Dapper 从入门到精通

版本	日期	参与人	备注
V1.0	2014/7/3	Micro	

Dapper 开场白

1) 首先对群友表示歉意，因为很久以前就答应写一系列 Dapper 的文章，由于各种事情耽误了，这次终于有时间了

2) 使用 Dapper 很长时间了，从第一次在 Google 上看到这个 Micro ORM 就喜欢上了这个小项目的精简，干练。

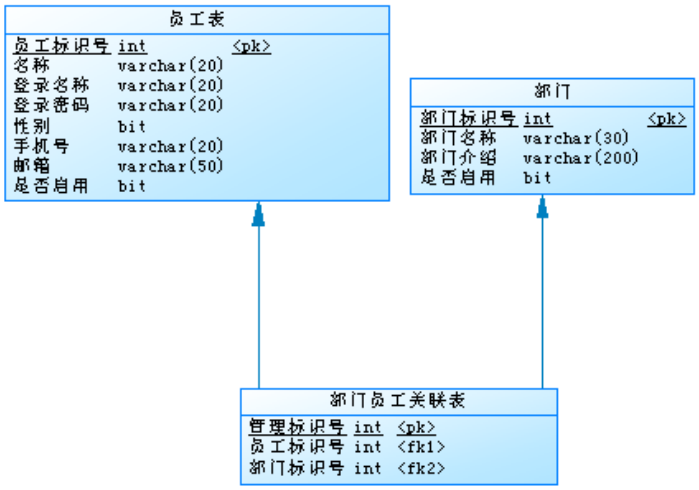
3) Dapper 不是庞大的 ORM，没有 Nhibernate 完善的机制，他只是实现了从 D→M、M→D(需要用到扩展)的过程，其实这个过程就已经给我们开发者提供了很大的帮助，因为我们每天的任务也是这些。

4) Dapper 是直接扩展 IDBConnection，而且是单独一个文件，可以直接嵌入到项目中使用。

1. 前期准备工作

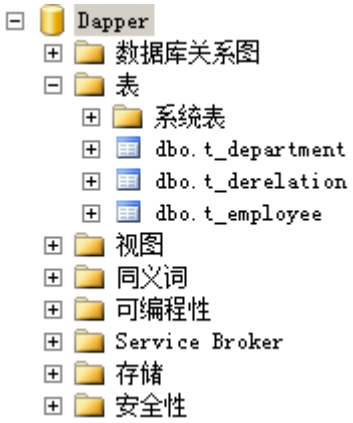
欲善其事,必先利其器，再正式接触 Dapper 之前，我们也要准备一下，祖先告诫的经验是很有价值的，有点扯远了。 ☺

1) 在 PD 中建一个物理数据模型，一共三个表，员工表、部门表、还有一个员工部门关联表。如下图(1-1)



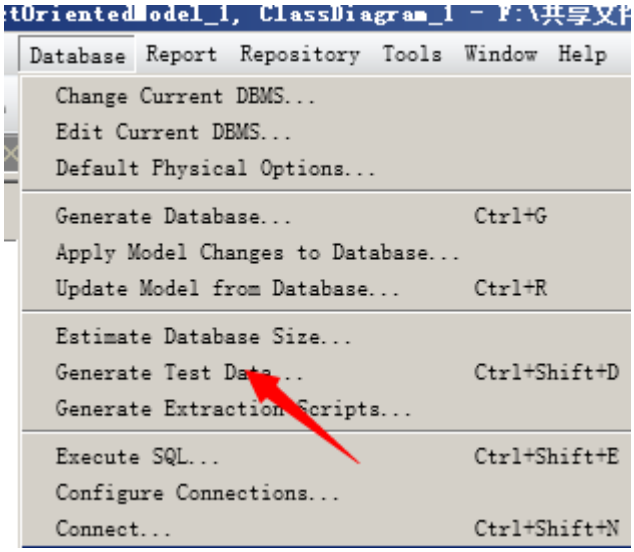
(1-1)

2) 建好物理模型后，直接生成数据结构就可以，如图(1-2)



(1-2)

3) 生成表后，用 PD 链接数据库，然后通过 Database 菜单下面的 Generate Test Data 生成测试数据 (1-3)



(1-3)

2. Dapper 的 CRUD 操作

1) 添加

a) 通过 SQL 添加数据，（2-1）是 手工写的 SQL,（2-2）是执行的结果,通过 Excute 方法,返回的不是主键值,而是影响的行数。

```
#region 手写Sql插入数据
/// <summary>
/// 手写Sql插入数据
/// </summary>
public int InsertWithSql()
{
    using (var conn = Connection)
    {
        string _sql ="INSERT INTO t_department(departmentname,introduce,[enable])VALUES('应用开发部SQL','应用开发部主要开始公司的应用平台',1)";
        conn.Open();
        return conn.Execute(_sql);
    }
}
#endregion
```

(2-1)

(2-2)

b) 通过实体插入数据，(2-3) 是实体数据，（2-4）是执行结果，可能你已经看出来了，这次返回的是主键值，是的，通过 Insert 方法插入返回的是主键值。

```
#region 手写Sql插入数据
/// <summary>
/// 手写Sql插入数据
/// </summary>
public int? InsertWithEntity()
{
    using (var conn = Connection)
    {
        var _entity = new t_department { departmentname = "应用开发部ENTITY", introduce = "应用开发部主要开始公司的应用平台"};
        conn.Open();
        return conn.Insert(_entity);
    }
}
#endregion
```

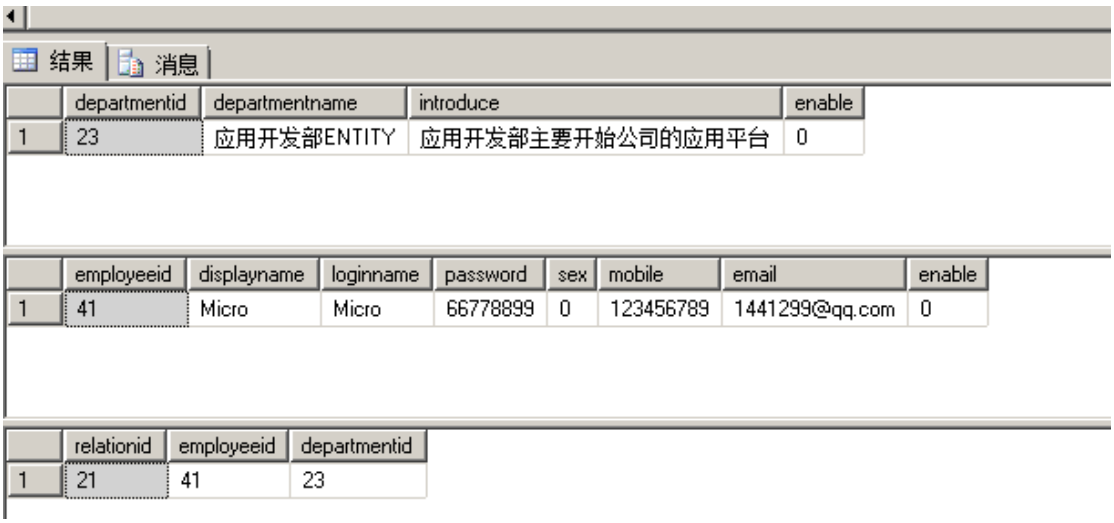
(2-3)

(2-4)

c) 在 IDbConnection 事务下多表插入，主要注意在执行 Insert 方法时传入 Transaction，在正常情况下 Commit 事务，在异常及 finally 中回滚事务。(2-5)为执行结果。

```
1   public bool InsertWithTran()
2       {
3           using (var conn = Connection)
4           {
5             int _departmentid = 0, _employeeid = 0,_rnum=0;
6              var _departmentname = new t_department { departmentname = "应用开发部
    ENTITY", introduce = "应用开发部主要开始公司的应用平台" };
7              var _employee = new t_employee {displayname = "Micro",email
    ="1441299@qq.com",loginname ="Micro",password = "66778899",mobile =
    "123456789"};
8              conn.Open();
```

```
9               var _tran=conn.BeginTransaction();
10              try
11              {
12                  _departmentid=conn.Insert(_departmentname, transaction: _tran).Value;
13                  ++_rnum;
14                  _employeeid = conn.Insert(_employee, transaction: _tran).Value;
15                  ++_rnum;
16                  conn.Insert(new t_derelation { departmentid = _departmentid, employeeid =
    _employeeid }, transaction: _tran);
17                  ++_rnum;
18                  _tran.Commit();
19              }
20              catch
21              {
22                  _rnum = 0;
23                  _tran.Rollback();
24              }
25              finally
26              {
27                  if(_rnum>0)
28                      _tran.Rollback();
29              }
30              return _rnum > 0;
31          }
32      }
```



(2-5)

d) 在存储过程事务下多表插入，首先我们写下面的 procedure（2-6），然后再写个调用存储过程的方法（2-7），在传递给存储过程参数的时候，要通过 DynamicParameters 这个类传递，再把命令类型赋值为 procedure，然后通过 Query 调用我们写的存储过程，（2-8）为执行结果。

```
1   alter PROCEDURE p_Insertdata
2   (
3   /*t_department parameter start*/
4   @departmentname VARCHAR(30),
5   @introduce VARCHAR(200),
6   /*t_department parameter end*/
7   /*t_employee parameter start*/
```

```
@displayname VARCHAR(20),
@loginname VARCHAR(20),
@password VARCHAR(20),
@mobile VARCHAR(20),
@email VARCHAR(50)
/*t_employee t_employee start*/
)
AS
BEGIN
  DECLARE @return BIT;/*1：成功，：失败*/
  DECLARE @departmentid int,@employeeid INT;
  SET @return=1;
  BEGIN TRANSACTION
  INSERT INTO
t_department(departmentname,introduce)VALUES(@departmentna
me,@introduce)
  IF(@@ERROR>0)
  BEGIN
   SET @return=0;
   ROLLBACK TRANSACTION
   END
   ELSE
   BEGIN
        SELECT @departmentid=scope_identity();
        INSERT INTO t_employee
        (
      displayname,loginname,[password],mobile,email)
        VALUES
        (@displayname,@loginname,@password,@mobile,@email)
        IF(@@ERROR>0)
        BEGIN
              SET @return=0;
              ROLLBACK TRANSACTION
        END
        ELSE
        BEGIN
              SELECT @employeeid=scope_identity();
              INSERT INTO
t_derelation(employeeid,departmentid)VALUES(@employeeid,@d
epartmentid)
               IF(@@ERROR>0)
               BEGIN
                    SET @return=0;
                    ROLLBACK TRANSACTION
               END
               ELSE
               BEGIN
                    COMMIT TRANSACTION
               END
        END
   END
SELECT @return;
END
```

```
1   public bool InsertWithProcTran()
2   {
3   var _parameter = new DynamicParameters();
4   _parameter.Add("departmentname","外网开发部门");
5   _parameter.Add("introduce","外网开发部门负责外部网站的更新");
6   _parameter.Add("displayname","夏季冰点");
7   _parameter.Add("loginname","Micro");
8   _parameter.Add("password","123456789");
9   _parameter.Add("mobile","1122334455");
10  _parameter.Add("email","123456789@qq.com");
11  using (var _conn = Connection)
12  {
13  _conn.Open();
14  return
15  _conn.Query<bool>("p_Insertdata", _parameter, commandType:
    CommandType.StoredProcedure)
16  .FirstOrDefault();
17  }
18  }
```

(2-7)

	departmentid	departmentname	introduce	enable
1	25	外网开发部门	外网开发部门负责外部网站的更新	1

	employeeid	displayname	loginname	password	sex	mobile	email	enable
1	43	夏季冰点	Micro	123456789	1	1122334455	123456789@qq.com	1

	relationid	employeeid	departmentid
1	23	43	25

（2-8）

2) 查询

a) 基本 SQL 查询

通过 SQL 查询是使用 Query <T>泛型方法，（2-9）是获取所有员工信息

```
1   public IEnumerable<t_employee> GetDepartmentListFirst()
2   {
3   string _sql = "SELECT * FROM t_employee";
4   using (var _conn = Connection)
5   {
6    conn.Open();
7   return _conn.Query<t_employee>(_sql);
8   }
9   }
```

（2-9）

b) 实体直接查询

用 GetList<T>获取员工信息（2-10）

```
1.  public IEnumerable<t_employee> GetDepartmentListSecond()
2.  {
3.  using (var _conn = Connection)
4.  {
5.  _conn.Open();
6.  return _conn.GetList<t_employee>();
7.  }
8.  }
```

（2-10）

c) 多表查询

在 Dapper 中多表查询可以用 QueryMultiple 方法，可以返回查询中每条 SQL 语句的数据集合，使用方法如（2-11）

```
1.  public void GetMultiEntity()
2.      {
3.          string _sql = "SELECT * FROM t_department AS a;SELECT * FROM
    t_employee AS a";
4.          using (var _conn = Connection)
5.          {
6.              var _grid = _conn.QueryMultiple(_sql);
7.              var _department = _grid.Read<t_department>();
8.              var _employee = _grid.Read<t_employee>();
9.          }
10.     }
```

(2-11)

d) 通用分页解决之道

通用分页是有一个 Procedure（2-12）协助的,然后再配合一个参数模型（2-13）,调用方式就很简单（2-14），这样就可以解决项目中的分页问题，开发效率，执行效率都不错。

```
1   Create Procedure p_PageList
2   @Tables varchar(200),
3   @Fields varchar(500) = '*',
4   @OrderFields varchar(100),
5   @Where varchar(100) = Null,
6   @PageIndex int = 1 ,
7   @PageSize int = 20,
8   @GroupBy varchar(100)=null,
9   @TotalCount int = 0 OUTPUT
10  as
11  begin
12  /*
13  *
14  *创建时间:201-07-05
15  *Author:Micro
16  *修改时间
17  */
18  SET NOCOUNT ON;
19  Declare @sql nvarchar(4000);
20  DECLARE @PageCount INT;
```

```
21  if(@GroupBy ='' or @GroupBy is null)
22  begin
23  set @sql = 'select @RecordCount = count(*) from ' +
    @Tables
24  if (@Where<>'' and @Where is not NULL)
25  set @sql = @sql + ' where ' + @Where
26  end
27  else
28  begin
29  set @sql = 'select @Recordcount=count(*)
    from(select 1 as total from ' + @Tables
30  if (@Where<>'' and @Where is not NULL)
31  set @sql = @sql + ' where ' + @Where
32  set @sql = @sql + ' group by ' + @GroupBy+') as t'
33  end

34  EXEC sp_executesql @sql,N'@RecordCount int
    OUTPUT',@TotalCount OUTPUT
35  select
    @PageCount=CEILING((@TotalCount+0.0)/@PageSize)

36  set @sql = 'Select * FROM (select ROW_NUMBER()
    Over(order by ' + @OrderFields + ') as rowId,' +
    @Fields + ' from ' + @Tables
37  if (@Where<>'' and @Where is not NULL)
38  set @sql = @sql + ' where ' + @Where
39  if(@GroupBy <>'' and @GroupBy is not null)
40  set @sql = @sql + ' group by ' + @GroupBy
41  if @PageIndex<=0
42  Set @PageIndex = 1
43  if @PageIndex>@PageCount
44  Set @PageIndex = @PageCount
45  Declare @StartRecord INT,@EndRecord int
46  select @StartRecord = (@PageIndex-1)*@PageSize +
    1,@EndRecord = @StartRecord + @PageSize - 1
47  set @Sql = @Sql + ') as ' + @Tables + ' where rowId
    between ' + Convert(varchar(50),@StartRecord) + '
    and ' +  Convert(varchar(50),@EndRecord)
48  Exec(@Sql)
49  ----------------------------------------------------
50  SET NOCOUNT OFF;
51  End
```

(2-12)

```
1.  public class p_PageList<T> where T:class,new()
2.  {
3.  public string Tables { get; set;}
4.  public string Fields { get; set;}
5.  public string OrderFields { get; set;}
6.  public string Where { get; set;}
7.  public int PageIndex { get; set;}
8.  private int _pagesize = 20;
9.  public int PageSize {
10. get { return _pagesize; }
11. set { _pagesize = value;}
12. }
13. public string GroupBy { get; set;}
14. public int TotalCount { get; set;}
15. public int PageCount
16. {
```

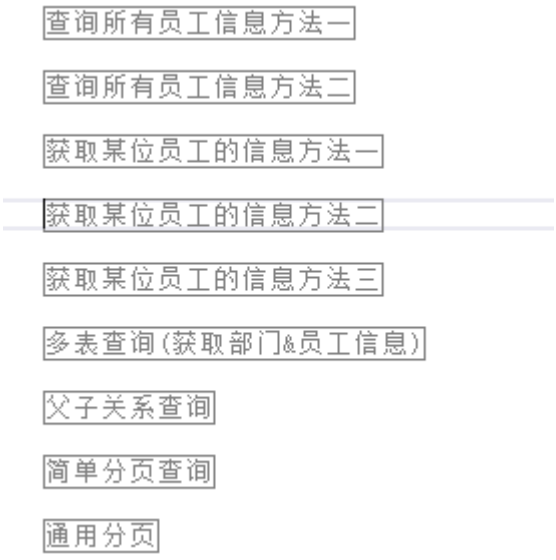
```
17.  get { return (int)Math.Ceiling(TotalCount/(double)PageSize); }
18.  }
19.  public IEnumerable<T> DataList { get; set;}
20.  }
```

(2-13)

```
1.   public int GetPaging()
2.   {
3.   ////实际开发可以独立出来处理//////////////
4.   var _ppaging = new p_PageList<t_employee>();
5.   _ppaging.Tables = "t_employee";
6.   _ppaging.OrderFields = "employeeid asc";
7.   ////////////////////////////////////
8.   var _dy = new DynamicParameters();
9.   _dy.Add("Tables", _ppaging.Tables);
10.  _dy.Add("OrderFields", _ppaging.OrderFields);
11.  _dy.Add("TotalCount",dbType:DbType.Int32,direction:
     ParameterDirection.Output);
12.  using (var _conn= Connection)
13.  {
14.  _conn.Open();
15.  _ppaging.DataList=_conn.Query<t_employee>("p_PageList", _dy,
     commandType: CommandType.StoredProcedure);
16.  }
17.  _ppaging.TotalCount = _dy.Get<int>("TotalCount");
18.  return _ppaging.PageCount;
19.  }
```

(2-14)

e) 更多查询方法你可以在附带项目中查看 (2-15)



(2-15)

f) 查询总结

Dapper 在查询方面还是比较方便的，唯一的不足就是不能直接的支持多表查询，尤其是多表联合查询，其实这也不是没有解决的办法，我建议的方案就是对多表联合查询，数据用单独的显示模型来承载，这样比较清晰。

Dapper 没有直接支持分页查询，这对我们来说，更能自由控制，自由发挥。

3. Dapper 的存储过程操作

Dapper 对 procedure 的支持是相当给力的，不管是参数赋值，还是输入输出参数及返回参数(整形)都支持的比较好，存储过程参数赋值主要是通过 DynamicParameters 传递的，同时可以设置参数的类型。（2-15）为演示用的 procedure，

```
1.  ALter PROCEDURE p_Procedemo
2.  (
3.  @employeeid INT=1,
4.  @displayname VARCHAR(20)=null OUTPUT

5.  )
6.  AS
7.  BEGIN
8.  DECLARE @mobile VARCHAR(20);
9.  SELECT @displayname=displayname,@mobile=mobile
    FROM t_employee WHERE employeeid=@employeeid;
10. select @mobile;
11. END
```

(2-15)

```
1   public Tuple<string,string> ProceDemo()
2   {
3   int employeeid = 1;
4   var _mobile = "";
5   var _dy = new DynamicParameters();
6   _dy.Add("employeeid", employeeid);
7   _dy.Add("displayname", string.Empty, dbType: DbType.String, direction:
    ParameterDirection.Output);
8   using (var _conn = Connection)
9   {
10  _conn.Open();
11  _mobile= _conn.Query<string>("p_Procedemo", _dy, commandType:
    CommandType.StoredProcedure).FirstOrDefault();
12  }
13  return Tuple.Create(_mobile, _dy.Get<string>("displayname"));
14  }
```

（2-16）

4. Dapper 的实体映射

在我们开发的过程中，经常为了方便显示或者特殊要求在实体中添加额外的属性，dapper 默认是按照实体名称<->字段名 双向映射，如果不让 Dapper 处理这些特殊的属性，我们需要额外处理，其实也很简单，就是添加一些特性。

a) 属性不编辑，用[Editable(false)]这个特性标记。默认是 true。

b) 类名到表名的映射，用[Table("tablename")]特性，tablename 对应物理数据表名称。

c) 主键映射，如果你的实体类中有 Id 属性，Dapper 会默认此属性为主键，否则要为作为主键的属性添加[Key]特性。

在员工实体类中添加一个 address 属性，并标记为[Editable(false)]，另外在为类加上[Table("t_employeecopy")]特性，结果为（4-1），为实体赋值（4-2）,最后我们还要 Copy 一份 t_employee 数据表结构，并把名称改为 t_employeecopy。这时执行 entitymapperdemo 方法，我们就会看到（4-3）效果，address 没有插入，（其实在 employeecopy 就是添加一列 address，数据也不会插入的，不行，你可以亲自测试下） 说明我们设置的特性是能正常工作。

```
1.  [Table("t_employeecopy")]
2.  public class t_employee
3.  {
4.  public t_employee()
5.  {

6.  }
7.  [Key]
8.  public int employeeid { get; set; }
9.  public string displayname { get; set; }
10. public string loginname { get; set; }
11. public string password { get; set; }
12. /// 1:男,0:女
13. public bool sex { get; set; }
14. public string mobile { get; set; }
15. public string email { get; set; }
16. /// 1:启用,0:禁用
17. public bool enable { get; set; }
18. [Editable(false)]
19. public string address { get; set;}
20. }
```

（4-1）

```
1.  public void entitymapperdemo()
2.  {
3.  var _employee = new t_employee {displayname = "Micro",email
    ="1441299@qq.com",loginname ="Micro",password =
    "66778899",mobile = "123456789",address = "我的地址，不能添
    加"};
4.  using (var _conn =Connection)
5.  {
```

```
6.      _conn.Open();
7.      _conn.Insert(_employee);
8.      }
9.       }
```

(4-2)

	employeeid	displayname	loginname	password	sex	mobile	email	enable
▶	1	Micro	Micro	66778899	False	123456789	1441299@qq.com	False
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

(4-3)

5. 结束语

a) 经过四天的努力，Dapper 这份教程终于告一段落，这只是暂时的告一段落，如果有新的发现，或者同行有更多的问题，我会不断更新，敬请期待。

b) 我创建了一个 NET 技术综合平台（http://www.dotneteye.com），会不定期的更新相关内容，敬请关注。

c) 未经作者本人同意，不得破坏文档的完整型,该教程包括（一份 PDF 文件，一个 Dapper 测试解决方案源码，一份数据库备份）。

d) 我的 NET 技术群（53135235），sharepoint 技术群 （3010435） 也欢迎您的加入 。

如果您有任何建议,请于我联系

邮箱: dotneteye@foxmail.com

QQ:1441299