# 比特币测量

## 一、概述

比特币区块链中存储有自比特币诞生以来，所有的交易信息。那么在以交易及账户构建的交易图谱中，存在着怎样的关系呢？如果是以账户为节点，交易为边，那么构成的图具有怎样的特点？如果是以交易为节点，账户为边，那么构成的图又具有怎样的特点？为了回答上述问题，我们开启了比特币测量项目。

## 二、开发环境

1. 操作系统：Windows 7
2. 编程语言：C#
3. 开发IDE: Visual Studio 2015
4. 数据库：SQL Server 2014企业版（如果是Express版本，它存在10G容量的限制）
5. 存储空间预留：500G

## 三、开发环境准备

1. SQL Server 2014企业版:

* 下载地址： 链接：https://pan.baidu.com/s/1pL1neoP9PkoGWZp1W1\_HRQ 密码：uf9w
* 安装： <https://blog.csdn.net/yym373872996/article/details/50643116>
* 使用说明：在安装过程中有个实例名<InstanceName>，则在本地连接使用时，数据库服务器名称为：localhost\<InstanceName>
* 项目准备：建立一个数据库实例，比如 BITCOIN。

1. 比特币区块链：

* 安装及下载比特币全节点：[比特币全节点安装教程](https://blockchain.liangnotes.com/ch01-bitcoin/examples/bitcoind-full-node.html)
* 比特币区块链所在文件位置：<安装路径>/.bitcoin/blocks/\* 在该目录下存在着两种.dat文件，分别是recvXXXXX.dat及blkXXXXX.dat，本项目需要的是blkXXXXX.dat文件，将其拷贝至所需文件夹中。目前使用的文件为blk00000.dat~blk01200.dat
* 注意：安装好比特币全节点客户端后，需要启动，等待其自动同步。

1. 软件开发：

* 参考项目：[BitcoinBlockchain](https://github.com/ladimolnar/BitcoinBlockchain)、[BitcoinDatabaseGenerator](https://github.com/ladimolnar/BitcoinDatabaseGenerator)。其中[BitcoinBlockchain](https://github.com/ladimolnar/BitcoinBlockchain)是[BitcoinDatabaseGenerator](https://github.com/ladimolnar/BitcoinDatabaseGenerator)运行需要的类库。
* 修改后的项目代码：<https://github.com/VictorJiangXin/Graduation_Class_Design>

## 四、背景知识

### 1. 比特币区块链相关

* 区块链存储路径及名称

路径： <安装路径>/.bitcoin/blocks/

文件名： blkxxxxx.dat 五位表示数字编号

* 区块结构（小端模式）

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| Magic | 4B | 0XD9B4BEF9，  区块边界 |
| BlockLength | 4B | 区块大小 |
| BlockHeader | 80B | 区块头 |
| TransactionsNum | 可变字节（1-9B） | 区块中，交易的数量 |
| Transactions | 可变字节 | 交易 |

* 区块头结构（小端模式）

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| Version | 4B | 版本号 |
| PreviousBlockHash | 32B | 前一区块哈希值 |
| MerkleRootHash | 32B | 默克尔树根 |
| BlockTimeStamp | 4B | 时间戳 |
| BlockTargetDifficulty | 4B | 难度目标 |
| BlockNounce | 4B | 计数值 |

* 交易结构（小端模式）

在2017年8月后，比特币区块链进行了一次软分叉，采用了隔离见证，因此交易结构存在两种，旧版本的交易结构、采用了隔离见证的交易结构。

1）非隔离见证版

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| Version | 4B | 交易版本号 |
| InputsCount | 可变（1-9B） | 交易输入数量 |
| Inputs | 可变 | 交易输入 |
| OutputsCount | 可变（1-9B） | 交易输出数量 |
| Outputs | 可变 | 交易输出 |
| LockTime | 4B | 交易锁定时间（大部分为0） |

2）隔离见证版

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| Version | 4B | 交易版本号 |
| Marker | 1B | 0x00 隔离见证标志 |
| Flag | 1B | 0x01 |
| InputsCount | 可变（1-9B） | 交易输入数量 |
| Inputs | 可变 | 交易输入 |
| OutputsCount | 可变（1-9B） | 交易输出数量 |
| Outputs | 可变 | 交易输出 |
| Witness | 可变 | 隔离见证域 |
| LockTime | 4B | 交易锁定时间（大部分为0） |

* 交易输入结构（小端模式）

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| SourceTXHash | 32B | 源交易哈希 |
| SourceOutputIndex | 4B | 源交易输出索引 |
| ScriptLen | 可变(1-9B) | 解锁脚本长度 |
| Script | 可变 | 解锁脚本 |

* 交易输入结构（Coinbase）

|  |  |  |
| --- | --- | --- |
| 结构 | 大小 | 备注 |
| SourceTXHash | 32B | 0 |
| SourceOutputIndex | 4B | 0xffffffff |
| ScriptLen | 可变(1-9B) | 解锁脚本长度 |
| Script | 可变 | 解锁脚本 |

在后期的Coinbase中，Script前部分为区块的高度，为了获取区块的高度也是先读取高度数据区的大小，其为可变字节（1-9B），然后读取该长度的数据域，获取区块的高度。

* 比特币脚本与地址

参考博客：[比特币脚本与地址](https://furrybear.github.io/%E7%AC%94%E8%AE%B0/%E5%8C%BA%E5%9D%97%E9%93%BE/%E6%AF%94%E7%89%B9%E5%B8%81%EF%BC%9A%E8%84%9A%E6%9C%AC%E4%B8%8E%E5%9C%B0%E5%9D%80/)

### 2. ADO.NET相关

参考：<http://www.cnblogs.com/best/p/7714500.html>

简单介绍（具体看参考的博客，十分详细）：

ADO.NET是.NET框架中的重要组件，主要用于完成C#应用程序访问数据库。其组成如下所示：

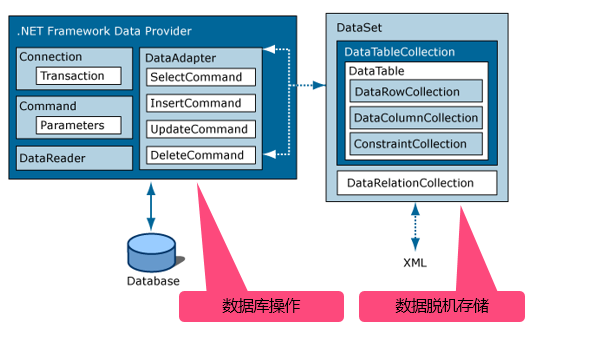


图 1 ADO.NET组成

本项目中涉及的其他相关知识：

* DataSet、DataTable、DataRow类：

介绍：DataSet是ADO.NET的中心概念，可以把DataSet当成内存中的数据库，DataSet是不依赖于数据库的独立数据集。DataSet所有数据都加载在内存上执行，可以提高数据访问速度，提高硬盘数据的安全性，极大的改善了程序运行的速度和稳定性。DataTable是DataSet中的一个表，DataRow是DataTable中的一行数据。三者关系如下图所示：

DataSet

DataTable

功能:

1.通过DataAdapter，使用数据源中的数据生成和填充DataSet中每个DataTable。

2.通过添加、更新或删除DataRow对象更改单个DataTable对象的数据。

* SqlBulkCopy类

介绍：将DataTable或DataRow中的数据直接复制到数据库中指定的表中。

核心方法：SqlBulkCopy.WriteToServer() 将所有行从数据源复制到SqlBulkCopy对象的DestinationTableName属性指定的目标表中。

注意事项：

1).表名、列名对大小写敏感。

2).DataTable的架构必须和目标表保持一致。

3).DataTable中Int类型的字段如果为Null，必须对其赋值为DBNull.Value。

4).SqlBulkCopy遇到重复字段只会直接报错，然后默认回滚所有复制操作即复制到一般出错了，那整个前面复制的也放弃掉。

## 五、项目架构

### 整体框架



图 2 系统程序框架

整个项目分为三部分，区块链文件解析、数据库生成及交易图谱的生成。BitcoinBlockchain工程主要是对比特币区块链文件进行解析，提取出区块链的所有区块信息。BitcoinDatabaseGenerator工程主要是生成数据库，调用BitcoinBlockchain的动态链接库，对比特币区块链文件进行解析，并将数据存入数据库。BitcoinTransactionGraphicGenerator主要就是利用存储在数据库中的区块链信息，构建交易图谱，并且存储在数据库中。

### 2. BitcoinBlockchain项目

**功能介绍：**解析区块链存储文件，返回区块Block类。

**测试运行：**项目中有个用于测试的BitcoinBlockchainSample工程，用于测试BitcoinBlockchain.dll的运行。编译完成后，从cmd命令行进入<工程路径>\Sources\BitcoinBlockchainSample\bin\Debug\，然后输入 BitcoinBlockchainSample.exe <.dat文件所在的路径> (可选<某个.dat文件名>) 如果有确定某个.dat文件，则工程从该文件开始向后解析文件，否则工程从blk00000.dat文件开始解析，直至将路径下文件解析完毕。

**程序框架介绍：**



图 3 BitcoinBlockchain程序框架

BitcoinBlockchain主要由四类文件构成（实际代码的类的分类并不是按照上图所示，为了便于解析工程，将其分为这四类），分别是Parser、Data、MException与Helper四类。它们的程序分布如下所示：

|  |  |
| --- | --- |
| 类别 | 类(class) |
| Data | Blocks, BlockchainFile, BlockHeader, ByteArray, Transaction, TransactionInput  TrasactionOutput, Witness |
| Parser | IBlockchainParser, BlockchainParser |
| MException | InvalidBlockchainContentException, InvalidBlockchainFilesException,  UnknownBlockVersionException |
| Helper | BinaryReaderExtension, ByteArrayExtension, BlockMemoryStreamReader,  Base58, Bench32, Hash160, BlockProcessedEventArgs |

其中几个类简单介绍：

ByteArray： byte的数组，所有的Hash值、脚本值、地址值都用该数组存储。

BlockMemoryStreamReader： 存储某个区块的二进制流。一个.dat文件中存储有多个区块，在解析时，根据区块中表示区块长度的量，读取该长度的二进制流，将其存储为该类，对区块各个部分的解析可通过对同一个BlockMemoryStreamReader对象操作。

几种编码类（具体看第四部分背景知识-比特币脚本与地址）

Base58: Base58Check编码。

Bench32: Bench32编码。

Hash160: Hash160运算。

**工程工作的流程如下所示：**

1、创建BlockchainParser: 开始->传入参数：区块链文件路径与开始文件（可选）->获取区块链文件路径下的所有文件信息->确认这些文件的命名是否是blkxxxxx.dat形式->根据开始文件参数，选择解析的文件（如果未设置开始文件，则从blk00000.dat文件开始解析）->对选中的文件，创建BlockchainFile文件，分别获取到每个文件的BinaryReader->返回List<BlockchainFile>->结束

2、解析文件，获取区块：开始->循环获取一个BlockchainFile对象->如果BlockchainFile中的BinaryReader未读取到文件末尾->读取4个字节，如果是BlockMagicId->读取4个字节，记为区块长度->读取该长度的区块流，开始解析该区块流->解析区块头->解析交易->返回区块->结束

其中解析区块各部分的流程按照每部分结构进行解析就行。

**相关注意点：**

* 区块中所有数据都是以小端模式存储的。无论是前一区块哈希、默克尔根、源交易哈希都需要做个翻转读入。
* 在计算区块头哈希及交易哈希时，计算完成后，做一个翻转才是正确的哈希值。
* 交易哈希的计算TXID:无论是老版本的交易还是隔离见证版的交易，计算方法是一样的都是Double\_SHA256（<version><TxInNum><TxIn><TxOutNum><TxOut><lockTime>）因此隔离见证版的交易要把这几个区间的数据单独取出来，再进行运算。

### 3. BitcoinDatabaseGenerator

**相关：**BitcoinDatabaseGenerator是Github相关项目（[BitcoinDatabaseGenerator](https://github.com/ladimolnar/BitcoinDatabaseGenerator)）原作者的工程，但是它存储的数据结构及相关程序，并不适合于本次项目的使用，因此对本项目进行了较大幅度的修改，修改了数据库的结构，并将Validation部分全部删除。修改后的项目名称变为MBitcoinDatabaseGenerator。

**功能介绍：**调用BitcoinBlockchain.dll解析比特币区块链，并将区块所有信息依次存入SQL Server数据库中。

**运行：**从cmd命令行进入

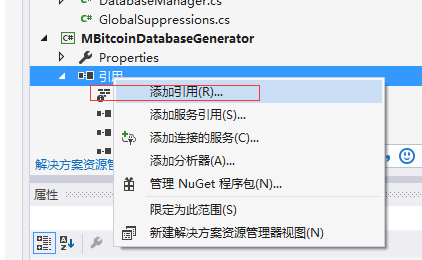
\MBitcoinDatabaseGenerator\Source\BitcoinDatabaseGenerator\bin\Debug\路径

运行MBitcoinDatabaseGenerator.exe /BlockchainPath <区块链文件路径> /SqlServerName <SqlServer 服务器名> /SqlDbName <数据库的名称>

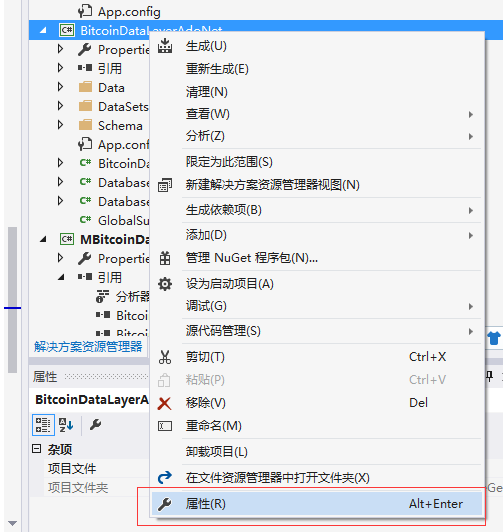
若还有其他相关参数，比如数据库如果设置了账户，则需要加些参数具体可使用 MbitcoinDatabaseGenerator.exe /? 查看相关的参数输入设置。但注意：有关Validation的参数无法使用！

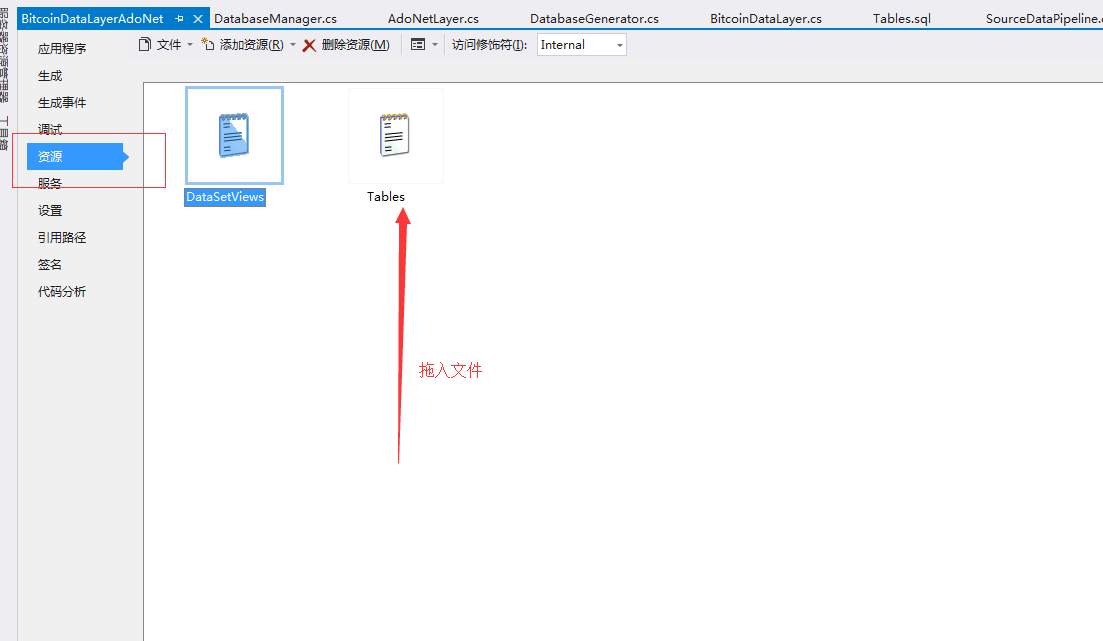
**VS开发设置：**

* 引用设置：由于本项目中，多个类库包括BitcoinBlockchain.dll，都需要在VS中设置引用。设置方法如下图所示：

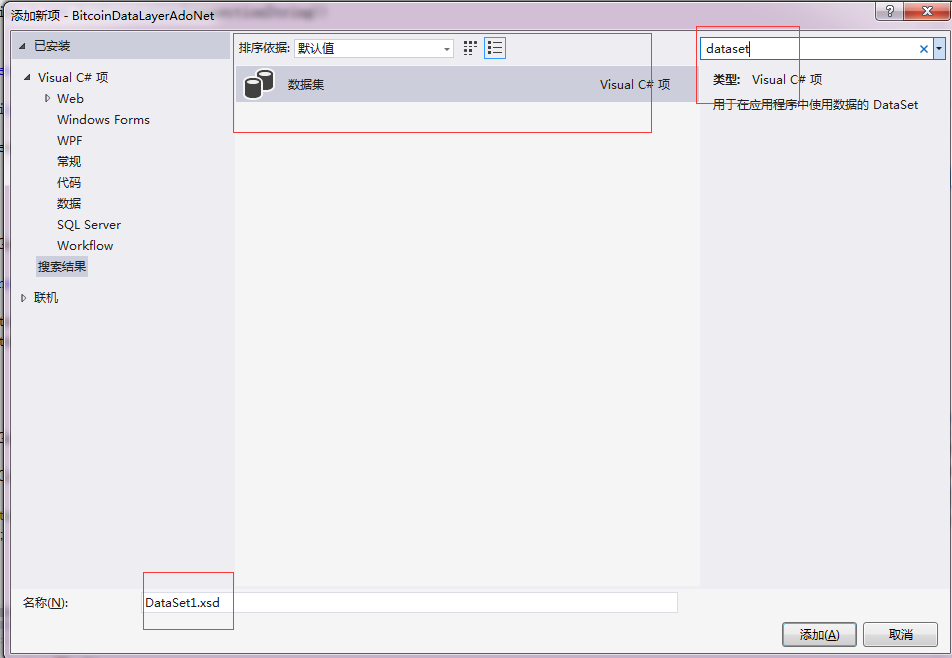


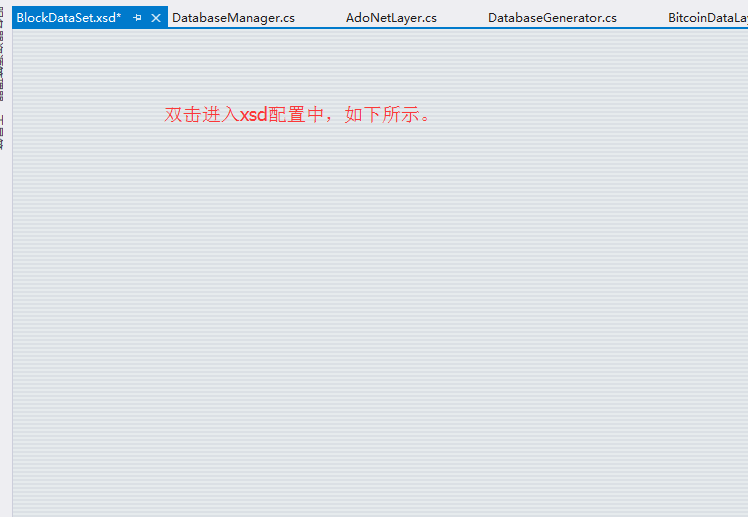
* 资源引用设置:在项目中会调用到某个文件的内容，比如项目中的Resources.Tables,它表示的则为Tables.sql这个文件。设置如下：1）右击项目，选择属性；2）选择资源。3）将文件拖入。

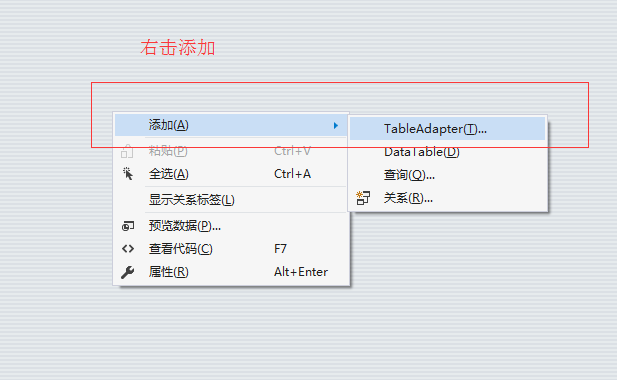


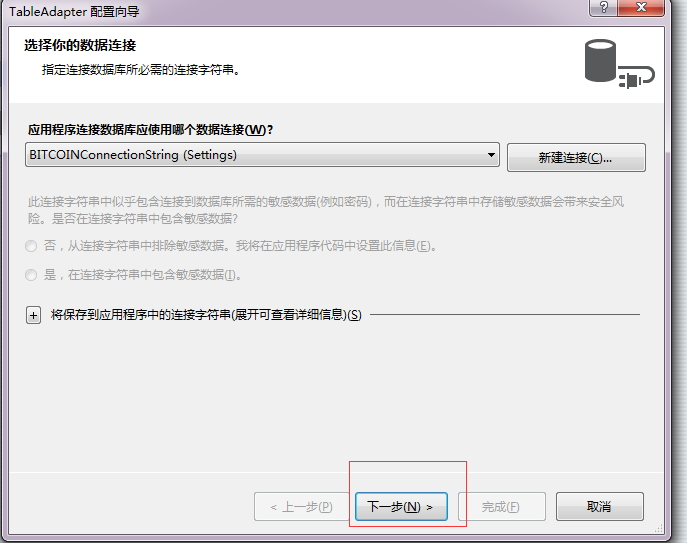


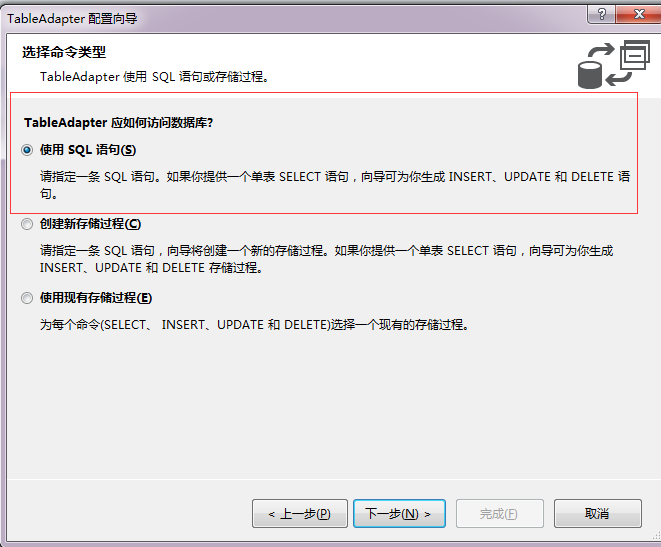
* DataSet类的添加，DataSet类在VS中以XSD的格式表示，该类的详细介绍请参考第四部分。下面介绍如果添加某种DataSet。1）在数据库中创建你需要的对应的表、视图或者其他。因为DataSet要与数据库直接进行数据交换，要求两者的格式完全一致，因此通常都现在数据库中建立好相应的表格、视图，然后创建对应的DataSet。2）在指定项目中添加数据集，如下图所示。3）双击数据集，进入编辑模式。4）右击，添加DataTableAdapter。5）如下图所示依次选择，找到需要的对应的表，选中后，点击确认，则成功生成DataSet.xsd文件。

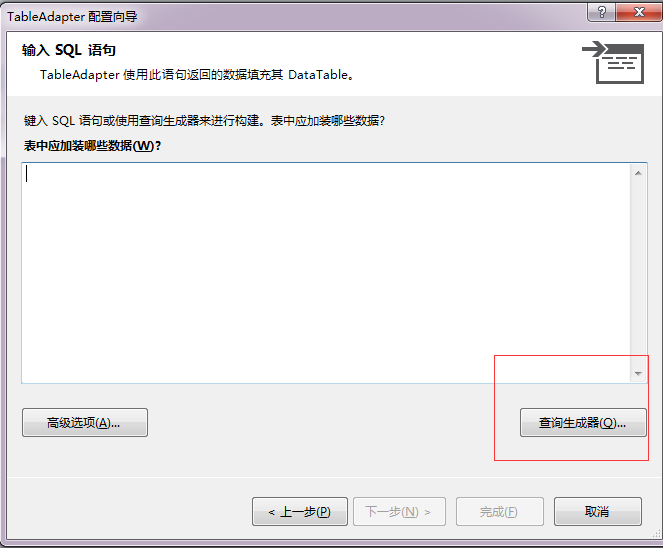


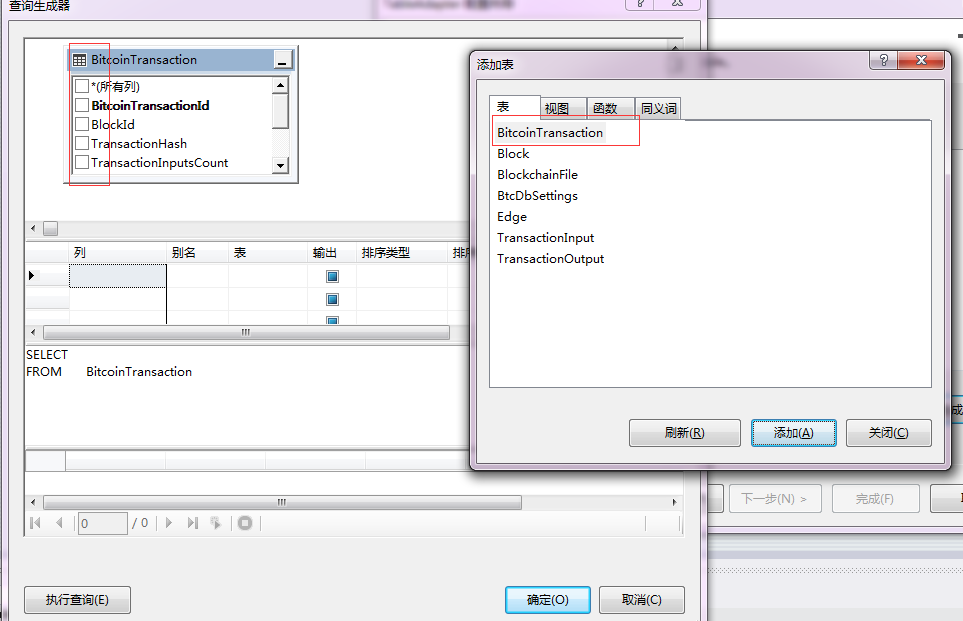


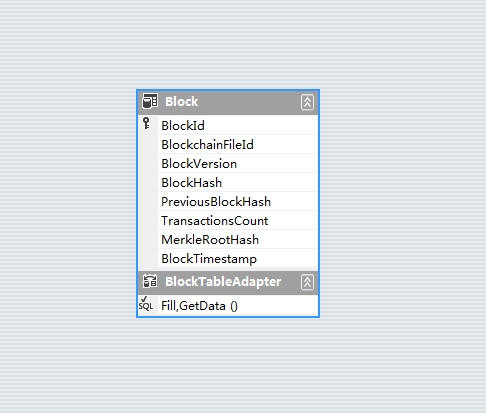












**程序架构：**



图 4 核心类间关系

AdoNetLayer是对ADO.NET对数据库操作的封装，仍旧是与数据库的几个常用操作。BitcoinDataLayer是根据比特币数据库各种表项及其他操作的需要，加的一层中间封装，是对比特币数据库的直接操作，它需要调用AdoNetLayer进行数据库的操作。BitcoinDatabaseGenerator是TOP层，它负责顶层的相关调控，解析文件并将其存入数据库，并设置好编号。而DataSet用于存储临时的数据，通过SourceDataPipeline类将解析的数据存储入DataSet中。然后通过BitcoinDatabaseGenerator调控，调用BitcoinDataLayer类，批量将DataSet中的数据存储入数据库中。

AdoNetLayer：调用ADO.NET框架，对ADO.NET使用的函数进行封装，是直接与数据库打交道的一层。

重要函数：

CreatParameter() 创建SqlCommand的参数，它的作用是用某一数值替代某一字符串，简化代码。比如string1 = “@a and @b” 则通过该函数，可以让变量A替代@a。比如：CreatInputParameter(“@a”, SqlDbType.Int, A)

ConvertDbValue() 将数据库返回的数据转化为需要的数据类型。

FillDataSetFromStatement() 根据查询语句，将数据库的反馈结果存入DataSet中。

FillDataSetFromStoredProcedure() 根据存储过程，将数据库的反馈结果存入DataSet中

ExecuteStatementNoResult() 执行增删改的操作。

ExecuteScalar() 执行只返回某一行某一列的一个数据。

ExecuteStatementReader() 获取返回数据流，可一行一行读数据

BulkCopyTable() 批量将DataSet中的数据存储入数据库中。

CreatCommand() 创建SQL执行指令。

BitcoinDataLayer: 根据比特币区块链对应的数据库进行的相关操作，在该类中存在大量的SQL语句，调用AdoNetLayer执行这些语句，实现对数据库特定表的增删改查。

DataSet: 存储在内存中的，数据库的相关结构。

SourceDataPipeline: 将解析的区块存入到对应的DataSet中。

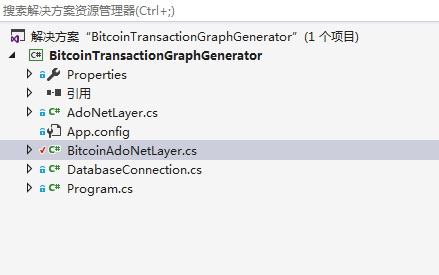
BitcoinDatabaseGenerator: 工程的TOP层，实现整个工程的调控。直接接触的两类为SourceDataPipline，BitcoinAdoNetLayer。

**数据流： 1）**BitcoinDatabaseGenerator调用BitcoinBlockchain.dll解析文件，得到Block。2）BitcoinDatabaseGenerator调用SourceDataPipeline，将Block所有信息存储入DataSet中。3）调用BitcoinDataLayer（它调用AdoNetLayer. BulkCopyTable），批量将DataSet中的内容存储入数据库中。

**其他相关类：**1）ZeroHelpers类库：提供对输入参数的解析。实现对多线程的任务调度。2）DatabaseManager、DatabaseConnection用于用于获取数据库的连接字符串、实现数据库方案建立等。

### BitcoinTransactionGraphGenerator

无论是为地址编号，亦或是生成相关的边。都只需要对数据库进行操作，因此我们可以新建工程，只添加AdoNetLayer与DatabaseConnection即可，然后在BitcoinAdoNetLayer对执行相关的SQL操作等进行封装。如下图所示。





与BitcoinDatabaseGenerator相似，通过执行特定SQL语句，生成交易图谱。因此主要是AdoNetLayer、BitcoinAdoNetLayer，直接执行SQL指令，在数据库中直接生成Edge表。

以交易为节点的图的生成：从第一个交易一直往下遍历，获得每个交易的输入对应的BitcoinTransactionId，然后生成边Source=交易输入的BitcoinTransactionId，Target=当前交易的BitcoinTransactionId，Value为交易输入对应的比特币的值。

## 六、还需进行的工作

目前以交易为节点的图已经以边的形式存储在数据库BITCOIN的Edge表中，可以通过数据库进行访问。

还需要进行以地址为节点，交易为边的图的生成。主要进行下面两种工作：

1. 为地址编号
2. 生成图。
3. 为地址编号及生成图的函数都在BitcoinTransactionGraphGenerator工程的BitcoinAdoNetLayer类中。