基于代码生成的 Web 信息系统工程化开发方法

张立勇　陈　平
（西安电子科技大学软件工程研究所　西安 710071）

摘　要　本文提出了一种基于代码生成的 Web 信息系统工程化开发方法，以用于以数据库为核心的 Web 信息系统的自动生成。采用类似编译器基础架构的特定域软件体系结构，实现了前端与后端的共享；提出了一种用于代码生成的专用代码方法，使得生成的代码便于二次定制与维护；提出了多表关联、基本操作组合以及数据库驱动的目标平台无关工作流机制等方法，支持复杂模式业务逻辑的自动生成。

关键词　代码生成，Web 信息系统，特定域软件体系结构，业务模式

Web Information System Construction Based on Code Generation

ZHANG Li-yong　CHEN Ping
（Software Engineering Institute, Xidian University, Xi'an 710071, China）

Abstract　An approach based on code generation is proposed for automatic construction of database-backed web information system. A domain specific software architecture similar to compiler infrastructure is used, which enables reusability of forends and backends. A private-code method for code generation is proposed to facilitate customization and maintanance of the generated code. Such methods as muti-tables ralating, basic operations composition and workflow mechanism are discussed for complex-patterns of business logic.

Keywords　Code generation, Web information system, Domain specific software architecture, Business patterns

1　引言

随着互联网技术与信息化进程的发展，越来越多的企业把信息系统基于 Web 方式实现，其中以数据库为核心的(database-backed)的系统占了相当大的比例。

这些以数据库为核心的信息系统尽管因其具体的业务功能不同而有所区别，但从软件工程的角度看，其开发过程是大致相同的。目前，在开发这类系统时，软件开发人员不断重复着一些不必要且耗时的技术性劳动；另一方面，这类系统用户需求变化很快，如何缩短开发周期并应对用户对系统需求的快速变化，成为前期研发和后期维护中最大的难题。

人们为了简化 Web 信息系统的开发，先后提出了一些开发框架，如 Struts[1]，Hibernate[2]，Ruby on Rails[3] 等，这些开发框架可以有效提高系统的开发效率，但是由于其未充分考虑实际系统的业务逻辑，因此要求开发人员至少进行基于某种脚本语言的开发。当系统的业务功能庞杂且用户需求变化很快时，系统的开发周期与所开发代码的质量仍很难满足实际需要。

本文针对数据库为核心的 Web 信息系统快速开发及维护需求，提出了一种基于代码生成的 Web 信息系统工程化开发方法，重点研究了用于自动生成的特定域软件体系结构、用于代码生成的专用代码方法以及复杂模式业务逻辑的自动生成方法等。

2　自动生成的基础和用于自动生成的特定域软件体系结构

在以数据库为核心的 Web 信息系统这一特定软件开发领域，开发技术的发展以及此类系统自身的特点，使得类似软件系统的自动生成成为可能。

首先当前主流软件开发技术的发展为此类系统的自动生成提供了必要条件：

(1)丰富的开发构件与基于源程序文本的构件使用方式。Java,. NET 等当前主流 Web 信息系统开发平台均提供了包括界面构件、数据验证构件、数据库访问构件等等在内的丰富开发构件，为快速开发以数据库为核心的 Web 信息系统提供了支持。而基于这些构件开发的程序单元(包括用户界面单元 JSP/ASP 和逻辑处理单元 JavaBean/C# 类等)均可以以源程序文本的方式来使用构件，即完全可以在脱离集成开发环境的情况下以文本方式编写出对等的程序单元。

(2)参数化的数据库访问接口。传统的 C/C++中访问数据库的程序单元一般利用集成开发环境生成。随着开发技术的发展，目前采用的主流语言，如 Java，C#，C++等，均提供了更为方便的参数化数据库访问接口，包括数据库连接串、执行的 SQL 语句、访问的数据库表结构(如字段名和字段类型)等均以参数的形式出现在程序单元的源程序文本中，仅通过编写源程序文本即可实现普通关系型数据库的访问。

上述两个条件使得程序单元的自动生成成为可能，仅通过生成其源程序文本即可得到完整的程序单元。

其次，在以数据库为核心的 Web 信息系统中，相当一部分业务属于录入/编辑、删除、查询、统计或基于上述基本功能的组合业务模式。同一业务模式的不同业务，其实现的程序单元结构完全相同，区别仅在于业务逻辑特定的信息，如录入的字段不同、查询条件及查询结果列出的字段不同等等。因

张立勇　博士研究生，讲师，主要研究领域为程序分析、分布处理等；陈　平　博士，教授，博士生导师，主要研究领域为电子信息系统软件开发理论语技术、面向对象技术等。

此，抽取常见的业务模式，并根据具体业务需求在同样结构的程序单元中填入业务逻辑的特定信息，即可得到实现既定业务模式功能的程序单元，从而实现类似业务逻辑软件/模块的自动化生成，提高此类系统开发的工程化与自动化程度。

为便于论述，引入如下定义。

定义 1（自动生成平台） 用于自动生成以数据库为核心的 Web 信息系统的平台。

定义 2（目标系统） 目标指自动生成平台生成的对象，目标系统即自动生成平台生成的 Web 信息系统。目标系统的实现平台（如 Java、.NET）称为目标平台。

近几年，人们在以数据库为核心的 Web 信息系统自动生成方面进行了有益的尝试，提出并实现了一些自动生成平台[4,5]，可以实现部分 Web 信息系统的自动生成，但是总结起来存在以下缺点：

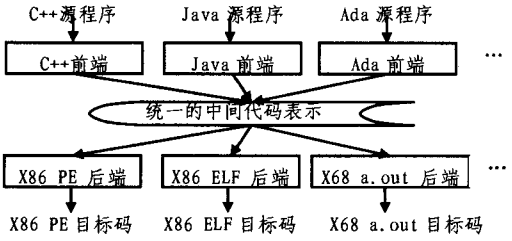
（1）目标平台单一。当前 Web 信息系统开发可采用 Java、.NET 等多种主流开发平台实现，现有自动生成平台均以其中之一作为最终生成系统的目标平台，自动生成平台核心功能的可重用性较差。而系统的业务逻辑本身是与具体实现平台无关的，同样的业务逻辑完全可以根据后期定制与维护的需要生成 Java、.NET 等不同的实现。

（2）核心功能简单。大多自动生成平台只是完成了对数据库的基本操作，如录入/编辑、删除、查询等操作，而且只针对单一表结构，不能适应实际系统的复杂业务需求。

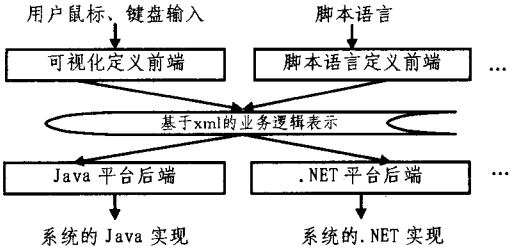
（3）流程控制不健全。Web 信息系统中操作的核心是数据，数据的处理流程也就是业务流程，在实际的信息系统中越来越重要，而现有自动生成平台业务流程支持不充分或局限于某目标平台的具体工作流机制，不能满足多目标实现平台的需要。

为了提高自动生成平台核心功能的可重用性，支持多目标平台，提出一种图 1 所示的类似现代编译器基础架构（compiler infrastructure）的特定域软件体系结构（Domain Specific Software Architecture，DSSA），用于构建自动生成平台。

图 1(a)所示的编译器基础架构通过将编译工作划分为前端与后端两个阶段，并在前端与后端之间提供统一的中间代码格式，实现多个后端共享同一前端、多个前端共享同一后端的高可重用性。类似地，将 Web 信息系统的自动生成划分为业务逻辑定义（类似于编译器前端）与目标代码生成（类似于编译器后端）两个主要阶段。业务逻辑可采用可视化、脚本语言等方式定义，定义生成统一格式的业务逻辑表示，供代码生成使用；目标代码生成阶段读入业务逻辑定义，根据后期维护需要生成不同目标平台的代码。在该自动生成平台上，一次定义的业务逻辑可以生成 Java、.NET 等不同平台的实现，而同一目标平台的业务逻辑也可以用不同的方式进行定义，从而实现自动生成平台核心功能的高度重用。



(a) 编译器基础架构



(b) 用于 Web 信息系统生成的特定域软件体系结构

图 1 类似编译器基础架构的特定域软件体系结构

基于以上“前端+后端”体系结构，抽取常见业务逻辑模式，如图 2 所示，针对每一种业务逻辑模式开发用于业务逻辑定义的前端与用于代码生成的后端，从而实现相应业务逻辑软件的自动生成。业务逻辑定义是待生成目标系统需求分析结果的精确表示，直接根据需求分析结果生成目标软件，跳过了一般软件系统开发的设计与编码阶段，同时自动生成的代码又可从根本上提高系统的正确性，有效减少调试工作量。

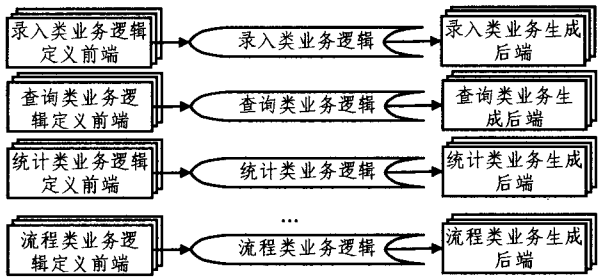


图 2 基于业务模式的自动生成

3 用于目标系统生成的专用代码方法

采用统一平台实现不同具体业务时，一种常用的实现方式是为不同业务生成对应的业务逻辑定义，然后采用同一份代码读入不同的业务逻辑定义，利用动态网页技术呈现相应的操作页面并完成业务逻辑操作[6]，以下简称为通用代码方法。以查询类业务为例，不同的查询业务生成不同的查询逻辑定义（如查询条件、结果显示方式等）。如图 3(a)所示，通用代码方法采用同一通用查询代码读入不同查询逻辑定义，由该通用查询代码实现所有查询业务的页面呈现与业务逻辑操作。

这种方法将目标系统核心功能代码集中在一起，直观看来是将目标系统的维护局限在有限范围内，可以增强系统的可维护性。但是在实际的应用中，平台中的通用业务模式很难适应所有业务逻辑的个性化需求（包括页面和处理逻辑的个性化需求等），因此往往需要对生成的目标系统进行二次定制与维护。而通用代码方式给目标系统的二次定制与维护带来了很大困难，主要表现在如下两个方面：

（1）目标系统界面定制与维护困难。通用代码方法通过程序代码动态呈现界面，使得用户无法使用可视化的方式对生成的界面进行定制与维护；并且采用通用代码实现所有同类业务的界面呈现时，由于只有一份实现代码，实现特定业务的个性化定制时，会导致在通用代码中插入大量定制不同业务的条件分支语句，使得通用代码变得越来越复杂，给维护带来很大困难。

（2）目标系统业务逻辑定制与维护困难。与界面的个性

化需求类似，同一类业务中的不同具体业务也经常存在业务逻辑的个性化需求，在通用代码中实现业务逻辑定制同样会导致在其中插入大量条件分支语句；并且在通用代码中，业务逻辑代码与界面呈现代码交织在一起，难以进行定制与维护。

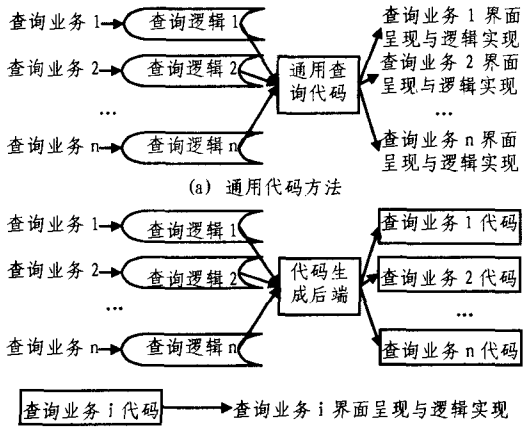


图 3 查询业务的实现方法

为解决上述问题，提出一种图 3(b)所示的专用代码方法，即针对每种特定业务的业务逻辑定义，业务生成后端直接生成该业务专用的实现代码，由每个业务自己的专用代码完成该业务的界面呈现与逻辑实现。在生成的代码中，利用目标平台的 MVC(Model-View-Controller)机制将生成的界面代码与业务逻辑代码分离。用户界面利用目标平台上的界面构件以静态的方式实现，业务逻辑则生成目标平台上的程序构件(一般为类)，供界面部分代码调用。如生成基于 Java 平台的目标系统时，界面部分利用静态的 HTML 与 JSP 界面构件实现，业务逻辑生成 Java Bean 供界面部分代码调用；生成基于. NET 平台的目标系统时，界面部分利用静态的 HTML 与 ASP 界面构件实现，业务逻辑则生成 ASP 页面关联的 C# 类或 C++类。采用这种方式生成的目标系统，用户界面可以在目标平台的开发环境中以可视化的方式进行定制与维护，实现了业务逻辑代码与界面呈现代码的分离，提高了目标系统的可定制性与可维护性；并且特定业务的定制与维护仅局限在本业务的专用代码中，不会影响其他业务；较好地解决了通用代码方法所带来的上述两个问题。

采用专用代码方法生成目标系统的具体业务时，业务逻辑定义中包含当前业务准确的需求定义，而要生成的程序单元的结构对于同一类业务来说是固定不变的。业务生成后端根据需求定义在固定结构的程序单元中填入当前业务特定的基本要素，可以得到完整的程序单元，实现业务的自动生成。以基本的单表查询类业务为例，业务逻辑定义中主要包含当前查询业务的查询条件、查询结果的显示内容等需求定义；待生成的程序单元包括界面单元和代码单元两部分，界面单元主要包括查询条件、操作按钮与查询结果三部分，代码单元主要包括初始化代码和查询执行代码两部分。按照图 4 所示的方式可以根据业务需求定义在界面单元和代码单元中填入当前查询业务的基本要素，得到本查询业务的完整实现。

4 复杂业务模式支持

在实际的信息系统中，除了基本数据库操作相关的业务逻辑外，往往存在更复杂的业务模式，并且常需要工作流的支持，以实现业务流程的自动化。在上述特定域软件体系结构的支撑下，抽取常见的复杂业务的业务模式，采用以下几种手段实现复杂业务的自动生成：

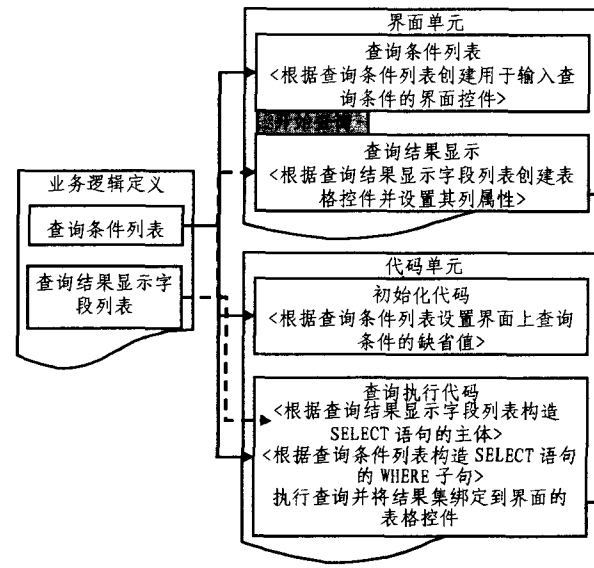


图 4 查询业务的自动生成原理

(1)引入包含外键关联的业务模式。实际信息系统的数据库中存在大量外键(foreign key)，而外键字段的值经常是唯一标识对应表中一条记录的数字串。业务操作时，直接录入外键值可读性很差，并且很难保证录入值的正确性。为此，在生成的业务代码中通过直接从外键关联的表中提取外键的值来实现录入。如果外键关联的表中记录较少，则在录入时采用下拉列表的方式列出所有可以选择的值，并且在下拉列表中不显示外键字段(一般为数字串)，而显示可读性更强的其它字段(如对应的文本说明字段)，利用下拉列表的“文本-值”关联实现通过可读性更强的显示字段选择外键值。如果外键关联的表中记录很多，则在外键录入时弹出对应表的查询页面，通过输入查询条件定位需要选择的外键值，选择的值利用页面中嵌入的脚本语言(如 JavaScript)返回。

实现包含外键关联的业务自动生成时所采用的业务逻辑定义和生成的关键要素为(采用 EBNF[7] 定义，后同)：

定义 3 业务逻辑定义中的外键信息

外键信息 → 外键关联表名 关联字段名 显示字段名 录入方式 ；

录入方式 → 下拉列表 | 弹出查询页面。

定义 4 外键录入生成的关键要素

界面单元要素 → 下拉列表控件 | 弹出项 ；

弹出项 → 编辑框 触发器 脚本代码 ； //脚本代码根据定义 3 生成弹出页面的 URL 及接收的返回值；

代码单元要素 → 初始化下拉列表控件 ； //根据定义 3 从外键关联表中提取数据，初始化下拉列表。

通过直接从外键关联的表中提取外键的值，既保证了录入的外键值的正确性，又通过可读性更强的显示字段或查询页面改善了人机交互效果。

(2)引入多表关联的查询业务模式。在查询类业务中，除了外键关联外，查询条件可能来自目标数据库表之外的其他表字段。如某机动车维修行业管理信息系统中，图 5 所示维修记录查询业务要求根据车辆颜色、品牌等基本信息(包含在

机动车基本信息表 GA_CarBaseInfo 中)查询特定机动车的维修记录(包含在维修记录表 GA_FixRecord 中)。为满足此类查询业务的自动生成需要,提出一种多表关联查询业务模式,即查询条件可以来自若干个目标数据库表之外的其他表字段,通过在业务逻辑定义前端配置查询条件表与目标表之间匹配的字段(如"CARNO"字段与"FIXCARNO"字段)建立它们之间的关联关系。在业务生成后端生成的查询代码中利用 SQL 语句的内连接(inner join)操作实现记录的筛选。

此类业务生成所采用的业务逻辑定义和生成的关键要素为:

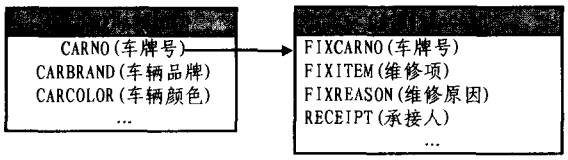


图 5 多表关联查询业务示例

定义 5 多表关联查询业务业务逻辑定义中的配置信息

关联查询配置信息 → 查询条件配置 查询结果配置;

查询条件配置 →{ 本表字段名 外表条件信息 };

外表条件信息 →(外表名 外表查询条件字段名 关联字段名) | ε; //为空表明查询条件来自本表

查询结果配置 →{ 查询结果显示字段名 };

定义 6 多表关联查询业务生成的关键要素

界面单元要素 →{ 查询输入条件} 查询按钮 查询结果表格;

代码单元要素 → 初始化代码 查询代码; //查询代码中利用 SQL 语句的内连接操作实现记录的筛选。

(3)基于基本操作组合构建复杂业务模式。实际信息系统中的某些复杂业务实际是基本数据库操作的组合,如订单录入类业务是查询操作与录入操作的组合。以报刊订单录入业务为例,查询操作查询客户期望订阅的报刊信息以进行确认,然后根据查询的结果录入订阅人以及订阅份数等订单信息。为支持基本数据库操作的组合,将基本数据库操作相关业务的前端与后端核心功能以软件构件(Component)的方式实现,并实现这些构件之间的连接件(Connector)。连接件分为前端连接件与后端连接件,分别实现前端构件与后端构件的组合,从而构造出组合业务的前端与后端,实现此类复杂业务的自动生成。连接件主要通过约定基本操作中字段之间的关联关系实现构件的组合,抽象出以下几种常见的关联关系:

定义 7 基本操作关联关系

基本操作关联关系 → 关联类型 源字段 目标字段;

关联类型 → 拷贝关联

| 修改关联

| 失效关联 ;

其中拷贝关联的语义为复制另一操作中某字段值到指定字段,如报刊查询结果中的报刊号字段复制到报刊订单记录中的对应字段。修改关联的语义为根据当前指定字段值修改另一操作中某字段值,如根据订票记录中订票数量字段值修改票源信息中剩余票数字段值。失效关联的语义为根据当前指定字段值修改另一操作中记录的状态,如根据房间预订记录中的房间号字段将对应房源记录标记为不可用。

组合业务生成的界面单元是基本操作界面单元的简单组合;生成的代码单元在包含基本操作代码要素的基础上,添加根据连接件配置的关联关系生成的关联相关操作代码而得到。

(4)提供数据库驱动的目标平台无关工作流机制。不同目标平台往往提供不同的工作流机制以支持业务流程,并且定义与执行业务流程的模式差别较大,如果采用目标平台的工作流机制,必然造成不同目标平台的后端实现差异很大,降低自动生成平台核心功能的可重用性。因此提出一种数据库驱动的目标平台无关工作流机制,基本思想是将工作流相关的流程定义、流程当前状态、流程执行的历史轨迹等信息以数据的形式存储在目标系统数据库中。流程的定义与执行均以数据库中存储的信息为驱动,从而使得不同目标平台的后端生成的代码模式相同,提高后端核心功能的可重用性。业务流程的每一步执行看作是基本的数据库操作附加流程状态转移与操作结果记录动作,因此生成的业务流程实现以基本的数据库操作为基础,扩展业务流程执行需要的程序要素而得到。

此类业务生成所采用的业务逻辑定义和生成的关键要素为:

定义 8 业务逻辑定义中的业务流程配置信息

业务流程配置信息 → 业务记录表名 {流程节点};

流程节点 →{ 节点基本信息 节点授权信息 节点跳转信息 };

节点基本信息 → 节点编号 节点名称;

节点授权信息 →{授权角色名 {可操作字段名 } };

节点跳转信息 →{ 跳转条件 目的节点编号 }。

定义 9 业务流程生成的关键要素

界面单元要素 → 数据库基本操作界面要素 业务流程界面扩展要素;

业务流程界面扩展要素 → 界面流程状态转移要素 界面操作结果记录要素;

界面流程状态转移要素 → 签收按钮 |(下一节点选择列表 转下一节点按钮);

界面操作结果记录要素 → 操作结果录入框;

代码单元要素 → 数据库基本操作代码要素 业务流程代码扩展要素;

业务流程代码扩展要素 → 查询代码扩展要素 | 录入(编辑)代码扩展要素;

查询代码扩展要素 → 查询条件扩展; //根据定义 8 配置的节点授权信息限制只能看到需要当前角色处理的信息;

录入(编辑)代码扩展要素 → 权限检查代码 流程状态转移代码; //权限检查代码根据定义 8 配置限制只能由当前节点授权的角色操作,并设置可修改的字段;流程状态转移代码根据定义 8 配置与用户界面操作修改流程状态。

5 应用案例分析

基于上述特定域软件体系结构实现了 Web 信息系统自动生成平台,采用专用代码方法生成目标系统的业务逻辑,并将实现的平台用于陕西华茂科技发展股份有限公司的治安管理信息系统[9]开发。业务逻辑定义采用可视化前端,目标平台为. NET,开发了机动车修理业、机动车报废业、印刷业治安管理信息系统,每个行业系统分部级、省级、地市级以及企业端四级,共计 12 个治安管理信息系统。系统包括表 1 中所示信息录入/编辑、查询、统计、布控报警分析等基本功能共计 314 项,

(下转封三)

```
include /usr/local/openldap/etc/openldap/schema/core.schema
include /usr/local/openldap/etc/openldap/schema/cosine.schema
include /usr/local/openldap/etc/openldap/schema/nis.schema
include /usr/local/openldap/etc/openldap/schema/inetorgperson.schema
include /usr/local/openldap/etc/openldap/schema/dns.schema
include /usr/local/openldap/etc/openldap/schema/gsinfoaccount.schema
include /usr/local/openldap/etc/openldap/schema/gsinfouua.schema
# Define global ACLs to disable default read access.
pidfile /usr/local/openldap/var/run/slapd.pid
argsfile /usr/local/openldap/var/run/slapd.args
database bdb
suffix"dc=gsinfo,dc=cn"
rootdn"cn=root,dc=gsinfo,dc=cn"
# Cleartext passwords, especially for the rootdn, should
# be avoid. See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw jsj322
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory /usr/local/openldap/var/openldap-data
# Indices to maintain
index objectClass eq
```

2.4 目录信息数据导入

目录信息导入有两种方法：即从命令行手工输入和通过数据库交换文件 LDIF 导入。从命令行手工输入是最直接的一种，但是对于大批量的数据，工作量很大；LDAP 提供一种更加简明的目录信息表达方法——数据库交换文件 LDIF，用于导入和导出目录信息。它是目录交换格式文件，以文本方式来表示 LDAP 中的条目。采用这种方法不需要启动目录服务进程，当需要创建大批量的条目时，使用这种方法比较适合。

总而言之，比起从命令行手工输入数据，LDIF 文件有着明显的优势，但必须遵守正确的语法来将用户信息输入文件，并将它导入目录中。

结束语 目前，本系统已应用于甘肃科技信息网的网络运行中，并在甘肃科技网络的建设中发挥着重要的作用。一方面降低了网络应用系统中用户管理的成本，提高了新应用系统的开发效率，尤其提供包括单点登录在内的认证服务，在应用系统集成中发挥了显著的作用，成为甘肃科技网络构架中不可缺少的一部分。另一方面有了通用的用户信息基础结构，可以集中管理用户角色，制定安全策略，减少各应用系统权限管理的维护量，提高工作效率和整体安全性。

参 考 文 献

[1] 焦静，李勇. 基于 LDAP 的统一身份认证的设计与实现[J]. 科学技术与工程，2007，7(4)：646-649

[2] 胡毅时，怀进鹏. 基于 Web 服务的单点登录系统的研究与实现[J]. 北京航空航天大学学报，2004，30(3)：236-239

[3] 涂德志. LDAP 协议研究与 LDAP 服务器的设计与实现[D]. 电子科技大学，2002(5)

[4] 许鑫，苏新宁，陆炯. 数字化校园身份认证系统的设计. 现代图书情报技术[J]，2005(4)：51-57

[5] 宋志伟. LDAP 协议研究[D]. 南京理工大学，2003(1)

(上接第 287 页)

其中 288 项功能(占总功能项数的 92%)由自动生成平台生成，自动生成的功能中有 11 项进行了二次定制与维护。

表 1 治安管理信息系统功能简表

	修理业	报废业	印刷业	合计
录入/编辑	46 项	42 项	44 项	132 项
查询	31 项	28 项	25 项	84 项
统计	22 项	19 项	18 项	59 项
布控报警	16 项	16 项	7 项	39 项
合计	115 项	105 项	94 项	314 项

上述系统借助自动生成平台用时 4 个月即开发调试完成，并通过了公安部关于此类治安管理信息系统的现场测试，目前已在陕西省咸阳、榆林等地市实施运行。

结束语 本文提出了一种基于代码生成的 Web 信息系统工程化开发方法，实践证明可以有效缩短以数据库为核心的 Web 信息系统开发周期。提出了类似编译器基础架构的特定域软件体系结构，实现了前端与后端的共享；提出的专用代码方法将生成的业务逻辑代码与界面呈现代码分离，提高了目标系统的可定制性与可维护性；并采用多表关联、基本操作组合以及数据库驱动的目标平台无关工作流机制支持了复杂模式业务逻辑的自动生成，适应了复杂 Web 信息系统的实际开发需要。

同一类业务的不同目标平台后端的核心逻辑是类似的，主要区别是待生成的目标平台的基本语句语法不同，因此后端的可重用性有待进一步加强。此外，在实际的 Web 信息系统中，仍存在一些第 5 节中采用的手段无法支持的复杂业务模式，这些业务模式在一定应用领域内具有较强的通用性。如何增强代码生成后端的可重用性以及支持应用领域内通用的复杂业务模式，是下一步的研究重点。

参 考 文 献

[1] Struts. http://struts.apache.org/

[2] Hibernate. http://www.hibernate.org/

[3] Ruby on Rails. http://www.rubyonrails.org/

[4] .NET Maker. http://www.hkvstore.com/aspnetmaker/

[5] UCML. http://www.ucml.com.cn/cpzx.asp

[6] 徐世莲. 基于软件体系结构的 WEB_MIS 应用平台设计. 计算机科学，2006，33(9)

[7] 刘坚. 编译原理基础. 第 1 版. 西安：西安电子科技大学出版社，2002：64

[8] Christopher A，Simmon C，Catherine A C. 关系数据库和 SQL 编程. 第 1 版. 皮人杰，等译. 北京：清华大学出版社，2005：164

[9] http://www.wisdomstock.com/web/hmrj.asp