# 宠物医院报告

[宠物医院报告](#宠物医院报告)  
 [项目要求](#项目要求)  
 [基本功能实现：](#基本功能实现)  
 [数据库创建代码：](#数据库创建代码)  
 [功能设计与实现](#功能设计与实现)  
 [基本的GUI连接数据库模型](#基本的gui连接数据库模型)  
 [TCP/IP模型](#tcpip模型)  
 [1对多CS模型](#Xafc39e441a39d5ce5080fd003d08d7449ea686e)  
 [NIO非阻塞模式](#nio非阻塞模式)  
 [结果分析](#结果分析)  
 [总结](#总结)

### 项目要求

实现一个GUI界面，分别通过在GUI中直接连接数据库、TCP/IP模型、1对多通信模型、NIO非阻塞通讯模型实现对宠物医院数据的管理，实现插入数据（出于现实性的考虑，将宠物的就诊时间作为数据库存储数据的主键，不可为空）功能、通过宠物主人的姓名或者电话号可实现对宠物就诊信息的查询。

GUI界面如下  


### 基本功能实现：

设置了17个文本域2个按钮，并将数据库的关闭操作连接到右上角的关闭键上。除就诊时间为必填项外，处于现实考虑在部分信息缺失的情况下可以空缺文本域。在填充相关信息后按“添加就诊记录”按钮可以将所给数据插入到数据库Pet中（数据库提前在Mysql Workbench中进行创建）在输入了主人电话号码或者主人姓名（设置为不可同时查询，因为考虑到了主人换手机的情况）可以进行查询操作，将结果写在默认文本为“此处显示查询结果”的文本域中（除此文本域以外的8个默认有文本的文本域设置为不可更改，在其下方空白文本域进行数据的填写）

#### 数据库创建代码：

CREATE DATABASE PetHospital;  
USE PetHospital;  
CREATE TABLE PET(  
 petname VARCHAR(8),  
 kind VARCHAR(8),  
 weight VARCHAR(8),  
 pettype VARCHAR(8),  
 oname VARCHAR(8),  
 brith DATETIME,  
 otelephone VARCHAR(12),  
 ctime DATETIME primary key  
);

### 功能设计与实现

#### 基本的GUI连接数据库模型

需要三个类*Frame*、*Pet*、*PetHospitalDatbase* 分别用于创建GUI界面、承载待插入的宠物信息、连接数据库提供Connection对象。

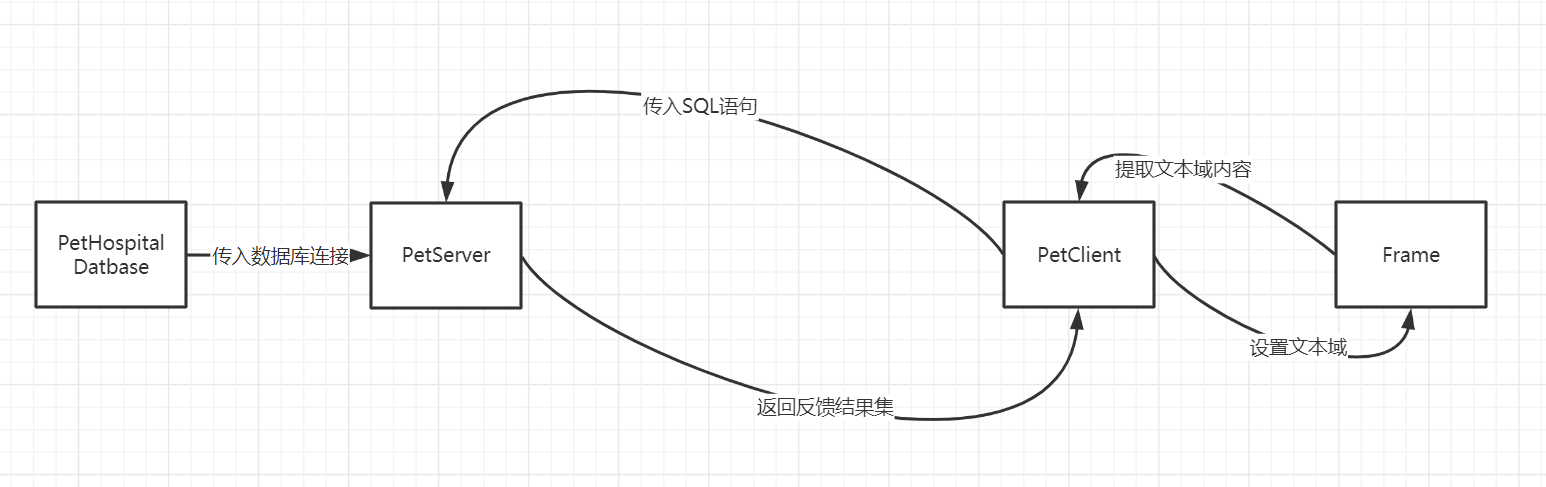
在Frame中通过构造器利用BoxLayout布局在中间区域放置文本域，在底部区域放置两个按键并利用setText方法为文本域设置默认内容，并利用setEditable(false)将其中几个文本域内容设置为不可更改。  
在actionPerformed方法中利用getSource方法分别对添加和查询两个按钮的操作内容进行设置：

1. Add按钮：通过文本域的getText方法将填写的内容赋值给Pet的实例化对象。并通过Pet对象的get…方法构造Insert类型的SQL语句。通过由PetHospitalDatbase类提供的Statement对象的executeUpdate方法进行数据库更新，由于更新操作只返回被影响行数，故不在GUI上输出
2. Search按钮：获取主人姓名和电话号码两个文本域内容对比哪一个不为空，用来设置SQL语句通过Statement对象的executeQuery方法进行执行，返回结果集，通过结果集对象的isBeforeFirst方法查看是否存在第一行数据从而确定结果集是否为空，若为空则通过JOptionPane.showMessageDialog弹出错误语句提示查无此结果。否则将返回的结果集进行字符串拼接操作，将所得字符串利用setText方法传输到查询结果的文本框中。（注意：下一次进行查询前应确保用来拼接的字符串为空，否则会重复上一次查询内容）

在addWindowListener方法中不仅通过System.exit方法做退出操作还需关闭数据库连接。

PetHospitalDatbase类通过Usedatabase方法向Frame类中返回Connection并作为Frame中的私有变量。

#### TCP/IP模型

TCP/IP模型基于基本的基本的GUI连接数据库模型进行功能设计。增设PetServer类和PetClient类并将原Frame中功能拆解到服务端和客户端中。  


服务端与客户端之间通过套接字进行输入输出流的通讯。通过socket的getOutputStream和getInputStream方法进行输入输出流的获取。

在服务端类的构造器中创建服务端连接8899端口。创建ServerExcute方法，分为两部分 1. 接受来自Client的SQL语句。 2. 利用获取的Connection方法对SQL语句进行处理，若为更新语句则不进行反馈，若为查询语句则还要进行结果集的返回。结果集的处理类似于基本模型，首先将通过结果集的next（）方法将结果集中的SQL逐条取出，若无结果则返回“Nothing”否则逐条写入输出流中，在客户端通过套接字进行获取。服务端变量ifClose在客户端传来“CloseServer”语句时关闭服务端，并将此方法关联到Frame中的退出操作监听中。

为客户端设置三个重载的构造器，若传入的时Pet方法则向服务端传入更新的SQL语句，若形参为字符串则向服务端传入由主人姓名或者电话构成的SQL查询语句，若无参数则向服务端传入“CloseServer”关闭服务端。

#### 1对多CS模型

1对多的CS模型基于基本的TCP/IP模型构成，首先需要将Frame模型线程化，由于PetClient在Frame中创建与结束故不必更改。

关键在于在服务端的执行环境中创建线程池。将由服务端接收的socket创建PetServer对象，并在线程池中处理（ServerSocket在PetServer类外创建，PetServer应继承Runable类专门用与SQL语句的执行与反馈）。

ss = new ServerSocket(8899);  
 // 创建线程池  
 ExecutorService executor = Executors.newCachedThreadPool();  
 while(true){  
 Socket socket = ss.accept();  
 executor.execute(new PetServer(socket, clientNo));  
 clientNo++;  
 }

#### NIO非阻塞模式

NIO非阻塞模式模型基于1对多CS模型在PetClient和PetServer类中进行修改。

NIO通过Selector分配的通道进行数据交流。分别应在客户端和服务端通过Selector.open方法创建两个选择器，并将创建的通道分别注册到选择器上。

服务端在创建并注册服务端通道后进入循环。在循环中假如当前选择器中就绪的通道数为0则输出“Loading…”并跳过当前的循环。否则由选择器的selectedKeys方法创建迭代器依次对就绪的客户端通道进行处理。

处理流程：通过SelectionKey对象的isAcceptable和isReadable方法确定当前通道的就绪类型（注：服务端通道应注册为Accept操作敏感）。并为两种情况分别写两个方法。

1. 接收客户端通道方法handleAccept：通过传入参数SelectionKey对象获取就绪客户端通道，利用register方法注册为Read操作敏感。
2. 从客户端通道读数据方法handleRead：通过传入参数SelectionKey对象获取就绪客户端通道（已确认为读就绪通道）由channel方法进行获取。由read方法将通道中数据写入缓冲区中，并将缓冲区字符串化，获取SQL语句。

// NIO模式读数据操作  
 ByteBuffer buffer = ByteBuffer.allocate(1024);  
 try {  
 while (((len = sc.read(buffer)) > 0)) {  
 // 将读取位置调整到开始位置  
 buffer.flip();  
 // 读取缓冲区内容，获取从客户端传来的SQL语句  
 SQL = new String(buffer.array(), 0, len);  
 buffer.clear();  
 }

1. 对读取的SQL语句进行分类，若为更新语句则直接连接数据库进行数据库更新，无后续操作，若为“CloseServer”则设置ifClose为true沿用之前模型关闭服务端的操作。若为查询语句则将独处的结果集拼接为一个String类型的变量，调用协会方法handleWrite将结果集写回客户端通道（若结果集为空沿用之前方法）。
2. handleWrite方法通过接收一个客户端通道与结果集字符串直接进行NIO写操作。利用ByteBuffer类的wrap方法将字符串字节化后打包发写入客户端通道。

// NIO写操作  
ByteBuffer buffer = ByteBuffer.wrap(Qresult.getBytes());  
 SChannel.write(buffer);

服务端每处理一起就绪的客户端通道将当前事件移除迭代器，在迭代器读完后重新下一轮循环由选择器提取新的就绪事件。

客户端将原SQL写入服务端的方法改为NIO的通道写入模式，并将原本获取服务端反馈的方法移除。获取反馈的操作改为在Frame中线程化执行。

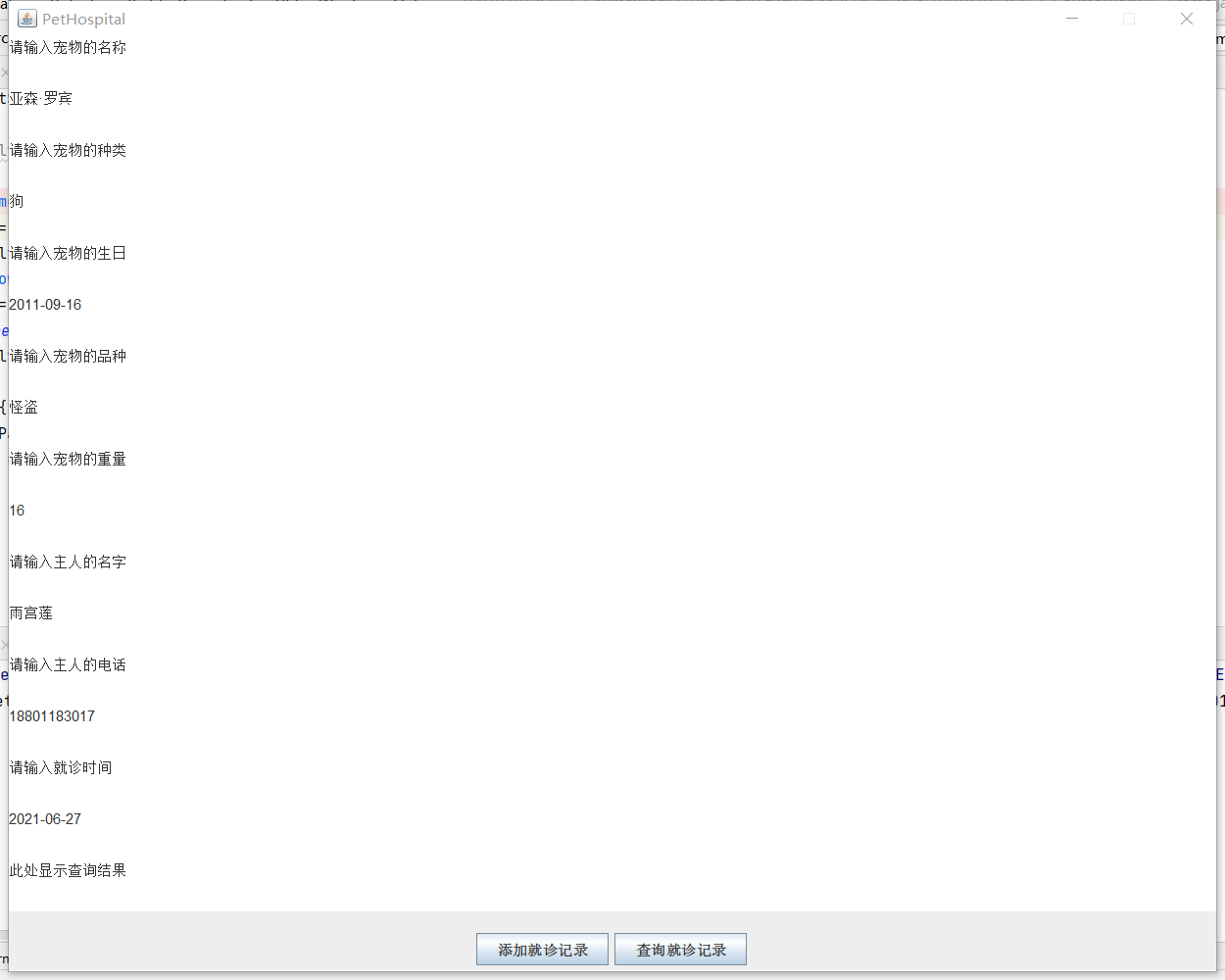
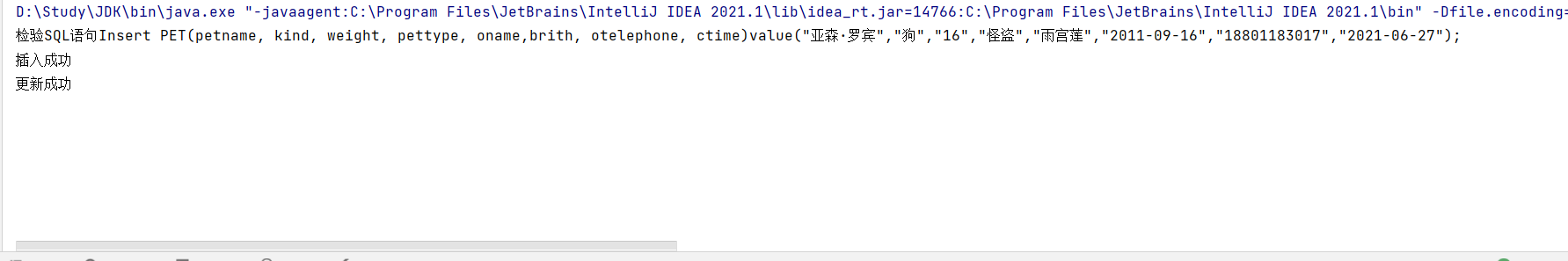
Frame中在原本将结果填入文本域的位置改为调用getResultset方法。

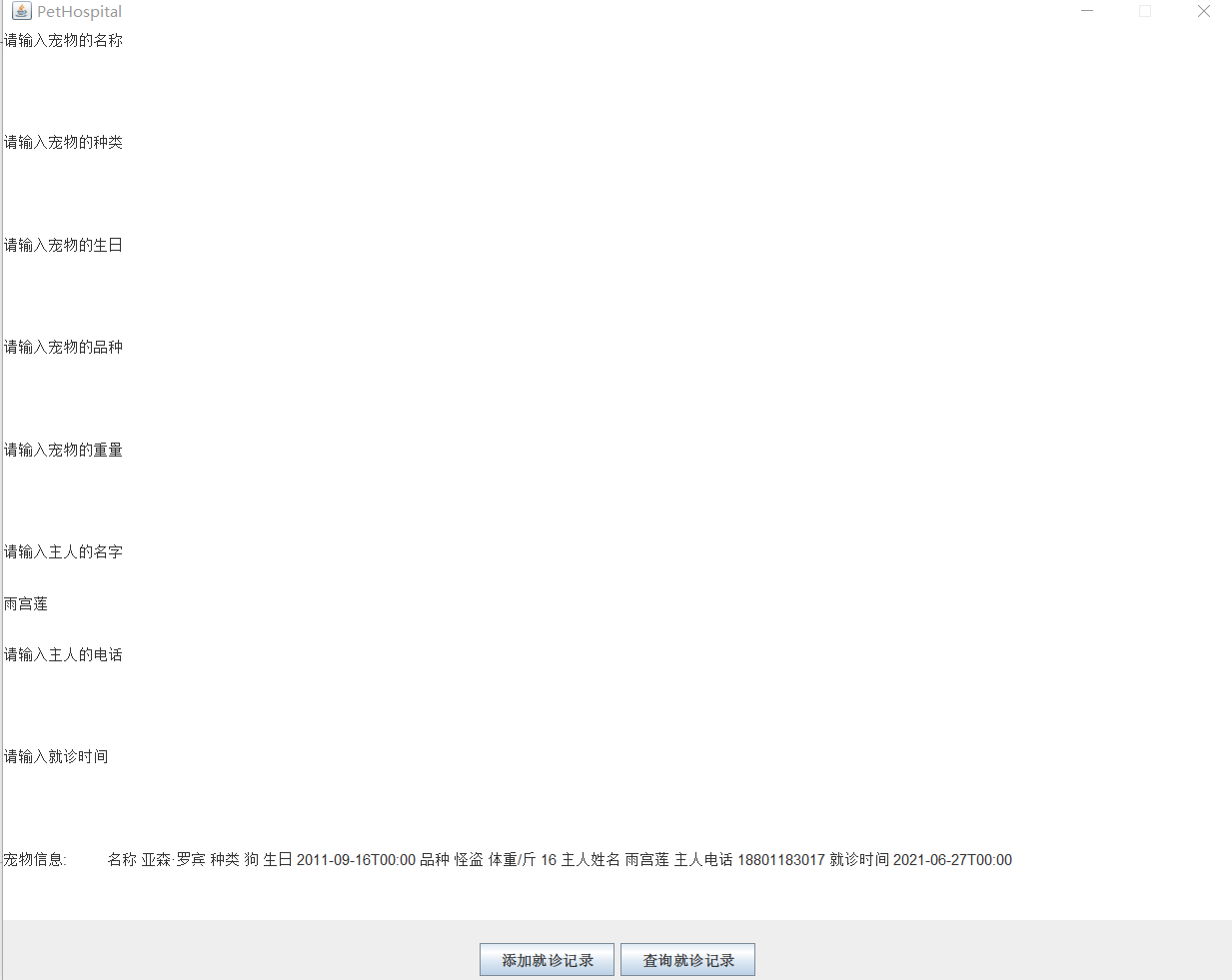
getResultset方法中创建线程thread，在线程中由客户端的选择器选取当前就绪的通道，将获取的SelectionKey对象迭代器化，将其中通道逐一进行NIO读操作，将读出的数据直接写入文本域（若为Nothing，则如上述模型调用弹窗，并将字符串变量清空，避免下次读入“Nothing”）。线程启动后，通过while(thread.isAlive()){}确保线程执行完毕后才可进行后面的操作。

### 结果分析

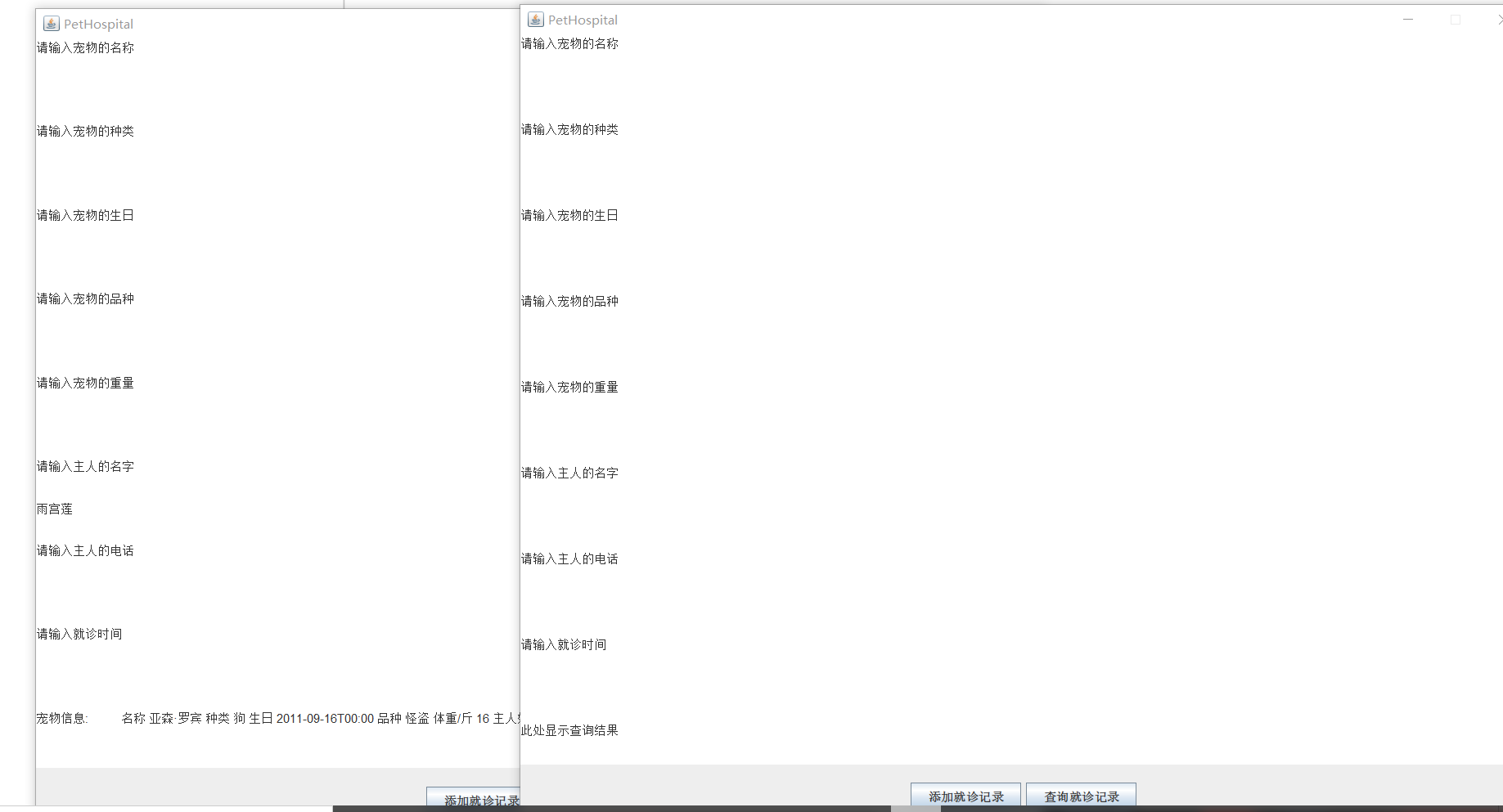
四个模型的结果在GUI界面上的结果显示基本一致，涉及多线程的模型可以同时启动多个GUI进行使用，故统一进行演示。

更新操作：

  
  
查询刚才插入的结果：

电话查询：  
 姓名查询：

多线程显示：



所有所需功能都已实现。

### 总结

熟悉了JDBC的使用，并且对java的网络编程以及非阻塞编程的认识更加深刻。对不同类之间读写数据的多种方式有了更深层次的掌握。