

第5章

功能设计



本章重点

- (1) 工作场景的撰写方式。(★★)
- (2) 如何进行功能划分。(★★★)
- (3) 常用的基础功能逻辑。(★)
- (4) 什么是工作流。(★)
- (5) 如何画工作流图。(★★★★★)
- (6) 常见的功能逻辑案例。(★★)
- (7) 如何提高软件功能的灵活性。(★★★★★)
- (8) 如何提高软件功能的可重用性。(★★★★★)
- (9) 如何提高软件功能的高效性。(★★★★★)



本章内容思维导图



软件的功能,从本质上说就是对数据进行输入、加工、输出的过程。对于面向数据库的软件,由于是以数据库为核心的,可以理解为两个方面:一是数据的收集与处理;二是围绕数据库对其中的数据进行的4大操作,即增加、删除、修改、查询,简称增删改查(对应SQL语句的4个处理数据的关键字:Insert、Delete、Update、Select)。所有面向数据库的软件功能,都可以概括成从数据库中读出数据(Select),经过用户一定的操作,或经过程序一定的加工,再写入数据库(Insert、Delete、Update)。当然也有些特殊的功能,可能只有写入没有读出,例如,通过物联网感应器材直接收集数据到数据库;也有可能只有读出没有写入,例如大量的报表、查询功能。



5.1 需求用例

5.1.1 什么是需求用例

用例(Use Case),指实际工作中可能发生的场景。在需求分析阶段所说的用例,称为“需求用例”,是指用户通过软件解决特定问题、完成指定任务的方式与步骤,当然也包括各步骤用到的约束、规则等。一个用例,往往对应着用户需要完成的某个明确而具体的任务,但也有两种特殊的用例:一种是上层用例,另一种是底层用例。上层用例,指结合一些有关联的普通用例完成一个抽象的由若干普通任务组成的大任务;底层用例,指完成某些小任务的用例,这种用例可能会在许多普通用例中被引用。



案例：什么是需求用例

某仓库需要管理本公司的原材料与半成品,就收货业务而言,本仓库包括两种基本的收货方式:一是收取供应商送来的原材料,二是收取车间送来的半成品。根据这两种业务,对应两个普通用例:一是仓管员收取供应商送货,在这个用例下,仓管员通过检索采购单,录入入库物料、数量、存放位置,确认入库等一系列步骤,完成了根据采购单收货这个具体的任务;二是仓管员收取车间送货,仓管员通过检索生产单,录入入库物料、数量、存放位置,确认入库等一系列步骤,完成了根据生产单收货这个具体的任务。但这里还有另外两种用例,一是“收货”,收货本身是个抽象的大任务,包括根据采购单收货与根据生产单收货两个普通用例,这是个上层用例;二是“入库”,两个普通用例都需要录入入库物料、数量、存放位置这些操作步骤,这是个底层用例。

5.1.2 用例的构成

一个完整的用例,一般包括用户、前置条件、后置条件、主场景、扩展场景、规则等方面。

1. 用户

用例的重点在于用户的操作场景,在考虑用例的场景之前,需要先确定用例的用户,因为所有的使用场景都是为某些特定的用户服务的。一个用例可以面向一种或多种用户,例如,用例“仓库结账”,只是面向仓库核算员的,而用例“仓库交易分析”,会面向多种用户,如仓管员、核算员、计划员、采购员等。根据面向的用户不同,可以将用例分成几大类:面向普通用户的用例、面向关键用户的用例、面向系统管理员的用例、面向所有用户的用例。

面向普通用户的用例,是普通用户从事业务处理的用例。由于使用者是一般工作人员,学习能力、文化水平、软件知识参差不齐,设计这种用例时往往会在易学性、健壮性、易用性上下更多的工夫。这种用例设计得不好,会严重影响工作效率。

面向关键用户的用例,主要用于系统数据的初始化、业务功能的配置、基础数据的管理等,如“用户管理”“仓库配置”“组织结构建立”之类的用例。使用这种用例的用户,虽然没有系统管理员对软件理解得那么深刻,但也是对软件比较熟悉的人员,使用这种功能进行操作

对系统的影响非常大,但一般来说影响的幅度与程度不及面向管理员的用例。

面向系统管理员的用例,主要用于系统配置、运行监控、异常分析、功能维护等,这些用例一般会牵涉系统的正常运行,操作稍有不慎就可能导致系统异常甚至崩溃,所以一般只能给系统管理员使用。

面向所有用户的用例,提供给所有登录本系统的用户的用例。如“修改密码”“工作日志”之类,只要是登录本系统的用户,就可以使用这些用例。当然,既然是面向所有用户的,自然也应该归于面向普通用户的用例,这是一种特殊形式。

2. 前置条件

所谓前置条件,是为了保证本用例可以成功执行,而需要满足的前提条件。例如,在某电商网站,用例“下订单”的前置条件是会员登录成功,并且会员信息中的一些必备资料填写完整;用例“撤销订单”的前置条件是会员登录成功,并且订单还没有发货;而用例“退货”的前置条件是订单已经发货。

3. 后置条件

所谓后置条件,是指用例执行结束后的系统状态,无论成功还是失败。例如,银行柜员机系统,用例“提款”的后置条件是:如果用例执行成功,柜员机钞票减少,减少额度等于用户的提款金额;如果用例执行不成功,柜员机钞票不变,用户的银行账号余额不变。

4. 主场景

“场景”这个词,在软件设计与实施过程中会经常使用,这里所谓的场景,指用户使用软件功能完成工作任务的操作过程。这里的场景是由一系列人机交互的步骤构成的,强调的是人做了什么操作,机(软件系统)有什么应答,来来往往,经过若干回合后,结束了某项任务,当然,有可能成功,也有可能不成功。

由于用例都是有明确的任务的,因此,每个用例都应该有个主目标,这个主目标就是支持用户通过这个用例完成某项具体任务,但为了使这个目标实现得更高效、更准确、更容易、更健壮,犯了错误可以得到纠正,一些异常事件可以得到处理,需要软件提供一系列的额外功能。根据二八法则,平均下来应该有 20%的功能是用来完成主目标的,而 80%的功能是为了提高效率、降低错误率、纠正错误的。例如,用例“仓管员根据采购单收货”,它的主要目标是将收货记录录入系统中,因此录入并保存收货记录是主目标,而编辑功能是为了纠正错误或应对变化,删除功能是为了纠正错误,导入功能是为了录入更快速,这些都不能称为这个用例的主目标。

用户为实现自己的主目标而进行操作的过程,称为用例的主场景。大部分情况下,一个用例只有一个主目标,只有一个主场景,如果主场景不明朗,往往说明这个用例是上层用例。例如“会员登录”这个用例,主场景包括用户录入会员卡号、密码,登录成功这个过程,别的处理密码输入错误、忘记密码之类的场景,显然不属于主场景的范畴,因为用户跑到这里是为了登录,输错了密码,或者忘记了密码只是一些意外情况。

要注意的是,主场景是实现用户主目标的过程,但未必是最常用的场景,不可混为一谈,例如“文员进行客户档案维护”这个用例,录入客户信息是这个用例的主目标,是主场景,但最常用的场景恐怕应该是浏览客户信息吧。



案例：用例的主场景

银行柜员机提供了很多软件功能,如提款、存款、查余额等。如果围绕柜员机设计用例,那么提款应该是这个用例的主目标,实现这个主目标的主场景大概包括如下步骤。

- (1) 用户插入银行卡。
- (2) 系统确认卡正确。(处理错误卡的操作不属于主场景。)
- (3) 用户录入密码。
- (4) 系统确认密码正确。(处理密码输入错误的操作不属于主场景。)
- (5) 用户录入取款金额。
- (6) 系统确认余额足够。(处理余额不足的操作不属于主场景。)
- (7) 系统吐钱,给绑定手机发短信。
- (8) 用户确认打印凭条。
- (9) 系统打印凭条。
- (10) 用户确认交易结束。(用户继续进行其他操作不属于主场景。)
- (11) 系统吐卡。
- (12) 用户取卡。
- (13) 系统确认用户已取卡,结束交易。(处理用户未取卡的操作不属于主场景。)

这些步骤是用户使用柜员机提款的主场景。再仔细想想,虽然提款是用得最多的功能,但绝对不能说是柜员机的主目标,例如存款跟取款几乎毫无关系,怎么着都不能说存款功能是对取款功能的补充吧,这应该是另外一个独立的主场景,因此,存款应该是另外一个用例。因此,面向整个柜员机的用例应该是一个上层用例,包括登录、提款、取款等普通用例。

5. 扩展场景

每一个用例,都有各种各样的使用场景,主场景只是这若干种场景中的一种,主场景之外的场景,称为“扩展场景”。例如,一个简单的用例“用户登录”,主场景显然是用户输入用户名、密码,验证成功后进入系统,但还有别的可能,如用户密码输错了怎么办,用户忘记了密码怎么办等,这些都要有相应的处理场景——扩展场景。



案例：用例的扩展场景

用例“用户登录”的扩展场景。

扩展场景一：密码输入错误。

- (1) 用户录入用户名、密码,确认登录。
- (2) 系统验证用户名、密码,密码验证错误,提醒用户只允许输入三次。
- (3) 用户重新输入密码。
- (4) 系统验证密码,如果验证正确,则进入系统。如果验证错误,且输入已经超过三次,则锁定该用户,并提醒用户账号已经被锁定,如果没有超过三次,则用户重新输入密码。

扩展场景二：用户忘记密码。

- (1) 用户录入用户名、密码,确认登录。
- (2) 系统验证用户名、密码,密码验证错误,系统提醒是否需要取回密码。

- (3) 用户确认取回密码。
- (4) 系统发送验证短信到本账号所登记的手机。
- (5) 用户提交短信验证码。
- (6) 系统确认验证码正确。
- (7) 用户录入新密码。
- (8) 系统将当前用户的密码重置为新密码。

6. 规则

规则是指本用例用到的业务规则、逻辑算法等。有的用例逻辑简单,几乎没有什么规则,无非只是些数据的录入、保存而已,而有些用例,逻辑非常复杂,需要进行大量的运算、判断,在这种情况下,就需要整理进行运算、判断的规则。在这里整理的规则更倾向于用户,措辞方式以一般用户能理解为基本要求,应当尽量避免使用太过技术化的 IT 术语;另外,这里也不是用户在需求调研时提供的规则的简单记录,应该有一个整理、分析、抽取、加工的过程。



案例：用例的规则

用例“考勤分析”的规则。

判断日班考勤正常的规则：8:00—9:03 打卡,并且在 17:27—23:59 打卡。

判断迟到的规则：在 9:03—9:30 打卡。

判断早退的规则：在 17:00—17:20 打卡。

上述时间段中打卡的记录为有效打卡,在当班没有有效打卡记录则为旷工。

5.1.3 用例编写

这里介绍一个会员下单的用例供读者在工作中参考(为了节约篇幅,做了精简)。



案例：电商平台会员下单用例

1. 用例编号
UC0210
2. 用例名称
会员下单
3. 前置条件
当前用户已登录。
4. 后置条件
用例执行成功,生成当前用户的新订单,减少商品的可供应数量;用例执行失败,不影响商品的可供应数量。
5. 主场景
 - (1) 用户检索商品,录入购买数量(L1)。
 - (2) 系统确认库存数量足够。
 - (3) 用户暂存商品。
 - (4) 系统将商品加入购物车,加载当前用户可能感兴趣的跟当前商品相关的商品。

- (5) 用户继续检索商品,重复 L1 步骤。
- (6) 用户确定下单。
- (7) 系统确认用户收货信息已经完善。
- (8) 系统生成新订单,减少相关商品的可供应数量,清空购物车。

6. 扩展场景

6.1 扩展场景一: 库存数量不足。

- (1) 用户检索商品,录入购买数量。
- (2) 系统发现当前商品的可供应数量不足。
- (3) 系统提醒用户可以发起预订请求。
- (4) 用户发起预订,输入到货通知方式。

6.2 扩展场景二: 用户没有收货地址信息。

- (1) 用户确定下单。
- (2) 系统发现用户没有收货地址信息。
- (3) 系统提示用户录入收货地址。
- (4) 用户录入收货地址。
- (5) 系统生成新订单。

7. 业务规则

7.1 当前用户可能感兴趣的商品的检索规则:

- (1) 跟当前商品属于同一系列的商品。
- (2) 当前用户浏览过相关主题的商品。
- (3) 跟当前商品可以打包销售的商品。
- (4) 同类商品正在搞活动促销的商品。

对于需求用例的编写,在实际工作中,不同的团队有不同的要求,有些团队,对需求用例的编写要求非常高,需要仔细描写每一个应用场景;而有些团队或项目的要求非常简单,甚至根本不需要进行需求用例的分析、编写就直接进入了功能点设计工作。每种方式都无所谓对与错,在这里还是要强调这句话——“适合的才是好的”。由于进行完备、详细的需求用例分析与编写需要花费大量的时间,有时候也确实有些得不偿失的感觉。为了提高工作效率,笔者的想法是,对于一些比较重要的、核心的业务,可以先进行需求用例的分析,为后面的功能设计奠定扎实基础;而对于那些比较简单的业务,可以直接进入功能设计,对于那些使用场景简单、业务逻辑也不复杂的项目,完全可以跳过这个步骤。

另外,编写需求用例,也完全可以根据业务需求进行有针对性的撰写,例如,某个扩展场景比较复杂,容易出现理解误差,那么就下工夫将这个扩展场景写清楚,既保证跟用户的沟通无误,也保证团队其他成员容易理解,而其他简单的场景就可以简单一点儿甚至略过。

撰写场景的技巧,在需求分析过程中很常用,需求人员必须非常熟悉。



案例: 灵活的需求用例撰写方式

小王在某原材料仓库做需求分析。原材料仓库需要向车间提供原材料,车间的工作方式是三班倒班,夜间正常运转,为了配合车间生产,势必要求仓库中也采用三班倒班的工作方式,一个班至少需要配置两个人,三个班就需要六个人,而仓库的工作量其实只需要两个人

就足够了,采用三班倒班的工作方式显然是非常不明智的。仓库目前只配置了两名员工,上日班,夜间不工作,为了解决车间夜班用料的问题,一般在当天仓库人员下班前,车间会将夜班需要用到的原材料先领出。进行信息化管理后,要求车间所有的原材料都根据生产任务单领出,但有些原材料,如包装塑料膜,并没有在产品的BOM中体现出来,因此无法根据生产任务单领出来,只能先将这部分物料“出借”给车间的临时仓库管理员,夜里车间领料时就从这些借出的物料中领用,早晨仓库上班后,再将多出的物料退回仓库。由于这个领料过程有些复杂,小王决定先撰写工作场景,然后再跟相关人员进行详细讨论。

车间夜班领料场景:

- (1) 用户选择本夜班需要生产的生产任务单。
- (2) 系统加载这些生产任务单需要的原材料。
- (3) 系统确认仓库相应原材料结存足够。
- (4) 用户确认领料。
- (5) 系统发料,打印领料单。
- (6) 用户补充录入需要借出的原料。
- (7) 系统确认仓库相应原材料结存足够。
- (8) 用户确认借料。
- (9) 系统发料,打印借料清单。

车间早晨还料场景:

- (1) 用户选择夜班借料单。
- (2) 系统加载借料信息。
- (3) 用户选择每种物料使用的生产任务单。
- (4) 用户录入还料数量。
- (5) 用户确认还料。
- (6) 系统发料,分摊到相应生产任务单,打印退料单。

5.2 功能建模



5.2.1 什么是功能建模

所谓功能建模,指根据系统要求设计功能构成模型,确定系统由哪些功能构成,每个功能应该输入什么,经过功能处理后应该输出什么,每个功能又包括哪些子功能,不断分解下去,直到最底层。

为了支持快速开发,本书所介绍的快速原型开发模型,并不强调严格的功能建模,在本阶段要做的事情是确定本软件系统需要哪些功能模块,每个功能模块包括哪些功能点,每个功能点包括哪些子功能等。

5.2.2 功能点

“功能点”这几个字,不同团队的使用含义并不相同,本书所谓的“功能点”,指可以提供给用户完成某一特定任务的功能组合,例如“客户档案维护”“物料基本信息管理”等,跟研发

人员所说的某类可以提供某功能是完全不同的两个概念。或者,可以将其看成是传统的功能菜单,大部分情况下可以简单粗暴地认为一个菜单算是一个功能点。当然,并不是所有的功能点都是有功能菜单对应的,例如某些固定时间触发的调度功能,某些给第三方调用的接口等。

一个功能点可以支持多个需求用例,一个需求用例可能依赖于多个功能点才能实现。功能点跟需求用例既有很强的关联性,又有根本区别。说有很强的关联性,因为用例要成功实现,离不开软件的功能点,软件的功能点是用来实现用例的工具;说有根本区别,因为需求用例强调用户如何通过软件处理问题,而功能点更强调软件提供哪些功能。

每个功能点由或多或少的一些子功能组成,如新增、编辑、删除、导入、导出等,用户通过这些功能的组合运用,可以处理某些特定的任务。

例如,某 CRM 系统的功能点“客户档案维护”,原型主界面如图 5-1 所示。

<input type="button" value="新增客户"/>		<input type="button" value="客户导入"/>					
客户代号:	<input type="text"/>	客户名称:	<input type="text"/>	客户类别:	<input type="text"/>	<input type="button" value="查询"/>	<input type="button" value="重置"/>
<input type="checkbox"/>	客户代号	客户名称	法人代表	邮编	省市	客户类别	操作
<input type="checkbox"/>	CZWF	常州五洋电子	杜芸	213100	江苏常州	潜在客户	详情 编辑 删除
<input type="checkbox"/>	NJAR	南京爱仁文具	冯莹莹	210000	江苏南京	VIP客户	详情 编辑 删除
<input type="checkbox"/>	NJDN	南京大楠科技	解学明	210000	江苏南京	一般客户	详情 编辑 删除
<input type="checkbox"/>	SHDY	上海大洋纸业	凌晓	200000	上海	潜在客户	详情 编辑 删除
<input type="checkbox"/>	SHQT	上海晴天商贸	刘士栋	200000	上海	一般客户	详情 编辑 删除
<input type="checkbox"/>	SZFCR	苏州丰辰润建材	苗金超	215300	江苏苏州	VIP客户	详情 编辑 删除
<input type="checkbox"/>	WXFF	无锡风帆物流	徐纪伟	214000	江苏无锡	VIP客户	详情 编辑 删除
<input type="checkbox"/>	WXKL	无锡昆仑电气	姚洪侠	214000	江苏无锡	一般客户	详情 编辑 删除
<input type="checkbox"/>	WXLX	无锡梁溪机械	张国栋	214000	江苏无锡	VIP客户	详情 编辑 删除
<input type="checkbox"/>	WXSL	无锡尚朗科技	朱纯鹤	214000	江苏无锡	一般客户	详情 编辑 删除
首页 上一页 下一页 末页							

图 5-1 客户档案维护界面

本功能点的目的是使用户可通过软件维护公司的所有客户档案信息。为了达到这个目的,当有了新客户时,用户可以通过“新增客户”功能录入新客户的档案信息到数据库,可以通过“客户导入”功能快速录入大批量的客户档案信息到数据库;如果客户档案信息发生了变化或者录入信息有误,可以通过“编辑”功能修改数据库中的信息;如果误录入客户,可以通过“删除”功能从数据库中删除信息——正是这些子功能的组合协作,才可以让用户顺利完成维护客户档案这项任务。

功能点的含义是可大可小的,没有必要过于拘泥,可以将其理解成介于功能模块与原子功能之间的一个概念。

所谓功能模块,指一些在业务上有一定关联性的功能点组合,这些功能点可以分别完成某些小任务,这些小任务又是为某一大任务服务的。例如,财务软件的“账务管理”功能模块,包括“会计科目设置”“会计期间设置”“记账凭证录入”“明细账生成”等功能点,这些功能点相互之间有一定的关联性,如记账凭证录入之前需要设置好会计科目、会计期间,正是这些功能点的联合操作,才能完成账务管理这个较大的任务。

有一种子功能,从用户的角度看,是无法分割的,一旦被触发就将控制权转让给了系统。

例如,用户单击“删除”按钮,系统获得控制权,执行 Delete 语句,删除了某条记录,在删除过程中用户是无法干预的;用户单击“保存”按钮,系统获得控制权,执行 Insert 语句,将数据保存到数据库,在保存过程中,用户是无法干预的。虽然从开发者的角度看,这些功能还包括更小的功能,如调用了一些函数,但从用户的角度看,这属于最小功能,具有原子性,因此本书称之为“原子功能”。各种对数据进行操作的按钮、功能图标、链接、快捷方式等,都属于原子功能,因为用户操作时,就不能对接下来系统要做的事进行干预了,除非系统在执行过程中发起干预请求,如询问是否确实要删除某条记录。

5.2.3 原子功能

每一个功能点都是由或多或少的原子功能构成的,一个典型的原子功能包括从数据库或界面获得数据,经过加工处理后提交到数据库,再将处理结果反馈到界面这样一个主要过程,如图 5-2 所示。

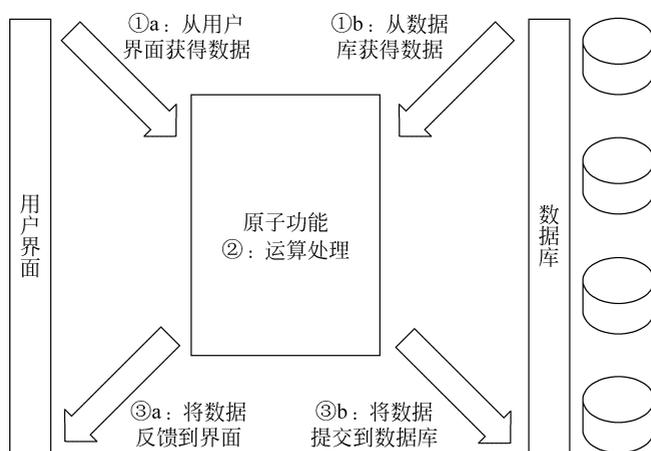


图 5-2 原子功能的数据处理过程

一般来说,原子功能在执行过程中包括三个大的方面:获得数据,处理数据,提交结果。

(1) 获得数据,是运算处理的准备阶段。获得数据一般有两种来源:一是从用户界面获得,例如用户在录入框中录入的内容,用户做出的某些选择等;二是从数据库中获得,例如执行“用户登录”功能时,需要从数据库中获得用户的菜单权限清单。

(2) 处理数据,是对获得的数据进行运算、处理的过程。有的功能这个过程非常简单,或者几乎不需要这个过程,有的功能这个过程就可能非常复杂,例如一次 MRP 运算,可能需要几分钟甚至几小时的运算。需要注意的是,图 5-2 只是个示意图,在现实中,不大可能先将所有需要的数据一把拿出来再运算处理,而是会根据运算的需要,不断从数据库中获得当前步骤需要的数据,尽量保证运算速度更快、开销更小、安全性更高。

(3) 提交结果,是运算处理的结束阶段,有两个可能:一是将结果提交到数据库,二是提交到用户界面。

当然,并不是每个原子功能都包括这所有的处理过程,有些功能只要从界面获得数据,不需要经过数据库,有些将处理结果直接保存到数据库,不需要反馈到界面,有些简单操作就几乎没有任何运算处理的过程(只是在应用层面,其实没有运算处理是不可能的,哪怕一

次简单的数据传输、显示也是需要计算机处理的),等等。

另外,每个原子功能还可能包括若干分支,每个分支代表这个功能执行的不同结果,有的时候是因为执行成功或不成功而有不同的结果,有的时候是因为某些选项、配置的不同又有不同的结果。



案例：用户登录功能的逻辑

某软件系统中的功能点“用户登录”,其中包括原子功能“登录”,用户录入用户名、密码后,执行这个“登录”功能。“登录”的主要功能是验证用户名、密码,然后加载系统首页。这个功能有三个分支:一是用户名不存在的分支,二是密码验证不通过的分支,三是用户名密码验证通过的分支。

下面从获得数据、处理数据、提交结果这三个方面分析一下这个原子功能的处理过程。

获得数据:从界面上获得用户录入的用户名、密码(明文),根据用户名从数据库中获得对应用户的密码(密文),根据用户名获得当前用户对应的角色,根据角色获得菜单权限清单。注意,这里列举了所有各个分支下需要的数据,在不同的分支下,所需要的数据并不相同。

处理数据:查找是否有本用户名,没有,则功能运行结束(分支一);如果有本用户,则根据用户录入的密码,通过加密算法获得密文,与从数据库中获得的密码密文比较,如果不相同,则密码验证不通过,功能运行结束(分支二);如果验证通过,则通过用户角色、角色权限,计算获得当前用户的权限清单,生成登录首页,功能运行结束(分支三)。

提交结果:如果用户名在系统中不存在,则在界面上反馈信息“没有当前用户”(分支一);如果密码不正确,则在界面上反馈信息“密码错误”(分支二);如果密码正确,则加载登录首页,并在日志表中记录登录日志(分支三)。

5.2.4 划分功能

进行功能设计首先要做的事情是进行功能划分,即设计者试图通过哪些功能组合,来解决用户的问题,从而达成企业信息化管理的目标。在这个阶段主要考虑这个软件系统会包括哪些功能模块,功能模块由哪些功能点组成,每个功能点包括哪些子功能,每个子功能包括哪些原子功能,每个功能需要输入什么、如何处理、输出什么,哪些用户使用这些功能,使用这些功能是为了解决什么问题,怎么使用这些功能等。

要做好这件事,可以从以下这些方面来思考。

(1) 用户需要通过本系统处理哪些需求用例?虽然功能点并非完全根据需求用例来划分,但绝对有很大的关系。

(2) 用户需要通过本系统处理的具体任务有哪些?虽然功能点并非完全根据用户的任务设置,但绝对有很大的关系。

(3) 如果你是管理者,需要将这些任务分配给不同的人员,你觉得可以接受的最小任务粒度是什么?一个功能点往往意味着一项任务的最小粒度。例如前面所说的“客户档案维护”,不大可能将“删除客户”作为一个单独的任务分配给某人吧?所以,“删除客户”不能作为一个功能点存在,它只能作为隶属于某个功能点的子功能或者原子功能。

(4) 为了完成每个任务,需要哪些功能支持?大部分功能点都需要包括对数据的增删

改查这些子功能。

(5) 有没有那种处理起来很复杂,需要的信息量很大,需要处理的数据很多,但绝不可能分拆给不同的人处理的任务?这种任务可以考虑分拆成多个功能点,将这些功能点组合成一个内聚性较强的功能模块。



案例：划分功能

某库存管理系统的功能划分见表 5-1。

表 5-1 某库存管理系统的功能划分

功能模块	功能点	子 功 能	原 子 功 能
基础模块	仓库配置	仓库基本信息维护、仓库配置	查询仓库、新增仓库、编辑仓库、删除仓库、会计期间查看、确定仓库责任人、确定仓库核算员
	库位配置	库位维护、库位属性设置	查询仓库、查询库位、录入库位、导入库位、导出库位、编辑库位、删除库位、保存库位属性
	包装物	包装物维护、包装物库位	查询包装物、录入包装物、导入包装物、导出包装物、分配库位、转移库位、编辑包装物、删除包装物、生成条形码、打印条形码
	物料	物料维护、物料属性设置	查询物料、录入物料、导入物料、编辑物料、删除物料、保存物料属性、生成条形码、打印条形码
库存交易	入库	直接入库、根据采购单入库、根据生产单入库、销售退货	查询采购单、查询生产单、查询销售单、查询仓库、查询库位、查询会计期间、查询物料、确认入库、打印入库单
	出库	直接出库、采购退回、根据生产单出库、根据销售单出库	查询采购单、查询生产单、查询销售单、查询仓库、查询库位、查询会计期间、查询物料、确认出库、打印出库单
	调拨	调拨录入、调拨结果确认	调拨源物料查询、结存查询、调拨目的物料查询、调拨交易录入、确认调拨、打印调拨单
	盘点	盘点表生成、盘点结果录入、盘点结果入账	查询仓库、查询物料结存记录、打印盘点表、录入盘点结果、确认盘点入账、查询盘盈盘亏记录
仓库核算	仓库结账	结账、撤销结账	查询结账历史、查询仓库、查询会计期间、结账、撤销结账、导出结账历史
	存货成本计算	计算方式设置、存货成本计算、计算结果确认	录入计算方式、查询仓库、查询会计期间、选择计算方式、计算存货成本、确认计算结果、打印存货成本报表
统计查询	仓库交易查询		查询库存交易、导出
	仓库结存查询		查询仓库结存、导出
	仓库明细账		查询仓库明细账、导出
	存货成本统计表		生成存货成本统计表、导出、打印

5.3 功能逻辑



每个功能都有其内部的运转逻辑,正是通过这些逻辑才能处理、传递信息,才能满足业务规则的要求,才能让计算机按照人的想法工作。功能逻辑越复杂,往往也意味着内部结构越复杂;业务规则越复杂,运算要求越复杂;功能逻辑越智能,越能体现计算机的优势,对于软件来说,这也往往表明它越有价值,越能帮人类做更多的事情。

5.3.1 基础功能逻辑

基础功能逻辑是指围绕数据进行的基础操作,大部分看上去复杂无比的功能都是由这些基础功能组合而成的。基础功能逻辑可以分成两大部分:一是针对数据库的操作,无非就是数据的增加、删除、修改、查询 4 种操作;二是跟数据库没有直接关系的操作,主要包括计算、显示、数据传递等。

1. 数据库操作

数据库操作指针对数据库进行的增删改查操作,下面分为增加数据、修改数据、删除数据、查询数据几个类别讲述一些常见的基础功能逻辑。

1) 增加数据

增加数据指在数据库或文件系统中添加内容。包括一次在数据库中新增一条记录、一次在数据库中新增多条记录、新建文件、在文件中添加内容等基础功能逻辑。

(1) 新增一条记录:使用 Insert 语句在数据库的某个表中增加一条记录。这种功能逻辑大部分都是伴随着新增、发起、填写等功能中的某个保存、提交、确认之类的按钮执行的。为了保证进入数据库的数据是合法的,一般在正式提交数据到数据库前都需要进行或多或少的数据验证工作,如某些必填字段是否填写了,某些字段之间的钩稽关系是否正确,某些引用的数据是否存在等,在实现时,这种验证性的代码量比提交数据的代码量要大得多。

(2) 新增多条记录:使用 Insert 语句在数据库的某个表中,或者多个表中同时增加多条记录。这种功能逻辑一般都是伴随着某个导入、批量生成、批量复制之类的功能执行的。从表面上看,新增多条记录无非是新增一条记录的重复执行,但实际上,新增多条记录的功能逻辑比新增一条记录要复杂很多,特别是准备同时新增的这些记录之间有一定的关联关系时。

(3) 新建文件:在系统相关的某个文件夹中添加文件。这种功能逻辑一般都伴随着上传附件、上传图片、上传照片、上传视频等功能中某个确认、保存之类的按钮执行的。为了节约系统的存储空间或者节约网络资源,一般都需要对文件的大小进行控制,有的软件会用求哈希值之类的方法判断系统中是否存在相同的文件,从而提高上传速度、节约存储空间。

(4) 在文件中添加内容:向系统中已经存在的某个文件添加文件记录。由于面向数据库的软件,数据一般都存放在数据库中,这种功能逻辑用得很少,但在有些特殊情况下还是用得多的,如在某个日志文件中不断添加系统的执行日志。

2) 修改数据

修改数据指修改已经存在的数据记录,包括编辑一条记录、编辑多条记录、修改特定字

段等。

(1) 编辑一条记录：使用 Update 语句修改某条记录的部分或全部字段，这种功能一般也是伴随着界面的某个保存、提交、确认之类的按钮执行的。处理编辑的功能逻辑比处理新建的要复杂得多，因为新建的数据是全新的，它不会被其他数据引用，只要保证本条记录合法就行了，而被编辑的数据不一样，它在数据库中已经存在了或长或短的一段时间，有可能有别的数据引用它、依赖它，如果处理不当，就有可能导致其他本来合法的数据不合法了，严重时可能会导致数据崩溃。

(2) 编辑多条记录：使用 Update 语句同时更新一个表或多个表中的多条记录。这种功能逻辑不常出现。由于这类操作会影响大批数据（包括需要更新的数据，以及依赖于这些数据的数据），在设计功能逻辑时需要非常慎重，除非确有必要，否则应该尽量避免这种操作。

(3) 修改特定字段：这其实应该属于编辑一条记录的范畴，但由于在实际工作中，这种特殊的功能逻辑出现频率很高，于是就单列出来了。这种功能逻辑一般用于某个特殊的操作，如锁定记录、修改状态等，跟编辑一条记录不同的是，它只是修改了某个特定的字段，因为这个字段值的变化，导致这条记录的性质发生了根本性的变化，如不再允许修改了，不可见了，属于不同的类别了，等等。

3) 删除数据

删除数据指删除数据库中已经存在的数据，也包括删除一条记录、删除多条记录两种基本功能，这里不再赘述。根据实现方式，删除数据可以分为逻辑删除与物理删除两种基本方式。

(1) 逻辑删除：从用户的角度，数据已经被删除，但数据其实还保存在数据库中。其实，这种删除方式只是修改数据的一种，实现方式一般是使用 Update 语句在数据库中对需要删除的记录做被删除标记，在展现数据时，根据这个标记做处理，使被标记的记录对用户来说变得透明。

(2) 物理删除：使用 Delete 语句将记录从数据库中删除。物理删除的处理逻辑比逻辑删除复杂得多，逻辑删除只是给待删除记录做了个删除标记，不会影响与之相关联的数据，需要的验证逻辑很少甚至根本不需要验证，而物理删除是直接删除记录，在删除前需要判断系统中是否存在跟它有关联的数据，处理稍有不慎就会带来可怕的结果，特别是一些基础数据，由于被大量引用，如果被误删除会导致整个系统数据崩溃。

4) 查询数据

查询数据指根据一定的条件从数据库中通过 Select 语句将数据提取出来，展现给用户。由于数据库保存大量数据的主要目的就是用于信息共享、统计分析等，因此查询数据是最常用的功能逻辑，99%的功能都离不开查询数据这个功能逻辑。用户使用软件时，几乎所有能通过界面看到的、听到的信息（对于有些特殊的软件界面，恐怕还要加上嗅到的、摸到的信息），都是通过查询数据逻辑从数据库中获得的。从功能上看，查询数据一般包括系统自动获取、用户间接查询、用户根据条件查询、数据导出、文件下载、生成报表等方式。

(1) 系统自动获取：那些并非由用户触发的功能，如调度任务、接口程序等，在运行过程中一般都需要从数据库中获取数据——在执行过程中根据条件不断从数据库中查询获得，这个过程跟用户无关，是系统自发进行的。

(2) 用户间接查询：在很多时候，查询获取数据的过程是用户发起的，但查询这些数据并不是用户的目的，用户另有目的，查询这些数据只是为了实现用户的目的而进行的辅助过程。例如，用户打开了某界面后，系统自动加载了某些初始化信息；用户确认登录后，系统从数据库中读取当前用户的密码用以验证。

(3) 用户根据条件查询：用户为了查询自己需要的信息，在界面上输入查询条件，系统根据这些查询条件在数据库中检索数据，然后将符合条件的数据组装成用户需要的信息，展现到界面上，平常所说的某查询功能一般就特指这种查询。例如，用户为了查询某个客户上个月的订单，在界面上输入客户代号、月份，系统根据这个客户代号、月份在数据库中检索这个客户的所有订单，然后呈现在用户界面上。

(4) 数据导出：根据用户要求，将符合条件的数据从数据库批量导出生成某种文件。最常见的是 Excel 文件，其他文件格式如 PDF、Word、PPT 等也经常用到。这种方式的执行过程跟“用户根据条件查询”类似，只是获取数据后的展现方式不同，是将查询获得的数据保存到某种文件中，而不是在用户界面上直接展现。

(5) 文件下载：将服务器中的文件下载到本机。

(6) 生成报表：类似于“用户根据条件查询”，也是根据用户输入的条件在数据库中检索数据，然后将符合条件的数据展现在界面上。但两者还是有很大差别的，一般的查询基本上还是以记录集的方式展现，以关系数据库擅长的表的方式呈现信息，而报表往往会经过大量的统计运算，按用户需要的格式呈现信息，有时候，这种格式会相当复杂。

2. 非数据库操作

有很多功能逻辑并不直接跟数据库打交道，它们对数据进行运算处理的过程，可能发生在提交数据到数据库之前，也可能发生在从数据库获得数据之后，也可能是跟提交、获取数据同时、交错进行的。

1) 计算

从数据库获得数据后，或者将数据保存到数据库之前，也许需要进行各种计算。这种计算过程可能非常简单，如只是做些字符串截取、加减乘除、数值统计、逻辑与或之类的处理，例如保存订单信息的同时减去下单客户的信用额度；也可能相当复杂，需要大量的业务规则、计算公式才能完成，例如，某生产调度系统，需要根据生产数据、计划数据进行综合运算后才能生成各生产中心的负荷预测表。

2) 显示

从数据库获取数据，经过加工后在界面上显示为用户所需要的信息。有些用于显示的功能逻辑是相当复杂的，如将查询出来的数据处理成用户需要的报表格式，将数据转变成图形等。例如，某工艺 CAD 系统，从数据库获取某种产品的工艺设计数据后，需要进行大量处理才能生成设计模型图。随着用户的要求越来越挑剔，“显示”也越来越重要了。

3) 传递

将数据从一点传递到另一点，可能是从一个界面传递到另外一个界面，或者是从一个用户传递到另外一个用户等。例如，通过消息框，一个用户向另外一个用户发送文字或图片。

需要注意的是，在实际工作中，这些功能逻辑往往是相互嵌套、相互依赖的，例如一次运算过程可能需要断断续续地查询数据，一边运算一边删除数据，一边新增数据，一边修改数据，仅包括其中一种逻辑的功能是很少见的。



案例：增删改查与计算交错进行的功能逻辑

某 ERP 系统 MRP 运算的功能逻辑。

1. 计算准备

- 如果 MRP 锁定标志被锁住,则报错,拒绝执行。
- 置 MRP 锁定标志为 Y。
- 将符合用户输入条件的销售单、采购单、生产单、库存物料、仓库安全库存相关信息复制到运算缓存区。

.....

2. 计算过程

.....

- 从物料主档案中获得当前层级物料的提前期。
- 如果物料主档案中该物料的“物料生产类别”以 P 开头,则置“是否采购件”为 Y,否则为 N。
- 计算计划开始日期,公式:计划开始日期=需求日期-提前期。
- 根据 MT005 中的参数“MRP 需求合并天数”或 MRP 运算参数,以及计划开始日期,计算“建议区间开始日期”“建议区间结束日期”。
- 根据 MT005 中的参数“MRP 建议日”或 MRP 运算参数,以及“建议区间开始日期”,计算“建议开始日期”。

.....

- 如果“是否采购件”为 N,且层级小于或等于用户输入的层级,且“已经分解”为 N,则进行以下计算。
- 根据物料主档案定义的“标准工艺路径”获得物料的工艺路径,如果没有定义,则根据物料主档案定义的“工艺路径搜索规则”搜索工艺路径,搜索成功后将工艺路径头插入表 PP728,将工艺路径步骤插入表 PP729,置“存在工艺路径”为 Y,将工艺路径号记录在表 PP721 中。

.....

- 一个物料计算完成,将计算阶段信息打印到日志文件。
- 循环计算,直到所有物料计算完成。
- 根据 MRP 参数“采购单满足需求的优先顺序”分配可用数量,如果参数为“A-需求日期急优先”,则按“建议开始日期”增序分配,如果参数为“B-数量小优先”,则按“毛需求数量”增序分配,如果参数为“C-数量大优先”,则按“毛需求数量”降序分配。

.....

- 将 PP721 中,净需求大于 0,相同“物料+建议开始日期”的记录汇总,插入 PP722。需要汇总的字段包括:分配采购单数量、分配生产单数量、分配库存数量、净需求数量。

3. 计算结果处理

- 采购件生成采购建议,写入表 PR038、PR040;加工件生成加工建议,写入表

PP038、PP040。

.....

- 清除表中存储的计算过程中间数据。
- 释放 MRP 锁定标志。
- 发站内消息提示用户计算完成。

5.3.2 数据流

面向数据库的软件,总是以数据为核心的,所有的功能都可以理解成对数据进行处理的过程。可以把数据库理解成一个仓库,表是仓库中的货架,数据是存放在货架中的货物,现实仓库里的货物是在不断流动的,进进出出,在供应商、制造单位、客户间流动,数据也一样,也是在不断流动的,在各个功能之间流动——这就是所谓的“数据流”。当然,数据流动跟货物流动是有本质区别的,数据流动是信息的流动,不是实物流动,是个复制的过程,不会影响原始数据。例如,两个人谈话,甲告诉乙这本关于需求分析的书写得很好,乙就知道了这个信息,这个信息就由甲复制给了乙,这是信息流;如果同时甲把这本书送给了乙,那么这本书就从甲手上流动到乙手上,乙有了,甲就没了,这是物流。

一个软件系统只要在使用中,其中的数据就在不断流动,如果数据静止不动了,那么往往表示这个系统死了——或者软件宕了,或者系统没人用了。数据可以从一个数据库流动到另一个数据库,从数据库流动到文件,从数据库的一个表流动到另一个表,从数据库流动到界面,从界面流动到数据库中,总之每一个功能的执行,都会带来数据流动。



案例：用户登录的数据流

分析一下用户登录过程的数据流。用户单击“登录”按钮后,系统根据用户名从数据库中找到存储的密码(密码是通过“用户管理”功能流入数据库的),密码就从数据库“用户”表中流动到界面,比对正确后,系统打开用户首页,加载用户的功能菜单,关于功能菜单的数据就从数据库的“功能菜单”表流动到界面(菜单是通过功能“用户权限配置”功能流入数据库的),并且记录登录日志,用户名就从用户界面流动到数据库的“登录日志”表。

从这个案例可知,数据流关心这些方面:数据是如何流入数据库的,是如何流出数据库的,数据库中已经存在哪些数据,这些数据是从哪里来的。

有一种表达数据在功能之间流动的模式图,叫作数据流图,它表达了触发者(大部分情况下指某种角色的用户或其他软件系统)在调用软件功能时,数据的存储、流动方式。

相信读者或多或少都对数据流图有些了解,如图 5-3 所示。

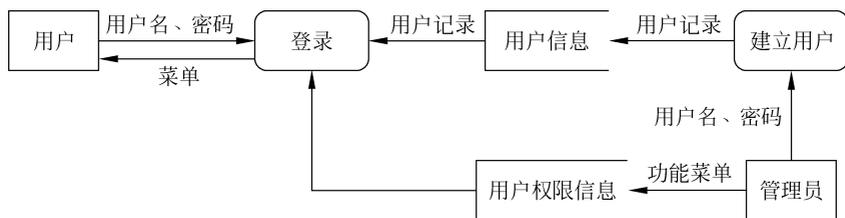


图 5-3 用户登录的数据流图

面向数据库的软件,由于数据存储、流动都是围绕数据库的,因此,数据流图大部分都是表达程序功能是如何与数据库交换数据的,往数据库写入了什么数据,从数据库取出了什么数据。由于有了数据库的存在,导致数据流动的复杂程度大幅度降低,在进行数据流分析时,完全可以不管这个功能需要的数据是从哪里来的,只要知道在数据库的什么地方可以找到就行了,也不需要知道这个功能产生的数据会给谁用,只要知道往数据库的什么地方按什么规范存储就行了。因此,面向数据库的应用开发,很少见到画数据流图进行功能逻辑分析的,大概觉得有些得不偿失吧。

但要知道,不画数据流图并不表示数据流分析不重要。其实在数据建模阶段就应该对数据的流动方式进行深刻分析,因为数据流动的方式不同,会严重影响到数据模型的设计。数据流跟数据、功能、界面都是紧密相关的,考虑到这一点,我们至少明白了一个道理,本书所说的各种步骤其实在实际工作中都应该是交错穿插进行的,绝不能割裂开来考虑。

5.3.3 workflow

1. 工作流程

企业所有的工作都是由或多或少的一些步骤构成的,每个步骤又可以分成更小的步骤,步骤之间有一定的先后顺序,这些有先后顺序的工作步骤就构成了所谓的工作流程。一般管理比较规范的企业,会有详细的业务流程图来描述自己的工作流程。

在实际工作中,有些工作流程要求属于理论上的,可能只是一种设计理念,不会或根本不可能强制执行。



案例：理念性的工作流程

某部门对员工上班的准备工作有明确的步骤要求：员工上班打卡→打开计算机打印昨天的业绩报表→向主管汇报昨天的工作情况→获得主管安排的工作任务。虽然大家每天上班基本上都是这么做的,但确实有很多特殊情况,如员工有可能忘记打卡了,有可能今天主管没有安排工作任务,有可能自己出差,等等。因此,这种工作流程只能算是一种设计理念,并非真正的有严格约束力的工作流程。

有的时候,针对某些工作流程的要求,企业有非常明确、严格的规定,可能在企业的任何一份工作操作指导书、业务流程图之类的文件中都有关于它的正规阐述,但因为执行力的问题,在真正的工作过程中未必就是这样做的——这种事情太多太多了。



案例：未必得到执行的工作流程

某公司的员工请假流程,根据人力资源部颁布的考勤管理规定,严格分为这么几步：员工填写请假单→主管审批→部门经理审批→人力资源部登记。这种流程看上去很严格,至少员工不填写请假单,就不可能有主管审批,因为主管需要在请假申请单上签字,而没有主管、部门经理审批的请假单,人力资源部也不会认可。然而,所有的流程都是需要人来执行的,如果不借助于某些手段——或者管理手段(如考核措施),或者技术手段(如信息化系统)——什么流程都有可能失去它的约束力。例如,这个公司确实有不少人因为跟人力资源

负责考勤的文员关系好,会趁主管出差,什么单子都不填写,直接跑过去跟她打个招呼就早早回家了。

管理者为了加强控制,确保工作流程的要求得到真正的执行,在管理的过程中往往都会在固化流程上面付出巨大的努力,传统上可以通过培训、考核、检查之类的方式来进行,但有了信息化系统后,相关的工作方式发生了巨大的变化。将流程固化,强制按照预先设好的步骤执行,不留情面——在这方面机器比人要强得多。

但是,并非所有的工作流程都可以固化,需求分析者不要看到别人拿出个业务流程图就想将它变成信息化系统中的固化工作流程,很多情况下这都是不现实的。例如,以软件行业为例,很多软件公司都是这样描述自己公司的软件实现流程的:调研→需求分析→软件设计→开发→测试→上线→实施→服务。但正如我们一再强调的,软件项目的这些步骤往往都是穿插进行的,根本不可能用一个流程将之固化下来。

2. 什么是工作流

管理软件中所谓的工作流,往往是指这样一种功能逻辑:它建立了工作流程中各步骤的制约关系,规定工作流程需要经过哪些步骤,进入或完成每个步骤有什么条件,每个步骤应该由什么人负责办理等。

由于软件承载了管理者对于固化流程的强烈期望,包含工作流的功能逻辑在管理软件开发中越来越常见了,特别是在一般的OA系统中,工作流往往会成为整个系统运行的核心功能,常见的办公流程包括“请假流程”“用车申请流程”“会议室申请流程”“办公用品领用流程”等。

为了节约成本、快速响应,许多OA软件提供一种称为“工作流引擎”的功能组件,通过这种组件,可以快速配置用户需要的不太复杂的特殊工作流。一般工作流引擎包括两大部分内容,一是表单处理,一是流程处理。表单承载业务信息,如用户填写的请假单、出差申请单等,流程承载的是办理过程。可以将工作流引擎简单理解成,将承载信息的表单在各个流程节点之间推送的机制。另外,为了配合工作流的处理,需要提供一些基本功能,如“待办任务”“已办理任务”“流程监控”等,这些在工作中遇到的话自然就无师自通了,不再赘述。

下面简单介绍一些跟工作流相关的基本概念。

节点:流程的节点就是工作的处理步骤,一条工作流至少包括一个节点。在进行工作流设计时,对于节点的处理,有两个重要的注意点:一是并不是实际工作中的所有步骤都应该在系统中处理,例如某公司用车申请流程,其中包括一个重要的步骤,就是确认驾驶员是不是有时间,这个步骤可能在系统中就没办法处理,只能由相关人员在系统外(所谓的“线下”)确认;二是节点的设置要有一个合理的粒度,一般情形下,相连两个节点之间的办理人应该是不同的人,如果相连两个节点都是由同一个人办理,那么往往意味着节点的粒度设计太细,需要将这些节点合并。

发起:发起流程就是启动一件事情,让某一件事按照设计好的工作流要求走起,例如在系统中发起请假申请,发起会议室申请等。

分支:有些工作流的所有步骤都是确定的,由启动到完成一定会经过所有节点,如某公司“用车申请流程”,一定是经过“发起申请→主管审核→后勤部安排车辆”这些步骤,但这种工作流其实并不多见,大部分工作流的步骤并不完全确定,需要在实际执行过程中根据某些

条件、要求才能确定,也就是说会出现某种“分支”。例如,某公司“采购物品流程”规定,如果采购金额不足 10 000 元由分管副总审核,超过 10 000 元需要总经理审核,超过 100 000 元需要董事长审核,显然这里有两个分支。在实际处理过程中,对于分支的处理,有时候由系统自动判断,如果满足什么条件则执行什么步骤,有时候由节点的办理者人为判断下一步执行哪个节点、由谁来执行。前面的“采购物品流程”,当分管副总审核通过时,可以由系统根据申请采购金额判断是结束本工作流还是继续送往总经理审核,而有的时候,由于采购物品金额并不明确,系统无法判断,也可以由审核人自己做出判断,决定是否需要送给总经理审核。

办理: 办理指 workflow 节点的处理过程。很多节点的工作只是决定是否同意,可以将这个办理过程称为“审核”或者“审批”,如某公司“请假流程”中的节点“主管审核”“总经理审批”。一般系统会为工作流的待办事项生成待办任务,处理者从自己的待办任务中可以知道要处理哪些事情,然后打开某条待办任务进行办理,办理完成后再推送到下一节点,这时系统再给下一节点的负责人生成一条待办任务。当然,信息系统只是管理信息的,很多情况下所谓的办理,可能只是在系统中登记办理结果,真正的办理过程还是在系统之外(线下)。例如,某公司“办公物品领用流程”,其中有“仓库发货”的节点,在实际执行过程中,只是仓库管理员将物品发放完后通过本节点登记,并非通过这个节点可以真正办理发货这件事。

委托: 委托就是办理者将某一个任务,或某一类任务,或某时间段内的所有任务委托给其他人处理。这种事情往往发生在办理者外出,或者觉得自己没有能力,没有职权处理的时候。

主办人与经办人: 流程节点的办理者可以包括主办人与经办人。在现实业务中,这往往表示需要一个工作团队,由经办人协助主办人完成这项工作;在软件功能逻辑中,一般的处理方式是,经办人可以报告办理情况,但不能将工作流向后推动,只有主办人才可以执行完成节点的操作并将工作流向后推动。

子流程: 工作流中,在办理某些节点时,可能会启动另外一条相对独立的流程,我们称这种流程为子流程。例如,某公司“办公物品领用流程”,在“仓库发货”这个节点,仓库管理员发货时,如果发现希望领用的物品不存在,那么会在这个节点发起另外一条流程,“办公物品采购申请流程”,这就是子流程。当这个流程办理完毕,仓库收到采购回来的办公用品后,仓库管理员才会将“仓库发货”节点办理下去。

会签: 会签指某个节点由多人同时办理,这个节点的办理结果是综合这些人的意见得出的,常见的功能逻辑包括所有会签人同意就同意,任意一个人同意就同意,达到某百分比的人同意就同意,任意一个人不同意就不同意,等等。

3. 工作流图

当功能逻辑中包含工作流时,往往意味着需要某种控制或推送机制,这就需要需求分析师去分析,这项工作在办理过程中经过哪些节点,由谁发起,每个节点往下推送时是否会有分支,这些分支需要满足什么条件,每个节点满足什么条件才能进入,满足什么条件才能流出,谁是主办人,谁是经办人,办理时是不是可能委托他人办理,每个节点是不是还有可能分解出另外的子流程,等等。工作流程经过需求分析后,需要形成文档,跟工作流相关的文档最主要的就是流程图,本书将面向工作流的流程图称为工作流图。



案例：工作流图

某公司的员工请假管理要求如下。

一般员工请假,如果请假天数不足1天,就由他的主管(直接上级)审批;如果请假天数多于1天不超过7天,不但要主管审批,还需要部门经理审批;如果请假天数多于7天,则需要主管、部门经理、总经理审批。

主管级别的员工请假,不需要主管审批这一步。

部门经理级别的员工请假,不需要主管审批、部门经理审批这两步。

总经理不需要请假。

需求分析人员根据业务需求进行了工作流的设计,并绘制了工作流图,如图5-4所示。

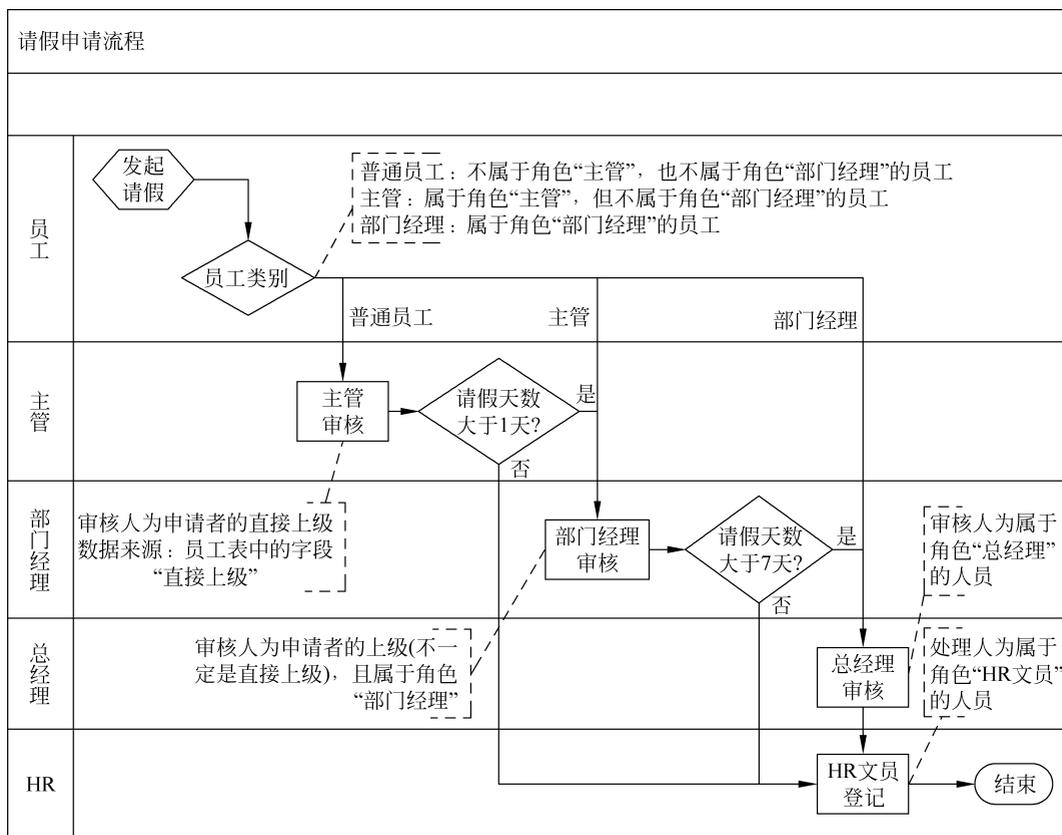


图5-4 请假申请工作流图

由于工作流图是可以投入研发的设计图,它的要求更规范、更明确、更精准,应尽量避免模糊的概念性的描述,以及容易引起歧义的描述。

画工作流图有以下一些注意点。

(1) 工作流图只表达流程步骤,业务信息还需要采用其他表达方式。如上述案例中,请假单中需要填写什么内容,遇到年假、婚假之类的请假如何处理等,就不在工作流图中表达。

(2) 工作流图中的每个矩形表示一个流程节点,需求分析者在绘制流程图时就应该有

一个设计节点的思维过程,不是要求开发者开发的节点,就不要画个矩形在图上。

- (3) 每个菱形表示有一个流程分支。
- (4) 需要把每个分支的判断规则写清楚,清楚得可以根据它写程序。
- (5) 要把每个节点的办理人员写清楚,清楚得可以根据它写程序。

5.3.4 一些功能逻辑案例

本节介绍一些功能逻辑案例,这些案例在软件系统中经常出现,包括权限控制逻辑、结转逻辑、可变逻辑、自定义逻辑等,只要你从事管理软件设计、开发,相信这些功能逻辑都是会频繁遇到的。

1. 权限控制逻辑

所谓权限控制,就是控制用户使用指定功能或数据的权限。这种逻辑一般可以分成两种,一是功能权限控制逻辑,一是数据权限控制逻辑。在实际工作中,用户对权限控制的要求千差万别,往往需要结合功能权限控制与数据权限控制才能满足需求。

1) 功能权限控制

所谓的功能权限控制,就是决定用户能否操作某功能。例如,是否允许打开某功能菜单,是否显示某功能按钮,是否可以打开某子页面等。



案例：功能权限控制逻辑

某软件建立了一套功能权限控制体系。所有的权限控制逻辑根据用户所在的“角色”展开,软件根据角色授权,如果某用户被分配到了某个角色,那么表示这个用户拥有了这个角色的所有权限,如果一个用户允许属于多个角色,那么,意味着这个用户获得了这些角色的所有权限。对角色授权分为三级授权,第一级是功能菜单,用户只能看到获得授权的功能菜单,也就是说,对于某用户来说,如果他所在的角色中没有获得某功能菜单的授权,那么他就不能使用该菜单下的所有功能,因为他根本看不到这个菜单;第二级是子页面,每个功能菜单下都有若干子页面,用户获得功能菜单的授权后,未必可以使用本功能点下的所有功能,他只能看到获得授权的子页面;第三级是功能操作,也就是常见的按钮、功能图标、操作链接等,对于没有授权的功能操作,即使打开了子页面,用户也不能操作——有的隐藏,有的置灰,有的单击后提醒不能操作。

2) 数据权限控制

所谓数据权限控制,指控制用户可以对哪些数据进行操作,也就是说,即使用户获得了某些功能的授权,也不一定有权对这些功能管辖的所有数据进行相应的操作。例如,常用的新浪微博,绝大部分功能都包含典型的数据权限控制逻辑,这么多用户使用完全相同的功能,但每个人只能查看跟自己相关的数据(如微博、粉丝、关注者、收藏等),大量跟自己无关的数据,普通用户是没有权限访问的。



案例：数据权限控制逻辑

某 OA 软件有一个“工作计划管理”功能,获得授权的用户可以通过本功能管理自己的工作计划,可以新增工作计划,可以对自己的工作计划进行增删改查,可以根据自己的工作

计划编写完成情况等。但对于管理者来说,不但可以管理自己的工作计划,还可以查看所有下级的工作计划,并且可以对下级工作计划的完成情况做出评点。

一般来说,数据权限控制的处理比功能权限控制要难得多,毕竟功能是有限的,而数据是无限的,如何既能满足用户的权限控制需求,又能应对未来可能发生的权限变更,又要节约开发成本,这一直是让软件设计者颇感头痛的课题。

2. 结转逻辑

所谓结转逻辑,指针对某些数据进行统计汇总或相关运算后,将计算的结果保存到数据库中用以存档或者提高系统性能。结转可以分成两种方式,一种是结转完成后锁定原始数据,不允许修改、删除,这种结转方式可以确保原始数据与结转结果始终都具有一致性;另一种是结转完成后不锁定原始数据,在这种方式下,如果原始数据发生变化,就会导致原始数据跟结转结果不一致,当然这种方式的处理逻辑比前者要简单得多。结转逻辑在实际工作中非常常见,一般会用在应对需要出具正式、严格的期间数据,需要提高检索性能等方面的需求,如财务管理中的结账,库存管理中的生成月报,数据仓库中的数据汇总等。



案例：结转逻辑

某财务软件,提供月结功能,用户可以每月进行一次结账操作。用户确认月结后,系统会根据这个月的记账凭证信息以及上个月结账的结果,计算生成这个月的结账结果,包括每个会计科目的上月结存、本月借方发生额、本月贷方发生额、本月结存等信息。结账后,这个月的所有记账凭证被冻结,不允许修改。有了结账记录后,资产负债表可以根据本月结账记录生成,不需要对历年所有的历史数据进行一次汇总运算,这样可以降低系统运算的资源消耗,不会因为数据量的膨胀而导致运算速度急剧下降。另外,从系统生成的会计报表需要报送相关机构,成为非常严肃的档案,在这种情况下必然要求赖以生成报表的原始数据不能被修改,否则会造成非法的会计报表。通过结账后锁定记账凭证,可以保证报表跟原始数据具有一致性。

3. 可变逻辑

所谓可变逻辑,指某种功能的逻辑并不确定,怎么处理和运算交由用户决定。常用的方式是通过某种参数或配置开关来实现。例如,某发布生产任务单的功能,主要的逻辑是将编排过的生产任务单发布成正式的生产指令,但这里有一个配置开关“是否给责任人发送短信提醒”,这个开关由用户自己设置,如果值为“是”,则系统在发布生产任务的同时还会给相关责任人发送短信提醒,否则就不发短信。

当然,可能从程序实现的角度来看,这貌似只是多了个条件判断的分支,但从用户的角度来看,这确实是一种可以随心操控的可变逻辑。可以由用户操控的可变逻辑在实际工作中用得相当广泛,越是大型的软件产品这种逻辑用得越多,因为只有这种逻辑才能够支撑不同客户的各种各样的要求。



案例：可变逻辑

某库存管理软件的账务模块,有一个重要的功能:库存价值计算。但不同的客户对库存价值的计算方式有不同的要求,如先进先出法、后进先出法、移动加权平均法等。为了应

对不同客户的多变需求,该软件的处理方式是设置一个用户配置参数,当需要系统计算库存价值时,用户需要先配置好这个参数,系统会根据用户配置的方式进行相应的计算,不同的方式,计算过程千差万别,需要的数据、计算步骤、数据流、保存方式等都完全不一样。

4. 自定义逻辑

所谓自定义逻辑,是指用户自己根据系统提供的某些功能定制工具,按照系统要求的语法、规范,而设置、编写的适合于自己需要的功能逻辑。有些自定义逻辑很简单,如允许用户嵌入查询 SQL 语句的查询生成器,允许用户通过设置正则表达式来进行数据录入验证;有些自定义逻辑相当复杂,不是一般的团队能做得出来的,如 Microsoft Office 的 VBA 编程工具,有些大型管理软件提供的报表开发工具等。当然,能使用自定义逻辑的用户并不是普通用户,他一定具有某种程度的软件造诣,而那些能够驾驭非常复杂的自定义工具的用户,恐怕只能认为他们也是软件开发团队的成员了。



案例：自定义逻辑

某 OA 软件,提供了工作流自定义工具,用户可以根据自己的业务流程配置出属于自己的工作流,而不需要软件供应商介入。该工作流自定义功能主要分成两大部分,一是流程自定义,一是表单自定义。用户根据工作流需要处理的信息自定义表单,该工具提供了录入框、单选框、复选框等常用的界面组件;根据工作流需要处理的步骤自定义流程,该工具提供了图形化的流程定义方式,用户可以通过拖拉节点、添加节点连线等方式确定流程路径及各种流程分支。该工具提供了大量的语法规则,用户可以根据这些语法规则编写判断语句用以确定工作流的分支条件、办理人员等。



5.4 功能优化

作为软件设计者,在进行功能设计时要时刻警醒自己——难道就没有什么更好的方法来处理这件事了吗?对于软件设计来说,设计的功能能够满足用户的需求只是最基本的要求,在这个基础之上应该还有更高层次的追求——我们不但提出解决方案,而且力图提出最优解决方案。本节从功能的灵活性、可重用性、高效性三个方面谈谈如何进行功能优化。

5.4.1 灵活性

软件的灵活性,指软件应对需求变化的能力。无论开始的需求工作做得多完善,用户的需求终究会有变化,或者需要修改某些需求,或者需要增加某些需求,或者需要取消某些需求,而灵活的软件具有柔性,即使用户的需求发生了变化,软件不修改也可以解决很多问题。当然,任何软件都不可能解决所有问题,作为设计者,能做的是尽量设计出可以应对更多需求变化的软件,也就是尽量增加软件的灵活性。

为了提高软件功能的灵活性,设计时可以试着问问自己以下这些问题。

(1) 这个地方一定要写死吗?

一般情况下,在程序代码中是不应该出现数据的,一旦在代码中出现了数据,就称之为写死了某些东西。例如,某审核功能只能总经理才能操作,但如果在代码中直接写“If

CurrentUser.Name='王老板'”之类的语句,就把这个规则写死了,写死的结果就是大大降低了软件的灵活性,什么时候总经理换成了张老板、李老板,就不得不修改软件了。当然,也必须承认,软件中总要写死一些东西,越是功能强大、规则丰富的功能,写死的东西就越多,因为在许多情况下,写活比写死要付出成倍的努力。

作为设计者,当需要写死某些规则时一定要问自己:这个地方一定要写死吗?写活不需要理由,写死一定需要理由,有说服力的理由。



案例：写死需要充足的理由

某销售管理系统,有一个分析客户信用额度的功能,需求概要是这样的:在近两年中如果客户成交金额超过100万,就归为A类客户;如果成交金额超过50万不到100万,就归为B类客户;如果成交金额超过10万不到50万,就归为C类客户;其他客户为D类客户。A类客户,如果近三个月内有销售记录,且结清货款,那么授予10万的信用额度;B类客户,如果近两个月内有销售记录,且结清货款,那么授予5万的信用额度;C类客户,如果近一个月内有销售记录,且结清货款,那么授予3万的信用额度;D类客户不允许赊账。

分析这个需求,发现规则里出现了大量的数值,可以考虑两个大的处理方向,一是在实现时写死这些数值(可能程序员在开发时会通过一些常量来处理,这不是本书考虑的内容);一是做一个规则配置功能,针对下单额度、信用额度、下单日期区间等进行配置,程序围绕这些配置结果进行运算,而不会把“100万”“50万”之类的区间额度写入代码中。分析这两种处理方式,明显前者实现起来要容易,但不够灵活,一旦这种区间规则发生了变化就需要修改程序;后者实现困难,但比前者灵活,如果区间额度发生了变化,用户只要修改配置结果就行了,不需要对软件进行任何更改。至于最终采用哪种方式,需要设计者去决定,如果考虑灵活性,考虑未来可能的变更,自然选择后者,如果考虑开发成本、开发速度,自然选择前者。不同的考虑出发点会有不同的方案,关键是决定写死这些数值时要问问自己,这个地方一定要写死吗?如果答案是“是”的话,需要有充足的理由。

在实际工作中,把案例中的这种数值区间写死的很少,一个正常的软件设计者简直无法忍受这种方式。但有些特殊情况写死也比较常见,如某种记录的状态(例如采购单的“审核通过”“审核不通过”状态——程序中未必就会写这种中文描述,有可能使用“A、B、C”“0、1、2”之类,下同),某种跟业务规则相关的业务数据类型(如员工的“在职员工”“离职员工”类型),某种配置型的数据(如存货价格核算方式包括“先进先出”“加权平均”等),某种特殊的角色(如只有属于角色“总经理”的人员才能进行某种操作),某种工作流程的步骤(如请假流程经过“提出申请”“主管审批”等步骤),等等,这种数据往往跟某些业务规则息息相关,不写死无法表达规则,或者虽然可以写活但会付出几倍的代价,让设计者觉得得不偿失。



案例：写死与写活的权衡

某人力资源管理系统,其中的“绩效考核审核”功能,需求是当考核者完成自己班组的考核时,需要将考核结果提交部门经理审核,部门经理审核完成再提交人力资源部经理审核,审核通过的考核成绩才会成为最终的考核结果。

这种情况下,设计者处理起来就比较挠头。这个审核过程有两个步骤,对于一个企业来

说,这种审核步骤一般是很少变动的,如果把两个步骤写死是完全可以接受的,如果要写活,那么要考虑设计可以让用户配置审核步骤的功能,如果没有一个像样的工作流引擎之类的组件,这项工作的开发量是相当大的。在这种情况下,如果是为某企业量身定制的功能,恐怕第一选择就是写死这个审核步骤,毕竟花那么大代价追求一个几乎用不到的灵活有些不划算。但如果是要开发一个产品投入市场,恐怕就要好好研究下如何写活了,毕竟不同的企业审核方式可能千差万别,想靠一个规则打天下显然是行不通的。

(2) 这个规则是必需的吗?

每个功能都有其逻辑规则,虽然逻辑规则越多往往意味着软件功能越强大,但也要知道,逻辑规则并非越多越好。设计时,在功能逻辑中应该只保留必需的规则,对于那些不是必需的规则,能去除就去除,能简化就简化。



案例：规则越少越好

某库存管理系统,有一“禁用物料”功能,被禁用的物料不允许用户在入库过程中使用,核心规则是置物料属性“禁用标志”为“Y”。有以下两种基本思路可以考虑。

思路一,当用户确认禁用某物料时,系统判断该物料是否已经被禁用,如果已经被禁用,则提示出错“该物料已经被禁用”,如果本物料没有被禁用,则禁用该物料,并提示“禁用成功”。

思路二,当用户确认禁用某物料时,不管该物料是否已经被禁用,直接修改它的“禁用标志”为“Y”,并提示“禁用成功”。

这两种思路处理的最终结果是相同的,就是物料的“禁用标志”被置为“Y”,只是思路一先判断该物料是否已经被禁用,思路二不做这种判断,根据简化规则的原则。应优先考虑思路二,少一个规则就少一分软件变更的风险,谁知道这个判断物料是否禁用的规则将来会不会因为某些现在意想不到的情况而发生变化呢?

为了简化规则,提高软件灵活性,在软件设计中的一个常用技巧是,将某些控制、验证之类的规则交给用户,当然,前提是用户没有做好这种工作不会造成灾难性的影响,并允许用户在发现错误后自己纠正。在实际工作中,特别是针对某些参数开关之类的配置型数据的修改,经常会用到本技巧。



案例：让用户负起验证责任

小王在某公司进行库存管理系统的设计。客户的原材料仓库分成两个仓库,一个是主料仓库,一个是辅料仓库,主料仓库中存放本公司生产过程中用到的主要原料,如塑料粒子、铜材等,辅料仓库中存放辅助原料,如螺丝、螺母等。由于这两个仓库除了存放的原料不同,其他的管理方式都相同,小王决定只设计一个仓库管理的功能,然后在仓库信息中加一个参数“允许存放的物料种类”,“A”代表只能存放主料,“B”代表只能存放辅料,为了将来有可能的需求扩展,另外用“C”代表两者都可以存放。

问题来了,当系统使用一段时间后,如果用户要修改这个参数,软件该如何处理呢?假设某仓库一开始时参数“允许存放的物料种类”被设为“A”,使用一段时间后,用户发现这个配置有误,需要将“A”改成“B”,也就是说由只能存放主料改成只能存放辅料,这时候该怎么

处理呢？

思路一，系统判断本仓库中是否存在主料，如果有，则报错，不允许修改，需要用户通过出库、调账、移库之类的方法将本仓库所有主料的结存数清零后才允许继续操作。

思路二，用户修改本参数时，软件不做判断，用户可以直接将本参数由“A”改成“B”，当然，用户在修改之前，他应该去查询下本仓库的结存情况，如果本仓库存在主料的结存，他就不应该做这种操作，但他要是强行修改，系统也不管。

比较这两种思路，思路一可以保证仓库中物料存放的正确性，但多了验证规则，降低了灵活性，因为如何判断是主料还是辅料、是否有结存之类的规则，将来都是有可能发生变更的。

思路二将验证的过程交给了用户，软件由于不需要提供验证功能，具有了更大的灵活性，但如果用户没有负起验证的责任，就有可能造成这种数据现象——明明配置的参数是只能存放辅料，可仓库中偏偏有主料存在。不过对系统而言这种错误没什么大不了的，用户发现后就可以处理，不会有多严重的影响。

(3) 这个地方用户需求真的很明确吗？

一个有经验的需求分析者，非常明白一个道理，跟你提需求的用户很少能把需求说得很明确，如果他描述的需求真的很明确，那更要谨慎对待，因为在很多情况下不是需求明确，而是很多异常情况被遗漏了。在设计过程中，要对各种需求有很强的敏感性，对于不明确的需求要搞明确，对于明确的需求，要追问自己：这个需求真的明确吗？

在很多情况下，用户真的不能把需求明确下来，他没有信息化建设的经验，他对未来的信息化管理体系的理解仅仅来自于书本、网络上的苍白描述，别的一无所知。当然，对于需求分析者，有责任引导用户逐步明确需求，但有的时候真是神仙也没办法，他也说不清楚，你的经验也无法继续深入，怎么办呢？这时候就要在软件的灵活性上下功夫，希望能够兼容一些不明确的需求。



案例：需求兼容

某学校管理系统中的调课功能。在设计阶段，就系统中调课成功后是否需要发短信通知任课老师与辅导员，学校管理方意见不统一，有说不需要通知任课老师，因为调课这种大事需要教务处电话通知相关任课老师，发短信容易引起依赖心理，反而可能造成教学事故；有说需要短信通知任课老师，毕竟多一个提醒的渠道总是好事；有说需要通知辅导员，好让他们对学生的课表有个了解；有说辅导员不负责学生的教学工作，通知他们没有意义，有些辅导员带十多个班，收那么多短信有什么意义？

争论了好久也没有什么定论，最后设计者决定加两个参数开关，一个是“调课后是否发送短信给任课老师”，一个是“调课后是否发送短信给辅导员”，这样，当前阶段就无须争论这个话题了，到软件投入使用时，想发短信给任课老师也好，想发短信给辅导员也好，用户都可以自己决定——这种通过参数开关兼容不同需求的方法，在实际工作中相当常见。当然，兼容的同时也增加了工作量。

(4) 这个地方用户需求发生变化的可能性大吗？

也许用户的需求非常明确，但设计者还要考虑，这种需求变化的可能性大吗？有些需求

变化的可能性小,不需要花成倍的成本将软件搞得过于灵活,例如,一些需要符合行业规范、国家规定的需求(如会计报表的生成方式,ISO 文件体系等),变化的可能性很小;而有些需求变化的可能性大,这时候设计者就需要在灵活性上多考虑,避免因为需求的频繁变化导致软件需要不断修改。



案例：变化可能性大的需求

小王在某学校做学校管理系统的设计工作,在“学生档案建立”这个功能中,学校管理方对学生档案的信息非常重视,要求分两级审核机制,班主任负责录入学生档案信息,然后送交院系审核,院系审核通过后,送交学工处审核,学工处审核完成的才会成为真正的学生档案信息。

小王分析了这个需求后,觉得这个审核流程在未来发生变更的可能性很大,如院系可能会让不同的科室审核不同的信息块(基本资料、家庭资料、联系方式等),如可能会让教务处审核学生的班级信息,可能会引进档案室审核、院长室审核等。于是,小王决定设计审核流程配置功能,将来一旦发生了审核流程的需求变更,用户可以直接通过修改配置处理,而不需要开发者修改程序。当然,这么做的开发难度、工作量增加了许多。

(5) 这个地方我抓住了业务的核心吗?

有时候,用户所提需求可能只是描述了业务过程中的某个特殊场景,远远没有触及业务的核心。例如,仓库管理员说他需要收取供应商送的货物,需要收取车间送来的半成品,需要收取包装车间送来的成品,要知道这些工作的业务核心只有一个,就是将物料入库。从调研的结果来看,这个业务核心目前有前述的三个特殊场景。作为设计者,在进行功能设计时要善于甄别这些特殊场景,透过现象看本质,一次力求解决“一类”问题,而不是“一个”问题。抓住业务核心设计出来的功能,有更大的灵活性,因为只要属于这个业务核心中的需求都可以通过这个功能满足,而如果是针对特殊场景设计的功能,就会欠缺灵活性,因为在使用过程中可能会面临许多不可预见的特殊场景,每个意想不到的特殊场景都会导致软件功能的变更。



案例：抓住业务核心

某工厂的装配车间,需要通过信息化系统建立装配工的计件工资体系。该车间有三个装配组,每个装配组包括1名装配组长与5名装配工人,装配一组负责装配A系列产品,装配二组负责装配B系列产品,装配三组负责装配C系列产品。装配组长的计件工资,除了根据自己的装配数量计算外还包含组长津贴,不同装配组,根据工作的特性不同,给装配组长计算津贴的方式并不一样。A系列的产品,每天装配前需要做比较多的准备工作,而这些准备工作主要由装配组长提前一个小时到工作岗位处理,因此车间会给一组组长发放定额津贴;B系列的产品,没有很多的准备工作,组长的主要工作在于工作过程中的组织与技术指导,因此车间会按照二组每天装配数量的百分比计算二组组长津贴;C系列的产品,每天下班后,需要做很多的清扫、整理工作,这些工作由组长在下班后加班半小时完成,车间除了按照每天各自组的装配数量的百分比提取三组的组长津贴外还会另外发放一定的定额津贴。

为了计算三个组长的计件工资,第一反应,设计者可以建立以下三种计算模型。

一组组长计件工资=亲自装配数量的计件工资+定额津贴

二组组长计件工资=亲自装配数量的计件工资+本组计件工资总数×比率

三组组长计件工资=亲自装配数量的计件工资+本组计件工资总数×比率+定额津贴

但是,这种方式显然没有抓住业务核心,会导致功能缺乏灵活性,例如,一旦什么时候一组组长的计件工资也要跟本组工作量相关时就需要修改了。这里的业务核心是,装配组长的计件工资是由三部分构成的,一是自己亲自装配的计件工资,一是跟装配组工作总量无关的绝对津贴,一是跟装配组工作总量相关的相对津贴。至于二组组长没有绝对津贴,只是因为在这个场景下定额津贴为0罢了,一组组长没有相对津贴,只是因为在这个场景下相对津贴的比率为0罢了。

(6) 这个地方的处理跟业务现实一致吗?

有时候,你会发现,为了解决一个问题可以有各种方案,条条大路通罗马,但往往最优方案只有一个,只不过是不是“最优”在设计阶段并不那么容易判断,需要在系统使用一段时间后(短则一天,长则几年)才能判断。根据经验,当两种方案都属于值得考虑的方案时,越是符合业务现实的方案越具有灵活性——将来发生了需求变更未必就不需要修改,但修改起来更容易一些。跟业务现实不一致的功能逻辑,总是缺少一种水到渠成的感觉,有拼拼凑凑、敷衍了事的嫌疑,看上去像曲线救国,其实蕴含着极大的风险。所谓“强扭的瓜不甜”,偏离业务现实的功能逻辑就是强扭的瓜。



案例：功能逻辑跟业务现实一致

小王在给某公司设计 OA 管理系统。数据建模已经完成,其中有两个跟“人”相关的重要的表,一个是“员工”表,一个是“用户”表,前者表达了一个人作为员工的信息,如所属部门、上级、身份证号、地址等,后者表达了一个人作为系统用户的信息,如登录名、密码、最近登录日期等,这两个表之间的关系是一一对一的关系。

在设计“请假申请”功能时,就如何处理申请人的问题,小王构思了两个方案:方案一,申请人为“员工”表中的员工,当用户提出请假申请时,根据当前用户所关联的员工,将本请假单关联到该员工,而不是关联当前用户;方案二,申请人为“用户”表中的用户,当用户提出请假申请时,直接将请假单关联到当前用户,而不是所关联的员工。是以员工的身份申请请假,还是以系统用户的身份申请请假呢?虽然这两种方式都是可行的,但小王经过思考还是决定采用方案一,因为这个方案更接近于业务现实。在系统中,虽然请假是由某个用户发起的,但“用户”本身所蕴含的业务现实是,这是一个进行软件系统操作的角色,登录、注销、权限之类的信息是属于用户的,而与现实员工相关的信息不应该属于用户。请假,是一个公司进行员工管理的业务,显然以员工的身份请假更符合实际。再想想未来可能的变更,假设未来要求允许将用户转换给其他员工,或者允许一个员工分配多个用户,如果采用方案二就麻烦了,因为用户本身承载了请假信息,这种变更对历史数据是致命的,但如果采用方案一,变更起来就很简单,甚至可能都不需要任何修改。

5.4.2 可重用性

可重用性,指本功能对不同系统环境的适应性。有些功能自成体系,跟本功能之外的功

能、数据没有任何关系,具有最强的可重用性,例如,开发者经常使用的各种开发组件;有些功能隶属于某个环境,离开这个环境就没有任何意义,那么这种功能就没有任何可重用性,例如,需要从大量的数据表中抽取数据的报表功能,离开了这些表结构就没有任何意义,自然就没有单独获得重用的可能。注意,这里所说的重用,是指功能级别的重用,跟代码级别的重用(封装函数、过程、类之类)是两码事,不可混为一谈。



案例：功能移植

小王在给某公司设计一款 CRM 软件,其中有一个“客户管理”功能,用以对客户的基本信息资料进行管理,管理的信息包括客户代号、名称、联系方式、联系人等,需要提供录入、修改、删除、导入、导出等子功能。由于小王的团队开发过一款 ERP 软件,其中就包括一个客户管理的功能,小王认真研究了这个功能后,发现这个功能不需要进行任何修改就可以移植到这款 CRM 软件中来。

为了提高功能的可重用性,可以考虑以下这些方面。

1. 尽量减少功能之间的关联性

这里所说的功能之间的关联,是指某功能跟别的功能所产生的数据之间的关联。越是孤立的功能,越具有可重用性。当然,软件作为一个整体,将所有的功能完全孤立自然是不可能的,由于数据的流动,功能之间总有千丝万缕的联系,作为设计者,要做的是尽量减少功能之间的关联性,关联性少了,即便不能直接移植到别的软件系统,也可以通过少量修改达到这个目标。这里所说的“孤立”,不是它跟别的功能没有数据往来,而是如果有数据往来,总是别的功能发起,它自己是不会发起的,如客户管理功能,它生成的数据——客户信息——会被销售订单管理、送货单管理、应收账款管理等许多功能模块使用,但它自己是不会主动发起去访问这些功能模块的信息的。功能模块之间的关联是有方向性的,可以这么说,客户管理跟销售订单管理没有关联,但销售订单跟客户管理是有关联的。



案例：减少功能之间的关联性

小王在设计某个知识管理软件,该软件需要管理大量的用户文档,用户可以在功能点“文档库”或“知识论坛”中上传文档。为了节省存储空间,大概的需求是这样的:当用户通过某个功能上传文档时,系统判断本文档在服务器文件存储系统中是否存在,如果存在,就给它建一个链接直接链接到已经存在的文件,如果不存在,就保存这个文件,然后建立链接。当用户在某个功能中删除附件时,系统删除这个链接,同时判断在其他地方是否有对这个文件的链接,如果没有,则删除服务器中的这个文件,如果有,保留这个文件。例如,用户为文档库中的某个知识节点上传一个文件 A,系统会判断文件 A 在文件系统中是否存在,如果不存在,就把文件 A 保存到服务器,同时在该知识节点下保存这个文件标题,建立一个到文件 A 的链接,如果已经存在,就直接建立一个到已存在文件的链接,对用户而言,这个文件已经上传成功;当用户在文档库中删除这个文件时,系统可以直接删除这个链接,然后再判断文档库中、知识论坛中是否有别的记录中链接了这个文件,如果有,则保存这个文件,如果没有,同时删除服务器中的文件,释放存储空间,如图 5-5 所示。

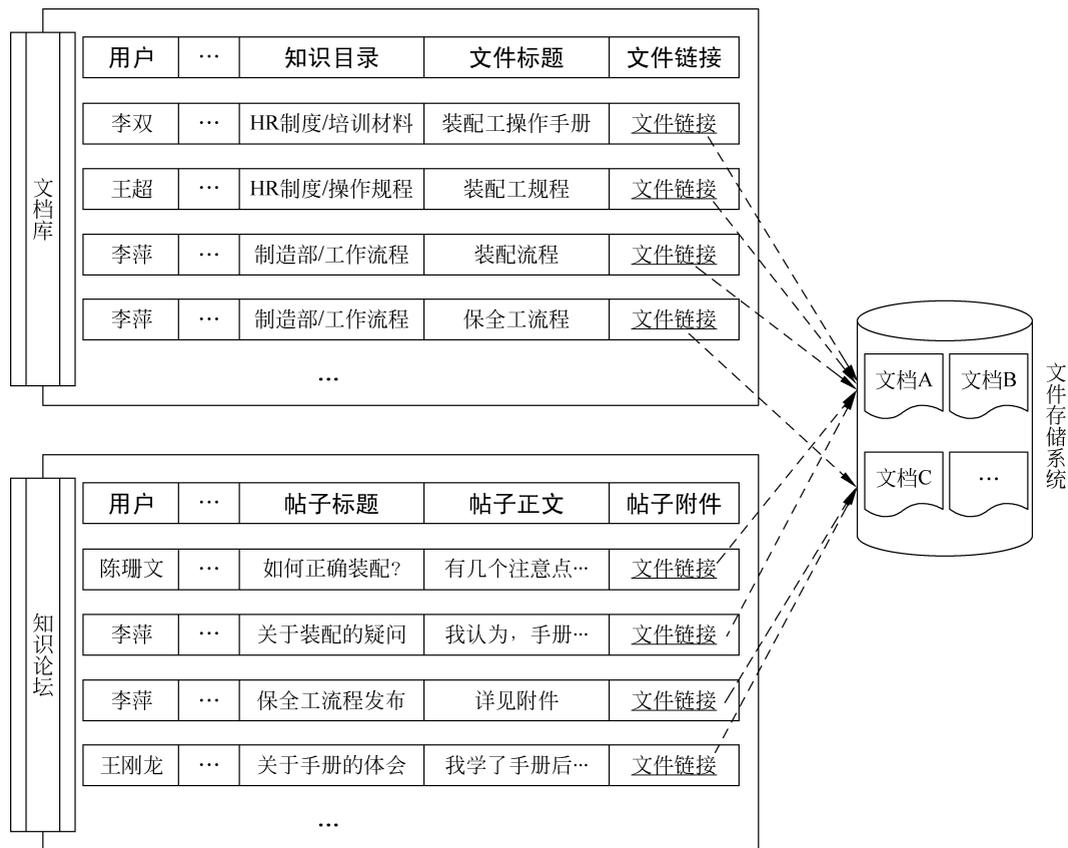


图 5-5 功能点文档库与知识论坛的关联

仔细分析一下上述案例中的这个删除文件的过程,无论是文档库中的删除文档功能,还是知识论坛中的删除附件功能,都需要对所有存在文件链接的表进行扫描,跟别的功能有很强的关联性,以后如果有新功能也需要链接文件,这个删除功能中的扫描文件逻辑可能需要不断修改。这样,文档库也好,知识论坛也好,都不可能独自移植到别的软件系统,也就是说不具有可重用性。为了解决这个问题,小王决定改变一下思路,当用户删除文档时,直接删除链接,不做任何其他处理,然后增加一个调度任务,每天定时执行,搜索没有被链接的文件,发现后从服务器删除掉,释放空间,由于删除文件释放空间的实时性要求并不高,这样做是可行的。通过这种处理方式,大大降低了这两个功能之间的关联性,未来被重用的可能性大大提高。

在很多情况下,可重用性跟灵活性是有相通之处的,越是灵活的功能,意味着跟别的功能的关联性越小,也即意味着功能越具有可重用性。

2. 注意数据的流动方向

当两个功能模块之间不得不建立关联时,需要考虑是否能引进一个简单的功能模块作为一个顶层模块,底层模块之间不进行直接的数据交流,而是通过顶层模块进行中转。要注意的是,当建立起顶层、底层这种结构时,发起数据流的应该是底层模块,顶层模块总是非常轻量的,有时候甚至不会提供任何功能,只是另外增加了某个数据字段而已。如果由顶层模

块发起,会大大降低软件的灵活性。



案例：从底层发起数据流

某库存管理系统,包括入库、出库、结账等功能模块。对于会计期间(会计术语,一般一个自然月为一个会计期间,如2014年3月份)有严格要求,如果某会计期间已经结账并生成了报表,由于这些数据已经上报了管理部门、政府机构,因此必须控制不允许针对该会计期间再做入库、出库操作(会计上,可能是为了防止舞弊,如果发现历史数据的错误,一般通过调账的方式处理,不允许直接在历史期间中直接修改、删除、增加数据,在手工记账时代,有个“严禁对会计账簿进行刮擦挖补”的要求)。

为了满足这个要求,设计了一个仓库会计期间管理的轻量级功能,为仓库会计期间设置两个标志字段,分别是“是否锁定标志”和“是否结账标志”。通过锁定标志用户可以自己人为锁定某个仓库的会计期间,被锁定的会计期间不允许进行入库、出库操作,一般用户会在结账之前执行这种锁定操作,以防在结账过程中发生数据冲突;当然也不排除用户误操作的可能,为了增加功能的健壮性,也提供“解锁”功能,允许用户去除锁定标志,但如果“是否结账标志”为“是”,则不允许解锁。这个“是否结账标志”来源于结账功能,当针对某个仓库会计期间结账时,在这里置“是否结账标志”“是否锁定标志”都为“是”。当入库、出库时,会判断“是否锁定标志”,如果为锁定状态,则不允许入库、出库,如图5-6所示。

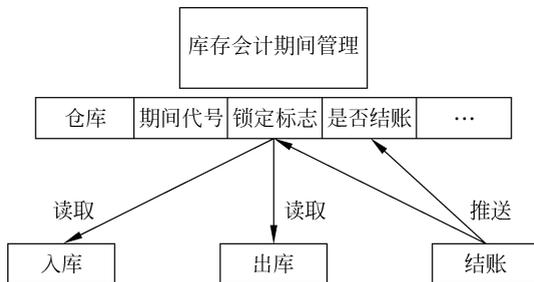


图 5-6 从底层发起数据流

分析上述案例,由于存在了这个顶层的轻量级模块,增加了软件的可重用性,因为随时可以将入库或出库功能跟仓库会计期间管理一起作为一个整体移植,而不需要同时移植结账那个特别复杂的功能。另外,也增加了功能的灵活性,因为入库、出库时只要判断这个锁定标志,就可以决定是否允许入库、出库,规则非常简单,发生变更的可能性小;如果需要从结账功能中判断,这个规则就复杂,将来不知道会有什么不可预知的变更。而且,将来锁定仓库会计期间的规则如果发生变化,也不需要对于入库、出库功能进行任何修改,例如,未来可能增加某个盘点功能,要求确认盘点结果后锁定仓库会计期间,那么只要在执行时将这个“是否锁定标志”置为“是”就可以了,而不需要将入库、出库这些牵涉仓库会计期间的功能都修改一遍。

3. 建立团队的通用规范与通用功能

一般每一款软件都有自己独特的功能、数据结构、数据流,有各自的设计者、开发者,它们的风格、习惯、能力也有或大或小的差别,这就会导致一个项目组开发的功能很难被其他

项目组重用。为了提高功能在项目组之间的通用性,建议有目的地建立一些团队规范,如界面设计风格、各种标题命名方式、文档编写格式等。另外,某些团队非常常用的、容易跟别的功能有关联性的功能,可以建立起团队的通用功能,至少要统一数据结构。



案例：建立团队通用功能

某软件公司为了提高各项目组开发的功能的可重用性,以平台的方式开发了用户管理、角色权限管理、员工管理、组织部门管理等功能,这些功能管理的都是系统或企业相当基础的信息,可以说是每个项目的必备功能,几乎每个功能点都与这些功能有关联性,如果没有这些通用功能,项目组开发的业务功能几乎不可能在其他项目组得到重用。

5.4.3 高效性

追求功能的高效性,指努力提高功能的运行效率,降低 IT 资源的消耗,让软件系统运行得更快、更经济——这不应该仅仅是研发人员需要考虑的事情,作为设计者,如果在设计阶段能有目的地在这方面做出努力,相信会给以后的工作带来更多的便利。

一般来说,设计者需要考虑的 IT 资源开销表现在这几个方面:数据存储能力、服务器运算能力、网络传输能力、客户端运算能力。不同的业务系统,不同的架构方式,甚至针对不同的用户,关于这几个方面的考虑重点并不相同。例如,偏向海量数据收集(如物联网相关的系统)、文件保存(如相册、文档库)的系统,需要在数据存储优化上下功夫;C/S 架构的系统由于被部署在局域网中,对于数据网络传输的考虑就不像 B/S 架构的系统那么严格。

这里推荐一些可以用来提高功能运行效率的小技巧,供读者在工作中参考。

1. 使用率不同的数据采用不同的保存方式

当设计的系统投入使用后,作为设计者,要问自己一个问题:如果这个系统连续使用 5 年,它还能像刚上线的时候一样顺利使用吗?正常情况下,如果系统是经过认真设计的,且数据膨胀速度没有过度偏离设计预期,那么软件的响应速度不应该有太大的变化。保证软件能够在当前业务规模下运行 5 年以上,不会出现明显的性能问题——这是在设计阶段必须考虑的。有时候,某些数据膨胀实在太快,如果不采用一些特殊的方法,恐怕别说 5 年,5 周都招架不住。



案例：数据迁移功能

某人力资源管理软件的考勤模块,提供了考勤分析功能。软件通过接口将员工的考勤打卡数据收集到本系统中,然后根据员工的排班信息分析员工是否有考勤异常,如迟到、早退、旷工等。对于一个小公司,考勤打卡数据一般不会太多,但对于一个集团性的大公司,每天的打卡数据也许会有几万条,如果对一个大型学校的上课进行考勤,那么恐怕还会更多。几年下来,这个数据量恐怕会达到亿级规模,如果不采用一定的方式处理,必然会跟这些数据相关的功能越拖越慢。为了解决这个问题,该软件提供了数据迁移功能,可以让用户在需要的时候将历史打卡数据(N 个月前的)迁移到别的表中。由于考勤分析是按月进行的,分析后生成的结果已经保存下来了,打卡的历史数据对后面的分析工作已经没有影响了,只是作为备查,如果要查比较久远的打卡数据,可以从历史表中检索。通过这种方式,可以让

打卡数据表的数据量维持在一个比较稳定的量级,这样就不会因为数据的大量积累拖死系统。

2. 利用中转数据

利用中转数据提高效率的方式,在很多大型报表中经常用到,读者可以参考需求获取中关于报表分析的部分。当然,这种方式并不仅仅适用于生成报表,在很多日常操作中,利用中转数据提高效率的处理方式也是非常常见的。



案例：利用中转数据提高效率

某库存管理系统,在出库时需要判断仓库中的结存数量是否足够,不允许出库数量超过结存数量。为了判断某物料的结存数量是否足够,有以下两种基本方法可以考虑。

方法一:出库时,将当前仓库中跟该物料相关的所有库存交易的出入库数量求和汇总,然后可以计算出当前的结存数量。

方法二:设立中转数据,记录物料在仓库中的结存数量,每次进行出入库操作时,如果入库,就增加该数量,如果出库就减少该数量。

分析这两种方法,方法一实现起来非常容易,但有一个致命的地方,就是每次出库时都要做一次针对大量数据的求和运算,这个需要较多的性能开销,随着历史数据的积累,这个操作会越来越慢;方法二实现起来稍微难些,每次入库、出库需要做额外工作,但有一个很大的好处,就是提高了出库操作的效率,库存交易数据的积累对它的执行效率影响很小。

3. 外键必填

对应数据库中一对多的关系,当保存“多”的一方的记录时,如有可能,不允许外键字段为空。例如,当录入员工时,部门不允许置空,当录入采购订单时,供应商不允许置空。这么做的好处是,可以减少使用“Left Join”查询信息的可能,从而可以提高查询效率。相信对于查询优化有些了解的读者都知道,使用“Left Join”的查询语句比使用“Inner Join”的查询语句难以优化,效率不容易得到保证。

要注意的是,虽然强制外键必填可以提高查询效率,但有时候这会给用户带来不便,当这个要求不合业务逻辑时,用户不可避免地会有反感情绪。这时候,可以考虑采用让系统自动生成默认值的方式处理。



案例：通过外键必填提高效率

小王在设计一款人力资源管理系统,为了提高查询效率,要求当用户录入员工时,同时要录入部门,也就是说部门不能为空,但用户认为,有些特殊员工在入职时并不知道他们会被分配到哪个部门,要求允许部门置空。小王试图说服用户,如果不清楚是哪个部门,可以先建立一个临时部门,等确定好部门之后再将这个员工从临时部门调整到新的部门,但用户坚持认为这么做会给他带来不方便。由于经常要根据部门查询员工,根据员工显示部门,如果部门不必填,会在软件中出现很多类似这种查询语句:

```
Select 员工.工号,员工.姓名,部门.名称
From 员工 Left Join 部门 on 员工.所属部门代号=部门.部门代号
Where...
```

显然,这样会降低软件功能的执行效率。其实不仅仅是这种查询语句,在很多时候,在许多使用到员工信息的功能中都不得不增加某种判断,对有部门的员工如何处理,对没有部门的员工如何处理——增加了判断规则,必然会降低软件的灵活性。

为了满足用户的需求,同时又不影响功能执行效率,小王决定引入默认部门,初始化系统时,在部门表中插入一个特殊的部门,用户在录入员工时,允许部门置空,但保存时,如果没有录入部门,系统会自动将该员工归入到这个特殊部门,这样就可以保证系统中每个员工都有所属部门,由于不存在没有所属部门的员工,查询员工时自然就不需要“Left Join”了:

```
Select 员工.工号,员工.姓名,部门.名称
From 员工 Inner Join 部门 on 员工.所属部门代号=部门.部门代号
Where...
```

4. 优先使用客户端资源

客户端设备一般包括 PC、手机、PAD 等,随着 IT 技术的进步,这些设备的能力越来越强大了。也不能不承认,大部分情况下,这些设备的利用率是很低的,性能的瓶颈往往在服务器端或者网络传输上,如果优先使用客户端资源,就可以降低服务器端的压力,从而提高系统的整体运行效率。很多运算,牵涉的数据量不大,但计算过程复杂,可以考虑将数据先从服务器端传输到客户端,然后在客户端计算、组装;也有很多运算,可以先在客户端做很多准备工作,直到确实需要时再提交到服务器端。



案例：优先使用客户端资源

一个最常见的例子,许多软件的用户登录功能都需要用户录入用户名、密码、验证码,如何验证用户录入的信息呢?

方案一:先验证用户名、密码,如果没有错误,再验证验证码。

方案二:先验证验证码,再验证用户名、密码是否正确。

一般情况下,验证用户名、密码需要访问服务器端的数据库,而验证验证码是不需要访问数据库的。比较两个方案,方案一只要用户提交一次,就会访问一次数据库,而方案二先验证验证码,如果验证码错误就不会访问数据库,可以节省服务器资源。



思考题

1. 假设需要给学校图书馆开发一款图书管理软件,根据你对图书馆管理图书业务的了解,进行功能划分(从功能模块到原子功能)。

【提示】 图书馆应该怎么管理图书,买书、上架、借书、还书、整理、收押金、退押金、收罚款等功能都是需要的,当然,还不止这些。

2. 撰写学生到图书馆借书的主场景。

【提示】 想想自己到图书馆时如何把书借出来的,注意强调人机交互,描述要精炼。

3. 某学校对学生请假的管理要求是:如果不超过 3 天,班主任批假;如果超过 3 天不超过 7 天,班主任批假后学工处批假;如果超过 7 天,班主任批假后学工处批假,最后李校

长批假。根据这个要求画出流程图。

【提示】 流程图的要求是可以将需求表达得足够清晰,可以让程序员据以开发,因此仅仅画出分支是不够的,还要注明判断规则(如怎么判断班主任,怎么判断李校长)。

4. 上一题中,如何处理“李校长批假”这个问题?设计两种方案,一种写活,一种写死。分析一下这两种方案的优缺点。

【提示】 写死的方案工作量小,但不利于后续工作;写活的方案工作量大,但可以带来长远收益。

5. 某社交软件,当好友更换头像后,在当前用户的通讯录中展示的还是以前的头像(只有跟对方聊天后才会将头像更换成最新的)。说说软件设计者为什么要这么做。

【提示】 从性能方面考虑。



案例分析

1. 下面是某拓展培训公司的信息化需求,请根据本需求列出信息化系统需要的所有功能点,并描述每个功能点包括哪些主要功能。

(1) 业务员接单后,判断这是不是新客户,如果是新客户,需要在系统中为该客户建立客户档案。

(2) 业务员在系统中录入业务接洽单,然后通过系统发短信给相关人员,如综合办公室相关人员、经理。

(3) 综合办公室接到通知后,在系统中查看业务接洽单,根据业务接洽单提前做好客户住宿、用餐准备。

(4) 经理在系统中查看已经安排的课程和教练的情况,并根据业务接洽单安排新的课程和教练。系统会保证课程的时间安排不会冲突,并且会给出教练的安排建议,尽量保证教练的课程负担平衡。

(5) 经理安排好课程后会在系统中发短信通知相关教练。

(6) 教练接通知后到系统中查看自己的课程安排情况,然后根据课程开展拓展培训。

(7) 培训完成后,如果客户有反馈意见,客服会把客户的反馈意见录入系统。

(8) 会计收款后,根据业务接洽单把收款信息录入系统。

(9) 月底,会计在系统中生成工资单。工资按“底薪+奖金”的方式生成;教练员按培训课程获得奖金(津贴);业务员按接洽单获取提成,提成会在业务员和综合办公室相关人员之间分配。

(10) 会计可以在系统中查询各客户的付款和欠款情况。

2. 下面这段文字摘自某人力资源管理软件的宣传册,结合软件功能的优化方式,谈谈他们想表达什么,说说你的理由。

本系统搭建在本公司自建的软件开发平台上,基础功能全平台通用,大大降低了研发成本,让您体会到什么叫真正的价廉物美。

每个模块采用组件的方式开发,可以任意组合,搭积木式的系统部署方法让您随心所欲。

所有模块统一接口方式、数据交换方式、消息传递方式,让您的二次开发驾轻就熟。

本系统开发采用平台级的统一规范,可以让新开发的功能无缝嵌入。

本系统支持评分法、排序法、选优法、关键事件法、指标法等各种绩效考评方法,用户可以根据管理要求灵活设置员工考评方式。

本系统经过海量数据的压力测试,10 000 人以内的客户,使用 5 年后也能达到以下指标:

个人考评结果加载速度 <100 毫秒;

图文发布速度 <200 毫秒;

移动端页面刷新速度 <1 秒;

后台大型查询速度 <2 秒;

流程引擎节点推送速度 <500 毫秒。