



プログラマーズ・リファレンス

jConnect™ for JDBC™

7.0

ドキュメント ID : DC38164-01-0700-01

改訂 : 2009 年 10 月

Copyright © 2010 by Sybase, Inc. All rights reserved.

このマニュアルは Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル・ノートで特に示されないかぎり、後続のリリースにも付属します。このマニュアルの内容は予告なしに変更されることがあります。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

このマニュアルの内容を弊社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても、複製、転載、翻訳することを禁じます。

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイバース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

Sybase の商標は、**Sybase trademarks ページ** (<http://www.sybase.com/detail?id=1011207>) で確認できます。Sybase およびこのリストに掲載されている商標は、米国法人 Sybase, Inc. の商標です。® は、米国における登録商標であることを示します。

Java および Java 関連の商標は、米国およびその他の国における Sun Microsystems, Inc. の商標または登録商標です。

Unicode と Unicode のロゴは、Unicode, Inc. の登録商標です。

このマニュアルに記載されている上記以外の社名および製品名は、当該各社の商標または登録商標の場合があります。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに	vii
第 1 章	概要
	JDBC の概要..... 1
	jConnect の概要..... 2
第 2 章	プログラミング情報..... 5
	jConnect の設定..... 5
	jConnect バージョンの設定..... 5
	jConnect ドライバの呼び出し..... 8
	J2EE サーバ向けの jConnect の設定..... 9
	接続の確立..... 10
	接続プロパティ..... 10
	Adaptive Server への接続..... 21
	sql.ini および interfaces ファイルのディレクトリ・ サービスの使用方法..... 23
	JNDI を使用してサーバに接続する方法..... 24
	国際化とローカライゼーションの処理..... 30
	jConnect を使用して Unicode データを渡す..... 30
	jConnect 文字セット・コンバータ..... 31
	データベースの作業..... 36
	高可用性フェールオーバー・サポートの実装..... 36
	サーバ間のリモート・プロシージャ・コールの実行..... 41
	Adaptive Server でのワイド・テーブル・サポートの使用..... 42
	データベース・メタデータへのアクセス..... 43
	結果セットでのカーソルの使用方法..... 45
	COMPUTE 句での Transact-SQL クエリの使用..... 55
	バッチ更新のサポート..... 56
	ストアド・プロシージャの結果セットからのデータベースの更新..... 57
	データ型の作業..... 58
	高度な機能の実装..... 64
	BCP 挿入の使用..... 64
	サポートされている Adaptive Server クラスタ・ エディションの機能..... 65
	イベント通知の使用方法..... 67

エラー・メッセージの処理	69
マイクロ秒の精度の time データ	74
パスワードの暗号化の使用	75
テーブル内のカラム・データとしての Java オブジェクトの格納	78
動的クラス・ロードの使用	82
JDBC 4.0 仕様のサポート	86
JDBC 3.0 仕様のサポート	86
JDBC 2.0 オptional・パッケージ拡張サポート	90
JDBC 標準の制約と解釈	98
サポートされない JDBC 4.0 仕様要件	98
Connection.isClosed と IS_CLOSED_TEST の使用	98
未処理の結果がある場合の Statement.close の使用	99
マルチスレッドに対する調整	100
ResultSet.getCursorName の使用	100
大きなパラメータ値での setLong の使用	101
サポートされるデータ型	101
ストアド・プロシージャの実行	102
第 3 章 セキュリティ	105
概要	105
制限事項	106
カスタム・ソケット・プラグインの実装	106
カスタム・ソケットの作成と設定	108
Kerberos	110
Kerberos を使用するための jConnect アプリケーションの設定	111
GSSMANAGER_CLASS 接続プロパティ	111
Kerberos 環境の設定	114
サンプル・アプリケーション	116
krb5.conf 構成ファイル	118
相互運用性	120
トラブルシューティング	121
関連マニュアル	122
第 4 章 トラブルシューティング	123
jConnect でのデバッグ	123
Debug クラスのインスタンスの取得	123
アプリケーションのデバッグをオンにする方法	124
アプリケーションのデバッグをオフにする方法	124
デバッグ用に CLASSPATH を設定する方法	124
Debug のメソッドの使用	125
TDS 通信の取得	126
PROTOCOL_CAPTURE 接続プロパティ	127
Capture クラスの pause メソッドと resume メソッド	127
接続エラーの解決	128
ゲートウェイ接続が拒否される	128

	jConnect アプリケーションでのメモリ管理	129
	ストアド・プロシージャのエラーの解決	130
	RPC が返す出力パラメータの数が登録されている数よりも少ない	130
	出力パラメータが返される場合のフェッチとステータスのエラー	130
	ストアド・プロシージャを非連鎖トランザクション・ モードでしか実行できない	130
	カスタム・ソケット実装エラーの解決	131
第 5 章	パフォーマンスとチューニング	133
	jConnect のパフォーマンスの改善	133
	BigDecimal の位取り変更	134
	REPEAT_READ 接続プロパティ	134
	SunIoConverter 文字セット変換	135
	動的 SQL の prepared 文のパフォーマンス・チューニング	135
	prepared 文かストアド・プロシージャかの選択	136
	移植可能なアプリケーションでの prepared 文	137
	prepared 文と jConnect の拡張機能	138
	Connection.prepareStatement	139
	DYNAMIC_PREPARE 接続プロパティ	139
	SybConnection.prepareStatement	140
	ESCAPE_PROCESSING_DEFAULT 接続プロパティ	141
	カーソルのパフォーマンス	141
	LANGUAGE_CURSOR 接続プロパティ	142
第 6 章	jConnect アプリケーションへのマイグレート	143
	jConnect 7.x へのアプリケーションのマイグレート	143
	Sybase 拡張機能の変更	144
	拡張機能の変更例	144
	メソッド名	145
	Debug クラス	145
第 7 章	Web サーバ・ゲートウェイ	147
	Web サーバ・ゲートウェイの概要	147
	TDS トンネリングの使用法	148
	jConnect とゲートウェイの設定	148
	使用上の条件	152
	index.html ファイルの読み込み	153
	サンプル Isql アプレットの実行	153
	TDS トンネリング・サーブレットの使用法	154
	要件の確認	155
	サーブレットのインストール方法	155
	サーブレットの呼び出し	157
	アクティブな TDS セッションのトラッキング	157
	TDS セッションの再開	157

付録 A	SQL の例外メッセージと警告メッセージ	159
付録 B	jConnect サンプル・プログラム	179
	IsqlApp の実行	179
	jConnect のサンプル・プログラムとサンプル・コードの実行	181
	サンプル・アプリケーション	181
	サンプル・コード	182
索引	185

はじめに

『jConnect for JDBC プログラマーズ・リファレンス』－ jConnect™ for JDBC™ 製品について説明し、この製品を使用してリレーショナル・データベース管理システムに保管されているデータにアクセスする方法について説明しています。

対象読者

このマニュアルは、Java プログラミング言語、JDBC、および Sybase® 版の SQL である Transact-SQL® についての知識を持っているデータベース・アプリケーション・プログラマの方を対象としています。

このマニュアルの内容

このマニュアルは、次のように構成されています。

- 「第 1 章 概要」では、jConnect for JDBC の概念とコンポーネントについて説明します。
- 「第 2 章 プログラミング情報」では、jConnect for JDBC のプログラミングに関する要件について説明します。
- 「第 3 章 セキュリティ」では、jConnect で使用できるセキュリティ・メカニズムについて説明します。
- 「第 4 章 トラブルシューティング」では、jConnect を使用しているときに発生することがある問題の解決法と対処方法について説明します。
- 「第 5 章 パフォーマンスとチューニング」では、jConnect を使用するアプリケーションのパフォーマンスを向上させる方法について説明します。
- 「第 6 章 jConnect アプリケーションへのマイグレート」では、アプリケーションを jConnect 7.x にマイグレートする方法について説明します。
- 「第 7 章 Web サーバ・ゲートウェイ」では、Web サーバ・ゲートウェイについて説明し、さらに jConnect での使い方について説明します。
- 「第 A 章 SQL の例外メッセージと警告メッセージ」は、jConnect を使用しているときに表示される可能性のある SQL の例外メッセージと警告メッセージのリストです。
- 「第 B 章 jConnect サンプル・プログラム」では、jConnect サンプル・プログラムについて説明します。

関連マニュアル

詳細については、これらのマニュアルを参照できます。

- 『リリース・ノート jConnect for JDBC』には、jConnect に関する重要な最新情報が記載されています。
- 使用しているプラットフォームの Software Developer's Kit リリース・ノートには、Software Developer's Kit (SDK) に関する重要な最新情報が記載されています。
- 『Software Developer's Kit/Open Server インストール・ガイド』には SDK および jConnect for JDBC コンポーネントのインストールについて説明します。
- Adaptive Server Enterprise の『インストール・ガイド』には、Adaptive Server のインストールについて説明します。
- 使用しているプラットフォームの Adaptive Server® Enterprise の『リリース・ノート』では、既知の問題および更新の詳細について説明します。
- jConnect extensions to JDBC の javadoc マニュアル – Java Software の Java Development Kit (JDK) には、ソース・コード・ファイルからコメントを抽出する javadoc スクリプトが含まれています。このスクリプトは、jConnect ソース・ファイルから jConnect のパッケージ、クラス、メソッドのマニュアルを抽出するために使用されています。フル・インストールまたは javadoc オプションを使用して jConnect をインストールするとき、javadoc の情報は次の *javadocs* ディレクトリ、*Installation_directory/docs/en/javadocs* に置かれます。

その他の情報

Sybase Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

一部のマニュアルは PDF 形式で提供されています。これらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の *README.txt* ファイルを参照してください。

- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使用してアクセスできます。また、製品マニュアルのほか、EBFs/Updates、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

Technical Library Product Manuals Web サイトにアクセスするには、Product Manuals (<http://www.sybase.com/support/manuals/>) にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ 製品認定の最新情報にアクセスする

- 1 Web ブラウザで Technical Documents を指定します。
(<http://www.sybase.com/support/techdocs/>)
- 2 [Partner Certification Report] をクリックします。
- 3 [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
- 4 [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

❖ コンポーネント認定の最新情報にアクセスする

- 1 Web ブラウザで Availability and Certification Reports を指定します。
(<http://certification.sybase.com/>)
- 2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。
- 3 [Search] をクリックして、入手状況と認定レポートを表示します。

❖ Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用カスタマイズできます。

- 1 Web ブラウザで Technical Documents を指定します。
(<http://www.sybase.com/support/techdocs/>)
- 2 [MySybase] をクリックし、MySybase プロファイルを作成します。

❖ EBF とソフトウェア・メンテナンスの最新情報にアクセスする

- 1 Web ブラウザで Sybase Support Page を指定します。
(<http://www.sybase.com/support>)
- 2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。
- 3 製品を選択します。
- 4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。
- 5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記規則

表 1: 構文の表記規則

キー	定義
command	コマンド名、コマンドのオプション名、ユーティリティ名、ユーティリティのフラグ、キーワードは sans serif で示す。
variable	変数 (ユーザが入力する値を表す語) は斜体で表記する。
{ }	中カッコは、その中から必ず 1 つ以上のオプションを選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには中カッコは入力しない。
()	このカッコはコマンドの一部として入力する。
	中カッコまたは角カッコの中の縦線で区切られたオプションのうち 1 つだけを選択できることを意味する。
,	中カッコまたは角カッコの中のカンマで区切られたオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

jConnect for JDBC と HTML マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

この製品のオンライン・ヘルプは HTML でも提供され、スクリーン・リーダーの読み上げで内容を理解できる機能があります。

注意 アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、**Sybase Accessibility** (<http://www.sybase.com/accessibility>) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

この章では、jConnect for JDBC を紹介し、その概念とコンポーネントについて説明します。

トピック名	ページ
JDBC の概要	1
jConnect の概要	2

JDBC の概要

Sun Microsystems, Inc. の Java Software Division の JDBC (Java Database Connectivity) は、Java アプリケーションで SQL (Structured Query Language) を使用して複数のデータベース管理システムにアクセスするための API (アプリケーション・プログラム・インタフェース) の仕様です。JDBC Driver Manager は、それぞれ異なるデータベースに接続する複数のドライバを処理します。

標準の JDBC API および JDBC Standard Extension API に含まれているインタフェースのセットを使用して、データベースへの接続のオープン、SQL コマンドの実行、結果の処理を行います。表 1-1 は、これらのインタフェースの説明です。

表 1-1: JDBC インタフェース

インタフェース	説明
<code>java.sql.Driver</code>	データベース URL に対するドライバを見つける。
<code>java.sql.Connection</code>	特定のデータベースへの接続に使用する。
<code>java.sql.Statement</code>	SQL 文を実行する。
<code>java.sql.PreparedStatement</code>	パラメータを使用する SQL 文を処理する。
<code>java.sql.CallableStatement</code>	データベースのストアド・プロシージャ・コールを処理する。
<code>java.sql.ResultSet</code>	SQL 文の結果を取得する。
<code>java.sql.DatabaseMetaData</code>	データベースへの接続に関する情報にアクセスするときに使用する。
<code>java.sql.ResultSetMetaData</code>	結果セットの属性を表す情報にアクセスするときに使用する。
<code>javax.sql.Rowset</code>	JDBC RowSet の実装を処理する。
<code>javax.sql.DataSource</code>	データ・ソースへの接続を処理する。
<code>javax.sql.ConnectionPoolDataSource</code>	接続プールを処理する。

リレーショナル・データベース管理システムごとに、これらのインタフェースを実装するためのドライバが必要です。JDBC ドライバには、次の 4 つのタイプがあります。

- タイプ 1 *JDBC-ODBC* ブリッジ – JDBC 呼び出しを ODBC 呼び出しに変換して ODBC ドライバに渡します。ODBC ソフトウェアの中には、クライアント・マシン上に常駐していなければならないものもあります。クライアント・データベースのコードも、クライアント・マシンに常駐する場合があります。
- タイプ 2 ネイティブ *API* / 一部 *Java* で実装されたドライバ – JDBC の呼び出しをデータベース固有の呼び出しに変換します。このドライバは、データベース・サーバと直接通信しますが、クライアント・マシン上にバイナリ・コードが必要です。
- タイプ 3 ネット・プロトコル / すべて *Java* で実装されたドライバ – DBMS に依存しないネット・プロトコルを使用して、中間層サーバと通信します。中間層のゲートウェイが、要求をベンダ固有のプロトコルに変換します。
- タイプ 4 ネイティブ・プロトコル / すべて *Java* で実装されたドライバ – JDBC 呼び出しをベンダ固有の DBMS プロトコルに変換し、クライアント・アプリケーションがデータベース・サーバと直接通信できるようにします。

JDBC とその仕様の詳細については、[Sun Developer Network for Java](http://java.sun.com) (<http://java.sun.com>) を参照してください。

jConnect の概要

jConnect は、Sybase が提供する、パフォーマンスに優れた JDBC ドライバです。jConnect は下記の両方をサポートします。

- 3 層環境でのネット・プロトコル / すべて *Java* で実装されたドライバ
- 2 層環境でのネイティブ・プロトコル / すべて *Java* で実装されたドライバ

jConnect が使用するプロトコルは、Adaptive Server Enterprise および Open Server™ アプリケーションのネイティブ のプロトコルである TDS 5.0 (Tabular Data Stream™ バージョン 5) です。jConnect は JDBC 標準を実装しており、Sybase 製品ファミリへの接続に最適です。これによって、次の製品をはじめとする 25 種類以上のエンタープライズ・システムおよび従来のシステムにアクセスできます。

- Adaptive Server Enterprise
- SQL Anywhere®
- Sybase® IQ
- Replication Server®
- DirectConnect™

さらに、jConnect for JDBC は、DirectConnect を使用して Oracle や AS/400 などのデータ・ソースにアクセスすることもできます。

jConnect の JDBC の実装は、いくつかの点で JDBC の仕様とは異なります。詳細については、「[JDBC 標準の制約と解釈](#)」(98 ページ) を参照してください。

この章では、jConnect for JDBC を構成する基本的なコンポーネントおよびプログラミングに関する要件について説明します。jConnect ドライバの呼び出し、接続プロパティの設定、データベース・サーバへの接続の方法を説明します。また、jConnect の機能の使い方についても説明します。

この章では、次の項目について説明します。

トピック名	ページ
jConnect の設定	5
接続の確立	10
国際化とローカライゼーションの処理	30
データベースの作業	36
高度な機能の実装	64
JDBC 標準の制約と解釈	98

JDBC プログラミングの詳細については、Sun Developer Network (<http://java.sun.com>) にある Java 開発者用のリソース ページを参照してください。

jConnect の設定

この項では jConnect の使用を開始する前に必要な作業について説明します。

jConnect バージョンの設定

jConnect のバージョン・プロパティである JCONNECT_VERSION は、ドライバの動作、およびアクティブにする機能を指定します。たとえば、Adaptive Server 15.5 では、jConnect 6.05 と 7.0 の両方をサポートしていますが、これら 2 つのバージョンでは **datetime** データと **time** データの処理方法が異なります。マイクロ秒の精度の時刻データをサポートしている Adaptive Server 15.5 の jConnect 7.0 に接続するときは、対象となる Adaptive Server カラムが **datetime** または **time** として定義されていても、**bigdatetime** または **bigtime** を使用します。ただし、jConnect 6.05 の場合は、マイクロ秒の精度をサポートしていないため、Adaptive Server 15.5 への接続時には常に **datetime** または **time** のデータを転送します。

jConnect バージョンは、`SybDriver.setVersion` メソッドまたは `JCONNECT_VERSION` 接続プロパティを使用して設定できます。

SybDriver.setVersion の使用

setVersion メソッドは、SybDriver オブジェクトによって作成されたすべての接続における jConnect のデフォルト動作に影響します。setVersion は、バージョン設定を変更するために何度も呼び出すことができます。新しい接続は、接続が確立したときのバージョン設定に対応する動作を継承します。セッション中にバージョン設定を変更しても、現在の接続には影響しません。com.sybase.jdbcx.SybDriver.VERSION_LATEST 定数を使用すると、その jConnect ドライバで可能な最新バージョンの値を要求できます。ただし、バージョンを com.sybase.jdbcx.SybDriver.VERSION_LATEST に設定すると、jConnect ドライバを新しいバージョンのドライバで置き換えた場合に動作が変わる可能性があります。

次のコード例は、jConnect ドライバをロードして、そのバージョンを設定する方法を示します。

```
import java.sql.DriverManager;
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
    Class.forName("com.sybase.jdbc4.jdbc.SybDriver")
        .newInstance();
sybDriver.setVersion(com.sybase.jdbcx.SybDriver.
    VERSION_7);
DriverManager.registerDriver(sybDriver);
```

JCONNECT_VERSION の使用

JCONNECT_VERSION 接続プロパティを使用して、特定の接続で SybDriver バージョン設定を上書きして別のバージョン設定を指定できます。表 2-1 に、有効な JCONNECT_VERSION 値と、これらの値に関連付けられた jConnect の特性を示します。

表 2-1: jConnect バージョン設定とその機能

JCONNECT_VERSION	機能
"7.0"	<p>jConnect 7.0 の動作は、次に示すこのバージョン特有の動作以外は、jConnect 6.05 と同じです。</p> <ul style="list-style-type: none">SQL データ型 bigdatetime および bigtime のサポートをサーバに要求する。15.5 より前のバージョンの Adaptive Server では、この要求は無視される。「マイクロ秒の精度の time データ」(74 ページ) 参照。JDBC 4.0 をサポートする。「JDBC 4.0 仕様のサポート」(86 ページ) および「JDBC 標準の制約と解釈」(98 ページ) 参照。ENABLE_BULK_LOAD の有効な値は、False、True、ARRAY_INSERT_WITH_MIXED_STATEMENTS、ARRAY_INSERT、および BCP。
"6.05"	<p>jConnect 6.05 の動作は、次に示すこのバージョン特有の動作以外は、jConnect 6.0 と同じです。</p> <ul style="list-style-type: none">メタデータ情報などの計算カラムをサポートする。長い識別子をサポートする。長い識別子を使用すると、最長 255 バイトの識別子またはオブジェクト名を使用できる。長い識別子は、テーブル名、カラム名、インデックス名などのほとんどのユーザ定義識別子に適用される。JDBC 3.0 をサポートする。「JDBC 3.0 仕様のサポート」(86 ページ) および「JDBC 標準の制約と解釈」(98 ページ) 参照。

JCONNECT_VERSION	機能
"6"	<p>jConnect 6.0 の動作は、次に示すこのバージョン特有の動作以外は、jConnect 5.x と同じです。</p> <ul style="list-style-type: none"> SQL データ型 date および time のサポートを要求する。12.5.1 より前のバージョンの Adaptive Server では、この要求は無視される。「date データ型と time データ型の使用方法」(63 ページ) 参照。 unichar データ型および univarchar データ型のサポートをサーバに要求する。12.5.1 より前のバージョンの Adaptive Server では、この要求は無視される。「jConnect を使用して Unicode データを渡す」(30 ページ) 参照。 ワイド・テーブルのサポートをサーバに要求する。12.5.1 より前のバージョンの Adaptive Server では、この要求は無視される。「Adaptive Server でのワイド・テーブル・サポートの使用」(42 ページ) 参照。 DISABLE_UNICHAR_SENDING のデフォルト値は false となる。
"5"	<p>jConnect 5.x の動作は、jConnect 4.0 と同じです。</p>
"4"	<p>jConnect 4.0 の動作は、次に示すこのバージョン特有の動作以外は、jConnect 3.0 と同じです。</p> <ul style="list-style-type: none"> LANGUAGE 接続プロパティのデフォルト値は null となる。 デフォルトの動作では、Statement.cancel を呼び出すと、その Statement オブジェクトだけがキャンセルされる。この動作は JDBC 標準に準拠する。 CANCEL_ALL を使用して Statement.cancel の動作を設定する。 JDBC 2.0 のメソッドを使用して Java オブジェクトをカラム・データとして格納および取得できる。「テーブル内のカラム・データとしての Java オブジェクトの格納」(78 ページ) 参照。
"3"	<p>jConnect 3.0 の動作は、次に示すこのバージョン特有の動作以外は、jConnect 2.0 と同じです。</p> <ul style="list-style-type: none"> CHARSET 接続プロパティに文字セットの指定がない場合は、データベースのデフォルト文字セットを使用する。 CHARSET_CONVERTER のデフォルト値は CheckPureConverter クラスとなる。
"2"	<ul style="list-style-type: none"> LANGUAGE 接続プロパティのデフォルト値は us_english となる。 CHARSET 接続プロパティに文字セットの指定がない場合のデフォルト文字セットは iso_1 となる。 CHARSET 接続プロパティにマルチバイトまたは 8 ビット文字セットが指定されていない場合の CHARSET_CONVERTER のデフォルト値は TruncationConverter クラスであり、指定されている場合のデフォルト CHARSET_CONVERTER は CheckPureConverter クラスである。「jConnect 文字セット・コンバータ」(31 ページ) 参照。 デフォルトの動作では、Statement.cancel を呼び出すと、そのオブジェクト、および実行を開始して結果を待っている他の Statement オブジェクトがキャンセルされる。この動作は、JDBC 標準に準拠しない。 CANCEL_ALL を使用して Statement.cancel の動作を設定する。

jConnect ドライバの呼び出し

jConnect ドライバを登録して呼び出すには、jConnect を `jdbc.drivers` システム・プロパティに追加します。`DriverManager` クラスは、初期化時に `jdbc.drivers` に登録されているドライバをロードしようとします。この方法は、効率の面では `Class.forName` を呼び出す方法に劣ります。このプロパティには、複数のドライバをコロン (:) で区切って指定できます。次のコード例は、プログラム内で `jdbc.drivers` にドライバを追加する方法を示します。

```
Properties sysProps = System.getProperties();
String drivers = "com.sybase.jdbc4.jdbc.SybDriver";
String oldDrivers =
sysProps.getProperty("jdbc.drivers");
if (oldDrivers != null)
    drivers += ":" + oldDrivers;
sysProps.put("jdbc.drivers", drivers.toString());
```

注意 `System.getProperties` は、Java アプレットには使用できません。代わりに `Class.forName` メソッドを使用してください。

Java 6 および JDBC 4 では、JDBC ドライバのインスタンス化が簡略化されました。Java システム・プロパティ `jdbc.drivers` を使用して、次のようにドライバ・クラスを指定できます。

```
java -Djdbc.drivers=com.sybase.jdbc4.jdbc.SybDriver UseDriver
```

`UseDriver` プログラムでは、ドライバを明示的にロードする必要はありません。

```
public class UseDriver
{
    public static void main(String[] args)
    {
        try {
            Connection conn = java.sql.DriverManager.getConnection
                ("jdbc:sybase:Tds:localhost:5000?USER=sa&PASSWORD=secret");
            // more code to use connection ...
        }
        catch (SQLException se){
            System.out.println("ERROR: SQLException "+se);
        }
    }
}
```

J2EE サーバ向けの jConnect の設定

com.sybase.jdbc4.jdbc.SybConnectionPoolDataSource クラスを使用して、EAServer などのアプリケーション・サーバで Adaptive Server サーバへの接続プールを設定できます。javax.sql.ConnectionPoolDataSource インタフェースに com.sybase.jdbc4.jdbc.SybConnectionPoolDataSource を実装することで、接続プロパティごとに getter メソッドと setter メソッドを使用できるようになります。

次の例のように、jConnect をプログラムの設定することもできます。

```
private DataSource getDataSource ()
{
    SybConnectionPoolDataSource connectionPoolDataSource = new
        SybConnectionPoolDataSource();
    connectionPoolDataSource.setDatabaseName("pubs2");
    connectionPoolDataSource.setNetworkProtocol("Tds");
    connectionPoolDataSource.setServerName("localhost");
    connectionPoolDataSource.setPortNumber(5000);
    connectionPoolDataSource.setUser("sa");
    connectionPoolDataSource.setPassword(PASSWORD);
    return connectionPoolDataSource;
}
private void work () throws SQLException
{
    Connection conn = null;
    Statement stmt = null;
    DataSource ds = getDataSource();
    try {
        conn = ds.getConnection();
        stmt = conn.createStatement();
        // ...
    }
    finally {
        if (stmt != null) {
            try { stmt.close(); } catch (Exception ex) { /* ignore */ }
        }
        if (conn != null) {
            try { conn.close(); } catch (Exception ex) { /* ignore */ }
        }
    }
}
```

接続の確立

この項では、jConnect を使用して Adaptive Server または SQL Anywhere データベースへの接続を確立する方法について説明します。

接続プロパティ

接続プロパティでは、サーバにログインするために必要な情報を指定し、クライアントとサーバで意図する動作を定義します。接続プロパティ名の太文字と小文字は区別されません。

接続プロパティの設定

接続プロパティは、サーバに接続する前に設定する必要があります。接続プロパティは次の 2 つの方法で設定できます。

- アプリケーションで `DriverManager.getConnection` メソッドを使用する。
- URL を定義するときに接続プロパティを設定する。

注意 URL の中に設定されたドライバ接続プロパティは、アプリケーション内で `DriverManager.getConnection` メソッドを使用して設定された、対応するドライバ接続プロパティよりも優先されることはありません。

次のサンプル・コードでは、`DriverManager.getConnection` メソッドを使用しています。jConnect 付属のサンプル・プログラムにも、これらのプロパティの設定例が含まれています。

```
Properties props = new Properties();
props.put("user", "userid");
props.put("password", "user_password");
/*
 * If the program is an applet that wants to access
 * a server that is not on the same host as the
 * web server, then it uses a proxy gateway.
 */
props.put("proxy", "localhost:port");
/*
 * Make sure you set connection properties before
 * attempting to make a connection. You can also
 * set the properties in the URL.
 */
Connection con = DriverManager.getConnection
    ("jdbc:sybase:Tds:host:port", props);
```

現在の接続設定の表示

ドライバの現在の接続設定を表示するには、`Driver.getDriverPropertyInfo (String url, Properties props)` を使用します。このコードは、次の項目を含む `DriverPropertyInfo` オブジェクトの配列を返します。

- ドライバ・プロパティ
- ドライバ・プロパティに基づいている現在の設定
- 渡された URL およびプロパティ

jConnect 接続プロパティのリスト

表 2-2 は jConnect の接続プロパティとそのデフォルト値を示します。これらのプロパティでは大文字と小文字は区別されません。

表 2-2: 接続プロパティ

プロパティ	説明	デフォルト値
ALTERNATE_SERVER_NAME	<p>ミラーリングされた SQL Anywhere 環境で、プライマリ・データベースおよびセカンダリ・データベースで使用される代替サーバ名を指定する。プライマリ・データベースおよびセカンダリ・データベースで同じ代替サーバ名を使用することで、クライアント・アプリケーションが現在のプライマリ・サーバに接続できるようになる (2 つのサーバのどちらがプライマリ・サーバであるかをあらかじめ認識しておく必要はない)。</p> <p>JDBC URL 構文は <code>jdbc:sybase:Tds:<hostname>:<port#>/database?connection_property=value;</code> のままである。ただし、<code>ALTERNATE_SERVER_NAME</code> を設定すると、jConnect では <code>hostname</code> 変数と <code>port</code> 変数の値が無視される。代わりに、jConnect は SQL Anywhere UDP 検出プロトコルを使用して、現在のプライマリ・サーバを判別する。</p> <p>データベースのミラーリングの詳細については、『SQL Anywhere Server - Database Administration』 (英語) を参照してください。</p> <p>注意 <code>ALTERNATE_SERVER_NAME</code> は、ミラーリングされていない SQL Anywhere でも使用できます。ただし、常に単一のサーバから同じホストとポートの値を取得することになります。</p>	Null
APPLICATIONNAME	アプリケーション名を指定する。ユーザ定義のプロパティ。このプロパティに指定された値を解釈するようにサーバ側をプログラミングできる。	Null

プロパティ	説明	デフォルト値
BE_AS_JDBC_COMPLIANT_AS_POSSIBLE	<p>jConnect メソッドの応答が JDBC 3.0 標準にできるだけ準拠するように、他のプロパティを調整する。</p> <p>このプロパティを“true”に設定すると、次のプロパティが影響を受ける(上書きされる)。</p> <ul style="list-style-type: none"> • CANCEL_ALL (“false”に設定) • LANGUAGE_CURSOR (“false”に設定) • SELECT_OPENS_CURSOR (“true”に設定) • FAKE_METADATA (“true”に設定) • GET_BY_NAME_USES_COLUMN_LABEL (“false”に設定) 	False
CACHE_COLUMN_METADATA	<p>SELECT クエリを実行する PreparedStatement オブジェクトまたは CallableStatement オブジェクトを繰り返して使用する場合、CACHE_COLUMN_METADATA を true に設定することで、パフォーマンスを向上させることができる。true に設定すると、文の最初の実行で得られた SELECT クエリの結果に関連する ResultSet Metadata 情報は文で記憶される。その後の実行では、メタデータが再利用されるため、再構成されることはない。これにより、追加メモリを使用する CPU 時間が短縮される。</p>	False
CANCEL_ALL	<p>Statement.cancel メソッドの動作を次のように指定する。</p> <ul style="list-style-type: none"> • CANCEL_ALL が false の場合は、Statement.cancel を呼び出すと、その Statement オブジェクトだけがキャンセルされる。したがって、たとえば stmtA が Statement オブジェクトならば、stmtA.cancel はデータベース内の stmtA に含まれる SQL 文の実行をキャンセルするが、他の文への影響はない。stmtA は、キャッシュ内で実行を待っているか、実行が開始して結果を待っているかにかかわらず、キャンセルされる。 • CANCEL_ALL が true の場合は、Statement.cancel を呼び出すと、そのオブジェクトだけでなく、同じ接続上の、既に実行を開始していて結果を待っている他の Statement オブジェクトもキャンセルされる。 <p>次の例では、CANCEL_ALL を“false”に設定している。<i>props</i> は接続プロパティを指定する Properties オブジェクト。</p> <pre>props.put("CANCEL_ALL", "false");</pre> <p>注意 サーバ上で実行を開始しているかどうかに関係なく、接続上にあるすべての Statement オブジェクトの実行をキャンセルする場合は、拡張メソッド SybConnection.cancel を使用します。</p>	<ul style="list-style-type: none"> • True – JCONNECT_VERSION <= “3” の場合 • False – JCONNECT_VERSION >= “4” の場合

プロパティ	説明	デフォルト値
CAPABILITY_TIME	<p>JCONNECT_VERSION >= 6 の場合にのみ使用。jConnect が接続しているサーバが TIME データ型をサポートしている場合、<code>java.sql.Time</code> 型のすべてのパラメータまたは <code>escape literals {t ...}</code> は TIME として処理される。</p> <p>以前のバージョンの jConnect では、このようなパラメータを DATETIME として扱い、<code>java.sql.Time</code> パラメータの前に '1970-01-01' を付加する。基本となるデータ型が <code>datetime</code> または <code>smalldatetime</code> である場合は、日付の部分もデータベースに格納される。jConnect 6.0 以降で TIME が処理される場合、サーバは時刻を基本となるデータ型に変換し、サーバ独自の基底の年を前に付加する。これにより、古いデータと新しいデータの間に非互換性が生じる可能性がある。<code>java.sql.Time</code> の代わりに <code>datetime</code> または <code>smalldatetime</code> データ型を使用する場合、下位互換性を保つために、CAPABILITY_TIME を <i>false</i> のままにする必要がある。</p> <p>このプロパティを <i>false</i> に設定すると、サーバに TIME データ型を扱う機能があるかどうかにかかわらず、jConnect は <code>java.sql.Time</code> パラメータまたはエスケープ・リテラル <code>{t ...}</code> を DATETIME として処理する。</p> <p>このプロパティを <i>true</i> に設定すると、jConnect は Adaptive Server 以降に接続した場合に <code>java.sql.Time</code> パラメータを TIME データ型として処理する。<code>smalldatetime</code> または <code>datetime</code> カラムを使用して時刻値を格納する場合は、このプロパティを <i>false</i> にすることが望ましい。</p>	False
CAPABILITY_WIDETABLE	<p>カラム名などの JDBC ResultSetMetaData がパフォーマンス向上策として必要ない場合は、<i>false</i> に設定できる。これにより、ネットワーク上のデータ交換が減り、パフォーマンスが向上する。EAServer を使用しない場合はデフォルト設定を使用することが望ましい。「Adaptive Server でのワイド・テーブル・サポートの使用」(42 ページ) 参照。</p>	False
CHARSET	<p>データベースに渡される文字列の文字セットを指定する。CHARSET の値が <i>null</i> の場合は、<code>string</code> データをサーバに送信するときにサーバのデフォルトの文字セットが使用される。CHARSET を指定する場合は、データベースがそのフォーマットの文字を処理できなければならない。処理できない場合は、文字変換が正常に実行されなかったことを示すメッセージが生成される。</p> <hr/> <p>注意 DISABLE_UNICHAR_SENDING が <i>false</i> に設定された状態で jConnect 6.05 以降を使用しているときに、クライアントがサーバに送信しようとする文字が、その接続で使用されている文字セットで表現できないものである場合は、jConnect がこのことを検知します。このとき、jConnect はその文字データを <code>unichar</code> データとしてサーバに送信します。これにより、クライアントは Unicode データを <code>unichar/univarchar</code> カラムおよびパラメータに挿入できます。</p>	Null

プロパティ	説明	デフォルト値
CHARSET_ CONVERTER_CLASS	jConnect で使用する文字セット・コンバータ・クラスを指定する。jConnect は、 <code>SybDriver.setVersion</code> からのバージョン設定、または <code>JCONNECT_VERSION</code> プロパティで渡されたバージョンを使用して、デフォルトの文字セット・コンバータ・クラスを決定する。詳細については、「 文字セット・コンバータの選択 」(32 ページ) を参照。	バージョンに依存。表 2-1 (6 ページ) 参照。
CLASS_LOADER	このプロパティには、作成した <code>DynamicClassLoader</code> オブジェクトを設定する。 <code>DynamicClassLoader</code> は、アプリケーション起動時に、データベースに格納されているけれども <code>CLASSPATH</code> にはない Java クラスをロードするときに使用する。詳細については、「 動的クラス・ロードの使用 」(82 ページ) を参照。	Null
CONNECTION_ FAILOVER	JNDI (Java Naming and Directory Interface) とともに使用する。「 CONNECTION_FAILOVER 接続プロパティ 」(27 ページ) 参照。	True
CRC	このプロパティを <code>true</code> に設定すると、返される更新カウントは、実行される文が直接影響する更新と、実行される文の結果として呼び出されるトリガが直接影響する更新とを含む累積数になる。	false
DATABASE	接続情報が Sybase の <code>interfaces</code> ファイルから取得される場合、このプロパティを使用して、接続するデータベース名を指定します。 <code>interfaces</code> ファイルの URL でデータベース名を指定することはできません。	null
DEFAULT_QUERY_ TIMEOUT	この接続プロパティを設定すると、この接続で作成されるすべての文に対して、この接続プロパティがデフォルトのクエリ・タイムアウトとして使用される。	0 (タイムアウトなし)
DISABLE_UNICHAR_ SENDING	クライアント・アプリケーションが <code>unichar</code> 文字を非 <code>unichar</code> 文字とともにサーバに送信すると、データベースへの文字データの送信のパフォーマンスがわずかに低下する。jConnect 6.05 以降では、このプロパティのデフォルトは <code>false</code> となる。古いバージョンの jConnect を使用しているクライアントから <code>unichar</code> データをデータベースに送信するには、このプロパティを <code>false</code> に設定する必要がある。「 jConnect を使用して Unicode データを渡す 」(30 ページ) 参照。	バージョンに依存
DISABLE_ UNPROCESSED_ PARAM_WARNINGS	警告が行われないようにする。ストアド・プロシージャの結果を処理するときに、jConnect はロー・データ以外の戻り値を読み込むこともある。アプリケーション側でこの戻り値を処理しなければ、jConnect の警告が発生する。この警告が行われないようにするには、このプロパティを “true” に設定する (このようにすればパフォーマンスが向上する)。	False
DYNAMIC_PREPARE	動的 SQL の prepared 文をデータベース内でプリコンパイルするかどうかを決定する。「 DYNAMIC_PREPARE 接続プロパティ 」(139 ページ) 参照。	False

プロパティ	説明	デフォルト値
ENABLE_BULK_LOAD	<p>データベースにローを挿入する際にバルク・ロードを使用するかどうかを指定する。値：</p> <ul style="list-style-type: none"> • False – バルク・ロードを無効にする。 • True または ARRAYINSERT_WITH_MIXED_STATEMENTS – ロー・レベルのロギングでバルク・ロードを有効にし、バルク・ロード・オペレーション中にアプリケーションで他の文を実行できるようにする。 • ARRAYINSERT – ロー・レベルのロギングでバルク・ロードを有効にするが、バルク・ロード・オペレーション中にアプリケーションで他の文を実行できない。 • BCP – ページ・レベルのロギングでバルク・ロードを有効にする。バルク・ロード・オペレーション中にアプリケーションで他の文を実行できない。 	False
ENABLE_SERVER_PACKETSIZE	<p>接続のパケット・サイズをサーバで指定された値に設定するかどうかを指定する。true に設定した場合、ドライバは PACKETSZ 接続プロパティを使用せず、サーバは 512 から最大パケット・サイズまで任意の値を使用できる。false に設定した場合、PACKETSZ 接続プロパティが使用される。</p>	True
ENCRYPT_PASSWORD	<p>セキュア・ログインを可能にする。このプロパティを true に設定すると、ログイン・パスワードおよびリモート・サイト・パスワードの両方が暗号化されてからサーバに送信される。パスワードはクリア・テキストで送信されない。</p> <p>ENCRYPT_PASSWORD は RETRY_WITH_NO_ENCRYPTION より優先度が高い。パスワードの暗号化の詳細については、「パスワードの暗号化の使用」(75 ページ)を参照。</p>	False
ESCAPE_PROCESSING_DEFAULT	<p>SQL 文中の JDBC 関数エスケープの処理を迂回する。デフォルトでは、jConnect はデータベースに送信されるすべての SQL 文を解析して、有効な JDBC 関数エスケープがあるかどうかを調べます。アプリケーションの SQL 呼び出しの中で JDBC 関数エスケープを使用しない場合は、この接続プロパティを“false”に設定すると、この解析を迂回できる。これにより、パフォーマンスが若干向上する可能性がある。</p> <p>また、ESCAPE_PROCESSING_DEFAULT を使用すると、現在 SQL 構文として中かっこを使用している Sybase IQ などのバックエンド・サーバで役に立つ。</p>	True
EXPIRESTRING	<p>ライセンスの有効期限日を示す。評価版の jConnect 以外は、有効期限は Never に設定される。これは読み取り専用プロパティである。</p>	Never

プロパティ	説明	デフォルト値
FAKE_METADATA	<p>偽のメタデータを返す。ResultSetMetaData のメソッド getCatalogName、getSchemaName、getTableName を呼び出したとき、サーバからは有効なメタデータが渡されないで、このプロパティが “true” に設定されている場合は空の文字列 (“”) が返される。</p> <p>このプロパティを “false” に設定すると、これらのメソッドを呼び出したときに、「実装されていない」という SQL 例外が発生する。</p> <p>注意 ワイド・テーブルが使用可能な状態で、Adaptive Server 12.5 以降のバージョンを使用している場合は、サーバから有効なメタデータが返されるので、このプロパティ設定は無視されます。</p>	False
GET_BY_NAME_USES_COLUMN_LABEL	<p>jConnect 6.0 より前のバージョンとの下位互換性を実現する。</p> <p>Adaptive Server バージョン 12.5 以降では、それまでのバージョンよりも多くのメタデータに jConnect からアクセスできる。12.5 より前のバージョンでは、column name と column alias は同じものを意味するが、Adaptive Server バージョン 12.5 以降を使用し、ワイド・テーブルが使用可能な状態のときは、この 2 つを区別できる。</p> <p>下位互換性を維持するには、このプロパティを “true” に設定する。</p> <p>getBytes、getInt、get* (String columnName) を呼び出してカラムの実際の名前を調べるには、このプロパティを “false” に設定する。</p>	True
GET_COLUMN_LABEL_FOR_NAME	<p>ResultSetMetaData.getColumnLabel への呼び出しでカラム名ではなくカラム・ラベルを返す場合に、jConnect 5.5 以前との下位互換性を維持する。値：</p> <ul style="list-style-type: none"> • True – ResultSetMetaData.getColumnLabel がカラム・ラベルを返す。 • False – ResultSetMetaData.getColumnLabel がカラム名を返す。 	False
GSSMANAGER_CLASS	<p>org.ietf.jgss.GSSManager クラスのサード・パーティ実装を指定する。</p> <p>このプロパティには文字列または GSSManager オブジェクトを設定できる。</p> <p>文字列の値を設定する場合は、サード・パーティ GSSManager 実装の完全修飾クラス名でなければならない。オブジェクトを設定する場合は、org.ietf.jgss.GSSManager クラスを拡張するオブジェクトでなければならない。詳細については、「第 3 章 セキュリティ」を参照。</p>	Null
HOSTNAME	現在のホストの名前を表す。	<p>なし</p> <p>最大 30 文字。それより長い場合は 30 文字になるようトランケートされる。</p>
HOSTPROC	ホスト・マシン上のアプリケーション・プロセスを表す。	なし
IGNORE_DONE_IN_PROC	中間更新結果 (ストア・プロセスの) は返さずに最終結果セットのみを返すことを指定する。	False

プロパティ	説明	デフォルト値
IMPLICIT_CURSOR_FETCH_SIZE	データベースに送信されるすべての <code>select</code> クエリで読み込み専用カーソルをオープンするように <code>jConnect</code> に強制する場合、 <code>SELECT_OPENS_CURSOR</code> プロパティと共にこのプロパティを使用する。 <code>Statement.setFetchSize</code> メソッドによって上書きされない限り、このプロパティで設定された値のフェッチ・サイズがカーソルに適用される。	0
INTERNAL_QUERY_TIMEOUT	このプロパティを使用して、 <code>jConnect</code> によって内部的に作成および実行される文に使用するクエリ・タイムアウトを設定する。クエリ・タイムアウトを設定することで、内部コマンドが妥当な時間内に完了しなかった場合にアプリケーションに障害が発生することを防止できる場合がある。	0 (タイムアウトなし)
IS_CLOSED_TEST	<code>Connection.isClosed</code> が呼び出されたときにデータベースに送られるクエリを指定できる。詳細については、「 Connection.isClosed と <code>IS_CLOSED_TEST</code> の使用 (98 ページ) を参照。	Null
J2EE_TCK_COMPLIANT	このプロパティを <code>true</code> に設定すると、 <code>jConnect</code> ドライバは、J2EE 1.4 TCK (Technology Compatibility Kit) テスト・スイートに準拠した動作を有効にする。これによりパフォーマンスが若干低下することがある。このため、デフォルト値の <code>false</code> を使用することが推奨される。	false
JCE_PROVIDER_CLASS	RSA 暗号化アルゴリズムで使用される JCE (Java Cryptography Extension) プロバイダを指定する。	バンドルされた JCE プロバイダ
JCONNECT_VERSION	バージョン固有の特性を設定する。「 JCONNECT_VERSION の使用 」 (6 ページ) を参照。	7
LANGUAGE	<code>jConnect</code> からのメッセージとサーバからのメッセージを表示する言語を指定する。サーバ・メッセージは、ローカル環境の言語設定に応じてローカライズされるため、この設定は <code>syslanguages</code> の言語と一致させる必要がある。サポートされている言語は、中国語、英語、フランス語、ドイツ語、日本語、韓国語、ポーランド語、ポルトガル語、スペイン語。	バージョンに依存。 「 JCONNECT_VERSION の使用 」 (6 ページ) 参照。
LANGUAGE_CURSOR	<code>jConnect</code> で「プロトコル・カーソル」ではなく「言語カーソル」を使用することを指定する。「 カーソルのパフォーマンス 」 (141 ページ) 参照。	False
LITERAL_PARAMS	“true” に設定すると、 <code>PreparedStatement</code> インタフェースの <code>setXXX</code> メソッドによって設定されたパラメータは、SQL 文の実行時にリテラルとして SQL 文に挿入される。 “false” に設定すると、パラメータ・マークは SQL 文内に残り、パラメータ値が別にサーバに送信される。	False
NEWPASSWORD	パスワードの有効期限の処理で使用される新しいパスワードを指定する。	Null
PACKETSIZE	ネットワーク・パケット・サイズを表す。Adaptive Server 15.0 以降を使用している場合は、このプロパティを設定せず、環境に適したネットワーク・パケット・サイズが <code>jConnect</code> と Adaptive Server によって選択されるようにすることが望ましい。	512

プロパティ	説明	デフォルト値
PASSWORD	ログイン・パスワードを表す。 getConnection(String, String, String) メソッドを使用する場合は自動的に設定される。getConnection(String, Props) を使用する場合は明示的に設定する必要がある。	なし
PRELOAD_JARS	ユーザが指定した CLASS_LOADER に関連付けられた .jar ファイル名のカンマ区切りのリスト。これらの .jar は接続時にロードされ、同じ jConnect ドライバを使用する他の接続でも使用できる。詳細については、「 jar ファイルの事前ロード 」(85 ページ) を参照。	Null
PROMPT_FOR_NEWPASSWORD	透過的なパスワード変更を実行するか、新しいパスワードの入力を要求するプロンプトを表示するかどうかを指定する。値： <ul style="list-style-type: none"> • True – 新しいパスワードを手動で設定するためのプロンプトを表示する。 • False – NEWPASSWORD を確認し、値が null でない場合はこの値を使用して、期限切れパスワードを置き換える。 	False
PROTOCOL_CAPTURE	アプリケーションと Adaptive Server の間の TDS 通信を取得するためのファイルを指定する。	Null
PROXY	ゲートウェイ・アドレスを指定する。HTTP プロトコルの場合の URL は http://host:port となる。 暗号化をサポートする HTTPS プロトコルを使用する場合の URL は https://host:port/servlet_alias となる。	なし
QUERY_TIMEOUT_CANCELS_ALL	読み取りのタイムアウトが発生したときに、接続上のすべての文を強制的にキャンセルする。この動作は、クライアントが execute() を呼び出したときに、デッドロック (たとえば、別のトランザクションが現在更新中であるテーブルからデータを読み取ろうとしている) が原因でタイムアウトが発生した場合に便利。デフォルト値は <i>false</i> 。	False
REMOTEPWD	サーバ間のリモート・プロシージャ・コールによるアクセスのためのリモート・サーバ・パスワード。「 サーバ間のリモート・プロシージャ・コールの実行 」(41 ページ) 参照。	なし
REPEAT_READ	カラムをランダムな順序で読み込んだり、繰り返し読み込んだりできるよう、ドライバがカラムおよび出力パラメータのコピーを保持するかどうかを決定する。「 REPEAT_READ 接続プロパティ 」(134 ページ) 参照。	True
REQUEST_HA_SESSION	接続しているクライアントが高可用性 (HA) フェールオーバー・セッションを開始するかどうかを示す。「 高可用性フェールオーバー・サポートの実装 」(36 ページ) 参照。 接続が確立された後にこのプロパティを再設定することはできない。フェールオーバー・セッションの要求に柔軟性を持たせるには、実行時に REQUEST_HA_SESSION を設定するようにクライアント・アプリケーションをコーディングする必要がある。	False

注意 “true” に設定すると、jConnect はフェールオーバー・ログインを試行する。この接続プロパティを設定しないと、サーバでフェールオーバーが設定されていても、フェールオーバー・セッションは開始されない。

プロパティ	説明	デフォルト値
REQUEST_KERBEROS_SESSION	<p>認証に Kerberos を使用するかどうかを指定する。このプロパティを“true”に設定した場合は、SERVICE_PRINCIPAL_NAME プロパティの値も指定する必要がある。</p> <p>GSSMANAGER_CLASS プロパティの値を指定することもできる。詳細については、「第3章 セキュリティ」を参照。</p>	False
RETRY_WITH_NO_ENCRYPTION	<p>サーバがクリア・テキスト・パスワードを使用したログインをリトライできるようにする。</p> <p>ENCRYPT_PASSWORD プロパティおよび RETRY_WITH_NO_ENCRYPTION プロパティを両方とも true に設定すると、jConnect は先に暗号化されたパスワードを使用してログインする。ログインが失敗した場合、jConnect はクリア・テキスト形式のパスワードを使用してログインする。パスワードの暗号化の詳細については、「パスワードの暗号化の使用」(75 ページ)を参照。</p>	False
RMNAME	<p>分散トランザクション (XA) を使用する場合はリソース・マネージャ名を設定する。このプロパティは、LDAP サーバ・エントリ内で設定されたリソース・マネージャ名よりも優先される。詳細については、「分散トランザクション管理のサポート」(96 ページ)を参照。</p>	Null
SECONDARY_SERVER_HOSTPORT	<p>クライアントが HA フェールオーバー・セッションを使用するときのセカンダリ・サーバのホスト名とポートを設定する。このプロパティの値は、hostName:portNumber の形式で設定する。REQUEST_HA_SESSION が“true”に設定されていないと、このプロパティは無視される。詳細については、「高可用性フェールオーバー・サポートの実装」(36 ページ)を参照。</p>	Null
SELECT_OPENS_CURSOR	<p>Statement.executeQuery が呼び出されたときのクエリに FOR UPDATE 句が含まれている場合に自動的にカーソルを生成するかどうかを指定する。</p> <p>同じ文に対してすでに Statement.setFetchSize または Statement.setCursorName が呼び出されている場合は、SELECT_OPENS_CURSOR を“true”に設定しても効果はない。</p> <p>注意 SELECT_OPENS_CURSOR を“true”に設定すると、パフォーマンスが若干低下することがあります。</p> <p>jConnect でカーソルを使用する方法の詳細については、「結果セットでのカーソルの使用方法」(45 ページ)を参照。</p>	False
SERIALIZE_REQUESTS	<p>サーバからの応答を待ってから次の要求を送信するかどうかを指定する。</p>	False
SERVER_INITIATED_TRANSACTIONS	<p>サーバによるトランザクション制御を可能にする。デフォルトでは、このプロパティは true に設定されており、Transact-SQL コマンドの set chained on を使用して、サーバがトランザクションの開始と制御を行うことができる。false に設定すると、Transact-SQL コマンドの begin tran を使用して jConnect によりトランザクションが開始され、制御される。トランザクションはサーバで制御できるようにすることが望ましい。</p>	True

プロパティ	説明	デフォルト値
SERVICENAME	DirectConnect ゲートウェイによって実行されるバックエンド・データベース・サーバの名前を示す。SQL Anywhere への接続時に使用するデータベースを示す場合にも使用される。	なし
SERVERTYPE	OpenSwitch に接続されている場合は、このプロパティを“OSW”に設定する。これにより、jConnect から OpenSwitch に特定の命令を送信し、OpenSwitch が別のサーバ・インスタンスに接続をリダイレクトした場合でも、独立性レベル、テキスト・サイズ、引用符付き識別子、オートコミットなどの初期接続設定を保持できる。	なし
SERVICE_PRINCIPAL_NAME	Adaptive Server に対して Kerberos 接続を確立するときに使用される。このプロパティの値は、KDC (Key Distribution Center) 内のサーバ・エントリと、データベースを実行しているサーバ名の両方に一致する必要がある。 REQUEST_KERBEROS_SESSION プロパティを“false”に設定すると、SERVICE_PRINCIPAL_NAME プロパティの値は無視される。詳細については、「 第3章 セキュリティ 」を参照。	Null
SESSION_ID	TDS セッション ID。このプロパティが設定されているとき、jConnect は、TDS トンネリング・ゲートウェイによってオープンされたままになっている既存の TDS セッション上でアプリケーションが通信を再開しようとしていると想定する。jConnect はログイン・ネゴシエーションをスキップし、アプリケーションからの要求をすべて指定のセッション ID に転送する。	Null
SESSION_TIMEOUT	HTTP トンネル・セッション (jConnect TDS トンネリング・サブレットで作成されたもの) のアイドル状態が保たれる時間を秒単位で指定する。指定した時間が経過すると、接続は自動的にクローズする。TDS トンネリング・サブレットの詳細については、「 TDS トンネリングの使用法 」(148 ページ) を参照。	Null
SQLINITSTRING	接続がオープンしたときにデータベース・サーバに渡されるコマンドのセットを定義する。コマンドは、 <code>Statement.executeUpdate</code> メソッドを使用して実行できる SQL コマンドでなければならない。	Null
STREAM_CACHE_SIZE	文の応答ストリームのキャッシュに使用する最大サイズを指定する。	Null (キャッシュ・サイズの制限なし)
SYB SOCKET_FACTORY	jConnect でカスタム・ソケット実装を使用できるようにする。 SYB SOCKET_FACTORY を次のいずれかに設定すること。 <ul style="list-style-type: none">• <code>com.sybase.jdbcx.SybSocketFactory</code> を実装するクラスの名前。• “DEFAULT”。この場合は、新しい <code>java.net.Socket()</code> がインスタンス化される。 このプロパティは、データベースへの SSL 接続を確立するために使用する。	Null
TEXTSIZE	テキスト・サイズを設定できる。Adaptive Server および SQL Anywhere では、デフォルトで、image または text カラムから 32,627 バイトを読み込み可能。jConnect の MDA テーブルがインストールされている場合、jConnect はその値を 2GB に変更する。OpenSwitch に接続する場合にこの値を設定すると、OpenSwitch が別のサーバ・インスタンスに接続をリダイレクトしたときに接続の設定を保持できる。	2GB

プロパティ	説明	デフォルト値
USE_METADATA	<p>接続を確立するときに <code>DatabaseMetaData</code> オブジェクトを作成して初期化する。<code>DatabaseMetaData</code> オブジェクトは指定のデータベースに接続する必要がある。</p> <p><code>jConnect</code> は、分散トランザクション管理サポート (JTA/JTS) や動的クラス・ロード (DCL) などの機能に対して <code>DatabaseMetaData</code> を使用する。</p> <p>アプリケーションにメタデータが必要であることを示すエラー 010SJ を受け取った場合は、<code>jConnect</code> 付属の、メタデータを返すストアド・プロシージャをインストールする必要がある。『<code>jConnect for JDBC</code> インストール・ガイド』の第 3 章の「ストアド・プロシージャのインストール」を参照。</p>	True
USER	<p>ログイン ID を指定する。</p> <p><code>getConnection(String, String, String)</code> メソッドを使用する場合は自動的に設定される。<code>getConnection(String, Props)</code> を使用する場合は明示的に設定する必要がある。</p>	なし
VERSIONSTRING	JDBC ドライバのバージョン情報 (読み取り専用)。	<code>jConnect</code> ドライバのバージョン

Adaptive Server への接続

Java アプリケーションでは、URL を定義し、`jConnect` ドライバを使用して Adaptive Server に接続します。URL の基本的なフォーマットは次のとおりです。

```
jdbc:sybase:Tds:host:port
```

各パラメータの意味は、次のとおりです。

- `jdbc:sybase` — ドライバを指定します。
- `Tds` — Adaptive Server と通信するための Sybase 通信プロトコルです。
- `host:port` — Adaptive Server のホスト名と受信ポートです。データベースや Open Server アプリケーションが使用するエントリについては、`$$SYBASE/interfaces` (UNIX) または `%SYBASE%\ini\sql.ini` (Windows) を参照してください。“query” エントリから `host:port` を取得してください。

次のフォーマットを使用すると、特定のデータベースに接続できます。

```
jdbc:sybase:Tds:host:port/database
```

注意 SQL Anywhere または DirectConnect を使用している特定のデータベースに接続するには、“/database” の代わりに、`SERVICENAME` 接続プロパティを使用してデータベース名を指定してください。

例 次のコードは、ホスト “myserver” 上のポート 3697 で受信する Adaptive Server への接続を作成します。

```
SysProps.put("user", "userid");
SysProps.put("password", "user_password");
String url = "jdbc:sybase:Tds:myserver:3697";
Connection_con =
    DriverManager.getConnection(url, SysProps);
```

URL 接続プロパティのパラメータ

URL を定義するときに、jConnect ドライバ接続プロパティの値を指定できます。

注意 URL の中に設定されたドライバ接続プロパティは、アプリケーション内で `DriverManager.getConnection` メソッドを使用して設定された、対応するドライバ接続プロパティよりも優先されることはありません。

URL 内で接続プロパティを設定するには、プロパティ名とその値を URL 定義に追加します。使用する構文：

```
jdbc:sybase:Tds:host:port/database?
    property_name=value
```

複数の接続プロパティを設定するには、各接続プロパティ値の前に “&” を付けて追加します。例：

```
jdbc:sybase:Tds:myserver:1234/mydatabase?
    LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=myhost
```

接続プロパティの値に “&” が含まれている場合は、その値の “&” の前に円記号 (¥) を追加してください。たとえば、ホスト名が “a&bhost” の場合は、次の構文を使用します。

```
jdbc:sybase:Tds:myserver:1234/mydatabase?
    LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=
    a¥&bhost
```

接続プロパティの値が文字列であっても、引用符は使用しないでください。たとえば、次のようにします。

```
HOSTNAME=myhost
```

次のように入力しないでください。

```
HOSTNAME="myhost"
```

sql.ini および interfaces ファイルのディレクトリ・サービスの使用方法

sql.ini ファイル (Windows 用) と *interfaces* ファイル (UNIX 用) を使用して、jConnect for JDBC に対するサーバ情報を提供できます。*sql.ini* または *interfaces* ファイルを使用することで、エンタープライズ・ネットワークで使用可能なサービスに関する情報を、Adaptive Server 向けの情報を含めてすべて集中管理できます。

sql.ini または *interfaces* ファイルを指定するには、接続文字列を使用します。jConnect for JDBC では、単一のディレクトリ・サービスの URL (DSURL) にのみ接続できます。

jConnect に対する単一の DSURL 用の接続文字列

DSURL に接続する場合は、*sql.ini* ファイルまたは *interfaces* ファイルへのパスとサーバ名を指定する必要があります。指定しない場合、エラーが返されます。

sql.ini ファイルへのパスは次のように指定します。

```
String url = "jdbc:sybase:jndi:file:///D:/syb1252/ini/mysql.ini?myaseISO1"
```

パラメータの意味は次のとおりです。

- server name = myaseISO1
- *sql.ini* file path = file:///D:/syb1252/ini/sql.ini

interfaces ファイルへのパスは次のように指定します。

```
String url = "jdbc:sybase:jndi:file:///work/sybase/interfaces?myase"
```

パラメータの意味は次のとおりです。

- server name = myase
- *interfaces* file path = file:///work/sybase/interfaces

sql.ini ファイルと interfaces ファイルの SSL 用フォーマット

SSL 用の *sql.ini* ファイルのフォーマットを次に示します。

```
[SYBSRV2]
master=nlwnsck,mangol,4100,ssl
query=nlwnsck,mangol,4100,ssl
query=nlwnsck,mangol,5000,ssl
```

interfaces ファイルのフォーマットを次に示します。

```
sybsrv2
master tcp ether mangol 5000 ssl
query tcp ether mangol 4100 ssl
query tcp ether mangol 5000 ssl
```

注意 jConnect では、*sql.ini* ファイルまたは *interfaces* ファイル内の同じサーバ名の下に指定した複数のクエリ・エントリをサポートします。jConnect は、*sql.ini* ファイルまたは *interfaces* ファイルに指定された順序に従って、クエリ・エントリから **host** または **port** の値に接続を試みます。クエリ・エントリ内に SSL が見つかった場合、jConnect は、アプリケーション固有のソケット・ファクトリを指定することで SSL 接続を処理するようにコーディングされたアプリケーションを必要とします。このアプリケーションがない場合、接続は失敗します。

JNDI を使用してサーバに接続する方法

jConnect では、JNDI (Java Naming and Directory Interface) を使用して接続情報を定義できます。これには、次のような特徴があります。

- サーバに接続するためのホスト名およびポートの指定を 1 か所で集中管理する。アプリケーション内に特定のホストとポート番号をハードコードする必要はありません。
- すべてのアプリケーションで使用できるように、接続プロパティとデフォルト・データベースの指定を 1 か所で集中管理する。
- 接続試行の失敗を処理するための jConnect CONNECTION_FAILOVER プロパティ。CONNECTION_FAILOVER が “true” に設定されているときは、jConnect は JNDI ネーム・スペース内の一連のホスト／ポート・サーバ・アドレスへの接続を順に試行し、いずれかに成功するまで続けます。

jConnect とともに JNDI を使用するには、JNDI がアクセスするディレクトリ・サービス内に情報を用意し、その必要な情報を `javax.naming.Context` クラス内に設定する必要があります。この項では、次の項目について説明します。

- [JNDI を使用するための接続 URL](#)
- [必要なディレクトリ・サービス情報](#)
- [CONNECTION_FAILOVER 接続プロパティ](#)
- [JNDI コンテキスト情報の提供](#)

JNDI を使用するための接続 URL

接続情報の取得に JNDI を使用するよう指定するには、“sybase” の後に URL プロトコルとして “jndi” を追加します。

```
jdbc:sybase:jndi:protocol-information-for-use-with-JNDI
```

この URL の “jndi” に続く部分はすべて JNDI を介して処理されます。たとえば、JNDI とともに LDAP (Lightweight Directory Access Protocol) を使用するには、次のように入力します。

```
jdbc:sybase:jndi:ldap://LDAP_hostname:port_number/servername=
Sybase11,o=MyCompany,c=US
```

この URL は、LDAP サーバから情報を取得するよう JNDI に通知します。使用する LDAP サーバのホスト名とポート番号を指定し、さらに LDAP 固有の形式でデータベース・サーバの名前を指定しています。

必要なディレクトリ・サービス情報

jConnect とともに JNDI を使用するときは、接続先のデータベース・サーバに関する次の情報が JNDI から返される必要があります。

- 接続先のホスト名とポート番号
- 使用するデータベースの名前
- 個々のアプリケーションが独自に設定することができない接続プロパティ

この情報は、接続情報の提供に使用されるディレクトリ・サービス内に、固定のフォーマットに従って格納される必要があります。このフォーマットは、返される情報（接続先データベースなど）の種類を指定するための数値のオブジェクト識別子 (OID) と、それに続くフォーマットされた情報で構成されます。

注意 OID の代わりにエイリアスを使って属性を参照することもできます。

表 2-3 に必要なフォーマットを示します。

表 2-3: JNDI に必要なディレクトリ・サービス情報

属性の説明	エイリアス	OID (object_id)
LDAP ディレクトリ・サービス でのインタフェース・エントリ の置換	sybaseServer	1.3.6.1.4.1.897.4.1.1
sybaseServer LDAP 属性の収集 ポイント	sybaseServer	1.3.6.1.4.1.897.4.2
バージョン属性	sybaseVersion	1.3.6.1.4.1.897.4.2.1
サーバ名属性	sybaseServer	1.3.6.1.4.1.897.4.2.2
サービス属性	sybaseService	1.3.6.1.4.1.897.4.2.3
ステータス属性	sybaseStatus	1.3.6.1.4.1.897.4.2.4
アドレス属性	sybaseAddress	1.3.6.1.4.1.897.4.2.5
セキュリティ・メカニズム属性	sybaseSecurity	1.3.6.1.4.1.897.4.2.6
再試行カウント属性	sybaseRetryCount	1.3.6.1.4.1.897.4.2.7
ループ遅延属性	sybaseRetryDelay	1.3.6.1.4.1.897.4.2.8
jConnect 接続プロトコル	sybaseJconnectProtocol	1.3.6.1.4.1.897.4.2.9
jConnect 接続プロパティ	sybaseJconnectProperty	1.3.6.1.4.1.897.4.2.10
データベース名	sybaseDatabasename	1.3.6.1.4.1.897.4.2.11
高可用性フェールオーバー・サー バ名属性	sybaseHAservname	1.3.6.1.4.1.897.4.2.15
リソース・マネージャ名	sybaseResourceManagerName	1.3.6.1.4.1.897.4.2.16
リソース・マネージャ・タイプ	sybaseResourceManagerType	1.3.6.1.4.1.897.4.2.17
JDBC データ・ソース・インタ フェース	sybaseJdbcDataSource- インタフェース	1.3.6.1.4.1.897.4.2.18
ServerType	sybaseServerType	1.3.6.1.4.1.897.4.2.19

注意 太字の属性は必須です。

次の例では、LDAP ディレクトリ・サービス下のデータベース・サーバ
“SYBASE11” に対して入力された接続情報を示します。例 1 では属性の OID
を使用し、例 2 では属性のエイリアス (大文字と小文字を区別しません) を使
用します。OID とエイリアスのどちらを使用してもかまいません。

例 1 dn: servername=SYBASE11,o=MyCompany,c=US
 servername:SYBASE11
 1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
 1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
 1.3.6.1.4.1.897.4.2.5:TCP#1#standby1 4444
 1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&
 PACKETSIZE=1024
 1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=true
 1.3.6.1.4.1.897.4.2.11:pubs2
 1.3.6.1.4.1.897.4.2.9:Tds

例 2 dn: servername=SYBASE11,o=MyCompany,c=US
 servername:SYBASE11
 sybaseAddress:TCP#1#giotto 1266
 sybaseAddress:TCP#1#giotto 1337
 sybaseAddress:TCP#1#standby1 4444
 sybaseJconnectProperty:REPEAT_READ=false&
 PACKETSIZE=1024
 sybaseJconnectProperty:CONNECTION_FAILOVER=true
 sybaseDatabasename:pubs2
 sybaseJconnectProtocol:Tds

この例では、SYBASE11 はホスト “giotto” のポート 1266 または 1337 を介してアクセスでき、ホスト “standby1” のポート 4444 を介してアクセスすることもできます。REPEAT_READ と PACKETSIZE の 2 つの接続プロパティは 1 つのエントリで設定されています。CONNECTION_FAILOVER 接続プロパティは別のエントリで設定されています。SYBASE11 に接続するアプリケーションは、最初は pubs2 データベースに接続されます。接続プロトコルを指定する必要はありませんが、指定する場合は、属性を “TDS” ではなく “Tds” と入力してください。

CONNECTION_FAILOVER 接続プロパティ

CONNECTION_FAILOVER は、jConnect が JNDI を使用して接続情報を取得する場合に使用できるブール値の接続プロパティです。

CONNECTION_FAILOVER が True に設定されている場合、jConnect はサーバへの接続を複数回試みます。サーバに関連付けられたホストとポート番号への接続に失敗すると、jConnect は JNDI を使用してそのサーバに関連付けられた次のホストとポート番号を取得し、接続を試みます。サーバに関連付けられたすべてのホストとポートに対して、順に接続が試行されます。

たとえば、CONNECTION_FAILOVER が True に設定されていて、データベース・サーバは、前述の LDAP の例で示したように、次のホストとポート番号に関連付けられているとします。

```
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
1.3.6.1.4.1.897.4.2.5:TCP#1#standby 4444
```

サーバに接続するために、jConnect はホスト “giotto” のポート 1266 への接続を試みます。失敗した場合は、“giotto” のポート 1337 への接続を試みます。これにも失敗した場合は、ホスト “standby1” のポート 4444 を介して接続を試みます。

CONNECTION_FAILOVER のデフォルトは True です。

CONNECTION_FAILOVER が False に設定されている場合、jConnect は最初のホストとポート番号への接続を試みます。失敗した場合は、SQL 例外が発生し、再試行は行いません。

JNDI コンテキスト情報の提供

JNDI とともに jConnect を使用するには、JNDI specification from Sun Microsystems (<http://java.sun.com/products/jndi>) の内容に精通している必要があります。

特に、JNDI と jConnect を組み合わせて使用するときに必要な初期化プロパティを `javax.naming.directory.DirContext` 内に設定する必要があります。これらのプロパティはシステム・レベルと実行時のどちらでも設定できます。

主なプロパティは次の 2 つです。

- `Context.INITIAL_CONTEXT_FACTORY`

このプロパティには、使用する JNDI の初期コンテキスト・ファクトリの完全修飾クラス名を指定します。これによって、`Context.PROVIDER_URL` プロパティで指定された URL で使用される JNDI ドライバが決定します。

- `Context.PROVIDER_URL`

このプロパティには、LDAP ドライバなどのドライバがアクセスするディレクトリ・サービスの URL を指定します。URL は “`ldap://ldaphost:427`” のような文字列として指定します。

次の例では、実行時にコンテキスト・プロパティを設定する方法と、JNDI および LDAP を使用した接続の方法を示します。この例では、`INITIAL_CONTEXT_FACTORY` コンテキスト・プロパティは Sun Microsystems の LDAP サービス・プロバイダの実装を呼び出すように設定されます。`PROVIDER_URL` コンテキスト・プロパティは、ホスト “`ldap_server1`” のポート 389 にある LDAP ディレクトリ・サービスの URL に設定されます。


```

Properties props = new Properties();

/* We want to use LDAP, so INITIAL_CONTEXT_FACTORY is set to the
 * class name of an LDAP context factory. In this case, the
 * context factory is provided by Sun's implementation of a
 * driver for LDAP directory service.
 */
props.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

/* Now, we set PROVIDER_URL to the URL of the LDAP server that
 * is to provide directory information for the connection.
 */
props.put(Context.PROVIDER_URL, "ldap://ldap_server1:389");

/* Set up additional context properties, as needed. */
props.put("user", "xyz");
props.put("password", "123");

/* get the connection */
Connection con = DriverManager.getConnection
    ("jdbc:sybase:jndi:ldap://ldap_server1:389" +
    "/servername=Sybasell,o=MyCompany,c=US",props);

```

`getConnection` に渡される接続文字列には、LDAP 固有の情報が含まれます。これは開発者が指定する必要があります。

前述の例で示したように実行時に JNDI プロパティが設定されると、プロパティは `jConnect` から JNDI に渡され、サーバの初期化に使用されます。次に例を示します。

```

javax.naming.directory.DirContext ctx =
    new javax.naming.directory.InitialDirContext(props);

```

次に、`jConnect` は、次の例に示すように `DirContext.getAttributes` を呼び出して JNDI から必要な接続情報を取得します。`ctx` は `DirContext` オブジェクトです。

```

javax.naming.directory.Attributes attrs =
    ctx.getAttributes("ldap://ldap_server1:389/servername=" +
    "Sybasell", SYBASE_SERVER_ATTRIBUTES);

```

`SYBASE_SERVER_ATTRIBUTES` は、`jConnect` 内で定義された文字列の配列です。配列の値は、[表 2-3](#) のリストに示した必要なディレクトリ情報の OID です。

国際化とローカライゼーションの処理

この項では jConnect に関する国際化とローカライゼーションについて説明します。

jConnect を使用して Unicode データを渡す

Adaptive Server バージョン 12.5 以降では、データベース・クライアントで `unichar` データ型と `univarchar` データ型を利用できます。この 2 つのデータ型により、Unicode データの効率的な格納や取り出しが可能になります。

以下は、Unicode 標準バージョン 2.0 の一部を翻訳したものです。

「Unicode 標準は、文字やテキストをコード化するための、固定幅の画一的なエンコード方式です。Unicode 標準は情報を処理するための国際的な文字コードで、世界中の主要なスクリプトで使われる文字のほか、一般的な技術記号が含まれています。Unicode の文字コードでは、アルファベット、表意文字、記号をまったく同じように扱います。すなわち、これらはどのような組み合わせでも同じように簡単に使用できます。Unicode 標準は ASCII 文字セットに基づいて作られていますが、多言語テキストをサポートするために 16 ビット・エンコードを使用します。」

つまり、ユーザは、サーバのデフォルト文字セットに関係なく、データベース・テーブルのカラムに Unicode データを格納するよう指定できます。

注意 Adaptive Server バージョン 12.5 ～ 12.5.0.3 では、Unicode データ型を使用するにはサーバのデフォルト文字セットが UTF-8 でなければなりません。Adaptive Server 12.5.1 以降では、サーバのデフォルト文字セットがどのようなものであっても、`unichar` と `univarchar` を使用できます。

サーバで `unichar` と `univarchar` のデータが使用可能なときは、jConnect は次のように動作します。

- `PreparedStatement.setString (int column, String value)` などを使用してクライアントからサーバに送信されるすべての文字データについて、文字列をサーバのデフォルト文字セットに変換できるかどうかを調べます。
- 文字をサーバの文字セットに変換できないと判断した場合 (たとえば、サーバの文字セットで表現できない文字がある場合) は、データを `unichar/univarchar` データとしてコード化してサーバに送信します。

たとえば、デフォルト文字セットとして `iso_1` を使用する Adaptive Server 12.5.1 に対して、クライアントが Unicode の日本語文字を送信する場合は、日本語の文字は `iso_1` 文字に変換できないので、文字列を Unicode データとして送信します。

クライアントから `unichar/univarchar` データをサーバに送信するときは、パフォーマンスが低下します。これは、サーバのデフォルト文字セットに直接マッピングできないすべての文字列と文字に対して、jConnect が文字からバイトへの変換を2回実行する必要があるためです。

6.05 よりも前の jConnect バージョンを使用している場合、`unichar` および `univarchar` データ型を使用するには、次のタスクを実行する必要があります。

- 1 `JCONNECT_VERSION` を6以降に設定します。詳細については、「[JCONNECT_VERSION の使用](#)」(6 ページ)を参照してください。
- 2 `DISABLE_UNICHAR_SENDING` 接続プロパティを `false` に設定する必要があります。jConnect 6.05 から、このプロパティはデフォルトで `false` に設定されます。詳細については、「[接続プロパティの設定](#)」(10 ページ)を参照してください。

注意 `unichar` データ型と `univarchar` データ型のサポートの詳細については、Adaptive Server Enterprise のマニュアルを参照してください。

jConnect 文字セット・コンバータ

jConnect は、すべての文字セット変換に対して特別なクラスを使用します。アプリケーション側で文字セット・コンバータ・クラスを選択することによって、シングルバイトおよびマルチバイトの文字セット変換を jConnect がどのように処理するかと、文字セット変換がアプリケーションのパフォーマンスに与える影響が決まります。

文字セット変換クラスは2つあります。jConnect が使用する変換クラスは、`JCONNECT_VERSION`、`CHARSET`、`CHARSET_CONVERTER_CLASS` 接続プロパティに基づいています。

- `TruncationConverter` クラスは、`iso_1` や `cp850` などの ASCII 文字を使用するシングルバイト文字セットでのみ動作します。マルチバイト文字セットや非 ASCII 文字を使用するシングルバイト文字セットでは動作しません。`TruncationConverter` クラスは、`JCONNECT_VERSION` が2に設定されている場合のデフォルト・コンバータです。

`TruncationConverter` クラスを使用する場合は、jConnect 7 は jConnect バージョン 2.2 と同じように文字セットを処理します。`TruncationConverter` クラスは、`JCONNECT_VERSION` が2の場合のデフォルト・コンバータです。

- **PureConverter** クラスは、pure Java のマルチバイト文字セット・コンバータです。JCONNECT_VERSION が 4 以降の場合は、このコンバータ・クラスが使用されます。JCONNECT_VERSION が 2 の場合も、CHARSET 接続プロパティで指定された文字セットが **TruncationConverter** クラスと互換性のないものであるときは、このコンバータが使用されます。

PureConverter クラスによって、マルチバイト文字セット変換が可能になりますが、jConnect ドライバのパフォーマンスに悪影響を与えることがあります。ドライバのパフォーマンスが問題となる場合は、「[文字セット変換パフォーマンスの向上](#)」(33 ページ) を参照してください。

文字セット・コンバータの選択

jConnect は、JCONNECT_VERSION を使用して、使用するデフォルトの文字セット・コンバータ・クラスを決定します。JCONNECT_VERSION が 2 以降の場合、デフォルトは **TruncationConverter** です。JCONNECT_VERSION が 4 以降の場合、デフォルトは **PureConverter** です。

CHARSET_CONVERTER_CLASS 接続プロパティを設定することによって、jConnect が使用する文字セット・コンバータを指定できます。これは、使用するバージョンの jConnect のデフォルト以外の文字セット・コンバータを使用する場合に便利です。

たとえば、JCONNECT_VERSION が 4 以降に設定されているときに、マルチバイトの **PureConverter** クラスではなく **TruncationConverter** クラスを使用する場合は、次のように CHARSET_CONVERTER_CLASS を設定します。

```
...
props.put("CHARSET_CONVERTER_CLASS",
          "com.sybase.jdbc4.utils.TruncationConverter")
```

CHARSET 接続プロパティの設定

CHARSET ドライバ・プロパティを設定することによって、アプリケーションで使用する文字セットを指定できます。CHARSET プロパティを設定していない場合は、次のようになります。

- JCONNECT_VERSION が 2 の場合は、iso_1 がデフォルト文字セットとして使用されます。
- JCONNECT_VERSION が 3 以降の場合は、データベースのデフォルト文字セットが使用され、クライアント側で必要な変換を実行するよう自動的に調整が行われます。
- 6.05 以降の iConnect バージョンでは、ユーザ・データからネゴシエートした文字セットへの変換を正常に実行できないとき、サーバが Unicode 文字をサポートしている場合は未変換の Unicode 文字がサーバに送信され、そうでない場合は例外が返されます。

IsqlApp アプリケーションに対して `-J charset` コマンドライン・オプションを使用して文字セットを指定することもできます。

Adaptive Server にどの文字セットがインストールされているかを調べるには、サーバに対して次の SQL クエリを発行します。

```
select name from syscharsets
go
```

PureConverter クラスの場合に、指定の CHARSET がクライアントの Java 仮想マシン (VM) では機能しないときは、接続は失敗し、Adaptive Server とクライアントの両方でサポートされている文字セットに CHARSET を設定するように指示する `SQLException` が生成されます。

TruncationConverter クラスを使用している場合は、指定の CHARSET が7ビット ASCII であるかどうかに関係なく、文字トランケーションが適用されます。したがって、アプリケーションで ASCII 以外のデータ (たとえばアジア言語) を処理する必要がある場合は、TruncationConverter を使用しないでください。使用すると、データが破損します。

文字セット変換パフォーマンスの向上

マルチバイト文字セットを使用していて、ドライバ・パフォーマンスを改善する必要がある場合は、jConnect サンプルに含まれている `SunloConverter` クラスを使用できます。詳細については、「[SunloConverter 文字セット変換](#)」(135 ページ) を参照してください。

また、アプリケーションで7ビットの ASCII データのみを処理する場合は、TruncationConverter を使用するとパフォーマンスを改善できます。

サポートされている文字セット

表 2-4 は、jConnect でサポートされている Sybase 文字セットのリストです。また、サポートされている文字セットのそれぞれについて、対応する JDK バイト・コンバータも示します。

jConnect は UCS-2 をサポートしていますが、現時点では Sybase データベースおよび Open Server では UCS-2 はサポートされません。

Adaptive Server バージョン 12.5 以降では、Unicode のバージョンのうち UTF-16 エンコーディングと呼ばれるものがサポートされています。

表 2-4: サポートされている Sybase 文字セット

SybCharset Name	JDK Byte Converter
ascii_7	ASCII
big5	Big5
big5hk (「注意」を参照)	Big5_HKSCS
cp037	Cp037
cp437	Cp437

SybCharset Name	JDK Byte Converter
cp500	Cp500
cp850	Cp850
cp852	Cp852
cp855	Cp855
cp857	Cp857
cp860	Cp860
cp863	Cp863
cp864	Cp864
cp866	Cp866
cp869	Cp869
cp874	Cp874
cp932	MS932
cp936	GBK
cp950	Cp950
cp1250	Cp1250
cp1251	Cp1251
cp1252	Cp1252
cp1253	Cp1253
cp1254	Cp1254
cp1255	Cp1255
cp1256	Cp1256
cp1257	Cp1257
cp1258	Cp1258
deckanji	EUC_JP
eucgb	EUC_CN
eucjis	EUC_JP
eucksc	EUC_KR
GB18030	GB18030
ibm420	Cp420
ibm918	Cp918
iso_1	ISO8859_1
iso88592	ISO8859-2
iso88595	ISO8859_5
iso88596	ISO8859_6
iso88597	ISO8859_7
iso88598	ISO8859_8
iso88599	ISO8859_9
iso15	ISO8859_15_FDIS
koi8	KOI8_R
mac	Macroman

SybCharset Name	JDK Byte Converter
mac_cyr	MacCyrillic
mac_ee	MacCentralEurope
macgreek	MacGreek
macturk	MacTurkish
sjis	MS932
tis620	MS874
utf8	UTF8

ヨーロッパ通貨記号のサポート

jConnect では、ヨーロッパの通貨の記号「ユーロ」の使用、および UCS-2 Unicode との間の変換がサポートされます。

ユーロは、Sybase の文字セット cp1250、cp1251、cp1252、cp1253、cp1254、cp1255、cp1256、cp1257、cp1258、cp874、iso885915、utf8 に追加されています。

ユーロ記号を使用するには、次の点に注意してください。

- PureConverter または CheckPureConverter クラス、つまり pure Java のマルチバイト文字セット・コンバータを使用してください。詳細については、「[jConnect 文字セット・コンバータ](#)」(31 ページ) を参照してください。
- 新しい文字セットがサーバにインストールされていることを確認してください。

ユーロ記号は、Adaptive Server と SQL Anywhere でサポートされています。

- クライアント側で適切な文字セットを選択してください。 詳細については、「[CHARSET 接続プロパティの設定](#)」(32 ページ) を参照してください。

サポートされていない文字セット

次の Sybase 文字セットは、類似する JDK バイト・コンバータが存在しないので、jConnect ではサポートされていません。

- cp1047
- euccns
- greek8
- roman8
- roman9
- turkish8

これらの文字の 7 ビット ASCII サブセットだけをアプリケーションで使用する場合は、これらの文字クラスに対して TruncationConverter クラスを使用できます。

データベースの作業

この項では、jConnect に関する次の項目について説明します。

- [高可用性フェールオーバー・サポートの実装](#)
- [サーバ間のリモート・プロシージャ・コールの実行](#)
- [Adaptive Server でのワイド・テーブル・サポートの使用](#)
- [データベース・メタデータへのアクセス](#)
- [結果セットでのカーソルの使用方法](#)
- [COMPUTE 句での Transact-SQL クエリの使用](#)
- [バッチ更新のサポート](#)
- [ストアド・プロシージャの結果セットからのデータベースの更新](#)
- [データ型の作業](#)

高可用性フェールオーバー・サポートの実装

jConnect は Adaptive Server のフェールオーバー機能をサポートします。

注意 高可用性システムでの Sybase フェールオーバーは、接続フェールオーバーとは別の機能です。この2つの機能を両方とも使用する場合は、この項を熟読してください。

概要

Sybase フェールオーバーを使うと、2つの Adaptive Server をコンパニオンとして設定できます。プライマリ・コンパニオンがダウンすると、そのサーバのデバイス、データベース、および接続がセカンダリ・コンパニオンに引き継がれます。

高可用性システムは、非対称型と対称型のどちらにも設定できます。

- 非対称型の設定では、2つの Adaptive Server がそれぞれ物理的に異なるマシンに配置されますが、一方のサーバがダウンしたときはもう一方のサーバがダウンしたサーバの負荷を引き受けるように接続されています。セカンダリ Adaptive Server は「ホット・スタンバイ」として機能し、フェールオーバーが発生するまでは何も処理を実行しません。

- 対称型の設定の場合も、2つの Adaptive Server がそれぞれ別のマシン上で稼働します。しかし、フェールオーバーが発生したときは、一方の Adaptive Server がもう一方の Adaptive Server のプライマリあるいはセカンダリ・コンパニオンとして動作します。この設定では、Adaptive Server はそれぞれシステム・デバイス、システム・データベース、ユーザ・データベース、ユーザ・ログインを持ち、完全に機能します。

どちらの設定でも、2つのマシンはデュアル・アクセス可能に設定されているため、両方のマシンからディスクの内容表示やアクセスが可能です。

jConnect でフェールオーバーを使用できるように設定すると、フェールオーバー可能に設定された Adaptive Server にクライアント・アプリケーションから接続することができます。プライマリ・サーバからセカンダリ・サーバへのフェールオーバーが発生すると、クライアント・アプリケーションの接続先も自動的にセカンダリ・サーバに切り替わり、ネットワーク接続が再確立されます。

注意 詳細については、Adaptive Server Enterprise の『高可用性システムにおける Sybase フェールオーバーの使用』を参照してください。

稼働条件、依存性、および制限

- 2つの Adaptive Server が、フェールオーバー可能に設定されている必要があります。
- クライアントでフェールオーバーが実行された場合、フェールオーバー前にデータベースにコミットされた変更のみが保持されます。
- jConnect 接続プロパティ REQUEST_HA_SESSION を “true” に設定してください (「[接続プロパティ](#)」(10 ページ) を参照)。
- フェールオーバーが発生したときは、jConnect のイベント通知は機能しません (「[イベント通知の使用方法](#)」(67 ページ) を参照)。
- 使用しない文はすべて終了させてください。jConnect は、フェールオーバーを可能にするために、文の情報を保存します。文を終了させないと、メモリ・リークが発生します。

jConnect でのフェールオーバーの実装

jConnect でのフェールオーバー・サポートを実装するには、次の 2 つの方法があります。

- 2 つの接続プロパティ REQUEST_HA_SESSION と SECONDARY_SERVER_HOSTPORT を次のように設定します。
 - REQUEST_HA_SESSION を “true” に設定します。
 - SECONDARY_SERVER_HOSTPORT を、セカンダリ・サーバが受信するホスト名とポート番号に設定します。詳細については、「[接続プロパティ](#)」(10 ページ) と SECONDARY_SERVER_HOSTPORT 接続プロパティを参照してください。
- JNDI を使用してサーバに接続します。「[JNDI を使用してサーバに接続する方法](#)」を参照してください。JNDI に必要なディレクトリ・サービス情報ファイルに、プライマリ・サーバ用のエントリとセカンダリ・サーバ用のエントリを別々に入力します。プライマリ・サーバのエントリには、セカンダリ・サーバのエントリを参照する属性 (HA OID) が必要です。

JNDI のサービス・プロバイダとして LDAP を使う場合は、HA 属性の形式には次の 3 つがあります。

- a 相対識別名 (RDN) – 検索ベース (通常は `java.naming.provider.url` 属性によって指定される) とこの属性の値の組み合わせは、セカンダリ・サーバを識別するのに十分であると見なされます。たとえば、プライマリ・サーバが “hostname:4200”、セカンダリ・サーバが “hostname:4202” にあると仮定すると、次のようになります。

```
dn: servername=happrimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary
objectclass: sybaseServer

dn: servername=hasecondary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

- b 識別名 (DN) – この形式では、HA 属性の値によってセカンダリ・サーバが一意に識別されると見なされ、検索ベース内で重複する値が見つかることも見つからないこともあります。例：

```
dn: servername=happrimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary,
    o=Sybase, c=US ou=Accounting
objectclass: sybaseServer

dn: servername=hasecondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

hasecondary はツリーの別のブランチに位置しています (追加されている ou=Accounting 修飾子に注目してください)。

- c 完全な LDAP URL – この形式では、検索ベースに関する想定は何も行われません。HA 属性は、セカンダリ・サーバの識別に使用される完全修飾 LDAP URL でなければなりません (別の LDAP サーバを指す場合もあります)。例：

```
dn: servername=hafailover, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15: ldap://ldapserver: 386/servername=secondary,
    o=Sybase, c=US ou=Accounting
objectclass: sybaseServer

dn: servername=secondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

- d JNDI に必要なディレクトリ・サービス情報ファイルで、REQUEST_HA_SESSION 接続プロパティを “true” に設定してください。これによって、接続すると必ずフェールオーバー・セッションが有効になります。

REQUEST_HA_SESSION 接続プロパティを使用すると、クライアントは、フェールオーバー可能に設定された Adaptive Server とのフェールオーバー・セッションの開始を要求していることを指定できます。true に設定すると、jConnect はフェールオーバー・ログインを試行します。この接続プロパティを設定しないと、サーバが正しく設定されていても、フェールオーバー・セッションは開始されません。REQUEST_HA_SESSION のデフォルト値は false です。

この接続プロパティは、他の接続プロパティと同じように設定してください。接続が確立された後はプロパティを再設定することはできません。

フェールオーバー・セッションの要求に柔軟性を持たせるには、実行時に `REQUEST_HA_SESSION` を設定するようにクライアント・アプリケーションをコーディングする必要があります。

次の例では、LDAP ディレクトリ・サービス下のデータベース・サーバ“SYBASE11”に対して入力された接続情報を示します。“tahiti”はプライマリ・サーバ、“moorea”はセカンダリ・コンパニオン・サーバです。

```
dn: servername=SYBASE11,o=MyCompany,c=US
1.3.6.1.4.1.897.4.2.5:TCP#1#tahiti 3456
1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&PACKETSIZE=1024
1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=false
1.3.6.1.4.1.897.4.2.11:pubs2
1.3.6.1.4.1.897.4.2.9:Tds
1.3.6.1.4.1.897.4.2.15:servername=SECONDARY
1.3.6.1.4.1.897.4.2.10:REQUEST_HA_SESSION=true
```

```
dn:servername=SECONDARY, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1#moorea 6000
```

- JNDI と LDAP を使用して接続を要求します。
 - jConnect は、LDAP サーバのディレクトリを使用してプライマリ・サーバとセカンダリ・サーバの名前と場所を特定します。

```
/* get the connection */
Connection con = DriverManager.getConnection
    ("jdbc:sybase:jndi:ldap://ldap_server1:389" +
     "/servername=Sybase11,o=MyCompany,c=US",props);
```

または

- 検索ベースを指定します。

```
props.put(Context.PROVIDER_URL,
    "ldap://ldap_server1:389/ o=MyCompany, c=US");
Connection con=DriverManager.getConnection
    ("jdbc:sybase:jndi:servername=Sybase11", props);
```

プライマリ・サーバへのログイン

Adaptive Server がフェールオーバー可能に設定されていない場合や、フェールオーバー・セッションを許可できない場合は、クライアントはログインできず、次のような警告メッセージが表示されます。

```
'The server denied your request to use the high-
availability feature.
```

```
Please reconfigure your database, or do not request a
high-availability session.'
```

セカンダリ・サーバへのフェールオーバー

フェールオーバーが発生すると、SQL 例外 JZ0F2 が発生します。

```
'Sybase high-availability failover has occurred. The
current transaction is aborted, but the connection is
still usable. Retry your transaction.'
```

クライアントは JNDI を使用して自動的にセカンダリ・データベースに再接続されます。

次のことに注意してください。

- クライアントが接続していたデータベースの ID、およびコミットされたトランザクションは保持されます。
- 部分的に読み込まれた結果セット、カーソル、ストアド・プロシージャの呼び出しは失われます。
- フェールオーバーが発生すると、アプリケーションはプロシージャを再起動するか、最後に完了したトランザクションまたはアクティビティに戻る必要がある場合があります。

プライマリ・サーバへのフェールバック

ある時点で、クライアントはセカンダリ・サーバからプライマリ・サーバにフェールバックします。いつフェールバックを行うかは、セカンダリ・サーバ上でシステム管理者が `sp_failback` を発行して設定します。その後、クライアントから見たプライマリ・サーバでの動作と結果は、[「セカンダリ・サーバへのフェールオーバー」\(41 ページ\)](#) の説明と同様になります。

サーバ間のリモート・プロシージャ・コールの実行

サーバ上で実行される Transact-SQL 言語コマンドやストアド・プロシージャから、別のサーバ上にあるストアド・プロシージャを実行できます。アプリケーションが接続されているサーバは、リモート・サーバにログインしてサーバ間のリモート・プロシージャ・コールを実行します。

アプリケーションはサーバ間の通信に「ユニバーサル」なパスワードを指定できます。このパスワードはすべてのサーバ間の通信に使用されます。接続がオープンした後は、サーバはどのリモート・サーバにログインするにも、このパスワードを使用します。デフォルトでは、jConnect は現在の接続のパスワードをサーバ間通信のデフォルト・パスワードとして使用します。

ただし、2つのサーバで同一ユーザに対するパスワードが異なり、そのユーザがサーバ間のリモート・プロシージャ・コールを実行している場合、アプリケーションは各サーバで使用するパスワードを明示的に定義する必要があります。

jConnect には、ユニバーサルな「リモート・パスワード」またはサーバごとに異なるパスワードを指定するためのプロパティがあります。このプロパティを設定するには、`SybDriver` クラスの `setRemotePassword` メソッドを使用します。

```
Properties connectionProps = new Properties();

public final void setRemotePassword(String serverName,
    String password, Properties connectionProps)
```

このメソッドを使用するには、アプリケーションで `SybDriver` クラスをインポートしてからメソッドを呼び出してください。

```
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
    Class.forName("com.sybase.jdbc4.jdbc.SybDriver").newInstance();
sybDriver.setRemotePassword
    (serverName, password, connectionProps);
```

注意 サーバごとに異なるリモート・パスワードを設定するには、各サーバに対して前述の呼び出しを繰り返します。

この呼び出しによって、指定したサーバ名とパスワードの組が、指定した `Properties` オブジェクトに追加されます。アプリケーションは、このオブジェクトを `DriverManager.getConnection(server_url, props)` で `DriverManager` に渡すことができます。

`serverName` が `null` の場合は、ユニバーサル・パスワードが `password` に設定されます。このパスワードは、既に `setRemotePassword` の呼び出しによって明示的にパスワードが定義されているサーバを除くすべてのサーバへの以降の接続に対して使用されます。

アプリケーションが `REMOTEPWD` プロパティを設定すると、それ以降は jConnect によるデフォルト・ユニバーサル・パスワードの設定は行われません。

Adaptive Server でのワイド・テーブル・サポートの使用

Adaptive Server では、以前のバージョンのデータベース・サーバに比べて、使用できるカラム数やパラメータ数が増えています。次に例を示します。

- テーブルに収容できるカラム数は 1,024 です。
- `varchar` カラムと `varbinary` カラムには 255 バイトを超えるデータを格納できます。
- ストアド・プロシージャを呼び出すときやテーブルにデータを挿入するときに、最大 2,048 個のパラメータを送信または取得できます。

jConnect からデータベースにワイド・テーブル・サポートの要求が行われるようにするには、デフォルト設定の JCONNECT_VERSION が 6 以上に設定されている必要があります。

注意 JCONNECT_VERSION を 6 より前に設定している場合も、jConnect は Adaptive Server バージョン 12.5 以降に対して機能します。ただし、ワイド・テーブル・サポートがなければ完全にデータを取り出すことができないテーブルからデータを選択しようとした場合、予期しないエラーやデータのトランケーションが発生する可能性があります。

ワイド・テーブルをサポートしない Sybase サーバのデータにアクセスするときも、JCONNECT_VERSION を 6 以降に設定してかまいません。この場合、サーバはワイド・テーブル・サポート要求を単純に無視します。

ワイド・テーブルのサポートでは、使用可能なカラム数とパラメータ数が増えるということに加えて、拡張結果セット・メタデータも提供します。たとえば、jConnect 6.0 より前のバージョンでは、ResultSetMetaData のメソッド `getCatalogName`、`getSchemaName`、および `getTableName` はいずれも「実装されていない」という SQL 例外を返していましたが、これは、そのメタデータがサーバから返されていなかったからです。ワイド・テーブルのサポートを有効にすると、サーバからこの情報が送り返されるので、上記 3 つのメソッドからは有用な情報が返されます。

データベース・メタデータへのアクセス

JDBC の DatabaseMetaData のメソッドをサポートするために、Sybase では、データベースに関するメタデータを取得するときに jConnect が呼び出す一連のストアード・プロシージャを用意しています。JDBC メタデータ・メソッドが機能するには、これらのストアード・プロシージャがサーバにインストールされている必要があります。

メタデータを返すストアード・プロシージャがまだ Sybase サーバにインストールされていない場合は、jConnect に付属している次のストアード・プロシージャ・スクリプトを使用してインストールしてください。

- `sql_server12.sql` は、バージョン 12.0.x の Adaptive Server データベースにストアード・プロシージャをインストールします。
- `sql_server12.5.sql` は、バージョン 12.5.x の Adaptive Server データベースにストアード・プロシージャをインストールします。
- `sql_server15.0.sql` は Adaptive Server 15.x 以降にストアード・プロシージャをインストールします。
- `sql_asa.sql` は、SQL Anywhere 9.x データベースにストアード・プロシージャをインストールします。

- *sql_asa10.sql* は、SQL Anywhere 10.x データベースにストアド・プロシージャをインストールします。
- *sql_asa11.sql* は、SQL Anywhere 11.x データベースにストアド・プロシージャをインストールします。

注意 これらのスクリプトの最新バージョンは jConnect のすべてのバージョンと互換性があります。

ストアド・プロシージャをインストールする手順については、『jConnect for JDBC インストール・ガイド』および『リリース・ノート jConnect for JDBC』を参照してください。

さらに、メタデータ・メソッドを使用するには、接続を確立するときに `USE_METADATA` 接続プロパティを “true” (デフォルト値) に設定する必要があります。

データベース内のテンポラリ・テーブルに関するメタデータを取得することはできません。

注意 `DatabaseMetaData.getPrimaryKeys` メソッドは、テーブル定義 (`CREATE TABLE`) またはテーブル変更 (`ALTER TABLE ADD CONSTRAINT`) で宣言されたプライマリ・キーを探します。 `sp_primarykey` を使用して定義されたキーは探しません。

サーバ側メタデータのインストール

メタデータ・サポートは、クライアント (ODBC、JDBC) とデータ・ソース (サーバ・ストアド・プロシージャ) のどちらにも実装できます。jConnect にはサーバ上のメタデータをサポートする機能があり、次のような利点があります。

- jConnect のサイズを小さく保ち、インターネットからドライバをダウンロードするときの時間を短縮する。
- データ・ソースに事前にストアド・プロシージャをロードすることにより、実行時の効率が向上する。
- 柔軟性の向上により、jConnect はさまざまなデータベースに接続できる。

結果セットでのカーソルの使用方法

jConnect は、JDBC 2.0 のカーソルと更新のメソッドの多くを実装しています。これらのメソッドを利用すれば、カーソルの使用と、結果セット内の値に基づくテーブル内のローの更新が簡単になります。

JDBC 2.0 では、**ResultSets** の特性はそのタイプと同時実行性によって決まります。タイプと同時実行性の値は `java.sql.ResultSet` インタフェースの一部であり、javadocs にその説明があります。

表 2-5 に、jConnect で使用できる `java.sql.ResultSet` の特性を示します。サーバが Adaptive Server 15.0 以降である場合、要求があれば jConnect はサーバ側スクロール可能カーソルをオープンします。

表 2-5: jConnect で使用できる `java.sql.ResultSet` のオプション

	タイプ		
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
同時実行性			
<code>CONCUR_READ_ONLY</code>	サポートしている	サポートしている	使用不可
<code>CONCUR_UPDATABLE</code>	サポートしている	使用不可	使用不可

この項では、次の項目について説明します。

- [カーソルの作成](#)
- [JDBC 1.x メソッドを使用した位置付け更新と削除](#)
- [JDBC 2.0 メソッドを使用した位置付け更新と削除](#)
- [PreparedStatement オブジェクトでのカーソルの使用方法](#)
- [jConnect での TYPE_SCROLL_INSENSITIVE 結果セットの使用](#)

カーソルの作成

jConnect を使用してカーソルを作成するには、2 とおりの方法があります。

- `SybStatement.setCursorName`

`SybStatement.setCursorName` を使用して、カーソルに明示的に名前を割り当てます。`SybStatement.setCursorName` のシグニチャを次に示します。

```
void setCursorName(String name) throws SQLException;
```

- `SybStatement.setFetchSize`

`SybStatement.setFetchSize` を使用してカーソルを作成し、1 回のフェッチでデータベースから返されるローの数を指定します。

`SybStatement.setFetchSize` のシグニチャを次に示します。

```
void setFetchSize(int rows) throws SQLException;
```

`setFetchSize` を使用してカーソルを作成すると、jConnect ドライバによってカーソルの名前が設定されます。カーソルの名前を取得するには、`ResultSet.setCursorName` を使用してください。

カーソルを作成する別の方法として、文から返される **ResultSet** のタイプを指定することもできます。その場合は、次に示す JDBC の接続に対するメソッドを使用します。

```
Statement createStatement(int resultSetType, int
resultSetConcurrency)throws SQL Exception
```

タイプと同時実行性は、表 2-5 に示した **ResultSet** インタフェースのタイプと同時実行性に対応します。サポートされていない **ResultSet** を要求すると、SQL 警告が接続に関連付けられます。返された **Statement** が実行されると、要求したものに最も近いタイプの **ResultSet** が返されます。このメソッドの動作の詳細については、JDBC の仕様を参照してください。

createStatement を使用しない場合の **ResultSet** のデフォルト・タイプは次のとおりです。

- **Statement.executeQuery** だけ呼び出すと、返される **ResultSet** は **TYPE_FORWARD_ONLY** と **CONCUR_READ_ONLY** の **SybResultSet** になります。
- **setFetchSize** または **setCursorName** を呼び出すと、**executeQuery** から返される **ResultSet** は **TYPE_FORWARD_ONLY** と **CONCUR_UPDATABLE** の **SybCursorResultSet** になります。

ResultSet オブジェクトのタイプが意図したものであることを確認するには、次に示す **ResultSet** のメソッドを使用します。

```
int getConcurrency() throws SQLException;
int getType() throws SQLException;
```

❖ カーソルの作成と使用

- 1 **Statement.setCursorName** または **SybStatement.setFetchSize** を使用してカーソルを作成します。
- 2 **Statement.executeQuery** を呼び出して文に対するカーソルをオープンし、カーソル結果セットを返します。
- 3 **ResultSet.next** を呼び出してローをフェッチし、結果セット内にカーソルを位置付けます。

次の例では、カーソルを作成して結果セットを返す 2 とおりの方法を両方とも使用しています。また、**SybStatement.setFetchSize** によって作成されたカーソルの名前を取得するのに **ResultSet.setCursorName** を使用しています。

```
// With conn as a Connection object, create a
// Statement object and assign it a cursor using
// Statement.setCursorName().
Statement stmt = conn.createStatement();
stmt.setCursorName("author_cursor");

// Use the statement to execute a query and return
// a cursor result set.
```

```
ResultSet rs = stmt.executeQuery("SELECT au_id,
    au_lname, au_fname FROM authors
    WHERE city = 'Oakland'");
while(rs.next())
{
    ...
}

// Create a second statement object and use
// SybStatement.setFetchSize() to create a cursor
// that returns 10 rows at a time.
SybStatement syb_stmt = conn.createStatement();
syb_stmt.setFetchSize(10);

// Use the syb_stmt to execute a query and return
// a cursor result set.
SybCursorResultSet rs2 =
    (SybCursorResultSet)syb_stmt.executeQuery
    ("SELECT au_id, au_lname, au_fname FROM authors
    WHERE city = 'Pinole'");
while(rs2.next())
{
    ...
}

// Get the name of the cursor created through the
// setFetchSize() method.
String cursor_name = rs2.getCursorName();
...
// For jConnect 6.0, create a third statement
// object using the new method on Connection,
// and obtain a SCROLL_INSENSITIVE ResultSet.
// Note: you no longer have to downcast the
// Statement or the ResultSet.
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
ResultSet rs3 = stmt.executeQuery
    ("SELECT ... [whatever]");
// Execute any of the JDBC 2.0 methods that
// are valid for read only ResultSets.
rs3.next();
rs3.previous();
rs3.relative(3);
rs3.afterLast();
...
```

JDBC 1.x メソッドを使用した位置付け更新と削除

次の例は、JDBC 1.x のメソッドを使用して位置付け更新を実行する方法について説明します。この例では2つの **Statement** オブジェクトを作成します。1つはカーソル結果セットにローを挿入するためのもので、もう1つは結果セットのローからデータベースを更新するためのものです。

```
// Create two statement objects and create a cursor
// for the result set returned by the first
// statement, stmt1. Use stmt1 to execute a query
// and return a cursor result set.
Statement stmt1 = conn.createStatement();
Statement stmt2 = conn.createStatement();
stmt1.setCursorName("author_cursor");
ResultSet rs = stmt1.executeQuery("SELECT
    au_id, au_lname, au_fname
    FROM authors WHERE city = 'Oakland'
    FOR UPDATE OF au_lname");

// Get the name of the cursor created for stmt1 so
// that it can be used with stmt2.
String cursor = rs.getCursorName();

// Use stmt2 to update the database from the
// result set returned by stmt1.
String last_name = new String("Smith");
while(rs.next())
{
    if (rs.getString(1).equals("274-80-9391"))
    {
        stmt2.executeUpdate("UPDATE authors "+
            "SET au_lname = "+last_name +
            "WHERE CURRENT OF " + cursor);
    }
}
```

結果セット内での削除

次の例では、前述のコード例で作成した **Statement** オブジェクト *stmt2* を使用して、位置付け削除を実行します。

```
stmt2.executeUpdate("DELETE FROM authors
    WHERE CURRENT OF " + cursor);
```

JDBC 2.0 メソッドを使用した位置付け更新と削除

この項では、JDBC 2.0 のメソッドを使用して、現在のカーソル・ローにあるカラムを更新する方法と、結果セット内の現在のカーソル・ローからデータベースを更新する方法を説明します。それぞれの説明の後に例を示します。

結果セット内でのカラムの更新

JDBC 2.0 の仕様には、クライアント上でメモリ内の結果セットのカラム値を更新するための多数のメソッドが定義されています。更新された値は、基本となるデータベースで更新、挿入、削除オペレーションを実行するのに使用されます。これらのメソッドはすべて **SybCursorResultSet** クラスに実装されます。

jConnect で使用できる JDBC 2.0 更新メソッドの例をいくつか示します。

```
void updateAsciiStream(String columnName, java.io.InputStream x, int length)
    throws SQLException;
void updateBoolean(int columnIndex, boolean x) throws SQLException;
void updateFloat(int columnIndex, float x) throws SQLException;
void updateInt(String columnName, int x) throws SQLException;
void updateInt(int columnIndex, int x) throws SQLException;
void updateObject(String columnName, Object x) throws SQLException;
```

結果セットからデータベースを更新するメソッド

JDBC 2.0 の仕様には、結果セット内の現在の値に基づいてデータベースのローを更新または削除するためのメソッドが2つ定義されています。これらのメソッドの形式は JDBC 1.x の **Statement.executeUpdate** よりも単純で、カーソル名を必要としません。これらのメソッドは **SybCursorResultSet** に実装されます。

```
void updateRow() throws SQLException;
void deleteRow() throws SQLException;
```

注意 結果セットの同時実行性は **CONCUR_UPDATABLE** でなければなりません。そうでない場合は、前述のメソッドで例外が発生します。**insertRow** には、**null** 以外のエントリを必要とするすべてのテーブル・カラムを指定してください。

これらの変更がいつ参照できるかは、**DatabaseMetaData** のメソッドによって指示します。

例

次の例では、カーソル結果セットを返すのに使用される1つの **Statement** オブジェクトを作成します。結果セットの各ローについて、メモリ内でカラム値を更新し、次に、そのローの新しいカラム値を使用してデータベースを更新します。

```
// Create a Statement object and set fetch size to
// 25. This creates a cursor for the Statement
// object Use the statement to return a cursor
// result set.
SybStatement syb_stmt =
(SybStatement)conn.createStatement();
syb_stmt.setFetchSize(25);
SybCursorResultSet syb_rs =
(SybCursorResultSet)syb_stmt.executeQuery(
    "SELECT * from T1 WHERE ...")
```

```
// Update each row in the result set according to
// code in the following while loop. jConnect
// fetches 25 rows at a time, until fewer than 25
// rows are left. Its last fetch takes any
// remaining rows.
while(syb_rs.next())
{
    // Update columns 2 and 3 of each row, where
    // column 2 is a varchar in the database and
    // column 3 is an integer.
    syb_rs.updateString(2, "xyz");
    syb_rs.updateInt(3,100);
    //Now, update the row in the database.
    syb_rs.updateRow();
}
// Create a Statement object using the
// JDBC 2.0 method implemented in jConnect 6.0
Statement stmt = conn.createStatement
(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
// In jConnect 6.0, downcasting to SybCursorResultSet is not
// necessary. Update each row in the ResultSet in the same
// manner as above
while (rs.next())
{
    rs.updateString(2, "xyz");
    rs.updateInt(3,100);
    rs.updateRow();
}
// Use the Statement to return an updatable ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM T1 WHERE...");
}
```

結果セットからのローの削除

カーソル結果セットからローを削除するには、次のように
SybCursorResultSet.deleteRow を使用します。

```
while(syb_rs.next())
{
    int col3 = getInt(3);
    if (col3 >100)
    {
        syb_rs.deleteRow();
    }
}
```

結果セットへのローの挿入

次の例は、JDBC 2.0 API を使用して挿入を行う方法を示します。
SybCursorResultSet にダウンキャストする必要はありません。

```
// prepare to insert
rs.moveToInsertRow();
// populate new row with column values
rs.updateString(1, "New entry for col 1");
rs.updateInt(2, 42);
// insert new row into db
rs.insertRow();
// return to current row in result set
rs.moveToCurrentRow();
```

PreparedStatement オブジェクトでのカーソルの使用方法

PreparedStatement を一度作成すれば、入力パラメータに同じ値や異なる値を指定して、何度も使用できます。カーソルとともに **PreparedStatement** オブジェクトを使用する場合は、使用が終わるたびにカーソルをクローズして、次に使用するとき再度オープンする必要があります。カーソルは結果セットをクローズするとクローズされます (**ResultSet.close**)。カーソルの prepared 文を実行すると、カーソルがオープンされます (**PreparedStatement.executeQuery**)。

次の例は **PreparedStatement** オブジェクトを作成し、カーソルを割り当て、**PreparedStatement** オブジェクトを2度実行してカーソルのクローズと再オープンを行う方法を示します。

```
// Create a prepared statement object with a
// parameterized query.
PreparedStatement prep_stmt =
conn.prepareStatement(
"SELECT au_id, au_lname, au_fname "+
"FROM authors WHERE city = ? "+
"FOR UPDATE OF au_lname");

//Create a cursor for the statement.
prep_stmt.setCursorName("author_cursor");

// Assign the parameter in the query a value.
// Execute the prepared statement to return a
// result set.
prep_stmt.setString(1, "Oakland");
ResultSet rs = prep_stmt.executeQuery();

//Do some processing on the result set.
while(rs.next())
{
    ...
}
```

```
// Close the result, which also closes the cursor.
rs.close();

// Execute the prepared statement again with a new
// parameter value.
prep_stmt.setString(1,"San Francisco");
rs = prep_stmt.executeQuery();
// reopens cursor
```

jConnect での TYPE_SCROLL_INSENSITIVE 結果セットの使用

jConnect は TYPE_SCROLL_INSENSITIVE の結果セットだけをサポートします。

jConnect は、Sybase 独自のプロトコルである Tabular Data Stream (TDS) を使用して Sybase データベース・サーバと通信します。Adaptive Server 15.0 以降は TDS スクロール可能カーソルをサポートします。TDS スクロール可能カーソルをサポートしないサーバのために、jConnect は、**ResultSet.next** の呼び出しのたびに、要求されたロー・データをクライアント上にキャッシュします。しかし、結果セットの最後に到達したときは、結果セット全体がクライアントのメモリに格納されています。これによってパフォーマンス上の問題が発生するので、TYPE_SCROLL_INSENSITIVE の結果セットは、Adaptive Server 15.0 のみに使用するか、または結果セットが比較的小さい場合のみに使用することをおすすめします。

注意 TYPE_SCROLL_INSENSITIVE **ResultSets** を jConnect で使用するとき、サーバが TDS スクロール可能カーソルをサポートしていない場合は、**ResultSet** の最後のローを読み出した後でなければ **isLast** メソッドを呼び出すことはできません。最後のローに到達する前に **isLast** を呼び出すと、**UnimplementedOperationException** が発生します。

jConnect の *sample2* ディレクトリに **ExtendResultSet** があります。このサンプルは、JDBC 1.0 インタフェースを使用して、制限付きの TYPE_SCROLL_INSENSITIVE **ResultSet** を作成します。

この実装は標準の JDBC 1.0 メソッドを使用して、スクロールの影響を受けない読み込み専用の結果セット、つまり、結果セットが開かれている間に行われた変更の影響を受けない、元のデータの静的ビューを生成します。**ExtendedResultSet** は、**ResultSet** のローをすべてクライアント上にキャッシュします。このクラスを大きな結果セットに対して使用する場合は、注意が必要です。

sample.ScrollableResultSet について次に説明します。

- JDBC 1.0 の java.sql.ResultSet の拡張機能です。
- JDBC 2.0 java.sql.ResultSet と同じシグニチャを持つ追加のメソッドを定義します。
- JDBC 2.0 のメソッドがすべて含まれているわけではありません。ここに含まれていないメソッドは、ResultSet を修正して対処します。

JDBC 2.0 API からの定義されているメソッドを次に示します。

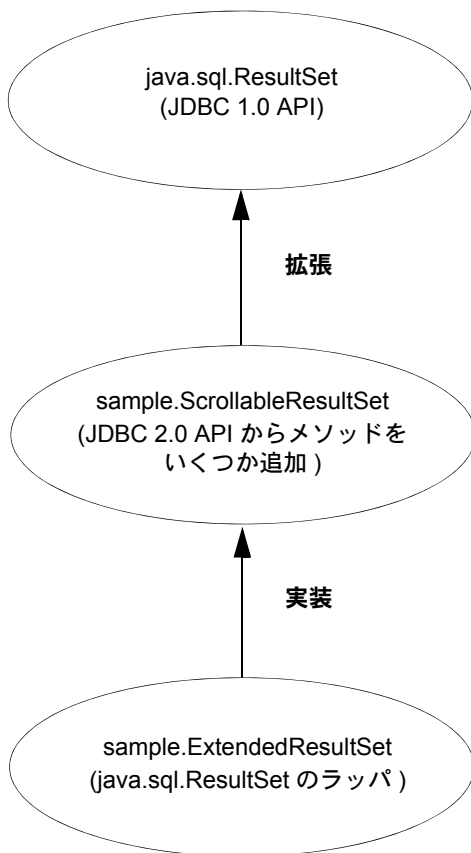
```
boolean previous() throws SQLException;
boolean absolute(int row) throws SQLException;
boolean relative(int rows) throws SQLException;
boolean first() throws SQLException;
boolean last() throws SQLException;
void beforeFirst() throws SQLException;
void afterLast() throws SQLException;
boolean isFirst() throws SQLException;
boolean isLast() throws SQLException;
boolean isBeforeFirst() throws SQLException;
boolean isAfterLast() throws SQLException;
int getFetchSize() throws SQLException;
void setFetchSize(int rows) throws SQLException;
void setFetchDirection(int direction) throws SQLException;
int getType() throws SQLException;
int getConcurrency() throws SQLException;
int getRow() throws SQLException;
```

サンプル・クラスを使用するには、任意の JDBC 1.0 java.sql.ResultSet を使用して ExtendedResultSet を作成します。関連するコードの部分を次に示します (Java 1.1 環境を想定しています)。

```
// import the sample files
import sample.*;
//import the JDBC 1.0 classes
import java.sql.*;
// connect to some db using some driver;
// create a statement and a query;
// Get a reference to a JDBC 1.0 ResultSet
ResultSet rs = stmt.executeQuery(_query);
// Create a ScrollableResultSet with it
ScrollableResultSet srs = new ExtendedResultSet(rs);
// invoke methods from the JDBC 2.0 API
srs.beforeFirst();
// or invoke methods from the JDBC 1.0 API
if (srs.next())
    String column1 = srs.getString(1);
```

図 2-1 はサンプル・クラスと JDBC API の関係を示します。

図 2-1: クラス図



詳細については、<http://java.sun.com/products/jdbc/jdbcse2.html> の JDBC 2.0 API を参照してください。

COMPUTE 句での Transact-SQL クエリの使用

jConnect for JDBC は、COMPUTE 句を含む Transact-SQL クエリをサポートしています。COMPUTE 句を使用すると、ディテールと計算結果を1つの **select** 文で参照できます。計算ローは、特定グループのディテール・ローの後に表示されます。たとえば、次のように使用します。

```
select type, price, advance
  from titles
  order by type
  compute sum(price), sum(advance) by type
```

type	price	advance
UNDECIDED	NULL	NULL

Compute Result:

NULL	NULL
------	------

type	price	advance
business	2.99	10,125.00
business	11.95	5,000.00
business	19.99	5,000.00
business	19.99	5,000.00

Compute Result:

54.92	25,125.00
-------	-----------

...

...

(24 rows affected)

jConnect が **COMPUTE** 句を含む **select** 文を実行すると、クライアントには複数の結果セットが返されます。結果セットの数は、使用できるユニークなグループの数によって異なります。各グループには、ディテール・ローごとに1つの結果セット、および1つの計算結果セットが含まれています。クライアントは、返されたローのすべての結果セットを処理する必要があります。すべてを処理しない場合は、返される最初の結果セットには最初のデータ・グループのディテール・ローだけが含まれます。

COMPUTE 句の詳細については、Adaptive Server Enterprise の『Transact-SQL ユーザーズ・ガイド』を参照してください。複数の結果セットを処理する方法の詳細については、Sun Microsystems の Web サイトで JDBC API のマニュアルを参照してください。

バッチ更新のサポート

バッチ更新を使用すると、Statement オブジェクトで複数の update コマンドを 1 つの単位 (バッチ) として基本のデータベースに送信し、一度に処理することができます。

注意 バッチ更新を使用するには、jConnect インストール・ディレクトリの *sp* ディレクトリにある最新のメタデータ・スクリプトをインストールする必要があります。

Statement、PreparedStatement、CallableStatement を使用してバッチ更新を行う例については、*sample2* サブディレクトリにある *BatchUpdates.java* を参照してください。

jConnect はバッチでの動的 PreparedStatements もサポートしています。

実装上の注意

jConnect は JDBC 2.0 API の仕様に従ってバッチ更新を実装しますが、次の例外があります。

- `BatchUpdateException.getUpdateCounts` の実装に関する JDBC 2.0 の標準が今後修正または緩和された場合も、jConnect は引き続き当初の標準を実装します。つまり、`BatchUpdateException.getUpdateCounts` は長さ $M < N$ の `int[]` を返しますが、これはバッチの最初の M 個の文が成功し、 $M+1$ 番目の文が失敗し、 $M+2$ 番目から N 番目までの文が実行されなかったことを示します。“ N ” はバッチ内の文の総数です。
- バッチ (非連鎖) モードでストアド・プロシージャを呼び出すには、そのストアド・プロシージャを非連鎖モードで作成する必要があります。詳細については、「[ストアド・プロシージャを非連鎖トランザクション・モードでしか実行できない](#)」(130 ページ) を参照してください。
- Adaptive Server でバッチ実行中にエラーが発生すると、`BatchUpdateException.getUpdateCounts` は長さ 0 の `int[]` だけを返します。エラーが発生した場合はトランザクション全体がロールバックされ、成功したローの数 は 0 になります。

注意 重複キーを持つローの挿入というエラーの場合は、トランザクションはロールバックされません。

- Adaptive Server では、重複キーを持つローを挿入しても、バッチ文が終了してロールバックすることはありません。**cancel** が発行されるか、バッチが完了するか、重複キーを持つローの挿入以外のエラーが発生するまで、サーバはバッチ内の文の処理を続けます。jConnect は、バッチ処理中に例外 (重複キーを持つローの挿入を含む) を検出するとサーバに **cancel** を送信するので、サーバが **cancel** を受け取る前にバッチのどの部分までが実行されたかを正確に判定することはできません。したがって、JDBC 仕様に従い、**autoCommit** を “false” に設定したトランザクション内でバッチを実行することを強くおすすめします。このようにすれば、トランザクションをロールバックし、データベースを既知の状態に戻してからバッチを再実行することができます。
- バッチ更新をサポートしていないデータベースでのバッチ更新 – データベースがバッチ更新をサポートしていなくても、jConnect はバッチ更新を **executeUpdate** ループ内で実行します。これによって、どのデータベースを参照するかに関係なく、同じバッチ・コードを使用できます。

バッチ更新の詳細については、JDBC API documentation (<http://java.sun.com>) を参照してください。

ストアド・プロシージャの結果セットからのデータベースの更新

jConnect では、**update** メソッドと **delete** メソッドを使用すると、ストアド・プロシージャによって返される結果セットに対するカーソルを取得することができます。このカーソルの位置を使用することによって、結果セットを返したテーブル内のローの更新や削除を行うことができます。これらのメソッドは **SybCursorResultSet** にあります。

```
void updateRow(String tableName) throws SQLException;
```

```
void deleteRow(String tableName) throws SQLException;
```

tableName パラメータには、結果セットを介したデータベース・テーブルを指定します。

ストアド・プロシージャによって返される結果セットに対するカーソルを取得するには、そのプロシージャが含まれる呼び出し可能な文を実行する前に、**SybCallableStatement.setCursorName** または **SybCallableStatement.setFetchSize** を実行する必要があります。次の例は、ストアド・プロシージャの結果セットに対するカーソルを作成し、結果セット内の値を更新してから、**SybCursorResultSet.update** メソッドを使用して基本となるテーブルを更新する方法を示します。

```
// Create a CallableStatement object for executing the stored
// procedure.
CallableStatement sproc_stmt =
    conn.prepareCall("{call update_titles}");

// Set the number of rows to be returned from the database with
// each fetch. This creates a cursor on the result set.
(SybCallableStatement)sproc_stmt.setFetchSize(10);

//Execute the stored procedure and get a result set from it.
SybCursorResultSet sproc_result = (SybCursorResultSet)
    sproc_stmt.executeQuery();

// Move through the result set row by row, updating values in the
// cursor's current row and updating the underlying titles table
// with the modified row values.
while(sproc_result.next())
{
    sproc_result.updateString(...);
    sproc_result.updateInt(...);
    ...
    sproc_result.updateRow(titles);
}
```

データ型の作業

この項では、numeric 型、image 型、text 型、date 型、time 型、char 型のデータの使用方法について説明します。

NUMERIC データの送信

SybPreparedStatement 拡張機能により、精度 (総桁数) と位取り (小数点以下の桁数) を指定できる NUMERIC データ型を、Adaptive Server で処理する方法がサポートされています。

NUMERIC データ型と Java での対応するデータ型 (java.math.BigDecimal) は若干異なります。この相違により、jConnect アプリケーションが setBigDecimal メソッドを使用して入出力パラメータの値を制御するときに問題が発生することがあります。特に、対応する SQL オブジェクトがストアド・プロシージャかカラムかを問わず、その精度と位取りが、パラメータの精度と位取りに完全に一致しなければならない場合があります。

SybPreparedStatement 拡張機能を次のメソッドとともに使用することで、jConnect アプリケーションで setBigDecimal をさらに強力に制御できます。

```
public void setBigDecimal (int parameterIndex, BigDecimal X, int scale,
    int precision) throws SQLException
```

詳細については、jConnect インストール・ディレクトリの /sample2 サブディレクトリにあるサンプル SybPrepExtension.java を参照してください。

データベース内の image データの更新

jConnect の `TextPointer` クラスには、Adaptive Server または SQL Anywhere データベース内の `image` カラムを更新するための `sendData` メソッドがあります。以前のバージョンの jConnect では、`image` データの送信に `java.sql.PreparedStatement` の `setBinaryStream` メソッドを使用しなければなりませんでした。`TextPointer.sendData` メソッドは `java.io.InputStream` を使用しており、`image` データを Adaptive Server データベースに送信するときのパフォーマンスが大幅に向上します。

警告！ `TextPointer` は標準 JDBC 形式ではないため、`TextPointer` クラスを `sendData()` メソッドで使用すると、アプリケーションに影響を及ぼす場合があります。

`image` データを送信する場合は、`PreparedStatement.setBinaryStream(int paramIndex, InputStream image)` を標準 JDBC 形式として使用することをおすすめします。ただし、大容量の `image` データを扱うときに `setBinaryStream()` を使用すると、プロシージャ・キャッシュで `TextPointer` クラスよりもメモリを消費する可能性があります。

`TextPointer` クラスは、代わりとなるものが実装されるまではサポートされる予定です。

`TextPointer` クラスのインスタンスを取得するには、`SybResultSet` の次の 2 つの `getTextPtr` メソッドのいずれかを使用します。

```
public TextPointer getTextPtr(String columnName)
public TextPointer getTextPtr(int columnIndex)
```

TextPointer クラスのパブリック・メソッド

`com.sybase.jdbcx` パッケージには `TextPointer` クラスが含まれています。このパブリック・メソッド・インタフェースを次に示します。

```
public void sendData(InputStream is, boolean log)
    throws SQLException
public void sendData(InputStream is, int length,
    boolean log) throws SQLException
public void sendData(InputStream is, int offset,
    int length, boolean log) throws SQLException
public void sendData(byte[] byteInput, int offset,
    int length, boolean log) throws SQLException
```

各パラメータの意味は、次のとおりです。

- `sendData(InputStream is, boolean log)` — 指定された入力ストリーム内のデータで `image` カラムを更新します。
- `sendData(InputStream is, int length, boolean log)` — 指定された入力ストリーム内のデータで `image` カラムを更新します。`length` は送信されるバイト数です。

- `sendData(InputStream is, int offset, int length, boolean log)` — 指定された入力ストリーム内のデータで **image** カラムを更新します。更新は `offset` パラメータで指定されたバイト・オフセットから始まり、`length` パラメータで指定されたバイト数まで行われます。
- `sendData(byte[] byteInput, int offset, int length, boolean log)` — `byteInput` パラメータで指定されたバイト配列に格納されている **image** データでカラムを更新します。更新は `offset` パラメータに指定されたバイト・オフセットから開始され、`length` パラメータに指定されたバイト数まで続行されます。
- 各メソッドの `log` パラメータは、**image** データ全体をデータベース・トランザクション・ログに記録するかどうかを指定します。`log` パラメータが“true”に設定されている場合は、バイナリ・イメージ全体がトランザクション・ログに書き込まれます。`log` パラメータが“false”に設定されている場合は、更新はログに記録されますが、**image** データそのものは記録されません。

❖ **TextPointer.sendData** による **image** カラムの更新

image データを使用してカラムを更新するには、次の手順に従います。

- 1 更新するローとカラムに対する **TextPointer** オブジェクトを取得します。
- 2 **TextPointer.sendData** を使用して更新を実行します。

次の2つの項で例を示します。この例では、**pubs2** データベースにある **au_pix** テーブルの **pic** カラムを更新するために、ファイル *Anne_Ringer.gif* から **image** データが送信されます。更新は author ID 899-46-2035 のローに対して行われます。

TextPointer オブジェクトの取得

text カラムと **image** カラムには、このカラムの **text** データや **image** データとは別に **timestamp** とページ位置情報が格納されています。データが **text** カラムまたは **image** カラムから選択されるとき、この情報は結果セットの中では隠されます。

image カラムを更新するための **TextPointer** オブジェクトはこの隠された情報を必要としますが、カラム・データの **image** 部分は必要としません。この情報を取得するには、そのカラムを選択して **ResultSet** オブジェクトに出力した後で、**SybResultSet.getTextPtr** を使用します。このメソッドはテキスト・ポインタ情報を取り出し、**image** データは無視して **TextPointer** オブジェクトを作成します。使用例については、この後のコードを参照してください。

カラムに格納されている **image** データのサイズが大きい場合に、1 つ以上のローからそのカラムを検索し、データがすべて取得されるまで待つのは効率的ではありません。そのデータは使用されないからです。この処理時間を短縮するには、**set textsize** コマンドを使用して、パケットで返されるデータ量をできるだけ少なくします。次の、**TextPointer** オブジェクトを取得するコード例では、この目的で **set textsize** を使用しています。


```
/*
 * Define a string for selecting pic column data for author ID
 * 899-46-2035.
 */
String getColumnData = "select pic from au_pix where au_id = '899-46-2035'";

/*
 * Use set textsize to return only a single byte of column data
 * to a Statement object. The packet with the column data will
 * contain the "hidden" information necessary for creating a
 * TextPointer object.
 */
Statement stmt= connection.createStatement();
stmt.executeUpdate("set textsize 1");

/*
 * Select the column data into a ResultSet object--cast the
 * ResultSet to SybResultSet because the getTextPtr method is
 * in SybResultSet, which extends ResultSet.
 */
SybResultSet rs = (SybResultSet)stmt.executeQuery(getColumnData);

/*
 * Position the result set cursor on the returned column data
 * and create the desired TextPointer object.
 */
rs.next();
TextPointer tp = rs.getTextPtr("pic");

/*
 * Now, assuming we are only updating one row, and won't need
 * the minimum textsize set for the next return from the server,
 * we reset textsize to its default value.
 */
stmt.executeUpdate("set textsize 0");
```

TextPointer.sendData を 使用した更新の実行

次のコード例では、前述の項で作成した **TextPointer** オブジェクトを使用して、ファイル *Anne_Ringer.gif* 内の **image** データで **pic** カラムを更新します。

```
/*
 *First, define an input stream for the file.
 */
FileInputStream in = new FileInputStream("Anne_Ringer.gif");

/*
 * Prepare to send the input stream without logging the image data
 * in the transaction log.
 */
boolean log = false;

/*
 * Send the image data in Anne_Ringer.gif to update the pic
 * column for author ID 899-46-2035.
 */
tp.sendData(in, log);
```

詳細については、jConnect インストール・ディレクトリの *sample2* サブディレクトリにあるサンプル *TextPointers.java* を参照してください。

text データの使用

以前のバージョンの jConnect では、Adaptive Server または SQL Anywhere データベース内の **text** カラムを更新するのに、**TextPointer** クラスと **sendData** メソッドを使用していました。

TextPointer クラスはすでに非推奨となっており、Java の今後のバージョンでは削除される可能性があります。

データ・サーバが Adaptive Server または SQL Anywhere の場合は、text データの送信には標準 JDBC 形式を使用してください。

```
PreparedStatement.setAsciiStream(int paramIndex,
    InputStream text, int length)
```

または

```
PreparedStatement.setUnicodeStream(int paramIndex,
    InputStream text, int length)
```

または

```
PreparedStatement.setCharacterStream(int paramIndex,
    Reader reader, int length)
```

date データ型と time データ型の使用方法

Adaptive Server は、SQL のデータ型である `datetime`、`smalldatetime`、`date`、`time` をサポートしています。`date` と `time` には次の利点があります。

- 日付値の範囲は 0001 年 1 月 1 日 ~ 9999 年 12 月 31 日であり、`java.sql.Date` で可能な値の範囲と完全に一致します。
- `java.sql.Date` と `date` データ型、`java.sql.Time` と `time` データ型が直接マッピングされます。

実装上の注意

- `date` カラムまたは `time` カラムを持つテーブルからの検索を行うときに、`jConnect` で `date` / `time` がサポートされるように設定されていない場合は (設定するにはバージョンを設定します)、サーバは `date` / `time` を `datetime` 値に変換して返そうとします。このため、返される日付が 1753 年 1 月 1 日よりも前であるときに、問題が発生する可能性があります。その場合は、変換エラーとなり、データベースからエラーが通知されます。
- SQL Anywhere は `date` データ型と `time` データ型をサポートしていますが、これらのデータ型と Adaptive Server バージョン 12.5.1 以降の `date` データ型および `time` データ型との間には、まだ直接の互換性はありません。`jConnect` を使用して SQL Anywhere と通信するときは、引き続き `datetime` データ型と `smalldatetime` データ型を使用してください。
- SQL Anywhere での `datetime` カラムの最大値は 7911 年 1 月 1 日 00:00:00 です。
- `jConnect` を使用して `datetime` 型のカラムまたはパラメータに 1753 年 1 月 1 日より前の日付を挿入しようとする、変換エラーが通知されます。
- `date` データ型と `time` データ型の詳細については、Adaptive Server のマニュアルを参照してください。特に、許容される暗黙の変換についての項を参照してください。
- Adaptive Server の `date`、`time`、または `datetime` のカラムに対して `getObject` を使用した場合の戻り値のデータ型は、それぞれ `java.sql.Date`、`java.sql.Time`、`java.sql.Timestamp` となります。

char / varchar / text データ型と getByte の使用

データが 16 進数、8 進数、10 進数の場合以外は、`char`、`univarchar`、`unichar`、`varchar`、または `text` のフィールドに対して `rs.getBytes` を使用しないでください。

高度な機能の実装

この項では、jConnect の高度な機能を使用する方法に関する次の項目について説明します。

- [BCP 挿入の使用](#)
- [サポートされている Adaptive Server クラスタ・エディションの機能](#)
- [イベント通知の使用](#)
- [エラー・メッセージの処理](#)
- [マイクロ秒の精度の time データ](#)
- [パスワードの暗号化の使用](#)
- [テーブル内のカラム・データとしての Java オブジェクトの格納](#)
- [動的クラス・ロードの使用](#)
- [JDBC 4.0 仕様のサポート](#)
- [JDBC 3.0 仕様のサポート](#)
- [JDBC 2.0 オプショナル・パッケージ拡張サポート](#)

BCP 挿入の使用

jConnect for JDBC では、Adaptive Server バージョン 12.5.2 以降に、バルク・ロード挿入を使用してローを大量に挿入できます。この機能ではサーバを設定する必要は特にありませんが、大きいページ・サイズ、ネットワーク・パケット・サイズ、最大メモリ・サイズにより、パフォーマンスは大幅に向上します。クライアント・メモリに応じてバッチ・サイズを大きくすることでもパフォーマンスが向上します。

この機能を有効にするには、ENABLE_BULK_LOAD を True に設定します。準備文を使用して ENABLE_BULK_LOAD を True にすると、jConnect は BULK ルーチンを使用して、レコードのバッチを Sybase データベースに挿入します。

BCP 挿入では以下をサポートしていません。

- unsigned 型の bigint および unitext
- 暗号化カラムと計算カラム

サポートされている Adaptive Server クラスタ・エディションの機能

jConnect は Adaptive Server クラスタ・エディション環境をサポートします。この環境では、複数の Adaptive Server が共有ディスクのセットと高速プライベート相互接続に接続します。この場合、複数の物理ホストと論理ホストを使用して、Adaptive Server を拡張できます。

クラスタ・エディションの詳細については、『Adaptive Server Enterprise Cluster ユーザーズ・ガイド』を参照してください。

ログインのリダイレクト

クラスタ・エディション環境では一般に、常にサーバ間で処理負荷の不均衡が発生しています。ビジー状態のサーバに対してクライアント・アプリケーションが接続を試みた場合、ログインのリダイレクト機能によって、サーバの負荷バランスが調整されます。具体的には、クラスタ内の負荷が少ない別サーバに対して、クライアント接続がリダイレクトされます。ログインのリダイレクトが発生するのはログイン・シーケンス中であり、リダイレクトが発生したことは、クライアント・アプリケーションには通知されません。ログインのリダイレクト機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 クライアントをリダイレクトするように設定されているサーバに対してクライアント・アプリケーションが接続すると、ログインに時間がかかる場合があります。これは、クライアント接続が別サーバにリダイレクトされるたびに、ログイン・プロセスが再開されるからです。

接続マイグレーション

接続マイグレーション機能を使用すると、クラスタ・エディション環境内のサーバは動的に負荷を分散できます。さらに、既存のクライアント接続とそのコンテキストをクラスタ内の別サーバにシームレスにマイグレートできます。この機能によって、クラスタ・エディション環境では、最適なりソース配分と処理時間の短縮が実現します。サーバ間のマイグレーションはシームレスに行われるので、接続マイグレーション機能は、可用性の高い「ダウン時間ゼロ」の環境を構築する場合にも役立ちます。接続マイグレーション機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 接続マイグレーション中には、コマンドの実行に時間がかかる場合があります。状況に応じて、コマンドのタイムアウト値を増やすことをおすすめします。

接続フェールオーバー

接続フェールオーバー機能を使用すると、停電やソケットの障害など、予想外の原因でプライマリ・サーバが使用不可になった場合に、クライアント・アプリケーションは接続先を別の Adaptive Server に切り替えることができます。クラスタ環境では、クライアント・アプリケーションは動的なフェールオーバー・アドレスを使用して、複数のサーバに対して何度もフェールオーバーできます。

高可用性オプションが有効な場合、フェールオーバー・ターゲットの候補をクライアント・アプリケーションにあらかじめ設定しておく必要はありません。Adaptive Server は、クラスタ・メンバシップ、論理クラスタの使用状況、負荷分散などに基づいて、最適なフェールオーバー・リストを常にクライアントに提供します。クライアントは、フェールオーバー時にフェールオーバー・リストの順序付けを参照して、再接続を試みます。ドライバがサーバに正常に接続した場合は、返されたリストに基づいて、ホスト値のリストが内部的に更新されます。それ以外の場合は、接続失敗例外が発生します。

接続のフェールオーバーの有効化

接続文字列を使用して接続フェールオーバーを有効にするには、REQUEST_HA_SESSION を true に設定します。次に例を示します。

```
URL="jdbc:sybase:Tds:server1:port1,server2:port2,...,  
serverN:portN/mydb?REQUEST_HA_SESSION=true"
```

server1:port1、server2:port2、...、serverN:portN の部分は、フェールオーバー・リストの順序付けです。

接続を確立する際、jConnect はフェールオーバー・リストで指定されている最初のホストとポートに接続を試みます。接続に失敗した場合は、接続が確立されるまで、またはリストの最後に達するまで、リストに表示された順に接続を試みます。

注意 接続文字列で指定された代替サーバのリストは、初期接続時にのみ使用されます。使用可能なインスタンスとの接続の確立後、高可用性をサポートしているクライアントは、最適なフェールオーバー・ターゲットを含む最新のリストをサーバから受信します。この新しいリストは、指定されたリストを上書きします。

イベント通知の使用方法

jConnect のイベント通知機能を使用すると、Open Server プロシージャが実行されるときにアプリケーションが通知を受けとることができます。

この機能を使用するには、**Connection** インタフェースを拡張した **SybConnection** クラスを使用する必要があります。**SybConnection** には、イベント通知をオンにするための **regWatch** メソッドと、イベント通知をオフにするための **regNoWatch** メソッドがあります。

アプリケーション側では、**SybEventHandler** インタフェースも実装する必要があります。このインタフェースには、1 つのパブリック・メソッド **void event(String proc_name, ResultSet params)** があり、指定されたイベントが発生するとこのメソッドが呼び出されます。イベントのパラメータは **event** に渡され、アプリケーションに応答方法を通知します。

アプリケーションでイベント通知を使用するには、**SybConnection.regWatch()** を呼び出して、アプリケーションをレジスタード・プロシージャの通知リストに登録します。使用する構文

```
SybConnection.regWatch(proc_name,eventHdlr,option)
```

各パラメータの意味は、次のとおりです。

- *proc_name* は、通知を生成するレジスタード・プロシージャの名前を示す文字列です。
- *eventHdlr* は、実装する **SybEventHandler** クラスのインスタンスです。
- *option* は、**NOTIFY_ONCE** または **NOTIFY_ALWAYS** のいずれかです。
NOTIFY_ONCE は、プロシージャが初めて実行されるときにだけアプリケーションが通知を受け取るようにする場合に使用します。
NOTIFY_ALWAYS は、プロシージャが実行されるたびにアプリケーションが通知を受け取るようにする場合に使用します。

指定された *proc_name* のイベントが Open Server 上で発生するたびに、jConnect は別のスレッドから **eventHdlr.event** を呼び出します。**eventHdlr.event** が実行されるときに、イベントのパラメータが渡されます。これは別のスレッドなので、イベント通知がアプリケーションの実行をブロックすることはありません。

proc_name がレジスタード・プロシージャでない場合や、Open Server がクライアントを通知リストに追加できない場合は、**regWatch** を呼び出すと SQL 例外が発生します。

イベント通知をオフにするには、次の呼び出しを使用します。

```
SybConnection.regNoWatch(proc_name)
```

警告！ Sybase のイベント通知拡張機能をアプリケーションで使用する場合は、**regWatch** の最初の呼び出しによって作成された子スレッドを削除するために、接続に対して **close** メソッドを呼び出す必要があります。これを実行しないと、アプリケーションを終了するときに仮想マシンがハングすることがあります。

イベント通知の例

次の例は、イベント・ハンドラを実装し、接続後にイベント・ハンドラのインスタンスでイベントを登録する方法を示します。

```
public class MyEventHandler implements SybEventHandler
{
    // Declare fields and constructors, as needed.
    ...
    public MyEventHandler(String eventname)
    {
        ...
    }

    // Implement SybEventHandler.event.
    public void event(String eventName, ResultSet params)
    {
        try
        {
            // Check for error messages received prior to event
            // notification.
            SQLWarning sqlw = params.getWarnings();
            if (sqlw != null)
            {
                // process errors, if any
                ...
            }
            // process params as you would any result set with
            // one row.
            ResultSetMetaData rsmd = params.getMetaData();
            int numColumns = rsmd.getColumnCount();
            while (params.next()) // optional
            {
                for (int i = 1; i <= numColumns; i++)
                {
                    System.out.println(rsmd.getColumnName(i) + " = "
                        + params.getString(i));
                }
                // Take appropriate action on the event. For example,
                // perhaps notify application thread.
                ...
            }
        }
        catch (SQLException sqe)
        {
            // process errors, if any
            ...
        }
    }
}

public class MyProgram
```



```

{
    ...
    // Get a connection and register an event with an instance
    // of MyEventHandler.
    Connection conn = DriverManager.getConnection(...);
    MyEventHandler myHdlr = new MyEventHandler("MY_EVENT");

    // Register your event handler.
    ((SybConnection)conn).regWatch("MY_EVENT", myHdlr,
        SybEventHandler.NOTIFY_ALWAYS);
    ...
    conn.regNoWatch("MY_EVENT");
    conn.close();
}

```

エラー・メッセージの処理

jConnect には、Sybase 固有のエラー情報を返すための **SybSQLException** と **SybSQLWarning** の 2 つのクラス、および jConnect がサーバから受信したエラー・メッセージを処理する方法をカスタマイズするための **SybMessageHandler** インタフェースがあります。

警告として返される数値エラーの処理

Adaptive Server 12.0 ～ 12.5 では、数値エラーがデフォルトでは重大度 10 として扱われます。重大度 10 のメッセージは、エラーではなくステータス情報メッセージに分類され、その内容は **SQLWarning** オブジェクトに転送されます。次のコードは、この処理を抜粋したものです。

```

static void processWarnings(SQLWarning warning)
{
    if (warning != null)
    {
        System.out.println ("¥n -- Warning received -- ¥n");
    } //end if
    while (warning != null)
    {
        System.out.println ("Message: " + warning.getMessage());
        System.out.println ("SQLState: " + warning.getSQLState());
        System.out.println ("ErrorCode: " + warning.getErrorCode());
        System.out.println ("-----");
        warning = warning.getNextWarning();
    } //end while
} //end processWarnings

```

数値エラーが発生したときは、結果セット・データを含まない `ResultSet` オブジェクトが返され、エラーに関する情報が `SQLWarning` から取得される必要があります。そのため、JDBC アプリケーションでは、`SQLWarning` の確認と処理を結果セットのある場所に依存しないようにしてください。たとえば、次のコードでは、`while` ループを処理するために結果セットの内と外の両方で `SQLWarning` データを確認し、処理します。

```
while (rs.next())
{
    String value = rs.getString(1);
    System.out.println ("Fetched value: " + value);

    // Check for SQLWarning on the result set.
    processWarnings (rs.getWarnings());

} //end while

// Check for SQLWarning on the result set.
processWarnings (rs.getWarnings());
```

ここで、コードは結果セット・データがない場合でも `SQLWarning` を確認します (`rs.next()` が `false` の場合)。次の例は、数値エラーを検出して報告するために適切に記述されたプログラムの出力です。エラーはゼロによる除算です。

```
-- Warning received --

Message: Divide by zero occurred.
SQLState: 01012
ErrorCode: 3607
```

Sybase 固有のエラー情報の取得

`jdbcConnect` の `EedInfo` インタフェースに、Sybase 固有のエラー情報を取得するためのメソッドが定義されています。`EedInfo` インタフェースは `SQLException` クラスと `SQLWarning` クラスを拡張する `SybSQLException` と `SybSQLWarning` に実装されています。

`SybSQLException` と `SybSQLWarning` には次のメソッドがあります。

- `public ResultSet getEedParams` — エラー・メッセージに付随するパラメータ値が格納された、1 ロウの結果セットを返します。
- `public int getStatus` — メッセージ内にパラメータ値がある場合は “1” を返し、ない場合は “0” を返します。
- `public int getLineNumber` — エラー・メッセージを引き起こしたストアード・プロシージャまたはクエリの行番号を返します。
- `public String getProcedureName` — エラー・メッセージを引き起こしたプロシージャの名前を返します。

- `public String getServerName` — エラー・メッセージを生成したサーバの名前を返します。
- `public int getSeverity` — エラー・メッセージの重大度を返します。
- `public int getState` — サーバ内のエラー・メッセージの内部ソースに関する情報を返します。これは、Sybase 製品の保守契約を結んでいるサポート・センタだけが使用します。
- `public int getTranState` — 次のいずれかのトランザクション・ステータスを返します。
 - 0 — 接続は現在拡張トランザクションにあります。
 - 1 — 直前のトランザクションは正常にコミットされました。
 - 3 — 直前のトランザクションはアボートされました。

エラー・メッセージの中には、`SySQLException` または `SybsQLWarning` とはならず、`SQLException` や `SQLWarning` となるものもあります。アプリケーション側では、処理している例外の型を確認してから `SySQLException` または `SybsQLWarning` にダウンキャストするようにしてください。

エラー・メッセージ処理のカスタマイズ

`SybMessageHandler` インタフェースを使用すると、サーバによって生成されたエラー・メッセージを `jConnect` が処理する方法をカスタマイズできます。エラー・メッセージを処理するための独自のクラスで `SybMessageHandler` を実装すると、次のような利点があります。

- 「ユニバーサルな」エラー処理
エラー処理論理を、アプリケーション全体で何度も記述する代わりに、エラー・メッセージ・ハンドラの中に置くことができます。
- 「ユニバーサルな」エラー・ロギング
エラー・メッセージ・ハンドラに、すべてのエラー・ロギングを処理するためのロジックを組み込むことができます。
- アプリケーションの要件に基づいた、エラー・メッセージ重大度の再マッピング

エラー・メッセージ・ハンドラには、特定のエラー・メッセージを認識して、その重大度をサーバの重大度レベルではなく、アプリケーションが重視する点に基づいてダウングレードまたはアップグレードするためのロジックを組み込むことができます。たとえば、古いローを削除するクリーンアップ・オペレーションを行っている間は、ローが存在しないというメッセージの重大度をダウングレードしますが、その他の状況では重大度をアップグレードします。

注意 `SybMessageHandler` インタフェースを実装するエラー・メッセージ・ハンドラは、サーバによって生成されたメッセージだけを受け取ります。`jConnect` によって生成されたメッセージは処理しません。

`jConnect` は、エラー・メッセージを受け取ると、メッセージを処理するための `SybMessageHandler` クラスが追加されているかどうかを調べます。追加されている場合は、SQL 例外を引数として受け取る `messageHandler` メソッドを呼び出し、`messageHandler` から返された値に基づいてメッセージを処理します。エラー・メッセージ・ハンドラは次のことを行います。

- SQL 例外をそのまま返します。
- `null` を返します。結果として、`jConnect` はメッセージを無視します。
- SQL 例外から SQL 警告を作成して返します。これによって警告メッセージ・チェーンに警告が追加されます。
- 元のメッセージが SQL 警告の場合に、`messageHandler` は SQL 警告を緊急と判断し、SQL 例外を作成して返します。制御が `jConnect` に返されると、この例外が発生します。

エラー・メッセージ・ハンドラのインストール

`SybMessageHandler` を実装するエラー・メッセージ・ハンドラをインストールするには、`SybDriver`、`SybConnection`、または `SybStatement` から `setMessageHandler` メソッドを呼び出します。`SybDriver` からエラー・メッセージ・ハンドラをインストールした場合は、それ以降のすべての `SybConnection` オブジェクトに継承されます。`SybConnection` オブジェクトからエラー・メッセージ・ハンドラをインストールした場合は、その `SybConnection` が作成するすべての `SybStatement` オブジェクトに継承されます。

この階層が適用されるのは、エラー・メッセージ・ハンドラ・オブジェクトがインストールされた時点以降だけです。たとえば、“myConnection,” という名前の `SybConnection` オブジェクトを作成してから

`SybDriver.setMessageHandler` を呼び出してエラー・メッセージ・ハンドラ・オブジェクトをインストールしたとき、“myConnection” でこのオブジェクトを使用することはできません。

現在のエラー・メッセージ・ハンドラ・オブジェクトを返すには、`getMessageHandler` を使用してください。

エラー・メッセージ・ハンドラの例

```
import java.io.*;
import java.sql.*;
import com.sybase.jdbcx.SybMessageHandler;
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybStatement;
import java.util.*;

public class MyApp
{
    static SybConnection conn = null;
    static SybStatement stmt = null;
    static ResultSet rs = null;
    static String user = "guest";
    static String password = "sybase";
    static String server = "jdbc:sybase:Tds:192.138.151.39:4444";
    static final int AVOID_SQLE = 20001;

    public MyApp()
    {
        try
        {
            Class.forName("com.sybase.jdbc4.jdbc.SybDriver").newInstance();
            Properties props = new Properties();
            props.put("user", user);
            props.put("password", password);
            conn = (SybConnection)
                DriverManager.getConnection(server, props);
            conn.setMessageHandler(new NoResultSetHandler());
            stmt = (SybStatement) conn.createStatement();
            stmt.executeUpdate("raiserror 20001 'your error'");

            for (SQLWarning sqw = _stmt.getWarnings();
                sqw != null;
                sqw = sqw.getNextWarning());
            {
                if (sqw.getErrorCode() == AVOID_SQLE);
                {
                    System.out.println("Error" + sqw.getErrorCode()+
                        " was found in the Statement's warning list.");
                    break;
                }
            }
            stmt.close();
            conn.close();
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
}

class NoResultSetHandler implements SybMessageHandler
{
    public SQLException messageHandler(SQLException sqe)
    {
        int code = sqe.getErrorCode();
        if (code == AVOID_SQLE)
        {
            System.out.println("User " + _user + " downgrading " +
                AVOID_SQLE + " to a warning");
            sqe = new SQLWarning(sqe.getMessage(),

                sqe.getSQLState(), sqe.getErrorCode());
        }
        return sqe;
    }
}

public static void main(String args[])
{
    new MyApp();
}
```

マイクロ秒の精度の time データ

jConnect for JDBC は、SQL データ型の **bigdatetime** と **bigtime** をサポートすることで、マイクロ秒レベルの精度の time データを提供します。

bigdatetime と **bigtime** は同様に機能し、SQL データ型の **datetime** および **time** とデータ・マッピングが同じです。

- **bigdatetime** は、Adaptive Server のデータ型 **bigdatetime** に対応し、0000 年 1 月 1 日の 0:00:00.000000 から経過したマイクロ秒数を示します。有効な **bigdatetime** 値の範囲は、0001 年 1 月 1 日の 0:00:00.000000 から 9999 年 12 月 31 日の 23:59:59.999999 までです。
- **bigtime** は、Adaptive Server のデータ型 **bigtime** に対応し、当日の午前 0 時から経過したマイクロ秒数を示します。有効な **bigtime** 値の範囲は、00:00:00.000000 から 23:59:59.999999 までです。

使用法

- Adaptive Server 15.5 への接続時に、jConnect for JDBC は **bigdatetime** および **bigtime** データ型を使用してデータを転送します。受信した Adaptive Server カラムが **datetime** および **time** として定義されている場合でも同様です。

これは、Adaptive Server が、Adaptive Server カラムに合わせるために、jConnect for JDBC から取得した値を暗黙的にトランケートする可能性があることを意味します。たとえば、**bigtime** の値 23:59:59.999999 は、**time** データ型の Adaptive Server カラムに 23:59:59.996 として保存されます。

- Adaptive Server 15.0.x 以前のバージョンへの接続時には、jConnect for JDBC ドライバは **datetime** および **time** データ型を使用してデータを転送します。

パスワードの暗号化の使用

jConnect for JDBC はデフォルトで、ネットワークを介してプレーン・テキストのパスワードを Adaptive Server に送信して認証を求めます。ただし、jConnect は、パスワードの対称／非対称暗号化もサポートしています。この機能を使用すると、パスワードを暗号化してからネットワークに送信できます。対称暗号化メカニズムでは、パスワードの暗号化と復号化に同じキーが使用されます。これに対して、非対称暗号化メカニズムでは、暗号化にはパブリック・キー、復号化には別のプライベート・キーが使用されます。プライベート・キーはネットワークを介して共有されないため、非対称暗号化の方が対称暗号化よりも安全であると考えられます。パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用されます。

注意 パスワードの非対称暗号化機能を使用するには、パスワードの暗号化をサポートするサーバ (Adaptive Server 15.0.2 など) が必要です。

パスワードの暗号化の有効化

ENCRYPT_PASSWORD 接続プロパティでは、パスワードが暗号化フォーマットで転送されるかどうかを指定します。このプロパティは、非対称キー暗号化を有効にする場合にも使用します。パスワードの暗号化が有効になっていて、サーバが非対称キー暗号化をサポートしている場合、非対称キー暗号化が対称キー暗号化の代わりに使用されます。

パスワードの暗号化を有効にするには、ENCRYPT_PASSWORD 接続プロパティを **true** に設定します。デフォルト値は **false** です。

注意 暗号化されたパスワードの使用をクライアントに要求するようにサーバを設定した場合、ユーザがプレーン・テキスト形式のパスワードを入力すると、ログインに失敗します。

クリア・テキスト・パスワードでのログイン・リトライを有効にする

ENCRYPT_PASSWORD プロパティを True に設定しているときに、サーバがパスワードの暗号化をサポートしない場合、サーバのログインは失敗します。パスワードの暗号化をサポートしないサーバでクリア・テキスト・パスワードを使用するには、RETRY_WITH_NO_ENCRYPTION プロパティを True に設定します。

ENCRYPT_PASSWORD プロパティおよび RETRY_WITH_NO_ENCRYPTION プロパティを True に設定すると、jConnect は暗号化されたパスワードを先に使用してログインします。ログインが失敗した場合、jConnect はクリア・テキスト形式のパスワードを使用してログインします。

JCE (Java Cryptography Extension) プロバイダの設定

非対称パスワード暗号化メカニズムでは、RSA 暗号化アルゴリズムを使用して、転送されるパスワードを暗号化します。RSA 暗号化を実行するには、適切な JCE (Java Cryptography Extension) プロバイダを使用して JRE を設定します。設定する JCE プロバイダでは“RSA/NONE/OAEPWithSHA1AndMGF1Padding”変形をサポートしている必要があります。

Sun JRE に付属している Sun JCE プロバイダでは、“RSA/NONE/OAEPWithSHA1AndMGF1Padding”変形を処理できない可能性があります。この場合に拡張パスワード暗号化機能を使用するには、この変形をサポートする外部の JCE プロバイダを設定してください。JCE で必要な変形を処理できない場合、ログイン時にエラー・メッセージが表示されます。

JCE_PROVIDER_CLASS 接続プロパティを使用すると、JCE プロバイダを指定できます。市販またはオープン・ソースの JCE プロバイダが数多く提供されており、その中から選択できます。たとえば、“Bouncy Castle Crypto API for Java”は、一般的なオープン・ソースの Java JCE プロバイダです。

JCE_PROVIDER_CLASS プロパティを指定しない場合、jConnect はバンドルされた JCE を使用しようとします。

JCE プロバイダを指定するには：

- JCE_PROVIDER_CLASS プロパティに、使用するプロバイダの完全修飾クラス名を設定します。たとえば Bouncy Castle JCE パッケージを使用するには、次のように入力します。

```
String url = "jdbc:sybase:Tds:myserver:3697";
Properties props = new Properties();
props.put("ENCRYPT_PASSWORD", "true");
props.put("JCE_PROVIDER_CLASS",
"org.bouncycastle.jce.provider.BouncyCastleProvider");

/* Set up additional connection properties as needed */
props.put("user", "xyz");
props.put("password", "123");

/* get the connection */
Connection con = DriverManager.getConnection(url, props);
```


- JCE プロバイダを (使用する前に) 設定します。次の2つの方法のいずれかでこれを行います。
- JCE プロバイダの *jar* ファイルを次の JRE 標準拡張ディレクトリにコピーします。
 - UNIX / Mac OS X プラットフォームの場合
`${JAVA_HOME}/jre/lib/ext`
 - Windows の場合
`%JAVA_HOME%\jre\lib\ext`
- JCE *jar* ファイルを適切なディレクトリにコピーできない場合は、**Sun JCE Reference Guide**
<http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html> で、外部 JCE プロバイダの設定方法を参照してください。

指定された JCE プロバイダを使用できない場合、jConnect は JRE セキュリティ・プロファイルで設定されている JCE プロバイダを使用しようとします。他の JCE プロバイダが設定されていない場合や、必要な変形およびパスワードの暗号化がサポートされていないプロバイダが設定されている場合、接続は失敗します。

GSE-J を使用した RSA パスワード暗号化の実行

Certicom Security Builder GSE-J を使用して、RSA パスワード暗号化を実行することができます。Certicom Security Builder GSE-J は、jConnect ドライバに付属している FIPS 140-2 準拠の JCE プロバイダです。このプロバイダには、*EccpressoFIPS.jar* および *EccpressoFIPSJca.jar* という2つの JAR ファイルがあり、これらのファイルはそれぞれ `$JDBC_HOME/classes` ディレクトリ、`$JDBC_HOME/devclasses` ディレクトリに格納されています。

Certicom Security Builder GSE-J プロバイダを使用するには、JCE_PROVIDER_CLASS 接続プロパティの値を “com.certicom.ecc.jcae.Certicom” に設定し、*EccpressoFIPS.jar* ファイルと *EccpressoFIPSJca.jar* ファイルを CLASSPATH に追加します。詳細については、「[JCE \(Java Cryptography Extension\) プロバイダの設定](#)」(76 ページ) を参照してください。

注意 パスワードの暗号化を、JCE_PROVIDER_CLASS 接続プロパティではなく ENCRYPT_PASSWORD 接続プロパティを設定することで有効にすると、jConnect は Certicom Security Builder GSE-J プロバイダを検索してロードしようとします。これは、*EccpressoFIPS.jar* と *EccpressoFIPSJca.jar* が、使用中の jConnect JAR ファイル (*jconn4.jar* または *jconn4d.jar*) と同じディレクトリにある場合にのみ成功します。

テーブル内のカラム・データとしての Java オブジェクトの格納

データベース製品には、Java オブジェクトをデータベース内のカラム・データとして直接格納できるものもあります。このようなデータベースでは、Java クラスはデータ型として扱われ、Java クラスをそのデータ型として持つカラムを宣言できます。

jConnect では、`PreparedStatement` インタフェース内で定義された `setObject` メソッドと、`CallableStatement` インタフェースおよび `ResultSet` インタフェース内で定義された `getObject` メソッドを実装することによって、Java オブジェクトをデータベースに格納できます。これによって、jConnect を使用するアプリケーションで、ネイティブの JDBC クラスおよびメソッドを使用して Java オブジェクトをカラム・データとして直接格納したり取り出したりすることができます。

注意 `getObject` メソッドおよび `setObject` メソッドを使用するには、jConnect のバージョンを `com.sybase.jdbcx.SybDriver.VERSION_4` 以降に設定してください。[「JCONNECT_VERSION の使用」\(6 ページ\)](#) を参照してください。

以降の項では、jConnect と JDBC を使用してオブジェクトをテーブルに格納し、取り出すための条件と手順について説明します。

- [Java オブジェクトをカラム・データとして格納するための前提条件](#)
- [データベースへの Java オブジェクトの送信](#)
- [データベースからの Java オブジェクトの受信](#)

注意 Adaptive Server バージョン 12.0 以降および SQL Anywhere バージョン 6.0.x 以降では Java オブジェクトをテーブルに格納できますが、いくつかの制限があります。『リリース・ノート jConnect for JDBC』を参照してください。

Java オブジェクトをカラム・データとして格納するための前提条件

ユーザ定義の Java クラスに属している Java オブジェクトをカラム内に格納するには、次の3つの条件を満たす必要があります。

- クラスは `java.io.Serializable` インタフェースを実装していなければならない。これは `jConnect` がネイティブの Java 直列化／直列化解除を使用してデータベースとの間のオブジェクトの送受信を行うためである。
- クラス定義が格納先データベースにインストールされていなければならない。または `DynamicClassLoader (DCL)` を使用しなければならない。DCL によって SQL Anywhere または Adaptive Server サーバから直接クラスをロードすると、ローカルの `CLASSPATH` に存在しているものと同様に使用できる。詳細については、「[動的クラス・ロードの使用](#)」(82 ページ)を参照。
- クライアント・システムは、ローカルの `CLASSPATH` 環境変数を経由してアクセスできる `.class` ファイルにクラス定義を持っていなければならない。

データベースへの Java オブジェクトの送信

ユーザ定義クラスのインスタンスをカラム・データとして送信するには、次のように `PreparedStatement` インタフェース内に定義されている `setObject` メソッドのいずれかを使用します。

```
void setObject(int parameterIndex, Object x, int targetSqlType,
    int scale) throws SQLException;
void setObject(int parameterIndex, Object x, int targetSqlType)
    throws SQLException;
void setObject(int parameterIndex, Object x) throws SQLException;
```

`jConnect` では、Java オブジェクトを送信するために、`target sql.Type` として `java.sql.Types.JAVA_OBJECT` を使用できます。または、`java.sql.Types.OTHER` を使用します。

次の例では `Address` クラスを定義し、次に、データ型が `Address` クラスである `Address` カラムを持つ `Friends` テーブルの定義を示します。その後で、テーブルにローを挿入します。

```
public class Address implements Serializable
{
    public String    streetNumber;
    public String    street;
    public String    apartmentNumber;
    public String    city;
    public int       zipCode;
    //Methods
    ...
}

/* This code assumes a table with the following structure
** Create table Friends:
```

```
** (firstname varchar(30) ,
**  lastname varchar(30),
**  address Address,
**  phone varchar(15))
*/

// Connect to the database containing the Friends table.
Connection conn =
    DriverManager.getConnection("jdbc:sybase:Tds:localhost:5000",
        "username", "password");

// Create a Prepared Statement object with an insert statement
//for updating the Friends table.
PreparedStatement ps = conn.prepareStatement("INSERT INTO
    Friends values (?, ?, ?, ?)");

// Now, set the values in the prepared statement object, ps.
// set firstname to "Joan."
ps.setString(1, "Joan");

// Set last name to "Smith."
ps.setString(2, "Smith");

// Assuming that we already have "Joan_address" as an instance
// of Address, use setObject(int parameterIndex, Object x) to
// set the address column to "Joan_address."
ps.setObject(3, Joan_address);

// Set the phone column to Joan's phone number.
ps.setString(4, "123-456-7890");

// Perform the insert.
ps.executeUpdate();
```

データベースからの Java オブジェクトの受信

クライアント JDBC アプリケーションは、データベースからの結果セットの一部として、またはストアド・プロシージャから返される出力パラメータの値として、Java オブジェクトを受け取ることができます。

Java オブジェクトがカラム・データとして結果セットに含まれている場合は、**ResultSet** インタフェース内の次のいずれかの **getObject** メソッドを使用して、オブジェクトを取り出します。

```
Object getObject(int columnIndex) throws SQLException;
Object getObject(String columnName) throws SQLException;
```

Java オブジェクトがストアド・プロシージャからの出力パラメータに含まれている場合は、次に示す **CallableStatement** インタフェース内の **getObject** メソッドを使用して、オブジェクトを取り出します。

```
Object getObject(int parameterIndex) throws SQLException;
```

次の例では、

ResultSet.getObject(int parameterIndex) を使用して、結果セットの一部として受け取ったオブジェクトをクラス変数に割り当てます。この例では、前の項で使った **Address** クラスと **Friends** テーブルを使用し、封筒に名前と住所を印刷する簡単なアプリケーションを示します。

```
/*
** This application takes a first and last name, gets the
** specified person's address from the Friends table in the
** database, and addresses an envelope using the name and
** retrieved address.
*/
public class Envelope
{
    Connection conn = null;
    String firstName = null;
    String lastName = null;
    String street = null;
    String city = null;
    String zip = null;

    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Usage: Envelope <firstName>
            <lastName>");
            System.exit(1);
        }
        // create a 4" x 10" envelope
        Envelope e = new Envelope(4, 10);
        try
        {
            // connect to the database with the Friends table.
            conn = DriverManager.getConnection(
                "jdbc:sybase:Tds:localhost:5000", "username",
                "password");
            // look up the address of the specified person
            firstName = args[0];
            lastName = args[1];
            PreparedStatement ps = conn.prepareStatement(
                "SELECT address FROM friends WHERE " +
                "firstname = ? AND lastname = ?");
            ps.setString(1, firstName);
            ps.setString(2, lastName);
            ResultSet rs = ps.executeQuery();
```

```
        if (rs.next())
        {
            Address a = (Address) rs.getObject(1);
            // set the destination address on the envelope
            e.setAddress(firstName, lastName, a);
        }
        conn.close();
    }
    catch (SQLException sqe)
    {
        sqe.printStackTrace();
        System.exit(2);
    }
    // if everything was successful, print the envelope
    e.print();
}

private void setAddress(String fname, String lname, Address a)
{
    street = a.streetNumber + " " + a.street + " " +
        a.apartmentNumber;
    city = a.city;
    zip = "" + a.zipCode;
}

private void print()
{
    // Print the name and address on the envelope.
    ...
}
}
```

より詳細な例については、jConnect インストール・ディレクトリの *sample2* サブディレクトリにある `HandleObject.java` を参照してください。

動的クラス・ロードの使用

SQL Anywhere および Adaptive Server では、次のものを指定するときに Java クラスを使用できます。

- SQL カラムのデータ型
- Transact-SQL 変数のデータ型
- SQL カラムのデフォルト値

以前のバージョンでは jConnect の CLASSPATH に含まれるクラス以外にはアクセスできませんでした。つまり、jConnect アプリケーションがローカルの CLASSPATH がないクラスのインスタンスにアクセスしようとすると、`java.lang.ClassNotFound` 例外が発生しました。

jConnect バージョン 6.05 以降では、DynamicClassLoader (DCL) を実装することによって、SQL Anywhere または Adaptive Server サーバから直接クラスをロードして、ローカルの CLASSPATH に存在しているクラスと同様に使用することができます。

スーパークラスに存在するセキュリティ機能はすべて継承されます。jConnect の動作は、Java 2 に実装されているローダ委任モデルに従っています。まず、要求されたクラスを CLASSPATH からロードしようとします。これに失敗したときは、DynamicClassLoader を試行します。

Java と Adaptive Server の使用方法の詳細については、『Adaptive Server での Java』を参照してください。

DynamicClassLoader の使用方法

DCL 機能を使用するには次の手順に従います。

- 1 クラス・ローダを作成して設定します。jConnect アプリケーションのコードは次のようになります。

```
Properties props = new Properties();

// URL of the server where the classes live.
String classesUrl = "jdbc:sybase:Tds:myase:1200";

// Connection properties for connecting to above server.
props.put("user", "grinch");
props.put("password", "meanone");
...

// Ask the SybDriver for a new class loader.
DynamicClassLoader loader = driver.getClassLoader(classesUrl, props);
```

- 2 クエリを実行する文が新しいクラス・ローダを使用できるように、CLASS_LOADER 接続プロパティを使用して設定します。クラス・ローダの作成後は、次のコード (手順 1 のコード例からの続き) に示すように、以降の接続にこのクラス・ローダを渡します。

```
// Stash the class loader so that other connection(s)
// can know about it.
props.put("CLASS_LOADER", loader);

// Additional connection properties
props.put("user", "joeuser");
props.put("password", "joespassword");

// URL of the server we now want to connect to.
String url = "jdbc:sybase:Tds:jdbc.sybase.com:4446";

// Make a connection and go.
Connection conn = DriverManager.getConnection(url, props);
```

Java クラスの定義は次のとおりであるとします。

```
class Addr {
    String street;
    String city;
    String state;
}
```

SQL テーブルの定義は次のとおりであるとします。

```
create table employee (char(100) name, int empid, Addr address)
```

- 3 クライアント・アプリケーションの CLASSPATH に Addr クラスがない場合は、次のクライアント側コードを使用します。

```
Statement stmt = conn.createStatement();
// Retrieve some rows from the table that has a Java class
// as one of its fields.
ResultSet rs = stmt.executeQuery(
    "select * from employee where empid = '19'");
if (rs.next() {
    // Even though the class is not in our class path,
    // we should be able to access its instance.
    Object obj = rs.getObject("address");
    // The class has been loaded from the server, so let's take a look.
    Class c = obj.getClass();
    // Some Java Reflection can be done here to access the fields of obj.
    ...
}
```

CLASS_LOADER 接続プロパティは、複数の接続間で1つのクラス・ローダを共有できる便利なメカニズムです。

接続間でクラス・ローダを共有してもクラス競合が発生しないように、アプリケーションを作成してください。たとえば、クラス `org.foo.Bar` のインスタンスが2つのデータベースにそれぞれ存在していて、これらのインスタンスがまったく異なるもので互換性もない場合に、同じローダを使用して両方のクラスにアクセスすると、問題が発生する可能性があります。最初の接続からの結果セットが検査されるときに、最初のクラスがロードされます。2番目の接続からの結果セットを検査するときには、クラスはすでにロードされています。したがって、2番目のクラスがロードされることはないの、jConnect がこの状態を直接検出する方法はありません。

ただし、Java に組み込まれているメカニズムによって、クラスのバージョンが、直列化解除後のオブジェクトのバージョン情報と一致することが保証されます。前述のような状態も、少なくとも Java によって検出されて報告されます。

クラスとそのインスタンスは同じデータベースまたはサーバに存在していなくてもかまいませんが、ローダと以降の接続が同じデータベースまたはサーバを参照できるようにするとよいでしょう。

非直列化の使用

次の例では、ローカル・ファイルからオブジェクトの直列化を解除する方法を説明します。直列化されたオブジェクトはサーバ上に存在するクラスのインスタンスで、CLASSPATH には存在しません。

`SybResultSet.getObject()` は `DynamicObjectInputStream` を使用します。これは `ObjectInputStream` のサブクラスで、デフォルト・システム(「ブート」)のクラス・ローダではなく `DynamicClassLoader` からクラス定義をロードします。

```
// Make a stream on the file containing the
//serialized object.
FileInputStream fileStream = new FileInputStream("serFile");
// Make a "deserializer" on it. Notice that, apart
//from the additional parameter, this is the same
//as ObjectInputStreamDynamicObjectInputStream
stream = new DynamicObjectInputStream(fileStream, loader);
// As the object is deserialized, its class is
//retrieved through the loader from our server.
Object obj = stream.readObject();stream.close();
```

.jar ファイルの事前ロード

jConnect バージョン 6.05 以降には、`PRELOAD_JARS` という接続プロパティがあります。`.jar` ファイル名をカンマで区切ったリストとして定義されているときは、指定された `.jar` ファイルがすべてロードされます。このコンテキストでは、“JAR”はサーバで使用される「保持された JAR 名」を意味します。これは、install Java プログラムで指定される `.jar` ファイル名です。次に例を示します。

```
install java new jar 'myJarName' from file '/tmp/mystuff.jar'
```

`PRELOAD_JARS` を設定すると `.jar` ファイルがクラス・ローダに関連付けられるため、接続するたびに事前にロードする必要はなくなります。1つの接続に対して `PRELOAD_JARS` を指定するだけで済みます。その後で、同じ `.jar` ファイルを事前ロードしようとする、`.jar` データが不必要にサーバから取り出されたとしてパフォーマンスの問題が発生することがあります。

注意 SQL Anywhere は、`.jar` ファイルを 1つのエンティティとして返すことはできません。このため、jConnect は各クラスを順に取り出す処理を繰り返します。ただし、Adaptive Server では、`.jar` ファイル全体が取り出され、そのファイルに含まれる個々のクラスがロードされます。

高度な機能

`DynamicClassLoader` にはさまざまなパブリック・メソッドがあります。詳細については、[JDBC_HOME/docs/en/javadocs](http://jdbc_home/docs/en/javadocs) にある javadocs 情報を参照してください。

この他に、一連のクラスがロードされることがあらかじめわかっている場合にローダのデータベース接続を維持する機能や、単一クラスを名前によって明示的にロードする機能が追加されました。`java.lang.ClassLoader` から継承されたパブリック・メソッドも使用できます。クラスのロードを処理する `java.lang.Class` 内のメソッドも使用できます。ただし、これらのメソッドには使用するクラス・ローダを仮定するものもあるため、注意して使用してください。特に、`Class.forName` は 3 つの引数の形式のものを使用してください。このようにしなければ、システムの (「ブート」) クラス・ローダが使用されます。「[エラー・メッセージの処理](#)」(69 ページ) を参照してください。

JDBC 4.0 仕様のサポート

JDBC 4.0 の次の仕様がサポートされます。

- 接続管理
- SQL ドライバの自動ロード
- データベース・メタデータ
- 各国文字セット変換
- ラッパ・パターン
- スカラ関数 `CHAR_LENGTH`、`CHARACTER_LENGTH`、`CURRENT_DATE`、`CURRENT_TIME`、`CURRENT_TIMESTAMP`、`EXTRACT`、および `OCTET_LENGTH`、`POSITION`

JDBC 4.0 の仕様の詳細については、[Sun Developer Network](http://developers.sun.com/) (<http://developers.sun.com/>) を参照してください。

JDBC 3.0 仕様のサポート

この項では、`jConnect 7.0` の現在のリリースでサポートされている JDBC 3.0 の機能について説明します。

セーブポイントのサポート

指定したセーブポイントにトランザクションを設定、解放、またはロールバックするためのメソッドが含まれた `Savepoint` インタフェースが追加されています。

トランザクションでのセーブポイントの使用	JDBC 2.0 でのトランザクションのサポートによって、トランザクションに対する制御を保ち、トランザクション内のすべての変更をロールバックできました。JDBC 3.0 では、セーブポイントに対する制御を強化できます。Savepoint インタフェースを使用して、トランザクションを論理ブレークポイントに分割し、ロールバックされるトランザクションの範囲を制御できます。
セーブポイントの設定とセーブポイントへのロールバック	JDBC 3.0 API では、現在のトランザクション内にセーブポイントを設定し、Savepoint オブジェクトを返すメソッド <code>Connection.setSavepoint</code> が追加されています。 <code>Connection.rollback</code> メソッドは、Savepoint オブジェクト引数を使用できるようにオーバーロードされます。
セーブポイントの解放	<code>Connection.releaseSavepoint</code> メソッドは、Savepoint オブジェクトをパラメータとして使用し、現在のトランザクションからそのセーブポイントを削除します。Savepoint が解放された後、ロールバック操作でそのセーブポイントを参照しようとする、 <code>SQLException</code> が発生します。 トランザクション内に作成したセーブポイントは、トランザクションがコミットされる時、またはトランザクション全体がロールバックされる時に自動的に解放され、無効になります。トランザクションをセーブポイントにロールバックすると、該当するセーブポイントの後に作成されたその他のセーブポイントはすべて自動的に解放され、無効になります。

注意 JDBC API の実装でセーブポイントがサポートされるかどうかを調べるには、`DatabaseMetaData.supportsSavepoints` メソッドを使用します。

パラメータ・データの取得

パラメータの数、型、およびプロパティを準備文に記述し、`DatabaseMetaData` の新しいメソッドや変更されたメソッドをサポートするインタフェース `ParameterMetaData` が追加されています。

自動生成されたキーの取得

自動生成された値を含んでいるカラムから値を取得する方法が追加されています。JDBC 3.0 では、自動生成キーまたは自動インクリメント・キーの値を取得する共通の必要性に対応しています。

自動生成キーの値の確認

自動生成キーを取得することをドライバに通知するには、定数 `Statement.RETURN_GENERATED_KEYS` を `Statement.execute()` メソッドの 2 番目のパラメータとして渡します。この文を実行した後は、`Statement.getGeneratedKeys()` を呼び出して生成されたキーを取得します。結果セットには、取得した自動生成キーごとのローが含まれます。

注意 Adaptive Server は、生成されたキーの結果セットを返すことができません。`insert` コマンドのバッチを実行するときに `Statement.getGeneratedKeys()` を呼び出すと、最後に生成されたキーの値のみが返されます。

サンプル・コードなどの自動生成キーの取得に関する詳細は、Sun Microsystems の Web サイトで、“retrieving automatically generated keys”を検索してください。

オープンした `ResultSet` オブジェクトを複数保持する機能

`getMoreResults(int)` が追加されています。このメソッドでは、`Statement` オブジェクトによって返される `ResultSet` オブジェクトが、その後の `ResultSet` オブジェクトが返される前にクローズされるかどうかを指定する引数を使用します。

JDBC 3.0 仕様では、変更の一環として、`Statement` インタフェースでオープンした `ResultSets` を複数サポートできます。JDBC 2 仕様では、複数の結果を返す文で `ResultSet` を特定の時間に 1 つしかオープンしておけませんが、JDBC 3.0 仕様ではその制限が取り払われます。オープンした結果を複数サポートするために、`Statement` インタフェースはメソッド `getMoreResults()` をオーバーロードしたものを追加します。`getMoreResults(int)` メソッドは、`getResultSet()` メソッドが呼び出されたときに、その前にオープンした `ResultSets` の動作を指定する整数フラグを使用します。このインタフェースでは次のようにフラグを定義します。

- `CLOSE_ALL_RESULTS` – `getMoreResults()` オブジェクトを呼び出すと、その前にオープンしたすべての `ResultSet` オブジェクトがクローズされます。
- `CLOSE_CURRENT_RESULT` – `getMoreResults()` を呼び出すと、現在の `ResultSet` オブジェクトがクローズされます。
- `KEEP_CURRENT_RESULT` – `getMoreResults()` を呼び出したとき、現在の `ResultSet` オブジェクトはクローズされません。

名前による *CallableStatement* オブジェクトへのパラメータの受け渡し

CallableStatement オブジェクトに設定するパラメータを文字列で識別できるようにするメソッドが追加されています。

CallableStatement インタフェースを使用して、以前のようにパラメータのインデックスを指定する方法ではなく、名前によってパラメータを指定できます。デフォルト値が設定されたパラメータがプロシージャに多数含まれている場合、この方法が役立ちます。デフォルト値が設定されていない値のみを指定するには、名前付きパラメータを使用します。

保持可能なカーソルのサポート

ResultSet オブジェクトの保持可能性を指定できる機能が追加されています。保持可能なカーソルまたは結果とは、カーソルが含まれているトランザクションがコミットされたときに、自動的にクローズされないカーソルまたは結果のことです。JDBC 3.0 では、カーソルの保持可能性を指定する機能のサポートが追加されています。*ResultSet* の保持可能性を指定するには、*createStatement()*、*prepareStatement()*、または *prepareCall()* メソッドを使用して、文を準備するときにそのように指定する必要があります。保持可能性には、次のいずれかの定数を指定できます。

- **HOLD_CURSORS_OVER_COMMIT** – *ResultSet* オブジェクト (カーソル) はクローズされません。**commit** 操作が暗黙的または明示的に実行されたとき、*ResultSet* オブジェクトはオープンしたまま保持されます。
- **CLOSE_CURSORS_AT_COMMIT** – **commit** 操作が暗黙的または明示的に実行されたとき、*ResultSet* オブジェクト (カーソル) はクローズされます。

トランザクションがコミットされたときにカーソルをクローズすると、通常パフォーマンスが向上します。トランザクションのコミット後にカーソルが必要でないかぎり、**commit** 操作の実行時にカーソルをクローズすることをおすすめします。仕様では、*ResultSet* のデフォルトの保持可能性が定義されていないので、その動作は実装によって異なります。

JDBC 2.0 オプション・パッケージ拡張サポート

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)には、JDBC 2.0 ドライバが実装できるさまざまな機能が定義されています。jConnect バージョン 6.05 以降では、次のオプション・パッケージ拡張機能が実装されています。

- [JNDI によるデータベースの命名](#)
(jConnect がサポートするすべての Sybase DBMS で動作します)
- [接続プール](#)
(jConnect がサポートするすべての Sybase DBMS で動作します)
- [分散トランザクション管理のサポート](#)は、Adaptive Server でのみ動作します。

注意 Java 1.1.6 以降と互換性のある JNDI 1.2 を使用することをおすすめします。

JNDI によるデータベースの命名

参照

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Chapter 5 JNDI and the JDBC API」

関連インタフェース

- `javax.sql.DataSource`
- `javax.naming.Referenceable`
- `javax.naming.spi.ObjectFactory`

この機能は、JDBC クライアントがデータベース接続を取得するときに、標準アプローチの代わりに使用できます。クライアントは、`Class.forName` (“com.sybase.jdbc4.jdbc.SybDriver”) を呼び出してから、JDBC URL を `DriverManager` の `getConnection()` メソッドに渡す代わりに、論理名を使用して JNDI ネーム・サーバにアクセスすることによって `javax.sql.DataSource` オブジェクトを取得できます。このオブジェクトはドライバをロードして、物理データベースへの接続を確立する役割を持ちます。ベンダ固有の情報は `DataSource` オブジェクト内に置かれているので、クライアントのコードはより単純で再使用可能になります。

Sybase での `DataSource` オブジェクトの実装は `com.sybase.jdbcx.SybDataSource` です (詳細については javadocs を参照してください)。この実装では JavaBean コンポーネントの設計パターンを使って、次の標準プロパティがサポートされています。

- `databaseName`
- `dataSourceName`
- `description`
- `networkProtocol`
- `password`
- `portNumber`
- `serverName`
- `user`

注意 `roleName` はサポートされていません。

jConnect では `javax.naming.spi.ObjectFactory` インタフェースが実装されているので、ネーム・サーバのエントリの属性から `DataSource` オブジェクトを構築できます。`javax.naming.Reference`、または `javax.naming.Name` と `javax.naming.DirContext` が指定されていれば、このファクトリで `com.sybase.jdbcx.SybDataSource` オブジェクトを構築できます。このファクトリを使用するには、`com.sybase.jdbc4.SybObjectFactory` が含まれるように `java.naming.object.factory` システム・プロパティを設定します。

使用法

`DataSource` はさまざまなアプリケーションでさまざまな方法で使用できます。以降の項では、すべてのオプションをコード例とともに紹介し、手順を説明します。詳細については、『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)、および Sun の Web サイトにある JNDI のマニュアルを参照してください。

1a. 管理者による設定 : LDAP

LDAP 接続は jConnect バージョン 4.0 以降でサポートされています。したがって、LDAP Data Interchange Format (LDIF) を使用して `DataSource` を LDAP エントリとして設定する方法をおすすめします。この方法では、カスタム・ソフトウェアは必要ありません。例：

```
dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
```

1b. クライアントによる
アクセス

これは一般的な JDBC クライアント・アプリケーションです。唯一の相違点は、**DriverManager** にアクセスして JDBC の URL を指定する代わりに、ネーム・サーバにアクセスして **DataSource** オブジェクトへの参照を取得する点です。接続を取得した後のクライアントのコードは、他の JDBC クライアントのコードとまったく同じです。このコードはごく一般的なもので、Sybase を参照するのは、環境の一部として設定されるオブジェクト・ファクトリ・プロパティの設定時だけです。

jConnect インストール環境には、**DataSource** の使用方法を説明するサンプル・プログラム *sample2/SimpleDataSource.java* があります。このサンプルは参照用です。つまり、このサンプルを実行するには、インストール環境の設定と、サンプルの編集が必要です。*SimpleDataSource.java* のコードのうち、重要なのは次に示す部分です。

```
import javax.naming.*;
import javax.sql.*;
import java.sql.*;

// set necessary JNDI properties for your environment (same as above)
Properties jndiProps = new Properties();

// used by JNDI to build the SybDataSource
jndiProps.put(Context.OBJECT_FACTORIES,
    "com.sybase.jdbc4.jdbc.SybObjectFactory");

// nameserver that JNDI should talk to
jndiProps.put(Context.PROVIDER_URL, "ldap: some_ldap_server:238/" +
    "o=MyCompany,c=Us");

// used by JNDI to establish the naming context
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

// obtain a connection to your name server
Context ctx = new InitialContext(jndiProps);
DataSource ds = (DataSource) ctx.lookup("servername=myASE");

// obtains a connection to the server as configured earlier.
// in this case, the default username and password will be used
Connection conn = ds.getConnection();

// do standard JDBC methods
...
```

プロパティが仮想マシン内ですでに定義されている場合、つまり Java がブラウザ・プロパティの一部として、または次を使用して設定されたときに渡されている場合は、**Properties** を **InitialContext** コンストラクタに明示的に渡す必要はありません。

```
java -Djava.naming.object.factory=com.sybase.jdbc4.jdbc.SybObjectFactory
```


環境プロパティの設定の詳細については Java VM マニュアルを参照してください。

2a. プログラムでの設定

このフェーズは、通常、社内におけるシステム管理またはアプリケーション統合の担当者が実行します。目的は、データ・ソースを定義した後に、論理名でネーム・サーバに配備することです。サーバを再設定しなければならない場合は（別のマシンやポートに移動した場合など）、この設定ユーティリティ（以下に概要を示します）を実行して、論理名を新しいデータ・ソース設定に割り当て直します。クライアントが認識するのは論理名だけであるので、クライアント・コードは変更しません。

```
import javax.sql.*;
import com.sybase.jdbcx.*;
.....

// create a SybDataSource, and configure it
SybDataSource ds = new com.sybase.jdbc4.jdbc.SybDataSource();
ds.setUser("my_username");
ds.setPassword("my_password");
ds.setDatabaseName("my_favorite_db");
ds.setServerName("db_machine");
ds.setPortNumber(4000);
ds.setDescription("This DataSource represents the Adaptive Server
    Enterprise server running on db_machine at port 2638. The default
    username and password have been set to 'me' and 'mine' respectively.
    Upon connection, the user will access the my_favorite_db database on
    this server.");
Properties props = new Properties()
props.put("REPEAT_READ", "false");
props.put("REQUEST_HA_SESSION", "true");
ds.setConnectionProperties(props);
// store the DataSource object. Typically this is
// done by setting JNDI properties specific to the
// type of JNDI service provider you are using.
// Then, initialize the context and bind the object.
Context ctx = new InitialContext();
ctx.bind("java:comp/env/jdbc/myASE", ds);
```

DataSource を設定した後で、情報を格納する場所および方法を決定します。格納しやすくするため、**SybDataSource** は **java.io.Serializable** であり **javax.naming.Referenceable** です。しかし、JNDI に使用しているサービス・プロバイダに応じてデータをどのように格納するかは、管理者が決定します。

2b. クライアントによるアクセス

クライアントが **DataSource** オブジェクトを取り出すには、**DataSource** が配備されたときと同じ方法で JNDI プロパティを設定します。クライアントは、Java オブジェクトに格納された形式（直列化など）に従ってオブジェクトを変換できるよう、オブジェクト・ファクトリを有効化する必要があります。

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/myASE");
Connection conn = ds.getConnection();
```

接続プール

参照

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Chapter 6 Connection Pooling」

関連インタフェース

- `javax.sql.ConnectionPoolDataSource`
- `javax.sql.PooledConnection`

概要

従来のデータベース・アプリケーションでは、アプリケーションのセッションごとに、使用するデータベースへの接続を作成していました。しかし、Webベースのデータベース・アプリケーションでは、そのアプリケーションを使用している間に新しい接続を何回もオープンしたりクローズしたりする必要があります。

Webベースのデータベース接続を効率的に処理する方法として接続プールを使う方法があります。接続プールは異なるユーザ要求間で接続を共有して、データベース接続を維持し接続を管理することにより、パフォーマンスを管理しアイドル接続の数を減らします。接続要求のたびに、接続プールはまずプールにアイドル接続があるかどうかを判断します。ある場合は、接続プールはデータベースに新しい接続を作成する代わりに、そのアイドル接続を返します。

`com.sybase.jdbc4.jdbc.ConnectionPoolDataSource` クラスは、接続プールの実装と対話するために提供されています。`ConnectionPoolDataSource` を使用するときは、プール実装が `PooledConnection` を監視します。接続をクローズするか、エラーが発生して接続が切断されると、プール実装は通知を受け取ります。この時点で、プール実装は `PooledConnection` をどのように処理するかを決定します。

接続プールを使用しない場合は、トランザクションの処理は次のようになります。

- 1 データベースへの接続を作成します。
- 2 データベースにクエリを送信します。
- 3 結果セットを受け取ります。
- 4 結果セットを表示します。
- 5 接続を切断します。

接続プールを使用する場合の処理は次のようになります。

- 1 接続の「プール」の中に、使用されていない接続があるかどうかを調べます。
- 2 ある場合は、新しい接続を作成する代わりにその接続を使用します。
- 3 データベースにクエリを送信します。
- 4 結果セットを受け取ります。
- 5 結果セットを表示します。
- 6 接続を「プール」に返します。この場合もユーザは“close()”を呼び出しますが、接続はオープンのまま、プールに close 要求が通知されます。

クライアントがデータベースへの接続を確立する必要があるたびに新しい接続を作成するよりも、接続を再使用する方がコストを節約できます。

サード・パーティによる接続プールの実装を可能にするために、jConnect 実装には **ConnectionPoolDataSource** インタフェースがあり、これによって **PooledConnection** が生成されます。これは、**DataSource** インタフェースが **Connection** を生成するのに似ています。

プール実装は、**ConnectionPoolDataSource** の **getPooledConnection()** メソッドを使用して、「実際の」データベース接続を作成します。その後、プール実装自身を **PooledConnection** のリスナとして登録します。

現時点では、クライアントが接続を要求すると、プール実装は使用可能な **PooledConnection** の 1 つに対して **getConnection()** を呼び出します。クライアントが接続を終了して **close** を呼び出すと、接続が解放されて再使用が可能であることが **ConnectionEventListener** インタフェースを介してプール実装に通知されます。

プール実装は、クライアントがなんらかの理由でデータベース接続を正常に続行できなくなったときも **ConnectionEventListener** インタフェースを介して通知を受け取るので、プールからその接続を削除できます。

詳細については、『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Appendix B」を参照してください。

管理者による設定： LDAP

このアプローチは LDIF エントリに行を追加入力することを除いて、「[JNDI によるデータベースの命名](#)」で説明した [1a. 管理者による設定：LDAP](#) と同じです。次の例では、コードの追加された行が太字で表示されています。

```
dn:servername=myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:ConnectionPoolDataSource
```

中間層によるアクセス

このプロシージャでは3つのプロパティ (INITIAL_CONTEXT_FACTORY、PROVIDER_URL、および OBJECT_FACTORIES) が初期化され、ConnectionPoolDataSource オブジェクトが取り出されます。コードの詳細な例については *sample2/SimpleConnectionPool.java* を参照してください。相違点は次のとおりです。

```
...
ConnectionPoolDatabase cpds = (ConnectionPoolDataSource)
    ctx.lookup("servername=myASE");
PooledConnection pconn = cpds.getPooledConnection();
```

分散トランザクション管理のサポート

この機能により、いずれかの Adaptive Server で、標準 Java API による分散トランザクションを実行できます。

注意 この機能は大規模多層環境で使用するよう設計されています。

参照

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Chapter 7 Distributed Transactions」

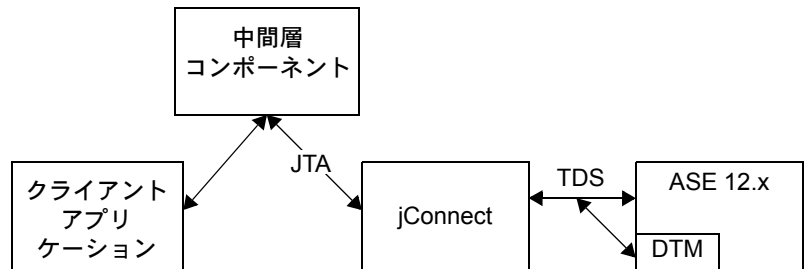
関連インタフェース

- javax.sql.XADataSource
- javax.sql.XAConnection
- javax.transaction.xa.XAResource

バックグラウンドとシステムの稼働要件

- jConnect は Sybase Adaptive Server バージョン 12.0 以降の内部のリソース・マネージャと直接通信するため、インストール環境には分散トランザクション管理のサポートが必要です。
- 分散トランザクションに参加するユーザに “dtm_tm_role” が付与されていなければ、トランザクションは失敗します。
- 分散トランザクションを使用するには、*/sp* ディレクトリにストアド・プロシージャをインストールする必要があります。『jConnect for JDBC インストール・ガイド』の第1章の「ストアド・プロシージャのインストール」を参照してください。

図 2-2: バージョン 12.x での分散トランザクション管理のサポート



管理者による設定：
LDAP

このアプローチは LDIF エントリに行を追加入力することを除いて、「[JNDI によるデータベースの命名](#)」(90 ページ) で説明した [1a. 管理者による設定:LDAP](#) と同じです。次の例では、コードの追加された行が太字で表示されています。

```

dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:XADatasource

```

中間層によるアクセス

このプロシージャでは 3 つのプロパティ (INITIAL_CONTEXT_FACTORY、PROVIDER_URL、および OBJECT_FACTORIES) が初期化され、XADatasource オブジェクトが取り出されます。例：

```

...
XADatasource xads = (XADatasource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection();

```

または、デフォルト設定に代わるユーザ名とパスワードを指定します。

```

...
XADatasource xads = (XADatasource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection("my_username", "my_password");

```

JDBC 標準の制約と解釈

この項では、jConnect での JDBC の実装が、JDBC 標準から逸脱している部分について説明します。次の項目について説明します。

- サポートされない JDBC 4.0 仕様要件
- `Connection.isClosed` と `IS_CLOSED_TEST` の使用
- 未処理の結果がある場合の `Statement.close` の使用
- マルチスレッドに対する調整
- `ResultSet.setCursorName` の使用
- 大きなパラメータ値での `setLong` の使用
- ストアド・プロシージャの実行

サポートされない JDBC 4.0 仕様要件

このリリースでは次をサポートしていません。

- `java.sql.RowID`
- BLOB、CLOB、または NCLOB メソッド
- JDBC 4.0 で導入された XML API

`Connection.isClosed` と `IS_CLOSED_TEST` の使用

JDBC 2.1 仕様のセクション 11.1 では次のように規定されています。

「`Connection.isClosed` メソッドが保証しているのは、`Connection.close` が呼び出された後に `true` を返すということだけです。基本的に、`Connection.isClosed` を呼び出して、データベース接続が有効か無効かを判定することはできません。通常のクライアントは、オペレーションを試みたときに発生する例外をキャッチすることによって、接続が無効であることを判断できます。」

jConnect での `isClosed` メソッドのデフォルトの解釈は、この仕様で定義されている動作とは異なります。`Connection.isClosed` が呼び出されると、jConnect は、この接続に対して `Connection.close` がすでに呼び出されたかどうかを調べます。`close` が既に呼び出されている場合は、`isClosed` に `true` を返します。

`Connection.close` がまだ呼び出されていない場合は、次に、そのデータベースに対して `sp_mda` ストアド・プロシージャを実行します。`sp_mda` ストアド・プロシージャは、標準メタデータの一部であり、jConnect をデータベースと組み合わせて使用する場合は、インストールしておく必要があります。

`sp_mda` を呼び出す目的は、データベース・サーバ上に存在することが明らかである (または、少なくともそう考えられる) プロシージャを `jConnect` が実行できるようにすることです。このストアド・プロシージャが正常に実行された場合は、データベース接続が有効で動作中であることを確認できたことになるので、`jConnect` は `isClosed` に `false` を返します。ただし、`sp_mda` を呼び出したときに `SQLException` が発生した場合は、`jConnect` はその例外をキャッチして `isClosed` に `true` を返します。これは、接続に何らかの問題があると考えられるからです。

`isClosed()` に関する `jConnect` の動作を、標準 JDBC の動作に厳密に準拠させるには、`IS_CLOSED_TEST` 接続プロパティを “INTERNAL” という特殊な値に設定してください。INTERNAL に設定した場合は、`isClosed` の呼び出しで `true` が返されるのは、`Connection.close` がすでに呼び出されているとき、または接続を使用不可能にした `IOException` を `jConnect` が検出したときだけとなります。

`isClosed` の呼び出し時に使用するクエリとして、`sp_mda` とは別のものを指定することもできます。たとえば、`isClosed` を呼び出したときに `select 1` が実行されるようにするには、`IS_CLOSED_TEST` 接続プロパティを `select 1` に設定します。

未処理の結果がある場合の `Statement.close` の使用

JDBC 仕様では、`Statement.execute` を呼び出した後で、その `Statement` から返された結果 (更新カウントや `ResultSet`) をすべて処理せずに同じ `Statement` オブジェクトに対して `close` を呼び出した場合のドライバの動作の規定が若干あいまいです。

たとえば、ロー挿入を7回実行するストアド・プロシージャがデータベース上にあるとします。あるアプリケーションが、`Statement.execute` を使用してこのストアド・プロシージャを実行します。この場合、Sybase データベースはアプリケーションに7個の更新カウント (挿入されたロー1つにつき1個) を返します。通常の JDBC アプリケーションの論理では、ループ内で `getMoreResults` メソッド、`getResultSet` メソッド、`getUpdateCount` メソッドを使用してこれらのカウントを処理します。これは、[Sun Web page for Java developers \(http://java.sun.com/\)](http://java.sun.com/) にある `java.sql.*` パッケージの javadocs で明確に説明されています。

しかし、アプリケーション・プログラマが、返された更新カウントをすべて読み取る前に、誤って `Statement.close` を呼び出す可能性もあります。この場合、`jConnect` はデータベースに `cancel` を送信しますが、予期できない二次的な悪影響が発生することがあります。

この例では、データベースが挿入を完了する前にアプリケーションが `Statement.close` を呼び出すと、データベースが挿入を一部しか実行できなくなる可能性があります。たとえば、ストアド・プロシージャが完了する前にデータベースに対する `cancel` が処理されたことによって、ローが5つ挿入された時点で停止するかもしれません。

この場合、実行されなかった挿入について、ユーザには何も報告されません。jConnect の今後のリリースでは、未処理の結果があるときに `Statement` をクローズしようとするとき `SQLException` が発生するようになりますが、それまでは次に示すガイドラインに従うことを強くおすすめします。

- `Statement.close` を呼び出したとき、結果 (更新カウントや `ResultSet`) の処理がアプリケーション側でまだ完了していない場合は、サーバに `cancel` が送信されます。`select` 文のみを実行する場合は、この動作で問題はありません。しかし、`insert` / `update` / `delete` オペレーションを実行する場合は、オペレーションの一部が予期したとおりには完了しない可能性があります。
- したがって、純粋な `select` 文以外の文を実行したとき、未処理の結果がある状態では、絶対に `close` を呼び出さないでください。
- 代わりに、`Statement.execute` を呼び出す場合は、`getUpdateCount`、`getMoreResults`、`getResultSet` の各メソッドを使用してすべての結果を処理するようにしてください。

マルチスレッドに対する調整

同一の `Statement`、`CallableStatement`、または `PreparedStatement` に対して複数のスレッドが同時にメソッドを呼び出すことは、おすすめしませんが、そのようにする場合は、`Statement` に対するメソッドの呼び出しを手動で同期させる必要があります。jConnect はこの処理を自動的にはい行いません。

たとえば、同一の `Statement` インスタンスを 2 つのスレッドが操作していて、1 つのスレッドがクエリの送信を、もう 1 つが警告の処理を実行している場合に、`Statement` に対するこれらのメソッドの呼び出しをアプリケーション側で同期させなければ、競合が発生する可能性があります。

`ResultSet.setCursorName` の使用

JDBC ドライバの中には、常に文字列が返されるように、SQL クエリに対してカーソル名を生成するものがあります。ただし、次のいずれかに該当しなければ、`ResultSet.setCursorName` が呼び出されても jConnect は名前を返しません。

- 対応する `Statement` に対して `setFetchSize` または `setCursorName` を呼び出した。
- `SELECT_OPEN_CURSOR` 接続プロパティを “true” に設定して、次に示すような `SELECT...FOR UPDATE` 形式のクエリを実行した。例：

```
select au_id from authors for update
```

対応する `Statement` に対して `setFetchSize` または `setCursorName` を呼び出していない場合や、`SELECT_OPEN_CURSOR` 接続プロパティを “true” に設定していない場合は、`null` が返されます。

JDBC 2.0 API ドキュメント (「Chapter 11 Clarifications」) の規定では、他のすべての SQL 文はカーソルをオープンして名前を返す必要はありません。

jConnect でカーソルを使用する方法の詳細については、「[結果セットでのカーソルの使用方法](#)」(45 ページ) を参照してください。

大きなパラメータ値での `setLong` の使用

`PreparedStatement.setLong` メソッドの実装によって、パラメータ値が SQL BIGINT データ型に設定されます。ほとんどの Adaptive Server データベースには、8 バイトの BIGINT データ型はありません。パラメータ値に 4 バイトを超える BIGINT が必要な場合に、`setLong` を使用すると、オーバフロー例外が発生する可能性があります。

サポートされるデータ型

jConnect は、次の Adaptive Server データ型をサポートします。

- `bigint` — 既存の `int` 型の範囲では不十分な場合に使用するよう設計されている真数値データ型です。
- `unsigned int` — 真数値 integer データ型 `unsignedsmallint`、`unsignedint`、および `unsignedbigint` の符号なしバージョンです。
- `unitext` — Unicode 文字用の可変長データ型です。

`bigint` データ型

Sybase は、ネイティブな Adaptive Server データ型としてサポートされる 64 ビット integer データ型である `bigint` をサポートします。Java では、このデータ型は java データ型 `long` にマップします。このデータ型をパラメータとして使用するには、`PreparedStatement.setLong(int index, long value)` を呼び出すと、jConnect はデータを `bigint` として Adaptive Server に送信します。`bigint` カラムから取得する場合は、`ResultSet.getLong(int index)` メソッドを使用できます。

`Unitext` データ型

`unitext` データ型の使用に関して jConnect の API に変更はありません。jConnect は、`unitext` カラムが使用されている場合の Adaptive Server へのデータの格納および Adaptive Server からのデータの取得を、内部的に処理できます。

unsigned int データ型

Adaptive Server では、*unsigned bigint*、*unsigned int*、*unsigned smallint* をネイティブ Adaptive Server データ型としてサポートします。Java には対応する符号なしデータ型がないので、データを正しく処理したい場合は、1 つ高いレベルの integer に対する **set** および **get** を使用する必要があります。たとえば、*unsigned int* からデータを取得する場合、Java データ型 *int* では正の大きい値を格納するのに小さすぎるため、結果として、**ResultSet.getInt(int index)** を呼び出すと正しくないデータが返されたり、例外が発生したりする可能性があります。データを正しく処理するには、1 つ高いレベルの integer 値である **ResultSet.getLong()** を **get** する必要があります。次のテーブルに示す方法で、データの **set** または **get** を行うことができます。

Adaptive Server データ型	Java のデータ型
unsigned smallint	setInt(), getInt()
unsigned int	setLong(), getLong()
unsigned bigint	setBigDecimal(), getBigDecimal()

ストアド・プロシージャの実行

CallableStatement オブジェクト内でストアド・プロシージャを実行するときに、パラメータ値を疑問符で表すようにすると、パラメータに疑問符とリテラルの両方を使用した場合よりも高いパフォーマンスが得られます。また、リテラルと疑問符の両方を使用した場合は、ストアド・プロシージャで出力パラメータを使用することはできません。

次の例はストアド・プロシージャ **MyProc** を実行するための **CallableStatement** オブジェクトとして *sp_stmt* を作成します。

```
CallableStatement sp_stmt = conn.prepareCall(
    "{call MyProc(?,?)}");
```

MyProc 内の 2 つのパラメータは疑問符として表現されています。**CallableStatement** インタフェースの **registerOutParameter** メソッドを使用すると、これらのいずれかまたは両方を出力パラメータとして登録できます。

次の例で、*sp_stmt2* は、ストアド・プロシージャ **MyProc2** を実行するための **CallableStatement** オブジェクトです。

```
CallableStatement sp_stmt2 = conn.prepareCall(
    "{call MyProc2(?, 'javelin')}");
```

sp_stmt2 では、1 つのパラメータがリテラルとして指定され、もう 1 つが疑問符として指定されています。どちらのパラメータも、出力パラメータとして登録することはできません。

パラメータのネーム・バインドを使用して RPC コマンドでストアド・プロシージャを実行するには、次のいずれかのプロシージャを使用します。

- 言語コマンドを使用して、**PreparedStatement** クラスを使用して Java 変数から直接入力パラメータを渡します。次のコード例にこれを示します。

```
// Prepare the statement
System.out.println("Preparing the statement...");
String stmtString = "exec " + procname + " @p3=?, @p1=?";
PreparedStatement pstmt = con.prepareStatement(stmtString);

// Set the values
pstmt.setString(1, "xyz");
pstmt.setInt(2, 123);

// Send the query
System.out.println("Executing the query...");
ResultSet rs = pstmt.executeQuery();
```

- **jConnect バージョン 6.05 以降では、次の例に示す com.sybase.jdbcx.SybCallableStatement インタフェースを使用します。**

```
import com.sybase.jdbcx.*;
....
// prepare the call for the stored procedure to execute as an RPC
String execRPC = "{call " + procName + " (?, ?)}";
SybCallableStatement scs = (SybCallableStatement)
con.prepareCall(execRPC);

// set the values and name the parameters
// also (optional) register for any output parameters
scs.setString(1, "xyz");
scs.setParameterName(1, "@p3");
scs.setInt(2, 123);
scs.setParameterName(2, "@p1");

// execute the RPC
// may also process the results using getResultSet()
// and getMoreResults()

// see the samples for more information on processing results
ResultSet rs = scs.executeQuery();
```


この章では、jConnect のセキュリティに関連する事項について説明します。

トピック名	ページ
概要	105
カスタム・ソケット・プラグインの実装	106
Kerberos	110

概要

jConnect には、クライアント・サーバ間の通信を保護する方法として次のオプションがあります。

- **SSL** — SSL は、ログイン時などに、クライアント・アプリケーションとサーバ・アプリケーションの間の通信を暗号化するときに使用します。
- **Kerberos** — Kerberos は、ユーザ名やパスワードをネットワーク経由で送信せずに、Adaptive Server に対する Java アプリケーションまたは Java アプリケーション・ユーザの認証を行うときに使用します。また、シングル・サインオン (SSO) 環境を設定して Java アプリケーションのデジタル ID と Adaptive Server Enterprise のデジタル ID の間で相互認証を行う場合にも Kerberos を使用します。

注意 Kerberos は通信の暗号化やデータ整合性チェックにも使用できませんが、jConnect にはこれらの機能は実装されていません。

Kerberos と SSL を組み合わせることによって、SSO の利点と、クライアント・アプリケーションとサーバ・アプリケーションの間で送受信されるデータの暗号化の利点の両方を生かすこともできます。

制限事項

Kerberos と SSL は、Adaptive Server で使用できます。現在のところ、SQL Anywhere は SSL セキュリティと Kerberos セキュリティのどちらもサポートしていません。

jConnect で SSL や Kerberos を使用する前に、SSL と Kerberos の関連ドキュメントに目を通しておくことをおすすめします。この章の内容は、SSL や Kerberos が動作するようにサーバが正しく設定されていることを前提としています。

Kerberos と SSL、および Adaptive Server Enterprise の設定方法の詳細については、『[関連マニュアル](#)』(122 ページ) を参照してください。また、Kerberos の設定に関するホワイト・ペーパーも参照してください。この資料の URL については、『リリース・ノート jConnect for JDBC』を参照してください。

カスタム・ソケット・プラグインの実装

この項では、カスタム・ソケットの実装をアプリケーションにプラグインして、クライアントとサーバの間の通信をカスタマイズする方法について説明します。javax.net.ssl.SSLSocket は、たとえば暗号化を有効にするようカスタマイズできるソケットの一例です。

com.sybase.jdbcx.SybSocketFactory は、Sybase 拡張インタフェースの 1 つで、この中の createSocket(String, int, Properties) メソッドは java.net.Socket を返します。jConnect でカスタムのソケット・ファクトリを使用するには、createSocket() メソッドを定義してアプリケーションにこのインタフェースを実装する必要があります。

jConnect は、以降の入出力オペレーションに新しいソケットを使用します。SybSocketFactory を実装するクラスによってソケットを作成し、このクラスを枠組みとして、次のようにパブリックなソケット・レベル機能を追加することができます。

```
/**
 * Returns a socket connected to a ServerSocket on the named host,
 * at the given port.
 * @param host the server host
 * @param port the server port
 * @param props Properties passed in through the connection
 * @returns Socket
 * @exception IOException, UnknownHostException
 */
public java.net.Socket createSocket(String host, int port, Properties props)
    throws IOException, UnknownHostException;
```

プロパティを渡すことによって、`SybSocketFactory` のインスタンスが接続プロパティを使用してインテリジェントなソケットを実装することが可能になります。

ソケットを生成するために `SybSocketFactory` を実装する場合は、ソケットを作成するさまざまな種類のファクトリ、または擬似ファクトリをアプリケーションに渡すことによって、同じアプリケーション・コードで異なる種類のソケットを使用できるようになります。

ソケットの構成に使用されるパラメータを使用して、ファクトリをカスタマイズできます。たとえば、返されるソケットにそれぞれ異なるネットワーク・タイムアウトを設定したり、セキュリティ・パラメータを設定済みの状態で返したりするように、ファクトリをカスタマイズできます。アプリケーションに返されるソケットを、`java.net.Socket` のサブクラスとすることもできます。このようにすれば、圧縮、セキュリティ、レコード・マーキング、統計収集、ファイアウォール・トンネリング (`javax.net.SocketFactory`) などの機能に対する新しい API を直接公開できます。

注意 `SybSocketFactory` は、`javax.net.SocketFactory` を非常に簡略化したものとなるように作られており、アプリケーションでの `java.net.*` から `javax.net.*` へのブリッジを可能にします。

❖ jConnect でのカスタム・ソケットの使用

- 1 `com.sybase.jdbcx.SybSocketFactory` を実装する Java クラスの名前を指定します。「[カスタム・ソケットの作成と設定](#)」(108 ページ) を参照してください。
- 2 ソケットを取得するための独自の実装を `jConnect` が使用できるように、`SYB_SOCKET_FACTORY` 接続プロパティを設定します。

`jConnect` でカスタム・ソケットを使用するには、`SYB_SOCKET_FACTORY` 接続プロパティを、次のいずれかに設定してください。

- `com.sybase.jdbcx.SybSocketFactory` を実装するクラス名。
- `DEFAULT` (この場合は、新しい `java.net.Socket` がインスタンス化されます)。

`SYB_SOCKET_FACTORY` の設定方法については、「[接続プロパティ](#)」(10 ページ) を参照してください。

カスタム・ソケットの作成と設定

jConnect は、カスタム・ソケットを取得すると、そのソケットを使用してサーバに接続します。ソケットの設定は、jConnect がソケットを取得する前に完了している必要があります。

この項では、`javax.net.ssl.SSLSocket` などの SSL ソケットの実装を jConnect でプラグインする方法について説明します。

次の例は、SSL の実装がどのように `SSLSocket` のインスタンスを作成して設定し、返すかを示します。この例では、`MySSLSocketFactory` クラスが `SybSocketFactory` を実装し、`javax.net.ssl.SSLSocketFactory` を拡張して SSL を実装します。このクラスには 2 つの `createSocket` メソッドがあります。1 つは `SSLSocketFactory` に対するもので、もう 1 つは `SybSocketFactory` に対するものです。これらは次のことを行います。

- SSL ソケットを作成します。
- `SSLSocket.setEnabledCipherSuites` を呼び出して、暗号化に使用可能な暗号スイートを指定します。
- jConnect が使用するソケットを返します。

例

```
public class MySSLSocketFactory extends SSLSocketFactory
    implements SybSocketFactory
{
    /**
     * Create a socket, set the cipher suites it can use, return
     * the socket.
     * Demonstrates how cipher suites could be hard-coded into the
     * implementation.
     *
     * See javax.net.SSLSocketFactory#createSocket
     */
    public Socket createSocket(String host, int port)
        throws IOException, UnknownHostException
    {
        // Prepare an array containing the cipher suites that are to
        // be enabled.
        String enableThese[] =
        {
            "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA",
            "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5",
            "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA"
        }
        ;
        Socket s =
            SSLSocketFactory.getDefault().createSocket(host, port);
        ((SSLSocket)s).setEnabledCipherSuites(enableThese);
        return s;
    }
}
```



```

    }
    /**
     * Return an SSLSocket.
     * Demonstrates how to set cipher suites based on connection
     * properties like:
     * Properties _props = new Properties();
     * Set other url, password, etc. properties.
     * _props.put(("CIPHER_SUITES_1",
     *      "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA");
     * _props.put("CIPHER_SUITES_2",
     *      "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5");
     * _props.put("CIPHER_SUITES_3",
     *      "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA");
     * _conn = _driver.getConnection(url, _props);
     *
     * See com.sybase.jdbcx.SybSocketFactory#createSocket
     */
    public Socket createSocket(String host, int port,
        Properties props)
        throws IOException, UnknownHostException
    {
        // check to see if cipher suites are set in the connection
        // properties
        Vector cipherSuites = new Vector();
        String cipherSuiteVal = null;
        int cipherIndex = 1;
        do
        {
            if((cipherSuiteVal = props.getProperty("CIPHER_SUITES_"
                + cipherIndex++)) == null)
            {
                if(cipherIndex <= 2)
                {
                    // No cipher suites available
                    // return what the object considers its default
                    // SSLSocket, with cipher suites enabled.
                    return createSocket(host, port);
                }
                else
                {
                    // we have at least one cipher suite to enable
                    // per request on the connection
                    break;
                }
            }
            else
            {
                // add to the cipher suit Vector, so that
                // we may enable them together
                cipherSuites.addElement(cipherSuiteVal);
            }
        }
    }

```

```
while(true);
// lets you create a String[] out of the created vector
String enableThese[] = new String[cipherSuites.size()];
cipherSuites.copyInto(enableThese);
Socket s =
    SSLSocketFactory.getDefault().createSocket
        (host, port);
// enable the cipher suites
((SSLSocket)s).setEnabledCipherSuites(enableThese);
// return the SSLSocket
return s;
}
// other methods
}
```

jConnect はソケットの種類に関する情報を必要としないので、ソケットを返す前に設定を完了しておいてください。

詳細については、次を参照してください。

- *EncryptASE.java* – jConnect ディレクトリの *sample2* サブディレクトリにあります。このサンプルは、jConnect アプリケーションで **SybSocketFactory** インタフェースを使用する方法を示します。
- *MySSLSocketFactoryASE.java* – これも jConnect ディレクトリの *sample2* サブディレクトリにあります。これは、**SybSocketFactory** インタフェースの実装例で、アプリケーションにプラグインして使用できます。

Kerberos

Kerberos は、クライアント／サーバ・アプリケーションの認証を行うための、暗号化を使用するネットワーク認証プロトコルです。ユーザおよびシステム管理者から見た Kerberos の利点には、次のものがあります。

- Kerberos データベースによって、ユーザ情報の格納場所を一元化できます。
- シングル・サインオン (SSO) 環境を容易に構築でき、ユーザはシステムに一度ログインするだけで、データベースへのアクセスに必要なクレデンシャルを取得できます。
- Kerberos は IETF の標準の 1 つです。異なる Kerberos 実装間での相互運用が可能です。

Kerberos を使用するための jConnect アプリケーションの設定

jConnect で使用するために Kerberos を設定する前に、次のものがあることを確認してください。

- JDK 1.6 以降
- 次のいずれかの Java GSS (Generic Security Services) Manager
 - a デフォルトの Sun GSS Manager (JDK に含まれる)
 - b Wedgetail JCSI Kerberos バージョン 2.6 以降
 - c CyberSafe TrustBroker Application Security Runtime Library バージョン 3.1.0 以降
 - d 他のベンダの GSSManager 実装
- GSS ライブラリが存在するサーバ側と GSSManager が存在するクライアント側の両方でサポートされていて相互運用可能な KDC

jConnect で Kerberos ログインを使用できるようにするには、次の手順に従います。

❖ jConnect で使用するための Kerberos の設定

- 1 REQUEST_KERBEROS_SESSION プロパティを “true” に設定します。
- 2 SERVICE_PRINCIPAL_NAME プロパティを、Adaptive Server Enterprise が実行されているマシンの名前に設定します。通常、これは、サーバの起動時に **-s** オプションで設定される名前です。サービスのプリンシパル名は、KDC にも登録されている必要があります。SERVICE_PRINCIPAL_NAME プロパティの値が設定されていない場合のデフォルトは、クライアント・マシンのホスト名です。
- 3 GSSMANAGER_CLASS プロパティを設定します (省略可能)。

REQUEST_KERBEROS_SESSION プロパティと SERVICE_PRINCIPAL_NAME プロパティの詳細については、「[第2章 プログラミング情報](#)」を参照してください。GSSMANAGER_CLASS プロパティの詳細については、「[GSSMANAGER_CLASS 接続プロパティ](#)」を参照してください。

GSSMANAGER_CLASS 接続プロパティ

Kerberos を使用するとき、jConnect は、GSS (Generic Security Services) API を実装する多数の Java クラスに依存します。この機能の多くは、org.ietf.jgss.GSSManager クラスによって実現されます。

ベンダの実装

Java では、GSSManager クラスの独自の実装をベンダが作成することが可能です。ベンダ提供の GSSManager 実装の例には、Wedgetail Communications や CyberSafe Limited から提供される実装があります。ユーザは、ベンダ提供の GSSManager クラスを、特定の Kerberos 環境で動作するように設定できます。ベンダ提供の GSSManager クラスには、Java 標準の GSSManager クラスよりも、Windows との相互運用性が高いものもあります。

ベンダ提供の GSSManager を実装する前に、各ベンダのドキュメントに必ず目を通してください。ベンダ実装では、Kerberos に関して Java 標準のシステム・プロパティとは別のシステム・プロパティ設定が使用されています。また、構成ファイル以外の場所からレルム名と KDC (Key Distribution Center) エントリを取得するものもあります。

GSSMANAGER_CLASS の設定

ベンダ実装の GSSManager を jConnect とともに使用するには、GSSMANAGER_CLASS 接続プロパティを設定します。このプロパティを設定するには、次の 2 とおりの方法があります。

- GSSManager のインスタンスを作成し、このインスタンスを GSSMANAGER_CLASS プロパティの値として設定します。
- GSSMANAGER_CLASS プロパティの値として、GSSManager オブジェクトの完全修飾クラス名を指定する文字列を設定します。jConnect はこの文字列を使用して `Class.forName().newInstance()` を呼び出し、返されたオブジェクトを GSSManager クラスとしてキャストします。

どちらの場合も、アプリケーションの CLASSPATH 変数に、ベンダ実装のクラスと *.jar* ファイルの場所が含まれている必要があります。

注意 GSSMANAGER_CLASS 接続プロパティが設定されていない場合は、jConnect は `org.ietf.jgss.GSSManager.getInstance` メソッドを使用して Java のデフォルトの GSSManager 実装をロードします。

GSSMANAGER_CLASS 接続プロパティを使用して完全修飾クラス名を渡すと、GSSManager の引数なしのコンストラクタが呼び出されます。これによってインスタンス化される GSSManager はベンダ実装のデフォルト設定であるので、アプリケーション側で GSSManager オブジェクトの設定を詳しく制御することはできません。GSSManager のインスタンスを独自に作成する場合は、コンストラクタ引数を使用して設定オプションを設定できます。

jConnect が GSSMANAGER_CLASS を使用する方法

jConnect は、初めに、GSSMANAGER_CLASS の値が、Kerberos 認証に使用する GSSManager クラス・オブジェクトであるかどうかを調べます。

GSSMANAGER_CLASS の値が、クラス・オブジェクトではなく文字列として設定されている場合は、その文字列を使用して、指定されているクラスのインスタンスを作成し、そのインスタンスを Kerberos 認証に使用します。

GSSMANAGER_CLASS の値に GSSManager クラス・オブジェクトでも文字列でもないものが設定されている場合や、ClassCastException が発生した場合は、問題を通知する SQLException が発生します。

例

次の例は、GSSManager のインスタンスを独自に作成する方法と、GSSMANAGER_CLASS 接続プロパティに完全修飾クラス名を設定して GSSManager オブジェクトを自動的に作成する方法を示します。どちらの例も、Wedgetail GSSManager を使用します。

❖ 例：GSSManager の独自のインスタンスの作成

- 1 アプリケーション・コードの中で GSSManager をインスタンス化します。例：

```
GSSManager gssMan = new com.dstc.security.kerberos.gssapi.GSSManager();
```

この例では、引数なしのデフォルト・コンストラクタを使用します。他のベンダ提供コンストラクタを使用して、各種オプションを設定することもできます。

- 2 新しい GSSManager インスタンスを GSSMANAGER_CLASS 接続プロパティに渡します。例：

```
Properties props = new Properties();
props.put("GSSMANAGER_CLASS", gssMan);
```

- 3 接続には、GSSMANAGER_CLASS を含むこれらの接続プロパティを使用します。例：

```
Connection conn = DriverManager.getConnection (url, props);
```

❖ 例：GSSMANAGER_CLASS に文字列を渡す

- 1 アプリケーション・コードの中で、GSSManager オブジェクトの完全修飾クラス名を指定する文字列を作成します。例：

```
String gssManClass = "com.dstc.security.kerberos.gssapi.GSSManager";
```

- 2 この文字列を GSSMANAGER_CLASS 接続プロパティに渡します。例：

```
Properties props = new Properties();
props.put("GSSMANAGER_CLASS", gssManClass);
```

- 3 接続には、GSSMANAGER_CLASS を含むこれらの接続プロパティを使用します。たとえば、

```
Connection conn = DriverManager.getConnection (url, props);
```

Kerberos 環境の設定

この項では、次の 3 つの Kerberos 実装を jConnect とともに使用するときの環境設定における注意事項を説明します。

- [CyberSafe](#)
- [MIT](#)
- [Microsoft Active Directory](#)

注意 この項を読む前に、Kerberos white paper に目を通してください。
(<http://www.sybase.com/detail?id=1029260>)

CyberSafe

暗号化キー

Java によって使用されるプリンシパルを CyberSafe KDC 内に作成するときに、DES (Data Encryption Standard) キーを指定します。Java リファレンス実装は 3DES (Triple Data Encryption Standard) キーをサポートしません。

注意 CyberSafe GSSManager を CyberSafe KDC とともに使用する場合に、GSSMANAGER_CLASS プロパティを設定すれば、3DES キーを使用できます。

アドレス・マッピングとレルム情報

CyberSafe Kerberos は、*krb5.conf* 構成ファイルを使用しません。デフォルトでは、CyberSafe は DNS レコードを使用して、KDC アドレス・マッピングとレルム情報を取得します。または、*krb.conf* ファイルと *krb.realms* ファイルを使用して、KDC アドレス・マッピングとレルム情報をそれぞれ取得します。詳細については、CyberSafe のドキュメントを参照してください。

Java 標準の GSSManager 実装を使用する場合は、Java によって使用される *krb5.conf* ファイルを作成する必要があります。CyberSafe の *krb.conf* ファイルのフォーマットは、*krb5.conf* ファイルとは異なります。Sun のマニュアル・ページまたは MIT のドキュメントの指定に従って、*krb5.conf* ファイルを作成してください。CyberSafe の GSSManager を使用する場合は、*krb5.conf* ファイルは必要ありません。

krb5.conf ファイルの例については、Kerberos の設定に関するホワイト・ペーパーを参照してください。この資料の URL については、『リリース・ノート jConnect for JDBC』を参照してください。

Solaris

Solaris 上で CyberSafe クライアント・ライブラリを使用する場合は、ライブラリ検索パス内で、CyberSafe ライブラリを他のすべての Kerberos ライブラリよりも先に指定してください。

MIT

暗号化キー

Java によって使用されるプリンシパルを MIT KDC 内に作成するときに、DES キーを指定します。Java リファレンス実装は、3DES キーをサポートしません。

Java 標準の GSSManager 実装のみを使用する場合は、des-cbc-crc タイプまたは des-cbc-md5 タイプの暗号化キーを指定してください。暗号化のタイプは次のように指定します。

```
des-cbc-crc:normal
```

normal はキー salt のタイプです。他のタイプの salt を使用することもできます。

注意 Wedgetail の GSSManager を使用する場合は、des3-cbc-sha1-kd タイプのプリンシパルを MIT KDC 内に作成できます。

Microsoft Active Directory

ユーザ・アカウントと
サービス・プリンシパル

ユーザ・プリンシパル (ユーザ) とサービス・プリンシパル (データベース・サーバを表すアカウント) に対するアカウントが Active Directory 内で設定されていることを確認してください。ユーザ・プリンシパルとサービス・プリンシパルの両方を Active Directory 内の Users として作成してください。

暗号化

Java リファレンスの GSS Manager 実装を使用する場合は、ユーザ・プリンシパルとサービス・プリンシパルの両方に DES 暗号化を使用してください。

❖ DES 暗号化の設定

- 1 Active Directory の Users リストで、特定のユーザ・プリンシパル名またはサービス・プリンシパル名を右クリックします。
- 2 プロパティを選択します。
- 3 [アカウント] タブをクリックします。[アカウント オプション] リストが表示されます。
- 4 ユーザ・プリンシパルとサービス・プリンシパルの両方に、DES 暗号化を使用することを指定してください。

クライアント・マシン

Java リファレンス実装を使用して SSO 環境を設定する場合は、Windows レジストリの変更が必要になる場合があります。Microsoft support site (<http://support.microsoft.com/>) で説明している手順に従ってください。

構成ファイル

Windows では、Kerberos 構成ファイルの名前は `krb5.ini` です。Java のデフォルトでは、`C:\WINNT\krb5.ini` の `krb5.ini` が使用されます。このファイルの場所を指定することもできます。`krb5.ini` のフォーマットは `krb5.conf` と同じです。

`krb5.conf` ファイルの例については、Kerberos の設定に関するホワイト・ペーパーを参照してください。この資料の URL については、『リリース・ノート jConnect for JDBC』を参照してください。

Microsoft Active Directory での Kerberos の詳細については、Microsoft Developer Network (<http://msdn.microsoft.com>) を参照してください。

サンプル・アプリケーション

`jConnect-7_0/sample2` ディレクトリにある次の 2 つのコード・サンプルでは、Kerberos を使用して Adaptive Server Enterprise への接続を確立する方法をコメント付きで説明しています。

- `ConnectKerberos.java` — Adaptive Server Enterprise への単純な Kerberos ログイン
- `ConnectKerberos.JAAS.java` — アプリケーション・サーバ・コード内に Kerberos ログインを実装する方法を示す詳細なサンプル

ConnectKerberos.java

`ConnectKerberos.java` サンプル・アプリケーションを実行するには、次の手順に従います。

❖ `ConnectKerberos.java` の実行

- 1 使用するマシンに有効な Kerberos クレデンシャルがあることを確認します。クレデンシャルを取得する方法は、マシンと環境によって異なります。

Windows — Active Directory 環境内のマシンでは、Kerberos 認証を使用してログインに成功したときに Kerberos クレデンシャルが確立されます。

UNIX または Linux — UNIX または Linux のマシンでは、Kerberos クライアント用の `kinit` ユーティリティを使用して、Kerberos クレデンシャルを確立できます。`kinit` を使用して最初のクレデンシャルを取得しない場合は、サンプル・アプリケーションを実行するときにユーザ名とパスワードの入力を求められます。

注意 Sun JDK では、使用できる暗号化タイプは `DES_CBC_MD5` と `DES_CBC_CRC` だけです。サード・パーティのソフトウェアを使用して、`GSSMANAGER_CLASS` プロパティを設定すれば、他の暗号化タイプを使用できる場合があります。

- 2 マシンのクレデンシャルの場所を確認します。

Windows — Active Directory 環境内で実行されるマシンでは、Kerberos クレデンシャルはメモリ内のチケット・キャッシュに格納されます。

UNIX または Linux — Sun Java, CyberSafe, Solaris, または MIT の Kerberos 実装を使用する UNIX または Linux のマシンでは、`kinit` を実行したときにデフォルトでは `/tmp/krb5cc_{user_id_number}` にクレデンシャルが作成されます。`{user_id_number}` はユーザ名固有の値です。

クレデンシャルが別の場所に配置されている場合は、`sample2/exampleLogin.conf` ファイル内で `ticketCache` プロパティを設定して場所を指定してください。

- 3 Java リファレンス実装に対して、KDC マシンのデフォルトのレルム名とホスト名を指定します。Java は、この情報を構成ファイル `krb5.conf` または `krb5.ini`、または Java System プロパティから取得することもできます。ベンダ提供の GSS Manager 実装では、DNS SRV レコードからホスト情報とレルム情報が取得される場合もあります。

Kerberos 構成ファイルでは、認証時に要求する暗号化タイプの指定など、Kerberos 環境の詳細な制御が可能となるので、構成ファイルを使用することをおすすめします。

注意 Linux では、Java リファレンス実装は `/etc/krb5.conf` の Kerberos 構成ファイルを使用します。

Kerberos 構成ファイルを使用せず、DNS SRV レコードを使用するように Kerberos を設定していないときは、システム・プロパティ `java.security.krb5.realm` と `java.security.krb5.kdc` を使用してレルムと KDC を指定できます。

- 4 接続 URL が目的のデータベースを指すように、`ConnectKerberos.java` を編集します。
- 5 `ConnectKerberos.java` をコンパイルします。

JDK バージョン 1.6 以降を使用してください。ソース・コードのコメントに目を通してください。また、`CLASSPATH` 環境変数で、`jConnect` インストール環境の `jconn4.jar` が指定されていることを確認してください。

- 6 `ConnectKerberos.class` を実行します。

```
java ConnectKerberos
```

java バージョン 1.6 の実行プログラムを使用してください。サンプル・アプリケーションの出力に、接続の確立に成功したと、次に示す SQL を実行することが表示されます。

```
select 1
```

- Kerberos 構成ファイルを使用せずにこのサンプルを実行するには、次のコマンドを使用します。

```
java -Djava.security.krb5.realm=your_realm  
-Djava.security.krb5.kdc=your_kdc ConnectKerberos
```

your_realm はデフォルトのレルム、*your_kdc* は使用する KDC です。

- 必要に応じて、サンプル・アプリケーションをデバッグ・モードで実行すると、Java Kerberos レイヤからのデバッグ出力を確認できます。

```
java -Dsun.security.krb5.debug=true ConnectKerberos
```

また、`isql` の Java バージョンである `IsqlApp` を使用して Kerberos 接続を確立することもできます。`IsqlApp` は `jConnect-7_0/classes` ディレクトリにあります。

```
java IsqlApp -S jdbc:sybase:Tds:hostName:portNum  
-K service_principal_name  
-F path_to_JAAS_login_module_config_file
```

`IsqlApp` の使用方法については、「`jConnect` サンプル・プログラム」の章を参照してください。

***krb5.conf* 構成ファイル**

次は、*krb5.conf* ファイルの例です。

CyberSafe または MIT KDC

クライアントで CyberSafe または MIT KDC とともに使用する *krb5.conf* ファイルの例を次に示します。

```
# Please note that customers must alter the  
# default_realm, [realms] and [doamin_realm]  
# information to reflect their Kerberos environment.  
# Customers should *not* attempt to use this file as is.  
#  
  
[libdefaults]  
    default_realm = ASE  
    default_tgs_enctypes = des-cbc-crc  
    default_tkt_enctypes = des-cbc-crc  
    kdc_req_checksum_type = 2  
    ccache_type = 2  
  
[realms]  
  
    ASE = {  
        kdc = kdchost  
        admin_server = kdchost
```

```

    }

[domain_realm]
    .sybase.com = ASE
    sybase.com = ASE

[logging]
    default = FILE:/var/krb5/kdc.log
    kdc = FILE:/var/krb5/kdc.log
    kdc_rotate = {

# How often to rotate kdc.log. Logs will get rotated
# no more often than the period, and less often if the
# KDC is not used frequently.

        period = 1d

# how many versions of kdc.log to keep around
# (kdc.log.0, kdc.log.1, ...)

        versions = 10
    }

[appdefaults]
    kinit = {
        renewable = true
        forwardable = true
    }

```

Active Directory KDC

クライアントで Active Directory とともに KDC として使用する *krb5.conf* ファイルの例を次に示します。

```

# Please note that customers must alter the
# default_realm, [realms] and [domain_realm]
# information to reflect their Kerberos environment.
# Customers should *not* attempt to use this file as is.
#

[libdefaults]
    default_realm = W2K.SYBASE.COM
    default_tgs_enctypes = des-cbc-crc
    default_tkt_enctypes = des-cbc-crc
    kdc_req_checksum_type = 2
    ccache_type = 2

[realms]

    W2K.SYBASE.COM = {
        kdc = 1.2.3.4:88
    }

```

```
        admin_server = adserver
    }

[domain_realm]
    .sybase.com = W2K.SYBASE.COM
    sybase.com = W2K.SYBASE.COM

[logging]
    default = FILE:/var/krb5/kdc.log
    kdc = FILE:/var/krb5/kdc.log
    kdc_rotate = {

# How often to rotate kdc.log. Logs will get rotated no
# more often than the period, and less often if the KDC
# is not used frequently.

        period = 1d

# how many versions of kdc.log to keep around
# (kdc.log.0, kdc.log.1, ...)

        versions = 10
    }

[appdefaults]
    kinit = {
        renewable = true
        forwardable= true
    }
```

相互運用性

Sybase では、表 3-1 に示す KDC、GSS ライブラリ、プラットフォームの組み合わせについて、Adaptive Server Enterprise への接続を正しく確立できることを検証しました。この表にない組み合わせでは、接続を確立できないというわけではありません。最新の検証結果は、jConnect for JDBC Web site (<http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>) で参照できます。

表 3-1: 相互運用可能な組み合わせ

クライアント・プラットフォーム	KDC	GSSManager	GSS C ライブラリ ^a	ASE プラットフォーム
Solaris 8 ^b	CyberSafe	Java GSS	CyberSafe	Solaris 8
Solaris 8	Active Directory ^c	Java GSS	CyberSafe	Solaris 8
Solaris 8	MIT	Java GSS	CyberSafe	Solaris 8
Solaris 8	MIT	Wedgetail GSS ^d	MIT	Solaris 8
Solaris 8	CyberSafe	Wedgetail GSS ^e	CyberSafe	Solaris 8

クライアント・プラットフォーム	KDC	GSSManager	GSS C ライブラリ ^a	ASE プラットフォーム
Windows 2000	Active Directory	Java GSS	CyberSafe	Solaris 8
Windows XP	Active Directory	Java GSS ^f	CyberSafe	Solaris 8

a. Adaptive Server Enterprise の GSS 機能用に使用されるライブラリです。

b. この表の Solaris 8 プラットフォームはすべて 32 ビット版です。

c. この表の Active Directory はいずれも Windows 2000 上で実行される Active Directory サーバを指します。Kerberos の相互運用を可能にするには、Active Directory ユーザの設定の「このアカウントに DES 暗号化を使う」を有効にする必要があります。

d. Wedgetail JCSI Kerberos 2.6 を使用しました。暗号化タイプは 3DES です。

e. Wedgetail JCSI Kerberos 2.6 を使用しました。暗号化タイプは DES です。

f. Java 1.4.x では、クライアントが `System.setProperty("os.name", "Windows 2000")` ; を使用していないと Windows XP クライアントのメモリ内のクレデンシャルを Java が検出できないというバグが発生します。

これらのライブラリの最新バージョンを使用することをおすすめします。古いバージョンを使用する場合や、Sybase 以外の製品の問題がある場合は、各ベンダにお問い合わせください。

暗号化タイプ

Sun が提供する Java 標準の GSS 実装では、DES 暗号化のみがサポートされます。暗号化標準 3DES、RC4-HMAC、AES-256、または AES-128 を使用する場合は、CyberSafe または Wedgetail の GSSManager を使用してください。

Wedgetail と CyberSafe の詳細については、それぞれのドキュメントを参照してください。

トラブルシューティング

この項では、Kerberos セキュリティのトラブルシューティングを行うときに考慮が必要な事項について説明します。

Kerberos

Kerberos セキュリティの問題のトラブルシューティングを行うときは、次の点を考慮してください。

- Java リファレンス実装では、DES 暗号化タイプのみがサポートされます。DES 暗号化を使用するように、Active Directory と KDC プリンシパルを設定する必要があります。
- SERVICE_PRINCIPAL_NAME プロパティの値は、データ・サーバの起動時に `-s` オプションで指定した名前と同じになるように設定する必要があります。
- `krb5.conf` ファイルと `krb5.ini` ファイルを確認します。CyberSafe クライアントの場合は、`krb.conf` ファイルと `krb.realms` ファイルまたは DNS SRV レコードを確認します。

- JAAS ログイン構成ファイル内の `debug` プロパティを “true” に設定できます。
- 次のように、コマンド・ラインで `debug` プロパティを “true” に設定できます。

```
-Dsun.security.krb5.debug=true
```

- JAAS ログイン構成ファイルには、目的に合わせて設定できる多数のオプションがあります。JAAS および Java GSS API の詳細については、次を参照してください。
 - JAAS login configuration file
(<http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/LoginConfigFile.html>)
 - Class Krb5LoginModule
(<http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html>)
 - Troubleshooting JGSS
(<http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/Troubleshooting.html>)

関連マニュアル

次のドキュメントには、Kerberos セキュリティに関するその他の情報が記載されています。

- Java tutorial on JAAS and the Java GSS API
(<http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/index.html>)
- MIT Kerberos documentation and download site
(<http://web.mit.edu/kerberos/www/index.html>)
- CyberSafe Limited (<http://www.cybersafe.ltd.uk>)
- CyberSafe Limited document on Windows-Kerberos interoperability
(http://www.cybersafe.ltd.uk/docs_cybersafe/Kerberos%20Interoperability%20-%20Microsoft%20W2k%20&%20ActiveTRUST.pdf)
- Description of how Windows implements authentication, including information about Active Directory Kerberos
(<http://www.windowsitlibrary.com/Content/617/06/1.html>)
- Kerberos RFC 1510 (<http://www.linuxdig.com/rfc/individual/1510.php>)

この章では、jConnect を使用しているときに発生することがある問題の解決法と対処方法について説明します。

トピック名	ページ
jConnect でのデバッグ	123
TDS 通信の取得	126
接続エラーの解決	128
jConnect アプリケーションでのメモリ管理	129
ストアド・プロシージャのエラーの解決	130
カスタム・ソケット実装エラーの解決	131

jConnect でのデバッグ

jConnect には、一連のデバッグ関数が含まれている **Debug** クラスがあります。**Debug** のメソッドには、さまざまな **assert**、**trace**、**timer** の関数があり、デバッグ処理の適用範囲と、デバッグ結果の出力先を定義できます。

jConnect のインストール環境には、デバッグに使用できる一連のクラスも含まれます。これらのクラスは、jConnect インストール・ディレクトリの下での *devclasses* サブディレクトリに置かれます。デバッグを行うときは、CLASSPATH 環境変数が jConnect 標準の *classes* ディレクトリではなく、デバッグ・モード・ランタイム・クラス (*devclasses/jconn4d.jar*) を参照するように変更する必要があります。または、Java プログラムを実行するときに **java** コマンドの **-classpath** 引数を明示的に指定します。

Debug クラスのインスタンスの取得

アプリケーションで jConnect のデバッグ機能を使用するには、**Debug** インタフェースをインポートし、**SybDriver** クラスの **getDebug** メソッドを呼び出すことによって、**Debug** クラスのインスタンスを取得する必要があります。

```
import com.sybase.jdbcx.Debug;
//
...
SybDriver sybDriver = (SybDriver)
Class.forName("com.sybase.jdbc4.jdbc.SybDriver").newInstance();
Debug sybdebug = sybDriver.getDebug();
...
```

アプリケーションのデバッグをオンにする方法

Debug オブジェクトの `debug` メソッドを使用してアプリケーション内でのデバッグをオンにするには、次の呼び出しを追加してください。

```
sybdebug.debug(true, [classes], [printstream]);
```

`classes` パラメータは、デバッグするクラスをコロンで区切って指定した文字列です。例：

```
sybdebug.debug(true, "MyClass")
```

および

```
sybdebug.debug(true, "MyClass:YourClass")
```

クラス文字列内で“STATIC”を使用すると、指定したクラスの他に jConnect のすべての `static` メソッドに対するデバッグがオンになります。例：

```
sybdebug.debug(true, "STATIC:MyClass")
```

“ALL”を指定すると、すべてのクラスに対するデバッグをオンにできます。例：

```
sybdebug.debug(true, "ALL");
```

`printstream` パラメータは省略可能です。`printstream` を指定しない場合は、デバッグ出力は `DriverManager.setLogStream` で指定したファイルに出力されます。

アプリケーションのデバッグをオフにする方法

デバッグをオフにするには、次の呼び出しを追加します。

```
sybdebug.debug(false);
```

デバッグ用に CLASSPATH を設定する方法

デバッグが有効なアプリケーションを実行する前に、最適化した jConnect ドライバの jar ファイル `jconn4.jar` をデバッグ・バージョンの `jconn4d.jar` と置き換えて、jConnect インストール・ディレクトリの `/devclasses` サブディレクトリで見つけることができるようにします。

CLASSPATH 環境変数を使用する場合は、次のようにします。

- UNIX の場合は、`$JDBC_HOME/classes/jconn4.jar` の代わりに `$JDBC_HOME/devclasses/jconn4d.jar` を使用します。
- Windows の場合は、`%JDBC_HOME%\classes\jconn4.jar` の代わりに `%JDBC_HOME%\devclasses\jconn4d.jar` を使用します。

Debug のメソッドの使用

デバッグ処理をカスタマイズするために、**Debug** の他のメソッドの呼び出しを追加することもできます。

次の各メソッドでは、最初の (オブジェクト) パラメータとして、通常は **this** を使用して呼び出し元のオブジェクトを指定します。これらのメソッドのうち、静的のものには、オブジェクト・パラメータとして **null** を使用してください。

- **println**

このメソッドは、デバッグが有効で、デバッグするクラスのリストにオブジェクトが含まれている場合に、出力ログに出力するメッセージを定義するために使用します。デバッグ出力は、**sybdebug.debug** で指定されたファイルに出力されます。

構文は次のとおりです。

```
sybdebug.println(object,message string);
```

次に例を示します。

```
sybdebug.println(this,"Query: "+ query);
```

このメッセージに似たメッセージが出力ログに出力されます。

```
myApp(thread[x,y,z]): Query: select * from authors
```

- **assert**

このメソッドは、条件を表明して、その条件が満たされないときに実行時例外を発生させるために使用します。条件が満たされない場合に出力ログに出力するメッセージを定義することもできます。構文は次のとおりです。

```
sybdebug.assert(object,boolean condition,message string);
```

次に例を示します。

```
sybdebug.assert(this,amount<=buf.length,amount+"
too big!");
```

この例では、“amount” が **buf.length** の値を超えると、出力ログに次のようなメッセージが出力されます。

```
java.lang.RuntimeException:myApp(thread[x,y,z]):
Assertion failed: 513 too big!
at jdbc.sybase.utils.sybdebug.assert(
sybdebug.java:338)
at myApp.myCall(myApp.java:xxx)
at .... more stack:
```

- `startTimer`
`stopTimer`

これらのメソッドは、イベント中に経過時間をミリ秒単位で計測するタイマを開始したり停止したりするために使用します。このメソッドは、オブジェクトごとに1つと、すべての静的メソッドに対する1つのタイマを保持します。タイマを開始する構文は次のとおりです。

```
sybdebug.startTimer(object);
```

タイマを停止する構文は次のとおりです。

```
sybdebug.stopTimer(object,message string);
```

次に例を示します。

```
sybdebug.startTimer(this);  
stmt.executeQuery(query);  
sybdebug.stopTimer(this,"executeQuery");
```

このメッセージに似たメッセージが出力ログに出力されます。

```
myApp(thread[x,y,z]):executeQuery elapsed time =  
25ms
```

TDS 通信の取得

Tabular Data Stream (TDS) は、クライアント・アプリケーションと Adaptive Server の間の通信を処理する Sybase 独自のプロトコルです。jConnect には、TDS パケットをそのままの形でファイルに取得するための `PROTOCOL_CAPTURE` 接続プロパティがあります。

アプリケーションで発生した問題を、アプリケーションでもサーバでも解決できない場合に、`PROTOCOL_CAPTURE` を使用してクライアントとサーバの間の通信をファイルに取得できます。このファイルには、直接には解析できないバイナリ・データが格納されています。このファイルを、Sybase 製品の保守契約を結んでいるサポート・センタに送付して解析を依頼してください。

注意 `Ribo` ユーティリティを使用して、クライアントとサーバの間で送受信されるプロトコル・ストリームを取得、変換、表示することができます。`Ribo` のインストールは、Sybase Software Developer's Kit のインストール時に実行できます。

PROTOCOL_CAPTURE 接続プロパティ

PROTOCOL_CAPTURE 接続プロパティは、アプリケーションと Adaptive Server の間で交換される TDS パケットを受信するファイルを指定するときに使用します。PROTOCOL_CAPTURE の設定はすぐに反映されるので、接続の確立中に交換された TDS パケットも指定したファイルに書き込まれます。Capture.pause が実行されるか、セッションがクローズするまで、すべてのパケットがファイルに書き込まれます。

次の例では、PROTOCOL_CAPTURE を使用して TDS データを *tds_data* ファイルに送ります。

```
...
props.put("PROTOCOL_CAPTURE", "tds_data");
Connection conn = DriverManager.getConnection(url, props);
```

url は接続 URL、*props* は接続プロパティを指定するための Properties オブジェクトです。

Capture クラスの pause メソッドと resume メソッド

Capture クラスは、com.sybase.jdbcx パッケージに含まれています。このクラスには、次の2つのパブリック・メソッドがあります。

- public void pause
- public void resume

Capture.pause は、ロー TDS パケットをファイルに取得する処理を停止します。Capture.resume は取得を再開します。

セッション全体の TDS を取得したファイルは、非常に大きくなる場合があります。アプリケーションのどの部分で TDS データを取得するかがわかっているれば、次の手順に従うことによって、このファイルのサイズを抑えることができます。

❖ 取得するファイルのサイズを制限するには

- 1 接続を確立した直後に、その接続に対する Capture オブジェクトを取得し、pause メソッドを使用して TDS データの取得を停止します。

```
Capture cap = ((SybConnection)conn).getCapture();
cap.pause();
```

- 2 TDS データの取得を開始する場所の直前に cap.resume を置きます。
- 3 データの取得を停止する場所の直後に cap.pause を置きます。

接続エラーの解決

この項では、接続を確立するときやゲートウェイを起動するときに発生する問題について説明します。

ゲートウェイ接続が拒否される

```
Gateway connection refused:  
HTTP/1.0 502 Bad Gateway|Restart Connection
```

このエラー・メッセージは、Adaptive Server に接続するために指定されたホスト名またはポート番号に何らかの問題があることを示します。
`$SYBASE/interfaces` (UNIX の場合) または `%SYBASE%\ini\sql.ini` (Windows の場合) の `[query]` エントリを調べてください。

ホスト名とポート番号が正しいことを確認した後も引き続き問題が発生する場合は、“verbose” システム・プロパティを使用して HTTP サーバを起動することによって、さらに情報を得ることができます。

Windows の場合は、DOS プロンプトで次のように入力します。

```
httpd -Dverbose=1 > filename
```

UNIX の場合は、次のように入力します。

```
sh httpd.sh -Dverbose=1 > filename &
```

filename は、デバッグ・メッセージの出力ファイルです。

Web サーバが **connect** メソッドをサポートしていない可能性があります。アプレットから接続できるホストは、そのアプレットのダウンロード元のホストだけです。

HTTP ゲートウェイと Web サーバは同じホストで稼働する必要があります。この場合、アプレットは、要求を適切なデータベースにルーティングする HTTP ゲートウェイによって制御されるポートを使用して、同じマシンおよびホストに接続できます。

これがどのように行われるかについては、jConnect インストール・ディレクトリの *sample2* サブディレクトリにある *Isql.java* と *gateway.html* のソースを参照してください。これらのファイルで、“proxy” を検索してください。

jConnect アプリケーションでのメモリ管理

jConnect アプリケーションでのメモリ使用量が増加したときは、次に示す状況とその解決法を参考にしてください。

- jConnect アプリケーションでは、メモリ内に文が累積することを防ぐために、**Statement** オブジェクトとサブクラス (たとえば **PreparedStatement**、**CallableStatement**) を、最後に使用した後で明示的にクローズするようにしてください。**ResultSet** をクローズするだけでは十分ではありません。

たとえば、次の文を使用すると問題が発生します。

```
ResultSet rs = _conn.prepareStatement(_query).execute();
...
rs.close();
```

代わりに、次のようにします。

```
PreparedStatement ps = _conn.prepareStatement(_query);
ResultSet rs = ps.executeQuery();
...
rs.close();
ps.close();
```

- スクロール可能または更新可能なカーソルのネイティブ・サポートは、接続中の **Adaptive Server** または **SQL Anywhere** のデータベースのバージョンによっては、利用できない場合があります。バックエンド・サーバでネイティブにサポートされていない場合、jConnect は **ResultSet.next** を呼び出すたびに要求されたロー・データをクライアント上にキャッシュすることで、スクロール可能または更新可能なカーソルをサポートします。しかし、結果セットの最後に到達したときは、結果セット全体がクライアントのメモリに格納されています。これによってパフォーマンス上の問題が発生するので、**TYPE_SCROLL_INSENSITIVE** の結果セットは、結果セットが比較的小さい場合のみに使用することをおすすめします。このリリースでは、jConnect は **Adaptive Server** 接続がネイティブなスクロール可能カーソル機能をサポートするかどうかを判別し、クライアント側キャッシュの代わりにそれを使用します。その結果、ほとんどのアプリケーションでは、順序が正しくないローにアクセスするときの大幅なパフォーマンスの向上と、クライアント側で必要なメモリの減少を期待できます。

ストアド・プロシージャのエラーの解決

この項では、jConnect とストアド・プロシージャを使用するときに発生する問題について説明します。

RPC が返す出力パラメータの数が登録されている数よりも少ない

SQLState: JZ0SG - An RPC did not return as many output parameters as the application had registered for it.

このエラーは、`CallableStatement.registerOutParam` を呼び出して登録したパラメータの数が、ストアド・プロシージャの“OUTPUT” パラメータとして宣言されている数よりも多い場合に発生します。該当するすべてのパラメータを“OUTPUT”として宣言していることを確認してください。それには、コードの次の行を調べてください。

```
create procedure yourproc (@pl int OUTPUT, ...
```

注意 SQL Anywhere を使用しているときにこのエラーが発生した場合は、SQL Anywhere バージョン 5.5.04 以降にアップグレードしてください。

出力パラメータが返される場合のフェッチとステータスのエラー

ロー・データを返さないクエリの場合は、`executeQuery` メソッドではなく、`CallableStatement.executeUpdate` メソッドまたは `execute` メソッドを使用してください。

JDBC 標準で要求されているように、`executeQuery` に結果セットがない場合は、jConnect は SQL 例外が発生します。

ストアド・プロシージャを非連鎖トランザクション・モードでしか実行できない

Sybase Error 7713 - Stored Procedure can only be executed in unchained transaction mode.

このエラーは、JDBC が接続を `autocommit(true)` モードに切り替えようとしたときに発生します。アプリケーションは、`Connection.setAutoCommit(false)` または“set chained on” 言語コマンドを使用することによって接続を連鎖モードに変更できます。このエラーは、ストアド・プロシージャが互換モードで作成されていない場合に発生します。

この問題を解決するには、次のシステム・プロシージャを使用します。

```
sp_procxmode procedure_name,"anymode"
```

カスタム・ソケット実装エラーの解決

SSL ソケットを設定しようとしているときに、`sun.security.ssl.SSLSocketImpl.setEnabledCipherSuites` を呼び出すと、次のような例外を受け取る場合があります。

```
java.lang.IllegalArgumentException:  
    SSL_SH_anon_EXPORT_WITH_RC4_40_MD5
```

この場合は、SSL ライブラリがシステム・ライブラリ・パスにあることを確認してください。

この章では、jConnect を使用するときのパフォーマンスの微調整または改善の方法について説明します。

トピック名	ページ
jConnect のパフォーマンスの改善	133
動的 SQL の prepared 文のパフォーマンス・チューニング	135
カーソルのパフォーマンス	141

jConnect のパフォーマンスの改善

jConnect を使用するアプリケーションのパフォーマンスを最適化するには、次のようにさまざまな方法があります。

- text データや image データを Adaptive Server データベースに送信するには、`TextPointer.sendData` メソッドを使用します。[「データベース内の image データの更新」\(59 ページ\)](#) を参照してください。
- セッション中に何度も使用される動的 SQL 文については、プリコンパイルされた `PreparedStatement` オブジェクトを作成します。[「動的 SQL の prepared 文のパフォーマンス・チューニング」\(135 ページ\)](#) を参照してください。
- バッチ更新を使用すると、ネットワーク・トラフィックが減少し、パフォーマンスが改善されます。具体的には、すべてのクエリが 1 つのグループとしてサーバに送信され、クライアントに返されるすべての応答が 1 つのグループとして送信されます。[「バッチ更新のサポート」\(56 ページ\)](#) を参照してください。
- セッションで、image データ、大量のロー・セット、長い text データを転送する可能性がある場合は、`PACKETSIZE` 接続プロパティを使用して可能な最大の packetsize を設定します。
- TDS-tunneled HTTP の場合は、最大 TDS packetsize を設定します。また、Web サーバが HTTP1.1 Keep-Alive 機能をサポートするように設定します。また、`SkipDoneProc` サブレット引数を “true” に設定してください。
- `LANGUAGE_CURSOR` 接続プロパティのデフォルト設定であるプロトコル・カーソルを使用します。詳細については、[「LANGUAGE_CURSOR 接続プロパティ」\(142 ページ\)](#) を参照してください。

- TYPE_SCROLL_INSENSITIVE の結果セットは、結果セットがあまり大きくない場合のみに使用してください。詳細については、「[jConnect での TYPE_SCROLL_INSENSITIVE 結果セットの使用](#)」(52 ページ)を参照してください。

以降の各項では、パフォーマンスを改善するためのその他の考慮事項について説明します。

BigDecimal の位取り変更

JDBC 1.0 仕様では、`getBigDecimal` には位取り係数が必須です。この場合、`BigDecimal` オブジェクトがサーバから返されるときに、`getBigDecimal` でのオリジナルの位取り係数を使用して、オブジェクトの位取りを変更する必要があります。

この位取りの変更に必要な時間を短縮するには、JDBC 2.0 の `getBigDecimal` メソッドを使用します。これは `jConnect` が `SybResultSet` クラスに実装するもので、`scale` 値を必要としません。

```
public BigDecimal getBigDecimal(int columnIndex)
    throws SQLException
```

次に例を示します。

```
SybResultSet rs =
    (SybResultSet)stmt.executeQuery("SELECT
        numeric_column from T1");
while (rs.next())
{
    BigDecimal bd rs.getBigDecimal(
        "numeric_column");
    ...
}
```

REPEAT_READ 接続プロパティ

`REPEAT_READ` 接続プロパティを “false” に設定することによって、データベースから結果セットを取り出すときのパフォーマンスを改善できます。ただし、`REPEAT_READ` を “false” にするときは、次のことに注意してください。

- カラム・インデックスに従って、カラム値を順番どおりに読み込まなければなりません。カラム番号ではなく名前でカラムにアクセスする場合は、この方法は困難です。
- 特定のローの特定のカラムの値を、2 回以上読み込むことはできません。

SunloConverter 文字セット変換

マルチバイト文字セットを使用するときのドライバのパフォーマンスを改善するには、jConnect のサンプルに含まれている **SunloConverter** クラスを使用できます。このコンバータは、Sun Microsystems, Inc. の Java Software Division から提供されている **sun.io** クラスに基づいています。

SunloConverter クラスは文字セット・コンバータ機能の pure Java 実装ではないので、標準の jConnect 製品には組み込まれていませんが、jConnect ドライバとともに使用して文字セット変換のパフォーマンスを改善できるように、参考のために提供されています。

注意 Sybase によるテストでは、テスト対象のすべての VM において、**SunloConverter** クラスによるパフォーマンス改善がみられました。ただし、Sun Microsystems, Inc. の Java Software Division は、JDK の今後のリリースで **sun.io** クラスを削除または変更する権利を保持しています。したがって、この **SunloConverter** クラスは、以降の JDK リリースとは互換性がなくなる可能性があります。

SunloConverter クラスを使用するには、jConnect サンプル・アプリケーションをインストールしてください。サンプルをインストールした後で、jConnect インストール・ディレクトリの *sample2* サブディレクトリにある **SunloConverter** クラスを参照するように `CHARSET_CONVERTER_CLASS` 接続プロパティを設定します。サンプル・アプリケーションを含む、jConnect とそのすべてのコンポーネントをインストールする方法については、『jConnect for JDBC インストール・ガイド』を参照してください。

デフォルトの文字セットが `iso_1` に設定されているデータベースを使用する場合や、ASCII の先頭 7 ビットのみを使用する場合は、**TruncationConverter** を使用することによってパフォーマンスを大幅に改善できます。[「jConnect 文字セット・コンバータ」\(31 ページ\)](#)を参照してください。

動的 SQL の prepared 文のパフォーマンス・チューニング

Embedded SQL™ では、動的文とは、静的にではなく実行時にコンパイルする必要がある SQL 文です。一般に、動的文には入力パラメータが含まれていますが、このことは必須ではありません。SQL では、**prepare** コマンドを使用して動的文をプリコンパイルし、保存しておくことによって、セッション中に再コンパイルすることなくその文を繰り返し実行できます。

文が同じセッション中に何度も使用される場合は、その文をプリコンパイルすると、使用のたびにデータベースに送信してコンパイルするよりもパフォーマンスは向上します。文が複雑であればあるほど、パフォーマンスの利点は大きくなります。

文が数回しか使用されない場合は、プリコンパイルは効率的ではないことがあります。これは、プリコンパイルして保存し、後でデータベース内での割り付けを解除する処理はオーバーヘッドを伴うためです。

実行する動的 SQL 文をプリコンパイルしてメモリ内に保存する処理は、時間もリソースも消費します。1つのセッションでその文が2回以上使用される可能性が低い場合は、データベースに対して **prepare** を実行するコストの方が、得られる利点を上回ってしまいます。また、データベース内で前もって処理された動的 SQL 文は、ストアド・プロシージャと同様と考えられます。場合によっては、アプリケーションで prepared 文を定義するのではなく、ストアド・プロシージャを作成してサーバに常駐させた方が良い場合があります。これについては、「[prepared 文かストアド・プロシージャかの選択](#)」(136 ページ)を参照してください。

jConnect を使用するときは、次のように Sybase データベースでの動的 SQL 文のパフォーマンスを最適化できます。

- 文が1つのセッション中に何度も実行されると考えられる場合は、プリコンパイルされた文を **PreparedStatement** オブジェクトに格納します。
- 文が1つのセッション中に数回しか実行されない場合は、コンパイルされていない SQL 文を **PreparedStatement** オブジェクトに格納します。

以降の項で説明するように、どのように **DYNAMIC_PREPARE** 接続プロパティを設定して **PreparedStatement** オブジェクトを作成するのが最適であるかは、アプリケーションを別の JDBC ドライバに移植できるようにする必要があるかどうか、または作成するアプリケーションで jConnect 固有の JDBC 拡張機能を使用できるようにするかどうかによって決まります。

jConnect には、動的 SQL 文に対するパフォーマンス・チューニング機能があります。

prepared 文かストアド・プロシージャかの選択

プリコンパイルされた動的 SQL 文が格納された **PreparedStatement** オブジェクトを作成する場合は、データベース内でコンパイルされた文は事実上ストアド・プロシージャとなり、メモリ内に保持されて、セッションに対応するデータ構造体に追加されます。データベース内にストアド・プロシージャを保持するか、アプリケーション内で **PreparedStatement** オブジェクトを作成してコンパイル済みの SQL 文を格納するかを決定するには、リソース要件およびデータベースとアプリケーションの管理を考慮することが重要です。

- コンパイルされたストアド・プロシージャは、すべての接続にわたってグローバルに使用できます。これとは対照的に、**PreparedStatement** オブジェクト内の動的 SQL 文は、この文を使用するセッションごとにコンパイルと割り付け解除を行う必要があります。

- アプリケーションが複数のデータベースにアクセスする場合に、ストアード・プロシージャを使用するということは、すべてのターゲット・データベース上に同じストアード・プロシージャを用意する必要があるということです。これは、データベース管理上の問題となることがあります。動的 SQL 文に対して **PreparedStatement** オブジェクトを使用すると、この問題を回避できます。
- アプリケーションで **CallableStatement** オブジェクトを作成してストアード・プロシージャを呼び出すようにすれば、SQL コードとテーブル参照をストアード・プロシージャにカプセル化できます。この場合は、アプリケーションを変更することなく、基本のデータベースや SQL コードを変更できます。

移植可能なアプリケーションでの prepared 文

さまざまなベンダのデータベース上でアプリケーションを実行するときに、一部の **PreparedStatement** オブジェクトにはプリコンパイルされた文を格納し、その他のオブジェクトにはコンパイルされていない文を格納する場合は、次の手順に従ってください。

- Sybase データベースにアクセスするときは、**DYNAMIC_PREPARE** 接続プロパティを必ず “true” に設定してください。
- プリコンパイルされた文が格納された **PreparedStatement** オブジェクトを返すには、通常どおりに **Connection.prepareStatement** を使用します。

```
PreparedStatement ps_precomp =
    Connection.prepareStatement(sql_string);
```

- コンパイルされていない文が格納された **PreparedStatement** オブジェクトを返すには、**Connection.prepareCall** を使用します。

Connection.prepareCall は **CallableStatement** オブジェクトを返しますが、**CallableStatement** は **PreparedStatement** のサブクラスであるため、次のように **CallableStatement** オブジェクトを **PreparedStatement** オブジェクトにアップキャストすることができます。

```
PreparedStatement ps_uncomp =
    Connection.prepareCall(sql_string);
```

プリコンパイルされた文が格納された **PreparedStatement** オブジェクトを返すように実装されているのは **Connection.prepareStatement** だけなので、**PreparedStatement** オブジェクト *ps_uncomp* に格納されるのはコンパイルされていない文であることが保証されます。

prepared 文と jConnect の拡張機能

ドライバ間の移植性が問題とならない場合は、アプリケーションで `SybConnection.prepareStatement` を使用して、プリコンパイルされた文とコンパイルされていない文のどちらを `PreparedStatement` オブジェクトに格納するかを指定できます。この場合に、prepared 文をどのようにコーディングするかは、アプリケーション内の動的文の大半がセッション中に何度も実行されるのか、数回しか実行されないのかによって決まります。

動的文の大半が数回しか実行されない場合

アプリケーションの動的 SQL 文の大半が、1 回のセッションで 1 ～ 2 回しか実行されないものである場合は、次の手順に従ってください。

- `DYNAMIC_PREPARE` 接続プロパティを“false”に設定します。
- コンパイルされていない文が格納された `PreparedStatement` オブジェクトを返すには、通常どおりに `Connection.prepareStatement` を使用します。

```
PreparedStatement ps_uncomp =  
    Connection.prepareStatement(sql_string);
```

- プリコンパイルされた文が格納された `PreparedStatement` オブジェクトを返すには、次のように `dynamic` を“true”に設定して `SybConnection.prepareStatement` を使用します。

```
PreparedStatement ps_precomp =  
    (SybConnection)conn.prepareStatement(sql_string, true);
```

動的文の大半がセッション中に何度も実行される場合

アプリケーションの動的文の大半が、1 つのセッション中に何度も実行されるものである場合は、次の手順に従ってください。

- `DYNAMIC_PREPARE` 接続プロパティを“true”に設定します。
- プリコンパイルされた文が格納された `PreparedStatement` オブジェクトを返すには、通常どおりに `Connection.prepareStatement` を使用します。

```
PreparedStatement ps_precomp = Connection.prepareStatement(sql_string);
```

- コンパイルされていない文が格納された `PreparedStatement` オブジェクトを返すには、`Connection.prepareCall` (「[移植可能なアプリケーションでの prepared 文](#)」を参照) を使用するか、`dynamic` を“false”に設定して `SybConnection.prepareStatement` を使用します。

```
PreparedStatement ps_uncomp =  
    (SybConnection)conn.prepareStatement(sql_string, false);
```

```
PreparedStatement ps_uncomp = Connection.prepareCall(sql_string);
```

Connection.prepareStatement

jConnect は `Connection.prepareStatement` を実装しているので、プリコンパイルされた SQL 文を `PreparedStatement` オブジェクトに返すことも、コンパイルされていない SQL 文を返すこともできます。プリコンパイルされた SQL 文を `PreparedStatement` オブジェクトに返すように `Connection.prepareStatement` を設定すると、`prepare` コマンドを直接実行したときとまったく同じように、動的 SQL 文がデータベースに送信され、プリコンパイルされて保存されます。コンパイルされていない SQL 文を返すように `Connection.prepareStatement` を設定すると、文はデータベースに送信されずに `PreparedStatement` オブジェクトに返されます。

`Connection.prepareStatement` が返す SQL 文のタイプは接続プロパティ `DYNAMIC_PREPARE` によって決定され、そのセッション全体に適用されます。

Sybase 専用のアプリケーション向けに、jConnect 6.05 以降では jConnect `SybConnection` クラスの下に `prepareStatement` メソッドが用意されています。 `SybConnection.prepareStatement` を使用すると、`DYNAMIC_PREPARE` 接続プロパティによるセッション・レベルの設定に関係なく、個々の動的 SQL 文をプリコンパイルするかどうかを指定できます。

DYNAMIC_PREPARE 接続プロパティ

`DYNAMIC_PREPARE` は、動的 SQL prepared 文を有効にするためのブール値の接続プロパティです。

- `DYNAMIC_PREPARE` が “true” に設定されているとき、セッション内で呼び出された `Connection.prepareStatement` は、プリコンパイルされた文を `PreparedStatement` オブジェクトに返そうとします。

この場合、`PreparedStatement` が実行されるときは、このオブジェクトに格納された文はすでにデータベースでプリコンパイルされており、動的に値を割り当てるためのプレースホルダがあるので、文の実行だけが必要です。

- 接続に対して `DYNAMIC_PREPARE` が “false” に設定されているときは、`Connection.prepareStatement` によって返される `PreparedStatement` オブジェクトにはプリコンパイルされた文は格納されていません。

この場合、`PreparedStatement` が実行されるたびに、このオブジェクトに格納されている SQL 文をデータベースに送信してコンパイルと実行の両方を行う必要があります。

`DYNAMIC_PREPARE` のデフォルト値は “false” です。

次の例では、動的 SQL 文のプリコンパイルを有効にするために `DYNAMIC_PREPARE` が “true” に設定されています。この例では、`props` は接続プロパティを指定するための `Properties` オブジェクトです。

```
...
props.put("DYNAMIC_PREPARE", "true");
Connection conn = DriverManager.getConnection(url, props);
```

DYNAMIC_PREPARE が “true” に設定されているときは、次のことに注意してください。

- すべての動的文を **prepare** コマンドでプリコンパイルできるわけではありません。SQL-92 標準では **prepare** コマンドで利用できる文にいくつかの制約が設けられています。また、個々のデータベース・ベンダが独自の制約を設けている場合もあります。
- **Connection.prepareStatement** を介してデータベースに送信された文をプリコンパイルして保存できないというデータベースのエラーが生成された場合は、jConnect はこのエラーをトラップして、コンパイルされていない SQL 文を格納した **PreparedStatement** オブジェクトを返します。**PreparedStatement** オブジェクトが実行されるたびに、文はデータベースに再送信され、コンパイルされて実行されます。
- プリコンパイルされた文はデータベースのメモリ内に常駐して、セッションの終わりまで、または **PreparedStatement** オブジェクトが明示的にクローズされるまで存続します。**PreparedStatement** オブジェクトに対してガーベジ・コレクションが行われても、データベースから prepared 文が削除されることはありません。

原則として、個々の **PreparedStatement** オブジェクトを最後に使用した後に明示的にクローズしてください。これは、セッション中にサーバのメモリに prepared 文が累積してパフォーマンスを低下させることを防ぐためです。

SybConnection.prepareStatement

アプリケーションで jConnect 固有の JDBC 拡張機能を使用できる場合は、**SybConnection.prepareStatement** 拡張メソッドを使用して動的 SQL 文を **PreparedStatement** オブジェクトに返すことができます。

```
PreparedStatement SybConnection.prepareStatement(String sql_stmt,  
boolean dynamic) throws SQLException
```

SybConnection.prepareStatement は、プリコンパイルされた SQL 文またはコンパイルされていない SQL 文を **PreparedStatement** オブジェクトに格納して返します。どちらを返すかは、*dynamic* パラメータの設定によって決まります。*dynamic* が “true” に設定されている場合は、**SybConnection.prepareStatement** が返す **PreparedStatement** オブジェクトには、プリコンパイルされた SQL 文が格納されています。*dynamic* が “false” に設定されている場合は、返される **PreparedStatement** オブジェクトにはコンパイルされていない SQL 文が格納されています。

次の例では、**SybConnection.prepareStatement** を使用して、プリコンパイルされた文が格納された **PreparedStatement** オブジェクトを返す方法を示します。

```
PreparedStatement precomp_stmt = ((SybConnection) conn).prepareStatement  
("SELECT * FROM authors WHERE au_fname LIKE ?", true);
```


この例では、`SybConnection.prepareStatement` を使用できるように、接続オブジェクト `conn` が `SybConnection` オブジェクトにキャストされています。`SybConnection.prepareStatement` に渡された SQL 文字列は、`DYNAMIC_PREPARE` 接続プロパティが “false” に設定されていてもデータベース内でプリコンパイルされます。

`SybConnection.prepareStatement` を介してデータベースに送信された文がプリコンパイルできないというデータベースのエラーが生成された場合は、`jConnect` によって `SQLException` がスローされ、`PreparedStatement` オブジェクトは返されません。これは、エラー発生時に SQL エラーをトラップして、コンパイルされていない文を格納した `PreparedStatement` オブジェクトを返す `Connection.prepareStatement` とは異なります。

ESCAPE_PROCESSING_DEFAULT 接続プロパティ

デフォルトでは、`jConnect` はデータベースに送信されるすべての SQL 文を解析して、有効な JDBC 関数エスケープがあるかどうかを調べます。アプリケーションの SQL 呼び出しの中で JDBC 関数エスケープを使用しない場合は、この接続プロパティを “false” に設定すると、この解析を迂回できます。これにより、パフォーマンスが若干向上する可能性があります。

カーソルのパフォーマンス

`SybCursorResultSet` クラスの `Statement.setCursorName` メソッドまたは `setFetchSize()` メソッドが実行されると、`jConnect` はデータベース内にカーソルを作成します。他のメソッドの場合は、`jConnect` はカーソルのオープン、フェッチ、更新を行います。

`jConnect` では、カーソルを作成して操作するには、SQL 文をデータベースに送信する方法と、カーソル・コマンドを TDS 通信プロトコルのトークンとしてコード化する方法があります。前者のタイプのカーソルが「言語カーソル」で、後者のタイプのカーソルが「プロトコル・カーソル」です。

プロトコル・カーソルの方が、言語カーソルよりもパフォーマンスが優れています。さらに、必ずしもすべてのデータベースが言語カーソルをサポートしているわけではありません。たとえば、SQL Anywhere データベースは言語カーソルをサポートしていません。

デフォルトでは、`jConnect` のカーソルはすべてプロトコル・カーソルです。ただし、`LANGUAGE_CURSOR` 接続プロパティを設定することによって、言語コマンドを使用してデータベース内にカーソルを作成して操作することを選択できます。

LANGUAGE_CURSOR 接続プロパティ

LANGUAGE_CURSOR は、プロトコル・カーソルと言語カーソルのどちらのカーソルを作成するかを決定する、jConnect のブール値の接続プロパティです。

- LANGUAGE_CURSOR が “false” に設定されている場合は、セッション中に作成されるカーソルはすべてパフォーマンスに優れたプロトコル・カーソルとなります。jConnect は、カーソル・コマンドを TDS プロトコルのトークンとして送信することによって、カーソルを作成および操作します。

デフォルトでは、LANGUAGE_CURSOR は “false” に設定されています。

- LANGUAGE_CURSOR が “true” に設定されている場合は、セッション中に作成されるカーソルはすべて言語カーソルとなります。jConnect は、SQL 文をデータベースに送信して解析およびコンパイルすることによって、カーソルを作成および操作します。

LANGUAGE_CURSOR を “true” に設定することの明確な利点はありませんが、LANGUAGE_CURSOR を “false” に設定したときにアプリケーションが予期しない動作をした場合に備えて、このオプションが用意されています。

jConnect アプリケーションへのマイグレート

この章では、アプリケーションを jConnect 5.x または 6.x から jConnect 7.x にマイグレートする方法について説明します。

トピック名	ページ
jConnect 7.x へのアプリケーションのマイグレート	143
Sybase 拡張機能の変更	144

jConnect 7.x へのアプリケーションのマイグレート

jConnect 7.x にアップグレードする手順は次のとおりです。

❖ jConnect 7.0 へのマイグレート

- 1 コード内で Sybase 拡張機能を使用している場合や jConnect クラスを明示的にインポートしている場合は、必要に応じてパッケージのインポート文を変更します。

たとえば、次のインポート文

```
import com.sybase.jdbc.*
```

および

```
import com.sybase.jdbc2.jdbc.*
```

を次のように変更します。

```
import com.sybase.jdbcx.*
```

Sybase 拡張機能 API の使用方法については、「[Sybase 拡張機能の変更 \(144 ページ\)](#)」を参照してください。

- 2 JDBC_HOME を、インストールした jConnect ドライバの最上位のディレクトリに設定します。

```
JDBC_HOME=jConnect-7_0
```

- 3 新しいインストール環境を反映するように、CLASSPATH 環境変数を変更します。CLASSPATH に次のパスが含まれている必要があります。

```
JDBC_HOME/classes/jconn4.jar
```

- 4 アプリケーションで新しいドライバが使用されるようにするために、ソース・コードの中でドライバをロードする部分を変更し、再コンパイルします。

```
Class.forName("com.sybase.jdbc4.jdbc.SybDriver");
```

- 5 jConnect 7.0 ドライバが、CLASSPATH 環境変数で指定されている最初の jConnect ドライバであることを確認します。

Sybase 拡張機能の変更

jConnect バージョン 4.1 以降には、JDBC への Sybase 拡張機能のすべてが含まれているパッケージ **com.sybase.jdbcx** が付属しています。4.1 より前のバージョンの jConnect では、これらの拡張機能は **com.sybase.jdbc** パッケージと **com.sybase.utils** パッケージに含まれていました。

com.sybase.jdbcx パッケージは、jConnect のさまざまなバージョンに対してインタフェースを統一します。Sybase の拡張機能はすべて Java インタフェースとして定義されているので、これらのインタフェースを使用して構築されたアプリケーションに何も影響を与えずに、基本となる実装を変更できます。

新しく開発するアプリケーションで Sybase 拡張機能を使用するときは、**com.sybase.jdbcx** を使用してください。このパッケージ内のインタフェースを使用すれば、バージョン 4.0 以降の jConnect にアップグレードするときに、アプリケーションの変更を最小限にすることができます。

Sybase 拡張機能の一部は、**com.sybase.jdbcx** インタフェースを取り入れるために変更されました。

拡張機能の変更例

アプリケーションで、たとえば **SybMessageHandler** を使用している場合は、コードの変更は次のようになります。

- **jConnect 4.0** のコード：

```
import com.sybase.jdbc.SybConnection;
import com.sybase.jdbc.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setMessageHandler(new ConnectionMsgHandler());
```

- **jConnect 6.0** のコード：

```
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybMessageHandler;
.
```

```

Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setSybMessageHandler(new ConnectionMsgHandler());

```

Sybase の拡張機能の使用方法的例については、jConnect 付属のサンプルを参照してください。

メソッド名

表 6-1 に、新しいインタフェースでのメソッド名の変更を示します。

表 6-1: メソッド名の変更

クラス	古い名前	新しい名前
SybConnection	getCapture()	createCapture()
SybConnection	setMessageHandler()	setSybMessageHandler()
SybConnection	getMessageHandler()	getSybMessageHandler()
SybStatement	setMessageHandler()	setSybMessageHandler()
SybStatement	getMessageHandler()	getSybMessageHandler()

Debug クラス

Debug クラスへの直接の静的参照はサポートされなくなりましたが、`com.sybase.utils` パッケージには非推奨の形で残っています。jConnect のデバッグ機能を使用するには、`SybDriver` クラスの `getDebug` メソッドを使用して `Debug` クラスへの参照を取得してください。例：

```

import com.sybase.jdbcx.SybDriver;
import com.sybase.jdbcx.Debug;
.
.
.
SybDriver sybDriver =
    (SybDriver)Class.forName
        ("com.sybase.jdbc4.jdbc.SybDriver").newInstance();
Debug sybDebug = sybDriver.getDebug();
sybDebug.debug(true, "ALL", System.out);

```

Sybase 拡張機能のリストについては、jConnect インストール・ディレクトリの `docs/` ディレクトリにある jConnect の javadoc マニュアルを参照してください。

この章では、Web サーバ・ゲートウェイについて説明し、さらに jConnect での使い方について説明します。

トピック名	ページ
Web サーバ・ゲートウェイの概要	147
使用上の条件	152
TDS トンネリング・サーブレットの使用方法	154

Web サーバ・ゲートウェイの概要

データベース・サーバが Web サーバとは別のホストで稼働している場合、または開発するインターネット・アプリケーションでファイアウォールを通してセキュア・データベース・サーバに接続する必要がある場合は、プロキシとしての役割を持ち、データベース・サーバへのパスとなるゲートウェイが必要になります。

jConnect には、Secure Sockets Layer (SSL) プロトコルを使用してサーバに接続するための Java サーブレットが用意されています。このサーブレットは、`javax.servlet` インタフェースをサポートする Web サーバにインストールできます。このサーブレットにより、jConnect が、Web サーバをゲートウェイとして使用する暗号化をサポートできるようになります。

注意 jConnect は、クライアント・システムでの SSL もサポートします。詳細については、「[カスタム・ソケット・プラグインの実装](#)」(106 ページ)を参照してください。

TDS トンネリングの使用法

jConnect は、TDS を使用してデータベース・サーバと通信します。HTTP を介した TDS のトンネリングは、要求を転送する場合に便利です。ゲートウェイを通してクライアントからバックエンド・サーバに送信される要求の本体に、TDS が含まれます。要求のヘッダは、要求パケットに含まれる TDS の長さを示します。

TDS は、HTTP とは異なり、接続指向型のプロトコルです。インターネット・アプリケーションでの暗号化のようなセキュリティ機能をサポートするために、jConnect は TDS トンネリング・サーブレットを使用し、HTTP 要求間での論理コネクションを管理します。このサーブレットは、最初のログイン要求のときにセッション ID を生成します。セッション ID は、次の要求のヘッダに組み込まれます。セッション ID を使用することによって、アクティブなセッションを識別できます。また、このように特定のセッション ID を使用してサーブレットが接続をオープンしている間は、セッションを再開することもできます。

TDS トンネリング・サーブレットの論理コネクション機能によって、jConnect が 2 つのシステムの間での暗号化された通信をサポートできるようになります。たとえば、jConnect クライアントで `CONNECT_PROTOCOL` 接続プロパティを “https” に設定すれば、TDS トンネリング・サーブレットを実行している Web サーバに接続できるようになります。

jConnect とゲートウェイの設定

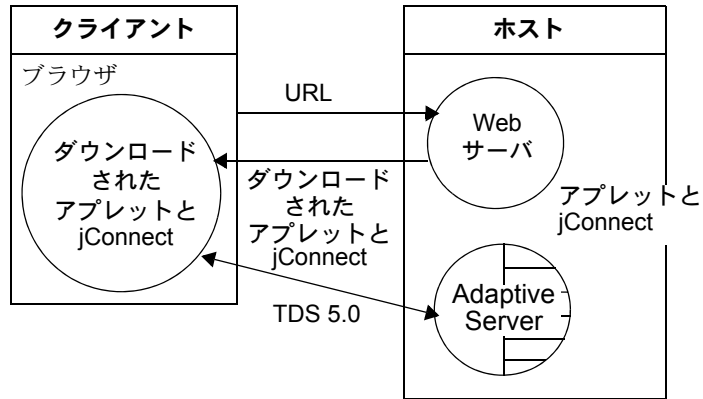
Web サーバと Adaptive Server の設定には、いくつかのオプションがあります。次の 4 つの一般的な設定例を通して、jConnect ドライバがどこにインストールされるかと、TDS トンネリング・サーブレットを実行するゲートウェイがいつ使用されるかを説明します。

Web サーバと Adaptive Server を同じホスト上に配置

この 2 層の設定では、Web サーバと Adaptive Server の両方が同じホストにインストールされます。

- jConnect は Web サーバ・ホスト上にインストールする
- ゲートウェイは必要ない

図 7-1: Web サーバと Adaptive Server を同じホスト上に配置

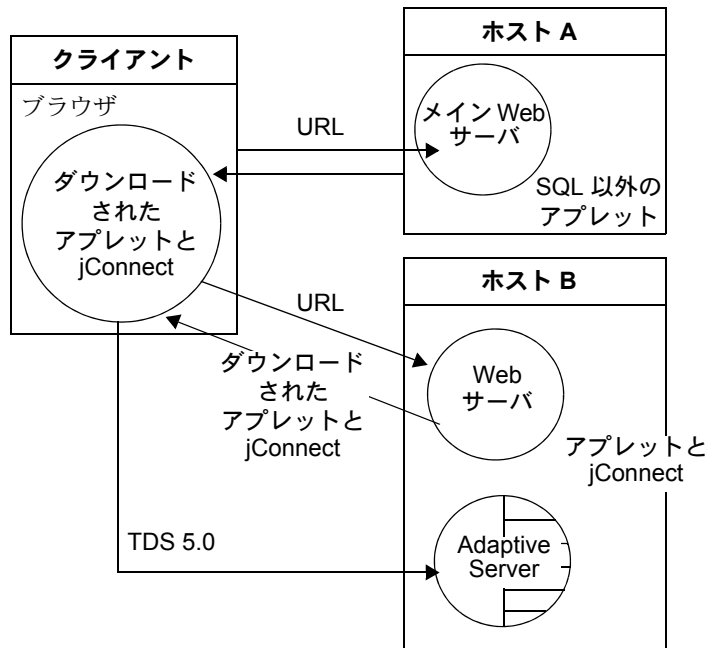


専用 JDBC Web サーバと Adaptive Server を同じホスト上に配置

この設定では、メイン Web サーバ用に別のホストを使用します。2 番目のホストは、Adaptive Server アクセス専用の Web サーバと Adaptive Server の両方に使用されます。SQL アクセスを必要とする要求は、メイン・サーバからのリンクによって専用の Web サーバに送信されます。2 番目のホストに次のようにインストールします。

- jConnect は 2 番目の (Adaptive Server) ホストにインストールする
- ゲートウェイは必要ない

図 7-2: 専用 JDBC Web サーバと Adaptive Server を同じホスト上に配置

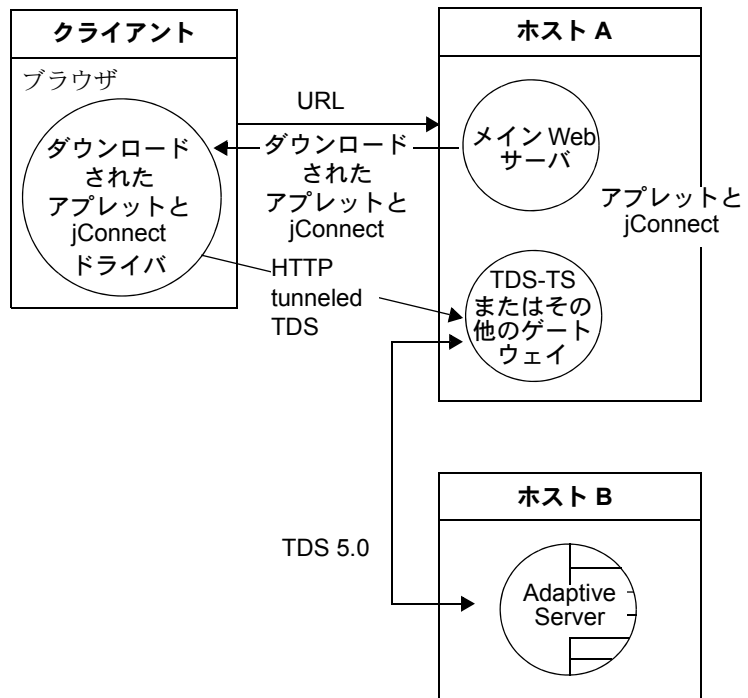


Web サーバと Adaptive Server をそれぞれ別のホストに配置

この 3 層の設定では、Adaptive Server は、Web サーバとは別のホスト上にあります。jConnect には、Adaptive Server のプロキシとしての役割を持つゲートウェイが必要です。

- jConnect は Web サーバ・ホスト上にインストールする
- TDS トネリング・サーブレットまたは別のゲートウェイをインストールする

図 7-3: Web サーバと Adaptive Server をそれぞれ別のホストに配置

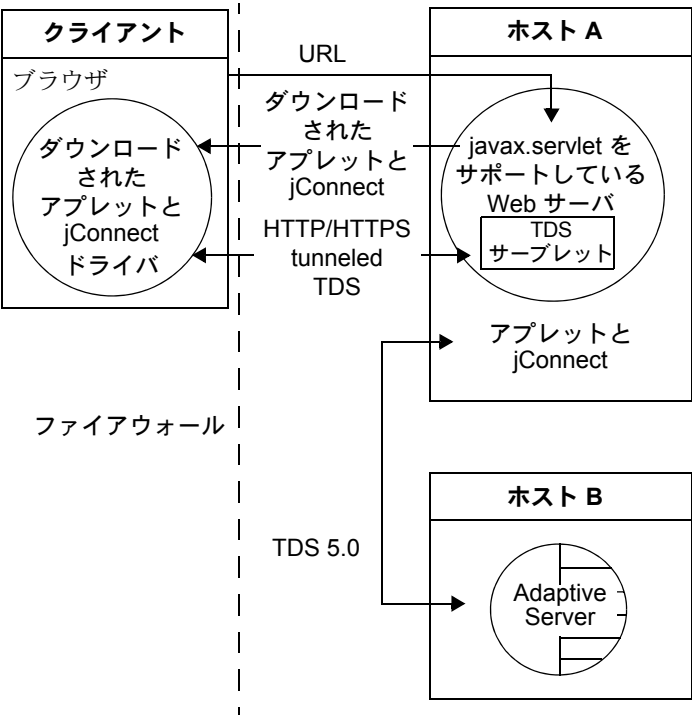


ファイアウォールを介したサーバへの接続

ファイアウォールで保護されているサーバに接続するには、データベース要求と応答をインターネット上で転送できるようにするための TDS トンネリング・サブレットを Web サーバ上で使用する必要があります。

- jConnect は Web サーバ・ホスト上にインストールする
- javax.servlet インタフェースをサポートする Web サーバが必要

図 7-4: ファイアウォールを介したサーバへの接続



使用上の条件

以降の項では、Web サーバ・ゲートウェイを使用する場合の要件について説明します。

***index.html* ファイルの読み込み**

Web ブラウザを使用して、jConnect インストール・ディレクトリにある *index.html* ファイルを表示してください。*index.html* には、jConnect のマニュアルとサンプル・コードへのリンクがあります。

注意 jConnect がインストールされているマシンで Netscape を使用する場合は、ブラウザが CLASSPATH 環境変数にアクセスしないことを確認してください。詳細については、『jConnect for JDBC インストール・ガイド』の第1章の「Netscape の使用による CLASSPATH の制限」を参照してください。

❖ index.html ファイルを表示するには

- 1 Web ブラウザを開きます。
- 2 実際の設定に合わせて URL を入力します。たとえば、ブラウザと Web サーバが同じホスト上で稼働している場合は、次のように入力します。

```
http://localhost:8000/index.html
```

ブラウザと Web サーバが異なるホスト上で稼働している場合は、次のように入力します。

```
http://host:port/index.html
```

host は Web サーバが稼働しているホストの名前で、*port* は受信ポートです。

サンプル Isql アプレットの実行

index.html ファイルをブラウザにロードしたら、次を実行してください。

❖ サンプル・アプレットを実行するには

- 1 [Run Sample JDBC Applets] をクリックします。
これによって、[jConnect Sample Programs] ページが表示されます。
- 2 [Sample Programs] ページの下の方に移動して、[Executable Samples] の下の表を見つけます。
- 3 この表の“Isql.java”のローの最後にある [Run] をクリックします。

サンプル Isql.java アプレットは、サンプル・データベースに対して簡単なクエリを要求し、その結果を表示します。表示される情報は、デフォルトの Adaptive Server ホスト名、ポート番号、ユーザ名 (*guest*)、パスワード (*sybase*)、データベース、クエリです。アプレットは、デフォルト値を使用して Sybase のデモ用データベースに接続します。[Go] をクリックすると、結果が返されます。

トラブルシューティング

UNIX で、アプレットが予期したとおりに表示されない場合は、次の手順でアプレット画面のサイズを変更できます。

❖ アプレット画面のサイズを変更するには

- 1 テキスト・エディタを使用して次のファイルを編集します。

`$JDBC_HOME/sample2/gateway.html`

- 2 7行目にある、高さを指定するパラメータを、650 に変更します。他の高さも試してみてください。
- 3 ブラウザで Web ページを再ロードします。

TDS トンネリング・サーブレットの使用法

TDS トンネリング・サーブレットを使用するには、Sun Microsystems の Java Web サーバなど、`javax.servlet` インタフェースをサポートする Web サーバが必要です。Web サーバをインストールするときに、jConnect TDS トンネリング・サーブレットをアクティブ・サーブレットのリストに追加してください。サーブレットのパラメータを設定して、接続タイムアウトと最大パケット・サイズを定義することもできます。

TDS トンネリング・サーブレットを使用するときは、クライアントからゲートウェイを通してバックエンド・サーバに送信される要求に、GET または POST コマンド、TDS セッション ID (最初の要求の後)、バックエンド・アドレス、および要求のステータスが含まれています。

TDS は、要求の本体内にあります。ヘッダには、TDS ストリームの長さ、ゲートウェイによって割り当てられたセッション ID を示す 2 つのフィールドがあります。

クライアントが要求を送信するとき、Content-Length ヘッダ・フィールドは TDS コンテンツのサイズを表し、要求コマンドは POST となります。クライアントがサーバからの応答データの次の部分を取り出そうとしている、または接続をクローズしようとしているので要求内に TDS データがない場合は、要求コマンドは GET です。

次の例では、TDS-tunneled HTTPS プロトコルを使用するクライアントと HTTPS ゲートウェイの間でどのように情報が渡されるかを示します。この例では、“DBSERVER” という名前のバックエンド・サーバのポート番号 “1234” に接続します。

表 7-1: クライアントからゲートウェイへのログイン要求。セッション ID なし。

クエリ	POST/tds?ServerHost=dbserver&ServerPort=1234& Operation=more HTTP/1.0
ヘッダ	Content-Length: 605
コンテンツ (TDS)	ログイン要求

表 7-2: ゲートウェイからクライアント。ヘッダには TDS サブレットによって 割り当てられたセッション ID が含まれる。

クエリ	200 SUCCESS HTTP/1.0
ヘッダ	Content-Length: 210 TDS-Session:TDS00245817298274292
コンテンツ (TDS)	ログイン確認 EED

表 7-3: クライアントからゲートウェイ。後続のすべての要求のヘッダにはセッション ID が含まれる。

クエリ	POST/tds?TDS-Session=TDS00245817298274292&Operation=more HTTP/1.0
ヘッダ	Content-Length: 32
コンテンツ (TDS)	Query “SELECT * from authors”

表 7-4: ゲートウェイからクライアント。後続のすべての応答のヘッダにはセッション ID が含まれる。

クエリ	200 SUCCESS HTTP/1.0
ヘッダ	Content-Length: 2048 TDS-Session:TDS00245817298274292
コンテンツ (TDS)	ローのフォーマットおよびクエリ応答からのいくつかのロー

要件の確認

TDS-tunneled HTTP を行うための jConnect サブレットを使用するには、次のものがが必要です。

- javax.servlet インタフェースをサポートする Web サーバ。サーバをインストールするには、サーバ付属のマニュアルの指示に従ってください。

サブレットのインストール方法

jConnect のインストール環境の *classes* ディレクトリの下に *gateway2* サブディレクトリがあります。このサブディレクトリには、TDS トンネリング・サブレットに必要なファイルがあります。

この jConnect gateway パッケージを、Web サーバの *servlets* ディレクトリの *gateway2* サブディレクトリにコピーしてください。サーブレットをコピーしたら、Web サーバの指示に従ってサーブレットをアクティブにしてください。

サーブレット引数の設定

サーブレットを Web サーバに追加するときに、オプションの引数を指定してパフォーマンスをカスタマイズできます。

- *SkipDoneProc* [true/false] – Sybase データベースは、クエリの実行中に中間処理手順を行っているときにロー・カウント情報を返すことがあります。通常、クライアント・アプリケーションはこのデータを無視します。*SkipDoneProc* を “true” に設定すると、サーブレットはこの余分な情報を応答から取り除き、ネットワーク使用量とクライアントでの処理要件を軽減します。これは、不要なデータが、暗号化や暗号化解除を行うことなく無視されるので、HTTPS/SSL を使用しているときは特に便利です。
- *TdsResponseSize* – Tunneled HTTPS の最大 TDS パケット・サイズを設定します。*TdsResponseSize* に大きな値を指定することによって効率が向上するのは、ユーザ数が少なく、大量のデータを扱う場合です。多数のユーザが小さなトランザクションを実行する場合は、*TdsResponseSize* の値を小さくします。
- *TdsSessionIdleTimeout* – サーバ接続のアイドル状態が維持される時間をミリ秒単位で定義します。この時間に達すると、接続は自動的にクローズされます。デフォルトの *TdsSessionIdleTimeout* は 600,000 (10 分) です。

対話型クライアント・プログラムで長時間のアイドル状態が発生する可能性がある場合に、接続が切断されないようにするには、*TdsSessionIdleTimeout* の値を大きくします。

SESSION_TIMEOUT 接続プロパティを使用して、jConnect クライアントから接続タイムアウト値を設定することもできます。これは、特定のアプリケーションで長時間のアイドル状態が発生する可能性がある場合に便利です。この場合は、サーブレットのタイムアウトを設定するのではなく、SESSION_TIMEOUT 接続プロパティを使用して接続のタイムアウトを長い値に設定します。

- *Debug* – デバッグ機能をオンにします。[「jConnect でのデバッグ」\(123 ページ\)](#) を参照してください。

サーブレット引数は、カンマで区切った文字列として入力してください。例：

```
TdsResponseSize=[size],TdsSessionIdleTimeout=[timeout],Debug=true
```

サーブレット引数を入力する方法については、Web サーバのマニュアルを参照してください。

サーブレットの呼び出し

TDS トンネリング・サーブレットがインストールされているゲートウェイを jConnect がいつ使用するかは、*proxy* 接続プロパティのパス拡張部分に基づいて決定します。jConnect は *proxy* のサーブレット・パス拡張部分を認識して、指定のゲートウェイ上にあるサーブレットを呼び出します。

次のフォーマットを使用して接続 URL を定義してください。

```
http://host:port/TDS-servlet-path
```

jConnect は Web サーバ上で TDS トンネリング・サーブレットを呼び出して、HTTP を介して TDS をトンネリングします。サーブレットのパスは、Web サーバのサーブレット・エイリアス・リストで定義したパスでなければなりません。

アクティブな TDS セッションのトラッキング

アクティブな TDS セッションに関する、サーバ接続などの情報を参照できます。Web ブラウザを使用して、次の管理用 URL を開きます。

```
http://host:port/TDS-servlet-path?Operation=list
```

たとえば、サーバが“myserver”で、TDS サーブレットのパスが */tds* ならば、次のように入力します。

```
http://myserver:8080/tds?Operation=list
```

アクティブな TDS セッションのリストが表示されます。セッションをクリックすると、サーバ接続などの情報を参照できます。

TDS セッションの終了

前述の URL を使用して、アクティブな TDS セッションを終了することができます。最初のページにあるセッションのリストからアクティブなセッションをクリックし、[Terminate This Session] をクリックします。

TDS セッションの再開

オープンしている既存の接続を必要に応じて再開できるように、SESSION_ID 接続プロパティを設定できます。SESSION_ID が指定されると、jConnect はプロトコルのログイン・フレーズをスキップし、指定されたセッション ID を使用してゲートウェイとの接続を再開します。セッション ID がサーブレット上に存在しない場合は、ユーザがその接続を使用しようとしたときに SQL 例外が発生します。

SQL の例外メッセージと警告メッセージ

次の表は、jConnect を使用しているときに表示される可能性のある SQL の例外メッセージと警告メッセージのリストです。

SQL ステータス	メッセージ／説明／対処方法
010AF	<p>重大な警告：指定が失敗しました。devclass を使用して、この重大なバグのソースを判断してください。メッセージ = _____</p> <p>説明：jConnect ドライバ内部のアサーション (指定) に失敗しました。</p> <p>対処方法：devclasses デバッグ・クラスを使用してこのメッセージの原因を調べ、Sybase 製品の保守契約を結んでいるサポート・センタに問題を報告してください。</p>
010DF	<p>Attempt to set database at login failed. Error message: _____.</p> <p>説明：jConnect は、接続 URL で指定されているデータベースに接続できません。</p> <p>対処方法：URL 内のデータベース名が正しいことを確認してください。また、SQL Anywhere に接続する場合は、SERVICENAME 接続プロパティを使用してデータベースを指定してください。</p>
010DP	<p>Duplicate connection property _____ ignored.</p> <p>説明：接続プロパティが二度定義されました。ドライバ接続プロパティ・リスト内で、大文字と小文字の指定を変えて二度定義されている可能性があります (たとえば、“password” と “PASSWORD”)。接続プロパティ名では大文字と小文字が区別されないため、jConnect は、大文字と小文字の指定が異なるだけで名前が同じであるプロパティどうしを区別することはできません。</p> <p>または、接続プロパティ・リスト内と URL 内の両方で定義されている可能性があります。このような場合には、接続プロパティ・リストの定義が優先されます。</p> <p>対処方法：アプリケーションで、接続プロパティを一度だけ定義するようにしてください。ただし、アプリケーションによっては、URL よりプロパティ・リストで定義された接続プロパティが優先されるという利点を利用するものもあります。このような場合には、この警告を無視してください。</p>
010HA	<p>サーバは、高可用性機能を使用する要求を拒否しました。データベースを再設定するか、高可用性セッションの要求をやめてください。</p> <p>説明：高可用性 (HA) 機能を使用する接続を試行しましたが、接続先サーバがこの接続を許可しませんでした。</p> <p>対処方法：高可用性フェールオーバをサポートするようにサーバを再設定します。または、REQUEST_HA_SESSION を “true” に設定しないようにしてください。</p>
010HD	<p>Sybase high-availability failover is not supported by this type of database server.</p> <p>説明：jConnect が接続しようとしたデータベースは、高可用性フェールオーバをサポートしていません。</p> <p>対処方法：高可用性フェールオーバをサポートするデータベース・サーバにのみ接続してください。</p>

SQL ステータス	メッセージ／説明／対処方法
010HN	<p>The client did not specify a SERVICE_PRINCIPAL_NAME Connection property. Therefore, jConnect is using the hostname of _____ as the service principal name</p> <p>対処方法:接続プロパティを使用して、サービスのプリンシパル名を明示的に指定してください。</p>
010HT	<p>Hostname property truncated, maximum length is 30.</p> <p>説明: HOSTNAME 接続プロパティに指定された文字列の長さが 30 文字を超えています。または、jConnect アプリケーションが実行されているホスト・マシンの名前の長さが 30 バイトを超えています。</p> <p>対処方法: 特に対処は必要ありません。これは、名前が 30 バイトにトランケートされることを示す jConnect の警告です。この警告が発生しないようにするには、HOSTNAME に 30 文字以下の長さの文字列を設定してください。</p>
010KF	<p>The server rejected your Kerberos login attempt. Most likely, this was because of a Generic Security Services (GSS) exception. Please check your Kerberos environment and configuration.</p> <p>対処方法: Kerberos 環境をチェックして、ユーザが KDC に対して正しく認証されることを確認してください。詳細については、「第 3 章 セキュリティ」を参照してください。</p>
010MX	<p>Metadata accessor information was not found on this database. Please install the required tables as mentioned in the jConnect documentation. Error encountered while attempting to retrieve metadata information:</p> <p>_____</p> <p>説明: メタデータ情報を返すのに必要なストアド・プロシージャがサーバ上に存在しない可能性があります。</p> <p>対処方法: メタデータを返すストアド・プロシージャがサーバ上にインストールされていることを確認してください。『jConnect for JDBC インストール・ガイド』の第 1 章の「ストアド・プロシージャのインストール」を参照してください。</p>
010P4	<p>An output parameter was received and ignored.</p> <p>説明: 実行したクエリが出力パラメータを返しましたが、アプリケーションの結果処理コードがフェッチしなかったため、無視されました。</p> <p>対処方法: アプリケーションで出力パラメータのデータが必要である場合は、パラメータを取得できるようにアプリケーションを作成し直してください。このためには、CallableStatement を使用してクエリを実行し、registerOutputParameter と getXXX の呼び出しを追加する必要があります。また、DISABLE_UNPROCESSED_PARAM_WARNINGS 接続プロパティを“true”に設定して、この警告が返されないようにすることもできます。このようにすれば、パフォーマンスが向上する可能性があります。</p>
010PF	<p>One or more jars specified in the PRELOAD_JARS connection property could not be loaded.</p> <p>説明: これは、PRELOAD_JARS 接続プロパティを .jar ファイル名のカンマ区切りリストに設定して、DynamicClassLoader を使用した場合に発生します。DynamicClassLoader は、クラスのロード元サーバへの接続をオープンするときに、この接続プロパティで指定されたすべての .jar ファイルを「事前にロード」しようとします。指定された .jar ファイル名の中に、サーバ上に存在しないものがある場合に、上記のエラー・メッセージが表示されます。</p> <p>対処方法: アプリケーションの PRELOAD_JARS 接続プロパティで指定された .jar ファイルがすべてサーバ上に存在し、アクセス可能であることを確認してください。</p>

SQL ステータス	メッセージ／説明／対処方法
010PO	<p>Property LITERAL_PARAM has been reset to "false" because DYNAMIC_PREPARE was set to "true".</p> <p>説明：プリコンパイルされた動的文を使用する場合は、その文にパラメータが送信されるようにする必要があります (パラメータを受け取る文の場合)。LITERAL_PARAMS を “true” に設定すると、サーバに送信される SQL のすべてのパラメータがリテラル値として送信されます。したがって、両方のプロパティを同時に “true” に設定することはできません。</p> <p>対処方法：この警告を回避するには、動的 SQL を使用するとき LITERAL_PARAMS を “true” に設定しないでください。詳細については、「動的 SQL の prepared 文のパフォーマンス・チューニング」(135 ページ) を参照してください。</p>
010RC	<p>The requested ResultSet type and concurrency is not supported. They have been converted.</p> <p>説明：サポートされていない ResultSet のタイプと同時実行性の組み合わせを要求しました。要求した値は変換が必要です。jConnect で使用できる ResultSet のタイプと同時実行性の詳細については、「結果セットでのカーソルの使用方法」(45 ページ) を参照してください。</p> <p>対処方法：サポートされている ResultSet のタイプと同時実行性の組み合わせを要求してください。</p>
010SJ	<p>Metadata accessor information was not found on this database. Please install the required tables as mentioned in the jConnect documentation.</p> <p>説明：メタデータ情報がサーバ上に設定されていません。</p> <p>対処方法：アプリケーションにメタデータが必要な場合は、jConnect 付属の、メタデータを返すストアード・プロシージャをインストールしてください (『jConnect for JDBC インストール・ガイド』の第 1 章の「ストアード・プロシージャのインストール」を参照してください)。メタデータが必要ない場合は、USE_METADATA プロパティを “false” に設定してください。</p>
010SK	<p>Database cannot set connection option _____.</p> <p>説明：アプリケーションは、接続先のデータベースがサポートしていないオペレーションを実行しようとしていました。</p> <p>対処方法：データベースをアップグレードする必要がある可能性があります。または、メタデータ情報の最新バージョンがインストールされていることを確認してください。</p>
010SL	<p>Out-of-date metadata accessor information was found on this database. Ask your database administrator to load the latest scripts.</p> <p>説明：サーバ上のメタデータ情報は古いため、更新する必要があります。</p> <p>対処方法：jConnect 付属の、メタデータを返すストアード・プロシージャをインストールしてください (『jConnect for JDBC インストール・ガイド』の第 1 章の「ストアード・プロシージャのインストール」を参照してください)。</p>
010SM	<p>This database does not support the initial proposed set of capabilities, retrying.</p> <p>説明：Adaptive Server Enterprise の 11.9.2 以前のバージョンには、サーバにない機能を要求するクライアントからのログインを拒否するというバグがあります。この警告は、jConnect がこの状況を検出したことと、サーバによって受け入れられる最大限の機能を使用して接続をリトライしていることを示します。jConnect によってこのバグが検出されたときは、サーバに対する接続が 2 回試行されます。</p> <p>対処方法：クライアントはこのメッセージを無視してもかまいませんが、この警告を回避して接続の試行が 1 回だけ行われるようにする場合は、ELIMINATE_010SM 接続プロパティを “true” に設定してください。Adaptive Server バージョン 12.0 以降に接続するときは、このプロパティを “true” に設定しないでください。</p>

SQL ステータス	メッセージ／説明／対処方法
010SN	<p>Permission to write to file was denied. File: _____. Error message: _____</p> <p>説明：VM でのセキュリティ違反のため、PROTOCOL_CAPTURE 接続プロパティで指定されているファイルへの書き込みパーミッションが拒否されました。これは、指定されたファイルにアプレットが書き込もうとしたときに発生することがあります。</p> <p>対処方法：アプレットからファイルへの書き込みを行うときは、書き込み先ファイル・システムへのアクセス権がアプレットに与えられていることを確認してください。</p>
010SP	<p>File could not be opened for writing. File: _____. Error message: _____</p> <p>対処方法：ファイル名が正しいこと、およびそのファイルへの書き込みが可能であることを確認してください。</p>
010SQ	<p>The connection or login was refused, retrying connection with the host/port address.</p> <p>説明：CONNECTION_FAILOVER 接続プロパティが“true”に設定されているときに、接続先サーバ・リストにあるデータベース・サーバの 1 つに接続できませんでした。したがって、jConnect はリスト内の次のサーバに対して接続を試みます。</p> <p>対処方法：jConnect が別のデータベース・サーバに接続できるのであれば、特に対処は必要ありません。ただし、接続警告の発生元となったサーバに jConnect が接続できなかった理由を調べてください。</p>
010TP	<p>The connection's initial character set,_____, could not be converted by the server. The server's proposed character set,_____, will be used, with conversions performed by jConnect.</p> <p>説明：サーバは jConnect から要求された文字セットを使用できないので、応答には別の文字セットを使用しました。jConnect はこの変更を受け入れて必要な文字セット変換を行います。</p> <p>このメッセージは情報メッセージです。これ以上の結果はありません。</p> <p>対処方法：このメッセージを回避するには、CHARSET 接続プロパティを、サーバがサポートする文字セットに設定してください。</p>
010TQ	<p>jConnect could not determine the server's default character set. This is likely because of a metadata problem. Please install the required tables as mentioned in the jConnect documentation. The connection is defaulting to the ascii_7 character set, which can handle only characters in the range from 0x00 through 0x7F.</p> <p>説明：jConnect は、サーバのデフォルト文字セットを判断できませんでした。この問題が発生したときに正しい変換が保証されるのは、最初の 127 個の ASCII 文字だけです。そのため、この場合は jConnect の文字セットは 7 ビット ASCII に戻ります。このメッセージは情報メッセージです。これ以上の結果はありません。</p> <p>対処方法：jConnect 付属の、メタデータを返すストアド・プロシージャをインストールしてください（『jConnect for JDBC インストール・ガイド』の第 1 章の「ストアド・プロシージャのインストール」を参照してください）。</p>

SQL ステータス	メッセージ／説明／対処方法
010UF	<p>Attempt to execute use database command failed. Error message: _____</p> <p>説明: jConnect は、接続 URL で指定されているデータベースに接続できません。考えられる原因は次の 2 つです。</p> <ul style="list-style-type: none"> URL 内の名前が正しくありません。 USE_METADATA は“true” (デフォルトの状態) ですが、メタデータを返すストアード・プロシージャがインストールされていません。結果として、jConnect は URL 内のデータベースを使用して <code>use database</code> コマンドを実行しようとしたましたが、失敗しました。Adaptive Server Anywhere データベースにアクセスしようとした可能性があります。Adaptive Server Anywhere データベースは、<code>use database</code> コマンドをサポートしていません。 <p>対処方法: URL 内のデータベース名が正しいことを確認してください。メタデータを返すストアード・プロシージャがサーバにインストールされていることを確認してください (『jConnect for JDBC インストール・ガイド』の第 1 章の「ストアード・プロシージャのインストール」、および『リリース・ノート jConnect for JDBC』を参照してください)。SQL Anywhere データベースにアクセスする場合は、URL 内でデータベース名を指定しないようにするか、USE_METADATA を“false”に設定してください。</p>
010UP	<p>Unrecognized connection property _____ ignored.</p> <p>説明: URL 内で設定しようとした接続プロパティは、現時点では jConnect が認識できないプロパティです。認識できないプロパティは無視されます。</p> <p>対処方法: アプリケーション内の URL 定義をチェックし、正しい jConnect ドライバ接続プロパティが参照されていることを確認してください。</p>
010OV	<p>The version of TDS protocol being used is too old. Version: _____</p> <p>説明: サーバは、要求されたバージョンの TDS プロトコルをサポートしていません。jConnect にはバージョン 5.0 以降が必要です。</p> <p>対処方法: 必要なバージョンの TDS をサポートしているサーバを使用してください。詳細については、『jConnect for JDBC インストール・ガイド』のシステム稼動条件の項を参照してください。</p>
01S08	<p>This connection has been enlisted in a Global transaction. All pending statements on the current local transaction(if any) have been rolled back.</p> <p>説明: jConnect は rollback を発行して、現在のローカル・トランザクションをクリアします。この処理は、XAResource.start() が発行された後に、グローバル・トランザクションが登録されると発生します。</p> <p>対処方法: XAResource.start() メソッドを発行する前にローカル・トランザクションをアクティブにした場合は、そのローカル・トランザクションをコミットまたはロールバックする必要があります。</p>
01S09	<p>The local transaction method _____ cannot be used while a global transaction is active on this connection</p> <p>説明: グローバル・トランザクションでローカル・オペレーションが実行されているときに警告します。ローカル・オペレーションの例としては、接続に対して commit() メソッドを呼び出す場合があります。使用できない他のオペレーションは次のとおりです。rollback()、rollback(Savepoint)、setSavepoint()、setSavepoint(String)、releaseSavepoint(Savepoint)、および setAutoCommit()。</p> <p>対処方法: ローカル・トランザクションは、グローバル・トランザクションとは別に実行する必要があります。すべてのローカル・トランザクションとそのオペレーションを完了してから、グローバル・トランザクションを開始するようにしてください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ001	User name property '_____' too long. Maximum length is 30. 対処方法：30 バイト以下にしてください。
JZ002	Password property '_____' too long. Maximum length is 30. 対処方法：30 バイト以下にしてください。
JZ003	Incorrect URL format. URL: _____ 対処方法：URL の形式を確認してください。「URL 接続プロパティのパラメータ」(22 ページ) を参照してください。 PROXY 接続プロパティを使用している場合に、PROXY プロパティのフォーマットが正しくないと、接続を試行するときに JZ003 例外が発生することがあります。 カスケード・プロキシの PROXY フォーマット： <i>ip_address:port_number</i> TDS トンネリング・サーブレットの PROXY フォーマット： <i>http[s]://host:port/tunneling_servlet_alias</i>
JZ004	User name property missing in DriverManager.getConnection(..., Properties) 対処方法：必須であるユーザ・プロパティを指定してください。
JZ006	Caught IOException: _____ 説明：予期しない I/O エラーが下位レイヤから検出されました。このような I/O 例外がキャッチされたときは、ERR_IO_EXCEPTION JZ006 という SQL ステータスを使用して、SQL 例外として再度発生します。多くの場合、このようなエラーはネットワーク通信の問題が原因で発生します。I/O 例外によってデータベース接続がクローズされた場合は、jConnect によって JZ0C1 例外が JZ006 のチェーンに追加されます。クライアント・アプリケーションは、チェーン内に JZ0C1 例外があるかどうかを調べることによって、接続がまだ使用可能かどうかを確認できます。 対処方法：元の I/O 例外メッセージのテキストを調べて、その内容に基づいて処理を進めます。
JZ008	Invalid column index value _____. 説明：要求したカラム・インデックス値が 1 より小さいか、使用可能な最大値を超えています。 対処方法：getXXX メソッドの呼び出しおよび元のクエリのテキストを確認してください。または rs.next を呼び出すようにしてください。
JZ009	Error encountered in conversion. Error message: _____ 説明：可能性のある原因を次に示します。 <ul style="list-style-type: none">• date から int への変換のように、互換性のない 2 つの型の間で変換しようとした。• 数字以外の文字が含まれている文字列を数値型に変換しようとした。• time / date 文字列のフォーマットが正しくないなどのフォーマット・エラーがある。 対処方法：実行しようとした型の変換が JDBC 仕様でサポートされていることを確認してください。文字列が正しくフォーマットされていることを確認してください。数字以外の文字が含まれている文字列から数値型への変換は行わないでください。
JZ00B	Numeric overflow. 説明：BigInteger を TDS の数値型として送信しようとしたましたが、値が大きすぎます。または Java の long を int として送信しようとしたましたが、値が大きすぎます。 対処方法：これらの値は、Sybase では格納できません。long の代わりに、Sybase の数値型を使用してください。Bignum については、解決方法がありません。

SQL ステータス	メッセージ／説明／対処方法
JZ00C	<p>The precision and scale specified cannot accommodate numeric value ____.</p> <p>説明: setBigDecimal メソッドを使用するときに、BigDecimal 値の精度または位取りが、指定された精度または位取りを超えています。</p> <p>対処方法: 指定した精度と位取りが、その BigDecimal 値に十分な大きさであることを確認してください。</p>
JZ00E	<p>Attempt to call execute() or executeUpdate() for a statement where setCursorName() has been called.</p> <p>対処方法: カーソル名が設定されている文に対して execute や executeUpdate を呼び出さないでください。カーソルを削除または更新するには、別の文を使用してください。詳細については、「結果セットでのカーソルの使用方法」(45 ページ) を参照してください。</p>
JZ00F	<p>Cursor name has already been set by setCursorName().</p> <p>対処方法: 同じ文に対してカーソル名を二度設定しないでください。現在のカーソル文の結果セットをクローズしてください。</p>
JZ00G	<p>No column values were set for this row update.</p> <p>説明: ローを更新しようとしたますが、そのカラム値がまったく変更されていません。</p> <p>対処方法: ローのカラム値を変更するには、updateXX メソッドを呼び出してから updateRow を呼び出してください。</p>
JZ00H	<p>The result set is not updatable. Use Statement.setResultSetConcurrencyType().</p> <p>対処方法: 結果セットを読み込み専用から更新可能に変更するには、Statement.setResultSetConcurrencyType メソッドを使用するか、または for update 句を SQL select 文に追加してください。</p>
JZ00I	<p>Invalid scale. The specified scale must be >=0.</p> <p>説明: 位取りは 0 以上でなければなりません。</p> <p>対処方法: 位取り値が負の値でないことを確認してください。</p>
JZ00L	<p>Login failed. Examine the SQLWarnings chained to this exception for the reason(s).</p> <p>対処方法: メッセージ・テキストを調べて、ログインの失敗の原因に応じた処置を取ってください。</p>
JZ00M	<p>Login timed out. Check that your database server is running on the host and port number you specified. Also check the database server for other conditions (such as a full tempdb) that might be causing it to hang.</p> <p>対処方法: エラー・メッセージで提示されている対処方法に従ってください。</p>
JZ010	<p>Unable to deserialize an Object value. Error text: ____</p> <p>対処方法: データベースからの Java オブジェクトが Serializable インタフェースを実装していること、およびローカル CLASSPATH 変数にあることを確認してください。</p>
JZ011	<p>Number format exception encountered while parsing numeric connection property ____.</p> <p>説明: 数値型の接続プロパティに整数以外の値が指定されました。</p> <p>対処方法: 数値型の接続プロパティには整数値を指定してください。</p>
JZ012	<p>Internal Error. Please report it to Sybase technical support. Wrong access type for connection property ____.</p> <p>対処方法: Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ013	<p>Error obtaining JNDI entry: _____</p> <p>対処方法：JNDI URL を訂正するか、またはディレクトリ・サービス内に新しいエントリを作成してください。</p>
JZ014	<p>You may not setTransactionIsolation(Connection.TRANSACTION_NONE). This level cannot be set; it can only be returned by a server.</p> <p>対処方法：アプリケーション・コードで Connection.setTransactionIsolation メソッドを呼び出している部分を調べて、このメソッドに渡している値を確認してください。</p>
JZ015	<p>Illegal value set for the GSSMANAGER_CLASS connection property. The property value must be a String or an Object that extends org.ietf.jgss.GSSManager.</p> <p>対処方法：GSSMANAGER_CLASS プロパティに設定されている値を確認してください。</p>
JZ0BD	<p>Out of range or invalid value used for method parameter.</p> <p>対処方法：メソッドのパラメータ値が正しいことを確認してください。</p>
JZ0BI	<p>メッセージ：setFetchSize：フェッチ・サイズは次の制限内で設定してください。－ 0 <= ロー数 <= (ResultSet 内の最大ロー数)</p> <p>説明：クライアント・アプリケーションが setFetchSize を呼び出すときに指定したロー数が正しくありません。</p> <p>対処方法：setFetchSize の呼び出しに指定しているパラメータ値が上記の範囲内にあることを確認してください。</p>
JZ0BP	<p>Output parameters are not allowed in Batch Update Statements.</p> <p>対処方法：アプリケーション・コードを調べて、バッチ内で出力パラメータを宣言していないことを確認してください。</p>
JZ0BR	<p>The cursor is not positioned on a row that supports the _____ method.</p> <p>説明：現在のローの位置では無効である ResultSet メソッドを呼び出そうとしました（たとえば、カーソルが挿入ローの位置にないときに insertRow を呼び出そうとした場合）。</p> <p>対処方法：現在のロー位置では無効となる ResultSet メソッドを呼び出さないでください。</p>
JZ0BS	<p>バッチ文はサポートされていません。</p> <p>対処方法：データベースに最新バージョンの jConnect メタデータ・ストア・プロシージャをインストールしてください。または、最新バージョンに更新してください。</p>
JZ0BT	<p>The _____ method is not supported for ResultSets of type _____.</p> <p>説明：呼び出そうとした ResultSet メソッドは、そのタイプの ResultSet に対しては無効です。</p> <p>対処方法：ResultSet のタイプに対して無効な ResultSet メソッドを呼び出さないでください。</p>
JZ0C0	<p>Connection is already closed.</p> <p>説明：アプリケーションが、すでにこの接続オブジェクトに対して Connection.close を呼び出しているので、これ以上は使用できません。</p> <p>対処方法：接続がクローズしたときに接続オブジェクトの参照が null になるように、コードを修正してください。</p>
JZ0C1	<p>An IOException occurred which closed the connection.</p> <p>説明：リカバリ不可能な IOException が発生したため、接続がクローズされました。これ以降は、データベースに関する作業にこの接続を使用することはできません。この例外が発生した場合は、必ず前述の JZ006 例外との例外チェーン内に追加されます。</p> <p>対処方法：接続をクローズさせた IOException の原因を調べてください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0CL	<p>You must define the CLASS_LOADER property when using the PRELOAD_JARS property.</p> <p>対処方法：PRELOAD_JARS を null 以外の値に設定するときは、必ず CLASS_LOADER を指定してください。</p>
JZ0CU	<p>getUpdateCount can only be called once after a successful call to getMoreResults, or execute methods.</p> <p>説明：JDBC API に従い、getUpdateCount は 1 つの結果につき 1 回だけ呼び出すようにしてください。</p> <p>対処方法：コード内で、1 つの結果について getUpdateCount を 2 回以上呼び出していないことを確認してください。</p>
JZ0D4	<p>Unrecognized protocol in Sybase JDBC URL: _____.</p> <p>説明：TDS 以外のプロトコルを使用して接続の URL を指定しました。TDS は jConnect が現在サポートする唯一のプロトコルです。</p> <p>対処方法：URL 定義を確認してください。URL で TDS をサブプロトコルとして指定する場合は、次のフォーマットで入力し、大文字と小文字の区別を確認してください。</p> <p>jdbc:sybase:Tds:host:port</p> <p>URL で JNDI をサブプロトコルとして指定する場合は、次の文字列で始まることを確認してください。</p> <p>jdbc:sybase:jndi:</p>
JZ0D5	<p>Error loading protocol _____.</p> <p>対処方法：CLASSPATH システム変数の設定を確認してください。</p>
JZ0D6	<p>Unrecognized version number _____ specified in setVersion. Choose one of the SybDriver.VERSION_* values, and make sure that the version of jConnect that you are using is at or beyond the version you specify.</p> <p>対処方法：メッセージ・テキストを参照してください。</p>
JZ0D7	<p>Error loading url provider _____. Error message: _____</p> <p>対処方法：JNDI URL が正しいことを確認してください。</p>
JZ0D8	<p>Error initializing url provider: _____</p> <p>対処方法：JNDI URL が正しいことを確認してください。</p>
JZ0DP	<p>This statement has no metadata because it was not dynamically prepared. Set the DYNAMIC_PREPARE connection property to true to ensure use of dynamic statements.</p> <p>対処方法：エラー・メッセージを参照してください。</p>
JZ0EM	<p>End of data.</p> <p>対処方法：Sybase 製品の保守契約を結んでいるサポート・センタに連絡してください。</p>
JZ0F1	<p>Sybase high-availability failover connection was requested but the companion server address is missing.</p> <p>説明：REQUEST_HA_SESSION 接続プロパティを “true” に設定するときは、フェールオーバー・サーバも指定する必要があります。</p> <p>対処方法：SECONDARY_SERVER_HOSTPORT 接続プロパティを使用してセカンダリ・サーバを指定するか、JNDI を使用してセカンダリ・サーバを設定します (「高可用性フェールオーバー・サポートの実装」(36 ページ) を参照してください)。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0F2	<p>Sybase high-availability failover has occurred. The current transaction is aborted, but the connection is still usable. Retry your transaction.</p> <p>説明：接続していたバックエンド・データベース・サーバが停止しましたが、セカンダリ・サーバにフェールオーバーしました。データベース接続は引き続き使用可能です。</p> <p>対処方法：クライアント・コードは、この例外をキャッチして、最後にコミットされた時点からトランザクションを再起動する必要があります。例外が正しく処理されていれば、同じ接続オブジェクトで JDBC 呼び出しの実行を続けることができます。</p>
JZ0GC	<p>Error casting a ____ as a GSSManager. Please check the value you are setting for the GSSMANAGER_CLASS connection property. The value must be a String that specifies the fully qualified class name of a GSSManager implementation. Or, it must be an Object that extends org.ietf.jgss.GSSManager.</p> <p>対処方法：メッセージ・テキストを参照してください。</p>
JZ0GN	<p>Error instantiating the class ____ as a GSSManager. The exception was _____. Please check your CLASSPATH and make sure the GSSMANAGER_CLASS property value refers to a fully qualified class name of a GSSManager implementation.</p> <p>対処方法：サード・パーティの GSSManager 実装に必要なすべての .jar ファイルが CLASSPATH 環境変数内で指定されていることを確認してください。</p>
JZ0GS	<p>A Generic Security Services API exception occurred. The major error code is _____. The major error message is _____. The minor error code is _____. The minor error message is _____.</p> <p>対処方法：メジャーおよびマイナーのエラー・コードとメッセージを調べてください。Kerberos 設定をチェックしてください。詳細については、「第3章 セキュリティ」を参照してください。</p>
JZ0H0	<p>Unable to start thread for event handler; event name = _____.</p> <p>対処方法：Sybase 製品の保守契約を結んでいるサポート・センタに連絡してください。</p>
JZ0H1	<p>An event notification was received but the event handler was not found; event name = _____.</p> <p>対処方法：Sybase 製品の保守契約を結んでいるサポート・センタに連絡してください。</p>
JZ0HC	<p>Illegal character '_____' encountered while parsing hexadecimal number.</p> <p>説明：バイナリ値を表す文字列に含まれている文字が、16 進数の文字の範囲 (0-9, a-f) にありません。</p> <p>対処方法：文字列内の文字値が 16 進数の範囲内にあることを確認してください。</p>
JZ0I1	<p>I/O Layer: Error reading stream.</p> <p>説明：接続で、要求された量を読み込めませんでした。ほとんどの場合は、文のタイムアウト時間が経過して、接続がタイムアウトしたことが原因です。</p> <p>対処方法：文のタイムアウト値を大きくしてください。</p>
JZ0I2	<p>I/O layer: Error writing stream.</p> <p>説明：接続で、要求された出力を書き込めませんでした。ほとんどの場合は、文のタイムアウト時間が経過して、接続がタイムアウトしたことが原因です。</p> <p>対処方法：文のタイムアウト値を大きくしてください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ013	<p>Unknown property. This message indicates an internal product problem. Report this error to Sybase Technical support.</p> <p>対処方法：製品内部の問題が発生したことを示します。Sybase 製品の保守契約を結んでいるサポート・センタに連絡してください。</p>
JZ015	<p>An unrecognized CHARSET property was specified: _____.</p> <p>説明：CHARSET 接続プロパティに対して、サポートされていない文字セット・コードを指定しました。</p> <p>対処方法：接続プロパティに対して有効な文字セット・コードを入力してください。「jConnect 文字セット・コンバータ」(31 ページ)を参照してください。</p>
JZ016	<p>An error occurred converting UNICODE to the charset used by the server. Error message: _____</p> <p>対処方法：jConnect クライアントの CHARSET 接続プロパティには、サーバへの送信に必要なすべての文字をサポートできる別の文字セット・コードを選択してください。サーバにも、異なる文字セットをインストールする必要がある場合があります。また、jConnect バージョン 6.05 以降と Adaptive Server Enterprise 12.5 以降を使用している場合は、データを unichar データ型または univarchar データ型としてサーバに送信できます。詳細については、「jConnect を使用して Unicode データを渡す」(30 ページ)を参照してください。</p>
JZ017	<p>No response from proxy gateway.</p> <p>説明：カスケードまたはセキュリティ・ゲートウェイが応答しません。</p> <p>対処方法：ゲートウェイが正しくインストールされ、実行されていることを確認してください。</p>
JZ018	<p>Proxy gateway connection refused. Gateway response: _____</p> <p>説明：PROXY 接続プロパティによって指定された Web サーバ／ゲートウェイが、接続要求を拒否しました。</p> <p>対処方法：プロキシのアクセスとエラー・ログを確認して、接続が拒否された理由を調べてください。プロキシが JDBC ゲートウェイであることを確認してください。</p>
JZ019	<p>This InputStream was closed.</p> <p>説明：getAsciiStream、getUnicodeStream、または getBinaryStream から取得した InputStream を読み込もうとしましたが、InputStream はすでにクローズしていました。このストリームがクローズした原因として、別のカラムに移動したか結果セットをキャンセルしたために、リソースが不足してデータをキャッシュできないことが考えられます。</p> <p>対処方法：キャッシュ・サイズを増やすか、カラムを順番に読み込んでください。</p>
JZ01A	<p>Truncation error trying to send _____.</p> <p>説明：文字列を送信する前の文字セットの変換時にトランケーション・エラーが発生しました。変換後の文字列の長さが、割り当てられたサイズを超えています。</p> <p>対処方法：jConnect クライアントの CHARSET 接続プロパティには、サーバへの送信に必要なすべての文字をサポートできる別の文字セット・コードを選択してください。サーバにも、異なる文字セットをインストールする必要がある場合があります。</p>
JZ01R	<p>getXXX may not be called on a column after it has been updated in the result set with a java.io.Reader.</p> <p>対処方法：Reader を使用して更新した ResultSet カラムに対する getXXX の呼び出しを削除してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0IS	<p>getXXXStream may not be called on a column after it has been updated in the result set.</p> <p>説明：結果セット内でカラムを更新した後に、SybResultSet のメソッド getAsciiStream、getUnicodeStream、または getBinaryStream を使用して、更新後のカラム値を読み込もうとしました。jConnect は、このような使い方をサポートしません。</p> <p>対処方法：更新しているカラムからの入力ストリームをフェッチしないでください。</p>
JZ0JO	<p>Offset and/or length values exceed the actual text/image length.</p> <p>対処方法：使用したオフセット値と長さの値が正しいことを確認してください。</p>
JZ0LC	<p>You cannot call the _____ method on a ResultSet which is using a language cursor to fetch rows. Try setting the LANGUAGE_CURSOR connection property to false.</p> <p>説明：言語カーソルを使用する ResultSet に対して、アプリケーションが ResultSet カーソル・スクロール・メソッドの 1 つを呼び出そうとしました。</p> <p>対処方法：エラー・メッセージを参照してください。</p>
JZ0NC	<p>wasNull called without a preceding call to get a column.</p> <p>説明：wasNull は、getInt や getBinaryStream などのカラムを取得する呼び出しの後にのみ呼び出しが可能です。</p> <p>対処方法：コードを変更して wasNull への呼び出しを移動してください。</p>
JZ0NE	<p>Incorrect URL format. URL: _____. Error message: _____</p> <p>対処方法：URL のフォーマットを確認してください。ポート番号には、数字だけが使用されていることを確認してください。</p>
JZ0NF	<p>Unable to load SybSocketFactory. Make sure that you have spelled the class name correctly, that the package is fully specified, that the class is available in your class path, and that it has a public zero-argument constructor.</p> <p>対処方法：メッセージ・テキストを参照してください。</p>
JZ0P1	<p>Unexpected result type.</p> <p>説明：データベースから返された結果が、アプリケーションに返すことのできないもの、またはアプリケーションがこの時点で受け取ることを想定していないものでした。これは一般的に、アプリケーションでの JDBC の使い方が正しくないために、クエリまたはストアド・プロシージャを実行できないことを示します。JDBC アプリケーションの接続先が Open Server アプリケーションの場合は、Open Server アプリケーションでエラーが発生したことが原因で、送信される結果の順序が予期したとおりにない可能性があります。</p> <p>対処方法：デバッグ・ツール com.sybase.util.Debug(true, "ALL") を使用して、予期しない結果の内容を調べ、原因を明らかにしてください。</p>
JZ0P4	<p>Protocol error. This message indicates an internal product problem. Report this error to Sybase technical support.</p> <p>対処方法：メッセージ・テキストを参照してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0P7	<p>Column is not cached; use RE-READABLE_COLUMNS property.</p> <p>説明：REPEAT_READ 接続プロパティを“false”に設定して、カラムを再読み込みしようとしたか、カラムを誤った順序で読み込みようとした。</p> <p>REPEAT_READ が“false”の場合は、ローのカラム値を一度だけ読み込むことができ、また、昇順カラム／インデックス順でのみカラムを読み込むことができます。たとえば、ローのカラム 3 を読み込んだ後に、その値をもう一度読み込むことや、ローのカラム 2 を読み込むことはできません。</p> <p>対処方法：REPEAT_READ を“true”に設定するか、カラム値を再度読み込まないようにしてください。また、昇順カラム／インデックス順にカラムを読み込むことを確認してください。</p>
JZ0P8	<p>The RSMDBA Column Type Name you requested is unknown.</p> <p>説明：jConnect は、ResultSetMetaData.getColumnTypeName メソッドの中で、カラム型の名前を特定できませんでした。</p> <p>対処方法：データベースに、メタデータ用の最新のストアド・プロシージャがインストールされていることを確認してください。</p>
JZ0P9	<p>A COMPUTE BY query has been detected. That type of result is unsupported and has been cancelled.</p> <p>説明：実行したクエリが COMPUTE 結果を返しましたが、その結果は jConnect でサポートされているものではありません。</p> <p>対処方法：クエリまたはストアド・プロシージャを、COMPUTE BY を使用しないように変更してください。</p>
JZ0PA	<p>The query has been cancelled and the same response discarded.</p> <p>説明：この接続上の他の文によってキャンセルが発行された可能性があります。</p> <p>対処方法：この文とその他の文の SQL 例外および警告のチェーンを確認して、原因を調べてください。</p>
JZ0PB	<p>The server does not support a requested operation.</p> <p>説明：jConnect は、サーバとの接続を確立するときに、どの機能のサポートが必要かをサーバに通知します。サーバは、サポートする機能を jConnect に通知します。最初の機能ネゴシエーションで拒否されたオペレーションをアプリケーションが要求すると、このエラー・メッセージが送信されます。</p> <p>たとえば、データベースが動的 SQL 文のプリコンパイルをサポートしていないけれども、プログラムが SybConnection.prepareStatement(sql_stmt, dynamic) を呼び出し、dynamic が“true”に設定されている場合は、このメッセージが生成されます。</p> <p>対処方法：プログラムを修正して、サポートされていない機能を要求しないようにしてください。</p>
JZ0PC	<p>The number and size of parameters in your query require wide table support. But either the server does not offer such support, or it was not requested during the login sequence. Try setting the JCONNECT_VERSION property to >=6 if you wish to request widetable support.</p> <p>説明：多数のパラメータを持つ文を実行しようとしています。サーバはその数のパラメータを処理するように設定されていません。この例外を生成する基準となるパラメータ数は、送信されるデータのデータ型によって異なります。送信するパラメータの数が 481 以下の場合には、この例外は発生しません。</p> <p>対処方法：このクエリは、Adaptive Server 12.5 以降のサーバに対して実行する必要があります。データベースに接続するときに、JCONNECT_VERSION プロパティを“6”に設定してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0PD	<p>The size of the query in your dynamic prepare is large enough that you require widetable support. But either the server does not offer such support, or it was not requested during the login sequence. Try setting the JCONNECT_VERSION property to >=6 if you wish to request widetable support.</p> <p>説明：多数のパラメータを持つ動的 prepared 文を実行しようとしています、サーバはその数のパラメータを処理するように設定されていません。</p> <p>対処方法：このクエリは、Adaptive Server 12.5 以降のサーバに対して実行する必要があります。データベースに接続するときに、JCONNECT_VERSION プロパティを“6”に設定してください。</p>
JZ0PE	<p>The number of columns in your cursor declaration OR the size of your cursor declaration itself are large enough that you require widetable support. But either the server does not offer such support, or it was not requested during the login sequence. Try setting the JCONNECT_VERSION property to >= 6 if you wish to request wide table support.</p> <p>説明：このエラーは、SELECT 文が 255 カラムを超えるデータを返そうとしたとき、または SELECT 文の実際の長さが非常に長いとき (約 65,500 文字以上) に発生します。</p> <p>対処方法：このクエリは、バージョン 12.5 以降の Adaptive Server に対して実行する必要があります。データベースに接続するときに、JCONNECT_VERSION プロパティを“6”に設定してください。</p>
JZ0PN	<p>Specified port number of _____ was out of range. Port numbers must meet the following conditions: 0<= portNumber <=65535.</p> <p>対処方法：データベース URL 内で指定されているポート番号を確認してください。</p>
JZ0R0	<p>Result set has already been closed.</p> <p>説明：結果セット・オブジェクトに対してすでに ResultSet.close メソッドが呼び出されています。この結果セットを他の処理に使用することはできません。</p> <p>対処方法：結果セットがクローズされたときに ResultSet オブジェクトの参照が null に設定されるように、コードを修正してください。</p>
JZ0R1	<p>Result set is IDLE as you are not currently accessing a row.</p> <p>説明：アプリケーションが、カラム・データを取り出すための ResultSet.getXXX メソッドの 1 つを呼び出しましたが、現在のローがありません。アプリケーションで ResultSet.next を呼び出していないか、ResultSet.next を呼び出したけれども、データがないことを示す“false”が返されています。</p> <p>対処方法：rs.next が“true”に設定されていることを確認してから、rs.getXXX を呼び出してください。</p>
JZ0R2	<p>No result set for this query.</p> <p>説明：Statement.executeQuery を使用しましたが、文からローがまったく返されませんでした。</p> <p>対処方法：ローを返さない文には executeUpdate を使用してください。</p>
JZ0R3	<p>Column is DEAD. This is an internal error. Please report it to Sybase technical support.</p> <p>対処方法：メッセージ・テキストを参照してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0R4	<p>Column does not have a text pointer. It is not a text/image column or the column is NULL.</p> <p>説明: text / image カラムが null の場合は、そのカラムの更新はできません。 null の text / image カラムには、テキスト・ポインタがありません。</p> <p>対処方法: text / image データをサポートしていないカラムに対するテキスト・ポインタを更新または取得しようとしていないことを確認してください。 null である text / image カラムを更新しようとしていないことを確認してください。データを挿入してから更新してください。</p>
JZ0R5	<p>The ResultSet is currently positioned beyond the last row. You cannot perform a get* operation to read data in this state.</p> <p>説明: アプリケーションで、ResultSet の最後のローを越えてロー・ポインタを移動しました。この位置には読み込むデータがないため、get* 操作はすべて無効になります。</p> <p>対処方法: ResultSet の現在位置が最後のローを越えているときはカラム・データを読み込まないように、コードを修正してください。</p>
JZ0RD	<p>You cannot call any of the ResultSet.get* methods on a row that has been deleted with the deleteRow() method.</p> <p>説明: アプリケーションは、すでに削除されたローからデータを取り出そうとしています。取り出すことができる有効なデータはありません。</p> <p>対処方法: 削除されたローからデータを取り出さないように、アプリケーションのコードを修正してください。</p>
JZ0RM	<p>refreshRow may not be called after updateRow or deleteRow.</p> <p>説明: データベース内のローを、SybCursorResult.updateRow を使用して更新した後、または SybCursorResult.deleteRow を使用して削除した後に、SybCursorResult.refreshRow を使用してローをデータベースからリフレッシュしようとした。</p> <p>対処方法: データベースのローを更新または削除した後に、そのローをリフレッシュしないでください。</p>
JZ0S0	<p>Statement state machine: Statement is BUSY.</p> <p>説明: このエラーは、Statement.setCursorname メソッド以外では発生しません。アプリケーションがカーソル名を設定しようとしているときに、文がすでに使用されていて、読み込む必要のある、カーソルを使用しない結果がある場合に発生します。</p> <p>対処方法: 文にカーソル名を設定してから、クエリを実行してください。または、文がビジーでない状態にするために、カーソル名を設定する前に Statement.cancel を呼び出してください。</p>
JZ0S1	<p>Statement state machine: Trying to FETCH on IDLE statement.</p> <p>説明: 文に対する内部エラーが発生しました。</p> <p>対処方法: 文をクローズして、別の文をオープンしてください。</p>
JZ0S2	<p>Statement object has already been closed.</p> <p>説明: Statement.close() メソッドがすでにこのオブジェクトに対して呼び出されているので、この文を他の処理に使用することはできません。</p> <p>対処方法: 文がクローズされたときに Statement オブジェクトの参照が null に設定されるように、アプリケーションを修正してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0S3	<p>The inherited method _____ cannot be used in this subclass.</p> <p>説明：PreparedStatement は、executeQuery(String)、executeUpdate(String)、execute(String) をサポートしません。</p> <p>対処方法：クエリの文字列を渡すには、PreparedStatement ではなく、Statement を使用してください。</p>
JZ0S4	<p>Cannot execute an empty (zero-length) query.</p> <p>対処方法：空のクエリ(“”)を実行しないでください。</p>
JZ0S5	<p>The local transaction method _____ cannot be used while a global transaction is active on this connection.</p> <p>説明：この例外は、分散トランザクションを使用するときに発生します。</p> <p>対処方法：問題の診断方法の詳細については、『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Chapter 7 Distributed Transactions」を参照してください。</p>
JZ0S6	<p>The local transaction method _____ cannot be used on a pre-System 12 XAConnection.</p> <p>説明：この例外は、分散トランザクションを使用するときに発生します。</p> <p>対処方法：問題の診断方法の詳細については、『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)の「Chapter 7 Distributed Transactions」を参照してください。</p>
JZ0S8	<p>An escape sequence in a SQL Query was malformed: '_____'. '</p> <p>説明：このエラーは、不正なエスケープ構文が原因です。</p> <p>対処方法：JDBC のマニュアルを参照して、正しい構文を確認してください。</p>
JZ0S9	<p>Cannot execute an empty (zero-length) query.</p> <p>対処方法：空のクエリ(“”)を実行しないでください。</p>
JZ0SA	<p>Prepared Statement: Input parameter not set, index: _____.</p> <p>対処方法：個々の入力パラメータに値が指定されていることを確認してください。</p>
JZ0SB	<p>Parameter index out of range: _____.</p> <p>説明：パラメータの最大数を超えて、パラメータを取得、設定、または登録しようとした。</p> <p>対処方法：クエリのパラメータ数を確認してください。</p>
JZ0SC	<p>Callable Statement: attempt to set the return status as an InParameter.</p> <p>説明：ステータスを返すストアド・プロシージャの呼び出しを準備しましたが、パラメータを 1 に設定しようとしています。この値は、リターン・ステータスを表します。</p> <p>対処方法：このタイプの呼び出しでは、パラメータを 2 以上に設定してください。</p>
JZ0SD	<p>No registered parameter found for output parameter.</p> <p>説明：これは、アプリケーションの論理エラーを示します。パラメータに対して getXXX または wasNull を呼び出しましたが、まだパラメータが 1 つも読み込まれていないか、出力パラメータが 1 つもありませんでした。</p> <p>対処方法：アプリケーションで、呼び出し可能な文の出力パラメータを登録したことを確認してください。また、文が実行され、出力パラメータが読み込まれたことを確認してください。</p>
JZ0SE	<p>Invalid object type specified for setObject().</p> <p>説明：PreparedStatement.setObject に渡された型引数が正しくありません。</p> <p>対処方法：JDBC のマニュアルを確認してください。引数は、java.sql.Types からの定数でなければなりません。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0SF	<p>No Parameters expected. Has query been sent?</p> <p>説明：パラメータのない文のパラメータを設定しようとしてしました。</p> <p>対処方法：クエリが送信されたことを確認してから、パラメータを設定してください。</p>
JZ0SG	<p>An RPC did not return as many output parameters as the application had registered for it.</p> <p>説明：このエラーは、ストアド・プロシージャ内の“OUTPUT”パラメータとして宣言したよりも多いパラメータに対して <code>CallableStatement.registerOutParam</code> を呼び出すと発生します。詳細については、「RPC が返す出力パラメータの数が登録されている数よりも少ない」(130 ページ) を参照してください。</p> <p>対処方法：ストアド・プロシージャおよび <code>registerOutParameter</code> 呼び出しを確認してください。該当するすべてのパラメータを“OUTPUT”として宣言していることを確認してください。それには、コードの次の行を調べてください。</p> <pre>create procedure yourproc (@p1 int OUTPUT, ...</pre> <p>注意 SQL Anywhere を使用しているときにこのエラーが発生した場合は、SQL Anywhere パージョン 5.5.04 にアップグレードしてください。</p>
JZ0SH	<p>A static function escape was used, but the metadata accessor information was not found on this server.</p> <p>対処方法：静的関数のエスケープを使用する前に、メタデータ・アクセサ情報をインストールしてください。</p>
JZ0SI	<p>A static function escape _____ was used which is not supported by this server.</p> <p>対処方法：このエスケープを使用しないでください。</p>
JZ0SJ	<p>Metadata accessor information was not found on this database.</p> <p>対処方法：メタデータ情報をインストールしてから、メタデータを読み出してください。</p>
JZ0SK	<p>The oj escape is not supported for this type of database server.</p> <p>Workaround: use server-specific outer join syntax, if supported. Consult server documentation.</p> <p>対処方法：エラー・メッセージを参照してください。また、最新バージョンの jConnect メタデータをインストールしてください。</p>
JZ0SL	<p>Unsupported SQL type _____.</p> <p>説明：アプリケーションで宣言されているパラメータの型が、jConnect ではサポートされていません。</p> <p>対処方法：可能であれば、別の型を使用してパラメータを宣言してください。Types.NULL や <code>PreparedStatement.setObject(null)</code> は使用しないでください。</p>
JZ0SM	<p>jConnect could not execute a stored procedure because there was a problem sending the parameter(s). This problem was likely caused because the server does not support a specific datatype, or because jConnect did not request support for that datatype at connect time. Try setting the JCONNECT_VERSION connection property to a higher value. Or, if possible, try sending your procedure execution command as a language statement.</p>
JZ0SN	<p>setMaxFieldSize: field size cannot be negative.</p> <p>対処方法：<code>setMaxFieldSize</code> を呼び出すときは、正の値または 0 (無制限) を使用してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0SO	Invalid ResultSet concurrency type: _____. 対処方法：宣言されている同時実行性が <code>ResultSet.CONCUR_READ_ONLY</code> または <code>ResultSet.CONCUR_UPDATABLE</code> であることを確認してください。
JZ0SP	Invalid ResultSet type: _____. 対処方法：宣言されている ResultSet のタイプが <code>ResultSet.TYPE_FORWARD_ONLY</code> または <code>ResultSet.TYPE_SCROLL_INSENSITIVE</code> であることを確認してください。jConnect は、 <code>ResultSet.TYPE_SCROLL_SENSITIVE</code> タイプの ResultSet はサポートしていません。
JZ0SQ	In valid UDT type _____. 説明：DatabaseMetaData.getUDTs メソッドを呼び出したときに、ユーザ定義型が <code>Types.JAVA_OBJECT</code> 、 <code>Types.STRUCT</code> 、 <code>Types.DISTINCT</code> のいずれかでない場合は、この例外が発生します。 対処方法：上記の 3 つの UDT のいずれかを使用してください。
JZ0SR	setMaxRows: max rows cannot be negative. 対処方法： <code>setMaxRows</code> を呼び出すときは、正の値または 0 (無制限) を使用してください。
JZ0SS	setQueryTimeout: クエリ・タイムアウトには負の値を設定できません。
JZ0ST	jConnect cannot send a Java object as a literal parameter in a query. Make sure that your database server supports Java objects and that the LITERAL_PARAMS connection property is set to false when you execute this query.
JZ0SU	A Date or Timestamp parameter was set with a year of ____, but the server can only support year values between ____ and _____. If you're trying to send data to date or timestamp columns or parameters on Adaptive Server Anywhere, you may wish to send your data as Strings, and let the server convert them. 説明：Adaptive Server Enterprise と SQL Anywhere では、 <code>datetime</code> と <code>date</code> に指定できる値の範囲が異なります。 <code>datetime</code> 値の年は 1753 年以降でなければなりません、 <code>date</code> データ型は 1 以降の年を保持できます。 対処方法：送信する <code>date</code> / <code>timestamp</code> 値が、許容される範囲内にあることを確認してください。
JZ0T2	Listener thread read error. 対処方法：ネットワーク通信を確認してください。
JZ0T3	Read operation timed out. 説明：クエリに対する応答の読み込みに割り当てられた時間が経過しました。 対処方法： <code>Statement.setQueryTimeout</code> を呼び出して、タイムアウト時間を長くしてください。
JZ0T4	Write operation timed out. Timeout in milliseconds: _____. 説明：要求の送信に割り当てられた時間が経過しました。 対処方法： <code>Statement.setQueryTimeout</code> を呼び出して、タイムアウト時間を長くしてください。
JZ0T5	Cache used to store responses is full. 対処方法： <code>STREAM_CACHE_SIZE</code> 接続プロパティにデフォルト値または現在の値よりも大きい値を使用してください。
JZ0T6	Error reading tunneled TDS URL. 説明：URL のヘッダの読み込み中に、Tunnelled プロトコルが失敗しました。 対処方法：接続に対して定義した URL を確認してください。

SQL ステータス	メッセージ/説明/対処方法
JZ0T7	<p>Listener thread read error -- caught ThreadDeath. Check network connection.</p> <p>対処方法: ネットワーク接続を確認して、アプリケーションを再度実行してください。スレッドが引き続きアボートされる場合には、Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。</p>
JZ0T8	<p>Data received for an unknown request. Please report this error to Sybase Technical Support.</p>
JZ0T9	<p>Request to send not synchronized. Please report this error to Sybase Technical Support.</p> <p>対処方法: メッセージ・テキストを参照してください。</p>
JZ0TC	<p>Attempted conversion between an illegal pair of types.</p> <p>説明: Java の型と SQL の型との間の変換に失敗しました。</p> <p>対処方法: 要求した型の変換が JDBC 仕様でサポートされていることを確認してください。</p>
JZ0TE	<p>Attempted conversion between an illegal pair of types. Valid database types are: '_____.'</p> <p>説明: データベース・カラムのデータ型と <code>ResultSet.getXXX</code> 呼び出しで要求されたデータ型は、暗黙的な変換が可能ではありません。</p> <p>対処方法: エラー・メッセージに表示された有効なデータ型のいずれかを使用してください。</p>
JZ0TI	<p>jConnect cannot make a meaningful conversion between the database type of _____ and the requested type of _____.</p> <p>説明: この種類の例外は、たとえば、データベースから返された <code>time</code> 値に対して、アプリケーションが <code>ResultSet.getObject(int, Types.DATE)</code> を呼び出した場合に発生します。</p> <p>対処方法: データベースのデータ型が、取り出そうとするオブジェクトの型に暗黙的に変換可能であることを確認してください。</p>
JZ0TO	<p>Read operation timed out.</p> <p>説明: この例外は、ソケット読み込みがタイムアウトになったときに発生します。</p> <p>対処方法: <code>Statement.setQueryTimeout</code> を呼び出して、タイムアウト時間を長くしてください。また、実行しているクエリまたはストアド・プロシージャを調べて、予想以上に時間がかかった原因を特定してください。</p>
JZ0TS	<p>Truncation error trying to send _____.</p> <p>説明: 指定された文字列の長さは、アプリケーションが送信しようとしていた長さを超えています。したがって、文字列は宣言された長さにトランケートされます。</p> <p>対処方法: トランケートされないように、長さのプロパティを設定してください。</p>
JZ0US	<p>The SybSocketFactory connection property was set, and the PROXY connection property was set to the URL of a servlet. The jConnect driver does not support this combination. If you want to send secure HTTP from an applet running within a browser, use a proxy URL beginning with "https://".</p> <p>対処方法: メッセージ・テキストを参照してください。</p>

SQL ステータス	メッセージ／説明／対処方法
JZ0XC	<p>_____ is an unrecognized transaction coordinator type.</p> <p>説明：メタデータ情報には、サーバは分散トランザクションをサポートしていますが、jConnect は使用されているプロトコルをサポートしていないことが示されています。</p> <p>対処方法：最新のメタデータ・スクリプトがインストールされていることを確認してください。エラーが解消されない場合は、Sybase 製品の保守契約を結んでいるサポート・センタに連絡してください。</p>
JZ0XS	<p>The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.</p> <p>説明：jConnect が接続しようとしたサーバは、分散トランザクションをサポートしていません。</p> <p>対処方法：このサーバで XADatasource を使用しないでください。または分散トランザクションを使用できるようにサーバをアップグレードするか、設定してください。</p>
JZ0XU	<p>Current user does not have permission to do XA-style transactions. Be sure user has _____ role.</p> <p>説明：データベースに接続しているユーザは、分散トランザクションの実行を許可されていません。ほとんどの場合は、ユーザに適切な役割 (ブランク部分に表示) が付与されていないことが原因です。</p> <p>対処方法：エラー・メッセージに表示されている役割をユーザに付与するか、その役割が付与された別のユーザとしてトランザクションを実行してください。</p>
S0022	<p>Invalid column name '_____'. </p> <p>説明：名前によってカラムを参照しましたが、その名前のカラムがありませんでした。</p> <p>対処方法：カラム名を確認してください。</p>
ZZ00A	<p>The method _____ has not been completed and should not be called.</p> <p>説明：実装されていないメソッドを使用しようとしました。</p> <p>対処方法：詳細については、使用している jConnect のバージョンに添付されていたリリース・ノートを確認してください。jConnect の最新バージョンがそのメソッドを実装しているかどうかを確認するには、jConnect web ページ (http://www.sybase.com) を参照してください。実装されていない場合は、そのメソッドを使用しないでください。</p>

jConnect サンプル・プログラム

この付録では、jConnect サンプル・プログラムについて説明します。

トピック名	ページ
IsqlApp の実行	179
jConnect のサンプル・プログラムとサンプル・コードの実行	181

IsqlApp の実行

IsqlApp を使用すると、コマンド・ラインから `isql` コマンドを発行して、jConnect サンプル・プログラムを実行できます。

IsqlApp の構文は次のとおりです。

```
IsqlApp [-U username]
        [-P password]
        [-S servername]
        [-G gateway]
        [-p {http|https}]
        [-D debug_class_list]
        [-v]
        [-I input_command_file]
        [-c command_terminator]
        [-C charset]
        [-L language]
        [-K service_principal_name]
        [-F JAAS_login_config_file_path]
        [-T sessionID]
        [-V <version {2,3,4,5}>]
```

パラメータ	説明
-U	サーバに接続するログイン ID。
-P	指定したログイン ID のパスワード。
-S	接続先のサーバの名前。
-G	ゲートウェイ・アドレス。HTTP プロトコルの場合の URL は http://host:port 。 暗号化をサポートする HTTPS プロトコルを使用する場合の URL は https://host:port/servlet_alias 。
-p	HTTP プロトコルを使用するか、暗号化をサポートする HTTPS プロトコルを使用するかを指定する。

パラメータ	説明
-D	すべてのクラスについて、またはカンマで区切って指定したクラスだけについて、デバッグをオンにする。たとえば、 -D ALL すべてのクラスについてデバッグ出力を表示する。 -D SybConnection, Tds SybConnection クラスと Tds クラスについてだけデバッグ出力を表示する。
-v	表示または印刷の冗長出力をオンにする。
-I	IsqlApp が、キーボードからではなく、ファイルからコマンドを取得するようにする。このパラメータの後に、IsqlApp への入力に使用するファイルの名前を指定する。このファイルには、コマンド・ターミネータを含むようにしてください (デフォルトでは "go")。
-c	行に単独で入力されたときにコマンドを終了させるキーワード (たとえば、"go") を指定できる。これによって、ターミネータ・キーワードを使用するまで複数行にわたってコマンドを入力できる。コマンド・ターミネータを指定しない場合は、コマンドは復帰改行ごとに終了する。
-C	TDS を介して渡される文字列用の文字セットを指定する。 文字セットが指定されていない場合は、IsqlApp はサーバのデフォルト文字セットを使用する。
-L	サーバから返されるエラー・メッセージおよび jConnect メッセージを表示する言語を指定する。
-K	Adaptive Server に対して Kerberos ログインを使用するかどうかを指定する。このパラメータによって、サービスのプリンシパル名を設定する。例： -K myASE この例は、Kerberos ログインを実行することと、サーバのサービス・プリンシパル名が myASE であることを指定している。 詳細については、「 第 3 章 セキュリティ 」を参照してください。
-F	JAAS ログイン構成ファイルのパスを指定する。-K オプションを使用する場合は、このプロパティを設定する必要がある。例： -F /foo/bar/exampleLogin.conf jConnect インストール環境の sample2 ディレクトリにあるサンプル ConnectKerberos.java を参照。詳細については、「 第 3 章 セキュリティ 」を参照してください。
-T	このパラメータが設定されているとき、jConnect は、TDS トンネリング・ゲートウェイによってオープンされたままになっている既存の TDS セッション上でアプリケーションが通信を再開しようとしていると想定する。jConnect はログイン・ネゴシエーションをスキップし、アプリケーションからの要求をすべて指定のセッション ID に転送する。
-V	バージョン固有の特性を使用できるようにする。「 JCONNECT_VERSION の使用 」(6 ページ) 参照。

注意 各オプション・フラグの後にスペースを 1 つ入力する必要があります。

コマンドライン・オプションの詳細な説明を表示するには、次のように入力します。

```
java IsqlApp -help
```

次の例は、ポート “3756” を使用して “myserver” というホスト上のデータベースに接続し、“myscript” という **isql** スクリプトを実行する方法を示します。

```
java IsqlApp -U sa -P sapassword  
-S jdbc:sybase:Tds:myserver:3756  
-I $JDBC_HOME/sp/myscript -c run
```

注意 GUI を使用して **isql** のコマンドを実行するためのアプレットが、次の場所にあります。

`$JDBC_HOME/sample2/gateway.html` (UNIX)

`%JDBC_HOME%\sample2\gateway.html` (Windows)

jConnect のサンプル・プログラムとサンプル・コードの実行

jConnect に付属しているサンプル・プログラムは、この章で説明している多くの項目について例を示すもので、さまざまな JDBC クラスとメソッドでの jConnect の動作方法の理解に役立ちます。この項では、参考のためにサンプル・コードの一部も示します。

サンプル・アプリケーション

jConnect をインストールするときに、サンプル・プログラムもインストールできます。これらのサンプルにはソース・コードが含まれており、jConnect がどのようにしてさまざまな JDBC クラスとメソッドを実装するかを確認することができます。サンプル・プログラムをインストールする方法については、『*jConnect for JDBC* インストール・ガイド』を参照してください。

注意 jConnect サンプル・プログラムはデモ用としてのみ提供されています。

サンプル・プログラムは、jConnect インストール・ディレクトリの *sample2* サブディレクトリ内にインストールされます。*sample2* サブディレクトリにある *index.html* ファイルには、使用できるサンプルのリストと、各サンプルの説明が含まれています。*index.html* では、サンプル・プログラムの内容を参照し、アプレットとして実行することもできます。

サンプル・アプレットの実行

使用している Web ブラウザで、サンプル・プログラムのいくつかをアプレットとして実行できます。これによって、出力結果を表示しながらソース・コードを参照することができます。

サンプル・プログラムをアプレットとして実行する場合は、Web ブラウザで <http://localhost:8000/sample2/index.html> と入力して、Web サーバ・ゲートウェイを起動します。

SQL Anywhere でサンプル・プログラムを実行する

サンプル・プログラムはすべて Adaptive Server に対応していますが、SQL Anywhere に対応しているものは限られています。SQL Anywhere に対応しているサンプル・プログラムの最新のリストについては、*sample2* サブディレクトリにある *index.html* を参照してください。

SQL Anywhere で使用できるサンプル・プログラムを実行するには、SQL Anywhere サーバに *pubs2_any.sql* スクリプトをインストールする必要があります。このスクリプトは、*sample2* サブディレクトリにあります。

Windows の場合は、DOS コマンド・ウィンドウで次のように入力します。

```
java IsqlApp -U dba -P password
-S jdbc:sybase:Tds:[hostname]:[port]
-I %JDBC_HOME%\sample2\pubs2_any.sql -c go
```

UNIX の場合は、次のように入力します。

```
java IsqlApp -U dba -P password
-S jdbc:sybase:Tds:[hostname]:[port]
-I $JDBC_HOME/sample2/pub2s_any.sql -c go
```

サンプル・コード

次のサンプル・コードは、どのように jConnect ドライバを呼び出し、接続を行い、SQL 文を発行して結果を処理するかを示します。

```
import java.io.*;
import java.sql.*;

public class SampleCode
{
    public static void main(String args[])
    {
        try
        {
            /*
             * Open the connection. May throw a SQLException.
             */
            DriverManager.registerDriver(
```

```
(Driver) Class.forName(
    "com.sybase.jdbc4.jdbc.SybDriver").newInstance());

    Connection con = DriverManager.getConnection(
        "jdbc:sybase:Tds:myserver:3767", "sa", "");
/*
 * Create a statement object, the container for the SQL
 * statement. May throw a SQLException.
 */
    Statement stmt = con.createStatement();
/*
 * Create a result set object by executing the query.
 * May throw a SQLException.
 */
    ResultSet rs = stmt.executeQuery("Select 1");
/*
 * Process the result set.
 */

    if (rs.next())
    {
        int value = rs.getInt(1);
        System.out.println("Fetched value " + value);
    }
    rs.close()
    stmt.close()
    con.close()
} //end try
/*
 * Exception handling.
 */
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
    System.exit(1);
} //end catch

catch (Exception e)
{
    e.printStackTrace();
    System.exit(1);
} //end catch

    System.exit(0);
}
}
```


索引

記号

.jar ファイル
事前ロード 90

A

Adaptive Server
接続 10
接続例 22
Adaptive Server Anywhere
image データの送信 59, 62
Java オブジェクトの格納と取得 78
SERVICENAME 接続プロパティ 21
ユーロ記号 35
ALTERNATE_SERVER_NAME 接続プロパティ 11
APPLICATIONNAME 接続プロパティ 11
ASE のデータ型
date、time、datetime 63

B

BE_AS_JDBC_COMPLIANT_AS_POSSIBLE 接続プロパティ 12
BigDecimal の位取り変更
ドライバ・パフォーマンスの向上 134
bigint
サポートされるデータ型 101

C

CACHE_COLUMN_METADATA 接続プロパティ 12
CANCEL_ALL 接続プロパティ 12
CAPABILITY_TIME 接続プロパティ 13
CAPABILITY_WIDETABLE 接続プロパティ 13
CHARSET 接続プロパティ 13
設定 32
CHARSET_CONVERTER_CLASS 接続プロパティ 14, 32

CLASS_LOADER 接続プロパティ 14
CLASSPATH
デバッグ用の設定 124
CONNECTION_FAILOVER 接続プロパティ 14, 24

D

Debug クラス 123
Debug サブレット引数 156
DISABLE_UNICHAR_SENDING 接続プロパティ 14
DISABLE_UNPROCESSED_PARAM_WARNINGS 接続プロパティ 14
DYNAMIC_PREPARE 接続プロパティ 14

E

ENABLE_BULK_LOAD 接続プロパティ 15
ENABLE_SERVER_PACKETSIZE 接続プロパティ 15
ENCRYPT_PASSWORD 接続プロパティ 15
EncryptPassword 75
error messages
error-message handler example 73
ESCAPE_PROCESSING_DEFAULT 接続プロパティ 15, 141
EXPIRESTRING 接続プロパティ 15

F

FAKE_METADATA 接続プロパティ 16

G

GET_BY_NAME_USES_COLUMN_LABEL 接続プロパティ 16
GSSMANAGER_CLASS 接続プロパティ 16

H

HOSTNAME 接続プロパティ 16
 HOSTPROC 接続プロパティ 16
 HTTP 147

I

IGNORE_DONE_IN_PROC 接続プロパティ 16
 image データの送信 59
 IS_CLOSED_TEST 接続プロパティ 17
 isql アプレット
 サンプルの実行 153
 IsqlApp ユーティリティ 179

J

Java オブジェクト
 ASA 6.0 での格納と取得 78
 テーブル内のカラム・データとしての格納 78
 jConnect
 アプリケーションでのメモリの問題 129
 カーソルの使用 45
 ゲートウェイ 147
 サンプル・プログラム 181
 接続プロパティの設定 10
 設定 5
 定義 2
 デバッグ 123
 パフォーマンスの向上 133
 呼び出し 8
 JCONNECT_VERSION 接続プロパティ 6, 17
 JDBC
 インタフェース 1
 制約、制限、逸脱 98
 定義 1
 ドライバ・タイプ 2
 JDBC 2.0
 オプション・パッケージ拡張サポート 90
 標準の拡張 90
 JDBC ドライバ
 JDBC-ODBC ブリッジ 2
 ネイティブ API / 一部 Java で実装された 2
 ネイティブ・プロトコル / すべて Java で実装された 2
 ネット・プロトコル / すべて Java で実装された 2

JDBC のデータ型

 date、time、timestamp 63
 jdbc.drivers 8
 JNDI
 コンテキスト情報 28
 使用 24
 データベースの命名 90

L

LANGUAGE 接続プロパティ 17
 LANGUAGE_CURSOR 142
 LANGUAGE_CURSOR 接続プロパティ 17
 Lightweight Directory Access Protocol (LDAP) 25
 LITERAL_PARAMS 接続プロパティ 17

N

NEWPASSWORD 接続プロパティ 17

P

PACKETSIZE 接続プロパティ 17
 PASSWORD 接続プロパティ 18
 PRELOAD_JARS 接続プロパティ 18
 PreparedStatement
 カーソルの使用 51
 PROMPT_FOR_NEWPASSWORD 接続プロパティ 18
 PROTOCOL_CAPTURE 接続プロパティ 18
 PROXY 接続プロパティ 18
 PureConverter クラス 31, 32

Q

QUERY_TIMEOUT_CANCEL_ALL 接続プロパティ 18

R

REMOTEPWD 接続プロパティ 18
 REPEAT_READ 134
 REPEAT_READ 接続プロパティ 18
 REQUEST_HA_SESSION 18
 REQUEST_KERBEROS_SESSION 19
 RETRY_WITH_NO_ENCRYPTION 接続プロパティ 19

RMNAME 接続プロパティ 19
rs.getBytes() 63

S

SECONDARY_SERVER_HOSTPORT 接続プロパティ 19
SELECT_OPENS_CURSOR 接続プロパティ 19
SERIALIZE_REQUESTS 接続プロパティ 19
SERVER_INITIATED_TRANSACTIONS 接続プロパティ 19
SERVERTYPE 接続プロパティ 20
SERVICE_PRINCIPAL_NAME 接続プロパティ 20
SERVICENAME 接続プロパティ 20
SESSION_ID 接続プロパティ 20
SESSION_TIMEOUT 接続プロパティ 20
setRemotePassword() 42
SkipDoneProc サブレット引数 156
SQL Anywhere 20
SQL の例外メッセージと警告メッセージ 159
SQLINITSTRING 接続プロパティ 20
Statement.cancel() メソッド 12
STREAM_CACHE_SIZE 接続プロパティ 20
Sybase 拡張機能の変更 144
SybEventHandler 67
SybMessageHandler 71
SYB SOCKET_FACTORY 接続プロパティ 20

T

TDS 2
 サブレット 148
 サブレットのインストール方法 155
 サブレットのシステム要件 155
 サブレット引数の設定 156
 セッションの再開 157
 セッションのトラッキング 157
 通信の取得 126
 トンネリング 148
TDS セッションのトラッキング 157
TDS 通信の取得 126
TdsResponseSize サブレット引数 156
TdsSessionIdleTimeout サブレット引数 156
TEXTSIZE 接続プロパティ 20
TruncationConverter クラス 31, 35
TYPE_SCROLL_INSENSITIVE の制限 52

U

unichar データ型および univarchar データ型 30
unitext 101
unsigned int
 サポートされるデータ型 101
URL
 構文 21
 接続プロパティのパラメータ 22
USE_METADATA 接続プロパティ 21
USER 接続プロパティ 21

V

VERSIONSTRING 接続プロパティ 21

W

Web サーバ・ゲートウェイ 147

X

XAServer 96

あ

アプリケーション
 デバッグ機能をオフにする 124
 デバッグ機能をオンにする 124
アプレット 152

い

位置付け更新と削除
 JDBC 1.x メソッドの使用 47
 JDBC 2.0 メソッドの使用 48
逸脱、JDBC 標準 98
イベント通知 67
 例 68
イメージ・データ
 TextPointer オブジェクトの取得 60
 TextPointer クラスのパブリック・メソッド 59
 TextPointer.sendData を使用した更新の実行 60
 TextPointer.sendData() を使用したカラムの更新 60
送信 59

索引

インストール
 TDS サブレット 155
 エラー・メッセージ・ハンドラ 72
インタフェース、JDBC 1

え

エラー
 ストアド・プロシージャ 130
 接続 128
エラー・メッセージ
 SQL 例外と警告 159
 Sybase 固有 69
 エラー・メッセージ・ハンドラのインストール 72
 処理 69
 処理のカスタマイズ 71

お

オフにする方法、アプリケーションのデバッグ 124
オンにする方法、アプリケーションのデバッグ 124

か

カーソル 45
 PreparedStatement の使用 51
 作成 45
カーソル結果セット
 JDBC 1.x メソッドを使用した位置付け更新と削除 47
 JDBC 2.0 メソッドを使用した位置付け更新と削除 48
 位置付け更新 48
 カラムの更新 48
 削除 48
 データベースを更新するためのメソッド 49
 ローの削除 50
 ローの挿入 50
カーソルの作成 45
カーソルのパフォーマンス 141
 LANGUAGE_CURSOR 接続プロパティ 141
拡張機能の変更、Sybase 144
格納、テーブル内のカラム・データとしての Java オブジェクト 78
 前提条件 79
 データベースからの Java オブジェクトの受信 80
 データベースへの Java オブジェクトの送信 79

カラム
 カーソル結果セットでの更新 48
 カーソル結果セットでの削除 48
関連マニュアル viii

け

ゲートウェイ 147
 構成 148
 接続の拒否 128

こ

高可用性 (HA) サポート 36
更新
 ストアド・プロシージャの結果セットからのデータ
 ベースの更新 57
高度な機能 64
国際化 30

さ

サーバ間のリモート・プロシージャ・コール 41
サブレット 147
 TDS 148
サブレット引数
 Debug 156
 SkipDoneProc 156
 TdsResponseSize 156
 TdsSessionIdleTimeout 156
再開
 TDS セッション 157
サポートされるデータ型 101
サンプル・プログラム 181

し

システム・プロパティ
 jdbc.drivers 8
事前ロード、.jar ファイル 90
処理
 エラー・メッセージ 69

す

ストアド・プロシージャ

エラー 130

結果セットからのデータベースの更新 57

実行 102

せ

接続

Adaptive Server 10

JNDI を使用してサーバに接続 24

エラー 128

ゲートウェイ接続の拒否 128

プール 94

接続プロパティ

ALTERNATE_SERVER_NAME 11

APPLICATIONNAME 11

BE_AS_JDBC_COMPLIANT_AS_POSSIBLE 12

CACHE_COLUMN_METADATA 12

CANCEL_ALL 12

CAPABILITY_TIME 13

CAPABILITY_WIDETABLE 13

CHARSET 13

CHARSET_CONVERTER_CLASS 14, 32

CLASS_LOADER 14

CONNECTION_FAILOVER 14, 24

DISABLE_UNICHAR_SENDING 14

DISABLE_UNPROCESSED_PARAM_WARNINGS
14

DYNAMIC_PREPARE 14

ENABLE_BULK_LOAD 15

ENABLE_SERVER_PACKETSIZE 15

ENCRYPT_PASSWORD 15

ESCAPE_PROCESSING_DEFAULT 15, 141

EXPIRESTRING 15

FAKE_METADATA 16

GET_BY_NAME_USES_COLUMN_LABEL 16

GSSMANAGER_CLASS 16

HOSTNAME 16

HOSTPROC 16

IGNORE_DONE_IN_PROC 16

IS_CLOSED_TEST 17

JCONNECT_VERSION 6, 17

LANGUAGE 17

LANGUAGE_CURSOR 17, 142

LANGUAGE_CURSOR とカーソルのパフォーマンス
141

LITERAL_PARAMS 17

NEWPASSWORD 17

PACKETSIZE 17

PASSWORD 18

PRELOAD_JARS 18

PROMPT_FOR_NEWPASSWORD 18

PROTOCOL_CAPTURE 18

PROXY 18

QUERY_TIMEOUT_CANCELS_ALL 18

REMOTEPWD 18

REPEAT_READ 18, 134

REQUEST_HA_SESSION 18

REQUEST_KERBEROS_SESSION 19

RETRY_WITH_NO_ENCRYPTION 19

RMNAME 19

SECONDARY_SERVER_HOSTPORT 19

SELECT_OPENS_CURSOR 19

SERIALIZE_REQUESTS 19

SERVER_INITIATED_TRANSACTIONS 19

SERVERTYPE 20

SERVICE_PRINCIPAL_NAME 20

SERVICENAME 20

SESSION_ID 20

SESSION_TIMEOUT 20

SQLINITSTRING 20

STREAM_CACHE_SIZE 20

SYB SOCKET_FACTORY 20

TEXTSIZE 20

URL での設定 22

USE_METADATA 21

USER 21

VERSIONSTRING 21

設定 10

設定

jConnect 5

jConnect 接続プロパティ 10

TDS サブレット引数 156

選択、文字セット・コンバータ・クラス 32

た

対象読者 vii

つ

通貨記号、ユーロ 35

索引

て

- データ
 - image 59
- データ型
 - ASE の date、time、datetime 63
 - JDBC の date、time、timestamp データ型 63
 - unichar および univarchar 30
- データベース
 - JNDI による命名 90
 - 格納、テーブル内のカラム・データとしての Java オブジェクト 78
- デバッグ 123
 - CLASSPATH の設定 124
 - Debug クラスのインスタンスの取得 123
 - アプリケーションでオフにする 124
 - アプリケーションでオンにする 124
 - メソッド 125

と

- 動的クラス・ロード 82
- ドライバ
 - JDBC タイプ 2
 - プロパティ 11
- トラブルシューティング 123
- トンネリング
 - TDS 148

ね

- ネイティブ API / 一部 Java で実装されたドライバ 2
- ネイティブ・プロトコル / すべて Java で実装されたドライバ 2
- ネット・プロトコル / すべて Java で実装されたドライバ 2

は

- パスワードの暗号化 75
- バッチ更新
 - ストアド・プロシージャ 56
- パフォーマンス、改善 133
 - BigDecimal の位取り変更 134
 - カーソル 141
 - 動的 SQL の prepared 文のチューニング 135
 - 文字セット変換 135

ひ

- 非直列化 85

ふ

- プール、接続 94
- プロパティ
 - ドライバ 11
- 分散トランザクションのサポート 96

ま

- マイグレート、jConnect アプリケーション
 - jConnect アプリケーションのマイグレート 143
- マルチスレッディング
 - マルチスレッドに対する調整 98
- マルチバイト文字セット
 - コンバータ・クラス 31
 - サポートされる 33

め

- メタデータ
 - USE_METADATA 21
 - アクセス 43
 - サーバ側の実装 44
- メモリの問題、jConnect アプリケーション 129

も

- 文字セット
 - コンバータ・クラス 31
 - サポートされる 33
 - 設定 32
- 文字セット・コンバータ・クラス
 - PureConverter 32
 - TruncationConverter 31
 - 選択 32
- 文字セット変換
 - ドライバ・パフォーマンスの向上 135
 - パフォーマンスの向上 33

ゆ

ユーティリティ
 IsqlApp 179
ユーロ通貨記号 35

よ

呼び出し、jConnect 8

り

リモート・プロシージャ・コール (RPC)
 サーバ間 41

ろ

ロー
 カーソル結果セットからの削除 50
 カーソル結果セットへの挿入 50
ローカライゼーション 30

わ

ワイド・テーブル 42

