無料電子ブック

学習

Dapper.NET

Free unaffiliated eBook created from
Stack Overflow contributors.

#dapper

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dapper-net

It is an unofficial and free Dapper.NET ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Dapper.NET.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Dapper.NETをいめる

Dapperとは

Dapperは、 `IDbConnection`をする.NetのマイクロORMであり、クエリの、、およびのみりをします。

どうすればできますか

- github https : //github.com/StackExchange/dapper-dot-net
- NuGet https : //www.nuget.org/packages/Dapper

なタスク

- なクエリ
- コマンドの

バ―ジョン

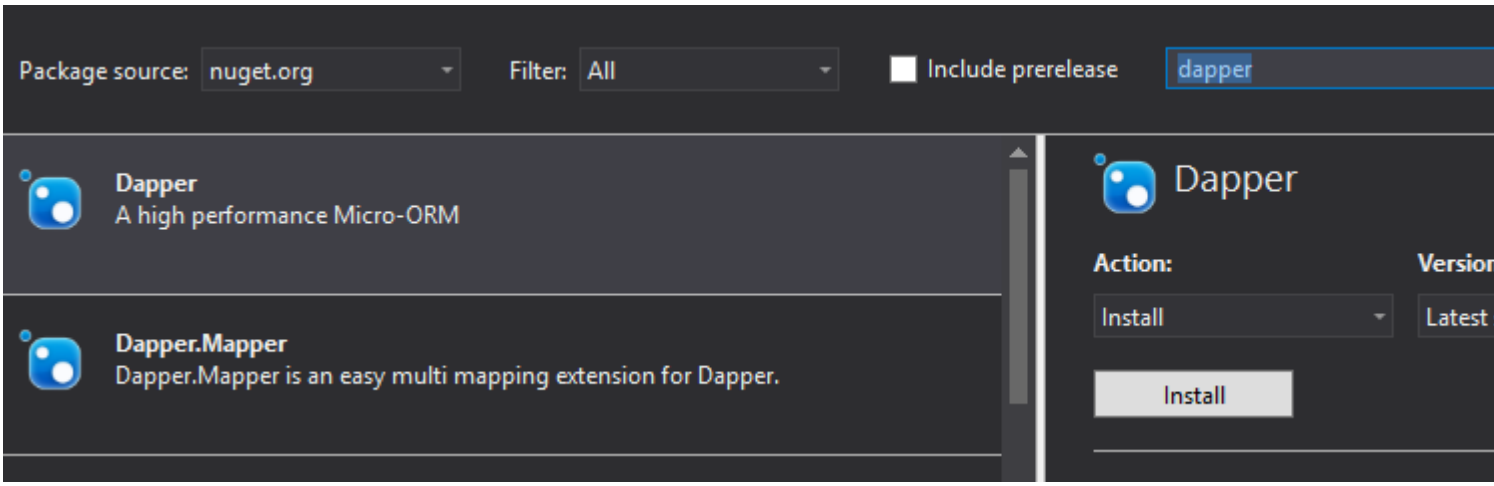
バ―ジョン	ノート	
1.50.0	core-clr / asp.net 5.0がRTMにしてビルドされています	2016629
1.42.0		2015-05-06
1.40.0		2015-04-03
1.30.0		2014-08-14
1.20.0		2014-05-08
1.10.0		2012-06-27
1.0.0		2011-04-14

Examples

NugetからDapperをインストールする

Visual Studio GUIでするか、

ツール> NuGet Package Manager>ソリューションのパッケージ...Visual Studio 2015



または、このコマンドをNuget Power Shellインスタンスでして、のをインストールします

```
Install-Package Dapper
```

またはのバージョン

```
Install-Package Dapper -Version 1.42.0
```

CでDapperをう

```
using System.Data;
using System.Linq;
using Dapper;

class Program
{
    static void Main()
    {
        using (IDbConnection db = new
SqlConnection("Server=myServer;Trusted_Connection=true"))
        {
            db.Open();
            var result = db.Query<string>("SELECT 'Hello World'").Single();
            Console.WriteLine(result);
        }
    }
}
```

`Using`ブロックにをラップすると、がじられます。

LINQPadでDapperをう

LINQPadは、データベースクエリのテストにで、 NuGetのもまれています。 LINQPadでDapperをするには、 **F4**キーをしてクエリのプロパティをき、[**Add NuGet]**をします。 **dapper dot net**をし、 **Add To Query**をします。また、LINQPadクエリにメソッドをめるには、[の]をクリックして[Dapper] をすることもできます。

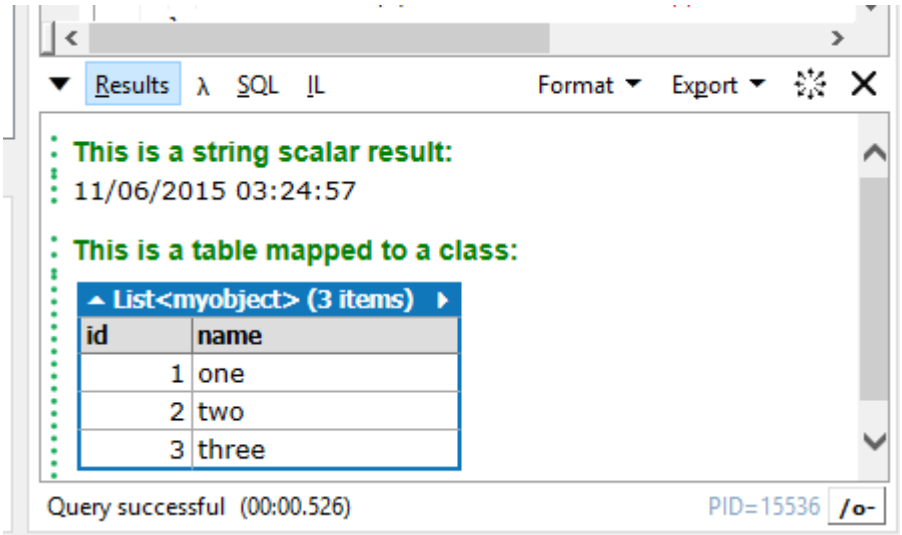
Dapperをにすると、ドロップダウンをCプログラムにし、クエリをCクラスにマップし、.Dump メソッドをしてをできます。

```
void Main()
{
    using (IDbConnection db = new SqlConnection("Server=myServer;Trusted_Connection=true")){
        db.Open();
        var scalar = db.Query<string>("SELECT GETDATE()").SingleOrDefault();
        scalar.Dump("This is a string scalar result:");

        var results = db.Query<myobject>(@"
        SELECT * FROM (
        VALUES (1,'one'),
            (2,'two'),
            (3,'three')
        ) AS mytable(id,name)");
        results.Dump("This is a table mapped to a class:");
    }
}

// Define other methods and classes here
class myobject {
    public int id { get; set; }
    public string name { get; set; }
}
```

プログラムをすると、はのようになります。



オンラインでDapper.NETをいめるをむ https://riptutorial.com/ja/dapper/topic/2/dapper-netをいめる

2: DbGeographyとDbGeometryの

Examples

がです

1. なMicrosoft.SqlServer.Typesアセンブリをインストールし Microsoft.SqlServer.Types 。 デフォルトではインストールされていないため、Microsoftからは「Microsoft®SQLServer®2012のMicrosoft®System CLR Types」としてできます。x86とx64のインストーラーが々であることにしてください。

2. Dapper.EntityFramework またはそれにするなをインストールします。これは、IDEの「Manage NuGet Packages ...」UI、またはパッケージマネージャコンソールでうことができます。

```
install-package Dapper.EntityFramework
```

3. なアセンブリバインディングのリダイレクトをします。これはMicrosoftがv11のアセンブリをしているためですが、Entity Frameworkはv10をします。 <configuration>のの app.config または web.config をでき <configuration> 。

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="Microsoft.SqlServer.Types"
          publicKeyToken="89845dcd8080cc91" />
      <bindingRedirect oldVersion="10.0.0.0" newVersion="11.0.0.0" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

4. なしいタイプのハンドラについて、データベースをするにスタートアップのどこかに "dapper"としてください。

```
Dapper.EntityFramework.Handlers.Register();
```

ジオメトリとをう

ハンドラがされると、すべてがにするはずで、これらのをパラメータまたはりとしてできるはずです。

```
string redmond = "POINT (122.1215 47.6740)";
DbGeography point = DbGeography.PointFromText(redmond,
    DbGeography.DefaultCoordinateSystemId);
DbGeography orig = point.Buffer(20); // create a circle around a point
```

```
var fromDb = connection.QuerySingle<DbGeography>(
  "declare @geos table(geo geography); insert @geos(geo) values(@val); select * from @geos",
  new { val = orig });

Console.WriteLine($"Original area: {orig.Area}");
Console.WriteLine($"From DB area: {fromDb.Area}");
```

オンラインでDbGeographyとDbGeometryのをむ
https://riptutorial.com/ja/dapper/topic/3984/dbgeographyとdbgeometryの

3: コマンドの

Examples

をさないコマンドをする

```
IDBConnection db = /* ... */
var id = /* ... */

db.Execute(@"update dbo.Dogs set Name = 'Beowoof' where Id = @id",
   new { id });
```

ストアドプロシ―ジャ

ない

Dapperはストアドプロシ―ジャをにサポ―トしています

```
var user = conn.Query<User>("spGetUser", new { Id = 1 },
                            commandType: CommandType.StoredProcedure)
           .SingleOrDefault();
```

、、およびリタ―ンのパラメ―タ

あなたはかもっとしいとうなら、あなたはできます

```
var p = new DynamicParameters();
p.Add("@a", 11);
p.Add("@b",
      dbType: DbType.Int32,
      direction: ParameterDirection.Output);
p.Add("@c",
      dbType: DbType.Int32,
      direction: ParameterDirection.ReturnValue);

conn.Execute("spMagicProc", p,
             commandType: CommandType.StoredProcedure);

var b = p.Get<int>("@b");
var c = p.Get<int>("@c");
```

テ―ブルのパラメ―タ

テ―ブルきパラメ―タをけれるストアドプロシ―ジャがあるは、SQL Serverのテ―ブルタイプとじをつDataTableをすがあります。ここでは、テ―ブルタイプとそれをするプロシ―ジャのをします。

```
CREATE TYPE [dbo].[myUDTT] AS TABLE([i1] [int] NOT NULL);
GO
CREATE PROCEDURE myProc(@data dbo.myUDTT readonly) AS
SELECT i1 FROM @data;
GO
/*
-- optionally grant permissions as needed, depending on the user you execute this with.
-- Especially the GRANT EXECUTE ON TYPE is often overlooked and can cause problems if omitted.
GRANT EXECUTE ON TYPE::[dbo].[myUDTT] TO [user];
GRANT EXECUTE ON dbo.myProc TO [user];
GO
*/
```

cからそのプロシ―ジャをびすには、のがです。

```
// Build a DataTable with one int column
DataTable data = new DataTable();
data.Columns.Add("i1", typeof(int));
// Add two rows
data.Rows.Add(1);
data.Rows.Add(2);

var q = conn.Query("myProc", new {data}, commandType: CommandType.StoredProcedure);
```

オンラインでコマンドのをむ https://riptutorial.com/ja/dapper/topic/5/コマンドの

4: トランザクション

- conn.Executesql、transactiontran; //でパラメ―タをする
- conn.Executesql、parameters、tran;
- conn.Querysql、トランザクションtran;
- conn.Querysql、parameters、tran;
- conn.ExecuteAsyncsql、transactiontran;をちます。 // Async
- conn.ExecuteAsyncsql、parameters、tran;をちます。
- conn.QueryAsyncsql、transactiontran;をちます。
- conn.QueryAsyncsql、parameters、tran;をちます。

Examples

トランザクションの

このではSqlConnectionをしていますが、IDbConnectionはすべてサポ―トされています。

また、すべてのIDbTransactionは、IDbConnectionからサポ―トされています。

```
public void UpdateWidgetQuantity(int widgetId, int quantity)
{
    using(var conn = new SqlConnection("{connection string}")) {
        conn.Open();

        // create the transaction
        // You could use `var` instead of `SqlTransaction`
        using(SqlTransaction tran = conn.BeginTransaction()) {
            try
            {
                var sql = "update Widget set Quantity = @quantity where WidgetId = @id";
                var parameters = new { id = widgetId, quantity };

                // pass the transaction along to the Query, Execute, or the related Async
methods.
                conn.Execute(sql, parameters, tran);

                // if it was successful, commit the transaction
                tran.Commit();
            }
            catch(Exception ex)
            {
                // roll the transaction back
                tran.Rollback();

                // handle the error however you need to.
                throw;
            }
        }
    }
}
```

インサートのスピードアップ

トランザクションにのグループをラップすると、このStackOverflowの/にってそれらをします。

このをすることも、コピーをしてのするをすることもできます。

```
// Widget has WidgetId, Name, and Quantity properties
public void InsertWidgets(IEnumerable<Widget> widgets)
{
    using(var conn = new SqlConnection("{connection string}")) {
        conn.Open();

        using(var tran = conn.BeginTransaction()) {
            try
            {
                var sql = "insert Widget (WidgetId,Name,Quantity) Values(@WidgetId, @Name,
@Quantity)";
                conn.Execute(sql, widgets, tran);
                tran.Commit();
            }
            catch(Exception ex)
            {
                tran.Rollback();
                // handle the error however you need to.
                throw;
            }
        }
    }
}
```

オンラインでトランザクションをむ https://riptutorial.com/ja/dapper/topic/6601/トランザクション

5: ヌルの

Examples

ヌルDBNull

ADO.NETでは、 `null`しくすることは、にのとなります。まかなでなのは、 あなたがするがないということです。それはにすべてそれをいます。

- `null`パラメータは、 `DBNull.Value`としてしくされ`DBNull.Value`
- あるみり`null`としてされている`null`にそのタイプにづくデフォルトをすがのタイプへのマッピングの、または

それだけでします

```
string name = null;
int id = 123;
connection.Execute("update Customer set Name=@name where Id=@id",
    new {id, name});
```

オンラインでヌルのをむ https://riptutorial.com/ja/dapper/topic/13/ヌルの

6: パラメータリファレンス

パラメータ

パラメータ	
`this cnn`	となるデータベース - `this`はメソッドをします。がいているはありません。いていなければにします。
`<T>` / `Type`	オプションすオブジェクトの。ジェネリック/`Type` APIがされて`dynamic`は、クエリごとにされるごとにがけられたプロパティをシミュレートする`dynamic`オブジェクトがごとにされますこの`dynamic`オブジェクトは`IDicionary<string,object>`もし`IDicionary<string,object>`。
`sql`	するSQL
`param`	オプションめるパラメータ。
`transaction`	オプションコマンドにけるデータベーストランザクション
`buffered`	オプションリストにデータをにするかどうかデフォルト、いている`IEnumerable`をライブリーダーでするか
`commandTimeout`	オプションコマンドでするタイムアウト。されていない、`SqlMapper.Settings.CommandTimeout`がされているとみなされます。
`commandType`	されるコマンドのタイプ。デフォルトは`CommandText`

パラメータをするためのは、RDBMSによってなります。のすべてのでは、SQL Server、つまり`@foo`しています。しかし、`?foo`と`:foo`もうまくするはずです。

Examples

なパラメータされたSQL

Dapperをすると、にパラメータされたSQLをしてベストプラクティスをにできます。

パラメータはなので、まかにそれをしくすることができます。あなたはのRDBMSのためののであなたのパラメータをするは`@foo`、 `?foo`または`:foo`とばれるメンバつオブジェクトDapperのえる*`foo`*。これをうもなは、です

```
int id = 123;
string name = "abc";
connection.Execute("insert [KeyLookup](Id, Name) values(@id, @name)",
    new { id, name });
```

です。 Dapperはなパラメータをし、すべてがするはずです。

オブジェクトモデルの

のオブジェクトモデルをパラメータとしてすることもできます。

```
KeyLookup lookup = ... // some existing instance
connection.Execute("insert [KeyLookup](Id, Name) values(@Id, @Name)", lookup);
```

Dapperはcommand-textをってオブジェクトのどのメンバーをするかをします。 `Description`、`IsActive`、 `CreationDate`ようななものはしません。これはらかにしたコマンドにはしません。えば、あなたのコマンドにのものがまれているとします。

```
// TODO - removed for now; include the @Description in the insert
```

はなるコメントであることをしようとはしません。

ストアドプロシージャ

ストアドプロシージャのパラメータはくじになりますが、dapperはをむべきか/まないべきかをできません。なものはすべてパラメータとしてわれます。そのため、はがされます。

```
connection.Execute("KeyLookupInsert", new { id, name },
    commandType: CommandType.StoredProcedure);
```

バリューインライン

によっては、やのでパラメ―タのが、パラメ―タとしてうためののコストよりもされることがあります。たとえば、ペ―ジサイズがによってされているなどです。または、ステ―タスが`enum`としています。してください

```
var orders = connection.Query<Order>(@"
select top (@count) * -- these brackets are an oddity of SQL Server
from Orders
where CustomerId = @customerId
and Status = @open", new { customerId, count = PageSize, open = OrderStatus.Open });
```

ののパラメ―タは`customerId`。の2つはにはされないパラメ―タです。これらをとしてすると、RDBMSはしばしばよりいをすることができます。 Dapperには、 `@name {=name}`わりに - `{=name}`というながあります。これはにのみされます。 これにより、SQLインジェクションからのがにえられます。はのとおりです。

```
var orders = connection.Query<Order>(@"
select top {=count} *
from Orders
where CustomerId = @customerId
and Status = {=open}", new { customerId, count = PageSize, open = OrderStatus.Open });
```

Dapperは、SQLをするにをリテラルにきえるので、RDBMSはにのようにえます。

```
select top 10 *
from Orders
where CustomerId = @customerId
and Status = 3
```

これは、RDBMSシステムがよりいをうだけでなく、のパラメ―タがするクエリプランをくときににです。たとえば、がパラメ―タにする、そのにのをつフィルタされたはできません。これは、 のに、されたの1つかられたパラメ―タ―があるがあるためです。

リテラルでは、クエリオプティマイザは、のクエリでができないことがわかっているので、フィルタされたインデックスをできます。

リスト

デ―タベ―スクエリのなシナリオは`IN (...)`です。ここでのリストはにされます。ほとんどのRDBMSにはこのためのれたメタファ―がありません。これにはのクロス*RDBMS*ソリュ―ションはありません。わりに、dapperはやかなコマンドをします。なのは、 `IEnumerable`されるパラメ―タです。 `@foo`をむコマンドは、 `(@foo0,@foo1,@foo2,@foo3)` 4つのアイテムのシ―ケンスにしてされています。これのもなは`IN`です。

```
int[] orderIds = ...
var orders = connection.Query<Order>(@"
select *
from Orders
where Id in @orderIds", new { orderIds });
```

これはにされ、フェッチのなSQLがされます。

```
select *
from Orders
where Id in (@orderIds0, @orderIds1, @orderIds2, @orderIds3)
```

`@orderIds0`などのパラメータが`@orderIds0`からされたとしてされます。このがってされていないことをするために、なSQLではないというはなものです。このは、SQL Serverの`OPTIMIZE FOR UNKNOWN`クエリヒントでもしくします。あなたがする

```
option (optimize for
    (@orderIds unknown))
```

これをしくします

```
option (optimize for
    (@orderIds0 unknown, @orderIds1 unknown, @orderIds2 unknown, @orderIds3 unknown))
```

のセットにするの

々、あなたはじことをもやりたいとっています。 ものパラメータはのまたはドメインモデルインスタンスがに`IEnumerable`シーケンスとしてされている、Dapperはこれを`Execute`メソッドでサポートします。えば

```
Order[] orders = ...
// update the totals
connection.Execute("update Orders set Total=@Total where Id=@Id", orders);
```

ここで、dapperはにたちのデータをにループしています。

```
Order[] orders = ...
// update the totals
foreach(Order order in orders) {
    connection.Execute("update Orders set Total=@Total where Id=@Id", order);
}
```

このは、すべての「のアクティブなセット」ににされたで`async` APIとみわせるとにいものになります。このでは、dapperはにをパイプラインするので、ごとのコストをっていません。これにはややないがですが、

```
await connection.ExecuteAsync(
    new CommandDefinition(
        "update Orders set Total=@Total where Id=@Id",
         orders, flags: CommandFlags.Pipelined))
```

ただし、テーブルのパラメータをべることもできます。

パラメータきパラメータをサポートしないプロバイダ

のADO.NETプロバイダにOleDBは、 きパラメータをサポートしていません。パラメータはによってのみされ、 `?`プレースホルダ。 Dapperはどのメンバをこれらのためにうのかからないので、dapperはのをします`?foo?`;これは、のSQLの`@foo`または`:foo`とじですが、dapperはパラメータトークンをにきえます`?`クエリをするに

これはリストなどののとみわせてするので、がです

```
string region = "North";
int[] users = ...
var docs = conn.Query<Document>(@"
     select * from Documents
     where Region = ?region?
     and OwnerId in ?users?", new { region, users }).AsList();
```

したがって、 `.region`および`.users`メンバーはされ、されるSQLはたとえば、3のユーザーがです。

```
     select * from Documents
     where Region = ?
     and OwnerId in (?,?,?)
```

ただし、このをする、dapper はじパラメータをすることはできません 。これは、じパラメータきなでもよいをするがないようにするためです。じをするがあるは、のようにをすることをしてください。

```
declare @id int = ?id?; // now we can use @id multiple times in the SQL
```

ができないは、パラメーターにメンバーをできます。これにより、がされていることがらかになります。

```
int id = 42;
connection.Execute("... where ParentId = $id0$ ... SomethingElse = $id1$ ...",
      new { id0 = id, id1 = id });
```

オンラインでパラメータリファレンスをむ https://riptutorial.com/ja/dapper/topic/10/パラメータリファレンス

7: バルクインサート

`WriteToServer`および`WriteToServerAsync`は、コピーのデータのソースとして、IDataReaderでられる、DataTable、およびDataRow `DataRow[]` をけれるオーバーロードがあります。

Examples

バルクコピー

このサンプルでは、ここでする`ToDataReader`メソッドをして、 `ToDataReader` のリストDataReaderをします。

これは、メソッドをしてうこともできます。

```
public class Widget
{
    public int WidgetId {get;set;}
    public string Name {get;set;}
    public int Quantity {get;set;}
}

public async Task AddWidgets(IEnumerable<Widget> widgets)
{
    using(var conn = new SqlConnection("{connection string}")) {
        await conn.OpenAsync();

        using(var bulkCopy = new SqlBulkCopy(conn)) {
            bulkCopy.BulkCopyTimeout = SqlTimeoutSeconds;
            bulkCopy.BatchSize = 500;
            bulkCopy.DestinationTableName = "Widgets";
            bulkCopy.EnableStreaming = true;

            using(var dataReader = widgets.ToDataReader())
            {
                await bulkCopy.WriteToServerAsync(dataReader);
            }
        }
    }
}
```

コピー

このサンプルでは、ここでする`ToDataReader`メソッドをして、 `ToDataReader` のリストDataReaderをします。

これは、メソッドをしてうこともできます。

```
public class Widget
{
    public int WidgetId {get;set;}
    public string Name {get;set;}
```

```
    public int Quantity {get;set;}
}

public void AddWidgets(IEnumerable<Widget> widgets)
{
    using(var conn = new SqlConnection("{connection string}")) {
        conn.Open();

        using(var bulkCopy = new SqlBulkCopy(conn)) {
            bulkCopy.BulkCopyTimeout = SqlTimeoutSeconds;
            bulkCopy.BatchSize = 500;
            bulkCopy.DestinationTableName = "Widgets";
            bulkCopy.EnableStreaming = true;

            using(var dataReader = widgets.ToDataReader())
            {
                bulkCopy.WriteToServer(dataReader);
            }
        }
    }
}
```

オンラインでバルクインサートをむ https://riptutorial.com/ja/dapper/topic/6279/バルクインサート

8: マルチマッピング

- `public static IEnumerable<TReturn> Query<TFirst, TSecond, TReturn>(this IDbConnection cnn, string sql, Func<TFirst, TSecond, TReturn> map, object param = null, IDbTransaction transaction = null, bool buffered = true, string splitOn = "Id", int? commandTimeout = null, CommandType? commandType = null)`
- `public static IEnumerable<TReturn> Query<TFirst, TSecond, TThird, TFourth, TFifth, TSixth, TSeventh, TReturn>(this IDbConnection cnn, string sql, Func<TFirst, TSecond, TThird, TFourth, TFifth, TSixth, TSeventh, TReturn> map, object param = null, IDbTransaction transaction = null, bool buffered = true, string splitOn = "Id", int? commandTimeout = null, CommandType? commandType = null)`
- `public static IEnumerable<TReturn> Query<TReturn>(this IDbConnection cnn, string sql, Type[] types, Func<object[], TReturn> map, object param = null, IDbTransaction transaction = null, bool buffered = true, string splitOn = "Id", int? commandTimeout = null, CommandType? commandType = null)`

パラメータ

パラメータ	
cnn	あなたのデータベースはにオープンしているがあります。
SQL	するコマンド。
タイプ	レコードセットのの。
	されたのをする`Func<>`
パラメータ	パラメータをするオブジェクト。
トランザクション	このクエリがあるはそのトランザクションのであるトランザクション。
	せのをバッファリングするかどうか。これはなパラメータであり、デフォルトはtrueです。 bufferedがtrueの、は`List<T>`バッファされ、のでな`IEnumerable<T>`としてされます。 bufferedがfalseの、sqlはみみがするまでかれたままになり、メモリーののをできます。のにより、データベースへののがされます。バッファリングされたのメモリをするためのにですが、あなたはレコードだけのにさなをするならば、それはってさかなりのパフォーマンス·オーバーヘッドをにセットをにべて。に、のバッファリングされていないSQLがしているは、プールがになると、がになるまでブロックがすることをするがあります。
splitOn	フィールドは、2のオブジェクトをしてみむがありますデフォルトid。レコードにのがまれているは、コンマでられたリストにすることができます。
commandTimeout	コマンドタイムアウトまでの。
commandType	それはストアドプロシージャかバッチですか

Examples

シンプルなマルチテーブルマッピング

Personクラスをするがあるりののクエリがあるとします。

	うまれた	レジデンス
ダニエルデネット	1942	アメリカ
サムハリス	1967	アメリカ
リチャード・ドーキンス	1941	イギリス

```
public class Person
{
    public string Name { get; set; }
    public int Born { get; set; }
    public Country Residience { get; set; }
}

public class Country
{
    public string Residence { get; set; }
}
```

されたインスタンスをするためにできる`Func<>`をとるオーバーロード`Query<>`をして、 `Func<>`クラスとResidenceプロパティをCountryのインスタンスに`Func<>`ことができます。 `Func<>`は7つのタイプをとり、はにりのです。

```
var sql = @"SELECT 'Daniel Dennett' AS Name, 1942 AS Born, 'United States of America' AS
Residence
UNION ALL SELECT 'Sam Harris' AS Name, 1967 AS Born, 'United States of America' AS Residence
UNION ALL SELECT 'Richard Dawkins' AS Name, 1941 AS Born, 'United Kingdom' AS Residence";

var result = connection.Query<Person, Country, Person>(sql, (person, country) => {
        if(country == null)
        {
            country = new Country { Residence = "" };
        }
        person.Residience = country;
        return person;
    },
    splitOn: "Residence");
```

> `splitOn: "Residence"`のにしてください。このは、のクラスタイプこのは`Country` の1です。 Dapperはにする*Id*というをしますが、つからず`splitOn`がされていないは、なメッセージがされて`System.ArgumentException`がスローされます。したがって、オプションですが、は`splitOn`をする`splitOn`ます。

1マッピング

1のをむよりなをてみましょう。クエリにデ―タがまれるのがまれるようになりました。これをするがあります。これはクロ―ジャ―でします。

サンプルクラスとにクエリがわずかにされます。

イド		うまれた	CountryId		BookId	BookName
1	ダニエルデネット	1942	1	アメリカ	1	ブレインスト―ム
1	ダニエルデネット	1942	1	アメリカ	2	エルボ―ル―ム
2	サムハリス	1967	1	アメリカ	3	モラルの
2	サムハリス	1967	1	アメリカ	4	をますのないの
3	リチャ―ド·ド―キンス	1941	2	イギリス	5	のどのようにたちがっているかに
3	リチャ―ド·ド―キンス	1941	2	イギリス	6	のためのの

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Born { get; set; }
    public Country Residience { get; set; }
    public ICollection<Book> Books { get; set; }
}

public class Country
{
    public int CountryId { get; set; }
    public string CountryName { get; set; }
}

public class Book
{
    public int BookId { get; set; }
    public string BookName { get; set; }
}
```

`remainingHorsemen`には、にマテリアライズされたオブジェクトのインスタンスがりまれます。ク

エリのにして、lambdaでされたのインスタンスのマップされたがされ、これをどのようにするかはあなたです。

```
            var sql = @"SELECT 1 AS Id, 'Daniel Dennett' AS Name, 1942 AS Born, 1 AS
CountryId, 'United States of America' AS CountryName, 1 AS BookId, 'Brainstorms' AS BookName
UNION ALL SELECT 1 AS Id, 'Daniel Dennett' AS Name, 1942 AS Born, 1 AS CountryId, 'United
States of America' AS CountryName, 2 AS BookId, 'Elbow Room' AS BookName
UNION ALL SELECT 2 AS Id, 'Sam Harris' AS Name, 1967 AS Born, 1 AS CountryId,  'United States
of America' AS CountryName, 3 AS BookId, 'The Moral Landscape' AS BookName
UNION ALL SELECT 2 AS Id, 'Sam Harris' AS Name, 1967 AS Born, 1 AS CountryId,  'United States
of America' AS CountryName, 4 AS BookId, 'Waking Up: A Guide to Spirituality Without Religion'
AS BookName
UNION ALL SELECT 3 AS Id, 'Richard Dawkins' AS Name, 1941 AS Born, 2 AS CountryId,  'United
Kingdom' AS CountryName, 5 AS BookId, 'The Magic of Reality: How We Know What`s Really True'
AS BookName
UNION ALL SELECT 3 AS Id, 'Richard Dawkins' AS Name, 1941 AS Born, 2 AS CountryId,  'United
Kingdom' AS CountryName, 6 AS BookId, 'An Appetite for Wonder: The Making of a Scientist' AS
BookName";

var remainingHorsemen = new Dictionary<int, Person>();
connection.Query<Person, Country, Book, Person>(sql, (person, country, book) => {
    //person
    Person personEntity;
    //trip
    if (!remainingHorsemen.TryGetValue(person.Id, out personEntity))
    {
        remainingHorsemen.Add(person.Id, personEntity = person);
    }

    //country
    if(personEntity.Residience == null)
    {
        if (country == null)
        {
            country = new Country { CountryName = "" };
        }
        personEntity.Residience = country;
    }

    //books
    if(personEntity.Books == null)
    {
        personEntity.Books = new List<Book>();
    }

    if (book != null)
    {
        if (!personEntity.Books.Any(x => x.BookId == book.BookId))
        {
            personEntity.Books.Add(book);
        }
    }

    return personEntity;
},
splitOn: "CountryId,BookId");
```

`splitOn`が、ののののカンマりリストであることにしてください。

7のマッピング

マッピングするタイプのが、をうFunc <>によってされる7をえることがあります。

Query<>にジェネリックのをするわりに、としてマッピングすると、それにくマッピングをします。のマニュアルやのキャストは、りのはされません。

```
            var sql = @"SELECT 1 AS Id, 'Daniel Dennett' AS Name, 1942 AS Born, 1 AS
CountryId, 'United States of America' AS CountryName, 1 AS BookId, 'Brainstorms' AS BookName
UNION ALL SELECT 1 AS Id, 'Daniel Dennett' AS Name, 1942 AS Born, 1 AS CountryId, 'United
States of America' AS CountryName, 2 AS BookId, 'Elbow Room' AS BookName
UNION ALL SELECT 2 AS Id, 'Sam Harris' AS Name, 1967 AS Born, 1 AS CountryId,  'United States
of America' AS CountryName, 3 AS BookId, 'The Moral Landscape' AS BookName
UNION ALL SELECT 2 AS Id, 'Sam Harris' AS Name, 1967 AS Born, 1 AS CountryId,  'United States
of America' AS CountryName, 4 AS BookId, 'Waking Up: A Guide to Spirituality Without Religion'
AS BookName
UNION ALL SELECT 3 AS Id, 'Richard Dawkins' AS Name, 1941 AS Born, 2 AS CountryId,  'United
Kingdom' AS CountryName, 5 AS BookId, 'The Magic of Reality: How We Know What`s Really True'
AS BookName
UNION ALL SELECT 3 AS Id, 'Richard Dawkins' AS Name, 1941 AS Born, 2 AS CountryId,  'United
Kingdom' AS CountryName, 6 AS BookId, 'An Appetite for Wonder: The Making of a Scientist' AS
BookName";

var remainingHorsemen = new Dictionary<int, Person>();
connection.Query<Person>(sql,
    new[]
    {
        typeof(Person),
        typeof(Country),
        typeof(Book)
    }
    , obj => {

        Person person = obj[0] as Person;
        Country country = obj[1] as Country;
        Book book = obj[2] as Book;

        //person
        Person personEntity;
        //trip
        if (!remainingHorsemen.TryGetValue(person.Id, out personEntity))
        {
            remainingHorsemen.Add(person.Id, personEntity = person);
        }

        //country
        if(personEntity.Residience == null)
        {
            if (country == null)
            {
                country = new Country { CountryName = "" };
            }
            personEntity.Residience = country;
        }

        //books
        if(personEntity.Books == null)
        {
            personEntity.Books = new List<Book>();
```

```
        }

        if (book != null)
        {
            if (!personEntity.Books.Any(x => x.BookId == book.BookId))
            {
                personEntity.Books.Add(book);
            }
        }

        return personEntity;
},
splitOn: "CountryId,BookId");
```

カスタムマッピング

クエリのがクラスとしないは、のマッピングをできます。このは、カスタムマッピングとに`System.Data.Linq.Mapping.ColumnAttribute`をしたマッピングをしています。

> マッピングは、タイプごとに1だけするがあるため、アプリケーションのにするか、または1だけされるようにします。

One-to-Manyのとじクエリをし、クラスをよりいにリファクタリングします。

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Born { get; set; }
    public Country Residience { get; set; }
    public ICollection<Book> Books { get; set; }
}

public class Country
{
    [System.Data.Linq.Mapping.Column(Name = "CountryId")]
    public int Id { get; set; }

    [System.Data.Linq.Mapping.Column(Name = "CountryName")]
    public string Name { get; set; }
}

public class Book
{
    public int Id { get; set; }

    public string Name { get; set; }
}
```

> `Book`が`ColumnAttribute`しないにしてください。ただし、 `if`ステートメントをするがあります

はこのマッピングコードをアプリケーションのどこかにいて、だけします

```
Dapper.SqlMapper.SetTypeMap(
```

```
    typeof(Country),
    new CustomPropertyTypeMap(
        typeof(Country),
        (type, columnName) =>
            type.GetProperties().FirstOrDefault(prop =>
                prop.GetCustomAttributes(false)
                    .OfType<System.Data.Linq.Mapping.ColumnAttribute>()
                    .Any(attr => attr.Name == columnName)))
);


var bookMap = new CustomPropertyTypeMap(
    typeof(Book),
    (type, columnName) =>
    {
        if(columnName == "BookId")
        {
            return type.GetProperty("Id");
        }

        if (columnName == "BookName")
        {
            return type.GetProperty("Name");
        }

        throw new InvalidOperationException($"No matching mapping for {columnName}");
    }
);
Dapper.SqlMapper.SetTypeMap(typeof(Book), bookMap);
```

その、の`Query<>`のいずれかをしてクエリがされます。

このには、マッピングをするながされています 。

オンラインでマルチマッピングをむ https://riptutorial.com/ja/dapper/topic/351/マルチマッピング

9: テーブル

Examples

がいたまます る

テンポラリテーブルがでされると、がいているもテンポラリテーブルはそのまま ります。

```
// Widget has WidgetId, Name, and Quantity properties
public async Task PurchaseWidgets(IEnumerable<Widget> widgets)
{
    using(var conn = new SqlConnection("{connection string}")) {
        await conn.OpenAsync();

        await conn.ExecuteAsync("CREATE TABLE #tmpWidget(WidgetId int, Quantity int)");

        // populate the temp table
        using(var bulkCopy = new SqlBulkCopy(conn)) {
            bulkCopy.BulkCopyTimeout = SqlTimeoutSeconds;
            bulkCopy.BatchSize = 500;
            bulkCopy.DestinationTableName = "#tmpWidget";
            bulkCopy.EnableStreaming = true;

            using(var dataReader = widgets.ToDataReader())
            {
                await bulkCopy.WriteToServerAsync(dataReader);
            }
        }

        await conn.ExecuteAsync(@"
            update w
            set Quantity = w.Quantity - tw.Quantity
            from Widgets w
                join #tmpWidget tw on w.WidgetId = tw.WidgetId");
    }
}
```

テンポラリテーブルの

テーブルについてのポイントは、それらがのにされていることです。 Dapperは、まだかれていないはにをき、じます。つまり、Dapperにされたがかれていない、テーブルはにわれます。

これはしません

```
private async Task<IEnumerable<int>> SelectWidgetsError()
{
  using (var conn = new SqlConnection(connectionString))
  {
    await conn.ExecuteAsync(@"CREATE TABLE #tmpWidget(widgetId int);");

    // this will throw an error because the #tmpWidget table no longer exists
    await conn.ExecuteAsync(@"insert into #tmpWidget(WidgetId) VALUES (1);");
```

```
    return await conn.QueryAsync<int>(@"SELECT * FROM #tmpWidget;");
  }
}
```

、これらの2つのバージョンはします

```
private async Task<IEnumerable<int>> SelectWidgets()
{
  using (var conn = new SqlConnection(connectionString))
  {
    // Here, everything is done in one statement, therefore the temp table
    // always stays within the scope of the connection
    return await conn.QueryAsync<int>(
      @"CREATE TABLE #tmpWidget(widgetId int);
        insert into #tmpWidget(WidgetId) VALUES (1);
        SELECT * FROM #tmpWidget;");
  }
}

private async Task<IEnumerable<int>> SelectWidgetsII()
{
  using (var conn = new SqlConnection(connectionString))
  {
    // Here, everything is done in separate statements. To not loose the
    // connection scope, we have to explicitly open it
    await conn.OpenAsync();

    await conn.ExecuteAsync(@"CREATE TABLE #tmpWidget(widgetId int);");
    await conn.ExecuteAsync(@"insert into #tmpWidget(WidgetId) VALUES (1);");
    return await conn.QueryAsync<int>(@"SELECT * FROM #tmpWidget;");
  }
}
```

オンラインでテーブルをむ https://riptutorial.com/ja/dapper/topic/6594/テーブル

10: パラメータ

Examples

な

1つのオブジェクト/コールですべてのパラメータをきれいにまとめることはずしもではありません。よりなシナリオをするために、dapperは`param`パラメータを`IDynamicParameters`インスタンスにします。これをうと、カスタム`AddParameters`メソッドがなタイミングでびされ、にするコマンドがされます。ただし、ほとんどの、の`DynamicParameters`タイプをすればです。

```
var p = new DynamicParameters(new { a = 1, b = 2 });
p.Add("c", dbType: DbType.Int32, direction: ParameterDirection.Output);
connection.Execute(@"set @c = @a + @b", p);
int updatedValue = p.Get<int>("@c");
```

これはのとおりです

- のオブジェクトからのオプションの
- オプションオンザフライでパラメータをする
- パラメータをコマンドにす
- コマンドがしたにされたをする

RDBMSプロトコルがどのようにするかによって、、 `Query`またはQueryMultipleからのデータがにされたにされたパラメータをすることはできることにしてくださいたとえば、SQL Serverではされたパラメータがです TDSストリームの。

Dapperのパラメータ

```
connection.Execute(@"some Query with @a,@b,@c", new
{a=somevalueOfa,b=somevalueOfb,c=somevalueOfc});
```

テンプレートオブジェクトの

オブジェクトのインスタンスをしてパラメータをすることができます

```
public class SearchParameters {
  public string SearchString { get; set; }
  public int Page { get; set; }
}

var template= new SearchParameters {
  SearchString = "Dapper",
  Page = 1
};
```

```
var p = new DynamicParameters(template);
```

オブジェクトまたは`Dictionary`することもできます

オンラインでパラメータをむ https://riptutorial.com/ja/dapper/topic/12/パラメータ

11: ハンドラ

ハンドラをすると、データベースを.Netカスタムにできます。

Examples

varcharからIHtmlStringへの

```
public class IHtmlStringTypeHandler : SqlMapper.TypeHandler<IHtmlString>
{
    public override void SetValue(
        IDbDataParameter parameter,
        IHtmlString value)
    {
        parameter.DbType = DbType.String;
        parameter.Value = value?.ToHtmlString();
    }

    public override IHtmlString Parse(object value)
    {
        return MvcHtmlString.Create(value?.ToString());
    }
}
```

TypeHandlerのインストール

タイプハンドラはにインストールすることができる`SqlMapper`い`AddTypeHandler`。

```
SqlMapper.AddTypeHandler<IHtmlString>(new IHtmlStringTypeHandler());
```

では、パラメータをできます。

```
SqlMapper.AddTypeHandler(new IHtmlStringTypeHandler());
```

な`Type`をとる2のオーバーロードもあります。

```
SqlMapper.AddTypeHandler(typeof(IHtmlString), new IHtmlStringTypeHandler());
```

オンラインでハンドラをむ https://riptutorial.com/ja/dapper/topic/6/ハンドラ

12: なクエリ

- public static IEnumerable <T> Query <T>このIDbConnectionのcnn、sql、オブジェクトparam = null、SqlTransactionトランザクション= null、boolバッファリング= true
- public static IEnumerable <dynamic>クエリこのIDbConnection cnn、sql、オブジェクトparam = null、SqlTransactionトランザクション= null、bool buffered = true

パラメーター

パラメータ	
cnn	あなたのデータベースはにオープンしているがあります。
SQL	するコマンド。
パラメータ	パラメータをするオブジェクト。
トランザクション	このクエリがあるはそのトランザクションのであるトランザクション。
	せのをバッファリングするかどうか。これはなパラメータであり、デフォルトはtrueです。 bufferedがtrueの、は`List<T>`バッファされ、のでな`IEnumerable<T>`としてされます。 bufferedがfalseの、sqlはみみがするまでかれたままになり、メモリーののをできます。のにより、データベースへののがされます。バッファリングされたのメモリをするためのにですが、あなたはレコードだけのにさなをするならば、それはってさかなりのパフォーマンス·オーバーヘッドをにセットをにべて。に、のバッファリングされていないSQLがしているは、プールがになると、がになるまでブロックがすることをするがあります。

Examples

のクエリ

コンパイルにのについては、 `Query<T>`パラメータをしてください。

```
public class Dog
```

```
{
    public int? Age { get; set; }
    public Guid Id { get; set; }
    public string Name { get; set; }
    public float? Weight { get; set; }

    public int IgnoredProperty { get { return 1; } }
}

//
IDBConnection db = /* ... */;

var @params = new { age = 3 };
var sql = "SELECT * FROM dbo.Dogs WHERE Age = @age";

IEnumerable<Dog> dogs = db.Query<Dog>(sql, @params);
```

のクエリ

ジェネリックをすると、にクエリをすることもできます。

```
IDBConnection db = /* ... */;
IEnumerable<dynamic> result = db.Query("SELECT 1 as A, 2 as B");

var first = result.First();
int a = (int)first.A; // 1
int b = (int)first.B; // 2
```

パラメータをしたクエリ

```
var color = "Black";
var age = 4;

var query = "Select * from Cats where Color = :Color and Age > :Age";
var dynamicParameters = new DynamicParameters();
dynamicParameters.Add("Color", color);
dynamicParameters.Add("Age", age);

using (var connection = new SqlConnection(/* Your Connection String Here */))
{
    IEnumerable<dynamic> results = connection.Query(query, dynamicParameters);
}
```

オンラインでなクエリをむ https://riptutorial.com/ja/dapper/topic/3/なクエリ

13: の

- public static SqlMapper.GridReader QueryMultipleこのIDbConnection cnn、sql、オブジェクトparam = null、IDbTransactionトランザクション= null、intcommandTimeout = null、CommandTypecommandType = null
- public static SqlMapper.GridReader QueryMultipleこのIDbConnection cnn、CommandDefinitionコマンド

パラ メーター

パラ メータ	
cnn	あなたのデータベースはすでにいているがあります
SQL	するSQL。のクエリがまれます。
パラメータ	パラメータをするオブジェクト
SqlMapper.GridReader	Dapperクエリからのセットをみむためのインターフェイスをします。

Examples

の

1つのクエリでのグリッドをフェッチするには、 `QueryMultiple`メソッドがされます。これにより、された`GridReader`にするしたびしによってグリッドをにできます。

```
var sql = @"select * from Customers where CustomerId = @id
            select * from Orders where CustomerId = @id
            select * from Returns where CustomerId = @id";

using (var multi = connection.QueryMultiple(sql, new {id=selectedId}))
{
   var customer = multi.Read<Customer>().Single();
   var orders = multi.Read<Order>().ToList();
   var returns = multi.Read<Return>().ToList();
}
```

オンラインでのをむ https://riptutorial.com/ja/dapper/topic/8/の

14: の

Examples

ストアドプロシージャのびし

```
public async Task<Product> GetProductAsync(string productId)
{
    using (_db)
    {
        return await _db.QueryFirstOrDefaultAsync<Product>("usp_GetProduct", new { id =
productId },
            commandType: CommandType.StoredProcedure);
    }
}
```

ストアドプロシージャをびしてをする

```
public async Task SetProductInactiveAsync(int productId)
{
    using (IDbConnection con = new SqlConnection("myConnectionString"))
    {
        await con.ExecuteAsync("SetProductInactive", new { id = productId },
            commandType: CommandType.StoredProcedure);
    }
}
```

オンラインでのをむ https://riptutorial.com/ja/dapper/topic/1353/の

クレジット

S. No		Contributors
1	Dapper.NETをいめる	Adam Lear, balpha, Community, Eliza, Greg Bray, Jarrod Dixon, Kevin Montrose, Matt McCabe, Nick, Rob, Shog9
2	DbGeographyとDbGeometryの	Marc Gravell
3	コマンドの	Adam Lear, Jarrod Dixon, Sklivvz, takrl
4	トランザクション	jhamm
5	ヌルの	Marc Gravell
6	パラメータリファレンス	4444, Marc Gravell, Nick Craver
7	バルクインサート	jhamm
8	マルチマッピング	Devon Burriss
9	テーブル	jhamm, Rob, takrl
10	パラメータ	Marc Gravell, Matt McCabe, Meer
11	ハンドラ	Benjamin Hodgson, Community, Marc Gravell
12	なクエリ	Adam Lear, Chris Marisic, Cigano Morrison Mendez, Community, cubrr, Jarrod Dixon, jrummell, Kevin Montrose, Matt McCabe
13	の	Marc Gravell, Yaakov Ellis
14	の	Dean Ward, Matt McCabe, Nick, Woodchipper