DB Connector for iOS/Android

Linux version 3.5

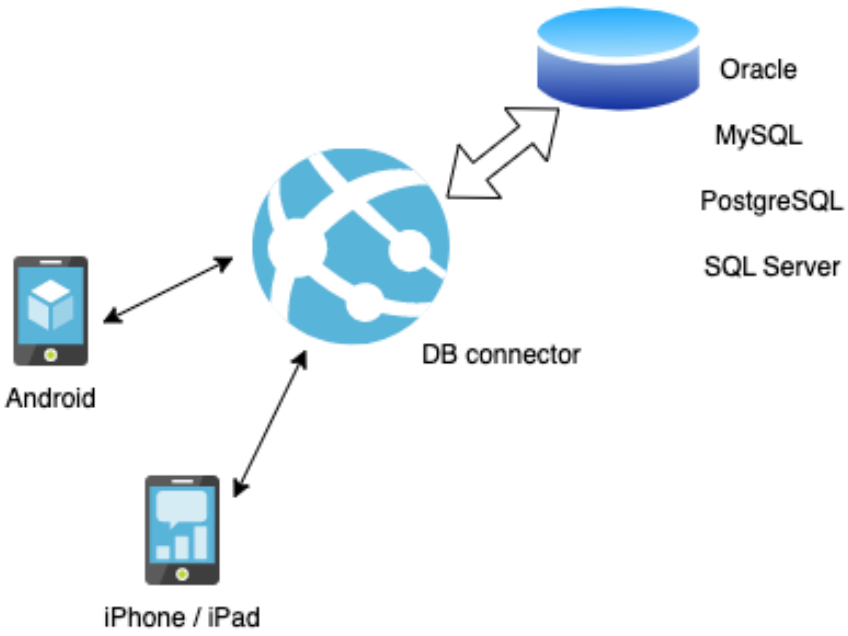
TechKnowledge

目次

はじめに

DB connector for iOS/Androidについて

本製品はiOSやAndroidモバイルアプリとLinuxサーバー上にあるOracle MySQL PostgresSQL SQL Server などのデータベースを接続するミドルウェアです。既存のSQLスキルと最小限のAPIでスマートフォンアプリを作れます。アプリケーションロジックをモバイルアプリに集約することができます。



以下は製品の概要です。

1. Linux サーバー上で稼働。
2. Oracle/SQL Server/MySql/PostgreSQLをサポート。
3. クライアントライブラリは以下の形式で提供します。
   - Xcode/Objective-Cに対応ユニバーサルスタティックライブラリ (arm64, x86_X64)
   - Android SDK 対応 .aar ファイル
   - flutter/dart 対応 package
4. iOSクライアントライブラリは非同期通信でdelegateに処理完了などが通知されます。
5. トランザクションサポートしてますので更新系業務も対応可能です。
6. TLS対応。
7. BASIC認証対応。
8. flutterライブラリはdartで記述されておりflutterがサポートするプラットフォーム全般で動作します。
9. 通信プロトコルはOracle Connector/ MSSQL Connectorと互換性があります。クライアントライブラリ仕様も同一です。

ご利用上の注意事項

本製品では以下のような注意事項・ご利用制限がございます。

1. BLOB/CLOBなどラージオブジェクトには対応していません。
   本製品についてはHTTP Soapプロトコルで通信しているのでバイナリ転送はサポートしません。

2. Xcodeバージョンアップについて
   例年9月のXcodeバージョンアップに伴いiOS frameworkバージョンアップによる仕様変更で動作が変わることがあります。迅速に対応しますのでサポートにお問い合わせください。

3. futter バージョンアップについて
   頻繁にアップデートがありますのでgithubに対応版をアップロードする予定です。

本製品の使用許諾について

本製品のご利用ライセンスはサーバー毎となります。１台のサーバーに複数のモバイルデバイスから接続数は無制限でご利用いただけます。開発ライブラリは複数の開発環境でご利用いただけます。複数サーバーでの運用、異なるアプリケーションを別サーバーで運用する場合などはご利用サーバー数と同じ数のライセンスを販売会社システムラボからご購入ください。

禁止事項

1. 本製品の不正複製を禁止します。
2. 本製品のリバースエンジニアリングを禁止します。
3. 本製品をラップし同種の製品を作成し販売することを禁止します。

保証規定

本製品、および付随する著作物に対して商品性及び特定の目的への適合性などについての保証を含むいかなる保証もそれを明記するしないに関わらず提供されることはありません。

本製品の著作者及び、製造、配布に関わるいかなる者も、当ソフトウェアの不具合によって発生する損害に対する責任は、それが直接的であるか間接的であるか、必然的であるか偶発的であるかに関わらず、負わないものとします。それは、その損害の可能性について、開発会社に事前に知らされていた場合でも同様です。

プロダクト・サポート

- ユーザー登録
  まことにお手数ですが販売会社システム・ラボにてユーザー登録をお願いします。ユーザー登録が行われていないとお客様がユーザー・サポートが受けられない場合がございます。

- お問い合わせの方法
  解決できない問題が発生した場合には、技術サポートをご利用ください。あらかじめ後ページの調査依頼書にお問い合わせ事項を記入していただき、インターネット・メールまたはファックスでお送りいただければ、折り返しご連絡をさせていただきます。**本製品につきましては、複雑な内容のお問い合わせになることが多い為、電話によるユーザーサポートは実施しておりません。ご了承をお願いいたします。**また、問い合わせの内容によっては、再現調査などのために、回答までに時間がかかる場合がありますので、かさねてご了承をお願いいたします。

  サポートメールアドレス：　support@techknowledge.co.jp

- 登録内容の変更について
  転居などによるご住所や電話番号など登録内容に変更が生じた場合には、メールまたはファックスにて、販売会社システム・ラボまでご連絡をいただきますようお願いいたします。なお、電話による口頭での連絡変更は受けかねますのでよろしくお願いいたします。

- 併用される他社製品について
  当社製品と併用される、他社製品の使用方等についてのご質問をお受けすることがあります。しかし、他社製品に関しましては、お答えできない場合があります。他社製品につきましては、該当開発・販売会社にご連絡ください。

- サポート対象
  ご質問はご登録ユーザー様に限定させていただきます。ご登録ユーザー様以外からのご質問にはお答えできません。当ソフトウェアの料金にはご登録ユーザー様１名に限りサポート料が含まれています。

- サポート期間
  製品のユーザー登録後、初回のお問い合わせから90日間は無償サポート期間とさせていただきます。また無償サポートは2件を上限とさせていただきます。無償サポート上限を超える場合には無償サポート終了以降もサポートをご希望の場合は有償サポートを承ります。有償サポートにつきましては販社システム・ラボにてお取り扱いしております。キャンペーン製品などディスカウント販売に該当する製品では無償サポート期間および回数の設定が短くなる場合がありますのであらかじめご了承ください。

- 最新版のご提供について
  弊社webにて最新版の実行モジュールや技術情報、サンプル・コードの提供をしてしておりますのでサポートにご連絡になる前に弊社webをご参照いただけるようお願いいたします。URLはhttp://www.techknowledge.co.jpとなります。

販売元

（株）システムラボ

東京都北区田端6-1-1 田端アスカタワー12F

電話	03-5809-0893
FAX	03-4578-9261
Internet-Mail	info@systemlab.co.jp
URL	www.systemlab.co.jp

開発元、サポート

（株）テクナレッジ

東京都世田谷区駒沢２丁目１６番１号　サンドービル９Ｆ

電話	03-3421-7621
FAX	03-3421-6691
Internet-Mail	info@techknowledge.co.jp
URL	www.techknowledge.co.jp

インストール

DB connector iOS/Android Linux版のインストールについてご説明します。

サーバーシステム要件

以下のソフトウェアがサーバーシステム構成の要件となります。

1. Linux X64 サーバー (Kernel version 2.6.23 以降)
2. Oracle または MySql または PostgreSQL または SQL Server

モバイルデバイス要件

Apple社のモバイルデバイスとして以下をサーバーに接続可能です。

1. iPhone/iPad/iPod touch iOS 12以降
2. Android 携帯およびAndroidタブレット Android OS 6以降

開発環境

iOSモバイルデバイス向けアプリケーション開発環境としては以下が必要になります。Apple社のMac App Storeから無償でご利用いただけます。

Apple Xcode 14 以降

モバイルデバイス実機でのデバッグやアプリケーションの配布にはApple Developer Program の有償会員になる必要があります。（年次契約）

Apple Developer Program のウェブサイト

https://developer.apple.com/jp/

Androidモバイルデバイス向けアプリケーション開発環境としてはAndroid Studio が必要となります。開発環境の設定等についてはGoogleの該当サイトをご参照ください。

https://developer.android.com/studio/

Flutterでの開発に関してはAndroid StudioまたはVS code等で開発環境を整えてください。flutter 2.10.5～3.10.0で動作確認しました。

サーバーソフトウェアインストール

製品パッケージのZIPを展開します。以下が含まれます。

1. サーバー実行ファイル
2. サーバー設定サンプルファイル
3. システムサービス設定ファイル
4. インストールシェルスクリプト

実行ファイルはどのディレクトリでも動作しますが想定している設置ディレクトリは以下です。install.shで以下の配置を実行しサービス登録します。

#	ファイル名とパス	内容
1	/usr/local/bin/db_proxy	データベースプロキシーサーバー
2	/usr/local/etc/db_proxy.ini	サーバー設定ファイル
3	/etc/systemd/system/db_proxy.service	Linux サービス設定ファイル

db_proxy.ini サーバー設定ファイルについて

サーバー動作設定ファイルは　/usr/local/etc/db_proxy.ini　をデフォルトで参照します。実行ファイルdb_proxyには -c オプションで指定することができます。設定項目とその説明は以下です。

#	セクション	キー	値
1	basic_auth	uid	ベーシック認証ユーザーID。利用しない場合はキーを削除または空文字列を設定。
2	basic_auth	pwd	ベーシック認証パスワード。
3	license	key	ライセンスキー。指定しない場合はトライアル版モード。
4	license	file	ライセンスファイル。フルパス指定。
5	server	auth_table_name	ユーザー毎認証する場合のテーブル名。
6	server	constr	データベース接続文字列。詳細は後述。
7	server	db	データベースタイプを設定。”oracle”, “mysql”,”postgres”,”sqlserver”のいずれかを設定。
8	server	interface	バインドするインターフェースアドレスを指定。”127.0.0.1" または “0.0.0.0”
9	server	logfile	ログ出力ファイルを指定。デフォルト /var/log/db_proxy.log
10	server	lower_case_column	tune 設定にするとカラム名を小文字で返す。デフォルトは大文字。
11	server	max_rows	QueryRows の結果行数の最大値。0設定は制限なし。デフォルトは0。

#	セクション	キー	値
12	server	port	サーバーにオープンするポート番号。
13	server	verbose	true 設定でログに詳細情報を出力します。
14	ssl	certfile	TLS利用時の証明書ファイル。TLS利用しない場合はキーを定義しないか空文字設定。
15	ssl	keyfile	TLS利用時の証明書キーファイル。

サンプル設定ファイル

```
[ssl]
certFile = "/etc/letsencrypt/live/domain.co.jp/fullchain.pem"
keyFile = "/etc/letsencrypt/live/domain.co.jp/privkey.pem"

[server]
port = 443
db = "mysql"
constr = "root:@(localhost:3306)/dbname?charset=utf8mb4&interpolateParams=true"
lower_case_column = false
logfile = "/var/log/db_proxy.log"
max_rows = 10000
interface = “127.0.0.1"
verbose = false

[license]
key = “1234567890”
file = “/usr/local/etc/license.txt”

[basic_auth]
uid = "uid"
pwd = "pass"
```

データベース接続文字列について

本サーバープログラムはgo言語で database/sql を使っています。接続文字列はそれぞれのデータベースのドライバーの仕様に依存します。サーバープログラムでは以下をインポートしています。

```
_ "github.com/sijms/go-ora/v2"
```

_ "github.com/go-sql-driver/mysql"
_ "github.com/lib/pq"
_ “github.com/microsoft/go-mssqldb”

以下はデータベース毎にlocalhostへscott: tiger で接続する接続文字列例です。

DB	接続文字列例
MySQL	scott:tiger@(localhost:3306)/test?charset=utf8mb4&interpolateParams=true
PostgreSQL	user=scott password=tiger dbname=postgres sslmode=disable
Oracle	oracle://scott:tiger@localhost:1521/pdb01
SQL Server	sqlserver://scott:tiger@localhost?database=master&connection+timeout=30

サポートデータベースバージョン

当バージョン 3.5リリース時点での情報です。

DB	バージョン
MySQL	5.6 以降
PostgreSQL	11 以降
Oracle	10.2 以降
SQL Server	2017 以降

ユーザー毎認証

ログイン実行時にuid, pwd プロパティを指定してユーザー毎認証ができます。サーバーのデータベースに認証テーブルを用意してテーブル名を設定ファイルのauth_table_name に指定します。認証テーブルには文字列型で uid ,pwd カラムを用意します。ログイン時にuid でクエリして得られるpwdをクライアントからの値と比較して合致すれば認証OKとします。パスワードはmd5などで適宜ハッシュや暗号化などを導入されることをお勧めします。

サブフォルダーへの配置

本サーバーをサーバーurlのルート以外に設置したい場合はnginxを利用してください。以下はngnixのconfファイル例です。

```
upstream db_proxy_server {
    server 127.0.0.1:8080;
}

server {

        index index.html;
        server_name _;

        location / {
                root /var/www/html;
        }

        location /api {
                proxy_pass https://db_proxy_server;
        }

        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/server.jp/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/server.jp/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

}
```

linux サービス登録と起動

インストールイメージにあるdb_proxy.servce ファイルを /etc/systemd/system へコピーして以下のコマンドを実行してください。

```
sudo systemctl enable db_proxy
sudo systemctl start db_proxy
```

サンプルコードとクライアントライブラリ

githubにありますのでダウンロードしてご利用ください。

https://github.com/techknowledge-dev/db_connector_samples

サンプルコードは昔のOracleにあったemp表へのアクセスサンプルコードです。sql フォルダーに各データベース向けテーブル作成sqlがあります。

1. Android サンプル
2. Objective-C サンプル
3. swift サンプル
4. flutter サンプル
5. emp 表作成 sql

サンプルコードにはクライアントライブラリが含まれています。flutterサンプルはpubspec.yamlからgithubライブラリを参照する設定です。

モバイルデバイスプログラミング

サンプルコードは昔のOracleにあったEMPテーブルにクエリを実行します。EMPテーブルを作成してデータを追加する必要がある場合はサンプルコードgithubのsqlフォルダーにテーブルを生成、データを挿入するsqlがありますのでご利用ください。

iOS swift/objective-cサンプルコード

iOS向けサンプルコードはgithubのobjc/swift.フォルダーに以下の2サンプルがあります。

1. querySample
2. queryRowsSample

ダウンロードしてXcodeで.xcodeprojファイルを読み込んでください。urlプロパティに設定しているサーバーのアドレス等は変更する必要があります。（プライベートメソッド startConnect でurlプロパティを設定しています）

EMP表にアクセスするようにサーバー側の接続文字列を設定していただければすぐに動作するようなります。

Xcodeプロジェクトでの設定について

Xcodeのプロジェクトから本製品でデータベースアクセス機能を追加するには以下の2ファイルをプロジェクトに追加してください。

1. dblib.h
2. libdblib.aar

これらのファイルはサンプルコードやサーバー側にinclude/lib　フォルダーに保存されています。liboralib.aはデバイス用とシュミレータ用に別々のファイルがご用意されておりますのでご注意ください。ヘダーファイルは共通でご利用いただけます。

Info.plist 設定

iOS14以降のデバイスでローカルネットワークを使う場合の設定例です。

```
<key>NSLocalNetworkUsageDescription</key>
<string>${PRODUCT_NAME} uses the local network.</string>
```

Androidサンプルコード

Android向けサンプルコードはgithubのandroidフォルダーに以下の3サンプルがあります。

1. querySample
2. queryRowsSample
3. executeSample

それぞれ展開してAndroid　StudioからJavaプロジェクトをインポートしてください。dbLibクラスのurlプロパティに設定しているサーバーのアドレス等は変更する必要があります。

OracleのサンプルEMP表にアクセスするようにサーバー側の接続文字列を設定していただければすぐに動作するようなります。

Androidアプリケーション参照について

アプリケーションプロジェクトのLibsには以下のaarファイルを追加してください。

dblib.aar

ソースコード上では以下のimport が必要になります。

```
import jp.co.techknowledge.dbLib;
import jp.co.techknowledge.dbLib.Row;
import jp.co.techknowledge.dbLib.Column;
import jp.co.techknowledge.dbLib.status;
```

Build.gradle dependency 追加について

okhttp3を使っていますので以下をdependencyへ追加してください。

```
dependencies {
  // snip …
  implementation 'com.squareup.okhttp3:okhttp:4.9.0'
  implementation 'com.squareup.okhttp3:okhttp-urlconnection:4.9.0'
  implementation fileTree(dir: 'libs', include: ['*.aar', '*.jar'], exclude: [])
}
```

AndroidManifest 設定について

当クラスライブラリの利用にあたって以下のパーミション指定を必要とします。

. INTERNET

Android.Manifestファイルでの設定例は以下です。

```
<uses-permission android:name="android.permission.INTERNET" />
```

ローカルサーバーとSSLなしで接続する場合は以下を追加してください。

```
android:usesCleartextTraffic="true"
```

flutter / dart サンプルコード

サンプルコードはgithubにflutterフォルダーにありますのでご参照ください。null safety 対応です。
pubspec.yaml の dependency: に以下のように設定してください。

```
dependencies:
  dblib:
    git:
      url: https://github.com/techknowledge-dev/dblib_dart.git
```

github が最新版をキャッシュしない場合はgithubからZIPでソースをダウンロードしてpath 指定で解決できます。

各プラットフォームでのビルド・実行方法はflutterのウェブなどをご確認ください。

https://docs.flutter.dev/get-started/codelab

実デバイスで必要なネットワーク利用系のパーミッションはiOS/Androidの項と同様です。

iOSクラスリファレンス

メソッド

beginTrans

書式
-(dbLibStatus) beginTrans

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションを開始します。サーバーには接続完了状態、トランザクションクローズ状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

commitTrans

書式
-(dbLibStatus) commitTrans

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションをコミットします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

connect

書式
-(dbLibStatus) connect

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
プロパティURLで指定されるサーバーに接続します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

disconnect

書式
-(dbLibStatus) disconnect

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
サーバー接続を遮断します。サーバー側ではDB接続の遮断、トランザクション中の場合はトランザクションの破棄が実行されます。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

execute

書式
-(dbLibStatus) execute:(NSString*) sql

パラメータ

非クエリ系 SQL

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
非クエリ系SQLを実行します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

query

書式
-(dbLibStatus) query:(NSString*) sql

戻り値

Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行して1行ずつrowFetchedデリゲートにデータが通知されます。データの読み込みが完了するとqueryCompletedデリゲートに通知されます。

queryRows

書式
-(dbLibStatus) queryRows:(NSString*) sql: (NSInteger) maxRows

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行して複数行をqueryRowsCompletedデリゲートに通知します。HTTPプロトコルで転送されるデータ量に上限がありますので一定のデータ量で制限したいときには第2パラメータmaxRowsに取得するデータ行の最大数を設定します。maxRowsにゼロ設定の場合は全データを取得します。

rollbackTrans

書式
-(dbfLibStatus) rollbackTrans

戻り値
Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションをロールバックします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

プロパティ

lastErrorString

データ型
NSString*

解説

本製品のメソッドを呼び出してエラーが発生した場合にその理由が明確な場合はこのプロパティに保持されます。エラー情報は以下となります。

1. ネットワークエラー情報
2. サーバー側エラー情報
3. Oracleエラーメッセージ

lowerColName

データ型

bool

解説

True設定時にクエリー結果のカラム名を小文字に変換します。

pwd

データ型

NSString*

解説

ユーザー毎認証を使う場合にはこのプロパティにパスワードを設定してください。

rawUrl

データ型

bool

解説

Windows　サーバー版と互換性のために存在するプロパティです。false指定しない場合はurlに/dbLibSerber.asmxを追加指定します。

results

データ型

NSMutableArray*

解説

Query　/　QueryRowsメソッドの結果を保持します。配列の要素はNSMutableDirectory*型でカラム名指定でカラムデータを取得する事が出来ます。

uid

データ型
NSString*

解説
ユーザー毎認証を使う場合にはこのプロパティにユーザーIDを設定してください。

url

データ型
NSString*

解説
接続先のサーバURLを指定します。（例： http://some.server.jp/） サーバーの構成によりSSL接続,ポート番号,BASIC認証などを指定します。

verbose

データ型
bool

解説
サーバーから受信したXMLデータ内容をログに出力します。

デリゲート

queryCompleted

書式
-(void) queryCompleted

戻り値
なし。

解説
queryメソッドが完了したときに呼び出されます。データがそろった状態なのでUITableViewのリロードなどを呼び出します。

queryRowsCompleted

書式
-(void) queryRowsCompleted:(NSMutableArray*) rows

戻り値
なし。

解説
queryRowsメソッドが完了したときに呼び出されます。パラメータにはNSMutableArrayで取得したレコード行が複数返されます。NSMutableArrayが保持するのはNSDictionaryオブジェクトでカラム名をキーとしてデータを取得します。キーのカラム名は通常大文字となります。以下サンプルコードです。

```
NSMutableDictionary* row = [_rows objectAtIndex:indexPath.row];
NSString* empno = [row objectForKey:@"EMPNO"];
NSString* ename = [row objectForKey:@"ENAME"];
cell.textLabel.text = [NSString stringWithFormat:@"%@ %@", empno, ename];
```

requestCompleted

書式
-(void) requestCompleted:(NSString*)methodName;

戻り値
なし。

解説

メソッドの正常完了を通知するデリゲートです。クエリ系以外のメソッドでこのデリゲートに通知となります。methodNameには発行したメソッドの名前が通知されます。ただしconnectとdisconnectは通知メソッド名Login/Logoutにとなります。

requestFailed

書式
-(void) requestFailed:(NSString*)methodName :(NSError*) err

戻り値
なし。

解説

メソッドの異常終了を通知するデリゲートです。クエリ系以外のメソッドでこのデリゲートに通知とな

ります。methodNameには発行したメソッドの名前が通知されます。ただしconnectとdisconnectは通知メソッド名Login/Logoutにとなります。NSErrorについてはネットワーク系エラー以外はnilが指定されます。NSErrorの詳細はAppleのマニュアルをご参照ください。

rowFetched

書式
-(bool) rowFetched:(NSMutableDictionary*) row

戻り値
Trueを返すと次のレコードを取得します。Falseを返すとこのレコードで終了となります。

解説
queryメソッド実行後にレコードを受信するたびにこのデリゲートが呼び出されます。パラメータにはレコードイメージがNSDictionary型で保持されます。キーはカラム名となります。カラム名は通常大文字になります。lowerColNameプロパティのTrue設定により小文字に変換することが出来ます。以下はサンプルコードです。

```
NSString* empno = [row objectForKey:@"EMPNO"];
NSString* ename = [row objectForKey:@"ENAME"];
```

Androidクラスリファレンス

dbLibクラスメソッドの呼び出しにつきましてはサーバーとの通信が伴いますのでメインスレッド以外から呼び出すように実装してください。

メソッド

beginTrans

書式
status beginTrans();

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションを開始します。サーバーには接続完了状態、トランザクションクローズ状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

commitTrans

書式
status commitTrans();

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションをコミットします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

connect

書式
status connect();

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
プロパティURLで指定されるサーバーに接続します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

disconnect

書式
status disconnect();

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
サーバー接続を遮断します。サーバー側ではDB接続の遮断、トランザクション中の場合はトランザクションの破棄が実行されます。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

execute

書式
status execute(String sql);

パラメータ

非クエリ系 SQL

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
非クエリ系SQLを実行します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

query

書式

status query(String sql);

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行して1行ずつrowFetchedデリゲートにデータが通知されます。データの読み込みが完了するとqueryCompletedデリゲートに通知されます。

queryRows

書式
status queryRows(String sql, int maxRows);

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行して複数行をqueryRowsCompletedデリゲートに通知します。HTTPプロトコルで転送されるデータ量に上限がありますので一定のデータ量で制限したいときには第2パラメータmaxRowsに取得するデータ行の最大数を設定します。maxRowsにゼロ設定の場合は全データを取得します。

rollbackTrans

書式
status rollbackTrans();

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
トランザクションをロールバックします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

プロパティ

lastErrorString

データ型
String

解説
本製品のメソッドを呼び出してエラーが発生した場合にその理由が明確な場合はこのプロパティに保持されます。エラー情報は以下となります。

1. ネットワークエラー情報
2. サーバー側エラー情報
3. DBエラーメッセージ

lowerColName

データ型
bool

解説
True設定時にクエリー結果のカラム名を小文字に変換します。

pwd

データ型
NSString*

解説
ユーザー毎認証を使う場合にはこのプロパティにパスワードを設定してください。

rawUrl

データ型
bool

解説
Windows　サーバー版と互換性のために存在するプロパティです。false指定しない場合はurlに/dbLibSerber.asmxを追加指定します。

Row

データ型
Row クラス

解説

Query メソッドの結果を保持します。

Rows

データ型

ArrayList<Row>

解説

Query メソッド(オーバーロードの上限レコード数指定)の結果を保持します。

uid

データ型

NSString*

解説

ユーザー毎認証を使う場合にはこのプロパティにユーザーIDを設定してください。

url

データ型

String

解説

接続先のサーバURLを指定します。（例： http://some.server.jp/） サーバーの構成によりSSL接続やポート番号指定も可能です。

verbose

データ型

bool

解説

サーバーから受信したXMLデータ内容をログに出力します。ログのタグは”dblib”となります。

flutter / dart クラスリファレンス

DbLib クラスに定義されるメソッドはネットーワークI/Oを伴いますので全て async 属性を持ちます。設定必須プロパティは url のみです。

メソッド

beginTrans

書式
Future<status> beginTrans() async;

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
Oracleトランザクションを開始します。サーバーには接続完了状態、トランザクションクローズ状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

commitTrans

書式
Future<status> commitTrans() async;

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
Oracleトランザクションをコミットします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

connect

書式
Future<status> connect() async;

戻り値

status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
プロパティURLで指定されるサーバーに接続します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

disconnect

書式
Future<status> disconnect() async;

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
サーバー接続を遮断します。サーバー側ではOracle接続の遮断、トランザクション中の場合はトランザクションの破棄が実行されます。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

execute

書式
Future<status> execute(String sql) async;

パラメータ

非クエリ系Oracle SQL

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
非クエリ系Oracle SQLを実行します。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

query

書式
Future<QueryResult> query(String sql) async;

戻り値
QueryStaus.status == normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行成功すると先頭1行のデータがQueryStatus.resultに戻されます。後続データはfetchメソッドで同様に取得します。

queryRows

書式
Future<QueryRowsResult> queryRows(String sql, int maxRows) async;

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
SQLで指定されるクエリを発行して複数行をQueryRowsResult.resultに取得します。HTTPプロトコルで転送されるデータ量に上限がありますので一定のデータ量で制限したいときには第2パラメータmaxRowsに取得するデータ行の最大数を設定します。maxRowsにゼロ設定の場合は全データを取得します。

rollbackTrans

書式
Future<status> rollbackTrans() async;

戻り値
status.Normal時は呼び出し成功。それ以外はAppendix-A参照。

解説
Oracleトランザクションをロールバックします。サーバーには接続完了状態、トランザクション状態で呼び出してください。サーバーからの実行結果はrequestCompleted/requestFailedデリゲートに通知されます。

プロパティ

lastException

データ型
Exception

詳細
最後に発生した例外を保持します。

lastHTTPStatusCode

データ型
int

詳細
最後に発生したhttp status codeを保持します。

lastServerError

データ型
String

詳細
最後に発生したサーバー側のエラーや例外を保持します。

networkTimeout

データ型
int

詳細
サーバーとのhttp/https通信タイムアウトをミリ秒単位で指定します。

pwd

データ型
String

解説

ユーザー毎認証を使う場合にはこのプロパティにパスワードを設定してください。

rawUrl

データ型

bool

詳細

Windows　サーバー版と互換性のために存在するプロパティです。false指定しない場合はurlに/dbLibSerber.asmxを追加指定します。

uid

データ型

String

解説

ユーザー毎認証を使う場合にはこのプロパティにユーザーIDを設定してください。

url

データ型

String

詳細

サーバーURIを指定します。プロトコル指定必要でサーバーアプリをルートに設置した場合はサーバーアセンブリ名は省略できます。

verbose

データ型

bool

詳細

true設定でログを詳細に出力します。

Appendix-A エラーコード

iOS / Android メソッド呼び出し時に返るdbLibStatusです。

定義値	値	詳細
Normal	0	正常終了
NetworkFail	1	ネットワーク接続が出来ませんでした。URLプロパティと実際のネットワーク接続状態をご確認ください。
NoURL	2	URLプロパティの指定がありませんでした。
NoUID	3	UIDプロパティの指定がありませんでした。
NoPwd	4	PWDプロパティの指定がありませんでした。
WebServiceFailed	5	Web Serviceがエラーを返しました。
WebServiceException	6	Web Servcieが例外を返しました。lastErrorTextに詳細が保持される場合がありますのでご確認ください。
SqlEmpty	7	SQLパラメータ指定がありません。
StillInRequest	8	他のリクエストが終了していないため、新たなメソッドの呼び出しが出来ません。
ioError	9	ネットワーク通信時にエラーとなりました。安定したネットワークに接続して、再度実行してください。
protocolError	10	Soapサービスのプロトコルに沿っていないデータを受信しました。接続先のサーバーが当製品のサーバーでは無い可能性があります。
resultParseError	11	サーバーから戻されたレスポンスを解析出来ませんでした。
NoSql	12	メソッドパラメータに必要なSQL指定がありませんでした。
unknownSoapService	13	当システムにて認識できないSoapService名がサーバーから戻されました。

flutter メソッド呼び出し時に返るStatusEnumです。

定義値	詳細
normal	正常終了
notConnected	connectメソッドが呼び出されていません。
connectFail	サーバーに接続できませんでした。lastHttpStatusCodeプロパティなどを確認してください。
endOfData	Web Serviceがエラーを返しました。
methodSequeneError	メソッドの呼び出し順序が違います。
parseXMLError	サーバーからのデータをパーズできませんでした。
sqlEmpty	SQLパラメータ指定がありません。
networkFail	ネットワークが正常に動作していません。
serverError	サーバーから戻されたレスポンスを解析出来ませんでした。
httpError	http Status 200以外がサーバーから返されました。
fatal	致命的なエラー。例外発生していますのでlastExceptionを参照するかverbose=true設定でログを参照してください。

Appendix-B swift サンプル

```
import UIKit

class ViewController: UIViewController,UITableViewDelegate,UITableViewDataSource,
dbLibDelegate {
    @IBOutlet weak var _tableView: UITableView!
    private var _rows:NSMutableArray!
    private var _db:dbLib!

    //
    //  MARK: dblib code
    //
    func startConnect() {

        _db = dbLib.init()
        _db.delegate = self
        _db.url = "http://192.168.179.8/"
        _db.verbose = true
        let rc = _db.connect()
        if rc != Normal {
            print("connect failed \(rc)")
            return
        }

    }

    func requestCompleted(_ methodName: String!) {
        print("request completed:" + methodName)

        if methodName == "Login" {
            let qrc = _db.queryRows("select EMPNO,ENAME from emp order by empno", 100)
            print("query result= \(qrc)")
        }

    }

    func requestFailed(_ methdName: String!, _ error: Error!) {
        print("request failed \(String(describing: error))")
    }

    func queryRowsCompleted(_ rows: NSMutableArray!) {
        DispatchQueue.main.async {
            self._rows = rows
            self._tableView.reloadData()
            self._db.disconnect()
        }
    }
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        _tableView.register(UITableViewCell.self, forCellReuseIdentifier: "Cell")
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.startConnect()
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if _rows == nil {
            return 0
        }
        return _rows.count
    }

    unc tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
        let cell:UITableViewCell = tableView.dequeueReusableCell(withIdentifier: "Cell")!

        let row:NSDictionary = _rows[indexPath.row] as! NSDictionary
        let empno = row.value(forKey: "EMPNO") as! String
        let ename = row.value(forKey: "ENAME") as! String
        cell.textLabel!.text = empno + " " + ename
        return cell
    }

     func tableView(_ tableView: UITableView, willDisplay cell: UITableViewCell, forRowAt indexPath:
IndexPath) {

        tableView.separatorInset = UIEdgeInsets.zero
        tableView.layoutMargins = UIEdgeInsets.zero
        cell.layoutMargins = UIEdgeInsets.zero

    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)
    }
}
```

Appendix-C iOS Objective-Cサンプルコード

```
-(void) startConnect {

	[UIApplication sharedApplication].networkActivityIndicatorVisible = YES;

	_db = [[dbLib alloc] init];
	_db.url = @"http://some.domain.jp";
	_db.delegate = self;

	dbLibStatus rc = [_db connect];

	if(rc != Normal) {

		NSLog(@"connect failed %d",rc);
		return;
	}

}

-(void) startQueryRows {

	bool rc = [_db queryRows:@"select empno,ename from emp order by empno" :0];

	if(rc != Normal){
		NSLog(@"query rows request failed");
	}
}

-(void) queryRowsCompleted:(NSMutableArray *)rows {

	_rows = [rows retain];

	[_table reloadData];
	[_db disconnect];
	[UIApplication sharedApplication].networkActivityIndicatorVisible = NO;

}
```

Appendix-D サンプル db_proxy.ini

```
[ssl]
certFile = "/etc/letsencrypt/live/domain.co.jp/fullchain.pem"
keyFile = "/etc/letsencrypt/live/domain.co.jp/privkey.pem"

[server]
port = 443
db = "oracle"
constr = "oracle://scott:tiger@localhost:1521/pdb01"
logfile = "/var/log/db_proxy.log"
lower_case_column = false
max_rows = 0
verbose = true
interface=“0.0.0.0”
```

Appendix-E Android サンプルコード

```
try {
	dbLib.status st;
	dbLib db = new dbLib();
	db.setURL("http://192.168.0.5");

	st = db.connect();
	if(st == status.Normal){
		st = db.query("select empno,ename from emp",100);
		if(st == status.Normal){
			//
			ArrayList<dbLib.Row> rows = db.getRows();
			//
			for(int i=0; i<rows.size(); i++){
				//
				Row row = rows.get(i);
				Column col = row.columns.get(0);
				String empno = col.value;
				col = row.columns.get(1);
				String ename = col.value;
				Log.v("test","result=" + empno + "," + ename);

			}
		}
	}
	db.disconnect();
}
catch(Exception ex){
	Log.v("test",ex.getMessage());
	ex.printStackTrace();
}
```

Appendix-F flutter/dart サンプルコード

```
import 'package:dblib/dblib.dart';

Future<void> sample() async {

  final dblib = DbLib();
  dblib.url = "http://192.168.128.10";
  dblib.verbose = true;
  var st = await dblib.connect();
  var res = await dblib.query("select * from EMP");
  while (res.status==StatusEnum.normal){
    _dumpRow(res.result);
    res = await dblib.fetch();
  }
  st = await dblib.disconnect();
}

void _dumpRow(res){
  print('---------');
  res.forEach((key,item){
    print('$key = $item');
  });
}
```

DB connector for iOS/Android 調査依頼

日付	
会社名	
登録ユーザー名	
製品シリアル番号	
製品バージョン	
電話番号	
ファックス番号	
メールアドレス	
使用パソコン機種	
利用端末名 (iPhone/iPad/Android)	
iOSまたはAndroid バージョン	
サーバー種類とバージョン	
お問合わせ内容、問題記述など、具体的に再現可能なようにご記入ください。	
添付資料	

DB connector for iOS/Android
Linux version 3.5

第1版

2023年5月22日