面向 J2EE 主流框架的 MDA 模型转换研究

谢培基,余金山
(华侨大学 计算机科学与技术学院,福建 泉州 362021)

摘 要:提出一种基于元模型和 UML Profile 的模型转换方法。利用 UML Profile 来描述基于 SSH 整合框架系统的 PIM 和 PSM,解决了基本的建模语言不能完整地描述出 J2EE 主流框架下的平台相关模型的问题。提出了一个能适合 SSH 整合框架的 Web 应用层次结构作为 PIM 元模型,解决元模型间映射规则的复杂、难以匹配的问题。利用基于元模型的模型转换方法制定 PIM 元模型到 PSM 元模型的转换规则,间接实现 PIM 到 PSM 的模型转换。

关键词:MDA;J2EE 主流框架;模型转换;元模型
中图分类号:TP311 **文献标识码**:A **文章编号**:1674-7720(2012)17-0011-04

The research on MDA model transformation oriented to J2EE mainstream framework

Xie Peiji, Yu Jinshan
(College of Computer Science and Technology, Huaqiao University, Quanzhou 362021, China)

Abstract: This paper provides a model transformation method which based on the meta-model and the UML Profile, describe the PIM and PSM system based on Struts, Spring and Hibernate(SSH) integration framework by the UML Profile. The method solves the problem of the basic modeling language can't descript the PSM based on the J2EE mainstream framework. This paper also proposes a Web application hierarchy as a PIM meta-model which suitable for the SSH framework, solve the problem of complex mapping rules between element model. Then use the model transformation method to formulate model transformation rules to transform the PSM to the PSM.

Key words: MDA; J2EE mainstream framework; model transformation; meta-model

计算机业界一直在探索一种新方法,既能够提高软件的开发效率,又能使所开发出来的软件能够有更高的质量和更长的生命周期。面向对象思想、分布式计算开发、基于组件开发等新方法都为这一探索起到了应有的作用。而模型驱动架构(MDA)同样也为软件开发的进步发挥了重要作用。

MDA (Model Driven Architecture)[1] 是 OMG 组织在 2001 年正式提出的一种模型组织管理框架,它定义了平台无关模型(PIM)和平台相关模型(PSM),同时把 PIM 到 PSM 之间的自动映射定义为模型转换。MDA 提供了一系列指导软件开发的方法,这种开发方法将传统的重编程过程提高到了重系统模型设计的层次,使得开发人员能更好地设计系统,有效地提高了软件的开发效率,同时降低了编程开发费用、解决了系统需求不断变化带来的维护性困难问题。

1 MDA 模型转换方法

在基于 MDA 的软件开发过程中[2],主要的两个步骤是 PIM 到 PSM 的模型转换和 PSM 到代码的转换。因为 PSM 与其对应的系统实现平台技术紧密相关,因此这一转换相对较为直接,且有众多 MDA 工具的支持。而将 PIM 转换到 PSM 是一个较为复杂的过程[3],也是当前研究 MDA 的一个重点。本文归纳出了几种模型转换方法。

(1)手动转换方法:即系统开发人员使用已经定义好的模型操作的 API,将源模型转换到目标模型的方法。例如 JMI,在 J2EE 中提供了一套完整的 JMI 的类库,系统开发人员可以通过编程实现相应的模型转换。这种转换并没有使模型转换自动化,所以很难在实际应用中得到推广。

(2)基于 XMI 的模型转换方法[4]:实际上也是一种文本转换方法,该转换方法将源模型用 XMI 文本的形式表示,也相当于用 XMI 来存储源模型,然后将 XMI 里的元素一个个按步骤转换到目标模型。这种转换方法的优点是相对简单,缺点是转换速度慢、步骤较多且繁杂、不够直观、容易导致模型前后的不一致。

(3)基于模式的模型转换方法[5]:该方法建立在其他模型转换方法的基础上,引入模式的概念,将一些可用的模型转换方案包装成模式,然后以模式的形式放到存储库里以实现模型转换方案的添加删除和复用。这种模型转换方法通常会改变源模型的结构信息,使其更加符合设计理念,而没有改变模型的抽象级别和增加它的语义。

(4)结构驱动转换方法:该方法主要分为两个不同的步骤[6]:①为目标模型建立多层次的体系结构;②设置目标模型中的索引和属性。这种转换方法设计了一个总的规则调度框架和转换机制,用户需要做的就是提供转换规则,然后实现模型的转换。著名的模型驱动开发工具 OptimalJ 中的模型转换框架就是使用结构驱动的方法。

面向 J2EE 主流框架 Struts、Spring 和 Hibernate 等 MDA 模型转换,当前还没有标准化的模型转换方法,以上的几种模型转换方法也未能提供有效的支持,其中方法(1)和方法(2)过于简单、手动化;方法(3)建立在其他模型转换方法上,未能提供完整的解决方案;方法(4)有现成的工具,但偏向于模型到代码的转换。

针对 Struts、Spring 和 Hibernate 的整合框架(SSH 整合框架)的技术特性,本文提出了基于元模型和 UML Profile 相结合的模型转换方法。其思想是先抽象出平台无关模型 PIM 和平台相关模型 PSM 的元模型,然后制定元模型间的转换规则,间接实现 PIM 到 PSM 的模型转换。而在建立 PIM 和 PSM 的元模型时,UML 并不能完整地描述出针对特定平台或技术的模型信息,但 UML 自身具备的扩展机制 UML Profile,为构建 MDA 的各种特殊模型提供了有力的支持,因此本文借助 UML Profile 来建立 PIM 和 PSM 的元模型。其转换过程如图 1 所示。

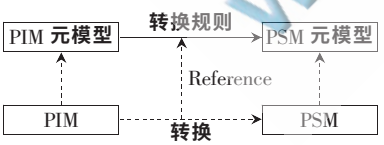

图 1 基于元模型的模型转换

2 面向 J2EE 主流框架的模型转换

根据上文提出的模型转换方法,首先建立 PIM 和 PSM 的元模型,然后分析元模型间的映射关系,建立具体的转换规则,间接实现 PIM 到 PSM 的模型转换。

2.1 建立 PIM 元模型

建立 PIM 元模型应充分考虑 PSM 所属平台的特性,确定使用何种建模技术来建立 PIM 以及建立一个能与 PSM 元模型相呼应的 PIM 元模型,以便制定模型转换规则。当前基于动态行为建模的模型很少应用到 MDA 中,且这方面的模型转换技术还不够成熟,因此,本文采用静态结构的类图对 SSH 框架建模,对于 Struts 框架的建模,本文忽略其视图层,对控制器层和模型层采用类图表示。

在使用静态的类图结构来描述 PIM 时,往往是用描述业务的领域对象模型来表示 PIM,但是一个系统的领域对象模型不足以完整描述系统的总体架构,因此需要进一步对 PIM 模型进行精化,使其既能描述领域对象间的关系又能描述系统的总体架构。

SSH 整合框架是一种多层架构模式,可分为表示层、控制器层、业务逻辑层、实体层、数据持久层、数据库层,这种 N 层架构模式实际上是对经典的 Web 应用软件体系结构的扩展,也可以说是一个适合于 SSH 整合框架的 Web 应用层次结构。同时这种分层结构又是一种广泛应用的软件体系结构,不涉及到特定的技术或与平台相关。因此,本文构建出一个与平台无关又能适合于 SSH 整合框架的 Web 应用层次结构,以此来弥补领域对象模型的不足,展现系统的总体架构,这也是对 PIM 的精化。

本文把这种 Web 应用层次结构的模型作为 PIM 元模型。利用 UML Profile 来建立 Web 应用层次结构模型如图 2 所示。

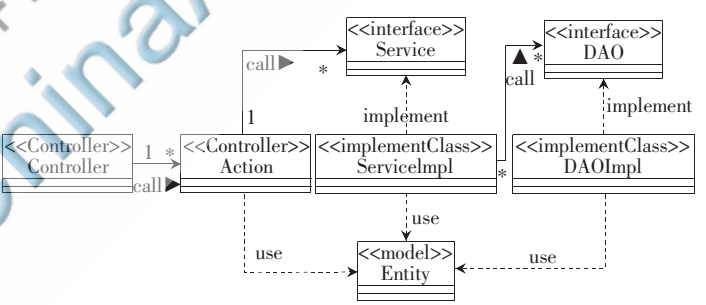

图 2 Web 应用层次结构

图 2 中,Controller 是控制器,负责将视图层发过来的用户请求委派给相应的 Action;Action 调用需要的 Service 类来实现业务逻辑功能;DAO 则提供数据访问功能;Entity 是实体类,与 Action、ServiceImpl、DAOImpl 是依赖关系,这三个类可以声明实体类的对象,或接收实体类对象参数,因此它们之间是使用依赖的关系。

在进行 PIM 到 PSM 的模型转换时,考虑到 PIM 的平台独立性,假如不对 PIM 的元模型进行扩展,则有可能造成 PIM 与 PSM 元模型元素间的映射关系复杂化等情况,在模型转换过程中也将使得转换不够明朗,最终影响模型转换的效果。因此,本文采用了 UML Profile 对 Web 应用层次进行了相应的扩展。表 1 列出了层次结构中的构造型,表 2 列出了构造型的属性值。

表 1 构造型

构造型	对应 MOF 中的元类	说明
<<Controller>>	Class	控制器
<<Action>>	Class	控制器
<<Service>>	Interface	业务逻辑层服务类的接口
<<ServiceImpl>>	Class	服务类接口的具体实现类
<<DAO>>	Interface	数据访问类接口
<<DAOImpl>>	Class	数据访问层接口的具体实现类
<<Entity>>	Class	实体类,数据库关系表映射成对象类
<<Action_Service>>	Association	描述控制器调用对应的业务逻辑
<<ServiceImpl_DAO>>	Association	ServiceImpl 类调用数据访问接口 DAO
<<Controller_Action>>	Association	Controller 将请求分发给对应 Action
<<EntityRelation>>	Association	实体类 Entity 之间的关联关系
<<PrimaryKey>>	Property	实体类的关键字,对应关系表的主键

表 2 构造型的属性值

属性值	类型	所属构造型	说明
aopType	Enumerated	<<Action>>、<<ServiceImpl>>、<<DAOImpl>>	利用切面类给相应的类添加 AOP 操作种类:none,log 等
advice	Enumerated	<<Action>>、<<ServiceImpl>>、<<DAOImpl>>	指明切面逻辑在切入点位置 around,before,after
pointCut	String	<<Action>>、<<ServiceImpl>>、<<DAOImpl>>	使用 AOP 对业务类等添加事务、日志等的切入点

2.2 建立 PSM 元模型

对抽象 SSH 整合框架的元模型,本文先抽象出各个框架的元模型,然后对这些元模型进行整合以得到 SSH 整合框架的准元模型。下面对 Spring、Hibernate 两个框架的元模型抽取做具体的研究,而对 Struts 框架,前面已经忽略掉了对视图层的处理,在此也不对 Struts 的核心 MVC 进行元模型抽取,而是简单地使用 Web 应用层次结构来表达控制器与业务逻辑的关系。

2.2.1 抽象 Spring 框架的元模型

Spring 框架的两大核心是:面向切面编程(AOP)和控制反转(IoC)。其核心在 Web 应用开发中起着举足轻重的作用。

(1)抽象 Spring IoC 的元模型

在 IoC 中,Spring 容器把所有需要管理的类声明成 bean,一个 bean 包括 id、class 和 properties 等。抽象出的 Spring IoC 的元模型如图 3 所示。

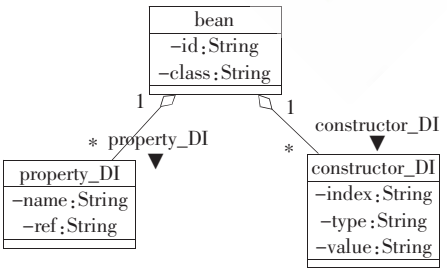

图 3 IoC 元模型

(2)抽象 Spring AOP 的元模型

AOP 的提出是软件开发史的一次重大创新,它用面向切面编程的思想有效地解决了横切关注点问题。AOP 的原理如图 4 所示。图中 AOP 在类 A、类 B 等的方法上加入一个切面,并把一个可以声明成 bean 的切面类织入到切面中,当程序执行到切面的时候先执行 Aspect 对象再往下执行。切面类可以为记录日志、错误处理、数据验证等提供服务。

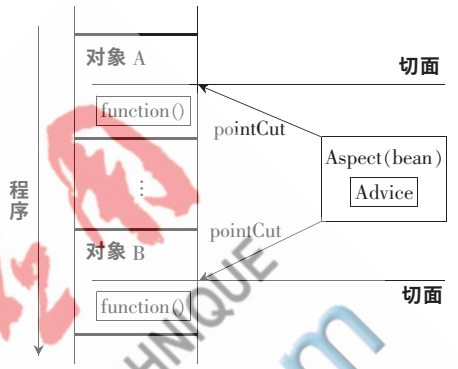

图 4 AOP 原理

根据图 4 的 AOP 原理,对照应用开发,抽象出 AOP 的元模型如图 5 所示。

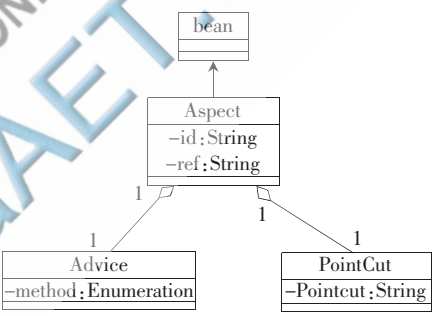

图 5 AOP 元模型

2.2.2 抽象 Hibernate 框架的元模型

Hibernate 框架在业务逻辑层中又分出了数据持久层,数据持久层除了实现对象关系的映射,还封装了底层数据库操作类,为程序开发带来了很好的灵活性,也避免了数据库表的直接暴露。在抽象 Hibernate 框架的元模型时,为每一个实体类 Entity 创建一个对应的数据访问接口 EntityDAO,这样既使得持久层的结构清晰,又能与业务逻辑层的服务接口很好地衔接。对 Hibernate 框架抽象出的元模型如图 6 所示。

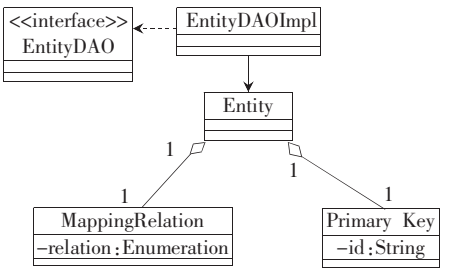

图 6 Hibernate 框架元模型

2.2.3 SSH 整合框架的 PSM 元模型

在抽象出各个框架元模型的基础上，对这三个框架的元模型进行整合得到如图 7 所示的 SSH 整合框架的 PSM 元模型。

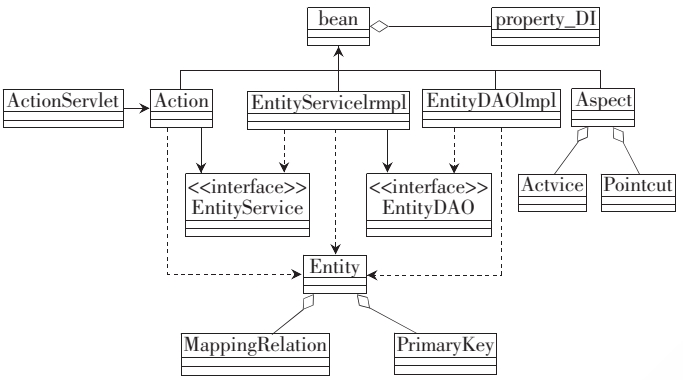

图 7 SSH 整合框架的 PSM 元模型

该 SSH 整合框架元模型不但串联了各个框架的元模型，同时也融合了 Web 应用的层次结构，为 PIM 到 PSM 的模型转换提供了极大的便利。在对 Spring 框架元模型整合中，控制器类、业务逻辑类、数据访问类、面向切面编程类都继承 bean 类，亦即这些类都可以声明成 bean 被 IoC 容器来管理。对 Hibernate 框架元模型的整合，主要是对业务逻辑层和数据持久层的整合。

SSH 整合框架元模型同样采用了 UML 的扩展机制 UML Profile 进行了相应的扩展。表 3 列出了该元模型的构造型。

表 3 PSM 元模型的构造型

构造型	对应 MOF 中的元类	说明
<<bean>>	Class	Spring IoC 容器管理的对象
<<ActionServlet>>	Class	控制器类
<<Action>>	Class	控制器类
<<EntityService>>	Interface	业务逻辑层服务类接口
<<EntityServiceImpl>>	Class	业务逻辑层服务类接口的实现类
<<EntityDAO>>	Interface	数据访问接口
<<EntityDAOImpl>>	Class	数据访问接口实现类
<<Entity>>	Class	实体类
<<Property_DI>>	Class	为 IoC 的属性注入提供服务
<<Aspect>>	Class	AOP 中的切面类
<<Advice>>	Class	AOP 中的横切逻辑
<<Poincut>>	Class	AOP 中的切入点
<<MappingRelation>>	Class	实体类中的关联映射
<<Entity_ID>>	Class	实体类中的主键

2.3 PIM 到 PSM 的模型转换

根据本文提出的模型转换方法，模型转换过程从一个初始的领域对象模型开始，根据定义的精化规则对其求精形成准 PIM 模型，然后根据模型转换规则，转换成相应的 PSM 模型。

2.3.1 PIM 的精化

要实现 PIM 到 PSM 的转换，首先需要建立适应 Web 应用层次结构的 PIM 模型，而在实际应用中，一开始往往建立的是领域对象模型，不能很好地满足 Web 应用层次结构。因此可以参考系统的其他 UML 图(如用例图对这一初始 PIM 进行精化形成准 PIM)，使其能遵循 PIM 元模型的规范。本文对 PIM 的精化定义了如下的精化规则：

(1)为每个领域对象生成一个同名的实体类，领域对象的属性对应实体类的属性，关键字对应实体类的 ID，领域对象间的关联关系对应实体类的关联。

(2)为每个实体类生成一个对应的数据访问接口 EntityDAO 和实现类 EntityDAOImpl，并添加需要的方法，EntityDAOImpl 能调用对应的实体类。

(3)在业务逻辑层中，为每个实体类根据需要建立相应的 EntityService 接口和其实现类 EntityServiceImpl 或其他的服务类，添加需要的属性和方法，EntityServiceImpl 类可以访问实体类 Entity 和数据访问类 EntityDAO。

(4)结合其他 UML 图生成控制器类 Controller 和 Action，Action 可以调用需要的服务类。

2.3.2 PIM 到 PSM 的模型转换规则

PIM 到 PSM 的模型转换是 MDA 应用开发的核心，模型转换规则则是 PIM 到 PSM 的转换的核心，可以说模型转换规则是 MDA 应用开发核心的核心，足以见得模型转换规则的重要性。下面定义了具体的模型转换的转换规则：

(1)<<Entity>>转换到 PSM

生成与<<Entity>>相同的实体类，将领域对象间的关联<<EntityRelation>>映射成实体类的关联映射<<MappingRelation>>，将领域对象的主键<<primaryKey>>映射到<<Entity_ID>>。主要转换规则如下：

R1?{(pim.Entity,psm)|?(x? pim.Entity)? ?(y? psm.Entity)(y.classname=x.classname)??(x? pim.Entity.attributes)? ?(y? psm.Entity.attributes)(y.name=x.name?y.type=x.type?y.visibily=private)??(z? psm.Entity.methods)(z.visibility=privatey?z.name?'get'?x.name?z.type?x.type?z.parameterlist?[])??(x? pim.Entity.primary? ?(y? psm.Entity.id)(y.value=x.value)}

R2={(pim.EntityRelation,psm)|?(x? pim.EntityRelation)? ?(y? psm.Entity)(y.classname=x.relation_source.name?y.attributes.name=x.relation_target.name?y.attributes.visibily=private?y.attributes.type=x.relation_target.name? y.mappingrelation.name=x.classname? y.mappingrelation.value=x.relation_souce.value?'to'+x.relation_target.value)}

(2)<<DAOImpl>>、<<ServiceImpl>>、<<Action>>、<<Aspect>>转换到 PSM

将 PIM 中的每个<<DAOImpl>>、<<ServiceImpl>>、<<Action>>、<<Aspect>>转换成对应的<<bean>>，主要转换规则如下：

R1?{(pim.DAOImpl,psm)|?(x? pim.DAOImpl)? ?(u? psm.bean)(u.classname=x.classname-'Impl'?u.attributes=x.attributes?u.methods=x.methods??(y? psm.bean.id)(y.value=x.classname-'Impl')??(z? psm.bean.class)(z.value? x.classname)}

R2={(pim.Action,psm)|?(x? pim.Action)? ?(u? psm.bean)(u.classname=x.classname? u.attributes=x.attributes? u.methods=x.methods)??(y? psm.bean.id)(y.value=x.classname)??(z? psm.bean.class)(z.value=x.classname)}

R3={(pim.Aspect,psm)|?(x? pim.Aspect)? ?(u? psm.bean)(u.classname=x.classname? u.attributes=x.attributes? u.methods=x.methods)??(y? psm.bean.id)(y.value=x.classname)??(z? psm.bean.class)(z.value=x.classname)}

R4={(pim.ServiceImpl,psm)|?(x? pim.ServiceImpl)? ?(u? psm.bean)(u.classname? x.classname-'Impl'?u.attributes? x.attributes? u.methods? x.methods??(y? psm.bean.id)(y.value=x.classname?'Impl')??(z? psm.bean.class)(z.value=x.classname)}

（3）<<Action_Service>>、<<ServiceImpl_DAO>>转换到 PSM

将每个<<Action_Service>>、<<ServiceImpl_DAO>>关联映射成 PSM 中的<<Property_DI>>，根据关联的命名来定义规则，如<<ServiceImpl_DAO>>关联，是将 DAO bean 注入到 ServiceImpl bean 中，则主要转换规则如下：

```
R1={(pim.Action_Service,psm)|?(x? pim.Action_Service)? ?(y? psm.bean)(y.classname=x.sourceBean.value? ?(z? y.Property_DI)(z.classname=x.classname? z.attributes.name='name'? z.attributes.type='String'? z.attributes.value=x.sourceBeanname? z.attributes.name='ref'? z.attributes.type='String'? z.attributes.value=x.sourceBeanname))}
R2={(pim.ServiceImpl_DAO,psm)|?(x? pim.ServiceImpl_DAO)? ?(y? psm.bean)(y.classname=x.sourceBean.value? ?(z? y.Property_DI)(z.classname=x.classname? z.attributes.name='name'? z.attributes.type='String'? z.attributes.value=x.sourceBeanname? z.attributes.name='ref'? z.attributes.type='String'? z.attributes.value=x.sourceBeanname))}
```

（4）对<<Action>>、<<ServiceImpl>>、<<DAOImpl>>实现到 PSM（AOP）中的转换

根据<<Action>>、<<ServiceImpl>>、<<DAOImpl>>构造属性<<aopType>>的值，选择对应的 PSM 中的 Aspect 类。属性<<advice>>、<<pointCut>>对应 PSM 中的<<Advice>>、<<PointCut>>，本文选取<<ServiceImpl>>实现到 PSM（AOP）中的转换，其他的两个转换规则类似。主要转换规则如下：

```
R1={(pim.ServiceImpl,psm)|?(x? pim.ServiceImpl)? ?(y? psm.Aspect)(y.classname? x.aopType.enum+'Aspect'? ?(z? y.id)(z.value=x.aopType.enum)? ?(u? psm.y.ref)(u.value=x.classname)? ?(h? y.Advice)(h.attributes.name='advice'? h.attributes.type='String'? h.attributes.value=x.advice.enum)? ?(l? y.PointCut)(l.attributes.name='pointCut'? h.attributes.type='String'? h.attributes.value=x.pointCut.value))}
```

本文分析了几种主流的模型转换方法，但这些方法并未对 J2EE 主流框架提供有效的支持，而本文提出的基于元模型和 UML Profile 相结合的模型转换方法，通过建立 PIM 元模型和 PSM 元模型，以及制定元模型间的转换规则，间接实现了 PIM 到 PSM 的模型转换。

参考文献

[1] MILLER J，MUKER J I. MDA guide version 1.0 [M]. Object Management Group，2003：3-7.

[2] FRANKEL D S.应用 MDA[M].鲍志云译.北京：人民邮电出版社，2003：138-142.

[3] OLDEVIK J，SOLBERG A. Framework for model transformation and code generation. In：Enterprise Distributed Object Computing Conference [C]. EDOC′02. Proceedings. Sixth International，2002：181-189.

[4] 张德芬，李师贤，古思山.MDA 中的模型转换技术综述[J].计算机科学，2006，33（10）：228-230.

[5] 张天，张岩.基于 MDA 的设计模式建模与模型转换[J].软件学报，2008，19（9）：2203-2217.

[6] 王永涛，刘勇.基于 MDA 的模型转换研究[J].计算机工程，2011，37（16）：84-87.

（收稿日期：2012-04-03）

作者简介：

谢培基，男，1986 年生，硕士，主要研究方向：软件工程。

余金山，男，1952 年生，教授，硕士生导师，主要研究方向：软件工程。