开发框架-审批工作流
功能介绍说明书

V3.0

1. 引言

1.1. 背景

一讲到工作流，很多人第一反应就是这个东西很深奥，有时候又觉得离我们较为遥远，确实完善的工作流涉及到很多方面，而正是由于需要兼顾很多方面，一般通用的工作流都难做到尽善尽美。微软也提供了几个版本的工作流框架支持，也有一些厂家是基于这个框架基础上开发的工作流应用。

以前由于项目的需要，参与过一些工作流的项目开发，其中有些是基于我简易工作流的原理上进行拓展的，包括一个广州市各区县使用的行业审批业务平台，由于基于自己的流程处理，界面设计、流程流转等方面可以很好符合客户需求，定制的弹性较好，缺点是不够通用，也需要编写表单部分代码。

后面由于业务的需要，工作流方面的业务逐渐显得迫切，公司是想采用一个较为通用工作流框架来组织目前的业务，因此找了广州一家做工作流的公司，购买了他们的产品，虽然号称完全通过后台配置，零代码实现工作流业务表单的处理，但是由于客户对表单的设计要求比较多，有时候需要结合一些外部的数据接口，流程处理方面也有着进一步的需要，这样可能就打破了他们原来的格局，导致无论在表单设计、流程配置等方面，都需要购买他们工程师的现场服务，来进一步完善整个项目的内容，导致整个项目进展缓慢，遭遇水土不服的处境。

因此感觉，一个工作流模块，号称再强大，如果不能很好结合项目应用，即使零代码的功能配置，也可能使你处于尴尬的境况之中，因为通过配置，可能在代码里面平常很容易实现的表单功能，要通过零代码配置，花费的时间更多更难掌握，因为零代码是有代价的，需要您很好利用他们的 API，他们的业务对象，有时候还需要很曲折的摸索参数，而这一切可能就是非常致命的弱点。

1.2. 编写目的

本文档介绍基于框架的基础上开发的工作流审批模块，主要介绍《工作流模块》中的一些设计理念和功能特性，使得开发过程我们更加了解整个工作流的运行机制和相关功能。

1.3. 参考资料

序号	名称	版本/日期	来源
1	《Winform 开发框架-架构设计说明书.doc》		内部
2	《Winform 开发框架-数据库设计说明书.doc》		内部
3	《WCF 开发框架-系统部署使用说明书.doc》		内部
4	《开发框架-系统数据库还原操作说明.doc》		内部
5	《混合式开发框架主体框架项目说明.doc》		内部
6	《权限管理系统功能介绍白皮书.doc》		内部

1.4. 术语及缩写

1 在本文件中出现的“开发框架”一词，除非特别说明，适用于《Winform开发框架》、《WCF开发框架》和《混合式开发框架》。

2 在本文安装.NET框架中，除非特别说明，均指.NET 4.0框架。

3 如无特别说明，工作流模块是指本框架的工作流模块内容。

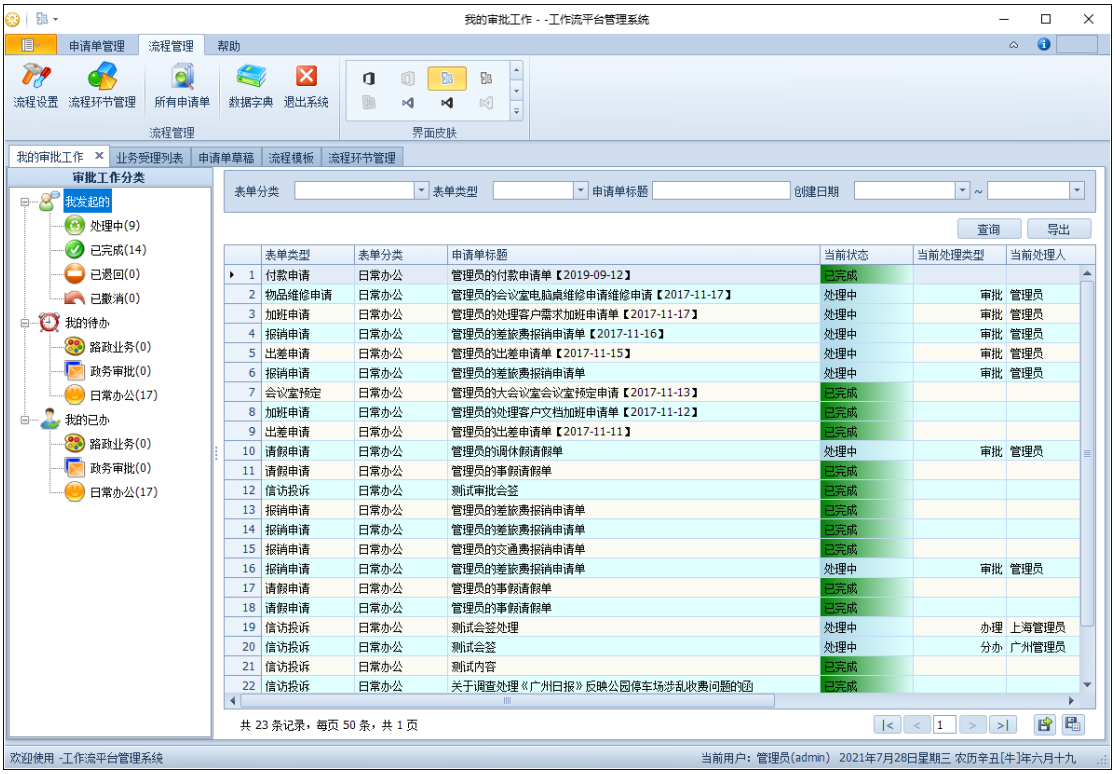
2. 工作流功能介绍

2. 1. 工作流登录界面

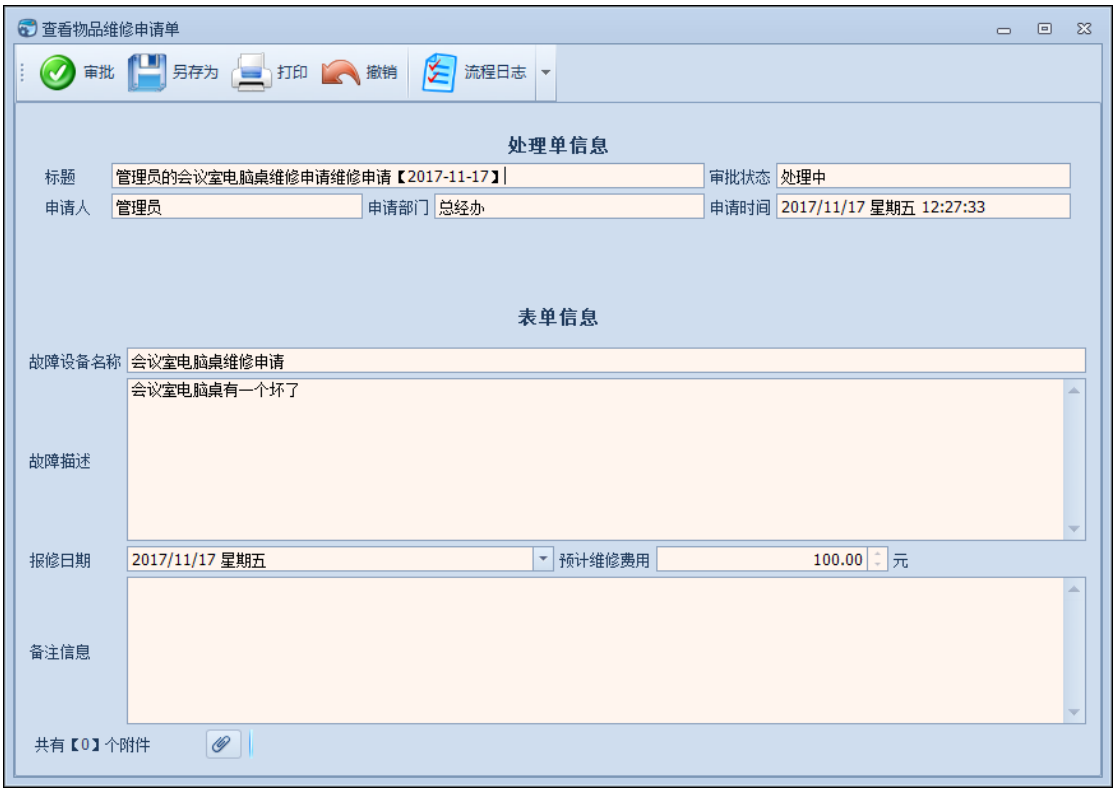
工作流是单独登录的模块应用，使用权限管理系统的身份认证，登录界面如下所示：

2.2.我的审批工作

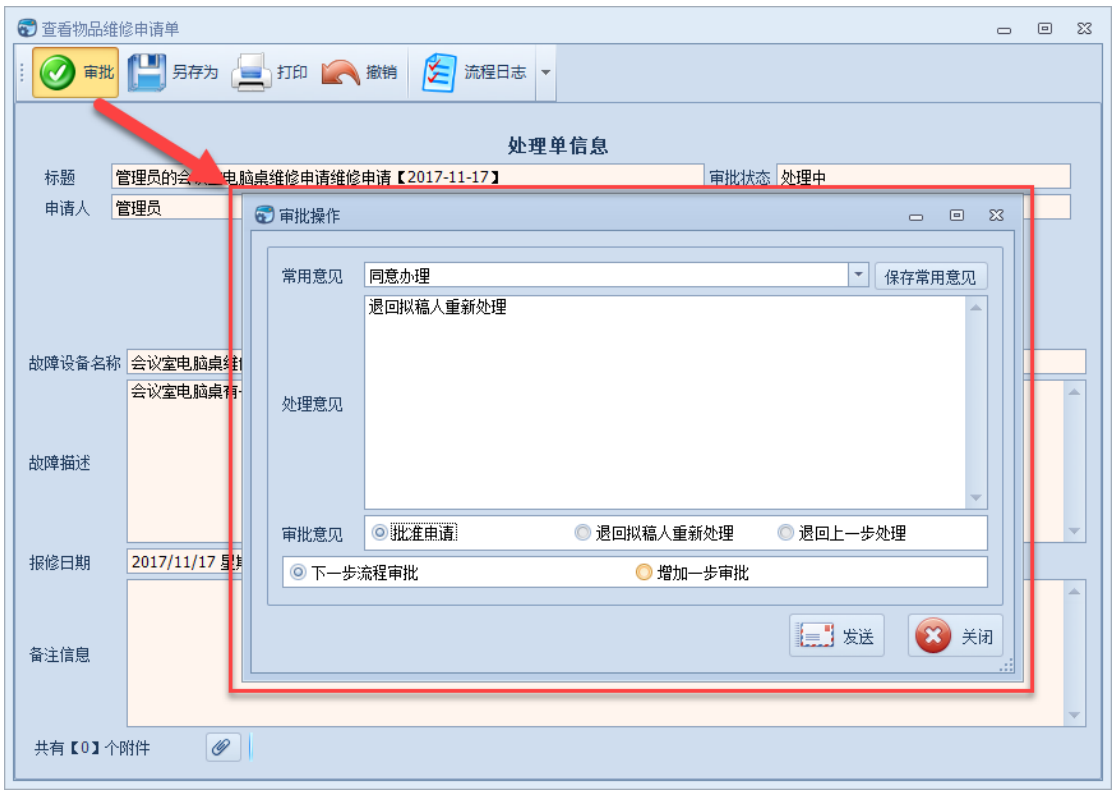
我的审批工作，包括【我发起的】、【我的待办】、【我的已办】几类大项，并可以根据查询条件进行过滤列表，界面如下所示：



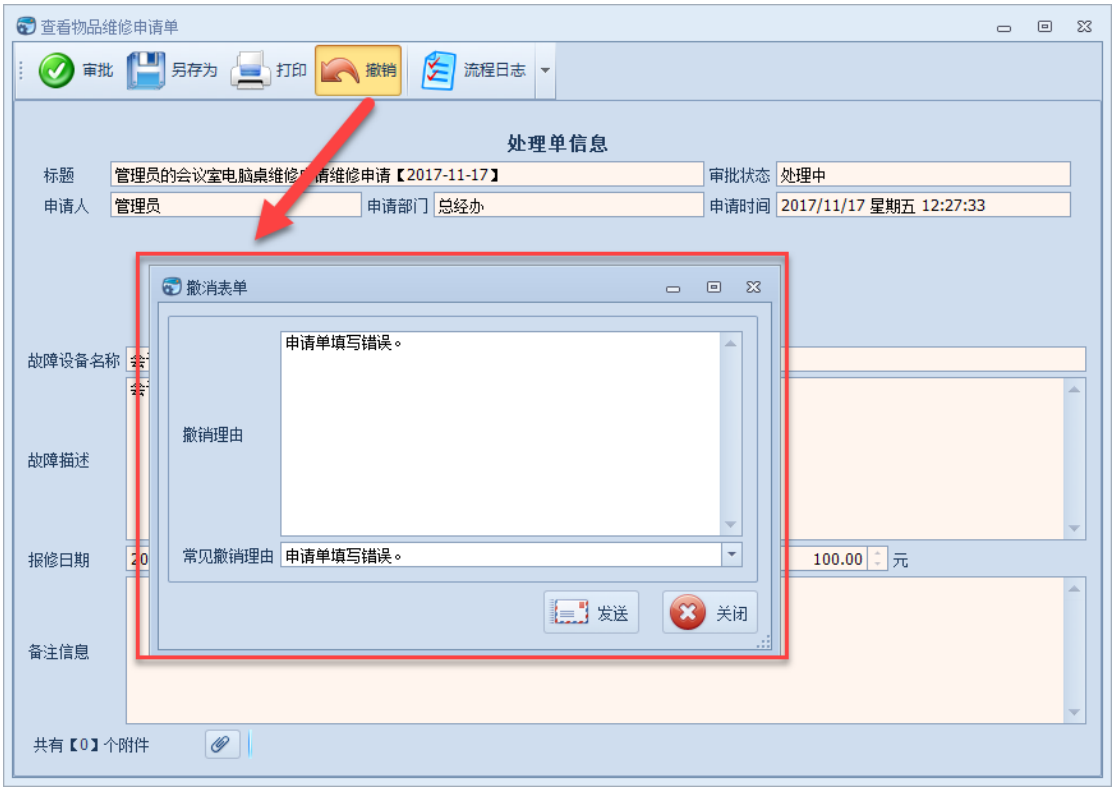
双击申请单，可以打开查看详细的申请单信息，如下界面所示。



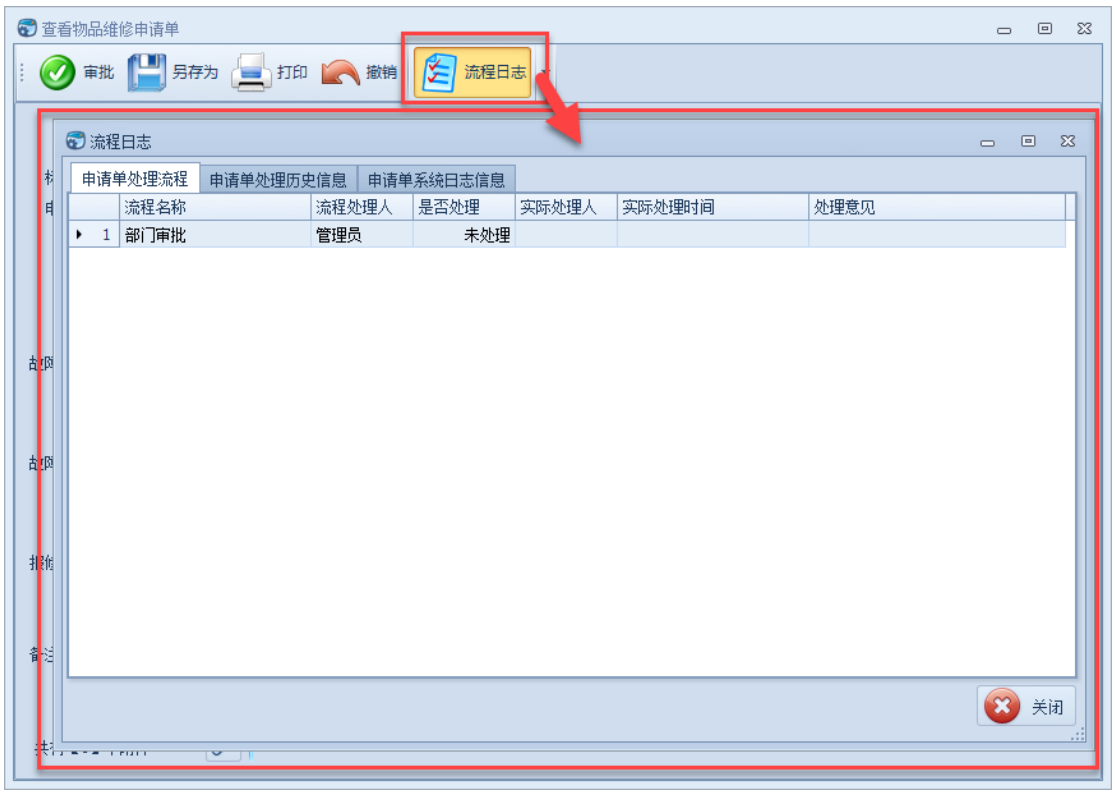
如果审批单可以进行审批处理，则会出现相关的审批按钮，单击【审批】按钮，弹出审批对话框，可以录入相关的审批意见，如下界面所示。



如果表单需要撤销，单击【撤销】按钮，会弹出申请单撤销的确认界面，在其中录入撤销意见确认即可。

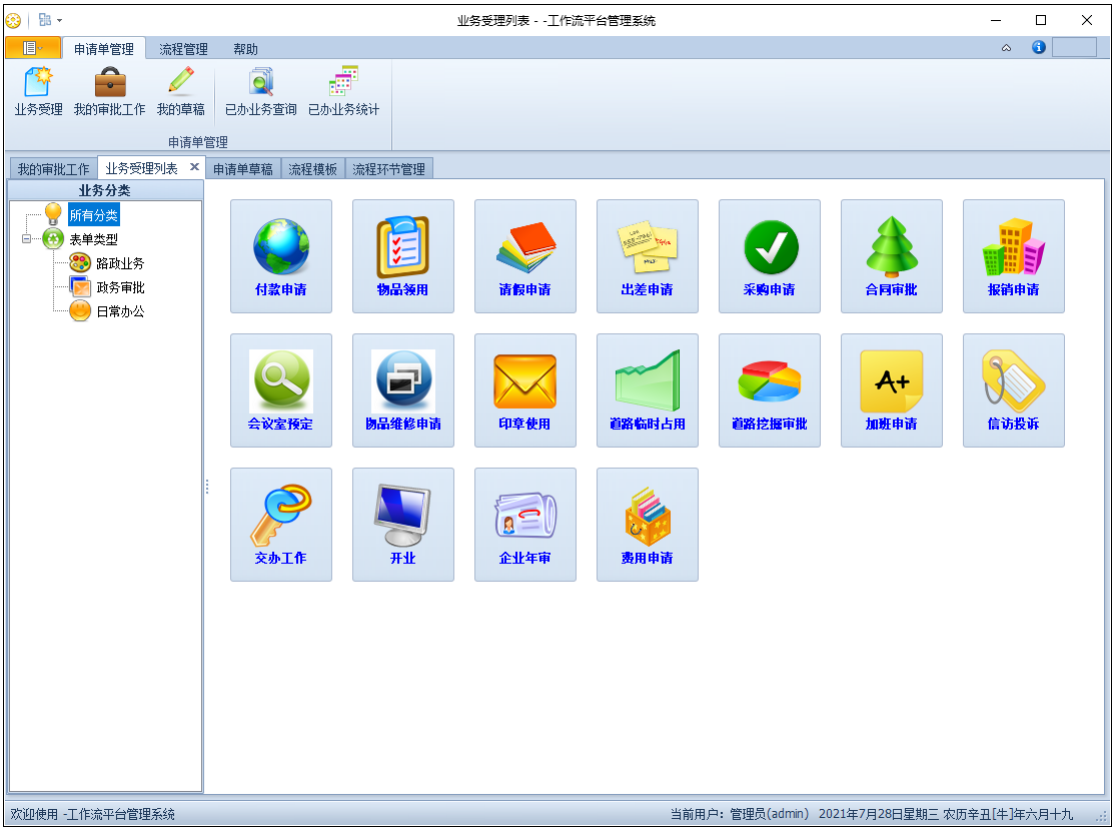


如果需了解申请单的流程日志信息，单击【流程日志】按钮，弹出申请单的相关日志信息，如下界面所示。

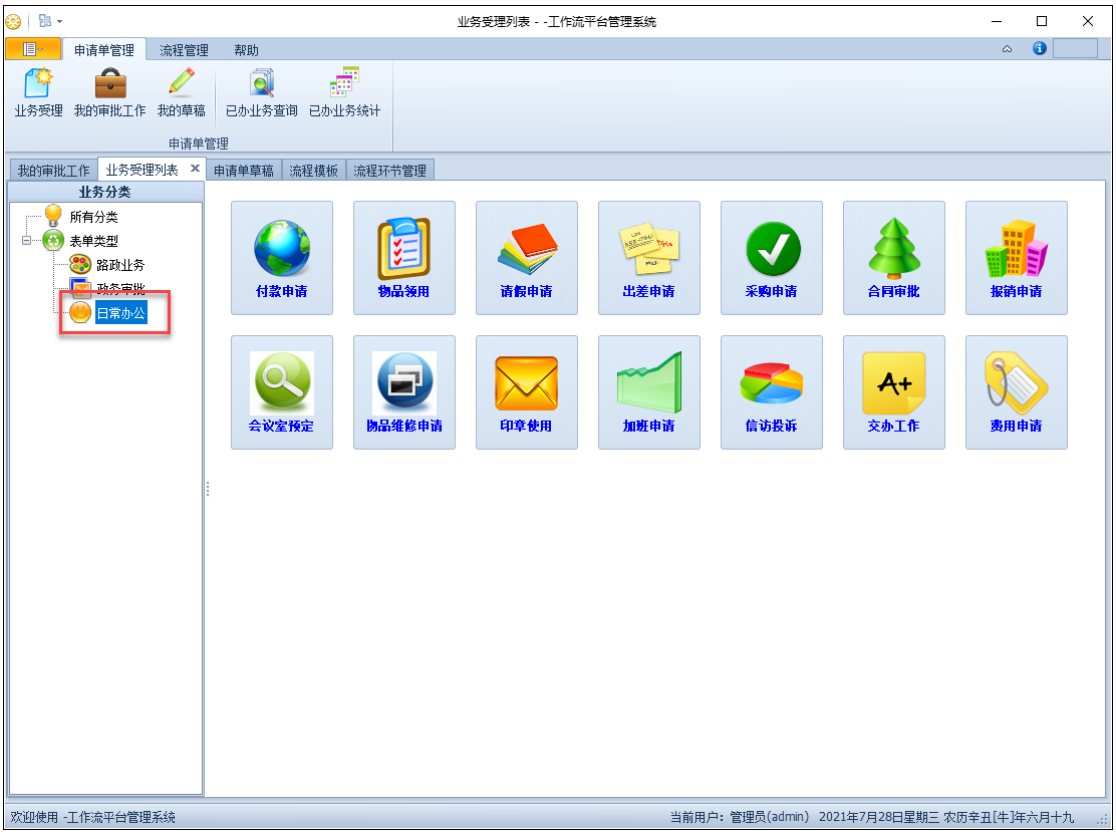


2.3. 业务受理列表

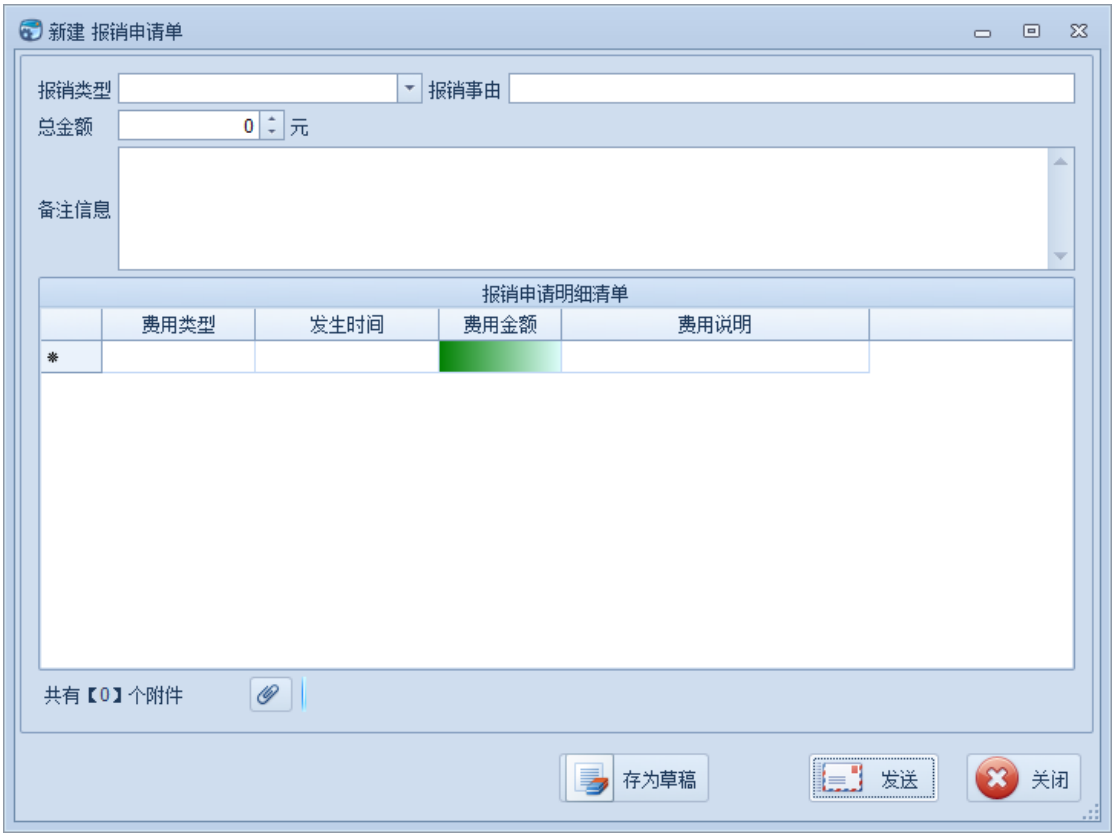
业务受理列表展示当前工作流模块中可以创建的申请表单入口，如下界面所示。



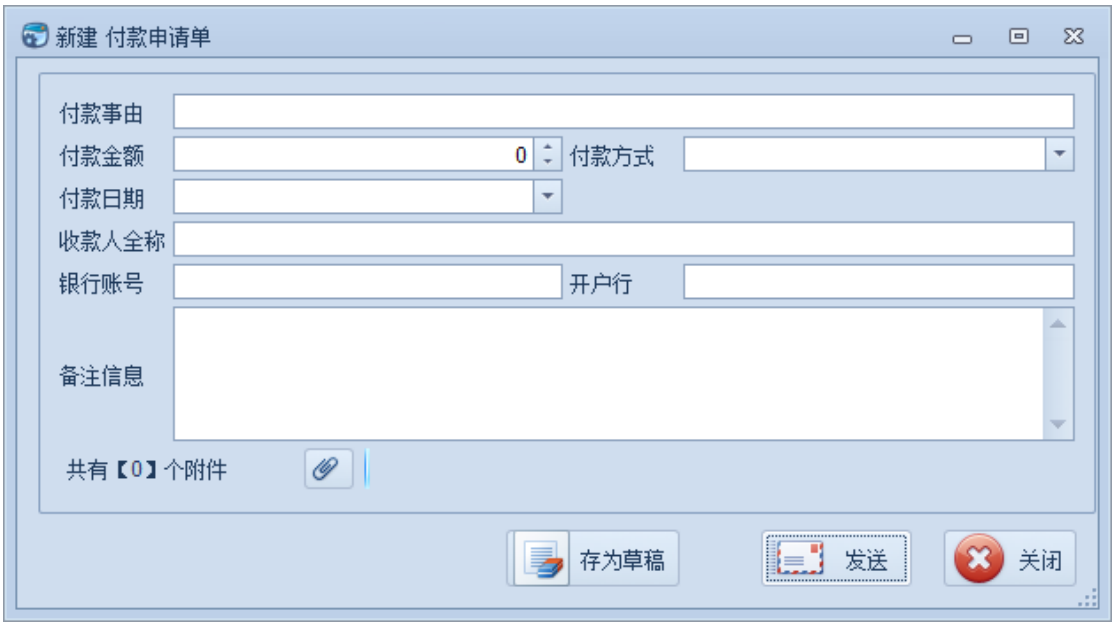
我们可以根据左侧树列表的业务分类进行过滤，展示不同分类的申请单。



单击其中图形按钮，可以创建一个新的申请单，申请单界面如下所示。



或者

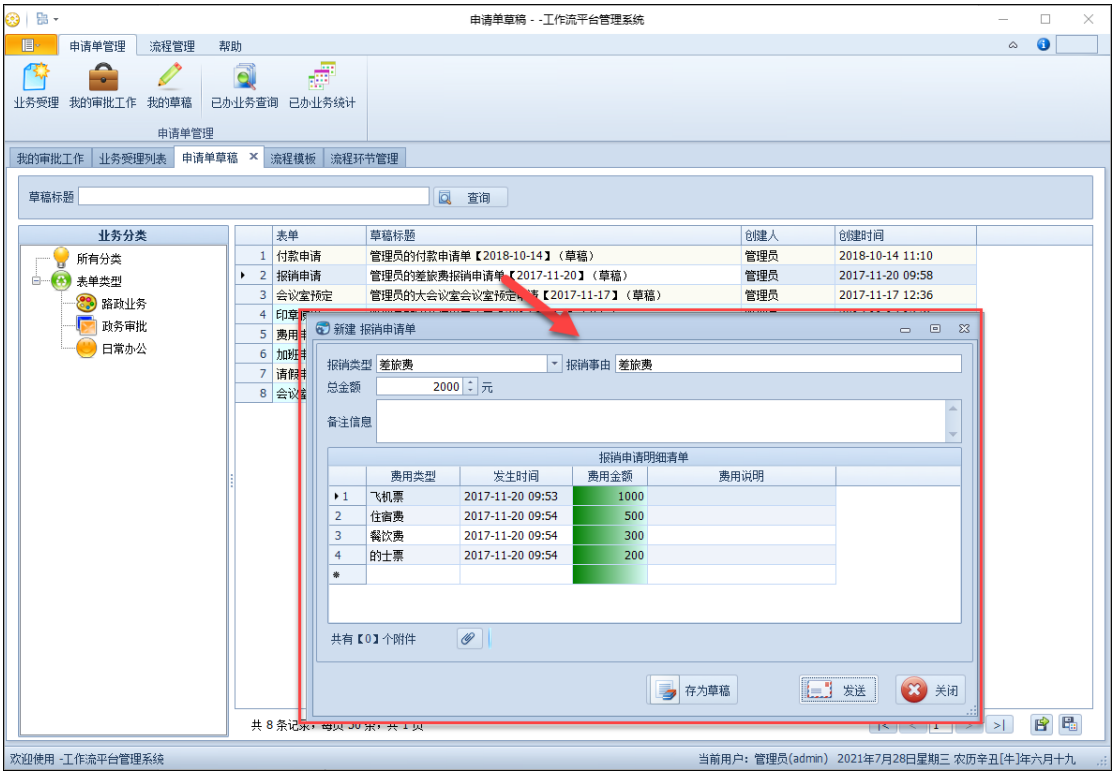


不同的表单界面，需要在开发模式进行开发处理。

2.4. 我的草稿

我的草稿是保存用户尚未完成的申请单，在用户编辑申请单的时候，可以暂存在草稿里面，方便下次继续完成填写工作。我的草稿列表界面如下所示。

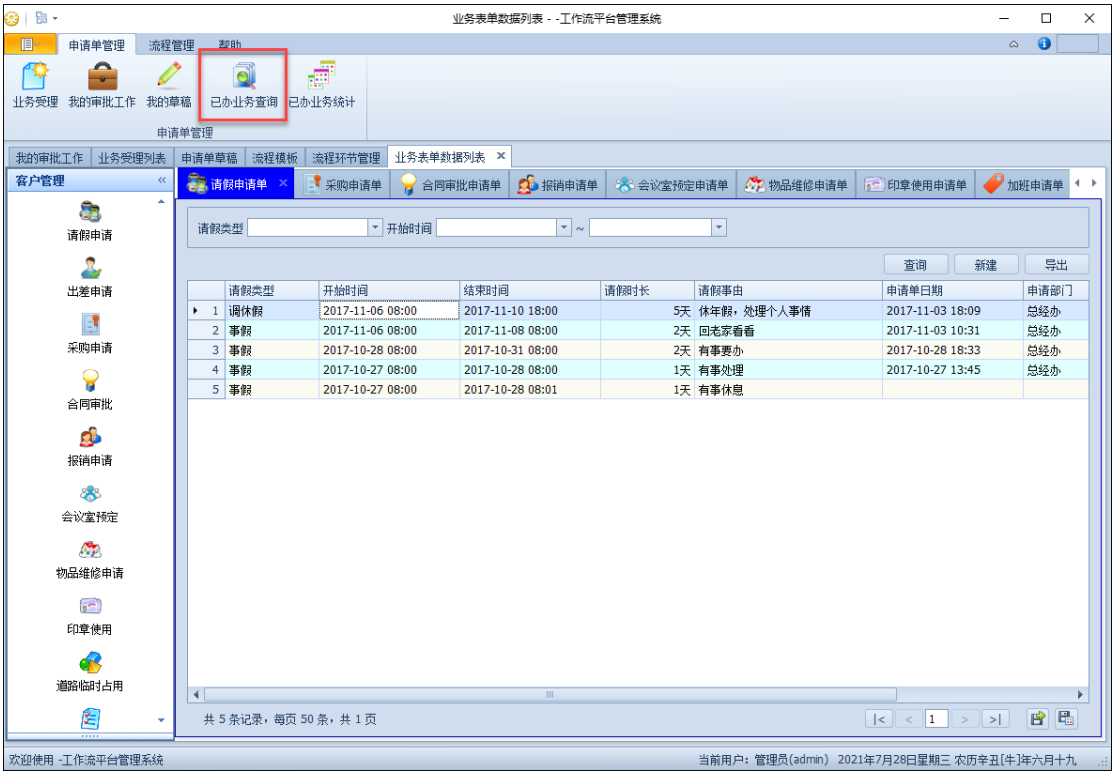
双击草稿列表的申请单，可以打开申请单的填写界面，如下所示。



选择【存为草稿】，则继续保存数据为草稿状态，如果选择【发送】则提交申请单进入流程中处理。

2.5. 已办业务查询

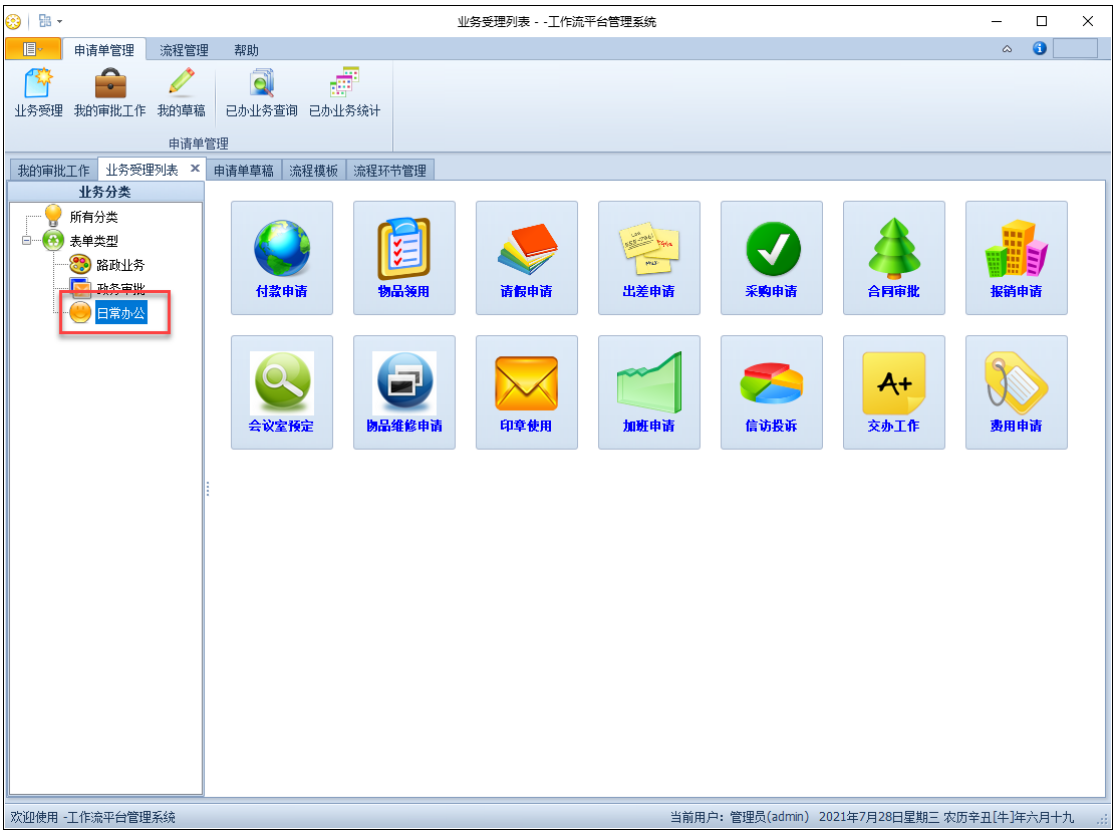
已办业务查询，按照不同的业务分类，展示相关申请单的列表，列表界面如下所示。



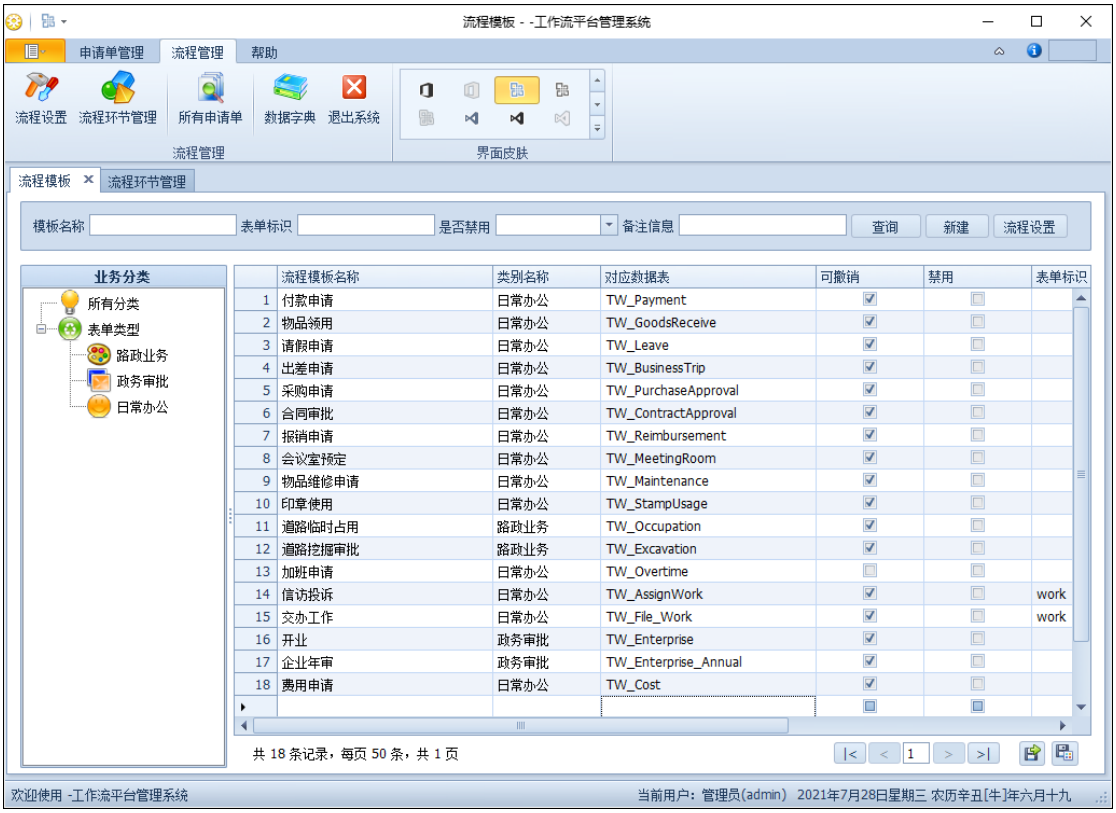
双击列表记录，可以查看具体的申请单详细信息。

2.6. 流程模板

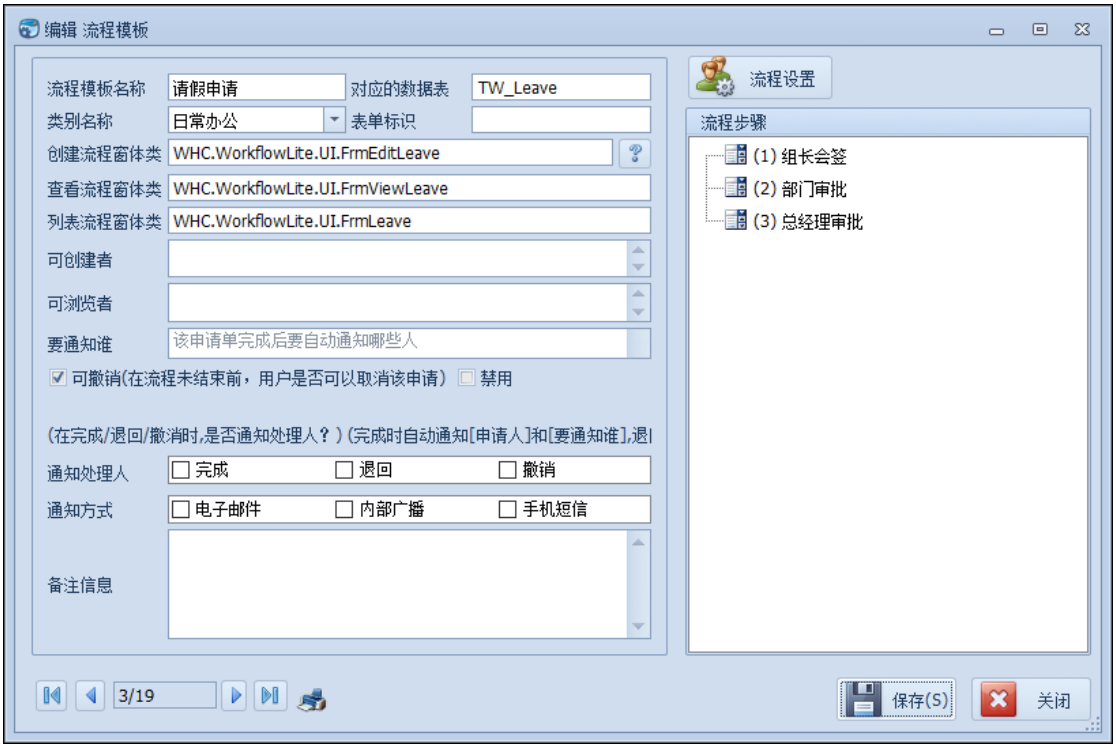
前面介绍过的工作流的业务受理列表，展示的是可以创建申请单的图形按钮列表，如下界面所示的效果列表。



其中的数据就是在这个流程模板中配置管理的，流程模板中管理工作流业务中可以使用的申请单配置信息，流程模板界面如下所示。



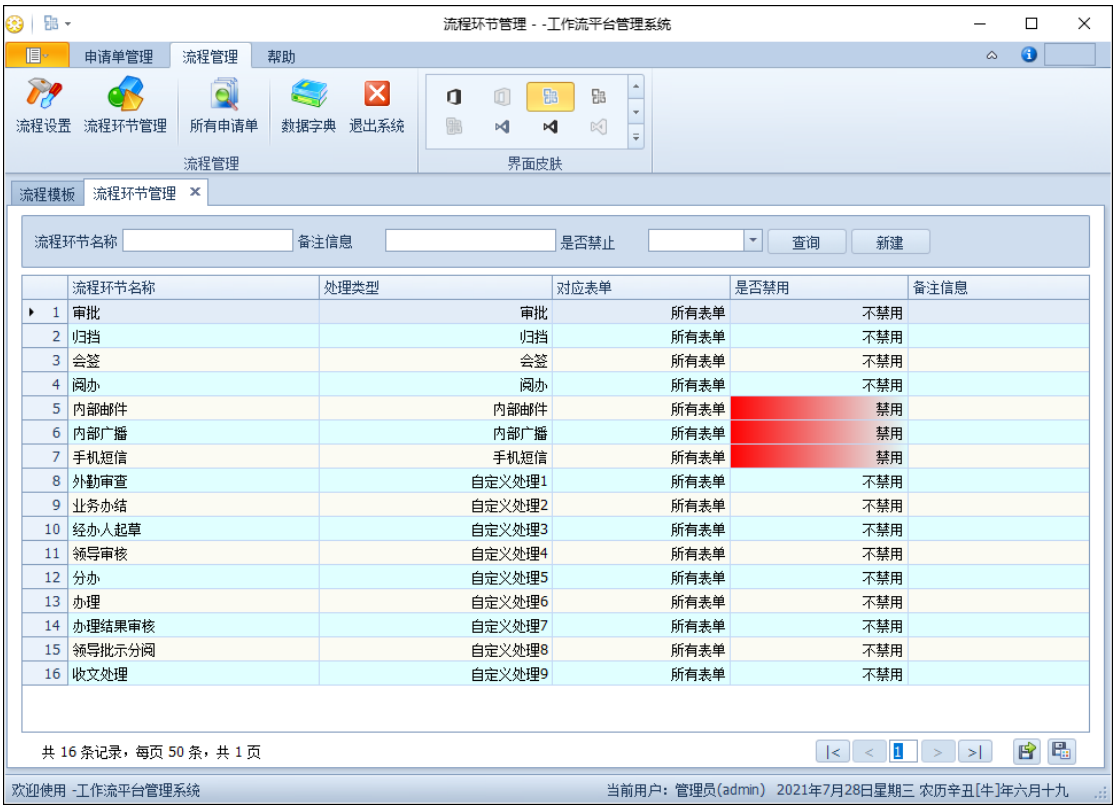
其中详细的配置窗体界面如下所示，这里配置框架处理中所需的相关参数数据，以及配置的流程节点。



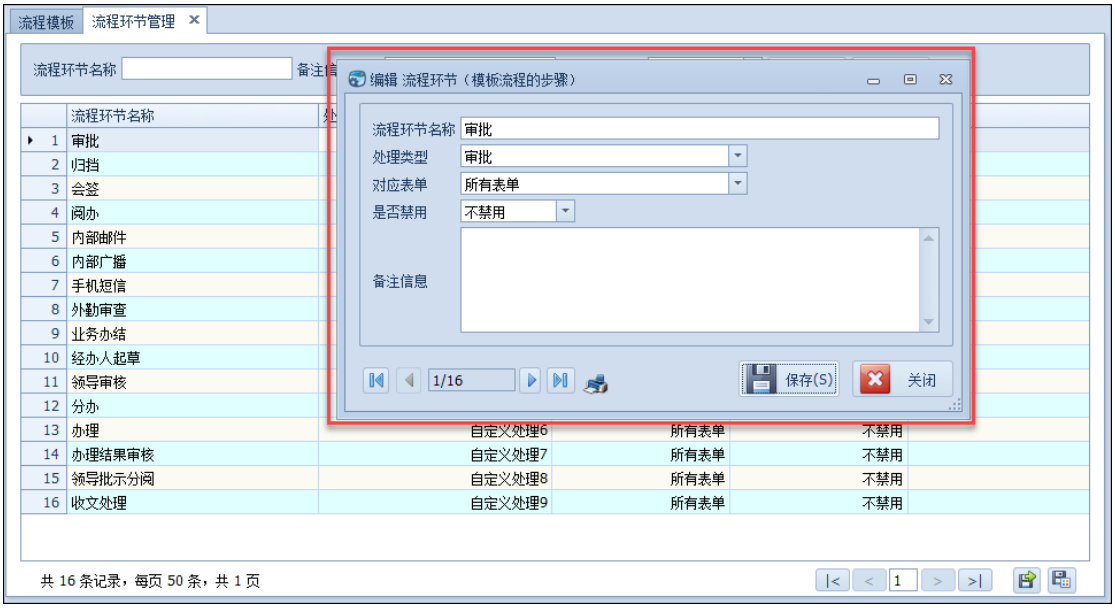
单击【流程设置】，可以设置该申请单模板的流程步骤，如下界面所示。

2.7. 流程环节管理

流程环节管理，是定义系统工作流中用到的流程环节类型（处理类型）信息，如下界面所示。



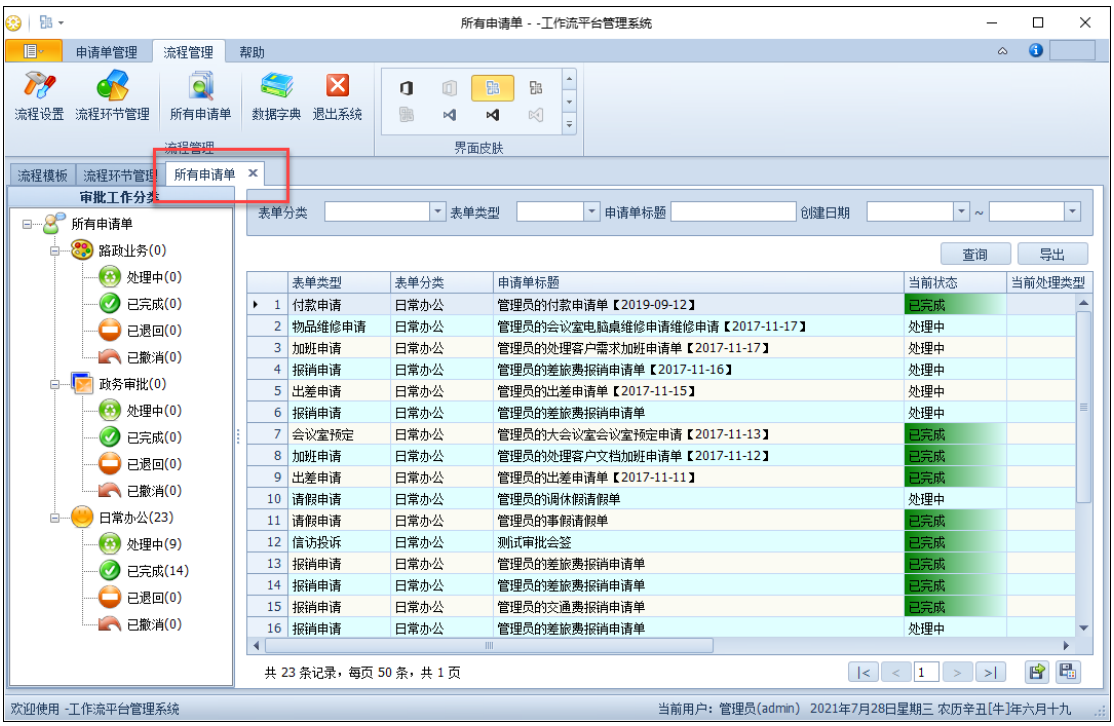
其中有常规的审批、归档、会签、阅办等操作，工作流需要根据这些流程类型来显示不同的处理界面。双击记录，可以进行编辑修改处理，如下所示。



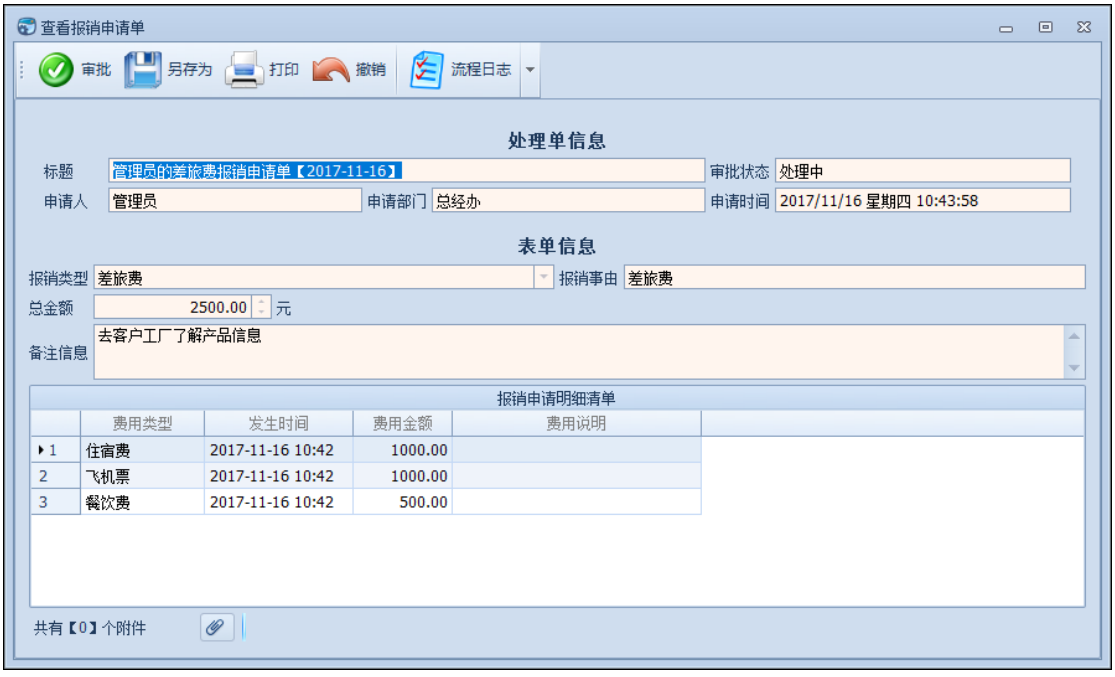
2.8. 所有申请单

所有申请单列出工作流系统中的所有流程表单的信息，便于管理员针对性的查找申请单

信息进行调整处理。



双击申请单列表，可以打开具体的申请单进行查看详细信息，如下所示。

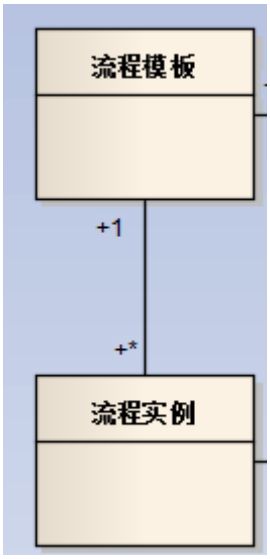


3. 工作流设计和开发

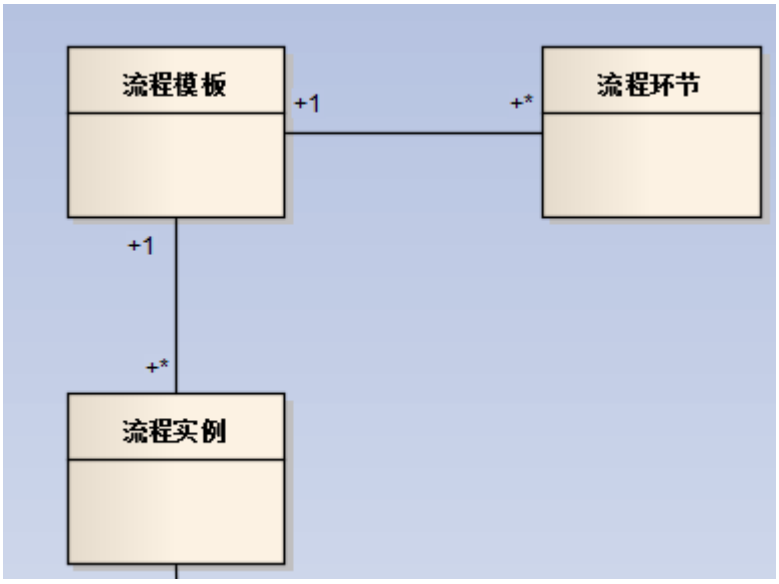
3.1. 简易工作流的设计模型

在没有第三方工作流模块的情况下，简易工作流就是利用数据库和业务对象之间的协作关系，构建的一个半模块化的流程引擎，它能通过整合到项目代码中进行更好的融合以便实现工作流的相关功能。

首先我们知道，我们在 Office 里面创建任何文档，都有一个模板的概念，这样我们方便利用一些现成的数据和布局，工作流也一样，有一个流程模板的概念，如下所示。

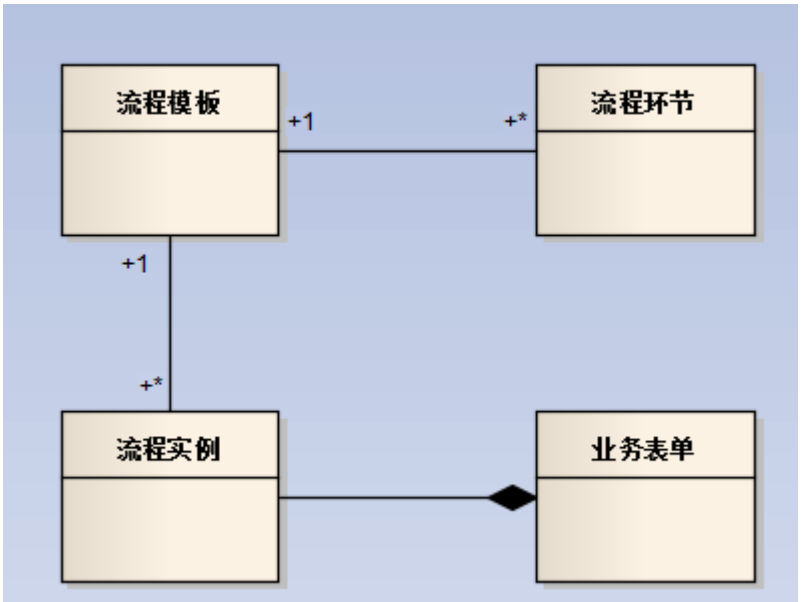


然后每个流程模板，本身会预定义了一系列的处理流程，以便在流程实例里面进行不同的处理，因此流程模板还包含了多个流程步骤对象，他们的关系构成如下。



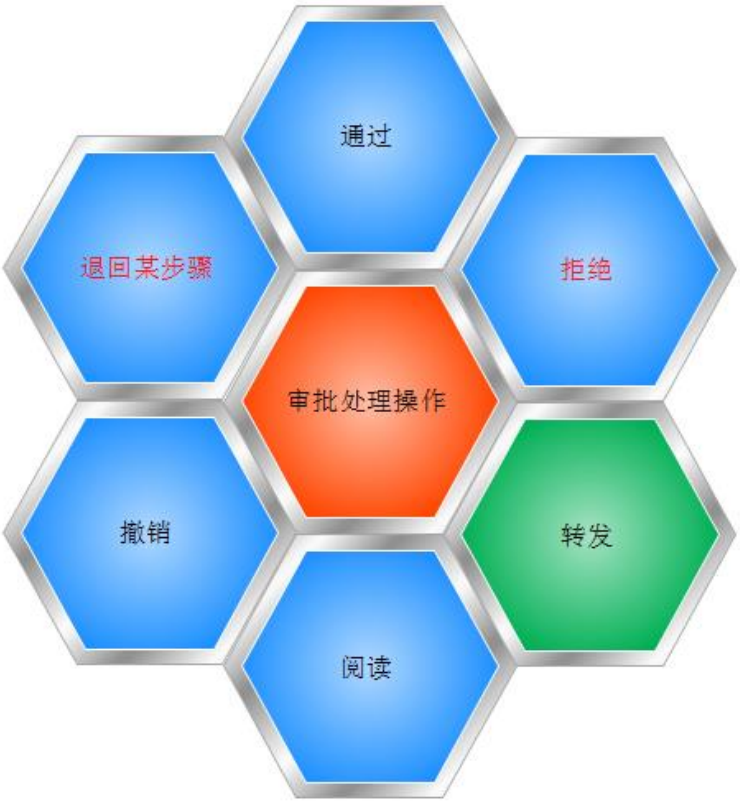
每个流程实例，除了他们自己的流程数据和字段信息外，它本身还有一个表单设计的问题，如费用审批，可能包含填写的费用清单数据等，所以流程实例还应该包含了流程的业务表单对象。

这样他们构成了一个完整的流程业务对象关系，如下所示。



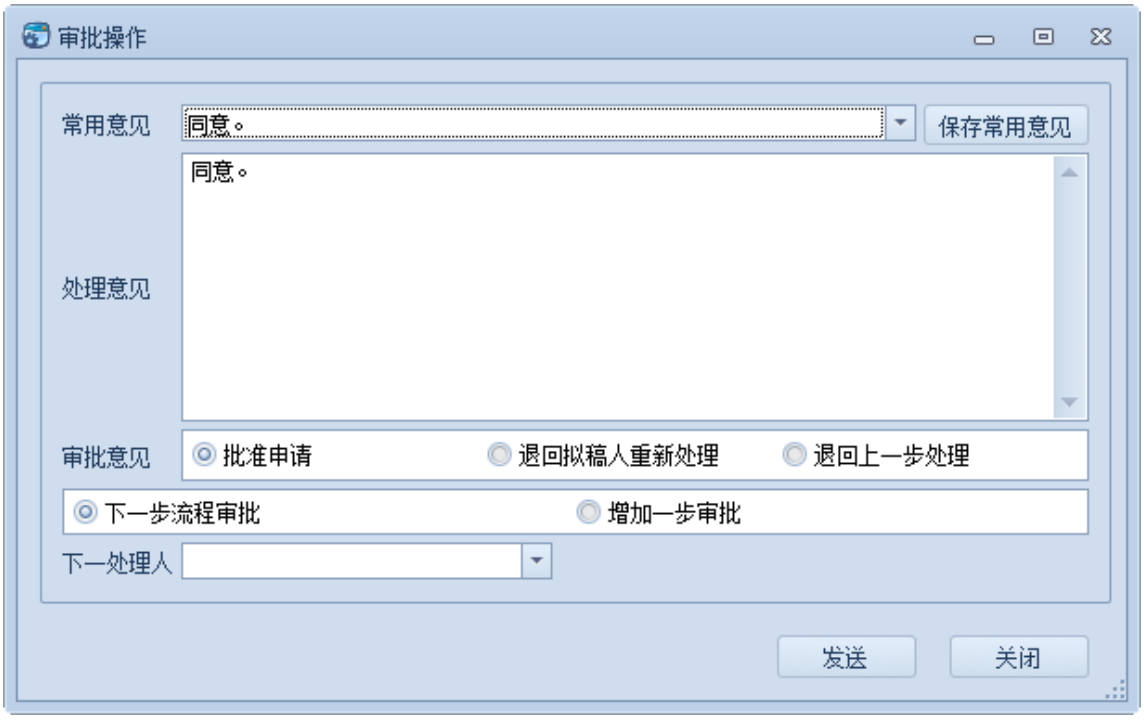
3.2. 流程审批操作

对于一个流程处理操作，我们知道一般有审批通过、拒绝、退回到某步骤、转发到内部阅读、阅读，以及包括起草者能撤销表单呢等操作，当然如果还有一些具体的业务，可能还会有一些流程的处理才操作，不过基本上也可以归结为上面几种，只是他们每步处理的数据内容不同而已。因此审批的操作步骤分类如下所示。



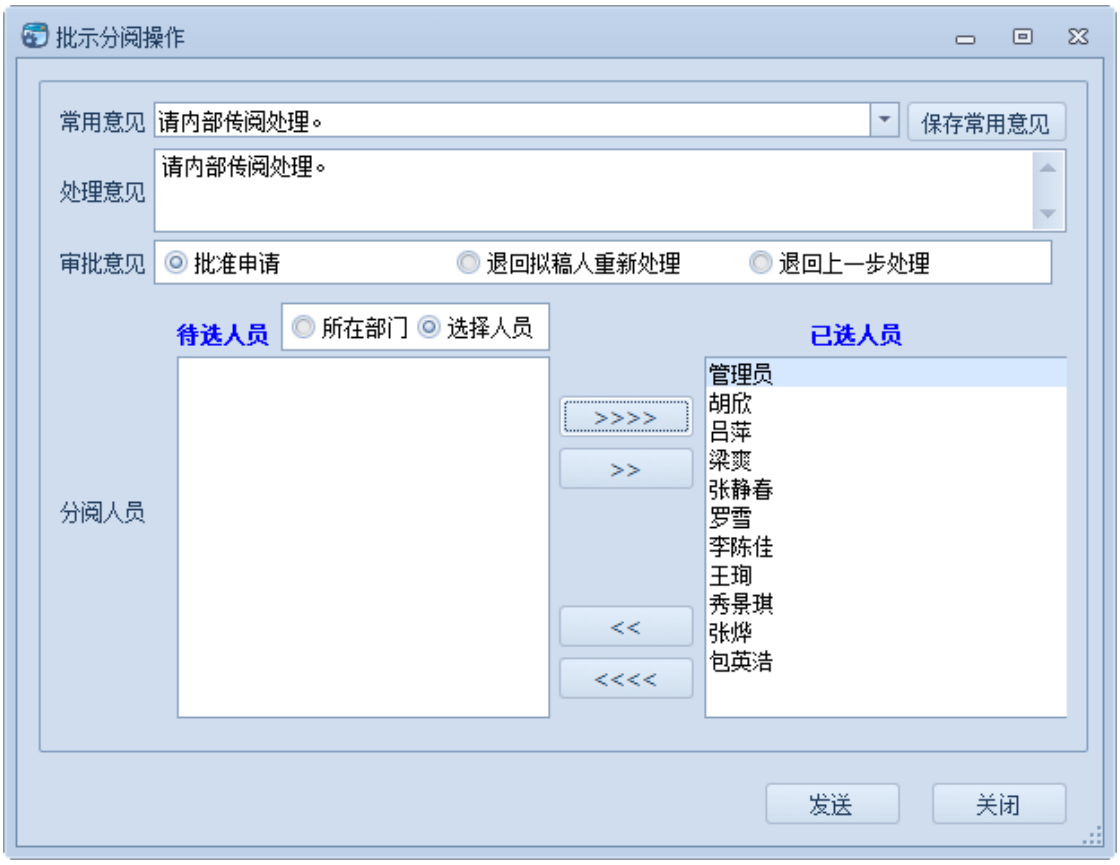
这些操作我们都可以通过一些界面操作的封装实现，因为他们基本上都是通用的，我们传入一些流程 ID 等相关标识后，就能交给这些标准的操作界面完成了。

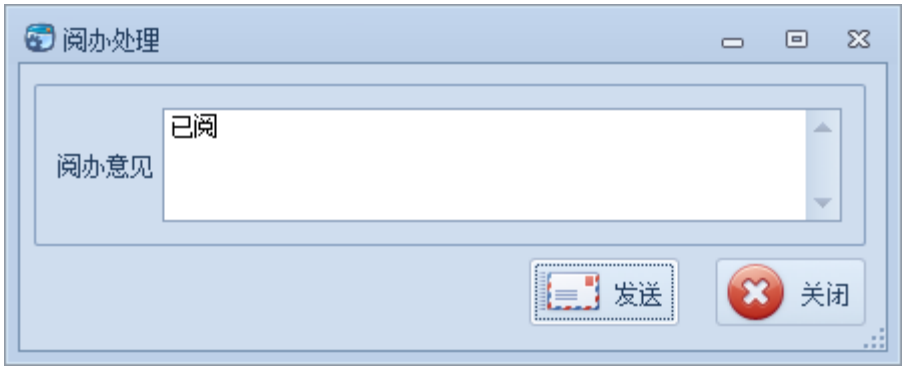
如审批界面如下所示，里面包含了通过、拒绝，跳回到某步骤，增加步骤等功能集合。



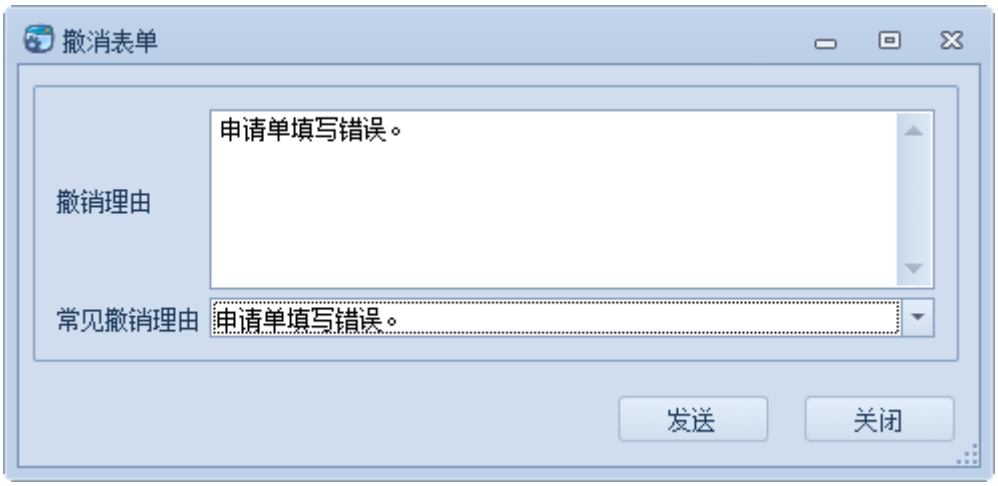
上面的界面是审批过程中，对于某一个流程处理人员实现的操作，而有时候，我们可能需要针对多个人进行某个步骤的处理，如会签处理，会签处理处理界面效果如下所示。

以及传递给内部人员进行分阅操作，那么就应该选定多个人员进行处理，大概的处理界面效果如下所示。





当然，若申请人的申请单填写错误，需要撤销的话，那么也应该有这个操作，撤销表单后，就可以重新填写表单，然后再次提交进行流程。

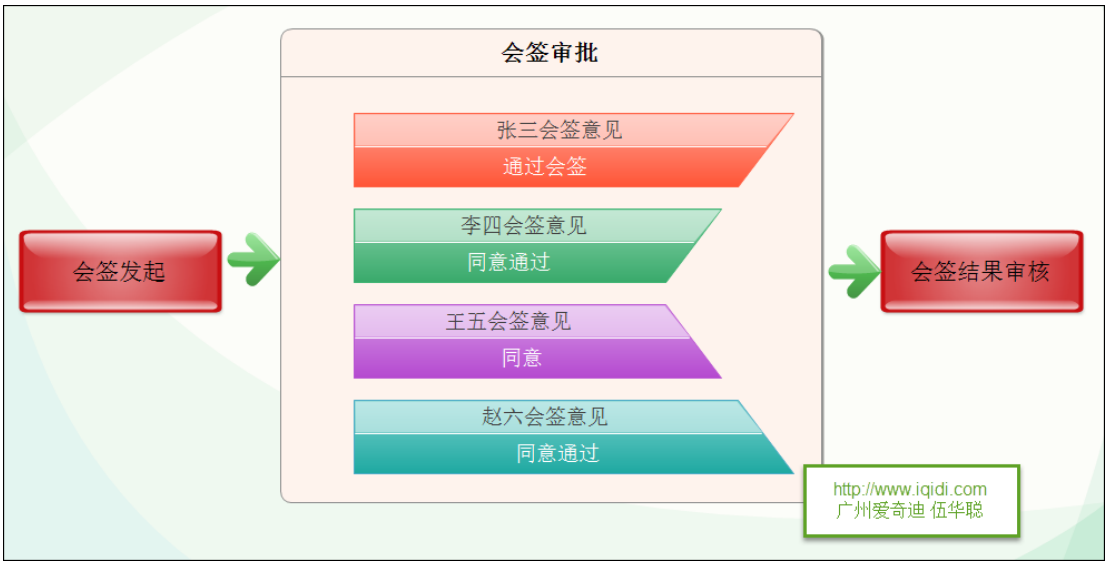


3.3. 流程会签操作

3.3.1. 会签流程定义

会签是指创建一个或多个子流程供相关人员进行审批，等待全部人员完成处理后再次回到主流程上，然后决定是否继续流转到下一个流程步骤上去，一般的申请单的主流程如下所示。

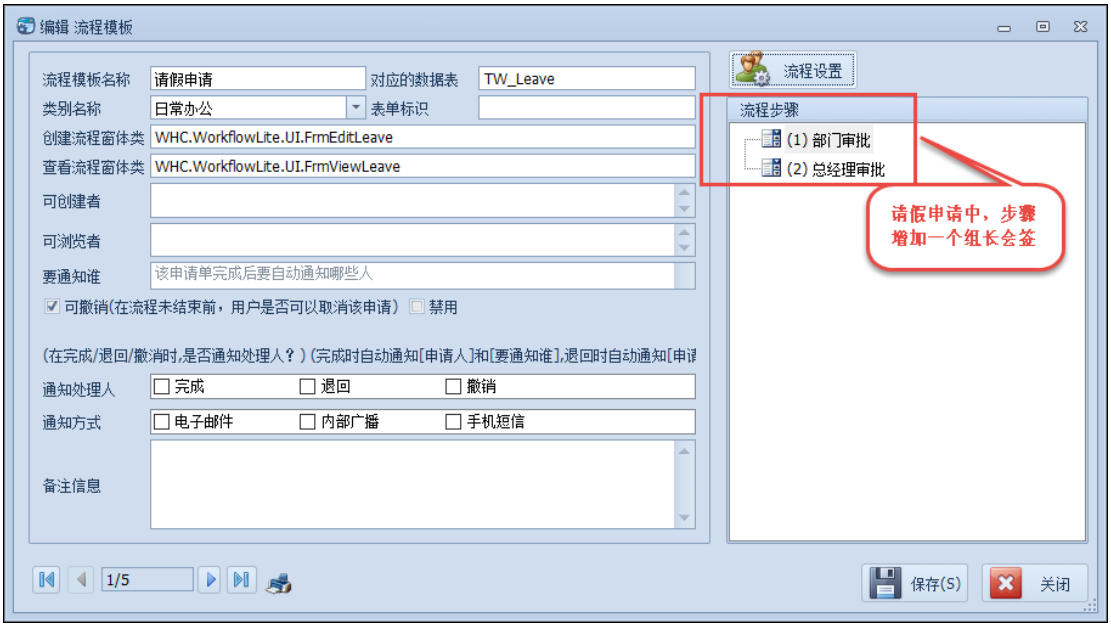
这里设置的会签处理就是其中一个步骤，一旦会签处理步骤发起会签，就会构建多个可供审批的子流程了，如下所示。



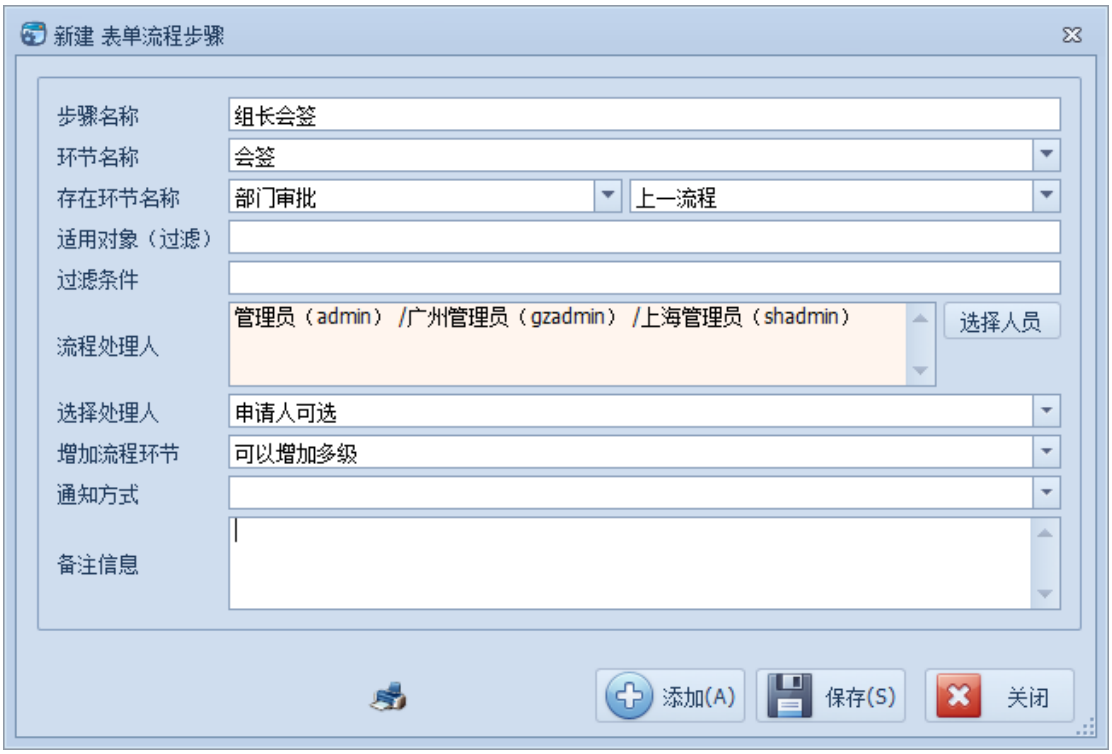
在会签发起的步骤，指定参与具体流程会签审批的人员，然后流程则会流转到不同人员进行相关的处理【待办事项】。

我在工作流中定义会签完成后，由会签发起人审核（会签结果审核），决定是否进入下一步流程，在审核过程中决定如何处理这个申请单。

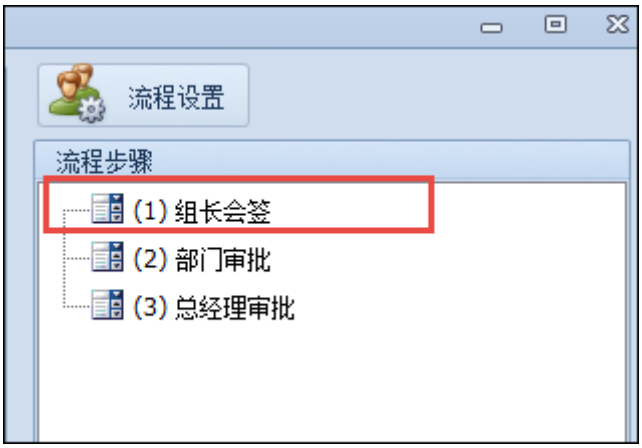
在流程定义里面，我们创建一个会签的流程步骤，我们以请假单为例，加入我们要求请假需要由各组长会签通过，然后在继续下面的部门审批、总经理审批步骤，如下所示。



增加会签后的流程步骤如下所示。

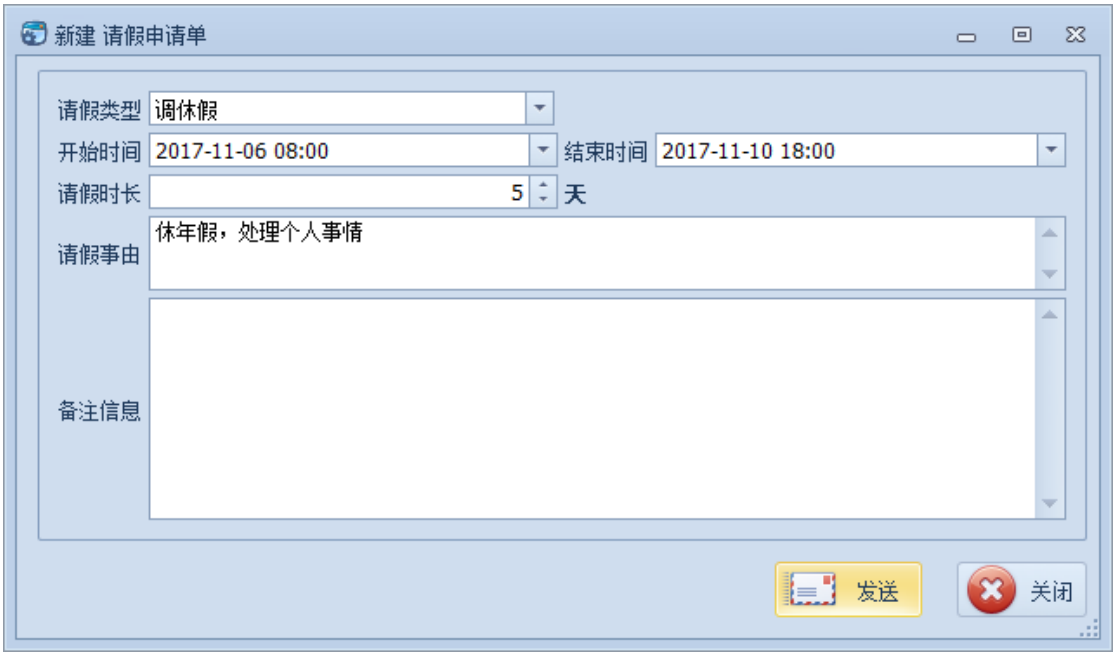


完成后可以在流程步骤列表中看到会签的步骤了，如下所示。

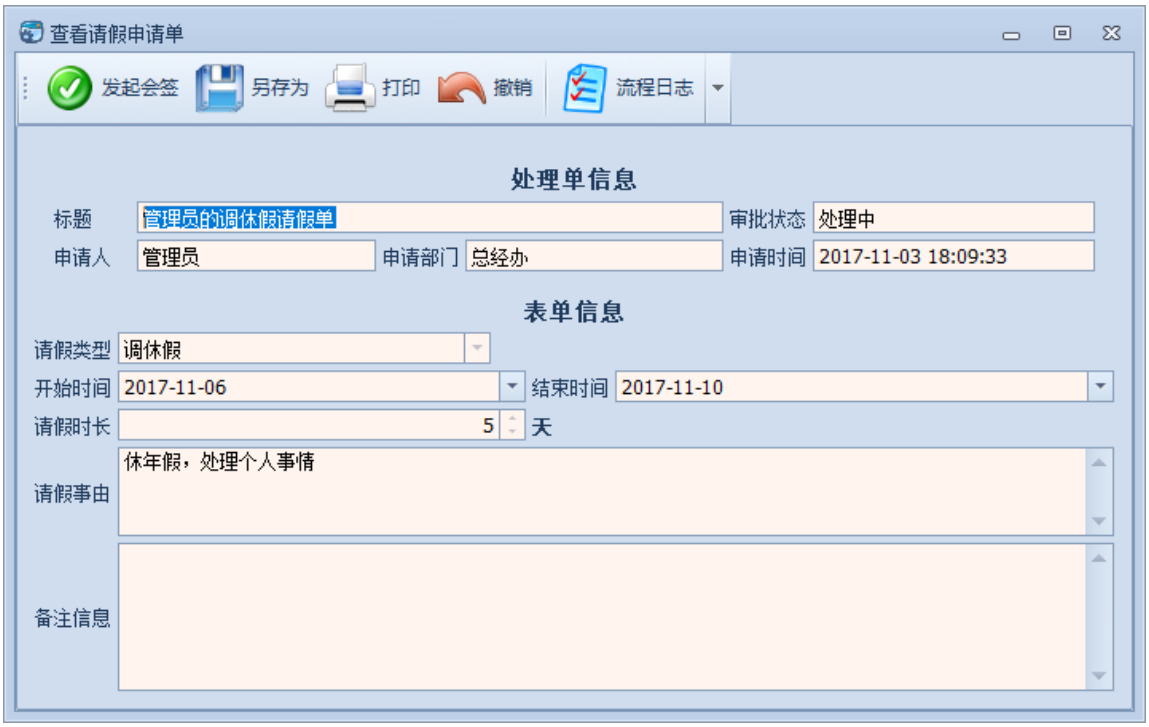


3.3.2. 会签流程处理

了解了会签的处理过程，并完成了上面的会签定义后，我们创建一个请假申请单，用来发起会签处理，介绍会签的步骤说明。



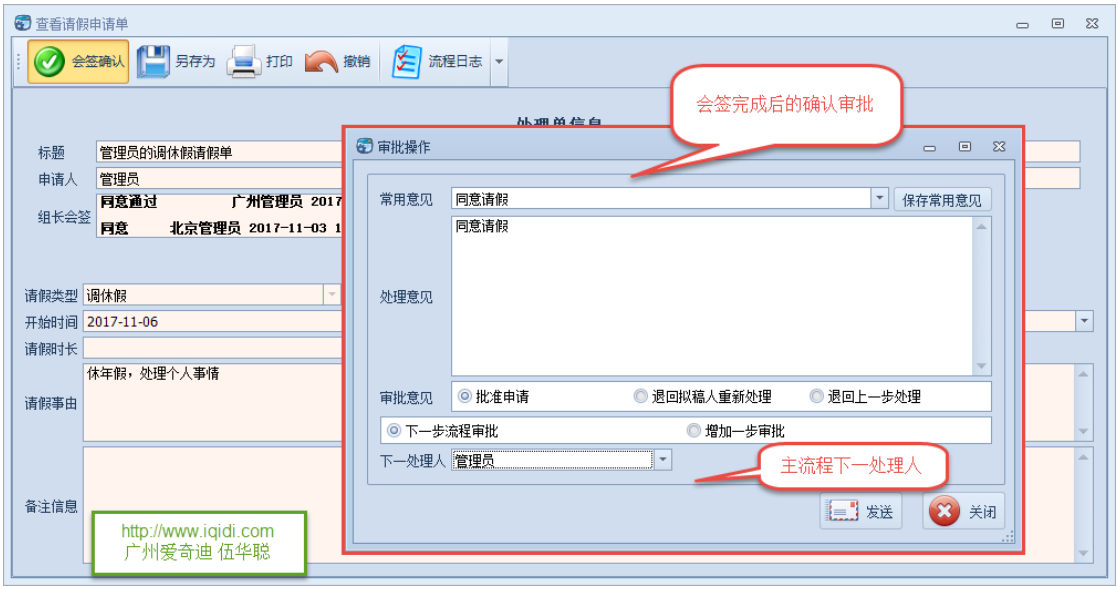
完成请假单后提交给相关处理人，处理人员在待办事项中查看申请单，如下界面所示。



那么他会发起【发起会签】的处理操作，把相关的投票权发送给各个组长进行会签处理。

会签发起后，各个参与会签的人员在【待办事项】里面处理会签意见，如下所示。

各个待审批的人员进行处理后，最后返回给会签发起人决定是否进入下一步流程，如下所示。



我们查看相关的流程日志，看到会签的流程步骤已经完成了，其他步骤还需要进行处理。

这样这个会签流程就算整体完成了，剩下的就是其他步骤的处理，按正常的审批处理即可。

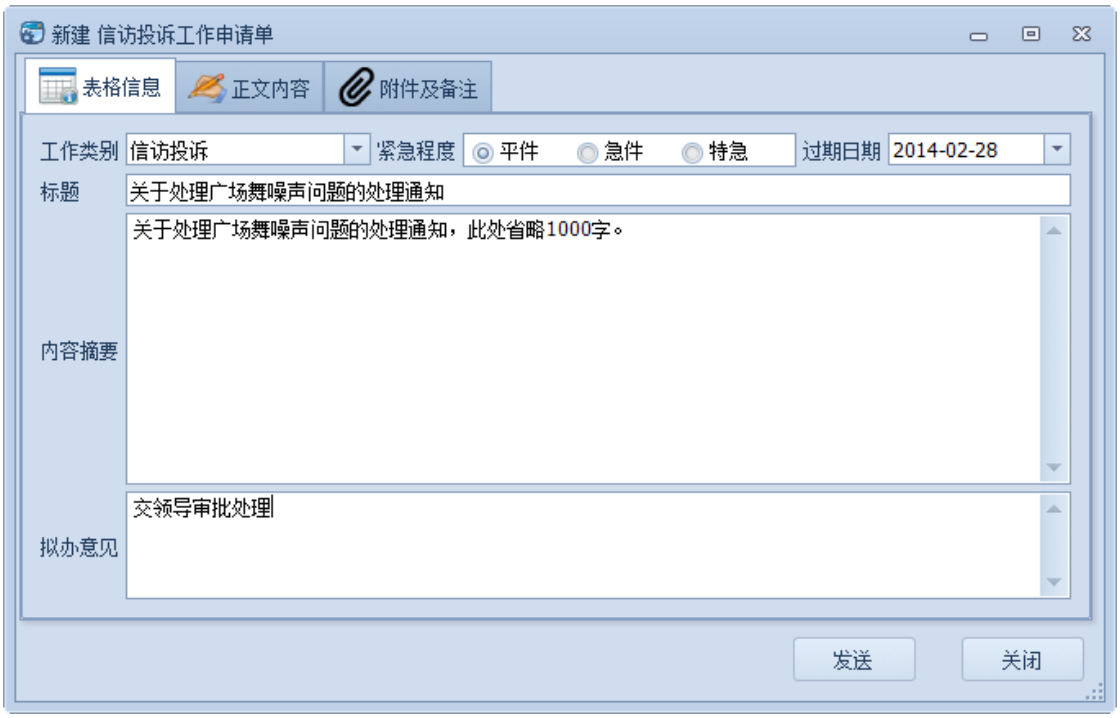
3.4. 流程审批的表单处理

3.4.1. 信访投诉工作表单处理

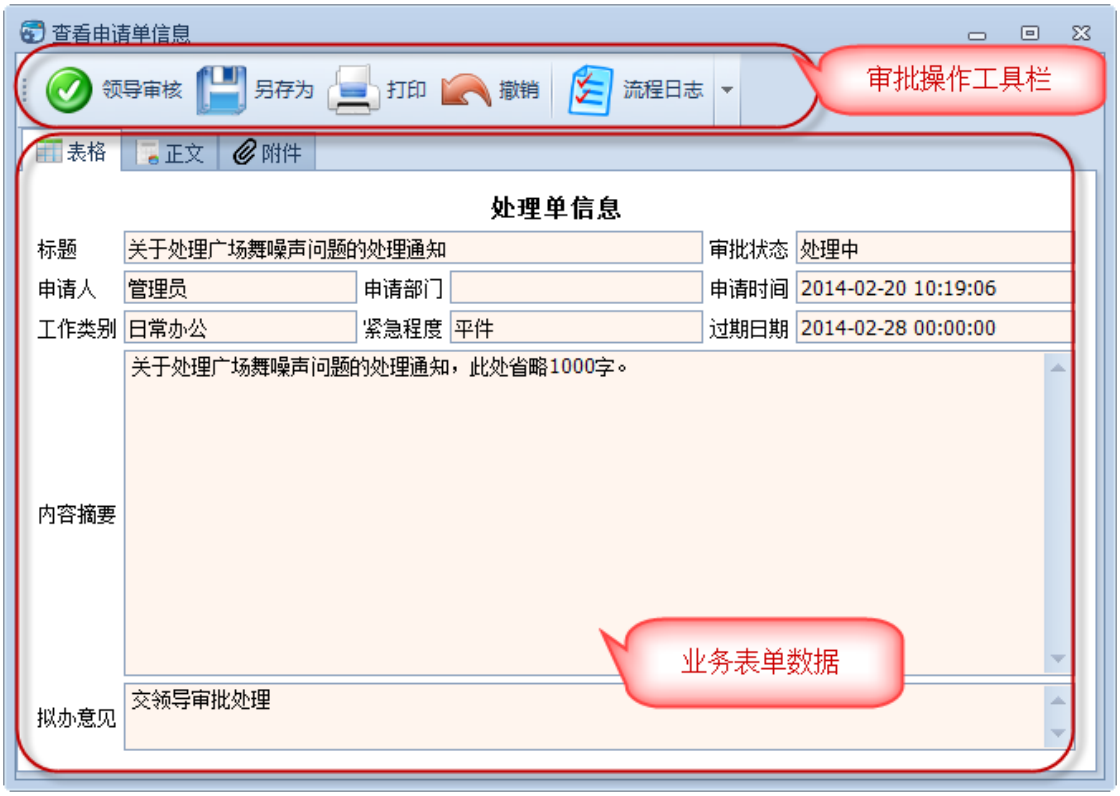
在表单的动态设计和显示方面，如果倾向零代码的动态处理，则牺牲很多内在逻辑处理；如果每个表单开发过于繁琐，则降低开发效率。

因此我觉得把流程模块作为半模块化即可，把部分界面通过代码编写方式进行整合，因此把表单的填写，表单查看做到用户控件里面，然后在界面里面引用即可。

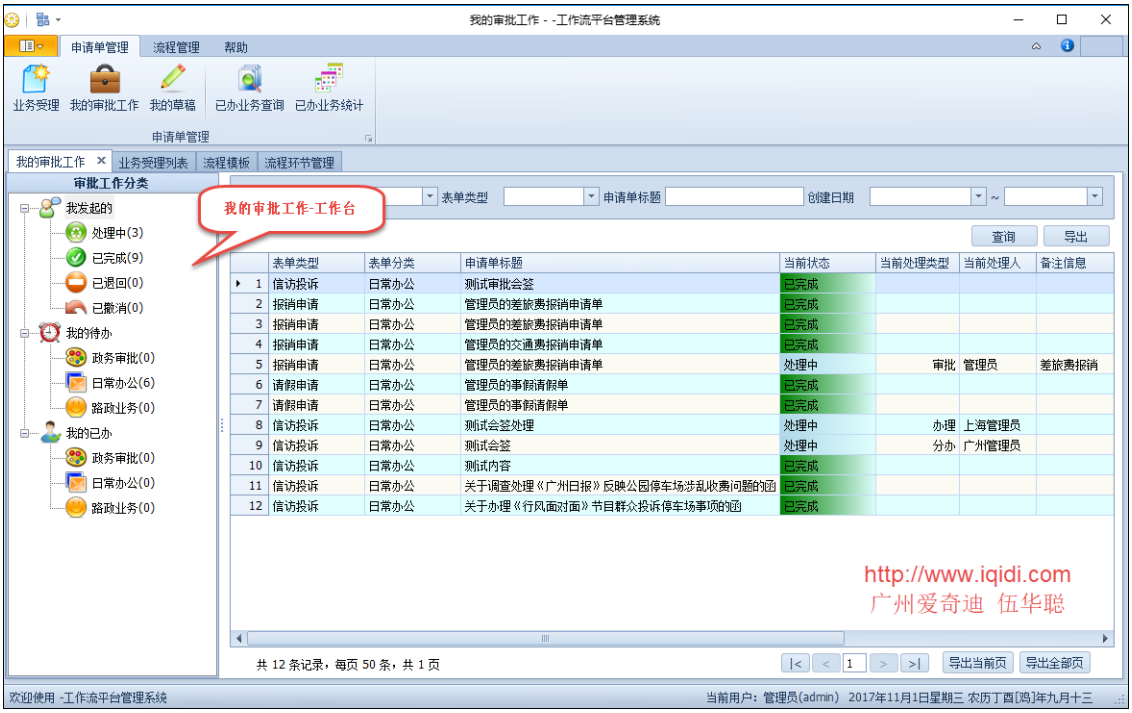
如下面的表单填写操作界面如下所示，对不同的流程表单，在项目中增加一个表单的填写界面和保存操作即可。



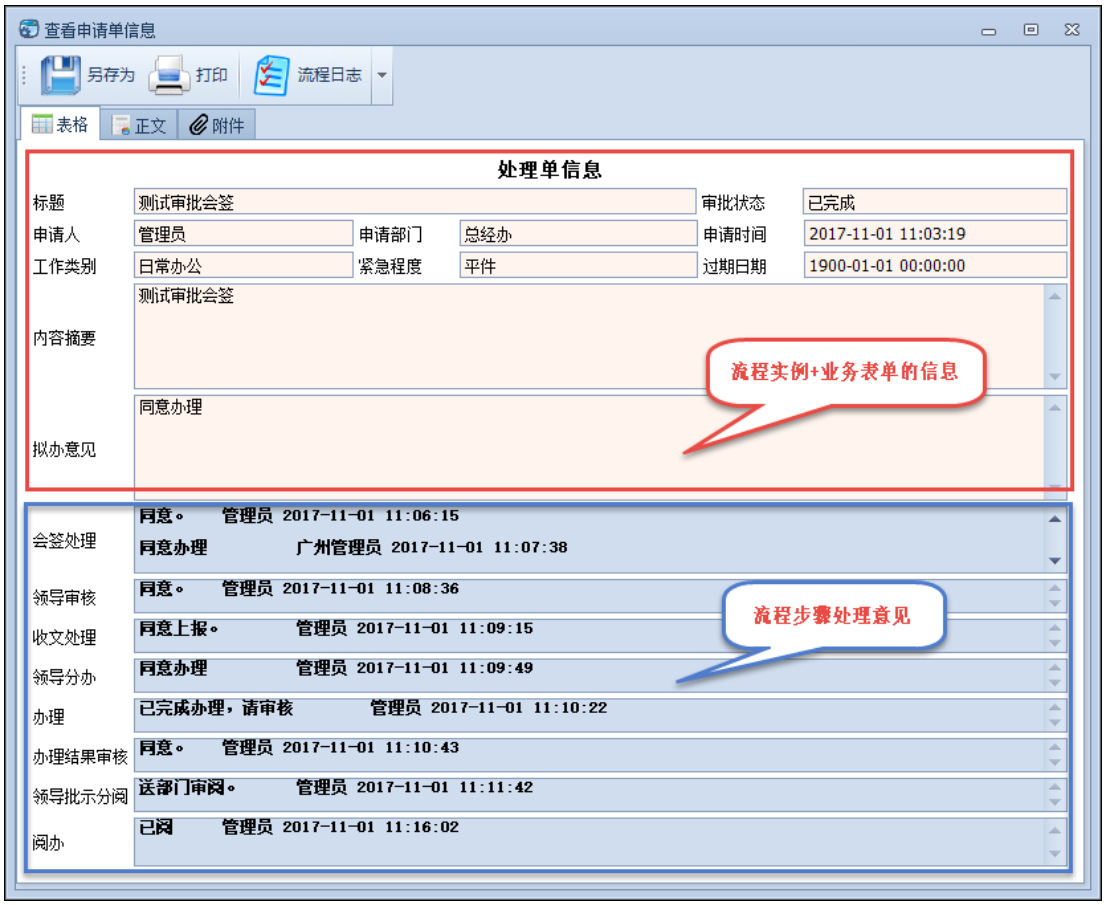
查看并处理的表单操作，我们可以先做一个查看表单信息的界面，然后整合流程的处理工具栏，组合成一个查看、处理操作一体化的流程操作。



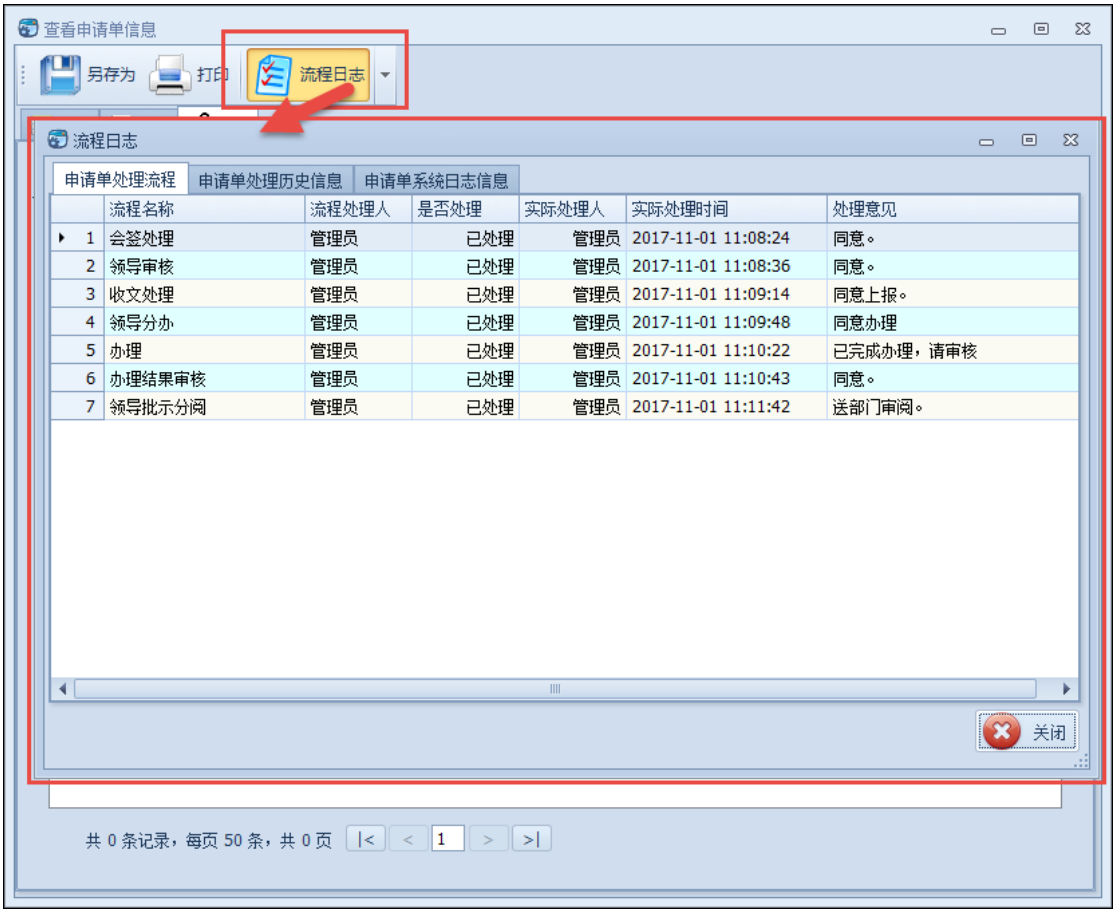
下面是具体表单的查看信息，包含了相关的处理步骤信息，以及相关的流程日志信息。

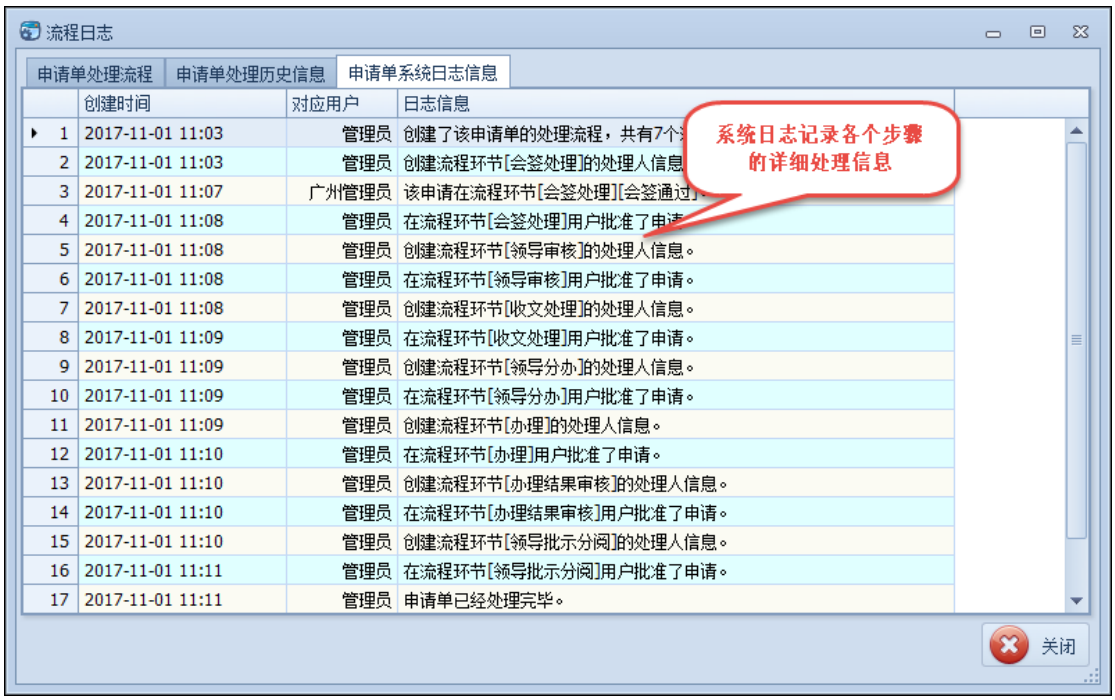


详细表单查看界面如下所示。



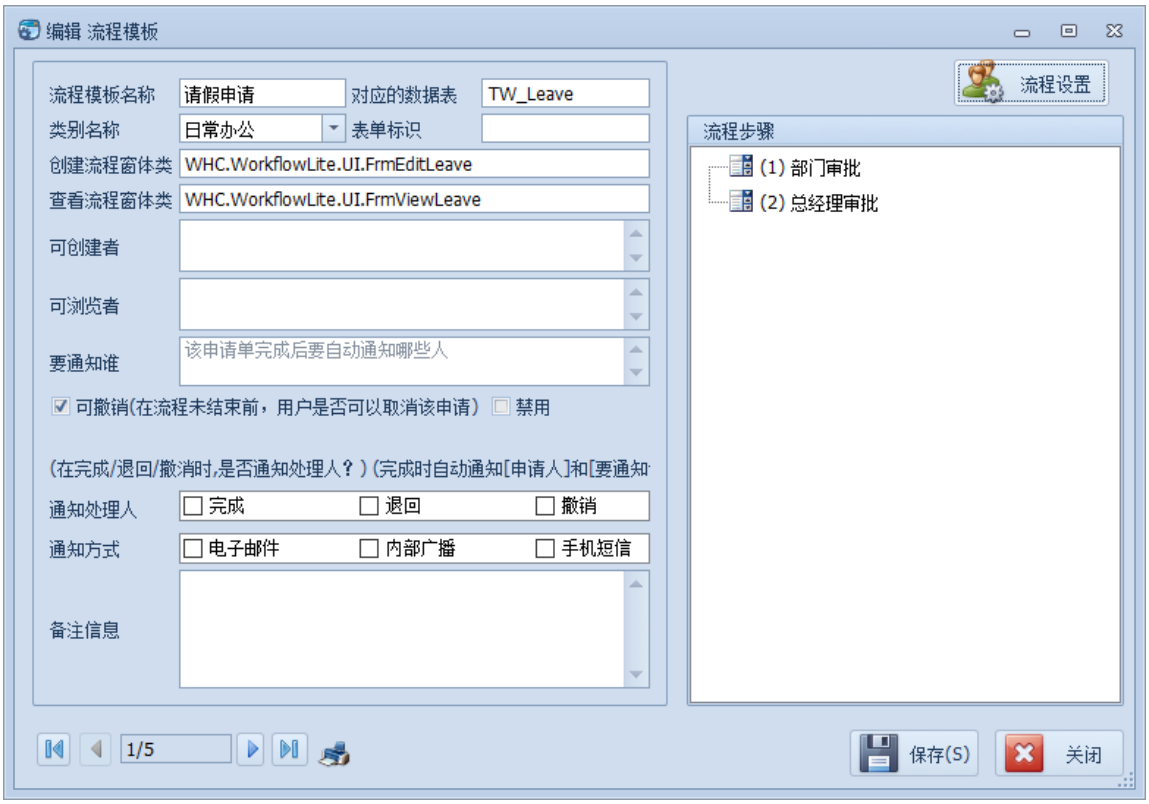
流程日志分为几个部分：申请单处理日志、申请单处理历史信息、申请单系统日志等几个部分


查看申请单信息
另存为
打印
流程日志
流程日志
申请单处理流程
申请单处理历史信息
申请单系统日志信息
流程名称 流程处理人 是否处理 实际处理人 实际处理时间 处理意见
1 会签处理 管理员 已处理 管理员 2017-11-01 11:08:24 同意。
2 领导审核 管理员 已处理 管理员 2017-11-01 11:08:36 同意。
3 收文处理 管理员 已处理 管理员 2017-11-01 11:09:14 同意上报。
4 领导分办 管理员 已处理 管理员 2017-11-01 11:09:48 同意办理
5 办理 管理员 已处理 管理员 2017-11-01 11:10:22 已完成办理，请审核
6 办理结果审核 管理员 已处理 管理员 2017-11-01 11:10:43 同意。
7 领导批示分阅 管理员 已处理 管理员 2017-11-01 11:11:42 送部门审阅。
关闭
共 0 条记录，每页 50 条，共 0 页
1

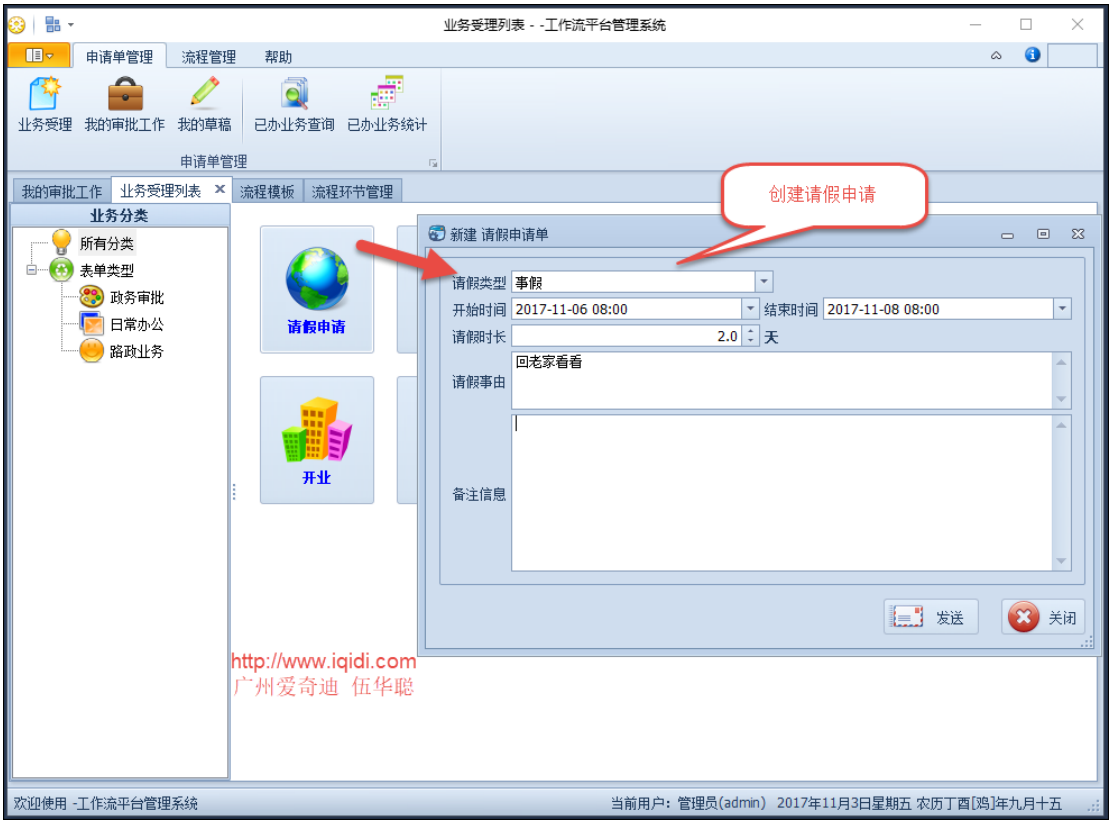

流程日志
申请单处理流程
申请单处理历史信息
申请单系统日志信息
创建时间 对应用户 日志信息
1 2017-11-01 11:03 管理员 创建了该申请单的处理流程，共有7个
2 2017-11-01 11:03 管理员 创建流程环节[会签处理]的处理人信息。
3 2017-11-01 11:07 广州管理员 该申请在流程环节[会签处理][会签通过]
4 2017-11-01 11:08 管理员 在流程环节[会签处理]用户批准了申请
5 2017-11-01 11:08 管理员 创建流程环节[领导审核]的处理人信息。
6 2017-11-01 11:08 管理员 在流程环节[领导审核]用户批准了申请。
7 2017-11-01 11:08 管理员 创建流程环节[收文处理]的处理人信息。
8 2017-11-01 11:09 管理员 在流程环节[收文处理]用户批准了申请。
9 2017-11-01 11:09 管理员 创建流程环节[领导分办]的处理人信息。
10 2017-11-01 11:09 管理员 在流程环节[领导分办]用户批准了申请。
11 2017-11-01 11:09 管理员 创建流程环节[办理]的处理人信息。
12 2017-11-01 11:10 管理员 在流程环节[办理]用户批准了申请。
13 2017-11-01 11:10 管理员 创建流程环节[办理结果审核]的处理人信息。
14 2017-11-01 11:10 管理员 在流程环节[办理结果审核]用户批准了申请。
15 2017-11-01 11:10 管理员 创建流程环节[领导批示分阅]的处理人信息。
16 2017-11-01 11:11 管理员 在流程环节[领导批示分阅]用户批准了申请。
17 2017-11-01 11:11 管理员 申请单已经处理完毕。
系统日志记录各个步骤的详细处理信息
关闭

3.4.2. 请假申请表单

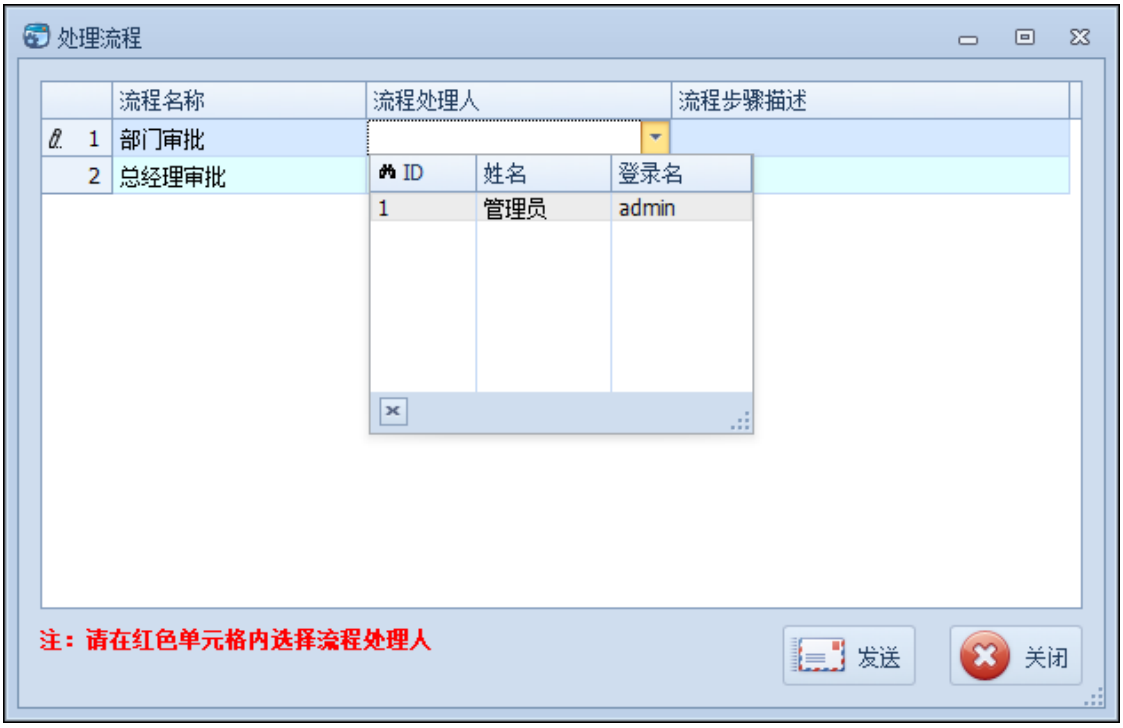
这里介绍的具体功能，是在业务表单已经完成好，具体的使用过程，我们以请假申请单为例，在我们创建对应的流程步骤如下所示后，



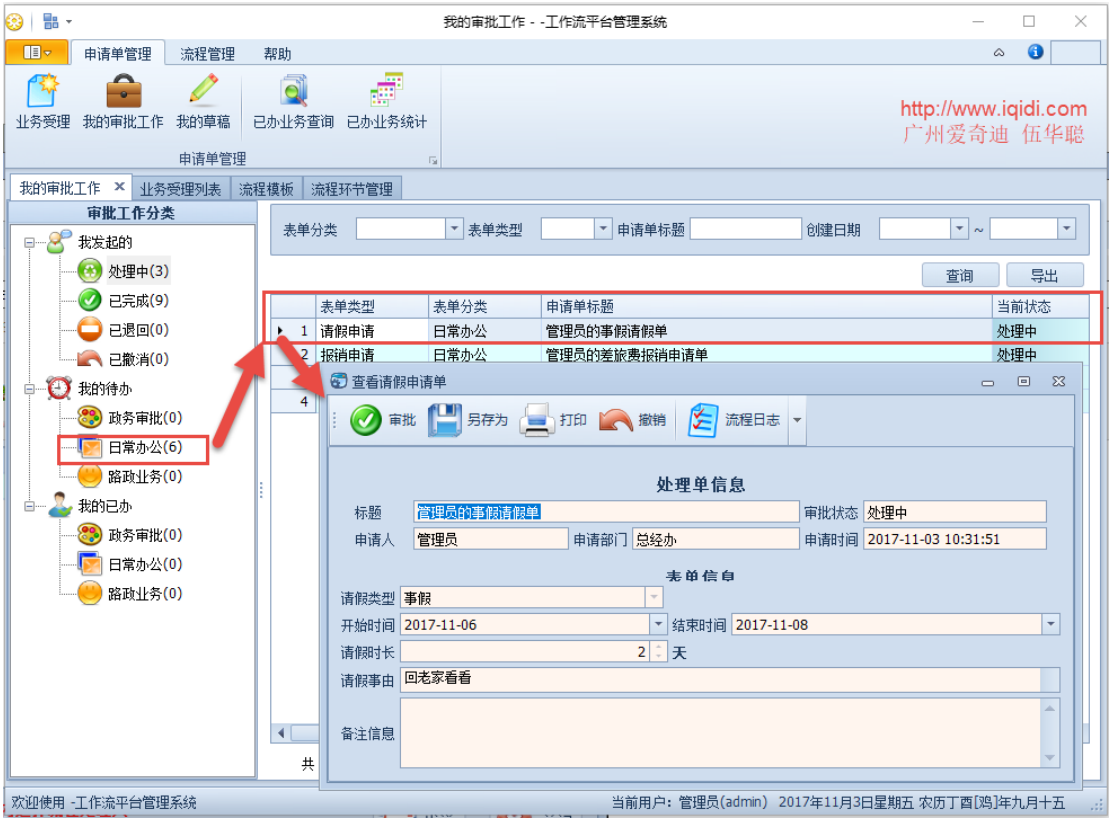
我们就可以开始创建具体的业务审批流程了，在业务受理列表里面选择对应的请假申请，弹出相应的创建业务流程表单窗体。



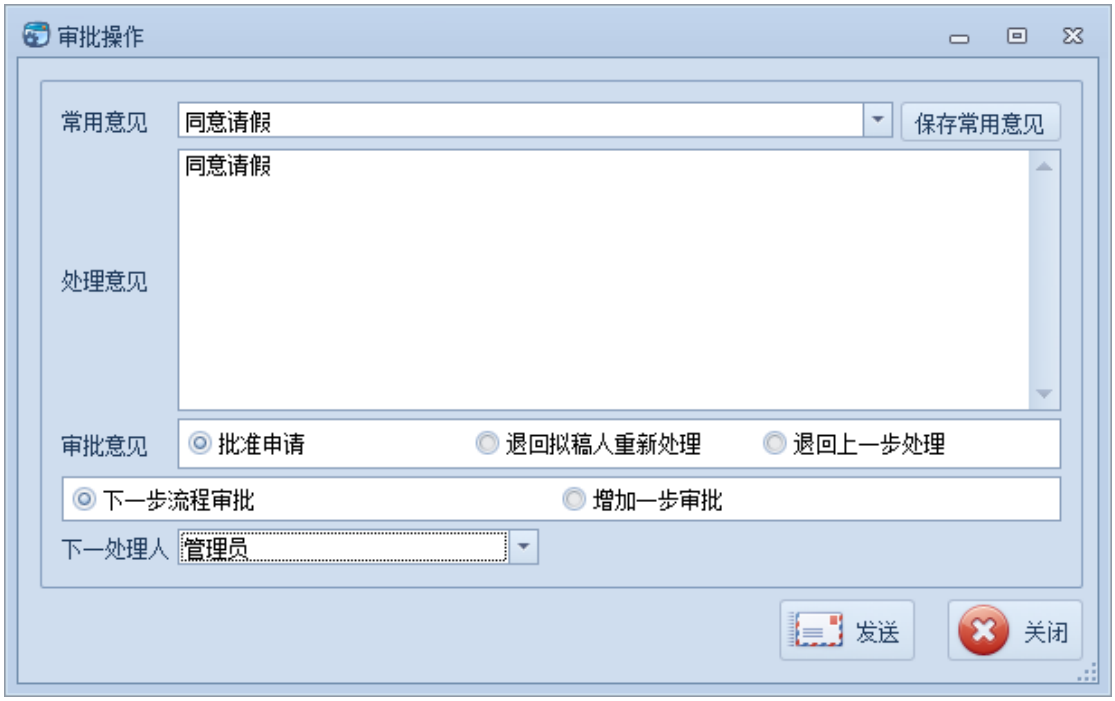
单击【发送】按钮，会要求选择下一步的审批人，如下所示。



这样对应的审批人员进去后，在我的待办业务列表里面，就可以看到刚才的表单了，双击可以进行查看，以及相关的审批处理工作。

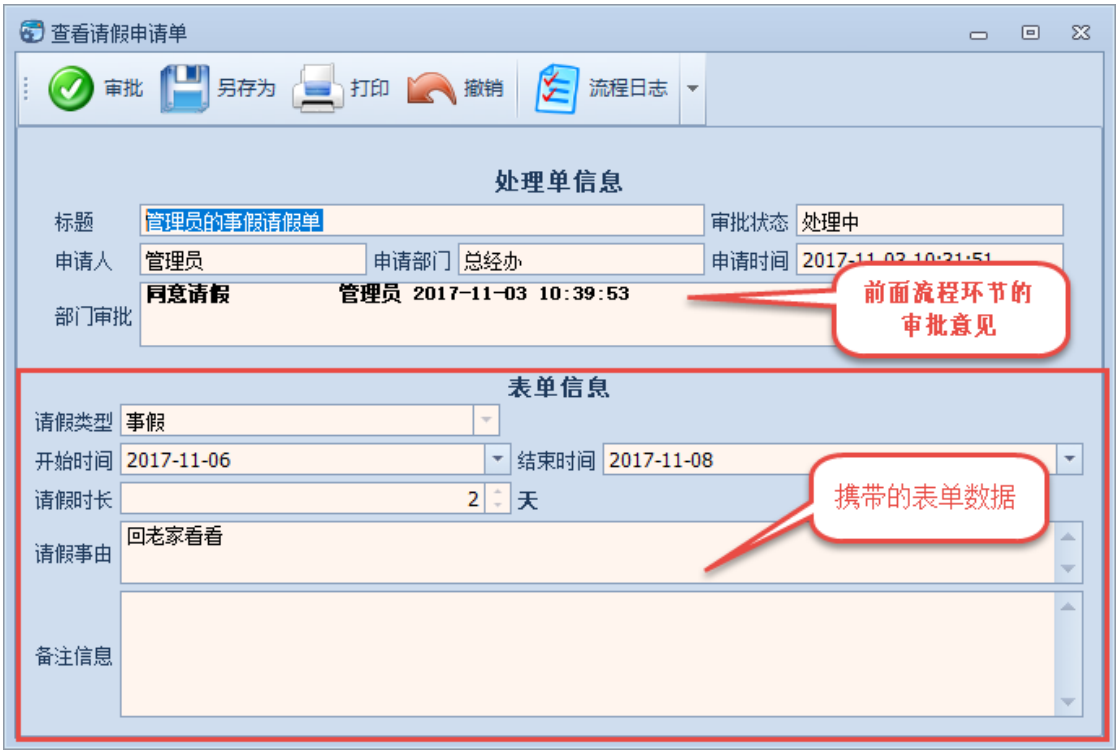


单击表单左上角的工具栏按钮【审批】，可以进行流程的审批处理操作了。

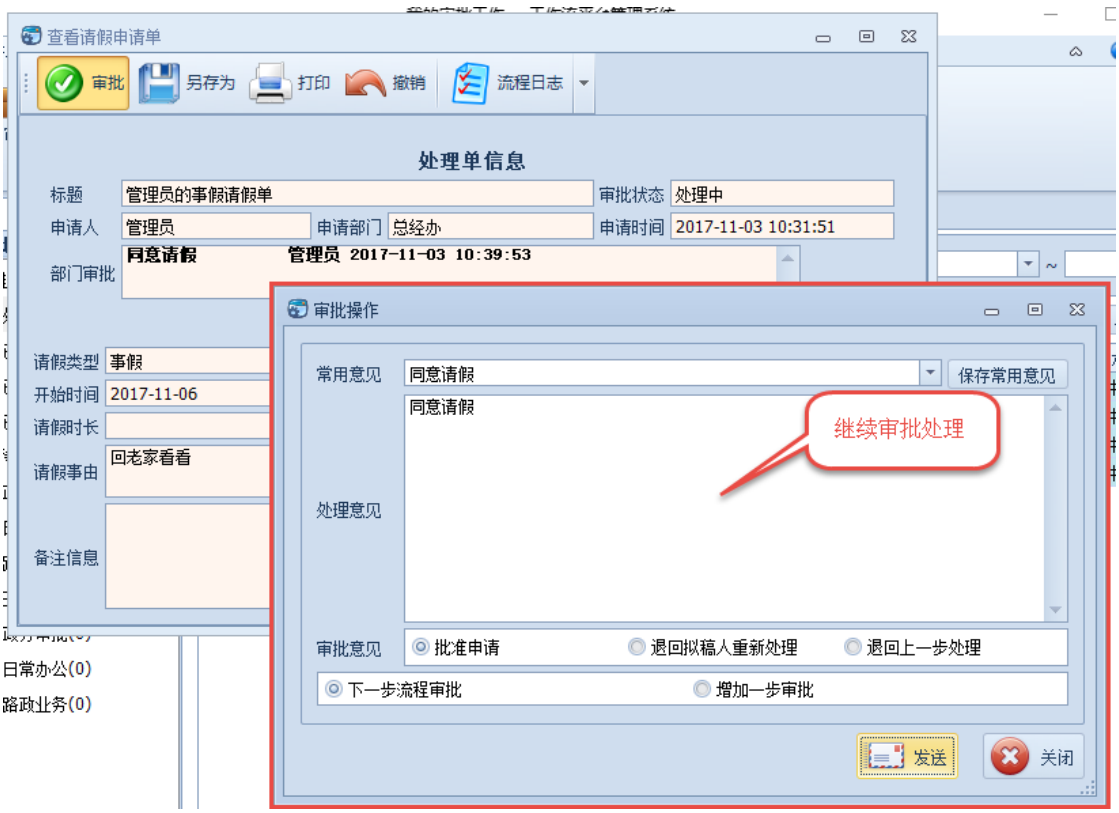


在处理完成相关的审批操作后，流程就会自动流转到了表单定义的下一流程人了，这里为了方便选择了管理员的角色来处理而已。

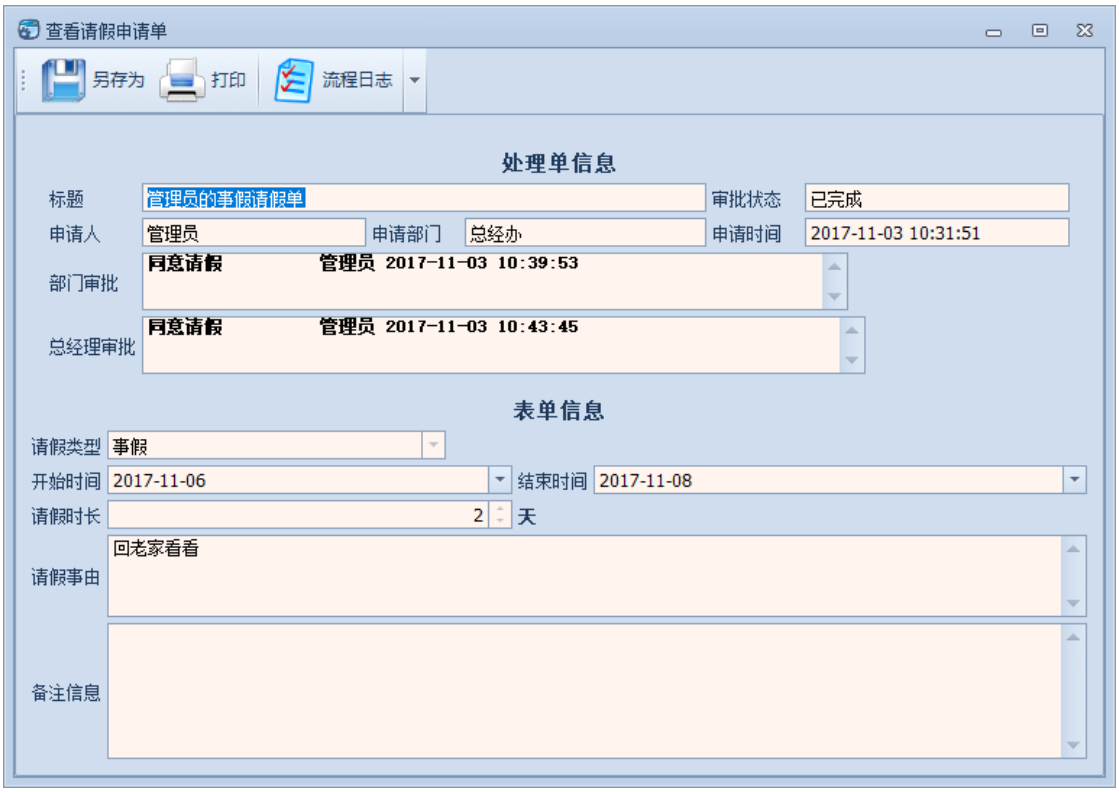
流程人登陆后查看具体业务申请单后，可以看到前面的审批意见等信息，如下所示。



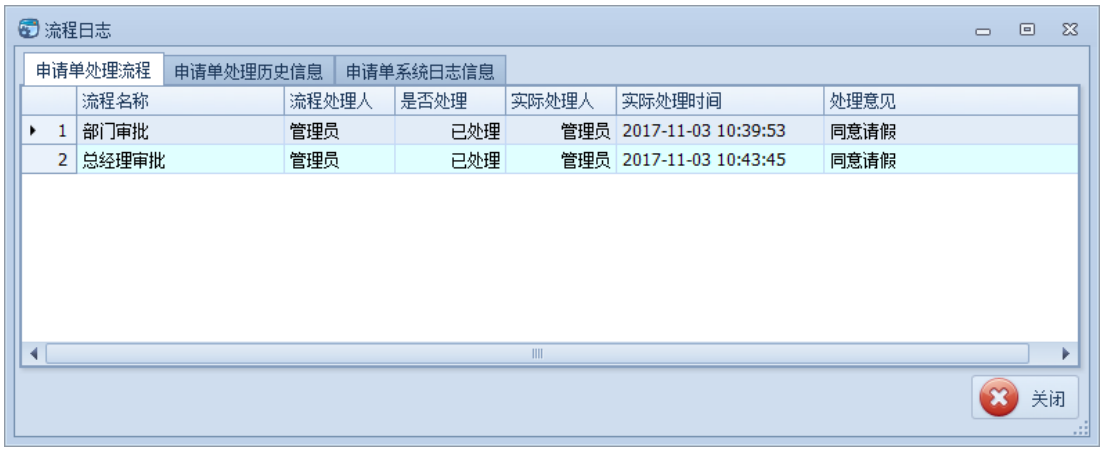
继续【审批】处理环节，这样两个环节的流程就处理完成了。



最后我们查看完成的业务申请单，可以看到相关的审批信息了。



当然也可以查看具体的业务流程日志。

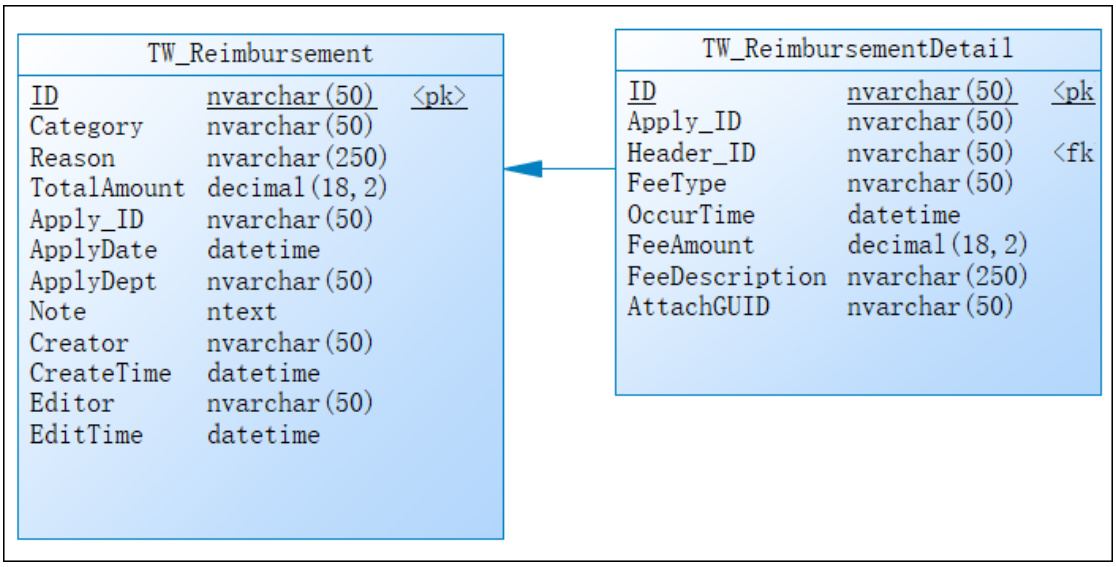


整个流程步骤全部完成了。

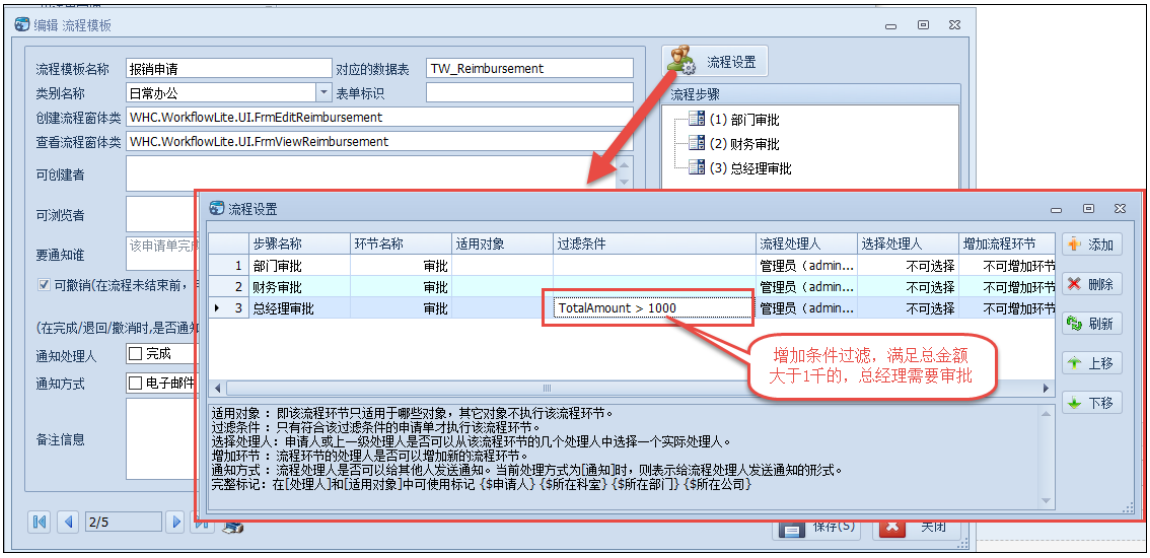
3.4.3. 报销申请单

为了增加复杂一点的表单我们引入一个含有主从表的业务表单，明细表包括报销的具体列表内容。

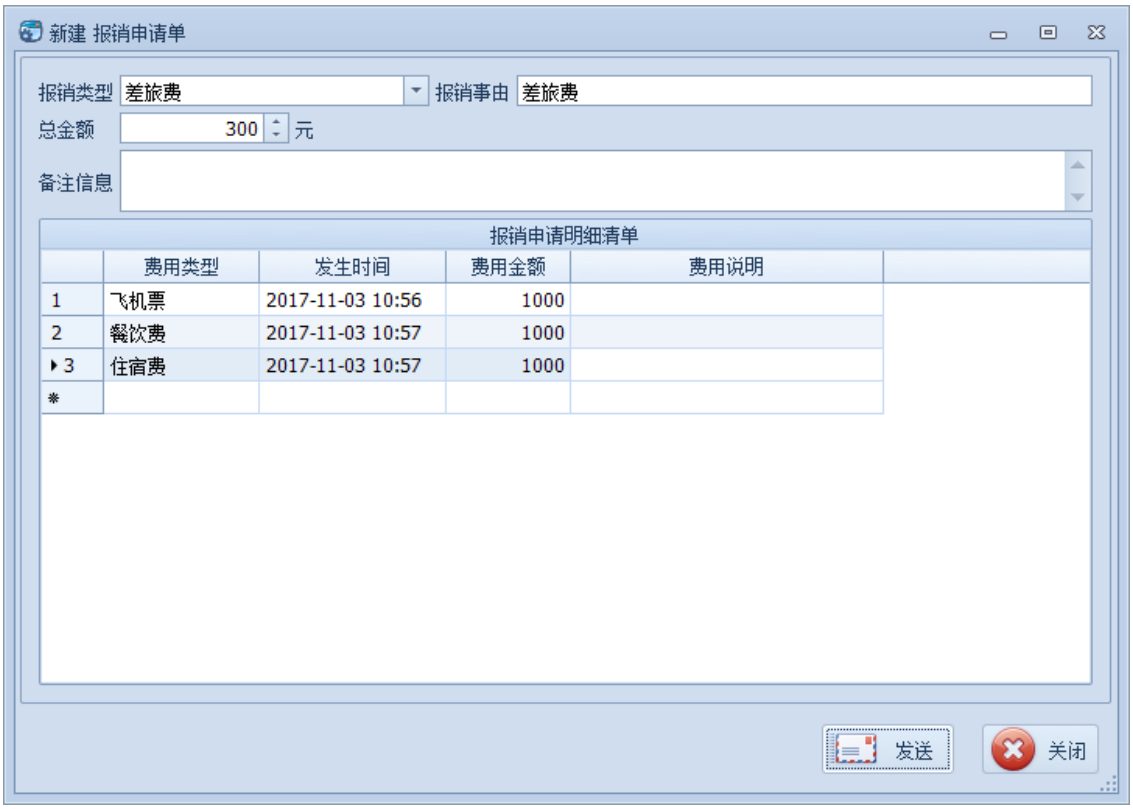
具体的业务表单设计如下所示。



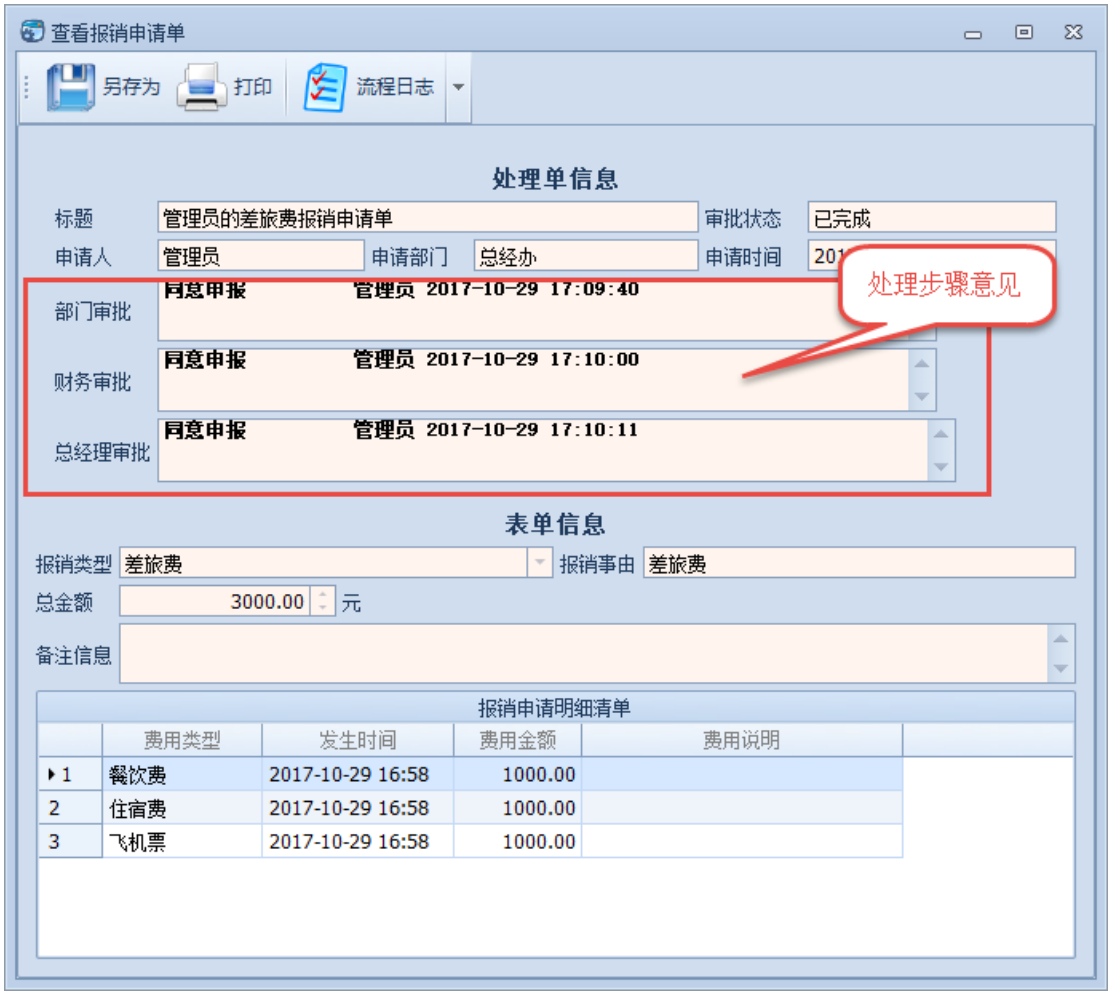
首先我们定义好业务流程，如下所示。



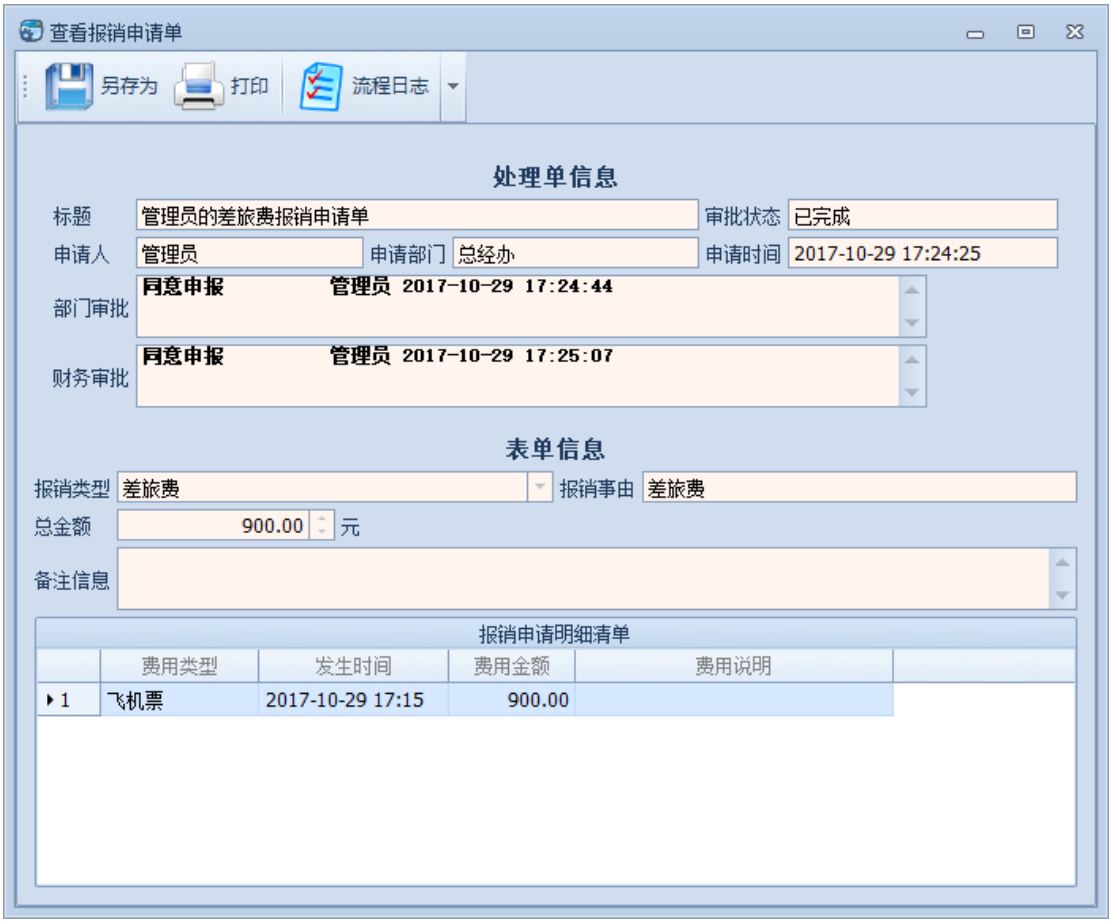
创建一个业务表单，如下所示。



其中几个流程的审批和上面的处理操作差不多，最后完成表单后，那么这个总共参与的步骤是三个步骤，包括了总经理审批环节了，和下面的表单过程类似，界面效果如下所示。

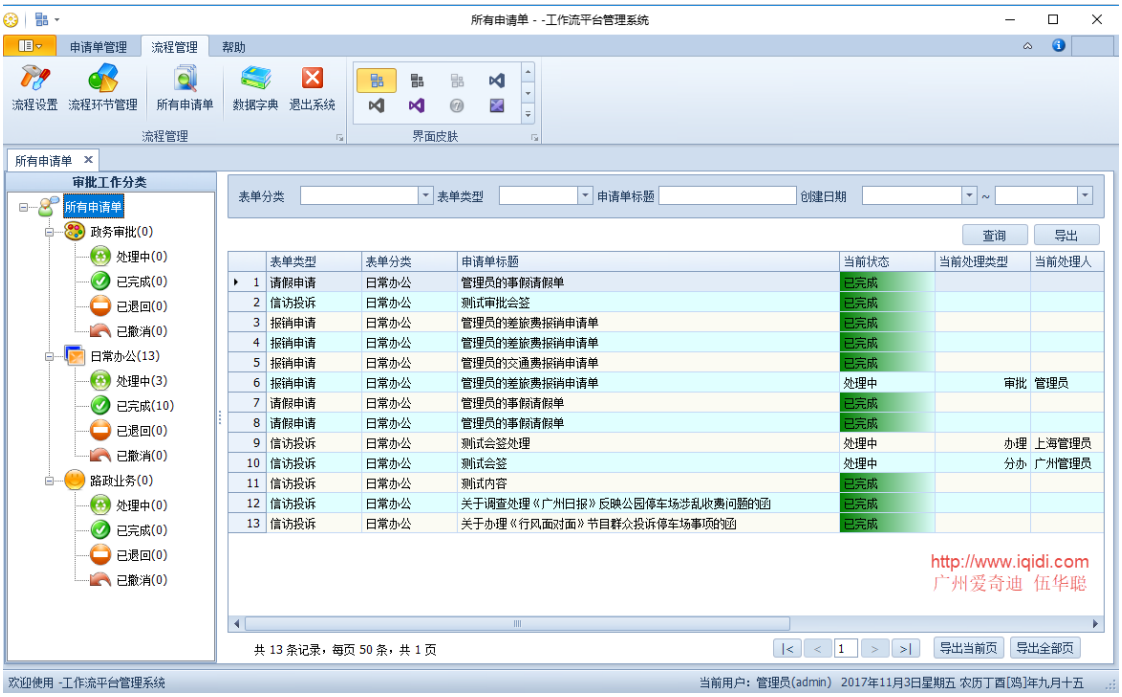


如果是报销费用少于条件值（3000）的，那么审批的环节就不包括总经理审批环节了，如下表单所示。

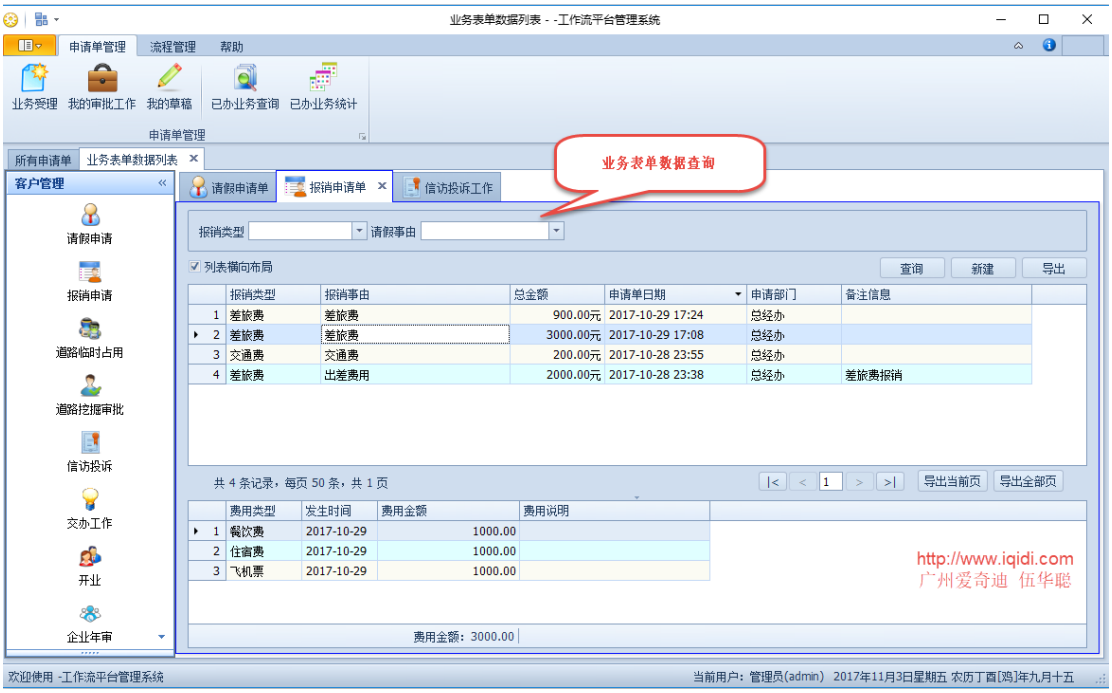


以上就是两类表单，一个是普通的单表表单、一个是包含明细表的主从表表单的流程处理，以及步骤中包含相关的过滤条件，满足条件的则包含该条件的审批处理，否则系统自动跳过这个步骤，直接跳转到下一步或者完成整个申请单的处理。

有时候为了方便查看对应的业务表单，一般提供一个所有表单的入口给管理查看 ，方便进行维护管理，如下所示。



管理员可以在这里对业务表单进行删除、撤回等特殊处理，另外，我们也可以提供一个业务数据的入口方便查询统计，如下所示的界面。

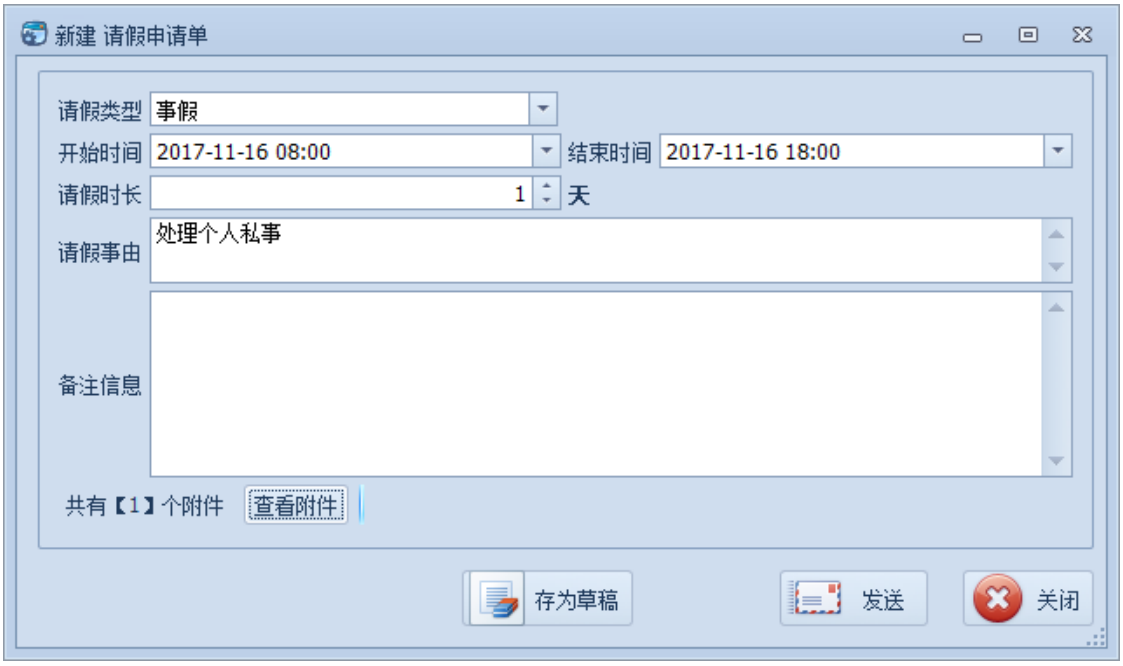


3.5. 申请单草稿处理

在我们开发工作流模块的时候，有时候填写申请单过程中，暂时不想提交审批，那么可以暂存为草稿，以供下次继续填写或者提交处理，那么这个草稿的功能是比较实用的，否则对于一些填写内容比较多的申请单，每次要重填写很多数据，那会被用户骂的，从用户的角

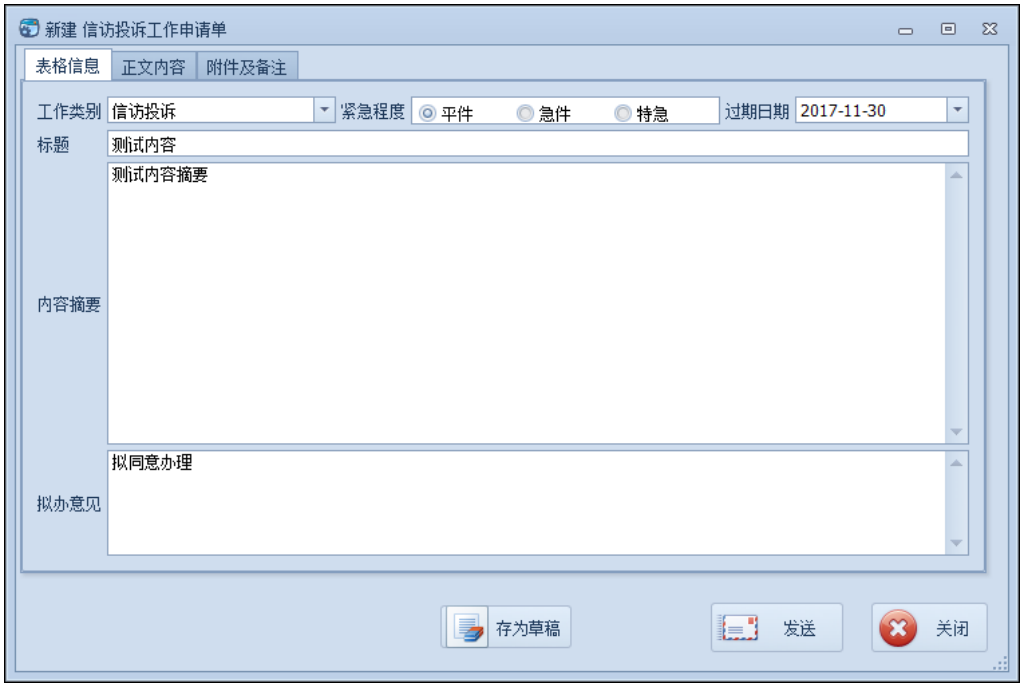
度上来讲，提供草稿保存的功能是比较友好的。本篇随笔介绍在工作流模块中使用一种通用的存储方式来存储及显示申请单草稿的信息。

在我们提交申请前，我们一般是需要填写一些相关的资料，如下界面所示。



这个表单记录的信息不多，不过提供存为草稿的功能也是要的，我们所有申请单都提供这个标准功能。

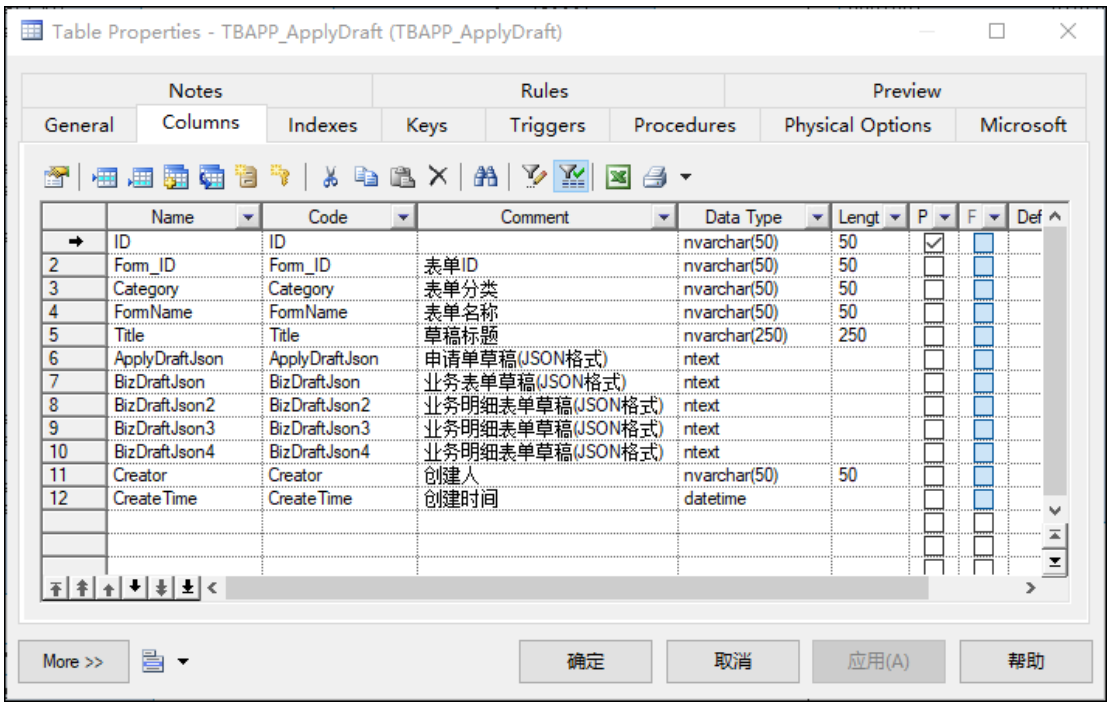
或者复杂一点的申请单



以往做过草稿保存，把记录复制在正式的申请单里面，设置它为草稿状态即可，这种方式可以实现，不过不好统一处理，本篇随笔介绍的是所有草稿存储在一个表里面，我们定义

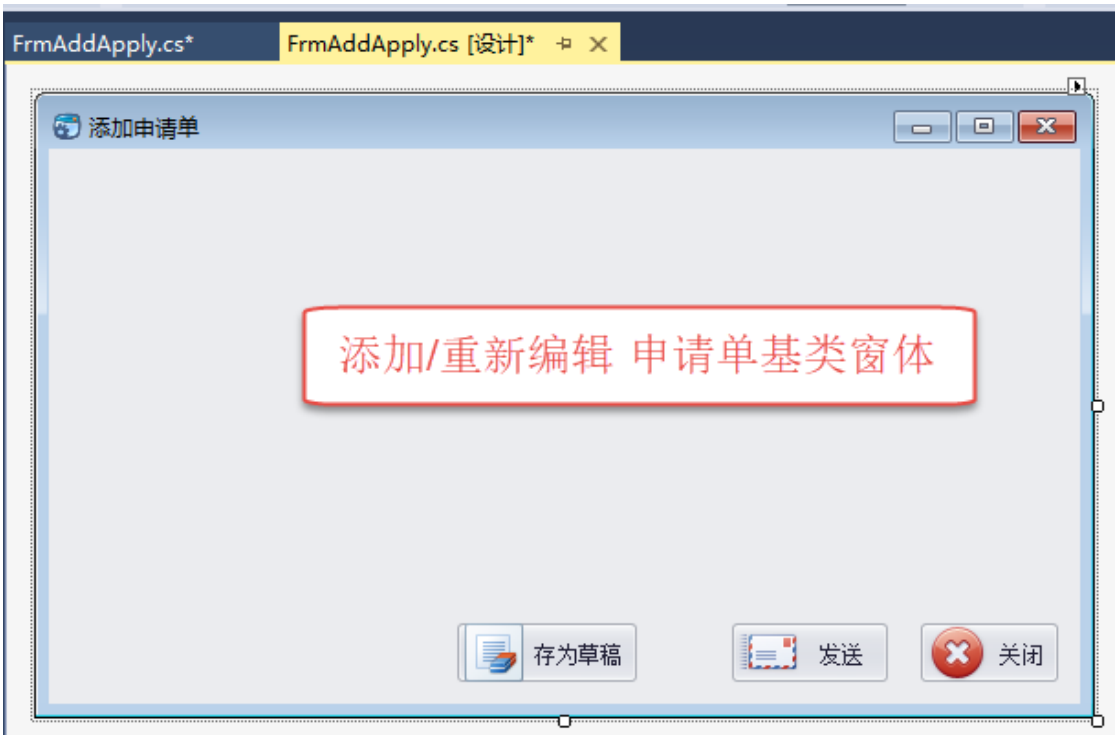
一些字段用来存储对应信息的 JSON 数据，然后需要的时候，把它们逐一解析为对应的对象即可，这种我们可以在基类窗体里面封装它的【存为草稿】的逻辑处理了。

首先我们定义一个存储草稿信息，可以对单表，也可以对主从表的数据，我们把它存储为对应的 JSON 字段即可，设计草稿的数据表如下所示。



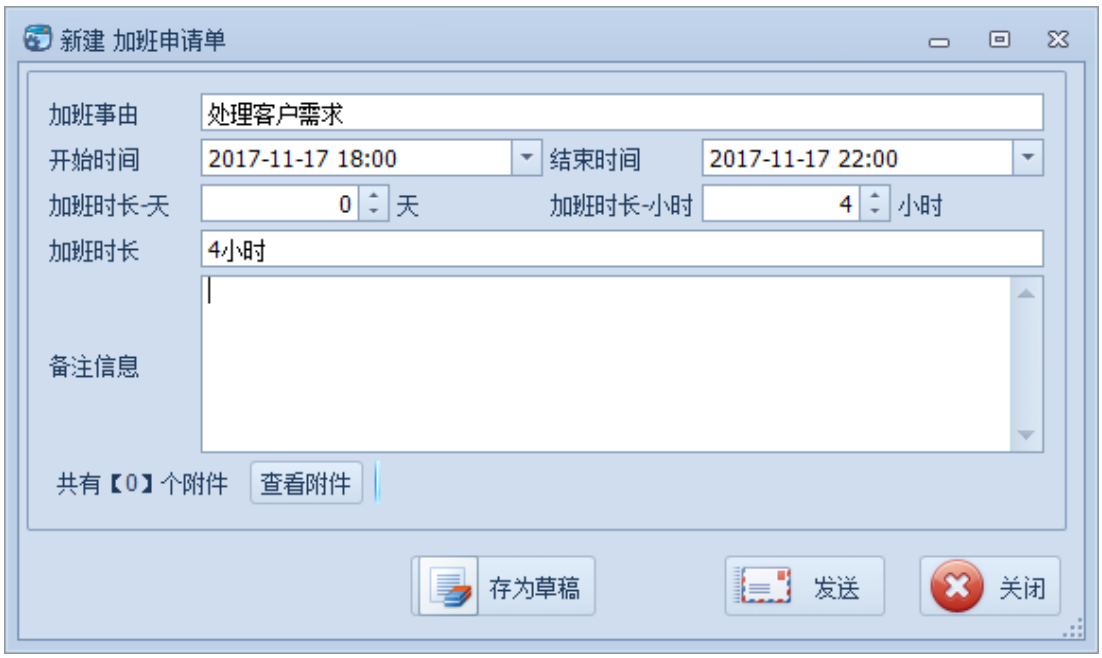
	Name	Code	Comment	Data Type	Lengt	P	F	Def
➡	ID	ID		nvarchar(50)	50	☑		
2	Form_ID	Form_ID	表单ID	nvarchar(50)	50			
3	Category	Category	表单分类	nvarchar(50)	50			
4	FormName	FormName	表单名称	nvarchar(50)	50			
5	Title	Title	草稿标题	nvarchar(250)	250			
6	ApplyDraftJson	ApplyDraftJson	申请单草稿(JSON格式)	ntext				
7	BizDraftJson	BizDraftJson	业务表单草稿(JSON格式)	ntext				
8	BizDraftJson2	BizDraftJson2	业务明细表单草稿(JSON格式)	ntext				
9	BizDraftJson3	BizDraftJson3	业务明细表单草稿(JSON格式)	ntext				
10	BizDraftJson4	BizDraftJson4	业务明细表单草稿(JSON格式)	ntext				
11	Creator	Creator	创建人	nvarchar(50)	50			
12	CreateTime	CreateTime	创建时间	datetime				

在申请单填写的基类窗体里面，我们定义界面如下所示。



然后我们在基类提供一个通用的业务草稿保存处理函数，供子类进行调用即可。

对于普通单表的申请单处理，如下界面所示。

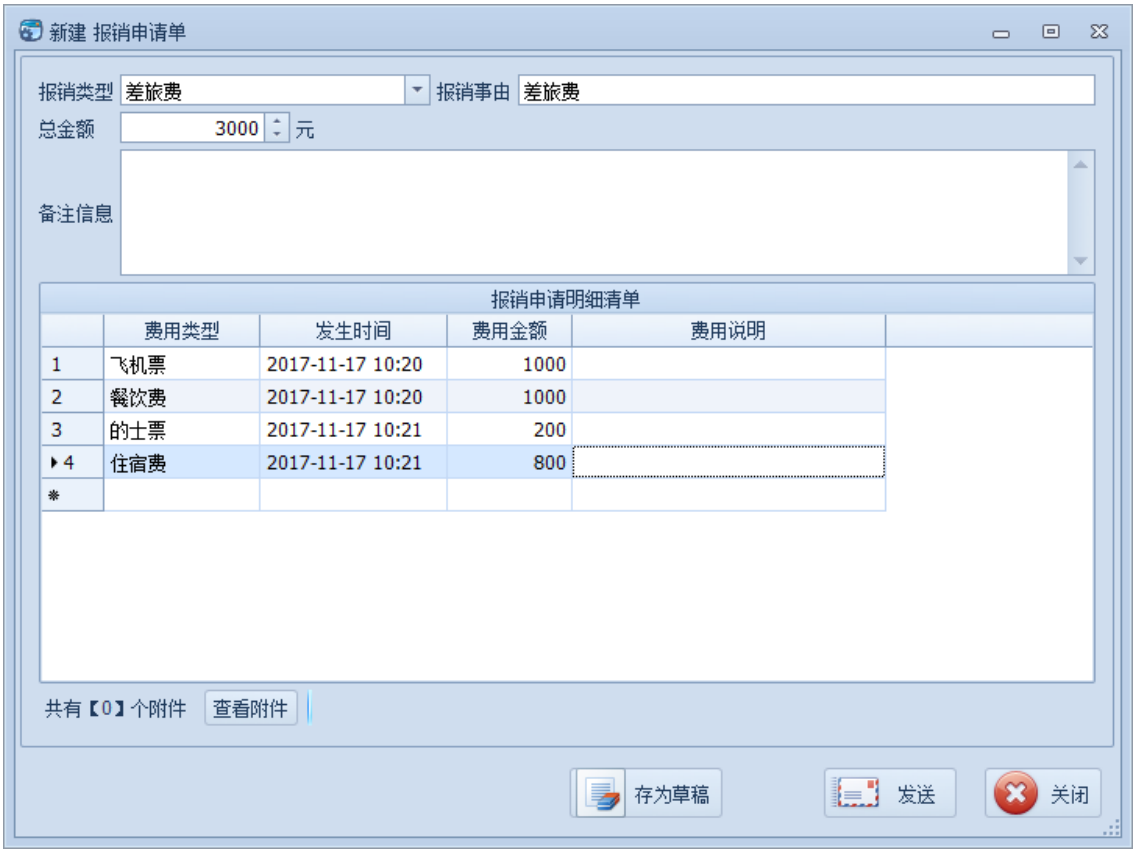


那么它的保存草稿的功能代码是如何实现的？

```
/// <summary>
/// 保存草稿处理
/// </summary>
private void btnSaveDraft_Click(object sender, EventArgs e)
{
    string title = string.Format("{0}的付款申请单【{1}】（草稿）", LoginUserInfo.FullName, DateTime.Now.ToShortDateString());

    var info = tempInfo;//必须使用存在的局部变量，因为部分信息可能被附件使用
    SetInfo(info);
    info.Creator = base.LoginUserInfo.ID;
    info.CreateTime = DateTime.Now;

    //保存草稿：对象信息转换为JSON进行保存
    SaveDraft(title, info.ToJson());
}
```

这里保存实际上就是获取对应的表单信息转换为 JSON 存储即可。

例如对于费用及费用明细的报销处理界面，如下所示。



那么我们的草稿处理有什么不同呢？

在填写申请单的子类我们实现按钮【存为草稿】的单击事件处理，代码如下所示。

```
/// <summary>
/// 保存申请单草稿的处理
/// </summary>
private void btnSaveDraft_Click(object sender, EventArgs e)
{
    string title = string.Format("{0}的{1}报销申请单【{2}】(草稿)", LoginUserInfo.FullName, this.txtCategory.Text, DateTime.Now.ToShortDateString());
    var info = tempInfo;//必须使用存在的局部变量，因为部分信息可能被附件使用
    SetInfo(info);
    info.Creator = base.LoginUserInfo.ID;
    info.CreateTime = DateTime.Now;

    //获取费用明细
    var list = GetDetailList();

    //保存草稿处理：如果有多个明细，可以增加在后面
    SaveDraft(title, info.ToJson(), list.ToJson());
}
```

我们这里需要把费用信息、明细信息的对象转换为 JSON 对象，然后统一调用基类的保存草稿函数即可。

而对于草稿信息加载，还原为实际表单的信息显示，我们处理代码就是先解析 JSON 对象，转换为实际表单对象，然后进行界面赋值展示即可，如下代码所示。

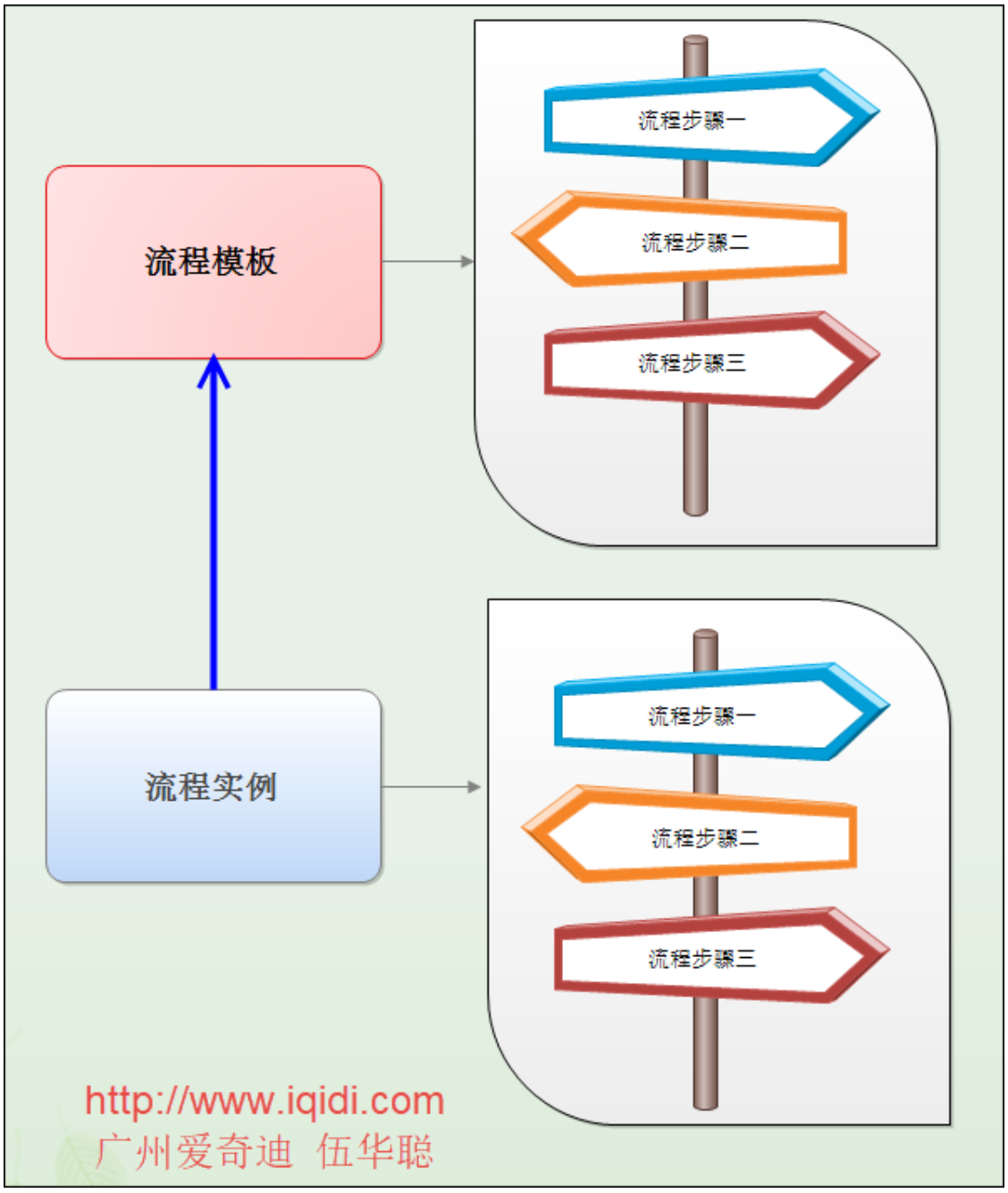
```
/// <summary>
/// 数据显示的函数
/// </summary>
15 个引用
public override void DisplayData()
{
    InitDictItem();//数据字典加载（公用）

    //由于申请单一般是用申请表单入口，而非业务数据表，因此只能传入ApplyId获取信息
    if (!string.IsNullOrEmpty(ApplyId))
    {
        #region 显示信息
        var applyInfo = BLLFactory<Apply>.Instance.FindByID(ApplyId);
        var info = BLLFactory<Payment>.Instance.FindByApplyId(ApplyId);
        if (applyInfo != null && info != null)
        {
            this.FormID = applyInfo.FormId;//赋值给FormID
            DisplayInfo(info);
        }
        #endregion
        //this.btnOK.Enabled = HasFunction("Payment/Edit");
    }
    else if (!string.IsNullOrEmpty(DraftId))
    {
        var draftInfo = BLLFactory<ApplyDraft>.Instance.FindByID(DraftId);
        if (draftInfo != null)
        {
            var info = JsonConvert.DeserializeObject<PaymentInfo>
            if (info != null)
            {
                DisplayInfo(info);
            }
        }

        //this.btnOK.Enabled = Portal.gc.HasFunction("Payment/Add");
    }

    //tempInfo在对象存在则为指定对象，新建则是全新的对象，但有一些初始化的GUID用于附件上传
    SetAttachInfo(tempInfo);

    //SetPermit(); //默认不使用字段权限
}
```

string FrmAddApply.DraftId
申请单草稿的ID

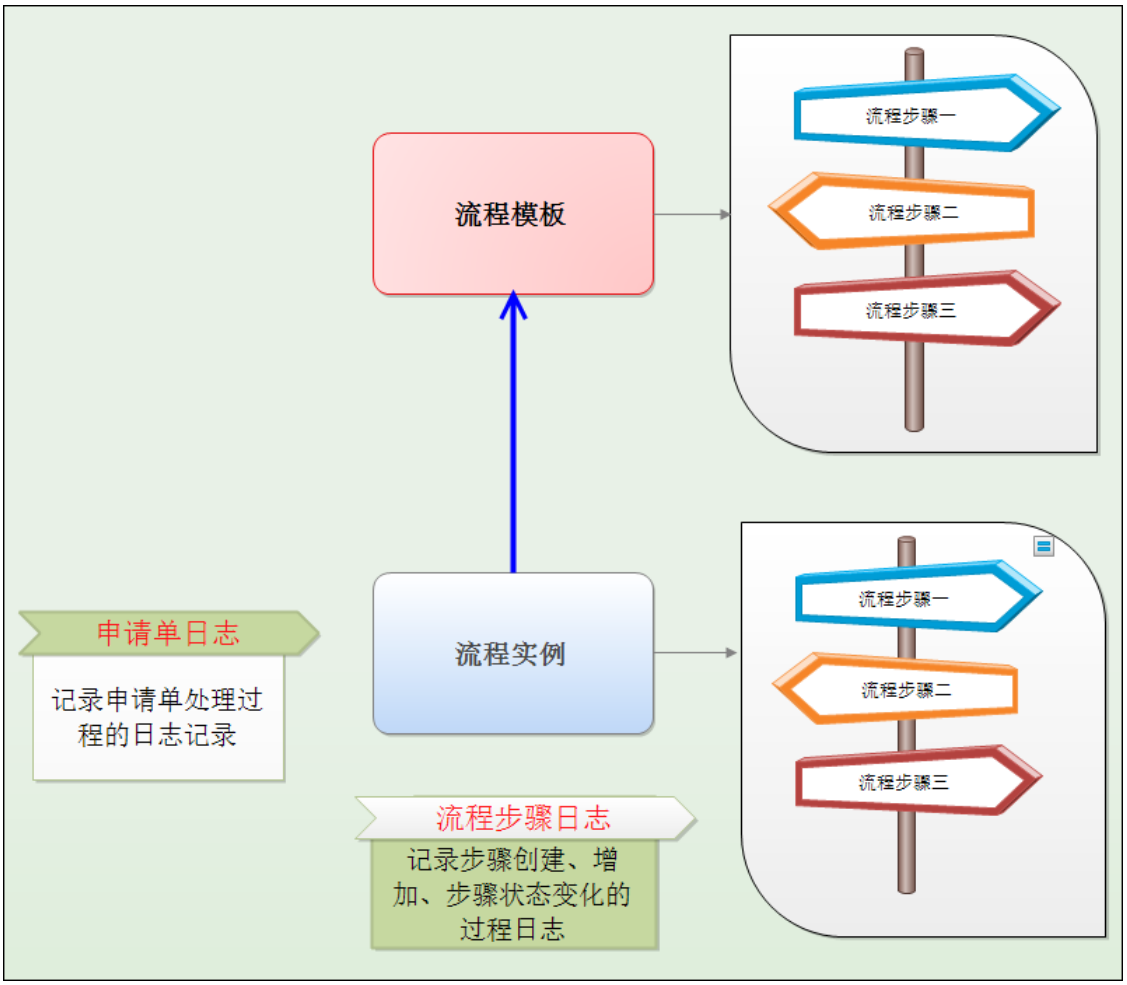
完成这些，我们就可以在实际申请单业务中进行草稿的存储和显示了。

4. 工作流模块的表设计分析

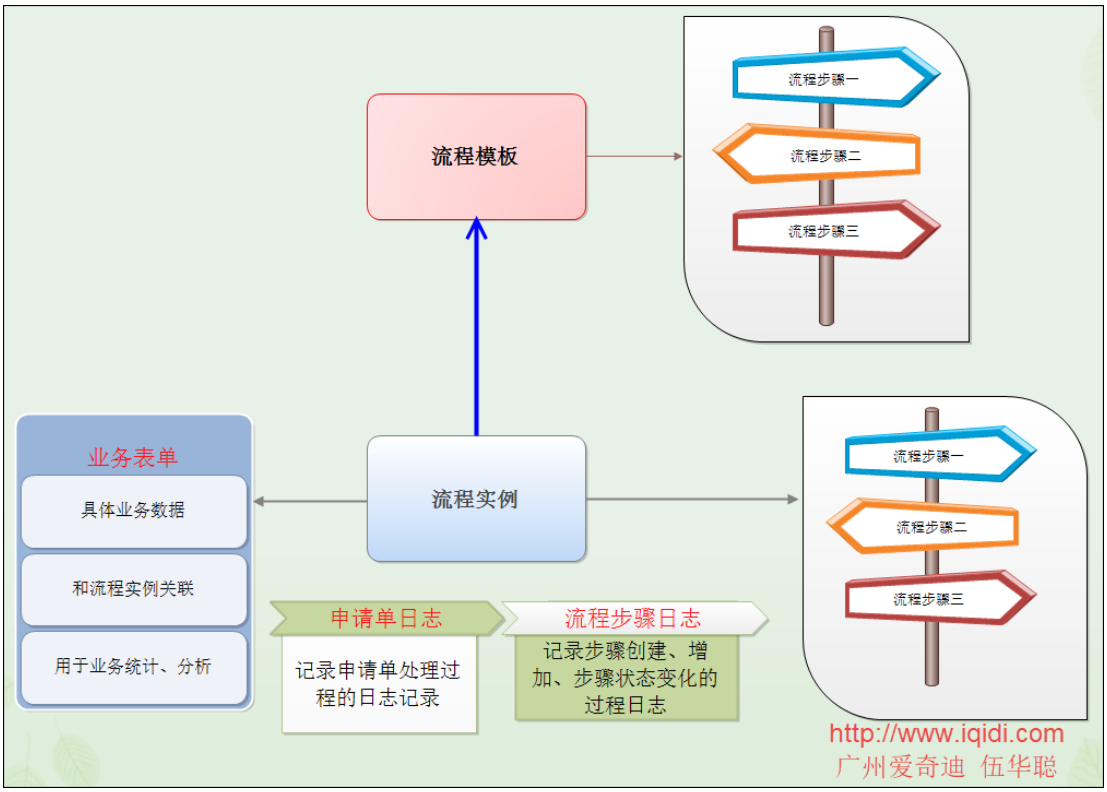
在工作流处理表中，首先我们区分流程模板和流程实例两个部分，这个其实就是类似模板和具体文档的概念，我们一份模板可以创建很多个类似的文档，文档样式结构类似的。同理，流程模板实例为流程实例后，就是具体的一个流程表单信息了，其中流程模板和流程实例表单都包括了各个流程步骤。在流程实例的层次上，我们运行的时候，需要记录一些日志方便跟踪，如流程步骤的处理日志，流程实例表单的处理日志等这些信息。



当然实际的流程实例里面需要记录很多信息，其中流程步骤日志、申请单处理日志等信息是必须要记录的，方便我们跟踪相关的处理记录。因此工作流业务表包含多两个日志记录的表，如下所示。

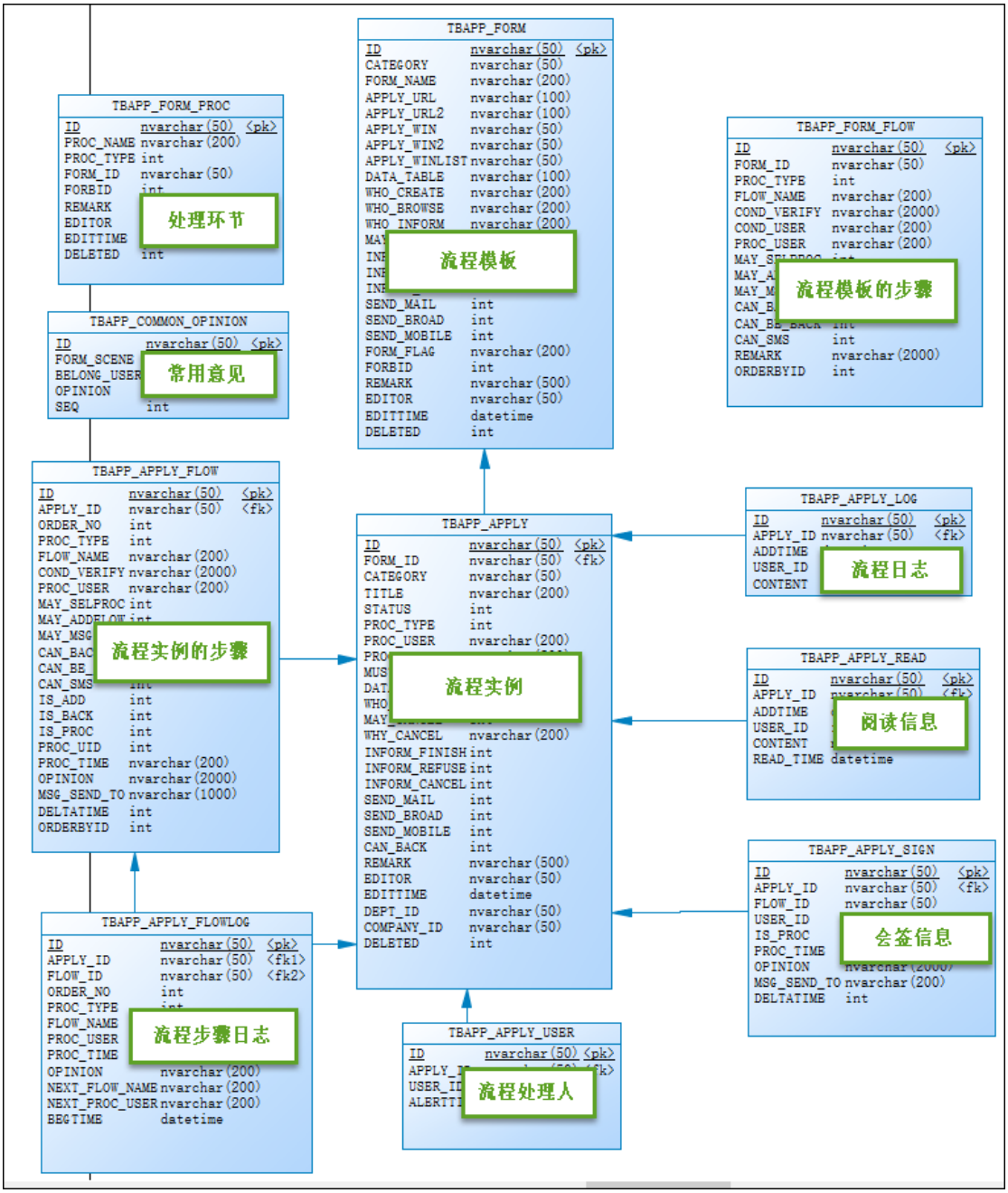


一旦流程实例根据模板创建后，流程先根据模板初始化后，在处理过程还可以动态增加一些审批步骤，使得我们的处理更加弹性化。



当然，为了更好的处理流程的相关信息，还需要记录流程处理人，流程会签人、流程阅办人，以及常用审批意见等相关辅助表，以便对流程的各个处理信息进行合理处理和展示。

整个工作流的核心基础表如下图所示。

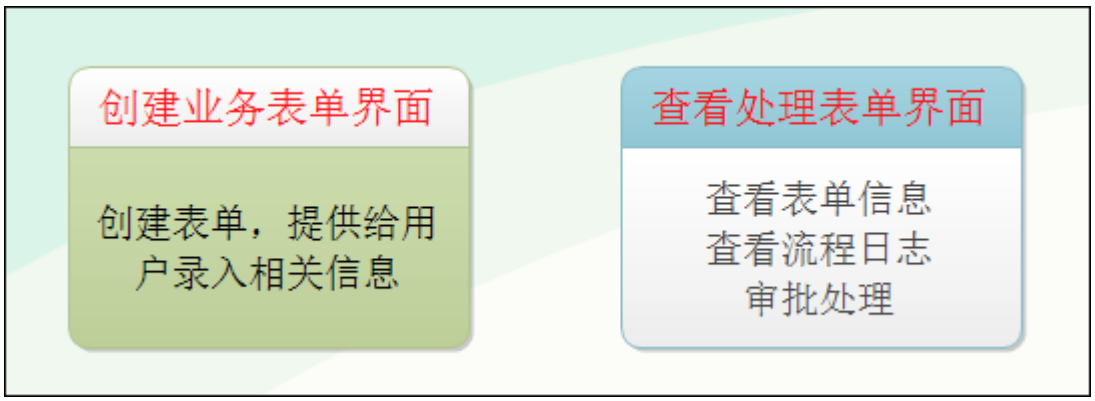


5. 工作流模块的业务表单开发

在我们开发工作流的时候，往往需要设计到具体业务表单信息的编辑，有些是采用动态编辑的，有些则是在开发过程中处理的，各有各的优点，动态编辑的则方便维护各种各样的表单，但是数据的绑定及处理则比较麻烦，而自定义开发的，则数据弹性很大，方便修改调整。本篇随笔基于表单的开发设计过程，介绍在工作流中如何新增一个业务表单，以便快速的实现审批业务的上线处理。

5.1. 业务表单的基类继承

首先我们来了解一下业务表单的对应关系，一般创建一个业务流程处理，都需要有一个具体的创建业务表单的界面，以及一个查看处理表单的界面。



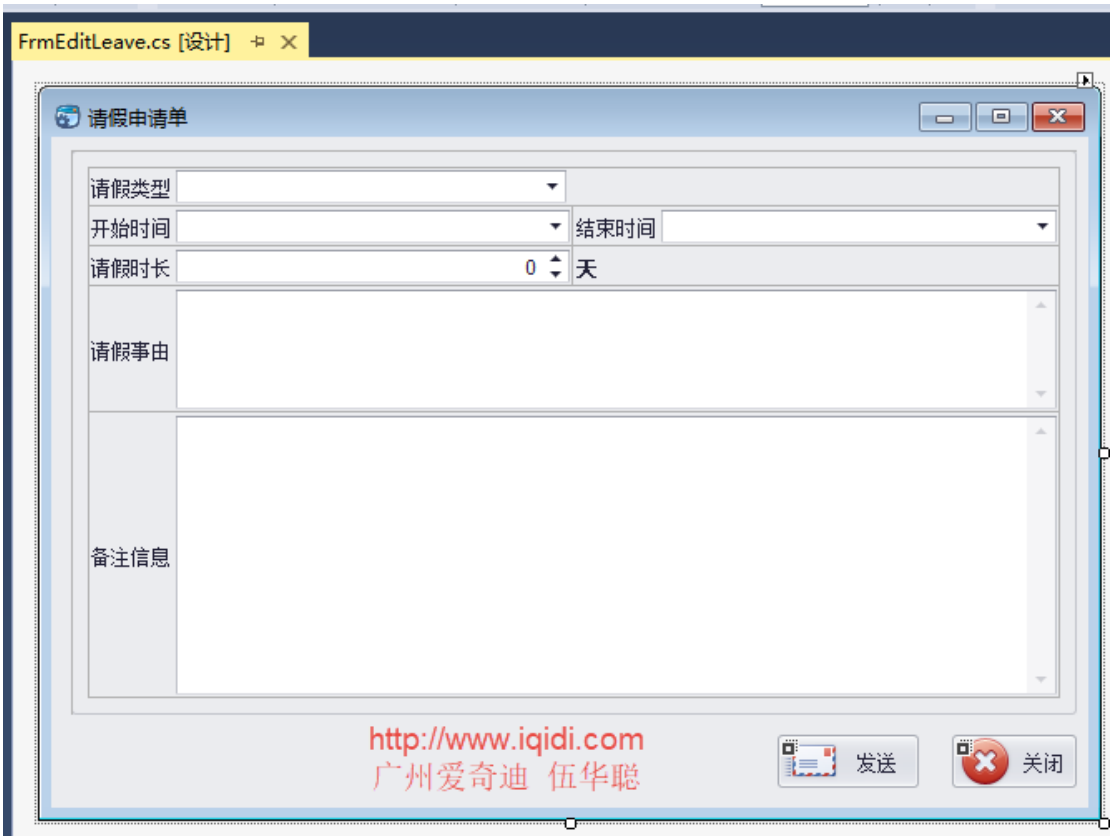
为了方便，我们尽可能减少代码编写，我们需要把大多数的逻辑处理放在基类实现，这样我们在新增一个业务表单的时候就可以减少很多代码编写及维护了。

如对于 FrmAddApply 类定义如下，我们定义一些抽象接口用于下面的业务表单实现

```
/// <summary>
/// 创建申请单的窗体基类
/// </summary>
public partial class FrmAddApply : BaseForm
{
    /// <summary>
    /// 表单ID
    /// </summary>
    public string FormID { get; set; }

    /// <summary>
    /// 申请单ID
    /// </summary>
    public string ApplyId { get; set; }

    public FrmAddApply()
    {
        InitializeComponent();
    }

    /// <summary>
    /// 显示数据的函数(子类必须实现)
    /// </summary>
    public virtual void DisplayData() { }

    /// <summary>
    /// 实现控件输入检查的函数(子类必须实现)
    /// </summary>
    /// <returns></returns>
    public virtual bool CheckInput() { return true; }

    /// <summary>
    /// 编辑状态下的数据保存(子类必须实现)
    /// </summary>
    /// <returns></returns>
    public virtual bool SaveUpdated() { return true; }

    /// <summary>
    /// 新增状态下的数据保存(子类必须实现)
    /// </summary>
    /// <returns></returns>
    public virtual bool SaveAddNew() { return true; }

    ..................
```

这样我们创建一个新的业务表单，只需要利用代码生成工具，生成所需要的各层框架代码，然后再生成 Winform 窗体代码，复制部分界面处理代码过来这个业务表单的子类即可。

下面是一个请假申请的业务表单设计，如下所示。



我们看到这个表单可以使用代码生成工具Database2Sharp快速生成后进行一定调整的，而这个编辑表单的界面，我们只需要使用自动生成的部分代码即可。

相关代码只需要复制上面的新增、更新、显示数据的代码即可。

```
/// <summary>
/// 请假申请单的编辑界面
/// </summary>
7 个引用
public partial class FrmEditLeave : FrmAddApply
{
    /// <summary>
    /// 创建一个临时对象，方便在附件管理中获取存在的GUID
    /// </summary>
    private LeaveInfo tempInfo = new LeaveInfo();

    2 个引用
    public FrmEditLeave()
    {
        InitializeComponent();
    }

    /// <summary>
    /// 实现控件输入检查的函数
    /// </summary>
    /// <returns></returns>
    4 个引用
    public override bool CheckInput()...

    /// <summary>
    /// 初始化数据字典
    /// </summary>
    1 个引用
    private void InitDictItem()...

    /// <summary>
    /// 更新界面控件数据显示
    /// </summary>
    4 个引用
    public override void DisplayData()...
```

对于查看申请单的基类 FrmViewApply 类，我们更加简单，我们需要把它的自定义界面控件加载出来即可。

下面是查看申请单的基类，封装了相关的处理逻辑。

```
/// <summary>
/// 本窗体是通用的查看申请单界面基类。
/// 为减少开发相关页面的工作量，只需要创建一个新窗体，并继承本窗体，然后在子窗体Form_Load函数里面，初始化对应的申请单显示控件即可。
/// </summary>
public partial class FrmViewApply : BaseDock
{
    /// <summary>
    /// 申请单ID
    /// </summary>
    public string ApplyId { get; set; }

    /// <summary>
    /// 申请单自定义控件
    /// </summary>
    public BaseUserControl ApplyControl { get; set; }

    /// <summary>
    /// 默认构造函数
    /// </summary>
    public FrmViewApply()
    {
        InitializeComponent();
    }

    private void FrmViewApply_Load(object sender, EventArgs e)
    {
        if (!this.DesignMode)
        {
            InitToolBar();
        }
    }

    /// <summary>
    /// 初始化申请单控件
    /// </summary>
    protected virtual void InitApplyControl(BaseUserControl control)
    {
        if (control != null)
        {
            this.ApplyControl = control;
            this.ApplyControl.Dock = DockStyle.Fill;
            this.Controls.Add(control);
        }
```

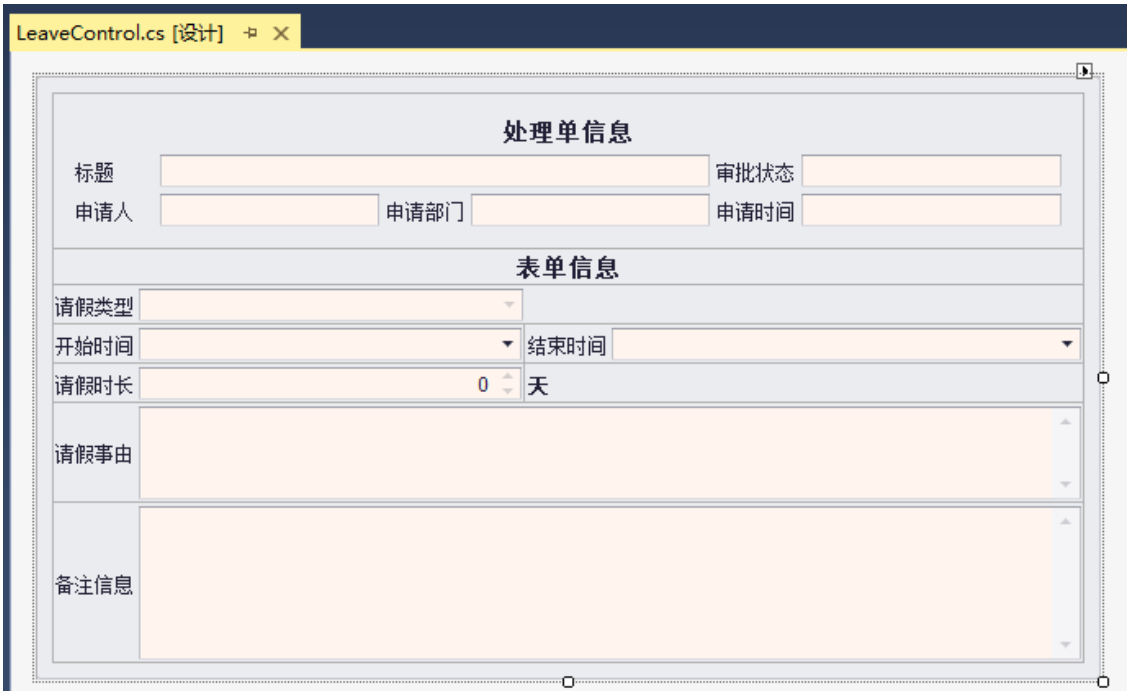
```
        /// <summary>
        /// 打印申请单控件内容(默认调用窗体打印)
        /// </summary>
        protected virtual void PrintApplyControl()
        {
            if(this.ApplyControl != null)
            {
                PrintFormHelper.Print(this.ApplyControl, false);
            }
        }

        /// <summary>
        /// 表单另存为
        /// </summary>
        protected virtual void ApplySaveAs()
        {
        }

        /// <summary>
        /// 初始化工具栏的按钮和状态
        /// </summary>
        protected virtual void InitToolBar()
        {
               ................//基类实现，控制什么时候该做什么审批处理，以及一些常见按钮
        }

    ................
```

查看请假申请单的窗口就是继承这个 FrmViewApply 即可，如下所示。

```
/// <summary>
/// 查看请假申请单的窗体
/// </summary>
public partial class FrmViewLeave : FrmViewApply
{
    private LeaveControl control = null;

    public FrmViewLeave()
    {
        InitializeComponent();
    }

    private void FrmViewLeave_Load(object sender, EventArgs e)
    {
        //初始化控件并展示在基类窗体里面
        control = new LeaveControl();
        control.ApplyId = this.ApplyId;
        control.DisplayData();

        base.InitApplyControl(control);
    }
}
```

这个就是全部的窗体源码了，主要的内容我们看到是在 LeaveControl 这个用户控件类里面的了，

而这个控件主要就是上面编辑请假申请单的界面设计，并复制相关的显示数据代码即可。

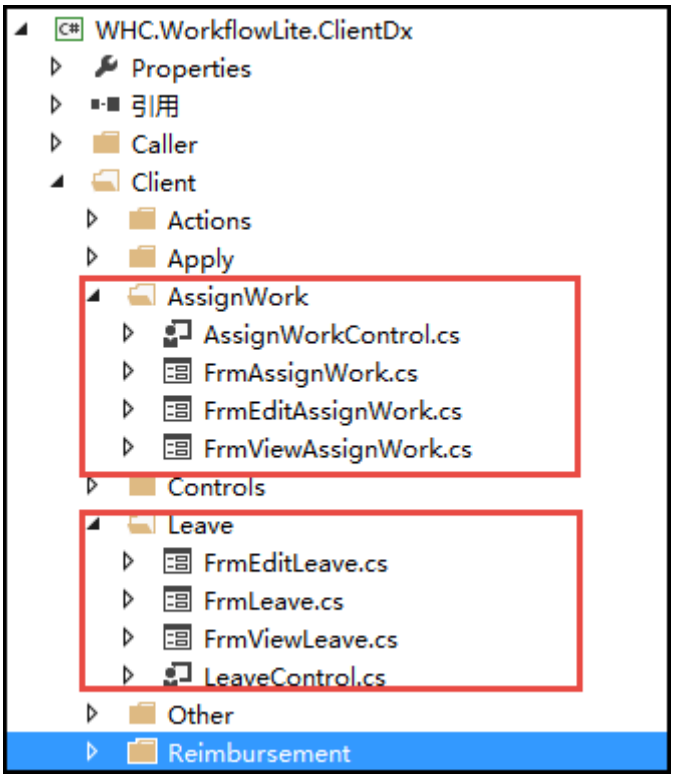

LeaveControl.cs [设计]
处理单信息
标题
审批状态
申请人
申请部门
申请时间
表单信息
请假类型
开始时间
结束时间
请假时长
0
天
请假事由
备注信息

相关界面代码如下所示。

```
/// <summary>
/// 查看请假申请单的内容显示控件
/// </summary>
public partial class LeaveControl : BaseUserControl
{
    /// <summary>
    /// 申请单ID
    /// </summary>
    public string ApplyId { get; set; }

    public LeaveControl()
    {
        InitializeComponent();

        SetReadOnly();
    }

    /// <summary>
    /// 设置整个窗体布局为只读并设置只读的背景颜色
    /// </summary>
    private void SetReadOnly()
    {
        this.layoutControl1.OptionsView.IsReadOnly = DevExpress.Utils.DefaultBoolean.True;
        this.layoutControl1.Appearance.ControlReadOnly.BackColor = Color.SeaShell;
    }

    private void LeaveControl_Load(object sender, EventArgs e)
    {
        this.applyInfoControl1.ApplyId = this.ApplyId;
        this.applyInfoControl1.BindData();
    }

    /// <summary>
    /// 初始化数据字典
    /// </summary>
    private void InitDictItem()
    {
        //初始化代码
    }

    /// <summary>
    /// 数据显示的函数
    /// </summary>
    public void DisplayData()
    {
        InitDictItem();//数据字典加载（公用）
```

通过上面定义的对应表单的窗体基类，可以减少我们重复编码的需要，我们只需要利用最有效率的生成界面，然后复制代码后调整即可快速生成我们所需要的不同表单界面。

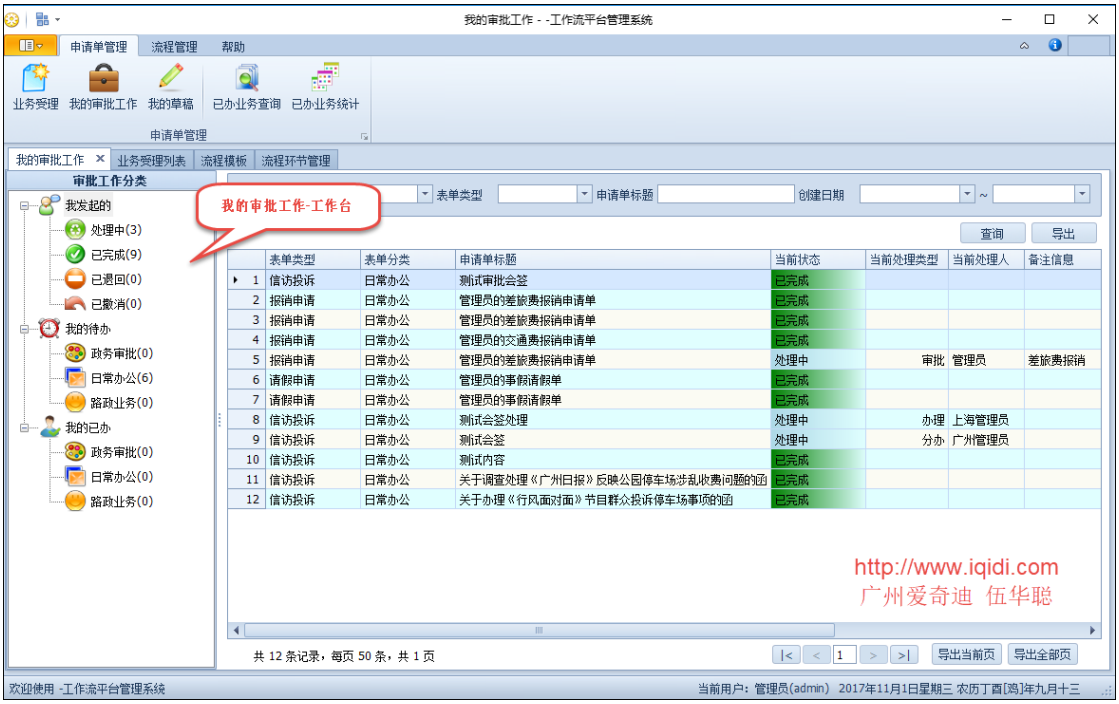
每个表单我们放在一个目录上，这样我们就可以很好管理它们了。



5.2. 业务表单的动态展示处理

上面介绍了业务表单的填写、查看两个不同的窗口，我们在申请单的审批界面里面，统一显示不同的表单，以及创建不同的业务表单界面，这种动态的处理可以实现不同业务表单的创建及显示界面。

如我的审批工作中，表单的显示界面如下所示，查看具体表单后，可以动态展示不同的业务窗口界面。



另外我们在创建业务表单的时候，根据数据库的配置信息，动态展示所有可以展示的创建入口，单击相关的按钮，可以动态调用创建对应的表单界面。

创建流程业务表单的入口如下所示。

在我的审批工作界面，动态创建对应的查看表单窗体代码如下所示。

```
/// <summary>
/// 分页控件编辑项操作
/// </summary>
private void winGridViewPager1_OnEditSelected(object sender, EventArgs e)
{
    //获取记录ID和表单ID
    string ID = this.winGridViewPager1.gridView1.GetFocusedRowCellDisplayText("ID");
    string FormId = string.Concat(this.winGridViewPager1.gridView1.GetFocusedRowCellValue("FormId"));

    if (!string.IsNullOrEmpty(ID) && !string.IsNullOrEmpty(FormId))
    {
        var formInfo = BLLFactory<BLL.Form>.Instance.FindByID(FormId);
        if (formInfo != null && !string.IsNullOrEmpty(formInfo.ApplyWin2))
        {
            try
            {
                //根据配置的查看窗体，动态构建查看申请单对象
                FrmViewApply dlg = Assembly.GetExecutingAssembly().CreateInstance(formInfo.ApplyWin2) as FrmViewApply;
                if (dlg != null)
                {
                    dlg.ApplyId = ID;
                    dlg.OnDataSaved += new EventHandler(dlg_OnDataSaved);

                    if (DialogResult.OK == dlg.ShowDialog())
                    {
                        BindData();
                    }
                }
            }
            catch (Exception ex)
            {
                LogHelper.Error(ex);
                MessageDxUtil.ShowError(ex.Message);
            }
        }
    }
}
```

这个代码替代了需要手动创建不同对象的处理。

```
var dlg = new FrmViewAssignWork();
dlg.ApplyId = ID;
dlg.OnDataSaved += new EventHandler(dlg_OnDataSaved);

if (DialogResult.OK == dlg.ShowDialog())
{
    BindData();
}
```

同理，对于创建编辑界面，我们也可以同样的方法动态创建相关的编辑表单界面，如下代码所示。

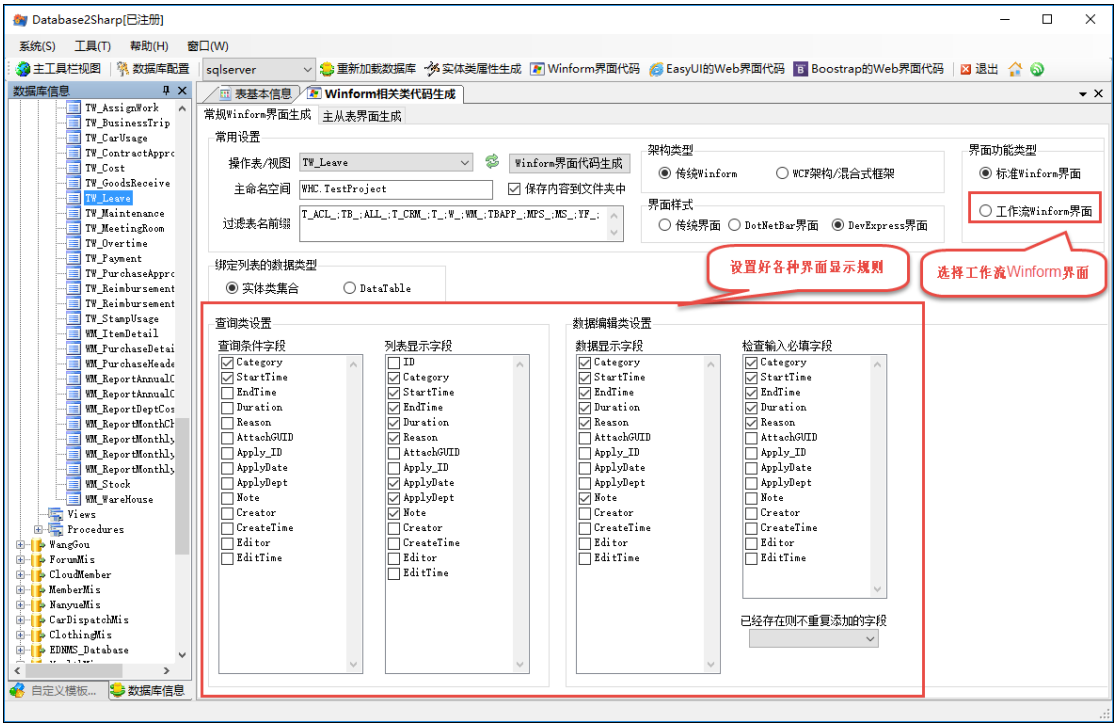
```
/// <summary>
/// 单击某个动态生成的按钮，触发的申请表单创建界面
/// </summary>
1 个引用
void button_Click(object sender, EventArgs e)
{
    SimpleButton button = sender as SimpleButton;
    if (button != null)
    {
        var formId = button.Tag.ToString();
        var formInfo = BLLFactory<BLL.Form>.Instance.FindByID(formId);
        if (formInfo != null && !string.IsNullOrEmpty(formInfo.ApplyWin))
        {
            try
            {
                //动态构建创建申请单的界面窗体并赋值
                var dlg = Assembly.GetExecutingAssembly().CreateInstance(formInfo.ApplyWin) as FrmAddApply;
                dlg.FormID = button.Tag.ToString();
                dlg.ShowDialog();
            }
            catch(Exception ex)
            {
                LogHelper.Error(ex);
                MessageDxUtil.ShowError(ex.Message);
            }
        }
        else
        {
            MessageDxUtil.ShowTips(button.Text + "暂未开通");
        }
```


5.3. 工作流业务界面的代码生成

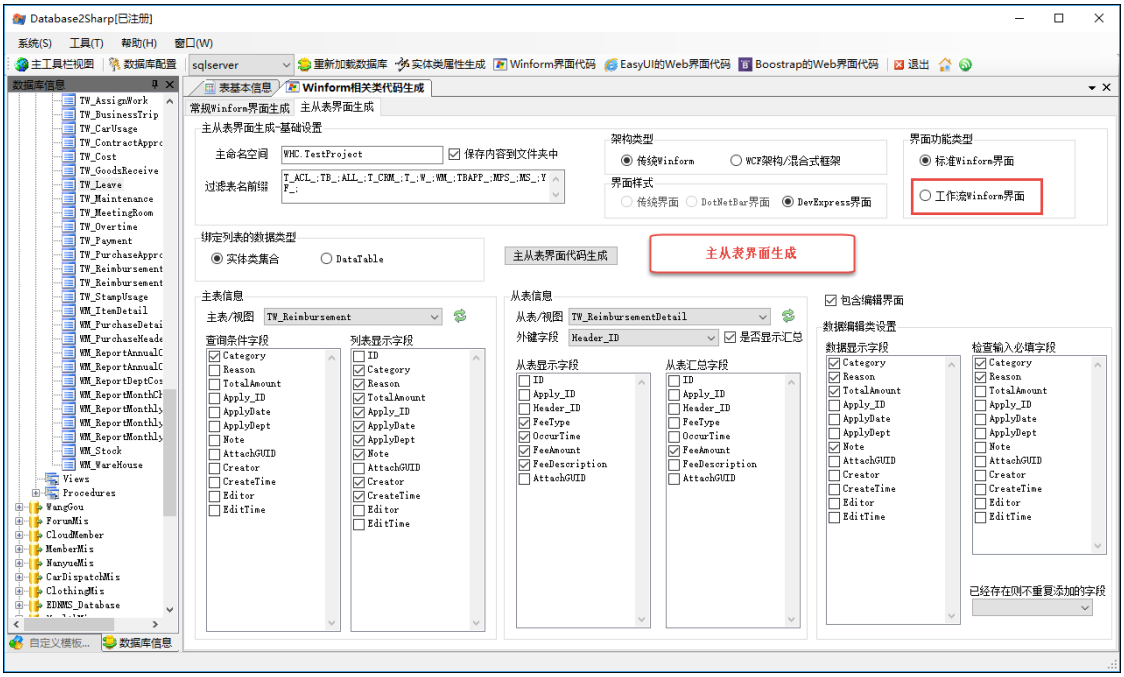
从上面我们可以看到，其中对于工作流业务表单的窗体界面都可以实现标准的处理了，继承自某个基类，然后整合相关的数据处理规则即可。

那么我们提炼业务信息后，可以使用代码生成工具快速生成，这样可以极大提高我们的开发效率。

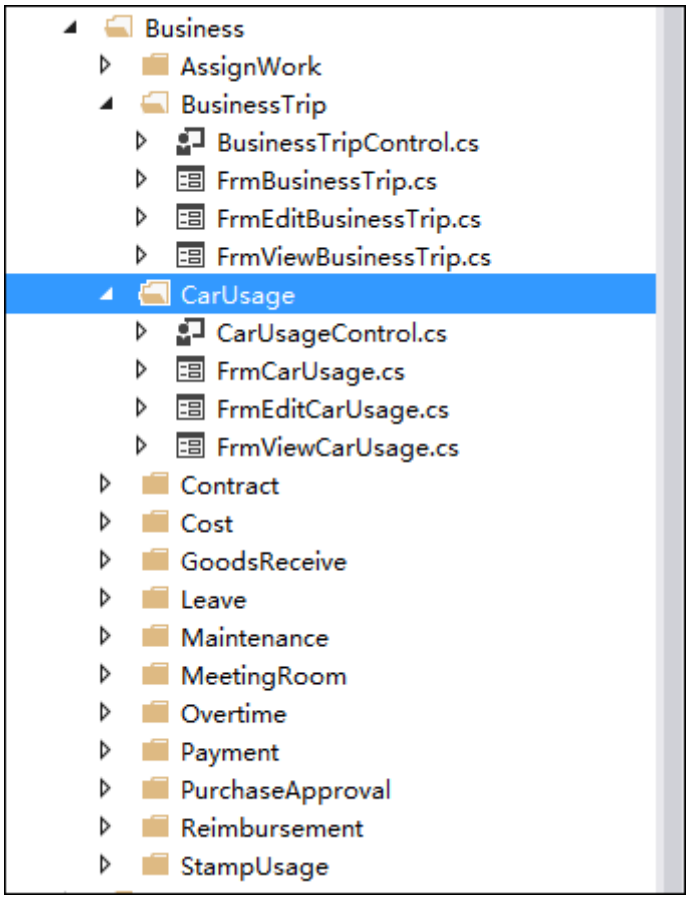
针对上面的构想，我们花费了好几天的时间，创建了工作流界面的自动生成规则和反复校验，最终整合到代码生成工具中方便开发。



对于主从表表的界面，我们依旧也可以使用代码生成工具进行快速的工作流界面生成。



根据代码生成工具生成相应的业务表单，在项目中展示如下所示，其中每个业务表单包括查看控件、申请单查询界面、新增申请单界面、查看申请单界面。


Business
AssignWork
BusinessTrip
BusinessTripControl.cs
FrmBusinessTrip.cs
FrmEditBusinessTrip.cs
FrmViewBusinessTrip.cs
CarUsage
CarUsageControl.cs
FrmCarUsage.cs
FrmEditCarUsage.cs
FrmViewCarUsage.cs
Contract
Cost
GoodsReceive
Leave
Maintenance
MeetingRoom
Overtime
Payment
PurchaseApproval
Reimbursement
StampUsage