摘要

为了克服传统工业控制软件柔性差、开发周期长、不易维护、重复使用率低且价格昂贵等缺点，本文研究监控组态软件的关键技术，在 Windows 环境下，以 VC7.0 作为主要开发工具，完成了其中图形组态模块、数据采集存储模块、数据存储模块的设计与实现。

本文采用设计模式中工厂模式来设计组态软件图形组态模块。设计模式是面向对象技术的实际应用。在实际软件项目中，工厂模式(Factory Method)是应用最广泛的设计模式。工厂模式定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂模式的应用使本系统的结构精巧简洁、易于理解。在以后维护中，很容易找到需要修改的地方，减少了维护的工作量。在需要添加新增功能时，也只需编写新的功能的代码，而不用去修改以前的逻辑，减少了新添加功能时带来的工作量。本文就图元的创建、保存、修改、绘图界面闪烁等项目中遇到的基本问题进行了详尽描述。

数据采集模块收集数据处理单元通过以太网发送上来的实时数据，只在运行状态时运行。它采集工业实时数据，放入内存以供 View 模块访问，如实时趋势。数据采集模块利用共享“内存映像文件”即 FileMapping 技术，解决组态软件与下位机数据处理单元的交互通信问题。数据采集模块还对采集来的数据按照历史库、SOE、报警等业务逻辑进行处理。

本文引入数据库缓冲访问技术，解决数据库访问慢和缓冲数据。数据库缓冲访问是通过多个进程可访问的队列来实现的。该队列利用内存映像文件实现，由于该队列有几个不同的线程会同时访问，本论文采用 Windows 内核对象：互斥量、信标实现同步。需要访问数据库的进程将数据库访问请求放入队列中，由另一个数据库访问模块进行实际的读取数据操作。该技术的应用，提高了数据库访问速度，提高了本系统可采集点的数目，可达到 4 万个点的采集。

关键词：集散控制系统，组态软件，图形组态，数据采集，工厂模式

Abstract

The traditional industrial control software has obvious shortcomings, such as weak flexibility, comparatively long development period, inconvenience to maintain, etc. In order to overcome those disadvantages, design and implementation of configuration software. This paper investigates the graphics module and Data Acquisition Module Design and Implementation.

In this paper, a design model of the factory model to graphic design configuration software configuration module. Object-oriented design model is the practical application of technology. In the actual software projects, factory mode (Factory Method) is the most widely used mode of the design. Factories used to create a model definition of the target interface, an example of the type of decision of which category. Application of the factory model of the structure of the system compact concise, easier to understand. In subsequent maintenance, it is easy to find the need to be amended, reduced maintenance workload. In the need to add new features, they only function of the preparation of a new code, rather than amending spent the previous logic, a decrease of the new features bring added workload. In this paper, graphic element on the creation, preservation, modification, graphics interface, and other items flashing the basic problems encountered in a detailed description.

Data Acquisition Module to collect data sent via Ethernet processing units of the real-time data, only to run in the running state. It industrial real-time data acquisition, View Add memory modules for the visit, such as real-time trend. Data Acquisition Module using shared "memory image files," namely FileMapping technology, and configuration software solution for the data-processing unit of interactive communications. Data Acquisition Module also on the data collected in accordance with the historical library, SOE, alarm processing, and other business logic.

This paper introduces a buffer database access technology, and slow to resolve database access data buffer. Database buffer visit by more than one process can access the queue to achieve. The queue memory image files used to achieve, as the queue several different threads will visit at the same time, this paper is based on Windows

kernel objects: Exclusive volume beacon synchronized. Need access to the database will process database requests for visits Add to queue from another database access module to carry out the actual data read operation. Application of the technology, improving speed access to the database, the system can increase the number of collection points, can reach 40,000 points collection.

Keyword: Configuration Software, Graphical Editor, Data Acquisition, Factory Method

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，论文中不包含其它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。

签名：李杰　　日期：2018 年 5 月 24 日

关于论文使用授权的说明

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）

签名：李杰　　导师签名：李毅

日期：2018 年 5 月 24 日

第一章 绪论

组态的概念最早来自英文“Configuration“，含义是使用软件工具对计算机及软件的各种资源进行配置，达到使计算机或软件按照预先设置，自动执行特定任务，满足使用者要求的目的[1]。

1.1 监控组态软件简介

在工业控制技术的不断发展和应用过程中，PC 机相比以前的专用系统具有的优势日趋明显。这些优势主要体现在：PC 的各种相关技术已趋成熟；由 PC 构建的工业控制系统具有相对较低的成本；PC 的软件资源和硬件资源日益丰富，软件之间的互操作性增强。在 PC 向工业控制领域的渗透中，组态软件占据着非常特殊而且重要的地位。组态软件是数据采集与过程监控的专用软件，是在自动控制系统监控层一级的软件平台和开发环境，能以灵活多样的组态方式(而不是编程方式)提供良好的用户开发界面和简捷的使用方法，其预置的各种软件模块可以非常容易地实现和完成监控层的各项功能，并能同时支持各种硬件厂家的 I/O 板卡,与高度可靠的工控计算机和网络系统结合，可向控制层和管理层提供软件、硬件的全部接口，进行系统集成[2]。

在开发传统的工业控制软件时，当工业被控制对象一旦有变动，就必须修改其控制系统的源程序，导致其开发周期过长；已开发成功的工控软件又由于每个控制项目的不同，而使其重复使用率很低，导致它的价格十分昂贵；在修改工控软件的源程序时，倘若原来的编程人员离去时，则必须同其他人员或新手进行源程序的修改，因而更是相当困难。通用工业自动化组态软件的出现，为解决上述实际问题提供了一种崭新的方法，它能够很好地解决传统工业控制软件存在的种种问题，使用户能够根据自己的控制对象和控制目的任意组态，完成最终的自动化控制工程[3]。

监控组态软件是面向监控和数据采集的软件平台工具，功能强大。它的发展是伴随着计算机技术的突飞猛进发展起来的。主要有以下两个方面的表现：

1、个人计算机操作系统日趋可靠稳定，实时处理能力大大增强；

2、个人计算机的软件及开发工具丰富，使组态软件的功能强大，开发周期相

应变短，软件升级和维护也比较方便。随着集散控制系统的蓬勃发展，组态软件的地位日益重要。组态软件被推到了自动化系统主力军的位置，成为工业自动化系统中的灵魂。

组态软件应该向更多的应用领域拓展和渗透。目前的组态软件均产生于过程工业自动化，很多功能没有考虑其他应用领域的需求。例如：化验分析（色谱仪、红外仪等，包括在线分析）、虚拟仪器（例如 LabView 的口号是 The Software is the Instrument）、测试（如测井、机械性能试验、碰撞试验等的数据记录与回放等）、信号处理（如记录和显示轮船的航行数据：雷达信号、GPS 数据、舵角、风速等）。这些领域需要良好的人机界面，但是由于现有组态软件为这些应用领域考虑得太少，不能充分满足系统的要求，因而目前这些领域仍然是专用软件占统治地位。随着计算机技术的飞速发展，组态软件应该更多地总结这些领域的需求，设计出符合应用要求的开发工具，更好地满足这些行业对软件的需求，进一步减少这些行业在自动测试、数据分析方面的软件成本，提高系统的开放程度。

另外，当前国外组态软件占据着市场的大部分份额，但由于价格偏高，成套系统达几十万至几百万且界面西化，有时不能支持国内的一些硬件设备，同时国外组态软件通用性太强，不能满足一些特殊要求，对于中小型企业来讲，许多功能用不上，造成极大的浪费，另外由于国外产品的实现细节是保密的，他们的软件对我们来说是个黑匣子，安全上受制于人，具有很大的安全隐患。国产化的组态软件逐渐成为市场上的一支生力军。但总体上讲，由于资料来源缺乏，软件工程的组织薄弱等原因，使国产化组态软件的开放性、集成性、可靠性都有一定局限性，市场竞争力较弱[4][5]。

从 80 年代开始，由于个人计算机的普及，开始有人研究如何利用 PC 进行工业监控，同时出现了基于 PC 总线的 A/D、D/A、计数器等各类 I/O 板卡。当时有人在 MS-DOS 基础上用汇编语言或 C 语言编制带后台处理能力的监控组态软件，也有一些机构在实时多任务操作系统 iRMX86 或 VRTX 上做文章，但均未形成有竞争力的产品。随着微软公司 Windows 操作系统的普及，基于 PC 的监控组态软件才迎来了发展机遇。世界上第一个把组态软件作为商品进行开发、销售的专业软件公司是美国的 Wonderware 公司，它于 80 年代末率先推出第一个商品化监控组态软件 Intouch。此后监控组态软件在全球得到了蓬勃发展。

进入 90 年代以后，国内外推出了不少组态软件产品，有的是随集散系统一起推出的作为专用配套软件，有的是通用软件，如美国 Intellution 的 FIX、德国 SIMATIC 公司的 WinCC，美国 Wonderware 公司的 Intouch 的更新版本及国内的

组态王 KingView 等等[6][7][8]。

目前的监控组态软件中，国外的产品占据了绝大多数的市场份额，其中典型的组态软件有：

1、InTouch

Intouch 是美国 Wonderware 公司的产品，该软件的最大特点是 I/O 点数和最大画面数不受限制。Intouch 作为一个实时的人机界面程序的生成器，可以生成管理级别上的监控和数据采集程序，依靠菜单驱动在多种 Windows 环境下运行，它主要由两大部分组成：

(1)Windows Maker(应用开发环境)。用以建立窗口的图形显示，并定义与工业控制器、I/O 系统和其他窗口应用程序的连接；

(2)Windows Viewer(实时运行环境)。用以显示由 Windows Maker 建立的图形窗口。

2、Fix

Fix 是美国 Intellution 公司的产品，该软件既可单机运行，也可构成复杂的、功能强大的工厂控制网络系统，它是目前全世界范围内应用最为广泛的工控组态软件。Fix 是一个真正模块化的工控软件，它提供了 10 多个基本功能模块和扩展功能模块，支持多种软件平台，如 Win3.X，Win95，Win98，Windows NT 及 OS/2 等，其人机界面功能特别强大，除具有一般的动态显示外，还能方便地实现画面漫游、局域缩放和在线拷贝以及网络环境下的报警处理。它的二次开发接口开放、完备，还提供了一个专门用于 I/O 驱动开发的软件包 ITK。其最新产品命名为 IFix，在 IFix 中 Intellution 提供了强大的组态功能，并在内部集成了微软的 VBA 脚本语言的开发环境。另外，Intellution 是 OPC 组织的发起成员之一。

3、Genie

Genie 是台湾研华公司的产品，它的主要特点为具有清晰、简洁的界面、丰富的用户使用工具。它主要包括 3 大部分：策略编辑器、图形生成器、实时运行系统。其中，策略构成部分是系统的关键，它完成设置各点扫描、计算、数据登录和监控任务；图形生成部分可制作人机界面；

运行管理系统主要实现系统运行时各任务的调配。另外，还有一个程序开发环境可提供类 C 语言和内嵌的 VBA 语言，以便用户开发特殊的功能。

4、CiTech

CiT 公司的 CiTech 是较早进入中国市场的产品。它具有简洁的操作方式，但其操作方式更多的是面向程序员，而不是工控用户。它提供的脚本语言类似于 C

语言，二次开发有一定的难度。一些原 DCS 的系统厂商，如 Rosemount，Honeywell 等公司也陆续推出了新型的、更开放的控制系统以及现场总线产品，并配以“开放的”组态软件。但这些组态软件的开放性主要是面对管理层，对下的控制一般仍只针对本公司产品，因此不能称为通用组态软件。这些专用的组态软件都有很强的特色，且价格不菲。

国产的组态软件产品逐渐被市场接受，应用比较成功的有组态王，开物，虎翼等。组态王是国内第一家较有影响的组态开发公司。它提供了脚本语言的支持，COM 技术的支持，OPC 技术的支持，另外也提供了大量的驱动程序。华富计算机公司的开物 2000，提供了完备的实时曲线、历史曲线、报警、数据报表等功能。开物内建 OPC 支持，提供面向对象的脚本语言编译器，支持 ActiveX 组件和插件的即插即用，并支持通过 ODBC 连接外部数据库。该软件同时提供网络支持和 WebSever 功能。国产的组态软件具有较强的价格竞争优势，但总的来讲，由于资料来源缺乏，

软件工程的组织薄弱，因此商品化的程度比较差，主要使用于一些小型和非重要的项目应用中。

从对以上组态软件的整体分析来看，目前组态软件具有以下特点[9]：

1、组件化结构

在 FIX 中，其组件化结构可让用户十分方便地插入 Intellution 公司提供的 32 位组件产品，再加上第三方组件程序，可集成一个完整的工业控制系统。FIX 推出的组件对象有：具有图形编辑功能的 SCADA 和 HMI，基于 PC 机的软逻辑控制软件 softlogic TM，网络服务器软件 WEB ServerTM，用户可根据工业现场需要集成所需的监控系统。

2、具有远程诊断和易维护功能

美国 FOXBORO 公司的 I/A 系统是一个新一代的开放型 DCS 工业控制系统，其 51 系列采用 SUIT 工作站和 Solaris 操作系统，它的网络连接采用通用的以太网，通讯协议采用当前较流行的 TCP/IP 协议，因此 I/A 系统可直接与符合 TCP/IP 协议的本地信息管理网连接，并可通过通讯接口和通讯网络与远程的局域网连接。PC 机可通过电话线和调制解调器与工厂内的 I/A 系统连接，成为 I/A 系统的一个工作站，调出 I/A 上的过程画面和数据，从而对 I/A 系统进行远程诊断和维护。

3、开放性

所谓开放性，是要求各厂家的产品具有互换性，互操作性，可扩充性并提供多平台支持。DCS 产品随着网络的标准化，逐渐具备了开放性。同时，通用商品

化软件包在DCS中得到广泛应用，许多DCS操作平台向Windows NT移植，以便于互通信息，新一代DCS普通采用动态链接库(DLL)，对象连接与嵌入(OLE)，结构化查询语言(SQL)，应用编程界面接口(API)等软件技术，解决了与商用软件的接口问题。以FIX，INTOUCH为例，它们均提供了工业标准接口、界面和通信技术，支持Windows NT，OLE，OPC，VBA，ActiveX和COM技术，用户不需要编写程序代码，就可以轻松享受这些功能，实现了系统的最大程度的开放。

1.2 组态软件在监控系统中的地位

在一个自动监控系统中，投入运行的监控组态软件是系统的数据处理收集中心、远程监控中心和数据转发中心，处于运行状态的监控组态软件与各种控制、检测设备(如PLC、智能仪表、DCS等)共同构成快速响应/控制中心[10]。控制方案和算法一般在设备组上组态并运行，也可以在PC上组态，然后下载到设备中运行，根据设备的具体要求而定，如图1-1所示。

监控组态软件投入运行后，操作人员可以在它的支持下完成以下六项任务[11]：

1、查看生产现场的实时数据库及流程画面；

2、自动打印各种实时/历史数据报表；

3、自由浏览各个实时/历史趋势画面；

4、及时得到并处理各种过程报警和系统报警；

5、在需要时，人为干预生产过程，修改生产过程参数和状态；

6、与管理部门的计算机互连，为管理部门提供生产的实时数据。

组态软件应该向更多的应用领域拓展和渗透。目前的组态软件均产生于过程工业自动化，很多功能没有考虑其他应用领域的需求。例如：化验分析（色谱仪、红外仪等，包括在线分析）、虚拟仪器（例如LabView的口号是The Software is the Instrument）、测试（如测井、机械性能试验、碰撞试验等的数据记录与回放等）、信号处理（如记录和显示轮船的航行数据：雷达信号、GPS数据、舵角、风速等）。这些领域需要良好的人机界面，但是由于现有组态软件为这些应用领域考虑得太少，不能充分满足系统的要求，因而目前这些领域仍然是专用软件占统治地位。随着计算机技术的飞速发展，组态软件应该更多地总结这些领域的需求，设计出符合应用要求的开发工具，更好地满足这些行业对软件的需求，进一步减少这些行业在自动测试、数据分析方面的软件成本，提高系统的开放程度。

另外，当前国外组态软件占据着市场的大部分份额，但由于价格偏高，成套系统达几十万至几百万且界面西化，有时不能支持国内的一些硬件设备，同时国外组态软件通用性太强，不能满足一些特殊要求，对于中小型企业来讲，许多功能用不上，造成极大的浪费，另外由于国外产品的实现细节是保密的，他们的软件对我们来说是个黑匣子，安全上受制于人，具有很大的安全隐患。国产化的组态软件逐渐成为市场上的一支生力军。但总体上讲，由于资料来源缺乏，软件工程的组织薄弱等原因，使国产化组态软件的开放性、集成性、可靠性都有一定局限性，市场竞争力较弱[12][13]。

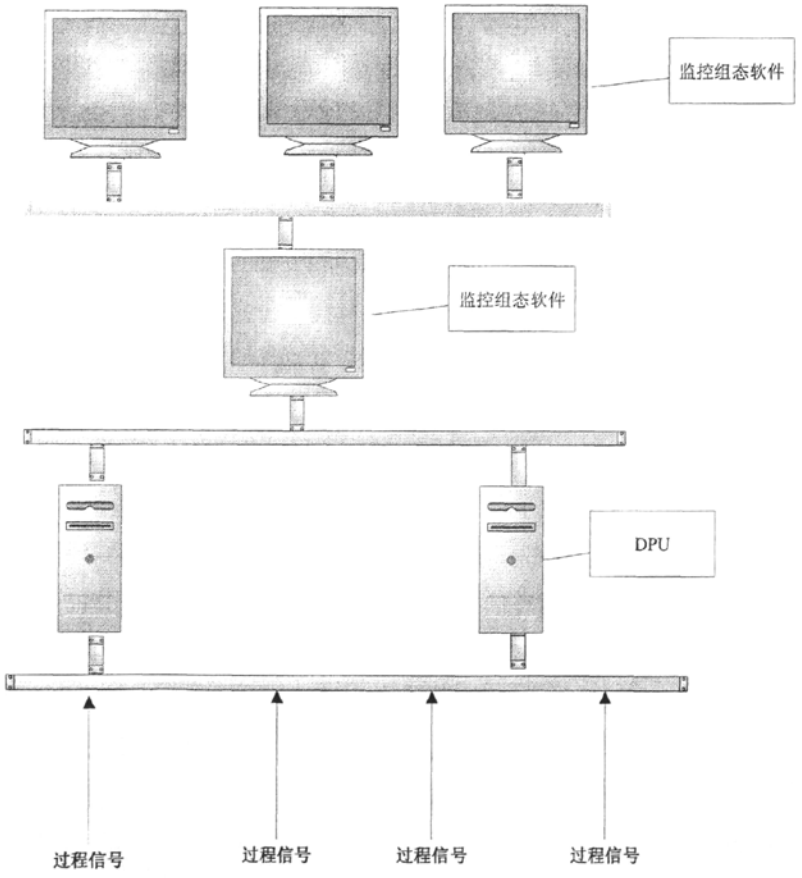

图 1-1 组态监控软件在自动控制系统中的地位

1.3 课题来源

本课题作为计算机学院与某大型公司合作项目工业组态软件设计的一部分，

是发电厂综合自动化系统的重要组成部分，用来对发电机组的运行进行监视，控制和管理。

1.4 课题所研究的主要内容

本论文工作为与某大型公司合作开发的组态软件，着重讲述了图形组态模块、数据采集模块、数据库访问模块的设计与实现。其主要工作有：

在第二章中采用设计模式中工厂模式来设计组态软件图形组态模块。就图元的创建、保存、修改、图形组合和分解、图形对象拾取判断、绘图界面闪烁等项目中遇到的基本问题进行了详尽描述；动画连接组态模块完成图形的动画属性，与实时数据库中定义的变量建立相关性的连接关系，作为动画图形的驱动源。动态属性与设备的 I/O 变量等相关，它反映图形大小、颜色、位置、可见度、闪烁性等状态的特征参数，随着表达式的值的变化而变化；

在第三章中叙述了该项目采用的数据库 Berkeley DB 的优点，以及对比传统大型关系数据库，Berkeley DB 更适合本项目。

在第四章中描述利用网络编程技术采集下位机发送到人机接口站的数据，并对这些数据进行相应的处理，比如报警、存历史库等。

第五章引入数据库缓冲访问技术，解决数据库访问慢和缓冲数据。数据库缓冲访问是通过多个进程可访问的队列来实现的。该技术的应用，提高了数据库访问速度，提高了本系统可采集点的数目，可达到 4 万个点的采集。

1.5 本章小结

本章就组态软件的重要性、课题背景做了说明。下面论文就组态软件中两个关键技术图形组态和数据存储进行研究。

第二章 图形组态模块设计

图形组态作为组态软件中用户直接面对的模块，要完成在系统和用户工程师之间沟通桥梁的作用，实现用户工程师绘制监控界面、显示监控图形、文件管理等功能，其重要性不言而喻。用户界面由若干图元组成。在图形组态中，图元一般分为两大类：基本图元和动态图元。基本图元是指直线、折线、圆、多边形等；动态图元则是指能够随着时间或者输入数据等条件而改变自身表现形式的对象[14]。

用户工程师通过图形组态模块组织图元、建立图形文件、形成用户控制和管理的直接界面[15]。这样用户工程师就可以在显示器前观察到这个流程的工作情况。

2.1 图形组态模块的功能

图形组态系统应该具有以下功能：

（1） 图形显示功能：能够在窗口中显示当前活动的图形。

（2） 图形编辑功能：

1） 用户工程师可以从系统中选择基本的图元工具，这些工具包括直线、折线、圆弧、矩形、圆、椭圆、多边形、文本、按钮。

2） 可以设置作图模式，定义画笔的宽度、颜色、样式、画刷、填充颜色、填充样式、图形的背景颜色。

3） 对显示在屏幕上的各种图元可以进行选中、移动、缩放、支持复制、剪贴、拷贝、删除等操作，可对图元在屏幕上的位置进行微调。

（3） 图形文件管理：在编辑时，图元对象是一个个分离的对象，通过指针将分离的对象连接在一起以便管理。通过此种方式可将建立的图形一编辑定义的顺序打开、修改、装载和存储在磁盘中。

2.2 图形组态设计思想

用户希望集散控制系统不但能按给定的生产工艺进行控制，还能在不影响生

产的情况下调整现有的工艺流程和操作界面，目前比较先进的 DCS 都有专门的图形界面生成工具，允许用户根据特定的生产工艺来生成图形操作界面。图形组态系统的设计是组态软件设计的重点，本文充分利用面向对象的思想对图形组态系统进行设计。

2.2.1 目前通用的图形设计方法

目前图形设计的方法大致可分为两大类：基于象素(点阵光栅)的方法与基于图元(矢量)的方法基于象素的图形界面设计，以象素为单位进行图形界面的显示和动态刷新，在 Windows 环境中，大部分作图软件的作图格式是基于象素的，大部分图形的存储格式也是基于象素的，如 BMP，JPG，PCX 等，合理地利用这些通用文件格式，并通过采用一定的算法，就能满足组态软件实时刷新的要求[16]。其中位图格式的图形界面设计一般是采用 Windows 提供的 API 函数，通过运用一定的数据压缩算法和双缓冲显示算法，加快位图的加载和显示速度，达到组态软件的实时性的要求。基于图元的图形界面设计中，以一个图元为单位，例如画圆、线、点等，通过记录用户的作图顺序，然后在需要显示时加以播放，并根据要求动态刷新其中某部分。一般讲，基于图元的图形界面显示速度和刷新速度比基于象素的要快，Windows 支持面向对象的作图格式的图元文件。这两种图形界面设计方法，基本上可满足组态软件图形界面设计，但随着计算机图形图象技术的飞速发展，用户对图形界面的设计要求也越来越高，例如要求画面越来越精细，动画更形象直观，这样如果仍采用上述两种设计方法，势必增加程序设计的难度与维护的工作量，给日后升级带来难度，而采用面向对象编程(OOP)的思想进行图形界面的设计，则能较好地解决上述问题。

2.2.2 面向对象设计思想

面向对象技术被认为是程序设计的一场革命，与传统的结构化程序设计相比较，有许多的优点[17]。面向对象技术力求更客观地描述现实世界，使分析、设计和实现的方法同认识客观世界的过程尽可能一致，它是一种从组织结构上模拟客观世界的方法，从组成客观世界的对象着眼，通过抽象，将对象映射到计算机系统中，又通过模拟对象间的相互作用、相互联系来模拟现实客观世界，描述客观世界的运动规律。面向对象技术以基本对象模型为单位，将对象内部处理细节封装在模型内部，重视对象模块间的接口联系和对象与外部环境间的联系，能层次

清晰地表示系统全局对象模型。其主要特征概括为：抽象性、继承性、封装性和多态性[18]。封装是指把数据结构同操作数据的函数组合在一起，使数据和过程实现了一体化，避免了传统程序设计中大量的数据传递，减少了数据误操作的可能性，提高了软件的可靠性和可维护性。继承使得类库中各个类按一定的层次组织起来，通过类层次把类进行了体系化。多态使得各个类允许一个操作有多个可实现的版本，通过利用类的多态性实现了灵活多样的类对象生成方式和功能函数操作，为用户提供了高度的灵活性[19][20]。

对象的本质是一种特殊的数据结构，对象的抽取过程大致为：将要由程序实现的若干事件按照性质的特征分类，由一组具有共同性质的对象组成类。面向对象程序设计着重解决类的问题，即解决同类对象的共同问题，概括这一组对象共同性质的数据和函数，封装成一个类型的对象。通过定义基本的类，使得物质世界中的对象被有机地分解，然后遵循一定的原则，用程序将这些模块组合、装配、扩充，这就按照用户的要求将现实世界的对象以软件形式实现面向对象的系统分析与实现的主要步骤有[21]：

1、面向对象的系统分割、识别对象：一般以分级的方法进行，先按系统较大的方面分割成若干个领域，再将领域分割成若干个主题，对每个主题又分割成若干个数据子类。相关性大的分割到一起，相关性小的则向其它方向分割。域、主题、数据子类的分割都遵循相关性的原则。

2、对象的抽象和定义：以主题为核心抽象得到的对象，不可能完全规范，由于不同主题之间的交叉和关联很多，必须对原始主题进行分析、归纳、抽象得出逻辑上相互独立的数据体系和专门的数据流，对应专用处理流程。

3、面向对象建模：对每一对象分别建立静态模型、动态模型和功能模型。静态模型用对象及其数据子类的数据字典表示，动态模型用对象内部数据处理图形界面表示，功能模型反映对象内部各数据子类间的数学关系。

4、对象模块设计及对象接口联系设计。

5、系统总体设计。

在本课题中，充分利用了面向对象的思想，较方便地实现了图形界面开发系统的框架设计，并利用类的封装性、继承性、多态性等特点来设计基本的图形控件，实现了图形界面的组态功能。

2.3 图形组态系统的设计

设计模式是软件设计过程中经常出现的问题，一个好的模式能使所生成的系统体系结构更加精巧简洁和易于理解。在程序开发过程中，设计模式的选择非常重要，软件领域中的设计模式为开发人员提供了一种使用专家设计经验的有效途径。所有结构良好的面向对象的软件体系结构中都包含了许多经典模式。在面向对象的编程中，软件编程人员更加注重以前的代码的重用性和可维护性[22]。而软件设计模式选择和应用的是否恰当，正是评判一个面向对象的软件系统质量好坏的重要标准。

2.3.1 工厂模式

在实际软件项目中，工厂模式(Factory Method)是应用最广泛的设计模式之一。工厂模式定义一个用于创建对象的接口，让子类决定实例化哪一个类。类工厂是一个生产不同对象的类，并将不同的类对象作为接口返回。即工厂模式可以根据不同的条件产生不同的实例。当然这些不同的实例通常属于相同的类型，有共同的父类，工厂模式把创建这些实例的具体过程封装起来了，简化了客户端应用，使得将来做最小的改动就可以加入新的待创建的类。工厂模式真正的目的在于可以灵活的、有弹性的创建不确定的对象。工厂模式的结构图如图 2-1 所示。

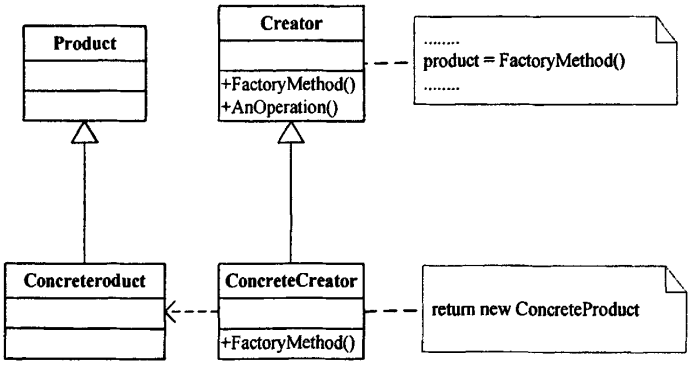

图 2-1 工厂模式结构图

其中 Product 定义工厂方法所创建的对象接口，ConceteProduct 实现 Product 接口，Creator 声明工厂方法，该方法返回一个 Product 类型对象。可以调用工厂方法实现一个 Product 对象，ConcreteCreator 方法重定义工厂方法以返回一个

ConcreteProduct 实例。工厂模式设计思想适用于下列情况：

1、当一个类不知道其必须创建的对象的类的情况；

2、当一个类希望由其子类指定所创建的对象的时候。

鉴于工厂模式的上述特征，并结合组态软件图形系统的自身特点和要求，采用工厂模式来实现本系统的主体架构。

2.3.2 基于工厂模式的类设计

在图形组态系统中，根据系统的特点，设计图元基类 CDrawMeta 以及图元绘制 CDrawTool。其中，图元基类抽象了所有图元的属性及相关操作，下设直线、矩形、椭圆、圆、控件等图元子类，其层次结构如图 2-2 所示。

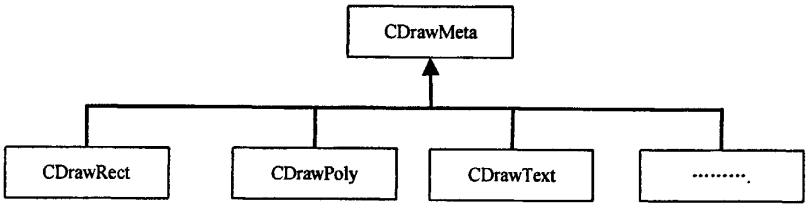

图 2-2 图元层次结构图

图元绘制工具类用于处理绘图时坐标的记录和鼠标事件的处理，并创建新的图元实例。下设子类分别用来处理矩形、多边形、文本等不同类型的图元，其层次结构如图 2-3 所示。

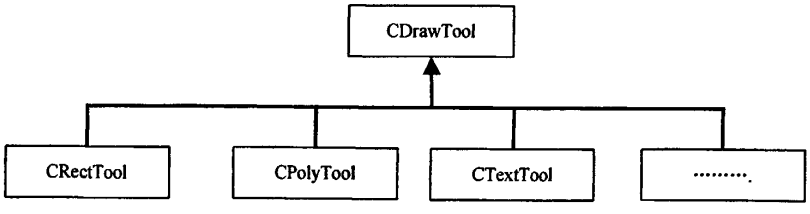

图 2-3 图元绘制结构图

图元基类 CDrawMeta 和图元绘制类 CDrawTool 共同构成了工厂模式。对应于工厂模式，CDrawMeta 为 Product，也即需要创建的实例的抽象类，而 CDrawRect、CDrawPoly 等组成的子类(ConcreteProduct)；CDrawTool 为 Creator，也就是抽象创建器的接口，具体的创建器(ConcreteCreator)由 CRectTool、CPolyTool 等子类实现。对应的结构如图 2-4。

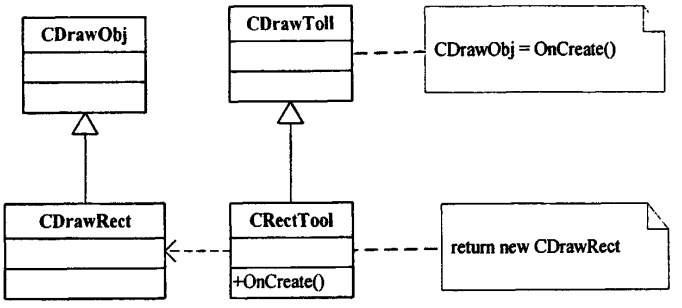

图 2-4 系统类关系模式

2.4 图形组态模块的实现

2.4.1 基本图元分类

组态软件的图元就用途而言，分为静态和动态两部分[23]。静态部分用来模拟监控现场没有设置相应测点的背景对象，比如车间的设备布置、房间等。这些图形元素的设置有利于给操作人员建立一种现场感。在设计静态图元时通常引入现场照片作为监控背景。监控对象本身的静态部分则一般可用软件提供的图形构造工具产生，以便于在设计监控画面时动态调整监控对象在背景中的布局，并方便在监控系统的投入运行后，如果被控对象发生改变后对相应的图形元素进行再编辑。

动态部分的任务是反映被控对象的变化。比如从屏幕上了解罐的液位、管道内流体的流向等，一般需要直接在屏幕上显示数据变化，同时辅以直观的动画来模拟现场，这部分功能可以通过基本矢量图形。

基于以上的分析，将系统的图形元素分为基本矢量图元、位图两大类[24]。

2.4.2 图形元素类的实现

在对图形元素分类的基础上，基于面向对象的思想，对要设计的图形元素进行抽象，设计出元基类 CDrawMeta，一般图形元素基类包含下列函数和变量：声明支持序列化、构造函数、得到点的位置函数、得到编辑图元边框的矩形函数、设置线条的颜色、设置填充的颜色、绘制图元、在不同状态下显示图元(跟踪位置状态)、移动图元到新的位置、测试鼠标是否选中图元、矩形内的逻辑判断、移动图元句柄、打开新的界面、图元属性修改、删除对象、图元动态行为修改、设置

图元对象的名称、获取图元对象的类型、定义图元的位置、指向文档类的指针变量、与数据库相关的模拟量或者开关量，且其名称和区域可以通过其成员函数来录入、定义一个唯一的ID，便于解释器以后调用、判断笔是否被选中、判断画刷是否被选中、逻辑画刷,填充图元、逻辑画笔,绘制图元、画笔颜色、画刷颜色、画笔大小、画笔形式等等。

在基类定义中，多次使用了虚函数定义的目的是为了简化程序的设计，在程序的使用时只需指定为基类类型，运行时动态判断类的类型并执行对应类的成员函数。而使用函数的重载也可以在类定义中使用同名函数，但使用时必须明确指定类的类型。由图元基类 CDrawMeta 派生出其他具体图元类，包括 CDrawRect、CDrawText、CDrawPoly 等，各派生类实现基类全部或部分接口功能。

2.4.3 图元的保存

用户绘制好的基本图元，需要将它以文件形式保存，在绘图模块中，图元信息应保存在内存中以便用户对图元进行动态修改。一般采用的保存方法有以下几种[25]：

（1）数组方式：该方法是用固定的存储空间来保存每个图元的数据结构，产生的图形文件中所有的数据结构都保存在一个数组中。打开和重绘时只需将数组元素顺序地读出。这种方式简单方便，实现起来比较容易，不过存在插入图元和删除图元不方便的问题。并且在程序运行时需要在内存中分配一块连续的、足够大的存储空间，如果图元比较多，系统运行效率会降低。

（2）链表方式：该方法是在一个用链表来存储图元的数据结构，不同于数组方式的是各个图元的物理地址和他们的顺序没有关系，插入图元和删除图元实现起来比较方便。不过，链表是在程序运行的时候动态地在内存中保存图元属性的工具。

（3）关系数据库文件保存：该方法是用一个通用的关系数据库表来保存图元的属性和信息，使用通用的数据库文件打开方式来打开文件。修改、删除和查询等功能的实现可通过直接调用数据库提供的接口函数，不用重新设计接口函数，非常方便。但是系统开销大，时间慢。

本系统采用的方法：文档视图分离是 VC 编程的一大特点，可以利用 VC 的这一特点，在用 MDI 窗口实现图形编辑功能时在视图对应的文档中保存图元基本信息，重绘时只需将相应的文档调出即可[26]。结合链表在内存中灵活的存储方式，

实现对图元信息的保存。这样在打开时直接打开文档读取基本图元的基本信息，然后对图元对象进行重绘即可。在打开图元文件时，涉及到对保存链表的恢复与根据链表对图元进行重绘。这样便于用户对保存的图形文件的图元重新修改和编辑。具体实现策略：在文档类中将实例指针保存到成员变量m_objects中，m_objects的类型说明如下：

typedef CTypedPtrList<CObList, CDrawMeta*>CdrawObjList;

2.4.4 图元的创建

在程序中，图元的创建主要分为下面三个步骤：

1、创建新的图元实例。

2、跟踪鼠标移动修改图元，获得所见即所得的视觉效果。

3、确定并保存新建的图元实例。

此三个步骤都是结合鼠标操作来完成，在VC++中主要的鼠标事件如表所示[27]

表 2-1 鼠标事件

事件名称	消息名称	功能描述
OnLButtonDown	ON_WM_LBUTTONDOWN	鼠标左键单击触发
OnMouseDown	ON_WM_MOUSEMOVE	鼠标移动触发
OnLButtonUp	ON_WM_LBUTTONUP	鼠标左键弹起触发

下面以绘制矩形图元为例，概述整个图元的绘制过程。

第一步：创建新的图元实例

首先通过鼠标左键在工具栏中选择图元类型，当鼠标左键在绘图区按下时，创建图元类实例，将实例指针保存到成员变量m_objects中。相关流程图如图2-5所示。

第二步：调整图元

当进入图元移动状态后，图元将随着鼠标的移动而随时移动。该过程在CDrawTool的OnMouseMove事件程序中实现[28]。使用调整图元可以获得动态的图元创建效果，实现所见即所得的功能，使用户在绘图时能立即看到绘图效果，增强软件的友好性和互动性。实现调整图元功能的逻辑是：

改变当前矩形的右下角坐标；

改变鼠标的形状；

改变鼠标的形状；

进行拖动处理；

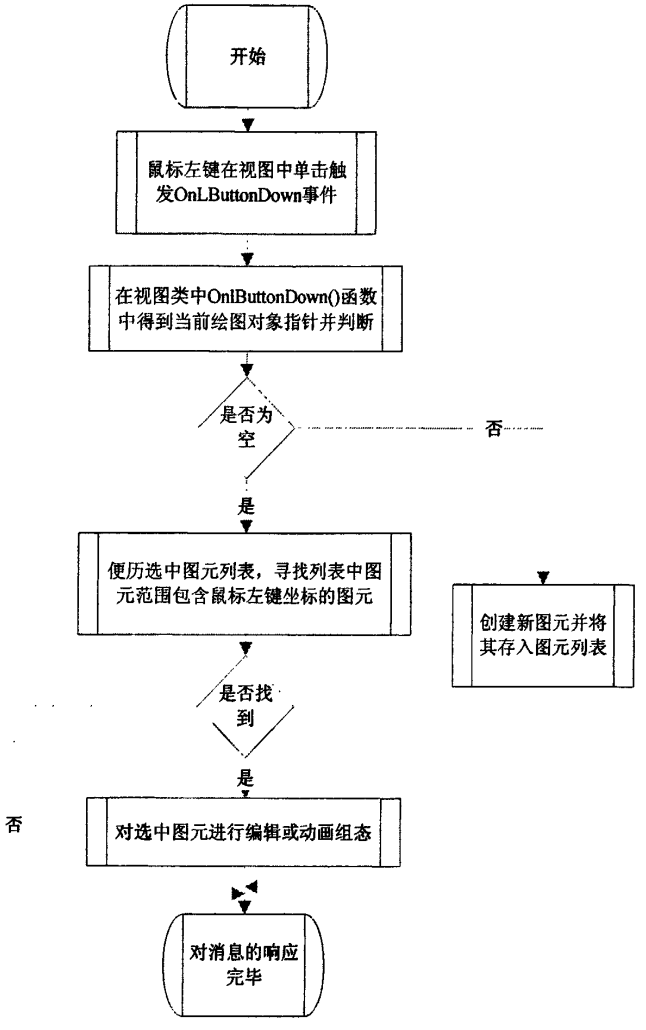

图 2-5 创建图元流程图

第三步：保存数据

当用户抬起左键时，代表已完成图元绘制。程序需保存图元数据，如果没有绘制成功，则恢复绘图前的程序设置并更新视图显示。在 CDrawTool 的 OnLButtonUp 中逻辑步骤如下：

释放鼠标锁定

如果没有进行绘制则进入选择状态并在 CRectTool 类的 OnLButtonUp 中执行如下代码；

如果没有进行绘制则删除链表对象，并恢复绘图前的设置；

左键弹起，保存绘制好的图元；

2.4.5 典型图元编辑操作实现

1、图元检取

图元检取的过程也即通过鼠标选中图元的过程，图元的检取方式分为鼠标点击检取和矩形区域检取两种。系统的检取方案与图元的绘制方法紧密相联，图元的绘制过程中除不规则多边形的所有图元都有外接矩形。

在系统中，对于鼠标点击检取，采取的思路为：首先设定标志为检取标志，然后判断鼠标在视图中点击是否在任一图元的外接矩形内，(对于多边形，则判断是否在其封闭区域内)，若在，则该图元的指针存入被检取图元的链表中。矩形区域检取，是指用户按下鼠标左键在视图中移动，且移动过程中左键一直处于按下状态，即认为开始一个矩形区域检取[29]。鼠标左键释放后，以起点和终点为两个对角顶点形成的一个封闭矩形，如果图元的外接矩形(多边形为其封闭区域)与该封闭矩形有任一交集，则该图元视为被检取，并将其指针存入被检取图元链表中。

2、图元的缩放

任一图元都有若干个柄(handle)，柄在程序中通过小矩形来实现。在对图元的编辑过程中，就是通过拖动这些柄来实现缩放功能。图元被选中后，这些柄将显现出来，对应于不规则多边形，柄是多边形的顶点；而其它图元则有固定的 8 个柄，分别对应其外接矩形的四个顶点和四条边的中点。

拖动每条边的柄可以改变边的位置和与这条边垂直两条边的长度，拖动顶点的柄可以同时调整相交于该点的两条边的长度和位置。通过 PtInRect 函数判断鼠标坐标是在选中图元四周的哪个小矩形框内。

图元的缩放过程中设计的鼠标事件主要是 WM_MOUSEMOVE。当鼠标在被检取图元上移动。则光标调整为相应的形状，并进入可缩放状态。

2.4.6 图形组合和分解

用户在对图元的编辑过程中，需要对几个图元整体采用移动、删除等操作，软件除了提供全部选中之外，还提供组合和分解功能。具体操作过程，几个图元被选中以后，从菜单命令或工具栏按钮，右键快捷菜单选择“组合”，即可实现组合功能，此后这几个图元将作为一个整体对象供用户使用。用户选中组合后的图

形，通过菜单或工具栏按钮，选择“分解”，即可将组合的图元还原为原有的多个图元。当然用户可以多次使用组合功能，即组合后的图形再和其它图元组合，生成一个大的组合图形。依次类推，可以不断组合。实现方法，设计一个组合类CCombineObj，该类继承于基类CDrawMeta，同时又是由其它图元类组合而成。图形组合流程如下：

```
创建组合类对象指针；
在选中的图形元素里，得到处于链表中的最前面对象的位置信息；
While(指针)
{
    逐个得到被选中的图形元素指针；
    逐个将被选中图形元素加入组合类对象链表的尾端；
    在文挡类中找到被选中的图形元素的位置；
    文档中删除被选中的图形元素；
}
将组合类对象加入到文档类的链表中；
清空选择集；
设置文档修改标志；
更新视图；
```

图形分解函数流程如下：

```
在选中的图形元素里，得到处于链表中的最前面对象的位置信息；
while(指针)
{
    逐个得到被选中的图形元素指针；
    If(是组合类对象)
    {
        转换为组合类对象指针；
        在组合类对象链表中，得到处于链表中最前面对象的位置信息；
        定义基类对象指针；
        while(组合类中图形头指针)
        {
            逐个得到组合类中的图形元素指针；
            把从组合类得到的图形元素指针加入文档中；
```

```
            }
            在文档 m_objects 得到组合类对象的位置信息;
            删除文档中组合类对象;
        }
    }
    文档修改标志;
    更新视图;
```

2.4.7 图形对象拾取判断

图形对象的拾取操作方法有多种，本文主要采用点选和区域选中两种方法。图形对象拾取功能的关键是确定图形对象边界矩形以及判断鼠标点是否在图形对象边界矩形一定的范围内。图元的边界矩形是指与图元对象外接的最小矩形，各种图元对象的边界矩形都可以从它的特征点计算出来。具体地讲，就是判断鼠标位置与图形对象的边界之间的距离是否在给定的识别精度范围之内。在本文设计中，识别精度是可以改变的。图形对象拾取的操作实现需要下面的代码：

1)用于设置选择模式的代码。

2)使用 CDrawMeta 类和它的派生类中实现图形选择判断代码。

3)判断一个图形是否被选中的代码。

4)绘制被选中的图形对象代码，高亮显示该对象的边界矩形，本文采用一个红色的矩形表示该图形对象被选中。

（1）图元的点选

点选图形的方法是根据鼠标点的位置来判断图元是否被选取。在窗口中点击鼠标左键，得到鼠标点位置坐标，将这个坐标转换为实际坐标，通过调用 MFC 中的 CRect 类的 PtInRect（CPoint）函数实现。如果在图元的边界矩形内，则可以精确判断是否被选中。以矩形为例，可以通过计算鼠标点到矩形四条边的距离，如果该距离小于识别精度，则代表被选中。其流程如图 2-6 所示。

以上方法可以判断某一具体的图元是否被选取，而在实际选取过程中，视图中可能存在很多图元对象，应该依次判断这些图元对象是否被选取。因此，基于文档/视图结构的程序设计中，在文档中定义 CDrawMetaList 链表对象 m_objects，存放所有图元对象的指针，它可以存储指向 CObject 类的派生类的对象指针，可以有效地对图元进行存储和选取，移动，删除等操作。在文档中定义

CDrawMetaList 链表对象 m_selection 选择集，存放被选中的图元对象的指针。当选取图形时，对图元链表中的每一个元素进行遍历来判断图形元素链表中是否有图元被选中。如果被选中，把该图元加入选择集中 m_selection 中，以便以后对图元进行操作。

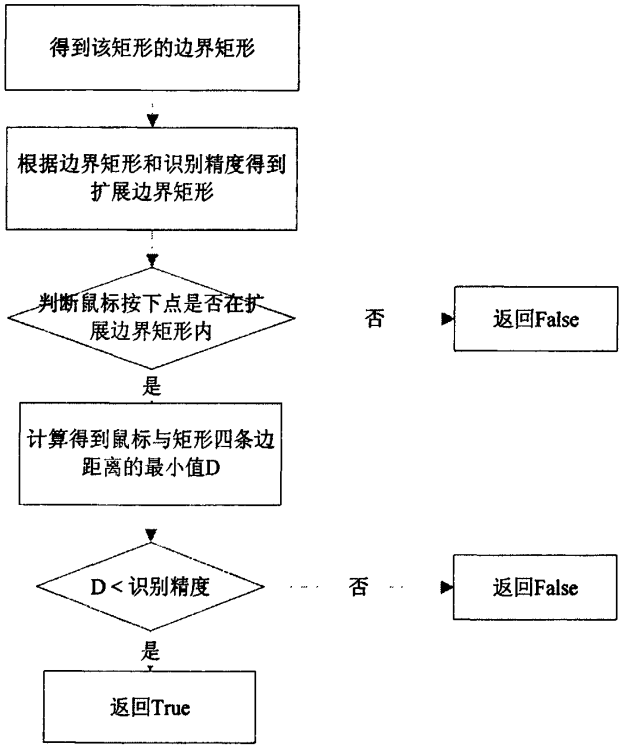

图 2-6 判断矩形是否是被点选的程序流程图

（2）区域选择图形

区域选择方法是，按下鼠标左键，拖动鼠标画出一个矩形区域，判断这个区域的矩形范围是否与图形对象的边界矩形相交，如果相交，则表示该图形对象被选中。该种方法可以选择多个图形对象，实现过程的程序流程图如图 4-8 所示。

2.4.8 绘图界面闪烁处理

工程组态过程中，通常需要绘制很多图形，并对其进行移动，缩放等操作，在这种情况下，尤其是载入较大的位图作为背景时，会产生严重的闪烁现象，给用户带来很大的不便，这是在绘图系统中必须解决的问题[30]。

产生屏幕闪烁的原因是：绘图过程大多放在 OnDraw 或者 OnPaint 函数中，

OnDraw 在进行屏幕显示时是由 OnPaint 进行调用的。当窗口由于任何原因需要重绘时，总是先用背景色将显示区清除，然后与屏幕绘图区域一致的对象，将图形绘制到内存中的这个对象。才调用 OnPaint，而背景色往往与绘图内容反差很大，这样在短时间内背景色与显示图形的交替出现，使得显示窗口看起来在闪[31]。

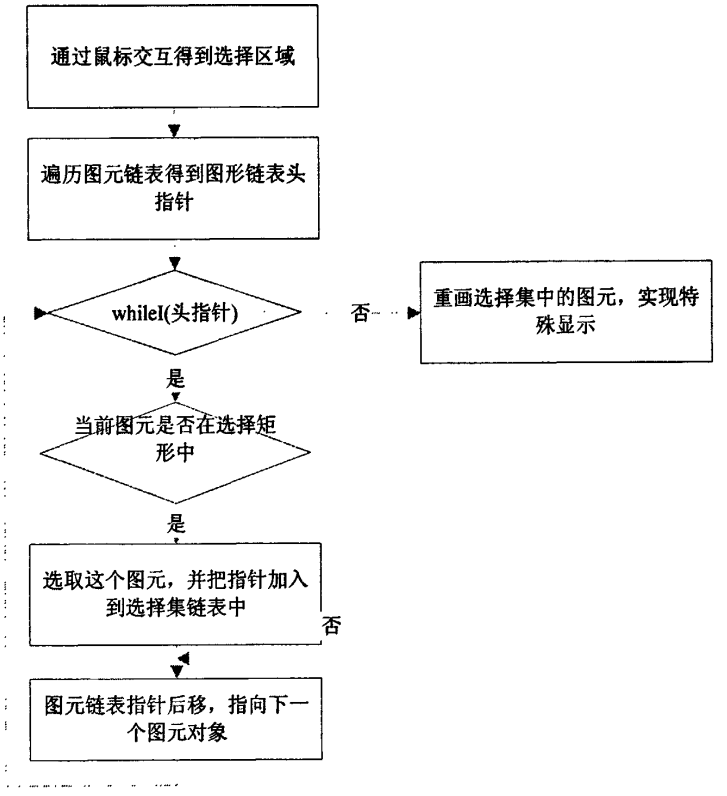

图 2-7 区域选择法的程序流程图

在本系统中，采用双缓冲机制来绘制图形。利用双缓冲机制绘图就是在屏幕绘图区域中显示想要的图画之前，先在内存中创建一个上，再一次性将这个对象上的图形拷贝到屏幕上，这样能大大加快绘图的速度。双缓冲绘图实现的过程如下：在内存中创建与屏幕绘图区域一致的缓冲区->在缓冲区绘图->将在缓冲区绘制的图形拷贝到当前屏幕绘图区域上->释放内存缓冲区。下面是双缓冲技术在程序中实现的部分代码：

```
void CDrawView:: OnDraw(CDC*pDC)
{
```

首先定义一个显示设备对象；

定义一个位图对象；

随后建立与屏幕显示兼容的内存显示设备；

下面建立一个与屏幕显示兼容的位图；

将位图选入到内存显示设备中；

先用背景色将位图清除干净；

用 MemDC 绘图；

将内存中的图拷贝到屏幕上进行显示；

绘图完成后的清理；

}

2.5 动画连接模块的实现

2.5.1 动画连接概述

集散控制系统上位机组态软件要求图形界面友好，能够模拟系统的实际运行情况，即实现动画。工艺流程图的制作目的是使操作员通过它监视现场的情况，以便对紧急情况做出处理。一幅完整的工艺流程图中，有些图形是起修饰作用的，而有些图形则反映现场的某些检测量的变化或现场某些控制装置（开关、阀门等等）的状态。这些图形元素在组态软件中称为热点元素。热点元素是随现场情况的变化而变化的，这就需要将这些热点元素和控制现场的检测量或控制装置联系起来，从而使热点元素能反映检测量的变化。由于在变量登记时，已将现场的检测量的变量名称存储于上位机的数据库中，下位机采集的数据会按一定的时间间隔送到上位机数据库的对应变量中。因此需要将热点图与上位机数据库中某一变量对应起来即可。上位机程序会按一定的时间间隔到上位机数据库中去取数据并通过热点图形元素的动画效果反映给操作员。

为了在组态软件运行时实现动画效果，需要在图形组态时定义动画连接，即在需要实现动画的图形对象与系统变量之间建立一个联系，让工作站运行软件了解某一个图形对象应该根据系统哪一个变量的数值动作。动画连接组态模块完成图形的动画属性，与实时数据库中定义的变量建立相关性的连接关系，作为动画图形的驱动源。动画属性与设备的 I/O 变量等相关，它反映图形大小、颜色、位置、可见度、闪烁性等状态的特征参数，随着表达式的值的变化而变化。动画连

接建立画面中对象与数据变量或表达式的对应关系。建立了动画连接以后，在图形界面运行环境下，根据数据变量或表达式的变化，图形对象就可以按动画连接的要求进行改变。

当用户需要为某一图形对象建立动画连接时，只需让鼠标指向该对象，点击鼠标右键选择“动画连接”，就会弹出一个动画连接对话框，在此对话框内，输入与此图形对象相联系的一个变量名称以及与动画绘制和显示有关的信息，如填充方向，线及颜色属性等内容，单击确定即可。利用动画连接对话框，系统将生成一系列组态信息，并将他们保存在数据文件中，供工作站运行软件调用。

动画连接组态模块完成图形的动画属性，与实时数据库中定义的变量建立相关性的连接关系，作为动画图形的驱动源。动态属性与设备的 I/O 变量等相关，它反映图形大小、颜色、位置、可见度、闪烁性等状态的特征参数，随着表达式的值的变化而变化。

创建动画制作连接的基本步骤如下：

1、创建或选择连接对象(线、填充图形、文本等)。

2、右键单击图形对象，弹出控件属性控制对话框，选中控件连接属性页。

3、选择对象想要进行的连接。

4、为连接定义输入详细资料。

2.5.1 动画连接组态的实现

动画连接包括颜色连接、填充连接、位置与大小变化连接、图形连接。

1、颜色连接

具有该功能的图元是直线、矩形、圆角矩形、箭头、椭圆。此动画属性由图不难看出只能关联具有离散型的变量。当变量为 1，也即为 ON 状态时，选中的图元在运行过程中就显示选中的颜色。

2、填充连接

具有此功能的图元为矩形、圆角矩形、椭圆。当图元的填充方向变量 process3=1、2、3、4 分别表示向上，向下、向左、向右填充。其实现过程为：在系统运行过程中，通过设定图元中的另一矩形边界 cx1，cy1，cx2，cy2。把需要填充的矩形大小赋予给这个矩形边界。然后在 OnPaint 函数中对该图元进行绘制。

3、位置与大小变化连接

位置和大小变化连接包括三种连接：分别为水平移动连接、垂直移动连接和

缩放连接，规定了图形对象如何随变量的变化而改变位置或大小。下面对这三部分进行详细介绍。(1)缩放连接缩：缩放连接是使被连接对象的大小随连接表达式的值而变化。（2）水平移动连接：水平移动连接是使被连接对象在画面中随表达式的植以象素为单位，以被连接对象在画面制作系统中的原始位置为参考基准的。水平移动通常用来表示图形对象实际水平移动。（3）垂直移动：垂直移动连接是使被连接对象在画面中的位置随连接表达式的值而垂直变动。移动的距离以象素为单位，以被连接对象在画面制作系统中的原始位置为参考基准。垂直移动通常用来表示对象实际的垂直运动。

2.6 工程浏览器模块

该模块实现与组态王中浏览器功能相似的管理工程的功能，当打开程序时，在客户区的左边显示该窗口，在窗口的内部用树形控件编辑选项，在树形控件形成的过程中，配以图像列表控件，用于使工程的界面的选项表达的更加清晰和美观。

1．工程浏览器实现原理

浏览器窗口的实现类是从 CControlBar 派生出来的，在此并不需要从头到尾实现该类，因为 Cristi Posea 已经为我们实现了一个称为 CSizingControlBar 的类，该类实现了能够拖动、更改大小的控制栏窗口，而且做得相当完美！所要做的便是好好地利用该类，通过继承该类在多文档窗口中创建控制栏。

2、工程浏览器的实现

(1)首先，在 VC 中新建一个多文档工程，将 sizecbar.cpp 和 sizecbar.h 复制到工程的文件夹中，并将新文件加入到工程中。这样工程中加入了个新类 CSizingControlBar。然后，建立自己的新类 CMyBar 继承 CSizingControlBar；

(2)在 CMyBar 类中加入 OnUpdateCmdUI (CFrameWnd*pTarget, BOOL bDisableIfNoHndler) 其内部代码为 UpdateDialogControls (pTarget, bDisableIfNoHndler)；

(3)生成继承 CTreeCtrl 类的 cmytreectrl 类，该类主要的 OnMouseMove 函数用来得到捕捉鼠标在树形控件的位置；

(4)在 CMyBar 类中把 cmytreectrl m_tree 声明为公有成员。并在 CMyBar 类的 OnSize 函数中实现放置树形控件在控制栏中的位置；

(5)声明全局变量 CMyBarba，然后在 CMainFrame 类中的 OnCreate 函数创建

该对象，使之具体化。设置其显示的初始大小、名称、停靠的位置和默认水平垂直的窗口大小。并在 OnCreate 函数中调用全局函数 SetImage 进行图标数组的设置，然后调用 SetTree 函数进行树形控件的初始化。如此完成了带有图标的树形浏览器窗口的设置。浏览器的窗口可以随意拖放、更改大小。

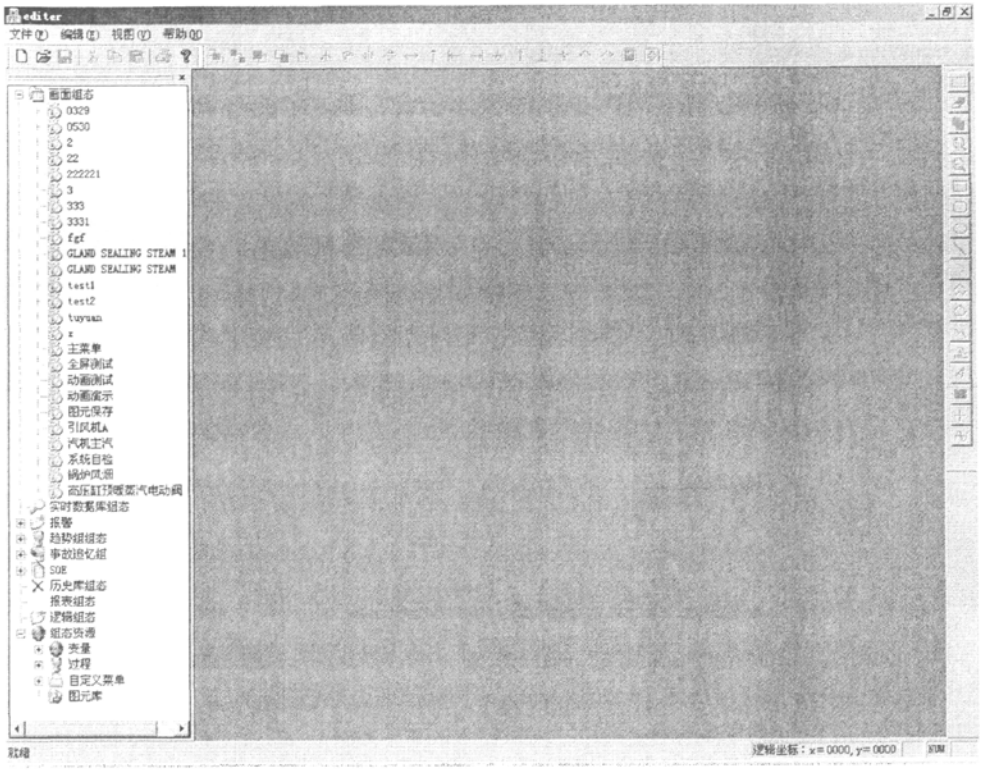

图 2-8 工程浏览器

2.7 本章小结

本章主要详细介绍了组态软件各模块中最复杂的一个模块——图形组态系统模块。并描述了该模块的若干子模块，如图形生成子模块、组态数据的工程管理模块的实现原理和方法。该模块最终实现了图形接口的功能，遵循所见即所得的规则，使工程设计人员在设计应用程序时简便灵活，不易出错。配有图库，后续设计人员可以参照类似的方法进行扩充、补充和替换。

第三章 数据存储模块设计

数据存储模块，存放组态软件在运行过程中的实时数据，如报警异常、SOE数据、历史数据等。需要采用数据库的形式存放数据。

3.1 数据存储模块整体架构

由于本系统要求每秒钟要处理2万个监控点的数据，即8万个字节的数据，如果采用SQL SERVER等大型关系数据库，由于复杂的数据库事务操作以及低效的高层接口，导致系统的存储，查询效率低下，满足不了项目的要求，并且SQL SERVER运行时占有CPU使用率比较高，影响其他进程有效的执行。而采用内存数据库，由于其对数据的操作都是在内存中执行，每次插入、检索、更新数据的操作都非常快，通常，在400MHz的处理器上一个交易的时间只有1～2微秒，这要比静态数据管理或常规数据库的性能要高很多。为了满足项目的需要并支持以上的功能，本文采用的数据库系统采用Berkeley DB。

本项目采用的内存数据库为Berkeley DB嵌入式数据库，数据库系统以C/C++的Library的形式提供给用户，与用户的程序无缝集成在一个运行程序之中，没有客户端程序和数据库服务器之间昂贵的网络通讯开销，也没有本地主机进程之间的通讯。Berkeley DB对所有操作都使用一组API接口，因此不需要对某种查询语言进行解析，也不用生成执行计划，大大提高了运行效。并且其数据库是以文件的格式保存在磁盘上，在物理组织上，可供选择的四种文件存储结构分别是：哈希文件、B树、定长记录(队列)和变长记录(基于记录号的简单存储方式)。所以历史数据库在这里只是磁盘文件[32]。数据处模块结构如图3-1所示。

(1)实时数据采集模块

实时数据采集模块负责接收下位机DPU发给HMI的工业设备数据。这些工业设备数据是工业现场设备运行的实时值。如锅炉的温度，压力，水位等。实时数据模块将采集的数据放入内存缓冲块中，并且根据报警，历史数据保存，SOE等业务逻辑进行处理。内存缓冲快是用内存映射文件实现的，可多线程的访问。

(2)数据库操作缓冲

数据库操作缓冲是由内存映射文件实现的队列。每一项数据是一个三元组，

既(操作、数据库名、数据)。如要对历史数据库进行写操作(WRITE，HISDATA，*PDATA)。该内存缓冲对数据库的操作，平滑数据库的访问，保证实时数据采集模块的实时行。该数据库操作缓冲被三个线程访问，数据库访问线程、实时数据采集线程、Editor View。数据库访问线程负责提取操作缓冲中的操作，并进行实质的数据库读写操作。Editor、View、实时数据采集线程将对数据库的访问打包成三元组，写到数据库操作缓冲中。

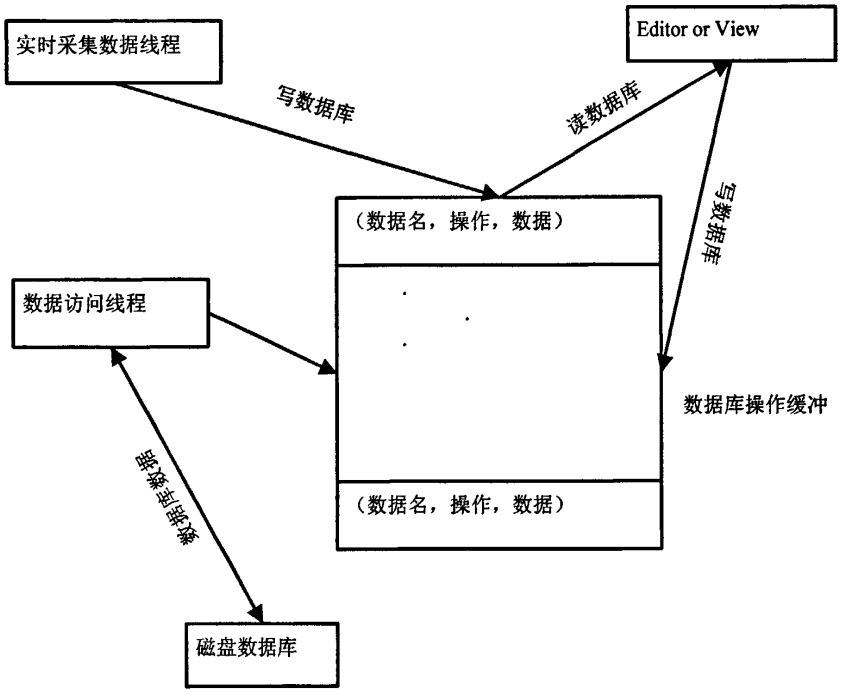

图 3-1 存储模块整体架构

(3)数据访问线程

在对数据库的操作中，任务主要包括读数据库任务和写数据库任务，其中写数据库任务为定时任务，读数据库任务为非定时任务。此模块的功能就是对数据库的读写进行协调，使它们可以同步的访问实时数据库，而不会引起读写操作的冲突，其中写数据库的优先级要比读数据库的优先级高。数据库缓冲访问算法：用来协调读写进(线)程，使它们可以同步的访问数据库，其算法思想及实现见第五章。

(4)磁盘数据库

本系统所用数据库系统 BERKELEY DB 存储的数据库，以文件形式保存数

据。为方便程序的管理，按照不同类型数据分成文件夹保存数据。其中包括下类文件：执行文件、机群配置、用户账号、操作日志、图元库、系统帮助、工程文件、画面组态、历史趋势、实时趋势、SOE 组态：日志文件、SOE 日志数据文件、SOE 日志文件结构、SOE 点表文件、SOE 点表文件结构、报警日志数据文件、报警日志文件结构、报警点数据文件、报警点数据结构文件、元数据文件、元数据文件、元数据文件结构、历史库组态数据文件、历史库模拟量数据文件、历史库开关量数据文件、历史库模拟量数据文件结构、历史库开关量数据文件结构、事故组组态、事故组态文件事故组日志文件、事故组日志文件结构、事故组文件、事故组。

3.2 组态软件中需要存放在数据库中的数据

1．数据采集监控点的配置元数据：

根据实际监控现场，配置每一个监控点的属性，并保存在数据库中，即数据库组态。本项目设置了一些必要的基本属性，组态工程师可以根据实际项目需求添加监控点的属性。监控点的基本属性如下：

IO 标签名：系统唯一的人可识别名字。

序号：便于计算机对数据操作；

全局点号：与硬件配置的点好对应；

参数说明：对该监控点的解释说明；

测点类型：四种类型：模拟输入，模拟输出，数字输入，数字输出；

信号类型：所采集的监控点的单位；

小数位数：采集精度；

量程上限、量程下限：该采集点的最大值和最小值；

是否报警：配置该检测点是否在其值有异常情况时，是否给检测员声光报警；

是否历史库：该 IO 点的值是否进入历史库保存；

报警优先级：配置报警优先级；

报警上上限、报警上限、报警下限、报警下下限、控制站号、板卡号、端口地址。

2．在运行阶段，根据历史库组态的结果，采集数据后，将要进入历史库 IO 点的数据值保存到历史库中。

3．报警数据库：为了支持浏览以前报警的日志，需要将报警信息也保存在数据库中。

4．SOE 数据库：将 DPU 发送上来的数据保存在数据库中。

3.3 本系统对数据库的性能要求

本项目所开发的组态软件是用于火力发电厂实时监控系统的，所要处理的数据量庞大，实时性要求高，对于数据库至少要满足以下功能：

1.具有快速存储功能：要求在数据库中每秒钟要存储 4 万个监控点的，大约 16 万字节的数据。

2.具有实时的处理各种任务，其中写任务为定时事务，每秒钟执行一次，读任务为非定时事务，不定期的执行。

大型关系数据库在存储和管理永久性、非短暂数据方面虽然有着广泛的应用，但是它并不适合管理本系统的数据。大型关系数据库不能用作工业数据库的主要原因是由数据库系统本身的构成和工控软件的特点造成的。以微软的大型关系数据库 SQL SEVER 为例，有以下几个方面原因[33][34][35]：

1. SQL SEVER 关系数据库一般为磁盘数据库，数据主要存储在慢速的外部存储设备中，执行时间不可预测。当进行数据库的读写时，涉及到磁盘 IO 与内外存的交换，特别是象我们这样频繁的对数据库进行大数据量的读写，磁盘 IO 调度就会变的非常频繁，效率就会降低，其实时性相对较差。

2. SQL SEVER 关系数据库屏蔽了底层系统，呈现给用户的是它的高层接口。比如用户使用的操作语言是 SQL 语句，而 SQL 语句要经过编译、优化等步骤，转化为底层语言，然后再执行，这样在效率上就低很多。而本系统所用数据库系统就直接调用相关的 API 函数，不需要对 SQL 等数据库语句进行解析，大大提高了运行效率。再者，关系数据库系统还要维护事务，因表与表之间的数据相互依赖，执行效率再打折扣。

3.SQL SERVER 等关系数据库分为客户端和服务器端两部分，客户端与服务器端通信要消耗时间。客户端程序负责商业逻辑和向用户提供数据，服务器端程序负责对数据库的数据进行操作和管理。客户端与服务器端一般运行在不同的机器上，有专门的机器做服务器。当要进行数据库的相关操作时，就需要与服务器端进行通信，如果服务器端与客户端在不同的机器上，则需要进行网络通信，数据量大的话，就会消耗相当的时间。就算客户程序运行在服务器端，它与数据库服务器之间的通讯属于进程间的通信，也需要消耗一定的时间。

4.多表的操作消耗大量的时间。项目所需要的数据库是应用在工业控制中的，

主要是以表的结构来存储实时数据。在表中，每一条记录都是以时间为顺序插入表中的。在 SQL SEVER 数据库中表的记录数字段数是有限制的，最多一个表有 1024 个字段。在一般的应用中，这个字段数是足够的，但是在工业控制中是不够的。在工控软件中，所设计的表的字段数是由现场监控点 IO 点的个数来决定的，一条记录是由多个 IO 点组成的，每一个 IO 点都是记录中的一个字段。一般的现场监控制系统中，监控点都有上万个，也就是说一条记录都是有上万个字段。特别是本问所描述的项目中，可能需要处理多达 2 万个 IO 点的数据，也就是把 2 万个点的数据作为一条记录插入数据库中。这样的话，SQL SERVER 中一个表是装不了一条记录的。可以用多个表来实现，把一条记录分成多段，插入多个表中，由多个表来共同管理一条记录。在理论上是可以这样存储的，但在现实中对于记录的操作将会相当烦琐，操作一条记录时需要 20 个表来共同合作完成，耗费大量的时间。如要插入一条数据时，需要在程序中把记录分好段，并记录好这个记录段对应于哪个分表，把不同的记录段插入对应的分表中，每插入一次都还要调用 INSERT 语句一次。如果项目采用 SQL SEVER 的话，则需要 20 个分表来协作完成，插入一条记录则需要向 20 个分表中分别插入，需要执行 20 次 SQL 语句。这样做实时性就变的很差，特别是我们要求每秒钟都需要向表中插入记录，利用 SQL SERVER 作为实时数据库是满足不了项目的实时性要求的。做了以下测试，用来进一步验证 SQL SERVER 数据库满足不了本系统对数据库的性能要求。根据项目要求，每秒钟都要向实时数据库中插入 2 万个 IO 点数据，在 SQL SERVER 中也就是每秒钟要插入一条记录，记录的长度为 2 万个字段，大小为 8 万个字节，由以上可知，需要 20 个分表来合作完成。下面的试验 SQLSERVER 能否满足上述要求。

测试环境：

软件平台：Windowsxp sp2 VC SQL Server2000

硬件平台：奔四 1.5GHz CPU，512M 内存，80G 硬盘

测试结果如图 3-2 和图 3-3。

从图 3-2 和图 3-3 中可以看出不同程度的出现了数据丢失的现象，在第一个表中，数据从 19 秒直接跳到了 21 秒，第 20 秒的数据被丢失。并且记录的排序也比较混乱，不按时间顺序来排，如测试结果 2 所示。所以，SQL SERVER 数据库满足不了项目中对于数据的访问要求，需要一个小巧快速的数据库。

时	分	秒	G6_20000	E99_20001	C30_20002	M83_20003	Z79_20004	U86_20005	I
14	33	13	198	26920	31009	30734	4137	25825	:
14	33	15	198	26920	31009	30734	4137	25825	:
14	33	17	198	26920	31009	30734	4137	25825	:
14	33	19	198	26920	31009	30734	4137	25825	:
14	33	21	3875	7062	31559	3908	13199	31132	2
14	33	24	3875	7062	31559	3908	13199	31132	2
14	33	28	3875	7062	31559	3908	13199	31132	2
14	33	33	12712	14028	21336	29689	182	23123	:
14	33	39	12712	14028	21336	29689	182	23123	:
14	33	42	9053	10673	21660	19542	9510	7790	'
14	33	48	9053	10673	21660	19542	9510	7790	'
14	33	50	21323	18221	9819	17026	17000	341	5
14	33	54	21323	18221	9819	17026	17000	341	5
14	33	57	21323	18221	9819	17026	17000	341	5
14	34	0	25720	17960	17373	1190	28163	25200	6
14	34	4	25720	17960	17373	1190	28163	25200	6
14	34	7	25720	17960	17373	1190	28163	25200	6
14	34	10	25720	17960	17373	1190	28163	25200	6
14	34	15	11759	16778	31849	17641	12671	17704	:
14	34	19	11759	16778	31849	17641	12671	17704	:
14	34	22	15029	14395	29512	14708	2661	20707	6

图 3-2 微软 SQL 测试结果

分	秒	G6_20000	E99_20001	C30_20002
20	24	8224	23867	11904
20	25	6616	19096	2268
20	26	27864	20828	10050
20	27	9257	28783	11515
20	28	32466	2745	4431
20	29	28635	26403	8070
20	13	20123	18782	3935
20	15	31288	3212	9670
20	16	15565	28690	11113
20	17	6498	2774	31630

图 3-3 微软 SQL 测试结果

3.4 Berkeley DB 数据库介绍

Berkeley DB 是由美国 Sleepycat Software 公司开发的一套开放源码的嵌入式数据库的程序库(database library)，它为应用程序提供可伸缩的、高性能的、有事务保护功能的数据管理服务。Berkeley DB 为数据的存取和管理提供了一组简洁的函数调用 API 接口[36][37]。

它是一个经典的 C-library 模式的 toolkit，是为应用程序开发者提供工业级强度的数据库服务而设计的。其主要特点如下：

嵌入式(Embedded)：它直接链接到应用程序中，与应用程序运行于同样的地址空间中，因此，无论是在网络上不同计算机之间还是在同一台计算机的不同进

程之间，数据库操作并不要求进程间通讯。

Berkeley DB 为多种编程语言提供了 API 接口，其中包括 C、C++、Java、Perl、Tcl、Python 和 PHP，所有的数据库操作都在程序库内部发生。多个进程，或者同一进程的多个线程可同时使用数据库，有如各自单独使用，底层的服务如加锁、事务日志、共享缓冲区管理、内存管理等等都由程序库透明地执行。

轻便灵活(Portable)：它可以运行于几乎所有的 UNIX 和 Linux 系统及其变种系统、Windows 操作系统以及多种嵌入式实时操作系统之下。它在 32 位和 64 位系统上均可运行，已经被好多高端的因特网服务器、台式机、掌上电脑、机顶盒、网络交换机以及其他一些应用领域所采用。一旦 Berkeley DB 被链接到应用程序中，终端用户一般根本感觉不到有一个数据库系统存在。

可伸缩(Scalable)：这一点表现在很多方面。Database library 本身是很精简的(少于 300KB 的文本空间)，但它能够管理规模高达 256TB 的数据库。Berkeley DB 能以足够小的空间占用量运行于有严格约束的嵌入式系统，也可以在高端服务器上耗用若干 GB 的内存和若干 TB 的磁盘空间。

与其他大多数数据库系统相比，BDB 提供了相对简单的数据访问服务。BDB 只支持对记录所做的几种逻辑操作。它们是：

①在表中插入一条记录。

②从表中删除一条记录。

③通过查询键(key)从表中查找一条记录。

④更新表中已有的一条记录。

BDB 不是一个独立的数据库服务器。它是一个函数库，和调用它的应用程序是运行在同一地址空间中的。可以把 BDB 作为数据库管理系统来构建服务器程序。比如，有许多商业的和开源的轻量级目录访问协议(LDAP)服务器都使用 BDB 存储记录。LDAP 客户端通过网络连接到服务器。服务器调用 BDB 的 API 来查找记录并返回给客户。而在它本身而言，BDB 却不是数据库的服务器端。所以，BDB 是一种完全不同于其它数据库管理系统的数据库，而且它也不是一个数据库服务器端。Berkeley DB 在实时应用中比关系数据库和面向对象数据库要好，有以下两点原因[38]：

(1)BDB 数据库系统以 C/C++的 Library 的形式提供给用户，与用户的程序无缝集成在一个运行程序之中，没有客户端程序和数据库服务器之间昂贵的网络通讯开销，也没有本地主机进程之间的通讯。在一台机器的不同进程间或在网络中不同机器间进行进程通讯所花费的开销，要远远大于函数调用的开销。

(2)因为 Berkeley DB 对所有操作都使用一组 API 接口，因此不需要对某种查询语言进行解析，也不用生成执行计划，大大提高了运行效。Berkeley DB 所管理数据的逻辑组织单位是若干个独立或有一定关系的数据库(database)，每个数据库由若干记录组成，这些记录全都被表示成(key，value)的形式。Berkeley DB 不对记录里的数据进行任何包装，记录和它的键都可以达到 4G 字节的长度。如果把一组相关的 key/data 对也看作一个表的话，那么每一个数据库只允许存放一个 table，这一点不同于一般的关系数据库。实际上，在 Berkeley DB 中所提到的“数据库”，相当于一般关系数据库系统中的表；而“key/data”对相当于关系数据库系统中的行(rows)；Berkeley DB 不提供关系数据库中列直接访问的功能，而是在“key/data”对中的 data 项中通过实际应用来封装字段(列)。在数据库领域中，数据访问算法对应了数据在硬盘上的存储格式和操作方法。在编写应用程序时，选择合适的算法可能会在运算速度上提高 1 个甚至多个数量级。大多数数据库都选用 B+树算法，BDB 也不例外，同时还支持 HASH 算法、Recno 算法和 Queue 算法。在物理组织上，每一个数据库在创建的时候可以由应用程序根据其数据特点来选择一种合适的存储结构。可供选择的四种文件存储结构分别是：哈希文件、B 树、定长记录(队列)和变长记录(基于记录号的简单存储方式)。一个物理的文件中可以只存放一个单独的数据库，也可以存放若干相关或不相关的数据库，而且这些数据库可以分别采用除队列之外任意不同的组织方式，以队列组织的数据库只能单独存放于一个文件，不能同其他存储类型混合存放。一个文件除了受最大文件长度和存储空间的约束之外，理论上可以存储任意多个数据库。因此系统定位一个数据库通常需要两个参数－“文件名”和“数据库名”，这也是 BerkeleyDB 不同于一般关系数据库的地方。

Berkeley DB 的核心数据结构[39]:

数据库句柄结构 DB：包含了若干描述数据库属性的参数，如数据库访问方法类型、逻辑页面大小、数据库名称等；同时，DB 结构中包含了大量的数据库处理函数指针，大多数形式为(*dosomething)(DB*,arg1,arg2,…)。

数据库记录结构 DBT：BDB 中的记录由关键字和数据构成，关键字和数据都用结构 DBT 表示。实际上完全可以把关键字看成特殊的数据。结构中最重要的两个字段是 void*data 和 u_int32_t size，分别对应数据本身和数据的长度。

数据库游标结构 DBC：游标(cursor)是数据库应用中常见概念，其本质上就是一个关于特定记录的遍历器。BDB 支持多重记录(duplicate records)，即多条记录有相同关键字，在对多重记录的处理中，使用游标是最容易的方式。

数据库环境句柄结构 DB_ENV：环境在 DB 中属于高级特性，本质上看，环境是多个数据库的包装器。当一个或多个数据库在环境中打开后，环境可以为这些数据库提供多种子系统服务，例如多线/进程处理支持、事务处理支持、高性能支持、日志恢复支持等。

DB 中核心数据结构在使用前都要初始化，随后可以调用结构中的函数（指针）完成各种操作，最后必须关闭数据结构。从设计思想的层面上看，这种设计方法是利用面向过程语言实现面对对象编程的一个典范[40]。

3.5 本章小结

以上给出了数据存储模块的整体设计以及为什么本项目选择 BDB 作为永久数据存储的数据库。下面两章将对数据采集模块和数据库访问模块数据库缓冲访问算法进行着重讨论，并给出重要代码。

第四章 数据采集模块设计

新型监控软件都支持基于 TCP/IP 协议的网络化运行环境，采用 C/S 模式进行网络间的通信。由于现场设备的分布性以及对数据采集和管理的远程化操作的要求，监控组态软件不仅要和现场设备通信，而且经常要用多台计算机构成分布式监控网络。随着网络技术的发展和企业管理的信息化，监控网络与企业信息管理系统融合以成为一大发展趋势。因此，合理设计监控组态软件的网络体系结构对系统运行性能的充分发挥和对工业过程对象的有效管理具有重要意义。在对监控系统通行网络的设计中满足以下几个要求：

1）良好的可靠性。单个点发生故障时，其他站点可以继续工作。

2）数据传输的快速。在实时系统中，快速的网络对提高系统实时性意义重大。

4.1 数据采集模块整体架构

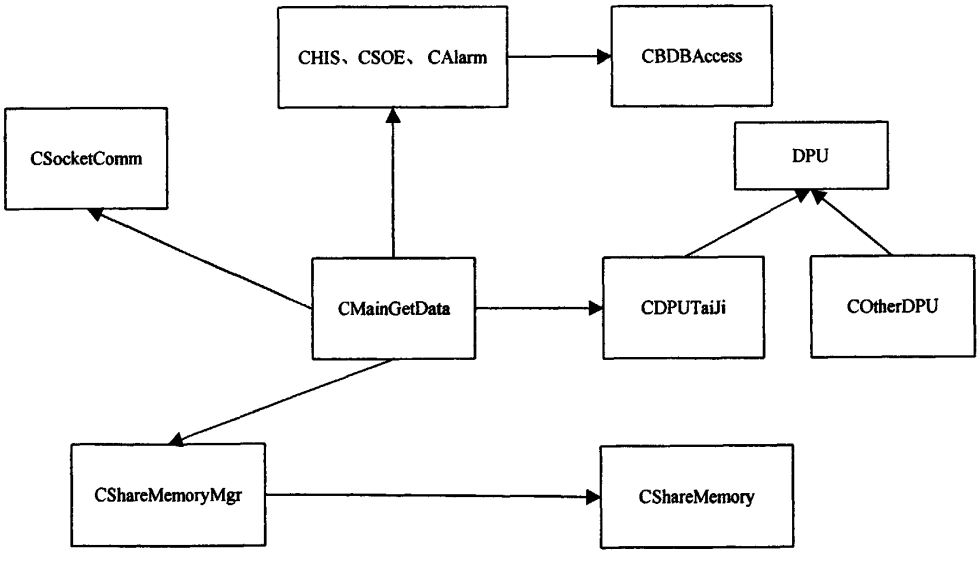

图 4-1 存储模块类结构图

数据采集模块只在运行状态时运行。它采集工业实时数据，放入内存以供 View 模块访问，如实时趋势。数据采集模块还对采集来的数据按照历史库、SOE、

报警等业务逻辑进行处理。其整体结构如图 4-1，其中各模块功能如下：

CSocketComm：所有关于网络的操作都在其中。

CHIS、CSOE、CAlarm：报警、历史库、SOE 业务逻辑。

CBDBAccess：处理访问数据库的操作，与数据库缓冲访问模块协同工作。

CShareMemoryMgr、CShareMemory：共同管理内存映射文件的操作。

DPU、CDPUThisProject、COtherDPU：根据不同的协议实现不同的代码。

4.2 Windows 网络编程介绍

在网络通信中使用最多的就是 Winsock API。Winsock 是 TCP/IP 编程最低级的 WindowsAPI。其代码的一部分位于 wsock32.dll，另一部分位于 Windows 核心。对于众多的基层网络协议，Winsock 是访问它们的首选接口。而且在每个 win32 平台上 Winsock 都以不同的形式存在着。Winsock 是网络编程接口、而不是协议。它从 Unix 平台的 Berkeley(BSD)套接字借鉴了许多东西、后者能访问多种网络协议。在 win32 环境中，Winsock 接口最终成为一个真正的“与协议无关”接口。它能无缝的在多台主机的进程之间进行通信，并提供了完善的通信接口。因此，在本软件系统网络通信模块中使用 Winsock 完成[41][42]。

在 TCP/IP 网络中两个进程间的相互作用的主机模式是客户机/服务器模式(Client/Server model)。该模式的建立基于以下两点：一是非对等作用；二是通信完全是异步的。客户机/服务器模式在操作过程中采取的是主动请示方式[43]。

1.服务器方要先启动，并根据请示提供相应服务，过程如下:

(1)打开通信通道并告知本地主机，它愿意在某一个公认地址上接收客户请求。

(2)等待客户请求到达该端口。

(3)接收到重复服务请求，处理该请求并发送应答信号。

(4)返回第二步，等待另一客户请求。

(5)关闭服务器。

2.客户方:

(1)打开通信通道，并连接到服务器所在主机的特定端口。

(2)向服务器发送服务请求报文，等待并接收应答；继续提出请求。

(3)请求结束后关闭通信通道并终止。

3.套接字的使用步骤:

(1)启动 Winsock:对 Winsock DLL 进行初始化，协商 Winsock 的版本支持并分配必要的资源(服务器端和客户端):

int WSAStartup(WORD wVersionRequested,LPWSADATA lpWSAData)

(2)创建套接字:(服务器端和客户端)

SOCKET socket(int af,int type,int protocol)

(3)套接字的绑定:将本地地址绑定到所创建的套接字上。(服务器端和客户端)

int bind(SOCKET s,const struct sockaddr_in*name,int namelen)

(4)套接字的监听:(服务器端)

int listen(SOCKET s,int backlog)

(5)套接字等待连接:(服务器端)

SOCKET accept(SOCKET s,struct sockaddr_in*addr,int*addrlen)

(6)套接字的连结:将两个套接字连结起来准备通信。(客户端)

int connect(SOCKET s,const struct sockaddr_in*name,int namelen)

(7)套接字发送数据:(服务器端和客户端)

int send(SOCKET s,const char*buf,int len,int flags)

(8)套接字的数据接收:(客户端)

int recv(SOCKET s,char*buf,int len,int flags)

(9)中断套接字连接:通知服务器端或客户端停止接收和发送数据。(服务器端和客户端)

int shutdown(SOCKET s,int how)

(10)关闭套接字:释放所占有的资源。(服务器端和客户端)

int closesocket(SOCKET s)

首先在服务端创建 socket，并通过函数 WSAAsyncSelect 选择异步通讯方式，开始在指定端口监听，函数将对在它参数中定义的消息 FD_ACCEPT 和 FD_CLOSE 做出响应。然后监听到 FD_ACCEPT 或 FD_CLOSE，系统就会通过定义的消息通知相应的消息处理函数进行处理。系统将分别响应连接和关闭消息。接着在消息处理函数中，判断消息类型，如果为连接请求消息，并没有超出服务器端最大连接数，则服务端会创建一个 socket 和这个客户 socket 建立通信，原来的 socket 仍然保持监听状态，同时将和客户端连接的 socket 设置为异步通讯方式并监听客户端的读和关闭消息。如果监听到 FD_READ 读消息说明客户端数据已经到达操作系统缓冲区，准备接收。则可调用 recv 函数接收数据。在程序主框架中，可以声明一个 socket 数组用来保存服务端所有和客户端建立连接的 socket。

如果为关闭消息则关闭服务端 socket。整个工作过程中服务端需要定时监测客户端连接状况，及时删除无效的不可用的 socket 连接。这是通过定时器来实现的。OnTimer 可以很好的达到我们的希望。这里问题的关键是如何检测 socket 连接的状态。Winsock 提供了很好的 I/0 方法可以对套接字上的 I/0 行为加以控制。Winsock 提供了一些有趣的 I/O 模型，有助于应用程序通过一种“异步”方式一次对一个或多个套接字上的通信加以管理。这些模型包括 select(选择)、WSAAsyncSelect(异步选择)、WSAEventSelect(事件选择)、Overlapped I/O(重叠式 I/O)以及 Completion Port(完成端口)等等。这里仅简要介绍一下 select 和 WSAAsyncSelect 模型，因为组态软件中主要用到的也就是这两种模型：

1.select 模型

select 模型是 winsock 中最常见的一种 I/0 模型。之所以称其为 select 模型。是由于它的中心思想便是利用 select 函数。它使那些想避免在套接字调用过程中被无辜锁定的应用程序，采取一种有序的方式，同时进行对多个套接字的管理。利用 select 函数，判断套接字上是否存在数据，或者能否向另一个套接字写入数据。之所以要设计这个函数，唯一的目的就是防止应用程序在套接字处于锁定模式中时，在一次 I/0 绑定调用过程中，被迫进入锁定状态，同时防止在套接字处于非锁定模式中时，产生 WSABWOULDBLOCK 错误。

Select 函数原形如下：

```
int select(int nfds, fd_set*readfds, fd_set*writefds, fd_set*exceptfds,
const struct timeval*timeout);
```

其中三个 fd_set 参数，一个用于检查可读性(readfds)，一个用于检查可写性(writefds)，另一个用于例外数据(exceptfds)。从根本上说 fd_set 数据类型代表着一系列特定套接字的集合。其中:(1) readfds 集合包括符合下述任何一条件的套接字:a.有数据可以读入。b.连接已经关闭，重设或中止。c.加入已调用了 listen，而且一个连接正在建立，那么 accept 函数调用会成功。(2)writefds 集合包括符合下述任何一个条件的套接字:a.有数据可以发出。b.如果已完成了对一个非锁定连接调用的处理，连接就会成功。(3)exceptfds 集合包括符合下述任何一个条件的套接字:a.假如已完成了一个非锁定连接调用的处理，连接尝试就会失败。b.有带外数据可供读取。

例如，假定想测试一个套接字是否可读，必须将自己的套接字增添到 readfds 集合，再等待 select 函数完成。Select 完成之后，必须判断该套接字是否仍然为 readfds 集合的一部分，若答案是肯定的，便表明该套接字可读，可立即着手从它

上面读取数据。在三个参数中(readfds,writefds 和 exceptfds)，任何两个都可以是空值(NULL)。但是，至少有一个不能为空。任何不为空的集合中，必须包含至少一个套接字句柄；否则 select 函数便没有任何东西可以等待。最后一个参数 timeout 对应的是一个指针，它指向一个 timeval 结构，用于决定 select 最多等待 I/O 操作完成多久的时间。如 timeout 是一个空指针，那么 select 调用会无限期的锁定或停顿下去，直到至少有一个描述符符合指定的条件后结束。用 select 套接字进行监视之前，在自己的应用程序中，必须将套接字句柄分配给一个集合。设置好一个或全部读、写以及例外 fd_set 结构。将一个套接字分配给一个集合后，再来调用 select。便可知道一个套接字上是否正在发生上述的 I/0 活动。Winsock 提供了下列宏操作，可用来针对 I/0 活动，对 fd_set 进行处理与检查:

a.FD_CLR(s,*set):从 set 中删除套接字 s；

b.FD_ISSET(s,*set):检查 s 是否 set 集合的一名成员，如答案是肯定的，则返回 TRUE；

c.FD_SET(s,*set):将套接字 s 加入集合 set；

d.FD_ZERO(*set):将 set 初始化成空集合。

例如，假如想知道何时可从一个套接字中安全的读取数据，同时不会陷入无休止的锁定状态，便可以使用 FD_SET 宏。将自己的套接字分配给 fd_set 集合，再来调用 select。要想检测自己的套接字是否仍属于 fd_set 集合的一部分，可使用 FD_ISSET 宏。采用下述步骤，便可完成用 select 操作一个或多个套接字句柄的全过程:

a.FD_ZERO 宏，初始化自己感兴趣的每一个 fd_set；

b.使用 FD_SET 宏，将套接字句柄分配给自己感兴趣的每个 fd_set；

c.调用 select 函数，然后等待在指定的 fd_set 集合中，I/0 活动设置好一个或多个套接字句柄。Select 完成后，会返回在所有 fd_set 集合中设置的套接字句柄总数，并对每个集合进行相应的更新；

d.根据 select 的返回值,应用程序便可判断出哪些套接字存在着尚未完成的 I/0 操作。具体的方法是使用 FD_ISSET 宏，对每个 fd_set 集合进行检查；

e.知道了每个集合中待决的 I/0 操作之后，对 I/0 进行处理，然后返回步骤 a,继续进行 select 处理。select 返回后，它会修改每个 fd_set 结构，删除那些不存在待决 I/0 操作的套接字句柄。这正是在上述步骤 d 中，为何要使用 FD_ISSET 宏来判断一个特定的套接字是否仍在集合中的原因。

4.3 数据采集模块实现

在数据采集模块中，人机接口站负责接受下位机发送来的时实数据。下位机是一些工业计算机，处理能力差，多为嵌入式工业计算机。下位机通常进行与实际工业设备相连接，采集工业数据后，经网络送数据到人机接口站。该组态软件就运行在人机接口站上。在该项目中，数据采集模块是服务器，等待接受数据。下位机经过以太网，采用 UDP 协议发送数据。数据采集模块算法如下：

打开通信通道并告知本地主机，它愿意在某一个公认地址上接收客户请求；

等待客户数据到达该端口；

接收到客户数据，根据通信协议处理数据；

返回第二步，等待客户数据到达；

关闭服务器；

4.4 采集模块与分散控制站通信数据

由于监控组态软件的信息复杂，在同一时刻，网络中的信息多种多样，不同种类的信息对应着监控软件不同的功能，如报警信息，数据请求信息，过程数据，事故追忆信息等。为了实现通信信息的规格化，提高系统的代码冲重用程度，有必要设计统一的通信信息格式，这样不仅有利于数据管理，也为后续的扩展保留余地。

本文所设计的通信信息包格式采用同意的结构对数据信息进行包装，通过对信息包所具备的不同标示信息 ID 来给予区分。这种信息的结构可以由用户完成对不同信息包的定义，信息包的长度不是确定的，用户可以根据发送信息的长度进行配置，在信息包可以获得数据长度的信息。

工业设备的实时数据：工业设备实时数据分两种：一种是模拟量的值，在计算机中用浮点型表示，如锅炉的温度；另一种是开关型数据，如阀门的开闭状态，只有两个状态，一个布尔型足够表示了。

DPU 在发送数据时，以封包的形式发送，一个封包包含 2000 个开关量或者包含 1000 个模拟量；这是由于受 TCP/IP 最大传输单元的限制；

4.5 事故追忆、Sequence Of Event、报警处理

在采集了数据后，要对数据进行处理：一些数据要根据业务逻辑的需要写入历史数据库；一些数据偏离正常数据值范围要报警；具体处理如下：

4.5.1 事故追忆

当现场有事故发生时，为了分析事故发生的原因，必须把和事故有关的各个参数在事故前后一段时间内的数据提取出来进行综合分析判断，这就要求跟踪和事故有关的一系列参数，并在事故发生时把该段时间内的历史数据保存下来，即事故追忆[44]。在变电站监控软件中，事故追忆是一个非常重要的功能。目前事故追忆比较普遍的实现方式为专用软件记录，这种方式可以使用各种通信手段采取各种通信协议和多种厂商的仪表通信，可扩展，可维护性都较高。但是因为此类软件仅仅用于记录事故,所以它缺乏灵活的数据显示手段和丰富的数据分析方法，也缺少有效的控制手段[45]。

事故追忆功能模块由以下几个部分组成[46]:

(l)事故组态部分它嵌人到单元的组态界面中，提供一个简单的界面让用户定义事故追忆点、事故触发条件、相关记录点和保存时间等信息。事故追忆点指的是事故发生的具体单元的具体变量，如 MMP111 单元的 YCI，后面的事故处理部分就会不断地跟踪事故追忆点的数据并且进行判断处理。

事故触发条件是需要用户定义的另一个比较重要的信息，如模拟量越限需要指明事故发生的上限或者下限，开关量变位需要指明开关位变化方式(由 O 变 1 或者由 1 变 0)。当跟踪的数据满足用户定义的触发条件时，系统认为事故已经发生，进行后续处理。记录点就是指用于所需要保存的变量名称，如当事故发生后，用户想查看 YCZ 的信息，那么 YCZ 就是一个记录点，或者称为相关记录点，用户定义的这些信息都将被保存到数据库中。

(2)事故处理部分它作为服务器上的一个事务，能够对用户定义的一系列事故追忆点的值连续不停地进行跟踪，一旦它发现用户定义的事故已经出现，就会按照用户的设定把跟踪的数据保存下来。在创建事故追忆点时，用户可以指定一组记录点和记录范围。记录的范围是一个时间段，它由“事件前时间”和“事件后时间”组成。当系统运行时，事故追忆处理事务就会连续地跟踪所有相关记录点的值。当事故发生后，不管事故立即消失还是一直保持，事故追忆的处理过程都不会立刻结束，处理事务会继续跟踪各个记录点的值，直到事件发生后的时间段

和“事件后时间”相等，这时处理事务会把此时间段(长度为事件前时间加上事件后时间)的数据保存，并且继续跟踪各个记录点准备记录下一个事故。如果一个事故发生后很快又消失并在较短的时间间隔后又重新产生，那么很可能两次事故发生的间隔小于用户定义的“事件后时间”，也就是说，当第二个事故发生时，第一个事故的记录还没有结束，这时处理事务就会立刻结束第一个事故将其保存并且立即开始处理第二个事故，而忽略掉用户定义的“事后时间”。

(3)事故分析部分作为一个独立的模块设计，在运行时，它可以显示出用户定义的所有事故追忆点，并针对每一个事故追忆点，按照时间的先后顺序依次列举出已经发生的所有事故。对于每一个已经发生的事故，用户可以方便地查看所有相关记录点的数据曲线，并且可以任意放大曲线的任何部分，对曲线进行分析。

4.5.2 SOE 处理

随着火电机组随着火电机组日趋规模化和复杂化，生产过程信息瞬间千变万化。当机组发生故障时需要查找出真实原因，并采取相应措施，这时就需要对事件进行追忆打印。

SOE 为英文 Sequence Of Event 的缩写，即事件顺序记录[47]。而一般的事件记录只能做到秒级的分辨率，当事件发生后，往往同一秒内出现的信息很多，且不能分出先后顺序，这就给事故分析造成了很大的困扰。而 SOE 就能更精确地反映事件情况，它能以毫秒级的分辨率获取事件信息，为事故分析提供有力的证据。SOE 系统的输入信号全部为开关量信号，它以高分辨率分辨各个信号状态变化的先后次序，帮助在事故情况下分辨故障的原因，找出首出故障，因此 SOE 成为分析事故的主要记录手段。

4.5.3 报警处理

组态软件自动对实时数据库中报警定义有效的数据变量进行监控。当发生报警事件，把这些事件存于内存中的报警事件缓冲器中，报警事件在缓冲器中是以先进先出的队列形式存储，所以只有最近的报警事件存于内存中。历史报警窗口的报警事件都是取自报警缓冲器。如果一个数据变量包含于报警窗口的报警组，而且变量定义时报警优先级不大于报警窗口的报警优先级时，报警事件就会在报警窗口中显示出来，其显示格式由报警窗口定义时报警信息格式选项规定。报警窗口类型有实时报警窗口和历史报警窗口两种。报警数据在实时数据库中处理和

保存。报警记录是用来显示和确认报警数据的窗口，使用两种预定义的类型：实时报警和历史报警。实时报警只反映当前未确认和确认的报警。如果经过处理后一个报警返回到正常状态，则这个变量的状态变为恢复状态，它前面产生的报警状态从显示出中消失。历史报警反映了所有发生过的报警事件。

4.6 本章小结

网络通信是现代监控软件的不可或缺的一部分，它不仅提高系统的分布性，而且分散系统风险，并为监控网络融入企业信息管理系统提供良好接口。在数据传输设计时，必须进行通用化的设计，通过定义数据类 ID 可以区分不同的数据传输桢，因此可在网络上同时传输多种类型数据。数据采集模块是组态软件数据输入输出的关键，是组态软件联系数据模型的纽带，是实现分布式系统的载体。本章主要是初步探讨了数据通信系统设计的所需技术和实现方案。网络通信设计，采用 Winsock 来实现接口。

第五章 数据库访问模块设计与实现

在本应用中，对数据处理的实时性要求很高。数据采集模块采集到数据是一个网络操作，耗时、时间不确定，数据采集模块要随时准备好采集 DPU 送到 HMI 的实时数据。DPU 发送实时数据的周期是 1s。在送上来的这些数据，要根据本系统的业务逻辑，将需要永久保存的数据写入磁盘，以备以后查询。写数据库也是一个相当耗时的操作。要在 1s 中进行以上两个操作，是基本不可能的。因此引入了数据库缓冲访问技术。

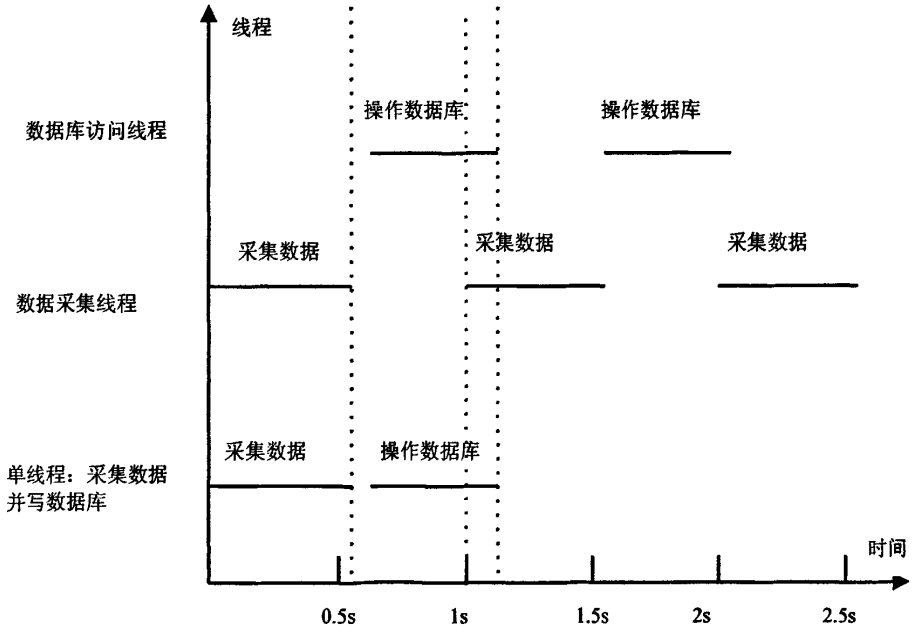

图 5-1 单线程和数据库缓冲访问对比

从以上图表可一看出，如果不引入数据库缓冲访问时，采集数据和操作数据库要耗时约 1.2 秒，超出 DPU 发送数据周期 1s。那么当数据采集线程在 1.2 秒时再去采集数据时将错过 DPU 发送的一些数据，这对该系统将是不可恢复的错误。引入数据库缓冲访问技术后，不仅可以保证数据采集模块可以采集到正确的实时数据，而且可以平滑对数据库的访问。因为，所有对数据库的操作都被放入一个

队列中，实现缓冲。该缓冲队列涉及到多线程的读写，主要应用多线程编程的线程间通信，互斥和同步。下面将对 Windows 平台下这些对象进行介绍。

5.1 进(线)程间通信及同步互斥

5.1.1 Windows 内核对象

内核对象只是内核分配的一个内存块，并且只能由该内核访问[48]。该内存块是一种数据结构，它的成员负责维护该对象的各种信息。内核对象有以下属性：

（1）内核对象的数据结构只能被内核访问，应用程序中不能直接改变这些数据结构，Windows提供了一组函数，以便用定义得很好的方法来对这些结构进行操作。这些内核对象始终都可以通过这些函数进行访问。当调用一个用于创建内核对象的函数时，该函数就返回一个用于标识该对象的句柄。该句柄可以被视为一个不透明值，进程中的任何线程都可以使用这个值。将这个句柄传递给Windows的各个函数，这样，系统就能知道是在操作哪个内核对象。

（2）内核对象由内核所拥有，而不是由进程所拥有。如果进程调用了一个创建，内核对象的函数，然后进程终止运行，那么内核对象不一定被撤消。在大多数情况下，对象将被撤消，但是如果另一个进程正在使用进程创建的内核对象，那么该内核知道，在另一个进程停止使用该对象前不要撤消该对象，因此内核对象的存在时间可以比创建该对象的进程长。内核对象通过引用计数做到的。内核知道有多少进程正在使用某个内核对象，因为每个对象包含一个使用计数。使用计数是所有内核对象类型常用的数据成员之一。当一个对象刚刚创建时，它的使用计数被置为1。然后，当另一个进程访问一个现有的内核对象时，使用计数就递增1。当进程终止运行时，内核就自动确定该进程仍然打开的所有内核对象的使用计数。如果内核对象的使用计数降为0，内核就撤消该对象。这样可以确保在没有进程引用该对象时系统中不保留任何内核对象。

下面将要提到的对象都是Windows的内核对象。

5.1.2 Win32 中进程间通信

在 Windows 系统中，为实现进程间的数据交换，可采用以下几种方法[49]：使用内存映射文件、管道、WM_COPYDATA 消息、Windows 套接字(Windows Socket)。

1.利用内存映射文件实现 WIN32 进程间的通讯

Windows98 中的内存映射文件的机制为我们高效地操作文件提供了一种有效途径，它允许我们在 WIN32 进程中保留一段内存区域，把目标文件映射到这段虚拟内存中，通过内存映射文件，可以将磁盘上文件的全部和部分映射为一个视图到进程虚拟地址空间的某个位置，一旦完成了映射视图，对文件内容的访问就如同在该地址区域内直接对指针取值一样简单。原来内存映射文件只映射类似磁盘一类的存储器上的文件，而为了更快速地在进程之间通信，内存映射文件还可提交物理内存。可以利用内存映射文件实现进程间的通信，通过访问同一个内存映射文件对象，两个进程或多个进程就能够访问到同一块物理内存，这样一个进程写到物理内存的数据，其它进程就能够看到了。在程序实现中必须考虑各进程之间的同步。下面给出使用内存映射文件的一般方法：首先要通过 CreateFile()函数来创建或打开一个文件内核对象，这个对象标识了磁盘上将要用作内存映射文件的文件。在用 CreateFile()将文件映像在物理存储器的位置通告给操作系统后，只指定了映像文件的路径，映像的长度还没有指定。为了指定文件映射对象需要多大的物理存储空间还需要通过 CreateFileMapping()函数来创建一个文件映射内核对象，以告诉系统文件的尺寸以及访问文件的方式。与虚拟内存类似，内存映射文件的保护方式可以是 PAGE_READONLY 或是 PAGE_READWRITE。如果多进程对同一内存映射文件对象进行写访问，则必须保持进程间同步。文件映射内核对象还可以指定 PAGE_WRITECOPY 标志，可以保证其原始数据不会遭到破坏，同时允许其他进程在必要时自由的操作数据的拷贝。

在创建了文件映射内核对象后，还必须为文件数据保留一个地址空间区域，并把文件数据作为映射到该区域的物理存储器进行提交，可以通过 MapViewOfFile()函数将文件映射对象的全部或部分映射到进程的地址空间。其他进程访问该内存映射文件，只需要获得文件映射对象名并通过调用 OpenFileMapping()函数就可以实现。一旦其他进程获得了文件映射内核对象的句柄后，就可以调用 MapViewOfFile()函数来映射对象视图，用户可以使用该对象视图来进行数据读写操作，以达到数据通讯的目的。当用户进程当完成对内存映射文件的使用后，调用 UnMapViewOfFile()函数以取消其地址空间内的视图。

2.管道

管道是连接读写进程的一个特殊文件，允许进程按先进先出方式传送数据，也能使进程同步执行操作。发送进程以字符流形式把大量数据送入管道，接收进程从管道中接收数据，所以叫管道通信，管道分为匿名管道和命名管道。匿名管

道(Anonymous Pipe)是在父进程和子进程之间，或同一父进程的两个子进程之间传输数据的无名字的单向管道。通常由父进程创建管道，然后由要通信的子进程继承通道的读端点句柄或写端点句柄以实现通信。父进程还可以建立两个或更多个继承匿名管道读和写句柄的子进程，这些子进程可以使用管道直接通信，不需要通过父进程。匿名管道是在单机上实现子进程标准 I/O 重定向的有效方法，它不能在网上使用，也不能用于两个不相关的进程之间通信。命名管道(Named Pipe)是服务器进程和一个或多个客户进程之间通信的单向或双向管道。不同于匿名管道的是命名管道可以在不相关的进程之间和不同计算机之间使用，服务器建立命名管道时给它指定一个名字，任何进程都可以通过该名字打开管道的另一端，根据给定的权限和服务器进程通信。命名管道提供了相对简单的编程接口，使通过网络传输数据并不比同一计算机上两进程之间通信更困难，不过它不适合同时和多个进程通信。

3.用于传输只读数据的 WM_COPYDATA

传输只读数据可以使用 Win32 中的 WM_COPYDATA 消息，该消息的主要目的是允许在进程间传递只读数据。SDK 文档推荐用户使用 SendMessage 函数，接受方在数据拷贝完成前不返回，这样发送方就不可能删除和修改数据。SendMessage(hwnd,WM_COPYDATA,wParam,lParam);其中 wParam 设置为包含数据的窗口的句柄，lParam 指向一个 COPYDATASTRUCT 的结构，该结构用来定义用户数据。

4.Windows Sockets

网络套接口，可在不同主机间交换数据，分为服务器方和客户方。现在通过 Sockets 实现进程通信的网络应用越来越多，这主要的原因是 Sockets 的跨平台性要比其它 IPC 机制好得多，另外 WinSock 2.0 不仅支持 TCP/IP 协议，而且还支持其它协议(如 IPX)。

5.1.3 进(线)程间的同步互斥

线程同步是指线程之间所具有的一种制约关系，一个线程的执行依赖另一个线程的消息，当它没有得到另一个线程的消息时应等待，直到消息到达时才被唤醒。线程互斥是指对于共享的操作系统资源(指的是广义的"资源"，而不是 Windows 的.res 文件，譬如全局变量就是一种共享资源)，在各线程访问时的排它性。当有若干个线程都要使用某一共享资源时，任何时刻最多只允许一个线程去

使用，其它要使用该资源的线程必须等待，直到占用资源者释放该资源。线程互斥是一种特殊的线程同步。实际上，互斥和同步对应着线程间通信发生的两种情况：

1)当有多个线程访问共享资源而不使资源被破坏时；

2)当一个线程需要将某个任务已经完成的情况通知另外一个或多个线程时在WIN32 中，互斥同步机制主要有以下几种[50]：临界区、事件(Event)、信号量(semaphore)、互斥量(mutex)；

临界区(Critical Section)是一段独占对某些共享资源访问的代码，在任意时刻只允许一个线程对共享资源进行访问。如果有多个线程试图同时访问临界区，那么在有一个线程进入后其他所有试图访问此临界区的线程将被挂起，并一直持续到进入临界区的线程离开。临界区在被释放后，其他线程可以继续抢占，并以此达到用原子方式操作共享资源的目的。临界区在使用时以 CRITICAL_SECTION 结构对象保护共享资源，并分别用 EnterCriticalSection()和 LeaveCriticalSection()函数去标识和释放一个临界区。

在使用临界区时，一般不允许其运行时间过长，只要进入临界区的线程还没有离开，其他所有试图进入此临界区的线程都会被挂起而进入到等待状态，并会在一定程度上影响程序的运行性能。尤其需要注意的是不要将等待用户输入或是其他一些外界干预的操作包含到临界区，如果进入了临界区却一直没有释放，同样也会引起其他线程的长时间等待。换句话说，在执行了 EnterCriticalSection()语句进入临界区后无论发生什么，必须确保与之匹配的 LeaveCriticalSection()都能够被执行到，可以通过添加结构化异常处理代码来确保 LeaveCriticalSection()语句的执行。虽然临界区同步速度很快，但却只能用来同步本进程内的线程，而不可用来同步多个进程中的线程。

事件(Event)是 WIN32 提供的最灵活的线程间同步方式，事件可以处于激发状态(signaled or true)或未激发状态(unsignal or false)。根据状态变迁方式的不同，事件可分为两类：

1)手动设置：这种对象只能用程序手动设置，在需要该事件或者事件发生时，采用 SetEvent 及 ResetEvent 来进行设置。

2)自动恢复：一旦事件发生并被处理后，自动恢复到没有事件状态，不需要再次设置。在创建线程前，首先利用 CreateEvent()函数创建一个可以自动复位的事件内核对象 hEvent，而线程函数则通过 WaitForSingleObject()函数无限等待 hEvent 的置位，只有在事件置位时 WaitForSingleObject()才会返回，被保护的代

码将得以执行。对于以自动复位方式创建的事件对象，在其置位后一旦被WaitForSingleObject()等待到就会立即复位。使用临界区只能同步同一进程中的线程，而使用事件内核对象则可以对进程外的线程进行同步，其前提是得到此事件对象的访问权。可以通过 OpenEvent()函数获取得到，如果事件对象已创建(在创建事件时需要指定事件名)，函数将返回指定事件的句柄。对于那些在创建事件时没有指定事件名的事件内核对象，可以通过使用内核对象的继承性或是调用DuplicateHandle()函数来调用 CreateEvent()以获得对指定事件对象的访问权。在获取到访问权后所进行的同步操作与在同一个进程中所进行的线程同步操作是一样的。

信号量(Semaphore)内核对象对线程的同步方式与前面几种方法不同，它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目[51]。在用 CreateSemaphore()创建信号量时即要同时指出允许的最大资源计数和当前可用资源计数。一般是将当前可用资源计数设置为最大资源计数，每增加一个线程对共享资源的访问，当前可用资源计数就会减 1，只要当前可用资源计数是大于 0 的，就可以发出信号量信号。但是当前可用计数减小到 0 时则说明当前占用资源的线程数已经达到了所允许的最大数目，不能在允许其他线程的进入，此时的信号量信号将无法发出。线程在处理完共享资源后，应在离开的同时通过 ReleaseSemaphore()函数将当前可用资源计数加 1。在任何时候当前可用资源计数决不可能大于最大资源计数。使用信号量内核对象进行线程同步主要会用到 CreateSemaphore()、OpenSemaphore()、ReleaseSemaphore()、WaitForSingleObject()和 WaitForMultipleObjects()等函数。其中，CreateSemaphore()用来创建一个信号量内核对象，由于其创建的是一个内核对象，因此在其他进程中可以通过该名字而得到此信号量。OpenSemaphore()函数即可用来根据信号量名打开在其他进程中创建的信号量。在线程离开对共享资源的处理时，必须通过 ReleaseSemaphore()来增加当前可用资源计数。否则将会出现当前正在处理共享资源的实际线程数并没有达到要限制的数值，而其他线程却因为当前可用资源计数为 0 而仍无法进入的情况。WaitForSingleObject()和 WaitForMultipleObjects()主要用在试图进入共享资源的线程函数入口处，主要用来判断信号量的当前可用资源计数是否允许本线程的进入。只有在当前可用资源计数值大于 0 时，被监视的信号量内核对象才会得到通知。

互斥(Mutex)是一种用途非常广泛的内核对象。能够保证多个线程对同一共享资源的互斥访问[52]。同临界区有些类似，只有拥有互斥对象的线程才具有访问资

源的权限，由于互斥对象只有一个，因此就决定了任何情况下此共享资源都不会同时被多个线程所访问。当前占据资源的线程在任务处理完后应将拥有的互斥对象交出，以便其他线程在获得后得以访问资源。以互斥内核对象来保持线程同步可能用到的函数主要有 CreateMutex()、OpenMutex()、ReleaseMutex()、在使用互斥对象前，首先要通过 CreateMutex()或 OpenMutex()创建或打开一个互斥对象。如果在创建互斥对象时指定了对象名，那么可以在本进程其他地方或是在其他进程通过 OpenMutex()函数得到此互斥对象的句柄。当目前对资源具有访问权的线程不再需要访问此资源而要离开时，必须通过 ReleaseMutex()函数来释放其拥有的互斥对象。在编写程序时，互斥对象多用在那些被多个线程所访问的内存块的保护上，可以确保任何线程在处理此内存块时都对其拥有可靠的独占访问权。事件，信号量，互斥量为内核对象，可以进行进程间的同步。

5.2 数据库缓冲访问设计思想

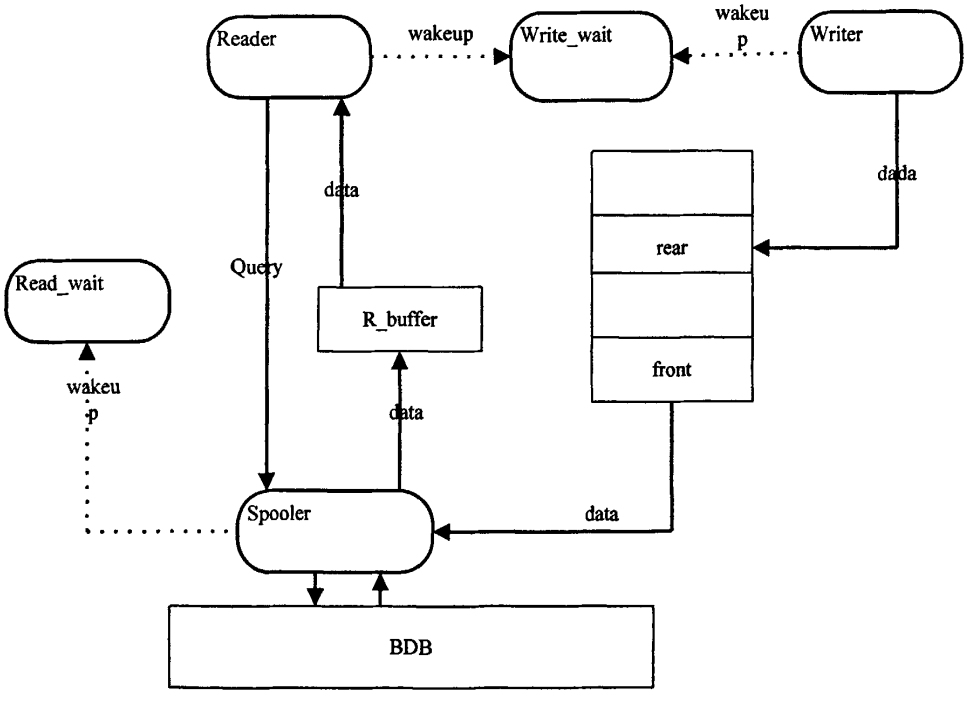

图 5-2 数据库访问模块结构图

本系统共需 3 个进(线)程，数据库缓冲访问进程和读写进(线)程（Eidtor 、View、数据采集进程均划为这两者之一或都是），如图 5-2 所示。

Write 进(线)程：每秒钟向数据库插入 8 万字节的数据；Read 进(线)程：不定时的向数据库提出读数据请求。

整个调度系统的设计思想是在内存中设置了一块缓冲区，包含一个写缓冲队列和读缓冲块，共 3 个进(线)程来对这个缓冲区进行操作，写进(线)程每秒钟从网络上接收 IO 点数据后就把数据插入到写缓冲队列的队尾，并进入等待状态，等待下一秒的数据到来，同时唤醒数据库缓冲访问进程，而数据库缓冲访问进程则不断的从队头读取数据并写入实时数据库，与此同时，读进(线)程处于等待或者未调用状态。而当读进(线)程需要读取数据库时，就会向数据库缓冲访问进程发送一个读取数据的请求，数据库缓冲访问进程从数据库里读取数据到读缓冲块(数据库缓冲访问进程根据读进(线)程所发出的请求，按照一定的时间规则从数据库里查找读取一条记录到缓冲块)，并通知读进(线)程已经准备好数据，然后读进(线)程从读缓冲块里读取数据，此时写进(线)程处于空闲或等待状态。当读写任务共同请求时，写任务优先。

读/写进程负责将写请求放入数据库访问队列，数据库缓冲访问进程按照队列先进先出的规则，取出访问请求，根据参数访问数据库。在这中间涉及到多进程访问同步，写进程要访问队列，数据库缓冲进程也要取下数据库请求。对队列的访问要进行严格的保护，要加锁。多个进程访问同一块内存，涉及到进程间通信，在 Windows 中，最有效率的的进程见通信是通过映射文件实现的。在该项目中，也是采用这一方案。这里给出一个模型，其相关算法如下：

1.写进(线)程所调用的算法

```
对写队列缓冲区上锁；
取队尾元素；
对队尾元素进行赋值；
判断等待信号量是否为真；
If（信号量为真）
{
    把信号量置为0；
    唤醒数据库缓冲访问进程；
}
对写队列缓冲区解锁；
```

2.读进(线)程所调用的算法：

```
读进(线)程发出读数据请求；
```

```
对写队列缓冲区上锁；
判断等待信号量是否为真；
if(等待信号量为真)
{
    把信号量置为0；
    唤醒数据库缓冲访问进程；
}
对写队列缓冲区解锁；
对事件信号量赋值，阻塞等待读操作完成；
读取读缓冲块中的元素；
```

3. 数据库缓冲访问进程所调用的算法

```
While(true)
{
    对写队列缓冲区上锁；
    把等待信号量置为 1；
    对写队列缓冲区解锁；
    睡眠等待读写进程来唤醒此进程；
    while(有读请求||写队列不空)
    {
        while(写队列不空)
        {//有写请求
            对写队列缓冲区上锁；
            从缓冲区中取数据；
            对写队列缓冲区解锁；
            把数据写入数据库；
        }
        如果有读请求；
        从数据库取数据写入读缓冲块；
        置位完成读数据请求；
    }
}
```

5.3 生产者-消费者模型

数据库缓冲访问被抽象出来后，实质上是一个生产者-消费者模型。生产者-消费者（producer-consumer）问题是一个著名的进程同步问题它描述的是：有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置一个具有 n 个缓冲区的缓冲池，生产者进程将它所生产的产品放入一个缓冲区中；消费者进程可以从一个缓冲去中取走产品去消费。尽管所有的生产者和消费者进程都是以异步方式运行的，但它们之间必须保持同步，即不允许消费者进程到一个空缓冲区中去取产品；也不允许生产者向一个已装满产品且尚未被取走的缓冲区中投放产品。

假定在生产者和消费者之间的公用缓冲池中，具有 n 个缓冲区，这时可利用互斥量 Mutex 实现诸进程对缓冲池的互斥使用；利用信号量 empty 和 full 分别表示缓冲池中空缓冲和满缓冲的数量。

在生产者-消费者问题中应注意：首先，在每个程序中用于实现互斥的 Wait(Mutex)和 Signal(Mutex)必须成对地出现；其次，对资源信号量 empty 和 full wait 和 signal 操作，同样需要成对地出现，但它们分别处于不同的程序中。例如，wait(empty)在计算进程中，而 signal(empty)则在打印进程中，计算进程若因执行 wait(empty)而阻塞，则以后将由打印进程将它唤醒；最后，在每个程序中的多个 wait 操作顺序不能颠倒。应先执行对资源信号量的 wait 操作，然后再执行对互斥信号量的 wait 操作，否则可能引起进程的死锁。

5.4 多进程同步访问队列的实现

在数据库缓冲访问模块中，最重要的是实现一个队列，该队列要满足下列要求：

1) 多进程访问。不止一个进程要访问该队列。如何实现多个进程访问同一块内存呢。每个进程都被赋予它自己的虚拟地址空间。对于 32 位进程来说，这个地址空间是 4GB，由于每个进程可以接收它自己的私有的地址空间，因此当进程中的一个线程正在运行时，该线程可以访问只属于它的进程的内存。属于所有其他进程的内存则隐藏着，并且不能被正在运行的线程访问。因此普通内存地址分配不适合。必须运用 Windows 提供的，最有效的进程间通信机制：内存映射文件。

2）多线程同步访问。在对队列的访问中必须实现同步。

综合以上两个要求：用于本项目的队列是一个在内存映射文件中实现的，线程访问安全的队列。

5.4.1 多线程同步访问队列

队列使用一个互斥对象和一个信标来控制数据元素的队列。服务器线程在队列中出现第一个元素之前一直无事可做。当第一个元素出现时，一个服务器线程醒来，对该请求进程处理。服务器线程无法以足够快的速度来处理客户程序的请求，队列中的请求达到了最大的容量。对队列数据结构进行了初始化，使它一次能够存放的元素不能超过 32 个，这将导致队列很快被填满。

图 5-3 为队列类的类结构图， 其中 m_pElements 它指向一个固定大小的 Elements 结构的数组。这是需要保护使之不受多线程访问影响的数据；m_nMaxElements 指队列的最大数；m_h[2]是两个 Windows 句柄，互斥量和信标句柄；&m_hmtxQ：互斥量句柄的引用；m_hsemNumElements：信标句柄的引用。

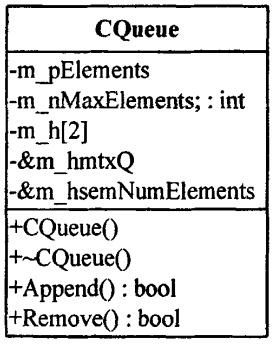

图 5-3 队列类结构图

Append 方法试图将一个 Elements 附加给队列。但是，线程首先必须确保它拥有对该队列的独占访问权。Append 方法通过调用 WaitForSingleObject 函数，传递 m_hmtxQ 互斥对象的句柄，实现其独占访问权。如果返回 WAIT_OBJECT_0，那么该线程就拥有对该队列的独占访问权。

接着，Append 方法必须调用 ReleaseSemaphore 函数，传递释放数量 1，以便使队列中的元素数量递增。如果 ReleaseSemaphore 函数调用成功，队列中的元素没有放满，那么新元素就可以附加给队列。幸好 ReleaseSemaphore 函数返回了 lPreviousCount 变量中的队列元素的前一个数量。这确切地告诉你新元素应该放入

哪个数组索引中。当该元素被拷贝到队列的数组中后，该函数就返回。一旦该元素完全附加给队列，Append 便调用 ReleaseMutex 互斥对象，这样其他线程就能够访问该队列。Append 函数的剩余部分与故障情况和错误处理有关。

如何调用 Remove 方法，以便从队列中取出元素的。首先，线程必须确保它拥有对队列的独占访问权，同时队列中至少必须拥有一个元素。当然，如果队列中没有任何元素，那么服务器线程就没有理由被唤醒。因此，Remove 方法首先要调用 WaitForMultipleObject，并且同时传递互斥对象和信标的句柄。只有当这两个对象都得到通知时，服务器线程才被唤醒。

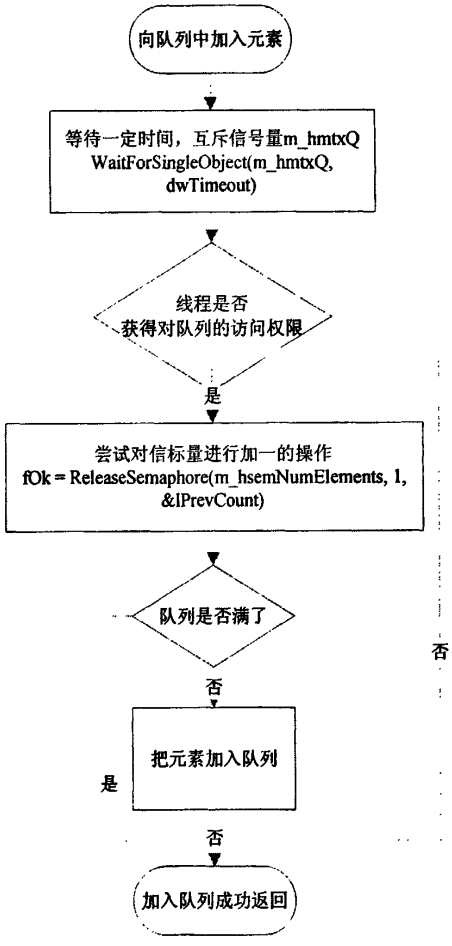

图 5-4 函数 Append 的流程图

如果返回 WAIT_OBJECT_0，该线程将拥有对队列的独占访问权，并且队列

中至少必须有一个元素。这时，代码就取出数组中索引号为 0 的元素，最后，要调用 ReleaseMutex 函数，这样其他线程就能安全地访问该队列。

注意，信标对象能够随时跟踪某个时间队列中存在多少个元素。你能够看到这个数字在递增。当一个新元素被附加给队列时，Append 方法就调用 ReleaseSemaphore。但是，当一个元素从队列中删除时， 你无法立即看到这个数字递减。递减是由 Remove 方法调用 WaitForMultipleObjects 函数来进行的。成功地等待信标的副作用是它的数量递减 1。

该队列可以实现多个线程的同时访问，但是队列内存地址空间在一个进程中被分配，另外一个进程不能访问到该地址空间。对于 32 位进程来说，这个地址空间是 4GB，由于每个进程可以接收它自己的私有的地址空间，因此当进程中的一个线程正在运行时，该线程可以访问只属于它的进程的内存。属于所有其他进程的内存则隐藏着，并且不能被正在运行的线程访问。为了实现该项目所需的多个进程同步访问队列，为此引入了内存映射文件。接下来将要讲到的就是内存映射文件的实现。

5.4.2 内存映射文件实现

上一节实现了一个多线程同步访问的队列，将该队列的地址空间用内存映射文件分配，该内存映射文件是 Windows 内核对象，在系统范围内有效。用于多线程同步的互斥量、信号量也都是 Windows 内核对象，因此该模型完全符合项目要求。

1.创建文件内核对象

```
HANDLE hFile=CreateFile("c:\\mapping.txt",//文件名
                GENERIC_READ|GENERIC_WRITE,//具有读写权限
                FILE_SHARE_READ|FILE_SHARE_WRITE,//文件共享读写
                NULL,
                CREATE_ALWAYS,//如果磁盘文件不存在，就重新创建一个
                FILE_ATTRIBUTE_NORMAL,//文件属性
                NULL);
```

2.创建文件映射内核对象

```
m_hMap=CreateFileMapping(hFile,NULL,PAGE_READWRITE,0,1400000,"
                    SHARED");
```

文件映射对象所需的物理存储器为 1.4M，并定义了文件映射对象的名字，以方便多个进程共享此文件映射对象。当其他进程使用该对象时，只需要通过调用 OpenFileMapping()函数就可打开此文件映射对象。

```
HANDLE m_hMap=OpenFileMapping(FILE_MAP_ALL_ACCESS,FALSE,
        "SHARED");
```

3.将文件映射对象的全部或一部分映射到进程地址空间中

当创建了一个文件映射对象后，仍然必须让系统为文件的数据保留一个地址空间区域，并将文件的数据作为映射到该区域的物理存储器进行提交。可以通过调用 MapViewOfFile 函数来进行这项操作：

```
LpData=(LPBYTE)MapViewOfFile(m_hMap,FILE_MAP_ALL_ACCESS,0,0,
    0);
```

5.5 数据库访问模块测试

在未使用缓冲访问数据库时，如果历史库组态时进入历史库的点太多，当达到 3 万个点时，就会出现数据丢失的现象并且 CPU 的使用率也达到 70%以上。采用缓冲访问数据库后，缓冲了对数据库的访问，平滑了磁盘读写，降低了 CPU 使用率。

5.5.1 硬件环境

运行环境工程师平台：Intel 奔腾 4 2.0G 处理器，1G 内存，80G 硬盘

测试网络：为本地局域网、教育网。

5.5.2 软件环境

基于 Windows XP SP2 平台。

5.5.3 测试结果

查找，读取，存储记录的速度是衡量数据库性能好坏的重要标准，对数据库进行了大数据量测试，看其在进行大数据量操作时，实时性能如何。并且模拟现场工业设备运行。

从表 5-1 中的数据可以看到，当对数据库进行大数据量的操作时，平均插入

一条记录所消耗的时间不超过 25 毫秒，完全能够满足工程的需求。

表 5-1 数据库插入操作测试

字段数	字段类型	字段大小(字节)	插入条数	花费时间（秒）
40000	整数	4	1	0.004
40000	整数	4	10	0.15
30000	整数	4	1	< 0.002
30000	整数	4	10	0.08

在数据库中，数据是以时间为关键字进行存储的，每一秒的数据都作为一条记录存储在数据库中。根据项目的需要，实时趋势，运行画面等模块可能需要读取当前一秒或多秒的数据，下面将会对查询读取记录的操作进行测试。

表 5-2 数据库读取操作测试

读取记录条数	消耗时间（秒）
1	0.001
10	0.005

配置下位机 4 万个点，进入历史库 3 万个点，同时配置 SOE、报警等随机事故，把整个系统运行起来后，系统运行平稳，无丢数据现象。下面是采用数据库缓冲访问和未采用数据库缓冲访问的对比测试结果：

表 5-3 数据库访问模块测试

	数据采集丢失率	写入数据库失败	CPU 利用率
未使用缓冲访问	0.1%	0.05%	70%~100%
使用缓冲访问	无	无	2%~6%

5.6 本章小结

本章介绍了数据库访问模块，将数据库访问抽象成三元组（数据库，操作，数据），该三元组唯一标识一个数据库访问。并将该三元组放入一个缓冲队列中，实现对数据库的缓冲访问。

第六章 结论和展望

本课题的研究工作是以组态软件图形组态系统设计为中心，以数据模块为支撑，目前该系统已通过验收。

本课题所作的研究工作具有以下特点：

1.系统架构设计合理、结构清晰、可扩展性强。组态软件图形组态系统的架构设计充分利用了面向对象的建模技术。

2.画面组态元素丰富。系统除提供基本矢量图元之外，又提供了丰富的图形库，以及控件库，这使得图形界面开发系统能以生动的画面来模拟并监控现场的运行情况。

3.画面编辑工具灵活方便。开发系统提供灵活的画面编辑工具，用户可通过鼠标、键盘方便的创建图元、调整图元位置、对图元复制删除以及对图元进行动画连接参数设置。这些都大大减少了自动化工程师创建组态工程的工作量。

4.支持监控组态软件的数据模块的实现，并完成了实时数据与图形界面图元之间的数据通信。

5.本系统由于采用了数据库缓冲访问技术，大大提高了该软件处理工业 IO 点的数目，可以流畅处理 4 万个工业采集点数据。

由于组态软件图形组态系统结构复杂，功能繁多，涉及到的技术先进，所以本系统与国内外优秀的同类产品相比，还有很多方面需要进一步完善，具体表现在：

1)考虑到目前基于 Windows 系统的平台在大型业务应用中存在一些局限，比如系统负载能力不足、稳定性差等，而基于 Unix 的大型机和小型机在这个方面具有得天独厚的优势。这方面国外产品如 OSI PI 就同时推出基于 Windows NT,Solaris,VMS 三种操作系统版本的产品。我们以后完全可以考虑开发基于 Unix 系统的实时数据库系统，充分利用硬件资源，使其在大型应用中更好地发挥作用。

2)在工业控制中，控制现场的数据随着生产的进行而不断地产生，对整个工业过程来说，历史数据的数据量是非常庞大的。虽然可以利用增大磁盘空间和历史数据转出来增加历史数据的储存量，但是，研究历史数据的压缩算法还是非常重要的，国外著名的实时数据库系统 PI 就是采用旋转门压缩算法来对历史数据进行压缩[51]

致谢

本文是在导师李毅教授的悉心指导下完成的。在论文的选题以及课题的研究过程中，导师给予了我认真的指导和大力的支持。在两年半的研究生学习生涯中，导师灵活的思维和宽广的思路、严谨的治学和工作态度、简朴的生活作风，这些都使我受益终身；在研究生学习期间他对我进行了认真的指导，使我在学业上受益匪浅。在论文完成过程中他给予我思路上的点拨，使我少走了不少弯路，并且在论文的修改过程中给出了很多有用的建议和指出了不足的地方，使本文趋于完善。在此，我向李老师表示衷心的感谢！

同门的姜松、冯浩、陈小军等师兄师弟在论文的完成过程中给予了积极的合作和帮助，在此一并谢过。

最后，我还要感谢我的父母，他们给予了我生活和精神上的支持和鼓励，使我能够安心地完成学业和论文工作。

参考文献

[1] 李丹.用于工业过程控制的一种新工具——组态软件[J].计算机技术与自动化,1995,（3）:5-8.

[2] 王亚民，编著.组态软件设计与开发.西安:西安电子科技大学出版社，2003.

[3] 马国华.监控组态软件及其应用.北京：清华大学出版社，2001.1-3

[4] 倪星．工业控制组态软件的产品对比及发展趋势[J]．测控技术，2000,19(9)：38-15(4)

[5] 欧金城，欧世乐，林德杰．组态软件的现状与发展[J]．工业控制计算机，2002,15(4)：1-5

[6] 葛玻，沈文杰，赵旎．工控组态软件的对比及应用[J]．计算机测量与控制，2002,10(8)：550-552

[7] 孙浩．工业控制组态软件的发展现状及应用[J].冶金自动化,1996,（6）:13-15.

[8] 曾庆波，孙华，周卫宏编著．监控组态软件及其应用技术．哈尔滨:哈尔滨工业大学出版社.2005.

[9] 阚宏进.基于VC++工控组态软件的总体框架性设计与实现.甘肃工业大学.2000.4

[10] http://www.gongkong.com.cn/

[11] http://www.kingview.com/kingview for internet

[12] Arnold D.B.,Reynolds,G.J.Configuring graphics systems components. SoftwareEngineering Journal,Volume：3,Issue：6,Nov.1988：248-256

[13] Kramer J, Magee J.Graphical support for configuration programming. System Sciences, 1989.Vol.2: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference, Volume：2,3-6 Jan. 1989: 860-870 vol.2

[14] 宿鲁艳. Windows环境下监控系统组态软件的开发研究．东南大学硕士论文. 2006.06

[15] Wheater,S.M.The design and implementation of a framework for configurable software. Configurable Distributed Systems,1996.Proceedings,Third International Conference on 6-8 May 1996：136-143

[16] Frank Buschmann, Regine Meunier, Hans Rohnert,Peter Sommerlad, Michael Stal. Pattern Oriented Software Architecture(Volume 1:A System of Patterns)[M]，机械工业出版社，2003.1

[17] 王燕.面向对象的理论与C++实践[M].北京：清华大学出版社，1997

[18] (美)Bjarne Stroustrup 著，裘宗燕译. C++程序设计语言（特别版）[M].北京：机械工业出版社，2002

[19] 李明. 基于面向对象技术的监控组态软件的研究[J]. 计算机应用与软件，2002,19(10):19-21

[20] Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides. Design Patterns(Elements of Reusable Object-Oriented Software)[M]. 机械工业出版社，2002.3

[21] 许斌.面向 DCS 应用的软件集成与控制技术的研究.东南大学硕士学位论文.2004.33

[22] 陈建春，矢量图形系统开发与编程[M]，北京：电子工业出版社，2004.1

[23] 严蔚敏，吴伟民.数据结构.北京：清华大学出版社，1997.63-65

[24] 希望图书创作室，Visual C++6.0 技术内幕[M]，北京:希望电子出版社，1999

[25] 钱能，C++程序设计教程[M]，北京:清华大学出版社，1999,124-128

[26] David J.Kruglinski.VC++技术内幕.北京：清华大学出版社，1999.222-225

[27] 吴修国，工控组态软件中图形组态子系统的设计与实现[J]，工业控制计算机，2001,14(7):31-33

[28] 徐晓东，杨振坤.中小型 DCS 组态软件的设计与实现.计算机工程与应用.

[29] MSDN 开发精选. 北京:《硅谷》杂志社. 2005 年 12 月，第六期.

[30] Hongtao Yang, Eagleson, R. Design and implementation of an Internet-based embedded control system. Control Applications, 2003, Vol. 1:1181-1185

[31] 赵磊. 监控组态软件中实时数据库接口的设计与实现. 电子科技大学硕士论文. 2007.05

[32] Ben Kao, K.Y.Lam,Brad Adelberg,et al. Updates and view maintenance in softreal-time database systems.In:Gauch, Susan, Soong, et al eds. Proceedings of the eighth international conference on Information and knowledge management. New York:ACM Press, 1999，300-307

[33] Michael, A.Olson. Selecting and implementing an embedded database system. Computer 2000, IEEE, 2000

[34] Anton Okmianski 编译/林崇亮.嵌入式数据库-朴素但实用的数据库选择.《程序员》. 2003.1

[35] http://www.oracle.com/database/berkeley-db,2007

[36] A Comparison of Oracle Berkeley DB and Relational Database Management Systems.An Oracle Technical White Paper, November 2006

[37] Building Robust Storage Systems with Oracle Berkeley DB,2006

[38] Oracle Berkeley DB:Performance Metrics and Benchmarks.An Oracle White Paper, August2006

[39] 周明天、汪文勇. TCPIIP 网络原理与技术[M]. 北京:清华大学出版社，1993.

[40] William Stallings. Operating Principles(Fourth Edition)[M],Systems: Internals and Design. 电子工业出版社，2001

[41] 谢希仁. 计算机网络. 大连:大连理工大学出版社

[42] 电力工业标准汇编电气卷电网调度自动化及通信. 北京:中国电力出版社，1999

[43] 李广春. 综合自动化技术应具有的基本功能研究. 辽宁工学院学报，2000(4)

[44] 赖明江, 耿英三, 张国钢, 王建华. 变电站监控软件中事故追忆功能模块的设计与实现. 电气应用，2005，Vol.24, NO12:27-28

[45] 樊大飞, 陈晓, 麦云飞. 西门子功能的实现及应用 工业控制计算机 2006.Vol19, NO1:77-79

[46] 木林森，高峰霞，奚红宇.Visual C++6.0 使用与开发.清华大学出版社，1997

[47] 王险峰，刘宝宏. Windows 环境下的多线程编程原理与应用. 北京：清华大学出版社，2002.55-95

[48] David J.Kruglinski.VC++技术内幕. 北京：清华大学出版社，1999.222-225

[49] 王文磊.多线程编程技术实现经典进程同步问题.计算机技术与发展，2006，Vol.16，No.3:110-114

[50] Jeffreyrichter. Windows 核心编程. 北京：机械工业出版社，1999.248-252

[51] 袁枚、袁文，数据压缩技术及其应用，电子工业出版社，1995

攻读硕士学位期间的研究成果

[1] 李杰，李毅. P2P 实时点播系统. 电脑知识与技术, 2008.04.

[2] 参加了某电气集团《DEA 工控组态软件》的设计与开发。