电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 基于 QT 的可移植组态软件的设计与实现

专业学位类别	工 程 硕 士
学　　号	*****
作 者 姓 名	*****
指 导 教 师	***** 教授

分类号 _____ 密级 _____

UDC [注1] _____

学　位　论　文

基于 QT 的可移植组态软件的设计与实现

（题名和副题名）

*****

（作者姓名）

指导教师 ***** 教　授

电子科技大学 成　都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工 程 硕 士

工程领域名称 计算机技术

提交论文日期 2017.3 论文答辩日期 2017.5

学位授予单位和日期 电子科技大学 2017 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

DESIGN AND IMPLEMENTATION OF DISTRIBUTED CONTROL SYSTEM BASED ON EMBEDDED SOFT PLC

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: *****

Advisor: **Professor *****

School: **School of Computer Science and Engineering**

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：_____ 日期：　　年　　月　　日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期：　　年　　月　　日

摘要

工业控制技术一直是人们研究的重点内容，而组态软件作为其研究的一项重要成果，也在不断地发展和完善。随着国内制造业的繁荣，很多中小型企业需要利用组态软件来进行监控系统的设计，因此组态软件具有很好的市场。但是目前国内的组态软件往往存在授权费用高、功能冗余、平台限制和系统封闭等缺点，因此开发一款价格低廉的、跨平台的、系统开放的轻量型组态软件就显得很有必要。

本文主要讨论如何基于 Qt 这个 C++开发框架来设计一款组态软件，由于 Qt 开发平台提供了跨平台的特性，因此该款组态软件可以在多个平台上运行。

组态软件一般分为三个部分：开发系统、运行系统和工程文件。本文就这三个部分进行了详细的设计与实现。

开发系统是用户设计控制系统的工具，它为用户提供了丰富的图形显示和系统配置，使得用户只需简单的操作就能够完成系统构建。本文重点描述了图元模块、变量配置模块、系统配置模块和报警模块的设计与实现，实现过程中充分利用了 Qt 提供的图形处理相关的工具。

运行系统作为监控系统的运行环境，由交互界面模块、控制模块、通信模块和历史数据处理模块四个部分组成。设计过程中利用了 Qt 的信号与槽机制以及线程间的同步机制来完成模块间的通信与同步，同时利用 Qt 提供的丰富的数据库操作接口来实现历史数据的存储与访问。

工程文件作为监控系统的设计文档，其主要功能是保存设计人员的各种配置信息，从而方便开发系统对工程进行修改以及运行系统运行。其设计实现的过程主要利用了 Qt 提供的相应的序列化和反序列化工具，这样很好地实现了数据的存储与解析。

最后本文通过构造一个应用场景，充分地测试了该组态软件的各个模块，并且同时在 Windows 平台和 Linux 平台进行运行效果的展示，从而体现了其可移植的特性。同时本着系统开放的目标，在设计的过程中充分考虑了可扩展性，并为开发人员提供了相应扩展接口。

关键字：组态软件，Qt，跨平台，可移植

ABSTRACT

Industrial control technology has always been the focus of people's research, as an important achievement of its research, configuration software gets developed and improved continuely. With the prosperity of the domestic manufacturing industry, many small and medium enterprises need to use the configuration software to design surveillance system, so the configuration software has a very good market. But the current configuration software often has high cost of authorization, functional redundancy, platform restrictions, closure and other shortcomings, so developing a low price, cross-platform, open and lightweight configuration software is very necessary.

This thesis mainly discusses how to design a configuration software based on Qt, as a C++ development framework. Because Qt development platform provides cross-platform features, the configuration software designed by it can run on multiple platforms.

Configuration software is generally divided into three parts: development system, operating system and project file. This thesis has described the detailed design and realization about these three parts.

Development system provides tools for users to design the surveillance system, which includes a wealth of graphical display and system configuration, and allows users to design the system easily. This part focuses on the design and implementation of the graphic module, the variable configuration module, the system configuration module and the alarm module. And, in this process, some graphic tools provided by Qt were used fully.

Operating system as the operating environment of the surveillance system, which includes the interactive interface module, the control module, the communication module and the historical data processing module. Qt’s signal and slot mechanism and the synchronization mechanism between the threads were used to achieve the communication and synchronization between modules, while a wealth of database operation interface provided by Qt was used to achieve the historical data storage and access.

Project file as design documents, saves the designer's configuration information, and provides development system and operation system with data. Data stored and analyzed by data serialization and deserialization tools provided by Qt.

Finally, we designed an application scenario, in this scenario, most modules of the configuration software were tested. And for testing its portability, we made it run

simultaneously on the Windows platform and the Linux platform. At the same time, this configuration software provide expansion interfaces for users to add other functions, which makes it scalable.

Keywords: Configuration Software, Qt, cross-platform, portability

目录

第一章 绪论 1
1.1 课题研究背景 1
1.2 国内外研究现状 3
1.2.1 组态软件现状 3
1.2.2 组态软件发展趋势 4
1.3 课题研究意义 5
1.4 论文主要内容及结构 6
第二章 组态软件的总体框架设计及相关技术研究 8
2.1 组态软件的结构 8
2.2 开发系统的总体框架 9
2.3 运行系统的总体框架 11
2.4 工程文件设计 13
2.5 相关技术研究 14
2.6 本章小结 15
第三章 组态软件开发系统的设计与实现 16
3.1 图元模块 16
3.1.1 图元简介 16
3.1.2 图元的设计与实现 16
3.1.2 图元管理器的设计与实现 31
3.1.4 画布管理器的设计与实现 35
3.2 变量配置模块 37
3.2.1 普通变量的设计与实现 37
3.2.2 结构变量 39
3.3 系统配置模块 39
3.3.1 监控站点配置 40
3.3.2 驱动配置 41
3.3.3 通信方式配置 42
3.4 查询报警模块 42
3.4.1 报警系统 42
3.4.2 历史数据查询 45
3.5 本章小结 46
第四章 组态软件工程文件的设计与实现 47
4.1 工程文件的分类 47
4.2 工程文件的数据组织 48
4.3 数据的序列化与反序列化 49
4.4 工程导入流程 50
4.5 本章小结 51
第五章 组态软件运行系统的设计与实现 52
5.1 交互界面模块 52

5.2 控制模块 55
5.2.1 变量管理器 55
5.2.2 系统信息管理器 57
5.3 通信模块 59
5.3.1 驱动管理器 59
5.3.2 数据采集器和发送器 60
5.4 历史数据处理模块 62
5.5 模块间同步 62
5.6 本章小结 64
第六章 组态软件的测试 65
6.1 应用场景描述 65
6.2 系统设计 66
6.3 测试过程及结果分析 67
6.3.1 测试过程 67
6.3.2 测试结果分析 68
6.4 本章小结 71
第七章 总结 72
致谢 73
参考文献 74

第一章 绪论

1.1 课题研究背景

随着工业规模的不断扩大，对应的工业控制技术也变得越来越复杂。最初，各种控制面板分布在不同的工业现场，因此需要大量的人力资源来维护这些相互独立的面板，然而这种工作方式并不能对整个生产过程形成一个全局的掌控[1]。

随后人们发现，可以手动记录每个面板的信息，然后将其传输到控制中心，这样就可以让控制中心很容易了解到各个生产间的状况，从而形成一个整体了解。这种控制方式一般包含一个控制中心，多个控制点。控制点负责采集现场的实时生产数据，控制中心负责结合各个控制点反馈的实际数据做出相应的处理[2]。尽管这种控制方式较早期的控制方式进步了，但仍存在着需要大量人力资源这个问题，因为每个控制点必须配置一个以上的操作员，负责记录数据以及执行相关的命令，而且还要周期地向控制中心反馈数据。大量的人工操作还会导致三个问题，一是不能及时将现场数据反馈给控制中心，这样就会使得有些紧急事件无法被恰当适时地处理，从而造成整个生产流程的中断，导致巨大损失；二是由于控制中心的工作人员分析不到位，导致不能得到正确的结果，从而下达错误的命令；三是由于命令是由控制中心下达，各个控制点可能不能及时的收到命令，因此造成命令不能按时执行，也有可能中断整个生产过程[3]。

随着通信协议的不断成熟以及图形显示技术的发展，上述问题得到了解决。人们为每个控制点设计一套独立的控制系统，该系统能够采集实时数据，处理相关指令，从而很好地监控生产设备。同时利用一个监控中心来协调各种设备，但是此时控制点和控制中心的数据交换已经不再通过人力进行，而是通过一些可靠的通信协议进行传输。由于图形显示技术的发展，以及计算机计算能力的提升，设计一套复杂的算法来对整个控制流程进行控制以及显示就变得现实和可行了[4]。

控制技术经过不断发展，整个工业控制已经达到了高度自动化。人们也总结出了整个工业控制的层次，这样更有利于对每个层次设计出高效的算法以及解决方案，同时也使得功能独立，降低了耦合性。一般工业控制分为五个层次[5]：

- 工业现场(Field Level)：包含各种现场设备，如温度传感器、终端控制器等。
- 直接控制层(Direct Control)：包含各种工业化的 I/O 模块，一般都带有自己的处理器。
- 工厂监控层(Plant Supervisory)：包含一台或多台监控计算机，它们负责从各个控制点收集数据，为监控人员展示整个生产状况，同时负责命令转发。

- 产品控制层(Production Control)：目标是对整个生产过程进行全局监控，以及做出相关上层决策。
- 生产调度层(Production Scheduling)：对多条产品线进行安排。

其中工厂监控层是比较复杂的一层，主要的软件部分和计算部分都在这一层，因此是研究的核心。国外将其称为 SCADA(Supervisory Control and Data Acquisition)，国内对应地称作组态软件，它利用计算机，网络数据交换协议和高层的图形化界面来对生产设备进行监控和管理，在 SCADA 监控计算机系统，操作人员可以通过图形化操作界面来监视生产情况，同时还可以通过它向下层控制器设置参数。因此 SCADA 的核心理念就是为了找到一种通用的方法，使得能够和分布在各个地方的终端控制器通信[6]。为此，随着科技的不断的发展，SCADA 的结构也在不断的变化。到目前为止总共经历了四代变化：

- 集中式时代：早期的 SCADA 系统的计算任务是交给小型机来完成的，由于当时的通用网络服务还没有建立起来，因此只能用一些使用环境比较受限的通信协议，从而使得 SCADA 系统无法与外界或远程的系统交互，只能相对独立地工作。而且此时的 SCADA 系统一般会配置一台主机来进行数据备份[7]，以防发生意外导致信息丢失，因此相应的系统也产生了，如数字设备公司(Digital Equipment Corporation)生产的 PDP-11 系列。
- 分布式时代：SCADA 系统将信息和命令分发到各个站点，这些站点通过局域网连接起来，因此可以达到近似的实时性。每个站点将会负责与各自相关的特殊任务，与第一代 SCADA 系统相比将节省不少开支。但是由于网络协议还没有普及化和标准化，局域网所用的传输协议仍是私有的，除协议的开发人员之外，很少有人能够正确的使用这些协议，因此会为 SCADA 系统带来一些安全隐患[8]。
- 网络时代：相对于分布式时代，网络时代提供了功能完善的网络通信协议，使得其信息交互更加高效更加安全。而且对于地域跨度很大的 SCADA 系统，只有网络协议才能满足其需求。同时网络的加入，使得几个平行的 SCADA 系统能够进行交互，从而让控制范围更大[9]。
- 物联网时代：随着云计算技术的成熟，一些企业开始提供商业的云计算服务，SCADA 系统也开始采用物联网的相关技术来缩减成本，增加系统的可整合性。同时由于云服务环境提供了良好的水平伸缩性，在这种环境下实现较为复杂的算法也变得更加容易[10-11]。早先，SCADA 系统的数据收集采取一对一的映射方式，该方法比较简陋，因此提出了数据模型。在数据模型中，每种设备在 SCADA 软件中都有相应的虚拟表示，它可以反映设备的地址映射情况，而且

还携带了其它相关信息，非常有利于 SCADA 系统对设备的管理[12]。

1.2 国内外研究现状

1.2.1 组态软件现状

上文简要地介绍了 SCADA 系统，国内将其称为组态软件，两者在软件结构上是保持一致的，因此下文将统一使用组态软件来表示两者。组态软件拥有很好的伸缩性，小到只有几十个控制点的系统，大到上千控制点的系统，都可以通过组态软件进行组态，而且往往能够得到很好的成本效益。组态软件一般用在工业领域（如工业生产，过程控制，能源控制，加工等），基础设施领域（如水利，物流，灌溉系统，油气管道控制，电力传输等）和一些基于大型设施的处理过程（如楼宇，机场，船舶，太空站等）[13]。

组态软件一般由以下几个部分组成：

(1) **监控计算机：**这是整个组态软件系统的核心部分，涉及到计算机，负责与现场控制器通信的软件以及一个运行在操作台的人机交互软件。它汇集来自多个处理过程的数据，还要负责发送控制命令到各个现场设备。在稍小一些的组态软件运行系统中，监控计算机可能就只有一台 PC 机，人机交互界面是其中的一部分。而在稍大一些的组态软件运行系统中，整个监控中心可能会包含多个有显示界面的计算机，多台负责采集数据的服务器。

(2) **远程终端单元：**也称作 RTU(Remote terminal unit)，往下与各种传感器相连，往上通过网络与监控计算机相连。RTU 作为一种智能 I/O，往往具有独立的控制功能，如利用梯形图来完成布尔逻辑操作[14]。

(3) **可编程逻辑控制器：**即 PLC(Programmable Logic Controller)，与 RTU 相同，往下与各种传感器相连，往上通过网络与监控计算机相连。但相比 RTU，PLC 具有更强的控制能力，而且支持一种或多种 IEC 61131-3 编程语言对其编程[16]。由于 PLC 相较于 RTU 更加经济，更加通用，扩展性更好而且支持可配置，所以目前使用 RTU 的地方都被 PLC 替换掉了。

(4) **通信设施：**即用于监控计算机系统和 RTUs 或 PLCs 之间的通讯模块，通常使用的是工业标准协议或制造商专有协议。通过监控计算机给出的最近的一条命令，RTUs 或 PLCs 能够自主地完成控制操作。同时通信设施还要保证，通信连接的部分损坏并不会影响整个工业生产过程的控制，而且一旦通信恢复，操作人员将能够继续进行监控。

(5) **人机界面：**即 HMI(Human-Machine Interface)，监控系统的操作窗口。人机界面通过虚拟图表，警告页面，事件日志页面以及趋势图等构成[16]。利用采集的

数据，改变各种虚拟图表以及其他页面的状态，从而形成一个工业现场的概览图。一般人机界面是为操作人员准备的，其目的就是收集外部设备的数据，形成报告，发出相关警告，发送相关通知。虚拟图表是由一些代表与处理过程相关的元素的线或符号组成的，往往随着数据的变化，形成一种动画的效果。对整个工业控制过程的操作也是通过人机界面完成的，一般涉及鼠标操作，键盘操作或屏幕触摸。

(6) **报警处理：**是组态软件中比较重要的一部分。系统监视器将会不断地检测报警条件是否满足，而且还要决定何时可以触发一个报警事件。一旦一个报警事件被侦测到了，系统将会采取行动（如触发一个或多个报警指示器，邮件或短信通知远程管理人员等）。在多数情况下，组态软件运行系统的操作人员可以肯定某些报警事件的存在，但这也只能关闭部分报警指示器，一定要报警条件被清除，这时所有的报警指示器才可以关闭。报警指示器的目的是为了吸引操作人员的注意，一般包括报警声，或在屏幕上弹出一个警示框，或在屏幕相关区域进行特殊颜色的标识[17]。

1.2.2 组态软件发展趋势

如前文所述，组态软件的概念源于 SCADA，而 SCADA 是国外的产物，因此组态软件的发展也主要集中在国外，目前市场上比较知名的国外组态软件有 Wonderware 公司的 InTouch，西门子公司的 WinCC 和 Intellution 公司的 iFIX 等；随着国内工控技术的累积与发展，同时借鉴了国外知名的组态软件，也出现了一批优秀的组态软件，如北京亚控公司的组态王，大庆三维公司的力控。国内的组态软件主要的优势在于本土化做得很好，同时积极吸取国外组态软件的优点，也使得自己的市场竞争力得到提升。

目前组态软件的发展主要体现在以下两个方面：

- **运行平台：**早期由于 DOS 系统比较盛行，组态软件主要运行在此操作系统下，由于此时的处理器大多是 16 位的，因此对应的组态软件也是 16 位的，这就大大限制了组态软件的功能，不能提供生动的图像显示，不能承载过多的计算，不能运行复杂的算法等等，同时所控制的设备数也受限制。随着 Windows 系列系统的诞生，处理器的升级，组态软件也朝着功能丰富化的方向发展，组态软件可以运行复杂的动画，可以模拟复杂的器件，可以实现复杂的控制算法，同时控制能力也得到大大提升。今后组态软件的发展是朝着多平台的方向发展的，要求不仅能够在传统的 PC 平台上运行，还要能够在移动设备上运行。
- **通信环境：**起初组态软件的通信协议只是局限在一些商用的授权通信协议上，

可互联的范围很小，随着网络技术的发展和成熟，组态软件也开始逐渐接纳TCP/IP 协议，这就使得组态软件的控制范围可以跨越生产间、跨越工厂。随着云计算技术的发展，为了更好的管理数据，组态软件也开始采用云服务。

由于组态软件从使用一些私有的授权协议转为使用一些标准协议或一些开源协议，从而更好的网络化，这样虽然使得组态软件运行系统可以更好的互联，但也为网络攻击提供了方便之门。存在的威胁主要分为两类：一是在未授权的情况下得到控制权，通常是人为的或意外被病毒感染。二是通过网络途径攻击，通过网络将病毒植入系统或劫持替换控制命令来达到获取控制权的目的，因为这些控制协议没有使用加密技术[18]。

组态软件一般用在一些大型的设施中，如电力控制系统等，这些设施往往影响到人们生活的方方面面，如果一旦被攻击，将会造成巨大损失，因此组态软件的通信安全也是组态软件的研究内容之一。

1.3 课题研究意义

通过使用组态软件提供的相关工具，开发人员可以通过拖拉的方式来构造人机界面，然后利用参数配置框来对系统监控流程进行配置，同时加上适当的动画效果可以达到再现整个生产过程的目的，这就是组态软件的使用方式。整个过程中开发人员不会进行大量的编程工作，只需要对生产工艺有深刻的了解就可以开发出带有人机界面的控制系统。在组态软件产生之前，开发一个控制系统，一般是先分析整个控制流程，然后用高级语言进行开发，整个开发周期很长，而且人力物力消耗很大，最关键的还是伸缩性扩展性非常不好。如果遇到需求的变动，可能将会导致整个软件架构的变动，将会带来不能承受的损失。同时，由于每个系统都是独立开发的，有些具有相同功能的模块被多次开发，这样费时又费力。组态软件通过将通用功能进行模块化，提供良好的扩展性，使得可以被反复利用，同时由于这些模块都是经过反复验证过的，因此可靠性更高[19]。

通过以上简述，研究组态软件的架构与实现是非常有意义的，这将为相关领域带来更好的效益。

表 1-1 主流组态软件对比

对比项 / 组态软件	价格	功能	运行平台	应用场景	扩展性
InTouch	按模块收费	多而复杂	Windows	工业控制	较封闭
Citech	软件授权	多而复杂	Windows	工业控制	封闭
WinCC	软件授权	使用困难	Windows	工业控制	封闭

iFIX	软件授权	多而复杂	Windows	工业控制	封闭
组态王	按点收费	覆盖多领域	Windows	工业控制	允许扩展
ForceControl	按点收费	覆盖多领域	Windows	工业控制	允许扩展
RealInfo	按点收费	覆盖多领域	Windows	工业控制	允许扩展

如表 1-1 所示，组态软件主要在如下方面存在问题，这也就成为本文研究的出发点。

(1) 价格高：目前组态软件一般是以控制点的数量来进行收费的。即将一个控制系统中需要监控的设备抽象成一个个控制点，然后按点收费，即使采用这种按需付费的方式也将是难以承受的，一个小型的系统可能将会为此支付数万元，大一些的系统将支付数十万元。这还不包括开发费用，只是授权使用费而已。

(2) 功能繁杂：目前的组态软件讲究的是大而全。整个组态软件开发系统很大，包含各种各样的图元、图标、虚拟仪表等，提供多领域的解决方案。由于包含的内容多，因此使用流程也变得复杂，需要投入更多的时间进行学习。

(3) 平台限制：目前组态软件一般运行在 Windows 系统上，由于移动设备的不断发展，以及其运算功能的不断增强，希望组态软件能够运行在移动设备或 Linux 平台的需求越来越强烈[20]。

(4) 应用场景单一：目前组态软件主要应用在工业控制领域，基础设施领域和一些大型控制环境中。随着各种智能设备的增加，设备之间的交互也变得很重要了，组态软件的控制方式很适合控制这些联网智能设备。

(5) 可扩展性限制：由于目前的组态软件都是商用产品，对外暴露的接口很少，如果想要对其进行二次开发根本是不可能的，因此这就限制了组态软件在一些特殊领域的使用。

本文主要探讨如何利用 Qt 这个跨平台应用程序开发框架，来开发一款轻量型组态软件[21]。由于 Qt 是基于 C++的，并且使用其提供的 API 进行相关开发，可以保证同一套代码能够在不同平台编译，得到相同的运行结果，这就使得该款组态软件满足了多平台支持的特性。其次，该组态软件的设计主要完成的是其整体框架的构建，其它比较特殊的模块（如与特殊领域相关的一些控制算法）不进行设计，但为其留好了接口，这样方便开发人员可以对该组态软件进行二次开发。同时该组态软件由于主要利用 TCP/IP 协议进行通信，因此可以将其利用到其他领域，如智能设备互联等。

1.4 论文主要内容及结构

本文主要给出一种组态软件的设计方法，并在 Qt 开发平台上将其实现。开发出的组态软件能够满足跨平台的工作特性，并且能够在多种应用场合中使用。由于

时间与工作量的关系，本文将主要描述整体框架的设计与实现，一些与特殊行业相关的功能模块或解决方案并没有进行设计与实现，但是为开发者保留了接口。

论文包括以下章节：

第一章：绪论。主要描述组态软件产生的背景，然后介绍其演化过程，最后描述其存在的问题，围绕这些问题描述本课题的研究意义。

第二章：组态软件的总体框架设计及相关技术研究。该章将描述组态软件的组成，然后讨论开发实现的工具与技术。

第三章：组态软件开发系统的设计与实现。该章将描述组态软件开发系统的主要组成模块，然后根据每个模块的功能需求，将给出具体的设计方案，最后给出实现过程。

第四章：组态软件工程文件的设计与实现。该章将讨论组态软件工程文件的设计，将重点描述工程文件的数据组织格式，并且将描述工程文件的分类。

第五章：组态软件运行系统的设计与实现。该章将描述组态软件运行系统的实现，将对运行系统的四个模块进行详细介绍，并给出设计方案。

第六章：组态软件的测试。该章将构造一个应用场景，尽可能全面地测试组态软件的各个模块。

第七章：总结。将对整个设计过程进行总结，肯定成果，提出展望。

第二章 组态软件的总体框架设计及相关技术研究

目前主流的组态软件主要用在一些大型的工业控制场景，但是随着互联网、物联网的发展，一些新兴的领域也开始需要组态软件这样的一类软件，然而由于监控的要求不一样，很多主流组态软件的一些必要功能并不是必需的，如强实时性、复杂的控制算法、丰富的控制模块和丰富的驱动等。因此设计一款可跨平台的，轻量型的并且可以高度自定义的组态软件是有必要的，本文将提出一种设计方案并且进行相应实现。

2.1 组态软件的结构

在组态软件出现以前，控制系统一般都会设计人机交互界面，这样可以对监控对象有一个全局的把控。相关的控制算法则需要专业人员才可以进行设计，而对于界面的开发则不需要那么高的要求。对于一个系统的设计，一般控制模块与界面的设计是同时进行的，因此会造成软件的复杂度、耦合性大大增加，同时导致开发人员不能发挥所长，这样只会延长整个项目开发周期，即使勉强完成了产品开发，也不一定能够满足实际需求[22]。

为了能够很好的适应现实需求的多变，组态软件产生了。组态软件通过将各个功能进行模块化，然后采用如“搭积木”的方式，将需要的各个功能加入到工程中，然后进行相应的配置，就可以生成一个满足需求的系统。

组态软件通过将各个功能进行模块化的方式，来使得整个系统的耦合性降低。一般组态软件会将控制部分和界面部分进行分离，这样就使得控制算法可以设计成一个个独立的模块。与以前相比，控制系统的控制部分和界面部分是作为一个整体来开发的，算法往往糅合到整个设计之中，这就使得算法的实现根本不能复用，从而导致重复开发，浪费了人力物力。

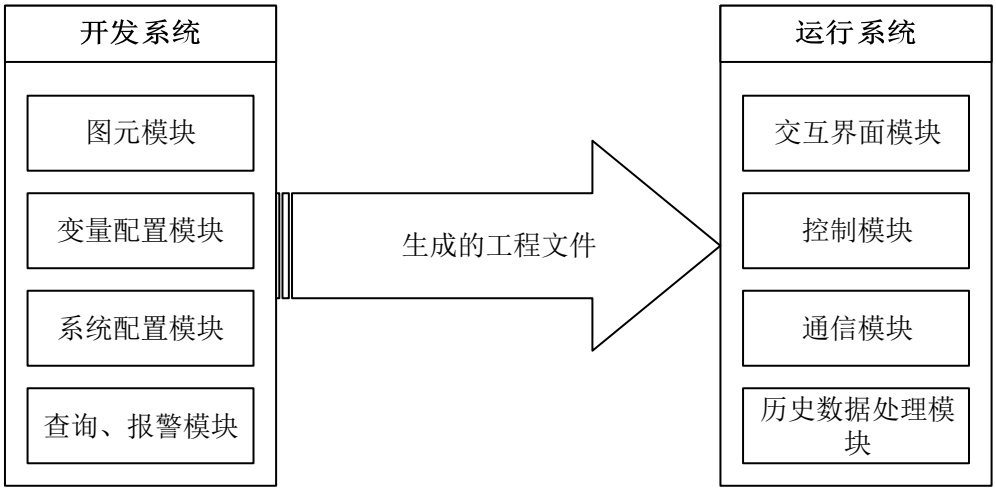

图 2-1 组态软件的工作方式

如图 2-1 所示，组态软件主要分为三个部分：开发系统部分、工程文件和运行系统部分[23]。

开发系统部分主要供系统开发工程师使用，工程师利用组态软件提供的交互式开发系统，通过拖拽、移动、配置和自定义添加功能等方式来对目标系统进行组态。整个开发系统由多个模块组成，各个模块提供了相应的控件以及参数配置界面，开发工程师选定工程需要的相应模块，然后对其进行参数的配置，从而满足生产工艺的需求。

工程文件部分主要是记录工程师通过开发系统进行开发后生成的各种文件，这些文件记录了开发过程中的配置参数。

运行系统部分由交互界面模块、控制模块、通信模块和历史数据处理模块组成。它利用生成的工程文件，形成交互式界面，供监控人员使用。

因此组态软件的工作流程也就非常清晰了：开发工程师通过开发系统，利用其提供的相关功能表达自己的设计意图，然后开发系统生成相应的工程文件，运行系统利用这些工程文件再现了开发工程师的设计意图，最终得到一个具有人机交互界面的监控系统。

本文的工作将要完成开发系统、工程文件组织和运行系统的相关模块的设计与实现。

2.2 开发系统的总体框架

开发系统由多个部分组成，而且为用户提供了良好的可交互式界面，用户可以通过指导完成工程设计。本文只是提出一种开发系统总体框架的设计，并且完成了基本模块的实现。

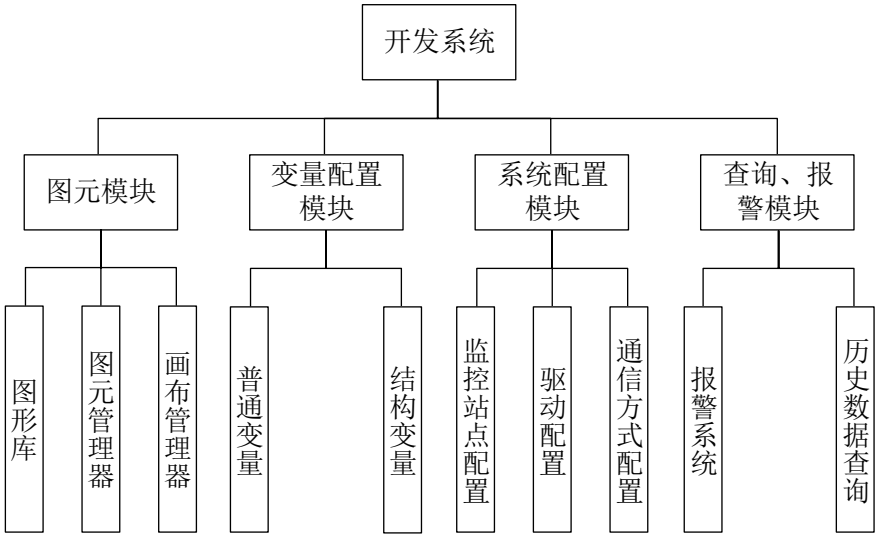

图 2-2 开发系统组成

如图 2-2 所示，开发系统主要由四个部分组成：图元模块、变量配置模块、系统配置模块和查询、报警模块[24]。下面将对四个模块进行分别介绍：

（1）图元模块

图元模块主要包含三个功能部分：图形库、图元管理器和画布管理器。

图形库：包含点、线（直线、折线和曲线）、多边形（三角形、矩形）以及圆等基本图形的绘制。此外，图形库还包含一些拟物化的图形，比如阀门、开关、电机、按钮、指示灯以及仪表等，这些图形能够生动地再现生产现场的生产过程。用户可以根据自己的需求对图元进行变形，从而满足工程的视图效果。同时每个图元都可以进行参数配置，与相应的变量进行关联，这样就使得图形能够根据实时的生产数据进行状态改变。同时图元之间还可以进行动画连接，因为在实际生产过程中，生产是一个动态的过程，用动画的方式将其展示出来是合理的[25]。

图元管理器：图元管理器首先要将图元进行分类，然后使得用户能够很方便的找到自己需要的图元；其次，图元管理器还要以适当的方式，在恰当的时候为用户在画板上生成图元，并将其显示出来；再者，图元管理器还需要允许用户添加自定义的图元；最后，图元管理器还要能从动态链接库中将图元导出。

画布管理器：画布管理器需要实时记录目前用户添加的图元，以及每个图元的配置信息。同时画布还需要记录用户的操作，这样方便用户能够将设计工作返回到某一状态。

（2）变量配置模块

变量配置模块由两个功能部分组成：普通变量和结构变量。

普通变量：包含整型、浮点型以及布尔型变量。这些都是一些基础型的变量，

在整个工程设计中，每个变量都有唯一的标识符。变量都有相应的属性，这是开发人员根据实际需求来确定的。每个变量都需要确定变量名、变量值的来源、变量的类型以及变量的报警属性，为此变量配置将会有专门的对话框进行处理。

结构变量：结构变量是由两个及以上的普通变量构成的，这样的好处在于对功能相同的变量进行统一管理。

（3）系统配置模块

系统配置模块包含三部分：监控站点配置、驱动配置和通信方式配置。

监控站点配置：主要对需要进行监控的站点进行相关配置。如站点需要管理的数据、命令操作以及相关的标识符等。

驱动配置：由于每个站点通信格式都不相同，因此需要为此提供相应的驱动，这样才能让监控系统采集到有效的数据。驱动配置只是配置相关驱动程序存放的位置。

通信方式配置：每个站点与主机的通信方式有很多（以太网、串口和蓝牙），因此为了能够让监控系统识别，从而使用适当的方法，开发人员需要对每个监控点与主机的通信方式进行配置。

（4）查询、报警模块

查询、报警模块主要包含两个功能部分：报警系统和历史数据查询。

报警系统：监控系统主要完成对生产过程的实时监控，因此它会分析采集回来的现场数据，然后做出相应的处理，其中比较重要的一项就是向相应的工作人员发出警报，使其能够迅速处理存在的问题。对于报警系统的设计，关键点在于对数据的处理，而现场数据在组态软件中都被抽象成了变量，因此变量中的一个重要属性就是与是否报警相关的。开发人员可以通过配置相关参数，达到设置报警界限、警报优先级以及警报显示等内容。

历史数据查询：在一个监控系统中，操作员往往需要了解过往的生产状况，因此需要获得历史数据，历史数据查询功能部分主要为操作员提供这样一种查询接口，方便其根据相应的条件进行查询。

2.3 运行系统的总体框架

运行系统是监控系统的运行环境，它提供了相应的控制逻辑来保证监控流程的正确。

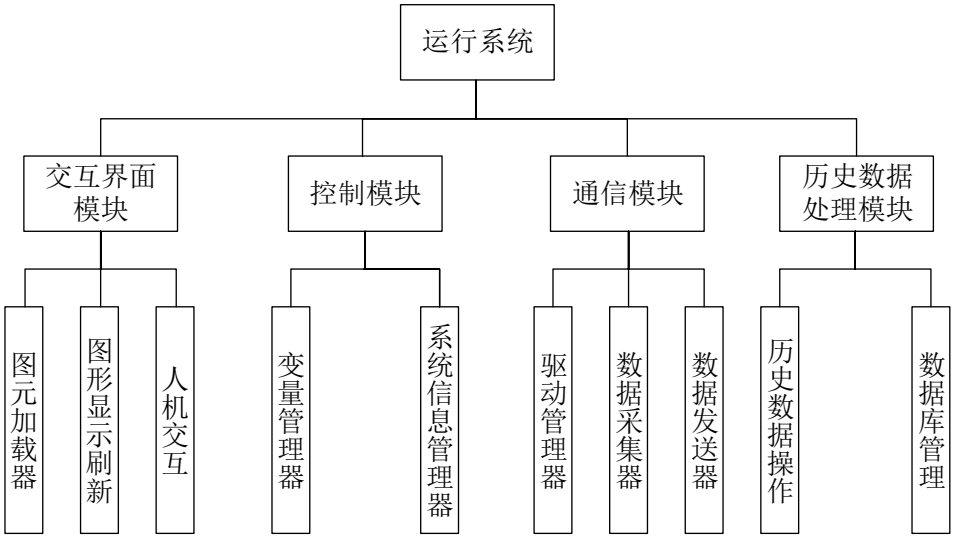

图 2-3 运行系统组成

如图 2-3 所示，运行系统由四个模块组成：交互界面模块、控制模块、通信模块和历史数据处理模块。下面将依次介绍这四个模块所完成的主要功能：

（1）交互界面模块

交互界面模块主要包含三部分：图元加载器、图形显示刷新和人机交互。

图元加载器：由于在开发过程中，设计者设计的界面最终存储在文件中，因此在运行系统中需要一个功能模块来加载图元，从而形成监视画面。

图形显示刷新：开发系统中提到过，很多图元需要根据实时数据来改变自己的显示状态，因此运行系统需要提供图形刷新的功能。

人机交互：在 HMI(Human-Machine Interface)中，操作人员可能会根据当前的监控结果下发一些命令，这就涉及到两个部分的工作。一、生成命令，并且将其发送到相应的生产现场；二、命令发出或有反馈后，将要在界面上形成相应的动画。同时，操作人员还会希望能够获取历史数据，因此在监控系统中会有与查询相关的交互界面[26]。

（2）控制模块

控制模块包含变量管理器和系统信息管理器两个部分。

变量管理器：上文介绍过变量在组态软件中的重要性，通过对它的引用，各个模块可以获得自己关心的实时数据。因此对于变量的一系列操作将会通过一个功能模块来实现，这个模块将负责变量的刷新，同时让各个模块能够向监控人员反馈实时数据；该功能模块还要负责最近变量的存储，使得可以快速地获取近期数据。

系统信息管理器：这个功能部分对系统的配置信息进行管理，比如站点的相关属性，驱动的位置，变量的解析等。系统信息管理器将这些信息通过适当的数据结

构组织起来，提供相应的接口，方便各个模块之间的引用。

（3）通信模块

通信模块由三部分组成：驱动管理器、数据采集器和数据发送器。

驱动管理器：主要对开发者配置的驱动程序进行管理，方便使用的时候进行索引。同时由于驱动大多时候是以动态链接库的形式提供的，因此驱动管理器还要完成驱动加载的工作。

数据采集器：主要完成从各个站点采集数据的工作。

数据发送器：主要完成向各个站点发送数据的工作。

（4）历史数据处理模块

历史数据处理模块包含历史数据操作和数据库管理两个部分。

历史数据操作：包含查询、删除历史数据。组态软件中有相应的控件可以完成数据库的查询和修改操作。

数据库管理：由于监控软件不仅需要实时了解监控现场的数据，而且有时还要查询以前的数据，因此所有的数据都应该进行存盘，为此需要有专门的数据库来管理。

2.4 工程文件设计

工程文件作为一座桥梁，将开发系统和运行系统联系其来，开发人员在开发系统上进行设计工作，然后在运行系统上展示设计意图。因此对于工程文件的设计是组态软件中很重要的一部分，良好的结构设计可以使得软件的设计变得简单，同时使得软件的可扩展性得到提升。

组态软件中生成的工程文件分为两个部分：文件信息和数据。

文件信息对文件的详细描述信息进行记录。一般包含如下几项[27]：

- 标识符：确定文件的类型。
- 版本号：版本号表示文件是由哪个版本的组态软件生成的，这样区分是为了防止不同版本之间的交错使用。
- 幻数：用来识别版本类型，有时可以通过后缀名来识别文件类型，但是后缀名可以修改，因此通过幻数可以唯一确定文件类型，从而使得文件得到正确地处理。
- 创建者、创建时间、修改时间：这些都是工程创建和处理时的一些基本信息，这些信息使得工程能够被很好的管理。

数据部分承载着文件的主要内容，其中通过开发系统进行的各种配置信息都将记录在这里。一般一个文件会包含多个数据项，而每个数据项由两部分组成：标

识符和数据。标识符作为这个数据项的唯一标识，数据则是其所带的内容。数据一般包含多个信息点，而信息点由三部分组成：类型、长度和数据。类型表示数据的形态，长度代表数据的大小，而数据则是真正的有效数据。通过这种方式一个数据项可以包含多个数据信息点，这和实际情况是相吻合的，比如在变量配置的过程中，有些变量不需要报警属性，而有些又需要，因此这样就会造成处理的不统一，如果有效数据是变长的，那么这种不统一将可以被避免。

2.5 相关技术研究

本文主要依托的开发工具是 Qt，它是一个著名的跨平台的 C++应用程序开发框架，利用它开发的应用程序可以在多个平台上运行，它提供了丰富的接口，并且都是用户友好型的。很多著名软件都是使用 Qt 进行开发的，如 Autodesk Maya、Google Earth、Skype 和 WPS Office 等。同时 Qt 也支持很多平台，如基于 Linux/Unix(Solaris、AIX、HP-UX 等)，基于 Apple 平台(macOS、iOS)，基于微软平台(Windows 系列、Windows CE 以及 Window RT 系列)，甚至于一些嵌入式中的实时操作系统平台也提供了支持，如 VxWorks 和 QNX[28-29]。

Qt 诞生于 1991 年，由 Trolltech（奇趣科技）发布，在 2008 年诺基亚斥资 1.5 亿美元收购了 Trolltech，将 Qt 用于 Symbian 程序的开发。在 2012 年，诺基亚又将 Qt 出售给 Digia 公司，由其继续开发。

组态软件是一款具有人机交互界面，并且还要生成人机交互程序的软件，因此具有强大的 GUI 接口是选择开发工具的重要参考因素。而 Qt 刚好能够满足这样的要求，它提供了丰富的 GUI 接口，并且还支持跨平台工作，能够满足这样的条件归结于它的整个库架构。

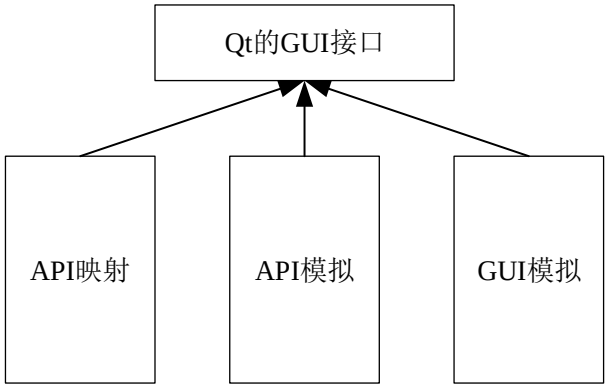

图 2-4 Qt 的 GUI 接口实现策略

如图 2-4 所示，Qt 的 GUI 接口的实现一共有三种类型，正因为这样构造整个

Qt 的 GUI 接口，从而使得通过 Qt 开发的软件只需要很少的适配工作就能在多个平台上进行移植。下面将具体介绍这三种策略[30-31]：

- **API 映射：**所谓映射，就是将操作系统中已经存在的具有同等功能的 API 进行一次封装，使得在不同的平台上具有相同的接口。例如比较常用的两个操作系统平台 Windows 和 Linux，它们都各自提供了自己平台的相关的 GUI 接口，比如划线的 API，但是两个操作系统提供的参数，或 API 名称都不一样，但是功能却是一样的，因此 Qt 对这类 API 进行一次映射，这样就屏蔽了系统差异。
- **API 模拟：**每个系统平台都有自己的独特性，能够提供的功能也不尽相同，因此存在这种可能，在某个平台上具有的 API，在另外一个平台上没有与之功能相对应的 API。这时 Qt 就会利用基本的 API 来模拟出这些功能，从而在多个平台上都能提供相同的接口。
- **GUI 模拟：**GUI 模拟与 API 模拟的差别在于 GUI 模拟提供的是多种 API 共同作用导致的一种结果。也就是说 GUI 模拟提供的是一种比较复杂的功能，例如一些复杂的控件，对话框等。这种模拟也是相当有必要的，比如常用的文件选择框，在有些平台上就没有提供这样的功能，因此通过 GUI 模拟就能够达到提供相同接口的目的。

对于组态软件的开发，提供强大的 GUI 接口是非常必要的，但是也需要很多其他功能的支持，例如网络模块、串口通信模块、蓝牙模块等。Qt 提供了很多额外功能的支持，如 XML、Bluetooth、Serialport、NFC、D-Bus 和 OpenGL 等库，这样既方便了开发者进行项目开发，同时也为开发者省去了移植这些内容的工作。

2.6 本章小结

本章主要介绍了组态软件的整体框架，包含三个部分：开发系统、工程文件和运行系统。分别对三个部分进行了整体的介绍，并且设计了整体的框架，为以后的详细设计以及实现奠定了基础。最后本章还对本文的开发工具 Qt 进行了介绍，详细讲解了为什么选择它进行组态软件开发的理由，同时还介绍了其对跨平台开发支持所采取的一些策略。

第三章 组态软件开发系统的设计与实现

组态软件的开发系统关键模块包含四部分：图元模块、变量配置模块、系统配置模块和查询报警模块。下文将详细论述这四个部分的设计思想，以及实现过程中遇到的关键难点的解决办法。

3.1 图元模块

3.1.1 图元简介

组态软件的开发系统为用户提供了丰富的图形库，利用这些图形能够开发出生动的图形界面。在组态软件中，图形与具体设备是相互对应的，因此每个图形代表着具体的功能，在组态软件开发过程中，往往把功能模块化，这样就形成了一个个图元，因此每个图元都代表着相应的功能，这样既方便扩展也方便维护。

图元一般具有以下属性：

- 图形：即图元在界面上的显示效果，可以是一些基本的图像，如线、三角形、矩形圆等；也可以是一些拟物化的图像，如阀门、开关、电机等。
- 拖拽：由于图元是作为一个独立的控件，因此最终需要在一个画布上显示，在画布上，图元的位置可以由用户随意决定，用户可以通过拖拽图元将其放在适当的位置。
- 缩放：用户可以根据需要将相应的图元进行放大和缩小。
- 属性配置：每个图元都具有自己的属性，用户可以通过属性配置来设置图元的颜色、动画效果以及变量关联信息。

3.1.2 图元的设计与实现

3.1.2.1 图元继承框架设计

上文已经介绍过，图元分为两类：基本图元和拟物化图元。在进行图元设计的时候，需要将两类图元统一起来，方便后续的管理与操作，因此基于 C++语言的多态特性，本文主要采用继承的方式来实现图元[32]。

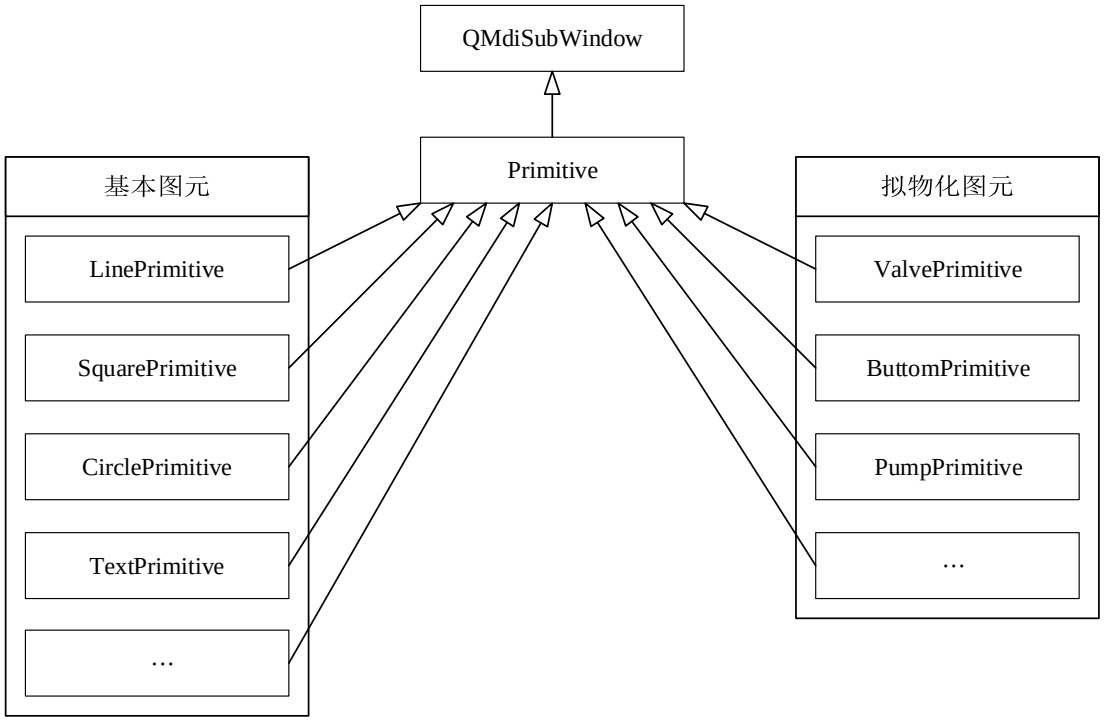

图 3-1 图元的继承框架

如图 3-1 所示，Primitive 类继承于 Qt 框架的 QMdiSubWindow 类，从而使得 Primitive 类具有 QMdiSubWindow 类相关属性和操作，因此只需要对 Primitive 的相关操作进行重载，那么就能实现相关功能。同时可以看出，不管是基本图元还是拟物化图元，都将继承于 Primitive，这就使得在管理这些图元的时候，利用其父类指针就可以设计出统一的接口。

用户对图元的操作主要包含两部分：变形和属性配置。

变形相关的操作，需要借助 Qt 的事件系统来实现。对于具有交互界面的系统来说，往往伴随着鼠标移动，鼠标点击，键盘输入等事件，在 Qt 中这些事件有统一的处理接口。一旦用户通过鼠标或键盘进行了相关操作，系统将会产生一个事件，Qt 的事件系统将会捕捉到该事件，然后将其分发给具有焦点的控件，此后会有事件分发器按照事件的类型将其分配到各自的处理函数中去[33]。因此如果需要改变这些事件的默认处理方法，用户只需要重写相关的事件处理函数即可实现。对于变形操作，涉及到的主要是鼠标的操作，因此需要对鼠标事件的处理函数进行重载，对于本组态软件来说，只需要重写 Primitive 类的相关函数即可，由于继承的关系，所有的图元都将具有相同的特性。

属性配置是用户对图元的个性化设置，由于每个图元具有的属性不同，因此需要通过用户来配置，图元将提供相应的属性配置框，每个图元将记录用户配置的结

果。

图元自身还具有两个关键属性：图形显示和存储。图形显示即图元的显示效果，利用相关的重绘函数实现。对于储存，由于对图元进行操作后，需要将图元的状态记录下来，因此需要有图元存储和读取的相关操作[34]。

3.1.2.2 图元变形

如上文所述，图元变形包括移动和缩放两部分。这两个动作都需要用户通过鼠标来完成，因此将会涉及到与鼠标事件处理相关的函数。

表 3-1 鼠标事件处理函数

处理函数	函数说明
mousePressEvent(QMouseEvent *event)	鼠标任意键按下时调用
mouseReleaseEvent(QMouseEvent *event)	鼠标按键释放时调用
mouseMoveEvent(QMouseEvent *event)	鼠标移动时调用

图元变形中会处理三类鼠标事件，分别是鼠标按键按下、鼠标按键释放和鼠标移动。在 Qt 中，相应的控件都有对应的处理函数，如表 3-1 所示。由于在 Qt 中，大多数与界面显示相关的类都继承于 QWidget 类，而表 3-1 的三个函数在 QWidget 类中是以虚函数的方式实现的，只是实现了一些默认的处理逻辑，因此如果需要对相应的事件进行特殊处理，只需继承相关的类，然后重写这三个函数即可。下文将描述如何利用这三个函数来实现图元的变形。

（1）Qt 的坐标系统[35]

Qt 的坐标系统分为两类：全局坐标和局部坐标。全局坐标即以屏幕的左上角（Windows 平台）为坐标系的原点，局部坐标即以相关控件的左上角为坐标原点，而对于鼠标操作来说，每个鼠标事件将会记录一个全局坐标和局部坐标，全局坐标是唯一的，局部坐标是相对于鼠标光标所在控件的坐标系而言的。

本节主要讨论鼠标事件对于图元变形的作用，如图 3-2 所示，在一次鼠标事件中将会涉及到屏幕、画布和图元。上文已经讨论过不管是全局坐标系还是局部坐标系，坐标原点都在控件的左上角，即图 3-2 中的 A、B 和 C 点。如果用户用鼠标在 D 点进行了点击，Qt 为每个鼠标事件提供了两个接口，globalPos()用于获取鼠标点击点在全局坐标系（即以 A 点为坐标原点的坐标系）中的坐标；pos()函数用于获取点击点在局部坐标系（即以 C 点为坐标原点的坐标系）中的坐标。对于图元来说，也存在全局坐标和局部坐标（相对于以 B 点为原点的画布坐标系），Qt 也为其提供了两个接口来获得这些坐标，通过调用 mapToGlobal(QPoint(0,0))可以将 C 点（即图元坐标系）映射为全局坐标，通过调用 pos()函数可以获得 C 点在画布坐标

系（以 B 点为坐标原点的坐标系）中的坐标，通过这两个坐标可以计算出该图元的父组件（即画布）的全局坐标，利用这个全局坐标，能够保证图元在变形和拖拉过程中不会超出画布的范围。

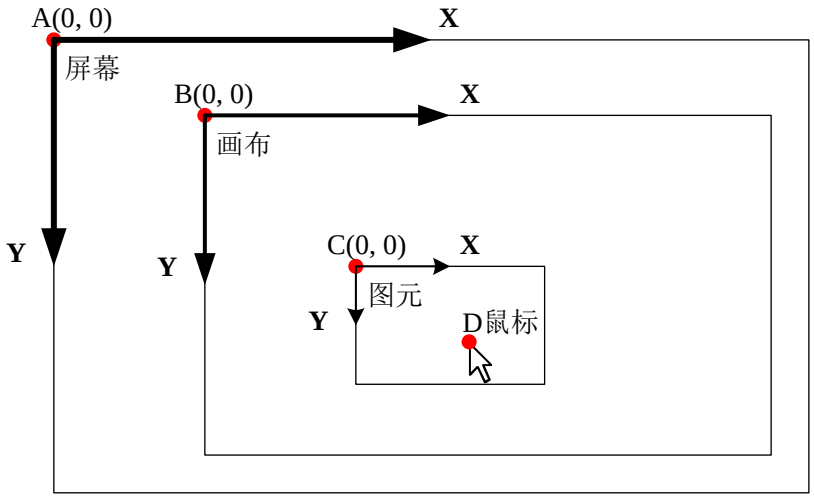

图 3-2 Qt 坐标系统

（2）图元的拖拉和缩放

每个图元都有默认的大小，当其被拖拉到画布上时，在画布上也有默认的位置，但用户可能会根据项目的需要调整图元大小和位置，因此图元的设计中变形是相当重要的一部分。图元的变形分为两种，一是在画布上的位置发生变化，本身的大小和形状并不会发生变化；二是图元的形状发生变化，如拉长、变大等，图元的形状改变一般是通过拖拽图元的某些区域产生的，一般的图形系统中，一个图形可拖拽的区域分为九个部分[36]，如图 3-3 所示。

在 Qt 中只需要确认 A 点的坐标和图形的长(Height)和宽(Width)就可以表示出一个图形，在通过鼠标对图元进行操作的过程中，鼠标只是一个点，因此要确定应该进行哪种操作，需要一个区域来标识。由于鼠标的作用范围应该是一个区域而不是一条线或一个点（这也是方便用户操作），一个图元的边界到鼠标作用区域的边界应该存在一定距离，这也叫做填充区(Padding)。如图 3-3 所示，鼠标的作用区域分为九部分，各部分的作用如表 3-2 所示。

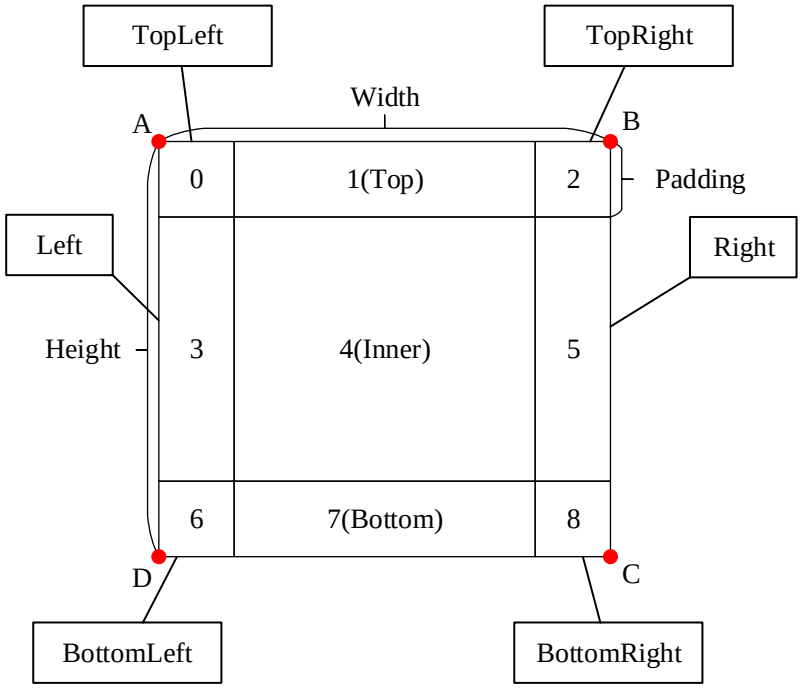

图 3-3 图元拖拽区域分布

表 3-2 图元拖拽区域介绍

编号	区域名称	功能简介	鼠标光标
0	TopLeft	使得图形左上角变形	⤡
1	Top	使得图形上部变形	↕
2	TopRight	使得图形右上角变形	⤢
3	Left	使得图形左边变形	↔
4	Inner	使得图形位置发生变化	✥
5	Right	使得图形右边变形	↔
6	BottomLeft	使得图形左下角变形	⤢
7	Bottom	使得图形底部变形	↕
8	BottomRight	使得图形右下角变形	⤡

在用户通过鼠标，对表 3-2 所述的区域进行操作时，各个区域对应的处理方式如下：

- TopLeft 区域：如图 3-3 所示，此时 D 点将保持不动，A 点将会被重新计算，图元的宽(Width)和高(Height)也将被重新计算。当鼠标移动，系统将利用 A 点坐标、图元宽和高，重新绘制图元，使得图元改变形状。
- Top 区域：如图 3-3 所示，此时 C 点和 D 点将保持不动，A 点坐标将会被重新计算，图元的高(Height)也将会被重新计算。当鼠标移动，系统根据新的 A 点坐标和宽高，重新绘制图元。
- TopRight 区域：如图 3-3 所示，此时 C 点将保持不动，A 点将会被重新计算，图元的宽(Width)和高(Height)也将被重新计算。当鼠标移动，系统根据新的 A 点坐标和宽高，重新绘制图元。
- Left 区域：如图 3-3 所示，此时 B 点和 D 点将保持不动，A 点坐标将会被重新计算，图元的宽(Width)也将会被重新计算。当鼠标移动，系统根据新的 A 点坐标和宽高，重新绘制图元。
- Inner 区域：如图 3-3 所示，如果鼠标是在这个区域，代表着操作者是想改变图元在画布中的位置，而图元的形状将保持不变。
- Right 区域：如图 3-3 所示，此时 A 点和 C 点将保持不动，图元的宽(Width)也将会被重新计算。当鼠标移动，系统根据新的宽，重新绘制图元。
- BottomLeft 区域：如图 3-3 所示，此时 B 点将保持不动，A 点将会被重新计算，图元的宽(Width)和高(Height)也将被重新计算。当鼠标移动，系统将利用 A 点坐标、图元宽和高，重新绘制图元。
- Bottom 区域：如图 3-3 所示，此时 A 点和 B 点将保持不动，图元的高(Height)将会被重新计算。当鼠标移动，系统根据新的高，重新绘制图元。
- BottomRight 区域：如图 3-3 所示，此时 A 点将保持不动，图元的宽(Width)和高(Height)将被重新计算。当鼠标移动，系统将利用图元宽和高，重新绘制图元，使得图元改变形状。

如前文所述，图元的变形主要涉及到鼠标的三个动作：鼠标按键按下、鼠标移动和鼠标按键释放。Qt 对于这些鼠标事件的处理分别委托给三个函数：mousePressEvent、mouseMoveEvent 和 mouseReleaseEvent。通过重写这三个函数，可以实现图元的变形。图元的变形处理流程如图 3-4 所示，可以看出主要分为三个部分：拖拽区域判断、边界判断和位置计算。

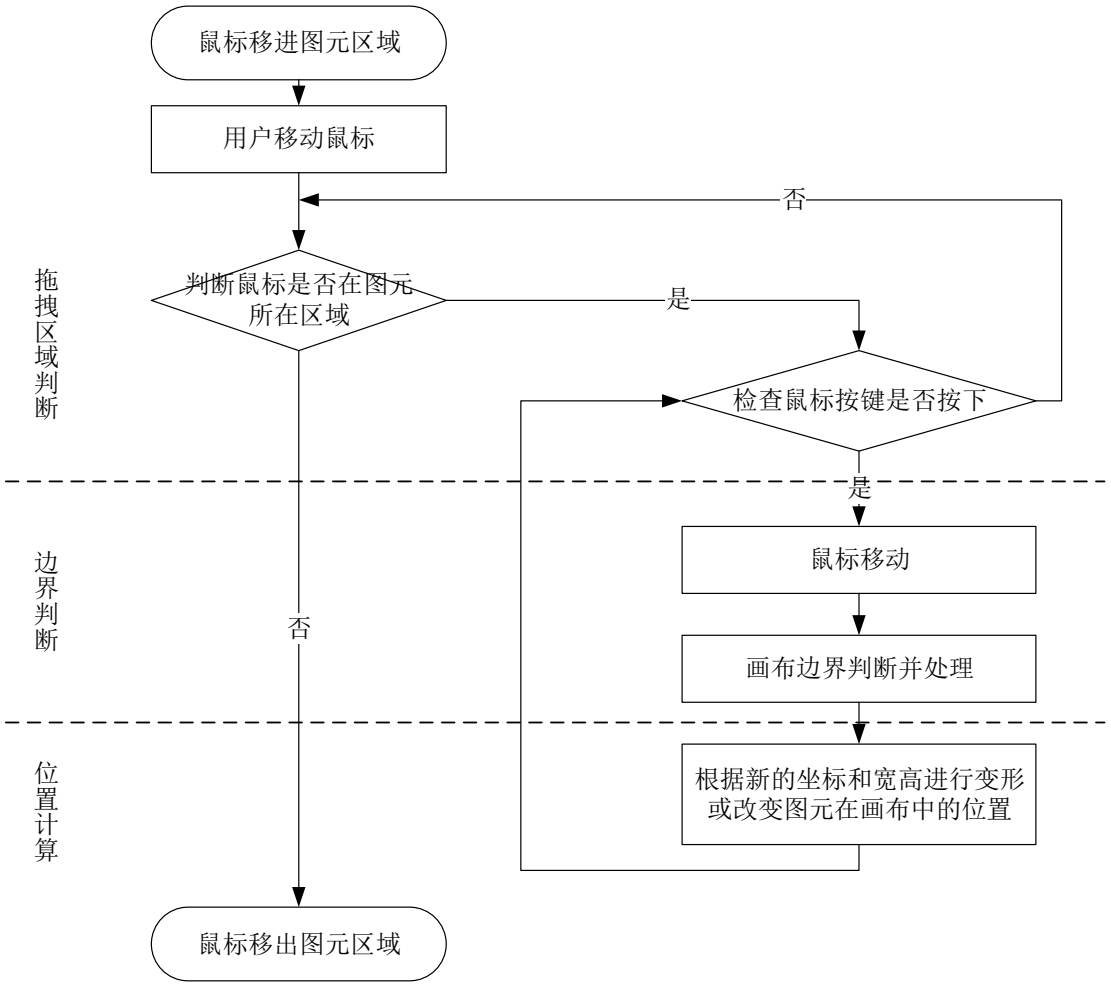

图 3-4 图元变形处理流程

- 拖拽区域判断：用户可以随意地移动鼠标，但只有当鼠标光标移进图元区域的时候才会触发图元相应的鼠标事件处理函数。当鼠标只是移动时，将只会调用 mouseMoveEvent 函数，在这个函数中将会根据图 3-3 所介绍的拖拽区域进行判断，记录目前鼠标正处在哪个区域，并将光标变化成表 3-2 中对应区域的鼠标光标形状（如鼠标在 Left 区域，将会显示↔图标），这样可以提醒用户此时正在对哪个区域进行操作。
- 边界判断：当用户在图元区域内按下鼠标左键，此时再移动鼠标，说明用户正在对图元进行变形，但由于图元是处在画布中的，而画布的大小是固定的，因此为了防止图元被移出画布边界，需要对移动的鼠标位置进行计算判断，如果在允许范围内将不做任何修改，如果超出范围将对鼠标位置进行修改。由于边界判断问题比较复杂，因此下部分将会进行详细讲解。
- 位置计算：一旦用户按下鼠标左键并且移动鼠标，就会在 mouseMoveEvent 函

数中重新计算图元的 A 点（如图 3-3 所示）和图元的宽高，并且对图元进行重绘。由于 A 点坐标是在画布坐标系中（如图 3-2 所示），而鼠标事件记录的位置在全局坐标中（即屏幕坐标，如图 3-2 所示），因此需要通过计算图元的 A 点的全局坐标与鼠标事件的位置坐标之间的相对距离，然后再作用到 A 点上（画布坐标系中），这样才能完成图元的变形。

上述过程的核心代码如下所示：

```
//拖拽区域判断
//获得当前鼠标所在图元的位置信息
QRect cur_primitive = this->rect();
//将图元坐标转换为全局坐标
QPoint cur_tl = mapToGlobal(cur_primitive.topLeft());
//获得鼠标当前的全局坐标
QPoint mouse_position = event->globalPos();

//region 0(top left)，处理 topleft 区域，其他区域处理方式相同
if (mouse_position.x() >= cur_tl.x() &&
                mouse_position.x() <= cur_tl.x() + PADDING &&
                mouse_position.y() >= cur_tl.y() &&
                mouse_position.y() <= cur_tl.y() + PADDING)
//进行区域判断，根据鼠标坐标与图元的左上角坐标判断
{
    direction = TopLeft; //记录鼠标区域位置
    this->setCursor(Qt::SizeFDiagCursor);//设置光标形状
    return;
}
```

```
//位置计算
switch (direction) {
//重新计算位置
    case TopLeft:
        if (br.x() - mouse_position.x() >= this->minimumWidth() &&
            br.y()-mouse_position.y()>= this->minimumHeight() &&
            br.x() - mouse_position.x() <= this->maximumWidth() &&
            br.y() - mouse_position.y() <= this->maximumHeight())
          //限制图元的最大最小形状
        {
          //重新计算 x 和 y 值，重新计算宽高
            int distance_x = mouse_position.x() - tl.x();
            int distance_y = mouse_position.y() - tl.y();
            adjust_tl.setX(adjust_tl.x() + distance_x);
            width -= distance_x;
            adjust_tl.setY(adjust_tl.y() + distance_y);
            height -= distance_y;
        }
        break;
//其他区域计算方式类似
 .......
}
```

（3）边界检查

上一部分讨论过，当用户对图元进行变形的时候，可能会使得图元超出画布的范围，因此在进行这些操作的时候，需要对移动变形的位置进行判断，这样让图元能够一直保持在画布上。

有两种情况使得图元可能超出画布的范围：一是移动图元；二是对图元进行变形或缩放。不管是哪种情况，对图元的改变都是通过计算鼠标位置的两次相对位置（鼠标按键按下时的位置和移动后的位置）而确定的，因此在进行边界检查的时候，也只需要对这两个鼠标位置进行处理。

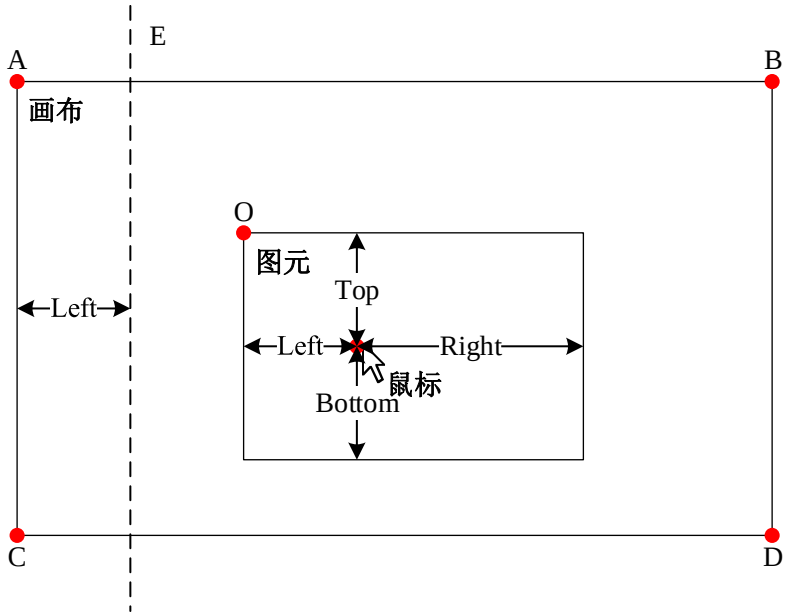

图 3-5 边界检查示意图

为了保证图元无论进行怎样的变形或移动都要处在画布中，因此需要弄清引起变形的原因。上文已经讨论过，鼠标事件是导致图元变形的原因，变形的各种参数是通过计算两次鼠标的相对位置而确定的，即鼠标按键按下时的位置和鼠标移动后的位置。对于鼠标按键按下时的位置是固定的（这取决于用户的操作，因为只要用户是在对当前画布的图元进行操作，这时的鼠标位置就是合法的），从而可知只有鼠标移动时的位置是不确定的，这个位置可能就会超出画布的范围，因此通过修正这个位置就可以保证图元不会超出画布范围。

对于图元的变形，用户能够操作的区域有 8 个（除了 Inner 区域，如图 3-3 所示）。此时鼠标只是作用在四条边和四个顶点上，当拖动鼠标，也就是这些边和这些点会移动。只要保证移动的位置在画布范围内，那么图元的变形将不会超过画布范围。在 Qt 中，由于通过左上角和右下角的坐标就能够确定一个图形，因此如图 3-5，只要使得鼠标坐标的 X 坐标介于 A 点的 X 坐标与 D 点的 X 坐标之间，鼠标坐标的 Y 坐标介于 A 点的 Y 坐标与 D 点的 Y 坐标之间，就能保证图元在画布中。

对于图元的移动，情况就比较复杂。图元的移动表示图元的形状不会改变，只是图元在画布上的位置发生变化。以图元向左移动为例，首先用户的操作是点击需要移动的图元，这时可以得到四个参数：点击点距离图元的四边的距离。如图 3-5 所示，距离上边的距离为 Top，距离下边的距离为 Bottom，距离左边的距离为 Left，距离右边的距离为 Right。如果用户往左移动意味着鼠标位置不能超过图 3-5 所示的虚线所标识的 E 边，也就是说画布的 AB 边到 E 边的距离应该为 Left，这样就

能保证图元能够在画布内完全显示。以此类推，图元的其他三个方向的移动都应该满足这样的条件。当鼠标移动导致位置超出这些范围的时候，只需将鼠标位置进行修改即可，继续上面的例子，如果图元左移超过了 E 边，那么只需将鼠标坐标的 X 坐标修正为 E 边的 X 坐标，然后再进行图元位置的计算，最后重绘图元，此时的图元一定在画布内。

图元边界位置判断的核心代码如下：

```
//获得画布的位置信息
QPoint parent_tl = mapToGlobal(QPoint(0, 0)) - this->pos();
QPoint parent_br;
parent_br.setX(parent_tl.x() + this->parentWidget()->width());
parent_br.setY(parent_tl.y() + this->parentWidget()->height());

//获得鼠标的位置信息
QPoint global_point = event->globalPos();
/*鼠标按下时会确定 top、bottom、left、right 的值，如图 3-5 所示，
如果是变形，那么将清除这四个值；如果是移动，将保留这四个值作
为以后的判断参数*/
if (direction != Inner)
    top = bottom = left = right = 0;
//鼠标往左边移动判断
if (global_point.x()    < parent_tl.x() + left)
    mouse_position.setX(parent_tl.x() + left);
//鼠标往右边移动判断
if (global_point.x()    > parent_br.x() - right)
    mouse_position.setX(parent_br.x() - right);
//鼠标往上移动判断
if (global_point.y() < parent_tl.y() + top)
    mouse_position.setY(parent_tl.y() + top);
//鼠标往下移动判断
if (global_point.y() > parent_br.y() - bottom)
    mouse_position.setY(parent_br.y() - bottom);
```

3.2.2.3 图元的图形显示与属性配置

每个图元除了一些共有的功能外，还具有两个重要属性：图形显示和属性配置。

图形显示即图元呈现给用户的外观形状，如直线图元呈现给用户的就是一条直线，矩形图元呈现给用户的就是一个矩形。

属性配置则包含两部分：图元自身相关参数的设置和图元的动画连接。自身相关参数包括颜色，是否透明和背景色等；动画连接即图元在整个画面中的运动情况，如图元根据相关的输入值，在画面中做出相应的动作[37]。

（1）图元的图形显示

图元主要分为两个大类：基本图元和拟物化图元。其中基本图元如图 3-6 所示，主要包含一些常见的二维图形（三角形和多边形等）以及三维图形（正方体和圆柱体等）。这些图形通常可以有两种呈现方式：带填充色的和不带填充色的。

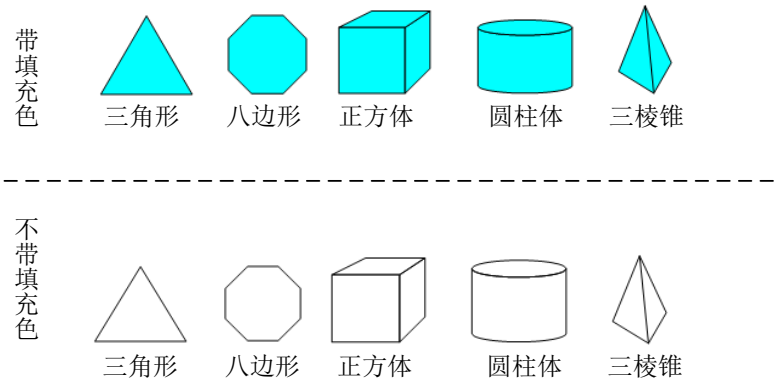

图 3-6 基本图元

拟物化图元指的是图元所呈现出来的图像和现实生活中实际存在的事物很相像，利用这些图元可以模拟现实生活中的相关设备。具体的拟物化图元如图 3-7 所示，包含气泵、锅炉、房屋和传送带等。

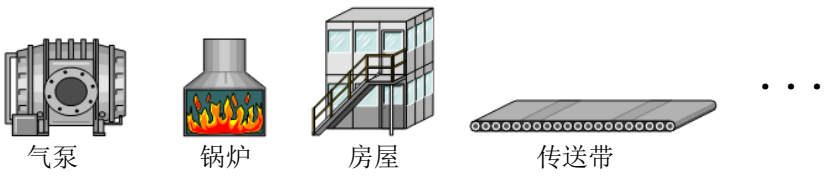

图 3-7 拟物化图元

在 Qt 中，如果要改变显示类的显示效果需要重写 paintEvent 函数，本文采用的也是这种方法，从而实现不同图元的差异显示。在计算机图形中有两种图形表示方式：位图和矢量图[38]。

- 位图：位图是由像素点构成的，每个像素点有对应的存储空间，这些存储空间可以记录图像的颜色、饱和度和明度。因此对于一张图片来说，一旦图片的效果确定下来了，那么这张图片的像素点的个数以及每个像素点所带的信息也就确定下来了。这时如果要改变图形的形状，那么就会导致图形的显示失真。
- 矢量图：矢量图是由多个基本对象组合而成的，而每个基本对象又是通过相关的绘制函数实现的，因此只要提供相应的绘制参数，就能够绘制出各个对象，从而组合出复杂的对象。由于矢量图是通过数学函数绘制的，在变形的过程中只会改变传入的参数信息，因此不会使得图形失真。

根据上面对位图和矢量图的分析，再结合图元的基本特性（图元可以根据用户的需要进行缩放和变形）。如果要得到良好的显示效果就应该选择使用矢量图来设计图元。SVG(Scalable Vector Graphics)是通用的矢量图格式[39]，通过将矢量图的相关参数信息以 XML(Extensible Markup Language)的格式存储起来，从而方便传输和解析，在 Qt 中有专门的类进行处理。

图元图形显示的设计主要分为三步：

- 第一步：使图元继承 Primitive 类，从而使得图元具有变形的相关特性。
- 第二步：选择相应的图片，如图 3-6 和 3-7 所展示的图片，利用 QSvgRenderer 类进行处理，其构造函数为 QSvgRenderer(const QString & filename)，其中 filename 参数表示的是图元的存储位置。通过 defaultSize()方法可以获得矢量图的默认显示尺寸，利用 resize()函数可以重置图元的大小，从而得到默认的图元显示。
- 第三步：重写 paintEvent 函数，新建一个 QPainter 实例，然后设置相关属性，调用 QSvgRenderer 类的 render()函数，并将 QPainter 实例以参数的形式传入，从而完成图元的图形显示。

（2）图元的属性配置

图元的属性包含两部分：自身相关参数和动画连接。自身相关参数包含颜色、是否透明、背景色和大小四部分。而动画连接则包含是否移动、输入输出值、滑动杆参数和特殊情况。其中是否移动表示的是在画面显示中，图元的位置是否会发生变化，如果需要发生变化，则需要配置相关参数；输入输出值表示图元是否会接收外部的输入值或向外部输出数值；滑动杆参数表示的是某些图元具有可移动的滑动杆，因此需要对滑动杆运动的参数进行设置；特殊情况表示的是图元可能会闪烁显示或隐藏[40]。

由于每个图元具有自己的独特性，因此不一定所有的图元都要进行上述讨论的设置，但是考虑到代码的复用性，因此对于配置对话框的设计将采用如下策略：

为属性配置对话框设计一个类 AttributeSetDialog，该类中将提供所有的属性配置选项；由于该类提供的是所有的属性配置选项，而有些图元的某些选项是不能进行设置的（如有些图元不能接受输入值属性配置），因此为了能够适应不同图元的需要，AttributeSetDialog 类将提供相应接口使得某些配置选项不能操作[41]。根据上述讨论给出如下具体实现。

属性配置对话框如图 3-8 所示，可以看出主要分为三部分：对象描述、基本属性和动画连接。其中对象描述中有两个信息需要用户输入，对象名称和提示文本，对象名称指的是用户为这个图元命的名，而提示文本表示的是当鼠标光标处在图元所在区域时出现的提示信息，特别需要注意的是对象名称在整个工程项目中是唯一的。

图 3-8 属性配置对话框

基本属性包含五个参数：图形颜色、背景颜色、图元的长、图元的宽和图元背景是否透明。用户可以通过点击工具按钮，在弹出的调色板中选择相应颜色，从而设置图形颜色和背景颜色。除了通过拖拽的方式改变图元大小外，用户还可以在基本属性中直接对图元的长宽进行设置，从而改变其大小。对于透明指的是图元的背景是否透明，对比效果如图 3-9 所示。

不透明　　透明

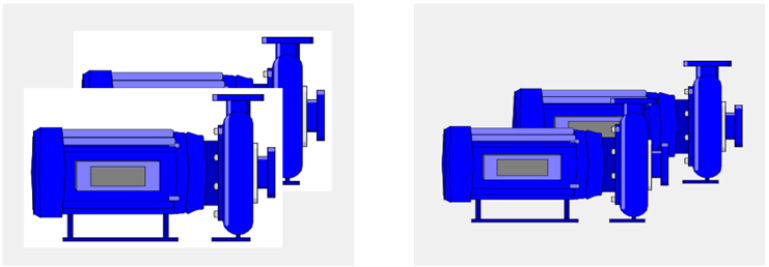

图 3-9 透明效果对比

动画连接包含四个部分：移动、输入输出值、滑动杆输入和特殊。每个图元可以根据自身需求对这几个部分设置使能还是不使能。其中移动这一块表示的是图元能不能够在画面中移动，当复选框选中时，这时移动参数才可以进行设置，用户需要对移动的起点和移动的终点坐标进行设置，从而限制了图元移动的范围，其次用户还需要将移动的条件与特定变量进行关联，当变量发生变化时，系统会根据变量的值来决定图元移动的距离。输入输出部分表示的是图元与外界交流的数据，用户可以规定数据的最大最小值，并且用户需要将输入或输出同变量进行关联。滑动杆输入是针对一些特殊的图元（如温度计等），这些图元有一根滑动杆可以指示数值，因此需要外界数据输入促使其变化，用户通过关联相应变量来达到目的，同时用户还可以设置滑动杆的变化范围。特殊部分表示图元在显示过程中是否会发生变化，可以设置三种变化：闪烁、隐藏和流动。用户通过关联相关变量，当变量满足某个条件时触发相应效果，通过下拉框可以选择比较条件。

下面将详细介绍 AttributeSetDialog 类的相关属性和方法，首先需要明确的一点是，AttributeSetDialog 类的主要目的是为图元服务，同时也只有图元会使用它，而且每个图元必须有一个 AttributeSetDialog 类的私有实例，所以为了方便数据的访问，AttributeSetDialog 类的成员变量都是公有的，但由于图元中的 AttributeSetDialog 实例是私有的，因此其成员变量也只能由相关图元才能访问，这也达到了数据封装的效果。AttributeSetDialog 类也提供了一些公共方法，使得图元类可以按照自己的需要选择能够使用的属性配置[42]。AttributeSetDialog 类详细属性和方法介绍如表 3-3 所示。

表 3-3 AttributeSetDialog 类属性和方法介绍

AttributeSetDialog 类属性		
属性名	**属性类型**	**描述**
name	QString	对象名称

tips	QString	提示文本
color	QColor	图形颜色
background_color	QColor	背景颜色
width	int	长
height	int	宽
is_transparent	bool	true 为透明，false 为不透明
is_movable	bool	true 为可移动，false 为不能移动
move_start	QPoint	移动位置的起点
move_end	QPoint	移动位置的终点
move_variant	QString	移动关联变量
input_out_flag	int	0 表示输入，1 表示输出
input_out_max	int	输入输出的最大值
input_out_min	int	输入输出的最小值
input_out_variant	QString	输入输出关联变量
…	…	…
AttributeSetDialog 类方法		
方法	**描述**	
void setInputOutDisabled()	使得输入输出值部分无效	
void setSliderInputDisabled()	使得滑动杆输入部分无效	
void setSpecialDisabled()	使得特殊部分无效	
void setBlinkDisabled()	使得闪烁功能无效	
void setHiddenDisabled()	使得隐藏功能无效	
void setFlowDisabled()	使得流动功能无效	

3.1.2 图元管理器的设计与实现

图元管理器的功能主要分为两部分：将图元进行分类管理和方便用户添加图元到画布中。

为了使得图元能够与整个系统的耦合度降低，因此图元一般以动态链接库的形式存在，这样就可以保证能够更新图元却不改变整个系统的框架。对于图元的管理需要配置一张图元信息表，格式为序号 标识字符串，序号和标识字符串在整个图元库中都是唯一的，标识字符串是作为索引动态链接库中相关图元的字符串，格式为 get+图元名。因此在设计动态链接库的时候，相应的函数设计将满足下列规则，以添加 CubePrimitive 图元为例，

第一步：添加获取图元函数

```
//返回图元基类的指针，方便统一操作
//函数名格式：get+图元名
//传入参数为父窗口，可以是画布或运行界面（运行系统中）
Primitive* getCubePrimitive(QWidget *parent)
{
    return new CubePrimitive(parent);
}
```

第二步：修改图元信息表

```
//并且在图元信息表中添加一条信息
1 getCubePrimitive
//即满足格式：序号  标识字符串
```

通过这种方法，在图元库新添图元的时候，添加这样一个函数并且修改图元信息表，就能够让整个系统识别到添加的图元。

图元管理器的一大功能就是对图元进行分类管理，通过下面这个枚举的定义可以看出图元的分类，

```
typedef enum{
    BasicShape,   //一些基本的图元
    PushButton,   //按钮形状的图元
    Arrow,   //箭头形状的图元
    Pipe,   //各类管状的图元
    Blower,   //鼓风机类的图元
    Boiler,   //锅炉类的图元
    Pump,   //泵形状的图元
    Controller,   //各类控制器形状的图元
    Motor,   //电机形状的图元
    Tank,   //各种罐子形状的图元
    Other,   //未分类的图元
    PrimitiveTypeSize   //图元类型的总数
}PrimitiveType;
```

每个图元的类型都是由图元设计者确定的，可以通过图元的 getType()方法获得。在为用户展现图元的时候用到了 Qt 的 QListWidget 控件，通过这个控件能够将同一类图元以列表的形式呈现出来。而此控件是通过添加 QListWidgetItem 来添

加数据项的，因此为了能够让图元控制自己的展示效果，每个图元都有getListWidgetItem 方法，其函数实现细节如下：

```
QListWidgetItem* getListWidgetItem()
{
    QListWidgetItem *item = new QListWidgetItem;
    item->setIcon(QIcon(“图片路径”));
    item->setText(QString(“图元名称”));

    return item;
}
```

该函数设置了图元在列表中的显示名称以及显示的图标。

上面讨论了图元管理器的主要实现策略，下面将详述 PrimitiveManager 类的实现细节。

PrimitiveManager 类具有两个重要属性：

1) QMap<int, Primitive *> primitive_set：该成员变量用来存储图元库，以图元序号作为关键字，图元实例指针作为值。
2) 各类图元的 QListWidget 实例：如 Pump 图元的 QListWidget 实例为pump_listwidget。通过这些实例可以对图元进行分类。

PrimitiveManager 类的重要方法：

1) QListWidget* getListWidgetByType(const PrimitiveType &type):该方法根据 type 值向调用者返回匹配的 QListWidget 实例，如 type 等于 Pump，那么就会返回pump_listwidget，然后调用者就可以根据返回值显示相应的图元列表。
2) Primitive* getPrimitiveById(const int &id):该方法根据 id 值，即图元的序号，向调用者返回相应的图元指针。该函数将 id 作为索引查找 primitive_set 结构，返回一个图元指针，然后调用图元的 clone()方法得到复制图元，再将复制图元的指针返回给调用者。
3) void init(const QString &filename, QWidget *parent):该函数主要完成加载图元库，然后完成 PrimitiveManager 类的相关结构的初始化。

由于图元管理器的初始化流程比较复杂，下面将进行详细介绍，如图 3-10 所示，初始化流程主要分为三个部分：对图元信息表的处理、动态链接库的加载以及提取图元并对其进行相应处理。其关键代码如下：

（1）对图元信息表的处理

```
//调用 checkPrimitiveInfoTable 函数，将信息存储在一个 map 中
//判断校验结果
QMap<int, QString> primitive_info = checkPrimitiveInfoTable();
if (primitive_info.isEmpty())
    return;
```

（2）加载动态链接库

```
//利用 Qt 的 QLibrary 类对动态链接库进行处理
//filename 为动态链接库的路径
QLibrary primitive_lib(filename);
```

（3）提取图元并进行相应处理

```
//定义提取函数的类型
typedef Primitive* (*GetPrimitive) (QWidget *);
//迭代处理所有图元
QMapIterator<int, QString> it(primitive_info);
while (it.hasNext())
{
    it.next();
    int id = it.key();
    QString func_name = it.value();

    //提取函数并设置 primitive_set 数据结构
    GetPrimitive func = primitive_lib.resolve(func_name);
    if (!func) continue;
    primitive_set[id] = func(parent);

    //根据图元类型设置对应的 ListWidget
    Primitive* pri = primitive_set[id];
    PrimitiveType type = pri->getType();
    QListWidgetItem *item = pri->getListWidgetItem();

     //记录图元 ID，方便提取
    item->setWhatsThis(QString(%1).arg(id));
    setListWidget(item, type);
}
```

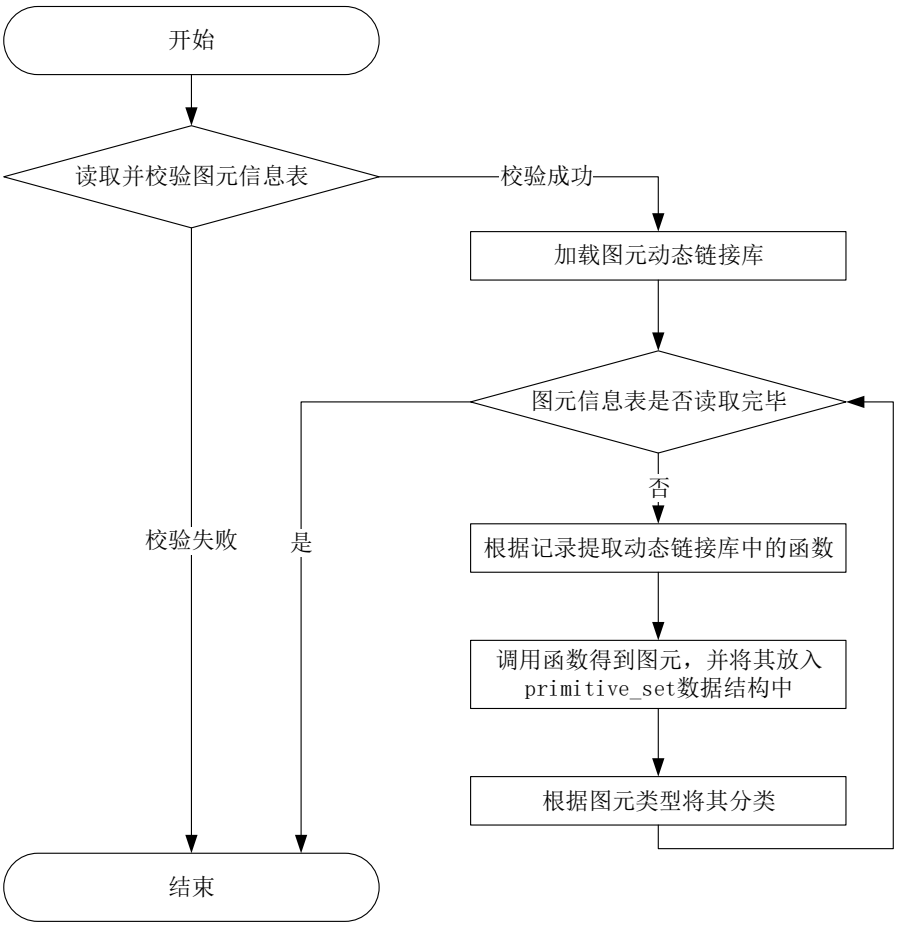

图 3-10 图元管理器初始化流程

3.1.4 画布管理器的设计与实现

画布管理器需要时刻记录画布的状态和用户的操作。记录画布的状态即记录用户添加到画布上的所有图元，记录用户的操作是为了让用户能够撤销或重做某些步骤，防止误操作带来的影响。为此需要设计一个类来进行管理，所以下面将描述 CanvosManager 类的实现。

CanvosManager 类利用 QMap<QString, Primitive*> canvos 数据结构来存储图元，其关键字是图元的名称，即属性配置对话框里的对象名称，通过图元的 getObjectName()方法获得，对应的值则是所添加图元的指针。同时为了保证对象名称的唯一性，CanvosManager 类会提供一个公有方法来判断对象名称是否已经存在，即 bool isObjectNameValid(const QString &name)，图元在设置属性的时候通过调用该函数判断用户填写的对象名称是否有效，如果有效该函数将会返回 true，否则返回 false。

用户能够向画布中添加图元，当然也能够从画布中删除图元。为此

CanvosManager 类提供了 deletePrimitiveFromCanvos()方法，利用该方法可以将指定图元从 canvos 数据结构中删除，然后画布重绘图元，被删除图元将从画布上消失。

CanvosManager 类同时还需要记录用户对图元的操作，因此需要管理两个栈：撤销栈和重做栈。其示意图如 3-11 所示，

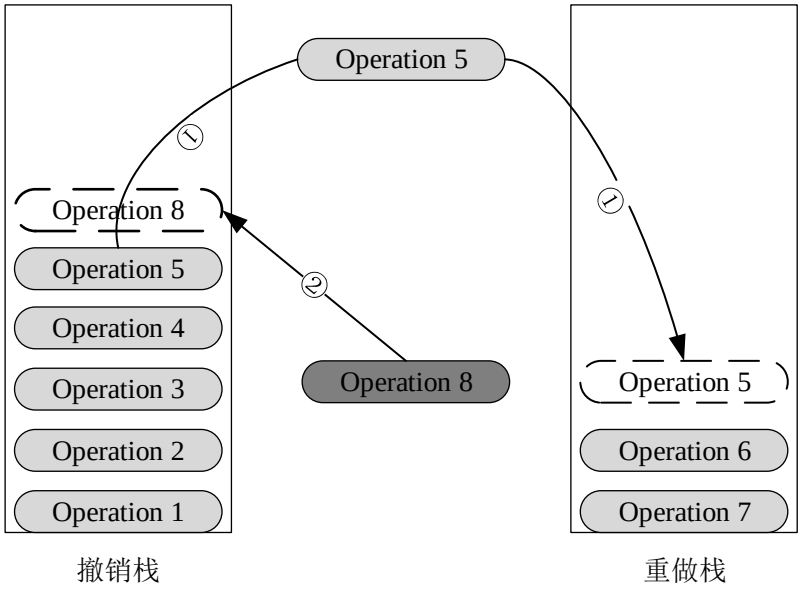

图 3-11 撤销重做示意图

当用户对画布中的图元操作时，每一步操作都将记录在撤销栈中，假设用户已经做了七步操作，那么此时撤销栈中应该有七次操作，这时用户如果撤销了两步操作，这两步操作将会被放入重做栈中，图 3-11 描述的就是这种情况，在撤销栈中有 Operation1-Operation5，重做栈中有 Operation7 和 Operation6。下面将具体描述撤销和重做动作对两个栈的影响。

图 3-11 中的①路径代表的是撤销动作，假如现在用户希望撤销 Operation5，当执行撤销动作时，CanvosManager 类会将撤销栈中的 Operation5 弹出，然后将其压入重做栈中，重做动作即是其逆过程。

图 3-11 中的②路径代表的是用户执行了一次新操作，假如用户执行了 Operation8 这个新操作，一旦这个操作完成，那么就会将其压入撤销栈中，同时 CanvosManager 类将会清空重做栈。

在 CanvosManager 类中，是通过两个 QList 来实现撤销栈和重做栈的[43]，并且提供了三个方法来管理它们：

(1) void insertOperation(Primitive *primitive)：这个函数完成新操作的插入，参数

primitive 即是正在操作的图元的克隆。

(2) Void undoOperation()：这个函数完成撤销动作，该函数会从撤销栈中弹出一个操作项，然后将其放入 canvos 数据结构中，并且让画布重绘所有图元。放入 canvos 数据结构中时存在两种情况，一是 canvos 中已经有相同对象名称的图元，因此会先删除 canvos 中的数据，然后再将其放入 canvos 中；二是 canvos 中不存在相同对象名称的图元（即代表这一步操作是由删除动作造成的），那么就直接放入 canvos 中。

(3) Void redoOperation()：这个函数完成重做动作，重做动作能够实施的前提是重做栈中有数据项存在。调用该函数将会弹出重做栈的一个数据项，然后将其放入 canvos 数据结构中，处理同 undoOperation 函数，并且还需要将数据项放入撤销栈中。

3.2 变量配置模块

组态软件中，变量是数据的抽象形式，它和设备产生的数据是一一对应关系，配置变量也就是在告诉系统应该如何解析数据，因此组态软件中有相应的变量配置模块。

3.2.1 普通变量的设计与实现

普通变量即组态软件中最常见的变量，通常一个变量就代表一个数据采集点。对于普通变量的配置主要分为两部分：基本属性和报警参数。

基本属性包含变量名、变量的描述、变量的类型以及变量的最大最小值。变量名在工程项目中是作为系统识别变量的唯一标识；变量的描述，对变量功能的简要说明；变量的类型，每一个变量都具有相应的类型（如整形、浮点型、布尔型等）；变量的最大最小值，通过配置变量的最大最小值可以确定该变量的取值范围，当获取的数据值不在这个范围的时候，系统可以进行正确地处理[44]。

报警参数包含对该变量是否需要报警以及如果需要报警，报警级别的配置。由于变量是实际生产数据的反应，因此当数据值达到某些异常的范围的时候，需要系统告知观察员，当然用户也可以为不同的变量设置不同的报警优先级，优先级越高的报警将会最先显示在监控界面上。

图 3-12 展示了变量属性配置的对话框，从图中可看出属性配置分为两部分，但只有当基本属性中的报警复选框被选中时，用户才可以设置报警参数相关的参数。对于报警参数，需要配置报警名称以及报警优先级，报警优先级数值越高表示优先级越高；报警限总共分为四个层次：低低、低、高和高高。当用户激活某一层

次报警限时，将同时需要设置界限值和报警文本。报警文本是当报警产生时显示给用户的文本，如发生“低低层次”的报警时，如果报警文本填写的是“低低”，则展现给用户的文本也是“低低”，这样用户就可以知道产生的是“低低层次”的报警[45]。界限值的含义如下：

- 低低：当变量值低于此界限值时，将会产生“低低层次”报警。
- 低：当变量值低于此界限值，但高于“低低层次”界限值时，将会产生“低层次”报警。
- 高：当变量值高于此界限值，但低于“高高层次”界限值时，将会产生“高层次”报警。
- 高高：当变量值高于此界限值时，将会产生“高高层次”报警。

图 3-12 变量属性配置对话框

本文设计了一个类来对配置数据进行管理，即 VariantSetDialog。对于用户添加的每一个变量都将对应一个 VariantSetDialog 类的实例，用户可以访问其相关的属性得到变量的配置信息。VariantSetDialog 类所具有的属性如表 3-4 所示。

表 3-4 VariantSetDialog 类属性

属性	类型	描述
variant_name	QString	变量名称
variant_type	VariantType	变量类型
value_max	QVariant	变量最大值
value_min	QVariant	变量最小值

info	QString	变量描述
alarm_flag	bool	是否报警，true 为报警，false 为不需要报警
alarm_name	QString	报警名称
alarm _priority	int	报警优先级
alarm _level_ll	bool	true 为选中“低低”层次报警，false 则不选中
alarm _level_ll_limit	QVariant	“低低”报警层次的界限值
alarm _level_ll_text	QString	“低低”报警层次的报警提示文本
alarm _level_l	bool	true 为选中“低”层次报警，false 则不选中
alarm _level_l_limit	QVariant	“低”报警层次的界限值
alarm _level_l_text	QString	“低”报警层次的报警提示文本
alarm _level_h	bool	true 为选中“高”层次报警，false 则不选中
alarm _level_h_limit	QVariant	“高”报警层次的界限值
alarm _level_h_text	QString	“高”报警层次的报警提示文本
alarm _level_hh	bool	true 为选中“高高”层次报警，false 则不选中
alarm_level_hh_limit	QVariant	“高高”报警层次的界限值
alarm _level_hh_text	QString	“高高”报警层次的报警提示文本

3.2.2 结构变量

结构变量是通过将多个普通变量组合起来，从而达到统一管理多个变量的目的。在实际生产中，可能会存在有多个数据是相互关联的，因此将这些数据用一个结构变量来管理，这样就可以使得用户的每一次操作都是针对这些数据的。

由于结构变量是由多个普通变量组成的，因此当要创建一个结构变量的时候，需要有相应的普通变量存在。所以即使用户只想创建一个结构变量，还是需要先创建相应的普通变量，并且为其配置好相应参数。

结构变量由如下成分组成：

- 结构变量名称：用户为该结构变量命的名，在整个工程中是唯一的。
- 描述：对该结构变量功能的简要描述。
- 普通变量个数：记录该结构变量包含的普通变量的个数。
- 普通变量的索引数组：该数组记录了该结构变量包含的所有普通变量的索引。

3.3 系统配置模块

组态软件的系统配置模块主要分为三部分：监控站点配置、驱动配置和通信方式配置。这三项配置都是与数据相关的，监控站点配置主要是完成下位机上传到上位机的数据的解析；驱动配置主要是为每一个设备分配相应的驱动进行数据的收发；通信方式配置可以为系统提供每个设备与监控系统的通信方法。

3.3.1 监控站点配置

组态软件中设备与系统进行通信时，数据都是具有统一格式的，只有设备按照规定格式发送数据，系统才能解析，因此在描述监控站点配置之前，首先得介绍本文描述的组态软件的通信数据格式。

如图 3-13，Device 代表着一个监控站点，该监控站点可能负责多个设备的信息采集，即对应的 Packet；每个设备可能又会产生多条数据，即 Item。因此对于一个监控站点来说可能会向系统发送多种 Packet，每个 Packet 可能又包含多个 Item。

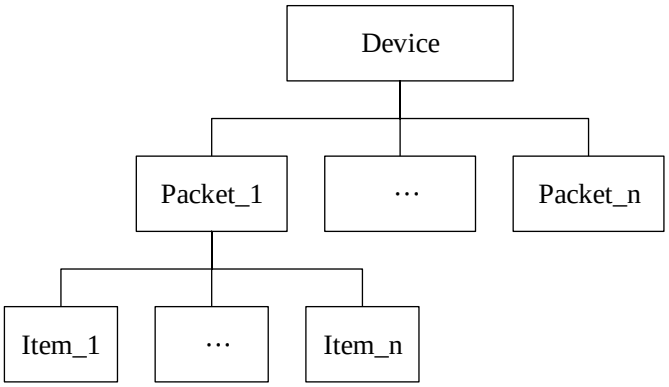

图 3-13 数据组织层次

Packet 数据包具有如下格式：

Packet ID	Item ID	Data Length	Data	⋯	⋯	Item ID	Data Length	Data

- Packet ID：区分不同设备传来的数据
- Item ID：由于每个 Packet 可以配置多个 Item，因此需要标识符进行区别。
- Data Length：数据的长度，方便系统提取有效数据。
- Data：实际的生产数据。

本文利用 DataDevice、DataPacket 和 DataItem 三个类来实现监控站点的配置，下面将描述这三个类的相关接口。

（1）DataDevice 类的接口

- 函数原型：DataDevice(const int &device_id)
  描述：这是 DataDevice 类的构造函数，device_id 作为设备的唯一标识。
- 函数原型：void setDeviceName(const QString &device_name)
  描述：为创建的设备设置一个名字。
- 函数原型：void addPacket(const int &packet_id, const QString &packet_name)
  描述：向 Device 中添加一个 DataPacket 实例，id 即所添加的 DataPacket 实例的标识符，name 即所添加的 DataPacket 的名字。

- 函数原型：void addItem(const int &packet_id, const int &item_id, cosnt QString &item_name)
  描述：向 Device 中指定的 Packet 添加一个 Item，通过 packet_id 指定了操作的 Packet，item_id 和 item_name 分别指定了 Item 的标识符和名字。
- 函数原型：DataPacket* findPacket(const int &packet_id)
  描述：调用该函数可以向用户返回指定 packet_id 的 Packet 实例。
- 函数原型：DataItem* findItem(const int &item_id)
  描述：返回指定的 Item。
- 函数原型：void itemAssociatingWithVariant(const int &packet_id, cosnt int &item_id, const QString &variant_name)
  描述：该函数实现指定的 Item 与指定的变量相关联。

（2）DataPacket 类接口

- 函数原型：void insertItem(DataItem *item)
  描述：将指定 Item 插入到 Packet 中。
- 函数原型：DataItem* getItem(const int &item_id)
  描述：返回指定 Item 标识符的 Item 指针。

（3）DataItem 类接口

- 函数原型：void setVariant(const QString &variant_name)
  描述：将 Item 与指定变量关联起来。

3.3.2 驱动配置

由于组态软件并不能假定用户在监控系统中所用的设备，因此对于用户使用的驱动程序只能由用户自己提供。用户一般以动态链接库的形式将驱动程序放在指定的目录，但为了让组态软件知道驱动程序的存在，那就需要用户对每一种设备的驱动进行配置，并且同上一小节所描述的站点配置信息关联起来。

为了达到描述驱动的目的，驱动配置至少需要有五个属性：驱动标识符、驱动路径、与之关联的 DataDevice 实例（即站点配置信息）、接收数据函数的字符串和发送数据函数的字符串。

驱动标识符是该驱动在系统中的唯一标识，任何模块想使用该驱动只能通过该标识符来查找；驱动路径即驱动动态链接库的文件路径，通过该路径系统可以加载动态链接库，从而使用相关函数来与设备进行数据交互；关联的站点配置，每一个驱动程序都是与某一设备相关的，因此需要为每个驱动关联相关的设备配置信息；接收数据函数的字符串和发送数据函数的字符串，通过这两个字符串，可以从

动态链接库中的到相关函数指针，从而实现与设备之间的数据收发。

3.3.3 通信方式配置

一般组态软件都支持多种通信方式，如以太网、蓝牙和串口等。通信方式的配置包含两部分：通信参数配置和驱动关联配置。通信参数配置即当用户选定通信方式后，进行的相关参数的配置。如图 3-14，以串口通信方式为例，当用户将通信方式选择为串口时，将会出现串口通信相关的参数配置，如图中所示，用户需要配置串口号、波特率、数据位、停止位和奇偶校验等信息。

图 3-14 通信方式配置界面

为了能够让通信方式配置信息与驱动配置信息关联起来，因此在通信方式配置界面用户还需要选定与该通信方式相关联的驱动标识符。

3.4 查询报警模块

3.4.1 报警系统

在生产过程中，设备会不断地向监控系统反馈自身状态信息，系统需要根据设备的反馈进行相应的处理。系统对反馈的处理包含报警信息，如果设备反馈的数据超出了规定的范围，将会产生报警。当观察员从界面发现相关的报警显示的时候，就会对产生异常的设备进行查看。如果设备恢复正常，那么数据就会恢复正常，当系统检测到数据处于正常范围内，那么就会向报警系统发送一个警报解除的指令，

从而让报警系统得知该警报已经解除，并对警报的显示做出相应的处理。

上面对报警系统的基本原理进行了阐述，可以得知与报警系统相关的数据分为两类：警报数据和警报解除数据。为了让系统能够很好的解析这两类数据，因此将两者都封装成了相应的类，同时为了使得报警系统的接口设计可以统一，与警报数据相关的两个类将拥有同一个父类，这样就可以利用 C++多态的性质，统一使用父类指针进行数据处理。类的继承关系及相关属性如图 3-15 所示，基类 AlarmData 具有 alarm_type 属性，通过它可以区分报警数据的类型，当 alarm_type 等于 0 的时候为 AlarmInvoke 类型，即警报数据；当 alarm_type 等于 1 时，为 AlarmRelease 类型，即警报解除数据。

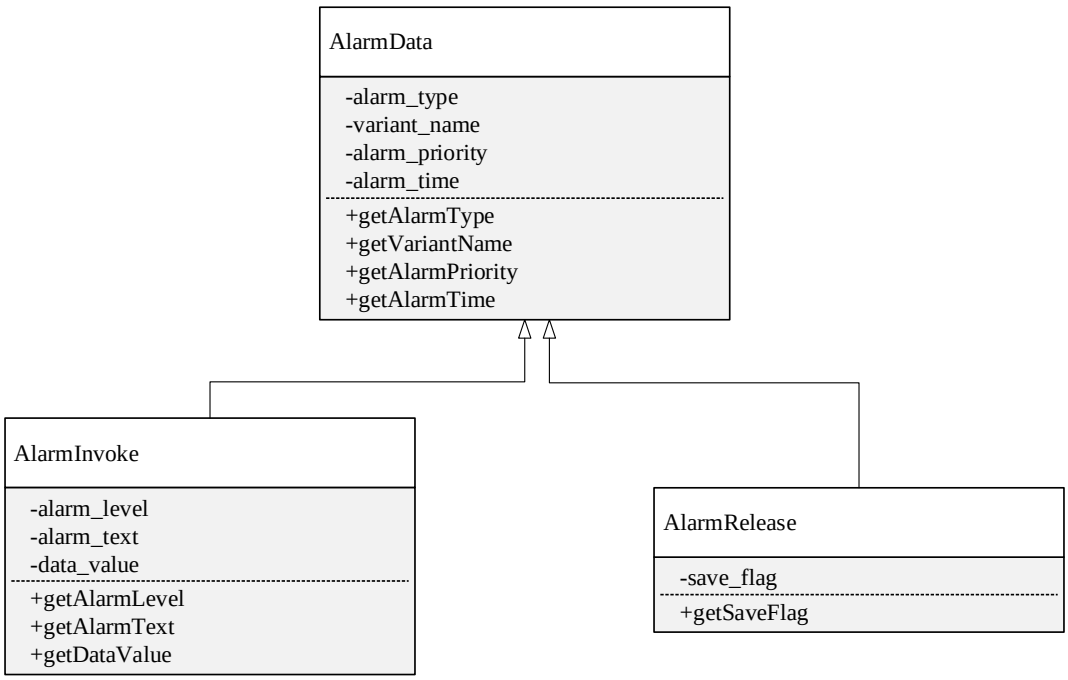

图 3-15 报警数据继承关系

基类 AlarmData 还具有 variant_name、alarm_priority 和 alarm_time 三个属性，其中 variant_name 表示变量的名称，即标识出警报是由哪个变量产生的；alarm_priority 表示警报的优先级；alarm_time 表示警报产生的时间。AlarmInvoke 类有三个独有属性，分别是 alarm_level、alarm_text 和 data_value，alarm_level 表示警报的层次，即“低低”、“低”、“高”和“高高”；alarm_text 记录了警报产生时显示给用户的信息；data_value 表示触发警报的变量的值。AlarmRelease 类具有一个私有属性，该属性向报警系统表明是否将该条警报解除指令存入数据库，有时警报解除指令也可以反映系统的运行情况，因此合理的记录可以方便后续的系统管理。

报警系统的功能是管理系统产生的各种警报，然后向用户反映到界面上，因此

报警系统对警报数据和警报解除数据的处理是其核心。报警系统有两种方式可以向观察员反映目前的警报情况：通过相关动画控件和以表格的形式显示警报信息。用户可以在报警界面添加相关的图元，使其关联相应的变量，设置相关的限制值，从而反映变量的变化情况。然而警报信息以表格的形式呈现才是报警系统的主要展现形式，报警系统通过警报的优先级对警报信息进行排序，然后使得优先级最高的显示在列表的前面，如果同一优先级有多个警报信息，那么将通过警报产生的时间先后顺序对其排序。

报警系统由 AlarmHandle 类实现，由于需要对警报信息进行排序，因此 AlarmHandle 具有两个数据结构：

- QMultiMap<QDateTime, AlarmInvoke*> date_sort：date_sort 是用 Qt 的 QMultiMap 实现的，该数据结构允许同一个关键字可以有多个不同的值，date_sort 数据结构的关键字是 QDateTime，即警报产生的时间，其对应的值则为 AlarmInvoke 类实例的指针。
- QMap<int, QMultiMap<QDateTime, AlarmInvoke*>> priority_sort：priority_sort 数据结构的关键字是警报优先级，优先级的总数在组态软件中已经是确定好了的，因此该表的大小也是确定了的。每个关键字对应的值则为 QMultiMap<QDateTime, AlarmInvoke*>类型的数据结构，这样就可以使得具有统一优先级的警报数据紧挨着存放，相同优先级的警报数据再根据时间进行排序，从而使得报警系统能够将高优先级的警报信息优先呈现给监控者。

报警系统对警报数据的处理统一交给 void handle(AlarmData *alarm_data)接口处理，其处理流程如图 3-16 所示。

从图 3-16 可以看出报警系统对数据的处理分为两类：当数据为警报数据时，系统会调用警报数据的 getAlarmPriority 方法获得该警报的优先级，然后根据优先级在 priority_sort 表中进行查找，如果查找到了相应的 date_sort 表，就调用 getAlarmTime 方法获得时间，用时间作为关键字将报警信息插入 date_sort 表，如果没找到对应的 date_sort 表，那么就创建，然后再插入，最后将警报信息插入数据库；当数据为警报解除数据时，同样是利用优先级查找到相应的 date_sort 表，然后遍历该表，删除其中变量名与警报解除数据中变量名相同的数据项，最后根据 save_flag 标识符决定是否将该条警报解除数据插入数据库。

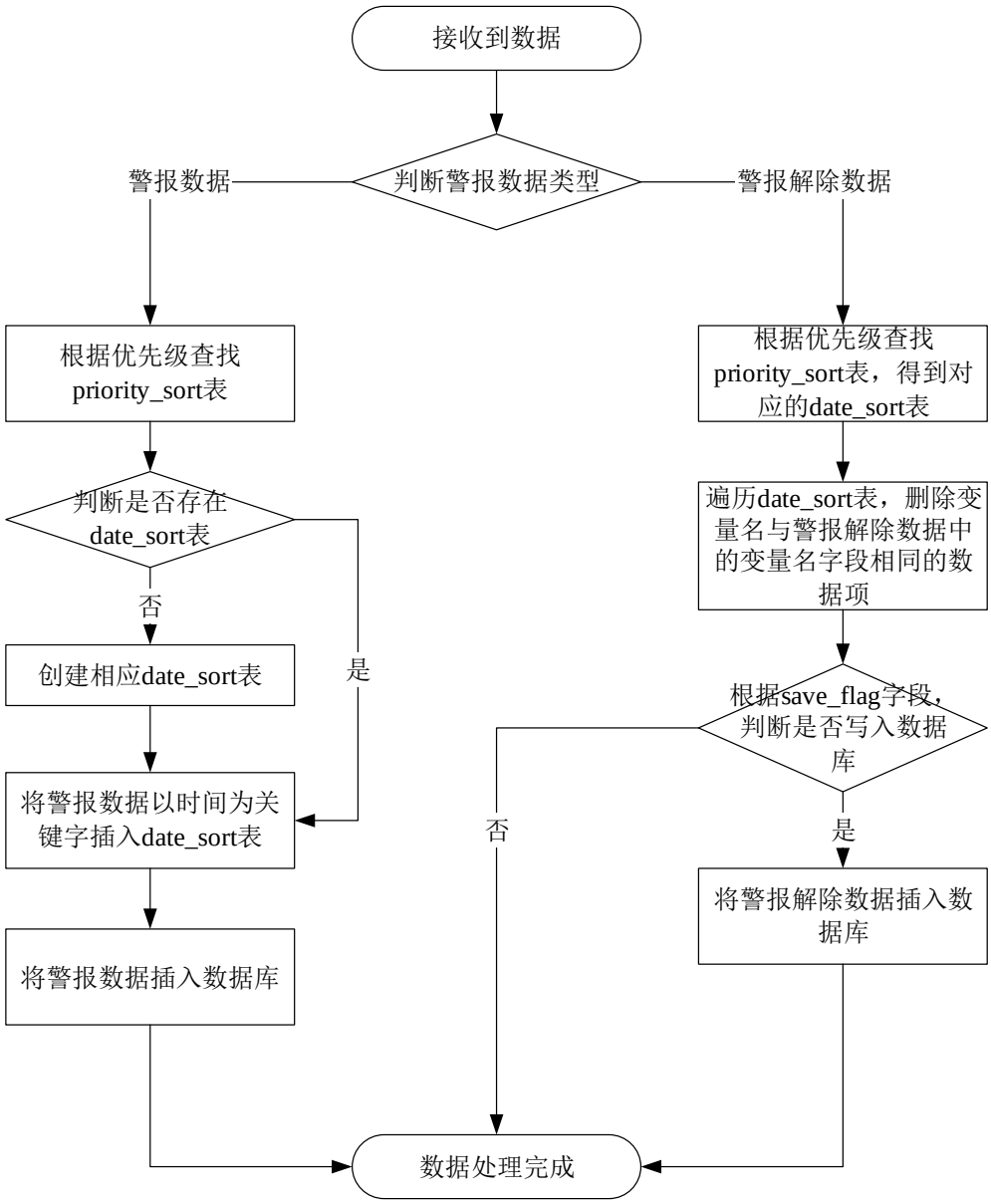

图 3-16 报警系统处理流程

3.4.2 历史数据查询

对于一个监控系统，不仅需要实时地反映生产状况，同时还需要为使用者提供浏览历史数据的功能。在很多情况下，历史数据可以帮助用户分析设备的运转情况，同时也为用户做决策提供了实际的数据支撑，因此本文设计的组态软件也为用户提供了相关历史数据查询的模块。

历史数据分为三类：变量数据、警报数据和发送数据。变量数据即监控系统从设备接收到的实时数据，这些数据反映了设备的运转情况；警报数据即系统产生的警报，对于警报数据来说又分为两类，一是警报数据，二是警报解除数据；发送数

据记录的是系统发往设备的数据。

对于上面三类数据都设计了相应的数据库进行管理，变量数据对应着variant.db；警报数据对应着 alarm.db；发送数据对应着 senddata.db。将各类数据分散到不同的数据库存储的好处在于，用户可以根据实际情况的需要对数据库进行裁剪。

3.5 本章小结

本章主要描述了组态软件的开发系统的设计与实现。开发系统分为四个模块：图元模块、变量配置模块、系统配置模块和查询报警模块。在图元模块的设计与实现中，主要对图元的变形、图形显示和属性配置进行了讨论，并给出了具体的实现细节。在变量配置模块中，重点阐述了变量在组态软件中的作用，然后介绍了其设计方法。在提供配置模块中，描述了监控站点配置、驱动配置和通信方式配置这三者的关系，并且给出了具体的设计方案。在查询报警模块中，详细介绍了报警系统的整体结构以及运作流程。

第四章 组态软件工程文件的设计与实现

用户通过开发系统设计监控系统，然后开发系统将所有的配置和操作都以文件的形式存储起来，这就是组态软件的工程文件。

组态软件的工程文件作为开发系统和运行系统之间的桥梁，它需要将各种数据分类组织好，这样才可以方便两个系统对相关文件进行操作。同时为了简化对文件的操作，工程文件的设计需要满足以下条件：

(1) 工程文件具有多种类型，这样可以达到对数据分类存储的目的。

(2) 工程文件具有统一的数据组织结构，这样可以方便数据的解析与存储。

(3) 工程文件提供了一种能够方便将数据写入文件和读出文件的机制。

本章将围绕工程文件需要满足的三个条件来展开对工程文件的设计与实现的详细讨论。

4.1 工程文件的分类

本文描述的组态软件的工程文件分为如下几类：

（1）与工程信息相关

对于一个经过组态软件开发系统所生成的工程来说，需要有一个文件来记录工程信息，包括工程名称、工程创建时间、工程修改时间、工程创建人、工程描述以及其他工程文件的路径信息等，这些信息都将记录在工程索引文件中，即“工程名.prj”文件（对于一个具体工程来说）。

其中其他工程文件的路径信息这一项非常重要，系统通过解析工程索引文件，然后根据该信息判断其他文件是否存在，从而确定工程的完整性，如果发现工程缺失文件，将会告知用户。

（2）与图元相关

上一章介绍过图元库是以动态链接库的形式存在的，同时图元库还对应着一张图元信息表。因此与图元相关的文件包括图元动态链接库和图元信息表，通过这两个文件可以将图元库导入系统。

（3）与数据库相关

组态软件中需要记录的数据可以分为三类：变量数据、警报数据和发送数据。该三类数据将分别用三个数据库（variant.db、alarm.db 和 senddata.db）进行管理，数据分为三个数据库存储是因为在某些系统中有些数据是不必要的，因此用户可以将某个数据库移除。

（4）与变量配置相关

前面已经介绍过，变量作为组态软件中各个模块交换数据的介质是非常重要的，因此变量配置的信息将会用一个文件来记录，即 variant.config 文件。

（5）与系统配置相关

与系统配置相关的文件包括三个：监控站点配置文件、驱动配置文件和通信方式配置文件，分别对应文件 site.config、driver.config 和 communication.config。

（6）与画布相关

画布作为用户图形化界面设计的载体，因此需要记录用户在上面进行的各种操作，而画布中的元素是图元，因此对于画布的存储即是记录图元的相关属性，通过将图元信息写入文件，以后通过文件内容又能再现画布。一个工程可能包含多个画面，因此每个画面都将存储为一个文件，然后将这些文件统一放入 canvos 文件夹中进行管理。

4.2 工程文件的数据组织

上一小节介绍了工程中涉及到的各类文件，虽然组态软件的工程文件有多种类型，但是文件中数据的组织形式是统一的，如图 4-1 所示，每一个工程文件都包含两部分：文件信息和数据。

文件信息主要记录该文件的一些基本属性，包括标识符、版本号、幻数以及创建相关的信息。标识符是文件类型的标识；版本号和幻数都是用来确定生成工程文件的组态软件的版本的，这样就可以防止组态软件和工程文件由于版本不匹配造成的问题；与创建相关的信息主要包括创建者和创建时间，同时还包括修改者和修改时间，通过这些信息用户可以得知对文件的操作状态，如当文件被修改了，那么可以查看是谁进行的修改以及修改的时间。

图 4-1 也展示了文件的数据部分的组织格式，文件的数据部分由多个数据项组成，例如在变量文件中，每一个数据项就代表着一个变量。由于每个数据项包含的内容也比较复杂（如每个变量包含多个属性），因此每个数据项分为标识符和数据两部分。标识符是该数据项的唯一标识（如果是变量，那么就是变量名），每个数据项的标识符不能相同。数据部分则是由多个数据段组成，如图 4-1 中的数据 1 和数据 2，每个数据段又通过三个属性进行描述：类型、长度和数据。类型表示的是该数据的数据类型，例如变量的值可以是整型、浮点型和布尔型，长度表示的是实际数据的长度，通过它系统能够确定读取了多少数据，数据即实际数据。

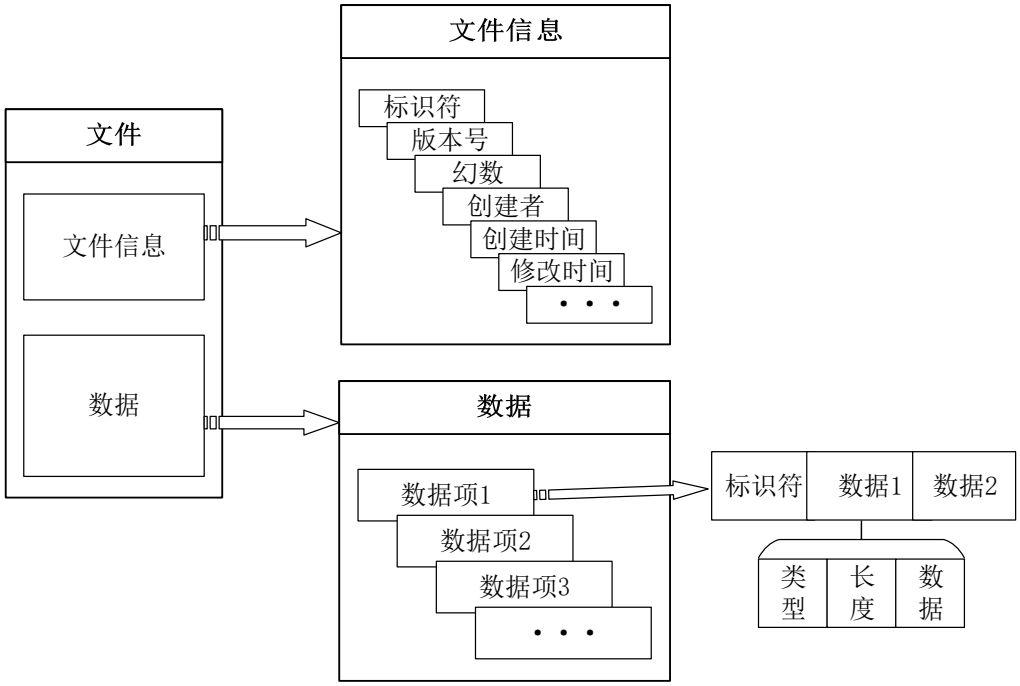

图 4-1 文件数据组织格式

4.3 数据的序列化与反序列化

用户保存设计文档的动作，也就是将组态软件中各个类序列化到文件中的过程，因此数据的存储也就变为类的序列化问题了。

Qt 为用户提供了 QDataStream 类，利用该类用户可以将 Qt 库自带的基本类型序列化到文件。利用该类来实现数据的存储有两大优势：一是该类提供了丰富的接口，利于软件的开发；二是该类对基本类型序列化后的结果是独立于操作系统和处理器的，因此不会存在大小端问题，这对多平台使用是必需的。

由于 QDataStream 类只能实现基本类型的序列化，因此对于本文描述的各个类的序列化需要进行相应的处理，下面以图元的序列化为例描述实现过程。

由于每一个图元所包含的信息是不同的，因此每个图元需要存储什么信息只有自己才知道，为此图元基类 Primitive 将包含 virtual void serialized(QDataStream &out)虚函数，而每个图元将重写该函数来保证自己能够被正确地序列化。对于每个图元序列化都需要进行两步：

(1) 将图元的名称写入文件，作为数据项的标识符。

(2) 依次处理图元的各个属性，例如处理图元的位置信息，首先存储数据类型即 QPoint（Qt 中基本类型都有相应的枚举值），利用 sizeof(QPoint)得到数据的长度，然后再将实际数据写入文件。其他各个属性依次进行上述处理，即可完成

该图元的序列化。

对于数据的读取（反序列化），即上述过程的逆过程。每一个需要反序列化的类都有 deserialized 方法，每一个类实例都知道该如何解析数据，使得自己的所有属性都具有正确的值。

4.4 工程导入流程

组态软件中，不管是开发系统还是运行系统都需要多次导入工程，开发系统导入工程是为了能够修改工程，运行系统导入工程是为了获得系统的相关配置信息，虽然两者的目的不同，但是其工程导入的流程大体都是一样的。

工程导入流程主要分为三个阶段：工程索引文件解析、各类文件解析和解析结果判断。如图 4-2 所示，

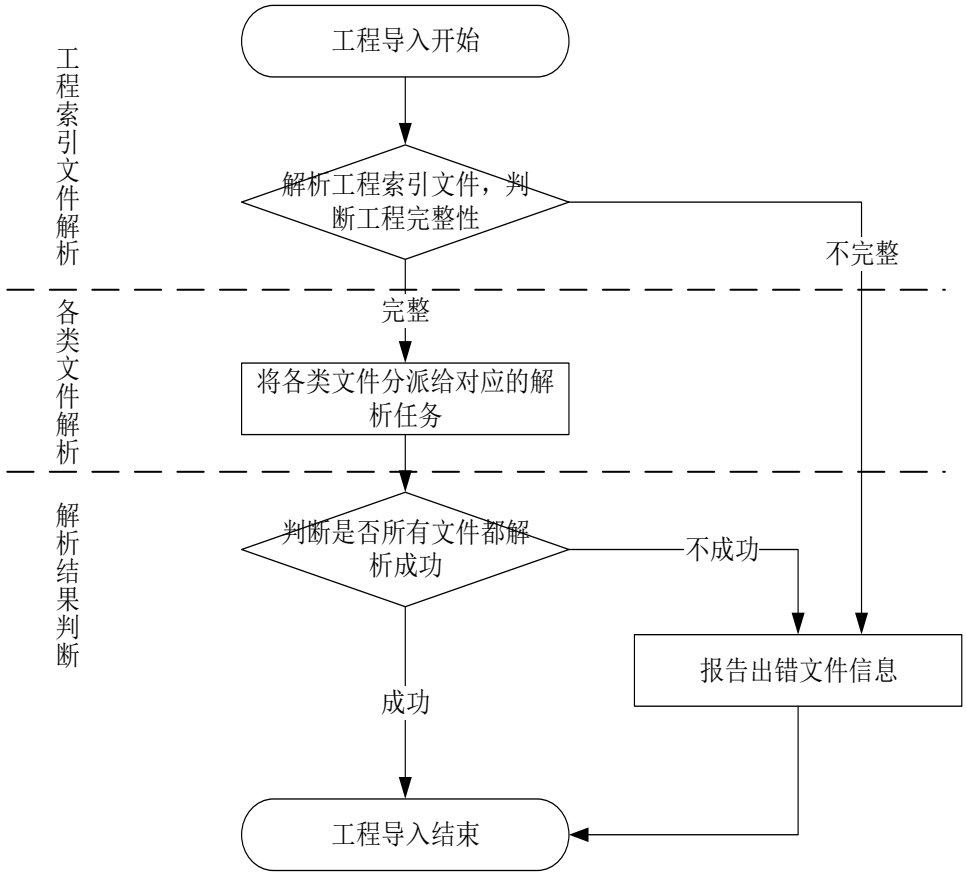

图 4-2 工程导入流程图

工程索引文件解析阶段，系统将首先确定文件的基本信息是否正确，如版本号等；其次系统将寻找索引文件中列出的各类文件的路径信息是否正确，如果不正确，将向用户做出相应的提示；最后在校验都成功的情况下，系统将各类文件的路径信息分配给相应的解析任务进行解析。

各类文件解析阶段，对于每一类文件都将有相应的解析任务来处理，如变量，就会存在相应的变量解析任务，解析任务通过解析文件，然后将数据放入指定的数据结构中，如果解析出错，将会向系统报告错误。

解析结果判断阶段，系统会汇集所有解析任务的解析结果，如果所有结果都正确，那么工程导入结束；如果有文件解析不成功，那么系统将向用户反馈具体的错误信息。

4.5 本章小结

本章描述了组态软件中工程文件的设计，工程文件作为连接开发系统和运行系统的桥梁以及用户设计思想的成果，在组态软件中具有重要地位。本章通过对工程文件的分类、工程文件的数据组织和数据的序列化和反序列化这三个方面的详细描述，展现了该组态软件工程文件的设计思想和方法。

本章最后还描述了组态软件中工程导入的具体流程的设计。

第五章 组态软件运行系统的设计与实现

前面两章对组态软件的开发系统和工程文件进行了详细描述，本章将对组态软件的运行系统的相关细节进行介绍。运行系统作为监控系统运行的环境，它主要由四个模块构成：交互界面模块、控制模块、通信模块和历史数据处理模块。

5.1 交互界面模块

交互界面模块主要具有三个功能：图元加载器、图形显示刷新和人机交互。图元加载器主要负责画布图元的管理；图形显示刷新描述的是图元根据变量的变化来改变自己的显示效果；人机交互完成了上位机和下位机的互动。

（1）图元加载器

上一章介绍过，系统首先解析工程索引文件，然后将各类文件的解析分配到相应的功能模块，因此与画布相关的文件的解析将由图元加载器完成。图元加载器的工程由 CanvosManager 类实现，在第三章已经介绍过该类，当时是在组态软件开发系统的设计中提到的，当用户在开发系统中对画布上的图元进行操作的时候，该类将记录所有的用户操作，方便以后的存档工作。由于开发系统和运行系统都需要进行图元加载工作，所以在设计的时候由同一个类实现。

CanvosManager 类有两个关键数据结构：QMap<QString, QStringList>，即图元变量关联表，该结构的关键字为变量名，值为与该变量相关联的图元的对象名称，通过该结构系统可以根据变量名索引到与之关联的所有图元，从而到达有目的的操作；另一个数据结构是 QMap<QString, Primitive*>，即图元索引表，该结构的关键字为图元的对象名称，值为相应图元的指针，通过该结构系统能够根据对象名称索引到图元实例。

图元加载器完成的任务就是对上述两个数据结构赋值，其流程如图 5-1 所示。

如图 5-1，图元的加载流程分为如下步骤：

- 获得相关文件，判断文件是否有效，若文件无效，记录相关错误信息，文件有效则继续处理。
- 依次读取文件中的每一个数据项，直到文件读取完。对数据项的处理过程如下：
  - 读取数据项的标识符，该标识符是图元的名称，通过该标识符可以提取图元名，从而构造对应字符串，即 get+图元名格式，然后利用该字符串从图元动态库中提取相关函数，从而获得相应图元的实例。
  - 继续读取数据项，设置图元相关属性。

- 当图元属性设置完成后，调用图元 getName 方法获得图元的对象名称，然后将图元存储在图元索引表中；再调用图元的 getVariantName 方法得到图元所关联的变量名称，然后设置好图元变量关联表。

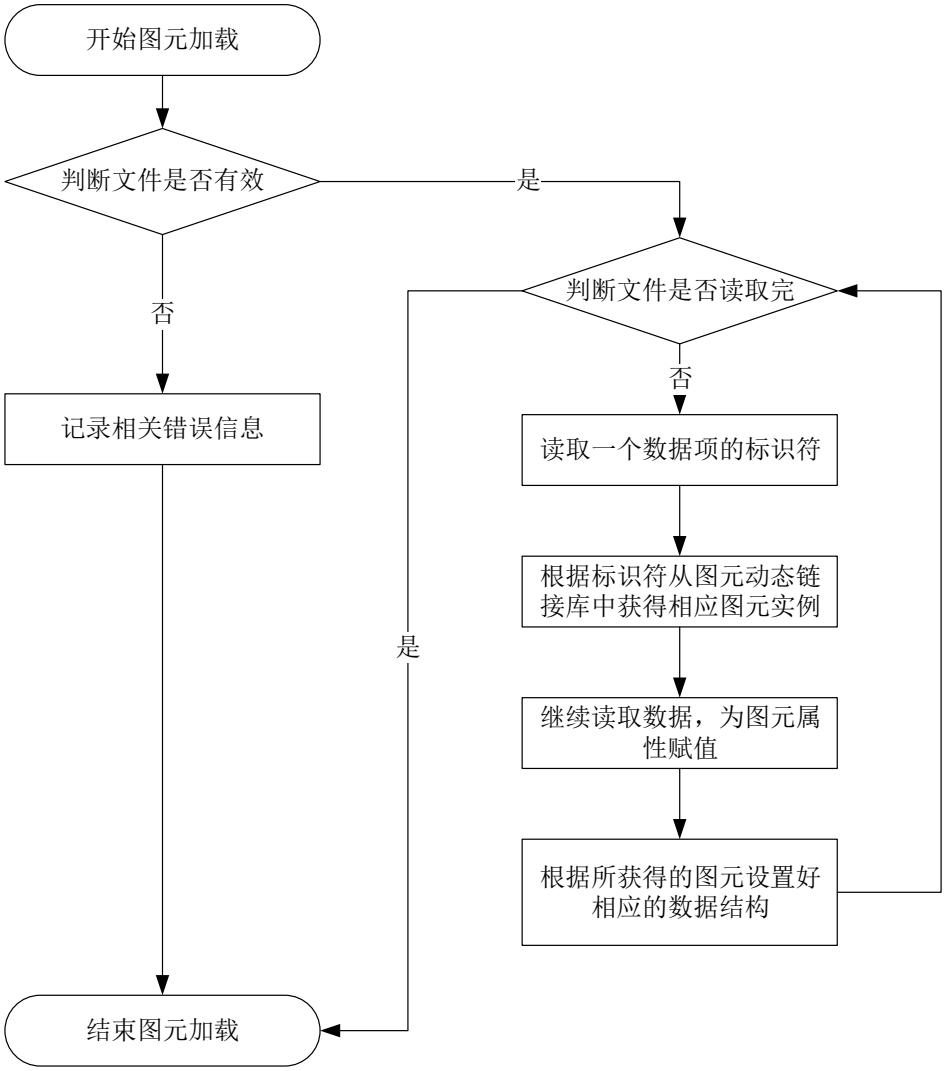

图 5-1 图元加载流程

（2）图形显示刷新

监控系统的图形显示是一个动态变化的过程，图元作为图形显示控制的最小单元，因此图形显示变化本质是图元变化。由于图元本身是静态的，因此它需要外部数据来促使其发生变化，而外界数据在组态软件中的映射物为变量，最终图形显示的变化就在于变量的变化。因此该功能模块需要解决的问题是，当设备通过相关通信方式将数据传送到监控系统从而导致与之关联的变量的值发生变化的时候，系统如何告知图元。

在 Qt 中有一种对象之间的通信机制，即信号/槽。该机制可以将两个对象进行

关联（对象可以存在于不同线程中），从而使得两个对象可以通信。事件发生时，对象可以发送相应的信号，关注该信号的对象，能够收到信号，一旦收到信号可以做出对应的处理。

在图形显示刷新中涉及到两个对象：CanvosManager 和 VariantManager。CanvosManager 管理的是画布中的相关图元，而 VariantManager（下一小节介绍）管理着系统中的所有变量。当 VariantManager 监测到有变量的值发生变化，会将所有发生变化的变量的名称放入一个 QStringList 中，然后产生 void variantChange(QStringList)信号，所带参数即变化了的变量的名称的集合。而 CanvosManager 则通过 connect 函数将自己的处理函数与该信号进行关联，一旦信号产生将调用处理函数进行处理。处理过程如图 5-2 所示。

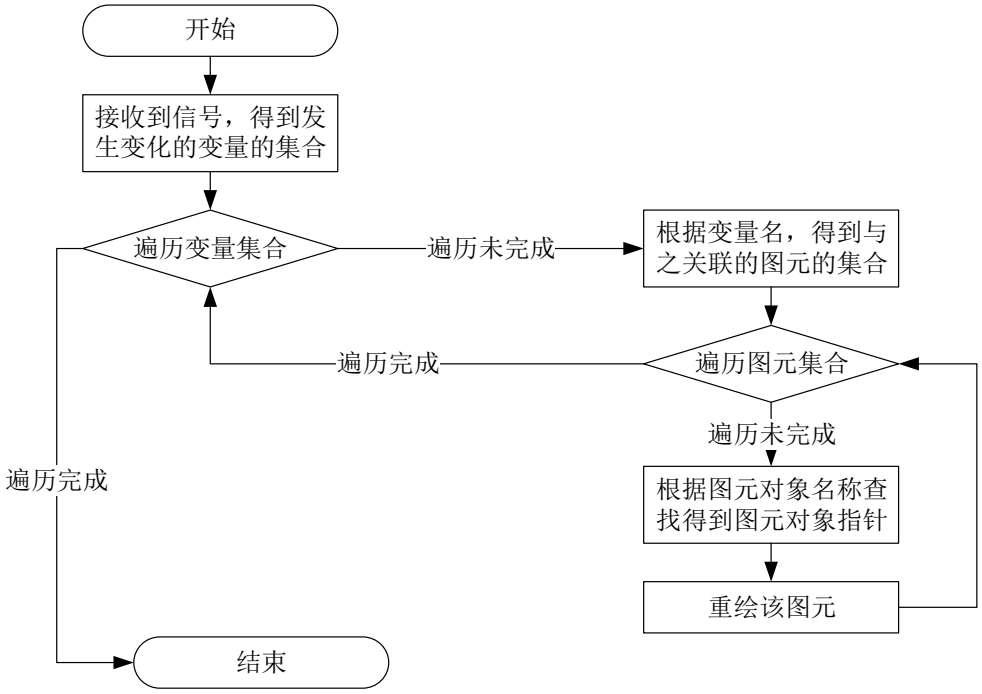

图 5-2 图形显示刷新处理流程

（3）人机交互

对于一个监控系统，监控界面是操作员能够实际感知到的东西，因此当需要和下位机（即受监控的设备）进行交互的时候，往往也是通过界面操作进行的。

在监控界面中，设计人员可能设计了相关按钮，当操作员对按钮进行操作的时候，这是就会触发系统产生一条命令，然后系统会将该命令交给数据发送器（后面小节会介绍）处理，数据发送器依次发送命令，最后命令将到达相关设备。监控系统正在运行的过程中，操作人员可以进行相关数据的查询，这属于人机交互的一部分。

5.2 控制模块

控制模块处理的内容主要包含配置文件和数据。控制模块需要将用户配置的信息导入内存，并以合理的结构进行存储，方便索引；同时控制模块还要负责处理下位机传来的数据。

5.2.1 变量管理器

上一章介绍了与变量相关的工程文件，而该工程文件则是由变量管理器来进行解析的。变量管理器主要完成如下工作：

（1）记录变量配置信息，如变量值类型，是否报警等属性。

（2）记录变量的最新值，方便其它模块使用。

（3）保留近期数据，这样可以不用访问数据库就能得到近期数据。

（4）在该变量需要报警的情况下，记录报警情况，分析是解除还是继续报警。

（5）每一次更新变量，将产生一条数据记录，发送给数据库处理部分。

（6）发送相关信号，通知各模块数据已经更新。

变量的相关信息是通过 Variant 类来存储的，该类具有如下属性：

- VariantAttribute attribute：记录工程文件中读取出来的变量配置信息，包括变量的名称、值的类型、是否报警以及报警相关的配置信息。
- QVariant latest_value：记录变量的最新值。
- bool is_alarmed：值为 false 时，表示不在报警状态；为为 true 时，表示在报警状态。
- QMap<QTime, QVariant> value_buf：该结构关键字为时间，值为变量在该时刻的值，通过该结构能够记录一段时间内变量的值。

变量管理器的功能由 VariantManager 类实现，该类存储了变量的相关属性，由于一个系统具有多个变量，因此为了很好的索引各个变量，VariantManager 通过使用 QMap<QString, Variant*> variants 结构进行管理，该结构可以使用变量名作为关键字查询到变量的相关信息。

除此之外，VariantManager 还提供了下列几个重要方法。

（1）void init(QString &file)：该方法完成变量工程文件的解析，初始化 Variant 类实例。其处理流程如下：

- 校验文件信息，判断是否为有效文件。
- 遍历提取数据项，获得变量名称，创建 Variant 实例，继续读取数据初始化 Variant 实例。当变量信息初始化完成后，将该变量记录到 variants 结构中。

（2）QVariant getLatestValue(QString &name)：通过该方法，用户可以查询到指定

变量名的变量的最新值。

（3）QList<QVariant> searchVariantValue(QString &name, QTime &from, QTime &to)：用户可以使用该方法查询某段时间内某个变量的值，结果以链表的形式返回，name 代表需要查询的变量名，from 代表查询时间段的开始时间，to 代表结束时间。

（4）void updateVariant(QMap<QStirng, QVariant> values)：当变量管理器获得新数据的时候，调用该函数。该函数处理流程如图 5-3 所示。

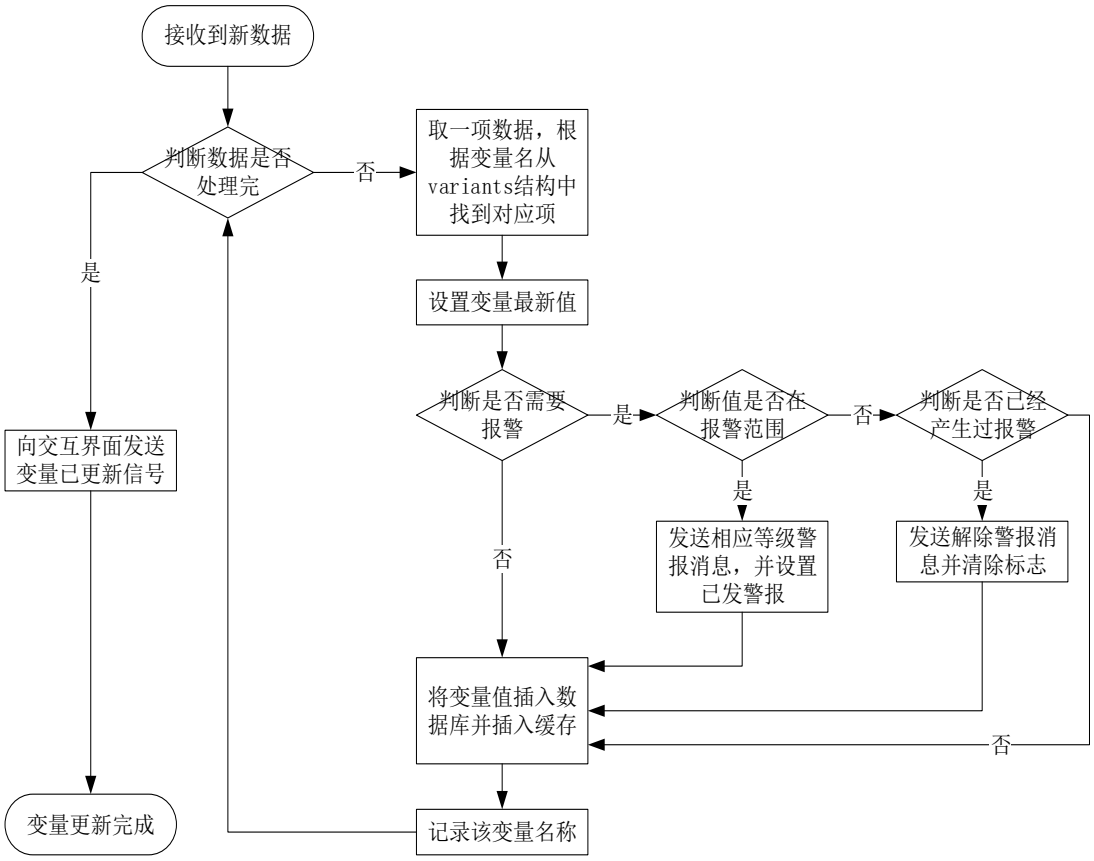

图 5-3 变量更新处理流程

从图 5-3 可以看出，变量管理器接收新数据的处理步骤如下：

（1）获取一项数据，数据格式为<变量名，变量值>。通过变量名可以从 variants 结构中查找到相应的 Variant 类的实例，通过 VariantAttribute 类实例可以得知变量值的类型，然后将变量存储在 latest_value 属性中。

（2）根据最新的变量值，读取 attribute 属性中的变量报警信息的描述，决定是否产生报警信息。如果变量需要设置报警，那么将根据变量的实际值与变量配置属性中的界限值进行比较，从而确定是否产生警报。如果变量值处在警报界限值范围内，那么发送报警数据；如果变量值处在正常范围，那么根据 is_alarmed 的值进行下一步处理，当其值为 false 时，表示没有发生过报警，系统不做处理，当其值为 true

时，表示发生过报警，那么此时将发送警报解除消息并且将 is_alarmed 置为 false。

（3）将变量的最近值插入到数据库并且添加到 value_buf 中。变量管理器构造一条格式为变量名 变量值 日期 时间的数据，然后放入数据库模块的数据缓存中，数据库管理器将在合适的时间把数据写入数据库。

（4）将该变量的名称记录到一个链表中，这样当通知图元变量发生变化时，图元管理器可以得知哪些变量发生了变化。

（5）当所有的变量都更新完毕之后，变量管理器将向图元管理器发送信号，通知其变量已经更新。

5.2.2 系统信息管理器

在第三章详细介绍过与系统配置相关的功能模块，主要包含三部分：站点配置信息、驱动配置信息和通信方式配置信息。为了能够对这三类信息进行统一的管理，因此组态软件中有对应的系统信息管理功能，在本课题中被称为系统信息管理器。系统信息管理器主要的功能包括加载与系统配置相关的工程文件和管理对应的配置信息。

系统信息管理器由 SystemInfoManager 类实现，该类包含的主要属性和方法如表 5-1 所示。

表 5-1 SystemInfoManager 类的属性和方法

SystemInfoManager 类的属性		
属性名	类型	描述
site_info	QMap<int, DataDevice*>	记录所有的站点配置信息
driver_info	QMap<int, DataDriver*>	记录所有的驱动配置信息
com_info	QMap<QString, DataComMethod*>	记录所有的通信方式配置信息
SystemInfoManager 类的方法		
方法名		描述
DataDevice* getSiteInfo(int site_id)		获得指定 site_id 的站点配置信息
DataDriver* getDriverInfo(int driver_id)		获得指定 driver_id 的驱动配置信息
DataComMethod* getComInfo(QString &com_id)		获得指定 com_id 的通信方式配置信息
void initSiteInfo(QString &file)		根据工程文件初始化 site_info
void initDriverInfo(QString &file)		根据工程文件初始化 driver_info
void initComInfo(QString &file)		根据工程文件初始化 com_info

SystemInfoManager 类的三个属性都是通过 QMap 来存储的，这样可以快速地找到需要的配置信息，对 site_info 属性，QMap 的关键字为站点的 id 号，值为 DataDevice 类型的指针；对于 driver_info 属性，QMap 的关键字为驱动的 id 号，值为 DataDriver 类型的指针；对于 com_info 属性，QMap 的关键字为一个字符串

（以太网通信的时候即为设备 IP 地址，串口通信时则为串口号），值为 DataComMethod 类型的指针。关于 DataDevice 类的结构在 3.3.1 小节已经做过详细介绍，下面主要介绍 DataDriver 和 DataComMethod 两个类的详细结构。

DataDriver 类主要记录从驱动配置工程文件读出的数据，由于存在多个驱动程序需要管理，因此 DataDriver 的实例通常也不止一个，其属性如表 5-2 所示。

表 5-2 DataDriver 类的属性

属性名	类型	描述
driver_id	int	驱动配置信息的标识符
driver_path	QString	对应驱动动态库的路径
site_id	int	与该驱动关联的站点配置信息的标识符
rcv_str	QString	驱动动态库中接收函数的函数名
send_str	QString	驱动动态库中发送函数的函数名

其中 rcv_str 和 send_str 属性是在系统加载驱动动态库的时候使用的，通过这两个字符串，系统能够获得动态库中相应的函数地址指针。

DataComMethod 记录了通信方式配置的相关信息，监控系统中存在多少需要监控的设备，那么就需要相等数量的 DataComMethod 实例来记录，其属性如表 5-3 所示。

表 5-3 DataComMethod 类的属性

属性名	类型	描述
com_id	QString	通信方式配置信息的标识符
com_type	CommunicationType	通信方式类型
driver_id	int	与该通信方式关联的驱动信息
com_attr	CommunicationAttribute	通信方式的相关属性

CommunicationType 为枚举类型，其定义如下：

```
typedef enum{
    Ethernet,       //以太网方式
    SerialPort,     //串口方式
    Bluetooth,      //蓝牙方式
    TypeSize
}CommunicationType;
```

其中 com_attr 属性为 CommunicationAttribute 类型的指针，由于通信方式的不同将会导致配置信息的不同，因此利用 c++多态的特性，使得每一种通信方式的配

置都由一个类管理，而该类继承于 CommunicationAttribute 类 ，属性配置相关类的继承关系如图 5-4 所示。

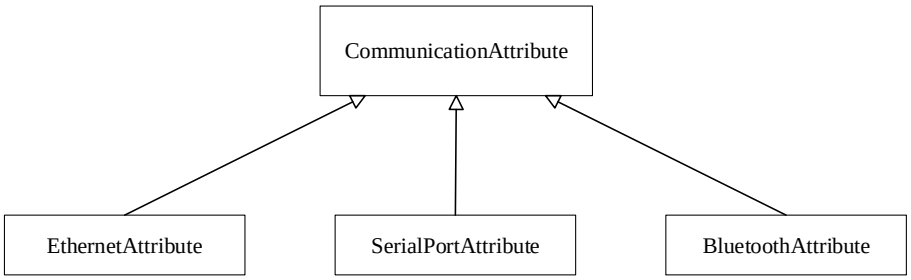

图 5-4 通信方式属性配置类继承关系

5.3 通信模块

5.3.1 驱动管理器

驱动管理器作为通信模块的核心，它将系统与设备建立好的连接和对应的驱动程序进行关联。驱动管理器需要完成如下工作：

- 记录当前连接设备的相关信息，关联相应的驱动。
- 解析来自设备的数据，方便变量管理器对变量进行更新。

（1）活动设备信息表

为了能够很好的管理监控软件正在监控的设备的情况，因此每一个与监控软件建立了连接的设备都将在驱动管理器中存在一条记录，通过该记录系统才能调用相应的驱动程序以及正确地解析数据。

组态软件通过 DeviceConnectingInfo 结构体来记录相关信息，其定义如下：

```
/*驱动程序的读数据函数原型，buf 表示缓存区，len 记录读到的数据
大小*/
typedef void (*ReadMethod) (char *buf, int *len);
/*驱动程序的写数据函数原型，buf 代表数据，len 代表要写的数据长
度*/
typedef void (*WriteMethod) (char *buf, int len);
typedef struct{
    QIODevice *device; /*设备建立连接后的指针*/
    int site_id; /*该设备关联的站点信息标识符*/
    CommunicationType com_type; /*通信方式*/
    ReadMethod read; /*读数据函数指针*/
    WriteMethod write; /*写数据函数指针*/
}DeviceConnectingInfo;
```

DeviceConnectingInfo 结构体中，由于在 Qt 中不管是以太网通信还是串口通信，其相关的类都是继承于 QIODevice 类，因此可以利用 QIODevice 类型的指针来存储各种通信方式建立的连接；记录 site_id 是为了能够查询到站点配置信息；通过 com_type 能够获知该连接属于什么通信方式，从而可以对 device 的指针类型进行转换；read 和 write 是该连接的读写函数，通过提取对应驱动中的函数来为这两个属性赋值。

在驱动管理器中，每当有一个设备请求建立连接的时候，都将初始化一个 DeviceConnectingInfo 实例，其处理流程如下。

- 根据连接的通信方式标识符，调用系统信息管理器的 getComInfo 方法得到相应的通信方式配置项。
- 根据通信配置项中的 driver_id，调用 getDriverInfo 方法得到对应的驱动配置项。
- 根据驱动配置项中的驱动路径信息，加载驱动，提取相关函数，为 read 和 write 赋值。
- 提取驱动配置项中的站点信息配置的标识符，赋值给 site_id。
- 将该连接信息加入到活动设备信息表中，完成一条连接的信息初始化。

（2）数据解析

文章的前面章节已经介绍过站点信息配置的详细过程，可以知道来自设备的数据应该具有如下格式。

Packet ID	Item ID	Data Length	Data	…	…	Item ID	Data Length	Data

通过从站点配置信息得到信息，可以将每一个 Data 与相应的变量关联起来，从而形成一个数据对<变量名，数据>，然后将该数据对放入 QMap 中，当数据解析完毕，驱动管理器将发送相应信号。

5.3.2 数据采集器和发送器

数据采集器和数据发送器负责接收和发送数据，为了能够对多种通信方式进行统一的管理，需要解决如下问题：

- 组态软件支持多种通信方式（如以太网、串口和蓝牙），该如何统一地管理这些连接？
- 当得知有数据到达时，利用什么机制通知系统？
- 系统收到数据的时候，该如何进行读取？

下面将对上述三个问题给出详细的解决方案。

（1）管理连接

在 Qt 中，所有与外设通信相关的类都有共同的基类 QIODevice，因为不管是

以太网通信还是串口通信，建立连接后返回的都是对应通信类的指针，这些指针都可以通过基类的指针来存储，从而在管理上也就比较统一了，对于系统来说不需要记住每个连接的通信方式，也不需要对每个连接进行特殊的处理。

（2）数据到达通知与数据读取

由于数据采集器需要管理多个连接，因此不可能以轮询的方式来判断每个连接是否有新的数据需要处理，这样会降低处理速率。为了达到预期的处理效果，因此需要利用Qt的信号/槽机制，由于每一个连接一旦有新的数据将会发送readyRead信号，如果将该信号与相应的处理函数进行关联，那么就可以达到异步处理的效果。

但是上面也介绍了，为了使得各种通信方式能够统一的管理，所有的连接实例都是用 QIODevice 类型的指针来存储的，然而 readyRead 信号是不带参数的，因此为了能够确定是哪个通路发的信号，那么就需要调用 QObject::sender()方法获得发送信号的实例的指针，关键代码如下。

```
//与系统建立连接的设备集合
QList<QIODevice*> connections;

//建立好每个连接的信号与槽函数关系
for(auto item : connections)
{
    connect(item, SIGNAL(readyRead()), this, SLOT(updateData()));
}

//信号处理函数
void updateData()
{
    //获得发送信号的连接
    QObject* obj = QObject::sender();

    //根据 obj 查找活动设备表
    //利用相应的 read 函数进行数据读取
    //将数据发送给驱动管理器
}
```

从上面的代码可以看出，一旦某个连接产生了新数据，那么该连接将会发送 readyRead 信号，触发相应的槽函数，而所有的信号都共用一个槽函数，在这个槽

函数中将通过发送信号的对象的指针查找活动设备表，从而得到一个DeviceConnectingInfo 实例，通过该实例中的 read 属性就可以读取设备发送过来的数据，数据读取成功后将通知驱动管理器进行数据解析。

数据的发送过程是数据接收过程的逆过程，首先将数据封装好，然后查找到对应的连接，利用 write 函数进行数据发送。

5.4 历史数据处理模块

历史数据处理模块包含两个功能部分：历史数据操作和数据库管理。历史数据操作主要实现了用户对历史数据的查询和删除；数据库管理主要实现将各种数据写入数据库，从而可以长期保存。

Qt 提供了访问数据库的相应 API，因此能够方便的对数据库进行管理。当有数据被放入数据库缓存的时候，数据库线程将会被唤醒，然后判断存储的数据是否已经达到上限值，如果达到将一次将所有缓存的数据写入数据库中，如果没有就继续等待，这样可以减少 IO 读写时间。

5.5 模块间同步

本章前几小节已经对运行系统的交互界面模块、控制模块、通信模块和历史数据处理模块四个模块的设计与实现进行了详细描述。虽然每个模块具有相对独立的功能，但是运行系统要完成监控任务需要各个模块之间的协调工作。对于模块之间的交互主要是数据之间的同步，因此解决模块之间数据的共享问题是解决模块间交互问题的关键。图 5-5 展示的是模块的关系，该图主要以数据为中心描述了模块之间的交互。

从图 5-5 可以看出，对于数据的处理主要由三条路径组成：接收到新数据后的处理、发送命令的处理和历史数据的处理。

（1）收到数据

通信模块中的数据采集器负责接收各设备传回来的监控数据，当数据采集器接收到完整的数据后将数据交给驱动管理器，驱动管理器需要从控制模块中的系统信息管理器获得该设备数据的解析方式，根据用户的配置，驱动管理器将数据解析成<变量名，值>的格式放入缓存中，即图 5-5 中所示的新接收的数据。当驱动管理器处理完数据采集器发送给它的数据的时候，驱动管理器将通知变量管理器，让其知道变量已经更新。

变量管理器接收到变量更新信号后，将更新内部的数据结构，同时完成这些工作后，它将向交互界面模块的图元管理相关的功能部分发送信号，然后图元根据变

量最新的值改变自身的状态。

（2）发送数据

当用户在监控界面上进行相关操作的时候，这时会向被监控的设备发送命令，命令产生后先是放入命令发送队列中，然后经过驱动管理器的处理后交给数据发送器，将其发送至目标设备。

（3）历史数据处理

历史数据包括变量数据、命令数据和警报数据。当变量管理器更新自身数据结构后，会产生一条数据将其放入数据库的缓存中。当交互界面产生一条命令的时候，也会将该命令放入数据库缓存中。同时若系统产生了报警，那么还将警报消息放入数据库缓存中。数据库管理部分将会在恰当的时候将缓存中的数据写入相应的数据库。对于历史数据，用户可以通过界面进行查询，由历史数据操作功能完成。

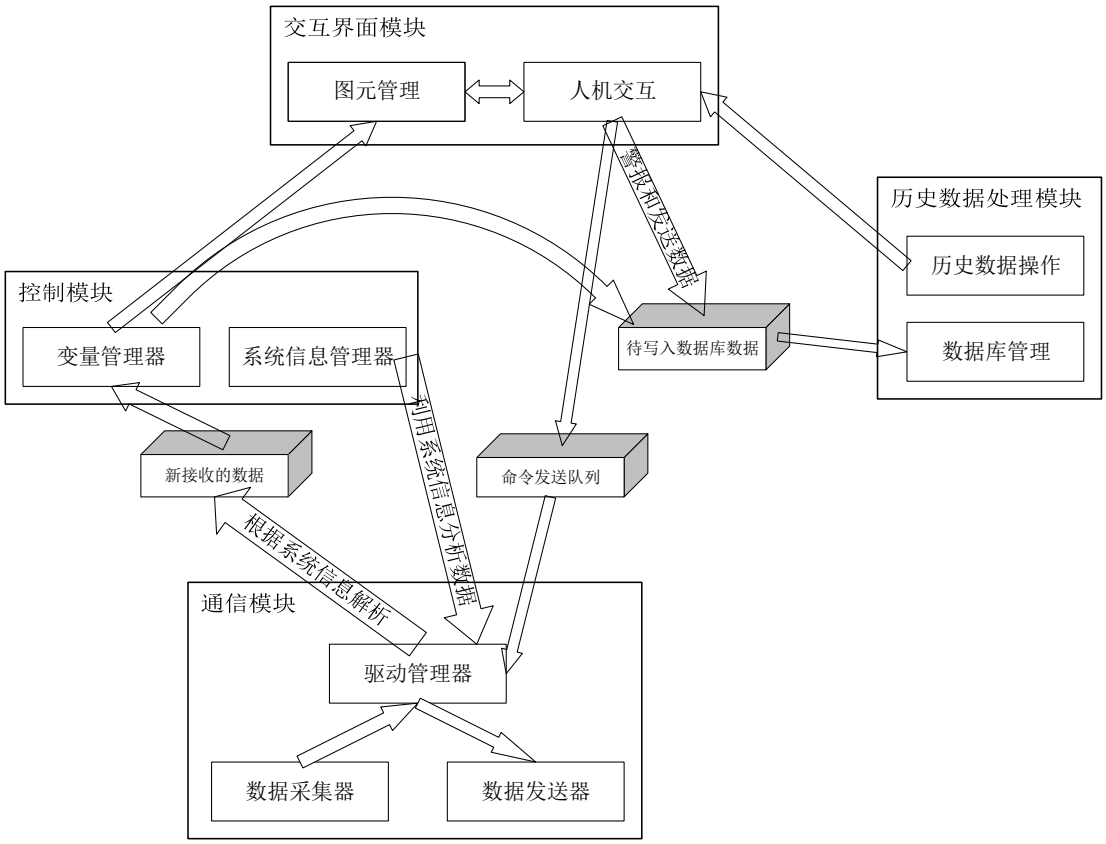

图 5-5 模块间关系

在对运行系统进行设计的过程中，界面交互模块、控制模块、通信模块和历史数据处理模块四个模块分别由各自的线程进行管理，因此完成线程之间的同步即可达到模块间的同步。

在对线程进行同步的过程中，主要用到了 Qt 提供的两种技术。

- 信号/槽机制：该机制可以实现多个对象之间的通信，通过将对象之间的信号和槽函数进行连接，一旦某个对象产生某个信号，那么与之相关联的对象将被通知，然后该对象就可以进行相关的操作。例如当变量管理器更新了变量，通过发送变量更新完成信号，图元管理器将接收到该信号，从而更新图形显示。
- 互斥量和条件变量机制：对于两个线程需要共享同一块内存数据的时候，这时需要将线程的操作进行序列化处理，这样才能保证数据安全。互斥量提供了线程能够序列化访问共享数据的功能；条件变量使得线程可以等待条件发生再进行相关操作。两者通常需要同时使用。例如在历史数据处理模块中，一旦数据库缓存中插入了新数据，那么此时有新数据的条件满足，那么该线程将被唤醒，从而可以进行相应的处理。

5.6 本章小结

本章详细介绍了组态软件中运行系统的详细设计与实现，分别对运行系统中的交互界面模块、控制模块、通信模块和历史数据处理模块的实现策略进行了详细阐述。在交互界面模块中主要描述了与图元相关功能的实现，在控制模块中详细展示了变量管理的细节，在通信模块中主要讲解了数据的流动与解析，在历史数据处理模块中简述了数据库的相关操作。

本章的最后还对整个运行系统的各模块的同步作了详细介绍。

第六章 组态软件的测试

本文前面章节详述了组态软件各个部分的设计与实现，本章将设计一个应用场景，从而可以尽可能多地测试该组态软件的各个模块。

6.1 应用场景描述

不管是工厂还是生活小区中，往往存在着热水供应这个问题，为了能够达到资源最高效的利用，利用一个系统来进行管理是很有必要的。

本文假定存在这样一个场景：有一家工厂，需要为四个生产车间供应热水，需要实时关注热水箱的相关情况，以及各个生产车间的用水量。该系统的网络拓扑结构如图 6-1 所示。

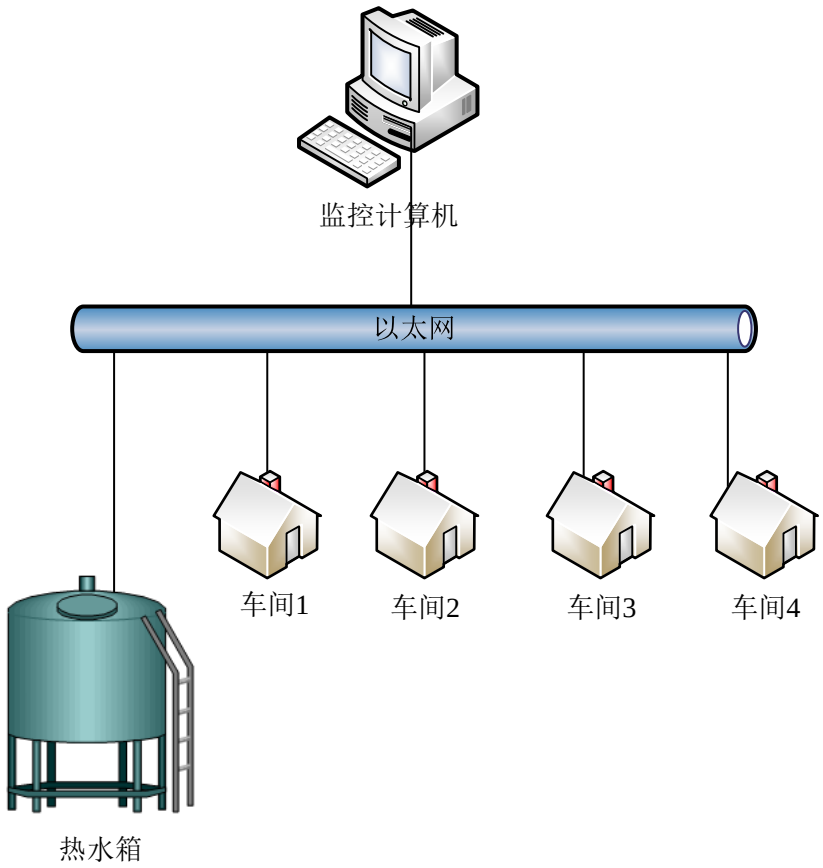

图 6-1 热水控制系统网络拓扑图

如图 6-1，热水箱为车间 1-4 供应热水，为了能够关注热水供应情况，需要热水箱提供两个数据：储水量和水温。车间 1-4 将提供水流量数据。因此热水箱和车间 1-4 这五个受监控点需要通过以太网与监控计算机相连，从而才能将数据反馈给

上层监控者。

6.2 系统设计

（1）变量配置

在整个监控系统中总共需要六个变量，热水箱两个：温度和储水量。四个车间各自一个变量记录水流量。在表 6-1 中列出了相关变量的关键属性。

表 6-1 系统变量表

变量名	变量类型	最大值	最小值	是否报警	报警条件
水温	浮点型	100	0	是	小于 70 或大于 90
储水量	浮点型	200	0	是	小于 20 或大于 190
一车间水量	浮点型	50	0	是	小于 10
二车间水量	浮点型	100	0	是	小于 70
三车间水量	浮点型	30	0	否	无
四车间水量	浮点型	70	0	是	小于 50

（2）系统信息配置

系统信息包含监控站点信息、驱动信息和通信方式信息。

热水箱和车间 1-4 都属于监控站点，因此需要进行相关配置，具体配置信息如表 6-2 所示。

表 6-2 监控站点信息配置

站点名	站点 ID	关联变量		数据格式	
热水箱	1	水温	储水量	水温	储水量
一车间	2	一车间水量		水流量	
二车间	3	二车间水量		水流量	
三车间	4	一车间水量		水流量	
四车间	5	一车间水量		水流量	

表 6-2 中，热水箱传输的数据包含两个数据项，其他车间传输的数据只包含一个数据项。

驱动信息配置如表 6-3 所示。

表 6-3 驱动信息配置

驱动名	ID	驱动路径	关联站点	接收函数字符串	发送函数字符串
热水箱驱动	1	./driver/tank.dll	1	receive_data	send_data
一车间驱动	2	./driver/plant1.dll	2	receive_data	send_data
二车间驱动	3	./driver/plant2.dll	3	receive_data	send_data
三车间驱动	4	./driver/plant3.dll	4	receive_data	send_data
四车间驱动	5	./driver/plant4.dll	5	receive_data	send_data

通信方式配置信息如表 6-4 所示。

表 6-4 通信方式信息配置

设备名	标识符	通信方式	通信地址	关联驱动
热水箱	192.168.1.60	以太网	192.168.1.60	1
一车间	192.168.1.61	以太网	192.168.1.61	2
二车间	192.168.1.62	以太网	192.168.1.62	3
三车间	192.168.1.63	以太网	192.168.1.63	4
四车间	192.168.1.64	以太网	192.168.1.64	5

（3）监控界面设计

在界面设计阶段，利用本组态软件提供的相关图元，构造出了一个仿真现实环境的监控界面。设计效果图如图 6-2 所示。

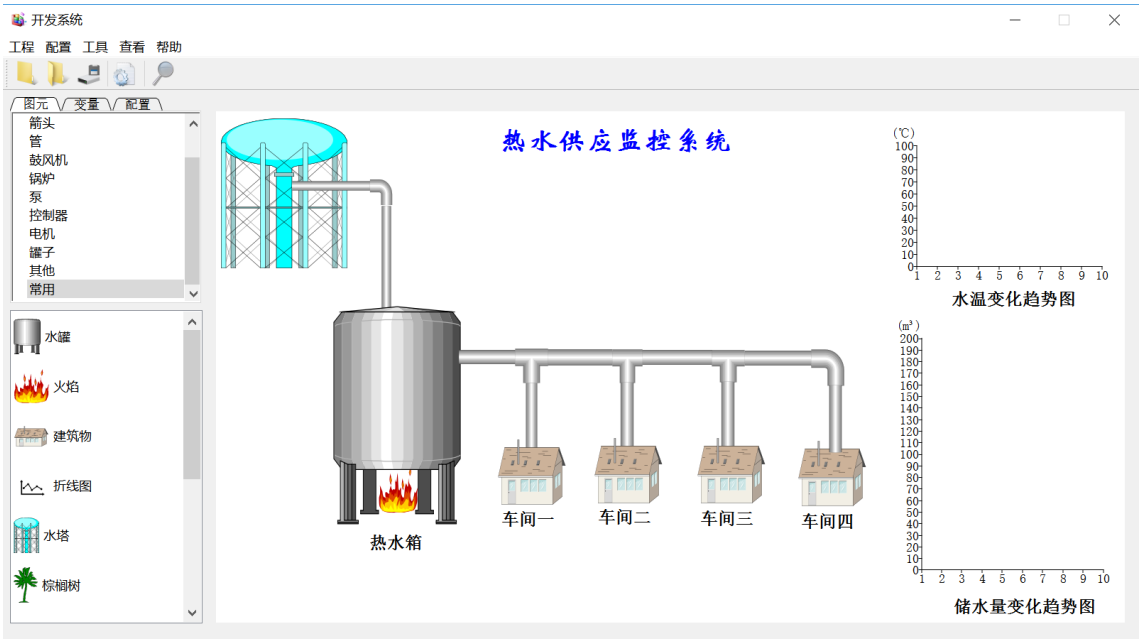

图 6-2 监控界面设计效果

6.3 测试过程及结果分析

本小结将对整个测试的过程和结果进行分析，主要介绍测试环境、测试流程和结果的分析。

6.3.1 测试过程

（1）测试环境

本组态软件的设计初衷是要支持多平台，因此测试方案也将在多个平台上实

施，具体参数如表 6-5 所示。

表 6-5 测试环境参数

开发环境	Qt 5.3.0	
测试环境	Windows 平台	Windows 10（64 位）
	Linux 平台	Ubuntu 16.04（64 位）

（2）测试数据

热水箱和四个车间将模拟生成十组数据进行测试，测试数据如表 6-6 所示。

表 6-6 测试数据

设备 / 组号	热水箱		一车间	二车间	三车间	四车间
	水温	储水量				
1	81.2	145.0	11.2	72.3	12.3	53.4
2	83.0	142.2	17.8	74.3	13.4	46.7
3	85.2	134.3	15.4	75.6	14.3	34.5
4	80.1	130.0	23.4	76.2	15.4	51.2
5	86.1	150.7	32.6	77.8	15.2	53.4
6	66.1	167.0	31.2	79.5	16.0	56.4
7	80.0	172.4	9.2	68.7	20.3	55.3
8	87.1	175.6	13.4	70.5	21.4	56.7
9	88.3	192.3	34.4	80.1	25.3	60.2
10	82.6	187.3	44.3	92.1	24.9	63.5

在表 6-6 中，每一种设备产生的十组数据中存在数据不在正常值范围的情况，因此会产生报警。

6.3.2 测试结果分析

整个系统分为两个部分，监控系统和下位机（即热水箱和车间），热水箱和车间通过模拟程序进行模拟，产生的数据如表 6-6 所示。

（1）运行效果

组态软件可以在多个平台上运行，因此将设计的系统分别在 Windows 平台和 Linux 平台运行，运行效果如图 6-3 和图 6-4 所示。其中折线图表示的是热水箱的温度和储水量的变化趋势，数据来源于表 6-6。

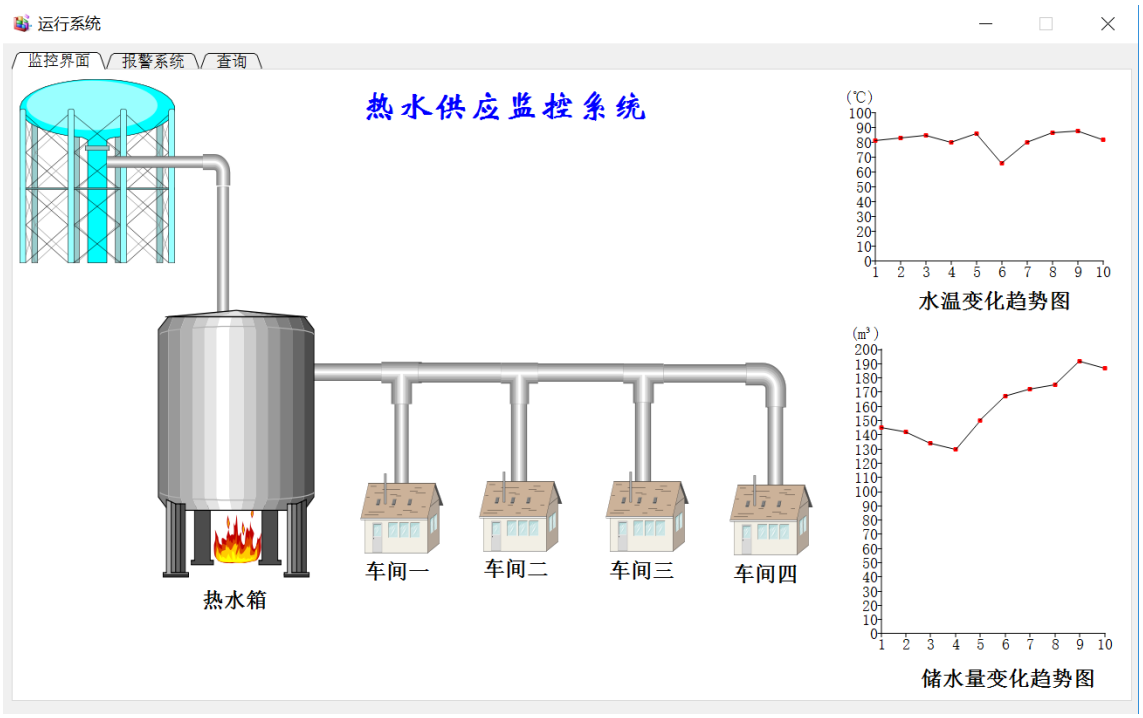

图 6-3 Windows 平台下运行效果

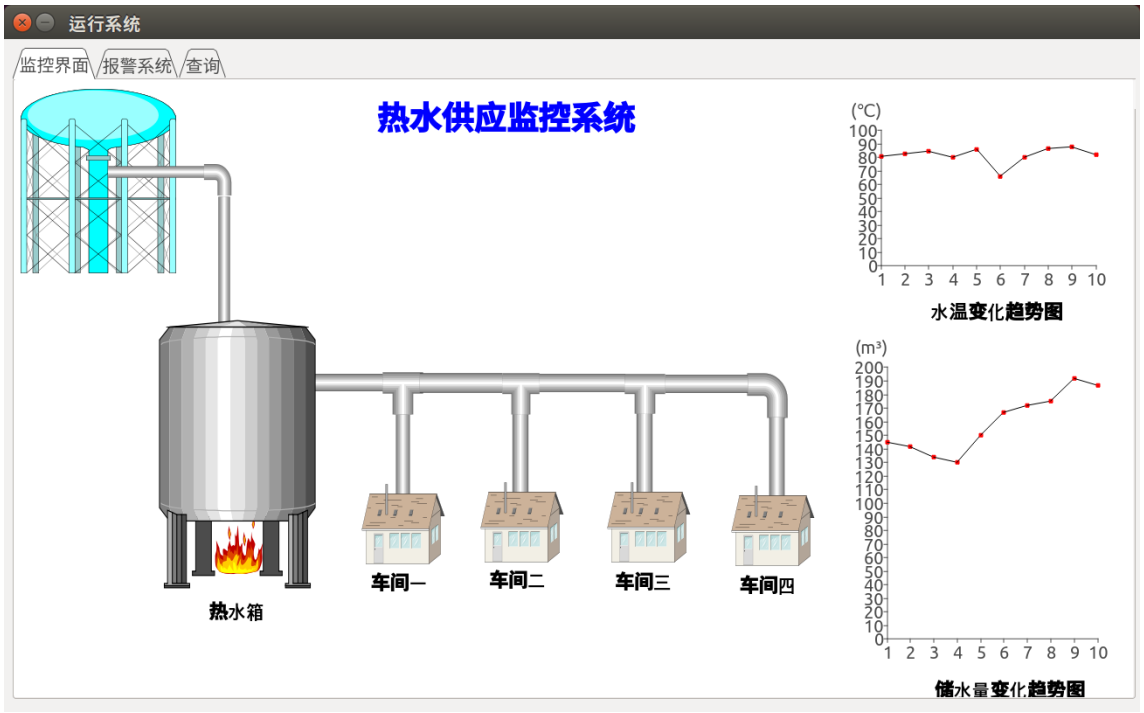

图 6-4 Linux 平台下运行效果

（2）历史数据

组态软件提供了对历史数据的查询功能，图 6-5 展示了对热水箱温度进行查询

后的结果。

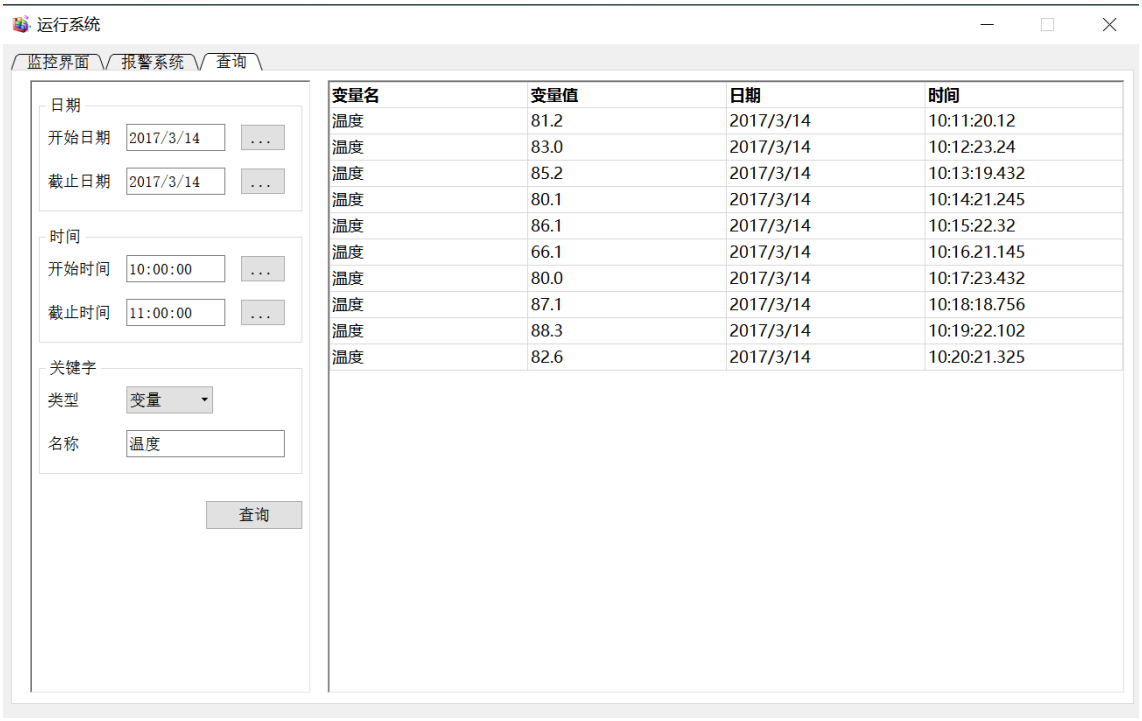

图 6-5 历史数据查询

（3）报警系统

由表 6-1 和表 6-6 可知，将会产生六条报警数据，分别是如下情况：

- 第二组数据，四车间水流量不在正常范围。
- 第三组数据，四车间水流量不在正常范围。
- 第六组数据，热水箱水温不在正常范围。
- 第七组数据，一车间和二车间水流量不在正常范围。
- 第九组数据，热水箱储水量不在正常范围。

从图 6-6 中可知，上述情况在报警系统中都有记录，由于所有测试在最后都恢复为正常情况，因此所有警报都被解除了，在报警系统中只能看到最近报警情况，而警报则为空。

图 6-6 报警系统界面

上述测试过程和运行过程对组态软件的关键模块都进行了相关测试，并且系统能够稳定的运行，说明该组态软件的基本功能是完善的。

6.4 本章小结

本章通过构造一个热水供应监控系统对本课题设计的组态软件进行了测试，对图元模块、报警系统和系统配置等方面都进行了验证，所得结果与期待结果非常接近，证明本组态软件的设计与实现是较为成功的。

第七章 总结

本文基于 Qt 这个 C++应用程序开发框架设计了一款轻量型的组态软件，它具有使用成本低、可跨平台和系统开放等优点。组态软件一般由三个部分组成：开发系统、运行系统和工程文件。本文详述了这三个部分的设计与实现，解决了设计过程中遇到的难题。

对于组态软件的开发系统，本文主要描述了图元模块、变量配置模块、系统配置模块和报警系统模块的设计与实现。在图元模块的设计中，充分利用了 Qt 提供的相关图形处理的工具，包括对于矢量图的显示和图形的绘制。同时图元的设计秉持开放的原则，为用户提供了扩展图形库的方法和接口。在变量配置模块和系统配置模块的设计中，主要考虑了系统和实际需求的易变性，因此设计的目标是为了达到灵活的配置。报警系统设计的过程中，实现警报的快速人机交互是设计的终极目标。

对于组态软件的运行系统，本文主要描述了交互界面模块、控制模块、通信模块和历史数据处理模块四个模块的设计与实现。在交互界面模块的设计中主要完成了图元加载和图形刷新等功能。在控制模块的设计中主要针对如何管理变量以及如何更新变量进行了详细阐述，同时讨论了变量的更新如何影响其它模块的工作。在通信模块的设计中，主要描述了数据的传输流程，同时描述了数据的封装格式与解析方法。在历史数据处理模块中，简单介绍了与数据库相关的操作。运行系统的设计难点主要在于多个模块之间的通信与同步，为此本文采用了 Qt 的信号/槽机制和线程间同步的相关机制来实现模块间的交互。

对于组态软件的工程文件，本文详细描述了工程文件的数据组织格式，并且介绍了数据的存储与提取。

最后本文设计了一个应用场景，尽量全面地测试了该组态软件的各个模块，取得了满意的测试结果，因此本文设计的轻量型组态软件是成功的。

但是该组态软件也存在一些不足，首先图元库比较少，由于图元是与具体行业相关的，因此设计组态软件的时候没能全面地考虑，但是相信后续经过不断完善能够得到一个丰富的图元库；其次支持的通信方式比较少，由于考虑到该组态软件需要跨平台运行，因此采用的通信方式都是 Qt 所支持的，如果后续能够优化驱动，使得能在多平台上运行，那么丰富各种通信方式也是可能的；最后就是变量的属性不够全面，支持的功能不够多。然而该款组态软件作为一款轻量型组态软件，其功能还是足够的。

致谢

时光匆匆，转眼三年的研究生生活已经接近尾声，这三年里从老师同学那里学到了很多，在这里想向他们致以衷心的谢意。

首先，非常感谢我的指导老师*****教授，*****在科研上一丝不苟，不容得我们有半点马虎，因此在此过程中，我学到了如何克己，如何保持耐心。特别是在论文写作期间，他会要求我们周期地向他报告进展，同我们讨论，然后提出相关建议，这也使得我能够在规定的时间内完成课题的研究。在生活上，崔老师则表现出对学生的关爱，他总是会与我们进行私下的谈话，关心我们的生活，为我们指点迷津。

其次，我得感谢我的师兄师弟师妹们，他们时常与我交流，扩展了我的思路，同时他们也给我带来很多欢乐，让我的生活更多姿多彩。特别的，我得感谢同届好友，我们常常一起讨论学习，专研课题，在学习和生活上互帮互助。

当然，我最应该感谢的还是我的父母，他们为我提供求学的环境，同时在精神上给予我支柱，无论何时他们都是我的依靠。

最后，向以上我提及的所有人表示衷心的感谢！同时感谢各位参与论文评审的专家和教授。

参考文献

[1] 尹阿峰.浅谈现代工业控制技术[J].科技资讯,2011,第 8 卷(第 1 期):102

[2] Galloway Brendan, Hancke Gerhard P.Introduction to Industrial Control Networks[J].IEEE Communications Surveys & Tutorials,Second Quarter 2013,Vol.15(2):860-880

[3] 王常力.分布式控制系统的现状与发展[J]. 电气时代,2004,第 1 期:82

[4] 鲍金雨.工业控制自动化技术的现状与发展趋势[J].科技与企业,2016,(第 12 期):90

[5] 白云飞.DCS 技术及其发展展望[J].机械管理开发,2011,第 26 卷(第 3 期):202-203

[6] Reisacher K.Supervisory control and data acquisition system[J].Springer Science & Business Media,1999,Vol.116(1):40-49

[7] Hieb J L.Security hardened remote terminal units for SCADA networks[J].Dissertations & Theses - Gradworks,2008:271-276

[8] Tesfahun Abebe, Bhaskari D.A SCADA testbed for investigating cyber security vulnerabilities in critical infrastructures[J].Automatic Control and Computer Sciences,2016,Vol.50(1):54-62

[9] Turc Traian.SCADA Systems Management Based on WEB Services[J].Procedia Economics and Finance,2015,Vol.32:464-470

[10] Combs Larry.Cloud Computing for SCADA[J].Control Engineering, Dec 2011

[11] 万勇.一种基于云计算技术的 SCADA 系统设计[J].高压电器,2013,第 49 卷(第 7 期):89-91

[12] Exforsys.The History of Data Modeling[OL].http://www.exforsys.com/tutorials/data-modeling/the-history-of-data-modeling.html:Exforsys Inc,11 January 2007

[13] 熊伟.工控组态软件及应用[M].北京:中国电力出版社,2012,1-20

[14] Shahzad Aamir, Lee Malrey, et al.The protocol design and New approach for SCADA security enhancement during sensors broadcasting system[J].Multimedia Tools and Applications,2016,Vol.75(22):14641-14668

[15] Vasileios Deligiannis, Stamatis Manesis.Implementing global automata in PLCs based on the IEC 61131-3 programming norm[J].Int. J. of Industrial and Systems Engineering,2010,Vol 6 Issue 2:163-186

[16] 吕品超, 侯立刚, 孟帅.小型组态软件图形界面系统的研究与开发[J].自动化仪表,2010,(第 6 期):14-16

[17] 陈煜强, 郑耿, 刘国平, 许志艳.组态软件报警事件模块组件化设计[J].计算机应用与软件,2006, 第 23 卷(第 1 期):64-66

[18] Rezai Abdalhossein, Keshavarzi Parviz, Moravej Zahra.Key management issue in SCADA

networks: A review[J].Engineering Science and Technology, an International Journal,February 2017,Vol.20(1):354-363

[19] 李郴，孙继平，王瑛.组态软件中功能模块设计[J].煤炭工程,2000,(第 10 期):17-19

[20] 王克亮.监控组态软件的相关技术发展趋势[J].山东工业技术,2015,(第 1 期):185

[21] Pan Hongxiang.The Design and Implementation of Information Integration Platform Configure Software Based on Qt[C].2011 Asia-Pacific Power and Energy Engineering Conference, March 2011, 1-4

[22] Design patterns for object-oriented software development by[J].Communications of the ACM, Nov 1994, Vol.37(11):8

[23] 王淑红.工控组态软件及应用[M].北京:中国电力出版社,2016,4-20

[24] 王亚民等编著.组态软件设计与开发[M].西安:西安电子科技大学出版社,2003,6-9

[25] 朱尚明，郎文辉，王俊.一种基于图元监视组态软件的设计与实现[J].基础自动化,2001,第 8 卷(第 6 期):30-32

[26] 张杰，关永.工控组态软件的可视化[J].中国图象图形学报(A 辑),2002,第 7 卷(第 2 期):201-204

[27] 王光.基于 C#的监控组态软件开发[D].哈尔滨：哈尔滨工业大学,2012,26-27

[28] 纪陵.基于 QT 的跨平台 SCADA/EMS 一体化系统设计和应用[D].南京:东南大学,2013,21-23

[29] An Jiyu, Yet Tao, Li Yongjun.Design and Achievement of Linux SCADA Based on Qt[J].Journal of Electron Devices,2006,Vol.29(2):532-535

[30] 邹茜.基于 Qt 的 GUI 应用程序开发[J].科技信息,2010,(第 18 期):I0200-I0201

[31] Brun R., Fine V..Cross-platform approach to create the interactive applications based on ROOT and Qt GUI libraries[J].Nuclear Inst. and Methods in Physics Research, A, 2004, Vol.534(1):94-97

[32] 王书平.组态软件若干关键技术研究[D].西安:西安电子科技大学,2012,14-16

[33] 缪雨润.基于 Qt 的图形用户界面的研究与实现[D].南京:东南大学,2015,5-6

[34] 杨晨.基于 Qt 的监控组态软件的研究与开发[D].大连:大连理工大学,2015,10-16

[35] Jasmin Blanchette, Mark Summerfield.C++ GUI Qt4 编程 第 2 版[M].(闫锋欣，曾泉人，张志强译).北京:电子工业出版社,2013,142-147

[36] 李学忱.工控组态软件运行平台的研究与实现[D].大连:大连理工大学,2005,44-47

[37] 赵保涛.基于 LINUX 的控制系统组态软件研究与开发[D].保定:华北电力大学,2014,16-19

[38] 陈丽如.计算机中的位图和矢量图研究[J].科技创新与应用,2015,(第 1 期): 56

[39] 刘文彪，詹成国.跨平台 SCADA 图形系统的分析与设计[J].华电技术,2008,第 30 卷(第 8

期): 20-23

[40] 钱琳.基于 VC 的组态软件设计[D].大连:大连理工大学,2011,33-36

[41] Hsueh Nien-Lin, Kuo Jong-Yih.Object-oriented design: A goal-driven and pattern-based approach[J].Software & Systems Modeling,2009,Vol.8(1):67-84

[42] Beck Kent.Design Patterns: Elements of Reusable Object-Oriented Software[J].IBM Systems Journal, 1995, Vol.34(3):544-545

[43] 卞劲松.基于组件的多媒体软件撤销重做设计[J].计算机工程,2004,第 30 卷(第 A1 期): 388-390

[44] 周欣然.组态软件的设计[D].长沙:中南大学,2003,20-23

[45] 钱刚，吴坚.HMI 组态软件报警系统功能拓展的设计及实现[J].自动化仪表,2014,第 35 卷(第 z1 期):13-15