公开论文

涉密论文



http://www.nit.net.cn/__local/C/EE/E5/AEF522EFB83938A8BD165923DEC_001BEC5E_9976.png

**毕业论文（设计）**



题 目 信号检测分析系统的设计与实现

姓 名 宋一得

学 号 3180421011

专业班级 计算机科学与技术求是181

指导教师 陈根浪

学 院 计算机与数据工程

日 期 2022/4/25

浙大宁波理工学院本科毕业论文（设计）承诺书

1.本人郑重承诺所呈交的毕业论文（设计），是在指导教师的指导下严格按照学校和学院有关规定完成的。

2.本人在毕业论文(设计)中引用他人的观点和参考资料均加以注释和说明。

3.与我一同工作过的同学对本研究所做的任何贡献均已在论文中做了明确的说明并表示谢意。

4.本人承诺在毕业论文（设计）工作过程中没有抄袭他人研究成果和伪造数据等行为。

5.若本人在毕业论文（设计）中有任何侵犯知识产权的行为，由本人承担相应的法律责任。

6.本人完全了解浙大宁波理工学院有权保留并向有关部门或机构送交本论文（设计）复印件和电子文档，允许本论文（设计）被查阅和借阅。本人授权浙大宁波理工学院可以将本论文（设计）的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或者扫描等复制手段保存和汇编本论文（设计）。

作者签名： 日期： 年 月 日

致谢

能在大学期间取得如此大的进步，从一个迷茫无知的少年，成长为一个有理想，有抱负的青年，我要感谢理工老师们的悉心指导，感谢每一位在我成长路上的陪伴者。

首先我要感谢我的指导老师、班导师陈根浪教授和实验室指导老师宋光慧老师。你们除了在专业知识上倾囊相授，给了我很大的启发，在生活上也对我处处体贴。在今年如此艰难的就业环境下，两位老师为我四处推荐，为我加油鼓劲，帮助我顺利找到工作。

我要感谢信息学院的赵红祥老师，在我整理毕业设计期间为我提供了专业性的指导以及写作方面的帮助。所有帮助过我的老师们，我也将负载着他们的期望奋勇前行。

我要感谢我们的班长——周立旋同学，他为我们求是班默默付出，

我要感谢我的两位好朋友：我的同学赵泓睿和我的室友王涛。我在教室里听闻风云故事，在寝室里讨论人生意义，在快活的静夜中尽情对酒当歌，在思辨的穹宇下空叹人生几何。感谢你们，让我不局限于专业知识，更是坚定了自己脚下的道路。

最后，我要感谢实验室无数个日夜里与我一起并肩作战的黄广进、方逸雄、王锐、郑明辉，还有我的学弟张锦航。在遇到困难时，我们互帮互助，在收获成功时，我们分享喜悦。是你们让我在迷茫的深夜里感到自己不是一个人在战斗，祝你们学业有成，前程似锦！

摘要

自20世纪30年代示波器发明以来，这种能够显示电压信号动态波形的电子测量仪器在科学、医学、工程、汽车、电信行业等领域发挥了重大作用。时至今日，示波器的性能和功能已得到充分的发展，然而作为硬件设备，其笨重的体型和笨重的数据传输方式与当下灵活的信息交互时代格格不入。随着计算机技术以及虚拟化技术的发展，一种用软件方式实现示波器功能的名为虚拟示波器的仪器极大提高了示波器的便携性和扩展性。

本项目旨在从软件开发角度设计一款扩展良好的虚拟示波器软件，同时具备信息管理的功能，根本目的在于探索示波器设备与当前互联网技术结合的可能性，具体内容描述如下：

1. 首先是系统设计，这部分对整个项目技术实现思路进行了大致说明，阐述了软件编程融入硬件通信后其中产生的重难点和解决思路，对于程序设计中关键部分的实现技术也进行了详细的展开描述。

2. 虚拟示波器部分，本项目采购了虚拟示波器硬件用以采集数据，以python为络合剂，整合了Qt以绘制用户界面，用matlablib库完成了基本的信号分析功能。

3. 管理系统部分，本项目以MySQL作为数据存储工具，用互联网的管理系统模式将虚拟示波器融入到实际雷达电路分析检测的应用中，配备数据导入导出以及信息管理功能，真正实现了便携的数据交互。

本文描述了如何基于程序设计思想开发可用的虚拟仪器系统以及在原有系统基础上如何开发新功能，信号分析算法的探究不在本文讨论范围内。

关键词：示波器；虚拟化；管理系统；物联网

**Abstract**

Since the invention of the oscilloscope in the 1930s, this electronic measuring instrument capable of displaying the dynamic waveform of a voltage signal has played a major role in the fields of science, medicine, engineering, automotive, and telecommunications. Today, the performance and functions of the oscilloscope have been fully developed, but as a hardware device, its bulky size and bulky data transmission method are out of tune with the current era of flexible information interaction. With the development of computer technology and virtualization technology, an instrument called virtual oscilloscope that realizes the function of oscilloscope by software greatly improves the portability and expansibility of oscilloscope.

This project aims to design a well-expanded virtual oscilloscope software from the perspective of software development, and at the same time has the function of information management. The fundamental purpose is to explore the possibility of combining oscilloscope equipment with current Internet technology. The specific content is described as follows:

1. The first is the system Design, this part gives a general description of the technical realization idea of the whole project, expounds the difficulties and solutions after the software programming is integrated into the hardware communication, and also describes the realization technology of the key parts in the program design in detail.

2. For the virtual oscilloscope part, this project purchased virtual oscilloscope hardware to collect data, used python as a complexing agent, integrated Qt to draw the user interface, and used the matlablib library to complete the basic signal analysis functions.

3. In the management system part, this project uses MySQL as a data storage tool, and uses the Internet management system mode to integrate the virtual oscilloscope into the application of actual radar circuit analysis and detection.

It is equipped with data import and export and information management functions, which truly realizes portable data exchange. This paper describes how to develop a usable virtual instrument system based on programming ideas and how to develop new functions on the basis of the original system. The exploration of signal analysis algorithms is not within the scope of this paper.

**Keyword:** Oscilloscope; Virtualization; Management system; Internet of things

目录

[1 绪论 3](#_Toc103003377)

[1.1 研究背景与意义 3](#_Toc103003378)

[1.2 国内外研究现状 6](#_Toc103003379)

[1.3 论文内容和结构 8](#_Toc103003380)

[2 系统设计 9](#_Toc103003381)

[2.1 准备工作 9](#_Toc103003382)

[2.1.1 需求分析 9](#_Toc103003383)

[2.1.2 软件技术选型 10](#_Toc103003384)

[2.1.3 硬件选型 10](#_Toc103003385)

[2.2 系统结构设计 11](#_Toc103003386)

[2.2.1 逻辑结构 11](#_Toc103003387)

[2.2.2 物理结构 12](#_Toc103003388)

[2.3 虚拟示波器结构设计 14](#_Toc103003389)

[2.4 数据库设计 14](#_Toc103003390)

[2.4.1 角色表 15](#_Toc103003391)

[2.4.2 用户表 15](#_Toc103003392)

[2.4.3 采集地点表 16](#_Toc103003393)

[2.4.4 信号类型表 16](#_Toc103003394)

[2.4.5 历史信号表 16](#_Toc103003395)

[2.4.6 历史信号数据表 17](#_Toc103003396)

[2.4.7 信号库表 17](#_Toc103003397)

[2.4.8 信号库数据表 17](#_Toc103003398)

[2.4.9 菜单表 18](#_Toc103003399)

[2.4.10 权限表 18](#_Toc103003400)

[3 虚拟示波器 19](#_Toc103003401)

[3.1 硬件驱动及SDK 19](#_Toc103003402)

[3.1.1 设备介绍 19](#_Toc103003403)

[3.1.2 驱动、API和固件 20](#_Toc103003404)

[3.2 数据采集和持久化 21](#_Toc103003405)

[3.3 图像展示和继承复用 24](#_Toc103003406)

[4 信号管理系统 28](#_Toc103003407)

[4.1 分页表格 28](#_Toc103003408)

[4.1.1 分页器 29](#_Toc103003409)

[4.1.2 表格封装 33](#_Toc103003410)

[4.2 基础信息管理 36](#_Toc103003411)

[4.2.1 检测点位 37](#_Toc103003412)

[4.2.2 信号类别 37](#_Toc103003413)

[4.2.3 用户管理 38](#_Toc103003414)

[4.3 信号相关组件 38](#_Toc103003415)

[4.3.1 历史数据 39](#_Toc103003416)

[4.3.2 诊断维修 40](#_Toc103003417)

[4.3.3 标准信号库/异常信号库 40](#_Toc103003418)

[5 结论和展望 42](#_Toc103003419)

[6 参考文献 43](#_Toc103003420)

[附录A python ORM实现 45](#_Toc103003421)

# 绪论

## 研究背景与意义

二十世纪七十年代以来,数字技术理论、数字电路、大规模集成电路以及微处理器技术高速发展，在这样的环境下，数字存储示波器应运而生并迅速发展起来。这种设备随即在科学实验和国防军工装备的瞬态信号检测、分析和实时处理显示领域发挥了重要作用。示波器作为一种时域电子测量仪器，它因其直观、准确、通用等优点受广泛使用。然而，在如今信息化时代高速发展的背景下，传统示波器很快已经不再能满足新兴技术发展的需要。

传统示波器体型笨重，高性能多功能数字示波器造价不菲，外在的制约条件和高昂的成本大大限制了它作为传感设备在物理世界中的大规模应用；一种虽然性能功能略有不足，但轻便廉价的手提型示波器适应了这部分市场。但手提型示波器依然没有摆脱对硬件的过度依赖，完全基于嵌入式的架构注定其无法灵活扩展；虚拟示波器的出现彻底改变了这种情况。作为虚拟仪器，它可复用地在计算机上增加硬件和软件，并由计算机显示器提供仪器可见操作界面。基于PC平台，其大部分功能由软件实现，这同时意味着它拥有比手持示波器更具竞争力的优惠价格[1]。



图 1.1 本项目所选用的虚拟示波器

当然虚拟示波器也不是万能的。首先是硬件限制，其最大采样率往往无法超过100MHz，缺少嵌入式高速计算和渲染的支持，设备的准确率和高性能也将大大折扣；虚拟示波器需要由上位机进行控制，而其中的数据通信又将面临数据传输和校验的性能损耗。由此看来，虚拟示波器更适合精度速度要求不高，而对数据共享性，业务需求灵活性有较高要求的场景，因此不关注应用场景却纠结于技术本身是一种本末倒置的行为。

随着军队信息化建设的不断深入，如何依靠各种先进的技术手段，尤其是运用信息技术的最新成果，来进一步提高通信保障和再生能力，是摆在军队各级通信部门领导面前的当务之急[2]。

在军队信息化的庞大工程中，数据采集是其中重要的一部分，部队运作期间将采集分析大量数据。现有装备虽然能够满足日常测量管理的需求，然而需要专业人员付出大量时间精力，且仍旧可能导致不准确、不及时，数据采集自动化的需求十分强烈。理想情况下，可以设计一套系统能够对重要目标环境实现环境数据自动化采集（温湿度、气体浓度、电流、电压等），使管理者及时掌握设备环境实际情况变化以及设备的运行实际状况。根据现场环境要求，设定各种环境参量的上下限以及各种报警参数，当监测数据达到设定触发条件时，即可产生报警信号并联动视频监控，把现场图像及时传给主控中心和各级分控中心值班人员得到及时处理。对特殊地点、敏感区域通过智能视频监控安全策略，可设定各种事件安全侦测规则，达到某个区域的智能化防范管理。

从本人不甚详尽的了解情况来看，我国基础军工设施的自动检测维修工作并不见长。以雷达电路维修为例，这样的重要基础设施目前大部分情况下都是出现故障后由专业人员进行维修[3]，这种人力处理的方面明显存在以下几个缺点：

1. 较高的技术要求。信号分析通常需要信息工程专业相关人才，功能强大的示波器更需要经验丰富的专业人员才能驾驭，对于专业人才短缺的军队来说是相当大的阻力。

2. 人工检测不准确、不及时。纵使采用最优秀的人才也难免出现疏漏，受制于生理限制，人类无法做到永远准确并且随时待命。

3. 数据传输繁琐，共享性差。专业人士使用的示波器大多具有数据存储的功能，然而数据的转储往往需要经历安装设备驱动，连接USB口，运行示波器配套软件，通过厂商自定的协议读取并保存为指定的数据格式如csv。这个过程不仅繁琐而且效率低下，支持的文件格式也有限。

本项目正是基于这些痛点，基于物联网技术提出一套电路故障智能检测系统的解决方案。本项目以雷达电路检测维修为切入点，使用示波器设备对雷达电路运行状况进行实时监测，将雷达使用过程中产生的信号进行采集分析，通过人工智能算法分析电路中可能存在的异常随时引发警报，具体来说就是将采集到的信号与正常信号分别提取其特征进行对比，使用某种算法分析信号是否异常以及异常可能发生的位置。对于电路中可能存在的隐患，系统可以提前分析并予以警告，这将大大降低人力检测维修成本，使得基础设施更加可靠稳定。

可以设想将计算机技术运用于信号检测分析，以软件的方式解决硬件的诸多痛点，不仅能够大大降低系统成本，还能保持系统的不断迭代发展，提高系统的容错率，为技术的发展提供无限的可能。首先，考虑虚拟化技术，用软件的方式实现物理示波器的功能可以大大降低设备成本；基于PC实现的程序可以很大程度上发挥计算机的优势，开发出易用的，用户友好的程序并辅以指导手册，提供在线服务等，能够大大降低用户使用的门槛；人类无法机械地工作，而这正是机器的功能，机器能够24小时监测异常情况，辅以专业人员修正，系统可靠性将得到大大提升；计算机结合互联网技术能够轻松解决数据传输的痛点，在系统上搭建可靠的通信网络甚至将系统本身设计为庞大的物联网系统在如今已并非难事，届时数据的传输只需一条小小的指令或简单地按下一个按钮即可获得各种格式的信号数据[4]。

另外，将目前成熟的信息管理系统模式应用于示波器信号数据，能更有效地管理设备采集的数据，便于后续实现历史数据查阅、数据统一分析处理、数据访问权限隔离等功能，这是传统的信号分析系统不可能具备的优势。

从本课题研究的背景来看，本项目与物联网模式非常相似，似乎是向物联网大家族引入了示波器这么一位新成员。然而站在历史的角度上看，示波器这位成员可不“新”，甚至应该用“老”、“传统”这类形容词。作为科学军工的重要仪器，从阴极射线应用的时间——十九世纪末期开始计算，示波器的诞生历史已有百年之久。可为何如今摄像头、电子横杆遍布街头，而不见示波器的踪影？我认为主要是两方面原因：首先传统示波器成本过高，难以直接应用到大规模网络当中，此时虚拟示波器的优势就得以发挥；其次示波器主要应用于科学研究、军工制造等领域，由于示波器本身的复杂性和专业性，其商业价值没有被充分发掘。商业产品之所以能成功很大程度上得益于其易用性，切不可盲目追求技术的大而全，放弃了用户的使用体验。

借助此次机会，本课题尝试探索这种廉价的示波器探索示波器的新兴赛道，力求开发出易用的系统。由于笔者并非信号专业人士，在信号处理分析问题上无法提供更有效、更切合需要的信号分析算法，本项目在这方面稍作简化，将主要目的放到设计一套可扩展的，可迭代的，稳定的，易于非计算机专业人士上手的系统架构，为后续提供更丰富的信号分析功能创造环境。

综上所述，该项目的主要意义在于将计算机技术与信号分析领域相结合，借鉴物联网架构的经验，提供了一种电路自动检测分析的解决方案，不仅是军工领域，任何暴露在物理环境较为恶劣的电路而系统对可靠性要求的情形下，这套系统都能很好地胜任。

## 国内外研究现状

台式示波器往往功能参差不齐，对于二次开发功能的支持更是聊胜于无，用互联网时代的眼光来看，传统的台式示波器在扩展性、移植性等方面注定跟不上时代的潮流。

1986 年，美国 NI 公司率先提出虚拟仪器的概念，随后开发了当前虚拟示波器领域最炙手可热的 Lab View软件，仪器产业从数字化仪器开始向虚拟仪器过渡[5]。通信技术、测量技术和计算机技术在虚拟仪器的协调下产生了前所未有的融合，传统仪器的观念发生了巨大的变化，例如，虚拟示波器软件已经不仅仅局限于 C/S 架构，也可以采用B/S 的架构设计[6]。国内的虚拟仪器产业与国外存在较大差距，且发展态势不容乐观。我国计算机、通信电子等产业的起步较晚，对虚拟仪器上的研究更是直到上世纪末才开始，目前我国从事虚拟仪器研究和开发的科研单位数量很少，只是起步阶段。在国外优秀产品的挤压下，我国虚拟仪器的竞争力相当有限，产业发展缓慢，且没有同国际虚拟仪器产业接轨，大部分只是作为国外知名的虚拟仪器厂家的销售代理商，没有自主知识产权。

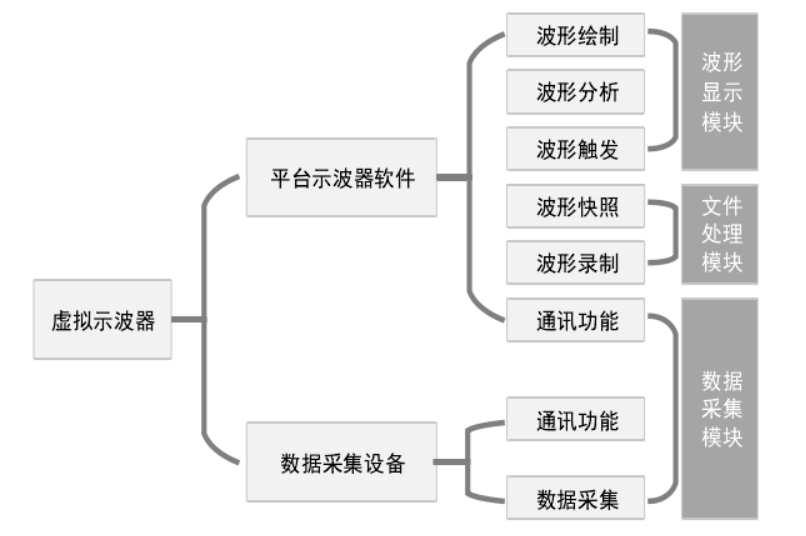


图 1.2 虚拟示波器系统结构设计

图1.2展示了[7]中关于虚拟示波器系统结构的设计。不同的课题对于虚拟示波器设计的侧重有所不同，但都局限在虚拟示波器实现的架构创新，或是将虚拟示波器迁移到新的平台上，示波器本身移植性差，扩展性低的状况没有得到改变。

Lab VIEW虽然功能强大，但作为一款专业工业软件，其费用并不亲民，更重要的是，依靠一款美国完全自主产权的软件发展必然受到其技术方面的制约，想要在一个领域发展到世界前列必然要摆脱国外的技术垄断，形成一套自主研究、自主开发、自主销售的产业机制。

此外，虚拟示波器软件的易用程度和开发效率也是十分值得考虑的因素。从易用程度上来说，目前大多虚拟示波器都针对信号分析专业人士设计，其软件界面通常都是专业技术名词，且缺少软件帮助功能和软件使用说明，软件使用门槛较高。从开发效率来看，虽然Lab VIEW开发效率高，但始终是站在硬件层面进行软件设计，很难对数据进行更进一步的汇总分析，能否设计出扩展良好的软件更是要打上一个问号。

正如上文所言本项目似乎是将物联网技术创新性地应用到了示波器上。其实示波器与物联网的碰撞并非首次[8-10]，曾有研究尝试将示波器部署到物联网络当中，站在网络建设和数据流通的层面设计了整个系统。在如今的视角看来，不论是应用方式过于简单粗暴，还是不愿摒弃示波器的复杂性，都足以杀死这种方式继续发展的可能。国内示波器+物联网的领域尚处蓝海，本项目更是创新性地将物联网应用于虚拟示波器。

## 论文内容和结构

本课题的主要内容是基于虚拟示波器开发一款信号检测分析系统程序，以此说明虚拟示波器在物联网中投产的可行性，并进一步探究分析虚拟示波器在物联网建设项目中的应用前景。

第一章对比分析了几种示波器的优缺点，提出了虚拟示波器更加灵活，更适应数据交互场景。随后以军队信息化建设为切入点，分析了虚拟示波器在物联网中可能存在的应用场景。

第二章首先分析了核心需求，随后描述了项目整体结构的设计，软件硬件选型，数据库设计等内容，简单介绍了项目的重难点及解决思路。

第三章描述了项目中核心功能——虚拟示波器软件部分的实现。章节首先对虚拟示波器硬件使用方式和参数进行了一定说明，随后以源码片段的方式较为详细地解释了程序代码的设计结构，以部分函数为例，阐述了数据采集展示等功能具体是如何实现的。

第四章主要描述了信号管理系统的运行机制和实现原理。章节特别展开说明了本项目中基础数据表格展示的支撑组件是如何封装实现的，并以分页器为例展示了底层组件封装时需要考虑到的种种细节。最后展示了信号分析相关的几个额外功能。

第五章总结了本次课题的成果并对虚拟示波器的发展趋势绘制了一份笔者心中的美好蓝图。

# 系统设计

## 准备工作

### 需求分析

本课题的主题为设计并实现信号检测分析系统，一般来说完整的系统需要较为详尽的需求说明文档具体指出需要何种类型的功能，然而本课题的研究领域处于一片蓝海，国内外对标的应用相对较少，大部分需求尚不明确。本项目的研究尚处探索阶段，且人力物力成本有限，应以实现具备核心功能的最小运行单元为目标，视情形在这基础上补充额外功能，因此简单讨论后我们总结了以下关键需求：

1. 考虑到该项目后续的发展，应用程序宜部署到更加稳定的Linux系统上，因此技术和设备选型应以Linux为基准。考虑到信息交互性和安全性，系统应具备本地存储，数据导入导出能力，并且避免连接到公网。需要具备基础的客户端程序界面。

2. 设备自检功能，能自动检测设备连接状态，设备连接到计算机后，程序能自动安装驱动程序，使设备进入可用状态。

3. 显示实时信号波形图像，频谱图像，提供必要的示波器参数和图像参数如电压、时基、示波器采样率等参数的调整功能。

4. 信号类别、检测地点、用户等基础数据的管理，视具体情况提供增删改查功能。

5. 提供信号采集按钮，将一定时间内采集到的信号数据持久化存储，分为历史数据和信号库数据。采集的数据自动进入历史数据表，分析正常的数据可导入到标准信号库，分析异常的数据可导入到异常信号库。

6. 提供历史数据与信号库数据分析对比的功能，采用信号处理算法计算出两份数据的相关程度，将低于阈值的历史数据标记为异常。

7. 页面、数据访问权限隔离，提供管理员角色能对整个系统进行管理。

此处提到的需求在后文也会提到，并以需求编号的形式进行引用。

### 软件技术选型

有关需求1实际上是需要确定开发的语言、工具等环境。出于操作系统环境的要求，可以选用主流的跨平台开发工具，如UI可以选择Qt进行开发[11]，那么编程语言自然就会选择与Qt配合较好的C++或python。python作为胶水语言，能灵活高效地整合各种技术。出于开发效率的角度，简洁而强大的python是本项目的首选编程语言。python中使用Qt需要引入pyqt库，下文将多次提及pyqt这个词。

数据存储可以选用开源的关系型数据库MySQL，MySQL作为国内中小企业应用最广泛的数据库，其优势已经无需赘述，是理想的数据存储介质。

设备通信可以选用python的libusb库调用设备的API。

图像处理和展示可以选用python的matlablib库。matlab是通信专业的强大工具，在许多研究中都证实了其具备虚拟示波器的开发能力[12-13]，python中的matlablib库正好可以引用成熟的数据分析算法。

### 硬件选型

首先是设备选型，物理数据的采集必然离不开采集设备，信号采集的主要设备为示波器。传统的示波器在成本、传输、扩展性等方面都存在明显的不足，本文的重点就是虚拟示波器。

虚拟示波器硬件选用了国产的Hantek6022BL设备，这种价格不到400元的低端设备无疑是我们实验型课题的理想器材。更重要的是，国内仪器供应商提供的软件都以Windows系统为主，甚至少有完整的API接口提供给用户二次开发，而具备这些条件的国外虚拟示波器又普遍价格过高。虽然Hantek6022BL同样没有Linux的驱动支持，所幸我们在GitHub上发现了国外电子爱好者为这个系列开发的虚拟示波器开源软件和SDK（软件开发工具包），经过实践后确实可用，这款设备便成为了本项目的不二之选。能站在如此多巨人的肩膀上是计算机工作者的一大幸事。

项目的主要成本来源。低端的虚拟示波器售价不超过400元，高端虚拟示波器成本也在2000元左右。系统环境仅需一台笔记本电脑，操作系统为Linux/Ubuntu，内存8G，CPU4核，网络传输只需设备自带的USB2.0线即可。

## 系统结构设计

本项目虽说结合物联网技术，然而开发资源着实有限，试验性工程的目的在于探索其可行性。因此系统设计的目标并非大规模物联网系统，而是以单机应用为前提，实现最小可执行单元，以敏捷型地，小步快走的方式验证这种产品理念。

Qt在Ubuntu上和Windows上运行的视觉效果有一定差异，从文档编写便携性的角度出发，大部分系统界面图片展示所处的环境为Windows。

### 逻辑结构

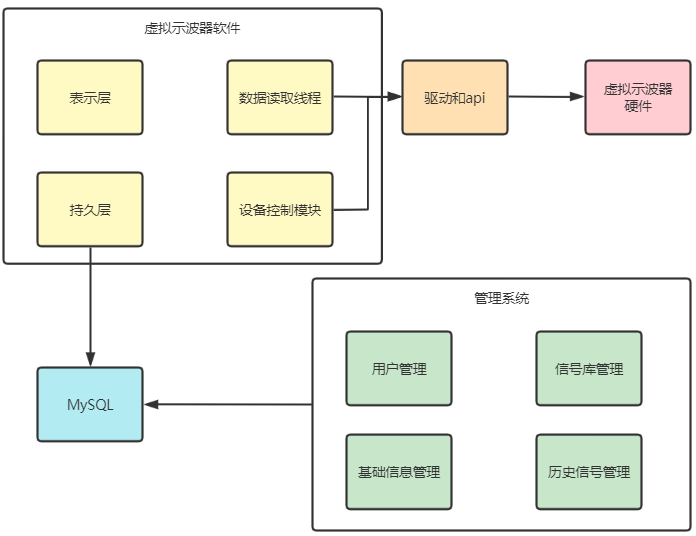


图 2.1 逻辑结构图

如图2-1所示，项目从逻辑上可以分为虚拟示波器和管理系统两个部分。其中虚拟示波器又包括软件和硬件部分以及硬件配套的驱动和相应API。其中虚拟示波器软件和管理系统是本项目中程序实现的主要部分，图中其他内容均来自于开源社区的贡献以及设备供应商。从另一个角度来说，虚拟示波器硬件包含了数据采集和部分下位机的功能，开源社区帮助我弥补了下位机的功能，屏蔽了许多底层通信的难度和复杂性，而本项目所要达成的目标是实现上位机程序并应用到具体场景中。

### 物理结构

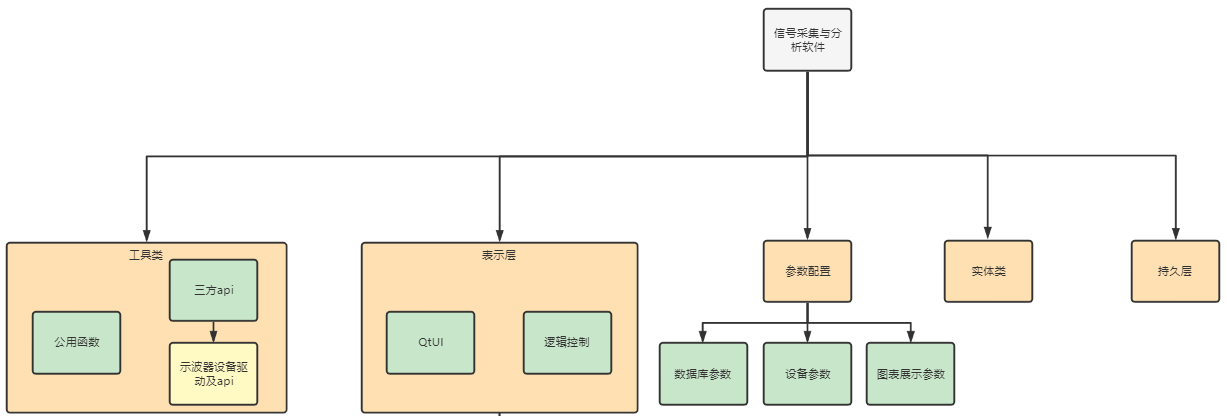


图 2.2 整体物理结构图

如图2.2所示，虽然逻辑上可以将项目分为两个部分，然而业务逻辑上的模块在代码层面有时是强依赖的，逻辑上理清每个模块功能相对简单，而设计良好的代码结构很难。本人仅凭有限的系统设计能力，尝试设计了如图所示的小型应用能够适用的代码结构。由于笔者手头python开发资料些许稀疏，没有成熟的项目构建框架，因此除去业务逻辑的编写，我还需要投入一定精力着眼于项目框架的搭建。为此，我将整个系统代码分为五个主要部分：

1. 工具类。由于本项目研究方向偏向底层并且相关研究较少，可用的API功能往往较为简单，此时将使用频繁的功能封装起来，将逻辑相似的流程抽象成接口，这些工作将大大提高代码的复用性。
2. 表示层。包含了QtUI设计文件（.ui文件）和逻辑代码文件（与之对应的.py文件），ui文件用于生成大致的用户界面框架，py文件用于对组件进行精确的控制。
3. 参数配置。包含数据库连接参数配置、图像显示参数配置、设备参数配置等，同时包括开发环境和生产环境配置的区分，以便部署时快速切换。
4. 实体类。大部分是业务实体的抽象，也包含设备操作的封装和应用广泛的结构体。
5. 持久层。编写SQL语句以及简单的数据库操作逻辑，引用了pymysql库以及工具类中封装的数据库连接池。

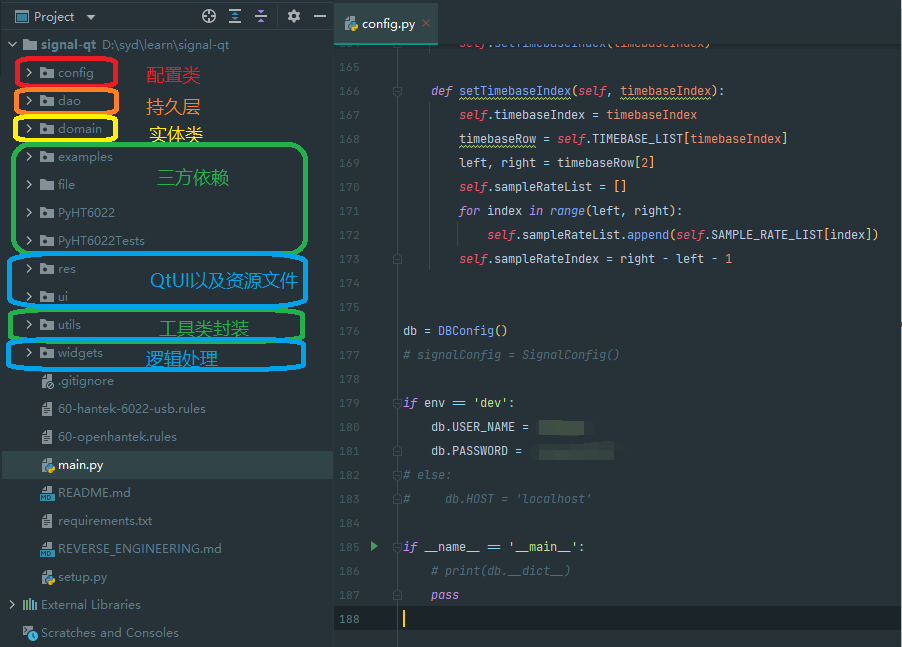


图 2.3 代码层次以及配置文件的一部分

以上介绍了本项目代码整体的物理结构，主要业务逻辑和所有UI界面都包含在表示层中，如下图：

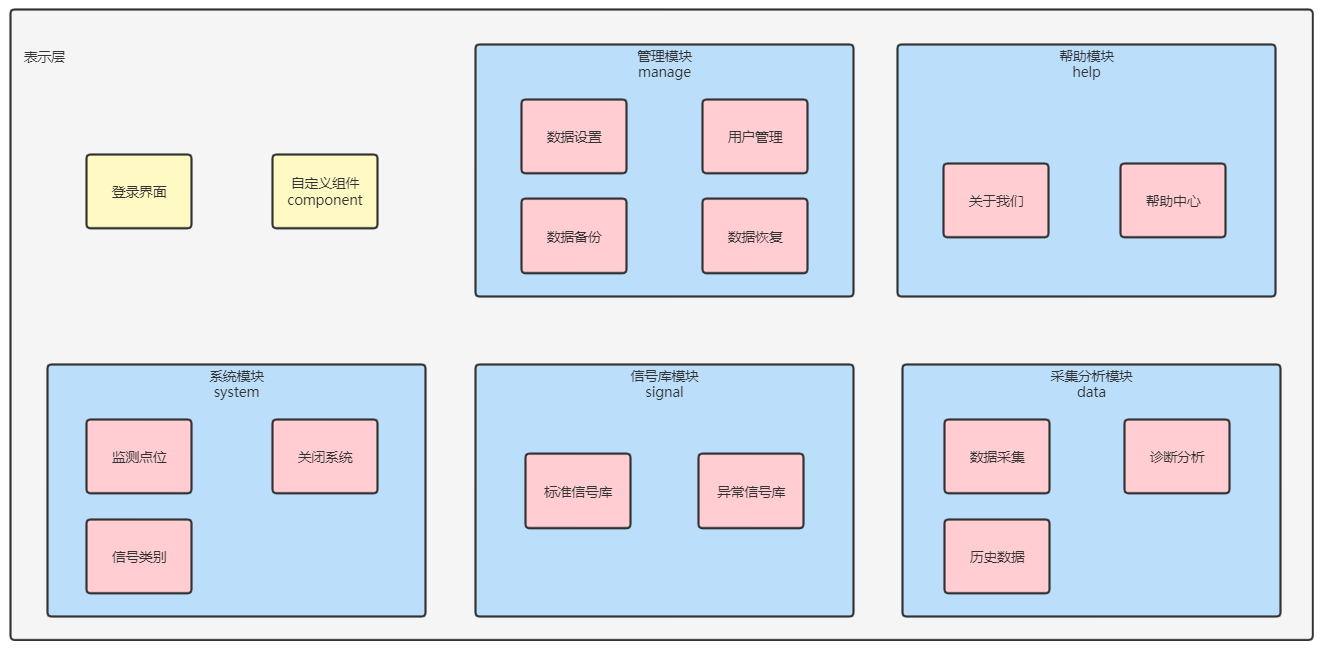


图 2.4 表示层内容

## 虚拟示波器结构设计

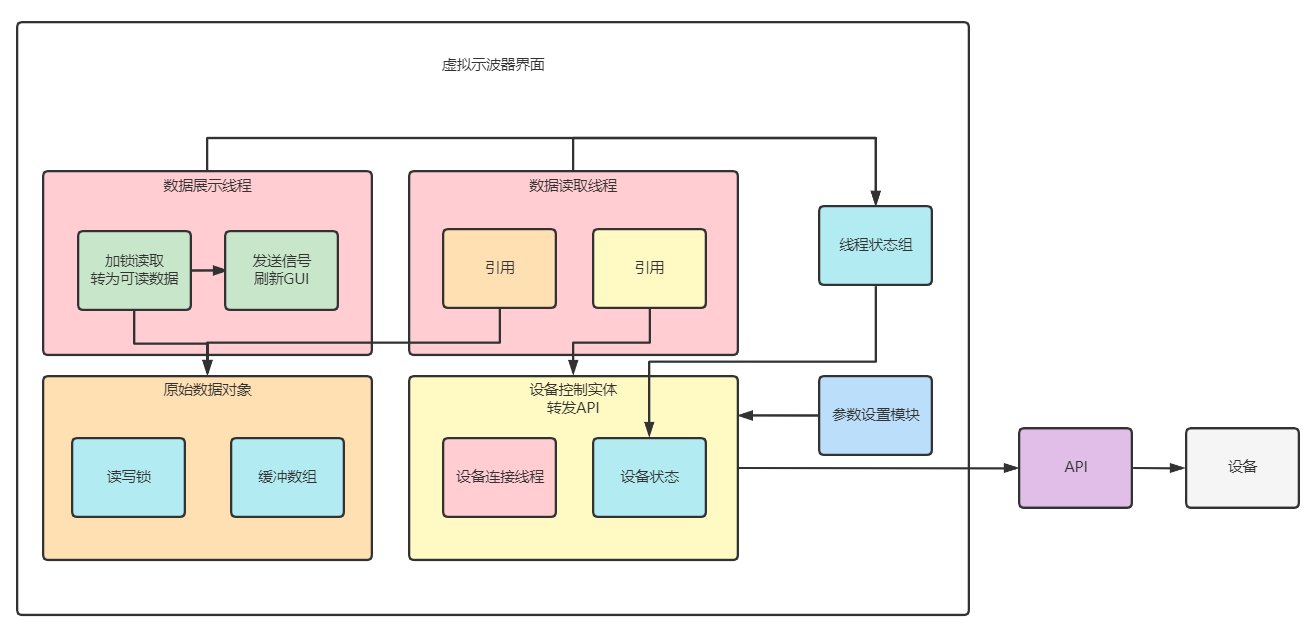


图 2.5 虚拟示波器结构设计图

如图2.5所示，虚拟示波器软件部分的设计一定程度上借鉴了OpenHantek的源码结构，但由于系统的差异性，最终还是融入了大量本人个人的想法，最终形成了这种稍显复杂的结构。

首先是非常明显的读写分离结构，数据的读取和展示（对缓冲数组来说即是写和读）分属两个线程控制。在高采样率下（默认1MHz），每组20000个采样点，每秒就要采集50组数据，而数据图像有时并不需要这么快的刷新频率，一般设置为10-20Hz就已经足够，这是一个读少写多的场景，数据间同步最好的方式是直接用互斥锁。项目中实际运用了Qt提供的ReadWriteLock读写锁实际上是受OpenHantek的影响。

## 数据库设计

最初为了将历史信号与标准、异常信号进行区分，竟然额外使用了一张信号库表，并用复杂的导入导出语句实现了信号入库，一个字段可以解决的问题却这般大动干戈，现在想来设计时考虑还应更加周全。

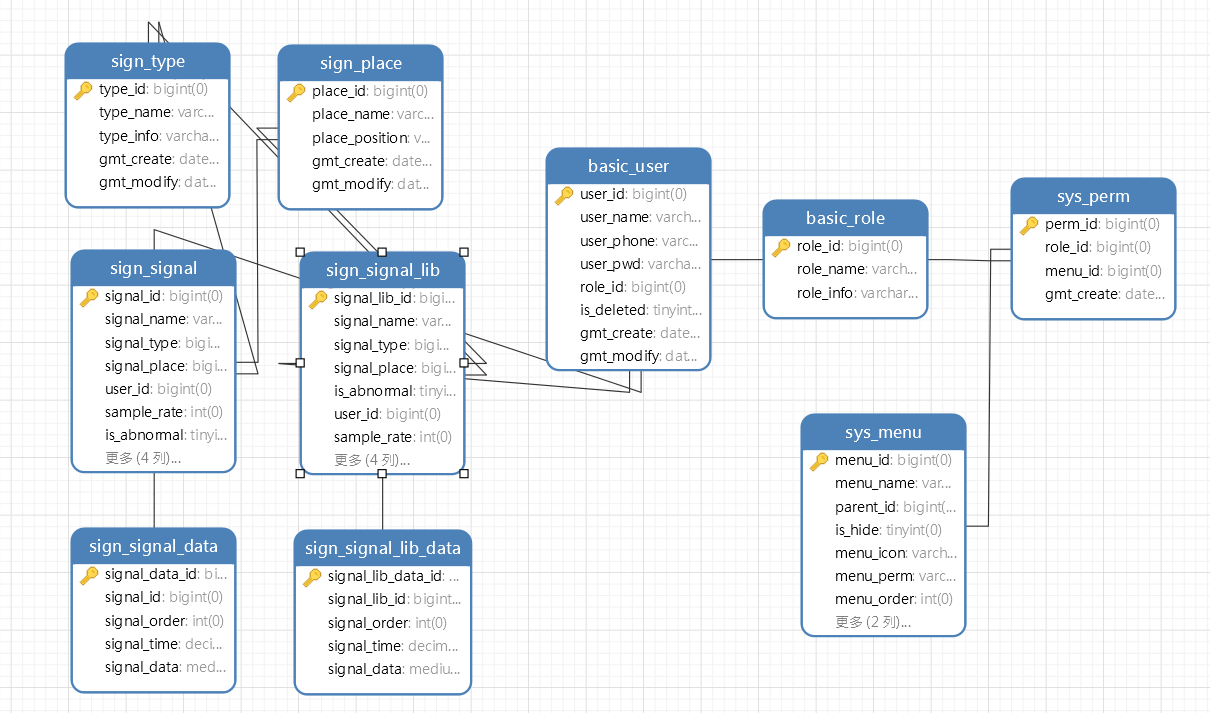


图 2.6 E-R图

### 角色表

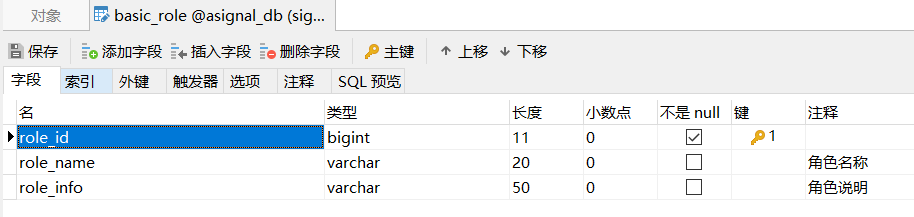


图 2.7 basic\_role

### 用户表

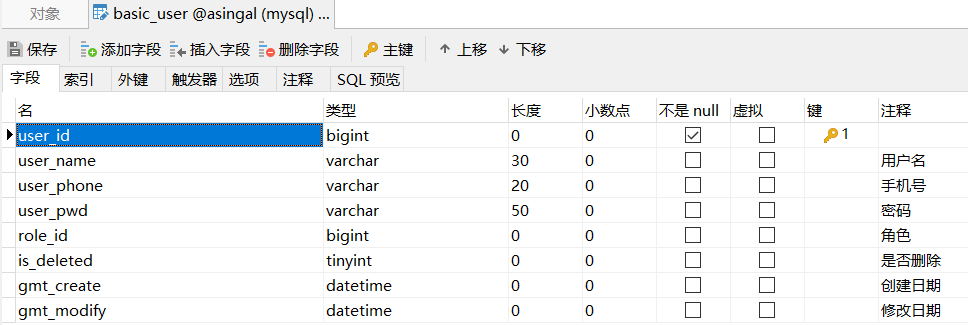


图 2.8 basic\_user

### 采集地点表

数据采集的地点

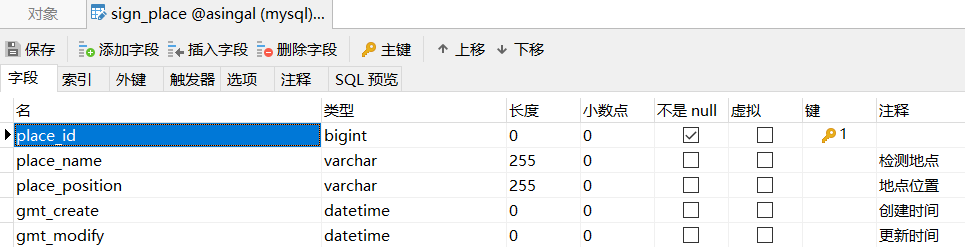


图 2.9 sign\_place

### 信号类型表

采集信号的类型

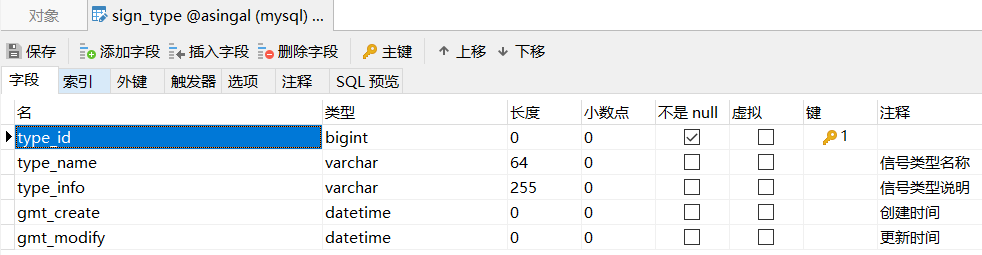


图 2.10 sign\_type

### 历史信号表

采集到的数据，包含采集地点、信号类型和采集者

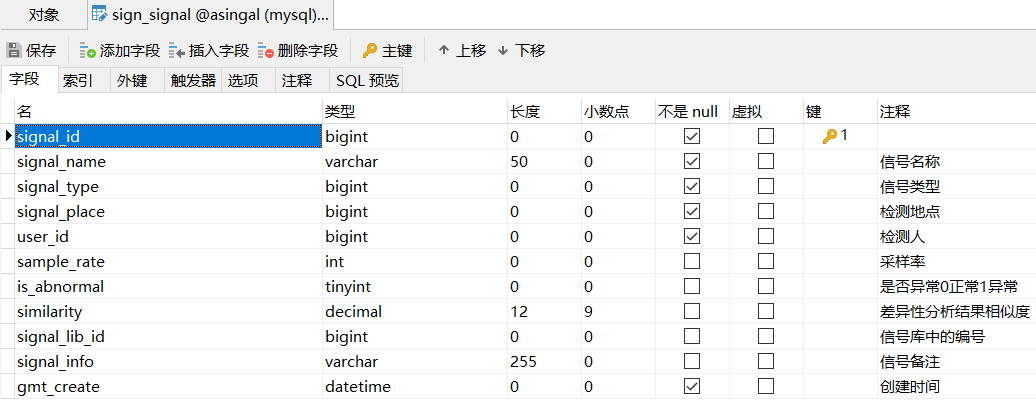


图 2.11 sign\_signal

### 历史信号数据表

保存历史信号的数据

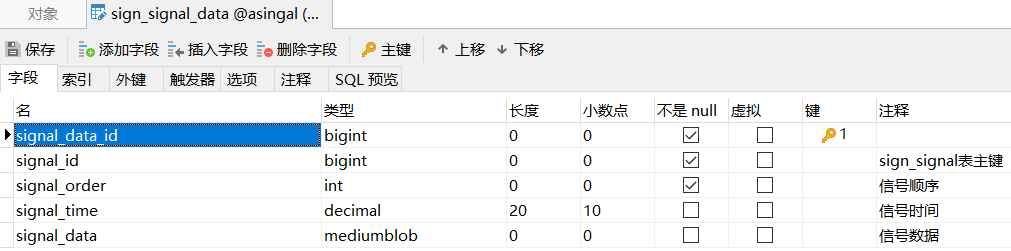


图 2.12 sign\_signal\_data

### 信号库表

标准信号和异常信号

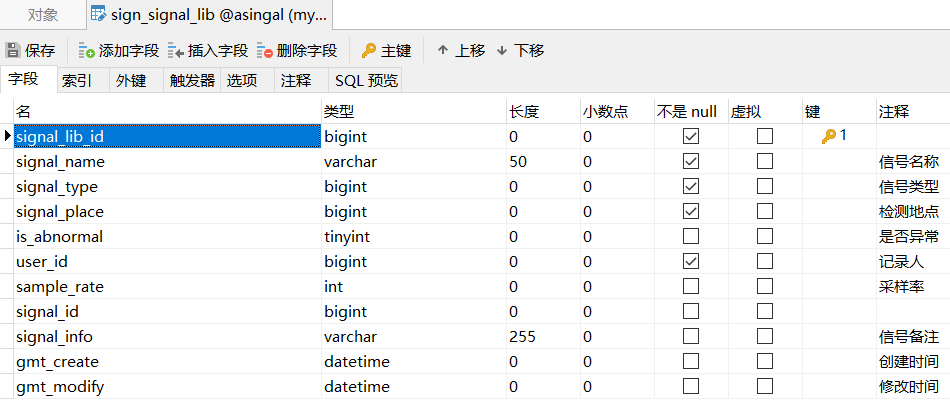


图 2.13 sign\_signal\_lib

### 信号库数据表

保存标准信号和异常信号的数据

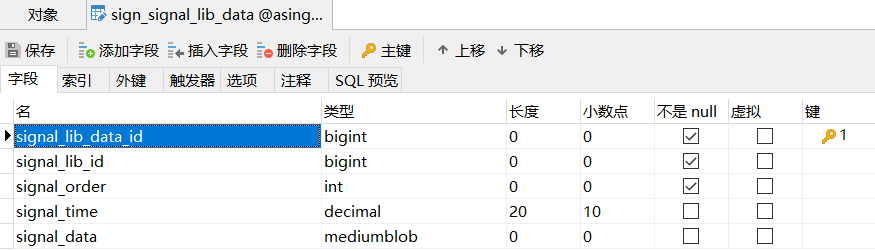


图 2.14 sign\_signal\_lib\_data

### 菜单表

对应界面展示的菜单

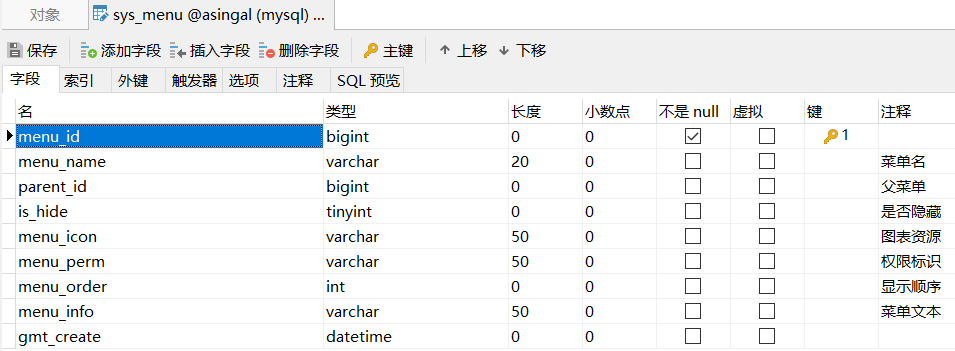


图 2.15 sys\_menu

### 权限表

用户角色能够访问的菜单

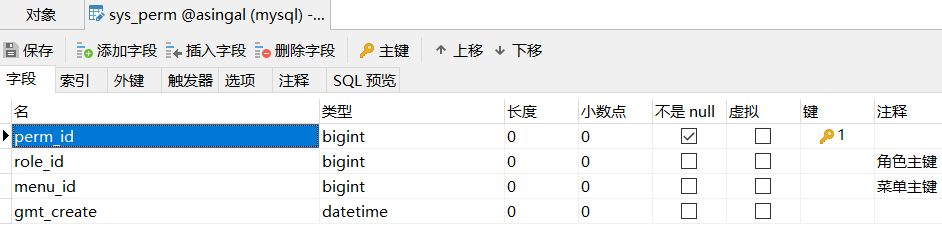


图 2.16 sys\_perm

# 虚拟示波器

虚拟示波器相关的设计部分和第三方硬件驱动、API主要来源于GitHub开源项目：C++编写的虚拟示波器软件项目OpenHentek[<https://github.com/OpenHantek/openhantek>]以及提供了python SDK的Hantek6022API[<https://github.com/rpcope1/Hantek6022API>]。

## 硬件驱动及SDK

### 设备介绍



图 3.1 示波器的两个通道、校准信号和地线

如图3.1所示，数字存储示波器Hantek6022BE（以及类似的Hantek6022BL）允许测量两个电压通道CH1（黄色）和CH2（蓝色）。输入阻抗为1 MΩ| | 25 pF（示波器输入的通用值）。可测量的输入信号范围为±5 V。输入电压超出±35 V的安全范围可能导致永久性损坏！校准输出（蓝色，右侧）提供一个方波0 V/2 V@1 kHz，可用于调整10倍探头的频率补偿[14]。

频率可在32 Hz和100 kHz之间变化，以提供多功能测试信号。每个输入部分都包含一个8位ADC（模数转换器），允许以选定的采样率（10 kS/s…30 MS/s）对输入电压进行采样，并通过USB将数字值传输到PC，在PC上进行处理和显示。

Hantek6022使用USB 2.0高速传输。它必须直接连接到PC，中间不带USB集线器。确保设备不与其他设备共享其高速总线，这可以通过Linux命令lsusb -t进行检查。该设备通过USB供电，一般的电流消耗小于500毫安，因此优质的USB 2.0电缆就足够了，无需使用设备上奇怪的红/黑色Y形电缆。

固件（firmware）是一种写入硬件的EPROM（可擦写可编程只读存储器）或EEPROM(电可擦可编程只读存储器)的程序，是控制硬件的核心部分。固件通常包括驱动、控制、解码、传送、检测等功能，通过固件，操作系统才能按照标准的设备驱动实现特定机器的运行动作。然而Hantek6022系列示波器不会将固件永久存储在闪存或EEPROM（电可擦除存储器）中，因此每次通电后必须上传固件，并将其保存在RAM中，直到断电[15]。如果不同虚拟示波器软件写入了不同的固件，则必须重启示波器硬件的电源，以启用正确固件的自动加载。如果固件上传成功，CH1通道左侧的LED将闪烁红色。

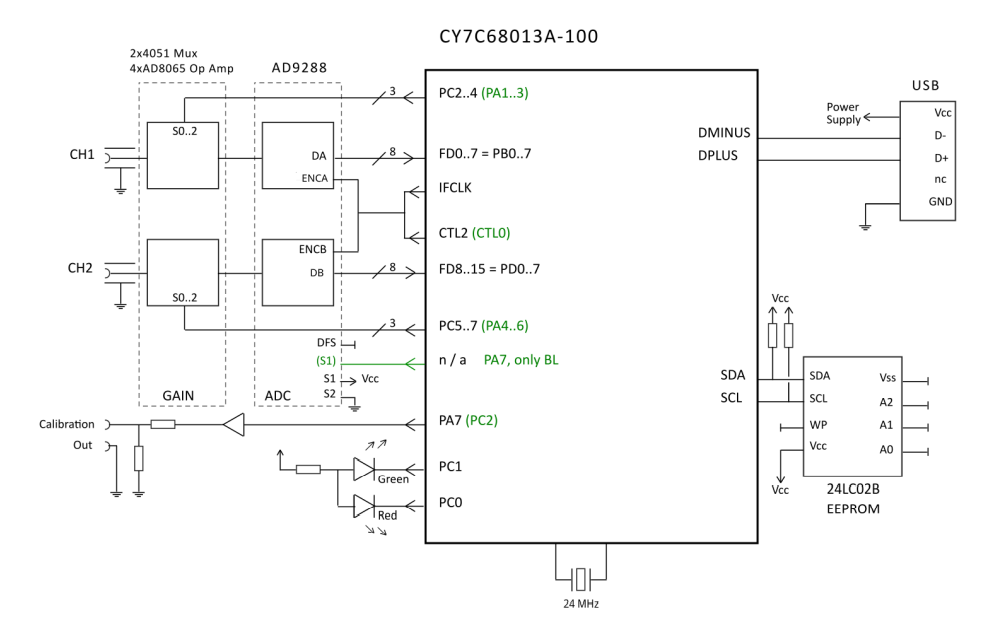


图 3.2 Hantek6022系列电路图

### 驱动、API和固件

在Linux系统上，为了能够正确地通过USB线与设备通信，需要将如图3.3所示的配置文件复制到”/etc/udev/rules.d/”或”/usr/lib/udev/rules.d/”目录下并重新插入设备[16]。

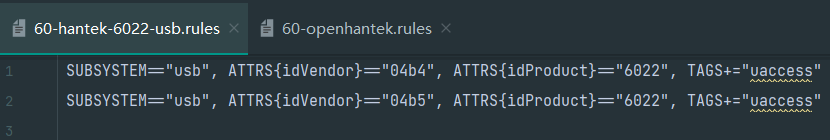


图 3.3 Linux上的设备配置文件

完成这些工作后即可调用API，以下展示了Hantek6022API项目提供的python SDK中的几个关键API：

1. 尝试找到要运行的示波器

*setup(self)*

2. 打开示波器的设备句柄

*open\_handle(self)*

3. 关闭当前示波器设备句柄

*close\_handle(self, release\_interface=True)*

4. 将闪存示波器固件连接到目标示波器设备

*flash\_firmware(self, firmware=default\_firmware, supports\_single\_channel=True, timeout=60)*

5. 从设备读取两个通道的ADC数据

*read\_data(self, data\_size=0x400, raw=False, timeout=0)*

6. 将示波器的ADC计数转换为精确缩放的电压

*adc\_to\_voltage(adc\_count, voltage\_range, probe\_multiplier=1)*

7. 为要采样的范围设置采样率索引

*set\_sample\_rate(self, rate\_index, timeout=0)*

8. 在通道1（CH1）的范围内设置电压比例因子

*set\_ch1\_voltage\_range(self, range\_index, timeout=0)*

如上一小节所说，为了能够与设备正常通信，将汇编编译出的二进制固件程序写入设备的EEPROM是关键步骤，为此需要在第一次连接设备时按顺序调用1、2、3函数。

## 数据采集和持久化

数据的采集部分是系统的关键，接下来这两节将描述本项目如何将虚拟示波器的结构设计转换为真正可以运行的代码。由于API、语言完全不同，系统需求也不尽相同，GitHub上由高级工程师编写的源码更多只是提供了参考价值，笔者只得运用并不熟练的图形开发技能勉强编写出了外观看起来还过得去的软件界面。首先是效果展示。

进入数据采集页面后会自动开启设备连接线程尝试连接设备，一旦连接设备成功即可开启数据读写两条线程，界面随之显示数据图表。选定监测点位和信号类别后等待信号稳定，点击右上角“开始采集”按钮即可采集到接下来3秒内的数据并自动保存到数据库。

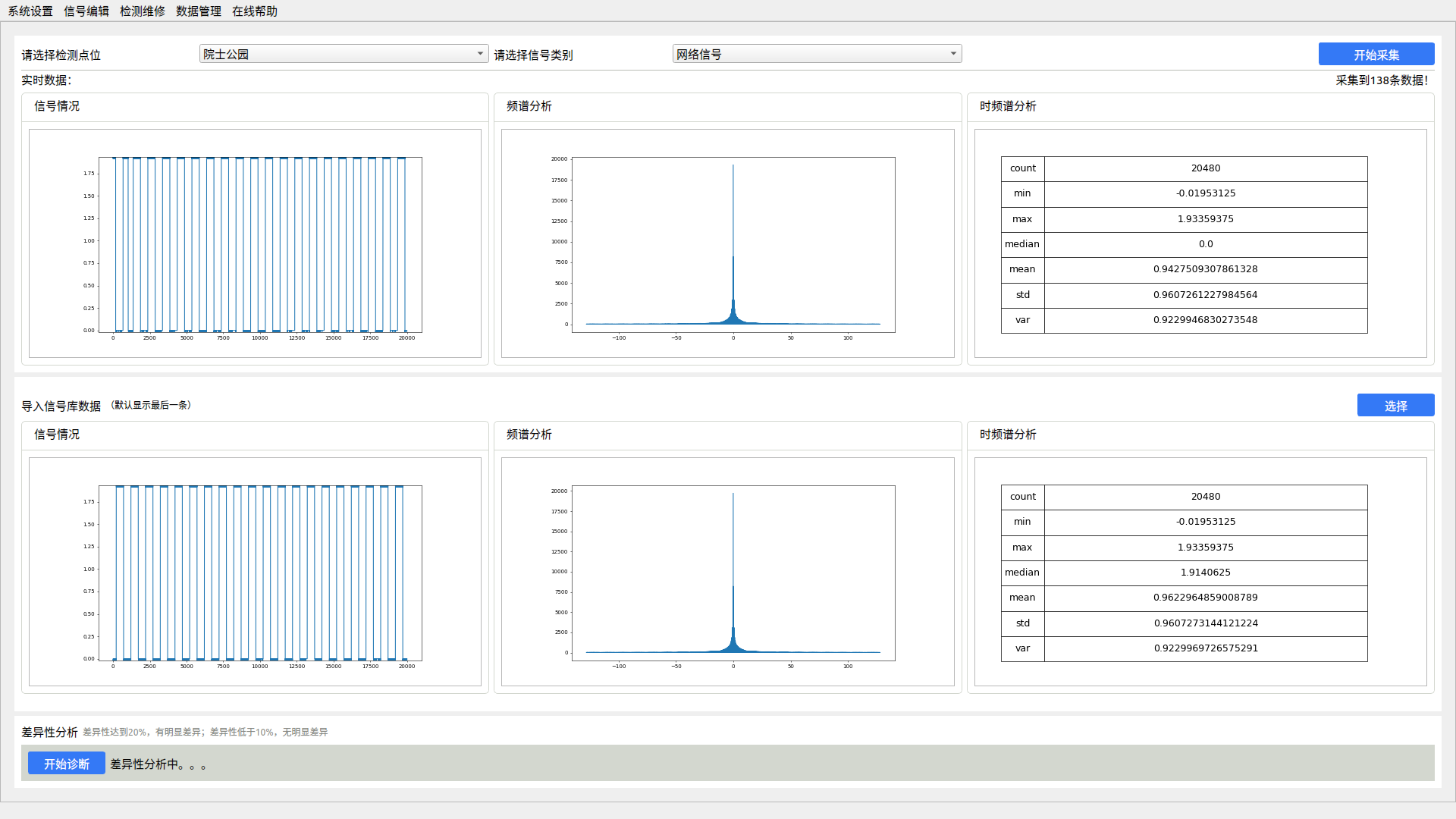


图 3.4 数据采集界面

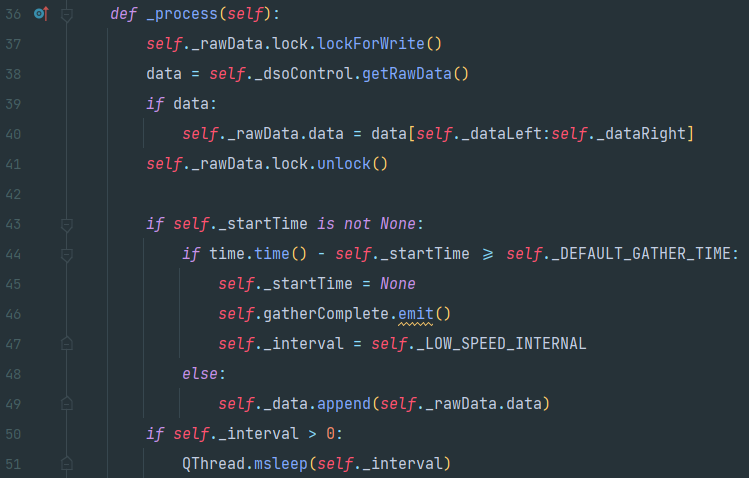


图 3.5 数据采集线程核心代码

如图3.5所示，当点击“开始采集”按钮后，\_startTime变量将被设置为当前时间戳，并将3秒内的数据持续添加到列表中，3秒过后\_startTime重新被设置为None，并发送采集完成信号，此时可以调用getData()方法获取这段时间采集到的数据。



图 3.6 主线程接收到信号后执行持久化操作

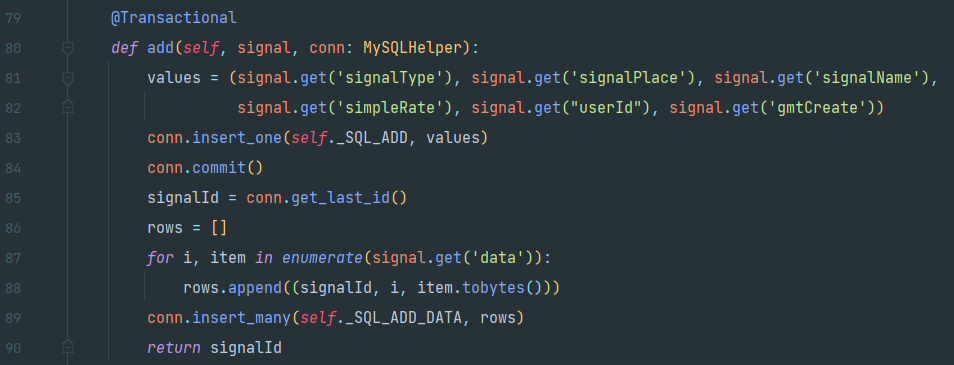


图 3.7 DAO层代码

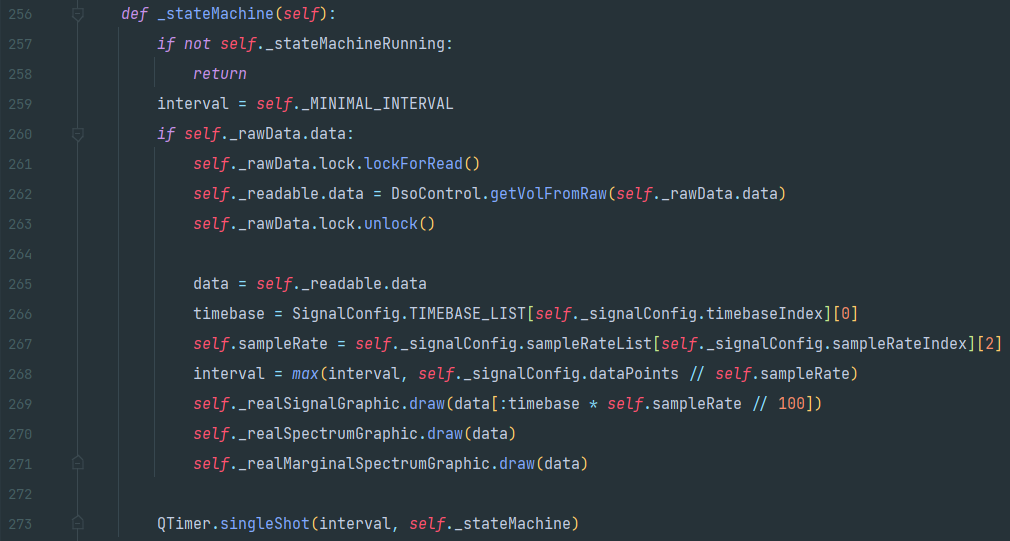


图 3.8 数据展示线程核心代码

如图3.8是数据展示线程的核心逻辑，其中self.\_rawData.lock就是原始数据实体中的读写锁，后面几个Graphic结尾变量调用的draw方法就是图像渲染的方法。至于中间对数据复杂的处理以及其中的硬编码是由于这部分代码依旧是试验性代码，以验证程序设计思想为主要目的。

## 图像展示和继承复用

图像展示算法运用了经典的FFT算法绘制了数据的频谱图。后文中的数据差异性对比则使用了小波变换提取数据特征值进行相关性分析计算[17]。



图 3.9 图像绘制组件

如图3.9，图像绘制组件GraphicWidget实例化了一个Qt的图片组件并传入了依赖的抽象画布，代码复用的关键点就在于这个抽象画布AbstractCanvas。draw方法作为暴露的函数供其他组件使用，而调用者往往不是主线程，子线程直接更新GUI会导致同步问题并引发Qt报警，因此采用了pyqtSignal信号这种曲线救国的方式，让抽象画布的doDraw方法渲染图片。

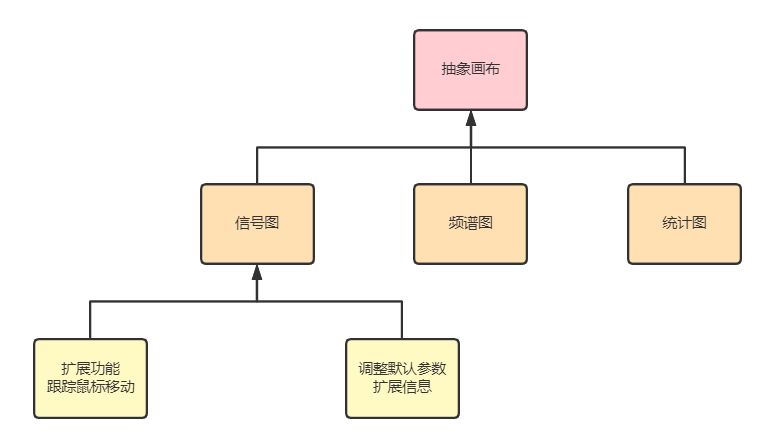


图 3.10 画布继承关系

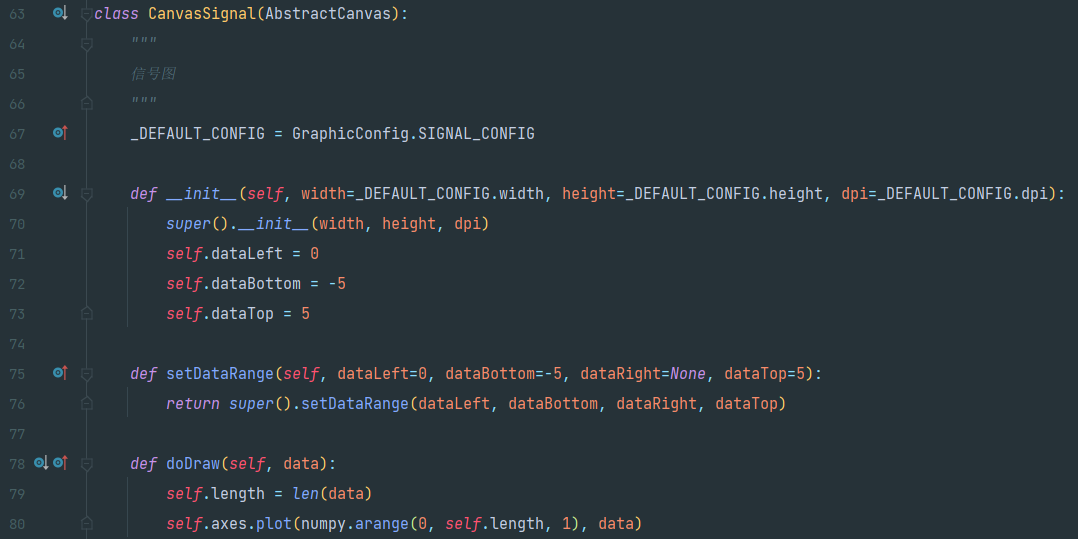


图 3.11 信号图代码

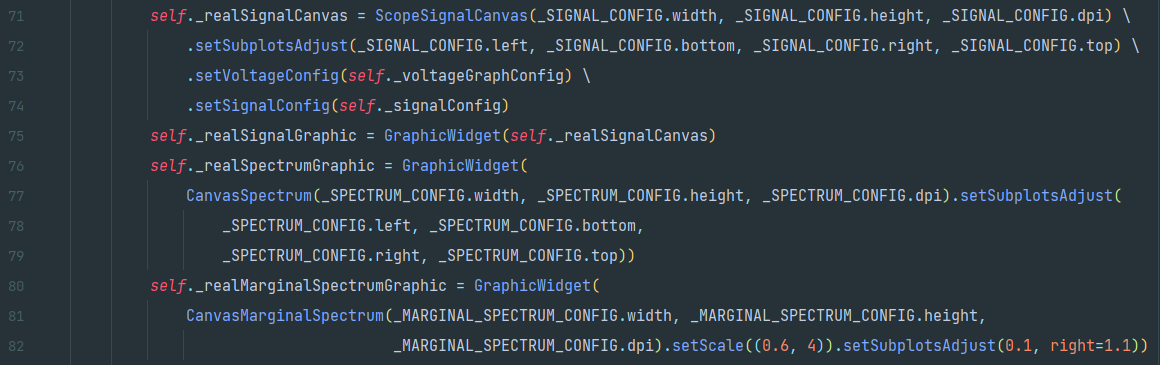


图 3.12 构造器中的图表组件实例化

从这些代码中可以看出来，抽象出绘图的行为，把具体的绘图实现作为参数，这样业务就只须与GraphicWidget交互，而不用关心它所依赖的具体是抽象画布的哪一个实例。

# 信号管理系统

## 分页表格

业务抽象出的一个实体数据结构往往对应数据库的一条记录，用户对基础数据的管理功能往往体现为数据表记录的增删改查。这种增删改查的功能实现往往需要使用复杂的表格和按钮以及频繁出现的样板代码和重复代码。数据量增大时，一次性查询所有数据是不必要也损耗性能的。目前常见的互联网应用会使用分页表格技术来解决上述问题。

分页表格通常主要由分页器、普通表格和精巧的封装代码组成。就代码实现角度来说，分页表格实现的主要难点在于分页器的实现。看似简单的分页器想要发挥良好的表现需要严谨地处理代码细节。由于Qt只提供了表格展示的组件，而笔者在官方和互联网上都没有找到分页器和分页表格的Qt实现，无奈之下只好自己简易实现一份。

分页表格的原理根据用户访问的页码、页大小对数据库查询结果进行筛选，并将数据库中符合查询条件的条数返回给分页器用于生成页码按钮供用户点击。按钮的点击和按钮的生成是一个双向反馈的过程，代码处理稍有不慎就可能出现意料之外的错误甚至进入死循环。

页码和页大小是两个关键的通用参数，我将其封装在一个实体类中，考虑到后续功能扩展的可能额外加入了排序条件和查询条件的参数。



图 4.1 PageDomain实体类

本节便是笔者基于pyqt实现的分页表格组件TableWidget的相关介绍，Windows系统上该组件运行效果如下：

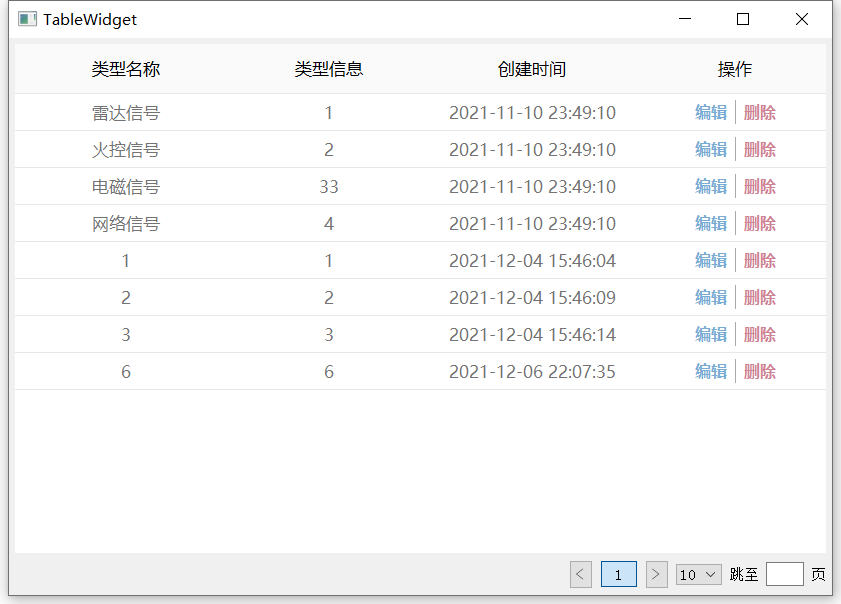


图 4.2 Windows下TableWidget运行效果

### 分页器

分页器的具体功能包括许多细节，这里仅挑选几个具有代表性的细节进行说明。

分页器的主要功能是供用户点击选择页码，而它不仅仅是供用户点击，如果数据量较大，它还需隐藏多余按钮，保持按钮数量以及可选按钮表示的页码在一个合理范围。假设数据总量是156条，分页器一页展示10条数据，那么可供用户选择的有1~16共16页，一次性展示16个按钮也许还能够接收，可如果是1560条，甚至15600条，那么随着用户崩溃的同时，程序崩溃的可能性也非常大。

同样以上述数据为例，当用户处于中间位置时，应当显示相邻的2个页码以及首尾页码，以便用户随时从前往后、从后往前查看数据。如下图：

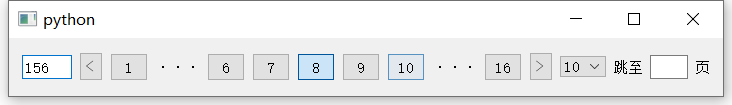


图 4.3 分页器在第8页时的行为

当用户处在第1页时一方面需要适当地隐藏离1较远的页码，如10、11、12页；另一方面，如果仅展示第1页右边2个页码即显示到页码3，用户的选择受到了一定限制。更好的做法是当总页数充足的情况下，显示分页器当前页码相邻的至少4个页码即显示到页码5，如下图。



图 4.4 分页器在第1页时的行为

省略号出现的时刻也是值得关注的细节，不恰当的UI会使用户感到困惑。如第4页时左边不应出现省略号，左右省略号出现的时机需要临界值。

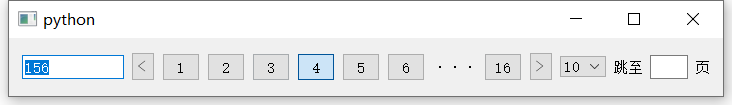


图 4.5 分页器在第4页的行为

如果数据总数比较少，过早出现省略号也会引起用户不满，设置一个阈值用来检测分页器是否需要省略号会相对比较合理。



图 4.6 总页数不超过8页时不显示省略号

当分页器允许用户自由选择页码时，参数的检测尤为重要，但为了配合用户，往往会采取用户想要的结果。如用户填了一个超出当前页数的页码，分页器应该选择没有响应，还是选择跳转到最后一页呢。考虑到查询条件改变的情况下，当前页数往往未知，用户输入偏大的页码时前往更靠近这一数字的最大页码往往是用户想要的结果。



图 4.7 用户跳至第100页的行为

说了这么多场景，不妨来看看代码是如何实现的。虽然我的代码尚不成熟，但也可以从这费尽心思的只言片语中窥见广大底层组件提供者的良苦用心。



图 4.8 组件中的静态变量

如上图所示，changed用于在恰当的时机发出消息，常量8和2就是我在上文提到的两个默认参数，这个参数可以随时更改以适应需求变动，直接在代码中硬编码则会使这种改动的成本提高许多，这只是代码编写中的一小部分。

如图4.9所示，关键的函数不仅要细心地处理每一处细节，更是要详尽地提供代码注释使得无论重新审视这段代码时都能重拾作者的思路。



图 4.9 更新组件列表的关键函数

对许多应用来说，单纯实现功能并不难，如何将应用做的尽善尽美是需要花大功夫的。上述场景只是举了其中一部分细节，当分页器融入分页表格时将会产生更加复杂的化学反应，如何适当地提供组件的API也是十分考验组件设计者的功力。

### 表格封装

封装好分页器后，分页表格的工作相对简单，只需在内部封装一个表格和分页器，在接收到分页器信号时获取数据即可。这里的获取数据指的是调用依赖方传入的具有规定格式的函数用于获取数据和符合条件的数据总数。对于每一项数据的处理策略，我们可以再简单定义一个实体类允许调用者有特殊需求时自定义字段处理逻辑。



图 4.10 表示表格字段处理方式的实体类

如下图，TableWidget的关键部分在于调用者传入的两个参数，一个作为页面发生改变时的回调，一个作为获取到数据后对每个字段进行处理时执行的回调，因此，其构造器是代码设计中的一个重要部分。

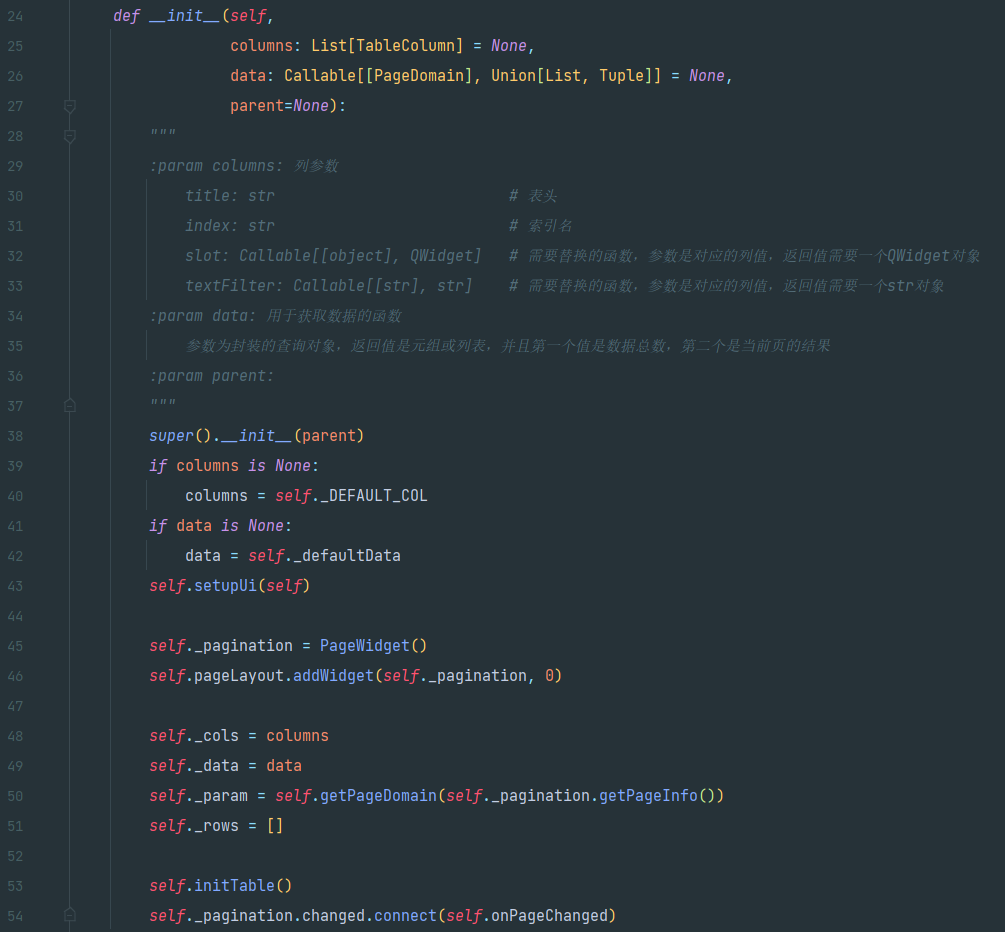


图 4.11 TableWidget构造器

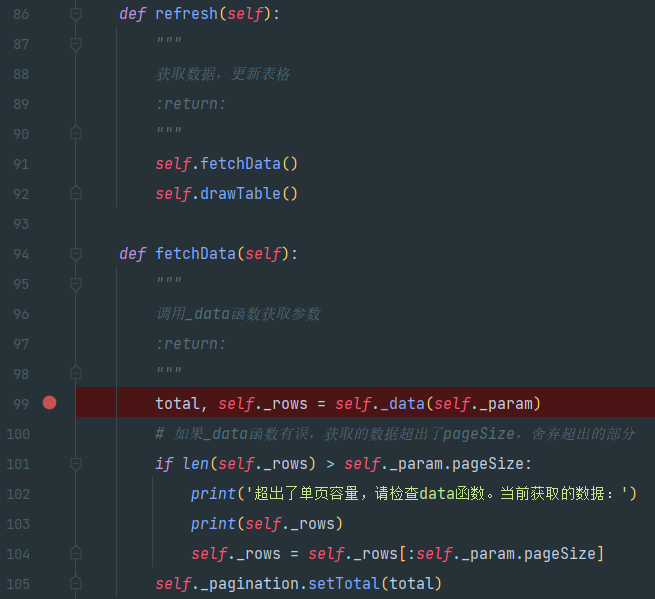


图 4.12 数据发生改变时执行调用者传入的回调

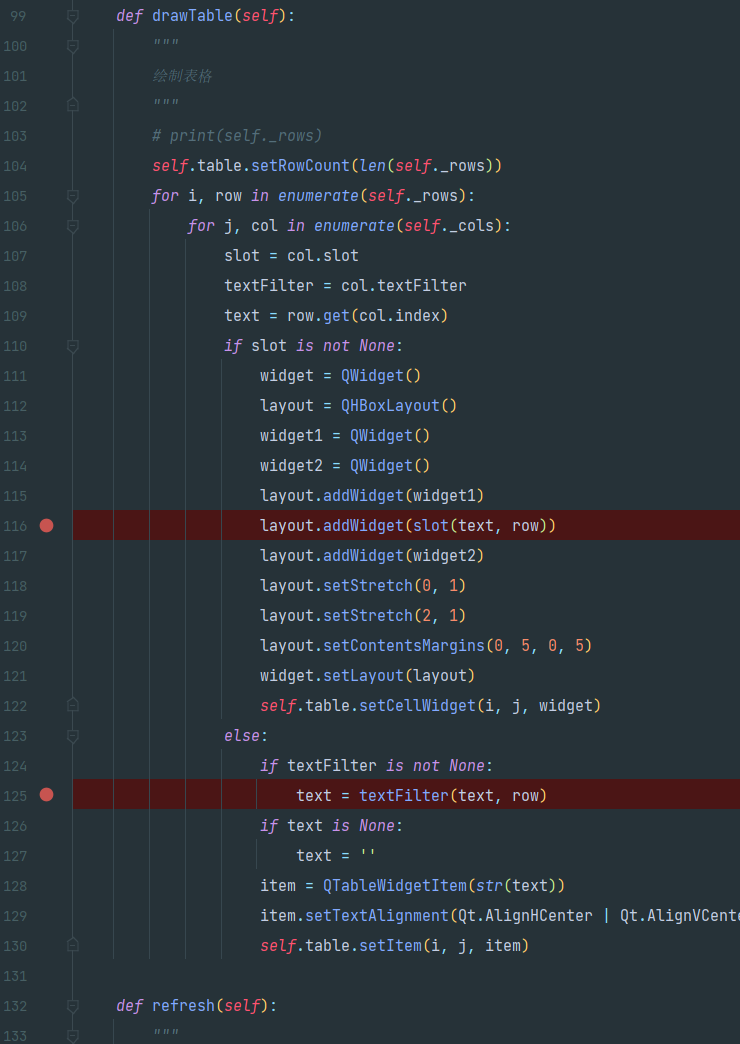


图 4.13 获取参数后对数据的每个字段施以调用者的处理策略

源码看起来终究枯燥，不妨看一个调用方的简单例子。下图正是图4.2所演示效果的源码。其中，typeFilter用于将查得的typeId转换为对应的typeName，actionButtons函数用于返回一个组件列表代替列值，操作列的字段处理方式是返回两个没有边框的按钮，并能从入参中获取当前行数据以便完成数据更新等操作。list4Table函数能够查询到符合查询条件的数据和总数。



图 4.14 调用方的例子

分页表格的封装到这里就结束了，然而纵使笔者抱着谨慎小心的态度希望避开每一个漏洞，漏洞仍然一如既往地不可避免。笔者在项目后期发现这个组件在某些情况下会造成不良的用户体验，而开发阶段中由于数据量较小一直没有发现，方才感叹自身能力经验尚浅，还需多多历练。

## 基础信息管理

提供基本业务对象的管理功能。从代码层面上说就是使用TableWidget组件对基础信息进行增删改查操作。项目开发期间，由于时间精力有限，数据库查询得到的字典型的结果几乎直接被程序全盘使用，造成项目中大量通过键值获取字段的硬编码现象，如果出现同名字段，那么当某个字段发生变动时将会花费巨大的代价修改代码。一种解决方案是引入ORM映射框架将数据库查询结果转换成实体对象。笔者在项目结束后饶有兴致地探索了python在这方面的实现方式，并最终简单实现了基于装饰器语法的ORM框架，详见附录A。

### 检测点位

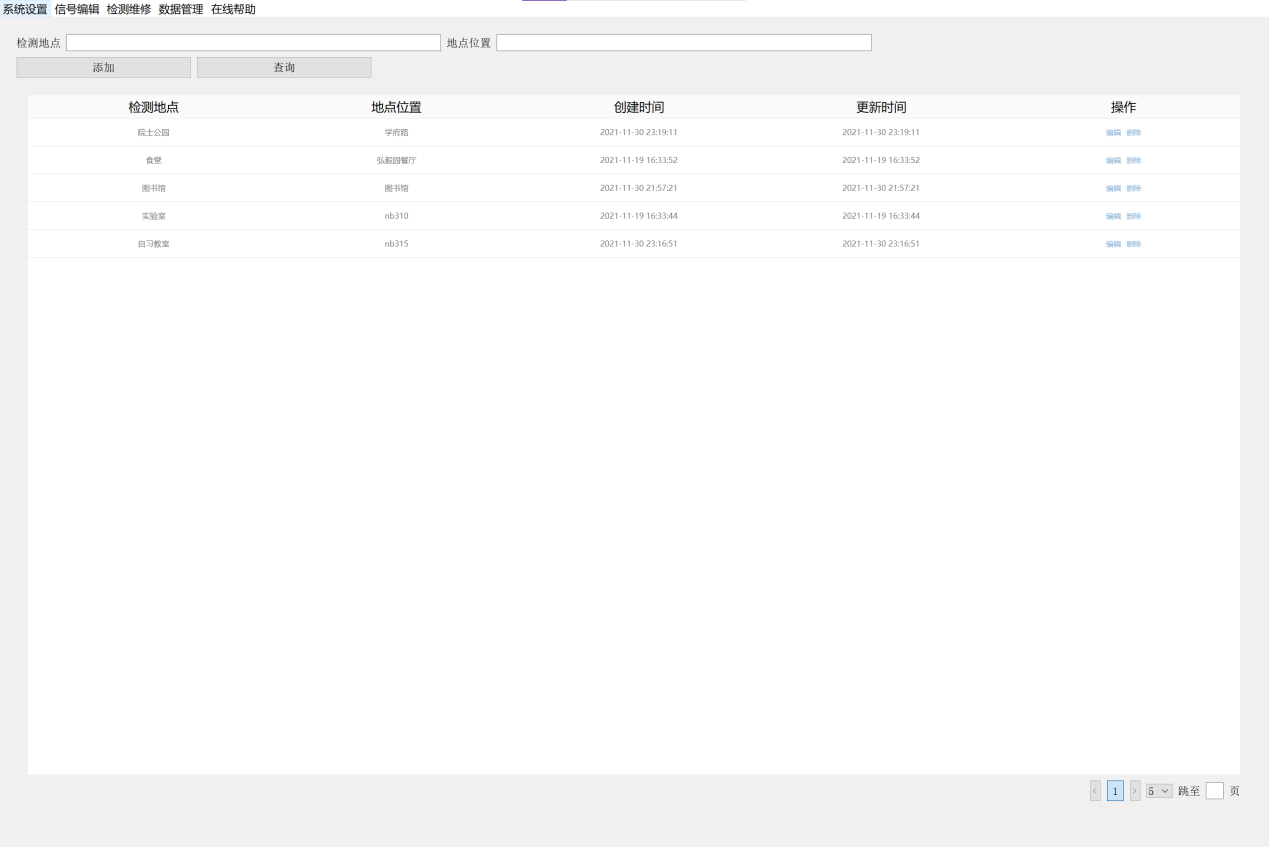


图 4.15 检测点位界面

### 信号类别

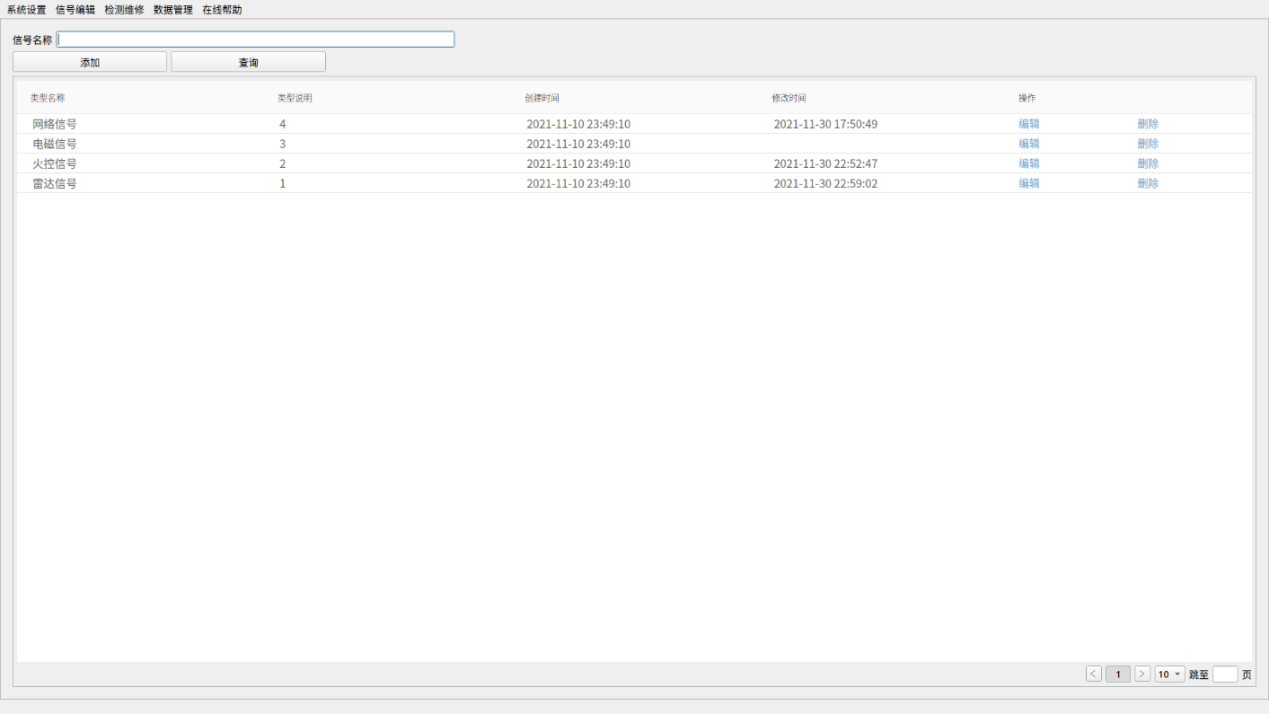


图 4.16 信号类别界面

### 用户管理

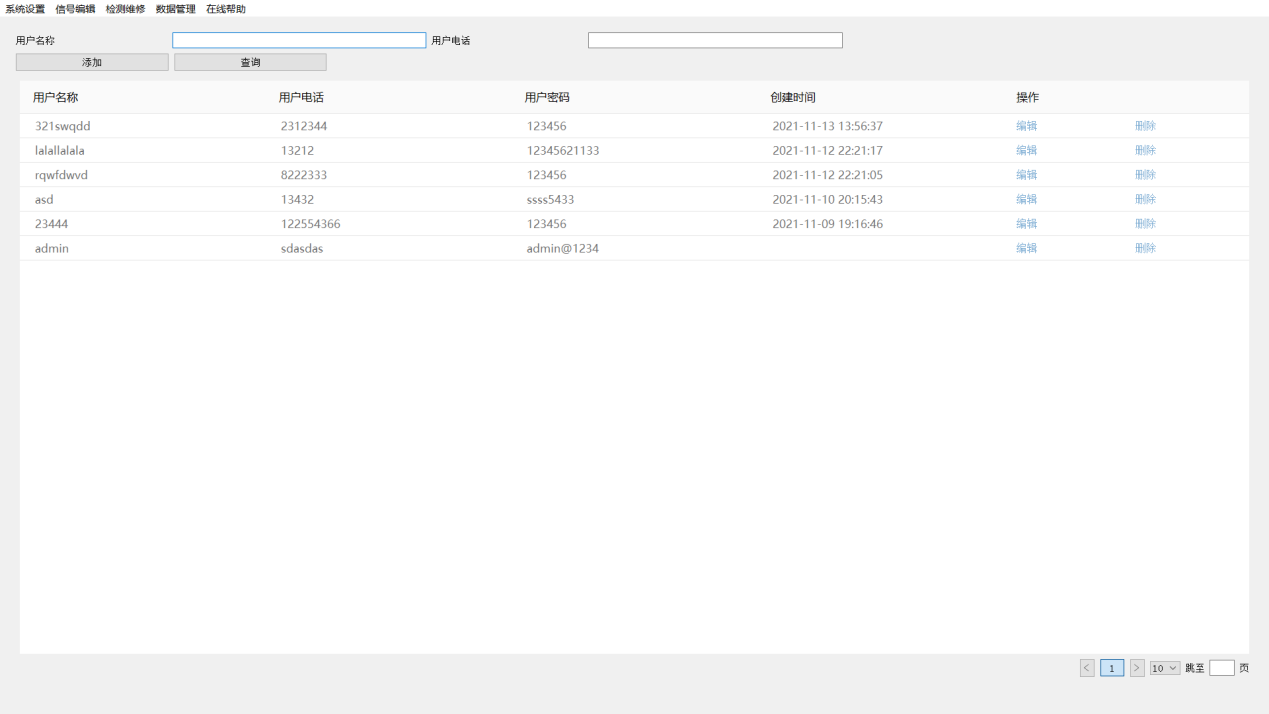


图 4.17 用户管理界面

## 信号相关组件

信息管理系统对信号数据的管理与基础数据管理模块差异较大。信号数据处理是本项目的核心业务，就自然需要进行图像处理算法应用、数据差异性分析等额外处理。

### 历史数据

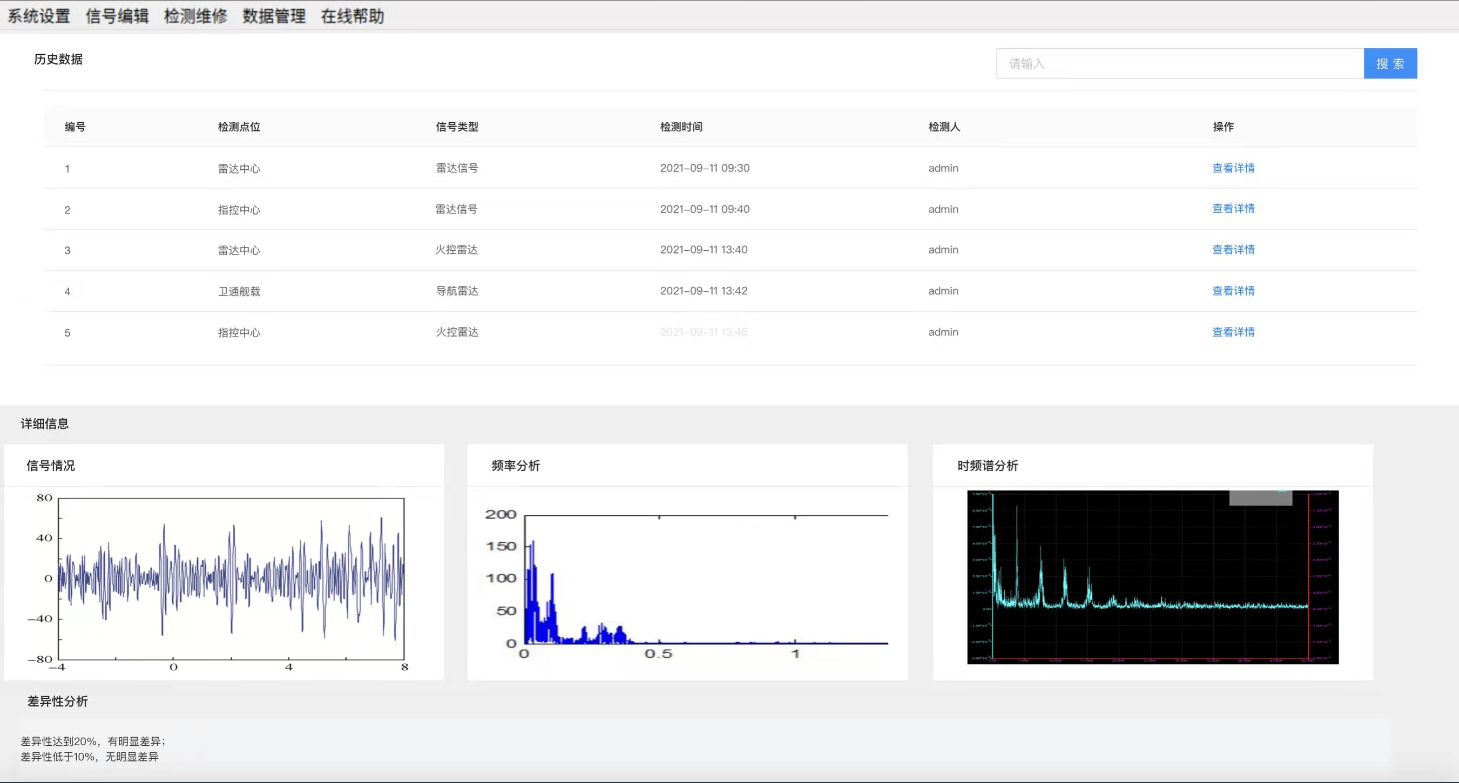


图 4.18 历史数据原型图

上图是为数不多的产品设计图中唯一被保留下来的一张，作为见证本次课题从无到有的颇具意义的纪念张贴于此。

历史数据界面能够查看所有采集到的数据的具体信息，包括其数据波形、差异性分析结果、维修日志等。历史数据仅支持对维修日志的修改。点击导入按钮即可将该记录导入到标准信号库中。

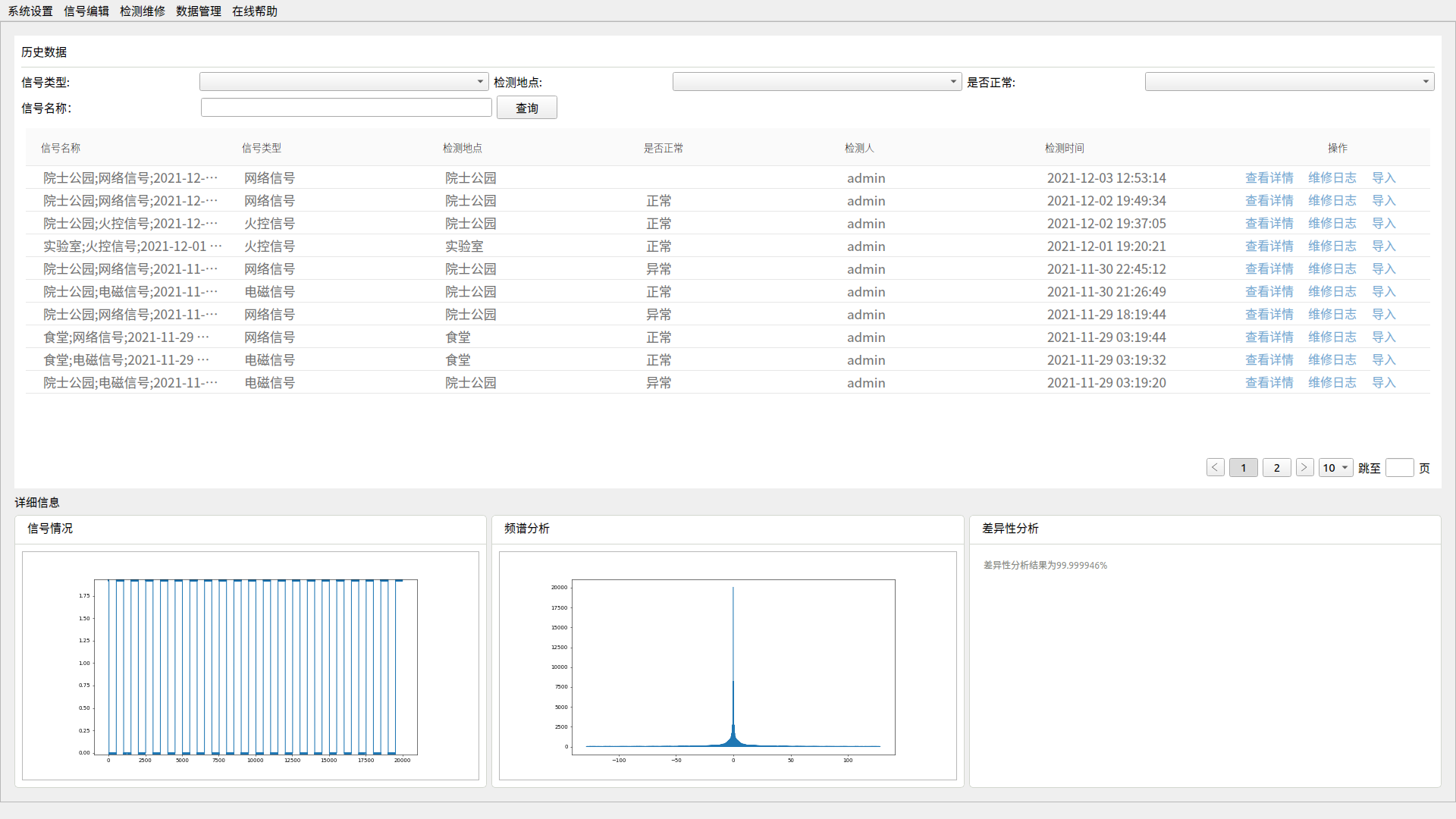


图 4.19 历史数据界面

### 诊断维修

采集到的信号数据入库后，可以在此页面将数据与信号库中的标准数据进行差异性分析比较。点击开始诊断按钮后，算法将计算两组数据相似程度，相似度超过标准线将被标记为标准信号；相似度低于异常线将被标记为异常信号。

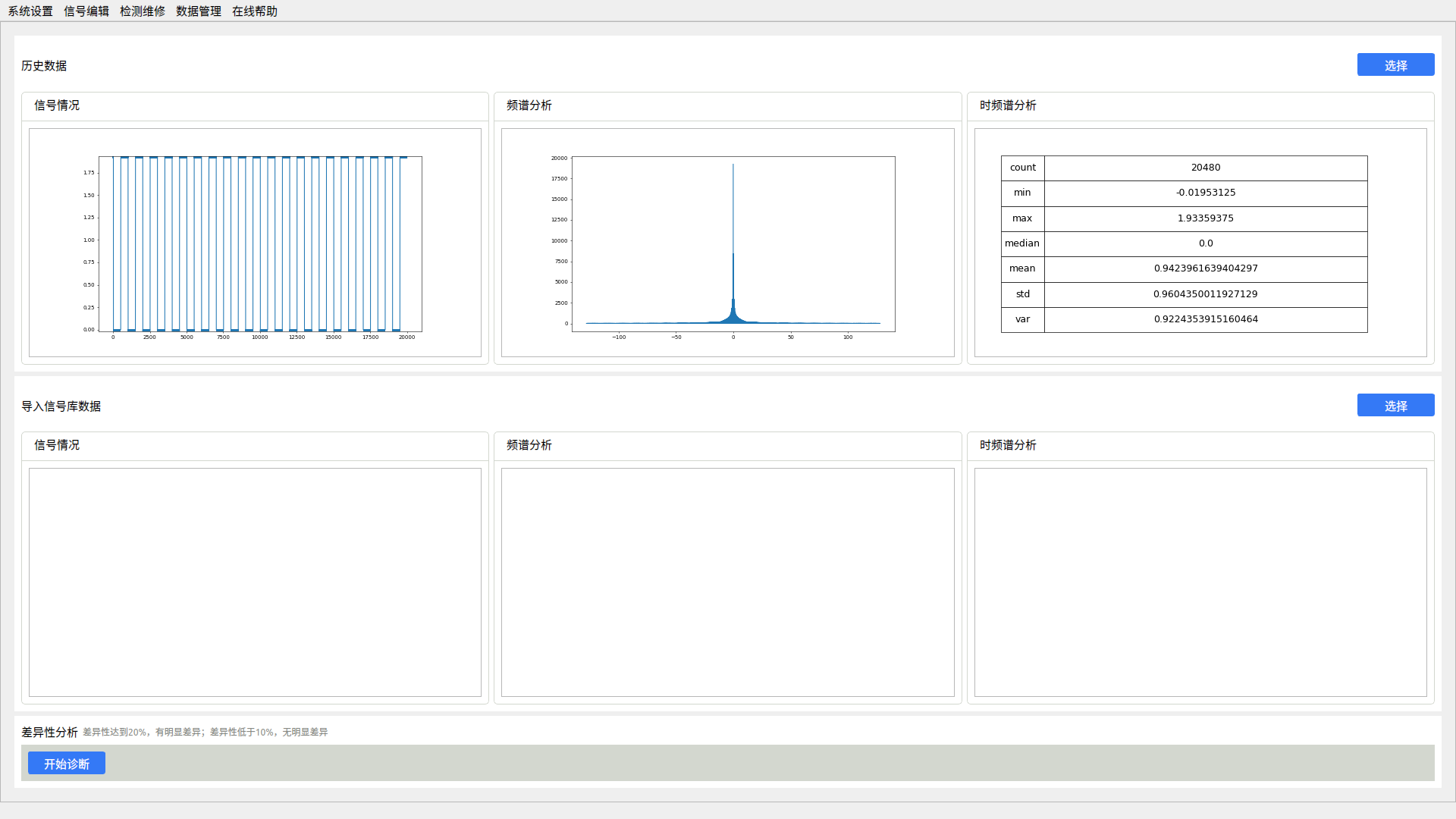


图 4.20 诊断维修界面

### 标准信号库/异常信号库

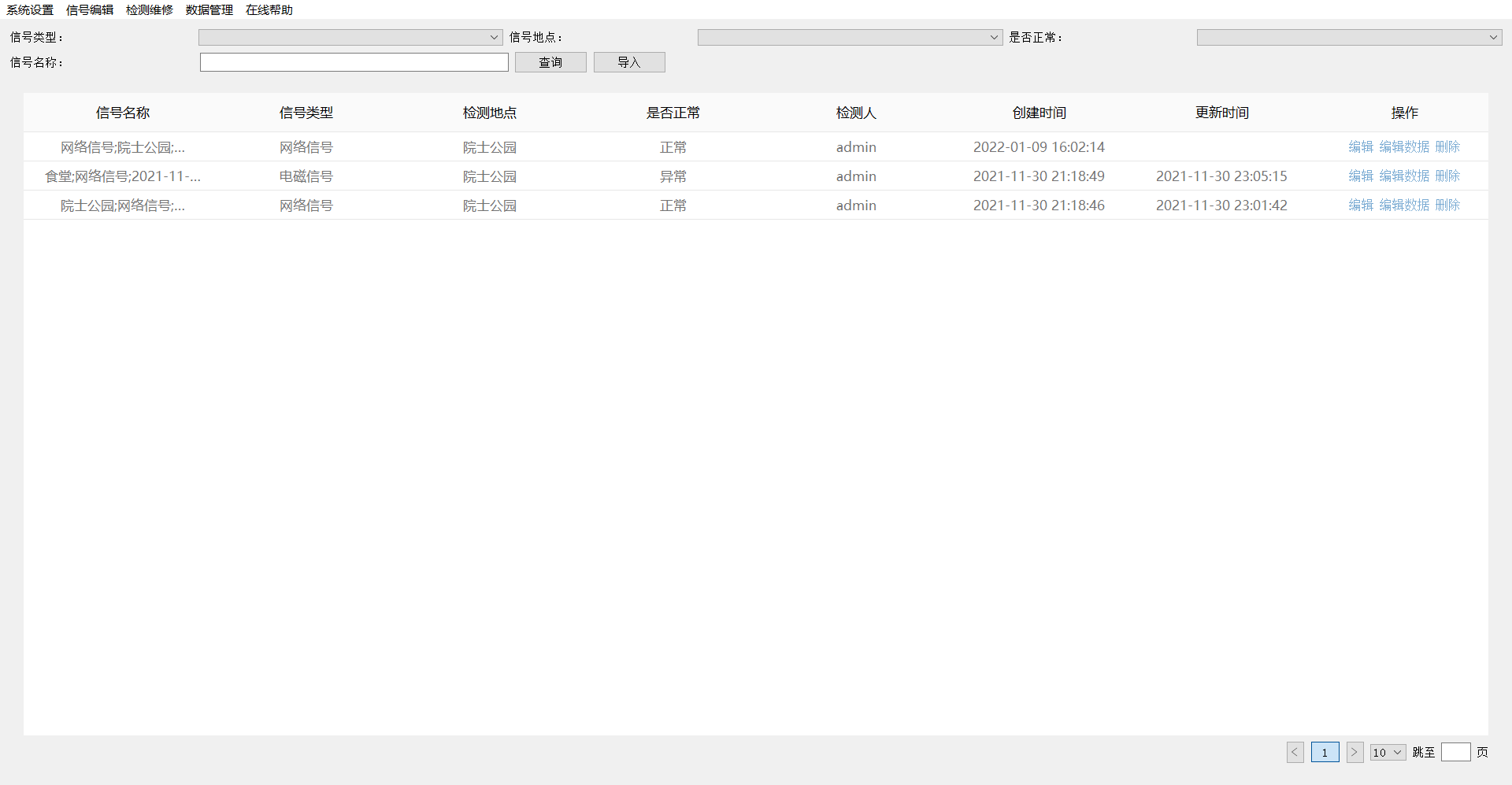


图 4.21 标准信号库界面

包含从历史数据中导入数据的功能。

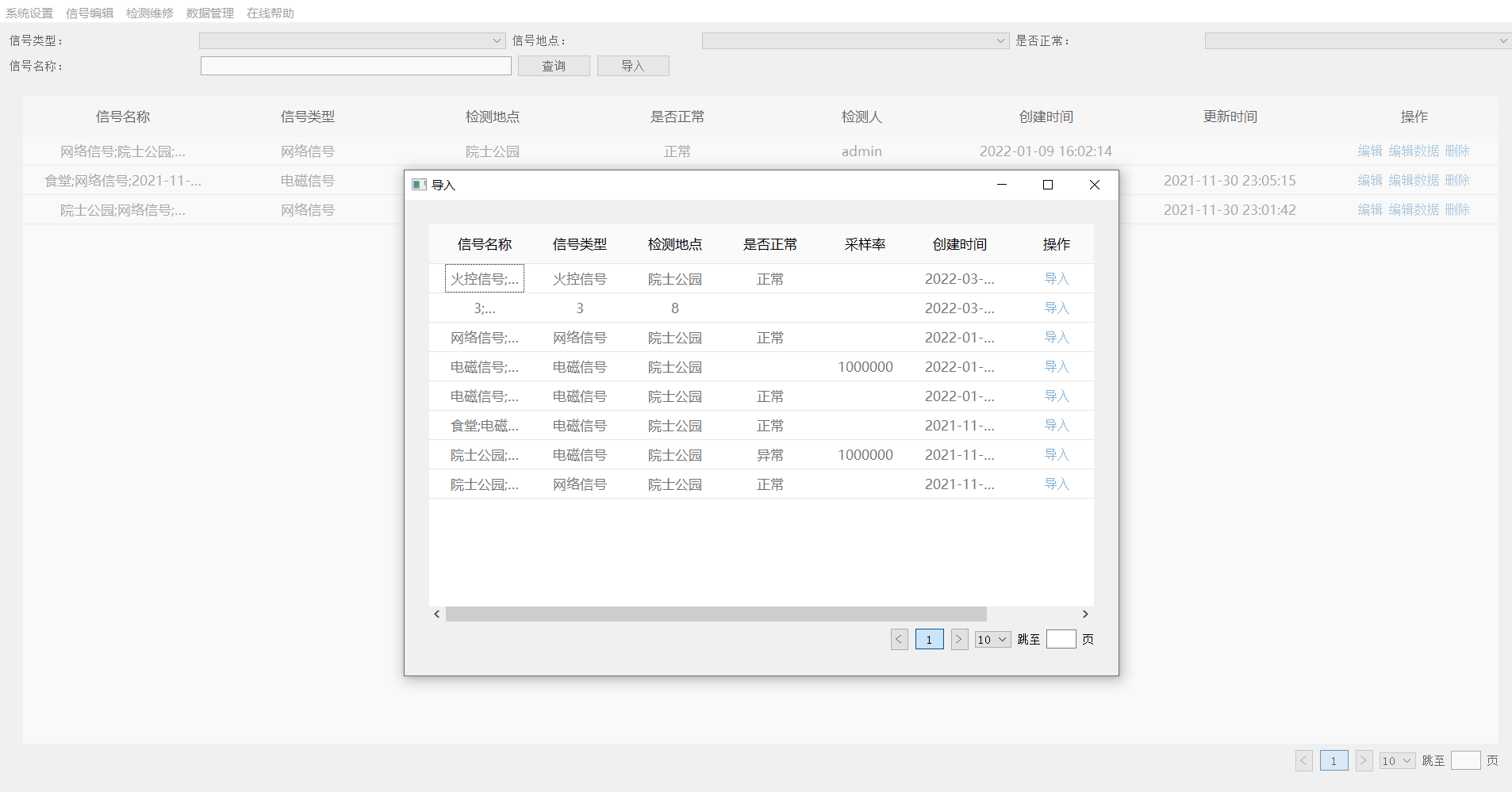


图 4.22 从历史数据导入信号库

特别包含数据编辑功能，能够对信号库数据进行微调。组件下方的拖动条是数据的一维地址，左上的地址选择框是二维地址，通过设置数值可以对信号进行微调并保存。复原按钮可以恢复到数据上一次保存的状态。

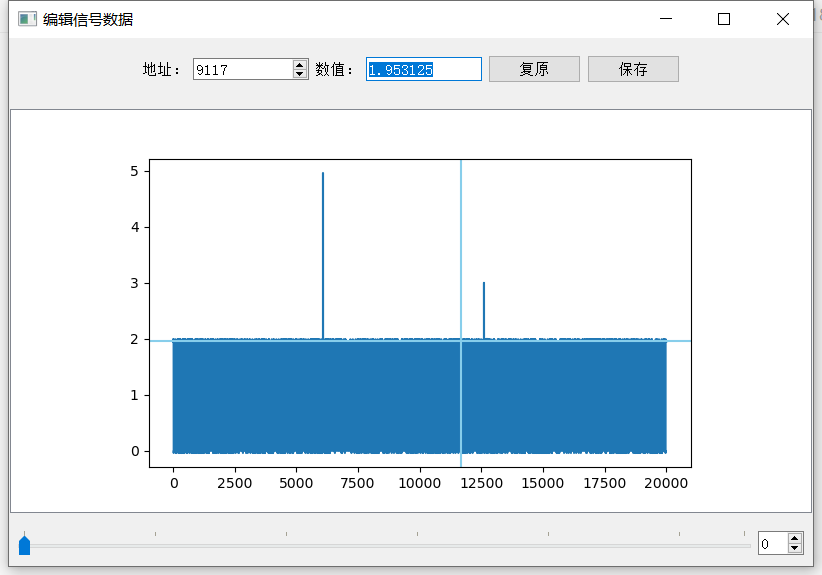


图 4.23 编辑数据界面

该功能的实现使用数字输入框，封装了双向绑定的滑动条和数字输入框，使用了备忘录模式实现了该功能。

# 结论和展望

本课题结合理论和实践阐述了虚拟示波器设备有别于传统示波器的独特优势，证明了虚拟示波器与物联网技术结合的可能性。因此笔者有充分理由相信虚拟示波器能够作为未来主流示波器应用于各种场景下。

回想虚拟示波器的历史，最初的虚拟示波器大多由美国国家仪器（NI）公司的Lab VIEW软件开发，具有测试测量、控制、仿真等强大功能并具有快速开发、跨平台等优势。然而，在物联网、大数据技术不断改变我们生活的今天，执着于硬件的研究会陷入边际效应的困境，对于虚拟示波器的进一步发展可以从软件层面探索，投入到物联网络当中来。我们不妨畅想一下，在数字化高度发达的未来社会中，5G、6G网路下，数据传输的实时性得到充分保障；成熟的云计算技术使得软件计算速度大大超过硬件；得益于网络和云计算，人工智能提出了各种信号分析模型能快速应用到新采集的信号数据并对异常情况进行迅速响应，而未来技术实现电路自动校正修复技术也不无可能，人们可以从手机等各种智能终端随时远程监控各节点状态，这将大大降低人工检测、维护的门槛和成本。我们也可以将本课题所设计的系统改造升级一下，不再是单机系统，而是将PC替换成无需界面的微型计算机，甚至直接改造虚拟示波器，使其成为具备更多功能的下位机，在物联网中心部署计算集群处理并控制各节点设备，一套大规模电路自动检测修复系统得以建立。

如果有人问这套系统如此费尽周章将要应用于什么重要设施呢？其实物联网本身便是最好的应用场景。物联网中的大量设施暴露在各种物理环境下，其维护检测的工作量可想而知。倘若有一套能够自检的物联网系统，也许其维护成本可以降低许多。

纵向来看，我认为虚拟示波器的广泛应用还面临较大阻力，一方面我国虽然在物联网、5G、云计算领域处于领先，但在示波器上还一直是短板，尤其是虚拟示波器，国内相关研究凤毛麟角，技术上还有很长的路要走；另一方面虚拟示波器缺乏标准化，各个厂家生产设备的协议不同，指令不同，行为方式也有差异，尽早完成设备标准化生产可能有助于加速示波器行业的发展。

# 参考文献

[1] Ping G, Wei Z. Design and Implementation of Multifunctional Virtual Oscilloscope Using USB Data-Acquisition Card[J]. Procedia Engineering, 2012, 29:3245-3249.

[2] 吴彤, 张昌芳, 吴东坡,等. 军队信息化建设的几个基本理论问题[J]. 国防科技, 2010, 031(003):14-19.

[3] 舒毅, 李栋, 任雍,等. 数字示波器在雷达维护维修中的应用[J]. 气象水文海洋仪器, 2012, 29(2):4.

[4] 郭恩全, 赵兴奋. 虚拟仪器发展趋势及其对军用测试技术的影响[J]. 国外电子测量技术, 1999(6):3.

[5] 陈尚松, 李智, 雷加,等. 虚拟仪器回顾与展望[J]. 国外电子测量技术, 2009, 28(12):17-21.

[6] Ko C C, Chen B M, Chen S H, et al. A large-scale Web-based virtual oscilloscope laboratory experiment[J]. Iee Engineering Science & Education Journal, 2000, 9(2):69-76.

[7] 郭琢. 基于Windows平台虚拟示波器的设计与实现[D]. 吉林大学, 2020.

[8] 杜超,林开伟,陈曼雯,高浩.物联网智能示波器研究与设计[J].物联网技术,2017,7(11):36-39+41.DOI:10.16667/j.issn.2095-1302.2017.11.008.

[9] Maier A , Sharp A , Vagapov Y . Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things[C]// 7th International Conference on Internet Technologies and Applications. 2017.

[10] Zhang S . INTERNET OF THINGS-BASED NOVEL OSCILLOSCOPE:, WO2017152326A1[P].

[11] Wang S J , Cheng Z Q , Han X . Design of Virtual Oscilloscope Soft Panel Based on Qt[J]. Computer Technology and Development, 2013.

[12] 张明照, 刘政波, 刘斌. 应用MATLAB实现信号分析和处理[M]. 科学出版社, 2006.

[13] Mahafza, B.R. (2010). Radar Signal Analysis and Processing Using MATLAB (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781420066449

[14] Zhang X H , CAEP. Research of Impedance Matching of Oscilloscope and Probe Selection[J]. Nuclear Electronics & Detection Technology, 2013.

[15] Robinson K B , Christopherson M , Kendall T . Interface for flash EEPROM memory arrays[J]. 2001.

[16] Kroah-Hartman, Greg. udev--Persistent Device Naming in User Space.[J]. Linux Journal, 2004.

[17] Daubechies, I. The wavelet transform, time-frequency localization and signal analysis[J]. IEEE Transactions on Information Theory, 1990, 36(5):961-1005.

附录A python ORM实现

本项目中对实体数据的处理不太妥当，直接使用字典的实现方式与数据库字段名形成了强依赖，如果字段名修改，那么代码层面上的修改代价会非常高。使用ORM（Object Relational Mapping，对象关系映射）框架将数据库查询结果解析为实体类对象能够解决这个问题。

ORM是互联网应用常用技术，python并不是为web诞生的语言，没有主流的ORM框架，但是python语言极其灵活，能否使用python实现一个ORM框架并且应用到项目中这个问题激发起了我的兴趣。

查阅一番资料之后，我借鉴了java中流行的MyBatis框架，仿照其注解+动态代理的实现手段，基于python的装饰器语法成功实现了几个简单功能。

*class* Proxy:  
 *def \_\_init\_\_*(*self*, clz, pclz):  
 *self*.clz = clz  
 *self*.pclz = pclz  
 *self*.proxies = {} *# 查表提高效率  
  
 def \_\_call\_\_*(*self*, \*args, \*\*kws):  
 *self*.obj = *self*.clz(\*args, \*\*kws) *# 调用被代理类的构造函数创建实例  
 return self  
  
 def \_\_getattr\_\_*(*self*, attr): *# 调用方法之前需要通过 getattr 查找方法* ret = *None  
 if hasattr*(*self*.obj, attr): *# 查看被代理实例成员是否存在* ret = *getattr*(*self*.obj, attr)  
 *if isinstance*(ret, types.MethodType): *# 如果该成员是方法  
 if* ret *not in self*.proxies: *# 如果该方法的代理没有被生成  
 self*.proxies[ret] = *self*.pclz(*self*.obj, ret) *# 创建该方法的代理  
 return self*.proxies[ret]  
 *return* ret  
  
  
*class* ProxyFactory:  
 *def \_\_init\_\_*(*self*, pclz):  
 *self*.pclz = pclz  
  
 *def \_\_call\_\_*(*self*, clz):  
 *return* Proxy(clz, *self*.pclz)

以上代码段中的两个类分别是代理类和代理工厂，代码中大量运用了python动态语言的特性，将一切皆对象这个理念发挥得淋漓尽致。

*def* SQLWrapper(sql, sqlType, returnType):  
 *def* decorate(func):  
 *setattr*(func, 'sqlType', sqlType)  
 *setattr*(func, 'sql', sql)  
 *setattr*(func, 'returnType', returnType)  
 *return* func  
 *return* decorate  
  
  
*def* Select(sql, returnType):  
 *return* SQLWrapper(sql, Select, returnType)  
  
  
*def* Insert(sql):  
 *return* SQLWrapper(sql, Insert, *int*)  
  
  
*def* Update(sql):  
 *return* SQLWrapper(sql, Update, *int*)  
  
  
*def* Delete(sql):  
 *return* SQLWrapper(sql, Delete, *int*)  
  
  
*class* OrmProxy:  
 LOCAL\_PAGE = threading.local()  
 FROM\_RE = re.compile('from (.\*)', re.M | re.I | re.S)  
 FIRST\_WORD\_RE = re.compile('\w\*')  
  
 *def \_\_init\_\_*(*self*, obj, method):  
 *self*.obj = obj  
 *self*.method = method  
 *self*.pageSize = *None  
 self*.pageNum = *None  
 self*.orderBy = *None  
 self*.sql = *getattr*(*self*.method, 'sql')  
 *self*.sqlType = *getattr*(*self*.method, 'sqlType')  
 *self*.returnType = *getattr*(*self*.method, 'returnType')  
 *self*.argsGenerator = *None  
 self*.sqlGenerator = *None  
 self*.preparedSql = *None  
 self*.replaceSqlArgs()  
  
 @staticmethod  
 *def* underscoreToCamelCase(s: *str*):  
 words = s.split('\_')  
 res = words[0]  
 *for* word *in* words[1:]:  
 res += word.title()  
 *return* res  
  
 @staticmethod  
 *def* camelCaseToUnderscore(s: *str*):  
 res = ''  
 *for* i *in* s:  
 res += '\_' + i.lower() *if* i.isupper() *else* i  
 *return* res  
  
 @classmethod  
 *def* getCamelCaseMap(*cls*, returnType, mapUnderscoreToCamelCase=*True*):  
 res = {}  
 *for* key *in* returnType().*\_\_dict\_\_*:  
 underscore = *cls*.camelCaseToUnderscore(key)  
 res[underscore] = key *if* mapUnderscoreToCamelCase *else* underscore  
 *return* res  
  
 @staticmethod  
 *def* getEntityFromDict(returnType, row: *dict*, keyMap: *dict*):  
 *# res = returnType()  
 # for key, value in row.items():  
 # key = keyMap[key]  
 # if key:  
 # setattr(res, key, value)  
 # return res* kwargs = {}  
 *for* key, value *in* row.items():  
 key = keyMap.get(key, key)  
 kwargs[key] = value  
 *return* returnType(\*\*kwargs)  
  
 @classmethod  
 *def* getDynamicParam(*cls*, expression, kwargs):  
 key = *cls*.FIRST\_WORD\_RE.match(expression).group()  
 value = *eval*("kwargs['{}']{}".format(key, expression[*len*(key):]))  
 *return* value  
  
 @classmethod  
 *def* judgeDynamicExpression(*cls*, expression: *str*, kwargs):  
 *for* key, value *in* kwargs.items():  
 varExpr = re.findall('({}(?:\.\w+)?)'.format(key), expression)  
 *if* varExpr:  
 *for* expr *in* varExpr:  
 key = *cls*.FIRST\_WORD\_RE.match(expression).group()  
 expression = expression.replace(expr, "kwargs['{}']{}".format(key, expr[*len*(key):]))  
 *try*:  
 *return eval*(expression)  
 *except NameError as* e:  
 *return False  
  
 def* replaceSqlArgs(*self*):  
 ifLabels = re.findall('<if test="(.\*?)">(.\*?)</if>', *self*.sql, re.M)  
 *if* ifLabels:  
 sqlPart = re.split('<if test=".\*?">.\*?</if>', *self*.sql, re.M)  
  
 *def* sqlGenerator(kwargs) -> *str*:  
 sql = sqlPart[0]  
 *for* i *in range*(*len*(ifLabels)):  
 judgeExpr, dynamicPart = ifLabels[i]  
 part0 = sqlPart[i + 1]  
 *if* self.judgeDynamicExpression(judgeExpr, kwargs):  
 sql += dynamicPart  
 sql += part0  
 *return* sql  
  
 *def* argsGenerator(sql, kwargs) -> *list*:  
 dph = re.findall('#{(.\*?)}', sql, re.M)  
 dsp = re.split('#{.\*?}', sql, re.M)  
 res = []  
 self.preparedSql = dsp[0]  
 *for* part1 *in* dsp[1:]:  
 self.preparedSql += '%s' + part1  
 *for* ph *in* dph:  
 res.append(self.getDynamicParam(ph, kwargs))  
 *return* res  
 *else*:  
 placeholders = re.findall('#{(.\*?)}', *self*.sql, re.M)  
 sqlPart = re.split('#{.\*?}', *self*.sql, re.M)  
 *self*.preparedSql = sqlPart[0]  
 *for* part *in* sqlPart[1:]:  
 *self*.preparedSql += '%s' + part  
  
 *def* sqlGenerator(kwargs) -> *str*:  
 *return* self.preparedSql  
  
 *def* argsGenerator(\_, kwargs) -> *list*:  
 res = []  
 *for* ph *in* placeholders:  
 res.append(self.getDynamicParam(ph, kwargs))  
 *return* res  
 *self*.sqlGenerator = sqlGenerator  
 *self*.argsGenerator = argsGenerator  
  
 *def* \_query(*self*, conn: MySQLHelper, sql: *str*, returnType, args):  
 total = *None* page = *getattr*(*self*.LOCAL\_PAGE, 'page') *if hasattr*(*self*.LOCAL\_PAGE, 'page') *else None  
 if* page:  
 *delattr*(*self*.LOCAL\_PAGE, 'page')  
 fromMatch = *self*.FROM\_RE.search(sql)  
 countSql = 'select count(\*) as cnt ' + fromMatch.group() *if* fromMatch *else* ''  
 total = conn.select\_one(countSql, args)[1]['cnt']  
 *if* page.orderBy:  
 sql += ' order by {} '.format(page.orderBy)  
 sql += ' limit %s, %s'  
 args.append(page.getStart())  
 args.append(page.pageSize)  
 sqlResult = conn.select\_many(sql, args)  
 *if isinstance*(returnType, *type*(List)) *and getattr*(returnType, '\_name') == 'List':  
 t = *getattr*(returnType, '\_\_args\_\_')[0]  
 res = PageResult(total) *if* total *else* []  
 keyMap = *self*.getCamelCaseMap(t)  
 *for* sqlRow *in* sqlResult[1]:  
 res.append(OrmProxy.getEntityFromDict(t, sqlRow, keyMap))  
 *else*:  
 keyMap = *self*.getCamelCaseMap(returnType)  
 res = OrmProxy.getEntityFromDict(returnType, sqlResult, keyMap)  
 *return* res  
  
 *def \_\_call\_\_*(*self*, \*\*kwargs):  
 conn = MySQLHelper()  
 ret = *None  
 try*:  
 sql = *self*.sqlGenerator(kwargs)  
 args = *self*.argsGenerator(sql, kwargs)  
 *if self*.sqlType == Select:  
 ret = *self*.\_query(conn, *self*.preparedSql, *self*.returnType, args)  
 *elif self*.sqlType == Insert:  
 ret = conn.insert\_one(*self*.preparedSql, args)  
 *elif self*.sqlType == Update:  
 ret = conn.update(*self*.preparedSql, args)  
 *elif self*.sqlType == Delete:  
 ret = conn.delete(*self*.preparedSql, args)  
 conn.commit()  
 *except Exception as* e:  
 *print*(e)  
 conn.commit('rollback')  
 *finally*:  
 conn.close()  
 *return* ret  
  
 @classmethod  
 *def* startPage(*cls*, pageNum, pageSize, orderBy=*None*):  
 *cls*.LOCAL\_PAGE.page = PageDomain(pageNum, pageSize, orderBy)  
  
 @staticmethod  
 *def* getTotal(result: *list*):  
 *if isinstance*(result, PageResult):  
 *return* result.total  
 *return len*(result) *if* result *else* 0

这段十分冗长的代码展示了ORM的具体实现细节，包括四种数据库操作对应的装饰器，用正则表达式解析获取sql关键字和条件标签，在其中使用eval函数通过动态字段关系表达式获取动态字段。考虑到数据分页的需求，使用正则表达式额外处理了数据条数查询语句的自动生成。

@ProxyFactory(OrmProxy)  
*class* TypeDao:  
 @Select("""select type\_id, type\_name, parent\_id, create\_time  
 from note\_type""",  
 List[TypeDomain])  
 *def* list4Table(*self*, typeId):  
 *pass* @Select("""select type\_id, type\_name, parent\_id from note\_type""",  
 List[TypeDomain])  
 *def* list(*self*):  
 *pass* @Insert("""insert into note\_type(type\_name,parent\_id) values(  
 #{domain.typeName},#{domain.parentId})""")  
 *def* add(*self*, domain: TypeDomain):  
 *pass* @Update("""update note\_type set   
 <if test="domain.typeName">type\_name=#{domain.typeName},</if>  
 <if test="domain.parentId">note\_parent\_id=#{domain.parentId},</if>  
 type\_id=type\_id  
 where type\_id=#{domain.typeId}""")  
 *def* update(*self*, domain: TypeDomain):  
 *pass* @Delete("""delete from note\_type where type\_id=#{typeId}""")  
 *def* delete(*self*, typeId):  
 *pass*

以上是这个实现的一个测试用例，可以看到只需将SQL语句写入装饰器中并申明返回的实体类型（数组也同样可以解析），方法不需要实现语句，即可获取解析好的实体对象，这将大大简化业务的开发。