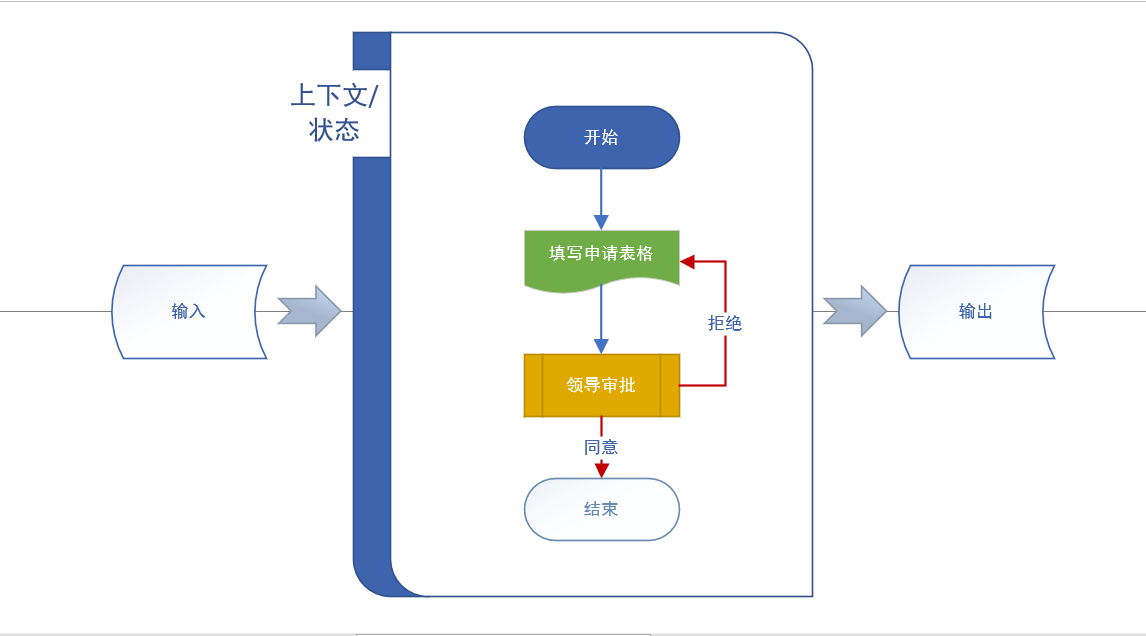
# 工作流

## 概念

### 定义

**工作流workflow(设计时称为流程图Diagram)**是指：按一定顺序或条件，执行并完成一系列活动/工作的过程



### 过程

工作流是一个**过程Process(设计时称为Procedure)**，意味着它具有**输入Inputs**，进行**处理Deal**后，产生某些**输出Outputs**。在处理时，会产生**中间状态或上下文Context**

### 活动

工作流是要完成一系列的**活动Activity**(设计时称为**节点node**)。每个活动也符合过程定义，即每个互动也有输出/输出/上下文。

### 路径/顺序

工作流的活动activity是按照一定的顺序或条件Route(设计时称为关联Association)去执行的执行

### 执行

一个工作流从外部可以看作是一个没有路径的单一节点的活动。

工作流也可以看作一个状态机。状态的变化特定的活动结束，特定的活动开始。

**工作流的流动是以流程图为导向，由状态变化做推动**。

状态的变化是由(且只会由)以下两种方式引起:

1. 外部设置工作流或活动状态
2. 活动的执行引起的当前上下文状态变化

## 术语

### 设计期/运行期

工作流的图形表示，可分为设计期Design与运行期Runtime。设计期的术语更偏重描述工作流作为的“图”的特性。而运行期的术语更偏重描述工作流的“步骤/行动”特定。

以下详细解释设计期于运行期的术语，这些术语会应用于源代码中。

#### 流程图Diagram

设计期在工作流开始之前，指定了活动与连接各活动的路径，通常我们称这个为**流程图Diagram**,

#### 节点Node

其中定义的活动为**节点node**.

#### 连接Association

节点与节点之间的连接为连接Association。

#### 工作流实例Workflow Instance

工作流开始后，工作流执行引擎根据流程图定义的路径，真实的执行各活动。通常我们称正在执行中的工作流为**工作流实例Workflow Instance**,

#### 行动Action

其中载入内存，并准备执行或正在执行的活动称为**行动Action**,

#### 路径Route

运行期中，行动与行动之间的关联为**路径Route**.

### 有向图

图是指由节点与连接构成的数据结构。

有向图是指节点与节点之间的连接是有指向的图结构。

工作流是一个有向图数据结构

### 父节点/子节点

工作流的活动可以进一步分解成不同的活动。这样在静态图上，可以看到活动有时候是树形结构。

### 执行引擎

执行引擎(Workflow Execute Engine，WEE)是工作流或活动的工厂，它创建了活动的实例并执行活动的相关代码。

当某个活动处于完成状态，执行引擎会根据流程图中的路径，加载下一个应该执行的活动。

### 上下文

工作流或工作流活动Activity本身是可以看作状态集合。在编程视角下，这些状态被称为上下文context.

上下文的内容，最初来自工作流的输入Inputs与配置的状态。其二来自于上一个活动的输出。

如果有子节点，子节点的上下文内容，也可以从父行动的上下文中**导入Import**；也可以**导出export**某些状态的值到父行动的上下文。

## 使用

### 创建工作流

Using(Var activity = Engine.CreateFlow(“流程图名”,creator,inputs)){}

### 外部调整工作流状态

Using(var activity = Engine.LockActivity(“活动Id”,dealer)){

Actrivity.MyName = “yiy”;

}

### 进程内/进程外

进程内使用，引擎会在调用代码的进程内执行工作流代码。

进程外使用，Engine实际上会创建一个代理Activity.该Activity结束时会调用远程接口，朝远程的工作流引擎实例发送请求。

## 特点

本工作流组件特别针对审批流做实现。如果你需要实现一个很复杂的业务流程，使用这个组件是很恰当的。而如果你需要一个业务逻辑相对固定、强调执行效率的工作流，本工作流并不是一个很好的选择。本小节用于阐述这个工作流的特定，方便技术决策者判定是否该组件是否适合自己的项目目标。

### 执行的可中断可重入

工作流可以执行到中间某个环节被人为/意外中断。其后重入/重启该工作流并不收中断影响(执行效果就像未中断一样)。

该特点主要用于以下场景:

审批流转给某人后，某个人未必能第一时间收到审批通知，或收到了通知未必能在足够短的时间内审批。工作流(审批流)实际上是中断了，并被移出了内存或执行队列。等该审批人点击审批后，该工作流会重新载入，并按照原先的样子继续执行。

该能力的实现在于，每走一个完整的步骤，工作流都会保存自己的内在状态。中断后，它可以从存储中重新加载这些状态恢复现场。

该实现也决定了，它并不是一个高性能的工作流。如果要让该工作流组件达到高性能，需要负载均衡等许多额外的设计。

### 流程图的动态性

工作流的执行路径，可以部分的在设计期规划好。也可以在执行期动态规划。

该设计主要适用于以下场景:

某些审批任务，并非是要特定的人，可以拉入任何人做审批，直到满足特定的条件；被拉入审批中的人，要按照职务高低做排序，职务高的排后面。对于这种审批路径会随着审批节点的执行情况发生变化，且该变化无法事先规划好的场景，需要我们从工作流执行机制上去解决。

该特点的实现为开放了某些接口，允许活动内部修改流程图Diagram.由于引擎的执行是以流程图为导向，改变流程图就改变了工作流的执行。同时，由于流程图Diagram会在运行期发生改变，运行期的Diagram也会作为状态之一，在工作流的执行过程中反复存储。即使我们针对该情况做了优化，但也导致潜在的性能低下。

### 流程图的可热替换

工作流设计期流程图的变更，不会影响到已经以该流程图启动的工作流实例。这允许我们热机更新工作流设计流程图配置。这个特点对工作流会随着业务发展而变化的场景非常有用。把支持业务变更所需要的的支持工作量降低到了非常小的地步。

### 脚本式的活动

工作流活动，大部分都需要手写代码，并编译成二进制包。但本工作流也支持直接在工作流的设计时，写一些简单的脚本来处理简单的节点。

该实现主要依赖于dotnet core的Roslyn编译平台。

### 活动可注入

活动的实例，可以与依赖注入配合。

同时，活动的实例，也可以注入Context的数据。