Noebe User Manual Script

www.pixysoft.net

Content

License

张辰
Reborn_zhang@hotmail.com

Introduction

Noebe 是 ERP 系统的持久层，通过 xml 配置，提供以下功能。

- 提供面向关系数据库的持久层操作
- 根据 DataTable 的 Schema 生成 CRUD 操作
- 自定义 SQL 操作
- 配置文件定义 SQL 操作
- 分布式数据库一致性操作
- 事务操作
- 主键自动填充、生成
- 基于 MRU 策略的缓存操作

允许完全代码配置、xml 配置两种方式初始化持久层 Noebe。配置文件默认读取路径与本项目运行路径相同

DemoCode 代码范例

Querion 查询操作

```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");
DatabaseManager.Instance.Querion.Select(table);
```

Session 增、删、改操作

```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");
DataRow row = table.NewRow();
row["Column1"] = "Update Me!";
table.Rows.Add(row);
Session session = DatabaseManager.Instance.Session.Update(table);
```

Extension 自定义查询操作

```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");
Dictionary<string, object> paraList = new Dictionary<string, object>();
paraList.Add("COLUMN1", "SELECT ME!");
Extension extension = DatabaseManager.Instance.Extension.SelectBySQL(table,
"SELECT * FROM TESTTABLE WHERE COLUMN1=:COLUMN1", paraList);
```

Transaction 增、删、改事务操作

```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");

DataRow row = table.NewRow();
row["Column1"] = "Update Me!";
table.Rows.Add(row);

Transaction transaction = DatabaseManager.Instance.Transaction;
transaction.BeginTransaction();
transaction.Update(table);
transaction.Commit();
```

SyncSession 数据库并发同步增、删、改操作

```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");
DataRow row = table.NewRow();
row["Column1"] = "Update Me!";
table.Rows.Add(row);
SyncSession session = DatabaseManager.Instance.SyncSession.Update(table);
```

SyncTransaction 数据库并发同步增、删、改事务操作

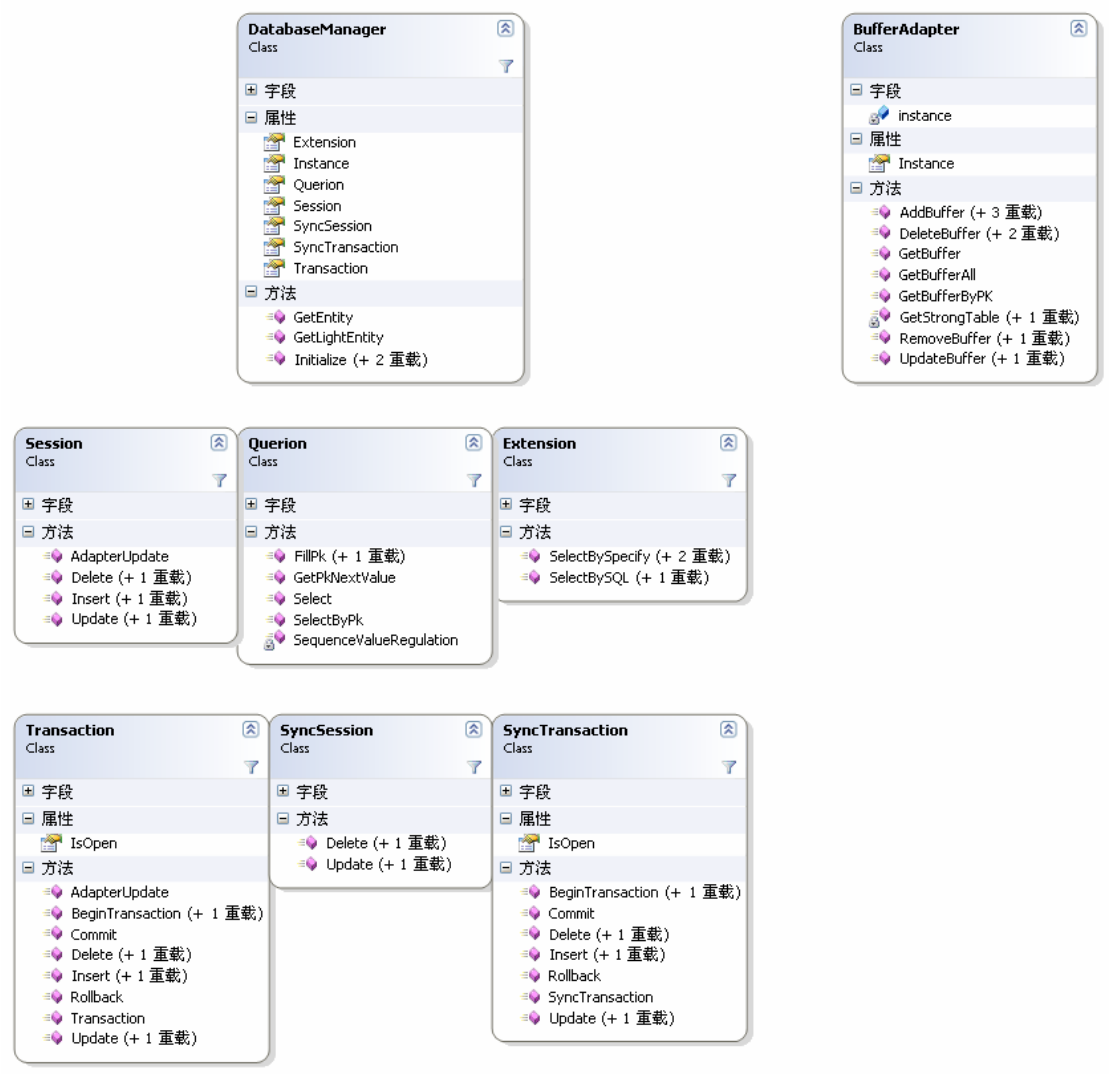
```
DataTable table = DatabaseManager.Instance.GetEntity("TestTable");

DataRow row = table.NewRow();
row["Column1"] = "Update Me!";
```

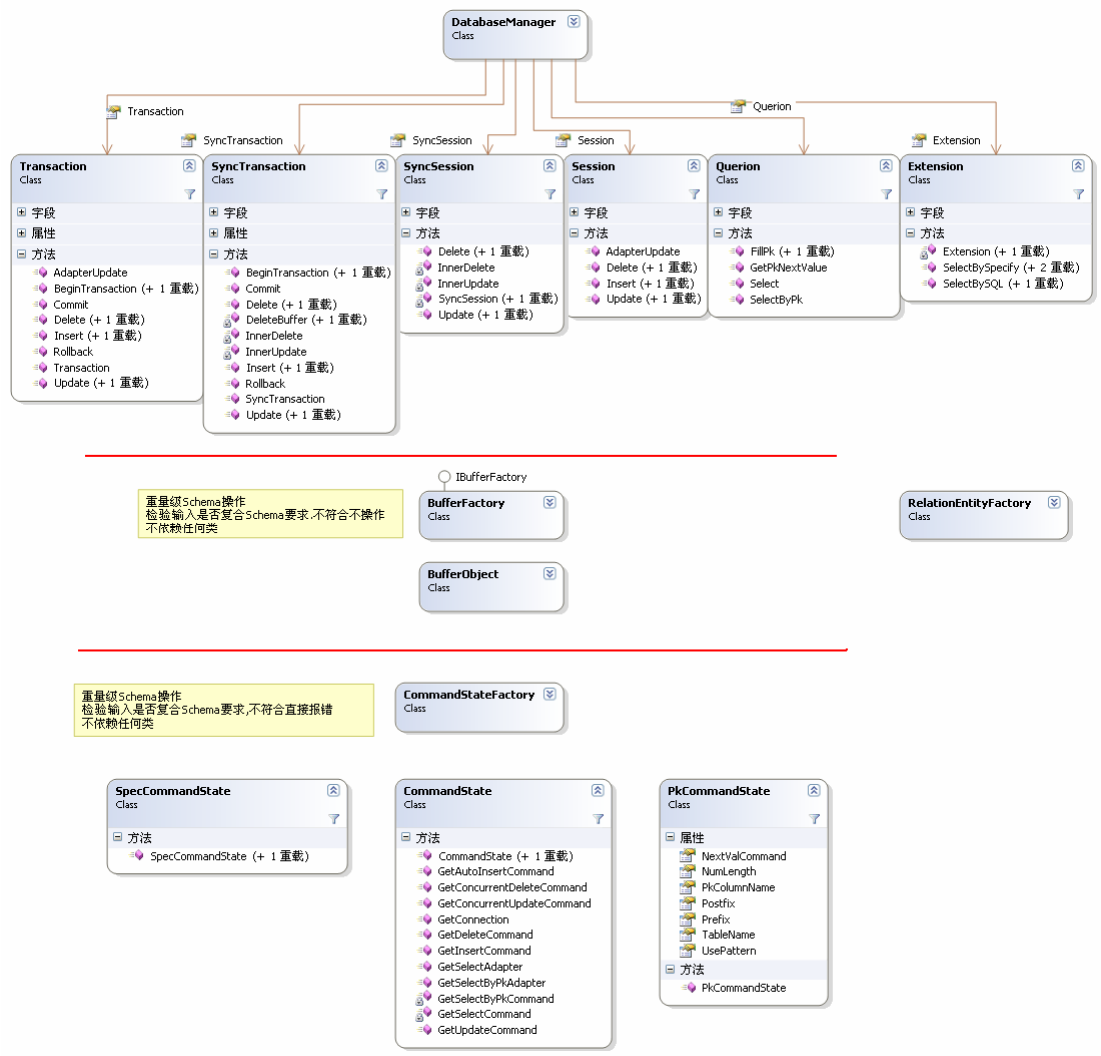
```
table.Rows.Add(row);

SyncTransaction transaction = DatabaseManager.Instance.SyncTransaction;
transaction.BeginTransaction();
transaction.Update(table);
transaction.Commit();
```

API 接口描述


DatabaseManager
Class
字段
属性
Extension
Instance
Querion
Session
SyncSession
SyncTransaction
Transaction
方法
GetEntity
GetLightEntity
Initialize (+ 2 重载)
BufferAdapter
Class
字段
instance
属性
Instance
方法
AddBuffer (+ 3 重载)
DeleteBuffer (+ 2 重载)
GetBuffer
GetBufferAll
GetBufferByPK
GetStrongTable (+ 1 重载)
RemoveBuffer (+ 1 重载)
UpdateBuffer (+ 1 重载)
Session
Class
字段
方法
AdapterUpdate
Delete (+ 1 重载)
Insert (+ 1 重载)
Update (+ 1 重载)
Querion
Class
字段
方法
FillPk (+ 1 重载)
GetPkNextValue
Select
SelectByPk
SequenceValueRegulation
Extension
Class
字段
方法
SelectBySpecify (+ 2 重载)
SelectBySQL (+ 1 重载)
Transaction
Class
字段
属性
IsOpen
方法
AdapterUpdate
BeginTransaction (+ 1 重载)
Commit
Delete (+ 1 重载)
Insert (+ 1 重载)
Rollback
Transaction
Update (+ 1 重载)
SyncSession
Class
字段
方法
Delete (+ 1 重载)
Update (+ 1 重载)
SyncTransaction
Class
字段
属性
IsOpen
方法
BeginTransaction (+ 1 重载)
Commit
Delete (+ 1 重载)
Insert (+ 1 重载)
Rollback
SyncTransaction
Update (+ 1 重载)

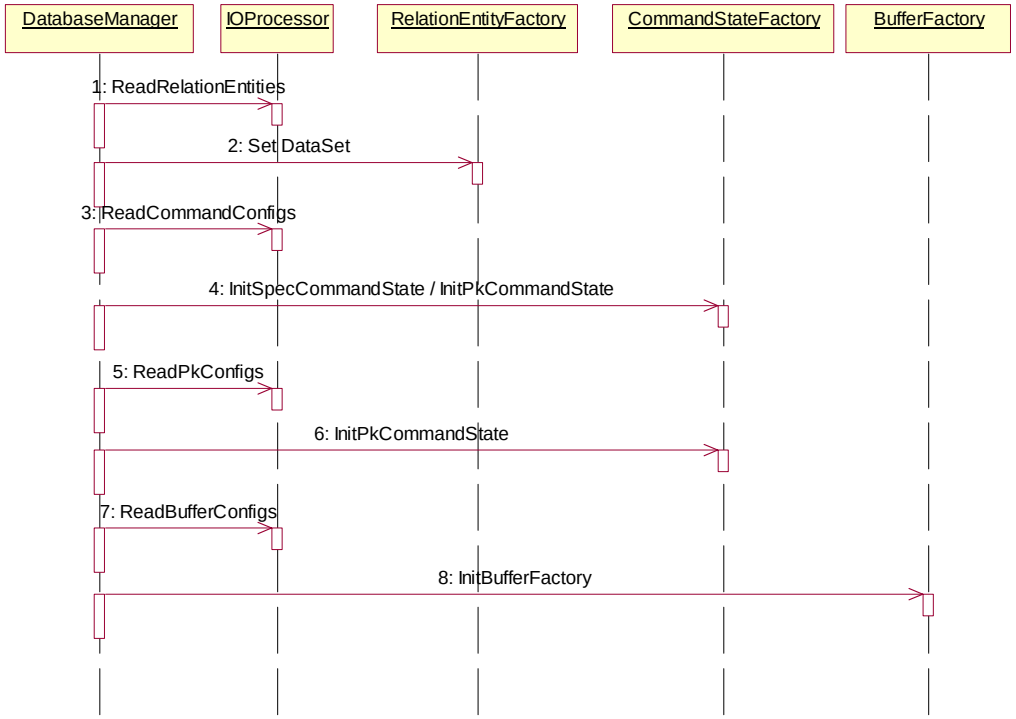
Architecture 架构



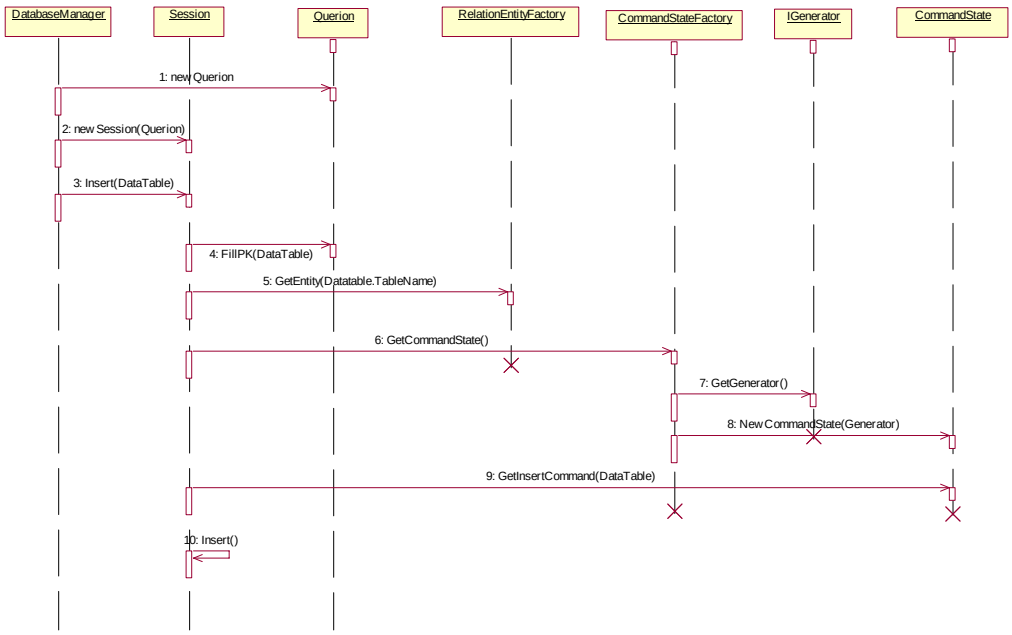
Sequence Diagrams 时序图

初始化

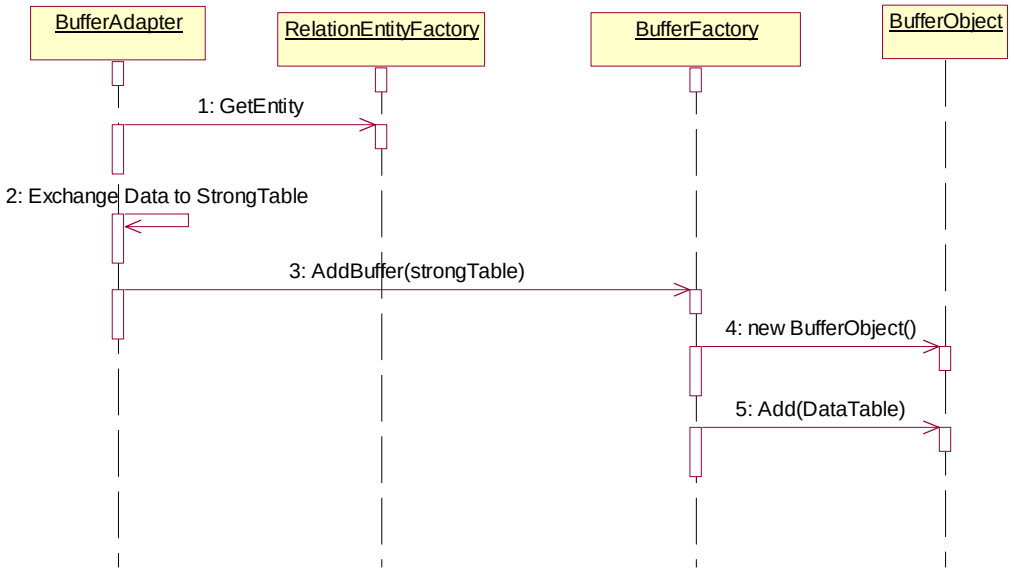
- 读取配置文件（或者代码加入配置），初始化持久层



基本的数据库操作



缓存原理



Configurations 配置文件详解

ManagerConfiguration

文件名：
Noebe. config

实例：

```
<ManagerConfiguration>
  <ConnectionString>Data Source=xjh;Persist Security Info=True;User
ID=zhangchen;Password=zhangchen;Unicode=True</ConnectionString>
  <DatabaseType>Oracle</DatabaseType>
  <GlobalCreateBuffer>false</GlobalCreateBuffer>
  <GlobalSynchronizeBuffer>false</GlobalSynchronizeBuffer>
</ManagerConfiguration>
```

解释：

ConnectionString	数据库连接字符串
DatabaseType	数据库类型，包括： Oracle OleDb
GlobalCreateBuffer	是否建立全局缓存，只有打开这个选项，缓存设置才能生效。
GlobalSynchronizeBuffer	是否实时更新缓存，当数据库执行 update，delete 的时候实时更新缓存

BufferConfiguration

文件名：
Noebe.Buffer.*.config

实例：

```
<ArrayOfBufferConfiguration>
  <BufferConfiguration>
    <TableName>HSY_WAREHOUSE</TableName>
    <PreFilled>true</PreFilled>
  </BufferConfiguration>
  <BufferConfiguration>
    <TableName>HSY_DEPARTMENT</TableName>
    <PreFilled>true</PreFilled>
    <BufferCapacity>60</BufferCapacity>
  </BufferConfiguration>
</ArrayOfBufferConfiguration>
```

解释：

TableName	必选，需要缓存的表名
PreFilled	是否预缓存所有数据到内存，用于提高查询速度。
BufferCapacity	缓存容量，最大值为 200
CreateBuffer	是否建立缓存，如果全局建立的时候，此选项可以单独控制
SynchronizeBuffer	是否同步缓存，如果全局同步的时候，此选项可以单独控制

PrimaryKeyConfiguration

文件名：
Noebe.PrimaryKey.*.config

实例：

```
<ArrayOfPrimaryKeyConfiguration>
  <PrimaryKeyConfiguration>
    <TableName>HST_ITEMMOVELOG</TableName>
    <PrimaryKeyName>BILLID</PrimaryKeyName>
    <PrimaryKeyCreateType>Sequence</PrimaryKeyCreateType>
    <SequenceName>HST_ITEMMOVELOG_S</SequenceName>
    <UsePattern>true</UsePattern>
    <NumLength>8</NumLength>
  </PrimaryKeyConfiguration>
</ArrayOfPrimaryKeyConfiguration>
```

解释：

TableName	必选，需要设置主键读取的表名
PrimaryKeyName	必选，表中需要填写的主键名

PrimaryKeyCreateType	必选，主键创建类型，包括：
	//手动添加
	Manual = 0,【默认】
	//使用Sql参数求最大值
	SqlSum = 1,
	//使用数据库自动添加
	Triger = 2,
	//使用数据库序列器生成
	Sequence = 3

SequenceName	数据库中序列器名，当 PrimaryKeyCreateType 为 Sequence 的时候必选。
UsePattern	是否使用模式，主要对生成的主键进行字符串处理，包括

	前缀和后缀
Prefix	主键前缀，当 UsePattern 打开的时候生效
Postfix	主键后缀，当 UsePattern 打开的时候生效
NumLength	主键数字长度，默认为数据库返回的长度

CommandConfiguration

文件名：
Noebe.Command.*.config

实例：

```
<ArrayOfCommandConfiguration>
  <CommandConfiguration>
    <Identifier>HCT_COUNTER__STORECODE</Identifier>
    <SQL>
      SELECT * FROM HCT_COUNTER WHERE STORECODE=:STORECODE
    </SQL>
    <CommandParameters>
      <CommandParameter Name="STORECODE" DataType="String" Direction="Input"
IsNullable="true" />
    </CommandParameters>
  </CommandConfiguration>
  <CommandConfiguration>
    <Identifier>ITM_STOREITEM__COUNTERCODE</Identifier>
    <SQL>SELECT * FROM ITM_ITEM WHERE ITEMCODE IN (SELECT ITEMCODE FROM
ITM_STOREITEM WHERE COUNTERCODE = :COUNTERCODE )      </SQL>
    <CommandParameters>
      <CommandParameter Name="COUNTERCODE" DataType="Char" Direction="Input"
IsNullable="true" />
    </CommandParameters>
  </CommandConfiguration>
</ArrayOfCommandConfiguration>
```

解释：CommandConfiguration:

Identifier	自定义数据操作全局 ID
SQL	自定义 SQL
Memo	备注
CommandParameter	SQL 的参数

解释：CommandParameter:

Name	参数名，直接对应程序中的调用
ChName	参数中文备注
SrcColumnName	参数对应与数据表的字段名
DataType	数据类型，包括： String【默认】 char int long float double boolean byte object byte[] DateTime

Direction	参数传递方向，包括：
	// 参数是输入参数。
	Input = 1【默认】
	//参数是输出参数。
	Output = 2,
	//参数既能输入，也能输出。
	InputOutput = 3,
	// 参数表示诸如存储过程、内置函数或用户定义函数之类的操作的返回值。
	ReturnValue = 6,

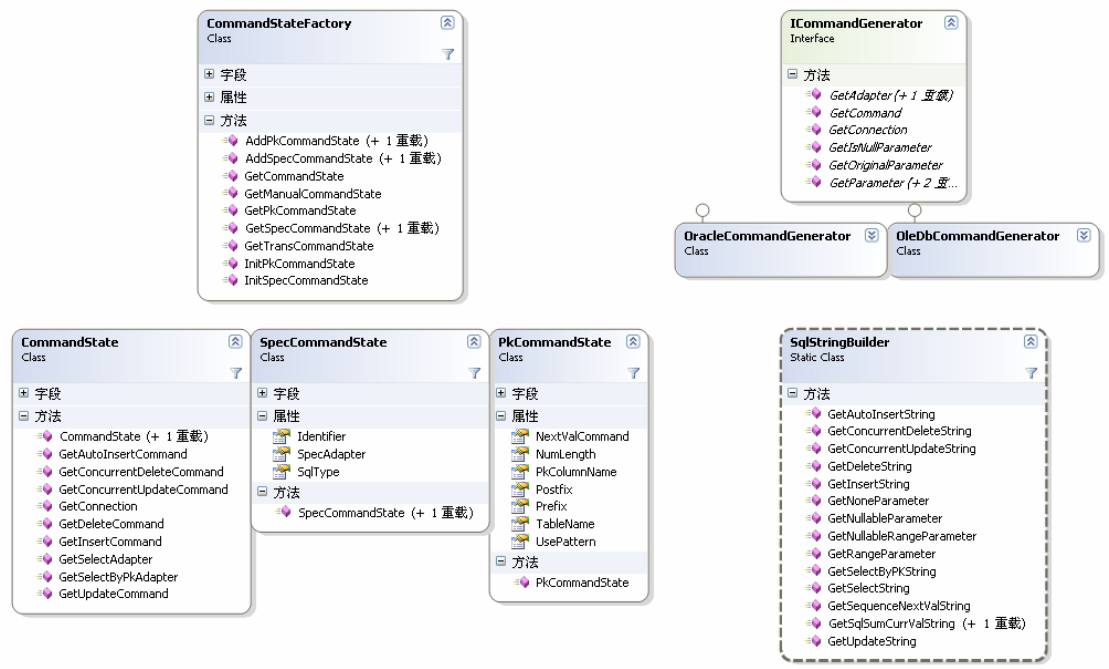
IsNullable	参数是否可以允许空，需要与系统生成的 SQL 配合使用

CommandStateFactory

一句话概述

Command 是整个持久层的核心，包含了根据 DataTable 的 Schema 生成 CRUD 操作；自定义 SQL 操作；数据库主键的操作（配合 CRUD），为外部类（Session）等提供需要的 DbCommand 等。

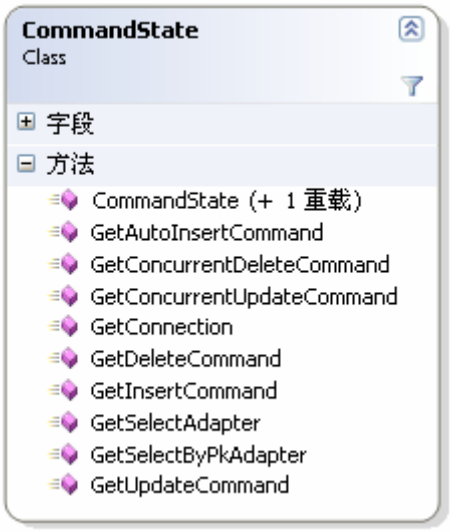
UML



注意事项

- 内部类，外部不可见
- 数据库结构默认只有一个主键，而且每张表必须有一个主键

CommandState



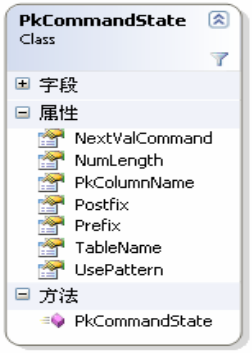
- 一句话：根据 DataTable 的 Schema 生成 CRUD 对应的 SQL 操作
- 依赖 ICommandGenerator 得到需要的 ACommand / ADataAdapter 等
- 依赖 SqlStringBuilder 得到需要的 Sql String
- 具体提供的操作包括：
  - AutoInsert: 忽略主键插入，用于数据库自动生成主键的时候使用
  - ConcurrentDelete: 同步删除
  - ConCurrentUpdate: 同步修改
  - GetConnection: 取得连接，用于后面的事务操作
  - GetDeleteCommand: 删除
  - GetInsertCommand: 插入
  - GetSelectAdapter: 全选
  - GetSelectByPkAdapter: 根据主键选择
  - GetUpdateCommand: 更新

SpecCommandState

- 一句话：提供代码自定义 Sql / 配置文件自定义 Sql 操作
- 代码自定义操作：传递 sql string
- 配置文件自定义操作：依赖 CommandConfiguration
- 代码自定义操作不适合负责的操作，例如日期等（默认所有 Sql 输入参数为 String）
- 自定义操作只支持 Select，其他操作使用 CommandState
- Identifier: 配置文件定义 Sql 的时候，对应 Sql 的标识符

- SpecAdapter: 操作的 Adapter
- SqlType: 取得 Sql 的类型（Insert / Update / Delete / Select）

PkCommandState

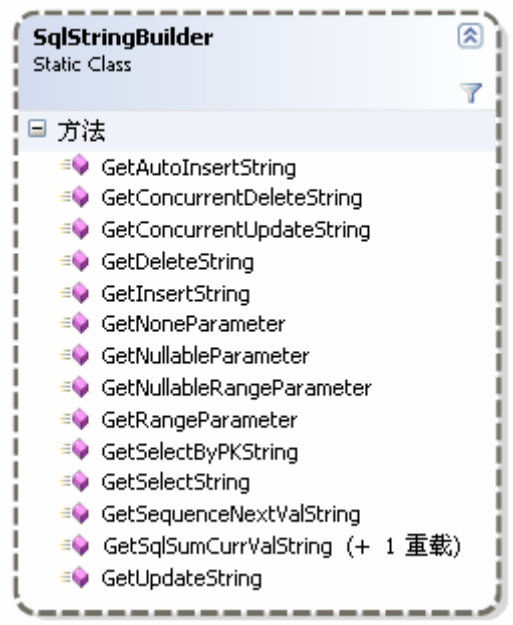


- 一句话：根据配置文件要求生成表主键
- 依赖 PrimaryKeyConfiguration
- NextValCommand：获取下一主键的 Command
- 其余的就是配置文件的设置

ICommandGenerator

- 提供需要的 DbCommand / DbDataAdapter / DbConnection / DbParameter
- 支持 Oracle / OleDb 类型

SqlStringBuilder



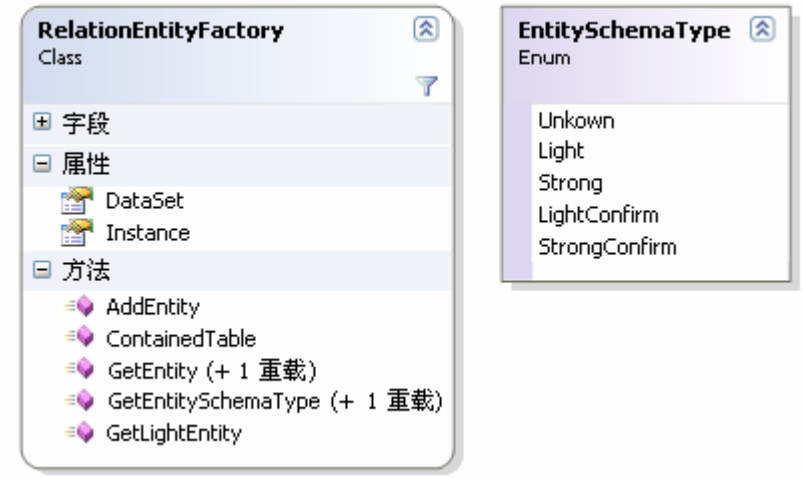
- 提供需要的 Sql String

RelationEntityFactory

一句话概述

RelationEntityFactory 保存了 DataSet，提供了强 Schema、弱 Schema

UML



RelationEntityFactory

- 强 Schema：就是包含主键等约束关系的数据表 Schema。
- 弱 Schema：仅包含字段的数据表 Schema.
- 整个持久层依赖强 Schema，但是对外需要弱 Schema，为了扩展可用性，通过这个层可以根据弱 Schema 生成强 Schema

EntitySchemaType

- 判断当前 DataTable 的 Schema 类型
- Light：弱，RelationEntityFactory 不包含
- String：强，RelationEntityFactory 不包含
- LightConfirm：弱，RelationEntityFactory 包含
- StringConfirm：强，RelationEntityFactory 包含

BufferFactory

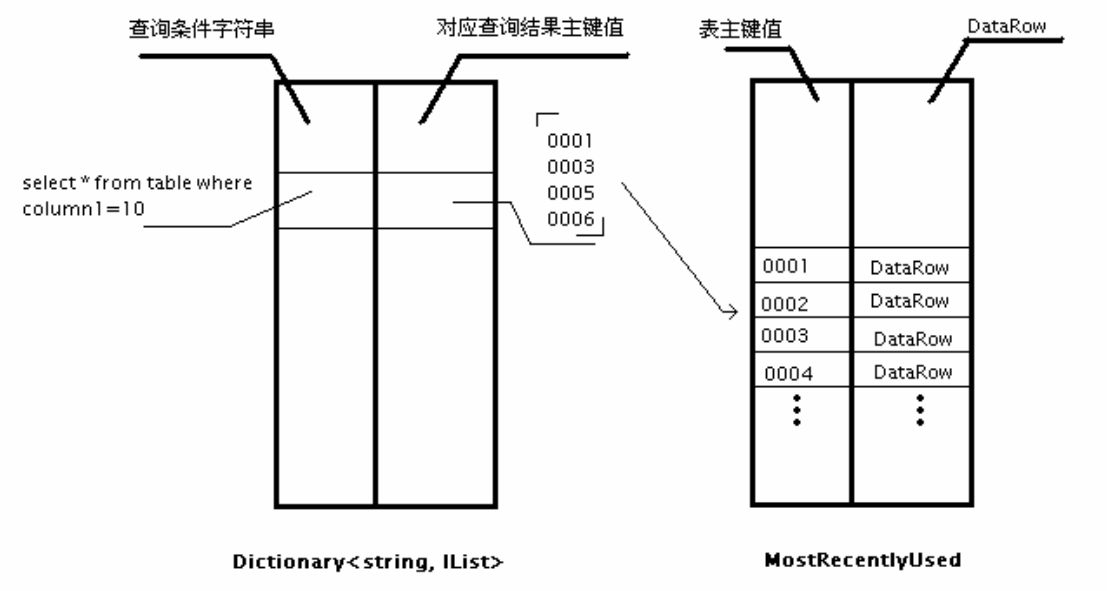
一句话概述

为数据库操作提供缓存；全局配置 GlobalCreateBuffer 打开的时候启动缓存；全局配置 GlobalSynchronizeBuffer 打开的时候同步缓存；个体配置可以指定某表的具体缓存操作

UML

BufferObject

- 对某张表进行缓存
- 使用双层数据结构，查询条件对应主键值，主键值再得到 DataRow



- 使用 MRU 算法进行数据淘汰
- 使用时，查询条件存放查询 string 的值，而不是一个带参数的 sql
- 存在缺陷：由于首层保存查询条件对应的主键值列表，如果当主键值不再满足查询条件的时候，缓存不区分。例如：查询当前售价＝10 的商品，得到列表保存在缓存；当其中一个商品售价变为 11，缓存中的数据变化了，但是首层并没有去除这个商品，所以缓存仍然错误。
- 一句话：如果表被修改字段包含在查询条件字段里面，由于查询条件字段不变，缓存失效
- 支持：添加、删除、修改缓存
- 默认缓存容量为 200，最大值为 400

BufferFactory

- 缓存操作，需要强 schema 支持
- 内部类

BufferAdapter

- 缓存适配器，主要将外部弱 schema 转化为强 schema，提高扩展性
- 外部类

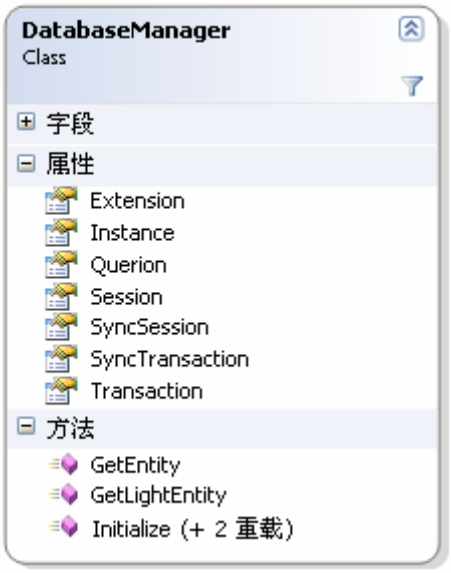
DatabaseManager

一句话概述

整个持久层的操作类，提供了基本 CRUD 操作、自定义 SQL 操作、配置文件 SQL 操作、自动填充主键、获取主键、缓存、数据库同步操作、事务操作

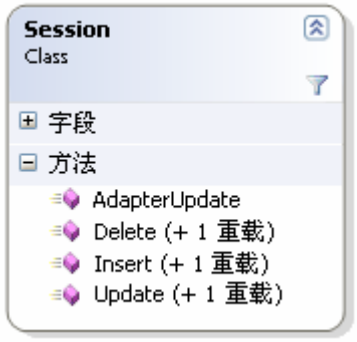
UML

DatabaseManager



- 总控制器，提供各种操作的类、初始化持久层、提供持久对象

Session



- 基本 CRUD 操作
- 插入的时候自动填写主键（需要与 Querion 配合）
- 支持 AdapterUpdate 操作，简化了数据库操作流程（需要与 Querion 配合）

Querion

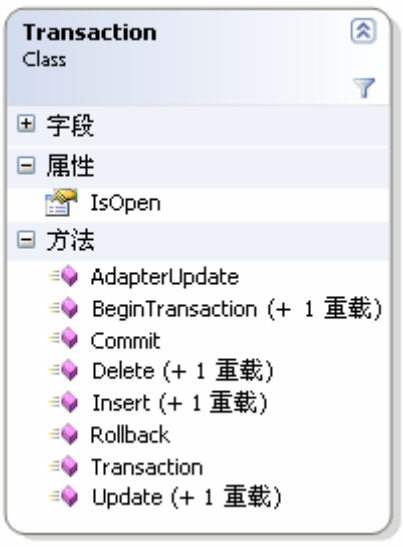
- 数据库查询操作
- 包括填写主键、取得主键下一值
- 全选查询
- 主键查询

Extension

- 自定义 Sql 操作、配置文件操作
- 注意，如果 sql 里面使用了模式，如下，则需要与 SqlParameterBuilder 配合建立查询条件值

```
        /// HELLO = :HELLO
        None = -1,
        /// ((:NULL_HELLO = 1) OR (HELLO = :HELLO))
        Nullable = 1,
        /// (HELLO >=:HELLOFROM AND HELLO<=:HELLOTO)
        Range = 2,
        /// (( NULL_HELLOFROM = 1 ) OR ( HELLO >= :HELLOFROM ) AND ( NULL_HELLOTO = 1)  OR ( HELLO <= :HELLOTO ))
        NullableRange = 3
```

Transaction

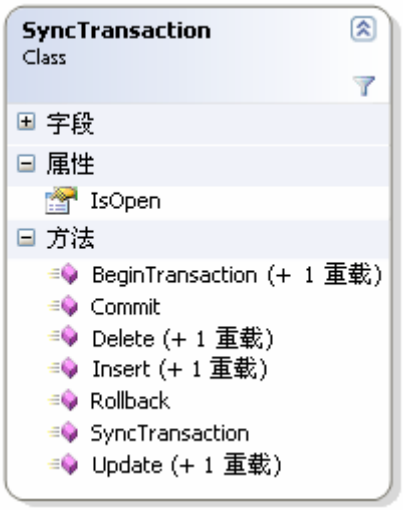


- 事务操作

SyncSession

- 与数据库同步的操作，保证了分布式操作的一致性、同步性
- 需要缓存的配合
- 原理：当用户选择了数据的时候，旧数据会被保存在缓存；用户修改了数据再反馈到数据库的时候，生成的 Sql 利用缓存的旧数据约束，限制了分布式数据库操作的不一致性。
- 具体看文献：如何使用 sql 控制分布式数据库一致性

SyncTransaction



- 与 SyncSession 类似，支持事务
- Insert 不属于同步操作，存在只是为了提高事务的扩展性

In Future

- 提供面向对象数据库操作
- 提高缓存策略，实现完全透明化缓存
- 支持线程操作