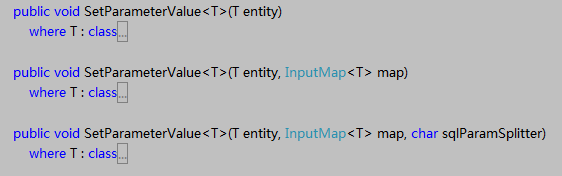
在Service端的解决方案里添加了ECCentral.Service.Utility.DataAccess工程，里面实现了数据访问的部分，抛弃了微软企业库，但暴露的类和方法，保持了和Oversea以前API一致，所以API的基本用法和以前Oversea的基本一样，另外SQL脚本仍然通过xml文件来配置，兼容以前Oversea的SQL脚本的xml配置文件。

另外提供了一些新的方法，里面直接集成了DataMapper，以方便使用。

在DataCommand类里下添加了如下的新方法 ：

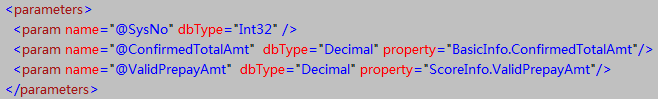
**一、针对SQL参数赋值的**：直接根据Entity属性来为SQL Parameter赋值；



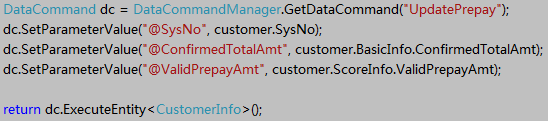
关于SQL Parameter的赋值，有以下几种方式：

1. 传统方式（之前Oversea框架提供的方式），直接在代码里用SetParameterValue(string paramName, Object val) 方法，例如：

配置文件里：

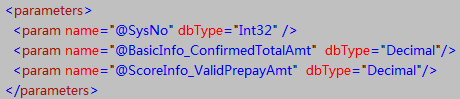


代码里：



1. 直接在SQL Parameter的参数名字上做映射，然后代码里使用新增的SetParameterValue<T>(T entity)及其的相关重载方法，例如：

配置文件里：



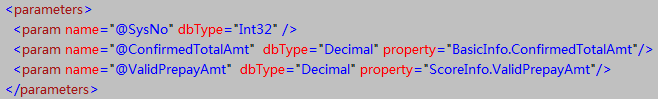
把SQL参数名字和所对应的实体属性映射起来，把多级属性之间的”. ”改为”\_”来分割多级属性；（所以如果遇到实体属性本身就包含有“\_”，那就不适合用这个方法，而改用1、3或4的方式；当然也可以使用其他的SQL参数名字合法的字符来作为多级属性分割符，只是在代码里要使用相应的重载方法，用方法参数来指明自定义的多级属性分隔符），然后在代码里：



方法SetParameterValue<T>会自动根据SQL参数的Name去寻找对应的属性，取出属性的值赋给该SQL参数；

1. 在SQL配置文件中的dataCommand的<parameters>的<param name="@MailAddress" dbType="String" size="200" property="Mail.Address" />里面的property来指定所映射的实体属性，然后代码里使用新增的SetParameterValue<T>(T entity)及其的相关重载方法，具体代码同第2点，例如：

配置文件：



代码里：



对于配置中param节点的property属性，有3个关键字：

（注：这3个关键字不区分大小写，所以[UserSysNo]和[usersysno]是一样的，都可以正常运行）

[UserSysNo] --- DataAccess框架自动赋值当前用户系统编号给SQL参数

[UserAcct]"   --- DataAccess框架自动赋值当前用户登录账号给SQL参数

[Now]"   --- DataAccess框架自动赋值当前应用服务器时间给SQL参数（一般情况下可以取消该入参，直接在SQL里GetDate()，只是SQL的GETDATE取到的是DB服务器的时间，如果应用服务器和DB服务器不在一个时区，但又想用应用服务器上的时间的话，就需要加上这个参数了）

Example：

<param name="@CreateUserSysNo" dbType="Int32" property="[UserSysNo]"  />

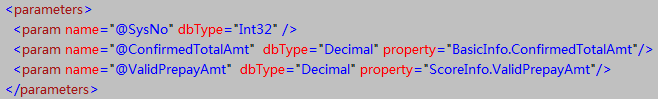
<param name="@EditUserSysNo" dbType="Int32" property="[usersysno]"  />

<param name="@CreateDate" dbType="DateTime" property="[Now]"  />

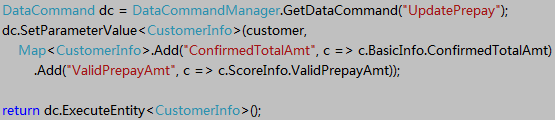
<param name="@CreateUserAcct" dbType="String" property="[UserAcct]"  />

1. 可以结合2、3的配置与代码里的映射配置一起使用：

配置文件：



代码里：



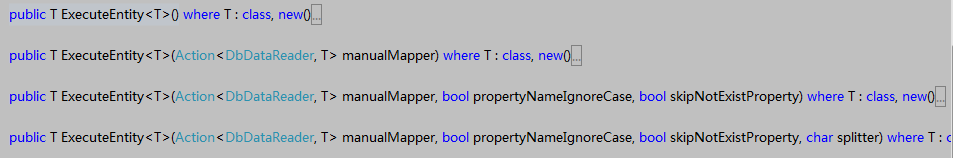
此时会优先使用InputMap<T>参数里所设定的映射关系去匹配实体属性；如果没有为该SQL Parameter指定InputMap<T>，则使用配置文件里param节点的property的值去匹配实体属性；如果该SQL Parameter在配置文件里也没有配置对应的property，那么最后将直接使用该SQL Parameter的Name去匹配；如果还是没有匹配上，那么该SQL Parameter将不会被赋值；

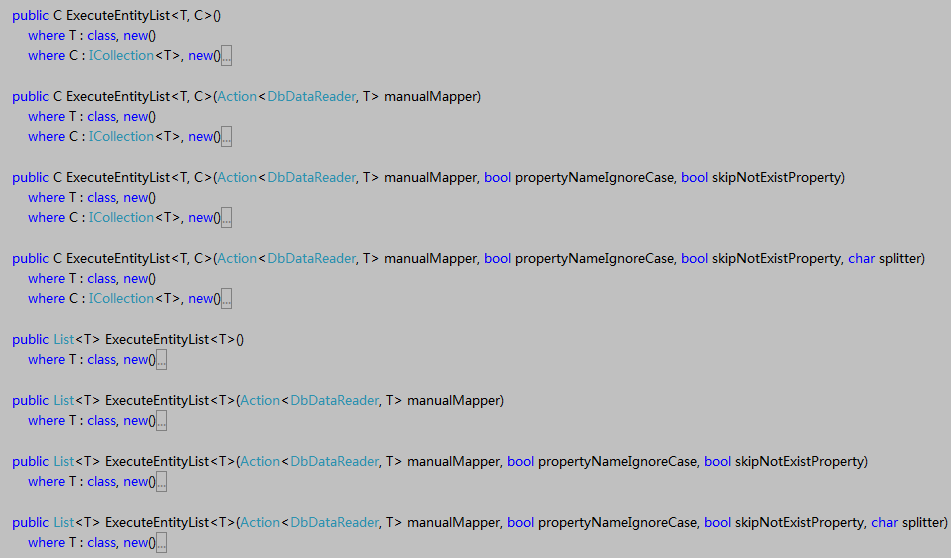
1. 最后，1、2、3、4多种方式可以混用，SetParameterValue也就是给参数赋值，所以参数的最终值会以最后执行的赋值动作为准。

**二、针对返回值映射到实体的**：

ExecuteEntity是用来将查询的第一行记录返回为指定类型的实体对象的；

ExecuteEntiltyList则是用来将所有的查询结果都返回为指定类型的实体对象，每一行记录将是一个实体对象，最后返回的是实体对象的集合；



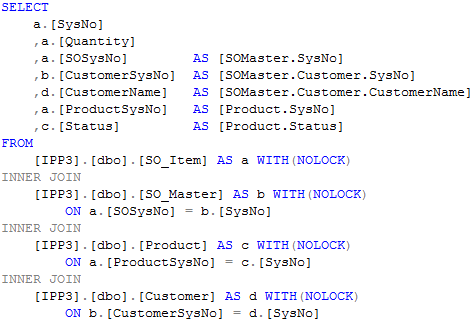


而对于返回的DbDataReader里的数据和实体属性的映射关系，则是通过DbDataReader的列的名称（即SQL查询的字段的名称）与实体的属性进行映射的，比如下面的例子：

假设有SOItemEntity 、SOMasterEntity、CustomerEntity、ProductEntity的实体类型，其关系如下：



那么可以定义如下的SQL脚本，通过SELECT的字段的名字（或别名）来映射到对应的属性上：



最后使用ExecuteEntity<SOItemEntity>或ExecuteEntityList<SOItemEntity>方法来直接获得SOItemEntity实体的对象实例或对象的集合；（如果SQL字段的名称，没有找到对应的实体属性，默认情况下则该字段的数据就会被忽略掉，不会填充到实体里，但可以通过一个重载方法的参数 bool skipNotExistProperty 来控制这种情况下是自动忽略还是抛出异常；）

同时还提供了一个重载方法，可以传入一个自定义的Action<DbDataReader, T>的回调方法，该回调方法的第一个参数是数据源DataReader，第二参数是即将作为返回值的数据实体对象，在这个回调方法里可以做人工的更灵活的mapping赋值动作。

如：

ExecuteEntity<SOItemEntity>((r, e) => {

e.Rebate = Convert.ToDecimal(r["CustomRebate"]);

e.InTime = (DateTime)r["CreateTime"];

……

});

ExecuteEntity<T>方法会先执行自动的映射赋值，然后再执行参数Action<DbDataReader, T>的回调方法。