工作流开发手册

# **说明**

文档旨在说明中职工作流接口以及控件的使用方式。主要面向高铁一号线的开发人员，作为开发人员流程开发的参考。本文档仅包括工作流的API接口和控件使用，不包括流程模板的设计，关于流程模板设计，请参考其它文档。

# **前置任务**

在使用工作流接口之前，需首先配置工作流的设计器、接口服务等，具体配置方法请参考$/CIICSHGT1/Sources/Main/TSP/TSP.WF/Document/工作流开发部署说明.docx，在部署好相关的服务后，控件和接口才能正常使用。

# **流程控件列表**

流程控件主要有下面几种，每种控件对应一种业务行为，如StartWorkflow方式启动一个工作流。

|  |  |  |
| --- | --- | --- |
| 名称 | 类名 | 说明 |
|  |  |  |
| 启动 | WFStartWorkflow | 根据指定的模板启动一个工作流。 |
| 流转 | WFMoveTo | 让流程继续向前流转，绝大部分的业务场景，按钮名一般是“同意”，“拒绝”。 |
| 撤回 | WFWithdraw | 把流程撤回一步，在流程已经向前走2个（包含2个）以上时，不允许撤回。 |
| 作废 | WFCancel | 废弃一个流程，流程立即终止，可以使用WFRestore，将流程恢复。 |
| 恢复 | WFRestore | 将某一个废弃的流程恢复，恢复的流程处于废弃时的状态。 |
| 暂停 | WFPause | 暂停流程，流程处于等待状态。 |
| 重启 | WFResume | 使暂停的流程继续运行。 |
| 保存 | WFSave | 保存流程状态。 |
| 跟踪 | WFTrack | 跟踪流程 |
| 意见文本框 | WFComments | 输入用户审核意见 |
| 意见列表 | WFOpinionGridList | 显示用户审核意见 |
| 流程导航 | WFGraph | 显示流程的主线流程 |
| 刷新流程参数 | WFUpdateProcess | 流程实例信息变更以后，刷新流程实例的相关信息 |
| 启动按钮横向显示 | ToolbarStart | 工作流启动按钮非下拉显示 |
| 流转按钮横向显示 | ToolbarMoveTo | 工作流流转按钮非下拉显示 |
|  |  |  |

备注：所以控件的Enable状态都会根据当前流程状态和当前登录人的权限自动设置

# 流程控件使用

## 流程服务配置

流程接口以WCF服务提供，需要配置服务的相关地址以及解析配置等信息。请参考如下配置信息，修改ProcessDescriptorService和ProcessRuntimeService为实际地址。

<configuration>

<configSections>

<section name="jsonConverterRegisterSettings" type="MCS.Web.Library.Script.Configuration.JsonConverterRegisterSettings, MCS.Web.Library.Script.Json" />

<section name="wfContractsProxySettings" type="MCS.Library.WF.Contracts.Proxies.Configuration.WfContractsProxySettings, MCS.Library.WF.Contracts.Proxies" />

</configSections>

<wfContractsProxySettings>

<paths>

<add name="processDescriptorService" uri="http://localhost/MCSWebApp/WfOperationServices/Services/ProcessDescriptorService.svc" />

<add name="processRuntimeService" uri="http://localhost/MCSWebApp/WfOperationServices/Services/ProcessRuntimeService.svc" />

</paths>

</wfContractsProxySettings>

<jsonConverterRegisterSettings>

<registers>

<add name="wfClientConverters" type="MCS.Library.WF.Contracts.Json.Converters.WfClientJsonConverterHelper, MCS.Library.WF.Contracts.Json" />

</registers>

</jsonConverterRegisterSettings>

</configuration>

## 注册用户上下文

流程在运行过程中，一直需要使用某个用户身份作为操作用户，如流程启动人、任务创建人等。工作流API提供了IWFUserContext接口，用于与业务系统的上下文用户交互。此流程API提供了一个默认的接口实现，此默认实现试图从HttpContext.Current.GetCurrentUserInfo()中获取用户信息。如果业务系统不是使用此方式获取用户信息，则需要自定义个继承自IWFUserContext的一个实现。

默认注册方式：

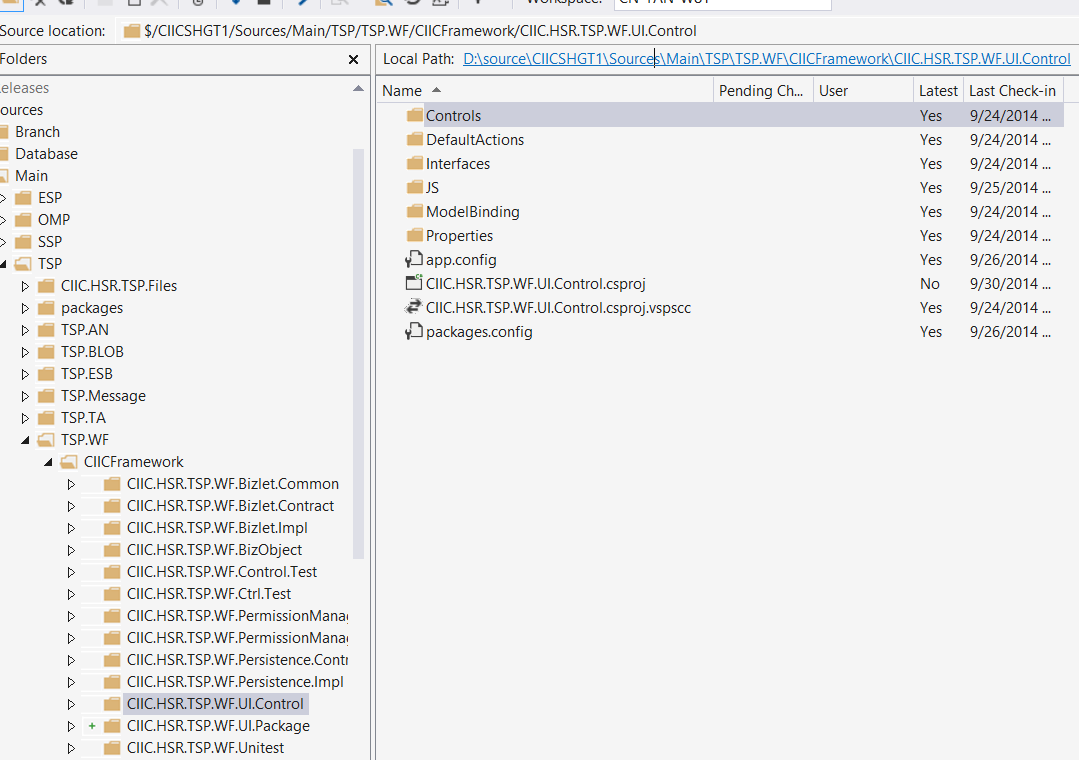
sl.Register<IWFUserContext, DefaultUserContext>(alwaysNew: true);

自定义实现方式：

sl.Register<IWFUserContext, 你的自定义类>(alwaysNew: true);

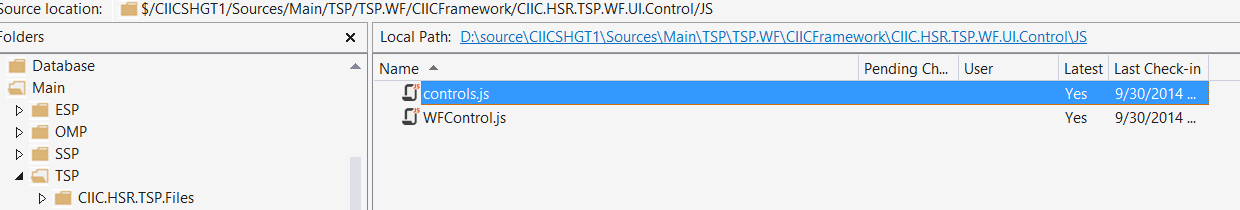
## 引用控件类库

控件源代码位置 $/CIICSHGT1/Sources/Main/TSP/TSP.WF/CIICFramework/CIIC.HSR.TSP.WF.UI.Control/ CIIC.HSR.TSP.WF.UI.Control.csproj，工作流控件一类与基础控件，因此需要首先引用CIIC.HSR.TSP.WebComponents。具体位置如下图所示：



## 引用JS脚本--**（控件会自动引入）**

工作流控件是基于基础控件Button开发，因此除了使用工作流控件的脚本之外，还需要引用基础控件的JS库，从如下位置获取并引用脚本，controls和WfControls具需要引用。



## 提交控件

提交控件的任务是根据指定流程模板启动一个工作流，一般在业务上按钮名称为“提交”，当然具体的名称由业务而定。提交控件本质上仍然是一个按钮，因此除了流程相关的属性，它拥有Button的所有属性，如ButtonType、ButtongStyle等，工作流所有的控件都具有此特点，在其它控件说明中不再赘述。

### 书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性，如TemplateKey（流程模板名称）。

@(

Html.HSRUIWF().WFStartWorkflow().ActionUrl(Url.Action("StartWorkflow","home"))

.AfterClick("afterClick").BeforeClick("beforeClick").DialogText("你确定要提交吗？")

.Name("ExprenSubmit").TemplateKey("CtrlTest01").Text("提交报销").ProgressBar(false)

.BusinessUrl(Url.Action("StartWorkflow","home")).TaskTitle("测试报销")

)

### 属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 属性 | 说明 |
|  |  |  |
| Action地址 | ActionUrl | 指定业务请求的地址，即Action地址。此参数必须提供。 |
| 调用前事件 | BeforeClick | 在向服务器递交数据前触发的客户端事件，此事件形如function beforeClick(sender, progressbar)，此方法要求返回表单数据，以便将数据提交给Action，可以调用Jquery的serialize()搜集数据。 |
| 调用后事件 | AfterClick | 请求返回后调用的客户端事件，此事件形如function afterClick(result, sender, progressbar)，result携带了从服务端返回的数据，result.BusinessData携带业务数据，具体数据结构由业务决定，result.ProcessInfo携带有当前流程详细信息。 |
| 流程模板 | TemplateKey | 待启动的流程模板名称，此参数必须提供。 |
| 表单地址 | BusinessUrl | 业务表单地址，从待办列表进入业务页面时，需要使用此参数。此参数必须提供。 |
| 任务标题 | TaskTitle | 待办任务的标题，此参数必须提供。 |
| 动态角色审批人列表 | DictionaryWfClientUser | 添加动态角色审批人，需要选择审批人的情况下使用 |
|  |  |  |

### Action创建

Action创建于普通的Action类似，仅仅多了一个参数，具体参数类型由操作的不同而不同，提交控件Action形如下

[HttpPost]

public ActionResult StartWorkflow(WFStartWorkflowParameter paras, Expense expense)

{

//设置当前操作用户，如果不使用登陆用户，则可以使用如下方式修改

paras.ProcessStartupParams.Creator.ID = "4EF6BE7E-9300-416D-B390- ";

paras.ProcessStartupParams.Creator.Name = "曹节";

//待办标题

paras.TaskTitle = "测试流程启动";

paras.BusinessUrl = Url.Action("MoveTo","home");

//加入流程参数

paras.ProcessStartupParams.ApplicationRuntimeParameters["amount"] = expense.Amount;

//处理业务

expense.Amount = expense.Amount + 10;

//执行流程启动操作

ResponseData data= paras.Execute();

//处理客户端返回数据

data.BusinessData = expense;//业务数据

return Json(data);

}

上面的代码以paras.Excute()为分水岭，被分为流程启动前和启动后，可以根据业务具体的需要加入更多需要的业务处理。

这里的expense不是工作流的一部分，仅仅是测试时的一个模拟报销数据的Model，在实际开发中，需要根据自己的需要修改。

Data.BusinessData不是必须的，如果客户端不需要返回数据，则可以忽略此数据设置。

## 流转按钮

### 书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFMoveTo().ActionUrl(Url.Action("MoveTo","home")).AfterClick("afterClick").BeforeClick("beforeClick").DialogText("您确定要流转吗？").Name("ExpenseApprove")

)

### 属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 属性 | 说明 |
| 调用前事件 | BeforeClick | 在向服务器递交数据前触发的客户端事件，此事件形如function beforeClick(sender, progressbar)，此方法要求返回表单数据，以便将数据提交给Action，可以调用Jquery的serialize()搜集数据。 |
| 调用后事件 | AfterClick | 请求返回后调用的客户端事件，此事件形如function afterClick(result, sender, progressbar)，result携带了从服务端返回的数据，result.BusinessData携带业务数据，具体数据结构由业务决定，result.ProcessInfo携带有当前流程详细信息。 |
| 动态角色审批人列表 | DictionaryWfClientUser | 需要选择流程审批人的情况使用 |
|  |  |  |

### Action创建

[HttpPost]

public ActionResult MoveTo(WFMoveToParameter paras, Expense expense)

{

//加入流程参数

paras.SetParameter("amount", "100");

//这里可以做流程操作执行前处理

//执行流程处理

ResponseData data = paras.Execute();

//这里可以做流程操作执行后处理

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

**备注：**

1.按钮名称：流转控件可设置Text属性，默认为按钮名称且对下拉选项按钮名称没作用。若Text属性没设置，则必须将流程设计时将线的名称设置上且‘送签按钮名称和线的名称一样’属性设置成True，控件将会把线的名称设置到按钮上。



## 4.7作废按钮

### 4.7.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFCancelWorkflow().ActionUrl(Url.Action("CancelWorkflow","home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("作废").DialogText("您确定要作废吗？").Name("btnCancelWf").ProgressBar(false)

)

### 4.7.2属性说明

同4.6.2

### 4.7.3 Action创建

[HttpPost]

public ActionResult CancelWorkflow(WFCancelParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 撤回按钮

### 4.8.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFWithdrawWorkflow().ActionUrl(Url.Action("WithdrawWorkflow", "home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("撤回") .DialogText("您确定要撤回吗？").Name("btnWithdrawWf").ProgressBar(false)

)

### 4.8.2属性说明

同4.6.2

### 4.8.3 Action创建

[HttpPost]

public ActionResult WithdrawWorkflow(WFWithdrawParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 恢复按钮

### 4.9.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFRestoreWorkflow().ActionUrl(Url.Action("RestoreWorkflow", "home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("恢复").DialogText("您确定要恢复吗？").Name("btnRestoreWf").ProgressBar(false)

)

### 4.9.2属性说明

同4.6.2

### 4.9.3 Action创建

[HttpPost]

public ActionResult RestoreWorkflow(WFRestoreParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 暂停按钮

### 4.10.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFPauseWorkflow().ActionUrl(Url.Action("PauseWorkflow","home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("暂停").DialogText("您确定要暂停吗？").Name("btnPauseWf").ProgressBar(false)

)

### 4.10.2属性说明

同4.6.2

### 4.10.3 Action创建

[HttpPost]

public ActionResult PauseWorkflow(WFPauseParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 重启按钮

### 4.11.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFResumeWorkflow().ActionUrl(Url.Action("ResumeWorkflow","home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("重启").DialogText("您确定要重启吗？").Name("btnResumeWf").ProgressBar(false)

)

### 4.11.2属性说明

同4.6.2

### 4.11.3 Action创建

[HttpPost]

public ActionResult ResumeWorkflow(WFResumeParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 保存按钮

### 4.12.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFSaveWorkflow().ActionUrl(Url.Action("SaveWorkflow","home")).AfterClick("afterClick").BeforeClick("beforeClick").Text("保存").DialogText("您确定要保存吗？").Name("btnSaveWf").ProgressBar(false)

)

### 4.12.2属性说明

同4.6.2

### Action创建

[HttpPost]

public ActionResult SaveWorkflow(WFSaveParameter paras, Expense expense)

{

ResponseData data = paras.Execute();

//设置返回客户端的业务数据，如果不需要，可以不设置

data.BusinessData = expense;

return Json(data);

}

## 跟踪按钮

### 4.13.1书写方式

书写方式与Button等控件使用类似，支持连写，同时设置控件的相关属性

@(

Html.HSRUIWF().WFTrackWorkflow().Text("跟踪").Name("btnTrackWf").ProgressBar(false)

)

### 4.13.2属性说明

跟踪按钮其实就是一个连接，连接到当前Web.config配置好的下的一个地址，格式如下

<configSections>

<section name="urlSetting" type="CIIC.HSR.TSP.WF.Bizlet.Impl.UrlSection,CIIC.HSR.TSP.WF.Bizlet.Impl"/>

</configSections>

<urlSetting>

<urls>

<!--流程跟踪页面Url配置->

<url name="wfRuntimeViewerlinkPageUrl" description="流程跟踪页面" url="http://localhost/MCSWebApp/OACommonPages/AppTrace/appTraceViewer.aspx" default="" ></url>

</urls>

</urlSetting>

## 意见输入文本框

### 4.14.1书写方式

书写方式与Text控件使用类似，支持连写，同时设置控件的相关属性

@(  
         Html.HSRUIWF().WFComments().Name("wfCommentsTest").Collumns(5).Lines(3).OpinionContainerId("OpinionContainer")  
         )

### 4.14.2属性说明

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Enabled | bool | 是否可以输入，默认是True |
| Name | string | 控件名称，默认是wfComments |
| Collumns | int | 文本框列数，根据实际页面自行设置 |
| Lines | int | 文本框行数，根据实际页面自行设置，默认是1行 |
| OpinionContainerId | string | 意见文本框label的id 或者为 意见文本框外围容器DIV的id。根据页面实际布局情况，自行设置。设置后且非InMovetoModel状态下会隐藏该id对应的控件 |

备注：若不采用默认的Name ，自己重新定义了，则必须设置（启动，流转，取消，保存）控件的CommentsControlId属性与其一致。  
如下：  
      @(  
                    Html.HSRUIWF().WFMoveTo().ActionUrl(Url.Action("MoveTo","home"))  
                    .AfterClick("afterClick").BeforeClick("beforeClick")  
                     .DialogText("您确定要流转吗？").Name("ExpenseApprove").CommentsControlId("wfCommentsTest")  
        )

## 意见列表控件

### 4.15.1书写方式

@(

Html.HSRUIWF().WFOpinionGridList().Name("wfOpinionListTest").PageSize(5).Height(250).Title("流程审批意见")

)

### 4.15.2属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 类型 | 说明 |
| Name | string | 控件名称，默认是wfOpinionList |
| PageSize | int | 当前列表显示的件数，默认是5 |
| Height | int | 列表的高度 |
| Title | string | 列表的Title,默认值是流程意见列表 |

意见输入和意见输入列表结合使用场景以及注意事项  
  
流转时到某人时，此人可以输入意见。这个人当前环节的意见不可出现在列表中

如果一个环节涉及到两个人，那么当前人只能编辑它自己的意见，另一个人如果写了意见，可以在列表中看到。

在某些业务场景中，流程提交人也可以输入意见。其意见会出现在后续环节的意见列表中。  
  
在表单上的Controller上，不要忘记WFAclAuthorize属性。禁止流程以外的人员（除管理员）通过Url复制打开表单。  
  
        [WFAclAuthorize]  
        public ActionResult MoveTo()  
        {  
            Expense expense = new Expense() { Amount = 101, Department = "IT", Name = "测试名称", TransitionDate = DateTime.Now };  
            return View(expense);  
        }

## 流程导航控件

### 书写方式

* 初始化

@Html.HSRUIWF().WFGraph().Name("wfgraph")

* 流程启动页面刷新

JS事件中调用：

$.fn.HSR.Controls.WFGraph("#wfgraph").Reload(result.ProcessInfo.ID);

* 流程申请页面刷新

提供两种方式刷新：

1. JS事件中调用：

$.fn.HSR.Controls.WFGraph("#wfgraph").Reload();

1. 结合刷新流程参数使用，参照刷新流程参数的使用说明

### 属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 类型 | 说明 |
| Name | string | 控件名称，默认是wfgraph |
| EnableDefaultUserName | bool | 激活用户为空时的默认显示（默认为True） |
| DefaultUserName | string | 用户为空时，显示的信息(默认我为"流转中指定") |

## 刷新流程参数控件

### 书写方式

* HTML页面初始化：

@( Html.HSRUIWF().WFUpdateProcess().Name("WFUpdateProcess").BeforeClick("beforeClick").ActionUrl(Url.Action("UpdateProcess", "Home")).AddUpdateElement("toolbarContainer", "WFToolbar").AddUpdateElement("wfGraphWrap", "WFGraph")

)

* 执行流程参数刷新：

$("#WFUpdateProcess").click();

### 说明

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| ActionUrl | 用于处理流程的刷新逻辑（必须客户自己实现） |
| AddUpdateElement | 其中AddUpdateElement方法用于设定需要刷新的区域容器（必须设定，至少一个容器），key为区域容器的ID，Value为区域中的PartialView 名称。区域容器中除了PartialView 不应该包含其它的HTML元素。比如：  <div class="row">  <div class="col-lg-12" id="wfGraphWrap">  @Html.Partial("WFGraph")  </div>  </div> |

ActionUrl 对应的Action处理逻辑

public ActionResult UpdateProcess(WFUpdateProcessParameter param, Expense expense)

{

try

{

param.RuntimeContext.ApplicationRuntimeParameters["Amount"] = expense.Amount;

param.RuntimeContext.AutoCalculate = true;

SaveExpense(param.RuntimeContext, expense);

param.SetViewModel(expense);//如果PartialView使用的Model为主视图的View，则通过该方法设置

//param.SetPartialViewModel("WFGraph", expense.ProcessId);//如果PartialView需要Model，则通过该方法设置 ，可以与 SetViewModel方法共存

ResponseData data = param.Execute();//执行调用

//通过UpdateElementsHtml返回HTML信息，另外必须返回JsonSuccess格式的ActionResult

return this.JsonSuccess("True", param.UpdateElementsHtml);

}

catch (System.Exception ex)

{

return this.JsonError(ex.Message, ex);

}

}

因为有些PartView可能有写动态的值，所以需要为PartView设置Model,目前通过以下方法:

* SetViewModel 设置默认的Model，一般与主视图的Model一致。类似于页面中的 @Html.Partial("WFGraph"，Model)
* SetPartialViewModel 设置PartialView的Model，一般在与主视图的Model不一致的时候使用

类似于页面中的 @Html.Partial("WFGraph"，Model.Id)

## 流程启动按钮横向显示

### 书写方式

@(

Html.HSRUIWF().WFToolbarStartWorkflow()

.Direction(d => d.Default())

.SizeMode(p => p.Default())

.AfterClick("afterClick")

.BeforeClick("beforeClick")

.DialogText("你确定要提交吗？")

.ActionUrl(Url.Action("StartWorkflow", "home"))

.Name("ExprenSubmit2")

.TemplateKey("ProcessDemo")

.Text("提交报销").ProgressBar(true)

.BusinessUrl(Url.Action("StartWorkflow", "home"))

.TaskTitle("测试报销")

.IconList("glyphicon glyphicon-plus,glyphicon glyphicon-envelope")

.IsSplit(true)

.DictionaryWfClientUser(Model.DictionaryWfClientUser)

)

### 属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 属性 | 说明 |
|  |  |  |
| Action地址 | ActionUrl | 指定业务请求的地址，即Action地址。此参数必须提供。 |
| 调用前事件 | BeforeClick | 在向服务器递交数据前触发的客户端事件，此事件形如function beforeClick(sender, progressbar)，此方法要求返回表单数据，以便将数据提交给Action，可以调用Jquery的serialize()搜集数据。 |
| 调用后事件 | AfterClick | 请求返回后调用的客户端事件，此事件形如function afterClick(result, sender, progressbar)，result携带了从服务端返回的数据，result.BusinessData携带业务数据，具体数据结构由业务决定，result.ProcessInfo携带有当前流程详细信息。 |
| 流程模板 | TemplateKey | 待启动的流程模板名称，此参数必须提供。 |
| 表单地址 | BusinessUrl | 业务表单地址，从待办列表进入业务页面时，需要使用此参数。此参数必须提供。 |
| 任务标题 | TaskTitle | 待办任务的标题，此参数必须提供。 |
| 动态角色审批人列表 | DictionaryWfClientUser | 添加动态角色审批人，需要选择审批人的情况下使用 |
| 按钮组显示方向 | Direction | 按钮组显示方向 |
| 图标列表 | IconList | 图标列表，以逗号分隔 |
| 控件类型 | ButtonType | 是提交按钮还是普通按钮，默认提交按钮 |
| 是否需要保存按钮 | IsShowDraft | 是否显示保存草稿按钮 ，默认true |
| 是否分隔 | IsSplit | 是否将按钮分隔开 ，默认false |

备注：流程后台action同启动按钮一致

## 流程流转按钮横向显示

### 书写方式

@(

Html.HSRUIWF().WFToolbarMoveToWorkflow().Direction(p => p.Default()).ActionUrl(Url.Action("MoveTo", "home"))

.AfterClick("afterClick").BeforeClick("beforeClick")

.Name("ExpenseApprove").ProgressBar(true).IsSplit(true)

.DictionaryWfClientUser(Model.DictionaryWfClientUser)

)

### 属性说明

|  |  |  |
| --- | --- | --- |
| 属性名称 | 属性 | 说明 |
|  |  |  |
| 调用前事件 | BeforeClick | 在向服务器递交数据前触发的客户端事件，此事件形如function beforeClick(sender, progressbar)，此方法要求返回表单数据，以便将数据提交给Action，可以调用Jquery的serialize()搜集数据。 |
| 调用后事件 | AfterClick | 请求返回后调用的客户端事件，此事件形如function afterClick(result, sender, progressbar)，result携带了从服务端返回的数据，result.BusinessData携带业务数据，具体数据结构由业务决定，result.ProcessInfo携带有当前流程详细信息。 |
| 确认框文本 | DialogText | 确认框文本内容 |
| 进度条 | DialogText | 是否需要进度条，默认True |
| 动态角色审批人列表 | DictionaryWfClientUser | 添加动态角色审批人，需要选择审批人的情况下使用 |
| 按钮组显示方向 | Direction | 按钮组显示方向 |
| 图标列表 | IconList | 图标列表，以逗号分隔 |
| 控件类型 | ButtonType | 是提交按钮还是普通按钮，默认提交按钮 |
| 是否分隔 | IsSplit | 是否将按钮分隔开 ，默认false |

备注：流程后台action同流转按钮一致

# 待办

## 待办查询示例

UserTaskQueryCondition condition = new UserTaskQueryCondition();

condition.TaskType = TaskStatus.Unprocessed;

ServiceFactory sf = new ServiceFactory();

var readerService = sf.CreateService<ITaskPluginBizlet>();

return readerService.QueryTask(tenantCode, userId, condition, 1, 10, null)

## 获取最新N条

此方法一般在首页使用，直接返回当前登陆人最新的N条数据，下面的示例返回10条。

UserTaskQueryCondition condition = new UserTaskQueryCondition();

condition.TaskType = TaskStatus.Unprocessed;

ServiceFactory sf = new ServiceFactory();

var readerService = sf.CreateService<ITaskPluginBizlet>();

return readerService. QueryTaskUnProcessedTop (tenantCode, userId, condition, 10)

## 待办缓存

在有些业务场景，需要缓存待办信息，以避免过多的访问数据库。

### 接口定义

/// <summary>

/// 业务从数据库加载接口

/// </summary>

public interface ITaskDbLoader

{

/// <summary>

/// 过期时间

/// </summary>

TimeSpan ExpiredTimeSpan { get; set; }

/// <summary>

/// 缓存Key

/// </summary>

string ChacheKey { get; set; }

/// <summary>

/// 在缓存中无数据时，需要加载的业务数据接口

/// </summary>

/// <typeparam name="T">数据类型</typeparam>

/// <param name="userId">用户Id</param>

/// <param name="tenantCode">租户编码</param>

/// <param name="IsTenantMode">是否是多租户模式</param>

/// <returns>预警信息</returns>

T LoadDataFromDb<T>(string userId, string tenantCode, bool IsTenantMode)

}

### 接口实现示例

在LoadDataFromDb的方法中，编写业务欲调用的待办接口即可。

public class OMPHomeTop10TaskDbLoader : ITaskDbLoader

{

private string \_CacheKey = CommonConfig.CacheKey\_OMPHomeWorkflowTop10;

private TimeSpan \_ExpiredTimeSpan = new TimeSpan(0, 10, 0);

public string ChacheKey

{

get

{

return \_CacheKey;

}

set

{

\_CacheKey = value;

}

}

public TimeSpan ExpiredTimeSpan

{

get

{

return \_ExpiredTimeSpan;

}

set

{

\_ExpiredTimeSpan = value;

}

}

public T LoadDataFromDb<T>(string userId, string tenantCode, bool IsTenantMode)

{

UserTaskQueryCondition condition = new UserTaskQueryCondition();

condition.TaskType = TaskStatus.Unprocessed;

ServiceFactory sf = new ServiceFactory();

var readerService = sf.CreateService<ITaskPluginBizlet>();

return readerService.QueryTask(tenantCode, userId, condition, 1, 10, null) as T;

}

}

### 接口实现注册

在Portal和OguService的Ioc中注册：

sl.Register<ITaskDbLoader, OMPHomeTop10TaskDbLoader>(alwaysNew: true);

### 从缓存取数据

UserTaskQueryCondition condition = new UserTaskQueryCondition();

condition.TaskType = TaskStatus.Unprocessed;

ServiceFactory sf = new ServiceFactory();

var readerService = sf.CreateService<ITaskPluginBizlet>();

return readerService.GetTaskFromCache(cacheKey,userId, tenantCode);