暗号化解除するページ数。 • pagesize - 暗号化解除するページのサイズ。 • page - 暗号化解除されるデータ・バッファーへのポインター。データ・バッファーのサイズは以下のとおりです。 *n*pagesize*

戻り値

SSCSetCipher 関数は値を返しません。この関数は、SSCStartServer 関数を使用してリンク・ライブラリー・アクセス・サーバーを始動する前に呼び出す必要があります。

コメント

暗号化解除関数は、ページの暗号化解除に成功した場合はゼロ以外の値を返し、何らかの理由で暗号化解除に失敗した場合は 0 値を返すようになっています。0 値が返された場合、サーバーは処理を続行できなくなるので緊急時シャットダウンを実行します。この関数は、'page' パラメーターで指定された同じバッファーに暗号化データを返すようになっています。

リンク・ライブラリー・アクセス暗号化 API の使用

以下のコードは、AcceleratorLib 暗号化 API の使用法を示しています。暗号化方式には、一般的な XOR 難読化が使用されます。

```
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 関数は、アプリケーションが提供する暗号および暗号化/暗号化解除関数を設定します。提供された暗号は、SSCSetDataCipher で関連するデータベース暗号化コマンド行引数が使用されるときに自動的に使用されます。

構文

```
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 パラメーター

パラメーター	使用タイプ	説明
cipher	in	アプリケーション固有の暗号オブジェクト (暗号化パスワードなど) へのポインター。solidDB サーバーがこのポインターを使用したり解釈したりすることはありません。アプリケーションが提供する暗号化/暗号化解除関数にこのポインターを渡すだけです。

表 27. *SSCSetDataCipher* パラメーター (続き)

パラメーター	使用タイプ	説明
encrypt	in	アプリケーションが提供する暗号化関数へのポインター。この関数は、サーバーがデータベース・ファイルまたはログ・ファイルのページを暗号化する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • cipher - アプリケーションが提供する暗号オブジェクトへのポインター。 • page_no - サーバーが提供するページ番号。暗号化アルゴリズムでは、この情報を無視するか、暗号鍵の一部として使用できます。 • n - 暗号化するページ数。 • pagesize - 暗号化するページのサイズ。 • page - 暗号化されるデータ・バッファーへのポインター。データ・バッファーのサイズは以下のとおりです。 `n*pagesize` この関数は、*n*pagesize* サイズのファイルに書き込まれる暗号化されたデータ・バッファーへのポインターを返す必要があります。 暗号化されたデータ・バッファーへのポインターをサーバーが解放することはありません。 関数に page パラメーターとして渡されたデータ・バッファーは、暗号化関数で自由に上書きまたは操作することができます。例えば、暗号化関数でデータを適切に暗号化し、page ポインターを返すことができます。

表 27. *SSCSetDataCipher* パラメーター *(続き)*

パラメーター	使用タイプ	説明
decrypt	in	アプリケーションが提供する暗号化解除関数へのポインター。この関数は、サーバーが暗号化されたデータベースまたはログ・ファイルの一部を読み取って暗号化解除する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • cipher - アプリケーションが提供する暗号オブジェクトへのポインター。 • page_no - サーバーが提供するページ番号。暗号化解除アルゴリズムでは、この情報を無視するか、暗号化解除鍵の一部として使用できます。 • n - 暗号化解除するページ数。 • pagesize - 暗号化解除するページのサイズ。 • page - 暗号化解除されるデータ・バッファーへのポインター。データ・バッファーのサイズは以下のとおりです。 *n*pagesize*

戻り値

SSCSetDataCipher 関数は、値を返しません。この関数は、SSCStartServer 関数を使用してリンク・ライブラリー・アクセス・サーバーを始動する前に呼び出す必要があります。

コメント

暗号化解除関数は、ページの暗号化解除に成功した場合はゼロ以外の値を返し、何らかの理由で暗号化解除に失敗した場合は 0 値を返すようになっています。0 値が返された場合、サーバーは処理を続行できなくなるので緊急時シャットダウンを実行します。この関数は、'page' パラメーターで指定された同じバッファーに暗号化データを返すようになっています。

リンク・ライブラリー・アクセス暗号化 API の使用

以下のコードは、AcceleratorLib 暗号化 API の使用法を示しています。暗号化メソッドには、XOR 難読化が使用されます。

```
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 関数は、アプリケーションが提供する暗号および暗号化/暗号化解除関数を設定します。提供された暗号は、SSCSetDefaultCipher で関連するデータベース暗号化コマンド行引数が使用されるときに自動的に使用されます。

構文

```
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 パラメーター

パラメーター	使用タイプ	説明
cipher	in	アプリケーション固有の暗号オブジェクト (暗号化パスワードなど) へのポインター。solidDB サーバーがこのポインターを使用したり解釈したりすることはありません。アプリケーションが提供する暗号化/暗号化解除関数にこのポインターを渡すだけです。

表 28. *SSCSetDefaultCipher* パラメーター *(続き)*

パラメーター	使用タイプ	説明
encrypt	in	アプリケーションが提供する暗号化関数へのポインター。この関数は、サーバーがパスワード、データベース・ファイルまたはログ・ファイルのページを暗号化する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • `cipher` - cipher は、暗号化パスワードなど、アプリケーションが提供するセキュリティー・コンテキスト (cipher オブジェクト) を参照します。同じパラメーターが、アプリケーションが提供する暗号化/暗号化解除関数に渡されます。 • `encrypt` - 暗号化関数 • `data` - アプリケーション関数で暗号化されるデータ。このパラメーターは、暗号化されたデータの入出力に使用されます。 • `datalen` - 暗号化/暗号化解除されるデータの長さ。
decrypt	in	アプリケーションが提供する暗号化解除関数へのポインター。この関数は、サーバーがパスワード、データベース、またはログ・ファイルの暗号化された部分を読み取って、それを暗号化解除する必要があるときに呼び出されます。この関数のパラメーターは以下のとおりです。 • `cipher` - cipher は、暗号化パスワードなど、アプリケーションが提供するセキュリティー・コンテキスト (cipher オブジェクト) を参照します。同じパラメーターが、アプリケーションが提供する暗号化/暗号化解除関数に渡されます。 • `decrypt` - 暗号化解除関数 • `data` - アプリケーション関数で暗号化解除されるデータ。このパラメーターは、暗号化解除されたデータの入出力に使用されます。 • `datalen` - 暗号化解除されるデータの長さ。

戻り値

SSCSetDefaultCipher 関数は、値を返しません。この関数は、SSCStartServer 関数または SSCStartSMAServer 関数を使用して、リンク・ライブラリー・アクセス・サーバーを始動する前に呼び出す必要があります。

コメント

暗号化解除関数は、ページの暗号化解除に成功した場合はゼロ以外の値を返し、何らかの理由で暗号化解除に失敗した場合は 0 値を返すようになっています。0 値が返された場合、サーバーは処理を続行できなくなるので緊急時シャットダウンを実行します。この関数は、'page' パラメーターで指定された同じバッファーに暗号化データを返すようになっています。

暗号化 API の使用

以下のコードは、カスタム暗号化 API の使用方法を示しています。暗号化メソッドには、XOR 難読化が使用されます。

```
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 は、リンク・ライブラリー・アクセス・サーバーが始動または停止されるときに呼び出すコールバック関数を設定します。この関数には対応する ADMIN COMMAND がありません。

構文

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

SSCSetNotifier 関数で使用されるパラメーターは以下のとおりです。

表 29. SSCSetNotifier のパラメーター

パラメーター	使用タイプ	説明
h	in	サーバーへのハンドル。

表 29. *SSCSetNotifier* のパラメーター (続き)

パラメーター	使用タイプ	説明
what	in	通知の対象となるイベントを指定します。オプションは以下のとおりです。 • SSC_NOTIFY_EMERGENCY_EXIT サーバーが SSCStartServer() で活動化された後にクラッシュした場合に、この関数が呼び出されます。 SSCStartServer() の前に、通知関数の呼び出し SSCSetNotifier() を実行する必要があります。 • SSC_NOTIFY_SHUTDOWN シャットダウン時に関数が呼び出されます。 • SSC_NOTIFY_SHUTDOWN_REQUEST サーバーがシャットダウン要求を受信すると関数が呼び出され、ユーザー定義関数がその要求を受け付けた場合にシャットダウンを行います。通知を受けた関数から SSC_ABORT を返すことでシャットダウンを拒否できます。要求の処理に進む場合は SSC_CONT を返します。 • SSC_NOTIFY_ROWSTOMERGE Bonsai 索引ツリーにストレージ・サーバーにマージする必要があるデータがある場合に、関数が呼び出されます。 • SSC_NOTIFY_MERGE_REQUEST solid.ini 構成ファイルの MergeInterval パラメーターの設定を超過し、マージを開始する必要がある場合に、関数が呼び出されます。 • SSC_NOTIFY_BACKUP_REQUEST バックアップが要求されたときに関数が呼び出されます。バックアップを拒否するには、通知を受けた関数から SSC_ABORT を返します。 • SSC_NOTIFY_CHECKPOINT_REQUEST チェックポイントが要求されたときに関数が呼び出されます。チェックポイントを拒否するには、通知を受けた関数から SSC_ABORT を返します。 • SSC_NOTIFY_IDLE サーバーがアイドル状態に切り替えられたときに関数が呼び出されます。 • SSC_NOTIFY_NETCOPY_REQUEST このコールバック関数は HotStandby コンポーネントのみに適用されます。次サーバーからネットコピー要求 (1 次データベースのネットワーク・コピーを 2 次サーバーに送信する要求) を受信したときに、関数が呼び出されます。netcopy コマンドの詳細については、「*IBM solidDB 高可用性ユーザー・ガイド*」を参照してください。 • SSC_NOTIFY_NETCOPY_FINISHED このコールバック関数は HotStandby コンポーネントのみに適用されます。ネットコピー要求が完了したときに関数が呼び出されます。完了すると、ネットワーク・コピー (ネットコピー) を通じて受け取った新しいデータベースでサーバーが始動され、SSC_NOTIFY_FINISHED が呼び出されてサーバーが再び使用可能になったことがアプリケーションに通知されます。
notify_fun_handler	in、hold	呼び出すユーザー関数。

表 29. *SSCSetNotifier* のパラメーター *(続き)*

パラメーター	使用タイプ	説明
userdata	in、hold	通知関数に渡されるユーザー・データ。 64 ページの『パラメーター』に記載されている、使用タイプ *hold* のパラメーターの解放に関する注意事項を読んでおいてください。

戻り値

- SSC_SUCCESS - サーバーの要求が受け付けられました。

  HotStandby のみ:

  SSC_NOTIFY_NETCOPY_FINISHED が SSC_SUCCESS を返した場合は、他のアプリケーション接続がすべて終了し、サーバーが「ネットコピー listen モード」に設定されます。このモードでは、サーバーが 1 次サーバーからの接続を受け付けます。また、2 次サーバーでは、HotStandby の netcopy コマンドからデータを受け取ることだけが可能となります。「ネットコピー listen モード」の詳細については、「*IBM solidDB 高可用性ユーザー・ガイド*」を参照してください。 (以前は「ネットコピー listen モード」が「バックアップ listen モード」とも呼ばれていたことに注意してください)。
- SSC_ABORT - サーバーの要求が拒否されました。

  HotStandby のみ:

  SSC_NOTIFY_NETCOPY_REQUEST が SSC_ABORT を返した場合は、ネットコピーが開始されず、エラー・コード (SRV_ERR_OPERATIONREFUSED) が 1 次サーバーに返されます。
- SSC_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。

  SSC_SERVER_NOTRUNNING - サーバーが稼働していません。

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

使用タイプ *hold* のパラメーターを保留解除する場合は注意が必要です。 64 ページの『パラメーター』の hold に関する注意事項を参照してください。

ユーザー定義の通知関数では、SA、SSC、ODBC の各関数を呼び出さないようにする必要があります。

ユーザー定義の通知関数を作成するときは、以下のプロトタイプに準拠する必要があります。

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

SSC_CALL を使用してユーザー関数の規則を明示的に定義したら、SSCSetNotifier 関数を使用してその関数を登録して、その関数が指定のイベントで呼び出されるようにします。以下に例を示します。

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

例

シャットダウン時の関数の呼び出し

シャットダウンが要求されるたびに呼び出される user_own_shutdownrequest という関数をユーザーが作成したとします。

```
int SSC_CALL user_own_shutdownrequest(SscServerT h, SscNotFunT what, void
    *userdata);
{
  if (shutdown not needed) {
    return SSC_ABORT;
  }
  return SSC_CONT; /* シャットダウンに進む */
}
```

サーバーがシャットダウンされる前に user_own_shutdownrequest が呼び出されるように指定するには、SSCSetNotifier 関数を次のように呼び出します。

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

注:

関数 user_own_shutdownrequest から SSC_ABORT が返された場合はシャットダウンが許可されず、SSC_CONT が返された場合はシャットダウンに進むことができます。

SSCSetState

SSCSetState は、リンク・ライブラリー・アクセス・サーバーの状態を設定します。これにより、サーバーが後続の接続を受け付けるかどうか、またサーバーがプリフェッチを使用するかどうかを制御できます。

サーバーを「open」に設定すると、サーバーは接続を受け付けるようになります。サーバーを「closed」に設定すると、サーバーは以降の接続を受け付けなくなります (これにはローカル接続とリモート接続の両方が該当します)。ただし、既に作成されている接続は引き続き使用できます。

プリフェッチをオンにすると、サーバーはすぐに参照される可能性が高いデータを「先読み」によってフェッチします。プリフェッチには、追加のメモリーまたはディスク・キャッシュ・スペースが必要です。プリフェッチをオンにすると、一般にはパフォーマンスが向上します。プリフェッチをオフにすると、必要なメモリーが少なくなります。サーバーの大規模な順次スキャンを必要とする照会がある場合は、プリフェッチをオンにすると非常に役立ちます。例えば、レポートまたは集約関数を使用してデータベース全体 (またはその大部分) の値を取得する場合は、プリフェッチが役立ちます。すべての照会が 1 件または数件のレコードしか必要としないのであれば、通常プリフェッチは役立ちません。プリフェッチはメモリーを使い果たすため、使用可能メモリーの少ないシステムでは、プリフェッチによって実際にパフォーマンスが低下することがあります。

プリフェッチをいつ使用すればよいかを判断するには、以下のガイドラインが役立ちます。

プリフェッチの使用が適切である場合: 使用可能メモリー (またはディスク・キャッシュ・スペース) が大量にあり、照会に大規模な順次スキャンが必要である。

プリフェッチの使用が適切でない場合: 使用可能メモリーが少なく、ほとんどの照会で 1 度に 1 回無関係なレコードが読み取られる。

構文

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

SSCSetState 関数で使用されるパラメーターは以下のとおりです。

表 30. SSCSetState のパラメーター

パラメーター	使用タイプ	説明
h	in、use	サーバーへのハンドル。
runflags	in	SSC_STATE_OPEN フラグ (新たなリモート接続を許可) および SSC_STATE_PREFETCH フラグ (必要に応じてユーザーがサーバーにプリフェッチの実行を許可) の組み合わせを指定できます。指定可能な組み合わせの例を以下に示します。 • サーバーを open に設定: state = state | SSC_STATE_OPEN; • サーバーを closed に設定: state = state & ~SSC_STATE_OPEN; • プリフェッチを on に設定: state = state | SSC_STATE_PREFETCH; • プリフェッチを off に設定: state = state & ~SSC_STATE_PREFETCH;

戻り値

- SSC_SUCCESS - 操作が正常に終了しました。
- SSC_ERROR - 操作が失敗しました。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。
- SSC_SERVER_NOTRUNNING - サーバーが稼働していません。

コメント

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND があります。以下のコマンドです。

ADMIN COMMAND 'close';

SSCStartDisklessServer

SSCStartDisklessServer は、リンク・ライブラリー・アクセスを使用するディスクレス・サーバーを始動します。

構文

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

SSCStartDisklessServer 関数で使用されるパラメーターは以下のとおりです。

表 31. SSCStartDisklessServer のパラメーター

パラメーター	使用タイプ	説明
argc	in	コマンド行引数の数。
argv	in、use	関数呼び出しで使用されるコマンド行引数の配列。引数 argv[0] は、ユーザー・アプリケーションのパスとファイル名用として予約されており、必ず指定する必要があります。 使用可能な引数のリストについては、「*IBM solidDB 管理者ガイド*」のセクション『*solidDB*コマンド行オプション』を参照してください。
h	out	始動されたサーバーへのハンドルを返します。このハンドルは、他の制御 API 関数でサーバーを参照する際に必要となります。
runflags	in	このパラメーターには以下のオプションのみを指定できます。 SSC_STATE_OPEN - リモート接続が許可されます。 runflags = SSC_STATE_OPEN
lic_string	in	solidDB ライセンス・ファイルを含んだストリングを指定します。
ini_string	in	solidDB 構成ファイルを含んだストリングを指定します。

戻り値

- SSC_SUCCESS - サーバーが始動されました。
- SSC_ERROR - サーバーの始動に失敗しました。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。
- SSC_INFO_SERVER_RUNNING - サーバーは既に稼働しています。
- SSC_INVALID_HANDLE - 無効なローカル・サーバー・ハンドルが指定されました。
- SSC_INVALID_LICENSE - ライセンスがないか、無効なライセンス・ファイルが見つかりました。

コメント

デフォルトでは、状態は SSC_STATE_OPEN に設定されます。

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

例

SSCStartDisklessServer

```
SscStateT runflags = SSC_STATE_OPEN;
SscServerT h;
char* argv[4]; /* 4 つのパラメーター・ストリングへのポインター */
int argc = 4;
char* lic = get_lic(); /* ライセンスを取得 */
char* ini = get_ini(); /* solid.ini を取得 */
SscRetT rc;
argv[0] = "appname"; /* ユーザー・アプリケーションのパスとファイル名 */
argv[1] = "-Udba"; /* ユーザー名 */
argv[2] = "-Pdba"; /* ユーザーのパスワード */
argv[3] = "-Cdba"; /* カタログ名 */
/* ディスクレス・サーバーを始動 */
rc = SSCStartDisklessServer(argc, argv, &h, runflags, lic, ini);
```

注:

この例の get_ini() と get_lic() は、ユーザーが作成する関数です。それぞれの関数で、solid.ini ファイル・テキストまたは solid.lic ライセンス・ファイルを含んだストリングを返す必要があります。

カタログ名を指定しないと、solidDB からエラーが返されます。

関連項目

SSCStopServer

49 ページの『6 章 ディスクレス機能の使用』も参照してください。

SSCStartServer

SSCStartServer は、リンク・ライブラリー・アクセスを開始します。マルチスレッド環境では、サーバーがクライアントとは別のスレッドで稼働します。アプリケーションが実行されている間は、アプリケーションで必要に応じてサーバー・サブルーチンを開始または停止できます。

3 番目のパラメーターは「out」パラメーターであることに注意してください。サーバーが正常に始動されると、SSCStartServer ルーチンはそのサーバーのハンドルを指すようにこのパラメーターを設定します。

注:

ディスクレス・サーバーを始動する場合は、制御 API 関数 SSCStartDisklessServer を使用する必要があります。 81 ページの『SSCStartDisklessServer』を参照してください。

構文

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

SSCStartServer 関数で使用されるパラメーターは以下のとおりです。

表 32. SSCStartServer のパラメーター

パラメーター	使用タイプ	説明
argc	in	コマンド行引数の数。
argv	in、use	コマンド行引数の配列。 使用可能な引数のリストについては、「*IBM solidDB 管理者ガイド*」のセクション『*solidDB*コマンド行オプション』を参照してください。
h	out	始動されたサーバーへのハンドルを返します。このハンドルは、他の制御 API 関数でサーバーを参照する際に必要となります。
runflags	in	以下のいずれか、または両方を指定できます。 • SSC_STATE_OPEN - リモート接続が許可されます。 • SSC_STATE_PREFETCH - サーバーが必要に応じてプリフェッチを実行します。 例えば、以下のように指定します。 `runflags = SSC_STATE_OPEN & SSC_STATE_PREFETCH`

例: SSCStartServer の始動

サーバー名、カタログ名、および管理者のユーザー名とパスワードを指定して SSCStartServer を開始します。

```
SscStateT runflags = SSC_STATE_OPEN; SscServerT h; char* argv[5];
argv[0] = "appname"; /* ユーザー・アプリケーションのパスとファイル名 */
argv[1] = "-nsolid1"; argv[2] = "-UDBA" argv[3] = "-PDBA";
argv[4] = "-CDBA"; /* サーバーの始動 */ rc =
SSCStartServer(argc, argv, &h, run_flags);
```

注: データベースが既にある場合は、ユーザー名とパスワード、またはカタログ名を指定する必要はありません。

戻り値

- SSC_SUCCESS - サーバーが始動されました。
- SSC_ERROR - サーバーの始動に失敗しました。
- SSC_ABORT
- SSC_BROKENNETCOPY - 不完全なネットコピーが原因でデータベースが破損しました。
- SSC_FINISHED
- SSC_CONT
- SSC_CONNECTIONS_EXIST

- SSC_UNFINISHED_TASKS
- SSC_INVALID_HANDLE - 無効なローカル・サーバー・ハンドルが指定されました。
- SSC_INVALID_LICENSE - ライセンスがないか、無効なライセンス・ファイルが見つかりました。
- SSC_NODATABASEFILE - データベース・ファイルが見つかりませんでした。
- SSC_SERVER_NOTRUNNING
- SSC_INFO_SERVER_RUNNING - サーバーは既に稼働しています。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。
- SSC_DBOPENFAIL - データベースを開く操作に失敗しました。
- SSC_DBCONNFAIL - データベースへの接続に失敗しました。
- SSC_DBTESTFAIL - データベース・テストが失敗しました。
- SSC_DBFIXFAIL - データベースの修正に失敗しました。
- SSC_MUSTCONVERT - データベースを変換する必要があります。
- SSC_DBEXIST - データベースが存在します。
- SSC_DBNOTCREATED - データベースが作成されていません。
- SSC_DBCREATEFAIL - データベースの作成に失敗しました。
- SSC_COMINITFAIL - 通信の開始に失敗しました。
- SSC_COMLISTENFAIL - 通信の listen に失敗しました。
- SSC_SERVICEFAIL - サービスの操作に失敗しました。
- SSC_ILLARGUMENT - コマンド行引数が正しくありません。
- SSC_CHDIRFAIL - ディレクトリーの変更に失敗しました。
- SSC_INFILEOPENFAIL - 入力ファイルを開く操作に失敗しました。
- SSC_OUTFILEOPENFAIL - 出力ファイルを開く操作に失敗しました。
- SSC_SRVCONNFAIL - サーバーの接続に失敗しました。
- SSC_INITERROR - 操作の開始に失敗しました。
- SSC_CORRUPTED_DBFILE - アサートまたはその他の致命的エラーです。
- SSC_CORRUPTED_LOGFILE - アサートまたはその他の致命的エラーです。

コメント

デフォルトでは、状態は SSC_STATE_OPEN に設定されます。

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

solidDB サーバーを新たに始動するときは、solidDB で新しいデータベースを作成するように明示的に指定する必要があります。そのためには、関数 SSCStartServer() を、-U *username* -P *password* -C *catalogname* (デフォルトのデータベース・カタログ名) の各パラメーターを指定して実行します。詳しくは、36 ページの『SSC API 関数 SSCStartServer による明示的な始動』を参照してください。

データベース・サーバーを再始動する場合 (つまりデータベースが既にディレクトリーに存在する場合) は、SSCStartServer で既存のデータベースが使用されます。

SSCStartServer 関数は、複数のスレッドを作成してサーバー・タスクを実行することがあります。サーバー・タスクには、ローカルおよびリモートのクライアント要求の処理だけでなく、チェックポイントやマージなどの各種バックグラウンド・タスクの実行も含まれます。

関連項目

SSCStopServer

SSCStartDisklessSMAServer

SSCStartDisklessSMAServer は、SMA を使用してディスクレス・サーバーを始動します。

構文

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

SSCStartDisklessSMAServer 関数で使用されるパラメーターは以下のとおりです。

表 33. SSCStartDisklessSMAServer パラメーター

パラメーター	使用タイプ	説明
argc	in	コマンド行引数の数。
argv	in、use	関数呼び出しで使用されるコマンド行引数の配列。引数 argv[0] は、ユーザー・アプリケーションのパスとファイル名用として予約されており、必ず指定する必要があります。 使用可能な引数のリストについては、「*IBM solidDB 管理者ガイド*」のセクション『*solidDB*コマンド行オプション』を参照してください。
h	out	始動されたサーバーへのハンドルを返します。このハンドルは、他の制御 API 関数でサーバーを参照する際に必要となります。
runflags	in	このパラメーターには以下のオプションのみを指定できます。 SSC_STATE_OPEN - リモート接続が許可されます。 runflags = SSC_STATE_OPEN
lic_string	in	solidDB ライセンス・ファイルを含んだストリングを指定します。
ini_string	in	solidDB 構成ファイルを含んだストリングを指定します。

戻り値

- SSC_SUCCESS - サーバーが始動されました。
- SSC_ERROR - サーバーの始動に失敗しました。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。
- SSC_INFO_SERVER_RUNNING - サーバーは既に稼働しています。
- SSC_INVALID_HANDLE - 無効なローカル・サーバー・ハンドルが指定されました。
- SSC_INVALID_LICENSE - ライセンスがないか、無効なライセンス・ファイルが見つかりました。

コメント

デフォルトでは、状態は SSC_STATE_OPEN に設定されます。

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

例

```
TBA
```

関連項目

SSCStopServer

SSCStartSMAServer

SSCStartSMAServer は、SMA を使用してサーバーを始動します。

構文

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

SSCStartSMAServer 関数で使用されるパラメーターは以下のとおりです。

表 34. SSCStartSMAServer パラメーター

パラメーター	使用タイプ	説明
argc	in	コマンド行引数の数。
argv	in、use	関数呼び出しで使用されるコマンド行引数の配列。引数 argv[0] は、ユーザー・アプリケーションのパスとファイル名用として予約されており、必ず指定する必要があります。 使用可能な引数のリストについては、「*IBM solidDB 管理者ガイド*」のセクション『*solidDB*コマンド行オプション』を参照してください。
h	out	始動されたサーバーへのハンドルを返します。このハンドルは、他の制御 API 関数でサーバーを参照する際に必要となります。

表 34. *SSCStartSMAServer* パラメーター *(続き)*

パラメーター	使用タイプ	説明
runflags	in	このパラメーターには以下のオプションのみを指定できます。 SSC_STATE_OPEN - リモート接続が許可されます。 runflags = SSC_STATE_OPEN

戻り値

- SSC_SUCCESS - サーバーが始動されました。
- SSC_ERROR - サーバーの始動に失敗しました。
- SSC_ABORT
- SSC_BROKENNETCOPY - 不完全なネットコピーが原因でデータベースが破損しました。
- SSC_FINISHED
- SSC_CONT
- SSC_CONNECTIONS_EXIST
- SSC_UNFINISHED_TASKS
- SSC_INVALID_HANDLE - 無効なローカル・サーバー・ハンドルが指定されました。
- SSC_INVALID_LICENSE - ライセンスがないか、無効なライセンス・ファイルが見つかりました。
- SSC_NODATABASEFILE - データベース・ファイルが見つかりませんでした。
- SSC_SERVER_NOTRUNNING
- SSC_INFO_SERVER_RUNNING - サーバーは既に稼働しています。
- SSC_SERVER_INNETCOPYMODE - サーバーはネットコピー・モードです (HotStandby のみ)。
- SSC_DBOPENFAIL - データベースを開く操作に失敗しました。
- SSC_DBCONNFAIL - データベースへの接続に失敗しました。
- SSC_DBTESTFAIL - データベース・テストが失敗しました。
- SSC_DBFIXFAIL - データベースの修正に失敗しました。
- SSC_MUSTCONVERT - データベースを変換する必要があります。
- SSC_DBEXIST - データベースが存在します。
- SSC_DBNOTCREATED - データベースが作成されていません。
- SSC_DBCREATEFAIL - データベースの作成に失敗しました。
- SSC_COMINITFAIL - 通信の開始に失敗しました。
- SSC_COMLISTENFAIL - 通信の listen に失敗しました。
- SSC_SERVICEFAIL - サービスの操作に失敗しました。
- SSC_ILLARGUMENT - コマンド行引数が正しくありません。
- SSC_CHDIRFAIL - ディレクトリーの変更に失敗しました。

- SSC_INFILEOPENFAIL - 入力ファイルを開く操作に失敗しました。
- SSC_OUTFILEOPENFAIL - 出力ファイルを開く操作に失敗しました。
- SSC_SRVCONNFAIL - サーバーの接続に失敗しました。
- SSC_INITERROR - 操作の開始に失敗しました。
- SSC_CORRUPTED_DBFILE - アサートまたはその他の致命的エラーです。
- SSC_CORRUPTED_LOGFILE - アサートまたはその他の致命的エラーです。

コメント

デフォルトでは、状態は SSC_STATE_OPEN に設定されます。

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

solidDB サーバーを新たに始動するときは、solidDB で新しいデータベースを作成するように明示的に指定する必要があります。そのためには、関数 SSCStartSMAServer を、-U *username* -P *password* -C *catalogname* (デフォルトのデータベース・カタログ名) の各パラメーターを指定して実行します。詳しくは、36 ページの『SSC API 関数 SSCStartServer による明示的な始動』を参照してください。

データベース・サーバーを再始動する場合 (データベースが既にディレクトリーに存在する場合) は、SSCStartSMAServer で既存のデータベースが使用されます。

SSCStartSMAServer 関数は、複数のスレッドを作成してサーバー・タスクを実行することがあります。サーバー・タスクには、ローカルおよびリモートのクライアント要求の処理だけでなく、チェックポイントやマージなどの各種バックグラウンド・タスクの実行も含まれます。

例

```
TBA
```

関連項目

SSCStopSMAServer

SSCStopServer

SSCStopServer は、リンク・ライブラリー・アクセス・サーバーを停止します。

暗黙的な方法 (SQLConnect など) で始動されたサーバーは、明示的な方法 (SSCStopServer など) でシャットダウンできます。ただし、その逆は不可能です。例えば、SSCStartServer で始動されたサーバーを SQLDisconnect で停止することはできません。

アプリケーションが実行中にサーバーを始動および停止できる回数は 1 回に制限されていません。サーバーが停止された後に、アプリケーションで SSCStartServer を使用してサーバーを再始動することができます。

構文

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

SSCStopServer 関数で使用されるパラメーターは以下のとおりです。

表 35. SSCStopServer のパラメーター

パラメーター	使用タイプ	説明
h	in、use	サーバーへのハンドル。
force	in	オプションは以下のとおりです。 • TRUE - あらゆる場合においてサーバーを停止します。 • FALSE - 開いている接続がない場合にサーバーを停止します。それ以外の場合は停止が失敗します。

戻り値

- SSC_SUCCESS - サーバーが停止されました。
- SSC_CONNECTIONS_EXIT - 開いている接続があります。
- SSC_UNFINISHED_TASKS - 実行中のタスクがあります。
- SSC_ABORT
- SSC_ERROR

コメント

リモート・ユーザーは、ADMIN COMMAND 'shutdown' を使用することで solidDB を停止できます。詳しくは、55 ページの『付録 B. リンク・ライブラリー・アクセスのパラメーター』を参照してください。

FALSE オプションを指定すると、データベースまたは既存のユーザーとの接続が開いている場合に、シャットダウンが許可されなくなります。このオプションは、solidDB SQL 拡張 ADMIN COMMAND 'shutdown' に相当します。

SSCSetState() に &~SSC_STATE_OPEN オプションを指定すると、solidDB との新たな接続を作成できなくなります。

関連項目

SSCStartServer

SSCSetState

SSCUnregisterThread

SSCUnregisterThread は、サーバーに対する solidDB アプリケーション・スレッドの登録を抹消します。SSCUnregisterThread 関数は、 SSCRegisterThread 関数で自己を登録したすべてのスレッドで呼び出す必要があります。この関数は、スレッドが終了する前に呼び出されます。

構文

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

SCCUnregisterThread 関数で使用されるパラメーターは以下のとおりです。

表 36. SCCUnregisterThread のパラメーター

パラメーター	使用タイプ	説明
h	in、use	サーバーへのハンドル

戻り値

- SSC_SUCCESS
- SSC_INVALID_HANDLE

コメント

SSC_CALL は、ユーザー関数の呼び出し規則を明示的に定義するために必要です。SSC_CALL は、`sscapi.h` ファイルで各プラットフォームに合わせて定義されます。

この関数には、対応する solidDB SQL 拡張 ADMIN COMMAND がありません。

関連項目

SSCRegisterThread

付録 E. SolidServerControl クラス・インターフェース

SolidServerControl クラスの完全なパブリック・インターフェースを以下に説明します。

パラメーター、および対応する ADMIN COMMAND に関する詳細は、 64 ページの『SSC API リファレンス』のセクションの SSC 関数の説明に含まれています。

このクラスの一部のメソッドを使用する LLA プログラムの例については、samples/aclib_java ディレクトリーにある LLA for Java サンプルを参照してください。

戻り値の定数

```
        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;

/**
 * SolidServerControl クラスを開始します。出力はどの
 * PrintStream にも送信されません。
*
 * @戻り値               SolidServerControl インスタンス
 */
public static  SolidServerControl instance()
     throws SolidServerInitializationError;


/**
 * SolidServerControl クラスを開始します。出力はパラメーター
 * os で指定された PrintStream オブジェクトに送信されます。
*
 * @パラメーター os      出力用の PrintStream
 * @戻り値               SolidServerControl インスタンス
 *
 */
public static SolidServerControl instance( PrintStream os )
     throws SolidServerInitializationError;

/**
 * setOutStream メソッドは、出力を指定の PrintStream に設定します。
 *
 * @パラメーター os      出力用の PrintStream
 */
public void setOutStream( PrintStream os );


/**
 * getOutStream は SolidServerControl クラスの出力に使用される
 * ストリームを返します。
```

```
*
 * @戻り値                このオブジェクトの出力ストリーム
 */
public PrintStream getOutStream();

/**
* startDisklessServer は、solidDB リンク・ライブラリー・アクセス・サーバーを
* ディスクレス・モードで始動します。
*
* @パラメーター argv       アクセラレーター・サーバーのパラメーター・ベクトル。
*
* @パラメーター runflags   このパラメーターのオプションは、SSC_STATE_OPEN (リモート
*                  接続を許可) および SSC_STATE_PREFETCH (必要に応じてサーバーが
*                  「プリフェッチ」を実行) です。プリフェッチでは、
*                  表内容の先読み機能を備えたメモリー・キャッシュが
*                  参照されます。runflags パラメーターは以下のように
*                  入力します。
*
* @パラメーター lic_file   ディスクレス・バージョンはディスクから
*                  情報を読み取れないため、ストリングとしての solidDB
*                  ライセンス・ファイルの内容
*
* @パラメーター ini_file   ディスクレス・バージョンはディスクから情報を
*                  読み取れないため、ストリングとしての solid.ini 構成
*                  ファイルの内容
*
* @戻り値    サーバーからの戻り値は以下のとおりです。
*             SSC_SUCCESS
*             SSC_ERROR
*    SSC_INVALID_LICENSE - ライセンスまたはライセンス・ファイルが見つからない場合
*             SSC_NODATABASEFILE  - データベース・ファイルが見つからない場合
*/
public static long startDisklessServer( String[] argv, long runflags,
String lic_string, String ini_string )


/**
 * startServer は、solidDB リンク・ライブラリー・アクセス・サーバーを始動します
 *
 * @パラメーター argv       LLA サーバーのパラメーター・ベクトル
 *                  注意! solidDB ライセンス・ファイルが置かれている
 *                  作業ディレクトリー (-c¥tmp など) を先に指定し、その後
 *                  に他のパラメーターを指定してください。
 *
 * @パラメーター runflags このパラメーターのオプションは、SSC_STATE_OPEN
 *                       (リモート接続を許可) および SSC_STATE_PREFETCH
 *                       (必要に応じてサーバーがプリフェッチを実行) です。
 *                       プリフェッチでは、表内容の先読み機能を備えたメモリー
 *                       およびディスク・キャッシュ、またはその両方が参照
 *                       されます。runflags パラメーターは以下のように
 *                       入力します。
 *
 *                       runflags |= SSC_STATE_OPEN & SSC_STATE_PREFETCH
 *
 * @戻り値    サーバーからの戻り値は以下のとおりです。
 *              SSC_SUCCESS
 *              SSC_ERROR
 *              SSC_INVALID_LICENSE - ライセンスがないか、無効なライセンス・
 *              ファイルが見つかった場合
 *              SSC_NODATABASEFILE  - データベース・ファイルが見つからない場合
 */
public long startServer( String[] argv, long runflags );


/**
* stopServer は、LLA サーバーを停止します
 *
```

```
 * @パラメーター runflags   LLA サーバーを停止するための runflags。
 * 詳細については、『SSCStopServer』セクションを
 * 参照してください。
*
 * @戻り値           サーバーからの戻り値は以下のとおりです。
 *              SSC_SUCCESS サーバーが停止した場合
 *              SSC_CONNECTIONS_EXIT - 開いている接続がある場合
 *              SSC_UNFINISHED_TASKS - 実行するタスクが残っている場合
 *              SSC_SERVER_NOTRUNNING - サーバーが稼働していない場合
 */
public long stopServer( int runflags );


/**
 * サーバーの状態、つまりサーバーが稼働しているかどうかを返します。
 *
 * @戻り値 SSC_STATE_OPEN - サーバーが稼働状態にある場合
 */
public int getState();




/**
* registerThread は、このユーザー・スレッドを solidDB リンク・ライブラリー・
* アクセス・サーバーに登録します。
*
*
* @戻り値           サーバーからの戻り値は以下のとおりです。
*                   SSC_SUCCESS            登録が成功した場合
*                   SSC_ERROR              登録が失敗した場合
*                   SSC_INVALID_HANDLE     無効なローカル・サーバー・ハンドルが
                                           指定された場合
*                   SSC_SERVER_NOTRUNNING  サーバーが稼働していない場合
*/
public long registerThread();



/**
* unregisterThread は、solidDB リンク・ライブラリー・アクセス・サーバーから
* このユーザー・スレッドの登録を抹消します。
*
*
* @戻り値           サーバーからの戻り値は以下のとおりです。
*                   SSC_SUCCESS            登録が成功した場合
*                   SSC_ERROR              登録が失敗した場合
*                   SSC_INVALID_HANDLE     無効なローカル・サーバー・ハンドルが
                                           指定された場合
*                   SSC_SERVER_NOTRUNNING  サーバーが稼働していない場合
*/
public long unregisterThread();
```

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ラ行]

B

C

F

I

J

L

M

O

S

特記事項

Solid Information Technology Ltd. または International Business Machines Corporation の書面による明示的な許可がある場合を除き、本製品のいかなる部分も、いかなる方法においても使用することはできません。

本製品は、米国特許 6144941、 7136912、 6970876、 7139775、 6978396、 7266702、 7406489、および 7502796 により保護されています。

本製品は、米国輸出規制品目分類番号 ECCN=5D992b に指定されています。

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

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

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年)。このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

商標

IBM、IBM ロゴ、ibm.com®、Solid、solidDB、InfoSphere™、DB2®、Informix®、および WebSphere® は、International Business Machines Corporation の米国およびその他の国における商標です。これらおよび他の IBM 商標に、この情報の最初に現れる個所で商標表示 (® または ™) が付されている場合、これらの表示は、この情報が公開された時点で、米国において、IBM が所有する登録商標またはコモン・ロー上の商標であることを示しています。このような商標は、その他の国においても登録商標またはコモン・ロー上の商標である可能性があります。現時点での IBM の商標リストについては、「Copyright and trademark information」(www.ibm.com/legal/copytrade.shtml) をご覧下さい。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

Printed in Japan

SC88-8166-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21