**Dapper - a simple object mapper for .Net**

**Release Notes**

[Located at stackexchange.github.io/Dapper](https://stackexchange.github.io/Dapper/)

**Features**对象关系映射的一种ORM

Dapper is a [NuGet library](https://www.nuget.org/packages/Dapper) that you can add in to your project that will extend your IDbConnection interface.

It provides 3 helpers:

**Execute a query and map the results to a strongly typed List该框架查询数据库返回的结果为一个强类型的list**

publicstaticIEnumerable<T>Query<T>(thisIDbConnectioncnn, stringsql, objectparam=null（参数为一个对象）, SqlTransaction（事务）transaction=null, boolbuffered=true)

Example usage:

publicclassDog

{

publicint? Age{ get; set; }

publicGuidId { get; set; }

public stringName { get; set; }

public float? Weight{ get; set; }

publicintIgnoredProperty { get { return1; } }

}

varguid= Guid.NewGuid();

vardog = connection.Query<Dog>("select Age = @Age, Id = @Id", new { Age = (int?)null, Id = guid });

select Age = @Age, Id = @Id为sql语句

new { Age = (int?)null, Id = guid }为参数对象

由此可见，thisIDbConnectioncnn可省略；SqlTransaction（事务）transaction=null可省略；boolbuffered=true也可以省略

dog.Count()

.IsEqualTo(1);

dog.First().Age

.IsNull();

dog.First().Id

.IsEqualTo(guid);

**Execute a query and map it to a list of dynamic objects**

publicstaticIEnumerable<dynamic>Query (thisIDbConnectioncnn, stringsql, objectparam=null, SqlTransactiontransaction=null, boolbuffered=true)

This method will execute SQL and return a dynamic list.

Example usage:

varrows = connection.Query("select 1 A, 2 B union all select 3, 4");

可以只指定sql语句，这时返回的是一个类似表的结构

((int)rows[0].A)

.IsEqualTo(1);

((int)rows[0].B)

.IsEqualTo(2);

((int)rows[1].A)

.IsEqualTo(3);

((int)rows[1].B)

.IsEqualTo(4);

**Execute a Command that returns no results（执行一条无返回结果的命令）**

publicstaticintExecute(thisIDbConnectioncnn, string sql, objectparam = null, SqlTransactiontransaction = null)

Example usage:

connection.Execute(@"

setnocount on

create table #t(i int)

setnocount off

insert #t

select @a a union all select @b

setnocount on

drop table #t", new {a=1, b=2 })

.IsEqualTo(2);

**Execute a Command multiple times（多次执行一条命令）**

The same signature（签名）also allows you to conveniently and efficiently execute a command multiple times (for example to bulk-load data)

Example usage:

connection.Execute(@"insert MyTable(colA, colB) values (@a, @b)",

new[] { new { a=1, b=1 }, new { a=2, b=2 }, new { a=3, b=3 } }

).IsEqualTo(3); // 3 rows inserted: "1,1", "2,2" and "3,3"

This works for any parameter that implements IEnumerable for some T.

**Performance**

A key feature of Dapper is performance（性能）. The following metrics（.net中的一种度量工具） show how long it takes to execute 500 SELECT statements against a DB and map the data returned to objects.

The performance tests are broken in to 3 lists:

* POCO serialization for frameworks that suppor（支持）pulling static typed objects from the DB（从数据库中拉取静态类型对象）. Using raw SQL.
* Dynamic serialization（动态序列化） for frameworks that support returning dynamic lists of objects.
* Typical framework usageo（典型框架的使用）. Often typical framework usage differs from the optimal（最佳的） usage performance wise. Often it will not involve writing SQL.

**Performance of SELECT mapping over 500 iterations - POCO serialization**

|  |  |  |
| --- | --- | --- |
| **Method** | **Duration** | **Remarks** |
| Hand coded (using a SqlDataReader) | 47ms | [Can be faster](http://www.toptensoftware.com/blog/posts/94-PetaPoco-More-Speed) |
| Dapper ExecuteMapperQuery | 49ms |
| [ServiceStack.OrmLite](https://github.com/ServiceStack/ServiceStack.OrmLite) (QueryById) | 50ms |
| [PetaPoco](http://www.toptensoftware.com/petapoco/) | 52ms |
| BLToolkit | 80ms |
| SubSonicCodingHorror | 107ms |
| NHibernate SQL | 104ms |
| Linq 2 SQL ExecuteQuery | 181ms |
| Entity framework ExecuteStoreQuery | 631ms |

**Performance of SELECT mapping over 500 iterations - dynamic serialization**

|  |  |  |
| --- | --- | --- |
| **Method** | **Duration** | **Remarks** |
| Dapper ExecuteMapperQuery (dynamic) | 48ms |  |
| [Massive](https://github.com/FransBouma/Massive) | 52ms |
| [Simple.Data](https://github.com/markrendle/Simple.Data) | 95ms |

**Performance of SELECT mapping over 500 iterations - typical usage**

|  |  |  |
| --- | --- | --- |
| **Method** | **Duration** | **Remarks** |
| Linq 2 SQL CompiledQuery | 81ms | Not super typical involves complex code |
| NHibernate HQL | 118ms |  |
| Linq 2 SQL | 559ms |  |
| Entity framework | 859ms |  |
| SubSonicActiveRecord.SingleOrDefault | 3619ms |  |

Performance benchmarks are available [here](https://github.com/StackExchange/dapper-dot-net/blob/master/Dapper.Tests/PerformanceTests.cs).

Feel free to submit patches（补充） that include other ORMs - when running benchmarks, be sure to compile in Release and not attach a debugger (ctrl F5).

Alternatively, you might prefer FransBouma's [RawDataAccessBencher](https://github.com/FransBouma/RawDataAccessBencher) test suite or [OrmBenchmark](https://github.com/InfoTechBridge/OrmBenchmark).

**Parameterized queries（参数化请求）**

Parameters are passed in as anonymous（匿名类） classes. This allow you to name your parameters easily and gives you the ability to simply cut-and-paste SQL snippets（片段） and run them in Query analyzer.

new {A = 1, B = "b"} // A will be mapped to the param @A, B to the param @B

**List Support（支持list）**

Dapper allow you to pass in（传递进） IEnumerable（枚举的） and will automatically parameterize(参数化) your query.

For example:

connection.Query<int>("select \* from (select1 as Id union all select 2 union all select 3) as X where Id in @Ids", new { Ids = newint[] { 1, 2, 3 } });

Will be translated to:

select \* from (select1 as Id union all select 2 union all select 3) as X where Id in (@Ids1, @Ids2, @Ids3)" // @Ids1 = 1 , @Ids2 = 2 , @Ids2 = 3

**Buffered vs Unbuffered readers（缓冲与否）**

Dapper's default behavior is to execute your sql and buffer the entire reader on return. This is ideal in most cases as it minimizes shared locks（共享锁） in the db and cuts down on db network time.

However when executing huge queries you may need to minimize memory footprint and only load objects as needed. To do so pass, buffered: false into the Query method.耗内存的大量搜索可以将buffer关闭

**Multi Mapping（多重映射）**

Dapper allows you to map a single row to multiple objects. This is a key feature if you want to avoid extraneou（不相关的） querying and eager load associations（关系）.

Example:

Consider 2 classes: Post and User

Class Post

{

publicintId { get; set; }

public stringTitle { get; set; }

public stringContent { get; set; }

public UserOwner { get; set; }

}

Class User

{

Publicint Id { get; set; }

public string Name { get; set; }

}

Now let us say that we want to map a query that joins both the posts and the users table. Until now if we needed to combine the result of 2 queries, we'd need a new object to express it but it makes more sense in this case to put the User object inside the Post object.

This is the user case for multi mapping. You tell dapper that the query returns a Post and a User object and then give it a function describing what you want to do with each of the rows containing both a Post and a User object. In our case, we want to take the user object and put it inside the post object. So we write the function:

(post, user) =>{ post.Owner = user; return post; }

The 3 type arguments to the Query method specify what objects dapper should use to deserialize the row and what is going to be returned. We're going to interpret both rows as a combination of Post and User and we're returning back a Postobject. Hence the type declaration becomes

<Post, User, Post>

Everything put together, looks like this:

varsql=

@"select \* from #Posts p

left join #Users u on u.Id = p.OwnerId

Order by p.Id";

vardata = connection.Query<Post, User, Post>(sql, (post, user) => { post.Owner = user; return post;});

varpost = data.First();

post.Content.IsEqualTo("Sams Post1");

post.Id.IsEqualTo(1);

post.Owner.Name.IsEqualTo("Sam");

post.Owner.Id.IsEqualTo(99);

Dapper is able to split the returned row by making an assumption that your Id columns are named Id or id, if your primary key is different or you would like to split the wide row at point other than Id, use the optional splitOn parameter.

**Multiple Results**

Dapper allows you to process multiple result grids in a single query.

Example:

varsql=

@"

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}))

{

varcustomer = multi.Read<Customer>().Single();

varorders = multi.Read<Order>().ToList();

varreturns = multi.Read<Return>().ToList();

...

}

**Stored Procedures**

Dapper fully supports stored procs:

varuser = cnn.Query<User>("spGetUser", new {Id = 1},

commandType: CommandType.StoredProcedure).SingleOrDefault();

If you want something more fancy, you can do:

varp = newDynamicParameters();

p.Add("@a", 11);

p.Add("@b", dbType: DbType.Int32, direction: ParameterDirection.Output);

p.Add("@c", dbType: DbType.Int32, direction: ParameterDirection.ReturnValue);

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

intb = p.Get<int>("@b");

intc = p.Get<int>("@c");

**Ansi Strings and varchar**

Dapper supports varchar params, if you are executing a where clause on a varchar column using a parambe sure to pass it in this way:

Query<Thing>("select \* from Thing where Name = @Name", new {Name = newDbString { Value = "abcde", IsFixedLength = true, Length = 10, IsAnsi = true });

On SQL Server it is crucial to use the unicode when querying unicode and ansi when querying non unicode.

**Type Switching Per Row（多行类型转换）**

Usually you'll want to treat all rows from a given table as the same data type. However, there are some circumstances where it's useful to be able to parse different rows as different data types（将每一行解析为不同的数据类型，不同的对象类型）. This is where IDataReader.GetRowParser comes in handy.

Imagine you have a database table named "Shapes" with the columns: Id, Type, and Data, and you want to parse its rows into Circle, Square, or Triangle objects based on the value of the Type column.

varshapes = newList<IShape>();

using (var reader = connection.ExecuteReader("select \* fromShapes"))

{

// Generate a row parser for each type you expect.

// The generic type <IShape> is what the parser will return.

// The argument (typeof(\*)) is the concrete type to parse.

varcircleParser = reader.GetRowParser<IShape>(typeof(Circle));

varsquareParser= reader.GetRowParser<IShape>(typeof(Square));

vartriangleParser= reader.GetRowParser<IShape>(typeof(Triangle));

vartypeColumnIndex= reader.GetOrdinal("Type");

while (reader.Read())

{

IShapeshape;

vartype = (ShapeType)reader.GetInt32(typeColumnIndex);

switch (type)

{

caseShapeType.Circle:

shape = circleParser(reader);

break;

caseShapeType.Square:

shape = squareParser(reader);

break;

caseShapeType.Triangle:

shape = triangleParser(reader);

break;

default:

thrownewNotImplementedException();

}

shapes.Add(shape);

}

}

**Limitations and caveats（警告，备注）**

Dapper caches（缓存） information about every query it runs, this allow it to materialize（实质化，具体化） objects quickly and process parameters quickly. The current implementation caches this information in a ConcurrentDictionary object. The objects it stores are never flushed. If you are generating SQL strings on the fly without using parameters it is possible you will hit memory issues. We may convert the dictionaries to an LRU Cache.

Dapper's simplicity means that many feature that ORMs ship with are stripped out. It worries about the 95% scenario, and gives you the tools you need most of the time. It doesn't attempt to solve every problem.

**Will Dapper work with my DB provider?**

Dapper has no DB specific implementation details, it works across all .NET ADO providers including [SQLite](http://www.sqlite.org/), SQL CE, Firebird, Oracle, MySQL, PostgreSQL and SQL Server.

**Do you have a comprehensive list of examples?**

Dapper has a comprehensive test suite in the [test project](https://github.com/StackExchange/dapper-dot-net/blob/master/Dapper.Tests/Tests.cs)

**Who is using this?**

Dapper is in production use at:

[Stack Overflow](http://stackoverflow.com/), [helpdesk](https://www.jitbit.com/web-helpdesk/)

(if you would like to be listed here let me know)