

《网络协议分析与设计》

课程论文

|  |  |
| --- | --- |
| 课 程 名 称 | 网络协议分析与设计 |
| 团 队 成 员 | 徐 通 （20171346073）  徐宇尧（20171346074）  赵文鑫（20171346071）  曾 实 （20171346078）  梁 创 （20171346077） |
| 指 导 教 师 | 刘琦 |

2019 年 12 月 30 日

**目 录**

[简单的能源收集协议 4](#_Toc28970656)

[第1章 题目描述 6](#_Toc28970657)

[1.1 问题描述 6](#_Toc28970658)

[1.2 完成的任务 6](#_Toc28970659)

[第2章 开发环境相关介绍 8](#_Toc28970660)

[**2.1** Microsoft Visual Studio介绍 8](#_Toc28970661)

[2.1.1 Microsoft Visual Studio简介 8](#_Toc28970662)

[2.1.2 Microsoft Visual Studio的特点及功能 8](#_Toc28970663)

[**2.2** C#介绍 8](#_Toc28970664)

[2.3 Java介绍 9](#_Toc28970665)

[2.4 Matlab介绍 9](#_Toc28970666)

[第3章 基本分析 10](#_Toc28970667)

[3.1 协议功能结构 10](#_Toc28970668)

[3.2 协议设计目标 10](#_Toc28970669)

[第4章 详细设计 11](#_Toc28970670)

[4.1服务端 11](#_Toc28970671)

[4.1.1 服务端的设计 11](#_Toc28970672)

[4.1.2服务端和老师给的客户端之间调试 13](#_Toc28970673)

[4.2 数据库 14](#_Toc28970674)

[4.2.1 数据库的设计 14](#_Toc28970675)

[4.2.2 数据库的连接 16](#_Toc28970676)

[4.2.4 数据库中改进的问题 20](#_Toc28970677)

[4.3服务端和客户端的双向通信 20](#_Toc28970678)

[4.3.1服务端发送简单反馈信息 20](#_Toc28970679)

[4.4利用JAVA和MySQL实现数据的动态实时监控 21](#_Toc28970680)

[4.4.1设计DBUtil工具类模块。 22](#_Toc28970681)

[4.4.2设计Entity模块， 22](#_Toc28970682)

[4.4.3设计DAO模块 22](#_Toc28970683)

[4.4.4设计Charts模块 24](#_Toc28970684)

[4.4.5遇到的问题 25](#_Toc28970685)

[4.5 瞬时电功率可视化实时动态监控 26](#_Toc28970686)

[4.5.1 可视化实时动态监控平台的选取 26](#_Toc28970687)

[4.5.2 Matlab连接MySQL数据库 26](#_Toc28970688)

[4.5.3 Matlab实现动态可视化监控 29](#_Toc28970689)

[4.5.4 C#调用Matlab，实现Matlab图形Figure嵌入Winform窗体 32](#_Toc28970690)

[4.6客户端及其对应控制机制实现 34](#_Toc28970691)

[4.6.1 客户端 34](#_Toc28970692)

[4.6.2 控制机制 37](#_Toc28970693)

[4.6.3 运行实例 38](#_Toc28970694)

[4.6.4 控制机制中改进的问题 38](#_Toc28970695)

[第5章 软件测试 39](#_Toc28970696)

[5.1数据库连接测试 39](#_Toc28970697)

[5.1.1 client端与server端发送数据 39](#_Toc28970698)

[5.1.2 数据库存储数据 39](#_Toc28970699)

[5.2 Java连接测试 40](#_Toc28970700)

[5.2.1测试Java连接提取数据库 40](#_Toc28970701)

[5.2.2 测试每秒动态更新折线图 40](#_Toc28970702)

[参考文献: 42](#_Toc28970703)

[附 录: 43](#_Toc28970704)

# 简单的能源收集协议

摘 要：今天，能源已成为人类社会不可或缺的基本要素。在这个星球上, 随着能源日益紧张和环境恶化, 获得经济方便环保的能源变成一个关系人类生存与可持续发展的急迫问题, 寻找提高能源利用效率的解决之道成为小到社会家庭,大到企业与政府等全社会的共同责任。因此，家庭能源管理系统(HEMS)正在成为一种潮流，其已经是物联网中的一个典型应用。

一个HEMS系统中通常包含能源数据传感器、自定义的默认网关和服务器。本次简单的能源收集协议设计主要是通过C#、Matlab、Java、数据库等技术，实现简单的能量收集服务，并将其实现在局域网中。可以将通信协议设计为分析传入的能量数据的格式，并将其存储在预定义的数据库中。

本文开始介绍了本课程论文所要求的题目与本组完成的基本贡献，然后介绍开发简单的能源收集协议所使用的一些工具软件及基本知识，就整个系统的各模块做了简单介绍并做了图示分析。最后对整个系统进行了需求分析，对代码的各个模块功能做了基本的说明，对功能模块的实现做了代码解释说明。在测试报告中对系统测试情况做了介绍，重点说明了测试过程中发现并解决的问题。

关键词：能源收集协议，需求设计，总体设计

**A Simple Energy Collection Protocol**

**Abstract:** Today, energy has become an essential element of human society. On this planet, with increasing energy tension and environmental degradation, access to economically convenient and environmentally friendly energy has become an urgent issue related to human survival and sustainable development. Finding solutions to improve energy efficiency has become from social families to the joint responsibility of the whole society, such as enterprises and governments. Therefore, the home energy management system (HEMS) is becoming a trend, which is already a typical application in the Internet of Things.

A HEMS system usually contains energy data sensors, custom default gateways, and servers. The design of this simple energy collection protocol is mainly to implement simple energy collection services through C #, Matlab, Java, database and other technologies, and it will be in the local area network. The communication protocol (server) can be designed to analyze the incoming energy data in a format and store it in a predefined database.

This article begins by introducing the topics required for the course thesis and the basic contributions completed by this group, and then introduces some of the tool software and basic knowledge used to develop a simple energy collection protocol and graphical analysis. Finally, the requirements analysis of the entire system was performed, the basic functions of each module of the code were explained, and the implementation of the functional modules was explained. In the test report, the system test situation is introduced, and the problems found and solved during the test are highlighted.

**Keywords:** Energy collection protocol, Demand analyze, Whole design, Function realizing

第1章 题目描述

## 1.1 问题描述

在此次任务中，您需要设计简单的能量收集服务，并将其实现在局域网中。可以将通信协议设计为分析传入的能量数据的格式，并将其存储在预定义的数据库中。数据传感器的模拟器已实现用于您的测试。您无需实施整个HEMS系统，但必须设计和实施协议。作业评估的详细信息如下：

1、[强制]在数据传感器和服务器之间进行通信，该服务器包括至少一个模拟数据传感器和服务器程序；

2、[强制]完整的报告非常重要；

3、[可选]数据传感器，即客户端程序也可以设计和实现；

4、[可选]鼓励使用图形服务器界面；

5、[可选]鼓励可以处理多个数据传感器的服务器；

6、[可选]从建议的服务器到数据传感器的控制机制将获得更多的赞赏。

## 1.2 完成的任务

在本次课程设计中，根据系统功能要求，本组成员将问题解决分为以下步骤并将其解决：

（1）梁创利用TCP三次握手的基本原理与socket编程，实现了server端的基本设计。

（2）徐通利用数据库的基础知识，建立了server数据库，在client端与server端在进行通信的过程中，将相互传输的能量数据按照格式存储到数据库中。

（3）徐宇尧利用C#与Java的具体知识接入数据库，进行server端与client端的双向通信，并将能量数据中的瞬时电功率中最新的10行数据的变化趋势通过图形化的界面进行展示。

（4）曾实利用Matlab的相关知识接入数据库，将数据库中全部的瞬时电功率通过图形化的界面展示其整体的变化趋势。

（5）赵文鑫利用C#的相关知识，实现了client端的基本设计与控制机制。

**Chapter 1 Title Description**

Problem Description

In this assignment, you are required to design your simple energy collection service and implement it in a local area network. A communication protocol can be designed to analysis the format of incoming energy data and store them in the predefined database. A simulator of the data sensor has been implemented for your testing. You are required to design and implement a protocol. Details on the assignment assessment is shown below:-

[compulsory] communicate between a data sensor and a server, which include at least one simulated data sensor and the server programme;

[compulsory] a complete report is critically required;

[optional] the data sensor, i.e. the client programme can be designed and implemented as well;

[optional] a graphical server interface is encouraged;

[optional] a server which can handle multiple data sensors is encouraged;

[optional] a control mechanism from the proposed server to a data sensor will gain extra appreciation.

Implemented tasks

In this course design, according to the system function requirements, the members of my group divided the problem into the following steps and resolved them:

(1) Chuang Liang used the basic principle of TCP three-way handshake and socket programming to realize the basic design of the server.

(2) Tong Xu uses the basic knowledge of the database to establish a server database. During the communication between the client and the server, the energy data transmitted to each other is stored in the database according to the format.

(3) Yuyao Xu uses the specific knowledge of Java to access the database, and displays the trend of the latest 10 lines of the instantaneous electric power in the energy data through a graphical interface.

(4) Shi Zeng used Matlab's relevant knowledge to access the database, and showed all the instantaneous electric power in the database through a graphical interface to show its overall change trend.

(5) Wenxin Zhao used the relevant knowledge of C # to realize the basic design and control mechanism of the client.

第2章 开发环境相关介绍

* 1. Microsoft Visual Studio介绍

2.1.1 Microsoft Visual Studio简介

[**Microsoft Visual Studio**](https://baike.baidu.com/item/Microsoft%20Visual%20Studio)**（简称VS）是美国**[**微软公司**](https://baike.baidu.com/item/%E5%BE%AE%E8%BD%AF%E5%85%AC%E5%8F%B8)**的开发工具包系列产品。VS是一个基本**[**完整**](https://baike.baidu.com/item/%E5%AE%8C%E6%95%B4/32785)**的开发工具集，它包括了整个**[**软件生命周期**](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F/861455)**中所需要的大部分工具，如UML工具、代码管控工具、**[**集成开发环境**](https://baike.baidu.com/item/%E9%9B%86%E6%88%90%E5%BC%80%E5%8F%91%E7%8E%AF%E5%A2%83/298524)**(IDE)等等。所写的目标代码适用于微软支持的所有平台，包括**[**Microsoft Windows**](https://baike.baidu.com/item/Microsoft%20Windows)**、**[**Windows Mobile**](https://baike.baidu.com/item/Windows%20Mobile)**、**[**Windows CE**](https://baike.baidu.com/item/Windows%20CE)**、**[**.NET Framework**](https://baike.baidu.com/item/.NET%20Framework)**、**[**.NET Compact Framework**](https://baike.baidu.com/item/.NET%20Compact%20Framework)**和Microsoft**[**Silverlight**](https://baike.baidu.com/item/Silverlight)**及**[**Windows Phone**](https://baike.baidu.com/item/Windows%20Phone)**。**

[**Visual Studio**](https://baike.baidu.com/item/Visual%20Studio)**是最流行的**[**Windows**](https://baike.baidu.com/item/Windows)**平台应用程序的**[**集成**](https://baike.baidu.com/item/%E9%9B%86%E6%88%90)**开发环境。最新版本为 Visual Studio 2019 版本，基于.NET Framework 4.8。**

2.1.2 Microsoft Visual Studio的特点及功能

1、支持[Windows Azure](https://baike.baidu.com/item/Windows%20Azure)，微软[云计算架构](https://baike.baidu.com/item/%E4%BA%91%E8%AE%A1%E7%AE%97%E6%9E%B6%E6%9E%84)迈入重要里程碑。

2、助力移动与嵌入式装置开发，三屏一云商机无限。

3、实践当前最热门的 Agile/Scrum 开发方法，强化团队竞争力。

4、升级的软件测试功能及工具，为[软件质量](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E8%B4%A8%E9%87%8F)严格把关。

5、搭配[Windows 7](https://baike.baidu.com/item/Windows%207)，[Silverlight](https://baike.baidu.com/item/Silverlight)4 与 Office，发挥多核并行运算威力，

6、创建美感与效能并重的新一代软件。

7、支持最新C++标准，增强IDE，切实提高程序员开发效率。

* 1. **C#介绍**

C#是[微软公司](https://baike.baidu.com/item/%E5%BE%AE%E8%BD%AF%E5%85%AC%E5%8F%B8/732128)发布的一种面向对象的、运行于[.NET Framework](https://baike.baidu.com/item/.NET%20Framework)和.NET Core(完全开源，跨平台)之上的高级程序设计语言。并定于在微软职业开发者论坛(PDC)上登台亮相。C#是微软公司研究员Anders Hejlsberg的最新成果。C#看起来与Java有着惊人的相似；它包括了诸如单一继承、接口、与Java几乎同样的语法和编译成中间代码再运行的过程。但是C#与Java有着明显的不同，它借鉴了Delphi的一个特点，与COM（组件对象模型）是直接集成的，而且它是微软公司 .NET windows网络框架的主角。

C#是一种安全的、稳定的、简单的、优雅的，由C和[C++](https://baike.baidu.com/item/C%2B%2B)衍生出来的[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/2262089)的编程语言。它在继承C和C++强大功能的同时去掉了一些它们的复杂特性（例如没有[宏](https://baike.baidu.com/item/%E5%AE%8F/2648286)以及不允许多重继承）。C#综合了[VB](https://baike.baidu.com/item/VB)简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向[组件](https://baike.baidu.com/item/%E7%BB%84%E4%BB%B6)编程的支持成为[.NET](https://baike.baidu.com/item/.NET)开发的首选语言。

C#是[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/2262089)的编程语言。它使得程序员可以快速地编写各种基于MICROSOFT .NET平台的应用程序，MICROSOFT .NET提供了一系列的[工具](https://baike.baidu.com/item/%E5%B7%A5%E5%85%B7)和服务来最大程度地开发利用计算与通讯领域。

C#使得C++程序员可以高效的开发程序，且因可调用由 C/C++ 编写的本机原生函数，而绝不损失C/C++原有的强大的功能。因为这种[继承](https://baike.baidu.com/item/%E7%BB%A7%E6%89%BF/2261239)关系，C#与C/C++具有极大的相似性，熟悉类似语言的开发者可以很快的转向C#。

## **2.3 Java介绍**

Java是由Sun Microsystems公司（现已被oracle公司收购）于1995年5月推出的Java面向对象程序设计语言和Java平台的总称。由James Gosling和同事们共同研发，并在1995年正式推出，据oracle官方数据指数，目前全球已有上亿的系统是使用Java开发的。

Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、指针等概念，因此Java语言具有功能强大和简单易用两个特征。Java语言作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式进行复杂的编程。

Java具有简单性、面向对象、分布式、健壮性、安全性、平台独立与可移植性、多线程、动态性等特点。Java可以编写桌面应用程序、Web应用程序、分布式系统和嵌入式系统应用程序等。

### **2.4 Matlab介绍**

MATLAB是美国[MathWorks](https://baike.baidu.com/item/MathWorks)公司出品的商业[数学软件](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6%E8%BD%AF%E4%BB%B6)，用于算法开发、数据可视化、数据分析以及[数值计算](https://baike.baidu.com/item/%E6%95%B0%E5%80%BC%E8%AE%A1%E7%AE%97/3729797)的高级技术计算语言和交互式环境，主要包括MATLAB和Simulink两大部分。

MATLAB是matrix&laboratory两个词的[组合](https://baike.baidu.com/item/%E7%BB%84%E5%90%88/1218690)，意为矩阵工厂（矩阵实验室）。是由美国[mathworks](https://baike.baidu.com/item/mathworks/5887044)公司发布的主要面对科学计算、可视化以及交互式程序设计的高科技计算环境。它将[数值分析](https://baike.baidu.com/item/%E6%95%B0%E5%80%BC%E5%88%86%E6%9E%90)、[矩阵计算](https://baike.baidu.com/item/%E7%9F%A9%E9%98%B5%E8%AE%A1%E7%AE%97/8089413)、科学数据可视化以及非[线性](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7)动态系统的[建模](https://baike.baidu.com/item/%E5%BB%BA%E6%A8%A1/814831)和仿真等诸多强大功能集成在一个易于使用的视窗环境中，为科学研究、工程设计以及必须进行有效数值计算的众多科学领域提供了一种全面的解决方案，并在很大程度上摆脱了传统非交互式[程序设计语言](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80/2317999)（如C、Fortran）的编辑模式，代表了当今国际科学计算软件的先进水平。

MATLAB和[Mathematica](https://baike.baidu.com/item/Mathematica)、[Maple](https://baike.baidu.com/item/Maple/2306572)并称为三大数学软件。它在数学类科技应用软件中在数值计算方面首屈一指。MATLAB可以进行[矩阵](https://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)运算、绘制函数和数据、实现算法、创建用户界面、连接其他编程语言的程序等，主要应用于工程计算、控制设计、[信号处理](https://baike.baidu.com/item/%E4%BF%A1%E5%8F%B7%E5%A4%84%E7%90%86/84717)与通讯、[图像处理](https://baike.baidu.com/item/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86/294902)、[信号检测](https://baike.baidu.com/item/%E4%BF%A1%E5%8F%B7%E6%A3%80%E6%B5%8B/10340196)、[金融建模](https://baike.baidu.com/item/%E9%87%91%E8%9E%8D%E5%BB%BA%E6%A8%A1/8576275)设计与分析等领域。

MATLAB的基本数据单位是矩阵，它的[指令](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4/3225201)[表达式](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F/7655228)与数学、工程中常用的形式十分相似，故用MATLAB来解算问题要比用C，[FORTRAN](https://baike.baidu.com/item/FORTRAN)等语言完成相同的事情简捷得多，并且MATLAB也吸收了像Maple等软件的优点，使MATLAB成为一个强大的[数学软件](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6%E8%BD%AF%E4%BB%B6/3363734)。在新的版本中也加入了对[C](https://baike.baidu.com/item/C/7252092)，[FORTRAN](https://baike.baidu.com/item/FORTRAN)，[C++](https://baike.baidu.com/item/C%2B%2B)，[JAVA](https://baike.baidu.com/item/JAVA/85979)的支持。

第3章 基本分析

## 3.1 协议功能结构

本协议设计共分5大部分，10个完整的功能模块。协议结构图3.1如下图所示：

图3.1协议结构图

## 3.2 协议设计目标

本协议是根据家庭能源管理系统的实际需求而开发的，基本能够实现家庭对于能量传送数据的监控，通过本协议可以达到以下目标：

（1）协议分析运行稳定，安全可靠。

（2）界面美观、友好。

（3）信息管理灵活、方便、快捷、准确，数据存储安全可靠。

（4）采用控制机制实时控制数据。

第4章 详细设计

## 4.1服务端

4.1.1 服务端的设计

在服务端的设计中我们需要用到[**Microsoft Visual Studio**](https://baike.baidu.com/item/Microsoft%20Visual%20Studio)**（简称VS）这一款软件来进行编程此时我们需要选择新建一个项目->控制台应用->下一步->填写项目名称就新建完成了。如图4.1和4.2所示。**

****

图4.1新建项目界面

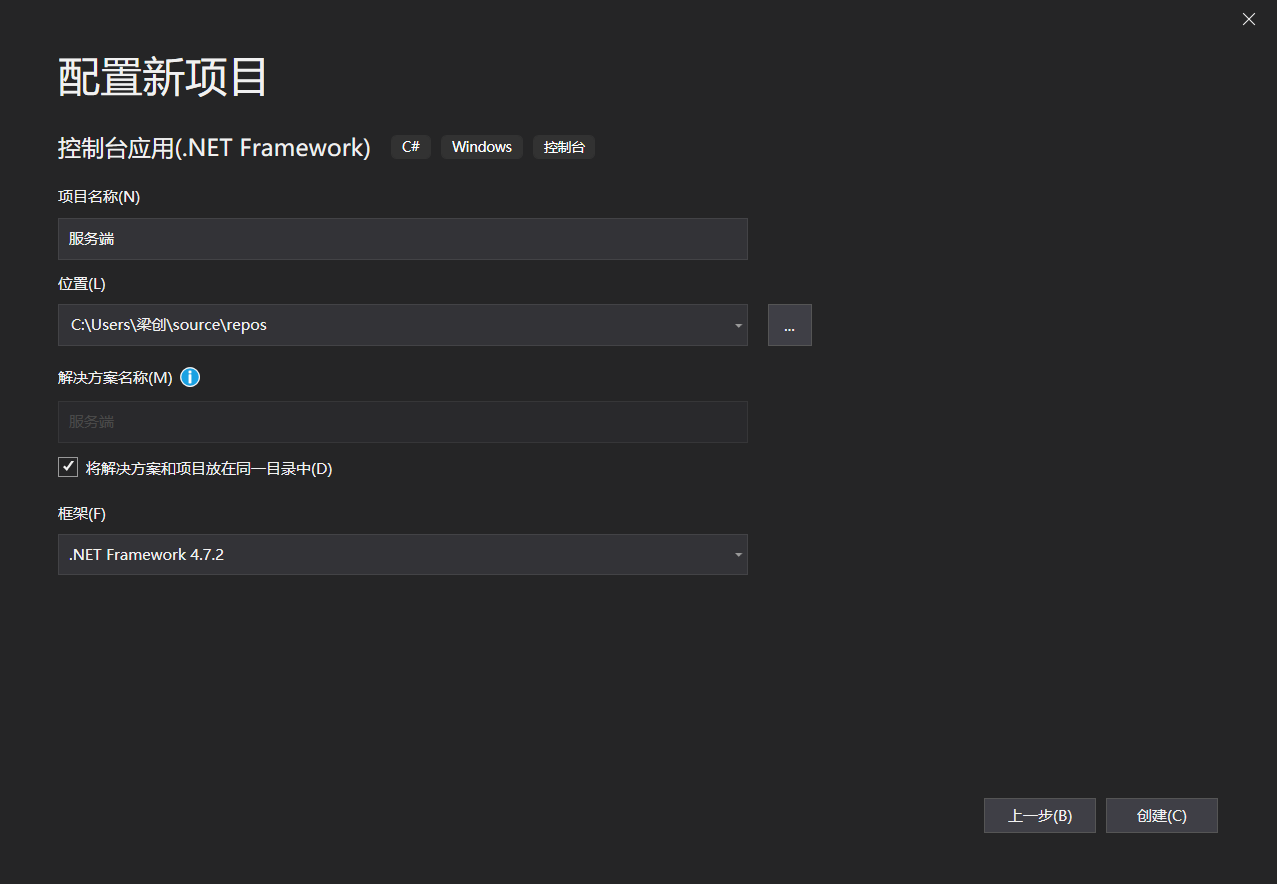
****

图4.2输入项目名称界面

1. 本次服务端设计的总体框架主要为以下几个步骤：
2. 创建用于监听的套接字（socket）。
3. 将套接字绑定到本地地址和端口上（bind）。
4. 将套接字设为监听模式（listen）。
5. 等待客户请求（accept），此处要不断的调用accept。
6. 通信（send/receive）。
7. 关闭套接字（closesocket）。
8. 在知道我们整体的设计思路之后我们就开始写代码，首先我们引入命名空间，在图4.3中各个命名空间的含义和用处如下：
9. using System：是最基础的系统命名空间必须包含。
10. using System.Net：这个是.net framework框架的内置类，在工程中添加引用就可以得到。
11. using System.Net.Socket：提供Sockets网络应用程序的相关类.
12. using System.Text：提供常用的字符串操作，在后面的字符串和ASCII的转换中用到。

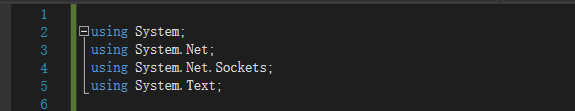
****

图4.3命名空间界面

1. 定义一个数组 Byte[]让他的大小不超过1024，目的是接收从客户端发送的数据的ASCLL码。下面用下面所示语句在服务端获取服务器当前网络下的ip，刚开始做的时候一直都找

PHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());

不到正确的IP，经过一系列的查阅和思考之后想到了一个方法就是使用for循环语句遍历ip地址列表获取到的所有ip地址都输出在屏幕上然后筛选ipv4地址，将最后显示出来的ipv4地址赋给ipAddress当作本机IP；并且将客户端的ip地址设置为此地址。下图4.4和4.5分别是实现此功能的代码和将此功能实现之后的运行界面。

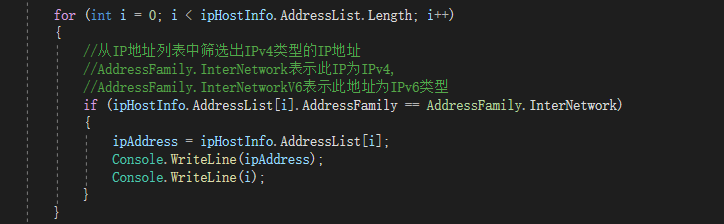


图4.4代码展示界面

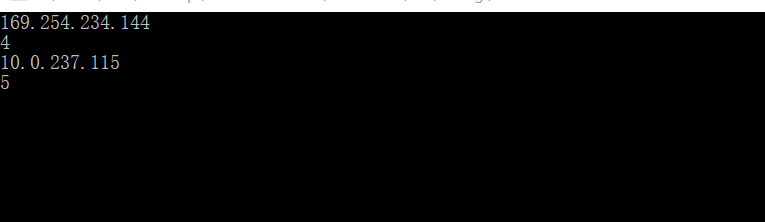


图4.5运行界面

1. 将服务端的地址设置为ipAddress端口号设置为8023，以便和客户端实现交互。使用语句创建一个TCP/IP套接字，并且将本地ip地址ipAddress和端口号8023与套接字使用bind（）语句绑定起来，使用listen（）监听并输出【Waiting for a connection...】等待连接请求。使用while循环语句，让里面的判断值一直为ture来实现当客户端发送的消息包含结束符\r\n需要break出循环时，只能跳出4步骤的while（）语句，转而会继续执行下面的语句（关闭连接）而不是服务端只接受一次数据就关闭连接。并定义一个字符串变量msgnow用来保存即时信息。下图4.6是相关代码。

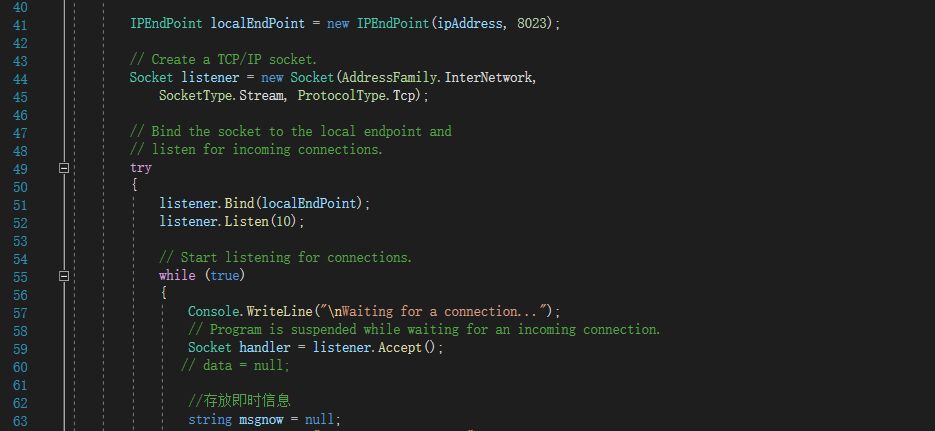


图4.6代码展示界面

1. 通过Receive语句从连接处接收一些数据，由于客户端在发送时字符串是无法直接发送给服务器的，因此客户端用Encoding将字符串编码为byte数组后才能发送，所以要将接收的byte数组通过ASCLL转化成原来的数据，并且赋值给msgnow打印输出，并且判断发来的数据是否带有结束符\r\n,如果有则跳出当前代码段转而执行下面的语句即关闭连接。下图4.7是相关代码实现。

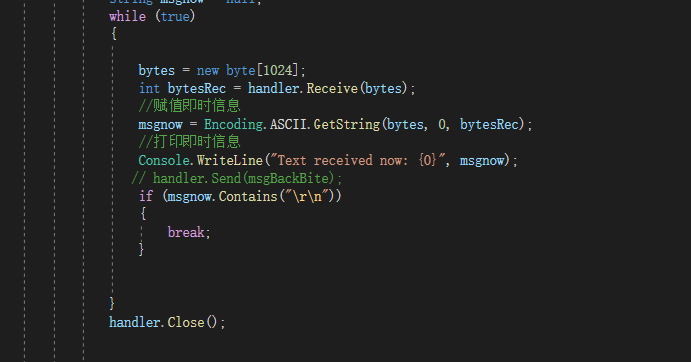


图4.7代码展示界面

4.1.2服务端和老师给的客户端之间调试

1、在写好服务端之后，我们就开始了和老师给的客户端之间的调试。首先根据图4.5所示的ipv4地址，将老师的config.json文件中的第二行改成和此ip地址相同的地址，如图4.8所示。

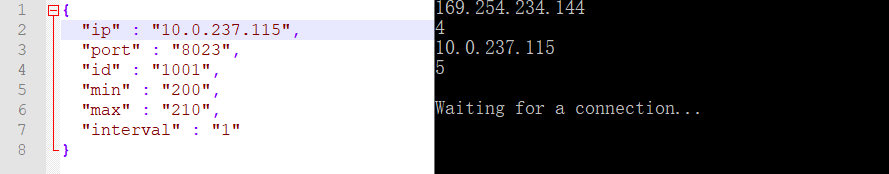


图4.8修改界面

1. 此时开启服务端和客户端，传输数据成功！如图4.9所示。

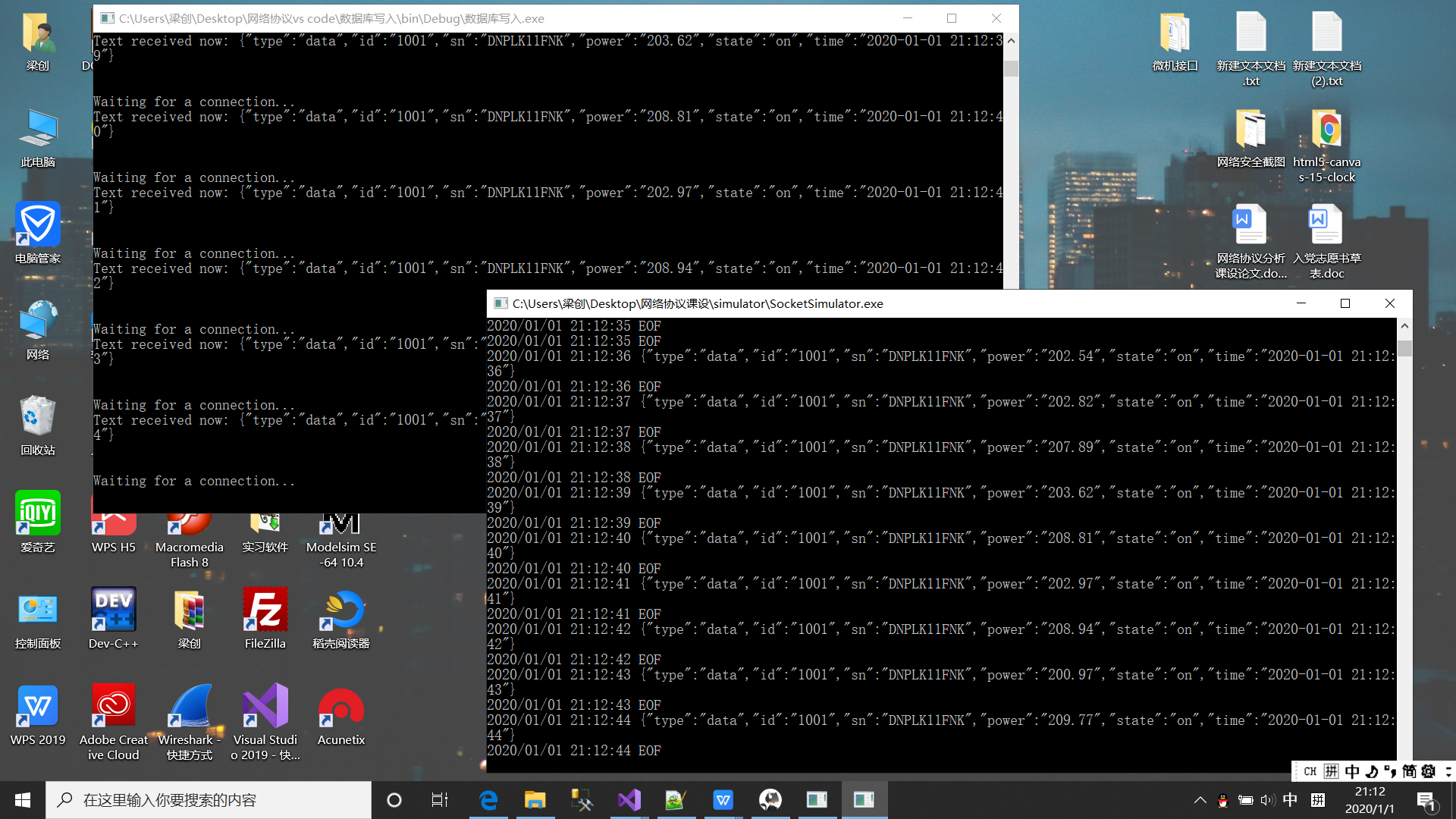


图4.9成功传输界面

## 4.2 数据库

4.2.1 数据库的设计

1、首先我们打开Navicat Premium这款软件，图形化界面如下图4.10所示，在菜单栏中选择建立与数据库的连接（或者从菜单栏【文件】🡪【新建连接】🡪【MySQL】进入）。

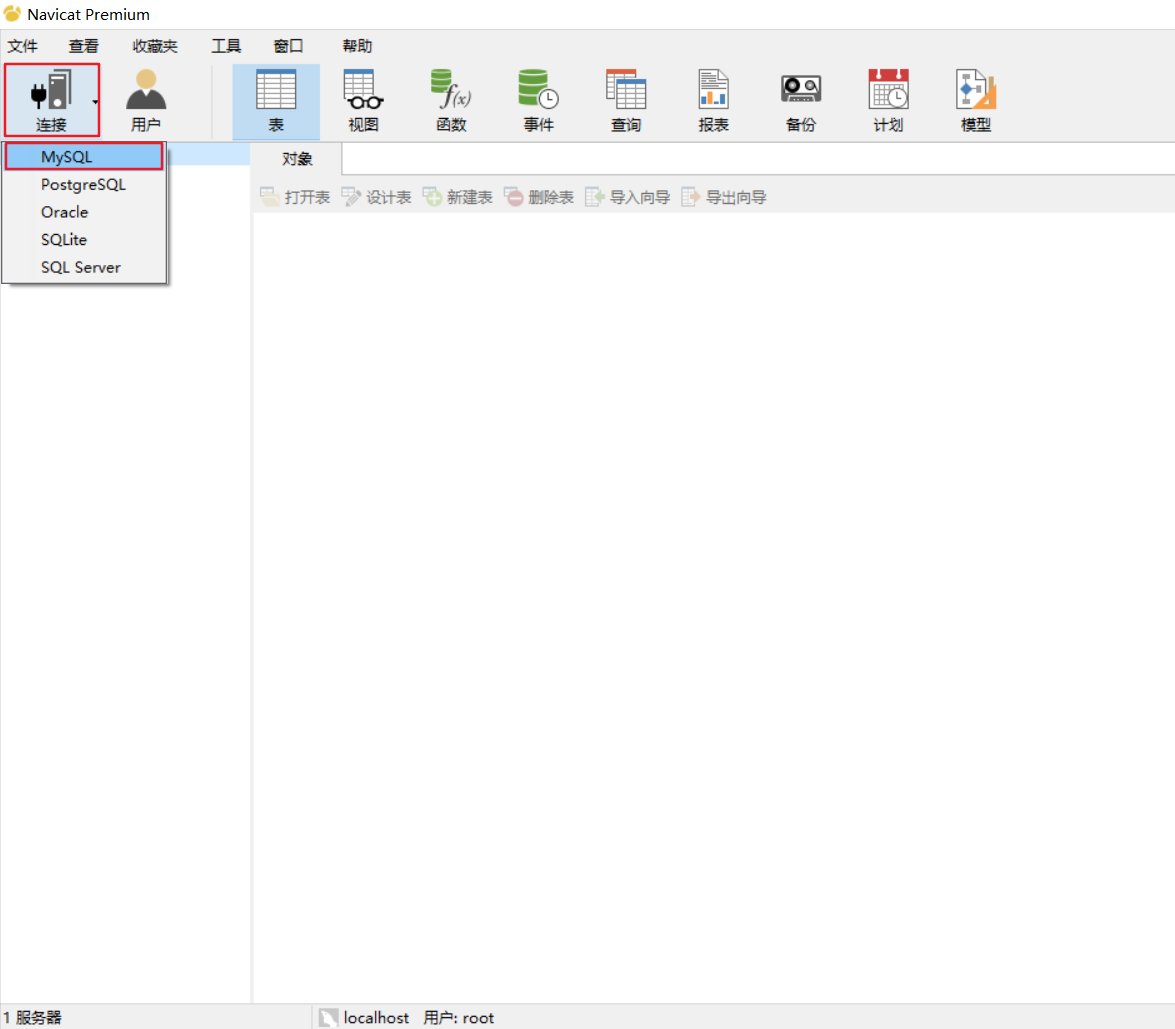


图4.10图形化界面

2、在打开的新建连接对话框中输入连接名和密码，验证端口号等信息，点击【连接测试】按钮，如果提示连接成功，就说明连接上了，如下图4.11所示（如果提示连接不成功，有可能密码输入错误或者是MySQL的服务没有启动，并且root用户初始是没有密码的）。



图4.11连接成功界面

3、连接成功之后我们可以在连接列表里看到我们刚才创建的连接，现在就可以创建数据库了，双击我们刚才创建的连接名，可以看到小海豚变绿了，此时右键单击小海豚，点击【新建数据库】，如下图4.12所示。

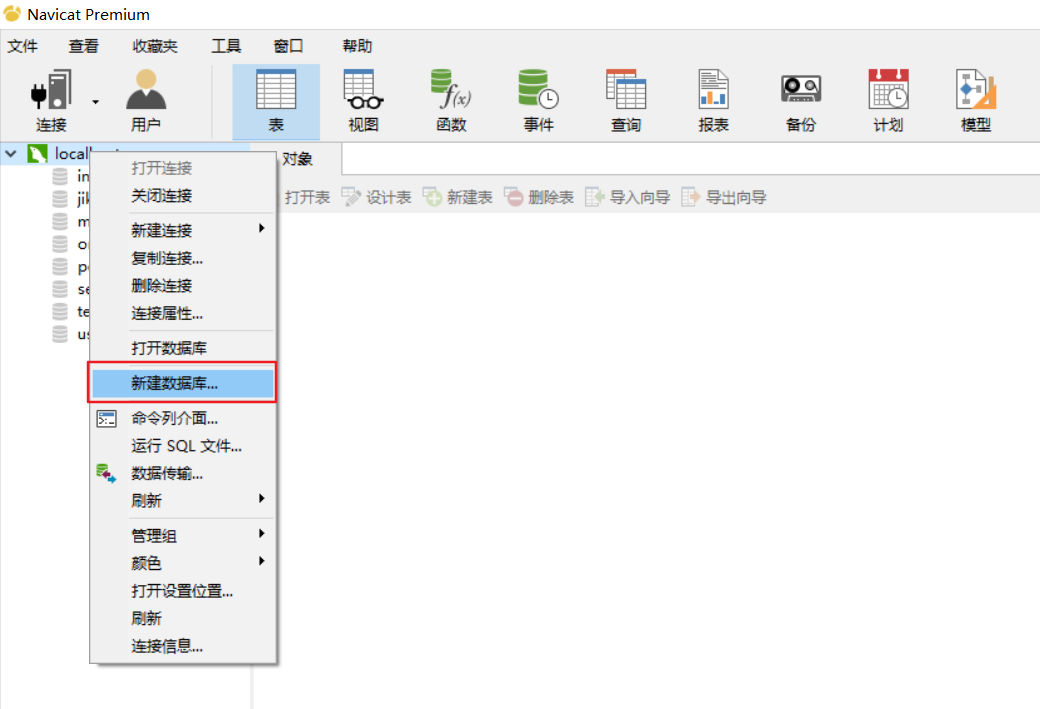


图4.12新建数据库界面

4、填写数据库信息，点击【确定】（填写数据库名即可，字符集和排序规则系统会默认生成一个），如下图4.13所示。

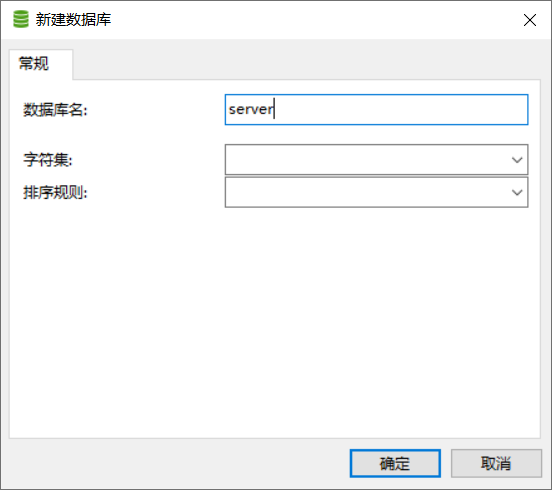
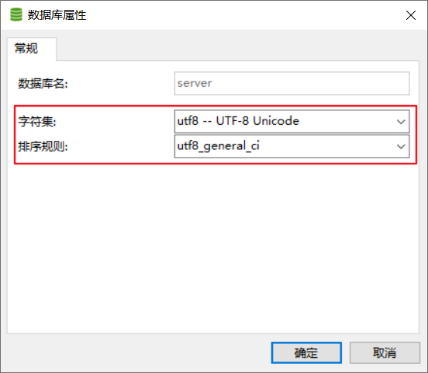


图4.13新建数据库基本信息

5、数据库创建成功，在连接下可以看到刚才创建的数据库server。

6、数据库创建成功之后我们在这个数据库下面创建表，点击数据库server下的表，右侧对象页点击新建表。

7、在这个页面上填入表字段信息，卡片部分是对应字段的信息，如下图4.14所示。

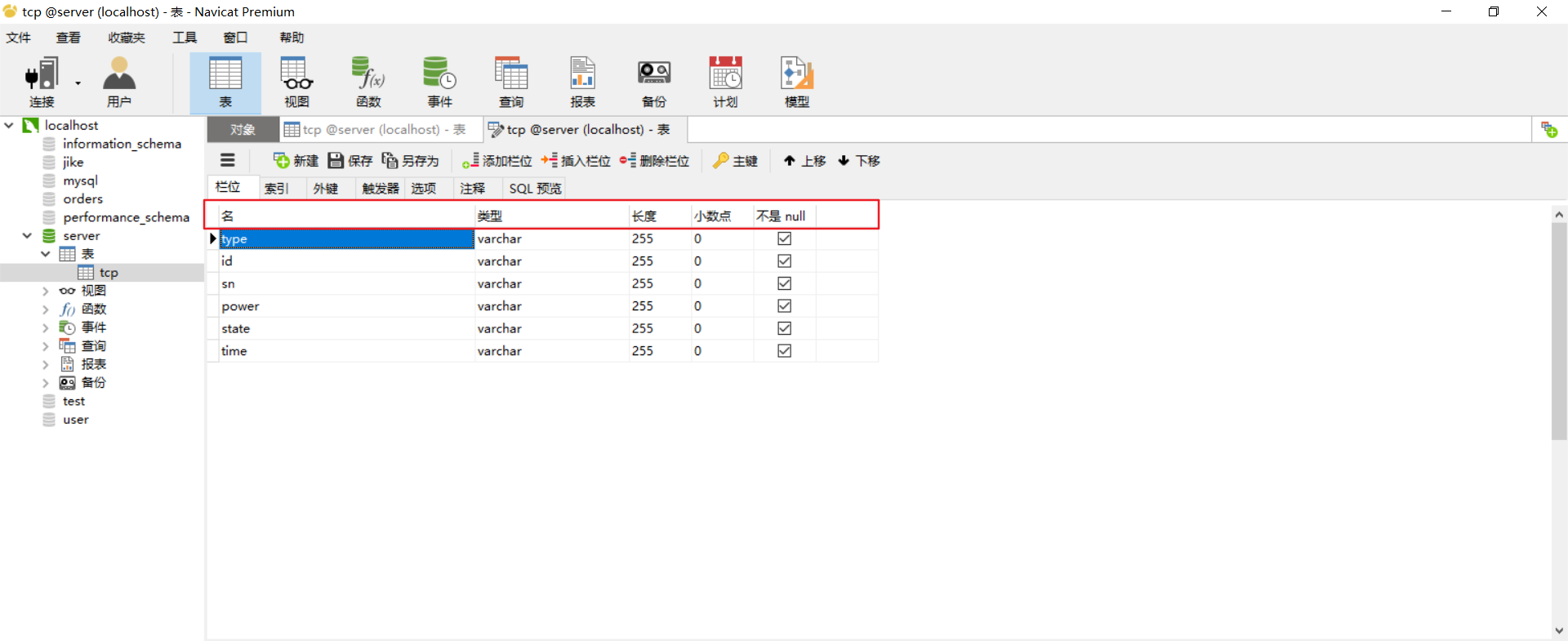


图4.14设计数据库

8、点击上方【保存】按钮，弹出设置表名提示框，输入表名tcp。

9、表创建成功，在数据库下可以看到我们刚才创建的表tcp，如下图4.15所示。

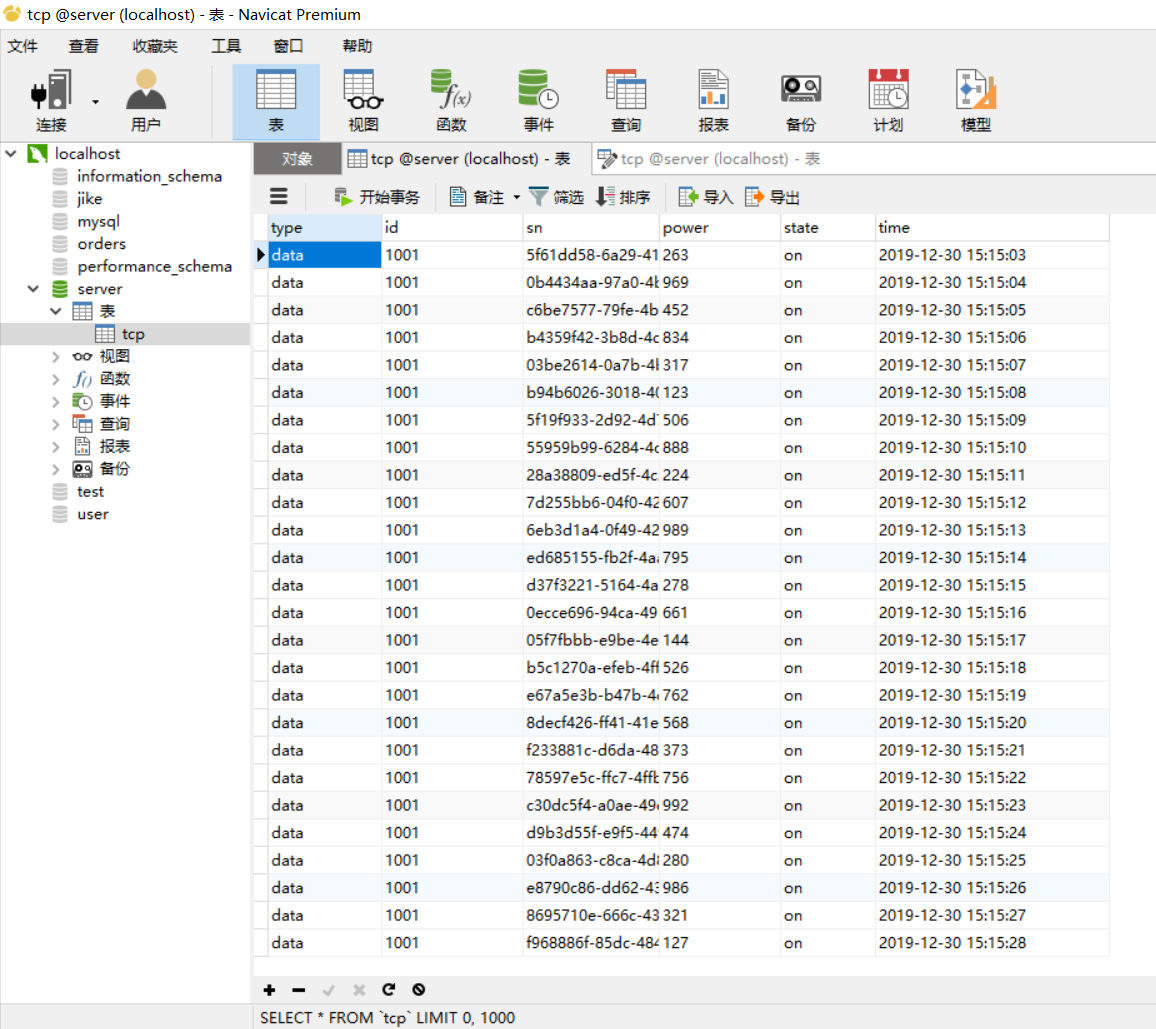


图4.15表中的基本信息

4.2.2 数据库的连接

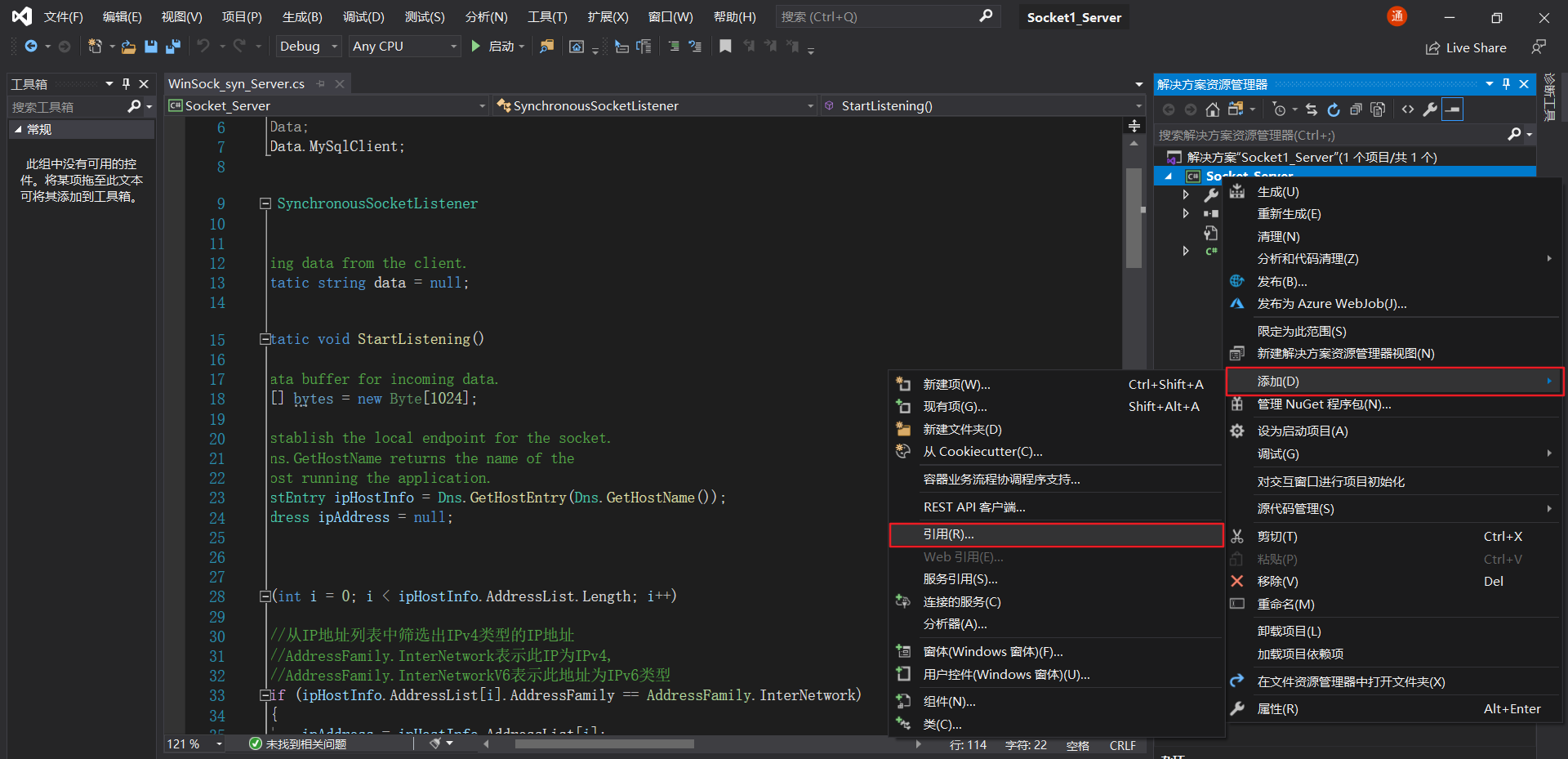
数据库表建立好后，我们现在开始在Microsoft Visual Studio 2019中编写具体的代码，将编写好的server端与后端的MySql数据库相连接。

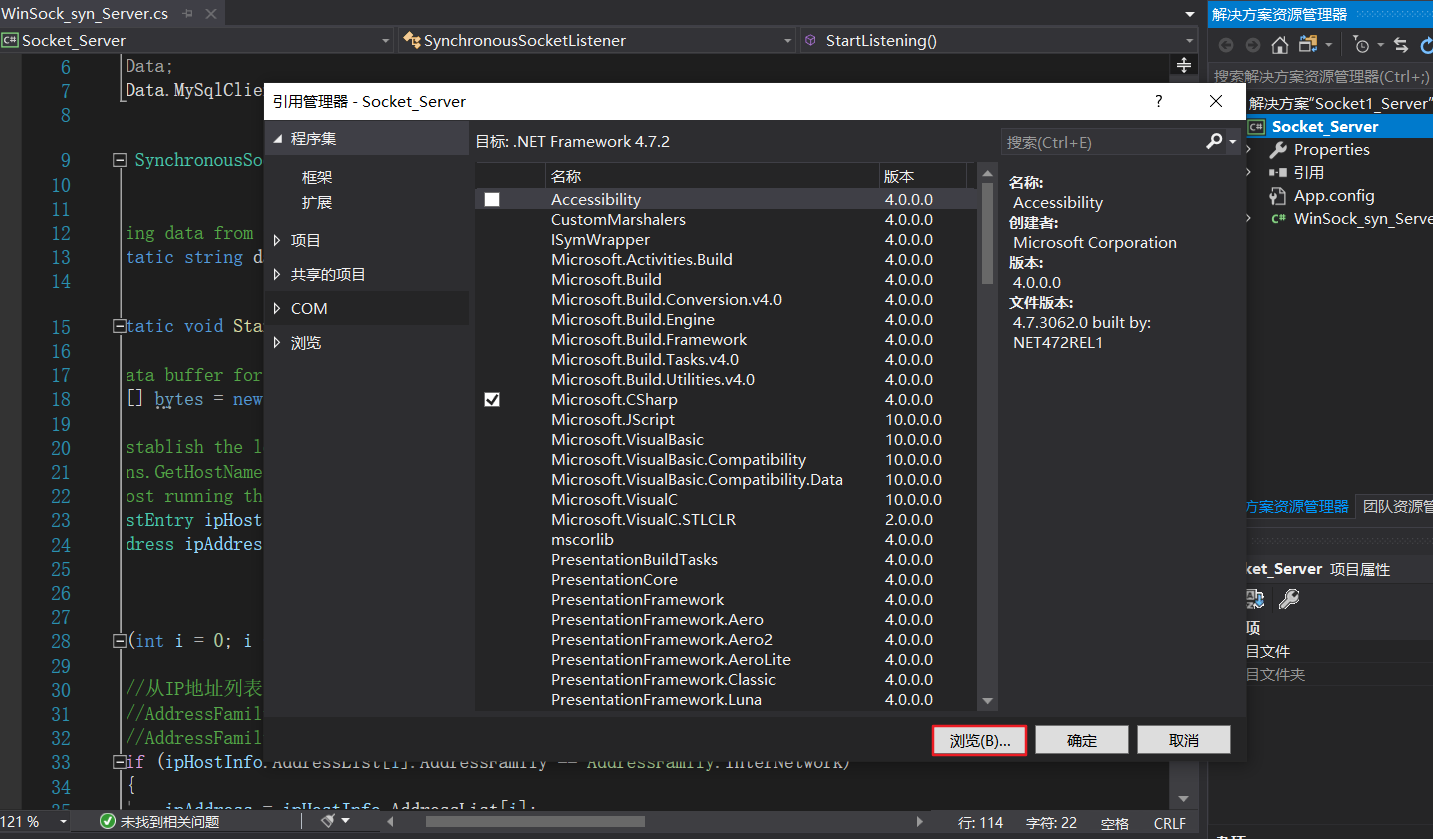
1.添加MySql.Data.dll引用连接数据库：

如果是没有添加MySql.Data.dll引用导致无法连接数据库，那么只需要添加这个引用就可以了，具体方法如下：

1. 添加动态链接库文件

在官网下载connector/net(连接：[http://dev.mysql.com/downloads/connector/net/](https://link.zhihu.com/?target=http%3A//dev.mysql.com/downloads/connector/net/)), 我下载的是8.0.18版本。下载安装后即可在项目中添加MySql.Data.dll驱动文件的引用了。在项目中右键引用🡪添加引用🡪浏览🡪选中MySql.Data.dll文件🡪确定。操作如下图4.16所示：





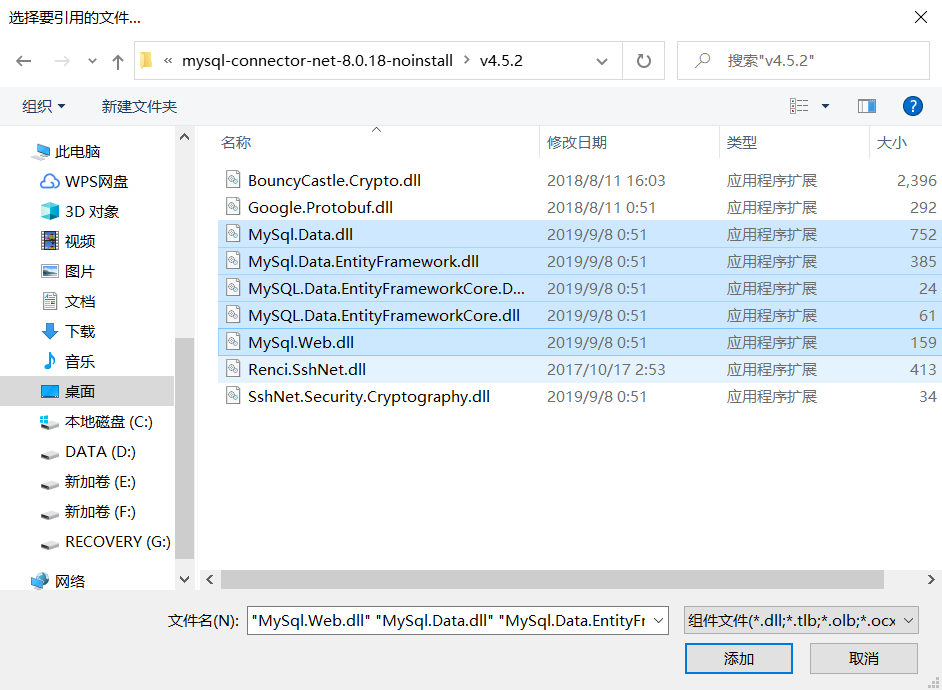


图4.16导入MySql数据包

1. 添加引用后需要在代码页添加using MySql.Data.MySqlClient，如下图4.17所示：

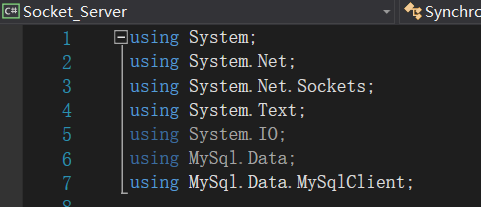


图4.17代码页

1. 连接数据库具体代码如下图4.18所示。string str = “server=localhost; User Id = root; password = root; Database = server”是连接MySQL字符串。其中server=localhost表示的是建立在本地的数据库，也可以写127.0.0.1; User Id指的是新建MySQL连接时的用户名为root，password是本机的密码，Database指的是建立在Navicat Premium中的数据库名。MySqlConnection mycon = new MySqlConnection(str)是为了实例化一个MySqlConnection类型的变量mycon用于开启与关闭数据库。因为本节是为了实现将server端接收到的能量数据插入到自建的数据库中，所以本段代码的核心为：insert into tcp(type, id, sn, power, state, time)，其中tcp为表名。最后用if语句来判断插入数据库是否成功。

string str = "server=localhost; User Id=root; password=root; Database=server";

//连接MySQL的字符串

MySqlConnection mycon = new MySqlConnection(str);//实例化链接

mycon.Open();//开启连接

MySqlCommand mycmd = new MySqlCommand("insert into tcp(type,id,sn,power,state,time) values('" + type1 + "','" + id1 + "','" + sn1 + "','" + power1 + "','" + state1 + "','" + time1 + "')", mycon);

if (mycmd.ExecuteNonQuery() > 0)

{

Console.WriteLine("数据插入成功！");

}

mycon.Close();//关闭

图4.18数据库连接

4.2.3 能量数据存入数据库的方式

在上文中我们已经成功建立了数据库和设计表，并在已经写好的server端中成功与数据库取得连接。但是如何将能量数据存入数据库，以什么样的格式存入数据库还是我们需要思考的问题。在我们用C#编写的client端代码中，我们将能量数据以{"type":"data", "id":"1001", "sn":"DNPLK11FNK", "power":"1200", "state":"on", "time":"2016-04-18 08:31:58"}的形式发向server端的控制台，结果如下图4.19所示。

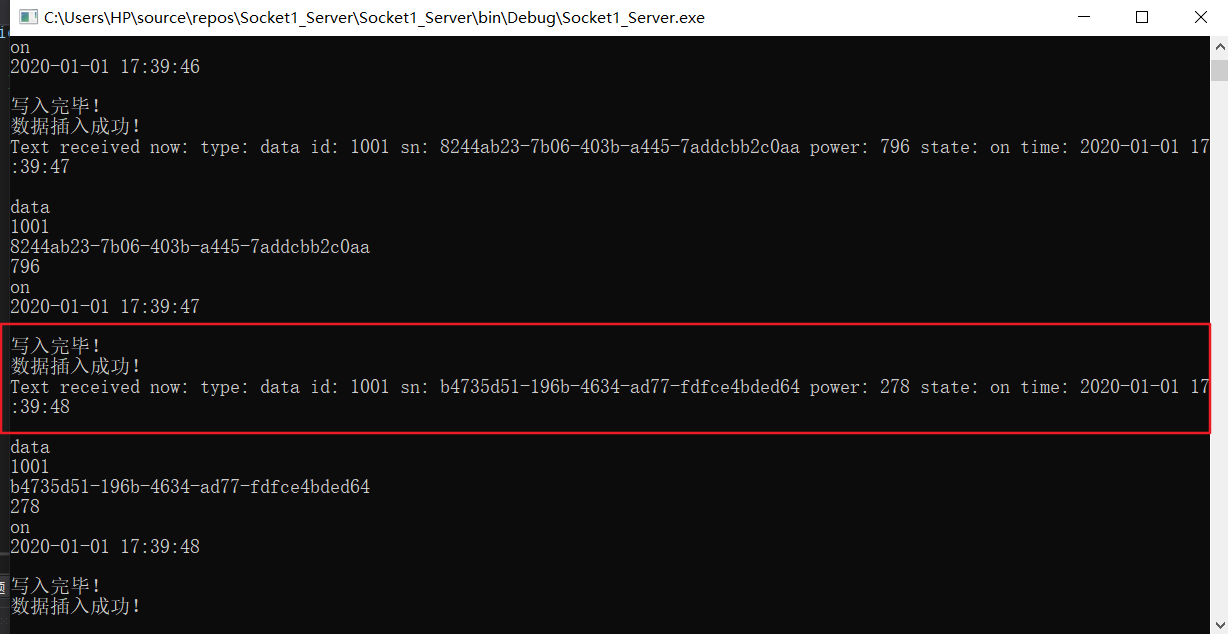


图4.19控制台中字符串

我们需要思考的是，如何将字符串按照我们设计的表中type、id、sn、power、state、time的形式依次存入数据库。这里我们利用C#中的Split()函数将能量数据存入数据库中，具体代码如下图4.20所示。System.Text.Encoding.Default.GetBytes(s)意思是将字符串s按照当前系统的编码存入data数组中。然后我们利用Socket编程中的函数clientSocket.Send()，将字符串s发到server端进行分割。在如下图的代码中我们利用Split()函数，以空格为分隔符，将数组中的奇数位依次存入type1、id1、sn1、power1、state1、time1字符串中，然后输出到控制台中，最后利用insert语句将控制台中的信息存入到数据库中。

//获取随机数

string s = "type:" + " data " + "id:" + " 1001 " + "sn: " + guid + " power: " + n + " state: " + "on" + " time: " + strFu + "\r\n";

byte[] data = System.Text.Encoding.Default.GetBytes(s);

clientSocket.Send(data);

Console.WriteLine("Send Complete");

string type1 = msgnow.Split(' ')[1];

string id1 = msgnow.Split(' ')[3];

string sn1 = msgnow.Split(' ')[5];

string power1 = msgnow.Split(' ')[7];

string state1 = msgnow.Split(' ')[9];

string time1 = msgnow.Split(' ')[11] + ' ' + msgnow.Split(' ')[12];

Console.WriteLine(type1);

Console.WriteLine(id1);

Console.WriteLine(sn1);

Console.WriteLine(power1);

Console.WriteLine(state1);

Console.WriteLine(time1);

Console.WriteLine("写入完毕！");

图4.20 Split()函数分割字符串

最后存取成功的结果如下图4.21所示：

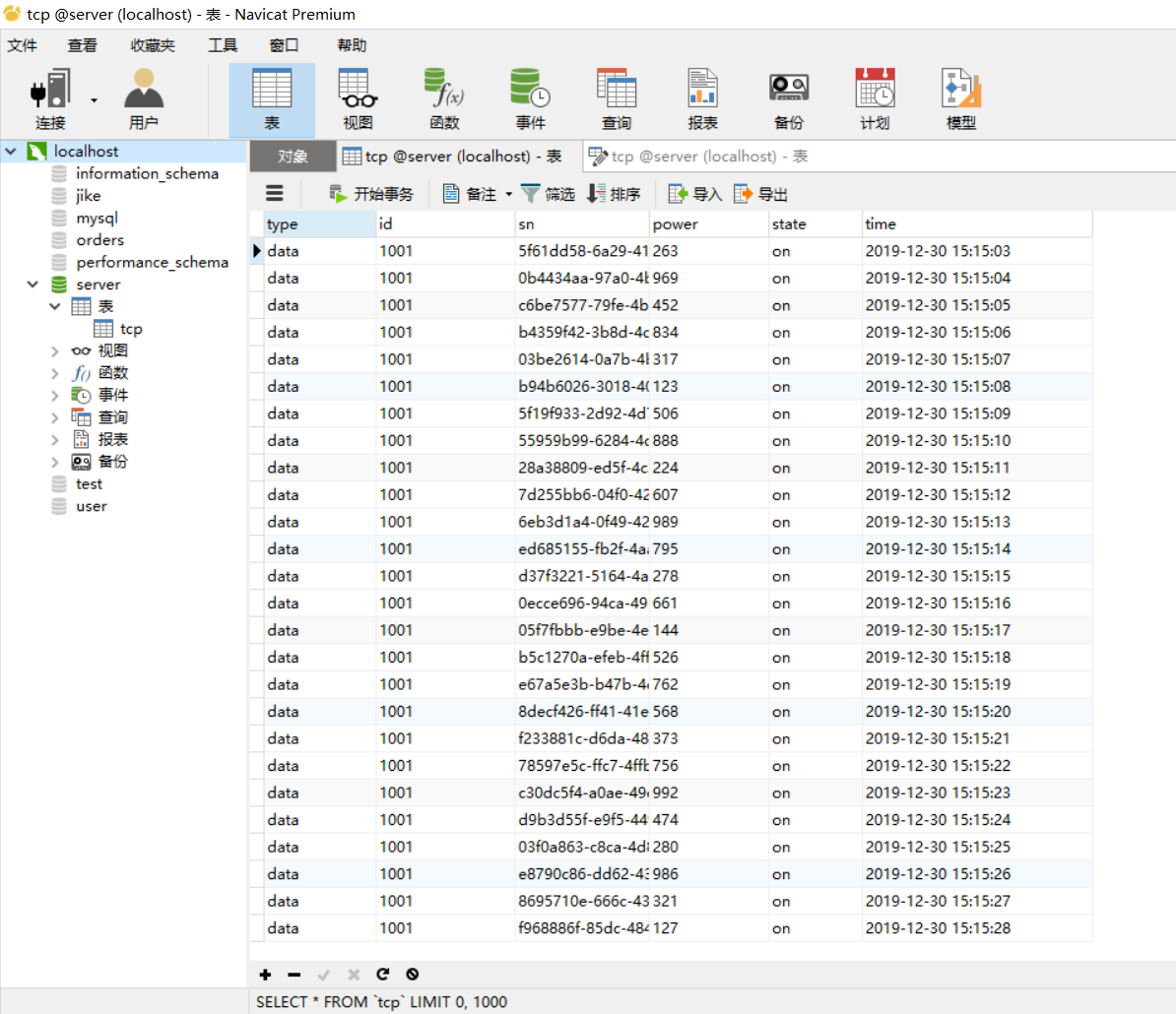


图4.21 数据库中的结果

### 4.2.4 数据库中改进的问题

在一开始分析字符串的时候，我没有利用Split()函数，而是准备通过引入Json包的方式调用using Newtonsoft.Json语句，然后利用C#语句string[] strdatass = new string[6]和Root rt = JsonConvert.DeserializeObject<Root>(strdata); 控制字符串的序列化，建立序列化对象rt实现对字符串的分组。但是，在程序运行的过程中一直有“无法分析boolean型变量”的问题。所以我通过查找众多博客找到了Split()函数，利用String.Split方法中的重载函数解决了字符串分割的问题。

## 4.3服务端和客户端的双向通信

服务端和客户端的双向通信，在基于服务端接收完客户端的信息后，需要向客户端发送反馈信息，以告知客户端，信息已经成功被接收，建立此反向通信机制后，也可通过从服务端发送相关控制信息给客户端，来对客户端进行相关操作。

### 4.3.1服务端发送简单反馈信息

在服务端接收完毕客户端数据，并将数据存入数据库成功后。服务端开始发送反馈信息

1、服务端发送：

**byte**[] dataToC = System.Text.Encoding.**ASCII**.GetBytes(**"Received successfully"**);  
handler.Send(dataToC);

2、客户端接收：

**byte**[] bytes = **new** Byte[1024];  
**int** bytesRec = clientSocket.Receive(bytes);  
**string** dataBack = Encoding.**ASCII**.GetString(bytes, 0, bytesRec);  
Console.WriteLine(dataBack);

就此即可完成服务端的自定义语句，在客户端接收并显示。

3.设计思想：在服务端建立控制开关机制，以判断向客户端传输的具体反馈信息，客户端根据收到的反馈信息判断具体的执行任务。

服务端控制函数：

初步采用命令台内，键盘输入进行控制，在每一次循环建立端口监听时，都进行对键盘的监听，键盘输入相应的终止按键，则switch语句便执行相应的终止语句，否则默认跳出判断语句。以下为相应代码。

*//非阻塞式监听键盘输入***if** (Console.**KeyAvailable**)  
{  
 ConsoleKeyInfo key = Console.ReadKey(**true**);  
 **switch** (key.**Key**)  
 {  
 **case** ConsoleKey.***F2***:  
 Console.WriteLine(**"You pressed F2!"**);  
 **byte**[] data = System.Text.Encoding.**ASCII**.GetBytes(**"ShutDown"**);  
 handler.Send(data);  
 **break**;  
 **default**:  
 **break**;  
 }  
}

客户端判断函数：

**if** (dataBack == **"ShutDown"**)  
{  
 clientSocket.Close();  
}

进一步优化：

为便于连接GUI控制，应将判断内容改为具体的某一个变量，而非对键盘输入的监听，在服务端与客户端都应以语句中的state的值为“on”或“off”来控制客户端的状态。

## 4.4利用JAVA和MySQL实现数据的动态实时监控

设计思想：在服务端分割信息存入数据库后，用JAVA连接数据库，运用JfreeChart实现设备电压变化的可视化以及动态更新。

主要步骤如下：

### 4.4.1设计DBUtil工具类模块。

用于连接Mysql数据库。

主要代码如下：

Connection conn = **null**;

Class.*forName*("com.mysql.cj.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/server?serverTimezone=UTC";

String username = "root";

String password = "\*\*\*\*\*\*";

conn = DriverManager.*getConnection*(url,username,password);

并包裹try/catch异常获取，打印错误提示以及错误信息。

### 4.4.2设计Entity模块，

用于创造存储Mysql中每条数据的类，并构造get函数用于获取确定信息，构造set函数用于设定或者修改具体信息。

类内的具体属性值如以下代码，一一对应于信息传递中规定的信息格式，即数据库中的数据格式，全部定义为String字符串类型。为了避免外部的随意干涉，设定为private私有属性。

**public** **class** Entity {

**private** String type;

**private** String id;

**private** String sn;

**private** String power;

**public** String state;

**private** String time;

为了方便对类内的属性进行获取和设定，需要对每个属性都构造get以及set函数，以下为Type属性的get以及set函数，其余省略，不再赘述。

**public** String getType() {

**return** type;

}

**public** **void** setType(String type) {

t**his**.type = type;

}

…………

}

### 4.4.3设计DAO模块

用于构造各类与数据库相关的功能函数，比如getAll() 获取数据库中的全部数据，getTen() 获取数据库中最新的十条数据，以及update() 修改函数，用于修改数据库中的某条数据。

以下为主要源代码，因为获取的结果为多个类，所以设定获取函数的类型为java中的ArrayList数组链表，此函数既拥有数组的特性，也拥有链表的特性，极为方便。

**public** ArrayList<Entity> getAll(){

ArrayList<Entity> ar = **new** ArrayList<Entity>();

设定conn调用DBUtil中的连接数据库函数，ps为执行语句函数，rs为结果存放函数。

Connection conn = DBUtil.*getConnection*();

PreparedStatement ps = **null**;

ResultSet rs = **null**;

设定sql语句如下，并执行，且存放执行结果。

String sql = "select type,id,sn,power,state,time from tcp";

ps = conn.prepareStatement(sql);

rs = ps.executeQuery();

以while循环不断调用Entity中的set函数，保存获取的数据库信息至新建的ArrayList类中。

**while**(rs.next()) {

Entity ent = **new** Entity();

ent.setType(rs.getString("type"));

ent.setId(rs.getString("id"));

ent.setSn(rs.getString("sn"));

ent.setPower(rs.getString("power"));

ent.setState(rs.getString("state"));

ent.setTime(rs.getString("time"));

ar.add(ent);

}

至此数据全部保存在ArrayList ar中。最后return ar函数即可获得结果。需要注意的是一定要记得释放rs、ps、conn三个函数，并且注意要依次释放。

将SQL语句改为

"select \* from tcp Order By time Desc limit 10"

即可编写获取最新的十个数据的功能函数 getTen()，为了给gui控制界面做准备，也写了一个修改函数update，用以从java更改数据库中的某个ID的设备的状态，以进行开启和关闭操作，具体仍有待实现，此为java数据库接口，主要源码如下：

String sql = "UPDATE tcp SET state= ? WHERE id=?";

ps = conn.prepareStatement(sql);

ps.setString(1, ent.getState());

### 4.4.4设计Charts模块

用于将数据通过JfreeChart实现数据库中的信息可视化，绘制成折线图显示近十条数据的变化趋势。并构造定时器，实现每隔一秒动态刷新，重新获取最新的十条数据库内数据。

JFreeChart是JAVA平台上的一个开放的图表绘制类库。它完全使用JAVA语言编写，是为applications, applets, servlets 以及JSP等使用所设计。JFreeChart可生成饼图（pie charts）、柱状图（bar charts）、散点图（scatter plots）、时序图（time series）、甘特图（Gantt charts）等等多种图表，并且可以产生PNG和JPEG格式的输出。

本次我们使用它的折线图绘制。要使用JfreeChart需要官网下载Jcommon以及JfreeChart两个包，目前最新的包为jcommon-1.0.23.jar以及jfreechart-1.0.19.jar下载完毕后导入工程，并构造路径，即可使用JfreeChart。

通过StandardChartTheme 可以设置Chart的外观格式，如字体大小等。

StandardChartTheme mChartTheme = **new** StandardChartTheme("CN");

mChartTheme.setLargeFont(**new** Font("黑体", Font.***BOLD***, 20));

mChartTheme.setExtraLargeFont(**new** Font("宋体", Font.***PLAIN***, 15));

mChartTheme.setRegularFont(**new** Font("宋体", Font.***PLAIN***, 15));

通过for循环依次设定折线图的数据集。

ArrayList<Entity> ar=**new** DAO().getTen();

**int** index = ar.size()-1;

**for**(Entity ne; index>=0 ;index--) {

ne = ar.get(index);

mDataset.addValue(Double.*valueOf*(ne.getPower()), "设备"+ne.getId(), ne.getTime().split(" ")[1]);

}

新建折线图，设定表标题及x轴y轴及其他相关信息。

JFreeChart mChart = ChartFactory.*createLineChart*(

"电压波动折线图",//图名字

"时间",//横坐标

"电压",//纵坐标

mDataset,//数据集

PlotOrientation.***VERTICAL***,

**true**, // 显示图例

**true**, // 采用标准生成器

**false**);// 是否生成超链接

最终实现以下画面4.22，以电压为纵坐标，时间每秒为横坐标，显示电压波动状况。

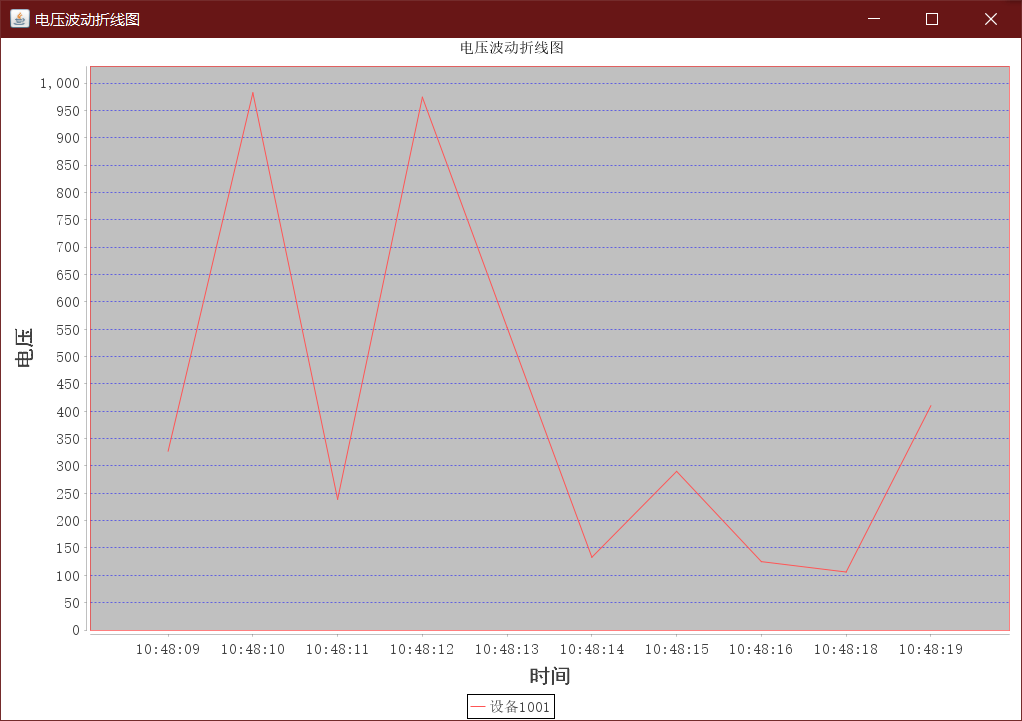


图4.22 动态折线图

将数据集设置函数放置在while无限循环中，并设置线程休眠1000ms，以实现每秒动态更新折线图。

**while**(**true**){

mPlot.setDataset(*GetDataset*());

Thread.*sleep*(1000);

}

### 4.4.5遇到的问题

在动态实现java折线图显示最新的十条数据的时候，由于SQL语句为

select \* from tcp Order By time Desc limit 10

即将数据按时间顺序倒序排序，并输出前十个，这导致了越新的数据会保存在数组的越前面，使输出绘制折线图的时候，最新的数据显示在了折线图的左侧，整个折线图动态更新时，会整体从左向右移动，这不符合传统的用户使用逻辑，所以需要对从数据库中提取出的数据，在赋值给折线图数据集时，进行逆序操作。具体更改如下：

原代码：

ArrayList<Entity> ar=**new** DAO().getTen();

**for**(Entity ne:ar) {

mDataset.addValue(Double.*valueOf*(ne.getPower()), "设备"+ne.getId(), ne.getTime().split(" ")[1]);

}

更改后代码：

ArrayList<Entity> ar=**new** DAO().getTen();

**int** index = ar.size()-1;

**for**(Entity ne; index>=0 ;index--) {

ne = ar.get(index);

mDataset.addValue(Double.*valueOf*(ne.getPower()), "设备"+ne.getId(), ne.getTime().split(" ")[1]);

}

先用size()获取数组大小，再以for循环依次从数组最末尾依次递减，进行数据集添加。更改后实现了动态刷新的方向更改。

## 4.5 瞬时电功率可视化实时动态监控

4.5.1 可视化实时动态监控平台的选取

1、实现可视化实时动态监控的前提条件是成功连接数据库并将数据库中所保存的瞬时电功率值取出，最后通过一系列的画图函数将其显示在图形窗口中。由于在大二学年学习过Java的相关课程，所以对其较为熟悉，且已经基本具备连接MySQL数据库的条件，所以首先选用Java环境实现瞬时电功率可视化实时动态监控。

2、在Java数据库连接数据库出现了一定的阻碍后，由于参加了一整年数学建模培训和比赛的缘故，再加上这里的瞬时电功率实质上就是一组存在数据库中的数字，要对数字进行一系列的运算和可视化Matlab无疑是不二之选，唯一的问题就是我原来没有尝试过用Matlab连接数据库，不知道其是否能够实现及实现的难度较Java来说又是如何，但是想再多也没用先查阅资料动手试试再说。

4.5.2 Matlab连接MySQL数据库

Matlab提供了两种方式能够实现连接MySQL数据库，分别为利用ODBC数据源方式连接MySql数据库和利用JDBC驱动连接MySQL数据库。经过简要了解比较后我选择了步骤较为简单的利用JDBC驱动连接MySQL数据库。

1、要利用JDBC驱动连接MySQL数据库，最首要的就是要加载MySQL的JDBC驱动。

（1）下载MySQL中JDBC的相关JAR包（mysql-connector-java-5.1.46-bin.jar）。

（2）将mysql-connector-java-5.1.46-bin.jar拷贝到Matlab安装文件夹下的java文件夹的jar文件夹的toolbox文件夹中，拿我的电脑举例就是（F:\MATLAB\R2017a\java\jar\toolbox）。（图4.23）若找不到当时Matlab的安装路径，可以在命令行窗口中使用指令”matlabroot”找到其准确的安装路径。（图4.24）

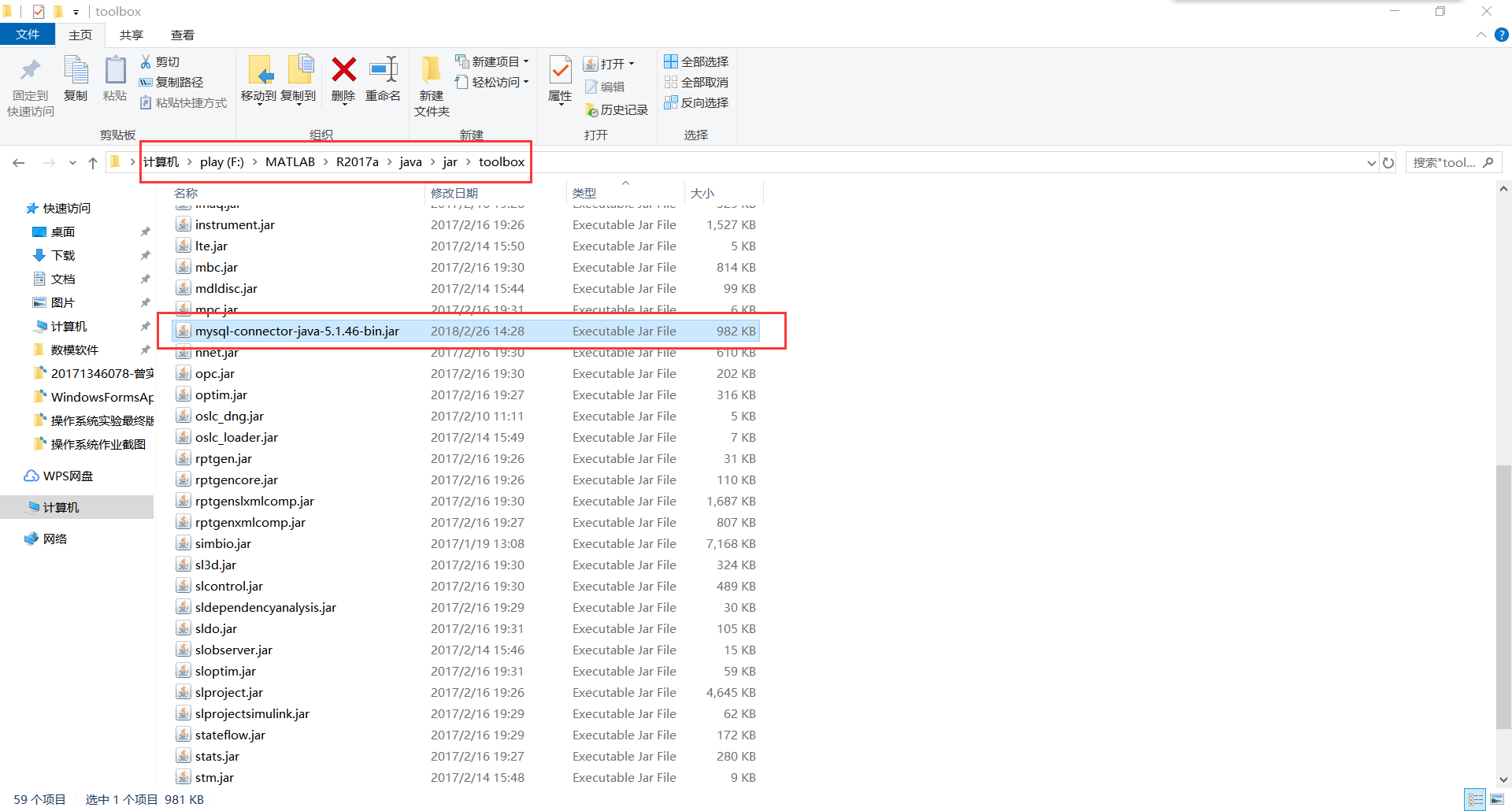


图4.23 导入JDBC的相关JAR包图

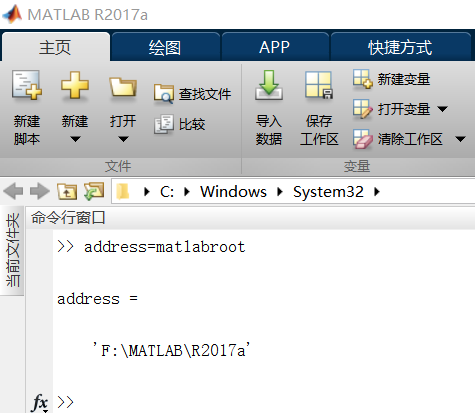


图4.24 根目录路径查询图

（3）打开根目录下的toolbox/local目录并在其中找到classpath.txt文件（图4.25），向其中添加用来加载MySQL的JDBC驱动的语句，使Matlab启动自动加载步骤二中所添加的JAR包，语句如下：”$matlabroot/java/jar/toolbox/mysql-connector-java-5.1.46-bin.jar”（图4.26）。

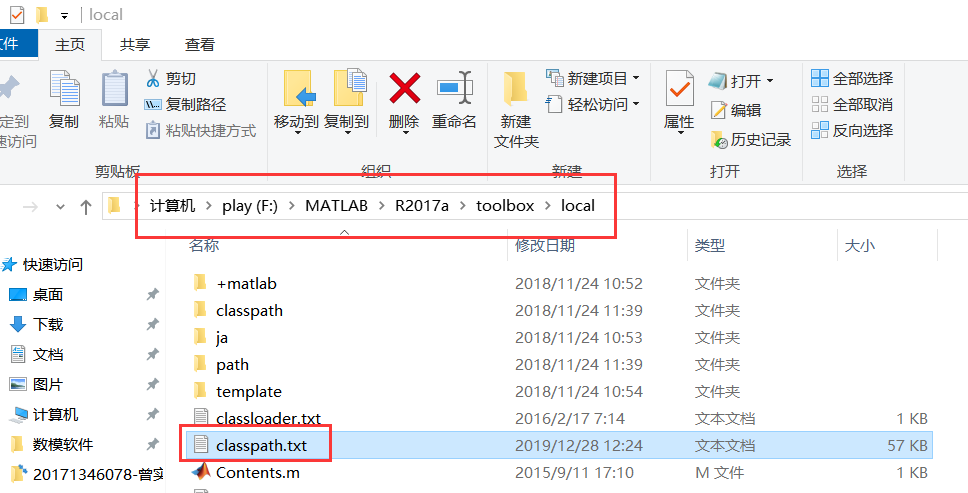


图4.25 找到存放驱动语句的相关文件图

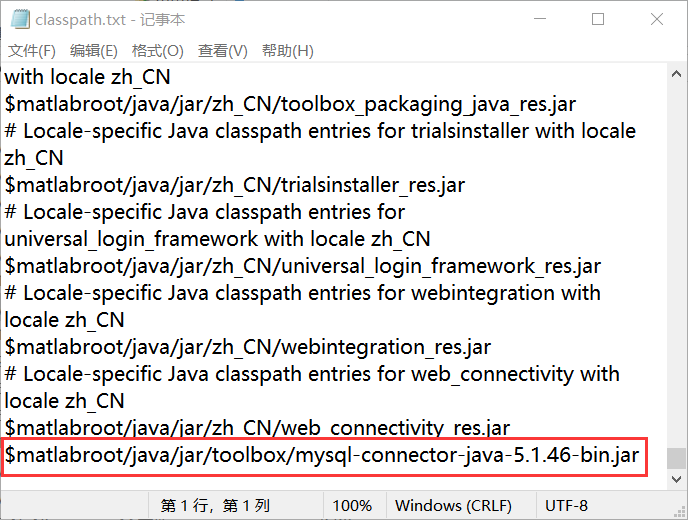


图4.26 导入JDBC相关驱动语句图

（4）重启Matlab软件，使其通过刚刚所添加的用来加载MySQL的JDBC的JAR包的语句来自动加载MySQL的JDBC驱动，从而为利用JDBC驱动连接MySQL数据库配置好环境。

2、其次要在Matlab的脚本文件中通过和MySQL数据库建立连接的相关语句编写程序段，实现数据库的连接。相关代码段如下：

conn = database('databasename', 'username', 'password', 'driver', 'databaseurl')

其中参数说明如下：

databasename: 数据库名称。（'server'）

username: 用户名。（ 'root'）

password: 密码。

driver: JDBC驱动器。（'com.mysql.jdbc.Driver'）

databaseurl: 类似于jdbc:subprotocol:subname。（'jdbc:mysql://localhost:3306/server'）

3、运行连接MySQL数据库的相关”database”语句建立与所创建数据库的连接，在成功建立连接后将会在命令行窗口输出与所连接的数据库相关的一系列信息，此时Matlab语句同数据库建立了连接（图4.27）。



图4.27 数据库连接成功图

4.5.3 Matlab实现动态可视化监控

1、在成功连接数据库后，使用语句”curs=exec(conn,'select \* from tcp limit 1000');”对所创建的表名为tcp的表实现查询操作，紧接着使用语句”d=fetch(curs);”获取查询结果（图4.28）, 最后使用语句”data=d.Data;”将查询结果的数据传入变量data中（图4.29）。

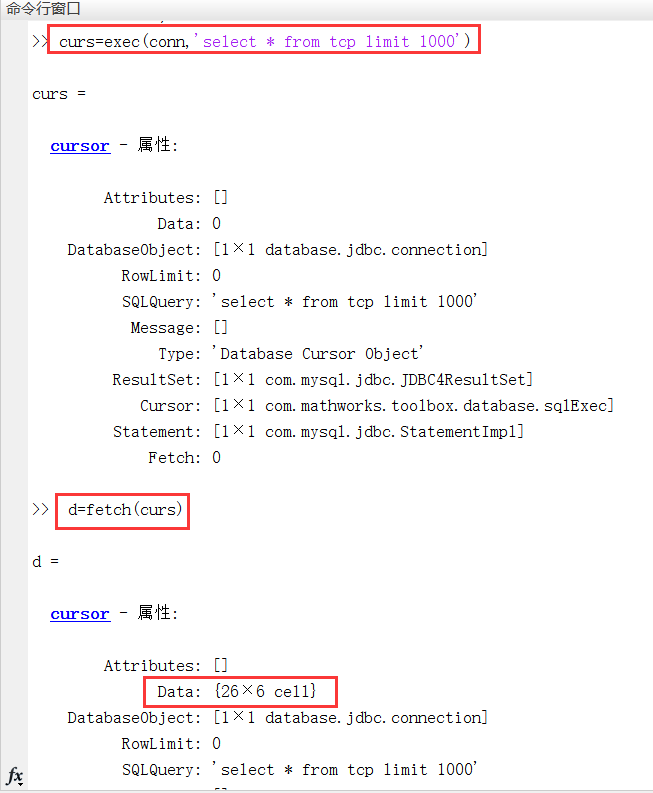


图4.28 获取查询结果图

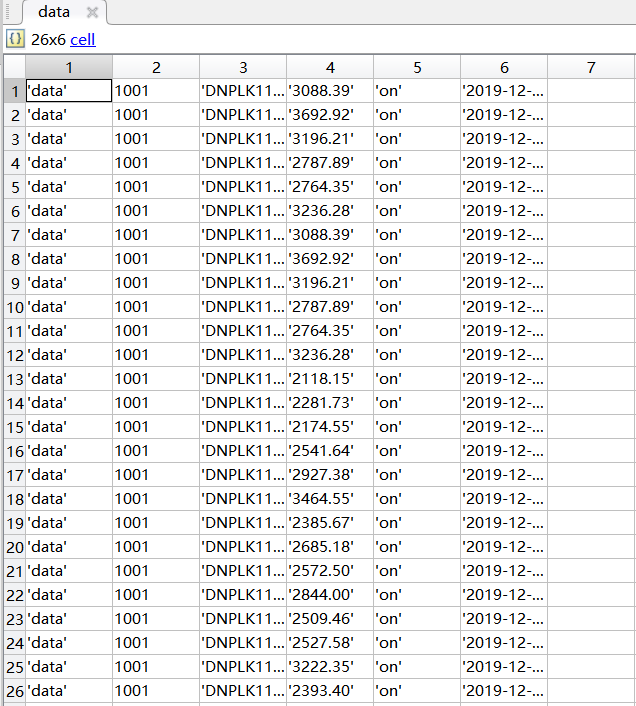


图4.29 获取表中数据图

2、在将数据库中的数据提取到data变量中后，从上图可以看出此时的变量类型为cell类型，无法对其操作，必须通过语句”cell=data(:,4);cell\_str=cell2mat(cell);str\_num=str2num(cell\_str);”

将其从cell类型转化为double类型并筛选出瞬时电功率一列，才能实现一系列可视化操作。

3、在格式转化完毕后，首先简单通过plot对库中现有的数据进行静态可视化操作（图4.30）。

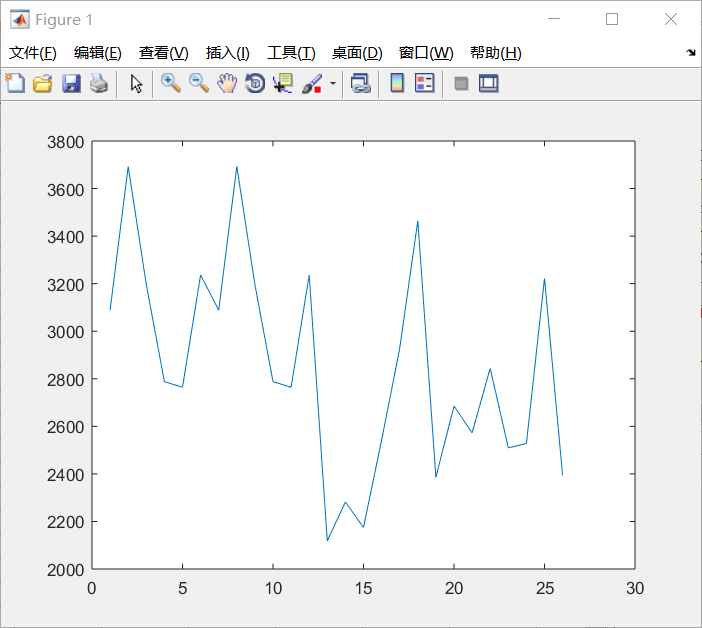


图4.30 瞬时电功率静态可视化图

4、利用Matlab中自带的drawnow命令实现屏幕的刷新。drawnow 更新图窗并处理任何挂起的回调。使用该命令可以修改图形对象并且在屏幕上立即查看这次更新。此处我们将数据库的记录数目装入变量m之中并设立循环”t=1:1:m”实现对数据库现有各记录的遍历，利用plot函数和hold on命令实现图形窗口的停滞并用drawnow命令实现屏幕的刷新，最终人肉眼所看到的就是图形随着数据库中数据的更新而实时的监控更新（图4.31）。

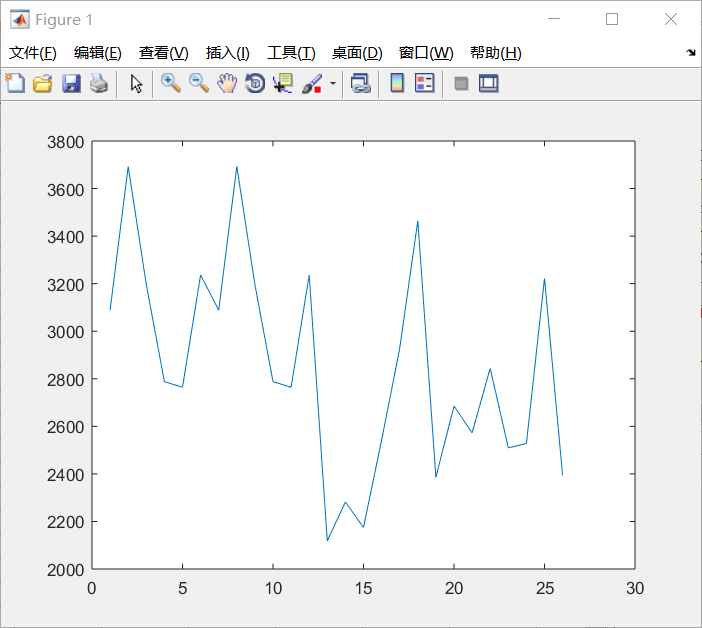


图4.31 瞬时电功率实时可视化监控图

4.5.4 C#调用Matlab，实现Matlab图形Figure嵌入Winform窗体

1、首先第一步建立一个工程文件。

2、添加Matlab Application(Version 9.3)Type libaray。在References这里右键点击——Add References，然后会弹出如图所示的对话框，按照图片所示的顺序操作，（1） 选择COM（2）输入matlab进行搜索（3） 选择（这里要和你的Matlab版本相对应，我安装的是Matlab2017A,对应的是9.3）（图4.32）。

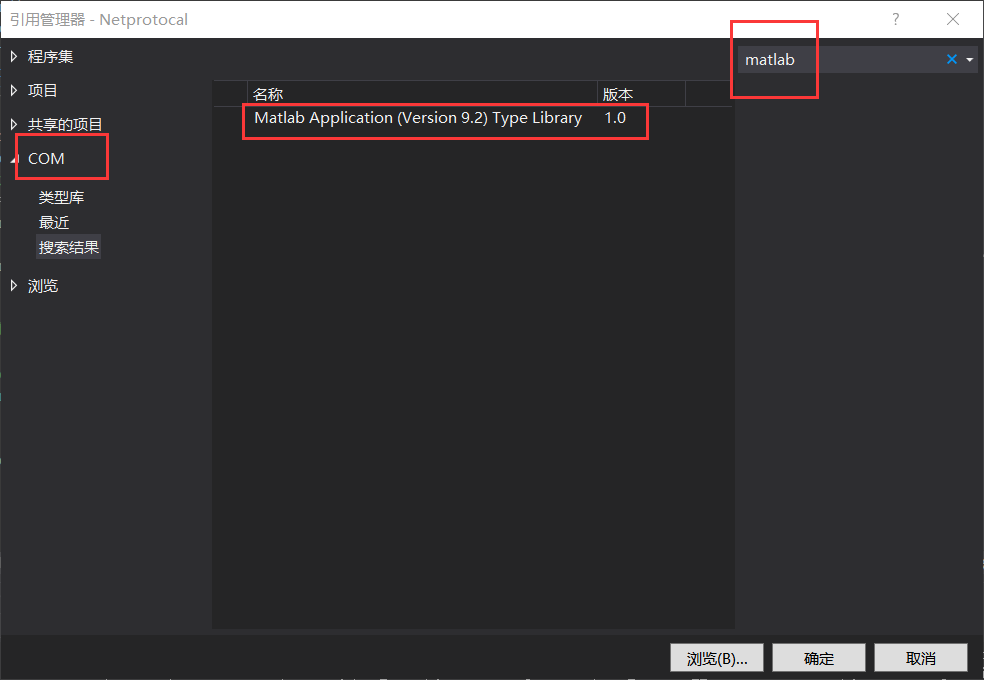


图4.32 Matlab的调用包导入图

添加引用后，引用（References）目录下多了一个MLApp文件（图4.33），如果在编程是采用MLAppClass的话还需要将该文件的属性Embed Interop Types的True值改为False，不然会报错。

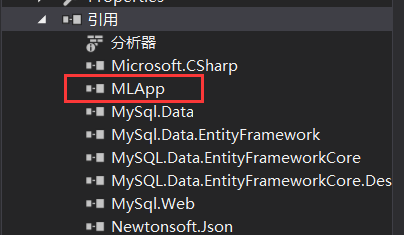


图4.33 Matlab的调用包导入成功图

2、添加Properties。第一步，在项目下面新建一个文本文件，并将其改名，如“ttt.txt”，然后把需要的matlab代码复制进去并保存（图4.34）。第二步，在你的工程名那里，右键选择“Properties”，然后选择Resources——Add Existing File，弹出的对话框之后，选择刚刚添加的txt文件，这样在资源文件夹下面会多出来一个ttt.txt的文件。这样做的目的是：在我们需要调用matlab代码的时候，我们用这个资源文件去生成一个.m文件，然后再执行这个.m文件，结束了再删除.m文件。这样做的原因是如果直接在工程下面粘贴一个.m文件，使用VS自带的打包工具打包发布时， 这个文件是不会一起打包的，就造成了发布后调用不成功。

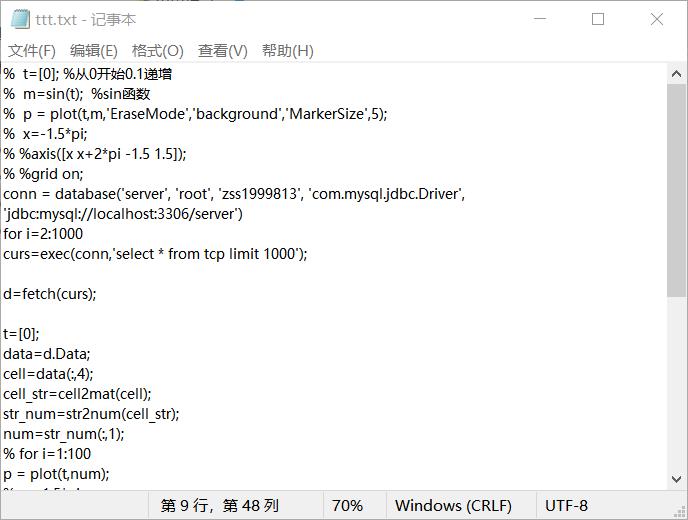


图4.34 Matlab代码文件创建图

3、这里是关键的一步，调用matlab。

MLApp.MLApp matlab = null;

Type matlabAppType = System.Type.GetTypeFromProgID("Matlab.Application");

matlab = System.Activator.CreateInstance(matlabAppType) as MLApp.MLApp;

string path\_project = Directory.GetCurrentDirectory(); //工程文件的路径，如bin下面的debug

string path\_matlab = "cd('" + path\_project + "')"; //自定义matlab工作路径

matlab.Execute(path\_matlab);

matlab.Execute("clear all");

matlab.Execute("close all");

string command;

command = @"ttt(10,4)";

matlab.Execute(command);

4、这一步骤实现将弹出的Figure对话框嵌入到窗体内。

（1）首先在窗体内定义使用的Windows API。

#region //Windows API

[DllImport("user32.dll")]

public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);//

[DllImport("user32.dll")]

public static extern IntPtr SetParent(IntPtr hWndChild, IntPtr hWndNewParent);

[DllImport("user32.dll", CharSet = CharSet.Auto)]

public static extern int MoveWindow(IntPtr hWnd, int x, int y, int nWidth, int nHeight, bool BRePaint);

const int GWL\_STYLE = -16;

const int WS\_CAPTION = 0x00C00000;

const int WS\_THICKFRAME = 0x00040000;

const int WS\_SYSMENU = 0X00080000;

[DllImport("user32")]

private static extern int GetWindowLong(System.IntPtr hwnd, int nIndex);

[DllImport("user32")]

private static extern int SetWindowLong(System.IntPtr hwnd, int index, int newLong);

（2）下一步，定义全局变量，在触发事件内实例化线程，因为这里是点击按钮，所以后面的实例化写在了private void button1\_Click(object sender, EventArgs e)里面。

//定义在窗体内

public delegate void UpdateUI();//委托用于更新UI

Thread startload;//线程用于matlab窗体处理

IntPtr figure1;//图像句柄

//写在触发事件内，如Form\_Load，这里我写在了Button\_Click内

//实例化线程，用来初次调用matlab，并把图像窗体放到winform

startload = new Thread(new ThreadStart(startload\_run));

//运行线程方法

startload.Start();

5、解决只能捕捉一次窗体的问题。上述的方法，只能实现在第一次弹出Figure的时候能够捕捉嵌入，当再次运行的时候就会弹出来。摸索了很久后，发现了解决方案。在你触发线程实例化之前加上这句话：”figure1 = IntPtr.Zero;”，即可把之前的figure1给释放了，，从而就能再次捕捉了。

## 4.6客户端及其对应控制机制实现

4.6.1 客户端

1、首先与之前服务端创建的流程一样，我们先打开Visual Studio 2019，新建一个.NET Framework的C#控制台应用；

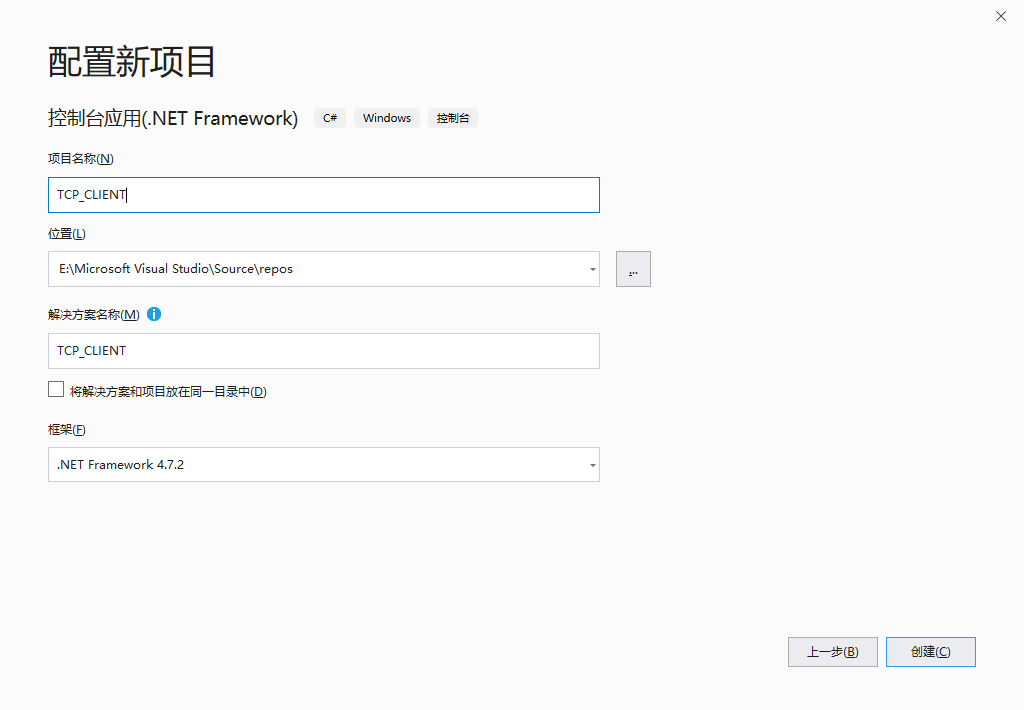


图4.35 新建客户端项目

**2、接着我们要明确设计思路，通过查阅资料，我们知道了面向连接的套接字流程为：**服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。而流式传输的客户端中主要函数：1.socket()函数；2.connect()函数，填写服务端的地址与端口；3.send()函数；4.recv()函数。

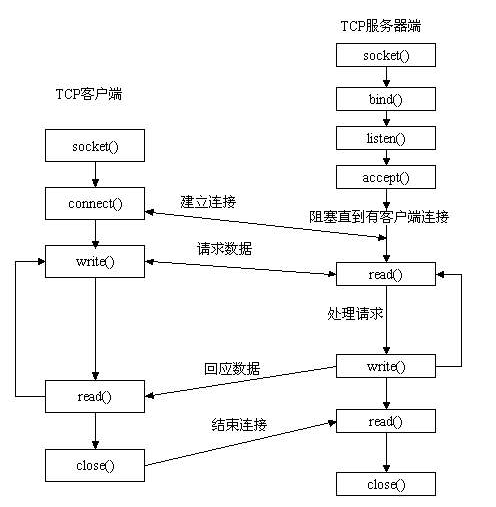


图4.36 面向连接的套接字流程

**3、然后我们就可以开始客户端的代码实现了，实现是命名空间部分，我们在客户端中引入了如下命名空间：Ⅰ.** using System：最基础的系统命名空间；Ⅱ. using System.Text：引入这个命名空间是因为我们后面在客户端与服务器之间传输信息，为了使用Encoding函数而引入的；Ⅲ. using System.Net.Socket：主要提供制作Sockets网络应用程序的相关类，是为了完成实验必不可少的一个命名空间；

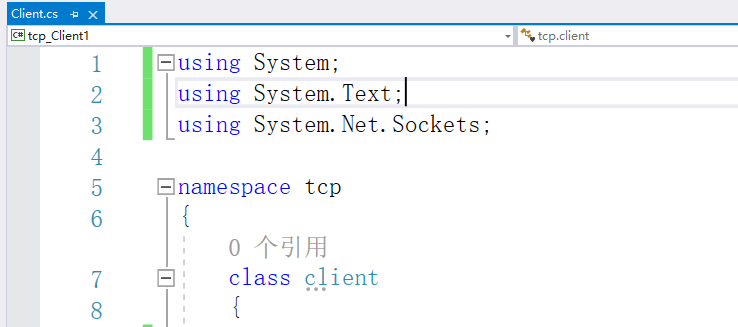


图4.37 命名空间

**4、为了实现与服务器的连接，我们**定义一个私有整型常量portnum用来存放我们指定的与服务器进行通信的端口号，定义一个私有字符串常量hostname用来存放我们要连接的服务器的IP地址；接着用Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)创建一个基于TCP的套接字；

创建Socket后我们就可以与远程服务器进行连接了，我们通过Connect函数指定服务器端口与IP地址来与服务器完成连接，当 Connect 函数成功返回后，表明tcp的“三次握手”连接已完成，向显示器输出Connect Complete表明连接成功。



**图4.38 连接服务器部分代码**

**5、为了能够不断地向服务器发送自身设备的状态信息，我们**接着我们首先检查客户端与服务器的连接是否还存在，如果仍在连接中，则允许执行下面的发送信息步骤；

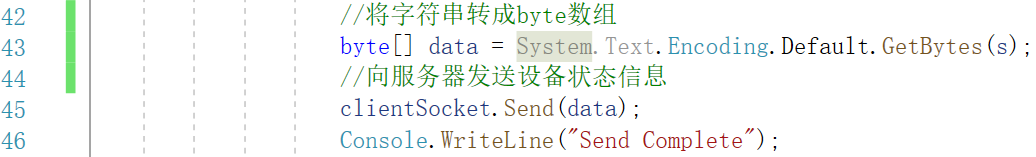
首先使用一个循环函数以便让客户端不断地向服务器发送设备信息，但我们要先使用System.Threading.Thread.Sleep(1000)语句将当前线程休眠1s再向下执行，否则发送信息频率太快；

接着我们定义一个字符串guid用来获取设备的全球唯一标识符用来表示设备的sn号；定义一个字符串strFu用来存储限定格式的当前时间；定义一个变量n用于描述设备的电功率变化。接着定义一个字符串s用来存储限定格式为{"type":"data","id":"1001","sn":"DNPLK11FNK","power":"1200","state":"on","time":"2016-04-18 08:31:58"“/r/n”}类型的设备信息。



**图4.39 不间断发送信息部分代码**

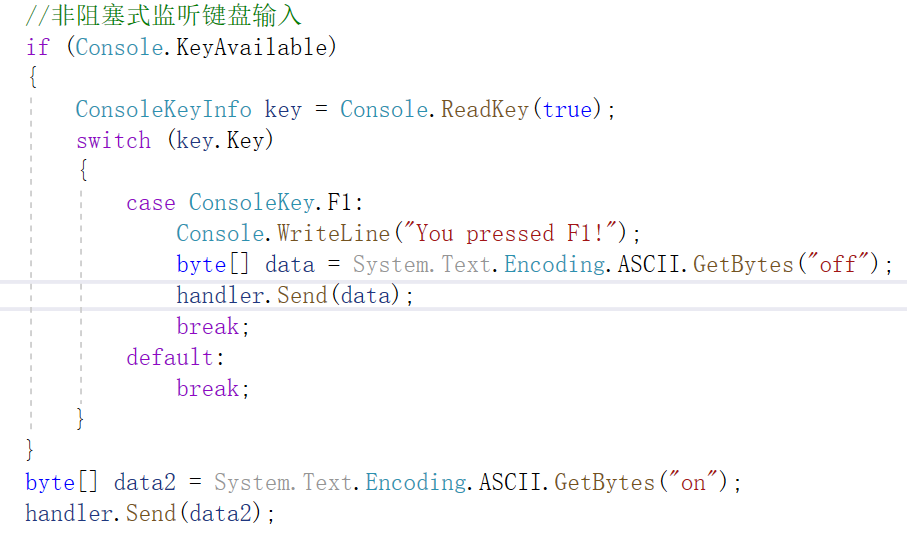
**6、但是上一步还存在着一个问题，就是**字符串是无法直接发送给服务器的，所以我们需要向用Encoding将字符串编码为byte数组后才能发送；当使用Send语句发送成功后向显示器输出Send Complete。



**图4.40 转义字符串部分代码**

4.6.2 控制机制

1、为了实现服务器能够实现对客户端的控制，我们首先在服务器中加入如下代码，用来监听键盘是否有输入以及如果有输入则根据输入的值采取不同措施，当键盘没有输入或输入与设定值不同时，服务器向客户端发送on信息表明希望客户端继续运行，但当键盘输入为F1时，服务器向客户端发送off信息表示服务端希望客户端能够关闭；



**图4.41 非阻塞式监听键盘输入部分代码**

**2、为了实现客户端能够监听服务器消息并根据不同消息做出不同选择，我们还需要在客户端加入如下代码：**

我们定义一个长度为1024的byte数组来接收来自服务器的消息并将其转换为字符串，如果检测到字符串为on，则表示服务器希望客户端继续开启，不做任何操作；但是当检测到字符串为off时，表示服务器希望传感器关闭，则断开连接。

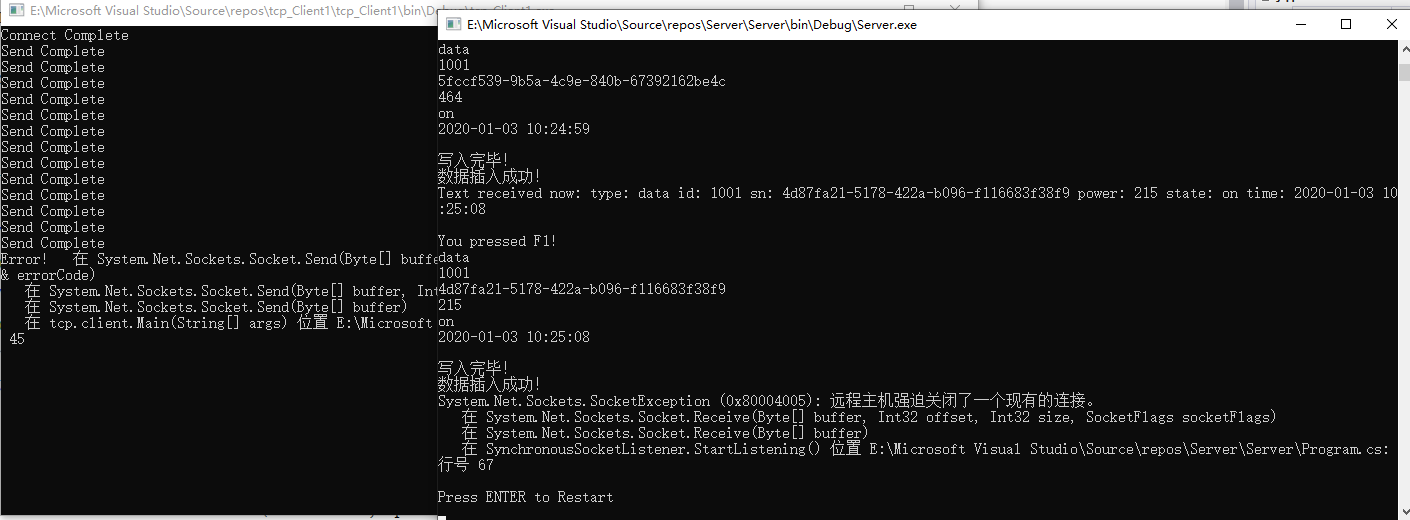
最后还有个catch语句用于捕获程序运行中的异常，但发生异常时向显示器输出Error！和错误信息。



**图4.42 客户端接受服务端消息部分代码**

4.6.3 运行实例

**当我们先开启服务端再开启客户端后两者能正常通信，按下F1键后，成功关闭连接。**

**图4.43 运行实例**

4.6.4 控制机制中改进的问题

**在一开始监听键盘输入的时候，我们采用的是**Console.Read()函数，这就导致了一个问题，就是Read()函数会阻碍下面代码的运行，当程序执行到Read()函数后，程序会一直在这里监听键盘输入若无输入则不会执行下一步，阻塞了程序的正常运行，所以通过查阅资料后，我们采用了这种非阻塞监听键盘输入的方法，这样程序就可以在监听键盘输入的同时接收并向控制台输出来自客户端的信息了。

第5章 软件测试

## 5.1数据库连接测试

5.1.1 client端与server端发送数据

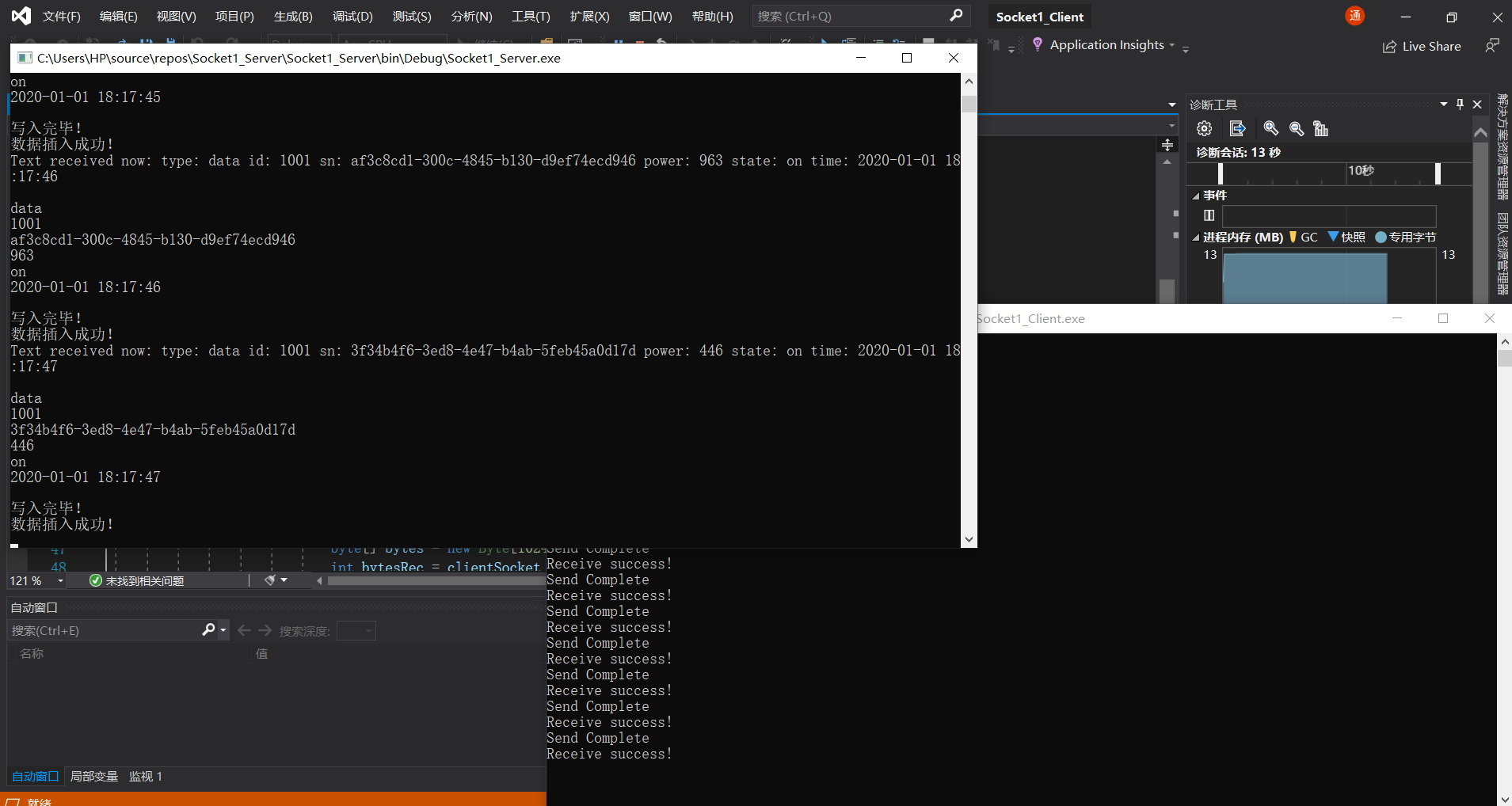


图5.1 server端发送数据

5.1.2 数据库存储数据

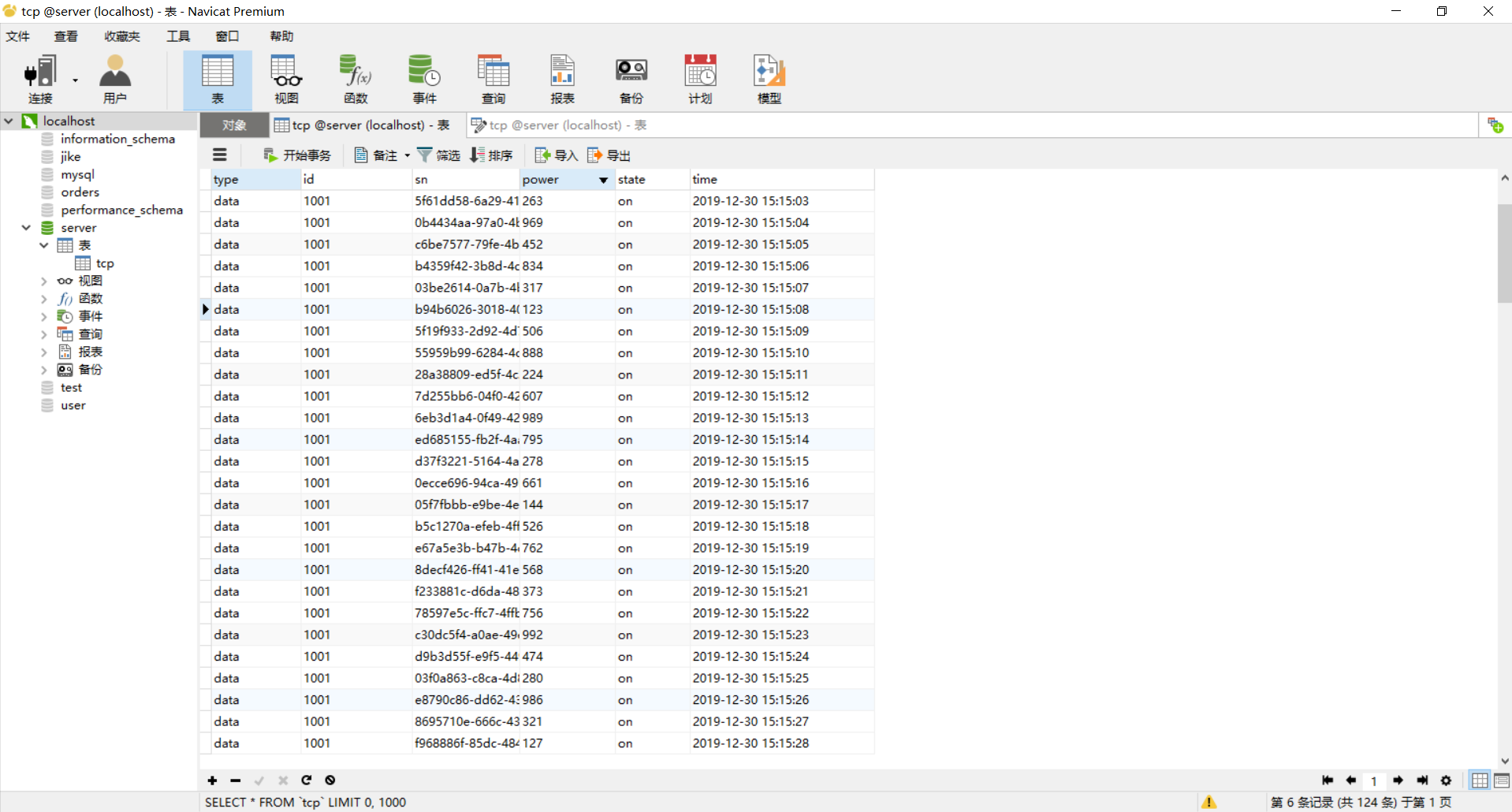


图5.2 数据库存储数据成功

## 5.2 Java连接测试

5.2.1测试Java连接提取数据库

设计OP测试类，此测试类用于测试DBUtil、Entity、DAO三个类是否正确。

以下为测试主要代码：

ArrayList<Entity> ar=**new** DAO().getAll();

**for**(Entity ne:ar) {

System.***out***.println("消息类型:"+ne.getType()+"\t设备ID:"+ne.getId()+"\t设备序列号:"+ne.getSn()+"\t电压:"+ne.getPower()+"\t设备状态:"+ne.getState()+"\t时间:"+ne.getTime());

}

即调出显示所有数据库内的数据在控制台内。

测试结果如下图5.3所示：



图5.3 Java连接提取数据库成功

成功显示相关内容，正面代码正确无bug。

5.2.2 测试每秒动态更新折线图

在开启服务端与客户端通信之后，开启Java可视化界面，折线图如下图5.4所示不断变化，折线自右向左不断移动，在最右侧出现最新的电压数据。



图5.4 动态折线图（1）

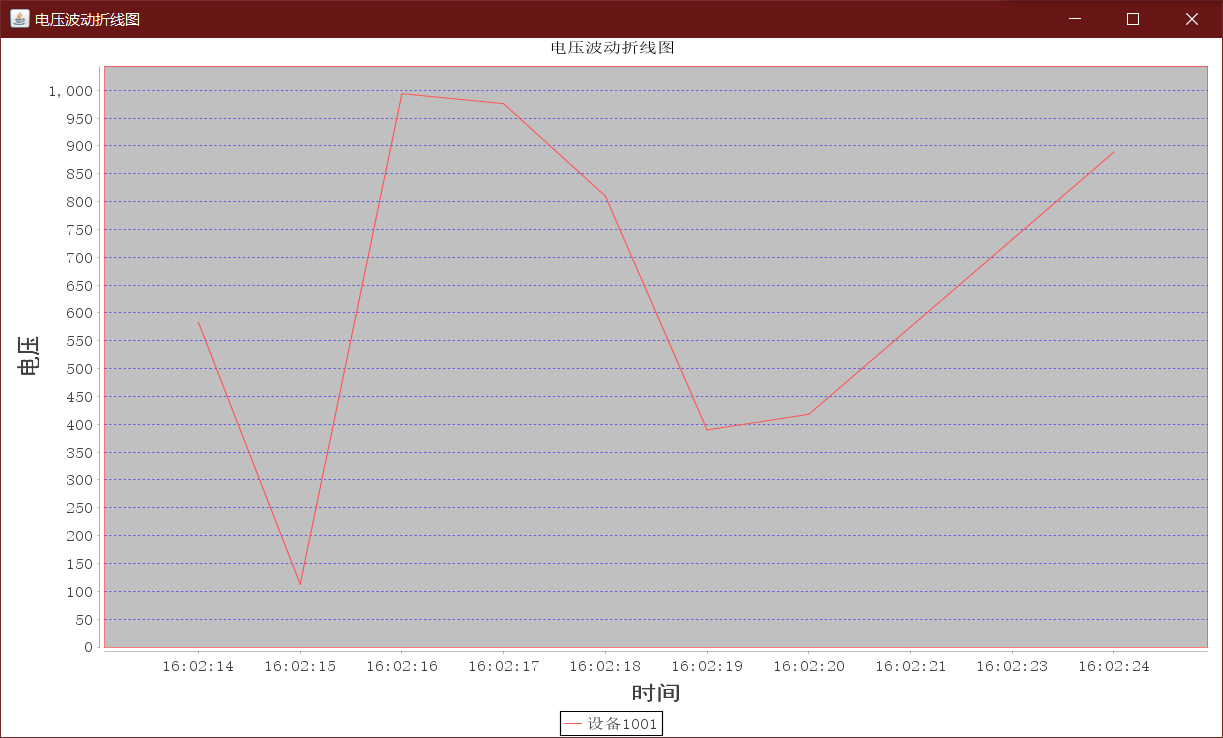


图5.4 动态折线图（2）

# 参考文献:

【1】<https://baike.baidu.com/item/Microsoft%20Visual%20Studio/>

【2】<https://download.csdn.net/download/emeralddream/2683238>

【3】<http://www.ltesting.net/ceshi/ruanjianceshikaifajishu/rjcskfyy/dotnet/2007/0630/99765.html>

【4】<https://blog.csdn.net/weixin_30566149/article/details/97662595>

附 录:

1、Client端：

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** System.Net.Sockets;
7. **using** System.Net;
9. **namespace** tcp
10. {
11. **class** client
12. {
13. **private** **const** **int** portnum = 8023;
14. **private** **const** **string** hostname = "192.168.0.108";
15. //hostname是服务器当前网络下的ip
16. **static** **void** Main(**string**[] args)
17. {
18. Socket clientSocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
19. clientSocket.Connect(hostname, portnum);
20. //Connect方法使用指定的IP地址和端口号将客户端连接到远程TCP主机
21. Console.WriteLine("Connect Complete");
22. **try**
23. {
24. **if** (clientSocket.Connected)
25. {
26. // 如果客户与服务器有连接，并且还未断，则允许发送信息
28. **if** (clientSocket != **null**)
29. {
30. **while** (**true**)
31. {
32. System.Threading.Thread.Sleep(1000);
33. //获取黑框上的输入字符串
34. DateTime dt = **new** DateTime();
35. String guid = Guid.NewGuid().ToString();
36. dt = System.DateTime.Now;
37. **string** strFu = dt.ToString("yyyy-MM-dd HH:mm:ss");
38. //获取时间
39. Random ran = **new** Random();
40. **int** n = ran.Next(100, 1000);
41. //获取随机数
42. **string** s = "type:" + " data " + "id:" + " 1001 " + "sn: " + guid + " power: " + n + " state: " + "on" + " time: " + strFu + "\r\n";
43. **byte**[] data = System.Text.Encoding.Default.GetBytes(s);
44. clientSocket.Send(data);
45. Console.WriteLine("Send Complete");
46. // 接收数据
47. **byte**[] bytes = **new** Byte[1024];
48. **int** bytesRec = clientSocket.Receive(bytes);
49. **string** data1 = Encoding.ASCII.GetString(bytes, 0, bytesRec);
50. Console.WriteLine(data1);
51. **if** (data1 == "off")
52. {
53. clientSocket.Close();
54. }
55. }
56. }
57. }
58. }
59. **catch** (Exception e)
60. {
61. Console.WriteLine("Error!" + e.StackTrace);
62. Console.ReadLine();
63. }
64. }
65. }
66. }

2、Server端：

1. **using** System;
2. **using** System.Net;
3. **using** System.Net.Sockets;
4. **using** System.Text;
5. **using** System.IO;
6. **using** MySql.Data;
7. **using** MySql.Data.MySqlClient;
9. **public** **class** SynchronousSocketListener
10. {
12. // Incoming data from the client.
13. **public** **static** **string** data = **null**;
15. **public** **static** **void** StartListening()
16. {
17. // Data buffer for incoming data.
18. **byte**[] bytes = **new** Byte[1024];
20. // Establish the local endpoint for the socket.
21. // Dns.GetHostName returns the name of the
22. // host running the application.
23. IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
24. IPAddress ipAddress = **null**;


28. **for** (**int** i = 0; i < ipHostInfo.AddressList.Length; i++)
29. {
30. //从IP地址列表中筛选出IPv4类型的IP地址
31. //AddressFamily.InterNetwork表示此IP为IPv4,
32. //AddressFamily.InterNetworkV6表示此地址为IPv6类型
33. **if** (ipHostInfo.AddressList[i].AddressFamily == AddressFamily.InterNetwork)
34. {
35. ipAddress = ipHostInfo.AddressList[i];
36. Console.WriteLine(ipAddress);
37. Console.WriteLine(i);
38. }
39. }
40. IPEndPoint localEndPoint = **new** IPEndPoint(ipAddress, 8023);
42. // Create a TCP/IP socket.
43. Socket listener = **new** Socket(AddressFamily.InterNetwork,
44. SocketType.Stream, ProtocolType.Tcp);
46. // Bind the socket to the local endpoint and
47. // listen for incoming connections.
48. **try**
49. {
50. listener.Bind(localEndPoint);
51. listener.Listen(10);
53. // Start listening for connections.
54. **while** (**true**)
55. {
56. Console.WriteLine("\nWaiting for a connection...");
57. // Program is suspended while waiting for an incoming connection.
58. Socket handler = listener.Accept();
59. data = **null**;
61. //存放即时信息
62. **string** msgnow = **null**;
64. // An incoming connection needs to be processed.
65. **while** (**true**)
66. {
67. bytes = **new** **byte**[1024];
68. **int** bytesRec = handler.Receive(bytes);
69. data += Encoding.ASCII.GetString(bytes, 0, bytesRec);
70. //赋值即时信息
71. msgnow = Encoding.ASCII.GetString(bytes, 0, bytesRec);
72. //打印即时信息
73. Console.WriteLine("Text received now: {0}", msgnow);
74. //非阻塞式监听键盘输入
75. **if** (Console.KeyAvailable)
76. {
77. ConsoleKeyInfo key = Console.ReadKey(**true**);
78. **switch** (key.Key)
79. {
80. **case** ConsoleKey.F1:
81. Console.WriteLine("You pressed F1!");
82. **byte**[] data = System.Text.Encoding.ASCII.GetBytes("off");
83. handler.Send(data);
84. **break**;
85. **default**:
86. **break**;
87. }
88. }
89. **byte**[] data2 = System.Text.Encoding.ASCII.GetBytes("Receive success!");
90. handler.Send(data2);
92. **string** type1 = msgnow.Split(' ')[1];
93. **string** id1 = msgnow.Split(' ')[3];
94. **string** sn1 = msgnow.Split(' ')[5];
95. **string** power1 = msgnow.Split(' ')[7];
96. **string** state1 = msgnow.Split(' ')[9];
97. **string** time1 = msgnow.Split(' ')[11] + ' ' + msgnow.Split(' ')[12];
99. Console.WriteLine(type1);
100. Console.WriteLine(id1);
101. Console.WriteLine(sn1);
102. Console.WriteLine(power1);
103. Console.WriteLine(state1);
104. Console.WriteLine(time1);
105. Console.WriteLine("写入完毕！");
107. **string** str = "server=localhost;User Id=root;password=root;Database=server";//连接MySQL的字符串
108. MySqlConnection mycon = **new** MySqlConnection(str);//实例化链接
109. mycon.Open();//开启连接
110. MySqlCommand mycmd = **new** MySqlCommand("insert into tcp(type,id,sn,power,state,time) values('" + type1 + "','" + id1 + "','" + sn1 + "','" + power1 + "','" + state1 + "','" + time1 + "')", mycon);
111. **if** (mycmd.ExecuteNonQuery() > 0)
112. {
113. Console.WriteLine("数据插入成功！");
114. }
115. // Console.ReadLine();
116. mycon.Close();//关闭

119. }
120. }
121. }
122. **catch** (Exception e)
123. {
124. Console.WriteLine(e.ToString());
125. }
126. Console.WriteLine("\nPress ENTER to Restart");
127. Console.Read();
129. }
130. **public** **static** **int** Main(String[] args)
131. {
132. StartListening();
133. **return** 0;
134. }
135. }

3、SQL文件：

1. /\*
2. Navicat MySQL Data Transfer
4. Source Server         : localhost
5. Source Server Version : 50536
6. Source Host           : localhost:3306
7. Source **Database**       : server
9. Target Server Type    : MYSQL
10. Target Server Version : 50536
11. File Encoding         : 65001
13. **Date**: 2020-01-01 18:22:52
14. \*/
16. **SET** FOREIGN\_KEY\_CHECKS=0;
18. -- ----------------------------
19. -- Table structure for tcp
20. -- ----------------------------
21. **DROP** **TABLE** IF EXISTS `tcp`;
22. **CREATE** **TABLE** `tcp` (
23. `type` **varchar**(255) NOT NULL,
24. `id` **varchar**(255) NOT NULL,
25. `sn` **varchar**(255) NOT NULL,
26. `power` **varchar**(255) NOT NULL,
27. `state` **varchar**(255) NOT NULL,
28. `**time**` **varchar**(255) NOT NULL
29. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;
31. -- ----------------------------
32. -- Records of tcp
33. -- ----------------------------
34. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '5f61dd58-6a29-4143-8f3c-293d24cb9b30', '263', 'on', '2019-12-30 15:15:03\r\n');
35. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0b4434aa-97a0-4b7b-b753-9f68bb3ebd9b', '969', 'on', '2019-12-30 15:15:04\r\n');
36. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'c6be7577-79fe-4b8f-a4cd-efe6a93357fd', '452', 'on', '2019-12-30 15:15:05\r\n');
37. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b4359f42-3b8d-4d3e-8f64-56aa9104710c', '834', 'on', '2019-12-30 15:15:06\r\n');
38. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '03be2614-0a7b-4b5e-96ce-0465d84d2ecd', '317', 'on', '2019-12-30 15:15:07\r\n');
39. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b94b6026-3018-40a2-a7e5-c090eac2b404', '123', 'on', '2019-12-30 15:15:08\r\n');
40. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '5f19f933-2d92-4d7b-ba75-f99ea3f23dba', '506', 'on', '2019-12-30 15:15:09\r\n');
41. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '55959b99-6284-4ced-b3f7-c887e6944654', '888', 'on', '2019-12-30 15:15:10\r\n');
42. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '28a38809-ed5f-4c22-8630-d233c3bb04cf', '224', 'on', '2019-12-30 15:15:11\r\n');
43. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '7d255bb6-04f0-42ac-92a7-575458c05c74', '607', 'on', '2019-12-30 15:15:12\r\n');
44. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '6eb3d1a4-0f49-42d3-ab45-ff7c37d223f7', '989', 'on', '2019-12-30 15:15:13\r\n');
45. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'ed685155-fb2f-4aa1-a9c7-4c872f223ea9', '795', 'on', '2019-12-30 15:15:14\r\n');
46. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd37f3221-5164-4a4e-8380-48192d726767', '278', 'on', '2019-12-30 15:15:15\r\n');
47. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0ecce696-94ca-49b2-8d5a-d6dc017a73a0', '661', 'on', '2019-12-30 15:15:16\r\n');
48. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '05f7fbbb-e9be-4e51-ba73-d6b543132293', '144', 'on', '2019-12-30 15:15:17\r\n');
49. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b5c1270a-efeb-4ff9-a8a0-78f2b9bd5d1e', '526', 'on', '2019-12-30 15:15:18\r\n');
50. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'e67a5e3b-b47b-4dd3-875a-3ed52786f194', '762', 'on', '2019-12-30 15:15:19\r\n');
51. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8decf426-ff41-41ea-8302-bb61468c20c2', '568', 'on', '2019-12-30 15:15:20\r\n');
52. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'f233881c-d6da-48b8-904c-5fc7c05e606d', '373', 'on', '2019-12-30 15:15:21\r\n');
53. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '78597e5c-ffc7-4ffb-a5da-8fc5d8be06b2', '756', 'on', '2019-12-30 15:15:22\r\n');
54. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'c30dc5f4-a0ae-49ec-bfef-108b340f034e', '992', 'on', '2019-12-30 15:15:23\r\n');
55. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd9b3d55f-e9f5-4495-86f6-a592d7dc2884', '474', 'on', '2019-12-30 15:15:24\r\n');
56. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '03f0a863-c8ca-4d89-833c-852d5e6733ce', '280', 'on', '2019-12-30 15:15:25\r\n');
57. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'e8790c86-dd62-43f9-bb9c-0e667ea353e0', '986', 'on', '2019-12-30 15:15:26\r\n');
58. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8695710e-666c-4322-968d-f4b5b7b727e5', '321', 'on', '2019-12-30 15:15:27\r\n');
59. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'f968886f-85dc-484d-a742-cea1a5cb71b7', '127', 'on', '2019-12-30 15:15:28\r\n');
60. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '6f1cd174-9e8d-4bab-ba92-918da18efcf2', '510', 'on', '2019-12-30 15:15:29\r\n');
61. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'db3f37b9-8063-403e-92f3-0d6ab4ec2e07', '893', 'on', '2019-12-30 15:15:30\r\n');
62. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '3461ecc5-5607-4399-9308-6f042a0dd756', '228', 'on', '2019-12-30 15:15:31\r\n');
63. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '50915d32-8ce8-4f44-b7f4-fe2946f23df6', '934', 'on', '2019-12-30 15:15:32\r\n');
64. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '7202e276-93e2-4552-b13e-d9e3a2b12dc0', '417', 'on', '2019-12-30 15:15:33\r\n');
65. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '196f5a7f-7957-47ac-859c-6617e2918eff', '222', 'on', '2019-12-30 15:15:34\r\n');
66. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0db2ad32-4699-4cea-be0e-4dbef70d1c84', '458', 'on', '2019-12-30 15:15:35\r\n');
67. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd389d391-0cda-4c9f-a252-f560ac8f77ed', '841', 'on', '2019-12-30 15:15:36\r\n');
68. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '23bbfd02-800c-4aa3-aea3-1a36cb935c9c', '646', 'on', '2019-12-30 15:15:37\r\n');
69. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '55396d41-2e54-4ccf-a293-98273899dbd6', '129', 'on', '2019-12-30 15:15:38\r\n');
70. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8962f109-4c60-4398-919c-eacef16fa966', '835', 'on', '2019-12-30 15:15:39\r\n');
71. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'ecefbcf6-a914-4927-963b-09e03755b9c9', '170', 'on', '2019-12-30 15:15:40\r\n');
72. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'e9034ffd-52f4-4c7e-8534-0ed38a9d2afe', '553', 'on', '2019-12-30 15:15:41\r\n');
73. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'bee6405c-e515-4026-986f-0d73999d6d49', '936', 'on', '2019-12-30 15:15:42\r\n');
74. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'f86b951a-4cbb-4527-a503-5ac7a5e39f5d', '742', 'on', '2019-12-30 15:15:43\r\n');
75. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '7e59159c-ffe2-4ba2-89f5-b0caac48b855', '977', 'on', '2019-12-30 15:15:44\r\n');
76. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '58b65c88-71cb-4dd5-8996-b489ee393740', '460', 'on', '2019-12-30 15:15:45\r\n');
77. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b080613b-155f-42ed-aacc-2743b389868d', '266', 'on', '2019-12-30 15:15:46\r\n');
78. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '146f29a2-a292-44ed-8bd8-2880f0d081cf', '971', 'on', '2019-12-30 15:15:47\r\n');
79. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'bee30e10-547f-4892-8c45-14fb8d370525', '454', 'on', '2019-12-30 15:15:48\r\n');
80. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '25e9fb80-7427-4dc1-b16d-fba51cdf9dda', '690', 'on', '2019-12-30 15:15:49\r\n');
81. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0f0d1634-6468-41ee-aa68-39486bc6892c', '172', 'on', '2019-12-30 15:15:50\r\n');
82. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '3e2f1b2a-6d6b-41b3-bec9-30b735fd2d98', '878', 'on', '2019-12-30 15:15:51\r\n');
83. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'a0e8dd31-becd-4682-8c38-8c221090484b', '361', 'on', '2019-12-30 15:15:52\r\n');
84. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '13ea62ed-48ff-4860-98ae-417c8289136f', '167', 'on', '2019-12-30 15:15:53\r\n');
85. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'a69bfc92-cca5-4129-ac0a-87a95115cf34', '549', 'on', '2019-12-30 15:15:54\r\n');
86. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '06128c23-cc31-4307-9fba-bff00d04d602', '785', 'on', '2019-12-30 15:15:55\r\n');
87. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8a005338-91d7-4013-a15d-81d170d2ae16', '268', 'on', '2019-12-30 15:15:56\r\n');
88. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '77470f3f-02c9-4161-b0b7-ff9369fe139c', '203', 'on', '2020-01-01 15:50:08\r\n');
89. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '689495e1-4a15-4317-b401-bc2d8e28b7b2', '586', 'on', '2020-01-01 15:50:09\r\n');
90. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '493fb621-1d44-41ac-b28e-b78ac612bbeb', '969', 'on', '2020-01-01 15:50:10\r\n');
91. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0e91ffbf-6011-40a6-9b5c-b5e433fa4324', '452', 'on', '2020-01-01 15:50:11\r\n');
92. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'e3806ae4-52da-4ff0-81e4-498220555d7e', '835', 'on', '2020-01-01 15:50:12\r\n');
93. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0aab61df-292d-408f-9f26-b0b8b049c5af', '640', 'on', '2020-01-01 15:50:13\r\n');
94. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'e887065d-fbc9-4535-a40b-d8d7deb2dddc', '123', 'on', '2020-01-01 15:50:14\r\n');
95. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b1c00c4a-cb31-4e54-b3e1-037c451414c4', '506', 'on', '2020-01-01 15:50:15\r\n');
96. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8385e2ff-c896-4cd3-8af7-79bd25614e81', '889', 'on', '2020-01-01 15:50:16\r\n');
97. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '4712359a-afd3-4439-97ee-ddf66a4752f2', '371', 'on', '2020-01-01 15:50:17\r\n');
98. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '745cf15d-9896-4cad-83a4-e3797ec214a0', '312', 'on', '2020-01-01 17:39:43\r\n');
99. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd46f5c66-0420-4f9d-a8ae-b78a2ea2953b', '694', 'on', '2020-01-01 17:39:44\r\n');
100. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '6e7e6da8-c843-4f41-96d8-e9c03feb553d', '177', 'on', '2020-01-01 17:39:45\r\n');
101. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0cc8dfaa-8ff1-4a78-9ba1-b60321d4c657', '413', 'on', '2020-01-01 17:39:46\r\n');
102. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '8244ab23-7b06-403b-a445-7addcbb2c0aa', '796', 'on', '2020-01-01 17:39:47\r\n');
103. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'b4735d51-196b-4634-ad77-fdfce4bded64', '278', 'on', '2020-01-01 17:39:48\r\n');
104. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '10589010-7705-4615-bde7-f4ac5ef1be3b', '661', 'on', '2020-01-01 17:39:49\r\n');
105. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '01b5dcca-bc12-4577-a136-8ed06ab9dd11', '467', 'on', '2020-01-01 17:39:50\r\n');
106. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '95198239-4353-4788-b455-48775a745d0a', '850', 'on', '2020-01-01 17:39:51\r\n');
107. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '6f04f7b6-c01a-4d7e-a8cd-93b279958336', '332', 'on', '2020-01-01 17:39:52\r\n');
108. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'f4e9e86b-7e75-40bd-a2fa-1b8ccfcdc046', '715', 'on', '2020-01-01 17:39:54\r\n');
109. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'fcbf61df-b774-4548-bfce-4e6f955ade69', '521', 'on', '2020-01-01 17:39:55\r\n');
110. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'dce705cb-777f-49a5-a725-7426d22eb9a5', '904', 'on', '2020-01-01 17:39:56\r\n');
111. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '598190bd-0fce-4f06-87b6-d14cbec3c918', '386', 'on', '2020-01-01 17:39:57\r\n');
112. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'ce8df887-324e-41bd-92ce-1c7fad31abc9', '769', 'on', '2020-01-01 17:39:58\r\n');
113. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd6de76ad-fccb-4f77-8376-0dc1640431ba', '105', 'on', '2020-01-01 17:39:59\r\n');
114. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'd42261c9-64f6-409c-a282-fa100501d918', '487', 'on', '2020-01-01 17:40:00\r\n');
115. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '73d2ca00-dd35-42b9-bd1b-43a158e1795f', '293', 'on', '2020-01-01 17:40:01\r\n');
116. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '134e1fc8-f53c-4af5-af6d-04a0fb35cc06', '676', 'on', '2020-01-01 17:40:02\r\n');
117. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'fba7860c-77b7-404a-b7c9-7688fd67f80d', '482', 'on', '2020-01-01 17:40:03\r\n');
118. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '284e26d9-e0a0-4ba6-a78f-6839bcbeb7fc', '864', 'on', '2020-01-01 17:40:04\r\n');
119. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '557babde-33b7-4f17-b528-8a1d199e98e4', '200', 'on', '2020-01-01 17:40:05\r\n');
120. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'cddb1fe7-8859-4aa8-bea6-24b731d2a0e0', '583', 'on', '2020-01-01 17:40:06\r\n');
121. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'bd25c20a-1a60-41bd-965a-b1b1a35b144d', '388', 'on', '2020-01-01 17:40:07\r\n');
122. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '0d5285fc-bf01-4f50-9df1-a75a30643ba5', '771', 'on', '2020-01-01 17:40:08\r\n');
123. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '61455d16-740f-458e-a75f-30700b2c18ae', '107', 'on', '2020-01-01 17:40:09\r\n');
124. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '826a8cff-0d7a-4de6-9609-8cc841bb05e2', '490', 'on', '2020-01-01 17:40:10\r\n');
125. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'db85fee9-ae61-41f2-97cf-6e57dbac9aaa', '872', 'on', '2020-01-01 17:40:11\r\n');
126. **INSERT** **INTO** `tcp` **VALUES** ('data', '1001', '6e09625e-00cf-4ad9-92d0-ff910ee8d38e', '678', 'on', '2020-01-01 17:40:12\r\n');

**INSERT** **INTO** `tcp` **VALUES** ('data', '1001', 'a6b54e07-5fc7-4b9a-80e5-1e1f1f65288f', '161', 'on', '2020-01-01 17:40:13\r\n');

4、Java可视化动态更新实现

①DBUtil：

1. **package** dbutil;
3. **import** java.sql.Connection;
4. **import** java.sql.DriverManager;
5. **import** java.sql.SQLException;
7. **public** **class** DBUtil {
8. **public** **static** Connection getConnection() {
9. Connection conn = **null**;
11. **try** {
12. Class.forName("com.mysql.cj.jdbc.Driver");
13. } **catch** (ClassNotFoundException e) {
14. System.out.println("加载驱动失败");
15. e.printStackTrace();
16. }
18. String url = "jdbc:mysql://localhost:3306/server?serverTimezone=UTC";
19. String username = "root";
20. String password = "oipopo09";

23. **try** {
24. conn = DriverManager.getConnection(url,username,password);
25. } **catch** (SQLException e) {
26. System.out.println("连接失败");
27. e.printStackTrace();
28. }
30. **return** conn;
32. }
33. }

②Entity:

1. **package** entity;
3. **public** **class** Entity {
4. **private** String type;
5. **private** String id;
6. **private** String sn;
7. **private** String power;
8. **public** String state;
9. **private** String time;

12. **public** String getType() {
13. **return** type;
14. }
15. **public** **void** setType(String type) {
16. **this**.type = type;
17. }
18. **public** String getId() {
19. **return** id;
20. }
21. **public** **void** setId(String id) {
22. **this**.id = id;
23. }
24. **public** String getSn() {
25. **return** sn;
26. }
27. **public** **void** setSn(String sn) {
28. **this**.sn = sn;
29. }
30. **public** String getPower() {
31. **return** power;
32. }
33. **public** **void** setPower(String power) {
34. **this**.power = power;
35. }
36. **public** String getState() {
37. **return** state;
38. }
39. **public** **void** setState(String state) {
40. **this**.state = state;
41. }
42. **public** String getTime() {
43. **return** time;
44. }
45. **public** **void** setTime(String time) {
46. **this**.time = time;
47. }
49. }

③DAO:

1. **package** dao;
3. **import** java.sql.Connection;
4. **import** java.sql.PreparedStatement;
5. **import** java.sql.ResultSet;
6. **import** java.sql.SQLException;
7. **import** java.util.ArrayList;
9. **import** dbutil.DBUtil;
10. **import** entity.Entity;
12. **public** **class** DAO {
14. //查找所有数据
15. **public** ArrayList<Entity> getAll(){
16. ArrayList<Entity> ar = **new** ArrayList<Entity>();
18. Connection conn = DBUtil.getConnection();
19. PreparedStatement ps = **null**;
20. ResultSet rs = **null**;
22. String sql = "select type,id,sn,power,state,time from tcp";
23. **try** {
24. ps = conn.prepareStatement(sql);
25. } **catch** (SQLException e) {
26. System.out.println("SQL语句预编译失败");
27. e.printStackTrace();
28. }
29. **try** {
30. rs = ps.executeQuery();
31. } **catch** (SQLException e) {
32. System.out.println("SQL语句存放失败");
33. e.printStackTrace();
34. }
36. **try** {
37. **while**(rs.next()) {
38. Entity ent = **new** Entity();
39. ent.setType(rs.getString("type"));
40. ent.setId(rs.getString("id"));
41. ent.setSn(rs.getString("sn"));
42. ent.setPower(rs.getString("power"));
43. ent.setState(rs.getString("state"));
44. ent.setTime(rs.getString("time"));
45. ar.add(ent);
46. }
47. } **catch** (SQLException e) {
48. System.out.println("存放数据至ar失败");
49. e.printStackTrace();
50. }
52. **finally** {
53. **if** (rs != **null**) {
54. **try** {
55. rs.close();
56. } **catch** (SQLException e) {
57. System.out.println("rs释放失败");
58. e.printStackTrace();
59. }
60. }
61. **if** (ps != **null**) {
62. **try** {
63. ps.close();
64. } **catch** (SQLException e) {
65. System.out.println("ps释放失败");
66. e.printStackTrace();
67. }
68. }
69. **if** (conn != **null**) {
70. **try** {
71. conn.close();
72. } **catch** (SQLException e) {
73. System.out.println("conn释放失败");
74. e.printStackTrace();
75. }
76. }
77. }
79. **return** ar;
80. }

83. //查找最新十个数据
84. **public** ArrayList<Entity> getTen(){
85. ArrayList<Entity> ar = **new** ArrayList<Entity>();
87. Connection conn = DBUtil.getConnection();
88. PreparedStatement ps = **null**;
89. ResultSet rs = **null**;
91. String sql = "select \* from tcp Order By time Desc limit 10";
92. **try** {
93. ps = conn.prepareStatement(sql);
94. } **catch** (SQLException e) {
95. System.out.println("SQL语句预编译失败");
96. e.printStackTrace();
97. }
98. **try** {
99. rs = ps.executeQuery();
100. } **catch** (SQLException e) {
101. System.out.println("SQL语句存放失败");
102. e.printStackTrace();
103. }
105. **try** {
106. **while**(rs.next()) {
107. Entity ent = **new** Entity();
108. ent.setType(rs.getString("type"));
109. ent.setId(rs.getString("id"));
110. ent.setSn(rs.getString("sn"));
111. ent.setPower(rs.getString("power"));
112. ent.setState(rs.getString("state"));
113. ent.setTime(rs.getString("time"));
114. ar.add(ent);
115. }
116. } **catch** (SQLException e) {
117. System.out.println("存放数据至ar失败");
118. e.printStackTrace();
119. }
121. **finally** {
122. **if** (rs != **null**) {
123. **try** {
124. rs.close();
125. } **catch** (SQLException e) {
126. System.out.println("rs释放失败");
127. e.printStackTrace();
128. }
129. }
130. **if** (ps != **null**) {
131. **try** {
132. ps.close();
133. } **catch** (SQLException e) {
134. System.out.println("ps释放失败");
135. e.printStackTrace();
136. }
137. }
138. **if** (conn != **null**) {
139. **try** {
140. conn.close();
141. } **catch** (SQLException e) {
142. System.out.println("conn释放失败");
143. e.printStackTrace();
144. }
145. }
146. }
148. **return** ar;
149. }
151. //查找最新一个数据
152. @SuppressWarnings("null")
153. **public** **static** Entity getLatest(String id){
154. Entity ent = **new** Entity();
156. Connection conn = DBUtil.getConnection();
157. PreparedStatement ps = **null**;
158. ResultSet rs = **null**;
160. String sql = "select state from tcp where id = ? Order By time Desc limit 1";
161. **try** {
162. ps.setString(1, id);
163. } **catch** (SQLException e1) {
164. // TODO Auto-generated catch block
165. e1.printStackTrace();
166. }
167. **try** {
168. ps = conn.prepareStatement(sql);
169. } **catch** (SQLException e) {
170. System.out.println("SQL语句预编译失败");
171. e.printStackTrace();
172. }
173. **try** {
174. rs = ps.executeQuery();
175. } **catch** (SQLException e) {
176. System.out.println("SQL语句存放失败");
177. e.printStackTrace();
178. }
180. **try** {
181. ent.setType(rs.getString("type"));
182. ent.setId(rs.getString("id"));
183. ent.setSn(rs.getString("sn"));
184. ent.setPower(rs.getString("power"));
185. ent.setState(rs.getString("state"));
186. ent.setTime(rs.getString("time"));
188. } **catch** (SQLException e) {
189. System.out.println("提取失败");
190. e.printStackTrace();
191. }
193. **finally** {
194. **if** (rs != **null**) {
195. **try** {
196. rs.close();
197. } **catch** (SQLException e) {
198. System.out.println("rs释放失败");
199. e.printStackTrace();
200. }
201. }
202. **if** (ps != **null**) {
203. **try** {
204. ps.close();
205. } **catch** (SQLException e) {
206. System.out.println("ps释放失败");
207. e.printStackTrace();
208. }
209. }
210. **if** (conn != **null**) {
211. **try** {
212. conn.close();
213. } **catch** (SQLException e) {
214. System.out.println("conn释放失败");
215. e.printStackTrace();
216. }
217. }
218. }
220. **return** ent;
221. }
223. // 修改
224. **public** **void** update(Entity ent) {
225. Connection conn = DBUtil.getConnection();
226. PreparedStatement ps = **null**;
227. String sql = "UPDATE tcp SET state= ? WHERE id=?";
228. **try** {
229. ps = conn.prepareStatement(sql);
230. ps.setString(1, ent.getState());
231. **int** a = ps.executeUpdate();
232. **if** (a > 0) {
233. System.out.println("修改成功");
234. } **else** {
235. System.out.println("修改失败");
236. }
237. } **catch** (SQLException e) {
238. e.printStackTrace();
239. } **finally** {
240. **try** {
241. // 关闭
242. **if** (ps != **null**) {
243. ps.close();
244. }
245. **if** (conn != **null**) {
246. conn.close();
247. }
248. } **catch** (Exception e2) {
249. e2.printStackTrace();
250. }
251. }
252. }
253. }

④Charts:

1. **import** java.awt.Color;
2. **import** java.awt.Font;
3. **import** java.util.ArrayList;
4. **import** java.util.Timer;
5. **import** java.util.TimerTask;
7. **import** org.jfree.chart.ChartFactory;
8. **import** org.jfree.chart.ChartFrame;
9. **import** org.jfree.chart.JFreeChart;
10. **import** org.jfree.chart.StandardChartTheme;
11. **import** org.jfree.chart.plot.CategoryPlot;
12. **import** org.jfree.chart.plot.PlotOrientation;
13. **import** org.jfree.data.category.CategoryDataset;
14. **import** org.jfree.data.category.DefaultCategoryDataset;
16. **import** dao.DAO;
17. **import** entity.Entity;
19. **public** **class** charts {
20. @SuppressWarnings("unused")
21. **public** **static** **void** main(String[] args) {
23. StandardChartTheme mChartTheme = **new** StandardChartTheme("CN");
24. mChartTheme.setLargeFont(**new** Font("黑体", Font.BOLD, 20));
25. mChartTheme.setExtraLargeFont(**new** Font("宋体", Font.PLAIN, 15));
26. mChartTheme.setRegularFont(**new** Font("宋体", Font.PLAIN, 15));
28. ChartFactory.setChartTheme(mChartTheme);
29. CategoryDataset mDataset = GetDataset();
30. JFreeChart mChart = ChartFactory.createLineChart(
31. "电压波动折线图",//图名字
32. "时间",//横坐标
33. "电压",//纵坐标
34. mDataset,//数据集
35. PlotOrientation.VERTICAL,
36. **true**, // 显示图例
37. **true**, // 采用标准生成器
38. **false**);// 是否生成超链接
40. CategoryPlot mPlot = (CategoryPlot)mChart.getPlot();
41. mPlot.setBackgroundPaint(Color.LIGHT\_GRAY);
42. mPlot.setRangeGridlinePaint(Color.BLUE);//背景底部横虚线
43. mPlot.setOutlinePaint(Color.RED);//边界线

46. ChartFrame mChartFrame = **new** ChartFrame("电压波动折线图", mChart);
47. mChartFrame.pack();
48. mChartFrame.setVisible(**true**);

51. **while**(**true**){
53. mPlot.setDataset(GetDataset());

56. **try** {
57. Thread.sleep(1000);
58. } **catch** (InterruptedException e) {
59. // TODO Auto-generated catch block
60. e.printStackTrace();
61. }
62. }
64. }


68. **public** **static** CategoryDataset GetDataset()
69. {
70. DefaultCategoryDataset mDataset = **new** DefaultCategoryDataset();
72. ArrayList<Entity> ar=**new** DAO().getTen();
73. **int** index = ar.size()-1;
74. **for**(Entity ne; index>=0 ;index--) {
75. ne = ar.get(index);
76. mDataset.addValue(Double.valueOf(ne.getPower()), "设备"+ne.getId(), ne.getTime().split(" ")[1]);
77. }
79. **return** mDataset;
80. }
81. }