# Dapper简明教程

[Dapper](https://github.com/StackExchange/dapper-dot-net)是一款轻量级的ORM框架，有关Dapper优缺点的文章网上一大堆，这里就不再赘述啦。下面直接进入正题：

**使用前准备**

添加对Dapper的引用

　　在使用Dapper之前，我们要首先添加对Dapper的引用，这里使用NuGet来添加引用。因为使用的是MySQL数据库，所以也要在项目中添加对MySql.Data的引用。

　　Dapper是一款ORM框架，用于数据表和实体模型间的映射，所以在使用前我们还需要创建数据表和实体模型。

创建数据表

[复制代码](javascript:void(0);)

CREATE TABLE `t\_schools` (

`Id` int(**11**) NOT NULL AUTO\_INCREMENT ,

`Name` varchar(**20**) NOT NULL ,

`Address` varchar(**50**) NOT NULL ,

PRIMARY KEY (`Id`)

)

CREATE TABLE `t\_students` (

`Id` int(**11**) NOT NULL AUTO\_INCREMENT ,

`Name` varchar(**15**) NOT NULL ,

`Number` varchar(**15**) NOT NULL ,

`SchoolId` int(**11**) NOT NULL ,

`Gender` enum('男','女','保密') NOT NULL ,

PRIMARY KEY (`Id`)

)

[复制代码](javascript:void(0);)

创建模型

[复制代码](javascript:void(0);)

class School

{

/\*

若属性名和数据库字段不一致（不区分大小写）则查询不出数据，如果使用EF则可以通过Column特性

建立属性和数据表字段之间的映射关系，Dapper则不行

\*/

//[Column("Name")]

public string Title { set; get; }

public string Address { set; get; }

}

class Student

{

public string Name { set; get; }

public string Number { set; get; }

public int SchoolId { set; get; }

}

[复制代码](javascript:void(0);)

**Dapper的基本用法**

[复制代码](javascript:void(0);)

const string \_connectionString = "Database=Dapper;Data Source=127.0.0.1;User Id=root;Password=root;pooling=false;CharSet=utf8;port=3306;";

using (IDbConnection dbConnection = new MySqlConnection(\_connectionString))

{

dbConnection.Open();

//通过匿名类型插入单条数据

dbConnection.Execute("insert into t\_schools(Name,Address) values(@Name,@Address)", new { Name = "西南大学", Address = "重庆市北碚区天生路2号" });

//批量插入数据

List<School> schools = new List<School>()

{

new School() {Address="China·BeiJing",Title="清华大学" },

new School() {Address="杭州",Title="浙江大学" },

new School() {Address="不知道，US?",Title="哈弗大学" }

};

//在执行参数化的SQL时，SQL中的参数（如@title可以和数据表中的字段不一致，但要和实体类型的属性Title相对应）

dbConnection.Execute("insert into t\_schools(Address,Name) values(@address,@title);", schools);

//通过匿名类型批量插入数据

dbConnection.Execute("insert into t\_schools(Address,Name) values(@address,@name)",

new[] {

new {Address="杨浦区四平路1239号",Name="同济大学"},

new {Address="英国",Name="剑桥"},

new {Address="美国·硅谷",Name="斯坦福大学"}

});

}

[复制代码](javascript:void(0);)

**使用Dapper进行查询操作**

　　默认情况下Dapper会将查询到的整个数据集放到内存中，可以在Query方法中通过参数buffered来设置是否将查询结果存放到内存中

查询结果映射到强类型

var schools = dbConnection.Query<School>("select \* from t\_schools where Name=@name", new { Name = "西南大学" });

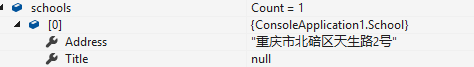
foreach (var school in schools)

{

Console.WriteLine(school.Address);

}

查询变量如下图：



　　有上图我们可以看到，因为School类中的Title属性在数据库中没有与之对应的字段，所以Title的值为null。查询结果见下图：

http://images2015.cnblogs.com/blog/784150/201609/784150-20160909095043723-1471868288.png

查询结果映射到匿名类型

　　在上面的查询中，我们将查询结果映射到了自定义的类型上。除了将查询结果映射到强类型之外，Dapper中还提供了匿名查询方式。

[复制代码](javascript:void(0);)

//查询结果result是匿名类型

var result = dbConnection.Query("select \* from t\_schools limit 0,3");

var resultList = result.AsList();

foreach (var l in resultList)

{

Console.WriteLine(l.Name);

}

[复制代码](javascript:void(0);)

查询结果如下：

http://images2015.cnblogs.com/blog/784150/201609/784150-20160909095812348-55019449.png

in

[复制代码](javascript:void(0);)

var result = dbConnection.Query<Student>("select \* from t\_students where SchoolId in @schoolId", new { schoolId = new int[] { 2, 3 } });

foreach (var r in result)

{

var ps = r.GetType().GetProperties();

foreach (var p in ps)

{

Console.Write(p.Name + "=" + p.GetValue(r) + " ");

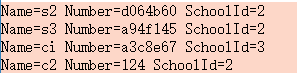
}

Console.WriteLine();

}

[复制代码](javascript:void(0);)

查询结果如下：



between

[复制代码](javascript:void(0);)

var result = dbConnection.Query<School>("select Name,Address from t\_schools where Id between @start and @end", new { start = 2, end = 4 });

foreach (var r in result)

{

var ps = r.GetType().GetProperties();

foreach (var p in ps)

{

Console.Write(p.Name + "=" + p.GetValue(r) + " ");

}

Console.WriteLine();

}

[复制代码](javascript:void(0);)

 查询结果如下：

http://images2015.cnblogs.com/blog/784150/201609/784150-20160909100123598-1017002561.png

join

　　使用join查询时需要用到Query方法中的splitOn参数，话说这个参数让纠结了很久才弄明白。关于splitOn参数的说明，可参考stackoverflow上的一篇文章[Correct use of Multimapping in Dapper](http://stackoverflow.com/questions/7472088/correct-use-of-multimapping-in-dapper/7478958#7478958)

[复制代码](javascript:void(0);)

var result = dbConnection.Query<Student, School, string>("select s.Name,sc.Address from t\_students s,t\_schools sc where s.SchoolId=sc.Id and binary sc.Address like '%BeiJing%'",

(s, sc) =>

{

return s.Name + " " + sc.Address;

},

/\*

还有一点需要特别注意，泛型参数的顺序必须和SQL语句查询数据表的顺序一致，

即Student对应t\_students表的查询结果s.Name，否则Query方法的查询结果

可能为null（这点也是困扰很久......）

\*/

splitOn: "Address"

);

foreach (var r in result)

{

Console.WriteLine(r);

}

[复制代码](javascript:void(0);)

查询结果如下：

http://images2015.cnblogs.com/blog/784150/201609/784150-20160909100204785-1722076081.png

**Dapper执行多条SQL语句**

[复制代码](javascript:void(0);)

string sql = "select Address from t\_schools;select SchoolId from t\_students;select Name from t\_students";

using (var multipleReader = dbConnection.QueryMultiple(sql))

{

//一次执行N条SQL语句则最多只能调用N次Read方法，否则抛异常:the reader has been disposed.

//Dapper读取查询结果数据的顺序必须要和查询语句中的查询顺序一致，否则可能读取不到数据

var schoolList = multipleReader.Read<School>();

foreach (var s in schoolList)

{

Console.Write(s.Address + " ");

}

Console.WriteLine();

var studentSchools = multipleReader.Read<Student>();

foreach (var s in studentSchools)

{

Console.Write(s.SchoolId + " ");

}

Console.WriteLine();

var studentNames = multipleReader.Read<Student>();

foreach (var s in studentNames)

{

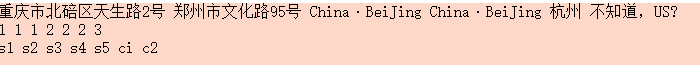
Console.Write(s.Name + " ");

}

}

[复制代码](javascript:void(0);)

查询结果如下：



**事务**

使用Dapper执行事务倒是没有什么需要特别说明的。

[复制代码](javascript:void(0);)

using (IDbTransaction tran = dbConnection.BeginTransaction())

{

try

{

dbConnection.Execute("delete from t\_schools where Id=3");

throw new Exception();

tran.Commit();

}

catch

{

tran.Rollback();

}

}

[复制代码](javascript:void(0);)

**存储过程**

首先先创建一个存储过程

DROP PROCEDURE IF EXISTS `GetSchoolName`;

CREATE PROCEDURE `GetSchoolName`(in schoolId int,out scname varchar(**20**))

BEGIN

select `Name` into scname from t\_schools where Id=schoolId;

select scname;

END;

然后在程序中调用存储过程

//在程序中调用存储过程时，存储过程名要小写，传递的参数名要和存储过程中的参数名一致（不区分大小写）

//连接字符串中的数据库名也要小写，否则抛异常：在数据库\*\*\*中找不到存储过程×××

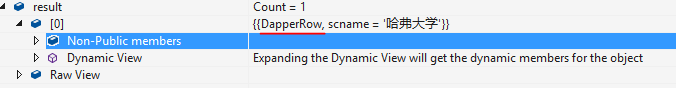
var parameters = new DynamicParameters();

parameters.Add("@scname", dbType: DbType.String, direction: ParameterDirection.Output);

parameters.Add("schoolid", 6, direction: ParameterDirection.Input);

var result = dbConnection.Query("getschoolname", parameters, commandType: CommandType.StoredProcedure);

执行结果如下：



从上图可以看出，返回值类型是Dapper中定义的DapperRow类型。

**结语**

* Dapper是一个轻量级的ORM框架，它是通过**扩展IDbConnection接口**来实现一系列的功能的。相比EF、NHibernate，它的功能较为简单。
* Dapper在执行查询语句时会**缓存SQL语句**的相关信息，这样就保证了Dapper拥有较高的性能(原文：Dapper caches information about every query it runs, this allow it to materialize objects quickly and process parameters quickly)。