InterSystems ソフトウェアでの Java の使用法

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

InterSystems ソフトウェアでの Java の使用法
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

InterSystems での Java の概要

このドキュメントで取り上げる内容の詳細なリストは、"目次" を参照してください。

InterSystems IRIS® には、広範囲にわたる堅牢な Java 接続オプションが用意されています。

- JDBC、Java オブジェクト、または InterSystems 多次元ストレージを使用したデータベース・アクセスを提供する軽量の SDK
- InterSystems IRIS サーバ・アプリケーションから Java アプリケーションおよび外部データベースに直接アクセスできるようにするゲートウェイ
- Hibernate などのサード・パーティ・ソフトウェアの実装。

これらの Java ソリューションはすべて、InterSystems **JDBC ドライバ**に支えられています。これは JDBC の完全に準拠した強力なタイプ 4 (純正 Java) の実装で、InterSystems IRIS と緊密に連携して最大の速度と効率性を実現します。

このドキュメントの最初のセクションでは、JDBC ドライバによって有効になるすべての InterSystems IRIS Java テクノロジについて概説します。

- "InterSystems Java 接続オプション" では、InterSystems Java ソリューションの概要を示します。

ドキュメントの残りの部分では、JDBC ドライバ自体の使用方法について詳しく説明します。

- "JDBC ドライバの使用法" では、InterSystems IRIS または外部データベースへの JDBC 接続を確立するさまざまな方法について詳しく説明します。
- "構成と要件" では、クライアント構成および InterSystems Java クラス・パッケージの詳細を示します。
- "JDBC の基本" では、JDBC の概要を示し、頻繁に使用するクラスとメソッドの例を示します。
- "JDBC クイック・リファレンス" では、InterSystems 固有の拡張メソッドについて説明します。

関連ドキュメント

以下のドキュメントには、InterSystems IRIS が提供する Java ソリューションに関する詳細が記載されています。

- "サードパーティ・ソフトウェア用インターシステムズ実装リファレンス" の "JDBC ドライバのサポート" では、InterSystems JDBC ドライバのサポートおよび準拠に関する詳細が説明されており、すべてのオプション機能のサポート・レベルおよび InterSystems IRIS 固有の追加機能の全リストも記載されています。
- "Native SDK for Java の使用法" では、以前は ObjectScript を介してのみ使用可能であったリソースに、Java Native SDK を使用してアクセスする方法について説明しています。
- "InterSystems XEP による Java オブジェクトの永続化" では、Event Persistence API (XEP) を使用して Java オブジェクトの永続性を迅速に実現する方法を説明しています。

2

InterSystems Java 接続オプション

InterSystems IRIS® には、広範囲にわたる堅牢な Java 接続オプションが用意されています。

- 主要データ・アクセス用 SDK は、リレーショナル・テーブル、オブジェクト、またはグローバルを使用した軽量のデータ・アクセスを提供します。
- InterSystems 外部サーバは、同じコンテキストおよび接続内で、ObjectScript オブジェクトと Java オブジェクトの両方に容易にアクセスして操作する方法を提供します。
- InterSystems SQL ゲートウェイは、SQL インタフェース経由での外部データベースおよび Java アプリケーションへのカスタマイズされた接続を提供します。
- サードパーティのフレームワークのサポートには、Hibernate 向けのインタフェースの実装が含まれます。

InterSystems **JDBC driver** は、すべての InterSystems IRIS Java ソリューションの中心にあります。これは、JDBC の完全に準拠した強力なタイプ 4 (純正 Java) の実装で、InterSystems IRIS と緊密に連携して最大の速度と効率性を実現します。

2.1 主要データ・アクセス用 SDK

InterSystems JDBC ドライバは、リレーショナル・テーブル、オブジェクト、および多次元ストレージを使用した InterSystems IRIS データベースへの直接アクセスを提供する 3 つの軽量 Java SDK をサポートしています。

重要　主要データ・アクセス用 SDK と外部サーバ・ゲートウェイは、すべてが同じ基盤となる再入可能接続コンテキストを共有できる統合型ユーティリティ・スイートを構成します。スイートの任意の部分から必要な機能を自由に組み合わせて、アプリケーションで使用できます。

リレーショナル・テーブル・アクセス用 JDBC ドライバ

InterSystems JDBC ドライバ (このドキュメントで説明しています) は、リレーショナル・テーブルへの SQL ベースのアクセスを提供します。これは、以下の機能をサポートします。

- リレーショナル・アクセス
  - SQL を介したテーブルの保存とクエリ
  - 保存された JDBC テーブルに InterSystems IRIS オブジェクトとしてアクセス可能
- InterSystems IRIS に最適化された JDBC ドライバ
  - 完全に実装されたタイプ 4 (純正 Java) の JDBC

  - InterSystems IRIS に固有のプロパティ設定のための拡張
  - 高速なバッチ読み取り
  - 自動接続プーリング

- サードパーティの Java フレームワークのサポート
  - Hibernate のサポート

詳細は、このドキュメントの後半の章と、"サードパーティ・ソフトウェア用インターシステムズ実装リファレンス" の "JDBC ドライバのサポート" の章で提供しています。

オブジェクト・アクセス用 XEP SDK

InterSystems XEP SDK は、データ・オブジェクトをきわめて高速にリアルタイムに取得するために設計されており、便利な汎用 ORM インタフェースとしても使用できます。これは、以下の機能をサポートします。

- 処理速度のために最適化
  - 超高速のリアルタイムのデータ取得。XEP API は、標準の JDBC の数倍も速くデータを取得できます。
  - バッチ読み取り
  - データのシリアル化の細かい制御
- オブジェクト・ベースのアクセス
  - Hibernate に代わる軽量な選択肢
  - オブジェクトの保存とクエリ (作成/読み取り/更新/削除)
  - スキーマのインポートとカスタマイズ
  - ほとんどの標準データ型のマッピング
  - 保存されたオブジェクトに JDBC テーブルとしてもアクセス可能
- 完全なプロセス・コントロール
  - インデックス作成とフェッチ・レベルの制御
  - トランザクションとロックの制御

詳細は、"InterSystems XEP による Java オブジェクトの永続化" を参照してください。

InterSystems IRIS リソースへの直接アクセスのための Native SDK

InterSystems Native SDK for Java は、以前は ObjectScript を介してのみ使用可能であったリソースに、Java アプリケーションを通して直接アクセスできるようにする軽量ツールセットです。これは、以下の機能をサポートします。

- グローバル配列への直接アクセスと操作
  - ノードの作成と削除
  - ノードの反復処理、および値の作成/読み取り/更新/削除
  - トランザクションとロックの制御
- サーバ側 ObjectScript コードの呼び出し :
  - コンパイル済みクラスからのメソッドおよびアクセス・プロパティの呼び出し

- コンパイルされた任意の .mac ファイルからの関数またはプロシージャの呼び出し

- ObjectScript クライアントが外部サーバ・ゲートウェイを介して Java オブジェクトに直接アクセスできるようにする Java サーバ・アプリケーションを作成します。

詳細は、"Native SDK for Java の使用法" を参照してください。

2.2 InterSystems 外部サーバ

InterSystems 外部サーバは、同じ接続を使用して、同じコンテキストで ObjectScript オブジェクトと Java オブジェクトの両方に容易にアクセスして操作する方法を提供します。外部サーバ・ゲートウェイは完全に再入可能であるため、同じ双方向接続とコンテキスト (データベース、セッション、およびトランザクション) を使用して、Java アプリケーションで ObjectScript オブジェクトを操作し、ObjectScript アプリケーションで Java オブジェクトを操作することができます。

ObjectScript アプリケーションは、ObjectScript コードを大幅に変更することなく、ODBC を代替接続オプション (.NET オブジェクトと ADO へのアクセスを提供) として使用することもできます。

詳細は、"InterSystems 外部サーバの使用法" を参照してください。

2.3 InterSystems SQL ゲートウェイ

InterSystems SQL ゲートウェイを使用すると、JDBC 経由で InterSystems IRIS を外部データベースに接続できます。さまざまなウィザードを使用して、外部ソースのテーブル、ビュー、またはストアド・プロシージャへのリンクを作成できます。これにより、オブジェクトや SQL クエリを使用して、InterSystems IRIS 上の場合と同じように外部データベースのデータを読み取ったり保存したりできます。対応する外部ストアド・プロシージャと同じアクションを実行するクラス・メソッドを生成することさえできます。

SQL ゲートウェイ・アプリケーションは ObjectScript で記述され、サーバ上で実行されます。ObjectScript コードを大幅に変更することなく、ODBC を代替接続オプション (.NET オブジェクトと ADO へのアクセスを提供) として使用することもできます。

JDBC および ODBC オプションの詳細は、"SQL ゲートウェイの使用法" を参照してください。

2.4 サード・パーティのフレームワークのサポート

Hibernate などの Java フレームワークは、JDBC を使用してデータベースを操作し、特定のデータベースに固有の機能を利用するために実装できるインタフェースを含みます。InterSystems IRIS は、Hibernate 言語インタフェースの実装を提供します。

Hibernate 言語

InterSystems Hibernate 言語は、Hibernate 言語インタフェースの完全に準拠した実装で、Hibernate と InterSystems IRIS との間のカスタマイズされたインタフェースを提供します。ほとんどの主な言語実装と同様に、これは Hibernate ディストリビューションに含まれています。

詳細は、"サードパーティ・ソフトウェア用インターシステムズ実装リファレンス" の "Hibernate のサポート" の章を参照してください。

3

JDBC ドライバの使用法

この章では、アプリケーションと InterSystems IRIS 間で JDBC 接続を確立する方法と、JDBC ドライバの拡張メソッドとプロパティを使用する方法について説明します。

- JDBC 接続の確立 ― DriverManager または DataSource を使用して接続を確立および制御する方法について説明します。
- バルク挿入のための列方向のバインディング ― バッチ挿入をより速く、使いやすくする拡張メソッドについて説明します。
- 接続プーリング ― 接続プーリングと監視のオプションについて説明します。
- 最適化とテストのオプション ― ログ、共有メモリ、および文プーリングについての情報を提供します。

"Connecting Your Application to InterSystems IRIS" にも、サンプル・コードを含め、JDBC を使用して Java アプリケーションから InterSystems IRIS サーバに接続する手順が示されています。

3.1 JDBC 接続の確立

このセクションでは、DriverManager または DataSource を使用して接続を確立および制御する方法について説明します。

- JDBC 接続 URL の定義 ― JDBC 接続を定義するパラメータの指定方法を説明します。
- 接続のための IRISDataSource の使用法 ― IRISDataSource を使用してドライバをロードし、java.sql.Connection オブジェクトを作成する方法について説明します。
- 接続のための DriverManager の使用法 ― DriverManager クラスを使用して接続を作成する方法について説明します。

3.1.1 JDBC 接続 URL の定義

java.sql.Connection URL は、アクセス先のホスト・アドレス、ポート番号、およびネームスペースに関する情報を持つ接続を提供します。InterSystems JDBC ドライバでは、オプションの URL パラメータもいくつか使用できます。

3.1.1.1 必須の URL パラメータ

最小限必要な URL 構文は以下のとおりです。

```
jdbc:IRIS://<host>:<port>/<namespace>
```

ここで、必須パラメータは以下のように定義されます。

- host ー IP アドレス、または完全修飾ドメイン名 (FQDN)。例えば、`127.0.0.1` と `localhost` は両方ともローカル・マシンを表します。
- port ー InterSystems IRIS スーパーサーバが待ち受け状態にある TCP ポート番号。既定値は `1972` です。詳細は、"構成パラメータ・ファイル・リファレンス" の "DefaultPort" を参照してください)。
- namespace ー アクセス先の InterSystems IRIS ネームスペース。

例えば、以下の URL では、host が `127.0.0.1`、port が `1972`、namespace が `User` に指定されます。

```
jdbc:IRIS://127.0.0.1:1972/User
```

3.1.1.2 オプションの URL パラメータ

host、port、および namespace に加えて、オプションの URL パラメータをいくつか指定することもできます。完全な構文は、以下のとおりです。

```
jdbc:IRIS://<host>:<port>/<namespace>/<logfile>:<eventclass>:<nodelay>:<ssl>
```

ここで、オプションのパラメータは以下のように定義されます。

- logfile ー JDBC ログ・ファイルを指定します (“JDBC のログ” を参照してください)。
- eventclass ー この IRISDataSource オブジェクトにトランザクション・イベント・クラスを設定します。
- nodelay ー IRISDataSource オブジェクト経由で接続している場合に `TCP_NODELAY` オプションを設定します。このフラグを切り替えると、アプリケーションのパフォーマンスに影響する場合があります。有効な値は、`true` および `false` です。設定されていない場合、既定値として `true` が設定されます。
- ssl ー IRISDriver と IRISDataSource の両方で TLS を有効にします ("セキュリティ管理ガイド" の “TLS の構成” を参照)。有効な値は、`true` および `false` です。設定されていない場合、既定値として `false` が設定されます。

これらのオプションの URL パラメータは、他のパラメータを指定しなくても、それぞれ個別に定義できます。例えば、以下の URL は、必須のパラメータと nodelay オプションのみを設定します。

```
jdbc:IRIS://127.0.0.1:1972/User/::false
```

他の接続プロパティを指定するには、Properties オブジェクト内の DriverManager に渡します (“接続のための DriverManager の使用法” を参照してください)。

3.1.2 接続のための IRISDataSource の使用法

com.intersystems.jdbc.IRISDataSource を使用してドライバをロードした後、java.sql.Connection オブジェクトを作成します。これは、データベースに接続する推奨のメソッドで、InterSystems IRIS で完全にサポートされています。

IRISDataSource との接続の開き方

以下の例では、ドライバをロードしてから IRISDataSource を使用して接続を作成し、ユーザ名およびパスワードを指定します。

```
try{
  IRISDataSource ds = new IRISDataSource();
  ds .setServerName("127.0.0.1");
  ds .setPortNumber(51776);
  ds .setDatabaseName("USER");
  ds .setUser("_SYSTEM");
  ds .setPassword("SYS");
  IRISConnection connection = (IRISConnection) ds.getConnection();
}
catch (SQLException e){
  System.out.println(e.getMessage());
}
catch (ClassNotFoundException e){
  System.out.println(e.getMessage());
}
```

この例では、意図的に `localhost` ではなくリテラル・アドレス `127.0.0.1` を使用します。ホスト名が Ipv4 と Ipv6 で同じように解決されるシステムでは、`localhost` を使用する場合、Java は Ipv6 経由で接続しようとします。

注釈 IRISDataSource クラスは、接続プロパティ・アクセサの拡張セットを提供します (この例では、setUser()、setPassword() など)。アクセサの完全なリストについては、クイック・リファレンスの “IRISDataSource クラス”、およびすべての接続プロパティの詳細は、このリファレンスの後半にある “接続パラメータのオプション” を参照してください。

3.1.3 接続のための DriverManager の使用法

IRISDataSource を使用して接続することをお勧めしますが、DriverManager クラスを使用して接続を作成することもできます。以下のコードは、その方法の一例です。

```
Class.forName ("com.intersystems.jdbc.IRISDriver").newInstance();
   String  url="jdbc:IRIS://127.0.0.1:1972/User";
   String  username = "_SYSTEM";
   String  password = "SYS";
   dbconnection = DriverManager.getConnection(url,username,password);
```

以下のコードに示すように、DriverManager の接続プロパティを Properties オブジェクトで指定することもできます。

```
String  url="jdbc:IRIS://127.0.0.1:1972/User";
   java.sql.Driver drv = java.sql.DriverManager.getDriver(url);

   java.util.Properties props = new Properties();
   props.put("user",username);
   props.put("password",password);
   java.sql.Connection dbconnection = drv.connect(url, props);
```

利用できるプロパティの詳細なリストは、このリファレンスの後半にある “接続パラメータのオプション” を参照してください。

3.2 バルク挿入のための列方向のバインディング

JDBC では、事前入力されたデータのバルク挿入は通常、addBatch() をループで呼び出すことによって実行されます。これは、データが既に配列内にあり、サーバに送信する準備ができている場合には最適ではありません。InterSystems IRIS は、ループをバイパスして、配列全体を 1 つの setObject() 呼び出しで渡すことができる拡張機能を提供します。

例えば、一般的なコードは、以下のように各項目に対して setObject() を呼び出します。

```
// Typical AddBatch() loop
  for (int i=0;i<10000;i++){
    statement.setObject(1,objOne);
    statement.setObject(2,objTwo);
    statement.setObject(3,objThree);
    statement.addBatch();
  }
  statement.executeBatch();
```

すべての項目を 1 つの Object 配列に読み込んで、配列全体を 1 つの呼び出しで追加すれば、コードはより高速かつ単純になります。

```
// Adding an ArrayList named objArray with a single call
  IRISPreparedStatement.setObject(objArray);
  statement.addBatch();
  statement.executeBatch();
```

列方向のバインディングでは、arraylist としてバインドされた最初のパラメータが、その arraylist のサイズ (複数存在する場合) を使用してバッチ内の行数を表すことが前提となります。arraylist としてバインドされたその他のパラメータは、同じサイズであるか、または 1 つの値のみ (ユーザ定義の既定値など) を指定している必要があります。そうでない場合、addbatch() の呼び出し時に、″ #rows # parameter! #count ″という例外がスローされます。

例えば、3 つのパラメータと 10 個の行をバインドする場合、パラメータ 1 で 10 個の値を arraylist にバインドできます。パラメータ 2 と 3 には同様に arraylist で 10 個の値を指定するか、1 つの値のみを指定する (すべての行に 1 つの既定値を指定する場合) 必要があります。すべてを入力するか、1 つのみ入力する必要があります。それ以外では例外がスローされます。

以下に、行方向のバインディングと列方向のバインディングで同じ操作を行う例を示します。

bindRowWise()

```
        public static void bindRowWise() throws Exception {

          int rowCount = cName.size();
          String insert = "INSERT INTO CUSTOMER VALUES(?,?,?,?)";
          try {
            PreparedStatement ps = conn.prepareStatement(insert);
            for (int i=0;i<rowCount;i++){
              ps.setObject(1,cName.get(i));
              ps.setObject(2,cAddress.get(i));
              ps.setObject(3,cPhone.get(i));
              ps.setObject(4,cAcctBal.get(i));
              ps.addBatch();
            }
            ps.executeBatch();
          }
          catch (Exception e) {
            System.out.println("\nException in RowBinding()\n"+e);
          }

        } // end bindRowWise()
```

bindColumnWise()

```
public static void bindColumnWise() throws Exception {

  String insert = "INSERT INTO CUSTOMER VALUES(?,?,?,?)";
  try {
    PreparedStatement ps = conn.prepareStatement(insert);
    ps.setObject(1, new ArrayList<>(cName)    );
    ps.setObject(2, new ArrayList<>(cAddress) );
    ps.setObject(3, new ArrayList<>(cPhone)   );
    ps.setObject(4, new ArrayList<>(cAcctBal) );
    ps.addBatch();
    ps.executeBatch();
  catch (Exception e) {
    System.out.println("\nException in bindColumnWise()\n"+e);
  }

} // end bindColumnWise()
```

3.3 接続プーリング

3.3.1 IRISConnectionPoolDataSource メソッドの使用法

IRISConnectionPoolDataSource クラスは、ConnectionPoolDataSource を実装し、プールされた接続のテストと監視に役立つ一連の独自の拡張機能によってそれを拡張したものです。以下の拡張機能を使用できます。

- getConnectionWaitTimeout() 接続プール・マネージャが、接続が利用可能になるまで待機する秒数を返します。
- getMaxPoolSize() 許可される接続の最大数を返します。
- getPoolCount() 接続プール内の現在のエントリ数を返します。
- restartConnectionPool() 物理接続をすべて切断し、接続プールを空にします。
- setMaxPoolSize() プール内で許可される接続の最大数を指定する int 値を取ります。既定値は 40 です。
- setConnectionWaitTimeout() 接続の待機タイムアウト間隔を秒数で指定する int 値を取ります。タイムアウト時間が過ぎても利用可能な接続がない場合は、例外がスローされます。既定では 0 に設定されています。これは、接続が即時に利用可能になることを示します。プールがいっぱいである場合は例外がスローされます。

IRISConnectionPoolDataSource クラスは、IRISDataSource に実装された独自の拡張機能も継承します (クイック・リファレンスの “接続パラメータのオプション” を参照)。

InterSystems IRIS でこのクラスを使用する手順は次のとおりです。

1. 必要なパッケージをインポートします。

```
import com.intersystems.jdbc.*;
import java.sql.*;
```

2. IRISConnectionPoolDataSource オブジェクトをインスタンス化します。reStart() メソッドを使用して物理接続をすべて切断し、プールを空にします。setURL() (IRISDataSource から継承) を使用して、プールの接続にデータベース URL (″JDBC 接続 URL の定義″ を参照) を設定します。

```
IRISConnectionPoolDataSource pds = new IRISConnectionPoolDataSource();
   pds.restartConnectionPool();
   pds.setURL("jdbc:IRIS://127.0.0.1:1972/User");
   pds.setUser("_system");
   pds.setPassword("SYS");
```

3. 最初は、getPoolCount() は 0 を返します。

```
System.out.println(pds.getPoolCount()); //outputs 0.
```

4. getConnection() を使用して、プールからデータベース接続を取得します。

```
Connection dbConnection = pds.getConnection();
```

注意　InterSystems JDBC ドライバ接続は、常に IRISDataSource の getConnection() メソッドを呼び出して取得する必要があります。このメソッドは、自動の透過的な接続プーリングを提供するために拡張されています。**ConnectionPoolDataSource**.getPooledConnection() メソッドは、JDBC 標準で要求されているため実装されていますが、直接呼び出してはなりません。

5. 接続を切断します。getPoolCount() が 1 を返すようになります。

```
dbConnection.close();
System.out.println(pds.getPoolCount()); //outputs 1
```

3.4 最適化とテストのオプション

このセクションには、開発およびテストの際に役立つ可能性のある専用の情報が含まれています。

- JDBC のログ ー JDBC アプリケーションをテストする際にログを有効にする方法を説明します。
- 共有メモリ接続ー サーバとクライアントが同じマシン上にある場合に接続がどのように機能するかについて説明します。
- 文のプーリング ー 文を初めて使用するときに、最適化された文をキャッシュに格納する方法について説明します。

3.4.1 JDBC のログ

アプリケーションで問題が発生した場合、ログを有効にしてアプリケーションを監視できます。アプリケーションを実行し、エラー条件をトリガしたことを確認した後、エラー・メッセージや、異常な活動のすべてのログ記録をチェックします。エラーの原因は通常、メッセージ内に表れています。

注釈　ログは、トラブルシューティングを実行する必要がある場合のみ有効にします。ログを有効にするとパフォーマンスが大幅に低下するため、通常の操作時は有効にしないでください。

InterSystems IRIS に接続するときに JDBC のログを有効にするには、JDBC 接続文字列の末尾にログ・ファイル名を追加します。接続時に、ドライバによってアプリケーションの作業ディレクトリに保存されるログ・ファイルが保存されます。

例えば、元の接続文字列が以下であるとします。

```
jdbc:IRIS://127.0.0.1:1972/USER
```

ログを有効にするには、この文字列を以下のように変更して再接続します。

```
jdbc:IRIS://127.0.0.1:1972/USER/myjdbc.log
```

このログには、InterSystems IRIS データベースから見た対話処理が記録されます。

指定されたログ・ファイルが存在する場合、既定では新しいログ・エントリがそのファイルに追加されます。既存のファイルを削除して、新しいファイルを作成するには、接頭語としてログ・ファイル名の前にプラス (+) 文字を付けます。例えば、次の文字列では、myjdbc.log を削除し (既存の場合)、同じ名前で新しいログ・ファイルを作成します。

```
jdbc:IRIS://127.0.0.1:1972/USER/+myjdbc.log
```

3.4.2 共有メモリ接続

InterSystems IRIS では、Java アプリケーションが InterSystems IRIS サーバ・インスタンスと同じマシン上で実行されている場合、TCP/IP ではなく共有メモリ接続が使用されます。このセクションでは、共有メモリが動作するしくみと、開発およびテストの目的で共有メモリを無効にする方法について説明します。

共有メモリ接続は、高いコストのかかる可能性があるカーネル・ネットワーク・スタックの呼び出しを回避して、JDBC 操作のために最適な低遅延と高スループットを実現することで、パフォーマンスを最大化します。

接続でサーバ・アドレス `localhost` または `127.0.0.1` が指定されている場合、既定で共有メモリが使用されます。実際のマシン・アドレスが指定されている場合は、TCP/IP が使用されます。共有メモリ・デバイスに障害が発生した場合、または共有メモリ・デバイスが利用できない場合、接続は自動的に TCP/IP にフォールバックします。

共有メモリを無効にするには、接続文字列で SharedMemory プロパティを `false` に設定します。以下の例では、サーバ・アドレスが `127.0.0.1` に指定されている場合でも共有メモリを使用しない接続を作成します。

```
  Properties props = new Properties();
  props.setProperty("SharedMemory", "false");
  props.setProperty("user", "_system");
  props.setProperty("password", "SYS");
  IRISConnection conn = (IRISConnection)DriverManager.getConnection("jdbc:IRIS://127.0.0.1:1972/USER/
",props);
```

アクセサ DataSource.getSharedMemory() および DataSource.setSharedMemory() を使用して、現在の接続モードの読み取りおよび設定を行うことができます。IRISConnection.isUsingSharedMemory() メソッドを使用して、接続モードをテストすることもできます。

共有メモリは TLS または Kerberos 接続には使用されません。共有メモリ接続が試行されたかどうか、およびその接続が成功したかどうかの情報は、JDBC ログに記録されます (“JDBC のログ” を参照してください)。

注釈 **共有メモリ接続はコンテナの境界を越えて機能しない**

現在のところ、2 つの異なるコンテナ間の共有メモリ接続はサポートされていません。クライアントが `localhost` または `127.0.0.1` を使用してコンテナの境界を越えて接続を試行した場合、接続モードは既定で共有メモリになり、接続は失敗します。このことは、Docker の `--network host` オプションが指定されているかどうかに関係なく適用されます。サーバ・アドレスの実際のホスト名を指定するか、接続文字列で共有メモリを無効にすることで (上記の例のように)、コンテナ間の TCP/IP 接続を保証できます。

サーバとクライアントが同じコンテナにある場合は、問題なく共有メモリ接続を使用できます。

3.4.3 文のプーリング

JDBC 4.0 は、追加のインフラストラクチャである「文のプーリング」を追加しています。この機能により、最適化された文は、最初に使用されたときにキャッシュに格納されます。文のプールは、接続プールによって維持され、プールされた文を接続間で共有できます。実装の詳細は、ユーザに対して完全に透過的です。必要な機能を提供するかどうかはドライバで決まります。

InterSystems JDBC は、この概念が JDBC 仕様の一部になるかなり前から文のプーリングを実装していました。一方、InterSystems IRIS ドライバは、この仕様で推奨されているテクニックに類似したテクニックを使用しており、実際にプーリングを実装すると高度に最適化されます。ほとんどの実装と異なり、InterSystems JDBC は、3 つの別々の文プーリング・キャッシュを備えています。1 つは JDBC 仕様で定義されている文プーリングにほぼ相当しますが、その他の 2 つは

InterSystems IRIS 特有の最適化です。必要に応じて、InterSystems JDBC 文プーリングは、ユーザに対して完全に透過的になります。

InterSystems JDBC の実装は、Statement メソッドである setPoolable() および isPoolable() を、当該の文をプールする必要があるかどうかのヒントとしてサポートします。InterSystems IRIS は、独自のヒューリスティックを使用して、3 つの文プールすべての適切なサイズを決定します。したがって、IRISConnectionPoolDataSource の maxStatements プロパティを設定することによる文プール・サイズの制約をサポートしません。オプションの javax.sql.StatementEventListener インタフェースは同じ理由でサポートされません (また、重要ではありません)。

4

構成と要件

InterSystems JDBC ドライバを使用するには、Java プログラミング言語に精通し、使用するオペレーティング・システムで Java を構成する方法をある程度理解していることが必要です。UNIX® で InterSystems JDBC ドライバにカスタム構成を設定する場合は、コードのコンパイルとリンク、シェル・スクリプトの記述、およびその他の関連タスクにも精通している必要があります。

4.1 InterSystems Java クラス・パッケージ

主要な InterSystems Java クラス・パッケージは、以下のファイルに含まれています (<version> は、3 つの部分で構成されるパッケージ・バージョン番号です (3.3.0 など))。

- intersystems-jdbc-<version>.jar — 主要 JDBC jar ファイル。このリスト内のその他のファイルはすべてこのファイルに依存しています。

  このファイルには、主要 JDBC API のほかに、Native SDK を実装するクラスも含まれています ("Native SDK for Java の使用法" を参照)。

- intersystems-xep-<version>.jar — XEP Java 永続アプリケーションに必要です ("InterSystems XEP による Java オブジェクトの永続化" を参照)。JDBC jar に依存しています。

- intersystems-uima-<version>.jar — UIMA サポートに必要です ("InterSystems UIMA の使用法" を参照)。JDBC jar に依存しています。

Java のサポートされている各バージョンに対してこれらのファイルの個別のバージョンがあります。個別のバージョンは、<install-dir>/dev/java/lib のサブディレクトリにあります (例えば、<install-dir>/dev/java/lib/JDK18 には Java 1.8 用のファイルが含まれています)。

InterSystems IRIS のインスタンスの <install-dir> (InterSystems IRIS ルート・ディレクトリ) の場所は、そのインスタンスで InterSystems ターミナルを開き、以下の ObjectScript コマンドを発行することによって確認できます。

```
write $system.Util.InstallDirectory()
```

最新バージョンの JDBC および XEP クラス・パッケージは、"InterSystems IRIS Driver Packages" ページからもダウンロードできます。

4.2 クライアントとサーバ間の構成

Java クライアントと InterSystems IRIS サーバは同じ物理マシンに配置することも、別々のマシンに配置することもできます。InterSystems IRIS のコピーが必要なのは、InterSystems IRIS サーバ・マシンのみです。クライアント・アプリケーションにはローカル・コピーは不要です。

4.2.1 Java クライアント要件

InterSystems IRIS Java クライアントでは、サポート対象バージョンの Java JDK が必要になります。クライアント・アプリケーションでは、InterSystems IRIS のローカル・コピーは必要ありません。

このリリース用のドキュメント "インターシステムズのサポート対象プラットフォーム" には、すべての Java ベース・クライアント・アプリケーションの現在の要件が明記されています。サポートされている Java リリースは、"サポート対象 Java テクノロジ" に関するセクションを参照してください。

Java バインディングのコア・コンポーネントは、intersystems-jdbc-3.2.0.jar という名前のファイルです。このファイルには、InterSystems IRIS サーバと通信するための接続とキャッシングのメカニズム、および JDBC 接続性を提供する Java クラスが含まれています。クライアント・アプリケーションでは InterSystems IRIS のローカル・コピーは必要ありませんが、Java プロキシ・クラスのコンパイル時または使用時には、intersystems-jdbc-3.2.0.jar ファイルがアプリケーションのクラス・パス上に存在している必要があります。これらのファイルの詳細は、"InterSystems IRIS Java クラス・パッケージ" を参照してください。

4.2.2 InterSystems IRIS サーバ構成

Java クライアントが InterSystems IRIS サーバに接続するには、ユーザ名およびパスワードに加えて、サーバ IP アドレス、TCP ポート番号、および InterSystems IRIS ネームスペースを指定した URL が必要です。

Java または JDBC のクライアント・アプリケーションを実行するには、インストール環境が以下の要件を満たしていることを確認してください。

- クライアントは、互換性のあるバージョンの InterSystems IRIS サーバを現在実行しているマシンにアクセスできる必要があります (このリリース用のドキュメント "インターシステムズのサポート対象プラットフォーム" を参照してください)。クライアントとサーバは同じマシン上で動作できます。
- クラス・パスに、クライアント・バージョンの Java JDK に対応する intersystems-jdbc-3.n.n.jar のバージョンが含まれている必要があります ("InterSystems IRIS Java クラス・パッケージ" を参照してください)。
- InterSystems IRIS サーバに接続するために、クライアント・アプリケーションに以下の情報が必要です。
  - InterSystems IRIS スーパーサーバが動作しているマシンの IP アドレス。Java サンプル・プログラムは、ローカル・マシン上でサーバのアドレスを使用します (`localhost` または `127.0.0.1`)。サンプル・プログラムから別のシステムに接続する場合は、接続文字列を変更して、リコンパイルする必要があります。
  - InterSystems IRIS スーパーサーバが待ち受け状態にある TCP ポート番号。Java サンプル・プログラムは、既定のポート ("構成パラメータ・ファイル・リファレンス" の "DefaultPort" を参照) を使用します。サンプル・プログラムに別のポートを使用させる場合は、接続文字列を変更して、リコンパイルする必要があります。
  - 有効な SQL ユーザ名とパスワード。SQL ユーザ名とパスワードは、管理ポータルの System Administration > Security > Users ページで管理できます。Java サンプル・プログラムは、管理者のユーザ名 `"_SYSTEM"` と既定のパスワード `"SYS"` または `"sys"` を使用します。通常、サーバのインストール後に既定のパスワードを変更します。サンプル・プログラムで異なるユーザ名とパスワードを使用する場合は、ユーザ名とパスワードを変更した後でリコンパイルする必要があります。
  - クライアント・アプリケーションが使用するクラスとデータを含むサーバ・ネームスペース。

InterSystems IRIS サーバへの接続の詳細は、“JDBC 接続の確立”を参照してください。

4.2.3 Transact-SQL 言語の有効化

既定では、JDBC は InterSystems IRIS SQL 言語を使用します。以下の方法で、Transact SQL (TSQL) 言語をサポートするように言語を変更できます。

```
connection.setSQLDialect(int);
```

または

```
statement.setSQLDialect(int);
```

利用可能な int オプションは、0 = InterSystems IRIS SQL (既定)、1 = MSSQL、2 = Sybase です。

ドライバのプロパティで言語を定義することもできます。

言語が 0 より大きい場合、JDBC を介して準備または実行される SQL 文は、サーバ上で若干異なる方法で処理されます。文は指定した言語を使用して処理されてから、InterSystems IRIS SQL または ObjectScript 文に変換されます。

5
JDBC の基本

JDBC では経験豊富な Java データベース開発者向けに概要を説明する必要はありませんが、使用頻度の低い小型のユーティリティ・アプリケーションでのみ Java を使用する場合であっても、この説明は非常に役立つ可能性があります。このセクションでは、データベースにクエリを実行して結果を操作するために頻繁に使用する JDBC のクラスとメソッドの例を提供します。

- "簡単な JDBC アプリケーション" は、JDBC の基本機能を示す完全だが非常に簡単なアプリケーションです。
- 以下のセクションでは、PreparedStatement と CallableStatement を使用してデータベースにクエリを実行して ResultSet を返す方法を示します。
  - "作成済み文の実行" ― 暗黙結合構文を使用する例です。
  - "CallableStatement を使用したストアド・プロシージャの実行" ― ストアド・プロシージャを実行する例です。
  - "複数の結果セットを返す" ― InterSystems IRIS ストアド・プロシージャによって返される複数の結果セットへのアクセス方法です。
- 以下のセクションでは、JDBC 結果セットを使用して、データベースにデータを挿入して更新する方法を紹介します。
  - "データの挿入および生成されるキーの取得" ― PreparedStatement および SQL INSERT コマンドを使用します。
  - "結果セットのスクロール" ― 結果セットの任意の行にランダムにアクセスします。
  - "トランザクションの使用法" ― JDBC トランザクション・モデルを使用して変更をコミットまたはロールバックします。

注釈　ほとんどの場合、このセクションの例は、アプリケーション全体ではなくコード断片として表されています。これらの例は、基本機能をできる限り簡単かつ明確に示しており、すぐれたコーディング方法の例示を目的としているわけではありません。この例では、dbconnection という接続オブジェクトが既に開かれていて、コード断片すべてが適切な `try/catch` 文の内側にあることを前提とします。

5.1 簡単な JDBC アプリケーション

このセクションでは、最も一般的な JDBC クラスの一部の使用法を例示する非常に簡単な JDBC アプリケーションについて説明します。

- IRISDataSource オブジェクトは、JDBC アプリケーションを InterSystems IRIS データベースにリンクする Connection オブジェクトの作成に使用されます。

- Connection オブジェクトは、ダイナミック SQL クエリを実行可能な PreparedStatement オブジェクトの作成に使用されます。
- PreparedStatement クエリは、要求された行を含む ResultSet オブジェクトを返します。
- ResultSet オブジェクトには、特定の行に移動し、その行の特定の列の読み取りまたは更新を行うために使用することができるメソッドがあります。

これらのクラスについては、すべて以降のセクションで詳しく説明します。

SimpleJDBC アプリケーション

最初に、JDBC パッケージをインポートし、try ブロックを開きます。

```
import java.sql.*;
import javax.sql.*;
import com.intersystems.jdbc.*;

public class SimpleJDBC{
  public static void main() {
    try {

// Use IRISDataSource to open a connection
      Class.forName ("com.intersystems.jdbc.IRISDriver").newInstance();
      IRISDataSource ds = new IRISDataSource();
      ds.setURL("jdbc:IRIS://127.0.0.1:1972/User");
      Connection dbconn = ds.getConnection("_SYSTEM","SYS");

// Execute a query and get a scrollable, updatable result set.
      String sql="Select Name from Demo.Person Order By Name";
      int scroll=ResultSet.TYPE_SCROLL_SENSITIVE;
      int update=ResultSet.CONCUR_UPDATABLE;
      PreparedStatement pstmt = dbconn.prepareStatement(sql,scroll,update);
      java.sql.ResultSet rs = pstmt.executeQuery();

// Move to the first row of the result set and change the name.
      rs.first();
      System.out.println("\n Old name = " + rs.getString("Name"));
      rs.updateString("Name", "Bill. Buffalo");
      rs.updateRow();
      System.out.println("\n New name = " + rs.getString("Name") + "\n");

// Close objects and catch any exceptions.
      pstmt.close();
      rs.close();
      dbconn.close();
    } catch (Exception ex) {
      System.out.println("SimpleJDBC caught exception: "
             + ex.getClass().getName() + ": " + ex.getMessage());
    }
  } // end main()
} // end class SimpleJDBC
```

5.2 作成済み文を使用するクエリ

以下のクエリは、作成済み文を使用して、"M" ～ "Z" で始まる名前の会社を担当する “A” ～ "E" で始まる名前の従業員すべてのリストを返します。

```
Select ID, Name, Company->Name from Demo.Employee
Where Name < ? and Company->Name > ?
Order By Company->Name
```

注釈　この文は、暗黙結合構文 (-> 演算子) を使用しています。この構文は、Demo.Employee が参照する Company クラスにアクセスする簡単な方法を提供します。

作成済み文を実装するには、以下の手順に従います。

- クエリを含む文字列を作成し、これを使用して PreparedStatement オブジェクトを初期化した後、クエリ・パラメータ値を設定し、クエリを実行します。

```
String sql=
   "Select ID, Name, Company->Name from Demo.Employee " +
   "Where Name < ? and Company->Name > ? " +
   "Order By Company->Name";
PreparedStatement pstmt = dbconnection.prepareStatement(sql);

pstmt.setString(1,"F");
pstmt.setString(2,"L");
java.sql.ResultSet rs = pstmt.executeQuery();
```

- 結果セットを取得して表示します。

```
java.sql.ResultSet rs = pstmt.executeQuery();
ResultSetMetaData rsmd = rs.getMetaData();
int colnum = rsmd.getColumnCount();
while (rs.next()) {
   for (int i=1; i<=colnum; i++) {
      System.out.print(rs.getString(i) + "  ");
   }
   System.out.println();
}
```

5.3 CallableStatement によるストアド・プロシージャを使用するクエリ

以下のコードは、ByName (Demo.Person に格納されている InterSystems IRIS ストアド・プロシージャ) を実行します。

- java.sql.CallableStatement オブジェクトを作成し、これをストアド・プロシージャの名前で初期化します。プロシージャの SqlName は、SP_Demo_By_Name です。これは、Java クライアント・コード内でこの名前がどのように参照されるかを示します。

```
String sql="call Demo.SP_Demo_By_Name(?)"
CallableStatement cs = dbconnection.prepareCall(sql);
```

- クエリ・パラメータ値を設定し、このクエリを実行した後、結果セットを繰り返し処理してデータを表示します。

```
cs.setString(1,"A");
java.sql.ResultSet rs = cs.executeQuery();

ResultSetMetaData rsmd = rs.getMetaData();
int colnum = rsmd.getColumnCount();
while (rs.next()) {
   for (int i=1; i<=colnum; i++)
      System.out.print(rs.getString(i) + "  ");
}
System.out.println();
```

5.4 複数の結果セットを返すクエリ

InterSystems IRIS では、複数の結果セットを返すストアド・プロシージャを定義できます。InterSystems JDBC ドライバは、このようなストアド・プロシージャの実行をサポートしています。次に、2 つの結果セットを返す InterSystems IRIS ストアド・プロシージャの例を示します (2 つのクエリ結果の列構造は異なっています)。

```
/// This class method produces two result sets.
ClassMethod DRS(st) [ ReturnResultsets, SqlProc ]
{
 $$$ResultSet("select Name from Demo.Person where Name %STARTSWITH :st")
 $$$ResultSet("select Name, DOB from Demo.Person where Name %STARTSWITH :st")
 Quit
}
```

$$$ResultSet は、SQL 文 (文字リテラルとして指定) を作成して実行し結果セットを返す事前定義 InterSystems マクロです。

以下のコードは、このストアド・プロシージャを実行し、返された結果セット両方の繰り返し処理を行います。

- java.sql.CallableStatement オブジェクトを作成し、これをストアド・プロシージャの名前を使用して初期化します。クエリ・パラメータを設定し、execute を使用して、クエリを実行します。

```
CallableStatement cs = dbconnection.prepareCall("call Demo.Person_DRS(?)");
cs.setString(1,"A");
boolean success=cs.execute();
```

- データを表示する結果セットのペアに繰り返し処理を行います。getResultSet が現在の結果セットを取得したら、getMoreResults は、その結果セットを閉じて、CallableStatement オブジェクトの次の結果セットへ移動します。

```
if(success) do{
   java.sql.ResultSet rs = cs.getResultSet();
   ResultSetMetaData rsmd = rs.getMetaData();
   for (int j=1; j<rsmd.getColumnCount() + 1; j++)
      System.out.print(rsmd.getColumnName(j)+ "\t\t");
   System.out.println();
   int colnum = rsmd.getColumnCount();
   while (rs.next()) {
      for (int i=1; i<=colnum; i++)
         System.out.print(rs.getString(i) + " \t ");
      System.out.println();
   }
   System.out.println();
 } while (cs.getMoreResults());
```

5.5 データの挿入および生成されるキーの取得

以下のコードは、新しい行を Demo.Person に挿入し、生成される ID キーを取得します。

- PreparedStatement オブジェクトを作成し、これを SQL 文字列を使用して初期化し、生成されるキーが返されるように指定します。

```
String sqlIn="INSERT INTO Demo.Person (Name,SSN,DOB) " + "VALUES(?,?,?)";
int keys=Statement.RETURN_GENERATED_KEYS;
PreparedStatement pstmt = dbconnection.prepareStatement(sqlIn, keys);
```

- クエリ・パラメータの値を設定し、更新を実行します。

```
String SSN = Demo.util.generateSSN(); // generate a random SSN
java.sql.Date DOB = java.sql.Date.valueOf("1984-02-01");

pstmt.setString(1,"Smith,John"); // Name
pstmt.setString(2,SSN); // Social Security Number
pstmt.setDate(3,DOB); // Date of Birth
pstmt.executeUpdate();
```

- 新しい行を挿入するたびに、システムは、その行のオブジェクト ID を自動的に生成します。生成される ID キーは、結果セットに取得され、SSN と共に表示されます。

```
java.sql.ResultSet rsKeys = pstmt.getGeneratedKeys();
rsKeys.next();
String newID=rsKeys.getString(1);
System.out.println("new ID for SSN " + SSN + " is " + newID);
```

このコードは ID が rsKeys で生成される最初で唯一のキーであることを想定していますが、実際にはこの想定が常に安全であるとは限りません。

- ID で新しい行を取得し、これを表示します (Age は DOB に基づいて計算される値です)。

```
String sqlOut="SELECT IName,Age,SSN FROM Demo.Person WHERE ID="+newID;
pstmt = dbconnection.prepareStatement(sqlOut);
java.sql.ResultSet rsPerson = pstmt.executeQuery();

int colnum = rsPerson.getMetaData().getColumnCount();
rsPerson.next();
for (int i=1; i<=colnum; i++)
   System.out.print(rsPerson.getString(i) + "  ");
System.out.println();
```

5.6 結果セットのスクロール

InterSystems JDBC ドライバは、スクロール可能な結果セットをサポートします。これにより、Java アプリケーションでこの結果セット・データを順方向および逆方向に移動できます。prepareStatement() メソッドは、以下のパラメータを使用して、結果セットがどのように機能するかを決定します。

- resultSetType パラメータは、変更の表示方法を決定します。
  - ResultSet.TYPE_SCROLL_SENSITIVE は、別のプロセスで基本データに対して行われた変更を表示するスクロール可能な結果セットを作成します。
  - ResultSet.TYPE_SCROLL_INSENSITIVE は、現在のプロセスで行われた変更のみを表示するスクロール可能な結果セットを作成します。
- resultSetConcurrency パラメータは、結果セットを更新する場合、ResultSet.CONCUR_UPDATABLE に設定する必要があります。

以下のコードは、スクロール可能な結果セットを作成して使用します。

- PreparedStatement オブジェクトを作成し、クエリ・パラメータを設定して、このクエリを実行します。

```
String sql="Select Name, SSN from Demo.Person "+
   " Where Name > ? Order By Name";
int scroll=ResultSet.TYPE_SCROLL_SENSITIVE;
int update=ResultSet.CONCUR_UPDATABLE;

PreparedStatement pstmt = dbconnection.prepareStatement(sql,scroll,update);
pstmt.setString(1,"S");
java.sql.ResultSet rs = pstmt.executeQuery();
```

新しい行が挿入される結果セットには、InterSystems IRIS ID 列を含めないでください。ID 値は、InterSystems IRIS によって自動的に定義されます。

- このアプリケーションでは、この結果セットを順方向および逆方向にスクロールできます。結果セットのカーソルを最後の行の後ろに移動するには、afterLast を使用します。逆方向にスクロールするには、previous を使用します。

```
rs.afterLast();
int colnum = rs.getMetaData().getColumnCount();
while (rs.previous()) {
   for (int i=1; i<=colnum; i++)
      System.out.print(rs.getString(i) + "  ");
   System.out.println();
}
```

- absolute を使用して、特定の行に移動します。このコードでは 3 番目の行の内容が表示されます。

```
rs.absolute(3);
for (int i=1; i<=colnum; i++)
   System.out.print(rs.getString(i) + "  ");
System.out.println();
```

- relative を使用して、現在の行を基準にした特定の行に移動します。以下のコードでは、最初の行に移動して 2 行下方にスクロールした後、3 行目を再度表示します。

```
rs.first();
rs.relative(2);
for (int i=1; i<=colnum; i++)
   System.out.print(rs.getString(i) + "  ");
System.out.println();
```

- 行を更新するには、カーソルを更新する行に移動し、目的の列を更新してから updateRow を呼び出します。

```
rs.last();
rs.updateString("Name", "Avery. Tara R");
rs.updateRow();
```

- 行を挿入するには、カーソルを“挿入行”に移動してからその行の列を更新します。NULL 値が許容されない列がすべて更新されていることを確認します。最後に、insertRow を呼び出します。

```
rs.moveToInsertRow();
rs.updateString(1, "Abelson,Alan");
rs.updateString(2, Demo.util.generateSSN()));
rs.insertRow();
```

5.7 トランザクションの使用法

InterSystems JDBC ドライバは、標準の JDBC トランザクション・モデルをサポートしています。

- SQL 文をトランザクションにグループ化するには、最初に次のように setAutoCommit() を使用して、自動コミット・モードを無効化する必要があります。

```
dbconnection.setAutoCommit(false);
```

- commit() の最後の実行またはロールバック以降に実行されたすべての SQL 文をデータベースに対してコミットするには、commit() を使用します。

```
pstmt1.execute();
pstmt2.execute();
pstmt3.execute();
dbconnection.commit();
```

- トランザクション内のすべてのトランザクションをロールバックするには、rollback() を使用します。ここで、SQLException がトランザクションで任意の SQL 文によってスローされる場合、rollback() が呼び出されます。

```
catch(SQLException ex) {
   if (dbconnection != null) {
      try {
         dbconnection.rollback();
      } catch (SQLException excep){
         // (handle exception)
      }
   }
}
```

次に、この例で使用する java.sql.Connection メソッドの概要を説明します。

- setAutoCommit()

  既定では、Connection オブジェクトは自動コミット・モードになっています。このモードでは、SQL 文は実行されるとすぐにコミットされます。複数の SQL 文を 1 つのトランザクションにグループ化するには、最初に `setAutoCommit(false)` を使用して Connection オブジェクトの自動コミット・モードを終了します。`setAutoCommit(true)` を使用して、Connection オブジェクトを自動コミット・モードに再設定します。

- commit()

  commit() を実行すると、commit() または rollback() の最後の実行以降に実行されたすべての SQL 文がコミットされます。最初に自動コミットを `false` に設定しないで commit() を呼び出すと、例外が返されないことに注意してください。

- rollback()

  rollback を実行すると、トランザクションが中止され、トランザクションによって変更されたすべての値が元の状態にリストアされます。

注釈　**Native SDK for Java トランザクション・モデル**

Native SDK for Java は、ここで説明した java.sql トランザクション・モデルに代わるトランザクション・モデルを提供します。Native SDK トランザクション・モデルは ObjectScript トランザクション・メソッドに基づいており、JDBC モデルと互換性はありません。トランザクションに Native SDK メソッド呼び出しが含まれる場合は、Native SDK モデルを使用する必要があります。詳細は、"Native SDK for Java の使用法" を参照してください。

6

JDBC クイック・リファレンス

この章は、以下の拡張クラスとオプションに関するクイック・リファレンスです。

- ConnectionPoolDataSource クラス ー InterSystems 接続プーリングに関連するメソッド。
- IRISDataSource クラス ー InterSystems 固有の接続プロパティ。
- 接続パラメータのオプション ー `properties` ファイルで使用できる接続パラメータのリストを提供します。

注釈　このリファレンスでは、このドキュメントで別途説明する拡張メソッドとバリアントのみをリストします。オプションの JDBC 機能の拡張、バリアント、実装を含む、InterSystems JDBC ドライバのすべての機能の完全な説明は、"Java サードパーティ API 用実装リファレンス" の "JDBC ドライバのサポート" を参照してください。

6.1 ConnectionPoolDataSource クラス

com.intersystems.jdbc.ConnectionPoolDataSource クラスは、javax.sql.ConnectionPoolDataSource インタフェースを完全に実装します。また、InterSystems IRIS 接続プーリングを制御するための、次の拡張メソッドのセットも含まれます。詳細は、"IRISConnectionPoolDataSource メソッドの使用法" を参照してください。

getConnectionLifetime()

ConnectionPoolDataSource.getConnectionLifetime() は、接続の存続時間を秒単位で返します ("setConnectionLifetime()" も参照してください)。

```
int getConnectionLifetime()
```

getConnectionWaitTimeout()

ConnectionPoolDataSource.getConnectionWaitTimeout() は、接続プール・マネージャが、接続が利用可能になるまで待機する秒数を返します ("setConnectionWaitTimeout()" も参照してください)。

```
int getConnectionWaitTimeout()
```

getMaxPoolSize()

ConnectionPoolDataSource.getMaxPoolSize() は、接続プールの現在の最大サイズを表す int を返します ("setMaxPoolSize()" も参照してください)。

```
int getMaxPoolSize()
```

getMinPoolSize()

ConnectionPoolDataSource.getMinPoolSize() は、接続プールの現在の最小サイズを表す int を返します (″setMinPoolSize()″ も参照してください)。

```
int getMinPoolSize()
```

getPoolCount()

ConnectionPoolDataSource.getPoolCount() は、接続プール内の現在のエントリ数を表す int を返します。SQLException をスローします。

```
int getPoolCount()
```

getPooledConnection()

このメソッドの代わりに常に IRISDataSource.getConnection() を使用してください。

このメソッドは呼び出さないでください。

ConnectionPoolDataSource.getPooledConnection() はインタフェースに必要ですが、直接呼び出すことはできません。InterSystems JDBC ドライバは接続プーリングを透過的に制御します。

InterSystems JDBC ドライバ接続は、常に IRISDataSource の getConnection() メソッドを呼び出して取得する必要があります。このメソッドは、自動の透過的な接続プーリングを提供するために拡張されています。ConnectionPoolDataSource.getPooledConnection() メソッドは、JDBC 標準で要求されているため実装されていますが、直接呼び出してはなりません。

getValidateOnConnect()

ConnectionPoolDataSource.getValidateOnConnect() は、データソースの `PingOnConnect` 設定を返します (″setValidateOnConnect()″ も参照してください)。

```
boolean getValidateOnConnect()
```

restartConnectionPool()

ConnectionPoolDataSource.restartConnectionPool() は、接続プールを再起動します。物理接続をすべて切断し、接続プールを空にします。SQLException をスローします。

```
void restartConnectionPool()
```

setMaxPoolSize()

ConnectionPoolDataSource.setMaxPoolSize() は、接続プールの最大サイズを設定します。最大サイズが設定されていない場合、既定値として `40` に設定されます (″getMaxPoolSize()″ も参照してください)。

```
void setMaxPoolSize(int max)
```

- `max` — 接続プールのオプションの最大サイズです (既定値は `40`)。

setMinPoolSize()

ConnectionPoolDataSource.setMinPoolSize() は、接続プールの最小サイズを設定します (″getMinPoolSize()″ も参照してください)。既定値は `0` です。

```
void setMinPoolSize(int min)
```

- `min` — 接続プールのオプションの最小サイズです (既定値は `0`)。

setConnectionWaitTimeout()

ConnectionPoolDataSource.setConnectionWaitTimeout() は、接続の待機タイムアウト間隔を指定した秒数に設定します (″getConnectionWaitTimeout()″ も参照してください)。既定値は 0 です。

```
void setConnectionWaitTimeout(int timeout)
```

- timeout — タイムアウト間隔を秒数で指定します (既定では 0)。

タイムアウト時間が過ぎても利用可能な接続がない場合は、例外がスローされます。既定では 0 に設定されています。これは、接続が即時に利用可能になることを示します。プールがいっぱいである場合は例外がスローされます。

setConnectionLifetime()

ConnectionPoolDataSource.setConnectionLifetime() は、接続の存続時間の値を秒単位で設定します (″getConnectionLifetime()″ も参照してください)。既定値は 0 (制限なし) です。

```
void setConnectionLifetime(int conLifeTime)
```

- conLifeTime — 秒単位での存続時間

setValidateOnConnect()

ConnectionPoolDataSource.setValidateOnConnect() は、データソースの現在の PingOnConnect を設定します (″getValidateOnConnect()″ も参照してください)。既定は true です。

```
boolean setValidateOnConnect(boolean p)
```

- p — 新しい PingOnConnect 設定

6.2 IRISDataSource クラス

com.intersystems.jdbc.IRISDataSource クラスは、javax.sql.DataSource インタフェースを完全に実装します。また、InterSystems IRIS の接続プロパティを取得または設定するための拡張メソッドも多数含まれます (詳細は、“接続パラメータのオプション” を参照してください)。

IRISDataSource は、InterSystems JDBC ドライバでサポートされていない javax.sql.CommonDataSource のメソッドを継承しません。

getConnection()

拡張機能を持つ必須メソッド

必須メソッド IRISDataSource.getConnection() は java.sql.Connection を返します。SQLException をスローします。

このメソッドは、InterSystems IRIS ドライバ接続を取得する際には常に使用する必要があります。また、InterSystems IRIS ドライバは、getConnection() が返す java.sql.Connection オブジェクトを使用して、プーリングを透過的に提供します。

```
java.sql.Connection getConnection()
java.sql.Connection getConnection(String usr,String pwd)
```

- usr — 対象の接続のオプションのユーザ名引数。

- pwd — 対象の接続のオプションのパスワード引数。

このメソッドはプーリングを提供し、常に getPooledConnection()、および PooledConnection クラスのメソッドの代わりに使用する必要があります (詳細は、“ConnectionPoolDataSource クラス” を参照してください)。

getConnectionSecurityLevel()

IRISDataSource.getConnectionSecurityLevel() は、現在の接続セキュリティ・レベルの設定を表す int を返します。"setConnectionSecurityLevel()" も参照してください。

```
int getConnectionSecurityLevel()
```

getDatabaseName()

IRISDataSource.getDatabaseName() は、現在のデータベース (InterSystems IRIS ネームスペース) の名前を表す String を返します。"setDatabaseName()" も参照してください。

```
String getDatabaseName()
```

getDataSourceName()

IRISDataSource.getDataSourceName() は、現在のデータ・ソース名を表す String を返します。"setDataSourceName()" も参照してください。

```
String getDataSourceName()
```

getDefaultTransactionIsolation()

IRISDataSource.getDefaultTransactionIsolation() は、現在の既定のトランザクション分離レベルを表す int を返します。"setDefaultTransactionIsolation()" も参照してください。

```
int getDefaultTransactionIsolation()
```

getDescription()

IRISDataSource.getDescription() は、現在の記述を表す String を返します。"setDescription()" も参照してください。

```
String getDescription()
```

getEventClass()

IRISDataSource.getEventClass() は、イベント・クラス・オブジェクトを表す String を返します。"setEventClass()" も参照してください。

```
String getEventClass()
```

getKeyRecoveryPassword()

IRISDataSource.getKeyRecoveryPassword() は、現在のキー・リカバリ・パスワード設定を表す String を返します。"setKeyRecoveryPassword()" も参照してください。

```
String getKeyRecoveryPassword()
```

getNodelay()

IRISDataSource.getNodelay() は、現在の TCP_NODELAY オプション設定を表す Boolean を返します。"setNodelay()" も参照してください。

```
boolean getNodelay()
```

getPassword()

IRISDataSource.getPassword() は、現在のパスワードを表す String を返します。"setPassword()" も参照してください。

```
String getPassword()
```

getPortNumber()

IRISDataSource.getPortNumber() は、現在のポート番号を表す int を返します。"setPortNumber()" も参照してください。

```
int getPortNumber()
```

getServerName()

IRISDataSource.getServerName() は、現在のサーバ名を表す String を返します。"setServerName()" も参照してください。

```
String getServerName()
```

getServicePrincipalName()

IRISDataSource.getServicePrincipalName() は、現在のサービス・プリンシパル名を表す String を返します。"setServicePrincipalName()" も参照してください。

```
String getServicePrincipalName()
```

getSharedMemory()

IRISDataSource.getSharedMemory() は、接続が共有メモリを使用しているかどうかを示す Boolean を返します。"setSharedMemory()" も参照してください。

```
Boolean getSharedMemory()
```

getSQLDialect()

IRISDataSource.getSQLDialect() は、現在の SQL 言語設定を表す int を返します ("setSQLDialect()" も参照してください)。

```
int getSQLDialect()
```

getSSLConfigurationName()

IRISDataSource.getSSLConfigurationName() は、現在の TLS 構成名の設定を表す String を返します。"setSSLConfigurationName()" も参照してください。

```
String getSSLConfigurationName()
```

getTransactionIsolationLevel()

IRISDataSource.getTransactionIsolationLevel() は、現在のトランザクション分離レベルを返します ("setTransactionIsolationLevel()" も参照してください)。

```
int getTransactionIsolationLevel()
```

getURL()

IRISDataSource.getURL() は、対象のデータソースの現在の URL を表す String を返します。"setURL()" も参照してください。

```
String getURL()
```

getUser()

IRISDataSource.getUser() は、現在のユーザ名を表す String を返します。"setUser()" も参照してください。

```
String getUser()
```

setConnectionSecurityLevel()

IRISDataSource.setConnectionSecurityLevel() は、このデータソースの接続セキュリティ・レベルを設定します。"getConnectionSecurityLevel()" も参照してください。

```
void setConnectionSecurityLevel(int level)
```

- `level` ー 接続セキュリティ・レベルの数値です。許容される値については、"接続パラメータのオプション" の [接続セキュリティレベル] のエントリを参照してください。

setDatabaseName()

IRISDataSource.setDatabaseName() は、対象のデータソースのデータベース名 (InterSystems IRIS ネームスペース) を設定します。"getDatabaseName()" も参照してください。

```
void setDatabaseName(String databaseName)
```

- `databaseName` ー InterSystems IRIS ネームスペースの文字列です。

setDataSourceName()

IRISDataSource.setDataSourceName() は、対象のデータソースのデータ・ソース名を設定します。`DataSourceName` はオプション設定で、接続には使用されません。"getDataSourceName()" も参照してください。

```
void setDataSourceName(String dataSourceName)
```

- `dataSourceName` ー データ・ソース名の文字列です。

setDefaultTransactionIsolation()

IRISDataSource.setDefaultTransactionIsolation() は、トランザクションの既定の分離レベルを設定します。"getDefaultTransactionIsolation()" も参照してください。

```
void setDefaultTransactionIsolation(int level)
```

- `level` ー 既定のトランザクション分離レベルの数値です。

setDescription()

IRISDataSource.setDescription() は、対象のデータソースの記述を設定します。`Description` はオプションの設定で、接続には使用されません。"getDescription()" も参照してください。

```
void setDescription(String desc)
```

- `desc` ー データソースの記述の文字列です。

setEventClass()

IRISDataSource.setEventClass() は、対象のデータソースのイベント・クラスを設定します。このイベント・クラスは InterSystems IRIS JDBC 固有のメカニズムです。この機能は完全にオプションで、ほとんどのアプリケーションで必要ありません。"getEventClass()" も参照してください。

```
void setEventClass(String eventClassName)
```

- `eventClassName` — トランザクション・イベント・クラスの名前です。

InterSystems JDBC サーバは、トランザクションのコミット時およびロールバック時にクラスで実装されているメソッドにディスパッチします。これらのメソッドが実装されているクラスは、“イベント・クラス” と呼ばれます。イベント・クラスがログイン時に指定されると、JDBC サーバは、現在のトランザクションをコミットする直前に %OnTranCommit にディスパッチし、現在のトランザクションをロールバック (中止) する直前に %OnTranRollback にディスパッチします。ユーザ・イベント・クラスは %ServerEvent を拡張する必要があります。このメソッドは、いかなる値も返さず、現在のトランザクションを中止できません。

setKeyRecoveryPassword()

IRISDataSource.setKeyRecoveryPassword() は、対象のデータソースのキー・リカバリ・パスワードを設定します。"getKeyRecoveryPassword()" も参照してください。

```
void setKeyRecoveryPassword(String password)
```

- `password` — データソースのキー・リカバリ・パスワードの文字列です。

setLogFile()

IRISDataSource.setLogFile() は、対象のデータソースのログ・ファイル名を無条件に設定します。

```
void setLogFile(String logFile)
```

- `logFile` — データソースのログ・ファイル名の文字列です。

setNodelay()

IRISDataSource.setNodelay() は、対象のデータソースの `TCP_NODELAY` オプションを設定します。このフラグを切り替えると、アプリケーションのパフォーマンスに影響する場合があります。設定されていない場合、既定値として `true` が設定されます。"getNodelay()" も参照してください。

```
void setNodelay(boolean noDelay)
```

- `noDelay` — データソースの `TCP_NODELAY` オプション設定です (既定値は `true`)。

setPassword()

IRISDataSource.setPassword() は、対象のデータソースのパスワードを設定します。"getPassword()" も参照してください。

```
void setPassword(String pwd)
```

- `pwd` — データソースのパスワードの文字列です。

setPortNumber()

IRISDataSource.setPortNumber() は、対象のデータソースのポート番号を設定します。"getPortNumber()" も参照してください。

```
void setPortNumber(int portNumber)
```

- `portNumber` – データソースのポート番号です。

setServerName()

IRISDataSource.setServerName() は、対象のデータソースのサーバ名を設定します。"getServerName()" も参照してください。

```
void setServerName(String serverName)
```

- `serverName` – データソースのサーバ名の文字列です。

setServicePrincipalName()

IRISDataSource.setServicePrincipalName() は、対象のデータソースのサービス・プリンシパル名を設定します。"getServicePrincipalName()" も参照してください。

```
void setServicePrincipalName(String name)
```

- `name` – データソースのサービス・プリンシパル名の文字列です。

setSharedMemory()

IRISDataSource.setSharedMemory() は、対象のデータソースの共有メモリ接続を設定します。"getSharedMemory()" も参照してください。

```
void setSharedMemory(Boolean sharedMemory)
```

- `sharedMemory` – オンは 0、オフは 1 です。

setSQLDialect()

IRISDataSource.setSQLDialect() は、現在の SQL 言語を設定します ("getSQLDialect()" も参照してください)。dialect が `0`、`1`、または `2` でない場合は SQLException をスローします。

```
void setSQLDialect(int dialect) throws SQLException
```

- `dialect` – 現在の SQL 言語を表す int。許容される値は、`0`、`1`、および `2` です。

setSSLConfigurationName()

IRISDataSource.setSSLConfigurationName() は、対象のデータソースの TLS 構成名を設定します。"getSSLConfigurationName()" も参照してください。

```
void setSSLConfigurationName(String name)
```

- `name` – TLS 構成名の文字列です。

setTransactionIsolationLevel()

IRISDataSource.setTransactionIsolationLevel() は、現在のトランザクション分離レベルを設定します（"getTransactionIsolationLevel()" も参照してください）。level が 1、2、または 32 でない場合は SQLException をスローします。

```
void setTransactionIsolationLevel(int level) throws SQLException
```

- level ― レベルを示す int。許容される値は、1、2、および 32 です。

setURL()

IRISDataSource.setURL() は、対象のデータソースの URL を設定します。"getURL()" も参照してください。

```
void setURL(String u)
```

- u ― URL の文字列です。

setUser()

IRISDataSource.setUser() は、対象のデータソースのユーザ名を設定します。"getUser()" も参照してください。

```
void setUser(String username)
```

- username ― ユーザ名の文字列です。

6.3 接続パラメータのオプション

このセクションでは、jdbc.IRISDataSource（インターシステムズでの javax.sql.DataSource の実装）で提供されている接続プロパティのリストと説明を示します。接続プロパティは、DriverManager（"接続のための DriverManager の使用法" を参照）に渡すことによって、または接続プロパティ・アクセサ（完全なリストは、"IRISDataSource クラス" を参照）を呼び出すことによって設定できます。

以下の接続プロパティがサポートされています。

connection security level

オプション。接続セキュリティ・レベルを表す整数です。有効なレベルは、0、1、2、3、または 10 です。既定値は 0 です。

0 - インスタンス認証（パスワード）

1 - Kerberos（認証のみ）

2 - Kerberos パケット整合性

3 - Kerberos 暗号化

10 - TLS

IRISDataSource のメソッド getConnectionSecurityLevel() および setConnectionSecurityLevel() を参照してください。

host

オプション。サーバの IP アドレスまたはホスト名を指定する文字列です。

接続パラメータ host ― IP アドレス、または完全修飾ドメイン名 (FQDN)。例えば、`127.0.0.1` と `localhost` は両方ともローカル・マシンを表します。

DataSource のメソッド getServerName() および setServerName() を参照してください。

key recovery password

オプション。現在のキー・リカバリ・パスワード設定を含む文字列です。既定値は `null` です。IRISDataSource のメソッド getKeyRecoveryPassword() および setKeyRecoveryPassword() を参照してください。

password

必須項目。パスワードが含まれる文字列です。既定値は `null` です。IRISDataSource のメソッド getPassword() および setPassword()

port

オプション。接続の TCP/IP ポート番号を指定する整数。

接続パラメータ port ― InterSystems IRIS スーパーサーバが待ち受け状態にある TCP ポート番号。既定値は `1972` です (複数の InterSystems IRIS インスタンスがインストールされている場合は、これ以降で最初に使用可能なポート番号です。"パラメータ・ファイル・リファレンス" の "DefaultPort" を参照)。

DataSource のメソッド getPortNumber() および setPortNumber() を参照してください。

service principal name

オプション。サービス・プリンシパル名を表す文字列です。既定値は `null` です。IRISDataSource のメソッド getServicePrincipalName() および setServicePrincipalName()

SharedMemory

オプション。`localhost` と `127.0.0.1` に常に共有メモリを使用するかどうかを示すブーリアン値です。既定値は `null` です。IRISDataSource のメソッド getSharedMemory() および setSharedMemory() を参照してください。“共有メモリ接続” も参照してください。

SO_RCVBUF

オプション。TCP/IP の `SO_RCVBUF` 値 (ReceiveBufferSize) を表す整数です。既定値は `0` です (システムの既定値を使用)。

SO_SNDBUF

オプション。TCP/IP の `SO_SNDBUF` 値 (SendBufferSize) を表す整数です。既定値は `0` です (システムの既定値を使用)。

TLS 構成名

オプション。このオブジェクトの現在の TLS 構成名を含む文字列です。既定値は `null` です。IRISDataSource のメソッド getSSLConfigurationName() および setSSLConfigurationName() を参照してください。

TCP_NODELAY

オプション。TCP/IP の `TCP_NODELAY` フラグ (Nodelay) を示すブーリアン値です。既定値は `true` です。

接続パラメータ nodelay ― IRISDataSource オブジェクト経由で接続している場合に `TCP_NODELAY` オプションを設定します。このフラグを切り替えると、アプリケーションのパフォーマンスに影響する場合があります。有効な値は、`true` および `false` です。設定されていない場合、既定値として `true` が設定されます。

IRISDataSource のメソッド getNodelay() および setNodelay()

TransactionIsolationLevel

オプション。トランザクションの分離レベルを表す java.sql.Connection 定数です。有効な値は、TRANSACTION_READ_UNCOMMITTED または TRANSACTION_READ_COMMITTED です。既定値は null です (システムの既定値 TRANSACTION_READ_UNCOMMITTED を使用)。

IRISDataSource のメソッド getTransactionIsolationLevel() および setTransactionIsolationLevel()

user

必須項目。ユーザ名が含まれる文字列です。既定値は null です。IRISDataSource のメソッド getUser() および setUser()

6.3.1 接続プロパティのリスト

以下のようなコードを使用して、任意の JDBC 準拠のドライバに使用可能なプロパティをリストできます。

```
java.sql.Driver drv = java.sql.DriverManager.getDriver(url);
java.sql.Connection dbconnection = drv.connect(url, user, password);
java.sql.DatabaseMetaData meta = dbconnection.getMetaData();
System.out.println ("\n\n\nDriver Info: ============================");
System.out.println (meta.getDriverName());
System.out.println ("release " + meta.getDriverVersion() + "\n");

java.util.Properties props = new Properties();
DriverPropertyInfo[] info = drv.getPropertyInfo(url,props);
for(int i = 0; i <info.length; i++) {
   System.out.println ("\n" + info[i].name);
   if (info[i].required) {System.out.print("   Required");}
      else {System.out.print ("   Optional");}
   System.out.println (", default = " + info[i].value);
   if (info[i].description != null)
      System.out.println ("   Description:" + info[i].description);
   if (info[i].choices != null) {
      System.out.println ("   Valid values: ");
      for(int j = 0; j < info[i].choices.length; j++)
         System.out.println("      " + info[i].choices[j]);
   }
}
```