申请同济大学工学硕士学位论文

基于 CAR 构件的浏览器架构研究

（国家 863“软件重大专项”项目　编号： 2001AA113400）

培养单位：电子与信息工程学院

一级学科：计算机科学与技术

二级学科：计算机应用

研 究 生：赵金钟

指导教师：顾伟楠　教授

二〇〇六年二月

A dissertation submitted to
Tongji University in conformity with the requirements for
the degree of Master of Science

The Study of Browser Framework Based on CAR Component

(Supported by National 863 High Tech Program, Grant No. 2001AA113400)

School/Department: School of Electronics and Information Engineering
Discipline: Computer Science and Technology
Major: Computer Application
Candidate: Zhao Jinzhong
Supervisor: Prof. Wei-Nan Gu

February, 2006

基于CAR构件技术的浏览器研究

赵金钟　同济大学

学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年　　月　　日

经指导教师同意，本学位论文属于保密，在　　年解密后适用本授权书。

指导教师签名：　　学位论文作者签名：

年　　月　　日　　年　　月　　日

同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

签名：

年　　月　　日

摘要

和欣平台[1][2]成功地把网络服务编程模型应用到嵌入式程序设计中，在嵌入式领域首创了类似于JAVA，.NET的分布式解决方案。和欣平台上“软件无处不在，软件与网络无关”的设计理念， 是开始此系统得最原始目的。浏览器在和欣平台的网络服务编程模型中扮演着重要的角色，不但要实现传统嵌入式浏览器访问Internet和Web服务的功能，同时还必须作为和欣操作系统的Shell， 为和欣平台上的应用程序提供了一种方便的表示层实现方式，实现因此和欣平台上基于CAR构件技术的浏览器的研究具有十分重要的意义。

论文在分析目前主流的浏览器的系统结构，同时借鉴国外一些浏览器开源项目，并结合和欣操作系统的实际情况，提出了适合和欣操作系统的基于浏览器模型的图形系统体系架构。该架构具有模块化、可定制行为、方便扩展的特性。该架构除包含数据处理，布局，显示，网络等基本功能模块外，还抽象出了文档对象树模块。利用文档对象树进行页面元素的管理，有效简化了系统的开发，并提供了良好的扩展性。模块之间利用消息机制和回调机制进行通讯，有效降低了模块间的耦合性。由于基于自定义的面向对象机制和消息机制，减少了平台依赖性。

论文阐述了 Everest 中如果把浏览器模式的表示层方法与和欣操作系统中原有图形机制结合起来的设计理念。包括主要的数据结构的抽象－文档对象树的设计，以及 Parser, Layout 和 Render 几个功能模块的设计。同时介绍了设计中采用的一些优化技术，比如数据分段下载解析，利用表格定位标签，优化写屏次数等。

论文在最后介绍了 Everest 的实现，并讨论 Everest 对逻辑 CAR 构件的支持。在分析 CAR 支持需要解决的问题的同时，根据目前系统的实现提出相应的解决方案。系统的实现完全采用和欣平台提供的 CAR 构件语言实现。Everest 中分析模块具备分段数据传输解析，传输解析异步进行，多线程取数据等特点。Debug 版的 Everest 的可执行程序代码为 300k。目前已经成功运行于 X86，ARM 等平台上。

关键词：嵌入式操作系统，浏览器引擎，XML，图形系统架构

ABSTRACT.

Key Words: DTV, component technology, middleware

目录

第 1 章 引言

1.1 概述

图形系统发展到今天已经经历了很多的风风雨雨。主要有Linux平台上的Xwindow， KDE， Gnome，和Windows平台上的win31，win95到目前一直推迟被推迟的Longhorn。不管事Linux平台下的Gnome， 还是Windows平台下的桌面系统，其图形应用程序都有着不同的编程方式，并且其紧密的平台相关性，使其只能绑定在一定的平台上运行。Java 推出以后，“一次编写，到处运行”的理念，让你程序轻松实现了跨平台，这个也是由于在不同平台上JVM的实现带来的。

谁也不否认，在80年代，凭借MS DOS和后来的Win3.1图形操作系统，技术天才比尔.盖茨创建的微软成为PC时代的霸主。谁也不会想到，93年到94年，浏览器的出现给大家带来了革命性的变革，Netscapte正式走上历史舞台，开始了真正的所谓“Internet”时代。用户可以用简单简单html语言来把自己的文字信息，图片信息，在Internet上发布。

浏览器带给了人们网络的革命，使得可以通过单机获取世界上任何地方的信息，达到了信息本地化，信息透明的目的。冯·诺伊曼曾经提出“程序就是数据”原则，而作为下一代的和欣嵌入式中间件平台Elastos[1]，将利用浏览器图形系统实现运算透明化的目标，从而达到“网络就是计算机，软件无处不在”的目的。当计算机无法在硬盘上找到所需的应用程序时，也应该完全可以从网络上进行搜索、加载到内存，就像我们现在浏览网页一样，这个加载、删除过程对用户是完全自动、透明的[1][2]。通过自动加载，和动态运行，对于普通用户，只要编写简单的XML文件来实现所需要的程序，通过我们的图形系统就可以完成程序的自动装载和运行。

经过一段时间的沉寂，随着Firefox[5]的推出，浏览器的技术有了新的发展。但是更惊人的是，Firefox的界面部分的程序全部采用XML来编写，这就是Mozilla组织提供的XUL技术，也是通过XML和浏览器技术来实现图形界面。而微软的也推出Avalon，Avalon会定义一个可在Longhorn中使用的新标记语言，其代号为“XAML”（可扩展应用程序标记语言）。可以使用XAML来定义文本、图像和控

件的布局，这与使用HTML非常相似。大多数写入Avalon的应用程序均可能同时包含程序代码和XAML。可以看到目前网络服务编程发展的方向是客户端提供更丰富的客户体验，也就是富互联网应用（RIA）[7][8]。而同时随着3G时代的到来，网络带宽增加，对嵌入式的应用也提出了新的要求，嵌入式应用可以通过更好的网络带宽提供给用户更丰富的体验，而事实上目前嵌入式平台上浏览器技术和图形系统的融合也势在必行。目前嵌入式平台的图形技术还远远落后于网络技术的发展。

和欣操作系统是为了3G而量身定做的操作系统，其中微内核结构，点击运行，自滚动下载等理念充分体现了网络时代3G操作系统的特色[1][2]。和欣操作系统的这些特性为提供良好的嵌入式应用的体验创造了条件，通过在图形系统中添加浏览器引擎，能够方便地实现智能客户端，从而提供嵌入式领域富互联网应用的解决方案。

综上所述，和欣操作系统中的点击运行需要浏览器引擎的支持，同时为了提供嵌入式应用更好的体验，CAR构件技术的浏览器也将适应下一代网络服务编程模型，[9]。

1.2 课题来源及研究意义

本课题来源于国家863重大软件专项项目——“基于中间件技术的因特网嵌入式操作系统及跨操作系统的中间件运行平台”。本课题的主要目标是研究和开发基于嵌入式操作系统的图形系统开发平台，改平台作为系统中间件，运行于面向构件的嵌入式操作系统中，为新一代的面向3G时代的各类应用系统的开发建立高效可靠的软件平台，基于该平台所开发的应用程序不经任何修改，甚至不必重新编译、链接就可以运行在不同硬件厂商提供的平台上,实现应用程序的跨平台性，从而可以做到“一次编写,多处运行”的目的，从而加快应用系统的研制与普及，降低开发成本。本文是在该课题的基础上进行的。首先研究了浏览器引擎和图形系统中间件的关键技术；然后分析了基于解析语言的图形系统技术架构和技术本质，提出了基于“和欣”操作系统的构件化基于浏览器模式图形开发平台的总体设计方案；最后给出了该软件平台的详细设计和实现。

和欣操作系统上基于浏览器模型的图形系统研究的课题属于国家“863”项目------“和欣操作系统”，主要是对此在技术的研究并在和欣操作系统上进行实

现，一方面在和欣平台上实现基于浏览器模型的图形系统应用，可以在和欣平台通过此系统访问Internet，另一方面，更重要的是为将来实践点击运行，为网络服务解决方案提供操作系统级的支持和实践。

1.3 论文结构

随着嵌入式领域的发展，嵌入式平台上提供接入因特网的主要应用软件，方兴未艾。论文将结合面向网络服务编程的下一代手机操作系统，和欣操作系统，介绍在嵌入式图形系统领域所作的研究和工作。

论文在第二章对相关技术进行了介绍，首先介绍了CAR构件技术，CAR构件技术是和欣运行平台的基础；其次介绍了和欣操作系统，及图形系统Atlas；这一章最后介绍了嵌入式浏览器引擎和图形系统及相关技术HTML， XML和DOM。

第三章首先对目前的嵌入式浏览器引擎和图形系统框架进行分析，并总结出嵌入式图形系统框架的一般特点和设计框架时需要注意的问题；接着提出了和欣平台上嵌入式图形系统融合浏览器引擎的框架设计，该框架具有灵活，扩展性好等特点；下面结合此框架实现中存在的问题详细说明了Everest框架的特色；在最后介绍了基于该框架实现的和欣平台上的基于浏览器模式图形系统Everest。

第四章详细介绍了Everest中图形系统的接口设计以及浏览器引擎实现，包括了底层图形系统，消息控制模块，图形系统构件，浏览器引擎中的文档对象树，数据处理模块，布局和显示等模块。同时介绍了设计中采用的一些优化技术，比如数据分段解析，利用表格来查找数据项等。

第五章介绍了Everest的实现情况，给出了一些实验数据和效果截图，并对Everest的特点进行总结，最后讨论如何在进一步和CAR构件融合支持。

第六章对目前的研究工作进行了总结，并提出了未来可能的研究方向和关键性问题。

第 2 章 “和欣”嵌入式操作系统及其构件技术

2.1 “和欣”操作系统概述

2.1.1 “和欣”操作系统简介

“和欣”是32位嵌入式操作系统。该操作系统可以从多个侧面进行描述：

32位嵌入式操作系统。操作系统基于微内核，具有多进程、多线程、抢占式、基于线程的多优先级任务调度等特性。提供FAT兼容的文件系统，可以从软盘、硬盘、FLASH ROM启动，也可以通过网络启动。和欣操作系统体积小，速度快，适合网络时代的绝大部分嵌入式信息设备。

完全面向构件技术的操作系统。操作系统提供的功能模块全部基于CAR构件技术，因此是可拆卸的构件，应用系统可以按照需要剪裁组装，或在运行时动态加载必要的构件。

从传统的操作系统体系结构的角度来看，和欣操作系统可以看成是由微内核、构件支持模块、系统服务器组成的。

- 微内核：主要可分为 4 大部分：硬件抽象层（对硬件的抽象描述，为该层之上的软件模块提供统一的接口）；内存管理（规范化的内存管理接口，虚拟内存管理）；任务管理（进程管理的基本支持，支持多进程，多线程）；进程间通信（实现进程间通信的机制，是构件技术的基础设施）。
- 构件支持模块：提供了对 CAR 构件的支持，实现了构件运行环境。构件支持模块并不是独立于微内核单独存在的，微内核中的进程间通讯部分为其提供了必要的支持功能。
- 系统服务器：在微内核体系结构的操作系统中，文件系统、设备驱动、网络支持等系统服务是由系统服务器提供的。在和欣操作系统中，系统服务器都是以动态链接库的形式存在。

2.1.2 和欣灵活内核简介

和欣操作系统的实现全面贯穿了CAR构件思想，CAR构件可以运行于不同

地址空间或不同的运行环境。可以把操作系统的内核地址区看成是一段特殊的地址空间，用户可以根据运行时的需求，自主选择将操作系统的某些系统服务构件、文件系统、图形系统、设备驱动构件等运行于内核地址空间或用户地址空间。与传统的操作系统的“大内核”、“微内核”体系结构相比，和欣操作系统内核里提供的系统服务，完全可以由用户依据系统自身的需求动态决定。因此称和欣操作系统内核为“灵活内核” （Agile Kernel）。

和欣灵活内核的体系结构，利用构件和中间件技术解决了长期以来困扰操作系统体系结构设计者的大内核和微内核在性能、效率与稳定性、安全性之间不能两全其美的矛盾。

下图来表示和欣灵活内核及其与系统构件和应用构件的关系：

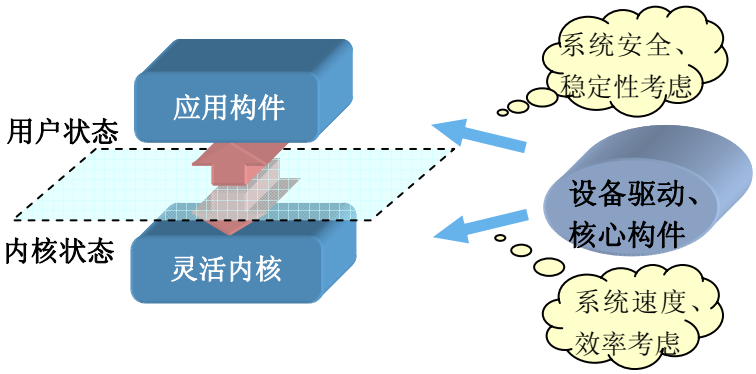

图 2.1 “和欣”灵活内核与系统构件和应用构件的关系

2.1.3 和欣操作系统提供的功能

从应用编程的角度看，和欣操作系统提供了一套完整的、符合COM规范的系统服务构件及系统API，为在各种嵌入式设备的硬件平台上运行CAR二进制构件提供了统一的编程环境。

和欣操作系统还提供了一组动态链接构件库，这些构件库通常是开发嵌入式应用系统时不可缺少的：

- 图形系统构件库（方便开发图形用户操作界面）；
- 设备驱动构件库（各种输入输出设备的驱动）；
- 文件系统构件库（FAT 兼容，包括对 FLASH 等的支持）；
- 网络系统构件库（TCP/IP 等网络协议支持）。

系统提供的构件库，以及用户开发的应用程序构件都是通过系统接口与内

核交互，从这个意义上说，他们处于同样的地位。用户可以开发性能更好或者更符合需求的文件系统、网络系统等构件库，替换这些构件库，也可以开发并建立自己的应用程序构件库。这就是基于构件技术操作系统的优势之一。

此外，为了方便用户编程，在和欣SDK中还提供了以下函数库：

- 与微软 Win32 API 兼容的应用程序编程接口(zeew32 API)；
- 标准 C 运行库（libc）；
- 和欣提供的工具类函数（zeeutil）。

对程序员来说，和欣操作系统提供的用户编程接口与上一节中介绍的和欣构件运行平台完全一样。所以，在相互兼容的硬件平台上，不管运行的是和欣操作系统还是Windows操作系统，应用程序可以不加区分地在其上运行。

和欣操作系统实现并支持系统构件及用户构件相互调用的机制，为CAR构件提供了运行环境。关于CAR构件的运行环境，其描述与“和欣构件运行平台”是一样的，在此从简。因此，可以把和欣操作系统看成是直接运行在硬件平台上的“和欣构件运行平台”。

可以用下图来表示和欣操作系统及其主要构成：

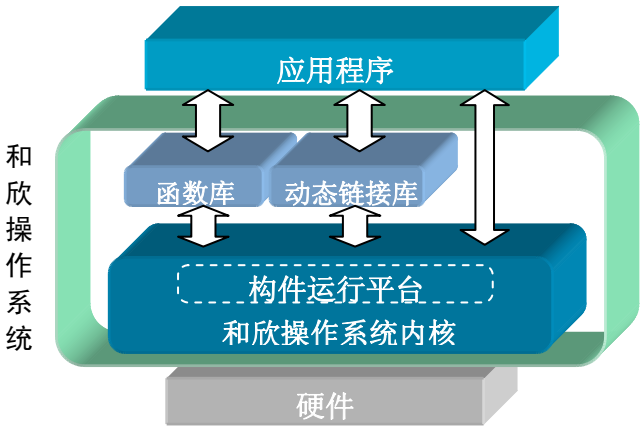

图 2.2 “和欣”操作系统的系统结构图

2.1.4 和欣操作系统的应用软件开发

和欣SDK提供了应用软件的开发环境和工具。开发“和欣”应用软件的开发环境如下图所示：

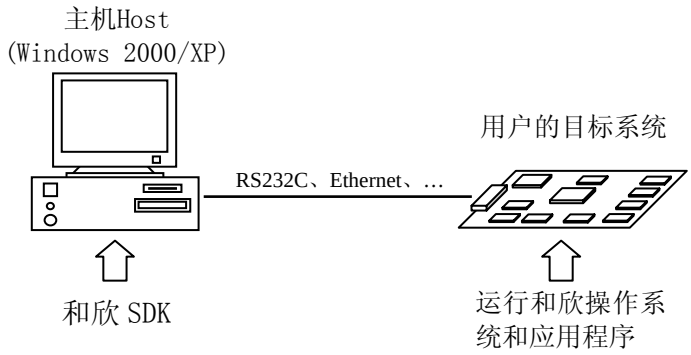

图 2.3 “和欣”应用软件的开发环境

开发“和欣”应用软件的过程，如下图所示：

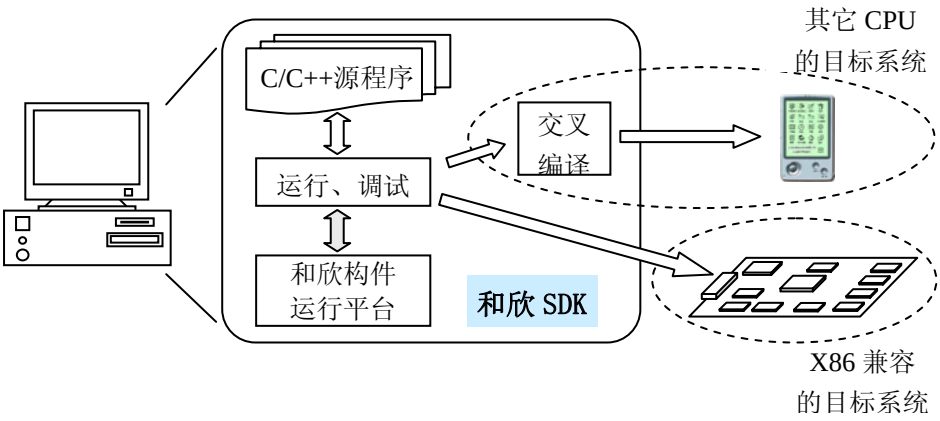

图 2.4 “和欣”应用软件的开发过程

2.1.5 和欣操作系统的优势

和欣操作系统的最大特点就是：

- 全面面向构件技术，在操作系统层提供了对构件运行环境的支持；
- 用构件技术实现了“灵活”的操作系统。

这是和欣操作系统区别于其它商用嵌入式操作系统产品的最大优势。

在新一代因特网应用中，越来越多的嵌入式产品需要支持网络服务，而网络服务的提供一定是基于构件的。在这种应用中，用户通过网络获得服务程序，这个程序一定是带有自描述信息的构件，本地系统能够为这个程序建立运行环境，自动加载运行。这是新一代因特网应用的需要，是必然的发展方向。和欣操作系统就是应这种需要而开发，率先在面向嵌入式系统应用的操作系统中实现了面向构件的技术。

因此，构件化的和欣操作系统可以为嵌入式系统开发带来以下好处：

- 在嵌入式软件开发领域，导入先进的工程化软件开发技术。嵌入式软件一般用汇编语言、C 语言，在少数系统中已经支持了 C++开发，但是由于还没有一个嵌入式操作系统能够提供构件化的运行环境，可以说，嵌入式软件开发还是停留在手工作坊式的开发方式上。和欣操作系统使得嵌入式应用的软件开发能够实现工程化、工厂化生产。
- 可以动态加载构件。动态加载构件是因特网时代嵌入式系统的必要功能。新一代 PDA 和移动电话等移动电子产品，不能再像以前那样由厂家将所有的功能都做好后固定在产品里，而要允许用户从网上获得自己感兴趣的程序。
- 随时和动态地实现软件升级。动态加载构件的功能，同样可以用于产品的软件升级，开发商不必为了添加了部分功能而向用户重新发布整套软件，只需要升级个别构件。
- 灵活的模块化结构，便于移植和剪裁。易于定制成针对不同硬件配置的紧凑高效的嵌入式操作系统。添加或删除某些功能模块也非常简单。
- 嵌入式软件开发商容易建立自己的构件库。在不同开发阶段开发的软件构件，其成果很容易被以后的开发所共享，保护软件开发投资。软件复用使得系列产品的开发更加容易，缩短新产品开发周期。
- 容易共享第三方软件开发商的成果。面向行业的构件库的建设，社会软件的丰富，使得设备厂家不必亲自开发所有的软件，可以充分利用现有的软件资源，充分发挥自己的专长为自己的产品增色。
- 跨操作系统平台兼容，降低软件移植的风险。在和欣开发环境上开发的软件所具有的跨平台特性，使得用户可以将同样的可执行文件不加修改地运行在和欣操作系统（嵌入式设备）与 Windows 2000/XP（PC）上。特别是对于需要将 Windows 上的软件移到嵌入式系统以降低产品成本的用户，这一特点不仅可以大大节约软件移植的费用，还可以避免因移植而带来的其它隐患。
- 功能完备的开发环境和方便的开发工具，帮助嵌入式开发人员学习和掌握先进的构件化软件编程技术，提高软件开发效率。应用软件可以在开发环境下开发调试，与硬件研制工作同时进行，缩短产品研制周期。

2.2 和欣构件运行平台

2.2.1 和欣构件运行平台简介

和欣构件运行平台提供了一套符合CAR（详见2.3节）规范的系统服务构件及支持构件相关编程的API函数，实现并支持系统构件及用户构件相互调用的机制，为CAR构件提供了编程运行环境。和欣运行平台有在不同操作系统上的实现，符合CAR编程规范的应用程序通过该平台实现二进制级跨操作系统平台兼容。

在和欣操作系统中，和欣构件运行平台与“和欣灵活内核”共同构成了完整的操作系统。

在Windows 2000、WinCE、Linux 等其它操作系统上，和欣构件运行平台屏蔽了底层传统操作系统的具体特征，实现了一个构件化的虚拟操作系统。在和欣构件运行平台上开发的应用程序，可以不经修改、不损失太多效率、以相同的二进制代码形式，运行于传统操作系统之上。

下图显示了和欣构件运行平台在Windows 2000/XP、和欣操作系统中的位置。

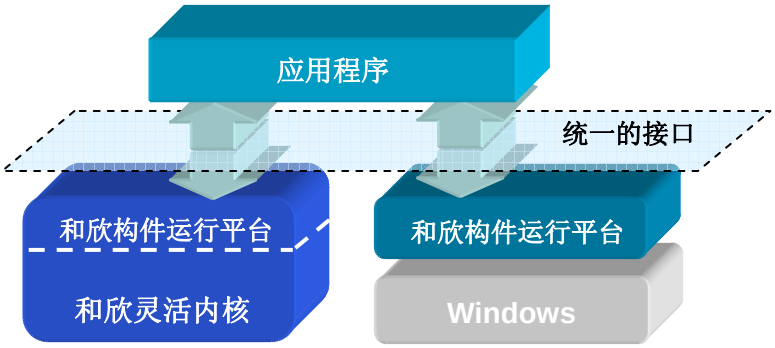

图 2.5 “和欣”构件运行平台与操作系统的关系

2.2.2 和欣构件运行平台的功能

从和欣构件运行平台的定义，知道该平台为CAR提供了运行环境。从这个意义上，这里说的CAR技术也可以理解为在运行环境中对CAR规范提供支持的程序集合。

从编程的角度看，和欣构件运行平台提供了一套系统服务构件及系统API

（应用程序编程接口），这些是在该平台上开发应用程序的基础。

和欣操作系统提供的其它构件库也是通过这些系统服务构件及系统API实现的。系统提供的这些构件库为应用编程开发提供了方便：

- 图形系统构件库；
- 设备驱动构件库；
- 文件系统构件库；
- 网络系统构件库。

从和欣构件运行平台来看，这些构件和应用程序的构件是处于同样的地位。用户可以开发性能更好或者更符合需求的文件系统、网络系统等构件库，替换这些构件库，也可以开发并建立自己的应用程序构件库。

右图显示出和欣构件运行平台的功能及其与构件库、应用程序的关系。

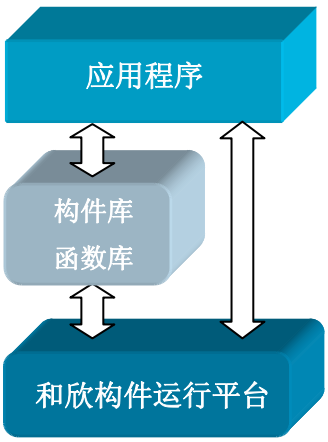

图 2.6 和欣构件运行平台功能图

从支持CAR构件的运行环境的角度看，和欣构件运行平台提供了以下功能：

- 根据二进制构件的自描述信息自动生成构件的运行环境，动态加载构件；
- 提供构件之间的自动通信机制，构件间通信可以跨进程甚至跨网络；
- 构件的运行状态监控，错误报告等；
- 提供可干预构件运行状态的机制，如负载均衡、线程同步、访问顺序控制、安全（容错）性控制、软件使用权的控制等；
- 构件的生命周期管理，如进程延续（Persistence）控制、事务元（Transaction）控制等；

总之，构件运行平台为CAR构件提供了对程序员完全透明的运行环境，构件可以运行在不同地址空间，不同环境，甚至跨网络。构件运行平台自动为构件运行提供支持，配置必要的网络协议、针对不同的输入输出设备的协议。程序员不必过多地去关心诸如网络协议转换及构件运行控制等与其它构件互操作时的协调问题，只需专注于自己需要解决的程序算法的实现。从而可以从繁杂庞大的应用环境体系中解放出来，大大提高编程的效率。

和欣构件运行平台直接运行二进制构件，而不是像JAVA和.NET那样通过虚

拟机在运行程序时解释执行中间代码。因此，与其它面向构件编程的系统相比，具有资源消耗小，运行效率高的优点。

2.2.3 和欣构件运行平台的技术优势

作为总结，和欣构件运行平台的主要技术优势列举如下：

- 开发跨操作系统平台的应用软件；
- 对程序员透明的 CAR 构件运行环境，提高编程的效率；
- 直接运行二进制构件代码，实现软件运行的高效率；
- 构件可替换，用户可建立自己的构件库。

需要说明的是，和欣构件运行平台实现的应用软件跨操作系统平台兼容是以具有同样的硬件体系结构为前提的。目前，和欣构件运行平台还不能支持不同指令系统的CPU间的“跨平台”兼容。

2.2.4 利用和欣构件运行平台编程

对程序员来说，编写运行于和欣构件运行平台上的程序，运用CAR技术和跨平台技术的具体方法，体现在对构件库的接口方法、通用API函数的调用上。应用程序运行所需要的动态链接库，则是在程序运行时由和欣构件运行平台自动加载的。

下图简明地表示了编写运行于和欣构件运行平台上的应用程序所需的相关要素之间的关系的示意。

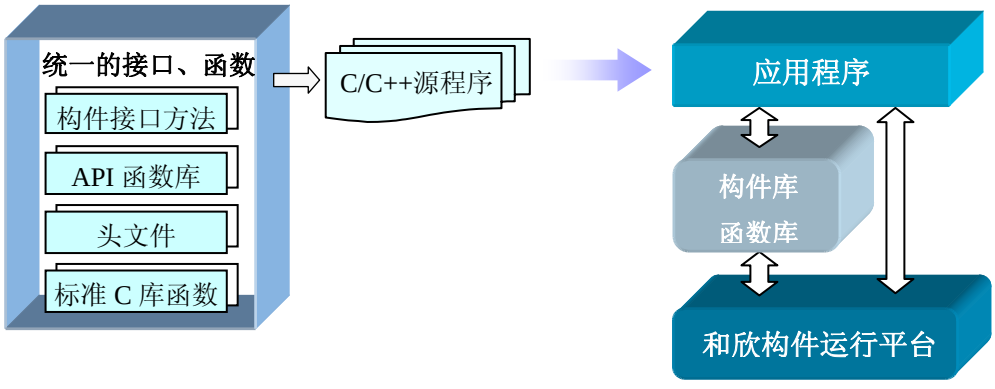

图 2.7 如何编写基于“和欣”构件运行平台的应用程序

2.3 CAR 构件技术

2.3.1 CAR 技术的由来

80年代以来，目标指向型软件编程技术有了很大的发展，为大规模的软件协同开发以及软件标准化、软件共享、软件运行安全机制等提供了理论基础。其发展可以大致分为以下几个阶段。

（1） 面向对象编程

通过对软件模块的封装，使其相对独立，从而使复杂的问题简单化。面向对象编程强调的是对象的封装，但模块（对象）之间的关系在编译的时候被固定，模块之间的关系是静态的，在程序运行时不可改变模块之间的关系，就是说在运行时不能换用零件。其代表是C++语言所代表的面向对象编程。

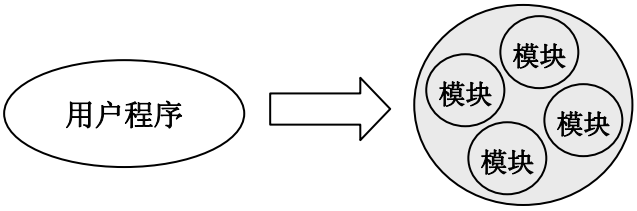

图 2.8 面向对象的编程模型

（2） 面向构件编程

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，构件之间的连接和调用要通过标准的协议来完成。构件化编程模型强调协议标准，需要提供各厂商都能遵守的协议体系。就像公制螺丝的标准一样，所有符合标准的螺丝和螺母都可以相互装配。构件化编程模型建立在面向对象技术的基础之上，是完全面向对象的，提供了动态构造部件模块（运行中可以构造部件）的机制。构件在运行时动态装入，是可换的。其代表是COM技术。

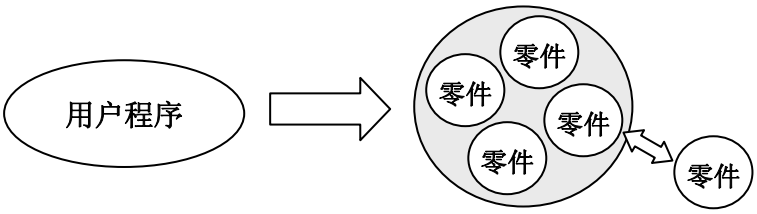

图 2.9 面向构件的编程模型

（3） 面向中间件编程

由于因特网的普及，构件可来自于网络，系统要解决自动下载，安全等问题。因此，系统中需要根据构件的自描述信息自动生成构件的运行环境，生成代理构件即中间件，通过系统自动生成的中间件对构件的运行状态进行干预或控制，或自动提供针对不同网络协议、输入输出设备的服务（即运行环境）。中间件编程更加强调构件的自描述和构件运行环境的透明性，是网络时代编程的重要技术。其代表是CAR、JAVA和.NET（C#语言）。

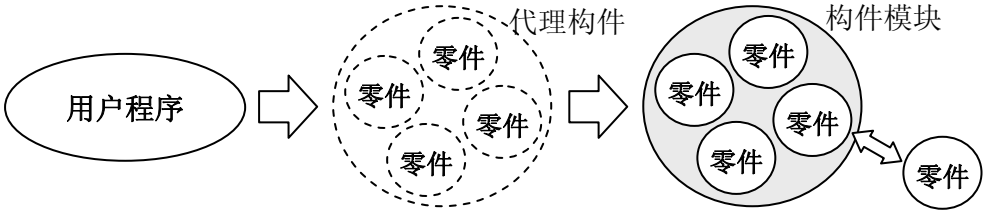

图 2.10 面向中间件的编程模型

在这样的发展过程中，人们逐步深化了对大规模软件开发所需的科学模型、网络环境下软件运行必要机制的理解，使软件技术达到了更高的境界，实现了：

- 构件的相互操作性。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。
- 软件升级的独立性。实现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。
- 编程语言的独立性。不同的编程语言实现的构件之间可以实现互操作。
- 构件运行环境的透明性。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。同时提供系统运行的安全、保护机制。

CAR 技术技术就是在总结面向对象编程、面向构件编程技术的发展历史和经验，为更好地支持面向以 Web Service（WEB 服务）为代表的下一代网络应用软件开发而发明的。CAR 很大程度地借鉴了 COM 技术，保持了和 COM 的兼容性，同时对 COM 进行了重要的扩展。

2.3.2 CAR 构件技术概要

CAR构件技术是面向构件编程的编程模型，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。

CAR兼容微软的COM。但是和微软COM相比，CAR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的用户界面进行了简化包装，易学易用。

从上面的定义中，我们可以说CAR是微软COM的一个子集。CAR很大程度地借鉴了COM技术，保持了和COM的兼容性，同时对COM进行了重要的扩展。在和欣SDK工具的支持下，使得高深难懂的构件编程技术很容易被C/C++程序员理解并掌握。CAR中的“ez”源自与英文单词“easy”，恰如其分地反映了这一特点。

CAR技术是在总结面向对象编程、面向构件编程技术的发展历史和经验，为更好地支持面向Web Service（WEB服务）的下一代网络应用软件开发而开发的。

CAR的编程思想是“和欣”技术的精髓，它贯穿于整个技术体系的实现中。

为了在资源有限的嵌入式系统中实现面向中间件编程技术，同时又能得到C/C++的运行效率，CAR没有使用JAVA和.NET的基于中间代码—虚拟机的机制，而是采用了用C++编程，用和欣SDK提供的工具直接生成运行于和欣构件运行平台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、中间件编程的技术。在不同操作系统上实现的和欣构件运行平台，使得CAR构件的二进制代码可以实现跨操作系统平台兼容。

2.3.3 CAR 技术的意义

对于面向WEB服务的应用软件开发，以及开发操作系统这样的大型系统软件而言，采用CAR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。
- 实现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。同时提供系统运行的安全、保护机制。

- CAR 构件与微软的 COM 构件二进制兼容，但是 CAR 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；
- CAR 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，CAR 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。

2.3.4 CAR 技术对软件工程的作用

大型软件的生产，已经不是传统的手工作坊式的开发所能够胜任的。所谓的软件工厂化生产，是软件工程研究几十年来追求的理想。实现这样的目标，需要有一个科学的编程模型，为所有参与项目开发的软件工程师开发的软件能够正确地对接运行提供技术上的保障。这些软件可以是由一个公司的不同部门提供，也可以是不同企业，不同地点、不同时间生产的。COM技术在微软经过十多年的应用与提炼，被证明是行之有效的编程模型，基于COM技术，微软造就了Windows NT以及配套应用软件这样的大型软件产品。汲取了COM技术的精华，并对其进行了扩展和简易包装的CAR技术，将成为软件工厂化生产的利器。

CAR的重要特点就是：构件的简易化包装；构件的相互操作性；软件升级的独立性；编程语言的独立性；进程运行透明度。在实际的编程应用中，CAR技术的优势体现在以下方面：

（1）易学易用

基于COM的构件化编程技术是大型软件工程化开发的重要手段。但是微软COM的繁琐的构件描述体系令人望而生畏。CAR的开发环境和欣SDK提供了结构简洁的构件描述语言和自动生成辅助工具等，使得C++程序员可以很快地掌握CAR编程技术。

（2）可以动态加载构件

在网络时代，软件构件就相当于零件，零件可以随时装配。CAR技术实现了构件动态加载，使用户可以随时从网络得到最新功能的构件。

（3）采用第三方软件丰富系统功能

CAR技术的软件互操作性，保证了系统开发人员可以利用第三方开发的，符合CAR规范的构件，共享软件资源，缩短产品开发周期。同时用户也可以通过动态加载第三方软件扩展系统的功能。

（4）软件复用

软件复用是软件工程长期追求的目标，CAR技术提供了构件的标准，二进制构件可以被不同的应用程序使用，使软件构件真正能够成为“工业零件”。充分利用“久经考验”的软件零件，避免重复性开发，是提高软件生产效率和软件产品质量的关键。

（5）系统升级

传统软件的系统升级是一个令软件系统管理员头痛的工程问题，一个大型软件系统常常是“牵一发而动全身”，单个功能的升级可能会导致整个系统需要重新调试。CAR技术的软件升级独立性，可以圆满地解决系统升级问题，个别构件的更新不会影响整个系统。

（6）实现软件工厂化生产

上述几个特点，都是软件零件工厂化生产的必要条件。构件化软件设计思想规范了工程化、工厂化的软件设计方法，提供了明晰可靠的软件接口标准，使软件构件可以像工业零件一样生产制造，零件可用于各种不同的设备上。

（7）提高系统的可靠性、容错性

由于构件运行环境可控制，可以避免因个别构件的崩溃而波及到整个系统，提高系统的可靠性。同时，系统可以自动重新启动运行中意外停止的构件，实现系统的容错。

（8）有效地实现系统安全性

系统可根据构件的自描述信息自动生成代理构件，通过代理构件进行安全控制，可以有效地实现对不同来源的构件实行访问权限控制、监听、备份容错、通信加密、自动更换通信协议等等安全保护措施。

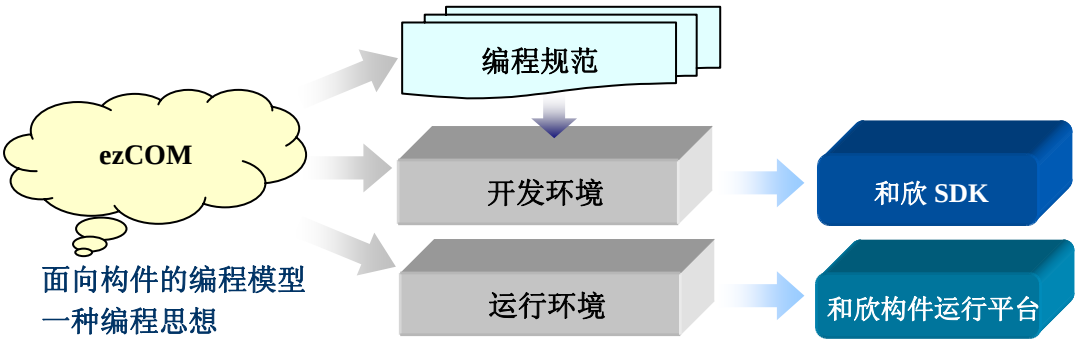

图 2.11 面向中间件的编程模型

2.4 浏览器引擎概述

综观嵌入式系统的发展，大致经历了 3 个阶段。第一阶段是以单芯片为核心的可编程控制器形式的系统，同时具有与监测、指示设备相配合的功能。这种系统大部分应用于工业控制系统中，一般没有操作系统的支持。第二阶段以嵌入式实时操作系统为标志。这一阶段系统的主要特点是：嵌入式操作系统能运行于各种不同类型的微处理器上，操作系统内核精小、效率高，并且具有高度的模块化和扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口（API），嵌入式应用软件丰富，但与互联网无关。第三阶段是以基于Internet为标志的嵌入式系统，目前大多数嵌入式系统还孤立于Internet之外，但随着Internet的发展，Internet技术与信息家电、工业控制技术等结合日益密切，嵌入式设备与Internet的结合将代表着嵌入式技术的真正未来，而嵌入式浏览器正是其中必不可少的一环[11]。

在因特网时代新的应用模式下，浏览器已经不仅是显示网络页面信息的界面，它包含诸如解释网络页面信息，通过对组件中元数据的直接解释执行，与底层的系统软件协同调用相关的组件等功能。基于网络标准的浏览器可以成为强有力的工具，实现跨平台的应用，实现丰富多彩的用户界面和传统桌面系统的应用程序所能完成的功能。实质上，浏览器已经成为操作系统不可缺少的组成部分。

嵌入式浏览器还没有一个准确、严格的定义，但从其可以完成的功能来看，可以从两个方面进行描述。第一，它必须是一个网络信息浏览器，必须支持Http或者WAP等其它传输协议、支持HTML、JavaScript或扩展XML、WML等标记语言，可以完成网页的浏览功能；第二，这个浏览器必须适合在非PC的嵌入式信息设备中存在、运行并完整实现通讯传输协议、标记语言所规定的功能，必须能够根据嵌入式设备的多样性需要而方便地进行裁减和修改，并满足信息设备使用者对获取文字、图像、声音、视频等信息的需求[11][12][13][14][15][16]。

嵌入式浏览器不同于普通浏览器就在于，嵌入式浏览器需要满足嵌入式平台的需求，嵌入式平台在硬件和软件方面都有一定的要求，因此对于嵌入式浏

览器来说，不仅需要满足浏览器的功能，还需要限制自身对运行环境的要求。但是随着嵌入式领域的发展，出现了一些高端的嵌入式设备，比如机顶盒，智能手机等等，对用户体验提出了更高的要求，要求嵌入式软件更好的满足要求。

在未来的和欣平台上，浏览器引擎将直接运行于操作系统内核之上，操作系统和组件技术、浏览器技术紧密结合，为网络应用提供高效率的运行平台。浏览器引擎支持业界广泛使用的Web标准（HTML 4.0、XML、JavaScript等）、具有完备的浏览器功能；通过用户接口语言可以开发跨平台、跨设备的用户接口；通过跨平台的CAR实现可扩展的图形体系结构。比如支持同一用户图形软件完成远程图形功能（类似X-Window）或在同一进程内的高速图形显示；同一段用户图形组件代码可以不经修改，运行在硬件上或台式机的网页里。

2.4.1　HTML

HTML语言是超文本标记语言（Hyperlink Markup Languange）的缩写，它基于SGML（标准通用标记语言，Standard General Markup Language ）语言，由W3联合会推出。HTML语言是超文本标记语言（Hyperlink Text Markup Language）的缩写。它是一种描述文档结构的语言，而不能描述实际的表现形式。HTML语言使用描述性的标记符（称为标签）来指明文档的不同内容。标签是区分文本各个组成部分的分界符，用来把HTML文档划分成不同的逻辑部分（或结构），如段落、标题和表格等。标签描述了文档的结构，它向浏览器提供该文档的格式化信息，以传送文档的外观特征。

用HTML语言写的页面是普通的文本文档（ASCII），不含任何与平台和程序相关的信息，它们可以被任何文本编辑器读取。HTML文档包含两种信息：

页面本身的文本

表示页面元素、结构、格式、和其它超文本链接的HTML标签。

HTML标签规定Web文档的逻辑结构，并且控制文档的显示格式，也就是说，设计者用标签定义Web文档的逻辑结构，但是文档的实际显示则由浏览器来负责解释。我们可以使用HTML标签来设置链接、标题、段落、列表和字符加亮区域等等。大部分HTML标签是这种形式的：

〈标签名〉相应内容〈 / 标签名〉

标签的名字用尖括号括起来。HTML标签一般有起始标签与结束标签两种，分别放在它起作用的文档两边。起始标签与结束标签非常相似，只是结束标签

在“〈”号后面多了一个斜杠“ / ”。后面将会看到，某些HTML元素只有起始标签而没有相应的结束标签，例如换行标签，由于不包括相应的内容，所以只使用〈BR〉就可以了。还有一些元素的结束标签是可以省略的，如分段结束标签〈/〉、列表项结束标签〈/LI〉、词语结束标签〈/DT〉和定义结束标签〈/DD〉等等。

标签名不区分大小写，但是我们建议使用大写字母，这样标签可以更容易从文本中分辨出来。

起始标签中可以包含属性（attribute）域，其位置是从标签名之后空一格的地方开始，在结束符（〉）之前结束。属性域向客户端提供了关于页面元素内容以及如何处理的附加信息。

2.4.2 XML

表面上, XML看起来像HTML. 两者都从标准通用标记语言(SGML)起源。产生HTML的工具能常常被再用来产生XML。XML在两个重要区域中是和HTML不同: 句法和语义.

- XML句法

HTML和XML使用 <, >, 和 & 创造要素和属性结构. 在HTML浏览器接受或者忽视混合标记语言时，XML语法分析器建立在这些分析器上的应用程序比较严格。在XML句法中的错误停止文件处理, 用户或者应用得到错误启示, 而不是对文档结构进行猜测。

XML文件必须遵守标记文件部分和创造嵌入要素结构的规定. 在XML文件中要素不能重叠. 如果要素的起始标签是在另一要素以内,它必须去同一包含要素中中止。例如, 下列的HTML代码表示了黑体和斜体字的混合结构。

This is bold text. This is bold italic text. This is italic text.

在一些HTML浏览器中它会正常显示。在一位XML从语法上分析上,就行不通了。为了取得在XML中同样的格式, 可以使用下列的句法.

This is bold text. This is bold italic text. This is italic text.

这为XML文件创建者的额外工作提高了互操作性。XML同样对句法的其它方面要求严格.所有的属性值一定是加以引用的，你不能在你文件的文本内使用<,>或者&，而要用<, >或&代替。

XML语义

虽然XML关于句法是严格的, 它为开始者在XML文件中明确定义提供更多选项.<b>总是对HTML处理器有同样的意义.用XML, 你能建立你的自己标记词汇或者在各种各样的适合于你的工业或者项目类型提价标记词中挑选一个.图表和文件类型定义(DTDs)让你描绘这些词汇, 但是你能也建立文件没有正式规定使用的词汇. 使用Namespaces帮助你分辨认出使用的词汇。

这个方法要求适应各种不同浏览器要求的结构。开发者不能指望XML应用懂得他们的标记意味着什么，并理解这些标记。 浏览器能使目前XML表示出来, 但是要求style sheet以CSS或XSLXSL Transformations(XSLT)来规范用户的说明。一些浏览器，包含IE5.0+, 包含默认的style sheet，但它主要用于诊断而不是用于最终用户设计。

XML应用也能使他们的自己逻辑进入XML词汇,而不单纯依赖style sheets。这逻辑可以采取简单脚本的形式或绑定到特殊的表示模型，或者它可以涉及编写整个应用程序。这些应用利用内建的包含在XML文件的标签结构来处理在那些文件中信息, 传递给用户, 把他们与其它数据来源连接起来,或者重定向它们到其它适合用户。

- **XML 格式的优势**

在某一方面，XML 是仅仅另一数据格式，在其它方面，XML 比其它格式有几个重要优势，它可以更有效地存储信息。

XML 允许开发者建立他们的属于自己的保存信息的标记结构.

XML 解析语法是非常明确，而且是一种广泛应用的工具，它能从在各种各样的环境中 XML 文件使获得知识.在 Unicode 基础的基础上建立 XML 使它更容易建立使国际化文件.

应用能依赖 XML 分析器确定结构的可靠性，以及进行数据类型检查

XML 格式置于文本使他们变得更有阅读,更容易用文件保证其有效性，更容易纠正错误。

XML 文件能够利用大部建立在浏览器中的资源。

XML 是并非适合于每一个环境，XML 文件比他们取代的二进制格式的更哆嗦.他们需要更多网络带宽和存储空间，需要更多的处理器时间，但是，优秀应用程序设计能避免某些问题.

2.4.3　DOM[20][21]

DOM 全称是 Document Object Model(文档对象模型)。假设把你的文档看成一个单独的对象，DOM 就是如何用 HTML 或者 XML 对这个对象进行操作和控制的标准。

作为结构的 DOM

DOM 是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中导航仪寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。

对于特别大的文档，解析和加载整个文档可能很慢且很耗资源，因此使用其他手段来处理这样的数据会更好。这些基于事件的模型，比如 Simple API for XML（SAX），适用于处理数据流，即随着数据的流动而依次处理数据。基于事件的 API 消除了在内存中构造树的需要，但是却不允许开发人员实际更改原始文档中的数据。

另一方面，DOM 还提供了一个 API，允许开发人员添加、编辑、移动或删除树中任意位置的节点，从而创建一个引用程序。

解析器是一个软件应用程序，设计用于分析文档（这里是指 XML 文件），以及做一些特定于该信息的事情。在诸如 SAX 这样基于事件的 API 中，解析器将向某种监听器发送事件。在诸如 DOM 这样基于树的 API 中，解析器将在内存中构造一颗数据树。

作为 API 的 DOM

从 DOM Level 1 开始，DOM API 包含了一些接口，用于表示可从 XML 文档中找到的所有不同类型的信息。它还包含使用这些对象所必需的方法和属性。

Level 1 包括对 XML 1.0 和 HTML 的支持，每个 HTML 元素被表示为一个接口。它包括用于添加、编辑、移动和读取节点中包含的信息的方法，等等。然而，它没有包括对 XML 名称空间（XML Namespace）的支持，XML 名称空间提供分割文档中的信息的能力。

DOM Level 2 添加了名称空间支持。Level 2 扩展了 Level 1，允许开发人员检测和使用可能适用于某个节点的名称空间信息。Level 2 还增加了几个新的模块，以支持级联样式表、事件和增强的树操作。

第 3 章　Everest 的体系架构设计

这里首先介绍我们对嵌入式浏览器的研究成果，指出嵌入式浏览器设计和实现中需要注意的问题。然后在 3.2 中根据我们进行的研究，结合和欣操作系统的特点，提出和欣浏览器架构的设计原则，并给出我们的设计思路。在 3.3 中我们介绍和欣上浏览器架构的主要特点，这些特点同时就是我们提供的嵌入式浏览器中主要问题的解决方案。在 3.4 中我们将介绍基于该框架在 Elastos/Atlas 平台上实现的浏览器。

3.1　嵌入式浏览器框架结构的研究

通过分析浏览器的应用，并对一些已有的浏览器进行研究[11][13][15][16][22][23][24]，可以发现浏览器主要提供通过解析请求取回数据，并进行解析显示的功能，同时还将提供一定的用户和页面交互的能力。进行归纳总结，可以详细描述为以下几大功能模块：

1. 通讯模块(Communication Modular)：负责解析用户发出的请求，并根据请求利用相关的协议(http, ftp, file 等)取回请求的数据，也可以提供缓存功能；
2. 语法解析模块(Parser Modular)：对取回的数据根据数据类型进行解析生成中间数据结构，主要解析的数据类型有 HTML 文档，各种图片等；
3. 布局模块(Layout Modular)：通过对语法解析生成的中间数据进行操作，一方面获取一些平台相关的布局单位的信息，比如根据字体的信息计算文本的尺寸等，同时根据这些布局信息，依据一定的布局的算法，对页面上的元素进行布局；
4. 显示模块(Render Modular)：根据页面的布局信息和页面的内容，对最终结果进行显示；
5. 脚本语言支持模块(Script Modular)：通过脚本语言的解析，一方面影响页面的布局，另一方面负责用户和页面的交互，脚本语言主要是 JAVASCRIPT；
6. 插件模块：通过插件提供浏览器功能的扩展。

其中 1-4 模块属于基本功能模块，5-6 模块属于功能扩展模块。在 1-4 模块中通讯模块和其他三个模块耦合较少，主要负责取数据，而其他三个模块分别对应对取来的数据进行分析的三个阶段，因此数据交互比较多，如何进行模块之间的数据交互是进行模块化时需要考虑一个主要问题。功能模块的划分，各个浏览器区别不大，浏览器之间区别体现在体系结构上，其中包括模块之间的耦合度，模块之间的通讯方式。还有对模块的加载方式等，就是功能模块是动态加载的，还是静态绑定的等。

目前在桌面浏览器上，浏览器发展的趋势是构件化，动态加载等[25][26][27]。其中Firefox、

Grand-Rapid，ICEbrowser等在这方面做了很好的工作，这同时是嵌入式浏览器的未来发展方向。综上考虑，我们认为嵌入式浏览器的框架设计中，在满足基本功能的情况下，需要考虑得主要问题就是浏览器模块化的处理，其可以归结为以下两个具体问题：

1. 模块之间的数据交互的问题，由于网络模块比较独立，因此该问题主要是在数据解析，布局和显示模块体现出来；
2. 模块之间的通讯，模块之间的行为需要同步等。

而为了解决第一个问题，目前浏览器大部分采用 DOM 机制。通过把浏览器中的中间数据结构用 DOM 来描述，而模块操作中间数据结构通过 DOM 提供的 API 来进行。采用 DOM 的好处在于，其本身是 W3C 推荐的标准，利于扩展，其次通过数据模块的抽象，其他模块被抽象为更纯粹的浏览器的功能模块，同时各个模块数据的交互通过 DOM 来进行，有效减少了模块相互之间的数据交互，从而大大降低个功能模块之间的耦合性。

对于第二个问题，通用的解决办法是设置一个浏览器控制中心，由它来控制和协调模块之间的交互，模块之间尽量减少通讯，模块之间的通讯主要依靠浏览器控制中心来完成，而浏览器控制中心为了降低自身和其他模块的耦合，主要通过消息机制或回调机制等来控制模块之间的通讯。

嵌入式浏览器主要应用在嵌入式设备上，在设计的时候还需要满足嵌入式领域的要求，具体来说就是应该考虑软件的尺寸，运行效率等问题。好的嵌入式浏览器体系结构应该简单，灵活，易于扩展，并且适合于二次开发。一方面尺寸应该达到一定的要求，另一方面效率也是需要考虑的一个重要问题。

3.2　Everest 体系架构的设计

3.2.1　设计原则

和欣操作系统是提供网络服务编程平台的嵌入式操作系统，我们认为基于该平台浏览器架构应该具有如下特性：

1. 简单。由于在嵌入式操作系统上，简单的实现有利于控制浏览器的大小，并且有助于人们理解浏览器技术，方便地进行扩展；
2. 好的模块化。作为构件平台上的浏览器，可以方便的根据需求进行构件化；
3. 良好的扩展性。由于简单的实现，因此需要有好的扩展性，这样才可以满足将来更复杂的扩展性；

在嵌入式操作系统上设计软件，其中的硬件软件环境是需要考虑的问题，考虑到浏览

器作为在操作系统和图形系统上的应用，对硬件平台的适应性主要由下层软件来提供，而且 Elastos 本身既可以运行于 X86，ARM，MIPS 等单片机上，也可以运行在 Windows，Linux 等桌面操作系统上，而浏览器只要满足在 Elastos 平台上运行即可。另一个需要考虑的是软件环境问题，在 Elastos 上，目前可以使用的编程语言主要包括 C/C++，而 C 语言可以更容易的控制代码的尺寸及效率，因此初步使用 C 语言来实现，对于进一步的构件化或对 CAR 的扩展，C++是更好的选择。

浏览器的研究和实现是一个循序渐进的过程，目前和欣平台上已经提供了较为成熟的操作系统以及图形系统，其中构件技术，网络技术都已成型，其自滚动下载等理论体系已经相对完善。因此在满足基本功能的基础上，简单和易扩展的浏览器是一方面可以满足目前的需求，另一方面也更容易实现，为下一步浏览器技术的继续研究以及浏览器技术的扩展打下基础。

3.2.2　架构的设计

综合以上分析，我们设计了和欣操作系统上的浏览器框架，并且在 Elastos/Atlas 上基于该框架实现了浏览器 Everest。参见图 3.1

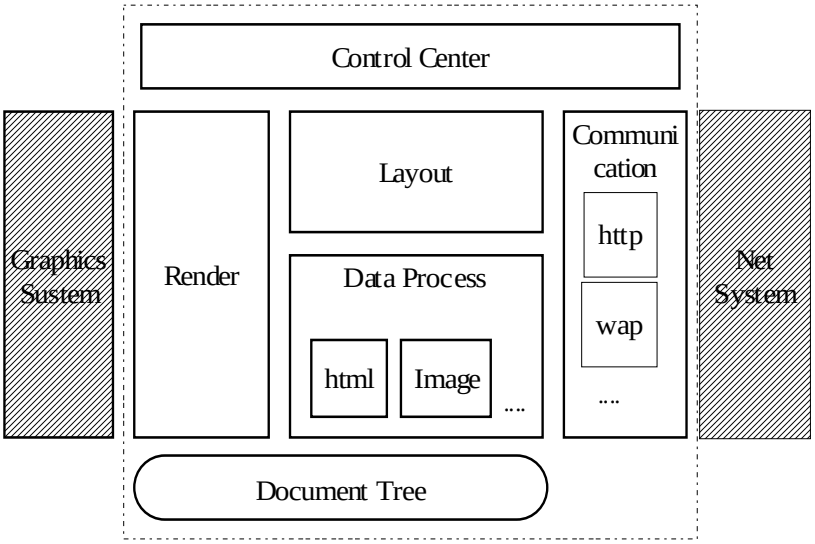

图 3.1　浏览器框架结构图

模块主要包括文档对象树（Document Tree），数据处理（Data Process），布局（Layout），绘图（Render），通讯（Communication），控制中心（Control Center）模块。这些模块包含了浏览器的基本功能模块，并且参照了目前的主流浏览器的设计，把文档对象树抽象出来，该模块是数据处理、布局和显示模块进行数据交换的基础，通过抽象出该模块，一方面可

以很好的进行浏览器内容的管理，另一方面可以很方便扩展为 DOM 模型。而网络模块和其他模块之间耦合比较小，比较独立。我们把除网络模块之外的其他几个模块统称为浏览器引擎（Browser Engine）。

在模块间通讯主要采用了消息机制，有效减少了模块的耦合，而降低模块之间的耦合度也是该架构设计时考虑的地方，下文中将详细介绍。

图 3.2 表示 Everest 的体系结构，是基于上面介绍的浏览器的体系结构进行实现的，其中 Browser Window 和 Scope Window 部分对应体系结构中的 Control Center，而 Browser Engine 则包含了浏览器中语法解析，布局和显示等功能模块。

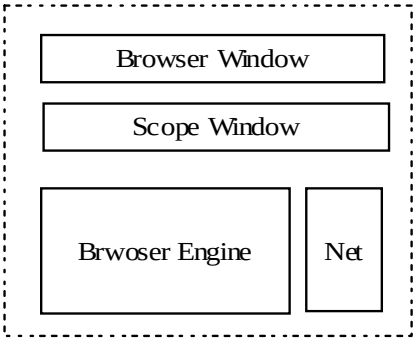

图 3.2　Everest 的体系结构

图 3.3 介绍了 Everest 工作的流程图。首先由用户进行地址输入，由 Browser Window 截获该消息，并向客户端窗口 Scope window 发出打开链接的消息，Scope Window 接到相应的消息回会调用相应的 Browser Engine 提供的 API，进行相应的处理。最后 Browser Engine 会把网页结果内容输出到 Scope Window 中，并会把一些状态信息输出到 Browser Window（目前并未实现），并呈现给最终用户。如果页面中有链接或者按钮，当用户进行点击，Scope Window 会截获这些消息，并把相应的消息转发给 Browser Engine 进行相应的处理，比如提交表单等等。

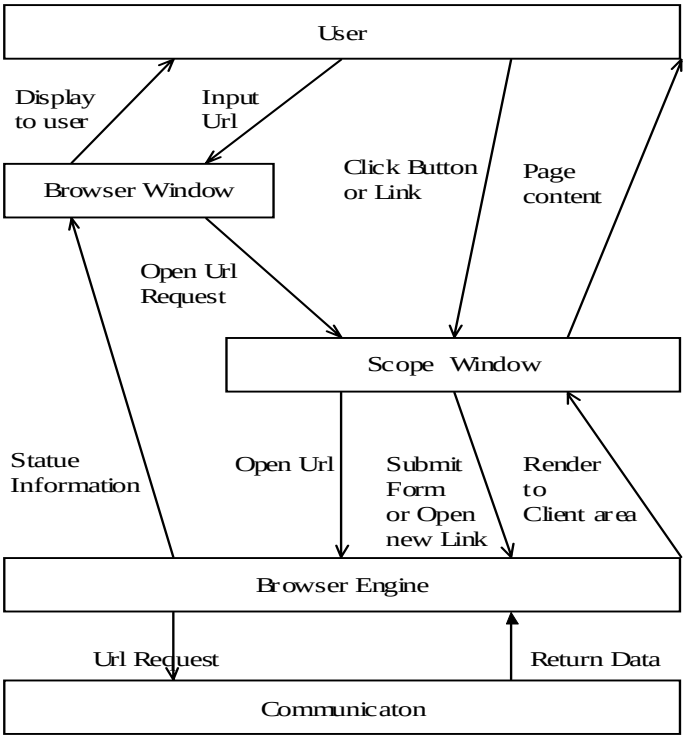

图 3.3　Everest 工作流图

下面对各个节点进行简要说明：

1. Browser Window：是主窗口，也是浏览器消息的总调度处。窗口包括菜单和输入框，还有相应的前进后退按钮等。一方面负责取得用户输入，另一方面回显一些状态信息，比如浏览器访问进度等等。该窗口负责接受用户所有的消息，比如鼠标键盘等消息并进行分发，用户消息也可以发送到这里来进行处理；
2. Scope Window：文档显示窗口，窗口大小为 Browser window 客户端区域大小，主要对 Browser window 发来的消息进行处理，并把消息映射为 Browser Engine 可以识别的消息并进行响应，同时负责显示网页的输出内容，并控制响应用户同网页上的控件交互；
3. Browser Engine：浏览器引擎，负责对用户的输入进行解析，并取出数据，对数据进行分析并将最后结果输出到 Scope Window，同时会将一些状态信息输出到 Browser Window 上。

3.3　Everest 体系架构设计的特点

该体系结构是在研究其他浏览器技术及和欣平台的基础上进行设计的，考虑了嵌入式应用

的需求，也结合了和欣作为网络操作系统的特点。满足了 3.2.1 中提出的设计原则中提出的需求。经 3.1 中的分析，可以了解目前的浏览器中模块的结构及模块之间的通讯是浏览器需要考虑的主要问题，下面我们将通过介绍如何解决这几个问题来介绍浏览器的主要特点。考虑到通讯模块比较独立，因此模块之间的通讯，我们分两部分介绍，一部分介绍模块内部的通信，一部分介绍引擎同通讯模块之间的通讯

3.3.1　良好的模块化特性

作为嵌入式浏览器，选择了基本的功能模块作为体系结构的基础，而且在实现中，这些基本的功能模块满足了应用的要求。同时对这些功能模块进行了很好的模块化，在第 4 章我们将结合实现的浏览器对各个模块细节进行介绍。

具有良好的扩展性，一方面通过抽象出文档对象树，使得浏览器可以方便地扩展为 DOM 模型，进一步支持 JavaScript；另一方面，模块之间采用消息机制来进行通讯，有效降低了模块之间的耦合，并且这种松耦合状态，使得可以很方便的添加对新的数据类型的支持；同时，通过扩展机制，有效地把 CAR 构件融合到浏览器中，使得 CAR 构件可以方便地在浏览器中支持。

3.3.2　Browser Engine 内部的模块通讯机制的设计

在 Browser Engine 内部模块通讯是利用消息机制来进行的。考虑到代码尺寸和效率的问题，我们选择了 C 语言来实现浏览器引擎，而由于引入了文档对象树，因此需要在 C 语言的基础上提供对象框架，这里我们参考了 GObject 的实现，利用 C 语言实现自定义的对象支持框架，该框架被证明是灵活的、实用的。由于利用 C 来实现，有效的控制了代码的行为和尺寸，但是同时也带来了开发和维护的问题，可以考虑以后利用 C++机制进行替代和选择。下面首先介绍内部通讯机制的基础，基于 C 实现的对象框架以及事件机制，然后介绍通讯机制的实现。

- **基于 C 的对象支持框架**

类型机制提供了基于C的面向对象的框架，提供类的注册和管理机制[28]。管理系统会在程序运行的开始启动。每个对象在创建的时候，类型系统会检查是否该类存在，如存在则根据已经注册的类信息进行对象的创建，否则，将先对未注册类的类信息进行注册，并在注册信息里添加并维护类之间的继承关系信息。类型系统也提供了API访问某个对象的类型信息，用来动态获取对象的类型信息。

每个对象主要包括两个数据结构，一个数据结构存储类型的属性，一个接口结构指明该类型提供的接口函数，代表类的行为，这个数据结构是共享的，每个类仅有一个这样的数据结构，数据结构内存布局图见图 3.4。每个类的继承均同时指数据结构的继承和接口结构的继承。每个类的数据结构包含一个父类的数据结构，而其对应的接口结构也包含父接口结构，从而保证了子类型对父类型属性和接口的继承。所有的类均继承自 Object 类，其中包括 Object 数据结构和 Object 接口结构，而在 Object 结构中包含一个指向 Object 接口结构的指针，这样可以通过对象的数据结构访问到接口结构，因此文档对象树只需要保存一个类实例指针也就是数据结构指针即可。

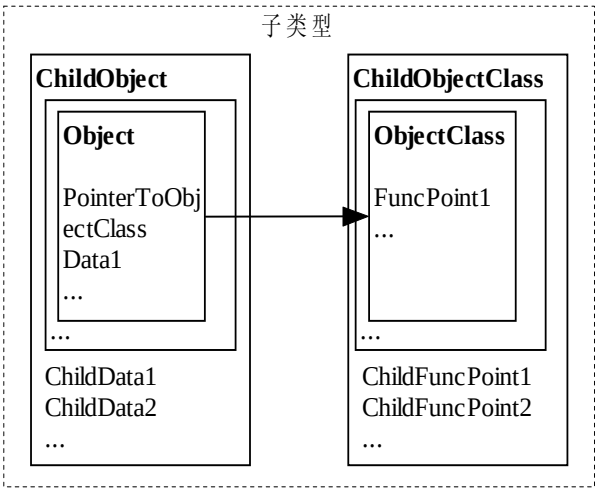

图 3.4　子类型数据结构内存布局图

由于每个类提供的是接口，而且仅提供接口，因此可以通过动态对对象接口函数赋值定制一个对象的行为。这种特性使得可以方便的控制浏览器的行为。

下面介绍一些主要的数据结构和相应的 API 函数。

Object 为基础类，所有类型都会直接或间接从 Object 中继承下来。Object 包含两部分数据结构，一部分包含属性内容，另一部部分包含相应的接口函数指针，详细说明如下：

```
typedef struct _GTypeClass GTypeClass;
typedef struct _GTypeInstance GTypeInstance;
struct    Object
{
    GTypeInstance g_type_instance;
    /*< private >*/
    guint           ref_count;
    GData           *qdata;
    int             floating;
};
struct _GTypeInstance
{
    GTypeClass *g_class;
};
struct _GTypeClass
{
    GType g_type;
};
```

说明：

其中 g_type_instance 指向相对应的接口函数指针数据结构，属性数据结构和接口函数指针数据结构通过这个成员变量联系在一起，所有从 Object 类继承出的类同理；ref_count 为引用计数值；floating 是虚计数标记，由于在浏览器中控制类的生存使用了引用计数机制，而对于引用计数机制，要明确的进行 ref 和 unref 配对操作。但是在浏览器中，页面元素创建和销毁不在同一模块进行，而创建的时候，其 ref_count 被置为 1，否则会被销毁，而在以后进行的操作中 ref 和 unref 是完全配对的，而这时会出现问题，就意味着对象被销毁时，其引用计数还是 1，导致无法销毁，这个计数 1 被称为虚计数。为了解决这个问题，增加了虚计数开关，当创建对象的时候，先打开虚计数开关，当其他对象引用该对象时首先调用 g_object_ref 进行计数增加，然后调用 g_object_sink 进行虚计数处理（详见下面的关于 g_object_sink）的说明。从而保证最后对象可以被销毁。

```
void g_object_type_init (void)：
```

说明：

在调用 g_type_init 函数时会调用，主要功能就是注册并初始化 Object 类。

```
gpointer g_object_new (
                GType object_type,
    const gchar       *first_property_name,
    ...):
```

说明：

用来创建一个对象第一个参数是已注册的对象标识 ID；第二个参数表示后面参数的数量，如果为 0，则没有第三个参数。目前在使用中调用时第二个参数都是 0，因此此函数已经简化处理。

```
gpointer g_object_ref(gpointer _object):
```

说明：

对对象的引用计数加 1。

```
void g_object_unref(gpointer _object):
```

说明：

对象的应用计数减 1，当引用计数为 0 时把对象的资源进行回收。

```
void g_object_sink(GObject *object)
```

说明：

对对象的虚计数进行处理，当虚计数开关被设置时，对计数减一，同时关闭虚计数开关。从而保证了虚计数被消除。

- **自定义消息机制**

浏览器消息机制提供消息注册、关联和激活功能。消息机制主要支持布局消息和绘图消息以及超链接相关消息和对按钮类点击消息等，引入该消息机制减少了浏览器对底层图形系统的消息或事件机制的依赖。

1．接口注册。提供 API 将类接口注册到相应的消息上。子类只需要负责注册自己的接口到相应的消息即可，由于父类会注册自己的接口到相应的消息上，同时子类继承了该接口，这样子类也会对相应的消息进行响应。同时子类可以修改继承来的接口，使其指向不同的函数，从而定制对消息的响应的行为。

2．消息关联。关联特定的对象实例和回调函数到指定的消息上。这为特定的实例响应消息提供了可能，不同于上面介绍的消息注册，接口注册提供的是某一类的接口对消息的注册，意味着，该类的所有实例都将响应注册过得消息，而消息关联只是将某个特定的实例关联到响应的消息上。当消息被激发时，只有该实例将回对消息响应，会调用注册的回调函数。一般通过关联管理的消息是用来管理链接以及管理点击事件的消息。

3．消息的激活。通过发出消息给消息管理系统，后者将负责激活注册的函数。发消息时要求提供对象句柄及消息对应函数所需参数，部分参数在注册的时候也可以指定。被激活的函数是该对象所属类或其父类注册在指定消息上的接口函数以及该对象曾经关联在指定消息上的回调函数。

4．消息的注销。负责把特定对象从消息中移除，避免消息系统维护太多的信息，导致效率下降。

下面首先我们介绍比较重要的数据结构。
消息的种类：

```
enum
{
	SIZE_REQUEST = 0,
	FAST_SIZE_REQUEST,
	SIZE_ALLOCATE,
	SET_WIDTH,
	SET_ASCENT,
	SET_DESCENT,
	DRAW,
	REALIZE,
	UNREALIZE,
	DESTROY,
	BUTTON_PRESS_EVENT,
	BUTTON_RELEASE_EVENT,
	MOTION_NOTIFY_EVENT,
	ENTER_NOTIFY_EVENT,
	LEAVE_NOTIFY_EVENT,
	LINK_ENTERED,
	LINK_PRESSED,
	LINK_RELEASED,
	LINK_CLICKED,
	CLICKED,
	CLICKED_AT,
	LAST_SIGNAL
};
```

其中前 15 个为 Widget 的消息，也就是所有的对象都拥有的消息。{LINK_ENTERED..LINK_CLICKED}为 Page 和 Image 对应的消息。{CLICKED，CLICKED_AT}为 Button 对应的消息。
下面我们介绍主要的接口：

```
guint g_signal_new (const gchar *signal_name,
        GType itype,
        GSignalFlags signal_flags,
        guint class_offset,
...)
```

参数：

signal_name　消息对应的字符串，是消息的 ID。

Itype　　注册消息的数据结构的类型。

signal_flags　新注册消息的标记，表明其级别以及其他一些属性。

class_offset　新注册消息对应函数在行为数据结构中的偏移。

返回值：　该消息在系统中注册的数字 ID

说明：首先根据字符串取出消息在数组中的下标，然后调用函数判断是否该消息已注册，如注册，则返回，如未注册，则把消息添加到队列里。

```
long g_signal_connect_data(gpointer instance,
                           const gchar *detailed_signal,
                           GCallback    c_handler,
                           gpointer    data,
                          ...)
```

参数：　*instance*　需要关联到特定消息的实例的地址指针。

detailed_signal　指定消息的字符串 ID。

c_handler　　需要关联到该消息的处理函数。

Data　　需要关联的该消息的参数。

返回值：返回该函数连接到的消息的 ID。

说明：关联的过程同注册的过程类似，只是在细节有区别。首先这里判断是否已存在对应的消息函数对，需要判断 *instance, c_handler* 参数是否同函数节点中的保留值相等，如相等，还需判断是否该节点已被标为删除等。如存在则返回，如不存在则将新节点加到函数链表的末尾。该函数将只对 2，4 级别的函数链进行操作。该函数同 g_signal_new 不同在于该函数是面向实例的，不是面向对象的，是用来关联某一个实例和消息处理函数及相应的参数到某消息上，当该实例接到其关联的消息时，会调用相应的关联函数。而 g_signla_new 中注册的消息，某实例是否接受该消息的判断标准是是否该实例所属类注册过该消息。因此对于一个实例，其相应的消息有两种，一个是它所属类通过 g_siganl_new 注册的函数，还有实例本身通过 g_signal_connect 关联的函数。

```
void g_signal_emit(gpointer instance,
    guint signal_id,
   ...)
```

参数：　*instance*　　发出消息的实例的指针。

signal_id　发出的消息的数字 ID。
detail　　未使用。

返回值：无

说明：用户通过调用该函数激发相应的消息，该函数被调用时需要指定函数调用的实参。当某个实例激发消息时，会调用该实例所属类型的消息，和该实例曾经关联到该消息上的函数。

```
void g_signal_destroy()
```

参数：　无

返回值：无

说明：　该函数在整个程序结束时调用，清除所有的链表，释放相应的内存。

3.3.3　内部通讯机制的设计

内部通讯依赖于类型系统和消息系统。对象初始化通过调用 g_object_new 类的消息处理函数的注册仅在第一次实例化类对象的时候进行，因为该类的接口数据结构部分的指针是被所有实例所共享的，只需进行一次初始化。在对接口部分进行实例化的过程中，一方面是调用 g_signal_new 来注册该接口数据结构中的接口函数指针到相应的消息上，比如布局和显示消息等，另一方面是对相应的接口函数指针进行初始化工作。这样当实例收到某消息的时候，就会通过注册的接口函数指针找到对应的处理函数。这样，当数据分析完毕，浏览器控制中心会调用 g_sianal_emit 发出相应的消息给文档对象树上的实例，并根据前面注册到该消息上的接口函数找到对应的相应函数，进行调用，从而完成消息的响应。还有一类事件注册是通过 g_signal_connect 注册的面向实例的响应消息函数，当实例接收到某消息是，先判断是否注册过相应的处理函数，如注册过则调用相应的处理函数，否则忽略该消息。

通过以上讨论，说明了模块之间如何通过消息机制来进行通讯的，但是有一个问题还需要说明，就是消息激活的时机问题。对于消息的注册和关联是在文档对象树构造的时候完成，而消息的注销是在程序退出的时候完成。消息激活主要是靠控制中心来控制，其中对于布局和绘图消息，当控制中心发现文档解析完毕，会发出相关布局消息进行布局；在布局完成后，布局模块发出消息给控制中心要求写屏，控制中心收到消息后，会发出消息给绘图模块进行输出；而当页面显示完成后，控制中心会等待用户的交互，或者将用户的鼠标消息转化为消息系统可以识别的消息，发给文档对象树响应的节点进行处理，或者直接操作子控件完成交互。而控制中心同时是依赖于图形系统及其消息机制，因此控制中心也起到了图形系统消息到自定义消息系统消息映射的作用。

3.3.4　Browser Engine 同网络模块的通讯机制的设计

经过分析我们可以发现，网络模块的负责的功能比较独立，其主要负责接受浏览器控制中心的请求或浏览器引擎的取数据的请求，并对请求进行解析，然后根据不同的协议进行网络通讯取得数据，在取得数据后把数据返回给相应的模块进行处理。参见图 3.5。然而虽然接入点不多，然而发出请求的模块不是一个地方，可能是用户输入，也可能是点击的连接等等，并且请求的数据类型不一致，而如果各个模块都和网络模块以自己的方式直接通讯的话，代码将很难维护。同时这种方案中，好的网络模块应该是根据请求进行多线程服务的，从而是异步响应请求的，这样还需要保存发出请求模块的信息，以便取得数据后可以返回给相应的数据处理函数。然而有些情况下在发出请求的时候，数据类型是不好判断的，或者是未知的。

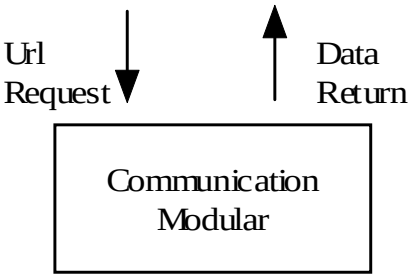

图 3.5　网络接口部分数据流图

针对上述问题，我们提出了利用回调机制进行通讯的解决方案。首先抽象出网络模块接收请求接口，所有的请求通过包装成该接口可以识别的参数，然后通过接口和网络模块进行通讯。

```
int     a_Cache_open_url(void *Web, CA_Callback_t Call, void *CbData);
```

参数：
Web　　对请求进行的包装，包含请求的 URL 的信息。
Call　　回调函数指针，数据达到后将利用该函数指针进行处理。
CbData　传递给 *Call* 的参数，不同数据类型不同。
返回值：在网络处理模块队列中的主键值。
说明：　满足了基本的网络接口的需求，其中 *Call* 和 *CbData* 可以为空。

其次抽象出接受数据模块的入口接口，通过该接口，网络模块可以获得相应的数据处理函数的句柄，从而对数据进行处理。

```
typedef   void   (*CA_Callback_t) (int Op, CacheClient_t *Client);
```

参数：
Op　　操作数，表示取数据的状态，是否已经取得全部数据。
Client　传递给客户的网络的数据，包含缓存指针及缓存大小等
返回值：无
说明：　该接口是数据处理函数的入口，对于新的数据，实现该接口，并注册一个相应的处理函数，用来提供该接口，就可以支持新的数据类型的解析。

具体的处理流程参见图 3.6。各个模块均可以通过调用接口发出网络请求，而浏览器可以处理的数据，需要在开始进行注册，以数据类型和相应的一个处理函数对的形式注册。如果发出请求的时候，了解具体的数据类型及相应的接受数据模块的入口接口，可以作为参数直接传输给网络模块，网络模块负责保存这些数据，当取得数据后，直接调用相应的接受数据模块的入口接口。而对于未知数据类型的数据，在取得相应的数据后，网络模块可以根据协议头判断出数据类型，然后根据数据类型向控制中心请求接受数据模块的入口接口，如果是已注册的数据类型，则调用相应得函数取得相应得数据类型接口并返回给网络模块，从而可以进行处理，如果是未知类型，则返回特定的接口处理函数，该函数将直接返回。

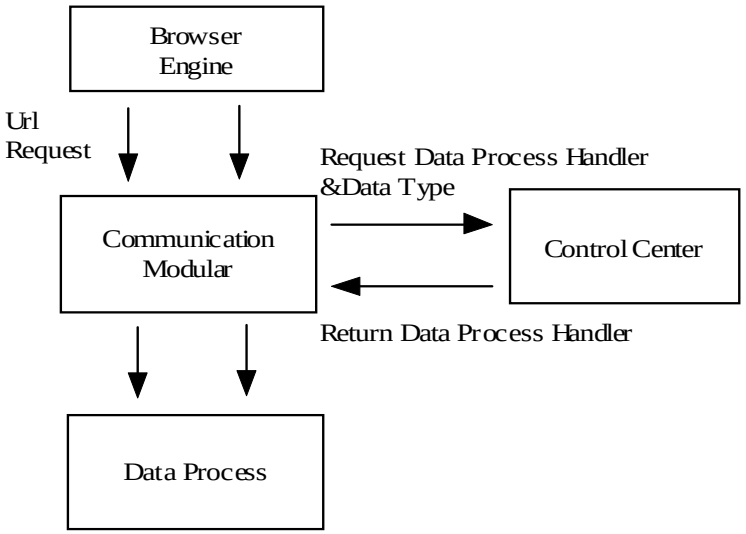

图 3.6　网络模块同其它模块之间通讯机制

这种机制通过抽象出两类接口有效的降低了通讯模块和浏览器其他模块的耦合性。并且增加了灵活性，对新的数据处理函数，只要提供相应的数据类型入口接口即可，只需在控制中心注册数据类型和相关的处理函数的句柄即可。当取得相应类型的数据时，网络模块会调用相应的函数，根据数据类型查用到相应的处理函数句柄，从而利用已经注册好的函数进行数据处理。

下面我们详细该机制的实现。

■　数据类型及相应处理函数的注册

首先需要对需要处理的数据类型和相应的处理函数进行注册，注册通过调用 Mime_add_minor_type 来实现

```
static int Mime_add_minor_type(const char *Key, Viewer_t Method)
```

参数：

Key　　数据类型字符串，作为数据类型的标识。

Method　用来提供回调接口的函数。

返回值：0

说明：　利用该函数进行数据类型和相应处理函数的注册，这样可以根据数据类型找到相应的数据处理函数句柄。其中 *Method* 一方面做一些相关数据类型分析时需要的初始化工作，同时还将返回相应的数据处理函数句柄。

其中接口函数的声明，

```
typedef DwWidget* (*Viewer_t) (const char *Type, void *P,
                               CA_Callback_t *Call, void **Data)
```

参数：

Type　　类型信息，目前并没有使用。

P　　Web 数据结构，是发出请求时已打包好的数据。

Call　　请求的回调函数。

Data　　传递给 *Call* 的参数。

返回值： 文档对象树的节点。

说明： 该函数类型是对数据类型处理函数的抽象，新的数据类型需要提供该函数类型的函数进行注册，并在数据模块取得数据后进行调用。该注册函数一方面进行一定的初始化，另一方面取得数据处理模块的入口接口，并返回给网络模块，网络模块将通过入口接口调用相应的数据处理函数。每个相关类型注册的处理函数具体说明如下，一个是对未进行初始化的中间数据结进行初始化工作；更重要的一点则是返回数据处理函数的入口点句柄，比如对于 HTML 类型的数据来说，就是 HTML parser 的入口点，而对于 Image 来说，是 Image 的解码器。如果不支持某类型，将返回一个空的处理函数 a_Callbakc_null_client，该函数什么也不处理，只是直接返回。关于这部分我们在数据处理模块里将结合实际的数据处理函数进行详细的介绍。

Control Center 用来存储数据类型的数据结构

```
typedef struct {
    const char *Name;      /* MIME type name */
    Viewer_t Data;         /* Pointer to a function */
} MimeItem_t;
```

■ 网络部分入口接口及相关数据类型

网络部分入口声明：

```
int     a_Cache_open_url(void *Web, CA_Callback_t Call, void *CbData);
```

函数说明见前，利用该函数向网络模块发出数据请求，其中 Web 包含一些全局信息，比如对于图片请求来说，包含其在文档对象树节点的位置等。如果已知数据类型，可以直接把回调函数，及相应的数据结构作为参数传递到网络模块，这样当网络模块取到数据的时候可以直接调用相应的处理函数进行能够处理。同时后面两个参数可以为空，这样将通过下一个机制取得相应的函数接口。

■ 根据数据类型取得相应数据处理函数接口

当网络模块取到数据后，如果其数据处理句柄未知，则需要根据不同的类型调用 a_Web_diaptch_by_type 来获取相应的数据处理函数并返回数据处理函数接口。

```
int a_Web_dispatch_by_type (const char *Type, DilloWeb *Web,
                            CA_Callback_t *Call, void **Data)
```

参数：

Type　　类型信息，目前并没有使用。

Web　　Web 数据结构，是发出请求时已打包好的数据。

Call　　请求的回调函数。

Data　　传递给 *Call* 的参数。

返回值： 文档对象树的节点。

说明： 该函数是浏览器控制中心提供给通讯模块使用，当取得数据类型后，通过调用该函数，请求控制中心返回数据处理函数句柄以及相应的初始化工作。

```
DwWidget *a_Mime_set_viewer(const char *content_type, void *Ptr,
                            CA_Callback_t *Call, void **Data)
```

参数：

Type　　类型信息，目前并没有使用。

Web　　Web 数据结构，是发出请求时已打包好的数据。

Call　　请求的回调函数。

Data　　传递给 *Call* 的参数。

返回值：文档对象树的节点。

说明：　该函数是浏览器控制中心提供的，根据参数指定的数据类型，查找到相应的处理函数，并进行调用。并返回对应的文档树的节点。

第 4 章　Everest 的浏览器引擎设计

下面我们将介绍 Everest 的浏览器引擎的实现。首先总体介绍 Everest 的引擎的模块结构及数据流，然后介绍引擎中各个模块的具体实现。

4.1　浏览器引擎介绍

图 4.1 表示了目前的 Everest Engine 的主要功能模块以及各个模块之间的交互关系，图中每一个方框均代表一个功能模块。

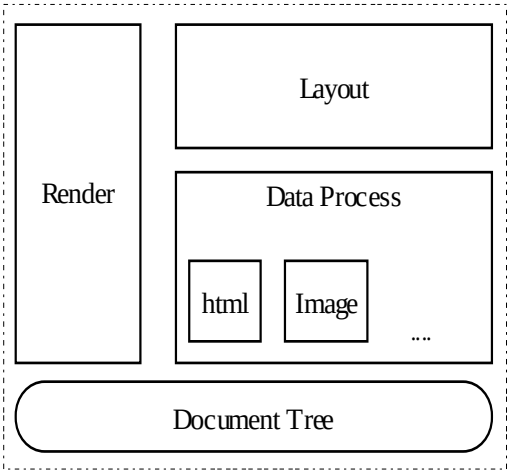

图 4.1　Everest 工作流图

图 4.2 对 Browser Engine 中的数据流进行说明：

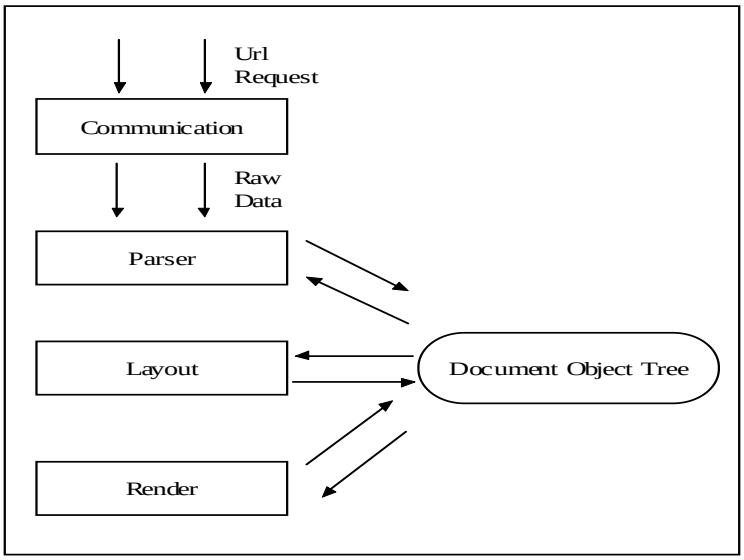

图 4.2　Browser Engine 中的数据流图

其中数据流是从 url 提交到通讯模块开始，通讯模块取回数据后交给 Parser(数据处理模块)进行解析，生成文档对象树，然后 Layout(布局模块)会根据文档对象树上的节点进行布局整理，把布局信息存储到文档对象树中，最后 Render(绘图模块)会对客户端进行绘制。

4.2　文档对象树模块

该文档对象树模型是参照 DOM 中结构的思想，通过把 HTML 文档元素抽象为对象，使得逻辑更加清晰，也更容易维护。树上的每个节点对应一个浏览器页面上的 Widget（我们认为页面上的布局单位为 Widget），比如 Page, Button, Table 节点等。我们之所以可以这样抽象，依赖于 HTML 页面实际上是层次结构的，同时我们可以把层次的嵌套关系对应为树节点的父子关系，从而生成文档对象树。这个树是在对数据进行解析时生成的，由数据处理模块负责生成，而布局和绘图的操作都将是通过操作这个树来完成的。

这里我们把 HTML 页面元素进行了抽象，分为以下几个类，见图 4.3

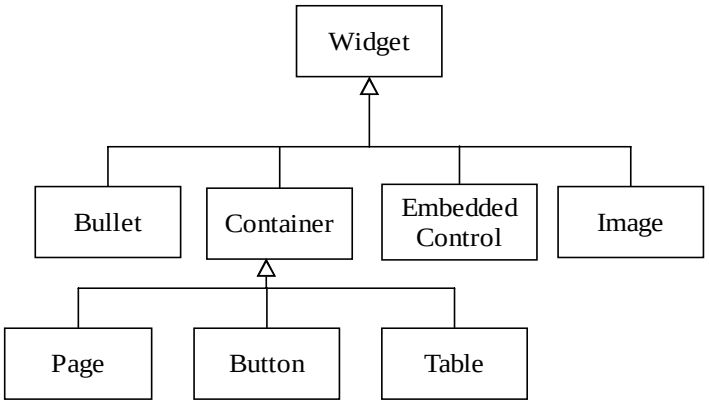

图 4.3　类关系图

从图 4.3 中可以看出，所有的类都是从 Widget 类继承出来的。这些类满足了目前的需要，可以很好地对页面进行显示。下面分别介绍相应的类。

4.2.1　Widget

Widget 类是所有类的基类，它抽象出了作为一个页面对象基本特征，其中包括布局和显示所需要的信息。其接口指针函数部分包含了布局和显示函数接口以及部分鼠标类消息响应函数接口。其中鼠标消息包含鼠标移动消息和鼠标进入 Widget 的消息和离开 Widget 的消息，鼠标点下和释放的消息。在实例进行初始化的时候会把这些接口函数指针关联到响应的消息上并对接口函数指针进行初始化，指向对应的消息处理函数。文档对象树上的节点均为 Widget 类型，但是对于特定的类型，在初始化的时候可以修改接口指针所指向的函数，也就意味着可以通过把接口指针指向子类提供的函数，从而保证在该类对应节点实例接收到相应消息的时候，是由子类自己提供的消息处理函数来处理的，体现了多态的特性。

4.2.2　Embedded Control

用来提供对其他非 Widget 类控件和 Widget 类之间的转接。由于在文档对象树中，每个节点都是 Widgt 类型，这里 Widget 类型和自定义的消息系统很好的结合，是一个完整的系统，而且对于布局操作来说，是一个整体考虑的行为，需用对整个对象树进行操作，需要节点作为文档对象树的节点接收消息进行处理。为了充分利用图形系统本身的控件，以及其他非 Widget 类型的控件，比如 CAR 构件等，或者作为一种 Widget 扩展机制，提供了 Embedded Contorl 控件类，该类本身是 Widget 类，可以作为文档对象树的一个节点来接收和响应消息等，而控件本身的任务是把这些消息转发给其它控件，并负责取回其它控件的反馈并返回给文档对象树。该类作为 Widget 类和其他控件直接的桥梁，有效的把其它控件融入到文档对象树中，同时也提供了好的扩展机制。对 CAR 的支持将依赖该类的实现。

目前 Embedded Control 主要是对<form>提供支持。当解析到<input>的时候，需要创建本地图形控件，并在上面包装一层 Embedded Control，然后把该节点接入文档对象树中。

4.2.3　Image

提供对图像的支持。主要包含图像对应的 url 地址信息，以及图片的大小信息。其中如果在页面中制定了图片的大小，则布局和显示均按照指定大小进行，否则根据图片的实际大小进行。还包含两个指针，分别指向解码后的图像缓存和伸缩后的解码的图像缓存，这两个缓存将在显示时利用。

Image 类修改了 Wdget 类的布局和显示的接口指针，并且在接口函数部分提供了链接部分的消息函数指针。该链接部分消息函数指针是为了支持<map>来预留的。

4.2.4　Container

该类只是提供行为的抽象，不包含具体的数据，扩展了几个接口函数指针，用来进行添加，删除，遍历容器内子元素的操作。因此继承自该类的类将具有增加、删除和遍历子控件的功能。

4.2.5　Table

该数据结构提供对<table>的支持。Table 中内容是一个最大行数乘最大列数的数组。其中每个元素是 DW_TABLE_CELL 或 DW_TABLE_SPAN_SPACE，当为第一种的时候，其中的子元素就是一个 Page，而属于该单元格的东西将都添加到对应的 Page 实例当中，包括新的 Table 实例。通过这种嵌套可以实现任意复杂的页面。

4.2.6 Page

Page 类使用来存放页面文档内容的，本身是容器，可以存放其他 Widiget。其数据结构中的主要部分如下：

```
struct _Page{…
    Container container;
    gint32 avail_width, avail_ascent, avail_descent;
    PageLine *lines;
    int num_lines;
    int num_lines_max; /* number allocated */
    PageWord *words;
    gint num_words;
    gint num_words_max; /* number allocated */
...};
```

其中主要数据项为 words 数据项，是一个动态数组，页面元素的内容存放在该数组里，对于文本对应的类型是 DW_CONTENT_TEXT, 对于 Widget 对应的类型是 DW_CONTENT_WIDGET。lines 数据项包含行的信息，主要为了布局方便以及通过保存中间结果，有效的优化显示速度。

Page 类是一个使用最普及的容器类，用来显示文本，同时可以包含其它 Widget。在这里，我们把需要同时显示文本和 widget 的页面单位都抽象为 Page，比如 Table 中的单元格就是 Page，对于自定义的按钮，其子元素也是 Page，而因为是 Page，所以既可以在 Page 上添加文本内容，也可以添加其他的 Widget，同过这种嵌套关系，一方面简化了管理的逻辑，另一方面可以解析复杂的页面，满足 HTML 的要求。网页本身是一个页面，因此在文档队象树的根节点就是一个 Page 对象（当直接访问图片的时候例外，此时的根节点是一个 Image 对象）。这里 Words 中存放的是最小的布局单元，也就是一个汉字或者一个英文单词，或者一个 Widget。其中可以存放的类型见下表：

```
typedef enum{
    DW_CONTENT_START    = 1 << 0,
    DW_CONTENT_END      = 1 << 1,
    DW_CONTENT_TEXT     = 1 << 2,
    DW_CONTENT_WIDGET = 1 << 3,
    DW_CONTENT_ANCHOR = 1 << 4,
    DW_CONTENT_BREAK    = 1 << 5,
    DW_CONTENT_ALL      = 0xff
} DwContentType;
```

例如对于如下的页面：

```
<HTML>
<body>
Hello World
<table>
<tr><td>Hi</td><td>您好</td></tr>
</table>
</body>
</HTML>
```

解析后生成的文档对象树将如下图所示：

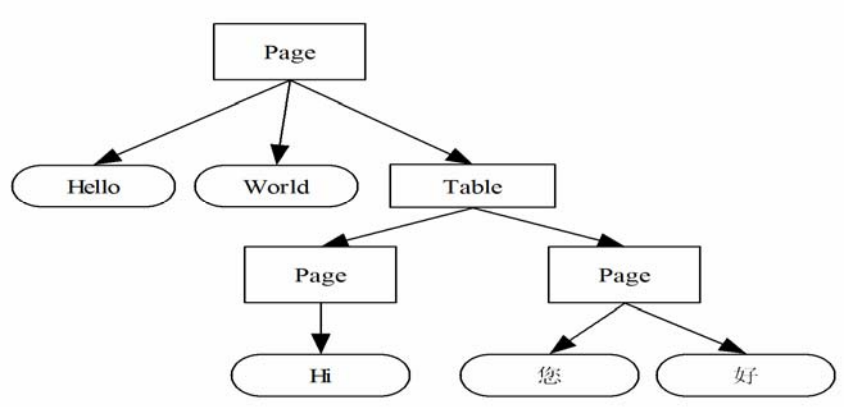

图 4.4　文档分析结果树图

根节点为 Page 节点，该节点包含三个 word，前面两个为文本，分别是“Hello”和“World”，第三个为 Widget，这里具体是 Table，而 Table 有两个单元格，因此有两个 Page 子 Widget，

第一个 Page 中存放文本“Hi”，而第二个 Page 有两个 word，分别为“您”，“好”。

其中 Page 类接口还扩展了处理超链接的函数指针，并注册到相应的超链接函数的消息上，其中包括鼠标进入链接，离开链接，点击和释放链接的函数。同时还包含一个 Page 自己的布局接口函数，该函数是用来根据页面的宽度来调整每行元素个数的。其定义参见如下说明：

```
struct _PageClass
{
    ContainerClass parent_class;
    gboolean    (*link_entered)     (…);
    gboolean    (*link_pressed)     (…);
    gboolean    (*link_released)    (…);
    gboolean    (*link_clicked)     (…);
    void         (*word_wrap)      (…);
};
```

4.2.7 Button

自定义的按钮类，本身是容器。在Everest中，按钮有两类，一类是底层图形系统提供的控件，该控件的外形是由图形系统来控制的[29]，另外一类就是由浏览器提供的解析<button>生成的自定义的按钮，该按钮的外观和点击的行为完全由浏览器自己控制，因此可以达到比较好的效果。由于是容器类，而且Button的孩子节点是Page，因此可以在Button摆放任意的页面内容。同时这也是HTML4.01 的标准要求的。其中对Widget扩展了点击接口指针函数。用来提供对鼠标点击事件的响应。

4.3 数据处理模块

负责对通讯模块取回的数据进行解析，主要包括 HTML，Jpeg，Gif（以后将扩展支持对 CAR 数据进行解析处理），解析过程中将生成文档对象树。

当 Cache 取回数据后，会根据回接口函数调用相应的数据处理模块的函数，数据处理模块将对取回的数据进行分析。目前可以处理的数据有 HTML，Jpeg，Gif。关于具体处理我们在下面进行介绍。对于每一个部分的处理函数，都将从其注册的处理函数及抽象出的入口开始介绍，并简单介绍具体的分析过程。

4.3.1 HTML 文档的分析处理

HTML 的注册处理函数是 a_html_text，其入口函数为 a_html _callback。

```
DwWidget *a_ html _text (const char *Type, void *P, CA_Callback_t *Call,
                         void **Data)
```

参数：

Type　　类型信息，目前并没有使用。

P　　Web 数据结构，是发出请求时已打包好的数据。

Call　　请求的回调函数。

Data　　传递给 *Call* 的参数。

返回值：文档对象树的根节点。

说明：　这里将进行一些初始化工作，首先会分配 HTML 的数据结构，会初始化好状态栈等资源，同时将创建文档对象树的根节点 Page 并进行初始化，返回。*Call* 将返回 a_ html _callback，而 *Data* 将赋值为指向 HTML 类型数据结构的指针。

```
void a_ html _callback(int Op, CacheClient_t *Client)
```

参数：

Op　　操作状态，表明数据是否结束。

Client　　用来传输数据，主要包含 HTML 数据结构，还有缓存及尺寸。

返回值：无

说明：　该函数是 HTML 解析模块的入口，其中 *Client* 中存储了指向缓存区的指针及相应的缓存的大小，并且存储有前面初始化好的 HTML 数据结构，可以根据这些数据进行分析。

经过分析，HTML 文档是嵌套式的语法，由于 HTML 语言本身是标记语言，其中的标签（tag）是来表示格式的，在分析的过程中，应该转化为风格信息进行保存。因此我们采用了栈的办法进行语法解析，这样当遇到新的标签，进行进栈操作，这样栈顶的状态代表了当前的标签指示的状态，当遇到标签结束符的时候，进行出栈操作，恢复到上一层的状态指示，满足了嵌套式的要求。而具体的文档内容的风格，只需要根据当前栈顶的状态进行修改就可以了。

通过对页面进行分析，我们认为页面中的数据分为三类，一类是标签（tag），是 HTML 语法的关键字；一类是空格和特殊字符比如"\r\n”等，由于在 HTML 中多余的空格是会被处理掉的，而且页面的显示完全按照标签说明，其中的特殊字符没有语义上的作用，等同于空格；一类就是需要显示的文档内容。对于这三类数据，我们分别处理，而对于这三类中基本的分析单位，我们统一称为符号。关于如何具体处理，不进行详细说明。下面介绍一下设计中进行优化的几个地方：

1. 对标签的定位，为了取得比较快的速度，采用了表格的方法，表格是有序的，在定位的时候，采用二分法，提高了速度。其中标签表如下所示：

```
static const TagInfo Tags[] = {
      {"a", 'R', html_tag_open_a, html_tag_close_a},
      {"abbr", 'R', html_tag_open_abbr, html_tag_close_nop},
      ...
      {"var", 'R', html_tag_open_var, html_tag_close_default}
};
```

2. 对汉字的支持，这里对汉字进行了拆分，由于英文单词是通过空格来区分，比较好区分一个单词，但是对于汉字，只有通过内码来进行判断，目前还是一个概率性的判断，基本满足了要求。其中的关键判断如下：

```
#define ISGBHANZI(z) (((unsigned char)(z)) >= 0xA1 && \
                      ((unsigned char)(z)) <= 0xFE)
```

3. 对数据分段解析的研究。考虑到有一些页面数据量比较大，需要分批次传输，因此对解析器进行了数据分段的研究。这样每次取回部分数据后，可以先交给处理器进行处理，当下一部分数据到达的时候，继续解析，效果同数据一起到达没有区别。通过添加了这样的支持，可以很好的进行并行处理，网络模块取得部分数据后，先提交给解析模块，然后继续取数据，而数据解析模块则进行 HTML 的语法解析，当新的数据到达的时候，继续进行，这样有效的提高了效率。这样做需要解决的问题是保存当前分析的状态以及保证数据的连续性。这里我们利用栈来保存状态，栈顶数据保存了当时分析的状态。对于数据的连续性，我们采取了每次分析完一个符号后，记录缓存区指针位置的办法，该指针将指向下一个符号的开始，直到该符号分析结束。这样如果数据在符号中间断开，解析将会中止，而指针仍指向符号开始的位置，当新的数据到达的时候，从符号开始的地方继续进行解析。

4.3.2 图像的分析处理

我们支持图像的格式有 BMP，JPEG/JPG，GIF 等，这里我们以 GIF 为例进行介绍，其它类似。注册处理函数是 a_Image_Gif，其入口函数为 a_Image_Gif_Callback。

```
DwWidget *a_image_gif (const char *Type, void *Ptr, CA_Callback_t *Call,
                       void **Data)
```

参数：

Type 类型信息，目前并没有使用。

Ptr Web 数据结构，是发出请求时已打包好的数据。

Call 请求的回调函数，需要填充。

Data 传递给 *Call* 的参数。

返回值： Image 类节点。

说明： 对 Call 赋值为 image_gif_callback，Data 赋值为中间数据结构 ScopeImage。

```
void image_gif_callback (int Op, CacheClient_t *Client)
```

参数：

Op　　操作状态，表明是否还有数据。
Client　　包含 ScopeImage 数据结构，还有缓存及尺寸
返回值：　无
说明：　该函数是 Image 解析模块的入口，其中 *Client* 中存储了指向缓存区的指针及相应的缓存的大小供解码用。
图像的处理有一点不同于 HTML 文档的解析，因为图像大部分是作为页面的一部分来请求的，因此在发出请求的时候，已经生成了文档树上对应的 Image 节点，因此在发出请求时，需要保留指向树上 Image 节点的指针。这里图片的解码还不能支持断点续分析，只有当数据全部到达之后，才可以进行解码。

4.4　布局模块

该模块主要负责处理布局类消息，从而对生成的文档对象树进行布局.布局完成后的文档树包含了显示用的所有信息。布局模块会根据数据处理阶段生成的一些初始的布局信息，比如文字的大小等，利用特定的算法进行布局。

布局的过程主要是一个收集布局请求信息，进行布局然后进行布局设置的过程。其中请求和布局的设置分别对应SIZE_REQUEST的处理过程和SIZE_ALLOCATE消息的处理过程。在分析完树之后，首先对根节点进行操作。先设置根节点的可用的宽度高度，也就是视窗的大小，并不直接操作，而是通过向根节点发送 SET_WIDTH，SET_ASCENT，SET_DESCENT 等消息来完成的，这样节点可以根据自己的情况提供不同的处理函数进行处理。然后发送 SIZE_REQUEST 消息给根节点，去请求根节点的尺寸信息，根节点将根据自己孩子的情况，把该消息给文档树上的其他节点对象，从而遍历文档对象树。当该消息处理完毕，实际对应整个文档树对消息处理完毕。最后取得的根节点的尺寸信息实际上就是页面显示需要的尺寸信息。然后以视窗的尺寸和收集的根节点的尺寸信息为参数，发送 SIZE_ALLOCATE 消息给根节点，根节点将根据这些信息计算具体的布局。同 SIZE_REQUEST 处理，根节点将把消息发送给孩子节点，使该消息遍历一遍文档对象树。

对于文档对象树上的 Widget 节点，都需要响应 SIZE_REQUEST 和 SIZE_ALLOCATE 消息，其过程同和根节点类似。这里根节点是 Page 节点，并且具有一定代表性，其孩子节点既可以是文字，也可以是 widget，所以我们这里用 Page 的布局过程来进行说明，其他类似。首先介绍 Page 节点的 SIZE_REQUEST 的响应过程。

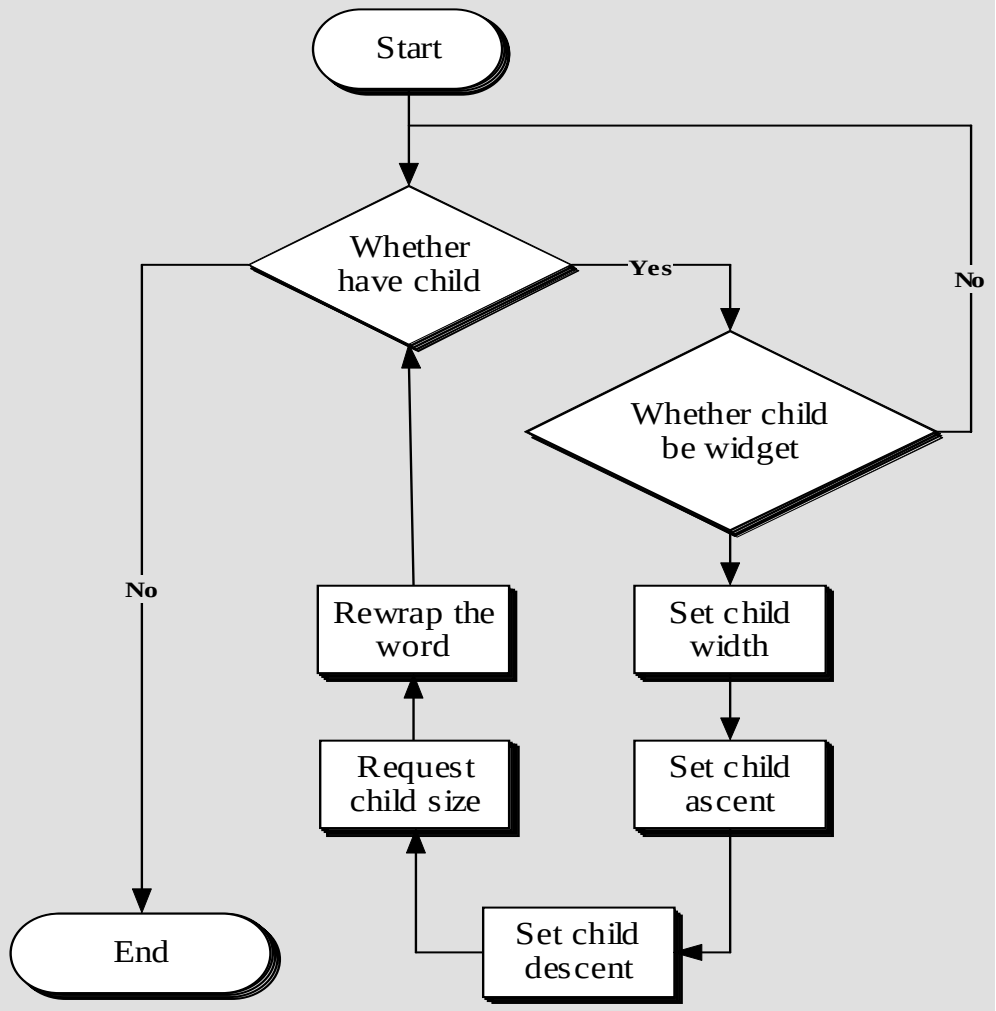

图 4.5　Page 对象布局流程图

由于在生成文档对象树的时候，Page 中需要生成关于行的信息，而在一行中，其中的文字的布局信息是已知，而且是常量，只有子 Widget 的布局信息时未知的，需要计算。因此在上图，我们没有看到计算文字信息的计算，实际上文字的布局信息已经在行的信息里包含了，而行的信息会在子 Widget 计算完后，对该行重新计算信息时更新，这是由 Rewrap 来完成的。对于子 Widget 来说，首先设置可以得到的宽度和高度，然后发 SIZE_REQUEST 消息给该子 Widget，而该子 Widget 会根据注册的消息处理函数进行处理，其过程类似于 Page 的处理。

而对于 SIZE_ALLOCATE 来说，每一个 Widget 有自己的实现，并且直接给自己的孩子节点发出 SIZE_ALLOCATE 消息既可。

布局是在分析完数据生成文档树之后进行的。在文档树分析的过程中，也要进行一些布局信息的收集，比如基本的布局单元，如 word 的尺寸等。而且页面增加元素后要进行重新排版并进行布局计算等等。

4.5　绘图模块

该模块主要来处理绘图消息，其机制类似于布局消息的处理。当布局完成之后，布局模块发消息给控制中心，控制中心将发出 DRAW 消息给文档对象树根节点，根节点处理消息的同时，会发消息给其它子节点，直到文档树处理完毕。

对于 Page 的绘制，是逐行绘制的，对于行中的 words，如果是文本直接绘制，如果是 Widget，这发消息给子 Widget。通过对目前的 Internet 内容的分析，可以发现网络内容以文本为主，因此这里做了优化：对于风格统一的文本内容，我们首先存入缓存，直到行结束或者有其他风格的文本出现，然后一次性输出，这样大大减少了对底层图形系统的调用，有效地提高了显示的效率。

第 5 章　Everest 的实现及扩展的研究

5.1　Everest 的实现

Everest 是 Elastos/Atlas 平台上的应用程序浏览器。用来提供在 X86/Elastos 上对 Internet 的访问，并且将来作为点击运行实现等功能的前端，并将作为 Elastos 上开发的浏览器的原型随 Elastos 迁移到手机、数字电视等嵌入式平台。目前支持 HTML4.01, HTTP1.0/HTTP1.1, Jpeg, Gif 等，对 CSS 部分支持。

图 5.1 表示 Everest 和底层操作系统的关系，主要依赖于 Atlas，Glib 的仿真层；并且还直接依赖于 Elastos，主要包括一些底层的系统调用，比如网络，文件操作等。其中 Glib 仿真层主要包括对 GHash，GString 等的仿真。

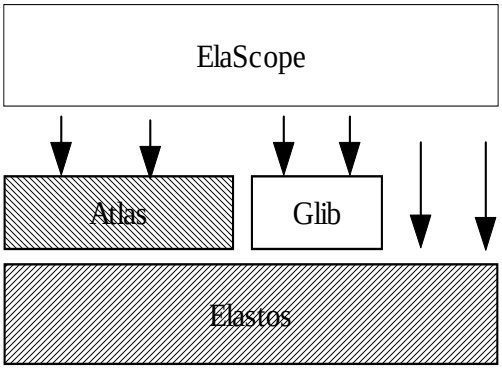

图 5.1　Everest 和 Elastos 的关系

在 Everest 中，利用体系结构很好的新协议支持的特性，很好地添加了对图像的支持，目前可以通过配置文件实现浏览器是否支持图形的显示。而且各个模块之间耦合性比较小，可以比较方便的对各个模块进行独立的修改。由于基于 C 实现，使得目前 DEBUG 版本仅为 300k。由于利用了回调机制，使得其解析和显示速度在 win2k 虚拟机上达到了甚至超过了通用浏览器。同时 Everest 具有很好的扩展机制，可以方便的添加对于 CAR 构件的支持。由于基于 Atlas 实现，浏览器可以运行于 WinCE/Windows 平台

5.2　Everest 的特点

1. 比较小的尺寸，及较好的运行速度。目前 Debug 版本的 Everest 代码共 300K，下表中是 Everest 在 Win2k 虚拟机上和 Elastos 单板机上对语法解析部分的实验数据，

主要是对 HTML parser 进行测试，由于网页是分段获得，取得的数据是对各个数据段解析后的累加。从数据可以看出虚拟机上的速度是单板机上速度的 5 到 10 倍，而且从数据本身来看，解析速度还是可以接受的，然而，由于我们采用了边下载，边解析的技术，下载和解析可以并行，因此网络 I/O 是浏览器速度的关键。

表5. 1　Everest访问网站数据解析速度表

网站	PC 机(ticks)	单板机(ticks)
www.sina.com.cn	903	3912
http://www.163.com	292	2746
www.sohu.com	554	4435
www.google.com.sg	31	151
www.nytimes.com	237	2109
www.sun.com	62	444
www.microsoft.com	47	567
www.w3.com	78	581
www.baidu.com	15	194
www-900.ibm.com/developerworks/cn	219	1983
www.msn.com	126	1435

注：PC机配置：　奔腾1.7G；256M RAM；win2k/Elastos

单板机配置：赛扬433；　32M RAM；　Elastos

2. 对 HTML4.01 的支持，包括图片及 FORM，TABLE 等，很好的容错性，比较好的布局显示。参见图 5.2, 图 5.3。

图 5.2　Everest 访问纽约时报主页图

图 5.3　Everest 访问百度搜索结果图

3. 边下载，边解析。通过支持分段数据解析来实现这个特色，前面我们已经介绍过了，通过这个特性，大大提高了浏览器的相应速度；

4. 通过配置文件控制图片的支持。考虑到一些嵌入式设备不需要浏览图片，这样我

们添加了控制文件选项，这样，当不需要显示图片的时候，只要关闭开关即可，这是会显示图片的提示文本在显示图片的地方；

5. 好的移植性。由于基于 Elastos 实现，而 Elastos 在主流的 windows 和 Linux 上提供了虚拟机，因此可以随着 Elastos 而在各种嵌入式设备上实现。并且由于本身是用 C 实现，也可以很好地移植到其他嵌入式平台，由于 Atlas 提供的 wince 的兼容性，Everest 可以运行在 wince 平台上。
6. 很好的扩展特性，方便的添加对新的标签，新的数据类型的支持。这是因为基于良好的体系架构导致的。

5.3　Everest 的 CAR 构件的扩展的研究

Everest 是在 CAR 构件平台上实现的，因此利用 CAR 构件具有天然的优势，通过在浏览器中支持 CAR 构件的加载运行，一方面可以很好的扩展浏览器的功能，同时是实现网络服务编程思想所必须的。这里我们首先简单介绍 CAR 在浏览器中实现的设想，然后结合 Everest 目前的一些技术实现说明对 CAR 支持的方案。

1.1.1 5.3.1　Everest 中 CAR 支持目标及问题分析

浏览器中对 CAR 进行支持，是把浏览器作为对象容器，而 CAR 构件作为对象在容器中运行。
未来 CAR 在文档中的描述类似下面的描述：

```
<EZXML  car=”http://www.elasots/com/cars/checkpass.dll”     width="132"  height="4"
EVENT….>
```

其中 car 属性指明了需要运行的 CAR 构件的位置，该属性是必须的，宽和高的属性可选，事件主要是浏览器和 CAR 构件之间交互的事件等。当页面加载时，根据页面内容加载不同的构件进行布局和现实以及交互等。
通过分析，需要解决的主要问题有：

1. 标签的扩展，在网页里支持 CAR 构件的声明；
2. CAR 构件的定位和获取；
3. CAR 的加载和运行；
4. CAR 运行安全机制的保证；
5. CAR 构件在页面中的布局；
6. CAR 构件同浏览器的交互。

1.1.2 5.3.2　Everest 中实现 CAR 支持的研究

我们认为上面提出的问题，需要依赖于浏览器和 Elastos 平台共同解决，下面给出关于以上问题的解决方案的初步设想：

1. 对于标签的扩展，我们可以在目前的 HTML 解析器里方便的进行添加，通过在 tags 数组中添加 EZXML 关键字，以及相应的函数对即可。
2. CAR 构件的定位和获取，可以通过网络模块来完成；
3. CAR 构件的加载和运行，由 Elastos 来进行，其中包括自滚动下载运行，因此只要通过网络下载指定的 CAR 构件即可。CAR 构件需要实现专门的入口接口，这样保证浏览器可以通过该接口进行 CAR 构件的初始化；
4. 安全机制的保证，可以在多个环节进行，通过安全认证以及 Elastos 提供的安全运行机制等；
5. CAR 构件的布局。CAR 构件本身是一个非 Widget 构件，而浏览器提供了 Embedded Control 机制，这样 CAR 构件可以包装一层 Embedded Control，从而可以实现 CAR 构件挂到文档对象树，参与布局和显示。这里要求 CAR 构件实现布局接口，这样可以获得 CAR 构件的尺寸信息及对 CAR 构件进行布局；
6. CAR 同浏览器的交互的实现，一方面可以通过文档对象树来进行，另一方面由于 CAR 包装了一层 Embedded Control，它可以接受文档对象树上的鼠标消息等。如果 CAR 作为字窗口，可以直接响应图形系统发出的消息，如同在<form>的支持中，一些图形系统的空间，直接同图形系统进行交互。

关于 CAR 构件的支持，可以参考<form>的实现，<form>中既利用了 Atlas 提供的控件，同时还需要响应点击消息，并发出提交请求。而关于 CAR 构件的运行和安全机制，以及 CAR 构件同浏览器之间的通讯等还需要进一步的研究。

第 6 章　结论与展望

6.1　结论

本文提出了一个适合于 Elastos 的嵌入式浏览器框架，该框架具有简单，模块化好，模块耦合性小，易扩展等特点。

该框架在包含基本功能模块的基础上，参考了 DOM 的机制，对页面数据进行对象化，抽象出文档对象树模块。以对象化的方式对页面管理，使得对页面的管理更加方便，逻辑性更好，同时有利于提供对 DOM 的支持。该框架提供了扩展机制，可以很好的支持 CAR 构件等第三方控件。

该框架中的面向对象机制由自定义的基于 C 语言的对象支持框架提供，该对象框架通过几个简单的接口可以很有效的提供对象的支持，很好的支持了继承，多态等特性。该框架同时依赖于自定义的消息机制，该消息机制提供了注册，关联，激活和销毁消息的机制。通过利用自定义的对象框架机制和消息机制，很好的解决了浏览器引擎模块之间耦合的问题

该框架抽象出网络模块和浏览器其他模块之间的接口，很好的提供了网络部分模块化的机制。同时通过一套注册和查询机制，可以使得网络模块取得数据后动态取得处理函数的句柄。这也提供了方便的对浏览器支持数据类型的扩展机制。

浏览器引擎是框架中重要的一部分。包含了文档对象树，数据处理，布局和绘图等模块。其支持下载和解析异步进行，而且可以数据分段解析等特性。

最后对浏览器对 CAR 构件的扩展进行了研究。并结合当前框架提出了对 CAR 构件支持的设想，为进一步研究做准备。

本人的工作主要有：Everest 框架的设计和实现；Everest 引擎的实现；完善对 HTML4.01 的支持，包括 FROM 的支持，图片的支持，及通过指导他人，完善对 TABLE 的支持；通过配置文件控制图片显示等。

本论文工作中，在面向网络服务编程的和欣操作系统上实现了浏览器。一方面为和欣操作系统实现“点击运行”，“网络就是计算机”等理念提供现实基础，供进行进一步的研究并进行商业化，以转化为生产力；同时对嵌入式浏览器技术研究有很好的参考价值；另一方面通过结合 CAR 构件进行扩展，为嵌入式领域实现网络服务编程提供了新的思路。

6.2　工作展望

浏览器的实现，仅仅是网络服务编程模型研究的开始，需要在以下方面继续进行深入研究：

1. 文档对象树的抽象。目前仅仅抽象出数据模型，相关的操作接口还没有抽象出来，目前是通过指针来直接操作文档对象树。而文档对象树是浏览器的数据核心，通过提供 API，为以后的添加新的协议的支持以及 CSS 更全面的支持，脚本语言的支持都是非常重要的；
2. 脚本语言的支持。通过支持脚本语言，可以更好的实现浏览器和客户端的交互，同时可以更好的控制页面元素；
3. 浏览器和 CAR 构件的消息互通，之间协议的抽象。可以通过脚本语言操作 CAR 构件，并且 CAR 构件可以发消息给浏览器，完善浏览器的扩展机制；
4. 对浏览器的优化。可以使浏览器变得更小，更快，更好地满足嵌入式平台的需要，以及商业化的需要；
5. 点击下载运行机制的研究，是网络服务编程模型的基础。

参考文献

[1] 陈榕, 刘艺平. 技术报告: 基于构件、中间件的因特网操作系统及跨操作系统的构件、中间件运行平台(863 课题技术鉴定文件), 2003.

[2] Elastos, Inc. Elastos 1.1 Information Repository. Available online at http://www.elastos.com/download.php.

[3] Sun Microsystems, Inc. Java 2 Platform, Enterprise Edition (J2EE) version 1.4 Specification. Available online at http://java.sun.com/j2ee/download.html.

[4] Microsoft .NET Home. Available online at http://www.microsoft.com/net/

[5] Ian Gorton, Dave Thurman, Judi Thomson. Next Generation Application Integration: Challenges and New Approaches. Proceedings of 27th Annual International Computer Software and Applications Conference. November 2003, pp.576—581.

[6] Firefox WebSite. http://www.mozilla.org/products/firefox/

[7] Rich Internet Application. http://www.riacn.com/web/showArticle.asp?id=161.

[8] 迎接RIA时代的来临. http://www.riacn.com/web/showArticle.asp?id=2

[9] 朱剑民,陈榕,倪光南. 和欣操作系统的浏览器设计模型. 计算机工程与应用 2003 年 13 期

[10] Dillo WebSite. http://www.dillo.org/

[11] 杨玉平. 嵌入式浏览器 DeltaBrowser 的设计与实现：[硕士学位论文]. 成都：电子科技大学计算机应用专业，2003

[12] 徐会建. 嵌入式浏览器 DeltaBrowser 表示层的设计与实现：[硕士学位论文]. 成都：电子科技大学软件与理论专业，2004

[13] 彭丽娟. 嵌入式浏览器的研究：[硕士学位论文]. 北京：北京工业大学计算机应用专业，2001

[14] 申波. 基于 XML DOM 的嵌入式浏览器研究及核心模块的设计和实现:[硕士学位论文]. 成都：电子科技大学计算机应用技术专业，2002

[15] 郭相国. 嵌入式浏览器研究:[硕士学位论文]. 成都:电子科技大学计算机应用专业，2003

[16] 王志利. 嵌入式浏览器的研究与实现：[硕士学位论文]. 成都：电子科技大学计算机应用专业，2003

[17] Dale Rogerson（美）著，杨秀章 译. COM 技术内幕(Inside COM)，清华大学出版社.

[18] Box, Don. Essential COM. Menlo Park, CA: Addison-Wesley Publishing Company, 1998. ISBN 0-201-63446-5.

[19] D.Raggett. HTML 4.01 Specification. W3C. Dec, 1999.

[20] 什么是DOM， http://51js.zahui.net/html/1/27301.htm .

[21] P.Hegaret, et al (ed.). “Document Object Model (DOM) Specification” Ver.1.0 W3C, Mar. 2003

[22] Embedding Gecko. http://www.mozilla.org/projects/embedding/

[23] 周正勇，阳富民，胡贯荣. 一种嵌入式浏览器的核心技术及特色. 计算机工程与设计. 24(3): 21-23. 2003.

[24] 王光磊，张跃，杜伟力. 基于虚拟客户/服务器的清华嵌入式浏览器 THEWEB 的设计与实现. 计算机应用与研究. 1(4):96-99. 2002

[25] Yoshinori Saida, Hiroshi Chishima, Satoshi Hieda, Naoki Sato, Yukikazu Nakamoto. An Extensible Browser Architecture for Mobile Terminals[C]. Proceedings of the 24th International Conference on Distributed Computing Systems Workshops. 2004 IEEE.

[26] Deepak Mulchandani. Java for Embedded Systems. IEEE Internet Computing, pp. 30-39, MAY 1998.

[27] Hal Berghel. The Client side of World-Wide Web. Communications of the ACM, vol.39, no.1, pp. 33-40, Juanuary 1996

[28] Stanley B. Lippman, Josée Lajoie. C++ Primer(3rd Edition), ISBN 0-201-82470-1, March 26, 1998.

[29] Charles Petzold(著),余孟学(译). Windows 程式设计开发指南. 北京大学出版社. 2001

致谢

在论文完成之际，我要衷心感谢所有关心和帮助过我的老师和同学们。
首先，我要感谢我的导师顾伟楠教授。
在此，我向他们表示最衷心的感谢。

个人简历　在读期间发表的学术论文与研究成果

个人简历：

已发表论文：

[1]