Deeppaas零代码开发平台
技术白皮书

文 件 编 号

现 行 版 本 V1

总 页 数

编 制

校 对

2023 年 02 月

目录

1 概述

基于用户需求，梳理信息管理业务流程及数据管理需求，设计插件化、服务化、低代码开发的信息管理平台软件，实现前端界面、业务流程、后台服务、通信接口、底层数据的图形化配置、动态构建以及扩展管理。

2 需求分析

2.1 总体需求分析

基于概述中的描述，需要一套能够快速实现用户需求、快速响应用户需求变更的开发平台，实现开发资源共享以及经验积累。

经过分析，平台用户主要有三种：

开发者。开发者用平台快速实现最终用户的需求；

最终用户。即开发后产品的使用者；

泛 IT 开发人员。该人员非专业开发者，也非业务人员，但可能需要对平台做一些配置以快速迭代用户需求。

整个平台的需求来源主要有最终用户以及开发者，同时在设计时满足一定的易用性要求以满足泛 IT 开发人员的需求。

3 平台功能设计

3.1 平台技术架构

图 1 平台技术架构

备注：

1、接口目前只支持 rest，其他类型的接口或者协议需要定制开发；

2、页面配置完成支持生成微信小程序代码，但是目前暂时没有实现；

3、移动端采用 react Native 方案。

表格 1 架构分层概述表

层级	模块	描述
运行层	端	在 PC、平板电脑、智能手机上均可运行。
	运行形式	支持以指定网址，通过网页形式进行使用 支持微信小程序，支持 app 形式使用
设计层	数据源/数据表	支持将指定关系型数据库直接使用，包括 mysql、sqlserver、oracle 等，可将数据库和数据库中的表直接通过配置后在平台进行使用。目前暂时只支持 MySQL，其

		他类型数据库需要进行适配。
	规则表达式	以表达式形式进行部分逻辑处理，如判断表单字段 A 是否大于字段 B，返回布尔类型，表达式即为：A>B
	界面设置	通过配置方式进行直接的使用界面维护，颗粒度自控，可直接配置一个从无到有的页面，包括页面内的输入信息、展示信息，并可配置相应页面逻辑，如保存输入表单内容，按钮点击审批处理、按钮点击触发页面信息展示等等。 可配置界面的路由规则，自定义登录界面、系统首页等。
	规则设置	规则是与事件向对应的事件发生后做什么动作，事件可以是页面某个按钮的点击事件，也可以是某个审批流程完成时触发的流程审批完成事件，事件在各模块都有相应的固定事件。规则则是一系列的动作组成的一整套处理逻辑，比如界面一个按钮的点击事件绑定了一个规则，这个规则第一个动作就是保存表单数据，第二个动作则是界面弹出保存成功并且返回到某个页面。
接口层	接口设置	为便于与其他系统集成，无需代码编写。可将其他系统接口进行配置，配置后即可在本系统直接使用。另外可在本系统内通过规则配置出所需逻辑接口，比如通过一个 ID 获取某条业务数据，那面只需配置一个接口，接口参数为 ID，然后在接口规则内通过动作获取指定数据模型数据并返回即可，无需任何代码编写。

		接口集成支持 IP 白名单、密钥签名、oauth2 的安全策略，保证接口调用安全。
设置层	基础设置	基础设置包括：首页设置、角色设置、权限设置、全局属性设置等基础设置，比如登录后的首页如何展示，即通过首页设置进行设置，使用人员可通过在系统内配置即可完成系统首页界面的任何细节控制，无需编写任何代码。
	组织结构	组织结构是平台便于用户使用儿提供的一套通用且业务场景覆盖性较全面的组织结构设置，如部门、岗位、人员、角色，以及可能设计到的个性化的部门属性等等。
存储层	数据库	平台本身支持指定关系型数据库包括 mysql、sqlserver、oracle 等，此外可将第三方数据库和数据库中的表直接通过配置后在平台进行使用，第三方数据库使用时权限受第三方授权账号影响，如未授权写权限则只能读取数据。
	文件存储	平台支持本地存储（即本地文件夹）和阿里云 OSS 对象存储两种方式进行文件存储，如需其他存储可通过适配集成开放进行集成

3.2 平台软硬件环境

表格 2 系统软件环境

序号	项目	说明
1	操作系统	Windows 或 LINUX 操作系统或麒麟（可以适配）

2	后端开发语言	JAVA
3	前端开发语言	javascript
4	托管代码编程模型框架	springboot、react
5	控件组	低代码开发平台、EChart、AntDesign
6	数据库	MySQL5.7 及以上版本

表格 3 系统硬件环境要求

序号	项目	说明
1	硬盘剩余空间	100G 以上
2	内存	2G 以上
3	CPU	双核 2G hz 以上
4	显卡	1GB 以上显存，分辨率 1680×1050，32 位彩色

3.3 前端界面设计

3.3.1 界面功能描述

平台采用前后端分离的架构模式。前端界面的主要功能是以图形化的形式，通过拖拽基础组件、配置组件属性等方式实现页面的配置文件，目前配置文件以 JSON 格式存储。平台通过对页面配置文件的解析、渲染实现页面的展示。

一个基础的页面是由页面基础控件、画布、属性三大核心部分组成。控件的样式属性满足 CSS 要求，可以通过可视化的方式配置。

图 2 界面配置示例

配置完成后的界面支持较新版本的 Chrome、FireFox、EDGE 等，不支持比较老旧的 IE 浏览器。PC 端和移动端采用相同的页面搭建方式。

3.3.2 前端界面控件

前端界面控件包含布局控件、操作控件、显示控件、输入输出控件、图表控件等。理论上，所有界面都可以通过基础控件的组合生成。

控件类型	序号	控件名称	作用
布局控件	1	水平布局	布局内控件水平方向排列
	2	垂直布局	布局内控件垂直方向排列
	3	普通容器	多个控件组合，控制控件的位置
	4	子页面容器	在父页面中显示子页面
操作控件	1	按钮	点击可产生事件
	2	分页器	点击可进行数据列表的翻页
显示控件	1	图标	显示一个小图标
	2	头像	显示头像
	3	矩形块	显示一个矩形块
	4	iFrame	套用其他网页

	5	视频	播放视频
	6	音频	播放音频
	7	富文本预览	显示富文本预览
	8	消息时间线	按时间显示序列
	9	消息面板	显示消息
	10	文字	显示文字信息
	11	超链接	显示超链接
	12	控件分组	控件分组，批量操作禁用等属性
输入输出控件	1	标签	设置或者显示标签
	2	数据表格	显示数据列表，可以自定义表头、表体
	3	列表	循环显示指定控件
	4	注入代码	可以自定义前端代码
	5	输入控件组	带标签的单行输入
	6	表单	自动生成 4 个输入控件组以及按钮
	7	单行输入	输入单行文本
	8	多行输入	输入多行文本
	9	下拉单选	
	10	单选按钮组	
	11	多选按钮组	
	12	下拉多选	
	13	开关切换	
	14	数值滑块	
	15	数字输入	
	16	日期选择	
	17	图标选择	
	18	文件上传	
	19	选择文件	
	20	复选框	
	21	手写板	手写签名

	22	富文本	
图表控件	1	柱状图	
	2	条形图	
	3	饼图	
	4	折线图	
	5	多折线图	
	6	面积图	
	7	堆积面积图	
	8	词云	

3.3.3 界面布局与控制-网页

网页是带 URL 的页面，可以在浏览器地址栏通过网址访问。网页支持以参数的形式进行数据的传递和调用。

用户生成网页的过程即是通过拖拽控件生成界面的过程。理论上，通过合理的规划，平台能够支持生成 UI 级的页面。

通过控件属性窗口，可以设置页面大小、颜色、布局等。

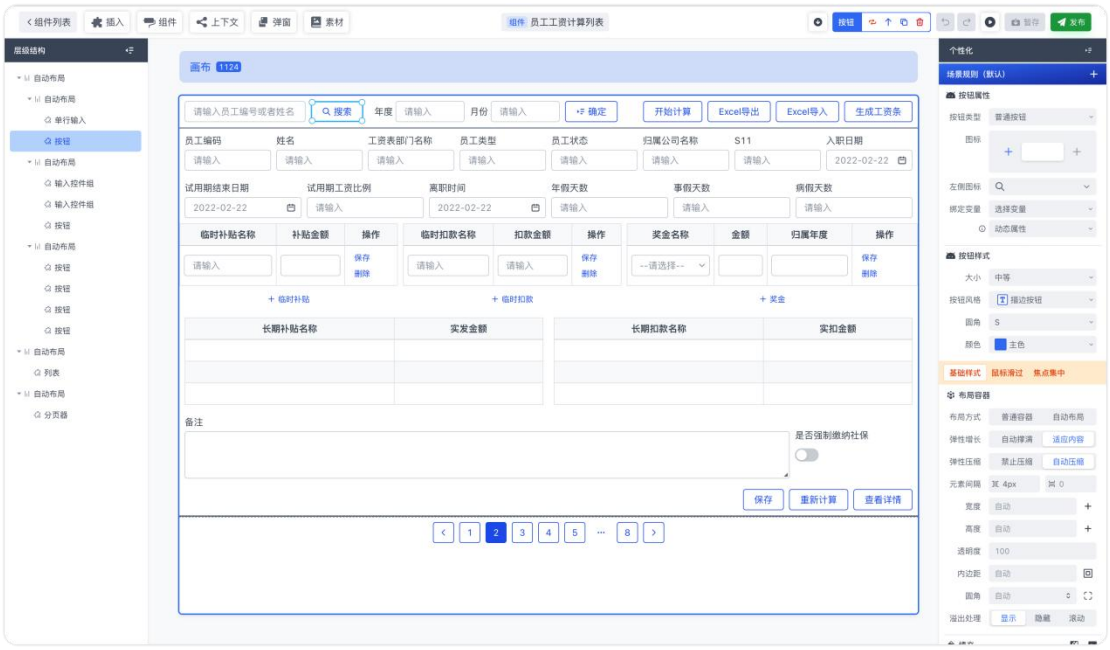


图 3 页面配置示例图

3.3.4 界面布局与控制-组件（子表单）

组件是一组控件的集合。组件的参数可以定义成变量或者规则，并可以在其他页面中调用。因为在不同的调用页面中传入了不同的变量或者规则，因此同一组件可以在不同的引用中显示出不同的效果，从而达到组件复用的目的。

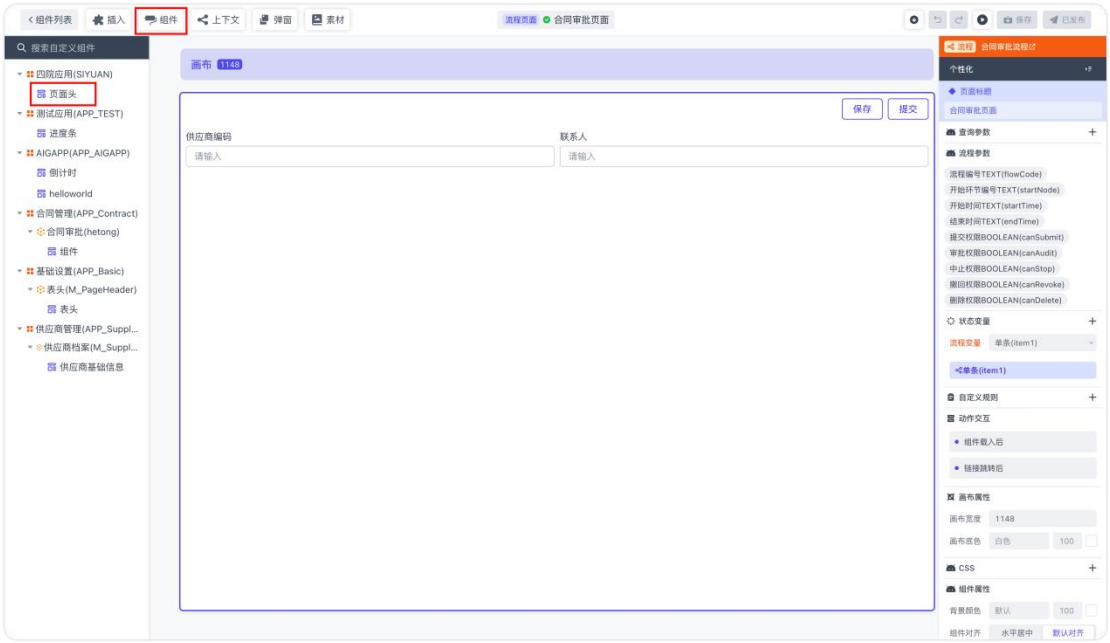


图 4 插入组件示例

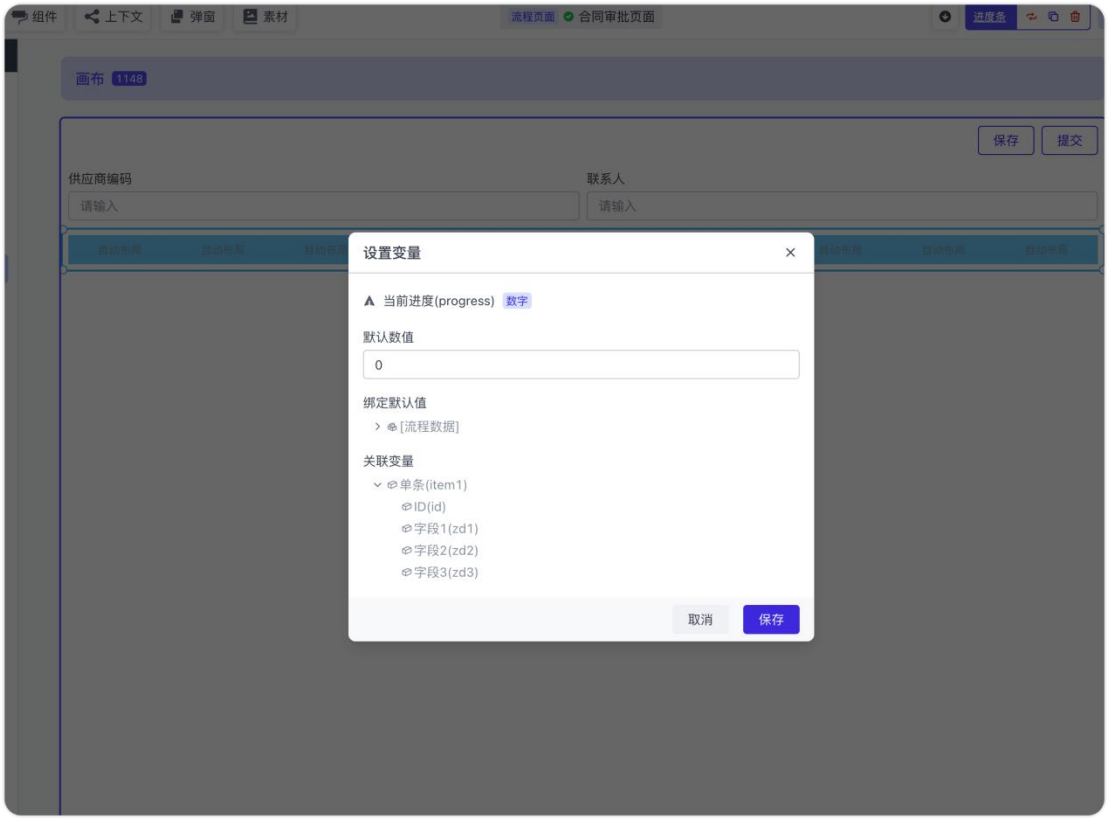


图 5 组件变量和当前页面变量绑定

3.3.5 界面布局与控制-上下文

数据页面是一组公共变量以及公共函数的集成。对于系统的全局变量和公共函数可以在此定义。数据上下文提供了一个引用端的数据监听和刷新的机制。

当全局变量变化时，所有引用此变量的页面数据都会同步变化。

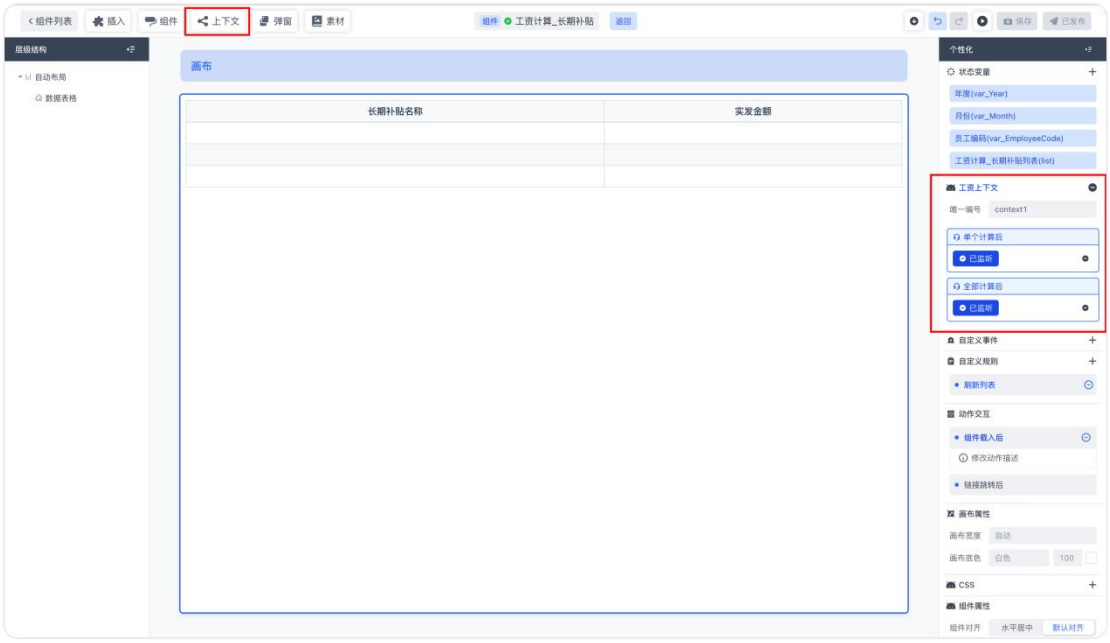


图 6 数据上下文示例

3.3.6 URL 路由与内置固定页面

每一个网页类型的页面都可以自定义 URL，URL 路由之间存在父子关系。子路由界面会遵循父路由界面的界面设计，从而实现在同一个父页面进行不同子页面的跳转。

系统内置几个固定路由，固定路由的界面由用户自由定义。固定路由界面列表如下：

表格 4 系统默认路由

序号	页面名称	路由地址	说明
1	系统首页	/	系统默认首页
2	系统登陆页	/login	系统登录页。当设置了此路由时，系统默认登陆页将由此页面替代

3.3.7 界面场景模式（动态样式）

页面控件支持场景模式的定义，在不同的场景模式下显示不同的样式。页面控件样式同时支持鼠标的相关操作，如改变鼠标指针形状、鼠标滑过时改变样式等。

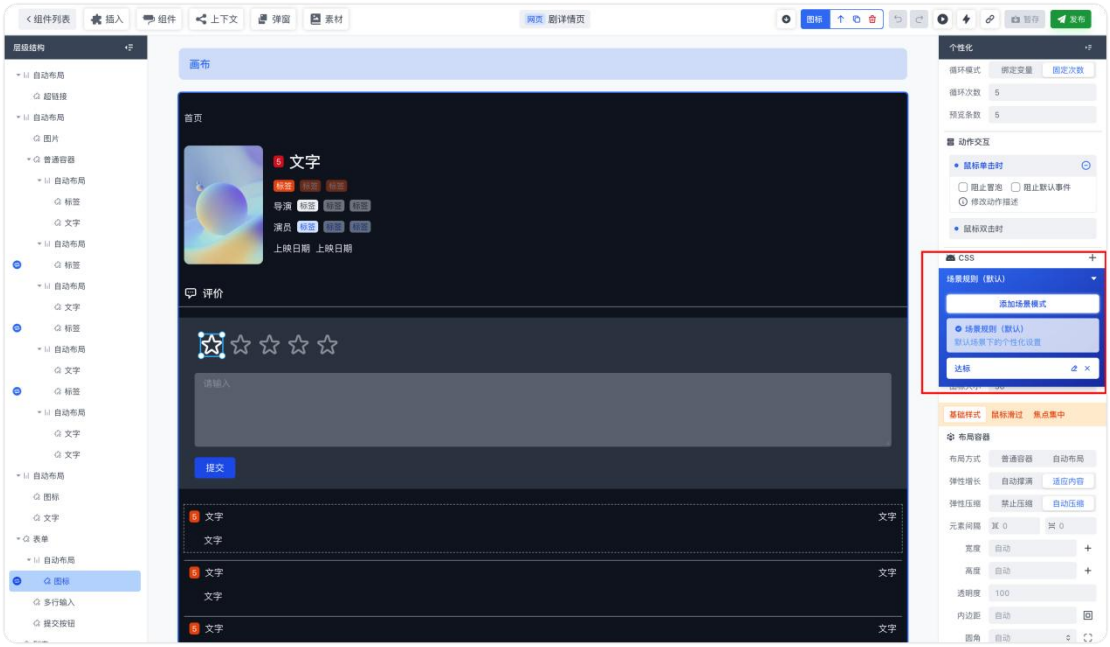


图 7 场景模式示例

3.3.8 控件事件

页面控件支持对应的事件以影响用户动作。事件定义了动作触发的时机。

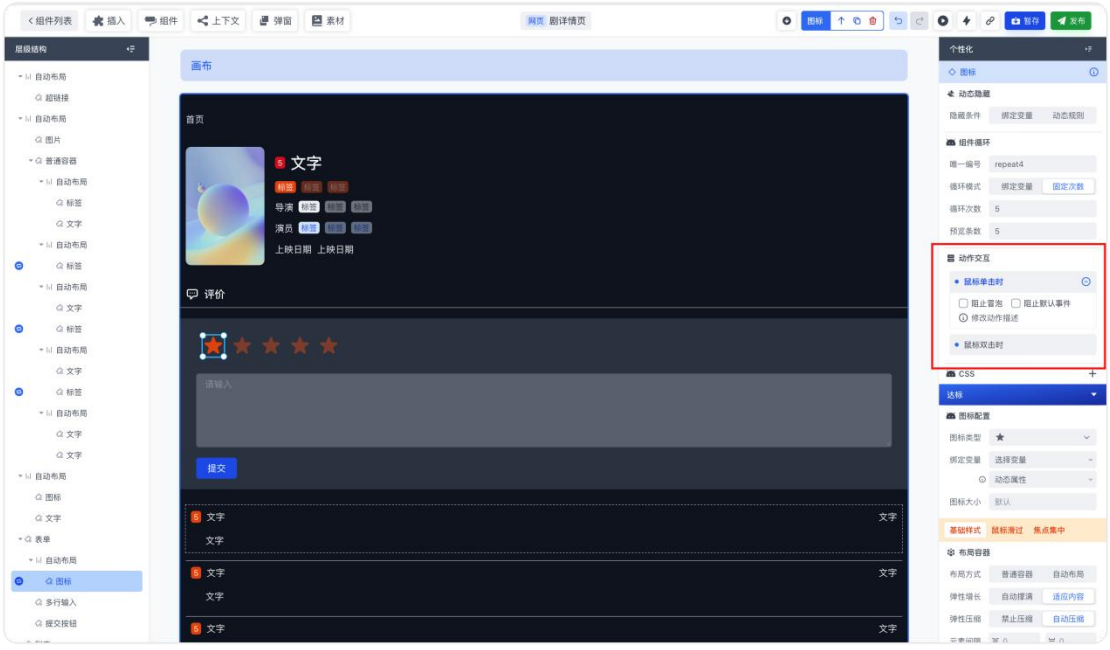


图 8 控件事件示例

3.3.9 事件规则

事件规则确定了事件要执行的一系列动作的逻辑组合。事件规则引擎以可视化的方式提供了基础的变量操作和动作执行流程。从而实现无需编写代码即可实现业务逻辑的功能。

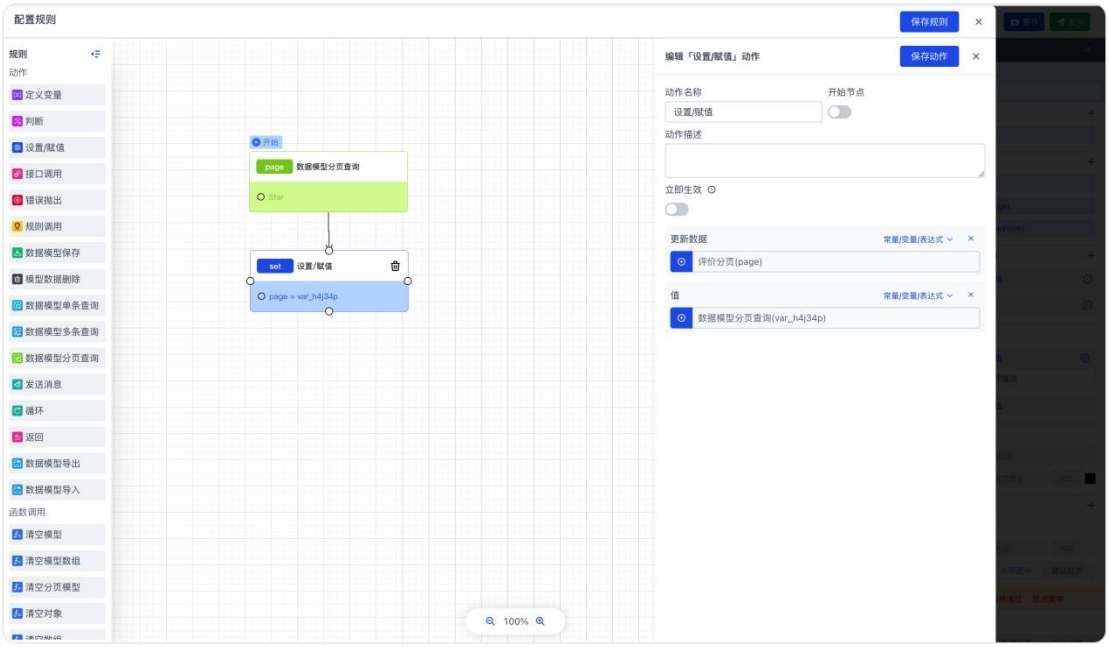


图 9 规则引擎示例

规则支持自定义，并支持在对应的作用范围内调用。

3.3.10 前端业务表达式

业务表达式提供了类似 Excel 公式的操作数据的方式。业务表达式可以实现数据的计算、逻辑的判断等功能。

同时业务表达式支持高级扩展功能，可以利用 JavaScript 编写高级的脚本，从而实现更加复杂的业务逻辑。

图 10 业务表达式示例

3.4 流程管理

3.4.1 流程配置可视化

平台提供流程配置可视化功能。通过托拉拽方式实现流程图的配置。通过流程和页面分离的架构，流程管理可以实现对审批流和业务流两种业务方式的处理。

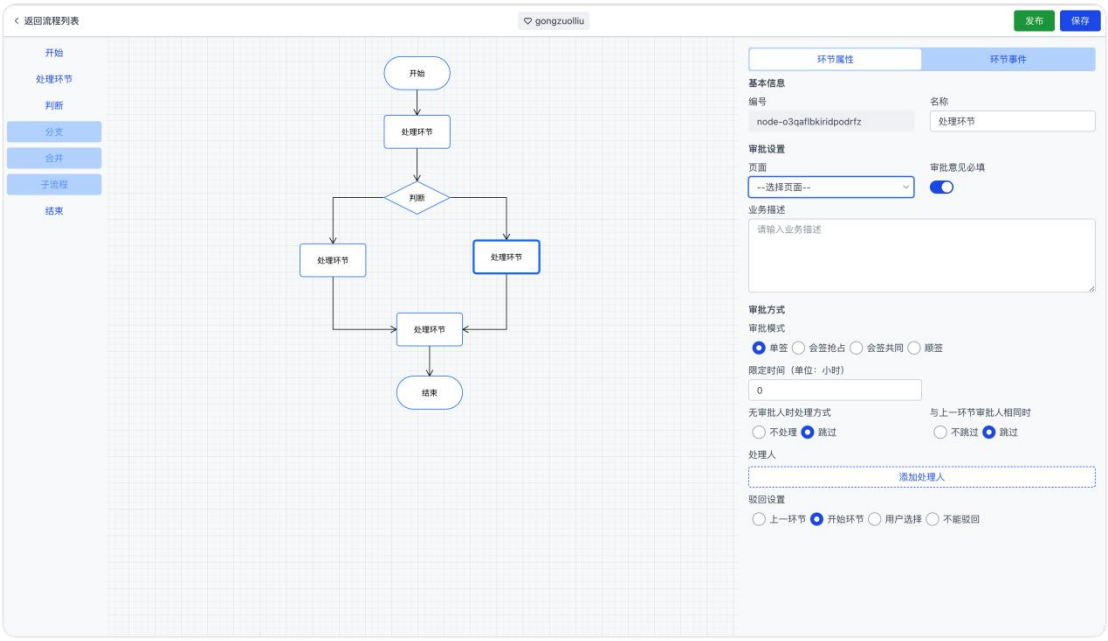


图 11 流程可视化配置示例

3.4.2 流程环节类型以及配置

平台流程管理提供了开始结束环节、处理环节、判断环节、分支环节、合并环节、子流程环节等类型的节点。

开始环节：流程入口环节；

结束环节：流程结束环节。结束环节对应页面为流程的默认页面；

处理环节：为流程的正常处理环节；

判断环节：可根据业务表达式判断流程走向；

分支环节：几个环节并行；

合并环节：并行环节合并；

子流程环节：可以由主流程进入子流程。

3.4.3 审批设置

业务处理人支持单签、抢占会签、全部会签、顺签等处理方式；

处理人可根据业务规则确定处理人类型；

支持环节的驳回设置等。


环节属性
环节事件
基本信息
编号
node-o3qaflbkiridpodrfz
名称
处理环节
审批设置
页面
--选择页面--
审批意见必填
业务描述
请输入业务描述
审批方式
审批模式
单签
会签抢占
会签共同
顺签
限定时间（单位：小时）
0
无审批人时处理方式
不处理
跳过
与上一环节
不跳过
处理人
组织结构(部门/岗位)
通用岗位
用户
组织结构(用户)
表单字段(部门)
表单字段(人员)
角色
生效条件
true
生效范围
类型
范围
+
添加处理人
驳回设置
上一环节
开始环节
用户选择
不能驳回

图 12 环节属性设置

3.4.4 环节事件

流程审批过程中支持对流程以及环节事件的处理。在流程流转过程中支持数据更新等一系列后端规则操作。

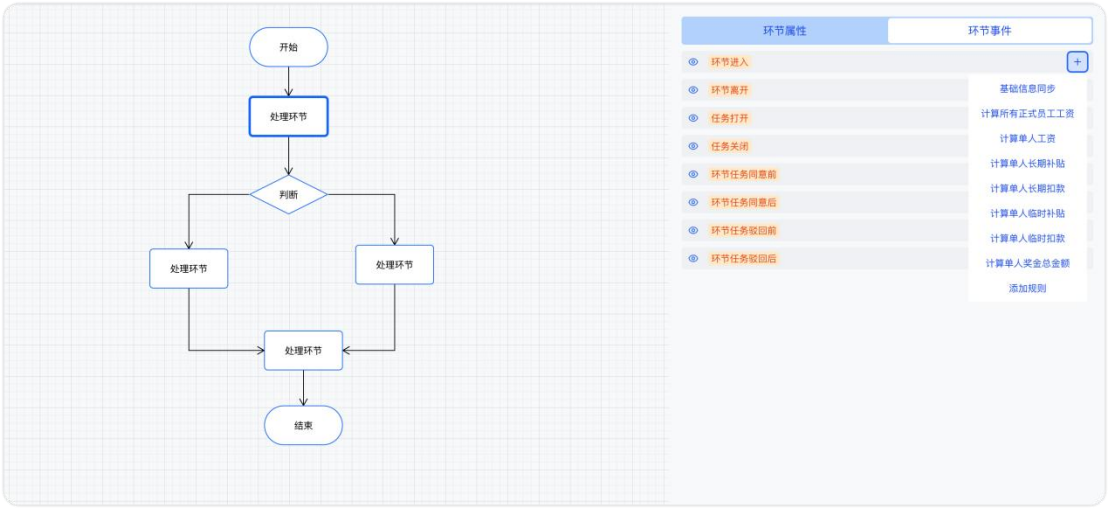


图 13 环节事件设置

3.4.5 组织结构

组织结构是公司管理架构的表现形式，同时公司的各种业务架构也通过组织结构进行配置和管理。通常，流程审批的审批人与组织结构密切相关。平台组织结构特征：

1、支持集团管控与分级授权；

2、支持通用岗位设置；

3、支持部门、岗位、人员属性扩展；

4、支持部门级别设置；

5、支持一人多岗；支持默认岗位设置；

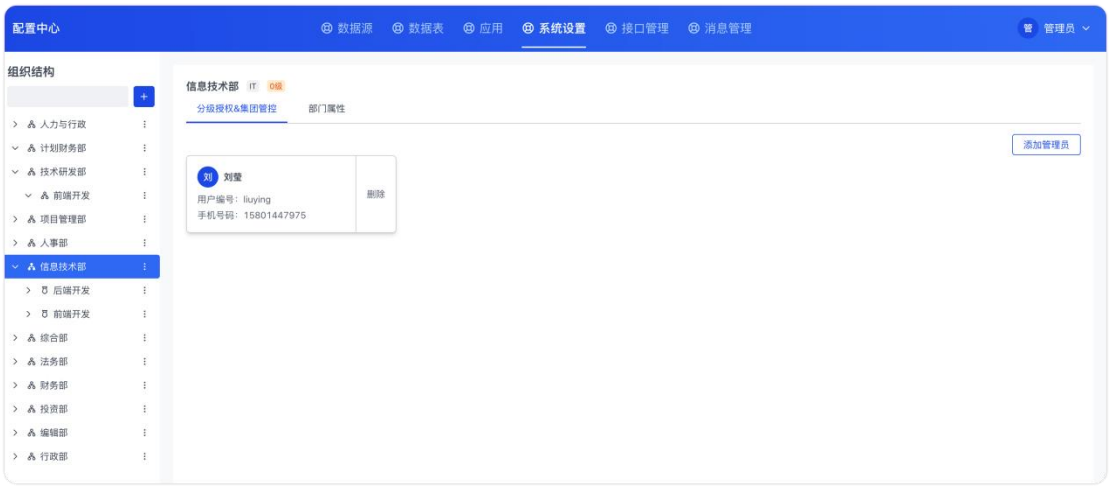


图 14 组织结构示例

3.5 后台服务

3.5.1 后台规则引擎

平台提供后端业务规则服务功能。后端规则是指在服务器端运行的规则。其表现形式和前端规则一样，也是通过规则引擎以及表达式的方式。后端服务能够实现的功能包括数据传输、数据转换、数据更新、接口调用等。

图 15 后端规则示例

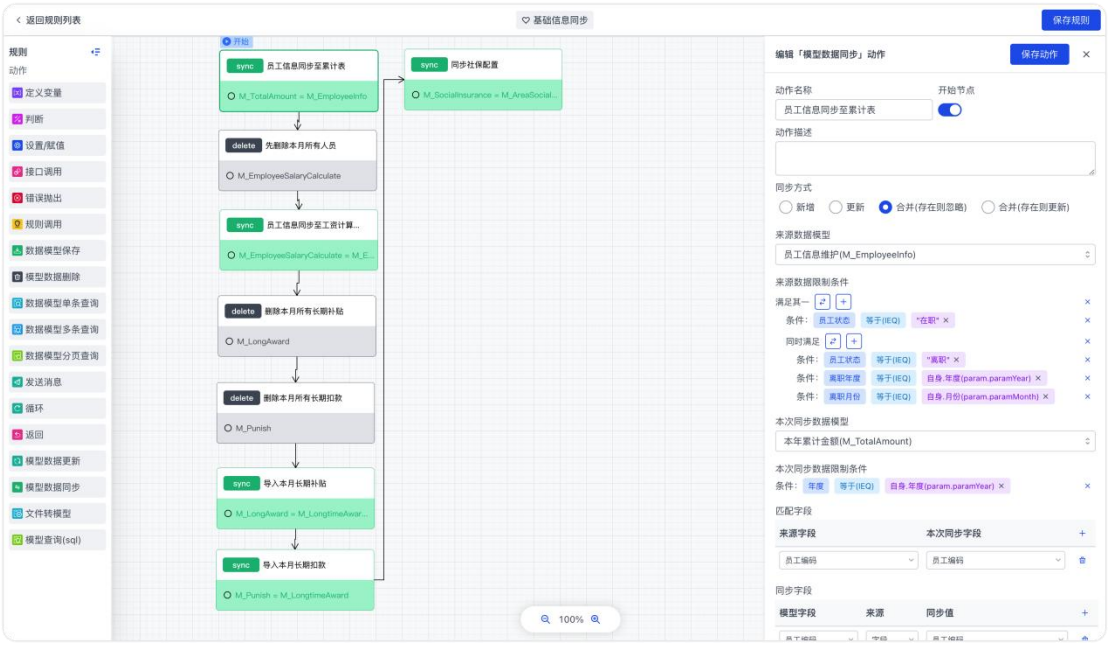


图 16 后端规则引擎示例

3.5.2 后台监控服务

平台后端提供专门的工具以监事系统状态。软件监控为一个独立模块，可提供如下功能。

- 显示健康状态及详细信息，如 JVM 和内存指标、数据源指标、缓存指标
- 跟踪并下载日志文件
- 查看 jvm 系统-和环境属性
- 查看 Spring 启动配置属性方便 loglevel 管理
- 查看线程转储视图 http-traces
- 查看 http 端点查看计划任务
- 查看和删除活动会话(使用 spring-session)
- 状态更改通知(通过电子邮件、Slack、Hipchat…)
- 状态变化的事件日志(非持久性)
- 下载 heapdump
- 查看 Spring Boot 配置属性
- 支持 Spring Cloud 的环境端点和刷新端点
- 易用的日志级别管理
- 与 JMX-beans 交互
- 查看线程转储

- 查看 http 跟踪
- 查看 auditevents
- 查看 http-endpoints
- 查看计划任务
- 查看和删除活动会话（使用 Spring Session ）
- 查看 Flyway/Liquibase 数据库迁移
- 状态变更通知
- 状态更改的事件日志（非持久化）

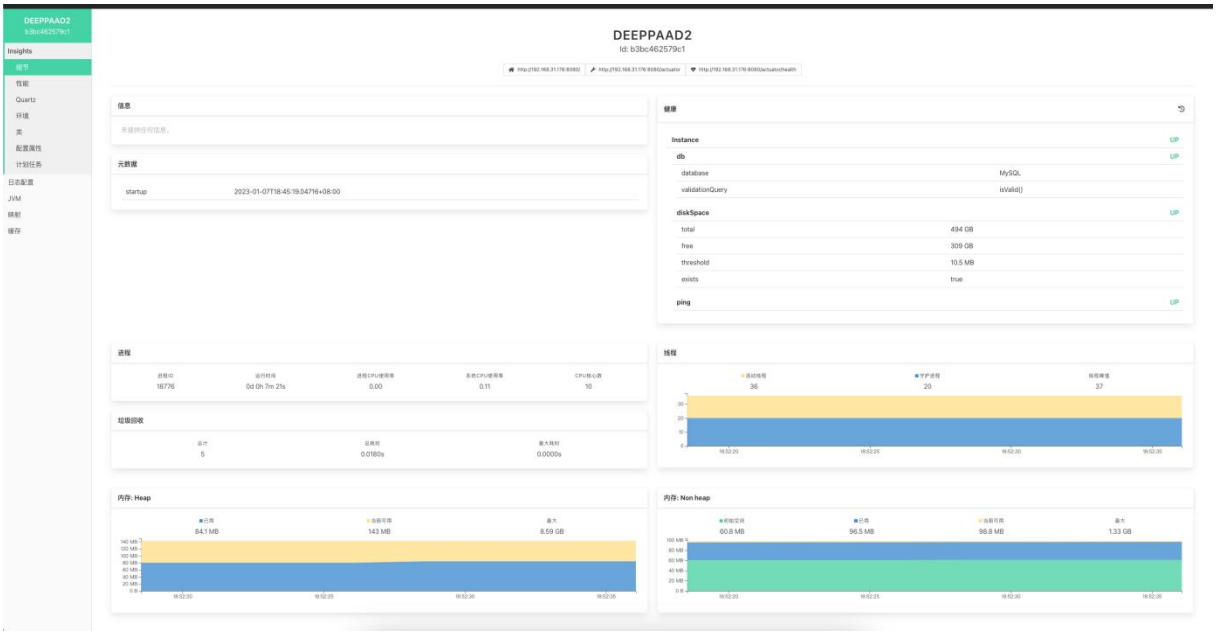


图 17 服务器状态监控示例

3.6 接口管理

接口管理提供了平台与其他第三方系统和硬件对接的基础能力。通过接口管理功能，第三方平台或者硬件可以调用平台的功能或者数据，平台也可以通过接口调用其他第三方的数据以及功能。

目前平台默认支持 webservice 的 rest 格式的接口，其他类型的接口需要根据实际协议进行个性化定制。

3.6.1 计算模型

计算模型是定义接口数据的元数据模型。计算模型可以在平台中当做数据

模型使用，实现接口数据和平台数据的统一。

计算模型也可以作为平台动作规则等的参数格式定义。

图 18 计算模型定义示例

3.6.2 平台接口

平台接口是平台为其他第三方系统提供的接口，第三方系统可以通过平台提供的接口来调用平台的服务和数据。平台接口以 rest 方式提供服务。

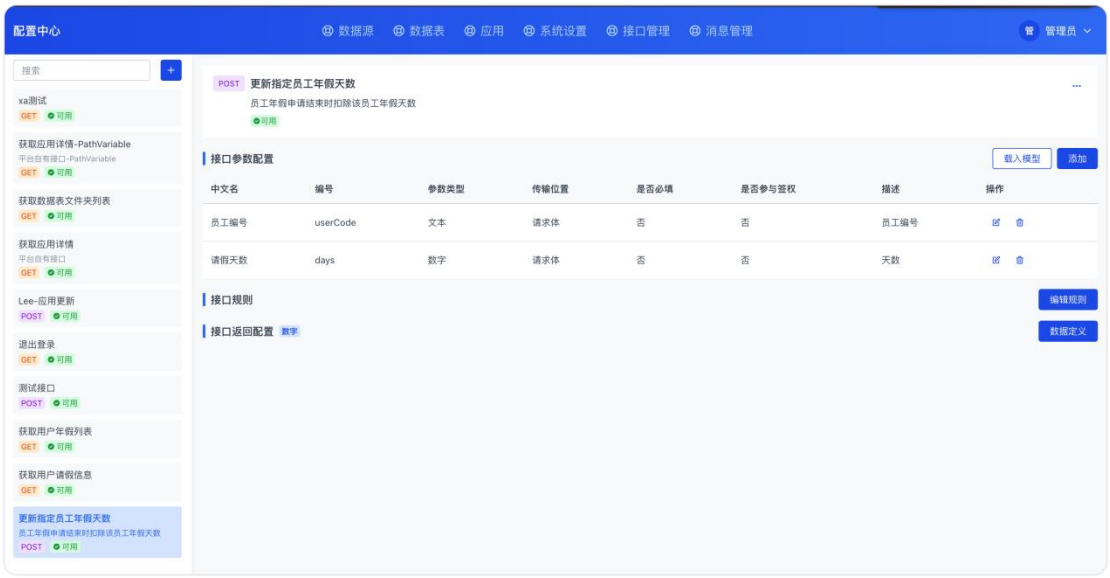


图 19 平台接口示例

3.6.3 第三方服务

平台内置接口可以调用第三方服务，通过和计算模型的结合，可以在平台内部像调用自己本身的接口一样来调用第三方服务。系统默认支持基于 TCP 的的 rest 服务，若要拓展其他类型接口或者协议，需进行定制化开发。

图 20 第三方接口示例

3.7 底层数据设计

3.7.1 数据源设置

平台在架构上支持多种类型的数据源，诸如 MySQL，SQL Server，Oracle 等。在平台中支持对多异构数据源数据同时编辑、使用。目前我们暂时只支持 MySQL5.7 以及以上版本，增加新类额的数据源的时候需要提前做数据源的适配。

图 21 新增数据源

3.7.2 数据表设置

数据表是用来存储数据或者展示报表的基础。数据表可以由系统的数据表建模功能动态生成，也可以反向加载数据中已有的物理表或者视图，也可使用自定义 SQL 的方式。视图或者自定义 SQL 不可对数据进行编辑。

数据表设置支持动态增加或者删除字段，对于备选类型的字段，可设置字段的 key-display 属性。

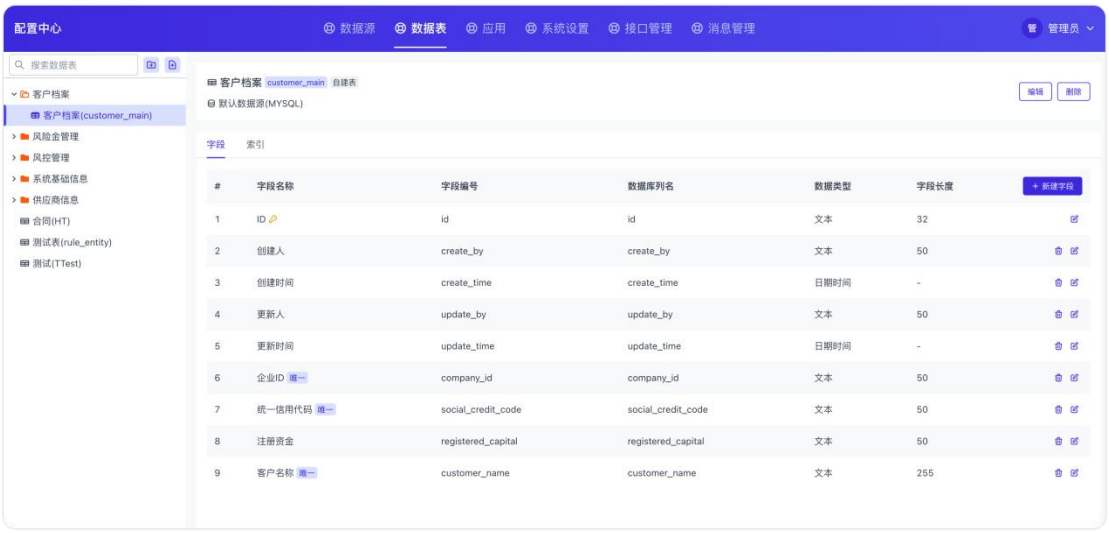


图 22 数据表设置界面 数据表元数据管理

支持在平台中创建删除索引等操作。

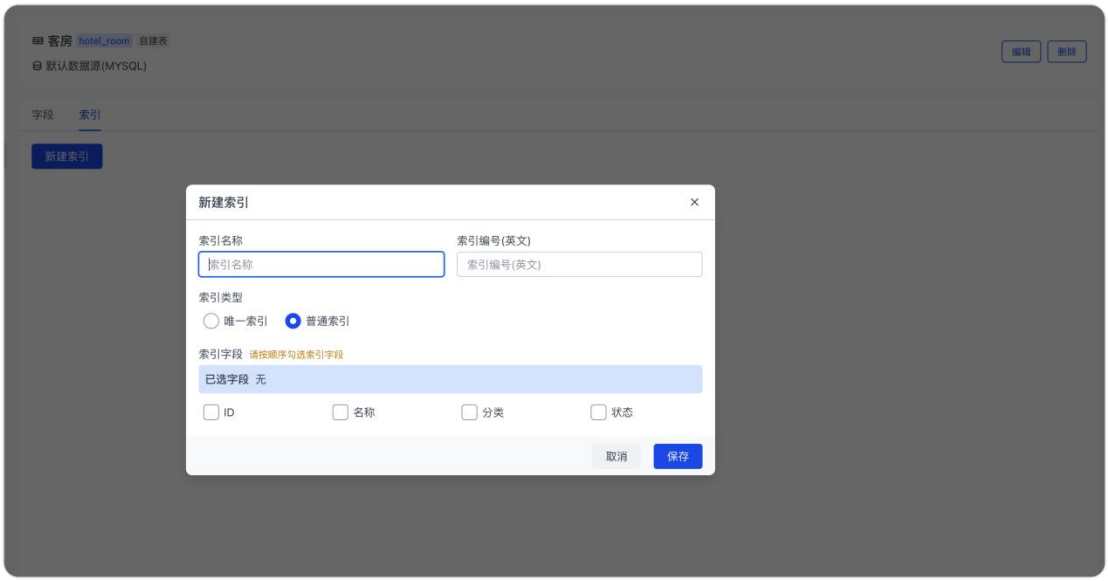


图 23 创建索引示例

3.7.3 文件存储

平台文件存储支持本地文件存储，也支持对接其他第三方 OSS 服务。

图 24 文件组配置示例

4　平台开发部署结构

4.1　工程结构

整个工程以前后端分离方式进行开发，前后端工程结构如下。

4.1.1 前端工程结构

前端工程是一个 monorepo 架构的工程，基于 React.js + Typescript 开发，部分依赖来自第三方开源包，且支持免费商用。UI 部分采用内部开发的运行时框架，支持全局更换主题风格。

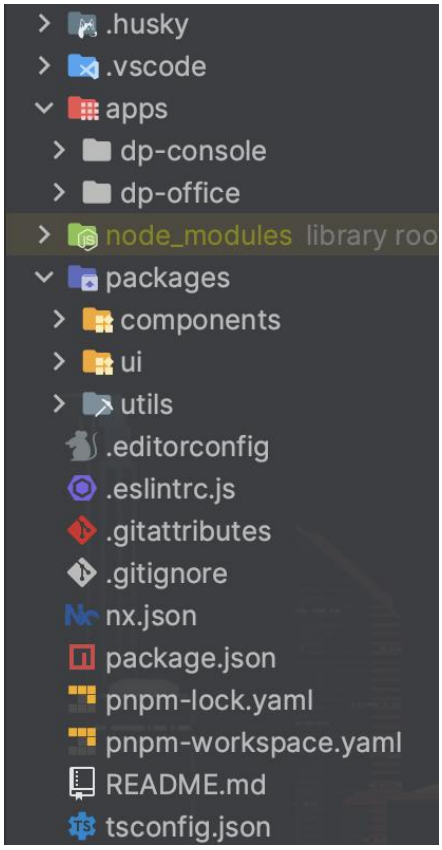


图 25 前端工程示例

其中子项目分为应用包(apps/*)和公用依赖包(packages/*)

packages/utils：整个项目公用工具函数和配置，包括规则函数、接口工具、自定义 React Hooks 等。

packages/ui：项目 UI 界面风格工具，和内置组件样式定义工具，支持丰富的颜色主题和响应式媒体查询。

packages/components：项目内置组件包，包括表单控件、按钮、图标、表

格列表、图表、弹窗提示反馈、视图媒体和布局，具体组件代码文件见下图。

apps/dp-console：平台项目核心代码，包括数据表、应用模型、组织结构、用户权限、流程定义、接口平台、消息、前端规则配置和解析引擎、自定义组件配置和渲染等。具体代码结构见下图。

apps/dp-office：平台官网

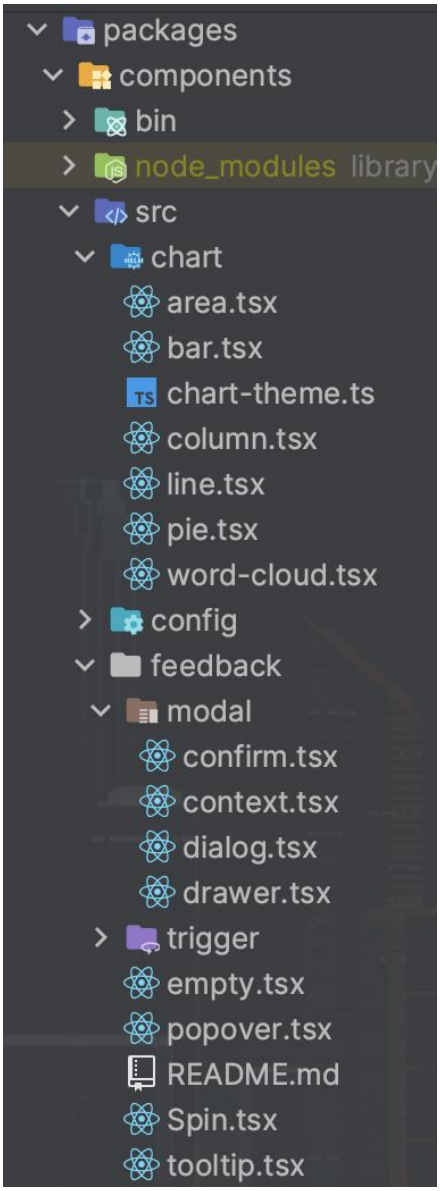


图 26 内置组件代码文件结构

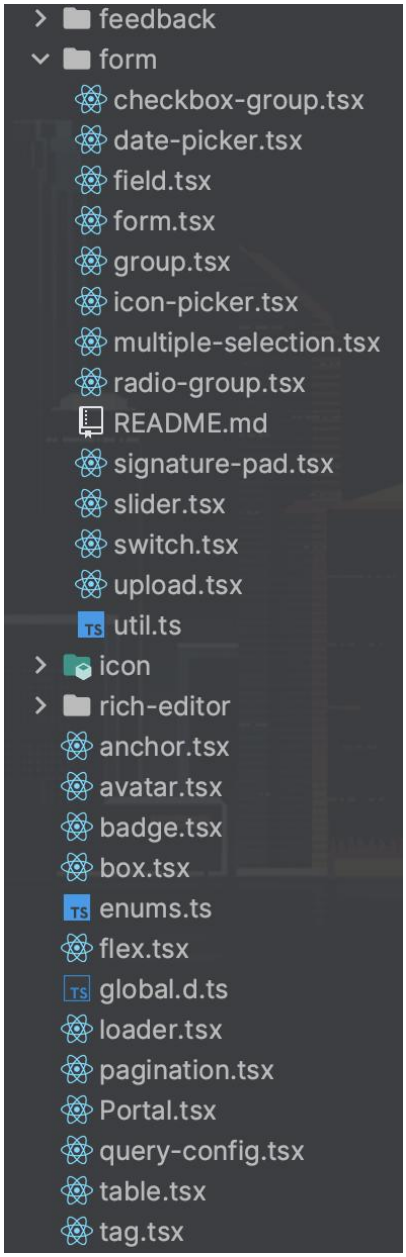


图 27 内置组件代码文件结构

图 28 平台工程核心业务代码结构截图

4.1.2 后端工程结构

后端工程基于 springboot 进行开发，使用 maven 进行构建管理，主要工程

拆分如下。

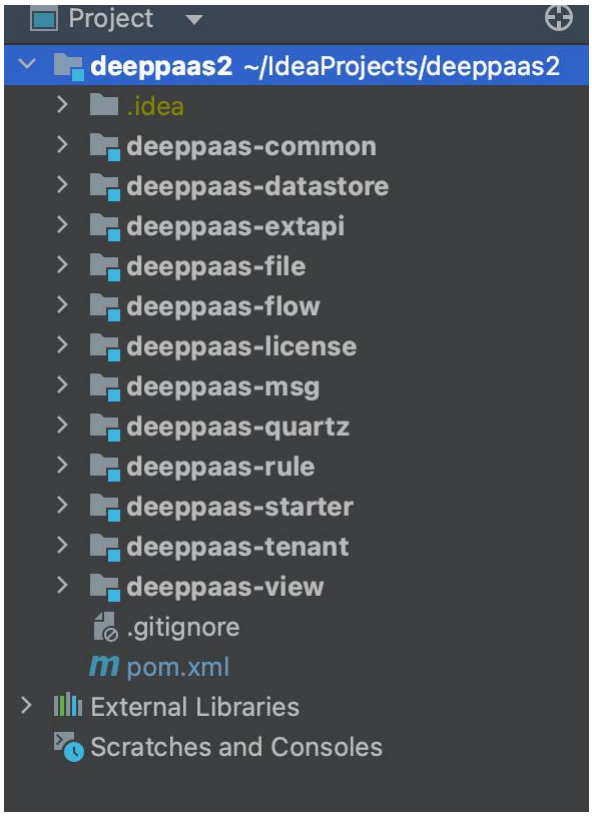


图 29 后端工程结构

工程拆分以 common、api、biz 进行区分，如

xxx-common：通常为公共代码工程，包括不限于常量、基类、接口、工具类等等；

xxx-api：通常为服务接口工程，主要为对外提供调用服务的接口以及接口参数、返回值数据类型定义；

xxx-biz：通常为业务实现工程，包括从数据库、业务逻辑、服务接口实现、前端交互接口实现等等。

deeppaas2 具体工程拆分如下表：

表格 5 工程简介表

序号	工程模块	备注
1	deeppaas-common	deeppaas2 通用公共代码工程
2	deeppaas-datastore	deeppaas2 数据仓库父类工程
3	deeppaas-datastore-api	deeppaas2 数据仓库服务接口工程

4	deeppaas-datastore-common	deeppaas2 数据仓库公共代码工程
5	deeppaas-datastore-biz	deeppaas2 数据仓库业务实现工程
6	deeppaas-extapi	deeppaas2 接口集成父类工程
7	deeppaas-extapi-api	deeppaas2 接口集成服务接口工程。
8	deeppaas-exapi-biz	deeppaas2 接口集成业务实现工程
9	deeppaas-file	deeppaas2 文件服务父类工程
10	deeppaas-file-api	deeppaas2 文件服务接口工程
11	deeppaas-file-biz	deeppaas2 文件服务业务逻辑实现工程
12	deeppaas-flow	deeppaas2 流程模块父类工程
13	deeppaas-flow-api	deeppaas2 流程服务接口工程
14	deeppaas-flow-common	deeppaas2 流程模块公共代码工程
15	deeppaas-flow-biz	deeppaas2 流程模块业务逻辑实现工程
16	deeppaas-license	deeppaas2 授权模块父类工程
17	deeppaas-license-api	deeppaas2 授权服务接口工程
18	deeppaas-license-open	deeppaas2 授权对应开放授权实现工程
19	deeppaas-license-single	deeppaas2 授权对应单机授权实现工程
20	deeppaas-msg	deeppaas2 消息模块父类工程
21	deeppaas-msg-api	deeppaas2 消息服务接口工程
22	deeppaas-msg-biz	deeppaas2 消息业务实现工程
23	deeppaas-rule	deeppaas2 规则动作父类工程
24	deeppaas-rule-api	deeppaas2 规则动作服务接口工程
25	deeppaas-rule-biz	deeppaas2 规则动作业务实现工程
26	deeppaas-tenant	deeppaas2 租户系统父类工程
27	deeppaas-tenant-api	deeppaas2 租户服务接口工程
28	deeppaas-tenant-biz	deeppaas2 租户业务实现工程
29	deeppaas-view	deeppaas2 界面组件父类工程
30	deeppaas-view-api	deeppaas2 界面组件服务接口工程
31	deeppaas-view-biz	deeppaas2 界面组件业务逻辑实现工程

32	deeppaas-starter	deeppaas2 单机启动入口工程

4.2 程序设计

4.2.1 数据仓库设计

4.2.1.1 数据仓库表定义

表格 6 数据仓库表定义

表	备注
datastore_source	数据源
datastore_table	数据表
datastore_table_field	数据表字段
datastore_table_field_option	数据表字段备选
datastore_index	数据表索引
datastore_entity	数据模型
datastore_entity_field	数据模型字段
datastore_entity_field_option	数据模型字段备选
datastore_entity_join	数据模型连接
datastore_entity_join_match	数据模型连接匹配

4.2.1.2 数据仓库接口

- DataStoreDataClient:提供数据仓库内数据模型中数据条目的增删查改操作；
- DataStoreEntityClient:提供数据仓库内数据模型定义的增删查改操作。

4.2.1.3 数据仓库集成扩展

数据库类型扩展：除平台以提供数据库外，如需增加特殊数据库，可能增加 DBDialect 中的类型进行适配，比如扩展北大金仓数据库 KINGBASE 支持，则增加对应 Dialect 实现即可。新增数据库类型需继承基类 DataStoreDialect 实现一个新的数据库适配器新，接口如下图：

```java
public interface DataStoreDialect {

    String dbType();

    String limitOffset(Pageable pageable);

    String backtick(String name);

    String createSelfTableSql(String schema, String tableName, String owner);

    String createSelfTableColumnSql(String schema, String tableName, String fieldCode, String columnName, String data

    String modifySelfTableColumnSql(String schema, String tableName, String fieldCode, String columnName, String data

    String createTableIndexSql(String schema, String tableName, String indexName, TableIndexType indexType, List<Stri

    String deleteTableIndexSql(String schema, String tableName, String indexName);

    String conditionSql(String alias, String columnName, DataStoreOperator operator, String paramMarker);
}
```

图 30 数据库类型扩展示例

实现后在可在启动时在自定义继承 AbstractDataStoreConfiguration 中进行新的注册即可，如下

```java
@Configuration
public class DataStoreConfig extends AbstractDataStoreConfiguration {

    @Override
    public List<DataStoreDialect> userDataStoreDialects() {
        List<DataStoreDialect> dialects = new ArrayList<>();
        dialects.add(new KingbaseDataStoreDialect());
        return dialects;
    }
}
```

图 31 数据库扩展注册

4.2.2 界面引擎设计

4.2.2.1 界面引擎表定义

表格 7 界面引擎表定义

表	备注
view_component	界面组件表
deeppaas_file_resource	界面系统使用文件资源表

界面引擎中界面组件采用 json 方式进行存储。

4.2.2.2 界面引擎接口

ComponentClient：提供界面实体的增删查改。

4.2.2.3 界面引擎集成扩展

新开发内置组件，需要进行以下步骤：

1. 在 packages/components 包中创建新的组件代码文件(React 组件)，并向外暴露可配置属性。(组件开发)
2. 在 apps/dp-console/src/builtin/index.ts 中将新组件定义仿照其他组件格式加入到组件列表中。(加入可选组件列表)
3. 在 apps/dp-console/src/routes/settings/design/config/properties/builtin-component-configure.tsx 中添加新组件的属性配置组件。(组件属性配置界面)
4. 在 apps/dp-console/src/routes/settings/design/window/dnd-item.tsx 中仿照其他组件，将新组件的实例引用到内置组件列表中。(拖拉拽界面预览)
5. 在 apps/dp-console/src/routes/page/composer-item.tsx 中仿照其他组件，将新组件的实例加入到内置组件渲染列表中。(最终线上组件渲染)

4.2.3 流程引擎设计

软件流程引擎遵循 BPMN2.0 规范，基本结构与 BPMN 一致，下面简单介绍引擎对应 BPMN 元素的实现。

表格 8BPMN 与 DeepPaaS 流程引擎元素对应

BPMN	自研流程引擎
启动事件	开始环节
结束事件	结束环节
人工活动	人工环节
子流程活动	子流程环节
单一网关	判断环节
并行网关	分支环节
默认顺序流	流转线

条件顺序流	流转线（附加条件信息）
流程对象	流程实例
用户任务	任务

流程图标均参照 BPMN2.0 进行设计

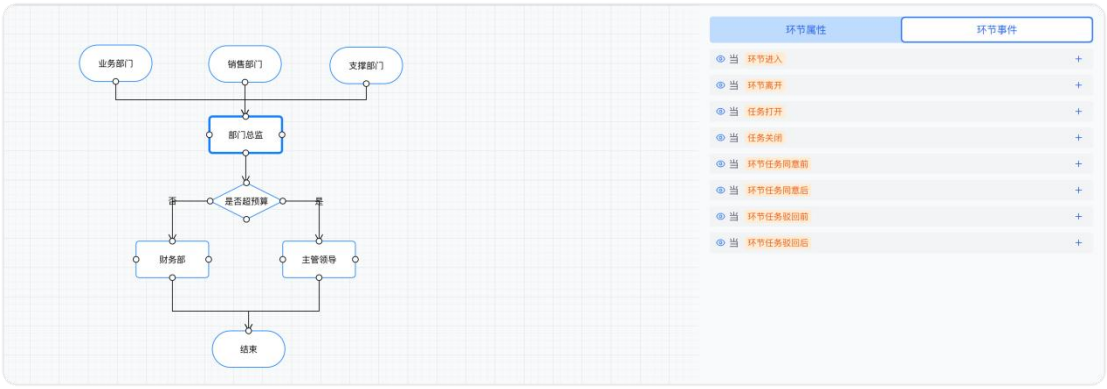


图 32DeepPaas 流程引擎示例

4.2.3.1 流程引擎表定义

以下为流程引擎主要使用表

表格 9 流程引擎实例表

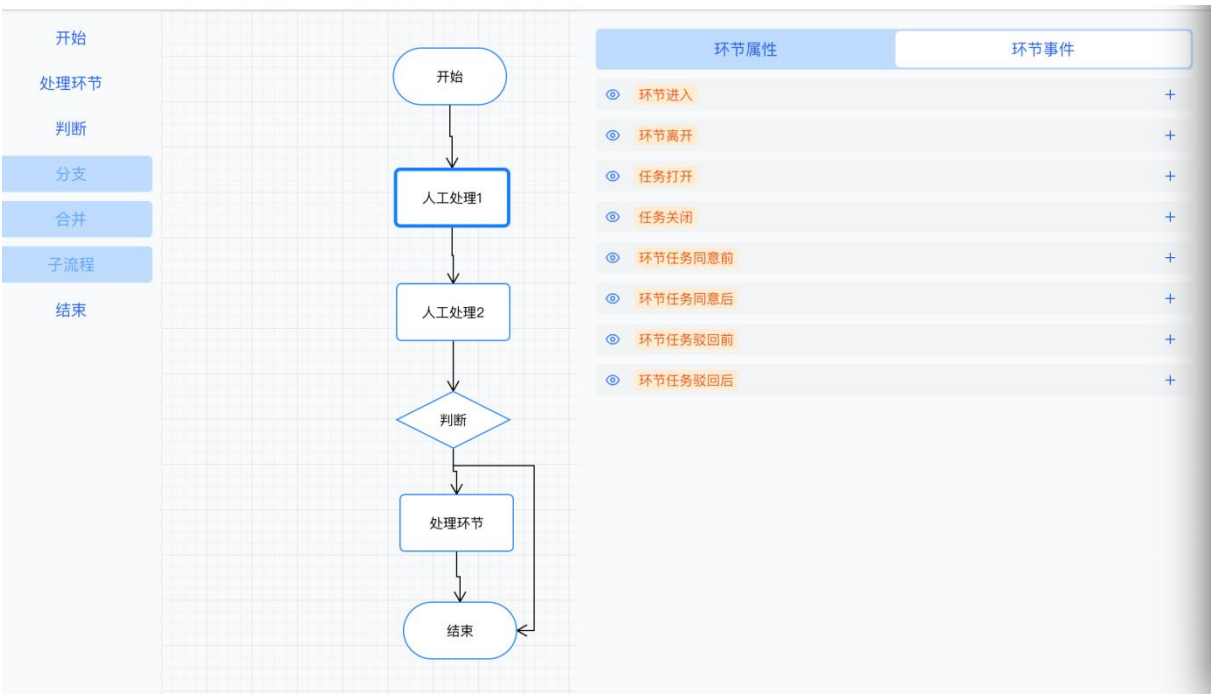
表	备注
flow_define	流程定义表
flow_page	流程定义关联界面表
flow_instance	流程实例表
flow_instance_user	流程实例参与人表
flow_position	流程实例位置表
flow_record	流程实例操作记录表
flow_task	流程任务表
flow_token	流程令牌表
flow_track	流程轨迹表
flow_user_proxy	流程代理人表

4.2.3.2 流程引擎接口

FlowClient：提供流程启动，流程实例控制各种接口。

4.2.3.3 流程引擎集成扩展

流程设置除 BPMN2.0 事件外，软件本身提供如下默认事件：环节进入、环节离开、任务打开、任务关闭、任务流转后、任务驳回前、任务驳回后、流程实例提交前、流程实例提交后、流程实例完成、流程实例撤回前、流程实例撤回后、流程实例中止前、流程实例中止后。以上内置事件均可触发配置的业务逻辑，逻辑由由规则引擎进行处理，如规则无法完全处理，可在规则引擎中扩展规则动作进行逻辑实现。



4.2.3.4 BPMN 实现扩展

本软件未对 BPMN 所有元素进行实现，如需新增 BPMN 实现，如增加活动，则可自定义流程环节扩展，方式为继承并实现一个 FlowNodeAdapter，新的 FlowNodeAdapter 同时需实现新环节的属性结构类以及从 json 构建新环节对应代码。

```
public abstract class FlowNodeAdapter<T extends FlowNode> {
    /**
     * 动作类型，必须唯一
     */
    private final String nodeType;

    public FlowNodeAdapter(String nodeType) {
        this.nodeType = nodeType;
    }

    /**
     * 解析环节配置
     */
    public abstract T buildNode(JsonNode nodeJson);
    /**
     * 进入环节
     */
    public abstract void enter(T flowNode, ExecuteItem executeItem);

    /**
     * 执行，传递令牌
     */
    public abstract void execute(T flowNode, ExecuteItem executeItem);

    /**
     * 向前流转，根据流程线流转
     */
    public abstract void runOutTransition(T flowNode, ExecuteItem executeItem);

    /**
     * 模拟进入
     */
    public abstract List<AssignInfo> mockEnter(T flowNode, MockExecuteItem executeItem);

    /**
     * 模拟执行，传递令牌
     */
    public abstract List<AssignInfo> mockExecute(T flowNode, MockExecuteItem executeItem);

    /**
     * 模拟流转
     */
    public abstract List<AssignInfo> mockRunOutTransition(T flowNode, MockExecuteItem execu

    /**
     * 检查
     * @return
     */
    public abstract List<String> check(T flowNode);

}
```

实现后可在启动配置中实现覆盖实现 AbstractFlowConfiguration 增加新的环节类型适配注册。

```java
@Configuration
public class CustomFlowConfig extends AbstractFlowConfiguration {
    @Override
    protected List<FlowNodeAdapter> userFlowNodeAdapters() {
        List<FlowNodeAdapter> userNodeAdapters = new ArrayList<>();
        userNodeAdapters.add(new EndNodeAdapter("END"));
        return userNodeAdapters;
    }
}
```

4.2.4 规则引擎设计

4.2.4.1 规则引擎表定义

表格 10 规则引擎使用表列表

表	备注
rule	规则定义表
rule_event_rule	事件触发规则记录表
rule_entity	计算模型定义表
rule_entity_property	计算模型属性定义表

4.2.4.2 规则引擎接口

- EventRuleClient：提供事件规则查询接口；
- RuleClient：提供规则查询，调用接口。

4.2.4.3 规则引擎集成扩展

规则引擎主要处理逻辑方式为：表达式、内置函数，在使用时大部分逻辑可通过这两种方式进行实现。同时可对内置函数进行扩展，函数由开发自行实现，实现后加入规则函数即可。

函数添加方法为：新增类实现 RuleFunction 接口，并在自定义的 RuleConfiguration 的 addFunctions 覆盖方法中添加即可。

```java
public interface RuleFunction {
    /**
     * 方法名
     * @return
     */
    String name();

    /**
     * 执行
     * @param args
     * @return
     */
    Object execute(Object... args);
}
```

动作添加方法为：新增类实现 RActionAdapter，新的 RActionAdapter 同时需实现新动作对应属性结构类以及从 json 构建动作对应代码。实现后可在启动配置中实现覆盖实现 RuleConfiguration 的 addActionAdapter 覆盖方法添加即可。

```java
public abstract class RActionAdapter<T extends RAction> {
    /**
     * 动作类型，必须唯一
     */
    private final String actionType;

    public String getActionType() {
        return actionType;
    }

    public RActionAdapter(String actionType) {
        this.actionType = actionType;
    }

    /**
     * 构建动作
     * @param actionNode
     * @param paramDataType
     * @return
     */
    public abstract T buildAction(JsonNode actionNode, SimpleDataType paramDataType);

    /**
     * 执行动作
     * @param action
     * @param ruleContext
     * @return
     */
    public abstract Object execute(T action, RuleContext ruleContext);

}
```

自定义配置

```java
@Configuration
public class CustomRuleConfig extends AbstractRuleConfiguration {

    public CustomRuleConfig(UserClient userClient, DeptClient deptClient,
                            FileClient fileClient, DataStoreDataClient dataStoreDataClient,
                            ExtApiClient extApiClient, MsgClient msgClient) {
        super(userClient, deptClient, fileClient, dataStoreDataClient, extApiClient, msgClient);
    }

    @Override
    protected List<RuleFunction> userFunctions() {
        List<RuleFunction> functions = new ArrayList<>();
        functions.add(new DateAddFunction());
        return functions;
    }

    @Override
    protected List<RActionAdapter> userActionAdapters() {
        List<RActionAdapter> actionAdapters = new ArrayList<>();
        actionAdapters.add(new RVarActionAdapter("ACTION_TYPE_VARIABLE"));
        return actionAdapters;
    }
}
```

4.2.5 租户体系设计

4.2.5.1 租户体系表定义

表格 11 租户体系主要使用表

表	备注
tenant_setting	租户设置表
tenant_app	应用定义表
tenant_app_menu	应用菜单定义表
tenant_app_menu_group	应用菜单组定义表
account_user	用户表
org_dept	部门表
org_dept_manager	部门管理表
org_dept_user	部门用户表
org_post	岗位表
org_post_scope	岗位人员管理范围表
account_permission	权限表
account_permission_group	权限组表
account_role	角色表

account_role_group	角色组表
account_role_permission	角色授权表
account_role_user	角色人员表
account_user_proxy	岗位代理人表
account_attribute_setting	组织结构相关属性定义表
account_attribute_record	组织结构相关属性记录表

4.2.5.2 租户体系接口

- UserClient：提供人员信息相关查询接口；
- RoleClient：提供角色以及角色人员相关查询接口；
- DeptClient：提供部门以及部门人员岗位等相关查询接口；
- PostClient：提供岗位以及对应人员授权等相关接口。

4.2.5.3 租户体系集成扩展

租户设置相关扩展只需在租户设置增加 key 和值即可；

人员、岗位等增加属性只需在属性定义中增加后添加相应记录即可；

如需登录授权额外集成，需开发人员增加 spring 的 HandlerInterceptor 进行自定义实现进行配置即可。

4.2.6 消息系统设计

4.2.6.1 消息系统表定义

表格 12 消息系统使用表

表	备注
deeppaas_user_msg_box	消息盒子定义表
deeppaas_user_msg	消息记录表
deeppaas_user_msg_template	消息模版定义表

4.2.6.2 消息系统接口

- MsgClient：提供消息发送相关查询接口。

4.2.6.3 消息系统集成扩展

消息目前为系统内消息，消息内容可完全自定义，即通过模版配置或在规则引擎中进行文本定义等等。如需扩展其他消息发送，则开发可对 MsgClient 进行自定义实现，已实现更多消息对接。

4.2.7 通用接口设计

4.2.7.1 通用接口表定义

表	备注
ext_api	通用接口
ext_api_param	通用接口参数
ext_api_user	第三方应用
ext_api_user_right	第三方应用授权

4.2.7.2 通用接口接口

- ExtApiClient:提供通用接口调用操作；

4.2.7.3 通用接口集成扩展

软件通用接口部分默认支持为 HTTP 协议调用，可通过配置进行实现即可。如通过规则配置不能满足，可由开发自定义实现 Controller 编写逻辑实现，自定义接口也可配置到通用接口进行发布。如非 HTTP 协议（如 UDP 协议），需开发根据实际场景进行开发编码实现，实现后加入软件即可，加入方式可参考应用部署部分。

4.3 应用部署

根据使用场景需求不同，可使用不同的安装部署模式，以及数据存储方式。程序开发时已在工程结构做拆分，每个大功能模块可以微服务模式提供服务，也可以通过 jar 包方式引入主应用进行部署。

4.3.1 单机模式部署

运维开发人员使用单机部署模式，可使用 deeppaas-starter 工程进行编译打包，该工程以通过 maven 配置构建方式将所有功能即实现包含在同一个应用中。

并且单机模式使用单个数据库进行启动即可。如有开发编码进行扩展情况，则可参考 deeppaas-starter 工程自建一个 springboot 启动工程，将所需功能模块以及二次开发的程序工程使用 maven 配置方式引入即可。完成开发编码后对自建启动工程进行编译打包即可部署。

4.3.2 微服务模式部署

软件默认使用 Spring Cloud Alibaba 体系进行服务实现，由 Nacos 进行服务注册管理，Sentinel、Seata 等组件需根据部署具体环节进行适配调整。如使用其他服务实现，可能需要根据实际情况进行服务接口开发适配。微服务模式下进行扩展可采用类似单机模式在源有服务上进行扩展也可通过提供新服务方式进行。

4.3.3 数据库使用

软件本身不对数据库分库分表以及备份等进行处理，所有数据库分库分表以及备份等交由数据库中间件（如 Cobar、MyCAT 等）进行处理。

1、分表分库中间件

分布式数据库分表分库中间件，负责和上层应用打交道，对应用可表现为一个独立的数据库，而屏蔽底层复杂的系统细节。分布式数据库中间件除了基本的分表分库功能，还可以丰富一下，比如讲读写分离或者水平扩容功能集成在一起，或者比如读写分离本身也可以作为一个独立的中间件。(Cobar, MyCAT, TDDL, DRDS, DDB)

2、数据增量订阅与消费中间件

增量数据订阅和消费，用户对数据库操作，比如 DML, DCL, DDL 等，这些操作会产生增量数据，下层应用可以通过监测这些增量数据进行相应的处理。这个是基于对数据库增量日志解析，提供增量数据订阅和消费；最有名的是阿里的 Canal。根据 MySQL 的 binlog 实现，Canal 通过监听 Mysql 的 binlog 日志来获取数据，binlog 设置为 row 模式，能够获取到每一条新增、删除、修改的日

志，同时还能获取到修改前后的数据。通常我们可以利用这个中间件，实时感知到 Mysql 中的数据变化，将其数据更新到 NoSQL 数据中，比如 MongoDB、ES 等等；通常项目组加入这些非关系数据库，可以减轻数据库查询压力、在分库分表的架构中，还能起到全局查询的作用。也有针对 Oracle(redolog)的增量数据订阅与消费的中间件。（Canal, Erosa）

3、数据库同步中间件

数据库同步中间件涉及数据库之间的同步操作，可以实现跨（同）机房同步以及异地容灾备份、分流等功能。可以涉及多种数据库，处理之后的数据也可以以多种形式存储。（Otter, JingoBus, DRC）

4、跨数据库（数据源）迁移中间件

数据库与数据库之间会有数据迁移（同步）的动作，同款数据同步原理比较简单，比如 MySQL 主备同步，只要在数据库层进行相应的配置既可，但是跨数据库同步就比较复杂了，比如 Oracle->MySQL. 数据迁移一般包括三个步骤：全量复制，将原数据库的数据全量迁移到新数据库，在这迁移的过程中也会有新的数据产生；增量同步，对新产生的数据进行同步，并持续一段时间以保证数据同步；原库停写，切换新库。将“跨数据库”这个含义扩大一下——“跨数据源”，比如 HDFS, HBase, FTP 等都可以相互同步。（yugong, DataX）

5 平台扩展与二次开发

系统具备强大的可扩展性能，具备完善的二次开发机制。

5.1 扩展平台功能

目前平台所有的逻辑架构均自主可控，使用的第三方中间件均开源且商业免费。

扩展平台功能主要包含扩展前端页面组件、扩展前端功能、扩展规则引擎功能、扩展流程引擎功能等。

关于此部分内容的扩展，请参考平台开发部署结构章节，此章节不赘述。

5.2 引用平台 JAR 包方式二次开发

第二种二次开始方式是引用平台 JAR 包的方式进行二次开发。此种模式的二次开发对开发语言和开发模式有一定的限制。

5.3 基于接口的二次开发

基于平台提供的接口管理功能，可以通过调用接口的方式实现二次开发。此种形式的二次开发，只限制接口的格式，不限制具体的开发语言和开发模式，只要接口能互相调通即可。

6 关键技术及解决途径

表格 13 关键技术及解决途径表

序号	关键技术	解决途径
1	界面托拉拽	自研
2	界面动态场景	自研
3	界面规则设置	自研
4	界面组件属性设置	自研
5	前端表达式	扩展 JS 引擎
6	流程引擎	自研，遵守 BPMN2 规范
7	规则引擎	解析自研，绘图工具利用第三方工具
8	接口配置	自研