◎博士论坛◎

基于领域驱动设计的应用系统模型

李 引[1,2],袁 峰[1,2]
LI Yin[1,2], YUAN Feng[1,2]

1.中国科学院软件技术研究所 互联网软件技术实验室,北京 100190
2.广州中国科学院软件应用技术研究所,广州 511458
1.Laboratory for Internet Software Technologies, Institute of Software of Chinese Academy of Sciences, Beijing 100190, China
2.Institute of Software Application Technology, Guangzhou & Chinese Academy of Sciences, Guangzhou 511458, China

LI Yin, YUAN Feng. Application model based on domain-driven design. Computer Engineering and Applications, 2013, 49(16):1-8.

Abstract: Domain-Driven Design(DDD) is introduced by Evans E. to track complexity in the software, which has been proved effectively in practical.However, it lacks of fine-grained definition of some details and depends on the high-quality of the developers and so on.Based on the analysis of a number of business systems, the common operations of business object are abstracted, and an application model of DDD is proposed to guide the system design and development.Meanwhile, a framework is proposed to encapsulate common properties and operations of business system.In practical, this framework has been proved to assist the software development and improve the efficiency and reduce effort.

Key words: Domain-Driven Design(DDD); application model; development framework

摘 要: 领域驱动设计(Domain-Driven Design,DDD)是Evans提出来的用来处理软件系统核心复杂性的方法。该方法的有效性在实践中得到证明,但是方法在细节上存在不够清晰、对设计人员素质要求高等问题。在对大量业务系统进行分析和实践的基础上,对业务对象的公共操作进行了抽象,提出了基于DDD的应用系统模型来指导系统设计和开发。研发了开发框架对业务系统中通用的属性和操作进行了封装。实际项目中的应用证明了该框架能够辅助进行系统设计开发,提高软件开发效率和减少缺陷。

关键词: 领域驱动设计;应用系统模型;开发框架
文献标志码: A **中图分类号:** TP311 doi:10.3778/j.issn.1002-8331.1303-0192

1 引言

软件工程经过40多年的发展,提出了很多新的理论、方法、语言、框架等,从软件开发瀑布模型到螺旋模型,再到敏捷模型,从汇编语言到C/C++/VB,再到Java/C#以及动态语言,从面向过程编程到面向对象编程再到面向构件编程等等。这些都是为了更好地解决越来越复杂的业务环境的问题,使得我们开发的软件能够更好地满足客户需求,同时缩短开发周期,减少开发成本。

领域驱动设计[1](DDD)方法的提出为解决该问题提供了一个很好的方案。简单说,DDD是一种指导面向复杂领域的软件开发项目的思维方式,该方法将领域模型(Domain model)[2-3]作为业务分析设计的核心,通过实体(Entity)、值对象(Value Object)、服务(Service)、仓储(Repository)、聚合根(Aggregate)等模型元素来进行业务建模,并将这些模型元素分配到各个层中,能够有效地实现对业务逻辑的封装,保证业务模块高内聚低耦合的特性,使系统能够更好地进行维护和扩展,实现敏捷开发的目的。

采用DDD思想建立的领域建模具有内聚性高、扩展性好、易重构等特点。文[4-6]将DDD方法应用到了挪威国家石油公司的石油交易供应链系统中,再与之前的遗留系统进行对比发现,采用DDD构建的新系统架构明显优于遗留架构,系统的扩展性、性能和代码质量有了明显提高。文

基金项目: 广州市科技计划项目(No.201200000039)。
作者简介: 李引(1981—),男,博士,副研究员,研究领域为软件构件,云计算;袁峰(1977—),男,博士,硕士生导师,副研究员,研究领域为模型驱动,云计算,智慧城市。E-mail:liyin@gz.iscas.ac.cn
收稿日期: 2013-03-13 **修回日期:** 2013-05-10 **文章编号:** 1002-8331(2013)16-0001-08
CNKI出版日期:2013-05-15 http://www.cnki.net/kcms/detail/11.2127.TP.20130515.1015.003.html

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

[7-9]等研究将DDD方法应用到了具体业务领域的项目开发中,也取得比较好的效果。基于这些项目经验分享,将DDD应用到了实际系统的设计开发中,在带来业务架构改进的同时,也发现一些问题:(1)DDD方法需要对业务领域进行全局考虑并进行整体设计,这对分析人员素质要求很高[3];(2)模型元素存放的层次、元素之间的依赖关系存在很多选择,不同人有不同看法,难以统一;(3)公共的业务没有按照DDD进行抽象和封装,在设计开发时出现多种风格。这些问题使得DDD在实际的项目中应用还不太广泛。

针对这些问题,基于对大量企业级业务系统的深入分析,采用DDD方法来对业务对象的公共的属性和操作进行抽象封装,提出了基于DDD的应用系统模型DDDAppModel。该模型对DDD的分层和各模型元素进行了定义,并建立了业务领域的公共属性和操作(比如工作流操作)的领域模型,用来规范和指导业务分析建模。DDDAppModel已经被实现为一套软件开发框架,并在十余个实际项目中使用,业务代码总量超过百万行。从实际项目的反馈来看,DDDAppModel能够很好地引导业务领域模型的建立,其封装的公共属性和操作能够统一开发风格,提高设计和开发效率,降低系统产生的缺陷。

本文在第二章对业务对象的公共操作进行分析并建立统一的模型。基于该模型,在第三章提出DDDAppModel。在第四章介绍DDDAppModel的实现框架和相应的实例分析。最后进行总结。

2 业务对象公共操作模型

本文不针对基础框架的共性问题进行讨论,比如邮件、打印功能,而是将以业务领域为中心,研究其中的公共属性和操作。针对这些共性问题,利用DDD方法构建模型。

对于业务系统,业务对象是核心元素,按照DDD思想,业务对象是封装业务规则的主要元素之一,因此对其进行建模将对实际应用系统设计开发具有积极作用。

业务对象主要具有增删改查等四类基本操作,进一步的,还有一些公共的复杂操作,包括工作流处理、历史记录操作和保存草稿等。从实际项目中发现,绝大多数业务对象都具有描述“状态”的属性,因此状态将作为业务对象的属性之一。

定义1(业务对象) BizObject=<properties,operations>,其中properties是业务对象的属性集合,表示为n元组$<state, p_1, p_2, \cdots, p_{n-1}>$,其中state的取值为{NORMAL,DRAFT,DELETED,HISTORY,PENDING},分别表示正常状态、草稿状态、被删除状态、历史状态和处理中状态。operations是该对象具有的行为集合,表示为m元组$<o_1, o_2, \cdots, o_m>$。

该state不同于Hibernate中的POJO[10]和.NET中DataRow[11]的生命周期模型,该state不是从系统运行的角度考虑对象的创建、加载、刷新、销毁等问题,而是从更高的业务层面对业务领域的实体状态的公共变化进行的建模。

NULL被定义为特殊的业务对象,表示不存在。

定义2(业务操作) Operation=<input,output,automicOperations,∏,C>,其中input表示输入,output表示操作输出,automicOperations表示该操作包含的原子操作集合。∏为[input,automicOperations]→output,表示input经过原子操作automicOperations之后输出output。C表示执行该操作后满足的约束。基于该形式,可以将业务对象的操作进行定义。

2.1 基本操作模型

基本操作主要包含增删改查操作。

定义3(新建操作) New=<{NULL},{object},{edit,persist},{NULL→object},{object.state=NORMAL}>。其中edit表示在内存中的编辑操作,persist表示将对象状态设置为NORMAL并进行持久化,object.state=NORMAL表示新建操作完成后对象的状态处于NORMAL。

定义4(修改操作) Mod=<{object},{newObject},{edit,persist},{object→newObject},{newObject.state=NORMAL}>。

对于删除操作分为真删除和假删除,真删除为真正地将其从持久化介质中移除,而假删除一般通过设置对象的状态,在查找时通过状态过滤来实现。

定义5(真删除操作) RealDel=<{object},{NULL},{delete},{object→NULL},{}>。其中delete表示执行持久化的永久删除操作。

定义6(假删除操作) FakeDel=<object,delObject,{edit,persist},{object→delObject},{delObject.state=DELETED}>。从该定义可以看出,假删除是一种特殊的修改操作,仅仅是修改状态为DELETED。

定义7(查找操作) Find=<{conditions},{objects},{select},{conditions→objects},{object.state=NORMAL}>。在该定义中,查找操作的返回对象的状态限定为NORMAL,这使得定义6能够实现业务对象的假删除功能。

2.2 保存草稿操作模型

将业务对象保存为草稿是一种比较常见的操作,在对象编辑之后执行。草稿中的对象与正常的对象一般采用状态来进行区分。

定义8(新建并保存草稿操作) NewSave=<{NULL},{object},{edit,save},{NULL→object},{object.state=DRAFT && object.createdBy=someone}>。新建保存草稿操作与新建操作类似,只是其中业务对象的状态需要设置为DRAFT。从业务逻辑上来说,新建保存草稿的对象是不能被其他人所见的,只有创建者someone才能看到并继续打开进行编辑后存储。根据定义7可知,一般的查找操作是无法获取新建草稿对象的,保证了新建草稿对象的隐蔽性。

对于修改后保存的草稿,应该是谁保存的草稿谁能看到。同一个业务对象,不同的人可以生成不同的草稿对象,也就是一个业务对象可能对应多个草稿对象。

定义9(修改并保存草稿操作) ModSave=<{object},{$object_{newObject}$},{edit,save&relate},{NULL→newObject,object ⊥ newObject→$object_{newObject}$},{object.state=NORMAL,newObject.state=DRAFT&newObject.modifiedBy=someone}>。在该操作完成后,将存在新旧两个对象,老对象是不能覆盖掉的。为了草稿对象能被创建者查看到,需要将老对象与草稿对

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

象建立关联,采用 $object_{newObject}$ 表示 newObject 作为 object 草稿的含义;object⊥newObject 表示建立以 object 为主,newObject 为辅的对象关联。

查找草稿需要与正常对象进行区分,状态就作为该操作的过滤属性。

定义 10(查找草稿操作) FindDraft=<{object.createdBy=someone||object.modifiedBy=someone&Find.input},{objects},{select},{conditions→objects},{ object.state=DRAFT&object.createdBy&object.state≠DELETED}>,其中如果是查找新建对象则需要输入 object.createdBy 创建人作为过滤,如果是查找修改对象则输入 object.modifiedBy 修改人作为过滤条件。

2.3　工作流处理操作模型

工作流处理可以认为是对业务对象操作之上的复合操作,比如一个请假单审批流程,可以看成是新建请假单业务对象的过程中插入了流程处理操作,当审批通过之后,请假单对象创建成功,当审批失败之后,请假单对象创建失败。

定义 11(工作流处理) Process=<nodes,transitions,operations,pstates,∑>,其中 nodes 是流程的节点处理节点集合,包括开始节点。transitions 是预先设定的节点之间的流程路径,可以包括串行、并行、决策等。operations={start,accept,reject}是流程处理的操作,start 是第一个节点中发起流程操作,accept 和 reject 是后续节点中的处理操作,accept 是根据 transitions 继续推进流程,reject 是终止流程。pstates={SUCCESSS,FAIL,DEALING},分别表示流程成功、失败和处理中三个状态。∑是不同节点在不同的操作下,能够使得流程处于不同的状态,即[nodes,transitions,operations]→pstates。

与 BPEL[12-13] 等定义不同,本文对工作流的定义不侧重于流程的节点及节点间转移路径,而是侧重于工作流对业务对象操作的驱动,operations 中定义的操作能够代表流程的其他处理对业务对象的影响,因此不讨论其他的流程处理行为。在定义 11 中没有与处理对象进行关联,因此将其与业务对象处理绑定后,业务对象的操作能够与工作流进行绑定,主要包括新建流程 NewProcess、修改流程 ModProcess 和删除流程 DelProcess 操作,也即 NewProcess.operations={StartNew,AcceptNew,RejectNew},ModProcess.operations={StartMod,AcceptMod,RejectMod},DelProcess.operations={StartDel,AcceptDel,RejectDel},下面分别进行定义。

定义 12(发起新建对象流程) StartNew=<{NULL&$p.t_0$},{pendingObject&$p.t_1$},{edit,persist&start},{NULL→pendingObject,$p.t_0$→$p.t_1$},{pendingObject.state=PENDING&&$p.t_1$≠$p.t_{end}$}>,其中 $p.t_0$ 表示处于流程发起节点,$p.t_1$ 表示处于第一个节点,$p.t_{end}$ 表示流程最后一个节点;persist&start 表示在进行对象持久化的同时发起流程,新建对象的状态为 PENDING,表示还在流程处理中。

定义 13(同意并结束新建对象流程) AcceptNew=<{pendingObject&$p.t_m$},{pendingObject&$p.t_{end}$},{persist&accept},{pendingObject.state→NORMAL,$p.t_m$→$p.t_{end}$},{pendingObject.state=NORMAL&&$p.t_{end}$=true}>,当处于最后一步时,流程自动执行 accept 操作后流转到 $p.t_{end}$ 最后一个节点,同时 pendingObject 将变为正常状态 NORMAL。

定义 14(拒绝并结束新建对象流程) RejectNew=<{pendingObject},{object},{persist&reject},{pendingObject.state→FAIL },{pendingObject.state=FAIL}>,当处理人拒绝流程时,将执行 reject 操作,流程失败,并且 pendingObject 对象的状态变为失败 FAIL。

在工作流中处理修改对象与新建对象有一定区别,在于修改对象还需要维持老对象的可访问性,修改之后的对象值不能直接覆盖老对象,而是需要以某种方式进行临时存储,同时建立于老对象之间的关联。

定义 15(发起修改对象流程) StartMod=<{object&$p.t_0$},{$object_{pendingObject}$&$p.t_1$},{edit,persist&start},{object⊥pendingObject→$object_{pendingObject}$,$p.t_0$→$p.t_1$},{$object_{pendingObject}$.State=PENDING,pendingObject.state = PENDING && $p.t_1 \neq p.t_{end}$} > ,其中 $object_{pendingObject}$ 表示 pendingObject 作为 object 在修改流程处理中的临时对象,并且进行关联。

定义 16(同意并结束修改对象流程) AcceptMod=<{$object_{pendingObject}$&$p.t_m$},{object&$p.t_{end}$},{persist&accept},{$object_{pendingObject}$↑object},{object.state = NORMAL}>,其中 $object_{pendingObject}$↑object 表示将 object 替换为临时对象 pendingObject,并且设置状态为 NORMAL。

定义 17(拒绝并结束修改对象流程) RejectMod=<{$object_{pendingObject}$&$p.t_m$},{object&$p.t_{end}$},{reject},{$object_{pendingObject}$↓object},{object.state=NORMAL,pendingObject.state=FAIL}>,其中 $object_{pendingObject}$↓object 表示丢弃 pendingObject 的修改,并且 object 的状态变为 NORMAL,pendingObject 的状态变为 FAIL。这里不能直接删除 pendingObject 对象,因为用户还可以再基于 pendingObject 修改后发起新的流程。

在流程处理中,如果还没有结束,则业务对象的状态不会改变,只是将流程推进到下一个节点。

定义 18(同意并推进流程) Proceed = <{pendingObject&$p.t_m$},{$p.t_{m+1}$},{proceed},{$p.t_m$→$p.t_{m+1}$},{pendingObject.state=PENDING&&$p.t_{m+1}$≠$p.t_{end}$}>,对于新建和修改操作来说,同意并推进到下一步都对业务对象没有影响。

2.4　历史记录操作模型

历史记录是将变化前的业务对象生成快照并进行持久化。对于新建操作,没有变化前的对象,因此在新建时不需要进行记录;对于修改操作,历史记录需要将新老对象变化的属性进行存储;对删除操作,需要将整个对象进行存储。因此,历史记录操作是增删改操作的后续操作,只需要传入要记录的对象作为参数即可。

定义 19(记录变化历史) HistoryRecord=<{object},{$object_{history}$},{persist},{object→$object_{history}$},{$object_{history}$.state=HISTORY}>,其中 $object_{history}$ 表示对象 object 变为历史对象。

将修改操作与历史记录操作综合在一起,则表示为 ModHistoryRecord=<{object},{newObject,$object_{history}$},{edit,persist},{object→newObject,object→$object_{history}$},{newObject.state = NORMAL,object.state = HISTORY}>,其中

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

$object_{history}$表示对象object变为历史对象。

定义 20（查找历史记录）FindHistory=<{conditions},{$objects_{history}$},{select},{conditions→$objects_{history}$},{object.state=HISTORY}>。其中conditions中需要指定查找的对象类型。

2.5 业务操作模型统一

将上述业务操作进行汇总并分析后，发现业务对象需要一些基础的属性来驱动业务操作，同时一些操作是可以进行统一处理的。

2.5.1 公共属性的统一

（1）state属性是所有的操作都能涉及到的，其需要抽取出来作为基础公共属性。

（2）如果需要支持草稿操作，那么createdBy和modifiedBy属性也是需要的，用来根据草稿创建人进行过滤。

（3）对于草稿记录和工作流处理操作，为了保证原对象的持续可查看，草稿对象和工作流处理对象都需要进行临时记录，该临时记录需要与原对象建立关联关系，也就是用定义9中原子操作中的relate来进行连接。因此，临时对象的记录也将作为一个公共属性。

2.5.2 业务操作统一

（1）新建操作和修改操作所具有的原子操作是一样的，唯一不同的是新建操作没有老对象作为输入。将定义3与定义4以谓词逻辑的方式进行表达如下：

$$edit(NULL,o)\wedge persist(NULL,o)\rightarrow Normal(o) \quad (1)$$

$$edit(o_1,o_2)\wedge persist(o_1,o_2)\rightarrow Normal(o_2)\wedge NotExisted(o_1) \quad (2)$$

其中$edit(X,Y)$表示将X进行编辑后变成Y，$persist(X,Y)$表示将Y持久化到X的位置，$Normal(X)$表示X的状态为NORMAL，$NotExisted(X)$表示X不存在。

从以上两个公式可以看出，新建操作与修改操作是可以统一到一起的，NULL对象作为特殊的对象看，可以采用式（2）来统一表达。

（2）工作流新建发起操作中包含操作符persist&start，与新建操作中的persist不同的是引入了流程的发起操作start，导致最终输出的对象状态为PENDING，而不是NORMAL。与此同时，当流程执行结束时，执行同意处理操作中包含persist&accept，使得输出对象从状态PENDING变为了NORMAL。

根据定义12，新建发起流程表示为：

$$edit(NULL,o)\wedge startAndPersist(NULL,o)\wedge stay(p,t_0)\wedge trans(t_0,t_1)\rightarrow Pending(o)\wedge stay(p,t_1)\wedge\neg end(p,t_1) \quad (3)$$

其中$startAndPersist(NULL,o)$表示新建时发起工作流操作，对应persist&start操作符，$stay(p,t)$表示处于流程p的节点t，$trans(t_0,t_1)$表示流程节点从t_0转移到t_1，$Pending(X)$表示X的状态为PENDING，$end(p,t)$表示节点t是流程p的结束节点。

从工作流推进的逻辑可知：

$$stay(p,t_0)\wedge trans(t_0,t_1)\rightarrow stay(p,t_1)\wedge\neg end(p,t_1) \quad (4)$$

因此式（3）可化简为：

$$edit(NULL,o)\wedge startAndPersist(NULL,o)\rightarrow Pending(o) \quad (5)$$

根据定义13，同意并结束新建对象流程表示为：

$$Pending(o_1)\wedge acceptAndPersist(o_1,o_2)\wedge stay(p,t_m)\wedge trans(t_m,t_{end})\rightarrow Normal(o_2)\wedge stay(p,t_{end})\wedge end(p,t_{end}) \quad (6)$$

其中$acceptAndPersist(o_1,o_2)$表示接受对象$o_1$的修改并且持久化为$o_2$，对应操作符persist&accept。

从工作流推进的逻辑可知：

$$stay(p,t_m)\wedge trans(t_m,t_{end})\rightarrow stay(p,t_{end})\wedge end(p,t_{end}) \quad (7)$$

因此式（6）可化简为：

$$Pending(o_1)\wedge acceptAndPersist(o_1,o_2)\rightarrow Normal(o_2) \quad (8)$$

最后，将式（5）代入式（8），可以推出：

$$edit(NULL,o_1)\wedge startAndPersist(NULL,o_1)\wedge acceptAndPersist(o_1,o_2)\rightarrow Normal(o_2) \quad (9)$$

综合式（1）与式（9）后，可以得出：

$$startAndPersist(NULL,o_1)\wedge acceptAndPersist(o_1,o_2)\Leftrightarrow persist(NULL,o_2) \quad (10)$$

也就是发起新建对象流程与同意并结束新建对象流程等同于新建对象操作。因此，可以将新建操作看成是由工作流发起与工作流处理两个子操作构成，也就是New=StartNew+AcceptNew。由于新建操作与修改操作可以统一，因此修改操作与工作流修改操作可以统一为Mod=StartMod+AcceptMod。

（3）FindDraft和FindHistory操作以Find操作为基础，并通过输出条件state的不同而进行区分，可以统一到Find操作中。

3 应用系统模型DDDAppModel

3.1 领域驱动设计简介

DDDAppModel是基于DDD的一套模型，其建模思想与DDD相符合，因此需要对DDD的核心思想和元素进行简要说明。

DDD是一种模型驱动的方法，领域模型是其核心，从概念上来说，领域模型是由一些模型元素构成，并且采用分层的方式来进行业务的隔离。

领域模型主要由实体、值对象、聚合、仓储、工厂、服务这几类元素构成。实体、值对象和聚合可以是对业务逻辑的封装的主要元素，不但具有数据，还有丰富的业务行为。聚合是一种包含了其他实体或值对象的实体，聚合控制了内部对象的访问，外部如果想要访问内部对象，则必须通过聚合来进行，因为只有这样才能保证聚合内部的不变量约束，也就是业务逻辑约束；仓储代表了资源的储存仓库，可以存放实体、值对象和聚合这三类元素，它对外屏蔽了底层的持久化技术；工厂与设计模式中的工厂是类似的，它负责领域对象的构建；服务分为应用层服务（Application Service）和领域层服务（Domain Service），应用层服务只是一个对外的门面，关注于功能的定义，而不是业务逻辑的实现。领域层服务关注于业务逻辑实现，业务中属于实

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

体、值对象和聚合的逻辑就将封装到其中。

DDD将整个系统分为四层,包括UI层、应用层、领域层和基础设施层。UI层就负责界面的展示,主要采用MVC的模式;应用层定义了系统能做什么,它仅仅是将工作委托给领域对象来完成,是很薄的一层;领域层是系统的核心,封装了系统的所有业务逻辑;基础设施层提供了系统基础服务支持,比如数据访问、网络通信、邮件发生等。

3.2　系统分层

按照领域驱动设计方法,应用系统架构可以分为领域层、应用层、表现层和基础设施层,其中领域层是核心。区别于传统的三层/四层等架构将业务逻辑散布在应用层、数据访问层甚至表现层不同,领域驱动设计的思想将侧重点放在领域层,在该层中采用实体、值对象、服务、工厂等设计元素来对业务逻辑进行封装,避免将业务逻辑暴露给其他层,使得以领域层为中心可以实现与其他层的松耦合,从而能够以最小代价实现对其他层的替换。

定义21(系统分层模型) System=<DomainLayer, ApplicationLayer, InfrastructureLayer, UILayer, H>。其中:DomainLayer=<{Entity, ValueObject, Aggregate, DomainService, Factory, RepositoryInterface}, {InfrastructureLayer}>代表领域层,主要包含实体、值对象、聚合、领域服务、工厂和数据仓储接口这几种元素。

ApplicationLayer = <{AppService}, {DomainLayer, InfrastructureLayer}>代表应用层,主要包含应用服务,依赖于领域层和基础设施层。

InfrastructureLayer = < { RepositoryImplementation}, {DomainLayer}>代表基础设施层,主要包含数据仓储的实现RepositoryImplementation(还包含其他基础设计服务的实现,比如邮件,这一部分不在本文讨论范围),依赖于领域层的数据访问接口和实体。

UILayer=<{Controller}, {ApplicationLayer, DomainLayer}>代表界面层,主要包含界面控制类,依赖于应用层和领域层。

从以上定义可以发现DomainLayer和InfrastructureLayer存在互相依赖的问题,这是因为将数据仓库接口放在DomainLayer,而实现放在InfrastructureLayer造成的。但是实际上不会造成循环依赖,因为在编译时DomainLayer不需要依赖于InfrastructureLayer,DomainLayer只依赖于RepositoryInterface。当运行时,系统会通过IOC(控制反转)技术将RepositoryImplementation注入到DomainLayer中,也就是在运行时DomainLayer依赖于InfrastructureLayer。

H代表各层之间的依赖调用关系,如图1所示。

(1)对于简单的增删改查操作,只需要按照A->B1的调用路径。这一部分的操作是可以通过一些模板方法来实现的。

(2)对于一些较复杂业务逻辑的操作,其业务逻辑主要封装在DomainService和Entity中,所以基本上按照A->B2->C1->D路径执行。比如一个涉及到多个业务对象之间的交互的操作,该逻辑将封装在DomainService和Entity中。

(3)对于一些较复杂的构造逻辑操作,将按照A->B3->C2的路径执行,构造逻辑封装在factory中。比如一个对象的属性由其他对象的属性按照业务逻辑拼接而成。

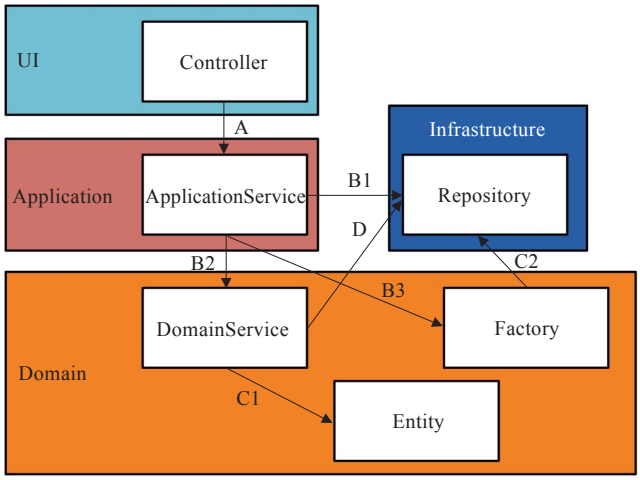

图1　分层调用关系

3.3　实体模型(Entity Model)

实体模型主要由三部分构成,第一部分是公共属性,第二部分是公共操作,第三部分是属性在操作下进行的转化。

定义22(实体模型) Entity=<properties={id, state, tempObject, userRelatedProps, otherProps}, operations={submit, accept, reject, save, startDel, snapshot, otherOperations}, H>。其中properties是实体的属性集,operations是操作集合,H是properties在operations下的转换。

(1)id为该实体的唯一标识。

(2) state = {NORMAL, PENDING, DRAFT, FAIL, DELETED, HISTORY},分别表示正常、处理中、草稿、失败、已删除和历史状态。但是由于需要对一些业务操作进行统一,所以这几个状态是不够的,需要根据操作的类型生成一些新的状态,具体见图2。这里的state只是概念上的模型,最终的实现会有很多种方式,比如不一定是作为该实体的属性,可以作为另外的关联表存在。

(3)tempObject是实体对应的临时对象,可以对应草稿对象和工作流处理中的临时对象。该字段主要针对如何保证对象的修改在还没有确认前,对象的旧值还能继续被查看,新值只有在审批中才能看到对问题进行处理。当该实体操作还未正式生效时,之前实体属性还是保持原状,保证了系统的数据的一致性。同时,当操作最终生效时,tempObject将会把原对象的属性值给替换掉。这里的tempObject只是概念上的模型,最终的实现会有很多种方式。

(4) userRelatedProps={createdBy, createdTime, modifiedBy, modifiedTime}是实体与操作用户相关的元数据。从2.5节的分析,createdBy是实体创建者,modifiedBy是实体修改者,createdTime和modifiedTime表示新建和修改的时间。实体一旦创建,createdBy和createdTime将不会再发生变化,但是修改可能会有很多次,因此modifiedBy和modifiedTime是会发生变化的。

(5) operations={submit, accept, reject, save, startDel, snapshot}是该对象具有的操作集合。根据2.5节,可以将新建/修改操作与工作流的操作统一到一起,新建/修改操作可以拆分为start与accept操作,在实体模型中将start用

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

submit来表示。当对象的accept操作被截取时,也就是只执行了submit操作时,就相当于发起了流程。工作流处理时,当处于最后一个流程节点,同意操作就调用accept使得流程结束,并且对象新建或修改成功。当拒绝操作时,就调用reject,将临时对象抛弃,并且恢复老对象为正常状态。该操作集合主要是设置实体在内存中的公共属性state、userRelatedProps和tempObject。当处于不同状态时,执行相应的操作能够驱动状态的变化,见图2。tempObject属性在submit时进行设置,将绑定临时对象。userRelatedProps属性在accept操作中进行设置,设置创建信息和修改信息。

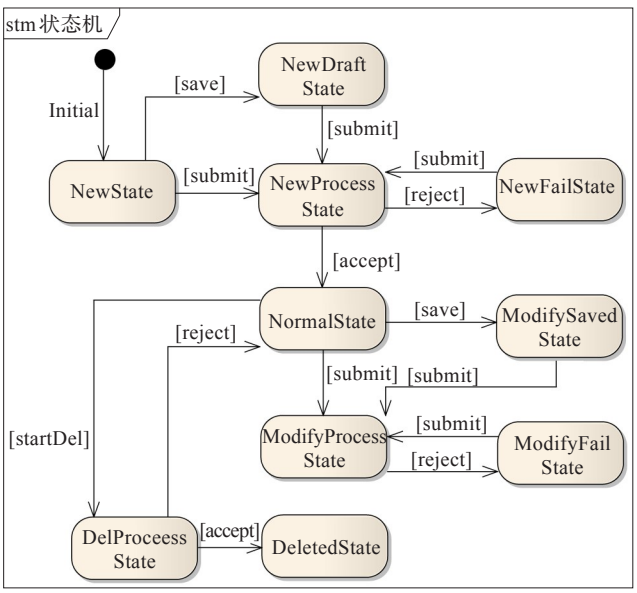

图2　实体状态转移

(6)H:state×operations→state。H是实体在业务操作下进行的状态转换,当处于不同的状态时,能够执行的操作是不同的。

定义23(聚合模型) Aggregate=<id,state,tempObject,userRelatedProps,subEntities,operations,H>。聚合也是实体,不同在于内部还包含了其他实体,subEntities属性表示子实体集合。

定义24(值对象模型) ValueObject=<state,tempObject,userRelatedProps,operations,H>。与实体的不同在于值对象只是代表某一种概念的表述性特征,而不需要概念性标识,其主要表现为不需要唯一标识。

3.4　服务模型(Service Model)

定义25(领域服务模型) DomainService=<{repositoryInterface,other domainServices},domainOperations>,该定义为一个二元组,其中第一个元组表示所依赖的DDD元素。领域服务依赖于数据访问层的服务,以及其他领域服务的服务。第二个元组domainOperations是领域服务的操作方法集合,根据业务逻辑进行封装而成。后面的数据仓储、控制器的定义同样按照二元组的方式进行定义。

领域服务可以分为工作流、历史服务、草稿服务以及其他业务相关的领域服务。

对于工作流领域服务,其中的domainOperations={submit,accept,reject,startDel}。这几个操作将调用实体模型中的submit、accept、reject等操作,完成对实体属性的驱动,并调用数据访问元素来进行实体的持久化操作。

历史服务中的domainOperations={record,findHistories}。record将传入的任意类型的对象转换成统一的结构进行持久化,findHistories根据指定的对象来获取相关的历史对象集合。

草稿服务中的domainOperations={save,findDrafts}。save将传入的任意类型的对象转换成统一的结构进行草稿存储,findDrafts根据指定的对象,以及创建人的信息来获取相关的草稿对象。

定义26(应用服务模型) AppService=<{repositoryInterface,domainServices,otherAppServices},appOperations>。应用服务可以调用数据仓库的功能,也能调用领域服务和其他应用服务的功能。

针对领域层服务,应用层服务可以直接进行封装调用,不在这里详细阐述。

3.5　数据仓储模型(Repository Model)

定义27(数据仓储模型) BizRepository=<{dataSource},{get,find,add,save,update,delete},otherOperations>,其中dataSource表示该数据访问类的数据源,可以是数据库、XML文件、文件系统、分布式数据源等。get,find,add,save,update,delete是用来处理对象的增删改查的持久化操作。otherOperations是根据业务逻辑扩展的操作。

针对工作流处理、历史操作和草稿操作的数据仓储接口与一般的业务领域的数据仓库模型BizRepository类似。

3.6　控制器模型(Controller Model)

控制器模型是基于MVC的框架来进行定义的,控制器负责为页面提供数据的展现和收集,并与后端进行交互。在对Java的PetStore[14]等系统进行分析的基础上,发现业务操作主要就是围绕着页面的加载、关闭以及存活期间来进行的。

从业务对象的角度,页面的加载主要分为新建对象页面、修改与查看对象页面、列表搜索页面。划分的依据主要是根据控制器中逻辑的不同,新建对象页面的加载是在后端构建新的对象并展示,修改与查看对象页面的加载都是根据某对象id来获取对象的信息进行展示,列表搜索页面是获取某类对象的集合。

页面的关闭主要分为提交新建与修改、保存草稿和取消编辑或关闭三种情况。提交新建与修改都是将页面编辑的内容提交到后台进行持久化为NORMAL状态,保存草稿是将编辑内容提交后台持久化为DRAFT状态,取消编辑或关闭无需进行操作直接关闭即可。

页面存活期间的操作没有规律可循,在技术上可以通过form提交、ajax等手段来实现特定业务的操作。

根据上述分析,控制器的公共操作集合为{initViewUpdate,initCreate,initList,submit,save},其中initViewUpdate处理查看与修改对象页面加载,initCreate处理新建对象页

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

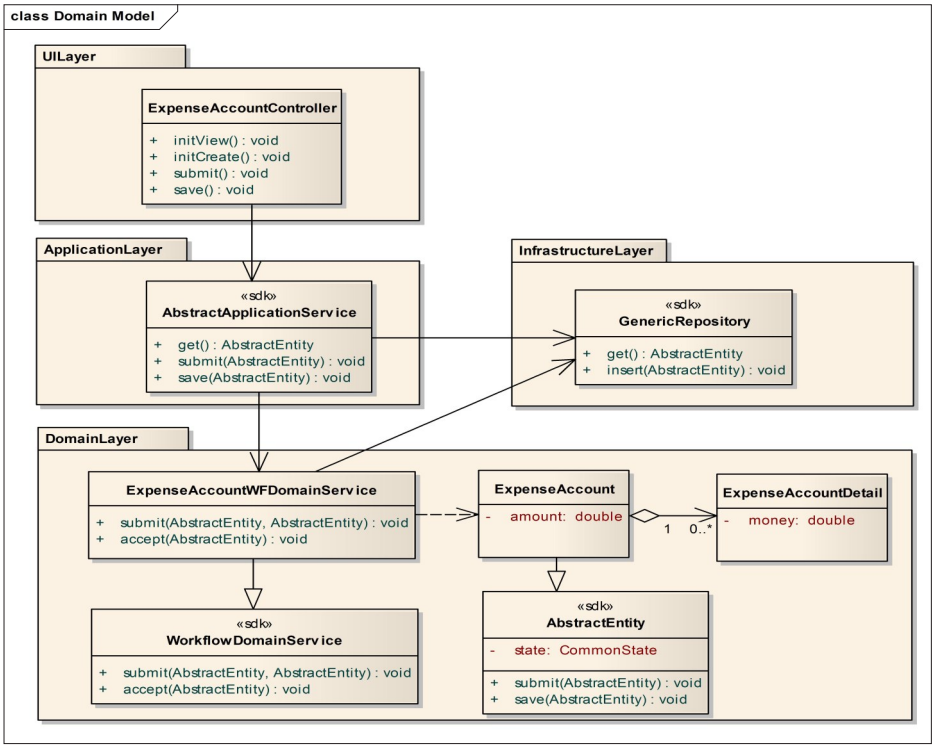

图3　系统架构图

面加载,initList处理列表搜索页面加载,submit处理新建或修改对象操作,save处理新建或修改对象的草稿保存操作。

定义28(控制器模型) Controller=<{AppService},{initView, initCreate, initList, submit, save, otherOperations}>,AppService是控制器主要调用的服务,控制器不直接调用其他元素。

4　模型实现及实例

DDDAppModel不仅仅是一套指导系统设计和开发的方法,还将其中的公共的和基础的部分抽取出来,封装为一个开发框架。下面将给出所实现的框架,并结合开发实例来验证有效性。

4.1　模型实现

整个开发框架基于Java语言,包含物理上独立的四个构件,分别对应领域层、数据访问层、应用服务层和UI层。

在领域层中,将定义22的实体模型进行封装抽象为基类AbstractEntity,所有的实体需要继承该抽象实体来实现业务对象的公共业务逻辑,比如工作流驱动。另外还包含了工作流领域服务WorkflowDomainService,历史操作领域服务HistoryDomainService和草稿操作领域服务DraftDomainService。

数据访问层中预定义了数据源的封装和数据访问的抽象类GenericRepository,该类包含了业务对象公共的持久化方法,比如get,find,insert等原子操作逻辑,该类可以根据需要被继承和扩展。

应用服务层中预定义了AbstractApplicationService类,采用设计模式中的门面(facade)模式将业务操作进行了封装。该类直接将业务对象的公共持久化方法暴露出来,可以由UI层直接调用。这就是3.2节中的简单操作逻辑只需要经过应用层和数据访问层即可完成的原因。根据业务需要,可以继承AbstractApplicationService类,定义与业务相关的复杂操作接口。

UI层中定义了界面的框架,包括菜单加载、页面加载、MVC映射等基础。

该框架的目标是能够支持实际项目的设计开发,所以其主要采用了业界成熟的框架作为基础,比如Spring、Hibernate、JBPM等,并综合采用了IOC、AOP、动态字节码、元数据驱动、状态模式等技术和方法来实现了整个框架功能。

4.2　实例分析

本文基于DDDAppModel,实现了一个企业的报销管理系统。其主要需求包括:报销表单包含多条报销明细集合,并需要汇总报销明细金额;申请人在新建报销表单时,可以将表单存为草稿,下次可以继续进行草稿编辑直到最终提交申请;报销表单可以通过配置实现是否需要审批以及哪些人审批的功能。

按照分层模型,整体系统架构如图3所示。

在UI层,建立ExpenseAccountController对象,其中包含方法initView、initCreate、submit和save,分别用来处理查看报销单时的数据加载逻辑、新建报销单时的初始化逻辑、提交新建报销单和保存草稿四个业务。

由于该业务简单,因此无需针对报销单建立应用层服务类,只需要直接使用框架的AbstractApplicationService,也就是ExpenseAccountController直接调用AbstractApplicationService的get、submit和save方法。

在数据访问层同样不需要专门建立数据访问类,直接使用框架的GenericRepository中的持久化方法get、insert

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

方法。

在领域层,建立报销单ExpenseAccount和ExpenseAccountDetail实体,分别包含报销总额amount和明细金额money属性。ExpenseAccount包含ExpenseAccountDetail的集合,形成了聚合根,对ExpenseAccount的持久化操作需要同时对ExpenseAccountDetail进行持久化操作。默认的WorkflowDomainService针对工作流处理的服务,其中的submit和accept方法中只是针对传入的处理对象进行状态的控制,而没有处理子对象的逻辑,这就需要对这两个方法进行重载,加入子对象处理的逻辑。在这里就构建了ExpenseAccountWFDomainService服务类,重载了submit和accept方法,在其中加入了汇总报销单明细金额的逻辑。

从以上设计开发流程可以看出:

(1)对于简单的业务逻辑,该框架的已有功能可以直接调用。

(2)针对较复杂的逻辑,业务逻辑的扩展点已经进行了定义(比如工作流中的submit和accept)。

(3)DDDAppModel对各层划分定义清晰,对业务系统公共的常规的操作进行了建模,能够指导系统设计和代码开发。

(4)面向领域的业务逻辑抽取和封装思想明确,通过模板方法的方式将业务逻辑的扩展点进行了抽象,使得开发人员能够更多关注于实际业务。

(5)能够提高开发效率,减少缺陷。在开发框架中已经将公共的和常规的业务逻辑(比如工作流处理)进行了封装,业务系统不需要编码即可直接调用。

因此,基于DDD的DDDAppModel能够很好地支持业务系统的设计开发。在实际生产环境中,该框架已经被应用,目前已经支持了十余个企业级业务系统,代码量已经超过一百万行。

5 结束语

本文提出了基于领域驱动设计的应用模型DDDAppModel,该模型对业务中公共的和基础的逻辑进行了封装,同时能够对DDD的设计开发提供指导。本文的贡献有三部分:

(1)对业务对象的操作进行了建模,并将一些操作进行了统一。

(2)提出了基于DDD的应用系统模型,定义了系统分层、实体模型、服务模型等,并对DDD元素在各层的分配进行了明确。

(3)根据该模型实现了开发框架,对工作流、历史操作、草稿操作等公共业务逻辑进行了封装,能够更好支持面向领域的设计开发,提高开发效率减少缺陷。

参考文献:

[1] Evans E.Domain-driven design--Tracking complexity in the heart of software[M].New Jersey:Addison-Wesley,2003.

[2] Fowler M.Analysis patterns:reusable object models[M].[S.l.]:Addison Wesley/Pearson,1996.

[3] Fowler M.企业应用架构模式[M].影印版.北京:中国电力出版社,2004.

[4] Landre E,Wesenberg H,Ronneberg H.Architectural improvement by use of strategic level domain-driven design[C]// The 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems,Languages,and Applications,2006:809-814.

[5] Wesenberg H,Landre E,Ronneberg H.Using domain-driven design to evaluate commercial off-the-shelf software[C]// The 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems,Languages,and Applications,2006:824-829.

[6] Landre E,Wesenberg H,Olmheim J.Agile enterprise software development using domain-driven design and test first[C]// The 22nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion,2007:983-993.

[7] 丁涛.基于领域驱动设计的物流平台系统实现[D].成都:电子科技大学,2010.

[8] 孙全智.Java EE项目开发与领域驱动设计实践[D].大连:大连理工大学,2009.

[9] 王鹏,刘渊,冷文浩.领域驱动设计在SPP系统中的应用[J].计算机工程与设计,2008,29(13):3362-3364.

[10] Bauer C,King G.Hibernate in action[M].[S.l.]:Manning Publications,2004.

[11] Hamilton B,MacDonald M.ADO.NET in a nutshell-a desktop quick reference:includes bonus visual studio.NET add-in[M].[S.l.]:O'Reilly,2003.

[12] 苏焕程,黄志球,刘林源.基于接口自动机的BPEL4WS Web服务组合形式化模型[J].计算机应用研究,2009,26(5):1774-1777.

[13] 辜希武,卢正鼎.基于Pi一演算的BPEL4WS Web服务组合形式化模型[J].计算机科学,2007,34(3):69-74.

[14] Oracle.Java pet store[EB/OL].(2012-10-25).http://www.oracle.com/technetwork/java/petstore1-3-1-02-139690.html.

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net