

C#10 审批流系统开发计划

技术架构与系统模块划分

图 1: 审批流系统的总体技术架构示意

本审批流系统采用分层架构，前端包括 WinForms 桌面客户端和 Web 流程设计器，后端基于 .NET 6+ 实现核心流程引擎和服务。架构支持 **SaaS 多租户** 部署和私有云独立部署。各模块划分如下：

- **客户端前端**：主要由 WinForms 应用构成，提供日常审批操作的用户界面。同时包含一个嵌入的 Web 流程设计器界面（可通过嵌入浏览器控件或单独的管理门户访问），供流程管理员设计和配置审批流程。
- **流程设计器前端**：采用现代 Web 前端技术实现（后续详述），支持流程可视化建模、节点配置、路由规则定义等功能，供管理员通过浏览器进行流程设计。
- **后端服务**：基于 .NET 6/7 Web API 架构，实现审批流引擎、用户与组织机构服务、通知服务等。后端负责解释流程定义、调度任务流转、状态管理和持久化。核心流程引擎模块类似于 Activiti，引擎接口与实现相分离，支持模块化扩展。
- **数据库存储**：使用关系型数据库（如 PostgreSQL 或 SQL Server）存储流程定义、流程实例、任务、历史记录以及组织和用户数据等。针对 SaaS 场景，每个租户的数据**物理隔离**存储，建议为每租户使用独立数据库实例以确保数据隔离性和合规^①。私有部署则使用独立数据库即可。
- **集成接口与服务**：包括组织/权限管理模块、通知消息推送模块、数据导入模块等独立组件，通过后端 API 与引擎交互。权限与组织服务确保用户和角色数据的获取与隔离；通知服务负责对接邮件、短信、钉钉、企业微信等外部消息系统；数据导入模块提供从 CSV 文件或外部 API 同步用户/部门数据的功能。

上述架构保证引擎核心聚焦于流程控制，将表单展现、权限控制、通知发送等作为可插拔模块分离。这种解耦设计符合“**引擎只负责流转，不处理表单和权限**”的原则^②，提高系统的灵活性和可维护性。

流程建模器选型与嵌入方案

流程建模器是系统的关键前端组件，负责提供流程图形化配置界面。本方案不采用传统的 bpmn.js，而选择**现代化的开源节点图编辑框架**来实现，例如 **React Flow**、**Drawflow** 或 **JointJS** 等。React Flow 是一个基于 React 的可定制组件库，适合构建节点式流程编辑器^③；Drawflow 和 JointJS 则提供类似能力，也支持拖拽节点、连接线等功能。

选型方案：考虑到 React 生态活跃以及高度可定制性，建议采用 **React Flow** 作为流程建模器框架。React Flow 拥有良好的文档和示例，支持我们定义自定义的节点类型、边和交互逻辑，能够满足审批流程建模的需求，如开始/结束节点、审批任务节点、条件分支、汇聚节点等元素的绘制和配置。JointJS 亦是成熟的图形库，但相比之下 React Flow 在社区支持和现代 UI 方面更有优势。

嵌入实现：由于前端主体为 WinForms 桌面应用，我们有两种方式集成 Web 流程设计器界面：

1. **嵌入浏览器控件**：在 WinForms 内嵌入一个 WebView（如使用 CefSharp 等浏览器组件），加载承载 React Flow 流程设计器的本地网页或内置的网页资源。这样管理员在桌面客户端中即可直接打开流程设计器界面，进行流程的绘制和配置。

2. **独立Web门户**：搭建一个供管理员使用的独立Web应用（采用 ASP.NET Core + React 技术栈），集成 React Flow 流程设计器。管理员可通过浏览器访问该设计器门户。WinForms 客户端中则提供一个入口（如按钮或菜单）快速打开该Web设计器地址（可以是内置浏览器控件，也可以外部浏览器）。

综合考虑用户体验和实现工作量，**推荐**使用 WinForms 内嵌浏览器控件的方式，以提供一致的一体化客户端。但同时也支持在纯浏览器中访问流程设计器，以便在SaaS场景下无需安装客户端也能进行配置。

流程模型与保存：流程设计器中绘制的流程图可采用 BPMN 2.0 标准或自定义JSON格式保存。由于BPMN具有统一的符号和语义标准，有利于业务人员和开发人员对流程的理解⁴，后端引擎也可直接基于BPMN定义执行流程。因此建议建模器支持导出 BPMN XML 或等价的JSON配置，并通过后端API将流程定义保存到数据库中（含版本信息）。设计器应提供版本管理功能，保存时区分流程定义的版本号，以支持后续流程的版本升级和兼容。

后端流程引擎实现方案（类 Activiti 引擎）

后端采用 .NET 6/7 实现核心的审批流程引擎，目标是提供媲美 Java Activiti/Flowable 引擎的能力。引擎负责解析流程定义、创建和推进流程实例、分配和管理任务、以及执行各种审批操作。以下是实现方案细节：

引擎架构与组件

引擎将设计为**可嵌入的库/服务**，通过公开的API供应用调用。主要组件包括：

- **流程定义模块**：负责存储和管理流程模型（例如 BPMN/XML 或 JSON 定义）。提供流程发布、版本管理功能，每次修改流程都会生成新版本，已启动的流程实例继续按其创建时的版本执行，新流程实例则使用最新版本。这确保流程升级不中断既有流程。
- **流程实例管理**：负责流程实例的创建、状态流转和生命周期管理。每个实例包含当前节点、已完成节点、流程变量等状态信息。引擎通过读取流程定义，按照定义的顺序和条件推进实例的节点状态。
- **任务调度与执行**：引擎内置任务调度器，用于推进流程节点。对于用户审批节点，引擎会生成**待办任务**指派给相应审批人；对于系统自动节点（如脚本任务、邮件发送等），引擎可直接执行动作。为了处理异步任务和定时器事件，可集成 HangFire 等定时作业组件，实现节点超时检查、定时触发等功能⁵。
- **审批操作控制**：提供对常见审批操作的支持逻辑，包括串签、会签、加签、退回、跳转、撤回等：
- **串签（顺序审批）**：引擎支持在一个审批节点上设定多位审批人按顺序依次审批。实现上，可将该节点视为子流程或队列，按照顺序依次生成单人审批任务，前一人通过后再生成下一人的任务。
- **会签（并行审批）**：引擎允许配置某节点由多个审批人同时审批，即生成并行的多个审批任务。可配置会签通过条件，例如“任意一人通过/全部通过/过半通过”等，当达到条件时该节点通过，否则若有人拒绝则判定驳回。
- **加签（加补审批人）**：在流程进行过程中，允许当前审批人临时增加额外审批人。引擎实现时，可在当前节点动态插入新的审批任务（可在当前或后续步骤），该任务完成后流程再继续原流程。需记录加签操作痕迹。
- **退回**：审批人可将流程退回到**上一步**或某个指定的前序节点。引擎需提供机制将流程实例的当前活动点重置到目标节点，并可能标记当前节点为退回状态。退回后，已完成的后续节点可能需要作废或保留历史视图，流程状态回溯。
- **跳转**：管理员或特殊权限人可以将流程直接跳转到任意节点（前进或后退）。实现需慎重，通常通过API提供“跳转”操作，指定目标节点，重置流程状态并生成该节点的任务。这相当于非正常流转，用于异常处理场景。
- **撤回**：流程发起人在特定条件下（如流程未结束且尚在首节点或下一节点）可以撤回申请。引擎实现为将该流程实例标记为撤回状态并终止后续流转，相关待办任务取消。需记录撤回人和时间，用于审计。
- **引擎服务接口**：设计统一的服务接口，如 `StartProcess`，`ApproveTask`，`RejectTask`，`AddSigner`，`JumpToNode` 等，用于外部调用以对流程实例执行操作。接口层校验权限、状态有效性，然后调用引擎核心逻辑变更流程状态。

- **状态持久化与事务**：引擎操作需具备事务性，确保流程状态一致性。利用数据库事务机制，当流程推进涉及多个更新（流程实例、任务、历史记录），要么全部成功要么全部回滚。同时采用**持久化队列**或事件日志记录关键节点变化，支持故障恢复。比如使用工作流引擎支持的作业执行日志，或结合消息队列确保关键操作的可靠记录。

引擎核心实现将遵循工作流引擎的常见模式，如使用**有限状态机**或**令牌(Token)机制**管理流程实例的状态迁移，确保流程按定义顺序执行任务、由正确的人员或系统执行⁶。通过模块化设计，引擎支持未来扩展新的节点类型（例如集成新的服务任务）、新的网关决策规则，以及与外部系统的交互。

流程定义与条件节点配置

流程定义采用图形化方式创建，但需在引擎内部表示为可执行模型。建议采用 **BPMN 2.0** 作为流程定义标准格式，或在数据库中建立等价的数据模型（流程、节点、连线、网关等表）。使用BPMN的好处是标准化和易于沟通⁴，同时很多现成工具支持BPMN，可降低学习成本。引擎解析 BPMN 时，需重点处理各类元素：

- **任务节点**：对应用户任务（需审批人处理）或服务任务（系统自动执行）。用户任务节点会绑定候选人规则（详见下一节），服务任务节点可配置调用外部接口或执行脚本。
- **网关节点**：包括并行网关和排他网关。并行网关用于实现会签（并行分支同步），排他网关用于条件路由，只允许满足条件的路径通过。
- **事件节点**：如开始、结束，以及中间事件（如定时器事件）。引擎需要支持定时器事件来实现如“节点超时自动转交”这样的需求。通过HangFire等调度在后台监控定时器事件，触发相应的流程跳转。
- **子流程/嵌套流程**：引擎可以支持子流程定义，用于封装可复用的审批片段或者实现串签逻辑（顺序审批可以建模为一个子流程顺序执行多个任务）。

历史日志与监控

系统须提供**流程日志**和**历史记录**功能，将每个流程实例的关键操作（提交、审批、驳回、转发、退回、跳转等）记录到历史表中，包含操作人、时间、意见等。这样不仅方便业务查询，还支持审计追踪。引擎完成节点后，应将任务移至历史数据表，减小运行时表的压力。提供查询接口按流程实例或申请单号查询流程进展和历史轨迹。

同时，为了运维需要，可对引擎运行状况进行监控，比如当前运行中的流程数量、各节点平均处理时间、各审批人处理量等，作为后续优化流程和分析瓶颈的依据（这些属于扩展的报表分析功能，可在二期实现）。

条件路由与审批人配置机制

本系统支持灵活的流程路由和审批人指定机制，通过图形化界面配置，后端引擎解析执行。路由规则和找人机制包括以下几种方式：

- **固定审批人/角色**：直接在流程节点配置指定的审批用户、用户组或角色。例如某节点规定由申请人部门经理审批，则可在设计时将“部门经理”角色赋给该节点，运行时引擎根据申请人数据查找其部门经理用户。
- **条件表达式**：使用布尔表达式决定流程走向或审批人。例如可配置条件“申请金额 > 10000”时走向财务总监审批，否则走向普通经理审批。表达式使用简单的脚本语言（如基于Excel公式风格或JavaScript），引擎在运行时计算表达式结果，选择匹配条件的路径或人员。我们可以利用开源的表达式解析库，如 **DynamicExpresso** 或 **NCalc** 等，来安全地计算用户定义的表达式。
- **脚本/代码片段**：允许在流程配置中编写脚本代码来决定下一步审批人或处理逻辑。例如使用 C# 脚本（Roslyn 编译）或 Python 脚本（通过嵌入解释器）来灵活指定审批人列表。脚本方式适合复杂逻辑，但需要对使用者权限进行控制，防止脚本滥用。实现时，引擎在到达节点时执行该脚本，脚本从上下文获取流程数据、表单数据，输出一个用户列表或下一节点ID。

- **规则引擎**：对于更加复杂、多条件组合的审批人选择和路由决策，可集成业务规则引擎。例如采用 **NRules** 这类基于Rete算法的规则引擎⁷。管理员可以在规则库中定义多条规则，如基于申请类型、金额、部门等因素选择不同的审批流转路径。引擎在运行时将当前流程的事实数据（申请属性等）提交给规则引擎，规则引擎推理出应该选取的审批人或决定流程走向⁸。NRules 是 .NET 平台成熟的开源规则引擎，实现高效规则匹配并允许用熟悉的C# DSL编写规则⁷。通过引入规则引擎，能够将复杂的找人策略从代码中解耦，业务人员也可参与配置⁹。

审批人查找机制方面，引擎需要与组织架构数据结合。对于配置的角色或组织节点，系统应能从已导入的“用户-部门-角色”数据中找到对应的用户列表。例如：节点指定“申请人上级主管”，则引擎根据申请人的部门关系或直属上级字段找到其主管用户；如果指定“财务部”则找到财务部下具有审批权限的用户。为支持这一点，我们在组织模块中设计了便捷的查询接口，如 `GetManager(userId)` 或 `GetUsersByRole(deptId, roleName)` 等供引擎调用。

流程设计器前端需要提供**可视化的界面**来配置上述规则。例如：在流程节点属性面板中，支持用户选择“审批人来源”为固定人员/角色、或者选择“使用规则/脚本”，并弹出对应的编辑框让管理员填写表达式或编写脚本。对于复杂规则引擎的配置，可考虑提供简单的规则配置表单或直接引用预先定义好的规则名称。

通过以上机制，系统实现高度灵活的审批路由配置，既可满足简单场景（直接指定人员），又可应对复杂场景（跨部门多条件选人）。这样的规则引擎与表达式结合设计，使流程**审批人机制**既直观又强大。

数据持久化与运行时流程状态管理

数据持久化方案将围绕**流程引擎的元数据和实例数据**进行设计。采用关系型数据库能够提供事务支持和方便查询，推荐使用 PostgreSQL 或 SQL Server。主要的数据库表设计如下：

- **用户和组织表**：如 `Users`，`Departments`，`UserRoles` 等，用于存储从外部导入的组织架构信息。包含用户基本信息、所属部门、上级领导、角色等字段。
- **流程定义表**：如 `ProcessDefinitions`，存储流程模型的元数据（名称、版本、状态等），以及流程图模型本身（可存XML或JSON文本）。还可拆分出 `ProcessNodes`，`ProcessTransitions` 等表，存储每个流程各节点及连接关系，便于查询分析。
- **流程实例表**：如 `ProcessInstances`，每个记录对应流程定义一次运行实例，包括所属流程定义ID、发起人、当前状态（进行中/已完成/撤回等）、当前所在节点、启动时间、完成时间等。
- **任务表**：如 `Tasks`，存储具体到达的用户任务（审批任务）。字段包括对应流程实例ID、节点ID、审批人ID、状态（待处理/已处理/转交等）、开始时间、完成时间、审批意见等。对于并行会签，会有多条任务记录对应一个流程节点。
- **历史记录表**：如 `ProcessInstanceHistory`，`TaskHistory`，用于归档已结束的流程实例及任务。每当流程实例完成或终止时，将相关记录移动到历史表，以保持运行中表数据量适中。另外每次节点通过/驳回都记录到历史表，包含操作类型、操作者、时间、意见等，实现审计追溯。

运行时状态管理：引擎运行过程中，为了高效操作，会将当前执行状态缓存在内存中（例如一个流程实例的活动令牌位置）。但是所有关键状态仍需持久化到数据库以防止服务重启或故障造成状态丢失。采用以下策略：

- 每当流程实例状态变迁（进入下一个节点、任务完成等）时，及时更新 `ProcessInstances` 和相关 `Tasks` 表记录，标记新的状态。使用数据库事务保证关联更新一致性。如果引擎崩溃或重启，可根据数据库记录恢复流程进行中的状态。
- 对于运行中的长流程，可定期持久化检查点（如果流程定义复杂可序列化整个状态机）。一般审批流不需要复杂的序列化，因为状态已体现在当前节点和未完成任务上。
- 引擎应支持**乐观锁或悲观锁**机制避免并发更新冲突。例如并行审批时，多个任务完成几乎同时推进后续节点，需要确保同步点正确处理一次。可在流程实例表增加版本号用于乐观锁，或利用数据库锁保证同一流程实例的状态更新串行化。

- 利用索引优化常用查询：如按审批人查找待办任务、按部门或流程类型统计流程。针对这些查询在 Tasks表、Instances表建立必要索引，加速查询性能。

此外，针对条件路由和规则，可能需要存储**流程变量**（Process Variables）。这可以在流程实例表或单独的 ProcessVariables 表存储键值对。在流程执行过程中，变量用于条件判断或传递数据（如表单字段值）。这些变量也需要持久化，以支持流程跳转或暂停后的恢复。

系统部署与多租户方案

本系统设计同时支持 **SaaS 模式**和**私有云部署**。为此需要实现多租户隔离和灵活的部署拓扑。

多租户支持：在SaaS场景下，多个租户（客户）共享同一套应用程序实例，但数据相互隔离。我们的多租户设计遵循**数据隔离优先**的原则，每个租户具有独立的数据库存储。推荐做法是**为每个租户配置独立数据库实例**或独立的数据库 schema，从物理上隔离数据¹。这样能满足诸如不同行业对数据合规的要求，确保各租户数据互不可见。同时应用层通过一个租户ID来路由到正确的数据库连接。应用启动时可根据配置初始化多个 DbContext 或连接字符串池，根据当前登录用户的租户身份切换。另一个层面的隔离是**逻辑隔离**：在代码中对多租户的数据操作都要基于租户ID过滤，例如查询任务时会限定 `WHERE TenantId = 当前租户`。可以使用框架（如EF Core的多租户过滤器）简化实现。

我们还将提供**租户管理界面**（类似FlowWright的Multi-tenant Manager¹⁰）：用于 SaaS 平台运营者创建/停用租户、为新租户初始化数据（如创建数据库、预置管理员账号）等。这部分可作为一个后台管理模块。

部署拓扑： - SaaS部署：典型采用云端多实例负载均衡部署。应用服务器可以是无状态的 Web API，部署多个节点 behind a load balancer，共享用户请求。各节点通过连接池访问各租户数据库。为了扩展性，也可以按租户分配服务器（比如大客户独占一些节点）。如FlowWright所述，还可将不同租户的流程负载分布到不同服务器以扩容¹¹。所有应用节点访问统一的组织和权限服务以及消息服务。消息队列（如 Azure Service Bus 或 RabbitMQ）可用于在分布式节点之间传递流程事件（例如某节点生成了新任务通知，由队列消费者处理发送通知）。 - 私有部署：以单租户模式独立部署在客户环境。通常将所有后端组件部署在客户的服务器或容器中，连接客户自有数据库。支持Docker容器部署或传统IIS部署。私有部署可根据规模调整，轻量情况下前后端在一台服务器，复杂情况也可拆分多个服务。需要提供简便的安装部署脚本或容器编排配置（如Docker Compose或 Kubernetes Helm charts）来帮助客户快速部署。

无论SaaS还是私有，系统都支持**横向扩展**。业务无状态部分（如Web前端、API服务）均可多实例部署；状态部分主要在数据库，通过提高数据库性能或使用读写分离等手段提升并发。对于审批流程引擎这种长流程场景，也可考虑**分布式锁**方案确保在多节点环境下同一流程实例只由一个节点处理（例如利用 Redis 锁或数据库锁）。

安全方面，每个租户有自己的用户体系和权限配置，在SaaS模式下需要**统一的身份认证服务**来区分租户。可采用JWT令牌携带租户信息，用户登录时系统会验证其所属租户并加载对应数据上下文。

移动端支持方案

为了满足用户随时随地处理审批的需求，系统需要提供移动端的支持方案。考虑几种可行途径：

- **响应式Web界面 (H5)**：开发响应式的Web前端，使审批界面可以在移动设备浏览器上良好显示和操作。采用自适应布局的HTML5页面，保证常用操作（查看待办、审批通过/驳回、填写意见等）在手机上方便点击。由于后端是Web API，可以直接复用同一套接口。移动Web的优点是免安装更新方便，缺点是体验略逊于原生。
- **移动App**：提供专门的移动应用，可以使用跨平台技术如 Xamarin.Forms / .NET MAUI 或 Flutter 等开发 iOS/Android 双平台的APP。App通过调用后端API实现功能，可嵌入部分Web组件以减少重复开发

（例如表单展示可以用内嵌Web）。如果采用 .NET MAUI，我们可以与桌面部分共享部分业务逻辑代码。原生App优势是用户体验好、可使用推送通知等手机能力。

- **小程序集成**：针对国内常用的 **微信企业微信**、**钉钉** 等办公IM，可开发对应的小程序或微应用，嵌入在这些平台内实现审批操作。例如企业微信的小程序，与我们后台通过API通讯，用户在微信中即可收到待办通知并审批。这利用了用户现有的办公App，提高使用便利性。
- **超级App内嵌**：如果客户已有自己的移动App（如ERP/OA系统App），我们可以提供一个H5页面或SDK供其App嵌入，从而将审批功能集成进去。比如提供一个WebView加载审批H5页面，以及一套移动端UI组件，App调用我们SDK打开审批界面并完成操作。

综合考虑，**首期**可以采用**移动Web(H5)**方案快速提供支持，保证所有功能至少在浏览器可用。同时针对有钉钉/企业微信使用偏好的客户，开发相应的轻量小程序以便直接在IM平台中处理审批。后续视用户量和需求，再考虑推出完整的独立移动App。在所有移动方案中，统一调用后端API，使功能和权限与PC端保持一致。

必须确保移动端的安全性，如使用 HTTPS 加密通讯，移动端登录采取OAuth单点登录或与PC端共用令牌，并提供设备管理和必要的远程登出功能，防止账户冒用。

通知消息推送整合

审批流程往往需要实时的通知提醒，例如当用户有新的待办事项，或流程被退回时通知相关人员。本系统将设计一个**通知推送模块**，支持集成多种消息渠道：

- **邮件 (Email)**：通过SMTP或企业邮件服务器发送邮件通知给用户邮箱，包括待办提醒、流程结果通知等。邮件模板内容支持自定义，可包含流程名称、申请人、链接等信息。
- **短信 (SMS)**：集成短信网关服务，在用户未及时处理时发送短信提醒到手机号。这需要对接第三方短信服务提供商的API（如阿里云短信、Twilio等）。
- **钉钉**：通过钉钉开放接口向指定用户发送工作通知消息。需要企业内部有钉钉应用的Webhook或通过钉钉应用的机器人API发送。消息内容可直接跳转至我们的H5审批页面。
- **企业微信**：类似地，对接企业微信的消息接口，推送审批通知给对应企业微信用户，可在其中打开小程序或H5链接进行处理。
- **应用内通知**：对于我们的WinForms客户端和将来的移动App，也内置通知提示功能（如桌面弹窗提醒或App推送）。WinForms可通过托盘消息或窗口提示来提醒新任务。

为实现以上集成，我们抽象出**通知服务接口**，不同渠道作为实现类插件。系统配置上允许启用/停用特定通知渠道，并为每种渠道设置参数（如SMTP服务器配置、钉钉的AppKey/Secret等）。当流程引擎产生事件（如某任务创建或超时），会调用通知服务接口，传入事件类型和接收人，由通知服务负责调度各渠道发送¹²。例如：新任务产生时，默认通过邮件和钉钉通知审批人。

需要注意消息推送的**频率和聚合策略**，避免骚扰用户。可以设置例如每隔N分钟汇总待办，或者工作时间段内发送。另外对短信这种收费渠道要可配置哪些紧急情况才使用。所有通知发送结果也应记录日志，便于排查问题（如用户声称未收到通知时可查日志）。

通过多渠道通知，系统可确保用户及时获知审批请求。¹³ 指出用户界面可以包括多种形式（web、移动、集成到现有系统），结合消息推送提升了整个流程的**协同和及时性**。

数据导入模块方案（CSV 与 API）

组织架构数据（用户、部门等）是审批找人机制的基础。为了减少初始配置工作量和保持数据同步，我们提供**数据导入模块**来从外部系统导入用户和部门信息。

CSV导入：管理员可以通过系统后台界面上上传包含用户或部门数据的CSV文件。我们将提供标准的模板格式，例如用户CSV包含字段：员工编号、姓名、部门、职位、邮箱、手机号、上级主管等；部门CSV包含：部门编号、部门名称、上级部门编号、部门主管等。导入流程：系统解析CSV文件（使用成熟的CSV解析库，如CsvHelper），映射字段到用户/部门实体。进行必要的清洗和校验（如必填字段是否有值，引用的上级部门是否存在等）。然后批量写入或更新数据库中的用户、部门表。需要处理并发和差异更新问题：如已存在的用户则更新信息，新用户则插入，离职用户可标记为无效。导入操作完成后给出结果报告（成功数量、失败原因）。

API接入：对于数据实时性要求高的场景，系统支持通过API同步组织数据。提供一组 RESTful 接口或定制的数据同步服务，例如：`POST /api/import/users` 提交用户列表JSON，`POST /api/import/departments` 提交部门列表JSON。也可以按单条提供 `PUT /api/users/{id}` 更新用户信息等。这样外部人事系统或AD域可以编程调用这些接口，实现与审批系统的组织数据同步。出于安全考虑，API可以使用Token认证或限定IP调用，数据传输采用HTTPS。还可以设计调度任务从外部系统拉取数据：例如配置每天凌晨通过LDAP读取AD用户，或通过HTTP调用HR系统的员工列表接口，同步更新到本系统数据库。

数据映射与转换：导入模块需要考虑外部系统字段与本系统字段的不一致。可以提供字段映射配置功能，让管理员在CSV导入时选择CSV列与系统字段的对应关系；对于API方式，则可能需要根据对接系统定制解析。为了灵活性，数据导入处理可实现为插件式，例如对接不同HR系统（SAP, Oracle HR等）开发不同适配器，只要将数据转成统一格式再导入。

增量更新：如果数据量大，不可能每次全量导入。需要支持根据时间戳或标记进行增量更新。比如记录上次导入时间，下次只处理新增加或更新的记录。对于已删除的用户/部门，也需要有方案（例如标记为删除，不在审批人选择中出现）。

通过CSV和API两种方式并存，既满足一次性批量导入初始化数据，又能支持后续的自动同步，保证审批流系统内的组织与人员信息是最新的，为准确找人提供数据支撑。

权限控制与组织结构建模设计

组织结构建模：系统将组织架构建模为部门（组织单元）树和用户两类主要实体，并支持角色与用户组：
- **部门（组织单元）**：以树状层级表示公司组织结构，每个部门有唯一ID和名称，关联上级部门ID。部门可附加属性如部门主管UserID等。
- **用户**：具有个人基本信息，关联所属部门ID、上级主管ID（或从部门主管派生）、及其全局角色列表。用户和部门是多对一关系（一个用户只属于一个主部门），但也可扩展支持用户在多个部门挂职的情况。
- **角色**：系统预定义或自定义的角色集合，如“部门经理”、“财务人员”、“人事专员”等。角色可以是全局的或者按部门设定（如某用户在A部门是经理，在B部门不是）。角色用于授权和审批人筛选。
- **用户组**：为了方便在流程中选人，还可支持用户组/团队的概念，把多个用户归为一组，如“项目审批小组”，然后流程指定整个组为候选人，由组内成员协作完成任务。

权限控制：采用基于角色的访问控制（RBAC）模型，结合必要的细粒度权限：
- 定义系统级角色，如**管理员**、**流程设计员**、**审批人**、**普通用户**等。管理员拥有全部权限（租户管理员只管自己租户的数据）；流程设计员可以新建发布流程定义；审批人可以处理分配给他的任务；普通用户只能发起申请、查看自己流程状态等。
- 功能权限方面，将系统功能点（如“创建流程”、“导入数据”、“管理租户”、“处理任务”、“查看报表”等）映射到角色。例如管理员和设计员能访问流程设计器模块，普通用户不能。
- 在多租户SaaS环境下，需确保不同租户用户即使角色相同也只能操作自己租户的数据。通过租户ID隔离，权限判定时会加上租户条件。
- **审批权限**：流程中具体到某个任务的审批权限通过流程定义的候选人规则来决定。但系统仍需二次校验——当某用户尝试处理一项任务时，后台确认此用户是该任务的受理人或所在候选组成员，否则拒绝操作。这保证即使恶意请求也无法越权审批他人任务。
- **组织权限**：某些管理操作可能限定在一定组织范围内，例如部门经理可以查看本部门的流程统计。这可以通过在权限校验时参考用户的部门属性来决定是否授权。

为实现权限控制，可以利用 .NET 常用的身份认证与授权框架。例如集成 **ASP.NET Core Identity** 用于用户身份管理和登录，Identity本身支持角色管理，可直接使用其Role模型。然后在应用中使用声明式授权 (Policy) 来校验角色和附加条件。

权限控制也涉及UI层面的控制：前端应根据当前用户角色，动态显示相应菜单和功能入口，避免越权操作。此外，重要操作（如流程跳转、撤回）可考虑加入**审核或双人确认**机制，以防止单人误用高权限操作。

最后，**审计和日志**也是权限体系的一部分。系统将记录所有敏感操作（如权限变更、流程定义发布、任务处理）的日志，包括操作者、时间和详情，以备安全审计。通过严格的权限划分和组织结构建模，系统能够确保**“用户只能访问与其角色相关的流程和数据”**¹⁴，维护数据完整性和保密性。

通过以上完整的规划设计，我们采用前后端分离架构，结合现代UI框架和成熟的后端技术栈，打造一个模块化、可扩展的审批流系统。该系统在流程建模、引擎执行、规则配置、数据管理、多端支持等方面都具备企业级能力，并注重安全与多租户部署需求，为用户提供媲美 Activiti 引擎的强大功能和良好的使用体验。¹⁵ ¹⁶

¹ ¹⁰ ¹¹ .Net Workflow SaaS, SaaS Workflow Automation software

<https://www.flowwright.com/saas-workflow-multi-tenant>

² ⁴ ⁵ .NET 开源工作流, .NET开源工作流, 工作流引擎, .NET 流程设计器, 工作流管理系统, 工作流开发框架, 企业级流程引擎组件, Slickflow

<https://www.slickflow.com/>

³ React Flow: Node-Based UIs in React

<https://reactflow.dev/>

⁶ ⁸ ¹² ¹³ ¹⁴ ¹⁵ ¹⁶ Workflow Management System Architecture Guide | Nected | Nected Blogs

<https://www.nected.ai/us/blog-us/workflow-management-system-architecture>

⁷ ⁹ NRules：基于.NET的高效规则引擎-CSDN博客

https://blog.csdn.net/gitblog_00825/article/details/143616342