# OFBIZ2.0 精简版本应用概论

***OFBIZ2 研究 ——中科学院软件所 互联网软件技术实验室 软件工程组***

**雷辉 （leihui@intec.iscas.ac.cn） [2003-4-28]**

# 1 使用 OFBIZ 的理由

## 1.1 什么是 OFBIZ

OFBIZ 是由 Sourceforge 维护的一个最著名的开源项目之一，提供创建基于最新 J2EE/XML 规范和技术标准，构建大型企业级、跨平台、跨数据库、跨应用服务器的多层、分布式电子商务类 WEB 应用系统的框架。

OFBIZ 的 Web 应用框架以 MVC 模式搭建而成，整体采用了很多被大多数企业级应用系统公认的位于业务逻辑层和集成层（Business Tier and Integration Tier）的设计模式。许多表示层（Presentation Tier）的设计模式也被引入进 OFBIZ，但是仅仅体现在 Servlet 控制器（the servlet controller）中,没有包括在实体引擎中。在实体引擎中使用的设计模式包括：业务代表（Business Delegate），值对象（Value Object）， 符合实体（Composite Entity(variation)），值对象组装器（Value Object Assembler），服务定位器（Service Locator）和数据访问对象（Data Access Object）。OFBIZ 正在计划逐步引入其它设计模式和完善已经引入的设计模式的实现。

使用 OFBIZ 的框架和组件，可以大大缩短开发企业级 WEB 应用系统的进度和成本。详细情况请参见：http://sourceforge.net/projects/ofbiz/。

## 1.2 OFBIZ 和其它项目的比较

与 ofbiz 类似的项目还有很多，ofbiz 与这些项目的最主要的不同点是 ofbiz 提供了一整套的开发基于 Java 的 web 应用程序的组件和工具。一个优秀的 web 应用程序应该是至少三层结构：表示层，业务逻辑层和数据层。大多数应用框架，比如 Struts, Cocoon, 和 Velocity 将主要精力都集中在了表示层。比如 Struts，遵循了(MVC)构架，使用 Java Bean 和 Action 类与 JSP 页面进行通讯。Struts 是一个很好的 web 应用框架，但它并没有提供访问数据库的组件，也没有提供控制工作流的组件。如果要使用，你必须自己创建这些组件。如果已经在利用其它的应用框架（如 Struts），你也可以很容易的将 ofbiz 的组件添加到自己的工程中。

与其它类似开源项目相比，OFBIZ 是一套有血有肉的包含编译打包部署工具、应用组件、示例应用等内容的企业级 Web 应用系统实现框架。

## 1.3 开源的优势

OFBIZ 是一个开源项目，由几个牛人在维护，它具有开源项目的一切优势，如免费（随时下载）；集市式开发方式；成千上万的人在维护，也在测试等等。也具备开源项目的所有缺点，如缺乏技术文档，提交的系统没有全面测试；不稳定等等，但无论如何，我们要清醒的认识到：

1、 OFBIZ 是一个开源项目。

2、 OFBIZ 只仅限于系统开发者使用。

## 1.4 完善的实体引擎

OFBIZ 实体引擎提供了一组工具和设计模式来对现实世界中特定的实体（数据对象）进行建模和管理。在本系统的上下文环境中，一个实体就是一个由多个数据域（fields）和该实体与其它实体之间的关系所组成的一个数据对象。这个定义来自于关系型数据库对实体关系模型（Entity-Relation modeling）概念的标准定义。实体引擎的目标是简化企业级应用中对实体数据（对应关系型数据库表）的大量操作，包括定义、维护、通用操作（增、删、改、查实体和实体之间的关系）的开发工作

实体引擎采用了很多被大多数企业级应用系统公认的位于业务逻辑层和集成层（Business Tier and Integration Tier）的设计模式。许多表示层（Presentation Tier）的设计模式也被引入进 OFBIZ，但是仅仅体现在 Servlet 控制器（the servlet controller）中，没有包括在实体引擎中。在实体引擎中使用的设计模式包括：业务代表（Business Delegate），值对象（Value Object）， 符合实体（Composite Entity(variation)），值对象组装器（Value Object Assembler），服务定位器（Service Locator）和数据访问对象（Data Access Object）。OFBIZ 正在计划逐步引入其它设计模式和完善已经引入的设计模式的实现。

实体引擎的一个主要目标是尽可能的提供一种通用的代码结构，来消除在针对每一个实体的事物处理过程中，所有写死（hard code）的代码。 这种系统抽象所关注的问题，与那些把数据从数据库中提取出来，并以报表的形式进行输出和显示处理的报表管理或类似系统是不同的，而是类似于每日都可能发生很多事物处理的商业应用系统，实体引擎能大量节省构建类似应用系统的开发费用和戏剧性的减少因为系统存在大量写死的事务处理代码所产生的 bug。这种类型的应用系统目前在 OFBIZ 中实现了一些，如电子商务，入库、出库的帐目管理，任务分配资源管理等等。这些工具能够用来报告和分析系统，但是并不意味着，它能包容千差万别的客户的应用需求，在实际应用中，我们可以基于它来做一些二次开发。

为了达到尽可能少的在系统中出现与针对特定实体操作有关的代码， 存储实体属性值的对象结构必须设计成通用的，可以用一个 MAP 对象来存贮实体的所有域（也可以叫字段或属性），通过实体的名称来区分它们是哪个实体的。根据字段的名称，用一个简单操作 String 数据的方法，来从值对象中读出或写入某字段的值，并且还可以验证给定名称的字段是否是该值对象的一个合法域。在实体引擎和应用系统之间建立了一个约定（体现实体结构定义文件和字段类型、Java 数据类型、SQL 字段类型映射关系的定义中），这个约定被定义在特定的 XML 文件中，以减少这种灵活性可能存在对系统不利的风险（如数据类型问题可能引起数据库系统崩溃等）。

代替不在系统中书写针对特定实体操作的的代码的方法之一，是把实体的定义放在 XML 文件中，在系统启动的时候，由实体引擎负责把这些结构定义加载进内存，并且在应用程序和数据源（通常指一个数据库或其它资源）之间建立一些对这些定义的使用规则。在这些 XML 实体定义中，对实体和属于它们的域，以某种规则和数据库中的表和表的字段建立映射关系。而且实体与实体之间的关系，实体域的数据类型，Java 语言的数据类型，SQL 数据类型之间的映射关系，也被定义在 XML 文件中。 实体之间的关系还可以被命名，以用来区分不同实体组合之间的特定关系。

基于实体引擎这个抽象层，与特定实体操作有关的代码的编写就变的很容易创建和修改。 使用实体引擎所提供的 APIs，编写处理实体持久性（增、删、改、查）的代码，可以不同的方式来配置，以便于实现针对实体持久性操作（增、删、改、查）有变化时，可以不改变代码本身，因为它并没有写死。这种抽象的一个典型应用场景就是你既可以通过 JDBC

直连方式，也可以通过调用运行在 EJB 服务器上的实体 Bean（Entity Beans）的方式或者以其它方式，甚至在系统所提供的框架范围内，使用者运用自己扩充的方式去完成对实体持久性的改变等等。这些不同方式的切换并不需要对代码做任何改动，只需要修改配置文件。

OFBIZ 已经完全实现了自己设计的一套实体引擎，可以直接使用。

实体引擎的好处举例说明如下：

以前的问题背景。

1、 你需要借助工具或手工去维护已经存在的或新增加的数据库结构（库结构，表结果等的定义和更新），如果要修改表结构和定义的话，怎么做？

2、 假设你的应用涉及 200 张表（实体），假设每张表（实体）都存在增、删、改、查，则需要在你的应用中静态构造（硬编码）800 个 sql 语句。

3、 假设这 200 张表之间存在 100 种关系，维护每一种关系的增、删、改、查，又需要 400 个静态构造的 sql 语句。

4、 假设这些 sql 语句由 10 个不同水平的程序员来构造，构造出来的 sql 语句在执行性能上可能存在巨大差异，而且形态各异。

5、 这些硬编码的 sql 语句分布在大量 Java 程序的各个角落，一旦某张表的结构发生变化或者修改某一字段名或表名，意味着什么？意味着混乱！

OFBIZ 是如何解决这些问题的：

OFBIZ 拒绝这种混乱，一套 EntityEngine 机制轻松解决上述所有问题。

1、 涉及 1 张表（实体）的增、删、改、查，它提供一套处理机制（不到 12 个类，大约 5 千行代码），应用的规模是 10000 张表，它还是这套处理机制（不到 12 个类，大约 5 千行代码），而且这些处理机制由 JAVA 程序高手生成和维护，可以保证其合理性、可靠性和安全性。

2、 EntityEngine 提供了一个构造复杂 sql 操纵语句的机制，你可以根据需要随时构造任意复杂的 sql 语句，完成你想要做的事情，这样你可以在开发过程中，随时修改你的数据库定义，OFBIZ 在系统启动时会自动加载并检测数据库中的不一致性和参考完整性。

3、 实体引擎大大简化了涉及关系型数据库的管理和维护，但这还只是一小块好处，大的好处是你在实现一个复杂需求的应用时，实体引擎用为数不多的几个类解决了你所有的问题，实现任意复杂的数据库存取业务和商业逻辑，而且与需求的复杂度和数量无关。

4、 好处太多了，在使用的过程中，会进一步的体会到。

## 1.5 锦上添花的服务引擎

服务框架（Services Framework）是 OFBiz2.0 新增加的内容。服务（Services）被定义成一些相对独立的逻辑处理单元（服务具有业务逻辑处理的原子性），能够被灵活的组合成不同的形式去实现不同的商业逻辑需求。服务有多种类型的实现形式：工作流（Workflow），（规则） Rules, Java 程序（Java），简单对象访问控制协议（SOAP），轻量级 Java 程序脚本语言解释器（BeanShell）等等。 如果一个服务被定义成"java"类型，意味着实现该服务的机制可能就是 Java 类的一个 static 方法，而且，OFBIZ 提供的服务框架不限于使用在一个基于 Web 的应用程序系统中。服务需要输入一个 Map 形式的参数，服务处理完毕后，返回的也是一个 Map 形式的结果集。这个思路是非常好的，因为 Map 类型的数据格式很容易被序列化（serialized，序列化成字节流），并且通过 HTTP(或 SOAP)的协议进行存储和传输。

在 OFBIZ 里，服务被定义 XML 文件里，定义后的服务被分派给一个特定的 服务引擎（Service Engine） 。 服务引擎 具体负责以合适的方式进行服务的定义、管理和调用。 因为服务不一定被绑定在某基于 Web 的应用程序运行环境中，所以服务处理的结果也就不一定会和某 erquest 请求的响应 reponse 联系在一起，这样就允许服务可以在预先设置好的和时间点上定时触发（因为它不需要一个 Http Request 请求），一般是通过系统提供的 工作日程管理器（Job Scheduler） 运行环境触发（用定时器来控制对服务的调用）。

服务还可以互相调用调用，即一个服务被设置去调用任何其它的服务。这样，我们可以用更小粒度的已经定义好的服务组合成一个服务链，来完成一个比较大的任务，而且这种组合是任意的，从已经定义好的服务本身来讲，是很容易复用的。使用不同的应用程序系统中的服务，可以通过创建一个"全局服务定义文件"只被定义一次（因为服务本身是实现了特定的商业逻辑，它和具体应用的关系应该是松耦合的），当然，服务也可以通过一些限制，被指定为特定的应用程序所用。

在一个基于 Web 的应用系统中，服务可以被用来实现基于 Web 的事件（web events），利用服务实现事件处理，可以在服务框架内最大可能的复用相对固定的一些业务逻辑。而且，服务还可以被定义成对可输出的（exportable），意思是它们可以被系统外部的东西（可能是一个应用系统或其它）远程访问。 目前系统实现了一个基于简单对象访问控制协议（SOAP）的事件处理器，该事件处理器，就允许外部应用通过 SOAP 协议对运行（或定义）在其上的服务进行远程访问 。在将来，会有更多的远程调用形式被加到服务框架里

## 1.6 双管齐下

实体引擎和服务引擎各有利弊，在实际应用中，可以把服务引擎和实体引擎结合起来使用，实体引擎主要用于处理实体（Entities）对象的增、删、改、查，服务引擎主要用于处理商务逻辑，这种商务逻辑的定义范围，不大会遇到上面所说的要求一次查询返回一个结果集这样的服务定义（这完全可以用实体引擎来处理）。

服务引擎可以用处理跨平台、跨操作系统、跨应用系统之间的业务逻辑。把实体引擎和服务引擎结合起来，可以这解决企业级 Web 应用系统的绝大部分需求所涉及到的技术实现细节。

## 1.7 其它甜饼

OFBIZ 的野心太大，几乎想通吃所有最新的关于企业级、多层、分布式应用系统的构建技术。除最成熟的实体引擎和服务引擎之外，它还涉及到以下系统实现引擎。

1、 工作流引擎
2、 规则引擎
3、 消息引擎

这三项工作，除工作流引擎尚为 alpha 版本之外，其它两个都在建设之中，看来，已经把这几个人累的够呛。

## 1.8 主流技术的采用和跟踪

OFBIZ 的框架中引入了最先进的主流开发技术 Web 应用系统构建技术，如：Xerces (xml.apache.org) ；Xalan (xml.apache.org) ；Axis (xml.apache.org) ；Log4J

(jakarta.apache.org) ；Castor (www.exolab.org) ；ORO (jakarta.apache.org) ；BeanShell (beanshell.org) ；J2EE1.3，XML1.2 等，而且整个系统，在原来的基础上不断的被重构和修订。

## 1.9 扩展性和可移植性

OFBIZ 所提供的系统框架，是一个纯 Java 的应用程序，在具体实现中采用了大良的设计模式，本身系统的实现完全符合面向对象的设计原则中的几乎所有要求，除采用 J2EE 核心设计模式、数据库设计模式之外，OFBIZ 的实现代码中，大量引入和 Java 设计模式，其应用系统本身的扩展性和可移植性已经没有问题。

OFBIZ 开发者同时维护和 Weblogic，Tomcat，Jboss，Resin，orion 等 16 个厂商的 Web 和 App 应用服务器的兼容版本.

OFBIZ 开发者同时维护和 oracle，Mysql，Sybase，PostgreSQL，Hsql 等数据库产品的兼容支持，包括编译、打包、部署到这些数据库产品或应用服务器产品的运行环境下。

OFBIZ 开发者同时在 Unix 和 Windiws 两大操作系统上进行开发和测试，而且具备 Java 应用系统的所有跨平台的特点。

有了这些，对于大型企业级应用系统的具体、特定实现来说，你有信心实现所谓的“一次开发，到处运行”。

## 1.10改进的事务处理功能

目前 OFBIZ 提供 4 种数据库连接方式的支持（在“ EntityEngine.xml”文件中配置，被“ EntityConfigUtil ”类装载进内存）。

用在：

GenericDelegator ——> GenericHelper.

GenericHelper ——>GenericHelperDAO

GenericHelperDAO ——>GenericDAO

GenericDAO ——> SQLProcessor

SQLProcessor ——> ConnectionFactory 类的 get Collection()方法得到数据库连接。然后构造 PrepareStatement 来实施数据库操作。

OFBIZ2.0 改进了 GenericDAO 和 SQLProcessor，综合利用 JDBC 的事务管理和应用服务器的事务管理功能实现多层分步式事务管理功能，因为不同的实体操作可以对应不同的实体引擎（在 EntityEngine.xml 中通过实体所在的组了配置），这样可以在 OFBIZ 的主运行环境下，通过实体引擎的配置实现对远程数据源的访问操作，而一旦连接上远程数据源，OFBIZ 就提供一套机制，把针对本地和远程数据源的操作纳入到同一事务管理范围内，实现分布式事务处理。

OFBIZ 利用 JDBC 提供的数据库操作事务管理 API（commit，rollback 等）和第三方工具所实现的数据库操作事务管理 API，实现了 OFBIZ 的实体引擎对事务的控制。

# 2 不使用 OFBIZ 的理由

## 2.1 系统过于庞杂

确实如此!!它用到了 XXX，XXX，XXX 标准，体现着 XXX，XXX，XXX 技术，维护着诸多的概念、理念、包含着这么多的设计模式，光配置文件就有 30 个之多，涉及到的配置项不下 200 种，它要用到很多工具，这一个理由足以让大多数人望而却步！

OFBIZ 太复杂，把基础、公用的东西和特殊应用混到一块，特别在实体定义的时候（考虑关联关系）。

## 2.2 夸夫追日

如果把 OFBIZ 比做太阳的话，使用 OFBIZ 的人就是夸父，因为一旦你采用了 OFBIZ ，它就会诱惑你，永远跟踪下去，和其保持同步，和它保持同步的同时，意味着你必须不端的充电，和 XXX，XXX，XXX 规范保持一致；和 XXX，XXX，XXX 标准保持一致；和 XXX，XXX，XXX 工具保持一致，这样你会很累，那有我用 JDK 写的东西维护起来的轻松？

可是纯粹使用 JDK，你又能写出什么东西？

## 2.3 它不适合我的应用规模

的确如此，你如果是开发一个。。。。，或者。。。。或者。。。。这都用不上 OFBIZ，OFBIZ 是用在什么规模上的（大家自己领悟吧）。

## 2.4 关于自带的应用

OFBIZ 自带的应用，美国化的东西太多，应用背景和我们差距太大。除用户管理，系统维护，知识管理几大块之外，其它的基本上用不到。

## 2.5 它有 bug?

祝贺你，你竟然发现了 OFBIZ 的一个 BUG，但这是开源项目最为常见的一个问题，你可以想想，再回头看看“开源项目的维护方式”，这些 BUG 本身就是希望你发现并改正（或提修改建议）的，所以 BUG 不是什么大事。在使用开源项目，来缩短开发周期和降低开发成本的应用前提下，更需要一种“杨弃”的理念。

## 2.6 它连个论坛都没有?

OFBIZ的开发者没想到企业级应用关注的却是一个论坛？这不是他们关心的内容，而且你完全可以利用 OFBIZ 提供的一切，迅速构建一个论坛（前提是论坛的需求必须清楚）。

## 2.7 其它理由

考虑一下，OPENSOURCE 的东西可能存在其它问题，如版权问题？协议问题等等。这好象对于我们来说，并不是问题。

# 3 简化 OFBIz2.0

本着实际应用的目的，我在 OFBIZ 最新版本的基础上，进行了简化，主要包括如下措施。

## 3.1 数据模型改造

1、 数据模型只保留：common，content，party，security。

2、 改造所有实体定义文件，把 DTD 包含在每一个文件里。

3、 首先整理 entityGroup.xml 文件，确定将要删除的实体定义，包括保留的数据模型包和在保留的包中删除不需要的，如下：

（1）common 和 security 没动。

（2）content，保留大部分内容（如文档管理，支持网站统计的实体等，其它的删除），删除 content.preference，content.subscription，content.website。

（3）party。删除一部分和 party 无关的实体模型定义，如 party.agreement，party.communication，party.need 等。

4、 根据删除的实体名称，查询确定被删除的实体所有的关联（包括实体和基础数据定义，java 文件,jsp,xml,properties 文件中出现这些实体的地方，逐一进行清除和改造）

5、 修改 entityengine.xml，删除加载的实体定义文件（eccomerence 里）

6、 删除加载的与应用有关的服务定义文件。

7、 修改 serviceengine.xml，删除加载的与应用有关的服务定义文件，删除加载的基础数据（与应用有关的）。

## 3.2 删除特殊应用

1、 删除 accounting；catalog；ecommerce；facility；marketing；partymgr；workeffort 应用及与应用有关的存在于 core 和 commonapp 里的程序文件。commonapp/src 目录下只保留：common，party，content，security。如果有类似被删除应用的应用背景，再考虑引进。

2、 修改 setup/ofbiz/build.xml 文件，把这些特殊应用有关的内容全部清除，把 images 应用从 ecommerce 应用里移到 content 应用里。

3、 修改 setup/catalina41/conf/server.xml 文件。

4、 修改部署环境，保留目前 ofbiz 支持的最好的部署环境：bea,jboss,resin,catalina41.其它以后再说。

## 3.3 简化后的系统

其实简化后的系统， OFBIZ 的内核并没有动，只是把它自带的应用全部删除（包括和应用相关的一切痕迹），但保留下来了常用的应用

1、 内核管理系统——Commonapp 应用。

2、 用户管理——PartyMgr 应用。

3、 知识管理——Content 应用。

4、 系统维护工具——WebTools 应用。

精简后的内核框架具备如下功能

（1） 全面支持 EntityEngine，ServiceEngine；workflowEngine。

（2） 支持用户管理，权限管理，知识管理（信息发布就是小意思了），联系方式管理，还具备一些通用基础数据的定义和数据本身。

这是我们目前所面临或即将面临的大多数“基于 J2EE/XML 技术的多层、分部式企业级 Web 应用系统”所必须具备的功能。

简化或的框架可以作为我们开发自己应用的一个原始内核。在该原始内核的基础上，可迅速开发和部署我们自己的应用。

这个和适用于培训体系的 OFBIZ 版本的研究不冲突，可同时进行。

简化后的好处，直接体现在系统规模大大减少，系统外观复杂度大大降低，系统中我们琢磨不定的东西，但可能永远都用不上的垃圾基本被清除了。

## 3.4 系统规模

统计规则：每行 80 字节，10%空行率。

### 3.4.1 简化前

jsp+java 文件，共 6405083 bytes。
Java 文件，共 4197776 bytes。
约 4.7225 万行代码；497 张表 24 个视图。

### 3.4.2 简化后

jsp+java 文件，共 3104918 bytes。
Java 文件，共 4197776 bytes。
约 2.8772 万行代码；107 张表 8 个视图。

# 4 客户程序编写者需要了解的类和接口

如果有一个完整的基于 OFBIZ2.0 的开发过程定义和 OFBIZ 资深顾问做基础，一个开发团队中的开发者只需要了解以下类所提供的接口即可在 OFBIZ2.0 框架的基础上，开发基于实体引擎和服务引擎的应用程序。

## 4.1 实体引擎核心应用类（客户端 API）

涉及到 12 个类，GenericDelegator，GenericValue，GenericPK，EntityCondition，EntityExpr，EntityFieldMap，EntityConditionList，EntityWhereString，EntityOperator，

EntityOperator，EntityListIterator，这些类都是为 GenericDelegator 的接口服务的。用户端程序和数据库之间的所有交往多是通过“ GenericDelegator ”完成的。

## 4.2 服务引擎应用类（服务器端 API）

涉及 LocalDispatcher， GenericDispatcher； ServiceDispatcher；ServiceUtil； DispatchContext ；ServiceConfigUtil 等 6 个类。

## 4.3 常用工具类

工具类主要在包 org.ofbiz.core.util 中。

1、 属性文件访问工具类：UtilProperties。
2、 Map、List 对象操作工具类：UtilMisc。
3、 UtilFormatOut ：通用格式化输出工具类（主要用在 Jsp 文件或 View Helper 中）。
4、 UtilURL：得到文件流的 URL 地址类。
5、 UtilCache：缓存管理类。
6、 UtilValidate：通用数据输入输出数据校验（合法性和有效性）类，可任意扩展。.
7、 UtilDateTime：java.util.Date 和 java.sql.Date 格式的日期/时间处理类。
8、 StringUtil：增强的字符串处理类。
9、 UtilXML：增强的符合 JAXP & DOM 规范的 XMl 解析器处理工具类。
10、 SiteDefs：常数定义类，定义所有 Web 程序用到的和环境有关的常量。
11、 Debug：格式化输出程序调试信息类。
12、 HttpClient：模拟一个 HttpServlet 请求类。
13、 HttpRequestFileUpload：接受一个通过 Http 上传的文件工具类。
14、 SendMailSMTP：符合 SMTP 协议的邮件发送处理类（实现发送邮件服务器的功能）。

## 4.4 其它

作为一个被动的开发者来说，用好上述这些类加上基于 OFBIZ 的开发过程定义就可以了；但对于一个真正要用好 OFBIZ 的开发者，远不止上面这些，需要全面理解和掌握 OFBIZ 的流程、框架和代码。

本文旨在抛砖引玉，能让更多的 Web 应用开发者从中收益，是开源的本意，也是我们软件所软件工程组的责任和研究理念。

中国科学院软件所互联网软件技术实验室成立于 1998 年，是软件所的直属研究开发部门，具有一支以系统支撑软件技术、数字化技术、软件工程和软件质量保证技术、Internet 技术为主要研究领域的科研开发队伍。笔者所在的开发团队从 2001 年就开始跟踪和使用 J2EE/XML 技术，并在 OFBIZ1.0 的基础上实现了北京软件产业基地公共技术支撑体系——综合服务管理平台（www.bsw.net.cn）。从今年 2 月份开始成立研究小组，跟踪和研究基于基于 OFBIZ2.0.0 的通用 Web 应用系统开发框架。

该简化的版本已经经过测试，如果感兴趣的话，可以和本人或研究小组联系，以索取简化版本的源码和总体介绍文档。

联系方式：leihui@intec.iscas.ac.cn。 雷 辉