jCaoSQL

ユーザーズ ガイド

version 1.01

April 24, 2015

【備考】

【改版履歴】

日付	版数	内容
2006/02/27	1.0	初版.
2015/04/24	1.01	RMI 連携機能の追加

目次

1. はじめに

jCaoSQL は java から CaoSQL のコントロールをする為のフレームワークで，CaoSQL のインタフェースを介して，ロボットコントローラなどのリソースにアクセスすることができます．現在のところ，一部制限はありますが，CaoSQL の機能は全て java から使用することが可能です．

本書の位置付けとしては，jCaoSQL を使用した java の CaoSQL クライアントアプリケーションの作成手引きですので，CaoSQL インタフェースの詳細は「CaoSQL 外部仕様書」などを参照して下さい．

2. システム構成

この章では, jCaoSQL のシステム構成について説明します.

2.1. システム要件

jCaoSQL は表 2-1 の環境で動作することが確認されています. Java 言語で構成されていますが, JNI を使用しネイティブコードを使用していますので, OS 間のポータビリティは保証されないことに留意して下さい.

表 2-1 jCaoSQL の動作環境

OS	Windows7
Java	JDK 1.7.x
CaoSQL	version 1.0

2.2. クラス構成

jCaoSQL では CaoSQL の COM オブジェクトモデルに対応し, 表 2-2 のようなクラスを持ちます.

CaoSQLHistory に関してはデータベースを参照する機能のみなので, 特に CaoSQL のインタフェースを使用する必要性が無く, jCaoSQL では対応していません. データベースを参照する必要があれば, 汎用的な O/R マッピングフレームワークなどを使用して下さい[1].

表 2-2 jCaoSQL と CaoSQL のオブジェクトモデル対応

CaoSQL オブジェクトモデル	jCaoSQL オブジェクトモデル
CaoSQLEngine	jp.co.denso.fa.jcaosql.CaoSQLEngine
CaoSQLController	jp.co.denso.fa.jcaosql.CaoSQLController
CaoSQLItem	jp.co.denso.fa.jcaosql.CaoSQLItem
CaoSQLHistory	-

またjCaoSQL のクラス図に関しては「付録 A.クラス図」を参照して下さい.

[1] Apache Torque, Apache Commons DbUtil, (Castor) JDO, Hibernate などが挙げられます.

3. 開発環境セットアップ

この章では jCaoSQL の開発環境のセットアップについて説明します. jCaoSQL のコードを直接参照したり, 修正したりすることがなければ, これらの設定は特に必要ありません.

3.1. 開発環境について

jCaoSQL は表 3-1 の環境で開発されました. ソースコードからのビルドの際にはこれらの環境を必要とします.

表 3-1 jCaoSQL の開発環境

OS	Windows7
Java	JDK 1.7.x
C++	Microsoft Visual C++2005
ビルドツール	Apache Ant 1.6.2
ドキュメント生成ツール	Doxygen 1.2.18 + Graphviz 1.10
テスティングフレームワーク	JUnit 3.8.1
CaoSQL	version 1.0

3.2. 開発環境セットアップ

それぞれ以下の手順で開発環境のセットアップを行います.

3.2.1. java 環境セットアップ

Sun の公式サイトから JDK 1.7.x をダウンロードし, 開発マシンにインストールを行います.

開発マシンの環境変数に **JAVA_HOME** として JDK のインストールフォルダを指定します.

さらに環境変数の **PATH** に**%JAVA_HOME%¥bin** を追加します.

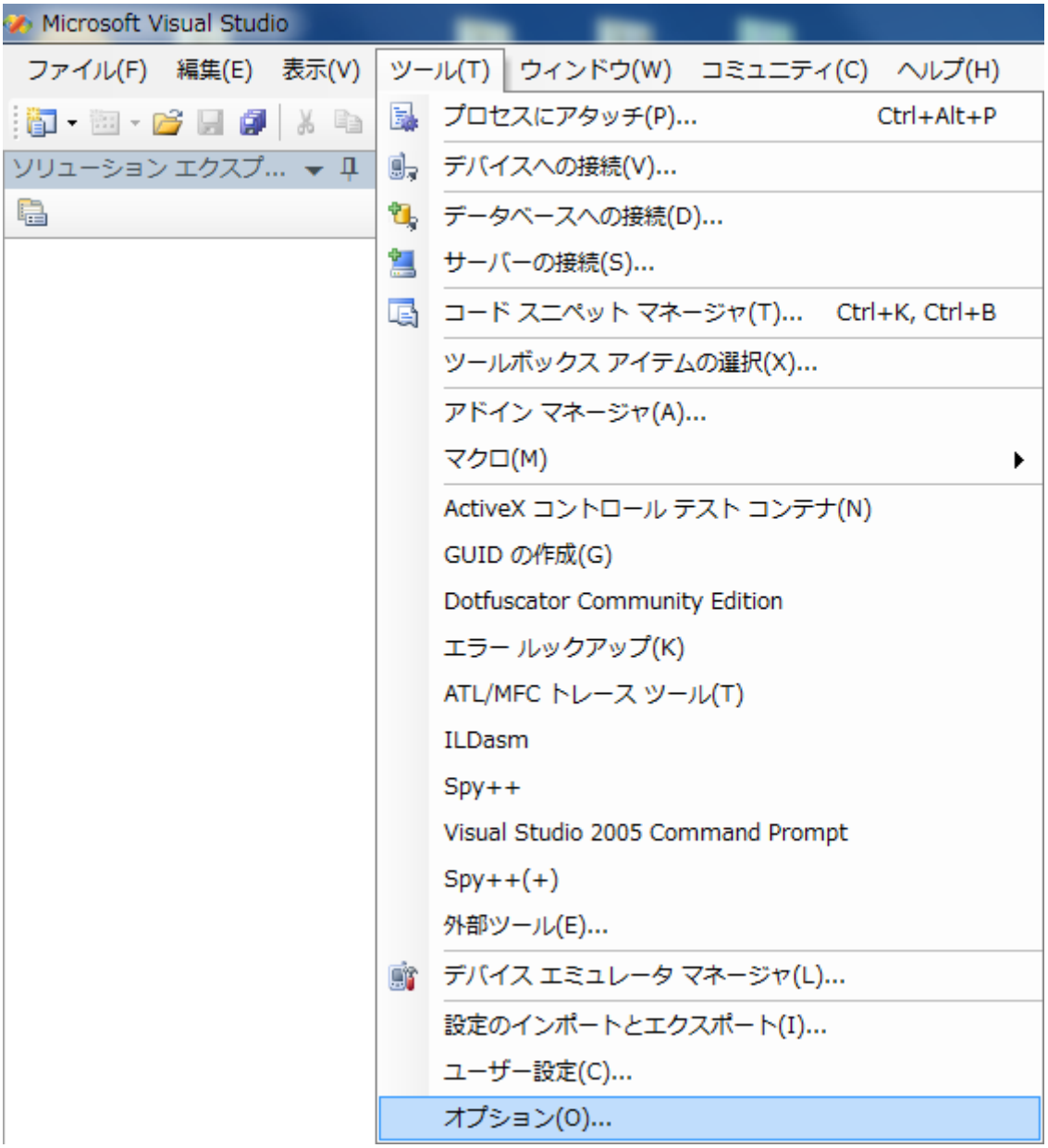
環境変数の設定方法が不明な場合はこちらを参照して下さい.

JUnit の公式サイトから JUnit3.8.1 をダウンロードします. ダウンロードしたファイルを解凍し, 開発マシンの環境変数に **CLASSPATH** として解凍したフォルダ内にある“junit.jar”を指定します.

3.2.2. VC++環境セットアップ

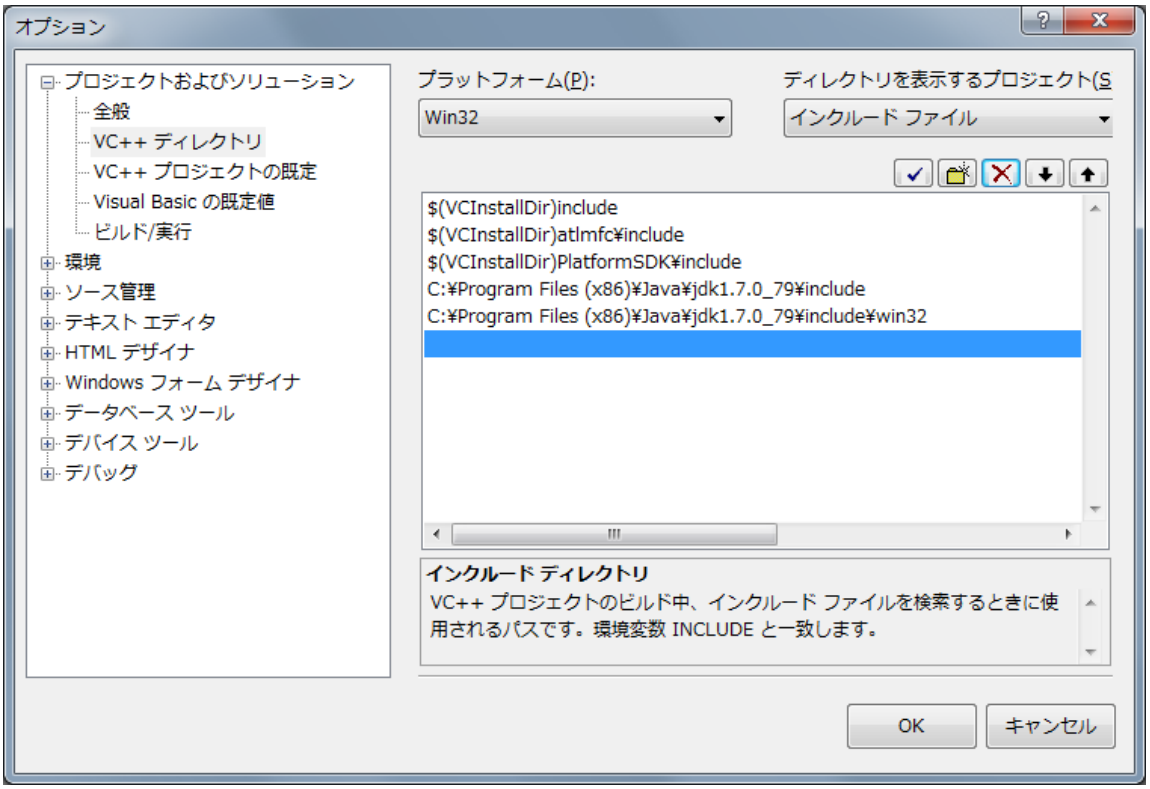
JNI のコンパイル環境を整えるため, 以下の設定を行う必要があります.

VC のメニューから, ツール→オプションを選択します.

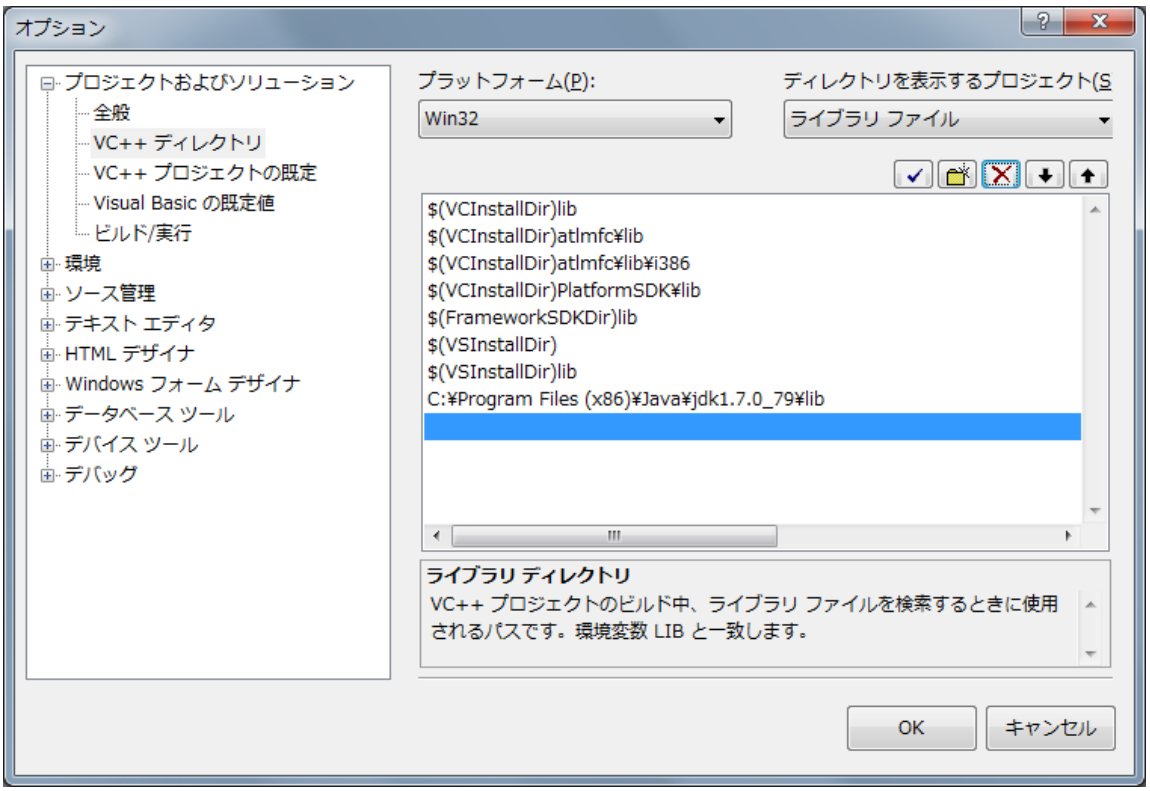


開いたウィンドウの *ディレクトリタブ*をクリックして，*プラットフォーム*に「Win32」，*表示するディレクトリ*に「インクルードファイル」として，*ディレクトリリスト*に%**JAVA_HOME**%¥include と%**JAVA_HOME**%¥include¥win32 を追加します．[2]

[2] %JAVA_HOME%は 3.2.1 で設定した環境変数変数に置き換えて下さい．



さらに，*表示するディレクトリ*を「ライブラリファイル」に変更し，*ディレクトリ*リストに**%JAVA_HOME%**¥libを追加します[3].



[3] 上に同じ.

3.3. ビルド方法

3.3.1. Ant のセットアップ

Apache Ant のサイトから Ant 1.6.2 をダウンロードし，アーカイブを適宜展開します．

開発マシンの環境変数に **ANT_HOME** として Ant を展開したフォルダを指定します．

さらに環境変数の **PATH** に%**ANT_HOME**%¥bin を追加します．

3.3.2. Ant の使用方法

3.3.2.1. 個別設定

jCaoSQL では個別の開発環境の設定ファイルとして build.properties ファイルが用意されていますので，これを環境に合わせて修正します．修正するのは以下 2 つの項目です．

caosql.src=[CAOSQL のソースディレクトリ]

msvc.path=[Visual C++へのパス]

ここでパスのデリミタは'¥'ではなくて'/'とすることに注意して下さい．

3.3.2.2. Ant タスクの説明

JCaoSQL プロジェクトでは以下の Ant タスクが用意されています．必要に応じてタスクを実行します．

表 3-2 jCaoSQL の Ant タスク一覧

タスク名	詳細
clean	ワークディレクトリの削除
init	ワークディレクトリの作成
jni	JNI ヘッダの生成
compile	compile-java 及び compile-native タスクの実行
compile-java	java ソースのコンパイル
compile-native	C++ネイティブコードのコンパイル
dll	jni 及び ompile-native タスクの実行
rebuild	clean 及び，compile タスクの実行
Test	全ての単体テストの実行と，レポートの作成
suitetest	指定したテストスィートの実行と，レポートの作成 （-Dtest.class=[test suite]でテストスィートの指定）
xdoc	javadoc 及び，doxygen タスクの実行
javadoc	Java コードの API ドキュメント作成
doxygen	nnative コードの API ドキュメント作成
jar	jar アーカイブを作成する．
dist	dll, jar, ドキュメントを作成し，dist フォルダに配置する

3.3.2.3. Ant タスクの実行

コマンドラインから

ant [タスク]

とすることで指定したタスクを実行します.

引数を必要とするタスクの場合はタスクの指定の後に引数を与えます.

ant [タスク] (引数)

例えば sutietest で CaoSQLTest というテストスィートのみ実行したい場合は,

ant suitetest –Dtest.class=CaoSQLTest

とします.

4. jCaoSQL フィーチャー

4.1. インスタンスの生成と破棄

jCaoSQL は表 2-2 に示した COM オブジェクトモデルに対応する java オブジェクトが，COM の IF を生成し，IF ポインタの保持や，参照カウント管理を行っています[4].

CaoSQLEngine のみがインスタンス生成可能なファクトリメソッド[5]を持ち，これ以外のオブジェクトは親オブジェクトから取得することで IF ポインタを確保します．これら以外の方法で，例えばクラスを new しただけでは COM の IF ポインタは確保されず，従って CaoSQL の操作を行うことは出来ません．

これらのオブジェクトが保持する IF ポインタは addRef メソッドと，release メソッドで参照カウントをコントロールすることが可能であり，逆に言えば IF ポインタを保持する java オブジェクトを破棄する場合は，参照カウントを 0 にする必要があります．参照カウンタの適切な操作が行われない場合は，終了時に CaoSQL の COM オブジェクトが破棄されず，メモリリークしてしまう可能性があります．

このインスタンス管理の詳細は 5.2.1 に示すサンプルを参考にして下さい．

4.2. VARIANT 型のマッピング

COM の持つ汎用的な型として VARIANT 型があり，CaoSQL でもタグや Item の値などに VARIANT 型が使用されています．これに対して java で純粋に VARAINT に対応する型が無いため，jCaoSQL では java.lang.Object 型にマッピングすることで対応しています．但し，VARIANT が持つ全ての型に対応してなく，以下の型に限定しています．

表 4-1 VARIANT マッピング

VAIANT 型	詳細
VT_BOOL	java.lang.Boolean
VT_I1	java.lang.Character
VT_UI1	java.lang.Byte
VT_I2(VT_UI2)	java.lang.Short
VT_I4(VT_UI4)	java.lang.Integer
VT_R4	java.lang.Float
VT_R8	java.lang.Double
VT_BSTR	java.lang.String
VT_DATE	java.util.Date
VT_ARRAY | VT_BOOL	java.lang.Boolean[]
VT_ARRAY | VT_I1	java.lang.Character[]
VT_ARRAY | VT_UI1	java.lang.Byte[]

[4] COM の参照カウント以外に java オブジェクトで参照カウントを持っていて，java オブジェクトから参照した回数以上はリリース不可能です．
[5] CaoSQLEngine#createInstance メソッド

VT_ARRAY | VT_I2(VT_UI2)	java.lang.Short[]
VT_ARRAY | VT_I4(VT_UI4)	java.lang.Integer[]
VT_ARRAY | VT_R4	java.lang.Float[]
VT_ARRAY | VT_R8	java.lang.Double[]
VT_ARRAY | VT_BSTR	java.lang.String[]
VT_ARRAY | VT_DATE	java.util.Date[]
VT_ARRAY | VT_VARIANT	java..lang.Object[]

ここで幾つかの制限事項があります.

- java には符号無しの short(VT_UI2)や long(VT_UI4)型が無い為, これらの符号無しの値を受け取るときは, 符号有りの値で格納されます. また, これらの符号無しの値を設定することは不可能です. (受け取りのみ)
- 配列型は 1 次元の配列のみ可能です. 2次元以上の値は型例外が発生します[6].
- VT_VARIANT の配列は, 型が混合した配列, 例えば Object[] array = {new Integer(1), new String("aa")})などに対応しています. これらの Object 内でまた他の Object を入れ子にすることはできません.
- Java では Currency(VT_CY)型はない為, 通貨型の表現には java.text.NumberFormat を利用します.

4.3. イベントシンク

jCaoSQL では CaoSQLControllerListener インタフェースを実装したクラスを作り, CaoSQLController の attachListener に渡すことで, CaoSQL でイベントが生起した際に Listener オブジェクトにコールバックすることが可能です. また, 同様に dettachListener をコールすることで, イベント受信を終了することができます. この処理はインスタンスの管理と同様で, attachListener でイベント受信をしています. CaoSQLController は, オブジェクト破棄時に dettachListener をコールする必要があります.

詳細な実装方法は 5.2.3 のサンプルを参照して下さい.

[6] jp.co.denso.fa.UnSupportedTypeException が発生します.

5. チュートリアル

5.1. クライアント開発環境セットアップ

ここではjCaoSQLを利用したクライアント開発の方法について触れます. クライアント開発ではjCaoSQLプロジェクト内の以下のファイルを使用します.

表 5-1 jCaoSQL の使用ファイル

ファイル名	概要
	設定方法
jCaoSQL.jar	jCaoSQL の java バイナリパッケージ
	ClassPath に通します.
jCaoSQL.dll	jCaoSQL のネイティブコード
	クライアントから参照できる場所に置く. または, 環境変数の%**PATH**%に場所を登録します.
log4cpp.dll	ログ情報を出力するためのライブラリ
	環境変数の%**PATH**%に場所を登録します.
jcaosql_log.config	ネイティブコードのログ設定情報
	必須ではないが, ネイティブコードのログ情報を必要とする場合は, アプリケーション実行フォルダに置きます.

具体的なクライアントの開発方法については, 以下のドキュメントと, パッケージに追加されているsampleフォルダを参考にして下さい.

5.2. サンプル解説

5.2.1. インスタンスの生成と破棄

CaoSQLEngine, CaoSQLController, CaoSQLItem オブジェクトは, それぞれ, CaoSQLEngine#createInstance, CaoSQLEngine#getController, CaoSQLController#getItem がファクトリメソッドになっていて, この際に対応する COM のインスタンスが生成されます. 従って, これらのオブジェクトを破棄する際には必ずreleaseをコールする必要があります. javaではデストラクタの概念が無い為, クライアントアプリケーションで明示的にコールして下さい.

このサンプルは sample¥sample1 フォルダにあり, sample フォルダから以下のコマンドでサンプルの実行を確認することができます.

ant sample1

5.2.2. VARIANT 型の対応

jCaoSQL では VARIANT クラスを java.lang.Object クラスにマッピングしています. 詳細な対応は表 4-1 を確認して下さい. ここでは, java クライアント側での VT 型の判別サンプルを示します.

java では instanceof 演算子を使うことで, オブジェクトの型を持つかどうか判定することが可能です. 例えば, VT_I4 型の VARIANT であれば java.lang.Integer クラスにマッピングされている筈なので,

```
if (value instanceof Integer) {
        // VT_I4 型変数
}
```

という処理で VT_I4 であることが判別可能です.

また, 配列型に関してはオブジェクトから java.lang.Class を取得し, この Class オブジェクトのメソッド isArray を呼ぶことで, 配列型かどうか判別可能です.

このサンプルは sample¥sample2 フォルダにあり, sample フォルダから以下のコマンドでサンプルの実行を確認することができます.

ant sample2

5.2.3. イベントシンクの作成

jCaoSQL ではサンプルのようにイベント受信が可能ですが, CaoSQL は非同期的にイベントを発生させるので, java オブジェクト側で同期を取りたい場合は jp.co.denso.fa.CaoSQLControllerListener の実装時に OnChangeItem メソッド, OnChangeState メソッド, あるいはメソッド内で適宜 synchronized を宣言し, 同期を図って下さい.

また, OnChangeItem イベントの引数で渡された CaoSQLItem オブジェクトは 5.2.1 と同じく破棄(明示的な release のコール)しなければなりません.

このサンプルは sample¥sample3 フォルダにあり, sample フォルダから以下のコマンドでサンプルの実行を確認することができます.

ant sample3

5.2.4. カスタムクラスの作成

jCaoSQL を作成している際に, CaoSQLjava オブジェクトで独自の機能を実装したカスタムクラスを作成した方が設計や実装上で便利な場合があります. こういった場合は jcaosql.properties にカスタムクラスを定義し, 既存の CaoSQL java オブジェクトの派生クラスを作成することで実現が可能です.

以下にその方法を示します.

表 5-2 jcaosql.properties のキー一覧

キー名	意味
caosql.engine.iid	CaoSQLEngine のインタフェース ID（通常は変更不要）
caosql.engine.progid	CaoSQLEngine のプログラム ID（通常は変更不要）

caosql.engine.class	ファクトリが作成する CaoSQLEngin クラス デフォルトは”jp.co.denso.fa.jcaosql.CaoSQLEngine”
caosql.controller.class	ファクトリが作成しようとする CaoSQLController クラス デフォルトは”jp.co.denso.fa.jcaosql.CaoSQLController”
caosql.item.class	ファクトリが作成しようとする CaoSQLItem クラス デフォルトは”jp.co.denso.fa.jcaosql.CaoSQLItem”

例えば，CaoSQLItem を基底クラスとする com.example.MyCaoSQLItem クラスを使用したい場合は jcaosql.properties という名前のファイルを作成し，キー<caosql.item.class>に，値<com.example.MyCaoSQL>を書き，実行時のルートにこの properties ファイルを配置します．この処置でファクトリクラスは CaoSQLItem の作成時に com.example.MyCaoSQLItem クラスを作成するようになります．

このサンプルは sample¥sample4フォルダにあり，sample フォルダから以下のコマンドでサンプルの実行を確認することができます．

ant sample4

5.2.5. アイテムの動的追加，削除

jCaoSQL は CaoSQL と同じく動的にアイテムを追加する事が可能です．削除可能なアイテムが動的に追加したアイテムのみであるのも CaoSQL と同じです．

このサンプルは sample¥sample5フォルダにあり，sample フォルダから以下のコマンドでサンプルの実行を確認することができます．

ant sample5

6. RMI 連携機能

6.1. 概要

Linux などの非 Windows 系 OS から CaoSQL.exe へアクセスする場合は RMI 連携機能を使用することで通信が可能です.

下記に Windows OS から CaoSQL.exe へアクセスする場合と Linux OS から CaoSQL.exe へアクセスする場合の仕組みの違いを示します.

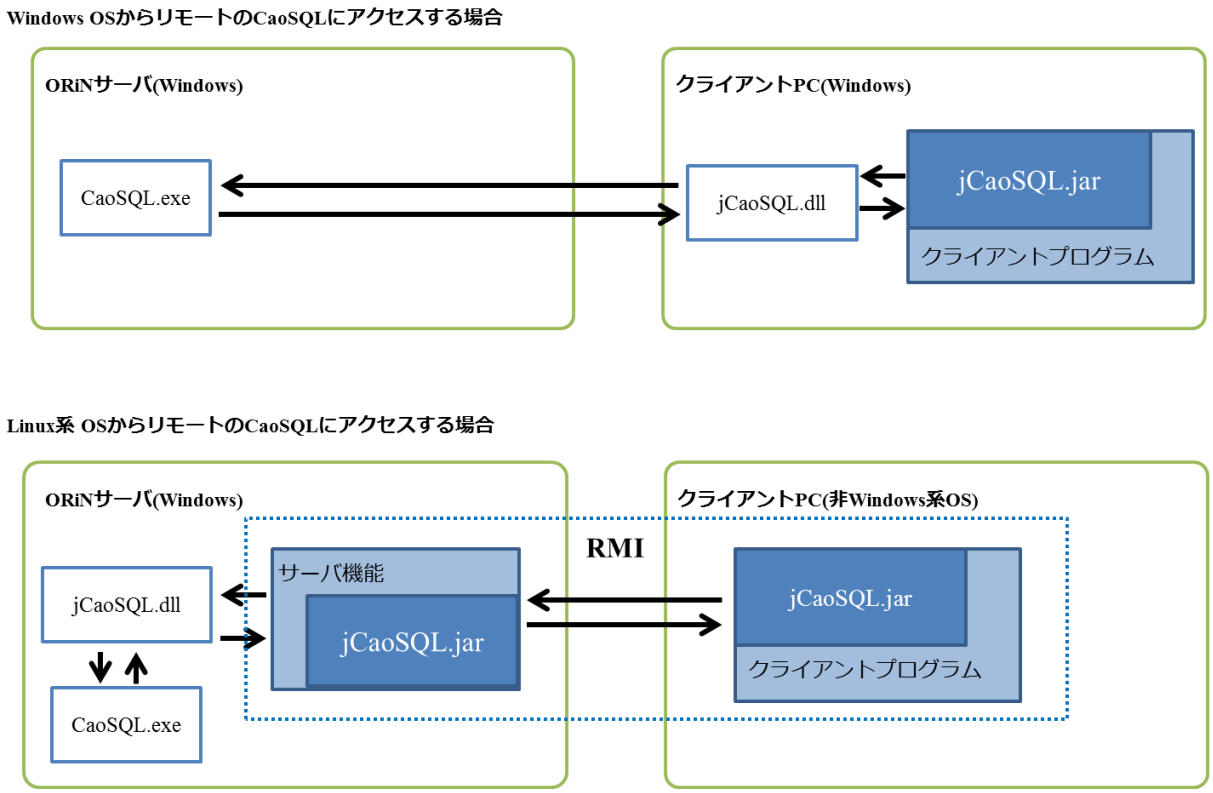


図 6-1 RMI 連携機能 概要

jCaoSQL では CaoSQL.exe と COM 通信を行うため,jCaoSQL.jar と jCaoSQL.dll で構成されています.
非 Windows 系 OS では dll の実行ができない為,ORiN サーバ内の jCaoSQL.jar をサーバ機能として実行することで jCaoSQL.jar の関数をリモートで実行し CaoSQL へアクセスします.
クライアント側のプログラムは使用パッケージの切り替えによって非 Windows 系 OS からでも実行可能な Java アプリケーションへ切り替えることが可能です.

一部,実装の変更が必要な箇所があります.詳細は 6.3.1 を参照下さい

6.2. 実行環境のセットアップ

6.2.1. ORiN サーバ側の環境構築

表 6-1 にあるファイルを使用し,jCaoSQLServer.cmd を実行します.

表 6-1jCaoSQL の使用ファイル(ORiN サーバ側)

ファイル名	概要
	設定方法
jCaoSQL.jar	jCaoSQL の java バイナリパッケージ
	jCaoSQLServer.cmd から参照できる場所に置く. または, 環境変数の%**PATH**%に場所を登録します.
jCaoSQL.dll	jCaoSQL のネイティブコード
	jCaoSQL.jar から参照できる場所に置く. または, 環境変数の%**PATH**%に場所を登録します.
policy.txt	サーバ機能を起動させるためのセキュリティポリシー
	jCaoSQLServerStart.cmd と同じディレクトリに置きます.
jCaoSQLServer.cmd	サーバ機能を起動させるためのバッチファイル
	アプリケーション実行フォルダに置きます. jCaoSQL.jar へのパスを -Djava.rmi.server.codebase へ,policy.txt へのパスを-Djava.security.policy へ指定します.
log4cpp.dll	ログ情報を出力するためのライブラリ
	環境変数の%**PATH**%に場所を登録します.
jCaosql_log.config	ネイティブコードのログ設定情報
	必須ではないが, ネイティブコードのログ情報を必要とする場合は, アプリケーション実行フォルダに置きます.

6.2.2. クライアント側の環境構築

表 6-2 にあるファイルを使用し,クライアントプログラムの実装を行います.

表 6-2jCaoSQL の使用ファイル(クライアント側)

ファイル名	概要
	設定方法
jCaoSQL.jar	jCaoSQL の java バイナリパッケージ
	ClassPath に通します.

6.3. 実装

6.3.1. 実装方法

RMI 連携機能は Java RMI を使用して ORiN サーバ側の関数をリモートで呼び出しています.
その為,下記の点に留意して実装を行ってください.

- ライブラリの参照

  下記 2 ファイルは ORiN サーバ側,クライアント側で同一のバージョンに統一します.

  ・jCaoSQL.jar

  ・policy.txt

  ※可能であれば,ORiN サーバ側,クライアント側の双方から同一のファイルが参照できる場所に配置します.

- 使用パッケージ

  RMI 連携機能を使用して CaoSQL.exe にアクセスするには jp.co.denso.fa.jcaosql.rmi 以下の API を使用します.既存のプログラムを変更の場合は使用パッケージの変更(import 文の書き換え)よって対応することが可能です.

  (例)

  ```
  // import jp.co.denso.fa.jcaosql.*;
  import jp.co.denso.fa.jcaosql.rmi.*;
  ```

  ※クラス名は同一となっておりますので,混同しないようにご注意ください.

- **CaoSQLEngine.createInstance()メソッド**

  引数に”Conn＝< RMI サーバ名(ORiN サーバの IP アドレスもしくはマシン名)>”を指定します.
  (例)localhost につなぐ場合

  CaoSQLEngine.createInstance(“Conn=localhost”)

- **JavaVM のセキュリティポリシー設定**

  同梱の policy.txt を Java VM 実行時引数として渡す,もしくは実行する Java のセキュリティポリシーファイル(%JAVA_HOME%/jre/lib/security/java.policy)に設定を行います.

  ✧ Java VM 実行時引数として渡す場合

  引数に下記を指定します.

  -Djava.security.manager -Djava.security.policy=《policy.txt のパス》

  ✧ Java のセキュリティポリシーファイルを設定する場合

  %JAVA_HOME%/jre/lib/security/java.policy に下記を追記します.

```
grant {
        permission java.lang.RuntimePermission "loadLibrary.jCaoSQL";
        permission java.lang.RuntimePermission "createSecurityManager";
        permission java.lang.RuntimePermission "setSecurityManager";
        permission java.util.PropertyPermission "user.dir", "read";
        permission java.util.PropertyPermission
                "java.rmi.server.hostname", "read";
        permission java.net.SocketPermission
                "< RMI サーバ名(ORiN サーバの IP アドレスもしくはマシン
                名>:1024-65535","connect,accept";
        permission java.net.SocketPermission
                "<クライアント名>:1024-65535","connect,accept";
        };
```

  ※サーバ側・クライアント側双方でセキュリティポリシーの設定が必要です.

- **サーバ機能の実行**

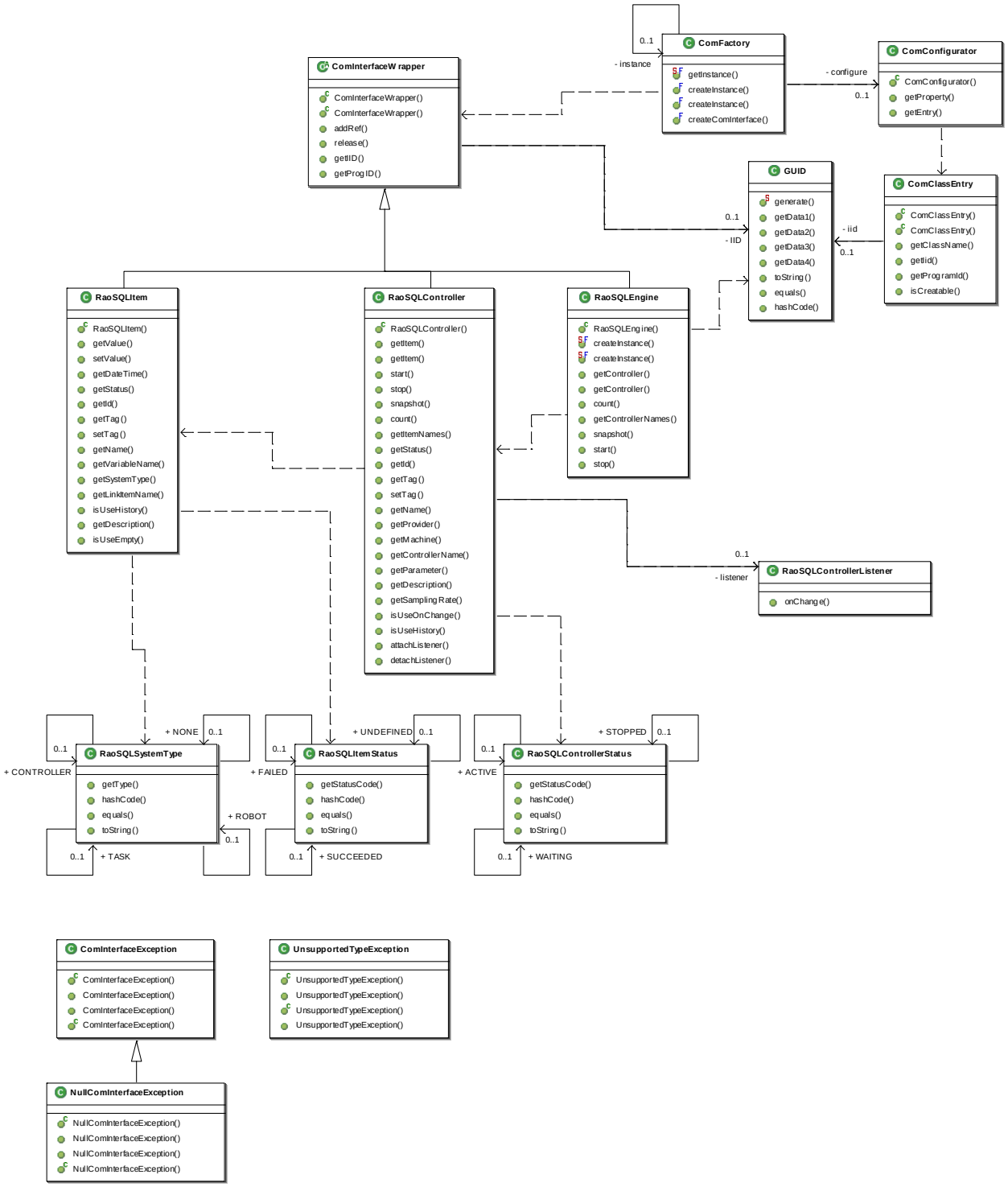
  ORiN サーバ側で jCaoSQLServer.cmd を実行します.

※その他の点については通常の jCaoSQL と同様の仕様で動作します.詳細は「4.jCaoSQL フィーチャー」を参照してください.

6.3.2. サンプルコード

```
public class Main {
	public static void main(String[] args) {
		CaoSQLEngine eng = null;
		CaoSQLController ctrl = null;
		CaoSQLItem item = null;
		try {
			eng = CaoSQLEngine.createInstance("localhost");  //CaoSQLEngine インスタンスの取得
			String[] ctrl_names = eng.getControllerNames();  //Controller 名一覧の取得
			ctrl = eng.Controller(ctrl_names[0]);  //CaoSQLController インスタンスの取得
			String[] item_names = ctrl.getItemNames();  //Item 名一覧の取得
			item = ctrl.Item(item_names[0]);  //CaoSQLItem インスタンスの取得
			Object value = new Integer(100);
			item.putValue(value);  //Item へ値の書き込み
			//Item から設定値の取得
			System.out.println("VariableName: " + item.getSettingData(new Integer(0)));
			//Item から値の取得
			System.out.println("Value: " + item.getValue().toString());
			//Item から値更新時刻の取得
			System.out.println("Updated: " + item.getDateTime().toString());
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			if (item != null) item.release();
			if (ctrl != null) ctrl.release();
			if (eng != null) eng.release();
		}
	}
```

付録A. クラス図


ComInterfaceW rapper
ComInterfaceWrapper()
ComInterfaceWrapper()
addRef()
release()
getIID()
getProgID()
ComFactory
getInstance()
createInstance()
createInstance()
createComInterface()
- instance
0..1
- configure
0..1
ComConfigurator
ComConfigurator()
getProperty()
getEntry()
GUID
generate()
getData1()
getData2()
getData3()
getData4()
toString()
equals()
hashCode()
0..1
- IID
- iid
0..1
ComClassEntry
ComClassEntry()
ComClassEntry()
getClassName()
getIid()
getProgramId()
isCreatable()
RaoSQLItem
RaoSQLItem()
getValue()
setValue()
getDateTime()
getStatus()
getId()
getTag()
setTag()
getName()
getVariableName()
getSystemType()
getLinkItemName()
isUseHistory()
getDescription()
isUseEmpty()
RaoSQLController
RaoSQLController()
getItem()
getItem()
start()
stop()
snapshot()
count()
getItemNames()
getStatus()
getId()
getTag()
setTag()
getName()
getProvider()
getMachine()
getControllerName()
getParameter()
getDescription()
getSamplingRate()
isUseOnChange()
isUseHistory()
attachListener()
detachListener()
RaoSQLEngine
RaoSQLEngine()
createInstance()
createInstance()
getController()
getController()
count()
getControllerNames()
snapshot()
start()
stop()
0..1
- listener
RaoSQLControllerListener
onChange()
+ NONE
0..1
0..1
+ CONTROLLER
RaoSQLSystemType
getType()
hashCode()
equals()
toString()
+ ROBOT
0..1
0..1
+ TASK
+ UNDEFINED
0..1
0..1
+ FAILED
RaoSQLItemStatus
getStatusCode()
hashCode()
equals()
toString()
0..1
+ SUCCEEDED
+ STOPPED
0..1
0..1
+ ACTIVE
RaoSQLControllerStatus
getStatusCode()
hashCode()
equals()
toString()
0..1
+ WAITING
ComInterfaceException
ComInterfaceException()
ComInterfaceException()
ComInterfaceException()
ComInterfaceException()
UnsupportedTypeException
UnsupportedTypeException()
UnsupportedTypeException()
UnsupportedTypeException()
UnsupportedTypeException()
NullComInterfaceException
NullComInterfaceException()
NullComInterfaceException()
NullComInterfaceException()
NullComInterfaceException()

付録B. オープンソースの著作権情報

jCaoSQL.dll はオープンソースソフト log4cpp を使用しています。以下に著作情報を表示します。

Log for C++ (short name: log4cpp), a C++ library for flexible logging.

Version 1.1.2

See http://log4cpp.sf.net/

==========

log4cpp は添付の License.txt に示す LGPL の元で公開されています。