摘 要

遗留软件大都是面向对象技术在软件开发中广泛使用之前编写的，它们已经深深的融入企业的业务运行之中。为了节约资源，充分利用已有的成果，遗留代码重用不可避免。

本文提出了 LC-XML 方法，该方法使用 XML 技术包装遗留代码，将包装后的遗留代码 fhfml 代码作为网格信息，供网格用户访问，从而实现遗留代码信息共享。通过该方法，用户不用直接分析冗长难懂的遗留代码，而根据 fhfml 代码就可以方便地获取源代码信息。

很多情况下，企业不仅关注遗留代码本身，更希望重用遗留代码的功能，实现服务共享。但是遗留软件大多只适用于集中式系统平台，现在的分布式系统平台中无法直接实现其功能重用。根据这一需求，本文提出了遗留代码服务化 LC-GS 方法。该方法将遗留代码转化成用户可以访问的服务，将其功能提供给用户，从而实现遗留代码功能重用。

我们提出的 LC-XML 和 LC-GS，以前期开发的基于 web 服务的企业信息网格 Loglo 系统为平台，文章在介绍了 loglo 系统之后，重点分析了通过这两种方法实现遗留代码重用的整体框架设计和关键技术，给出了两种方法在 Loglo 系统中的设计和开发步骤。初步测试表明，LC-XML 和 LC-GS 分别可实现遗留代码的重用。

关键词：遗留代码；重用；包装；服务；XML

Abstract

Legacy software is almost designed before object-oriented development was widely used in industry, and has been deeply incorporated in business operation. In order to save resources, fully utilize the existing achievement, it is inevitable to reuse legacy code.

LC-XML method is proposed in the thesis to realize the sharing of legacy code information, which wraps legacy code into fhfml code by using XML that make it accessible to the users. Then, the users don't need to analyze tedious legacy code, and they can get source code information only in the fhfml code.

Mostly, businesses concerned not only the legacy code itself but also the reusing function of legacy code as to realize the sharing service. Because most of the legacy code are only designed for centralized computing platform, and they can not be reused directly in current distributed computing platform. Considering the requirement above, we propose the method of LC-GS to convert legacy code into service which is reachable to the users. Thus, reusing the function of legacy code can be realized.

LC-XML and LC-GS are based on Loglo system which has been developed on Web Services. Having introduced the Loglo System, the thesis discussed the framework and key techniques in detail to explain how we implement the reusage of legacy code by the methods. And put forward the design and development step of the two methods in Loglo system. The test done by our research team has proved that LC-XML and LC-GS can implement the reusage of legacy code.

Keywords: legacy code; reusage; wrap; service;XML

独创性声明

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

签名：邹燕飞　　日期：2007.5.17

关于论文使用授权的说明

本人完全了解西北师范大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：邹燕飞　导师签名：　　日期：2007.5.20

第一章 绪 论

1.1 论文选题依据

目前在银行、证券和保险业中存在的大量商用程序是采用从诞生到现在已有近二十年的传统语言编写的[1]。由于现代编程语言以其优越性替代了传统编程语言，目前很少有软件从业人员去学习和使用传统语言，因此用这些传统语言编写的程序代码就成为了遗留代码，软件成了遗留软件。一般认为遗留软件是目前正在使用的并且需要更新的软件系统。近年来，这些遗留软件系统的规模越来越大，功能也越来越复杂，蕴含了大量的领域知识，从需求设计、商业逻辑到历史数据，都具有可以重复利用的价值[2]。

当然，新系统的开发可以完全扔掉遗留软件，在零的基础上重新进行开发，但无论在费用、开发周期、开发风险上，都会面临很大的挑战[3]。如果重用遗留软件资源，就会明显地节约系统更新的费用、缩短开发周期、降低开发风险。软件重用可以在不同的抽象级别实现，本文的研究集中于面向源代码级别的软件重用。通过重用遗留代码使遗留软件在信息计算平台和范型中得到使用。

本文的研究内容源于甘肃省科技攻关项目（2GS047-A52-002-04）《企业信息网格关键技术研究》。

1.2 国内外研究现状评述

遗留软件都是面向对象技术在软件开发中广泛使用之前编写的[3]，它们使用集中式体系结构，编写遗留软件的遗留代码讲究的是面向过程、逐步求精的编程理念。这些代码通常被组织为子程序或函数的集合。每个子程序提供系统功能的一部分，并且按照需要由其它子程序调用。面向过程的设计策略为自顶向下的设计[4-6]，将程序分解成交互的函数或子程序的集合并且由这些函数共享集中式的系统状态如图 1.1。这种设计策略隐藏了函数的实现细节，使之对于其它函数来说是透明的，但其内部算法的改变（如共享变量值的改变），却容易导致数据不一致。

当今，随着分布式的广泛应用，遗留软件系统也逐渐由采用集中式体系结构进化为采用分布式的体系结构，其原因有[7]：

① 硬件花费，购买分布式客户/服务器系统以及后期维护的费用通常比购买同等能力的

大型机的费用要小。

② 用户接口的需求，遗留软件系统通常采用字符、文本的界面。新的趋势要求系统具有用户容易理解的、美观和便于交互的图形用户界面。这种界面需要更多的本地计算，只有在客户/服务器系统中才能提供这种功能。

③ 对系统分布式访问的需求，由于企业在发展的过程中逐渐将组织分布在各个地方，客户和企业员工希望可以访问位于公司不同地方的软件系统。

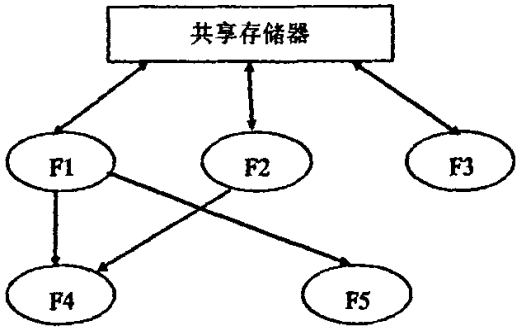

图 1.1 面向过程的设计

迁移遗留系统到分布式体系结构中，并开发有效的接口以减少硬件花费，从而以更为现代的"感观"方式来支持分布式工作。但是这些集中式的软件系统经过多年的维护，其用户接口、服务、数据库之间已经错综复杂，造成了遗留系统体系结构进化过程中最实质性的困难。针对这种情况，对遗留系统的解决方案也各不相同，在新的体系结构中重用遗留代码就是其中一种解决方案。

1996 年 Sneed 提出一种使用中间件技术整合遗留系统到异构分布式计算机环境的方法[8]，此方法主要是封装遗留系统使之在客户/服务器平台使用。Sneed 使用 CORBA 作为中间件封装汇编语言编写的代码和 COBOL 语言编写的代码验证此方法的可行性。

1997 年 A.De Lucia 等人[9]提出一种迁移遗留系统到面向对象平台的方法，包括 6 个连续的步骤（遗留代码的静态分析、分解批处理程序、面向对象模型的抽象化、根据分解和抽象重构系统、标识对象并将其封装、使用面向对象的语言转化前几步标识的对象）。A.De Lucia 的这些步骤中包含了逆向工程[10,11]和再造工程[12]，其中逆向工程的目标是将程序分解为用户接口组件和实现应用域对象组件，再造工程的核心是包装技术。同年，Burd[13]等人提出了一种通过重建遗留代码的方法来标识可重用单元，但是作者提出的 10 步标识可重用的单元的方法只能使用于 COBOL 语言。

1999 年 Canfora[14]提出了一种标识用户接口组件语句的程序切片算法，但是此算法只适用于非递归程序，且是由 RPG、COBOL 和 FORTRAN 所编写的程序。

2001 年，Chia[15]提出把遗留系统从大型机迁移到以用户为中心的分布式对象计算环境中。

此方法直接在大型机上包装遗留系统并且暴露遗留系统的接口给远程客户。作者实现了使用COBRA技术整合遗留系统到分布式计算环境，其主要通过ORB-ORB之间的访问完成。但是，中间件将成为整个系统的瓶颈。同年Y.Bi等人[16]提出以XML的方法重用遗留代码，主要集中在用户接口的XML化处理上，通过集中研究最本质的，必不可少的用户交互以及基于字符接口和图形用户界面字段之间的关系，整合这些关系到新的图形接口，从而实现迁移遗留应用到瘦客户/服务器平台的目的。文章中作者提出了以下几个关键的步骤，理解遗留系统功能、通过跟踪基于字符接口的交互区分信息和处理逻辑、由前面步骤提供的信息来决定目标系统、使用XML作为客户端产生新的图形接口、通过JNI技术包装原始系统的功能、迁移遗留应用到目标系统。同年，Emmerich[17]定义了基于XML DTD增量代码移动的方法，此方法能够增加、减少、替换代码的某些部分。作者在增量代码移动的实现中考虑了最简单的程序设计语言Karel，为我们解决遗留代码的重用提供了一种新的思路。

2003年Bodhuin[18]针对的是那些不可分割的遗留系统提出了迁移遗留系统到Web平台的工具。但是此工具只适合COBOL语言开发的软件系统，而且对于不同版本COBOL需要开发不同的解析器。Yan Huang[19]提出了一种由C语言编写的科学计算程序到基于Triana的计算服务（面向服务的的分布式体系结构，类似于网格计算）的半自动化转化方法。

2005年Delatittre[20]提出的GEMLCA(Grid Execution Management for Legacy Code Architecture)是一种通用的体系结构，可以部署遗留应用作为网格服务，而不需要重建代码。GEMLCA由四个基本组件构成，被设计为三层，第一层作为前端层，提供网格服务的功能；第二层为核心层，负责管理每个遗留代码进程和遗留代码作业；最后一层为后端层，被连接到GEMLCA体系结构所部署的网格中间件，此层可以被看作是一个插入层。但是这种体系结构只适合于GT3和GT4，而且对于它们GEMLCA的后端层都要发生相应的改变。

国内研究企业信息网格的中科院计算所提出的织女星企业信息网格[21-23]的主要特色是Vega，即通用服务、辅助智能、全局一体、自主控制。西安交通大学的基于Internet的信息网格的软件框架研究[24]参照计算网格的开放式网格服务构架，实现了信息搜索、登记、发现、预定等功能。但是它们都没有解决在企业信息网格中如何使用遗留系统的应用功能，动态部署新、旧服务，达到真正意义上的服务共享的问题[25]。

从遗留代码研究现状来看，这些解决遗留问题的方法，有各自的优缺点，2000年以前主要集中在包装迁移遗留系统到基于Web的分布式平台，随着分布式的变化，出现了网格，现有的主要技术成果是GEMLCA，但它有如下缺点，①只适用于GT3或者GT4作为中间件开发的网格。②将整个可执行的遗留代码作为网格服务，因而造成较大的花费。③需要一个计算服务器，从而增加了系统的冗余度。④这种方法在遗留系统迁移期间，必须停止遗留系统的使用。这些缺点，对企业来说是很难接受的。因此，我们从代码重用的角度，使遗留代码

成为共享信息资源，同时将其功能提供给网格用户。

1.3 主要工作

本文在我们现已构建的基于 Web 服务的企业信息网格 Loglo 系统之上，结合遗留问题已有解决方案，提出了一个新地遗留代码重用方案。文中所研究的遗留代码是指源代码。我们研究的主要目标是：

（1）将遗留代码包装成信息提供给网格用户，实现信息共享。通过 XML 语言，使用包装技术将遗留代码转化为 XML 中间表示，并以一种友好的方式提供给网格用户。这种转化的优点是，利用了 XML 独立于平台、半结构化等优点，当遗留代码作为信息在其他平台重用时，只要直接利用 XML 中间表示，而不用再次分析冗长难懂的遗留代码，同时客户端可以使用友好的 GUI 界面来显示包装后的遗留代码。

（2）将遗留代码转化成企业网格用户可以访问的网格服务，将其功能提供给网格用户，实现服务共享。

由于大部分遗留系统由面向过程的 C 语言编写，因此本文主要研究 C 语言编写的遗留代码。如图 1.1 所示对遗留代码重用的策略。LC（legacy code）表示遗留代码。

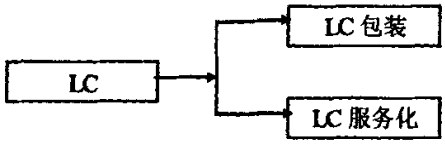

图 1.2 遗留代码重用策略

1.4 论文结构

本文各章节内容安排如下：

第一章作为整个论文的绪论，介绍相关背景知识和此项研究的必要性、可行性和现状，并简要的介绍论文的结构。

第二章介绍了遗留代码包装和服务化的基础，即基于 Web 服务的企业信息网格 Loglo 系统。主要对 Loglo 系统的总体设计以及信息的包装规范 reml 和 ttpml 进行了简单的介绍，最后以一个典型的应用实例结束本章。

第三章介绍遗留代码的包装，提出了一种遗留代码的包装方法 LC-XML。具体讨论使用 LC-XML 方法包装遗留代码的整体框架及关键技术，其中关键技术包括，包装遗留代码的规范、步骤以及相关流程，最后用一个简单的实例说明了 Loglo 系统中包装后的遗留代码 fhfml

发布一发现的过程。

第四章介绍遗留代码服务化，提出了一种遗留代码服务化的方法 LC-GS。具体讨论使用 LC-GS 方法将遗留代码服务化的整体框架以及关键技术，其中关键技术包括，遗留代码服务化的原理和步骤，最后通过一个企业中常见的人民币金额大小写转化的实例来说明使用 LC-GS 方法将遗留代码服务化的过程。

第五章对整个论文研究和实践工作进行总结，介绍了遗留代码重用所取得的成果，并指出我们现有工作的局限性和有待提高与改进的方面。

第二章 企业信息网格 Loglo 系统设计

2.1 框架结构

网格系统有哪些组成部分，组成部分之间的关系以及如何协同工作是网格结构研究需要解决的关键问题。我们提出的企业信息网格 Loglo 系统结构如图 2.1 所示。Loglo 系统是基于 Web 服务范型基础，由全局注册中心、提供者、消费者构成。全局注册中心结点实现了信息全局注册、信息查询和信息的动态发现，消费者在全局注册中心发现提供者发布的信息，最后通过访问提供者结点的信息服务获得信息。

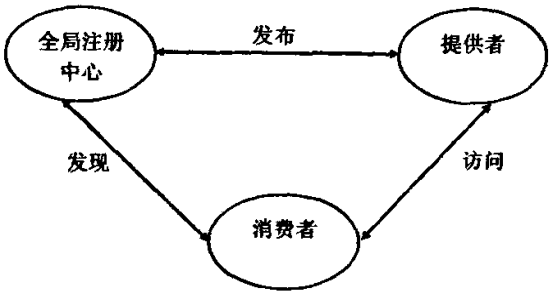

图 2.1 Loglo 系统结构图

2.1.1 提供者结点

提供者结点结构如图 2.2 所示。其中共享信息模块提供共享信息服务，本地注册模块完成信息的本地注册，全局发布模块实现信息的发布和注销。在 Loglo 系统中，根据 Web 服务的全局注册中心，我们提出了本地注册的概念，实现信息的一致性维护和管理的灵活性，其中本地注册功能模块是我们研究的重点。

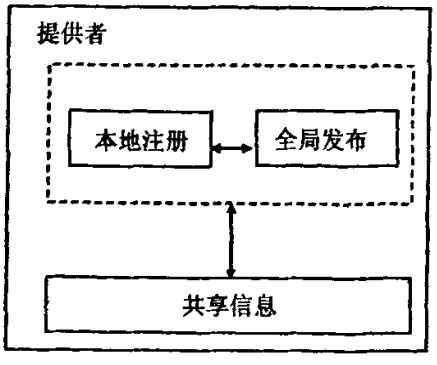

图 2.2 提供者结点结构图

本地注册模块采用存储系统和文件实例方式来抽象底层的资源。把这些底层可读可写的物理单元抽象成一个更高层次的本地注册表，目的为给上层提供一致的、透明的访问，即上层操作的只是一些注册表，不需要知道物理资源的存储方式和存储位置。其中本地注册中心由资源模块、JDBC 接口模块、基于 XML 元信息管理模块、自动访问模块、信息发布员管理模块构成。如图 2.3 所示。

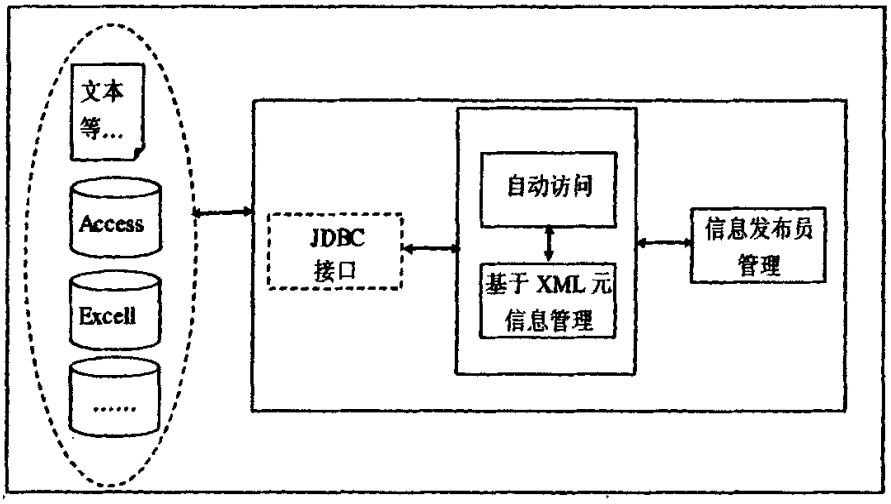

图 2.3 本地注册模块结构

图中椭圆虚线表示为提供信息的物理资源实体，他们构成企业信息网格的信息资源基础，该层主要包括各种数据库资源、文本资源、传感器采集的数据资源、媒体等资源。

2.1.1.1 JDBC 接口模块

JDBC 是一种能通过 Java 语言访问结构化数据的应用程序接口（API），为 SQL 兼容的数据库提供了一个标准的接口集合。JDBC/ODBC 允许用户使用统一的接口通过 ODBC 去连接数据库，而不需要专门研究 ODBC 的技术和规范。对于 Windows 操作系统下，注册者本地原有的数据库通过 JDBC/ODBC 桥进行动态连接。其它类型操作系统通过 JDBC 驱动进行连接。如图 2.4 所示。JDBC 接口只针对数据库信息资源，对于其它类型的信息资源没有此模块。

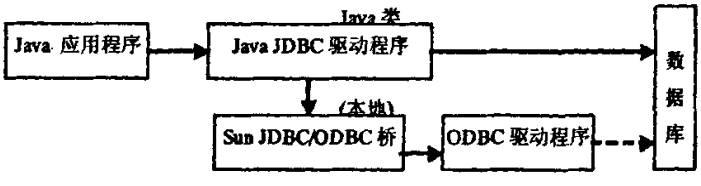

图 2.4 JDBC 接口

2.1.1.2 自动访问模块

自动访问模块其核心为本地注册表 reg.xml 和 database_reg.xml 以及对它们的一些操作。

信息发布员通过本地注册 API 对本地信息资源进行注册，应用程序根据注册者所注册资源的基本信息，如对于数据库资源，将注册资源的 JDBC 驱动、数据源、基本表名等控制信息自动提取写入本地注册表 reg.xml、database_reg.xml，信息服务开发者根据企业中经常用的几种类型数据库如，Excel、Access、MYSQL、MSSQL、ORACLE 开发统一的信息查询服务，并将这些服务封装一次性部署，当有新的数据库资源注册时，避免了手工部署的麻烦。当信息访问请求到达时，应用程序通过读本地注册表 reg.xml、database_reg.xml 完成了应用程序和数据库的动态连接。

对于文本资源，将注册资源的物理位置等信息写入本地注册表，根据对文本的常用查询方法开发通用的服务，即使有新的文本信息资源注册，也无须开发新的服务。当信息访问请求到达时，应用程序通过读本地注册表完成了应用程序对文本信息资源的定位。

2.1.1.3 基于 XML 元信息管理模块

基于 XML 元信息管理模块，其核心为本地注册表 table_info.xml，以及对它的一些操作。完成基于 XML 的应用元数据表示和物理资源之间的映射，即把物理资源用基于 XML 应用元数据表示，描述所有共享数据（物理资源）的属性，其中数据库包括字段名、类型、主键等。一般的元数据包括，数据库名、构架、表名、列名、数据类型、长度、小数位数、精度，是否允许空值等[26]，按照此体系结构需要，应用元数据表中的字段可以定义为数据表定义字段、扩展字段。如图 2.5 所示为应用元数据重要组成部分表示。其中序号、类别、表名、列名、数据类型、主键构成元数据定义字段，参照键等构成扩展字段。

图 2.5 中描述了 Excel 数据库“规划部.xls”和 Access 数据库“规划部 A.mdb”的应用元数据图，其中“规划部.xls”包含两个基本表“Sheet1”和“Sheet2”，“规划部 A.mdb”包含一个基本表“仪器”。

序号 resource_id	类别（数据库名） database_name	表名 table_name	列名 col_info	数据类型 type	主键 pri-key	参照键 key(id,table_on)
1	规划部	Sheet1	财政收支	varchar	null	(null,null)
1	规划部	Sheet1	财政预算	varchar	null	(null,null)
1	规划部	Sheet2	统一编号	vatchar	null	(null,null)
1	规划部	Sheet2	设备名称	vatchar	nul	(null,null)
2	规划部 A	仪器	产品代号	varchar	null	(null,null)
2	规划部 A	仪器	状态	varchat	null	(null,null)

图 2.5 应用元数据图

根据应用元数据表构建本地注册表 table_info.xml，将表头作为元素，表头对应的值作为

元素值。在应用元数据表中，相同序号 reource_id 为同一个数据库下的表（对应一个信息资源目录）构造基于 XML 元信息管理表结构如图 2.6 所示。

其中对于文本文件的元数据可以定义为，文件名、文件标题、文件段落数。

```
<database_info resource_id="1">
  <table_info>
    <table_name id="1"> Sheet</table_name>
    <col_info>
      <col type="varchar">财政收支</clo_1>
      <col type="varchar">财政预算</clo_2>
    </col_info>
  </table_info>
  <table_info>
    <table_name id="2"> Sheet2</table_name>
    <col_info>
      <col_1 type="varchar">统一编号</clo_1>
      <col_2 type="varchar">设备名称</clo_2>
    </col_info>
  </table_info>
</database_info>
```

```
<database_info resource_id="2">
  <table_info>
    <table_name id="1">仪器</table_name>
    <col_info>
    <col type="varchar">产品代号</clo_1>
    <col type="varchar">状态</clo_2>
    </col_info>
  </table_info>
</database_info>
```

图 2.6 基于元数据的 XML 表示

通过上面的分析我们得到由物理资源到本地注册表 table_info.xml（见附录 3）的抽象过程如图 2.7 所示。

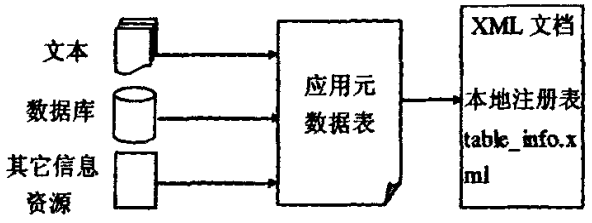

图 2.7 物理资源到基于 XML 元信息管理表抽象示意图

综上所述，基于 XML 元信息管理模块和自动访问模块，即可协助完成信息查询功能。对数据库信息资源的查询操作除了面向数据库的表不同（具体的说，就是数据库驱动和数据源不同），而导致其表名、字段名、类型等信息不同外，其他内容基本都是相同的的，在具体实现上，数据库连接时 Class.forName()和 DriverManager.getConnection()参数值不同而已，因此我们对数据库信息资源操作时通过本地注册表得到 JBDC 接口信息（Class.forName()和 DriverManager.getConnection()参数值）进行统一操作在理论上和实践上都是可行的。同时，又是用户访问数据库时动态完成数据库驱动和数据源连接较好的一个方案。对于文本信息资

源通过读本地注册表就可以定位到某一个具体的文件甚至可以定位到文件的某一段落。

提供者结点通过 JDBC 接口模块、基于 XML 元信息管理模块和自动访问模块屏蔽了资源层中分散、动态异构的各种资源，为企业信息资源的共享、集成和互操作提供透明的、一致的使用接口，以支持企业用户访问。

2.1.1.4 信息发布员管理模块

信息发布员管理模块核心为一个本地信息发布员注册表 uddireg.xml，以及对其的一些操作。在信息发布员发布信息之前，需要信息发布员管理模块向全局注册中心注册一个帐户，全局注册中心验证注册信息的用户合法性之后，开通合法用户并授权其发布信息，同时返回注册者一个发布号 id，并在本地信息发布员注册表写入信息发布员的相关信息。

2.1.2 全局注册中心

信息发布员通过本地注册 API 在本地对信息资源进行注册，注册成功后，信息提供者结点的全局发布模块通过调用全局注册 API 在全局注册中心进行信息资源的二次注册即信息的发布，发布成功后，消费者就可以在全局注册中心发现信息资源。全局注册中心由信息发布员注册模块、信息发布员授权模块、信息管理模块和信息查询模块构成。图 2.8 所示全局注册中心结构图。

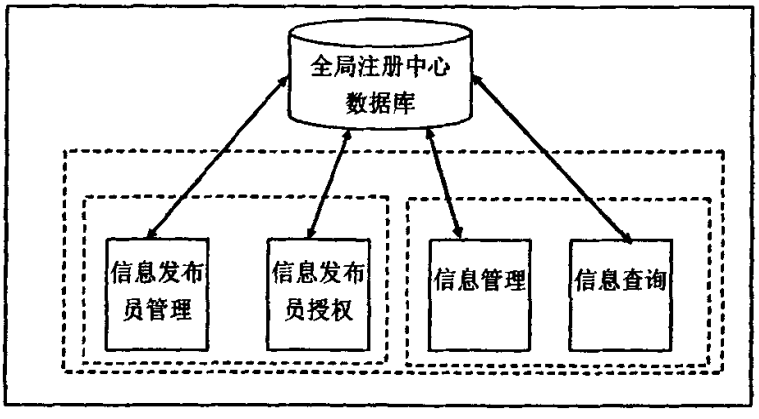

图 2.8 全局注册中心结构图

信息发布员管理模块由本地信息发布员注册管理模块调用，完成对信息发布员的基本信息在全局注册中心数据库中信息发布员表的注册或者注销。主要填写信息发布员的用户名、密码、发布者所属部门、发布者联系方式、发布者角色信息。

信息发布员授权模块主要是验证由信息发布员的合法性，通过信息发布员的合法证件，由全局注册中心管理员对其身份进行合法性验证。

信息管理模块提供一组 API 主要用于提供者结点的全局发布模块完成对其的调用，从而

实现对信息的发布或者注销。

信息查询模块主要给消费者提供了使用 Web 页面提交的关键字的信息发现功能。

2.2 信息的包装

在 Loglo 系统中，对数据库信息资源和文本信息资源都有不同的包装规范，分别为 reml（root-element Markup Language）和 ttpml(txt-title-par Markup Lamguage)。信息消费者得到包装后的信息，通过不同的规范对信息解析使用。

2.2.1 数据库信息 reml 包装规范

对数据库信息包装是按照数据库的格式而进行包装的。以下是数据库包装 reml 规范的 schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="root">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="element" />
            </xs:sequence>
        </xs:complexType>
    <xs:element name="element">
        <xs:complexType>
            <xs:element ref="fieldname" >
        </xs:complexType>
    <xs:element ref="fieldname" type="xs:string" maxOccurs="unbounded"/>
</xs:schema>
```

规范中根元素 root，定义一个数据库中基本表的开始，根元素下有一个或多个 element 元素，类型为字符串。

element 元素，定义数据库中的一条记录，其中每个记录中包含多个字段，类型为字符串。

fieldname 元素，定义数据库中每条记录中的一个字段，类型为字符串。

图 2.9 是对图 2.4 应用元数据图中“Sheet2”的包装结果。

图 2. 9 reml 包装数据库实例图

2.2.2 文本信息 ttpml 包装规范

对文本信息的包装也是完全按照文本的格式进行的。以下是文本包装 ttpml 规范的 schema。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="Tex">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Txt_title"/>
                <xs:element ref="Txt_par_num"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Txt_title" type="xs:string"/>
    <xs:element name="Txt_par_num" type="xs:string" maxOccurs="unbounded"/>
</xs:schema>
```

根元素 Tex，定义一个文本的开始，类型为字符串。Tex 中包含一个 Txt_title 子元素和多个 Txe_par_num 元素。

元素元素 Txt_title，定义一个文本的标题，类型为字符串。

元素 Txt_par_num，定义文本的每一个段落，类型为字符串。

图 2.10 为一文本的包装结果。

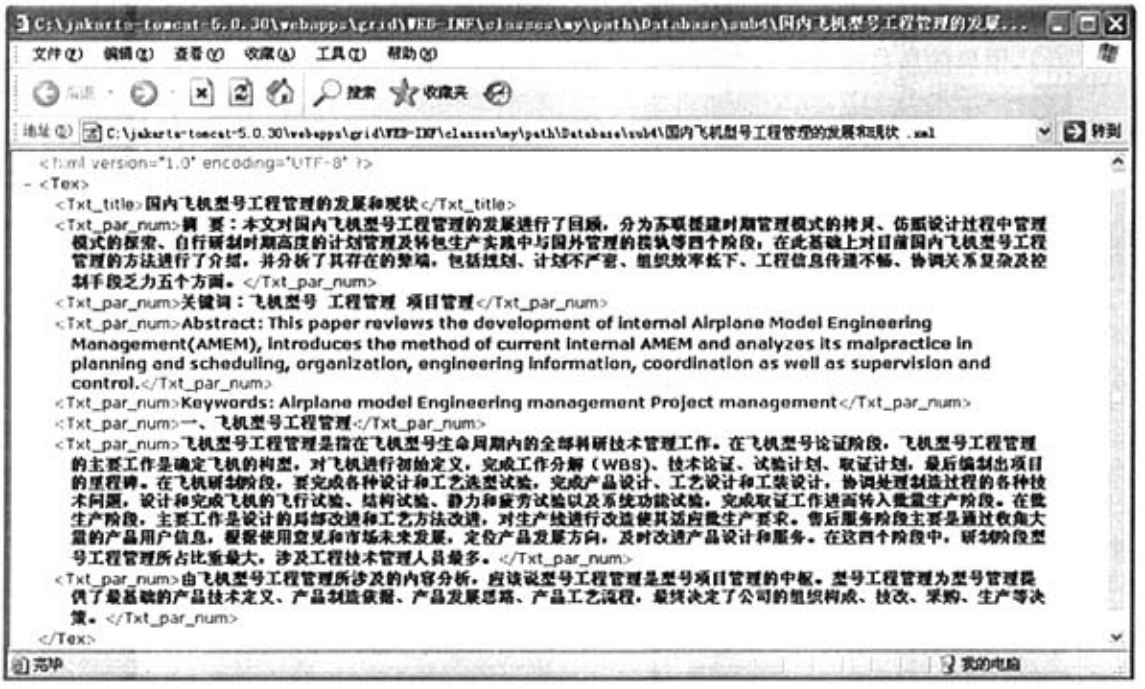

图 2. 10 ttpml 包装数据库实例图

2.3 一个典型的信息访问过程

一个典型的信息访问过程应为：发布—发现过程。信息发布员在全局注册中心注册一个账户，全局注册中心对信息发布员授权以后，该信息发布员可以发布信息。发布信息之前，首先对所发布信息本地注册，本地注册成功后，访问信息的 url 发布到全局注册中心。消费者通过基于关键字的查询发现所需信息的 url，最后消费者通过 url 访问信息，最终得到包装后的 reml 信息或者 ttpml 信息。总括为信息发布员注册、本地注册发布信息、用户发现信息、访问信息这几个重要的过程。

2.3.1 信息发布员注册

主要完成信息发布员本地注册和全局注册。其中信息发布员本地全局注册必须提供以下信息：用户名、密码、信息发布者类别、信息发布者联系方式、工作证号、用户角色。注册成功后，全局注册中心返回一个发布号，以供发布信息使用。本地注册表 uddireg.xml 中写入信息发布员注册信息。信息发布员注册信息成功后，需要拿自己的工作证证明自己的身份，申请全局注册中心授权此用户为信息发布员，具有发布信息的权利。

2.3.2 信息本地注册、发布

信息发布员注册以后，就具有发布信息的权限。如图 2.11 是发布信息的界面。

图 2. 11 注册发布信息界面

信息资源类型对应信息所属部门，例如，技术中心、财务部等。信息资源类型包括 EXCEL、ACESS、SQLSERVER、TXT、XML、PIC 资源，可选其一进行注册。资源物理位置为所注册目录资源所在绝对路径。信息资源描述可填入的信息概要属性。资源状态可选择静态或动态。资源生命周期为“YYYY-MM-DD”格式。信息资源属性可选择只读、读写、只写三者其一。若注册的资源为静态资源则信息资源表名文本框内填入基本表名，如某 Excel 数据库中基本表的名字 Sheet1。若注册的资源为动态资源则信息资源表会自动变为动态资源变化周期，对应文本框内填入动态资源变化周期。数据源只对数据库资源而言，对应注册资源前期通过 JDBC-ODBC 连接数据库时数据源名字。对于其他信息此项为不可编辑状态。发布者名字和发布者密码为全局注册中心注册的用户和密码。发布资源描述信息，描述信息的详细内容，方便用户在全局注册中心发现信息。点击按钮“注册本地资源并发布”将自动触发本地基本表注册服务，并向信息中心发布此信息资源。在图中我们了发布一个规划部数据库中的基本表“十一五主要经济指标规划”。

2.3.3 信息发现

信息发布成功后，用户可以在信息中心的 JSP 页面进行基于关键字的查询发现信息。查询结果如图 2.12 所示。

图 2.12 全局注册中心发现信息界面

2.3.4 信息访问

Loglo 系统提供三种客户端的访问方式：JSP 客户端、Java 客户端、C 客户端。如图 2.13 为 JSP 客户端查询结果。

财务收支	二零零五年	二零零六年	二零零七年	二零零八年	二零零九年	二零一零年
null	null	null	null	null	null	null
备 注	null	null	null	null	null	null
工业增加值（万元）	6000.0	8000.0	9300.0	11500.0	13200.0	16300.0
工业总产值（万元）	18000.0	25000.0	27500.0	35000.0	38500.0	46000.0
劳动生产率（万元/人）	5.0	6.5	7.8	9.5	10.9	13.6
利润总额（万元）	2700.0	3100.0	3800.0	4500.0	5400.0	6800.0
销售收入（万元）	17900.0	24500.0	25000.0	32000.0	37500.0	41000.0

图 2.13 JSP 访问结果图

第三章 遗留代码包装

3.1 遗留代码包装简介

包装遗留代码，采用 XML 技术将遗留代码包装成为 XML 中间表示，便于上层应用的处理和无二义的传输。本文提出的 LC-XML（Legacy Code-XML）方法包装遗留代码是在确定遗留代码包装规范的基础上，对遗留代码添加标签，将遗留代码包装成 fhfml（functions—head—function Markup Language）格式。

在我们的 Loglo 系统中，首先要将遗留代码包装成 fhfml 代码，然后对 fhfml 代码进行本地注册。遗留代码在本地注册成功后，将遗留代码的功能描述信息发布到全局注册中心。信息消费者在全局注册中心通过基于关键字的查询发现该信息，并通过调用提供该信息的服务得到包装后的遗留代码 fhfml，客户端应用程序得到包装后的遗留代码 fhfml 后，根据我们已经确定的 fhfml 规范中标签的意义，准确地读取预期需要的遗留代码，并以友好的方式显示给信息消费者，同时客户端应用程序也可以将 fhfml 作为中间表示缓存在本地，供其他应用程序使用。

本章节中把未经过包装的遗留代码称为裸代码，经过包装的遗留代码称为 fhfml 代码。

3.2 整体框架设计

通过拷贝等方法将遗留裸代码移动到提供者结点，提供者结点首先对遗留裸代码进行预处理、分析，然后完成遗留裸代码的包装，生成 fhfml 代码，通过本地注册模块完成遗留代码的本地注册。最后由提供者结点的发布模块将本地注册的遗留裸代码的功能描述信息和访问方式发布在全局注册中心。消费者结点在全局注册中心发现 fhfml 信息的访问方式，通过访问提供者结点的共享信息服务模块实现对包装后的遗留代码 fhfml 代码的访问。整体框架如图 3.1。

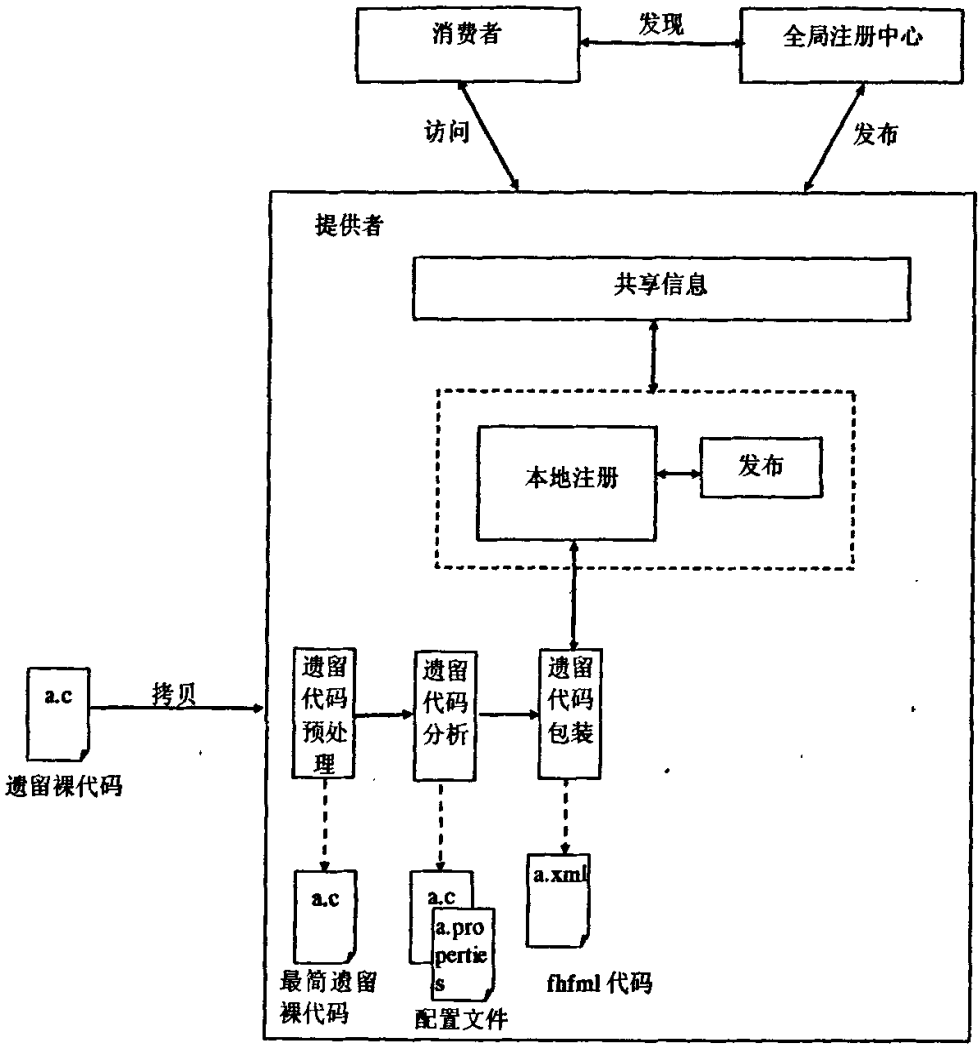

图 3.1 整体框架图

3.2.1 遗留代码预处理

遗留代码预处理的主要目的是得到遗留代码中与某一指定功能实现相关的语句。此模块用到了程序切片技术[27-33]。程序切片技术可以将一个大的程序分解成一个小的、只包含与某个给定变量的计算有关的语句。所有这些语句的集合就叫做一个切片。切片能完整复现原始程序中我们所关注的某一部分行为。遗留代码预处理功能模块主要是根据信息发布员确定的切片准则[34]，使用 CASE 程序切片工具 Unravel[35]截取遗留裸代码中的特定切片。与此同时，为了方便包装遗留代码，我们删除切片代码中所有与某一语句同居一行的注释语句。通过遗留代码预处理后，仅包含与目标功能所对应的那部分代码，我们把通过预处理以后生成的代码称为最简遗留裸代码。

3.2.2 遗留代码分析

遗留代码分析功能模块主要借助第三方分析工具 cflow[36]得到最简遗留裸代码中每一个用户自定义函数的开始行标号和结束行标号，最终生成一个与最简遗留裸代码同名的配置文件，如 a.properties。我们在配置文件中记录最简遗留裸代码每一个函数的开始行标号和结束行标号以及我们自定义的功能描述，如配置文件中"cp.func.begin1=3"表示某个 C 代码的第一个函数开始行号为 3，而"cp.func.end1=15"表示此 C 代码的第一个函数结束行号为 15，"cp.func.descrip1=/*C 函数功能:人民币金额大小写转化,返回大写人民币金额;输入参数:参数 1 小写人民币金额*/"表示某个函数的功能为"/*"和"*/"中的内容。

3.2.3 遗留代码包装

遗留代码分析之后，对最简遗留裸代码包装。遗留代码包装功能模块根据预先确定的遗留代码包装规范 fhfml，通过最简遗留裸代码的配置文件对最简遗留裸代码进行分析，最后包装最简遗留裸代码并生成 fhfml 代码。遗留代码包装流程如图 3.2 所示。输入 fhfml 规范、最简遗留裸代码 a.c、最简遗留裸代码配置文件 a.properties，通过最简遗留裸代码包装处理过程，最终输出 fhfml 代码 a.xml。

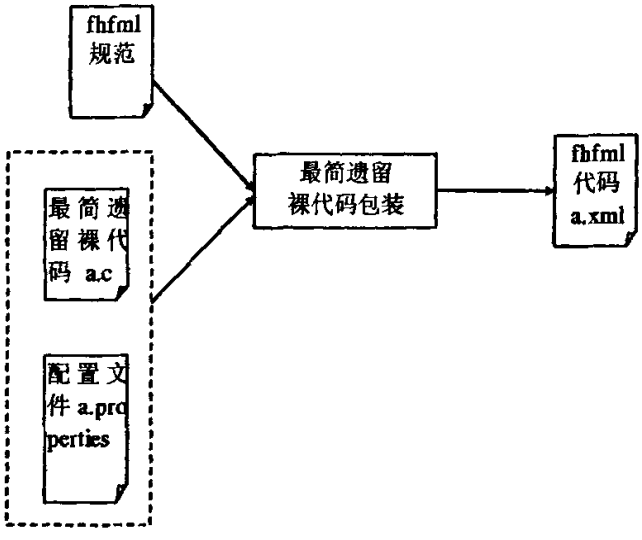

图 3.2 遗留裸代码包装流程图

通过上述分析，要最终完成将遗留代码包装成 fhfml 代码，要经过遗留代码的预处理、分析、包装三个重要的过程。下表概括了各模块的目标及相关说明。

处理过程	遗留代码预处理	遗留代码分析	遗留代码包装
目标	生成最简遗留裸代码。	生成最简遗留裸代码对应的配置文件。	生成 fhfml 代码。
说明	根据信息发布员确定的切片准则，删除遗留裸代码中冗余语句，仅包含与指定功能所对应的那部分代码	生成与最简遗留裸代码同名的配置文件。该文件中写入最简遗留裸代码中每个用户自定义函数的开始行标号、结束行标号以及功能描述。	根据 fhfml 规范生成与遗留裸代码同名的 XML 文件。

3.2.4 本地注册及发布

（1）遗留代码本地注册

通过本地注册模块完成遗留代码的提供者结点本地注册。在 Loglo 系统中本地注册模块由 JDBC 接口模块、基于 XML 元信息管理模块、自动访问模块和信息发布员管理模块构成。对遗留代码的注册不需要访问 JDBC 接口模块和信息发布员管理模块。遗留代码本地注册时，首先由自动访问模块在本地注册表 reg.xml 中写入最简遗留裸代码的信息，并生成相应的以 sub 为前缀，再加上 resource_id 为名的目录（以后提此目录名均简称为 subresource_id）。然后由自动访问模块将由包装模块生成的 fhfml 代码放在 subresource_id 目录下，随后自动访问模块在本地注册表 database.xml 中写入遗留代码的控制信息，最后自动访问模块调用元信息管理模块，将遗留代码元信息写入本地注册表 table_info.xml 中。遗留代码本地注册如图 3.3。

具体过程如下：

①自动访问模块通过访问本地注册表 reg.xml 得到一个信息种类号 resource_id，并生成一个以 subresource_id 为名的目录（如以 sub5 为名的目录），并在本地注册表 reg.xml 中写入最简遗留裸代码信息。最简遗留裸代码信息包括：信息种类号 resource_id、信息资源所属目录物理位置。

②自动访问模块调用拷贝 fhfml 代码模块。

③拷贝 fhfml 代码模块将由遗留代码包装模块在最简遗留裸代码同目录下生成与其同名的 fhfml 代码（如最简遗留裸代码为 a.c 则其对用的 fhfml 代码为 a.xml）拷贝到以 subresource_id 为名的目录下。

④自动访问模块调用控制信息写入注册表模块。

⑤控制信息写入本地注册表 database.xml 时先生成一个信息种类号 table_id，然后将最简遗留代码和 fhfml 代码的相应信息写入本地注册表 database.xml。其控制信息包括：信息种类号 resource_id、信息号 table_id、信息资源所属部门、信息资源状态、信息资源生命周期、信息资源属性、信息资源描述、最简遗留裸代码物理位置、生成 fhfml 代码名、生成 fhfml 代码物理位置、fhfml 信息等级。

⑥自动访问模块调用基于 XML 元信息管理模块。

⑦基于 XML 元信息管理模块将最简遗留裸代码的元信息写入本地注册表 table_info.xml 中。其中元信息包括：信息种类号、信息号、遗留代码用户自定义函数名以及函数功能描述。

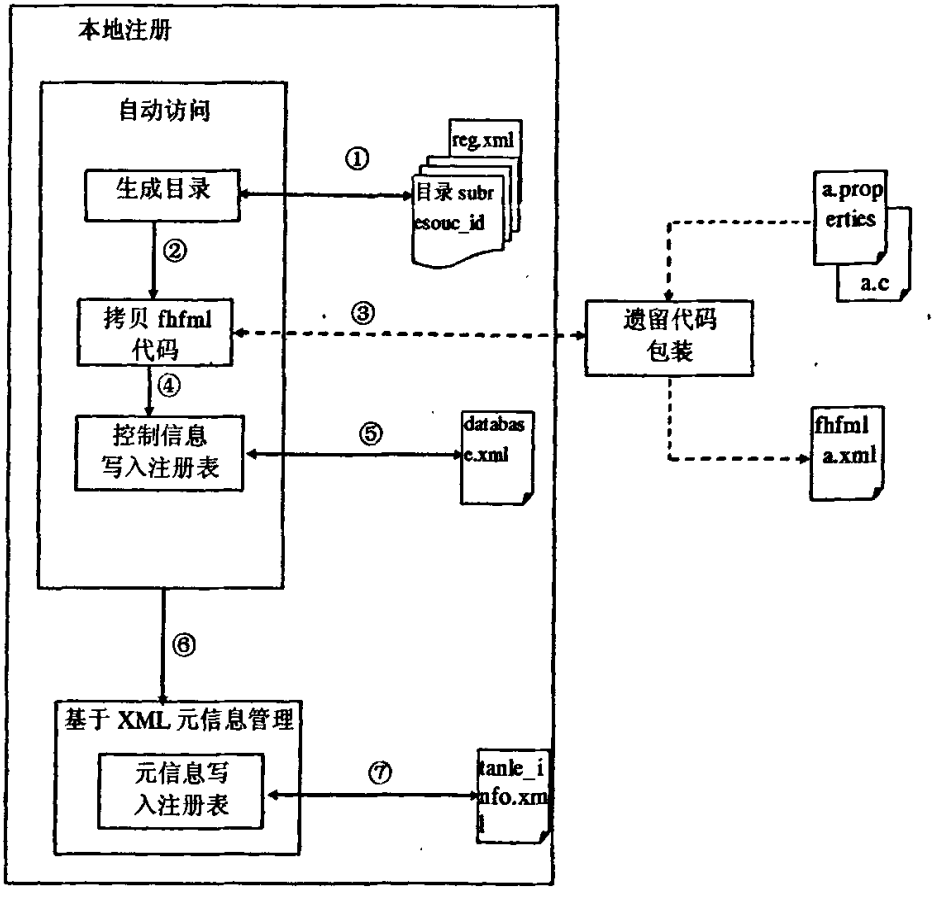

图 3.3 遗留裸代码本地注册图

（2）遗留代码的发布

遗留代码发布时首先通过应用程序自动解析提供遗留代码信息服务的 WSDL 文档，获得提供信息服务的 url，同时将服务 url、信息种类号 resource_id、信息号 table_id、访问遗留代码的方法名构成访问遗留代码信息的 url，然后将此 url 以及本地注册表 table_info.xml 中所注册的遗留代码中每一个用户自定义函数的功能 des 发布到全局注册中心。如图 3.4 所示发布的

关键信息如图中虚框所示。

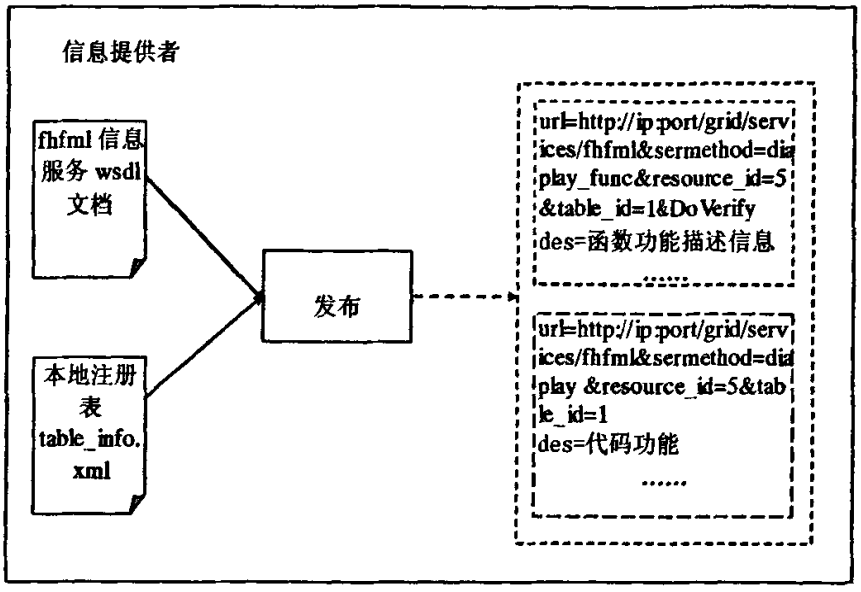

图 3.4 遗留代码发布图

3.3 关键技术

3.3.1 XML 技术

XML 是从 SGML 和 HTML 发展而来。随着应用的发展，SGML 和 HTML 内在的缺陷已不能满足应用进一步的需求，XML 正是为了解决它们的不足而诞生的。XML 实际上是一种元语言，可以让人们根据需要，自行定义标签及属性名，从而突破了 HTML 固定标记集合的约束。

XML 一推出就被广泛地采用，并且得到越来越多的数据库、Internet 软件厂商的支持。从总体来讲，XML 具有以下的特点[37]：

（1）自描述性：XML 允许自定义标签（tag），并且这些标签可以说明数据的语义，而不是 HTML 中的格式说明。

（2）半结构化：不同于纯文本中的数据，XML 数据可以用 DTD 或者 Schema 来规范；但同时 XML 表示的是一种层次型的数据，因此比传统的数据库更适合描述现实中的信息。

（3）机器可处理的：相对于完全无规则的文本，计算机很容易处理 XML 文档，同时相对于无法表示语义的 HTML 文档，计算机很容易理解 XML 文档的语义，并且这种可读性对人同样适用。

（4）可扩展性：XML 是一种元标记语言，可用来定义各种实例标记语言标准，用户可以为数据定义他们自己的词汇表，从而定义数据的处理方式。

（5）广泛的支持：XML 得到了众多的软件厂商比如 Microsoft、IBM、Sun、Oracle 等大公司的支持。

正是因为 XML 具有以上的特点，所以我们采用 XML 技术包装遗留代码。

3.3.2 LC-XML 方法包装规范

3.3.2.1 确立规范过程

包装遗留代码，应该遵循以下三点：

首先，保证不改变经过分析以后遗留代码的功能；

其次，包装后的遗留代码结构清晰，易于理解；

最后，包装后的遗留代码易于客户端解析以及其它应用的使用。

依据以上三点，确立遗留 C 代码包装规范的 schema。

分析 C 代码的结构。一个独立的 C 源程序包含一个主函数 main。程序开始执行的时候，首先调用主函数，然后通过主函数中的函数调用语句调用其它函数，其它函数之间也可以互相调用。在某种程度上可以说，C 源程序全部是由函数组成[38]。当一个源程序太大时可分为几个源文件。图表示 C 代码的程序结构。

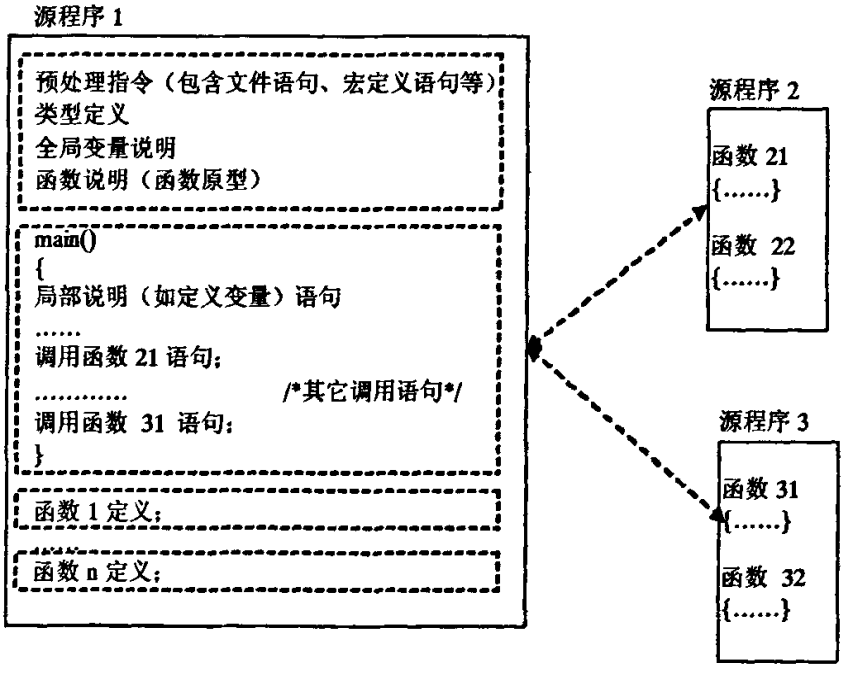

图 3.5 C 代码结构图分析

从上图可以看出一个 C 源代码由预处理指令、类型定义、全局变量说明、函数说明以及用户自定义函数构成，而预处理指令包括定义宏指令、取消宏定义指令、文件包含指令、条

件编译指令、出错指令、报告现行程序有汇编代码指令等以“#”开头的指令。

根据C源代码的结构，包装代码的时候将源程序分为两个部分，预处理指令、类型定义、全局变量说明、函数说明构成第一部分，我们称为“Head”部分，用户根据需要编写的专门用来实现特定功能的用户自定义函数构成第二部分，我们称为“Function”部分。其中Head部分可以包含预处理指令、类型定义、全局变量说明、函数说明中的全部、任意一个或者几个的组合。根据这样的划分，一个C源代码可以表示为这样的结构“Head部分、Function部分、Head部分、Function部分、……”，依据程序设计的风格和程序的具体功能，其中预处理指令、类型定义、全局变量说明、函数说明是可有可无的，因此Head部分可有可无。因为C源程序中函数定义都是独立的一部分，即函数不允许嵌套定义[38]，因此不存在Function部分中包含Function部分这样的结构。通过上述分析，使用XML对遗留代码加标签，实现遗留代码的包装。将Head部分放在XML标签<Head>元素中，将Function部分放在XML标签<Function>中。下图是对一个C源程序的初步包装轮廓。

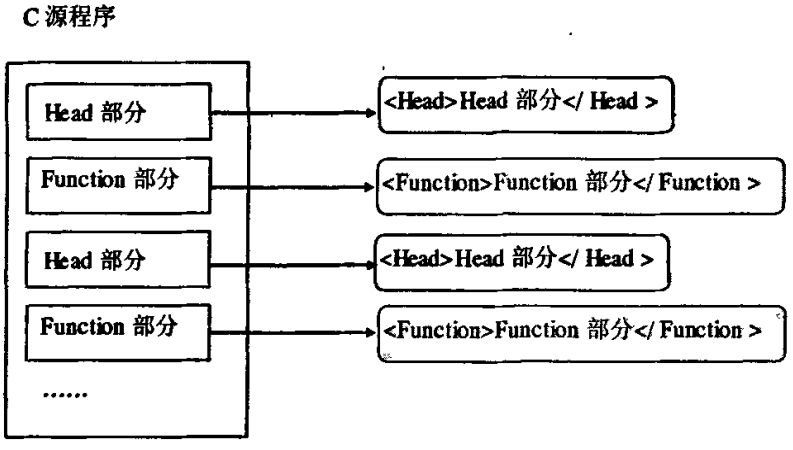

图3.6 C源程序的初步包装轮廓

因为Head部分所包含的预处理指令、类型定义、全局变量说明、函数说明没有严格的顺序规定，灵活性比较大，因此不在对Head部分进行更细的分析，进一步加标签。

每一个Function部分就是一个用户自定义函数，图3.7表示用户自定义函数说明。

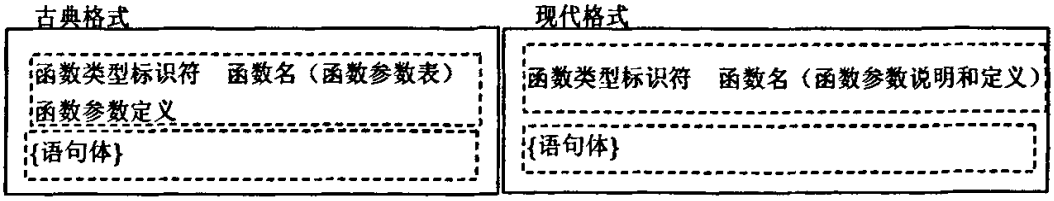

图3.7 函数定义

从上图看出 C 代码的用户自定义函数包括函数头和函数体，为了包装我们将函数头称为 Function_head 部分、函数体称为 Function_body 部分。因此一个用户自定义函数可以看作由 Function_head 部分和 Function_body 部分构成。对 Function_head 部分和 Function_body 部分分别加标签。下图表示对 Function 部分加 xml 标签的轮廓。

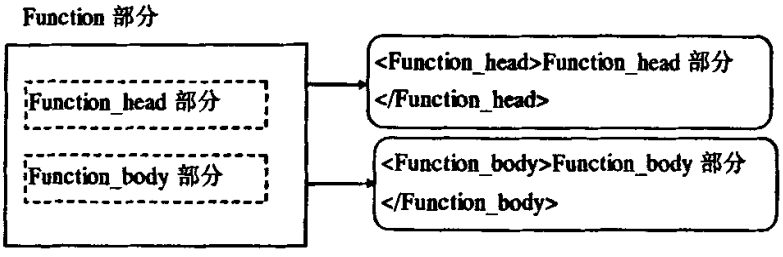

图 3.8 Function 部分加 xml 标签图

使用一个函数需要知道函数名、返回值类型、输入参数个数以及各个参数的类型，因此需要对 Function_head 部分细分。Function_head 部分包括函数类型标识符、函数名以及函数参数定义即 Type_identifier 部分、Function_name 部分、Parameters 部分。其中 Type_identifer 部分包含区分内外部函数的 static_extern 部分、函数的返回类型 Return_type 部分、区别函数是近调用还是远调用 near_far 部分以 TC 特有的修饰符 interrupt 部分、pascal_cedecl 部分，如图 3.9 所示。由于函数功能具体实现 Function_body 部分中使用的是函数参数名，因此在 Parameters 部分不仅包含每一个用户自定义函数中参数的类型 Parameter_Type 部分还应该包含函数名即 Parameter_Name 部分。

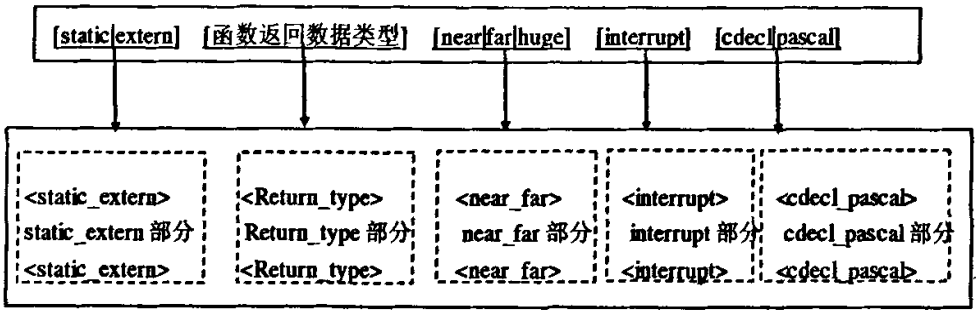

图 3.9 函数标识符部分加标签示意图

Function_body 部分中为函数具体功能的实现，如果再将 Function_body 分析，进一步加标签，难度大，而且加标签以后程序难于解析，因此我们不对 Function_body 部分进一部分析加标签。

由于每个用户自定义函数完成了整个代码的一部分功能，而函数的功能直观的通过 Function_body 部分看出比较困难，并且对于一个消费者来说在重用代码之前看懂代码是一个

不合理的要求。因此我们在每一个 Function 部分增加用户自定义函数的功能描述，将其放在标签<Description>中。

通过上面的分析，使用我们的这种方法经过加标签以后的遗留裸代码即没有改变原始代码的功能而且包装后的代码结构清晰、易于人的理解和应用程序的解析。

3.3.2.2 规范的 schema

经过上面的分析，得到一个简单的包装后遗留代码的结构图，如图 3.10 所示。根据 XML 规范，在一个 XML 文档中必须有一个根元素，而 Head 元素和 Function 元素属于同级元素，整个包装后 XML 文档没有根元素，因此增加根元素 Functions，也符合“一定程度上，C 代码是由函数构成的”这种说法。因为在 Functions 根元素下，只能出现 Head 元素和 Function 元素，所以我们把包装后的遗留代码称为 fhfml（Functions-Head-Function Market Language）代码。

```
<?xml version="1.0" encoding="UTF-8"?>
<Functions>
<Head>Head 部分</Head>
<Function>
  <Description>函数功能描述部分</ Description >
  <Function_head >
   <Type_identifer>
     <Return_type>Return_tyep 部分</Return_type>
     ......
   </Type_identifer>
   <Function_name>Function_name 部分</Function_name>
   <Parameters parameter_num="参数个数">
     < Parameter>
        <type>参数类型</type>
        <name>参数名</name>
     </ Parameter>
      ......
   </Parameters>
  </Function_head>
  <Function_body>
   Function_body 部分
  </Function_body>
</Function>
  ......
<Functions>
```

图 3.10 包装后遗留代码的结构图

通过对 C 源代码结构的分析，得到对遗留 C 代码加 xml 标签包装的后的框架结构，因为包装后的遗留代码 fhfml 代码是 XML 文件格式，而 XML 有自己的描述文件结构的语言，DTD 和 XML Schema。我们采用 xml schema 来描述 fhfml，在 fhfml schema 中定义和描述 XML 文档的结构和内容，元素与元素之间的关系以及元素和属性的数据类型。包装遗留代码生成 fhfml schema 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="Functions">
        <xs:complexType>
            <xs:sequence>
                <xs:choice>
                    <xs:group ref="type" maxOccurs="unbounded"/>
                    <xs:element ref="Head"/>
                </xs:choice>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Head" type="xs:string"/>
    <xs:element name="Function">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Description"/>
                <xs:element ref="Function_head"/>
                <xs:element ref="Function_body"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Description" type="xs:string"/>
    <xs:element name="Function_head">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Type_identifier"/>
                <xs:element ref="Function_name"/>
                <xs:element ref="Parameters"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Type_identifier">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="static_extern" minOccurs="0"/>
                <xs:element ref="Return_type"/>
                <xs:element ref="near_far" minOccurs="0"/>
```

```
                <xs:element ref="interrupt" minOccurs="0"/>
                <xs:element ref="cdecl_pascal" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="static_extern">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="static"/>
                <xs:enumeration value="extern"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="near_far">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="near"/>
                <xs:enumeration value="far"/>
<xs:enumeration value="huge"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="interrupt" type="xs:string" fixed="interrupt"/>
    <xs:element name="cdecl_pascal">
        <xs:simpleType>
        <xs:restriction base="xs:string">
        <xs:enumeration value="cdecl"/>
        <xs:enumeration value="pascal"/>
        </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="Return_type" type="xs:string"/>
    <xs:element name="Function_name" type="xs:string"/>
    <xs:element name="Parameters">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Parameter" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="parameter_num" type="xs:integer" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Parameter">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Parameter_Type"/>
```

```
            <xs:element ref="Parameter_Name"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Parameter_Type" type="xs:string"/>
<xs:element name="Parameter_Name" type="xs:string"/>
<xs:element name="Function_body" type="xs:string"/>
<xs:group name="type">
    <xs:sequence>
        <xs:element ref="Head" minOccurs="0"/>
        <xs:element ref="Function"/>
    </xs:sequence>
</xs:group>
</xs:schema>
```

fhfml schema 中的关键元素：

（1）元素 Functions 是复杂类型，定义了 fhfml 的根元素表示 fhfml 代码的开始，同时定义了在根元素 Functions 只包含子元素 Head 和 Function，schema 中规定元素的可以出现任意多次，但是对 Head 和 Function 出现次序有一定的限制。元素 choice 定义了 Functions 元素中子元素 Function 和元素 Head 有如下两种合法的出现序列组合：

①group 元素 type 定义了这样的出现序列：

无 Head 元素或者 Head 元素仅出现一次，Function 元素仅出现一次。即这样的两种可能形式，<Function>和<Head><Function>。同时在 choice 元素中引用 type 元素时，定义这两种序列均可出现无数次。如根元素 Functions 中的子元素 Head 和 Function 出现次序如下：

有多个 Head 元素和 Function 元素间隔出现如，<Head>、<Function>、<Head>、<Function>、……；只有一个 Head 元素，Head 元素位置不限制，其余都是 Function 元素<Head>、<Function>、<Function>、<Function>、……；没有 Head 元素只有 Function 元素<Function>、<Function>、<Function>均为合法的 Functions 子元素出现序列。

②根元素 Functions 中没有 Function 元素只有一个子元素 Head。

通过上述分析得出，在 fhfml shema 中不允许多个 Head 元素连续出现，即<Head>、<Head>、……，这是不合法的 Functions 子元素出现序列。因为在包装的过程中，我们把函数作为一个整体 Function 部分包装，连续两个函数之间的部分预处理等语句都作为 Head 部分包装或者代码中第一个函数开始前面所有语句都作为 Head 部分包装，也就是说遇到非函数并且没有被函数隔开的语句就归为 Head 部分，因此不存在连续出现 Head 元素的情况。

（2）元素 Head，定义了 fhfml 的根元素下的子元素 Head。此元素对应 C 代码的 Head 部分，其元素值表示 C 代码中预处理指令、类型定义、全局变量说明、函数说明。

（3）元素 Function 是复杂类型，定义了 fhfml 的根元素下的子元素 Function，其元素值

表示 C 代码的一个用户自定义函数。该元素下有三个子元素 Description、Function_head 和 Function_body，scheme 中对子元素 Description、Function_head 和 Function_body 出现次数和出现次序都做了限制，即出现次序仅能如<Description>、<Function_head>、<Function_body>且各个元素只能出现一次。

Function 元素对应我们在前面分析的 C 代码的 Function 部分，元素 Description、Function_head 和元素 Function_body 分别对应前分析的函数功能描述部分、Function_head 部分和 Function_body 部分。

①元素 Description，定义了元素 Function 的第一个子元素。其元素值表示 C 代码的函数功能描述部分。

②元素 Function_head 是复杂类型，定义了元素 Function 的第二子元素，对应上节分析的 C 代码的 Function_head 部分，即函数头。Function_head 包含三个子元素 Function_identifier 元素、Function_name 元素和 Parameters 元素。

a、Function_identifier 元素是复杂类型，定义了元素 Function_head 的第一个子元素。其元素值为用户自定义函数类型标识部分。该元素包括 5 个子元素依次为：

static_extern 元素定义了一个用户自定义函数类型，内部函数或外部函数，并规定元素值为 static 或 extern。此元素为可选项。

Return_type 元素定义了用户自定义函数的返回值。

near_far 元素定义了用户自定义函数的调用方式，近调用或远调用，并规定元素值为 near、far 或者 huge。此元素为可选项。

interrupt 元素规定元素值为 interrupt，interrupt 为 TC 特有的修饰符。此元素为可选项。

cdecl_pascal 元素规定元素值为 cdecl 或 pascal，cdecl 和 pascal 为 TC 特有的修饰符。此元素为可选项。

b、Function_name 元素定义了元素 Function_head 的第二个子元素，元素值表示用户自定义函数名字。

c、Parameters 是复杂元素类型，定义了用户自定义函数的每个参数说明，此元素包含 0 个或多个 Parameter 元素。

Parameter 元素定义了函数的一个参数，其中子元素 Parameter_name 定义参数名，Parameter_type 定义了函数参数类型。

Shema 中对 Parameters 增加了属性说明，属性名为“parameter_num”表示 C 代码中函数的参数个数。

③元素 Function_body，定义了元素 Function 的第三个子元素。对应前分析的 C 代码的 Function_body 部分，其元素值表示函数体。

综上所述，通过对 fhfml schema 中元素之间关系的分析，可以得到 fhfml 文档内容层次关系结构图如 3.11。从图中可以看出 fhfml schema 规定的代码包装以后有一个根元素 Functions，其下子元素可以任意出现，但不允许连续的 Head 元素出现。其中 Function 元素有三个子元素 Description、Function_head 和 Function_body，它们仅出现一次。

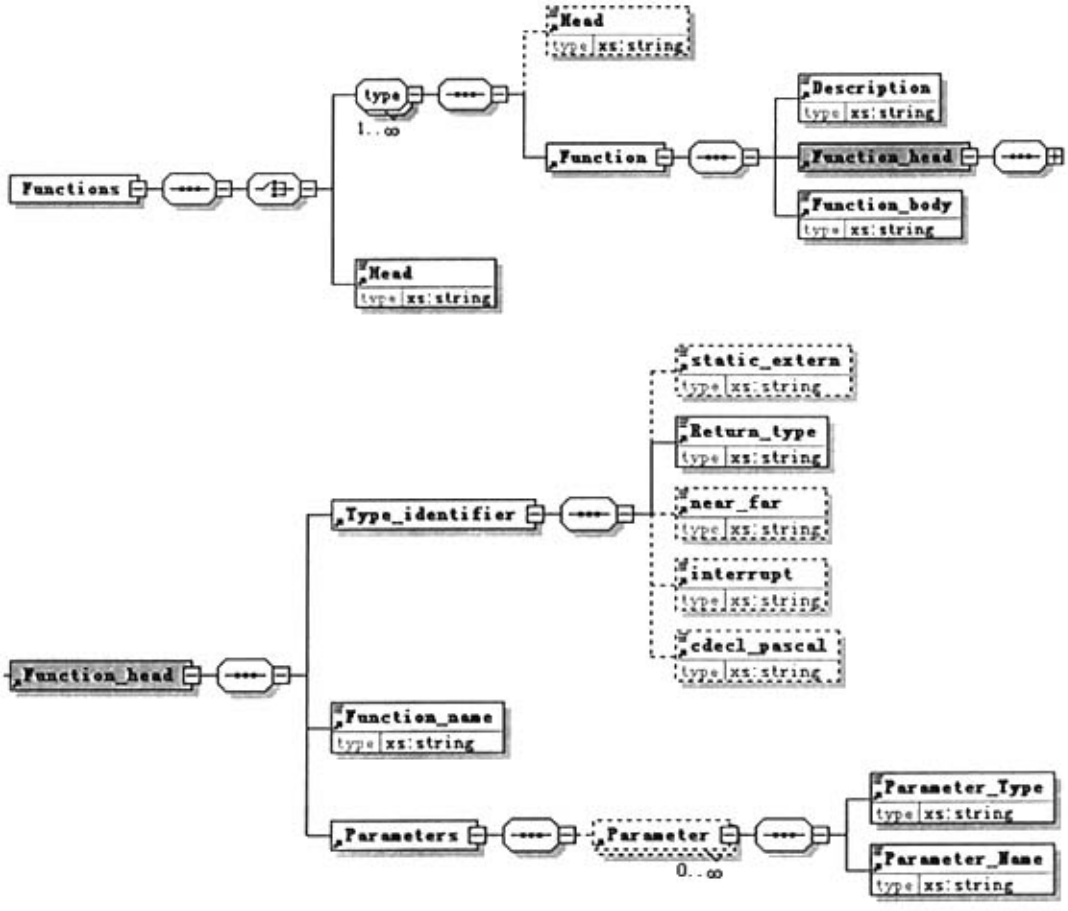

图 3. 11 fhfml 规范 schema 结构图

3.3.2.3 规范补充

在 XML 中，有五个字符在解析的时候须替换成相应的实体引用[39]。即“<”、“>”、“&”、“””、“’”须分别替换成“<”、“>”、“&”、“"”、“'”。只有“<”字符和“&”字符在 XML 文档中是严格禁止使用的，剩下的都是合法的。在遗留代码中出现大于号、小于号、双引号以及和符号是很常见的。如果不对这些转义字符处理，使用时就会出问题。使用下面的例子来说明会出现的问题，从而对 fhfml 规范进行补充。使用 fhfml 规范包装遗留代码 isprime.c，isprime.c 源代码如下所示：

```
int isprime(int x)
```

```
{int div,flag,r;
 div=2;
 flag=1;
 while ((div<=sqrt(x)) && (flag==1))
{r=x%div;
if (r==0) flag=0;
else div=div+1;
}
if(flag==1) return(1);
else return(0);
}
```

如果采用记事本的方法打开包装后的遗留代码 isprime.xml 如图 3.12。

图 3.12 isprime.xml 用记事本打开图

从上图可以看到遗留裸代码中的"<"字符和"&"字符全部在包装的过程中由包装程序自动转化为实体引用"<"和"&",而以浏览器的方式打开 fhfml 时候,浏览器会自动将实体转化为原来的字符,因此在浏览器的方式下这些实体引用会被浏览器自动解析成对应的字符。信息消费者使用信息的方式是多样的,我们不能假定永远使用浏览器打开 fhfml 代码。为了解决遗留代码在包装过程中的转义字符问题,我们使用 XML 中的 CDATA(character data 字

符数据）段[40]，CDATA 段可以出现在字符数据可以出现的任何地方，它们用于转义包含会被识别为标记的字符串文本。通过将文本放在 CDATA 段中来处理元素内容中的特殊字符，在 CDATA 段中所有文本都是纯字符数据，XML 处理器无论如何是不会解释这些字符的。因此在包装遗留代码的时候，需要将遗留代码放在 CDATA 段中，具体用 JAVA DOM 实现的时候只要将 doc.createTextNode(lcode)，改成 doc.createCDATASection(lcode)既可以实现，lcode 表示遗留裸代码。最后遗留代码包装结构如图 3.13。

```
<?xml version="1.0" encoding="UTF-8"?>
<Functions>
<Head><![CDATA[Head 部分]]></Head>
<Function>
  <Description><![CDATA[函数功能描述部分]]></ Description >
  <Function_head >
   <Type_identifer>
     <Return_type>Return_tyep 部分</Return_type>
     ......
   </Type_identifer>
   <Function_name>Function_name 部分</Function_name>
   <Parameters parameter_num="参数个数">
     < Parameter>
        <type>参数类型</type>
        <name>参数名</name>
     < /Parameter>
      ......
   </Parameters>
  </Function_head>
  <Function_body>
   <![CDATA[Function_body 部分]]>
  </Function_body>
</Function>
  ......
<Functions>
```

图 3.13 修正后遗留代码包装结构图

可以看出在 xml 标签之间增加了“<![CDATA[”和“]]>”段。通过使用 CDATA 段对 fhfml 规范补充，上例遗留裸代码包装后，通过记事本打开其内容如图 3.14 所示。

6111.xml - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?xml version="1.0" encoding="UTF-8"?>
<Functions>
<Head><![CDATA[#include "math.h"
]]></Head>
<Function>
<Description><![CDATA[/*函数功能:判断是否素数,返回值:"1"表示是素数,"0"表示不
是素数; 输入参: 参数1表示判断该整数是否为素数*/]]></Description>
<Function_head>
<Type_identifier>
<Return_type>int</Return_type>
</Type_identifier>
<Function_name>isprime</Function_name>
<Parameters parameter_num="1">
<Parameter>
<Parameter_Type>int</Parameter_Type>
<Parameter_Name>x</Parameter_Name>
</Parameter>
</Parameters>
</Function_head>
<Function_body><![CDATA[{int div,flag,r;
 div=2;
 flag=1;
 while ((div<=sqrt(x)) && (flag==1))
{r=x%div;
if (r==0) flag=0;
else div=div+1;
}
if(flag==1) return(1);
else return(0);
}]]></Function_body>
</Function>
</Functions>
```

图 3.14 isprime.xml 用记事本打开图

包装后的遗留代码 fhfml 代码的元素值就是遗留裸代码。将这样的 fhfml 信息传送给消费者，信息消费者根据 fhfml 规范，可以得知此遗留 C 代码的只有一个用户自定义函数，没有预处理指令、类型定义、全局变量说明、函数说明，同时此函数头部分为 Function_head 中的内容"int isprime(int x)"，函数中只有一个参数，函数体为 Function_body 中的内容。在遗留代码包装的时候使用 CDATA 段，信息消费者得到 fhfml，可以以各种形式使用信息，而不仅仅局限于浏览器方式。

3.3.3 LC-XML 方法包装步骤

根据 fhfml 规范，我们得到包装遗留代码 LC-XML 方法的步骤。

LC-XML 方法主要包括以下几个重要的步骤：

（1）读与最简遗留裸代码同一目录下的同名配置文件（如最简遗留裸代码为 a.c，则它对应的配置文件为 a.properties），得到最简遗留裸代码每个函数的开始行号和结束行号，即每个

Function 部分的开始和结束行标号（如[3,13,14,35]表示第一个函数开始行标号为 3，结束行标号为 13，第二函数开始行标号为 14，结束行标号为 35）。

（2）生成 Document 对象，并增加根元素 Functions。

（3）识别是否 Head 部分，即函数头。

不存在 Head 部分的两种情况。若第一个函数的开始行标号为 1 则不存在 Head 部分。若前一个函数的结束行标号与后一个函数的开始行标号相差 1，则这两个函数之间不存在 Head 部分。若满足其一则执行步骤 5。

（4）若存在 Head 部分，得到其开始和结束行标号。

分两种情况，若第一个函数的开始行标号大于 1，则 Head 部分的开始行标号为 1，结束行标号为第一个函数的开始行标号减 1。代码中其余的可能存在 Head 部分的开始行标号为前一个函数的结束行标号加 1，对应 Head 部分的结束行标号为下一个函数的开始行标号减 1。

从代码中得到 Head 部分，并为其加标签 Head，将其增加为 Functions 的子元素。

（5）代码是否到达末端，若到达末端则结束。

（6）识别 Function 部分，即函数体。

读配置文件，得到用户自定义函数的功能描述，生成 Description 元素。

根据（1）所得到一个 Function 部分的开始行标号和结束行标号。

①识别 Function 部分的 Function_head 部分和 Function_body 部分即函数头和函数体。

②由 Function_head 部分识别出函数类型标识符 Type_identifier 部分、函数名 Function_name 部分、函数参数定义 Parameters 部分。

在 Type_identifier 部分中识别出函数返回类型标识以及其它的一些标识，并根据 fhfml 规范生成相应的标签，将其作为 Type_identifier 元素的子元素。

为 Function_name 部分增加标签 Function_name。

为 Parameter 部分增加标签 Parameters，同时识别每个参数的参数名和参数类型，分别增加标签 Parameter_name 和 Parameter_type，并将其作为 Parameter 的子元素。元素 Parameter 增加为 Parameters 的子元素，同时为 Parameters 增加表示参数个数的属性 Parameter_num。

生成 Function_head 标签，并为其增加子元素 Type_identifier、Function_name 以及 Parameters。

③给 Function_body 部分加标签 Function_body。

生成标签 Function，并为其增加子元素，Description、Function_head 和 Function_body，同时为根元素 Functions 增加子元素 Function。

LC-XML 方法，通过上面 6 个步骤，按照 fhfml 规范，在保证最简遗留裸代码结构不改变的基础上，完成遗留代码的包装。包装后的遗留代码，通过本地注册和发布最终成为网格

用户可以识别的一类 fhfml 代码信息。

3.3.4 LC-XML 方法包装流程

LC-XML 方法包装遗留代码流程如图 3.15 所示。

第一，读相关文件。遗留代码包装过程中读相关文件即为读与最简遗留裸代码同名的配置文件（如 a.properties），其主要执行 LC-XML 方法的步骤 1。

第二，产生 Document 对象。即生成 DOM 树，其执行 LC-XML 方法的步骤 2。

第三，增加 xml 标签。其执行 LC-XML 方法的步骤（3）、（4）、（5）、（6）。其具体流程如图中虚框所示。其描述如下：

根据步骤（3），识别是否 Head 部分，如果是，则执行步骤（4），否则执行步骤（5）。

根据步骤（4），识别 Head 部分并增加标签。

判断最简遗留裸代码是否达到代码末端，若是则结束加标签，若未达到代码末端则继续向下执行。

根据步骤（6）识别 Function 部分并加标签。在 fhfml 规范的 schema 中规定不允许连续几个 Head 元素出现，在这里只能是 Function 元素，不用判断是否 Function 部分。

第四，生成 fhfml 代码。根据得到的 Doument 对象，将其转化为 xml 文件表示形式。

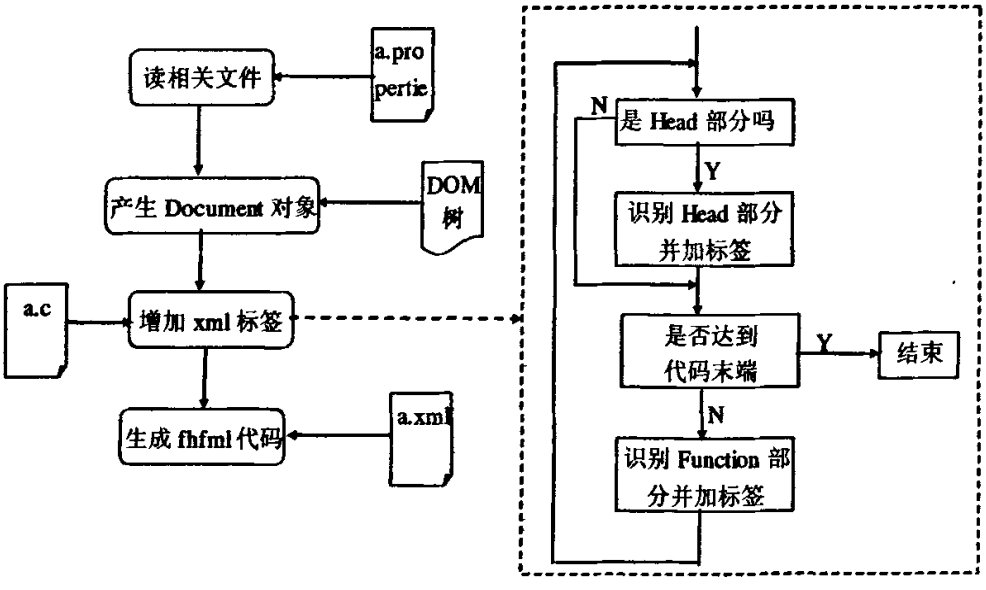

图 3.15 LC-XML 方法流图

3.3.5 基于浏览器的 fhfml 代码解析

遗留代码如果不进行包装，作为一般的信息发布到全局信息注册中心，当信息消费者发

现并且访问的仅仅是一些字符串，按照 fhfml 规范在原本"无意义的字符串"加上严格意义上的标签后，就成为有意义的数据即信息，成为可以被重用的财富。

信息消费者通过响应 SOAP 消息得到的是 fhfml 格式的信息，并不能提供给用户很直观的信息理解，即使是直接使用浏览器打开 fhfml，也只是按照原样带有标签的显示。为了弥补这一缺点，我们基于 DOM 解析器和 JSP 界面编程语言，对 fhfml 信息进行有意义的解析并通过浏览器将获得的信息显示，方便用户对于信息的直观理解。下图是基于浏览器的 fhfml 代码解析框架。

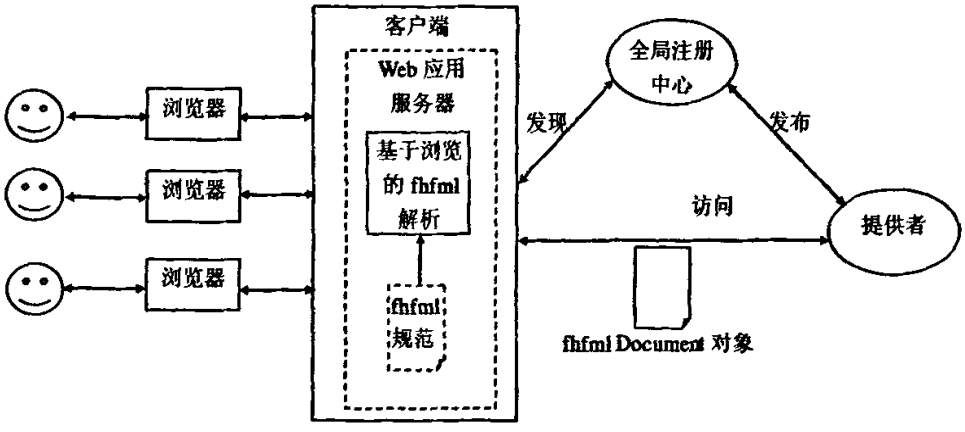

图 3. 16 基于浏览器的 fhfml 解析框架

信息消费者在全局注册中心发现遗留代码的元信息和访问信息的 url，客户端应用程序通过 Web 应用服务器向提供者发送请求并得到响应消息 XML DOCUMENT 对象，应用程序即可以直接对 DOCUMENT 对象解析，也可以在本地缓存一个 fhfml 文件，将此 fhfml 可作为一种中间表示，供上层应用使用。基于浏览器的 fhfml 代码解析过程如下：

（1）客户端应用程序通过访问遗留代码信息服务，得到信息提供者返回的 fhfml DOCUMENT 对象。

（2）根据 fhfml 规范，Functions 标签定义了遗留 C 代码的开始，Head 标签定义了代码中预处理指令、类型定义、全局变量说明、函数说明，Function 标签定义了代码中一个用户自定义函数，而且依据 fhfml 规范生成的 fhfml 是遵循遗留代码的结构。

①客户端应用程序解析 Functions 对象，得到 NodeList 对象，其中包含 Function 元素和 Head 元素。

②根据 JSP 显示界面的设计和 JSP 网页设计语言，将每一个 Function 标签内容和 Head 标签内容各放置在 JSP 的"table"标签中。

③解析 Function 元素，将其子元素 Description、Function_head 和 Function_body 的标

签内容读出，用 JSP 页面分隔标签"hr"将其分开。

④解析 Function_head 元素，读出其子元素 Type_identifier 中的子元素值，将其作为函数标识符，将 Function_head 的子元素 Function_name 作为函数名，将 Parameters 中的每个子元素 Prameter 解析出作为函数的参数定义。

（3）最后在基于浏览器的 JSP 页面中显示除所访问的 fhfm 信息。

从上图和分析可以看到，作为应用层的浏览器可以直接通过访问 JSP 页面而获得 fhfml 代码。这种方法主要依赖于每个用户桌面的浏览器，对用户计算机来说，唯一要做的就是有一个浏览器，以及浏览器可以访问全局注册中心的应用服务器以及和基于浏览器 fhfml 解析结点 Web 应用服务器。

3.4 LC-XML 方法应用举例

用来一个简单的例子来说明我们的 LC-XML 方法包装过程以及结果。

3.4.1 遗留代码预处理和分析

提交一段身份证号码由 15 位升为 18 位的 C 代码，verify.c，如下：

```
#include "string.h"
#define SRC_ID "622801810315022"
char DoVerify(pszSrc)
const char* pszSrc;
{
int iS = 0;
int iW[]={7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2};
static char szVerCode[]="10X98765432";
int i;
int iY;
for(i=0;i<17;i++)/*由 17 位根据相关公式计算出一个校验位，作为 18 位身份证的最后一位.*/
{
 iS += (int)(pszSrc[i]-'0') * iW[i];
 }
iY = iS%11;
printf("%d %% 11 = iY = %d\n",iS, iY);
printf("%c \n",szVerCode[iY]);
return szVerCode[iY];
}
void Make15To18(pSrc, pDes)
char *pSrc;
char *pDes;
{
char szTemp[19] = "";
```

```
int n;
strncpy(szTemp, pSrc, 6); /*把原来 15 位中的年份补全，在前 6 位后加上年份 19.*/
strcat(szTemp, "19");
strcat(szTemp, &pSrc[6]);
n = DoVerify(szTemp);
szTemp[17] = n;
strcpy(pDes, szTemp);
return;
}
void main()
{
char szSrc[16] = SRC_ID;
char szDes[19] = "";
clrscr();
if(strlen(szSrc)   == 15)
{
printf("The 15-bit id: %s\n", szSrc);
Make15To18(szSrc, szDes);
printf("The 18-bit id: %s", szDes);
}
getch();
}
```

如果我们只关心主函数中身份证位数由 15 位转化位 18 位的结果，选择切片准则<44,szDes>，通过 unravel 工具得到与遗留代码的 44 句的 szDes 变量相关的语句，同时删除遗留代码中与代码同行的注释，得到最简遗留裸代码。

```
1. #include "string.h"
2. #define SRC_ID "622801810315022"
3. char DoVerify(pszSrc)
4. const char* pszSrc;
5. {
6. int iS = 0;
7. int iW[]={7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2};
8. static char szVerCode[]="10X98765432";
9. int i;
10. int iY;
11. for(i=0;i<17;i++)
12. {
13. iS += (int)(pszSrc[i]-'0') * iW[i];
14.  }
15. iY = iS%11;
16. printf("%d %% 11 = iY = %d\n",iS, iY);
17. printf("%c \n",szVerCode[iY]);
18. return szVerCode[iY];
19. }
```

```
20. void Make15To18(pSrc, pDes)
21. char *pSrc;
22. char *pDes;
23. {
24. char szTemp[19] = "";
25. int n;
26. strncpy(szTemp, pSrc, 6);
27. strcat(szTemp, "19");
28. strcat(szTemp, &pSrc[6]);
29. n = DoVerify(szTemp);
30. szTemp[17] = n;
31. strcpy(pDes, szTemp);
32. }
33. void main()
34. {
35. char szSrc[16] = SRC_ID;
36. char szDes[19] = "";
37. if(strlen(szSrc)  == 15)
38. {
39. printf("The 15-bit id: %s\n", szSrc);
40. Make15To18(szSrc, szDes);
41. printf("The 18-bit id: %s", szDes);
42. }
43. }
```

进入遗留代码分析阶段，使用 cflow 工具得到 verify.c 中每一个用户自定义函数的开始和结束行标号。手工生成配置文件 verify.properties 内容如下。

```
cp.func.descrip1=/*C 函数功能:由给出的 17 位身份证号码数字计算出最后一位校验码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准);输入参数:参数 1 给出的 17 位身份证号码;返回值:最后一位校验码*/
cp.func.begin1=3
cp.func.end1=19
cp.func.descrip2=/*C 函数功能:给定的老式 15 位身份证号码升为对应的 18 位新式身份证号码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准);输入参数:参数 1 用于存放 15 位身份证号码,参数 2 用于存放计算出的 18 位新式身份号码;返回值:无*/
cp.func.begin2=20
cp.func.end2=32
cp.func.descrip3=/*C 函数功能：主函数，调用子函数 Make15To18 实现 15 位身份证号码身份证号码 622801810315022 升为 18 位身份证号码;输入参数：无;返回值：无*/
cp.func.begin3=33
cp.func.end3=43
```

3.4.2 遗留代码包装和发布

注册 verify.c 代码，注册界面如下。信息资源物理位置主要填写遗留 C 裸代码在本地的

物理位置，信息资源表名和数据源名在注册 LCODE（遗留代码）信息资源的时候为不可编辑状态。其中发布资源描述信息填写的越为详细，信息消费者越是容易发现信息。界面中其它内容可参看 2.3.3 节。

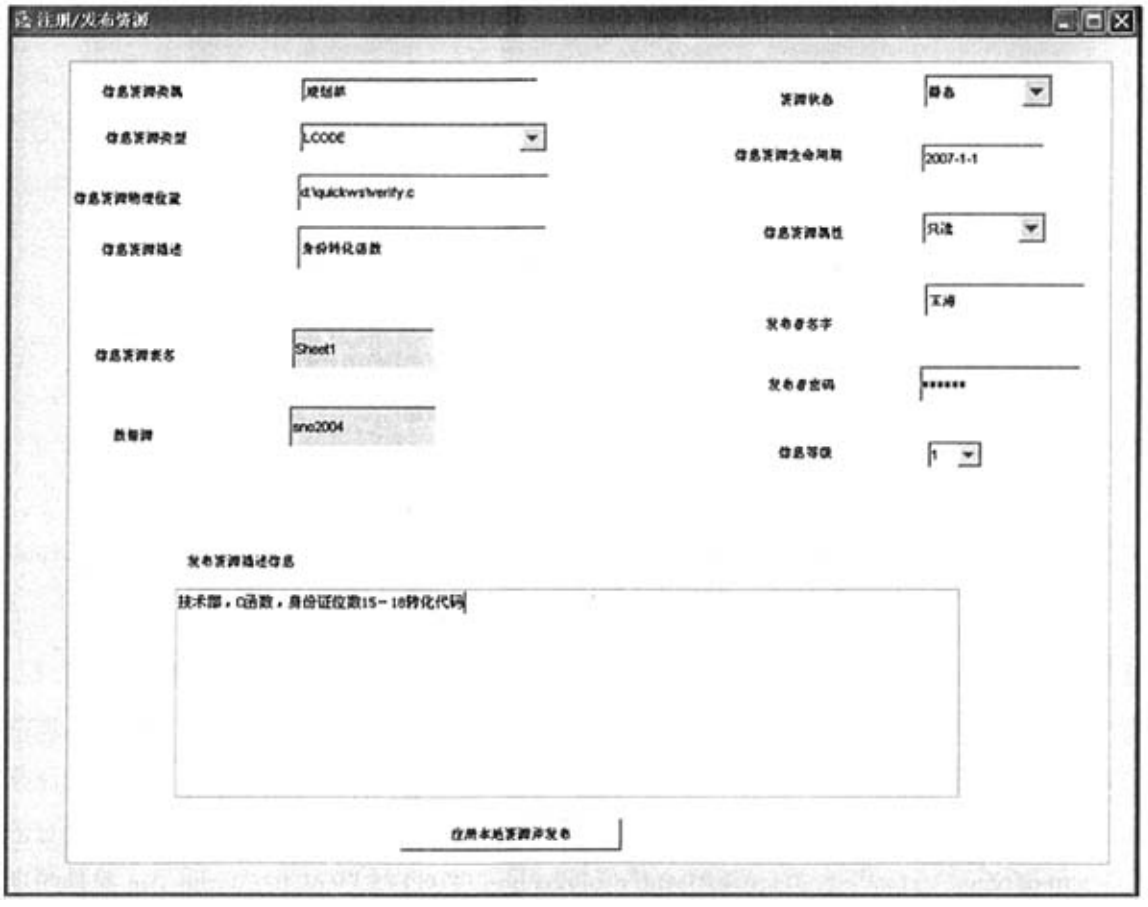

图 3. 17 遗留代码注册发布界面

触发“注册本地资源并发布”按钮后，应用程序自动调用本地注册模块，将包装好的遗留代码拷贝到以 sub 为前缀，再加上 resource_id 为名的目录下，进行遗留代码的本地注册和发布。

3.4.3 发现 fhfml 信息

信息消费者在全局注册中心使用基于关键字发现包装后的遗留裸代码 fhfml 代码。如下图所示。其中关键字“C 函数”和“身份证”是注册发布界面的“发布资源信息描述”框里的部分内容。

图 3.18 基于关键字的发现信息界面

图 3.19 是基于关键字的查询后发现的信息结果。查询结果为四条信息，在 Loglo 系统中信息是通过服务提供的。我们为遗留代码的信息查询提供两种查询方法，其一为显示整个包装后的 fhfml；其二为查询 fhfml 中的某一个函数，如可以查询发布包装后的遗留代码，如身份证位数转化代码中的一个函数 DoVerify 或者 main 函数。下图的访问点描述了获得一个信息的具体 url，http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_leg&rsource_id=5&table_id=1 定位到一条 fhfml 信息，即 C 代码 verify.c 包装后的 verify.xml。http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_func&rsource_id=5&table_id=1&par1=DoVerify 定位 C 代码 verify.c 中的包装后的函数 DoVerify；方法名 display_leg 表示调用此方法可以显示整个包装的遗留代码，方法名 display_func 表示调用此方法可以选择显示 fhfml 中的某一个函数；输入参数个数和输入参数表示访问方法的输入参数个数以及输入参数的类型；返回参数表示访问信息以后返回信息的类型；信息的描述为分为两种情况，其一，信息发布员在全局注册 fhfml 时所写的信息发现的参数；其二，遗留代码中每个用户自定义函数的功能描述信息。信息等级表示了哪些信息消费者可以查看信息，其中 1 表示所有消费者都可以查询到，其中 3 表示最高等级，只有授权消费者才可以查询到；所属部分表示此条信息隶属哪个部门。点击“单击此处进入浏览器查询”，进入基于浏览器的 fhfml 代码解析，同时将解析结果显示在浏览器中。


http://210.26.100.116:8080/infcenter/servlet/finservicenull - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
地址(D) http://210.26.100.116:8080/infcenter/servlet/finservicenull
转到
关于" 快速排序 all "的信息您已经成功找到。单击查找 继续查找信息，单击退出系统关闭窗口。
服务信息一览表
共 4 条
访问点: http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_leg&rsource_id=5&table_id=1
方法名: display_leg
输入参数个数: 2
输入参数: xsd:string xsd:string
返回参数: apachesoap:document
信息的调用方法: resource_id:5*table_id:1**main* DoVerify*Make15To18
信息的描述: 技术部，C 函数，身份证位数 15－18 转化代码
信息的等级: 1
所属部门: 技术部
单击此处进入浏览器浏览查询
访问点: http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_func&rsource_id=5&table_id=1&par1=DoVerify
方法名: display_func
输入参数个数: 3
输入参数: xsd:string xsd:string xsd:string
返回参数: apachesoap:document
信息的调用方法: resource_id:5*table_id:1**main* DoVerify*Make15To18
信息的描述: 技术部， C 函数功能:由给出的 17 位身份证号码数字计算出最后一位校验码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准),输入参数:参数 1 给出的 17 位身份证号码,返回值:最后一位校验码
信息的等级: 1
所属部门: 技术部
单击此处进入浏览器浏览查询
访问点: http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_func&rsource_id=5&table_id=1&par1=Make15To18
方法名: display_func
输入参数个数: 3
输入参数: xsd:string xsd:string xsd:string
返回参数: apachesoap:document
信息的调用方法: resource_id:5*table_id:1**main* DoVerify*Make15To18
信息的描述: 技术部， C 函数功能:给定的老式 15 位身份证号码升为对应的 18 位新式身份证号码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准),输入参数:参数 1 用于存放 15 位身份证号码,参数 2 用于存放计算出的 18 位新式身份号码,返回值:无
信息的等级: 1
所属部门: 技术部
单击此处进入浏览器浏览查询
访问点: http://210.26.97.253:8080/grid/services/fhfml&sermethod=diaplay_func&rsource_id=5&table_id=1&par1=main
方法名: display_func
输入参数个数: 3
输入参数: xsd:string xsd:string xsd:string
返回参数: apachesoap:document
信息的调用方法: resource_id:5*table_id:1**main* DoVerify*Make15To18
信息的描述: 技术部， C函数功能，主函数，调用子函数 Make15To18 实现 15 位身份证号码身份证号码 622801810315022 升为 18 位身份证号码,输入参数，无;返回值，无
信息的等级: 1
所属部门: 技术部
单击此处进入浏览器浏览查询
Internet

图 3.19 查询结果

3.4.3 访问 fhfml 信息

信息消费者发现身份证由 15 转化为 18 位 C 代码元信息以后，根据元信息访问此条信息。

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <display_leg soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <par1 xsi:type="xsd:string">5</par1>
      <par2 xsi:type="xsd:string">1</par2>
    </display_leg>
  </soapenv:Body>
</soapenv:Envelope>
```

其中，参数 5 和 1 分别代表信息种类号、信息号，访问方式遵循 Loglo 系统的访问模式，根据服务 URL、信息种类号（resource_id）、信息号（table_id）定位到一条信息。

响应 SOAP 消息如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <display_legResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <display_legReturn href="#id0"/>
    </display_legResponse>
    <multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="ns1:Document"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://xml.apache.org/xml-soap">
      <Functions>
        <Head>
          <![CDATA[#include "string.h"
            #define SRC_ID "622801810315022"
            ]]>
          </Head>
          <Function>
            <Description>
              <![CDATA[/*函数功能:由给出的17位身份证号码数字计算出最后一位校验码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准);输入参数:参数 1 给出的 17 位身份证号码;返回值:最后一位校验码*/]]></Description>
                <Function_head>
                  <Type_identifier>
                    <Return_type>int</Return_type>
                  </Type_identifier>
```

```
<Function_name>DoVerify</Function_name>
<Parameters parameter_num="1">
<Parameter>
<Parameter_Type>const char*</Parameter_Type>
<Parameter_Name>pszSrc</Parameter_Name>
</Parameter>
</Parameters>
</Function_head>
<Function_body>
<![CDATA[{
int iS = 0;
int iW[]={7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2};
static char szVerCode[]="10X98765432";
int i;
int iY;
for(i=0;i<17;i++)
{
iS += (int)(pszSrc[i]-'0') * iW[i];
}
iY = iS%11;
printf("%d %% 11 = iY = %d\n",iS, iY);
printf("%c \n",szVerCode[iY]);
return szVerCode[iY];}]>
</Function_body>
</Function>
<Function>
<Description>
<![CDATA[/*函数功能:给定的老式15位身份证号码升为对应的18位新式身份证号码(根据国家质量技术监督局实施的 GB11643-1999<公民身份证号码>标准);输入参数:参数 1 用于存放15位身份证号码,参数 2 用于存放计算出的18位新式身份号码;返回值:无*/]]></Description>
<Function_head>
<Type_identifier>
<Return_type>int</Return_type>
</Type_identifier>
<Function_name>Make15To18</Function_name>
<Parameters parameter_num="2">
<Parameter>
<Parameter_Type>char</Parameter_Type>
<Parameter_Name>*pSrc</Parameter_Name>
</Parameter>
<Parameter>
<Parameter_Type>char</Parameter_Type>
<Parameter_Name>*pDes</Parameter_Name>
</Parameter>
</Parameters>
```

```
                </Function_head>
                <Function_body>
                  <![CDATA[{
                    char szTemp[19] = "";
                    int n;
                    strncpy(szTemp, pSrc, 6);
                    strcat(szTemp, "19");
                    strcat(szTemp, &pSrc[6]);
                    n = DoVerify(szTemp);
                    szTemp[17] = n;
                    strcpy(pDes, szTemp);]]>
                  </Function_body>
                </Function>
                <Function>
                  <Description>
                    <![CDATA[/*函数功能：主函数，调用子函数 Make15To18 实现 15 位身份证号码身份证号码 622801810315022 升为 18 位身份证号码;输入参数：无;返回值：无*/]]>
                  </Description>
                  <Function_head>
                    <Type_identifier>
                      <Return_type>int</Return_type>
                    </Type_identifier>
                    <Function_name>main</Function_name>
                    <Parameters parameter_num="0"/>
                  </Function_head>
                  <Function_body>
                    <![CDATA[{
                      char szSrc[16] = SRC_ID;
                      char szDes[19] = "";
                      if(strlen(szSrc)   == 15)
                      {
                      printf("The 15-bit id: %s\n", szSrc);
                      Make15To18(szSrc, szDes);
                      printf("The 18-bit id: %s", szDes);
                      }]]>
                    </Function_body>
                  </Function>
                </Functions>
              </multiRef>
            </soapenv:Body>
          </soapenv:Envelope>
```

从服务返回的 SOAP 消息中，我们可以看出其中标签 multiRef 中包含的内容为网络中传输的 fhfml Document，即图中的黑体部分。客户端应用程序得到 fhfml Document 对象，通过基于浏览器的 fhfml 代码解析，在浏览器中可以看到遗留代码如图 3.20 所示。

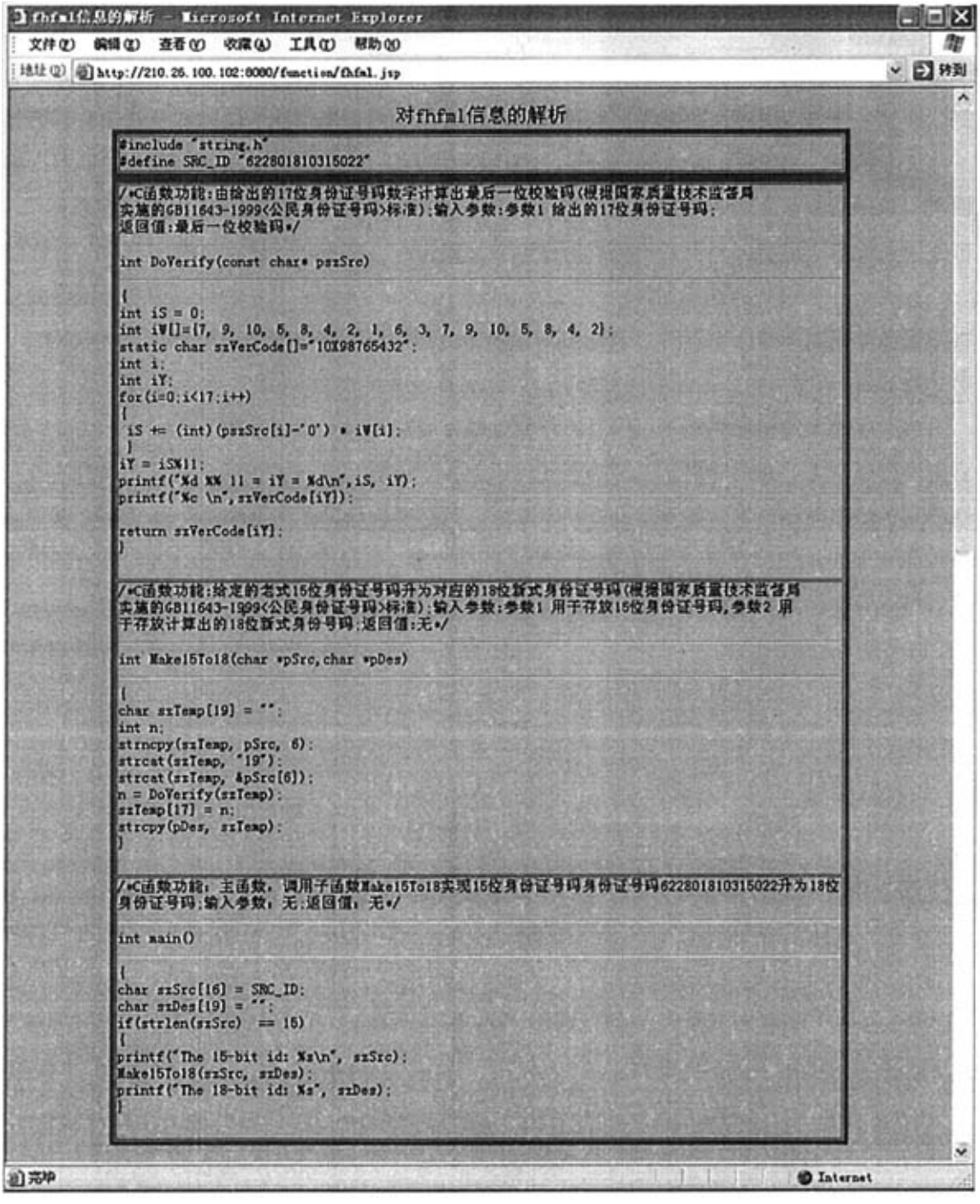

图 3.20 基于浏览器的 fhfml 解析结果图

3.5 总结

本章介绍了 Loglo 系统中遗留代码包装的框架的设计，遗留代码包装 LC-XML 方法规范的 schema、LC-XML 方法包装步骤以及流程，最后以一个身份证位数由 15 位转化位 18 位的

实例说明了遗留代码发布－发现的过程。

我们提出的包装遗留代码的方法 LC-XML，将遗留代码包装以后在本地进行注册，整个过程符合前期开发的基于两级注册表的企业信息网格 Loglo 系统体系结构，注册、发布、发现、访问信息的流程。

对遗留裸代码加标签进行包装，将其包装成有意义的数据即信息，成为企业中可以重用的财富，客户端获得包装后的代码，可以根据实际需要将代码做简单的解析，重新应用在自己的应用程序中。

LC-XML 方法包装遗留代码规范的确立依据主要是根据 C 代码中函数不能嵌套的原则，将整个 C 代码分为多个 Head 部分和 Function 部分，因此我们包装的时候，是以 Head 部分和 Function 部分为单位进行包装。LC-XML 方法包装遗留代码，不改变遗留代码的原始结构，只是在代码的某些位置增加标签，便于上层应用的处理和无二义性的传输，客户端根据接收到应答 SOAP 消息体中不同的信息包装格式，识别不同的信息类型，客户端的应用可以根据不同的包装格式（reml，ttpml，fhfml），建立不同的可扩展样式语言 XSL（eXtensible Stylesheet Language）显示 fhfml，用友好的图形用户界面显示给客户。或者对 fhfml 代码文件不进行任何处理作为一种中间表示保存本地以便下次使用。

第四章 遗留代码服务化

4.1 遗留代码服务化简介

我们的遗留代码服务化 LC-GS（Legacy Code-Grid Services）方法将遗留代码转化成网格用户可以访问的网格服务。使用 LC-GS 方法重用遗留代码总体框架包括，遗留代码的预处理、分析、通过 JNI 包装遗留代码生成动态链接库，最终由 legcode 服务调用由 JNI 技术生成 C 遗留代码动态链接库。

4.2 整体框架设计

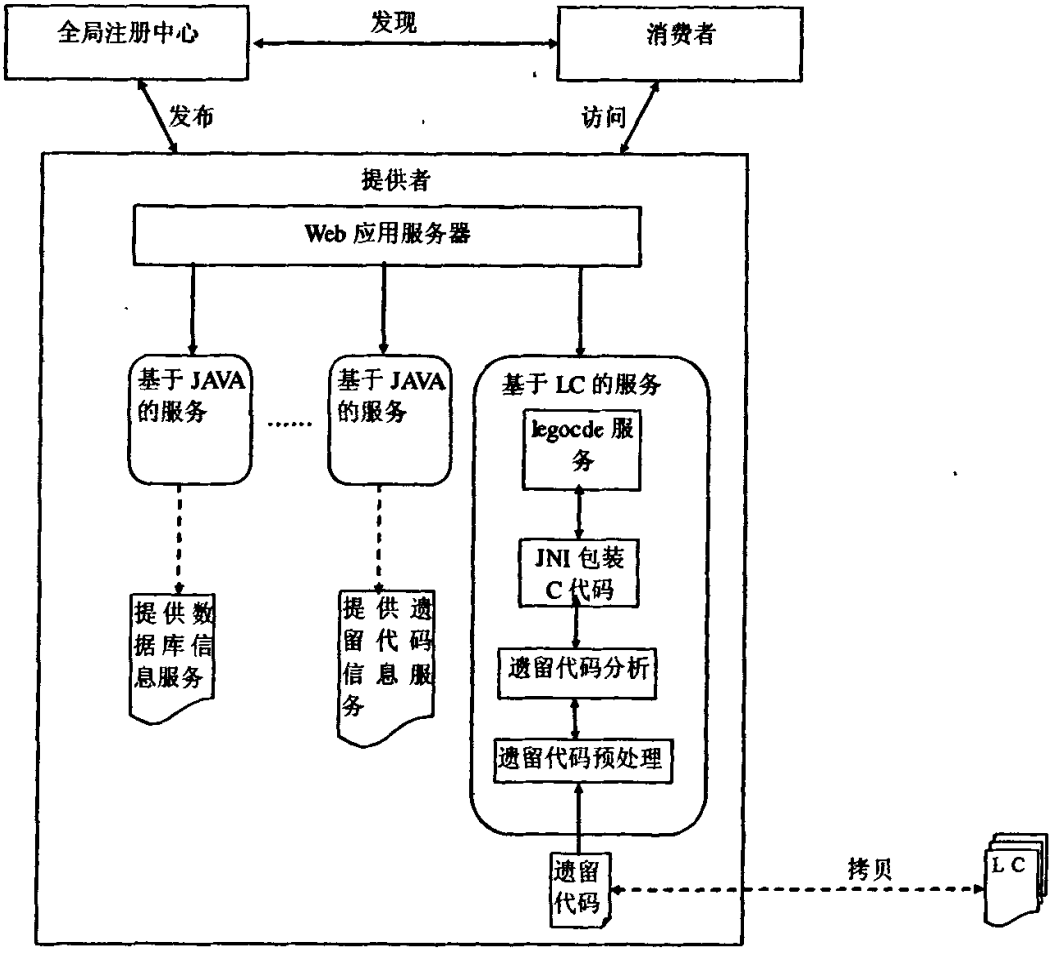

图 4.1 整体框架图

通过 LC-GS 方法实现遗留代码服务化，其关键使用 JNI 技术，生成 C 代码的动态链接库，使用 Web 服务调用动态链接库，从而使网格用户可以通过访问网格服务而实现遗留代码功能的使用。

在我们前期所构建的基于 Web 服务的企业信息网格 Loglo 系统中服务是由纯 JAVA 语言所实现，即基于 JAVA 的服务。在此基础上，通过 LC-GS 方法开发基于遗留代码的服务（基于 LC 的服务），其服务的真正功能由遗留代码提供。其整体框架如上图 4.1。

4.2.1 基于 JAVA 的服务

在 Loglo 系统中，提供数据库信息、文本信息、遗留代码信息的服务都是提前已经开发好的，即框架图中基于 JAVA 开发的服务。本地注册中心的信息发布员只需要在本地注册信息，而不需要注册提供信息的服务，在全局注册中心发布访问信息的服务接口以及元信息。基于 JAVA 服务模块为访问每一类信息提供访问接口，如 http://210.246.209.209:8080/grid/services/General_c 为访问数据库信息提供的服务，http://210.246.209.209:8080/grid/services/fhfml 为访问遗留代码信息提供的服务。这些服务都是使用纯 JAVA 语言开发的，当有信息请求到达的时候，由基于 JAVA 的服务模块中相关的类直接访问信息，最终将包装好的 reml 或者 fhfml 信息返回给信息消费者。

4.2.2 基于 LC 的服务

基于 LC 的服务模块其核心功能由遗留代码所提供，即服务是由面向过程的 C 代码所实现。此模块主要包括以下子模块，遗留代码预处理模块、遗留代码分析模块、JNI 包装 C 代码模块以及 legcode 服务模块。

（1）遗留代码预处理模块

此模块主要针对拷贝到本地的遗留代码，使用第三方工具，根据我们确定的切片准则，删除遗留裸代码中冗余语句，仅包含与指定功能所对应的那部分代码。

（2）遗留代码分析模块

我们提出的 LC-GS 方法，将遗留代码转化成服务，在 Web 服务的 WSDL 中定义了四个终端支持的交换原语，即，单向、请求－响应、恳求—响应、通知，我们只考虑请求－响应这种类型的交换原语。C 代码有一个主函数 main，程序的执行从 main 开始，在主函数中通过调用其它函数来完成功能，遗留代码分析模块主要通过人工对主函数中调用的函数进行简单的理解，最终得到能最大覆盖主函数功能的子函数集合或者通过对 main 函数修改，使其成为带有返回值的函数。

（3） JNI 包装 C 代码模块

此模块主要通过 JAVA 本地接口 JNI 实现 JAVA 对 C 代码的调用。下图为 JNI 原理图。其实质是通过 JAVA 调用由 C 代码生成动态链接库。由 JNI 生成 JAVA 代码对应 C 代码头文件，而 C 代码是对此头文件中申明的函数的实现。

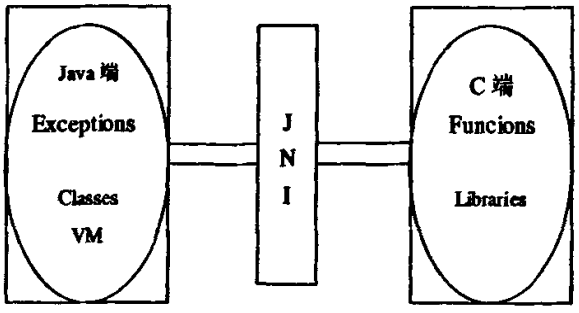

图 4.2 JNI 原理图

（4）legcode 服务模块

legcode 服务模块提供的服务是通过 JAVA 调用 C 的动态链接库而实现的。其真正的功能是由遗留代码 C 生成的动态链接库实现的。

通过上述分析，基于 LC 的服务中各模块主要功能如下表：

处理过程	遗留代码预处理	遗留代码分析	JNI 包装 C 代码	legcode 服务
目标	生成最简遗留裸代码。	得到功能独立的 C 函数。	生成动态链接库 DLL。	生成服务服务的主类。
说明	根据切片准则，删除遗留裸代码中冗余语句。	分析主函数 main，最终得到具有返回值的用户自定义函数。	通过 JNI 生成遗留代码的动态链接库。	在服务中增加方法，由该方法调用遗留代码生成的 DLL。

4.3 关键技术

4.3.1 JNI 技术

Java 本地接口 JNI（Java Native Interface）[41]属于 JDK 的一部分，已经被集成到标准 Java 平台之中，通过使用 JNI 编写程序，可以让代码方便地实现跨平台特性。JNI 允许运行在 Java 虚拟机 JVM(JavaVirtualMachine)上的 Java 代码操作其他语言编写的应用程序和库。使用 JNI 提供的方法，Java 代码能够直接与特定操作系统和硬件平台中的二进制库进行交互。这个交互过程发生在相同的 JVM 进程之中，使得来自 Java 类并被标志为“本地”的 Java 方法的调用能够被映射到共享二进制库中的相应函数上，并且将这两者加载到相同的进程空间。

JNI 框架使得本地方法使用 Java 对象的方式就和 Java 代码使用这些对象一样。一个本地方法能够创建 Java 对象，然后检查和使用这些对象，也可以检查和使用由 Java 应用程序代码创建的对象，甚至能够修改它创建的或传递给它的 Java 对象，并且这些修改过的对象对于 Java 应用程序也有效。因此，本地语言一方和 Java 应用程序一方都能创建、修改、访问 Java 对象，并在它们之间共享这些对象。本地方法也能够很容易地调用 Java 方法。使用 JNI 框架，本地方法能够调用现有的 Java 方法，传递该方法所需的参数，得到返回的结果。从以上描述中可以很容易地看到，JNI 起到 Java 和本地应用程序之间的"粘连"作用。JNI 规定了基本类型与本地类型的对应关系如图 4.3 所示。

Java 类型	本地类型	位数
boolean	jboolean	8，unsigned
byte	jbyte	8
char	jchar	16，unsigned
short	jshort	16
int	jint	32
long	jlong	64
Float	jfloat	32
double	jdouble	64
void	void	n/a

图 4.3 Java 基本类型与本地类型的对应关系

4.3.2 LC-GS 方法原理

LC-GS 方法将通过分析以后的遗留代码通过 JNI 进行包装，生成动态连接库 dll 文件，由服务主类调用此连接库，其工作原理如图 4.4。在 LC-GS 方法原理图中三个结点分别表示 Loglo 系统中的提供者结点（遗留代码转化为服务的结点）、全局注册中心结点以及消费者结点。其中提供者结点提供的服务分别由纯 Java 类实现以及由 Java 调用遗留 C 代码所生成的动态链接库 DLL 实现，

legcode 服务由 JAVA 类型实现，此服务中并没直接对动态链接库调用，而是由 Legcode 类调用一个中间类如 a.java。在 a.java 中真正实现了对动态链接库的调用。关键代码片段如下。

legcode 类如下：

```
public class legcode{
public String lge1(String par)
{ return a.aa(par) //调用类 a 中的动态链接库
}
```

```
……//其它方法
}
a.class 如下：
public static class   a{
    public satic String aa{
   static {
        System.loadLibrary("a");//加载动态链接库
        }
     public native static String m1(String par); //本地方法
     }
```

LC-GS 方法原理图中的动态链接库 a.dll 是通过 JAVA JNI 技术对遗留进行包装 a.c 以后而生成。

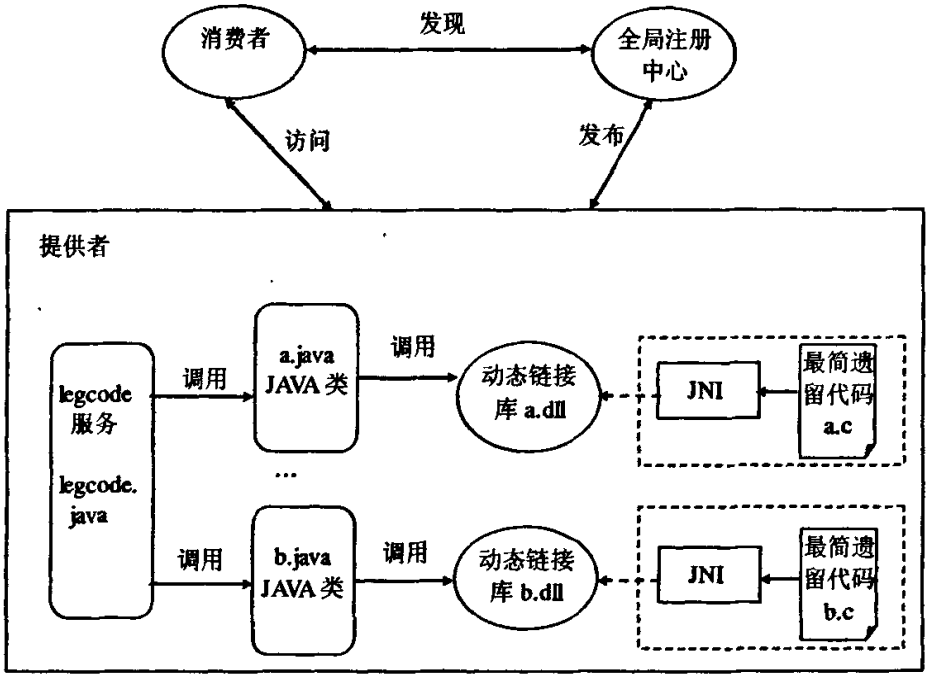

图 4.4 LC-GS 方法原理图

4.3.3 LC-GS 方法步骤

将遗留代码通过 JNI 技术转化成网格服务的 LC-GS 方法包括以下几个步骤：

（1）编写 legcode 类，部署成为网格服务；

（2）通过对遗留代码的简单分析，编写 Java 类，在这个类中包含了需要调用的本地方法；

（3）JNI 包装遗留代码；

①通过 JAVAC 命令编译步骤（2）中生成的 Java 类；

②通过 JAVAH 产生 C 代码的头文件，这个头文件中包括本地方法实现的函数原型声明；

③编写本地方法 C 实现(调用遗留代码 C 函数);

④编译本地 C 实现函数，并产生本地链接库;

（4）在 legacy 类中增加方法调用步骤（2）中产生的 Java 类。

将步骤（1）中编写 legcode 类，部署成为网格服务。legcode 类是网格用户调用服务的主类，初始状态下 legcode 类没有其他方法，只有当遗留代码使用 JNI 包装后以后才会增加相应的方法，legcode.java 代码参照前 4.3.2。

其部署文件 deploy-legcode.wsdd 如下。

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="legcode" provider="java:RPC">
  <parameter name="className" value="legcode"/>
  <parameter name="allowedMethods" value="*"/>
 </service>
</deployment>
```

其中 allowMethods 的 value 值为“*”，指出 legcode 类中的所有方法都部署成服务。当有新的方法增加时，避免多次部署的麻烦。

步骤（2）中通过分析遗留代码，编写 Java 类，在此类中包含了需要调用的本地方法的描述。本地方法 Java 代码的编写，首先分析遗留代码中函数的输入参数个数、类型、返回值的类型，其次编写可以产生 C 函数的头文件本地 Java 代码的方法，最后在 JAVA 代码中载入动态链接库。在编写的本地方法与一般的 JAVA 方法有所不同，其一，本地方法的声明必须包含标识符 native,以表示此方法是用另一种语言实现的。其二，本地方法声明结束符必须以分号结尾，因为本地方法所属类中没有此方法的实现。在本地方法调用之前，写在静态标识符里的实现本地方法的本地链接库先被加载。在类中，调用任何一个方法之前，Java 虚拟机自动运行初始化标识符 static，这样保证了在调用本地方法之前先调用本地链接库。System.loadLibrary 根据动态链接库的名字定位本地链接库，参前 4.3.2 中的 a.calss。。

分析留代码是编写本地方法代码的前提，在遗留代码预处理阶段，通过程序切片技术，对遗留代码进行简化，使其结构清晰，通过遗留代码的分析阶段已经得到遗留代码中 main 函数中的主要功能子函数。只要初级的了解遗留 C 代码就可以得到函数声明，根据函数声明编写 JAVA 本地方法代码。如图所示遗留代码与本地方法的映射关系。其中箭头所表示遗留代码类型与本地方法类型必须匹配。

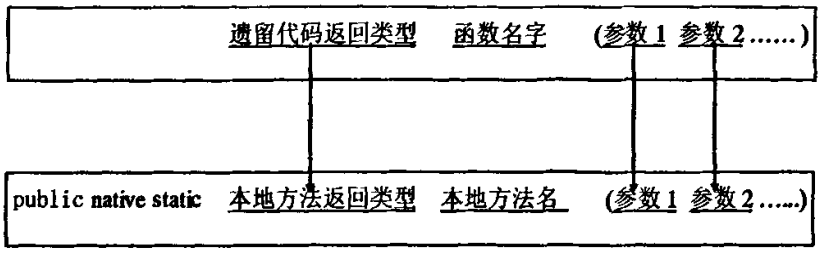

图 4.5 遗留代码与本地方法映射图

步骤（3）主要使用 JNI 技术规范，生成由 JAVA 类可以调用的动态链接库。

最后一步在 legcode 类中增加方法调用第一步中产生的 JAVA 类。如果还有其它的遗留代码可以通过步骤（2）、（3）生成动态链接库，最后在 legcode 类中增加新方法。

从客户访问遗留代码所提供服务的接口来看，遗留代码调用基本流程如图 4.6 所示。

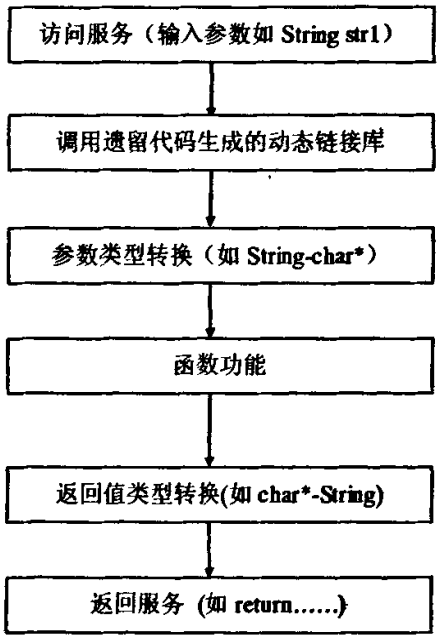

图 4.6 遗留代码调用基本流程图

4.4 LC-GS 应用举例

下面以一个人民币金额小写转化成大写的 C 语言代码为例，详细说明 Loglo 系统中，如何使用 LC-GS 方法完成遗留代码功能的重用。

4.4.1 遗留代码预处理和分析

如下遗留代码，选择切片准则所在语句“fprintf(fout,"%s = %s\n",strNum,str)”和变量 str。

人民币金额小写转化成大写的 C 语言代码 Getfee.c 如下：

```
/**
 * @note 转换根据：中国人民银行会计司编写的最新《企业、银行正确办理支付结算
 *       指南》的第 114 页-第 115 页
 */
#include "stdio.h"
#include "string.h"
#include "math.h"
char *zh();
#define YIWE   11 /*亿位*/
main()          /*将数字金额转换为大写金额程序*/
{               /*根据杨晓加的程序改写并注释*/
    char *str;
    FILE *fout;
    char strNum[14] = "123456789.145"; /*最大数为几亿，超过则不允许转换*/
    str= chineseFee (strNum);
    printf("%s = %s\n",strNum,str);
    fout = fopen("output.txt", "w");
    fprintf(fout,"%s = %s\n",strNum,str);
    fclose(fout);
    getch();
}
char * chineseFee (strNum)
char* strNum
{
    int i,n,bz;
    double x = atof(strNum);
    double xx=100*x,yy;
    char *c_je,je[14],temp[13];
    char f1[10][3]={"零","壹","贰","叁","肆","伍","陆","柒","捌","玖"};
    char f2[11][3]={"亿","仟","佰","拾","万","仟","佰","拾","元","角","分"};
    /*整数转换成字符，乘 100 是取到分，分后面第一位四舍五入*/
    yy=floor(1000*x)-(floor(xx)*10);
    xx=yy>4?ceil(xx):floor(xx);      /*如 0.145 将转换成 15*/
    sprintf(je,"%.0lf",xx);
    n=strlen(je);
    if(n>YIWE)
    {
        printf("对不起，数太大(%.3lf),整数位数(%d)太多（最大允许%d 位），不能处理！
\n",x,n-2,YIWE-2);
        exit(1);
    }
    c_je=(char *)calloc(80,1);
    bz=1;       /*正常标志*/
    for(i=0;i<n;i++)      /*对 n 位数字逐位处理*/
```

```
    {
        strcpy(temp,&je[i]);        /*temp 中存放从&je[i]开始的数字串*/
        if(atoi(temp)==0)           /*把数字符串变成整数，为 0 表示位全为'0'*/
        {
            bz=2;
            break;          /*跳出 i 循环*/
        }
        if(je[i]!='0')    /*判断当前位*/
        {
            if(bz==0)
                strcat(c_je,f1[0]);      /*写上"零"*/
            strcat(c_je,f1[je[i]-'0']);  /*写上"壹"~"玖"*/
            /*可用 Ctrl-F4 键操作，用类似'9'-'0'那样的估算表达式观察*/
            bz=1;
            strcat(c_je,f2[YIWE- n + i]);    /*写上"亿"~"分"*/
        }
        else    /*je[i]='0'时*/
        {       /*判断第 8 位即拾万位且后三位都不为 0 时写"万"字*/
         if((n-i==7)&&(je[i-1]!='0'||je[i-2]!='0'||je[i-3]!='0'))
            strcat(c_je,"万");
         if(n-i==3)    /*判断写"元"字*/
            strcat(c_je,"元");
         bz=0;          /*写"零"标志*/
        }
    } /*i 循环结束*/
    if(bz==2)/*特殊情况处理*/
    {
        if(n-i>=7&&n-i<10)    /*拾万位后全 0 时*/
           strcat(c_je,"万");
        if(n-i>=3)        /*拾元位后全 0 时*/
           strcat(c_je,"元");
           strcat(c_je,"正");      /*分位为 0 时*/
    }
    Strcat(c_je,"人民币");
    return(c_je);    /*返回指向大写金额字符串的指针*/
}
/*程序输出：123456789.145=壹亿贰仟叁佰肆拾伍万陆仟柒佰捌拾玖元壹角伍分*/
```

清单 5.1 人民币金额小写转化成大写的 C 语言代码 Getfee.c

这是一个比较典型的 C 语言代码，其中使用了 C 语言的基本数据类型。经过遗留代码预处理后得到的代码为上图中黑体代码以及 main 函数中部分语句。遗留代码分析以后得到的代码如上图黑体所示。

4.4.2 JNI 包装遗留代码

简单分析 Getfee.c 中的函数 chineseFee，其函数原型为：char* chineseFee(char* src)，其中输入为一个字符串类型的参数，输出为字符串类型。编写本地 Java 代码 RMB.java 如下：

```
public class RMB {
    static {
        System.loadLibrary("chineseFee");
    }
    public native static String chineseFee(String rmb);
}
```

其中 chineseFee 为动态连接库名，其中本地方法与遗留代码函数 chineseFee 对应关系如图 4.8。

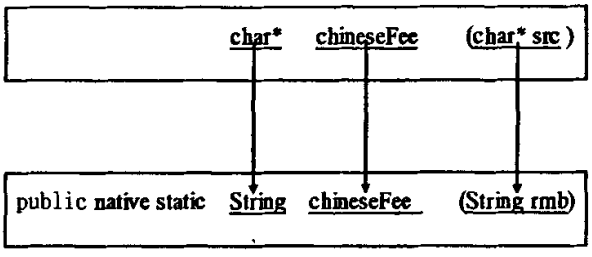

图 4.7 本地方法与函数 chineseFee 对应图

定义本地链接库后，编译源代码，生成相应的类文件 RMB.class。用 Javah 产生 JNI-style 头文件帮助调用遗留代码 C 函数。根据类 RMB、方法生成的头文件 RMB.h 内容如下：

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class RMB */
#ifndef _Included_RMB
#define _Included_RMB
#ifdef __cplusplus
extern "C" {
#endif
/* Class:     RMB
 * Method:    chineseFee
 * Signature: (Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_RMB_chineseFee (JNIEnv *, jclass, jstring);
#ifdef __cplusplus
}
#endif
#endif
```

其中头文件 jni.h 提供遗留代码所要调用 JNI 函数。 (Ljava/lang/String;)这是函数的标记

符，当从本地方法端访问 Java 端的方法时，会用到这个标记符。RMB.java 中的本地方法 public native static String chineseFee(String rmb) 被影射为 JNIEXPORT jstring JNICALL Java_RMB_chineseFee (JNIEnv *, jclass, jstring)。每一个方法映射到本地 C 函数后都会增加两个参数：JNIEnv *和 jclass。第一个参数 JNIEnv 是一个指针，指向在 jni.h 中定义的一个数据结构，结构中包含了一系列的函数的指针，称之为 JNI 函数。使用这些 JNI 函数可以从本地函数这一侧完成对 Java 的操作，如访问 Java 字符串、数组、调用 Java 的方法、成员变量、甚至处理 Java 端的异常等。第二个参数会根据 Java 类中本地方法的定义不同而不同，如果是定义为 static 方法，类型会是 jclass，表示对特定 Class 对象的引用，如果是非 static 方法，类型是 jobject，表示当前对象的引用，相当于 this。C 函数通过 JNI 接受来自 Java 的传参也是按基本类型直接传值，对象传引用。对于基本类型可以按照图 4.1 来使用，对象类型都必须使用 JNI 函数进行转换后才能在 C 函数中使用。在本例中遗留代码的参数为字符串 String 对象类型，必须通过 JNI 函数进行转换。

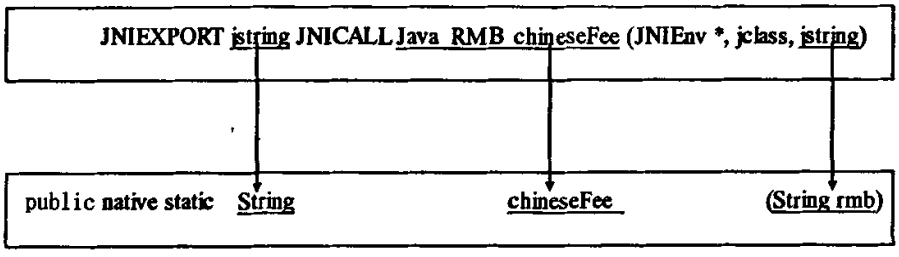

图 4.8 本地方法与 JNI 类型对应图

编写 RMB.H 头文件中函数 Java_RMB_chineseFee 的 c 实现函数 legCcode.c，在此代码中实现 Java 参数类型到 C 参数类型的转换，并调用遗留代码 Getfee.c。legCcode.c 如下：

```
#include "RMB.h"
#include "Getfee.c"
#include "WindowsTojstring.c"
JNIEXPORT jstring JNICALL Java_RMB_chineseFee(JNIEnv * env, jclass jcls ,jstring str1)
{   char * result;
    char tmp_str1[128];
    int len=(*env)->GetStringLength(env,str1);
    (*env)->GetStringUTFRegion(env,str1,0,len,tmp_str1);/
    result=chineseFee(tmp_str1);   //调用遗留代码 Getfee.c
    return WindowsTojstring(env,result);}
```

其中 GetStringLength 函数返回字符串长度，GetStringUTFRegion 函数以 UTF-8 编码拷贝字符串到字符缓冲区，WindowsTojstring.c 为 Java 和 C 之间通过 JNI 传递含有中文字符串的参数,解决中文乱码问题。遗留代码调用基本流程如图 4.7 所示。最后生成本地链接库 rmb。

4.4.3 legcode 服务中增加遗留代码功能

根据 RMB.java 中的本地方法 public native static String chineseFee(String rmb)，在 legcode 类中增加方法调用 chineseFee：

```
public   String chineseFee(String url,String rmb)
    {
        return RMB.chineseFee(rmb);
    }
```

编译后 chineseFee 成为遗留代码 Getfee.c 所提供的服务，在浏览器中可以看到部署后的服务。

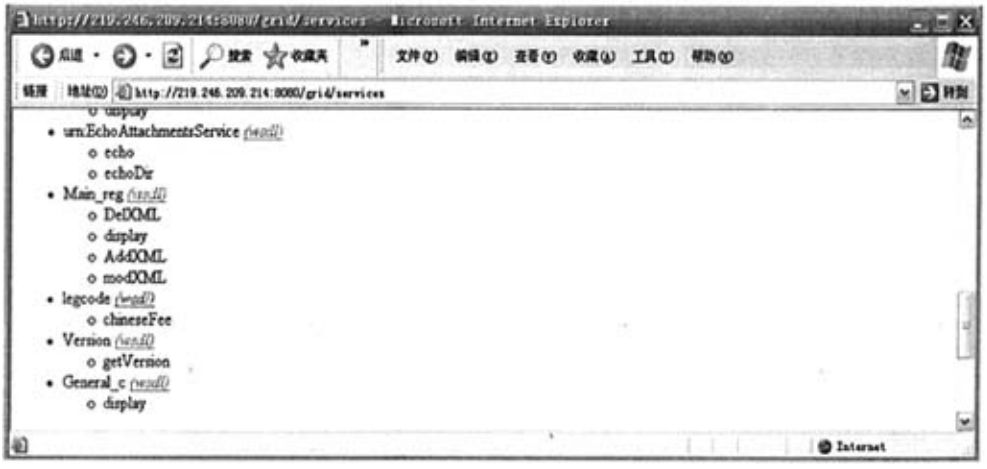

图 4.9 遗留代码部署为网格服务

wsdl 文件为图 4.11 所示。根据 WSDL 机制的基本描述，分析图 4.11 中 legcode 服务的 WSDL 文档，语法中使用了 5 个元素对服务进行定义： message（消息）包含一个 part 元素，消息的 name 提供名字，此名字封装在 WSDL 文档中的所有消息中是唯一的。part 中 name 属性提供的名字在封装消息的所有 part 中是唯一的。portType（端口类型）中定义输入和输出消息，端口类型的 name 属性提供一个名字，此名字封装在 WSDL 文档中所有端口类型名中是唯一的，通过 name 属性名进行操作。WSDL 定义了四个终端支持的交换原语，即，单向、请求一响应、恳求—响应、通知，上图 legcode 服务使用请求一响应操作。binding（绑定）为由特殊端口类型定义的操作和消息定义消息格式和协议细节，name 属性提供名字，此名字封装在 WSDL 文档中是唯一的，绑定使用 type 属性引用了它所绑定的端口类型 legcode。消息 port（端口）指出 legcode 服务用于绑定的地址，定义了单个通讯终端。service（服务）集成相关端口。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://219.246.209.214:8080/grid/services/legcode"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://219.246.209.214:8080/grid/services/legcode"
    xmlns:intf="http://219.246.209.214:8080/grid/services/legcode"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  + <!--  -->
  - <wsdl:message name="chineseFeeResponse">
      <wsdl:part name="chineseFeeReturn" type="xsd:string" />
    </wsdl:message>
  - <wsdl:message name="chineseFeeRequest">
      <wsdl:part name="rmb" type="xsd:string" />
    </wsdl:message>
  - <wsdl:portType name="legcode">
    - <wsdl:operation name="chineseFee" parameterOrder="rmb">
        <wsdl:input message="impl:chineseFeeRequest" name="chineseFeeRequest" />
        <wsdl:output message="impl:chineseFeeResponse" name="chineseFeeResponse" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="legcodeSoapBinding" type="impl:legcode">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="chineseFee">
        <wsdlsoap:operation soapAction="" />
      - <wsdl:input name="chineseFeeRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace" use="encoded" />
        </wsdl:input>
      - <wsdl:output name="chineseFeeResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://219.246.209.214:8080/grid/services/legcode" use="encoded" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="legcodeService">
    - <wsdl:port binding="impl:legcodeSoapBinding" name="legcode">
        <wsdlsoap:address location="http://219.246.209.214:8080/grid/services/legcode" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

图 4. 10 遗留代码服务 wsdl 文档

4.4.4 legcode 服务的测试

部署成功服务以后，Loglo 客户端访问遗留代码 Getfee.c 所提供的服务,客户输入小写人民币金额“675403.980”后，服务返回大写人民币金额“人民币陆拾柒万伍仟肆佰零叁圆玖角捌分”。通过 SOAP Monitor 捕捉 SOAP 消息如下：

请求 SOAP 消息：

```
<?xml version="1.0" encoding="utf-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
  <chineseFee soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <par2 xsi:type="xsd:string">675403.980</par2>
  </chineseFee>
  </soapenv:Body>
  </soapenv:Envelope>
```

响应 SOAP 消息：

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
  <chineseFeeResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <chineseFeeReturn xsi:type="xsd:string">人民币金额陆拾柒万伍仟肆佰零叁圆玖角捌分
  </chineseFeeReturn>
  </chineseFeeResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

4.5 总结

本章介绍了将遗留代码转化成网格用户可以访问的网格服务的框架设计以及方法LC-GS，介绍了该方法的原理和实现步骤。

此方法将面向集中式系统的、无法集成到现在的平台中的遗留代码成功的集成到基于Web服务的企业信息网格Loglo系统之中。在开始部署好的服务legcode中，增加由Java调用遗留代码生成动态链接库方法，可以避免多次部署的麻烦。使用LC-GS方法转化遗留代码成为网格服务，本地注册者只需要对C语言有初级的了解即可，即可以看出C函数的申明，输入参数类型、输入参数个数以及返回值。

但是我们研究的LC-GS方法只是适合有返回值的C函数代码，对于遗留代码没有返回值的C函数只能通过LC-XML方法包装后返回给Loglo系统的客户端。同时，LC-GS的方法现阶段使用的是手工完成，并没有实现自动化的过程，这是我们今后要努力的方向。

第五章 结束语

5.1 论文的主要工作

本文是以我们前期开发的基于 Web Service 的企业信息网格 Loglo 系统为基础，并结合时下流行的 XML 和 JAVA 技术、解决了遗留代码在企业信息网格中重用的问题。文中对遗留代码在企业信息网格中的重用研究做了较为全面的分析和介绍，提出了遗留代码在企业信息网格中重用的两种方法。

论文的主要内容：

1、企业信息网格 Loglo 系统的框架设计

分别从 Loglo 系统的整体框架设计、数据库信息和文本信息的包装规范方面进行了详细的介绍，最后以一个实例介绍了一个典型的信息发布－发现的过程。

2、遗留代码的包装

本文提出了一种遗留代码的包装 LC-XML 方法，主要介绍了使用 LC-XML 方法包装遗留代码的整体框架设计，以及关键技术即包装规范的 schema、包装方法的步骤以及流程。最后以一个身份证位数由 15 位转化为 18 位的例子介绍了 LC-XML 包装遗留代码中的具体方法以及实验结果。

3、遗留代码服务化

本文提出了一种遗留代码的服务化的 LC-GS 方法，主要介绍了使用 LC-GS 方法将遗留代码转化成网格服务的整体框架设计，以及关键技术即转化方法 LC-GS 的原理以及流程。最后一个企业中常见的人民币金额大小写转化的例子介绍了 LC-GS 实现中的具体方法以及实验结果。

5.2 后继研究工作的展望

本文研究的内容已经在我们的 Loglo 系统上实现原型，有待于得到广泛应用。整个研究对象为遗留 C 代码，还不能应用到其他面向过程的语言，而且目前 LC-GS 方法部分是手工完成的，需要本地注册者对 C 代码有初级的了解。

下一步需要做的是进一步研究遗留代码转化成网格服务 LC-GS 方法的自动化过程，以及如何对其他面过程的语言如 COBOL、FORTRAIN 等使用 LC-XML 方法和 LC-GS 方法。

参考文献

[1] Lientz，B.P.，Swanson，et al. Software Maintenance Management[J]. Addison-Wesley，1980.

[2] 徐宝峰. C-Java 自动程序转换系统原型的设计和实现. 上海：上海师范大学硕士学位论文，2006.

[3] William M.Ulrich. Legacy Systems: Transformation Strategies[M]. Publisher: Prentice Hall Ptr，2002(6):15.

[4] Myers，Beatrice. Introductory grammar: Lecture and lab materials. Huntington Beach, CA: Summer Institute of Linguistics，1975.

[5] N.Wirth，Algorithms + Data Structures = Programs[M]. Publisher ： Prentice Hall，1976.

[6] E.Yourdon，L.Constantine. Structured Design: Fundamentals of a Discipline of Computer Programming and Design[M]. Publisher: Prentice Hall，1979.

[7] Robert Arnold，Shawn Bohner. Software Change Impact Analysis[M]. Publisher: Wiley-IEEE Computer Society Pr 1st edition ，1996.

[8] H.M.Sneed. Encapsulation Legacy Software for use in Client/Server. Proceeding of WCRE'96 1996.

[9] A.De Lucia，G.A. Di Lucca，A.R. Fasolino，et al. Migrating Legacy System towards Object-oriented Platform[C]. 13th International Conference on Software Maintenance (ICSM'97), 1997.

[10] Nelson Weiderman，Linda Northrop，Dennis Smith，et al. Implications of Distributed Object Technology for Reengineering[R]. Tehnical Report CMU/SEI-97-TR-005 ESC-TR-97-005，1997.

[11] Tilley，R.Scott，Smith，et al. Perspectives on Legacy Systems Reengineering[OL]. http://www.sei.cmu.edu/~reengineering/pubs/lsysree.

[12] Dr. Linda，H. Rosenberg. Software Re-engineering[J]，Software Assurance Technology Center，1995.

[13] Elizabeth Burd，Malcolm Munro. A Method for identification of reusable units through the reengineering of legacy code[J]. The Journal of System and Software，1998.

[14] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia，et al. Decomposing legacy programs:a

first step towards migrating to client-server platform[J]. The Journal of Systems and Software, 2000.

[15] Chia-chu Chiang. Wrapping legacy system for use in heterogeneous computing environment[J]. Information and Software Technology , 2001.

[16] Y.Bi, M.E.C.Hull, P.N.Nicholl. An XML approach for legacy code reuse[J]. The Journal of Systems and Software , 2002.

[17] Wolfgang Emmerich, Cecilia Mascolo, Anthony Finkelstein. Implementing Incremental Code Migration with XML[J]. 22nd International Conference on Software Engineering (ICSE '00), 1999.

[18] Thierry Bodhuin, Enrico Guardabascio, Maria Tortorella. Migration of non-decompo-sable software systems to the Web using screen proxies[J]. Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03), 2003.

[19] Yan Huang, Ian Taylor, David, et al. Wrapping Legacy Codes for Grid-Based Applications[OL]. http://www.geo600.uni-hanoverde.

[20] Thierry Delaittre, Tamas Kiss, Ariel Goyeneche, et al. GEMLCA: Running Legacy code Application as Grid Services[R]. CoreGRID Technical Report Number TR-0004, 2005.

[21] 杨新刚，李晓林，唐宁九. 织女星企业信息网格资源接口模块设计与实现的研究[J].计算机应用研究，2003 (10)：36－38.

[22] 周险峰，唐宁九，段武明，等. 织女星企业信息网格资源目录服务的研究[J]. 计算机应用研究，2003 (11)：134－136.

[23] 徐志伟. 织女星信息网格的体系结构研究[J]. 计算机研究与发展，2002(8)：948－951.

[24] 桂小林. 基于Internet的信息网格的软件框架研究[J]. 西安交通大学学报，2004(6)：551-554.

[25] 段武明. 企业信息网格资源共享的研究[D]. 四川：四川大学硕士学位论文，2003.

[26] 刘瑞，陈微，闫继忠等.MICROSOFT SQL 2000 宝典[M].北京：中国铁道出版社.

[27] hen Zhen Qiang, Xu Bao Wen. An approach to analyzing dependency of concurrent programs[J]. IEEE APAQS'2000, Hong Kong, 2000.

[28] Chen Zhen-Qiang, Xu Bao-Wen. Dependency analysis based dynamic slicing for debugging[J]. Wuhan University Journal of Natural Sciences, 2001(6):398-404.

[29] S.Horwitz. Interprocedural slicing using depedence graphs[J]. ACM Trans Programming Languages and Systems, 1990(12):35-46.

[30] Chen Zhen-Qiang, Xu Bao-Wen. Slicing concurrent Java programs[J]. ACM SIGPLAN

Notices，2001(4):41-47.

[31] Chen Zhen-Qiang，Xu Bao-Wen. Slicing object-oriented Java programs[J]. ACM SIGPLAN Notices，2001 (4):33-40.

[32] Chen Zhen-Qiang，Xu Bao-Wen，Yang Hong-Ji. Slicing tagged objects[J]. in A da 95. In：LNCS 2043 Springer-Verlag，2001.

[33] Xu Bao-Wen，Chen Zhen-Qiang，Zhou Xiao-Yu. Slicing object-oriented[J]. A da 95 programs based on dependencd analysis Journal of Software，2001(12):208-213.

[34] 李必信. 程序切片技术及其应用[M]. 北京：科学出版社，2006：3－55.

[35] Jim Lyle , Dolores R. Wallace , David W. Binkley. The Unravel Program Slicing Tool[OL], http://www.itl.nist.gov/div897/sqg/unravel/unravel.html.

[36] Sergey Poznyakoff. GNU cflow[OL]. http://www.gnu.org/software/cflow.

[37] W3C. http://www.w3.org/TR/WD-xml[OL].

[38] 朱茂华，Turobo C 2.0 高级编程与分析[M]. 四川：成都科技大学出版社，1999.

[30] Tim Bray, Textuality , et al. Extensible Markup Language (XML) 1.0 (Second Edition)[OL]. http://www.w3.org/TR/2000/REC-xml-20001006.

[40] D.Fallside，P.Walmsley. XML Schema Part 0: Primer Second Edition. Oct，2004[OL]. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028.

[41] Sheng Liang，The Java™ Native Interface Programmer's Guide and Specification[S]. Sun Microsystems，1999.

致 谢

首先深深感谢我的导师冯百明教授。本文的完成自始至终得到导师的悉心指导。导师渊博的知识、严谨的治学态度和对国内外学术动态的敏锐的洞察力，使我受益匪浅。导师在生活中的关怀，学习中的鼓励和鞭策，使我得以顺利地完成学业。因此，在这三年艰辛而又充实的研究生岁月里，我所取得的每一份成绩、每一点进步都离不开导师无微不至的教诲和关怀，在此对导师致以我最诚挚的感谢，您的教导将令我终身难忘。

感谢王治和教授、党小超教授、王彩芬教授、赵学峰副教授、王小牛副教授等老师给予的关心和帮助。

感谢马元元、赵媛、苟和平、蒲兴彦、于成尊等同学在我毕业论文撰写和审校中的热心帮助，正是在大家共同努力下才使本文得以顺利完成。

感谢好朋友石世群给我提供的技术指导，为我顺利完成论文撰写提供了很大帮助。

感谢评委老师们的评阅和答辩老师们的指正，你们的严谨治学态度不仅能助我提高论文的质量，同时也会对我今后的学习和生活产生重要影响。

还要感谢我的父母和家人，他们对我生活上的关心让我有了足够的精力和时间完成三年的研究生学习生活和最后论文的撰写。

三年寒窗生活即将结束，而同窗好友们与我兄弟姐妹般的情意将永留心中，终生难忘，他们是我永远值得留恋的学友，仅以此文与他们共勉。向所有关心、帮助过我的人们表示衷心的感谢！

攻读硕士学位期间学术成果

发表论文：

1 邹燕飞，马元元. 用 Apache Axis1.2 构建 web 服务.电脑知识与技术，2006，2。
2 邹燕飞，马元元. 多银行电子现金系统方案研究.福建电脑，2006，6。

参与项目：

1 甘肃省科技攻关项目(2GS047-A52-002-04)：企业信息网格关键技术研究。

2 西北师范大学大学生科研资助金项目：高校信息网格关键技术研究。