PUBLIC (公開)
SQL Anywhere サーバ
ドキュメントバージョン: 17.01.0 – 2023-09-12

SQL Anywhere - Node.js API リファレンス

THE BEST RUN SAP

目次

1 Node.js アプリケーションプログラミング

Node.js API は、SQL Anywhere データベースへの接続、SQL クエリの発行、結果セットの取得に使用できます。

Node.js ドライバを使用すると、Joyent の Node.js ソフトウェアプラットフォームで JavaScript を使用してデータベースに接続し、クエリを実行できます。様々なバージョンの Node.js. のドライバが使用可能です。

API インタフェースは SAP HANA Node.js Client にきわめて似ており、文の接続、接続解除、実行、準備が可能です。

ドライバは、NPM (Node Packaged Modules) Web サイト (https://npmjs.org/) からインストールできます。

また、https://github.com/sqlanywhere/node-sqlanywhere からダウンロードすることもできます。

このセクションの内容:

Connection クラス [3 ページ]
データベースへの接続を表します。

Statement クラス [12 ページ]
準備された文を表します。

1.1 Connection クラス

データベースへの接続を表します。

構文

```
class Connection
```

メンバー

Connection のすべてのメンバー (継承されたメンバーも含みます) を次に示します。

メソッド

タイプ	メソッド	説明
	commit(Function) [6 ページ]	接続に対してコミットを実行します。
	connect(String, Function) [7 ページ]	既存の接続を使用して接続します。
	disconnect(Function) [8 ページ]	現在の接続を閉じます。

タイプ	メソッド	説明
結果	exec(String, Array, Function) [9 ページ]	指定された SQL 文を実行します。
文	prepare(String, Function) [10 ページ]	指定された SQL 文を準備します。
	rollback(Function) [11 ページ]	接続に対してロールバックを実行します。

備考

次の例では、同期呼び出しを使用してデータベースサーバへの新規接続を作成し、サーバに対して SQL クエリを発行し、結果セットを表示し、サーバとの接続を切断します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( { ServerName: 'demo17', UserID: 'DBA', Password: 'sql' } )
console.log('Connected');
result = client.exec("SELECT * FROM Customers");
console.log( result );
client.disconnect()
console.log('Disconnected');
```

次の例では、コールバックを使用したほぼ同じ非同期呼び出しを実行します。エラーチェックが含まれています。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql",
   function( err )
   {
       if( err )
       {
           console.error( "Connect error: ", err );
       }
       else
       {
           console.log( "Connected" )
           client.exec( "SELECT * FROM Customers",
               function( err, rows )
               {
                   if( err )
                   {
                       console.error( "Error: ", err );
                   }
                   else
                   {
                       console.log(rows)
                   }
               }
           );
           client.disconnect(
               function( err )
               {
                   if( err )
                   {
                       console.error( "Disconnect error: ", err );
                   }
                   else
                   {
                       console.log( "Disconnected" )
```

```
                    }
                }
            );
        }
    }
);
```

次の例では、コールバックを使用しますが、関数はインライン化されておらず、分かりやすいコードになっています。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql", async_connect );
function async_connect( err )
{
   if( err )
   {
       console.error( "Connect error: ", err );
   }
   else
   {
       console.log( "Connected" )
       client.exec( "SELECT * FROM Customers", async_results );
       client.disconnect( async_disco );
   }
}
function async_results( err, rows )
{
   if( err )
   {
       console.error( "Error: ", err );
   }
   else
   {
       console.log(rows)
   }
}
function async_disco( err )
{
   if( err )
   {
       console.error( "Disconnect error: ", err );
   }
   else
   {
       console.log( "Disconnected" )
   }
}
```

接続パラメータを createConnection 関数に渡すことができます。また、それらのパラメータを connect() 関数呼び出しと組み合わせると、接続で使用する接続文字列を取得できます。接続パラメータのハッシュまたは接続文字列フラグメントをどの呼び出しでも使用できます。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection( { uid: 'dba'; pwd: 'sql' } );
client.connect( 'server=MyServer;host=localhost' );
// the connection string that will be used is
// "uid=dba;pwd=sql;server=MyServer;host=localhost"
```

このセクションの内容:

connect(String, Function) メソッド [7 ページ]
　既存の接続を使用して接続します。

disconnect(Function) メソッド [8 ページ]
　現在の接続を閉じます。

exec(String, Array, Function) メソッド [9 ページ]
　指定された SQL 文を実行します。

prepare(String, Function) メソッド [10 ページ]
　指定された SQL 文を準備します。

rollback(Function) メソッド [11 ページ]
　接続に対してロールバックを実行します。

1.1.1 commit(Function) メソッド

接続に対してコミットを実行します。

構文

```
connection.commit (callback)
```

パラメータ

タイプ	名前	説明
関数	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは、接続に対してコミットを実行します。デフォルトでは、データベースサーバからの切断時に挿入、更新、削除はコミットされません。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err ) {
};
```

次の同期の例は、commit メソッドの使用方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare(
   "INSERT INTO Departments "
   + "( DepartmentID, DepartmentName, DepartmentHeadID )"
```

```
   + "VALUES (?,?,?)" );
result = stmt.exec( [600, 'Eastern Sales', 902] );
result += stmt.exec( [700, 'Western Sales', 902] );
stmt.drop();
console.log( "Number of rows added: " + result );
result = client.exec( "SELECT * FROM Departments" );
console.log( result );
client.commit();
client.disconnect();
```

1.1.2 connect(String, Function) メソッド

既存の接続を使用して接続します。

{·} 構文

```
connection.connect (conn_string, callback)
```

パラメータ

タイプ	名前	説明
String	conn_string	有効な接続文字列 (型: 文字列)。
Function	callback	オプションコールバック関数 (型: 関数)。

備考

新しい接続を作成します。

このメソッドは、パラメータとして渡される接続文字列または接続パラメータのハッシュを使用して新しい接続を作成します。プログラムの終了前に、disconnect メソッドを使用してリソースを解放し、接続を切断する必要があります。

CharSet (CS) 接続パラメータ CS=UTF-8 は、すべての文字列がそのエンコーディングで送信する必要があるため、ドライバによって接続文字列の最後に常に付加されます。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err )
{
};
```

次の同期の例は、connect メソッドの使用方法を示します。CHARSET=UTF-8 接続パラメータは自動的に追加されるため、指定する必要はありません。

```
var sqlanywhere = require( 'sqlanywhere' );
```

```
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql;CHARSET=UTF-8" );
```

関連情報

disconnect(Function) メソッド [8 ページ]

1.1.3 disconnect(Function) メソッド

現在の接続を閉じます。

構文

```
connection.disconnect (callback)
```

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは現在の接続を閉じます。リソースを解放するためにプログラムを終了する前に呼び出す必要があります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err )
{
};
```

次の同期の例は、disconnect メソッドの使用方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
client.disconnect()
```

関連情報

connect(String, Function) メソッド [7 ページ]

1.1.4 exec(String, Array, Function) メソッド

指定された SQL 文を実行します。

{·} 構文

```
connection.exec (sql, params, callback)
```

パラメータ

タイプ	名前	説明
String	sql	実行される SQL 文 (型: 文字列)。
Array	params	バインドパラメータのオプション配列 (型: 配列)
関数	callback	オプションコールバック関数 (型: 関数)。

戻り値

コールバックが指定されていない場合、結果が返されます。

備考

このメソッドは、SQL 文および実行するバインドパラメータのオプション配列を取ります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err, result )
{
};
```

結果セットを生成するクエリの場合、結果セットオブジェクトがコールバックの 2 番目のパラメータとして返されます。insert、update、delete 文の場合、影響を受けるローの数がコールバックの 2 番目のパラメータとして返されます。他の文の場合、結果は未定義です。

次の同期の例は、exec メソッドの使用方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
result = client.exec("SELECT * FROM Customers");
console.log( result );
client.disconnect()
```

次の同期の例は、バインドパラメータを指定する方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
result = client.exec(
   "SELECT * FROM Customers WHERE ID >=? AND ID <?",
   [300, 400] );
console.log( result );
client.disconnect()
```

1.1.5 prepare(String, Function) メソッド

指定された SQL 文を準備します。

{·} 構文

```
connection.prepare (sql, callback)
```

パラメータ

タイプ	名前	説明
String	sql	実行される SQL 文 (型: 関数)。
Function	callback	オプションコールバック関数 (型: 関数)。

戻り値

コールバックが指定されていない場合、Statement オブジェクトが返されます。

備考

このメソッドは、SQL 文を準備し、成功した場合に Statement オブジェクトを返します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err, Statement )
{
};
```

次の同期の例は、prepare メソッドの使用方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
console.log( result );
client.disconnect();
```

1.1.6 rollback(Function) メソッド

接続に対してロールバックを実行します。

{·} 構文

```
connection.rollback (callback)
```

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは、接続に対してロールバックを実行します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err ) {
};
```

次の同期の例は、rollback メソッドの使用方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare(
```

```
   "INSERT INTO Departments "
   + "( DepartmentID, DepartmentName, DepartmentHeadID )"
   + "VALUES (?,?,?)" );
result = stmt.exec( [600, 'Eastern Sales', 902] );
result += stmt.exec( [700, 'Western Sales', 902] );
stmt.drop();
console.log( "Number of rows added: " + result );
result = client.exec( "SELECT * FROM Departments" );
console.log( result );
client.rollback();
client.disconnect();
```

1.2 Statement クラス

準備された文を表します。

構文

```
class Statement
```

メンバー

Statement のすべてのメンバー (継承されたメンバーも含みます) を次に示します。

メソッド

タイプ	メソッド	説明
	drop(Function) [13 ページ]	文を削除します。
result	exec(Array, Function) [13 ページ]	準備された SQL 文を実行します。

このセクションの内容:

drop(Function) メソッド [13 ページ]
文を削除します。

exec(Array, Function) メソッド [13 ページ]
準備された SQL 文を実行します。

関連情報

prepare(String, Function) メソッド [10 ページ]

1.2.1 drop(Function) メソッド

文を削除します。

構文

```
statement.drop (callback)
```

パラメータ

タイプ	名前	説明
関数	callback	オプションコールバック関数。

備考

このメソッドは準備文を削除し、リソースを解放します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err )
{
};
```

次の同期の例は、準備された文で drop メソッドを使用する方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
stmt.drop();
console.log( result );
client.disconnect();
```

1.2.2 exec(Array, Function) メソッド

準備された SQL 文を実行します。

構文

```
statement.exec (params, callback)
```

パラメータ

タイプ	名前	説明
Array	params	バインドパラメータのオプション配列。
Function	callback	オプションコールバック関数。

戻り値

コールバックが指定されていない場合、結果が返されます。

備考

このメソッドは、実行するバインドパラメータの配列を取ります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err, result )
{
};
```

結果セットを生成するクエリの場合、結果セットオブジェクトがコールバックの 2 番目のパラメータとして返されます。insert、update、delete 文の場合、影響を受けるローの数がコールバックの 2 番目のパラメータとして返されます。他の文の場合、結果は未定義です。

次の同期の例は、準備された文で exec メソッドを使用する方法を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
stmt.drop();
console.log( result );
client.disconnect();
```

重要免責事項および法的情報

ハイパーリンク

リンクの一部は、アイコンやマウスオーバーテキストで分類されています。これらのリンクから、追加の情報を得ることができます。
アイコンについて。

- このアイコンが付いたリンク: SAP がホストしているものではない Web サイトに移動します。これらのリンクを使用することで、お客様は (お客様と SAP との契約書に別段の明示的な記載がない限り) 以下のことに同意することになります。
  - リンク先のサイトのコンテンツが SAP のドキュメンテーションではないこと。お客様は、この情報に基づいて SAP に対する製品クレームを推断することはできません。
  - SAP が、リンク先のサイトのコンテンツについて同意することも反対することもなく、また SAP がその利用可能性や正確性について保証しないこと。SAP は、かかるコンテンツの使用により発生した損害が、SAP の重大な過失又は意図的な違法行為が原因で発生したものでない限り、その損害に対して一切責任を負いません。
-  このアイコンが付いたリンク: 当該の特定の SAP 製品又はサービスのドキュメンテーションから離れ、SAP がホストしている Web サイトに移動します。これらのリンクを使用することで、お客様は (お客様と SAP との契約書に別段の明示的な記載がない限り)、この情報に基づいて SAP に対する製品クレームを推断することはできないことに同意します。

外部プラットフォームでホストされているビデオ

一部のビデオは、サードパーティのビデオホスティングプラットフォームに置かれている場合があります。SAP では、これらのプラットフォームに保存されているビデオが将来にわたって利用できると保証することはできません。また、これらのプラットフォームにホストされている、いかなる広告またはその他のコンテンツ (関連ビデオまたは同じサイトでホストされている別のビデオに移動する場合など) については、SAP の管理外であり責任を負いません。

ベータおよびその他の試験的機能

試験的機能は、SAP が将来のリリースを保証する正式に提供される機能の範囲外です。これは、試験的機能は、SAP により通知なく理由の如何を問わず随時変更される場合があることを意味します。試験的機能は、本稼働使用のためのものではありません。お客様は、試験的機能を実際の運用環境で、又は十分なバックアップがとられていないデータとともに、デモンストレーション、テスト、試験、評価その他の方法で使用してはなりません。
試験的機能の目的は、早期にフィードバックを得ることで、それに応じて顧客の皆様やパートナーが将来の製品に影響を与えることを可能にすることです。SAP コミュニティなどにおいてフィードバックを提供することで、お客様は、投稿物や二次的著作物の知的財産権が SAP の独占的所有物であり続けることを承認することになります。

コード例

ソフトウェアのコーディングやコードスニペットはすべて、例です。それらは、本稼働使用のためのものではありません。コード例は、構文や表現規則を分かりやすく説明し視覚化することのみを目的としています。SAP は、コード例の正確性や完全性について保証しません。SAP は、コード例の使用により発生した過誤や損害が、SAP の重大な過失又は意図的な違法行為が原因で発生したものでない限り、損害に対して一切責任を負いません。

偏見のない表現

SAP は、ダイバーシティ & インクルージョンの文化を支持しています。SAP の文書では、可能な限り、文化、民族性、ジェンダー、および障がいの有無を問わず、すべての人々に対する偏見を伴わない表現を採用します。

www.sap.com/contactsap

THE BEST RUN SAP