プログラマーズリファレンス

SAP jConnect™ for JDBC 16.0

ドキュメント ID： DC38164-01-0707100-01
改訂： 2014 年 3 月

目次

SAP jConnect for JDBC

SAP® jConnect™ for JDBC は SAP® が提供する高性能の JDBC ドライバです。

SAP jConnect for JDBC は、次の両方のドライバです。

- ネイティブプロトコル/すべて Java で実装されたドライバ
- ネットプロトコル/すべて Java で実装されたドライバ

SAP jConnect が使用するプロトコルは、SAP® Adaptive Server® Enterprise (SAP® ASE) および SAP® Open Server™ アプリケーションのネイティブプロトコルである Tabular Data Stream™ 5.0 (TDS バージョン 5) です。 SAP jConnect は JDBC 標準を実装しており、SAP 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。

- SAP ASE
- SAP® SQL Anywhere®
- SAP® IQ
- SAP® Replication Server®
- SAP® DirectConnect™

さらに、SAP jConnect for JDBC は、SAP 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。

SAP 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。

参照：

- JDBC 標準の制約と解釈 (117 ページ)

Java Database Connectivity (JDBC)

Oracle Corporation 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 : JDBC インタフェース

インタフェース	説明
`java.sql.Driver`	データベース URL に対するドライバを見つける。
`java.sql.Connection`	特定のデータベースに接続する。
`java.sql.Statement`	ユーザが SQL 文を実行できるようにする。
`java.sql.Prepared-Statement`	パラメータを使用する SQL 文を処理する。
`java.sql.CallableS-tatement`	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。
`java.sql.ResultSet`	SQL 文の結果を取得する。
`java.sql.DatabaseMe-taData`	このインタフェースを使用して、接続先のデータベースに関する情報にアクセスする。
`java.sql.ResultSet-MetaData`	このインタフェースを使用して、ResultSet に関する情報を取得する。
`javax.sql.Rowset`	JDBC RowSet の実装を処理する。
`javax.sql.DataSource`	データソースへの接続を処理する。
`javax.sql.Connec-tionPoolDataSource`	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 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、Oracle Technology Network for Java を参照してください。

プログラミング情報

SAP jConnect for JDBC の基本的なコンポーネントとプログラミングに関する要件について説明します。

SAP jConnect ドライバの開始、接続プロパティの設定、データベースサーバへの接続を行い、SAP jConnect 機能の使用方法を説明します。 JDBC プログラミングの詳細については、Oracle Technology Network for Java の Java 開発者用のリソースページを参照してください。

SAP jConnect のバージョンプロパティ

JCONNECT_VERSION 接続プロパティは、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、SAP Adaptive Server 15.5 では、SAP 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 SAP Adaptive Server 15.5 の SAP jConnect 7.0 に接続するときは、対象となる SAP 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 SAP 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、SAP 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。

SAP 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 メソッド

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 SAP 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 SAP 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、SAP 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、SAP 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 接続プロパティ

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します。

有効な JCONNECT_VERSION 値と、これらの値に関連付けられた jConnect の特性を参照してください。

表 2 : SAP jConnect バージョンに関連する機能

JCONNECT_VERSION	機能
7.0	SAP jConnect 7.0 の動作は、次のサポートを要求する 7.0 特有の動作以外は SAP jConnect 6.05 と同じである。 • bigdatetime および bigtime SQL データ型をサーバに要求する。 15.5 より前のバージョンの SAP Adaptive Server では、この要求は無視される。 • JDBC 4.0 をサポート。 • ENABLE_BULK_LOAD の有効な値は null (デフォルト)、ARRAYINSERT_WITH_MIXED_STATEMENTS、ARRAYINSERT、BCP、および LOG_BCP である。
6.05	SAP jConnect 6.05 の動作は、次のサポートを要求する 6.05 特有の動作以外は SAP jConnect 6.0 と同じである。 • メタデータ情報を含む計算カラム。 • 長い識別子。 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。

JCON-NECT_VER-SION	機能
6.0	SAP jConnect 6.0 の動作は、次のサポートを要求する 6.0 特有の動作以外は SAP jConnect 5.x と同じである。 • `date` および `time` SQL データ型。 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。 • `unichar` および `univarchar` データ型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。 • ワイドテーブル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。 • `DISABLE_UNICHAR_SENDING` のデフォルト値は false に設定される。
5.0	SAP jConnect 5.x の動作は、SAP jConnect 4.0 と同じである。
4.0	SAP jConnect 4.0 の動作は、次のサポートを要求する 4.0 特有の動作以外は SAP jConnect 3.0 と同じである。 • `LANGUAGE` 接続プロパティのデフォルト値は null である。 • 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。
3.0	SAP jConnect 3.0 の動作は、次に示す 3.0 特有の動作以外は SAP jConnect 2.0 と同じである。 • `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。

JCONNECT_VERSION	機能
2.0	• 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 Truncation-Converter クラスであり、指定されている場合の CHARSET_CONVERTER のデフォルト値は CheckPureConverter クラスである。 • 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。

参照：

- JDBC 4.0 仕様のサポート (106 ページ)
- JDBC 標準の制約と解釈 (117 ページ)
- SAP jConnect の文字セットコンバータ (48 ページ)
- date および time データ型 (80 ページ)
- JDBC 3.0 仕様のサポート (106 ページ)
- Adaptive Server でのワイドテーブルのサポート (59 ページ)
- 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 (97 ページ)
- SAP jConnect を使用して Unicode データを渡す (46 ページ)

SAP jConnect ドライバの呼び出し

SAP jConnect を登録および呼び出して、SAP 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。

次のサンプルコードは、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： Java アプレットでは、`System.getProperties` を使用できません。 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、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 プログラムを使用して、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 SAP 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 SAP Adaptive Server サーバへの接続プールを設定します。

`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、SAP jConnect をプログラム的に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

SAP jConnect を使用して SAP Adaptive Server または SAP SQL Anywhere データベースへの接続を確立します。

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大文字と小文字は区別されません。

接続プロパティの設定

サーバに接続する前に、接続プロパティを設定する必要があります。

次のいずれかの方法で接続プロパティを設定します。

- 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。

このサンプルコードでは、`DriverManager.getConnection` メソッドを使用します。 SAP 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

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が基づいている現在の設定
- 渡された URL およびプロパティ

SAP jConnect 接続プロパティ

SAP jConnect の接続プロパティとそのデフォルト値について説明します。

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。

`getClientInfo()` および `setClientInfo()` 標準メソッドを使用して、次のようにプロパティを動的に設定できます。

表 3 : 接続プロパティ

プロパティ	説明
ALTERNATE_SERVER_NAME	ミラーリングされた SAP 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)。 JDBC URL 構文は、`jdbc:sybase:Tds:<hostname>:<port#>/database?connection_property=value;` である。 ただし、ALTERNATE_SERVER_NAME を設定すると、SAP jConnect では *hostname* 変数と *port* 変数の値が無視される。 代わりに、SAP 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。 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 Guide』(英語) を参照。 `ALTERNATE_SERVER_NAME` は、ミラーリングされていない SAP SQL Anywhere サーバでも使用できる。 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る。 デフォルト値 は NULL。 このプロパティは静的である。
APPLICATION-NAME	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。 このプロパティに提供される値を分析するように、サーバ側をプログラミングできる。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
BE_AS_JDBC_COMPLIANT_AS_POSSIBLE	SAP 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。 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)。 • 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) デフォルト値は false。 このプロパティは静的である。
CACHE_COLUMN_METADATA	**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は、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** メタデータ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。 SAP Adaptive Server 15.7 ESD #1 以降に接続する場合は、SUPPRESS_ROW_FORMAT 接続プロパティを使用する。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
PRE_CACHE_DATA-TYPE_IN-FO	データ型メタデータを取得するために **Statement** またはその派生インタフェースを繰り返し使用する場合、**PRE_CACHE_DATATYPE_INFO** 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。 **PRE_CACHE_DATATYPE_INFO** が true に設定されると、さまざまな `ResultsetMetadata` API に対応するすべてのユーザ定義データ型に関する情報 (`isCaseSensitive`、`isSearchable` など) が接続時にキャッシュされる。 この情報に次にアクセスするときは、キャッシュからアクセスできる。 **PRE_CACHE_DATATYPE_INFO** が false (デフォルト) の場合は、SAP jConnect はユーザ定義データ型情報をキャッシュしない。 **注意：** 接続先のデータベースに存在するユーザ定義データ型の数によっては、接続の確立に要する時間が長くなることがあります。 デフォルト値は false。 このプロパティは動的である。

プロパティ	説明
CANCEL_ALL	Statement.cancel メソッドの動作を次のように指定する。 • 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、stmtA が Statement オブジェクトであれば、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、実行が開始され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、すで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。 この例では、CANCEL_ALL を false に設定する。 *props* は接続プロパティを指定するための Properties オブジェクトである。 `props.put("CANCEL_ALL", "false");` 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 を使用する。 デフォルト値は次のとおり。 • true - JCONNECT_VERSION <= “3” の場合 • false - JCONNECT_VERSION >= “4” の場合 このプロパティは静的である。

プロパティ	説明
CAPABILITY_TIME	JCONNECT_VERSION >= 6 の場合にのみ使用。 SAP 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、java.sql.Time 型のすべてのパラメータまたは エスケープリテラル {t ...} は TIME として処理される。 6.0 以前のバージョンの SAP jConnect では、このようなパラメータを DATETIME として扱い、java.sql.Time 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 datetime または smalldatetime である場合は、日付の部分もデータベースに格納される。 SAP 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。 java.sql.Time の代わりに datetime または smalldatetime 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 false のままにする。 このプロパティを false 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、SAP jConnect は java.sql.Time パラメータまたはエスケープリテラル {t ...} を DATETIME として処理する。 このプロパティを true に設定すると、SAP jConnect は SAP Adaptive Server に接続した場合に java.sql.Time パラメータを TIME データ型として処理する。 smalldatetime または datetime 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 false にすることが望ましい。 デフォルト値は false。 このプロパティは静的である。
CAPABILITY_WIDETABLE	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、このプロパティを false に設定する。 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。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
CHARSET	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 null の場合は、string 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。 DISABLE_UNICHAR_SENDING が false に設定された状態で SAP 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、SAP jConnect がこのことを検知する。 このとき、SAP jConnect はその文字データを unichar データとしてサーバに送信する。これにより、クライアントは Unicode データを unichar/univarchar カラムおよびパラメータに挿入できる。 デフォルト値 は NULL。 このプロパティは静的である。
CHARSET_CONVERTER_CLASS	SAP jConnect で使用する文字セットコンバータクラスを指定する。 SAP jConnect は、SybDriver.setVersion からのバージョン設定、または JCONNECT_VERSION 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。 デフォルト値はバージョンに依存する。 このプロパティは静的である。
CLASS_LOADER	このプロパティには、作成した DynamicClassLoader オブジェクトを設定する。 DynamicClassLoader 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が CLASSPATH に存在しない Java クラスをロードする。 デフォルト値 は NULL。 このプロパティは静的である。
CONNECTION_FAILOVER	JNDI (Java Naming and Directory Interface) とともに使用する。 デフォルト値は true。 このプロパティは静的である。

プロパティ	説明
CRC	このプロパティを true 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。 デフォルト値は false。 このプロパティは静的である。
DATABASE	接続情報が SAP の `interfaces` 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する。 `interfaces` ファイルの URL でデータベース名を指定することはできない。 デフォルト値 は NULL。 このプロパティは静的である。
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。 デフォルト値は 0 (タイムアウトなし)。 このプロパティは動的である。
DELETE_WARNINGS_FROM_EXCEPTION_CHAIN	SQLWarning を SQLException チェーンで保持するか削除するかを指定する。 値: • true - SAP jConnect は SQLWarning オブジェクトを SQLException チェーンから削除する。 • false - SAP jConnect は SQLWarning オブジェクトを SQLException チェーンに保持する。 デフォルト値は true。 このプロパティは静的である。

プロパティ	説明
DISABLE_UNICHAR_SENDING	クライアントアプリケーションが `unichar` 文字を非 `unichar` 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。 SAP jConnect 6.05 以降では、このプロパティのデフォルトは false となる。 古いバージョンの SAP jConnect を使用しているクライアントから `unichar` 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 false に設定する必要がある。 デフォルト値はバージョンに依存する。 このプロパティは静的である。
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、SAP 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、SAP 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)。 デフォルト値は false。 このプロパティは静的である。
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。 デフォルト値は true。 このプロパティは動的である。
EARLY_BATCH_READ_THRESHOLD	読み込むローの数を指定する。この数に達すると、リーダスレッドではバッチへのサーバ応答の送信が減らされる。 早期読み込みが必要になることがない場合、この値を -1 に設定する。 デフォルト値は -1。 このプロパティは静的である。

プロパティ	説明
ENABLE_BULK_LOAD	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。 有効な値は次のとおり。 • null - バルクロードを無効にする。 • 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。 • LOG_BCP - BCP と同様。ただし完全リカバリに備えて完全トランザクションをダンプする。 デフォルト値 は NULL。 このプロパティは動的である。
ENABLE_LOB_ LOCATORS	SAP jConnect がクライアント側のマテリアライズされた LOB とサーバ側の LOB ロケータのどちらを作成するかを指定する。 有効な値は次のとおり。 • false - SAP jConnect は、クライアント側のマテリアライズされた LOB を使用する。 つまり、LOB の全データがクライアント側で処理されキャッシュされる。 • true - autocommit が false に設定されている場合にのみ機能する。設定されていない場合は内部的に値が false に変更される。 true に設定されている場合は、LOB データをクライアント側に保管する代わりにサーバロケータが使用される。 デフォルト値は false。 このプロパティは動的である。

プロパティ	説明
ENABLE_SERVER_PACKET-SIZE	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 PACKETSIZE 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、PACKETSIZE 接続プロパティが使用される。 デフォルト値は true。 このプロパティは静的である。
ENCRYPT_PASS-WORD	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。 ENCRYPT_PASSWORD は RETRY_WITH_NO_ENCRYPTION より優先度が高い。 デフォルト値は false。 このプロパティは静的である。
ESCAPE_PROCESS-ING_ DE-FAULT	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、SAP 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る。 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、パフォーマンスが若干向上する可能性がある。 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 SAP® IQ などのバックエンドサーバで役に立つ。 デフォルト値は true。 このプロパティは静的である。

プロパティ	説明
EXECUTE_BATCH_PAST_ERRORS	SAP jConnect が個々の文の実行中に遭遇した致命的でないエラーをバッチ更新オペレーションが無視し、バッチ更新を完了できるようにするか、バッチ更新オペレーションをアボートするかを指定する。 有効な値は次のとおり。 • true - 個々の文の実行中に遭遇した致命的でないエラーをバッチ更新オペレーションが無視し、バッチ更新を完了できるようにする。 • false - 致命的でないエラーに遭遇した場合にバッチ更新をアボートする。 デフォルト値は false。 このプロパティは静的である。
EXPIRE-STRING	ライセンスの有効期限日 `date` を示す。 評価版の SAP jConnect 以外は、有効期限は Never に設定される。 デフォルト値は never。 このプロパティは静的かつ読み取り専用。
FAKE_METADATA	偽のメタデータを返す。 `ResultSetMetaData` のメソッド `getCatalogName`、`getSchemaName`、および `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ので、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。 このプロパティが false の場合は、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。 ワイドテーブルが使用可能な状態で、SAP 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る。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
GET_BY_NAME_USES_COLUMN_LABEL	SAP jConnect 6.0 より前のバージョンとの下位互換性を実現する。 SAP 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 SAP 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。 SAP 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。 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。 `getByte`、`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。 デフォルト値は true。 このプロパティは静的である。
GET_COLUMN_ LABEL_FOR_NAME	`ResultMetaData.getColumnName` 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、SAP jConnect 5.5 以前との下位互換性を維持する。 有効な値は次のとおり。 • true - `ResultMetaData.getColumnName` がカラムラベルを返す。 • false - `ResultMetaData.getColumnName` がカラム名を返す。 デフォルト値は false。 このプロパティは静的である。
GSSMANAGER_CLASS	`org.ietf.jgss.GSSManager` クラスのサードパーティ実装を指定する。 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。 プロパティに文字列を設定する場合は、文字列の値はサードパーティ 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。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
HOMOGENEOUS_BATCH	SAP Adaptive Server の最適化されたパッチプロトコルを呼び出し、バッチオペレーションを `PreparedStatement` オブジェクトに対して高速化する。 有効な値は次のとおり。 • true - 最適化されたバッチプロトコルを使用する。 • false - SAP jConnect が新しい最適化されたバッチプロトコルをサポートする SAP Adaptive Server に接続されている場合でも、最適化されていないバッチプロトコルを使用する。 デフォルト値は true。 このプロパティは動的である。
HOSTNAME	現在のホストの名前を表す。 デフォルト値は none。 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。 このプロパティは静的である。
HOSTPROC	ホストマシン上のアプリケーションプロセスを表す。 デフォルト値は none。 このプロパティは静的である。
IGNORE_DONE_IN_PROC	(ストアドプロシージャの) 中間更新結果は返さずに最終結果セットのみを返す。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
IGNORE_WARN-INGS	警告メッセージのチェックと生成を行うかどうかを指定する。 タイムスタンプ値を SAP Adaptive Server の `date` データ型と `time` データ型に格納するときに、Java タイムスタンプよりも精度が下がる。このプロパティでは、この格納時の精度の損失に関する警告だけをチェックする。 有効な値は次のとおり。 • true - SAP jConnect は警告メッセージのチェックと生成を行わないため、パフォーマンスが向上する。 • false - デフォルト値。この値に設定すると、SAP jConnect は警告メッセージのチェックと生成を行う。 `IGNORE_WARNINGS` を true に設定する場合は、これらの設定がアプリケーションに及ぼす影響を十分にテストしてから設定するようにする。 デフォルト値は false。 このプロパティは静的である。
IMPLICIT_CURSOR_FETCH_SIZE	データベースに送信されるすべての **select** クエリで読み込み専用カーソルをオープンするように SAP jConnect に強制する場合、`SELECT_OPENS_CURSOR` プロパティと共にこのプロパティを使用する。 `Statement.setFetchSize` 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。 デフォルト値は 0。 このプロパティは静的である。
INTER-NAL_QUERY_TIMEOUT	このプロパティを使用して、SAP jConnect 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。 デフォルト値は 0 (タイムアウトなし)。 このプロパティは動的である。
IS_CLOSED_TEST	**Connection.isClosed** 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。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
J2EE_TCK_COMPLIANT	このプロパティを true に設定すると、SAP jConnect 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 false を使用することが推奨される。 デフォルト値は false。 このプロパティは静的である。
JAVA_CHARSET_MAPPING	ユーザ定義の文字セットマッピングを指定し、SAP Adaptive Server のデフォルトの文字セットマッピングより優先する。 デフォルト値は none。 このプロパティは静的である。
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。 デフォルト値はバンドルされた JCE プロバイダ。 このプロパティは静的である。
JCONNECT_VERSION	バージョン固有の特性を設定する。 デフォルト値は 7。 このプロパティは静的である。
LANGUAGE	SAP jConnect 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 `syslanguages` 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。 デフォルト値はバージョンに依存する。 このプロパティは静的である。
LANGUAGE_CURSOR	SAP jConnect 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。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
LITERAL_PARAMS	true に設定すると、`PreparedStatement` インタフェースの `setXXX` 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、パラメータマーカは SQL 文内に残り、パラメータ値が別にサーバに送信される。 デフォルト値は false。 このプロパティは静的である。
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。 デフォルト値 は NULL。 このプロパティは静的である。
OPTIMIZE_FOR_ PERFORMANCE	SAP jConnect のパフォーマンスを強化するプロパティを有効にするかどうかを指定する。 このプロパティが制御するのは、`IGNORE_WARNINGS` プロパティだけである。 有効な値は次のとおり。 • true - SAP jConnect はパフォーマンス強化モードで実行される。 • false - デフォルト値。この場合、SAP jConnect は通常モードで実行される。 `OPTIMIZE_FOR_PERFORMANCE` を true に設定する場合は、これらの設定がアプリケーションに及ぼす影響を十分にテストしてから設定するようにする。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
OPTIMIZE_STRING_CONVERSIONS	文字列変換の最適化を有効にするかどうかを指定する。 この最適化動作によって、SQL の prepared 文の実行にクライアントで character データ型が使用される場合の SAP jConnect のパフォーマンスが向上する可能性がある。 有効な値は次のとおり。 • 0 - 文字列変換の最適化は有効にならない。 • 1 - SAP jConnect で UTF8 またはサーバのデフォルト文字セットが使用される場合に、文字列変換の最適化が有効になる。 • 2 - すべての状況で文字列変換の最適化が有効になる。 デフォルト値は 0。 このプロパティは静的である。
PACKETSIZE	ネットワークパケットサイズを表す。 SAP 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 SAP jConnect と SAP Adaptive Server によって選択されるようにすることが望ましい。 デフォルト値は 512。 このプロパティは静的である。
PASSWORD	ログインの password.StringString 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。 デフォルト値は none。 このプロパティは静的である。
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 SAP jConnect ドライバを使用する他の接続でも使用できる。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
PROMPT_FOR_ NEW-PASS-WORD	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。 有効な値は次のとおり。 • 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 の値を確認し、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。 デフォルト値は false。 このプロパティは静的である。
PROTO-COL_ CAP-TURE	アプリケーションと SAP 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。 デフォルト値 は NULL。 このプロパティは動的である。
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。 デフォルト値は none。 このプロパティは静的である。
QUERY_TIMEOUT_CANCELS_ALL	読み込み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 false。 このプロパティは動的である。

プロパティ	説明
RELEASE_LOCKS_ON_CURSOR_CLOSE	カーソルがクローズされた場合に SAP Adaptive Server が独立性レベル 2 および 3 で共有読み込み専用カーソルのロックを解除するかを指定する。 • false - クローズ時に共有カーソルのロック解除を無効にする。 • true - クローズ時に共有カーソルのロック解除を有効にする。 デフォルト値は false。 このプロパティは静的である。
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。 デフォルト値は none。 このプロパティは静的である。
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。 デフォルト値は true。 このプロパティは静的である。
REQUEST_HA_ SESSION	接続しているクライアントが高可用性 (HA) フェールオーバセッションを開始するかどうかを示す。 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ない。 フェールオーバセッションの要求の柔軟性を高めるには、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。 true に設定すると、SAP jConnect はフェールオーバ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が設定されていても、フェールオーバセッションは開始されない。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
REQUEST_KERBEROS_SESSION	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 プロパティの値の入力も必要である。 GSSMANAGER_CLASS プロパティの値を指定することもできる。 デフォルト値は false。 このプロパティは静的である。
RETRY_WITH_NO_ENCRYPTION	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。 ENCRYPT_PASSWORD プロパティおよび RETRY_WITH_NO_ENCRYPTION プロパティの両方を true に設定すると、SAP 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、SAP 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。 デフォルト値は false。 このプロパティは静的である。
RMNAME	分散トランザクション (XA) を使用する場合の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。 デフォルト値 は NULL。 このプロパティは静的である。
SECONDARY_SERVER_HOSTPORT	クライアントが HA フェールオーバ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 に設定されていなければ、このプロパティは無視される。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
SELECT_OPENS_CURSOR	`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。 同じ文に対して `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。 `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る。 デフォルト値は false。 このプロパティは静的である。
SEND_BATCH-PARAMS_IMMEDIATE	SAP jConnect が現在のローのパラメータを送信するのは `PreparedStatement.addBatch()` を呼び出した直後か、`PreparedStatement.executeBatch()` を呼び出した後のみかを指定する。 • true - SAP jConnect は `PreparedStatement.addBatch()` を呼び出した直後に現在のローのパラメータを送信する。 これにより、クライアントのメモリの使用が最小化され、サーバにはバッチパラメータを処理するための時間がより多く与えられる。 • false - SAP jConnect は、`PreparedStatement.executeBatch()` を呼び出した後にのみバッチパラメータを送信する。 デフォルト値は false。 このプロパティは動的である。
SERIALIZE_ REQUESTS	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。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
SERVER_INITIATED_TRANSACTIONS	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、SAP 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 SAP 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。 デフォルト値は true。 このプロパティは静的である。
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。 SAP 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。 デフォルト値は none。 このプロパティは静的である。
SERVERTYPE	SAP® Open Switch™ 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、SAP jConnect から SAP Open Switch に特定の命令を送信し、SAP Open Switch 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。 デフォルト値は none。 このプロパティは静的である。
SERVICE_PRINCIPAL_NAME	SAP 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。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
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、SAP 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。 SAP 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。 デフォルト値 は NULL。 このプロパティは静的である。
SESSION_TIMEOUT	HTTP トンネルセッション (SAP jConnect TDS トンネリングサー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。 デフォルト値 は NULL。 このプロパティは静的である。
SETMAXROWS_ AFFECTS_SELECT_ ONLY	`setMaxRows` が **select** 文が返すローのみを制限して、JDBC 仕様との一貫性を保つかどうかを指定する。 有効な値は次のとおり。 • true - `Statement.setMaxRows(int max)` は **select** 文の結果として返されるロー数のみを制限する。 • false - `Statement.setMaxRows(int max)` は **select** 文、 **insert** 文、**update** 文、および **delete** 文の結果として返されるロー数を制限する。 SAP Adaptive Server 15.5 以前に接続している場合、`SETMAXROWS_AFFECTS_SELECT_ONLY` は無視される。 デフォルト値は true。 このプロパティは静的である。
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、`Statement.executeUpdate` 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。 デフォルト値 は NULL。 このプロパティは静的である。

プロパティ	説明
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。 デフォルト値は null (キャッシュサイズの制限なし)。 このプロパティは動的である。
STRIP_BLANKS	文字列値をテーブルに格納する前に、サーバで強制的に文字列値から先行ブランクおよび後続ブランクを削除する。 有効な値は次のとおり。 • false - クライアントから送信された文字列値がそのまま格納される。 • true - 文字列値がテーブルに格納される前に、先行ブランクと後続ブランクがその文字列値から削除される。 デフォルト値は false。 このプロパティは静的である。
SUPPRESS_CONTROL_TOKEN	コントロールトークンを抑制する。 有効な値は次のとおり。 • false - コントロールトークンは送信される。 • true - コントロールトークンは抑制される。 デフォルト値は false。 このプロパティは静的である。
SUPPRESS_PARAM_FORMAT	準備された動的 SQL 文の実行時に、SAP jConnect クライアントは `SUPPRESS_PARAM_FORMAT` 接続文字列プロパティを使用してパラメータフォーマットメタデータを抑制することができる。 クライアントは可能な場合には送信するパラメータメタデータを減らしてパフォーマンスを改善する。 有効な値は次のとおり。 • false - 選択、挿入、更新の各オペレーションではパラメータフォーマットメタデータは抑制されない。 • true - デフォルト値。パラメータフォーマットメタデータが可能な限り省略される。 デフォルト値は true。 このプロパティは静的である。

プロパティ	説明
SUPPRESS_ROW_FORMAT	SAP jConnect で、クライアントは SUPPRESS_ROW_FORMAT 接続文字列プロパティを使用して、準備された動的 SQL 文のローフォーマットが変更されたときに限り SAP Adaptive Server が TDS_ROWFMT データまたは TDS_ROWFMT2 データを送信するように強制することができる。 SAP Adaptive Server がクライアントに送信するデータを少なくすることができるため、パフォーマンスが向上する。 有効な値は次のとおり。 • false - ローフォーマットが変更されていない場合でも、TDS_ROWFMT データまたは TDS_ROWFMT2 データが送信される。 • true - デフォルト値。ローフォーマットが変更された場合にのみサーバが TDS_ROWFMT または TDS_ROWFMT2 を送信するように強制する。 デフォルト値は true。 このプロパティは静的である。
SUPPRESS_ROW_FORMAT2	SAP Adaptive Server が可能な場合には TDS_ROWFMT2 バイトシーケンスではなく TDS_ROWFMT バイトシーケンスを使用してデータを送信するよう指定する。 有効な値は次のとおり。 • false - デフォルト値。TDS_ROWFMT2 は抑制されない。 • true- サーバがデータをできる限り TDS_ROWFMT で送信するように強制する。 SAP Adaptive Server 15.7 ESD #1 以降に接続している場合は、SUPPRESS_ROW_FORMAT 接続プロパティを使用する。 デフォルト値は false。 このプロパティは静的である。

プロパティ	説明
SYBSOCKET_ FACTORY	SAP jConnect でカスタムソケット実装を使用できるようにする。 SYBSOCKET_FACTORY を次のいずれかに設定する。 • com.sybase.jdbcx.SybSocketFactory を実装するクラスの名前。 • DEFAULT。この場合は、新しい java.net.Socket 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。 デフォルト値 は NULL。 このプロパティは静的である。
TEXTSIZE	テキストサイズを設定できる。 SAP Adaptive Server および SAP 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。 SAP jConnect のメタデータテーブルがインストールされている場合、SAP jConnect はその値を 2GB に変更する。 ただし、SAP Open Switch に接続する場合にこの値を設定すると、SAP Open Switch 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。 デフォルト値は 2GB。 このプロパティは静的である。
USE_METADATA	接続を確立するときに DatabaseMetaData オブジェクトを作成して初期化する。 DatabaseMetaData 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。 SAP jConnect 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 DatabaseMetaData を使用する。 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、SAP 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する必要がある。 『jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」を参照。 デフォルト値は true。 このプロパティは静的である。

プロパティ	説明
USER	ログイン ID を指定する。 `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。 デフォルト値は none。 このプロパティは静的である。
VERSION-STRING	JDBC ドライバのバージョン情報 (読み込み専用)。 デフォルト値は、SAP jConnect ドライバのバージョン。 このプロパティは静的である。

参照：

- DYNAMIC_PREPARE 接続プロパティ (157 ページ)
- パスワードの暗号化 (95 ページ)
- セキュリティ (123 ページ)
- SAP jConnect での最適化されたバッチ処理 (159 ページ)
- カーソルのパフォーマンス (161 ページ)
- フェールオーバのサポート (54 ページ)
- Adaptive Server でのワイドテーブルのサポート (59 ページ)
- CONNECTION_FAILOVER プロパティ (44 ページ)
- ラージオブジェクトのロケータのサポート (83 ページ)
- Connection.isClosed と IS_CLOSED_TEST の使用 (117 ページ)
- デフォルトの文字セットマッピングより優先使用 (52 ページ)
- JCONNECT_VERSION 接続プロパティ (6 ページ)
- カーソルクローズ時のロックの解放 (68 ページ)
- TDS トンネリング (167 ページ)
- DynamicClassLoader の使用方法 (102 ページ)
- SAP jConnect を使用して Unicode データを渡す (46 ページ)
- 文字セットコンバータの選択 (48 ページ)
- .jar ファイルの事前ロード (105 ページ)

SAP Adaptive Server への接続

Java アプリケーションでは、URL を定義し、SAP jConnect ドライバを使用して SAP 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
```

構文の説明は次のとおりです。

- *jdbc:sybase* - ドライバを指定します。
- *Tds* - SAP Adaptive Server と通信するための SAP 通信プロトコルです。
- *host:port* - SAP Adaptive Server のホスト名と受信ポートです。 データベースや SAP 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： SAP SQL Anywhere または DirectConnect を使用して、特定のデータベースに接続します。 “/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。

次のコードは、ホスト “myserver” 上のポート 3697 で受信する SAP 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、SAP jConnect ドライバ接続プロパティの値を指定します。

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し、プロパティ名とその値を URL 定義に追加します。 次の構文を使用します。

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

複数の接続プロパティを設定し、各接続プロパティ値の前に “&” を付けて追加します。 次に例を示します。

```
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、SAP jConnect for JDBC に対するサーバ情報を提供します。

`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。

SAP 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。

パスを設定しないと、SAP jConnect はエラーを返します。

`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”
```

構文の説明は次のとおりです。

- 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"
```

構文の説明は次のとおりです。

- 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

sql.ini ファイルと interfaces ファイルの SSL 用フォーマットについて説明します。

sql.ini ファイルの SSL 用フォーマット:

```
[SYBSRV2]
master=nlwnsck,mango1,4100,ssl
query=nlwnsck,mango1,4100,ssl
query=nlwnsck,mango1,5000,ssl
```

interfaces ファイルのフォーマットを次に示します。

```
sybsrv2
master tcp ether mango1 5000 ssl
query tcp ether mango1 4100 ssl
query tcp ether mango1 5000 ssl
```

注意： SAP 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で複数のクエリエントリをサポートします。 SAP 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、SAP 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

SAP jConnect では、JNDI (Java Naming and Directory Interface) を使用して接続情報を指定します。

SAP jConnect では、次の情報を指定します。

- サーバに接続するためのホスト名およびポートを指定できる中央の場所。 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が使用する接続プロパティとデフォルトデータベースを指定できる中央の場所。
- 接続試行の失敗を処理するための SAP jConnect の CONNECTION_FAILOVER プロパティ。 CONNECTION_FAILOVER が true に設定されているときは、SAP 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。

SAP 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が存在することと、その必要な情報が `javax.naming.Context` クラスで設定されていることを確認します。

参照：

- JNDI を使用するための接続 URL (42 ページ)
- 必要なディレクトリサービス情報 (42 ページ)
- CONNECTION_FAILOVER プロパティ (44 ページ)
- JNDI コンテキスト情報の提供 (45 ページ)

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。

次に例を示します。

```
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

SAP jConnect とともに JNDI を使用するときに必要なディレクトリサービス情報について説明します。

JNDI は、接続先のデータベースサーバに関する次の情報を返す必要があります。

- 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します。 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名を使って属性を参照することもできます。

表 4 : JNDI 用の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	sybaseVersion	1.3.6.1.4.1.897.4.2.1
サーバ名	sybaseServer	1.3.6.1.4.1.897.4.2.2
サービス	sybaseService	1.3.6.1.4.1.897.4.2.3
ステータス	sybaseStatus	1.3.6.1.4.1.897.4.2.4
(必須) アドレス	sybaseAddress	1.3.6.1.4.1.897.4.2.5
セキュリティメカニズム	sybaseSecurity	1.3.6.1.4.1.897.4.2.6
リトライ回数	sybaseRetryCount	1.3.6.1.4.1.897.4.2.7
ループ遅延	sybaseRetryDelay	1.3.6.1.4.1.897.4.2.8
(必須) jConnect 接続プロトコル	sybaseJconnectProtocol	1.3.6.1.4.1.897.4.2.9
(必須) jConnect 接続プロパティ	sybaseJconnectProperty	1.3.6.1.4.1.897.4.2.10
(必須) 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サーバ名	sybaseHAservername	1.3.6.1.4.1.897.4.2.15
リソースマネージャ名	sybaseResourceManager-Name	1.3.6.1.4.1.897.4.2.16
リソースマネージャタイプ	sybaseResourceManager-Type	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- Interface	1.3.6.1.4.1.897.4.2.18
サーバタイプ	sybaseServerType	1.3.6.1.4.1.897.4.2.19

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。 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 –** 属性の OID を使用します。

```
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 –** 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。

```
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 プロパティ

`CONNECTION_FAILOVER` は、SAP 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 (デフォルト) に設定されている場合、SAP 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、SAP 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、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、SAP 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 が false に設定されている場合、SAP 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 SAP jConnect を使用するには、JNDI の仕様に精通している必要があります。

Oracle Technology Network の JNDI 仕様を参照してください。

特に、JNDI と SAP 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をシステムレベルまたは実行時のどちらかで設定します。

プロパティは次のとおりです。

- Context.INITIAL_CONTEXT_FACTORY - 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 - 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。 INITIAL_CONTEXT_FACTORY コンテキストプロパティは Oracle 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。 Context.PROVIDER_URL 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=Sybase11,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 SAP 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、SAP jConnect は、次の例に示すように `DirContext.getAtributes` 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=" +
     "Sybase11", SYBASE_SERVER_ATTRIBUTES);
```

SYBASE_SERVER_ATTRIBUTES は、SAP jConnect 内で定義された文字列の配列です。 配列の値は、必要なディレクトリサービス情報 (42 ページ) 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

SAP 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。

SAP jConnect を使用して Unicode データを渡す

SAP 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、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。

以下は、Unicode 標準バージョン 2.0 の一部を引用したものです。

> `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。

注意： SAP 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。 ただし、SAP 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。

char 文字データ型および varchar 文字データ型が使用できる場所であれば、構文を変更することなく、unichar データ型および univarchar データ型をどこでも使用できます。

- unichar - *n* を使用して Unicode 文字数を指定します (割り付けられる記憶領域は各文字に対して 2 バイトです)。
- univarchar - *n* を使用して、可変長データ型の文字単位で最大長を指定します。

サーバで unichar と univarchar のデータが使用可能なときは、SAP 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 SAP 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 文字に変換できません。 そのため SAP jConnect は文字列を 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、SAP 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 SAP 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、次の操作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。
2. DISABLE_UNICHAR_SENDING 接続プロパティを false に設定します。

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

参照：

- JCONNECT_VERSION 接続プロパティ (6 ページ)
- 接続プロパティの設定 (10 ページ)

SAP jConnect の文字セットコンバータ

文字セット変換クラスは 2 つあります。 SAP 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、SAP jConnect 16.0 は SAP 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、SAP 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。

参照：

- 文字セット変換のパフォーマンスの向上 (50 ページ)

文字セットコンバータの選択

SAP 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 2.0 または 3.0 の場合、デフォルトは TruncationConverter です。 JCONNECT_VERSION が 4.0 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、SAP 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 4.0 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.charset.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します。

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 2.0 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 3.0 ～ 6.05 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 SAP 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。

SAP 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 (JVM) 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。

文字セット変換の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、SAP jConnect サンプルに含まれている `SunIoConverter` クラスを使用できます。

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。

参照：

- SunIoConverter 文字セット変換 (152 ページ)

サポートされる文字セット

SAP jConnect でサポートされる文字セットと、サポートされる各文字セットに対応する JDK バイトコンバータを示します。

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

SAP 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 5 : サポートされている SAP jConnect の文字セット

SybCharset 名	JDK Byte バイトコンバータ
ascii_7	ASCII
big5	Big5
big5hk (JDK 1.3 以降向け)	Big5_HKSCS
cp037	Cp037
cp437	Cp437
cp500	Cp500
cp850	Cp850
cp852	Cp852
cp855	Cp855
cp857	Cp857
cp860	Cp860

SybCharset 名	JDK Byte バイトコンバータ
cp863	Cp863
cp864	Cp864
cp866	Cp866
cp869	Cp869
cp874	Cp874
cp932	MS932
cp936	GBK
cp949	Cp949
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
is088595	ISO8859_5

SybCharset 名	JDK Byte バイトコンバータ
iso88596	ISO8859_6
iso88597	ISO8859_7
iso88598	ISO8859_8
iso88599	ISO8859_9
iso15	ISO8859_15_FDIS
koi8	KOI8_R
mac	MacRoman
mac_cyr	MacCyrillic
mac_ee	MacCentralEurope
macgreek	MacGreek
macturk	MacTurkish
sjis	MS932
tis620	MS874
ucs2	Unicode
utf8	UTF8

サポートされていない文字セット
次の一部の文字セットは、類似する JDK バイトコンバータが存在しないため、SAP 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、これらの文字セットに対して `TruncationConverter` クラスを使用できます。

デフォルトの文字セットマッピングより優先使用
`JAVA_CHARSET_MAPPING` 接続プロパティは、SAP Adaptive Server のデフォルトの文字セットマッピングより優先して使用します。

- **例 –** サーバ文字セット cp949 を ms949 にマッピングします。

```
props.put("CHARSET", "cp949"); /* Server character set */
props.put("JAVA_CHARSET_MAPPING", "ms949"); /* Java character set
mapping */
```

  SAP Adaptive Server の文字セットのほとんどには、マッピング先の Java 文字セットと同じ名前が付いています。 異なる名前の Java 文字セットにマッピングされる文字セットについては、「サポートされる文字セット (50 ページ)」を参照してください。

ヨーロッパ通貨記号のサポート

SAP jConnect では、ヨーロッパの通貨記号「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、SAP jConnect の文字セット cp1250、cp1251、cp1252、cp1253、cp1254、cp1255、cp1256、cp1257、cp1258、cp874、iso885915、utf8 に含まれています。

ユーロ記号を使用するには、次の点に注意してください。

- `PureConvertor` 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。
- 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。
- 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。

参照：

- SAP jConnect の文字セットコンバータ (48 ページ)
- CHARSET 接続プロパティの設定 (49 ページ)

データベースの問題

SAP jConnect に関するデータベースの問題について説明します。

参照：

- バッチ更新のサポート (73 ページ)
- データ型 (75 ページ)
- フェールオーバのサポート (54 ページ)
- サーバ間のリモートプロシージャコール (58 ページ)
- Adaptive Server でのワイドテーブルのサポート (59 ページ)
- 結果セットでのカーソルの使用 (61 ページ)
- COMPUTE 句での Transact-SQL クエリ (72 ページ)

- データオンリーロックテーブルの可変長ロー (82 ページ)
- ラージオブジェクト (LOB) のサポート (83 ページ)
- ラージオブジェクトのロケータのサポート (83 ページ)
- データベースメタデータへのアクセス (60 ページ)
- 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 (74 ページ)

フェールオーバのサポート

SAP jConnect は、SAP Adaptive Server のフェールオーバをサポートします。

SAP Adaptive Server フェールオーバを使うと、2 つの SAP Adaptive Server をコンパニオンとして設定できます。

注意： 高可用性システムでの SAP Adaptive Server フェールオーバは、接続フェールオーバとは別の機能です。 この 2 つの機能をどちらも使用する場合は、この項を熟読してください。

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、非対称型にも対称型にも設定できます。

- 非対称型の設定では、2 つの SAP 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 SAP Adaptive Server は「ホットスタンバイ」として機能し、フェールオーバ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 SAP Adaptive Server がそれぞれ別のマシン上で稼働します。 しかし、フェールオーバが発生したときは、一方の SAP Adaptive Server がもう一方の SAP 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、SAP 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。 SAP jConnect でフェールオーバを使用できるように設定すると、フェールオーバ可能に設定された SAP 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。

詳細については、Adaptive Server マニュアルの「高可用性システムでのフェールオーバの使用」を参照してください。

フェールオーバ方式の一部として SAP jConnect を使用する場合、次の点に注意してください。

- 2 つの SAP Adaptive Server をフェールオーバ可能に設定します。
- クライアントでフェールオーバが実行された場合、フェールオーバ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。
- `REQUEST_HA_SESSION` jConnect 接続プロパティを true に設定します。
- フェールオーバが発生したときは、SAP jConnect のイベント通知は機能しません。
- 使用しない文はすべて終了させます。 SAP jConnect は、フェールオーバ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。

SAP jConnect でのフェールオーバの実装

SAP jConnect でのフェールオーバのサポートの実装について説明します。

1. 設定:
   - `REQUEST_HA_SESSION` を true に設定します。
   - `SECONDARY_SERVER_HOSTPORT` 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。
2. JNDI を使用してサーバに接続します。 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。
   - 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=haprimary, 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
     ```

- 識別名 (DN) - 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。
  次に例を示します。

```
dn: servername=haprimary, 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)。

- 完全な LDAP URL - 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)。
  次に例を示します。

```
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
```

`REQUEST_HA_SESSION` 接続プロパティを使用すると、クライアントは、フェールオーバ可能に設定された Adaptive Server とのフェールオーバ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、SAP jConnect はフェールオーバ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 SYBASE1 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
```

3. JNDI と LDAP を使用して接続を要求します。
   a) 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
   b) 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);
```

フェールオーバプロセスでは、次のことが可能です。

- **プライマリサーバへのログイン** – SAP Adaptive Server がフェールオーバ可能に設定されていない場合や、フェールオーバ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ません。

```
'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** – フェールオーバが発生すると、SQL 例外 JZ0F2 が発生します。

```
‘SAP Adaptive Server Enterprise 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します。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。
    - プロシージャでアプリケーションを再起動するか、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ります。
- **プライマリサーバへのフェールバック** – システム管理者は、セカンダリサーバで **sp_failback** を発行して、フェールバックのタイミングを決定します。 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。

  フェールバック後、クライアントは、セカンダリサーバへのフェールオーバ時と同じ動作と結果をプライマリサーバでも見ることができます。

参照：

- 接続プロパティ (10 ページ)
- JNDI を使用してサーバに接続する (41 ページ)

サーバ間のリモートプロシージャコール

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、SAP 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。

SAP 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設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、すでに 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後続の接続で使用されます。

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 SAP 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のサポート

SAP Adaptive Server 15.7 ESD #1 では、以前のバージョンのデータベースサーバに比べて、制限やパラメータが増えています。

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 PreparedStatement を実行するパラメータとして、最大 2,048 個のパラメータを送信または取得できます。
- SAP Adaptive Server 15.7 ESD #1 以降に接続されている場合は、PreparedStatement へ最大 32,767 個のパラメータを送信または取得できます。

SAP 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 6.0 以上に設定されている必要があります。

注意： JCONNECT_VERSION を 6.0 より前に設定した場合でも、SAP jConnect は SAP 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ときは、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 SAP Adaptive Server のデータにアクセスするときも、JCONNECT_VERSION を 6.0 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、SAP 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

DatabaseMetaData のメソッドをサポートするために、SAP Adaptive Server 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 SAP jConnect が呼び出す一連のストアドプロシージャを用意しています。

メタデータを返すストアドプロシージャがまだ SAP Adaptive Server サーバにインストールされていない場合は、SAP jConnect に付属している次のストア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_server.sql - 12.0 より前のバージョンの SAP Adaptive Server 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_server12.sql - バージョン 12.0.x の SAP Adaptive Server 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_server12.5.sql - バージョン 12.5.x の SAP Adaptive Server 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_server15.0.sql - SAP Adaptive Server 15.0 から 15.5 にストアドプロシージャをインストールします。
- sql_server15.7.sql - SAP Adaptive Server 15.7 または 15.7 ESD #1 にストアドプロシージャをインストールします。
- sql_server15.7.0.2.sql - SAP Adaptive Server 15.7 ESD #2 以降にストアドプロシージャをインストールします。
- sql_server16.0.sql - SAP Adaptive Server 16.0 にストアドプロシージャをインストールします。
- sql_asa.sql - バージョン 9.x の SAP SQL Anywhere 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_asa10.sql - バージョン 10.x の SAP SQL Anywhere 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` - バージョン 11.x の SAP SQL Anywhere 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_asa12.sql` - バージョン 12.x の SAP SQL Anywhere 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_asa16.sql` - バージョン 16.x の SAP SQL Anywhere 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 SAP 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、『SAP jConnect for JDBC インストールガイド』および『リリースノート SAP 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から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。

結果セットでのカーソルの使用

SAP 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、Javadoc にその説明があります。

サーバが SAP Adaptive Server 15.0 以降である場合、要求があれば SAP 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 6 : SAP jConnect で使用できる java.sql.ResultSet のオプション

同時実行性	タイプ		
	TYPE_FOR-WARD_ ONLY	TYPE_SCROLL_ INSENSITIVE	TYPE_SCROLL_ SENSITIVE
CONCUR_READ_ ONLY	サポートしている	サポートしている	使用不可
CONCUR_UPDAT-ABLE	サポートしている	使用不可	使用不可

参照：

- 位置付け更新と削除のための JDBC 2.0.メソッド (65 ページ)
- カーソルと PreparedStatement オブジェクト (69 ページ)
- SAP jConnect での TYPE_SCROLL_INSENSITIVE 結果セット (70 ページ)
- JDBC 1.x メソッドを使用した位置付け更新と削除 (65 ページ)
- カーソルの作成と使用 (63 ページ)

カーソル

SAP jConnect を使用してカーソルを作成する方法を説明します。

- `SybStatement.setCursorName` - 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` - 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、SAP 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.getCursorName` 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
```

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 だけを呼び出すと、返される ResultSet は TYPE_FORWARD_ONLY と CONCUR_READ_ONLY の SybResultSet になります。
- 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。
- setFetchSize を呼び出すと、executeQuery から返される ResultSet は TYPE_FORWARD_ONLY と CONCUR_READ_ONLY 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

Statement.setCursorName メソッドまたは SybStatement.setFetchSize メソッドを使用して、カーソルを作成します。

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 とおりの方法をそれぞれ使用しています。 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.getCursorName 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

JDBC 1.x を使用するメソッドを確認します。

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

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);
```

位置付け更新と削除のための JDBC 2.0.メソッド

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。

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。

更新された値を使用して、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できます。 これらのメソッドはすべて `SybCursorResultSet` クラスに実装されます。

SAP 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 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(ResultSet.TYPE_SCROLL_INSENSITIV
E,
    ResultSet.CONCUR_UPDATABLE);
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します。

ローを削除するには、`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
JDBC 2.0 API を使用してローを挿入します。

`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();
```

カーソルクローズ時のロックの解放
SAP Adaptive Server 15.7 では、`release_locks_on_close` オプションを含めるように `declare cursor` 構文が拡張されています。このオプションは、カーソルのクローズ時に独立性レベル 2 および 3 でカーソルの共有ロックを解放します。

SAP jConnect は、release-lock-on-close セマンティックをサポートしています。

SAP jConnect 接続を使用するには、`RELEASE_LOCKS_ON_CURSOR_CLOSE` 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。

この設定は、`release_locks_on_close` をサポートしているサーバに接続されている場合にのみ有効です。

`release_locks_on_close` の詳細については、SAP Adaptive Server Enterprise の『リファレンスマニュアル: コマンド』を参照してください。

select for update のサポート
SAP Adaptive Server 15.7 以降は、同じトランザクション内の後続の更新用にローをロックできる **select for update** と、更新可能なカーソル用の排他ロックをサポートしています。

SAP Adaptive Server Enterprise の『Transact-SQL ユーザーズガイド』の「クエリ: テーブルからのデータの選択」を参照してください。

この機能は、`for update` 句が **select** 文に追加されたときと、クライアント内で開いている更新可能なカーソルに追加されたときにクライアントで自動的に使用可能になります。

カーソルと PreparedStatement オブジェクト

PreparedStatement は、入力パラメータに同じ値や異なる値を指定して、何度でも使用できます。

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、次に使用するときに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)。 カーソルの準備文を実行すると、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
```

SAP jConnect での TYPE_SCROLL_INSENSITIVE 結果セット

SAP jConnect では、TYPE_SCROLL_INSENSITIVE 結果セットをサポートしています。

SAP jConnect は、SAP 独自のプロトコルである Tabular Data Stream (TDS) を使用して SAP データベースサーバと通信します。 SAP 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、SAP 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、SAP 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 SAP 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。

SAP 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` インタフェースについて次に説明します。

- JDBC 1.0 `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;
int getFetchDirection() 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;
```

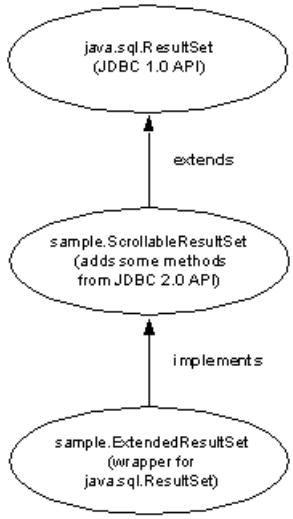
```
// Get a reference to a JDBC 1.0 ResultSet
ResultSet rs = stmt.executeQuery(_query);
```

```
// 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 1 : サンプルクラスと JDBC API の関係を示すクラス図



詳細については、Oracle Technology Network for Java の JDBC 2.0 API に関する情報を参照してください。

COMPUTE 句での Transact-SQL クエリ

SAP 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。 次に例を示します。

```
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)
```

SAP 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、ディテールローの結果セットが 1 つと計算結果の結果セットが 1 つ組み込まれます。 クライアントが返されたローを完全に処理するには、すべての結果セットを処理する必要があります。このように処理しない場合は、最初のデータグループのディテールローのみが、最初に返される結果セットに組み込まれます。

`COMPUTE` 句の詳細については、SAP 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、Oracle Technology Network for Java 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文を 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。

バッチに追加された文は更新カウントのみを返し、`ResultSet` は返しません。

`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。

SAP 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

SAP jConnect は JDBC 2.0 API の仕様に従ってバッチ更新を実装します。

次のような例外があります。

- `EXECUTE_BATCH_PAST_ERRORS` 接続プロパティはバッチ実行での失敗の処理方法を制御します。
  デフォルトでは、`EXECUTE_BATCH_PAST_ERRORS` は false に設定され、SAP jConnect は最初の失敗後に処理を停止します。
  `BatchUpdateException.getUpdateCounts` は長さが M < N の `int[]` アレ

イを返します。これは、バッチ内の最初の M 文は成功、M+1 文は失敗、M +2..N 文は実行されていないことを示します。 “N” はバッチ内の文の総数です。 `EXECUTE_BATCH_PAST_ERRORS` が true に設定されている場合、SAP jConnect は致命的でない失敗が発生した場合にも処理を続行します。 `BatchUpdateException.getUpdateCounts` は長さが N の `int[]` アレイを返します。“N” はバッチ内の文の総数です。 各更新カウントを調べ、各文の実行ステータスを決定します。

- 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。
- SAP Adaptive Server でバッチ実行中に致命的エラーが発生すると、 `BatchUpdateException.getUpdateCounts` は長さ 0 の `int[ ]` だけを返します。 致命的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。
- 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、SAP 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 のドキュメントを参照してください。

参照：

- 非連鎖トランザクションモードでのストアドプロシージャの実行 (150 ページ)

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

SAP 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、結果セットを返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` を実行します。 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}",
     ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

// 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

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 データ型

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、SAP 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、SAP 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、SAP 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、SAP 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。

image データ型

SAP jConnect の TextPointer クラスには、SAP Adaptive Server または SAP 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。

4.0 より前のバージョンの SAP 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。 バージョン 5.0 以降では、TextPointer.sendData メソッドは java.io.InputStream を使用しており、image データを SAP 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) または LOB ローケータのサポートを標準 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

SAP jConnect の TextPointer クラスのパブリックメソッドについて説明します。

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 オブジェクト

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。

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 による更新の実行

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。

サンプルコード:

```
/*
 *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、SAP 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。

TextPointer.sendData による image カラムの更新
TextPointer.sendData を使用して、image データでカラムを更新します。

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。

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。

text データ型
SAP jConnect 3.0 以前のバージョンでは、 TextPointer クラスを sendData メソッドとともに使用して、SAP Adaptive Server または SAP SQL Anywhere データベース内の text カラムを更新します。

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 SAP Adaptive Server または SAP 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 および time データ型

SAP jConnect for JDBC がサポートする SAP Adaptive Server のデータ型は、datetime、smalldatetime、bigdatetime、bigtime、date、および time です。

- datetime は、1753 年の 1 月 1 日から 9999 年 12 月 31 日までの日付を保持します。プラットフォームの能力が対応可能であれば、300 分の 1 秒のレベルまで正確です。
- smalldatetime は、1900 年 1 月 1 日から 2079 年 6 月 6 日までの日付を、分の単位まで正確に保持します。
- bigdatetime は、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 00:00:00.000000 から 9999 年 12 月 31 日の 23:59:59.999999 までです。
- bigtime は、当日の午前 0 時ちょうど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。
- date が保持可能な日付の範囲は 0001 年 1 月 1 日から 9999 年 12 月 31 日までであり、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 データ型との間には直接マッピングが存在します。
- time が保持可能な時間の範囲は 00:00:00:000 から 23:59:59:990 です。java.sql.Time と time データ型との間には直接マッピングが存在します。

Date、Time、Datetime、Smalldatetime データ型

SAP jConnect for JDBC は、date、time、datetime、smalldatetime データ型をサポートします。

date カラムまたは time カラムを持つテーブルからの選択を行うときに、SAP 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。
- SAP 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 SAP 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。 SAP jConnect を使用して SAP 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。

- SAP SQL Anywhere での datetime カラムの最大値は 7911 年 1 月 1 日 00:00:00 です。
- SAP 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と、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、SAP 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情報を参照してください。
- SAP 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。

bigdatetime および bigtime データ型

SAP Adaptive Server 15.5 以降への接続時に、SAP jConnect 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 SAP 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、SAP Adaptive Server が、SAP Adaptive Server カラムに合わせるために、SAP jConnect 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。
- SAP Adaptive Server 15.0.x 以前のバージョンへの接続時には、SAP jConnect for JDBC は datetime および time データ型を使用してデータを転送します。

Char、Varchar、Text、GetByte データ型

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.getByte を使用しないでください。

サポートされているその他のデータ型

SAP jConnect でサポートされている SAP Adaptive Server のその他のデータ型について説明します。

SAP jConnect は、SAP Adaptive Server の次の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

`bigint` は、ネイティブな SAP Adaptive Server データ型としてサポートされる 64 ビット integer データ型です。

`bigint` は、Java データ型の `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)` を呼び出します。これにより SAP jConnect がデータを `bigint` として SAP Adaptive Server に送信します。`bigint` カラムから取得する場合は、`ResutSet.getLong(int index)` メソッドを使用します。

unitext データ型

SAP jConnect は、`unitext` カラムが使用されている場合の SAP Adaptive Server へのデータの格納および SAP Adaptive Server からのデータの取得を、内部的に処理します。

unsigned int データ型

SAP Adaptive Server では、`unsigned bigint`、`unsigned int`、`unsigned smallint` をネイティブ SAP 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する場合は、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` します。

SAP Adaptive Server データ型	Java データ型
`unsigned smallint`	`setInt()`、`getInt()`
`unsigned int`	`setLong()`、`getLong()`
`unsigned bigint`	`setBigDecimal()`、`getBigDecimal()`

データオンリーロックテーブルの可変長ロー

16K 論理ページサイズを使用するように設定されている SAP Adaptive Server 15.7 より前のバージョンでは、可変長カラムが行の先頭から 8191 バイトを超えた位置か

ら始まっている場合、可変長ローを含むデータオンリーロック (DOL) テーブルは作成できませんでした。

この制限は、SAP Adaptive Server 15.7 以降では取り除かれています。 SAP Adaptive Server Enterprise の『パフォーマンス＆チューニングシリーズ: 物理データベースのチューニング』の「データの格納」を参照してください。

JDBC クライアントがこの機能を使用するにあたって特別な設定を行う必要はありません。 長い DOL ローを受信するように設定されている SAP Adaptive Server バージョン 15.7 に接続すると、これらのクライアントは長いオフセットを使用して自動的にレコードを挿入します。 クライアントが長い DOL ローを以前のバージョンの SAP Adaptive Server に接続しようとするか、長い DOL ローオプションが無効になっている 15.7 SAP Adaptive Server に接続しようとすると、エラーメッセージが送信されます。

ラージオブジェクト (LOB) のサポート

SAP jConnect は、ラージオブジェクト (LOB) データ型の `text`、`unitext`、および `image` の次の使用法をサポートします。

- ロー内格納の LOB カラム - SAP Adaptive Server では、ロー内に格納するようにマークされている LOB カラムは、ロー全体を格納するのに十分なメモリがある場合、ロー内に格納されます。 ロー内のカラムが更新されたために、ローのサイズが定義されている制限を超えた場合、ロー内に格納されている LOB カラムは制限を満たすためにロー外に移動されます。 SAP Adaptive Server Enterprise の『Transact-SQL ユーザーズガイド』の「ロー内/ロー外の LOB」を参照してください。
  SAP jConnect のバルク挿入ルーチンは、SAP Adaptive Server の `text`、`image`、`unitext` の LOB カラムのロー内およびロー外の記憶領域をサポートしています。 以前のクライアントバージョンからのバルク挿入ルーチンでは、LOB カラムは常にロー外に格納されます。
- ストアドプロシージャのパラメータとしての LOB オブジェクト - SAP jConnect は、ストアドプロシージャ内での入力パラメータ、およびパラメータマーカデータ型としての `text`、`unitext`、`image` の使用をサポートしています。

ラージオブジェクトのロケータのサポート

SAP jConnect はラージオブジェクト (LOB) ロケータをサポートしています。 LOB ロケータには、データ自体ではなく、LOB データへの論理ポインタが含まれているため、SAP Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。

LOB ロケータに対するサーバのサポートは、SAP Adaptive Server 15.7 に導入されています。

SAP jConnect では、LOB ロケータをサポートする SAP Adaptive Server に接続しており、`autocommit` がオフになっている場合に、サーバ側のロケータを使用して LOB データにアクセスします。 それ以外の場合、jConnect はクライアント側で LOB データを実体化します。 クライアント側で実体化された LOB データを完全な LOB API で使用することはできますが、データ量が大きくなるため、LOB ロケータを使用した場合よりも API のパフォーマンスが低下することがあります。

注意： LOB ロケータを使用している場合、各ローに LOB データを含む大きな結果セットを取得すると、アプリケーションのパフォーマンスに影響が及ぶ場合があります。 SAP Adaptive Server では LOB ロケータを結果セットの一部として返します。LOB データを取得するには、SAP jConnect が残りの結果セットをキャッシュに格納する必要があります。 結果セットは小さいサイズを保持するか、カーソルのサポートを有効にしてキャッシュに格納するデータのサイズを制限することをおすすめします。

LOB ロケータのサポートを有効にするには、`ENABLE_LOB_LOCATORS` 接続プロパティを true に設定して SAP Adaptive Server への接続を確立します。 有効にすると、クライアントアプリケーションは `java.sql` パッケージの `Blob`、`Clob`、`NClob` クラスを使用して、ロケータにアクセスできるようになります。

注意： LOB ロケータと `autocommit` の両方が有効になっている場合、SAP jConnect では SAP Adaptive Server での LOB ロケータのサポートが可能であっても、クライアント側でマテリアライズされている LOB ロケータに自動的に切り替えます。 これにより、クライアントが使用するメモリが増加し、パフォーマンスが低下する場合があります。 したがって、`autocommit off` の状態で LOB ロケータを使用することをおすすめします。

`Blob`、`Clob`、`NClob` クラスの詳細については、Java のマニュアルを参照してください。

SAP jConnect の拡張機能

SAP jConnect には、イベント通知、エラーメッセージ処理、パスワード暗号化、動的クラスロード、JDBC 仕様のサポートなどの拡張機能を備えています。

SAP jConnect でサポートされている拡張機能を使用するための手順を説明します。

参照：

- BCP 挿入 (85 ページ)

- サポートされている SAP Adaptive Server クラスタエディションの機能 (86 ページ)
- イベント通知 (88 ページ)
- エラーメッセージ (90 ページ)
- パスワードの暗号化 (95 ページ)
- JDBC 4.0 仕様のサポート (106 ページ)
- 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 (97 ページ)
- 動的クラスロード (102 ページ)
- JDBC 3.0 仕様のサポート (106 ページ)
- JDBC 2.0 オプションパッケージ拡張機能のサポート (108 ページ)

BCP 挿入

SAP jConnect では、SAP 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。

この機能では特別なサーバ設定は必要ありませんが、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。

また、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。

バルクロード挿入を有効にするには、`ENABLE_BULK_LOAD` を次のいずれかの値に設定します。

- 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します。
- ARRAYINSERT - ローレベルのロギングでバルクロードを有効にしますが、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ません。
- BCP - ページレベルのロギングでバルクロードを有効にします。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ません。
- LOG_BCP - SAP Adaptive Server 高速ログ BCP 機能を使用してページレベルのロギングでバルクロードを有効にします。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ません。

準備文を使用して `ENABLE_BULK_LOAD` を有効な値に設定すると、SAP jConnect は BULK ルーチンを使用して、レコードのバッチを SAP データベースに挿入します。

バルクロードを有効にする際の制限
ENABLE_BULK_LOAD には、次のような制限があります。

- 選択されたテーブルでは、トリガは無視されます。
- null と参照制約は検証されません。
- 計算カラムと暗号化カラムはサポートされていません。
- 重複した identity 値の範囲を指定すると、重複した identity 値が生成される可能性があります。
- 同時データ入力が行われると、重複した IDENTITY 値が挿入される可能性があります。

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

SAP jConnect は SAP 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 SAP 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、SAP 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、SAP Adaptive Server Enterprise の『Cluster ユーザーズガイド』を参照してください。

ログインリダイレクト

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。

クラスタ環境では一般に、常にサーバ間で処理負荷の不均衡が発生しています。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を使用すると、クラスタ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、クラスタ環境では、最適なリ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

接続フェールオーバ機能を使用すると、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 SAP 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できます。

高可用性に対応したシステムでは、フェールオーバ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。 SAP 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、フェールオーバ時にフェールオーバ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。

注意： 接続プロパティ **DEFAULT_QUERY_TIMEOUT** および **INTERNAL_QUERY_TIMEOUT**、または **DriverManager.setLoginTimeout(xx)** は、フェールオーバ発生後に、障害が発生したノードから可用性の高いノードへの切り替えを行う、極めて重要な役割を担います。

接続のフェールオーバの有効化

接続文字列を使用して接続フェールオーバを有効にするには、`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、順序付けされたフェールオーバリストです。

SAP jConnect はフェールオーバ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。 この新しいリストで、指定されたリストが上書きされます。

イベント通知

イベント通知を使用すると、SAP 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。

```
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。
- *eventHdler* 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 SAP Open Server 上で発生するたびに、SAP 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、SAP 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構文は次のとおりです。

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

警告！ SAP jConnect 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応答を停止することがあります。

イベント通知の例

イベントハンドラを実装し、接続の確立後にイベントハンドラのインスタンスにイベントを登録する方法について説明します。

イベント通知のサンプルコード

```
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

SAP jConnect には、SAP jConnect 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 SAP 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

SAP 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` オブジェクトで転送されます。

次のコードはこの処理を示します。

```
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 の確認と処理を行うコードが結果セットに依存しないようにします。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 (`rs.next( )` が false) 場合でも SQLWarning を確認します。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
```

SAP jConnect 固有のエラー情報の取得

SAP jConnect の `EedInfo` インタフェースに、SAP jConnect 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 ロー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` - SAP 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する、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。
- `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、`SybSQLException` 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 `SybSQLException` 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して、サーバによって生成されたエラーメッセージを SAP jConnect が処理する方法をカスタマイズします。

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 - エラー処理ロジックを、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。SAP jConnect によって生成されたメッセージは処理しません。

SAP 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、SAP 例外を引数として受け取る messageHandler メソッドを呼び出します。 次に 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、SAP 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 SAP 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 を使用します。

エラーメッセージハンドラの例

SAP jConnect でのエラーメッセージハンドラの例を次に示します。

```
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").newIn
stance();
               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();
}
```

パスワードの暗号化

SAP 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 SAP Adaptive Server に送信して認証を求めます。

ただし、SAP 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化され、サーバで非対称暗号化がサポートされる場合は、非対称暗号化形式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 (SAP Adaptive Server 15.0.2 以降など) 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の有効化

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 接続プロパティを true に設定します。

ENCRYPT_PASSWORD プロパティと RETRY_WITH_NO_ENCRYPTION プロパティの両方を 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/ECB/OAEPWithSHA1AndMGF1Padding” または “RSA/NONE/OAEPWithSHA1AndMGF1Padding” 変形をサポートしている必要があります。 SAP jConnect には FIPS 140-2 認定 JCE プロバイダが付属していて、デフォルトで使用されます。

JCE_PROVIDER_CLASS 接続プロパティを使用すると、代替の 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は、バンドルされたデフォルトの FIPS 140-2 認定プロバイダが使用されます。

デフォルトの JCE プロバイダを使用したパスワード暗号化の実行

JCE_PROVIDER_CLASS が指定されていない場合、SAP jConnect はバンドルされたデフォルトの FIPS 140-2 認定 JCE プロバイダを使用して RSA パスワード暗号化を実行します。

カスタム JCE プロバイダの指定

SAP jConnect でカスタム JCE プロバイダを指定します。

1. 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 connnection properties as needed */
props.put("user", "xyz");
props.put("password", "123");

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

2. JCE プロバイダを設定します。

   次のいずれかを行います。
   - 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 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、『JCE リファレンスガイド』で、外部 JCE プロバイダの設定方法を参照してください。

   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。

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

データベース製品では、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。

SAP 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、SAP 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、SAP 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。

SAP Adaptive Server バージョン 12.0 以降および SAP 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 SAP jConnect for JDBC』を参照してください。

参照：
- 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 (98 ページ)
- データベースからの Java オブジェクトの受信 (100 ページ)
- JCONNECT_VERSION 接続プロパティ (6 ページ)
- データベースへの Java オブジェクトの送信 (98 ページ)

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します。

- 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りません。これは SAP 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す。
- 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りません。または `DynamicClassLoader` (DCL) を使用しなければなりません。DCL によって SAP SQL Anywhere または SAP 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ます。
- 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りません。

参照：
- 動的クラスロード (102 ページ)

データベースへの Java オブジェクトの送信

データベースに Java オブジェクトを送信するには、`setObject` メソッドを使用します。

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、ターゲット `sql.Type` として `java.sql.Types.JAVA_OBJECT` を使用できます。また、`java.sql.Types.OTHER` を使用することもできます。

次の例では、`Address` クラスを定義し、データ型が `Address` クラスである `Address` カラムを持つ `Friends` テーブルの定義を表示して、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、オブジェクトを取得します。

```
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、`CallableStatement` インタフェース内の次の `getObject` メソッドを使用して、オブジェクトを取得します。

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

次の例は、`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る方法を示しています。 この例では、`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、SAP 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

SAP SQL Anywhere および SAP Adaptive Server では、Java クラスを指定できます。

- 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

SAP 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、SAP SQL Anywhere または SAP Adaptive Server サーバからクラスを直接ロードし、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。

SAP jConnect 6.0 以前のバージョンでは、SAP 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、SAP jConnect アプリケーションがローカルの CLASSPATH に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。

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。 SAP 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 SAP 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

`CLASS_LOADER` 接続プロパティを使用すると、複数の接続間で 1 つのクラスローダを共有する便利なメカニズムを利用できます。

1. クラスローダを作成して設定します。

   SAP 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. `CLASS_LOADER` 接続プロパティを使用して、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 stmnt = conn.createStatement();

// Retrieve some rows from the table that has a Java class
// as one of its fields.

ResultSet rs = stmnt.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.

      ...

}
```

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で、SAP 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

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。

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。

`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

SAP 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と、.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： SAP 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、SAP 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、SAP 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。

動的クラスロードの追加機能

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 から継承したパブリックメソッドを使用できます。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。

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 にある Javadoc 情報を参照してください。

参照：

- エラーメッセージ (90 ページ)

JDBC 4.0 仕様のサポート

SAP jConnect でサポートされている JDBC 4.0 仕様のいくつかを示します。

- 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 仕様については、Oracle Technology Network for Java を参照してください。

JDBC 3.0 仕様のサポート

SAP jConnect 16.0 でサポートされている JDBC 3.0 仕様を示します。

セーブポイントのサポート

`Savepoint` インタフェースを使用できます。このインタフェースには、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 3.0 の `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 には、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 `Connection.setSavepoint` が組み込まれています。 `Connection.rollback` 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ロードされます。
- **セーブポイントの解放** – `Connection.releaseSavepoint` 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と、SQLException 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とき、またはトランザクション全体がロールバックされるとき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。

パラメータメタデータの取得

JDBC 3.0 の `ParameterMetaData` インタフェースは、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` の最新のメソッドをサポートします。

自動生成されたキーの取得

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カラムから取得する共通の必要性に対応しています。

自動生成キーを取得するには、`Statement.execute()` メソッドの 2 番目のパラメータとして定数 `Statement.RETURN_GENERATED_KEYS` を渡します。

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、取得したキーごとにローが 1 つずつ含まれます。

注意： SAP 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、自動生成キーの取得の詳細については、Oracle Java の Web サイトで「retrieving automatically generated keys」を検索してください。

オープンした複数の ResultSet オブジェクト

JDBC 3.0 には `getMoreResults(int)` が組み込まれています。このメソッドでは、`Statement` オブジェクトによって返される `ResultSet` オブジェクトを、その後の `ResultSet` オブジェクトが返される前にクローズする必要があ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、`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 2.0 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()` 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()` を呼び出しても、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ます。

`CallableStatement` インタフェースを使用して、パラメータのインデックスではなく、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

保持可能なカーソルまたは結果は、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ません。 `ResultSet` オブジェクトの保持可能性を指定する必要があります。

JDBC 3.0 では、カーソルの保持可能性を指定する機能がサポートされています。 `createStatement()`、`prepareStatement()`、または `prepareCall()` メソッドを使用して文を準備するときには、`ResultSet` の保持可能性を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、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、通常は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機能のサポート

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』) では、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。

SAP 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。

- 命名規則用 JNDI - SAP jConnect がサポートするすべての SAP DBMS で動作します。
- 接続プール - SAP jConnect がサポートするすべての SAP DBMS で動作します。
- 分散トランザクション管理のサポート - SAP Adaptive Server のみで動作します。

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

JNDI によるデータベースの命名について説明します。

参考書

『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)

関連インタフェース

関連インタフェースは、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。

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

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。

SAP による `DataSource` オブジェクトの実装は `com.sybase.jdbcx.SybDataSource` です (詳細については Javadoc 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。

SAP 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されます。

すべてのオプションをコード例とともに紹介します。 詳細については、『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)、および Oracle Java の Web サイトにある JNDI のマニュアルを参照してください。

管理者による設定: LDAP

LDAP 接続は SAP 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。

次に例を示します。

```
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
```

クライアントによるアクセス

JDBC クライアントアプリケーションを使用すると、DriverManager にアクセスして JDBC の URL を指定する代わりに、サーバ名にアクセスして DataSource オブジェクトへの参照を取得できます。

これは一般的な JDBC クライアントアプリケーションです。 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。 このコードは汎用的なもので、SAP jConnect を参照するのは、オブジェクト

ファクトリプロパティの設定時だけです。この設定は、環境設定の一部として実行できます。

SAP 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、サンプルを実行するには、環境を設定し、サンプルを適宜編集する必要があります。 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、ブラウザプロパティの一部として、または次を使用して Java が設定されたときにプロパティが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。

プログラムでの設定

プログラムでの設定を行う目的は、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する場合など)、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。 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。 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 = newProperties()
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。

クライアントによる DataSource オブジェクトの取得
クライアントが DataSource オブジェクトを取得するには、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

SAP jConnect の接続プールについて説明します。

参考書
『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』) を参照してください。

関連インタフェース
JDBC 内の関連インタフェースを確認してください。

- 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が作成されます。 しかし、Web ベース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しなければならない場合があります。

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、接続プールを使用する方法があります。接続プールは、複数のユーザ要求間で接続を共有し、データベース接続をオープンのままにして接続を管理することにより、パフォーマンスを確保し、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で、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生じ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。

サードパーティが接続プールを実装できるように、jConnect の実装には `ConnectionPoolDataSource` インタフェースがあります。このインタフェースは、`DataSource` インタフェースが `Connections` を生成するのと同様に `PooledConnections` を生成します。プールの実装は、`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、クライアントが何らかの方法でデータベース接続を切断した場合にも `ConnectionEventListener` インタフェースを介して通知を受け取るので、その接続をプールから削除できます。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
LDIF エントリに追加の行を入力して、LDAP を設定します。

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
```

参照：
- JNDI によるデータベースの命名 (109 ページ)
- 管理者による設定: LDAP (110 ページ)

中間層クライアントによるアクセス
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` オブジェクトが取得されます。

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。 クライアントによるアクセスと中間層クライアントによるアクセスの基本的な違いは、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
SAP 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。 この機能は大規模多層環境で使用するように設計されています。

参考書
『JDBC 2.0 Optional Package』(旧『JDBC 2.0 Standard Extension API』)

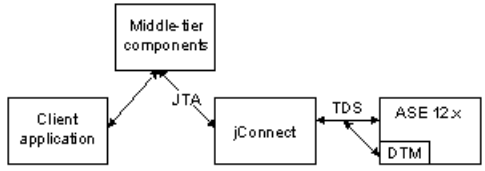
関連インタフェース
JDBC 内の関連インタフェースを確認してください。

- `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
分散トランザクション管理のサポートを有効にするには、`dtm_tm_role` を使用します。

- SAP jConnect は SAP 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必要があり、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。『SAP jConnect for JDBC インストールガイド』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 : 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
LDIF エントリに追加の行を入力して、LDAP を設定します。

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
```

参照：

- JNDI によるデータベースの命名 (109 ページ)
- 管理者による設定: LDAP (110 ページ)

中間層クライアントによるアクセス
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` オブジェクトが取得されます。

次に例を示します。

```
...
XADataSoruce xads = (XADatasource) ctx.lookup ("server=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

SAP jConnect の JDBC の実装は、JDBC 標準とは異なります。

参照：

- マルチスレッドのための調整 (119 ページ)
- サポートされない JDBC 4.0 仕様要件 (117 ページ)
- Connection.isClosed と IS_CLOSED_TEST の使用 (117 ページ)
- 未処理の結果がある場合の Statement.close (118 ページ)
- ResultSet.getCursorName (119 ページ)
- ストアドプロシージャの実行 (120 ページ)

サポートされない JDBC 4.0 仕様要件

このリリースでサポートされていない JDBC 4.0 文を確認してください。

- java.sql.RowID
- JDBC 4.0 で導入された XML API

Connection.isClosed と IS_CLOSED_TEST の使用

SAP jConnect における `isClosed` メソッドのデフォルトの解釈は、JDBC 4.0 仕様で定義されている動作とは異なります。

`Connection.isClosed` を呼び出すと、SAP jConnect は、この接続で `Connection.close` が呼び出されているかどうかを確認します。 `close` が呼び出されている場合は、`isClosed` に true を返します。 `Connection.close` が呼び出されていない場合は、データベースに対して **sp_mda** を実行します。 **sp_mda** は、標準メタデータの一部であり、SAP 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。

JDBC 4.0 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どう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。

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 SAP 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、SAP 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、SAP 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。

SAP jConnect が isClosed() に関して JDBC 標準の動作を厳密に順守するように強制するには、IS_CLOSED_TEST 接続プロパティを特殊値 “INTERNAL” に設定します。 INTERNAL に設定すると、Connection.close が呼び出されたとき、または接続を無効にした IOException を SAP jConnect が検出したときにのみ、isClosed に true が返されます。

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

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 ResultSets) 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が明確に規定されていません。

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、SAP 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、Java SE のマニュアルにある java.sql.* パッケージの Javadoc 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、

SAP 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。 未処理の結果がまだあるときに Statement をクローズしようとすると、SQLException が発生します。

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。 SAP jConnect のプログラマは、次に示すガイドラインを順守する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。

マルチスレッドのための調整

同一の Statement インスタンス、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。SAP jConnect はこの処理を自動的には行いません。

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、一方のスレッドがクエリの送信を、もう一方のスレッドが警告の処理を行う場合、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.getCursorName

JDBC ドライバは、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します。 ただし、ResultSet.getCursorName が呼び出されても SAP jConnect は名前を返しません。

このような状況が発生するのは、次のいずれかに該当する場合です。

- 対応する文に対して `setFetchSize` または `setCursorName` を呼び出した。
- `SELECT_OPENS_CURSOR` 接続プロパティを true に設定して、SELECT...FOR UPDATE 形式のクエリを実行した。 次に例を示します。

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

対応する文に対して `setFetchSize` または `setCursorName` を呼び出していない場合や、`SELECT_OPENS_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。

SAP 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」を参照してください。

参照：

- 結果セットでのカーソルの使用 (61 ページ)

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値の両方を使用する場合よりもパフォーマンスが向上します。

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。

次の例では、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、一方のパラメータ値がリテラル値として指定され、もう一方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 変数から入力パラメータを直接渡します。

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

- SAP jConnect バージョン 6.05 以降では、次のように 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();
```

セキュリティ

クライアント/サーバ通信を保護するためのオプションとして、SAP jConnect は SSL (Secure Sockets Layer) と Kerberos を備えています。

- SSL - 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場合に使用します。
- Kerberos - 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、SAP Adaptive Server に対して Java アプリケーションまたは Java アプリケーションユーザを認証する場合に使用します。 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 SAP 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すが、SAP 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、SAP Adaptive Server で使用されます。現在のところ、SAP 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。

SAP jConnect で SSL や Kerberos を使用する前に、SSL と Kerberos の関連マニュアルに目を通しておくことをおすすめします。 設定に関する情報は、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 SAP Adaptive Server Enterprise の設定方法の詳細については、「関連マニュアル (140 ページ)」を参照してください。 また、『リリースノート SAP jConnect for JDBC』で紹介されている、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。

カスタム SSL ソケットプラグインの実装

クライアントとサーバの間の通信をカスタマイズするには、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ます。

`javax.net.ssl.SSLSocket` は、暗号化を有効にするためにカスタマイズできるソケットの一例です。

`com.sybase.jdbcx.SybSocketFactory` は、SAP jConnect 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。 SAP 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。

SAP jConnect は、以降の入出力オペレーションにこの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。

`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

SAP jConnect でカスタムソケットを使用するための手順を説明します。

1. com.sybase.jdbcx.SybSocketFactory を実装する Java クラスを指定します。
2. ソケットを取得するための独自の実装を SAP jConnect が使用できるように、SYBSOCKET_FACTORY 接続プロパティを設定します。

   SAP jConnect でカスタムソケットを使用するには、SYBSOCKET_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)。

参照：
- 接続プロパティ (10 ページ)
- カスタムソケットの作成と設定 (125 ページ)

カスタムソケットの作成と設定

SAP jConnect がソケットを取得する前に、SSL ソケットのインスタンスを作成し、ソケットを設定できます。

SAP jConnect は、ソケットを使用してサーバに接続します。

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。 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 に対するものであり、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.setEnableCipherSuites 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。

- SAP 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 cither 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
```

```
  // properites
  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、次のサンプルを参照してください。

- `EncryptASE.java` - SAP 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ります。このサンプルは、SAP 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 SAP 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ります。これは、`SybSocketFactory` インタフェースのサンプル実装で、アプリケーションにプラグインして使用できます。

SAP jConnect での SSL サポート

15.7 SP 100 以前のバージョンの SAP jConnect で SSL ソケットを使用するには、[SybSocketFactory] インタフェースの実装を作成し、**SYBSOCKET_FACTORY** 接続プロパティを設定してその実装を使用する必要があります。

バージョン 15.7 SP 100 から、SAP jConnect には、SSL ソケットを使用して SAP Adaptive Server に接続するためのサポートが組み込まれています。 新しい接続プロパティ **ENABLE_SSL** の設定に応じた動作は次のとおりです。

- false - (デフォルト) SAP jConnect では SSL ソケットは使用されません。
- true - SAP jConnect で SSL ソケットが使用され、対象の SAP Adaptive Server で SSL ソケット接続が有効になっている必要があります。 SAP jConnect では、**SYBSOCKET_FACTORY** 接続プロパティは無視されます。

注意： **DriverManager.setLoginTimeout** プロパティを使用してログインタイムアウトを設定して、SSL が有効になっていない SAP Adaptive Server で SSL 接続を試行中に、接続のタイムアウトを許可することをおすすめします。

SSL ソケット機能は、次の標準の Java プロパティに依存します。

- **javax.net.ssl.keyStore**
- **javax.net.ssl.keyStorePassword**
- **javax.net.ssl.trustStore**
- **javax.net.ssl.trustStorePassword**
- **javax.net.ssl.trustStore**
- **javax.net.ssl.trustStoreType**

Java の標準プロパティの詳細については、Java J2SE 6 のドキュメントを参照してください。

Kerberos

Kerberos は、クライアント/サーバアプリケーションの認証に暗号化を使用するネットワーク認証プロトコルです。

Kerberos には、ユーザとシステム管理者にとって次のような利点があります。

- 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。

SAP jConnect で使用するための Kerberos の設定

Kerberos セキュリティメカニズムを使用するために SAP jConnect を設定する手順を説明します。

前提条件

SAP jConnect で使用するために Kerberos を設定するには、次のような前提条件があります。

- JDK 6 以降
- 次のいずれかの Java GSS (Generic Security Services) Manager
  - デフォルトの GSS Manager (JDK に含まれる)
  - Wedgetail JCSI Kerberos バージョン 2.6 以降
  - CyberSafe TrustBroker Application Security Runtime Library バージョン 3.1.0 以降
  - 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 (Key Distribution Center)。

手順

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、SAP 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。 このプロパティの値を設定しなかった場合は、クライアントマシンのホスト名が使用されます。

3. GSSMANAGER_CLASS プロパティを設定します (省略可能)。

   REQUEST_KERBEROS_SESSION と SERVICE_PRINCIPAL_NAME の詳細については、「SAP jConnect 接続プロパティ (11 ページ)」を参照してください。

参照：

- GSSMANAGER_CLASS 接続プロパティ (130 ページ)
- プログラミング情報 (5 ページ)

GSSMANAGER_CLASS 接続プロパティ

Kerberos を使用するとき、SAP jConnect は、GSS (Generic Security Services) API を実装するいくつかの 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。

SAP jConnect は、`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` の値が `GSSManager` クラスオブジェクトでも文字列でもないものに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。

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、設定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 プロパティの設定

ベンダ実装の GSSManager を SAP 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 プロパティの値を、GSSManager オブジェクトの完全修飾クラス名を指定する文字列として指定します。 SAP 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、SAP 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。

GSSManager の例

要件に合わせて GSSManager のインスタンスを作成する方法、または GSSMANAGER_CLASS 接続プロパティが完全修飾クラス名に設定されている場合に SAP jConnect で GSSManager オブジェクトが作成されるようにする方法について説明します。

GSSManager のインスタンスの作成

GSSManager のインスタンスを作成し、GSSMANAGER_CLASS プロパティに渡します。

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。

```
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。

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

3. GSSMANAGER_CLASS を含むこれらの接続プロパティを接続に使用します。

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

GSSMANAGER_CLASS に文字列を渡す

アプリケーション内で GSSMANAGER_CLASS に文字列を渡します。

1. 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。 次に例を示します。

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

2. GSSMANAGER_CLASS 接続プロパティに文字列を渡します。 次に例を示します。

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

3. GSSMANAGER_CLASS を含むこれらの接続プロパティを接続に使用します。 次に例を示します。

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

Kerberos 環境

3 種類の Kerberos 実装環境で SAP jConnect を使用できます。

- CyberSafe
- MIT
- Microsoft Active Directory

Kerberos ホワイトペーパーを参照してください。

CyberSafe

SAP jConnect の CyberSafe Kerberos 実装について説明します。

- **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 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。

  CyberSafe Kerberos は、krb5.conf 設定ファイルを使用しません。 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 のマニュアル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。 Java SE のマニュアルまたは MIT のマニュアルの指定に従って、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。URL は『リリースノート SAP jConnect for JDBC』に記載されています。
- **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。

CyberSafe または MIT KDC を使用する場合、クライアントは krb5.conf ファイルを使用します。 次に例を示します。

```
# 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
 }
```

MIT

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

Microsoft Active Directory サーバ内の Kerberos 用コンポーネントについて説明します。

- **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 (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。

- **クライアントマシン** – SSO 環境の設定に Java リファレンス実装を使用するように、Windows レジストリを変更します。

  Windows レジストリを変更するには、Microsoft サポートサイトの説明を参照してください。
- **設定ファイル** – Windows では、Kerberos 設定ファイルの名前は `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、『リリースノート SAP jConnect for JDBC』で紹介されている Kerberos ホワイトペーパー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 を参照してください。

Active Directory を KDC として使用する場合、クライアントは `krb5.conf` ファイルを使用します。 次に例を示します。

```
# 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
 }
```

DES 暗号化の設定

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。

1. Active Directory 内で、特定のユーザプリンシパル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。
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。

サンプルアプリケーション

`jConnect-16_0/sample2` ディレクトリにある 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
- `ConnectKerberosJAAS.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 の実行

`ConnectKerberos.java` ファイルのサンプルアプリケーションを実行する手順を説明します。

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 クレデンシャルを確立できます。
- 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： 通常、標準の JDK で提供される GSSManager プロバイダの実装では DES_CBC_MD5 と DES_CBC_CRC の暗号タイプのみ使用可能です。 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 - 提供された JRE、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 `/tmp/krb5cc_{user_id_number}` にクレデンシャルが配置されます。*{user_id_number}* はユーザ名にユニークな値です。

   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、この情報を設定ファイル `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 設定ファイルでは、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、設定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 設定ファイルを使用します。

   Kerberos 設定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 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 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ことと、次に示す SQL を実行することが表示されます。

   ```
   select 1
   ```

   - Kerberos 設定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-16_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
   ```

相互運用性

SAP jConnect は、相互運用可能な KDC、GSS ライブラリ、プラットフォームの組み合わせをサポートしており、これらの組み合わせについて SAP Adaptive Server Enterprise への接続を正しく確立できることが検証されています。

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。最新の状況は jConnect for JDBC Web サイトで確認できます。

表 7 : 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]	SAP 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
Windows 2000	Active Directory	Java GSS	CyberSafe	Solaris 8
Windows XP	Active Directory	Java GSS[f]	CyberSafe	Solaris 8

a. SAP Adaptive Server Enterprise の GSS 機能用に使用されるライブラリ。

b. この表の Solaris 8 プラットフォームはすべて 32 ビット版である。

c. この表の Active Directory に関する項目はすべて、Windows 2000 上で稼働する Active Directory サーバを指している。 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る。

d. Wedgetail JCSI Kerberos 2.6 を使用した。 暗号化タイプは 3DES。

e. Wedgetail JCSI Kerberos 2.6 を使用した。 暗号化タイプは DES。

f. Java 1.4.x では、クライアントが System.setProperty("os.name", "Windows 2000"); を使用していなければ 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ある。

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、SAP 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

標準 JRE 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、それぞれのマニュアルを参照してください。

Kerberos のトラブルシューティング

Kerberos のセキュリティに関する問題をトラブルシューティングする際の考慮事項を説明します。

- 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 ログイン設定ファイル内の `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 ログイン設定ファイルには、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 ログイン設定ファイル
  - クラス Krb5LoginModule
  - JGSS のトラブルシューティング

関連マニュアル

Kerberos のセキュリティに関する詳細情報を確認してください。

- JAAS および Java GSS API に関する Java チュートリアル
- MIT Kerberos ドキュメントおよびダウンロードサイト
- CyberSafe Limited
- Windows と Kerberos の相互運用性に関する CyberSafe Limited のドキュメント
- Kerberos RFC 1510

トラブルシューティング

SAP 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。

SAP jConnect でのデバッグ

SAP 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。

SAP 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、SAP jConnect インストールディレクトリの下の 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 SAP 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

Debug インタフェースをインポートし、Debug クラスのインスタンスを取得するには、SybDriver クラスの getDebug メソッドを呼び出します。

```
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

アプリケーション内でのデバッグをオンにするには、Debug オブジェクトの debug メソッドを使用します。

次の呼び出しを追加します。

```
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。 次に例を示します。

```
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 SAP 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。 次に例を示します。

```
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。 次に例を示します。

```
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、`DriverManager.setLogStream` で指定したファイルにデバッグ出力が出力されます。

アプリケーションのデバッグをオフにする方法

デバッグメソッドをオフにする方法を説明します。

次の呼び出しを追加します。

```
sybdebug.debug(false);
```

デバッグのための **CLASSPATH** の設定

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 SAP jConnect `jconn4.jar` ファイルをデバッグバージョンの `jconn4d.jar` と置き換えて、SAP 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。

環境変数を設定するには、次の手順を実行します。

- 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。

デバッグ方法の使用

SAP jConnect でのデバッグ方法をカスタマイズします。

他の `Debug` メソッドへの呼び出しを追加できます。

これらのメソッドのうち、静的なものには、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` - 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` - 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` と `stopTimer` - 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および停止します。 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
```

動的ロギング

15.7 ESD #4 以降、SAP jConnect for JDBC では、Java ロガーの標準メカニズムを実装することでロギングメカニズムをサポートしています。

例
アプリケーションで SAP jConnect ロガーのハンドルを取得し、必要に応じてロギングのオンとオフを切り替えます。

```
try
{
// Get logger for all classes present in
//"com.sybase.jdbc4.jdbc" package

Logger LOG = Logger.getLogger("com.sybase.jdbc4.jdbc");

// To log class-specific log message,
// provide complete class name, for example:
//Logger.getLogger("com.sybase.jdbc4.jdbc.
//SybConnection");
//Get handle as per user's requirement
Handler handler = new ConsoleHandler();

//Set logging level
handler.setLevel(Level.ALL);

//Added user specific handler to logger object
LOG.addHandler(handler);

//Set logging level
LOG.setLevel(Level.ALL);

Class.forName("com.sybase.jdbc4.jdbc.SybDriver");
Properties properties = new Properties();
properties.put("USER", USER_NAME);
properties.put("PASSWORD", PASSWORD);
Connection con =   DriverManager.getConnection("jdbc:sybase:Tds:" +
    HOST_PORT, properties);
Statement stmt = con.createStatement();
stmt.execute("select @@version");

//Dynamically turn off logging mechanism
LOG.setLevel(Level.OFF);
con.close();
...
```

```
}
```

ロギングレベル
SAP jConnect では、アプリケーションユーザがメッセージの細分性を Level.FINE、Level.FINER、および Level.FINEST に設定できます。 次に例を示します。

- ユーザが `SybConnection` クラスでロギングレベルを Level.FINE に設定すると、jConnect は次の内容を報告します。 `Dr1_Col setClientInfo(Properties)`
- `SybConnection` クラスで Level.FINER に設定した場合は、次のように報告されます。 `Dr1_Co1 setClientInfo(Properties.size = [3])`
- `SybConnection` クラスで Level.FINEST に設定した場合は、次のように報告されます。 `Dr1_Co1 setClientInfo(Properties = [[ClientUserValue, ApplicationNameValue, ClientHostnameValue]])`

SAP jConnect でのロギングの動的な有効化

アプリケーションで `LogHandler` API を使用して、ロギングをプログラムで動的に有効化します。

1. `LogHandler` API を使用して、ロギングをプログラムで有効化または無効化します。 次のコードを入力して、`SybConnection` クラスと `SybStatement` クラスでのコンソールレベルのロギングを有効化します。

```
LogManager logManager = LogManager.getLogManager();
    Handler handler = new ConsoleHandler();
    handler.setLevel(Level.ALL);
    Logger connLOG =
Logger.getLogger(SybConnection.class.getName());
    connLOG.addHandler(handler);
    connLOG.setLevel(Level.FINE);
    logManager.addLogger(connLOG);
    Logger stmtLOG =
Logger.getLogger(SybStatement.class.getName());
    stmtLOG.addHandler(handler);
    stmtLOG.setLevel(Level.FINE);
    logManager.addLogger(stmtLOG);
```

2. アプリケーションを実行します。
3. 手順 1 で示すように、`LogHandler` を使用してロギング設定を動的に調整できます。 これを実行するいくつかの方法を次に示します。
   - ロギングレベルを内部的に管理するプロパティを変更できるインタフェースを公開します。

- アプリケーション内で必要に応じてロギングを調整するサロゲートスレッドを実行します。

SAP jConnect でのロギングの静的な有効化

標準の Java ロギングメカニズムを実装する SAP jConnect でロギングを静的に有効化します。

1. テキストエディタを使用して、`$JRE_DIR/lib/logging.properties` の標準ロギング ファイルの内容を変更します。

```
handlers= java.util.logging.FileHandler
java.util.logging.FileHandler.formatter =
com.sybase.jdbc4.utils.LogUtil
.level= ALL
```

2. 次の内容をファイルに追加します。

```
com.sybase.jdbc4.jdbc.SybDriver.level = FINEST
com.sybase.jdbc4.jdbc.SybConnection.level = FINEST
com.sybase.jdbc4.jdbc.SybStatement.level = FINER
com.sybase.jdbc4.jdbc.SybPreparedStatement.level = FINE
com.sybase.jdbc4.jdbc.SybResultSet.level = FINE
```

3. ロギングのレベルを Level.FINE、Level.FINER、および Level.FINEST に調整して、適切なロギングの細分性を設定します。

注意： SAP jConnect では、パッケージレベルのロギングはサポートしていません。

4. `logging.properties` ファイルを保存します。

TDS 通信の取得

TDS は、クライアントアプリケーションと SAP Adaptive Server の間の通信を処理する SAP jConnect 独自のプロトコルです。

SAP jConnect には、ロー TDS パケットを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します。 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、SAP 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： ファイルに保存された取得済みの TDS プロトコルデータには、ユーザ認証に関する機密情報が含まれており、会社または顧客に関する機密データが含まれていることもあります。 この機密データが許可なく開示されたり、誤って開示さ

れたりすることがないように、ファイルパーミッションや暗号化を使用して、取得したデータを含むファイルを適切に保護してください。

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 SAP 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ために使用します。

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。

次の例は、PROTOCOL_CAPTURE を使用して TDS データを tds_data ファイルに送信する方法を示します。

```
...
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 パッケージ内にあります。このクラスには、pause メソッドと resume メソッドが含まれています。

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、非常に大きくなることがあります。 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れば、取得ファイルのサイズを制限することができます。

取得ファイルのサイズの制限

取得ファイルのサイズを制限するための手順を説明します。

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に cap.resume を置きます。
3. TDS データの取得を停止する場所に cap.pause を置きます。

接続エラーの解決

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可能性がある問題について説明します。

```
Gateway connection refused:
```

```
HTTP/1.0 502 Bad Gateway|Restart Connection
```

このエラーメッセージは、Adaptive Server への接続に使用されている *hostname* または *port#* 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。

hostname と *port#* 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 サーバを起動すると、さらに詳細な情報を確認することができます。

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。

SAP jConnect アプリケーションでのメモリ管理

SAP jConnect アプリケーションのメモリ使用量が増大していることがわかった場合は、`Statement` オブジェクトとサブクラスを使用します。

- SAP 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.prepareCall(_query).execute();

...
rs.close();
```

  代わりに、次の文を使用してください。

```
PreparedStatement ps = _conn.prepareCall(_query);
ResultSet rs = ps.executeQuery();

...
rs.close();
ps.close();
```

- 接続中の SAP Adaptive Server または SAP SQL Anywhere のデータベースのバージョンによっては、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を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、SAP 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が低下することがあるので、TYPE_SCROLL_INSENSITIVE の結果セットを使用するのは、結果セットが比較的小さい場合のみにすることをおすすめします。 SAP jConnect は、SAP 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

SAP jConnect とストアドプロシージャを使用する際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

`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は、エラーが発生します。

```
SQLState: JZ0SG - An RPC did not return as many output parameters as
the application had registered for it.
```

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 (@p1 int OUTPUT, ...
```

注意： SAP 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、SAP 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。

フェッチ/状態エラー

フェッチ/状態エラーは、クエリがローデータを返さない場合に発生します。

`executeQuery` メソッドではなく、`CallableStatement.executeUpdate` メソッドまたは `execute` メソッドを使用できます。

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、SAP jConnect では SQL 例外が発生します。

非連鎖トランザクションモードでのストアドプロシージャの実行

JDBC が `autocommit(true)` モードで接続を送信しようとすると、次のエラーが発生します。

`SAP Adaptive Server エラー 7713 - ストアドプロシージャは、非連鎖トランザクションモードでのみ実行できます。`

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、カスタムソケット実装エラーが発生します。

```
java.lang.IllegalArgumentException:
SSL_SH_anon_EXPORT_WITH_RC4_40_MDS
```

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。

パフォーマンスとチューニング

SAP 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。

SAP jConnect のパフォーマンスの改善

SAP jConnect を使用してアプリケーションのパフォーマンスを最適化する方法について説明します。

- text データや image データを SAP 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。
- 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。
- 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。
- 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パケットサイズを設定します。
- TDS-tunneled HTTP の場合は、最大 TDS パケットサイズを設定します。また、Web サーバが HTTP1.1 キープアライブ機能をサポートするように設定します。また、*SkipDoneProc* サーブレット引数を 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。
- 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。

参照：

- バッチ更新のサポート (73 ページ)
- image データ型 (76 ページ)
- 動的 SQL の準備文のパフォーマンスチューニング (153 ページ)
- SAP jConnect での TYPE_SCROLL_INSENSITIVE 結果セット (70 ページ)
- LANGUAGE_CURSOR 接続プロパティ (161 ページ)

BigDecimal の位取り変更

JDBC 1.0 仕様では、getBigDecimal メソッドには位取り係数が必須です。

BigDecimal オブジェクトがサーバから返されるときに、getBigDecimal で使用した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処理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 SAP 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 に設定すると、データベースから結果セットを取得するときのパフォーマンスを改善できます。

REPEAT_READ を false にする場合は、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。

SunIoConverter 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、SAP jConnect のサンプルに含まれている SunIoConverter クラスを使用します。

このコンバータは、Oracle Corporation から提供されている sun.io クラスに基づいています。

SunIoConverter 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 SAP jConnect 製品には統合されていません。このコンバータクラスは

参考用に提供されており、文字セット変換のパフォーマンスを改善するために SAP jConnect ドライバで使用できます。

注意： SAP によるテストでは、テスト対象のすべての VM において、`SunIoConverter` 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、Oracle Corporation 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 `SunIoConverter` 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。

`SunIoConverter` クラスを使用するには、SAP 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 `SunIoConverter` 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、SAP 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、『SAP 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。

参照：
- SAP jConnect の文字セットコンバータ (48 ページ)

動的 SQL の準備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のある 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** コマンドは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準備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。

SAP jConnect を使用するときは、次のオブジェクトを作成することによって、SAP 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。

- プリコンパイルされた文を格納する `PreparedStatement` オブジェクト (文が 1 つのセッション中に何度も実行される可能性がある場合)
- コンパイルされていない SQL 文を格納する `PreparedStatement` オブジェクト (文が 1 つのセッション中にほんの数回しか実行されない場合)

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 SAP 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。

SAP 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。

参照：

- 準備文かストアドプロシージャかの選択 (154 ページ)

準備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、ストアドプロシージャを使用すると、すべてのターゲットデータベース上に同じストアドプロシージャを用意する必要が生じます。 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準備文

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、移植可能なアプリケーション内で PreparedStatement を使用します。

- SAP 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。

準備文と SAP 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、準備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 文がセッション中に 1 ～ 2 回しか実行されない場合は、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、*dynamic* を true に設定して SybConnection.prepareStatement を使用します。 次に例を示します。

```
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

セッションの実行中に動的文をアプリケーション内で何度も実行する場合は、DYNAMIC_PREPARE と PreparedStatement オブジェクトを使用します。

- 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 を使用するか、*dynamic* を false に設定して SybConnection.prepareStatement を使用します。 次に例を示します。

```
PreparedStatement ps_uncomp =
   (SybConnection)conn.prepareStatement(sql_string, false);
```

```
PreparedStatement ps_uncomp = Connection.prepareCall(sql_string);
```

参照：

- 移植可能なアプリケーションでの準備文 (155 ページ)

Connection.PrepareStatement

SAP 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** オブジェクトで返すように設定することも、コンパイルされていない SQL 文を返すように設定することもできます。

プリコンパイルされた SQL 文を PreparedStatement オブジェクトで返すように 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 オブジェクトで返されます。

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。

SAP 専用のアプリケーション向けに、SAP jConnect 6.05 以降では SAP 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 準備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 (デフォルト) に設定されている場合、セッション内で Connection.prepareStatement が呼び出されるたびに、プリコンパイルされた文を PreparedStatement オブジェクトで返そうとします。
  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 に設定されている場合、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た動的 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。

次の例では、動的 SQL 文のプリコンパイルを無効にするために DYNAMIC_PREPARE が false に設定されています。**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", "false")
Connection conn = DriverManager.getConnection(url, props);
```

DYNAMIC_PREPARE が true に設定されている場合は、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** コマンドで使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、SAP 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動的 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、データベースから準備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準備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 メソッド

SybConnection.prepareStatement 拡張メソッドを使用すると、動的 SQL 文を PreparedStatement オブジェクトで返すことができます。

アプリケーションで SAP jConnect 固有の JDBC 拡張機能を使用できる場合、次のようになります。

```
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ためにデータベースのエラーが生成された場合は、SAP 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、SAP 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 に設定して、この解析を迂回してください。 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。

SAP jConnect での最適化されたバッチ処理

SAP jConnect は、PreparedStatement オブジェクトのバッチオペレーションを高速化するための内部アルゴリズムを実装しています。

このアルゴリズムは HOMOGENEOUS_BATCH 接続プロパティが true の場合に呼び出されます。

注意： 同種バッチ処理は、この機能をサポートしているサーバにクライアントアプリケーションが接続している場合にのみ使用できます。 SAP Adaptive Server Enterprise 15.7 には同種バッチ処理のサポートが導入されています。

次の例は、addBatch および executeBatch メソッドを使用した PreparedStatement バッチオペレーションを示します。

```
String sql = "update members set lastname = ? where member_id = ?";
```

```
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Forrester");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
prep_stmt.setString(1, "Robinson");
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
prep_stmt.setString(1, "Servo");
prep_stmt.setLong(2, 45131);
prep_stmt.addBatch();
prep_stmt.executeBatch();
```

ここで、connection は接続インスタンス、prep_stmt は準備文インスタンス、? は準備文のパラメータ用プレースホルダを表します。

ラージオブジェクト (LOB) カラムの同種バッチ処理

HOMOGENEOUS_BATCH および ENABLE_LOB_LOCATORS プロパティが true に設定されている場合、クライアントアプリケーションが同じバッチ内で LOB 準備文と LOB 以外の準備文の setter メソッドを混合することはできません。

たとえば、以下は無効です。

```
String sql = "update members SET catchphrase = ? WHERE member_id
= ?";
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Push the button, Frank!");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
Clob myclob = con.createClob();
myclob.setString(1, "Hi-keeba!");
prep_stmt.setClob(1, myclob);
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
pstmt.executeBatch();
```

ここでのキャッチフレーズは、カラムのデータ型 text です。 setString メソッドと setClob メソッドが同じカラムに対して同じバッチで使用されているので、このコードは失敗します。

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、SAP 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、SAP jConnect はカーソルのオープン、フェッチ、更新を行います。

SAP 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、一部のデータベースは言語カーソルをサポートしていません。 たとえば、SAP 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、SAP 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と、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ができます。

LANGUAGE_CURSOR 接続プロパティ

LANGUAGE_CURSOR は SAP jConnect のブール値の接続プロパティで、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します。

- LANGUAGE_CURSOR が false (デフォルト) の場合、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です。 SAP jConnect は、TDS プロトコルのトークンとしてカーソルコマンドを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。
- LANGUAGE_CURSOR が true の場合、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です。 SAP jConnect は、データベースに SQL 文を送信して解析とコンパイルを行うことによって、カーソルを作成および操作します。 LANGUAGE_CURSOR を true に設定することに利点は特にありませんが、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。

SAP jConnect アプリケーションのマイグレート

SAP jConnect 5.x、6.x、7.x、および SAP jConnect 16.x からアプリケーションをマイグレートするための手順を説明します。

SAP jConnect 16.x へのアプリケーションのマイグレート

SAP jConnect 16.x にアプリケーションをマイグレートするための手順を説明します。

1. コード内で SAP jConnect 拡張機能を使用している場合や SAP 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、次のようなインポート文があります。
   ```
   import com.sybase.jdbc.*
   ```
   および
   ```
   import com.sybase.jdbc2.jdbc.*
   ```
   上記を次のように変更します。
   ```
   import com.sybase.jdbcx.*
   ```
2. JDBC_HOME を、インストールした SAP 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-16_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。次のパスを指定する必要があります。
   ```
   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. SAP jConnect 16.0 ドライバが、CLASSPATH 環境変数で指定されている最初の SAP jConnect ドライバであることを確認します。

参照：

- SAP jConnect 拡張機能の変更 (164 ページ)

SAP jConnect 拡張機能の変更

SAP jConnect バージョン 4.1 以降には、SAP jConnect の JDBC 拡張機能がすべて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 SAP 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、SAP jConnect のさまざまなバージョンに対して一貫したインタフェースを提供します。 SAP jConnect 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 SAP jConnect 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 SAP jConnect にアプリケーションを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。

SAP jConnect 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

アプリケーションが `SybMessageHandler` を使用する場合のコードの違いを説明します。

- SAP 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());
```

- SAP 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());
```

SAP jConnect 拡張機能の使用方法の詳細な例については、SAP jConnect 付属のサンプルを参照してください。

メソッド名

インタフェース内で名前が変更されたメソッドのリストを示します。

表 8 : メソッド名の変更

実際のメソッド名	バージョン 4.0 以前	バージョン 4.0 以降
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。

SAP 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。 次に例を示します。

```
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);
```

SAP jConnect 拡張機能の完全なリストについては、SAP jConnect インストールディレクトリの docs/ ディレクトリにある SAP jConnect の Javadoc マニュアルを参照してください。

Web サーバゲートウェイ

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。

SAP jConnect には、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、SAP 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： SAP jConnect は、クライアントシステムでの SSL もサポートします。

参照：
- カスタム SSL ソケットプラグインの実装 (124 ページ)

TDS トンネリング

SAP jConnect は、TDS を使用してデータベースサーバと通信します。 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。

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。 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、SAP jConnect は TDS トンネリングサーブレットを使用して、HTTP 要求間で論理接続を維持します。 このサーブレットは最初のログイン要求時にセッション ID を生成し、以降のすべての要求のヘッダにそのセッション ID が格納されます。 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で提供される論理接続によって、SAP jConnect は 2 つのシステム間での暗号化された通信をサポートできます。たとえば、SAP 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 サーバに接続することができます。

SAP jConnect とゲートウェイの設定

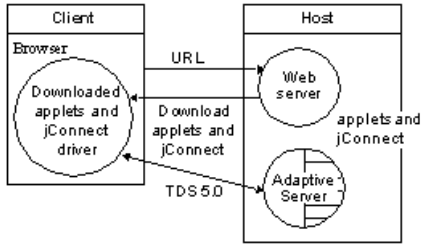
SAP jConnect ドライバをインストールし、TDS トンネリングサーブレットでゲートウェイを使用するために、Web サーバと SAP Adaptive Server を設定するためのオプションがいくつかあります。

Web サーバと SAP Adaptive Server を同じホスト上に配置

2 層設定では、Web サーバと SAP Adaptive Server の両方が同じホストにインストールされます。

- SAP jConnect を Web サーバホスト上にインストールします。
- ゲートウェイは必要ありません。

図 3 : Web サーバと SAP Adaptive Server を同じホスト上に配置



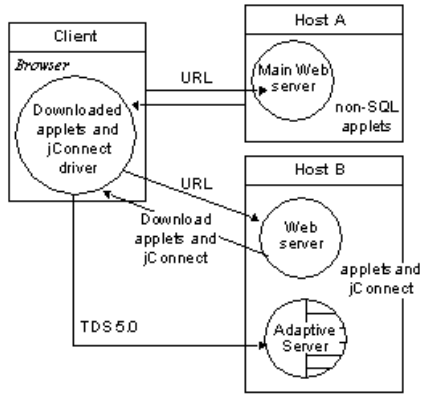
専用 JDBC Web サーバと SAP Adaptive Server を同じホスト上に配置

単一ホスト設定では、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、SAP Adaptive Server アクセス専用の Web サーバと SAP 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 (SAP Adaptive Server) ホスト上に SAP jConnect をインストールします。
- 2 番目の (SAP Adaptive Server) ホストにゲートウェイは必要ありません。

図 4 : 専用 JDBC Web サーバと SAP Adaptive Server を同じホスト上に配置

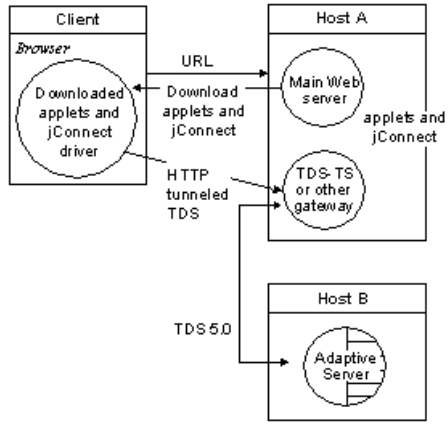


Web サーバと SAP Adaptive Server をそれぞれ別のホスト上に配置

3 層設定では、SAP Adaptive Server を Web サーバとは別のホストに配置します。SAP jConnect には、SAP 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。

- SAP jConnect を Web サーバホスト上にインストールします。
- TDS トン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します。

図 5 : Web サーバと SAP Adaptive Server をそれぞれ別のホスト上に配置



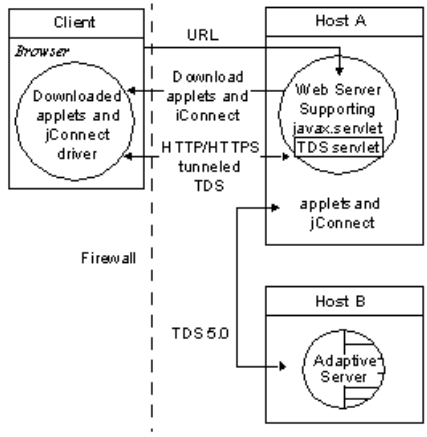
ファイアウォールを介したサーバへの接続

ファイアウォールで保護されたサーバに接続します。

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、TDS トンネリングサーブレットを Web サーバ上で使用する必要があります。

- SAP jConnect を Web サーバホスト上にインストールします。
- **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。

図 6 : 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

Web サーバゲートウェイの使用上の条件について説明します。

Index.html ファイルの表示

Web ブラウザを使用して、SAP jConnect インストールディレクトリ内の `index.html` ファイルを表示します。 `index.html` には、SAP 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。

SAP 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、『SAP jConnect for JDBC インストールガイド』の「Netscape の使用による CLASSPATH の制限」を参照してください。

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。

サンプルアプレットの実行

SAP jConnect でサンプルアプレットを実行する手順は次のとおりです。

1. [Run Sample JDBC Applets] をクリックします。
2. [Executable Samples] 表の中で、`Isql.java` を見つけて、行の最後にある [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。 デフォルトの SAP 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、クエリが表示されます。 [Go] をクリックした後、アプレットはデフォルト値を使用して SAP jConnect デモンストレーションデータベースに接続し、結果を返します。

アプレット画面のサイズ変更

UNIX プラットフォームで、アプレットが予期したとおりに表示されない場合は、アプレット画面のサイズを変更できます。

1. テキストエディタを使用して、`$JDBC_HOME/sample2/gateway.html` を開きます。
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

TDS トンネリングサーブレットを使用するには、Oracle Corporation 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、SAP 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 ストリームの長さと、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。

- **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 は必要ありません。
  - 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
- **ゲートウェイからクライアント** – ヘッダには、TDS サーブレットによって割り当てられたセッション ID が含まれます。
  - 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
- **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 が含まれます。
  - 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”
- **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 が含まれます。

- 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 トンネリングで SAP jConnect サーブレットを使用するには、`javax.servlet` インタフェースをサポートしている Web サーバが必要です。

サーバをインストールするには、Java サーブレット付属のマニュアルの指示に従ってください。

インストールとサーブレット引数の設定

SAP 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 トンネリングサーブレットに必要なファイルがあります。

1. この SAP 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。
2. サーブレットを Web サーバに追加します。パフォーマンスをカスタマイズするには、オプションの引数を設定します。
   - **SkipDoneProc** [true|false] - 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。 不要なデータは暗号化や暗号化解除されないため、これは 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、SAP 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。

参照：

- SAP jConnect でのデバッグ (141 ページ)

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 SAP 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されます。

SAP 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
```

SAP 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がセッションごとに表示されます。

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 であれば、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

TDS セッションを終了するには、アクティブな TDS セッションで定義した URL を使用します。

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選択し、[Terminate This Session] をクリックします。

TDS セッションの再開

SESSION_ID が指定されると、SAP 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。

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。

SAP jConnect サンプルプログラム

SAP jConnect サンプルプログラムについて説明します。

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、SAP 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 を使用する場合、URL は http://*host:port*。 HTTPS を使用する場合、URL は https://*host.port*/*servlet_alias*。
-p	HTTP と HTTPS のどちらを使用するか。

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。 次に例を示す。 -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` クラスについてのみ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める必要がある (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 SAP jConnect メッセージを表示する言語を指定する。
-K	SAP 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。 このパラメータでは、サービスプリンシパル名を設定する。 次に例を示す。 `-K myASE` サーバのサービスのプリンシパル名は **myASE**。
-F	JAAS ログイン設定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。 次に例を示す。 `-F /foo/bar/exampleLogin.conf` SAP 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。

パラメータ	説明
-T	このパラメータが設定されているとき、SAP 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。 SAP 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。

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
```

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)

参照：

- セキュリティ (123 ページ)
- JCONNECT_VERSION 接続プロパティ (6 ページ)

SAP jConnect のサンプルプログラムとサンプルコード

SAP jConnect には、SAP jConnect がさまざまな JDBC クラスおよびメソッドでどのように動作するかを理解するために役立つサンプルプログラムが付属しています。

サンプルアプリケーション

SAP jConnect をインストールする際に、サンプルプログラムもインストールできます。これらのプログラムにはソースコードが含まれており、SAP jConnect がさまざまな JDBC クラスおよびメソッドをどのように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の詳細については、『SAP jConnect for JDBC インストールガイド』を参照してください。

注意： SAP 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、SAP 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` サブディレクトリ内の `index.html` ファイルには、使用可能なサンプルすべて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

サンプルプログラムの一部は Web ブラウザ内でアプレットとして実行できます。これにより、出力結果を検討しながらソースコードを確認できます。

サンプルプログラムをアプレットとして実行するには、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。

SAP SQL Anywhere でのサンプルプログラムの実行

サンプルプログラムはすべて SAP Adaptive Server に対応していますが、SAP SQL Anywhere に対応しているものは限られています。

SAP 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。

SAP SQL Anywhere で使用できるサンプルプログラムを実行するには、SAP 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/pubs2_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 SAP jConnect ドライバを呼び出し、接続を確立し、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);
       }
   }
```

SQL の例外メッセージと警告メッセージ

SAP 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を示します。

SQL ステータス	メッセージ/説明/対処方法
010AF	`SEVERE WARNING: An assertion has failed, please use devclasses todetermine the source of this serious bug. Message = _____.` 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、SAP 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する。
010CP	`AutoCommit option has changed to true. All pending statements on this transaction (if any) are committed.`
010DF	`Attempt to set database at login failed. Error message:_____.` 説明: SAP jConnect は、接続 URL で指定されているデータベースに接続できない。 対処方法: URL 内のデータベース名が正しいことを確認する。 また、SAP 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する。
010DP	`Duplicate connection property _____ ignored.` 説明: 接続プロパティが 2 回定義されている。 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 2 回定義されている可能性がある (“password” と “PASSWORD” など)。 SAP jConnect は、大文字と小文字の指定が異なるだけで名前が同じプロパティを区別しない。 または、接続プロパティリスト内と URL 内の両方で接続プロパティが定義されている可能性がある。 このような場合は、接続プロパティリスト内のプロパティ値が優先される。 対処方法: アプリケーションで接続プロパティを 1 回だけ定義する。 ただし、URL で定義された接続プロパティよりも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動作をアプリケーションで利用することもできる。 このような場合は、この警告を無視してかまわない。

SQL ステータス	メッセージ/説明/対処方法
010HA	`The server denied your request to use the high-availability feature. Please reconfigure your database, or do not request a high-availability session.` 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する。または、REQUEST_HA_SESSION を true に設定しない。
010HD	`SAP Adaptive Server high-availability failover is not supported by this type of database server.` 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する。
010HN	`The client did not specify a SERVICE_PRINCIPAL_NAME Connection property. Therefore, jConnect is using the hostname of _____ as the service principal name` 対処方法: 接続プロパティを使用してサービスプリンシパル名を明示的に指定する。
010HT	`Hostname property truncated, maximum length is 30.` 対処方法: 名前が 30 バイトにトランケートされることを示す SAP jConnect の警告にすぎないので、特に対処する必要はない。この警告が表示されないようにするには、HOSTNAME を 30 バイト以下の長さの文字列に設定する必要がある。
010KF	`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.` 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する。詳細については、「セキュリティ (123 ページ)」を参照。
010MX	`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: _____.` 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ない可能性がある。 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する。『SAP jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」を参照。

SQL ステータス	メッセージ/説明/対処方法
010P4	`An output parameter was received and ignored.` 説明: 実行したクエリが出力パラメータを返したが、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ので、無視された。 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す。 このためには、**CallableStatement** を使用してクエリを実行し、**registerOutputParameter** と **getXXX** の呼び出しを追加しなければならない場合がある。 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る。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る。
010P6	`A row was received and ignored.` 説明: タイプ 0xD1 の予期しないオブジェクトが処理中の結果セット内で検出され、無視された。 対処方法: 結果セットを生成したクエリをチェックし、必要に応じて修正する。
010PF	`One or more jars specified in the PRELOAD_JARS connection property could not be loaded.` 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する。 `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する。 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る。 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する。

SQL ステータス	メッセージ/説明/対処方法
010PO	`Property LITERAL_PARAM has been reset to false because DYNAMIC_PREPARE was set to true.` 説明: プリコンパイルされた動的文を使用するには、それらの文にパラメータが送信されるようにする (パラメータを受け取る文の場合)。 LITERAL_PARAMS を true に設定すると、サーバに送信する SQL のすべてのパラメータがリテラル値として送信される。 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ない。 対処方法: この警告を回避するには、動的 SQL を使用するときに LITERAL_PARAMS を true に設定しない。 「動的 SQL の準備文のパフォーマンスチューニング (153 ページ)」を参照。
010RC	`The requested ResultSet type and concurrency is not supported. They have been converted.` 説明: SAP jConnect で使用可能な結果セット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 (61 ページ)」を参照。 対処方法: 結果セットについてサポートされているタイプと同時実行性の組み合わせを要求する。
010SJ	`Metadata accessor information was not found on this database. Please install the required tables as mentioned in the jConnect documentation.` 説明: メタデータ情報がサーバ上に設定されていない。 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ドプロシージャをインストールする。『jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」を参照。 メタデータが必要ない場合は、USE_METADATA プロパティを false に設定する。
010SK	`Database cannot set connection option _____.` 説明: 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アプリケーションが実行しようとした。 対処方法: データベースをアップグレードするか、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する。

SQL ステータス	メッセージ/説明/対処方法
010SL	`Out-of-date metadata accessor information was found on this database. Ask your database administrator to load the latest scripts.` 説明: サーバ上のメタデータ情報が古いため、更新する必要がある。 対処方法: SAP 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する。『SAP jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」を参照。
010SM	`This database does not support the initial proposed set of capabilities, retrying.` 説明: 11.9.2 以前のバージョンの SAP Adaptive Server Enterprise 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問題がある。 この警告は、SAP 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す。 SAP jConnect によってこのバグが検出されると、サーバに対する接続が 2 回試行される。 対処方法: クライアントはこの警告を無視してもかまわないが、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には、`ELIMINATE_010SM` 接続プロパティを true に設定する。 SAP Adaptive Server バージョン 12.0 以降に接続する場合は、このプロパティを true に設定しない。
010SN	`Permission to write to file was denied. File: _____. Error message: _____.` 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る。 対処方法: アプレットからファイルへの書き込みを行う場合は、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する。
010SP	`File could not be opened for writing. File: _____. Error message: _____.` 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する。

SQL ステータス	メッセージ/説明/対処方法
010SQ	`The connection or login was refused, retrying connection with the host/port address.` 説明: CONNECTION_FAILOVER 接続プロパティが true に設定されているときに、接続先サーバのリストにあるデータベースサーバの 1 つに SAP jConnect が接続できない。 そのため、SAP jConnect はリスト内の次のサーバへの接続を試行する。 対処方法: SAP jConnect が別のデータベースサーバに接続できるのであれば、特に対処する必要はない。 ただし、接続警告の発生原因となったサーバに SAP jConnect が接続できなかった理由を調べる必要がある。
010TP	`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.` 説明: サーバは SAP jConnect によって最初に要求された文字セットを使用できず、別の文字セットを使用して応答した。 SAP jConnect は変更を受け入れ、必要な文字セット変換を実行する。 このメッセージは情報メッセージであり、これ以上の結果はない。 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する。
010TQ	`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.` 説明: 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 文字だけである。 そのため、SAP jConnect の文字セットは 7 ビット ASCII に戻る。 このメッセージは情報メッセージであり、これ以上の結果はない。 対処方法: SAP 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する。 『SAP jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」を参照。

SQL ステータス	メッセージ/説明/対処方法
010UF	`Attempt to execute use database command failed. Error message:_____.` 説明: SAP jConnect は、接続 URL で指定されているデータベースに接続できない。 次の 2 つの原因が考えられる。 • URL 内の名前が正しくない。 • `USE_METADATA` が true (デフォルト) に設定されているが、メタデータを返すストアドプロシージャがインストールされていない。 その結果、SAP jConnect は URL 内のデータベースを使用して **use** *database* コマンドを実行しようとしたが、失敗した。 この理由としては、**use** *database* コマンドをサポートしない SQL Anywhere データベースにアクセスしようとしたことが考えられる。 対処方法: URL 内のデータベース名が正しいことを確認する。 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する。『SAP jConnect for JDBC インストールガイド』の「ストアドプロシージャのインストール」、および『リリースノート SAP jConnect for JDBC』を参照。 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 に設定する。
010UP	`Unrecognized connection property _____ ignored.` 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ある。認識できないプロパティは無視される。 対処方法: アプリケーション内の URL 定義をチェックして、有効な SAP jConnect ドライバ接続プロパティのみが参照されていることを確認する。
0100V	`The version of TDS protocol being used is too old. Version: _____.` 説明: SAP jConnect にはバージョン 5.0 以降が必要である。 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する。『SAP jConnect インストールガイド』の「システム稼働条件」を参照。
01S07	`Adaptive Server may round or truncate nanosecond values.` 説明: 1/300 秒より高い精度の時間値が検出された。 SAP Adaptive Server はそのような精度をサポートしないため、jConnect が値を拒否した。 対処方法: 時間値の精度が 1/300 秒以下であることを確認する。

SQL ステータス	メッセージ/説明/対処方法
01S08	`This connection has been enlisted in a Global transaction. All pending statements on the current local transaction (if any) have been rolled back.` 説明: SAP 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する。 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する。 対処方法: `XAResource.start()` メソッドを発行する前に、アクティブな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。
01S09	`The local transaction method _____ cannot be used while a global transaction is active on this connection.` 説明: ローカルオペレーションの例としては、接続に対する commit() メソッドの呼び出しがある。 使用できない他のオペレーションは次のとおり。 `rollback()`、`rollback(Savepoint)`、`setSavepoint()`、`setSavepoint(String)`、`releaseSavepoint(Savepoint)`、および `setAutoCommit()`。 対処方法: ローカルトランザクションをグローバルトランザクションとは別に実行する。 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。
01S10	`The local transaction method _____ cannot be used on a pre-System 12 XAConnection.` 説明: SAP SQL Anywhere バージョン 12 より前のバージョンで機能しないローカルトランザクションメソッドを使用した。 対処方法: そのメソッドを使用しない。
01S11	`WARNING: Your data might be truncated.` 説明: ユーザが指定したストリームまたは LOB の長さが、**ResultSet.updateXXX** メソッドの制限 (Integer.MAX_VALUE) を超えている。 対処方法: 長さが制限内であることを確認する。

SQL ステータス	メッセージ/説明/対処方法
01S12	`Unable to continue with HOMOGENEOUS_BATCH protocol, falling back to normal batching.` 説明: DYNAMIC_PREPARE が false に設定されている場合、SAP Adaptive Server はパラメータメタデータを送信しない。 HOMOGENEOUS_BATCH が true に設定されている場合、SAP jConnect は最適化のためにこの情報を必要とする。 このため、jConnect は通常バッチに戻る。 対処方法: 最適化されたバッチ (HOMOGENEOUS_BATCH が true) は、プリコンパイルされた動的 SQL 準備文 (DYNAMIC_PREPARE が true) のみで使用する。

01S13	`Connected Adaptive Server server does not support the set option 'logbulkcopy' needed for logging BCP.`
	`Falling back to normal bulk load without logging which is equivalent to setting ENABLE_BULK_LOAD=BCP.`
	説明: SAP Adaptive Server が 15.7 ESD #1 より前のバージョンで、ログを取るバルクロードをサポートしない場合、SAP jConnect は通常バッチに戻る。
	対処方法: ENABLE_BULK_LOAD=LOG_BCP の設定は、SAP Adaptive Server 15.7 ESD #1 以降でのみ使用する。

01ZZZ	エラーコード 4022: `Password has expired Please set the NEWPASSWORD property with the new password or use sp_password to change passwords.` 対処方法: SAP Adaptive Server に接続するためのパスワードを再設定する。
JZ001	`User name property '_____' too long. Maximum length is 30.` 対処方法: 最大長 30 バイトを超えないようにする。
JZ002	`Password property '_____' too long. Maximum length is 30.` 対処方法: 最大長 30 バイトを超えないようにする。

SQL ステータス	メッセージ/説明/対処方法
JZ003	`Incorrect URL format. URL: _____.` 対処方法: URL の形式を確認する。 「URL 接続プロパティのパラメータ (39 ページ)」を参照。 PROXY 接続プロパティを使用している場合に、PROXY プロパティの形式が正しくないと、接続を試行するときに JZ003 例外が発生することがある。 • カスケードプロキシの PROXY の形式: *ip_address:port_number* • TDS トンネリングサーブレットの PROXY の形式: http[s]://*host:port/tunneling_servlet_alias*
JZ004	`User name property missing in DriverManager.getConnection(..., Properties)` 対処方法: 必須のユーザプロパティを指定する。
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、これらの例外が SQL 例外として再度発生する。 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する。 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、SAP jConnect によって JZ0C1 例外が JZ006 のチェーンに追加される。 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る。 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る。
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る。 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する。または **rs.next** を呼び出す。

SQL ステータス	メッセージ/説明/対処方法
JZ009	`Error encountered in conversion. Error message: _____.` 説明: 次のような原因が考えられる。 • `date` を `int` に変換するなど、互換性のない 2 つのデータ型の間で変換を行おうとした。 • 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、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する。 文字列が正しくフォーマットされていることを確認する。 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ない。
JZ00A	`Invalid precision and scale specified for a numeric value.` 説明: `setBigDecimal` メソッドを使用しているときに、BigDecimal 値が 1 未満の精度値、負の位取り値、位取り値未満の精度値、または 127 を超える精度値に設定されている。 対処方法: クエリを確認し、有効な精度/位取り値を指定して修正する。
JZ00B	`Numeric overflow.` 説明: **BigInteger** を TDS の数値型として送信しようとしたが、値が大きすぎた。または Java の `long` を `int` として送信しようとしたが、値が大きすぎた。 対処方法: これらの値は、SAP jConnect では格納できない。 `long` の場合は、SAP jConnect の数値型を使用することを検討する。 **Bignum** については、解決方法がない。
JZ00C	`The precision and scale specified cannot accommodate numeric value _____.` 説明: setBigDecimal メソッドを使用しているときに、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る。 対処方法: 指定した精度と位取りが BigDecimal 値に十分な大きさであることを確認する。
JZ00E	`Attempt to call execute() or executeUpdate() for a statement where setCursorName()has been called.` 対処方法: カーソルを削除または更新するには、別の文を使用する。 「結果セットでのカーソルの使用 (61 ページ)」を参照。

SQL ステータス	メッセージ/説明/対処方法
JZ00F	`Cursor name has already been set by setCursorName().` 対処方法: 同じ文に対してカーソル名を 2 回設定しない。 現在のカーソル文の結果セットをクローズする。
JZ00G	`No column values were set for this row update.` 説明: ローを更新しようとしたが、カラム値が変更されていない。 対処方法: **updateXX** メソッドを呼び出してから **updateRow** を呼び出す。
JZ00H	`The result set is not updatable. Use Statement.setResultSetConcurrencyType().` 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** 文に追加する。
JZ00I	`Invalid scale. The specified scale must be >=0.` 説明: 位取り値は 0 より大きくなければならない。 対処方法: 位取り値が負の値でないことを確認する。
JZ00L	`Login failed. Examine the SQLWarnings chained to this exception for the reason(s).` 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る。

JZ00M	`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.`
	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う。

JZ010	`Unable to deserialize an Object value. Error text: _____.` 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する。
JZ011	`Number format exception encountered while parsing numeric connection property _____.` 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た。 対処方法: 数値型の接続プロパティには整数値を指定する。

SQL ステータス	メッセージ/説明/対処方法
JZ012	`Internal Error. Please report it to SAP technical support. Wrong access type for connection property _____.` 対処方法: SAP 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る。
JZ013	`Error obtaining JNDI entry: _____.` 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する。
JZ014	`You may not setTransactionIsolation(Connection.TRANSACTION_NONE). This level cannot be set; it can only be returned by a server.` 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する。

JZ015	`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.`
	対処方法: `GSSMANAGER_CLASS` プロパティに設定されている値を確認する。

JZ017	`Savepoint is not valid.` 説明: ロールバックまたは解放に存在しないセーブポイントを指定した。 対処方法: クエリを確認し、存在するセーブポイントを指定して修正する。
JZ018	`This method can not be applied to this type of savepoint.` 説明: `getSavepointId()` メソッドは名前付きセーブポイント (ID なし) では機能せず、`getSavepointName()` メソッドは名前のないセーブポイント (名前なし) では機能しない。 対処方法: クエリを確認して修正する。
JZ019	`Error obtaining SERVERNAME : _____.` 説明: **jdbc::jndi:file** を使用して設定された URL で、`sql.ini` ファイル (Windows) または `interfaces` ファイル (UNIX) またはサーバ名が指定されていない。 対処方法: URL コマンドを確認して修正する。

SQL ステータス	メッセージ/説明/対処方法
JZ021	`The Specified _____ file not found.` 説明: 接続 URL で指定された `sql.ini` ファイル (Windows) または `interfaces` ファイル (UNIX) が見つからない。 対処方法: 接続 URL をチェックして修正する。
JZ022	`The Specified _____ file has an unknown format.` 説明: `sql.ini` ファイル (Windows) または `interfaces` ファイル (UNIX) の接続 URL 文字列のフォーマットが正しくない。 対処方法: 接続 URL 文字列をチェックして修正する。
JZ024	`The Specified Server : _____ has no entry in the interfaces/sql.ini file :_____.` 対処方法: 接続 URL 文字列をチェックして修正する。
JZ025	`The TLI format for Specified Server in interfaces/sql.ini is Invalid.` 対処方法: 設定をチェックして修正する。
JZ026	`The Specified Protocol : _____ for Server : _____ in interfaces/sql.ini file :_____ is not Supported.` 説明: TLI、TCP、および NLWNSCK 以外のプロトコルが `sql.ini` ファイル (Windows) または `interfaces` ファイル (UNIX) で指定されている。 対処方法: サポートされているプロトコルを指定する。
JZ027	`The Specified SECMECH entrys: _____ for Server : _____ in interfaces/sql.ini file :_____ are not Supported.` 説明: Kerberos 接続 URL で無効な値が指定されている。 対処方法: URL を確認して修正する。
JZ028	`Illegal value set for JCE_PROVIDER_CLASS connection property. The property value must be a fully qualified provider class name passed as a String or an instance of java.security.Provider.` 対処方法: 有効な値を指定する。

SQL ステータス	メッセージ/説明/対処方法
JZ029	`Error looking up address for ALTERNATE_SERVER_NAME _____, (_____).` 説明: SAP jConnect は、SAP SQL Anywhere UDP 検出プロトコルを使用して、ALTERNATE_SERVER_NAME プロパティで指定されたサーバを検出できない。 対処方法: ALTERNATE_SERVER_NAME 接続プロパティで指定されたサーバ名をチェックして修正する。
JZ030	`The method _____ is not supported.`
JZ031	`Failed to unwrap the object of _____.` 説明: SAP jConnect は、クラスパスにないカスタムクラスのオブジェクトをアンラップできない。 対処方法: クラスをクラスパスに追加する。
JZ032	`A Date or Timestamp parameter exceeds the BigDateTime/BigTime range. The server can only support BigDateTime values between 0001/01/01 12:00:00:000000AM to 9999/12/31 11:59:59.999999PM or BigTime values between 12:00:00:000000AM to 11:59:59.999999PM.`
JZ033	`Unknown Blob type returned by the server.` 説明: SAP jConnect は、カラムの SAP Adaptive Server データ型を BLOB データ型にマップできない。 対処方法: SAP Adaptive Server カラムが BLOB データ型に変換可能であることを確認する。
JZ034	`The connected server is not capable of handling Large Objects [LOB].` 対処方法: 通常のストリームメソッドを使用して LOB にアクセスする。
JZ035	`To handle Large object [LOB], please set connection property 'ENABLE_LOB_LOCATOR' to true.`
JZ036	`Reference to this Large Object [LOB] is no longer valid in database. Check if you have called free() or check if transaction ended.`
JZ037	`Value of offset/position/start should be in the range [1, len] where len is the length of Large Object[LOB].`

SQL ステータス	メッセージ/説明/対処方法
JZ038	`Length of the object should be >= 0.`
JZ040	`_____ operation failed. The _____ has been closed alread.` 説明: 入力ストリームまたはLOBリーダ(出力ストリームまたはLOBライタ)がすでにクローズされているため、読み取り(書き込み)オペレーションが失敗した。 対処方法: アプリケーションをチェックして競合の理由を検出し、修正する。
JZ041	`_____ operation failed on the _____.` 説明: 入力ストリームまたはリーダ(出力ストリームまたはライタ)(入力ストリーム)がすでにクローズされているため、読み取り(書き込み)(available())オペレーションが失敗した。 対処方法: アプリケーションをチェックして競合の理由を検出し、修正する。
JZ042	`Large Object setters can not be mixed with other setters when ENABLE_LOB_LOCATOR and HOMOGENEOUS_BATCH are set to TRUE. java.sql.Types _____ was mixed with java.sql.Types _____.`
JZ043	`LOB objects are not supported for any of the possible variants of 'ENABLE_BULK_LOAD property', but false. Please consider using alternate setter APIs to insert the data.`
JZ044	`Server-side locators can not be created within the batch if,SEND_BATCHPARAMS_IMMEDIATE is TRUE. Try using client side LOBs or set SEND_BATCHPARAMS_IMMEDIATE to FALSE.`
JZ0BD	`Out of range or invalid value used for method parameter.`
JZ0BI	`setFetchSize: The fetch size should be set with the following restrictions  0 <= rows <= (maximum number of rows in the ResultSet).` 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する。
JZ0BJ	`The value set for the IMPLICIT_CURSOR_FETCH_SIZE connection property must be > 0.`
JZ0BP	`Output parameters are not allowed in Batch Update Statements.`

SQL ステータス	メッセージ/説明/対処方法
JZ0BR	`The cursor is not positioned on a row that supports the _____ method.` 対処方法: 現在のローの位置に無効な **ResultSet** メソッドを呼び出さない。
JZ0BS	`Batch Statements not supported.` 対処方法: データベースに最新バージョンの SAP jConnect メタデータストアドプロシージャをインストールするか、または最新バージョンに更新する。
JZ0BT	`The _____ method is not supported for ResultSets of type _____.` 対処方法: **ResultSet** の型に無効な **ResultSet** メソッドを呼び出さない。
JZ0C0	`Connection is already closed.` 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する。
JZ0C1	`An IOException occurred which closed the connection.` 説明: これ以降、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ない。 この例外が発生した場合は、必ず JZ006 例外との例外チェーン内に追加される。 対処方法: 接続をクローズさせた IOException の原因を調べる。
JZ0CL	`You must define the CLASS_LOADER property when using the PRELOAD_JARS property.`
JZ0D4	`Unrecognized protocol in JDBC URL:_____.` 説明: TDS 以外のプロトコルを使用して接続 URL を指定した。TDS は、SAP jConnect が現在サポートする唯一のプロトコルである。 対処方法: URL 定義を確認する。 URL で TDS をサブプロトコルとして指定する場合は、次のフォーマットと大文字/小文字の指定に従って入力する。 jdbc::Tds:*host*:*port* 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する。 jdbc::jndi:
JZ0D5	`Error loading protocol _____.` 対処方法: CLASSPATH システム変数の設定を確認する。

SQL ステータス	メッセージ/説明/対処方法
JZ0D6	`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.`
JZ0D7	`Error loading url provider _____. Error message: _____.` 対処方法: JNDI URL が正しいことを確認する。
JZ0D8	`Error initializing url provider: _____.` 対処方法: JNDI URL が正しいことを確認する。
JZ0EM	`End of data.` 対処方法: SAP 製品の保守契約を結んでいるサポートセンタに連絡する。
JZ0F1	`SAP Adaptive Server Enterprise high-availability failover connection was requested but the companion server address is missing.` 説明: `REQUEST_HA_SESSION` 接続プロパティを true に設定するときは、フェールオーバサーバも指定する必要がある。 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する。
JZ0F2	`SAP Adaptive Server Enterprise high-availability failover has occurred. The current transaction is aborted, but the connection is still usable. Retry your transaction.` 説明: 接続していたバックエンドデータベースサーバが応答しなくなったが、セカンダリサーバにフェールオーバしたので、データベース接続は引き続き使用可能である。 対処方法: クライアントコードはこの例外をキャッチし、最後にコミットされた時点からトランザクションを再開する必要がある。 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行できる。
JZ0FP	`Incorrect value passed for parameter ____.` 説明: 現在の結果セットの状態に対して指定されたパラメータの値が無効である。 対処方法: 値が有効 (CLOSE_CURRENT_RESULT、KEEP_CURRENT_RESULT、CLOSE_ALL_RESULTS) であることを確認する。

SQL ステータス	メッセージ/説明/対処方法
JZ0GC	`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.`
JZ0GK	`The _____ array can not be null and has to contain only one key.` 説明: 自動生成キーカラム名/インデックス配列に NULL が指定されているか、複数のキーが指定されている。 IDENTITY カラムに関連するため、配列で許可されるキーは 1 つだけである。 対処方法: クエリをチェックして修正する。
JZ0GN	`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.` 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 環境変数で指定されていることを確認する。
JZ0GS	`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 ____.` 対処方法: メジャーおよびマイナーのエラーコードとエラーメッセージを調べる。 Kerberos 設定をチェックする。 「セキュリティ (123 ページ)」を参照。
JZ0H0	`Unable to start thread for event handler; event name = _____.` 対処方法: SAP 製品の保守契約を結んでいるサポートセンタにこのエラーを報告する。
JZ0H1	`An event notification was received but the event handler was not found; event name = _____.` 対処方法: SAP 製品の保守契約を結んでいるサポートセンタにこのエラーを報告する。

SQL ステータス	メッセージ/説明/対処方法
JZ0HC	`Illegal character '_____' encountered while parsing hexadecimal number.` 説明: バイナリ値を表す文字列で、16 進数に必要な範囲 (0 ～ 9、a ～ f) 外の文字が使用されている。 対処方法: 文字列内の文字値が必要な範囲内にあることを確認する。
JZ0I3	`Incorrect argument _____ passed to method _____.` 対処方法: 製品マニュアルまたは JDBC API で、正しい引数が渡されていることを確認する。
JZ0I5	`An unrecognized CHARSET property was specified: _____.` 対処方法: `CHARSET` 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する。 「jConnect 文字セットコンバータ (48 ページ)」を参照。
JZ0I6	`An error occurred converting UNICODE to the charset used by the server. Error message: _____.` 対処方法: SAP jConnect クライアントの `CHARSET` 接続プロパティで、サーバに送信する必要があるすべての文字をサポートする別の文字セットコードを選択する。 サーバにも別の文字セットをインストールしなければならない場合がある。 また、SAP jConnect バージョン 6.05 以降と SAP 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る。
JZ0I7	`No response from proxy gateway.` 説明: `PROXY` 接続プロパティで指定されたプロキシゲートウェイから応答がないため、接続を確立できない。 対処方法: `PROXY` の設定をチェックして修正する。
JZ0I8	`Proxy gateway connection refused.  Gateway response: %1s.` 対処方法: プロキシゲートウェイの設定をチェックする。
JZ0I9	`This InputStream was closed.` 説明: `getAsciiStream`、`getUnicodeStream`、または `getBinaryStream` から取得した InputStream を読み込もうとしたが、InputStream はすでにクローズしていた。 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る。 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む。

SQL ステータス	メッセージ/説明/対処方法
JZ0IA	`Truncation error trying to send_____.` 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た。 変換後の文字列の長さが、割り付けられたサイズを超えている。 対処方法: SAP jConnect クライアントの CHARSET 接続プロパティで、サーバに送信する必要があるすべての文字をサポートする別の文字セットコードを選択する。 サーバにも別の文字セットをインストールしなければならない場合がある。
JZ0IB	`The server's default charset of _____ does not map to an encoding that is available in the client Java environment. Because jConnect will not be able to do client-side conversion, the connection is unusable and is being closed. Try using a later Java version, or try including your Java installation's i18n.jar or charsets.jar file in the classpath.`
JZ0IR	`getXXX may not be called on a column after it has been updated in the result set with a java.io.Reader.` 対処方法: リーダを使用して更新した ResultSet カラムに対する getXXX の呼び出しを削除する。
JZ0IS	`getXXXStream may not be called on a column after it has been updated in the result set.` 説明: 結果セット内でカラムを更新した後、SybResultSet のメソッド getAsciiStream、getUnicodeStream、getBinaryStream のいずれかを使用して、更新後のカラム値を読み込もうとした。 SAP jConnect は、このような使い方をサポートしていない。 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。
JZ0J0	`Offset and/or length values exceed the actual text/image length.`
JZ0LA	`Failed to instantiate Cipher object. Transformation %1s is not implemented by any of the loaded JCE providers.` 対処方法: CLASSPATH の JCE_PROVIDER_CLASS 接続プロパティで実装が正しく指定されていることを確認する。

SQL ステータス	メッセージ/説明/対処方法
JZ0LC	`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.` 説明: 言語カーソルを使用して作成された 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た。
JZ0MD	`ResultSet metadata is not available.` 対処方法: メタデータストアドプロシージャをインストールする。
JZ0NC	`wasNull called without a preceding call to get a column.` 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すことができる。 対処方法: コードを変更して **wasNull** への呼び出しを移動する。
JZ0NE	`Incorrect URL format. URL: _____. Error message: _____.` 対処方法: URL で、ポート番号に数字のみが使用されていることを確認する。
JZ0NF	`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.`
JZ0NK	`Generated keys are not available because either the Statement.NO_GENERATED_KEYS was used or no keys were automatically generated.` 説明: 文が .NO_GENERATED_KEYS で実行されたか、文が自動生成キーを作成しなかったため、getGeneratedKeys() メソッドは自動生成キーを返せない。 対処方法: RETURN_GENERATED_KEYS で実行された文またはキーを自動生成する文でのみ getGeneratedKeys() を使用する。
JZ0NS	`The method ____ is not supported and should not be called.`

SQL ステータス	メッセージ/説明/対処方法
JZ0P1	`Unexpected result type.` 説明: 文がアプリケーションに返すことができない結果、またはアプリケーションがこの時点で想定していない結果がデータベースから返された。 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す。 JDBC アプリケーションの接続先が SAP Open Server アプリケーションの場合は、SAP 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っていない可能性がある。 対処方法: デバッグツール `com..utils.Debug(true, "ALL")` を使用して、予期しない結果の内容を調べ、原因を明らかにする。
JZ0P4	`Protocol error. This message indicates an internal product problem. Report this error to SAP technical support.`
JZ0P7	`Column is not cached; use RE-READABLE_COLUMNS property.` 説明: `REPEAT_READ` 接続プロパティを false に設定して、カラムを再度読み込もうとしたか、誤った順序でカラムを読み込もうとした。 `REPEAT_READ` が false の場合、ローのカラム値を読み込むことができるのは一度だけである。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る。 たとえば、ローのカラム 3 を読み込んだ後、その値をもう一度読み込んだり、ローのカラム 2 を読み込んだりすることはできない。 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する。また、必ず昇順カラム/インデックス順でカラムを読み込む。
JZ0P8	`The RSMDA Column Type Name you requested is unknown. This is a SAP internal error; please report it to technical support.` 説明: SAP jConnect は、`ResultSetMetaData.getColumnTypeName` メソッドでカラム型の名前を特定できなかった。 対処方法: メタデータ用の最新のストアドプロシージャがデータベースにインストールされていることを確認する。
JZ0P9	`A COMPUTE BY query has been detected. That type of result is unsupported and has been cancelled.` 対処方法: **COMPUTE BY** を使用しないようにクエリまたはストアドプロシージャを変更する。

SQL ステータス	メッセージ/説明/対処方法
JZ0PA	`The query has been cancelled and the same response discarded.` 対処方法: この文と他の文の SQL 例外および警告のチェーンをチェックして原因を調べる。
JZ0PB	`The server does not support a requested operation.` 説明: SAP 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する。サーバは、サポートする機能を SAP jConnect に通知する。 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る。 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場合に、プログラムが `SybConnection.prepareStatement`(*sql_stmt*, *dynamic*) を呼び出し、*dynamic* が true に設定されていると、このメッセージが生成される。 対処方法: サポートされていない機能を要求しないようにプログラムを修正する。
JZ0PC	`The number and size of parameters in your query 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.` 説明: 多数のパラメータを持つ文を実行しようとしているが、サーバが処理するように設定されているパラメータの数を超えている。 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る。 送信するパラメータの数が 481 以下の場合には、この例外は発生しない。 対処方法: SAP Adaptive Server 12.5 以降のサーバに対してこのクエリを実行する。 データベースに接続するときに、`JCONNECT_VERSION` プロパティを “6” に設定する。

SQL ステータス	メッセージ/説明/対処方法

JZ0PD	`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.`
	
	説明: 多数のパラメータを持つ動的準備文を実行しようとしているが、サーバが処理するように設定されているパラメータの数を超えている。
	
	対処方法: SAP Adaptive Server 12.5 以降のサーバに対してこのクエリを実行する。 データベースに接続するときに、JCONNECT_VERSION プロパティを “6” に設定する。

JZ0PE	`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.` 説明: このエラーは、SELECT 文が 255 を超えるカラムから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 (約 65,500 文字を超える) ときに発生することがある。 対処方法: バージョン 12.5 以降の SAP Adaptive Server に対してこのクエリを実行する。 データベースに接続するときに、JCONNECT_VERSION プロパティを “6” に設定する。
JZ0PN	`Specified port number of ____ was out of range. Port numbers must meet the following conditions: 0<= portNumber <=65535.` 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する。
JZ0R0	`Result set has already been closed.` 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する。

SQL ステータス	メッセージ/説明/対処方法
JZ0R1	`Result set is IDLE as you are not currently accessing a row.` 説明: カラムデータを取得するための `ResultSet.getXXX` メソッドのいずれかをアプリケーションが呼び出したが、現在のローがない。アプリケーションが `ResultSet.next` を呼び出していないか、データがないことを示す false が `ResultSet.next` から返された。 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` を呼び出す。
JZ0R2	`No result set for this query.` 説明: `Statement.executeQuery` を使用したが、文からローが返されなかった。 対処方法: ローを返さない文には `executeUpdate` を使用する。
JZ0R3	`Column is DEAD. This is an internal error. Please report it to SAP technical support.`
JZ0R4	`Column does not have a text pointer. It is not a text/image column or the column is NULL.` 対処方法: text/image データをサポートしていないカラムへの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する。また、null の `text/image` カラムを更新しようとしていないことを確認する。 データを挿入してから更新する。
JZ0R5	`The ResultSet is currently positioned beyond the last row. You cannot perform a get* operation to read data in this state.` 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する。
JZ0RD	`You cannot call any of the ResultSet.get* methods on a row that has been deleted with the deleteRow() method.` 対処方法: 削除されたローからデータを取得しないようにアプリケーションのコードを修正する。

SQL ステータス	メッセージ/説明/対処方法
JZ0RM	`refreshRow may not be called after updateRow or deleteRow.` 説明: データベースのローを、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、SybCursorResult.refreshRow を使用してそのローをデータベースからリフレッシュした。 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。
JZ0S0	`Statement state machine: Statement is BUSY.` 説明: このエラーは、Statement.setCursorname メソッド以外では発生しない。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おり、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する。 対処方法: 文にカーソル名を設定してからクエリを実行する。または、文がビジー状態にならないように、Statement.cancel を呼び出してからカーソル名を設定する。
JZ0S1	`Statement state machine: Trying to FETCH on IDLE statement.` 対処方法: 文をクローズして、別の文をオープンする。
JZ0S2	`Statement object has already been closed.` 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する。
JZ0S3	`The inherited method _____ cannot be used in this subclass.` 説明: PreparedStatement は、executeQuery(String)、executeUpdate(String)、orexecute(String) をサポートしない。 対処方法: クエリの文字列を渡すには、PreparedStatement ではなく、Statement を使用する。
JZ0S4	`Cannot execute an empty (zero-length) query.` 対処方法: 空のクエリ (“”) を実行しない。

SQL ステータス	メッセージ/説明/対処方法
JZ0S5	`The local transaction method ____ cannot be used while a global transaction is active on this connection.` 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することがある。 対処方法:「分散トランザクション管理のサポート (115 ページ)」を参照。
JZ0S6	`The local transaction method _____ cannot be used on a pre-System 12 XAConnection.` 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することがある。 対処方法:「分散トランザクション管理のサポート (115 ページ)」を参照。
JZ0S8	`An escape sequence in a SQL Query was malformed: '_____.` 対処方法: JDBC のマニュアルを参照して、正しい構文を確認する。
JZ0S9	`Cannot execute an empty (zero-length) query.` 対処方法: 空のクエリ (“”) を実行しない。
JZ0SA	`Prepared Statement: Input parameter not set, index: _____.` 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する。
JZ0SB	`Parameter index out of range: _____.` 対処方法: クエリのパラメータ数を確認する。
JZ0SC	`Callable Statement: attempt to set the return status as an InParameter.` 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たが、パラメータを 1 に設定しようとしている。この値はリターンステータスを表す。 対処方法: このタイプの呼び出しではパラメータを 2 以上に設定する。
JZ0SD	`No registered parameter found for output parameter.` 説明: これは、アプリケーションの論理エラーを示す。 パラメータに対して `getXXX` または `wasNull` を呼び出したが、まだパラメータを読み込んでいないか、出力パラメータがない。 対処方法: アプリケーションに CallableStatement の出力パラメータが登録されていること、文が実行されていること、出力パラメータが読み込まれたことを確認する。

SQL ステータス	メッセージ/説明/対処方法
JZ0SE	`Invalid object type specified for setObject().` *説明:* `PreparedStatement.setObject` に渡された型引数が正しくない。 *対処方法:* JDBC のマニュアルを確認する。 引数は、`java.sql.Types` からの定数でなければならない。
JZ0SF	`No Parameters expected. Has query been sent?` 説明: パラメータがない文でパラメータを設定しようとした。 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する。

JZ0SG	`An RPC did not return as many output parameters as the application had registered for it.`
	説明: 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した数より多い場合に発生する。 「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 (149 ページ)」を参照。
	対処方法: ストアドプロシージャおよび `registerOutParameter` 呼び出しを確認する。 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する。 そのためには、コードの次の行を調べる。
	`create procedure yourproc (@p1 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 にアップグレードします。

JZ0SH	`A static function escape was used, but the metadata accessor information was not found on this server.` 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する。
JZ0SI	`A static function escape _____ was used which is not supported by this server.` 対処方法: このエスケープを使用しない。
JZ0SJ	`Metadata accessor information was not found on this database.` 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呼び出す。

SQL ステータス	メッセージ/説明/対処方法
JZ0SK	`The oj escape is not supported for this type of database server. Workaround: use server-specific outer join syntax, if supported. Consult server documentation.` 対処方法: メッセージの指示に従うとともに、SAP jConnect メタデータの最新バージョンをインストールする。
JZ0SL	`Unsupported SQL type _____.` 対処方法: 可能であれば、SAP jConnect がサポートする型のパラメータを宣言する。 `Types.NULL` や `PreparedStatement.setObject(null)` は使用しない。

JZ0SM	`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.`

JZ0SN	`setMaxFieldSize: field size cannot be negative.`
JZ0SO	`Invalid ResultSet concurrency type:_____.` 対処方法: 宣言した同時実行性が `ResultSet.CONCUR_READ_ONLY` または `ResultSet.CONCUR_UPDATABLE` であることを確認する。
JZ0SP	`Invalid ResultSet type:_____.` 対処方法: 宣言した ResultSet タイプが `ResultSet.TYPE_FORWARD_ONLY` または `ResultSet.TYPE_SCROLL_INSENSITIVE` であることを確認する。 SAP jConnect は、`ResultSet.TYPE_SCROLL_SENSITIVE` タイプの ResultSet をサポートしていない。
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 `Types.JAVA_OBJECT`、`Types.STRUCT`、`Types.DISTINCT` のいずれかでない場合は、この例外が発生する。 対処方法: 上記の 3 つの UDT のいずれかを使用する。
JZ0SR	`setMaxRows: max rows cannot be negative.`

SQL ステータス	メッセージ/説明/対処方法
JZ0SS	`setQueryTimeout:query timeout cannot be negative.`
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: SAP Adaptive Server Enterprise および SAP SQL Anywhere では、`datetime` と `date` に指定できる値の範囲が異なる。 `datetime` 値の年は 1753 年以降でなければならない。 一方、`date` データ型には 1 以上の年を格納できる。 対処方法: 送信する `date/timestamp` の値が、指定可能な範囲内であることを確認する。
JZ0SV	`Combination of setting parameters by name and by index is not allowed in the same CallableStatement.` 説明: 名前とインデックス (序数で表した位置) で指定されたパラメータが CallableStatement にある。 混合使用は無効である。 対処方法: 名前のみまたはインデックス (序数で表した位置) のみでパラメータを指定する。
JZ0SW	`Invalid ResultSet holdability type: ____.` 説明: **setHoldability()** メソッドで無効な値を指定した。 対処方法: 有効な値のみ (HOLD_CURSORS_OVER_COMMIT または CLOSE_CURSORS_AT_COMMIT) を使用する。
JZ0T2	`Listener thread read error.` 対処方法: ネットワーク通信を確認する。
JZ0T3	`Read operation timed out.` 対処方法: **Statement.setQueryTimeout** を呼び出して、タイムアウト時間を長くする。

SQL ステータス	メッセージ/説明/対処方法
JZ0T4	`Write operation timed out. Timeout in milliseconds: _____.` 対処方法: **Statement.setQueryTimeout** を呼び出して、タイムアウト時間を長くする。
JZ0T5	`Cache used to store responses is full.` 対処方法: `STREAM_CACHE_SIZE` 接続プロパティにデフォルト値またはそれよりも大きい値を使用する。
JZ0T6	`Error reading tunneled TDS URL.` 説明: URL ヘッダの読み込み中に Tunnelled プロトコルが失敗した。 対処方法: 接続に定義した URL を確認する。
JZ0T7	`Listener thread read error -- caught ThreadDeath. Check network connection.` 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する。スレッドが引き続きアボートされる場合には、SAP 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る。
JZ0T8	`Data received for an unknown request. Please report this error to SAP Technical Support.`
JZ0TC	`Attempted conversion between an illegal pair of types.` 説明: Java 型と SQL 型との間の変換に失敗した。 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する。
JZ0TD	`Caught ThreadDeath.` 説明: SAP jConnect の定期 I/O オペレーション実行中に呼び出しアプリケーションスレッドが強制終了された。 対処方法: アプリケーションコードをチェックして競合を検出し、修正する。
JZ0TE	`Attempted conversion between an illegal pair of types. Valid database types are: '_____'.` 説明: データベースカラムのデータ型と `ResultSet.getXXX` 呼び出しで要求されたデータ型は、暗黙的に変換できない。 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する。

SQL ステータス	メッセージ/説明/対処方法
JZ0TI	`jConnect cannot make a meaningful conversion between the database type of _____ and the requested type of _____.` 説明: この種類の例外は、データベースから返された `time` 値に対して、アプリケーションが `ResultSet.getObject(int, Types.DATE)` を呼び出そうとした場合などに発生することがある。 対処方法: データベースのデータ型が、取得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する。
JZ0TO	`Read operation timed out.` 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する。 対処方法: `Statement.setQueryTimeout` を呼び出して、タイムアウト時間を長くする。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する。
JZ0TS	`Truncation error trying to send _____.` 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る。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る。 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する。
JZ0US	`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://”.`
JZ0XC	`_____ is an unrecognized transaction coordinator type.` 説明: メタデータ情報によると、サーバは分散トランザクションをサポートしているが、SAP jConnect は使用されているプロトコルをサポートしていない。 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する。エラーが解消されない場合は、SAP 製品の保守契約を結んでいるサポートセンタに連絡する。

SQL ステータス	メッセージ/説明/対処方法
JZ0XS	`The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.` 説明: SAP 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ない。 対処方法: このサーバでは `XADataSource` を使用しない。または、サーバをアップグレードするか、分散トランザクションを使用できるように設定する。
JZ0XU	`Current user does not have permission to do XA-style transactions. Be sure user has _____ role.` 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ない。ほとんどの場合は、ユーザに適切な役割が付与されていないことが原因である。 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する。
JZBK1	`SybBCP class is NOT initialized Please Re-Run MDA sqls to update the MDA stored procedures.` 対処方法: MDA ストアドプロシージャをインストールする。
JZBK3	`Bulk load table does not exists.` 対処方法: テーブル名を修正する。
JZBK4	`Illegal usage of sql statements mixed with batches in bcp/arrayinsert mode.` 説明: バッチオペレーション中に非バッチオペレーションを実行しようとしている。 対処方法: バッチオペレーションが完了してから非バッチオペレーションを実行する。
JZBK5	`autocommit should be set to true when running bulk load in bcp mode.`
JZBK6	`Adaptive Server 15.7 or latter and 'allow wide dol rows' DB option must be enabled to insert rows with offsets greater than 8191.`
JZBK7	`Failed to insert data. Total data size (_____ bytes) exceeds the maximum row size (_____ bytes)allowed for the table _____.`

SQL ステータス	メッセージ/説明/対処方法
JZBKI	`Invalid value set for property ENABLE_BULK_LOAD.` 対処方法: ENABLE_BULK_LOAD を有効な値 (ARRAYINSERT_WITH_MIXED_STATEMENTS、ARRAYINSERT、BCP、LOG_BCP) のいずれかのみに設定する。
JZNNA	`Column does not allow null values.` 説明: 準備文で setNull() を使用して BIT 型カラムを NULL 値に設定しようとした。 対処方法: クエリを確認し、BIT 型カラムに値 0 または 1 を設定して修正する。
S0022	`Invalid column name '_____'.` 対処方法: カラム名のスペルが正しいこと、および指定したカラムが存在することを確認する。
ZZ00A	`The method _____ has not been completed and should not be called.` 対処方法: 使用している SAP jConnect のバージョンに付属しているリリースノートで詳細情報を確認する。 SAP 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 ページを参照する。 実装されていない場合は、そのメソッドを使用しない。

用語解説

SAP jConnect for JDBC で使用される用語を解説します。

- **アプリケーションプログラムインタフェース (API: application program interface)** – ソフトウェアコンポーネントが相互通信するためのインタフェースとして使用することを目的としたソースコードベースの仕様。
- **SAP Adaptive Server Enterprise** – 複数のデータベースおよび複数のユーザの管理、ディスク上のデータの実際のロケーションの監視、論理データ記述から物理データ記憶領域へのマッピングの管理、およびメモリ内のデータキャッシュとプロシージャキャッシュの保守を行います。
- **Certicom Security Builder GSE-J** – FIPS 140-2 検証済み暗号アルゴリズムをサポートする JCE (Java Cryptography Extension) ソフトウェア暗号プロバイダ。
- **CyberSafe TrustBroker** – jConnect で使用可能な GSS (Generic Security Services) Manager。
- **データベースサーバ** – クライアントまたはサーバのアーキテクチャを使用する、データベースアプリケーションのバックエンドシステム。
- **データ型** – 変数に有効な型、値、および演算を表す定義属性。
- **デッドロック** – データにロックを保持している 2 人のユーザが、互いに他方のデータのロックを獲得しようとしたときに起こる状態。
- **DirectConnect** – SAP 以外のデータソースへの基本的な接続を提供する ECDA コンポーネント。 特に、DirectConnect Manager によるアクセス管理、トランザクション管理、リモートシステム管理を提供します。
- **識別名 (DN)** – ディレクトリサーバのエントリをユニークに識別する文字列。DN は、ディレクトリ情報ツリー (DIT) のエントリの位置を識別する相対識別名 (RDN) コンポーネントを持たない場合と、1 つ以上持つ場合があります。DN は、名前と階層位置の両方を指定するという点で、ファイルシステムの絶対パスと類似しています。
- **GSS ライブラリ** – 汎用セキュリティサービスアプリケーションプログラムインタフェース (GSS-API: Generic Security Service Application Program Interface) を実装するライブラリ。
- **IETF** – Internet Engineering Task Force。 インターネットの主要標準化機構であり、 インターネットアーキテクチャの発展とインターネットの円滑な運用に関心を持つネットワークエンジニア、オペレータ、ベンダ、研究者の大規模でオープンな国際的コミュニティです。
- **SAP jConnect ドライバ** – TDS (Tabular Data Stream) 通信プロトコルを使用する SAP サーバ (Adaptive Server Enterprise など) 向けの JDBC ドライバ。

- **JCE (Java Cryptography Extension)** – Java のセキュリティ機能を実装するための統一フレームワークを提供する API。
- **JDBC** – Java Database Connectivity。 JDBC は、Java プログラムによる SQL 文の実行を可能にする Java API です。
- **JDK** – Java Development Kit。 Java プログラムを作成するための SDK です。
- **Java GSS (Generic Security Services) Manager** – 認証とセキュアメッセージングの汎用インタフェースを提供します。
- **JNDI** – Java Naming and Directory Interface。 JNDI を使用すると、Java プラットフォームベースのアプリケーションが複数のネーミングサービスおよびディレクトリサービスにアクセスできます。

  JNDI は、DNS、LDAP、NDS などのネーミングサービスおよびディレクトリサービスに Java プログラムを接続するための Oracle の API です。
- **JRE (Java Runtime Environment)** – Java アプリケーションを開発するためのプログラミングツールセットである JDK (Java Development Kit) の一部。 Java Runtime と呼ばれることもあります。
- **JTA (Java Transaction API)** – アプリケーションと J2EE サーバにトランザクションへのアクセスを許可する API。
- **JTS (Java Transaction Service)** – JTA (Java Transaction API) をサポートするトランザクションマネージャの実装を指定し、API より下位の Object Management Group Object Transaction Service 1.1 仕様の Java マッピングを実装します。
- **Java 仮想マシン (JVM)** – Java バイトコードをマシン言語に変換して実行する、プラットフォームに依存しない実行環境を提供する仮想マシン。
- **J2EE** – Java 2 Platform Enterprise Edition。 J2EE は、Web ベースのエンタープライズアプリケーションをオンラインで開発、構築、および展開するための、プラットフォームに依存しない Java を中心とした環境です。
- **Kerberos** – コンピュータネットワーク内のサービスに対する要求のセキュアな認証方式。
- **KDC (Key Distribution Center)** – 認証作業とチケット生成作業を実行するシングルサインオン (SSO) の一部。
- **ラージオブジェクト (LOB) データ型** – 通常はラージ文字オブジェクト (text) またはバイナリオブジェクト (image)。
- **ラージオブジェクト (LOB) ロケータ** – データ自体ではなく、LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。
- **LDAP** – Lightweight Directory Access Protocol。 LDAP は、公開されたインターネットでも企業のイントラネットでも、ネットワーク内の組織、個人、ファイルやデバイス等のリソースを誰でも見つけることができるようにするためのソフトウェアプロトコルです。

- **LDIF (LDAP Data Interchange Format) –** ディレクトリデータをテキスト形式で表すメカニズム形式。 LDIF 仕様は RFC 2849 に含まれ、ディレクトリデータを表す形式だけでなく、そのデータを変更するためのメカニズムを記述します。
- **ネイティブプロトコル –** クライアント/サーバ間の要求/応答交換のために DBMS がサポートするネイティブプロトコル。
- **ネットプロトコル –** 中間層ゲートウェイとデータベース間の要求と応答の送受信に使用するプロトコル。
- **オブジェクト識別子 (OID) –** オブジェクトの命名に使用される識別子。 構造的に、OID は階層的に割り当てられたネームスペースのノードを含みます。
- **プライマリサーバ –** 高可用性 (HA) 環境では、プライマリサーバはクライアントが最初に接続を試行するサーバです。
- **Replication Server® –** 複数のデータベースの複写データを管理し、データの整合性と一貫性を維持します。 複写システム内のデータベースを使用するクライアントには、ローカルデータへのアクセスが提供されるので、ネットワークと集中管理されたコンピュータシステムの負荷が軽減されます。
- **相対識別名 (RDN) –** 識別名内の 1 つの構成要素。 RDN は、名前と値が等号で区切られた 1 つまたは複数のペアで構成されます (たとえば、RDN が "*uid=ann*" の場合、名前が "*uid*" で値が "*ann*")。名前と値のペアが複数ある場合は、プラス記号で区切られます (たとえば、RDN が "*cn=Jon Doe+employeeNumber=12345*" の場合、名前と値のペアは "*cn=John Doe*" と "*employeeNumber=12345*")。 実際には、名前と値のペアが複数ある RDN (*複数値 RDN*) はまれですが、エントリにユニークな属性がない場合や、エントリの DN に有益な識別情報が確実に含まれるようにする場合には役立つことがあります。
- **RPC –** リモートプロシージャコール。 ネットワークの詳細を理解していなくても、1 つのプログラムがネットワーク上の別のコンピュータにあるプログラムにサービスを要求するために使用できるプロトコル (プロシージャコールは、ファンクションコールまたはサブルーチンコールと呼ぶ場合もあります)。 RPC はクライアント/サーバモデルを使用します。 要求側プログラムはクライアント、サービス提供側プログラムはサーバです。 通常またはローカルのプロシージャコールと同様に、RPC は同期オペレーションであるため、リモートプロシージャの結果が返されるまで要求側プログラムを中断する必要があります。 ただし、同一のアドレス空間を共有する軽量なプロセスまたはスレッドを使用することで、複数の RPC を同時に実行することができます。
- **RSA 暗号化 –** 高度にセキュアな暗号方式。
- **セカンダリサーバ –** 高可用性 (HA) 環境では、セカンダリサーバはクライアントがプライマリサーバとの接続に失敗した場合に接続を試行するサーバです。
- **シングルサインオン (SSO) –** ユーザが 1 つの名前とパスワードを入力すれば複数のアプリケーションにアクセスできるセッションまたはユーザ認証プロセス。 権限を持つすべてのアプリケーションに対してユーザが認証され、特定

のセッション中にアプリケーションを切り替えても名前やパスワードの入力を求められることはありません。

- **SAP SQL Anywhere** – 豊富な機能を備えたリレーショナルデータベースおよびデータ管理ツール。
- **SSL** – Secure Sockets Layer。 SSL は、インターネット上のメッセージ転送のセキュリティ管理に広く使用されているプロトコルです。
- **SAP IQ** – データウェアハウジングに特化して設計された高性能な意思決定支援サーバ。

  SAP IQ は、SAP Adaptive Server Enterprise や SAP SQL Anywhere を含む SAP 製品群の一部です。 SAP IQ 内のコンポーネント統合サービスは、メインフレーム、UNIX、または Windows サーバ上のリレーショナルデータベースおよび非リレーショナルデータベースへの直接アクセスを提供します。
- **TDS (Tabular Data Stream)** – 2 台のコンピュータ間のデータ転送を記述するアプリケーションレベルプロトコルで、 送信可能なメッセージのタイプおよびメッセージの送信順序を定義します。 TDS は、接続指向型のトランスポートサービスを利用します。
- **TDS トンネリングサーブレット** – HTTP パケットまたは HTTPS パケットを介して TDS ストリームを渡すサーブレット。
- **UCS-2** – 汎用文字セット (UCS: Universal Character Set) は、文字セットのコード化に使用する ISO/IEC フォーマットです。 ISO/IEC 10646 は Unicode と同期していましたが、Unicode に制約が追加されたため、10646 に準拠していても Unicode との互換性は保証されなくなりました。
- **UTF-16** – UTF-16 (Unicode Transformation Format-16) は、Unicode コーディングシステムの 2 バイトのフォーマットです。
- **Wedgetail JCSI** – SAP jConnect で使用可能な GSS (Generic Security Services) Manager。

索引

K

P

R

S

T

U

W

あ

い

う

え

お

か

け

こ

さ

し

す

せ

そ

ち

て

と

は

ひ

ふ