分类号：TP31　　学校代码：10699
密　级：　　学　　号：99711129

西北工业大学

博 士 学 位 论 文

（学位研究生）

题目 基于灵活内核的和欣操作系统研究

作 者　杜 永 文

指导教师 何 华 灿 **专业技术职务** 教 授

学科(专业) 计算机软件与理论

答辩日期 _____ **学位授予日期** _____

年　月　日

分类号：TP31　　学校代码：10699
密　级：　　学　　号：99711129

西北工业大学

博 士 学 位 论 文

（学位研究生）

题目 基于灵活内核的和欣操作系统研究

作　　者　　杜　永　文

指导教师　　何华灿教授

学科（专业）　　计算机软件与理论

西北工业大学计算机学院

2004 年 7 月

Researches on Agile-Kernel based Elastos Operating System

A Dissertation
Submitted For the Doctor's Degree of Philosophy
on Computer Software and Theory

by **Du Yongwen**

Under the Supervision of

Prof. **He Huacan**

College of Computer

Northwestern Polytechnical University, Xi'an, China

December, 2003

目　录

摘　要

为了真正实现中华复兴的伟大理想，我国必须拥有具有自主知识产权的 CPU 芯片和操作系统，北京科泰世纪有限公司是我国目前唯一一家正在研发具有自主知识产权操作系统的公司。本文是作者在该公司参加两年多研究工作的基础上完成的。

尽管现代操作系统研究已获得丰硕的成果，它仍然面临着严峻的挑战。由于操作系统处于快速变化的外部环境之中，与操作系统相关的各种因素也在不断变化，如何构建灵活的操作系统以应对各种变化，已成为操作系统研究必须解决的核心问题之一。

人们已经提出了许多解决方法，基本思路都是提高系统的可扩展性能。但由于操作系统模型的限制，这些工作大多存在一些固有的缺陷和不足。

本文在作者参加和欣操作系统(Elastos Operating System)的早期研发工作的基础上，提出了一种全新的操作系统模型——灵活内核模型，并且给出灵活内核操作系统的基本准则，探讨了实现灵活内核的基本问题。基于灵活内核模型的系统可以根据性能需要，将系统扩展装载到内核空间或用户空间，当效率是系统的首要问题时，系统扩展可以装载到内核空间，以获得最好的效率；当系统要求很高的安全性时，系统扩展可以装载到用户空间。

灵活内核模型的思想已经在后来的和欣操作系统设计中得到了体现。和欣操作系统是基于构件技术的操作系统，系统的内核对象和系统扩展都采用了 ezCOM 构件技术封装，同时“和欣”构件平台提供强大的构件运行时支持。所有这些，为和欣操作系统建立了一套非常灵活的系统扩展机制，保证系统快速应对变化。另外，和欣操作系统是一个完全拥有自主知识产权的操作系统，它已经被列为国家 863 计划重大软件项目，它的目标是设计开发相当于国际水平的面向网络的嵌入式操作系统。

本文还分析了传统设备驱动模型的缺陷，提出了构件化设备驱动模型，并且论述了该模型在和欣操作系统上的设计和实现。构件化设备驱动模型采用构件封装设备驱动，设备驱动的接口不再严格依赖于操作系统，使得驱动

程序更能够体现硬件设备的特性；采用构件化设备驱动模型，操作系统能够很容易地处理硬件设备的变化。

操作系统扩展技术为操作系统灵活适应变化提供了重要手段，这项技术的使用同时也能够降低系统复杂度，提高系统可移植性。本文在这方面进行了一定的探索，提出了新的操作系统模型，并在和欣操作系统上对这一模型进行了初步实现。但这只是一个良好的开端，今后，还需要在有关方面进行更加深入、广泛的研究。

关键词：操作系统，灵活内核模型，构件技术，ezCOM 构件技术，和欣操作系统，设备驱动模型

Abstract

For realizing the revival of China, our country must possess CPU chip and operating system about which we have all the knowledge properties. At Present Beijing Koretide Century Co. is the only one which is developing operating system about which we have all the knowledge properties. Author has worked in this corporation for about two years and this paper is written according to the researches and work which author did in this corporation.

Although plentiful fruits have been got in modern operating system theory research so far, there are still many challenges in this field. Software and hardware technology developed rapidly, and also computer system was used extensively. As a result of this situation, factors related to operating systems change continuously and how to build flexible operating systems which can be adapted to various changes has become main problem which must be solved in operating system research.

People have carried out much work, for example researches on extensible operating system. Basically, all of them aimed at promoting system extensibility. Because of restriction of operating system model which they are based on, they have some intrinsic defects.

According to my early researches and work on Elastos Operating System, I present a new operating system model, agile-kernel model, in this paper. Also I give the criterions for agile-kernel system and inquire into the common issues in implementing agile-kernel model. Systems, based on agile-kernel model, can load system extensions into kernel address space or user address space according to consideration for system property. If the system performance is the most important issue, system extensions can be loaded into kernel address space to achieve good performance. If systems emphasize security, system extensions can be loaded into user address space.

The idea if agile-kernel model has been applied to the later design of

Elastos Operating System. Elastos Operating System is component-based operating system. Its kernel objects and system extensions are encapsulated with ezCOM component technique. Meanwhile, Elastos Component Platform provides powerful runtime support for these kinds of components. With all these design and techniques, a more flexible system extension mechanism is constructed on Elastos Operating System that make system can cope with changes more quickly and easily.

In addition, I propose component-based device driver model and, also discuss the design and implementation of this model on Elastos Operating System. Component-based device driver model applies component technique to device driver. Compared with traditional device driver in Unix, the interface of device driver under new model isn't relied on operating system any more that make device driver is good at reflecting the feature of devices. Operating system can more easily deal with the changes of computer device with new model.

System extension technique provides important means for operating system handling changes. Also, using this technique can reduce complexity of system and improve system portability. Some explorations about system extension technique are made in this paper. We present a new operating system model and discuss Elastos Operating System based on new model. It is a good start, however, deeper and more extensive researches will be needed in the future.

Keyword Operating System, Agile-kernel Model, Component Technique, ezCOM Component Technique**,** Elastos Operating System, Device Driver Model

引　言

近 20 年来，我国国民经济一直呈现稳步发展的态势，国力日益强盛，各行各业都在向着技术化、产业化方向发展。为了获得进一步持续发展的动力，国家越来越重视基础性研究。在计算机领域，国产 CPU 龙芯的诞生，标志我国在计算机硬件方面有了突破性进展。然而，作为基本系统软件的操作系统，在我国的发展远远落后于国外水平。拥有自主知识产权的操作系统对于国家计算机产业发展的重要性是不言而喻的，本文所讨论的和欣操作系统正是在这方面的一个重要尝试，它已经被列为 863 计划重大软件项目，而且于 2003 年初正式发布 1.0 版。和欣操作系统以国际先进水平为目标，在技术上做出了许多重大创新，本文就是作者在深入参与和欣操作系统项目基础上完成的。

0.1 操作系统研究面临的挑战

操作系统是介于硬件和应用程序之间的软件层，负责掩盖硬件的复杂性，并提供便于应用程序使用的系统接口。作为计算机系统的重要组成部分，关于操作系统的研究一直是计算机理论研究的重要分支。操作系统研究受到诸多因素的影响，主要方面有：与操作系统相关的软硬件技术、应用程序的需求和操作系统的应用环境等。在不同时期，操作系统研究总是与这些因素在当时的状态和发展趋势紧密联系的。

在操作系统的早期发展过程中，计算机系统的使用范围有限，因此操作系统的应用环境相对比较单一和固定，与系统相关的软硬件技术发展缓慢，应用程序的需求也很少发生变化。由于影响操作系统的各种因素比较稳定，因此早期的传统操作系统在设计实现后，会有一段较长时间的稳定期，在这段时间内，系统结构和系统功能不会有太多的改变，系统在应用中配置使用以后也很少更改或重新配置。

近几年来，信息技术已逐渐成为推动社会生产力发展的重要核心技术

之一，计算机技术与传统产业相结合，将导致操作系统的应用环境向着多元化方向发展。随着与操作系统相关的软件技术的飞速发展，运行于系统上的应用程序也不断向系统提出新的需求。影响操作系统的各种因素呈现出空前的不稳定性，主要表现在以下几个方面：

- 现代计算机系统拥有的处理器越来越多、运行速度越来越快，内存和磁盘的容量越来越大，总线也越来越宽、速度越来越快，网络带宽也逐渐增大，在过去的五年里各项硬件技术的发展至少提高了一个数量级，而且这种发展速度并未见减缓的趋势。新的硬件技术推动着操作系统的重新设计，32 位处理器到 64 位处理器的转换，促使一些操作系统必须进行大量的修改。早期版本的 UNIX 操作系统只支持一种文件系统，如果要支持多种类型的文件系统，则修改是无可避免的。有时还应该设计文件系统的扩展接口，以保证新的文件系统出现时，系统无需太多的修改就能够予以支持。另外，GB 量级的网络技术已经出现，它势必要求重新评估系统底层网络的设计。硬件技术的发展也改变了系统中的性能瓶颈，比如先前的许多文件系统限制了文件或文件系统的规模(例如 4GB)，已不能适应现代磁盘驱动的容量；
- 由于现代计算机软件技术的发展，应用也向操作系统的设计提出了新的要求，这些要求的重要性一点不亚于硬件引发的需求。多媒体应用要求大容量存储、宽带访问等性能以及快速的网络访问功能；而大型数据库系统则要求能够高效访问超大规模的存储；现代信息计算越来越依赖于分布式应用，而分布式应用迫使操作系统必须支持新的计算模型，比如移动计算要求操作系统处理动态变化的操作环境和信息的移动可获取性。有些情况下，应用程序可能要求操作系统提供一些在设计阶段没能预见到的服务，甚至于有时系统服务的设计根本不符合应用程序的要求，比如实时应用要求特殊的调度策略和性能保证，然而大多数通用操作系统很少提供这方面的支持；

- 操作系统的应用环境随着计算机技术的广泛使用也发生着巨大变化。由于硬件技术的发展，以往只用于高端产品中的计算机系统，现在已经深入到低端电子产品中。从大型计算机到工业服务器，从台式计算机到基于嵌入式系统的智能电子产品，到处都有操作系统的身影。这些应用环境之间存在很大差异，因此对运行于其上的操作系统要求也各异。即使在嵌入式环境中，不同产品之间的硬件设备也不尽相同，比如：有的嵌入式系统中有外存，有的则没有，而且各个系统的内存容量差异较大。不同的环境通常意味着不同的要求，有的要求不同的服务，有的对服务的性能有特殊要求。

传统操作系统根本不能适应这样多变的环境，实际上，传统操作系统在设计时就很少考虑适应变化的外部因素。当系统不能适应某些要求时，通常的做法是修改部分代码，甚至更改原有的设计，虽然这样比起重新设计和实现一个操作系统的确容易许多，但这种做法依然耗时耗力，而且再遇变化时系统可能还需修改，这种修修补补的做法显然不是解决问题的根本途径。

综上所述，现代操作系统处于快速变化的环境中，诸多变化向操作系统研究提出了前所未有的挑战，然而，挑战预示着发展的契机，预示着广阔的前景。

0.2 解决途径

由于现代操作系统处于快速变化的环境中，如何适应各种变化就成为必须正视和解决的问题，基于传统操作系统的修修补补的办法并非解决之道。随着时间的推移，这个问题越来越突出，操作系统研究者也逐渐意识到问题的重要性，许多研究开始着力于解决这个问题。

微内核操作系统的研究在这方面迈出了一大步，它提出将一些系统服务以服务器的形式置于用户空间，应用程序与服务器之间的交互通过进程间通信机制来实现，用户空间的服务器可以根据需要进行替换、增加和删除，从而大大提高了系统的灵活性和应变能力。然而，与整体内核操作系

统相比，微内核系统由于大量使用进程间通信，将导致系统的整体效率较低。有些研究又基于整体内核系统，提出了可装载内核模块技术，将一些系统功能封装成独立于内核的二进制映像文件；而在系统启动时或运行中，再动态装载这些系统功能模块到内核中，并向外提供相应的服务。这种技术，在一定程度上提高了系统的可扩展能力，但是装载到内核的模块会损害系统的安全性，而且扩展模块的接口也受到系统的限制。

关于微内核和整体内核两种操作系统模型的争论很多，各自的支持者想方设法弥补自身系统存在的缺陷，然而有一点很明确，它们都希望实现更加有效的系统扩展机制。认识到提高系统的可扩展性才是最本质的需求以后，一些研究者开始研究基于具体技术的可扩展操作系统。这方面的研究内容相当丰富，已经提出许多具体的扩展技术。但是，归纳各种研究工作我们发现，所有扩展技术仍是基于微内核系统或者整体内核系统。它们只是做了一些局部改良，并没有突破整体内核系统和微内核系统的限制。因此，它们也继承了系统所固有的缺陷，也就是，微内核系统的效率低；整体内核系统随着扩展模块的引入，系统安全性有所下降。

整体内核系统和微内核系统的固有缺陷是操作系统模型本质的体现，要想避免这些缺陷，必须从系统模型入手，改造现有的操作系统模型，或者建立新的操作系统模型。

本文在对相关研究工作进行深入分析的基础上，提出了一种全新的操作系统模型——灵活内核模型，并且给出了灵活内核系统的基本准则。灵活内核系统可以通过系统扩展对系统功能进行扩展。不同于微内核系统和整体内核系统，灵活内核系统的系统扩展可以根据需要装载到内核空间或用户空间运行，系统扩展的运行位置可以动态配置。灵活内核模型为系统带来了更大的灵活性，系统可以在不同的性能之间进行选择，比如：当安全性是系统首要关心的问题时，系统扩展可以配置在用户空间运行，以提高系统安全性和稳定性；如果系统希望获得更高的效率，并且系统扩展经过长期考验已证明其非常稳定，这时，可以让系统扩展运行于内核空间。

作为灵活内核模型的一个实践，本文还详细论述了和欣操作系统基于

灵活内核模型的设计与实现。和欣操作系统从设计目标来看，它是面向Web服务的嵌入式操作系统；从系统模型来看，它是灵活内核操作系统；从系统设计和实现的手段来看，它是构件化操作系统。和欣操作系统是北京科泰世纪有限公司的一个操作系统项目，英文名称为Elastos。和欣操作系统于 2003 年被列为国家 863 重大软件专项*，并于 2003 年元月正式发布 1.0 版。作者在攻读博士学位期间，作为主要研发人员有幸参与了和欣操作系统的设计与实现。

和欣操作系统采用的构件技术是 ezCOM 构件技术。ezCOM 构件技术是北京科泰世纪有限公司的一项重要技术，作者的一部分工作就是设计和实现和欣操作系统上的 ezCOM 构件基础设施。ezCOM 技术与微软 COM 技术兼容，并且根据需要对 COM 技术进行了扩展，以适应开发系统软件的需要。ezCOM 技术的扩展包括：CDL 语言、自描述数据类型、智能指针和自描述信息的封装。这些技术上的扩展有两方面的作用，首先，它简化了构件编程，能够有效防止构件编程容易出现的错误；其次，ezCOM 构件自描述信息的封装是和欣操作系统构件运行机制的基础。

在和欣操作系统中，使用 ezCOM 构件封装的系统扩展，又称为系统扩展构件，系统扩展构件可以在内核空间和用户空间运行。从开发和使用的角度来看，系统扩展构件可以很方便地对系统功能进行扩展。而且用户能够通过配置系统扩展构件的运行位置，在不同的系统性能之间进行选择，以往的可扩展操作系统不可能做到这一点。和欣操作系统的 ezCOM 基础设施——“和欣”构件平台，在构件运行时，充分使用构件的自描述信息，为系统扩展构件的运行提供了强大的支持，保障了系统扩展构件能够正确有效地扩展系统功能。与 Windows 平台的构件运行机制相比，和欣构件平台提供的构件运行机制更加稳定，更能够适应构件运行的需要。

和欣操作系统的内核功能也采用构件技术封装，系统功能以内核构件接口的形式向外界提供。这种做法保证了系统接口和扩展接口形式上的一

* 课题编号：2001AA113400，课题名称：基于中间件技术的因特网嵌入式操作系统及跨操作系统中间件运行平台。作者的工作和研究受到该课题的资助。

致性，降低了系统复杂性，便于系统用户使用。其它可扩展操作系统的系统调用和系统扩展采用两套独立的设施予以支持，增加了系统设计和实现的难度，而且不易于维护。

和欣操作系统提高了系统的可扩展能力，同时，它提供的构件运行机制使其具有了网络时代操作系统的先进特征。网络时代，软件的搭建和运行都与网络联系越来越紧密，传统软件的组装和运行方式逐渐不能适应发展。和欣操作系统的构件平台能够通过网络定位所需的构件，将构件下载到本地，把不同功能的构件动态组合成应用，然后在本机运行。这种软件动态组合方式是在严格的版本控制下进行，不会因为新旧版本软件之间的冲突导致软件运行异常。显然，这不同于传统系统的软件先安装后运行的做法，实际上，传统做法的弊端很多，从 Windows 系列操作系统就可见一斑。

另外，本文还提出了构件化设备驱动模型，并论述了该模型在和欣操作系统上的设计与实现。构件化设备驱动模型采用构件封装设备驱动，设备驱动的接口不再严格依赖于操作系统，使得驱动程序更能够体现硬件设备的特性。和欣操作系统内核模块设备管理器负责对设备驱动进行管理，它利用“和欣”系统的构件基础设施管理设备驱动对象，如创建和销毁设备对象。采用构件化设备驱动模型，操作系统能够很容易地处理硬件设备的变化。

0.3 论文的工作背景和创新点

作者于 2000 年 10 月到北京科泰世纪有限公司参与和欣操作系统项目，在该项目上连续工作了近两年半时间，有幸参与了和欣操作系统早期设计和其后的一些相关工作。到作者离开这个项目时，和欣操作系统已经正式发布 1.0 版。在和欣操作系统项目工作期间，作者主要涉入的工作有和欣操作系统 ezCOM 基础设施的设计与实现、和欣构件化设备驱动模型的设计与实现、和欣构件平台的设计与实现，以及 ezCOM 技术研发，本文就是作者在深入参与和欣操作系统项目的基础上完成的。

本文主要围绕和欣操作系统进行论述。和欣操作系统是群体智慧和团

队合作的结晶，作者的工作只是其中的一部分，这里将作者的比较独立的工作亦即本文的创新点总结如下：

- 提出了灵活内核模型，给出了灵活内核系统的基本准则，而且探讨了灵活内核系统设计的一般性问题。应当指出，灵活内核模型是作者对和欣操作系统设计进行理论提升的结果，它是一般意义上的模型。在提出以后，它不再与具体的实现相关联，但在理论上对具体系统的实现有一定的指导意义。
- 作为对 COM 技术的扩展，在 ezCOM 构件技术中，引入了自描述信息的封装技术。ezCOM 封装的自描述信息包括类信息和导入信息。类信息描述构件中的所有类；导入信息描述构件之间的依赖关系。通过自描述信息的封装，可以实现构件的无注册运行，并可以支持构件的动态升级和自滚动运行。
- 和欣操作系统使用系统扩展构件对系统功能进行扩展，需要将内核空间的构件接口取回到用户空间，这个过程称为接口远程化。作为创新，本文提出了利用构件自描述信息完成接口远程化的一套方法。在接口远程化过程中，“和欣”构件平台会根据自描述信息自动生成代理构件，也就是所谓的中间层。接口远程化能够有效掩盖远程接口和本地接口之间的差异，从而实现了 ezCOM 构件的位置透明性，便于用户使用。
- 针对远程构件接口的使用，本文在“和欣”构件平台上实现了构件接口参数的自动化散列集。参数的自动化散列集是指将调用参数和返回值在不同的地址空间传递的过程。“和欣”构件平台实现的参数自动化散列集是利用构件的自描述信息完成的，它使得构件运行无需额外设施的支持，如注册表等。有效提高了构件运行的效率和稳定性。
- 提出了构件化设备驱动模型，作为该模型的特色，“和欣”系统采用 ezCOM 构件封装设备驱动。与 Unix 系统的驱动模型相比，由于设备驱动构件的接口无需依赖于系统规定，“和欣”的构件化设

备驱动模型更能体现不同设备的各自特性。

- 由于设备驱动采用构件封装，“和欣”系统的设备管理不同于传统操作系统。作为本文的另一创新，文中详细论述了“和欣”系统设备管理器的设计和实现，设备管理器负责对设备驱动进行管理，它利用“和欣”系统的构件基础设施管理设备驱动对象，如创建和销毁设备对象。通过使用设备管理器，操作系统能够很容易地处理硬件设备的变化。

0.4 论文的结构

论文的主旨是介绍基于灵活内核的“和欣”操作系统，各章的内容或直接或间接服务于这个主题，其中第一章至第七章是论文的主体部分，下面分别介绍各章的内容。

第一章回顾了现代操作系统的部分研究现状。操作系统作为计算机系统的主要组成部分，这方面一直不乏大量的研究工作，要讨论所有研究工作几乎是不可能的，而且也没有必要。第一章重点讨论了操作系统模型以及微内核和整体内核的系统模型之争。另外，可扩展操作系统在操作系统研究领域提出了许多有效的问题解决途径，第一章也涉及了可扩展操作系统的研究工作。

第二章论述软件构件技术的相关研究。和欣操作系统在实现灵活内核模型过程中使用了构件技术，而且和欣操作系统的驱动程序模型也采用了构件技术，因此论文的第二章首先介绍了构件技术的由来和发展过程、构件技术中的相关概念和当今流行的构件体系结构。然后，详细阐述了 ezCOM 构件技术。和欣操作系统在具体实现过程中所采用的就是 ezCOM 构件技术，针对 ezCOM 的讨论内容涉及 ezCOM 技术的基本思想、技术特点和实际意义。

第三章主要论述和欣操作系统灵活内核的设计和实现。依次讨论了灵活内核模型、和欣操作系统灵活内核设计特点以及“和欣”内核模型、“和欣”构件平台、和欣操作系统灵活内核的技术优势等。

和欣操作系统可以通过系统扩展构件进行系统功能扩展，第四章阐明什么是系统扩展构件，并且从动态运行的角度讨论了系统扩展构件。着重阐述了 ezCOM 构件自描述信息封装、系统扩展构件的运行方式以及“和欣”构件平台提供的构件运行时支持。

第五章提出了基于和欣操作系统的构件化设备驱动模型。详细论述了“和欣”设备驱动模型的设计实现，对构件化设备驱动模型和传统 UNIX 设备驱动模型进行了深入比较。

第六章主要从理论和测试数据两个角度分析了和欣操作系统的性能，从测试数据来看，和欣操作系统在一些传统的性能指标方面和先进的嵌入式操作系统没有太大差距，但在架构方面和欣操作系统具有较强的优势。另外，面向嵌入式系统是和欣操作系统的一个重要射击目标，第六章还分析了和欣系统在这方面的一些考虑和优化。

第一章　操作系统研究背景

和欣操作系统和灵活内核模型的研究都是在对操作系统相关研究进行充分认识的基础上展开的。在论述和欣操作系统和灵活内核模型之前，有必要回顾相关的研究背景。

1.1 操作系统模型

一般而言，操作系统提供两种功能[Tane01]：作为资源管理者，有系统地在互相竞争的进程之间分配计算机资源；作为计算机的扩展，为用户提供功能强大的开发环境和应用环境。现代操作系统仍然主要实现这两种基本功能，但是它们的内涵与早期操作系统相比有了极大的扩展，操作系统复杂性的提高就明显说明了这一点。由于系统复杂性与日俱增，系统的体系结构对系统性能和系统功能的影响也越来越突出。

系统的体系结构是由构建系统时所采用的操作系统模型决定的。操作系统模型提供的是一种框架，它是大量系统特征的总和，该框架明确界定了操作系统提供哪些服务和执行哪些任务以及通过何种方式实现这些服务和任务。现代操作系统按照模型划可以分为以下三类：

- 整体内核(Monolithic kernel or Macro-kernel)操作系统
- 层次式(Layered)操作系统
- 微内核(Micro-kernel)操作系统

1.1.1 整体内核操作系统

整体内核操作系统有如下特征：操作系统按功能定义模块，系统服务和驱动程序都在内核空间，分别定义成不同的功能模块；任何模块之间可以遵循特定的接口规范互相调用；所有模块链接在一起，形成一个可执行文件，使用时整个文件都应装载到计算机的内存中；系统模块都运行于超

级模式(supervisor mode)或者内核模式(kernel mode)下，可直接取用计算机的内核资源，如硬件等；应用程序若要取用内核资源，它需要执行系统调用，请求一个或多个系统模块协作，帮它获取并访问和使用相应的资源。图 1.1 展示了整体内核操作系统的组织形式。

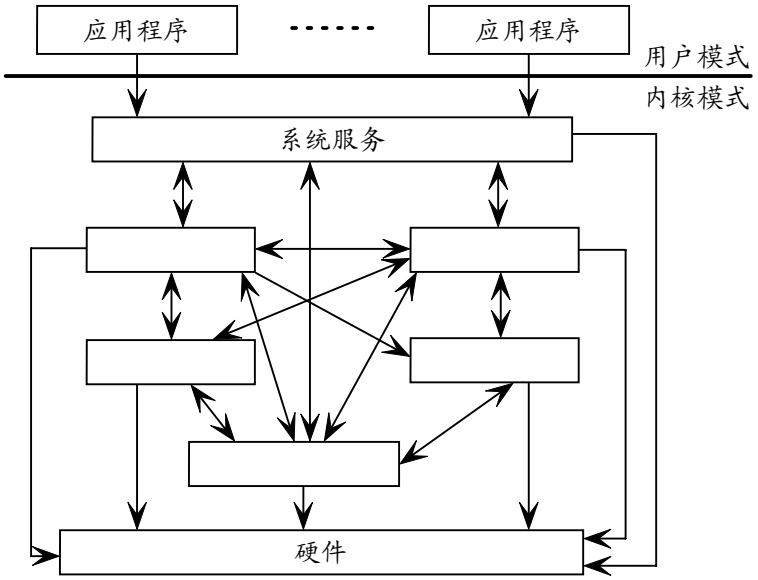


图1.1 整体内核操作系统

整体内核系统的系统调用工作方式通常是：参数先装入预定的寄存器或堆栈中，接着执行一条特殊的中断指令，切入到超级模式或内核模式，然后进入操作系统的某个模块并执行相应操作，执行完后返回到用户模式，最后通过参数和返回值向应用程序报告执行情况。

典型的整体内核操作系统有 MS DOS，Windows95 等。

1.1.2 层次式操作系统

层次式操作系统在整体内核操作系统的基础上做了一些改进工作，系统内核按照功能分解成为相对独立的模块，并且把这些模块以层次形式组织成整个系统。按层次摆放的系统模块都提供一些功能接口，其它模块可以通过使用这些接口调用该模块实现的功能，但是所有系统模块通常只能使用它下层的模块提供的服务，图 1.2 展示了层次式操作系统的一种基本形

式。

层次式操作系统最突出的优点在于降低了系统模块之间的耦和程度，某一层模块实现的改变只会直接影响它上一层的模块，一般不会出现因系统局部发生变化而造成整个系统大规模修改的情况，相对而言，整体内核系统由于系统内部的联系错综复杂，系统的某一模块发生变化比较容易会导致“多米诺”骨牌似的连锁效应。

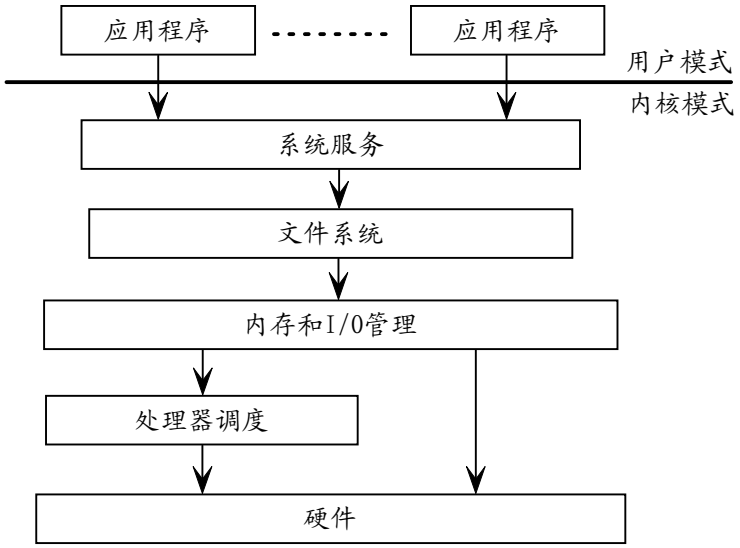


图1.2 层次式操作系统

在某种意义上，层次式操作系统仅仅是在设计阶段对整体内核系统进行了改良，改变了系统模块的布局，使其更有条理结构更清晰，从而降低系统开发和维护的难度。从用户使用的角度来说，它仍然和整体内核系统一样，没有带来任何有新意的特点，有很多的研究者直接将其归入到整体内核中，本文也采用这种观点，因此下面一节讨论体系结构之争时，也不会再单独讨论层次式系统。Linux[**Bove01**]、Multics[**Corb65**]和 UNIX[**Vaha96**]都是此类型操作系统的代表。

1.1.3 微内核操作系统

微内核操作系统的内核是由一些最基础的抽象系统模块构成，许多传

统的系统服务甚至包括驱动程序都以服务器的形式被放在了内核之外，核内一般只包括进程管理、I/O 处理、内存管理、进程间通信等基本功能，因此微内核操作系统又被称为客户/服务器式操作系统。

MACH 3.0[**Blac92**] [**Acce86**]是非常典型的微内核系统，MACH 3.0 的内核包括的抽象功能模块有任务管理、线程管理、内存对象管理、消息和端口管理。采用微内核模型使得 MACH 3.0 拥有可裁剪性、可扩展性和可移植性等良好特性，而这些特性是整体内核操作系统很难具备的。除了 MACH 3.0 之外，微内核操作系统还有 MINIX[**Tane97**]、CHORUS [**Bric91**]、AMORBA[**Tane94**]等。

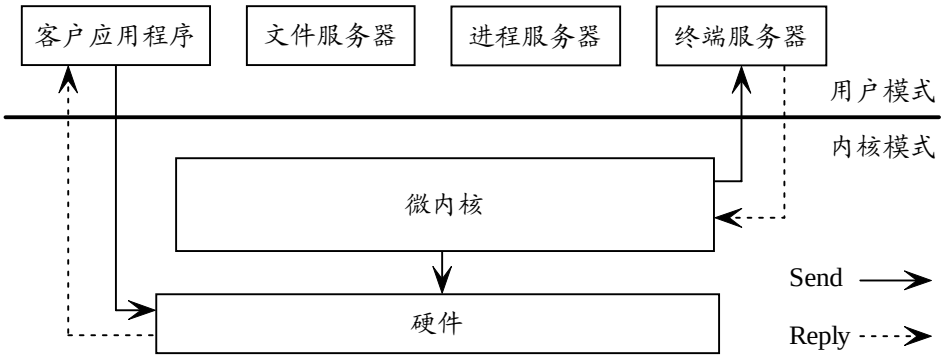


图1.3 微内核操作系统

如图 1.3 所示微内核操作系统内核的主要工作是处理客户端与服务器端之间的通信，操作系统被分成了多个部分，除了内核之外，每个部分仅仅处理某一方面的功能，如文件服务、进程服务、终端服务和存储器服务等。

微内核模型把一些传统的系统服务以服务器的形式置于核外，系统服务能够以比较独立的形式开发和部署，这极大地提高了操作系统的可扩展性，可以说微内核模型在提高系统的扩展性方面做出了相当有意义的工作。

1.2 微内核和整体内核之争

1.2.1 微内核和整体内核比较

在上个世纪 80 年代，微内核思想被引入到操作系统的研究中，它所具备的灵活性、安全性和模块化特征似乎预示着传统的整体内核模型将被新的模型所替代。微内核的“光明前景”使得一时之间关于微内核的研究成为操作系统研究的热点。

然而，在微内核出现后的 20 年里，微内核和整体内核两种模型之间的争论一直就没有中断过。其间曾经在 Tanenbaum 和 Linus Torvalds 之间发生过关于“linux 已经过时”的论战[**Dibo99**]，论战的核心就是微内核和整体内核，争论的结果是双方都没能说服对方。

下面对微内核系统和整体内核系统进行一些比较。在比较的过程中，微内核系统将作为主体，比较从两个方面进行，分别展示微内核系统相对于整体内核系统的优点和劣势。首先展示微内核系统的优点：

- **安全性**　微内核系统将许多系统服务以服务器的形式置于用户空间，这些服务器运行在与内核和应用程序完全独立的地址空间中，当服务程序运行出错时，一般不会导致内核的崩溃。微内核系统的许多模块独立运行于核外，系统能够以模块为单位把安全问题进行分解，使得系统服务程序严格按照安全要求运行。
- **可配置性**　一般微内核系统中的服务程序可以在整个系统不重新启动的情况下被更换，这一点对于整体内核的操作系统而言是难以实现的。
- **易于编程**　操作系统内核中的代码通常需要使用特殊的内存分配和输入输出等例行程序，用户空间的代码比内核代码容易编写，它无须去考虑内核特定的一些限制。在微内核系统中，运行于用户空间的服务程序在编写时不受内核的约束。
- **降低内存的固定使用量**　操作系统内核的代码和数据通常是常驻

在内存中，不允许被交换出内存。微内核系统把部分系统功能移到用户空间，系统服务程序只有在被用到时才会装入内存，这样就减少了系统的固定内存使用量。

- **可扩展性(extensibility)** 在微内核系统中添加系统模块就像编写用户程序一样简单，只要它遵从系统提供的设计接口；而整体内核系统添加模块首先需要你对操作系统内部工作机制有一定的了解，在编写好模块程序之后，还需要重新编译内核，以便把添加的模块连接到内核中去。

上面提到的都是微内核系统的进步之处，当然微内核系统也并非完美无缺，微内核系统和整体内核系统相比也有不足之处：

- **微内核规模并不小** 大多数微内核系统的内核规模并不小，尽管“微内核”的名字让人产生一种错觉。如果内核做得很大，相应系统的内存使用量就会有所增长，而且内核代码过多，也不利于系统代码的维护。
- **效率的缺失** 将许多系统的服务置于核外，就需要给这些服务程序之间提供相应形式化的消息传递机制。在整体内核操作系统中，不同的系统服务模块总是通过系统内存来互相传递信息，模块之间的调用不会跨越地址空间。在微内核系统中，用户到服务器和服务器之间的通信都要跨越地址空间，系统的性能会因为这些通信的使用而降低。
- **出现的新问题** 系统的部件之间会出现一些新类型的死锁或其他的条件错误，而这些在整体内核的系统中是不会出现的。

虽然微内核系统拥有良好的系统特性，但是它的固有缺点严重地影响了系统的可用性。与系统研究者对于微内核的热衷相比，微内核系统却迟迟未能得到工业界的认可，[**Lied96**]中也承认微内核系统并未取得成功，微软的 Windows NT 4.0[**Cust93**]就曾经采用过微内核模型，效果并不理想，结果 NT 5.0 又回到了传统的整体内核模型。

1.2.2 争论的意义

虽然微内核没有取得大范围的成功，但是微内核和整体内核的争论的确引发了大量新技术和新方法的出现。

为了提高系统的可扩展性，许多传统的整体内核操作系统纷纷开始考虑支持动态加载内核模块，比如作为 UNIX 变种的 Solaris[**Maur01**]及现在极为流行的 Linux[**Bove01**]都支持动态内核模块装载的机制，以获得系统灵活性，降低内核的复杂性。然而，这样的动态加载模块仍然不能很好地解决系统安全问题，有些研究工作提出了一些新技术试图提高加载模块的安全性，在下一节我们会看到一些类似的工作。

微内核系统的致命弱点是系统效率损失极其严重，因此微内核模型支持者的研究工作开始注重提高系统效率。[**Lied93**][**Lied97**]中曾经尝试提高进程间通信的性能，以便提高整个微内核系统的效率，但并不理想。另外，一些微内核系统的研究试图改变早期微内核系统的系统抽象观点，这类系统的代表有 Exokernel 和 L4，[**Lied96**]把它们总结为第二代微内核系统。Exokernel[**Engl95**]认为灵活性和系统抽象是导致效率下降的根本原因，它放弃构建通用操作系统的思想，转而坚持硬件平台相关的设计原则，效率较之 MACH 有明显提高，但它依赖于 Mips 体系；L4[**Lied95**]的微内核设计也与处理器相关，甚至于在 intel 486 和 Pentium 处理器上的实现都不一样。显然第二代微内核的开发成本是非常高的，应用范围也有限，而且存在难于移植的弊端。

另外，许多研究者借鉴了微内核系统和整体内核系统的研究经验，摒弃了系统模型之争，转而研究基于具体扩展方法的可扩展操作系统，下一节将介绍这方面的工作。

1.3 可扩展操作系统的研究

1.3.1 动机

微内核和整体内核之争的实质，简单而言就是效率和灵活性之争。理

想的解决途径是建立一种高效的系统扩展机制。效率是一切系统为了投入实用而必须考虑的；灵活性是系统需求多样化的必然要求。但是它们两者之间并非总是并行不悖的，多数情况下，它们是此消彼长的关系。

许多操作系统的研究者认清了微内核和整体内核模型之争的本质，放弃了基于系统体系模型的讨论，开始回到技术方法的研究上，展开了大量的可扩展操作系统的研究。如果从可扩展操作系统的视角观察微内核技术，它只不过是一种特殊的系统扩展技术。

可扩展操作系统的目标是为用户提供动态配置系统功能的机会，用户可以在不重新编译系统的情况下，动态地添加系统功能，从而使系统满足特定的需求，当然这种扩展技术应该能够满足一定的效率要求。

另外，可扩展操作系统研究还有一个更重要的出发点，也就是试图去解决引言中提出的操作系统研究面临的问题。

1.3.2 研究现状

在过去几年里，可扩展操作系统逐渐成为操作系统研究的重点之一，各种操作系统扩展技术不断涌现，其中前面提到的微内核技术是一个重要的分支。

由于微内核模型中频繁使用进程间通信，严重降低了微内核的效率。基于这样的认识，[**Guil91**]又直接将服务器的代码链接到内核空间，实现了内核服务器，但是系统内核的安全性因为这些代码的引入而大大降低。[**Wahb93**]沿用了[**Guil91**]的模型，同样把服务器放在内核中，但是它使用了 Sandbox 的软件出错隔离技术，这种机制比硬件保护机制的安全性要低一些，可以提供一定的安全保障。[**Cao94**]根据可扩展系统主要是提供给用户程序选择系统策略的理解，提出了把多重策略编译到内核中去，用户程序可以动态选择使用其中的某种策略，但是用户不能向内核添加新的策略。这显然违背了动态扩展的原则，而且事先预见所有可能出现的策略是根本不可能的。HiPEC 系统[**Lee94**]有些类似于[**Cao94**]所提出的思想，但 HiPEC 允许用户使用特定的解释型语言编写控制虚拟内存缓冲的策略，能够实现

动态改变控制策略，从效率上看也不会有太大损失，但这种语言的表达能力有限，很难适应其它应用。Spin[**Bers95**]系统则使用一种“安全”语言Module-3[**Nels91**]编写而成，而且它的系统扩展也使用这种语言编写。Module-3 是强类型语言，具有垃圾回收功能，而且这种语言能够避免指向已经删除数据的悬摆指针，也不能使用指向任意内存地址的指针，用这种语言编写的“安全”模块执行时不会超出模块范围。

[**Smal96**]通过试验分析和比较得出结论：使用编译技术建立通用的内核扩展是上佳选择(比如 Module-3)。其实质有两点：a. 使用“安全” 的语言；b. 系统支持装载到内核的扩展。这样的做法完全放弃了微内核的一些优秀思想，这种“安全性”的保障显然不比微内核系统更有效，而且这种模型也不适合分布式计算，只能用于实现系统扩展技术。而微内核系统的系统服务完全符合分布式计算模型，因此系统扩展和分布式计算可以采用同样的处理方式。

总结分析各种文献中的研究工作，关于可扩展操作系统的研究主要采用以下两种扩展方式：

- 允许用户开发的模块动态地加载到操作系统内核中，这些模块会导出用户可以使用的接口。这种方式依赖于具有安全特征的语言，即能够在编译期间进行安全方面的分析，并出于安全的考虑进行动态引用检查。使用这种方法的典型例子有 SPIN[**Pu88**]和 VINO，这种方法实际上采用软件技术实现安全保护；
- 提供信任路径(trusted path)的机制，如受保护的过程调用和进程间通信，扩展模块作为用户任务执行，使用标准的系统保护机制处理安全问题，客户端通过信任路径调用扩展模块。例如 MACH 和 Amoeba 采用的 client/server 扩展模型使用 IPC 作为信任路径。这种方式采用的是硬件技术实现安全保护。

但是这两种方法都存在着一些缺陷：第一种方法仅仅是在传统操作系统的基础上提高系统的模块化特征，提供模块接口标准，实现动态模块加载，系统的稳定性却会因为加载模块而降低，而且接口标准仍然限制了扩

展技术的灵活性；第二种方法就是通常所说的微内核技术，它的弊病是由于大量使用类似进程间通信的机制，频繁地在系统模式和用户模式之间切换，使得系统的效率大大降低。

另外，从深层次分析，这两种系统扩展方式并没有摆脱系统模型的阴影，前一种方式明显是基于整体内核模型，而后者是按照微内核模型的思路发展的，可以看出它们的缺陷也明显带着这些模型的痕迹。可扩展操作系统研究并没有突破操作系统模型的限制，这些模型具有的缺陷仍然存在。我们认为，这些缺陷是系统模型的本质造成的，如果要避免这些问题，那么只有从模型的角度出发，改造现有的模型，建立新的系统模型，建立更加灵活的系统模型。本文提出了一种全新的操作系统模型，新的模型会在一定程度上提高系统的扩展能力。

1.4 小结

本章介绍了与论文相关的操作系统研究背景。由于操作系统模型是影响操作系统特性的重要内部因素，因此讨论首先从系统模型入手。现有的操作系统基本采用整体内核模型、层次式模型和微内核模型之一，层次式模型一般被认为是整体内核模型的一种特例。基于这种观点，文中只对微内核模型和整体内核模型进行了深入比较，阐明了各自优缺点以及发展现状。不同于操作系统模型方面的研究，一些研究者提出了各种操作系统扩展技术来解决操作系统面临的问题。从实质上看，各种操作系统扩展技术的研究并没有突破现有的操作系统模型，在特性上，深受一些原有操作系统模型的影响，因此存在一些缺陷和不足。

第二章　软件构件技术

软件构件技术是面向对象技术的自然延伸，它已经成为搭建计算机应用软件的重要技术之一。一般构件技术都是作为面向应用的技术来使用，和欣操作系统则将构件技术引入到系统软件的设计和实现过程中，使得系统各部分具有高度一致的构件模型。为了更好地理解和欣操作系统，本章先概要介绍构件技术。和欣操作系统不同于传统的操作系统，其主要特点是将作者参与研究的 ezCOM 构件技术引入了这种操作系统，本章将重点详细地介绍 ezCOM 技术。

2.1 构件技术的产生

构件技术产生的原始动力是软件领域的复用需求。在软件技术刚开始发展时，工业发展已经相当繁荣，其中一个重要原因在于机械零件的标准化，它使得工业生产摆脱了手工作坊方式，实现了大规模机械生产。在软件发展到一定时期，人们希望将工业生产的经验引入到软件生产的过程中，从而引发了对于软件构件的思考。

早在 1968 年，McIlroy[**McIl68**]就提出了系统化软件复用的思想，其基本概念相当简单：开发规模合适的构件系统，并复用这些构件。但是由于当时软硬件发展水平还相当低，在理论上和应用中很难支持这种复杂的软件体系结构，另外，在软件生产中，软件的复杂度并不象现在这么高，软件复用需求并不广泛，因而这个思想并未形成大范围的研究。

在 80 年代末 90 年代初，软件复杂度有了大幅度提高，传统手工作坊式的软件生产导致了前所未有的软件危机，构件技术作为软件工程中的一个重要分支又重新提出并发展起来。在这个时期，构件技术的出发点仍是实现有效的软件复用，但同时研究者又赋予其他的历史重任，利用它在软件体系结构层面上建立通用构件模型。

传统的软件缺乏灵活性，软件在编译期就决定了各种功能；在运行期

和维护过程中都不能灵活地改变软件的固有功能。而构件可以作为功能的扩展，在构件模型中规定了如何生成扩展，在一些情况中，框架本身可能是由运行中的应用组成的，而这些应用是由扩展构件组建的，构件模型和框架要确保扩展之间不会发生意外的交互。只有这样，扩展构件才能独立地开发和部署。

构件模型需要确定标准，从而保证独立开发的构件能够部署到一个通用的环境中，而且保证不会发生预料之外的交互，例如资源竞争。框架中的支持服务集成也会简化构件的构造，而且提供一个平台，在这个平台上可以为了特殊应用设计一系列的构件。

出于缩短软件开发周期和保证软件质量的考虑，构件技术也是非常好的一种策略。精心设计的构件库可以有效地减少系统的设计、开发所需的时间，这就是软件工厂的概念。在组建新的软件时，可以选择一些已有的构件，现成的构件一般通过精心的测试和实践的验证，要比重新写一个构件可靠得多，同时也会减少软件开发后期的测试维护工作。

另外，在构件模型和框架设计中，还可以兼顾许多质量属性方面的问题，一些属性对于特殊应用领域是至关重要的。构件模型所表达的设计原则能够在所有构件中一致地体现出来，这里的“一致”，意味着各种各样的全局特性可以纳入到构件模型中，使得剪裁性、安全性等特性可以从构件模型总体上进行考虑和设计。

基于上述的潜在动机，构件技术迅速发展起来，成为软件发展的一项重要技术。从 90 年代早期开始，基于构件的软件工程不断发展，到现在为止，一些工业化构件技术已经广泛地使用，这些技术包括微软的 ActiveX/DCOM[**Micr95**][**Micr98**]，OMG 组织的 CORBA/ORB[**Obje01**]和 SUN 公司的 Javabean/RMI [**Koza98**]等等。构件技术已经成为构建大规模软件的首选技术[**Brow00**]。

2.2 构件技术的概念

构件技术是一种软件组建技术，它所提供的是一种概念模型。它和面

向对象技术有一定的相似之处，但它们之间有着巨大差别，关于这一点后面会专门论述。既然构件技术是软件组建技术，因此应该在软件组建的范围里阐述构件技术的相关概念。

2.2.1 什么是构件？

关于构件的定义有各种各样的观点，并没有一个统一的说法，首先介绍一些研究者所提出的定义：

- Jed Harris[**Orfa96**]：一个构件是一个软件块，一方面，它的规模足够小，适于创建和维护；另一方面，它的规模又足够大，适于部署和支持，而且它具有标准接口，能够实现互操作；
- 1994 年 Meta Group 的白皮书[**Szyp98**]： 软件构件是预先构造的、经过测试的自包含可复用的软件模块，这类软件模块中封装了实现特定功能的数据和函数过程；
- Clemens Szypershi[**Szyp98**]：一个软件构件是一些接口的组合体，这些接口由一定的契约所约束，并且有明确的上下文依赖关系。软件构件可以独立部署，能够被第三方用于软件合成。

这些只是众多概念中具有代表性的几个。总结以上观点，我们给出三条准则，符合这些准则的软件模块就称其为构件：

封装　功能用黑箱的方式实现；

使用　可以用于第三方合成；

标准　符合某种构件技术模型。

第一条准则基于两个原因。首先，虽然 Linux 和开放源代码取得了成功，但在市场中占有优势的成功软件构件商业模型还是建立在知识产权的基础上，构件实现中所涉及的知识是受到保护的，而且短期内这种趋势也不会改变；其次，在计算机科学中存在这样的共识，软件构件的客户端程序不应该依赖于可能发生改变的构件实现细节。

第二条准则比较容易理解。构件的使用不应该依赖于只有构件提供者

才拥有的工具或者关于构件的知识（应该作为构件的说明文档展示给用户）。这条准则意味着由构件搭建的系统中所包含的构件可能来源各异，而整个系统可能由并不提供构件的第三方合成。即使在系统中使用的构件都不是由外部提供，这条准则依然应保证成立。

最后一条准则是构件必须符合某一个构件模型，构件模型规定了构件之间的互操作方式，它表达的是一种全局的或体系结构的设计规范，构件模型完成了从软件实现到体系结构实现的转换，所有的构件系统本身都是基于统一的构件标准建立的。

这三条准则完全是在软件组建的过程中提出的，也就是符合准则的构件才能够满足开放的软件组建模式。

2.2.2 接口

接口是软件构件向外提供的服务的具体描述，是构件客户端与构件实例之间的联系纽带，是一些操作方法的集合，客户端可以通过调用这些方法实现对构件实例的操作。一个构件可以有多个接口，不同的构件可以具有相同形式的接口。

接口是客户端访问构件的唯一方式，客户端不能直接访问构件的内部数据。而且客户端对构件的使用只能依赖于接口的定义，不能对接口方法的实现有任何假设，也就是接口形式应该和接口实现完全分离，只有这样才可能保证基于相同定义的构件能够正常地相互替换。

一个接口定义包括该接口中包含了哪些方法、每个方法的参数个数及类型、返回值类型、先决条件和后决条件等。一些中立的接口描述语言已经出现并投入使用，比如：OMG 组织的接口定义语言 IDL(Interface Description Language)[Obje01]和微软的 MIDL 接口语言，由接口描述语言定义的构件接口可以使用工具映射到不同的具体编程语言。

2.2.3 构件模型和构件框架

理解构件框架的最好途径是把它看作一个小型操作系统，在这个对比

中，构件对于框架的作用就相当于进程对于操作系统。构件框架管理构件共享的资源，并且提供构件之间交互和通信的底层机制。就象操作系统，构件框架是动态概念，它作用于构件之上，从而实现对构件生命周期和其它资源的管理，如开始、悬挂、恢复或终止构件的执行。但是，通用的操作系统如 UNIX 支持一系列丰富的交互机制，而构件框架特定支持有限类型的构件，以及这些类型之间的交互。虽然构件框架相对于操作系统丧失了一些灵活性，但它大大提高了构件组合能力。

构件框架都会支持一些构件模型，构件模型是一套支持构件的服务，还包括构件利用这些服务而必须遵守的一套规则。构件模型需要解决诸如一个构件怎样使其它构件可以获得它的服务、构件怎样被命名以及怎样在运行时发现新的构件和它们的服务这样的问题。此外，那些企业级的构件模型还提供额外的服务，其中包括事务管理、持久化和安全性等。由于这些服务的通用性，它们逐渐成为许多构件模型的基本服务。

2.3 构件技术和面向对象技术

在讨论构件技术的时候，关于构件技术和面向对象技术之间的关系有很多争论。考察面向对象技术和构件技术之间的关系，有助于更好地理解构件技术。

2.3.1 面向对象技术

面向对象技术是一种软件开发方法,其中包括利用对象技术进行抽象、封装的类、通过消息进行通信、对象生命周期、类层次结构和多态等技术。在类似于 C++和 Java 等面向对象型的编程语言中，编程的模型如图 2.1 所示。

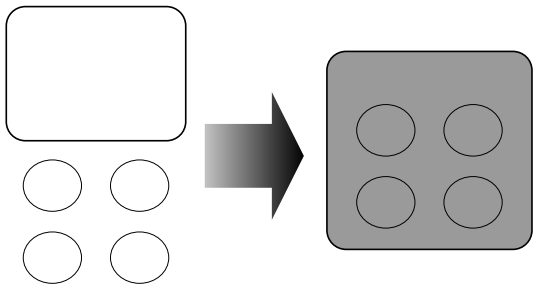


图2.1 面向对象编程的模型

在面向对象的编程模型中，通过对软件模块的对象化封装，使得模块之间独立，从而使复杂的问题简单化。该模型强调的是封装，模块（或者对象）之间的关系在编译的时候被固定，模块之间的关系基本上是静态的，在程序运行时不可改变模块之间的关系，这就是说在运行时是不能换用“零件”的。面向对象语言基本上都有类似 C++语言中的“晚绑定”技术，该技术可以实现多态，在运行期解决方法与对象的绑定，但是这种技术所处理的问题是基于各种编程语言，不适合于在不同的语言之间共享编程对象。

还有一个重要的特点，在面向对象的编程模型中，所有的模块是在同一个程序中使用，在同一地址空间运行，显然不适合分布式对象的处理。

2.3.2 构件技术

构件可以认为是一种包装对象实现的简便方法，并且可以使用它们组装成更大的软件系统。在构件的内部实现上，面向对象技术仍然是重要的手段，但是构件技术的着眼点并不在于内部功能的实现上，而在于提供功能的形式，这种形式一般来说就是一种广泛接受的标准。

为了解决不同软件开发商提供的构件模块（软件对象）可以相互操作使用，它们之间的连接和调用要通过标准的协议来完成。基于构件的编程模型强调的是协议标准，需要提供各厂商都能遵守的协议体系，这种协议标准是开放的。在标准的基础上，构件是可以动态装配的，而且也是可以更换的，即使使用不同语言编写的构件也能做到这一点。而面向对象技术中的对象在不同的语言之间基本上是不可互用的。图 2.2 展示了基于构件的

应用是在运行期动态组合的。

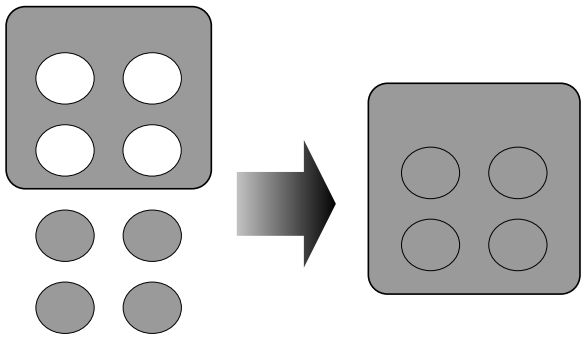
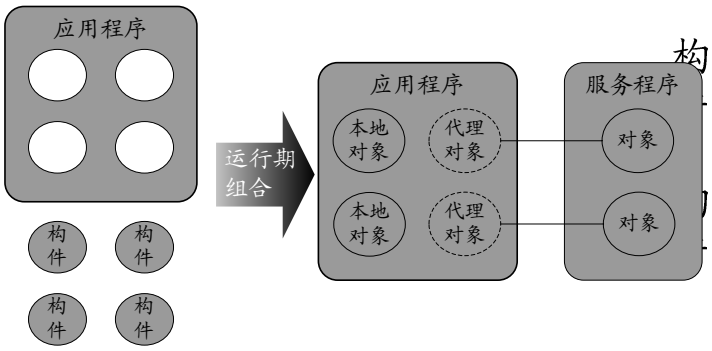


图2.2 基于构件的应用

应用程序

另外多数的构件模型不仅仅是软件架构体系，而且是很好的分布式架构体系，可以简化分布式对象的处理。在构件中包含了自描述信息，也就是所谓的元数据，当构件位于远程或异地进程时，构件框架会根据构件中包含的自描述信息自动生成代理构件，如图 2.3 所示。从使用的角度看，一个代理构件对象和本地对象是没有任何区别的。



构　构件　构件

图2.3 一般的构件运行形式

总的来说，构件和对象的区别主要在以下三个方面：

- 构件担当部署单元的作用，它是基于构件模型的，这种模型定义了符合这种模型的构件所要遵循的规则，而面向对象技术中对象在软件部署阶段已经与软件合为一体，不再具备对象的特征；
- 构件提供了对一个或多个对象实现的包装，它是对象的一种提升，使得对象适合于单独部署；

- 构件是一个组装单元，各种功能的构件可以灵活地被用于构件软件系统，而且实现相同功能的构件可以相互替换，而面向对象技术很难实现这样灵活的组装功能。

2.3.3 构件技术的优点

构件技术降低了大规模软件的开发维护复杂度，使得软件复用进入了一个崭新的阶段。然而更重要的在于由于对构件功能的访问是通过系统生成的中间层实现的，系统就可以通过适当的机制对构件的运行进行控制。通过中间层间接访问构件为编程的灵活性与透明性提供了清晰、坚实的理论模型，构件技术中使用的中间层模型又称代理构件模型。

人们逐步认识到中间层不但能帮助实现远程通信，而且还能够解决许多编程中的“正交”问题，正交问题不很严格地说就是那些与原始问题无关但又必须解决的程序问题。传统 C/S 编程模型中，客户端程序不仅要处理自己的事务逻辑，还要自行解决与服务端的远程通信问题，相对于客户端的事务逻辑，远程通信就是一个正交问题。代理构件模型可以将这种正交问题抽出来统一处理，从而简化了客户端程序的编程逻辑。代理构件模型主要解决的正交问题有：

- **访问顺序管理** 有些构件支持多线程访问，更多的构件只支持单线程访问。对于单线程构件的访问需要由系统决定访问顺序。
- **安全性管理** 对于来自于不同途径的构件，在运行时需要对运行空间、访问权限等加以相应的限制，保证个别构件的运行错误不会央及其他构件或整个系统。
- **使用权的管理** 对于那些同时访问或使用有版权限制的软件，需要系统提供相应的控制机制。
- **持久对象管理** 对于有些操作，当用户应用程序结束后，它所创建的进程还会延续一段时间，这也需要系统对其进行管理。
- **事务管理** 对数据库等进行操作的一个操作序列，由于这种操作序

列具有整体性，它要么被执行，要么完全不被执行。

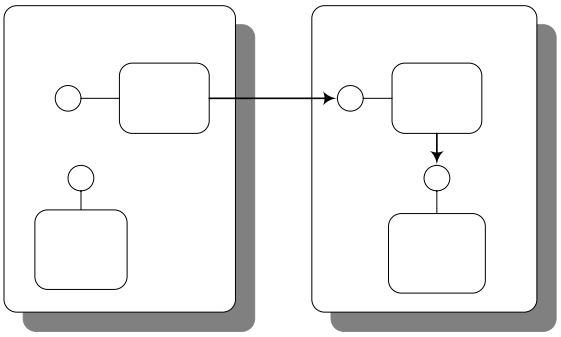


图2.4 构件运行空间的示意图

由于系统自动生成的中间层存在，构件的实际运行空间对用户程序是透明的。图 2.4 展示了进程 A 使用同进程构件 a 和位于进程 B 的跨进程构件 b 的情况。在使用构件 b 的时候，进程 A 显然不可能直接调用构件 b 的接口，而是通过位于一个进程 A 中的代理构件访问构件 b。进程 A 不会感觉到使用构件 a 和构件 b 之间有什么区别。图中所示的代理构件和存根构件就形成了中间层，它们是构件框架在运行时根据需要生成的。

2.4 ezCOM技术*介绍

ezCOM 构件技术[**beij02**]是面向构件的编程模型，它规定了一组构件间相互调用的标准，而且在构件中封装元数据（又称为自描述信息），使得二进制构件能够自描述，从而使构件框架可以根据自描述信息向构件提供运行时支持。

在技术上，ezCOM 技术兼容微软的 COM 技术，但和微软 COM 相比，ezCOM 删除了 COM 中过时的约定，禁止用户定义 COM 的非自描述接口；完备了构件及其接口的自描述功能，实现了对 COM 的扩展；对 COM 的用户界面进行了包装，简化了构件的开发和使用。

ezCOM 构件技术是面向嵌入式系统的构件技术，为了在资源有限的嵌

* ezCOM技术是由北京科泰世纪有限公司研发的软件构件技术，作者参与了该项技术的设计和在和欣操作系统上的实现。ezCOM为ezsy COM的缩写，含义是一种简单易学易用的构件技术。

入式系统中实现面向构件编程技术，同时又能获得 C/C++的运行效率。ezCOM 没有使用类似 JAVA 和.NET 的基于中间代码和虚拟机的机制，而是采用 C++编程，用和欣 SDK 提供的工具直接生成运行于“和欣”构件平台上的二进制代码的机制。用 C++编程实现构件技术，使得更多的程序员能够充分运用自己熟悉的编程语言知识和开发经验，很容易掌握面向构件编程的技术。在不同操作系统上实现的“和欣”构件平台，使得 ezCOM 构件的二进制代码可以实现跨操作系统平台兼容。

ezCOM 技术总结了面向对象编程、面向构件编程技术的发展历史和经验，它的目标是更好地支持面向 WEB 服务的下一代网络应用软件的开发。

2.5 ezCOM构件技术的基本原理

ezCOM 技术是从微软的 COM 发展而来的，对 COM 的认识有助于理解 ezCOM 技术，COM 规范[**Micr95**]是掌握 COM 技术的基础参考，另外，[**Roge97**][**Box98**]都对 COM 技术和使用有比较深入的讲解。

下面是 ezCOM 构件技术的几个基本概念，这些是理解 ezCOM 构件技术的基础。其中有些同样适用于 COM 技术，有些则是 ezCOM 技术独有的，这里并不作区分。

- 每个构件可提供一个或多个接口，应用程序必须通过接口访问构件提供的功能；
- 一个构件封装一个或多个类，这些类称为构件类，它们负责实现构件提供的接口；
- 构件都有唯一的标识与之对应，同时构件都有一个 urn 与之对应，urn 用于定位构件的网络位置；
- 构件中定义的类和接口都有唯一的标识与之对应；
- 接口至少提供两个功能：动态查找构件提供的其它接口；构件客户端程序可以通过接口引用记数控制构件的生命周期；

- 客户端程序可以通过构件接口确定该接口所属的构件；

从上面的条款可以明确看出 ezCOM 构件是对象的一种封装，在封装过程中添加了很多限制和要求，这些限制和要求的目的就是使得对象更适合于动态软件组装，更适合于在运行期由系统向对象实施控制。另外 ezCOM 技术还有一些问题相对比较复杂，下面分别讨论。

2.5.1 二进制标准

对任意平台(包括硬件平台和操作系统平台)，ezCOM 规定了内存中接口虚函数表的形式和通过虚函数表调用函数的标准方法。因此，任何能通过指针调用函数的编程语言(C、C++、VB 等)都可以用来编写可以互操作的构件，只要它们是以同样的二进制标准编写的。双重间址(接口指针是指针的指针)使得同一对象类的多个实例能实现虚函数表共享，在有许多对象实例的系统中，虚函数表共享可极大地减少对内存资源的需求。图 2.5 就是虚函数表在内存中摆放示意图，所有正确初始化的虚函数表在内存中都具有图中的形式，这种形式就是接口的二进制标准。

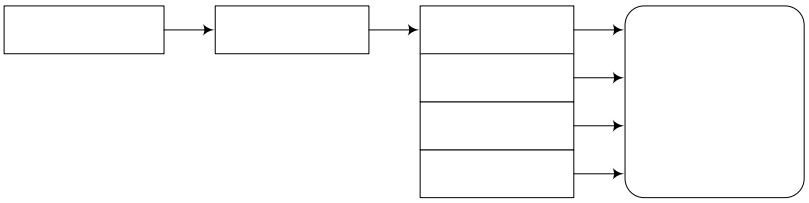


图2.5 接口虚函数和接口指针

2.5.2 构件与构件实例

构件是对象的逻辑封装，在使用构件进行相关操作之前，首先要生成构件的实例，称为构件实例。从“人”的概念与特定人之间的关系出发，可能更容易理解构件与构件实例的关系，如果要对“人”实施“对话”操作在逻辑上是错误的，只有在实例化到特定人，“对话”操作才能正常进行。构件只有在实例化后才会与特定的资源相联系(分配内存并初始化)，针对这些资源的操作就可以正常进行了。

在不产生理解歧意的情况下，为了讨论方便构件实例被简化为构件，读者可以根据语境进行判断。

2.5.3 构件接口

在 ezCOM 中，应用程序对于构件的操作是通过接口实现的，接口是一组在语义上相关的操作，这些操作称为接口方法，在构件实例化后，可以使用接口对构件实例进行相关操作。在 ezCOM 编程中，使用构件实际上多数情况下是使用接口，有时候用户也可得到构件实例本身，但一般它都是作为参数传递给其它操作，或者通过它获取相关接口。

接口是构件标明其服务的一种约定，对下述几点的理解是掌握接口的关键：

- 接口符合二进制标准。接口在内存中的摆放形式是标准化的，这样才能保证不同语言编写的程序可以按照相同的规则使用接口。与之相对应，构件实例在内存中的摆放形式是没有要求的，只要接口方法能够识别。构件定义一般都是使用一种语言实现，因此接口方法自然能够识别构件实例的内存摆放形式；
- 接口不是构件。构件是可以实例化的，而接口不能实例化，它只能与特定类型的构件实例绑定才能正确使用。只要遵循接口的定义，不同类型的构件可以实现相同的接口，这是实现多态性的基本原理；
- 构件至少拥有一个接口。接口是外部访问构件的唯一途径，没有接口的构件是毫无疑义的。当然，构件可以拥有多个接口，多个接口可以分解构件定义的粒度，而且按照逻辑划分接口方法更符合人的认知；
- 客户程序只能通过接口指针使用接口。当客户程序访问构件实例时，客户程序只能通过接口指针调用接口方法。这个指针隐藏了构件的内部实现，用户看不到构件实例内部构造。这与 C++对象指针不同，在 C++对象中，客户程序可直接访问对象的数据，而在

ezCOM 中客户程序只能通过调用具有指针的接口来访问对象。正是这种封装允许 ezCOM 提供能实现本地/远程透明的、高效的二进制标准；

- 接口是强类型的。由于每个接口都分配有接口标识符，从而消除了任何产生冲突的可能性。在使用接口时，使用接口的标识符来获取接口的指针，可以消除命名冲突从而避免运行失败，提高了系统的稳定性。

2.5.4 通用唯一标识符

ezCOM 技术使用 128 位整数的全球唯一标识符，它能保证在全世界范围内能唯一标识每一个接口和每一个构件。这样可确保即使在拥有庞大数量构件的网络环境下，ezCOM 构件都不会出现连接到错误构件和接口的情况。这种全球唯一标识符叫做 UUID(Universally Unique Identifier)，由开放软件基金组织在分布式计算机环境规范中定义[**Open98**]。

UUID 用于标识构件时称为 GUID(Globally Unique Identifier），用于标识接口的 UUID 被称为 IID (Interface ID)，用于标识类的 UUID 被称为 CLSID (Class ID)。“和欣”SDK 提供了能自动产生 UUID 的工具 uuidgen，另外，CoCreateGuid 函数是 ezCOM API 的一部分，可以使用该函数生成 UUID。

标识构件、类和接口的 UUID 都会封装在构件的二进制映像文件中，使用它们主要是为了正确定位和识别构件、类和接口。比如在运行时，构件客户端可以使用接口标识(IID)查询特定接口。

2.6 ezCOM对COM的技术扩展

ezCOM 技术继承了微软 COM 技术的许多特征，但同时针对 COM 的不足进行了一些重要扩展，有些扩展简化了构件的开发和使用，有些则提供了新的功能。

2.6.1 构件类别

构件客户端程序使用构件一般都是用 CLSID 指定构件，然后用 IID 获得构件的某个接口。然而，有时候，用户可能希望使用“遵循了某些语义限制”的构件，在有些情况下这种做法非常有用。为了满足这种需求，ezCOM 引入了构件类别的概念，使得构件能够按照提供的功能进行分类，用户可以根据需要使用某一类构件。

一个构件类别的定义有点类似于构件类的定义，它包含了一组接口，但构件类别本身并不提供这些接口的实现，这些接口只是定义该构件类别提供的服务。如果某个构件类要提供构件类别定义的服务，就要继承这个类别并实现这个类别包含的所有接口。根据以上分析，构件类别实际上是一个抽象类或超类。

每个构件类别都有一个类似于 CLSID 的标识，称为构件类别标识，当一个构件类继承构件类别后，它同时也继承了构件类别标识。用户可以使用构件类别标识创建构件类别，系统获得这个创建构件类别的请求后，通过标识定位某一个实现该构件类别的构件，然后创建构件实例返回给用户，用户则可以进行相关调用。

2.6.2 智能指针

在接口和对象的使用方面，微软的 COM 建立了一整套的使用规则，这些规则基本上都是机械式的操作。比如建立接口的新引用时，用户必须显式调用接口的 AddRef 方法。在废弃接口引用时，显式调用 Release 方法。它们与用户的编程逻辑并没有必然联系，有时用户在编写程序时往往会忘记遵守这些规则，导致内存泄漏和无效的接口引用。

ezCOM 技术将这些规则掩藏在智能指针中，使得用户编写面向构件的程序时只需关心自己的编程逻辑。在 ezCOM 中有三类智能指针：接口智能指针、类智能指针和类别智能指针。

接口智能指针

接口智能指针是接口指针的封装，它用于封装特定类型的接口。在 C++ 语言中，接口智能指针被定义为类，其中只有一个成员变量，这个成员变量就是一个接口指针。图 2.6 简单地展示了接口智能指针，图中有一个 IFoo 接口，IFooRef 是 IFoo 接口的智能指针，它的唯一的成员变量 m_pIface 指向 IFoo 接口。

接口智能指针能够自动处理引用记数。一个智能指针在定义的时候，它并不指向任何接口。在它被赋值以后，智能指针会自动调用 AddRef 方法，增加引用记数。当它的生命周期结束时，智能指针会调用 Release 方法，删减引用记数。接口智能指针在被赋值之前，如果它已经指向一个接口，它会先调用 Release 方法，删减旧接口的引用记数。然后指向新的接口，并调用 AddRef 方法，增加新接口的引用记数。

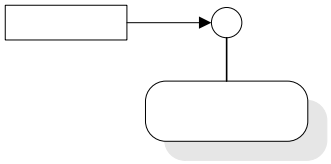


图2.6 接口智能指针

接口智能指针实现了它所封装的接口的所有方法，接口智能指针方法的实现很简单，只是通过成员变量调用构件实例接口的对应方法。用户可以通过调用智能指针的方法调用接口的相应方法。

类智能指针

在 COM 编程中，假设一个 Cobject 对象实现了 IA，IB，IC 三个接口，FA 是 IA 的方法，FB 是 IB 的方法，FC 是 IC 的一个方法。如果用户要调用 FA，FB，FC 三个方法，要通过比较烦琐的过程来完成，表 2.1 中的代码是一种可能形式，为了简单起见，这段代码没有考虑方法调用失败的情况。虽然只是简单的三个方法调用，却写了九行程序。ezCOM 技术引入了类智能指针，简化了类似的方法调用。

IFooRef
m_pIface

表 2.1 调用三个接口的方法

```
……
// 假设已经获得 IA 接口的指针 pIA
pIA->FA(…);
IB *pIB;
IC *pIC;
PIA->QueryInterface(IID_IB, &pIB);
pIA->QueryInterface(IID_IC, &pIC);
pIB->FB(…);
pIC->FC(…);
pIB->Release();
pIC->Release();
……
```

类智能指针是构件类指针的封装，针对刚才的例子，CObject 的类智能指针类型为 CObjectRef，它继承了三个接口的接口智能指针 IARef、IBRef 和 ICRef，而且它还实现了 Instantiate 方法，使用该方法可以创建构件实例。

使用类智能指针，前面的三个方法调用的代码可以写成表 2.2 的形式，代码大大简化了，而且用户无需关心 QueryInterface 这种与程序事务逻辑无关的操作。另外，类智能指针也封装了构件实例的创建，用户无需再使用复杂的 CLSID 和 IID。

表 2.2 类智能指针的使用

```
CObjectRef Object;
HRESULT hr = Object. Instantiate(CTX_SAME_DOMAIN);
if (Object.IsValid()){
    Object .FA(…);
    Object .FB(…);
    Object .FC(…);
}
……
```

类别智能指针

类别智能指针是对构件类别指针的封装。类别智能指针的功能和封装形式都与类智能指针相似。类别智能指针简化了构件类别的使用，它的使用方式也和类智能指针类似。

通过类别智能指针，用户可以创建特定构件类别的实例，而且可以方

便地调用构件类别的所有接口方法。类别智能指针创建构件实例的方式和类智能指针不太一样，前者需要系统通过构件类别标识定位一个构件类，然后才能创建该构件类的实例。用户不用关心这些区别，因为智能指针封装了这些区别。

2.6.3 自描述数据类型

为了更好支持面向构件编程，ezCOM 设计了自描述数据类型：EzStr、EzByteBuf、EzStrBuf、EzArray。所谓自描述数据类型简单说，就是数据结构本身包含描述自身的信息，通过这些信息可以掌握数据的全貌，如数据的内存布局和占用空间情况等。C 语言中的数组不是自描述数据结构，因为不能通过数组本身了解它所包含的元素个数，而 int 类型就是一个自描述数据结构。

自描述数据类型的作用表现在两个方面：在调用远程构件的接口方法时，只有接口的参数类型是自描述的，系统才可以在不同的地址空间传递参数；使用自描述数据类型还可以进行安全检查，保证数据操作不会超出数据边界。

EzStr数据结构

EzStr 用于存储常量字符串，它有一个定长的存储区，可以存储宽字符串。EzStr 数据结构中还保存这个定长区域的长度。图 2.7 是 EzStr 的存储结构示意图。

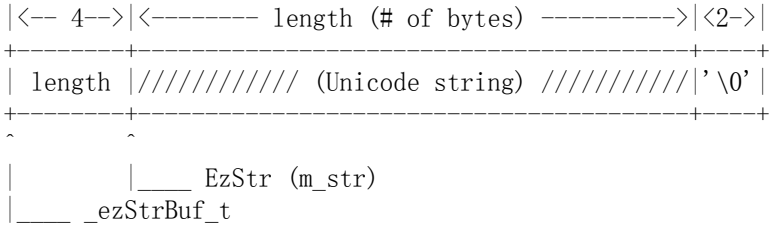


图2.7 EzStr的存储结构示意图

从图 2.7 可以看出 EzStr 由三部分组成，第一部分存放第二部分的长度，

第一部分占四个字节。第二部分存放 Unicode 字符串。第三部分存放两个字节的’/0’字符，作为字符串的结束标志。C 语言中一般使用 char 型指针或数组表示字符串，用户很难以一种安全有效的方式获得字符串的长度，EzStr 解决了这个问题。

EzByteBuf数据结构

EzByteBuf 是一种自描述的字节串缓冲区数据结构。它可以存储一定长度范围的字节串，EzByteBuf 能够描述它存储字节串的能力和当前存储区域的使用情况。图 2.8 是 EzByteBuf 的存储结构示意图。

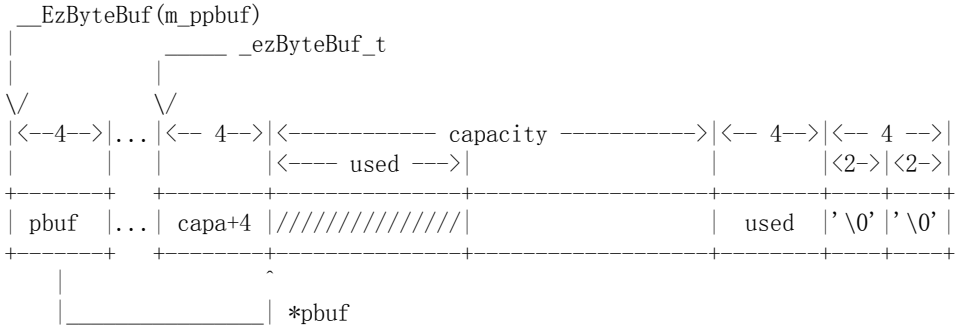


图2.8 EzByteBuf的存储结构示意图

从图 2.8 可以看出 EzByteBuf 的成员变量 m_ppbuf 间接指向存储区域。存储区域之前有一个 4 字节区，它与 EzByteBuf 的存储能力相关，而存储区域后的 4 字节区与存储区域的当前使用情况相关，最后的 4 字节区存放’/0’字符标识数据结尾。

EzStrBuf数据结构

EzStrBuf 是一种自描述的字符串缓冲区，它是 EzByteBuf 和 EzStr 的结合体。EzStrBuf 与 EzByteBuf 相比，最主要的区别是 EzStrBuf 中存放的是一个 EzStr 对象，而 EzByteBuf 存放字节串形式的数据。它的存储结构和 EzByteBuf 非常相似。

EzArray数据结构

EzArray 用来定义多维、定长、自描述数据结构的数组。图 2.9 是 EzArray 的存储结构示意图。

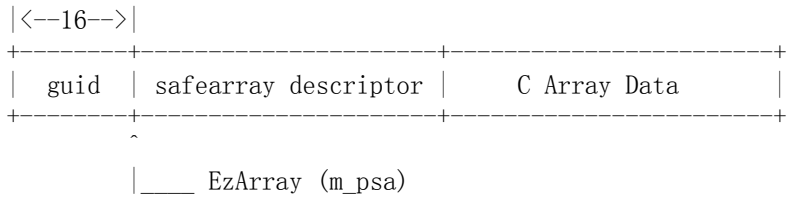


图2.9 EzArray的存储结构示意图

在 C++语言中，EzArray 被定义为一个类，它有一个成员变量 m_psa，m_psa 是 SAFEARRAY 指针类型的变量。SAFEARRAY 是微软 COM 定义的数据结构，有关 SAFEARRAY 数据结构的描述参见[**Micr95**]。EzArray 实际上是在 SAFEARRAY 前面增加了 16 个字节，用于存放 guid。

关于 ezCOM 的自描述数据类型使用，可以参看“和欣”SDK 提供的帮助文档[**Beij02**]，这里就不再详细讨论。

2.6.4 CDL 语言

微软的 COM 编程主要是以接口为核心，开发过程（C++语言为例）通常是先设计接口，使用 IDL 语言描述接口。通过编译 IDL 文件生成描述接口的头文件，然后使用 C++语言定义继承接口的类，实现接口中的方法。

ezCOM 构件使用 CDL 语言进行开发，CDL 语言是构件描述语言(Component Definition Language)，它用来定义构件的基本框架，构件接口和构件类的原型定义都可以在 CDL 文件中完成。

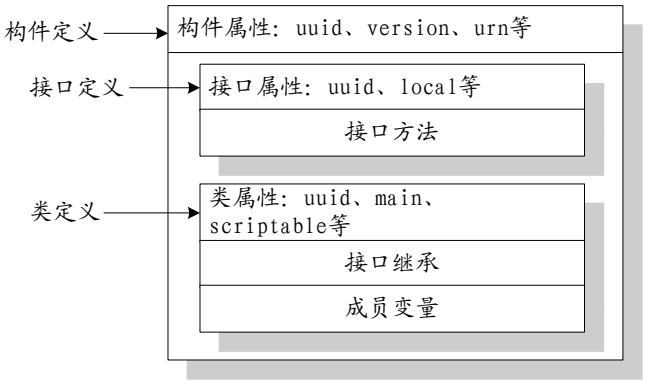


图2.10 CDL文件的基本格式

CDL 文件的出现使得构件开发的核心从接口转向了构件。从构件开发者和使用者的角度考虑，这种方式淡化了接口的地位，突出强调构件，实际上这种方式更符合逻辑思维。面向构件编程的目的是构件和构件提供的服务，接口只是一种辅助手段。从构件编写的角度看，最终是构件提供服务而非接口提供服务。从构件使用的角度看，用户创建了一个构件实例。通过这个实例，他就应该能够获得构件的服务，而不是说用户还需要 IID 获得接口，再通过接口获得服务。联系前面讨论的类智能指针，类智能指针也是这种以构件为中心的良好体现。

附录中通过示例说明了 ezCOM 构件的基本开发流程，在这个示例中开发 hello 构件就使用了 CDL 语言，参看附录可以增加对 CDL 语言的感性了解。图 2.10 展示了 CDL 文件的一种基本格式。

图 2.10 只是 CDL 文件的一种基本格式，CDL 文件也可以只定义接口，只定义接口的 CDL 文件不能单独用于创建构件。它的信息可以用于其它的 CDL 文件，使用方式可以参看 CDL 相关的文档。图 2.10 给出的文件格式是定义构件时使用的。在这种 CDL 文件中可以定义多个接口和类，类定义中的接口继承部分可以包含本文件定义的接口，也可以是别的 CDL 文件中定义的接口。

正如附录中的开发示例所显示，编写 CDL 文件只是 ezCOM 构件开发的第一步。使用 CDL 开发 ezCOM 构件可以简化开发过程，在设计和编写完 CDL 文件后，“和欣”SDK 提供的工具可以根据 CDL 文件生成构件的

代码框架，构件编写者只需在框架内添加实现代码。自动代码生成降低了用户编程工作量，而且可以减少代码输入出错的机会。

CDL 文件包含着大量的信息，如类和接口的原型信息、类和接口的标识以及构件定位信息 urn 等，ezCOM 技术在构件开发和运行过程都充分使用了这些信息。ezCOM 技术的智能指针也是在构件开发过程中通过 CDL 自动生成的。在构件开发过程中，一些构件自描述信息也会从 CDL 文件中提取出来，并且最终打包在构件的可执行文件中。在构件运行时，这些自描述信息会被系统使用，用以支持构件的运行，关于构件自描述信息的封装和使用会在后面的章节论述。

2.7 ezCOM技术的意义

对于面向网络的应用软件开发，及对开发操作系统这样的大型系统软件而言，采用 ezCOM 构件技术具有以下的意义：

- 不同软件开发商开发的具有独特功能的构件，可以确保与其他人开发的构件实现互操作；
- 在对某一个构件进行升级时不会影响到系统中的其它构件；
- 不同的编程语言实现的构件之间可以实现互操作；
- 提供一个简单、统一的编程模型，使得构件可以在进程内、跨进程甚至于跨网络运行。同时提供运行的安全、保护机制；
- ezCOM 构件与微软的 COM 构件二进制兼容，但是 ezCOM 的开发工具自动实现构件的封装，简化了构件编程的复杂性，有利于构件化编程技术的推广普及；
- ezCOM 构件技术是一个实现软件工厂化生产的先进技术，可以大大提升企业的软件开发技术水平，提高软件生产效率和软件产品质量；
- 软件工厂化生产需要有零件的标准，ezCOM 构件技术为建立软件

标准提供了参考，有利于建立企业、行业的软件标准和构件库。

2.8 小结

为了更好地理解 ezCOM 构件技术以及和欣操作系统的设计和实现，本章首先介绍了软件构件的相关背景。

构件技术的最初动机是实现软件领域的复用，经过发展构件技术的作用远远超出了复用的范畴，它已经被提升为软件体系结构层面的一种有效的软件组建手段。文中在软件组建的范围里阐述了构件技术的相关概念；进一步，还深入分析了构件技术和面向对象技术之间的联系和差异；最后讨论现有的构件技术模型，简单介绍了它们的发展现状。

在介绍了构件技术之后，本章重点论述和欣操作系中所使用的 ezCOM 构件技术，后面的章节中我们会看到，ezCOM 技术是和欣操作系统设计实现过程中的一项重要技术。

ezCOM 技术和微软 COM 技术兼容，在原理上它们有很多相似之处，但 ezCOM 技术针对 COM 技术的缺陷和不足进行了扩展。通过扩展，ezCOM 技术简化了面向构件的编程实践，ezCOM 构件的元数据封装也为建立有效的构件运行机制奠定了基础。另外，文中还简单介绍了 ezCOM 构件的开发和 ezCOM 技术的工程意义。作者参与 ezCOM 技术的研发，尤其在 ezCOM 技术对 COM 技术扩展方面做了较多的工作。

第三章　和欣操作系统灵活内核的设计与实现

和欣操作系统是面向因特网时代的构件化操作系统。在论述和欣操作系统之前，首先展示作者提出的一种新型操作系统模型——灵活内核模型。灵活内核模型是作者在深入参与和欣操作系统早期研发的基础上提出的，它的思想已经在后来的和欣操作系统 1.0 版中得以应用和体现。现代操作系统的设计和实现都相当复杂，和欣操作系统也不例外。本章并不讨论系统设计和实现的全部细节，而只是着重论述与灵活内核模型相关的关键技术。

3.1 灵活内核模型

3.1.1 灵活内核模型的动机

由于操作系统模型之争的广泛性，大多数从事操作系统研究和设计的学者都意识到无论整体内核模型还是微内核模型都不是尽善尽美的。然而，这些争论还是带来了一些积极的后果，双方的支持者都试图对自身有所改变，以弥补各自理论体系存在的缺陷。

多年来，微内核模型的支持者对整体内核模型的可移植性和可扩展性普遍存在质疑。整体内核模型的支持者尽量使其内核模块化和层次化，减少模块之间的依赖性，而且传统的操作系统也开始支持可动态装载的系统模块，提高系统的可扩展性和可移植性。

整体内核模型的支持者对微内核模型最有力的反击往往是效率问题。的确由于微内核系统将很多系统服务放在核外，应用程序同各种系统服务之间的交互是通过进程间通信等类似机制完成，整个系统的效率难免会受到影响。因此过去的十几年里，微内核模型的支持者忙于研究如何提高系统的效率，而一些微内核系统把有的系统服务又重新置于内核中，从某种程度上确实提高了效率，但是这种做法的代价在一定程度上违背了微内核的初衷。

操作系统扩展技术则希望摒弃微内核和整体内核之间关于系统模型的纷争，纯粹讨论通用的系统扩展技术，从而弥补整体内核和微内核的不足之处。如第一章中所分析的，现有的操作系统扩展技术并没有脱离这两种系统模型的阴影。基本上所有这方面的工作仍然集中在这两种模型之下，因此也很难解决它们之间的矛盾。另外，它们之间的某些矛盾不可能完全解决，例如：安全性和效率问题。操作系统扩展技术方面的研究工作只能弱化这些矛盾，如果不从系统模型上有所改变，这种情况也不会转变。

我们认为，微内核和整体内核的优点是不可兼得的，它们的优点以及缺点是模型的本质体现。要想突破这两种模型的限制，势必在系统模型上有所作为，灵活内核模型正是基于这样的认识而提出的。灵活内核模型是一种具有很强动态特性的系统模型，它可以在运行期决定系统扩展模块的运行位置。根据需要扩展模块可以运行在内核空间或者用户空间，从而获得用户所希望的系统特性。它的灵活性最主要体现在它能够对单个扩展模块设置配置策略，不像微内核模型的所有扩展模块都运行在用户空间，而整体内核模型的所有扩展模块都运行在内核空间。

3.1.2 灵活内核模型定义

灵活内核模型是一种全新的操作系统模型，它具有极强的动态扩展特性，因此它也是一种可扩展的操作系统模型。图 3.1 展示了灵活内核模型的基本思路，为了区别于其它的系统模型，下面是判断灵活内核模型的基本准则：

准则 1 系统内核的功能相对稳定，只实现相当有限的系统功能；

准则 2 在内核的基础上，系统支持与内核映像独立的、符合一定标准的系统扩展软件；

准则 3 用户可以动态配置系统扩展的运行位置，系统扩展可以配置在内核空间中运行，也可以配置在用户空间运行。

准则 1 中的系统内核称为灵活内核(agile-kernel)，与微内核相似，灵活内核也只提供相当有限的基本系统功能，其它的系统功能则由系统扩展来

提供。这样做的原因是多方面的，首先提高系统的可维护性和稳定性是重要的一个方面。另外，正如引言所阐述的，构造大而全的操作系统内核面临着各种各样的困难，因此大多数系统研究者已经不再采用这种方式构建系统。

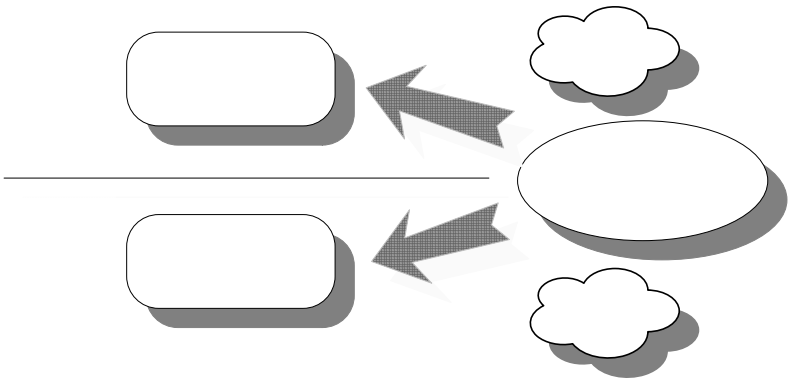


图3.1 灵活内核模型

准则 2 说明了灵活内核模型是一种可扩展的操作系统模型。其中提到的系统扩展软件是符合一定标准的功能软件模块，简称为系统扩展。这里的标准是指灵活内核和系统扩展之间的一种约定，灵活内核按照这个标准可以识别、装载和运行系统扩展，并且把系统扩展实现的功能通过一定的形式提供给用户，以达到扩展系统功能的目的。

准则 3 是灵活内核模型的核心要点，也是它不同于现有操作系统模型的根本所在。如图 3.1 所显示的，灵活内核系统可以根据性能需要动态配置系统扩展的运行位置。当系统的安全性和稳定性是关注的焦点时，或者系统扩展的稳定性没有保障时，系统扩展可以配置在用户空间运行，即使系统扩展的内部错误也不会导致操作系统内核的崩溃；当系统的效率是首要问题时，或者系统扩展具有极强的可信度时，系统扩展可以装载到内核空间运行。这种情况下，由于进程间通信的减少，使用内核中系统扩展的软件效率就会有很大提高。

灵活内核模型是在对整体内核模型和微内核模型充分认识的基础上建立的一种高度灵活的系统模型。可以说灵活内核模型提供了在这两种模型

之间的一种有效转换方式。系统扩展运行在用户空间则可以看作是采用了微内核模型的扩展方式，它以服务进程身份提供扩展功能；系统扩展运行在内核空间可以看作是采用了整体内核模型的扩展方式。需要说明一点，灵活内核模型是作者在参与和欣操作系统项目早期研发工作后，进行理论总结提出的。它提取了和欣操作系统的理论模型，对于系统设计有更加通用的指导意义。

3.1.3 关于灵活内核模型的实现

微内核和可扩展操作系统研究为灵活内核模型的实现提供了大量可借鉴的技术。譬如在考虑灵活内核模型的内核应该实现哪些基本功能，哪些传统的系统功能应该通过系统扩展的方式实现，这些完全可以参考现有微内核设计中的思路。但灵活内核模型的实现存在一些特有的问题，这些问题在现有的研究中是没有遇见过的。

首先，灵活内核模型对系统功能的提供方式有比较特殊的要求。一般系统扩展会频繁使用内核提供的各种系统功能，而系统扩展的运行位置是不确定的。系统扩展可能在内核空间中使用系统功能，也可能在用户空间使用系统功能，系统扩展应该能够同时应付这两种情况。在编写系统扩展的时候，当然不能要求编写者意识到系统扩展的运行位置带来的影响，否则编程的复杂性会大大增加。为了能够以同样的方式在内核和用户空间使用系统功能，这就要求系统功能具有一定的位置透明性。

其次，用户如何使用系统扩展提供的服务也是实现灵活内核需要解决的问题。系统扩展在装载运行后，其客户端需要通过某种途径使用它所提供的服务。当系统扩展运行于内核空间中，客户端对系统扩展的调用实际上是到内核空间的调用；而当系统扩展运行在用户空间时，客户端对系统扩展的调用就变成进程间通信的机制。理想的情况下，用户在使用系统扩展的时候无需关心系统扩展的运行位置，那么系统扩展提供的功能在使用上也应该是位置透明的。

剩下还有一些一般性的问题，比如系统扩展以何种形式封装、封装基

于什么样的标准等等，这些问题的解决方式是因系统而异的。

实际上操作系统的实现是非常复杂的工作，灵活内核操作系统也不例外，在实现过程中有大量的基础设施需要建立，如进程间通信、任务调度、硬件管理等，这些属于现代操作系统的基本范畴，因此在这里就不讨论这些内容了。

3.2 和欣操作系统

和欣操作系统是一个灵活内核系统，在实现和设计过程中，和欣操作系统采用了面向构件的技术方法，因此“和欣”系统是基于灵活内核的构件化操作系统。

由于“和欣”系统采用了全新的操作系统模型和先进的基于构件的软件工程思想，因此“和欣”系统具有一些鲜明的特点，这些特点也是“和欣”系统设计和实现的关键，下面分别进行阐述。

3.2.1 构件化系统扩展

在灵活内核模型中，可以使用系统扩展软件对系统功能进行有效的扩充。系统扩展必须符合系统的要求才能够保证系统能够正确装载和运行系统扩展，因此系统扩展的封装方式和系统的设计密切相关。

和欣操作系统中使用了 ezCOM 构件封装系统扩展，这时称系统扩展为系统扩展构件。当然这样的系统扩展是符合 ezCOM 标准的，这套标准很好地解决了系统扩展封装过程中需要处理的许多问题，比如：系统扩展构件使用 uuid 进行标识，它的功能通过构件接口向用户提供等。

上面只解决了系统扩展的封装问题，要使得系统扩展能够正确识别、装载和运行还需要系统提供相关的支持。在和欣操作系统中存在一个 ezCOM 构件平台，该平台提供了 ezCOM 构件的运行时支持，保障了系统扩展构件能够正确地被使用。

使用 ezCOM 构件封装还解决了一个重要的问题。ezCOM 构件平台可

以为异地空间构件生成本地代理构件，这个过程对于构件的客户端是完全透明的，从而实现了系统扩展的运行位置相对于其客户端是透明的。无论系统扩展运行在内核空间还是用户空间，客户端都能够正常使用系统扩展提供的功能。

3.2.2 内核对象构件化

和欣操作系统的设计与实现使用了 ezCOM 构件技术，它不仅向上提供了 ezCOM 构件平台，而且把构件的概念引入到了系统内核中，从而使得整个系统具有统一的运行模型。

在现代操作系统中，有许多抽象的系统概念，比如进程、线程、地址空间和同步互斥等，这些系统概念都与特定的物理资源或者抽象资源相联系。一些面向对象操作系统把这些系统概念连同相关的操作封装为对象。和欣操作系统也对系统概念进行了封装，但不同的是“和欣”系统使用 ezCOM 构件承载这些系统概念，称其为内核构件。

内核构件通过构件接口向外提供服务，通过使用这些接口，用户可以请求系统服务。比如用户在获取一个进程的接口后，可以使用该接口的功能执行杀死进程、挂起进程和装载模块到该进程的地址空间等操作。关于系统功能接口会在后面阐述。

“和欣”系统的内核构件和系统扩展构件具有相同的构件模型，使得系统扩展构件非常适合承担系统功能扩展的任务。另外，在许多问题上，系统可以使用相同的方式处理它们，比如在处理构件的异地址空间访问时，系统可以用相同的方式生成代理构件，这样大大简化了系统的设计。当然系统扩展构件和内核构件的情况并不完全一样的，系统扩展构件需要更复杂的系统支持。

3.2.3 内核进程

和欣操作系统的内核作为单独的进程运行，拥有独立的地址空间。这个进程被称为内核进程，内核进程拥有的线程称为内核线程。所有的内核

管理功能都是由运行于内核进程中的线程完成，在用户空间使用系统功能，实际上是通过特殊的进程间通信机制实现，因此和欣操作系统的内核是系统功能服务器。

传统操作系统使用系统调用的方式获取系统功能，系统调用要完成从用户态到内核态的切换，以便能够执行特权级指令，操作系统的内核就象一个特殊的静态代码库。

与传统的系统调用相比，在“和欣”系统中，调用系统功能所使用的进程间通信机制的效率要低一些，因为后者不仅要进行模式切换，还要完成调用参数的栈拷贝，以及调用返回时返回值的拷贝。但与两个普通用户进程之间的进程间通信比较，到内核进程的进程间通信效率要高得多，因为后者一次系统功能调用只需两次模式切换，而前者需要四次。

“和欣”系统以进程方式实现内核功能。内核是一个特殊的服务器，这个服务器和用户空间的服务器在形式上是一致的，使用同样一套基础设施即可支持它们，同样降低了系统设计的复杂性。

3.3 和欣操作系统内核结构

同微内核操作系统相似，“和欣”内核只提供相当有限的系统功能，其余大部分功能交由系统扩展构件实现。图 3.2 是和欣操作系统的内核结构示意图，从图上可以看出内核是严格按照模块化的设计思想。“和欣”的内核自下而上从与硬件交互的硬件抽象层到“和欣”构件平台层次分明，这种结构蕴含模块之间的依赖关系。

硬件抽象层将内核其余部分和底层硬件隔离开，无论在何种硬件平台上，它都负责向上提供统一的硬件操作，这样可以简化操作系统的移植工作。内核调试模块是内核调试器在和欣操作系统上的代理程序，实现了内核调试器与“和欣”内核之间的通讯。内核调试模块位于系统底层，可以对内核中的任意模块进行调试。

位于内核调试模块之上的是系统内核功能模块。内存管理模块管理内核进程及用户进程的内存分配和释放，以及地址空间的映射；任务管理模

块实现以线程为单位任务调度等；模块管理模块负责软件模块的装载和卸载等操作；设备管理模块管理所有设备驱动构件，是构件化驱动模型的基础设施。内核功能模块不仅负责与系统资源相关的管理工作，而且要管理各自的内核构件，比如进程管理模块要管理进程和线程内核构件，并且为这些内核构件提供必要的底层支持。内核功能模块并非完全独立，它们之间存在必要的交互，比如模块管理在装载软件模块的时候，要进行地址空间的映射，必然要使用内存管理模块提供的功能。

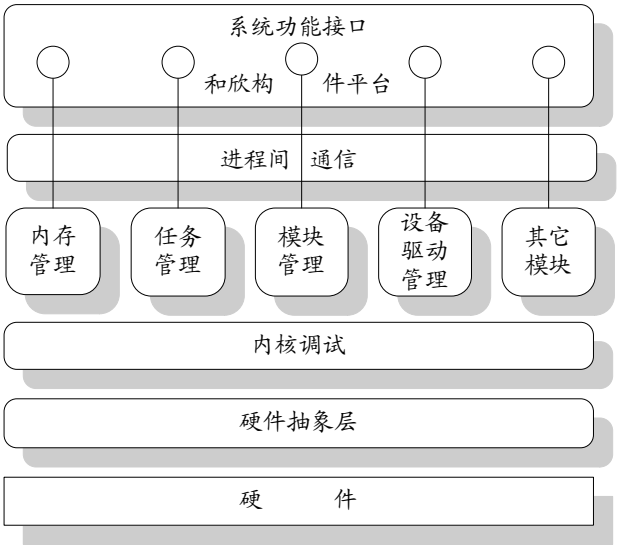


图3.2 和欣操作系统内核结构

进程间通讯模块是负责进程间信息交换的系统基础设施，实现从用户进程到内核进程、用户进程之间的基本通讯机制。“和欣”系统调用全部是基于 ezCOM 构件接口，它们都是通过进程间通讯模块实现跨地址空间调用；“和欣”构件平台包括系统功能接口和应用程序接口，内核通过它向用户提供系统编程接口，对于用户，“和欣”构件平台是系统的唯一可见的部分。另外“和欣”构件平台还提供 ezCOM 构件的运行时支持。

下面将对内核的主要模块进行阐述，“和欣”构件平台和设备驱动管理模块与灵活内核实现有着更加紧密的联系，需要详细论述，在其它章节会专门讨论。

3.3.1 内存管理

“和欣”系统的内存管理涉及两个概念：地址空间和堆的概念，而且它们都使用构件进行了封装。

逻辑上，地址空间就是一段或多段可被程序访问的内存地址，是内核管理的首要资源。地址空间构件负责管理地址空间的分配，以及地址空间之间的映射关系。内核中存在三类地址空间构件实例，它们分别管理物理地址空间、内核进程虚拟地址空间和用户进程虚拟地址空间。每个用户进程拥有一个独立的虚拟地址空间构件实例，负责管理本进程的虚拟地址空间。图 3.3 是内存管理基本示意图。从图上可以看出，虚拟内存是物理地址空间管理的硬件内存分配映射到虚拟地址空间的；进程堆则是从进程的虚拟地址空间管理的虚拟内存中分配来的；应用程序动态申请内存是从堆上分配的。这些操作都是通过每个构件的接口完成的。

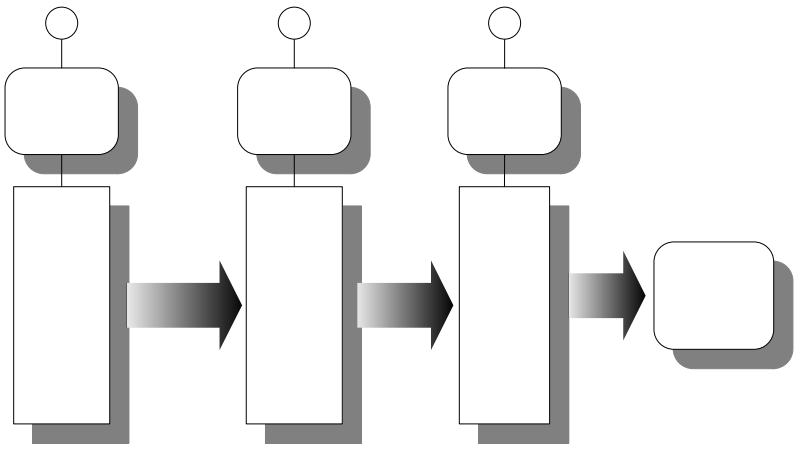


图3.3 内存管理示意图

“和欣”系统以页为单位管理地址空间。每个地址空间构件维护一个树状数据结构，以记录地址空间的分配和映射情况。树状数据结构的每个节点对应一段地址，用来记录这段地址的起始和结束位置、映射的目标地址空间构件的指针和映射起始位置。例如某个节点可以表示如下信息：内核虚拟地址空间上从 0x1000000 到 0x10001000 的这段地址被映射到物理地址空间上从 0x00123000 到 0x124000 的地址。对地址空间的管理通过节点的增加、删除、合并、映射等操作完成。从虚拟地址空间到物理内存地址

地址空间　　地址空间

空间映射的改变会导致硬件页表的修改。

对其它操作系统而言，堆的概念仅适用于应用程序；而和欣操作系统将堆的概念延伸到内核。同应用程序一样，“和欣”内核采用堆算法管理内存的分配和释放。对于地址空间和堆的关系，可以理解为：当内核需要一块内存时，程序首先从相应的地址空间上分配以页为单位的内存，然后将这段内存交付给堆算法，由后者对它进行细粒度的管理，以减小由于分配单位过大造成的内部碎片。从用户空间分配内存是在用户进程的虚拟地址空间的堆上进行，原理和内核中的内存分配是一样的。

堆算法依赖于地址空间分配的内存，而地址空间节点又必须从内核堆上分配。为解决这种递归的依赖关系，除了常规的内核堆，在内核中还设有一个固定尺寸的常驻堆，该常驻堆直接操作物理内存而不依赖任何地址空间对象，物理内存地址空间在该堆上分配节点。

3.3.2 任务管理

任务管理模块负责管理的内核构件包括所有进程构件和线程构件，所有内核进程、线程和用户进程、线程都对应唯一的构件实例，其它模块和用户可以使用它们的接口对进程或者线程实现相关的操作。

任务调度是任务管理模块的另一个重要工作。“和欣”内核使用带优先级的轮转(round robin)与先进先出(FIFO)混合调度算法，以线程为单位进行调度。调度算法支持 16 级线程和进程优先级。对于一般优先级的线程，“和欣”内核使用轮转算法，即分配给每个就绪线程一定长度的时间片(通常为 10 毫秒)，当前线程的时间片用尽后，内核将选择运行另一个优先级最高的线程；对于优先级为 0(0 为最高优先级)的线程，内核使用先进先出算法，即当前线程持续运行，直到阻塞或主动放弃 CPU。在默认情况下，每个线程继承其父进程的优先级。调度算法不区分内核进程与用户进程。

“和欣”内核通过三个链表管理线程的运行，分别是：就绪队列、阻塞队列和挂起队列。就绪队列中的线程按优先级顺序排队，调度算法总选择队首线程作为当前线程，刚刚被抢占的线程则被放在队尾。

3.3.3 进程间通信(IPC)

和欣操作系统的进程间通信机制比较复杂，仅在这里做一简单介绍。进程间通信模块支持跨内存地址空间的 ezCOM 接口方法调用，它采用的机制类似于远程过程调用(RPC)。

在获取一个远程构件的接口时，“和欣”构件平台会在本地生成远程构件的代理构件，这个代理构件的接口形式和远程构件的接口是一致的，但虚函数表填写的是陷入进程间通信模块的函数方法。调用代理构件的接口方法就会进行一次用户进程到内核进程的进程间通信。进程间通信是支持 ezCOM 构件运行机制的基础设施，它同构件平台一起建立了构件的运行支持环境，关于 ezCOM 构件的运行会在下一章详细讨论。

3.3.4 模块管理

模块是指以文件的形式保存在存储介质上的可执行程序，即通常所说的可执行文件。和欣操作系统支持的软件模块有两种形式：动态链接库和 EXE 文件。

ezCOM 构件是以动态链接库的形式编译链接和使用的。为执行一个程序，内核必须将相应的模块从存储介质装载到内存。连接函数导出表、导入表，重定位全局指针，初始化数据段、堆、栈等，以及模块运行必须的数据结构。对于动态链接库，需要装载到共享地址空间。对于 EXE 文件，需要初始化进程自身的地址空间。模块的生命周期也需要有严格的控制，这些工作都由模块管理模块完成。每个模块在装载以后，它都与一个模块构件相关联，模块构件实现模块操作功能，如查找模块内的资源等，这些功能可以通过模块构件的接口进行调用。

“和欣”内核使用延迟装载的方式加载模块，即文件以页为单位划分，每一页延迟到内核读取该页时才被装载到内存。这种方式有效降低了模块初始化的时间。

3.3.5 内核调试器

运行在 Windows2000 上的“和欣”调试器目前支持通过串口与和欣操作系统进行通讯。和欣操作系统上的内核调试模块负责执行调试命令和与调试器的通讯，调试模块与调试器之间的通讯基于 SLIP 协议。

调试模块分为独立的两个部分：用户程序调试桩和内核调试桩。用户程序调试桩负责用户进程的调试，它依赖于内核进程的执行，因而不能调试内核；内核调试桩专门负责内核的调试，其代码与内核代码分离。系统启动时首先初始化内核调试桩，之后内核的所有执行过程都可以被跟踪调试。如果在内核态执行了软中断 INT 3(Intel Architecture 指令)，或者由于内核程序错误陷入硬件异常，内核调试桩将被中断服务程序启动。调试桩启动后，所有线程都将被挂起，调试桩等待并执行从通讯端口传来的调试器命令。

调试器的调试命令包括读/写寄存器或内存、设置/清除断点或条件断点、继续、单步执行等。通过对这些基本命令的加工与组合，Windows2000 端调试器为用户提供了更完善和友好的命令集。

内核调试器使远程调试内核成为可能，是开发内核的利器，大大缩短了内核的开发周期，保证了内核代码的质量。

3.4 “和欣”构件平台

“和欣”构件平台是系统内核与应用程序交互的唯一接口，它包括三个部分：系统功能接口、应用编程接口和构件运行时支持。

系统功能接口是“和欣”向用户开放的内核构件接口，用户程序可以使用这些接口访问内核构件，实现系统功能的调用；应用编程接口是一些传统的函数，其中包括 ezCOM 服务相关的基本 API。实际上，它们形成了标准代码库，“和欣”系统使用代码映射机制保证了在用户和内核空间都可以使用这些代码；构件运行时支持是为 ezCOM 构件在“和欣”系统上运行提供各种支持，如创建构件实例、代理构件的生成等。关于“和欣”构件平台提供的系统功能接口和应用编程接口，在“和欣”SDK 的帮助文档

中都有详细功能介绍和使用说明，而且“和欣”SDK 还提供了一些程序样例，展示了部分系统功能接口和应用编程接口的使用。

3.4.1 构件平台设计目标

“和欣”构件平台为用户提供一套完整的与系统服务相关的 API 函数及系统构件服务，对用户而言，“和欣”构件平台提供了操作系统能提供的所有服务。

“和欣”构件平台要为 ezCOM 面向三层体系结构编程模型提供系统支持。当客户程序需要调用一个软件服务时，“和欣”构件平台可根据与应用相关的元数据动态生成代理构件，自动构造客户端与服务端相关的运行环境，客户端与服务端不再直接发生关系，所有的通信机制都通过“和欣”构件平台自动实现。

基于 ezCOM 构件技术的“和欣”构件平台提供了安全控制以及应用服务器的运行、管理与调度机制，可以提供对用户透明的服务，比如可以根据不同的运行环境进行动态加载、更换、卸载与通信协议相关的网络服务构件、文件系统构件、驱动程序构件、以及分布式的事务处理构件等，为和欣操作系统的灵活内核体系结构提供有力的支持。

3.4.2 构件平台基础设施

构件平台的功能是由相关的基础设施来实现和完成，他们提供了 ezCOM 构件在和欣操作系统上正确运行的保障。

基本构件服务

构件的使用是构件平台的首要任务，作为可执行程序存在的构件如果不运行时没有意义的，用户要通过创建构件对象来使用构件提供的功能。

基本构件服务会向用户提供创建构件对象的功能，构件服务根据用户的要求定位或者下载相应的构件模块，并且把构件模块装载到指定的地址空间，然后创建构件对象，将对象的接口返回给用户，用户就可以使用构

件对象的接口访问构件的功能。当然这仅仅是基本构件服务的一项功能，他还会负责管理构件的生命周期，当一个构件不再被使用后，相应的构件模块就应该从内存中卸载，这样能够保证系统内存资源的有效使用。

散列集

和欣操作系统支持在不同的地址空间创建构件对象，当用户和构件对象处于不同的地址空间时，构件调用要跨越不同的地址空间，调用的参数和返回值也会在地址空间之间进行传递。这些数据的传递是通过进程间通信完成的，但是在传递之前和之后，数据要进行必要的打包和解包工作，这些都是散列集机制来实现。

以上的工作对于散列集机制来说并不是最复杂的，最复杂也是最重要的是接口在不同地址空间的传递。通常，用户和构件对象在同一个地址空间时，用户可以直接获得构件的接口，调用也是直接进行的。而当用户和构件对象在不同的地址空间时，接口要从构件对象空间传递到用户空间，散列集机制在这个过程中会创建存根和代理，用户使用代理接口访问构件功能，正如图 2.4 所示。当用户调用代理接口，散列集机制又会负责数据的打包和解包工作。

命名服务

命名服务机制提供一种在系统内部发布、获取、使用 ezCOM 构件的方法。命名服务是一种以字符串为标识的服务，通过字符串和相应的组件服务相绑定，用户可以很方便的通过特定的字符串获得组件服务。

命名服务通过简单，友好的系统 API 函数，为系统服务以及用户组件提供了一套完整的 ezCOM 构件使用流程；它允许将本地服务，内核组件或者其它远程服务以命名服务的方式注册，极大的扩展了组件的可交互性；用户可以透明地获得服务，而无需考虑服务与自身的地址空间差别；在生命周期管理方面，命名服务亦同于普通的组件服务，所有适应于 ezCOM 构件生命周期管理的操作和语义亦适应于命名服务；相关组件服务动态更新

或者升级，不影响其它通过命名服务获取该服务的程序，其依然可以无需改动，无需重新编译地正常工作，其支持基于组件的动态更新和升级。

3.4.3 代码映射机制

“和欣”构件平台提供的应用程序接口是一套标准系统函数库，这些代码在编写系统扩展构件时会使用到，系统扩展构件的运行位置的不确定性要求在用户和内核空间都可以使用它们，“和欣”系统使用代码映射机制实现了这一要求。

“和欣”内核的部分代码段及部分数据段被映射到用户空间，使得可以在用户态执行内核代码，并以只读方式访问内核的部分数据。共享代码在映射以后，还需要用户程序在逻辑上知道这些代码的存在。“和欣”构件平台将系统的共享代码索引表生成一个虚拟的动态连接库，通过DLL(Dynamic Link Library)的导出机制，所有用户程序可以在运行时动态链接到这些代码，操作系统在加载用户进程的初始化时会自动在用户空间注册这个虚拟动态链接库，并将共享代码映射到用户空间。图 3.4 是代码映射机制示意图，它实现的是一份代码的多份映射。

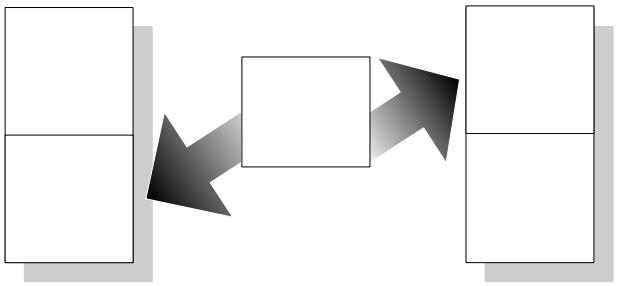


图3.4 代码映射机制

由于执行共享代码不需要在用户空间和内核空间之间切换，这种方式加快了关键性代码的运行，同时保证了足够的安全性。例如对于互斥锁、信号量等同步对象，它们的大部分代码都通过代码映射运行在用户态。由于常用的代码可以通过映射机制集中到内核，这种方法虽然在表面上增加了内核的尺寸，但实际上有效避免了由于应用程序静态链接运行库而导致整个系统尺寸的增加。

3.4.4 系统功能接口

和欣操作系统的系统功能是由内核中各个模块实现的，每个模块都管理各自的内核构件。“和欣”构件平台向用户公布了一些内核构件的接口，用户通过这些构件接口可以调用内核中实现的系统功能，这些接口总称为系统功能接口。

系统功能接口的作用与传统操作系统提供的系统调用是非常相似的。为了系统安全考虑，普通用户程序都是运行在低优先级状态，许多特权级指令是不允许运行的(多数硬件平台都有类似的设计)。用户程序只能在受控条件下间接使用这些指令，以完成特殊功能，系统调用就是出于这种目的而设计的。图 3.5 所示的就是系统调用[**Tane01**]的执行过程。

系统功能接口的使用与系统调用的使用有很大差异，系统服务接口的使用存在两种方式：系统扩展构件在内核中直接使用系统功能接口；在用户进程中调用系统功能接口。第一种情况，系统扩展构件能够直接获得内核构件的接口，可以直接调用接口方法；第二种情况，用户进程是不可能直接获得内核构件的接口。在获得系统功能接口时，“和欣”构件平台会自动在用户空间生成一个代理构件，并把代理构件的接口返回给用户。这个接口在形式上和需要获得的接口是一致的（虚函数表的尺寸一样），通过这个代理构件的接口用户可以陷入到内核调用系统功能，从用户空间调用系统功能接口的流程大致如下：

1) 调用代理构件的接口方法，用户线程陷入 IPC；
2) IPC 会把用户空间栈上的参数经过处理拷贝到内核空间的栈上，而且阻塞用户线程；
3) 启动一个内核服务线程，服务线程使用经过处理的参数调用真实构件实例的方法；
4) 调用返回后，服务线程回到 IPC；
5) IPC 处理返回参数和返回值，激活用户线程；
6) 用户线程调用返回。

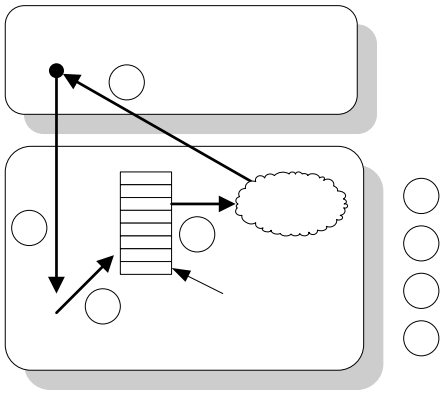


图3.5 系统调用的操作过程

以上两种情况，在系统扩展构件中都可能出现，“和欣”构件平台可以保证这两种情况都能运行正确。下面通过简单的示例说明，假如在系统扩展构件中有一段表 3.1 中所示的代码，其中 ZeeGetCurrentProcess 是“和欣”构件平台的一个 API，用以获取当前进程构件的接口，当系统扩展构件运行在内核中，iCurProcessRef 返回的是进程构件实例的 IProcess 接口的智能指针；当系统扩展构件运行在用户空间，iCurProcessRef 返回的是构件平台生成的代理构件的接口智能指针。

表 3.1 系统功能接口的使用

```
...
IProcessRef iCurProcessRef;
iCurProcessRef = ZeeGetCurrentProcess();
...
```

类似于 Linux 的一些传统操作系统支持动态装载内核模块[**Bove01**]，但是内核模块的开发不能使用应用程序的开发平台包括系统调用。和欣操作系统的系统功能接口却能支持用户程序和内核扩展模块，这完全得益于内核功能的构件化设计和系统级构件平台，另外系统功能接口也是实现灵活内核的主要技术之一。

3.5 和欣操作系统灵活内核的优点

和欣操作系统的灵活内核是构件技术和系统软件充分结合的产物，它

利用了构件技术这一先进的软件组织技术，使得操作系统的体系结构发生了根本性的变化。和通常研究的可扩展操作系统比较，它在许多系统性能方面都得到了极大的改善，主要表现在以下几个方面。

- **增强系统的灵活性** 在“和欣”灵活内核系统中，用户可以动态配置扩展构件的运行位置，让经过测试的可靠的扩展构件运行于内核态，从而获得较高的效率；把不够可靠的扩展构件置于用户空间。灵活内核不同于前面讨论的可扩展系统研究中的两种方法，其灵活性显然得到了很大提高。
- **提高系统扩展性能** 采用灵活内核技术不但使得系统功能可以扩展，而且扩展构件的接口设计也具有相当大的灵活性，只要接口形式符合一定的标准即可。这里主要是指接口方法的参数应能够被系统‘理解’，从而为其生成正确的中间件。一般研究中的可扩展系统，基本上要求扩展模块具有特定的接口形式。
- **更容易实现跨平台** 构件技术的出现本身就是为了在异种平台(包括软件平台和硬件平台)之间解决软件的互操作问题。“和欣”系统在系统级支持构件技术，为实现跨平台操作奠定了良好的基础。
- **提高开发工作效率** 可扩展系统中扩展模块的开发也是非常重要的问题。大部分可扩展系统为扩展模块设立单独的开发平台，开发系统扩展模块就需要学习新的开发平台。在“和欣”系统中，开发一个系统扩展构件和开发普通构件没有形式上的区别，这意味着在“和欣”系统上只需学习一套开发平台，提高了开发效率，同时构件的维护工作也会简单的多。
- **降低系统的复杂性** “和欣”系统灵活内核中的基本系统功能以构件技术封装，降低了系统模块之间的耦合，从而降低了系统内核的复杂性。另外，“和欣”内核中只实现最基本的系统功能，其它的系统服务能够以独立的扩展构件模块封装，这样也降低了系统内核的复杂性。

从工程应用的角度来看，灵活内核体系结构的技术优势也是非常明显

的。“和欣”系统的内核中只包含基本功能，因此内核的尺寸相对比较小，极其灵活的系统扩展特性使得系统定制变得更容易，因而灵活内核体系结构适合于许多嵌入式环境的应用；基于灵活内核的“和欣”系统易于开发，编写系统扩展构件和普通构件使用相同的开发平台，大大缩短了产品进入市场的时间。

3.6 小结

本章阐述了灵活内核模型，给出了灵活内核系统的基本准则，并且就灵活内核系统实现的一般问题进行了探讨。在此基础上，论述了和欣操作系统灵活内核的设计与实现，分别阐述和欣操作系统的主要特征、和欣操作系统的内核结构以及“和欣”构件平台，最后简单说明和欣操作系统灵活内核的意义。

第四章　系统扩展构件

具有灵活内核的和欣操作系统最显著的特点在于可以使用系统扩展构件的方式来实现系统功能的动态扩展，而且用户可以根据需要配置扩展构件的运行位置。这种功能的实现依赖于 ezCOM 构件自描述信息封装技术和“和欣”构件平台提供的构件运行时支持。

4.1 ezCOM构件自描述信息的封装

ezCOM 构件自描述信息的封装也是 ezCOM 对于微软 COM 技术的扩展之一，由于自描述信息的封装和 ezCOM 构件运行联系极为紧密，因此在本章阐述这部分内容。

4.1.1 技术基础

在 COM 规范中，强调构件和接口的自描述，以便于从二进制级上把接口与实现分离，并达到接口可以远程化的目的。但是微软的 COM 实现 COM 规范的过程中对于自描述信息的封装存在以下不足：

1) 在微软的 COM 中，为了保证构件能够正确运行，构件的一些相关运行信息必须在注册表中进行注册。ezCOM 构件技术进一步封装构件自描述信息，使构件运行无需借助类似于注册表的设施；

2) 微软的 COM 对构件接口的描述方法之一是使用类型库(TLB)元数据，类型库本身是跟构件的 DLL 文件打包在一起的。但类型库信息却不是由构件自身来解释，而是靠系统程序 OLE32.DLL 来提取和解释，这也不符合构件的自描述思想；

3) 大多数情况下，一个构件会使用到另一些构件的某种功能，也就是说构件之间存在相互的依存关系。微软的 COM 中，构件只有关于自身接口的自描述，而缺少对构件依赖关系的自描述。在网络计算时代，正确的构件依赖关系是构件滚动运行、动态升级的基础。

正是意识到微软的 COM 中存在的种种问题，ezCOM 在继承了 COM 构件自描述思想的同时，针对上述问题，对 COM 的具体设计和实现进行了扩展和改进。

4.1.2 基本思想

围绕着构件的自描述封装和运行，ezCOM 采用了如下的措施对微软 COM 进行改进和扩展：

1) ezCOM 把类信息(ClassInfo)作为描述构件的元数据，类信息所起的作用与微软 COM 的类型库相似，它是通过编译 CDL 文件获得，是 CDL 文件的二进制表述。与微软 COM 不同的是，在 ezCOM 中，可以使用一个特殊的 CLSID 从构件中提取元数据信息，构件元数据的解释不依赖于其它的 DLL 文件；

2) 在 ezCOM 的构件封装中，除了封装类信息，还对构件的依赖关系进行了封装。即把一个构件对其它构件的依赖关系也作为构件的元数据封装在构件中，我们把这种元数据称为构件的导入信息(ImportInfo)；

3) ezCOM 构件通过对类信息和导入信息的封装，可以实现构件的无注册运行，并可以支持构件的动态升级和自滚动运行。

4.1.3 构件的类信息

ezCOM 技术使用 CDL 文件描述构件，每个构件中一般都会定义一个或几个类，CDL 编译器在编译 CDL 文件时会生成所有类的自描述信息，这些信息称为类信息，之所以称其为自描述信息是因为只需通过这些信息就可以重新构建类的定义。在编译链接生成构件时，类信息会封装在构件的二进制映像文件中。

附录的示例中创建了 hello 构件，在编译生成 hello 构件时，镜像目录下会出现一个 C++文件 helloClsInfo.cpp，这个文件就包含了 hello 构件中所有类的类信息，这些信息最终会打包在 hello.dll 的资源段中。

类信息结构大致形式如图 4.1 所示，其中接口目录的条目如图 4.2 所示，类目录的条目如图 4.3 所示。图 4.2 和 4.3 分别展示了接口和类的所有信息。

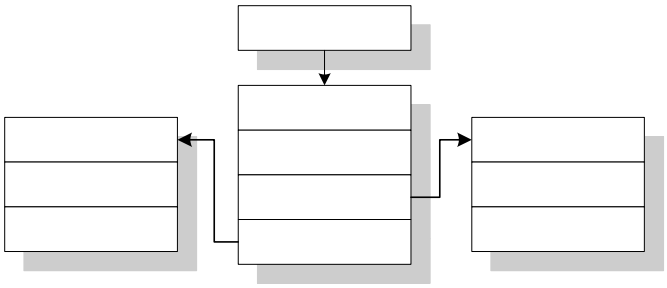


图4.1 构件的类信息结构

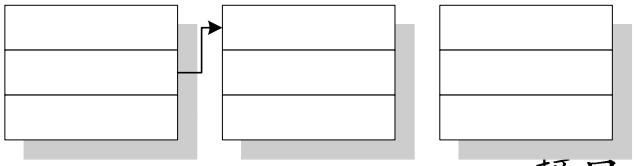


图4.2 接口的信息结构

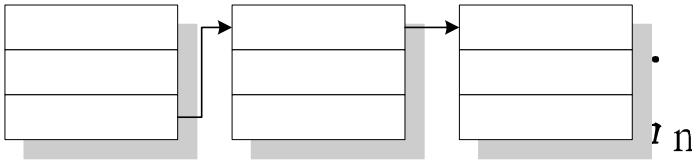


图4.3 类的信息结构

从前面的分析可以看出，如果只要能够获得类信息指针，即可获得构件的所有类信息，事实上 ezCOM 技术中正是采用这样的手段提取构件的类信息。ezCOM 构件都是以 DLL 的形式封装，每个 ezCOM 构件 DLL 都会导出函数 DllGetClassObject，使用这个导出函数可以很容易获得构件的类信息指针，“和欣”构件平台就是通过该函数获得 ezCOM 构件的类信息。

一般来说，普通用户根本不会使用到 ezCOM 构件的类信息，ezCOM 技术的初衷也是希望尽可能地掩藏用户无需了解的技术细节，从而简化面向构件的开发。“和欣”构件平台会在构件运行时使用这些类信息，提供 ezCOM 构件的运行时支持，本章后面会介绍“和欣”构件平台如何使用构

件的类信息。

4.1.4 构件的导入信息

微软的 COM 构件在使用之前必须注册。构件的客户端程序在使用构件时，系统通过注册表中的构件注册信息定位构件，并交付客户端程序使用。这种做法存在许多弊端：

- 系统在长期使用后，注册表会变得异常庞大且难以维护，注册表的查询效率自然会降低，导致构件的整个运行效率降低；
- 由于注册表是一个全局数据库，访问权限不容易控制，注册表容易成为病毒、黑客软件的入侵点，存在很大安全隐患；
- 构件间的依赖关系建立在完整安装的基础上，如：A 构件依赖于 B，当 B 没有安装到系统中时，由于系统没有任何关于 B 构件的信息，这时 A 构件也就不能正确运行了。

为了避免这些弊端，ezCOM 技术引入了构件导入信息的概念，使用导入信息能够实现构件的无注册运行，而且可以在没有安装的情况下正常运行构件的客户端程序。定义 ezCOM 构件的 CDL 语言规定每个构件必须指定它的 urn 属性，urn 属性就是构件导入信息的一部分。Urn 属性的值是一个字符串，通过它可以定位构件的网络位置。附录的示例中 hello 构件的 urn 属性值是 http://www.koretide.com/ezcom/ hello.dll，它指示 hello 构件的网络位置。

除了 urn 之外，构件导入信息还包括构件的版本号、类信息的版本号、最后修改日期和更新周期等，这些信息由 CDL 文件中相应的构件属性体现，使用这些信息可以对构件进行升级和错误恢复。

ezCOM 构件的 urn 信息封装在 EZCOM_CLSID 数据结构里，EZCOM_CLSID 是 ezCOM 技术对 COM 标准的 CLSID 进行扩展的产物。为了与 CLSID 兼容，EZCOM_CLSID 第一个域就是 CLSID，第二个域是指向 urn 的宽字符串指针。表 4.1 就是 EZCOM_CLSID 的 C/C++定义。

表 4.1 EZCOM_CLSID 的 C/C++定义

```
typedef struct EZCOM_CLSID {
    CLSID clsid;
    WCHAR *urn;
}EZCOM_CLSID;
```

导入信息是在生成构件的时候，封装在构件 DLL 文件中。确切地说这时这些信息还不能称其为导入信息，在创建构件的客户端程序时，开发工具会从构件中提取这些信息，并且把它们打包在客户端程序中，此时导入信息才算名至实归。

ezCOM 通过对构件 C/C++源程序的预处理生成构件导入信息。当构件客户端使用一个构件时，必须通过#import 预处理语句导入构件的定义，如附录中的示例所示。#import 语句的预处理程序是“和欣”SDK 提供的工具 mkimport，在“和欣”SDK 环境里，调用 C/C++编译器编译 C/C++程序之前会先调用 mkimport.exe 对 C/C++程序进行预处理。预处理完成后，编译链接工具对预处理后的程序进行编译链接，生成客户端程序。所有的导入信息也都打包在客户端程序中。

在构件客户端运行时，生成构件实例时会使用 EZCOM_CLSID 标识需要实例化的构件，“和欣”构件平台会根据 EZCOM_CLSID 中的 urn 信息找到并装载正确的构件，并根据真正的 EZCOM_CLSID 中 CLSID 找到相应的构件类。

在和欣操作系统上，所有的构件都存放在系统的构件缓存目录中，目录中的构件程序以 urn 为唯一标识。当客户指定的构件程序不在系统的构件缓存目录中时，和欣操作系统将自动根据构件的 urn 到网络上下载构件，并存放到系统构件缓存目录中。

4.2 系统扩展构件及其运行方式

4.2.1 系统扩展构件

系统扩展构件和普通的 ezCOM 构件在形式上没有任何区别，开发系统扩展构件和开发普通构件的方式是一样的，因此附录中开发 hello 构件的过程同样适用于开发系统扩展构件。

需要说明的是系统扩展构件虽然在形式上和构造上与普通构件没有任何区别，但是从内容上是它们应该有所不同，通常以下面的观点对它们进行区分：

- 系统扩展构件实现的功能具有一定的普遍性，即用户对这种功能存在着广泛的需求，而普通构件只属于个别程序使用；
- 系统扩展构件表达的内容与传统系统功能有紧密的联系，比如硬件设备的管理、系统资源的管理，而普通构件实现的功能一般与系统功能无关。

以上仅表示一般的区分观点，在和欣操作系统并没有这样明确的划分，之所以划分系统扩展构件和普通构件，主要是为了讨论的需要。但是从系统安全出发，系统扩展构件应该有别于普通构件。“和欣”系统对于构件的运行方式应该进行授权控制，毕竟构件运行于内核空间多少会影响系统的安全，在授权策略上可以对系统扩展构件和普通构件区别对待。然而现在和欣操作系统并没有这样的授权机制，所有的构件均能在内核空间运行。

4.2.2 系统扩展构件运行方式

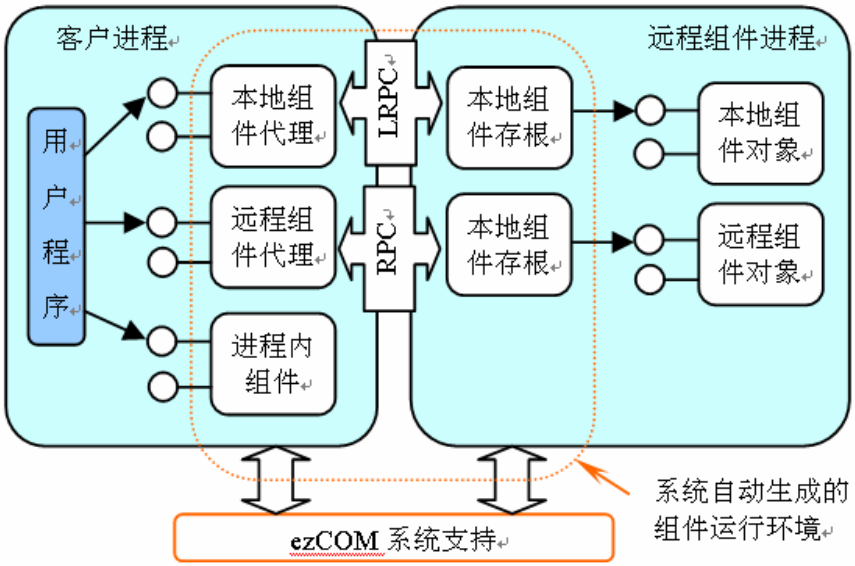
构件的运行总是与客户端程序紧密联系的，说明系统扩展构件运行方式最简单有效的方法是在应用程序中创建和使用构件实例，本节就采用这种方法说明系统扩展构件的运行方式。

根据应用程序与构件实例的相对运行空间划分，实际上系统扩展构件有三种运行方式：

1) 应用程序在本地空间创建构件实例；

2) 应用程序在内核空间中创建构件实例；

3) 应用程序在另一个用户空间创建构件实例。



“和欣”系统还没有实现对三种方式的支持，由于 2 和 3 很相似，因此这里只讨论 1 和 2 两种方式。在讨论时仍然以附录中 hello 构件为例。

在附录中，构件 hello 的 C++客户端程序使用 CHello 的智能指针创建了一个 CHello 的实例，这个实例就是本地用户空间的构件实例。由于构件实例和应用程序处于同一个地址空间，因此在调用接口方法时无需跨越地址空间，接口方法调用只不过是一个间址调用，比普通函数调用的复杂性相差无几。

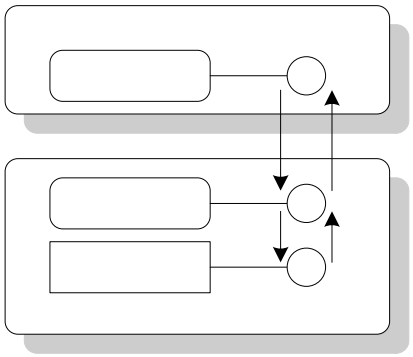


图4.4 调用内核构件接口

下面对构件 hello 的 C++客户端程序进行简单修改，给 Instantiate 方法添加一个参数 CTX_KERN_PROCESS，其它的代码不做任何修改。经过修改以后，这个程序就可以在内核空间创建一个 CHello 的实例，接口方法的调用过程会跨越地址空间。这时的接口方法调用是通过本地空间的代理构件和内核空间的存根构件完成的，代理构件和存根构件是“和欣”构件平台自动生成的。

图 4.4 是用户调用内核构件实例接口方法的示意图，图上标明的数字说明了方法调用的过程。客户代码首先发出调用请求，代理构件的相应方法被激发。代理构件的方法会通过 IPC 陷入内核，调用存根构件的相应方法。存根构件的相应方法会调用构件实例的对应接口方法，调用返回后存根构件又通过 IPC 回到用户空间，最后代理构件相应方法会返回到用户代码。

4.3 构件运行时支持

ezCOM 构件在“和欣”系统上的运行完全是建立在“和欣”构件平台运行支持环境之上，运行支持环境只是基于功能的抽象概念，它提供构件在运行时的所有支持功能。

相对于 ezCOM 构件的编写和使用，ezCOM 构件的运行是比较复杂的。从创建构件实例到接口方法调用无不涉及“和欣”构件平台，虽然按照构件运行的时序介绍构件运行时支持比较符合一般思维，但是这样会过于零散和琐碎，而且有些功能是互相交错的，容易混淆，因此还是按照功能介

绍。

4.3.1 创建 ezCOM 构件实例

在应用程序中，无论创建内核还是本地构件实例，系统都需要定位构件的位置，将其作为模块装载到内核中。通过模块的标准导出函数创建类厂，使用类厂创建构件实例。这些动作全部是由构件平台完成。下面按顺序说明。

定位构件

用户创建构件实例时，可以使用类智能指针的 Instantiate 方法，也可以使用标准 API 函数 CoCreateInstance 和 CoCreateInstanceEx。实际上前者最终也是调用后面的两个 API 函数，因此，下面的讨论就从两个 API 出发。

CoCreateInstance 和 CoCreateInstanceEx 都会通过参数指定 CLSID。前面章节已经介绍过 ezCOM 的 CLSID 实际上是 EZCOM_CLSID。EZCOM_CLSID 中包含 urn，使用 urn 的信息在“和欣”系统的构件缓存目录查找相应的构件。如果存在，定位构件的动作完成；如果不存在，构件平台会调用网络功能到 urn 指定的网络位置下载该构件，下载成功后构件会保存在构件缓存目录，定位构件的动作结束。但需要注意，如果要在内核空间创建构件实例，CoCreateInstance 或 CoCreateInstanceEx 会先陷入内核空间，在内核中再次调用 CoCreateInstance 或者 CoCreateInstanceEx。

装载构件

系统功能接口提供了进程构件接口(IProcess，是系统功能接口之一)。使用进程构件接口的 LoadModule 方法可以装载模块到相应进程的地址空间。在定位到构件后，“和欣”构件平台会使用当前进程的进程构件接口装载构件缓存目录中的相应构件到当前进程地址空间。

创建类厂

在模块装载到进程空间后，LoadModule 会返回该模块的模块接口(IModule，是系统功能接口之一)。使用这个接口的方法能够获取该模块的标准导出函数 DllGetClassObject。调用 DllGetClassObject 创建类厂。还需要说明一点，在创建类厂之前，系统还会使用 DllGetClassObject 获取类信息指针，然后把类信息注册在系统类信息注册表中。类信息注册表是系统启动以后在内存中动态建立的，这些类信息会在接口远程化时使用。

创建构件实例

前面创建了类厂，使用类厂的标准接口就可以创建构件实例，到此为止，“和欣”构件平台为用户创建构件实例的任务正确结束。如果 CoCreateInstance 或者 CoCreateInstanceEx 是在内核空间被调用的，构件实例自然在内核空间，否则构件实例在用户空间。

4.3.2 接口远程化

接口远程化是指获取另外一个地址空间构件实例接口的过程。和欣操作系统还不支持从一个用户空间到另外一个用户空间创建构件实例，所以这里只讨论从用户空间获取内核空间构件实例的接口。

不失一般性，下面通过 CoCreateInstance 的使用讨论接口远程化。用户空间直接或间接使用 CoCreateInstance 在内核创建构件实例时，前面已经提到，CoCreateInstance 会先陷入内核，其实这个陷入过程是通过一个特殊接口 IEzComSys。这个接口所属的构件是 ezCOM 基础设施之一，IEzComSys 的用法与系统功能接口类似，只是这个接口并未公开。

陷入内核后，IEzComSys 会再次调用 CoCreateInstance，用户空间调用 CoCreateInstance 函数时的参数会被 IEzComSys 带入内核，并且传递给 CoCreateInstance，CoCreateInstance 完成构件实例的创建。在用户调用 CoCreateInstance 时不但指定了 CLSID，还指定了 IID，构件实例创建成功后会返回 IID 指定的接口的指针。

IEzComSys 接口带着 IID 指定的接口指针准备返回，在返回用户空间

之前，系统会对需要返回的接口指针进行列集。列集的过程是在内核空间生成一个存根构件，存根构件会记录对应构件实例和 IID 对应的接口指针，而且系统会为存根构件生成一个标识 IPID。IEzComSys 会带着这个 IPID 而不是接口指针返回到用户空间，返回后系统会对 IPID 进行散集。散集的过程是在用户空间生成一个代理构件，代理构件会保存 IPID 和 IID。散集完成后，CoCreateInstance 把代理构件的接口交给用户。在接口的散列集过程中都要使用类信息，代理构件和存根构件都有一个接口，它们与需要散列集的接口的形式一致。这里的‘一致’是指它们具有相同个数的方法，生成代理存根构件时，系统会根据 IID 在类信息注册表中提取接口方法个数的信息。至此接口的远程化就完成了，实际上接口远程化的过程是非常复杂的，大多数情况下只能借助阅读系统实现代码才能了解细节，这里的介绍主要是原理性的，并不希望涉及所有细节。

在接口远程化后，用户可以调用使用代理构件的接口，代理构件的方法带着 IPID 通过 IPC 陷入内核，然后调用 IPID 对应的存根构件，存根构件调用它所对应的构件实例。

接口的远程化应该属于下面将要讨论的自动化散列集的一部分，但是它的过程具有特殊性，因此单独讨论。

4.3.3 自动化散列集

散列集为访问远程接口提供了重要支持。简单地说，散列集是调用远程接口方法时，对方法参数进行必要的处理的过程，经过处理后参数才能在不同的地址空间进行传递。

讨论散列集的前提是已经获取了一个远程接口，因此假设在用户空间已经存在一个内核构件实例的接口。当调用这个远程接口的方法时，实际是代理构件的相应方法被调用，代理构件的方法执行的功能是进入 IPC。在 IPC 中，构件平台可以根据代理构件得知 IID 和方法在接口中的序数(接口方法是接口的第几个方法)。然后构件平台使用 IID 和方法序数在类信息注册表中提取相应方法的参数信息(参数个数和参数类型)。根据获取的参数信

息，构件平台把用户空间栈上的参数进行处理并传递到内核空间中。然后IPC 会从用户空间切换到内核空间，执行方法调用。方法调用完毕，在回到用户空间之前，构件平台会把返回参数和返回值经过处理后传递回用户空间。这整个过程经过两次散列集，第一次是把参数从用户空间经过处理写到内核空间，第二次是把返回参数从内核空间经过处理写回用户空间。参数的处理必须使用参数信息，因为只有通过参数信息，构件平台才可以掌握参数的内存布局，从而对参数进行正确处理。还应强调一点，接口的参数必须是自描述数据类型，只有自描述数据类型才能自己描述自身的内存布局，ezCOM 技术中自描述数据类型的意义就是在这里体现。

上面的散列集过程并不适用于所有类型的参数，接口指针使用特殊的散列集过程——接口远程化过程。另外，这一节题目中‘自动’的意义是接口方法参数的散列集过程不用借助外部信息和外部手段，只要构件和系统就可以完成全部工作，在这个过程中它们的作用分别是：构件提供类信息；构件平台使用类信息。

“和欣”构件平台的散列集具有很强的特殊性，它和微软 COM 的散列集有很大不同，这一点可以参看微软 COM 的散列集机制[**Box98**]。微软 COM 的散列集大概过程是在客户端把参数打包，然后通过共享内存或者其它方式的进程间通信，把参数包传递给服务端，在服务端进行参数解包。其中参数打包称为列集，参数解包称为散集。

4.4 “和欣”系统的系统扩展构件

和欣操作系统不但提供了“和欣”构件平台，还以构件库的形式向用户提供了许多系统扩展构件，增强了和欣操作系统的可用性。其中典型的有图形系统构件库和文件系统构件库。图形系统和文件系统都是比较复杂的操作系统子系统，使用 ezCOM 技术实现这两个系统的确很好地考校了ezCOM 技术和“和欣”构件平台的能力。

4.4.1 文件系统服务构件

和欣操作系统的文件系统居于系统平台之上，以构件服务的方式提供给用户。具有虚拟文件系统层，提供了对各种不同文件系统进行操作和管理的统一框架，具有对多文件系统支持的扩展性和灵活性。和欣操作系统的文件系统还为用户提供了一套统一的文件操作接口，通过这套接口，用户无须关心具体文件属于什么文件系统以及具体文件系统的设计和实现。在和欣操作系统平台上，可以将 TFTP 文件系统和 msdos 文件系统“安装”到和欣系统平台上，用户程序却可以完全按照相同的方式来访问这两个文件系统。（目前和欣操作系统平台支持 TFTP 文件系统和 msdos 文件系统）

4.4.2 网络服务构件

Elastos 的网络构件，它分为三部分：

1) 硬件以太网卡驱动，运行在内核态,以接口方式向网络协议栈提供基本的服务；

2) 协议构件运行在用户态，在驱动提供的服务的基础上，基于接口(类 sock 功能)向用户提供服务，用户可自己实现一套网络协议构件，类替换系统的网络服务构件；

3) 系统在网络构件提供的接口服务基础上，以 winsock.lib 向应用程序提供连接库，连接了该库，应用程序就可以使用 winsock 的方式进行网络编程，或作简单修改，移植已有的网络应用。

4.4.3 图形服务构件

“和欣”图形系统为用户提供了各种图形构件。目前包括窗口构件和绘图构件，使用这些构件可以快速的开发各种图形应用。

窗口构件提供了操作控件，处理事件，菜单的各种方法。用户可以使用该构件轻松的创建出各种控件，改变它们的颜色，位置等。利用该构件提供的事件机制，用户可以注册处理自己所关心的事件。

绘图构件提供了绘制图形，图像剪裁和显示字体的各种方法。用户使

用该构件可以进行画点，画线，绘制位图，使用各种字体等。

4.5 小结

本章从动态运行的角度论述了和欣操作系统体系结构的运作机制。首先阐述了 ezCOM 构件自描述信息的封装技术，其次再讨论系统扩展构件运行方式，以及“和欣”构件平台如何使用自描述信息为构件提供运行时支持，最后，简单介绍了和欣操作系统提供的系统扩展构件。

第五章　构件化设备驱动模型

和欣操作系统的实现过程充分利用了 ezCOM 构件技术，比如前面论及的内核构件、系统扩展构件和构件平台等。同时，和欣操作系统把 ezCOM 技术也引入到硬件设备管理和设备驱动管理中，从而实现了构件化的设备驱动模型[**Du03**]。

5.1 技术基础

设备驱动是现代操作系统处理硬件设备的重要手段，它负责直接访问硬件，并且向系统的其它部分和用户提供设备无关的操作接口。至于设备驱动如何获得硬件资源、以何种方式向外提供服务以及系统如何管理硬件资源，这些都属于设备驱动模型的范畴。

设备驱动模型是一个抽象概念，它的功能主要包括：

- 搜集硬件系统中的设备信息。这些设备信息可以是通过硬件本身的功能检测得到，如 PCI 总线上的设备，也可以是通过手工配置的方式，把设备信息添加到系统中；
- 对系统中的设备进行标识，通过设备标识能够把设备与对应的驱动程序进行匹配。如：在传统的 Unix 系统中，通过主设备号和次设备号来标识设备并找到其相应的驱动程序入口；
- 在上层应用需要访问硬件设备时，操作系统负责激活其相应的驱动程序，通过驱动程序为上层应用提供硬件访问接口；
- 协调设备驱动程序的运行，管理共享硬件资源，检测及报告资源冲突，并能够为设备驱动提供设备信息服务。

这些功能连同它们在系统中的实现方式总称为设备驱动模型。在现代操作系统中，经典的设备驱动模型首推 UNIX 系统的驱动模型。

在 UNIX 操作系统上，应用程序通过使用系统调用接口陷入内核访问

I/O 子系统，内核的 I/O 子系统接受这些请求，然后利用设备驱动程序接口访问和操作硬件设备。图 5.1 展示了 UNIX 设备驱动程序在系统中的作用。

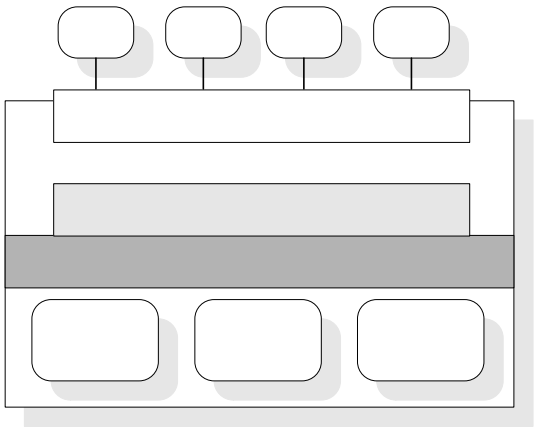


图5.1 UNIX设备驱动程序的作用

UNIX 系统中的驱动模型和 I/O 子系统的设计实际上是提供了统一的高级文件视图，所有设备处理都使用文件处理的方式。内核可以对所有设备类一视同仁，通过相同的驱动接口访问设备，驱动程序接口是由设备开关表定义。每个驱动程序提供了设备开关表的特定于某个设备的实现。UNIX 系统的驱动程序模型具有一定的先进性，正如[**Vaha96**]中所说，“从面向对象的角度来说，驱动程序接口是抽象的基类，每个驱动程序是一个子类，或者是基类的特殊实现”，但是它也存在一定的缺陷和局限性。

驱动程序的统一接口抹杀了硬件设备的多样性，打印机和鼠标是两种截然不同的硬件，具有不同的物理特性。在 UNIX 中它们却同属于字符设备，都要实现字符设备开关表中的功能。单从接口角度很难理解同一种方法在这两种不同设备上会产生什么效果，换言之，根本不可能从接口原型获取设备操作的丝毫概念，要使用具有这样接口的驱动程序就必须仔细阅读驱动程序的用户指南。

UNIX 系统的设备驱动一般都提供端口控制操作 d_ioctl，它把端口操作直接交由用户来控制，导致硬件设备在用户面前暴露无遗，其后果造成用户程序、驱动程序和硬件设备之间的紧耦合，不利于用户程序的移植。

另外，传统操作系统的设备管理还存在下述两个问题：

1) 无论是 Unix 还是 Windows 操作系统，驱动程序总是与操作系统绑定在一起，操作系统启动时加载所有的驱动程序，而不管用户会不会使用到这个驱动程序。这样既耗费 CPU 资源，也耗费内存资源；

2) 由于驱动程序运行在操作系统核内，驱动程序可以使用的资源受到限制。如：不能访问标准输入输出设备，不能使用一些在用户态可以使用的许多标准函数及系统调用。这些限制给驱动程序的开发带来了极大的不便，驱动程序员需要熟悉一大套与应用开发不一样的开发接口及编程约定，增加了编写驱动程序的复杂度。

将构件技术运用到设备驱动程序设计中的尝试还是相当鲜见的，但的确有人已经这么做过。Steve Maillet[**Mail00**]曾经提出了在嵌入式系统上运用 COM 技术封装设备驱动，但所涉及的内容也仅止于此，并没有从系统管理的层面来讨论 COM 技术封装设备驱动程序的意义及实现的方法，

和欣操作系统的设备驱动模型一改传统操作系统的设备管理模式，采用 ezCOM 构件技术封装设备驱动，设备驱动程序的运行可以直接利用“和欣”构件平台提供的构件运行支持环境，而且驱动管理模块在管理设备驱动时也充分利用了通用构件管理的手段。

5.2 基本设计

“和欣”驱动程序模型从逻辑上划分为三部分：驱动构件客户端程序、设备驱动构件和设备管理器，设备驱动模型主要是描述这三部分之间的协作和每部分的责任要求，下面分别讨论。

驱动构件客户端程序可以是普通用户程序也可以是系统程序，它负责向设备管理器发出创建驱动构件实例的请求；在获得驱动构件接口后，通过调用接口方法，获取构件实现的服务，也就是访问和操作硬件设备。

和欣操作系统的设备驱动程序使用 ezCOM 构件技术封装，因此也被称为设备驱动构件，简称驱动构件。驱动构件主要的工作是根据客户端程序

的请求，操作硬件设备。

驱动构件实现的接口分为两类：一是系统接口，是系统定义的标准接口，由操作系统调用；二是用户接口，由驱动构件的开发者定义，并由驱动构件的客户端程序调用。驱动构件实现两个标准的系统接口 IDriver 和 IDriverISR(可选)，这两个接口对于构件客户端程序是不可见的，只能由设备管理器使用。

设备管理器(Device Manager)是“和欣”设备驱动模型的核心部件，它同时也是一个内核构件，主要负责管理系统中的所有硬件设备和设备驱动构件。设备管理器的具体功能设计如下：

- 设备管理器收集、管理系统中所有硬件设备的信息，为每一个设备建立一个设备节点，并使用设备标识与设备号来唯一标识这些设备；
- 把驱动构件对象看作设备的一个属性，当且仅当设备节点的驱动构件实例被创建出来后，这个设备节点才被称为激活了的设备或活动设备；
- 设备管理器负责创建驱动构件实例，并使用设备标识、设备号来匹配设备节点与驱动构件实例；
- 设备管理器负责销毁驱动构件实例，释放与构件实例相关联的设备节点。

设备管理器向驱动构件实例提供 IDeviceInfo 接口，驱动构件实例可以使用该接口中得到与自己相关联的硬件设备信息；设备管理器还提供 IDeviceEnumerator 和 IDeviceManager 接口。IDeviceEnumerator 用于枚举系统中所有的设备信息及其状态；设备构件客户程序可以使用 IDeviceManager 中的 CreateDriver 方法来创建驱动构件实例。

图 5.2 是“和欣”构件化设备驱动模型的基本原理示意图，从图上可以清晰地看出设备驱动模型的三部分以及它们之间的基本协作关系。

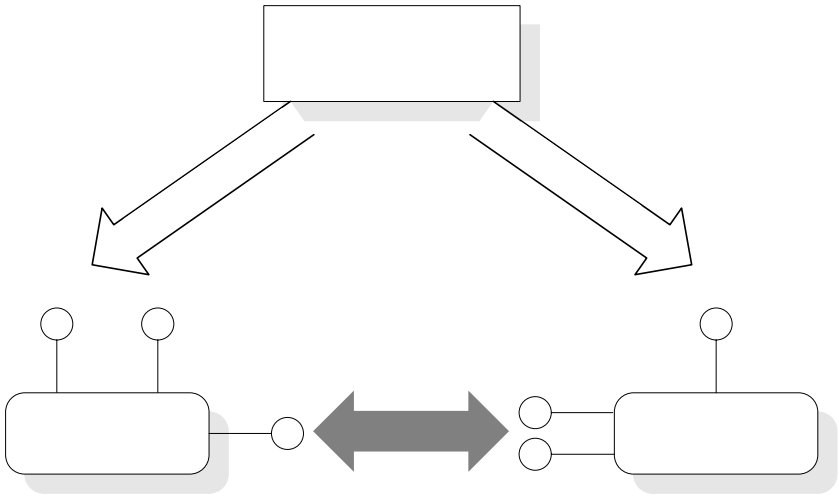


图5.2 构件化设备驱动模型基本原理

创建驱动构件实例

5.3 实现及工作原理

5.3.1 设备标识

IDeviceManager

IDeviceEnumerator

驱动程序的目的是驱动系统中的硬件设备，驱动程序的运作必然要与设备相关联。要让驱动构件实例与设备正确的关联，系统中的每个设备都需要有系统全局的唯一标识。和欣操作系统使用设备标识(DeviceId)和设备号(DeviceNo)来标识设备。

设备管理器

设备标识是一个字符串，它用于标识不同类型的设备。例如：串口的设备标识是”COM”；IDE 控制器的设备标识是”IDE”。设备号是数字形式的，用以区分系统中多个相同类型的设备，设备号从 1 开始编号。如第一个串行口的设备号为 1，第二个串行口的设备号为 2，以此类推。根据上面的规则，系统中的每个设备都可以使用一个设备标识连同一个设备号进行标识。

IDeviceInfo

5.3.2 驱动构件开发

本节的内容并不是讨论驱动构件的开发细节，也不是提供一个完整的开发的示例，而是讨论驱动构件开发过程中与设备驱动模型相关的内容。

与普通构件的开发一样，驱动构件的开发也是从编写 CDL 文件开始，表 5.1 是一个鼠标驱动构件的 CDL 文件样例，文件名是 mouse.cdl。

表 5.1 鼠标驱动 CDL 文件样例

```
[
    version(1.0), uuid(4f4e05dc-b1ec-4aa5-9af8-e7a54708396c),
    urn(http://www.koretide.com/ezcom/samples/mouse.dll)
]

component Mouse
{
    importlib("catman.dll");

    [ uuid(18099f72-78de-466f-b4a6-58a862935baf) ]
    interface IMouse {
        HRESULT GetMouseEvent( [out]PUINT px,
                               [out]PUINT py,
                               [out]PUINT pKeyType);
    }

    [ uuid(7770599a-a61f-425b-864b-32e058abe485) ]
    category CatMouse {
        interface IMouse;
    }

    [ uuid(80e44d0b-3ec6-4d56-9a75-3218e84e6bfb),
        driver]
    class CMouse : CatMouse {
    }
}
```

这个文件中定义了一个驱动构件类 CMouse，它和普通构件类有一点不同，它的类属性中使用了 driver 关键字，这个关键字标明它不是普通构件类，而是一个驱动构件类。

使用 driver 关键字意味着 CMouse 类自动继承了两个标准系统接口 IDriver 和 IDriverISR，驱动构件开发者最终要实现这两个接口。IDriver 接口是设备驱动构件向设备管理器提供的交互接口，IDriverISR 是设备驱动中断服务例程的入口。

对有些不需要中断服务程序的设备驱动而言，可以使用 noisr 关键声明

驱动构件类，声明方式是”driver(noisr)”。使用这样的声明方式，构件类就不会继承 IDriverISR。

类似于附录中的方式，使用 zmake 命令编译 mouse.cdl 文件就可以生成驱动程序的 C++源代码框架，分析源代码中 CMouse 类的声明，可以看到 driver 关键字产生的效果。

下面的工作就是驱动开发者填写实现代码，需要说明一点，实现驱动所需要的开发环境与普通构件的开发环境没有区别，使用的编程接口也一样。唯一不同的是，在中断服务例程中不能使用屏幕输出函数。

5.3.3 驱动构件实例的创建和销毁

驱动构件的客户端程序可以使用类智能指针的 Instantiate 方法来实例化一个驱动构件对象，Instantiate 方法实质是通过内核构件设备管理器来创建驱动构件实例，Instantiate 会调用 IDeviceManager 接口的 CreateDriver 方法。图 5.3 是设备管理器创建驱动构件实例的流程图。

与普通构件一样，一旦客户端不再使用驱动构件实例，并且系统中没有其它引用到驱动构件实例的地方，即驱动构件实例的引用计数变为零时，驱动构件实例的 Release 方法将把自己从内存中删除。编译工具为驱动构件自动生成的 Release 方法在删除自己之前会首先调用驱动构件 IDriver 接口的 CanUnloadNow 方法，以确定驱动对象能否立即被删除，最后调用设备管理器的 IDeviceManager 接口的 UnRegisterDriver 方法通知设备管理器，以便设备管理器能够正确地维护和管理设备信息。

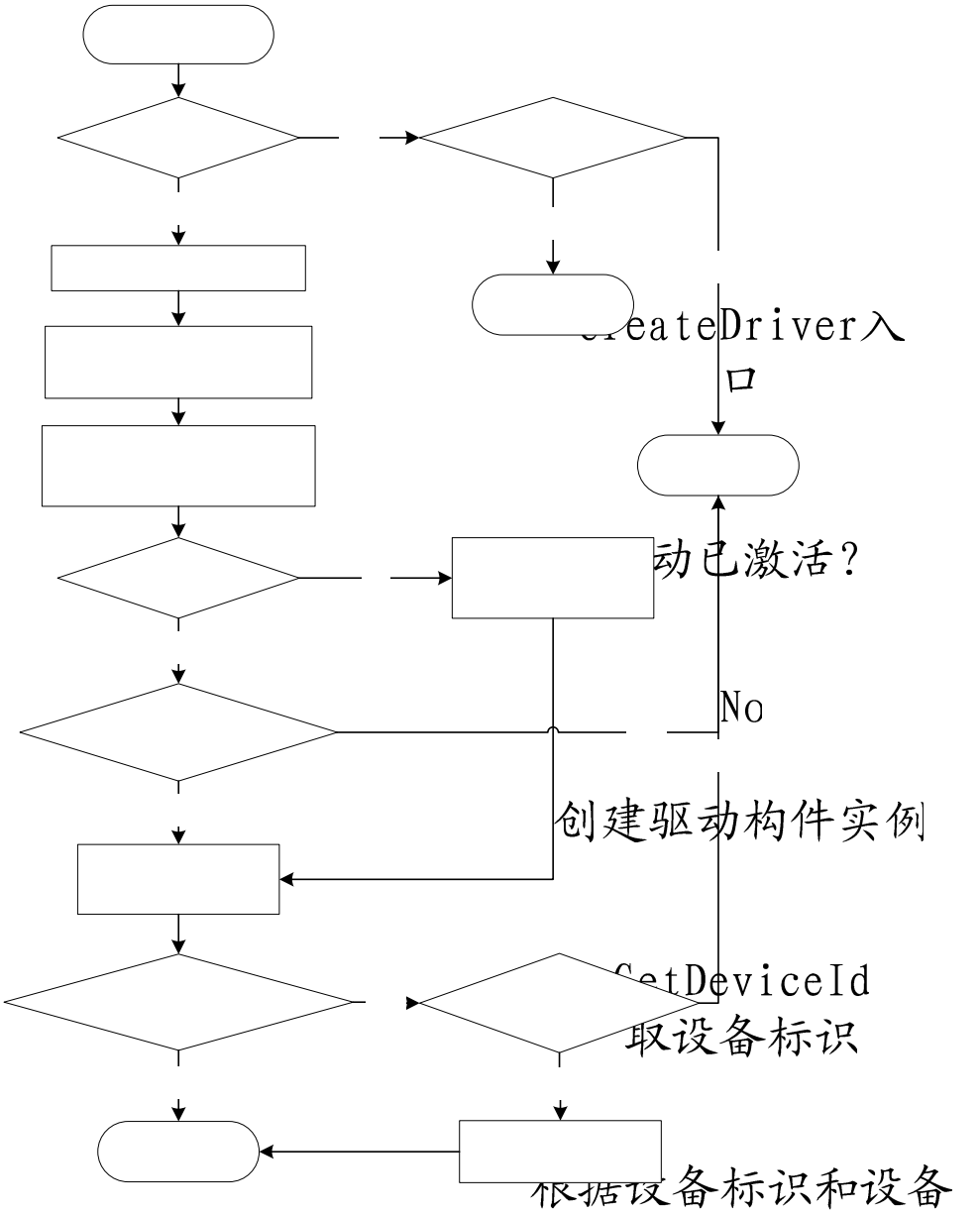
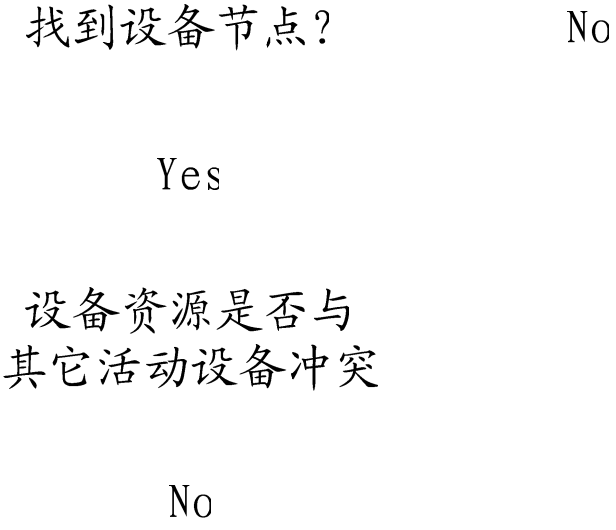


图5.3 创建驱动构件实例

图 5.4 是调用驱动构件 Release 方法的流程图。



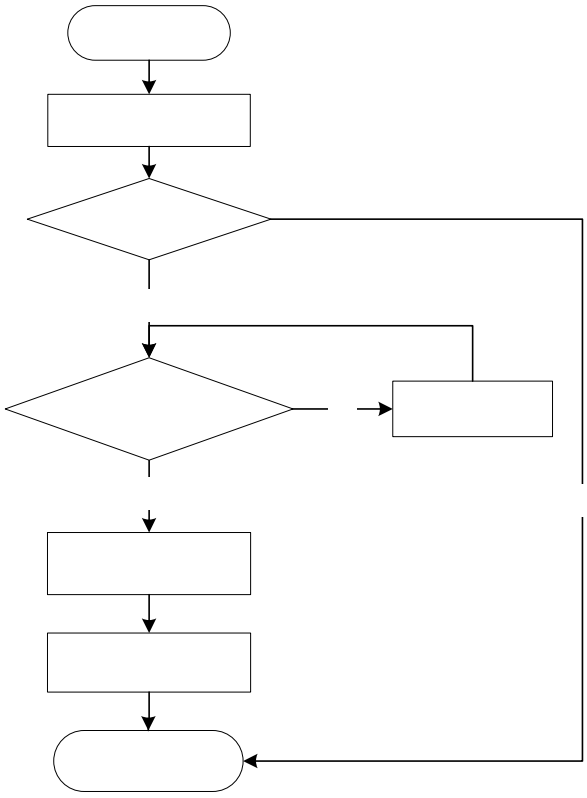


图5.4 销毁驱动构件实例

5.3.4 设备硬件资源的冲突检测

在创建驱动构件实例的过程中，设备管理器需要检验驱动构件实例申请占用的设备信息是否与其它设备冲突，检测内容包括中断号、I/O 地址、DMA 通道号等。

和欣操作系统的驱动程序是可以动态装载和卸载的，因此设备管理器确定资源是否冲突的方法是动态的，也就是把新创建的驱动构件实例申请的设备信息与其它活动设备(驱动构件实例已经被创建的设备)占用的信息进行比较。

这种动态资源冲突检测的优点在于：一些系统硬件资源可以复用。只要两个设备的驱动构件不会同时启用，就可以使用相同的硬件资源。如：系统上声卡使用 DMA 通道 1，光驱也使用 DMA 通道 1，但只要不同时使用声卡和光驱，就不会出现资源冲突。

5.3.5 常驻型驱动构件的实现

系统中的有些设备频繁地被使用，如硬盘和网卡等，我们当然不希望这类设备的驱动构件实例一再地被创建和销毁，这样显然会影响系统效率，因此需要一种能够支持常驻型驱动程序的机制。

解决的方法是：定义驱动构件时，在驱动构件类的属性中使用 resident 属性，声明的形式如：driver(resident)或 driver(resident, noisr)。驱动构件类使用了上述的声明方式后，编译工作自动生成的 Release 方法中就不会调用驱动构件的 CanUnloadNow 方法，也不会在引用计数变为零时销毁构件实例。

在常驻驱动构件实例的引用计数变为零时，Release 方法会把这个消息通过设备驱动管理器 IDeviceManager 接口的 NotifyNoReference 方法通知设备管理器。设备管理器会把这个驱动构件实例置为非激活状态，在下次驱动构件的客户端创建同样的驱动构件实例时，设备管理器会直接将已经存在的驱动构件实例的接口返回，并将驱动构件实例的状态置为激活态。

5.3.6 设备信息的收集

“和欣”驱动构件实例是通过设备管理器创建的，并且设备管理器要向驱动构件实例提供设备信息，因此设备管理器必须事先负责收集系统中的各种设备信息。

设备管理器通过两种渠道收集设备信息：一是从系统标准的设备配置文件读入设备信息；二是通过通过动态检测的方式获取设备信息。设备管理器把得到的设备信息以链表形式存放以便于检索和使用。

设备配置文件中以文本方式记录了设备的配置信息，这个文件的内容可以人为的编辑和修改。表 5.2 是串口配置信息的一种可能形式。

表 5.2 串口配置信息

```
[ COM#1 ]
IRQ = 4
IO = 0x3f8-0x3ff
```

NAME = Communication Port(COM1)
PROVIDER = Standard port types
LOCATION = PCI Bus

在和欣操作系统启动时，有一个专门的设备配置文件读入程序会把设备信息从文件中读出并传给设备管理器。

设备管理器获取设备信息的第二个渠道是通过设备枚举器。设备枚举器也是驱动构件程序，它使用硬件检测的方式收集设备信息，设备管理器可以使用它的 IDeviceEnumerator 接口读出收集到的信息。

典型的设备枚举器是各种总线及控制器驱动。如 PCI 总线驱动，通过其枚举器接口可以得到 PCI 总线上自动检测到的所有设备信息。

5.4 构件化设备驱动模型的意义

和欣操作系统的构件化设备驱动模型与系统其它部分的设计保持了高度的一致。这种高度一致性意味着：程序员可以使用统一的构件编程模式在和欣操作系统上实现应用、设备驱动和系统扩展构件；构件化设备驱动模型仍然是建立在“和欣”构件平台的基础之上，在一定意义上，可以说简化“和欣”系统的设计。

构件化驱动程序模型充分地利用构件技术的高度模块化特征，构件化驱动程序由独立的目标模块承载就体现了这一点。更重要的是驱动程序与外部的交互完全通过接口进行，保证了驱动程序实现细节不会影响驱动程序的使用程序，必要时系统可以根据需要动态地置换和升级驱动程序。

5.5 小结

本章论述构件化设备驱动模型以及该模型在和欣操作系统上的实现方式。构件化设备驱动模型是一套完整的设备管理方案，其中采用构件封装驱动程序，并利用类似于构件管理的方式管理设备驱动。和欣操作系统上实现的构件化设备驱动模型逻辑上可划分为三部分，文中明确了每部分的职责，并且展示了驱动模型的工作原理和基本设计。

第六章　系统性能分析

和欣操作系统采用了灵活内核模型，实现的过程中采用了 ezCOM 构件技术，以便增强系统的可扩展性、可剪裁性和可配置性等特性。虽然和欣操作系统引入了构件技术，但是在系统内核的最底层仍然承袭了许多传统的概念，如调度策略和内存管理等，在这些方面，和欣操作系统并没有更多的创新，而是沿用一些传统实现，因此在这些方面也没有体现出更好的性能。和欣操作系统的优越性更多地是体现在系统架构上，这种架构不仅使得系统的获得较好的特性，同时也为未来的应用提供了一个易于使用的平台。本章将从理论和测试数据方面分析和欣操作系统的总体性能，并阐述了和欣操作系统针对嵌入式系统在设计和实现方面的一些考虑。

6.1 嵌入式系统的考虑

嵌入式系统是一种用于对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统，它以应用为中心，以计算机技术为基础，且软硬件可裁剪。嵌入式系统的硬件一般包括处理器、存储器、外设器件和电源等。由于受到成本、功能和体积等诸多因素的限制，嵌入式系统具有如下特性：1. 内存空间有限，通常在几百 K 到几兆之间；2. 不同的嵌入式应用硬件设备差异明显；3. 通常嵌入式环境中要求较好的实时性和稳定性；4. 软硬件设计需要考虑节省功耗。

和欣操作系统目标是成为一个面向因特网的嵌入式操作系统，因此在设计和实现的过程中充分地考虑了嵌入式环境的需求。针对嵌入式系统的特性，和欣操作系统在设计和实现中主要做了以下一些着重考虑：

- 和欣操作系统的内核只实现了最基本的系统功能，从而保证内核的尺寸在一般嵌入式系统能够接受的范围。在前面的章节已经介绍，和欣的内核主要实现了内存管理、进线程管理、模块管理、进程间通信和内核调试器等功能，在实现这些功能的过程中，也进行了代

码优化，现在内核的尺寸是 117K(arm 平台)，基本系统功能的运行也只需要 1M 以内的内存。另外，和欣操作系统所支持的构件是“按需”运行的，也就是只有在构件对象被创建时构件模块才会装载到内存中，在使用完毕后构件还会被卸载，并释放占用的内存。传统的动态链接库则是随主程序一起装载到内存中，无论动态链接库的功能是否被主程序调用，它都会占用内存。

- 不同的嵌入式应用之间硬件差异非常普遍，“和欣”系统上设计实现的构件化驱动模型能够很好地解决这个问题。和普通构件的运行一样，构件化驱动模型中的驱动构件也是“按需”运行的，如果不是经常使用的驱动，完全可以在使用时装载，在使用之后就卸载。构件化驱动模型有效地将系统开发和驱动开发独立分开，提高了系统适应不同硬件设备的能力，也提高了驱动开发的速度。
- 在实时性方面，和欣操作系统实现了基于时间片的轮循调度加优先级调度算法，具有实时性要求的任务可以定义为高优先级任务，而且还支持设置特殊的实时任务，这种实时任务在执行结束之前系统不进行任务调度。在稳定性方面，“和欣”系统的构件运行机制支持将稳定性有疑问的构件运行在用户空间，类似于微内核的形式。对于稳定性良好的构件服务可以运行在内核空间，从而获得更好的效率。
- 许多嵌入式应用(如手持类的消费电子产品)都存在节省功耗的问题，虽然和欣操作系统的内核并没有直接提供节省能耗的功能，但以驱动形式存在的电源管理模块已经在设计和实现，由于电源管理模块还不成熟，本文没有过多的涉及。

在早期的嵌入式系统中很少使用类似于和欣操作系统这样功能全面的操作系统，由于资源的限制，那时的嵌入式系统中的软件只是简单的控制程序。和欣操作系统不是针对这样的嵌入式系统设计的，它的使用主要限制在比较高端的嵌入式系统中，因此设计和实现也主要是针对高端的嵌入式系统来开展的。

6.2 技术分析

和欣操作系统采用了灵活内核模型，为了进行系统技术对比和分析，本文选择了在嵌入式领域先进的操作系统 VxWorks AE 1.1 和 QNX RTOS v6.2 作为参考对象。

VxWorks AE 1.1 源自于 VxWorks 5.x 系列，它使用了整体内核的系统架构，其主要特点是采用了保护域的概念替代传统的进程概念。

QNX RTOS v6.2 是典型的微内核结构，由微内核和可选的协作进程构成整个系统，它是一个基于消息的操作系统，所有的跨进程调用都是通过消息机制实现。

和欣操作系统、VxWorks AE 1.1 和 QNX RTOS v6.2 之间的系统技术对比主要从任务处理、内存管理和中断处理三个方面进行。表 6.1、表 6.2 和表 6.3 详细列出了各项对比的细节。

表 6.1 任务处理方式比较

	Elastos	VxWorks AE 1.1	QNX NEUTRINO RTOS v6.2
模型	线程和进程	线程和保护域	线程和进程
优先级	256	256	64
最大任务数量	受限于内存量	受限于内存量	4095 进程，每个进程可以拥有 32767 个线程
调度策略	基于时间片的轮循调度加优先级调度	优先级占先 Round-robin	基于优先级的 FIFO 和轮循调度

表 6.2 内存管理比较

	Elastos	VxWorks AE 1.1	QNX NEUTRINO RTOS v6.2
MMU	支持	支持	支持
页交换	无	无	支持
物理页尺寸	依硬件体系结构而定	依硬件体系结构而定	依硬件体系结构而定

虚拟内存	支持	支持	支持

表 6.3 中断处理比较

	Elastos	VxWorks AE 1.1	QNX NEUTRINO RTOS v6.2
处理方式	支持基于优先级的中断嵌套	支持基于优先级的中断嵌套	支持基于优先级的中断嵌套
上下文	中断服务程序运行的某个被其绑定的线程的上下文中，中断处理程序拥有自己的上下文	中断服务程序处于特殊的上下文中	中断服务程序运行在某个被其绑定的线程的上下文中
栈	中断服务程序使用系统唯一的中断栈，中断处理程序使用自己的线程栈	中断服务程序使用特殊的中断栈，系统从内核的内存上分配了唯一的中断栈	中断服务程序使用自己的中断栈
中断和任务之间的通信	使用特殊的事件触发机制，可以访问内核地址空间	可以使用共享内存、信号量等基本消息机制	使用特殊的事件触发机制

从对比可以看到，在主要的系统技术细节上，和欣操作系统和这两个先进操作系统之间没有明显差异。

这三个系统根本的差异在于系统的整体架构上。VxWorks AE 1.1 采用整体内核的系统架构，系统服务全部由内核承担，系统不易于扩展，而且所有在内核中运行的服务都会影响系统内核的稳定性。QNX RTOS v6.2 的微内核只支持使用协作进程的方式扩展系统功能，大量的消息在不同的进程之间传递会影响系统的效率。和欣操作系统可以提供两种方式扩展系统功能，一种是扩展构件在内核中运行，一种是类似于 QNX 协作进程的方式在独立的用户进程中运行，这样可以兼顾性能和稳定性，使得系统具有更强的灵活性。

技术分析说明和欣操作系统在基本功能与先进嵌入式操作系统不存在明显差异，但同时实现了便于系统功能扩展的灵活内核模型。在新的系统功能出现的时候，和欣操作系统能够更容易的引入系统扩展构件来完成新

的功能。

6.3 实验数据及分析

前面对和欣操作系统、VxWorks AE 1.1 和 QNX RTOS v6.2 的基本系统技术进行了比较和分析，本节将三个系统的部分系统实验数据进行比较和分析。由于系统底层实验要求在系统源代码的基础上进行，因此针对 VxWorks AE 1.1 和 QNX RTOS v6.2 的实验数据是从 Dedicated Systems Experts 公司提供的资料中获取的[DSE]。

VxWorks AE 1.1 和 QNX RTOS v6.2 测试的硬件平台如下：

– PCI bus: 33MHz
– CPU: Intel Pentium 200Mhz (with 32KB L1 Cache)
– RAM: 32 Mb
– L2 Memory Cache: 512KB

由于设备限制，和欣操作系统上的实验是在下面的平台上进行的：

– 32-bit ARM9TDMI RISC Core
– CPU: High Performance (200 MHz)
– 16KB Cache, (8KB Instruction Cache, 8KB Data Cache)
– 16M ROM(MMU)

两个硬件平台具有相同的主频，从 Cache 和内存来看，后者略逊于前者，但执行速度基本上在一个数量级。表 6.4 定义了实验的内容，主要测试了系统内核的基本性能如中断处理和线程调度等。[DSE]中还给出了一些同步机制的实验数据，由于和欣系统的同步机制与 VxWorks AE 1.1 和 QNX RTOS v6.2 的同步机制在概念上存在差异，相关的数据没有可比性，因此没有在此列出。

表 6.4 实验内容说明

测试代号	描述
IL-a-1(_IST 或 _ISR)	中断延迟(从中断发生到中断处理框架的时间延迟)，只有一个线程。分别测试 ISR 和 IST 两种情况下的延迟。

IDL-a-1_ISR	中断分派延迟(从中断发生到中断分派到处理程序的延迟)，只有一个线程。分别测试 ISR 和 IST 两种情况下的延迟。
TF-a-1	线程创建时间
TF-b-1	线程删除时间
TSL-a-2	线程切换延迟，两个线程在同一个进程中
TSL-b-128	线程切换延迟，一共 128 个线程，切换发生在不同的进程之间

表 6.5 实验数据(单位：微秒)

	Elastos		VxWorks AE 1.1		QNX NEUTRINO RTOS v6.2	
测试代号	Average	Max.	Average	Max.	Average	Max
IL-a-1_ISR	9.4	25.2	1.7	4.3	1.7	6.8
IDL-a-1_ISR	11.2	29.3	1.9	2.7	1.9	8.4
IL-a-1_IST	15.5	30.3	2.3	7.7	6.5	14.2
TF-a-1	156.7	271	175	2880	260	299
TF-b-1	63	121	78	102	85	103
TSL-a-2	7.5	45	2.6	8.3	2.9	15.5
TSL-b-128	54	213	8.8	21.8	6.8	46.8

表 6.5 给出了三个系统的实验数据。从表 6.5 可以看出，和欣操作系统在中断处理和线程切换表现与另外两个操作系统还是有些差距，但差距基本上保持在数量级内。其实在这几项实验中，执行的代码量都是很少的，容易受到 Cache 性能的影响，造成这种结果差异有两个原因：一方面是受实验硬件平台影响，另外也的确说明和欣操作系统在这些方面和国际先进的嵌入式操作系统存在一定差距，需要进一步改进。

如表 6.5 显示的，和欣操作系统在 TF-a-1 和 TF-b-1 两项实验要略优于其它两个操作系统。TF-a-1 和 TF-b-1 分别是测试创建线程和删除线程的时间。在和欣操作系统上，这两项操作都是通过调用系统构件接口方法实现的，而其他两个操作系统则使用系统调用，这说明和欣系统功能没有因为使用构件技术而造成效率降低。由于和欣操作系统的系统功能是采用构件技术实现的，与系统扩展构件采用的是同样的技术，这也就说明通过系统扩展构件方式扩展系统功能是一种有效的方式。同时，系统扩展构件作为

一种系统扩展技术在没有降低效率的情况下，使得系统获得了更加易于扩展的系统架构，这正是和欣操作系统采用新系统架构设计的初衷。

6.4 小节

本章对和欣操作系统进行了系统性能分析。性能分析主要从技术和实验数据两个方面展开。在技术分析中，把和欣操作系统的系统技术细节与国外先进的嵌入式操作系统 VxWorks AE 1.1 和 QNX RTOS v6.2 进行了比较和分析。技术分析说明在系统底层技术方面，和欣操作系统与以上两个操作系统没有明显差异，功能基本相似，根本性的差异在于系统架构不同。在数据分析中，把和欣操作系统的实验数据和以上两个系统的实验数据进行了比较和分析。数据分析说明在一些性能指标上和欣操作系统要略逊色于另外两个系统，但并不明显，在系统调用的性能指标上，和欣操作系统要优于另外两个系统。综合技术分析和性能数据分析，和欣操作系统在没有造成性能明显损失的情况下，实现了具有很强扩展性的灵活内核模型，证明本文的设计是成功的。

第七章　结束语

7.1 论文总结

要从底层开发一个操作系统，至少需要投入数百人年的工作量、历时数年反复修改完善才能完成。我国具有完全自主知识产权的和欣操作系统从 2000 年开始投入开发，到 2003 年正式发布 1.0 版，前后历时两年多，作者作为该项目组的一员，见证了和欣操作系统的诞生和成长过程。开始作者只是作为一个普通的程序设计人员进入到项目中，后来有幸参与了和欣操作系统核心技术的研发，本文正是在这样的基础上完成的。文中阐述了和欣操作系统的设计和实现，主要详细说明其中所采用的关键技术，进而，通过与以往的操作系统进行对比，展示了和欣操作系统的技术特点和先进性。现将本文的内容总结如下：

- 回顾与“和欣”系统相关的操作系统研究背景，重点讨论了现代操作系统的基本系统模型和可扩展操作系统的研究工作。
- 介绍了软件构件技术的发展和一些通用的构件模型，归纳总结构件技术的相关概念，讨论了构件技术的意义。
- 详细阐述了 ezCOM 构件技术，重点讨论了 ezCOM 相对于 COM 技术所做的技术扩展，通过示例简单介绍了使用“和欣”SDK 开发 ezCOM 构件的流程。
- 论述灵活内核模型的基本准则，重点阐述了和欣操作系统基于灵活内核模型的设计和实现，比较完整地展示了“和欣”系统的内核体系结构和“和欣”构件平台。
- 阐述和欣操作系统的动态特性。简单说明系统扩展构件及其运行方式，通过详细讨论“和欣”构件平台在扩展构件运行时提供的各种支持，展示“和欣”系统的构件运行机制。

- 阐述了“和欣”系统的构件化设备驱动模型。在简单讨论传统系统的驱动程序模型的基础上，提出构件化驱动程序模型的基本设计，阐述了模型的实现和工作原理。
- 在理论和测试数据两个角度分析和欣操作系统的性能，根据分析和欣操作系统在理论架构上有一定的优势，在性能方面和先进的嵌入式操作系统没有明显的差距。另外，还着重阐述了和欣操作系统在嵌入式系统方面的一些考虑。

作者提出了一些理论和技术方面的创新，并在和欣操作系统 1.0 版中都得到了体现和应用，现将这部分研究即本文的创新点总结如下：

- 提出了灵活内核模型，给出了灵活内核系统的基本准则，而且探讨了灵活内核系统设计的一般性问题。应当指出，灵活内核模型是作者对和欣操作系统设计进行理论提升的结果，它是一般意义上的模型。在提出以后，它不再与具体的实现相关联，但在理论上对具体系统的实现有一定的指导意义。
- 作为对 COM 技术的扩展，在 ezCOM 构件技术中，引入了自描述信息的封装技术。ezCOM 封装的自描述信息包括类信息和导入信息。类信息描述构件中的所有类；导入信息描述构件之间的依赖关系。通过自描述信息的封装，可以实现构件的无注册运行，并可以支持构件的动态升级和自滚动运行。
- 和欣操作系统使用系统扩展构件对系统功能进行扩展，需要将内核空间的构件接口取回到用户空间，这个过程称为接口远程化。作为创新，本文提出了利用构件自描述信息完成接口远程化的一套方法。在接口远程化过程中，“和欣”构件平台会根据自描述信息自动生成代理构件，也就是所谓的中间层。接口远程化能够有效掩盖远程接口和本地接口之间的差异，从而实现了 ezCOM 构件的位置透明性，便于用户使用。
- 针对远程构件接口的使用，本文在“和欣”构件平台上实现了构件接口参数的自动化散列集。参数的自动化散列集是指将调用参数和

返回值在不同的地址空间传递的过程。“和欣”构件平台实现的参数自动化散列集是利用构件的自描述信息完成的，它使得构件运行无需额外设施的支持，如注册表等。有效提高了构件运行的效率和稳定性。

- 提出了构件化设备驱动模型，作为该模型的特色，“和欣”系统采用 ezCOM 构件封装设备驱动。与 Unix 系统的驱动模型相比，由于设备驱动构件的接口无需依赖于系统规定，“和欣”的构件化设备驱动模型更能体现不同设备的各自特性。
- 由于设备驱动采用构件封装，“和欣”系统的设备管理不同于传统操作系统。作为本文的另一创新，文中详细论述了“和欣”系统设备管理器的设计和实现，设备管理器负责对设备驱动进行管理，它利用“和欣”系统的构件基础设施管理设备驱动对象，如创建和销毁设备对象。通过使用设备管理器，操作系统能够很容易地处理硬件设备的变化。

和欣操作系统走的是一条自主开发的道路，它不是基于现有的开放源代码操作系统，也不是基于传统技术的传统操作系统。和欣操作系统项目的目的是开发具有自主知识产权的操作系统，在技术路线上则采用先进的软件技术和理论，其目标是成为具有国际水平、面向网络时代的新型嵌入式操作系统。

2003 年 1 月 11 日，和欣操作系统(1.0 版)通过了以汪成为院士为主任、邬贺铨院士为副主任，由七位科学院和工程院院士以及多位国家 863 计划信息组专家组成的鉴定委员会的技术鉴定。鉴定委员会认为：和欣操作系统是一个具有自主知识产权的操作系统，其体系结构和实施技术有创新性，在嵌入式操作系统领域达到了国内领先，国际先进水平，是中国软件界的一个重要成果，一致同意通过技术鉴定。

专家的鉴定结果从一个侧面说明和欣操作系统已经取得了一定的成功，而且它也已经被投入到具体的应用中，接受工业界的考验。

7.2 进一步的研究工作

7.2.1 关于灵活内核模型的实现

灵活内核模型是一种操作系统的理论模型，可以使用多种方式实现这种模型，和欣操作系统仅仅是其中之一。

本文并未探讨其它可能的灵活内核模型的实现，也就不可能在各种实现之间进行比较。但实际上，灵活内核模型的实现方式是多种多样的，比如可以采用其它构件技术代替 ezCOM 构件技术封装系统扩展，甚至也可以不采用构件技术封装系统扩展，这方面的工作还有待研究讨论。

7.2.2 和欣操作系统

和欣操作系统的构件平台提供了最基本的构件运行支持环境，但传统构件模型提供的一些服务并没有在“和欣”平台上有所体现，如持久对象等技术。这些需要在后面的研究和工作来完成。

根据测试结果分析，和欣操作系统在性能方面还有可以提高的空间，如中断处理、线程调度等。这些方面的改进进一步研究，充分吸收其他操作系统的优秀经验，逐步引入到和欣操作系统的项目中。

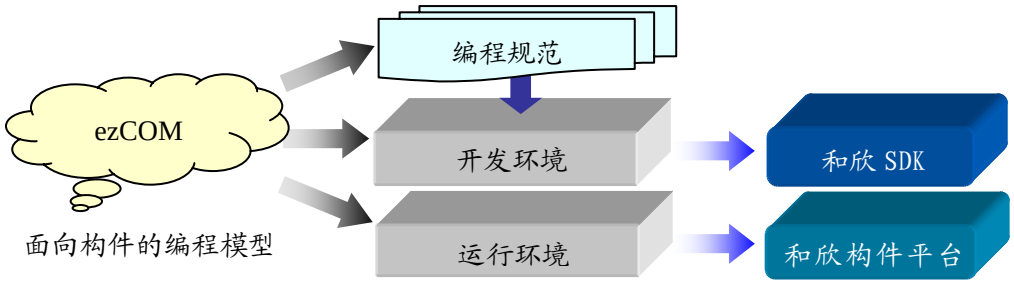
和欣操作系统的相关工作仍在进行之中，技术方面的不断更新是必然的，我期待它能有越来越好的表现。

附录 使用ezCOM构件编程

“和欣”SDK 提供了对 ezCOM 构件开发的强大支持。在“和欣”SDK 中提供了内容翔实的帮助文档，文档中详细介绍了在“和欣”SDK 上开发运行 ezCOM 构件的过程。

下面会通过简单的开发示例讲述 ezCOM 构件的开发流程。从这个示例可以看出 ezCOM 开发过程简单，但功能却很强大，通过这个示例，可以对 ezCOM 构件开发有一个最基本的认识，也是对 ezCOM 技术在和欣操作系统上的使用有比较感性的认识。

下图是 ezCOM 构件编程运行环境示意图，从图可以看出开发 ezCOM 构件需要了解 ezCOM 标准、“和欣”SDK 和“和欣”构件平台。



1 创建ezCOM构件

ezCOM 构件技术是基于微软 COM 提出的，但是在用户开发方面 ezCOM 做了很大的简化，这一节将通过创建一个简单的名为 hello 的 ezCOM 构件展示这一点。

整个开发过程是在“和欣”SDK 1.0 上进行的，如果需要获得最新的“和欣”SDK，可以到科泰世纪有限公司的网站 www.koretide.com 下载。“和欣”SDK 的操作系统平台可以是 Windows 2000 或者 Windows XP。

开发一个 ezCOM 构件分为下面三个步骤：

1) 编写 CDL 文件

2) 生成构件源程序框架并填写实现代码

3) 编译生成构件以及构件的注册

下面按照流程介绍 ezCOM 构件开发。

1.1 编写 CDL 文件

开发 ezCOM 构件的第一步是编写 CDL 文件，CDL 文件用于定义构件中的类、接口、方法及其参数等信息，关于 CDL 语言的文法可以参考“和欣”SDK 中的帮助文档。

表 8.1 是描述 hello 构件的 CDL 文件 hello.cdl 的内容。hello.cdl 文件中，定义了一个构件 hello，构件 hello 中定义了一个接口 IHello，该接口中包含一个 Hello 方法。构件中定义了一个类 CHello，该类包含一个接口 IHello。

文件最开头的一对方括号中的语句是构件 hello 的属性。按照 CDL 文法规定在构件属性中必须要有 uuid 属性和 urn 属性，否则不能通过编译。Uuid 属性定义了构件 hello 唯一的标识，用户可以使用相关的工具生成 uuid，uuid 是一个 128 位数据，它和微软中 COM 的定义是一致的。Urn 属性指定构件 hello 在互联网上的获取位置，该属性用于运行时自动从网络下载构件。另外 version 属性标识了构件的版本号。除了 hello.cdl 中用到的构件属性之外，CDL 文法中还规定了一些其它属性，详细情况请参看记录 CDL 的帮助文档。

表 8.1 描述构件 hello 的 CDL 文件

```
// 构件 hello 的定义
[
    version(1.0), uuid(e363b985-8a3a-40a6-b88c-b2e10274fe54),
    urn(http://www.koretide.com/ezcom/hello.dll)
]
component hello
{
    // 接口 IHello 的定义
    [uuid(70f1f7e4-1b9b-4e74-8c1b-fdc2fefb1ce1)]
    interface IHello {
        HRESULT Hello([in] EzStr InStr,[out, retval] EzStr *pOutStr);
    }
```

```
        // 类 CHello 的定义
        [uuid(3d19bc4c-b2c7-4ea5-8409-63db930ad1b7), scriptable ]
        class CHello {
            interface IHello;
        }
    }
```

注意：urn 属性中指定的构件名应该与 cdl 文件名以及最后生成的构件名三者必须保持一致，且大小写相关。如果书写不正确，该构件将无法正常运行。

Hello.cdl 文件中剩下的部分就是构件 hello 的定义，在构件 hello 中，定义了一个 IHello 接口和一个 CHello 类。

定义接口首先要给出接口的属性，CDL 文法规定接口属性中必须包含 uuid 属性，否则不能通过编译。接口属性中还包括一些其它可选属性，详细情况请参见 CDL 的帮助文档。

接口 IHello 中声明了方法 Hello，这个接口方法有两个参数 InStr 和 pOutStr。在 CDL 文件中，方法的定义与 C++语言中方法的声明方式类似。另外在 ezCOM 构件编程中，接口还应该遵守以下规定：

1) 接口方法必须使用 HRESULT 作为返回值；

2) 必须指明每个参数的属性，如文档中的[in]和[out, retval]。

关于接口参数属性可以参看 CDL 的帮助文档。

注意：在编写接口方法时，建议使用 ezCOM 支持的自描述数据类型作为接口方法的参数类型，否则构件将无法在远程方式下运行。示例中的参数使用了 EzStr 数据类型，EzStr 数据类型为 ezCOM 自定义的数据类型，关于其它自定义数据类型详细情况请参考“和欣”SDK 的帮助文档。

紧接着是类 CHello 的定义，与定义接口一样类的属性中也必须包含 uuid 属性，定义 CHello 时 还使用了 scriptable 属性，它标识类 CHello 的对象可以在网页上执行。

从文件内容可以看出，类 CHello 声明了接口 IHello，后面会实现这个

接口，在使用 hello 构件的时候，IHello 的方法会被调用到。

通过以上的过程，一个简单的 CDL 文件就完成了，下面就要通过它生成构件 hello 的代码框架，并添加接口实现。

1.2 代码实现

前面使用 CDL 语言编写了 hello.cdl 文件。在“和欣”SDK 开发环境下，使用 zmake 工具可以生成构件的源程序框架。这样可以减少程序员的输入量，并且有效避免输入错误。具体用法是执行下面的语句：

zmake <cdlfile>

其中<cdlfile>为 CDL 文件的全路径名。该命令执行后，在 CDL 文件所在目录中会出现几个新文件，以刚刚编写的 hello.cdl 文件为例，执行下面的命令：

zmake hello.cdl

在该文件所在的目录下会出现 CHello.h、CHello.cpp 和 sources 三个文件。CHello.h 中包含类 CHello 和接口 IHello 的声明；CHello.cpp 中包含类 CHello 的代码框架；sources 文件是“和欣”SDK 的工程文件，用于指定所在目录的工程处理信息，zmake 使用该文件对当前目录进行工程处理。

注意：不要轻易修改自动生成的 sources 文件，当然理解 sources 文件有助于更好地使用“和欣”SDK，因为它是“和欣”SDK 中一项重要的工程技术。关于 sources 文件的使用可以参考“和欣”SDK 的帮助文档。

表 8.2 是 CHello.cpp 中的内容，其中有 CHello::Hello 方法的实现，我们将对其进行适当修改，实现简单的功能。

表 8.2 CHello.cpp 文件

```
#include "CHello.h"
#include "_CHello.cpp"

DECLARE_CLASSFACTORY(CHello)

HRESULT CHello::Hello(
        /* [in] */ EzStr InStr,
```

```
        /* [out,retval] */ EzStr * pOutStr)
{
    // TODO: Add your code here
    return E_NOTIMPL;
}
```

经过简单地修改，CHello.cpp 的内容将变为：

表 8.3 修改后的 CHello.cpp 文件

```
#include "CHello.h"
#include "_CHello.cpp"

DECLARE_CLASSFACTORY(CHello)

HRESULT CHello::Hello(
        /* [in] */ EzStr InStr,
        /* [out,retval] */ EzStr * pOutStr)
{
    EzStrBuf_<50> buf;
    buf.Copy(InStr);
    buf.Append(L" world!");
    *pOutStr = EzStr::AllocString(buf);
    return NOERROR;
}
```

注意：在生成的文件框架中，方法的返回值为 E_NOTIMPL，表示此方法未实现逻辑功能。在填写完实现代码后，需要修改为适当的值，关于接口方法的返回值，请参看相关文档。

修改前 CHello::Hello 对参数不做任何处理直接返回 E_NOTIMPL；修改后 CHello::Hello 将在输入字符串 InStr 后添加“world”串，然后把新字符串赋值给字符串指针 pOutStr，然后无错返回。

完成修改以后，代码编写就此结束，下面进入代码编译阶段。

1.3 编译生成构件以及构件的注册

代码实现完成后，使用 zmake 工具编译源代码即可在镜像目录生成 ezCOM 构件。与生成构件的代码框架不同的是，这次执行 zmake 命令不需带参数。在刚才的目录下直接执行 zmake 命令后，如果命令正确执行，在镜像目录下会生成 hello.dll 文件，它就是 hello 构件的可执行文件。

如果要在 Windows 2000 上正确使用 hello 构件，还必须注册该构件。微软的 COM 信息都要注册，以便于系统定位构件的位置，为了与之兼容 ezCOM 构件也必须注册。对于刚才生成的 hello.dll，用户无需亲自注册，在编译构件的过程中，zmake 工具已经自动注册了该构件。当用户从其它的途径获得并需要使用一个 ezCOM 构件后，就需要注册该构件。

在 Windows 2000 端,可以使用“和欣”SDK 提供的 regcom 工具注册构件。比如注册 hello 构件，则可在“和欣开发环境中执行以下命令：

regcom /l hello.dll

在“和欣”操作系统上，系统使用 urn 来查找构件，因此无需注册。

通过以上过程，一个 ezCOM 构件就开发完成了，使用这个构件还需开发相应的客户端程序。

2 ezCOM构件的使用

为了使用已经开发的 ezCOM 构件 hello，仍需开发构件的客户端程序，下面开发两个客户端程序，一个是 C++客户程序，另一个是 VB Script 脚本应用。

2.1 构件 hello 的 C++客户程序

创建一个新目录，在该目录下编写构件 hello 的 C++客户程序 hello.cpp 文件，其内容如表 8.4 所示。

表 8.4 C++客户程序 hello.cpp 文件

```
#include <stdio.h>
#import <hello.dll>

int __cdecl main()
{
    EzStr outStr;
    CHelloRef cHello;                    //创建类智能指针
    HRESULT hr = cHello.Instantiate();   //创建类 CHello 的对象
    if (!cHello.IsValid()) {
        assert(0 && "Can't create cHello");
```

```
        return 1;
    }

    // 使用类智能指针 cHello 调用 Hello()方法
    hr = cHello.Hello(EZCSTR("Hello,"), &outStr);
    if (FAILED(hr)) {
        return 1;
    }

    printf("%S\n", (wchar_t*)outStr);    //打印输出结果
    EzStr::FreeString(outStr);

    return 0;
}
```

为了编译 hello.cpp 生成可执行程序，还必须在 hello.cpp 文件所在目录下创建一个文件名为 sources 的工程文件，其内容如表 8.5 所示。

表 8.5 编译 hello.cpp 文件的 sources 工程文件

```
#include <stdio.h>
#import <hello.dll>

TARGET_NAME=hello
TARGET_TYPE=exe

SOURCES=\
    hello.cpp \

ELASTOS_LIBS = \
    zeesys.lib \
    zeecrt.lib \
```

两个文件创建结束后，在文件所在目录下运行如下命令：

zmake

如果不出现错误，在镜像目录下会生成 hello.exe 程序，执行该程序，它会向屏幕输出如下结果：

Hello, world!

下面分析 hello.cpp 文件的内容，以便于更好地理解 hello.exe 程序。

文件的第二行使用 import 语句，它把构件 hello 的定义信息包含到该文件中，保证了下面使用的类智能指针 CHelloRef 有正确的定义。在编译 hello.cpp 时，编译器将按照以下顺序查找 hello.dll：当前目录、系统目录和

“和欣”SDK 开发环境的 bin 目录。由于在生成 hello 构件的时候，bin 目录下生成了 hello.dll，所以在编译 hello.cpp 时才能正确进行。

hello.cpp 文件中的 CHelloRef 是类 CHello 的智能指针类型，通过它可以调用类 CHello 的方法，当然在调用方法之前必须创建类对象，cHello.Instantiate 调用就是在创建 CHello 的对象。这里使用的方法 Instantiate 以及后面用到的 IsValid 方法都是智能指针的缺省方法。关于智能指针请参看“和欣”SDK 的帮助文档。

紧接着通过 cHello 智能指针调用 IsValid 方法判断创建对象是否成功，如果不成功则报错退出程序，如果成功则可以使用 cHello 调用类 CHello 的方法。

当 CHello 类对象创建成功后，cHello 调用了 CHello 类的方法 Hello，并且将“Hello,”字符串作为参数传给方法，Hello 方法会在“Hello,”字符串后添加“world!”，并通过第二个参数将新的字符串返回，所以在 printf 打印第二个参数时，正确的输出应该是：

Hello, world!

正如前面的运行结果。

2.2 构件 hello 的 VB Script 客户程序

在创建 hello 构件时，类 CHello 的属性中包含了 scriptable 关键字，它标明 CHello 类可以在脚本语言中使用，下面编写一个 hello 构件的 VB Script 脚本应用。

表 8.6 是文件 hello.htm 的内容，其中的 VB Script 代码演示了如何在脚本中使用 ezCOM 构件。

表 8.6 hello.htm 文件内容

```
<HTML>
<HEAD><TITLE>Hello World!</TITLE></HEAD>
<BODY>
    This is a VBScript sample that uses the component CHello<BR>
<BR>
```

```
<OBJECT id=hello
classid="CLSID:3d19bc4c-b2c7-4ea5-8409-63db930ad1b7">
</OBJECT>
<BR>
<SCRIPT LANGUAGE="VBScript">
        Document.Write hello.Hello("Hello")
</SCRIPT>
<BR>
</BODY>
</HTML>
```

可以使用 Internet Explorer 浏览 hello.htm 文件，浏览结果如图 8.1 所示。

注意：hello.htm 文件中的 classid=”CLSID:XXXX"语句使用到的 CLSID 应该是类 CHello 的 uuid，其中 XXXX 必须与类 CHello 的 uuid 完全匹配，否则浏览器输出结果会有误。

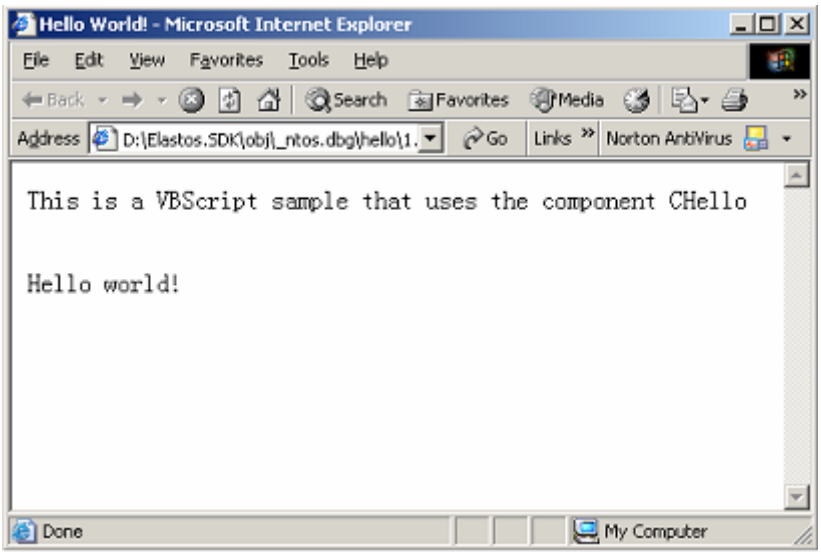


图8.1 hello.htm的浏览结果

3 总结

构件 hello 只是一个非常初级的 ezCOM 构件，使用它足以说明 ezCOM 构件的开发流程。由于这个示例过于简单，很多 ezCOM 技术都没能得到展示，比如：“和欣”构件平台支持可以直接运行的 ezCOM 构件，通过“和欣”SDK 提供的文档可以对这些技术有更深入的了解。

参考文献

[**Acce86**] Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, M. Young. *Mach: a New Kernel Foundation for UNIX Development[A]*. 1986 Summer USENIX Conference, July 1986.

[**Box98**] D. Box. Essential *COM: Object Technology Series[M]*. Addison-Wesley, 1998.

[**Bers95**] Bershad, B. N., S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. *Extensibility, safety and performance in the Spin operating system[A]*. In 15th ACM Symposium on Operating System Principles (SOSP), pp.267-284, 1995.

[**Blac92**] Black, D.L. and D.B. Golub. *Microkernel Operating System Architecture and Mach[A]*. USENIX Micro-kernels and Other Kernel Architectures Workshop Proc., pp.11-30, April 1992.

[**Bove01**] Bovet, D.P. and M. Cesati, *Understanding the Linux Kernel, 1st ed[M]*, 2001: O'Reilly.

[**Bric91**] Bricker, A., M. Gien, M. Guillemont, J. Lipkis, D. Orr, and M. Rozier. *A new look at microkernel-based Unix operating systems[R]*. Tech. Rep. CS/TR-91-7, Chorus systmes, Paris, France, 1991.

[**Brow00**] Brown A.W. Large-*Scale, Component-Based Development[M]*. Prentice Hall, 2000.

[**Brun99**] Bruno, J., J. Brustoloni, E. Gabber, A. Silberschatz and C. Small. *Pebble: A Component-Based Operating Systems for Embedded Applications[A]*. Proceedings of USENIX Workshop on Embedded Systems, Cambridge, MA, USA, March 29-31, pp.55-65, 1999.

[**Cao94**] Cao, P., E. Felten, K. Li. *Implementation and Performance of Application-Controlled File Caching[A]*. Proceedings of the First Usenix Symposium on Operating System Design and Implementation, pp.165-177, 1994.

[**Chen01A**] 陈榕. *因特网时代操作系统的演变[J]*. 计算机世界, No.42, October

2001.

[**Chen01B**] 陈榕. *中间件技术在嵌入式操作系统中的应用[A]*. Proc. of Workshop on Embedded System, Oct 2001.

[**Corb65**] Corbató, F. J. and V. A. Vyssotsky. *Introduction and Overview of the Multics System[A]*. AFIPS Conference Proceedings, 27, pp.185-196, 1965.

[**Cust93**] Custer, H. *Inside Windows NT[M]*. Microsoft Press, 1993.

[**Dibo99**] Dibora, C., S. Ockman and M. Stone. *Open Sources-voices from the Open Source Revolution[M]*. O'REILLY, 1999.

[**Du03**] 杜永文, 何华灿,陈榕. *构件化驱动程序模型[J]*. 计算机工程与应用, 2003,39(5),p28-30.

[**DSE**] Dedicated Systems Experts NV. *QNX NEUTRINO 6.2, VXWORKS AE 1.1, WINDOWS CE .NET AND ELDS 1.1 COMPARED.* http://www.dedicated-systems.com. 6, 2002.

[**Edwa00**] Edwards, A. and G. Heiser. *Components + Security = OS Extensibility[A]*. Australasian Computer Systems Architecture Conference, Goldcoast, Queensland Australia, IEEE Computer Society Press, pp.30-37, 2000.

[**Engl95**] Engler, D. R.， M. F. Kaashoek, and J. O'Toole Jr. *Exokernel: an operating system architecture for application-specific resource management[A]*. In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, December 1995.

[**Giem92**] Giemn, M. and L. Grob. *Microkernel based operating systems moving UNIX onto modern system architectures[A]*. In proc. Uniforum '92 conf., USENIX Assoc, 1992.

[**Guil91**] Guillemont, M., J. Lipkis, D. Orr and M. Rozier. *A Second-Generation Micro-Kernel Based UNIX; Lessons in Performance and Compatibility[A]*. In Proceedings of the Winter 1991 USENIX Conference,, pp.13–21, 1991.

[**Koza98**] Kozaczynski, W. and G. Booch. *Component-Based Software Engineering[J]*. IEEE Software, 15 (5) pp.34-36, 1998.

[**Lee94**] Lee, C. H., M. C. Chen, and R. C. Chang. *HiPEC: high performance external virtual memory caching[A]*. In Proceedings of the First Symposium on

Operating Systems Design and Implementation, pp.153--164, 1994.

[**Lied93**] Liedtke, J. *Improving IPC by kernel design[A]*. In 14th ACM Symposium on Operating System Principles (SOSP), pp.175-187, 1993.

[**Lied95**] Liedtke, J. *On MicroKernel Construction[A]*. SOSP'95, Cooper Mountain, Colorado, USA, pp.237-250, December 1995.

[**Lied96**] Liedtke, J. *Towards real microkernels[J]*. Communications of the ACM, 39(9) pp.70--77, September 1996.

[**Lied97**] Liedtke, J., K. Elphinstone, S. Schnberg, H. Hrtig, G. Heiser, N. Islam, and T. Jaeger. *Achieved IPC Performance (Still the Foundation for Extensibility)[A]*. Sixth Workshop on Hot Topics in Operating Systems, May 1997.

[**Mail00**] Maillet, S. *Using COM for Embedded Systems (Part II)[A]*. Embedded System Conference Session #425, 2000.

[**Maur01**] Mauro, J. and R. McDougall. *Solaris Internals: Core Kernel Architecture[M]*. Prentice Hall PTR, Upper Saddle River, New Jersey, 2001.

[**McIl68**] McIlroy, D. *Mass produced software component[A]*. 1968 NATO Conf. On Software Engineering, pp.138-155, 1968.

[**Micr95**] Microsoft Corporation. *The Component Object Model Specification[R], Version 0.9*. October 1995.

[**Micr98**] Microsoft Corporation. *DCOM Architecture[R]*. Technical report, 1998.

[**Nels91**] Nelson, G.(ed.). *Systems programming with Modula-3[M]*. Prentice Hall Series in Innovative Technology, Prentice Hall, 1991.

[**Obje01**] Object Management Group. *The Common Object Request Broker: Architecture and Specification Revision 2.6[R]*. OMG Technical Document Number 00-12-01, 2001.

[**Open98**] OpenGroup. *Distributed Computing Environment*. http://www.camb.opengroup.org/dce/, 1998.

[**Orfa96**] Orfali, R., D. Harkey, and J. Edwards. *The Essential Client/Server Survival Guide(2nd Edition)[M]*. John Wiley & Sons Inc., New York, 1996.

[**Pu88**] Pu, C., H. Massalin, and J. Ioannidis. *The Synthesis kernel[J]*. Computing Systems, 1(1): 11-32, Jan 1988.

[**Roge97**] Rogerson, D.. *Inside COM[M]*. Redmond, Washington: Microsoft Press, 1997.

[**Smal96**] Small, C. and M. I. Seltzer. *A Comparison of OS Extension Technologies[R]*. USENIX Annual Technical Conference, pp.41-54, 1996.

[**Szyp98**] Szyperski , C. *Component Software: Beyond Object-Oriented Programming[M]*. Addison Wesley, 1998.

[**Tane94**] Tanenbaum A.S. and M. F. Kaashoek. *The Amoeba Microkernel Distributed Open Systems[J]*. IEEE Computer Society Press, pp.11-30, 1994.

[**Tane97**] Tanenbaum, A.S. and A.S. Woodhull. *Operating Systems: Design and Implementation(2nd edition)[M]*. Prentice-Hall, 1997.

[**Tane01**] Tanenbaum, A.S. *Modern Operating Systems(2nd edition)[M]*. Prentice-Hall, 2001.

[**Vaha96**] Vahalia, U. *UNIX Internals: The New Frontiers[M]*. Prentice-Hall, 1996.

[**Wahb93**] Wahbe, R., S. Lucco, T. E. Anderson, and S. L. Graham. *Efficient Software-Based Fault Isolation[A]*. Proc. of the 14th ACM Symposium on Operating System Principles, pp.203—216, 1993.

攻读博士期间论文写作

1. 杜永文，何华灿，陈榕，构件化驱动程序模型，计算机工程与应用，Vol39(No.05)，2003,2,p28-30
2. 杜永文，何华灿，陈榕，基于灵活内核的构件化驱动程序，小型微型计算机系统，Vol25(No.4)，2004，4，p587-590
3. 杜永文，何华灿，陈志成，和欣操作系统的灵活内核技术，计算机工程与应用，Vol40(No.32)，2004，11，p106-108
4. 杜永文，何华灿，灵活内核模型，小型微型计算机系统，已投稿(2004年投稿)
5. 何华灿，王华，刘永怀，王拥军，杜永文，《泛逻辑学原理》(专著)，科学出版社，2001
6. 王红，杜永文，球面、柱面、锥面与非线性控制系统，系统科学与数学，Vol21(No.2)，2001，4，p163-171
7. 王拥军，何华灿，杜永文，模糊不一致需求的折衷分析，计算机科学，Vol28(No.5)，2001，5
8. 王拥军，何华灿，杜永文，利用广义粗近似挖掘缺省规则，模式识别与人工智能，Vol14(No.3)，2001，9
9. 陈志成，何华灿，毛明毅，杜永文，生物细胞生长的计算机分型模拟，计算机工程与应用，Vol39(No.36)，2003，12，p24-27
10. 王拥军，何华灿，艾丽蓉，杜永文，需求工程中的不确定性研究，计算机科学，Vol28(No.2)，2001，2

攻读博士期间参与的工作

- 在北京科泰世纪科技有限公司期间，作者还作为专利发明人参加了以下专利的撰写工作：

- 陈榕，杜永文，林清洪， 《基于动态内核实现跨地址空间创建构件对象的方法》，北京科泰世纪科技有限公司， 02160134.8
- 苏翼鹏， 陈榕， 杜永文， 梁宇洲， 《基于类别创建驱动构件对象实现设备驱动程序多态的方法》， 北京科泰世纪科技有限公司，02159488.0,
- 陈榕， 苏翼鹏， 杜永文， 邓康， 《基于构件的操作系统动态设备驱动的方法》，北京科泰世纪科技有限公司， 02159471.6,
- 陈榕， 苏翼鹏， 杜永文， 梁宇洲，叶忠强， 《构件自描述封装方法及运行的方法》，北京科泰世纪科技有限公司， 02160137.2,
- 陈榕， 杜永文， 刘亚东， 《基于构件技术的面向方面的实现方法》，上海科泰世纪科技有限公司， 已经提交律师审查

■ 1999 年，参加了《泛逻辑学》的编著工作。在编写期间，本人的工作主要是论证书中的部分数学公式，另外，还参加了后期的文字校正工作。

■ 2000 年 10 月－2003 年 6 月，参与了北京科泰世纪有限公司的和欣操作系统项目。在参与该项目期间，本人的工作有：

- 参与和欣操作系统的内核设计和代码开发。
- ezCOM 构件技术的研发。
- 构件化设备驱动模型的设计和实现。

致　谢

由衷地感谢导师何华灿教授！本文的研究工作是在何老师的悉心指导和耐心鼓励下得以完成的。从博士入学到学位论文完成的四年多时间，何老师那渊博的知识、严谨的治学态度、学术研究上淡泊名利的进取精神，对学生和蔼可亲、平易近人的大家风度，时时在激励着我；他对世界变化的敏锐洞察力和博览中西诸家后独特的研究视角，使得其抓住了当前科学研究的关键问题，始终使自己的研究工作站在领域前沿，这一切都在潜移默化中影响着我，鞭策我在未来的科学研究工作中始终保持不唯众、不唯名利、只唯真理的信念。在论文完成之际，从心底里向我的导师致以诚挚的谢意。

感谢北京科泰公司为我提供了参与和欣操作系统项目的机会，感谢公司的各位领导，尤其是首席设计师陈榕，他在多方面对我进行了指导，拓宽了我的知识面，而且论文中的很多细节的澄清都得益于我与他的讨论。感谢在公司期间帮助过我的同事张鼎文、林清洪、王维汉、苏翼鹏等，由于他们的帮助，我的论文研究工作才得以顺利进行，也使得我在公司的生活显得丰富多彩。

感谢在我的论文开题到完成过程中给予帮助的 1106 教研室全体老师和同学们。尤其是当我处于论文研究遇到困惑之际，王拥军博士、陈志成博士、李新博士、张静博士给予了的关键性帮助，其他如艾丽蓉博士、祝峰博士、陈丹博士、薛占鳌博士、毛明毅博士、张彦明硕士与我在诸方面的讨论、切磋都使我受益颇深。

感谢系办公室尹信琴老师在我学位期间给予的各种关心和帮助，提供的有利于论文完成的各项便利条件。

感谢我的父亲母亲，有了他们的理解和支持，我才能完成这么多年的求学。他们对我的爱如大海之深，难以报答万一。

最后，深深感谢我的妻子岳震娅。正是她几年来默默的奉献和全力的支持，我才得以全身心地投入研究工作中，她的支持很难用一个谢字当得，只想说本文是我与她的共同成果。